Journal of Machine Learning Research 1:1–12, 2012

Gradient Temporal Difference Networks

David Silver* D.SILVER@CS.UCL.AC.UK
Department of Computer Science, CSML, University College London, London, WC1E 6BT

Abstract

Temporal-difference (TD) networks (Sutton and Tanner, 2004) are a predictive representation of state in which each node is an *answer* to a *question* about future observations or questions. Unfortunately, existing algorithms for learning TD networks are known to diverge, even in very simple problems. In this paper we present the first sound learning rule for TD networks. Our approach is to develop a true gradient descent algorithm that takes account of all three roles performed by each node in the network: as state, as an answer, and as a target for other questions. Our algorithm combines gradient temporal-difference learning (Maei et al., 2009) with real-time recurrent learning (Williams and Zipser, 1994). We provide a generalisation of the Bellman equation that corresponds to the semantics of the TD network, and prove that our algorithm converges to a fixed point of this equation.

1. Introduction

Representation learning is a major challenge faced by machine learning and artificial intelligence. The goal is to automatically identify a representation of state that can be updated from the agent's sequence of observations and actions. *Predictive state representations* (PSRs) provide a promising approach to representation learning. Every state variable is observable rather than latent, and represents a specific prediction about future observations. The agent maintains its state by updating its predictions after each interaction with its environment. It has been shown that a small number of carefully chosen predictions provide a sufficient statistic for predicting all future experience; in other words that those predictions summarise all useful information from the agent's previous interactions (Singh et al., 2004). The intuition is that an agent which can effectively predict the future will be able to act effectively within its environment, for example to maximise long-term reward.

Temporal-difference networks (TD networks) are a type of PSR that may ask *compositional* predictions, not just about future observations, but about future state variables (*i.e.* predictions of predictions). This enables TD networks to operate at a more abstract level than traditional PSRs. However, temporal-difference networks suffer from a major drawback: the learning algorithm may diverge, even in simple environments.

The main contribution of this paper is to provide a sound and non-divergent learning rule for TD networks and related architectures. Unlike previous TD network learning rules, our approach is based on a true gradient descent algorithm. It uses *gradient temporal-difference* (GTD) learning to account for the compositions of predictions into the future; and backpropagation through time (BPTT) (Rumelhart et al., 1986) or *real-time recurrent*

* Supported by the Royal Society and CompLACS

learning (RTRL) (Williams and Zipser, 1989) to correctly account for the full history of previous state variables.

2. Gradient Temporal-Difference Learning

A key problem in reinforcement learning is to estimate the total discounted reward from a given state, so as to evaluate the quality of a fixed policy. In this section we consider a Markov reward process with state space $\mathcal{S}$, transition dynamics from state s to s' given by $\mathcal{T}_{s,s'} = \mathbb{P}[s'|s]$, reward r given by $\mathcal{R}_s = \mathbb{E}[r|s]$, and discount factor $0 < \gamma < 1$. The *return* counts the total discounted reward from time t, $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, and the *value function* is the expected return from state s, $V_s = \mathbb{E}[R_t \mid s_t = s]$. The Bellman equation defines a recursive relationship, $V = \mathcal{R} + \gamma \mathcal{T} V$. This equation has a unique fixed point corresponding to the true value function.

In general, the transition dynamics $\mathcal{T}$ and reward function $\mathcal{R}$ are unknown. Furthermore, the number of states $|\mathcal{S}|$ is often large, and therefore it is necessary to approximate the true value function V_s using a function approximator $V_s = f(s, \theta)$ with parameters θ. These parameters can be updated by stochastic gradient descent, so as to minimise the mean squared error between the value function and the return, $\Delta\theta_t = \alpha(R_t - V^\theta_{s_t}) \nabla_\theta V^\theta_{s_t}$, where α is a scalar step-size parameter. However, the return is high variance and also unknown at time t. The idea of *temporal-difference* (TD) learning is to substitute the return R_t with a lower variance, one-step estimate of value, called the *TD target*: $r_t + \gamma V^\theta_{s_{t+1}}$. This idea is known as *bootstrapping* (Sutton, 1988), and leads to the TD learning update, $\Delta\theta_t = \alpha \delta_t \nabla_\theta V^\theta_{s_t}$, where δ_t is the *TD error* $\delta_t = r_t + \gamma V^\theta_{s_{t+1}} - V^\theta_{s_t}$. Temporal-difference learning is particularly effective with a linear function approximator $V^\theta_s = \phi(s)^\top \theta$. Unfortunately, for non-linear function approximation, or for off-policy learning, TD learning is known to diverge (Tsitsiklis and Van Roy, 1997; Sutton et al., 2009).

