Apply Different Neural Networks in Evaluation Function of Computer Chess

He Ying
yhe@ntu.edu.sg
School of Computer Science and Engineering

***Abstract* -** This project aims at showing the performance of different architectures of neural network that acts as the statistic evaluation function of the computer chess engine. With training at 6 million distinct positions of games from ChessBase 16 and labeled by Stockfish 13, our neural networks get more than 90% top-2 accuracy in the 7-class classification task.

Keywords – deep learning, computer chess, evaluation function

1 BACKGROUND

Computer chess is one of the traditional computer science research fields regarding AI. Among all the components of a computer chess engine, the evaluation function is one of the most crucial ones. The function takes the position as input and gives a numeric representation as result to indicate which side is leading at the current position. Two methodologies have been used to develop the evaluation function. The traditional evaluator evaluates the position using numerous hand-crafted features with long-time tuning. While the neural-network-based evaluator trains and uses one or serval NNs as the evaluation function, which needs a shorter adjustment time and more dominant performant than a hand-craft evaluator.

2 DATA PREPARATION AND LABELING

2.1 LABELING WITH STOCKFISH 13

We extract games given by the strong players whose ratings > 2200 from the commercial chess database, ChessBase 16. We get 29,064,346 distinct positions from the above games. Under the time and computation resources constraint, we randomly extract 6 million positions in the set.

Then, these positions are label by a strong NNUE (Efficiently Updatable Neural Network) computer chess engine, Stockfish 13. In order to balance the accuracy of the labels and the time-consuming of the labeling process, we choose depth 15 as the restriction of Stockfish 13 searching. Under such depth limitation, our computer has the capability to label approximately 10,000 positions per hour.

2.2 INTERPRETATION OF LABELS

2.2.1 Change numeric labels to categorical

The label given by the Stockfish 13 is a float number defined as the centipawn score of the advantage of the white side.

For simplification, as the centipawn is not suitable for the situation of checkmate, we simply convert the checkmate status to ±100.

For the classification problem, we need to manually classify the centipawn score into 7 classes based on our prior chess knowledge. The following table shows the roles of our processing.

Number	Pawn score	Category
0	5 <= s	White is winning (ww)
1	3 <= s < 5	White has a decisive advantage (wa)
2	1 <= s < 3	White is better (wb)
3	-1 < s < 1	the position is equal (eq)
4	-3 < s <= -1	Black is better (bb)
5	-5 < s <= -3	Black has a decisive advantage (ba)
6	s <= -5	Black is winning (bw)

2.2.2 Data distribution

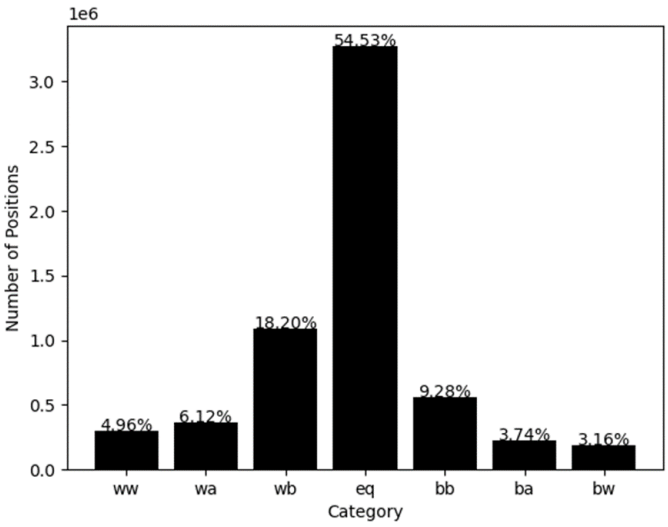

Figure 1 Category Distribution

It is easy to understand that most of the positions are equal positions for both sides. And, 'ww' or 'bw' will be the smallest category.

2.2.3 Data Instability

It is worth mentioning the inner instability in our data. We experimentally use Stockfish 12 which was released 8 months before Stockfish 13 to label the same positions. We compare the classes given by the two engines and find out that the average disagreement between these two engines is more than one class (i.e., the average difference of the pawn scores given by the two engines is approximately 2.7 and we use 2 as the boundary of our category). Such observation shows that there is some inner- fluctuation in our label. Hence, for a more ideal comparison, we decide to use top-2 accuracy.

