Connection between BPTT and RTRL

THOMAS HANSELMANN, ANTHONY ZAKNICH and YIANNI ATTIKIOUZEL

Centre for Intelligent Information Processing Systems
Department of Electrical and Electronic Engineering
The University of Western Australia
Nedlands, WA 6907, AUSTRALIA
thomash@ee.uwa.edu.au, tonko@ee.uwa.edu.au, yianni@ee.uwa.edu.au
http://ciips.ee.uwa.edu.au

Abstract: - This paper shows the connection between the Backpropagation Through Time ($BPTT$) algorithm, its truncated forms with truncation depth h, and the Recurrent Real Time Learning ($RTRL$) algorithm. The comparison is done by looking at a fully connected recurrent network, which is based on the same error function and calculations, using exact ordered derivatives. Two kind of formulas, based on total ordered derivatives, for $BPTT(h)$ are given and proven to be equivalent. Of the two formulae, the second one, can be interpreted by a target modification in the case of $h \to \infty$. Further, a combination of $BPTT$ and $RTRL$ is proposed to account for possible instabilities caused by weight adaptation. An overview of $BPTT$ and $RTRL$ and their implementations as well as their interpretations and uses are outlined.

Key-Words: - BPTT, RTRL, ordered derivatives, recurrent networks, backpropagation, forward-propagation.

1 Introduction

When $BPTT$ is applied to recurrent networks, it may be derived by a temporal unfolding that can be developed into a layered feedforward network, where the topology grows by one layer at every time step and then a standard backpropagation can be applied. $RTRL$ derives its name from the fact that adjustments to the synaptic weights, in a fully connected recurrent network, are made in real time, while the network continues to perform its signal processing function [1]. The objective of the learning process is to minimize some goal or target function. The target function could be the sum of instantaneous errors, $E(t)$, over some time. However, it often makes sense to weight these errors with a discount factor, γ, so that the target function to be minimized looks like $Tar = \sum_{t=k}^{\infty} \gamma^{t-k} E(t)$ where $E(t) = 1/2 \sum_{j \in Outputs} (d_j(t) - x_j(t))^2$, with $x_j(t)$ and $d_j(t)$ the actual and desired output j at time t, respectively. The discount factor has also an intuitive meaning when written as $\gamma := \frac{1}{1+p}$ with $p > 0$ acting as an interest rate which accounts for later error costs, $E(t)$, that have to be paid at time k. With this interpretation of the errors, the target, $Tar(k;h)$, can be seen as the cost-to-go function of dynamic programming from the state $\mathbf{x}(k)$ at time k up to the state $\mathbf{x}(k+h)$. In this context, the minimal cost-to-go function is of special interest, because it gives the minimal long-term cost in the interval h. However, this cost-to-go function is often not trivial to find because it is normally a complex function. In adaptive critic designs (ACD) which act as an approximation to dynamic programming, the cost-to-go is approximated by a critic-network, acting as a function approximator. Since recurrent networks are known to be more powerful than simple feedforward networks of the same size, and also able to capture temporal behavior, they are preferred for this task [2]. A drawback is that they are more difficult to adapt, but with the application of $BPTT(h)$ this simplifies to the same static-backpropagation of a feedforward network.

A quite general recurrent network structure is given by the equations (1),(2) and (3), which are also known as the generalized recurrent multilayer

perceptron (MLP), see Fig. 1.

$$x_i(t) := x_i^{ext}(t), \qquad 1 \le i \le m \quad (1)$$

$$net_i(t) := \sum_{j=1}^{i-1} w_{ij}(t)x_j(t) + \sum_{j=m+1}^{N} w_{ij}^1(t)x_j(t-1), \qquad m+1 \le i \le N \quad (2)$$

