Using Decision Trees as the Answer Networks in Temporal Difference-Networks

Laura-Andreea Antanas[1], **Kurt Driessens**[1], **Jan Ramon** [1] and **Tom Croonenborghs** [2]

1 Introduction

State representation for intelligent agents is a continuous challenge as the need for abstraction is unavoidable in large state spaces. Predictive representations offer one way to obtain state abstraction by replacing a state with a set of predictions about future interactions with the world. One such formalism is the Temporal-Difference Networks framework [2]. It splits the representation of knowledge in the question network and the answer network.

The question network defines which questions (interactions) about future experience are of interest. It contains nodes, each corresponding to a single scalar prediction about a future observation given a certain sequence of interactions with the environment. The nodes are connected by links, annotated with action-labels, which represent temporal relationships between the predictions made by the nodes, conditioned on the action-labels on the links (more details in [2]).

The answer network provides the predictive models to update the answers to the defined questions, which are expected values of the scalar quantities in the nodes. These values can be seen as estimates of probabilities. With each executed action of the agent, the predictions are updated using the answer network models to obtain a description of the new state. In classical TD-networks, logistic regression models are used, whose weight vector is obtained using a gradient learning approach.

We propose the use of probability-valued decision trees [1] in the answer network of TD-Nets. We believe that decision trees are a particular good choice to investigate, as they offer a different yet powerful form of generalization. Moreover, this aids in a better understanding of the strengths and weaknesses of TD-Nets and represents an important first step towards using them in worlds with more extensive observations. Furthermore, decision tree induction can be regarded as a prototypical example of a non-gradient learning approach.

2 Decision Trees as Answer Networks in TD-Nets

The (abstracted) state representation in a TD-Net consists of (1) the predictions made by the TD-Net in the previous timestep $\mathbf{y}_{t-1} = [y_{t-1}^{(1)}, \ldots, y_{t-1}^{(n)}]$ (one prediction for each node) with n the number of nodes in the question network, (2) the action executed during the last time frame a_{t-1} and (3) the current observation o_t, i.e., $\mathbf{x}_t = [\mathbf{y}_{t-1}, a_{t-1}, o_t]$. From this vector, the answer network will compute new predictions $\mathbf{y}_t = f(\mathbf{x}_t)$. In the original implementation of TD-Nets, this answer function is represented by a logistic regression function, i.e., $\mathbf{y}_t = f^W(\mathbf{x}_t) = \sigma(\mathbf{W}\mathbf{x}_t)$.

[1] Declarative Languages and Artificial Intelligence, Katholieke Universiteit Leuven, Leuven, Belgium, email: {laura,kurtd,janr}@cs.kuleuven.be
[2] Biosciences and Technology Department, KH Kempen University College, Geel, Belgium, email: tom.croonenborghs@khk.be

We investigate the use of probability trees [1] as an alternative to logistic regression for the answer network. The modification of the original TD-Nets framework consists solely in the introduction of a probability tree function f^T as the answer network, instead of the logistic regression function f^W. Both the semantics of TD-Nets as well as the temporal improvement learning principle, remains unchanged. The difference between the two approaches is that we choose to represent the predictive model with one tree for each node. In the original TD-nets implementation it is common to learn a set of weight vectors (aggregated in matrices), but one matrix for each <*action,observation*> combination is also practicable. In our approach, the action and observation are inputs for the probability trees which allows for more generalisation.

Generating the learning examples We build learning examples for the probability tree very similarly to the TD(1) learning scheme described in [3]. On a very high level, TD(1) generates target values for predictions looking as far into the future as possible. Figure 1 shows the timeline and dependancies between values of interest for the generation of learning examples. After the execution of an action, new predictions $\mathbf{y}_{t+1}$ for the nodes are made using the current tree function f_t^T and based on the previous predictions $\mathbf{y}_t$, the performed action a_t and the resulting observation o_{t+1}. Based on a Monte-Carlo approach, TD(1) uses these predictions and the made observations to derive target values for the old predictions of the question network, according to the structure of the network and the chosen actions.

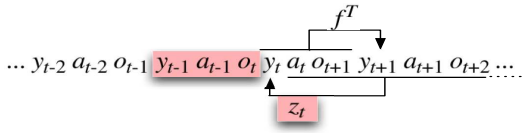

Figure 1. Dependancies between the different values used for the generation of learning examples. The structure of a learning example is shown by the shaded box, where z_t is the target.

For example, if the actions taken at time t and $t+1$ are a_t and a_{t+1} respectively, our version of TD(1) will generate the learning examples as a result of these interactions. If a node (n') in the network is conditioned by actions a_t and a_{t+1}, following this order in time, TD(1) will use the observation o_{t+2} as the target for the input vector $[\mathbf{y}_{t-1}, a_{t-1}, o_t]$. For another node (n''), conditioned only by action a_t, it will use $[\mathbf{y}_{t-1}, a_{t-1}, o_t]$ as the input vector and o_{t+1} as target. In practice, we implement this approach by using a history about chosen actions and observation with a length equal to the maximum depth of the question network.

We choose to use a TD(1) approach because it generates the most informative learning examples for the probability tree. At the start, the predictions $\mathbf{y}$ will be mostly noise. This means that both the input vector as well as the target, when not based directly on an observation, will contain noise. TD(1) will avoid the second source of noise when possible. Experiments reported in [3] show that TD(1) gives the best learning performance for a logistic regression approach too.