Temporal-difference learning is not a true gradient descent algorithm, because it ignores the derivative of the TD target. *Gradient temporal-difference* (GTD) learning addresses this issue, by minimising an objective function corresponding to the error in the Bellman equation, by stochastic gradient descent. This objective measures the error between value function V^θ and the corresponding target given by the Bellman equation $\mathcal{R} + \gamma \mathcal{T} V^\theta$. However, the Bellman target typically lies outside the space of value functions that can be represented by function approximator f. It is therefore projected back into this space using a projection Π^θ. This gives the mean squared projected Bellman error (MSPBE), $J(\theta) = ||V^\theta - \Pi^\theta(\mathcal{R} + \gamma \mathcal{T} V^\theta)||^2_\rho$, where the squared norm $||\cdot||^2_\rho$ is weighted by the stationary distribution $\rho(s)$ of the Markov reward process. To ensure tractable computation when using a non-linear function approximator f, the projection operator Π^θ is a linear projection onto the tangent space of V^θ, $\Pi^\theta = \Phi_\theta (\Phi_\theta^\top D \Phi_\theta)^{-1} \Phi_\theta^\top D$, where D is the diagonal matrix $D = diag(\rho)$; and Φ_θ is the tangent space of V^θ, where each row is a value gradient $\phi(s) = (\Phi_\theta)_{s,\cdot} = \nabla_\theta V^\theta_s$.

The *GTD2* algorithm minimises the MSPBE by stochastic gradient descent. The MSPBE can be rewritten as a product of three expectations,

$$J(\theta) = \mathbb{E}[\phi(s)\delta]^\top \mathbb{E}\left[\phi(s)\phi(s)^\top\right]^{-1} \mathbb{E}[\phi(s)\delta] \tag{1}$$

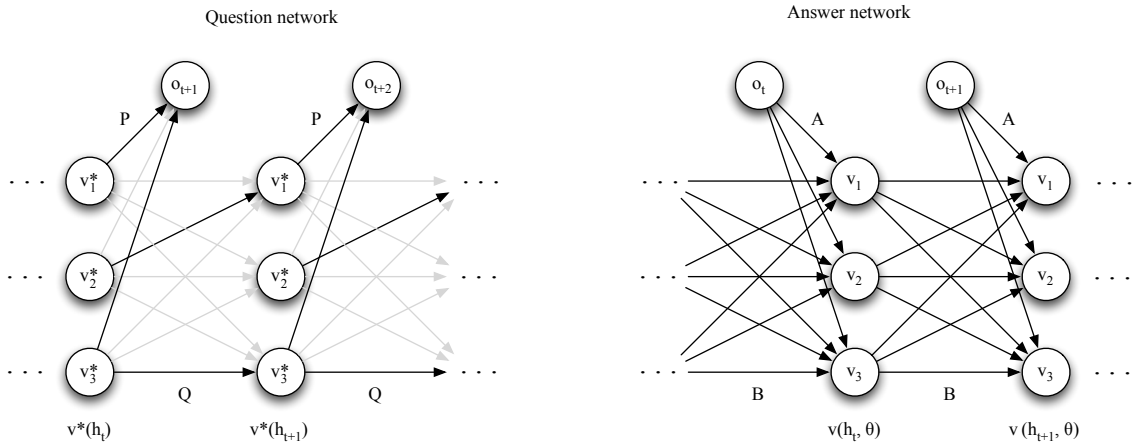

Figure 1: A TD network with 3 predictions and 1 observation. The question network gives the semantics of the predictions: v_1^* is the expected observation at the next time-step; v_2^* is the expected observation after two time-steps; and v_3^* is the expected sum of future observations. The structure of the question network is specified by weight matrices P, Q; black edges have weight 1 and grey edges have weight 0. The answer network specifies the mechanism by which answers to these questions are updated over time. Its weight matrices A, B are adjusted so that $v \approx v^*$.

which can be written as $J(\theta) = \mathbb{E}\left[\phi(s)\delta\right]^\top w_\theta$ where $w_\theta = \mathbb{E}\left[\phi(s)\phi(s)^\top\right]^{-1}\mathbb{E}\left[\phi(s)\delta\right]$. The GTD2 algorithm simultaneously estimates $w \approx w_\theta$ by stochastic gradient descent; and also minimises $J(\theta)$ by stochastic gradient descent, assuming that the current estimate is correct, i.e. that $w = w_\theta$. This gives the following updates, applied at every time-step t with step-size parameters α and β,

$$\psi_t = (\delta_t - \phi(s_t)^\top w)\nabla_\theta^2 V_s w \tag{2}$$

$$\Delta\theta_t = \alpha\left(\phi(s_t) - \gamma\phi(s_{t+1}))(\phi(s_t)^\top w\right) - \psi_t \tag{3}$$

$$\Delta w_t = \beta\phi(s_t)\left(\delta_t - \phi(s_t)^\top w\right) \tag{4}$$

The GTD2 algorithm converges to a local minimum of $J(\theta)$, even when using non-linear function approximation (Maei et al., 2009) or off-policy learning (Sutton et al., 2009). The *TDC* algorithm minimises the same objective but using a slightly different derivation,

$$\Delta\theta_t = \alpha\left(\phi(s_t)\delta - \gamma\phi(s_{t+1})(\phi(s_t)^\top w)\right) - \psi_t \tag{5}$$

3. Temporal Difference Networks

We focus now on the uncontrolled case in which an agent simply receives a time series of m-dimensional observations o_t at each time-step t. A history h_t is a length t sequence of observations, $h_t = o_1 ... o_t$. We assume that histories are generated by a Markov chain with

a probability $1-\gamma$ of terminating after every transition, to give a history of length τ. The Markov chain has (unknown) transition probabilities $T_{h,ho} = \gamma \mathbb{P}[o_{t+1} = o \mid h_t = h, \tau > t]$, with zero probability for all other transitions. This Markov chain has a well-defined distribution $d(h)$ over the set $\mathcal{H}$ of all histories, $d(h_t) = \mathbb{P}[\tau = t, o_1 ... o_t = h_t]$.

