Move Prediction in Go – Modelling Feature Interactions Using Latent Factors

Martin Wistuba and Lars Schmidt-Thieme

University of Hildesheim
Information Systems & Machine Learning Lab
{wistuba, schmidt-thieme}@ismll.de

Abstract. Move prediction systems have always been part of strong Go programs. Recent research has revealed that taking interactions between features into account improves the performance of move predictions. In this paper, a factorization model is applied and a supervised learning algorithm, Latent Factor Ranking (LFR), which enables to consider these interactions, is introduced. Its superiority will be demonstrated in comparison to other state-of-the-art Go move predictors. LFR improves accuracy by 3% over current state-of-the-art Go move predictors on average and by 5% in the middle- and endgame of a game. Depending on the dimensionality of the shared, latent factor vector, an overall accuracy of over 41% is achieved.

Keywords: go, move prediction, feature interaction, latent factors

1 Introduction

Since the early days in research of Computer Go, move prediction is an essential part of strong Go programs. With the application of Upper Confidence bounds applied to Trees (UCT) in 2006 [1, 2], which improved the strength of Go programs a lot, it became even more important. Go programs using UCT infer from semi-random game simulations which move is a good candidate. The policies for choosing the next move during the simulations are implied by predicting a human expert's move. Due to the fact that an average Go game has 250 turns with 150 possible move choices on average, the move position evaluation does not only need to be accurate but also fast to compute to achieve a positive impact on the strength of the Go program.

State-of-the-art move predictors are ranking moves on the board by the use of different features. Upfront, the strength of each feature is learned with various supervised learning algorithms. The prediction can be improved by using additional features, but as seen in [3, 4] it can also be improved by considering the impact of feature interactions.

The contribution of this paper are fourfold.

– A supervised move ranking algorithm for Go is presented which is by now the most accurate. Additionally, it is easy to implement and fast to compute.

- The model of Factorization Machines [5] is transfered from the domain of recommender systems to move prediction in Go.
- A new update rule for ranking with Factorization Machines is presented.
- Deeper insights into Go move features and its interactions are given and in detail investigated.

2 Related Work

Most move predictors for Go are either using Neural Networks [6, 7] or are estimating ratings for moves using the Bradley Terry (BT) model or related models [3, 4, 8]. Latter mentioned approaches model each move decision as a competition between players, the move chosen by the human expert player is then the winning player and its value is updated accordingly.

Another possibility to divide the Go move predictors into two classes is how they consider interactions between features. There are two variants, one models the full-interaction of all features [3, 9] and the others do not consider them at all [4, 6, 8].

The first mentioned approach which is modelling all interactions has the advantage that more information is taken into account. The disadvantage is that this approach does not scale because the amount of training data needed increases exponentially with the number of features. The latter approach does not have this disadvantage but therefore also has no information about the feature interactions. In practice, approaches not considering feature interactions at all proved to be more accurate. Stern's [3] full-interaction model used a learning set of 181,000 games with 4 feature groups but only predicted 34% of the moves correctly. Using the same approach with no interaction, Wistuba et al. [4] has shown that easily 9 feature groups can be used and, using a learning set of only 10,000 games, 37% of moves were predicted correctly. Ibidem, it was tried to combine advantages of both approaches by using an approach without feature interaction and adding a special feature that represented a combination of few features. It was shown that this can improve the prediction quality significantly.

The contribution of this work is to introduce a method which cannot be sorted into the before mentioned categories. It introduces an algorithm for the move prediction problem of Go that is combining both advantages by presenting a model which learns the strength of interactions between features but still scales with the number of features.

3 Game of Go

The game of Go is one of the oldest two player board games which was probably invented around the 4^{th} century B.C. in China. It is played on a board with $n \times n$ intersections (n is usually 9, 13 or 19). The players move alternately. At each turn the player has the option to place a stone at an intersection or to pass. Enemy stones that are surrounded by own stones will be removed from the board. The aim of the game is to capture enemy stones and territory. The

game ends after both players have passed, the winner is then the one with more points which are calculated by the number of captured stones and the size of the captured territory. Further informations can be found at http://gobase.org.

3.1 Technical Terms

Finally some technical terms in Go are explained to make it possible to understand the features used in this work.

Ko The ko rule is a restriction on the legal moves. Moves that change the board state to the same state like two moves before are forbidden.

