Proto-Predictive Representation of States with Simple Recurrent Temporal-Difference Networks[†]

Takaki Makino MAK@SCINT.DPC.U-TOKYO.AC.JP
Division of Project Coordinate, Tokyo University, 5-1-5 Kashiwa-no-ha, Kashiwa-shi, Chiba 277-8568 Japan

Abstract

We propose a new neural network architecture, called Simple Recurrent Temporal-Difference Networks (SR-TDNs), that learns to predict future observations in partially observable environments. SR-TDNs incorporate the structure of simple recurrent neural networks (SRNs) into temporal-difference (TD) networks to use proto-predictive representation of states. Although they deviate from the principle of predictive representations to ground state representations on observations, they follow the same learning strategy as TD networks, i.e., applying TD-learning to general predictions. Simulation experiments revealed that SR-TDNs can correctly represent states with an incomplete set of core tests (question networks), and consequently, SR-TDNs have better on-line learning capacity than TD networks in various environments.

1. Introduction

Predictive representations (Littman et al., 2002; Jaeger, 2000) are a relatively new group of approaches to expressing and learning grounded knowledge about partially-observable dynamical systems. These approaches represent the state of a dynamical system as a vector of predictions, based on the basic principle that important knowledge about the world can be represented strictly in terms of the relationships between predictions of observable quantities. In the *predictive state representations* (PSRs) introduced by Littman et al. (2002), each prediction is an estimate of the probability of *tests*, defined as some sequence of observations given a sequence of actions.

Temporal-Difference (TD) networks (Sutton & Tanner, 2005) were one of the first proposals for an on-line learning algorithm of predictive representations. The basic idea was to apply TD learning in reinforcement learning to general predictions, i.e., train each prediction targeting at the value of another prediction or observation at a later time. The targeting relations between predictions are given as a *question network*, and another *answer network* is trained like an artificial neural network. TD learning was also incorporated into PSRs and confirmed to work (Wolfe et al., 2005).

These approaches, however, share the same restrictions derived from the basic principle of predictive representation, i.e., an appropriate question network, or set of tests, must be given to the learning agents. A sufficient set of tests, namely *core tests*, depends on the underlying dynamics in the environments, which is hidden for a learning agent; if an insufficient set of tests is given, the agent fails to represent states, let alone make predictions. Several methods for automatically discovering the test set have been proposed (James & Singh, 2004; McCracken & Bowling, 2006; Makino & Takagi, 2008), but they are based on incremental searches in the space of test sets. They are not guaranteed to reach the sufficient test set, especially when the dynamics have deep temporal dependencies.

Our approach deviates slightly from the basic principle, and incorporates ideas from studies on connectionism (artificial neural networks). In the long history of connectionism, various network architectures have been proposed for predicting temporal sequences. One famous architecture is the simple recurrent networks (SRNs) (Elman, 1995). Although state representations in SRNs (*context layers*) are not grounded on observations, it is known that SRNs are capable of predicting more complex temporal sequences than the networks with only grounded state representations (Jordan, 1986). In particular, it has been shown (Elman, 1995; Rodriguez et al., 1999) that SRNs are capable of predicting temporal sequences belonging to context-free grammars,

[†]We made a preliminary report of this work as an extended abstract for a local conference (Makino, 2008), but the reported results were much worse due to handicapped settings. Change of the setting in this paper (SRNs and SR-TDNs to use feature vectors) significantly improved performance of the proposed models, and lead us to substantially different discussion and conclusion.

Appearing in *Proceedings of the 26^{th} International Conference on Machine Learning*, Montreal, Canada, 2009.

which is a superclass of regular expressions.

In this paper, we propose a new network architecture, named simple recurrent TD networks (SR-TDNs), which is a fusion of SRNs and TD networks. SR-TDNs use the state representation from SRNs and training strategy from TD networks, i.e., context layers and TD learning. Since the context layers in SR-TDNs contain the prototypic information prior to the output prediction, we call this a *proto-predictive representation* of states. Proto-predictive representation has an advantage over predictive representation because the former can correctly represent states with an incomplete question network.

2. Background

First, we briefly review neural networks and IO-HMMs for showing basic notations and the learning algorithm we use.

2.1. Neural Networks

A non-recurrent, single-layer feed-forward neural network produces an output vector $\boldsymbol{x}^O \in \mathbb{R}^{N_O}$ of N_O units, given an input vector $\boldsymbol{x}^I \in \mathbb{R}^{N_I}$ of N_I units in the following rule:

$$\boldsymbol{z}^O = W \cdot \boldsymbol{x}^I + \boldsymbol{\theta} \qquad \boldsymbol{x}^O = g(\boldsymbol{z}^O) \quad , \qquad (1)$$

where $\boldsymbol{z}^O$ is the vector of the weighted sum of inputs for each output unit, W is the connection weight matrix, $\boldsymbol{\theta}$ is the bias vector, and g is an output function.

