Université Paris-Sud

Ecole Doctorale Informatique de Paris-Sud
Laboratoire d'Informatique pour la Mécanique
et les Sciences de l'Ingénieur

Discipline : Informatique

Thèse de doctorat

Soutenue le 13 mai 2015 par

Théodore Bluche

Deep Neural Networks for Large Vocabulary Handwritten Text Recognition

Directeur de thèse :	M. Hermann Ney	Professeur (RWTH, Aachen; LIMSI, Paris)
Co-directeur de thèse :	M. Christopher Kermorvant	Docteur (Teklia, Paris; A2iA, Paris)

Composition du jury :

Président du jury :	M. Thierry Paquet	Professeur (LITIS, Rouen)
Rapporteurs :	M. Christophe Garcia	Professeur (INSA Lyon, LIRIS/IMAGINE)
	M. Enrique Vidal	Professeur (UPV, Espagne)
Examinateurs :	M. Balázs Kégl	Docteur (IN2P3, Paris)
	M. Thierry Paquet	Professeur (LITIS, Rouen)

Groupe Traitement
de la Langue Parlée
LIMSI-CNRS
B.P. 133
91403 Orsay Cedex, France

A2iA
39, rue de la Bienfaisance
75008 Paris, France

ED IPS
Université Paris-Sud
UFR Sciences Orsay
Bât. 650
Orsay, France

A Papa
A Papy

Acknowledgements

It is incredible how personal and professional lives are intertwined, and how what happens outside the office impacts what happens inside. So many people helped me go through the last few years in so many ways, directly or indirectly, maybe not even knowing it. To those I have known for ever and those I have met on the way, to those who believed in me and those who trusted me, thank you.

When I finished my Master, I did not plan to go for a PhD. I must first thank Christopher Kermorvant for contacting me, and for changing my mind on the subject. He played an important role in my doing a job that I love, in turning me from engineer into researcher, and in giving me confidence in my work.

I would like to express my deepest gratitude to Prof. Hermann Ney, who accepted to supervise this thesis, for very interesting discussions, useful advice, and for his valuable experience and priceless guidance.

I am so grateful to Enrique Vidal and Christophe Garcia, who kindly accepted to review my thesis, and to read this manuscript at a time when I could not find the courage to read it anymore, and more generally to the whole defense committee, including Thierry Paquet and Balázs Kégl, for doing me the honour of evaluating my work.

Thanks to A2iA to have welcomed me and made this PhD possible, in the best environment. A student could not dream of a better place to complete an industrial PhD, encouraged to publish, with a lot of freedom and autonomy in the work, and to follow exciting research directions.

I would like to thank everyone in the company for their kindness and support, and particularly my current and former colleagues in the research team: Anne-Laure, my coach, Farès, Jérôme, Vu, Bastien, Ronaldo, and Faouzi, for the laughs and stimulating discussions. Working with you guys is a real pleasure. Coffee breaks are crucial, and would not have been the same without the pleasing talks with Thomas.

The people of LIMSI should not be forgotten. They integrated me in the lab, and even though I went there only once a week, they made me feel part of it. A special thanks goes to François Yvon, who welcomed me when I arrived, and was very attentive with me throughout the thesis. Research is way easier with a clear mind, and I should high-five all the football players of the lab, who allowed me to let the steam off through contested games: Thiago, Phuong, Tom, Sylvain, Mathieu, Nico.

Vielen Dank to Michal, Patrick, Mahdi, and all the students of the i6 lab in RWTH who greeted me on my few stays in Aachen, and with whom I shared captivating discussions.

I am pleased to have received the help of many, at A2iA, LIMSI and EDIPS, for effective administrative and technical support, relieving me from much stress and worry, and allowing me to focus on research.

I have wonderful friends, who are there in easy times, and especially present in tough times, who elevate the ups and smooth the downs. To this respect, a huge “thank you” must go to Laura, for the countless hours on the phone, to Pierre, Antoine, Rudy,

Vincent, Alex, Julien, Matthieu, Stéphane, and everyone who was there when I most needed them, even at a time when all they heard from me was about this thesis. Thanks Letí for being in the front line at this particular moment, and for innumerable other reasons.

Last, but not least, and as always, I am indebted to my extraordinary family, for their inconditional love and support. I could not have gone so far without them. Thanks everyone, my brother Barnabé, my sisters Bertille, Pimprenelle, Pétronille, thanks Mom for the comfort, Isabelle, Nicolas for being there. And finally, I dedicate this work to two special persons. To my grandfather, who, from as far as I can remember, awakened my scientific mind in so many ways. To my father, who taught me so many valuable things in life, how to face problems and make choices, for telling me that I should do what I love.

Merci!

Résumé

La transcription automatique du texte dans les documents manuscrits a de nombreuses applications, allant du traitement automatique des documents à leur indexation ou leur compréhension. L'une des approches les plus populaires de nos jours consiste à parcourir l'image d'une ligne de texte avec une fenêtre glissante, de laquelle un certain nombre de caractéristiques sont extraites, et molélisées par des Modèles de Markov Cachés (MMC). Quand ils sont associés à des réseaux de neurones, comme des Perceptrons Multi-Couches (PMC) ou Réseaux de Neurones Récurrents de type Longue Mémoire à Court Terme (RNR-LMCT), et à un modèle de langue, ces modèles produisent de bonnes transcriptions. D'autre part, dans de nombreuses applications d'apprentissage automatique, telles que la reconnaissance de la parole ou d'images, des réseaux de neurones profonds, comportant plusieurs couches cachées, ont récemment permis une réduction significative des taux d'erreur.

Dans cette thèse, nous menons une étude poussée de différents aspects de modèles optiques basés sur des réseaux de neurones profonds dans le cadre de systèmes hybrides réseaux de neurones / MMC, dans le but de mieux comprendre et évaluer leur importance relative. Dans un premier temps, nous montrons que des réseaux de neurones profonds apportent des améliorations cohérentes et significatives par rapport à des réseaux ne comportant qu'une ou deux couches cachées, et ce quel que soit le type de réseau étudié, PMC ou RNR, et d'entrée du réseau, caractéristiques ou pixels. Nous montrons également que les réseaux de neurones utilisant les pixels directement ont des performances comparables à ceux utilisant des caractéristiques de plus haut niveau, et que la profondeur des réseaux est un élément important de la réduction de l'écart de performance entre ces deux types d'entrées, confirmant la théorie selon laquelle les réseaux profonds calculent des représentations pertinantes, de complexités croissantes, de leurs entrées, en apprenant les caractéristiques de façon automatique. Malgré la domination flagrante des RNR-LMCT dans les publications récentes en reconnaissance d'écriture manuscrite, nous montrons que des PMCs profonds atteignent des performances comparables. De plus, nous avons évalué plusieurs critères d'entrainement des réseaux. Avec un entrainement discriminant de séquences, nous reportons, pour des systèmes PMC/MMC, des améliorations comparables à celles observées en reconnaissance de la parole. Nous montrons également que la méthode de Classification Temporelle Connexionniste est particulièrement adaptée aux RNRs. Enfin, la technique du *dropout* a récemment été appliquée aux RNR. Nous avons testé son effet à différentes positions relatives aux connexions récurrentes des RNRs, et nous montrons l'importance du choix de ces positions.

Nous avons mené nos expériences sur trois bases de données publiques, qui représentent deux langues (l'anglais et le français), et deux époques, en utilisant plusieurs types d'entrées pour les réseaux de neurones : des caractéristiques prédéfinies, et les simples valeurs de pixels. Nous avons validé notre approche en participant à la compétition HTRtS en 2014, où nous avons obtenu la deuxième place. Les résultats des systèmes présentés dans cette thèse, avec les deux types de réseaux de neurones et d'entrées, sont comparables à l'état de l'art sur les bases Rimes et IAM, et leur combinaison dépasse les meilleurs résultats publiés sur les trois bases considérées.

Mots-clés *– Reconnaissance de formes • Modèles de Markov Cachés • Réseaux de Neurones • Reconnaissance de l'Ecriture Manuscrite*

Abstract

The automatic transcription of text in handwritten documents has many applications, from automatic document processing, to indexing and document understanding. One of the most popular approaches nowadays consists in scanning the text line image with a sliding window, from which features are extracted, and modeled by Hidden Markov Models (HMMs). Associated with neural networks, such as Multi-Layer Perceptrons (MLPs) or Long Short-Term Memory Recurrent Neural Networks (LSTM-RNNs), and with a language model, these models yield good transcriptions. On the other hand, in many machine learning applications, including speech recognition and computer vision, deep neural networks consisting of several hidden layers recently produced a significant reduction of error rates.

In this thesis, we have conducted a thorough study of different aspects of optical models based on deep neural networks in the hybrid neural network / HMM scheme, in order to better understand and evaluate their relative importance. First, we show that deep neural networks produce consistent and significant improvements over networks with one or two hidden layers, independently of the kind of neural network, MLP or RNN, and of input, handcrafted features or pixels. Then, we show that deep neural networks with pixel inputs compete with those using handcrafted features, and that depth plays an important role in the reduction of the performance gap between the two kinds of inputs, supporting the idea that deep neural networks effectively build hierarchical and relevant representations of their inputs, and that features are automatically learnt on the way. Despite the dominance of LSTM-RNNs in the recent literature of handwriting recognition, we show that deep MLPs achieve comparable results. Moreover, we evaluated different training criteria. With sequence-discriminative training, we report similar improvements for MLP/HMMs as those observed in speech recognition. We also show how the Connectionist Temporal Classification framework is especially suited to RNNs. Finally, the novel dropout technique to regularize neural networks was recently applied to LSTM-RNNs. We tested its effect at different positions in LSTM-RNNs, thus extending previous works, and we show that its relative position to the recurrent connections is important.

We conducted the experiments on three public databases, representing two languages (English and French) and two epochs, using different kinds of neural network inputs: handcrafted features and pixels. We validated our approach by taking part to the HTRtS contest in 2014. The results of the final systems presented in this thesis, namely MLPs and RNNs, with handcrafted feature or pixel inputs, are comparable to the state-of-the-art on Rimes and IAM. Moreover, the combination of these systems outperformed all published results on the considered databases.

Keywords *– Pattern Recognition • Hidden Markov Models • Neural Networks • Handwriting Recognition*

Contents

List of Tables 17

List of Figures 21

Introduction 25

I HANDWRITING RECOGNITION — OVERVIEW 31

1 Offline Handwriting Recognition – Overview of the Problem 33
1.1 Introduction . . . 35
1.2 Preliminary Steps to Offline Handwriting Recognition . . . 37
1.3 Reducing Handwriting Variability with Image Processing Techniques . 39
1.3.1 Normalizing Contrast . . . 39
1.3.2 Normalizing Skew . . . 40
1.3.3 Normalizing Slant . . . 41
1.3.4 Normalizing Size . . . 41
1.4 Extraction of Relevant Features for Handwriting Recognition . . . 42
1.4.1 Text Segmentation for Feature Extraction . . . 42
1.4.2 Features for Handwriting Representation . . . 43
1.5 Modeling Handwriting . . . 46
1.5.1 Whole-Word Models . . . 47
1.5.2 Part-Based Methods . . . 47
1.5.3 Segmentation-Free Approach . . . 48
1.6 Modeling the Language to Constrain and Improve the Recognition . . . 49
1.6.1 Vocabulary . . . 50
1.6.2 Language Modeling . . . 51
1.6.3 Open-Vocabulary Approaches . . . 53
1.7 Measuring the Quality of the Recognition . . . 53
1.8 Conclusion . . . 54

2 Handwriting Recognition with Hidden Markov Models and Neural Networks 55
2.1 Introduction . . . 57
2.2 Hidden Markov Models for Handwriting Recognition . . . 58
2.2.1 Definition . . . 59
2.2.2 Choice of Topology . . . 61
2.2.3 Choice of Emission Distribution . . . 62
2.2.4 Model Refinements . . . 62
2.2.5 Decoding . . . 63

2.3 Neural Networks for Handwriting Recognition . . . 63
2.3.1 The Multi-Layer Perceptron . . . 63
2.3.2 Recurrent Neural Networks . . . 66
2.3.3 Long Short-Term Memory Units . . . 67
2.3.4 Convolutional Neural Networks . . . 68
2.4 Handwriting Recognition Systems with Neural Networks . . . 69
2.4.1 The Hybrid NN/HMM scheme . . . 70
2.4.2 Predicting Characters . . . 71
2.4.3 NN Feature Extractors . . . 71
2.5 Training Models . . . 72
2.5.1 Training Hidden Markov Models with Generative Emission Models 72
2.5.2 Training Neural Networks . . . 73
2.5.3 Training Deep Neural Networks . . . 77
2.5.4 Training Complete Handwriting Recognition Systems . . . 81
2.6 Conclusion . . . 86

II EXPERIMENTAL SETUP 89

3 Databases and Software 91
3.1 Introduction . . . 93
3.2 Databases of Handwritten Text . . . 93
3.2.1 Rimes . . . 93
3.2.2 IAM . . . 94
3.2.3 Bentham . . . 95
3.3 Software . . . 96
3.4 A Note about the Experimental Setup in the Next Chapters . . . 97

4 Baseline System 99
4.1 Introduction . . . 101
4.2 Preprocessing and Feature Extraction . . . 102
4.2.1 Image Preprocessing . . . 102
4.2.2 Feature Extraction with Sliding Windows . . . 103
4.3 Language Models . . . 105
4.3.1 Corpus Preparation and Vocabulary Selection . . . 105
4.3.2 Language Models Estimation . . . 106
4.3.3 Recognition Output Normalization . . . 107
4.4 Decoding Method . . . 108
4.5 A GMM/HMM baseline system . . . 109
4.5.1 HMM topology selection . . . 109
4.5.2 GMM/HMM training . . . 109
4.5.3 Results . . . 110
4.6 Conclusion . . . 112

III DEEP NEURAL NETWORKS IN HIDDEN MARKOV MODEL SYSYEMS 113

5 Hybrid Deep Multi-Layer Perceptrons / HMM for Handwriting Recognition 115
5.1 Introduction . . . 117
5.2 Experimental Setup . . . 118
5.3 Study of the Influence of Input Context . . . 119
5.3.1 Alignments from GMM/HMM Systems . . . 119
5.3.2 Handcrafted Features . . . 121
5.3.3 Pixel Intensities . . . 123
5.4 Study of the Impact of Depth in MLPs . . . 124
5.4.1 Deep MLPs . . . 124
5.4.2 Deep vs Wide MLPs . . . 126
5.5 Study of the Benefits of Sequence-Discriminative Training . . . 128
5.6 Study of the Choice of Inputs . . . 130
5.7 Conclusion . . . 131

6 Hybrid Deep Recurrent Neural Networks / HMM for Handwriting Recognition 133
6.1 Introduction . . . 135
6.2 Experimental Setup . . . 136
6.2.1 RNN Architecture Overview . . . 136
6.2.2 Decoding in the Hybrid NN/HMM Framework . . . 138
6.3 Study of the Influence of Input Context . . . 139
6.3.1 Including Context with Frame Concatenation . . . 140
6.3.2 Context through the Recurrent Connections . . . 140
6.4 Study of the Influence of Recurrence . . . 142
6.5 Study of the Impact of Depth in BLSTM-RNNs . . . 143
6.5.1 Deep BLSTM-RNNs . . . 144
6.5.2 Deep vs Wide BLSTM-RNNs . . . 146
6.5.3 Analysis . . . 148
6.6 Study of the Impact of Dropout . . . 149
6.6.1 Dropout after the Recurrent Layers . . . 151
6.6.2 Dropout at Different Positions . . . 151
6.6.3 Study of the Effect of Dropout in Complete Systems (with LM) 155
6.7 Study of the Choice of Inputs . . . 156
6.8 Conclusion . . . 158

IV COMPARISON AND COMBINATION OF DEEP MLPs AND RNNs 161

7 Experimental Comparison of Framewise and CTC Training 163
7.1 Introduction . . . 165
7.2 Experimental Setup . . . 167

7.3 Relation between CTC and Forward-Backward Training of Hybrid NN/HMMs . . . 167
7.3.1 Notations . . . 167
7.3.2 The Equations of Forward-Backward Training of Hybrid NN/HMMs . . . 168
7.3.3 The Equations of CTC Training of RNNs . . . 169
7.3.4 Similarities between CTC and hybrid NN/HMM Training . . . 171
7.4 Topology and Blank . . . 172
7.5 CTC Training of MLPs . . . 173
7.6 Framewise vs CTC Training . . . 174
7.7 Interaction between CTC Training and the Blank Symbol . . . 176
7.7.1 Peaks . . . 176
7.7.2 Trying to avoid the Peaks of Predictions . . . 179
7.7.3 The advantages of prediction peaks . . . 181
7.8 CTC Training without Blanks . . . 182
7.9 The Role of the Blank Symbol . . . 183
7.10 Conclusion . . . 183

8 Experimental Results, Combinations and Discussion 185
8.1 Introduction . . . 187
8.2 Summary of Results on Rimes and IAM Databases . . . 188
8.2.1 MLP/HMM Results . . . 188
8.2.2 RNN/HMM Results . . . 190
8.2.3 Comparison of MLP/HMM and RNN/HMM Results . . . 191
8.2.4 Combination of the Proposed Systems . . . 193
8.3 The Handwritten Text Recognition tranScriptorium (HTRtS) Challenge 196
8.3.1 Presentation of the HTRtS Evaluation and of the Experimental Setup . . . 196
8.3.2 Systems Submitted to the Restricted Track . . . 197
8.3.3 Systems Submitted to the Unrestricted Track . . . 200
8.3.4 Post-Evaluation Improvements . . . 203
8.4 Conclusion . . . 205

Conclusions and Perspectives 207

List of Publications 213

Appendices 216

A Databases 217
A.1 IAM . . . 217
A.2 Rimes (ICDAR 2011 setup) . . . 221
A.3 Bentham (HTRtS 2014 setup) . . . 225

B Résumé Long **229**
B.1 Système de Base 231
B.2 Systèmes Hybrides Perceptrons Multi-Couches Profonds / MMC 232
B.3 Systèmes Hybrides Réseaux de Neurones Récurrents Profonds / MMC . 234
B.4 Une Comparaison Expérimentale de l'Entrai-nement CTC et au Niveau Trame 236
B.5 Combinaisons et Résultats Finaux 238
B.6 Conclusions et Perspectives 241

Bibliography **243**

List of Tables

3.1 Number of pages, lines, words and characters in each dataset 94
3.2 Software used in this thesis. 96

4.1 Perplexities of Rimes LM with different discounting methods and $n$gram orders, on the validation set. 107
4.2 Perplexities of Bentham LMs with different $n$gram orders and hyphenation, on the validation set. 107
4.3 Context dependency in GMM/HMM (results on lines of IAM validation set) 110
4.4 Applying LMs to text lines and paragraphs (results on IAM validation set) 110
4.5 Results on Rimes database 111
4.6 Results on IAM database 111
4.7 Results on Bentham database 112

5.1 Character widths, estimated a priori from the images and annotations, and from the alignments with the GMM/HMM system. 121
5.2 Effect of concatenating several consecutive frames on deep MLP results (Word Error Rates; **Feat.**/**Deep**/**Xent**). 121
5.3 Influence of the size of the sliding window for deep MLPs with pixel values inputs (Rimes; **Pix.**/**Deep**/**Xent**). 123
5.4 MLPs on handcrafted and pixel features. The Frame Error Rates (FERs) measure the classification error of the MLP alone, while the Word and Characer Error Rates (WER, CER) are obtained with lexicon and language model. (**Xent**) 125
5.5 Selected MLP architectures 128
5.6 Effect of sMBR training. The *cross-entropy* corresponds to framewise training, as oposed to sMBR, which is a sequence-discriminative criterion (**Deep**/**Seq.**). 130
5.7 Comparison of WERs (%) obtained with handcrafted features and pixel values. 130

6.1 Effect of context on BLSTM-RNN performance (**Feat.**/**Deep**/**CTC**). . 140
6.2 Effect of recurrence on the character error rate of the RNN alone, without lexicon and language model (RNN-CER%; **Deep**/**CTC**). 143
6.3 Effect of depth on the performance of RNNs (**CTC**). 146
6.4 Comparison of number of parameters for different architectures. 147
6.5 Effect of dropout after the top N LSTM layers (RNN-CER%; **Deep**/**CTC**). 151
6.6 Effect of dropout in complete pipelines (**Deep**/**CTC**). 151

6.7 Effect of dropout at different positions on Rimes database (RNN-CER%; **Deep**/**CTC**). . . . 152
6.8 Effect of dropout at different positions on IAM database (RNN-CER%; **Deep**/**CTC**). . . . 153
6.9 Effect of dropout at different positions on Bentham database (RNN-CER%; **Deep**/ **CTC**). . . . 153
6.10 Effect of dropout at different positions in complete pipelines (**Deep**/**CTC**). 155
6.11 Effect of dropout at different combinations of positions in complete pipelines (**Deep**/**CTC**). . . . 157
6.12 Comparison of WERs (%) obtained with handcrafted features and pixel values. . . . 158

7.1 WER improvements with forward-backward training of neural networks 166
7.2 WER% (CER%) of different standard systems with different topologies. RNNs are trained with CTC, MLPs with framewise training, and GMM/HMMs with Viterbi training (IAM Database, **Shallow**/**Feat.**). . 173
7.3 CTC training of MLPs. The *"HMM"* topology has six states per character while the *"CTC"* topology is standard one for CTC, with one output per character and a blank symbol (IAM Database, **Shallow**/**Feat.**). . . 174
7.4 Comparison of WER%/CER% with framewise and CTC training of neural networks with different output topologies (IAM Database, **Shallow**/**Feat.**) . . . 175
7.5 Error rates of neural networks alone without lexicon and language model with different topologies (*labels* are HMM states; **Shallow**/**Feat.**). . . . 177

8.1 Effect of adding linguistic knowledge in MLP/HMM systems (**Deep**/**Seq.**). . . . 189
8.2 Effect of adding linguistic knowledge in RNN/HMM systems (**Deep**/ **Drop.**/ **CTC**). . . . 191
8.3 Comparison of MLP and RNN optical models with different types of inputs (**Deep**). . . . 192
8.4 ROVER and Lattice combination of MLPs and RNNs with features and pixel inputs on Rimes and IAM (**Deep**). . . . 194
8.5 Final results on Rimes database . . . 195
8.6 Final results on IAM database . . . 195
8.7 MLP results (cross-entropy framewise training). The figures are WERs (%) (**Xent**). . . . 198
8.8 Improvement brought by sMBR sequence training, as oposed to the cross-entropy framewise training (**Deep**/**Seq.**). . . . 198
8.9 RNNs on handcrafted and pixel features (**Deep**/**CTC**). . . . 198
8.10 Summary of results of restricted systems. . . . 199
8.11 Comparison of combination techniques for the four restricted track systems. . . . 199
8.12 Competition Results for the Restricted Track. . . . 200
8.13 Data used for optical model training. . . . 201

8.14 BLSTM-RNN unrestricted results (RNN-CER% is the Character Error Rate with RNN alone, while the WER% is after adding the lexicon and LM ; GW and AL stand for G. Washington and A. Lincoln. *"s-"* indicates that only a subset was used) (**Deep**/**Drop.**). 202
8.15 Improvements brought by adding more LM data (WER% / CER%; **Deep**). 202
8.16 Competition Results for the Unrestricted Track. 203
8.17 WER% improvements brought by adding even more LM data (**Deep**). 204
8.18 Results on the evaluation set. 204
8.19 WER% of the refined models (**Deep**). 205
8.20 Refined results on the evaluation set (**Deep**) 205

A.1 Documents of the IAM database, in each dataset. The *Total* column are accumulated numbers, while the average, min/max and quartiles are computed per line. The width/char measure is obtained by dividing, for each line, the width by the number of characters in the annotation, giving only a rough estimation. 219
A.2 Text of the IAM database. The averages, quartiles, and min/max values are calculated from occurences of each token, *e.g.* 75% of the characters appear at least 3,511 times in the training set. 219
A.3 Selection of published results on IAM database 220
A.4 Documents of the Rimes database, in each dataset. The *Total* column are accumulated numbers, while the average, min/max and quartiles are computed per line. The width/char measure is obtained by dividing, for each line, the width by the number of characters in the annotation, giving only a rough estimation. 223
A.5 Text of the Rimes database. The averages, quartiles, and min/max values are calculated from occurences of each token, *e.g.* 75% of the characters appear at least 3,511 times in the training set. 223
A.6 Published results on Rimes database 224
A.7 Documents of the Bentham database, in each dataset. 226
A.8 Text of the Bentham database. 227
A.9 Published results on Bentham database 227

B.1 Comparaison du taux d'erreurs mots (%) obtenus avec des PMCs utilisant des caractéristiques pré-définies et des intensités de pixels 234
B.2 Comparaison du taux d'erreurs mots (%) obtenus avec des RNRs utilisant des caractéristiques pré-définies et des intensités de pixels 236
B.3 Résultats sur la base Rimes (taux d'erreur mots - WER% - et caractères - CER%). 240
B.4 Résultats sur la base IAM (taux d'erreur mots - WER% - et caractères - CER%). 240
B.5 Résultats sur la base Bentham (taux d'erreur mots - WER% - et caractères - CER%). 241

List of Figures

1 Overview of this thesis in the scope of a handwriting recognition system. Red boxes are the focus on this work: neural network optical models. . 28

1.1 A digital image of a word at different scales. 35
1.2 The French word *"vous"* written by different persons, from the Rimes database. 36
1.3 A similar shape can represent several handwritten characters. 36
1.4 Various images from the Maurdor campaign (Brunessaux et al., 2014). . 38
1.5 Example of skewed text line (Rimes database – left) and slanted handwriting (IAM database – right) 40

2.1 A Hidden Markov Model. 59
2.2 Hidden Markov Model topology for Handwriting Recognition. 61
2.3 Multi-Layer Perceptron. $x_1, \ldots, x_N$ are the inputs, $\mathbf{W}^{(i)}, \mathbf{b}^{(i)}$ are the weight matrice and bias vector of layer i. 65
2.4 Recurrent Neural Networks, simple form 66
2.5 Neurons for RNNs: (a) Simple Neuron (b) LSTM unit 67
2.6 Example of Convolutional Neural Network. 69
2.7 Multi-Layer Perceptron training by backpropagation of the error. . . . 75
2.8 Backpropagation Through Time. 75
2.9 The Dropout technique. 77
2.10 Unsupervised layer-wise RBM pretraining. 78
2.11 Supervised layer-wise MLP pretraining. 80
2.12 CTC graph. 83

3.1 Examples from Rimes Database. 94
3.2 Examples from IAM Database (the text to copy corresponds to the typed header paragraph). 95
3.3 Examples from Bentham Database. 95

4.1 Overview of the recognition system. 101
4.2 Optimization of the sliding window parameters (IAM) 104
4.3 Extraction of corresponding handcrafted and pixel features with two different sliding windows. 105

5.1 Including more context in MLPs. Left: a wider sliding window. Right: concatenation of successive sliding windows. Blue circles correspond to the NN inputs. 119
5.2 Statistics of the forced alignments with the GMM/HMM systems. From top to bottom: boxplot of the character length, character average length with average duration of each state, histograms of character lengths and state durations. (**Feat.**). 120
5.3 Effect of context and depth on MLP performance (**Feat.**/**Deep**/**Xent**). 122

5.4 MLP input filters (representing the weights between the input and first layer) with pixel inputs on three databases, after pretraining and after fine-tuning. (**Pix.**/**Deep**/**Xent**) . . . 126
5.5 Comparison of the effect of width and depth on MLP performance. The line styles represent the number of hidden units per layer (dots: 256, dashes: 512, solid: 1,024). The line colors represent different inputs (black: pixels, colors: several concatenation of features) (**Deep**/**Xent**). 127
5.6 WER and CER evolution during sequence-discriminative training (**Deep**/**Seq.**). . . . 129

6.1 Architecture of the BLSTM-RNNs used in this thesis. . . . 137
6.2 HMM and Lexicon for hybrid NN/HMM decoding with CTC-trained networks. . . . 139
6.3 Context used trough recurrent connections by LSTM-RNNs to predict *"a"* in Rimes, *"a"* in IAM, *"v"* in Bentham (sensitivity heatmaps, top: features, bottom: pixels ; **Deep**/**Drop.**/**CTC**). . . . 141
6.4 LSTM (**L**) and feed-forward (F) blocks. . . . 142
6.5 Input sensitivity and character predictions of RNNs with different architectures on Rimes database (**Pix.**/**Deep**/**CTC**). . . . 144
6.6 Effect of depth on RNN performance (alone and in the complete pipeline; **CTC**). . . . 145
6.7 Effect of depth vs width on RNN performance (RNN-CER, **CTC**). . . . 147
6.8 Weights of the first layer of pixel LSTM-RNNs (**Pix.**/**Deep**/**CTC**). . . . 149
6.9 Weights of the cell input of the first layer of pixel LSTM-RNNs with different depths. (**Pix.**/**Deep**/**CTC**). . . . 150
6.10 Sensitivity maps of LSTM-RNNs of increasing depths (1 to 9; **Pix.**/**Deep**/ **CTC**). . . . 150
6.11 Dropout position in LSTM-RNNs. . . . 152
6.12 Weights of the first layer of pixel LSTM-RNNs with different dropout strategies on Rimes database (**Pix.**/**Deep**/**CTC**). . . . 154

7.1 Comparison of CER% with CTC and framewise training, with and without blank (left: MLP; right: RNN; **Shallow**/**Feat.**). . . . 176
7.2 Outputs of different neural networks with different topologies and training methods. Each plot represents the NN posteriors at different timesteps. The blank output is represented in gray, and other outputs corresponding to HMM states, in color. . . . 177
7.3 Evolutions of the outputs of an RNN for a given text line during CTC training. As in Figure 7.2, gray corresponds to the blank output, and colors to other outputs. . . . 178
7.4 Visualisation of the state posteriors computed with the forward-backward algorithm during CTC training. . . . 179
7.5 Character durations in decoding when each label is repeated n times between each blank during training. . . . 180
7.6 Effect of different learning rates on the output of networks trained with CTC and the blank symbol. . . . 181

7.7 Outputs of a NN trained with CTC training without blank (**CTC**). . . 182

8.1 Influence of the decoding parameters (optical scale, word insertion penalty, prior scale) on the WER% of hybrid MLP/HMM systems (**Deep**/**Seq.**). 189
8.2 Influence of the decoding parameters (optical scale, word insertion penalty, prior scale) on the WER% of hybrid RNN/HMM systems (**Deep**/ **Drop.**/ **CTC**). 191
8.3 Influence of the number of free parameters in MLPs and RNNs. Circles correspond to handcrafted features, and + to pixels. RNNs are shown in blue and MLPs in green, each point is one network (**Deep**). 192

A.1 Examples from IAM Database (the text to copy corresponds to the typed header paragraph). 218
A.2 Examples from Rimes Database. 222
A.3 Examples from Bentham Database. 226

B.1 Comparaison des taux d'erreur caractères (CER%) avec l'entrainement CTC et au niveau trame, avec et sans le symbole spécial *blank*. (à gauche: PMC; à droite : RNR). 238

Introduction

We live in a digital world, where information is stored, processed, indexed and searched by computer systems, making its retrieval a cheap and quick task. Handwritten documents are no exception to the rule. The stakes of recognizing handwritten documents, and in particular handwritten texts, are manifold, ranging from automatic cheque or mail processing to archive digitalization and document understanding.

The treatment of handwritten documents raises some issues. The text they contain is not readily available in a form that could be easily handled by computed. Instead, it should be extracted from the image resulting from the digitalization of the document. Thus, the regions of the image containing handwritten text must be found, and converted into ASCII text. The latter process is known as offline handwriting recognition.

This field has enjoyed over sixty years of research. Starting with isolated characters and digits, the focus shifted to the recognition of words. Recognizing cursive words is significantly more difficult than characters, mainly for two reasons. First, there are many more different words in a language than there are characters. Moreover, the segmentation of a handwritten word image into characters is difficult because of the cursive nature of text, which introduces ambiguities. To a lesser extent, the segmentation of a line of text into words is also ambiguous, and the current strategy is to recognize lines of text directly, and use a language model to constrain the transcription, and help retrieve the correct sequence of words. Progressively, recognition systems evolve to end-to-end recognizers, that process the whole documents directly, without assuming that a segmentation into text lines is available.

Far from being solved, this problem still interests researchers. Projects are funded to improve the performance of recognition systems. Examples include the PACTE project, which aims at improving the techniques for the recognition and understanding of documents. One of its goal is to include linguistic knowledge and natural language processing methods in order to enhance the quality of automatic transcription systems. The MAURDOR project (Brunessaux et al., 2014) focuses on the processing of heterogeneous and complex documents, integrating the localization and logical ordering of text zones, and the recognition of handwitten and typed texts in three languages: English, French, and Arabic. Moreover, the state-of-the-art in handwriting recognition is regularly evaluated by international competitions, such as OpenHaRT (Arabic texts, Tong et al. (2014)), HTRtS (ancient manuscripts, Sánchez et al. (2014)), or the evaluations conducted during the MAURDOR project.

One of the most popular approaches nowadays consists in scanning the image with a sliding window, from which features are extracted. The sequences of such observations are modeled with character Hidden Markov Models (HMMs). Word models are obtained by concatenation of character HMMs. The standard model of observations in HMMs is Gaussian Mixture Models (GMMs). In the nineties, the theory to re-

place Gaussian mixtures and other generative models by discriminative models, such as Neural Networks (NNs), was developped (Bourlard & Morgan, 1994). Discriminative models are interesting because of their ability to separate different HMM states, which improves the capacity of HMMs to differentiate the correct sequence of characters.

A drawback of HMMs is the local modeling, which fails to capture long term dependencies in the input sequence, that are inherent to the considered signal. Recent improvements in Recurrent Neural Networks (RNNs), a kind of NN suited to sequence processing, significantly reduced the error rates. The Long Short-Term Memory units (LSTM, Gers (2001)), in particular, enable RNNs to learn arbitrarily long dependencies from the input sequence, by controling the flow of information through the network. Multi-Dimensional LSTM-RNNs (Graves & Schmidhuber, 2008) can process the whole text line image directly, scanning it along multiple directions, and extracting features learnt from the image automatically.

The current trend in handwriting recognition is to associate neural networks, especially LSTM-RNNs, with HMMs to transcribe text lines. NNs are used either to extract features for Gaussian mixture modeling (Kozielski et al., 2013a), or to predict HMM states and replace GMM optical models (Doetsch et al., 2014; Pham et al., 2014). On the other hand, in many machine learning applications, including speech recognition and computer vision, deep neural networks, consisting of several hidden layers, produced a significant reduction of error rates.

They were left aside in the past, due to the lack of resources and efficient training methods. Until 2006, almost only Convolutional Neural Networks had more than one or two hidden layers, and were successfully applied to computer vision problems, thanks to their limited number of free parameters. Deep neural networks have now regained considerable interest in the machine learning community, and present many interesting aspects, *e.g.* their ability to learn internal representations of increasing complexity of their inputs, reducing the need of extracting relevant features from the image before the recognition. In the last few years, they have become a standard component of speech recognition models, which are close to those applied to handwriting recognition.

In handwriting recognition, however, deep neural networks are limited to convolutional architectures, such as convolutional neural networks or MDLSTM-RNNs, with only a few weights and extracted features in bottom layers. Densely connected neural networks with more than one or two hidden layers are only found in smaller applications, such as the recognition of isolated characters (Ciresan et al., 2010; Cireşan et al., 2012) or keyword spotting (Thomas et al., 2013).

Recurrent neural networks, in particular those with Long Short-Term Memory units, were applied to handwriting recognition in 2007, won international evaluations, and have become a standard component of every state-of-the-art recognition system. Multi-Layer Perceptrons, one the other hand, tend to be neglected by the community in the last few years.

In this thesis, we focus on the hybrid NN/HMM framework, with optical models based on deep neural networks, for large vocabulary handwritten text line recognition. This is depicted on Figure 1, where we display a standard recognition system, and highlight in red boxes the components studied in this thesis. We concentrated on

neural network optical models, and propose a thorough study of their architecture and training procedure, but we also varied their inputs and outputs (in light red on Figure 1). We are interested in answering the following questions:

- *Is it still important to design handcrafted features when using deep neural networks, or are pixel values sufficient?*
- *Can deep neural networks give rise to big improvements over neural networks with one hidden layer for handwriting recognition?*
- *How (deep) Multi-Layer Perceptrons compare to the very popular Recurrent Neural Networks, which are now widespread in handwriting recognition and achieve state-of-the-art performance?*
- *What are the important characteristics of Recurrent Neural Networks, which make them so appropriate for handwriting recognition?*
- *What are the good training strategies for neural networks for handwriting recognition? Can the Connectionist Temporal Classification (CTC, Graves et al. (2006)) paradigm be applied to other neural networks? What improvements can be observed with a discriminative criterion at the sequence level?*

The experiments presented in this thesis were conducted on three public databases, Rimes, IAM and Bentham, representing two languages and epochs. We considered two input types: a set of handcrafted features, which have proven good for handwriting recognition, and the simpler raw pixel intensities.

Our first contribution lies in building deep neural networks for handwritten text recognition, and showing that they achieve consistent and significant improvements over shallow networks consisting of one or two hidden layers.

The second contribution of this thesis is the comparison of handcrafted features and pixel intensities as model inputs in the context of deep neural network optical models, and the observation that both types of inputs yield similar performance of the complete systems.

The third contribution is a direct comparison of LSTM-RNNs, which hold the current state-of-the-art and are omnipresent in handwriting recognition, with MLPs. We report comparable results with both types of neural networks, showing that RNNs are not the only model able to achieve good recognition results, in spite of their wide adoption by the community.

The fourth contribution is a study of the dropout technique applied in Recurrent Neural Networks. Extending previous works on the subject, we show that the position at which dropout is applied, relative to the recurrent connections, is important. By thorough experiments, we found a better application of dropout in BLSTM-RNNs than those proposed in the literature.

The fifth contribution is a comparison of different training methods for neural networks, and in particular the study of the CTC framework for MLPs and RNNs, which showed that it was particularly suited to RNNs, and the confirmation of the significant

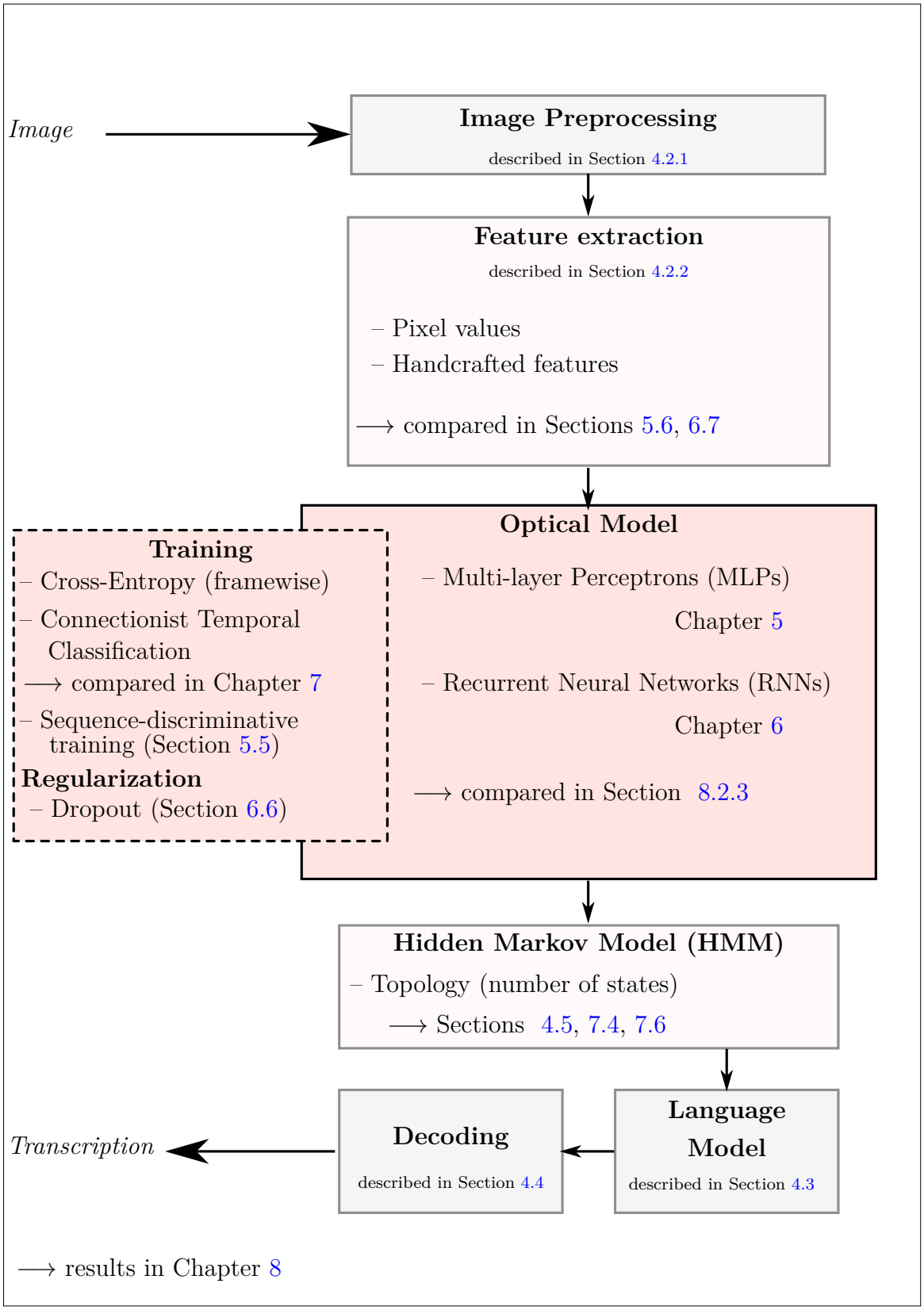


Figure 1: Overview of this thesis in the scope of a handwriting recognition system. Red boxes are the focus on this work: neural network optical models.

improvements brought by sequence-discriminative training, already observed in speech recognition.

Finally, we report state-of-the-art results with MLPs, RNNs, and both kinds of inputs, and the combination of our models outperformed all published results on the three databases.

This thesis is divided into four parts. We begin in Part I with an overview of the problem of handwriting recognition, and a literature review of different approaches and techniques proposed in the last decades. First, in Chapter 1, we present the difficulties of handwriting recognition, and the usual steps and components of recognition systems, namely the extraction of the text lines, the image processing to reduce the variability, the segmentation of text lines into observations, the extraction of relevant features, the modeling of the observations and of the language. In Chapter 2, we focus on the components studied in this thesis: hidden Markov models and neural networks. We present their application to handwriting recognition, in isolation or in association, along with their training procedures.

In Part II, we present the experimental setup chosen for this thesis. We conducted our experiments on three public databases, Rimes, IAM, and Bentham. The data, and the software used are introduced in Chapter 3. More details on the databases and on published results can be found in Annex A. We explore deep neural networks for optical modeling in handwriting recognition. The optical model is only a component of a complete recognition system, which typically also include an image processing and feature extraction module, and a language model. We decided to fix these components, and we explain them in Chapter 4. Besides the simple image preprocessing, we present the handcrafted features and pixel values extracted from sliding windows, and we give the details about the language models built for each database and about the decoding method. Finally, we validate our choices by training GMM-HMMs, which will serve as baselines and bootstrapping systems for the next experiments.

Part III focuses on the recognition of handwritten documents with Hidden Markov Models and deep neural networks. We study Multi-Layer Perceptrons (MLPs) in Chapter 5 and Recurrent Neural Networks (RNNs) in Chapter 6.

We compare different inputs of the neural networks: handcrafted features and pixel values, but also how much context should the network consider to make accurate predictions. We investigate the architectures of the network, especially the number of hidden layers, but also their size, and the importance of recurrence in RNNs. We apply training techniques to refine the models, namely a sequence-discriminative training method which provides improvements for speech recognition, and the dropout technique, a simple and popular method to regularize the networks. The networks are evaluated in isolation, to measure their ability to model and classify their inputs, and in the complete recognition systems described in Part II.

We show that deep neural networks achieve significantly better results than shallow ones, on all databases and with different inputs. We observe that these improvements result from a better modeling capacity, rather than from the augmented number of

free parameters. We also show that with deep neural networks, the performance gap between raw pixel inputs and handcrafted features is reduced, both proving the capacity of such models to extract meaningful representations and alleviating the need for the design and implementation of relevant feature extraction. We report important gains in performance from sequence discriminative training of MLPs. Moreover, we show that the dropout technique is very effective in RNNs, and that the choice of positions to apply it, relative to the recurrent connections, can improve a lot the performance of the network.

Finally, in Part IV, we focus on how the neural networks are integrated with the higher level components: the HMMs and language model.

In Chapter 7, we study the impact of the number of HMM states per character on the performance of the MLPs and RNNs, alone and in the complete systems. In particular, in the CTC framework, the characters are represented by a single output, and the network also has a special output to predict no character. We evaluate the role of this special label, and the situations in which it is useful. Moreover, the CTC criterion uses a forward-backward procedure to avoid the need of a segmentation of the input before training, which is reminiscent of the training procedure of HMMs. We compare framewise and CTC training of MLPs and RNNs, and show that the CTC paradigm, as defined by Graves et al. (2006), is especially suited to RNNs.

In Chapter 8, we evaluate the importance of the decoding parameters and of the language model. We also compare the performances of our best systems, MLPs and RNNs, on simple pixel values and handcrafted features. We combine these models using two techniques, and show the complementarity of different modelings. We present the systems we submitted to the HTRtS contest, which aimed at evaluating the quality of the automatic transcription of ancient manuscripts, written by the philosopher Jeremy Bentham and his staff in the nineteenth century.

We report comparable results with MLPs and RNNs, and with handcrafted features and pixels. The different models achieve state-of-the-art performance on Rimes and IAM databases, and their combination outperformed the best published results.

We conclude this thesis with a summary and a discussion of the presented results and findings. In this last section, we also propose some perspectives for future research in the field.

Part I

HANDWRITING RECOGNITION — OVERVIEW

Chapter 1

Offline Handwriting Recognition – Overview of the Problem

Contents

1.1 Introduction . . . 35
1.2 Preliminary Steps to Offline Handwriting Recognition . . . 37
1.3 Reducing Handwriting Variability with Image Processing Techniques . . . 39
1.3.1 Normalizing Contrast . . . 39
1.3.2 Normalizing Skew . . . 40
1.3.3 Normalizing Slant . . . 41
1.3.4 Normalizing Size . . . 41
1.4 Extraction of Relevant Features for Handwriting Recognition 42
1.4.1 Text Segmentation for Feature Extraction . . . 42
1.4.2 Features for Handwriting Representation . . . 43
1.5 Modeling Handwriting . . . 46
1.5.1 Whole-Word Models . . . 47
1.5.2 Part-Based Methods . . . 47
1.5.3 Segmentation-Free Approach . . . 48
1.6 Modeling the Language to Constrain and Improve the Recognition . . . 49
1.6.1 Vocabulary . . . 50
1.6.2 Language Modeling . . . 51
1.6.2.1 Statistical n-gram Language Models . . . 51
1.6.2.2 Neural Network Language Models . . . 52
1.6.3 Open-Vocabulary Approaches . . . 53
1.7 Measuring the Quality of the Recognition . . . 53
1.8 Conclusion . . . 54

1.1 Introduction

Handwriting recognition is the process of transforming a digital representation of the physical result of handwriting into a digital text, generally for further treatments, such as indexing, classification, or translation. One may acquire handwritten text in different manners. For example, with the advent of tablets, touchscreens or digital pens, it is now possible to have access to many physical parameters of the writing process. Therefore, we can know the pen position at every time, and possibly the pen pressure, inclination, and so on. On the other hand, without such tools, we may only have the result of handwriting in the form of a scanned document. The handwritten text must be extracted from the image, using image processing techniques or relevant feature extraction.

The first case is called *online* recognition, and the second one *offline* recognition. Historically, the two have been separated, and a clear distinction is made is some surveys (*e.g.* Plamondon & Srihari (2000)), while others are only focused on one branch (*e.g.* Vinciarelli (2002)). While their nature make them suited to different applications, *e.g.* touchscreen input for the former, and cheque processing for the latter, the techniques employed nowadays to perform the recognition tend to be similar. Namely, they attempt to turn a sequence of feature vectors into a sequence of characters or words, modeling an input signal at lower levels and the language at higher levels. In this respect, these methods are also close to those applied in speech recognition.

Offline handwriting recognition is also related to the recognition of printed text from document images, a problem known as Optical Character Recognition (OCR). While both recognize text from images, printed text tend to be much more regular than handwriting, hence generally easier to process. In this thesis, we are concerned with the offline recognition of handwritten text.

We humans see the image as a whole and make sense of it regardless of its resolution and size. Computers, on the other hand, do not see the *big picture*. For them, an image is merely a matrix of pixels, which are numbers. Figure 1.1 shows an image of a word from IAM database at different scales. Recognition algorithms must therefore make sense of a two-dimensional map of numbers, so that the image is interpreted at a higher level.

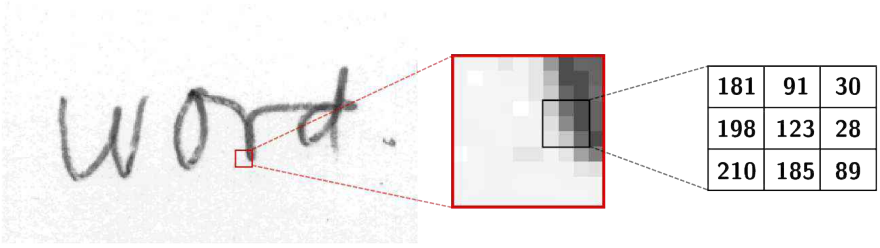


Figure 1.1: A digital image of a word at different scales.

Another difficulty for automatic recognition is that everyone has their own writing style. Some people write legibly, while some others are very hard to read (medical doctors are a common example), causing a lot of variability in the observed shapes of the same word. The tool used to write (*e.g.* pen, pencil) and the digitalization procedure also add to the diversity of handwriting. Figure 1.2 shows examples of the same word, written by different people.

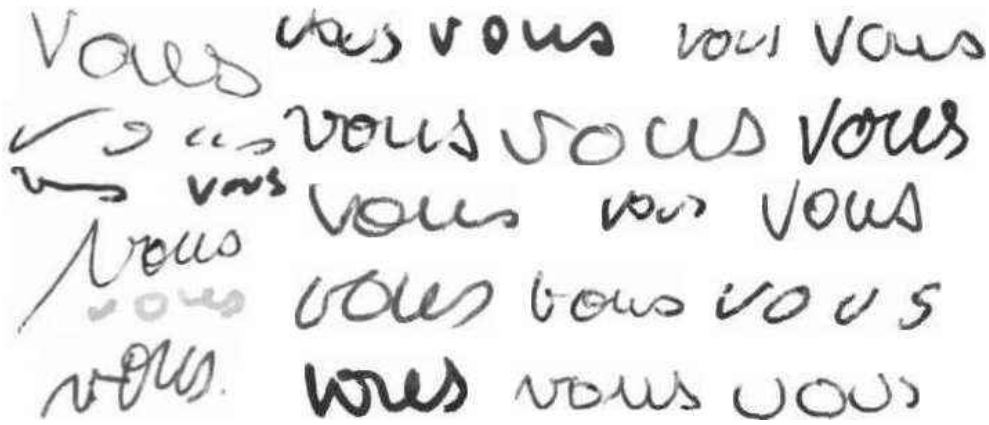


Figure 1.2: The French word *"vous"* written by different persons, from the Rimes database.

Character shapes also vary, even when written by the same person. Conversely, the same shape may also correspond to different characters. The four images on the left-hand side of Figure 1.3 look like reasonable examples of letter *'u'*. Actually, only one is (the blue one). They all come from the same word, from the Rimes database, shown on the right-hand side. The other examples are: *'n'*, *'en'*, and the beginning of an *'m'*. This shows that the identity of a shape can only be known in the context of the whole word, and illustrates the difficulty of cutting a word image into character images without knowing what the word is, known as Sayre's paradox (Sayre, 1973).

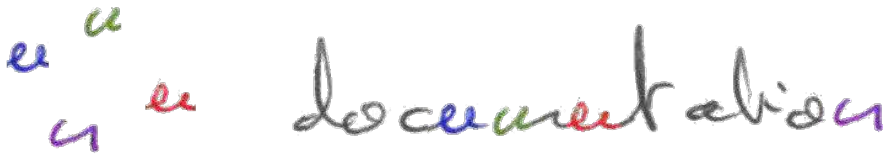


Figure 1.3: A similar shape can represent several handwritten characters.

Handwriting recognition has a long history, and has interested researchers since the 1950s (*e.g.* Bledsoe & Browning (1959)). The focus was first character recognition, especially handwritten digits recognition (*e.g.* for ZIP code recognition). As systems were improving, the interest shifted to word recognition, and the number of industrial

applications started to grow in the 1990s. The two main applications were address recognition for mail delivery (Kimura et al., 1993; Srihari, 2000; El-Yacoubi et al., 2002) and bank cheque processing (Le Cun et al., 1997; Guillevic & Suen, 1998; Gorski et al., 1999; Morita et al., 2001; Paquet & Lecourtier, 1993). Nowadays, recognition systems are applied to less constrained type of documents, involving larger vocabularies, such as the transcription of handwritten mail (Grosicki et al., 2009), *e.g.* for subsequent document classification (Louradour et al., 2012), information extraction (Chatelain et al., 2006), or the recognition of the text in ancient documents (Romero et al., 2011; Sánchez et al., 2013) for indexing and easy retrieval (Nion et al., 2013).

In this chapter, we present handwriting recognition through a selection of existing methods developed over the years to tackle this problem. Handwriting recognition consists of several steps, from the preparation of the image to the delivery of the recognized character or word sequences. Section 1.2 gives an overview of the potential preliminary steps required to obtain suitable inputs for the recognition systems. Generally, those are images of word or text lines, which must sometimes be extracted from document images.

Image processing techniques attempt to reduce the variability of writing style. Several examples are presented in Section 1.3, which normalize the image quality, the size of the writing, as well as a few common variations, such as slanted handwriting.

The extraction of relevant features from the image also eliminates some of the diversity, and aims at producing pertinent values which represent the problem in a data space where it is easier to solve. Examples of features for handwriting recognition, along with their extraction methods, are presented in Section 1.4.

Section 1.5 focuses on the models of handwriting, and how these models transform their inputs to produce characters or words. We will briefly overview three types of methods: namely whole word models, systems based on heuristics and explicit segmentation of the image, and segmentation-free approaches.

Finally, handwriting is a form of language representation. Usually, the characters to recognize are not random sequences, but existing words. Similarly, sequences of words normally convey a meaning, and form sentences that are syntactically, grammatically, and semantically coherent. Techniques to incorporate these constraints into the recognition process, adding linguistic knowledge to the models, are introduced in Section 1.6.

The interested reader can find surveys of offline handwriting recognition in (Steinherz et al., 1999; Plamondon & Srihari, 2000; Vinciarelli, 2002; Koerich et al., 2003).

1.2 Preliminary Steps to Offline Handwriting Recognition

Handwriting recognition systems operate on images of characters, words, or text lines. In practical applications, a digitalized document does not come in the desired form. The regions of interest must therefore be first extracted before being processed.

In some situations, *e.g.* very structured documents such as forms, the operation is trivial. Prior knowledge about the document structure can help to reliably retrieve text zones, for example in cheque and mail processing, or when a tabular structure is known.

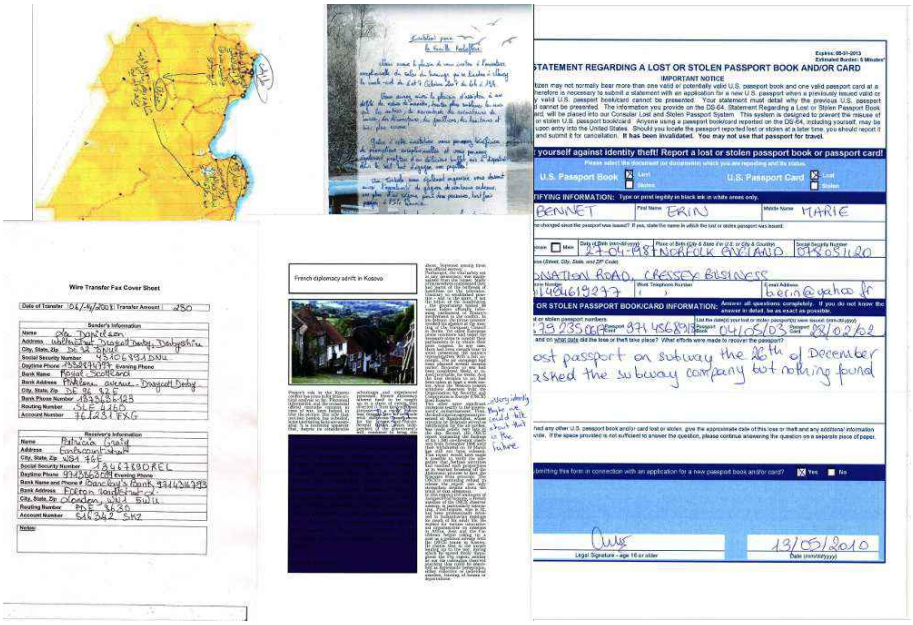


Figure 1.4: Various images from the Maurdor campaign (Brunessaux et al., 2014).

The recent Maurdor campaign (Brunessaux et al., 2014) aimed at evaluating a complete processing pipeline for heterogeneous document processing. Figure 1.4 shows the kind of documents of the database, which are representative of real-world applications, and illustrates the need of specialized methods for the extraction of handwritten parts before recognition.

First, the text in the document should be localized, and possibly grouped into text zones. For example, for mailroom applications, it is interesting to isolate the sender and recipient address, the core of the letter and the signature. In free forms, handwritten text zones might be detected, and separated from printed zones. When documents contain graphical elements, such as tables, diagrams, images, these must be discarded. This problem is known as text / non-text classification. The remaining zones of text should be organized in a meaningful way (*e.g.* one zone for each cell in a table, cf (Mao et al., 2003) for a survey of document structure analysis).

Sometimes, one should prepare the documents beforehand. The digitalization process may introduce noise in the image. The resolution can be an important aspect of the image (cf Figure 1.1), and may need to be normalized. Other examples of relevant operations include the correction of the document orientation (Hull, 1998) or the removal of line rulings (*e.g.* in Lopresti & Kavallieratou (2010); Shi et al. (2010)). When the language of the document is not known in advance, some methods can be applied

at the image level to determine the language, so as to pass the image to a specific recognition system (Ghosh et al., 2010).

Next, we should isolate text lines. This may be a difficult task, as shown by the organization of contests (Gatos et al., 2011), affected for example by the presence of accents, diacritics, or non-horizontal lines. Successive lines might also overlap, descenders and ascenders from different lines may touch one another, compromising simple methods based on projection profiles or connected components. Comprehensive surveys of text line segmentation for different applications are presented in (Likforman-Sulem et al., 2007; Louloudis et al., 2009).

As explained before, the segmentation of an image of cursive text into images of character is not an easy task. Yet, it is often more feasible to split the text lines into word images. In simple cases, this segmentation involves a connected component analysis or vertical projection profile. The result is obtained by deciding whether distances, between components, or between non-white columns of pixels, is indicative of a word separation (Louloudis et al., 2009). Word segmentation may be necessary, *e.g.* when the system models words directly, or when the feature extraction requires it. Moreover, one may want to keep several segmentation hypotheses, leaving the final decision to subsequent modules, as it is for example done in (Knerr et al., 1998; Menasri et al., 2012).

In this thesis, we focus on the recognition system, and thus assume that a reliable line segmentation is available, which provides either images of isolated lines or their bounding box coordinates in a document image.

1.3 Reducing Handwriting Variability with Image Processing Techniques

1.3.1 Normalizing Contrast

As we can see on Figure 1.2, in some images, the writing is darker, and it is fainter in others, either because of the pen or the digitalization. The background is also sometimes not plain white. Assuming that the light (white) pixels correspond to background and dark ones to foreground, *i.e.* writing, the dissimilarities across images can be reduced by contrast enhancement techniques.

Binarization is the crisper approach, mapping the range of pixel values from $[0, 255]$ (0 is black) to $\{0, 1\}$. In its simplest form it consists in setting a threshold value for pixel intensities, and set all values above it to 1, and those below to 0. Ostu adaptive thresholding (Otsu, 1979) avoids the choice of an arbitrary threshold. Instead, it considers the histogram of pixel values, and assumes that two modes are present: one for dark pixels (writing) and one for light ones (background). The thresholding value is automatically adjusted for every image to best separate the two modes. Local binarization methods are even finer, and examine local neighborhoods rather than

whole images (Sauvola & Pietikäinen, 2000; Gatos et al., 2006; Wolf et al., 2002). This kind of technique is especially suited to old or degraded documents.

Alternative and softer methods than binarization leave the image in the gray-level space. For example, Roeder (2009); Pesch et al. (2012) assume that the number of background pixels is much larger than the number of foreground ones. From the image histogram, they compute a threshold for the 5% darkest pixels, and map to 0 the values below it. Similarly, they find a threshold to map the 70% lightest pixels to white. A linear mapping is applied between the two thresholds. Espana-Boquera et al. (2011) perform this task with a neural network, trained to predict the pixel intensity in the clean image given the original one, using a local neighborhood.

1.3.2 Normalizing Skew

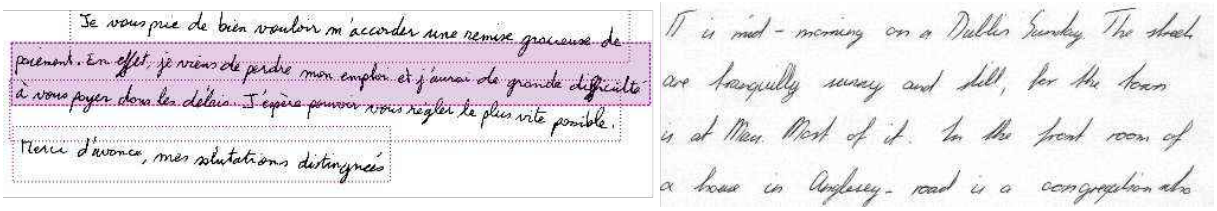


Figure 1.5: Example of skewed text line (Rimes database – left) and slanted handwriting (IAM database – right)

Text lines are generally not straight lines. Especially in free handwriting, it demands application and rigour to write on an (imaginary) horizontal line. An example of skewed line is shown on the left-hand side of Figure 1.5, from Rimes database. The writing goes down, and the bounding box of a line also includes part of the surrounding lines. Recognition systems must follow the text line, which is easier when the line is really horizontal.

Some methods estimate a global skew angle from the whole line image, and are generally based on the estimation of the baseline. Senior & Robinson (1998) compute the horizontal projection profile, heuristically remove the pixels corresponding to descenders and fit a line into the lower contour pixels. Vinciarelli & Luettin (2001) propose a similar method but they find the core region of the text with adaptive Otsu thresholding on the projection profile. Other methods for global skew correction are derived from algorithms applied to the whole document, such as (Bloomberg et al., 1995). A survey can be found in (Hull, 1998).

Some methods cope with more difficult situations. The text is often written on a curve rather than on an horizontal or sloped line. For example, in (Toselli et al., 2004; Vinciarelli et al., 2004; Pesch et al., 2012), text line images are split into segments to which line fitting methods such as those presented above are applied. In (Morillot et al., 2013b), a sliding window is scanned through the image, and angles are calculated, also with projection based methods, without requiring to explicitly split the image. Therefore, they propose a finer correction which better handles the fluctuations. Other

methods are based on run-length analysis (Shi & Govindaraju, 2003), or even neural networks (Espana-Boquera et al., 2011).

1.3.3 Normalizing Slant

The writing of some people tends to be inclined. An example of such slanted writing is shown on Figure 1.5. Slant correction methods roughly consists in finding the angle between the vertical axis and the strokes that should be vertical, which are mainly found in ascenders and descenders, and in applying a shear transformation to correct that angle. Bozinovic & Srihari (1989) first remove lines of pixels with long horizontal runs, and small horizontal regions between the removed lines. The remaining regions are horizontally divided in two. The slant angle is the average of the angles defined by the connection of the centers of gravity of each region. A similar method is presented in (Papandreou & Gatos, 2012).

Other works propose histogram-based methods. For example, Vinciarelli & Luettin (2001) compute the vertical pixel density for each slant angle in a given interval. The method require shearing the image for each considered angle, and the retained slant angle maximizes a measure of verticality. Pastor et al. (2004) state that images without slant yield projection profiles with maximum variance. They also shear the image with different angles, and the final one is the weighted average of those close to the maximal variance of vertical projection profiles. Buse et al. (1997) propose two methods. The first one consists in minimizing the entropy of the vertical projection profile. The second one uses the frequency domain to compute a weighted average of possible slant angles. Kozielski et al. (2013a) take advantage of several methods by selecting the median angle computed with three different projection-based algorithms.

Other approaches consider close to vertical strokes, and infer the angles from the 8-directional chain code of the thinned image (Kimura et al., 1993; Kim & Govindaraju, 1997; You & Kim, 2002) or from the contours (Marti & Bunke, 2001). Finally, as for skew, the slant angle can vary inside a word. Local approaches are required to achieve a finer correction. For example, Uchida et al. (2001) apply a Dynamic Programming (DP) algorithm to optimize a sequence of angles. They optimize a criterion which takes into account a confidence of the local angle – the height of black pixel components in a window slanted at different angle – and the smoothness of the sequence of angles. In (Espana-Boquera et al., 2011), the local sequence of angles is also a DP optimization of angles estimated with a neural network.

1.3.4 Normalizing Size

Writing size is also an important source of diversity across writers and documents. It includes the general area occupied by a word, as well as the size of some components (such as ascenders and descenders), the thickness of the stoke, and the tightness of writing. For height, rescaling all images to a common value is the simplest solution. However, the presence of ascenders/descenders will affect the size and the position of the core region. The size of ascenders and descenders can also be very different from a writer to another. Vinciarelli & Luettin (2001); Toselli et al. (2004); Pesch et al.

(2012) first find the regions for ascenders, descenders and core part, and scale each of them to a fixed height for a finer normalization. This method requires finding the baselines to determinate the regions, which is often already done for skew correction, using projection profiles (Bozinovic & Srihari, 1989; Vinciarelli & Luettin, 2001) or line fitting to the upper and lower contours (Toselli et al., 2004; Pesch et al., 2012). Espana-Boquera et al. (2011) find the zones locally by classifying pixels to belong to region extrema using a neural network. Romero et al. (2006) propose a methodology to determine to which height the images should be normalized. Finally, Kozielski et al. (2012) adopt a local approach based on a sliding window. The first and second order moments are computed, to reposition the window at the center of gravity, and rescale the writing so that each window have the same second order moments.

Differences in pens, or in the pressure applied to them during writing provoke stroke thickness variations. Among the different preprocessing operations presented in this section, the normalization of stroke width is the least used. The two main approaches consist in computing the skeleton of the black pixel components (*e.g.* in (Senior & Robinson, 1998)), in order to get a constant stroke width of 1px, or in applying morphological operations (erosion or dilatation) to yield a constant width (*e.g.* in Kozielski et al. (2012)). The interested reader can find a survey of thinning methods in (Lam et al., 1992).

1.4 Extraction of Relevant Features for Handwriting Recognition

Feature extraction is a next step where task-specific knowledge can be put in the system, and where unnecessary information can be discarded. In that sense, it complements pre-processing.

Ideally, the features should be representative of the problem, *i.e.* able to capture the important characteristics of handwriting, and discriminant, that is, simplify the problem such that different characters are farther in feature space than in original space.

Unlike speech recognition, where popular features have strong theoretical foundations in physiology or signal processing, the majority of features proposed for handwriting recognition are heuristic ones, handcrafted by researchers. However some of them (*e.g.* Gabor filters (Buse et al., 1997) or presence of ascenders/descenders (Côté et al., 1998)) can be related to the visual perception of humans.

1.4.1 Text Segmentation for Feature Extraction

When the system does not directly model words – in which case the features can be extracted from the whole word image – but characters, one should segment the image. As pointed out earlier, character segmentation is not an easy task for cursive text.

There are mainly two approaches for segmentation, both considering smaller regions of the image, corresponding to at most one character, and letting the recognition system

merge sub-parts into characters. More details about the two methods will be presented in Section 1.5. Basically, the first one consists in heuristically segmenting the ink into small chunks, at positions which are likely to be connections between characters. Features are extracted from the smaller images.

The second method does not make any assumptions or use any heuristic. It consists in scanning a sliding window through the image, and compute the features in each extracted frame. The main parameters of a sliding window are its width, which defines the amount of context considered at each position, and the overlap between two consecutive windows, or equivalently the step size to shift the window, controlling the number of observations extracted from a given image. Among proposed variations of the sliding windows, Doetsch et al. (2012) adjust the vertical position so that the center of the window coincide with the center of gravity of the foreground pixels at this position (Kozielski et al. (2012) shift the image with first-order moments), and Al-Hajj Mohamad et al. (2009) scan the image with windows slanted at different angles to account for slanted handwriting.

Finally, some features are sometimes extracted from different (horizontal or vertical) sub-regions of these images or sub-images. For example, in (Bianne-Bernard, 2011), there is one feature for each column of pixels in the sliding window, and some features computed in different horizontal regions (ascenders, core, descenders – found with the method of Vinciarelli & Luettin (2001)). Espana-Boquera et al. (2011) define 20 square cells organized vertically in each window and extract three features from each of them.

1.4.2 Features for Handwriting Representation

Features may be very simple (such as pixel intensities) or a higher-level representation of the image. Different complexities involve different amount of computation and image processing effort to obtain them. In this section, we present a selection of features used by handwriting recognition systems, organized by the level of abstraction from the image, starting from low-level features, close to the pixels values, to higher-level processing involving gradient computations, or simple shape detection. Finally, we present automatic feature extraction based on machine learning algorithms.

Feature transforms are sometimes applied. Their aim may be to decorrelate or reduce the dimensions of feature vectors, or to increase their discriminating power. Popular methods include PCA (*e.g.* in (Kozielski et al., 2012)), Linear Discriminant Analysis (LDA, *e.g.* in (Natarajan et al., 2008)), or Region-Dependent Transform (RDT, *e.g.* in (Chen et al., 2012)).

Pixels

In some situations, using the pixel intensities may be sufficient. For example, in (Kozielski et al., 2012, 2013a), a lot of effort is put on the image pre-processing, and even a simple recognition system yields good results with a Principal Components Analysis (PCA) to reduce and decorrelate the pixel dimensions, and only a few additional features. It may also be the case when the recognition system is powerful enough, *e.g.* in (Graves & Schmidhuber, 2009), or even (Bluche et al., 2014a), where no pre-processing is done.