A *state representation* is a function $v(h)$ mapping histories to a vector of state variables $v_i(h)$. For a state representation to be usable online, it must be *incremental* – *i.e.* the state can be updated from one observation to the next, $v(h_{t+1}) = f(v(h_t), o_{t+1})$. A *predictive state representation* (PSR) is a state representation in which all state variables are predictions of future events. The *target* of a prediction is a function of future observations, $z_t = g(o_{t+1}, o_{t+2}, ...)$. The *true answer* for a prediction is the expected value of the target, given a history of observations h, $v^*(h) = \mathbb{E}[z_t \mid h_t = h]$. The PSR learns an estimated *answer* $v(h, \theta) \approx v^*(h)$ to each prediction, by adjusting parameters θ. The answer vector $v(h_t, \theta)$ is the state representation used by the agent at time-step t. For example, in the original PSR, g was a vector of indicator functions over future observations, and f was a linear function (Singh et al., 2004).

A temporal-difference (TD) network is comprised of a *question network* and an *answer network*. The question network defines the semantics of the compositional predictions, i.e. how a target depends on subsequent targets. We focus here on *linear question networks*, where the target is a linear combination of the subsequent target and observation, $z_t = Po_{t+1} + Qz_{t+1}$. Linear question networks satisfy a *Bellman network equation*,

$$v^*(h_t) = \mathbb{E}[z_t \mid h_t] = \mathbb{E}[Po_{t+1} + Qz_{t+1} \mid h_t] = \mathbb{E}[Po_{t+1} + Qv^*(h_{t+1}) \mid h_t] \tag{6}$$

The simplest predictions are grounded directly in observations, for example the target might be the observation at the next time-step, $v_1^*(h) = \mathbb{E}[o_{t+1} \mid h_t = h]$. A compositional prediction might be the expected value at time-step $t+1$ of the expected observation at time-step $t+2$, $v_2^*(h) = \mathbb{E}[v_1^*(h_{t+1}) \mid h_t = h]$. Compositional questions can also be recursive, so that questions can be asked about temporally distant observations. For example, a value function can be represented by a prediction of the form $v_3^*(h) = \mathbb{E}[o_t + v_3^*(h_{t+1}) \mid h_t = h]$, where o_t can be viewed as a reward, and v_3^* is the value function for this reward.

The answer network is a non-linear representation of state containing n state variables. Each state variable $v_i(h)$ represents the estimated answer to the ith prediction. The answers are updated incrementally by combining the state $v(h_{t-1})$ at the last time-step – the previous answers – with the new observation o_t. In the original TD network paper, the answers were represented by a recurrent neural network, although other architectures are possible. In this case, $v(h_t, \theta) = \sigma(Ao_t + Bv(h_{t-1}))$ where $\sigma(x)$ is a non-linear activation function with derivative $\sigma'(x)$; $\theta = [A, B]$ is an $n \times (m+n)$ weight matrix; and $v(h_0) := 0$ by definition.

The key idea of temporal-difference networks is to train the parameters of the answer network, so that the estimated answers approximate the true answers as closely as possible. One way to measure the quality of our answers is using the Bellman network equation, i.e. to seek answers $v(h, \theta)$ that (approximately) satisfy this equation. One approach to solving the Bellman network equation is by *bootstrapping*, i.e. by using the right hand side of the equation, $Po_{t+1} + Qv(h_{t+1}, \theta)$, as a surrogate for the true value function $v^*(h_t)$. The parameters of the answer network can then be adjusted online by gradient descent, so as to

minimise the mean squared error, $MSE(\theta) = \mathbb{E}\left[\sum_{i=1}^{n} \left(v_i^*(h_t) - v_i(h_t,\theta)\right)^2\right]$,

$$\begin{aligned} -\frac{1}{2}\nabla_\theta MSE(\theta) &= \mathbb{E}\left[\sum_{i=1}^{n} \left(v_i^*(h_t) - v_i(h_t,\theta)\right) \nabla_\theta v_i(h_t,\theta)\right] \\ &= \mathbb{E}\left[\phi(h_t,\theta)\left(v^*(h_t) - v(h_t,\theta)\right)\right] \end{aligned} \tag{7}$$

where $\phi(h,\theta)$ is a matrix combining the gradients for each answer,

$$\phi(h,\theta) = [\nabla_\theta v_1(h,\theta), ..., \nabla_\theta v_n(h,\theta)] \tag{8}$$

Sampling the gradient gives a stochastic gradient descent algorithm,

$$\begin{aligned} \Delta\theta &= \alpha\phi(h_t,\theta)\left(v^*(h_t) - v(h_t,\theta)\right) \\ &\approx \alpha\phi(h_t,\theta)\left(Po_{t+1} + Qv(h_{t+1},\theta) - v(h_t,\theta)\right) \\ &= \alpha\phi(h_t,\theta)\delta(h_{t+1},\theta) \end{aligned} \tag{9}$$

where α is a scalar step-size; and $\delta(h_{t+1},\theta)$ is the *TD error*,

$$\delta(h_{t+1},\theta) := Po_{t+1} + Qv(h_{t+1},\theta) - v(h_t,\theta) \tag{10}$$

