Journal of Computer Science 10 (11): 2211-2219, 2014
ISSN: 1549-3636

doi:10.3844/jcssp.2014.2211.2219 Published Online 10 (11) 2014 (http://www.thescipub.com/jcs.toc)

INCORPORATING PRIOR KNOWLEDGE INTO TEMPORAL DIFFERENCE NETWORKS

Britton Wolfe and James Harpe

Department of Computer Science, Indiana University-Purdue University Fort Wayne (IPFW), Fort Wayne, IN, USA

Received 2014-06-06; Revised 2014-06-26; Accepted 2014-11-05

ABSTRACT

Developing general purpose algorithms for learning an accurate model of dynamical systems from example traces of the system is still a challenging research problem. Predictive State Representation (PSR) models represent the state of a dynamical system as a set of predictions about future events. Our work focuses on improving Temporal Difference Networks (TD Nets), a general class of predictive state models. We adapt the internal structure of the TD Net and we present an improved algorithm for learning a TD Net model from experience in the environment. The new algorithm accepts a set of known facts about the environment and uses those facts to accelerate the learning. These facts can come from another learning algorithm (as in this study) or from a designer's prior knowledge about the environment. Experiments demonstrate that using the new structure and learning algorithm improves the accuracy of the TD Net models. When tested in an in finite environment, our new algorithm outperforms all of the standard PSR learning algorithms.

Keywords: Predictive State, Temporal Difference, Modeling, Dynamical Systems

1. INTRODUCTION

This study addresses the problem of learning a model of a discrete-time dynamical system from a sequence of experience in the system. Such systems have long been of interest to reinforcement learning researchers (Sutton and Tanner, 2004; Sutton and Barto 1998; Dahmani and Benyettou, 2005) but still pose great challenges for learning and modeling.

Our work applies generally to a broad class of dynamical systems. In such systems, at every time step t = 1, 2, 3,..., the agent takes some action a^t and receives some observation o^t from the system. We restrict our attention to systems with a discrete set of possible actions and a discrete set of possible observations.

1.1. Models and State Representations

A model of a system predicts the likelihood of future observations given a sequence of actions that the agent might take in the future. These predictions are also conditioned upon the agent's experience in the system $a^1o^1a^2o^2 \ldots a^\tau o^\tau$ through the current time τ. This experience is called history. Although predictions depend upon history, a model cannot simply store history, because it will grow without bound as the agent continues to act in the world. Instead, a model maintains a summary of the current history called state. History changes after each time step, so the model's state also needs to change after each time step. The structure and parameters of a model determine the state update mechanism of the model: How the model computes the state at time t + 1 from the state at time t and the most recent action a^{t+1} and observation o^{t+1}.

For a given system, there are many possible ways to summarize history, which lead to different state representations. In general, state is simply a vector of numbers, but the meaning or semantics of those numbers differs among state representations. Models such as

Corresponding Author: Britton Wolfe, Department of Computer Science, Indiana University-Purdue University Fort Wayne (IPFW), Fort Wayne, IN, USA

Hidden Markov Models (HMMs) or Partially Observable Markov Decision Processes (POMDPs) represent state as a distribution over unobserved (latent) system states. This distribution is called the belief state. In contrast, Predictive State Representations (PSRs) represent state as a set of predictions about future events. For example, one element in a PSR's state might be the probability of seeing a particular observation if the agent takes a particular action.

PSRs are capable of representing partially observable, stochastic dynamical systems, including any system that can be modeled by a finite POMDP (Singh *et al*., 2004). There is evidence that predictive state is useful for generalization (Rafols *et al*., 2005) and helps to learn more accurate models than the state representation of a POMDP (Wolfe *et al*., 2005).

There are two main categories of predictive state models: Temporal Difference Networks (TDNets) (Sutton and Tanner, 2004; Tanner and Sutton, 2005a) and linear predictive models, which include linear PSRs (Littman *et al*., 2001) and transformed PSRs (Rosencrantz *et al*., 2004). The primary difference between the categories is their different state update mechanisms. Linear predictive models use a specific functional form that involves linear functions of the state vector. In contrast, TDNets do not specify a particular functional form for updating state (see Section 1.2 for details).

