Factorization Ranking Model for Move Prediction in the Game of Go

Chenjun Xiao and Martin Müller
Computing Science, University of Alberta
Edmonton, Canada
{chenjun,mmueller}@ualberta.ca

Abstract

In this paper, we investigate the move prediction problem in the game of Go by proposing a new ranking model named Factorization Bradley Terry (FBT) model. This new model considers the move prediction problem as group competitions while also taking the interaction between features into account. A FBT model is able to provide a probability distribution that expresses a preference over moves. Therefore it can be easily compiled into an evaluation function and applied in a modern Go program. We propose a Stochastic Gradient Decent (SGD) algorithm to train a FBT model using expert game records, and provide two methods for fast computation of the gradient in order to speed up the training process. Experimental results show that our FBT model outperforms the state-of-the-art move prediction system of Latent Factor Ranking (LFR).

Introduction

Go is an ancient Chinese board game. Two players, Black and White, take turns to place a single stone on an empty intersection on a Go board. Stones cannot be moved once placed, but are removed if surrounded by opponent stones. Both players' objective is to surround a larger total area of the board with their stones than the opponent by the end of the game. Even though its rules are very simple, Go is an very complex and deeply strategic game. Due to the fact that it's very difficult to construct a suitable evaluation function (Müller 2002), Go is one of the last classical board games that have not been mastered by computer programs.

When playing Go, human experts rely heavily on pattern recognition. Using their strong intuition, they can recognize promising moves and important regions of a Go board at a glance, without simulating many thousands of continuations like modern Go programs. Machine learning techniques can be used to acquire pattern knowledge from human game records. Such move prediction systems can directly serve as Go playing programs, but more importantly, they can play the role of an evaluation function to guide the tree search in an algorithm such as Monte Carlo Tree Search (MCTS) (Coulom 2006). A fast move prediction system can estimate all candidate moves and estimate how likely each move is to be played by an expert. A selective search algorithm can then focus on the most promising moves. Most state-of-the-art computer Go programs rely heavily on such knowledge (Browne et al. 2012). Using simple features such as patterns, popular offline training algorithms such as Minorization-Maximization (MM) (Coulom 2007) and Latent Factor Ranking (LFR) (Wistuba and Schmidt-Thieme 2013) produce models that are fast enough to evaluate at every node in a MCTS game tree.

Recently, Deep Convolutional Neural Networks (DCNN) have achieved human-level performance and have outperformed traditional techniques for predicting expert moves (Clark and Storkey 2015; Maddison et al. 2014). However, well-performing networks are very large, with millions of weights (Maddison et al. 2014), and even on specialized hardware their evaluation speed is orders of magnitude slower than traditional techniques. How to best combine DCNN with MCTS is still an open question. Therefore, it makes sense to continue research on high performance move prediction algorithms as well.

The current paper proposes and evaluates the *Factorization Bradley-Terry* model for move prediction. In popular move prediction algorithms for the game of Go, such as (Coulom 2007; Wistuba and Schmidt-Thieme 2013), each move in a game is modeled as a *group of features* which describe it. Best move selection then becomes a competition among the groups representing all legal moves in a Go position.

The major innovation of the proposed Factorization Bradley-Terry (FBT) model is to combine the strengths of the two leading approaches (Coulom 2007; Wistuba and Schmidt-Thieme 2013), and consider the interaction between individuals within the same group as part of a probability-based framework, which is inspired by the Factorization Machine model (Rendle 2010).

The expert move prediction problem is formulated within this new model. Similar to (Wistuba and Schmidt-Thieme 2013), a Stochastic Gradient Descent (SGD) algorithm is used to train a model. Two techniques accelerate the training process: an efficient incremental gradient update, and a Monte Carlo method for unbiased approximate gradient computation. The experiments test FBT by varying the size of data sets as well as the expressiveness of the feature sets. Results show that FBT outperforms Latent Factor Ranking.

Related Work

The popular high-speed move prediction systems represent each move as a combination of a group of features, learn weights for each feature from expert game records by supervised learning, and define an evaluation function based on the weights which is used to rank moves. Examples include Bayesian Full Ranking (Stern, Herbrich, and Graepel 2006), Bayesian Approximation for Online ranking (Wistuba, Schaefers, and Platzner 2012)(Weng and Lin 2011), Minorization-Maximization (MM) (Coulom 2007) and Latent Factor Ranking (LFR) (Wistuba and Schmidt-Thieme 2013).

