Determining Player Skill in the Game of Go with Deep Neural Networks

Josef Moudřík[1(✉)] and Roman Neruda[2]

[1] Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, Prague, Czech Republic
j.moudrik@gmail.com

[2] Institute of Computer Science, The Czech Academy of Sciences, Pod Vodárenskou veží 2, Prague, Czech Republic
roman@cs.cas.cz

Abstract. The game of Go has recently been an exuberant topic for AI research, mainly due to advances in Go playing software. Here, we present an application of deep neural networks aiming to improve the experience of humans playing the game of Go online. We have trained a deep convolutional network on 188,700 Go game records to classify players into three categories based on their skill. The method has a very good accuracy of 71.5 % when classifying the skill from a single position, and 77.9 % when aggregating predictions from one game. The performance and low amount of information needed allow for a much faster convergence to true rank on online Go servers, improving user experience for new-coming players. The method will be experimentally deployed on the Online Go Server (OGS).

Keywords: Computer Go · Machine learning · Board games · Skill assessment · Deep neural networks

1 Introduction

Computer Go is a field which mainly focuses on developing programs for playing the game. In this work, we focus on analyzing existing game records using deep neural networks with the aim to improve current ranking systems on online Go servers. This allows the servers to offer better suited opponents to the users, thus enabling them to have a better gaming experience.

Deep neural networks are currently a very hot topic of research, radically changing hard fields such as computer vision, or natural language processing [1,9,10,19]. The boom in deep architectures is allowed by abundance of data, graphical processing units (GPUs) with huge computational power and smart models. As there is probably much room for improvement in all three factors, we can only expect such models to proliferate.

Go is a two-player full-information board game played on a square grid (usually 19×19 lines) with black and white stones; the goal of the game is to control

C. Martín-Vide et al. (Eds.): TPNC 2016, LNCS 10071, pp. 188–195, 2016.
DOI: 10.1007/978-3-319-49001-4_15

the board by means of surrounded territory and captured enemy stones. Go has traditionally been an excellent testing ground for artificial intelligence, as the game has been considered to be extremely challenging. Only recently has the AI surpassed humans, when Google's AlphaGo beat world's top professional player Lee Se-dol [20], causing much surprise among most computer Go researchers.

In the game of Go, the strength of amateur players is measured by *kyu* (student) and *dan* (master) ranks. The kyu ranks decrease from about 25 kyu (an absolute beginner) to 1 kyu (a fairly strong player), the scale continues by dan ranks, 1 dan (somewhat stronger than 1 kyu), to 6 dan (a very strong player). Often, the ranks are modelled using an underlying continuous quantity called *rating*.

A rating system is a way to numerically express player's strength. In online Go servers, this is necessary so that users can be matched to other users with similar skill and so that users' skills can be compared. Often, mathematical models such as ELO [5] or Bayesian approaches [7] are used. After a game, these systems take information about the result (win/loss/draw) and modify the user's rating. As draws are quite rare in Go, this makes up for only a slightly more than 1 bit of information per game. As a result, the current systems take nontrivial number of games to converge to true players' rating, causing problems for new-coming players, often forcing them to play games with opponents of unfitting skill.

For instance, after the AlphaGo vs. Lee Se-dol match, Online Go Server (OGS) [17] had a large influx of beginners, who often chose incorrect rank as their baseline, causing some frustration and work for intervening administrators. A similar problem with ranking is also caused by players who are much stronger than they declare. Such players often cause dismay amongst similarly-ranked (yet weaker) players, and it is often again a task for administrator to find such players and correct their rating.

Obviously, some more information about the players' skill would be very helpful and one of possible source are the games themselves. In our previous work [2,11,14,15], we pioneered a machine-learning approach to predict the player's strength and playing style from game records. Our methods used features based on various statistics (patterns played, etc.), but these required a sample of at least 10 games to be reasonably accurate, which is still impractical to tackle the rating problem. This paper presents a new approach to this problem.

In this work, we set out to investigate the possibilities of estimating the strength of a player using as little information as possible, the natural starting point being a single position. Such small sample sizes naturally call for smarter utilization of the information; for this, we employ deep neural networks. The proposed methodology is being experimentally deployed on OGS.

The rest of the paper is organized as follows. Section 2 presents the dataset, preprocessing and augmentation used. Section 3 gives information about the architecture of the deep neural network used. Section 4 gives an overview about the experiments we have performed, and reflects on the results. The paper is concluded by Sect. 5, which sums up the work, discusses the application and proposes future work.