However, Sutton et al. did not in fact follow the full recurrent gradient, but rather a computationally expedient one-step approximation to this gradient,

$$\begin{aligned} &\nabla_\theta v_i(h_t) = \nabla_\theta \sigma(A_{i,:}o(h_t) + B_{i,:}v(h_{t-1}))_i \\ &\frac{\partial v_i(h_t)}{\partial A_{j,k}} \approx \begin{cases} o_k(h_t)\sigma'(v_i(h_t)) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \qquad \Delta A_{i,k} = \alpha\delta_i(h_{t+1},\theta)o_k(h_t)\sigma'(v_i(h_t)) \\ &\frac{\partial v_i(h_t)}{\partial B_{j,k}} \approx \begin{cases} v_k(h_{t-1})\sigma'(v_i(h_t)) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \qquad \Delta B_{i,k} = \alpha\delta_i(h_{t+1},\theta)v_k(h_{t-1})\sigma'(v_i(h_t)) \end{aligned} \tag{11}$$

Like simple recurrent networks (Elman, 1991), this *simple TD network* learning rule only considers direct connections from previous state variables $v(h_{t-1})$ to current state variables $v(h_t)$, ignoring indirect contributions from previous state variables $v(h_1), ..., v(h_{t-2})$.

Simple temporal-difference networks have been extended to incorporate histories (Tanner and Sutton, 2005a); to learn over longer time-scales using a TD(λ)-like algorithm (Tanner and Sutton, 2005b); to use action-conditional or option-conditional questions (Sutton et al., 2005); to automatically discover question networks (Makino and Takagi, 2008) and to use hidden nodes (Makino, 2009). The latter (Makino, 2009) uses BPTT to incorporate indirect contributions from previous state variables, but still does not follow the true gradient.

Unfortunately, it is well known that the simple TD network learning rule may lead to divergence, even in seemingly trivial problems such as a 6-state "cycle world" (Tanner, 2005). This is because each node in a TD network may perform up to three different roles. First, a node may act as state, i.e. the state variable $v_i(h_{t-1})$ at the previous time-step $t-1$ is used to construct answers $v_j(h_t)$ at the current time-step t. Second, a node may

act as an answer, i.e. $v_j(h_t)$ provides the current answer for the jth prediction at time t. Third, a node may act as a target, i.e. $v_k(h_{t+1})$ may be predicted by a predecessor answer $v_j(h_t)$. The simple TD network learning rule only follows the gradient with respect to the first of these three roles, and is therefore not a true gradient descent algorithm. Specifically, if an answer is selfishly updated to reduce its error with respect to its own question, this will also change the state representation, which could result in greater overall error. It will also change the TD-targets, which could again result in greater overall error. This last issue is well-known in the context of value function learning, causing TD learning to diverge when using a non-linear value function approximator (Tsitsiklis and Van Roy, 1997). In the remainder of this paper, we develop a sound TD network learning rule that correctly takes account of all three roles.

4. Bellman Network Equation

The standard Bellman equation provides a recursive one-step relationship for one single prediction: the value function. The main contribution of this paper is to generalise the Bellman equation to linear question networks (see equation 6). We can then find answer network parameters θ that solve, as effectively as possible, the Bellman network equation. We now make this notion precise.

In our Markov chain, there is one "state" for each distinct history. The Bellman network equation provides a recursive relationship between the answers $v_i(h)$ for every prediction i and every history h. To represent all of these values, we define an *answer matrix* $V^\theta_{h,i} = v_i(h, \theta)$, which is the $\infty \times n$ matrix of all answers given parameters $\theta = [A, B]$.

Each prediction is weighted by a non-negative *prediction weight* c_i indicating the relative importance of question i. Each history h is weighted by its probability $d(h)$. We define a weighted squared norm $||.||_{c,d}$ that is weighted both by the prediction weight c_i and by the history distribution $d(h)$,

$$||V||^2_{c,d} = \sum_{h \in \mathcal{H}} d(h) \sum_{i=1}^{n} c_i V^2_{h,i} \tag{12}$$

We define $\Pi(V)$ to be the non-linear projection function that finds the closest answer matrix V^θ to the given matrix V, using the weighted squared norm $||.||^2_{c,d}$,

$$\Pi(V) = \operatorname*{argmin}_\theta ||V^\theta - V||^2_{c,d} \tag{13}$$

The question network represents the relationship between true answers. However, the true expectations will not typically be representable by any parameter vector θ, and therefore we must project them back into the representable space of answer networks,

$$V^\theta = \Pi(T(OP^\top + V^\theta Q^\top)) \tag{14}$$

where O is the $\infty \times m$ matrix of last observations, $O_{h_t,i} = o_t$; P and Q are the parameters of the question network; and T is the history transition matrix. In practice, the non-linear projection Π is intractable to compute. We therefore use a local linear approximation Π_θ