MM is an efficient learning algorithm which is very popular in Computer Go. Move prediction is formulated as a competition among all possible moves in a Go position, and each move is represented as a group of features. A simple probabilistic model named Generalized Bradley-Terry model (Hunter 2004) defines the probability of each feature group winning a competition. The strength of each feature is estimated by maximizing the likelihood of the expert moves winning all competitions over the whole (large) training set.

LFR improves the performance of move prediction by taking pairwise interactions between features into account. This is modelled using a Factorization Machine (FM) (Rendle 2010), an efficient model especially for sparse interactions, which is widely used in the recommendation community (Rendle et al. 2011). Move prediction is considered as a binary classification problem, with the expert move in one class and all other legal moves in the other. A drawback of LFR is that the evaluation it produces does not provide a probability distribution over all possible moves. This makes it much harder to combine with other kinds of evaluations.

The successes of LFR and MM highlight two important aspects: the interactions between features within a group, and an efficient ranking model to describe the interactions between all possible groups. The Factorization Bradley Terry model combines these insights: like MM, move prediction is modeled as a competition among all moves using a Bradley-Terry model. As in LFR, a Factorization Machine models the interaction between the features of a move.

The Factorization Bradley-Terry Model

The Factorization Bradley-Terry (FBT) model is introduced for estimating the probability of a feature group winning competitions with other groups, while taking into account the pairwise interactions between features within each group. Let $k \in \mathbb{N}^+$ be the dimension of the factorization. Let $\mathcal{F}$ be the set of all possible features. For feature $f \in \mathcal{F}$, let $w_f \in \mathbb{R}$ be the feature's (estimated) *strength*, and $v_f \in \mathbb{R}^k$ be its *factorized interaction vector*. Here, the *interaction strength* between two features f and g is modeled as $\langle v_f, v_g \rangle = \sum_{i=1}^{k} v_{f,i} \cdot v_{g,i}$. The parameter space of this model is $\mathbf{w} \in \mathbb{R}^{|\mathcal{F}|}$ and $\mathbf{v} \in \mathbb{R}^{|\mathcal{F}| \times k}$.

The idea of using factorized interaction vectors comes from Factorization Machines (Rendle 2010). The matrix $\mathbf{v} \in \mathbb{R}^{|\mathcal{F}| \times k}$ is a sparse approximation of the full pairwise interaction matrix $\Phi \in \mathbb{R}^{|\mathcal{F}| \times |\mathcal{F}|}$, which is huge for large $|\mathcal{F}|$. A proper choice of k makes such models especially efficient for generalization under sparse settings (Rendle 2010). In Computer Go, settings $k = 5$ and $k = 10$ are popular (Wistuba and Schmidt-Thieme 2013). Richer feature sets and larger training sets might require increasing k for best performance.

The strength of a group $\mathcal{G} \subseteq \mathcal{F}$ in FBT is defined as

$$E_{\mathcal{G}} = \sum_{f \in \mathcal{G}} w_f + \frac{1}{2} \sum_{f \in \mathcal{G}} \sum_{g \in \mathcal{G}, g \neq f} \langle v_f, v_g \rangle \tag{1}$$

In FBT, the popular exponential model (Huang, Lin, and Weng 2006) is used to model winning probabilities in competition among groups. Given N groups $\{\mathcal{G}^1, \ldots, \mathcal{G}^N\}$, the probability that group $\mathcal{G}^i$ wins is given by:

$$P(\mathcal{G}^i \text{ wins}) = \frac{exp(E_{\mathcal{G}^i})}{\sum_{j=1}^{N} exp(E_{\mathcal{G}^j})} \tag{2}$$

Move Prediction in Go using the proposed model

Let $\mathcal{S}$ be the set of possible Go positions and $\Gamma(s)$ be the set of legal moves in a specific position $s \in \mathcal{S}$. The objective of the *move prediction problem* is to learn from a training set to predict expert moves. *Features* $\mathcal{F}$ are used to describe moves in a given game state. Each move is represented by its set of active features $\mathcal{G} \subseteq \mathcal{F}$. The training set $\mathcal{D}$ consists of cases $\mathcal{D}_j$, with each case representing the possible move choices in one game position s_j.

$$\mathcal{D}_j = \{\, \mathcal{G}_j^i \mid \text{for } i = 1, \ldots, |\Gamma(s_j)| \}$$

As in MM, the process of choosing a move is modeled as a competition, using the ranking model defined before. In this setting, the set of features $\mathcal{F}$ is the set of "players" which compete in groups. Let the winner of the competition among groups in $\mathcal{D}_j$ be $\mathcal{G}_j^*$. From (2), the probability of the current test case is

$$P(\mathcal{D}_j) = \frac{exp(E_{\mathcal{G}_j^*})}{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i})} \tag{3}$$

Parameter Estimation for FBT

In the proposed FBT model (3), each feature's *strength* and *factorized interaction vector* is learned from a training set $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$. This section first poses the learning of these parameters as an optimization problem, then describes a Stochastic Gradient Descent (SGD) algorithm for estimating them, and finally provides two methods for accelerating the learning process.