This study focuses on improving both the TDNets' design and the algorithms to learn TDNet models from training *data*. These improvements are motivated by the goal of easily incorporating knowledge of specific facts into the TDNet. As an example, consider the fact "If the most recent observation was blue, then the next observation will be blue if the agent takes the 'do nothing' action." This fact corresponds to a prediction that something will happen (in certain situations) with probability 1.0.

To incorporate this fact into a linear predictive model, we would need to alter its parameters so that the model will make the prediction of 1.0 in the appropriate situations. However, altering the model's parameters ends up changing all of the model's predictions, due to the particular state update mechanism of a linear predictive model. In contrast, Section 2 describes how one can easily incorporate specific facts into a TDNet model. In this study, those facts are themselves automatically learned from the training data by a separate algorithm, but incorporating someone's prior knowledge about the environment would work in the same way.

1.2. TDNets

There is a good deal of flexibility regarding what comprises a TDNet. The essential components are (1) the semantics (i.e., definition) of each element in the state vector and (2) the state update mechanism.

1.2.1. Defining State

Each element in the state vector of a TDNet is defined in terms of actions and observations, either past or future (or both). In order to make this more precise, we use some terminology from the predictive state literature. A history is a possible sequence of actions and observations $a^1o^1a^2o^2 \ldots a^\tau o^\tau$ from the beginning of time through the current time τ. A *test* is a sequence of possible future actions and observations $a_1o_1 \ldots a_ko_k$. The *prediction* for a test $t = a_1o_1 \ldots a_ko_k$ from a history $h = a^1o^1 \ldots a^\tau o^\tau$ is defined as the probability of seeing the observations of t when the actions of t are taken from history *h*. Formally, this prediction is:

$$p(t|h) \overset{def}{=} \prod_{i=\tau+1}^{\tau+k} Pr(o^i = o_{i-T} \,|\, a^1, o^1 \cdots a^\tau, o^\tau, a^{\tau+1} = a_1, o^{\tau+1} = o_1, \cdots a^i = a_{i-\tau})$$

The state of a TDNet typically consists of some features of a small, finite portion of history (e.g., an indicator variable that is 1 if the most recent observation was "blue" and 0 otherwise) and predictions about some tests (Tanner and Sutton, 2005b). Once the particular history features and tests are selected, the state representation of the TDNet is completely defined.

1.2.2. State Update Mechanism

The state update mechanism defines how the TDNet takes the state at time t and the most recent action/observation (at time $t + 1$) and uses them to compute the state at time $t + 1$. TDNets are not constrained to use any specific form for updating state. However, in practice, the following form is commonly used (Sutton and Tanner, 2004; Tanner and Sutton, 2005b; Sutton *et al*., 2005; Tanner and Sutton, 2005a). Let s_t be the state vector at time t and let s_{t+1}^{pred} be the part of state at time $t + 1$ that consists of predictions about tests. Let x_{t+1} be a vector of binary indicator variables for the action and observation at time $t + 1$. That vector includes a variable for every possible action, which is set to 1 if the action was taken at time $t + 1$ and 0 otherwise. This 1-of-K encoding is also used for each dimension of the observation vector at time $t + 1$. In this way, the action and observation at time $t + 1$ are encoded in x_{t+1},

which is concatenated with s_t to yield the vector s_t^+. Then $s_{t+1}^{Pred} = \sigma(Ws_t^+)$, where W is a matrix of weights and σ is the logistic function. The remainder of the state at time t + 1- the features of history-are set based upon the new history. Notice that this state update is like a neural network with logistic activation function and no hidden layers. The inputs to the network are the elements of s_t^+ and the outputs are s_{t+1}^{Pred}.

1.2.3. Learning a TDNet

Given some definition for the state of a TDNet and a functional form for updating state, "learning the model" involves estimating the parameters of the state update mechanism from training data. The training data consists of one or more sequences of experience $a^1o^1a^2o^2$ ... from the system (generated from the agent's exploration). Since the state update mechanism is supposed to compute s_{t+1}^{pred} from s_t^+, one way to learn the TDNet parameters is to solve a regression problem. That is, learn a function to map s_t^+ to s_{t+1}^{pred}. In order to learn this function, the TDNet learning algorithm uses estimated pairs $(\hat{s}_t^+, \hat{s}_{t+1}^{pred})$, one for each time step of the training data. These training examples are passed into the appropriate learning algorithm for whatever functional form is used to update state (e.g., back propagation to train a feed forward neural network). That learning algorithm yields estimates of the parameters for the state update. These are the parameters of the TDNet model.