Chain The connected string of stones of same color.

Liberty An empty intersection next to a chain is called liberty.

Atari A chain is in atari if there is only one liberty left, so that the opponent can capture the chain within one move.

Capture If you place your stone in such a way that the enemy chain has no liberties left. This chain will be removed from the board and each stone is called a prisoner and count as one point each.

Illegal move A move is illegal if it either breaks the ko rule, places a stone at an intersection that is already occupied or it captures an own chain.

3.2 Complexity

Go is one of the last board games not being mastered by computer programs. Actually, Go programs are still far away from beating professional players, only playing on the level of stronger amateurs on the 19×19 boards. One of the reasons is the high complexity of Go. The upper bound of possible board positions is $3^{361} \approx 10^{170}$ and still 1.2% of these are legal [10]. Comparing Go with Chess, not only the board size is bigger (19x19 vs. 8x8) but also the number of potential moves. The average number of potential moves per turn in Go is about 150, Chess has only a few dozen. Additionally, no static heuristic approximating the minimax value of a position was found so far. That is, it is not possible to apply depth limited alpha-beta search with reasonable results. Concluding, even from a perspective of complexity Go is by far more difficult than Chess. A perfect strategy for $n \times n$ Chess only requires exponential time but Go is PSPACE-hard [11] and even subproblems a player has to deal with in every turn has proven to be PSPACE-complete [12].

4 Move Prediction using Feature Interactions

This section first introduces the terminology and a model which is capable to represent interactions between features. Then, the Latent Factor Ranking algorithm is presented in Section 4.3. Finally, Section 4.4 describes the features used for the experiments.

4.1 Terminology

This work will use the terminology introduced in [4]. A single game in Go is formalized as a tuple $G := (\mathcal{S}, \mathcal{A}, \Gamma, \delta)$ where $\mathcal{S} := \mathcal{C}^{n \times n}$ is the set of possible states and $\mathcal{C} := \{black, white, empty\}$ is the set of colors. The set of actions $\mathcal{A} := \{1, \ldots, n\}^2 \cup \{pass\}$ defines all possible moves and $\Gamma : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ is the function determining the subset of legal moves $\Gamma(s)$ in state s. $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \cup \{\emptyset\}$ is the transition function specifying the follow-up state for a state-action pair (s, a), where $\delta(s, a) = \emptyset$ iff $a \notin \Gamma(s)$. In the following, a state-action pair (s, a) will be abstracted by m features represented by $x \in \mathbb{R}^m$. Even though x is only the abstracted state-action pair, in the following for notational convenience it will anyways be called state-action pair.

In this work only binary features $x_i \in \{0, 1\}$ are used and so the set of active features in (s, a) is defined as $\mathcal{I}(x) := \{i : x_i = 1\}$.

Given a training set $\mathcal{D}$ of move choice examples

$$\mathcal{D}_j := \left\{ x^{(1)} = x(s_j, a_1), \ldots, x^{(|\Gamma(s_j)|)} = x\left(s_j, a_{|\Gamma(s_j)|}\right) \right\},$$

it is assumed without loss of generality that $x^{(1)}$ is always the state-action pair chosen by the expert.

4.2 Problem Description and Model

The *move prediction problem* in Go is defined given a state s, to predict the action $a \in \Gamma(s)$ that is chosen by the expert player. Due to the fact that there might be several similar good moves and the application of move prediction in the UCT algorithm, a ranking of the legal moves is searched such that the expert move is ranked as high as possible. Therefore, a ranking function is sought that minimizes $\sum_{\mathcal{D}_i \in \mathcal{D}} \sum_{j=1}^{rank(a_1)} \frac{1}{j}$, where $rank(a_1)$ is the ranking of the action chosen by the human expert in the decision problem $\mathcal{D}_i$.