Low-level pixel features

Simple features can be derived from the image easily by mere counting or averaging operations. Examples of such features are:

- count of black pixels (*e.g.* in (Marti & Bunke, 2000; Bertolami & Bunke, 2008))
- pixels density (proportion of black pixels, or average gray-level *e.g.* in (Knerr et al., 1998; Toselli et al., 2010; Bianne et al., 2011; Espana-Boquera et al., 2011))
- run-length analysis: count of consecutive black pixels in rows or columns (*e.g.* in (Favata & Srikantan, 1996; Knerr et al., 1998; Chen et al., 2012))
- number of transitions between black and white pixels in different regions (*e.g.* in (Marti & Bunke, 2000; Bertolami & Bunke, 2008; Morillot et al., 2013b)) or along the median line (Mohamed & Gader, 1996)
- Position or relative position of the center of gravity (*e.g.* in (Knerr et al., 1998; Bianne et al., 2011; Morillot et al., 2013b))
- Profile of the foreground pixels (left, top, right, and/or bottom) (*e.g.* in (Knerr et al., 1998)), or position of the contour (*e.g.* in (Marti & Bunke, 2000))
- Projection-profile based measures, such as percentiles of the cumulated number of black pixels in the vertical direction (Natarajan et al., 2001; Cao et al.)
- Second-order moments (*e.g.* in (Marti & Bunke, 2000; Kozielski et al., 2012))

When the image is not the result of a sliding window extraction, the size (height, width) or aspect ratio may be an interesting information, *e.g.* in (Kim & Govindaraju, 1997), or even the position of the sub-image with respect to the text baseline or the original image, *e.g.* in (Knerr et al., 1998).

High-level features

More sophisticated features, giving a higher representation of the shape, may be computed with simple operations on images. For example, an approximation of the derivative of the pixel intensities in each dimension (*e.g.* with Sobel filters) provides an information about the orientation of the edges, as well as their position (which corresponds to abrupt changes in intensity).

Derivatives of pixel intensities are used for example in (Natarajan et al., 2001; Toselli et al., 2010; Espana-Boquera et al., 2011; Chen et al., 2012). The orientation of contours are either estimated from the gradient (Marti & Bunke, 2000) or from chain coding (Kim & Govindaraju, 1997). Histograms of gradients (HoG) pool the different angles in discrete bins, locally or in the whole image. These are popular features in computer vision in general, and may be helpful for handwriting recognition too (Bianne-Bernard, 2011; Morillot et al., 2013b). Other traditional features for computer vision are also utilized, such as Speeded Up Robust Features (SURF, (Wang et al., 2012)), or Scale-Invariant Feature Transform (SIFT, (Rothacker et al., 2012)).

Buse et al. (1997) apply n Gabor filters to images and compute features for each connected component in the binarized result images (such as orientation, distance and overlap between components, area and centroid of the components). Gabor filters are also integrated in the feature calculation of Cao et al.. Natarajan et al. (2001) perform a linear regression of the black pixels of some regions, to calculate the angle of the obtained line, as well as the correlation of the pixels to it.

One may also look for different local configurations of pixels (El-Hajj et al., 2005; Bianne et al., 2011), which could carry more complicated information that mere orientation of edges. In their Gradient-Structural-Concavity feature set, Favata & Srikantan (1996) also count the number of local patterns of pixel organization.

A way of taking into account more context, or capture the dynamics of features, is to add their derivative, either estimated by the difference between the current value and that of the previous window (Dreuw et al., 2011b):

$$\Delta_t x = x_t - x_{t-1}$$

or with a regression (Bianne-Bernard, 2011):

$$\Delta_t x = \frac{\sum_{i=1}^{K} i \times (x_{t-i} - x_{t-i})}{2 \sum_{i=1}^{K} i^2}$$

Shape features

To capture a more global view of the shapes in the image, one can also look for common structural elements of handwriting, such as loops (Knerr et al., 1998; Guillevic & Suen, 1998), T-crossings and other junctions (Senior & Robinson, 1998), check the presence of ascenders and descenders (Leroux et al., 1991; Madhvanath & Govindaraju, 1996; Guillevic & Suen, 1998), or combine these observations to extract anchor points or key letter (*e.g.* in (Côté et al., 1998)).

Learning Features

Finally, automatically learning the features from the image, with machine learning techniques, prevents the designer of the system from making assumptions about what is relevant in the image and what is not. Feature learning methods belong to two approaches: *unsupervised learning*, where the only objective is to map the image onto a compact feature vector which accurately represent the image, and *supervised learning*, where the obtained representation is for example the by-product of the discriminative task of classifying the shape.

In **unsupervised methods**, the parameters of a model are optimized so that the image can accurately be restored after some transformation. For example, one may build a neural network with one or more hidden layers, and adjust the connection weights to minimize the mean square error between the output and the original image. This model is called Auto-Encoder (AE) (Cottrell & Munro, 1988), and can be interpreted as an encoder of the image into a representation made of the activations of a hidden

layer, and a decoder from that representation to the restoration of the image. Only the encoder part is kept to extract the features (Hammerla et al., 2010). Variations of this method are Sparse AEs (Poultney et al., 2006), Contractive Auto-Encoders (Rifai et al., 2011a,b), Denoising AEs (Vincent et al., 2008).

Other methods include Restricted Boltzmann Machines (a bipartite undirected probabilistic graphical model with observed – pixels – and latent – features – variables, each set being easily inferred from the other), or Sparse Coding Methods, which iteratively build a dictionary of base images and finds the optimal sparse decomposition of the input image into elements of it (Olshausen et al., 1996).

In these methods, the features are a compact representation of the image, carrying enough of the information, so that the image can be reconstructed from it.

Alternative techniques are based on **supervised learning** methods. A model, *e.g.* a neural network, is trained to predict shape classes such as characters, HMM states, or prototype class from clustering. Intermediate variables, *e.g.* the activations of a hidden layer, are supposed to be optimized to map the input features into a space where the classification is easier: they are automatically tuned to discriminate between different shapes.

For example, Kozielski et al. (2013a) extract the activations of recurrent layers of a Recurrent Neural Network, trained to predict HMM states. One can also take the outputs of the network (posterior features). In (Knerr et al., 1998), it corresponds to predictions of the image belonging to a given cluster of shapes. Dreuw et al. (2011a) take the log-PCA reduction of HMM predictions made by a multi-layer perceptron. In (Bluche et al., 2013a), we built a similar feature extractor with a convolutional neural network.

1.5 Modeling Handwriting

In this section, we focus on the modeling of handwriting, *i.e.* the transformation of feature vectors into digital text. We call this module *optical model*, similarly as the acoustic model in speech recognition, as opposed to the language model (described in the next section), which puts some constraints on the handwriting recognition. The choice of optical model is related to the previous steps, especially the features and the segmentation of the input. Therefore, we divided the existing approaches in three broad categories. First, we will present systems that directly model whole words. Then, we describe methods applied to images segmented in sub-characters. Finally, we introduce some segmentation-free approaches.

The last kind is the most popular nowadays, in particular the association of a sliding window feature extraction with hidden Markov models or using neural networks. This is also the approach chosen in this thesis, and more details and related approaches will be given in the next chapter.

1.5.1 Whole-Word Models

In this first class of methods, the goal is to recognize a word directly as a whole, without relying on a segmentation or on an explicit representation of the parts. Often, a simplified representation of the word shape is extracted from the images and matched against a lexicon, for example with simplified upper and lower profiles (Parisse, 1996), or with holistic features such as ascenders, descenders, loops and so on (Madhvanath & Govindaraju, 1996; Madhvanath & Krpasundar, 1997; Guillevic & Suen, 1998). Côté et al. (1998) feed holistic features to a three-level network. The feature level is connected to a character level, itself connected to a word level. The intermediate character level allows an iterative bottom-up and top-down approach to find the features in the image which confirm the recognition of a word.

The disadvantage of whole word modeling is the inherent limitation of the vocabulary size. The number of models grows linearly with the number of words. Moreover, for methods based on a matching to a prototype, a new prototype must be created with every added word. Classifiers must be retrained. The method of Côté et al. (1998), with the intermediate character level, and hand-wired connections, could scale well with lexicon size, but the holistic approach might show its limitations with many different words with a similar shape. Most of these methods are applied to legal amount recognition in bank cheque processing, were the vocabulary does not exceed 40-50 words.

1.5.2 Part-Based Methods

In part-based methods, the image is divided into sub-regions corresponding to at most one character. We have already stressed the difficulty to segment characters from cursive words, without knowing the word's identity. Thus these systems often perform an over-segmentation of the image, or consider several segmentation alternatives. The goal is to merge different part to get complete characters, or to find the correct segmentation into characters among the different hypotheses.

The first step consists in finding possible segmentations from the image. It is often derived from heuristics about character connections. For example, Edelman et al. (1990) extract control points from vertical and horizontal extrema of the contour to extract simple strokes. Chen et al. (1995) apply mathematical morphology operators to isolate regularities, which are potential segmentation points. Then, a shortest path computation between the leftmost and rightmost segments allow to select the segmentation points among candidates, and heuristics are applied to split, merge and order segments. In (Baret, 1990; Knerr et al., 1998), the segmentation also uses regularity and singularity principles. In (Kim & Govindaraju, 1997), the segmentation process tries to find ligatures between characters from the contour, either by comparing with the average stroke width or by looking at concavity and convexity features. Morita et al. (2001) look at the local minima of the upper contour. They shift the segmentation point when a loop or an unacceptable width is found. A similar method can be

found in (El-Yacoubi et al., 1999). Tay et al. (2001) cut the image horizontally into slices of width chosen according to the height of core region of the word. The width is adjusted so that the number of black pixels traversed by the cut is locally minimal. In (Favata & Srikantan, 1996), several horizontal segmentation points are found with different algorithms. Bengio et al. (1995) combine online information (such as the pen velocity) and offline information to generate segmentation hypotheses, and use heuristics to limit the number of merges in the segmentation graph. A survey of character segmentation techniques was proposed by Casey & Lecolinet (1996).

For the actual recognition, we can differentiate the combination approach and the grapheme approach. In the combination approach, adjacent segments are merged to produce different character segmentation hypotheses, generally represented as a directed graph. In (Edelman et al., 1990) the stroke are recognized by prototype alignment, and characters are hypothesized from combinations of strokes. The word is found with a best-first search. Kim & Govindaraju (1997) represent characters by different code words obtained through a k-means clustering, and compute the distance from merged segments features to these code words. The character can also be predicted with k-nearest neighbors (Favata & Srikantan, 1996), or neural networks (Bengio et al., 1995; Tay et al., 2001). The segmentation graph is then transformed into a prediction graph, in which the best path corresponds to the word recognition. Lexical constraints may be taken into account (*e.g.* in (Favata & Srikantan, 1996)). In (Tay et al., 2001), the character predictions are used as emission probabilities in a hidden Markov model, which also serves to train the neural network at the word level.

In the grapheme approach, the part of characters (the graphemes) are directly modeled by the system, and a character is defined as a sequence of such units. Generally the characters are modeled by Hidden Markov Models (HMMs). El-Yacoubi et al. (1999); Morita et al. (2001) extract simple features and use discrete HMMs. In (Knerr et al., 1998), the graphemes in the training set are clustered into 100 classes, and a neural network is trained to predict the grapheme class. The predictions are fed to the HMM. Chen et al. (1995) build a continuous HMM with Gaussian mixture models emission probabilities. To cope with possible under-segmentations (*i.e.* a grapheme corresponds to more than one character), some HMMs include a skip transition, *e.g.* in (Knerr et al., 1998; El-Yacoubi et al., 1999).

1.5.3 Segmentation-Free Approach

In the segmentation-free approach, the recognition is accomplished without an explicit segmentation of the image, thus without relying on heuristics to find character boundaries, and limiting the risk of under-segmentation. This category of approaches is the most popular nowadays, receiving a lot of research interest, and achieving the best performance on standard benchmarks. In this section, we only present a brief overview of this kind of methods. They are mainly based on hidden Markov models and neural networks, which is the topic of the next chapter. More details on the modeling techniques will be presented there.

The first strategy consists in extracting a sequence of feature vectors with a sliding window (Kaltenmeier et al., 1993). The main parameters of the window are its width and shift, defining respectively the amount of context included in the feature vector and the overlap between two observations. Hidden Markov models are especially suited to transform the sequence of feature vectors into a sequence of characters. Each character is modeled by an HMM with several states. An emission model defines the probability of a state generating an observation, *i.e.* a feature vector. This probability model may be discrete (*e.g.* in (Kaltenmeier et al., 1993)), continuous (usually a Gaussian mixture model, *e.g.* in (Marti & Bunke, 2001)), or even estimated with more complex systems, such as neural networks (Bourlard & Morgan, 1994). A survey of Markov models for handwriting recognition can be found in (Plötz & Fink, 2009).

In the second strategy, the system directly models the two-dimensional image input. For example, Chevalier et al. (2005); Lemaıtre et al. (2008) use a two-dimensional HMM with Gaussian modeling of spectral features. Wang et al. (2012) match keypoints (SURF) of an image to reference keypoints of each character class. The final sequence of characters is found with a dynamic programming search from the classification of keypoints. Other notable techniques are based on neural networks. More particularly convolutional neural networks, in which the two dimensional structure of the image is kept in several layers. In this structure, local receptors are defined and repeated at each position in the image. The outputs of each receptor can be arranged in two-dimensional maps, allowing to apply the concept again, and iteratively extract more complex representations. Subsampling operations, such as max pooling, reduce the size of the maps. At the end, a classification is performed at each horizontal position to provide character sequence hypotheses. This concept was for example applied in (Bengio et al., 1995; LeCun et al., 1998), and used in the very successful Multi-Dimensional Recurrent Neural Networks (Graves & Schmidhuber, 2009).

In the models presented in this section, the input is not segmented. The segmentation of the image into characters is either performed after the recognition (*e.g.* in (Wang et al., 2012)), or a by-product of the recognition. For example, word or sentence recognition is achieved by concatenation of character models with HMMs, and the recognition consists in finding the most likely sequence of states, from which the character boundaries can usually be trivially inferred. To some extent, convolutional neural networks also allow to retrieve the general position of a character in the original image. Some systems, such as RNNs, model the sequence directly, and the segmentation might not be easily recovered.

1.6 Modeling the Language to Constrain and Improve the Recognition

Not any sequence of characters makes a syntactically or semantically correct sentence of existing words. Actually, the total number of possible character sequences is far larger than the space of acceptable word sequences. As we have seen in the previous

section, many recognition processes involve a search. Knowing that we deal with language allows us to constrain the search space, in order to get meaningful outputs.

In some applications, such as the recognition of reference/client code or phone number (*e.g.* in (Kermorvant et al., 2009)), a simple grammar, a regular expression or a graph representing it may be sufficient. For natural, unconstrained text there are mainly two aspects: how to select a vocabulary, *i.e.* a list of acceptable words, and how to build a Language Model (LM) to force the output sequence of words to be correct.

1.6.1 Vocabulary

When the system models words directly, the vocabulary is already embedded in the recognition engine, and limited to the set of words modeled, which is usually relatively small. In other cases, a vocabulary constrains the sequences of characters to form words from a predefined set. It helps alleviate ambiguities arising in the recognition procedure, and limits the size of the search space.

However, the content of the vocabulary also defines the recognizable words. Thus its content is crucial: any word not in the vocabulary cannot be recognized. Depending on the application, the size of the vocabulary may vary from tens of words, *e.g.* for cheque processing, to millions for unconstrained text recognition.

As the vocabulary size grows, so does the search space. Moreover, there are more words close to one another. On the other hand, the chance to encounter an Out-of-Vocabulary word (OOV) decreases. As a consequence, the choice of a vocabulary is usually a tradeoff between size and coverage. In practical situations, for example for unconstrained text transcription, coverage might be hard to estimate precisely.

In the experiments presented in this thesis, we focused on public databases with known ground-truth, hence the coverage is measured in terms of OOV rate, *i.e.* the proportion of words in the test dataset that are not in the vocabulary. The OOV rate is a lower bound on the error rate.

The vocabulary puts constraints on the recognition, and it can be organized in intelligent ways for an efficient retrieval of the recognized word. The simplest approach consists in computing a score for each word of the vocabulary independently, and return the word with the highest score. This is inefficient, since the complexity grows linearly with the vocabulary size. Moreover, it does not take advantage of the fact that many words may share letters at the same position.

In the prefix tree approach, the vocabulary is organized as a tree, which root corresponds to the beginning of a word. Each branch is associated with a character, and a terminal nodes contains the word made of the letters along the path. Therefore, the common prefix between several words will only be examined once. Moreover, the tree structure allows to apply conventional (and efficient) search algorithms.

A review of large vocabulary reduction and organization techniques for handwritten word recognition was proposed by Koerich et al. (2003).

Finite-State Transducers (FSTs) (Mohri, 1997) are another interesting representation. FSTs are directed graphs with a set of states, transitions, initial and final states.

The transitions from one state to another are labeled with an input symbol from an input alphabet, and an output symbol from an output alphabet. When provided with a sequence of input symbols, the FST follows the successive transitions for each symbol, and emits the corresponding output symbols. A valid sequence allows to reach a final state with such a series of transitions.

In the FST representation of a vocabulary (Mohri, 1996), the input alphabet may consist of the different characters modeled, and the output alphabet to the vocabulary. This representation is interesting, because FST operations, such as composition, determinization, or minimization, permit the integration of the language model (cf. next section) in the search graph, while keeping a compact structure similar to the prefix tree. FST representations of linguistic knowledge are popular in speech recognition (Mohri et al., 2002), and are applied to handwriting recognition (*e.g.* in (Toselli et al., 2004)) and in several recognition toolkits such as Kaldi (Povey et al., 2011) or RWTH-OCR (Dreuw, 2011).

1.6.2 Language Modeling

Even when the system outputs tokens from a vocabulary, not all sequences of words are valid, *i.e.* grammatically or semantically correct. Checking the grammatical validity of a sentence may not be easy, but many techniques are developed in the field of Natural Language Processing (NLP). Simple parsing with a set of grammar rules could allow to discard some word sequence hypotheses. For example, Zimmermann et al. (2006) rescore a list of sentence hypotheses with a probabilistic context-free grammar.

More generally, language modeling for handwriting recognition usually consists in giving a score to different word sequence alternatives. The score measures how likely is the observation of the given sequence of words. The common approach is statistical language modeling, overviewed in (Rosenfield, 2000), and in (Bunke et al., 2004) for the specific application to handwriting recognition. This class of methods estimates the *a priori* probability of observing a word sequence $\mathbf{W}$ from large amounts of digital texts. In this section, we focus on two popular methods, namely n-gram language models, and connectionist language models based on neural networks.

To measure the suitability of a language model to a given corpus, one usually computes its perplexity. It is derived from the entropy of the probability model, and can be expressed as follows:

$$PPL = (p(w_1, \ldots, w_{N_w}))^{-\frac{1}{N_w}} = 2^{\frac{1}{N_w} \sum_{k=1}^{N_w} \log_2 p(w_k | w_{k-1}, \ldots)} \tag{1.1}$$

where N_w is the number of words in the text. LMs with smaller perplexities are generally better at predicting a word given the history. However, lower perplexities do not guarantee better recognition results, as pointed out by (Klakow & Peters, 2002).

1.6.2.1 Statistical n-gram Language Models

The space of all possible word sequences is very large. Estimating a probability distribution over this space, even with very large corpora, is difficult, especially because

most word sequences will not appear. Factorizing $p(\mathbf{W})$ with the chain rule

$$p(\mathbf{W}) = p(w_1, \ldots, w_N) = p(w_1)p(w_2|w_1)\ldots p(w_N|w_1, \ldots, w_{N-1}) \tag{1.2}$$

yields a representation in which the probability of observing a word only depends on the previous words. The n-gram approach addresses the data scarcity problem by making Markovian assumptions: the probability of a word does not depend on the position of the word in the sequence, and only on an history of $n-1$ previous words:

$$p(w_k|w_1, \ldots, w_{k-1}) = p(w_k|w_{k-1}, \ldots, w_{k-n+1}) \tag{1.3}$$

The Maximum Likelihood estimator of n-gram probabilities is achieved by mere counting in the corpus:

$$p(w_k|w_{k-1}, \ldots, w_{k-n+1}) = \frac{C(w_k, w_{k-1}, \ldots, w_{k-n+1})}{\sum_w C(w, w_{k-1}, \ldots, w_{k-n+1})} \tag{1.4}$$

As n grows larger, the chances of having zero counts for some n-grams increases. For example, with a vocabulary of 10,000 words, there are a trillion different trigrams to be observed in the training corpus. Various techniques exist to overcome this problem. The simplest one is additive smoothing, which consists in adding a small value δ to all counts. Good-Turing smoothing (Good, 1953) transfers some of the probability mass of n-grams seen $c+1$ times to those seen c times. Witten & Bell (1991) propose another discounting method. Other methods involve n-grams of lower orders, such as simple linear interpolation (Jelinek & Mercer, 1980), or backing-off (Katz, 1987; Ney et al., 1994; Kneser & Ney, 1995). Chen & Goodman (1996) present a good overview and comparison of various smoothing techniques including those cited above.

Other models similar or related to n-grams exist, but are not used much in state-of-the-art systems, such as class-based language models (Brown et al., 1992), in which the n-gram probability is decomposed as

$$p(w_k|w_{k-1}, \ldots, w_{k-n+1}) = p(w_k|C_k)p(C_k|w_{k-1}, \ldots, w_{k-n+1})$$

or skip-grams (Guthrie et al., 2006) and word trigger pairs (Tillmann & Ney, 1996), which allow longer dependencies between words.

N-gram language models can easily be represented by weighted Finite-State Transducers (Mohri et al., 2008), which can be composed with the lexicon to build the search graph including all the linguistic constraints.

1.6.2.2 Neural Network Language Models

Connectionist language modeling is another approach worth mentioning, although it had received little attention in handwriting applications so far. These models are more often found in the automatic translation or speech recognition literature. They are based on neural networks.

The basic idea of these methods is to project each word of the history in a continuous space and to perform a classification with neural networks to predict the next word. For

example, Schwenk & Gauvain (2002); Bengio et al. (2003) use a multi-layer perceptron on the projection of the last n words. Mikolov et al. (2010) predict the next word with a recurrent neural network, which prevents from defining a fixed history, it being encoded in the memory of the recurrent layer.

Since the output space is very large in large vocabulary applications, special output structures have been proposed to train and use the network efficiently, *e.g.* in (Morin & Bengio, 2005; Mikolov et al., 2011; Le et al., 2011). In (Graves et al., 2013a), the acoustic and language models are integrated in a single recurrent neural network.

1.6.3 Open-Vocabulary Approaches

We have introduced n-gram models (or language models in general) for word sequences, but they can as well serve to model sequences of characters. The advantages of working with characters rather than words are multiple. First, as already mentioned for recognition models, there are less different tokens, and much more data for character modeling. Therefore, we can estimate more reliable probability distributions, and build higher order n-grams. Moreover, although the outputed words will not necessarily valid ones, there would be not such things as OOVs.

Several works considered mixing word LMs limited to a vocabulary, and character LMs to recognize potential OOVs. For example, in his PhD thesis, Bazzi (2002) proposed a model where the lexicon (list of words) and filler model (looping on phones) were examined. The word language model contains an OOV token guiding the search towards the phone language model. In FST search graphs (or FST representations of language models), replacing all OOV arcs by the character LM might consume too much memory. For handwriting recognition, Kozielski et al. (2013b) use a dynamic decoder, where the integration of the LM is done on the fly, which enables them to explore the character LM only when the OOV arc is explored. Messina & Kermorvant (2014) propose an over-generative method which limits the number of places where the character LM should be inserted, allowing this hybrid LM to be plugged in static decoding graphs.

1.7 Measuring the Quality of the Recognition

In order to assess the quality of a recognition system, a measure of the performance is required. For isolated character or word recognition, the mere accuracy (proportion of correctly recognized items) is sufficient. In the applications considered in this thesis, the transcript is a sequence of words. Counting the number of completely correct sequences is too coarse, because there would be no difference between a sentence with no correct word and another with only one misrecognition.

In a sequence, we do not only find incorrect words, but there might also be inserted or deleted words. Measures such as precision and recall may take these types of errors into account, but not the sequential aspect. The most popular measure of error, used in international evaluations of handwriting or speech recognition, is based on the Levenstein edit distance (Levenshtein, 1966).

This distance counts the number of edit operations required to transform one string into another. The possible edits are:

- *substitution* of one item for another
- *deletion* of one item of the sequence
- *insertion* of one item in the sequence

The minimum edit distance between two strings can be retrieved efficiently with a dynamic programming algorithm. The Word Error Rate (WER) is obtained by computing the minimum number of edits from the reference string to the output transcript, normalized by the number of reference words:

$$WER = \frac{n_{sub} + n_{ins} + n_{del}}{n_{ref}} \tag{1.5}$$

where $n_{ref}, n_{sub}, n_{ins}, n_{del}$ are respectively the numer of words in the reference, and the number of substituted, inserted, and deleted words in the hypothesis. Note that although it is generally expressed in percentage, it may go beyond 100% because of the potential insertions. In this thesis, the reported WERs are computed with the `SClite` implementation (Fiscus, 1998).

Similarly, we can consider an even finer measure of the quality of the output sequence in terms of characters, which penalizes less words with a few wrong characters and is less dependent on the distribution of word lengths: the Character Error Rate (CER). It is computed like the WER, with characters instead of words. The whitespace character should be taken into account in this measure, since this symbol is important to separate words. This measure is gaining interest with open-vocabulary recognition systems, which can output potentially any sequence of characters.

1.8 Conclusion

In this chapter, we have presented the problem of offline handwritten text recognition, including the different steps to solve it. We provided examples of methods and applications of each of these steps. This overview is of course not exhaustive. Handwriting recognition has been an active research area for more than half a century, and international conferences dedicated to the processing of documents and to the recognition of handwritten text are held every year.

For this thesis, we did not work on the image preprocessing, feature extraction, or language modeling, but we merely applied existing methods presented in this chapter. We focus on optical modeling, following the segmentation-free approach presented in Section 1.5.3. More precisely, we use the sliding window method, and build HMM-based models. We study deep neural network optical models. Thus, in the next chapter, we will present in more details these models, along with their training methods and applications to handwriting recognition.

Chapter 2

Handwriting Recognition with Hidden Markov Models and Neural Networks

Contents

2.1 Introduction 57
2.2 Hidden Markov Models for Handwriting Recognition 58
2.2.1 Definition 59
2.2.2 Choice of Topology 61
2.2.3 Choice of Emission Distribution 62
2.2.4 Model Refinements 62
2.2.5 Decoding 63
2.3 Neural Networks for Handwriting Recognition 63
2.3.1 The Multi-Layer Perceptron 63
2.3.2 Recurrent Neural Networks 66
2.3.3 Long Short-Term Memory Units 67
2.3.4 Convolutional Neural Networks 68
2.4 Handwriting Recognition Systems with Neural Networks . . 69
2.4.1 The Hybrid NN/HMM scheme 70
2.4.2 Predicting Characters 71
2.4.3 NN Feature Extractors 71
2.5 Training Models 72
2.5.1 Training Hidden Markov Models with Generative Emission Models 72
2.5.2 Training Neural Networks 73
2.5.2.1 Neural Network Training: an Optimization Problem . . 73
2.5.2.2 The Backpropagation Algorithm 74
2.5.2.3 Backpropagation Through Time 75

2.5.2.4 Regularization 76
2.5.3 Training Deep Neural Networks 77
2.5.3.1 Statement of the problem 77
2.5.3.2 Unsupervised pre-training 78
2.5.3.3 Supervised pre-training 80
2.5.4 Training Complete Handwriting Recognition Systems 81
2.5.4.1 Bootstrapping 81
2.5.4.2 Forward-Backward training of Hybrid NN/HMM 81
2.5.4.3 Connectionist Temporal Classification 82
2.5.4.4 Graph-Transformer Networks 85
2.5.4.5 Sequence-Discriminative 85
2.6 Conclusion 86

2.1 Introduction

Hidden Markov Models (HMMs) are particularly suited to model sequential signals. They are the model of choice in many problems involving sequences, such as speech recognition (Rabiner & Juang, 1986), computational biology (*e.g.* to model protein sequences (Krogh et al., 1994)), or handwriting recognition (Kaltenmeier et al., 1993). The sequential nature of text lends itself well to this kind of modeling. Character models may be concatenated to form word or sentence models, in the same way as phone models are in speech recognition.

However, unlike in speech processing or in computational biology, the inputs of offline handwriting recognition are not naturally sequential, they are images, *i.e.* maps of pixels in two dimensions. Although a few approaches have been proposed to apply Markov models to the two-dimensional signal that images form (Park & Lee, 1998; Chevalier et al., 2005; Lemaıtre et al., 2008), the common approach consists in extracting sequences of observations, and model these instead.

As pointed out in Section 1.5, sequences of observations are either retrieved by extracting and segmenting the ink (foreground pixels) in the image (*e.g.* in (Chen et al., 1995; Knerr et al., 1998)), or by scanning a sliding window in the reading order over the image (Kaltenmeier et al., 1993), and extracting features at each position.

Historically, HMMs have become attractive in speech and handwriting recognition for their capability to perform time alignment, and for the maximum likelihood formulation of the parameter estimation. In that formulation, their emission model is generative. However, for pattern recognition problems, discriminative models are preferred. Neural networks are such discriminative models, which enjoy considerable interest. They are very popular in pattern recognition applications, and suited to classification problems, where the task is to predict a class to which an input belongs. They have been applied to handwriting recognition, *e.g.* by LeCun et al. (1989) for handwritten digits.

While *"simple"* neural networks require a fixed size input to make a prediction, architectures have been proposed to deal with variable sized images and output sequences, such as Space-Displacement Neural Networks (Bengio et al., 1995), which are extensions of Convolutional Neural Networks (ConvNNs, (LeCun et al., 1989)), or Multi-Dimensional Recurrent Neural Networks (Graves & Schmidhuber, 2008).

Several solutions were proposed to train and use neural networks directly for handwriting recognition, without relying on HMMs (*e.g.* Graph-Transformer Networks (Bottou et al., 1997; Le Cun et al., 1997) or Connectionist Temporal Classification (Graves et al., 2006)), as well as neural networks able to deal with sequences, such as Recurrent Neural Networks (RNNs). Yet they are often combined with HMMs, applied to each observation vector, to replace the standard Gaussian mixture modeling (Bourlard & Morgan, 1994).

In this chapter, we focus on the recognition of handwritten text with hidden Markov models and neural networks. In a first part, we present the general aspects of HMMs and their specific application to handwriting recognition in Section 2.2. In a second time, we give an overview of neural networks in Section 2.3.

We show some applications of neural networks in the task of handwriting text recognition in Section 2.4. More specifically, we present the hybrid NN/HMM framework (Bourlard & Morgan, 1994), where neural networks are optical models in HMMs, and the tandem association of neural networks and HMMs (Hermansky et al., 2000), where neural networks are used to extract features for conventional GMM-HMMs. We also present some neural networks for handwriting recognition, which are not used in combination with HMMs.

Finally, in Section 2.5, we explain the training procedures for HMMs and neural networks. We present the issues arising for deep neural networks, and some proposed solutions. We also introduce training methods for complete handwriting recognition systems.

We conclude in Section 2.6 with an overview of the chosen approach in this work, in comparison with related methods presented in this chapter.

We will only give an overview of the different techniques and models found in the literature, and describe in more details the methods used in the following of the thesis, *e.g.* HMMs, the Connectionist Temporal Classification framework, the pre-training of deep neural networks, or the dropout technique.

2.2 Hidden Markov Models for Handwriting Recognition

The problem of handwritten text line recognition is now widely approached with hidden Markov models (Plötz & Fink, 2009) (the first applications of HMMs to that problem date from (Levin & Pieraccini, 1992; Kaltenmeier et al., 1993)). These models are designed to handle sequential data, with hidden states emitting observations. There are well-known techniques to *(i)* compute the probability of an observation sequence given a model, *(ii)* find the sequence of states which is most likely to have produced an observed sequence, and *(iii)* find the parameters of the model to maximize the probability of observing a sequence (Rabiner & Juang, 1986). The third problem is the actual training of these models, while the second allows to decode a sequence of observation. In this paradigm, the characters are each represented by a hidden Markov model. A simple concatenation of these produce word models. The advantages are twofold:

- from a few character models (around one hundred), we can build word models for potentially any word of the language. Thus this method is much more scalable the large vocabulary problems than systems attempting to model each word separately

- recognizing a word from character models does not require a prior segmentation of the word image into characters. Since the word model is a hidden Markov model of its own, the segmentation into characters is a by-product of the decoding procedure, which consists in finding the most likely sequence of states.

The second point extends to the line recognition problem: we do not need to split the text line into words to recognize word sequences.

Automatic speech recognition is a famous example of successful application of HMMs. This field has an experience of over 25 years of these systems, and many advances there can be transferred to handwriting recognition.

Amongst many aspects, handwriting recognition benefits from speech recognition techniques, such as decoding with linguistic constraints (vocabulary), adding statistical language models in the hidden Markov model framework (Marti & Bunke, 2000), the integration of these components in a Finite-State Transducer decoder (Mohri et al., 2002). Moreover, tools built for speech recognition, such as HTK (Young et al., 1997), or Kaldi (Povey et al., 2011) can be used for HMM-based handwriting recognition without modification.

2.2.1 Definition

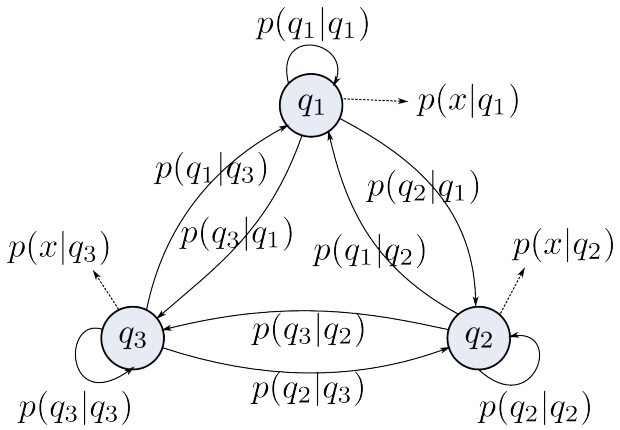


Figure 2.1: A Hidden Markov Model.

An HMM (Figure 2.1) is a doubly stochastic process, that we can represent as a probabilistic graphical model. It is made of states, and transitions between them defined by probabilities. It is a first-order Markov process: the state at time $t+1$ is independent of the state at time $t-1$, given the state at time t.

In HMMs, the observed information is not the state sequence, but a sequence of data $\mathbf{x} = x_1 \dots x_T$ generated by HMM states with an emission probability model.

An HMM λ is defined by the following elements:

- A set of states $\mathcal{Q} = \{s_1, \dots, s_N\}$. For convenience, we also define the set of all state sequences of length T : $\mathcal{Q}_T = \{\mathbf{q} = q_1 \dots q_T \ : \ q_i \in \mathcal{Q}, 1 \leq i \leq T\}$.

- A transition model, defined by the probabilities $p(q_t = s_i|q_{t-1} = s_j)$, for all $s_i, s_j \in \mathcal{Q}$.
- A probability distribution over initial states $p(q_1 = s), \forall s \in \mathcal{Q}$.
- An emission model, defined by the probabilities $p(x|s)$, where $s \in \mathcal{Q}$, and x is an observation, which can be drawn from a finite set in discrete HMMs, from $\{0,1\}^D$ in Bernoulli HMMs (Giménez et al., 2010), or from $\mathbb{R}^D$ for continuous HMMs (where D is the dimension of the observation).

The Markovian assumptions provide an easy formulation of the likelihood of an observation sequence $\mathbf{x} = x_1 \dots x_T$ given a state sequence $\mathbf{q} \in \mathcal{Q}_T$:

$$p(\mathbf{x}|\mathbf{q}, \lambda) = \prod_{t=1}^{T} p(x_t|q_t) \tag{2.1}$$

because the individual observations are independent given the current state, and of the probability of the state sequence $\mathbf{q} \in \mathcal{Q}_T$ in the model λ:

$$p(\mathbf{q}|\lambda) = p(q_1) \prod_{t=2}^{T} p(q_t|q_{t-1}; \lambda) \tag{2.2}$$

.

Therefore, the likelihood of an observation sequence under the model is obtained by summing over all state sequences:

$$p(\mathbf{x}|\lambda) = \sum_{\mathbf{q} \in \mathcal{Q}_T} p(q_1)p(x_1|q_1) \prod_{t=2}^{T} p(x_t|q_t)p(q_t|q_{t-1}; \lambda) \tag{2.3}$$

The summation over all possible state sequences is expensive to compute naively. The *forward-backward* procedure (Baum et al., 1967) alleviates this problem, by a recursive computation of so-called forward variables :

$$\begin{aligned} \alpha_t(s) &= p(x_{1:t}, q_t = s|\lambda) \\ \alpha_1(s) &= p(q_1 = s) \times p(x_1|s) \\ \alpha_t(s) &= p(x_t|q_t = s) \times \sum_{r \in \mathcal{Q}} \alpha_{t-1}(r)p(q_t = r|q_{t-1} = r; \lambda) \end{aligned}$$

The *forward-backward* procedure also defines backward variables, which are used for training HMMs (Section 2.5.1):

$$\begin{aligned} \beta_t(s) &= p(x_{t+1:T}|q_t = s, \lambda) \\ \beta_T(s) &= 1 \\ \beta_t(s) &= \sum_{r \in \mathcal{Q}} p(q_{t+1} = s|q_t = s; \lambda)p(x_{t+1}|q_{t+1} = r)\beta_{t+1}(r) \end{aligned}$$

Consequently, we have $p(\mathbf{x}|\lambda) = \sum_{s \in \mathcal{Q}} \alpha_T(s)$.

Finally, we can retrieve the most likely state sequence $\mathbf{q}^* \in \mathcal{Q}_T$ of a HMM λ given the observed sequence $\mathbf{x}$, that is:

$$\mathbf{q}^* = arg\max_{\mathbf{q}\in\mathcal{Q}_T} p(\mathbf{q}|\mathbf{x},\lambda) = arg\max_{\mathbf{q}\in\mathcal{Q}_T} \frac{p(\mathbf{q},\mathbf{x}|\lambda)}{p(\mathbf{x})} \tag{2.4}$$

with Viterbi algorithm (Viterbi, 1967). Ignoring $p(\mathbf{x})$, which is the same for all state sequences, it consists in replacing the summation by a maximization in the recurrence of the forward variable, and keeping track of the maximizing state r at each iteration.

In the last fifteen years, HMMs have become the standard model for handwriting recognition. An HMM is built for each character, allowing to concatenate these models to obtain word or sentence models. The recognition of a line therefore does not require a prior segmentation into characters.

2.2.2 Choice of Topology

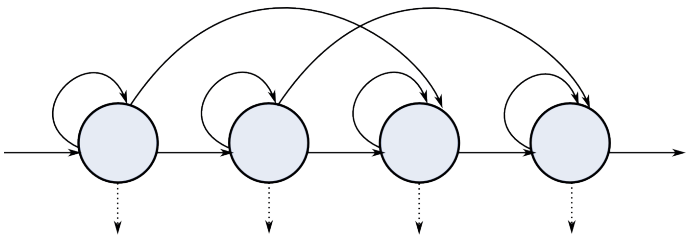


Figure 2.2: Hidden Markov Model topology for Handwriting Recognition.

The topology of explicit segmentation models takes into account the possible cutting of a character into several *graphemes*, usually one to three or four. It sometimes includes a skip transition to accept potential under-segmentation. Examples of such topologies may be found in (El-Yacoubi et al., 1999; Tay et al., 2001; Augustin, 2001).

In the sliding window approach, HMMs have a left-right topology, depicted on Figure 2.2, modeling the sequential progression of the window. Allowed transition for a given state are to itself, to the next one, and sometimes to the second next.

The number of states may be the same for all character HMMs, chosen empirically or heuristically. It can be adjusted to acknowledge the different length of characters. The letter *"i"* is for example likely to be written shorter than *"m"*. Automatic methods to adapt the length of individual HMMs were proposed by Augustin (2001); Zimmermann & Bunke (2002); Schambach (2003); Dreuw et al. (2008); Bianne-Bernard et al. (2012). Günter & Bunke (2004) compare several optimization strategies for choosing the number of states along with the training method.

2.2.3 Choice of Emission Distribution

The common emission models of HMM states are Gaussian Mixture Models (GMMs):

$$p(x|s) = \sum_{m=1}^{M} c_{sm}\mathcal{N}(x; \mu_{sm}, \Sigma_{sm}) \tag{2.5}$$

where $\mathcal{N}(\cdot, \mu, \Sigma)$ is the multivariate Gaussian distribution with mean μ and covariance matrix Σ. c_{sm}, μ_{sm} and Σ_{sm} are respectively the weight, mean, and covariance of the m-th component of the GMM for state s. The number of Gaussians M in GMMs is usually set to 1 at the beginning of training and increased during the EM procedure. In order to limit the number of parameters of the system, and to simplify the estimation procedure, diagonal covariance matrices are defined. However, this implies that the dimensions of the input features vectors must be decorrelated.

Giménez et al. (2010) used a Bernoulli distribution to model observation made of binary pixel values:

$$p(x|s) = \sum_{m=1}^{M} c_{sm} \prod_{d=1}^{D} p_{smd}^{x_d}(1 - p_{smd})^{1-x_d} \tag{2.6}$$

In the hybrid NN/HMM scheme presented in Section 2.4.1, the emission model is derived from the outputs of a neural network, providing state posterior probabilities $p(s|x)$:

$$p(x|s) = \frac{p(s|x)}{p(s)}p(x) \tag{2.7}$$

where $p(s)$ is the prior probability of state s, and $p(x)$ is not computeable in general, and can be ignored since it cancels out in recognition.

2.2.4 Model Refinements

Some modeling improvements were inspired from speech recognition. For example, context-dependent models correspond to the triphone approach, which models the co-articulation effect occurring in speech. In handwriting, it would model the cursive nature of the text, where the shape of connections between characters depends on the letters involved. In the context-dependent approach, there is one HMM for each character, given each possible left and right character. Since the number of states and distributions to estimate is very large, and most contexts do not appear in training sets, the emission probability distributions are shared among different HMM states, across context-dependent models. This is usually achieved by a decision tree, and the distribution is retrieved by asking questions regarding the character, the context, and the HMM state number. Examples of context-dependent models for handwriting recognition were presented in (Fink & Plotz, 2007; Natarajan et al., 2008; Bianne et al., 2011; Hamdani et al., 2014).

Writer adaptation is to handwriting recognition what speaker adaptation is in speech recognition. Techniques such as MLLR (Leggetter & Woodland, 1995), CMLLR (Gunawardana & Byrne, 2001), SAT (Anastasakos et al., 1996) are also applied to handwriting recognition, *e.g.* in (Vinciarelli & Bengio, 2002; Dreuw et al., 2009; Kozielski et al., 2013a). The improvements due to these methods, however, tend to be smaller than for speech recognition.

2.2.5 Decoding

The goal of handwriting recognition is to retrieve the most likely word sequence $\mathbf{W}^*$ given a sequence of observation vectors $\mathbf{x}$, which is achieved by maximizing the conditional probability $p(\mathbf{W}|\mathbf{x})$. Using Bayes' rule, the problem is formulated as:

$$\mathbf{W}^* = arg\max_{\mathbf{W}} \frac{p(\mathbf{x}|\mathbf{W})p(\mathbf{W})}{p(\mathbf{x})} = arg\max_{\mathbf{W}} p(\mathbf{x}|\mathbf{W})p(\mathbf{W}) \tag{2.8}$$

The prior probability of the observation sequence $p(\mathbf{x})$ is generally difficult to compute, and may be ignored since it is constant for all word sequences.

$\mathbf{W}$ is represented by its HMMs, and $p(\mathbf{x}|\mathbf{W})$ is described by Equation 2.3. The Viterbi algorithm presented in Section 2.2.1 can be applied to this maximization problem, and provides an exact solution at the state sequence level. On the other hand, $p(\mathbf{W})$ is provided by a language model, as described in Section 1.6.2.

2.3 Neural Networks for Handwriting Recognition

Neural Networks (NNs) are popular systems for pattern recognition in general. They are made from basic processing units, linked to each other with weighted and directed connections, such that the output of some units are inputs to others. The appellation *"(Artificial) Neural Network"* comes from the similarity between the units of these models and biological neurons.

The first formal description of artificial neural networks was proposed by McCulloch & Pitts (1943). Algorithms to adjust the weights of the connections lead to the perceptron (Rosenblatt, 1958), and Multi-Layer Perceptrons (MLPs, Rumelhart et al. (1988)). Among different kinds of neural networks, Recurrent Neural Networks (RNNs), able to process sequences, and Convolutional Neural Networks (ConvNNs, (LeCun et al., 1989)), suited to image inputs, are worth noticing.

2.3.1 The Multi-Layer Perceptron

The perceptron (Rosenblatt, 1958) is a binary classifier, which goal is to take a *"yes/no"* decision. The output y can take two values, corresponding to a negative and positive decision, and can be formulated as

$$y = f(\mathbf{x}) = sign(b + w_1x_1 + \ldots + w_nx_n) \tag{2.9}$$

where $\mathbf{x} = x_1, \ldots, x_n$ is an input feature vector, $b, w_1, \ldots, w_n$ are the free parameters (weights)[1], and $sign(x) = 1$ if $x > 0$ and -1 otherwise. Other choices of functions which can take only two values, such as the *heaviside* function, could be used instead of the *sign* function. However, for practical reasons, a differentiable (and continuous) function is preferred. A natural choice is the *sigmoid* function

$$\sigma(t) = \frac{1}{1 + e^{-t}} \tag{2.10}$$

which resemble the *heaviside* function, but is continuous and differentiable. We can still perform binary classification by taking a decision from the outcome of the sigmoid:

$$y = \begin{cases} 0, & \text{if } \sigma(\mathbf{w}^T\mathbf{x} + b) < 0.5 \\ 1, & \text{otherwise} \end{cases} \tag{2.11}$$

where $\mathbf{w} = [w_1 \cdots w_n]$ is the weight vector and b is called bias.

The thresholding of the outcome of the sigmoid defines the decision boundary of the binary classifier as a hyperplane in the input space, defined by the equation

$$b + w_1x_1 + \ldots + w_nx_n = 0 \tag{2.12}$$

Therefore, the perceptron is optimal only when the classification problem can be linearly separated in that space.

Parallels have been drawn between this model and biological neurons. In this analogy, the inputs x_i are received from other neurons through synapses. The weights w_i represent the strength and the excitatory/inhibitory nature of the synaptic connection. In the cell body, the contribution of all input connections are summed, and if greater than a predefined threshold, the neuron fires. Although this is a simplistic and inaccurate model of a neuron, this model is called *(artificial) neuron*, and associations of such neurons – *i.e.* connecting the output of one to the input of other ones – defines *(artificial) neural networks*.

In the following, we denote by $a = \sum_{i=1}^{N} w_i x_i + b$ the activation of a neuron (before the non-linear activation function), and by z its output.

Multi-Layer Perceptrons (MLPs, Rumelhart et al. (1988)) are an example of artificial neural networks, where the neurons presented in the previous section are connected to each other. An MLP, as its name indicates, contains neurons organized in layers. Instead of the single perceptron, several neurons are connected to the same inputs $x_1, \ldots, x_n$, with a different set of weights. The outputs of all these neurons are inputs for a new layer of neurons.

Considered altogether, the weights of each neuron k ($\mathbf{w}_k^{(i)}$), define a weight matrix from layer L_{i-1} to layer L_i: $\mathbf{W^{(i)}}$. Thus the output (vector) of a given layer L_i can be computed as the multiplication of the input vector $\mathbf{y^{(i-1)}}$ by the weight matrix $\mathbf{W^{(i)}}$,

[1] not to be confused with w_k in the previous chapter, which denoted words in sequences

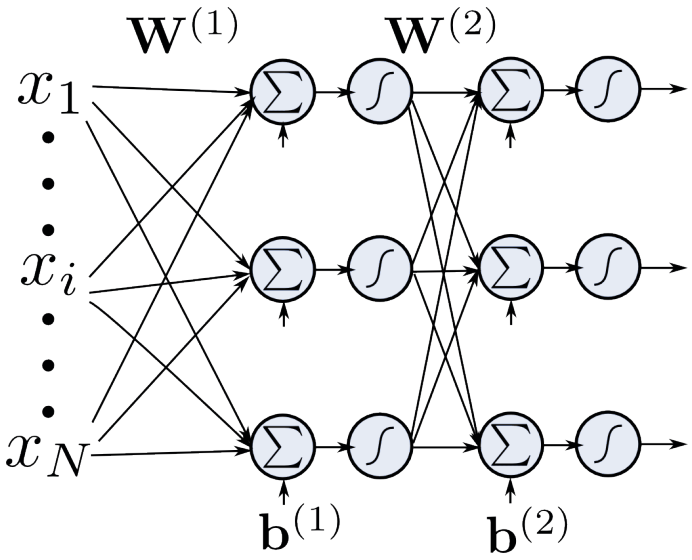


Figure 2.3: Multi-Layer Perceptron. $x_1, \ldots, x_N$ are the inputs, $\mathbf{W}^{(i)}, \mathbf{b}^{(i)}$ are the weight matrice and bias vector of layer i.

the addition of a bias vector $\mathbf{b^{(i)}}$, and the element-wise application of a non-linear function f_i.

$$\begin{aligned} y^{(1)} &= f_1(\mathbf{W^{(1)}} \cdot \mathbf{x} + \mathbf{b^{(1)}}) \\ &\vdots \\ y^{(i)} &= f_i(\mathbf{W^{(i)}} \cdot \mathbf{y^{(i-1)}} + \mathbf{b^{(i)}}) \\ &\vdots \\ y^{(N)} &= f_N(\mathbf{W^{(N)}} \cdot \mathbf{y^{(N-1)}} + \mathbf{b^{(N)}}) \end{aligned} \tag{2.13}$$

This organization in successive layers have some advantages. Since a given layer only receives inputs from previous layers and provides inputs for next layers, the output of the network can be computed in a single *feed-forward pass*, by sequentially determining the output of each layer. This is also interesting for the backpropagation algorithm, as we will see in the next section.

The neurons of the last layer of the MLP are linear binary classifiers sharing the same input features. Thus an MLP with several outputs is a multi-class classifier. It was shown (Bourlard & Wellekens, 1989) that the outputs of the network can be interpreted as posterior probabilities. The *softmax* function (Bridle, 1990b) is often applied instead of the *sigmoid* function at the output layer. For n neurons with activations $a_1, \ldots, a_n$, the *softmax* function is defined as follows:

$$z_i = softmax(a_i) = \frac{e^{a_i}}{\sum_{k=1}^{n} e^{a_k}} \tag{2.14}$$

With this function, the outputs $z_i \in]0, 1]$ sum up to one, *i.e.* $\sum_{i=1}^{n} z_i = 1$, and define probability distribution over the different classes, conditioned on the inputs

of the network. The advantages of such a property are *(i)* that the classification is associated with a confidence score with a meaningful interpretation on the one hand, and *(ii)* that the network can be a component of a larger system, where the posterior probabilities are important on the other hand, as is the case in hybrid NN/HMM systems.

2.3.2 Recurrent Neural Networks

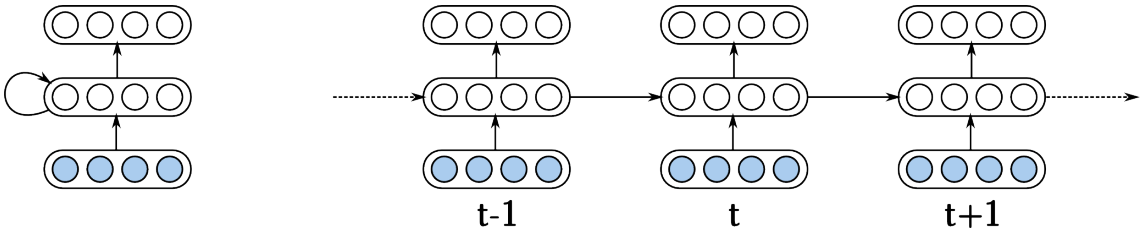


Figure 2.4: Recurrent Neural Networks, simple form

Recurrent Neural Networks (RNNs) are networks with a notion of internal state, evolving through time, achieved by recurrent connections. Hopfield networks (Hopfield, 1982) are an early form of recurrent neural network, where the recurrence is used to find a stable state rather than process time sequences.

Saul & Jordan (1990); Elman (1990) proposed neural network architectures for sequences of input vectors. These networks contain one hidden layer, as well as a separate *context* layer, implementing a memory in the network.

In its simplest form, an RNN is an MLP – *i.e.* neurons are organized in different layers – with recurrent layers. A recurrent layer does not only receive inputs from the previous layers, but also from itself, as depicted on the left-hand side of Figure 2.4.

The activations a_k^t of such a layer evolve through time with the following recurrence

$$a_k^t = \sum_{i=1}^{I} w_{ki}^{in} x_i^t + \sum_{h=1}^{H} w_{kh}^{rec} z_h^{t-1} \tag{2.15}$$

where x_is are the inputs and w_{ki}^{in} the corresponding weights, and z_h^{t-1} the layer's outputs at the previous timestep and w_{kh}^{rec} the corresponding weights.

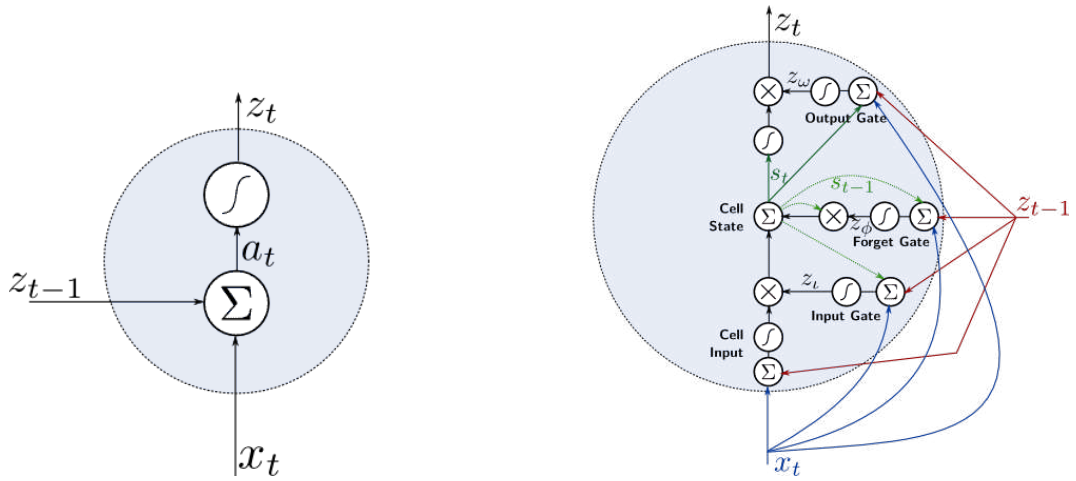
Bidirectional RNNs (BRNNs, Schuster & Paliwal (1997)) process the sequence in both directions. In these networks, there are two recurrent layers: a forward layer, which take inputs from the previous timestep, and a backward layer, connected to the next timestep. Both layers are connected to the same input and output layers. Graves et al. (2007) added convolutional aspects to build Multi-Dimensional RNNs (MDRNNs), which process an input image with four directions in recurrent layers.

Robinson (1994) proposed simple recurrent neural networks for speech recognition, and Senior (1994); Lee & Kim (1995); Senior & Robinson (1998) applied simple recurrent neural networks to handwriting recognition.

2.3.3 Long Short-Term Memory Units

In RNNs, the vanishing gradient issue prevents the network to learn long time dependencies. Hochreiter & Schmidhuber (1997) proposed improved recurrent neurons called Long Short-Term Memory units. In LSTM, the flow of information is controlled by a gating system, scaling the input information, the output activation, and the contribution of the internal state of the unit at the previous timestep to the current state, based on the input and recurrent information and the cell internal state.

An LSTM cell is shown on Figure 2.5, and compared to a basic recurrent neuron. The cell input and all gates receive the activation of the lower layer and of the layer at the previous timestep. The following equations define the behaviour of the LSTM unit.



(a) Simple Recurrent Neuron (b) Long Short-Term Memory Unit

Figure 2.5: Neurons for RNNs: (a) Simple Neuron (b) LSTM unit

The **Input Gate** controls whether the input of the cell is integrated in the cell state

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} z_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1}$$

$$z_\iota^t = f(a_\iota^t)$$

The **Forget Gate** controls whether the previous state is integrated in the cell state, or if it is forgotten.

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} z_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1}$$

$$z_\phi^t = f(a_\phi^t)$$

The **Cell state** is the sum of the previous state, scaled by the forget gate, and of the cell input, scaled by the input gate.

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} z_h^{t-1}$$

$$s_c^t = z_\phi^t s_c^{t-1} + z_\iota^t g(a_c^t)$$

The **Output Gate** controls whether the LSTM unit emits the activation $h(s_c^t)$.

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} z_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t$$

$$z_\omega^t = f(a_\omega^t)$$

The **Cell output** is computed by applying the activation function h to the cell state, scaled by the output gate.

$$z_c^t = z_\omega^t h(s_c^t)$$

The activation function f for the gates is usually the sigmoid function. Doetsch et al. (2014) propose a scaling factor in the activation, shared among all gates of each type: $z_\iota^t = f(\lambda_\iota a_\iota^t)$, $z_\omega^t = f(\lambda_\omega a_\omega^t)$, and $z_\phi^t = f(\lambda_\phi a_\phi^t)$. These factors are learnt, and make the gates more selective. The activation functions g and h are usually the sigmoid or hyperbolic tangent function.

The dependency of gate activations a_ι^t, a_ϕ^t and a_ω^t on cell state(s) corresponds to the so-called *peephole connections* (Gers & Schmidhuber, 2000; Gers et al., 2003), which are especially useful when the network should learn precise timings.

The LSTM cells provide abilities that standard RNNs lack, such as learning simple grammars (Hochreiter & Schmidhuber, 1997; Gers & Schmidhuber, 2001) or music composition (Eck & Schmidhuber, 2002). LSTM-RNNs have been successfully applied to phoneme (Graves & Schmidhuber, 2005) and speech (Graves et al., 2013a) recognition, and to handwriting recognition.

With Long Short-Term Memory neurons in recurrent layers, Bidirectional and Multi-Dimensional RNNs achieve very good results in handwriting recognition, and constitute the state-of-the-art in that domain (Doetsch et al., 2014; Graves & Schmidhuber, 2008; Bluche et al., 2014a; Moysset et al., 2014).

2.3.4 Convolutional Neural Networks

Convolutional Neural Networks (ConvNNs, LeCun et al. (1989)) have a structure similar to MLPs, but each layer is not fully-connected to the previous one. They implement the notion of local receptors, *via* local connections and weight sharing. The input is the two-dimensional image, and neurons in the hidden layers are organized in two-dimensional maps, each looking for a single feature. Each neuron of a map is only connected to a small neighborhood in the previous maps (or input image), with the same connection weights as other neurons of the map (Figure 2.6). One may interpret it as convolutions of the image with trainable filters.

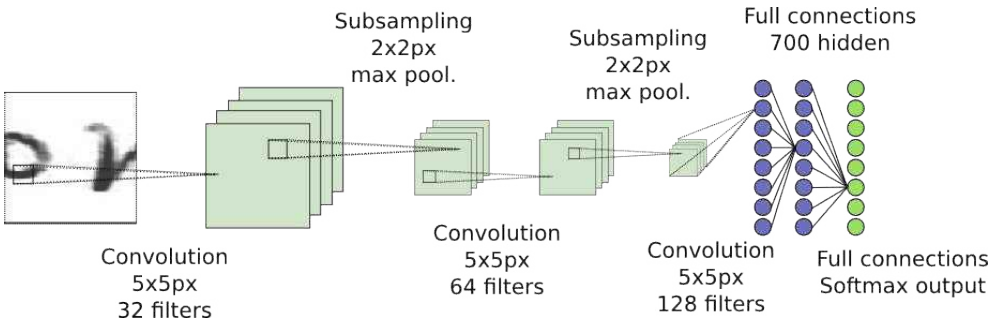


Figure 2.6: Example of Convolutional Neural Network.

The convolutional layers are often followed by subsampling layers, to limit the size of the network and implement some invariance to distortions. The subsampling operation may be a max-pooling or averaging over small neighborhoods. Usually, a few features (or maps, or trainable filters) are extracted in lower layers, and increasingly more features are extracted in upper layers, while the dimensions of the maps decrease (*e.g.* in LeNet-5 by LeCun et al. (1998)).

ConvNNs designed for classification can be easily extended to variable sized images, hence producing sequences of predictions, thanks to the local connections and shared weights. Such a structure is called Space-Displacement Neural Network (SDNN, Matan et al. (1991); Bengio et al. (1995)).

Finally, the aspects of ConvNNs may also be implemented in more complex architecture than MLPs, as for example in Multi-Dimensional RNNs (MDRNNs, Graves et al. (2007)).

ConvNNs are popular in computer vision, and achieve excellent results in various tasks (*e.g.* object recognition (Szegedy et al., 2014), image segmentation (Farabet et al., 2013)). They have also recently been applied to speech recognition (Abdel-Hamid et al., 2012). They were initially used to recognize handwritten digits (LeCun et al., 1989), and later handwritten text (Bengio et al., 1995; Le Cun et al., 1997; LeCun et al., 1998). More recently, we combined them with HMMs for word recognition on public databases (Bluche et al., 2013a,b).

2.4 Handwriting Recognition Systems with Neural Networks

As already mentioned in previous sections, different kinds of neural networks have been applied to the problem of handwriting recognition:

- **MLPs** in explicit segmentation systems (Knerr et al., 1998) or in segmentation-free approaches to predict HMM states (Espana-Boquera et al., 2011) or to extract features for GMM-HMM systems (Dreuw et al., 2011a).

- **ConvNNs** for character prediction in segmentation graphs (Bengio et al., 1995) or the SDNN version to predict sequences of characters (LeCun et al., 1998). We also used them in the HMM framework to classify HMM states or to extract features (Bluche et al., 2013a,b)

- **RNNs**, able to transform an input sequence into a sequence of predictions, have already been applied to handwriting recognition in (Senior, 1994). With LSTM cells, RNNs are now the state-of-the-art in handwriting recognition, and used in many research groups (Kozielski et al., 2013a; Morillot et al., 2013b; Moysset et al., 2014; Strauß et al., 2014). The multi-dimensional aspect (Graves et al., 2007; Graves & Schmidhuber, 2009), suited to images, was first applied to handwriting recognition.

Deep neural networks applications to handwriting recognition are however limited to convolutional architectures (LeCun et al., 1998; Graves & Schmidhuber, 2009), where the number of parameters is limited by the local receptors and weight sharing aspects, and the extraction of only a few features in lower layers.

Densely connected neural networks with more than one or two hidden layers are found in handwriting recognition, but limited to isolated character recognition (Ciresan et al., 2010; Cireşan et al., 2012). An application to keyword spotting is proposed by Thomas et al. (2013).

In this section, we present how neural networks are integrated in complete handwriting recognition systems: either replacing GMMs as the emission model of HMMs, or as character predictors. In the tandem approach, they are also used to extract features.

2.4.1 The Hybrid NN/HMM scheme

In the hybrid approach (Bourlard & Morgan, 1994), GMMs are replaced by neural networks for the emission model of HMMs. The NN does not provide generative likelihoods $p(x_t|s)$, but discriminative state posteriors $p(s|x_t)$. We can use Bayes' rule:

$$p(x_t|s) = p(x_t)\frac{p(s|x_t)}{p(s)} \tag{2.16}$$

Equation 2.8 becomes:

$$\begin{aligned} p(\mathbf{W},\mathbf{x}) &= p(\mathbf{W})\sum_q \prod_t p(x_t|q_t)p(q_t|q_{t-1},\mathbf{W}) \\ &= \prod_t p(x_t)p(\mathbf{W})\sum_q \prod_t \frac{p(q_t|x_t)}{p(q_t)}p(q_t|q_{t-1},\mathbf{W}) \end{aligned}$$

$\prod_t p(x_t)$ being the same for all word sequences in decoding, it does not influence the result, and we can effectively use $\frac{p(s|x_t)}{p(s)}$.

H. Bourlard and his colleagues thoroughly studied the theoretical foundations of hybrid NN/HMM systems in (Bourlard & Wellekens, 1989; Bourlard & Morgan, 1994; Renals et al., 1994; Konig et al., 1996; Hennebert et al., 1997). In particular, they show

in (Konig et al., 1996) how a discriminant formulation of HMMs (Bourlard & Wellekens, 1989), able to compute $p(\mathbf{W}|\mathbf{x})$ leads to a particular MLP architecture predicting local conditional transition probabilities $p(q_t|q_{t-1}, x_t)$, which allow to estimate global posterior probabilities.

Hybrid models for handwriting recognition were built, with different neural networks. Dreuw et al. (2011a); Espana-Boquera et al. (2011) use an MLP. Bengio et al. (1994a) built an hybrid ConvNN/HMM. In (Bluche et al., 2013b,a), we applied ConvNNs in the hybrid framework, for handwritten word recognition, using different segmentation methods.

2.4.2 Predicting Characters

Neural networks can also predict characters directly. This is for example the case in some recognition systems based on explicit segmentation (Bengio et al., 1995; Tay et al., 2001).

In segmentation-free approaches, the input of such neural networks is often the whole image. Due to the variable size of the image, the neural networks in this case implement some replication of the weights. ConvNNs may be trained to predict characters, and then applied to whole images (SDNNs, Bengio et al. (1995)). RNNs are already suited to recognize sequences, with a replication of the weights across time. They may implement convolutional aspects too, for example in MDRNNs. In the Connectionist Temporal Classification framework (Graves et al., 2006), RNNs are trained to predict sequences of labels, *e.g.* characters.

Some of these approaches also acknowledge the fact that no character may be present at a given position, by adding a non-character class in the output of the network. We find this kind of *"junk"* class for MLPs in (Rashid et al., 2012) for printed text, in (Tay et al., 2001) with an explicit segmentation system, and in the formulation of Connectionist Temporal Classification proposed by Graves et al. (2006) for RNNs.

2.4.3 NN Feature Extractors

As pointed out in Section 1.4.2, NNs may be feature extractors for a different system, for example a GMM-HMM. In the tandem approach (Hermansky et al., 2000), the network is also trained to predict HMM states or characters, but the posteriors are not directly used for optical modeling. Instead, the network acts as a feature extractor. Dreuw et al. (2011a) use the posteriors or an MLP as features for a GMM-HMM.

In (Bluche et al., 2013a,b), the posterior features are extracted from a ConvNN, for handwritten word recognition. One may extract the intermediate activations in hidden layers as features. For example, Kozielski et al. (2013b) build a GMM-HMM with the activations of LSTM layers of an RNN trained to classify HMM states.

Auto-encoders are neural networks trained to output their inputs. In the hidden layers, a compact representation of the input is computed. One can extract this representation as features, as presented for example in (Hammerla et al., 2010).

2.5 Training Models

2.5.1 Training Hidden Markov Models with Generative Emission Models

The standard method to adjust the parameters of HMMs consists in maximizing the likelihood of the observed sequences in a training set. This is achieved with an Expectation-Maximization (EM) procedure (Dempster et al., 1977), alternating the computation of an objective function for a fixed HMM λ, and the re-estimation of the parameters to maximize the criterion, yielding a new HMM $\hat{\lambda}$.

This iterative procedure for HMM is the Baum-Welch training (Baum et al., 1967, 1970). In the first step, the forward-backward procedure is applied to calculate the posterior probability of seeing state s at time t (with α and β defined in Section 2.2.1):

$$p(q_t = s|\mathbf{x}, \lambda) = \frac{\alpha_t(s)\beta_t(s)}{\sum_{r\in\mathcal{Q}} \alpha_t(r)\beta_t(r)} \tag{2.17}$$

and of seeing state s at t and r at $t+1$:

$$p(q_t = s, q_{t+1} = r|\mathbf{x}, \lambda) = \frac{\alpha_t(s)p(x_{t+1}|q_{t+1} = r)p(q_{t+1} = r|q_t = s, \lambda)\beta_{t+1}(r)}{\sum_{u\in\mathcal{Q}} \alpha_t(u)\beta_t(u)} \tag{2.18}$$

The probabilities of the transition and emission models can be updated using these values and the observation sequence $\mathbf{x}$ (Juang, 1985). The criterion maximized by this procedure is the auxiliary function:

$$Q(\lambda, \hat{\lambda}) = \sum_{\mathbf{q}\in\mathcal{Q}_T} p(\mathbf{q}|\mathbf{x}, \lambda) \log p(\mathbf{x}, \mathbf{q}|\hat{\lambda}) \tag{2.19}$$

which ensures an increase in likelihood (Baum et al., 1967):

$$\max_{\hat{\lambda}} Q(\lambda, \hat{\lambda}) \Rightarrow p(\mathbf{x}|\hat{\lambda}) \geq p(\mathbf{x}|\lambda) \tag{2.20}$$

Thus, Baum-Welch training guarantees to find a local maximum of likelihood.

An alternative to Baum-Welch training is Viterbi training. The forward-backward procedure is replaced by Viterbi decoding, and the equations become

$$p(q_t = s|\mathbf{x}, \lambda) = \begin{cases} 1 & \text{if } q_t^* = s \\ 0 & \text{otherwise} \end{cases} \tag{2.21}$$

$$p(q_t = s, q_{t+1} = r|\mathbf{x}, \lambda) = \begin{cases} 1 & \text{if } q_t^* = s \text{ and } q_{t+1}^* = r \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

Gradient-based techniques have also been proposed for maximum likelihood training, *e.g.* by Baldi & Chauvin (1994); Krogh (1994).

For some applications, such as handwriting or speech recognition, we are not really interested in the HMM ability to model observation sequences. We would rather want to find the best HMM among several HMMs (*e.g.* word HMMs), *i.e.* to discriminate between HMMs. Discriminative training criteria include the Maximum Mutual Information (MMI; Bahl et al. (1986); Normandin (1992)), the Minimum Phone Error (MPE, designed for speech recognition; Povey (2004)), or the Minimum Classification Error (MCE; Chou et al. (1993)). Moreover, a formulation of discriminant HMMs was proposed in (Bourlard & Wellekens, 1989).

2.5.2 Training Neural Networks

Training a neural network consists in adjusting its parameters, the connection weights, so that the model is able to perform the task at hand. As for HMMs, the goal is to optimize a criterion that reflects the quality of the network.

Given such a criterion, one may apply numerical optimization methods such as gradient descent. The backpropagation algorithm (Rumelhart et al., 1988) takes advantage from the structure of the networks to apply these methods. Algorithms were also designed to handle the temporal aspect in RNNs, such as the Backpropagation Through Time (BPTT) algorithm (Williams & Zipser, 1995).