We train the network by changing W and $\boldsymbol{\theta}$ so that the input-output relation of the network approaches to the given pairs of inputs and target outputs. In our setting, a target vector $\boldsymbol{\tau}^O$ that corresponds to an input $\boldsymbol{x}^I$ may be partially valid for units that satisfy conditions associated with them. We give a condition vector $\boldsymbol{c}^O \in \{0,1\}^{N_O}$ along $\boldsymbol{\tau}^O$ to calculate error $\boldsymbol{e}^O$ only for selected output units:

$$e^O_j = c^O_j \cdot (\tau^O_j - x^O_j) \quad (j = 1, \ldots, N_O) \quad . \qquad (2)$$

The gradient descent method for minimizing the squared sum of error $(\boldsymbol{e}^O)^2$ yields the following learning rule:

$$\Delta w_{ji} = -\eta \cdot e^O_j \cdot x^I_i \cdot \frac{\partial x^O_j}{\partial z^O_j} \quad \Delta\theta_j = -\eta \cdot e^O_j \cdot \frac{\partial x^O_j}{\partial z^O_j} \ , \ (3)$$

where $i = 1, \ldots, N_I$, $j = 1, \ldots, N_O$, and η is the learning factor. When g is the sigmoid function $\sigma(z) = 1/(1+e^{-z})$, we can use relation $\partial x^O_j / \partial z^O_j = x^O_j(1 - x^O_j)$.

A multi-layer feedforward neural network can be considered as a chain of single-layer neural networks, which transfers the output of a network to the input of the next network in the chain. The backpropagation algorithm (Rumelhart & McClelland, 1986) can be represented as calculating the error of the input vector:

$$e^I_i = c^I_i \cdot \sum_j \left[w_{ji} \cdot e^O_j \cdot \frac{\partial x^O_j}{\partial z^O_j} \right] \quad . \qquad (4)$$

The input error vector can then be transferred back to the output error vector of the previous network in the chain. Using this algorithm repeatedly, we can train every connection weight in a multi-layer neural network.

A neural network is called *recurrent* if it takes input from the network state at previous time steps. For example,

$$\boldsymbol{x}^O[t] = g(W^{OI} \cdot \boldsymbol{x}^I[t] + W^{OO}\boldsymbol{x}^O[t-1] + \boldsymbol{\theta}^O) \quad . \quad (5)$$

where t denotes the time. Given $\boldsymbol{x}^O[0]$ and $\boldsymbol{x}^I[t]$ ($t = 1, \ldots, T$), the calculation process for $\boldsymbol{x}^O[T]$ in the recurrent network can be unfolded into a T-layer feed-forward neural network. A training method known as backpropagation through time (BPTT) (Zipser, 1990) is equivalent to applying the backpropagation algorithm to this unfolded feed-forward network. To reduce the cost of calculation, we truncate the unfolded network in terms of T_{back}, i.e., the time steps going back; in other words, the errors backpropagate up to $\boldsymbol{x}^O[T - T_{\mathrm{back}}]$, and no further than that.

2.2. Input-Output Hidden Markov Models

Input-output Hidden markov models (IO-HMMs) (Bengio & Frasconi, 1995) is a Hidden markov model with input, which is similar to partially-observed Markov decision processes (POMDPs) except that IO-HMMs has no decisions (actions are given externally, and rewards are unused).

We denote an IO-HMM as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, P \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A} = \{\alpha_1, \ldots, \alpha_{N_A}\}$ is the input (action) space, $\boldsymbol{o} \in \mathcal{O} \subseteq \{0,1\}^{N_O}$ is the N_O-bit observation vector, and $P(s, a, \boldsymbol{o}, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \times \mathcal{S} \to [0, 1]$ is the probability of observing $\boldsymbol{o}$ and changing to state s' at the next time step given the current state s and action a. An agent is required to learn the prediction, $T(\boldsymbol{o}|a, t) = Pr(\boldsymbol{o}[t+1] = \boldsymbol{o} \,|\, a[t+1] = a, \boldsymbol{o}[t], a[t], \boldsymbol{o}[t-1], a[t-1], \cdots)$, i.e., conditional probability of future observations given a past sequence of observations and inputs, without knowing $P(s, a, \boldsymbol{o}, s')$.