Like other contributions on the topic of move prediction in Go, this work also is a supervised method that estimates the strength of different features based on a set of games between strong human players. The big difference is that additionally the strength of the interaction of two features is considered. The model of Factorization Machines [5] is applied which is defined as

$$\hat{y}(x) := w_0 + \sum_{i=1}^{m} w_i x_i + \sum_{i=1}^{m} \sum_{j=i+1}^{m} v_i^T v_j x_i x_j.$$

Because in this work only binary features are used, a notation-wise simpler model is applied

$$\hat{y}(x) := w_0 + \sum_{i \in \mathcal{I}(x)} \left(w_i + \sum_{j \in \mathcal{I}(x), i \neq j} \theta_{i,j} \right),$$

with

$$\theta_{i,j} := \frac{1}{2} v_i^T v_j.$$

where w_i is the influence of the feature i independent of all other occurring features, whereas $\theta_{i,j}$ is the interaction between features i and j. The matrix $V \in \mathbb{R}^{m \times k}$ implies the matrix $\Theta \in \mathbb{R}^{m \times m}$ and is the reason why LFR does not struggle with the problem of full-interaction models i.e. the lack of examples. The dimension $k \ll m$ has to be chosen by the user. The greater k, the more potential information can be stored in the interaction vectors v_i. The k latent factors per feature will then be shared and thus scalability problems are avoided when the number of features increases while the number of feature values is kept low. As shown in Figure 5(a), already for very small k LFR seems to be optimal. Thus, k can be treated as a constant and only $\Theta(m)$ values are needed. Nevertheless, for computational reasons, which are very important for Go playing programs based on UCT, it makes sense to precompute the matrix Θ.

We want to continue the discussion from Section 2 and explain the counterintuitive fact that no-interaction models achieve better results than full-interaction models. Also, we want to show why the model for LFR is capable of achieving better results.

There are various learning techniques using these models but they have the way how state-action pairs are ranked in common. No-interaction models learn weights w_i for each feature i whereas full-interaction models learn weights $w_{\mathcal{I}(x)}$. Then, all legal state-action pairs $x^{(j)}$ are ranked in descendend order of its predicted strengths $\sum_{i \in \mathcal{I}\left(x^{(j)}\right)} w_i$ respectively $w_{\mathcal{I}\left(x^{(j)}\right)}$. So far, it still looks like the full-interaction model considers more information. But useful values for $w_{\mathcal{I}\left(x^{(j)}\right)}$ can only be estimated if $\mathcal{I}\left(x^{(j)}\right)$ was seen at least once in the learning set. Thus, in practice, both kind of models do not have the same features to predict a state-action pairs strength. Normally, the no-interaction models have access to larger shape features which are very predictive and this additional information is worth more than the interaction.

The model of LFR is capable of using the same features as no-interaction models but still can consider feature interactions so that in this case indeed more information is used.

4.3 Latent Factor Ranking

The Latent Factor Ranking (LFR) is defined as follows. Each state-action pair is labeled with

$$y(x) = \begin{cases} 1 & \text{if } x \text{ was chosen in the example} \\ 0 & \text{otherwise} \end{cases}.$$

For the estimation of vector w and matrix Θ a stochastic gradient descent with $l2$ regularization is applied. The gradients are given as

$$\frac{\delta}{\delta \phi} \hat{y}(x) = \begin{cases} 1 & \text{if } \phi = w_0 \\ 1 & \text{if } \phi = w_i \text{ and } i \in \mathcal{I}(x) \\ \sum_{j \in \mathcal{I}(x) \setminus \{i\}} v_{i,f} & \text{if } \phi = v_{i,f} \text{ and } i \in \mathcal{I}(x) \\ 0 & \text{otherwise} \end{cases}$$

Instead of taking every state-action pair x into account, only those pairs with rank at least as high as the pair chosen by the expert i.e. $\hat{y}(x) \geq \hat{y}\left(x^{(1)}\right)$ are used. The idea behind this is that an implicit transitive relation between features is achieved, and moves that are not responsible for wrong rankings do not need to be touched.

Vector w is initialized with 0, V is initialized with values randomly drawn from the normal distribution with mean 0 and standard deviation 0.1. Learning rate α and regularization parameters λ_w and λ_v need to be estimated upfront as well as the dimension k. Algorithm 1 describes LFR in detail. In the following LFR with a specific dimension k is called LFRk. During the experiments, convergence was assumed when the prediction accuracy did not improve within the last three iterations.