Definition of the Optimization Problem

Suitable parameters in FBT can be estimated by maximizing the likelihood of the training data. For the jth training case D_j, the negative log loss function is given by

$$l_j = -\ln \frac{exp(E_{\mathcal{G}_j^*})}{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i})} = -E_{\mathcal{G}_j^*} + \ln \sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i}) \tag{4}$$

Assuming that competitions are independent, the loss function becomes simply the sum over all training examples:

$$L = \frac{1}{n} \sum_{j=1}^{n} l_j \tag{5}$$

The corresponding optimization problem is:

$$\min_{\mathbf{w} \in \mathbb{R}^{|\mathcal{F}|}, \mathbf{v} \in \mathbb{R}^{|\mathcal{F}| \times k}} L + \lambda_w R(\mathbf{w}) + \lambda_v R(\mathbf{v}) \tag{6}$$

R is an L_2 regularization function to avoid overfitting, and λ_w and λ_v are regularization parameters. The choice of values is discussed in the Experiments section.

Parameter Learning with Stochastic Gradient Descent

We propose a SGD algorithm to learn the model parameters. (1) can be expressed as a linear function with respect to every single model parameter $\theta \in \mathbf{w} \cup \mathbf{v}$ (Rendle et al. 2011), which means that if $\theta \in \mathcal{G}$,

$$E_{\mathcal{G}} = \theta h_{\mathcal{G}}(\theta) + g_{\mathcal{G}}(\theta) \tag{7}$$

where both $h_{\mathcal{G}}(\theta)$ and $g_{\mathcal{G}}(\theta)$ are independent of the value of θ, and if $\theta = w_s \in \mathbf{w}$, $h_{\mathcal{G}}(\theta) = 1$, if $\theta = v_{s,q} \in \mathbf{v}$, $h_{\mathcal{G}}(\theta) = \frac{1}{2} \sum_{t \in \mathcal{G}, t \neq s} v_{t,q}$. Note that if $s \notin \mathcal{G}$, both $h_{\mathcal{G}}(\theta)$ and $g_{\mathcal{G}}(\theta)$ are simply zero. Using these facts, the gradient of the loss function l_j for parameter θ can be written as:

$$\begin{aligned}
\nabla_j \theta &= -\frac{\partial E_{\mathcal{G}_j^*}}{\partial \theta} + \frac{\partial \ln \sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i})}{\partial \theta} \\
&= -h_{\mathcal{G}_j^*}(\theta) + \frac{\sum_{i=1}^{|\Gamma(s_j)|} \partial exp(E_{\mathcal{G}_j^i}) / \partial \theta}{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i})} \\
&= -h_{\mathcal{G}_j^*}(\theta) + \frac{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i}) h_{\mathcal{G}_j^i}(\theta)}{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i})}
\end{aligned} \tag{8}$$

The resulting gradients $\nabla_j \theta$ for updating a parameter θ are:

$$\begin{array}{ll}
-1 + \frac{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i}) \mathbb{I}\{s \in \mathcal{G}_j^i\}}{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i})} & \theta = w_s, s \in \mathcal{G}_j^* \\
-H_{\mathcal{G}_j^*,q} + \frac{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i}) \mathbb{I}\{s \in \mathcal{G}_j^i\} H_{\mathcal{G}_j^i,q}}{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i})} & \theta = v_{s,q}, s \in \mathcal{G}_j^* \\
\frac{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i}) \mathbb{I}\{s \in \mathcal{G}_j^i\}}{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i})} & \theta = w_s, s \notin \mathcal{G}_j^i \\
\frac{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i}) \mathbb{I}\{s \in \mathcal{G}_j^*\} H_{\mathcal{G}_j^i,q}}{\sum_{i=1}^{|\Gamma(s_j)|} exp(E_{\mathcal{G}_j^i})} & \theta = v_{s,q}, s \notin \mathcal{G}_j^*
\end{array} \tag{9}$$

with indicator function $\mathbb{I}$ and $H_{\mathcal{G},q} = \frac{1}{2} \sum_{t \in \mathcal{G}, t \neq s} v_{t,q}$.