Algorithm 1 Computing the TD Target for a Test

t = $a_1o_1 \ldots a_no_n$ in $(\hat{s}_t^+, \hat{s}_{t+1}^{pred})$ using the training data $a^{t+2}o^{t+2} \ldots$

```
for i = 1 to n do
  if a_i 6= a^{t+1+i} then
    /* Combine the test's success through step i-1
     with the TDNet's estimate that the rest of the test
     will succeed. */
     return TDNet's estimate that a_i o_i ... a_n o_n will
     succeed from time t + 1 + i
  else if oi 6= ot+1+i then
    return 0.0 /* The test failed */
  end if
end for
return 1.0 /* The test succeeded */
```

When learning a TDNet, the training examples $(\hat{s}_t^+, \hat{s}_{t+1}^{pred})$ are themselves estimated using some current parameters for the TDNet (details given below). Thus, training a TDNet is an iterative process: Initialize the TDNet parameters to some values θ_0. On each iteration $_i$, use the current parameters θ_i to compute a training example from every time step of the training data. Feed those training examples into back propagation (or other regression algorithm) to get new parameters θ_{i+1}. Repeat until the parameters converge.

1.2.4. Computing the TD Targets

The training examples for the regression algorithm are determined from the training data as follows. Each $\hat{s}_{t+1}^{Pred}$ is simply the value for the state at time t given the current TDNet parameters θ_i, concatenated with the indicators for the action and observation at time t+1. The targets $\hat{s}_{t+1}^{Pred}$ are computed using the TD(λ) algorithm (Algorithm 1)for learning TDNets with λ = 1, which is consistently the best value of λ (Tanner and Sutton, 2005a).

In expectation, the target for the prediction is equal to the true prediction when the estimates from the TDNet are accurate. Of course, the TDNet's estimates will not start out being accurate, but the idea behind the learning algorithm is that the cases where the targets are 1.0 and 0.0 (which do not rely upon TDNet estimates) should help the learning algorithm bootstrap. That is, those targets that come solely from the data should provide enough information to learn reasonable estimates for the predictions, which can then be used to iteratively improve the estimates. However, part of the motivation for our work is that these 1.0 and 0.0 targets tended to be insufficient in practice for learning a TDNet with reasonable estimates. This motivates our idea of using better targets for the learning algorithm when the values of certain predictions are known, either from another learning algorithm or from someone's knowledge of the environment (Section 2).

1.3. Simple Recurrent TD Networks

Simple recurrent TD networks (SR-TDNs) (Makino, 2009) are an adaptation of standard TDNets. Our improved TD network (Section 2) incorporates some of the concepts from SR-TDNs, which we describe in this section. SR-TDNs move beyond the standard TDNet structure (described in Section 1.2) in three ways. Firstly, the state update mechanism is a neural network with a hidden layer, which is not present in previous TDNets.

Secondly, the neural network is explicitly recurrent. Even in the original TDNet, the state update mechanism was effectively recurrent. That is, some of the inputs for computing s_{t+2}^{Pred} are the outputs s_{t+1}^{Pred} from the previous time step t + 1. However, the standard TDNet training algorithm uses simple back propagation applied to a feed

forward neural network. This treats the input/output values for one time step independently from the other time steps, not accounting for the fact that error in s_{t+2}^{Pred} could be due to error in s_{t+1}^{Pred} (the immediate inputs) or s_{t-i}^{Pred} (some predictions further back in time). In contrast, SR-TDNs are trained using Back Propagation Through Time (BPTT), which back propagates error in predictions across multiple time steps of state updates.

Thirdly, the state of an SR-TDN is not defined in terms of actions and observations, so it is not a "predictive state" model. That is, state contains neither predictions about future tests nor features of history. Instead, the SR-TDN's state consists of the activation levels of the neural network's hidden layer at the most recent time step. The authors call this a proto-predictive representation of state, because the state is used to compute predictions via the neural network's connections from the hidden layer to the output layer. The output layer is defined as predictions for a set of tests, so the SR-TDN uses the TDNet targets to train the network.

When compared with standard TDNets, the SR-TDNs are able to learn more accurate models of several simple environments when using a small set of tests in the output layer (Makino, 2009). This remains the case even when a hidden layer is added to the TDNet, indicating that simply adding a hidden layer does not significantly enhance the TDNets' accuracy.

