Performance Analysis of a New Updating Rule for TD(λ) Learning in Feedforward Networks for Position Evaluation in Go Game

Horace Wai-kit Chan wkchan@cs.cuhk.edu.hk
Irwin King king@cs.cuhk.edu.hk
John C. S. Lui cslui@cs.cuhk.edu.hk

Department of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong

ABSTRACT

In this paper, a new updating rule for applying Temporal Difference (TD) learning to multilayer feedforward networks is derived. Networks are trained to evaluate Go board positions by TD(λ) learning with different values of λ. Performance of each network is estimated by letting it play against other networks. Results show that non-zero λ gives better learning for the network and statistically, larger λ gives better performance.

1. Introduction

In reinforcement learning, there is no teacher available to give the correct output for each training example. The output produced by the learning agent is fed to the environment and a scalar reinforcement value (reward) is returned. The learning agent tries to adjust itself to maximize the reward.

It is very often that the actions taken by the learning agent to produce an output will affect not only the immediate reward but also the subsequent ones. In this case, the immediate reward only reflects partial information about the action. It is called delayed-reward.

Temporal difference (TD) learning is a type of reinforcement learning for solving delayed-reward prediction problems. Unlike supervised learning which measures error between each prediction and target, TD uses the difference of two successive predictions to learn. The advantage of TD learning is that it can update weights incrementally and converge to a solution faster [5].

Go is a perfect information 2-player zero-summed game with a search space complexity larger than Chess and other similar board games. Moreover, the Go game is rich in tactics and strategies which makes it difficult for computer to play. The current trend in designing computer Go program is to use knowledge base with a set of rules engineered by human experts. Programs based on this approach are very weak. Our goal is to have the computers to "learn" from its own experience.

TD learning has been successfully applied in evaluating static position in Backgammon [6][7]. The program (called TD-gammon) achieves human master level. TD learning has also been used to evaluate static board position of Go games [2][4]. With specially designed neural network architecture, the resulting program is able to beat a commercial Go program using a weak playing level. However, only TD(0) is used in that application. TD(0) degenerates to the same algorithm of supervised learning which is simpler in design.

In this paper, a new learning rule for applying TD(λ) to backpropagation network is derived. A simple backpropagation network is trained with TD(λ) learning to evaluate static positions of Go games. Performance of networks trained with different values of λ is compared.

2. Background : TD(λ) Learning

In a delay-reward prediction problem, the observation-outcome sequence has the form $(o(1), o(2), \ldots, o(m), \chi)$ where each $o(t)$ is an observation vector available at time $t, 1 \leq t \leq m$ and χ is the outcome of the sequence. For each observations, the learning agent makes a prediction of χ, forming a sequence : $(P(1), P(2), \ldots, P(m))$.

Assuming the learning agent is an artificial neural network, update for a weight w of the network with the classical *gradient descent* update rule for supervised learning is :

$$\begin{aligned}\Delta w &= -\eta \nabla_w E \\ &= \eta \sum_{t=1}^{m} (\chi - P(t)) \nabla_w P(t) \qquad (1)\end{aligned}$$

where η is the learning rate and $\nabla_w E$ is the gradient vector ,$\frac{\partial E}{\partial w}$, of the mean square error function :

$$E = -\frac{1}{2} \sum_{t=1}^{m} (\chi - P(t))^2$$

In [5], Sutton derived the incremental updating rule for equation (1):

$$\Delta w(t) = \eta (P(t+1) - P(t)) \sum_{k=1}^{t} \nabla_w P(k) \qquad \text{for } t = 1, 2, \ldots, m \text{ where } P(m+1) \overset{def}{=} \chi. \qquad (2)$$

To emphasize more recent predictions, an exponential factor λ is multiplied to the gradient term :

$$\Delta w(t) = \eta (P(t+1) - P(t)) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P(k) \qquad \text{where } 0 \leq \lambda \leq 1 \qquad (3)$$

This results in a family of learning rules, TD(λ), with different values of λ. There are 2 special cases. First, when $\lambda = 1$, Eq. (3) falls back to Eq. (2) which produce the same training result as the supervised learning in Eq. (1). Second, when $\lambda = 0$, since $0^0 = 1$, Eq. (3) becomes

$$\Delta w(t) = \eta \; (P(t+1) - P(t)) \; \nabla_w P(t)$$

which has a similar form as Eq.(1). So the same training algorithm for supervised learning can be used for TD(0).

3. TD Learning with Backpropagation Network

3.1. New Generalized Delta Rule

In this section, we are going to derive a new generalized delta rule for TD(λ) learning. The classical generalized delta rule for a multi-layer feedforward network [3] is:

$$\Delta w_l = \eta \, y_{l-1} \, \delta_l^T$$

where w_l is a $m \times n$ weight matrix connecting layer $(l-1)$ and l, m is the size of layer $l-1$ and n is the size of layer l,

η is the learning rate (a scalar),

y_{l-1} is a column vector of output in layer $l-1$,

δ_l^T is transpose of the column vector δ_l which is the error propagated from layer l to $l-1$,

$l = 0$ for the input layer.