2.5.2.1 Neural Network Training: an Optimization Problem

Provided with an input pattern, a neural network performs a series of simple computations to return an output, corresponding for example to a classification. To assess the quality of the neural network, one can compare the actual output $y(x)$ to the desired one z, and calculate a measure of error. For example, it may be the squared error $(y(\mathbf{x}) - z)^2$ for binary classification. For multi-class classification, the output is a vector $\mathbf{y}(\mathbf{x})$. The desired output may be encoded as a vector $\mathbf{z}$, with all components set to 0 except for the one corresponding to the true class, and the squared error is

$$E_{MSE}(\mathbf{x}, \mathbf{z}) = \|\mathbf{y}(\mathbf{x}) - \mathbf{z}\|^2 \tag{2.23}$$

The components of $\mathbf{y}$ are posterior probabilities. Moreover, with softmax, the outputs are positive and sum up to one, so one can use the Cross-Entropy (CE) criterion:

$$E_{CE}(\mathbf{x}, \mathbf{z}) = -\sum_{i=1}^{C} z_i \log y_i(\mathbf{x}) \tag{2.24}$$

Given a dataset $\mathcal{S} = (\mathbf{x}^{(i)}, \mathbf{z}^{(i)}), i = 1, 2, \ldots, N$ of examples labeled with the expected outputs, one can compute a global loss function

$$E(\mathcal{S}) = \frac{1}{N} \sum_{i=1}^{N} E(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}) \tag{2.25}$$

We obtain the optimal parameters $\theta = \theta_1, \ldots \theta_N$ by minimizing the error function. Numerical optimization methods, such as gradient descent, are applied to reach this minimum.

The gradient descent proceeds as follows.

1. For each training example $(\mathbf{x}^{(i)}, \mathbf{z}^{(i)})$, compute $\mathbf{y}(\mathbf{x}^{(i)})$ and $E(\mathbf{x}^{(i)}, \mathbf{z}^{(i)})$.
2. Compute the error $E(\mathcal{S})$, and its derivative with respect to the parameters $\frac{\partial E}{\partial \theta_i}$.
3. Update the parameters in the direction of the gradient $\theta_i \leftarrow \theta_i - \eta \frac{\partial E}{\partial \theta_i}$.

where η is called learning rate.

In order to compute the error, and therefore take one step in adjusting the parameters, one has to go through the whole dataset, which can lead to a slow convergence. In *stochastic* gradient descent (SGD), the parameters are updated after every training example, or every few ones, and allow to better explore and exploit the parameter space.

The momentum technique (Rumelhart et al., 1985) consists in adding a fraction of the previous update in the current one:

$$\Delta\theta(t) = -\eta \frac{\partial E}{\partial \theta} + \rho \Delta\theta(t-1)$$

The choice of learning rate may be crucial. Instead of keeping it fixed, one may change it during training. One can pre-define a schedule for learning rates, or decrease it when the objective function does not improve, or with rules such as in (Bottou, 2010), where its decrease is inversely proportional to the number of epochs, or the AdaGrad technique (Duchi et al., 2011), where the learning rate is different for each parameter, and decreases according to the sum of observed gradients for the parameter. Note that alternatives to the simple gradient descent algorithm presented here exist, such as RPROP (Riedmiller & Braun, 1993).

Finally, while we minimize the error on the training set, we also want the system to perform well on unseen examples, and avoid overfitting the training data. One manner of controlling the generalization power is to keep a validation set, separate from the training set, on which we can also compute the error. The early stopping method consists in stopping the training procedure when the error on these validation data increases.

2.5.2.2 The Backpropagation Algorithm

The neural networks presented in Section 2.3 have a layered structure. The outputs are computed from the inputs, one layer at a time. An error can be measured from the output. The backpropagation algorithm, introduced by Werbos (1974), and used in (Rumelhart et al., 1988) for MLPs, consists in calculating the gradient of the error with respect to the parameters of one layer at a time, starting from the output layer and going sequentially to the input layer, based on the following formulae:

$$\frac{\partial E}{\partial out_{i-1}} = \frac{\partial E}{\partial in_i} = \frac{\partial E}{\partial out_i} \frac{\partial out_i}{\partial in_i} \tag{2.26}$$

$$\frac{\partial E}{\partial \theta_k} = \frac{\partial E}{\partial out_i} \frac{\partial out_i}{\partial \theta_k} \tag{2.27}$$

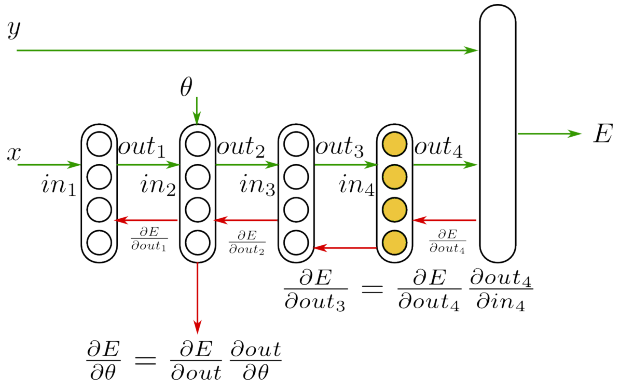


Figure 2.7: Multi-Layer Perceptron training by backpropagation of the error.

The algorithm can be applied when the connections between layers form a directed acyclic graph, and the error is propagated from the outputs to the inputs, as illustrated on Figure 2.7, hence the term *backpropagation*, and the parameter updates are computed on the way. The only requirement is to be able to compute the derivatives of the outputs of a layer with respect to its inputs.

2.5.2.3 Backpropagation Through Time

In RNNs, the inputs of recurrent layers include their outputs at the previous timestep. Thus, a sequential aspect is added to the layered structure of the network. The Backpropagation Through Time algorithm (BPTT, Werbos (1990)) consists in propagating the error both from the output to the input layer and to the previous timesteps.

When the network is unrolled in time, as shown on the right-hand side of Figure 2.4, one obtains a directed acyclic graph, on which backpropagation methods can apply. There are inputs and output layers at every timestep t, so the error for each t should be incorporated in the gradients computations, as shown on Figure 2.8.

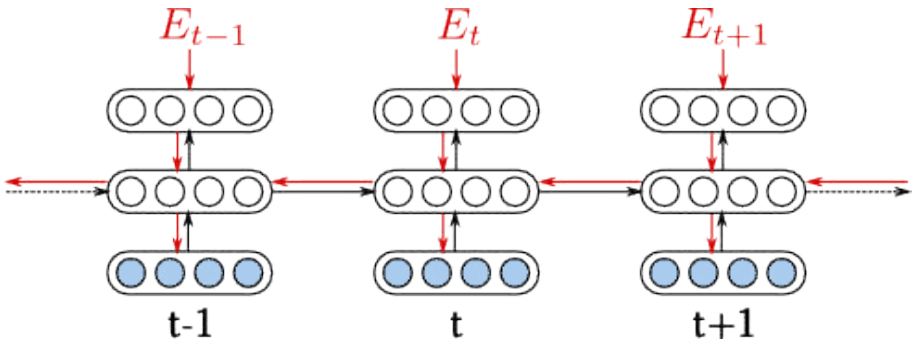


Figure 2.8: Backpropagation Through Time.

Williams & Zipser (1995) reviewed several methods to train RNNs, including BPTT and Real-Time Recurrent Learning, as well as methods to efficiently compute gradients.

2.5.2.4 Regularization

The criterion optimized in the training of neural networks is a measure of error on the training set. In the number of parameters in the model is large enough, the network can memorize the training examples, and be excellent at predicting the correct targets for examples of the training set, but perform poorly on unseen example.

The early stopping method, presented in Section 2.5.2.1, is a possible way of limiting this problem, called overfitting. We present two regularization methods which help reducing the overfitting phenomenon: weight decay, and dropout.

Weight Decay

The weight decay technique consists in adding a penalty to the cost function, which depends on the weights of the network. A common formulation is the following:

$$E^{reg}(\mathcal{S}) = E(\mathcal{S}) + \lambda \sum_{i,j,k} \left(w_{ij}^{(k)}\right)^2 \tag{2.28}$$

where the regularization term is the sum of squares of all the weights w_{ij} of all layers (k), $E(\mathcal{S})$ is the original cost function, and λ controls the relative importance given to the original cost compared to the regularization part.

The practical effect of weight decay is that the training procedure will promote solutions with small weights. It is generally observed that neural networks overfit less with that constraints, which might be explained by the fact that with small weights, the network is less sensible to small changes of the input.

Dropout

The dropout technique was recently proposed by Hinton et al. (2012) to reduce overfitting. It consists in randomly ignoring some of the units of the network during training. When dropout is applied to a hidden layer, a sample of units is dropped for each training example, with some probability. The forward pass compute the output of the network without those dropped units and corresponding connections. The backpropagation procedure is performed in this network with missing nodes. This is illustrated on Figure 2.9.

Using dropout is equivalent to train simultaneously 2^N network architectures which share weights. One architecture is randomly selected at each training step. At test time, no unit is dropped, and the whole network is used. Because a layer following dropout had less inputs during the training procedure, the weights are multiplied by $(1-p)$, where p is the dropping probability, when the whole network is used. For an MLP with a single hidden layer and a softmax output, this is equivalent to computing the geometric mean of the outputs of all 2^N possible architectures (Hinton et al., 2012).

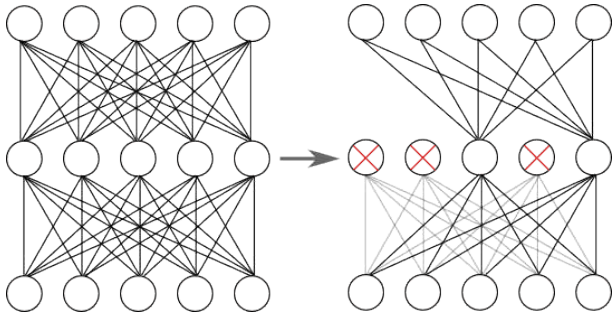


Figure 2.9: The Dropout technique.

One of the motivations of dropout is to prevent hidden units to rely on the output of others, and make them useful for classification by themselves. The underlying goal is to reduce overfitting, hence making it a form of regularization. Wager et al. (2013) have shown that dropout is equivalent to a first-order L_2 regularization after some transform of the features. Pham et al. (2014) observed that the effects of dropout were similar to those of an L_2 regularization on the classification weights.

This technique has been successfully applied to MLPs (Dahl et al., 2013), ConvNNs (Krizhevsky et al., 2012), LSTM-RNNs (Zaremba et al., 2014) and MDLSTM-RNNs (Pham et al., 2014).

2.5.3 Training Deep Neural Networks

2.5.3.1 Statement of the problem

Neural networks with many hidden layers are attractive due to their resemblance to the hierarchical organization of the brain. For example, in the visual system, successive areas process information of increasing complexity, starting from oriented edges, to actual understanding of the image. Similarly, simple features are extracted in lower layers of deep neural networks, and combined in higher layers to recognize complex structures.

Deep neural networks are very popular nowadays, in many applications of machine learning, including computer vision, speech recognition, and natural language processing. Error rates have drastically dropped with such models.

However, before sensible weight initialization techniques were proposed (Hinton et al., 2006; Bengio et al., 2007; Glorot & Bengio, 2010), it seemed difficult to train deep architectures. The successful exceptions are ConvNNs, which have a limited number of weights, shared across several positions.

Glorot & Bengio (2010) published a series of experiments designed to understand why training deep neural networks is difficult. They observe that a random initialization of weights induces an early saturation of activations in some layers. Saturated activations have small gradients, which provoke small weight updates, explaining the plateaus observed in training.

Moreover, the magnitude of gradients tends to decrease in the backpropagation procedure, and weight updates in layer that are far from the output may be too small. This is reminiscent of the *vanishing gradient* observed in standard RNNs (Bengio et al., 1994b; Hochreiter, 1998), which, in training procedures such as BPTT where the network is unfolded in time, may be interpreted as deep in time. This problem prevents RNNs to learn long-time dependencies, and deep MLPs to successfully learn dependency of the output on the input.

Methods were developed to overcome this issue. For MLPs, they deal mainly with a good initialization of weights. Hinton et al. (2006) proposed an unsupervised training method based on Restricted Boltzmann Machines to initialize the weights of Deep Belief Networks. Erhan et al. (2009) showed that this initialization places the network in a better position for the supervised training. Bengio et al. (2007) also use a greedy layer-wise pre-training, based on supervised training of each layer in turn. In (Glorot & Bengio, 2010), the authors propose a range for the uniform distribution used in random initialization of the weights, aiming at keeping activations and gradients in suitable range for effective gradient descent.

2.5.3.2 Unsupervised pre-training

In 2006, Hinton et al. (2006) proposed a method to efficiently train DNNs. The procedure consists in initializing the weights of each layer in turn. The weights are obtained by training a Restricted Boltzmann Machine (RBM). The training of RBMs is called *unsupervised* in the sense that we do not need target labels for the training data. The RBMs learn connections between observed variables and latent (hidden) ones, such that the probability of the observations, given by the model, is maximized. Thus, RBMs are generative models explaining the data. The connection weights learnt by this procedure, for each layer, are the initialization of the weights of the neural network. Then, the network can be trained in a classical supervised way (Bengio et al., 2007), as presented in the previous section.

Figure 2.10: Unsupervised layer-wise RBM pretraining.

A **Restricted Boltzmann Machine** is a generative stochastic model, made of visible and hidden binary units. It is a form of neural network with undirected connections

between visible and hidden units (and without visible-visible or hidden-hidden connections). It is an energy-based model, defined by the following energy for a given joint configuration of visible units $\mathbf{v}$ and hidden units $\mathbf{h}$:

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i=1}^{V} a_i v_i - \sum_{j=1}^{H} b_j h_j - \sum_{i=1}^{V}\sum_{j=1}^{H} w_{ij} v_i h_j \tag{2.29}$$

where $\mathbf{W} = (w_{ij})$ is the matrix of connections weights, and $\mathbf{a} = (a_i)$ and $\mathbf{b} = (b_j)$ are bias vectors respectively for the visible and hidden units.

That energy function allows us to define the joint probability $p(\mathbf{v}, \mathbf{h})$ as

$$p(\mathbf{v}, \mathbf{h}; \theta) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h}; \theta)} \tag{2.30}$$

where the normalization factor Z is the sum over all possible joint configurations. With this formulation, we can easily compute the conditional probabilities of unit activations, where the partition function Z does not appear:

$$p(v_i = 1 | \mathbf{h}; \theta) = \sigma(a_i + \sum_{j=1}^{H} w_{ij} h_j) \tag{2.31}$$

$$p(h_j = 1 | \mathbf{h}; \theta) = \sigma(b_j + \sum_{i=1}^{V} w_{ij} v_i) \tag{2.32}$$

Training With Equation 2.30, we can also compute the probability of a visible vector $\mathbf{v}$:

$$p(\mathbf{v}; \theta) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h}; \theta)} \tag{2.33}$$

with which we define a cost function for training: the negative log-probability of the training set with the model

$$E_{RBM}(\mathcal{S}) = -\sum_{\mathbf{v} \in \mathcal{S}} \log p(\mathbf{v}; \theta) \tag{2.34}$$

The derivative with respect to the parameters is (taking the notation of Hinton (2010)):

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{2.35}$$

$\langle v_i h_j \rangle_{data}$ is the expectation of v_i and h_j being active together, given by the data, which can be computed with Equation 2.32 and the training set. $\langle v_i h_j \rangle_{model}$ is the same expectation, given by the model, which should be computed with Equation 2.30. This calculation involves summing over all possible joint configurations, which is difficult to do in practice.

In (Hinton, 2002), Hinton proposes a method to train RBMs called Contrastive Divergence (CD). The underlying idea is that we can run Gibbs sampling alternatively on the hidden h_j and the visible v_i. Running the Markov chain defined by this process for an infinite number of steps, we can assume that $\langle v_i h_j \rangle_{CD_{\inf}}$ is a good replacement for $\langle v_i h_j \rangle_{model}$. In practice, running the chain for n steps is sufficient to train RBMs, and even a single step of CD works well.

The **training procedure** for the whole network is as follows (depicted on Figure 2.10). First, train an RBM with contrastive divergence, using the training observation vector as visible data. The product of gradient descent is a weight matrix $\mathbf{W_1}$, for visible-hidden connections, and biases vectors. Then, the biases for visible units are dropped, and we keep the hidden biases b_1 and the weights. The connections between visible and hidden units are oriented.

Then, the observations are forwarded through this first layer, using Equation 2.32, and the hidden units are now considered as visible units of a new RBM. Training that RBM produces a second weight matrix $\mathbf{W_2}$. After N such iterations, we have a network of N layers, with incoming weights learnt in an unsupervised manner.

Finally, on top of this network, we add the output (softmax) layer. The weights of the connection between the last pretrained layer and the output layer are randomly initialized. The obtained network is an MLP, which can be trained with the supervised methods described in Section 2.3.1.

2.5.3.3 Supervised pre-training

The method presented above initializes the weights by training generative models, *i.e.* modeling the probability of observing the actual data, in an unsupervised manner. Here, unsupervised means that we do not need labeled observations. By doing so, we learn intermediate, and possibly more and more complex hidden representations which explain or model the data.

The unsupervised part makes it possible to incorporate a virtually unlimited amount of training data. Since no labels are required, the wealth of data available today, especially through the Internet, can contribute to the creation of robust models.

The network obtained after the unsupervised pass, although being a possibly good feature extractor, is not suited to the initial classification problem, which is why supervised fine-tuning is necessary.

Another approach to weight initialization by training, patented in (Yu et al., 2011) and also explored in (Bengio et al., 2007), consists in applying supervised training methods. This approach, illustrated on Figure 2.11, also initializes the weights of one layer at a time.

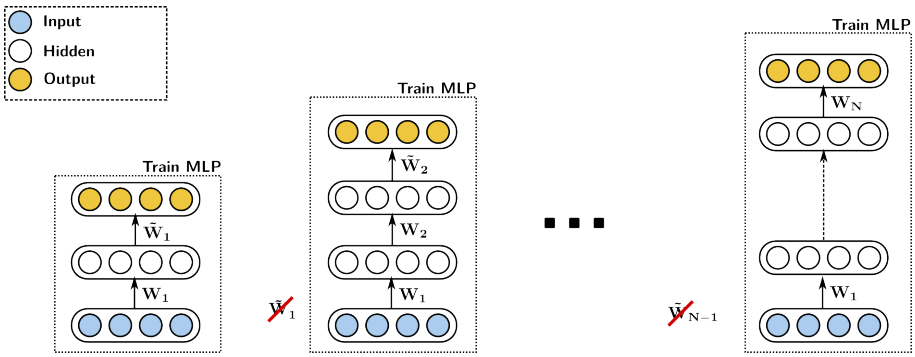


Figure 2.11: Supervised layer-wise MLP pretraining.

We first build the MLP containing only the first hidden layer and its weight matrix $\mathbf{W_1}$, and connect it directly to the output layer. We train this MLP as explained in Section 2.3.1. Thus, the weights are adjusted so that the produced layer outputs help discriminate the different classes.

Instead of waiting for training convergence of this MLP, we stop after a few iterations, *e.g.* one epoch. First, this is merely a method for weight initialization, and their values will be further adjusted in the next steps, so it is not necessary to waste too much time in this part. Moreover, in the end, we do not want the outputs of this particular layer to be the classification features, but rather to be intermediate features helping to build more complex representations in higher layers. If we wait until convergence of the network, we risk to get activations in the saturated parts of the sigmoid (or tanh) function, which will make the final training difficult and slow.

Then, we throw away the connections between the hidden and output layers, and add a second hidden layer after the first one. We keep $\mathbf{W_1}$ for the first layer, and randomly initialize a weight matrix $\mathbf{W_2}$ connecting the hidden layers. We repeat the training described above with this 3-layer MLP. Repeating this procedure N times, we get a neural network with N hidden layers, which weights have been discriminately initialized. This network is trained until convergence.

2.5.4 Training Complete Handwriting Recognition Systems

2.5.4.1 Bootstrapping

One may train the neural network as a classifier, with a labeled dataset $\mathcal{S} = \{x^{(i)}, z^{(i)}\}$ with methods presented in Section 2.5.2. For example, in the hybrid NN/HMM approach, $x^{(i)}$s are frames and $z^{(i)}$s, HMM states.

The targets may be obtained with uniform segmentation of observation sequences, or by alignment of the data using a trained system, *e.g.* GMM-HMM. One may re-align the observations with HMM states during the training procedure to refine the targets.

A similar approach for systems predicting character consists in using already segmented data. For example, one may train the network on datasets of isolated character.

The neural network is then plugged into the whole system, and its predictions provide scores for the decoding procedure.

2.5.4.2 Forward-Backward training of Hybrid NN/HMM

The bootstrapping procedure presented above assumes a prior segmentation of the input data. However, an advantage of HMMs is the possibility to train and apply them to unsegmented data. In the Baum-Welch training (Section 2.5.1), a forward-backward procedure is employed in the HMM of the true word sequence, in order to adjust the HMM parameters without making hard decisions about boundaries.

Replacing the GMM likelihoods $p(x|s)$ by the scaled NN posteriors $\frac{p(s|x)}{p(s)}$ in the HMM formulation, and in forward and backward variables, one can apply the forward-backward algorithm to obtain state posterior probabilities in the HMM, including NN

and transition probabilities:

$$\begin{aligned} \alpha_t(s)\beta_t(s) &= p(\mathbf{x}, q_t = s|\lambda) \\ \sum_s \alpha_t(s)\beta_t(s) &= \sum_s p(\mathbf{x}, q_t = s|\lambda) = p(\mathbf{x}|\lambda) \\ p(q_t = s|\mathbf{x}, \lambda) &= \frac{\alpha_t(s)\beta_t(s)}{\sum_r \alpha_t(r)\beta_t(r)} \end{aligned}$$

This training procedure, based on the forward-backward algorithm applied to HMMs or similar models, can already be found in Alpha-nets (Bridle, 1990a). Bengio et al. (1992); Haffner (1993) also propose a global training of the NN/HMM system using an MMI loss, computed with forward and backward variables. They report an improvement of results over the separate training of NN and HMMs.

Senior & Robinson (1996); Yan et al. (1997) first train a network with hard alignments, and then estimate the state posteriors with the forward-backward procedure to get a new, softer, estimate of targets, and use them for cross-entropy training of the NN.

Konig et al. (1996); Hennebert et al. (1997) focused on theoretical aspects, and explained the required assumptions for this training to achieve a global estimation of posterior probabilities. Namely, with the simplifications suggested in (Hennebert et al., 1997), they show that $p(\mathbf{W}|\mathbf{x})$ is directly optimized with this procedure.

2.5.4.3 Connectionist Temporal Classification

Connectionist Temporal Classification (CTC) was proposed by Graves et al. (2006), and corresponds to the task of labelling unsegmented data with neural networks. It is different from the previous methods, where there is one target at each timestep (or each frame in the sliding window approach).

The basic idea of this framework is that the output of the neural network, when applied to an input sequence, is directly the sequence of symbols of interest, in our case the sequence of characters. The main advantages presented in (Graves et al., 2006) are (i) that the training data does not need to be pre-segmented, *i.e.* we do not need one target for each frame to train the network, and (ii) that the output do not require any post-processing: it is already the sequence of characters, while usually neural networks predict posterior probabilities for HMM states, which should be decoded.

To make this possible, several artefacts are required. The input sequence has some length $T = |\mathbf{x}|$. Thus the length of the sequence of predictions (after the *softmax*) will also be T, while the length of the expected output sequence is generally smaller $|z| \leq T$. If no complicated post-processing is to be done, the simplest way is to have the network predict characters directly, and then look at the predictions. The output can be obtained by removing duplicates, *e.g.* $AAAABBB \longrightarrow AB$. A problem arises when two successive labels are the same, for example if we want to predict AAB. This is one of the reason why a *blank* symbol $\oslash$ is introduced in (Graves et al., 2006), corresponding to observing no label. Therefore, in the CTC framework, the network has one output for each label in an alphabet L, plus one *blank* output, *i.e.* the output alphabet is

$L' = L \cup \{\oslash\}$. A mapping $\mathcal{B} : L'^T \mapsto L^{\leq T}$ is defined, which removes duplicates, then blanks in the network prediction. For example: $\mathcal{B}(\oslash AA \oslash \oslash B) = \mathcal{B}(AAA \oslash BB) = AB$.

Provided that the network outputs for different timesteps are independent, given the input (because there is no connection from the output layer to intermediate layers (Graves et al., 2006)), the probability of a label sequence $\pi \in L'^T$ for a given $\mathbf{x}$ in terms of the RNN outputs is

$$p(\pi|\mathbf{x}) = \prod_t y^t_{\pi_t}(\mathbf{x}) \tag{2.36}$$

and the mapping $\mathcal{B}$ allows to calculate the posterior probability of a label (character) sequence $l \in L^{\leq T}$ by summing over all possible segmentations:

$$p(l|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(l)} p(\pi|\mathbf{x}) \tag{2.37}$$

With this formula, we can train the network to maximize the probability of the correct labelling of the unsegmented training data $\mathcal{S} = \{(\mathbf{x}, z), z \in L^{\leq |\mathbf{x}|}\}$ by minimizing the following cost

$$E_{CTC} = -\sum_{(\mathbf{x},z)\in\mathcal{S}} \log p(z|\mathbf{x}) \tag{2.38}$$

with gradient descent techniques.

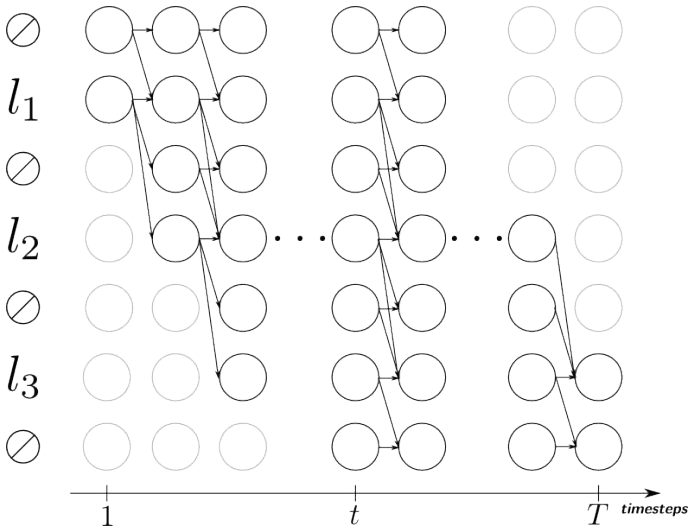


Figure 2.12: CTC graph.

The computation of $p(z|\mathbf{x})$ implies a sum over all paths in $\mathcal{B}^{-1}(z)$, each of which of length $T = |\mathbf{x}|$, which is expensive. Graves et al. propose to use the forward-backward algorithm in a graph representing all possible labelling alternatives. An example of this graph is shown on Figure 2.12. The forward (*resp.* backward) variables represent the probabilities of prefixes (*resp.* suffixes).

They are defined as follows:

$$\alpha_t(s) = p(\pi_{1:t} : \mathcal{B}(\pi_{1:t}) = l_{1:s/2}, \pi_t = l'_s|\mathbf{x}) = \sum_{\pi:\mathcal{B}(\pi_{1:t})=l_{1:s/2}} \prod_{t'=1}^{t} y^{t'}_{\pi_{t'}} \tag{2.39}$$

$$\beta_t(s) = p(\pi_{t+1:T} : \mathcal{B}(\pi_{t:T}) = l_{s/2:|l|}, \pi_t = l'_s|\mathbf{x}) = \sum_{\pi:\mathcal{B}(\pi_{t:T})=l_{s/2:|l|}} \prod_{t'=t+1}^{T} y^{t'}_{\pi_{t'}} \tag{2.40}$$

where $l \in L^{\leq T}$ is the considered label sequence and $l' \in L'^{T}$ is l, augmented with blanks, of length $|l'| = 2|l| + 1$.

With the blank being optional, and the transitions allowed by the mapping $\mathcal{B}$, we get the recurrences

$$\alpha_t(s) = \begin{cases} y^t_{l'_s} \sum_{n=0}^{1} \alpha_{t-1}(s-n), & \text{if } l'_s = \oslash \text{ or } l'_s = l'_{s-2} \\ y^t_{l'_s} \sum_{n=0}^{2} \alpha_{t-1}(s-n), & \text{otherwise} \end{cases} \tag{2.41}$$

for forward variables, and

$$\beta_t(s) = \begin{cases} y^{t+1}_{l'_{s+1}} \beta_{t+1}(s+1) + y^{t+1}_{l'_s} \beta_{t+1}(s), & \text{if } l'_s = \oslash \text{ or } l'_s = l'_{s-2} \\ y^{t+1}_{l'_{s+1}} \beta_{t+1}(s+1) + y^{t+1}_{l'_s} \beta_{t+1}(s) + y^{t+1}_{l'_{s+2}} \beta_{t+1}(s+2), & \text{otherwise} \end{cases} \tag{2.42}$$

for backward ones.

The probability of the labelling l is given by the forward variables

$$p(l|\mathbf{x}) = \alpha_T(|l'|) + \alpha_T(|l'| - 1) \tag{2.43}$$

and by the product $\alpha\beta$ at any time t

$$p(l|\mathbf{x}) = \sum_{s=1}^{|l'|} \alpha_t(s)\beta_t(s) \tag{2.44}$$

which are classical forward-backward results. The derivatives of the cost with respect to the network outputs are

$$\frac{\partial E}{\partial y^t_k} = -\frac{1}{p(z|x)} \frac{\partial p(z|x)}{\partial y^t_k} = -\frac{1}{\sum_{s'=1}^{|z'|} \alpha_t(s')\beta_t(s')} \frac{\partial \left(\sum_{s=1}^{|z'|} \alpha_t(s)\beta_t(s) \right)}{\partial y^t_k} \tag{2.45}$$

With simple calculus, we can show that

$$\frac{\partial p(z|x)}{\partial y^t_k} = \frac{1}{y^t_k} \sum_{s \in lab(z,k)} \alpha_t(s)\beta_t(s) \tag{2.46}$$

where $lab(z, k) = \{s : z'_s = k\}$. Thus we have the following derivatives with respect to the pre-softmax activations:

$$\frac{\partial E}{\partial a^t_k} = y^t_k - \sum_{s \in lab(z,k)} \frac{\alpha_t(s)\beta_t(s)}{\sum_{s'} \alpha_t(s')\beta_t(s')} \tag{2.47}$$

2.5.4.4 Graph-Transformer Networks

Bottou et al. (1997); Le Cun et al. (1997) proposed to train the neural network with backpropagation, in an integrated manner. In the Graph-Transformer Networks (GTN) approach, the character scores given by a ConvNN are included in a recognition graph. Modules transform the graph in successive steps, such as the composition with a grammar, and the extraction of the best path. The ConvNN can also be seen as a graph transformer, following a field locator and a segmenter.

Methods are proposed to back-propagate the gradients of the recognition error through the different graph-transformers (LeCun et al., 1998). The error compares the *constrained* recognition result, *i.e.* the best path for the desired character sequence, and the actual recognition result.

2.5.4.5 Sequence-Discriminative

Sequence-discriminative training optimizes criteria to increase the likelihood of the correct word sequence, while decreasing the likelihood of other sequences. This kind of training is similar to the discriminative training of GMM-HMMs with the Maximum Mutual Information (MMI) or the Minimum Phone Error (MPE) criteria.

The MMI criterion is defined as follows:

$$\mathcal{F}_{MMI}(\mathcal{S}) = \sum_{(\mathbf{x},\mathbf{W}_r)\in\mathcal{S}} \log \frac{p(\mathbf{W}_r|\mathbf{x})}{\sum_{\mathbf{W}} p(\mathbf{W}|\mathbf{x})} \tag{2.48}$$

The Minimum Phone Error (MPE, Povey (2004)) class of criteria has the following formulation:

$$\mathcal{F}_{MBR}(\mathcal{S}) = \sum_{(\mathbf{x},\mathbf{W}_r)\in\mathcal{S}} \frac{\sum_{\mathbf{W}} p(\mathbf{W}|\mathbf{x}) A(\mathbf{W},\mathbf{W}_r)}{\sum_{\mathbf{W}'} p(\mathbf{W}'|\mathbf{x})} \tag{2.49}$$

where $A(\mathbf{W}_1, \mathbf{W}_2)$ is a measure of accuracy between $\mathbf{W}_1$ and $\mathbf{W}_2$. It is the number of correct characters for MPE, or the number of correct HMM states in the recognized sequence (compared to the forced alignments) for state-level Minimum Bayes Risk (sMBR, Kingsbury (2009)).

These criteria involve a summation over all possible word sequences, which is difficult to compute in practice. Instead, recognition lattices are extracted with the optical and language models, and only word sequences in these lattices are considered in the cost function.

Sequence-discriminative training is popular in hybrid NN/HMM in speech recognition. As already mentioned earlier, Bengio et al. (1992); Haffner (1993) applied the MMI criterion to the global training of a NN/HMM system. In the past few years, these training methods arouse much interest with the advent of deep neural networks (Kingsbury, 2009; Sainath et al., 2013; Veselý et al., 2013; Su et al., 2013). Usually, a neural network is first trained with a framewise criterion. Then, lattices are generated, and the network is further trained with MMI, MPE, or sMBR. Regenerating the lattices during training may be helpful, but the gains are limited beyond the first epoch

(Veselý et al., 2013). In speech recognition, experiments with sequence-discriminative training yielded relative WER improvements in the range of 5-15%.

It should be noted that sequence-discriminative training of NN/HMM systems is also found for handwriting recognition, for example in the explicit segmentation system of Tay et al. (2001), and in (Bengio et al., 1995).

2.6 Conclusion

In this chapter, we have presented HMMs and various kinds of neural networks. We explained how they are built and trained, and showed examples of their individual and combined application to handwriting recognition. From these observations, a few questions may be asked.

Can deeper neural networks yield significant improvements over shallow ones?

Deep architectures with more than one or two hidden layers are limited to convolutional ones (ConvNNs (Le Cun et al., 1997), MDLSTM-RNNs (Graves & Schmidhuber, 2008)) for handwritten text recognition. The number of extracted features in early layers of these architectures is small, due to local receptors and weight sharing. Densely connected networks, such as MLPs (Dreuw et al., 2011a; Espana-Boquera et al., 2011) or BLSTM-RNNs (Graves et al., 2009; Kozielski et al., 2013a; Morillot et al., 2013b), have only one or two hidden layers.

The few deep MLPs found in the literature only focus on isolated character recognition (Ciresan et al., 2010; Cireşan et al., 2012) or keyword spotting (Thomas et al., 2013). On the other hand, deep MLPs are now the standard acoustic model in HMM-based speech recognition, and lead to significant reductions in error rates.

In this thesis, we will study deep architectures for MLPs in Chapter 5, and for BLSTM-RNNs in Chapter 6, and show that deeper networks can also give better results in handwriting recognition.

With deep neural networks, should we use pixel inputs rather than handcrafted features?

In speech recognition, we observe that deep neural networks perform well on less elaborated features than MFCCs (*e.g.* filterbanks). We will study that aspect by comparing the results using handcrafted features and pixel values.

Which criterion should be optimized to train neural networks for handwriting recognition?

Several procedures to train the neural networks to maximize the likelihood of the desired word sequences have been proposed in the nineties (forward-backward training, GTN, or CTC more recently). In recent publications in handwriting recognition, the

neural networks are either trained framewise with the bootstrapping procedure, or they are RNNs and trained with CTC.

On the one hand, the equation of CTC are similar to those of forward-backward training, and CTC-trained RNNs achieve very good results. Thus, we can wonder whether the CTC algorithm can be applied with more standard HMM structures rather than with the topology proposed by Graves et al. (2006), and whether CTC training of MLPs would improve their results (Chapter 7).

On the other hand, training of deep neural networks (especially MLPs), integrating lexicons and language models are common nowadays in speech recognition. More specifically, sequence-discriminative criteria, similar to those employed to train HMMs, such as MMI, MPE or state-level Minimum Bayes Risk (sMBR), generally improve the performance of neural networks. We will see that we can also achieve relative WER improvements in the range of 5-10% with deep MLPs in handwriting recognition (Chapter 5).

What are the strengths of RNNs, which are popular for handwriting recognition, and how do they compare to MLPs?

LSTM-RNNs produce excellent results for handwritten text recognition. They are involved in state-of-the-art systems (*e.g.* (Doetsch et al., 2014) for IAM database, (Pham et al., 2014) for Rimes), as well as in the winning systems of most international evaluations (ICDAR (Grosicki & El-Abed, 2011), OpenHaRT (Tong et al., 2014), Maurdor (Brunessaux et al., 2014), HTRtS (Sánchez et al., 2014)). We studied different aspects of LSTM-RNNs, in order to try to understand what makes them so good (Chapter 6), and compared them to deep MLPs (Chapter 8).

The next two chapters present the setup of our experiments, give details about the image processing, feature extraction, and language modeling, and provide preliminary GMM-HMM results.

Part II

EXPERIMENTAL SETUP

Chapter 3

Databases and Software

Contents

3.1 Introduction 93

3.2 Databases of Handwritten Text 93

3.2.1 Rimes 93

3.2.2 IAM 94

3.2.3 Bentham 95

3.3 Software 96

3.4 A Note about the Experimental Setup in the Next Chapters 97

3.1 Introduction

In this short chapter, we provide some details about the experimental setup. Namely, we start by briefly presenting the databases in Section 3.2. A more comprehensive analysis of the data, including images and language statistics, data collection, and published results, is given in Appendix A. In Section 3.3, we give an overview of the tools used to build and evaluate our systems. Finally, we introduce some notations employed to clarify the setup of the experiments explained in different part of this thesis, in Section 3.4.

3.2 Databases of Handwritten Text

We carried out the experiments on publicly available databases, in order to compare our work to other approaches and research groups (see Chapter 8). We limited the scope of this thesis to Latin script, and selected three databases. Overall, they contain two languages (French and English) and two epochs (contemporary and 19th century). The data creation/collection methods are specific to each database, and the textual content more or less controlled. In IAM (Marti & Bunke, 2002), the text to write were imposed to contributors, so the task was to copy. In Rimes (Augustin et al., 2006), only the topic of a letter was given. The last database consists of personal notes of Jeremy Bentham (Sánchez et al., 2014), making it differently constrained.

A common practice in handwriting recognition is to divide the databases in three distinct sets. The free parameters of the systems are optimized with the data of the *training set*. The *evaluation set* (or test set) contains images not seen during training, on which the performance of the system, its ability to generalize to unknown examples, is assessed. Training algorithms have some parameters themselves (the hyper-parameters), that should be set to train the systems. The *development* (or validation) set is another test set, on which we can check the best choice of values for these hyper-parameters. Most of the results presented in the thesis are on those validation sets. A selection of the best systems was made, and we evaluated them on the actual test sets in Chapter 8.

In the following, we briefly introduce the databases. More details and figures about the content of the documents (images and texts) can be found in Annex A. Please refer to the corresponding publications for precision about the data collection, digitalization and ground-truth preparation.

3.2.1 Rimes

The Rimes database (Augustin et al., 2006) consists of images of handwritten letters from simulated French mail. We followed the setup of the ICDAR 2011 competition. The available data are a training set of 1,500 paragraphs, manually extracted from the images, and an evaluation set of 100 paragraphs. We held out the last 149 paragraphs (approximately 10%) of the training set as a validation set, and trained the systems on the remaining 1,391 paragraphs. Figure 3.1 shows examples of images in Rimes database. Table 3.1 presents the number of words and characters in the different

Table 3.1: Number of pages, lines, words and characters in each dataset

Database	Set	#Pages	#Lines	#Words	(unique)	#Characters	(unique)
Rimes *(French)*	*Train*	1,391	10,203	73,822	(8,061)	460,201	(97)
	Dev.	149	1,130	8,380		51,924	
	Eval.	100	778	5,639		35,286	
IAM *(English)*	*Train*	747	6,482	55,081	(7,843)	287,727	(79)
	Dev.	116	976	8,895		43,050	
	Eval.	336	2,915	25,920		128,531	
Bentham *(English)*	*Train*	350	9,198	76,707	(12,104)	419,764	(93)
	Dev.	50	1,415	11,580		64,070	
	Eval.	33	860	7,868		40,231	

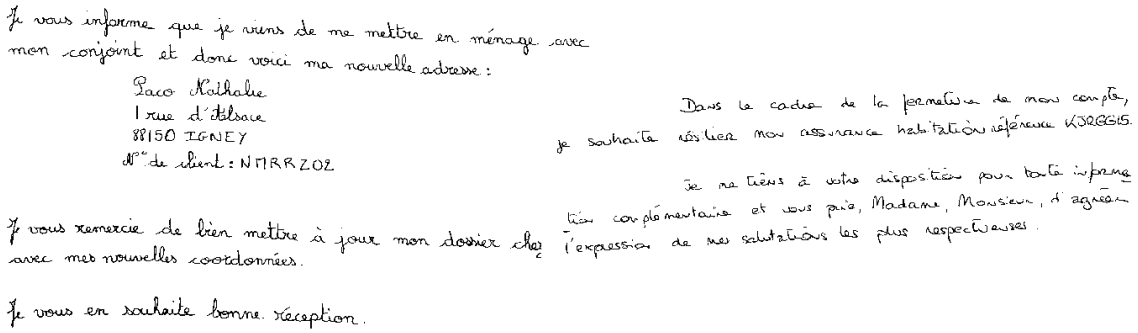


Figure 3.1: Examples from Rimes Database.

subsets. There are 460k characters distributed in more than 10k text lines, and 97 different symbols to be modeled (lowercase and capital letters, accentuated letters, digits and punctuation marks). The average character length, computed from the line widths, is 37.6 pixels at 300 DPI.

3.2.2 IAM

The IAM database (Marti & Bunke, 2002) consists of images of handwritten pages. They correspond to English texts extracted from the LOB corpus (Johansson, 1980), copied by different writers. The database is split into 747 images for training, 116 for validation, and 336 for evaluation. Note that this division is not the one presented in the official publication or on the website[1], but the one found in various publications (Bertolami & Bunke, 2008; Graves et al., 2009; Kozielski et al., 2013b). We obtained the subdivision from H. Bunke, one of the creator of the database. Figure 3.2 shows examples of images in IAM database. Table 3.1 presents the number of words and characters in the different subsets. There are almost 290k characters distributed in more than 6k text lines, and 79 different symbols to be modeled (lowercase and capital letters, digits and punctuation marks). The average character length, computed from the line widths, is 39.1 pixels at 300 DPI.

[1] http://www.iam.unibe.ch/fki/databases/iam-handwriting-database

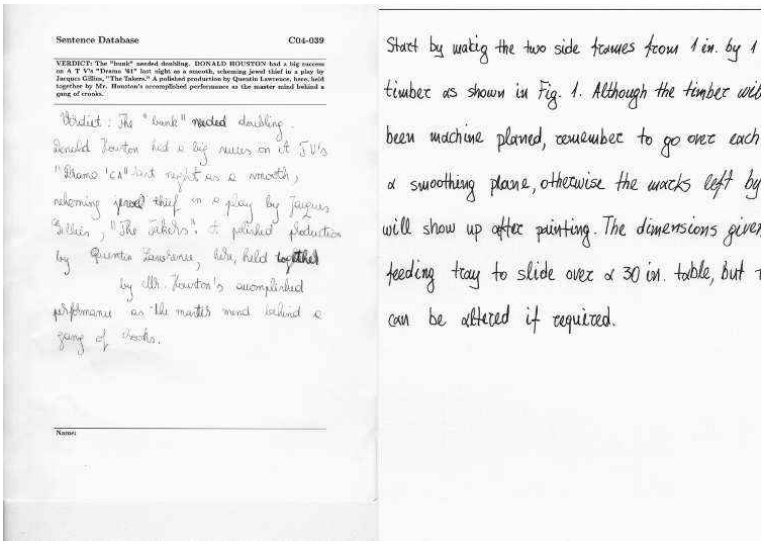


Figure 3.2: Examples from IAM Database (the text to copy corresponds to the typed header paragraph).

3.2.3 Bentham

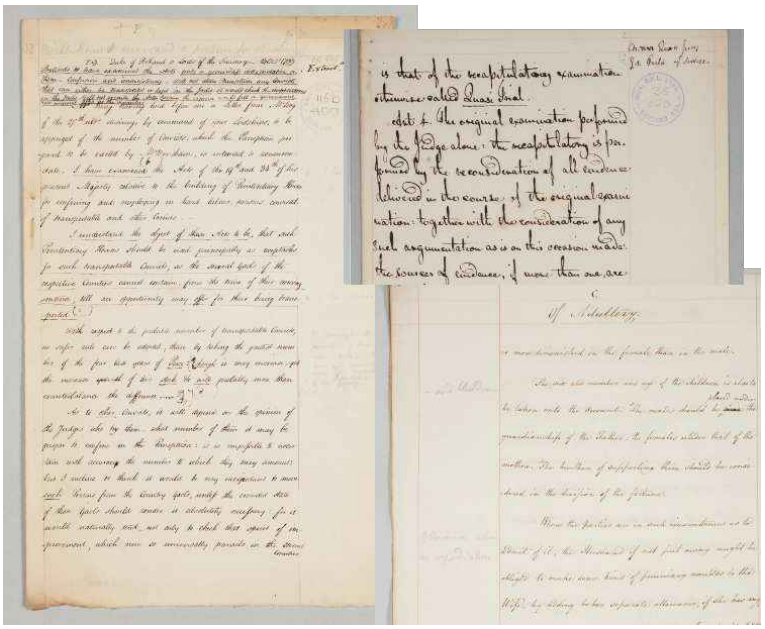


Figure 3.3: Examples from Bentham Database.

This database contains images of personal notes of the British philosopher Jeremy Bentham, written by himself and his staff in English, around the 18th and 19th centuries. The data were prepared by University College, London, during the tranScrip-

torium project[2](Sánchez et al., 2013). We followed the setup of the HTRtS competition (Sánchez et al., 2014). The training set consists of 350 pages. The validation set comprises 50 images, and the test set 33 pages. Figure 3.3 shows examples of images in the Bentham collection. Table 3.1 presents the number of words and characters in the different subsets. There are 420k characters distributed in almost 10k text lines, and 93 different symbols to be modeled (lowercase and capital letters, digits and punctuation marks). The average character length, computed from the line widths, is 32.7 pixels at 300 DPI.

3.3 Software

We used several software implementations to perform different tasks in the recognition pipeline, and to train our models. In this section, we briefly present them. Table 3.2 gives an overview of the versions and licences, as well as their application.

Table 3.2: Software used in this thesis.

Software	License	Version	Application
A2iA	Proprietary	-	image processing, feature extraction, NN training
Python Imaging Library (PIL)	©Secret Labs AB	1.1.7	image processing, feature extraction
Kaldi (Povey et al., 2011)	Apache v2	rev. 2600	HMMs, GMM-HMMs, MLP training, decoding, system combination
SRILM (Stolcke, 2002)	SRILM Research Community Licence	1.7.1	language modeling
SCTK (Fiscus, 1998)	Public Domain	2.4.7	scoring, system combination

A2iA software is a private implementation of document processing and text recognition algorithms, developed in *C++* by the company. We used some of its functionality for image processing, feature extraction, and neural network training.

Kaldi (Povey et al., 2011) is a speech recognition toolkit, written in *C++*, and released with an Apache v2 license. It is provided with many executable programs performing simple tasks, such as accumulating statistics in the Expectation step of GMM training, updating the GMMs, decoding with likelihood matrices. The modular architecture of the toolkit allows an easy integration with other programs. The code

[2] http://transcriptorium.eu/

is released with many example scripts. We used the decoders of Kaldi, as well as the programs for GMM, MLP, and HMM training, decoding graph preparation and lattice operations, such as rescoring or combination.

SRILM (Stolcke, 2002) is a language modeling toolkit, developed in *C++* in the Stanford Research Institute (SRI), released freely for research purposes, under the "SRILM Research Community License". Several executables allowed us to build language models, in the ARPA format, transformed into FSTs with Kaldi.

SCTK (Fiscus, 1998) is a scoring package released by the National Institute of Standards and Technology (NIST). We used the `sclite` program to compute the WER results. For the CERs, we did not use the `-c` option. Instead, we separated the characters and replaced the whitespace with a special symbol, in order to take them into account into the error rate. Moreover, we used the `rover` program for system combination.

The Python Imaging Library (PIL) was also used to implement some simple image processing algorithms.

3.4 A Note about the Experimental Setup in the Next Chapters

In the following chapters, we will study several aspects of deep neural networks, along with training techniques. The results presented in different sections, tables and figures might not always be comparable to each other, due to different setups.

Sometimes, a given aspect will be the focus of the experiment, and other parts will be fixed. When relevant and possible, the result tables and figures will be accompanied with the experimental setup regarding the following aspects:

- The input features of the systems: handcrafted features (**Feat.**) or pixel values (**Pix.**).
- The depth of neural networks: we focus on deep (**Deep**) networks, but sometimes, we will use shallow (**Shallow**) architectures, for rapid experimentation.
- For RNNs, we sometimes applied the Dropout (**Drop.**) regularization technique (Hinton et al., 2012; Pham et al., 2014).
- The training procedure of neural networks optimizes a framewise criterion (**Xent**), the CTC criterion (**CTC**), or a sequence-discriminative one (**Seq.**).

So, for example, the mention **Feat.**/**Deep**/**Drop.** in a result table means that all error rates presented are achieved with the handcrafted features, and a deep architecture with dropout. The experiment in this case might be a comparison of different training criterion.

Chapter 4

Baseline System

Contents

4.1 Introduction 101
4.2 Preprocessing and Feature Extraction 102
4.2.1 Image Preprocessing 102
4.2.2 Feature Extraction with Sliding Windows 103
4.2.2.1 Handcrafted Features 103
4.2.2.2 Pixel Values 104
4.3 Language Models 105
4.3.1 Corpus Preparation and Vocabulary Selection 105
4.3.2 Language Models Estimation 106
4.3.3 Recognition Output Normalization 107
4.4 Decoding Method 108
4.5 A GMM/HMM baseline system 109
4.5.1 HMM topology selection 109
4.5.2 GMM/HMM training 109
4.5.3 Results 110
4.6 Conclusion 112

4.1 Introduction

In the next chapters, we present many experiments, carried out on three public databases. In this thesis, we focus on the optical model of HMMs for handwriting recognition. More particularly, we study two kinds of deep neural networks in the hybrid NN/HMM framework presented in the Chapter 2. The inputs of our systems are sequences of feature vectors extracted from preprocessed line images. The outputs are posterior probabilities of HMM states.

We explored different aspects of the neural networks: their structure, their parameters, their training procedure. We present results of the whole recognition systems, *i.e.* with raw image inputs, and word sequence outputs. Most of the components of these systems, excluding the neural networks, are kept fixed throughout the experiments, unless stated otherwise.

This chapter is dedicated to the presentation of these fixed components:

- The text line image preprocessing
- The extraction of features
- The modeling with Hidden Markov Models (HMMs)
- The language models, and output normalizations

Figure 4.1 shows an overview of the recognition system and of its components. The blue ones are the fixed ones, presented in this chapter. In the last section, we also present a baseline optical model – a Gaussian Mixture Model.

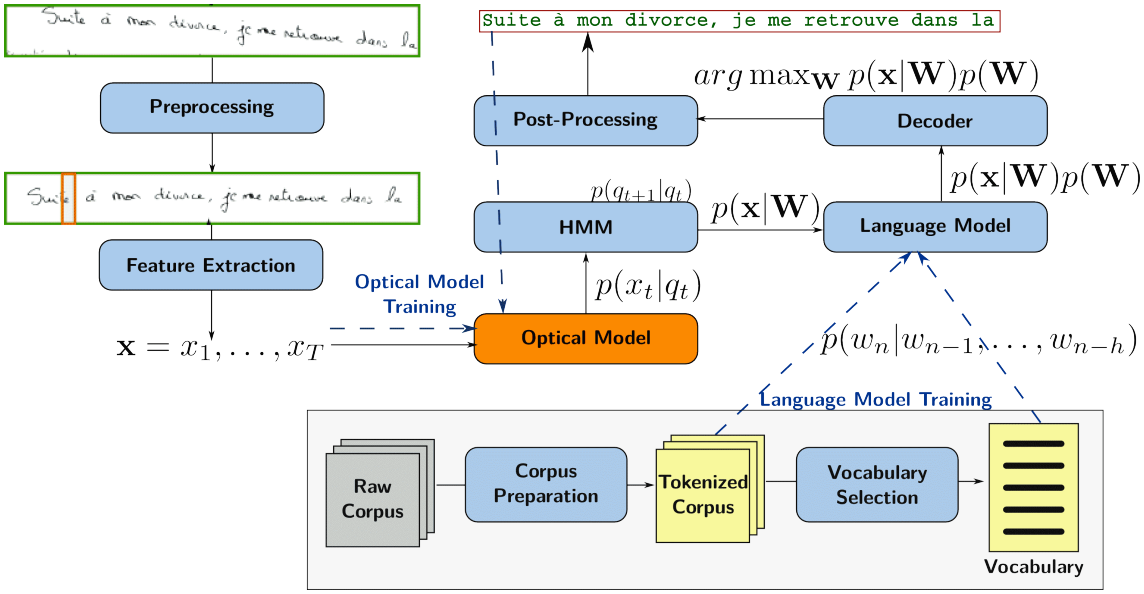


Figure 4.1: Overview of the recognition system.

In this chapter, we introduce the components, and justify the choice of parameters. In Section 4.2, we describe the image preprocessing and the extraction of two types of features – handcrafted features and pixel values – with sliding windows.

Language models are detailed in Section 4.3.

In Section 4.4, we present the decoding method, using the Kaldi toolkit (Povey et al., 2011), based on Finite-State Transducers.

In the last Section (4.5), we evaluate the quality of the choices of preprocessing, feature extraction and language models by training GMM/HMMs. These systems will also serve as baselines for the rest of the work, and provide forced alignments to train neural networks. We show that we obtain reasonable results, comparable to the best published ones for GMM/HMMs. Moreover, the comparison with the best systems in general, which involve Recurrent Neural Networks, proves that these discriminative models are more powerful for handwriting recognition, and justify the need to explore them.

4.2 Preprocessing and Feature Extraction

In this section, we present the image preprocessing applied and the features extracted. We experimented several options. The number of all different combinations was too large to try them all. The quick experiments consisted in training GMM/HMM systems with 10% of the training set, for only a few EM iterations and Gaussians per state, and recording the WER on 10% of the validation set, with a very small closed vocabulary (around 500 words without OOVs) and a unigram language model. We used the handcrafted features described in the second part of this section, exctracted with a sliding window. We present results on IAM, where we tried most of the configurations. We also tuned Rimes and Bentham systems, starting with setups that were good for IAM. High WERs are caused by the limited amount of data (image and vocabulary) and order of language model, but the selected methods produced reasonable GMM/HMM baselines in the end (cf. Section 4.5) and state-of-the-art final systems (cf. Chapter 8).

4.2.1 Image Preprocessing

First the potential skew in the image is corrected with the algorithm of Bloomberg et al. (1995). We applied the slant correction method presented in (Buse et al., 1997). For contrast enhancement, we tried adaptive thresholding (Otsu, 1979), and the interpolation method from (Roeder, 2009). The following table shows that the latter gives better WERs, with two choices of sliding window widths.

	Window size:	**6px**	**9px**
Method	None	54.2%	58.0%
	Adaptive	57.2%	58.5%
	Interpolation	**53.1%**	57.2%

We also tried to normalize the height of images, either to a fixed value of 72px or with region-dependent methods (Toselli et al., 2004; Pesch et al., 2012), with fixed

height for each region (ascenders, core, descenders – 22, 33 and 17px, or 24px for each). The regions are found after deskew and deslant with the algorithm of Vinciarelli & Luettin (2001). We selected the normalization of each region to 24px, based on the following results (WER):

	Window size:	6px	9px
Method	None	56.9%	59.6%
	Fixed (72px)	54.2%	58.7%
	Region (22px, 33px, 17px)	58.7%	63.8%
	Region (24px, 24px, 24px)	**53.1%**	57.2%

4.2.2 Feature Extraction with Sliding Windows

We used two kinds of features: **handcrafted features**, and raw **pixel intensities**. A sliding window is scanned through the line image to extract features.

4.2.2.1 Handcrafted Features

The handcrafted features are geometrical and statistical features extracted from the window. They were proposed by Bianne-Bernard (2011); Bianne et al. (2011), and derived from the work of El-Hajj et al. (2005). They gave good performance on several public databases (Menasri et al., 2012; Bianne et al., 2011).

The text baseline and core region are computed with the algorithm of (Vinciarelli & Luettin, 2001), and the following values are calculated:

- 3 pixel density measures: in the whole window, and in the regions above and below the baseline,
- pixel densities in each column of pixels (w_f values, where w_f is the width of the sliding window,
- 2 measures of the center of gravity: relative vertical positions with respect to the baseline and to the center of gravity in the previous window,
- 12 measures (normalized counts) of local pixel configurations: six configurations, computed from the whole window and from the core region,
- histogram of gradients (HoG) in 8 directions.

All these features form a $(25+w_f)$-dimensional vector, to which deltas are appended, resulting in feature vectors of dimension 56 ($w_f = 3$px). The parameters of the sliding window (width and shift) for the handcrafted features have been tuned using the same method as for preprocessing. The result is presented on Figure 4.2. Each shift value corresponds to a different subplot, and each width to a different line color. We tested different number of HMM states per character. The best parameters were a shift and width of 3px each (no overlap between windows). We ran a full training of a selection of the best systems, which confirmed the superiority of the selected parameters.

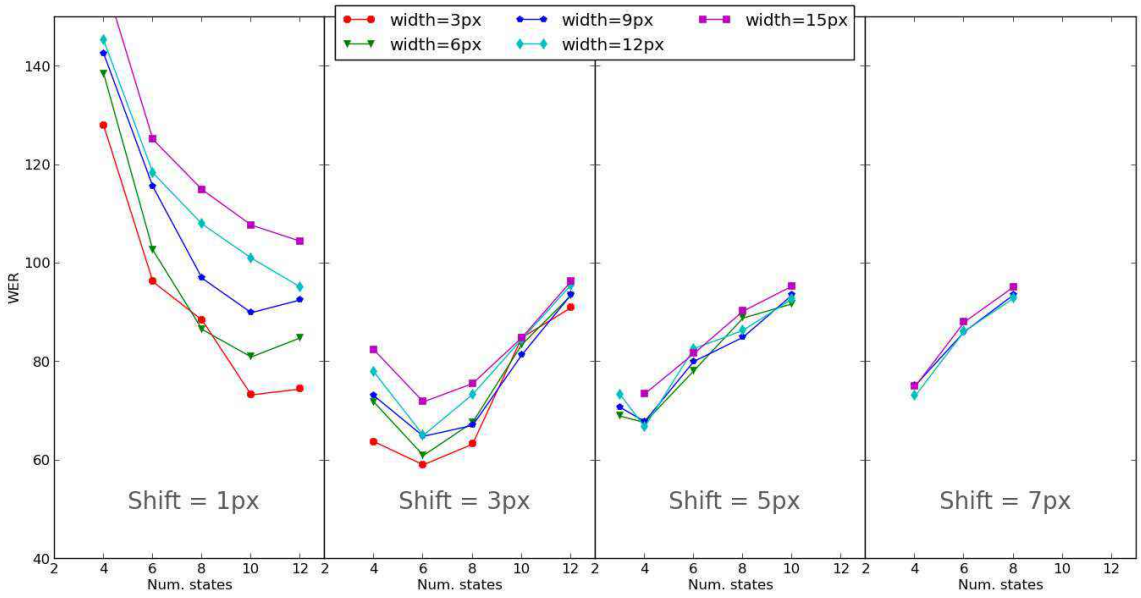


Figure 4.2: Optimization of the sliding window parameters (IAM)

For neural networks, the features are normalized to have zero mean and unit variance. We compute the mean and standard deviation along each dimension on the training set. A feature vector for neural networks is obtained by subtracting the mean and dividing by the standard deviation in each dimension.

4.2.2.2 Pixel Values

The "pixel features" are extracted with a sliding window. The width of the window was optimized for deep neural networks: 45px for Rimes and IAM, 57px for Bentham. We also tried variations of these values in specific experiments. In order to extract the same number of frames for both kinds of features (so we can keep the same HMM topologies, use GMM/HMM forced alignments for both input types, and get comparable results), the shift was fixed to be the same as for handcrafted features. Moreover, we padded the images with $(w_p - w_f)/2$ columns of white pixels on the left and right borders, where w_p (*resp.* w_f) is the the width of the window for pixel values (*resp.* handcrafted features). This is illustrated on Figure 4.3.

To limit the number of features, each frame is downscaled from a height of 72px to a height of 32px. The aspect ratio is kept constant (20x32px for Rimes and IAM, 25x32px for Bentham). This is done after the window extraction in order to keep the integer shift fixed equal to 3px. The pixels intensities are transformed with the following formula:

$$x \mapsto \frac{255 - x}{255}$$

to lie in the interval $[0, 1]$ (1 corresponding to black pixels). The frame is flattened to obtain the 640-dimensional feature vectors (800-dimensional for Bentham). No Principal Component Analysis or other decorrelation or dimensionality reduction algorithm

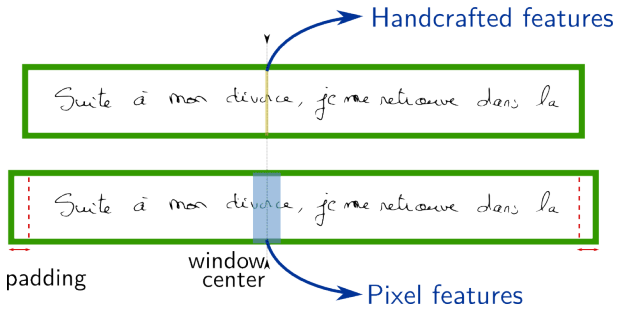


Figure 4.3: Extraction of corresponding handcrafted and pixel features with two different sliding windows.

was applied.

These features are only used in neural networks, and are also normalized to zero mean and unit variance. The difference with handcrafted features is that we do not normalize along each dimension separately, but compute the global pixel mean and standard deviation, independently of the pixel location.

4.3 Language Models

We trained language models for each database with the SRILM toolkit (Stolcke, 2002). First, we selected a vocabulary and prepare the corpus for LM training. The tokenization and normalizations applied to the training corpora, as well as the vocabularies are presented in the first part of this section. The output of the recognition system is a sequence of tokens from the vocabulary. After giving details about LM training, we present the post-processing applied to the transcriptions.

4.3.1 Corpus Preparation and Vocabulary Selection

Each database has specificities. For example, Rimes contains many reference codes and acronyms. Hyphenation appears a lot in Bentham database. We applied different tokenizations to take them into account and limit the size of the vocabularies.

For IAM, the language model is trained on the LOB, Wellington and Brown corpora. Since IAM consists of texts from the LOB corpus, the passages corresponding to the validation and test sets were removed from the corpus. The punctuation symbols were separated from words, with a special treatment of contractions, for which the apostrophe is assigned to the right part. We kept the 50,000 most frequent words in the corpus in the vocabulary.

For Rimes, we also split the words made only of digits and/or capital letters, so that the reference codes, likely to appear only once, are not included in the language model. The LM was trained on the tokenized training set paragraph annotations, with a vocabulary including all tokens (5,000).

For Bentham, we first extracted complete paragraphs of text from the line annotations. There were lines with a single word, which did not always fit into the sentence. When we re-built the paragraphs, we ignored these, but kept them for unigram counts. The hyphenation does not make sense either at paragraph level, so we reconstructed whole words from hyphenated ones. Hyphenation is signaled at the end and beginning of lines by three different symbols (`:`, `-`, and `=`). When we found a word ending with an hyphenation symbol, followed by a word beginning with one, we reassembled the word. If we found beginnings but no ending (or the opposite), we droped the sentence.

To limit the size of the vocabulary, we also isolated currency symbols, and split sequences of digits and capital letters.

A vocabulary of 7,318 words is extracted from this corpus. In order to recognize hyphenated words, we added hyphenated versions in the vocabulary. For all words with more than ten occurrences, we generated all possible (beginning, end) pairs using Pyphen[1]. For example, "`hyphenation`" produces `('hyphen', 'ation')` and `('hy', 'phenation')`. We added the three possible hyphenation symbols at the end (*resp.* beginning) of words beginnings (*resp.* endings), and included them in the vocabulary. It increased the size to 32,692 words, but decreased the Out-Of-Vocabulary (OOV) rate on the validation set from 7.1 to 5.6%.

4.3.2 Language Models Estimation

Note: the perplexities presented in this section do not take into account the out-of-vocabulary words.

IAM — We used a 3-gram language model trained on the tokenized LOB, Brown and Wellington corpora, with modified Kneser-Ney smoothing. The resulting model has a perplexity of 298 and an OOV rate of 4.3% on the validation set (respectively 329 and 3.7% on the evaluation set). This LM was trained by the handwriting recognition team at RWTH University in Aachen, and kindly provided to us by P. Doetsch, M. Kozielski and their colleagues.

Rimes — We trained *n*grams of different orders, with different discounting methods, and measured the perplexity on the normalized validation set annotations. The results are summarized on Table 4.1. We notice that Kneser-Ney discounting produce better language models. The improvement from 4 to 5gram was small, so we built a 4gram LM with Kneser-Ney discounting (Kneser & Ney, 1995) as a tradeoff between size and perplexity. The language model has a perplexity of 18 and OOV rate of 2.9% on the validation set (respectively 18 and 2.6% on the evaluation set).

Bentham — We estimate the LMs with the *n*gram counts from the corpus. The hyphenated word chunks are added to the unigrams with count 1. We generated 4grams with Kneser-Ney discounting. Table 4.2 presents the perplexities of different *n*grams. They are better without hyphenation, but we found that the hyphenated version gave better recognition results.

[1] http://pyphen.org

Table 4.1: Perplexities of Rimes LM with different discounting methods and $n$gram orders, on the validation set.

Discounting	1gram	2gram	3gram	4gram	5gram
Natural	282.1	31.1	21.9	20.4	20.2
Witten & Bell (1991)	288.0	30.7	21.0	19.4	19.4
Kneser & Ney (1995)	283.4	29.2	19.6	**18.0**	**17.9**

Table 4.2: Perplexities of Bentham LMs with different $n$gram orders and hyphenation, on the validation set.

Hyphenation	**Size**	**OOV%**	1gram	2gram	3gram	4gram
No	7,318	7.1	348.7	129.4	**101.7**	**96.7**
Yes	32,692	5.6	656.1	137.6	108.4	**103.1**

4.3.3 Recognition Output Normalization

We evaluated the system by comparing the recognition outputs to the ground-truth transcriptions. We compute the WER and CER. Different databases have different annotation conventions. For example:

- In Rimes, exclamation and question marks are separated from words most of the time. Full stops, commas, quotes, currencies, parentheses sometimes are, but not always. Contractions are mostly not.
- In IAM, the punctuation is separated from words, except for apostrophe: when it is a single quote mark, it is separated; when a possessive mark, it is not (*e.g.* `the party's supporters`, `the neutrals' conference`). Verb contractions are also separated (*e.g.* `he 's`), but negation contractions are not (*e.g.* `don't`).
- In Bentham, punctuation symbols are separated, even in contractions and possessives (*e.g.* `it ' s`), except when involved in hyphenation.

Our systems model whitespaces, but they are optional during decoding. The output is a sequence of LM tokens (words). In the initial transcription, we insert spaces between each item in the sequence. We transform the transcriptions obtained with our systems so they match the database conventions rather than the tokenization chosen for LMs. The transformations are simple rule-based ones, mainly based on regular expressions. Since we do not apply a complicated post-processing, it is not always possible to retrieve the right format (*e.g.* for the "`'s`" in IAM). The following post-processing is done:

Rimes — We remove spaces in sequences of digits/capital letters. We apply the rules for punctuation (*e.g.* full spaces, commas, parentheses are not separated from words), and remove the spaces around apostrophes.

IAM — We stick `'s` to any word which is not in a list of pronouns (*he, she, it, what, who, where, how, that, there*), and `'t` to the preceding word.

Bentham — We remove spaces in sequences of digits/capital letters.

4.4 Decoding Method

We decoded with the tools implemented in the Kaldi speech recognition toolkit (Povey et al., 2011), which consist in a beam search in an FST, with a token passing algorithm. The FST is the composition of the representation of each components (HMM, vocabulary and Language Model) as FSTs (H, L, G). The HMM and vocabulary conversion into FST is straightforward. The LM generated by SRILM is transformed by Kaldi. The final graph computation not only involves composition, but also other FST operations. The method proposed in Kaldi is a variation of the technique explained in (Mohri et al., 2002):

$$F = min(rm(det(H \circ min(det(L \circ G)))))$$

where $\circ$ denotes FST composition, and min, det and rm are respectively FST minimization, determination, and removal of some ϵ-transitions and potential disambiguation symbols. Refer to (Mohri et al., 2002; Povey et al., 2011) for more details concerning the FST creation.

The decoding start at initial states, and iteratively reads frame scores to advance to next states, following arcs. Tokens are propagated, computing the following score:

$$s(q_t) = s(q_{t-1}) + \alpha \times l(x_t, q_t) + \omega(q_{t-1} \rightarrow q_t) \tag{4.1}$$

where $\omega(q_{t-1} \rightarrow q_t)$ is the graph cost for transiting from q_{t-1} to q_t, including the LM and HMM transition negative log-probabilities, α is the *optical scale* parameter, tuned on the validation set. $l(x_t, q_t)$ is the score provided by the optical model. With GMMs, it is the negative log-likelihood of x_t for the GMM of q_t. For hybrid NN/HMM systems, as explained in the previous chapter:

$$l(x_t, q_t) = -\log \frac{p(q_t|x_t)}{p(q_t)^\kappa} \tag{4.2}$$

where $p(q_t|x_t)$ is the state posterior probability computed by the neural network, and $p(q_t)$ is the state prior probability, estimated from the training set. κ is the *prior scale* parameter, tuned on the validation set.

For each state, we keep only the token with the best score (Viterbi decoding), and delete all tokens with a score worse than the current best minus a beam value (beam search).

Lattices are generally extracted (Povey et al., 2012) and rescored with different optical scales and word insertion penalties. The word insertion penalty (WIP) is included by adding a constant value to all word arcs in the lattice FSTs. The final values are chosen according to the validation set results.

The final transcription is the sequence of output symbols (words) in the best path of the lattice, normalized as explained in the previous section. This decoding procedure is the same for GMM/HMMs and for all hybrid systems presented in this thesis.