Generally, even with a full knowledge of $P(s, a, \boldsymbol{o}, s')$, one cannot determine the current state, $s[t]$, from past observations and inputs. The best reasoning using $P(s, a, \boldsymbol{o}, s')$ is to maintain a belief state, $\boldsymbol{b}[t]$, whose element $b_s[t]$ specifies the conditional probability of the agent at time t being in state s.

$$\begin{aligned} b_{s'}[t+1] &= Pr(s'|\boldsymbol{o}[t+1], a[t+1], \boldsymbol{b}[t]) \\ &= \frac{\sum_{s \in \mathcal{S}} b_s[t] P(s, a[t+1], \boldsymbol{o}[t+1], s')}{T^*(\boldsymbol{o}[t+1]|a[t+1], t)} \ , \end{aligned} \quad (6)$$

where $T^*(\boldsymbol{o}|a, t)$ is an oracle, or the theoretically best prediction obtained by learning:

$$T^*(\boldsymbol{o}|a, t) = \sum_{s, s' \in \mathcal{S}} b_s[t] P(s, a, \boldsymbol{o}, s') \quad . \qquad (7)$$

In the following, the learning algorithms are measured in terms of mean squared error from the oracle $T^*(\boldsymbol{o}|a,t)$ for a given time window, L:

$$\mathrm{MSE}[t_0] = \frac{1}{L}\sum_{t=t_0-L}^{t_0-1}\sum_{\boldsymbol{o}\in\mathcal{O}}\Big[T(\boldsymbol{o}|a[t+1],t) - T^*(\boldsymbol{o}|a[t+1],t)\Big]^2 . \quad (8)$$

3. Network Architectures

In this section, we give the definitions of the network architectures. First, we explain two existing network architectures, i.e., simple recurrent networks (SRN) with inputs, and TD networks. After that, we propose a new network architecture, simple recurrent TD networks (SR-TDNs).

Figure 1 illustrates these network architectures with an emphasis on their similarities. The main differences lie in the source of state information (the bottom left of each subfigure) and the target of learning (the top of each subfigure).

3.1. Simple Recurrent Networks with Inputs

Simple recurrent networks (SRN) (Elman, 1995) are designed to predict observations in hidden Markov models, that is equivalent to IO-HMMs with only one input. We made a straightforward extension to SRNs so that it can predict IO-HMMs. Formally,

$$\begin{aligned}\boldsymbol{x}^I[t] &= f(\boldsymbol{x}^H[t-1], \boldsymbol{o}[t], a[t]) \quad ,\\ \boldsymbol{x}^H[t] &= \sigma(W^{HI}\cdot\boldsymbol{x}^I[t] + \boldsymbol{\theta}^H) \quad , \text{ and}\\ \boldsymbol{x}^O[t] &= \sigma(W^{OH}\cdot\boldsymbol{x}^H[t] + \boldsymbol{\theta}^O) \quad , \end{aligned} \quad (9)$$

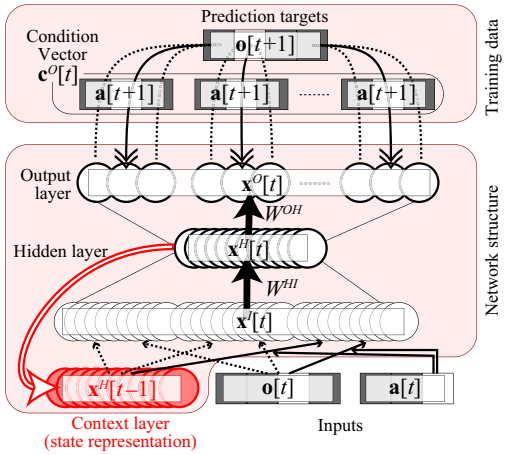
where $\boldsymbol{x}^I$ is N_I-length vector of input layer, $\boldsymbol{x}^H$ is N_H-length vector of hidden layer, $\boldsymbol{x}^O$ is the vector of output layer, σ is the sigmoid function, $W, \boldsymbol{\theta}$ are the weight matrix and bias vector that are modified through learning, and f is the feature vector function that constructs network input from external inputs and the state of the network, i.e., the information carried from the previous time step. In SRNs, the value of the hidden layer $\boldsymbol{x}^H[t-1]$ is used as the state; in SRNs, this state representation is called as a *context layer*.

Although feature vector is not used in the original proposal of SRN, in this study we use the feature vector function that is used in some previous studies of TD networks (Tanner & Sutton, 2005b; Tanner & Sutton, 2005a) for fair comparison between architectures:

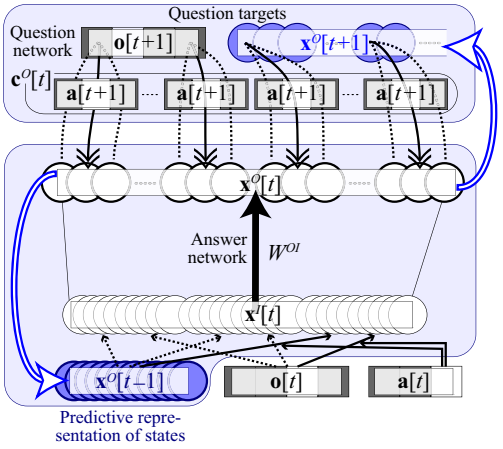
$$\begin{aligned} f(\boldsymbol{x},\boldsymbol{o},a) = (&a_1o_1, a_2o_1, \ldots, a_{N_A}o_1, a_1o_2, \ldots, a_{N_A}o_{N_O},\\ &a_1x_1, \ldots, a_{N_A}x_1, a_1x_2, \ldots, a_{N_A}x_N)^T , \end{aligned} \quad (10)$$

where N is the length of $\boldsymbol{x}$ and $\boldsymbol{a} = (a_1, \ldots, a_{N_A})^T$ is the N_A-bit vector representation of input a:

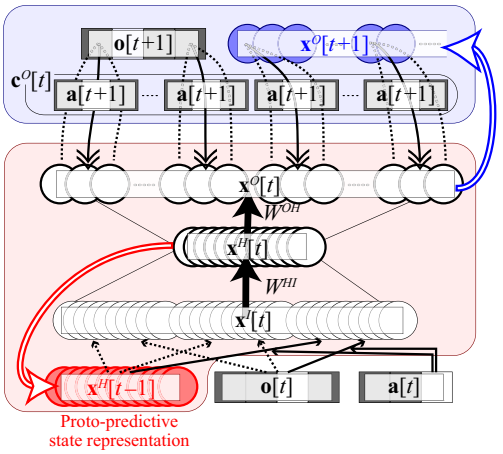
$$a_i = \begin{cases} 1 & a = \alpha_i \\ 0 & \text{otherwise} \end{cases} . \quad (11)$$



(a) SRN with Inputs



(b) TD network



(c) SR-TDN

Figure 1. Network architectures. Circles: neural units, thick black arrows: network connections, white arrows: copying over adjacent time steps, double-headed arrows: feedback from training data, dotted lines: feedforward/feedback relations whose condition is not satisfied.

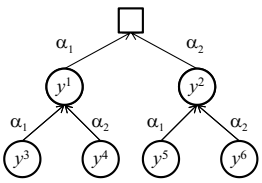


Figure 2. Example of TD network we focused on in this paper. Square denotes observation node.

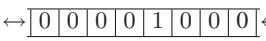


Figure 3. 8-state ring world. Both ends are connected. 0 and 1 denote observation from the places. Agent can move left and right.

Figure 4. 10-state Bomb Ring world. 1 denotes working bomb, $\overline{1}$ denotes faulty bomb. An agent can move left, move right, and kick. Observation consists of two bits, bomb and flash. All bombs seem identical. If the agent kicks a working bomb and takes any two actions, the agent observes a flash regardless of the agent's position, and the bomb is reset.

The target of the output $\boldsymbol{o}[t+1]$ is conditioned by each possible input at $t+1$. In training, the output units corresponding to the actual input $a[t+1]$ are trained with the result of observation $\boldsymbol{o}[t+1]$. Formally,

$$\langle \boldsymbol{\tau}^O[t], \boldsymbol{c}^O[t] \rangle = \left\langle \begin{pmatrix} o_1[t+1] \\ \vdots \\ o_1[t+1] \\ o_2[t+1] \\ \vdots \\ o_{N_O}[t+1] \end{pmatrix}, \begin{pmatrix} a_1[t+1] \\ \vdots \\ a_{N_A}[t+1] \\ a_1[t+1] \\ \vdots \\ a_{N_A}[t+1] \end{pmatrix} \right\rangle . \quad (12)$$

In case of $N_A = 1$, this network is equivalent to the original SRN. Predictions about future observations can be obtained from the output units of the network, i.e., $T(o_j|a_i,t) \propto x^O_{N_A(j-1)+i}[t]$. Other architectures explained below are also designed so that the same equation gives the prediction of the network.

3.2. TD Networks

The purpose of TD networks (Sutton & Tanner, 2005) is to learn the prediction of future observations obtained from the environment. A TD network consists of a set of nodes and links, and each node represents a single scalar prediction. A node has one or more links directed to other nodes or observations from the environment, which denotes the targets of the node's prediction. A link may have a condition, which indicates that the node is a conditional prediction of the target. This set of nodes and links is called a *question network* since each node represents some question about the environment.

As in the previous studies (Tanner & Sutton, 2005b), we have focused on a subset of TD networks, in which every node has a single target (the parent node), every link is conditioned with an action (input), and there are no loops in the question network. Figure 2 is an example of such a TD network for a scalar observation. Node y_1 predicts observation at the next step if action α_1 is taken. Node y_4 predicts the value of node y_1 at the next step if action α_2 is taken; consequently, y_4 gives the prediction for observation after two steps of actions, $\alpha_2\alpha_1$. In the following, we have denoted $p(y_i) \in \{o_1, \cdots, o_{N_O}, y_1, \cdots, y_{i-1}\}$ as the parent node of y_i, and $a(y_i)$ as the condition for the link between y_i and $p(y_i)$ ($a(y_i) = 1$ if the condition is met, otherwise 0).

To provide answers to the questions asked by the question network, each node in a TD network also works as a function approximator. The inputs to the function approximator of a node are defined by *answer network*, taking values from other nodes, available observations, and the next input. These function approximators are trained so that the nodes output the answers to the question asked by the question network.