$$x_i(t) := f(net_i(t)), \qquad m+1 \le i \le N \quad (3)$$

$$Tar := Tar(k;h) = \sum_{t=k}^{k+h} \gamma^{t-k} E(t) \quad (4)$$

$$\text{with } E(t) = \frac{1}{2} \sum_{i=m+1}^{N} (d_i(t) - x_i(t))$$

This is a network ordered in structure and in time. To satisfy the objective, using a steepest descent algorithm, the total derivatives of the target function, Tar, with respect to the parameters, $w_{ij}(k)$ and $w_{ij}^1(k)$, have to be calculated. This can be done efficiently using the extended chain-rule introduced by Werbos [3, 4], [5], by the algorithm called $BPTT$. In practice the target function will be a finite sum, truncated to some depth h, yielding $BPTT(h)$. Two kinds of formulas, based on total ordered derivatives, for $BPTT(h)$ are given and proved to be equivalent. $RTRL$ uses the same target function but instead of calculating the exact derivatives it uses instantaneous estimates, $\frac{\partial E(t)}{w_{ij}(t)}$, of the true or total gradient, $F\_w_{ij}(t)$, of the target with respect to the weights. It is then shown, that by using a moving target, a similar derivation can be made to yield ordered derivatives, that are analogous to those of $BPTT$. $RTRL$ can then be interpreted as a special case with truncation depth $h = 0$.

Further, it is argued that combining $RTRL$ and $BPTT(h)$ could result in a more stable weight update, when using $RTRL$ as an estimator of future weight trajectories while doing $BPTT(h)$, than using only $RTRL$ or $BPTT$.

2 Problem Formulation

Calculation of the total gradient of the target function Tar with respect to the nodes $x_i(t)$, is straightforward by using the extended chain rule

$$F\_x_i(t) \equiv \frac{\partial^+ Tar}{\partial x_i(t)} = \frac{\partial Tar}{\partial x_i(t)} + \sum_{ld(x_i(t))} \frac{\partial^+ Tar}{\partial ld(x_i(t))} \frac{\partial ld(x_i(t))}{\partial x_i(t)} \quad (5)$$

where $ld(x_i(t))$ are the later dependencies of $x_i(t)$ in the ordered system. The external nodes are given by (1) and are the external inputs. Without loss of generality, it can be assumed that the first input, $x_1^{ext}(t) \equiv 1$ and represents a fixed bias. We will always assume that all non-external nodes, $x_j(t)$ (i.e. $j \ge m+1$), will be used in the target calculations, so the formulas stay quite general. If a node should not be included into the target calculation the direct error $e_j(t) := d_j(t) - x_j(t)$ has to simply be set to zero in the following formulae:

2.1 BPTT(h)

For $BPTT(h)$ equation (5) yields (see Appendix A)

$$\begin{aligned} F\_x_i(t) &= -\gamma^{t-k}(d_i(t) - x_i(t)) \\ &\quad + \sum_{j=i+1}^{N} F\_x_j(t) f'(net_j(t)) w_{ji}(t) \quad (6) \\ &\quad + \sum_{j=m+1}^{N} F\_x_j(t+1) f'(net_j(t+1)) w_{ji}^1(t+1) \end{aligned}$$

$$F\_x_i(t) \equiv 0 \quad \forall t > k+h \quad (7)$$

$$F\_w_{ij}(t) = F\_x_i(t) f'(net_i(t)) x_j(t) \quad (8)$$

$$F\_w_{ij}^1(t) = F\_x_i(t) f'(net_i(t)) x_j(t-1) \quad (9)$$

$$\Delta w_{ij}(k) = -\eta\, F\_w_{ij}(k) \quad (10)$$

$$\Delta w_{ij}^1(k) = -\eta\, F\_w_{ij}^1(k) \quad (11)$$

The $BPTT(h)$ algorithm calculates the weight updates at time k, by calculating $Tar(k;h)$ and then going backwards in time and structure, from $t = k+h$ and $i = N, .., 1$ using equations (6) and (7). Finally, doing the updates according to (10) and (11), with a learning rate $\eta > 0$.