2. IMPROVING TDNETS WITH EXTERNAL KNOWLEDGE

Our work focuses upon enhancing the structure and learning algorithm for TDNets in order to improve their accuracy. As we have noticed in our work, the current learning algorithms for TDNets are insufficient for general-purpose application. Silver (2012) also note this fact in their work with TDNets.

Our improvements are based upon the idea that there is some external (to the TDNet learning algorithm) source of information about the environment. In our experiments, we use another learning algorithm as that source of information, but it could just as well be a set of rules that someone writes down (or a combination of the two).

We assume that the external information is in the form of predictions whose values are known in particular contexts. For instance, one fact about the environment might be that $p(t|h) = 1.0$ for some particular test t when observation "red" was seen more recently than "blue." The external information is used to "override" the state values of the TDNet whenever an external fact is applicable (e.g., history satisfies some criteria). For example, the neural network part of the TDNet might estimate one of its state values $p(t|h)$ as 0.9, but the external fact says the prediction should be 1.0. In that case, the external value overrides the 0.9, so the state value would be changed to 1.0.

Thus, our model's state update mechanism proceeds in two parts. The first part uses a neural network to estimate the values for the next state. The second part involves checking the list of external facts for rules that apply to the current context. For each rule that applies, the prediction estimate from the neural network is replaced or "overridden" with the value from the rule.

We assume that the external facts are encoded in terms of predictions and history features (i.e., actions and observations) because that is a common language for describing the environment. This is important for incorporating the external information into the state of the model. For a model where state is not a set of predictions about future events, such as a POMDP or SR-TDN, the external facts cannot be directly incorporated into the model's state. This is because the meaning of the unobserved states of a POMDP or the state in an SR-TDN (i.e., the hidden layer) is determined by the learning algorithm. Therefore, the meaning of their states may vary greatly depending upon the training data. Thus, prior knowledge about an environment cannot be encoded in terms of a POMDP or SR-TDN state. However, when state is defined in terms of actions and observations, other sources of information can speak the same "language" as the state. The ability to directly incorporate small pieces of information directly into state is one advantage of predictive state models. Our work is the first to take advantage of this feature by merging external knowledge with a predictive state model.

2.1. The Model's Neural Network

The neural network in our model adopts two of the three changes to the basic TDNet that were introduced by SRTDNs: Using BPTT to train the network and using a neural network with a hidden layer. However, the third change SR-TDNs made-a recurrent connection from the hid- den layer to the input layer at the next time step-is replaced in our model by a recurrent connection from the output layer to the input layer at the next time step. Because that output layer consists of predictions about tests, this change ensures that our model's state is predictive (in contrast with SR-TDNs). Thus, our model combines the advantage of using a recurrent network (Makino, 2009) with the ability of predictive state to easily accept external facts.

To reiterate, our model is most similar to an SR-TDN. The primary differences are (1) the change in the recurrent connection's source layer (from hidden layer to output layer) and (2) the extra step in the state update process where external facts are checked and state values are overridden if applicable. An additional difference between our model and any prior TDNet (including SR-TDNs) is that we use softmax groups for the output layer instead of a logistic function. The softmax groups ensure that the predictions for all the tests with the same action sequence sum to 1.0. For example, when the TDNet is built, if a test $a_1o_1a_2o_2$ ... is in the state, then all tests with action sequence a_1a_2 . . . will also be included in the state. Those tests form a softmax group. While the softmax groups ensure that subsets of tests' predictions sum to 1.0, that may no longer be true after a value is overridden. Thus, when evaluating the model (i.e., not when learning the parameters), after all the overrides are performed, each softmax group is normalized so its predictions sum to 1.0.

2.2. Learning the Model

We used Back Propagation Through Time (BPTT) with a quadratic regularization term to learn the parameters of the neural network. The training targets for the predictions were computed using the TD(λ = 1) algorithm described in Section 1.2. Whenever a target uses an estimate from the state of the TDNet, that estimate's value includes any overriding that is done based upon the external facts (because the overriding is part of the model's state update process). Thus, the external facts provide higher quality training targets for the network learning algorithm.