The vector of backpropagated error, δ_l, is defined differently for output layer and for hidden layer as :

$$\delta_l = \begin{cases} (T - Z) \otimes f_l'(net_l) & \text{if } l \text{ is an output layer} \\ f_l'(net_l) \otimes w_{l+1}\delta_{l+1} & \text{if } l \text{ is a hidden layer} \end{cases}$$

where $f_l'(\cdot)$ is the derivative of transfer function, f_l ,in layer l,
$\otimes$ denotes the element-wise vector multiplication,
net_l is the weighted sum in layer l,
δ_{l+1} is the delta value backpropagated from the upper layer of layer l,
T is the target vector,
Z is the output vector.

To apply TD learning to the multi-layer feedforward network, we extract the term $(T - Z)$ from the original δ_l and obtain a new delta, δ_l^* which is defined as

$$\delta_l^* = \ \mathbf{diag}\,[f_l'(net_l)]$$

where **diag** is the diagonal matrix and l is the output layer. If l is a hidden layer :

$$\delta_l^* = f_l'(net_l)\delta_{l+1}^* w_{l+1}^T$$

With the new delta, equation for change of each weight is rewritten as :

$$[\Delta w_l]_{ij} = \eta\,(T - Z)^T\,[\delta_l^*]_j\,[y_{l-1}]_i \tag{4}$$

where $[\delta_l^*]_j$ is the j-th column of vector δ_l^*,
$[y_{l-1}]_i$ is the i-th element in vector y_{l-1}.

Unlike the original delta which is a vector backpropagated from an upper to a lower layer, the new delta, δ_l^*, is now a $m \times n$ matrix where m is the size of output layer and n is the size of layer l. The error term $(T - Z)$ is needed for calculation of every weight increment.

Comparing gradient decent in supervised learning in Eq.(1) and the backpropagation with new delta in Eq. (4), $\nabla_w P(t)$, the gradient term at time t for weight w' is

$$\nabla_{w'} P(t) = [\delta_l^*(t)]_j\,[y_{l-1}(t)]_i$$

where $w' = [w_l(t)]_{ij}$ is the ij-th element in the weight matrix w_l at the time t.

By substituting this result to the formula of TD(λ) learning in Eq. (3), we get

$$[\Delta w_l(t)]_{ij} = \eta(P(t+1) - P(t))^T \sum_{k=1}^{t} \lambda^{t-k}\,[\delta_l^*(k)]_j\,[y_{l-1}(k)]_i \tag{5}$$

where $\Delta w_l(t)$ is the matrix of increment of weight connecting layer l and $l - 1$ for prediction $P(t)$. The term inside summation is called the *history vector*, denoted by $[h_l(t)]_{ij}$

We now obtain updating rules of TD learning by backpropagation. The weight update is performed by Eq. (5) with the new delta.

3.2. Training Procedure

Compared to the original backpropagation algorithm, the new procedures for TD(λ) learning requires some more storage for keeping the following values :

1. The *previous output vector*, $P(t-1)$, at time t, which is used in computing every weight change.
2. The *history vector*, $h_l(t, i, j) = \sum_{k=1}^{t} \lambda^{t-k}\,[\delta_l^*(k)]_j\,[y_{l-1}(k)]_i$ for each weight connecting i-th node in layer $l - 1$ to j-th node in layer l. It has the same size as the output vector. Each weight shall have its own history vector.

The training procedure involves 3 stages (at time t):

1. **Feedforward** : calculation of new prediction $P(t)$.
2. **Weight update** : calculation of weight increments by Eq. (5) using the history terms at time $t-1$.
3. **Backprop** : calculation of the new deltas at time $t+1$, $\delta^*_{l(t+1)}$, for each layer l, starting from the output layer. The history term is updated by :

$$h_l(t,i,j) = [\delta^*_l(t+1)]_j \, [y_{l-1}(t+1)]_i + \lambda h_l(t-1,i,j)$$

4. TD Learning in Position Evaluation

An important advantage of applying TD(λ) learning in the Go game is that network can be trained by self-playing. There is no need to handcraft many training examples as in supervised learning. The self-training strategy is used in TD-gammon [6].

4.1. Experimental Designs

We performed experiments using different values of λ to train networks of the same architecture and analyze the network performance by letting them play against one another.

