Convolutional Restricted Boltzmann Machine Features for TD Learning in Go

By Yan Largman and Peter Pham
Advised by Honglak Lee

1. Background & Motivation

Although recent advances in AI have allowed Go playing programs to become moderately strong even on full 19x19 boards, Go programs still cannot approach the highest levels of play. One of the strongest Go playing programs (strongest in 2007) utilizes the UCT algorithm, which is a Monte-Carlo search algorithm paired with a value function [1].

The value function was learned through temporal difference function approximation with all possible 3×3 templates[1] on all board positions. However, there may be features in such a set that are not very meaningful in Go. If more meaningful features could be found, then the state-of-the-art algorithm could be applied on top of these features to create an even stronger Go playing agent. In this project, we applied the convolutional restricted Boltzmann machine to Go data in an attempt to learn features better suited for TD learning. We then evaluated the features on a series of tests to determine the effectiveness of the Boltzmann machine in capturing features significant in Go.

[1] A template is an assignment of black, white, or empty to every position in a given area (for example a 3×3 area). Application of an $n \times n$ template features to a $b \times b$ Go board results in $3^{n \times n} \times (b - n + 1)^2$ features, where each feature is 1 if a template feature exactly matches a location on the board.

2. Algorithms

2.1 Convolutional Restricted Boltzmann Machines

Our primary goal for this project was to automatically generate translation-invariant features that are informative for encoding the state of a Go board. Therefore, a convolutional restricted Boltzmann machine (CRBM) seemed well suited to the task. A CRBM is a network that learns statistical relationship between a visible layer and a hidden layer [2]. In particular, by using the contrastive divergence, an RBM adjusts the value of the weights so that the reconstruction of the visible layer from its hidden layer representation matches the visible layer training data as close as possible. In a CRBM, weights are shared between groups of hidden layer units so that signal propagation from the visible layer to the hidden layer can be implemented as a convolution.

Our input layer for the CRBM consisted of a three-channel representation for a Go board, where the unit in the visible layer channels were 1 if the position was black, white, or empty positions, respectively, and 0 otherwise. We modeled these units in two ways for comparison: real-valued units and multinomial units. For CRBM training with real-valued visible units we used the mean-field approximation, so no sampling of visible units was done during the reconstruction phase of contrastive divergence. For CRBM training with multinomial visible units we used softmax sampling to determine in which

channel a unit would be active, according to the formula

$$\frac{\exp(I(h_{i,j}^c))}{\sum_{c=1}^{3}\exp(I(h_{i,j}^c))}$$

where $h_{i,j}^c$ is the hidden unit at position (i,j) in channel *c*, and

$$I(h_{i,j}^c) = \sum_b (W^b *_f h^b)_{i,j} + c$$

is the increase in energy resulting from the activation of that unit, which the top down signal propagated from each hidden layer h^b through the corresponding weights W^b by a full convolution plus a bias term *c*.

Because Go exhibits rotational and reflectional symmetries, we augmented an ordinary CRBM by adding rotational and reflectional invariance to the training process. This was accomplished by adding a layer that ties the weights of the different orientations of the kernels together. In practice this involved the following: We first apply the weights to the raw board in each possible configuration. Then, instead of sampling the hidden layer corresponding to each configuration independently, we take the softmax of the activations so only one of these hidden unit layers is active. Finally, we sum the contrastive divergence gradient for each configuration (rotating/reflecting appropriately to recover the original orientation) to find the gradient for the original shared weight that generated these configurations.

2.2 Temporal Difference Reinforcement Learning

Once our features were trained, we used temporal difference learning with a linear value function approximation for the afterstates of moves to train a Go-playing agent. Specifically, we used the update equations [5]

$$\delta \leftarrow r + \gamma V(s) - V(s)$$
$$e \leftarrow \gamma\lambda e + \nabla_\theta V(s)$$
$$\theta \leftarrow \theta + \alpha\delta e$$

Where

$$V(s) = sigmoid(\vec{\theta}^T \vec{\phi}_s),$$

$\vec{\theta}$ is the weight vector the agent learns, and $\vec{\phi}_s$ is the feature vector associated with state s. In this case, the gradient is simply

$$\nabla_\theta V(s) = sigmoid(\vec{\theta}^T \vec{\phi}_s)(1 - sigmoid(\vec{\theta}^T \vec{\phi}_s))\vec{\phi}_s$$

For the reward function, we gave a reward of 1 on a win, and all other rewards were 0. Therefore, the value function can be seen as approximating the probability of winning a game from the given state. Although intermediate rewards, such as a reward for capturing opponent stones or a penalty for losing stones, could be given, the true goal of the agent is only to win, and in some cases sacrificing stones or passing up an opportunity to capture stones could be beneficial. Furthermore, other similar applications of TD(λ) learning to Go have had success using only the reward for a win [4]. We also used λ=0, which is a one-step TD back-up, as this was a parameter value that achieved the most successful learning in related applications [4][6].