The external facts also help with another aspect of learning the neural network: Assigning blame for errors. In general BPTT, the error in the output layer at time t is back propagated to all previous time steps. This is because error at time *t* could have been due to error in any of the prior layers in the unrolled network. However, our algorithm assumes that the external facts are accurate predictions. In particular, when a value at an output neuron at time t has been overridden, we assume that passing that value along as an input at time *t*+1 does not contribute to errors at time *t* + 1 or thereafter. However, prior to overriding, the network made some estimate for the overridden value at time *t*. That estimate has some error, which is back propagated from time t backwards. Specifically, the partial derivative of the error with respect to the input (i.e., weighted sum) of the overridden neuron is set to the difference between the current network's prediction and the target value (given by the external fact). This is the same derivative as if there were no future time steps after the override. In other words, any error at those future time steps is not attributed to the overridden neuron. This change to BPTT provides more information to the learning algorithm about which neural network parameters to change in order to reduce the error.

3. RESULTS

We tested our new model and learning algorithm in a simulated environment where the agent has an "eye" that can see one single pixel. The agent can move the eye around an infinite canvas that contains several shapes. The canvas is divided into grid squares, each of which contains one shape. Each grid square is several pixels wide and tall, so the agent needs several observations from within a grid square in order to determine what shape is there. Each shape can be red or blue, a square or a triangle. The shapes are on a white background, so each observation is red, blue, or white. The agent has has nine actions available to it: It can move the eye one pixel in each of the four cardinal directions (north, east, south, or west), move to the "primary point" (i.e., the top center point) of the current shape, or leap to the primary point of an adjacent shape in any of the four cardinal directions.

Because of the unbounded canvas of shapes, learning everything about the environment is impossible, but there are some things that should be easy to pick out. For example, if the agent sees red, goes east, then goes back west, it should see red again. As our results will show, this type of domain-some simple aspects embedded in a large environment-presents difficulties for other predictive state models, but our algorithm is able to learn the simple aspects of the domain without getting befuddled by the large environment.

3.1. Learning External Facts

The external information for our model was the result of applying a second learning algorithm to the training data. The algorithm generates a list of rules of the following form: If the most recent $k \leq 2$ time steps of history were *h*', then the prediction for a test t is 1.0 (or 0.0). That is, this algorithm looks for things that will deterministically happen (or not) based upon the most recent k time steps. For our experiments, we allowed *h*' to begin with either an action (e.g., *h*' = *ao*) or an observation (e.g., *h*' = *oao*). In order to find these rules, we used two simple concepts from association rule mining.

The first concept is that of minimum support. Let the sup- port for a rule [$p(t| * h') = x$] for $x \in \{0.0, 1.0\}$ be the number of occurrences in the training data where the

actions of the test *t* were executed following some history with suffix *h*'. Suppose that the test t succeeds (or fails) in every one of those instances. Then we add a rule to the list if the support for that rule is at least the minimum support. For our experiments, we set the minimum support at 10. Evaluating different values for the minimum support threshold is left for future work.

The second concept from association rule mining lets us comb through the set of candidate rules without enumeratingall of them. In particular, if some sequence h' never occurred more than c times in the training data, then no rules about *h*' will have minimum support *c*. Furthermore, any rule about any extension of h' (i.e., *oh*' or *aoh*') will also fail to have minimum support. Thus, the set of candidates *h*' of length *k* are built by taking the sequences of length *k*-1 with minimum support and extending them with each observation and (optionally) action. For each of those candidates *h*', we check every test up to length 2 to see if the rule [$p(t| * h') = x$] has minimum support.

3.2. Experimental Details

We compared our learning algorithm against algorithms for learning several other types of models: Linear PSRs, transformed PSRs, POMDPs, second-order Markov models and a few variations of TDNets (with different state update mechanisms). Each of the learning algorithms was given the same type of training data: One sequence of experience in the environment. The action-selection policy tends to explore the current shape for several steps before leaping to another shape. The exact policy is as follows: 0.04 Probability for each action that leaps to an adjacent shape or to the primary point of the current shape. If the most recent action was a single-pixel action, then there was 0.26 probability of repeating that action, with 0.18 probability for the other three single-pixel actions. Otherwise, each single-pixel action had 0.20 probability.

For the linear PSRs and TPSRs, the respective learning algorithms (Wolfe *et al*., 2005; Rosencrantz *et al*., 2004) each have a free parameter that helps determine the number of tests in the state. The parameter value was chosen automatically using a validation set. That is, 10% of the training data was held out (i.e., not used for learning the model) and the parameter value that led to the highest likelihood on the validation set was used. For POMDPs, the free parameter is the number of hidden states in the POMDP, which was selected to be 30 by a manual parameter sweep. The second-order Markov model predicts the next observation based upon the most recent 1.5 time steps of history (i.e., an observation, action, observation).