that finds the closest point in the tangent space to the current answer matrix V^θ. The tangent space is represented by an $\infty \times (m+n)n$ matrix Φ_θ of partial derivatives. Each row of the tangent space corresponds to a particular combination of history h and prediction i, with a row index that we denote by $h \circ i$. Each column j corresponds to a parameter θ_j of the answer network, $(\Phi_\theta)_{h\circ i,j} = \frac{\partial V^\theta_{h,i}}{\partial \theta_j}$. The linear projection operator Π_θ projects an answer matrix V onto tangent space Φ_θ,

$$\Pi_\theta \overline{V} = \Phi_\theta (\Phi_\theta^\top G \Phi_\theta)^{-1} \Phi_\theta^\top G \overline{V} \tag{15}$$

where the operator $\overline{V}$ represents a reshaping of an $\infty \times n$ matrix V into a $\infty \times 1$ column vector with one row for each combined history and prediction $h \circ i$. G is a diagonal matrix that gives the combined history distribution and prediction weight for each history and prediction $h \circ i$, $G_{h\circ i, h\circ i} = c_i d(h)$. This linear projection leads to the *projected Bellman network equation*,

$$\overline{V^\theta} = \Pi_\theta \overline{T(OP^\top + V^\theta Q^\top)} \tag{16}$$

5. Gradient Temporal-Difference Network Learning Rule

Our approach to solving the linear TD network equation is based on the nonlinear gradient TD learning algorithm of Maei *et al.* (2009). Like this prior work, we solve our Bellman-like equation by minimising a mean squared projected error by gradient descent. However, we must extend Maei *et al.* in three ways. First, TD networks include *multiple* value functions, and therefore we generalise the mean-squared error to use the projection operator Π_θ defined in the previous section. Second, each answer in a TD networks may depend on multiple interrelated targets, whereas TD learning only considers a single TD target $r_t + \gamma V^\theta_{s_{t+1}}$. Third, we must generalise from a finite state space to an infinite space of histories. We proceed by defining the *mean squared projected Bellman network error* (MSPBNE), $J(\theta)$, for answer network parameters A. This objective measures the difference between the answer matrix and the expected TD-targets at the next time-step, projected onto the tangent space,

$$\begin{aligned} J(\theta) &= ||\Pi_\theta \overline{T(OP^\top + V^\theta Q^\top) - V^\theta}||^2_G = ||\Pi_\theta \Delta_\theta||^2_G = \Delta_\theta^\top \Pi_\theta^\top G \Pi_\theta \Delta_\theta = \Delta_\theta^\top G^\top \Pi_\theta \Delta_\theta \\ &= \Delta_\theta^\top G^\top \Phi_\theta (\Phi_\theta^\top G \Phi_\theta)^{-1} \Phi_\theta^\top G \Delta_\theta = \left(\Phi_\theta^\top G \Delta_\theta\right)^\top \left(\Phi_\theta^\top G \Phi_\theta\right)^{-1} \left(\Phi_\theta^\top G \Delta_\theta\right) \end{aligned} \tag{17}$$

where $\Delta_\theta := \overline{T(OP^\top + V^\theta Q^\top) - V^\theta}$ is the *TD network error*, a column vector giving the expected one-step error for each combined history and prediction $h \circ i$. Each of these three terms can be rewritten as an expectation over histories,

$$\begin{aligned} \Phi_\theta^\top G \Phi_\theta &= \sum_{h\in\mathcal{G}} \sum_{i=1}^{n} \phi_i(h,\theta) c_i d(h) \phi_i(h,\theta)^\top \\ &= \mathbb{E}\left[\phi(h,\theta) C \phi(h,\theta)^\top\right] \end{aligned} \tag{18}$$

$$\begin{aligned} \Phi_\theta^\top G \Delta_\theta &= \Phi_\theta^\top G \left(\overline{T(OP^\top + V^\theta Q^\top) - V^\theta}\right) \\ &= \sum_{h\in\mathcal{G}} \sum_{i=1}^{n} \phi_i(h,\theta) c_i d(h) \left(\sum_{h'\in\mathcal{G}} T_{h,h'} \delta_i(h',\theta)\right) \\ &= \mathbb{E}\left[\phi(h,\theta) C \delta(h',\theta)\right] \end{aligned} \tag{19}$$

where C is the diagonal prediction weight matrix $C_{ii} = c_i$. The MSPBNE can now be written as a product of expectations,

$$w_\theta = \mathbb{E}\left[\phi(h,\theta)C\phi(h,\theta)^\top\right]^{-1}\mathbb{E}\left[\phi(h,\theta)C\delta(h',\theta)\right]$$
$$J(\theta) = \mathbb{E}\left[\phi(h,\theta)C\delta(h',\theta)\right]^\top w_\theta \tag{20}$$

The *GTD network* learning rule updates the parameters by gradient descent, so as to minimise the MSPBNE. In the appendix we derive the gradient of the MSPBNE,

$$\psi(\theta, w_\theta) = \sum_{i=1}^{n} c_i \left(\delta_i(h',\theta) - \phi_i(h,\theta)^\top w_\theta\right)\nabla^2_\theta V_{h,i} w_\theta \tag{21}$$
$$-\frac{1}{2}\nabla_\theta J(\theta) = \mathbb{E}\left[-\nabla_\theta\delta(h',\theta)C\phi(h,\theta)^\top w_\theta - \psi(\theta, w_\theta)\right] \tag{22}$$