4.5 A GMM/HMM baseline system

4.5.1 HMM topology selection

We chose a left-right HMM topology for all the characters. Each state has one transition to itself, and one to the next state. The whitespace is modeled by a 2-state HMM. All other character HMMs have the same number of states, tuned along with the sliding window topology. On Figure 4.2, we observe a correlation between the shift and the optimal number of states. This is not surprising, since the shift controls the number of frames extracted, hence the average duration of characters in terms of number of observation vectors. Each state if associated with its own emission probability. We built:

- 96 character HMMs with 5 states for Rimes
- 78 character HMMs with 6 states for IAM
- 92 character HMMs with 6 states for Bentham

4.5.2 GMM/HMM training

We trained GMM/HMMs with Kaldi (Povey et al., 2011), using the handcrafted features, the Maximum Likelihood criterion and the EM procedure described in Chapter 2. At each iteration, the Viterbi alignments of the training set are computed. We add as many Gaussians in the mixtures as there are HMMs states, and weights, means and diagonal covariances are updated. Transition probabilities are also estimated from the alignments.

Every five iterations, we assess the performance on the validation set. When no improvement is observed for more than 15 iterations, the training stops, and the best model is kept. The GMM/HMMs have not been discriminately trained. Finally, we have:

- 33,630 diagonal-covariance Gaussians in 484 mixtures for Rimes
- 40,849 diagonal-covariance Gaussians in 472 mixtures for IAM
- 67,037 diagonal-covariance Gaussians in 556 mixtures for Bentham

Table 4.3: Context dependency in GMM/HMM (results on lines of IAM validation set)

Model	WER	CER
Context-independent	16.2	6.9
Context-dependent	16.3	6.6

On IAM, we tried to build context-dependent models, where the HMM state emission probability density function (PDF) also depends on the previous and next character. PDFs are shared across several states, and determining which one to use in a given context is done with a Clustering and Regression Tree (CART). We followed the data-driven approach implemented in Kaldi (Povey et al., 2011). From Viterbi alignments, questions are generated with hierarchical clustering of Gaussians, and the CART is built by iteratively splitting nodes to maximize the likelihood. The algorithm has hyper-parameters such as the maximum number of leaves, the minimum occupancy of leaves, or a likelihood improvement threshold. In the best setup we found, the improvement over the context-independent models was not significant, as illustrated in Table 4.3. Therefore, we only used context-independent GMM/HMMs.

4.5.3 Results

The goals of these GMM/HMMs are: (*i*) to check that we chose a good preprocessing, feature extraction, and HMM topology, (*ii*) to serve as baseline to compare hybrid models with, and (*iii*) to produce forced alignments to build training sets for neural networks. Therefore, it is important to yield good results with these models to validate the choices and to make sure that the baseline is good. Moreover, good models will certainly produce good alignments, *i.e.* a good ground-truth of the training sets.

We recorded the results of the recognition of validation and test sets of each database. For Rimes and IAM, we know the order of lines in paragraphs, and we applied the language models at paragraph level, which both makes more sense and gives better results (see Table 4.4).

Table 4.4: Applying LMs to text lines and paragraphs (results on IAM validation set)

LM scope	WER	CER
Lines	16.2	6.9
Paragraphs	15.2	6.3

The results are presented on Tables 4.5 (Rimes), 4.6 (IAM), and 4.7 (Bentham). We compare them to the best published results with pure GMM/HMMs on the one hand, and with other systems on the other hand.

On Rimes (Table 4.5), some publications do not report results on the development set, while others (such as (Kozielski et al., 2014a)) were directly tuned on the evaluation

set. Yet our GMM/HMM system performs well, achieving WER and CER competitive with the GMM/HMM of Kozielski et al. (2014a).

Table 4.5: Results on Rimes database

	Dev.		Eval.	
	WER	**CER**	**WER**	**CER**
Our GMM/HMM	17.2	5.9	15.8	6.0
GMM/HMM systems				
Kozielski et al. (2014a)	-	-	15.7	5.5
Grosicki & El-Abed (2011)	-	-	31.2	18.0
Other systems				
Pham et al. (2014)	-	-	12.3	3.3
Doetsch et al. (2014)	-	-	12.9	4.3
Messina & Kermorvant (2014)	-	-	13.3	-

The results are also reasonable on IAM (Table 4.6). The first line of the comparison (Kozielski et al., 2013b) uses an open-vocabulary approach, able to recognize any word (no OOV). The corresponding closed vocabulary result with a 20k word lexicon is displayed on the third line.

Table 4.6: Results on IAM database

	Dev.		Eval.	
	WER	**CER**	**WER**	**CER**
Our GMM/HMM	15.2	6.3	19.6	9.0
GMM/HMM systems				
Kozielski et al. (2013b)	12.4	5.1	17.3	8.2
Kozielski et al. (2014a)	12.6	4.7	-	-
Kozielski et al. (2013b)	18.7	8.2	22.2	11.1
Toselli et al. (2010)	-	-	25.8	-
Bertolami & Bunke (2008)	26.8	-	32.8	-
Other systems				
Doetsch et al. (2014)	8.4	2.5	12.2	4.7
Kozielski et al. (2013a)	9.5	2.7	13.3	5.1
Pham et al. (2014)	11.2	3.7	13.6	5.1

On Bentham database (Table 4.7), the only available GMM/HMM reference is given on the validation set. The system of Gatos et al. (2013) corresponds to the baseline provided by the organizers of the HTRtS competition[2].

Finally, the "*other systems*" are much better. They all involve recurrent neural networks, either as a feature extractor or as an optical model. In the following chapters,

[2]The organizers of the competition would like to point out that this baseline was provided to the contestants with the aim of encouraging participation. Its performance is therefore not that of a true baseline and can be easily beaten. Unofficial (unpublished) results obtained by improving this system, *e.g.* with discriminative training, are roughly 18% WER with GMM/HMMs, according to the organizers.

Table 4.7: Results on Bentham database

	Dev.	
	WER	**CER**
Our GMM/HMM	27.9	14.5
GMM/HMM systems		
Gatos et al. (2013)	32.6	-

we build such discriminative models: recurrent neural networks, but also deep multi-layer perceptrons. In the last chapter, we present comparable results to those displayed here.

4.6 Conclusion

In this chapter, we have introduced the baseline systems for the rest of the thesis. We will give our results on three different public databases: Rimes, IAM and Bentham. We have presented the decoding method and the choices of components which will remain fixed throughout the experiments, namely:

- the **preprocessing**, consisting of deskewing and deslanting the text lines, enhancing the contrast with an interpolation method, and normalizing the height of the image to a fixed value for different regions
- the **extraction of observation sequences** with sliding windows, with two kinds of features: handcrafted features, and pixel values
- the selection of **vocabularies** and training of n-gram **language models**

Finally, we validated our choices by training GMM/HMMs. We obtained results that are comparable to other published GMM/HMM systems. These models will serve as baselines for the next experiments.

Part III

DEEP NEURAL NETWORKS IN HIDDEN MARKOV MODEL SYSYEMS

Chapter 5

Hybrid Deep Multi-Layer Perceptrons / HMM for Handwriting Recognition

Contents

5.1 Introduction 117
5.2 Experimental Setup 118
5.3 Study of the Influence of Input Context 119
5.3.1 Alignments from GMM/HMM Systems 119
5.3.2 Handcrafted Features 121
5.3.3 Pixel Intensities 123
5.4 Study of the Impact of Depth in MLPs 124
5.4.1 Deep MLPs 124
5.4.2 Deep vs Wide MLPs 126
5.5 Study of the Benefits of Sequence-Discriminative Training . 128
5.6 Study of the Choice of Inputs 130
5.7 Conclusion 131

5.1 Introduction

Multi-Layer Perceptrons (MLPs) have been used in handwriting recognition systems with explicit segmentation (Knerr et al., 1998), and in HMM-based systems in the hybrid NN/HMM framework (Espana-Boquera et al., 2011), or to extract discriminant features for GMM modeling (Dreuw et al., 2011a). However, the architecture proposed in these papers are shallow, limited to one or two hidden layers. The application of deeper MLPs is found in handwriting recognition, but only for isolated digit or character recognition (Ciresan et al., 2010; Cireşan et al., 2012), or keyword spotting (Thomas et al., 2013).

In HMM-based speech recognition, major improvements were reported with deep MLPs in the last decade. They are now a standard component of state-of-the-art speech recognition systems. Depth seems to contribute to better modeling (Mohamed et al., 2012) and robustness (Deng et al., 2013). Sequence-discriminative training of MLPs is also commonly applied in speech recognition systems, bringing significant improvements over the framewise-trained networks with cross-entropy.

Finally, deep neural networks are known to build internal representations of increasing complexity of their raw inputs. Therefore, one may not have to design or implement the extraction of relevant features from the image, and feed instead the pixels directly to the network. Deep MLPs achieve good results in speech recognition even with less elaborated inputs than MFCC (Deng et al., 2013). In computer vision, the pixel values are used as the network inputs.

In this chapter, we investigate deep MLPs for handwriting recognition in hybrid NN/HMM systems. We are interested in answering the following questions:

- *Is it still important to design handcrafted features when using deep MLPs, or are pixel values sufficient?*
- *Can deep MLPs give rise to big improvements over MLPs with one hidden layer for handwriting recognition?*
- *Do we observe the same improvements with sequence-discriminative training of these systems as one does in speech recognition?*

First, in Section 5.2, we introduce the experimental setup followed in this chapter. We give some details about the MLP architecture and training procedure.

In Section 5.3, we study how the input context, *i.e.* the size of the sliding window or the number of considered frames, influences the performance of the neural network. We show that choosing the right amount of context is crucial, and has a significant impact on the results.

Then, in Section 5.4, we experiment different depths of MLPs. We show that MLPs with more than one hidden layer yield better Word Error Rates (WERs) than shallow MLPs, especially when the inputs are pixel values.

In Section 5.5, we apply a sequence-discriminative training criterion, the state-level Minimum Bayes Risk, to further improve the results of the deep MLPs. We

report relative WER improvements which are consistent with those observed in speech recognition.

Finally, in Section 5.6, we compare the results obtained with handcrafted features and with pixel values. The final WERs are similar for both types of inputs.

We conclude this Chapter in Section 5.7 with a discussion of the important aspects of deep MLPs.

5.2 Experimental Setup

The experiments are carried out on three public databases of handwritten text lines: Rimes, IAM and Bentham, with handcrafted features and pixel values.

To train the deep MLPs, we performed the forced alignments of the training set with the GMM/HMMs presented in Section 4.5, to have a target HMM state for each input observation. We held out 10% of the datasets for validation and early stopping. Overall, the datasets contain:

- 5.6M training examples (frames) for Rimes
- 3.8M training examples (frames) for IAM
- 4.3M training examples (frames) for Bentham

The networks presented in this chapter were trained with the Kaldi toolkit, and the GPU implementation, following the procedure described in Section 2.5.3.2. The weights of the first layer are initialized by contrastive divergence training of a Gaussian-Bernouilli RBM, for two epochs, with a learning rate of 0.01. Weights of higher layers are initialized by contrastive divergence training of Bernouilli-Bernouilli RBMs for one epoch, with a learning rate of 0.4. A penalty on the L_2-norm of the weights is added to the cost function with a factor 0.0002.

Once the weights initialized, the MLP with sigmoid hidden units is trained with the cross-entropy criterion and the labeled dataset. At each epoch, the cost on the validation set is computed. When the relative improvement is below 0.01%, the learning rate is halved. When the cost starts increasing, the training procedure stops, and the best MLP is kept.

The MLPs have a softmax output layer, and each output corresponds to an HMM state in the models presented in Section 4.5, that is:

- 484 outputs for Rimes (96 five-state charcter HMMs, and two space HMMs with two states)
- 472 outputs for IAM (78 six-state charcter HMMs, and two space HMMs with two states)
- 556 outputs for Bentham (92 six-state charcter HMMs, and two space HMMs with two states)

The decoding was performed with Kaldi, and the MLPs replaced the GMMs in the HMMs trained in Section 4.5. The complete system is a hybrid NN/HMM. The outputs of the networks $p(s|x_t)$ are scaled by the state priors $p(s)$, following Equation 2.7:

$$\frac{p(s|x_t)}{p(s)^\kappa} \tag{5.1}$$

where κ is a scaling factor on the state priors, tuned on the validation sets. The state priors are estimated from frequencies in the GMM/HMM forced alignments. The optical scale and word insertion penalty were also tuned.

5.3 Study of the Influence of Input Context

In this section, we are interested in the effect of adding context in the MLP input. We can introduce context in several ways. First, we can widen the sliding window, in order to take more of the input image into account. When working with pixel intensities, we indeed add more information, because the number of inputs is proportional to the width of the window. On the other hand, with the chosen handcrafted features, the dimension of the extracted vector is $50 + 2\omega$, where ω is the width of the window, typically between 3px and 12px. Increasing ω will not add many features, and the added information will only be summarized in the remaining 50 dimensions.

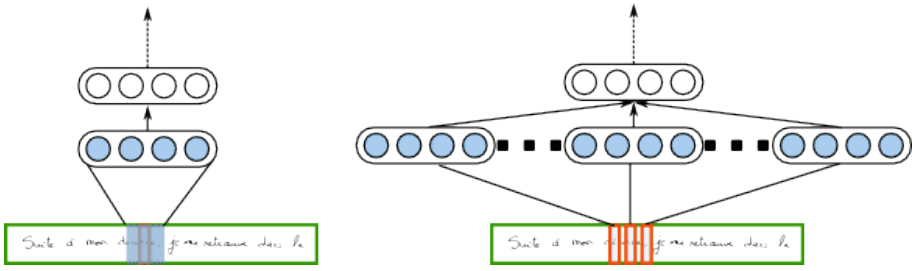


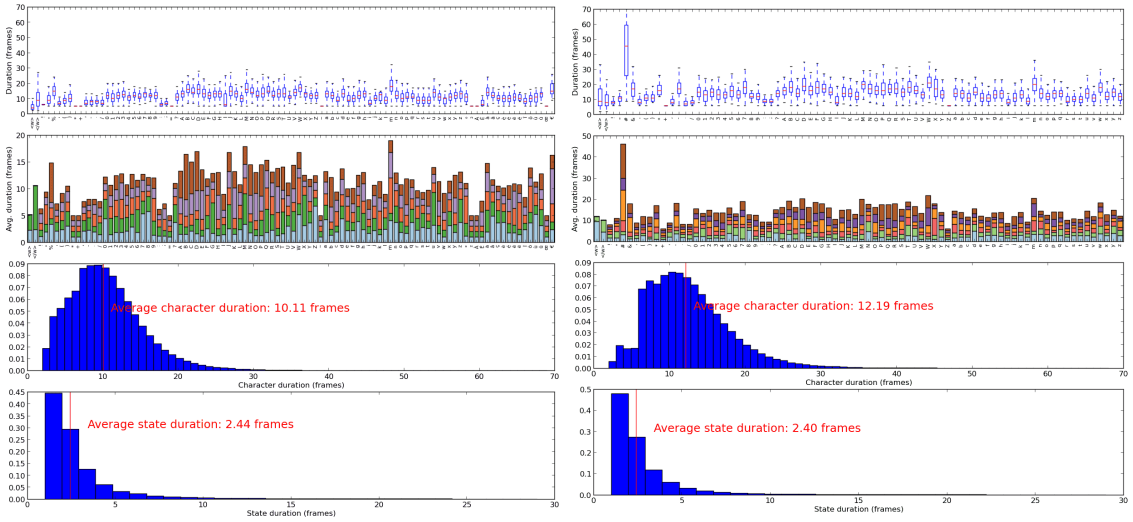
Figure 5.1: Including more context in MLPs. Left: a wider sliding window. Right: concatenation of successive sliding windows. Blue circles correspond to the NN inputs.

Therefore, we may also adopt the technique consisting in concatenating a fixed number of frames, before and after the considered one. Since most of our sliding windows are not overlapping, and the order of the inputs is irrelevant for MLPs, concatenating several frames or taking a wider window is strictly identical as far as pixel intensities are concerned. The different strategies are depicted on Figure 5.1. Note that the concatenation of feature vector is reminiscent of Time-Delay Neural Networks (TDNNs, Waibel et al. (1989)).

5.3.1 Alignments from GMM/HMM Systems

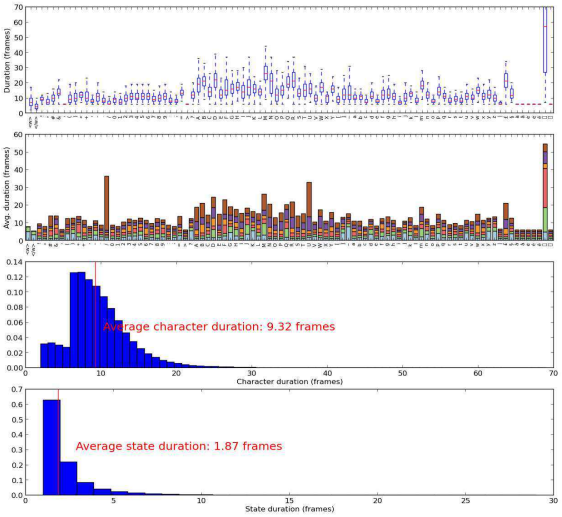
The targets for training the MLPs are obtained with a forced alignment of the text lines of the training set with the baseline GMM/HMM system. On Figure 5.2, we plot

some statistics of the retrieved alignments. We show the distributions of character lengths in terms of frames on the top plot, and the average duration for each state of each character. On the bottom two plots, we show the distributions of character and state durations.



(a) Rimes

(b) IAM



(c) Bentham

Figure 5.2: Statistics of the forced alignments with the GMM/HMM systems. From top to bottom: boxplot of the character length, character average length with average duration of each state, histograms of character lengths and state durations. (**Feat.**).

In Table 5.1, we report the average character width, calculated from the forced alignments (Figure 5.2). These values put the context sizes used below in perspective.

Table 5.1: Character widths, estimated a priori from the images and annotations, and from the alignments with the GMM/HMM system.

	Alignments	
Rimes	30.3px	(10.1 frames)
IAM	36.6px	(12.2 frames)
Bentham	28.0px	(9.3 frames)

5.3.2 Handcrafted Features

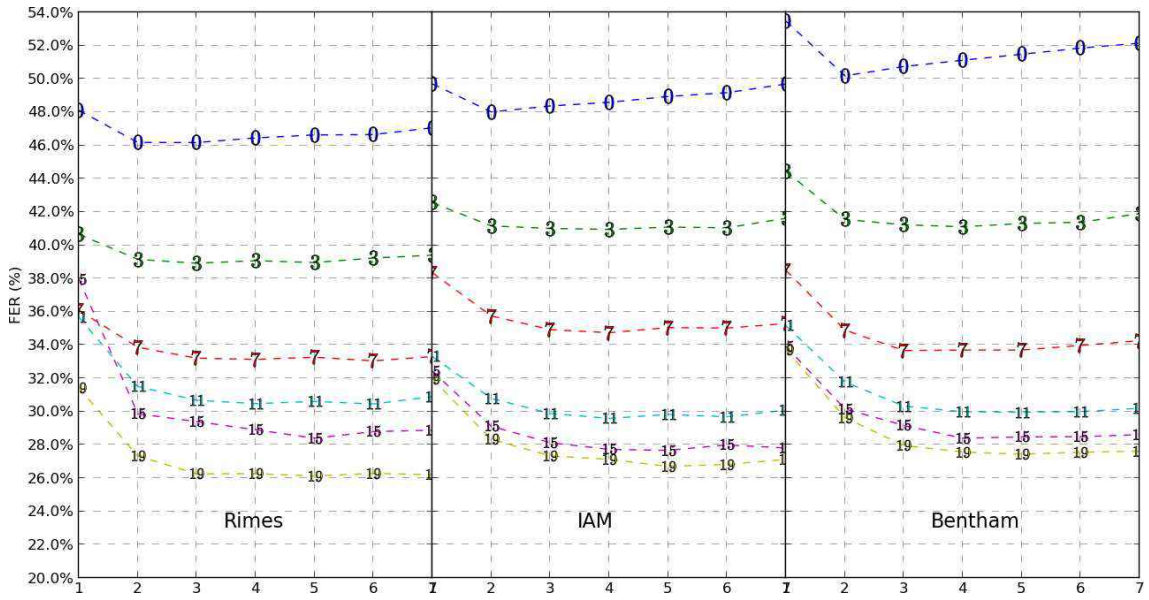
In this first experiment, we trained different deep MLPs with the procedure described previously, and the same targets obtained by forced alignments of the GMM/HMM baseline system, but different size of context, *i.e.* different number of frames concatenated on each side of the considered one.

Table 5.2: Effect of concatenating several consecutive frames on deep MLP results (Word Error Rates; **Feat.**/**Deep**/**Xent**).

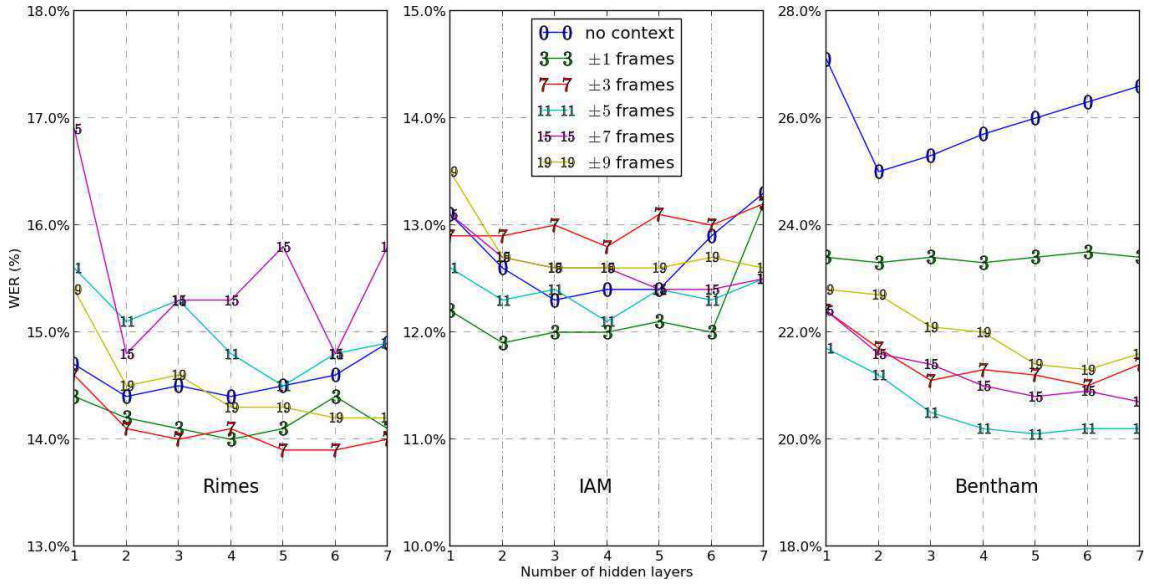
Context frames	Rimes	IAM	Bentham
no context	14.4%	12.3%	25.0%
±1 frame	14.0%	**11.9%**	23.3%
±3 frames	**13.9%**	12.8%	21.0%
±5 frames	14.5%	12.1%	**20.1%**
±7 frames	14.8%	12.4%	20.7%
±9 frames	14.2%	12.6%	21.3%

In Table 5.2, we show the WERs on the development sets of Rimes, IAM and Bentham databases, obtained with MLPs consisting of hidden layers with 1,024 sigmoid units. We varied the number of hidden layers, and for each size of context and each database, we report the results of the best MLP. We observe that adding the right amount of context brings relative WER improvements from 3 to 20% over the single frames without context in the complete hybrid NN/HMM systems with language models.

On Figure 5.3, we plot, for each database, the Frame Error Rate (FER), *i.e.* the classification error of the MLP alone, without lexicon and language model, and the Word Error Rate (WER) with language model, obtained with different context sizes and the depths of the MLP. Note that varying the size of context in this experiment has an effect on the total number of adjustable parameters. However, when the network becomes deeper, this difference remains constant and becomes negligible. Interestingly, although depth is important, as we will see in Section 5.4, the effect of choosing the right context size is much more noticeable. Therefore, this parameter should not be overlooked when designing an hybrid MLP/HMM system.



(a) Frame Error Rate (FER%)



(b) Word Error Rate (WER%)

Figure 5.3: Effect of context and depth on MLP performance (**Feat.**/**Deep**/**Xent**).

The MLPs are trained to classify inputs into HMM states. On Figure 5.3a, it is clear that adding more context in the input is highly beneficial to the classification accuracy, and the results are almost always better with more context. With no context, using the original frames, the high FERs may be explained by the fact that the windows are small. They are 3px wide, which is less than 10% of the average character size. Thus, similar observations will correspond to different parts of different characters, *i.e.* different HMM states, which makes the classification difficult.

With ±5 frames, a total of 11 frames are fed to the network, which approximately

corresponds to the average character size. We observe big improvements when we add context from none to ±5. There are some improvements beyond that number, that is, including more than a character on average, but smaller ones. It seems a good choice to select a context size about the size of a character, or slightly bigger.

The differences of WERs, when the MLP is included in the complete hybrid NN/HMM system, with language model, are not as visible. Choosing the right amount of context seems important nonetheless, and we notice that the difference between a bad and a good one can represent an absolute difference in WER of 1% on Rimes and IAM. On Bentham database, the differences are more obvious.

In any case, the best context size in the complete system is not quite correlated with the best one in terms of FER. Based on the experimental results shown on Figure 5.3b, we chose ±3 frames for Rimes, ±1 frames for IAM, and ±5 for Bentham database.

5.3.3 Pixel Intensities

Instead of building a bootstrapping system with pixel inputs to retrieve the forced alignments for the training set, we used the same ones as for the MLP based on handcrafted features. In order to have as many observation vectors as target HMM states, we applied the same shift to the sliding window of pixels as for the features, as explained in Section 4.2.2.2.

For pixels, we did not concatenate consecutive frames, but instead extracted the values from a wider window. Moreover, to limit the number of inputs, each frame is rescaled to a height of 32px, keeping the aspect ratio fixed (cf. Section 4.2.2.2).

In Table 5.3, we report the result of varying the sliding window width. 21px (*resp.* 45px) corresponds to the same context as ±3 (*resp.* ±5) feature frames, and 72px corresponds to a square sliding window. Again, we observe that the context has an important effect on the performance.

We chose a pixel sliding window of 45px for Rimes. Since with handcrafted features on IAM, the same context size was about as good as for Rimes, we also extracted pixel values with a window of 45px. For Bentham database, the best context size with handcrafted features was bigger, so we used a bigger pixel window of 57px. These choices are heuristical, and the results we report in the rest of this thesis for systems based on pixels could certainly be improved with a more careful choice of the window size.

Table 5.3: Influence of the size of the sliding window for deep MLPs with pixel values inputs (Rimes; **Pix.**/**Deep**/**Xent**).

Window size	WER%	CER%
21px	15.8	4.7
45px	14.1	4.2
72px	16.5	4.9

5.4 Study of the Impact of Depth in MLPs

In this section, we study the importance of the depth of MLPs, *i.e.* the number of hidden layers. We trained MLPs on handcrafted features and pixels, with one to seven hidden layers, for Rimes, IAM and Bentham databases. We performed the layerwise pretraining method to initialize the weights of seven hidden layers. To train an MLP with n hidden layers, we only kept the initialized weight matrices of the first n layers. Thus the pretraining procedure is done only once, and all networks of each configuration have undergone the same pretraining, and initially share the same weights in lower layers.

5.4.1 Deep MLPs

For this set of experiments, we trained MLPs with 1,024 hidden units per layer, with feature and pixel inputs. The results for handcrafted features were already reported on Figure 5.3 in Section 5.3. We have seen that the amount of context was important for the classification accuracy, but also for the performance of the whole system. On Figure 5.3, we observe that the variations in FER (no language model) and WER (with language model) are more important when the context varies than when depth does. However, we notice relative FER improvements up to 20% going from one to several hidden layers. The improvements due to depth are bigger for networks with more context. Overall, four or five hidden layers look like a good choice to get optimal FER: the improvements beyond that number are relatively small.

When the MLP is plugged in the complete hybrid system, including the language model, the improvements of WERs are less visible than those of FERs. For handcrafted features, we observe a relative improvement in WER between 2.5% (IAM) and 7.4% (Bentham), when we compare networks with a single hidden layer to MLPs with more hidden layers, for a given context size. The results are reported on Table 5.4, using the best context size found in Section 5.3.

The gap between one and more hidden layers is more manifest for pixel inputs than for handcrafted features. The comparison between one and two hidden layers shows a relative WER improvement between 6 and 10%. With the best depth, we report relative WER improvements up to 17%. The best deep MLPs with features and pixels yield similar WERs, except for Bentham database. It looks like with deep neural networks, it does not make a difference to use handcrafted features or pixels.

The results support the general agreement in the deep learning community that deep architectures automatically extract helpful features of increasing complexity as the input is propagated forward through the network (Bengio, 2009). Depth has less influence when the inputs are handcrafted features, but with pixels, the first layers extract features that are learnt from the frame images, and deeper networks are necessary to get good results.

Since each hidden unit has a connection to each pixel of the input frame, we can show the weights graphically, as images of the same size as the input frame. The intensity

Table 5.4: MLPs on handcrafted and pixel features. The Frame Error Rates (FERs) measure the classification error of the MLP alone, while the Word and Characer Error Rates (WER, CER) are obtained with lexicon and language model. (**Xent**)

	Depth	**Features**			**Pixels**		
		FER%	WER%	CER%	FER%	WER%	CER%
Rimes	1	36.1	14.6	4.4	38.3	15.4	5.3
	2	33.9	14.1	4.2	34.5	14.5	4.5
	3	33.2	14.0	4.0	33.4	14.6	4.5
	4	33.1	14.1	4.0	32.8	14.2	4.3
	5	33.3	**13.9**	**3.9**	32.2	**14.1**	**4.2**
	6	**33.0**	**13.9**	4.0	32.4	14.2	4.4
	7	33.4	14.0	4.0	**32.0**	**14.1**	**4.2**
IAM	1	42.6	12.2	4.3	42.2	13.6	5.2
	2	41.2	**11.9**	**4.2**	39.1	12.7	4.6
	3	41.0	12.0	**4.2**	37.9	12.6	4.5
	4	**40.9**	12.0	4.3	37.3	12.5	4.4
	5	41.1	12.1	4.3	**37.0**	**12.3**	**4.2**
	6	41.1	12.0	4.3	**37.0**	12.5	4.4
	7	41.6	13.2	4.3	37.6	**12.3**	4.4
Bentham	1	35.2	21.7	9.8	48.2	27.1	15.0
	2	31.8	21.2	9.7	43.3	24.6	12.6
	3	30.3	20.5	8.9	40.9	23.7	11.8
	4	**30.0**	20.2	**8.6**	40.1	23.5	11.6
	5	**30.0**	**20.1**	**8.6**	39.4	22.7	10.8
	6	**30.0**	20.2	8.8	39.2	22.6	10.9
	7	30.2	20.2	8.7	**39.0**	**22.5**	**10.5**

of each pixel reflects the value of the corresponding weight, so we can see this as a representation of the feature computed by the units, or as a filter applied to the input image.

On Figure 5.4, we show the weights of the first hidden layer of the pixel MLPs. On top, we display the weights after the RBM pre-training. Below, we show the same weights, after the cross-entropy training of the whole MLP. We see that the RBM weights are localized to a specific position in the image, and features computed by this first layer are mainly blobs or couple of blobs. Each unit detects the presence of black pixels at different positions. After supervised training of the whole network, the weights are modified. For a deep MLP (bottom), the weights did not change much, and the hidden units are still pixel detectors, *i.e.* very elementary feature detectors. When there is only one hidden layer, the weights are more modified, and the filters look like character parts, or local oriented edge detectors.

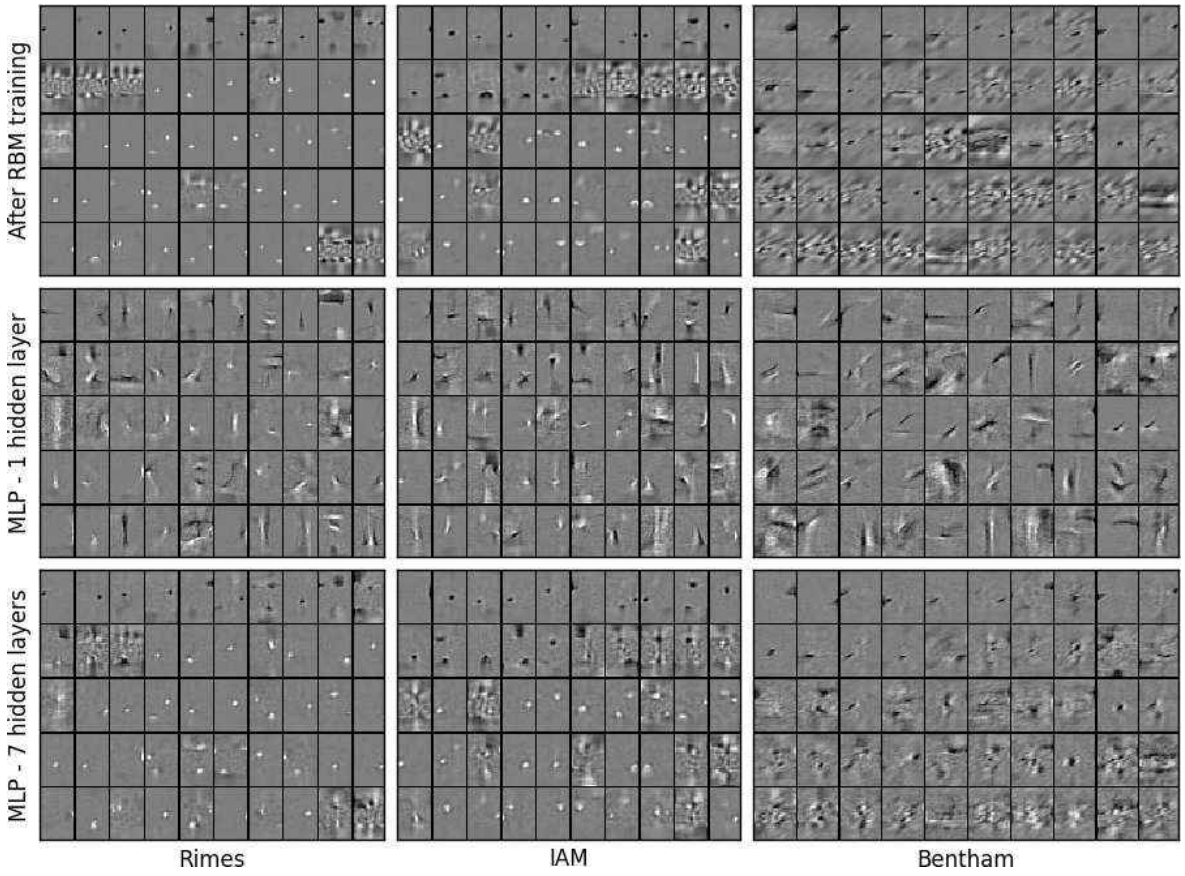
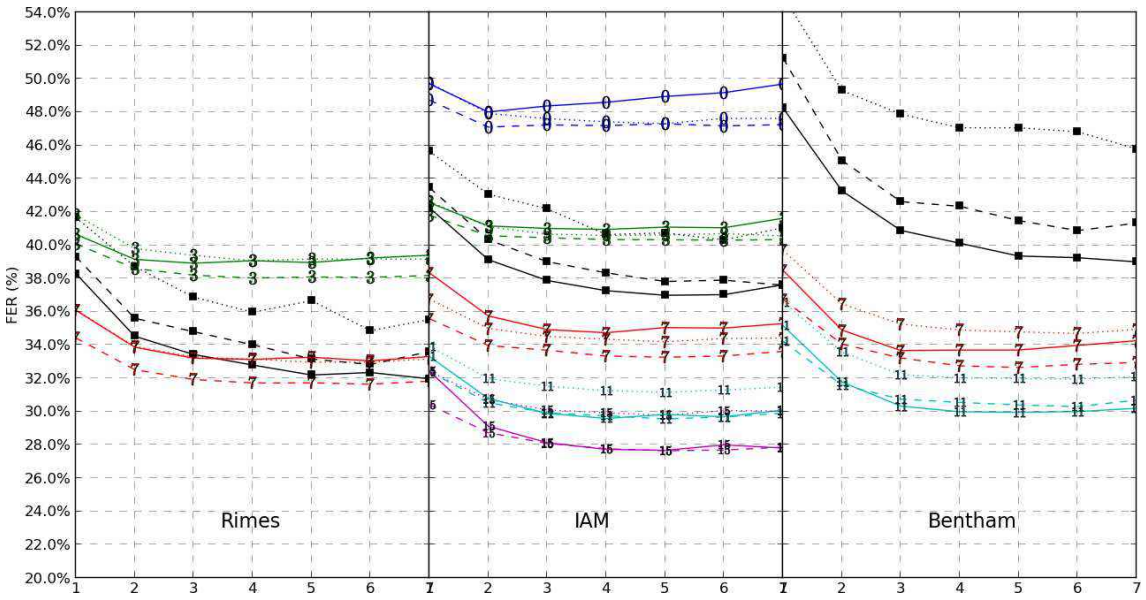


Figure 5.4: MLP input filters (representing the weights between the input and first layer) with pixel inputs on three databases, after pretraining and after fine-tuning. (**Pix.**/**Deep**/**Xent**)

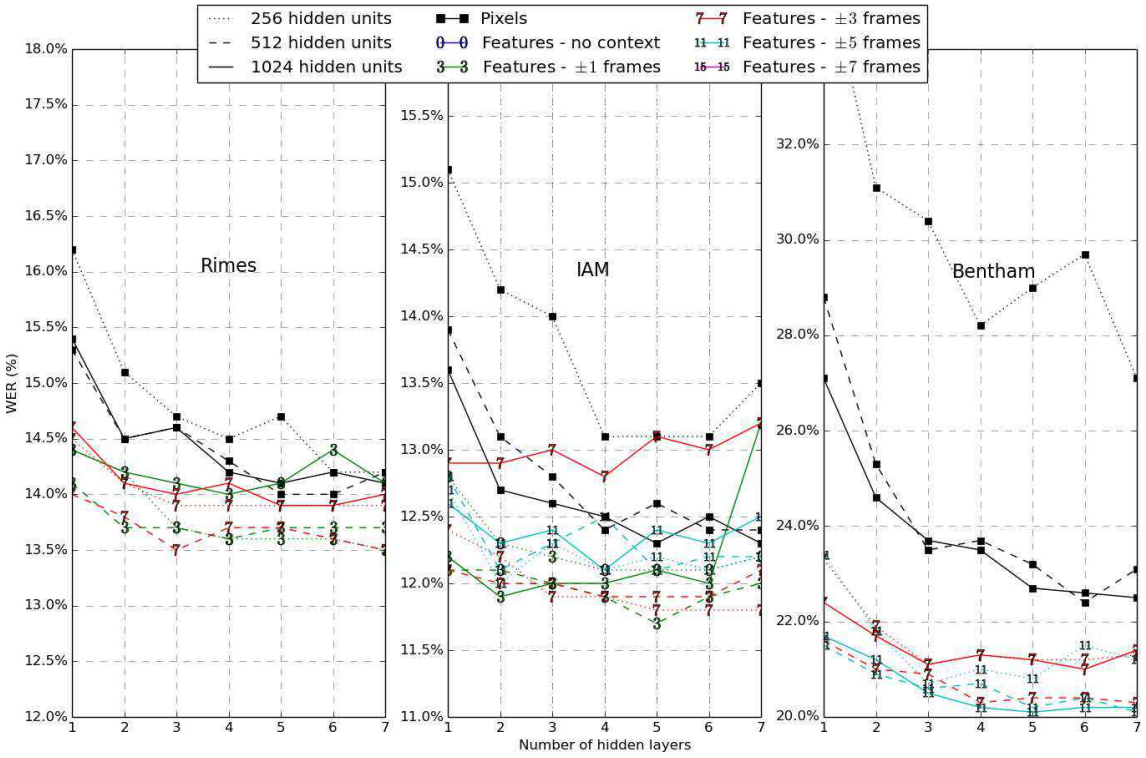
5.4.2 Deep vs Wide MLPs

For every added hidden layer in the previous experiment, one million parameters are introduced in the network. Therefore, the observed improvement may be caused by the augmented model capacity rather than by the depth. In this section, we present the results obtained with MLPs having 256, 512, and 1,024 hidden units in each layer, and one to seven hidden layers.

The FERs and WERs, for features and pixels, different sizes of context, and different sizes and numbers of hidden layers, are shown on Figure 5.5. We see that, although the number of hidden units in each layer influences the classification accuracy (Figure 5.5a), increasing the depth almost always yields some improvements, and deep MLP with a few units tend to be better than MLPs with one large hidden layer. For example, there are approximately the same number of parameters in a MLP with 7 hidden layers of 256 units and in an MLP with one hidden layer of 1,024 units, but the performance is better with the former.



(a) Frame Error Rate (FER%)



(b) Word Error Rate (WER%)

Figure 5.5: Comparison of the effect of width and depth on MLP performance. The line styles represent the number of hidden units per layer (dots: 256, dashes: 512, solid: 1,024). The line colors represent different inputs (black: pixels, colors: several concatenation of features) (**Deep**/**Xent**).

The effect of depth is more visible with pixel inputs than with features. We also notice that it is more visible in general when there are less hidden units per layer.

5.5 Study of the Benefits of Sequence-Discriminative Training

In this section, we explore the sequence-discriminative training of deep MLPs. In speech recognition, this procedure improves the results, generally by a relative 5 to 10% (Veselý et al., 2013; Su et al., 2013). Among different possibilities, we chose the state-level Minimum Bayes Risk (sMBR) criterion, described in (Kingsbury, 2009), which yields slightly better WERs than other sequence criteria on a speech recognition task (Switchboard; Veselý et al. (2013)). The objective function maximized by this training procedure is

$$\mathcal{F}_{MBR}(\mathcal{S}) = \sum_{(\mathbf{x},\mathbf{W}_r)\in\mathcal{S}} \frac{\sum_{\mathbf{W}} p(\mathbf{W}|\mathbf{x})A(\mathbf{W},\mathbf{W}_r)}{\sum_{\mathbf{W}'} p(\mathbf{W}'|\mathbf{x})} = \sum_{(\mathbf{x},\mathbf{W}_r)\in\mathcal{S}} \frac{\sum_{\mathbf{W}} p(\mathbf{W})p(\mathbf{x}|\mathbf{W})A(\mathbf{W},\mathbf{W}_r)}{\sum_{\mathbf{W}'} p(\mathbf{W}')p(\mathbf{x}|\mathbf{W}')} \tag{5.2}$$

where $\mathcal{S}$ is a training set and $A(\mathbf{W}_1, \mathbf{W}_2)$ is a count of correct HMM states encountered during the computation of the sum. In practice, the reference word sequence $\mathbf{W}_r$ is represented by a sequence of states, obtained with forced alignment. Both sums in the objective function are computed efficiently with a variant of the forward-backward algorithm in lattices. The propagated gradient to maximize the objective function $\mathcal{F}_{MBR}$ is also a product of the forward-backward procedure (Kingsbury, 2009). Instead of optimizing the HMM state classification, sequence discriminative training aims at maximizing the probability of the correct sequences, while minimizing the probability of competing ones.

Table 5.5: Selected MLP architectures

		Context	Hidden layers
Rimes	Features	±3 frames	3×512
	Pixel	-	5×512
IAM	Feature	±3 frames	5×256
	Pixels	-	$5 \times 1,024$
Bentham	Feature	±5 frames	7×512
	Pixels	-	6×512

We selected the best networks obtained previously for each database and input type (Table 5.5), and further trained them with a few epochs of sMBR. First, we re-aligned the training set using the cross-entropy-trained networks. Lattices are then extracted with a closed vocabulary and a language model, using the selected networks. We did not regenerate lattices during sequence training. We tried several language models $p(\mathbf{W})$, estimated from the annotations of the training set: zerogram, unigram and bigram. The zerogram is a uniform distribution over all words. We ran a few epochs of sMBR training with a learning rate of 10^{-4}.

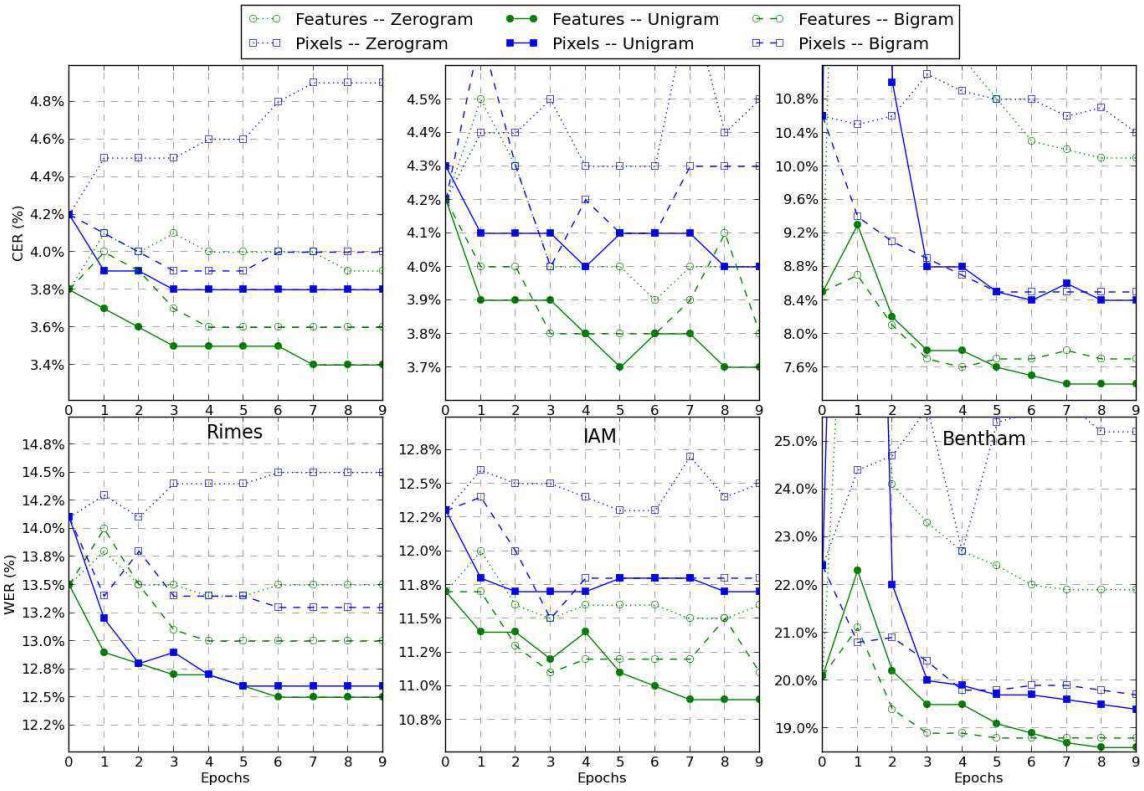


Figure 5.6: WER and CER evolution during sequence-discriminative training (**Deep**/**Seq.**).

The evolution of the WER and CER during sMBR training is shown on Figure 5.6 for all databases and types of inputs. The points at epoch 0 correspond to the performance of the MLPs trained with cross-entropy. Regarding the order of the language model used to generate lattices, a zerogram, where all words have the same probability, is not sufficient: in most cases, it lead to degraded performance of the sequence-trained networks. On the other hand, a bigram language model did not yield much improvement over a unigram one, and the results were even worse most of the time. With a unigram language model, for all configurations (solid lines in Figure 5.6), both the CER and the WER were improved by sequence training.

On Table 5.6, we report the results of the final systems, before and after sequence-discriminative training. We record relative WER improvements ranging from 5 to 13%, which is consistent with the observations made in speech recognition. With handcrafted features, these improvements are bigger than those observed by increasing the number of hidden layers.

The success of this training procedure seems to rely on the information brought by the language model, as shown by the lack of improvement with a zerogram. However, the systems seems to benefit from the variety of the candidate sequences of words. If we increase the language constraints, changing from a unigram to a bigram, the observed improvements with respect to a cross-entropy training tend to diminish.

Table 5.6: Effect of sMBR training. The *cross-entropy* corresponds to framewise training, as oposed to sMBR, which is a sequence-discriminative criterion (**Deep**/**Seq.**).

		Features		**Pixels**	
		WER%	CER%	WER%	CER%
Rimes	Cross-entropy	13.5	3.8	14.1	4.2
	+ sMBR	12.5	3.4	12.6	3.8
		(-7.4%)	(-10.5%)	(-10.6%)	(-9.5%)
IAM	Cross-entropy	11.7	4.2	12.3	4.2
	+ sMBR	10.9	3.7	11.7	4.0
		(-6.8%)	(-11.9%)	(-4.9%)	(-4.5%)
Benth.	Cross-entropy	20.1	8.5	22.4	10.6
	+ sMBR	18.6	7.4	19.4	8.4
		(-7.5%)	(-12.9%)	(-13.4%)	(-20.8%)

5.6 Study of the Choice of Inputs

In this section, we focus on the performance of deep MLPs with two kinds of inputs: handcrafted features and pixels. Reaching the same error rates with features and pixels may be interesting. First, it alleviates the need to implement the feature extraction methods. Moreover, features might be good for one script, *e.g.* Latin, but not suited to another one, for example Arabic. Finally, using pixels removes potential assumptions about what are the relevant features of handwriting, and leaves the decision to the machine learning algorithm.

Table 5.7: Comparison of WERs (%) obtained with handcrafted features and pixel values.

	Features	**Pixels**	
Rimes			
1-layer MLP	14.0%	15.3%	(+9.3%)
Deep MLP	13.5%	14.0%	(+3.7%)
+sMBR	12.5%	12.6%	(+0.8%)
IAM			
1-layer MLP	12.4%	13.6%	(+9.7%)
Deep MLP	11.8%	12.3%	(+4.2%)
+sMBR	10.9%	11.7%	(+7.3%)
Bentham			
1-layer MLP	21.5%	28.8%	(+34.0%)
Deep MLP	20.1%	22.4%	(+11.4%)
+sMBR	18.6%	19.4%	(+4.3%)

In Table 5.7, we compare the results of the modeling of features and pixels. We see that going from MLPs with a single hidden layer to deep MLPs greatly reduces the gap between handcrafted features and pixel inputs. The relative WER difference between features and pixels drops from about 9% to 4% for Rimes and IAM, and from 34% to 11% for Bentham. Sequence discriminative training provides another significant reduction of the difference, except for IAM.

5.7 Conclusion

In this chapter, we have studied MLPs for handwriting recognition on Rimes, IAM and Bentham databases, using handcrafted features and pixels. We have built deep networks, with one to seven hidden layers, trained with a framewise classification criterion, the cross-entropy, and a sequence-discriminative one, sMBR.

We have seen that using the single frames extracted with the sliding windows was not optimal, and that ***providing the MLP with the right amount of input context was crucial***. The classification accuracy is largely improved by giving more context. The improvements are not so evident in the complete system, but the context is still a source of variation, and a few percents of relative WER improvements can be expected by tuning this parameter.

We trained MLPs of different depths, and we have shown that ***deep MLPs achieve significantly better results than ones with a single hidden layer***. We observed that depth itself produced improvements, which were not only due to the larger number of parameters in deeper networks.

The improvements due to deeper networks are even clearer with pixel inputs. In that case, the gap between one and more hidden layers is bigger, suggesting that deep networks are particularly suited to that kind of inputs. This supports the idea that an automatic extraction of learnt features happens in the lower layers of the network. Thus ***MLPs with pixel inputs require more hidden layers, but finally achieve similar performance as MLPs operating with handcrafted features***, showing that carefully designing relevant features may not be necessary.

Finally, we have shown that ***fine-tuning the MLPs with a sequence discriminative criterion (sMBR) yields significant improvements***, between 5 and 13% of WER, and up to 20% of CER. This is consistent with what one can read in the speech recognition literature. The language model in the computation of the cost is important, although a simple unigram seems sufficient to attain these improvements.

Chapter 6

Hybrid Deep Recurrent Neural Networks / HMM for Handwriting Recognition

Contents

6.1 Introduction 135
6.2 Experimental Setup 136
6.2.1 RNN Architecture Overview 136
6.2.2 Decoding in the Hybrid NN/HMM Framework 138
6.3 Study of the Influence of Input Context 139
6.3.1 Including Context with Frame Concatenation 140
6.3.2 Context through the Recurrent Connections 140
6.4 Study of the Influence of Recurrence 142
6.5 Study of the Impact of Depth in BLSTM-RNNs 143
6.5.1 Deep BLSTM-RNNs 144
6.5.2 Deep vs Wide BLSTM-RNNs 146
6.5.3 Analysis 148
6.6 Study of the Impact of Dropout 149
6.6.1 Dropout after the Recurrent Layers 151
6.6.2 Dropout at Different Positions 151
6.6.3 Study of the Effect of Dropout in Complete Systems (with LM) . 155
6.7 Study of the Choice of Inputs 156
6.8 Conclusion 158

6.1 Introduction

In Recurrent Neural Networks (RNNs), the activations of hidden layers at a given timestep are fed to the network in the following timesteps, making them specially designed to process sequences. Potentially, the context of the whole sequence contributes to the decision at time t, while with MLPs, the amount of considered context has to be fixed in advance. Since handwriting recognition involves a processing of sequences, RNNs seem suited to the problem. Actually, most systems winning international evaluations in the last few years, as well as state-of-the-art ones, involve a RNN component.

For example, Graves & Schmidhuber (2008); Bluche et al. (2014a); Pham et al. (2014); Moysset et al. (2014) use Multi-Dimensional LSTM-RNNs, which are deep and implement convolutional concepts. The LSTM layers are connected to local neighborhoods in the image, and are followed by convolutional and subsampling layers. As in many Convolutional Neural Networks, only a few features are computed in lower layers, and more features in upper layers, as the size of input maps decreases.

On the other hand, Bidirectional LSTM-RNNs (BLSTM-RNNs) process sequences of input vectors, and are densely connected. More features are extracted in each layer, but, as for MLPs, there are only shallow architectures with one or two hidden layers in the literature. For example, Kozielski et al. (2013a, 2014a); Doetsch et al. (2014) build tandem and hybrid systems using RNNs with two hidden layers.

In this chapter, we focus on BLSTM-RNNs, for which we can use the same inputs as the previous GMM and MLP systems, trained with the Connectionist Temporal Classification (CTC) criterion. We performed handwriting recognition in the hybrid NN/HMM framework. We are interested in answering the following questions:

- *Is it still important to design handcrafted features when using deep RNNs, or are pixel values sufficient?*
- *Do deep BLSTM-RNNs yield improvements over RNNs with only one or two hidden layers?*
- *What is the importance of the recurrence at different depths in BLSTM-RNNs?*
- The input context was important for classification in MLPs. *Do RNNs benefit from more input context, or is it better to let the recurrence learn the amount of context needed?*
- Dropout (Hinton et al., 2012) is a regularization technique, which has proven very efficient for deep MLPs (Dahl et al., 2013), ConvNNs (Krizhevsky et al., 2012), and RNNs (Zaremba et al., 2014). Pham et al. (2014) proposed to apply it at a specific location in MDLSTM-RNNs and reported significant improvements. *Does it also help in deep BLSTM-RNNs? Where is the best position to apply dropout in these models?*

In Section 6.2, we introduce the experimental setup for this chapter. In particular, we describe the RNN architecture that we used, along with the training and decoding procedure.

In Section 6.3, we show that the concatenation of input frames has little effect on the final results in RNNs, supporting the idea that RNNs can effectively learn the necessary context.

In Section 6.4, we present experiments with recurrent layers at different depth in the network: close to the input or output layer, or in the middle and show that more recurrence is better, and that the recurrence in the top layer is especially important, in the CTC framework.

We trained BLSTM-RNNs with different depths. The results, reported in Section 6.5, prove that RNNs can benefit from being deep, even without pre-training.

We experimented the dropout technique in BLSTM-RNNs, and its application at different locations in the network. We show in Section 6.6 that the improvements of Pham et al. (2014) can be reproduced with BLSTM-RNNs on the one hand, and that we can achieve significant improvements by applying dropout at different positions from the one they suggested, on the other hand.

Finally, in Section 6.7, we point out that the discrepancy between the results obtained with handcrafted features and with pixel values is reduced by using deep BLSTM-RNNs.

We draw some conclusions in Section 6.8.

6.2 Experimental Setup

6.2.1 RNN Architecture Overview

In RNNs, the inputs to a given recurrent layer are not only the activations of the previous layers, but also its own activations at the previous timestep. This characteristic enables them to naturally work with sequential inputs, and to use the past context to make predictions. Long Short-Term Memory (LSTM) units are recurrent neurons, in which a gating mechanism avoids the vanishing gradient problem, appearing in conventional RNNs (Hochreiter & Schmidhuber, 1997; Graves & Schmidhuber, 2008), and allow to learn arbitrarily long dependencies.

In Bi-Directional LSTM-RNNs (BLSTM-RNNs), LSTM layers are doubled: the second layer is connected to the "next" timestep rather than the previous one. Thus the input sequence is processed in both directions, so past and future context are used to make predictions.

We used an RNN architecture which resembles the MDLSTM-RNN architecture of Graves & Schmidhuber (2008). The input sequence is fed to two parallel LSTM layers, one for each direction. The outputs of both LSTM layers at a given time are connected to a simple feed-forward layer. The sequence formed by the outputs of this layer may be connected to a new couple of LSTM layers.

Our architecture, depicted on Figure 6.1, alternates pairs of LSTM layers and feed-forward, non-recurrent layers. Every time that we add recurrent layers, we also add a

feed-forward layer before. The last LSTM layers are directly connected to the output layer. Thus, our RNNs will always have an odd number of hidden layers: N LSTM layers, and $N-1$ feed-forward layers. Moreover, we did not use the peephole connections in LSTM units.

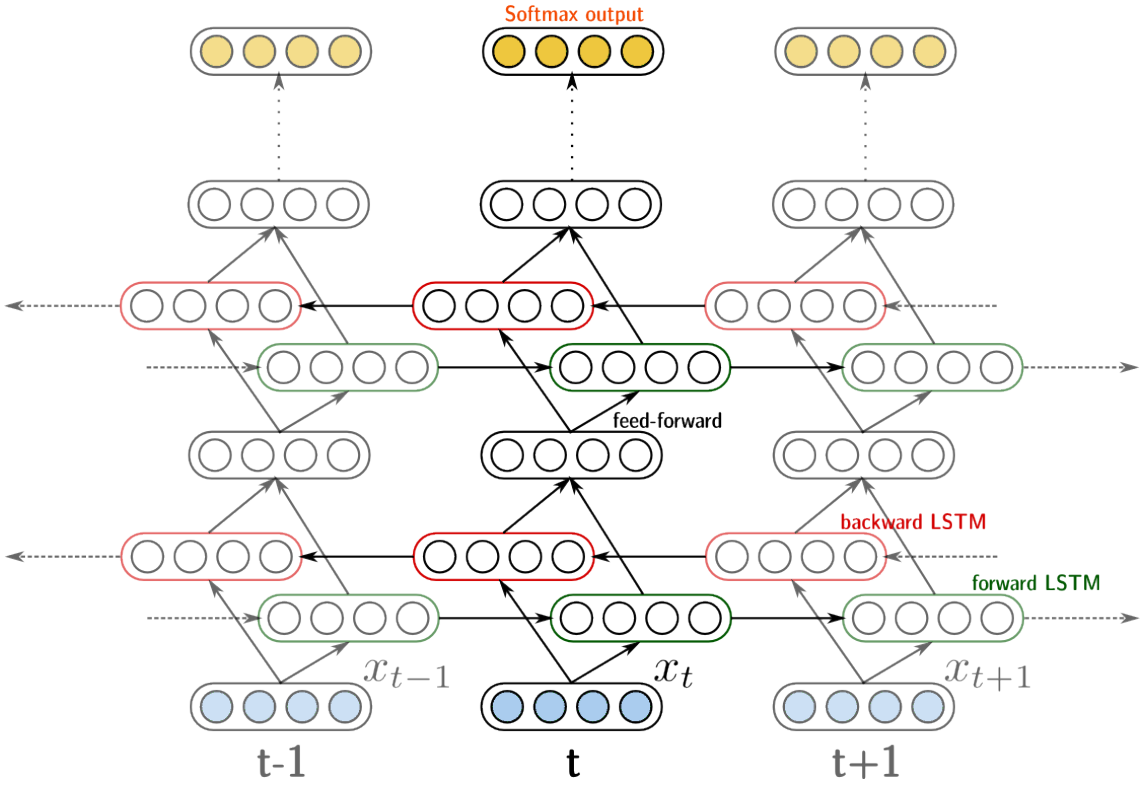


Figure 6.1: Architecture of the BLSTM-RNNs used in this thesis.

We used the Connectionist Temporal Classification (CTC) objective function (Graves et al., 2006) to train the RNNs. With CTC, no prior segmentation of the line images in the training data is required. Therefore, we do not need a bootstrapping procedure involving forced alignments with a previously trained HMM. Instead, we can select the target sequence to be the sequence of characters in the image annotation. In the CTC paradigm, there is one output for each character, plus one *blank* output, corresponding to no character prediction. We also have an output for whitespaces. Therefore, we have:

- 98 outputs for Rimes
- 80 outputs for IAM
- 94 outputs for Bentham

The RNN training was performed with A2iA software, and consisted in minimizing the Negative-Log Likelihood of the labeling, as explained in Section 2.5.4.3. It is done with stochastic gradient descent and BPTT. We noticed that with the CTC objective

function, a learning rate of 0.001 was generally a good choice, yielding fast convergence (around 20-30 epochs) and good results.

The CTC cost is computed at the end of each epoch on the validation set. When it does not decrease for at least 20 epochs, the training procedure is interrupted, and the best RNN is kept.

6.2.2 Decoding in the Hybrid NN/HMM Framework

The interesting property of CTC is that the output of the network can be easily mapped to a sequence of characters. Therefore, the RNN alone can produce transcriptions, using the mapping described in Section 2.5.4.3, and there is no need for an HMM. Wei et al. (2013) proposes methods to match the outputs with a dictionnary word, using the Levenstein distance, or Viterbi-like decoding. However, this method requires to evaluate the words one by one, and does not integrate the language model probabilities. An alternative method, described in (Graves et al., 2006), consists in modifying the forward-backward procedure of CTC training to apply it to a prefix tree representation of the vocabulary, calculating at each iteration the probability of extending the current prefix with a given label. An extension is proposed in (Graves & Jaitly, 2014), giving more details on a beam search decoding procedure and on how to integrate the language model. A similar method is employed by Maas et al. (2014) to decode with CTC trained networks and a language model.

The HMM framework, on the other hand, is interesting because of the mathematical formulation, which separates the optical model and the language model (LM):

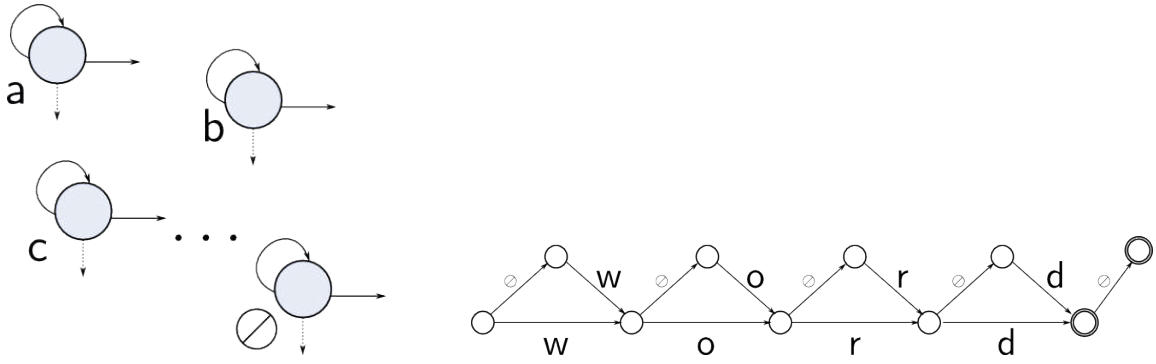
$$\mathbf{W}^* = arg\max_{\mathbf{W}} p(\mathbf{W}|\mathbf{x}) = arg\max_{\mathbf{W}} p(\mathbf{x}|\mathbf{W})p(\mathbf{W}) \tag{6.1}$$

In our decoding pipeline, the HMM, lexicon, and language models are represented as FSTs, and the decoder of the Kaldi toolkit is implemented for HMMs. For the simplicity of implementation, and for the consistency with the rest of this thesis, we chose the hybrid NN/HMM approach, which is to some extent similar to the beam search methods presented above.

Therefore, we integrated the RNNs in the hybrid NN/HMM framework, including the specific aspects coming from the CTC training, namely the blank symbol and allowed transitions. Since there is one output for each character in the RNN, we represented the characters and the blank symbol $\oslash$ with one-state HMMs with loop and outgoing transitions (Figure 6.2a). All transition probabilities are set to 0.5, so that the total weight of transition probabilities is the same for any word sequence for a given observation sequence. This is to reproduce the absence of transition probabilities in the CTC.

The allowed transitions between characters and blanks in a word are encoded in the FST representation of the lexicon (Figure 6.2b). The LM FST is the same as in the previous chapters.

In the hybrid NN/HMM scheme, the NN predictions $p(s|\mathbf{x})$ are scaled by HMM state prior probabilities $p(s)$. Since there are no prior probabilities in the CTC, we



(a) Hidden Markov Model (b) Lexicon FST

Figure 6.2: HMM and Lexicon for hybrid NN/HMM decoding with CTC-trained networks.

could use a uniform distribution for $p(s)$, for the same reason as for uniform transition probabilities. However, we noticed that including prior probabilities improved the global performance of the systems.

Since CTC training does not require pre-segmented data, we cannot estimate the label priors beforehand. We could estimate character priors from the training corpus, but we do not have counts for the blank symbol. Different methods are possible:

- We can record the forced alignments during or after CTC training, and use the label frequencies, as we did for MLPs.
- The forward-backward algorithm applied CTC training already computes the label posteriors $p(q_t = s|\mathbf{x})$, so we can set

$$p(s) \approx \frac{1}{\sum_{\mathbf{x}\in\mathcal{S}} |\mathbf{x}|} \sum_{(\mathbf{x},\mathbf{z})\in\mathcal{S}} \sum_{t=1}^{|\mathbf{x}|} p(q_t = s|\mathbf{x}, \mathbf{z}) \tag{6.2}$$

  where $\mathcal{S}$ is the training set, which is a softer estimate than the forced alignment counts, and is the method we used.

6.3 Study of the Influence of Input Context

In the previous chapter, we have seen that choosing the right amount of context (number of consecutive concatenated frames) to feed MLPs was to some extent more important than depth for classification accuracy.

The main characteristic of RNNs is the recurrent part, making the inner state of the network dependent of the past state(s), and allowing them to naturally process sequential inputs. Thus, they can make predictions based not only on a localized input, but potentially on the whole sequence, provided that the information can flow from remote timesteps. The most basic form of recurrence is difficult to train, due to the vanishing gradient problem. The Long Short-Term memory units were designed to overcome this problem, and the gating mechanism enables the cell to hold or release the information depending on its inputs.

In handwriting recognition, the symbols to recognize are quite localized, and one would not expect to see an influence from the inputs (pixels or feature vectors) outside or too far away from the character boundaries. We could argue that if we take into account a wide enough context, other deep architectures such as MLPs or Convolutional Neural Networks could perform equally well as RNNs. In other words, since the characters are localized, the RNNs should use only a local context, and local predictors, like MLPs with context, could be sufficient.

6.3.1 Including Context with Frame Concatenation

The first experiment consists in concatenating the feature vector of the considered position with a fixed amount of previous and next ones, as we did for MLPs in Section 5.3. The same information will then appear in the RNN input at different positions of successive timesteps. The goal is to see if explicitly giving such context in the input of the network helps, or whether RNNs can achieve better results by learning the necessary amount of context.

We carried out this experiment on Rimes and IAM, with BLSTM-RNNs made of five hidden layers, three LSTM layers and two feed-forward, and 100 hidden nodes in each layer. We trained these RNNs with the original observation sequences, and with the sequences obtained after concatenation of $\pm 1, 3$ or 5 frames.

Table 6.1: Effect of context on BLSTM-RNN performance (**Feat.**/**Deep**/**CTC**).

Context frames	**Rimes**			**IAM**		
	RNN-CER%	WER%	CER%	RNN-CER%	WER%	CER%
None	**8.1**	14.1	**4.1**	**10.4**	12.2	4.3
±1 frame	9.0	14.2	4.2	10.8	**12.1**	**4.1**
±3 frames	9.4	**13.7**	4.2	11.1	12.6	4.4
±5 frames	10.6	14.1	4.4	11.9	12.6	4.6

The results are reported on Table 6.1. The *RNN-CER* is the CER of the RNN used alone, outside the hybrid NN/HMM system, and without lexicon or LM. The *WER* and *CER* are computed in the complete pipeline, including the LM. We observe that the RNN-CER is always better without context, in contrast to MLPs, for which the classification accuracy was always improved by adding context. The effect of adding context on the WER and CER in the complete system is quite limited. We observe small improvements in some cases, but generally the system is not better with more input context in feature vectors.

6.3.2 Context through the Recurrent Connections

In this section, we try to see how RNNs incorporate the context to predict characters through the recurrent connections. Similarly as in (Graves et al., 2013b), we observed the sensitivity of the output prediction at a given time to the input sequence.

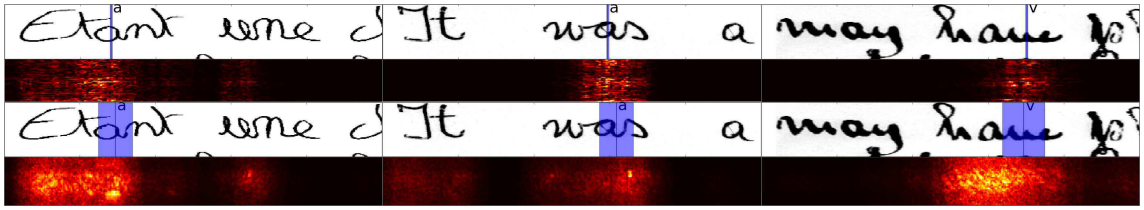
To do so, we computed the derivative of the RNN output y_τ at time $t = \tau$ with respect to the input sequence $\mathbf{x}$. We plotted the sensitivity map $S = (S_{t,d})_{1 \leq t \leq |\mathbf{x}|, 1 \leq d \leq D}$, where D is the dimension of the feature vector, and:

$$S_{t,d} = \left| \frac{\partial y_\tau}{\partial x_{t,d}} \right| \tag{6.3}$$

For pixel inputs, the sliding windows are overlapping, and the dimension of the feature vectors is $D = wh$, where w is the width of the window, and h its height. Therefore, we can reshape the feature vector to get the shape of the frame, and a sensitivity map with the same shape as the image. In overlapping regions, the magnitude of the derivative for consecutive windows are summed:

$$S_{i,j} = \sum_{k=-w/2\delta}^{w/2\delta} \left| \frac{\partial y_\tau}{\partial x_{i+\delta k, i+w(j-1)-\delta k}} \right| \tag{6.4}$$

where $\delta = 3px$ is the step size of the sliding window. This way, we can see the sensitivity in the image space.