The answer networks can be represented as the following neural network (Fig. 1b):

$$\begin{aligned} \boldsymbol{x}^I[t] &= f(\boldsymbol{x}^O[t-1], o[t], a[t]) \quad , \quad \text{and} \\ \boldsymbol{x}^O[t] &= \sigma(W^{OI} \cdot \boldsymbol{x}^I[t] + \boldsymbol{\theta}^O) \quad , \end{aligned} \quad (13)$$

where f is given by Eq. (10). Since the state of the network is the result of prediction at the previous time step, we say that the TD network uses a predictive state representation. This network can make an accurate prediction only if possible belief states in the environments can be distinguished by the representation; in other words, the set of questions must be sufficient to represent the state.

The training data and condition vector are as follows:

$$\langle \boldsymbol{\tau}^O[t], \boldsymbol{c}^O[t] \rangle = \left\langle \begin{pmatrix} p(y_1)[t+1] \\ \vdots \\ p(y_n)[t+1] \end{pmatrix}, \begin{pmatrix} a(y_1)[t+1] \\ \vdots \\ a(y_n)[t+1] \end{pmatrix} \right\rangle \quad (14)$$

where

$$p(y_i)[t] = \begin{cases} o_j[t] & \text{if } p(y_i) = o_j \ (j \in \{1, \ldots, N_O\}) \\ x^O_k[t] & \text{if } p(y_i) = y_k \ (k \in \{1, \ldots, i-1\}) \end{cases}. \quad (15)$$

In the experiments, we use question networks that are mechanically generated as follows. The first $N_O N_A$ nodes were targeted to predict every observation bit for every possible action. Then, for each prediction node sequentially from y_1, N_A child nodes are assigned, and conditioned by each possible action. This assignment is repeated until the number of nodes reaches the given limit N_P.

3.3. Simple Recurrent TD Networks

We propose a *simple recurrent TD network* (SR-TDN), which has the structure of an SRN with the learning strat-

egy of TD networks. The core idea underlying TD networks is to train a network to predict its own future output, in addition to the observation, through temporal differences. Although TD networks use $\boldsymbol{x}^O[t-1]$, the output prediction at time $t-1$, as the state representation at time t, it is reasonable to replace the state representation with $\boldsymbol{x}^H[t-1]$, i.e., the values of the hidden units at time $t-1$, because $\boldsymbol{x}^H[t-1]$ represents a prototypic data prior to $\boldsymbol{x}^O[t-1]$. Note that this replacement causes a slight deviation from the principle of predictive representation that uses state representation grounded on the observable quantities. To make distinction, we say that SR-TDNs use *proto-predictive representations* of states.

This idea can be implemented by incorporating the idea of TD networks into an SRN (Fig. 1c). Formally, an SR-TDN has the same connectivity as an SRN (Eq. 9). Input and training data for the SR-TDN is the same as a TD network, i.e., Eq. (10) as input, and Eq. (14) as the training data.

Unlike predictive state representation, proto-predictive representations are capable of carrying redundant information in addition to prediction for the given question network. The answer network can utilize this redundancy for representing prediction beyond the question network, especially when it uses temporal learning algorithms such as BPTT. Consequently, it is possible that proto-predictive representations learn to distinguish hidden states of IO-HMMs correctly, even if only an insufficient question network is given. In the next section, we empirically investigate the expressive power of proto-predictive representations.

4. Experiments

We conducted a series of simulation experiments in various partially-observable environments. Figures 3 and 4 show the 8-state ring world (Tanner & Sutton, 2005a) and the Bomb-ring world, respectively. All other environments are taken from the POMDP problem repository (Cassandra, 1999) and converted to IO-HMMs by stripping off the rewards. Table 1 summarizes the environments. For all environments, 30 sequences of 2×10^6 observations and inputs (actions) were generated with uniform random policy, and all network architectures were trained on this set of sequences. Before the beginning of each sequence, every network was initialized with a random connection. The mean squared error of prediction from the oracle was measured (Eq. 8) with $L = 10^4$, and averaged over 30 sequences.

In all experiments, we set N_P = N_H = 40. We applied BPTT to TD networks[1] even though no previous study had done so, because we wanted to focus on differences between network architectures. The parameter T_{back}, the number of backpropagating steps in BPTT, was set to 3. The learning rate η is initialized with 0.1, and after 1,000,000 steps, η is halved for every 150,000 steps.

[1]BPTT on TD networks can be easily derived by unfolding Eq. (13), like unfolding Eq. (5), and applying a standard backpropagation (Eq. 4) to it.