From Equation 10 we see the derivative of the TD error is a linear combination of gradients,

$$-\nabla_\theta\delta(h',\theta) = \phi(h,\theta) - \phi(h',\theta)Q^\top \tag{23}$$

As in GTD2, we separate the algorithm into an online linear estimator for $w \approx w_\theta$, and an online stochastic gradient descent update that minimises $J(\theta)$ assuming that $w = w_\theta$,

$$\Delta\theta = \alpha\left[\left(\phi(h,\theta) - \phi(h',\theta)Q^\top\right)C(\phi(h,\theta)^\top w) - \psi(\theta, w)\right] \tag{24}$$
$$\Delta w = \beta\phi(h,\theta)C(\delta(h',\theta) - \phi(h,\theta)^\top w) \tag{25}$$

This *recurrent GTD network* learning rule is very similar to the GTD2 update rule for non-linear function approximators (Equations 1 to 3), where states have been replaced by histories, and where we sum the GTD2 updates over all predictions i, weighted by the prediction weight c_i.

We can also rearrange the gradient of the MSPBNE into an alternative form,

$$\begin{aligned}-\frac{1}{2}\nabla_\theta J(\theta) &= \mathbb{E}\left[\phi(h,\theta)C\phi(h,\theta)^\top w_\theta - \phi(h',\theta)Q^\top C\phi(h,\theta)^\top w_\theta - \psi(\theta, w_\theta)\right] \\ &= \mathbb{E}\left[\phi(h,\theta)C\delta(h',\theta) - \phi(h',\theta)Q^\top C\phi(h,\theta)^\top w_\theta - \psi(\theta, w_\theta)\right]\end{aligned} \tag{26}$$

This gives an alternative update for θ, which we call the *recurrent TDC network* learning rule, by analogy to the non-linear TDC update (Maei et al., 2009).

$$\Delta\theta = \alpha\left[\phi(h,\theta)C\delta(h',\theta) - \phi(h',\theta)Q^\top C(\phi(h,\theta)^\top w) - \psi(\theta, w)\right] \tag{27}$$

The first term is identical to recurrent TD network learning when $c_i = 1, \forall i$ (see equation 9); the remaining two terms provide a correction that ensures the true gradient is followed.

Finally, we note that the components of the gradient matrix $\phi(h,\theta)$ are *sensitivites* of the form $\frac{\partial v_i(h)}{\partial A_{j,k}}$ and $\frac{\partial v_i(h)}{\partial B_{j,k}}$, which can be calculated online using the real-time recurrent learning algorithm (RTRL) (Williams and Zipser, 1989); or offline by backpropagation through time (BPTT) (Rumelhart et al., 1986). Furthermore, the Hessian-vector product required by

$\psi(\theta, w)$ can be calculated efficiently by Pearlmutter's method (Pearlmutter, 1994). As a result, if RTRL is used, then the computational complexity of the recurrent GTD (or TDC) network learning is equal to the RTRL algorithm, *i.e.* $O(n^2(m+n)^2)$ per time-step. If BTPP is used, then the computational complexity of recurrent GTD (or TDC) network learning is $O(T(m+n)n^2)$ over T time-steps, which is n times greater than BPTT because the gradient must be computed separately for every prediction. In practice, automatic differentiation (AD) provides a convenient method for tracking the first and second derivatives of the state representation. Forward accumulation AD performs a similar computation, online, to real-time recurrent learning (Williams and Zipser, 1989); whereas reverse accumulation AD performs a similar computation, offline, to backpropagation through time (Werbos, 2006).

6. Outline Proof of Convergence

We now outline a proof of convergence for the GTD network learning rule, closely following Theorem 2 of (Maei et al., 2009). For convergence analysis, we augment the above GTD network learning algorithm with an additional step to project the parameters back into a compact set $C \subset \mathbb{R}^{(m+n)n}$ with a smooth boundary, using a projection $\Gamma(\theta)$. Let K be the set of asymptotically stable equilibria of the ODE (17) in (Maei et al., 2009), i.e. the local minima of $J(\theta)$ modified by Γ.

Theorem 1 *Let $(h_k, h'_k)_{k\geq 0}$ be a sequence of transitions drawn i.i.d. from the history distribution, $h_k \sim d(h)$, and transition dynamics, $h'_k \sim P_{h,\cdot} \mid h_k$. Consider the augmented GTD network learning algorithm, with a positive sequence of step-sizes α_k and β_k such that $\sum_{k=0}^{\infty} \alpha_k = \infty, \sum_{k=0}^{\infty} \beta_k = \infty, \sum_{k=0}^{\infty} \alpha_k^2 < \infty, \sum_{k=0}^{\infty} \beta_k^2 < \infty$ and $\frac{\alpha_k}{\beta_k} \to 0$ as $k \to \infty$. Assume that for each $\theta \in C$, $\mathbb{E}(\sum_{i=1}^{n} \phi_i(h,\theta)\phi_i(h,\theta)^\top)$ is nonsingular. Then $\theta_k \to K$, with probability 1, as $k \to \infty$.*