2 Dataset

The dataset was created from 188,700 public game records from March 2015 to February 2016. The games were downloaded from OGS. Because OGS is a relatively new server, it does not have many strong players (also not many games played by them). Therefore, we only chose games in which players are between 2 dan and 25 kyu. Of all the games, 20,000 were used as a validation set (used for tuning hyper-parameters of the model) and 20,000 reserved for testing (used for final evaluation). Each game is viewed as a sequence of positions.

The dataset consists of 3,426,489 pairs (X, y). Each sample X (an encoding of a position and 4 last moves) is classified into 3 classes y based on the skill of the player. The three classes used are: strong players (2 dan–7 kyu), intermediate players (8 kyu–16 kyu) and weak players (17 kyu–25 kyu). Three classes were chosen instead of direct regression (or a classification with more classes) because the problem is quite hard and more precise methods would not work robustly enough given the low amount of information from a single position.

In the following paragraphs, we describe the steps we took to process the game records in order to create the dataset for the deep neural network.

Planes: Every single sample X was encoded as 13 binary planes, each of size 19×19. A point on each plane gives information about the particular intersection. Out of the 13 planes, 4 encode the number of liberties (an empty intersection next to a group of stones) for player whose turn it is (the planes being 1, 2, 3 or 4 and more liberties), 4 encode liberties for opponent stones in the same fashion. One plane encodes positions of empty intersections and the last four planes show the last move, the second-last move, the third-last move and the fourth-last move. The planes used were proposed by D. Schmicker [18].

Subsampling: During training, the network tended to memorize positions from the games, causing serious overfitting and poor performance on unseen data. To fight this, we increased the number of games in a dataset and subsampled the positions: every fifth position (plus previous four moves) is taken from each game (randomly). For instance, from a game of 250 moves we get 50 pairs (X, y) on average. Together with data augmentation, this prevented overfitting very efficiently.

Data Augmentation: To prevent overfitting, we devised the following simple data-augmenting strategy. During training, each sample from each mini-batch was randomly transformed into one of its 8 symmetries; this is possible since board in the game of Go is symmetrical under reflection and rotation. This helps the network to build representations that are invariant to symmetry and reflection, thus improving generalization.

Equalization and Shuffling: In the training data, we made sure that all the classes have precisely the same number of examples. This makes comparison of different models easy, as it makes sure that the network is not exploiting uneven distribution of the targets.

Finally, all the training examples were shuffled well, so that all batches in the learning process have roughly the same distribution of targets, the motivation being to improve the gradient in the batches. In our experience, uneven distributions of classes among many consecutive mini-batches cause the network to exploit the irregularities, leading to poor performance.

3 Architecture and Training

The network has 4 convolutional layers followed by two fully connected layers of 128 neurons and finally a 3-way softmax layer. The first convolutional layer has filters of size 5×5, the remaining layers have filters of size 3×3. All the neurons in the network were activated using non-saturating nonlinearity $f(x) = \max(x, 0)$, the so-called *rectified linear units* (ReLUs) [16]. Compared to saturating non-linear activations, such as tanh, ReLU units speed up the convergence considerably. Every convolutional layer operated on the full 19×19 input, outputs of the network were padded by zeros again to 19×19, the convolutions were applied with a stride of 1. The number of filters in the network is 512 for the first layer and 128 for all the other layers. Batch normalization [8] was used in all the convolutional layers. Batch normalization normalizes layer inputs as a part of the model, allowing higher learning rates while acting as a regularizer.

All weights in the network were initialized with normalized initialization as described in [6]. In total, the network has 6,985,475 parameters, majority between the last convolutional and the first fully connected layer.

The best generalization and performance on the validation set was achieved when we trained the network in two phases, see Fig. 2. For two epochs, we had used stochastic gradient descent (SGD) with Nesterov momentum [3] of 0.8, learning rate of 0.01, and mini-batches of size 32. After this, the network was fine-tuned for 4 epochs using SGD without momentum, learning rate of 0.01 and mini-batches of size 128.

The loss function used was the categorical cross-entropy:

$$L(t, p) = -\sum_i t[i] log(p[i]),$$

where t is one-hot vector specifying the true label and p is the probability predicted by the softmax. The categorical cross-entropy basically penalizes confident (high $p[i]$) predictions that are wrong, and the resulting numerical outputs can thus be interpreted as class membership confidence.

Network was trained for 6 epochs and in total, the learning took 2 days on NVIDIA GeForce GTX 580 with 1.5 GB of memory. To design, test and train the model, we have used the Keras deep-learning framework [4].