Remark. The new model (3) can be considered an extension of the generalized Bradley-Terry model (Hunter 2004), which applies a Factorization Machine to model the interaction between team members when computing a team's ability. It is very similar to the conditional exponential model (Pietra, Pietra, and Lafferty 1997), which is widely used in computational linguistics. From this standpoint, it seems like optimization algorithms for solving these models such as Alternating Optimization (AO) (Rendle et al. 2011), Maximization-Minimization (MM) (Hunter 2004) (Coulom 2007) and Improved Iterative Scaling (IIS) (Pietra, Pietra, and Lafferty 1997) might also be suitable for solving (6). Generally speaking, all of these methods try to solve the optimization problem iteratively. At each iteration, to update a parameter θ, they first construct a sub-optimization problem according to some functional properties (e.g. Jensen's inequality and the upper bound of logarithmic function), then find the optimizer as the starting point of the next iteration by fixing all the other parameters except θ and setting the derivative of θ to zero. However, when considering the problem (6), the derivatives of a group $\mathcal{G}_j^i$ for θ is

$$\frac{\partial exp(E_{\mathcal{G}_j^i})}{\partial \theta} = exp(\theta h_{\mathcal{G}_j^i}(\theta)) exp(g_{\mathcal{G}_j^i}) h_{\mathcal{G}_j^i}(\theta)$$

If $\theta \in \mathbf{v}$, the values $h_{\mathcal{G}_j^i}(\theta)$ will change with different i and j, and thus the update of $\theta \in \mathbf{v}$ at each iteration dose not have a closed form solution. Numerical method such as Newton's method is required to compute the update value for θ, which will introduce further computational complexity. In conclusion, it would be very inefficient to use such algorithms for problem (6).

Efficient Gradient Computation

The first bottleneck of computing the gradient of the loss function l_j via (8) is calculating $exp(E_{\mathcal{G}_j^i})$ for each $i \in \{1, \ldots, |\Gamma(s_j)|\}$. For one particular group $\mathcal{G}$, a direct computation of $exp(E_{\mathcal{G}})$ is obviously in $O(|\mathcal{G}|k)$. However, we can compute this term once and update it in constant time when some parameters corresponding to feature $s \in \mathcal{G}$ have been changed. Suppose that we update the parameter θ to θ', then for a group $\mathcal{G}$, $E_{\mathcal{G}} = \theta h_{\mathcal{G}}(\theta) + g_{\mathcal{G}}(\theta)$ should be updated to

$$E'_{\mathcal{G}} = \theta' h_{\mathcal{G}}(\theta) + g_{\mathcal{G}}(\theta) = E_{\mathcal{G}} + (\theta' - \theta) h_{\mathcal{G}}(\theta) \tag{10}$$

Therefore, at the beginning of training we precompute all $E_{\mathcal{G}}$ in the training set once, then update them efficiently through equation (10) when necessary, which only takes constant time.

Right now the computational complexity of $E_{\mathcal{G}}$ depends on the complexity of calculating $h_{\mathcal{G}}(\theta)$. For $\mathbf{w}$, the h term in (10) is just 1, so the update time is constant. For updating $\theta = v_{s,q} \in \mathbf{v}$, the direct computation of the h term in (10) takes $O(k)$ time. However, it can be updated in constant time by first precomputing $R_{\mathcal{G},q} = \frac{1}{2} \sum_{t \in \mathcal{G}} v_{t,q}$ for each group in the training set, since

$$h_{\mathcal{G}}(v_{s,q}) = R_{\mathcal{G},q} - \frac{1}{2}v_{s,q} \tag{11}$$

After an update of $v_{s,q}$ to $v'_{s,q}$, $R_{\mathcal{G},q}$ can be updated in constant time by

$$R'_{\mathcal{G},q} = R_{\mathcal{G},q} + \frac{1}{2}(v'_{s,q} - v_{s,q}) \tag{12}$$

Using equations (10) to (12), $exp(E_{\mathcal{G}})$ can be updated in constant time.

Regarding the space complexity of the fast gradient computation method described above, let γ=$\max_{j\in\{1,\ldots,n\}}|\mathcal{D}_j|$. For each group $\mathcal{G}$ in $\mathcal{D}_j$, storing one $E_{\mathcal{G}}$ and one $R_{\mathcal{G},q}$ for each factorization dimension $1 \leq q \leq k$, gives total space complexity $O(n\gamma k)$.

