GTE Laboratories Technical Note TN87-509.1   May 1987   Corrected Aug 1989

Implementation Details of the TD(λ) Procedure for the Case of Vector Predictions and Backpropagation

Richard S. Sutton
GTE Laboratories Incorporated

This note will probably ultimately become an appendix to the technical report "Learning to Predict by the Methods of Temporal Differences". We make a more detailed presentation of the equations for the "TD(λ)" procedure introduced in that report.

The outcomes z predicted by the TD(λ) procedure can be either scalars or vectors. In the technical report, for simplicity all equations were given for the case in which z is a scalar. A second limitation of the equations given in the report is that they did not spell out fully the algorithmic procedure for doing backpropagation in the TD case. Since both of these extensions involve substantial additional notation, and yet add little to the main ideas of that report, it is probably appropriate to not include them there. Instead, we present these details here.

Let y_i^t be the output at time t of the i^{th} unit of a multi-layer feedforward network. (Assume the observation vectors are provided to the network as the outputs of a set of "dummy" input units.) Let O be the set of the indices of the *output* units of the net. For $k \in O$, y_k^t is also denoted P_k^t. For $k \in O$, z_k is the component of the outcome vector corresponding to output unit k. Ideally, z_k is predicted by each P_k^t, $t = 1, \ldots, m$, where m is the number of observation vectors (the x_t in the technical report) presented before the outcome. By definition, $P_k^{m+1} = z_k$.

Let w_{ij}^t be the weight at time t of the connection from unit i to unit j. Let FO_j be the fan-out of unit j, i.e., the set of indices of the units with connections *from* unit j. Similarly, let FI_j be the fan-in of unit j, i.e., the set of indices of the units with connections *to* unit j. These latter units contribute to the weighted sum s_j^t at unit j as follows:

$$s_j^t = \sum_{i \in FI_j} w_{ij}^t y_i^t.$$

This sum then defines unit j's output as

$$y_j^t = f(s_j^t) = \frac{1}{1+e^{-s_j^t}}.$$

The case of TD(0)

The $\lambda = 0$ case has a particularly straightforward implementation that closely mirrors that of conventional backpropagation. We define an error or energy function for each step consisting of the sum of the squared temporal-difference errors:

$$E^t = \sum_{k \in O} (P_k^{t+1} - P_k^t)^2.$$

In forming the update rule, we only want to take into account the effects of the weights on the earlier predictions, P_k^t, not the later predictions, P_k^{t+1}. The weight update rule is then

$$\begin{aligned} w_{ij}^{t+1} &= w_{ij}^t - \alpha \sum_{k \in O} \left(\frac{\partial E^t}{\partial P_k^t} \frac{\partial P_k^t}{\partial w_{ij}^t} \right) \\ &= w_{ij}^t - \alpha \frac{\partial E^t}{\partial P_j^t} \frac{\partial P_j^t}{\partial w_{ij}^t} \\ &= w_{ij}^t - \alpha \frac{\partial E^t}{\partial s_j^t} \frac{\partial s_j^t}{\partial w_{ij}^t} \\ &= w_{ij}^t + \alpha \delta_j^t y_i^t, \end{aligned}$$

where $\delta_j^t = -\frac{\partial E^t}{\partial s_j^t}$ is computed by a backpropagation process defined by:

$$\delta_i^t = -\frac{\partial E^t}{\partial s_i^t} = \begin{cases} (P_i^{t+1} - P_i^t) y_i^t (1 - y_i^t), & \text{for } i \in O; \\ \sum_{j \in FO_i} -\frac{\partial E^t}{\partial s_j^t} \frac{\partial s_j^t}{\partial y_i^t} \frac{\partial y_i^t}{\partial s_i^t} = \sum_{j \in FO_i} \delta_j^t w_{ij}^t y_i^t (1 - y_i^t), & \text{otherwise.} \end{cases}$$

The case of $\lambda > 0$

The case of TD(0) is very like that of conventional backpropagation in that an error-like quantity—in this case a TD error—is backpropagated to each unit, which then multiplies that quantity by the signal on each of its input connections to determine the weight changes. The general case for TD(λ) is slightly different. In this case the backpropagation process produces an "eligibility" term for each weight. As a temporal difference is determined at each time step, it is broadcast to all weights, which combine it with their eligibilities of eligibility to determine the weight changes. The eligibilities thus perform a major portion of the credit assignment—they determine which weights are eligibile for what sort of modifications, should an overall (TD) error occur.

The weight update rule is

$$w_{ij}^{t+1} = w_{ij}^{t} + \alpha \sum_{k \in O} (P_k^{t+1} - P_k^t) e_{ijk}^t,$$

where e_{ijk}^t is the k^{th} eligibility at time t of the weight from unit i to unit j. The k^{th} eligibility is that which corresponds to output unit k, that is,

$$e_{ijk}^t = \sum_{n=1}^{t} \lambda^{t-n} \frac{\partial P_k^n}{\partial w_{ij}^n}.$$

These eligibilities are computed as follows:

$$\begin{aligned} e_{ijk}^{t+1} &= \lambda e_{ijk}^t + \frac{\partial P_k^{t+1}}{\partial w_{ij}^{t+1}} \\ &= \lambda e_{ijk}^t + \frac{\partial P_k^{t+1}}{\partial s_j^{t+1}} \frac{\partial s_j^{t+1}}{\partial w_{ij}^{t+1}} \\ &= \lambda e_{ijk}^t + \delta_{kj}^{t+1} y_i^{t+1}, \end{aligned}$$

where $\delta_{kj}^{t+1} = \frac{\partial P_k^{t+1}}{\partial s_j^{t+1}}$ is computed by a recursive backpropagation process defined as follows:

$$\delta_{ki}^t = \frac{\partial P_k^t}{\partial s_i^t} = \begin{cases} y_i^t(1-y_i^t), & \text{if } k = i; \\ 0, & \text{if } k \in O \text{ and } k \neq i; \\ \sum_{j \in FO_i} \frac{\partial P_k^t}{\partial s_j^t} \frac{\partial s_j^t}{\partial y_i^t} \frac{\partial y_i^t}{\partial s_i^t} = \sum_{j \in FO_i} \delta_{kj}^t w_{ij}^t y_i^t (1 - y_i^t), & \text{otherwise.} \end{cases}$$

We note that in the general case in which $\lambda > 0$, z is a vector, and in which each hidden unit potentially affects each output unit, the above TD procedure may be more complex than conventional backpropagation. The TD procedure uses more memory and computation by a factor of approximately the number of output units (to compute the e^t_{ijk}). However, as noted in the technical report, the TD procedure also saves memory and peak computation during online training by a factor of approximately the length of the sequence. Thus, which procedure is cheaper to implement for a particular application depends on the tradeoff between these factors, as well as on other issues, such as how each procedure can be approximated.