The proof of this theorem follows closely along the lines of Theorem 2 in (Maei et al., 2009), using martingale difference sequences based on the recurrent GTD network learning rule, rather than the GTD2 learning rule. Since the answer network is a three times differentiable function approximator, the conditions of the proof are met.[1]

7. Network Architecture

A wide variety of state representations based on recurrent neural network architectures have been considered in the past. However, these architectures have enforced constraints on the roles that each node can perform, at least in part to avoid problems during learning. For example, simple recurrent (SR) networks (Elman, 1991) have two types of nodes: *hidden* nodes are purely state variables, and *output* nodes are purely answers; no nodes are allowed to be targets. TD networks (Sutton and Tanner, 2004) have one type of node, representing both state variables and answers; no nodes are allowed to be purely a state variable (hidden) or purely an answer (output). Like SR networks, SR-TD networks (Makino, 2009) have both hidden nodes and output nodes, but only hidden nodes are allowed to be targets.

1. We have omitted some additional technical details due to using an infinite state space.

Clearly there are a large number of network architectures which do not follow any of these constraints, but which may potentially have desirable properties.

By defining a sound learning rule for arbitrary question networks, we remove these constraints from previous architectures. Recurrent GTD networks can be applied to any architecture in which nodes take on any or all of the three roles: state, answer and target. To make node i a hidden node, the corresponding prediction weight c_i is set to zero. To make node i an output node, the outgoing edges of the answer network $A_{i,j}$ should be set to zero for all j. Recurrent GTD networks also enable many intermediate variants, not corresponding to prior architectures, to be explored.

8. Empirical Results

We consider a simple *cycle world*, corresponding to a cyclic 6-state Markov chain with observation $o = 1$ generated in state s_1 and observation $o = 0$ generated in the five other states. We used a simple depth 6 "chain" question network containing M-step predictions of the observation: $v_1^*(h_t) = \mathbb{E}\left[o_{t+1} \mid h_t\right]$, $v_{i+1}^*(h_t) = \mathbb{E}\left[v_i^*(h_{t+1}) \mid h_t\right]$ for $i \in [1, 6]$; and a fully connected recurrent answer network with logistic activation functions. This domain has been cited as a counter-example to TD network learning, due to divergence issues in previous experiments (Tanner, 2005).

We used forward accumulation AD (Rump, 1999) to estimate the sensitivity of the answers to weight parameters. We used a grid search to identify the best constant values for α and β. Weights in $\theta = [A, B]$ were initialised to small random values, and w was initialised to zero. On each domain, we compared the simple TD network learning rule (Equation 11) (Sutton and Tanner, 2004), with the recurrent GTD and TDC network learning rule (Equations 25). 10 learning curves were generated over 50,000 steps, starting from random initial weights; performance was measured by the mean squared observation error (MSOE) in predicting observations at the next time-step. The results are shown in Figure 2. The simple TD network learning rule performs well initially, but is attracted into an unstable divergence pattern after around 25,000 steps; whereas GTD and TDC network learning converge robustly on all runs.

9. Conclusion

TD networks are a promising approach to representation learning that combines ideas from both predictive representations and recurrent neural networks. We have presented the first convergent learning rule for TD networks. This should enable the full promise of TD networks to be realised. Furthermore, a sound learning rule enables a wide variety of new architectures to be considered, spanning the full spectrum between recurrent neural networks and predictive state representations.

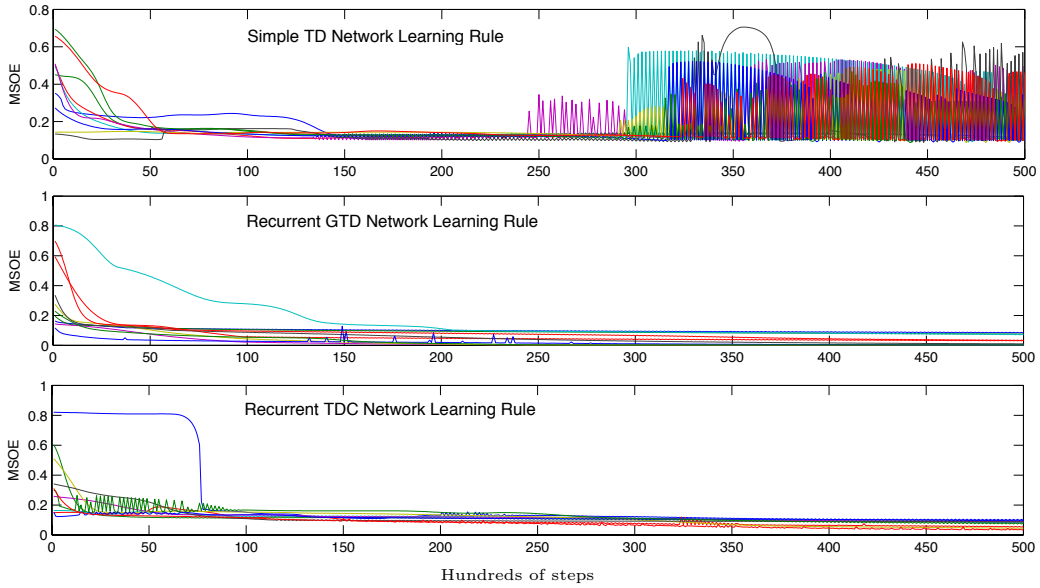

Figure 2: Empirical convergence of TD network learning rules on a 6 state cycle world. 10 learning curves are shown for each algorithm, starting from random weights.