Approximate Gradient

Another bottleneck of computing the gradient via (8) is traversing the set of all group. For example, in the *move prediction problem* in 19×19 Go, the average value of $|\Gamma(s_j)|$ is about 200, which implies significant computation cost. A Monte Carlo approach can address this problem by sampling an approximate gradient that matches the real gradient in expectation.

Let P_j^i be the probability that $\mathcal{G}_j^i$ is the winner of $\mathcal{D}_j$ using model (2). The probability distribution $P_j = (P_j^1, \ldots, P_j^{|\Gamma(s_j)|})$ over $\mathcal{D}_j$ allows us to construct an unbiased approximate gradient. Consider a mini-batch of groups created by sampling M groups $\{\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(M)}\}$ from $\mathcal{D}_j$ according to the probability distribution P_j. The **sampled approximate gradient** of the loss function l_j for parameter θ is:

$$\hat{\nabla_j}\theta = -h_{\mathcal{G}_j^*}(\theta) + \frac{1}{M}\sum_{i=1}^{M} h_{\mathcal{G}_j^{(i)}}(\theta) \tag{13}$$

We now show that the **sampled approximate gradient** matches the gradient (8) in expectation.

Lemma 1. $E_{P_j}[\hat{\nabla_j}\theta] = \nabla_j\theta$

Proof.

$$\begin{aligned} E_{P_j}[\hat{\nabla_j}\theta] &= -\,h_{\mathcal{G}_j^*}(\theta) + E_{P_j}[\frac{1}{M}\sum_{i=1}^{M} h_{\mathcal{G}_j^{(i)}}(\theta)] \\ &= -\,h_{\mathcal{G}_j^*}(\theta) + \sum_{i=1}^{|\Gamma(s_j)|} P_j^i h_{\mathcal{G}_j^i}(\theta) = \nabla_j\theta \end{aligned} \tag{14}$$

□

Experiments

The experiments in this section evaluate FBT and the approximate gradient computation for the move prediction task in the game of Go. FBT is compared with the state-of-the-art move prediction algorithm of Latent Factor Ranking (LFR) (Wistuba and Schmidt-Thieme 2013).

Setup

The training process is performed on a 2.40GHz machine with 64GB memory. Both FBT and LFR use the same hyper parameters as in (Wistuba and Schmidt-Thieme 2013), with learning rate $\alpha = 0.001$ and regularization parameters $\lambda_w = 0.001$, $\lambda_v = 0.002$. The same stopping criteria for the training process are applied: if an algorithm's prediction accuracy on a validation set is not increased for three iterations, the training is stopped and the best-performing weight set is returned. Three data sets of increasing size, $S_1 \subset S_2 \subset S_3$, are used, which contain 1000, 10000, and 20000 master games respectively. The games are in the public domain at https://badukmovies.com/pro_games.

The learned model is evaluated on a test set. Test and validation set contain 1000 games each, and are disjoint from the training sets and from each other. The prediction accuracy in all experiments is defined as the probability of choosing the expert moves over all test cases, not as the average accuracy over the 12 game phases used in (Wistuba and Schmidt-Thieme 2013). That metric gives higher percentages since it weighs the opening and late endgame more, where the prediction rate is above average.

Choice of Features

When using large patterns, many algorithms can get a high move prediction accuracy at the beginning of the game (Wistuba and Schmidt-Thieme 2013). This is because these moves are often standard opening moves which can be represented very accurately by large patterns. It is informative to also compare move prediction algorithms without such large patterns. Therefore, two different feature sets, the **small pattern feature set** F_{small} and the **large pattern feature set** F_{large} are used in all tests. Both sets contain the same non-pattern features. F_{small} adds only 3×3 patterns, while F_{large} includes large patterns. F_{small} helps provide a better comparison of FBT and LFR, while F_{large} yields a more powerful move prediction system overall.

The F_{small} features are the same as in the LFR implementation that is part of the open source Fuego program (Enzenberger et al. 2010). Large patterns in F_{large} were added for this study. Most features are similar to earlier work such as (Coulom 2007; Wistuba and Schmidt-Thieme 2013). For implementation details, see the Fuego code base (Enzenberger and Müller 2008 2015).

The simple features used in this work are:

- **Pass**
- **Capture, Extension, Self-atari, Atari** Tactical features similar to (Coulom 2007).
- **Line and Position (edge distance perpendicular to Line)** ranges from 1 to 10.
- **Distance to previous move** feature values are 2,…, 16, $\geq$ 17. The distance is measured by $d(\delta x, \delta y) = |\delta x| + |\delta y| + max\{|\delta x|, |\delta y|\}$.
- **Distance to second-last move** uses the same metric as previous move. The distance can be 0.