4 Experiments and Results

We have performed several experiments, the baseline being the prediction from a single position. Additionally, we have investigated possibilities for aggregating

Table 1. Summary of results. All accuracies reported are measured on testing data. The last column reports percentage of examples where the correct label was within top two classes. The last three rows show accuracies of aggregated predictions from a single game. Label *Augmented* indicates that prediction was made from 8 symmetrical board positions and averaged, instead of simply predicting from the position alone. Label *Cropped* indicates that only predictions from move 30 on were taken.

Model	Accuracy	Accuracy (Top-2)
Single position	71.5 %	94.6 %
Single position (Augmented)	72.5 %	94.9 %
Aggregated per game, mode (Augmented)	76.8 %	N/A
Aggregated per game, sum (Augmented)	77.1 %	96.4 %
Aggregated per game, sum (Augmented, Cropped)	77.7 %	96.7 %
Aggregated per game, sum (Augmented, Weighted)	77.9 %	96.8 %

multiple predictions. Finally, we investigated the dependency of accuracy on move number. All the experiments are described in paragraphs below. The results are summarized in Table 1.

Predictions from Single Position: Firstly, we measured the baseline accuracy of prediction for single positions, which is 71.5 % on testing data, the confusion matrix can be seen in Fig. 1. We further improved this accuracy by a novel trick. Since we train the network on augmented positions (8 symmetries), the network should be roughly invariant under reflection and rotation. Therefore, it makes sense to present the network with all 8 symmetrical copies of a single position and average the resulting predictions. This improves the accuracy by roughly 1 %. The intuition why this works is that the symmetrical positions essentially form an ensemble, averaging out outliers.

Aggregating Predictions: Another interesting thing to study is how to improve accuracy by aggregating multiple predictions. A natural way is to sum up predictions from individual positions and then take the class with the maximal sum (accuracy of 77.1 %). Another possibility we have tried is to take the most frequent (mode) class (accuracy of 76.8 %).

Accuracy by Move: Next, we investigated the dependence of accuracy on the move number, as shown in Fig. 3. The results indicate that the accuracy is bad at the beginning of the game, but improves rapidly afterwards. This is very understandable, as the games of Go often have very similar opening sequences (e.g. players usually start by playing into the corners). The figure also shows size of data samples at each move, as different games have different lengths. This also explains why the accuracy curve jumps near the end, this is caused by the low number of samples. The figure naturally suggests a way to improve the aggregation, and that is either to crop off moves at the beginning of the game (improvement in accuracy of 0.6 %, see Table 1), or — better — to weight the

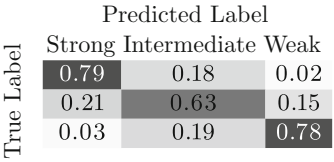

Fig. 1. Normalized confusion matrix of predictions on testing data.

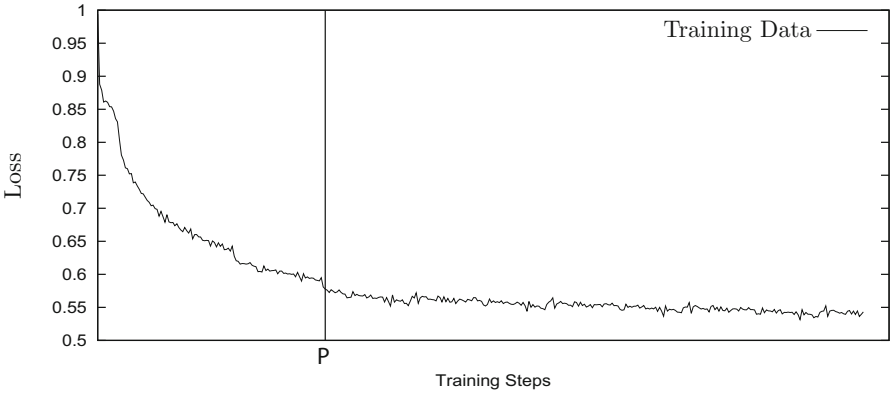

Fig. 2. Evolution of Loss function during training. The P point marks the place where the fine tuning phase of training was started, see Sect. 3.

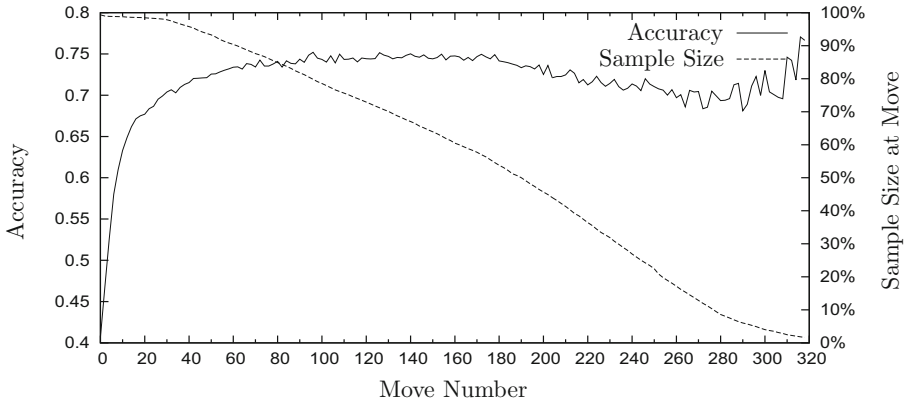

Fig. 3. Dependency of accuracy on the move number (measured on validation set). The dashed line shows sample size at given move in testing data — some games end earlier than others.