All of our TDNet variations selected the elements of the state vector in the same way. For the history features, we used binary indicator variables (i.e., 1-of-K encoding) for (1) the most recent observation, (2) the most recent action and (3) the most recent two time steps of history. The tests represented in the state included all of the length-one tests. With those in the state, the model can make any prediction (via a combination of making one-step predictions and rolling forward its state). In addition, tests of length two were randomly selected to add to the state until the total number reached a user-defined cutoff value. After each randomly selected test is added to the state, the algorithm adds additional tests as needed to ensure that two things remain true: (1) If a test is in the state, all of its suffixes are also in the state. This guarantees the existence of the TD target for each test. (2) If a test is in the state, all other tests with the same action sequence are also in the state. Thus, each test belongs to a group of tests whose predictions sum to 1.0.

The state update mechanism for each TDNet variation includes a neural network with softmax activation in the output layer. Each group of tests with the same action sequence formed one of several softmax groups in the output layer. The network had one hidden layer with a sigmoid activation function. The differences among the TDNet variations are the type of recurrence and overriding used in the network. We examined an SR-TDN; the new TDNet variation we developed in Section 2; and a variation that uses the same neural network structure as our new model but does not incorporate external information. The last variation helps to distinguish the effects of changing the neural network structure from the effects of incorporating external information.

For each TDNet variation, we set the number of tests used in the state vector, the size of the hidden layer, the regularization coefficient and the learning rate by doing a parameter sweep. Using the best parameter values, we ran another set of experiments. Those results are presented in Section 4.

3.3. Error Measures

For each type of model, we varied the amount of training data given to the learning algorithm. For each type of model and amount of training data, we learned 15 models (each one from a different set of training data), averaging the resulting error.

For each model, we evaluated the error for several different categories of predictions, each of which

corresponds to some fact about the domain that we would like a learning algorithm to discover. Here are the facts that correspond to each prediction category: (1) Jumping to the primary point of the current shape will not result in white (i.e., the probability of seeing white after jumping to the primary point of the current shape is 0.0). (2) Jumping to the primary point and going south (one pixel) will not result in white. (3) Jumping to the primary point and going north will result in white. (4) Leaping to any shape will not result in white. (5) Leaping to any shape then going south will not result in white. (6) Leaping to any shape then going north will result in white. (7) Making a single-pixel move will not result in a different (non-white) color than the last color seen. (8) Jumping to the primary point will result in the most recent (non-white) color that was seen. (9) If you just leapt some direction from a shape X to another shape Y, then if you leap in the opposite direction, the result will be the color of the shape *X*. (10) If you saw some observation *Z* and then took a single-pixel step in some direction, then if you take a single-pixel step in the opposite direction, the result will be *Z*. (11) Every blue shape has a red shape to the north.

This list covers predictions of several different forms, including predictions that are always the same (regardless of history), predictions that depend upon a finite window of history and predictions that depend upon a potentially unbounded amount of history (e.g., the last color seen). The list also measures predictions for one time step in future and predictions for further in the future. Roughly speaking, the categories of predictions get more challenging as one moves down the list.

At each time step of the testing sequence, the models' predictions were evaluated and compared against the correct predictions. The average absolute value of the error (i.e., L1 error) is shown in **Fig. 1** for the different models.