Table 1: Results for F_{small}: probability of predicting the expert move with FBT and LFR, for $k = 5$ and $k = 10$. Best results for each method in bold.

Training set	FBT5	FBT10	LFR5	LFR10
S_1	32.56%	**32.82%**	30.01%	**30.08%**
S_2	33.18%	**33.42%**	30.95%	**31.63%**
S_3	33.46%	**34.01%**	31.13%	**31.94%**

Table 2: Results for F_{large}, same format as Table 1.

Training set	FBT5	FBT10	LFR5	LFR10
S_1	35.83%	**35.96%**	34.38%	**34.47%**
S_2	38.26%	**38.31%**	37.12%	**37.34%**
S_3	38.48%	**38.75%**	37.56%	**37.69%**

- **Fuego Playout Policy** These features correspond to the rules in the playout policy used in Fuego. Most are simple tactics related to stones with a low number of liberties.
- **Side Extension** The distance to the closest stones along the sides of the board.
- **Corner Opening Move** Standard opening moves.
- **CFG Distance** Distance when contracting all stones in a block to a single node in a graph (Friedenbach 1980).
- **Shape Patterns** The *small pattern* set contains all patterns of size 3×3. The *large pattern* set includes circular patterns with sizes from 2 to 14, harvested as in (Stern, Herbrich, and Graepel 2006; Wistuba and Schmidt-Thieme 2013). All shape patterns are invariant to rotation, translation and mirroring.

Expert Move Prediction

To compare the prediction accuracy of FBT and LFR, on each data set, two different models were trained for each algorithm, setting the dimension of the *factorized interaction vector* to $k = 5$ and $k = 10$. These methods with a specific k value are called FBTk and LFRk respectively. All results are averaged over five runs, since both FBT and LFR randomize initial parameter values.

Experiment with F_{small} Results for the small pattern set F_{small} are presented in Table 1. Both methods improve with larger training sets. For each combination of training set S_i and k, FBT outperforms LFR. The best model learned by FBT, for $k = 10$ and S_3, outperforms the best LFR model by 2.07%. Note that the gap between the maximum and minimum prediction accuracy of the five runs over all tests of FBT is 0.64%, which shows that FBT is quite stable and the performance difference between LFR and FBT is significant. Prediction accuracy increases with growing k, confirming the observation in (Wistuba and Schmidt-Thieme 2013).

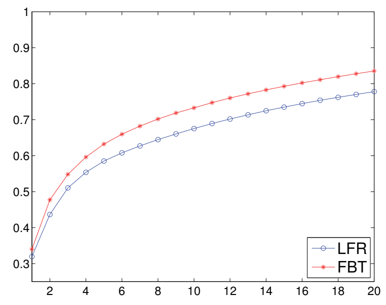
Figures 1(a) and 1(b) compare the details of the move prediction results of LFR and our method. Figure 1(a) compares the cumulative probability of predicting the expert's move within the top n ranked moves, for S_3 with $k = 10$. While both methods rank most expert moves within the top 20, the gap between FBT and LFR grows initially up to about rank 5, then holds steady throughout. Figure 1(b) presents the prediction accuracy per game stage, where each game phase consists of 30 moves as in (Wistuba and Schmidt-Thieme 2013). FBT outperforms LFR at every stage of the game.

Experiment with F_{large} For creating large pattern features, pattern harvesting collects all patterns that occur at least 10 times in the training set. For training sets S_1, S_2 and S_3, the number of such patterns is 9390, 84660 and 152872 respectively. Following (Wistuba, Schaefers, and Platzner 2012) and (Stern 2008), Figure 1(f) shows the distribution of largest matches for the different pattern sizes in each game phase. Large size patterns dominate in the opening (phase 1), then disappear rapidly. Later in the game only small size patterns are matched. Table 2 shows the prediction accuracy of FBT and LFR trained with F_{large}. Regarding the margin of error, the gap between the maximum and minimum prediction accuracy of the five runs over all tests of FBT is 0.82%. Figures 1(c) and (d) show the cumulative prediction probabilities and the accuracy per game stage. Both FBT and LFR learn better models with F_{large} than with F_{small}, with huge differences in the opening due to large patterns, and both methods achieve very high accuracy at the endgame. As with F_{small}, FBT outperforms LFR on every data set, but the differences between the two methods are smaller. This can be expected, especially for the opening stage, where both algorithms learn the same large-scale patterns for the standard opening moves. FBT retains its advantage for the middle game, which is important in practice since many games are decided there.