For our agent's policy, we used an ε-greedy policy, so the agent either picked the move that resulted in the board position with maximum value with probability (1-ε) or a random move with probability ε. The state space of Go is very large, so a soft-greedy policy was used so to ensure that the agent would explore the state space during training. The only constraint imposed on the policy was that suicide moves could not be chosen. This was done to prevent games with infinite length.

3. Experimental Setup

3.1 Feature Comparison using TD

We evaluated the effectiveness of the CRBM features against enumerated 2×2 template features by training a value functions for both features and then pitting the ε-greedy (ε=0.10) policies defined by these value functions against each other. The agents used ε-greedy policies so that games would not deterministically play out the same way each time.

To achieve a fair comparison, the number of template features and CRBM features that we used was approximately the same. We used 105 weights of size 3×3 for a total of 5145 CRBM features on a 9×9 board. In our experiments, non-rotational/reflection-invariant features performed better in TD learning play, so the features we used for comparison are regular CRBM features. For template features, we used all 81 enumerated 2×2 templates for a total of 5184 template features on a 9×9 board.

The value functions were trained offline on a thousand games played by GNUGo [7] against itself. Although TD learning is not guaranteed to converge in the offline case, we found that in practice this training was sufficient to establish a baseline at which we could compare the features.

Games were played until both sides passed or 100 moves had been played (at which point no winner was declared). Because scoring a board and deciding the winner of a Go game is non-trivial, we used GNUGo through GTP protocol to score the finished games. A standard komi (score handicap to offset black's initial move advantage) of 5.5 was used, and both features were given turns playing as black.

3.2 Self-play Bootstrapping using TD

As an additional confirmation that CRBM features can be useful for TD learning, we tested whether the model learned from CRBM features could learn to consistently beat itself through self-play bootstrapping.

Using the value functions learned from the offline training on GNUGo games, two models initially using the same value function played against each other. One of the models used TD learning to update itself online during play, while the other model remained static. We decayed ε of the learning model during training, so initially the algorithm gives more emphasis to exploration, and gradually it starts exploring less and exploiting more.

3.3 Winner Classification

We sought to indirectly measure how much information the CRBM kernels encode about the game by classifying the winner of professional games given the middle or end board state configurations. We used an SVM for classification, making sure to find the best kernels for each set of features and using k-fold cross-validation to find the best C value. As a baseline feature set, we used all possible 2x2 templates applied at each position on the board. The kernels that we used in this classification problem were trained so that the end number of features would be about the same as the 2x2 features for a fair comparison.

4. Results

4.1 Feature Comparison & Self-play Results

We ran 63 games of CRBM features vs. 2×2 template features. CRBM won 54 of these games, which is a win rate of 85.71%.

Furthermore, when we conducted the self-

play experiment with CRBM features, the agent was able to learn to consistently beat its former instance in 110 iterations. This seems to indicate that same TD learning process previously applied to enumerated template features in the state-of-the-art could be applied in the same fashion to CRBM features.

Unfortunately, because full convergence for TD in Go can take on the order of 60,000 games [3], we did not have the time or computational power to test the fully trained models for both features.

4.2 Qualitative Analysis

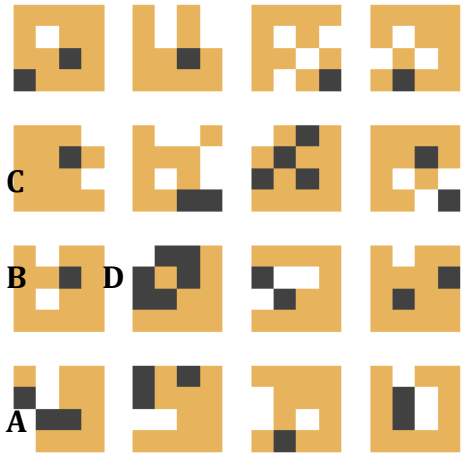

Figure 1 (Meaningful Shapes Learned by CRBM)

In order to obtain some sense of what the CRBM learns in each of our formulations, we visualized the weights of the first layer. Visualization was performed in several ways as there is not simply one configuration that corresponds to a particular kernel. One simple visualization we performed was assigning the state at a particular point of a kernel to the state with the highest weight in a given kernel (Fig 1). Qualitative analysis suggests that the multinomial CRBM learns more meaningful features than the real-valued CRBM.