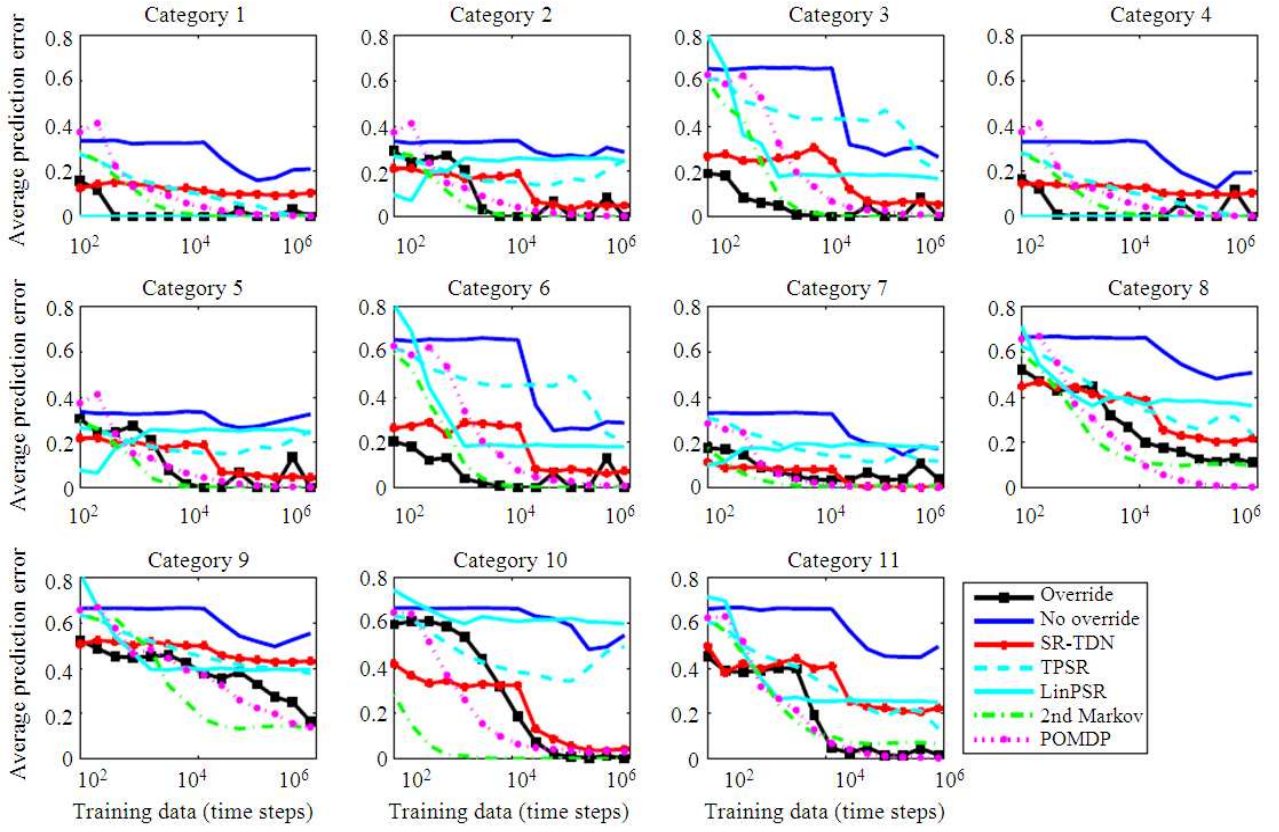

Fig. 1. Prediction error vs. amount of training data for several models: The new TDNet model ("Override"), the new TDNet model without using override information ("No Override"), SR-TDNs, TPSRs, Linear PSRs, 2nd-order Markov models and POMDPs. See Section 3.3 for details about the different error categories

4. DISCUSSION

The new TDNet model outperforms all of the other predictive state models when looking across the range of training data and the different categories. Thus, these results support the idea of incorporating external facts into the TDNet model. In particular, comparing the "Override" and "No Override" TDNet variations-where the only differenceis including or not including external facts, respectively-shows that incorporating those external facts dramatically improves the model's accuracy.

Because our primary motivation for the new TDNet model is to make accurate predictions about some aspects of a complex system, our experiments measure predictions about simple, mostly Markovian facts. Thus, we expected that the Markov-based models (2nd-order Markov Models and POMDPs) would perform very well. Indeed, **Fig. 1** illustrates that, for most categories, the Markov-based models leave little room for improvement in the accuracy as the amount of training data gets large. Thus, the primary question that these experiments aim to answer is "Did our changes to the standard TDNet model result in accuracy that is comparable to the Markov-based models?" Our results indicate that the answer is "yes." Specifically, as the amount of training data grows, the new TDNet model is comparable to the accuracy of the best models for all the categories except one (category 8).

Furthermore, for some prediction categories, the improved TDNet model performs the best over the range of training data amounts (e.g., categories 3 and 6). Overall, these experiments demonstrate that incorporating external facts into a TDNetmodel can significantly improve its accuracy, enabling it to accurately learn simple facts about an infinite, complex system. This is true even when the external facts are very simple. Using more sophisticated external facts is an interesting direction for future research, which we expect will further improve the models' accuracy.