An equivalent, but slightly more efficient formulation for $BPTT(h)$ can be achieved by defining

$$\gamma^{t-k} \tilde{F}\_x_i(t) := F\_x_i(t) \quad (12)$$

Starting with $t = k+h$ and $i = N,..,1$ yields $F\_x_i(t) = -\gamma^h e_i(t) + \sum_{j=i+1}^{N} F\_x_j(t) f'(net_j(t)) w_{ji}(t)$ where all $F\_x_j(t)$ contain a factor $\gamma^h = \gamma^{t-k}$ such that one can write $\tilde{F}\_x_i(t) = -e_i(t) + \sum_{j=i+1}^{N} \tilde{F}\_x_i(t) w_{ji}(t)$. Going one time step backwards, i.e. $t = k+h-1$, it follows from (6), that all the summands contain a factor γ^{h-1} and γ^h for the first and the second sum, respectively. Therefore one can write

$$\begin{aligned}\tilde{F}\_x_i(t) &= -(d_i(t) - x_i(t)) \\ &+ \sum_{j=i+1}^{N} \tilde{F}\_x_j(t) f'(net_j(t)) w_{ji}(t) \\ &+ \gamma \sum_{j=m+1}^{N} \tilde{F}\_x_j(t+1) f'(net_j(t+1)) w^1_{ji}(t+1)\end{aligned} \quad (13)$$

This is slightly more efficient, because there is only one multiplication by γ, of the second sum, but no multiplication of a power of γ, in the direct error term. At time k, $\tilde{F}\_x_i(t)$ is equal to $F\_x_i(t)$ due to (12) and so equation (13) could be used in the $BPTT(h)$ algorithm instead of (6). However, for all other times t within $k < t \leq k+h$ the total derivatives differ according to (12).

2.2 RTRL

A very compact formulation of $RTRL$ can be found in [1]. Following that derivation, but extending it slightly to account for ordered dependencies between output nodes ($x_j(t) = x_j(x_{m+1}(t),..,x_{j-1}(t))$ at time t. Defining a concatenated input vector, ξ^j, of dimension $m+2(N-m)$

$$\xi^j(t) := \begin{bmatrix} x^{ext}_1(t) \\ \vdots \\ x^{ext}_m(t) \\ x_{m+1}(t-1) \\ \vdots \\ x_N(t-1) \\ x_{m+1}(t) \\ \vdots \\ x_{j-1}(t) \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{x}^{ext}(t) \\ \mathbf{x}^j(t) \end{bmatrix} \quad (14)$$

and appropriate weight vectors, $\mathbf{w}_j(t)$ $(m+1 \leq j \leq N)$, whose elements connect elements of $\xi^j(t)$ with node $x_j(t)$, and state vector, $\mathbf{x}(t) = [x_{m+1}(t),..,x_N(t)]^T$, the system (1-3) can be written as

$$\mathbf{x}(t) := \begin{bmatrix} x_{m+1}(t) \\ \vdots \\ x_N(t) \end{bmatrix} = \begin{bmatrix} f(\mathbf{w}^T_{m+1}(t)\xi^{m+1}(t)) \\ \vdots \\ f(\mathbf{w}^T_N(t)\xi^N(t)) \end{bmatrix} \quad (15)$$

Differentiating (15) with respect to the weights $\mathbf{w}_j(t)$ using the chain rule once, yields

$$\frac{\partial \mathbf{x}^T(t)}{\partial \mathbf{w}_j(t)} = \Phi(t) \left[\frac{\partial \mathbf{x}^j{}^T(t)}{\partial \mathbf{w}_j(t)} \mathbf{W}^{int}(t) + \mathbf{U}^j{}^T(t) \right] \quad (16)$$

$$\Phi(t) := diag(f'(\mathbf{w}^T_{m+1}(t)\xi^{m+1}(t)), \ldots, f'(\mathbf{w}^T_N(t)\xi^N(t))) \quad (17)$$

$$\mathbf{W}(t) := \begin{bmatrix} \mathbf{W}^{ext}(t) \\ \mathbf{W}^{int}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{w}^{ext}_{m+1}(t),..,\mathbf{w}^{ext}_N(t) \\ \tilde{\mathbf{w}}^{int}_{m+1}(t),..,\tilde{\mathbf{w}}^{int}_N(t) \end{bmatrix} \quad (18)$$