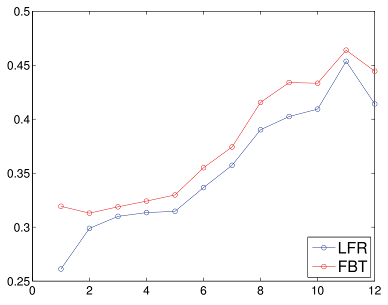
Sampling for Approximate Gradient Computation

To evaluate the new online sampling method for approximate gradient computation, experiments with $k = 5$ were run on all the three training sets, varying the number of samples as $m \in \{5, 10, 20, 30\}$. In the following, FBT_Sm denotes approximate FBT with m samples, while FBT with full gradient computation is labeled FBT_full. LFR is also included. The results are presented in Table 3. The performance of FBT_S increases with growing number of samples m, and approaches that of FBT_Full. It is especially notable that FBT_S performs better than LFR even with small number of samples. Regarding the cumulative rank, FBT_S30 achieves 82.26% for top 20 prediction, compared to 83.54% for FBT_Full and 77.81% for LFR (the right-most data points in Figure 1(a)).

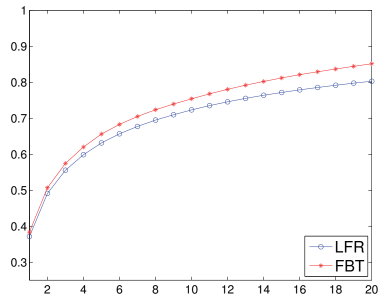
The purpose of using Monte Carlo approximation in the gradient computation is to accelerate the training process while still keeping the theoretical soundness and practical accuracy. Figure 1(e) plots the prediction accuracy of FBT_full and FBT_Sm as a function of time for all tested m. The plot shows the best prediction accuracy achieved so far on the validation set in 30 seconds intervals, starting from the same initial parameter value, on a training set of 1000 games with $k = 5$ using feature set F_{small}. All sampling algorithms finish the training process within 15 minutes, while FBT_Full needs about an hour. FBT_S20 and FBT_S30 outperform FBT_Full in the early phases. The performance of FBT_S30 comes very close to FBT_Full, while using less



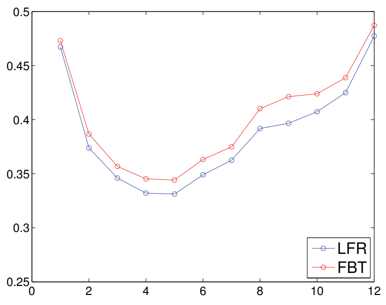
(a) F_{small} cumulative rank



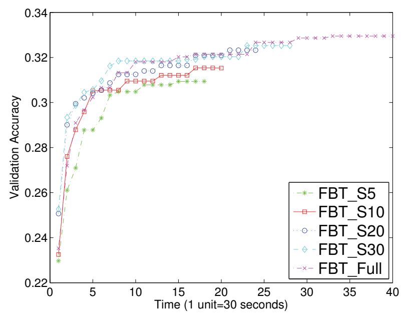
(b) F_{small} accuracy per game stage



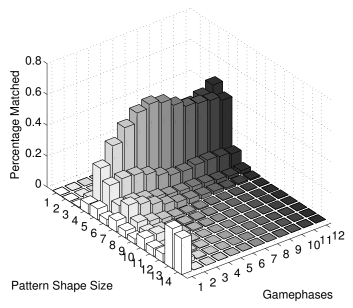
(c) F_{large} cumulative rank



(d) F_{large} accuracy per game stage



(e) Accuracy of approximation



(f) Pattern Size Distribution

Figure 1: Figures (a) and (b): FBT and LFR trained with F_{small}. Figures (c) and (d): results for F_{large}. In Figure (a) and (c), x-axis is the number of ranked moves, while in Figure (b) and (d) x-axis represents the game stage. Figure (e): accuracy of approximate sampling over time. Figure (f): Distribution of patterns with different size harvested at least 10 times in different game phases.

Table 3: Move prediction by FBT with different sample size on different size data set, results for LFR and FBT_Full shown for comparison. Bold values highlight best results among sampling methods.

Training set	FBT_S5	FBT_S10	FBT_S20	FBT_S30	LFR	FBT_Full
S_1	30.04%	30.81%	31.06%	**31.78** %	30.01%	32.56%
S_3	32.07%	32.90%	32.98%	**33.03**%	30.95%	33.18%
S_3	32.54%	33.01%	33.09%	**33.12**%	31.13%	33.46%

time to train (15 minutes vs 1 hour). The performance of the approximate gradient estimators increases with the number of samples, since the variance is reduced.