(a) Rimes (b) IAM (c) Bentham

Figure 6.3: Context used trough recurrent connections by LSTM-RNNs to predict *"a"* in Rimes, *"a"* in IAM, *"v"* in Bentham (sensitivity heatmaps, top: features, bottom: pixels ; **Deep**/**Drop.**/**CTC**).

On Figure 6.3, we display the results for BLSTM-RNNs with 7 hidden layers of 200 units, trained with CTC and the dropout technique applied after all LSTM layers, with feature and pixel inputs, for all three databases. On each plot, we show on top the preprocessed image, the position τ and the RNN prediction y_τ, as well as the sliding window at this position to put the sensitivity map in the perspective of the area covered by the window at $t = \tau$.

The step size δ of all sliding windows is 3px, *i.e.* the size of the sliding window for features, displayed on the top plots. We observe that the input sensitivity goes beyond ± 5 frames, as well as beyond the character boundaries in some cases, as if the whole word could help to disambiguate the characters.

It is also an indication that RNNs actually use their ability to model arbitrarily long dependency, an ability that MLPs lack. To provide an MLP with all the information handled by the RNN in the case of Figure 6.3a, one would need to concatenate around 70 frames, which seems difficult to achieve, because it would add a lot of variability as well as a lot of parameters, and would probably not improve the performance, considering the results presented in the previous chapter.

6.4 Study of the Influence of Recurrence

In this section, we study the influence and abilities of recurrence in LSTM-RNNs. To do so, we replace LSTM layers by feed-forward ones in BLSTM-RNNs made of five hidden layers (three LSTM and two feed-forward layers). To keep the number of parameters approximately constant, we replace the original LSTM layer with 100 units for each directions by a feed-forward layer with 900 units, as illustrated on Figure 6.4.

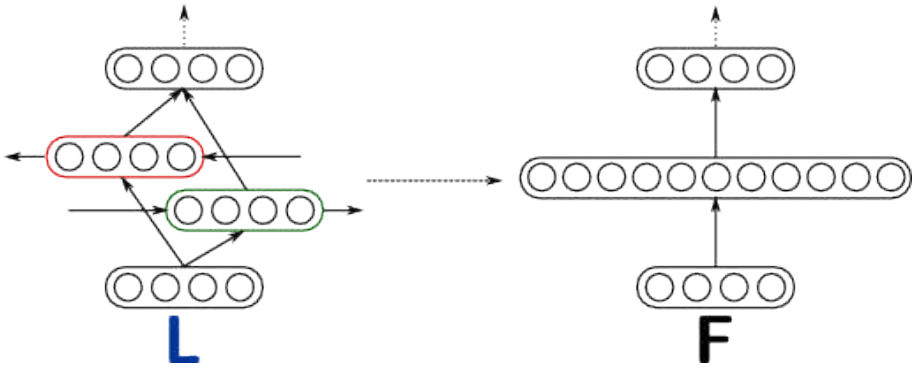


Figure 6.4: LSTM (**L**) and feed-forward (F) blocks.

We keep the intermediate feed-forward layers which were in between LSTM layers. In this architecture, there are three possible positions to include or not the recurrence: bottom, middle and top. We trained all eight possible combinations on Rimes and IAM, with pixel and feature inputs. We refer to the different architectures by a triplet indicating the type of layer at the original LSTM positions (bottom, middle, top). **LLL** represents the original RNNs, while FFF corresponds to a purely feed-forward MLP.

The results are expressed in Table 6.2 in terms of RNN-CER, *i.e.* the CER of the RNN alone, without a language model. The top part of the table are results with features, and the bottom ones are results with pixels. With the usual learning rate of 0.001, some trainings did not converge, which is indicated in the table by the dash symbol. Therefore, we also trained the networks with a smaller learning rate (0.0001).

The first remark is not related to the recurrence, but to the learning rate. With the small learning rate, the training converged in all cases. When it also did with the larger learning rate, the results were always better.

Regarding convergence, we notice that the networks which did not converge have the common property of not having a recurrent top layer. It seems to indicate that at least for CTC, the last LSTM layer plays an important role in the convergence of the training procedure to get good results.

If we look at the different positions of one LSTM layer (**L**FF, F**L**F and FF**L**), in almost all cases, the higher it is in the network, the better the results are. This is especially visible with pixel inputs.

Adding LSTM layers seems generally helpful, although it is not always the case. For example, adding LSTM in the first hidden layer degrades a lot the performance with pixel inputs. It might be due to the fact that for pixels, the lower layers extract

Table 6.2: Effect of recurrence on the character error rate of the RNN alone, without lexicon and language model (RNN-CER%; **Deep**/**CTC**).

	L.Rate	FFF	**L**FF	F**L**F	FF**L**	**LL**F	**L**F**L**	F**LL**	**LLL**
Features	0.001	44.0	-	-	11.9	-	9.3	9.4	8.1
(Rimes)	0.0001	94.1	13.2	12.3	13.0	11.6	11.6	11.6	9.7
	0.001	39.2	13.3	12.4	11.6	-	10.3	10.7	9.7
(IAM)	0.0001	39.6	13.7	13.2	12.5	23.1	11.8	12.0	11.4
Pixels	0.001	-	-	-	13.7	-	11.1	8.9	8.7
(Rimes)	0.0001	38.0	62.2	20.6	17.5	20.8	23.0	15.3	16.7
	0.001	-	-	-	12.3	-	11.8	10.6	11.3
(IAM)	0.0001	32.8	61.3	19.2	17.5	20.3	19.6	17.5	18.9

elementary features from the image. On the other hand, recurrence seems important in the CTC framework, as already stated, and as shown in the next chapter. Therefore, it is probably too difficult for this first layer to learn both the low-level features required to interpret the image and the dependency that are necessary for the convergence of the CTC.

On Figure 6.5, we see that the **L**FF architecture is sensitive to a wide area of the image. However, when we look at the sequence of predictions made by this RNN, we see that it converged to a very sub-optimal solution, where whitespace is predicted in all word segments. For other architectures, the sensitivity is more focused on the area of the predicted letter *"n"*, and networks with more recurrence on top even include more context of the whole word.

6.5 Study of the Impact of Depth in BLSTM-RNNs

RNNs with a single hidden layer are often regarded as deep models, because the output at a given time may depend on the input at a remote timestep. The information flows through many layers sharing weights. We can also increase the number of hidden layers, in order to extract representations of the sequence with increasing complexity. The Multi-Dimensional LSTM-RNN architecture proposed by Graves & Schmidhuber (2008) has five hidden layers, with a structure resembling Convolutional Networks. Few works, however, have studied BLSTM-RNN with more than one or two hidden layers. Graves et al. (2013b) recently reported significant improvements with BLSTM-RNNs with up to five hidden LSTM layers for a speech recognition task.

In this section, we investigate the depth of BLSTM-RNNs for handwriting recognition. In particular,we are interested to see whether adding more than one or two LSTM layers improves the results. We conducted the experiment on Rimes, IAM and Bentham databases, with both kinds of inputs. The networks are trained with the CTC algorithm to predict character and blank labels, with a learning rate of 0.001. We built RNNs with one to thirteen hidden layers of different sizes and studied the

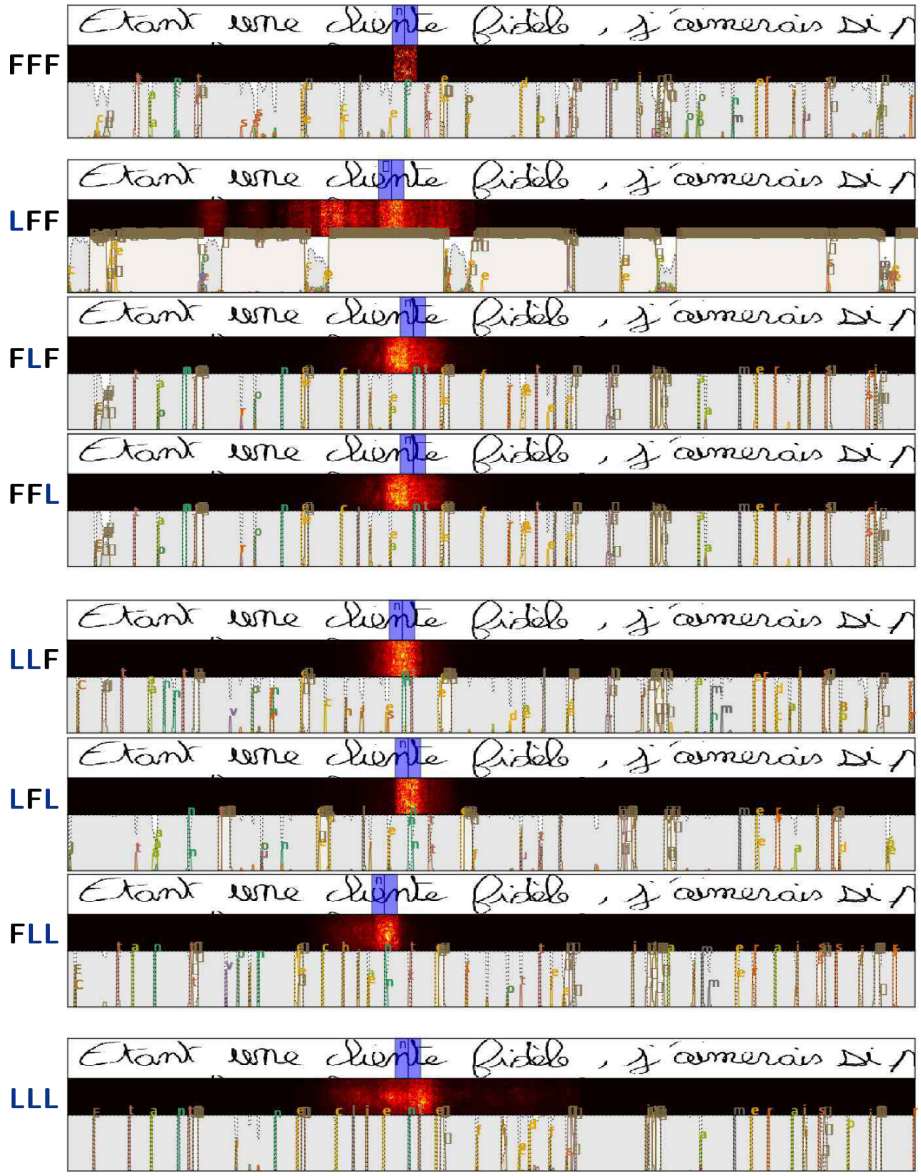


Figure 6.5: Input sensitivity and character predictions of RNNs with different architectures on Rimes database (**Pix.**/**Deep**/**CTC**).

influence of depth on the performance of the RNN and of the complete system.

6.5.1 Deep BLSTM-RNNs

The RNNs in this section have 100 hidden units on each layer, and we varied the number of hidden layers. The results are shown on Figure 6.6, where the CER without and with language models are displayed, and reported on Table 6.3. The RNN-CER is the character error rate when the recognition system is the RNN alone. The WER and CER are respectively the word and character error rates of the hybrid systems, with the vocabularies and language models described in previous sections.

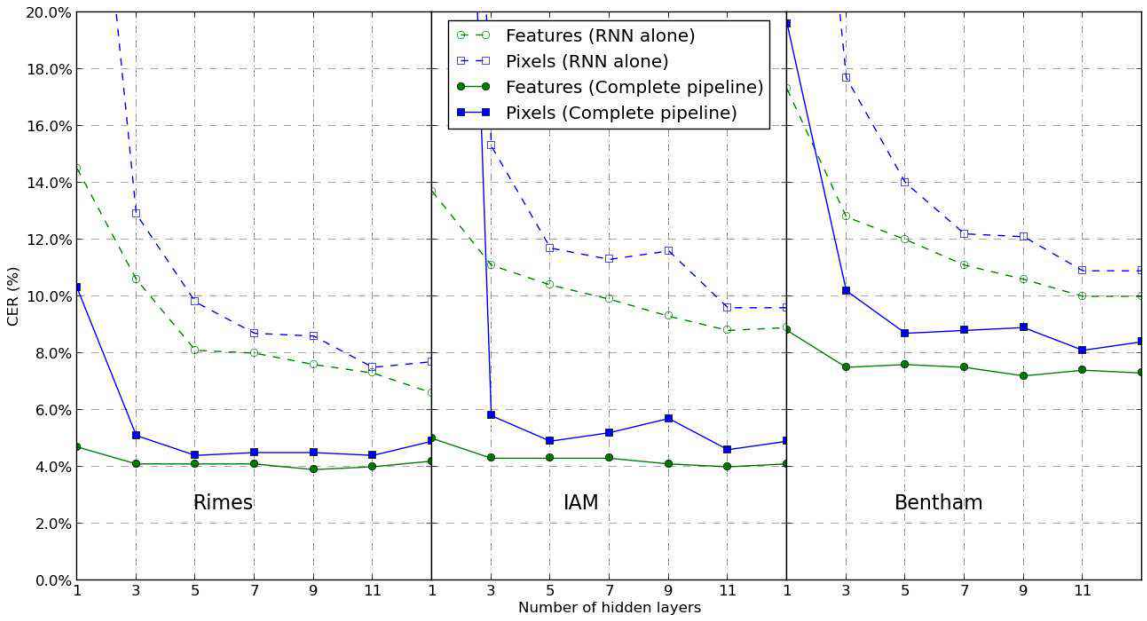


Figure 6.6: Effect of depth on RNN performance (alone and in the complete pipeline; **CTC**).

First, we observe that almost every time we add layers, the performance of the RNN is increased (dashed lines on Figure 6.6). For handcrafted features, adding a second LSTM layer and a feed-forward one brings around a relative 20-25% CER improvement. Adding a third one yields another 6-12% relative improvement.

For pixels, one hidden layer only is not sufficient, and adding another LSTM layer divides the error rates by more than two. A third LSTM layer is also significantly better, by another 20-25% relative CER improvement.

Not surprisingly, when we add the vocabulary and language model, the CER is significantly improved. We still observe the biggest performance gain from one to two LSTM layers. For handcrafted feature inputs, the CER improvement with more hidden layers is small, and a few percents relative WER improvement can be achieved. For pixel intensities, we notice significant WER and CER improvements when we build networks with more than two LSTM layers.

The improvements beyond three hidden layers for handcrafted features, and beyond five for pixels, although present, are not as visible in complete systems as they are for the RNNs alone. This might be explained by the fact that the character errors corrected by deeper networks are also corrected by the linguistic constraints anyway (vocabulary and language model). As we have seen on Figure 6.3, a character prediction can be influenced by regions in the image outside the predicted character, which might suggest that RNNs also learn a sort of language model from the image.

It is also interesting to note that in the method described here, no pretraining was performed, and the networks were directly trained from random weights. We tried

Table 6.3: Effect of depth on the performance of RNNs (CTC).

	Depth	Handcrafted Features RNN-CER	Handcrafted Features WER%	Handcrafted Features CER%	Pixels RNN-CER	Pixels WER%	Pixels CER%
Rimes	1	14.5	14.9	4.7	33.8	24.1	10.3
	3	10.6	13.6	4.1	12.9	15.1	5.1
	5	8.1	14.1	4.1	9.8	**14.0**	**4.4**
	7	8.0	13.8	4.1	8.7	14.5	4.5
	9	7.6	**13.2**	**3.9**	8.6	14.5	4.5
	11	7.3	13.3	4.0	**7.5**	14.8	4.4
	13	**6.6**	14.5	4.2	7.7	15.2	4.9
IAM	1	13.7	13.4	5.0	71.9	83.2	66.7
	3	11.1	12.4	4.3	15.3	14.9	5.8
	5	10.4	12.2	4.3	11.7	13.3	4.9
	7	9.9	12.2	4.3	11.3	13.5	5.2
	9	9.3	11.9	4.1	11.6	15.2	5.7
	11	**8.8**	**11.4**	**4.0**	**9.6**	**12.8**	**4.6**
	13	8.9	11.9	4.1	**9.6**	13.7	4.9
Bentham	1	17.3	20.6	8.8	38.9	33.8	19.6
	3	12.8	18.5	7.5	17.7	22.6	10.2
	5	12.0	19.0	7.6	14.0	20.8	8.7
	7	11.1	18.5	7.5	12.2	21.4	8.8
	9	10.6	**18.3**	**7.2**	12.1	21.6	8.9
	11	**10.0**	19.1	7.4	**10.9**	**20.3**	**8.1**
	13	**10.0**	18.8	7.3	**10.9**	20.5	8.4

to apply the following layerwise pretraining. First, we initialize a network with one LSTM layer and run one epoch of supervised gradient descent. Then we add one layer and train for one more epoch. Finally, when the final depth is reached we run the full training. No improvement were observed on IAM with this technique for various depths and hidden layer sizes.

6.5.2 Deep vs Wide BLSTM-RNNs

Every time we add a new hidden layer, the number of free parameters is increased. So the improvements may be caused by the larger capacity of the network rather than by its depth. In this section, we compare the benefits of depth to the improvements obtained by adding nodes in hidden layers, *i.e.* increasing width.

First, because of the many connections in LSTM units, a lot of parameters are added when we increase the size of an hidden layer. In Table 6.4, we compare the approximate number of parameters in RNNs of different widths and depths, with 100 outputs and 56 (features) or 640 (pixels) inputs.

Table 6.4: Comparison of number of parameters for different architectures.

(a) Features

# hidden units / *# hidden layers*	50	100	200
1	52k	145k	450k
3	97k	325k	1,170k
5	142k	505k	1,890k
7	187k	685k	2,610k

(b) Pixels

# hidden units / *# hidden layers*	50	100	200
1	286k	612k	1,384k
3	331k	792k	2,104k
5	376k	972k	2,824k
7	421k	1,152k	3,544k

We trained networks with 1, 3, 5 and 7 hidden layers, each containing 50, 100 or 200 units, with features and pixels on Rimes, IAM and Bentham databases. The results are shown on Figure 6.7 in terms of RNN-CER. As depth increases, the difference between different widths becomes less visible. Moreover, it is almost always better to add hidden layers than to multiply the number of units per layer by even four.

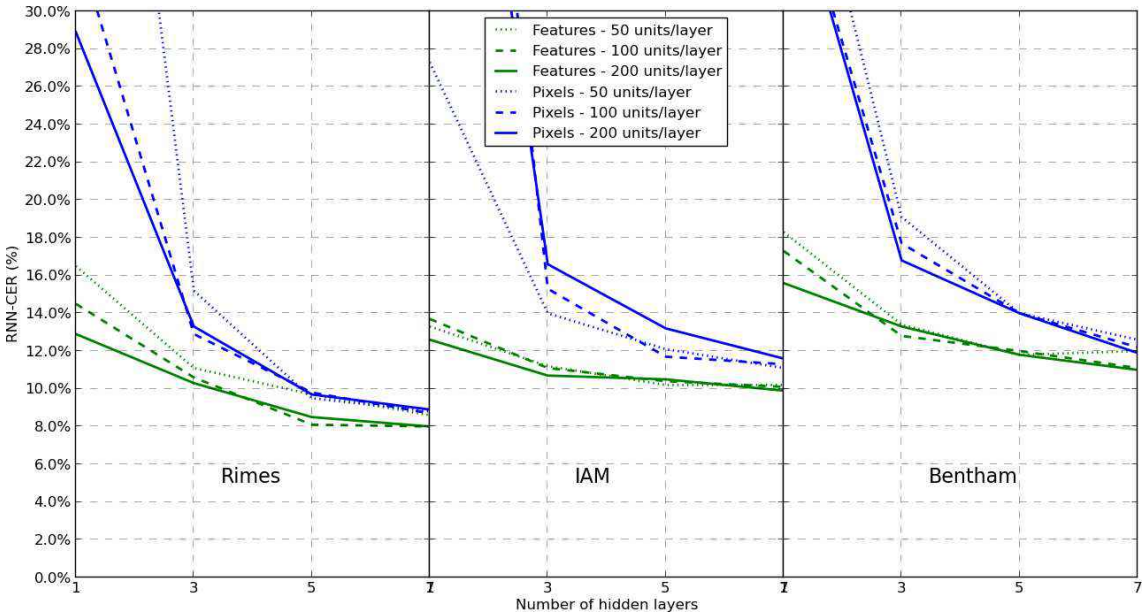


Figure 6.7: Effect of depth vs width on RNN performance (RNN-CER, CTC).

Deeper RNNs with less units yield better CERs than shallower RNNs with more

units, and also more parameters. This observation shows that it is not so much the augmented capacity which causes the improvement, but rather a consequence of the depth itself. We limited ourselves to 200 units per hidden layer, and adding even more units might decrease the error rate of the features RNNs with one hidden layer, until the performance matches that of the deep RNNs one. However, it would also mean a lot more free parameters, and the economy can be made by using deeper and skinnier networks.

6.5.3 Analysis

These experiments show that it is possible to achieve substantial performance gains by training deeper BLSTM-RNNs. When the RNN is trained with CTC to be used alone, deeper is almost always better. In all our experiments, the best results of the full pipeline including the language model were achieved with networks with more than two LSTM layers. Moreover, these recurrent networks require only a few units in each layer to yield good results.

In deep RNNs, the higher LSTM layers see an input sequence which have already been processed by lower recurrent layers. Thus, one may see it as an higher level representation of the whole sequence, possibly including long-term dependencies. This is consistent with what we observed in MLPs, where representations of the input of increasing complexities are computed by successive layers.

As we did for MLPs, we can display the weights of the first layer of RNNs based on pixels as images of the same size as the input frame (Figure 6.8). Unlike MLPs, the input units of LSTM networks also include recurrent connections, but we limited the display to the the input connections. Moreover, the gates also depend on the input pixels, so we also show the weights of the connections between gates and pixels. We observe, at least for the cell input, common shapes of filters in deep learning, which are detectors of oriented edges and blobs. The first layer of deep pixel RNNs seems to effectively extract elementary features.

On Figure 6.9, we show the weights of LSTM cell inputs for pixel RNNs of different depths. We see that when there are more layers in the RNN, the features extracted in the first layer are simpler. When there is only one layer, except for Bentham database, the weights do not look very informative of what is represented in the cells. For Rimes, we can see some structure, which is not elementary like oriented edge detector or Gabor-like filters, but rather complex shapes.

On Figure 6.10, we display the input sensitivity of the output of RNNs trained on pixels for Rimes and Bentham databases, with 1, 3, 5, 7 and 9 hidden layers of 100 units. We see that with one hidden layer, the sensitivity is concentrated in the current window, and does not extend much beyond it. When we increase the number of hidden layers, the prediction may depend on a wider area, including the whole word in some cases.

We have even seen on Figure 6.3b that the sensitivity could even include adjacent words, as if the RNN actually learnt to read words or sequences of words to recognize

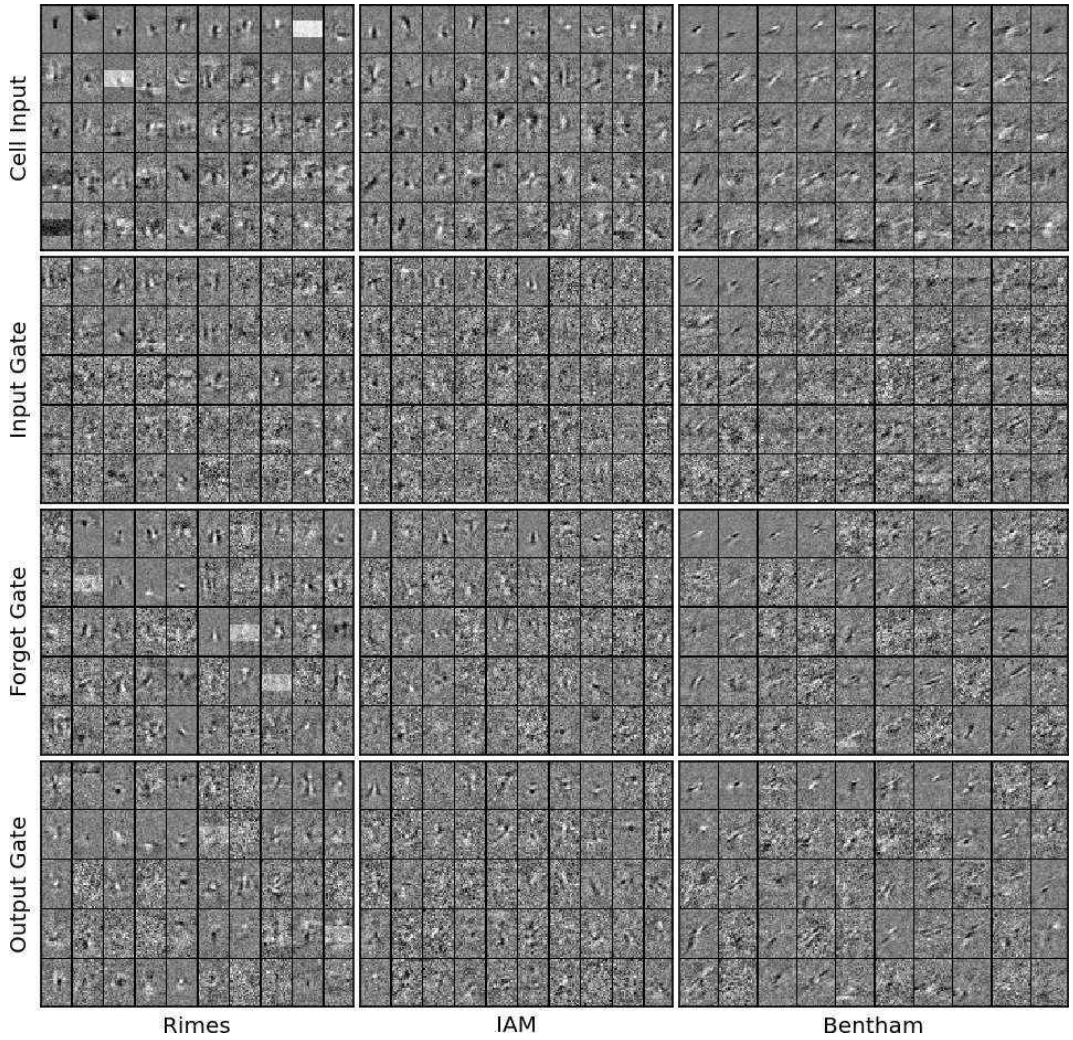


Figure 6.8: Weights of the first layer of pixel LSTM-RNNs (**Pix.**/**Deep**/**CTC**).

well the characters. If the RNN indeed models the characters in the context of the whole word, it would also explain why large improvements in RNN-CER do not lead to improvements when linguistic constraints are added, as we have seen in Table 6.3.

6.6 Study of the Impact of Dropout

The dropout technique (Hinton et al., 2012) is a regularization method for neural networks. It consists in randomly setting some activations from a given hidden layer to zero during training. Wager et al. (2013) described dropout as an adaptive regularizer, and showed that it is equivalent to a first-order L_2 regularization after some transform of the features. Moreover, they obtained better results with dropout than with L_2 regularization on a sentiment analysis classification task using logistic regression.

Pham et al. (2014); Zaremba et al. (2014) recently proposed a way to use dropout in LSTM networks. Pham et al. (2014) carried out experiments on handwritten word and line recognition with MDLSTM-RNNs, and reported relative WER improvements

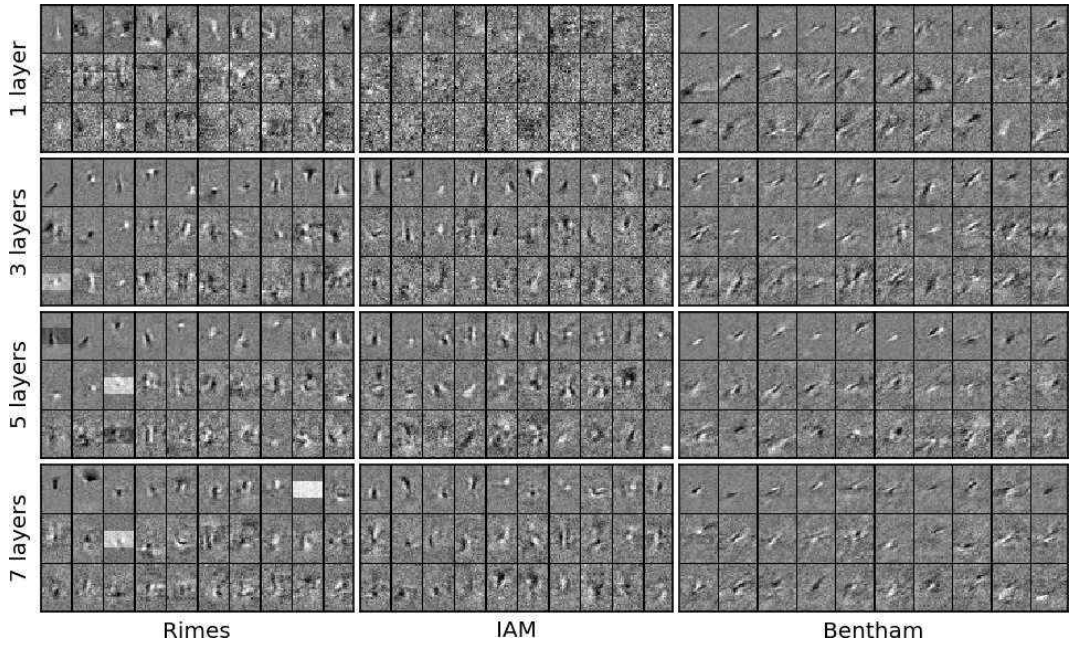
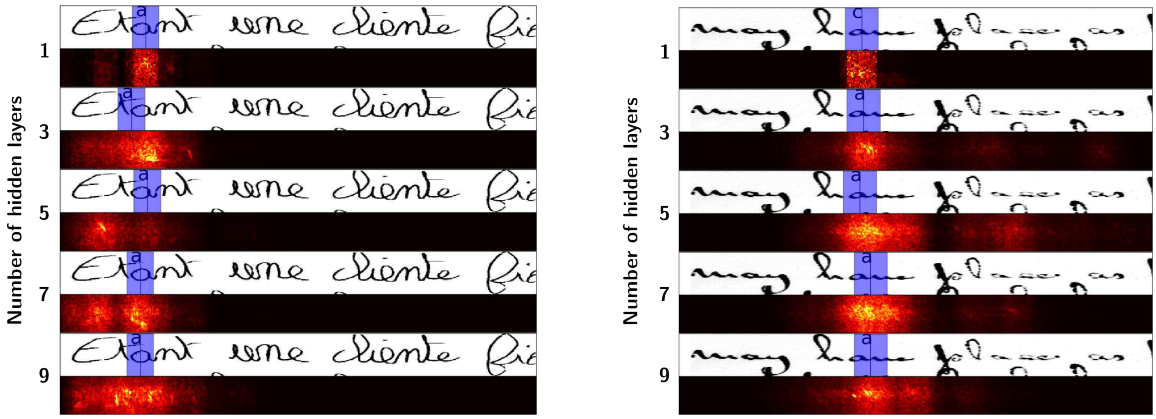


Figure 6.9: Weights of the cell input of the first layer of pixel LSTM-RNNs with different depths. (**Pix.**/**Deep**/**CTC**).



(a) Rimes (b) Bentham

Figure 6.10: Sensitivity maps of LSTM-RNNs of increasing depths (1 to 9; **Pix.**/ **Deep**/ **CTC**).

between 2 and 15% for line recognition with language models, and also observed effects on the classification weights similar to those of L_2 regularization.

The authors proposed to apply dropout with $p = 0.5$ in feed-forward connections following the LSTM layers, and show that in an architecture of three levels of LSTM layers, it is generally better to use dropout after every LSTM layer, rather than after the last one or two only. Here we propose to explore the dropout technique within our deep BLSTM-RNN architecture.

6.6.1 Dropout after the Recurrent Layers

First, we reproduced the dropout experiments of Pham et al. (2014) with BLSTM-RNNs on Rimes, IAM and Bentham databases, with pixel and feature inputs. We built RNNs with seven hidden layers. We applied dropout after LSTM layers with dropping probability $p = 0.5$.

On Table 6.5, we report the CER achieved with RNNs with 7 hidden layers, with 200 units each, and dropout after the top N LSTM layers. We observe that dropout on the topmost layer improves the performance. We get more improvement with dropout on the top N LSTM layers, and the best results are achieved with dropout after all LSTM layers, which is consistent with the findings of Pham et al. (2014) in MDLSTM-RNNs.

Table 6.5: Effect of dropout after the top N LSTM layers (RNN-CER%; **Deep**/ **CTC**).

	Dropout	**None**	**Top-1**	**Top-2**	**Top-3**	**All**
Rimes	Features	8.0	7.5	7.1	6.5	5.7
	Pixels	8.9	7.5	6.9	6.0	6.0
IAM	Feature	10.1	9.1	9.1	8.7	8.1
	Pixels	11.6	11.2	9.2	8.9	7.5

We evaluated the complete systems with RNNs with 100 and 200 hidden units, and with dropout. We observe on Table 6.6 relative WER improvements ranging from 4 to 10%, which is also consistent with the results of Pham et al. (2014).

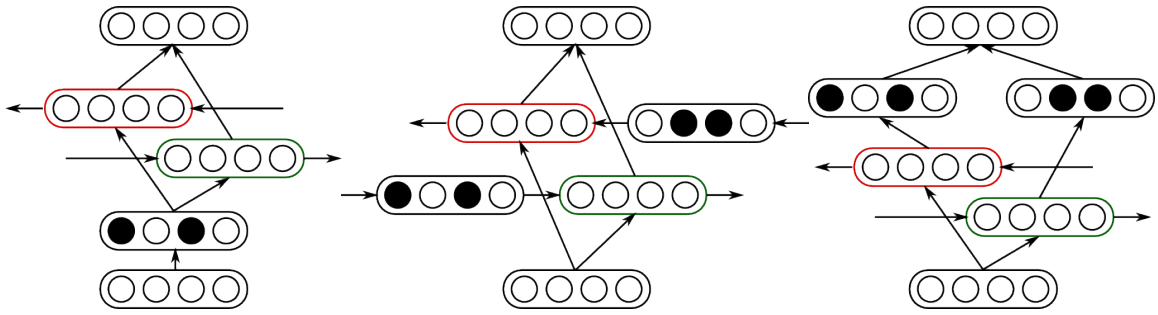
Table 6.6: Effect of dropout in complete pipelines (**Deep**/**CTC**).

		Handcrafted Features			**Pixels**		
		RNN-CER	WER%	CER%	RNN-CER	WER%	CER%
Rimes	7x100	8.0	13.8	4.1	8.7	14.5	4.5
	7x200	8.0	14.1	4.1	8.9	14.7	5.0
	+drop.	5.7	12.7	3.6	6.0	13.6	4.1
IAM	7x100	9.9	13.0	4.3	11.3	13.6	5.2
	7x200	10.1	12.9	4.6	11.6	14.6	5.5
	+drop.	8.1	11.9	3.9	7.5	11.8	4.0
Benth.	7x100	11.1	18.5	7.5	12.2	21.4	8.8
	7x200	11.0	18.0	7.0	11.9	20.6	8.4
	+drop.	8.9	17.2	6.7	9.2	18.7	7.3

6.6.2 Dropout at Different Positions

In (Pham et al., 2014), dropout is used only after the LSTM layers, so as not to affect the recurrent connections. In practice, we can apply it between any two layers. We complemented the previous experiments by using dropout at different positions, depicted on Figure 6.11, that is either before the LSTM layer, after it, or in the

recurrent connections. Moreover, instead of comparing the improvements by increasing the number of top N layers affected by dropout, we studied the effect of dropout in different layers in isolation.



(a) Before (b) Inside (c) After

Figure 6.11: Dropout position in LSTM-RNNs.

The chosen architecture for this set of experiments consists of five hidden layers, with 200 hidden units each. We compare the effect of the position of dropout with respect to the LSTM layer: before, inside or after, and to the layers in which it is applied: none, bottom, middle, top, or all. The experiments were carried out on Rime, IAM and Bentham.

Table 6.7: Effect of dropout at different positions on Rimes database (RNN-CER%; **Deep**/**CTC**).

	Dropout	Before	Inside	After	All
Rimes	None	8.2			
(Features)	Bottom	6.8	6.7	*8.2*	**6.7**
	Middle	**6.6**	7.3	7.2	6.8
	Top	7.4	*8.6*	8.0	*8.8*
	All	**5.0**	5.4	6.8	7.1
	None	9.7			
(Pixels)	Bottom	**7.1**	7.6	8.1	**7.2**
	Middle	7.5	9.6	9.0	8.8
	Top	8.0	9.2	7.8	9.1
	All	**5.8**	6.0	6.5	7.4

The RNN-CER results are reported on Table 6.7 (Rimes), Table 6.8 (IAM) and Table 6.9 (Bentham). Bold numbers are the best results for dropout at a single position in a single layer, at a single position in all layers, and at all positions in a single layer. The few cases when dropout did not decrease, or increased the error rate are signaled in italics.

Besides the fact that dropout almost always helps, we can draw several conclusions. When dropout is **only applied at one position**:

Table 6.8: Effect of dropout at different positions on IAM database (RNN-CER%; **Deep**/**CTC**).

	Dropout	Before	Inside	After	All
IAM	None	10.4			
(Features)	Bottom	9.1	**8.5**	9.8	8.8
	Middle	8.9	9.1	8.6	**8.7**
	Top	9.1	10.2	9.5	*10.4*
	All	7.9	**7.0**	9.0	9.4
	None	13.2			
(Pixels)	Bottom	10.0	**9.1**	11.4	**10.1**
	Middle	10.1	11.1	10.6	10.8
	Top	10.9	12.3	11.1	12.6
	All	8.6	**8.4**	10.1	11.4

Table 6.9: Effect of dropout at different positions on Bentham database (RNN-CER%; **Deep**/ **CTC**).

	Dropout	Before	Inside	After	All
Bentham	None	11.0			
(Features)	Bottom	**8.5**	9.9	12.3	**8.8**
	Middle	9.8	9.9	10.4	10.0
	Top	10.5	*11.2*	10.7	*12.3*
	All	**7.4**	8.1	10.0	8.5
	None	14.0			
(Pixels)	Bottom	10.4	**9.9**	13.4	**9.7**
	Middle	11.0	13.6	12.2	13.0
	Top	12.0	*15.1*	12.7	*14.4*
	All	**8.0**	9.4	10.8	12.3

- it is generally better in lower layers of the RNNs, rather than in the top LSTMs, except when it is after the LSTM.
- it is almost always better before the LSTM layer than inside or after it, and better after than inside, except for the bottom layer.

When it is **applied in all layers** (bottom, middle and top):

- among all relative positions to the LSTM, when dropout is applied to every LSTM, placing it after was the worst choice in all six configurations
- before LSTMs seems to be the best choice for Rimes and Bentham, and inside LSTMs is better for IAM.

Moreover, adding dropout **before, inside and after the LSTM at the same time** is not as good as choosing the right position.

For the position before the LSTM layer (Figure 6.11a), there are actually two possibilities. The first one, chosen for the results presented in this section, is to drop the same inputs for both directions in the LSTM. The other option is to sample a different set of inputs to drop for the forward and backward LSTM. We explored both methods, and got exactly the same results.

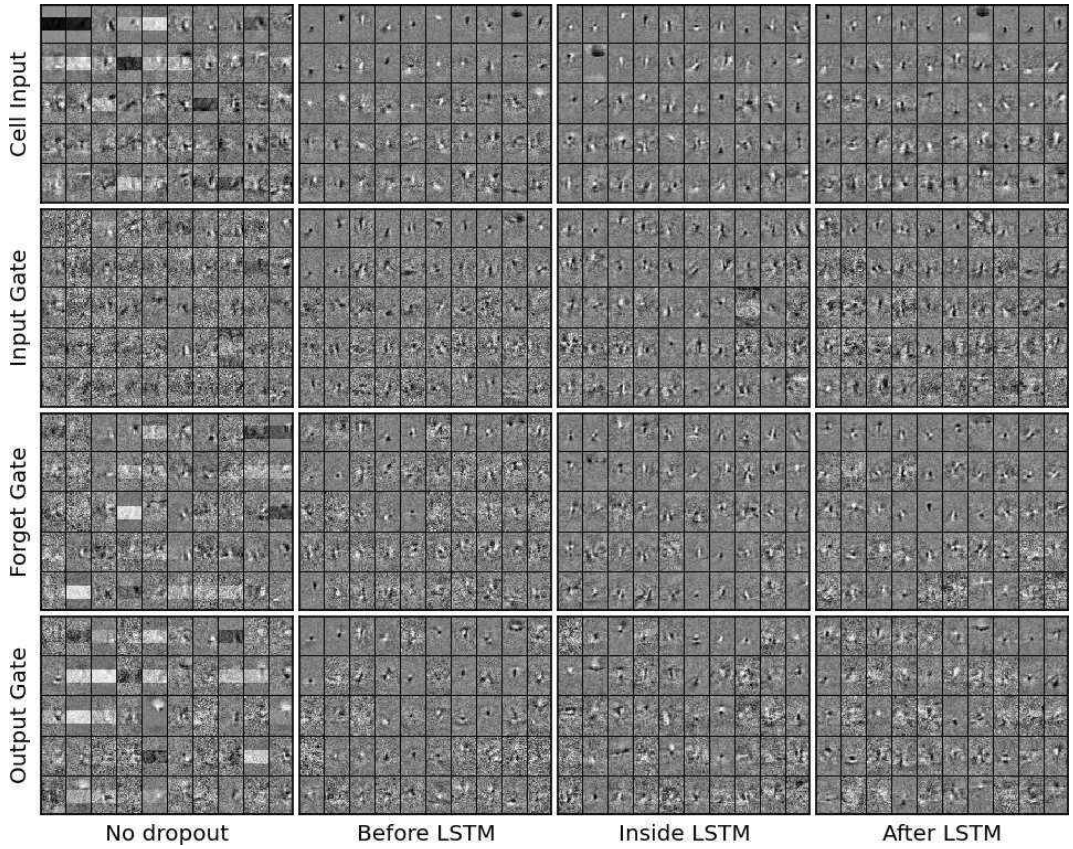


Figure 6.12: Weights of the first layer of pixel LSTM-RNNs with different dropout strategies on Rimes database (**Pix.**/**Deep**/**CTC**).

On Figure 6.12, we display the weights of the connections between the inputs and the first LSTM layer, including the gates, without and with dropout at different positions, on Rimes database, with pixel inputs. The recurrent connections are not shown. We notice that the filters are generally sharper with dropout, as if this regularization technique improved the selectivity of the cells, making them more focused on elementary features. This is especially visible for the weights of the gates. Note that the block artefacts that appear in the filters of the RNN without dropout is certainly caused by our three-zone height normalization. Some units seem to have specialized in the detection of the presence or absence of black pixels anywhere in the core region, which is generally sufficient to detect whitespace, for example, ignoring the possible descenders of the line above, or ascenders of the line below.

The values of neighboring pixels are highly correlated. If the model can always access one pixel, it might be sufficient to infer the values of neighboring ones, and the

weights will be used to model more complicated correlations. However, with dropout on the inputs, the local correlations are less visible. With half the pixels missing, the model cannot rely on regularities in the input signal, and should model them to make the most of each pixel. This may explain why the filters represent some local, elementary components, which capture the correlations between neighboring pixels.

6.6.3 Study of the Effect of Dropout in Complete Systems (with LM)

Table 6.10: Effect of dropout at different positions in complete pipelines (**Deep**/**CTC**).

		Handcrafted Features			Pixels		
Rimes		RNN-CER (%)	WER (%)	CER (%)	RNN-CER (%)	WER (%)	CER (%)
5 hidden layers	no dropout	8.2	12.9	3.7	9.7	15.5	4.8
	after	6.8	**12.8**	**3.6**	6.5	**13.3**	4.1
	inside	5.4	13.2	3.8	6.0	14.3	4.6
	before	**5.0**	13.1	3.7	**5.8**	13.8	**4.0**
7 hidden layers	no dropout	8.0	14.1	4.1	8.9	14.7	5.0
	after	5.7	**12.7**	**3.6**	6.0	**13.6**	**4.1**
	inside	5.3	**12.7**	3.7	5.9	14.2	4.6
	before	**4.8**	**12.7**	3.7	**5.3**	13.7	4.2
IAM							
5 hidden layers	no dropout	10.4	11.7	4.0	13.2	14.7	5.7
	after	9.0	11.8	4.1	10.1	13.2	4.7
	inside	**7.0**	**11.6**	**3.9**	**8.4**	13.3	5.0
	before	7.9	12.3	4.2	8.6	**12.4**	**4.5**
7 hidden layers	no dropout	10.1	12.9	4.6	11.6	14.6	5.5
	after	8.1	11.9	**3.9**	**7.5**	**11.8**	**4.0**
	inside	**7.1**	11.9	4.1	7.9	13.0	4.7
	before	7.4	**11.7**	4.1	8.3	13.2	4.8
Bentham							
5 hidden layers	no dropout	11.0	18.1	7.0	14.0	21.3	9.0
	after	10.0	17.3	6.9	10.8	19.1	7.7
	inside	8.1	17.7	6.8	9.4	20.0	8.5
	before	**7.4**	**16.6**	**6.2**	**8.0**	**17.8**	**6.9**
7 hidden layers	no dropout	11.0	18.0	7.0	11.9	20.6	8.4
	after	8.9	17.2	6.7	8.9	18.7	7.3
	inside	7.1	17.4	6.5	8.7	20.1	8.4
	before	**6.5**	**16.7**	**6.1**	**7.5**	**17.7**	**6.4**

On Table 6.10, we report the results of different dropout strategies in the complete systems including the language models, with 5 and 7 hidden layers. Dropout is applied

in every LSTM layer, at the different positions relative to it. The baselines are RNNs without dropout, and with dropout after LSTM, as proposed by Pham et al. (2014).

We observe that for Rimes, the best results are achieved with dropout after LSTMs, despite the superior performance of dropout before for the RNN alone. For IAM, dropout inside LSTMs is only slightly better. On Bentham database, dropout before LSTMs consistenly yields lower error rates than other solutions. The lack of a strong correlation between the performance of the network alone, and of the complete system was already noticed in previous sections, as well as in the previous chapter.

The main difference between the RNN alone and the complete system is that the former only considers the best character hypothesis at each timestep, whereas the latter potentially considers all predictions in the search of the best transcription, with linguistic constraints. Therefore, applying dropout after the LSTM in the top layer(s) might be beneficial for the beam search in the decoding with complete systems. Indeed, dropout after the last LSTMs forces the classification to rely on more units. Conversely, a given LSTM unit will contribute to the prediction of more labels. Thus, the classification will be more robust, because it relies on more evidence, and might keep competing character hypotheses in a closer range, leaving more room for error correction in the beam search. Moreover, the weight halving at decoding time stabilizes the response of the network to small changes to the LSTM outputs. On the other hand, dropout on the inputs of the LSTM layer prevents it to rely on their correlations, as explained in the previous section.

In Table 6.11, we report the results when the different choices of dropout positions (before or after) are combined. More specifically, we apply dropout before LSTMs in lower layers, and after LSTMs in upper layers. We see that for the RNN alone (inside parentheses), dropout before some LSTMs is always better than dropout after all LSTMs, but not necessarily much better than dropout before all LSTMs and no dropout after. On the other hand, for complete systems, adding dropout after the top LSTMs nearly always improves the results over dropout *before* only, and the best combination of *before* and *after* always outperforms both dropout before all and after all. The optimal combination seems to be before the first two and after the last LSTM for features, and before and after all LSTMs for pixels.

6.7 Study of the Choice of Inputs

In this section, we focus on the differences in the performance of RNNs with feature and pixel inputs. As we can see on Figure 6.6, as models are deeper, the difference between features and pixels decreases. With 11 hidden layers the RNN-CER is less than 1% higher for pixels.

On Table 6.12, we see that when we select the best deep BLSTM-RNN for each type of inputs, the relative difference between features and pixels is around 10%, while it is above 60% with only one layer. We also notice that deep pixel RNNs yield comparable results as feature RNNs with only one hidden layer. When moreover dropout is applied

Table 6.11: Effect of dropout at different combinations of positions in complete pipelines (**Deep**/**CTC**).

		Handcrafted Features			Pixels		
Rimes		RNN-CER (%)	WER (%)	CER (%)	RNN-CER (%)	WER (%)	CER (%)
5 hidden layers	after all	6.8	12.8	**3.6**	6.5	13.3	4.1
	before all	**5.0**	13.1	3.7	5.8	13.8	4.0
	bef. 1 / aft. 2-3	5.5	12.8	**3.6**	6.3	13.5	4.0
	bef. 1-2 / aft. 3	5.6	**12.7**	**3.6**	6.0	13.7	4.2
	bef.+aft. all	5.4	**12.7**	3.7	**5.3**	**12.7**	**3.9**
7 hidden layers	after all	5.5	**12.7**	**3.6**	6.0	13.6	4.1
	before all	4.8	**12.7**	3.7	**5.3**	13.7	4.2
	bef. 1-2 / aft. 3-4	5.3	**12.7**	3.7	6.2	13.6	4.1
	bef. 1-2-3 / aft. 4	5.1	13.3	3.8	5.9	13.6	4.1
	bef.+aft. all				5.6	13.7	4.2
IAM							
5 hidden layers	after all	9.0	11.8	4.1	10.1	13.2	4.7
	before all	7.9	12.3	4.2	8.6	12.4	4.5
	bef. 1 / aft. 2-3	8.2	11.6	4.0	8.0	11.9	4.1
	bef. 1-2 / aft. 3	8.1	**11.2**	**3.8**	8.3	11.8	4.2
	bef.+aft. all	7.8	12.2	4.1	7.9	11.6	4.1
7 hidden layers	after all	8.1	11.9	3.9	**7.5**	**11.4**	**3.9**
	before all	**7.4**	11.7	4.1	8.3	13.2	4.8
	bef. 1-2 / aft. 3-4	8.0	11.5	3.9	7.9	11.6	4.0
	bef. 1-2-3 / aft. 4	7.5	11.6	3.9	8.2	12.3	4.2
	bef.+aft. all				8.1	13.3	4.5
Bentham							
5 hidden layers	after all	10.0	17.3	6.9	10.8	19.1	7.7
	before all	7.4	16.6	6.2	8.0	17.8	6.9
	bef. 1 / aft. 2-3	7.1	16.1	5.8	8.4	17.6	6.7
	bef. 1-2 / aft. 3	7.4	**16.0**	6.0	8.7	18.1	6.7
	bef.+aft. all	7.3	17.1	6.3	7.5	17.5	6.7
7 hidden layers	after all	8.9	17.2	6.7	8.9	18.7	7.3
	before all	**6.5**	16.7	6.1	7.5	17.7	6.4
	bef. 1-2 / aft. 3-4	6.7	16.1	5.8	**7.1**	**17.0**	**6.2**
	bef. 1-2-3 / aft. 4	6.7	16.3	**5.7**	7.3	17.6	6.4
	bef.+aft. all				**7.1**	17.7	6.5

during the training of the network, the gap of performance between pixels and features decreases even more. There is finally no difference on Rimes and IAM between both kinds of inputs, and only 1% absolute difference on Bentham.

Table 6.12: Comparison of WERs (%) obtained with handcrafted features and pixel values.

	Features	**Pixels**	
Rimes			
1-layer BLSTM-RNN	14.9%	24.1%	(+65.1%)
Deep BLSTM-RNN	12.9%	14.0%	(+7.9%)
+dropout	12.7%	12.7%	(+0.0%)
IAM			
1-layer BLSTM-RNN	13.4%	83.2%	(+520.0%)
Deep BLSTM-RNN	11.4%	12.8%	(+12.3%)
+dropout	11.2%	11.4%	(+0.9%)
Bentham			
1-layer BLSTM-RNN	20.6%	33.8%	(+64.1%)
Deep BLSTM-RNN	18.0%	20.3%	(+12.8%)
+dropout	16.0%	17.0%	(+6.3%)

These results show that with deep RNNs, pixels might be sufficient. Instead of handcrafted features, one can rely on the automatic extraction of features relevant to the task, and learnt by the RNN during training. Note also that we did not experimented several sizes of sliding window for pixels. Regarding the results presented in Section 6.3, we could expect to see some improvements by carefully tuning the width of the window of pixels.

6.8 Conclusion

In this chapter, we have studied BLSTM-RNNs for handwriting recognition on Rimes, IAM and Bentham databases, using handcrafted features and pixels. We have built deep RNNs, with up to 13 hidden layers.

We have seen that ***explicitly including context in the observation sequences did not improve the results***, and that RNNs could effectively learn the dependencies in the input sequences, and the context necessary to make character predictions. The observations suggested that in some cases, the context of the whole word was relevant to a single character prediction.

Then, we have shown that the ***recurrence was especially useful in the top layers of RNNs*** in the CTC framework. The bottom layers, particularly with raw inputs such as pixel values, are more focused on the extraction of elementary features. Recurrent layers in these cases may not be crucial.

We have also demonstrated that ***significant improvements are brought by increasing the depth of RNNs***, adding many hidden layers. We limited the ex-

periments to up to thirteen hidden layers. We recorded relative CER improvements between 35 and 50% for features and between 70 and 80% for pixels, from RNNs with a single layer to deep RNNs, when used alone. In the complete pipeline including the linguistic constraints, the WER improvements are not as impressive, probably because depth help the RNNs to correct mistakes that are also corrected by the language model, but still range from 10 to 15% for handcrafted features and around 40% for pixels.

Moreover, we confirmed that the dropout technique was very efficient to get better RNN results, and almost always improved the CER of RNNs, wherever it was applied. We studied different positions for dropout in the networks, in particular its relative position to LSTM layers, and reported significant ***improvements over the method presented in (Pham et al., 2014) when dropout is applied before LSTM layers*** rather than after them. Yet, in complete systems, when the outputs of the RNNs are used in a decoder with linguistic constraints, ***dropout on the classification weights is still very important***.

Finally, ***the discrepancy between the performance of the systems with handcrafted feature and pixel inputs is largely decreased with deep RNNs***. As for deep MLPs, the need to design and implement good feature extraction may not be necessary. Deep networks automatically learn features from the image, without making a priori assumptions about what is relevant to the task.

We did not experiment many sizes of sliding windows for pixels, and kept the same ones as for MLPs. The extracted frames are about the same size as characters in the image, while the sliding window for features is much smaller. Considering the results of Section 6.3, we might expect further improvements with the pixel systems with smaller windows, and to bridge the remaining gap between the two kinds of inputs.

It is also worth pointing out that, as Convolutional Neural Networks are a good alternative to MLPs in Computer Vision problems (and are generally preferred to them), MDLSTM-RNNs are an excellent alternative to BLSTM-RNNs for handwriting recognition (Bluche et al., 2014a; Moysset et al., 2014; Pham et al., 2014), and also work with raw pixel values, their inputs being images. They were not presented in this thesis, and left for future investigation.

Part IV

COMPARISON AND COMBINATION OF DEEP MLPs AND RNNs

Chapter 7

Experimental Comparison of Framewise and CTC Training

Contents

7.1 Introduction 165

7.2 Experimental Setup 167

7.3 Relation between CTC and Forward-Backward Training of Hybrid NN/ HMMs 167

7.3.1 Notations 167

7.3.2 The Equations of Forward-Backward Training of Hybrid NN/ HMMs 168

7.3.3 The Equations of CTC Training of RNNs 169

7.3.4 Similarities between CTC and hybrid NN/HMM Training 171

7.4 Topology and Blank 172

7.5 CTC Training of MLPs 173

7.6 Framewise vs CTC Training 174

7.7 Interaction between CTC Training and the Blank Symbol . . 176

7.7.1 Peaks 176

7.7.2 Trying to avoid the Peaks of Predictions 179

7.7.3 The advantages of prediction peaks 181

7.8 CTC Training without Blanks 182

7.9 The Role of the Blank Symbol 183

7.10 Conclusion 183

7.1 Introduction

In the previous part, we built hybrid systems based on deep neural networks. The approach was different for MLPs and RNNs, mainly because we chose the Connectionist Temporal Classification framework (Graves et al., 2006; Graves, 2012) for RNNs, which has proven successful for handwriting recognition is the past years. More precisely:

- for ***MLPs***, we trained the networks with a **framewise** criterion, with **HMM states** targets, obtained through a **forced alignment** of the training data with the GMM/HMM systems of Section 4.5
- for ***RNNs***, we trained the networks with the **CTC** criterion, which considers all possible segmentations of **character sequences**, and does not require a bootstrapping system.

The main difference in the final hybrids is that for MLPs, we use the HMMs of the GMM/HMM systems, with 5 or 6 states, while for RNNs, we build an HMM to match the output of the CTC-trained networks, with one state per character, and a one-state blank HMM. Less states per character give several advantages. First, there is less classes to separate, and more training examples per class, which may be beneficial for the neural network. Moreover, the HMM are smaller, hence the decoding graph has less arcs, and requires less memory, as illustrated on the following table:

Decoding graph size		**MLP/HMM**	**RNN/HMM**
Rimes	Size (MB)	20	11
	# arcs	925k	546k
	# states	427k	188k
IAM	Size (MB)	357	180
	# arcs	17.5M	9.6M
	# states	7.8M	2.9M
Bentham	Size (MB)	35	17
	# arcs	1,7M	877k
	# states	775k	287k

It may be interesting to see if this particular topology, with only a few states, gives better results for hybrid systems. Furthermore, the non-character symbol of CTC is also found in (Tay et al., 2001; Rashid et al., 2012) for MLPs, and might be useful to model inter-characters.

The CTC training algorithm consists of a forward-backward computation in a graph representing all possible segmentations of the output sequence. This is reminiscent of the Baum-Welch training of hidden Markov models.

CTC is not the first algorithm proposed to train neural networks with unsegmented data. Researchers have investigated forward-backward training of hybrid systems, to include both the NN and HMM in the training procedure, and to avoid the need for a bootstrapping system (Senior & Robinson, 1996; Hennebert et al., 1997; Yan

et al., 1997). On Table 7.1, we report some of the improvements observed in the past with a forward-backward training of NNs. Forward-backward procedures are also found in sequence-discriminative training methods (Bengio et al., 1992; Haffner, 1993; Kingsbury, 2009; Veselý et al., 2013).

Table 7.1: WER improvements with forward-backward training of neural networks

Author	System	Training	Test	Framewise	FwdBwd
Senior & Robinson (1996)	RNN	N/A (25p. of text)		17.0%	15.4%
Yan et al. (1997)	MLP	2,090 utt.	500 utt.	4.1%	3.1%
	-		1,600 utt.	6.0%	4.9%
Hennebert et al. (1997)	MLP	9,000 utt.	2,000 utt.	13.7%	12.2%
	-	19,000 utt.	7,000 utt.	9.8%	10.1%

This chapter is a study of the CTC paradigm, in comparison to framewise training methods. We are interested in answering the following questions:

- Apart from the outputs consisting of one label for each character and a blank symbol, *what are the differences between the CTC training algorithm and the forward-backward training proposed in the nineties?*
- *How important is the number of HMM states in hybrid systems? Would it be better to have fewer states in MLP/HMMs, or more states in RNN/HMMs?*
- *Is the non-character symbol useful for handwriting recognition? In which conditions (training method, NN type) does it help?*
- *What is the role of this blank symbol?*
- *Can CTC training be applied to MLP? Does it improve the results?*
- *How CTC training compares to framewise training for RNNs.*

First, in Section 7.2, we present the experimental setup for this chapter. We are interested in the training methods and the output space of neural networks, and their integration in the hybrid NN/HMM framework. Therefore, we built smaller neural networks, in order to have quicker trainings.

In Section 7.3, we show that the equations of CTC are very similar to those of forward-backward training of neural networks, and we point out the differences.

Then, in Section 7.4, we study the influence of the number of HMM states on the performance of the neural networks in the hybrid NN/HMM integration, as well as the importance of adding a non-character (blank) model between character models.

In Section 7.5, we apply the CTC training algorithm to MLPs, either with the standard HMM topology, or with the topology defined in the CTC paradigm, with one NN output for each character, and a blank output.

We compare framewise training and CTC training of both MLPs and RNNs in Section 7.6, varying the number of states and the presence of the blank symbol. We show that the CTC framework is particularly suited for RNNs.

In Section 7.7, we study the interaction between the blank symbol and the CTC training algorithm, and underline the advantages of observing peaks of character predictions at localized timesteps, surrounded by many blank predictions. In Section 7.8, we observe what happens when the blank is not present in CTC, and suppose that this special symbol helps in the alignment procedure at early stages of training.

Finally, we investigate the role of the blank symbol, for training and modeling, in Section 7.9. We stress its interaction with CTC training, from which peaks of non-blank prediction emerge, and underline one more time that blank, CTC and RNNs work especially well together.

We conclude this chapter with a discussion of our findings in Section 7.10.

7.2 Experimental Setup

In this chapter, we trained GMMs, MLPs and RNNs with handcrafted features on the IAM database.

The training of GMMs follows the standard EM procedure, realigning the data with Viterbi algorithm and and updating the mixtures to maximize the data likelihood, increasing the number of Gaussian at each step, until no more improvement is observed in the validation data. We chose neural network architectures to be a tradeoff between model size (training time) and performance. The goal here is not to compare the absolute results of MLPs and RNNs, or to determine the exact optimal number of states, but rather to study, evaluate, and compare the trends when we vary the topology and the training procedure. Because of time constraints, we limited ourselves to relatively small neural networks in this chapter.

The MLPs have two hidden layers of 1,024 sigmoid units. The RNNs have one hidden layer with 100 units, made of an LSTM layer for each direction (BLSTM-RNN). The MLPs take 11 consecutive frames of 56-dimensional feature vectors to take into account the context. The RNNs directly consider the initial sequence of feature vectors.

In framewise training, the optimized criterion is the classification cross-entropy. For framewise training and CTC, we used early stopping: training stops when the cost on the validation data does not decrease for 20 epochs, and the best network is selected. All networks were trained with the software implementation of A2iA.

7.3 Relation between CTC and Forward-Backward Training of Hybrid NN/ HMMs

7.3.1 Notations

Let $\mathcal{Q}_n(\mathbf{W})$ be the set of all state sequences of length n representing some word sequence $\mathbf{W}$. For notation convenience, we also define $\mathcal{Q}(\mathbf{W})$, the list of all states in the model of $\mathbf{W}$ (a sequence in $\mathcal{Q}_n(\mathbf{W})$ is made of elements of $\mathcal{Q}(\mathbf{W})$). Since in

practice the same state can be used several times (and appear several times in $\mathcal{Q}(\mathbf{W})$ – *e.g.* the first state of the HMM for character *"e"* in word *"tee"*), we define $\mathcal{Q}^*$ the set of all distinct states, and a mapping $\mu : \mathcal{Q}(\mathbf{W}) \mapsto \mathcal{Q}^*$ which provides the identity of a considered state. This is needed since HMMs have a single emission model for each element of $\mathcal{Q}^*$, when the forward-backward algorithms often consider elements of $\mathcal{Q}(\mathbf{W})$.

7.3.2 The Equations of Forward-Backward Training of Hybrid NN/ HMMs

The basic idea of forward-backward training of neural networks is to use the scaled neural network posterior in the HMM formulation:

$$p(\mathbf{x}|\mathbf{W}) = \left(\prod_{t=1}^{|\mathbf{x}|} p(x_t)\right) \sum_{\mathbf{q}\in\mathcal{Q}_{|\mathbf{x}|}(\mathbf{W})} p(q_1, \mathbf{W}) \frac{p(q_1|x_1)}{p(q_1)} \prod_{t=2}^{|\mathbf{x}|} \frac{p(q_t|x_t)}{p(q_t)} p(q_t|q_{t-1}, \mathbf{W}) \tag{7.1}$$

and to apply the same forward-backward procedure in the HMM as for Baum-Welch training, with forward and backward variables:

$$\begin{aligned} \alpha_t(s) &= p(x_{1:t}, q_t = s|\mathbf{W}) \\ \beta_t(s) &= p(x_{t+1:T}|q_t = s, x_{1:t}, \mathbf{W}) \end{aligned}$$

where $s \in \mathcal{Q}(\mathbf{W})$. These variables are computed iteratively, and the following recurrence equations apply with the neural network posteriors:

$$\alpha_t(s) = \frac{p(q_t = s|x_t)}{p(s)} \times \sum_{r\in\mathcal{Q}(\mathbf{W})} \alpha_{t-1}(r) p(q_t = s|q_{t-1} = r, \mathbf{W})$$

$$\beta_t(s) = \sum_{r\in\mathcal{Q}(\mathbf{W})} \frac{p(q_{t+1} = r|x_{t+1})}{p(r)} p(q_{t+1} = r|q_t = s, \mathbf{W}) \beta_{t+1}(r)$$

Since $\frac{p(q|x)}{p(q)} = \frac{p(x|q)}{p(x)}$, this forward-backward procedure actually computes

$$\frac{p(\mathbf{x}|\mathbf{W})}{\prod_{t=1}^{|\mathbf{x}|} p(x_t)} = \sum_{s\in\mathcal{Q}(\mathbf{W})} \alpha_t(s)\beta_t(s) \tag{7.2}$$

and we can derive the state posteriors given the word sequence $\mathbf{W}$:

$$p(q_t = s \in \mathcal{Q}(\mathbf{W})|\mathbf{x}, \mathbf{W}) = \frac{\alpha_t(s)\beta_t(s)}{\sum_r \alpha_t(r)\beta_t(r)}$$

Summing over all occurrences of a given state in the word sequence HMM, we get the posterior probability of a state given the observation and HMM:

$$p(q_t = k \in \mathcal{Q}^*|\mathbf{x}, \mathbf{W}) = \sum_{s:\mu(s)=k} p(q_t = s|\mathbf{x}, \mathbf{W}) \tag{7.3}$$

At every time t, the neural network computes a posterior distribution over elements of $\mathcal{Q}^*$, and thus we can use the distribution computed with Equation 7.3 as the target distribution in the cross-entropy training criterion. The backpropagated error is

$$\frac{\partial E}{\partial a_k^t} = y_k^t - p(q_t = k \in \mathcal{Q}^* | \mathbf{x}, \mathbf{W}) = y_k^t - \sum_{s:\mu(s)=k} \frac{\alpha_t(s)\beta_t(s)}{\sum_r \alpha_t(r)\beta_t(r)} \tag{7.4}$$

where y_k^t is the output of the neural network at time t for class k, *i.e.* $p(q_t = k|x_t)$, and a_k^t are the activations before the softmax.

Several papers about forward-backward training of neural networks were published. The idea is to replace the Viterbi segmentation of the input sequence, where the classification targets are HMM states, by considering all possible segmentations, as it is done in the Baum-Welch algorithm for HMMs. In some works, such as (Senior & Robinson, 1996; Yan et al., 1997), it is assumed that $\frac{p(q|x)}{p(q)} \propto p(x|q)$, so Eqns. 7.1 and 7.2 both compute $p(\mathbf{x}|\mathbf{W})$. The "soft" targets are then obtained with Equation 7.3.

In (Hennebert et al., 1997), the same recurrences on α and β compute $\frac{p(\mathbf{x}|\mathbf{W})}{\prod_t p(x_t)}$, and posteriors are again obtained with Equation 7.3. The original goal of the work of Hennebert et al. (1997) was to optimize directly $p(\mathbf{W}|\mathbf{x})$, in which we are interested for decoding. Some assumptions, such as the common limitation of the dependency of the HMM state q_t to the current observation (or some local context) given the previous state ($p(q_t|\mathbf{x}, q_{t-1}) = p(q_t|x_t, q_{t-1})$) leads to the REMAP formulation and special kind of neural network (Konig et al., 1996). When furthermore the dependency on the previous state is dropped, given the current observation, we obtain Equation 7.3, but the equations do not exactly compute $p(\mathbf{W}|\mathbf{x})$. Earlier, other papers used a forward-backward procedure to train the NN with a Maximum Mutual Information criterion (Bengio et al., 1992; Haffner, 1993).

The idea of these works is to replace hard targets by soft ones, which take into account different possible segmentations of the input. After a first training with Viterbi alignments, the targets are re-estimated with the forward-backward procedure, and the network is trained with the obtained posterior probabilities over classes for each frame. We can show (simple derivations of the expression of the posteriors in terms of α and β), that when the cross-entropy is the training criterion, and the re-estimation are made after each batch (epoch or training sequence), the method of (Senior & Robinson, 1996; Yan et al., 1997) is equivalent to training the network with the negative log-likelihood $-\log p(\mathbf{x}|\mathbf{W})$.

7.3.3 The Equations of CTC Training of RNNs

The goal of CTC (Graves et al., 2006) is to use a neural network to transform an input sequence $\mathbf{x}$ into a (shorter) output sequence of labels $\mathbf{L}$ (*e.g.* a sequence of characters) using the NN predictions with no complicated post-processing. The proposed method defines the NN outputs to be the set of possible labels, plus a *blank* output ($\oslash$). This

way, a mapping transforms sequence of predictions into the target sequence of labels by first removing repetitions of labels, and then blanks. For example:

$$a\ a \oslash \oslash b\ b \oslash ba \mapsto abba$$

One of the motivation for the special blank label is to be able to use this simple mapping function and still output two consecutive identical labels in the output sequence (Graves et al., 2006). The authors also suggests that this label should model everything in between the relevant part of the input sequences, such as the connections or short whitespaces between characters in an image. A blank prediction is only mandatory between two consecutive and identical labels, and optional between other labels.

With this labeling problem at hand, the NN is trained to minimize the probability of the label sequence given the input sequence ($-\log p(\mathbf{L}|\mathbf{x})$). Several prediction sequences yield the same label sequence (*e.g.* $a\ a\ b\ b$, $a\ a\ a\ b$, $a \oslash b\ b$, ...). To simplify the analogy with the methods presented previously, let $\mathcal{Q}_n(\mathbf{L})$ be the set of all label (prediction) sequences mapping to $\mathbf{L}$. Thus

$$p(\mathbf{L}|\mathbf{x}) = \sum_{\mathbf{q} \in \mathcal{Q}_{|\mathbf{x}|}(\mathbf{L})} p(\mathbf{q}|\mathbf{x})$$

In (Graves et al., 2006), the authors assume that the predictions made at different timesteps are independent given the observation sequence, hence

$$p(\mathbf{L}|\mathbf{x}) = \sum_{\mathbf{q} \in \mathcal{Q}_{|\mathbf{x}|}(\mathbf{L})} \prod_{t=1}^{|\mathbf{x}|} p(q_t|\mathbf{x})$$

This quantity can also be efficiently computed with a forward-backward procedure. The mapping defines the allowed transitions between labels: one can either continue to predict the same label, jump to the next one if it is different, or jump to a blank. The forward and backward variables are defined as follows, with $\mathbf{L} = l_1 \dots l_n$ and $\mathbf{L}' = l'_1 \dots l'_n = \oslash l_1 \oslash \dots \oslash l_n \oslash$

$$\begin{aligned} \alpha_t(l'_s) &= p(q_{1:t} \in \mathcal{Q}_t(\mathbf{L}_{1:s/2}), q_t = l'_s|\mathbf{x}) \\ \beta_t(l'_s) &= p(q_{t+1:T} \in \mathcal{Q}_{T-t}(\mathbf{L}_{s/2+1:|\mathbf{L}|}), q_t = l'_s|\mathbf{x}) \end{aligned}$$

and the recurrences are:

$$\alpha_t(l'_s) = p(q_t = l'_s|\mathbf{x}) \sum_{n=0}^{k} \alpha_{t-1}(l'_{s-n})$$

$$\beta_t(l'_s) = \sum_{n=0}^{k} p(q_{t+1} = l'_{s+n}|\mathbf{x}) \beta_{t+1}(l'_{s+n})$$

where $k = 1$ whenever $l'_s = \oslash$ or $l'_s = l'_{s-2}$ (*resp.* $l'_s = l'_{s+2}$) for forward (*resp.* backward) variables, and $k = 2$ otherwise.