Algorithm 1 Latent Factor Ranking

Input: Training set $\mathcal{D}$ with move decisions $\mathcal{D}_j = \left\{x^{(1)}, x^{(2)}, \ldots, x^{(|\Gamma(s_j)|)}\right\}$ in state s_j where $x^{(1)}$ was chosen by the expert.
Output: V and w necessary to predict future moves.

```
w ← 0, v_if ~ N(0, 0.1)
while not converged do
  for all D_j ∈ D do
    for all x ∈ D_j do
      if ŷ(x) ≥ ŷ(x^(1)) then
        Δy ← ŷ(x) − y(x)
        w_0 ← w_0 − α · Δy
        for all i ∈ I(x) do
          w_i ← w_i − α(Δy + λ_w w_i)
          for f = 1 to k do
            v_{i,f} ← v_{i,f} − α(Δy δ/δv_{i,f} ŷ(x) + λ_v v_{i,f})
```

4.4 Features

In Go two different kinds of features are distinguished, shape features and non-shape features. Shape features take the shape around a specific intersection on the board into account, non-shape features are every other kind of features in a move situation you can imagine. How shapes are extracted, harvested and represented is explained very well in [3].

In this work the same features are used as in [4]. That is, a feature subset of those proposed in [8] are used in order to allow a comparison to further prediction models. Since [4] does not define the features explicitly, this is caught up here. Features are divided into nine groups, from each group at most one feature is active for a given state-action pair, first mentioned features have higher priorities within the feature group. All features are binary because the approaches LFR is compared to cannot deal with other features types.

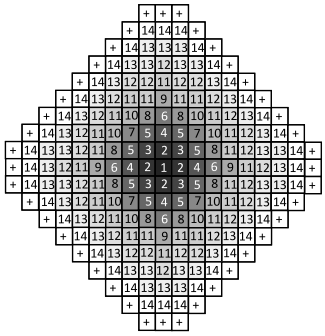


Fig. 1. The shapes are harvested as proposed in [3]: Fourteen circle shaped, nested templates are used which are regarded to be invariant to rotation, translation and mirroring. The shape template of size 14 considers the full board state.

1. **Pass** Passing in case of the last move was 1) no pass or 2) a pass.
2. **Capture** Capturing an enemy chain such that 1) an own chain is no longer in atari, 2) previous move is recaptured, 3) a connection to the previous move is prevented or 4) any other capture.
3. **Extension** Stone is placed next to an enemy chain such that it is in atari.
4. **Self-atari** Placing a stone such that your own chain is in atari.
5. **Atari** Placing a stone such that enemy chain is in atari when there is 1) a ko or 2) no ko.
6. **Distance to border** is one, two, three or four.
7. **Distance to previous move** is $2, \ldots, 16, \geq 17$ using distance measure $d(\Delta x, \Delta y) = |\Delta x| + |\Delta y| + \max\{|\Delta x|, |\Delta y|\}$
8. **Distance to move before previous move** is $2, \ldots, 16, \geq 17$ using distance measure $d(\Delta x, \Delta y) = |\Delta x| + |\Delta y| + \max\{|\Delta x|, |\Delta y|\}$
9. **Shape** Can be any shape that appeared at least ten times in the training set using the templates shown in Figure 1.

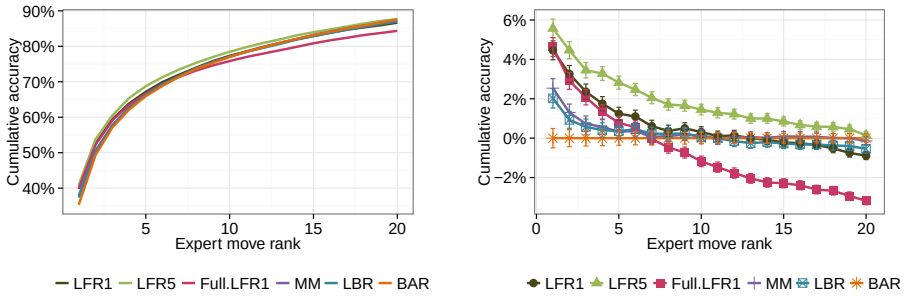
5 Experiments

In the following, at first LFR is compared to other Go move prediction algorithms and it is shown that it is significantly better for k. It will be shown that the interactions have a positive impact especially in situations where no big shapes are matched (shape sizes greater than 4) which finally results in the observed lift. Finally, the features and its interactions are discussed.