References

Jeffrey L. Elman. Distributed representations, simple recurrent networks and grammatical structure. *Machine Learning*, 7:195–225, 1991.

Hamid Maei, Csaba Szepesvari, Shalabh Bhatnagar, Doina Precup, David Silver, and Rich Sutton. Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation. In *Advances in Neural Information Processing Systems 22*, pages 1204–1212, 2009.

Takaki Makino. Proto-predictive representation of states with simple recurrent temporal-difference networks. In *26th International Conference on Machine Learning*, page 88, 2009.

Takaki Makino and Toshihisa Takagi. On-line discovery of temporal-difference networks. In *25th International Conference on Machine Learning*, pages 632–639, 2008.

Barak A. Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 6(1): 147–160, 1994.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*, pages 318–362. MIT Press, 1986.

Siegfried Rump. INTLAB - INTerval LABoratory. In *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, 1999.

Satinder P. Singh, Michael R. James, and Matthew R. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *20th Conference in Uncertainty in Artificial Intelligence*, pages 512–518, 2004.

R. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(9):9–44, 1988.

Richard S. Sutton and Brian Tanner. Temporal-difference networks. In *Advances in Neural Information Processing Systems 17*, 2004.

Richard S. Sutton, Eddie J. Rafols, and Anna Koop. Temporal abstraction in temporal-difference networks. In *Advances in Neural Information Processing Systems 18*, 2005.

Richard S. Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *26th Annual International Conference on Machine Learning*, page 125, 2009.

Brian Tanner. Temporal-difference networks. Master's thesis, University of Alberta, 2005.

Brian Tanner and Richard S. Sutton. Temporal-difference networks with history. In *19th International Joint Conference on Artificial Intelligence*, pages 865–870, 2005a.

Brian Tanner and Richard S. Sutton. Td(lambda) networks: temporal-difference networks with eligibility traces. In *22nd International Conference on Machine Learning*, pages 888–895, 2005b.

J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674690, 1997.

Paul Werbos. Backwards differentiation in ad and neural nets: Past links and new opportunities. In *Automatic Differentiation: Applications, Theory, and Implementations*, volume 50, pages 15–34. Springer Berlin Heidelberg, 2006.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Appendix A. Gradient of the MSPBNE

The gradient of the MSPBNE can be derived by an extension of Maei *et al.* (2009), making an equivalent assumption that $\mathbb{E}[P]\,\phi(h)C\phi(h)^\top$ is non-singular in a neighbourhood around A. We recall the definition of the MSPBNE $J\theta$), and formulate the derivative in terms of the term w_θ,

$$
\begin{aligned}
w_\theta &= \mathbb{E}\left[P\right]\phi(h)C\phi(h)^\top{}^{-1}\mathbb{E}\left[P\right]\phi(h)C\delta(h',\theta) \\
J(\theta) &= \mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right] \\
-\frac{1}{2}\nabla_\theta J(\theta) &= -\nabla_\theta \mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right] \\
&\quad -\frac{1}{2}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top \nabla_\theta \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right] \\
&= -\nabla_\theta \mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right] \\
&\quad +\frac{1}{2}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\nabla_\theta \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]\mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right] \\
&= -\nabla_\theta \mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top w_\theta + \frac{1}{2}w_\theta^\top \nabla_\theta \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]w_\theta \\
&= -\nabla_\theta \mathbb{E}\left[\sum_{i=1}^n c_i\phi_i(h,\theta)\delta_i(h')\right]^\top w_\theta + \frac{1}{2}w_\theta^\top \nabla_\theta \mathbb{E}\left[\sum_{i=1}^n c_i\phi_i(h,\theta)\phi_i(h,\theta)^\top\right]w_\theta \\
&= -\mathbb{E}\left[\sum_{i=1}^n c_i\phi_i(h,\theta)\nabla_\theta\delta_i(h')\right]^\top w_\theta - \mathbb{E}\left[\sum_{i=1}^n c_i\nabla_\theta\phi_i(h,\theta)\delta_i(h')\right]^\top w_\theta + \mathbb{E}\left[\sum_{i=1}^n c_i w_\theta^\top\phi_i(h,\theta)\nabla_\theta\phi_i(h,\theta)^\top w_\theta\right] \\
&= -\mathbb{E}\left[\nabla_\theta\delta(h',\theta)C\phi_i(h,\theta)^\top w_\theta\right] - \mathbb{E}\left[\sum_{i=1}^n c_i\nabla_\theta^2(V(h)w_\theta)\left(\delta_i(h') - \phi_i(h,\theta)^\top w_\theta\right)\right] \\
&= -\mathbb{E}\left[\nabla_\theta\delta(h',\theta)C\phi_i(h,\theta)^\top w_\theta + \psi(\theta,w_\theta)\right]
\end{aligned}
$$

where $\psi(\theta,w_\theta) = \sum_{i=1}^n c_i\nabla_\theta^2(V(h)w_\theta)\left(\delta_i(h',\theta) - \phi_i(h,\theta)^\top w_\theta\right)$