The α and β variables allow to compute

$$p(\mathbf{L}|\mathbf{x}) = \sum_{\mathbf{q}\in\mathcal{Q}_{|\mathbf{x}|}(\mathbf{L})} \alpha_t(q)\beta_t(q)$$

and the derivation of the cost $-\log p(\mathbf{L}|\mathbf{x})$ leads to the following backpropagated error:

$$\frac{\partial E}{\partial a_k^t} = y_k^t - \sum_{s:\mu(s)=k} \frac{\alpha_t(s)\beta_t(s)}{\sum_r \alpha_t(r)\beta_t(r)} \tag{7.5}$$

where y_k^t is the output of the neural network at time t for label k, and a_k^t are the activations before the softmax.

7.3.4 Similarities between CTC and hybrid NN/HMM Training

We notice that Equation 7.4 and Equation 7.5 are exactly the same. The differences lie in the way y_k^t and α and β are computed. First, the CTC uses $y_k^t = p(q_t = k|\mathbf{x})$ instead of $p(q_t = k|x_t)$. This is mainly because the CTC appears in the context of RNNs, which make predictions based on the whole input sequence. Note that in (Hennebert et al., 1997), the assumptions leading to the neural network optimizing $p(\mathbf{W}|\mathbf{x})$ include the fact that the dependency of the HMM state on the input sequence is limited to some local context. If we make this assumption, the CTC can be applied to any neural network, not necessarily RNNs.

Concerning the forward and backward variables, besides the limitation of the summation to allowed transitions in CTC, the main difference is the absence of transition and prior probabilities for CTC. Setting $\frac{p(r|s)}{p(r)} = 1$ when a transition from s to r exists and 0 otherwise in the formulation of Section 7.3.2, we obtain the CTC equations. Note that these probabilities do not appear either in the framewise training with Viterbi alignments.

Thus, CTC may be seen as a simplification of the forward-backward training of hybrid NN/HMMs, using an HMM with a simple topology where each character is modeled with a single state, plus an optional state between characters.

Added to the fact that we use CTC-trained RNN in the hybrid framework, these observations lead to the following remarks. First, the HMM remove the need of a simple post-processing of the RNN outputs, so having only one output for each character is not required. The blank symbol to separate the sequences of identical characters is not justified anymore. The equations could be applied to any HMM topology. Conversely, the CTC topology yields good results with RNNs. The few number of HMM states / NN outputs could be advantageous, as pointed out in the introduction of this chapter. Therefore, we may use less HMM states in NN/HMM hybrids, and investigate the benefits of a non-character model to represent the inter-character signal. Finally, the CTC training criterion is limited to RNNs only because of the dependency of the prediction on the whole input sequence, but assuming that only a local context is relevant, it could be applied to other neural networks, such as MLPs.

In the following of this chapter, we will study these different aspects, namely the influence of the number of states per character, of the blank symbol, and of the training criterion (framewise or CTC), as well as how they interact with each other.

7.4 Topology and Blank

The transition model in the CTC is principally designed so that the neural network can be used alone to predict the desired output – the character/word sequence –, outside the HMM framework. In practice, however, the HMM framework is convenient, in particular for the integration of the language model. We have included CTC trained RNNs in hybrid NN/HMM handwriting recognition systems and obtained good results with language models (*e.g.* in Chapter 6 and in (Bluche et al., 2014a; Moysset et al., 2014; Pham et al., 2014)). The HMMs were designed to match the CTC topology: one state for every character, plus an optional state for blanks, with self-loops. We also found that the inclusion of state priors gives better results.

In this hybrid NN/HMM framework, the CTC topology is not justified anymore, except to have smaller search graphs in decoding, and possibly an easier classification problem for NNs. The topology of CTC is moreover the extreme opposite of classical HMM topologies for character modeling, which consist of several states. In the first set of experiments, we varied the number of states in CTC training of RNNs, with a number of states per character ranging from one to seven. We compared the results of these variations with the trends observed in GMM-HMMs and framewise-trained MLPs.

First, we trained GMM/HMMs. Each state has its own emission probability distribution, hence its own set of parameters. We wanted to confirm that several states per character are better than one or two, and to check whether a *blank* model to take care of the inter-character parts could help. The results are shown on the top part of Table 7.2. We confirm that for GMM/HMM models, it is better to have several states per character: when we increase the size of the models, the error decreases.

We also notice by comparing the two lines that adding a blank model between characters helps too. However, we always got better results by incrementing the character model sizes than by inserting blanks between characters. In the former approach, the inter-characters are modeled differently for each character, which requires more parameters but is better than the latter method, using a generic inter-character model.

The MLPs were trained with the framewise cross-entropy criterion, which focuses on the classification of individual frames. The ground-truth targets are obtained by forced alignment of the training data with a bootstrapping system, the corresponding GMM/HMM system. The results are presented in the middle part of Table 7.2. Apart from the well-known huge improvements brought by these discriminative models over the generative GMMs, we draw the same conclusions concerning the number of states.

Table 7.2: WER% (CER%) of different standard systems with different topologies. RNNs are trained with CTC, MLPs with framewise training, and GMM/HMMs with Viterbi training (IAM Database, **Shallow**/**Feat.**).

States	**1**	**2**	**3**	**4**	**5**	**6**	**7**
GMM							
No blank		25.7 (15.5)	20.8 (10.7)	17.3 (8.2)	16.7 (7.7)	16.5 (7.4)	**16.3 (6.9)**
Blank	30.1 (18.0)	23.5 (12.6)	18.3 (8.7)	17.1 (7.7)	17.3 (7.4)	17.0 (7.4)	18.7 (8.6)
MLP							
No blank		17.8 (8.2)	15.0 (6.1)	13.6 (5.3)	13.2 (4.8)	**12.4 (4.6)**	14.8 (4.8)
Blank	19.6 (9.0)	16.0 (6.3)	14.4 (5.5)	14.1 (5.2)	13.9 (5.2)	14.3 (5.9)	16.0 (6.7)
RNN							
No blank		19.3 (8.0)	16.5 (6.1)	14.1 (5.3)	13.7 (5.0)	14.1 (4.9)	14.5 (5.1)
Blank	**13.1 (4.9)**	13.9 (5.0)	14.3 (5.2)	13.9 (5.1)	14.9 (5.4)	15.3 (5.8)	14.2 (5.4)

Namely, adding states in the character models improves the results. The blank model, however, only helps when the character models are relatively small.

The RNNs were trained with the CTC criterion, but without the CTC constraints of one output per character plus blank. Since we use RNNs in hybrid mode with HMMs, the outputs are the different HMMs state, as for MLPs. So we apply CTC with the transition model defined by the HMMs, or equivalently, we train the hybrid model with the method presented in Section 7.3.2, ignoring the transition and state prior probabilities. The results are presented in the bottom part of Table 7.2. Without blank, increasing the number of states still improve the performance. As for MLPs, adding a blank only helps when characters are modeled with a few states.

However, there is one noticeable difference with MLP results: with the blank symbol, adding states to the character models leads to worse results. For the previous models, the CTC topology gave the worst results, but for CTC-trained RNNs, it looks like this topology is the best choice. A reasonable question to ask at this point is whether it is due to the CTC training or to the RNN, or maybe a combination of both. The next two sets of experiments attempt to answer it. Anyway, this observation shows that the role of the blank in RNNs trained with CTC is not simply to model inter-characters.

7.5 CTC Training of MLPs

The CTC training criterion was proposed in the context of RNNs. The only part of the equations that makes it specific to RNNs is the representation of NN outputs as $p(q_t|\mathbf{x})$, meaning that the prediction depends on the whole input sequence. The limitation disappears when we assume as in (Hennebert et al., 1997) that a state label only depends on some local context. The algorithm is then applicable to any model of $p(q_t|x_t)$. Moreover, we outlined in Section 7.3 the resemblance of the CTC criterion to the forward-backward training of hybrid NN/HMMs, and like we did for the last RNN

experiments, we can apply the CTC criterion to MLPs, with the best HMM topology (6 states/character, no blank symbol).

Table 7.3: CTC training of MLPs. The *"HMM"* topology has six states per character while the *"CTC"* topology is standard one for CTC, with one output per character and a blank symbol (IAM Database, **Shallow**/**Feat.**).

Training	**Topology**	**WER%**	**CER%**
Framewise	HMM	12.4	4.6
CTC	HMM	12.6	4.3
	CTC	17.6	7.4

The results are presented on Table 7.3, and compared to the best MLP trained with a framewise criterion. With the 6-state topology without blank (*"HMM"* topology), we observe some limited improvement, from 4.6% CER to 4.3%, with CTC training. On the other hand, with the classical CTC topology, consisting of one state per character and a blank model, the results are far worse. This choice of topology, which was the best one for CTC training of RNNs, is not suited to MLPs, even with CTC training. Note however that CTC training still improved the error rates for this topology over framewise training (19.6% WER / 9.0% CER in Table 7.2). In the next experiment, we conduct a more thorough comparison of framewise and CTC training, with different topologies, for MLPs and RNNs.

7.6 Framewise vs CTC Training

In this section, we compare the results of framewise and CTC (forward-backward) training of neural networks. Note that in the literature, the comparison of framewise and CTC training is carried out with the standard HMM topology with several states and no blank for framewise training, and with the CTC topology for CTC training (Graves et al., 2006; Morillot et al., 2013a). Maas et al. (2014) compare CTC-trained deep neural networks with and without recurrence, using the topology defined by the CTC framework, and report considerably better results with recurrence, which we confirm in these experiments. Here, we take one step further, comparing framewise and CTC training using the same topology in each case, and observing the effect of both the training procedure and the output topology, for MLPs and RNNs.

For each topology (1 to 7 states, with and without blank), we trained MLPs and RNNs. The results are summarized on Table 7.4. With blank, CTC training gives better results than framewise training when characters are represented by a few (one or two) states. With more states, there was no improvement nor degradation for RNNs, but the results tended to be worse with CTC for MLPs. We make the opposite observation without blank: for MLPs and RNNs, the results are similar or improved with CTC training when there are many states per characters, but degraded for one or two states.

Table 7.4: Comparison of WER%/CER% with framewise and CTC training of neural networks with different output topologies (IAM Database, **Shallow**/**Feat.**)

		Without blank		With blank	
	States	**Framewise**	**CTC**	**Framewise**	**CTC**
MLP	1	-	-	19.6 / 9.0	17.6 / 7.4
	2	17.8 / 8.2	19.1 / 8.5	16.0 / 6.3	16.4 / 6.7
	3	15.0 / 6.1	15.2 / 6.1	14.4 / 5.5	16.4 / 6.5
	4	13.6 / 5.3	13.3 / 4.9	14.1 / 5.2	14.9 / 5.6
	5	13.2 / 4.8	13.0 / 4.5	13.9 / 5.2	14.9 / 5.6
	6	**12.4** / 4.6	12.6 / **4.3**	14.3 / 5.9	16.1 / 6.4
	7	12.8 / 4.8	12.7 / 4.3	16.0 / 6.7	17.4 / 7.0
RNN	1	-	-	18.7 / 8.2	**13.1 / 4.9**
	2	17.7 / 7.5	19.3 / 8.0	15.9 / 6.1	13.9 / 5.0
	3	15.4 / 5.6	16.5 / 6.1	14.4 / 5.8	14.3 / 5.2
	4	14.2 / 5.4	14.1 / 5.3	13.8 / 5.3	13.9 / 5.1
	5	14.2 / 5.1	13.7 / 5.0	14.3 / 5.2	14.2 / 5.1
	6	14.0 / 5.1	14.1 / 4.9	14.6 / 5.8	15.3 / 5.8
	7	14.6 / 5.2	14.5 / 5.1	15.7 / 6.4	14.2 / 5.4

Moreover, CTC training without blank and with less than 5 states per character converged to a poor local optimum, for both neural networks, and most of the predictions were whitespaces. The training algorithm did not manage to find a reasonable alignment, and the resulting WERs / CERs where above 90%. To obtain the presented results, we had to initialize the networks with one epoch of framewise training. This problem did not occur when a blank model was added, suggesting that this symbol plays a role in the success in the alignment procedure in early stages of CTC training.

On Figure 7.1, we plot the CERs of MLPs (left) and RNNs (right), without blank in solid lines, and with blank in dashed ones, and using framewise training (blue) and CTC (red). We observe that systems with blanks are better with a few states, and worse with many states. Moreover, all curves but one have a similar shape: the error decreases when the number of states increases, and start increasing when there are too many states. This increase appears sooner when we add a blank model.

The only different case concerns the RNN with CTC training and blank symbol. With one state per character, the CER is as good as the RNN with six states, CTC training and no blank. The error slightly increases with the number of states. These observations show that the HMM topologies such as those presented in Chapter 4 are good choices in general. The CTC framework, including the single state per character, blank symbol and forward-backward training is especially suited to RNNs.

The full systems are influenced by the HMM decoding, and by the linguistic constraints (vocabulary and language model). On Table 7.5, we report the error rates of the neural networks with the different topologies alone. For framewise training, we report the frame error rate, which is the classification error for each frame individually. For CTC training, we report the normalized edit distance, *i.e.* a label error rate, simi-

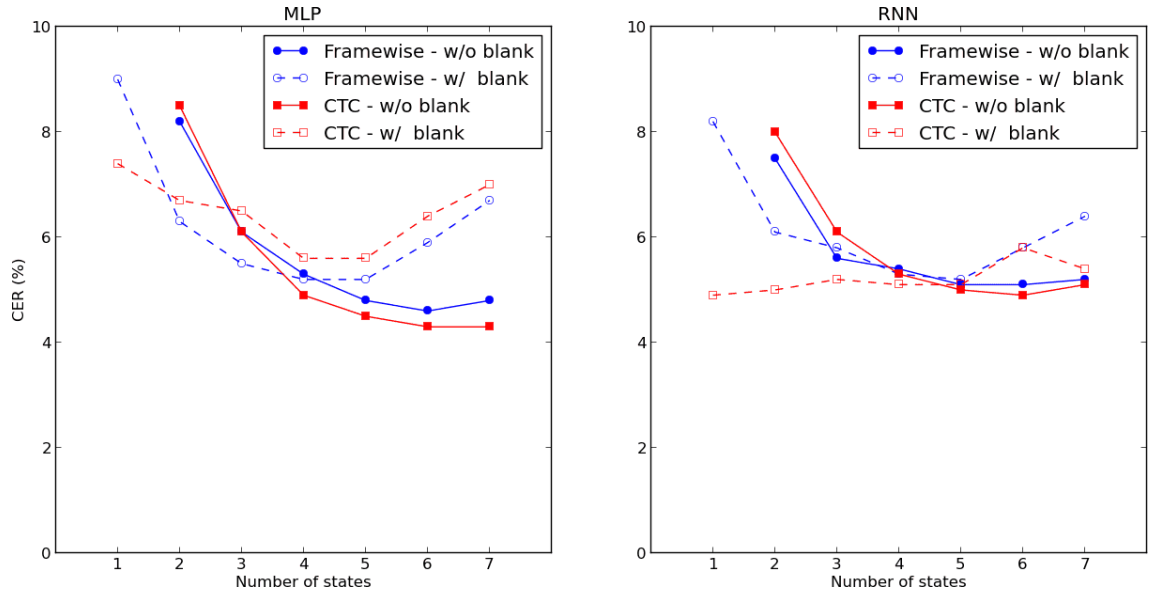


Figure 7.1: Comparison of CER% with CTC and framewise training, with and without blank (left: MLP; right: RNN; **Shallow**/**Feat.**).

lar to the RNN-CER of Chapter 6, between the target label sequences and the output sequences.

We observe that the error rates generally increase with the number of states. This is not really surprising: as we supposed in the introduction, if there are less states, there are also fewer classes to discriminate for the network, and more training sample per class on average. With framewise training, we also notice that the topologies with blank yield lower error rate. This is probably because the blank symbol, which is a junk class, and also the majority class, is easier to predict.

An interesting behaviour appears with CTC training. With the blank symbol, the same trend is observed, namely an increase of the error with the number of states per character. When there is no blank, both neural networks get better with more states. This is probably because CTC training relies on the forward-backward procedure, performing a sort of alignment of the outputs of the network with the target sequence. When the models are longer, and closer to the average character length, the alignment is certainly easier, which should favor a better learning.

7.7 Interaction between CTC Training and the Blank Symbol

7.7.1 Peaks

A typical observation in CTC-trained RNNs is the dominance of blank predictions in the output sequence, with localized peaks for character predictions. Interestingly, this behaviour still occurs with more states (*e.g.* with three states: three peaks of

Table 7.5: Error rates of neural networks alone without lexicon and language model with different topologies (*labels* are HMM states; **Shallow**/**Feat.**).

Framewise – Label Classification Error (frame level)								
	States	**1**	**2**	**3**	**4**	**5**	**6**	**7**
MLP	No blank		23.8	24.7	25.8	26.2	28.2	29.3
	Blank	17.1	18.8	20.8	22.0	23.2	25.4	28.5
RNN	No blank		14.4	15.4	16.3	17.2	19.6	20.7
	Blank	11.3	12.8	14.2	15.0	16.0	19.0	22.2
CTC – Label Edit Distance (sequence level)								
	States	**1**	**2**	**3**	**4**	**5**	**6**	**7**
MLP	No blank		77.0	53.8	44.4	39.6	34.8	32.6
	Blank	18.5	18.9	21.8	26.1	23.9	22.9	24.0
RNN	No blank		23.6	19.0	17.7	16.6	15.6	15.8
	Blank	9.2	10.7	11.5	11.6	12.2	13.0	13.0

states predictions with many blanks between each character). We also observe this with CTC-trained MLPs.

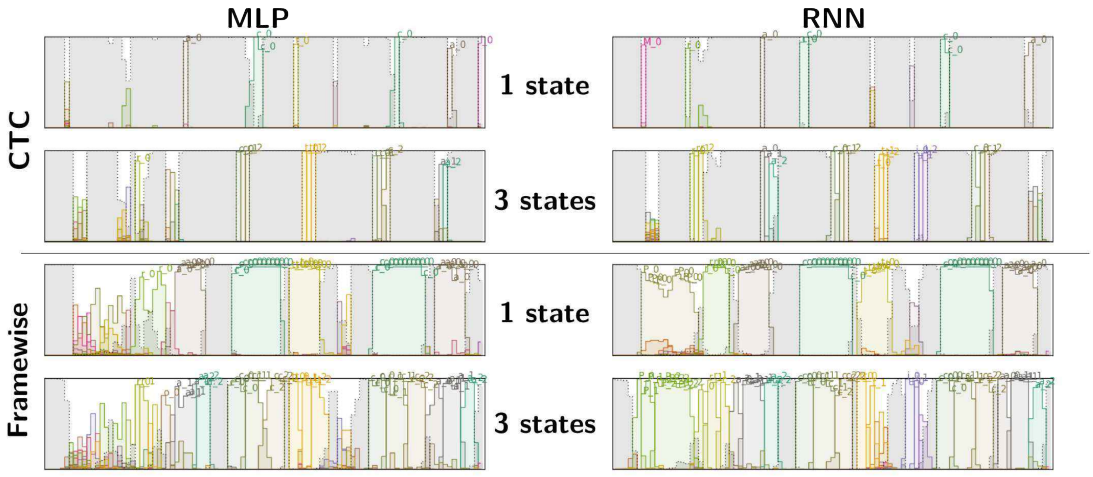


Figure 7.2: Outputs of different neural networks with different topologies and training methods. Each plot represents the NN posteriors at different timesteps. The blank output is represented in gray, and other outputs corresponding to HMM states, in color.

On Figure 7.2, we show the predictions of neural networks at different timesteps. Each output is represented with a different color, and the blank predictions are displayed in grey. We observe that the predictions are mainly blanks, for both types of neural networks, and with different numbers of states. Moreover, the peaks are not observed with framewise training of the same models. Thus, we can conclude that this typical output is due to the interaction between blank and CTC training, rather than a consequence of using RNNs or single-state character models.

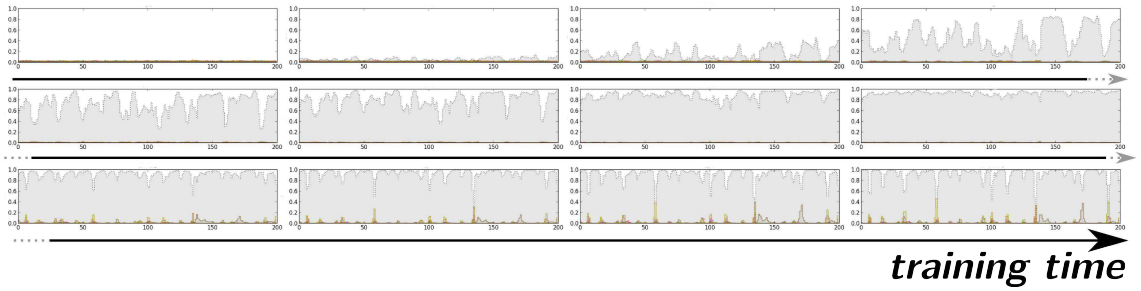


Figure 7.3: Evolutions of the outputs of an RNN for a given text line during CTC training. As in Figure 7.2, gray corresponds to the blank output, and colors to other outputs.

On Figure 7.3, we show how the outputs of a network with one state per character and blank evolve in the beginning of CTC training. In the top left plot, we see the outputs before training. Since the weights of the network have not yet been adjusted, the outputs are more or less random. As the training procedure advances, the predictions of the blank symbol increases, until the posterior probability is close to one for the whole sequence (end of the second line). At this point, the network predicts only blank labels. Then (third line), the probabilities of character labels start increasing, and peaks emerge at specific locations.

This is not surprising. In the CTC graph, there is one blank between each character. In early stages of training, the NN outputs are more or less random (at first approximation, drawn from a uniform distribution). Any path is equally likely, but summing all blank posteriors for a given timestep in the forward-backward procedure, the target for blank will be much higher than for any other character. The backpropagated error will make the network more likely to give a high posterior probability to the blanks, which in turn will favor path with many blanks in subsequent forward-backward computations of the cost.

Figure 7.4 shows the posterior probabilities computed with the forward-backward algorithm during training, which are the targets of the CTC training algorithm. We see that the fact that one in two labels is a blank will give a higher posterior probability for this symbol. When the network starts outputting only blanks, the paths with many blanks will be more likely. On the other hand, the difference between the network output for the blank label, and the posterior probability of the blank computed with the forward-backward algorithm will become small.

Predicting only blanks will penalize the cost function because of the low probabilities given to characters, and the network has only to find one location to predict each character, to "jump" from one sequence of blanks to another. Since a valid path must go through character labels, the posterior probabilities of these labels will also increase. We see on the second plot of Figure 7.4 that at some positions, the character labels have a higher posterior probability. Since the network predicts only blanks, the gradients for characters at these positions will be high, and the network will be trained

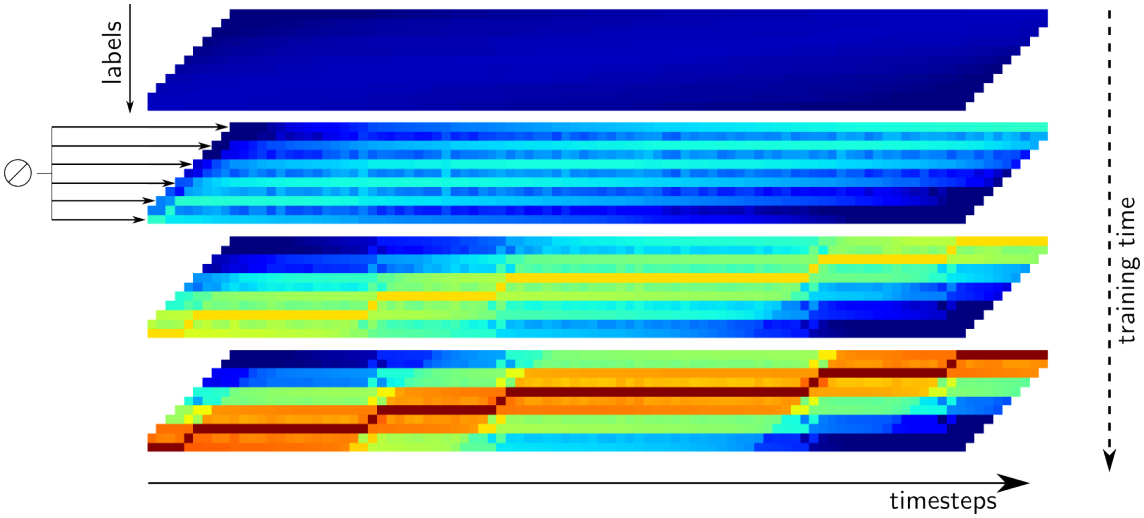


Figure 7.4: Visualisation of the state posteriors computed with the forward-backward algorithm during CTC training.

to output the character labels at specific locations. Finally, the path with blanks everywhere and characters at localized position will have a much higher probability, and the training algorithm may not consider much the other possible segmentations (bottom plot of Figure 7.4).

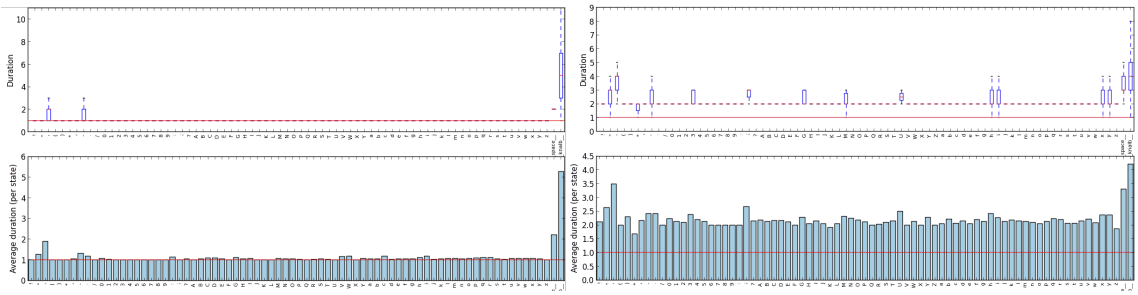
Learning to make localized character predictions is much easier for RNNs, which can use a context of arbitrary length to make a prediction at the specific timestep. Conversely, all predictions in MLPs must be made from an fixed context, which is makes more difficult to predict a character only at a specific timestep, and blank everywhere else.

7.7.2 Trying to avoid the Peaks of Predictions

From these observations, we attempted a few methods to try to avoid the peaks of predictions. First, since the peaks are not observed in framewise training, we initialized an RNN we a framewise training of one epoch. This way, we may hope that the considered paths in the forward-backward procedure give less importance to the blank symbols in the first steps. However, after CTC training, we still observe the peaks as if we used CTC from the start.

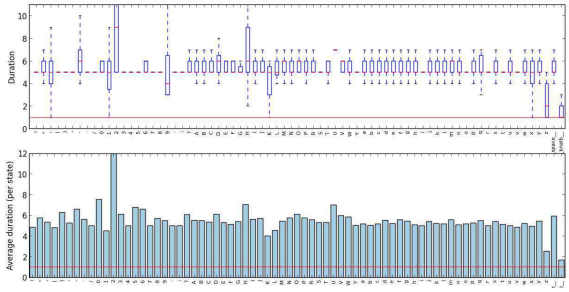
We have seen that even with several different states per character, the CTC training algorithm with the blank symbol still produced peaks. We supposed that it may be due to the much larger number of blanks in the CTC graph. To limit this phenomenon, we modified the graph to force each valid path to contain several consecutive repetitions of the same character before being allowed to transit to a blank. This way, the ratio between blanks and other labels is reduced.

We tried two, three and five label repetitions during training. For decoding, we kept the original graph representing each character with one HMM state. This training procedure effectively suppressed the localized peaks. But instead, when we repeated each label n times, the length of each character in decoding was always n. On Figure 7.5, we show the duration of each character in decoding. For each subfigure, the top plot is a boxplot for each character (the two rightmost characters are the whitespace and the blank). The bottom plot represents the average duration. Remember that during decoding, for all training strategies, *i.e.* all numbers of repetitions, the characters are still represented by only one HMM states, so no notion of duration is encoded in the search graph.



(a) Repeat once

(b) Repeat twice



(c) Repeat 5 times

Figure 7.5: Character durations in decoding when each label is repeated n times between each blank during training.

We observe that there is almost no variation in the durations, compared for example with Figure 5.2. With one repetition (original CTC), the character durations are mostly equals to one (peaks). With two or five repetitions, the RNN has learnt to output each character twice or five times. It did not solve our problem, which would be to have an output sequence representative of the input, *i.e.* to be able to retrieve the position of the characters, or their boundaries, from the output. However, it shows an interesting property of LSTM networks being able to learn exactly how many time to output a given label.

Finally, we trained the network with a smaller learning rate, 10^{-4} instead of 10^{-3}, so that the network does not learn too quickly to predict only blanks. The results are shown on Figure 7.6, with MLPs (left) and RNNs (right), three states per character,

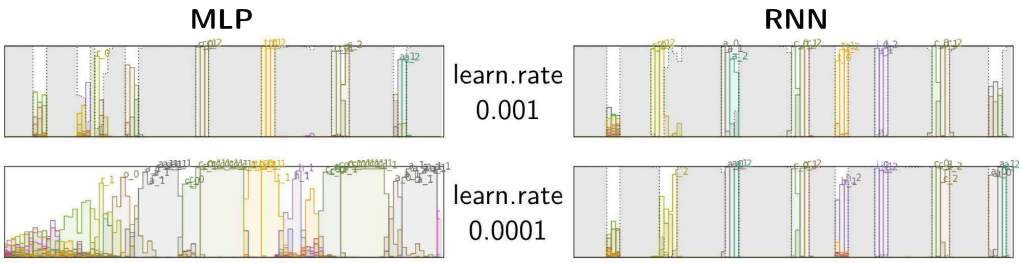


Figure 7.6: Effect of different learning rates on the output of networks trained with CTC and the blank symbol.

and a blank symbol. For RNNs, we see that the outputs are still peaks, even with the smaller learning rate. For MLPs however, the blank symbol, optional, disappears, and the outputs are not peaks anymore, but reflect better the content of the input signal. Moreover, we had better results for MLPs with this smaller learning rate, and it was used in the previous sections. It seems to confirm that it is difficult for an MLP to predict well very localized peaks for characters, and that better results are obtained when the output really represents the input at each timestep. For RNNs, the bigger learning rate yielded better results, and the presence of peaks even with the smaller learning rate seems to confirm that RNNs are able to use their recurrence efficiently to produce the minimum required number of character predictions.

7.7.3 The advantages of prediction peaks

Predicting characters only at specific timesteps is not necessarily a bad solution. Even though we cannot retrieve the character segmentation or the positions of the boundaries, this behaviour is interesting for decoding. First, models with one state and a blank yield much smaller decoding graphs. Furthermore, since the character predictions are very localized, and the blank is uninformative and shared by all word models, the number of predictions to change in order to recognize a different word is small. It means that correcting mistakes is not very costly. That also means that it is easier to keep more various hypotheses during beam search decoding.

We observed that the optimal optical scale in decoding is related to the length of the characters (in the sequence of predictions). Without blank, there are roughly between 10 and 15 frames per characters in this database, and the best optical scale is always between 10^{-1} and 15^{-1}. With CTC training and blank, the predictions are localized and correspond to 1 or 2 timesteps with one-state models, and around N timesteps for N-state ones, and the optimal optical scale is always around N^{-1}. With a larger optical scale, for a fixed beam, the decoding is much faster. Thus, when the NN is good with CTC and blank, one can obtain a fast decoding with as many alternatives, and good results.

7.8 CTC Training without Blanks

As already pointed out, when we trained a neural network, RNN or MLP, with CTC and a few states per character (less than five), but without a blank symbol, the training procedure did not converge to a good result. On Figure 7.7, we show the outputs of a neural network when there is no blank symbol between characters. On top, we display the output after framewise training. Below is the output after simple CTC training from a random initialization. The big gray area corresponds to the whitespace, and the character predictions, in color, only appear at the beginning and the end of the sequences.

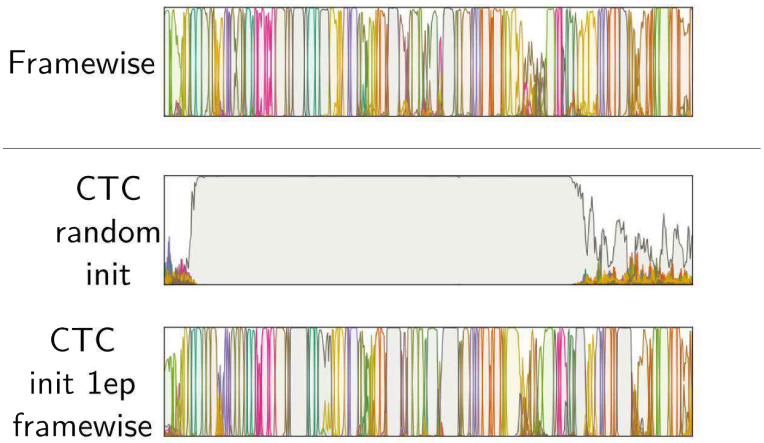


Figure 7.7: Outputs of a NN trained with CTC training without blank (**CTC**).

Like the blank was predominant in the CTC graph, now the whitespace symbol is the most frequent symbol, and the network learnt to predict it everywhere. Although it is not clear why we do not observe this behaviour with blank, it seems to indicate that this symbol is important in the CTC training from a random initialization, maybe helping to align the outputs of the network with the target sequence. A similar conclusion was drawn at the end of Section 7.6, comparing the error rates of CTC-trained network without blank and with different number of states (Table 7.5). As the number of states per character grows, the alignment in the forward-backward is easier, and the error decreases.

When we initialize the network by one epoch of framewise training, before switching to CTC training, this problem disappears, as we observe on the bottom plot of Figure 7.7. Thus, when there is no blank, we should follow the same procedure as in (Senior & Robinson, 1996; Hennebert et al., 1997), consisting of first training the network with Viterbi alignments, and then refining it with forward-backward training. Only when the number of states encodes some kind of duration of the character can we rely on the topology to yield reasonable alignments in the training procedure. However, we always got better results in that case with the framewise initialization.

7.9 The Role of the Blank Symbol

With our decoding pipeline, the role of the blank symbol cannot be limited only to the separation of identical consecutive labels. In the previous sections, we have seen the cases where the blank symbol helped to get better results. During the experiments, we observed the consequences of having or not a blank symbol in the models.

With framewise training, or GMM modeling, when the character are represented only be a few states, the addition of a blank symbol improved the result. In this case, it effectively modeled the inter-character parts in the image, which was one of its original justification. With CTC training, the blank symbol was also better when there were only a few states per characters. When the models are large enough, there may not be a need or a benefit by using this special symbol.

With CTC training and a blank symbol, we have seen that the output of the networks are localized peaks of character predictions. This observation is due to the interaction between this symbol and the CTC algorithm. These peaks are interesting for decoding, as underlined in Section 7.7. Thus one of the benefits of the blank might be to produce the peaks. However, they are more difficult to obtain with MLPs, while RNNs, with the long-term dependencies allowed by their structure, achieve very good results with CTC and blanks.

When there is no blank in the CTC algorithm, the training does not converge to a good result. The observed outputs shows that there might be a problem in the forward-backward alignments in the beginning of training. This suggests that the blank symbol is useful to get good alignments. Nonetheless, long enough HMM should be preferred for MLP, because of the difficult learning problem with blank, CTC and short model.

7.10 Conclusion

In this chapter, we have studied the training method for RNNs and MLPs. We focused on framewise training, and on the CTC algorithm yielding good results with RNNs. In the CTC paradigm, the network have one output for each character and a special blank symbol. The training method defines transitions between labels, and is reminiscent of forward-backward training of HMMs.

First, we have shown that ***the CTC training algorithm is very similar to the integrated training of NN/HMM systems with a forward-backward procedure***, already proposed in the nineties (Senior & Robinson, 1996; Hennebert et al., 1997; Yan et al., 1997). The main difference is the absence of transition and prior probabilities in CTC. Moreover, the topology defined in the CTC is very specific.

Then, we have trained GMMs, MLPs, and RNNs with different topologies. Both GMMs and ***MLPs get better results with many states and no blank model, when RNNs yield better results with the CTC topology***.

The CTC training algorithm is not limited to RNNs. We have trained MLPs with CTC, and got poor results with one state and a blank symbol. ***With the best HMM topology, CTC, or forward-backward training of MLPs produced limited improvements***, which is consistent with the findings of Hennebert et al. (1997): when the training set is large enough, forward-backward training is not much better than framewise training.

A more careful study of framewise and CTC training of MLPs and RNNs with different topologies showed that ***the CTC framework, with one state per character and a blank symbol, is especially suited to RNNs***. While in all cases, the WER decreased with the number of states, only with blank and CTC did we observe good results with short models. For MLPs, the best results where with 6 states and no blank, which was the HMM topology chosen in previous chapters

Moreover, for the networks alone, the classification error increased with the number of states, confirming that the classification problem is easier with a few states only. However, for CTC training, and without blank, adding states decreased the classification error, suggesting that a right length of models helped the alignment procedure during training.

Then, we studied the interaction between CTC and the blank symbol. We have seen that CTC without blank did not converge well, suggesting that ***when characters are represented with a few states, the blank symbol might help the alignement during CTC training***. Moreover, ***the presence of the blank in the CTC encourages the network to predict localized peaks of character probabilities, which can be helpful to have a rich and fast decoding***.

Finally, we can conclude that ***the CTC paradigm***, including the forward-backward procedure, but especially the single state per character and the blank symbol, is ***particularly suited to RNNs***, and allows them to produce very good results. For MLPs, we confirmed that the HMM topology chosen in this thesis was good to obtain good results. The forward-backward procedure is also found in the sequence discriminative training applied to fine-tune these networks.

Chapter 8

Experimental Results, Combinations and Discussion

Contents

8.1 Introduction 187

8.2 Summary of Results on Rimes and IAM Databases 188

8.2.1 MLP/HMM Results 188

8.2.2 RNN/HMM Results 190

8.2.3 Comparison of MLP/HMM and RNN/HMM Results 191

8.2.4 Combination of the Proposed Systems 193

8.2.4.1 ROVER Combination 193

8.2.4.2 Lattice Combination 194

8.2.4.3 Comparison with the State-of-the-Art 195

8.3 The Handwritten Text Recognition tranScriptorium (HTRtS) Challenge 196

8.3.1 Presentation of the HTRtS Evaluation and of the Experimental Setup 196

8.3.2 Systems Submitted to the Restricted Track 197

8.3.2.1 Deep MLPs 197

8.3.2.2 Deep RNNs 198

8.3.2.3 Combination 199

8.3.2.4 Competition Results for the Restricted Track 199

8.3.3 Systems Submitted to the Unrestricted Track 200

8.3.3.1 Adding Data to the Training of Optical Models 200

8.3.3.2 Adding Data to the Training of Language Models . . . 201

8.3.3.3 Competition Results for the Unrestricted Track 203

8.3.4 Post-Evaluation Improvements 203

8.3.4.1 A More “Author-Specific” Language Model 203

8.3.4.2 A More Careful Tuning of Neural Networks 204
8.4 Conclusion 205

8.1 Introduction

In the previous chapters, we proposed hybrid NN/HMM systems with deep MLPs and deep BLSTM-RNNs. We studied different aspects of these neural networks, namely:

- their ***inputs***: **handcrafted features**, and **pixel values**, including more or less **context**,
- their ***architecture***: we reported significant improvement by increasing the **depth** of the networks, and studied the importance of LSTM **recurrent** layers,
- their ***outputs***: a varying number of **HMM states**, and the CTC approach with one label for each **character**, and a **blank** symbol,
- the ***training criterion***: either **framewise** training with the cross-entropy, **CTC** training, or **sequence-discriminative** training with sMBR.

In this chapter, we compare the results of deep MLPs with those of deep BLSTM-RNNs, for handcrafted features and pixel values. Although RNNs are the state-of-the-art systems for handwriting recognition, leaving little room for MLPs in the past few years, we will see that they achieve similar performance. Moreover, as already pointed out, the discrepancy between results with handcrafted features and pixel values tends to be small with deep neural networks.

The two types of inputs and of neural networks give us four different systems, likely to make different mistakes. We explore two types of combination of the results produced by the transcription pipelines, namely ROVER (Fiscus, 1997) and lattice combinations (Xu et al., 2011), and report significant improvements.

Finally, we compare the performance of our systems with the results published on each database, and show that we achieve state-of-the-art performance. In particular, with the combinations of models, we report the best results on Rimes and IAM databases.

This chapter is divided into two parts.

In Section 8.2, we present our results on Rimes and IAM databases.

Section 8.3 is focused on the HTRtS contest, held in 2014 to evaluate the quality of the transcription of Bentham database. We first present the systems we submitted to the competition. In particular, we were the only contestant to participate to both the *restricted* track, where only the data from Bentham database are allowed to build the system, and the *unrestricted* track, where additional data can be used. Then, we present the improvements brought to the systems after the competition.

We present our conclusions in Section 8.4.

8.2 Summary of Results on Rimes and IAM Databases

8.2.1 MLP/HMM Results

In Chapter 5, we have seen the importance of including enough context in the MLP inputs, as well as the gains obtained with deep MLPs and a fine-tuning of the networks with a sequence-discriminative criterion. In this section, we are interested in the full decoding pipeline, including the linguistic knowledge (vocabulary and language model).

The MLPs retained for this chapter correspond to those achieving the best results from the previous chapters, that is:

Rimes:

- for *handcrafted features*, an MLP with 3 hidden layers, consisting of 512 units each, ±3 frames of context, trained with the cross-entropy criterion, and fine-tuned with 7 epochs of sMBR
- for *pixels*, an MLP with 5 hidden layers, consisting of 512 units each, trained with the cross-entropy criterion, and fine-tuned with 6 epochs of sMBR

IAM:

- for *handcrafted features*, an MLP with 5 hidden layers, consisting of 256 units each, ±3 frames of context, trained with the cross-entropy criterion, and fine-tuned with 7 epochs of sMBR
- for *pixels*, an MLP with 5 hidden layers, consisting of 1,024 units each, trained with the cross-entropy criterion, and fine-tuned with 8 epochs of sMBR

As seen in Section 4.4, several parameters are included in the decoding procedure:

- the **optical scale** α, balancing the scores of the optical model and the scores of the search graph, corresponding to transitions and LM probabilities
- the **word insertion penalty (WIP)**, controlling the number of words in the output
- the **prior scale** κ, applied to the state priors used to scale the NN posteriors.

We varied the values of these parameters, and show the results on Figure 8.1. For each parameter, the opaque lines are the best results with the given value of the parameter, optimizing the other ones. The transparent lines are with fixed values for all parameters. Overall, for both databases and kinds of inputs, we found similar optimal values. We observe that the optical scale was the most crucial parameter, with a best value around 0.1. The best prior scale is 1.0, and some improvement is also obtained by tuning the word insertion penalty.

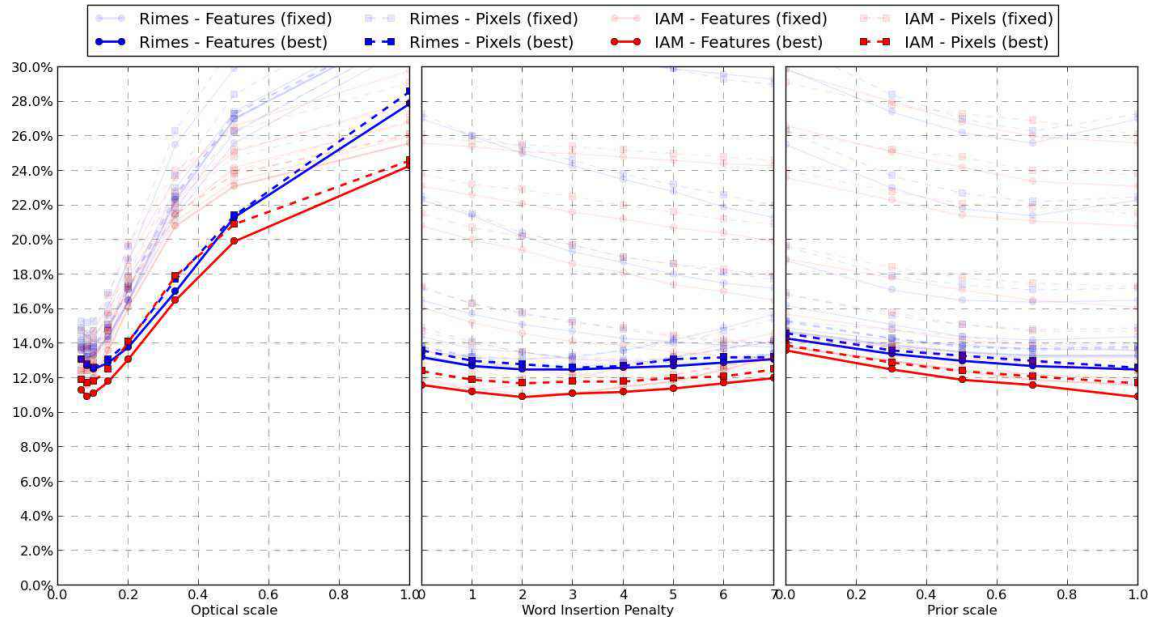


Figure 8.1: Influence of the decoding parameters (optical scale, word insertion penalty, prior scale) on the WER% of hybrid MLP/HMM systems (**Deep**/**Seq.**).

We have performed the decoding with different levels of linguistic constraints. The simplest one is to recognize sequences of characters. In this setup, all characters HMMs are in parallel, which only constrains the sequences of HMM states to correspond to characters. In the next level, the lexicon is added, so that output sequences of characters form sequences of valid words. Finally, the language model is added, to promote likely sequences of words.

Table 8.1: Effect of adding linguistic knowledge in MLP/HMM systems (**Deep**/**Seq.**).

		Features		**Pixels**	
		WER%	CER%	WER%	CER%
Rimes	*no lexicon*	61.1	17.8	59.5	17.8
	lexicon	26.9	6.8	26.1	7.2
	lexicon+LM	12.5	3.4	12.6	3.8
IAM	*no lexicon*	54.7	15.8	54.2	15.6
	lexicon	24.7	7.7	25.5	8.0
	lexicon+LM	10.9	3.7	11.7	4.0

The results are reported on Table 8.1. We see that without lexicon, when the only constraint is to recognize characters, *i.e.* valid sequences of HMM states, the results are not good. The WERs are high, partly because when training the models, the recognition of a whitespace between words was optional. Therefore, the missing whitespaces in the predictions induce a high number of word merges in the output, *i.e.* a large number of deletions and substitutions.

When a vocabulary is added, the error rates are roughly divided by two. Another reduction by a factor two is achieved when a language model is present. These results show the importance of the linguistic constraints to correct the numerous errors of the MLP/HMM system.

8.2.2 RNN/HMM Results

In Chapter 6, we observed the benefits of deep RNNs, and the significant improvements achieved with the dropout technique.

The RNNs retained for this chapter correspond to those achieving the best results from the previous chapters, that is:

Rimes:

- for *handcrafted features*, an RNN with 7 hidden layers, consisting of 200 units each, trained with dropout before every LSTM
- for *pixels*, an RNN with 5 hidden layers, consisting of 200 units each, trained with dropout before and after every LSTM

IAM:

- for *handcrafted features*, an RNN with 5 hidden layers, consisting of 200 units each, trained with dropout before the first two LSTMs and after the last one
- for *pixels*, an RNN with 7 hidden layers, consisting of 200 units each, trained with dropout after every LSTM

As we did for MLPs, we studied the influence of the decoding parameters, the optical and prior scales, and the word insertion penalty, in the full pipeline. The results are depicted on Figure 8.2. For RNNs, the optical scale is also the most crucial decoding parameter. The best values we found across all experiments was 1.0, which is quite different from MLPs. A tentative explanation was given in the previous chapter, where we observed that the optimal optical scale was related to the expected number of predictions to change in order to recognize another character. For CTC-trained networks, there is only one prediction for each character, the remaining being blank predictions, while in conventional systems, the number of prediction is related to the length of the characters, *i.e.* about 10 frames. Moreover, the best prior scale was around 0.5 for all RNN experiments, and tunig the word insertion penalty also provided some improvement.

We also applied different levels of linguistic constraints, and we present the results on Table 8.2. We notice that the differences between no constraints, and lexicon with LM are not as dramatic as for MLPs. The WERs are only multiplied by 2 to 2.5 when we remove the constraints, when it was roughly multiplied by 5 for MLPs. We may put it in perspective with Chapter 6, where we have seen that a lot of context was used by the network through the recurrent connections, which seems to enable the network to predict characters with some knowledge about the words.

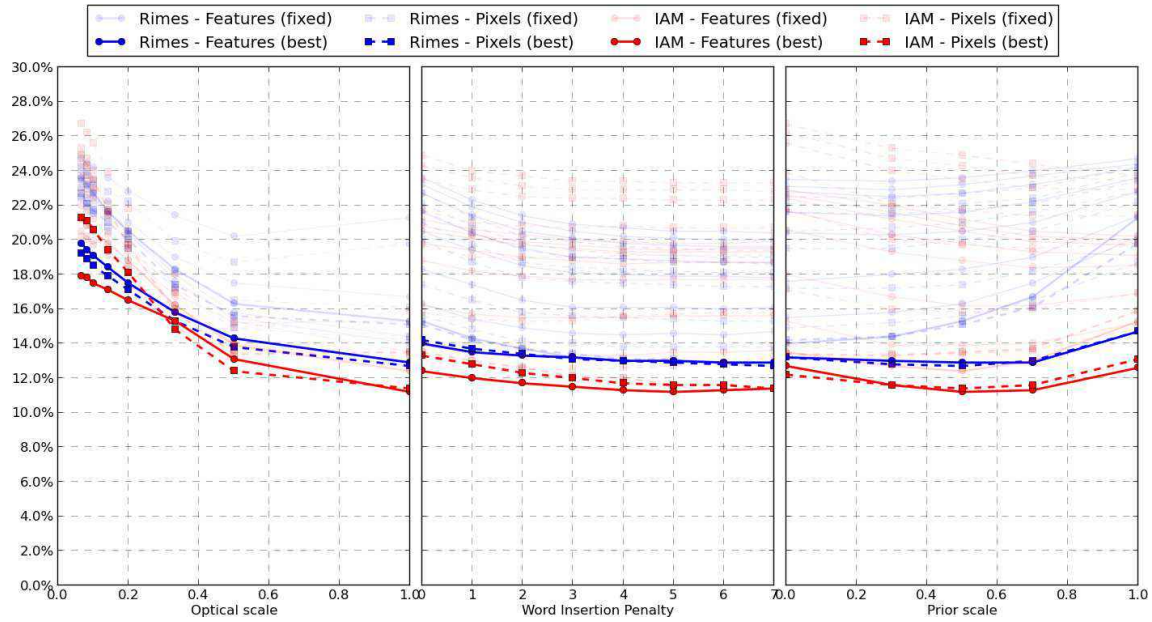


Figure 8.2: Influence of the decoding parameters (optical scale, word insertion penalty, prior scale) on the WER% of hybrid RNN/HMM systems (**Deep**/ **Drop.**/ **CTC**).

Yet, both the lexicon and the language model bring significant improvements, and remain very important to achieve state-of-the-art results. The fact that the RNNs produce reasonably good transcriptions by themselves should make them more suited to open-vocabulary scenarii (*e.g.* the approaches of Kozielski et al. (2013b); Messina & Kermorvant (2014)), where the language model is either at the character level, or an hybrid between a word and a character language model.

Table 8.2: Effect of adding linguistic knowledge in RNN/HMM systems (**Deep**/ **Drop.**/ **CTC**).

		Features		**Pixels**	
		WER%	CER%	WER%	CER%
Rimes	*no lexicon*	20.1	5.1	20.9	5.6
	lexicon	16.7	5.3	16.4	4.3
	lexicon+LM	12.8	3.8	12.7	4.0
IAM	*no lexicon*	27.5	7.9	24.7	7.3
	lexicon	17.6	5.5	16.7	5.3
	lexicon+LM	11.2	3.8	11.4	3.9

8.2.3 Comparison of MLP/HMM and RNN/HMM Results

In this section, we compare the results with MLPs and RNNs on the one hand, and with handcrafted features and pixels on the other hand. They are summarized on Table 8.3. As one can notice, similar error rates are achieved by the two kinds of

optical models, and of inputs, making a definite conclusion hard to draw about what are the best choices.

Table 8.3: Comparison of MLP and RNN optical models with different types of inputs (**Deep**).

		Features		**Pixels**	
		WER%	CER%	WER%	CER%
Rimes	*Deep MLP*	12.5	3.4	12.6	3.8
	Deep RNN	12.8	3.8	12.7	4.0
IAM	*Deep MLP*	10.9	3.7	11.7	4.0
	Deep RNN	11.2	3.8	11.4	3.9

What we may conclude however, is that:

- pixel values yielding similar performance as handcrafted features, the need to design and implement features vanishes, and one may simply use the pixels directly, as pointed out in previous chapters
- although RNNs are found in all the best published systems for handwritten text line recognition, they are not the only option, and MLPs should not be neglected.

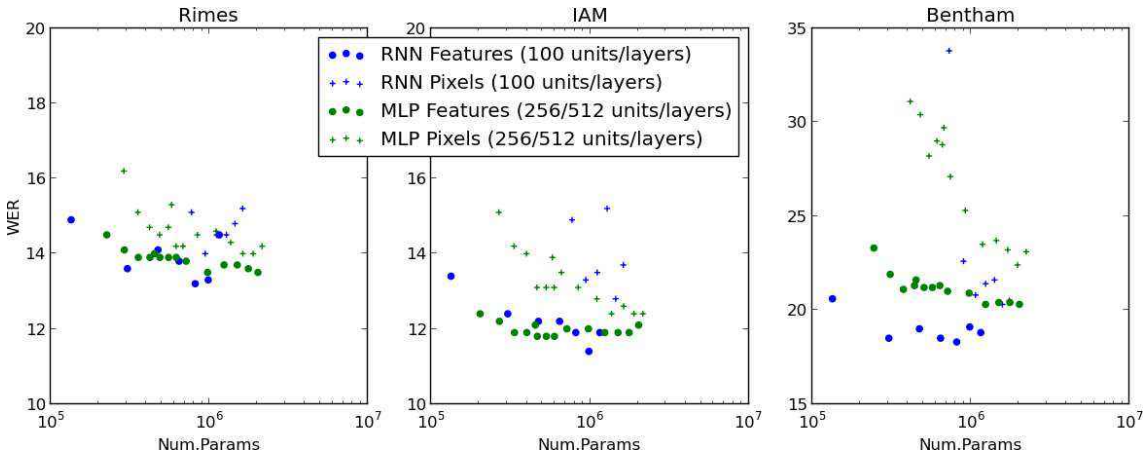


Figure 8.3: Influence of the number of free parameters in MLPs and RNNs. Circles correspond to handcrafted features, and + to pixels. RNNs are shown in blue and MLPs in green, each point is one network (**Deep**).

On Table 8.3, we have compared the best MLPs and RNNs, selected from the experiments presented in the previous chapters. Although the MLPs have between 512 and 1,024 units per layer, and RNNs have only 200, one should remember that the LSTM layer have actually 200 units in each direction, and that with the recurrent connections and with the gates, the number of free parameters in LSTM layers is roughly multiplied by 8.

On Figure 8.3, we selected the MLPs with 256 and 512 hidden units per layer, and the RNNs with 100 units per layer, for different depths and types of inputs. We compare the number of free parameters, and the word error rates achieved on Rimes, IAM and Bentham. Each point represents one network (blue for RNNs, green for MLPs). We see that, except for Bentham, where RNNs yield far better results than MLPs, the WERs of the two kinds of networks are also comparable for a given number of parameters.

8.2.4 Combination of the Proposed Systems

For each database, we have selected four systems, two MLPs and two RNNs, with feature and pixel inputs. We have seen that their performance was comparable. However, the differences between these systems probably lead to different errors. Thus, we combined their outputs, with two methods: ROVER (Fiscus, 1997), which combines the transcription outputs, and a lattice combination technique (Xu et al., 2011), which extracts the final transcript from the combination of lattice outputs. For both methods, we started by computing the decoding lattices, obtained with the decoder implemented in Kaldi.

8.2.4.1 ROVER Combination

ROVER stands for Recognizer Output Voting Error Reduction. It is a system combination method, which inputs are the sequences of words recognized by different systems. Each word of the output sequence of each system is associated with a confidence score, computed with the Minimum Bayes Risk (MBR) decoding method presented in (Xu et al., 2011).

The word sequences of all systems are iteratively aligned to build a confusion network. So-called *NULL* arcs are inserted to account for deletions in the alignments. Then, the voting step computes a score for each unique word w at each position i with the following formula:

$$s(w,i) = a\frac{N(w,i)}{N} + (1-a)C(w,i) \tag{8.1}$$

where N is the number of systems, and $a \in [0,1]$ is an interpolation parameter between the frequency of w at position i and the final confidence $C(w,i)$. $C(w,i)$ may be the maximum or the average confidence score of all occurrences of w at position i. The confidence of a *NULL* arc is another hyper-parameter of the algorithm. The final result is the sequence $\mathbf{W}^* = w_1^* w_2^* \dots w_n^*$, where $w_i^* = arg\max_w s(w,i)$.

The results of ROVER combination of the four systems are reported on Table 8.4. We varied the parameters of this combination method, and obtained the best results with $a = 0$, *i.e.* the word frequency in the confusion network is ignored, $C(w,i)$ being the average word score, and with a confidence of 1 for *NULL* arcs.

We see that the combination significantly outperforms the best single system. We tried different subsets of systems to be combined, and always recorded improvements,

but we still obtained better results by including all of them. The observed combination results support the hypothesis that the different systems make different mistakes, and can complement each other.

Table 8.4: ROVER and Lattice combination of MLPs and RNNs with features and pixel inputs on Rimes and IAM (**Deep**).

		Rimes		**IAM**	
		WER%	CER%	WER%	CER%
Deep MLP	*Features*	12.5	3.4	10.9	3.7
	Pixels	12.6	3.8	11.7	4.0
Deep RNN	*Features*	12.8	3.8	11.2	3.8
	Pixels	12.7	4.0	11.4	3.9
ROVER combination		11.3	3.5	9.6	3.6
Lattice combination		11.2	3.3	9.6	3.3

8.2.4.2 Lattice Combination

Xu et al. (2011) developed an algorithm to minimize Bayes risk with respect to the word error rate, or rather the Levenstein edit distance, from lattices. The algorithm iteratively finds the best word sequence in a lattice, using the forward-backward algorithm to compute an approximate edit distance of the lattice to a given word sequence. The posterior probabilities of having a word at a given position are used to update the reference word sequence. The forward-backward procedure is repeated with the new reference word sequence, and the whole process is repeated until convergence.

We used this decoding method to get the scored outputs for ROVER combination. The results of MBR decoding were not much better than those of the standard MAP decoding. This algorithm can take several input lattices. Given a weight for each one, the procedure can be applied to each lattice, and the score of each word is the weighted sum of the scores computed in individual lattices. We scaled the lattices of each system individually with the corresponding optimal acoustic scale, and added the optimal word insertion penalties.

We combined our systems with this lattice-based method, and the results are reported on Table 8.4. The only hyper-parameter of this combination algorithm is the weights assigned to different systems. We tried several, but obtained similar results as those presented, which correspond uniform weights. The difference between ROVER and lattice combinations is small on the validation sets of Rimes and IAM, as we see in Table 8.4. Nonetheless, lattice combination is never worse, and is clearly better than ROVER on the evaluation sets (Section 8.2.4.3), and on Bentham database (Section 8.3.2.3).

8.2.4.3 Comparison with the State-of-the-Art

The final results, comparing different models and input features on the one hand, and comparing our proposed systems with other published results on the other hand, are reported on Tables 8.5 (Rimes) and 8.6 (IAM). The error rates are reported on both the validation and evaluation sets. The conclusions of the previous sections about the small differences in performance between MLPs and RNNs and between features and pixels are still applicable to the evaluation set results.

Table 8.5: Final results on Rimes database

		Dev.		Eval.	
		WER%	**CER%**	**WER%**	**CER%**
GMM-HMM	Features	17.2	5.9	15.8	6.0
MLP	Features	12.5	3.4	12.7	3.7
	Pixel	12.6	3.8	12.4	3.9
RNN	Features	12.8	3.8	12.6	3.9
	Pixels	12.7	4.0	13.8	4.6
ROVER combination		11.3	3.5	11.3	3.7
Lattice combination		11.2	3.3	**11.2**	3.5
Pham et al. (2014)		-	-	12.3	**3.3**
Doetsch et al. (2014)		-	-	12.9	4.3
Messina & Kermorvant (2014)		-	-	13.3	-
Kozielski et al. (2013a)		-	-	13.7	4.6
Messina & Kermorvant (2014)		-	-	14.6	-
Menasri et al. (2012)		-	-	15.2	7.2

Table 8.6: Final results on IAM database

		Dev.		Eval.	
		WER%	**CER%**	**WER%**	**CER%**
GMM-HMM	Features	15.2	6.3	19.6	9.0
MLP	Features	10.9	3.7	13.3	5.4
	Pixel	11.4	3.9	13.8	5.6
RNN	Features	11.2	3.8	13.2	5.0
	Pixels	11.8	4.0	14.4	5.7
ROVER combination		9.6	3.6	11.2	4.7
Lattice combination		9.6	3.3	**10.9**	**4.4**
Doetsch et al. (2014)		**8.4**	**2.5**	12.2	4.7
Kozielski et al. (2013a)		9.5	2.7	13.3	5.1
Pham et al. (2014)		11.2	3.7	13.6	5.1
Kozielski et al. (2013a)		11.9	3.2	-	-
Messina & Kermorvant (2014)		-	-	19.1	-
Espana-Boquera et al. (2011)		19.0	-	22.4	9.8

Not suprisingly, the systems based on neural networks outperform by far the GMM-HMM baseline systems presented in Chapter 4: the relative improvement is about 30%.

Moreover, on Rimes, we see that all of our single systems achieve state of the art performance, competing with the systems of Pham et al. (2014), which uses the same language model with an MDLSTM-RNN with dropout, trained directly on the image, and of Doetsch et al. (2014), an hybrid BLSTM-RNN.

On IAM, it is worth noting that the decoders of Kozielski et al. (2013a); Doetsch et al. (2014) include an open-vocabulary language model which can potentially recognize any word, when the error of our systems is bound to be higher than the OOV rate of 3.7%. For (Kozielski et al., 2013a), the second result in Table 8.6 corresponds to the closed vocabulary decoding with the same system as the first one. Unfortunately, the results on the evaluation set are not reported with this setup, but from the validation set errors, we may consider that our single systems achieve similar performance as the best closed-vocabulary systems of Pham et al. (2014) and Kozielski et al. (2013a).

Finally, both combination methods clearly outperform the best published WERs on Rimes and IAM, even those obtained with open-vocabulary systems. The lattice combination yields lower WERs and CERs than the ROVER method.

8.3 The Handwritten Text Recognition tranScriptorium (HTRtS) Challenge

In this part, we present our results on the Bentham database. This database was released for the Handwritten Text Recognition tranScriptorium (HTRtS) challenge, held in 2014. Due to the time constraints for the competition, the submitted systems do not always match those presented in the previous chapters.

We will first present the competition in Section 8.3.1. There were two tracks to which we participated. In the **restricted track**, only the data of the Bentham database were allowed to train the systems, which we present in Section 8.3.2. In the **unrestricted track**, we could include more data, both for the optical and the language model. We introduce the additional data we used for this track in Section 8.3.3. We study the influence of adding data to optical models and to language models in isolation and in combination. Finally, in Section 8.3.4, we explain the refinements done after the challenge, and present the final results.

8.3.1 Presentation of the HTRtS Evaluation and of the Experimental Setup

The HTRtS contest was organized in the scope of the International Conference on Frontiers in Handwriting Recognition (ICFHR 2014; Sánchez et al. (2014)). The documents, prepared for the tranScriptorium project (Sánchez et al., 2013) by the University College, London, consist of a subset of the Bentham collection. They are handwritten manuscripts written by the British philosopher Jeremy Betham and his staff in the 18th and 19th centuries.

The data, presented in Section 3.2, were the only allowed source of information to build the recognition system for the restricted track of the competition. More data can

be used to train systems submitted to the unrestricted track. The other contestants both trained MDLSTM-RNNs (Sánchez et al., 2014). The *CITLab* team took part only in the restricted track. The *A2iA* team used the open-vocabulary modeling presented in (Messina & Kermorvant, 2014), and only submitted a system for the unrestricted track. The organizers of the contest provided a baseline GMM/HMM system with a bigram LM, achieving a WER of 32.6% on the development set.

We took part in both tracks. The image preprocessing and feature extraction applied in all the following experiments, and for the contest, are those presented in Section 4.2. The procedure followed to build the language models was explained in Section 4.3. For the restricted track, we used the language model of Section 4.3. For the unrestricted track and the complementary experiments, we only added more data to the language model estimation.

For this contest, we built several systems, using the provided data, as well as additional data. We trained hybrid NN/HMM systems, with deep MLPs and RNNs, working on different type of features (handcrafted and pixels values). For improved performance, we also combined the different systems.

8.3.2 Systems Submitted to the Restricted Track

Because of the limited time allowed to train the systems for the competition, the performance of the systems presented in this section may differ from those of previous chapters. We will give a brief description of the systems retained for the restricted track, as well as their results on the validation set. Then, we compare different combination schemes, and present the results of the competition.

8.3.2.1 Deep MLPs

The forced alignments computed with the GMM/HMM of Section 4.5 are used to create a training set for MLPs, with either handcrafted or pixel features. The networks are pre-trained with the unsupervised layerwise training method described in (Hinton et al., 2006), and fine-tuned with a cross-entropy criterion. Each hidden layer has 1,024 units and a sigmoid activation, and the output layer has 372 nodes, one for each HMM state.

We trained networks with different depths, and in the case of handcrafted features, different contexts. The WERs on the validation set are summarized on Table 8.7.

We further trained the best networks (in bold face in Table 8.7) with a sequence-discriminative training criterion (sMBR; Kingsbury (2009)), after realignment of the training set using the cross-entropy-trained networks. We record relative WER improvements up to 11.9% (Table 8.8).

The two selected networks are:

- **MLP features** : sMBR-trained MLP using ±9 input frames of handcrafted features, with 4 hidden layers of 1,024 nodes.

Table 8.7: MLP results (cross-entropy framewise training). The figures are WERs (%) (**Xent**).

Features	Context	Number of hidden layers 1	2	3	4	5	6	7
Hand-crafted	± 1	27.2	26.4	25.9	26.3	25.5	25.7	25.5
	± 3	26.2	25.9	26.3	26.2	26.0	26.2	25.7
	± 5	27.7	26.3	25.8	26.0	25.7	25.7	25.6
	± 7	27.7	27.2	26.0	26.2	25.7	26.1	25.8
	± 9	26.5	25.4	25.1	**24.4**	24.5	24.7	24.6
Pixels	-	33.2	25.0	24.4	23.5	23.8	**22.8**	22.9

Table 8.8: Improvement brought by sMBR sequence training, as oposed to the cross-entropy framewise training (**Deep**/**Seq.**).

Inputs	Training	WER	CER
Features	Cross-entropy	21.0%	8.9%
	+sMBR	19.4% (-7.6%)	7.9% (-11.2%)
Pixels	Cross-entropy	22.6%	10.7%
	+sMBR	19.9% (-11.9%)	8.2% (-23.4%)

- **MLP pixels** : sMBR-trained MLP using pixel features, with 6 hidden layers of 1,024 nodes.

8.3.2.2 Deep RNNs

For both types of inputs – handcrafted features and pixels, we trained BLSTM-RNNs. The inputs are the sequences of feature vectors, and there is one output for each character, plus one for a special non-character symbol, used to cope with the different sizes of the input and the output (94 outputs). We trained the networks with the Connectionist Temporal Classification (CTC) objective function. To improve the performance of the RNNs, we also applied the dropout technique after the LSTM layers.

Table 8.9: RNNs on handcrafted and pixel features (**Deep**/**CTC**).

	Handcrafted Features RNN-CER%	WER%	CER%	Pixels RNN-CER%	WER%	CER%
7x100	11.1	18.5	7.5	12.2	21.4	8.8
7x200	11.0	18.0	7.0	11.8	20.6	8.4
+ dropout	8.9	**17.2**	6.7	9.2	**18.7**	7.3

The results are presented on Table 8.9. As previously, the systems we kept are indicated in bold face:

- **RNN features** : CTC-trained BLSTM-RNN using dropout, handcrafted features, with 7 hidden layers of 200 nodes.
- **RNN pixels** : CTC-trained BLSTM-RNN using dropout, pixel features, with 7 hidden layers of 200 nodes.

8.3.2.3 Combination

On Table 8.10 we report again the performance of restricted systems. We see that NN optical models bring a huge improvement over the standard ML-trained GMMs. The handcrafted features seem better than raw pixel values, although the difference is small for MLPs. RNNs gave lower error rates than MLPs on this task.

Table 8.10: Summary of results of restricted systems.

System		WER%	CER%
GMM-HMM	*Features*	27.9	14.5
Deep MLP	*Features*	19.4	7.9
	Pixels	19.9	8.2
Deep RNN	*Features*	17.2	6.7
	Pixels	18.7	7.3

The different systems use different optical modeling and input features, and make different mistakes. We tried two system combination methods to improve the results: the ROVER combination described in (Fiscus, 1997) and the lattice combination method described in (Xu et al., 2011). We combined the four systems, and report the results on Table 8.11, showing the superior performance of the lattice-based combination. The relative improvements brought by the best combination are 10% WER and 11% CER over the best single system.

Table 8.11: Comparison of combination techniques for the four restricted track systems.

Method	WER%	CER%
ROVER combination	16.0	6.6
Lattice combination	15.4	5.9

8.3.2.4 Competition Results for the Restricted Track

We submitted the single systems and the result of the lattice combination the the restricted track of the competition. The WERs achieved by these systems are reported on Table 8.12. The performances are similar to those obtained on the validation data, although we got slightly better results with features and slightly worse ones with pixels.

The combination yielded 15.0% WER, just above the 14.6% of the CITlab system.