Conclusion and Future Work

The new Factorization Bradley Terry (FBT) model, combined with an efficient Stochastic Gradient Decent algorithm, is applied to predicting expert moves in the game of Go. Experimental results show that FBT outperforms LFR, the previous state of the art high-speed move predictor.

Future work includes: 1. apply FBT in a MCTS program such as Fuego. 2. combine FBT with Deep Convolutional Neural Networks (DCNN) to get a fast accurate move predictor. For example, in computer vision DCNN has been used to extract features of pictures and combined with a traditional classifier in a powerful object detection system (Girshick et al. 2014). A similar approach might also work for move prediction problem: use DCNN to extract new features, then apply FBT to learn feature weights. Another possible approach is to automatically switch between FBT and DCNN, in order to get a high prediction accuracy while still keeping a low processing time.

Acknowledgements

The authors wish to thank the anonymous referees for their valuable advice. This research was supported by NSERC, the Natural Sciences and Engineering Research Council of Canada.

References

Browne, C.; Powley, E.; Whitehouse, D.; Lucas, S.; Cowling, P.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of Monte Carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games* 4(1):1–43.

Clark, C., and Storkey, A. J. 2015. Training deep convolutional neural networks to play Go. In Bach, F. R., and Blei, D. M., eds., *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Proceedings*, 1766–1774. JMLR.org.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In van den Herik, J.; Ciancarini, P.; and Donkers, H., eds., *Proceedings of the 5th International Conference on Computer and Games*, volume 4630/2007 of *Lecture Notes in Computer Science*, 72–83. Turin, Italy: Springer.

Coulom, R. 2007. Computing Elo ratings of move patterns in the game of Go. In *Proc. Computer Games Workshop 2007 (CGW2007)*. 113–124.

Enzenberger, M., and Müller, M. 2008-2015. Fuego. http://fuego.sourceforge.net.

Enzenberger, M.; Müller, M.; Arneson, B.; and Segal, R. 2010. Fuego - an open-source framework for board games and Go engine based on Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games* 2(4):259–270.

Friedenbach, K. J. 1980. *Abstraction Hierarchies: A Model of Perception and Cognition in the Game of Go*. Ph.D. Dissertation, University of California, Santa Cruz.

Girshick, R.; Donahue, J.; Darrell, T.; and Malik, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, 580–587. IEEE.

Huang, T.-K.; Lin, C.-J.; and Weng, R. C. 2006. Ranking individuals by group comparisons. In *Proceedings of the 23rd international conference on Machine learning*, 425–432. ACM.

Hunter, D. R. 2004. MM algorithms for generalized Bradley-Terry models. *Annals of Statistics* 384–406.

Maddison, C. J.; Huang, A.; Sutskever, I.; and Silver, D. 2014. Move evaluation in Go using deep convolutional neural networks. *CoRR* abs/1412.6564.

Müller, M. 2002. Computer Go. *Artificial Intelligence* 134(1–2):145–179.

Pietra, S. D.; Pietra, V. D.; and Lafferty, J. 1997. Inducing features of random fields. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 19(4):380–393.

Rendle, S.; Gantner, Z.; Freudenthaler, C.; and Schmidt-Thieme, L. 2011. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, 635–644. ACM.

Rendle, S. 2010. Factorization machines. In Webb, G. I.; Liu, B.; Zhang, C.; Gunopulos, D.; and Wu, X., eds., *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, 995–1000. IEEE Computer Society.

Stern, D.; Herbrich, R.; and Graepel, T. 2006. Bayesian pattern ranking for move prediction in the game of Go. In *Proceedings of the 23rd international conference on Machine learning*, 873–880. ACM.

Stern, D. H. 2008. *Modelling Uncertainty in the Game of Go*. Ph.D. Dissertation.

Weng, R. C., and Lin, C.-J. 2011. A bayesian approximation method for online ranking. *The Journal of Machine Learning Research* 12:267–300.

Wistuba, M., and Schmidt-Thieme, L. 2013. Move prediction in Go - modelling feature interactions using latent factors. In Timm, I., and Thimm, M., eds., *KI2013*, volume 8077 of *Lecture Notes in Computer Science*, 260–271. Springer.

Wistuba, M.; Schaefers, L.; and Platzner, M. 2012. Comparison of bayesian move prediction systems for Computer Go. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, 91–99. IEEE.