$$\mathbf{U}^j(t) := \begin{bmatrix} \mathbf{0} \\ \xi^j{}^T(t) \\ \mathbf{0} \end{bmatrix} \leftarrow (j-m)^{th} \text{ row} \quad (19)$$

where $\mathbf{w}^{int}_j(t)$ is the subvector of $\mathbf{w}_j(t)$ without the elements connecting node $x_j(t)$ to external nodes $x^{ext}_k(t)$ in $\xi^j(t)$, and, $\tilde{\mathbf{w}}_j(t)$ is equal to $\mathbf{w}(t)$ with elements $\tilde{\mathbf{w}}_{jl}(t)$ set to 0 for $l \geq j+N-m$. Equation (16) now describes recursively, the state dynamics (note that the first $N-m$ elements of $\mathbf{x}(t)$ are equal to those of $\mathbf{x}^j(t+1)$).

Having an instantaneous error $E(t) = \frac{1}{2}\mathbf{e}^T(t)\mathbf{e}(t)$ with $\mathbf{e}(t) = \mathbf{d}(t) - \mathbf{x}(t)$, as in equation (4), also the same target, $Tar(k;h)$, could be used. It is

$$\frac{\partial E(t)}{\partial \mathbf{w}_j(t)} = \frac{\partial \mathbf{e}^T(t)}{\partial \mathbf{w}_j(t)} \mathbf{e}(t) \quad (20)$$

$$= -\frac{\partial \mathbf{x}^T(t)}{\partial \mathbf{w}_j(t)} \mathbf{e}(t) \quad (21)$$

$$\frac{\partial^+ Tar(k;h)}{\partial \mathbf{w}_j(k)} = \sum_{t=k}^{k+h} \gamma^{t-k} \frac{\partial^+ E(t)}{\partial \mathbf{w}_j(k)} \quad (22)$$

$$= -\sum_{t=k}^{k+h} \gamma^{t-k} \frac{\partial^+ \mathbf{x}^T(t)}{\partial \mathbf{w}_j(k)} \mathbf{e}(t) \quad (23)$$

$$\Delta \mathbf{w}_j(k) = -\eta \frac{\partial^+ Tar(k;h)}{\partial \mathbf{w}_j(k)} \quad (24)$$

$$= \eta \sum_{t=k}^{k+h} \gamma^{t-k} \frac{\partial^+ \mathbf{x}^T(t)}{\partial \mathbf{w}_j(k)} \mathbf{e}(t) \quad (25)$$

$$\stackrel{h=0}{=} \eta \frac{\partial \mathbf{x}^T(k)}{\partial \mathbf{w}_j(k)} \mathbf{e}(k) \quad (26)$$

However, linking $\frac{\partial \mathbf{x}^T(t)}{\partial \mathbf{w}_j(t)}$ together with the gradient, $\frac{\partial^+ Tar(k;h)}{\mathbf{w}_j(t)}$, of the total derivative of the target with respect to the weights, is done easily only when $h = 0$, meaning the total gradient is approximated by the instantaneous gradient. If one keeps the weights constant during the target window from time k up to $k+h$ an easy relation can be stated using ordinary derivatives

$$\Delta \mathbf{w}_j(k) \stackrel{\mathbf{w}_j(t)=\mathbf{w}_j(k)}{=} \eta \sum_{t=k}^{k+h} \gamma^{t-k} \frac{\partial^+ \mathbf{x}^T(t)}{\partial \mathbf{w}_j(t)} \mathbf{e}(t)$$

$$= \eta \sum_{t=k}^{k+h} \gamma^{t-k} \frac{\partial \mathbf{x}^T(t)}{\partial \mathbf{w}_j(t)} \mathbf{e}(t) \quad (27)$$

To make the $RTRL$ algorithm complete, the initial condition for the recursive formula (16) needs to be given. Assuming the network is in a constant state, we can set $\frac{\partial \mathbf{x}^T(0)}{\partial \mathbf{w}_j(0)} = \mathbf{0}$ for all j. One potential problem of the $RTRL$ algorithm is that when the learning rate η is too large, additional feedback produced by the weight changes can cause instability. Another possibility would be to update weights only every h^{th} time step ($h > 1$). This would be a "forward propagation through time".