Table 1. Summary of test environments

	N_A	N_O	$\lvert\mathcal{S}\rvert$
(a) Tiger	3	1	5
(b) Network	4	1	7
(c) Paint	4	1	4
(d) Shuttle	3	5	8
(e) Bridge Repair	12	5	5
(f) Cheese	4	7	11
(g) 8-state Ring	2	1	8
(h) Bomb Ring	3	2	29

Figure 5 plots the results of experiments. In addition to SRN with feature vector input, we also tested "SRN w/o fv", SRN with conventional input (which is just a concatenation of $\boldsymbol{x}$, $\boldsymbol{o}$ and $\boldsymbol{a}$). We can see that all the network architectures made good predictions in simple environments (a–c). As the environments gets complex (d–h), the predictions by TD networks became less accurate. The reason may be that the given question network was insufficient for correct representation of the states of these environments. This is likely because a question network is mechanically generated for every possible pair of observations and inputs up to a specified number of units ($N_P = 40$), and these environments have a relatively large number of observations and inputs. In particular, TD networks failed to learn Bomb-ring environment (h), because the agent cannot distinguish the faulty bomb from working bombs.

However, using the same insufficient question networks, the SR-TDNs successfully learned accurate predictions in these environments. This improvement is not due to the extra layer added to the SR-TDNs, because adding a hidden layer to TD networks did not improve learning (plotted as "TD network w/ hid"). These results demonstrate high expressive power of proto-predictive state representations, which can distinguish hidden states even when the predicted answers for the states are identical.

At the same time, we observed a slight instability of SR-TDNs. For example, in 1 of 30 trials for learning Bomb Ring environment (h), an SR-TDN had slipped back to erroneous prediction for a while (showed as a convex in the result). Such instability may be caused by the deviation from the principle of predictive representation, that is, to ground the state representation on observable quantities. However, we argue that the instability of SR-TDNs is less serious than that of SRNs. Context layers in SRNs tend to accumulate errors, especially when the dependency to the observation is long; we can see that SRNs show spiky results in 8-state Ring world (h). On the other hand, SR-TDNs are smoother in the same setting, suggesting that the SR-TDNs can learn better state representations through predicting answers for the question network.