For the experiments a set of 5,000 respectively 10,000 games (i.e. approximately 750,000 respectively 1,500,000 move decisions) from the KGS Go Server[1] was used. These games are without a handicap and were played between strong human amateurs i.e. both are ranked at least 6d or at least one has a rank of 7d. As mentioned before, shapes were used if they occurred at least 10 times in the

[1] http://www.gokgs.com/

training set. In this way, 48,071 respectively 94,030 shapes were harvested and used for the learning process. Hyperparameters for LFR were estimated on a disjoint validation set and sought on a grid from 0 to 0.01 with step size 0.001. The learning rate $\alpha = 0.001$ was selected. For LFR1 the regularization parameters $\lambda_w = 0.001$ and $\lambda_v = 0$ were chosen, while for LFR5 $\lambda_w = 0.001$ and $\lambda_v = 0.002$ are optimal. All experiments were made on the 10k learning set otherwise explicitly stated. LFR is compared to Coulom's Minorization Maximization [8] (MM) as well as two further algorithms introduced in [4]: These are on the one hand an improvement of Stern's algorithm [3] now capable to deal with arbitrary many features, the Loopy Bayesian Ranking (LBR), and a variant of Weng's Bayesian Approximation Method [13] based on the Bradley Terry model adapted to the Go move prediction problem, the Bayesian Approximation Ranking (BAR). The experiments are made on a testing set of 1,000 games which are disjoint from the training and validation set. The accuracy is defined as the average accuracy of the first 12 game phases, where a game phase consists of 30 turns.



(a) Cumulative prediction accuracy in respect to the expert move rank.

(b) The move prediction accuracy given by the difference of the accuracy of an algorithm and BAR. (95% conf. intervals)

Fig. 2. The cumulative prediction accuracy in respect to the expert move rank.

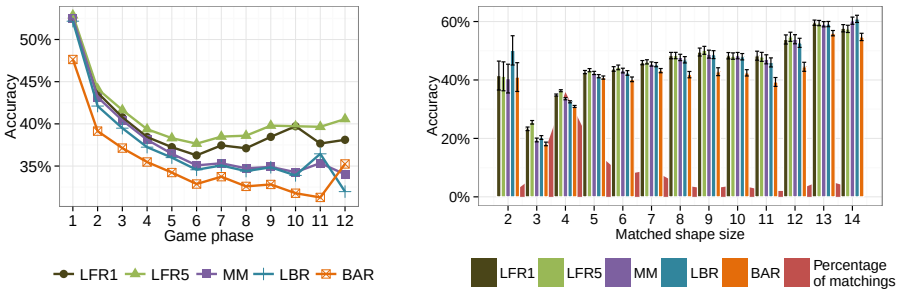
The resulting prediction quality of the aforementioned algorithms is depicted in Figure 2(a). Figure 2(b) shows this in detail by providing the results substracted by the results of BAR. The expert move rank is the rank assigned to the move chosen by the expert player in the actual game. Full-LFR1 is LFR1 which considers all state-action pairs for the update. Its results justify the choice of considering only state-action pairs $x^{(i)}$ where $\hat{y}\left(x^{(i)}\right) \geq \hat{y}\left(x^{(1)}\right)$ because Full-LFR performs poor for high expert move ranks. As can be seen, LFR outperforms the other algorithms significantly, especially for low expert move ranks. For FPR5 this holds even up to 18. Especially the lift for very low expert move ranks is notable.

Table 1. Probability for predicting the move chosen by the expert for different learning set sizes

Training set size	MM	LBR	BAR	LFR1	LFR5
5,000	37.00%	36.36%	34.24%	38.60%	**39.96%**
10,000	37.86%	37.35%	35.33%	39.78%	**40.90%**

Additionally, Table 1 compares the different prediction algorithms on two different sized training sets. Again, LFR outperforms every other algorithm. Finally, Figure 5(a) shows the predicting performance of LFR with growing k. The accuracy increases fast but then converges. It can be assumed that for $k > 10$ there will be no big improvements.

The intuition of learning move strengths by considering interactions between features was to achieve a higher prediction accuracy for cases where only smaller shapes are matched. Smaller shapes have less information and usually are more often matched in the later game phases. This goal is achieved by LFR as seen in Figure 3(a). The prediction accuracy in the first game phases (each game phase consists of 30 turns) is higher than the average accuracy due to the fact that there are standard opening moves. These can be learned very accurately by the shape features because most of these moves were harvested with very large shape sizes. This is also the reason why LFR is not better than the other algorithms because the shape features simply dominate all the others. Then, starting in game phase 6, when smaller shapes are matched and the other features gain more influence, the impact of the interactions becomes visible. Accuracy of LFR is then up to more than 5% better than all other approaches.