2.3 Change of targets

So far we have been looking at $BPTT(h)$ with a fixed target, $Tar(k;h)$, at a certain time k. Now, we consider 'shifted' and 'moving' targets, $Tar(k+q;h)$ and $Tar(t+q;h)$, for total derivatives with respect to quantities y at time $t+q$, where $q > 0$ and define them as

$$F\_y(t+q) \equiv \frac{\partial^+ Tar(k+q;h)}{\partial y(t+q)} \quad \text{shifted} \tag{28}$$

$$F_m\_y(t+q) \equiv \frac{\partial^+ Tar(t+q;h)}{\partial y(t+q)} \quad \text{moving} \tag{29}$$

$$Tar(k+q;h) = \sum_{t=k+q}^{k+q+h} \gamma^{t-(k+q)} E(t) \tag{30}$$

where equation (30) is simply equation (4) for $k := k+q$. The difference between 'shifted' and 'moving' target is that with the former k is fixed, whereas in the latter, the target calculation always starts at the current time $t+q$ (q is just a constant introduced to make the formulas more similar). Using equation (6) again, but this time with a target $Tar(t;h)$ yields

$$\begin{aligned} F_m\_x_i(t) &= -(d_i(t) - x_i(t)) \\ &+ \sum_{j=i+1}^{N} F_m\_x_j(t) f'(net_j(t)) w_{ji}(t) \\ &+ \sum_{j=m+1}^{N} F\_x_j(t+1) f'(net_j(t+1)) w^{int}_{ji}(t+1) \end{aligned} \tag{31}$$

$$F\_x_i(t') \equiv 0 \quad \forall t' > t+h \tag{32}$$

It is obvious that the 'shifted' targets are related by equation (34) and for $q = 1$ this yields equation (35).

$$\gamma^q Tar(k+q;h) = \gamma^q \sum_{t=k+q}^{k+q+h} \gamma^{t-(k+q)} E(t) = \sum_{t=k+q}^{k+q+h} \gamma^{t-k} E(t) \tag{33}$$

$$= Tar(k;h) - \sum_{t=k}^{k+q-1} \gamma^{t-k} E(t) + \sum_{t=k+h+1}^{k+h+q} \gamma^{t-k} E(t) \tag{34}$$

$$Tar(k;h) = \gamma Tar(k+1;h) + E(k) - \gamma^{h+1} E(k+1+h) \tag{35}$$

Using equation (35) to calculate

$$\frac{\partial^+ Tar(k;h)}{\partial x_j(t+1)} = \frac{\partial^+ (\gamma Tar(k+1;h) + E(k) - \gamma^{h+1} E(k+1+h))}{\partial x_j(t+1)} \tag{36}$$

$$= \gamma \frac{\partial^+ Tar(k+1;h)}{\partial x_j(t+1)} - \gamma^{h+1} \frac{\partial^+ E(k+1+h)}{\partial x_j(t+1)} \tag{37}$$

$$\stackrel{h\to\infty}{=} \gamma \frac{\partial^+ Tar(k+1;h)}{\partial x_j(t+1)} = \gamma\, F\_x_j(t+1) \tag{38}$$