5. CONCLUSION

We have introduced a new type of TDNet model and an algorithm for learning such a model from training data. The new model combines ideas from the original TDNets and SR-TDNs with the new idea of using an external source of information to help calculate the predictions that form the state of a predictive state model. From the original TDNets, we utilize the idea of predictive state that is updated via a neural network. From SR-TDNs, we utilize the ideas of using a hidden layer in the neural network and using backpropagation through time to learn the network parameters. However, as our experiments demonstrate (**Fig. 1**), the key to our new model's accuracy is the incorporation of external information.

The external information is used to accelerate the learning of the neural network part of the TDNet model by providing more accurate training data and guiding the backpropagation of errors across time. Furthermore, the model integrates the external information into its state update mechanism, overriding predictions' values in the state in order to improve their accuracy. Our experiments demonstrate that even when the external source of information only contains simple, history-based rules about deterministic events, that information can greatly improve the accuracy of the model. One direction for future research is to investigate the incorporation of more complicated forms of external information into a TDNet.

Our new TDNet was motivated by the goal of accurately learning simple facts about a complex system. Thus, we evaluated the models' predictions for simple, Markovian facts about an infinite system. Markovian models are tailor-made for these types of predictions and approach perfect accuracy. Nonetheless, our new TDNets' predictions are comparable to the Markovian models and are more accurate than other predictive state models (linear PSRs, TPSRs and SR-TDNs). The new TDNets even achieve near-zero error for some types of predictions, demonstrating an ability to make highly accurate predictions about simple aspects of complex systems.

6. REFERENCES

Dahmani, Y. and A. Benyettou, 2005. Seek of an optimal way by q-learning. J. Comput. Sci., 1: 28-30. DOI: 10.3844/ jcssp.2005.28.30

Littman, M.L., R. Sutton and S. Singh, 2001. Predictive representations of state. Proceedings of the Advances in Neural Information Processing Systems, (IPS' 01), MIT Press, pp: 1555-1561.

Makino, T., 2009. Proto-predictive representation of states with simple recurrent temporal-difference networks. Proceedings of the 26th International Conference on Machine Learning, Jun. 14-18, Montreal, QC, Canada, pp: 697-704. DOI: 10.1145/1553374.1553464

Rafols, E.J., M.B. Ring, R. Sutton and B. Tanner, 2005. Using predictive representations to improve generalization in reinforcement learning. Proceedings of the 19th International Joint Conference on Artificial Intelligence, (CAI' 05), Professional Book Center, pp: 835-840.

Rosencrantz, M., G. Gordon and S. Thrun, 2004. Learning low dimensional predictive representations. Proceedings of the 21st International Conference on Machine Learning, (CML' 04), ACM, pp: 88-95. DOI: 10.1145/1015330.1015441

Silver, D., 2012. Gradient temporal difference networks. J. Machine Learn. Res., 1: 1-12.

Singh, S., James, M.R. and M. Rudary, 2004. Predictive state representations: A new theory for modeling dynamical systems. Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence, (UAI' 04), AUAI Press, Virginia, pp: 512-519.

Sutton, R. and B. Tanner, 2004. Temporal-difference networks. Proceedings of the Advances in Neural Information Processing Systems, (IPS' 4), MIT Press, pp: 1377-384.

Sutton, R., E.J. Rafols and A. Koop, 2005. Temporal abstraction in temporal-difference networks. Proceedings of the Advances in Neural Information Processing Systems, (NIPS' 05), MIT Press, pp: 1313-1320.

Sutton, R.S. and A.G. Barto, 1998. Reinforcement Learning: An Introduction. 1st Edn., MIT Press, Cambridge, Mass, ISBN-10: 0262193981, pp: 322.

Tanner, B. and R. Sutton, 2005a. TD (lambda) networks: Temporal-difference networks with eligibility traces. Proceedings of the 22nd International Conference on Machine Learning, ACM, pp: 889-896. DOI: 10.1145/1102351.1102463.

Tanner, B. and R. Sutton, 2005b. Temporal-difference networks with history. Proceedings of the 19th International Joint Conference on Artificial Intelligence, Professional Book Center, pp: 865-870.

Wolfe, B., M.R. James and S. Singh, 2005. Learning predictive state representations in dynamical systems without reset. Proceedings of the 22nd International Conference on Machine Learning, ACM, pp: 985-992. DOI: 10.1145/1102351.1102475