(a) Move prediction accuracy in different game phases. Each game phases consists of 30 turns.

(b) Accuracy of predicting the right move depending on the shape size of the right move. The red part in the background is the percentage of shape sizes of the matched shapes in total. (95% conf. intervals)

Fig. 3. LFR is better in ranking moves where only small shapes are available.

Figure 3(b) also supports the claim of successfully estimating the right move if only small shapes are matched. It shows the prediction accuracy in respect of the matched shape size of the expert move. For shape sizes 5-13 there is no significant change in comparison to the other algorithms, for full board shapes it is even worse. Matters are quite different for shapes sizes 3 and 4. The interactions seem to be responsible for the significant improvement of the accuracy. More than 40% of matched shapes for the move chosen by the expert are of sizes 3 or 4. This is the reason for the dramatic lift of the average prediction accuracy and the prediction accuracy in the later game phases. Additionally, considering that full board shapes are only matched during the first game phases and probably being part of standard opening moves, the advantage of the other algorithms for full board shapes is even more weakened. Using opening books for Go AI and that LFR has still a similar prediction accuracy in the first game phase (see Figure 3(a)) does not justify the preference of one of the other algorithms.

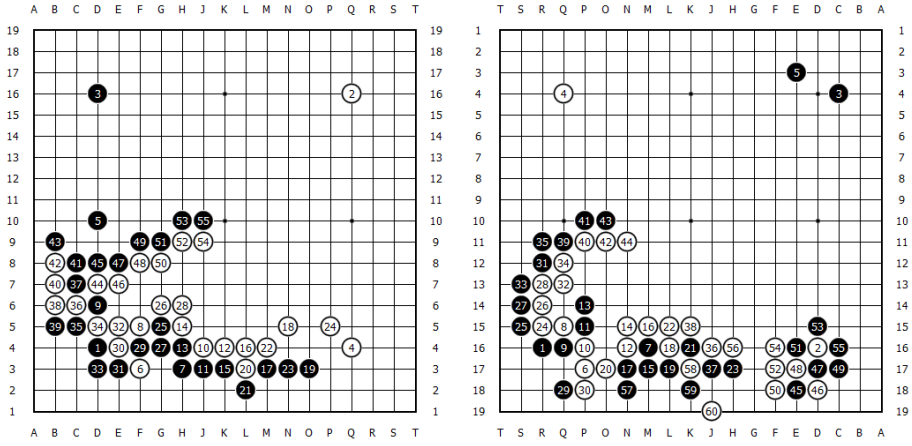
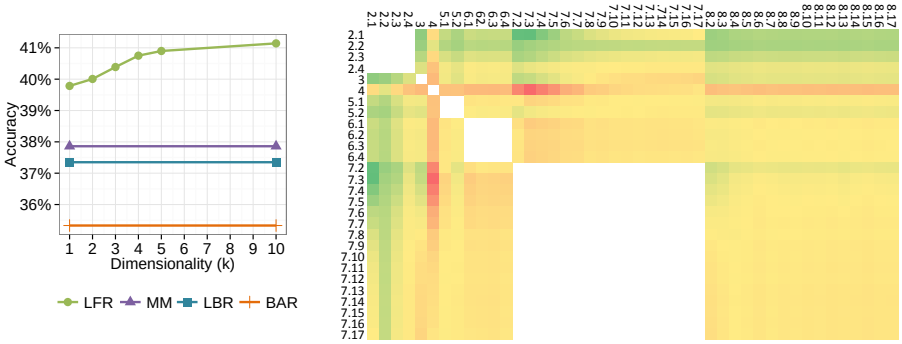


Fig. 4. On the left side are the first moves of a game played between two artificial players using the LFR1 always choosing the move with highest ranking. By means of comparison, the first moves of a game played between two of the ten strongest players on the KGS Go Server are shown on the right side.