The term $\frac{\partial^+ E(k)}{\partial x_j(t+1)}$ is zero, because $t \geq k$ always, and therefore $E(k)$ does not depend on $x_j(t+1)$. Using equation (37) with $k := t$ and substitution of $\frac{\partial^+ Tar(t;h)}{\partial x_j(t+1)}$ ($= F\_x_j(t+1)$) in equation (31) to have a 'moving' target yields

$$\begin{aligned} F_m\_x_i(t) &= -(d_i(t) - x_i(t)) \\ &+ \sum_{j=i+1}^{N} F_m\_x_j(t) f'(net_j(t)) w_{ji}(t) \\ &+ \gamma \sum_{j=m+1}^{N} \Big[\Big(F_m\_x_j(t+1) - \gamma^h \frac{\partial^+ E(t+1+h)}{\partial x_j(t+1)} \Big) \\ &\qquad \cdot f'(net_j(t+1)) w^1_{ji}(t+1) \Big] \end{aligned} \tag{39}$$

$$F_m\_x_i(t') \equiv 0 \quad \forall t' > t+h \tag{40}$$

But this has the form of equation (13) when $h \to \infty$ and therefore can be interpreted as $BPTT(\infty)$ with moving targets. $F_m\_x_i(t)$ are now the total derivatives of the moving targets, $Tar(t;h)$, with respect to the nodes $x_i(t)$.

On the other hand for $h = 0$ (note the second sum in (39) is always 0 when $h = 0$) it can be considered as $BPTT(0)$ with moving target (or fixed, since they are the same for $h = 0$) which is the instantaneous gradient of the current error $E(t)$ with respect to the node $x_i(t)$, or with respect to the weights $w_{ij}(t)$ when using (8), (9). The latter target is exactly as in $RTRL$.

Another interesting point is that when $\gamma \to 1$, $F\_x_i(t)$ of equation(6) must be equal to $F_m\_x_i(t)$ of equation (39) with $h \to \infty$, and assuming $E(\infty) \equiv 0$. This means that the past errors do

not count in weight updates but only future errors contribute to weight updates. This is not surprising because future errors have an infinite support, whereas the number of past errors from some past starting time up to the current time t, is always finite.

3 Problem Solution

In the previous sections we derived two formulae for $BPTT(h)$ and showed the relation with $RTRL$. Now we would like to improve the weight updates in the $BPTT(h)$ algorithm by estimating the influence of future weight updates. As it can be seen from equation (6) the current weights, $w_{ij}(t)$, at times t are used to calculate the total gradient, $F\_w_{ij}(k)$, which is then used to update the weights according to (8) and (9), respectively. However, it would be desirable to know how this weight update at time k will influence future values and therefore have estimates, $\hat{w}_{ji}(t)$, of $w_{ji}(t)$ (and similar for w^1_{ji}). It is proposed that these estimates be calculated with the $RTRL$ algorithm, which would then run in parallel with the forward calculation in $BPTT(h)$, which would use the estimated weights instead of the weights given at starting (or updating) time k. However, these weights would have to be stored for use in the backpropagation sweep as well. This is different from the combination of $RTRL$ and $BPTT$ proposed by Williams and Zipser [6] which we have given a summary, adapted to our notation, in the appendix B. Since $RTRL$ is much more computationally intensive for large networks than $BPTT$, one could also use $BPTT(h')$ with $(h' < h)$ to get estimated weights and then use them in the $BPTT(h)$ calculation. To reduce computation further, one could also just use estimates, $\hat{Tar}(k;h)$, for the target based on expected weight changes during its calculation from time k to $k+h$.

Before it was assumed that in the $RTRL$ algorithm at the start time $k = 0$ the instantaneous gradient $\frac{\partial \mathbf{x}^T(0)}{\partial \mathbf{w}_j(0)}$ is zero, however, one could now use $BPTT(h)$ for initial values.

4 Conclusions

While $BPTT(h)$ was introduced some time ago, [3, 7], and reinvented many times (see references in [6]), here it is developed in detail using the pragmatic notation of ordered derivatives introduced by Werbos. Two interpretations (equation (12) and (39)) of the ordered derivatives are given for the direct calculated ordered derivatives (6). Further, $RTRL$ is restated, taking account of the structural order $(x_j(t) = x_j(t)(x_1(t), .., x_{j-1}(t)))$, so that it can be directly compared with $BPTT$. With the update only on every h^{th} time step, $RTRL$ can be seen as forward propagation through time with truncation depth h. Also, a combination of $RTRL$ and $BPTT$ has been proposed. However, to reduce computational effort, it may be possible to estimate nodes, $\hat{x}_i(t)$, or even only targets, $\hat{Tar}(k;h)$, based on the expected influence of the next weight update and calculate the effective weight update $\Delta w_{ij}(k)$ using these estimates. Feldkamp and Puskorius have been very successful in combining estimation of weights based on Kalman filtering [8].