Network Architecture Feedforward network with 1 hidden layer is used. To reduce the network complexity and training time, we use small Go board of size 7×7. The architecture of the network is 51-30-1. The input vector consists of information of all 49 points on the board, color of next player and relative number of captured stones (called prisoner). The difference of number of prisoners of each color is important in position evaluation. It is represented by one value which is proportional to (*number of prisoner captured by black - number of prisoner captured by white*).

Issues in Training The network is used to evaluate board positions in a game. Before training, the weights are initialized randomly by values between -1.0 and 1.0. During self-playing, it is used to evaluate all valid moves of a board position and a move is chosen according to the scores. The same initial configuration is used in 4 networks which are trained with different values of λ : $0, 0.25, 0.5$ and 0.75. Totally 100,000 games are trained for each network. The same set of experiments is repeated four times to check the consistency of performance.

Unlike Backgammon, playing Go does not have the element of chance. If the moves with highest score given by the neural network evaluator is always played, training will be trapped in local minima easily. In that situation, the same game will be played repeatedly. To avoid this, the move is selected stochastically using Gibbs sampling according to the score in evaluation [4]. The probability for a valid move being selected is :

$$P(m_i) = \frac{1}{Z} \exp\left(\frac{score(m_i)}{T}\right) \tag{6}$$

where Z is a normalizing factor equals to $\sum_i \exp(\frac{score(m_i)}{T})$,
$score(m_i)$ is the output of the network for board position m_i,
T is the temperature determining the degree of randomness.

The temperature T is set to a high value (1.0) in the beginning and in the course of training it is reduced by a factor α : $T_{k+1} = \alpha T_k, 0 < \alpha < 1$. In the experiments, α is set to be 0.92 and is decreased after every 20 games. Finally, it becomes a very small value in which the "best" move will be played in most of the time.

Testing Method and Results Instead of playing against existing programs, the trained network are made to play against each other. The move is selected stochastically according to Eq.(6) with a very low temperature. This is to make the testing games non-deterministic to compare performance of the networks under many different board positions. If prediction made by a network is accurate, its playing strength should be high. Results of the testing games are summarized below.

Networks $\lambda =$	Average Winning Percentage with Opponents :				
	$\lambda = 0.00$	$\lambda = 0.25$	$\lambda = 0.50$	$\lambda = 0.75$	relative score
0.00	—	46.11%	43.58%	37.38%	127.06
0.25	53.89%	—	48.26%	43.41%	145.56
0.50	56.43%	51.74%	—	46.19%	154.35
0.75	62.63%	56.59%	53.81%	—	183.03

Table 1. *Results showing performance of networks trained by different TD(λ)*

4.2. Discussion

Each of the row entry represents the winning percentage out of 2,000 simulated games against each of the column entry, e.g. network trained by TD(0) wins 46.11% against the one trained by TD(0.25). We obtain the relative score of each network by summing up all the winning percentages in each row. From the scores, we can observe that performance of TD(0) is not as good as non-zero λ value. Performance is statistically better with larger λ.

Among the results in 4 replications, performance is not always consistent with others. It shows that final performance depends on the initial weights. However, in all cases, TD(λ), where $\lambda > 0$, out-performs TD(0). This shows that static board evaluation of Go game will be more accurate when supplied with the information of past states.

5. Conclusion

TD(λ) learning can be applied to train multi-layer feedforward networks using the *new generalized delta rule* derived in Eq. (5). More storage is needed in TD(λ) than in supervised learning but we can have incremental learning and faster convergence. [5].

TD(λ) learning is used in training neural networks to evaluate board positions in Go games. Different values of λ are tested. We find that TD(λ) with non-zero λ performs better than TD(0). Furthermore, the performance gets better with larger λ.

In the future, we plan to use TD learning to train networks to analyze some game specific properties of Go such as territory and shape.

References

[1] J.A.Bate, *A Beginner's Introduction to GO* Manuscript available by Internet anonymous FTP from bsdserver.ucsf.edu, file /Go/RULES.PS.Z.

[2] W.K.Chan and I.King, "Temporal Difference Learning and Its Application in Go," *Computer Strategy Game Programming Workshop,* The Chinese University of Hong Kong, May 1995.

[3] D.E.Rumelhart, G.E.Hinton, and R.J.Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing (PDP): Exploration in Microstructure of Recognition (Vol. 1),* Chapter 8, MIT Press, Cambridge, Massachusetts, 1986.

[4] N.N.Schraudolph, P.Dayan, and T.J.Sejnowski, "Temporal Difference Learning of Position Evaluation in the Game of Go," *Advances in Neural Information Processing 6,* 1994.

[5] R.S.Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine Learning,* 3: 9-44, 1988.

[6] G.Tesauro, "Practical Issues in Temporal Difference Learning," *Machine Learning,* 8: 257-278, 1992.

[7] G.Tesauro, "Temporal Difference Learning and TD-Gammon," *Communications of ACM,* Vol.38, No.3, March 1995.