On the left side, Figure 4 shows a game of Go between two LFR1 predictors which are always choosing the most likely action. The right side shows the first moves of a game played between two very strong players who were ranked within the top 10 of the Go GKS Server. At first glance, both games look very similar. On a closer look, the first moves are indeed almost the same. However, from move 10 on, LFR strongly prefers moves close to the moves made before and never takes the initiative by placing a stone somewhere else as seen in the game between the human players. The reason is simple: LFR is a move predictor optimized for accuracy. As one can see, in most cases a move is made close

to the last moves. Thus, it would be unreasonable to do these kind of moves. Nonetheless, this is exactly the reason why a move predictor alone is not a strong Go player. Anyways, it is very surprising how similar these games are.



(a) Move prediction accuracy depending on the dimension k.

(b) Feature interaction heat map learned on 10,000 games with LFR1 without shapes. Each intersection shows the influence of the interaction of two features. Red values have the worst, green the best positive influence.

Fig. 5. Influence of the dimensionality and the feature interactions.

The advantage of our model is that the received feature interaction weights also give an insight into Go and the importance of each feature. The main idea of combining features was that combinations of features might give more information. For instance, a feature appearing alone might indicate a bad move, but in interaction with another feature it might indicate a good move or vice versa. Unfortunately, restricting only to the non-shape features, an example of this kind of feature was not found. Nonetheless, the heat map in Figure 5(b) has exposed some interesting facts. Unsurprisingly, feature 4 (self-atari) indicates bad moves and feature group 2 (capture) indicates good moves. Feature groups 7 and 8 (distance to previous moves) has some kind of reinforcing effects. Feature values of moves close to the previous moves have a stronger impact than moves further away. So feature group 2 is a better feature for moves close to the last move. Furthermore, feature group 4 is a worse feature for these moves. A possible explanation for this observation is that a player is more aware of his actual area of interest. Additionally, if he decides not to do a move that has a positive feature but places stones in another part of the board, this could indicate that the move is probably not good.

6 Conclusion

This work has introduced a model for the move prediction problem of Go which is able to model interactions between features in an efficient way. The Latent Factor Ranking is not only easy to implement but learning can also be done online and hence does not have memory issues like MM. Finally, experiments have demonstrated the move prediction quality of LFR and how it can be used to gain insights into used features.

For future research interactions between more than two features could be of interest as well as user-specific predictions and folding in informations gained during a game.

References

1. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo Planning. Machine Learning ECML 2006 **4212** (2006) 282–293
2. Gelly, S., Wang, Y.: Exploration exploitation in Go: UCT for Monte-Carlo Go. In: NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop, Canada (December 2006)
3. Stern, D., Herbrich, R., Graepel, T.: Bayesian Pattern Ranking for Move Prediction in the Game of Go. In: ICML '06: Proceedings of the 23rd international conference on Machine learning, New York, NY, USA, ACM Press (2006) 873–880
4. Wistuba, M., Schaefers, L., Platzner, M.: Comparison of Bayesian Move Prediction Systems for Computer Go. In: CIG, IEEE (2012) 91–99
5. Rendle, S.: Factorization Machines. In: Data Mining (ICDM), 2010 IEEE 10th International Conference on. (2010) 995–1000
6. Werf, E., Uiterwijk, J., Postma, E., Herik, J.: Local Move Prediction in Go. In Schaeffer, J., Müller, M., Björnsson, Y., eds.: Computers and Games. Volume 2883 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2003) 393–412
7. Sutskever, I., Nair, V.: Mimicking Go Experts with Convolutional Neural Networks. In Kůrková, V., Neruda, R., Koutník, J., eds.: Artificial Neural Networks - ICANN 2008. Volume 5164 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2008) 101–110
8. Coulom, R.: Computing Elo Ratings of Move Patterns in the Game of Go. ICGA Journal **30**(4) (December 2007) 198–208
9. Araki, N., Yoshida, K., Tsuruoka, Y., Tsujii, J.: Move Prediction in Go with the Maximum Entropy Method. In: Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on. (2007) 189–195
10. Müller, M.: Computer Go. Artificial Intelligence **134** (2002) 145–179
11. Lichtenstein, D., Sipser, M.: GO Is Polynomial-Space Hard. J. ACM **27**(2) (April 1980) 393–401
12. Crâşmaru, M., Tromp, J.: Ladders are PSPACE-Complete. In Marsland, T., Frank, I., eds.: Computers and Games. Volume 2063 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2001) 241–249
13. Weng, R.C., Lin, C.J.: A Bayesian Approximation Method for Online Ranking. Journal of Machine Learning Research **12** (2011) 267–300