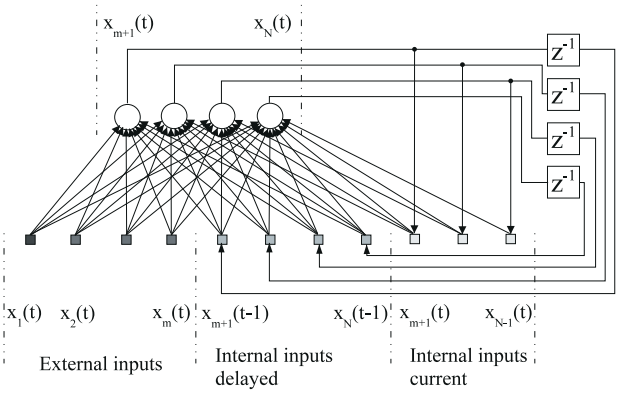

Fig. 1 *Graphical representation of the recurrent network given by equations (1-3)*

Appendix A

Here, it is shown in detail how to arrive at equation (6). The analogous steps are also valid for (31). Equation (5) can be written as

$$
\begin{aligned}
F\_x_i(t) &= \frac{\partial Tar}{\partial x_i(t)} + \sum_{t' \geq t} \sum_{j=1}^{N} \frac{\partial^+ Tar}{\partial net_j(t')} \frac{\partial net_j(t')}{\partial x_i(t)} \\
&= -\gamma^{t-k}(d_i(t) - x_i(t)) + \sum_{j=i+1}^{N} \frac{\partial^+ Tar}{\partial net_j(t)} \frac{\partial net_j(t)}{\partial x_i(t)} \\
&\quad + \sum_{j=m+1}^{N} \frac{\partial^+ Tar}{\partial net_j(t+1)} \frac{\partial net_j(t+1)}{\partial x_i(t)} \\
&\quad + \sum_{t'>t+1} \sum_{j=m+1}^{N} \frac{\partial^+ Tar}{\partial net_j(t')} \frac{\partial net_j(t')}{\partial x_i(t)}
\end{aligned}
$$

While $\frac{\partial net_j(t')}{\partial x_i(t)} \equiv 0, \forall t' > t+1$, the last line is always zero. Further, it follows from equations (2) and (3) that

$$\begin{aligned}
\frac{\partial net_j(t)}{\partial x_i(t)} &= w_{ji}(t) \\
\frac{\partial net_j(t+1)}{\partial x_i(t)} &= w^1_{ji}(t) \\
F_net_i(t) &= \\
&\frac{\partial Tar}{\partial net_i(t)} + \sum_{t' \geq t} \sum_{j=m+1}^{N} \frac{\partial^+ Tar}{\partial x_j(t')} \frac{\partial x_j(t')}{\partial net_i(t)} \\
&= F_x_i(t) f'(net_i(t)) \\
F_net_i(t+1) &= \\
&\frac{\partial Tar}{\partial net_i(t+1)} + \sum_{t' \geq t+1} \sum_{j=m+1}^{N} \frac{\partial^+ Tar}{\partial x_j(t')} \frac{\partial x_j(t')}{\partial net_i(t+1)} \\
&= F_x_i(t+1) f'(net_i(t+1))
\end{aligned}$$

Putting these expressions together then yields equation (6).