Incremental Tree Learning As stated before, we learn a single probability tree for each node in the question network. We will employ binary decision trees, where internal nodes in the tree test attribute-value combinations. The available decision tests are identified before the induction of the probability trees begins, using a *language bias* defined by the user of the system. It specifies the possible actions, ranges for prediction values and observations for the world that can be considered when building the tree. Since predictions from the question network nodes are needed while still learning the answer network, we need an incremental tree learning algorithm. The incremental tree induction algorithm we used is described in Algorithm 1.

Algorithm 1 Incremental Tree Induction

```
1: initialize by creating a tree with a single (empty) leaf
2: for each learning example do
3:    sort the example down the tree until it reaches a leaf
4:    update the statistics in the leaf and store the example
5:    if the statistics indicate a split is needed in the leaf and the
      number of examples in leaf > min_ex_size then
6:       generate an internal node using the indicated test
7:       grow 2 new empty leafs
8:    end if
9: end for
```

Each leaf in the tree stores statistical information about the examples it contains. This allows the algorithm to compute standard deviations of the target-value for all subsets created by all the available tests. The splitting criterium checks if the examples in one leaf are sufficiently coherent with respect to their target value. A leaf is split when it contains enough examples for the statistics to be significant reliable: *min_ex_size* $= 30$.

3 Empirical Evaluation

We compare the original logistic regression approach with the probability trees approach. To this extend, we implemented our own version of the original TD(λ) learning algorithm as described in [3]. We perfomed experiments in two different environments: a ring world and a simple grid world. Experimental results are presented only for the 5-state deterministic ring world, as also used in [3]. Our ring world contains 5 interconnected circles. A circle indicates a state in the world. The agent has two different actions $\mathcal{A}$={*N*,*P*}. *N* moves the agent to the adjacent state in clockwise rotation. *P* moves the agent in the counter-clockwise direction. The agent can only observe whether it is in state 1 or not, i.e. the observation bit is on (1) if the agent is in state 1 and off (0) otherwise.

As in [3], we used symmetric action-conditional networks of depth 1, 2 and 3 as question networks. For the experiments with the classical TD-networks we used a learning rate $\alpha = 0.5$, obtaining similar results to the ones presented in [3].

To compare the different learning algorithms, we used the root mean-squared error ($RMSE$) to determine the quality of the learned models. This error at time t is calculated by comparing for each node i the correct target z_i^* with the one predicted by the learned model y_t^i. The correct targets are computed using full knowledge of the environment. Hence, if the $RMSE$ converges to 0, a correct answer network has been learned. The experiments are performed in an episode-based fashion. All experiments present the average $RMSE$ as a function of the number of episodes over 10 different runs.

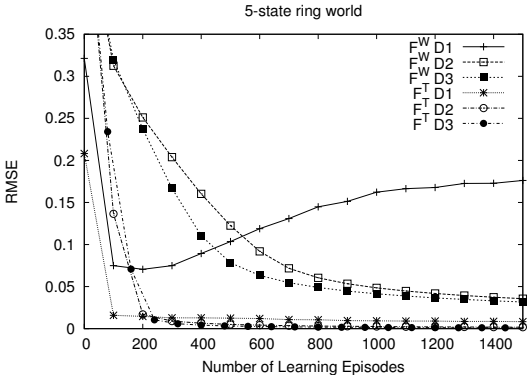

Figure 2. $RMSE$-curves for the symmetric action-conditional networks.

Figure 2 shows the results for the ring world with question networks of depth 1, 2 and 3. f^T always converges faster than f^W for networks of equal depth. For networks of depth 1 it is not possible to provide a completely accurate answer network, as the question network is too small to represent the full environment, but the probability tree learner quickly learns the best approximation. As expected, networks with larger depth perform better. The results for the grid world also show a faster learning performance for the decision trees.

4 Conclusion

We introduced the use of probability trees as answer networks in TD Networks. We illustrated how to translate the standard TD(1) learning approach into a training example generator and evaluated the performance of a simple incremental and greedy tree induction algorithm. The experimental evaluation shows consistently that the learning performance of the probability trees outperforms the original logistic regression approach. We consider this an important step towards the wider applicability of TD Networkss.

As we only regard this work as a proof of concept, a wide range of future work is possible. The current implementation of the tree induction algorithm could be significantly improved, for example by including tree restructuring operators or extending the learning algorithm to learn model-trees to combine the advantages of regression trees and logistic regression. Also, other non-parametric learners could be substituted for the probability tree learner. One exciting direction is the use of more elaborate observations, such as those for relational worlds. In this context decision trees offer the advantage of the more flexible parameterisation.

REFERENCES

[1] D. Fierens, J. Ramon, H. Blockeel, and M. Bruynooghe, 'A comparison of approaches for learning probability trees', in *Proceedings of the 16th European Conference on Machine Learning (ECML-05)*, pp. 556–563, (2005).

[2] R.S. Sutton and B. Tanner, 'Temporal-difference networks', in *Advances in Neural Information Processing Systems 17*, pp. 1377–1384, (2004).

[3] B. Tanner and R.S. Sutton, 'Td(λ) networks: temporal-difference networks with eligibility traces', in *Proceedings of the 22nd international conference on Machine learning (ICML-05)*, pp. 888–895, (2005).