A comparison between randomly sampled multinomial (Fig. 2) and real-valued kernels (Fig. 3) shows that the multinomial kernels are more sensitive to empty space on the board (this follows from the definition of our visualization). Appropriately considering empty space is important in Go because the primary objective is to surround it.

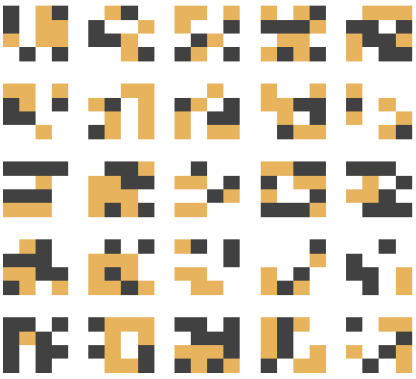

Figure 2 (Randomly Sampled Real-valued Bases)

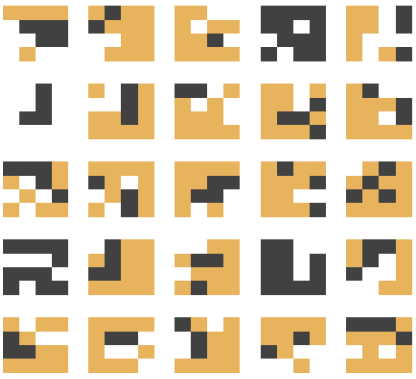

Figure 3 (Randomly Sampled Multinomial Bases)

From the visualization of the multinomial CRBM weights, it is possible to make out both common shapes and small, but meaningful situations in Go. For example, kernel A in Fig. 1 shows a set of black and white stones that form a formation called a pinwheel. Kernels B and C in the same figure show two orientations of a move on black's part called a "peek", which is characterized by a stone approaching a one space gap between two stones of the opposing color. Finally, kernel D

in the figure shows a partial eye shape, the essential shape that can determine the life or death of entire groups of stones.

4.3 Winner Classification Results

The results of the winner classification given a board are somewhat discouraging because no feature set we learned from a CRBM of equivalent size to the baseline feature set performed better. However, they show definitively that training the CRBM with rotational invariance improves the generalization of the features to the task of winner classification.

Training Samples	Exact 2x2 Templates	Multinomial CRBM	MCRBM with Invariances
20	0.5945	0.5055	0.5096
40	0.6439	0.5156	0.5629
60	0.6732	0.5204	0.6320
80	0.6807	0.5381	0.6225
100	0.7083	0.5324	0.6632
120	0.7006	0.5685	0.6549
140	0.7197	0.5926	0.6984

5 Conclusion & Future Work

Although our results have not shown definitively that CRBM features can be better than a set of naively enumerated features for high level play, there is hope that CRBM features can better cover the enormous state space in Go. In our experiments, CRBM features were able to outperform enumerated template features given the same number of features and the same TD training algorithm and data, so it certainly seems possible that applying the UCT and TD algorithms to CRBM could improve performance of the state-of-the-art techniques.

Furthermore, because expert players evaluate the board for shape patterns which can be decomposed into subpatterns, application of higher layers of a deep belief network could create more effective features. Also, instead of using the raw board as the input to the CRBM, the visible layer could be augmented with easily-computed Go-specific information at each position, such as the number of liberties or proximity to an edge.

6 Sources & Works Cited

1. Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In Ghahramani, Z., ed., ICML, volume 227, 273–280. ACM.

2. Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Bottou, L., & Littman, M. (Eds.), Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09). ACM, Montreal (Qc), Canada.

3. Schraudolph, N. N., Dayan, P. and Sejnowski, T. J. Temporal difference learning of position evaluation in the game of Go. In J. D. Cowan, et al. Eds., Advances in Neural Information Processing Systems 6, 817-824. Morgan Kaufmann, San Mateo, Calif., 1994.

4. Silver, D., Sutton, R., & Muller, M.(2007). Reinforcement learning of local shape in the game of Go. 20th International Joint Conference on Artificial Intelligence (pp. 1053–1058).

5. Sutton, Richard S.; Barto, Andrew G.. Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 1998. p 221. http://site.ebrary.com/lib/stanford/Doc?id=10225275&ppg=221 Copyright © 1998. MIT Press. All rights reserved.

6. Tesauro G. Tesauro. TD-gammon, a self-teaching backgammon program, achieves master-level play. Neural Computation, 6:215–219, 1994.

7. http://www.gnu.org/software/gnugo/