Appendix B

In [6], Williams and Zipser describe many variants of $BPTT$, $RTRL$ and also a combination of them. However, they use constant weights, $w_{ij} = w_{ij}(t) \forall t \in [k, .., k+h]$ during the computation and use the trick to split up the truncation depth $h = h_1 + h_2$ into two intervals. This yields, using our notation, equation (41), while (43) is the first sum and (44) the second sum of (41), respectively. Combining these three equations finally yields (45) where in the first sum the $RTRL$ algorithm, started at time k and stopped at time $k + h_1 - 1$, appears and in the second $BPTT(h_2)$ started at time $k + h_1$.

$$\begin{aligned}
\frac{\partial^+ Tar}{\partial w_{ij}} &= \sum_{t=k}^{k+h} \frac{\partial^+ Tar}{\partial w_{ij}(t)} \frac{\partial w_{ij}(t)}{\partial w_{ij}} = \sum_{t=k}^{k+h} \frac{\partial^+ Tar}{\partial w_{ij}} \\
&= \sum_{t=k}^{k+h_1-1} \frac{\partial^+ Tar}{\partial w_{ij}(t)} + \sum_{t=k+h_1}^{k+h} \frac{\partial^+ Tar}{\partial w_{ij}(t)} \qquad (41)
\end{aligned}$$

$$\begin{aligned}
\sum_{t=k}^{k+h_1-1} \frac{\partial^+ Tar}{\partial w_{ij}(t)} &= \sum_{t=k}^{k+h_1-1} \sum_{l=m+1}^{N} \frac{\partial^+ Tar}{\partial x_l(k+h_1)} \frac{\partial x_l(k+h_1)}{\partial w_{ij}(t)} \\
&= \sum_{l=m+1}^{N} \frac{\partial^+ Tar}{\partial x_l(k+h_1)} \sum_{t=k}^{k+h_1-1} \frac{\partial x_l(k+h_1)}{\partial w_{ij}(t)} \\
&= \sum_{l=m+1}^{N} \frac{\partial^+ Tar}{\partial x_l(k+h_1)} \frac{\partial x_l(k+h_1)}{\partial w_{ij}} \qquad (42) \\
&= \sum_{l=m+1}^{N} F_x_l(k+h_1) \frac{\partial x_l(k+h_1)}{\partial w_{ij}} \qquad (43)
\end{aligned}$$

$$\sum_{t=k+h_1}^{k+h} \frac{\partial^+ Tar}{\partial w_{ij}(t)} = \sum_{t=k+h_1}^{k+h} \frac{\partial^+ Tar}{\partial net_i(t)} x_j(t) \qquad (44)$$

$$\begin{aligned}
\frac{\partial^+ Tar}{\partial w_{ij}} &= \sum_{l=m+1}^{N} \frac{\partial^+ Tar}{\partial x_l(k+h_1)} \frac{\partial x_l(k+h_1)}{\partial w_{ij}} \\
&\quad + \sum_{t=k+h_1}^{k+h} \frac{\partial^+ Tar}{\partial net_i(t)} x_j(t) \qquad (45)
\end{aligned}$$

References:

[1] S. Haykin. *Neural Networks a Comprehensive Foundation*, chapter 15. Prentice Hall, Upper Saddle River, NJ, 2nd edition, 1998.

[2] Danil V. Prokhorov and Feldkamp L. Primitive adaptive critics. *Proceedings of the IEEE International Conference on Neural Networks (ICNN) Houston*, 78(10):2263–2267, September 1997.

[3] P. Werbos. *Beyond regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* Ph.d. dissertation, Harvard Univ., Cambridge, MA, 1974. Reprinted in The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting.

[4] P. Werbos. How to use the chain rule for ordered derivatives. In David A. White and Donald A. Sofge, editors, *Handbook of Intelligent Control*, chapter 10.6. Van Nostrand Reinhold, New York, 1992.

[5] Stephen W. Piché. Steepest descent algorithms for neural network controllers and filter. *IEEE Transactions On Neural Networks*, 5(2):198–212, March 1994.

[6] S. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Chauvin and Rumelhart, editors, *Backpropagation: Theory, Architectures and Applications*, pages 433–486. LEA, 1995.

[7] P. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE Control*, 78(10):1550–1560, March 1990.

[8] G.V. Puskorious and L.A. Feldkamp. Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks. *IEEE Transactions On Neural Networks*, 5(2):279–297, 1994.