predictions proportionally to their accuracy (improvement of 0.8 % in accuracy). As this weighting can be considered hyper-parameter tuning, the weights (as well as Fig. 3) were computed using the validation set.

5 Conclusions and Future Work

In this work, we have proposed a novel methodology for information-efficient way of predicting player strength using deep neural networks. In comparison with other approaches, much less information is needed. This is naturally balanced by a relatively coarse discretization of the target domain (strength is divided into 3 classes).

The methodology will be experimentally deployed on Online Go Server, aiming to improve convergence of the rating system. The precise way to incorporate the predicted class into the rating system is yet to be determined. A general idea on how to do this is to use the predicted classes as a prior for the underlying rating model. As the number of games in the model increases, the importance of the prior would naturally decrease.

The results hint that the performance of the network is touching the limit given by the amount of information provided. A natural next step would be to extend the system to use information from one whole game. Recurrent neural networks could be the ideal tool for this; we plan to investigate this in the near future.

Resources. To promote further research and to ease reproducibility of this study, we have published the game records, datasets and the model file (including the weights of the best network we have found). All the files can be found at [13].

The dataset processing was performed using our open-source toolkit, see [12].

Acknowledgments. Authors would like to thank Martin Pilát for valuable discussions. This research has been partially supported by the Czech Science Foundation project no. P103-15-19877S. J. Moudřík has been supported by the Charles University Grant Agency project no. 364015 and by SVV project no. 260 333.

References

1. Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., Collins, M.: Globally normalized transition-based neural networks. arXiv preprint (2016). arXiv:1603.06042
2. Baudiš, P., Moudřík, J.: On move pattern trends in a large go games corpus. Arxiv, CoRR. http://arXiv.org/abs/1209.5251
3. Bengio, Y., Boulanger-Lewandowski, N., Pascanu, R.: Advances in optimizing recurrent networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 8624–8628. IEEE (2013)
4. Chollet, F.: Keras (2015). https://github.com/fchollet/keras
5. Elo, A.E.: The Rating of Chessplayers, Past and Present. Arco, New York (1978)
6. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. Aistats **9**, 249–256 (2010)
7. Herbrich, R., Minka, T., Graepel, T.: Trueskill (tm): A bayesian skill rating system. In: Advances in Neural Information Processing Systems, pp. 569–576 (2006)
8. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint (2015). arXiv:1502.03167

9. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
10. Mikolov, T., Dean, J.: Distributed representations of words and phrases and their compositionality. Adv. Neural Inform. Process. Syst. **26**, 3111–3119 (2013)
11. Moudřík, J.: Meta-learning methods for analyzing Go playing trends. Master's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic (2013). http://www.j2m.cz/jm/master_thesis.pdf
12. Moudřík, J.: deep-go-wrap: Toolkit designed to ease development of your deep neural network models for the game of Go. (2016). https://github.com/jmoudrik/deep-go-wrap
13. Moudřík, J.: Source code and resources (2016). http://j2m.cz/~jm/archive/2016-08-02/
14. Moudřík, J., Baudiš, P., Neruda, R.: Evaluating go game records for prediction of player attributes. In: 2015 IEEE Conference on Computational Intelligence and Games (CIG), pp. 162–168. IEEE (2015)
15. Moudřík, J., Neruda, R.: Evolving non-linear stacking ensembles for prediction of go player attributes. In: 2015 IEEE Symposium Series on Computational Intelligence, pp. 1673–1680. IEEE (2015)
16. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp. 807–814 (2010)
17. OGS: Online Go Server (OGS) (2016). http://online-go.com
18. Schmicker, D.: Planes for move prediction, maillist entry. http://computer-go.org/pipermail/computer-go/2015-December/008324.html
19. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
20. Wikipedia: Alphago versus lee sedol – wikipedia, the free encyclopedia (2016). https://en.wikipedia.org/w/index.php?title=AlphaGo_versus_Lee_Sedol&oldid=732449782, Accessed 12 Sep 2016