Table 8.12: Competition Results for the Restricted Track.

Model		WER%
Deep MLP	*Features*	19.0
	Pixels	20.0
Deep RNN	*Features*	17.1
	Pixels	19.0
Lattice combination		15.0
CITlab		**14.6**

8.3.3 Systems Submitted to the Unrestricted Track

For the unrestricted track, we added more data to the training of RNNs with handcrafted features, and included a larger corpus to estimate the language model. We studied the effects of the additional data for optical modeling, and for language modeling.

8.3.3.1 Adding Data to the Training of Optical Models

To train better optical models, we used annotated images from other databases and from the web. We selected them because they corresponded to documents written in English, or historical documents, or manuscripts from the same epoch as Bentham documents.

The IAM database (Marti & Bunke, 2002) is a well-known database for handwriting recognition. The training set consists of 747 pages of pretty clean handwritten passages of the English LOB corpus, copied by different writers. The positions and transcripts of text lines are provided with the database.

The Georges Washington database (Fischer et al., 2012) is extracted from Washington Papers, dating from the 18th century. It is written in English by two writers, and the available version provided by the University of Bern consists of preprocessed text lines along with their transcription.

The NUMEN database, provided by Numen Digital, comprises 13,649 historical documents in old creole French containing land surveying reports. It contains annotations of line positions and transcriptions. We only used a subset of 11,710 lines for training.

The IBM UB 1 database[1] was collected at the University of Buffalo. It contains online and offline data, totalling around 6,000 pages of cursive handwritten texts produced by more than 40 writers. The line positions are unknown, and the transcript is provided only for whole pages or couple of pages.

The Abraham Lincoln database[2] is not an official database. It was retrieved from the Library of Congress, and is made of Lincoln's correspondence manuscript documents from the 19th century. The documents were produced by different writers, and again, the line positions are unknown and the transcript is available only for whole

[1] http://www.cubs.buffalo.edu/hwdata
[2] http://memory.loc.gov/mss/mal/

documents of several pages.

We extracted subsets of text lines with mapped transcript from the last two databases using the method presented in (Bluche et al., 2014b). We modified the algorithm parameters to retrieve only those lines for which we can be confident that the extracted position and mapped transcript are correct. Some errors are inevitably introduced by this automatic procedure, but we trusted the recognition systems to be able to cope with that "noise" during training.

A summary of the data used for training the optical models is presented on Table 8.13. Later, when we say that we use only a subset of some database, it corresponds to approximately 10% of randomly selected lines.

Table 8.13: Data used for optical model training.

Track	Name	Number of text lines
Restricted	Bentham	9,198
Unrestricted	IAM	6,482
	NUMEN	11,710
	G. Washington (GW)	642
	IBM UB 1	825
	A. Lincoln (AL)	3,960

We trained BLSTM-RNNs with this additional data for the unrestricted track. Due to time constraints, we only trained such systems with handcrafted features, and with the same architecture (7x200 hiddens) and training procedure as the *RNN features* system of the restricted track.

We built three such systems, using different subsets of the available images. We refer to these systems by:

- **uRNN1** : Bentham, G. Washington, subset of IAM, Numen and A. Lincoln
- **uRNN2** : Bentham, G. Washington, subset of IAM, Numen, IBM and A. Lincoln
- **uRNN3** : Bentham, G. Washington, IAM, IBM, A. Lincoln and Numen

The training data and results of these systems with the restricted-track language model are presented on Table 8.14. Adding more data to train the RNNs, we observe up to 26% gain in CER on the validation set of Bentham database with the RNN alone, and 8% of WER improvement of the complete systems. The best RNN is already almost as good as the system combination submitted to the restricted track.

8.3.3.2 Adding Data to the Training of Language Models

For the unrestricted track, we added the Open American National Corpus (Ide & Suderman, 2007) for the estimation of the language models. We used only the part extracted from written documents, which amounts to more than 11M running words.

Table 8.14: BLSTM-RNN unrestricted results (RNN-CER% is the Character Error Rate with RNN alone, while the WER% is after adding the lexicon and LM ; GW and AL stand for G. Washington and A. Lincoln. *"s-"* indicates that only a subset was used) (**Deep**/**Drop.**).

Name	**Training data**	**RNN-CER%**	**WER%**
RNN features	Bentham	8.9	17.2
uRNN1	Bentham, GW, sIAM, sNumen, sAL	7.5	16.5
uRNN2	Bentham, GW, sIAM, sNumen, sIBM, sAL	6.6	15.8
uRNN3	Bentham, GW, IAM, Numen, IBM, AL	6.6	15.8

The new vocabulary, including hyphenations contains 108k words, which represents an OOV rate on the validation set of 2.5%. We built a bigram language model for decoding, of perplexity 345 on the validation data, and a trigram for rescoring, of perplexity 245.

Table 8.15: Improvements brought by adding more LM data (WER% / CER%; **Deep**).

		Restricted LM	**Unrestricted LM**
Deep MLP	*Features*	19.4 / 7.9	16.7 / 6.9
	Pixels	19.9 / 8.2	17.5 / 7.5
Deep RNN	*Features*	17.2 / 6.7	14.9 / 5.7
	Pixels	18.7 / 7.3	16.3 / 6.4
Lattice combination		15.4 / 5.9	12.5 / 4.9
	uRNN1	16.5 / 6.1	13.4 / 5.1
	uRNN2	15.8 / 5.6	13.1 / 4.8
	uRNN3	15.8 / 5.6	13.1 / 4.8
Lattice combination		14.6 / 5.4	11.8 / 4.8

Adding more data for the language model estimation seems critical to solve this task, as shown on Table 8.15 (*"Restricted"* versus *"Unrestricted"* columns). We observe relative WER (*resp.* CER) improvements between 12 and 19% (*resp* 8 and 17%).

The final lattice combination of all systems, with the new language model, achieves 11.8% WER and 4.8% CER, against 15.4% and 5.9% for the system submitted to the restricted track. We notice that the separate improvements due to more optical model data, around 0.8% absolute WER, and to more LM data, around 3% absolute WER, add up to obtain the observed results.

8.3.3.3 Competition Results for the Unrestricted Track

For the unrestricted track, we submitted the single RNNs, and the lattice combination of all systems. Although we got significantly better results than for the restricted track, by more than 25%, our system is not as good as the MDLSTM-RNN of A2iA, which was trained on much more data, and was combined with an open-vocabulary approach. Their language model was moreover estimated on Bentham texts retrieved from the web, which are probably very representative of the content of the manuscripts to recognize for the competition.

Table 8.16: Competition Results for the Unrestricted Track.

Model	WER%
RNN features	14.7
uRNN1	12.9
uRNN2	12.7
uRNN3	12.4
Lattice Combination	11.1
A2iA production system	**8.6**

8.3.4 Post-Evaluation Improvements

After the competition, we brought some refinements to our systems. First, noticing the importance of a good language model in the unrestricted track, we built one with the Bentham texts used in the A2iA system. Then, we had more time training carefully the different neural networks, which corresponds to the experiments presented in Chapters 5 and 6, and yielded better results in some cases. Finally, we have observed in Chapter 6 that applying dropout before LSTM layers rather than after them was better, so we retrained the five RNNs presented above, changing the position of dropout.

8.3.4.1 A More "Author-Specific" Language Model

First, we studied the effect of the corpora used to train the language models. We complete the previous results with a new language model, built with exactly the same procedure as before, described in Section 4.3, but using the same corpus as the A2iA system. It is composed of additional Betham texts retrieved from the web[3]. The new bigram language model has a perplexity of 215 and an OOV rate of 1.47% on the validation set, for a vocabulary of 166k words. The lattices are rescored with a trigram of perplexity 156.

As we can see on Table 8.17, this new LM seems significantly better, bringing another 2% absolute WER improvement compared to the LM built on the OANC. Compared to the language models of the restricted track, this new one produce a relative WER improvement of 34%.

[3] http://oll.libertyfund.org/titles/bentham-works-of-jeremy-bentham-11-vols

Table 8.17: WER% improvements brought by adding even more LM data (**Deep**).

		OANC	**Bentham texts**
Deep MLP	*Features*	16.7	14.0
	Pixels	17.5	14.6
Deep RNN	*Features*	14.9	13.1
	Pixels	16.3	14.4
Lattice combination		12.5	10.7
	uRNN1	13.4	11.9
	uRNN2	13.1	11.3
	uRNN3	13.1	11.3
Lattice combination		11.8	9.7

Table 8.18: Results on the evaluation set.

	Model	**WER%**
Deep MLP	*Features*	13.1
	Pixels	14.3
Deep RNN	*Features*	12.4
	Pixels	13.7
Lattice combination		10.2
	uRNN1	11.1
	uRNN2	10.5
	uRNN3	10.4
Lattice combination		8.9
A2iA production system		**8.6**

When we use the same corpus to train the language model, our combined system is almost as good as A2iA's, even though they used an open-vocabulary approach, which enables their system to recognize any word (Table 8.18), while our result is bounded to be greater than the OOV rate.

8.3.4.2 A More Careful Tuning of Neural Networks

We have seen in Section 6.6 that applying the dropout technique before the LSTM layers in RNNs produced better results than after LSTMs. The new results with this different position of dropout, and with a more careful training and selection of MLPs, are presented on Table 8.19. The language model used in the decoder is the one for the restricted track.

All four systems are improved, by 0.7 to 1.8% absolute WER, and the combination of those is also significantly better than the one submitted to the competition. On Table 8.20, we report the results of our new systems on the evaluation set, for the restricted and unrestricted track, and compare them to the results of the competition (Sánchez et al., 2014).

On the restricted track, the combination of our refined systems outperforms the

Table 8.19: WER% of the refined models (**Deep**).

		Competition	**Refined**
Deep MLP	*Features*	19.4	18.6
	Pixels	19.9	19.2
Deep RNN	*Features*	17.2	16.2
	Pixels	18.7	16.9
Lattice combination		15.4	14.6

results of the winning team. On the unrestricted track, their combination, plus the same corpus to train the language model (although not with the open-vocabulary method of Messina & Kermorvant (2014), used in A2iA's system), achieves the same error rate as the competition winner.

Table 8.20: Refined results on the evaluation set (**Deep**)

(a) Restricted track

Model		**WER%**	**CER%**
Deep MLP	*Features*	18.6	7.5
	Pixels	20.9	8.2
Deep RNN	*Features*	16.2	5.4
	Pixels	16.9	5.9
Lattice combination		**14.1**	**5.0**
CITlab		14.6	-
Ours (Competition)		15.1	-

(b) Unrestricted track

Model		**WER%**	**CER%**
Deep MLP	*Features*	13.2	4.9
	Pixels	14.4	6.1
Deep RNN	*Features*	11.2	4.0
	Pixels	11.5	4.4
	uRNN1	10.9	4.0
	uRNN2	10.5	3.7
	uRNN3	10.2	3.6
Lattice combination		**8.6**	**3.1**
A2iA production system		**8.6**	-
Ours (Competition)		11.1	-

8.4 Conclusion

In this chapter, we have studied the complete hybrid NN/HMM systems, including the decoding with linguistic constraints. We have directly compared the different opti-

cal models, and the different types of inputs. Moreover, we have presented the results of all systems, and of their combination, on the official evaluation set of each database. We have also introduced the systems submitted to the HTRtS 2014 evaluation, and the obtained results.

We have confirmed the observation made in the last chapters, that ***pixel values yield competitive results with handcrafted features*** for optical models made of deep neural networks. Moreover, we have shown that ***deep MLPs can achieve similar performance to RNNs***, despite the dominance of the latter in the literature of handwriting recognition.

However, as seen in the previous chapter, the MLPs need an adjustment of the number of states in HMM models, and of the size of input context provided, and a sequence-discriminative training to attain this level of performance. ***The RNNs, with the CTC training, model sequences of characters directly, and are much easier to train***, coping with the input sequence and the length estimation automatically. No bootstrapping of the training is necessary. Yet for a fair comparison with the two kinds of models, it would be worth training MLPs with dropout and RNNs with sMBR.

We have also underlined the importance of the decoding parameters. ***The optical scale, in particular, has a big influence on the final results***. We found that 0.1 is the best value for MLPs in general, on those databases, and 1 is the best value for RNNs. Tuning the word insertion penalty and the prior scale may bring a few percent of improvement.

The ***linguistic constraints***, *i.e.* the lexicon and the language model, ***are crucial to achieve the presented results***. Without them, the error rates are dramatically higher, especially for MLPs, with which half of the words are misrecognized. With only a lexicon, more than one word out of four is still wrong.

By combining the systems with two methods, ***we have shown the complementarity of the different models***. We achieved over 1% absolute WER improvement compared to the best single system. By considering more alternatives, the lattice-based combination method of Xu et al. (2011) produces robuster results.

Finally, we applied our models to the evaluation sets, in order to compare their results to those published by others. On Rimes and IAM, ***all our single systems yield comparable results to the state-of-the-art***. The proposed model combinations ***outperform all published results on the studied databases***: Rimes, IAM and Bentham.

Conclusions and Perspectives

In this thesis, we focused on the problem of offline, unconstrained, large vocabulary handwritten text recognition, consisting of transforming images of cursive text into their digital transcription. More specifically, we concentrated on images of text lines, for which a segmentation into words or characters was unknown. We adopted the popular sliding window approach: a sequence of feature vectors is extracted from the image, processed by an optical model, and the resulting sequence is modeled by Hidden Markov Models and linguistic knowledge (a vocabulary and a language model) to obtain the final transcription.

In the interest of gaining a deeper knowledge or understanding of these models, we have carried out thorough experiments with deep neural networks optical models for hybrid NN/HMM handwriting recognition. We focused particularly on two popular architectures: Multi-Layer Perceptrons, and Long Short-Term Memory Recurrent Neural Networks. We studied and evaluated many aspects of those models: the type of inputs, the output model, the training procedure, and the architectures of the networks. We validated our approach on three public databases: Rimes, IAM, and Bentham. Our contributions lie in the answers to the following questions regarding neural network optical models.

$\longrightarrow$ *Is it still important to design handcrafted features when using deep neural networks, or are pixel values sufficient?*

Throughout this thesis, we have carried out our experiments with two kinds of inputs: handcrafted features, and raw pixel values. The role of features is to carry a meaningful representation of the input signal. By meaningful, we mean that it should discard the useless variations in the signal, and extract the characteristics that are relevant to the task at hand, here handwriting recognition. Manually designing, implementing, and validating handcrafted features through many experiments, may be a tedious task. Moreover, the choice of features is inevitably biased towards the beliefs and hypotheses of the designer about what is relevant to the problem in the input signal.

The current trend in handwriting recognition, as in many areas of machine learning, is to encode less and less assumptions in the systems, but to let the learning algorithm decide. For example, after trying to segment characters, or words, researchers have switched to whole line recognition, and obtained better results. Likewise, in many areas such as object detection or speech recognition, the extraction of handcrafted, and often popular features is being dismissed, and replaced by the input signal directly, for example the pixel intensities in an image. Deep learning methods have proved quite efficient in processing this raw input, yielding competitive if not better results than classical methods based on features.

Although we have seen that shallow networks tend to be much better when fed with handcrafted features, we showed that ***the discrepancy between the perfor-***

mance of the systems with handcrafted feature and pixel inputs is largely decreased with deep neural networks. With pixel inputs, the gap between one and more hidden layers is bigger than for handcrafted features, suggesting that deep networks are particularly suited to pixels. This supports the idea that an automatic extraction of learnt features happens in the lower layers of the network. ***Neural networks with pixel inputs require more hidden layers, or more parameters, but finally achieve similar performance as networks operating with handcrafted features***. The need to design and implement good feature extractions may therefore not be necessary. Deep networks automatically learn features from the image, without making a priori assumptions about what is relevant to the task.

$\longrightarrow$ *Can deep neural networks give rise to big improvements over neural networks with one hidden layer for handwriting recognition?*

Deep neural networks refer to neural networks with more than one or two hidden layers. For a long period, they were ignored due to the lack of efficient training methods. The only deep architectures found in the literature were convolutional neural networks, which contain a limited number of free parameters thanks to the locality and weight sharing aspects of their architecture. With better resources, more data, better hardware, and new techniques and *"tricks"*, it recently became possible to build deep, densely connected networks, quite simply and efficiently.

Deep neural networks brought significant improvements and reductions of error rates in many areas, including speech recognition and computer vision. While the inclusion of neural networks into HMM based systems, as optical model, has long been adopted and is widespread, both in speech and handwriting recognition, the only use of deep neural networks in continuous handwriting recognition is still limited to convolutional architectures. Deep, densely connected models can be found for isolated character recognition for example, but not for text line recognition of handwritten text.

In this thesis, we have trained two kinds of neural networks, namely Multi-Layer Perceptrons and Recurrent Neural Networks, and we have thoroughly evaluated the influence of the number of hidden layers on the performance of the system. We trained neural networks of different depths, and we have shown that ***deep neural networks achieve significantly better results than neural networks with a single hidden layer***.

It should be noted that increasing the depth of neural networks by adding hidden layers also increases the number of free parameters in the models, and therefore their capacity. By varying the number if units in hidden layers, we can also control the number of parameters. By doing so, we have shown that ***increasing the number of hidden layers yielded better improvements than increasing the number of units in the layer, proving that depth itself produced improvements, which were not only due to the larger number of parameters in deeper networks***. With deep neural networks, we recorded relative improvements of error rates in the range 5-10% for MLPs and 10-15% for RNNs. When the inputs of the network are pixels, the improvement can be much larger.

⟶ *What are the important characteristics of Recurrent Neural Networks, which make them so appropriate for handwriting recognition?*

Recurrent Neural Networks are very popular and achieve state-of-the-art results on many benchmarks of handwriting recognition. They have become a standard component of recognition systems, and receive a lot of research attention nowadays. Many aspects (recurrence, training method) contribute to their success, and we have carried out several experiments to evaluate the importance of different elements.

We have seen that explicitly including context in the observation sequences did not improve the results, as it does for MLPs, and that ***RNNs could effectively learn the dependencies in the input sequences, and the context necessary to make character predictions***. We have observed that the context used by the networks for a given, localized prediction is not limited to the character boundaries in the input signal, and sometimes spans whole words, suggesting that an handwritten character is better modeled in the context of surrounding characters, and that the RNN may learn some linguistic information.

The most evident characteristic of RNNs is the recurrent connections. We evaluated the importance of the recurrence in different layers of the networks, by replacing them with feed-forward layers. We have shown that the ***recurrence was especially useful in the top layers of RNNs***, at least in the CTC framework. The bottom layers, particularly with raw inputs such as pixel values, are more focused on the extraction of elementary features. Having recurrent layers at these positions may not be crucial.

Finally, we have shown that RNNs can take advantage of the CTC framework, which defines an objective function at the sequence level for training, but also the output classes of the network. These are characters directly, and a special *non-character* symbol, allowing the network to produce transcriptions with the neural network alone, without relying on an HMM or any other elaborated model. ***Trained with CTC, the RNNs could produce reasonable character and word error rates, even without a lexicon and a language model***, which is not the case of MLPs.

⟶ *How (deep) Multi-Layer Perceptrons compare to the very popular Recurrent Neural Networks, which are now widespread in handwriting recognition and achieve state-of-the-art performance?*

With the advent of RNNs, and their success in handriting recognition, MLPs tend to be neglected by the community. However, in speech recognition, a field where the systems are quite similar – HMM-based systems, with signal processing, feature extraction, acoustic and language modeling – deep MLPs have become the standard acoustic model, yielding a considerable drop of error rates, unseen for years. In this thesis, we have studied deep MLPs, and compared their results with those of RNNs.

We have shown that ***deep MLPs can achieve similar performance to RNNs***, and that both kinds of model give comparable results to the state-of-the-art on Rimes and IAM. We conclude that, despite the dominance of RNNs in the literature of handwriting recognition, MLPs, and possibly other kind of models, can be a good alternative, and therefore should not be put aside.

However, we have also shown that MLPs are more sensitive to the number of states in HMM models, and to the amount of input context provided. ***The RNNs, with***

CTC training, model sequences of characters directly, and are much easier to train, coping with the input sequence and the length estimation automatically.

Finally, most of our work on the Bentham database took place during the HTRtS contest. We had a limited amount of time to build and train the systems to be submitted to the competition. On this database, we report better results with RNNs than with MLPs. It required less effort to achieve good results with RNNs, for the reasons evoked above, and might explain the popularity of those models.

⟶ *What are the good training strategies for Neural Networks for handwriting recognition? Can the Connectionist Temporal Classification paradigm be applied to other Neural Networks? What improvements can be observed with a discriminative criterion at the sequence level?*

The optimized cost is an important feature of the training procedure of models with machine learning algorithms, and it may affect the quality of the system. The most common approach to train neural networks for hybrid NN/HMM systems consists in first aligning the frames to HMM states with a bootstrapping system, and then train the network on the obtained labeled dataset with a framewise classification cost function, such as the cross-entropy.

This strategy amounts to considering the segmentation of the input sequence into HMM states fixed, and to have the network predict it. A softer approach, similar to the Baum-Welsch training algorithm, would consist in ***summing over all possible segmentations of the input sequences yielding the same final transcription***. We have seen that in general, ***this approach produces only small improvements***.

The CTC framework is such a training procedure, but also defines the outputs of the neural network to correspond to the set of characters, and a special *non-character* output (*blank* label). We have shown that ***the typical CTC output, consisting mainly of blank predictions, and of localized peaks for characters, is due to the interaction between the training criterion and the blank label***, rather than by RNNs. Yet, RNNs are especially good to cope with this target output, and can achieve very good results with the CTC criterion. ***MLPs, which do not have recurrent connexions, can be trained with CTC but do not benefit from it***.

We have studied the effects of applying a discriminative training criterion at the sequence level, namely state-level Minimum Bayes Risk (sMBR). This kind of ojective function reflects the goal of recognition: to retrieve the most likely sequence of words. We have shown that ***fine-tuning the MLPs with sMBR yields significant improvements***, between 5 and 13% of WER, which is consistent with the speech recognition literature.

Moreover, we investigated a new regularization technique, dropout, in RNNs, extending the work of Pham et al. (2014); Zaremba et al. (2014). More specifically, we studied different positions for dropout in the networks, in particular its relative position to LSTM layers, and reported significant ***improvements over the method presented in (Pham et al., 2014) when dropout is applied before LSTM layers*** rather than after them.

Finally, all our models achieved error rates comparable to the state-of-the-art on Rimes and IAM, independently of the type of inputs (handcrafted features or pixels), and of the kind of neural network (MLP or RNN). ***The lattice combination of our systems, with the method of Xu et al. (2011), outperformed the best published systems for all three databases***, showing the complementarity of the developed models. We validated our approach by taking part to the HTRtS 2014 contest, and ranked second in both tracks of the competition.

Beside depth, a significant part of the improvements were caused by sequence-discriminative training of MLPs, and dropout in RNNs. A natural continuation of this work would be to apply sequence-discriminative training to RNNs and dropout to MLPs. Moreover, for an industrial application of our findings, we should apply the presented systems and models to more difficult and realistic scenarios, such as the Maurdor data, which contains heterogeneous, real-life documents, with complex layouts and three different languages.

This thesis was conducted in an industrial context, as a continuation of several other theses at A2iA (MLP/HMM hybrid system for word recognition with an explicit grapheme segmentation (Augustin, 2001); GMM/HMM system for word recognition using the sliding window technique (Bianne-Bernard, 2011)). The results presented here improve those of the latest one (Bianne-Bernard, 2011), even for GMM/HMMs, partly due to the application of language models after recognizing whole lines, instead of isolated words. Deep neural networks also brought huge improvements. The next step to use these models in industrial applications would consist in accelerating them, for example by taking advantage of the knowledge acquired through some of the presented experiments, such as removing the recurrent connections in lower layers, or building deeper but skinnier networks.

Most of the improvement observed in the last few years is due to several factors:

- replacing GMMs by discriminative models, such as RNNs as the emission model in HMMs
- modeling the sequence of character directly with CTC training of RNNs, rather than using multi-state character HMMs
- making the language more flexible, in particular combining language models at the word and character levels to alleviate the out-of-vocabulary (OOV) words problem

There has been some recent works on image pre-processing (Kozielski et al., 2012). While aiming at reducing the variability of the inputs, pre-processing and feature extraction eliminates some of the information. As models and machine learning techniques become more powerful and reliable, these steps become less and less important to obtain good results, and not worth the effort. The current tends is to make less assumptions, and using the pixel values of the raw images directly may be sufficient.

We observed that big improvements in the optical models did not transfer to the results of complete systems. It looks like the limiting factor is the language model. When the language model is good enough, it may correct many of the mistakes made by the optical model. Conversely, although building an appropriate language model in the constrained problems of public databases (mails similar to each other in Rimes, extracts of the Lancaster-Oslo-Bergen corpus in IAM, or collection of texts from the same author in Bentham) is possible, estimating the language model suited to industrial applications may be much more difficult.

We observed on IAM that half of the remaining errors are caused by OOVs. Now, the main problem might not be so much *how to improve the optical model* as *how can be build better language models*. Among the huge improvements observed in the last few years, a significant part was due to a better language modeling. Increasing the size of the vocabulary and of the corpus used in their estimation is only a partial solution, and several methods were developed to cope with out-of-vocabulary words, directly at decoding time (Kozielski et al., 2013b; Messina & Kermorvant, 2014), or as a post-processing (Oprean et al., 2013).

A significant part of the errors also concerned punctuation. While tuning the word insertion penalty decreased the error rates, it particularly affected punctuation symbols, which are small, limited to one character, and probably more difficult to predict with standard language models.

We believe that future improvements in handwriting recognition, both in research and in industry, may come from new ways of integrating language models and linguistic knowledge sources in the system. Some propositions were made for example in (Maas et al., 2014; Graves et al., 2013a). Moreover, Recurrent Neural Networks can already learn some dependencies between characters, and character language models might be sufficient.

The problem of handwriting recognition is still far from being solved. The current systems rely on a line segmentation, which may not be available, especially in industrial applications, and is not always straightforward to obtain. The development of end-to-end recognition systems, which, from the document image, and without making any assumption about the line positions, are able to produce their transcription, is a very exciting direction of research, in the vein of the evolution of handwriting recognition systems. Another interesting line of work would be the improvement of unsupervised training methods, lessening the need for human transcriptions, costly to obtain, for example inspired by the work of Kozielski et al. (2014b).

List of Publications

The work carried out during this PhD lead to several publications in conference proceedings. For the sake of consistency, not all published papers are covered by this thesis. For the most part, the work presented here extends the content of (Bluche et al., 2014c). Concerning specific points, Chapter 7 gives more details on the experiments presented in (Bluche et al., 2015c), and the study of dropout in RNNs, detailed in Section 6.6, was published in (Bluche et al., 2015a). The system submitted to the HTRtS contest in 2014 is overviewed in (Bluche et al., 2015b).

We started with the recognition of isolated handwritten words, from Rimes and IAM databases, using Convolutional Neural Networks (ConvNNs), before switching to line recognition with deep MLPs and RNNs.

In (Bluche et al., 2013b), we compare a tandem and an hybrid approach to combining HMM and ConvNNs. The results show that tandem models tend to produce better word recognition results. Combined with context-dependent, discriminatively trained GMM-HMMs, we obtained comparable error rates to RNNs.

In (Bluche et al., 2013a), we extend that work by considering different segmentations of images: with a sliding window, and with an explicit grapheme segmentation. While the later approach yields lower performance, the models are much faster and suited to industrial applications as a first-pass recognition system.

We also took part and won international evaluations with the research team at A2iA. The systems, based on Multi-Dimensional Long Short-Term Memory RNNs, are described in (Bluche et al., 2014a) for the OpenHaRT'2013 evaluation (recognition of handwritten Arabic documents), and in (Moysset et al., 2014) for the MAURDOR evaluation in 2013 (recognition of handwritten and printed texts in heterogeneous documents, in French, English, and Arabic).

Moreover, we developed a method to jointly segment document images and map their transcript, allowing to automatically create ground-truth material at the line level, which is required to train the recognition systems. The method, described in (Bluche et al., 2014b), was used during the MAURDOR evaluation, and during the HTRtS contest, for the unrestricted track presented in Section 8.3.3.

It was also applied for the ORIFLAMMS project, aiming at studing the shapes of letters, and their variation across space and time. The work carried out in this project lead to a publication to the Digital Humanities 2015 conference (Stutzmann et al., 2015).

Finally, we contributed to the experiments presented in (Louradour et al., 2012; Kermorvant et al., 2012), dealing with the classification of documents using their automatic transcripts along with the text recognition confidence score, and in (Pham et al., 2014), the first work applying the dropout technique to LSTM networks.

Publications:

Bluche, T., Kermorvant, C., & Louradour, J. (2015a). Where to Apply Dropout in Recurrent Neural Networks for Handwriting Recognition? In *13th International Conference on Document Analysis and Recognition (ICDAR)*, (pp. –). IEEE.

Bluche, T., Louradour, J., Knibbe, M., Moysset, B., Benzeghiba, M. F., & Kermorvant, C. (2014a). The A2iA Arabic Handwritten Text Recognition System at the Open HaRT2013 Evaluation. In *11th IAPR International Workshop on Document Analysis Systems (DAS)*, (pp. 161–165). IEEE.

Bluche, T., Moysset, B., & Kermorvant, C. (2014b). Automatic Line Segmentation and Ground-Truth Alignment of Handwritten Documents. In *14th International Conference on Frontiers in Handwriting Recognition (ICFHR2014)*, (pp. 667–672).

Bluche, T., Ney, H., & Kermorvant, C. (2013a). Feature Extraction with Convolutional Neural Networks for Handwritten Word Recognition. In *12th International Conference on Document Analysis and Recognition (ICDAR)*, (pp. 285–289). IEEE.

Bluche, T., Ney, H., & Kermorvant, C. (2013b). Tandem HMM with convolutional neural network for handwritten word recognition. In *17th International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (pp. 2390–2394). IEEE.

Bluche, T., Ney, H., & Kermorvant, C. (2014c). A Comparison of Sequence-Trained Deep Neural Networks and Recurrent Neural Networks Optical Modeling for Handwriting Recognition. In *International Conference on Statistical Language and Speech Processing*, (pp. 199–210).

Bluche, T., Ney, H., & Kermorvant, C. (2015b). The LIMSI Handwriting Recognition System for the HTRtS 2014 Contest. In *13th International Conference on Document Analysis and Recognition (ICDAR)*, (pp. –). IEEE.

Bluche, T., Ney, H., Louradour, J., & Kermorvant, C. (2015c). Framewise and CTC Training of Neural Networks for Handwriting Recognition. In *13th International Conference on Document Analysis and Recognition (ICDAR)*, (pp. –). IEEE.

Kermorvant, C., Bianne-Bernard, A.-L., Bluche, T., & Louradour, J. (2012). On using alternative recognition candidates and scores for handwritten documents classification. Tech. Rep. A2iA-RR-2012-1, A2iA.

Louradour, J., Bluche, T., Bianne-Bernard, A.-L., Menasri, F., & Kermorvant, C. (2012). De l'usage des scores et des alternatives de reconnaissance pour la classification d'images de documents manuscrits. In *Colloque International Francophone sur l'Ecrit et le Document (CIFED)*.

Moysset, B., Bluche, T., Knibbe, M., Benzeghiba, M. F., Messina, R., Louradour, J., & Kermorvant, C. (2014). The A2iA Multi-lingual Text Recognition System at the second Maurdor Evaluation. In *14th International Conference on Frontiers in Handwriting Recognition (ICFHR2014)*, (pp. 297–302).

Pham, V., Bluche, T., Kermorvant, C., & Louradour, J. (2014). Dropout improves recurrent neural networks for handwriting recognition. In *14th International Conference on Frontiers in Handwriting Recognition (ICFHR2014)*, (pp. 285–290).

Stutzmann, D., Bluche, T., Lavrentev, A., Leydier, Y., & Kermorvant, C. (2015). From Text and Image to Historical Resource: Text-Image Alignment for Digital Humanists. In *Digital Humanities (DH) – to appear*.

Appendix A

Databases

A.1 IAM

The IAM database, described in details in (Marti & Bunke, 2002) consists of images of handwritten documents. They correspond to English texts exctracted from the electronically available LOB corpus (Johansson, 1980). Fifteen categories have been selected, form which an overall amount of 500 texts where kept. Segments of a few sentences were extracted, and given to several writers to write by hand. The only constraint was to copy the printed text; writers were not required to write on lines, or at a specific size. 657 writers contributed to the database, producing between 1 and 59 handwritten documents. The annotation and segmentation of the database is provided at the word, line and paragraph level. We focused on the line recognition problem.

The database is divided into several subset (a-f,u,x), and the online version [1] comes with training, validation and evaluation splits. However, these are not the splits used for handwriting recognition experiments reported on the litterature. The community opted for an alternative splitting into 747 images for training, 116 for validation, and 336 for evaluation. In this setup, the sets of writers contributing for each dataset are disjoint.

In the following, we give more details on the images of the datasets, as well as on the corresponding text corpora. We also present a selection of published results and breifly present the systems.

Images

The images of the database are gray-level TIFF (Tagged Image File Format) images. They are produced by scanning the written documents at a resolution of 300dpi. They contain the printed text to copy, and the handwritten text. Provided are the lines bounding boxes and transcript. Some basic statistics extracted from the database are presented in Table A.1. We observe that these statistics are quite stable from one set to the other.

[1] http://www.iam.unibe.ch/fki/databases/iam-handwriting-database

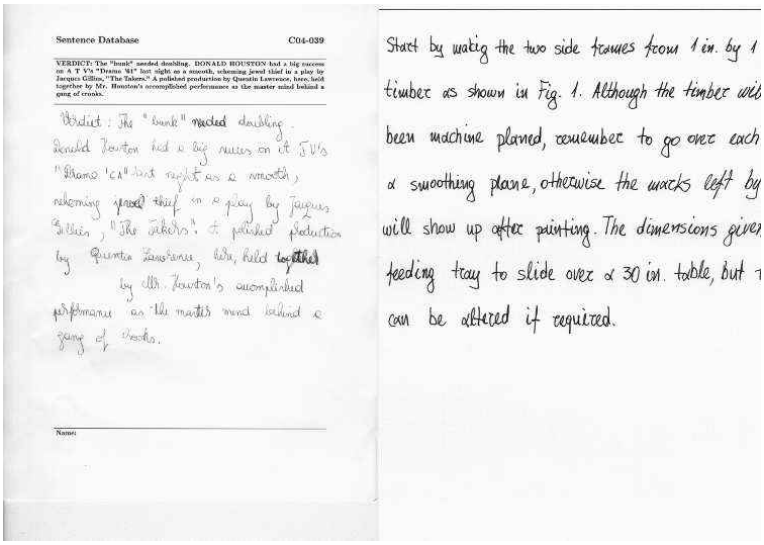


Figure A.1: Examples from IAM Database (the text to copy corresponds to the typed header paragraph).

Corpus

The text of the IAM database corresponds to texts extracted from the LOB corpus, which have some controlled linguistic properties. In Table A.2, we present some numbers about the text of the IAM Database, such as the number of words and characters in each subset, the number of unique words and characters, as well as some occurence statistics.

We checked that each character in the validation and evaluation set are also present in the training set, so that a system modeling characters can theoretically recognize all the words in the data. However, some words in the validation and evaluation set do not occur in the training set.

Published Results

In Table A.3, we present a list of published results on the IAM database, for comparison with the results presented in this thesis. Some papers gave several results (*e.g.* GMM-HMM and LSTM results), but we only kept the best ones here. Also, they did not all report figures like Character Error Rates or results on the validation data.

We notice a drop in error rates in the last couple of years, mainly due to improvement of the language models, and the creation of open-vocabulary decoders.

(Bertolami & Bunke, 2008) — The slant and skew in the images are corrected and the baseline position and average character width normalized. For each column of pixels, nine geometrical features are extracted (foreground pixels, moments, contour positions, black-white transitions). Ensemble methods (bagging, feature subspace) are applied to train several HMM recognition systems, which are then combined with ROVER. A bigram language model, trained on the LOB, Brown and Wellington corpora, and limited to the 20k most frequent words is applied.

Table A.1: Documents of the IAM database, in each dataset. The *Total* column are accumulated numbers, while the average, min/max and quartiles are computed per line. The width/char measure is obtained by dividing, for each line, the width by the number of characters in the annotation, giving only a rough estimation.

Train	**Total**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Pages	747	-	-	-	-	-	-
Lines(/page)	6,482	8.7	3.0	7.0	9.0	10.0	13.0
Words(/line)	55,081	8.5	1.0	7.0	8.0	10.0	19.0
Chars(/line)	287,727	44.4	2.0	39.0	44.0	50.0	81.0
Line Width (px)	10,995,371	1,696.3	104.0	1,668.0	1,754.0	1,830.1	2,260.0
Line Height (px)	794,758	122.6	44.0	99.0	119.0	142.0	342.0
Width/Char (px)	253,638	39.1	18.4	34.1	39.1	43.7	122.6
Validation	**Total**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Pages	116	-	-	-	-	-	-
Lines(/page)	976	8.4	2.0	7.0	8.0	10.0	12.0
Words(/line)	8,895	9.1	2.0	8.0	9.0	11.0	18.0
Chars(/line)	43,050	44.1	5.0	39.0	44.0	50.0	73.0
Line Width (px)	1,636,386	1,676.6	100.0	1,653.0	1,745.5	1,821.0	2,146.0
Line Height (px)	111,841	114.6	45.0	95.0	115.0	131.0	218.0
Width/Char (px)	37,761	38.7	20.0	34.5	38.0	42.7	70.5
Evaluation	**Total**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Pages	336	-	-	-	-	-	-
Lines(/page)	2,915	8.7	4.0	7.0	9.0	10.0	13.0
Words(/line)	25,920	8.9	1.0	7.0	9.0	10.0	22.0
Chars(/line)	128,531	44.1	6.0	38.0	44.0	50.8	94.0
Line Width (px)	4,983,925	1,709.8	150.0	1,683.2	1,770.0	1,844.0	2,180.0
Line Height (px)	379,209	130.1	38.0	104.0	128.0	153.0	337.0
Width/Char (px)	116,461	40.0	10.1	34.0	39.5	44.7	73.7

Table A.2: Text of the IAM database. The averages, quartiles, and min/max values are calculated from occurences of each token, *e.g.* 75% of the characters appear at least 3,511 times in the training set.

Train	**Total**	**Unique**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Words	55,111	7,843	7.0	1.0	1.0	1.0	3.0	2889.0
Chars	236,588	77	3072.6	2.0	94.4	390.0	3511.2	28050.0
Validation	**Total**	**Unique**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Words	8,845	2,448	3.6	1.0	1.0	1.0	2.0	442.0
Chars	34,767	75	463.6	1.0	17.6	50.0	629.4	3952.0
Evaluation	**Total**	**Unique**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Words	25,862	5,359	4.8	1.0	1.0	1.0	2.0	1322.0
Chars	104,473	73	1431.1	2.0	46.7	149.0	1897.4	12008.0

(Graves et al., 2009) — After extracting the line images, the skew and slant are corrected, the height of the ascenders, descenders and core regions normalized, and the

Table A.3: Selection of published results on IAM database

	Dev.		Eval.	
	WER	**CER**	**WER**	**CER**
Doetsch et al. (2014)	**8.4**	**2.5**	**12.2**	**4.7**
Kozielski et al. (2013a)	9.5	2.7	13.3	5.1
Pham et al. (2014)	11.2	3.7	13.6	5.1
Kozielski et al. (2013a)	11.9	3.2	-	-
Messina & Kermorvant (2014)	-	-	19.1	-
Espana-Boquera et al. (2011)	19.0	-	22.4	9.8
Toselli et al. (2010)	-	-	25.8	-
Graves et al. (2009)	-	-	25.9	18.2
Dreuw et al. (2011a)	22.7	6.1	28.8	10.1
Dreuw et al. (2011b)	23.7	6.5	29.2	10.3
Bertolami & Bunke (2008)	26.8	-	32.8	-

width is adjusted to normalize the mean character width. Nine geometrical features (contours, pixel transitions, mean value, center of gravity) are extracted from each column of pixels, and are given to a BLSTM-RNN, with CTC training. For decoding a bigram language model estimated on the LOB, Brown and Wellington corpora, and limited to the 20k most frequent words is applied.

(Toselli et al., 2010) — The skew and slant in the images are corrected and the height normalized to a fixed value for ascenders, descenders and core region. Then feature vectors of dimension 60 are extracted from a from a grid on the image (normalized gray level and vertical and horizontal derivatives). The recognition system is a GMM/HMM, and a bigram language model, trained on the LOB corpus is applied. The vocabulary contains all words present in the training and test sets.

(Dreuw et al., 2011b) — The images are deslanted and the size of ascenders and descenders normalized. They are then scaled to 16px high and appearence-based features and their horizontal derivatives are extracted from a sliding window scanned through the image. Seven consecutive frames are concatenated and the feature vector is reducted by PCA to 30 dimensions. The recognition system is a GMM/HMM, trained with confidence and margin-based MPE, and a trigram language model estimated on the LOB, Brown and Wellington corpora, and limited to the 50k most frequent words.

(Dreuw et al., 2011a) — This system is the same as the previous one, except that the features are used to train an MLP to predict HMM state posterior probabilities. The log-posteriors are used as the features of the M-MPE trained GMM/HMM.

(Espana-Boquera et al., 2011) — Different MLPs are trained and applied to the line image to enhance the contrast, correct the slope, the slant, and normalize the height. The normalized gray level and vertical and horizontal derivatives are extracted from a grid on the image, and each column of the grid is used to build the feature vector. The recognition system is an hybrid MLP/HMM, and the decoding is carried out with a bigram language model estimated on the LOB, Brown and Wellington corpora, and limited to the 20k most frequent words.

(Kozielski et al., 2013a) — The contrast and slant in the images are corrected. A

cosine window is scanned through the line image, and the obtained frames are normalized with respect to the first and second-order moments (translation of the center of gravity, standard-deviation normalization). The feature vectors are obtained by extracting the pixel values and applying a 20 dimensional PCA, and appending four moment features. A BLSTM-RNN is trained on these features to predict HMM states, and the activations of the first hidden layer are extracted and reduced to 20 dimensions by PCA. From these features, a GMM-HMM is trained (tandem approach) with writer adaptation and discriminative training. An open-vocabulary language model is used for decoding, consisting of a trigram word LM estimated on the LOB, Brown and Wellington corpora, and limited to the 50k most frequent words, and a 10gram character language model trained to represent OOV words.

(Doetsch et al., 2014) — This system is the same as the previous one, except for a modification of the gating mechanism in LSTMs, and for the integration of the RNN. It is an hybrid system where the RNN is directy used as an estimate of the emission model of the HMM.

(Pham et al., 2014) — The line images are passed through an MDLSTM-RNN, trained with CTC, without any preprocessing besides the mean and variance normalization of the pixel values. The dropout technique is applied after each LSTM layer, and the decoding is performed with a trigram language model estimated on the LOB, Brown and Wellington corpora, and limited to the 50k most frequent words.

(Messina & Kermorvant, 2014) — The system is the same as the previous one except for the dropout (not present in this paper) and for the language model, which here contains all the 106k unique words of the LOB, Brown and Wellington corpora.

A.2 Rimes (ICDAR 2011 setup)

Description

The Rimes database (Augustin et al., 2006) consists of images of handwritten paragraphs from simulated French mail. The goal is to evaluate the quality of automatic mail processing for companies and administrations. The contributors were asked to write letters according to predefined scenarios among nine themes (*e.g.* information request, complaint, damage declaration), leaving some freedom in the actual content of the letter and in its layout. The setup for the ICDAR 2011 competition is limited to the core of the letters. It defines a training set of 1,500 paragraphs, and an evaluation set of 100 paragraphs[2]. We held out the last 149 images from the training set for system validation.

In the following, we give more details on the images of the datasets, as well as on the corresponding text corpora. We also present a selection of published results and breifly present the systems.

[2]The competition had a validation and a test set of 100 documents (Grosicki & El-Abed, 2011), but the evaluation set is not publicly available, and published papers use the validation set, which we hence call evaluation set in this thesis, as oposed to the validation set we manually defined from the training set.

Images

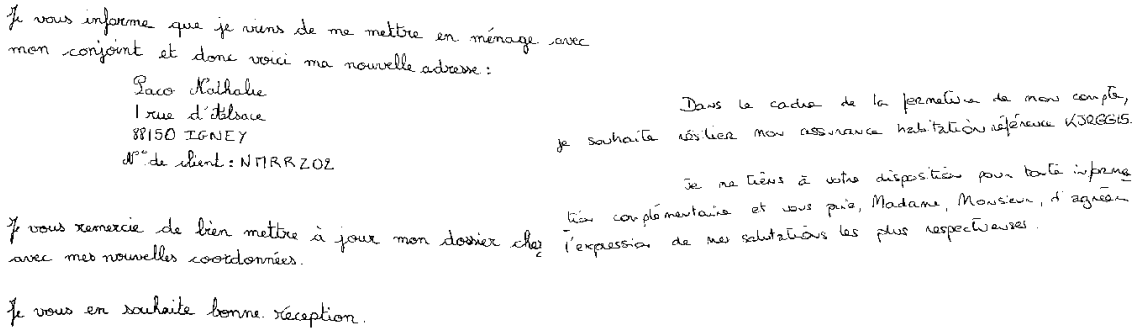


Figure A.2: Examples from Rimes Database.

The images of the database are gray-level images, cropped to the paragraph location. They are produced by scanning the written documents at a resolution of 300dpi. They contain the printed text to copy, and the handwritten text. The transcript of paragraphs is available, including the carriage return and the spelling error. The main difficulty in these images, compared to IAM for example, is the variety of layouts, with many irregular and not straight lines. Some basic statistics extracted from the database are presented in Table A.4. We observe that these statistics are quite stable from one set to the other.

Corpus

The text of the Rimes database is only limiteded by the scenario given to the contributors. Therefore, it is rather free despite a semantic constraint. Moreover, because of the scenarios, and of the fact that the documents are letters addressed by individuals to companies and administration, the language employed is also redundant, with a predominance of the first person, and of formulations of politeness, requests and wishes. The main difficulty lies in the fair amount of proper name, addresses, dates, and codes, prone to induce out-of-vocabulary words.

In Table A.5, we present some numbers about the text of the Rimes Database, such as the number of words and characters in each subset, the number of unique words and characters, as well as some occurence statistics.

Published Results

In Table A.3, we present a list of published results on the IAM database, for comparison with the results presented in this thesis. Some papers gave several results (*e.g.* GMM-HMM and LSTM results), but we only kept the best ones here. Note that since the official evaluation set has not been publicly released, the results presented on the literature are obtained on the validation set. Some systems were tuned on this set,

Table A.4: Documents of the Rimes database, in each dataset. The *Total* column are accumulated numbers, while the average, min/max and quartiles are computed per line. The width/char measure is obtained by dividing, for each line, the width by the number of characters in the annotation, giving only a rough estimation.

Train	**Total**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Pages	1,351	-	-	-	-	-	-
Lines(/page)	10,203	7.6	2.0	6.0	7.0	9.0	17.0
Words(/line)	73,822	7.2	0.0	5.0	8.0	9.0	18.0
Chars(/line)	460,201	45.1	0.0	34.0	47.0	57.0	110.0
Line Width (px)	16,642,312	1631.1	96.0	1333.0	1821.0	2039.0	2502.0
Line Height (px)	1,323,409	129.7	35.0	104.0	126.0	152.0	365.0
Width/Char (px)	383,532	37.6	0.0	32.3	36.4	41.5	1774.0
Validation	**Total**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Pages	149	-	-	-	-	-	-
Lines(/page)	1,130	7.6	3.0	6.0	7.0	9.0	15.0
Words(/line)	8,380	7.4	1.0	5.0	8.0	10.0	17.0
Chars(/line)	51,924	46.0	3.0	36.0	48.0	58.0	89.0
Line Width (px)	1,879,538	1663.3	125.0	1408.9	1864.0	2066.1	2470.0
Line Height (px)	148,050	131.0	48.0	105.0	127.0	152.0	319.0
Width/Char (px)	42,327	37.5	23.2	31.9	36.3	41.4	344.6
Evaluation	**Total**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Pages	100	-	-	-	-	-	-
Lines(/page)	778	7.8	4.0	6.0	8.0	9.0	18.0
Words(/line)	5,639	7.2	1.0	5.0	8.0	9.0	18.0
Chars(/line)	35,286	45.4	5.0	34.0	47.5	58.0	98.0
Line Width (px)	1,296,956	1667.0	133.0	1422.2	1867.5	2097.1	2432.0
Line Height (px)	100,592	129.3	46.0	103.0	124.0	152.0	296.0
Width/Char (px)	30,077	38.7	6.5	32.2	37.4	43.4	233.6

Table A.5: Text of the Rimes database. The averages, quartiles, and min/max values are calculated from occurences of each token, *e.g.* 75% of the characters appear at least 3,511 times in the training set.

Train	**Total**	**Unique**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Words	73,822	8,061	9.2	1.0	1.0	1.0	3.0	4630.0
Chars	395,186	98	4032.5	1.0	69.8	319.5	2152.6	57439.0
Validation	**Total**	**Unique**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Words	8,380	2,036	4.1	1.0	1.0	1.0	2.0	518.0
Chars	44,529	88	506.0	1.0	12.9	45.0	284.1	6411.0
Evaluation	**Total**	**Unique**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Words	7,631	1,246	6.1	1.0	1.0	1.0	3.0	376.0
Chars	30,325	85	356.8	1.0	9.7	36.0	193.6	4339.0

while we chose to isolate a part of the training data to define a new validation set. As for IAM, the CER is not always reported in papers.

Table A.6: Published results on Rimes database

	WER	CER
Pham et al. (2014)	**12.3**	**3.3**
Doetsch et al. (2014)	12.9	4.3
Messina & Kermorvant (2014)	13.3	-
Kozielski et al. (2013a)	13.7	4.6
Messina & Kermorvant (2014)	14.6	-
Menasri et al. (2012)	15.2	7.2
Telecom ParisTech (Grosicki & El-Abed, 2011)	31.2	18.0

Telecom ParisTech (Grosicki & El-Abed, 2011) — Lines are extracted from document images, deslanted, and binarized. Components belonging to surrounding lines are removed, and 28 geometrical and statistical features are extracted from a sliding window (mean pixel values, center of gravity, pixel configurations, black/white transitions). The recognition system is a context-dependent GMM/HMM modeling trigraphs, and a bigram language model estimated from the training transcriptions.

(Menasri et al., 2012) — The line images are automatically segmented into words, keeping several hypotheses represented as a graph. This system is a combination of three MDLSTM-RNNs trained on the word images directly, with CTC and a different initialization for each, and a grapheme based MLP-HMM hybrid, using 74 features extracted from an explicit over-segmentation of the word image. Each word in the segmentation graph is recognized and scored with the combination of models, and the best path is extracted after applying a bigram language model estimated from the training set annotations, with a vocabulary of 6k words.

(Kozielski et al., 2013a) — The contrast and slant in the images are corrected. A cosine window is scanned through the line image, and the obtained frames are normalized with respect to the first and second-order moments (translation of the center of gravity, standard-deviation normalization). The feature vectors are obtained by extracting the pixel values and applying a 20 dimensional PCA, and appending four moment features. A BLSTM-RNN is trained on these features to predict HMM states, and the activations of the first hidden layer are extracted and reduced to 20 dimensions by PCA. From these features, a GMM-HMM is trained (tandem approach) with writer adaptation and discriminative training. A 4gram language model estimated on the training annotations is used for decoding.

(Doetsch et al., 2014) — This system is the same as the previous one, except for a modification of the gating mechanism in LSTMs, and for the integration of the RNN. It is an hybrid system where the RNN is directy used as an estimate of the emission model of the HMM.

(Pham et al., 2014) — The line images are passed through an MDLSTM-RNN, trained with CTC, without any preprocessing besides the mean and variance normalization of the pixel values. The dropout technique is applied after each LSTM layer, and the decoding is performed with a 4gram language model estimated on the annotations of the training set, and limited to the 5k most frequent words.

(Messina & Kermorvant, 2014) — The system is the same as the previous one except for the dropout (not present in this paper) and for the language model, which here is an hybrid word/character language model for an open-vocabulary approach based on a static decoding graph, with a trigram word language model limited to the most frequent 3k words in the training annotations, and a 10gram character language model.

A.3 Bentham (HTRtS 2014 setup)

Description

This database contains images of personal notes of the British philosopher Jeremy Bentham, written by himself or copied by his staff in English, around the 18th and 19th centuries. The data were prepared by University College, London, during the tranScriptorium project[3](Sánchez et al., 2013). The transcription of Bentham manuscripts is a collaborative project (Causer & Wallace, 2012) and is carried out by amateur volunteers, and reviewed by experts.

In the setup of the HTRtS competition (Sánchez et al., 2014), there are a training set of 350 pages, a validation set of 50 images, and a test set of 33 pages. The data available for this competition contains lines positions and annotations, obtained by an automatic mapping of the transcripts (Gatos et al., 2014) In the following, we give more details on the images of the datasets, as well as on the corresponding text corpora. We also present a selection of published results and breifly present the systems.

Images

The images of the database are color images scanned at 300 DPI, and provided in JPG format. Some examples are shown on Figure A.3. They contains many difficulties, compared to the previous two databases, such as side-notes, stamps, fainted ink, or inclusion of words between lines. Some parts of the text were crossed out, and sometimes transcribed anyway, meaning the system has to recognize them despite the optical difficulty, but also the complication for a language model, and sometimes they are trascribed as a special *gap* token. Moreover, Bentham has a handwriting that is hard to read[4], compensating the few number of different writers for this database.

Corpus

The corpus of Bentham database is made of a limited number of topics, addressed by Bentham (*e.g.* the constitution, the panopticon prison scheme), and is entirely written in Bentham's prose, even his staff contributed to the writing. However, it contains some difficulties. Besides the crossed out words that are nonetheless transcribed, for which the language model may have trouble, there are also words in French in Latin,

[3] http://transcriptorium.eu/

[4] http://www.transcribe-bentham.da.ulcc.ac.uk/td/Help:Examples_of_Bentham%27s_handwriting

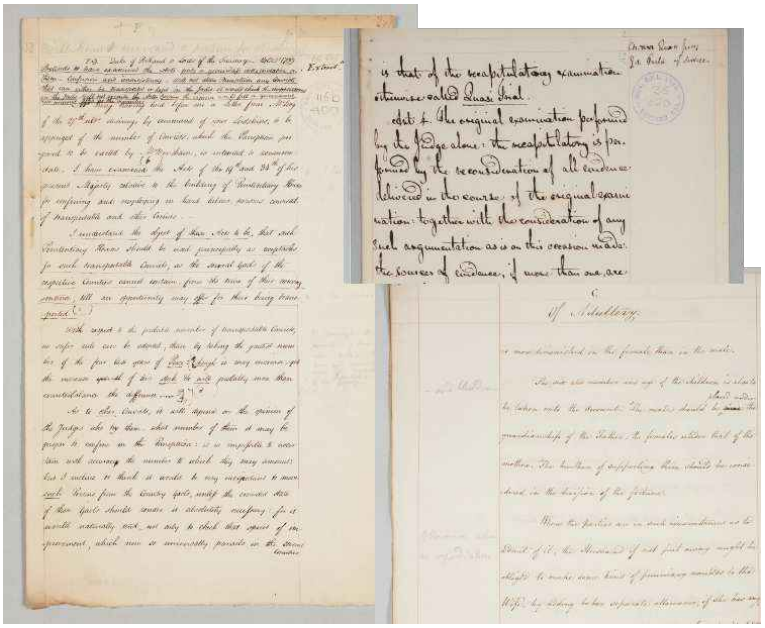


Figure A.3: Examples from Bentham Database.

Table A.7: Documents of the Bentham database, in each dataset.

Train	**Total**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Pages	350	-	-	-	-	-	-
Lines(/page)	9,198	26.3	3.0	23.0	26.5	30.0	55.0
Words(/line)	76,707	8.3	1.0	7.0	9.0	11.0	19.0
Chars(/line)	419,764	45.6	1.0	40.0	52.0	60.0	102.0
Line Width (px)	12,767,586	1,388.1	21.0	1,383.2	1,666.0	1,724.0	2,448.0
Line Height (px)	1,084,323	117.9	20.0	96.0	115.0	137.0	407.0
Width/Char (px)	300,673	32.7	5.1	27.4	30.3	35.0	273.0
Validation	**Total**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Pages	50	-	-	-	-	-	-
Lines(/page)	1,415	28.3	12.0	26.0	28.0	31.0	40.0
Words(/line)	11,580	8.2	1.0	7.0	9.0	10.0	17.0
Chars(/line)	64,070	45.3	1.0	43.0	50.0	57.0	82.0
Line Width (px)	2,066,281	1,460.3	31.0	1,504.5	1,681.0	1,761.5	2,463.0
Line Height (px)	180,092	127.3	28.0	100.0	122.0	148.0	384.0
Width/Char (px)	48,116	34.0	17.8	28.6	32.8	37.1	110.0

and Bentham used alternative spellings of some words[5]. The main problem, however, is hyphenation, which is quite frequent, and must be addressed to prevent the number of out-of-vocabulary words to explode.

In Table A.8, we present some numbers about the text of the IAM Database, such as the number of words and characters in each subset, the number of unique words and characters, as well as some occurence statistics.

[5] http://www.transcribe-bentham.da.ulcc.ac.uk/td/Alternative_spellings

Table A.8: Text of the Bentham database.

Train	**Total**	**Unique**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Words	87,146	8,518	10.2	1.0	1.0	1.0	3.0	5,162.0
Chars	347,393	92	3,776.0	1.0	114.4	338.5	2,135.0	42,769.0
Validation	**Total**	**Unique**	**Average**	**Min.**	**25%**	**Median**	**75%**	**Max.**
Words	13,269	2,627	5.1	1.0	1.0	1.0	2.0	855.0
Chars	53,136	85	625.1	1.0	24.0	56.0	539.2	6,450.0

Published Results

In Table A.9, we present a list of published results on the Bentham database, for comparison with the results presented in this thesis. Because of the novelty of this database, only the results of the HTRtS contest of 2014 are available.

Table A.9: Published results on Bentham database

	WER
A2iA (unrestricted track) (Sánchez et al., 2014)	**8.6**
CITlab (restricted track) (Strauß et al., 2014)	14.6

A2iA (Sánchez et al., 2014) —The line images are passed through an MDLSTM-RNN, trained with CTC, without any preprocessing besides the mean and variance normalization of the pixel values. The RNN is trained on two databases and adapted to the Bentham data. The dropout technique is applied after each LSTM layer, and the decoding is based on an hybrid word/character language model for an open-vocabulary approach based on a static decoding graph, with a trigram word LM limited to 30k words, and a 10gram character LM. The text corpora used for LM training are Bentham books retrieved from the web.

CITlab (Strauß et al., 2014) — After contrast enhancement, and height normalization to 64px (with a different fixed height for ascenders, descenders and core region), the components belonging to surrounding lines are removed and the slant corrected. The obtained images are passed through predefined Gabor filters, followed by an MDLSTM-RNN, trained with CTC and iterative inclusion of all the training images. The LSTM cells are replaced with a version where the internal state is computed as a convex combination of previous states (MDLeaky). The decoding is carried out by a dynamic programming search, where words and special characters are identified, then constrained by a vocabulary but leaving the option to output out-of-vocabulary words. No further language modeling is applied.

Appendix B

Résumé Long

Nous vivons dans un monde digital, où l'information est stockée et traitée automatiquement. L'écriture manuscrite n'échappe pas à la règle. Les enjeux de sa reconnaissance automatique sont multiples, allant du traitement automatique des chèques et de la dématérialisation de formulaires administratifs, à l'extraction automatique d'information d'archives, de registres, ou de courriers entrants au sein de grandes administrations et entreprises.

Le domaine de la reconnaissance automatique de l'écriture manuscrite a suscité l'intérêt de chercheurs depuis plus de soixante ans, et est toujours un champ de recherche très actif, comme l'attestent les nombreux projets financés et la tenue de conférences internationales sur le sujet. D'abord concentrés sur la reconnaissance de caractères isolés, les systèmes ont peu à peu évolué vers la reconnaissance de mots isolés, puis de lignes segmentées en mots, jusqu'à la reconnaissance directe de lignes aujourd'hui.

Les systèmes *"classiques"* de reconnaissance comportent aujourd'hui plusieurs composantes. Les lignes de textes doivent tout d'abord être extraites des pages, et nous supposons dans le cadre de cette thèse que cette étape est déjà faite. Ensuite, certaines sources de variabilités dans les images sont éliminées lors d'une étape de pré-traitement des images. Les étapes suivantes dépendent surtout de la granularité des modèles. Certains systèmes, peu adaptés au larges vocabulaires, modélisent les mots entiers directement, ce qui nécessite une segmentation de la ligne de texte en mots. Les systèmes modélisant les caractères passent plus facilement à l'échelle, mais la nature cursive de l'écriture manuscrite rend la segmentation en caractères difficile (Sayre, 1973). Alors que certains tentent une segmentation explicite et heuristique de l'image en caractères ou en partie de caractères (graphèmes; Baret (1990); Knerr et al. (1998)), la plupart des systèmes actuels se reposent sur une segmentation implicite, obtenue comme produit de la reconnaissance. L'une des approches principale, choisie dans cette thèse, consiste à extraire séquences de caractéristiques de l'image en la parcourant avec une fenêtre glissante (Kaltenmeier et al., 1993). La transformation de ces séquences d'observations en séquences de caractères est généralement obtenue avec des Modèles de Markov Cachés (MMCs; Rabiner & Juang (1986)). Les probabilités d'émission des MMCs peuvent être par exemple estimées avec des mixtures de Gaussiennes, ou avec des Réseaux de

Neurones (RNs; Bourlard & Morgan (1994)). L'autre possibilité consiste à de fournir à des réseaux de neurones avec une architecture appropriée, par exemple à convolutions, l'image de la ligne directemement, ce qui est fait dans les réseaux de neurones récurrents multi-dimensionnels (Graves & Schmidhuber, 2008). Un modèle de langue est ensuite appliqué aux séquences prédites afin de les contraindre à être plausibles.

Par certains aspects, ce type de systèmes peut être mis en parallèle avec ceux utilisés dans d'autre domaines. Par exemple, ils ressemblent aux modèles de reconnaissance de la parole où des séquences de caractéristiques sont extraites et tranformées en texte, également avec des MMCs et des modèles de langues. Les entrées étant des images, certaines composantes peuvent se rapprocher des systèmes de reconnaissance d'objets et de vision articifielle.

Dans ces deux domaines, on constate ces dernières années une réduction considérable des taux d'erreur dûe à l'utilisation de réseaux de neurones profonds. En reconnaissance d'écritures, ce type de réseaux est limité, soit à des architectures à convolutions, soit à des sous-problèmes, comme la localisation de mots-clés (Thomas et al., 2013) ou la reconnaissance de caractères isolés (Ciresan et al., 2010; Cireşan et al., 2012).

De plus, quand des réseaux de neurones profonds sont appliqués, on constate, notamment en reconnaissance de parole et d'objets, que l'extraction de caractéristiques devient moins cruciale.

D'autre part, des méthodes d'entrainements de ce type de réseau ont émergés ou regagné de l'intérêt ces dernières années, comme la technique de régularisation dite de *dropout*, ou les critères discriminants au niveau des séquences complètes.

Enfin, toujours concernant les réseaux de neurones, on constate en reconnaissance d'écriture un abandon progressifs des perceptrons multi-couches, pourtant toujours détenteurs d'excellents résultats en parole, au profit des réseaux de neurones récurrents, qui font partie de tous les meilleurs systèmes et gagnent la plupart des évaluations internationales.

Dans cette thèse, nous proposons d'étudier les réseaux de neurones profonds pour la reconnaissance d'écriture manuscrite, dans le cadre de systèmes hybrides RN/MMC. Les observations précédentes donnent naturellement lieu aux questions suivantes, auxquelles nous tentons d'apporter une réponse :

- *Avec des réseaux de neurones profonds pour la reconnaissance d'écriture, est-il toujours important de choisir et d'implémenter des caractéristiques pré-définies, ou bien les simples intensités de pixels sont-elles suffisantes?*
- *Des réseaux de neurones profonds, comportant plus qu'une ou deux couches cachées, peuvent-ils apporter des améliorations significatives en reconnaissance d'écriture, et dans quelles situations?*
- *Comment les perceptrons multi-couches (profonds) se comparent-ils aux désormais très populaires réseaux de neurones récurrents, très répendus en reconnaissance d'écriture manuscrite et produisant des résultats à l'état de l'art?*

- *Quelles caractéristiques des réseaux de neurones récurrents les rendent si appropriés à la reconnaissance d'écriture manuscrite?*

- *Quelles sont les bonnes stratégies d'entrainement de réseaux de neurones pour la reconnaissance d'écriture? Est-ce que la Classification Temporelle Connexionniste (CTC, Graves et al. (2006)) peut être appliquée à d'autre types de réseaux de neurones que des réseaux récurrents? Quelles améliorations peuvent être observées en utilisant un critère d'entrainement discriminant au niveau des séquences?*

B.1 Système de Base

Nous avons mené nos expériences sur trois bases de données publiques, Rimes (Augustin et al., 2006) , IAM (Marti & Bunke, 2002), et Bentham (Sánchez et al., 2014), afin de comparer nos résultats à l'état de l'art. Elle représentent deux langues (l'anglais et le français), et deux époques (contemporaine et XIXème sciècle). Les conditions de collection de ces bases sont plus ou moins contraintes, en terme de contenu. Une description détaillée se trouve en Annexe A.

Dans cette thèse, nous nous focalisons sur le modèle optique de systèmes hybrides réseaux de neurones profonds / modèles de Markov cachés. Néanmoins, le sysème de reconnaissance complet inclut un certain nombre d'autres composantes, que nous fixons à l'avance. Le choix de ces composantes est validé sur l'ensemble de développement de chaque base, en mesurant le taux d'erreurs mots d'un modèle de Markov caché gaussien (cf. Chapitre 4).

Pré-traitement des images. Tout d'abord, les inclinaisons de la ligne de texte (Bloomberg et al., 1995), et des caractères (Buse et al., 1997) sont corrigées. Le contraste est renforcé (Roeder, 2009). Les régions horizontales correspondantes aux ascendants et descendants sont détectées (Vinciarelli & Luettin, 2001) et normalisées à une hauteur de 24 pixels.

Extraction de caractéristiques. Des séquences de vecteurs de caractéristiques sont extraites à l'aide de fenêtres glissantes (Kaltenmeier et al., 1993). Nous avons utilisé et comparé deux types de caractéristiques:

- **pré-définies :** 56 caractéristiques statistiques et géométriques (configurations de pixels, centre de gravité, intensités moyennes de pixels, histogrammes de gradients), extraites avec une fenêtre glissante de largeur 3px et de pas 3px (Bianne-Bernard, 2011; Bianne et al., 2011)

- **intensités de pixels :** vecteur contenant tous les niveaux de gris d'une fenêtre de taille 20×32px, résultant d'un redimensionnement d'une fenêtre glissante de largeur 45px et de pas 3px (25×32px et 57px pour la base Bentham)

Modèles de langue Nous avons utilisé des modèles de langue de type "*n-gram*", estimant la probabilité d'un mot, conditionnée aux $n-1$ mots précédents, et limitée à un vocabulaire pré-défini.

Pour *IAM*, nous avons défini un modèle trigramme, estimé sur les corpus LOB, Brown et Wellington, avec un lissage Kneser-Ney modifié (Kneser & Ney, 1995). Les passages du corpus LOB qui apparaissent dans les ensembles de test d'IAM ont été retirés. Le vocabulaire est limité au 50,000 mots les plus fréquents. Le taux de mots hors vocabulaire et la perplexité de ce modèle de langue sont de 4.3% et 298 (*resp.* 3.7% et 329) sur l'ensemble de validation (*resp.* test).

Pour *Rimes*, nous avons défini un modèle quadrigramme, estimé sur les annotations de l'ensemble d'entraînement, avec un lissage Kneser-Ney modifié. Le vocabulaire contient 5,000 mots, et le taux de mots hors vocabulaire et la perplexité de ce modèle de langue sont de 2.9% et 18 (*resp.* 2.6% et 18) sur l'ensemble de validation (*resp.* test).

Pour *Bentham*, nous avons défini un modèle quadrigramme, estimé sur les annotations de l'ensemble d'entraînement, avec un lissage Kneser-Ney modifié. Les débuts et fins de mots sont ajoutés aux unigrammes pour prendre en compte le grand nombre de coupure de mots présents dans cette base. Le taux de mots hors vocabulaire et la perplexité de ce modèle de langue sont de 5.6% et 103.1 sur l'ensemble de validation.

MMCs gaussiens La topologie retenue pour les MMCs comporte plusieurs états par caractères (5 pour Rimes, 6 pour IAM et Bentham), et des transitions uniquement d'un état vers lui-même et vers l'état suivant. L'espace est modélisé par un MMC à deux états. Nous avons entrainé des MMCs gaussiens, afin de valider les choix de prétraitement, de fenêtres glissantes, de modèle de langues et de topologie de MMCs. Les taux d'erreurs atteints sur les trois bases sont comparables à l'état de l'art en matière de MMC gaussiens.

B.2 Systèmes Hybrides Perceptrons Multi-Couches Profonds / MMC

Alors qu'en reconnaissance de la parole, où les systèmes standards ressemblent beaucoup à ceux que nous utilisons (extraction de caractéristiques, modélisation par MMCs et modèle de langue), les perceptrons multi-couches (PMCs) sont devenus une composante essentielle, en particulier les PMCs profonds, en reconnaissance d'écriture ils tendent à être délaissés ces dernières années au profits des réseaux de neurones récurrents.

Dans le Chapitre 5, nous nous intéressons aux perceptrons multi-couches profonds, combinés aux MMCs. Nous entrainons des PMCs pour les trois bases et les deux types de caractéristiques, avec une à sept couche cachées, en utilisant un critère de classification local, et un critère discriminant au niveau de la séquence complète.

En premier lieu, nous avons étudié l'influence du contexte en entrée du réseau. Dans les systèmes hybrides RN/MMC, il est courant de concaténer plusieurs trames successives. Nous confirmons cette hypothèse : les réseaux utilisant un concaténation de la trame centrale avec un nombre fixe de trames précédentes et suivantes donnent de bien meilleurs résultats, comparés à ceux n'utilisant que la trame centrale. La différence est particulièrement visible sur les performances du réseau seul, sans MMC et modèle de langue. Avec ces derniers, qui ajoutent des contraintes sur les séquences d'états reconnus, les améliorations sont moindres quoique visible.

Ensuite, nous avons entrainé des PMC de différentes profondeurs. Nous montrons qu'en passant d'une à plusieurs couches cachées, il est possible d'atteindre une reduction relative du taux d'erreur au niveau trame de l'ordre de 20%, pour le réseau seul, sans contrainte linguisique. En ajoutant le MMC et le modèle de langue, les améliorations sont moins visibles, surtout pour les caractéristiques pré-définies, mais tout de même non négligeables, d'environ 5% relatifs.