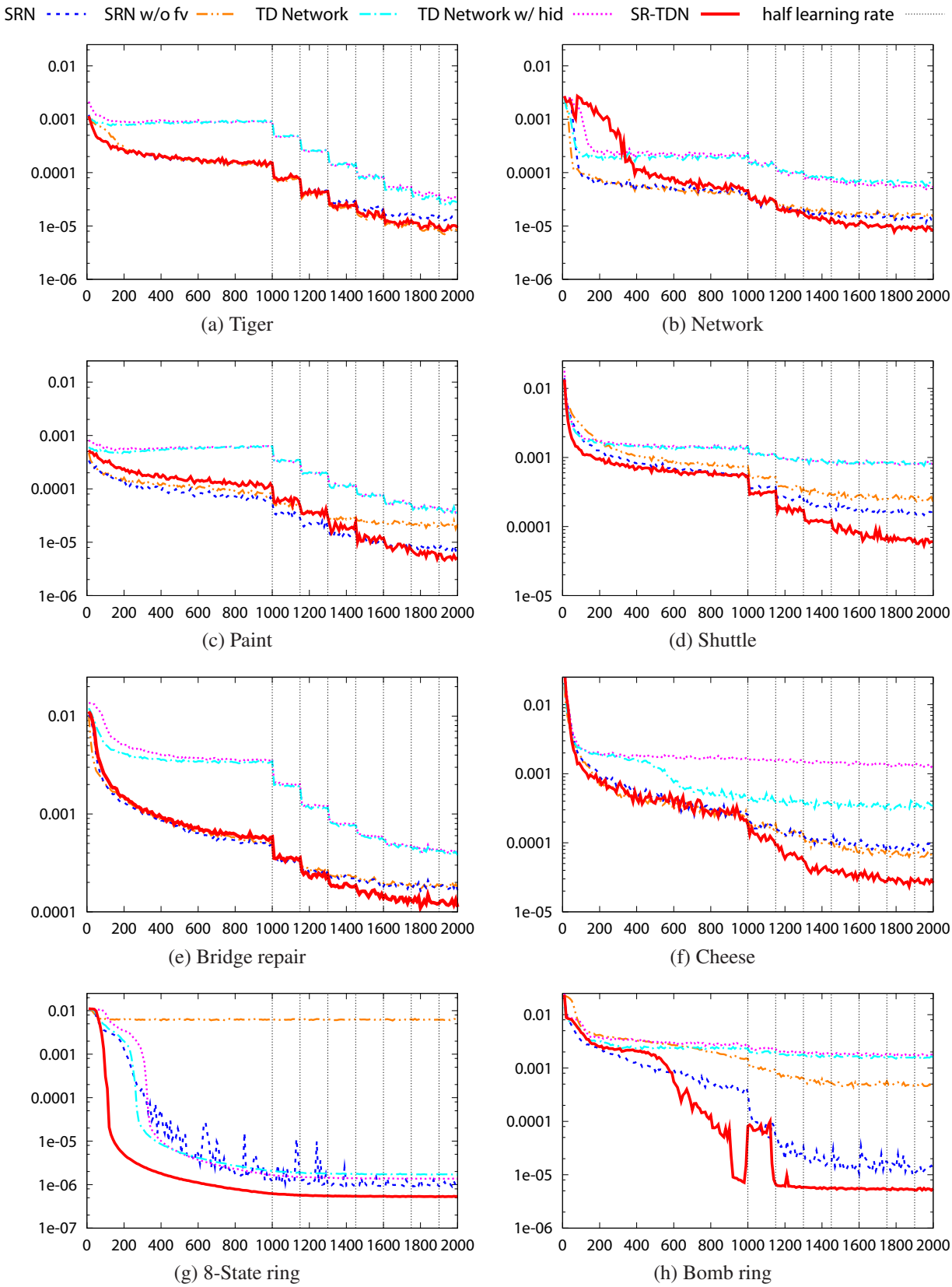


Figure 5. Test results. X-axis: simulation time ($\times 10^3$), Y-axis: mean squared error. Dotted vertical lines correspond to points when the learning rate η is halved.

5. Discussion

5.1. SR-TDNs as an extension of SRNs

So far, we presented SR-TDNs from the viewpoint of TD networks. However, we can also regard SR-TDNs as extension of SRNs that uses more complex learning signal derived from TD networks. To our knowledge, no previous study on SRNs has used such a kind of learning signal, i.e., the output of the network itself at the next time step. Here we discuss the advantage of SR-TDNs over SRNs.

As shown in the previous section, SR-TDNs showed better learning performance compared to SRNs in environments with deep temporal dependencies. It has been observed that SRNs (without BPTT) require exponentially long learning time for long temporal dependencies with indistinguishable intermediate sequences (Servan-Schreiber et al., 1988, p.28). The learning signal in SR-TDNs can bootstrap learning by producing some information in intermediate steps, which effectively reduces the temporal gap between dependencies that must be discovered by the hidden layer representation. We claim that, in this point, SR-TDNs are an important contribution to the research of SRNs.

5.2. Eligibility Traces and BPTT

Tanner and Sutton (2005a) proposed a TD(λ) network, which is an application of eligibility traces to a TD network. This may seem similar to the backpropagation through time (BPTT), i.e., the learning technique for recurrent neural networks. However, there is an important difference between them: Eligibility traces backpropagate through the question network, while BPTT backpropagates through the answer network. A combination of eligibility traces and BPTT would provide a more powerful learning algorithm for TD networks and SR-TDNs.

5.3. Possibility of Hybrid Architectures

One future direction for research is to investigate hybrid network architectures. Because of the flexible learning capacity of the neural network architecture, it is easy to combine several network architectures.

For example, a hybrid of a TD network and an SR-TDN would have a state representation consisting of two parts; one (the grounded representation) is taken from the output layer and the other (the intermediate representation) from the hidden layer. Another possibility is to combine an SR-TDN with history-based technique for TD network (Tanner & Sutton, 2005b). Although we didn't use history in our experiments, it is likely that the history-based technique helps SR-TDNs as it helped TD networks.

There are many other possibilities for combining architectures. If we explore and evaluate them, we may find a network architecture with more stable learning and deep context information without hand-tuned question networks.

5.4. Combining TD Networks and Connectionism

This paper is the first approach to explore the similarity between TD networks and connectionisms. It seems that the first proposal of TD networks (Sutton & Tanner, 2005) has been developed based on an idea of connectionisms, but nobody has explored this further.

Our claim is that the history of connectionism studies contains many helpful ideas for improving TD networks. In fact, we incorporated the structure of SRNs backpropagation through time (BPTT) into TD networks. There are many other studies in connectionism, which are likely to assist TD networks, such as real-time recurrent learning (RTRL) (Williams & Zipser, 1989) (or, more generally, an extended Kalman filter; Haykin, 1998, 762ff). Such a line of research would also be helpful for other approaches pertaining to predictive representations, such as PSRs. As the idea of TD learning has been incorporated into PSRs (Wolfe et al., 2005), many new research opportunities should emerge by importing ideas from connectionism through TD networks into predictive representations.

Moreover, similar ideas to some PSR studies can be found in connectionism studies. For example, PSRs with memory (James & Singh, 2005) and TD networks with history (Tanner & Sutton, 2005b) can be viewed as a kind of time-delay neural networks (Waibel et al., 1989) with recurrent connection (Ma, 2004). As another, more important example, transformed PSRs (Rosencrantz et al., 2004) can be regarded as homologous research into SR-TDNs within PSRs; the differences are that (1) SR-TDNs use sigmoid functions, which allow to represent a more powerful, nonlinear transformation of prediction, (2) learning with BPTT allows SR-TDNs to represent predictions beyond the given questions, and (3) SR-TDNs support on-line learning as TD networks do.