Quand les entrées sont les intensités de pixels, la différence entre une et plusieurs couches cachées est bien plus nette. Le simple passage à deux couches cachées permet d'obtenir une amélioration relative du taux d'erreur mots, incluant l'application du modèle de langue, allant jusqu'à 10%. Avec des réseaux plus profonds, la réduction observée monte à 17%. Ces résultats confirment l'hypothèse des modèles profonds que des représentations intermédiaires, de complexité croissante, sont calculées dans les couches basses du réseau, qui permettent une classification plus robuste.

En entrainant également des réseaux plus larges, c'est-à-dire avec plus d'unités dans les couches cachées, nous avons vérifié que ce n'est pas uniquement l'augmentation du nombre de paramètres libres, résultant de l'ajout de couches dans le PMC, qui sont à l'origine de l'amélioration des performances.

De plus, nous avons comparé les taux d'erreur mots obtenus avec les caractéristiques pré-définies et les intensités de pixels (cf. Table B.1). L'écart de performance entre les systèmes utilisant ces deux types d'entrée, d'environ 10% pour des PMCs à une couche cachée en faveur des caractéristiques, est réduit de plus que de moitié quand on considère des réseaux profonds. Ce résultat suggère que l'effort mis en oeuvre pour sélectionner et implémenter les caractéristiques adaptées au problème et leur extraction n'est pas forcément nécessaire, et qu'un système ne faisant aucun hypothèse, considérant directement le signal d'entrée, est suffisant pour obtenir des performances raisonnables.

Enfin, nous avons appliqué un critère d'entrainement des réseaux discriminant au niveau des séquences de mots. Ce type de critères vise à augmenter la probabilité *a posteriori* de la séquence de mots correcte, étant donnée la séquence d'observations, ce qui reflète mieux le but final des systèmes de reconnaissance. En pratique, pour des raisons computationnelles, cela consiste à extraire des treillis de reconnaissance, incluant le réseau de neurones et le modèle de langue, et à distinguer la séquence correcte des séquences concurrentes. L'entrainement discriminant au niveau des séquences

Table B.1: Comparaison du taux d'erreurs mots (%) obtenus avec des PMCs utilisant des caractéristiques pré-définies et des intensités de pixels

	Caractéristiques	**Pixels**	
Rimes			
PMC à une couche	14.0%	15.3%	(+9.3%)
PMC profond	13.5%	14.0%	(+3.7%)
+sMBR	12.5%	12.6%	(+0.8%)
IAM			
PMC à une couche	12.4%	13.6%	(+9.7%)
PMC profond	11.8%	12.3%	(+4.2%)
+sMBR	10.9%	11.7%	(+7.3%)
Bentham			
PMC à une couche	21.5%	28.8%	(+34.0%)
PMC profond	20.1%	22.4%	(+11.4%)
+sMBR	18.6%	19.4%	(+4.3%)

est classique en reconnaissance de la parole, où il permet d'obtenir des améliorations relatives des taux d'erreurs entre 5 et 10%.

Nous avons choisi le critère sMBR (Kingsbury, 2009), qui donnait de meilleurs résultats que d'autres sur une tâche de reconnaissance de la parole. Nous montrons qu'affiner un PMC entrainé au niveau trame avec sMBR permet d'atteindre des améliorations significatives, entre 5 et 13% en taux d'erreur mots (cf. Table B.1) et jusqu'à 20% en taux d'erreur caractères. Nous montrons également l'importance du modèle de langue dans le calcul de ce critère, bien qu'un simple unigramme semble suffisant pour obtenir ces améliorations.

B.3 Systèmes Hybrides Réseaux de Neurones Récurrents Profonds / MMC

Le Chapitre 6 est consacré aux réseaux de neurones récurrents (RNRs) profonds, combinés aux MMCs. Les neurones de ces réseaux sont des unités Longue Mémoire à Court Terme (LMCT, en anglais *Long Short-Term Memory*, Hochreiter & Schmidhuber (1997)). Les couches récurrentes contiennent en fait deux couches, pour parcourir les séquences dans les deux directions. Nous entrainons ces réseaux avec le critère de Classification Temporelle Connexionniste (CTC), défini par Graves et al. (2006). Ce type de réseau est populaire en reconnaissance d'écriture, et ses bonnes performances en ont fait un standard du domaine (Graves & Schmidhuber, 2008; Bluche et al., 2014a; Moysset et al., 2014; Kozielski et al., 2014a; Doetsch et al., 2014).

Afin de faire rentrer ces RNRs dans le cadre des systèmes hybrides RN/MMC, nous définissons des MMCs de caractères à un état, ainsi qu'un MMC pour le symbole *blank* de la CTC, optionnel entre deux caractères. Nous entrainons des RNRs pour les trois bases et les deux types de caractéristiques, avec une à treize couche cachées, alternant

des couches récurrentes et des couches standards.

Tout d'abord, nous avons reproduit l'expérience du chapitre précédent, consistant à concaténer les trames en entrée du réseau pour lui fournir un contexte plus grand. Nous montrons que cette stratégie, bénéfique pour les PMCs, tend à déteriorer les résultats des RNRs. Nous vérifions que la récurrence de ces réseaux leur permettent d'apprendre effectivement les dépendences et le contexte nécessaire pour prédire les caractères. Nous observons que ce contexte dépasse souvent la zone d'un caractère, s'étendant parfois au mot entier, suggérant qu'un large contexte est parfois nécessaire pour prédire de façon fiable un caractère, et supportant le choix des unités LMCT.

Ensuite, nous montrons que la récurrence dans ces réseaux, avec l'entrainement CTC, est particulièrement importante dans les couches supérieures. Les couches basses, en particulier quand les entrées sont les intensités de pixels, servent plus à extraire des caractéristiques élémentaires du signal, rendant la récurrence moins cruciale. Cette observation pourrait permettre de simplifier les RNRs, en n'implémentant la récurrence, plus chère en temps de calcul, que dans les couches hautes du réseau.

Nous reportons également des améliorations significatives grâce à l'augmentation de la profondeur des réseaux. Nous nous sommes limités à treize couche cachées. Nous avons observé des baisses de taux d'erreur au niveau caractère, quand le réseau est utilisé seul, sans modèle de langue, entre 35 et 50% pour les caractéristiques prédéfinies, et entre 70 et 80% pour les pixels, quand nous passons d'un réseau à une couche cachée à un réseau profond.

Les améliorations des taux d'erreurs ne sont pas aussi impressionnantes quand les réseaux sont inclus dans la chaîne de reconnaissance complète, incluant les contraintes liguistiques. Cela est certainement dû au fait que les erreurs corrigées par des réseaux plus profonds étaient également corrigées par le modèle de langue. Ceci dit, nous observons quand même des baisses de taux d'erreurs, entre 10 et 15% pour les caractéristiques, et aux alentours de 40% pour les pixels, ce qui n'est pas négligeable.

Comme nous l'avons fait pour les PMCs, nous avons vérifié que ces résultats n'étaient pas simplement dûs à l'augmentation du nombre de paramètres libres du modèle. Nous avons comparé des RNRs plus profonds à des RNRs plus "larges", c'est-à-dire avec plus d'unités dans les couches cachées, et obtenu de meilleurs résultats en augmentant la profondeur.

De plus, nous confirmons l'efficacité du *dropout*, une technique de régularisation des réseaux consistant à ignorer les réponses de certains neurones lors de l'entrainement. Cette méthode, introduite par Hinton et al. (2012), a récemment été appliquée aux RNRs (Pham et al., 2014; Zaremba et al., 2014). En particulier, avec une architecture de réseaux proche de la nôtre, mais pour des RNR multi-dimensionnels, Pham et al. (2014) ont fait le choix d'appliquer le *dropout* uniquement en sortie des couches LMCT.

En explorant de manière exhaustive les différentes positions pour le dropout, nous avons d'une part confirmé que cette technique permettait pratiquement toujours d'améliorer les performances des RNRs seuls. La position du dropout relative aux couches LMCT

semble particulièrement importante, et nous reportons des améliorations significatives par rapport à la méthode proposée par Pham et al. (2014) quand le dropout est appliqué en entrée des LMCTs plutôt qu'en sortie.

Cependant, dans le système complet, l'ensemble des sorties est passé au décodeur, et il apparaît que, dans ce cas, le dropout sur les poids de classification soit bénéfique. Finalement, les meilleurs résultats ont été obtenu en utilisant le dropout avant les couches récurrentes basses, et après les couches récurrentes hautes.

Table B.2: Comparaison du taux d'erreurs mots (%) obtenus avec des RNRs utilisant des caractéristiques pré-définies et des intensités de pixels

	Caractéristiques	**Pixels**	
Rimes			
LMCTB-RNR à une couche	14.9%	24.1%	(+65.1%)
LMCTB-RNR profond	12.9%	14.0%	(+7.9%)
+dropout	12.7%	12.7%	(+0.0%)
IAM			
LMCTB-RNR à une couche	13.4%	83.2%	(+520.0%)
LMCTB-RNR profond	11.4%	12.8%	(+12.3%)
+dropout	11.2%	11.4%	(+0.9%)
Bentham			
LMCTB-RNR à une couche	20.6%	33.8%	(+64.1%)
LMCTB-RNR profond	18.0%	20.3%	(+12.8%)
+dropout	16.0%	17.0%	(+6.3%)

Enfin, nous montrons que l'écart de performance entre les systèmes traitant les caractéristiques pré-définies et les simples valeurs de pixels est largement réduit quand les réseaux sont profonds, passant d'une soixantaine à une dixaine de pourcents, et devenant pratiquement négligeables quand le dropout est utilisé en entrainement (cf. Table B.2). Comme pour les PMCs, la nécessité de trouver et d'implémenter une extraction de caractéristiques pertinantes disparaît, au profit d'un apprentissage automatique, sans a priori spécifique, des caractéristiques par le réseau.

B.4 Une Comparaison Expérimentale de l'Entrainement CTC et au Niveau Trame

Au Chapitre 7, nous nous intéressons plus particulièrement aux sorties des réseaux et à la méthode d'entrainement. En effet, dans les expériences précédentes, les PMCs prédisent des états de MMC, six par caractères, et sont entrainés au niveau trame, tandis que les RNRs sont entrainés avec la CTC à prédire directement des caractères, ainsi qu'un symbole spécial, *blank*, correspondant à une absence de prédiction de caractère. Les deux types de système obtiennent de bons résultats.

L'entrainement CTC définit des transitions possibles entre les différents labels, et considère toutes les segmentations possibles du signal d'entrée en prédictions, ce qui rappelle l'entrainement Baum-Welsch des MMCs. Dans un premier temps, nous montrons que l'entrainement CTC est très similaire à l'entainement global des systèmes hybrides RN/MMC par une procédure *forward-backward*, déjà proposée par plusieurs groupes de recherche dans les années 90 (Senior & Robinson, 1996; Hennebert et al., 1997; Yan et al., 1997). Les principales différences sont l'absence de probabilités de transition et de probabilités à priori d'états dans la CTC, ainsi qu'une topologie spécifique incluant un symbole *blank*.

Nous avons étudié l'influence de la topologie, en faisant varier le nombre d'états des MMCs de caractères, et la présence ou non d'un MMC de blank à un état. Les mixtures de gaussiennes et les PMCs obtiennent de meilleurs résultats avec plusieurs états par caractère, et pas de blank, alors que les RNRs atteignent leur meilleure performance avec la topologie définie par la CTC.

D'autre part, l'entrainement CTC n'étant pas théoriquement limité aux RNRs, nous l'avons appliqué à un PMC mais les résultats obtenus sont relativement décevants. Par contre, en gardant plusieurs états par caractères, sans *blank*, l'entrainement CTC modifié permet une légère amélioration des résultats du PMC, quoique limitée, ce qui est cohérent avec les conclusions de Hennebert et al. (1997) : si l'ensemble d'entrainement est suffisamment grand, il y a peu ou pas de différence entre l'entrainement au niveau trame, avec des étiquettes de trames fixes, et l'entrainement consistant à considérer toutes les segmentations possibles en états de MMC.

Nous avons ensuite poussé plus loin la comparaison entre la CTC et l'entrainement au niveau des frames. Nous avons entrainé des PMCs et des RNRs avec les deux critères, en faisant également varier la topologie. Nous avons trouvé que le cadre strict de la CTC, avec un état par caractère et le *blank*, était particulièrement adapté aux RNRs. Dans toutes les situations sauf celle-ci, le taux d'erreur diminuait avec le nombre d'état (cf. Figure B.1). Pour les PMCs, nous avons obtenu les meilleurs résultats finaux avec six états par MMC, et sans blank, ce qui correspond à la topologie choisie pour les chapitres précédents.

De plus, quand les réseaux sont considérés seuls, sans modèle de langue, l'erreur de classification augmente avec le nombre d'états, à l'inverse du taux d'erreur dans la chaine complète. Cela confirme néanmoins que le problème de classification est plus simple avec un nombre limité d'états. En revanche, pour l'entrainement CTC sans blank, l'ajout d'états dans les modèles fait baisser le taux d'erreur, ce qui suggère que le choix d'une longueur de MMC adaptée permet d'aider l'algorithme à produire de bons alignements pendant la procédure d'entrainement.

Nous avons étudié l'interaction de l'entrainement CTC avec le *blank*. Nous avons observé qu'en utilisant directement la CTC sans blank, l'entrainement ne convergeait pas, ou vers des solutions sous-optimales. Dans ces cas-là, nous avons dû faire une première passe d'entrainement au niveau trame, avant de poursuivre avec la CTC. Cela suggère

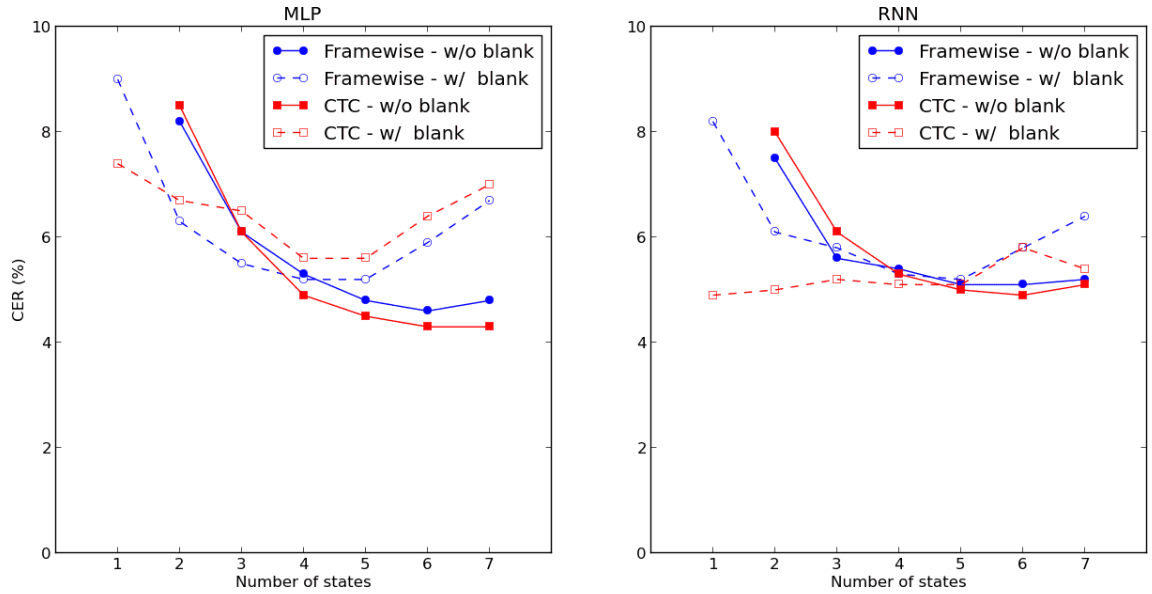


Figure B.1: Comparaison des taux d'erreur caractères (CER%) avec l'entrainement CTC et au niveau trame, avec et sans le symbole spécial *blank*. (à gauche: PMC; à droite : RNR).

que, quand les caractères sont représentés par seulement quelques états, le blank peut aider dans la procédure d'alignement. En fait, nous expliquons que la présence du blank, associé à la CTC provoque dans les premiers instants de l'entrainement une tendance du réseau à ne prédire que ce symbole. Les prédictions de caractères apparaissent ensuite sous forme de pics localisés, qui peuvent être intéressant pour avoir un décodage riche et rapide quand les contraintes linguistiques sont introduites.

Enfin, concernant cette partie, nous concluons que la CTC, incluant la procédure *forward-backward*, mais plus spécifiquement le *blank* et la représentation des caractères par une seule sortie de réseau, ou un seul état de MMC, est particulièrement adaptée aux RNRs, et leur permet d'obtenir relativement facilement de très bonnes performances. Quant aux PMCs, nous avons confirmé que la topologie des MMCs choisies pour cette thèse était bonne. La procédure *forward-backward*, que l'on trouve également dans l'entrainement discriminant au niveau des séquences, ne permet qu'une légère amélioration.

B.5 Combinaisons et Résultats Finaux

Dans le dernier chapitre, nous étudions les systèmes de reconnaissance complets, incluant le modèle hybride RN/MMC, et le décodage avec les contraintes linguistiques (vocabulaire et modèle de langue). Nous comparons directement les différents modèles optiques (RNRs et PMCs), et les différents types d'entrées (pixels et caractéristiques). Nous présentons également les résultats de nos systèmes sur les ensembles de test de

chaque base de données, afin de les comparer aux meilleurs résultats publiés. Nous donnons enfin dans ce chapitre une description détaillée des systèmes soumis à l'évaluation HTRtS 2014, ainsi que leurs résultats.

Nous confirmons nos observations des chapitres précédents, à savoir que les pixels permettent d'obtenir des résultats compétitifs avec ceux atteints avec les caractéristiques pré-définies, pour des modèles optiques comportant des réseaux de neurones profonds. D'autre part, nous montrons que des PMCs profonds atteignent des performances similaires à celles des RNRs, malgré la domination de ces derniers dans la littérature récente en reconnaissance d'écriture.

Cependant, comme vu précédemment, les PMCs requièrent un ajustement du nombre d'états dans les MMCs, et de la quantité de contexte à donner en entrée des réseaux. De plus, ce niveau de performance a été atteint grâce à l'entrainement discriminant au niveau des séquences. A l'inverse, les RNRs, avec l'entrainement CTC, modélisent les séquences de caractères directement, et sont plus simples à optimiser pour atteindre facilement les résultats présentés. Ils gèrent la séquence d'entrée, et notamment la taille de contexte nécessaire aux prédictions directement, et il n'y a pas besoin d'alignements forcés des trames avec les états de MMCs pour l'entrainement. Ceci dit, pour une comparaison plus juste de ces modèles, nous prévoyons d'appliquer la technique du *dropout* dans les PMCs, ainsi que la procédure d'entrainement en séquence aux RNRs.

Nous soulignons également l'importance des paramètres de décodage. Le facteur optique, qui contrôle l'influence du score du modèle optique par rapport à celui du modèle de langue, a le plus grand impact sur les résultats finaux. A travers les différentes expériences et configurations, nous avons trouvé que 0.1 semble être la meilleure valeur pour les systèmes basés sur des PMCs, tandis que 1.0 est le meilleur choix pour les RNRs. L'ajustement de la pénalité d'insertion de mot et du facteur appliqué aux probabilités a priori d'états de MMC peuvent également apporter quelques pourcents d'amélioration.

Le contraintes linguisiques, c'est-à-dire le vocabulaire et le modèle de langue, sont cruciaux pour atteindre les résultats présentés dans cette thèse. Sans eux, les taux d'erreur sont largement plus haut, en particulier pour les PMCs, pour lesquels la moitié des mots seraient alors mal reconnus. En n'appliquant qu'un vocabulaire, sans donner un score aux séquences de mots, plus d'un mot sur quatre serait encore faux.

Les deux types de réseau de neurones et d'entrées nous donnent quatre systèmes de reconnaissance, qui obtiennent tous des résultats comparables, mais se comportent différemment, et font des erreurs différentes. En combinant ces systèmes avec deux méthodes, au niveau des transcriptions (ROVER; Fiscus (1997)), et au niveau des treillis de reconnaissance (Xu et al., 2011), nous montrons la complémentarité des différents modèles. En effet, nous avons obtenu plus qu'un pourcent d'amélioration absolue du taux d'erreur mots, comparé à celui du meilleur système. La méthode basée sur les treillis de reconnaissance, qui considère un plus grand nombre d'alternatives, produit de meilleurs et plus robustes résultats.

Table B.3: Résultats sur la base Rimes (taux d'erreur mots - WER% - et caractères - CER%).

		Dev.		Eval.	
		WER%	**CER%**	**WER%**	**CER%**
GMMC	Caractéristiques	17.2	5.9	15.8	6.0
PMC	Caractéristiques	12.5	3.4	12.7	3.7
	Pixels	12.6	3.8	12.4	3.9
RNR	Caractéristiques	12.8	3.8	12.6	3.9
	Pixels	12.7	4.0	13.8	4.6
	Combinaison ROVER	11.3	3.5	11.3	3.7
	Combinaison de treillis	11.2	3.3	**11.2**	3.5
	Pham et al. (2014)	-	-	12.3	**3.3**
	Doetsch et al. (2014)	-	-	12.9	4.3
	Messina & Kermorvant (2014)	-	-	13.3	-
	Kozielski et al. (2013a)	-	-	13.7	4.6
	Messina & Kermorvant (2014)	-	-	14.6	-
	Menasri et al. (2012)	-	-	15.2	7.2

Table B.4: Résultats sur la base IAM (taux d'erreur mots - WER% - et caractères - CER%).

		Dev.		Eval.	
		WER%	**CER%**	**WER%**	**CER%**
GMMC	Caractéristiques	15.2	6.3	19.6	9.0
PMC	Caractéristiques	10.9	3.7	13.3	5.4
	Pixels	11.4	3.9	13.8	5.6
RNR	Caractéristiques	11.2	3.8	13.2	5.0
	Pixels	11.8	4.0	14.4	5.7
	Combinaison ROVER	9.6	3.6	11.2	4.7
	Combinaison de treillis	9.6	3.3	**10.9**	**4.4**
	Doetsch et al. (2014)	**8.4**	**2.5**	12.2	4.7
	Kozielski et al. (2013a)	9.5	2.7	13.3	5.1
	Pham et al. (2014)	11.2	3.7	13.6	5.1
	Kozielski et al. (2013a)	11.9	3.2	-	-
	Messina & Kermorvant (2014)	-	-	19.1	-
	Espana-Boquera et al. (2011)	19.0	-	22.4	9.8

Enfin, nous avons appliqué nos modèles sur les ensembles de test de chaque base, afin de comparer leur résultats à ceux publiés par d'autres sur les mêmes données. Nous les rappelons ici, Table B.3 pour Rimes, Table B.4 pour IAM, et Table B.5 pour Bentham. Nous voyons que sur Rimes et IAM, tous nos systèmes simples atteignent un niveau de performance comparable à l'état de l'art. De plus, la combinaison des reconnaisseurs égale ou dépasse tous les meilleurs résultats publiés sur les trois bases : Rimes, IAM et Bentham.

Table B.5: Résultats sur la base Bentham (taux d'erreur mots - WER% - et caractères - CER%).

(a) Tâche restreinte

Model		WER%	CER%
PMC profond	*Caractéristiques*	18.6	7.5
	Pixels	20.9	8.2
RNR profond	*Caractéristiques*	16.2	5.4
	Pixels	16.9	5.9
Combinaison de treillis		**14.1**	**5.0**
CITlab		14.6	-
Notre système (Compétition)		15.1	-

(b) Tâche non restreinte

Model		WER%	CER%
PMC profond	*Caractéristiques*	13.2	4.9
	Pixels	14.4	6.1
RNR profond	*Caractéristiques*	11.2	4.0
	Pixels	11.5	4.4
	uRNR1	10.9	4.0
	uRNR2	10.5	3.7
	uRNR3	10.2	3.6
Combinaison de treillis		**8.6**	**3.1**
A2iA (système de production)		**8.6**	-
Notre système (Compétition)		11.1	-

B.6 Conclusions et Perspectives

En conclusion, nous présentons dans cette thèse une étude poussée des réseaux de neurones profonds pour la reconnaissance d'écriture manuscrite et de leur intégration dans des systèmes hybrides RN/MMC.

Nous nous sommes intéressés aux entrées des réseaux, et avons montré qu'avec la profondeur des réseaux, l'écart de performance entre des caractéristiques extraites prédéfinies et les simples valeurs de pixels se réduisait, diminuant ainsi l'intérêt de la création et de l'implémentation d'une bonne extraction de caractéristiques. D'autre part, nous avons étudié la taille du contexte à fournir en entrée du réseau, et montré que ce paramètre était crucial pour les PMCs, alors que les résultats n'étaient pas améliorés par l'introduction d'un plus grand contexte pour les RNRs, qui peuvent apprendre à utiliser un contexte arbitrairement grand à travers leurs connexions récurrentes. Pour les valeurs de pixels, nous n'avons pas expérimenté beaucoup de tailles différentes des fénêtres glissantes, et des améliorations peuvent être attendues si une attention plus fine est portée à ce paramètre.

Les nombreuses expériences réalisées nous permettent d'affirmer que des gains significatifs de performance résultent d'une plus grande profondeur de réseau, c'est-à-dire d'un plus grand nombre de couches cachées. Cela constitue la majeure contribution de notre travail, sachant qu'aujourd'hui, en reconnaissance d'écriture manuscrite, très peu de travaux ont considéré des architectures à plusieurs couches cachées, mises à part les architectures à convolutions.

Il est important de signaler que les réseaux à convolutions sont une excellente alternative aux PMCs et obtiennent de très bons résultats en vision artificielle, de même que les RNRs multi-dimensionnels sont une excellente alternative aux RNRs présentés, et constituent l'état de l'art en reconnaissance d'écriture. Dans un souci de cohérence, nous ne discutons pas dans cette thèse de nos travaux sur les réseaux à convolutions (Bluche et al., 2013a,b) ou utilisant des RNRs multi-dimensionnels (Bluche et al., 2014a; Moysset et al., 2014; Pham et al., 2014), mais il va de soi qu'une étude comparative les incluant doit être menée.

Enfin, nous avons étudié l'influence des sorties des réseaux (notamment le nombre d'états de MMCs par caractère et la présence d'un symbole *blank*), et la méthode d'entrainement des réseaux, qui a en particulier souligné la bonne interaction entre la CTC, le *blank*, et les RNRs. Des gains intéressants ont été atteints avec un entrainement en séquence, et l'utilisation du dropout, et une poursuite logique de ces travaux consisterait à appliquer aux RNRs l'entrainement en séquence, et aux PMCs la technique du *dropout*.

Parmi les directions futures de recherche, outre l'application de tels systèmes à d'autres types de données, telles que des scripts différents comme l'arabe, où des données plus difficiles comme celles de l'évaluation MAURDOR (Brunessaux et al., 2014), il semble aujourd'hui important de trouver des pistes pour améliorer les modèles de langues, et pour passer de la reconnaissance de lignes, nécessitant une segmentation explicite, à une reconnaissance de bout en bout, qui transcrirait les images des documents entiers, et serait dans la veine de l'évolutions des systèmes faisant de moins en moins d'hypothèses.

Bibliography

Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., & Penn, G. (2012). Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, (pp. 4277–4280). IEEE.

Al-Hajj Mohamad, R., Likforman-Sulem, L., & Mokbel, C. (2009). Combining slanted-frame classifiers for improved HMM-based Arabic handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *31*(7), 1165–1177.

Anastasakos, T., McDonough, J., Schwartz, R., & Makhoul, J. (1996). A compact model for speaker-adaptive training. In *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, vol. 2, (pp. 1137–1140). IEEE.

Augustin, E. (2001). *Reconnaissance de mots manuscrits par systèmes hybrides Réseaux de Neurones et Modèles de Markov Cachés*. Ph.D. thesis, Université René Descartes, Paris V.

Augustin, E., Carré, M., Grosicki, E., Brodin, J.-M., Geoffrois, E., & Preteux, F. (2006). RIMES evaluation campaign for handwritten mail processing. In *Proceedings of the Workshop on Frontiers in Handwriting Recognition*, 1.

Bahl, L. B., de Souza, P., & P Mercer, R. (1986). Maximum mutual information estimation of hidden Markov model parameters for speech recognition. Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'86.

Baldi, P., & Chauvin, Y. (1994). Smooth on-line learning algorithms for hidden Markov models. *Neural Computation*, *6*(2), 307–318.

Baret, O. (1990). *Régularités singulairtés de représentations et leur complémentarité: application à la reconnaissance de l'écriture manuscrite non contrainte*. Ph.D. thesis.

Baum, L. E., Eagon, J., et al. (1967). An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bull. Amer. Math. Soc*, *73*(3), 360–363.

Baum, L. E., Petrie, T., Soules, G., & Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, (pp. 164–171).

Bazzi, I. (2002). *Modelling out-of-vocabulary words for robust speech recognition*. Ph.D. thesis, Massachusetts Institute of Technology.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, *2*(1), 1–127.

Bengio, Y., De Mori, R., Flammia, G., & Kompe, R. (1992). Global optimization of a neural network-hidden Markov model hybrid. *Neural Networks, IEEE Transactions on*, *3*(2), 252–259.

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*(6), 1137–1155.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, *19*, 153.

Bengio, Y., LeCun, Y., & Henderson, D. (1994a). Globally trained handwritten word recognizer using spatial representation convolutional neural networks and hidden Markov models. *Advances in Neural Information Processing Systems*, (pp. 937–937).

Bengio, Y., LeCun, Y., Nohl, C., & Burges, C. (1995). Lerec: A NN/HMM hybrid for on-line handwriting recognition. *Neural Computation*, *7*(6), 1289–1303.

Bengio, Y., Simard, P., & Frasconi, P. (1994b). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, *5*(2), 157–166.

Bertolami, R., & Bunke, H. (2008). Hidden Markov Model Based Ensemble Methods for Offline Handwritten Text Line Recognition. *Pattern Recognition*, *41*(11), 3452 – 3460.

Bianne, A.-L., Menasri, F., Al-Hajj, R., Mokbel, C., Kermorvant, C., & Likforman-Sulem, L. (2011). Dynamic and Contextual Information in HMM modeling for Handwriting Recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, *33*(10), 2066 – 2080.

Bianne-Bernard, A.-L. (2011). *Reconnaissance de mots manuscrits cursifs par modèles de Markov cachés en contexte*. Ph.D. thesis, Telecom ParisTech.

Bianne-Bernard, A.-L., Menasri, F., Likforman-Sulem, L., Mokbel, C., & Kermorvant, C. (2012). Variable length and context-dependent HMM letter form models for Arabic handwritten word recognition. In *IS&T/SPIE Electronic Imaging*, (pp. 829708–829708). International Society for Optics and Photonics.

Bledsoe, W., & Browning, I. (1959). Pattern recognition and reading by machine. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, (pp. 225–232). ACM.

Bloomberg, D. S., Kopec, G. E., & Lakshmi Dasari (1995). Measuring document image skew and orientation. *Proc. SPIE Document Recognition II*, *2422*(302), 302–316.

Bluche, T., Kermorvant, C., & Louradour, J. (2015a). Where to Apply Dropout in Recurrent Neural Networks for Handwriting Recognition? In *13th International Conference on Document Analysis and Recognition (ICDAR)*, (pp. –). IEEE.

Bluche, T., Louradour, J., Knibbe, M., Moysset, B., Benzeghiba, M. F., & Kermorvant, C. (2014a). The A2iA Arabic Handwritten Text Recognition System at the Open HaRT2013 Evaluation. In *11th IAPR International Workshop on Document Analysis Systems (DAS)*, (pp. 161–165). IEEE.

Bluche, T., Moysset, B., & Kermorvant, C. (2014b). Automatic Line Segmentation and Ground-Truth Alignment of Handwritten Documents. In *14th International Conference on Frontiers in Handwriting Recognition (ICFHR2014)*, (pp. 667–672).

Bluche, T., Ney, H., & Kermorvant, C. (2013a). Feature Extraction with Convolutional Neural Networks for Handwritten Word Recognition. In *12th International Conference on Document Analysis and Recognition (ICDAR)*, (pp. 285–289). IEEE.

Bluche, T., Ney, H., & Kermorvant, C. (2013b). Tandem HMM with convolutional neural network for handwritten word recognition. In *17th International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (pp. 2390–2394). IEEE.

Bluche, T., Ney, H., & Kermorvant, C. (2014c). A Comparison of Sequence-Trained Deep Neural Networks and Recurrent Neural Networks Optical Modeling for Handwriting Recognition. In *International Conference on Statistical Language and Speech Processing*, (pp. 199–210).

Bluche, T., Ney, H., & Kermorvant, C. (2015b). The LIMSI Handwriting Recognition System for the HTRtS 2014 Contest. In *13th International Conference on Document Analysis and Recognition (ICDAR)*, (pp. –). IEEE.

Bluche, T., Ney, H., Louradour, J., & Kermorvant, C. (2015c). Framewise and CTC Training of Neural Networks for Handwriting Recognition. In *13th International Conference on Document Analysis and Recognition (ICDAR)*, (pp. –). IEEE.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, (pp. 177–186). Springer.

Bottou, L., Bengio, Y., & Le Cun, Y. (1997). Global training of document processing systems using graph transformer networks. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, (pp. 489–494). IEEE.

Bourlard, H., & Morgan, N. (1994). *Connectionist speech recognition: a hybrid approach Chapter 7*, vol. 247 of *The Kluwer international series in engineering and computer science: VLSI, computer architecture, and digital signal processing*. Kluwer Academic Publishers.

Bourlard, H., & Wellekens, C. J. (1989). Links between Markov models and multilayer perceptrons. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *12*(12), 1167–1178.

Bozinovic, R. M., & Srihari, S. N. (1989). Off-line cursive script word recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *11*(1), 68–83.

Bridle, J. S. (1990a). Alpha-nets: a recurrent âneuralâ network architecture with a hidden Markov model interpretation. *Speech Communication*, *9*(1), 83–92.

Bridle, J. S. (1990b). Probabilistic interpretation of feedforward classification network outputs with relationships to statistical pattern recognition. In *Neurocomputing*, (pp. 227–236). Springer.

Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., & Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, *18*(4), 467–479.

Brunessaux, S., Giroux, P., Grilhères, B., Manta, M., Bodin, M., Choukri, K., Galibert, O., & Kahn, J. (2014). The Maurdor Project: Improving Automatic Processing of Digital Documents. In *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, (pp. 349–354). IEEE.

Bunke, H., Bengio, S., & Vinciarelli, A. (2004). Offline recognition of unconstrained handwritten texts using HMMs and statistical language models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *26*(6), 709–720.

Buse, R., Liu, Z. Q., & Caelli, T. (1997). A structural and relational approach to handwritten word recognition. *IEEE Transactions on Systems, Man and Cybernetics*, *27*(5), 847–61.

Cao, H., Natarajan, P., Peng, X., Belanger, K. S. D., & Li, N. (????). Progress in the Raytheon BBN Arabic Offline Handwriting Recognition System.

Casey, R. G., & Lecolinet, E. (1996). A survey of methods and strategies in character segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *18*(7), 690–706.

Causer, T., & Wallace, V. (2012). Building a volunteer community: results and findings from Transcribe Bentham. *Digital Humanities Quarterly*, *6*.

Chatelain, C., Heutte, L., & Paquet, T. (2006). Segmentation-driven recognition applied to numerical field extraction from handwritten incoming mail documents. In *Document Analysis Systems VII*, (pp. 564–575). Springer Berlin Heidelberg.

Chen, J., Zhang, B., Cao, H., Prasad, R., & Natarajan, P. (2012). Applying Discriminatively Optimized Feature Transform for HMM-based Off-Line Handwriting Recognition. In *ICFHR*, (pp. 219–224).

Chen, M.-Y., Kundu, A., & Srihari, S. N. (1995). Variable duration hidden Markov model and morphological segmentation for handwritten word recognition. *Image Processing, IEEE Transactions on*, *4*(12), 1675–1688.

Chen, S. F., & Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, (pp. 310–318). Association for Computational Linguistics.

Chevalier, S., Prêteux, F. J., Geoffrois, E., & Lemaitre, M. (2005). A generic 2D approach of handwriting recognition. In *ICDAR*, (pp. 489–493).

Chou, W., Lee, C.-H., & Juang, B.-H. (1993). Minimum error rate training based on N-best string models. In *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, vol. 2, (pp. 652–655). IEEE.

Ciresan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2010). Deep big simple neural nets for handwritten digit recognition. *Neural computation*, *22*(12), 3207–3220.

Cireşan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2012). Deep big multilayer perceptrons for digit recognition. In *Neural Networks: Tricks of the Trade*, (pp. 581–598). Springer.

Côté, M., Lecolinet, E., Cheriet, M., & Suen, C. Y. (1998). Automatic reading of cursive scripts using a reading model and perceptual concepts. *International Journal on Document Analysis and Recognition*, *1*(1), 3–17.

Cottrell, G. W., & Munro, P. (1988). Principal components analysis of images via back propagation. In *Visual Communications and Image Processing'88: Third in a Series*, (pp. 1070–1077). International Society for Optics and Photonics.

Dahl, G. E., Sainath, T. N., & Hinton, G. E. (2013). Improving deep neural networks for LVCSR using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, (pp. 8609–8613). IEEE.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, (pp. 1–38).

Deng, L., Li, J., Huang, J.-T., Yao, K., Yu, D., Seide, F., Seltzer, M., Zweig, G., He, X., Williams, J., et al. (2013). Recent advances in deep learning for speech research at Microsoft. In *38th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP2013)*, (pp. 8604–8608). IEEE.

Doetsch, P., Hamdani, M., Ney, H., Gimenez, A., Andres-Ferrer, J., & Juan, A. (2012). Comparison of Bernoulli and Gaussian HMMs using a vertical repositioning technique for off-line handwriting recognition. In *Proceedings of the 2012 International Conference on Frontiers in Handwriting Recognition*, (pp. 3–7). IEEE Computer Society.

Doetsch, P., Kozielski, M., & Ney, H. (2014). Fast and robust training of recurrent neural networks for offline handwriting recognition. (pp. –).

Dreuw, P. (2011). The RWTH OCR System For Handwritten and Machine Printed Text Recognition.
URL http://www-i6.informatik.rwth-aachen.de/rwth-ocr/

Dreuw, P., Doetsch, P., Plahl, C., & Ney, H. (2011a). Hierarchical hybrid MLP/HMM or rather MLP features for a discriminatively trained gaussian HMM: a comparison for offline handwriting recognition. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, (pp. 3541–3544). IEEE.

Dreuw, P., Heigold, G., & Ney, H. (2011b). Confidence-and margin-based MMI/MPE discriminative training for off-line handwriting recognition. *International Journal on Document Analysis and Recognition (IJDAR)*, *14*(3), 273–288.

Dreuw, P., Jonas, S., & Ney, H. (2008). White-space models for offline Arabic handwriting recognition. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, (pp. 1–4). IEEE.

Dreuw, P., Rybach, D., Gollan, C., & Ney, H. (2009). Writer adaptive training and writing variant model refinement for offline Arabic handwriting recognition. In *Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on*, (pp. 21–25). IEEE.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, *12*, 2121–2159.

Eck, D., & Schmidhuber, J. (2002). A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*.

Edelman, S., Flash, T., & Ullman, S. (1990). Reading cursive handwriting by alignment of letter prototypes. *International Journal of Computer Vision*, *5*(3), 303–331.

El-Hajj, R., Likforman-Sulem, L., & Mokbel, C. (2005). Arabic handwriting recognition using baseline dependant features and hidden markov modeling. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, (pp. 893–897). IEEE.

El-Yacoubi, A., Gilloux, M., Sabourin, R., & Suen, C. Y. (1999). An HMM-based approach for off-line unconstrained handwritten word modeling and recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *21*(8), 752–760.

El-Yacoubi, M. A., Gilloux, M., & Bertille, J.-M. (2002). A statistical approach for phrase location and recognition within a text line: an application to street name recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *24*(2), 172–188.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179–211.

Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., & Vincent, P. (2009). The difficulty of training deep architectures and the effect of unsupervised pre-training. In *International Conference on Artificial Intelligence and Statistics*, (pp. 153–160).

Espana-Boquera, S., Castro-Bleda, M. J., Gorbe-Moya, J., & Zamora-Martinez, F. (2011). Improving offline handwritten text recognition with hybrid HMM/ANN models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *33*(4), 767–779.

Farabet, C., Couprie, C., Najman, L., & LeCun, Y. (2013). Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *35*(8), 1915–1929.

Favata, J. T., & Srikantan, G. (1996). A multiple feature/resolution approach to handprinted digit and character recognition. *International journal of imaging systems and technology*, *7*(4), 304–311.

Fink, G. A., & Plotz, T. (2007). On the use of context-dependent modeling units for HMM-based offline handwriting recognition. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, vol. 2, (pp. 729–733). IEEE.

Fischer, A., Keller, A., Frinken, V., & Bunke, H. (2012). Lexicon-free handwritten word spotting using character HMMs. *Pattern Recognition Letters*, *33*(7), 934–942.

Fiscus, J. (1998). Sclite scoring package. *US National Institute of Standard Technology (NIST), URL http://www. itl. nist. gov/iaui/894.01/tools*.

Fiscus, J. G. (1997). A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (ROVER). In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU1997)*, (pp. 347–354). IEEE.

Gatos, B., Louloudis, G., Causer, T., Grint, K., Romero, V., Sanchez, J. A., Toselli, A. H., & Vidal, E. (2014). Ground-truth production in the tranScriptorium project. In *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, (pp. 237–241). IEEE.

Gatos, B., Louloudis, G., Stamatopoulos, N., Ntirogiannis, K., Papandreou, A., Pratikakis, I., Zagoris, K., Sánchez, J. A., Romero, V., Toselli, A., Vidal, E., Villegas, M., Álvaro, F., & Bosch, V. (2013). Description and evaluation of tools for DIA HTR and KWS (M12).
URL http://transcriptorium.eu/pdfs/deliverables/tranScriptorium-D3.1.2-31December2013.pdf

Gatos, B., Pratikakis, I., & Perantonis, S. J. (2006). Adaptive degraded document image binarization. *Pattern recognition*, *39*(3), 317–327.

Gatos, B., Stamatopoulos, N., & Louloudis, G. (2011). ICDAR2009 handwriting segmentation contest. *International Journal on Document Analysis and Recognition (IJDAR)*, *14*(1), 25–33.

Gers, F. (2001). Long Short-Term Memory in Recurrent Neural Networks TH ESE. *2366*.

Gers, F. A., & Schmidhuber, J. (2000). Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, vol. 3, (pp. 189–194). IEEE.

Gers, F. A., & Schmidhuber, J. (2001). LSTM recurrent networks learn simple context-free and context-sensitive languages. *Neural Networks, IEEE Transactions on*, *12*(6), 1333–1340.

Gers, F. A., Schraudolph, N. N., & Schmidhuber, J. (2003). Learning precise timing with LSTM recurrent networks. *The Journal of Machine Learning Research*, *3*, 115–143.

Ghosh, D., Dube, T., & Shivaprasad, A. P. (2010). Script recognition : A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *32*(12), 2142–2161.

Giménez, A., Khoury, I., & Juan, A. (2010). Windowed Bernoulli mixture HMMs for Arabic handwritten word recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2010 International Conference on*, (pp. 533–538). IEEE.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, (pp. 249–256).

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, *40*(3-4), 237–264.

Gorski, N., Anisimov, V., Augustin, E., Baret, O., Price, D., & Simon, J.-C. (1999). A2ia check reader: A family of bank check recognition systems. In *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on*, (pp. 523–526). IEEE.

Graves, A. (2012). Sequence Transduction with Recurrent Neural Networks. In *ICML*.

Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *International Conference on Machine learning*, (pp. 369–376).

Graves, A., Fernández, S., & Schmidhuber, J. (2007). Multi-Dimensional Recurrent Neural Networks. *CoRR*, *abs/0705.2*.

Graves, A., & Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, (pp. 1764–1772).

Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., & Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *31*(5), 855–68.

Graves, A., Mohamed, A.-r., & Hinton, G. (2013a). Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (pp. 6645–6649). IEEE.

Graves, A., Mohamed, A.-R., & Hinton, G. (2013b). Speech Recognition with Deep Recurrent Neural Networks. In *proc ICASSP*, 3.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*(5-6), 602–610.

Graves, A., & Schmidhuber, J. (2008). Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, (pp. 545–552).

Graves, A., & Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems*, (pp. 545–552).

Grosicki, E., Carree, M., Brodin, J.-M., & Geoffrois, E. (2009). Results of the rimes evaluation campaign for handwritten mail processing. In *Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on*, (pp. 941–945). IEEE.

Grosicki, E., & El-Abed, H. (2011). Icdar 2011-french handwriting recognition competition. In *International Conference on Document Analysis and Recognition (ICDAR2011)*, (pp. 1459–1463). IEEE.

Guillevic, D., & Suen, C. Y. (1998). Recognition of legal amounts on bank cheques. *Pattern Analysis and Applications*, *1*(1), 28–41.

Gunawardana, A., & Byrne, W. (2001). Discriminative speaker adaptation with conditional maximum likelihood linear regression. In *INTERSPEECH*, (pp. 1203–1206).

Günter, S., & Bunke, H. (2004). HMM-based handwritten word recognition: on the optimization of the number of states training iterations and Gaussian components. *Pattern Recognition*, *37*(10), 2069–2079.

Guthrie, D., Allison, B., Liu, W., Guthrie, L., & Wilks, Y. (2006). A closer look at skip-gram modelling. In *Proceedings of the 5th international Conference on Language Resources and Evaluation (LREC-2006)*, (pp. 1–4).

Haffner, P. (1993). Connectionist speech recognition with a global MMI algorithm. In *EUROSPEECH*.

Hamdani, M., Doetsch, P., & Ney, H. (2014). Improvement of Context Dependent Modeling For Arabic Handwriting Recognition. In *14th International Conference on Frontiers in Handwriting Recognition (ICFHR2014)*, (pp. –).

Hammerla, N., Plötz, T., Vajda, S., & Fink, G. (2010). Towards feature learning for HMM-based offline handwriting recognition. In *International Workshop on Frontiers of Arabic Handwriting Recognition, Istanbul, Turkey*.

Hennebert, J., Ris, C., Bourlard, H., Renals, S., & Morgan, N. (1997). Estimation of global posteriors and forward-backward training of hybrid HMM/ANN systems.

Hermansky, H., Ellis, D. P., & Sharma, S. (2000). Tandem connectionist feature extraction for conventional HMM systems. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, vol. 3, (pp. 1635–1638). IEEE.

Hinton, G. (2010). A practical guide to training restricted Boltzmann machines. *Momentum*, *9*(1), 926.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, *14*(8), 1771–1800.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*(02), 107–116.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, *79*(8), 2554–2558.

Hull, J. J. (1998). Document image skew detection: Survey and annotated bibliography . In *BT - Document Analysis Systems II. Word Scientific*, (pp. 40–64).

Ide, N., & Suderman, K. (2007). The open american national corpus (oanc). URL http://www.americannationalcorpus.org/OANC/index.html

Jelinek, F., & Mercer, R. L. (1980). Interpolated Estimation of Markov Source Parameters from Sparse Data. In *Proceedings of the Workshop on Pattern Recognition in Practice*.

Johansson, S. (1980). The LOB corpus of British English texts: presentation and comments. *ALLC journal*, *1*(1), 25–36.

Juang, B.-H. (1985). Maximum-Likelihood Estimation for Mixture Multivariate Stochastic Observations of Markov Chains. *AT&T technical journal*, *64*(6), 1235–1249.

Kaltenmeier, A., Caesar, T., Gloger, J. M., & Mandler, E. (1993). Sophisticated topology of hidden Markov models for cursive script recognition. In *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*, (pp. 139–142). IEEE.

Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, *35*(3), 400–401.

Kermorvant, C., Bianne, A.-L., Marty, P., & Menasri, F. (2009). From Isolated Handwritten Characters to Fields Recognition: There's Many a Slip twixt Cup and Lip. In *International Conference on Document Analysis and Recognition*.

Kermorvant, C., Bianne-Bernard, A.-L., Bluche, T., & Louradour, J. (2012). On using alternative recognition candidates and scores for handwritten documents classification. Tech. Rep. A2iA-RR-2012-1, A2iA.

Kim, G., & Govindaraju, V. (1997). A lexicon driven approach to handwritten word recognition for real-time applications. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *19*(4), 366–379.

Kimura, F., Shridhar, M., & Chen, Z. (1993). Improvements of a lexicon directed algorithm for recognition of unconstrained handwritten words. In *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*, (pp. 18–22). IEEE.

Kingsbury, B. (2009). Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2009)*, (pp. 3761–3764). IEEE.

Klakow, D., & Peters, J. (2002). Testing the correlation of word error rate and perplexity. *Speech Communication*, *38*(1), 19–28.

Knerr, S., Augustin, E., Baret, O., & Price, D. (1998). Hidden Markov model based word recognition and its application to legal amount reading on French checks. *Computer Vision and Image Understanding*, *70*(3), 404–419.

Kneser, R., & Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, vol. 1, (pp. 181–184). IEEE.

Koerich, A. L., Sabourin, R., & Suen, C. Y. (2003). Large vocabulary off-line handwriting recognition: A survey. *Pattern Analysis & Applications*, *6*(2), 97–121.

Konig, Y., Bourlard, H., & Morgan, N. (1996). Remap: Recursive estimation and maximization of a posteriori probabilities-application to transition-based connectionist speech recognition. *Advances in Neural Information Processing Systems*, (pp. 388–394).

Kozielski, M., Doetsch, P., Hamdani, M., & Ney, H. (2014a). Multilingual Off-line Handwriting Recognition in Real-world Images. (pp. 1–1).

Kozielski, M., Doetsch, P., Ney, H., et al. (2013a). Improvements in RWTH's System for Off-Line Handwriting Recognition. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, (pp. 935–939). IEEE.

Kozielski, M., Forster, J., & Ney, H. (2012). Moment-Based Image Normalization for Handwritten Text Recognition. In *ICFHR*, (pp. 256–261). Citeseer.

Kozielski, M., Nuhn, M., Doetsch, P., & Ney, H. (2014b). Towards Unsupervised Learning for Handwriting Recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, (pp. 549–554).

Kozielski, M., Rybach, D., Hahn, S., Schluter, R., & Ney, H. (2013b). Open vocabulary handwriting recognition using combined word-level and character-level language models. In *38th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP2013)*, (pp. 8257–8261). IEEE.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, (pp. 1097–1105).

Krogh, A. (1994). Hidden Markov models for labeled sequences. In *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, vol. 2, (pp. 140–144). IEEE.

Krogh, A., Brown, M., Mian, I. S., Sjölander, K., & Haussler, D. (1994). Hidden Markov models in computational biology: Applications to protein modeling. *Journal of molecular biology*, *235*(5), 1501–1531.

Lam, L., Lee, S.-W., & Suen, C. Y. (1992). Thinning methodologies-a comprehensive survey. *IEEE Transactions on pattern analysis and machine intelligence*, *14*(9), 869–885.

Le, H.-S., Oparin, I., Allauzen, A., Gauvain, J., & Yvon, F. (2011). Structured output layer neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, (pp. 5524–5527). IEEE.

Le Cun, Y., Bottou, L., & Bengio, Y. (1997). Reading checks with multilayer graph transformer networks. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, vol. 1, (pp. 151–154). IEEE.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541–551.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lee, S.-W., & Kim, Y.-J. (1995). A new type of recurrent neural network for handwritten character recognition. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, vol. 1, (pp. 38–41). IEEE.

Leggetter, C. J., & Woodland, P. C. (1995). Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer Speech & Language*, *9*(2), 171–185.

Lemaıtre, M., Grosicki, E., Geoffrois, E., & Prêteux, F. (2008). Off-line handwritten word recognition based on a bidimensional Markovian approach with a large vocabulary.

Leroux, M., Salome, J., & Badard, J. (1991). Recognition of cursive script words in a small lexicon. In *Proceedings Int. Conf. on Document Analysis and Recognition*, (pp. 774–782).

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions insertions and reversals. In *Soviet physics doklady*, vol. 10, (p. 707).

Levin, E., & Pieraccini, R. (1992). Dynamic planar warping for optical character recognition. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, vol. 3, (pp. 149–152). IEEE.

Likforman-Sulem, L., Zahour, A., & Taconet, B. (2007). Text line segmentation of historical documents: a survey. *International Journal of Document Analysis and Recognition (IJDAR)*, *9*(2-4), 123–138.

Lopresti, D., & Kavallieratou, E. (2010). Ruling line removal in handwritten page images. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, (pp. 2704–2707). IEEE.

Louloudis, G., Gatos, B., Pratikakis, I., & Halatsis, C. (2009). Text line and word segmentation of handwritten documents. *Pattern Recognition*, *42*(12), 3169–3183.

Louradour, J., Bluche, T., Bianne-Bernard, A.-L., Menasri, F., & Kermorvant, C. (2012). De l'usage des scores et des alternatives de reconnaissance pour la classification d'images de documents manuscrits. In *Colloque International Francophone sur l'Ecrit et le Document (CIFED)*.

Maas, A. L., Hannun, A. Y., Jurafsky, D., & Ng, A. Y. (2014). First-Pass Large Vocabulary Continuous Speech Recognition using Bi-Directional Recurrent DNNs. *arXiv preprint arXiv:1408.2873*.

Madhvanath, S., & Govindaraju, V. (1996). Holistic lexicon reduction for handwritten word recognition. In *Electronic Imaging: Science & Technology*, (pp. 224–234). International Society for Optics and Photonics.

Madhvanath, S., & Krpasundar, V. (1997). Pruning large lexicons using generalized word shape descriptors. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, vol. 2, (pp. 552–555). IEEE.

Mao, S., Rosenfeld, A., & Kanungo, T. (2003). Document structure analysis algorithms: a literature survey. In *Electronic Imaging 2003*, (pp. 197–207). International Society for Optics and Photonics.

Marti, U.-V., & Bunke, H. (2000). Handwritten sentence recognition. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, vol. 3, (pp. 463–466). IEEE.

Marti, U.-V., & Bunke, H. (2001). Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition system. *International journal of Pattern Recognition and Artificial intelligence*, *15*(01), 65–90.

Marti, U.-V., & Bunke, H. (2002). The IAM-database: an English sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, *5*(1), 39–46.

Matan, O., Burges, C. J., LeCun, Y., & Denker, J. S. (1991). Multi-digit recognition using a space displacement neural network. In *NIPS*, (pp. 488–495). Citeseer.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

Menasri, F., Louradour, J., Bianne-Bernard, A.-L., & Kermorvant, C. (2012). The A2iA French handwriting recognition system at the Rimes-ICDAR2011 competition. In *Document Recognition and Retrieval Conference*, vol. 8297.

Messina, R., & Kermorvant, C. (2014). Surgenerative Finite State Transducer n-gram for Out-Of-Vocabulary Word Recognition. In *11th IAPR Workshop on Document Analysis Systems (DAS2014)*, (pp. 212–216).

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., & Khudanpur, S. (2010). Recurrent Neural Network Based Language Model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Mikolov, T., Kombrink, S., Burget, L., Cernocky, J., & Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, (pp. 5528–5531). IEEE.

Mohamed, A.-r., Hinton, G., & Penn, G. (2012). Understanding how deep belief networks perform acoustic modelling. In *37th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP2012)*, (pp. 4273–4276). IEEE.

Mohamed, M., & Gader, P. (1996). Handwritten word recognition using segmentation-free hidden Markov modeling and segmentation-based dynamic programming techniques. *IEEE transactions on pattern analysis and machine intelligence*, *18*(5), 548–554.

Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, *2*(01), 61–80.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational linguistics*, *23*(2), 269–311.

Mohri, M., Pereira, F., & Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, *16*(1), 69–88.

Mohri, M., Pereira, F., & Riley, M. (2008). Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*, (pp. 559–584). Springer.

Morillot, O., Likforman-Sulem, L., & Grosicki, E. (2013a). Comparative study of HMM and BLSTM segmentation-free approaches for the recognition of handwritten text-lines. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, (pp. 783–787). IEEE.

Morillot, O., Likforman-Sulem, L., & Grosicki, E. (2013b). New baseline correction algorithm for text-line recognition with bidirectional recurrent neural networks. *Journal of Electronic Imaging*, *22*(2), 023028–023028.

Morin, F., & Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *AISTATS*, vol. 5, (pp. 246–252). Citeseer.

Morita, M., El Yacoubi, A., Sabourin, R., Bortolozzi, F., & Suen, C. Y. (2001). Handwritten month word recognition on Brazilian bank cheques. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, (pp. 972–976). IEEE.

Moysset, B., Bluche, T., Knibbe, M., Benzeghiba, M. F., Messina, R., Louradour, J., & Kermorvant, C. (2014). The A2iA Multi-lingual Text Recognition System at the second Maurdor Evaluation. In *14th International Conference on Frontiers in Handwriting Recognition (ICFHR2014)*, (pp. 297–302).

Natarajan, P., Lu, Z., Schwartz, R., Bazzi, I., & Makhoul, J. (2001). Multilingual machine printed OCR. *International Journal of Pattern Recognition and Artificial Intelligence*, *15*(01), 43–63.

Natarajan, P., Saleem, S., Prasad, R., MacRostie, E., & Subramanian, K. (2008). Multi-lingual offline handwriting recognition using hidden Markov models: A script-independent approach. In *Arabic and Chinese Handwriting Recognition*, (pp. 231–250). Springer.

Ney, H., Essen, U., & Kneser, R. (1994). On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, *8*(1), 1–38.

Nion, T., Menasri, F., Louradour, J., Sibade, C., Retornaz, T., Métaireau, P.-Y., & Kermorvant, C. (2013). Handwritten information extraction from historical census documents. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, (pp. 822–826). IEEE.

Normandin, Y. (1992). Hidden Markov models maximum mutual information estimation and the speech recognition problem.

Olshausen, B. A., et al. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, *381*(6583), 607–609.

Oprean, C., Likforman-Sulem, L., Popescu, A., & Mokbel, C. (2013). Using the Web to create dynamic dictionaries in handwritten out-of-vocabulary word recognition. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, (pp. 989–993). IEEE.

Otsu, N. (1979). A Threshold Selection Method from Grey-Level Histograms. *IEEE Transactions on Systems, Man and Cybernetics*, *9*(1), 62–66.

Papandreou, a., & Gatos, B. (2012). Word Slant Estimation Using Non-horizontal Character Parts and Core-Region Information. *2012 10th IAPR International Workshop on Document Analysis Systems*, (pp. 307–311).
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6195384

Paquet, T., & Lecourtier, Y. (1993). Automatic reading of the literal amount of bank checks. *Machine Vision and Applications*, *6*(2-3), 151–162.

Parisse, C. (1996). Global word shape processing in off-line recognition of handwriting. *IEEE transactions on pattern analysis and machine intelligence*, *18*(4), 460–464.

Park, H.-S., & Lee, S.-W. (1998). A truly 2-D hidden Markov model for off-line handwritten character recognition. *Pattern Recognition*, *31*(12), 1849–1864.

Pastor, M., Toselli, A. H., & Vidal, E. (2004). Projection profile based algorithm for slant removal. In *Proceedings of the 2004 International Conference on Image Analysis and Recognition (ICIAR04)*.

Pesch, H., Hamdani, M., Forster, J., & Ney, H. (2012). Analysis of Preprocessing Techniques for Latin Handwriting Recognition. *ICFHR*, *12*, 18–20.

Pham, V., Bluche, T., Kermorvant, C., & Louradour, J. (2014). Dropout improves recurrent neural networks for handwriting recognition. In *14th International Conference on Frontiers in Handwriting Recognition (ICFHR2014)*, (pp. 285–290).

Plamondon, R., & Srihari, S. N. (2000). Online and off-line handwriting recognition: a comprehensive survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *22*(1), 63–84.

Plötz, T., & Fink, G. A. (2009). Markov models for offline handwriting recognition: a survey. *International Journal on Document Analysis and Recognition (IJDAR)*, *12*(4), 269–298.

Poultney, C., Chopra, S., Cun, Y. L., et al. (2006). Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, (pp. 1137–1144).

Povey, D. (2004). *Discriminative training for large vocabulary speech recognition*. Ph.D. thesis, Ph. D. thesis, Cambridge University.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., et al. (2011). The Kaldi speech recognition toolkit. In *Workshop on Automatic Speech Recognition and Understanding (ASRU2011)*, (pp. 1–4).

Povey, D., Hannemann, M., Boulianne, G., Burget, L., Ghoshal, A., Janda, M., Karafiát, M., Kombrink, S., Motlicek, P., Qian, Y., et al. (2012). Generating exact lattices in the WFST framework. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, (pp. 4213–4216). IEEE.

Rabiner, L., & Juang, B.-H. (1986). An introduction to hidden Markov models. *ASSP Magazine, IEEE*, *3*(1), 4–16.

Rashid, S. F., Shafait, F., & Breuel, T. M. (2012). Scanning Neural Network for Text Line Recognition. In *Document Analysis Systems (DAS), 2012 10th IAPR International Workshop on*, (pp. 105–109). IEEE.

Renals, S., Morgan, N., Bourlard, H., Cohen, M., & Franco, H. (1994). Connectionist probability estimators in HMM speech recognition. *Speech and Audio Processing, IEEE Transactions on*, *2*(1), 161–174.

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Neural Networks, 1993., IEEE International Conference on*, (pp. 586–591). IEEE.

Rifai, S., Mesnil, G., Vincent, P., Muller, X., Bengio, Y., Dauphin, Y., & Glorot, X. (2011a). Higher order contractive auto-encoder. In *Machine Learning and Knowledge Discovery in Databases*, (pp. 645–660). Springer.

Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011b). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, (pp. 833–840).

Robinson, A. J. (1994). An application of recurrent nets to phone probability estimation. *Neural Networks, IEEE Transactions on*, *5*(2), 298–305.

Roeder, P. (2009). *Adapting the rwth-ocr handwriting recognition system to french handwriting*. Ph.D. thesis, Master's thesis, Human Language Technology and Pattern Recognition Group, RWTH Aachen University, Aachen. Germany.

Romero, V., i Gadea, M. P., Toselli, A. H., & Vidal, E. (2006). Criteria for handwritten off-line text size normalization. In *Procc. of The Sixth IASTED international Conference on Visualization, Imaging, and Image Processing (VIIP 06)*.

Romero, V., Sánchez, J.-A., Serrano, N., & Vidal, E. (2011). Handwritten text recognition for marriage register books. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, (pp. 533–537). IEEE.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Rosenfield, R. (2000). Two decades of statistical language modeling: Where do we go from here?

Rothacker, L., Vajda, S., & Fink, G. A. (2012). Bag-of-Features Representations for Offline Handwriting Recognition Applied to Arabic Script. In *ICFHR*, (pp. 149–154).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). Learning internal representations by error propagation. Tech. rep., DTIC Document.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*.

Sainath, T. N., Mohamed, A.-r., Kingsbury, B., & Ramabhadran, B. (2013). Deep convolutional neural networks for LVCSR. In *38th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP2013)*, (pp. 8614–8618). IEEE.

Sánchez, J. A., Mühlberger, G., Gatos, B., Schofield, P., Depuydt, K., Davis, R. M., Vidal, E., & de Does, J. (2013). tranScriptorium: a european project on handwritten text recognition. In *Proceedings of the 2013 ACM symposium on Document engineering*, (pp. 227–228). ACM.

Sánchez, J. A., Romero, V., Toselli, A., & Vidal, E. (2014). ICFHR 2014 HTRtS: Handwritten Text Recognition on tranScriptorium Datasets. In *International Conference on Frontiers in Handwriting Recognition (ICFHR)*.

Saul, L. K., & Jordan, M. I. (1990). Artificial Neural Networks. chap. Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, (pp. 112–127). Piscataway, NJ, USA: IEEE Press.

Sauvola, J., & Pietikäinen, M. (2000). Adaptive document image binarization. *Pattern recognition*, *33*(2), 225–236.

Sayre, K. M. (1973). Machine recognition of handwritten words: A project report. *Pattern recognition*, *5*(3), 213–228.

Schambach, M.-P. (2003). Model Length Adaptation of an HMM based Cursive Word Recognition System. In *ICDAR*, vol. 3, (p. 109).

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, *45*(11), 2673–2681.

Schwenk, H., & Gauvain, J.-L. (2002). Connectionist language modeling for large vocabulary continuous speech recognition. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, vol. 1, (pp. I–765). IEEE.

Senior, A., & Robinson, T. (1996). Forward-backward retraining of recurrent neural networks. *Advances in Neural Information Processing Systems*, (pp. 743–749).

Senior, A. W. (1994). *Off-line Cursive Handwriting Recognition using Recurrent Neural Networks*. Ph.D. thesis, Trinity College, University of Cambridge.

Senior, A. W., & Robinson, A. J. (1998). An off-line cursive handwriting recognition system. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *20*(3), 309–321.

Shi, Z., & Govindaraju, V. (2003). Skew detection for complex document images using fuzzy runlength. In *2013 12th International Conference on Document Analysis and Recognition*, vol. 2, (pp. 715–715). IEEE Computer Society.

Shi, Z., Setlur, S., & Govindaraju, V. (2010). Removing rule-lines from binary handwritten arabic document images using directional local profile. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, (pp. 1916–1919). IEEE.

Srihari, S. N. (2000). Handwritten address interpretation: a task of many pattern recognition problems. *International journal of pattern recognition and artificial intelligence*, *14*(05), 663–674.

Steinherz, T., Rivlin, E., & Intrator, N. (1999). Offline cursive script word recognition–a survey. *International Journal on Document Analysis and Recognition*, *2*(2-3), 90–110.

Stolcke, A. (2002). SRILM – An Extensible Language Modeling Toolkit. In *International Conference on Spoken Language Processing*.

Strauß, T., Grüning, T., Leifert, G., & Labahn, R. (2014). CITlab ARGUS for historical handwritten documents.

Stutzmann, D., Bluche, T., Lavrentev, A., Leydier, Y., & Kermorvant, C. (2015). From Text and Image to Historical Resource: Text-Image Alignment for Digital Humanists. In *Digital Humanities (DH) – to appear*.

Su, H., Li, G., Yu, D., & Seide, F. (2013). Error back propagation for sequence training of Context-Dependent Deep Networks for conversational speech transcription. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP2013)*, (pp. 6664–6668).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going Deeper with Convolutions. *CoRR*, *abs/1409.4842*.
URL http://arxiv.org/abs/1409.4842

Tay, Y. H., Lallican, P.-M., Khalid, M., Knerr, S., & Viard-Gaudin, C. (2001). An analytical handwritten word recognition system with word-level discriminant training. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, (pp. 726–730). IEEE.

Thomas, S., Chatelain, C., Paquet, T., & Heutte, L. (2013). Un modèle neuro markovien profond pour l'extraction de séquences dans des documents manuscrits. *Document numérique*, *16*(2), 49–68.

Tillmann, C., & Ney, H. (1996). Selection criteria for word trigger pairs in language modeling. In *Grammatical Interference: Learning Syntax from Sentences*, (pp. 95–106). Springer.

Tong, A., Przybocki, M., Maergner, V., & El Abed, H. (2014). NIST 2013 Open Handwriting Recognition and Translation (OpenHaRT13) Evaluation. In *11th IAPR Workshop on Document Analysis Systems (DAS2014)*.

Toselli, A. H., Juan, A., González, J., Salvador, I., Vidal, E., Casacuberta, F., Keysers, D., & Ney, H. (2004). Integrated handwriting recognition and interpretation using finite-state models. *International Journal of Pattern Recognition and Artificial Intelligence*, *18*(04), 519–539.

Toselli, A. H., Romero, V., Pastor, M., & Vidal, E. (2010). Multimodal interactive transcription of text images. *Pattern Recognition*, *43*(5), 1814–1825.

Uchida, S., Taira, E., & Sakoe, H. (2001). Nonuniform slant correction using dynamic programming. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, (pp. 434–438). IEEE.

Veselý, K., Ghoshal, A., Burget, L., & Povey, D. (2013). Sequence-discriminative training of deep neural networks. In *14th Annual Conference of the International Speech Communication Association (INTERSPEECH2013)*, (pp. 2345–2349).

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, (pp. 1096–1103). ACM.

Vinciarelli, A. (2002). A survey on off-line cursive word recognition. *Pattern recognition*, *35*(7), 1433–1446.

Vinciarelli, A., & Bengio, S. (2002). Writer adaptation techniques in HMM based off-line cursive script recognition. *Pattern Recognition Letters*, *23*(8), 905–916.

Vinciarelli, A., Bengio, S., & Bunke, H. (2004). Offline recognition of unconstrained handwritten texts using HMMs and statistical language models. *IEEE transactions on Pattern Analysis and Machine Intelligence*, *26*(6), 709–20.

Vinciarelli, A., & Luettin, J. (2001). A new normalisation technique for cursive handwritten words. *Pattern Recognition Letters*, *22*, 1043–1050.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, *13*(2), 260–269.

Wager, S., Wang, S., & Liang, P. S. (2013). Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems*, (pp. 351–359).

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, *37*(3), 328–339.

Wang, S., Uchida, S., & Liwicki, M. (2012). Part-based method on handwritten texts. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, (pp. 339–342). IEEE.

Wei, X., VIARD-GAUDIN, C., & MOUCHERE, H. (2013). University De Nantes.

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Williams, R. J., & Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. *Back-propagation: Theory, architectures and applications*, (pp. 433–486).

Witten, I. H., & Bell, T. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *Information Theory, IEEE Transactions on*, *37*(4), 1085–1094.

Wolf, C., Jolion, J., & Chassaing, F. (2002). Text localization enhancement and binarization in multimedia documents. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 2, (pp. 1037–1040). IEEE.

Xu, H., Povey, D., Mangu, L., & Zhu, J. (2011). Minimum Bayes Risk decoding and system combination based on a recursion for edit distance. *Computer Speech & Language*, *25*(4), 802–828.

Yan, Y., Fanty, M., & Cole, R. (1997). Speech recognition using neural networks with forward-backward probability generated targets. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, vol. 4, (pp. 3241–3241). IEEE Computer Society.

You, D., & Kim, G. (2002). Slant correction of handwritten strings based on structural properties of Korean characters. In *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on*, (pp. 467–472). IEEE.

Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., et al. (1997). *The HTK book*, vol. 2. Entropic Cambridge Research Laboratory Cambridge.

Yu, D., Deng, L., Seide, F. T. B., & Li, G. (2011). Discriminative pretraining of deep neural networks. US Patent App. 13/304,643.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zimmermann, M., & Bunke, H. (2002). Hidden Markov model length optimization for handwriting recognition systems. In *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on*, (pp. 369–374). IEEE.

Zimmermann, M., Chappelier, J., & Bunke, H. (2006). Offline grammar-based recognition of handwritten sentences. *IEEE transactions on pattern analysis and machine intelligence*, *28*(5), 818–821.