6. Conclusion

We investigated a new network architecture for learning IO-HMMs, that are derived from TD networks and SRNs. SR-TDNs are a combination of the learning strategy of TD networks and the state representation of SRNs. Through simulation experiments with limited number of nodes, we showed that the learning capacity of SR-TDNs is superior to that of SRNs and TD networks. This is because SR-TDNs use *proto-predictive representation of states*, which can represent information beyond the limit of the given question network. This paper also contributed to positioning TD networks within studies on artificial neural networks, opening the way to apply various techniques stud-

ied in connectionism to predictive representations. Future work includes importing other techniques from connectionism studies as well as exploring new hybrid architectures.

ACKNOWLEDGEMENTS

I'd like to thank Prof. Toshihisa Takagi for his kind support. This work was supported by KAKENHI (20700126).

References

Bengio, Y., & Frasconi, P. (1995). An input-output HMM architecture. In *Advances in neural information processing systems 7*, 427–434. Cambridge, MA: MIT Press.

Cassandra, A. (1999). Tony's POMDP file repository page. URL http://www.cs.brown.edu/research/ai/pomdp/examples/index.html.

Elman, J. L. (1995). Language as a dynamical system. In R. F. Port and van T. Gelder (Eds.), *Mind as motion: Explorations in the dynamics of cognition*, 195–223. Cambridge, MA: MIT Press.

Haykin, S. (1998). *Neural networks - a comprehensive foundation (2nd. ed.)*. Upper Saddle River, NJ.: Prentice-Hall.

Jaeger, H. (2000). Observable operator models for discrete stochastic time series. *Neural Computation*, *12*, 1371–1398.

James, M. R., & Singh, S. (2004). Learning and discovery of predictive state representations in dynamical systems with reset. *Proc. of the 21st Intl. Conf. on Machine learning* (p. 53). New York: ACM Press.

James, M. R., & Singh, S. (2005). Planning in models that combine memory with predictive representations of state. *Proc. of the 20th National Conf. on Artificial Intelligence* (pp. 987–992).

Jordan, M. (1986). *Serial order: A parallel distributed processing approach* (Technical Report ICS Report 8604). Institute for Cognitive Science, UCSD, La Jolla, CA.

Littman, M. L., Sutton, R. S., & Singh, S. (2002). Predictive representations of state. In *Advances in neural information processing systems 14*, 1555–1561. Cambridge, MA: MIT Press.

Ma, J. (2004). The capacity of time-delay recurrent neural network for storing spatio-temporal sequences. *Neurocomputing*, *62*, 19–37.

Makino, T. (2008). Automatic acquisition of TD-network in POMDP environments: Extension with SRN structure. *Proc. of The 22nd Annual Conf. of the Japanese Society for Artificial Intelligence*. (3A2-2).

Makino, T., & Takagi, T. (2008). On-line discovery of temporal-difference networks. *Proc. of the 25th Intl. Conf. on Machine Learning* (pp. 632–639). Helsinki, Finland: Omnipress.

McCracken, P., & Bowling, M. (2006). Online discovery and learning of predictive state representations. In *Advances in neural information processing systems 18*, 875–882. Cambridge, MA: MIT Press.

Rodriguez, P., Wiles, J., & Elman, J. L. (1999). A recurrent neural network that learns to count. *Connection Science*, *11*, 5–40.

Rosencrantz, M., Gordon, G., & Thrun, S. (2004). Learning low dimensional predictive representations. *Proc. of the 21st Intl. Conf. on Machine Learning* (pp. 88–95). New York: ACM Press.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing*, vol. 1. MIT Press.

Servan-Schreiber, D., Cleeremans, A., & McClelland, J. (1988). *Encoding sequential structure in simple recurrent networks* (Technical Report CMU-CS-88-183). School of Computer Science, Carnegie Mellon University, Pittsburg, PA.

Sutton, R. S., & Tanner, B. (2005). Temporal-difference networks. In *Advances in neural information processing systems 17*, 1377–1384. Cambridge, MA: MIT Press.

Tanner, B., & Sutton, R. S. (2005a). TD(λ) networks: temporal-difference networks with eligibility traces. *Proc. of the 22nd Intl. Conf. on Machine Learning* (pp. 888–895). New York: ACM Press.

Tanner, B., & Sutton, R. S. (2005b). Temporal-difference networks with history. *Proc. of the 19th Intl. Joint Conf. on Artificial Intelligence* (pp. 865–870).

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, Signal Processing*, *37*, 328–339.

Williams, R., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*, 270–280.

Wolfe, B., James, M. R., & Singh, S. (2005). Learning predictive state representations in dynamical systems without reset. *Proc. of the 22nd Intl. Conf. on Machine Learning* (pp. 980–987). New York: ACM Press.

Zipser, D. (1990). Subgrouping reduces complexity and speeds up learning in recurrent networks. In *Advances in neural information processing systems 2*, 638–641. Morgan Kaufmann.