In all our tasks, we use 98% (i.e., 5,880,000) of the data as the training set and 1% (i.e., 60,000) for the development set and test set.

2.3 NECESSARILY OF CONVERSION

Since the extracted distinct positions are stored in the form of FEN (a.k.a. Forsyth-Edwards Notation), it is necessary to convert them in the form that could be used to compute by NNs. In this paper, they will be represented in various forms of vectors (tensors) for different NNs according to architectures.

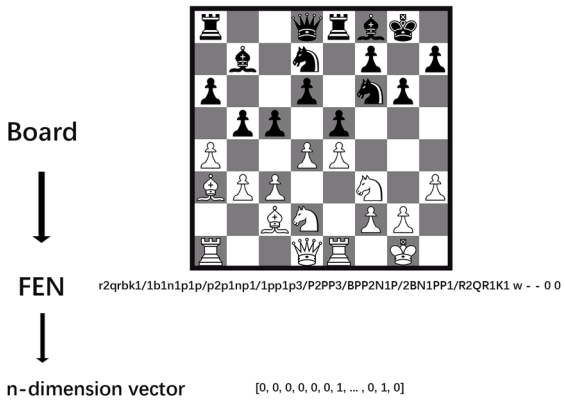

Figure 2 Conversion of Board

3 TRAINING OF DIFFERENT ARCHITECTURES

3.1 TRIVIAL FULLY CONNECTED NN

3.1.1 FEN-to-vector Conversion

We convert the position to the bitboard representation, which is one of the most intuitive and trivial methods used in computer chess. The bit inside the 64-dimension vector implies the existence of the piece on a certain square.

There is a total of 12 types of pieces of the two sides, which will give us a 768-dimension vector. We understand that some additional features like the moving side and the castling right also important for the evaluation. Eventually, we get a 773-dimension vector as our representation.

3.1.2 Training

The NN here refers to trivial NN simply because we use the most straightforward way to process the positions and to construct the NN. And the result given by the trivial NN will be used as the benchmark for the other architectures.

We use ReLU as the activation function and add 0.5, 0.3, 0.1 probability of dropout between each layer.

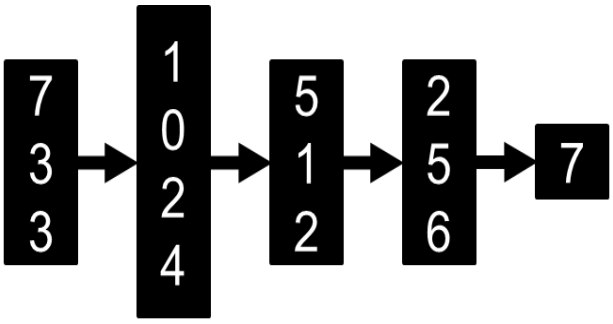

Figure 3 Architecture of Trivial NN

We use the following hyperparameters to train the NN:

Optimizer	Adam
Initial LR	0.001
Loss function	Cross Entropy Loss
Scheduler	Exponential 0.99
Training epoch	100

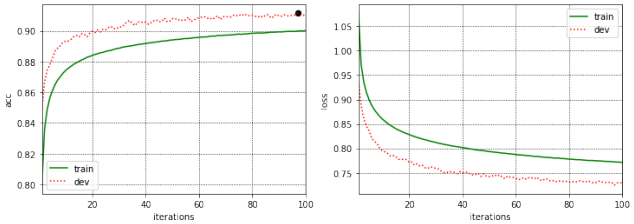

Figure 4 Result of Trivial NN

After training, we get 90.90% top-2 accuracy in the test set.

3.2 HAND-CRAFT-FEATURE FULLY CONNECTED NN

3.2.1 FEN-to-vector Conversion

Before some end-to-end solutions of the evaluator are proposed, human prior knowledge is still very important in the deep learning of computer chess. Giraffe [1] uses a 3-layer neural network that acts as its evaluator. The training instance for Giraffe converses to a 363-dimension vector. To be specific, the first 10 elements of the vector contain material information like the number of each piece. The second 225 elements contain the piece information such as the location, safety count, and parallel or oblique mobility count of each piece. The last 128 elements are used for the square information, which are attack and defense maps in the form of an 8*8 chessboard. Noted that, all the values discussed above have been normalized, which are beneficial for the propagation algorithm.

3.2.2 Training

In Giraffe, the author implements the first layer not fully-connected on purpose. He proposes the idea that data from different modalities mix at a low level will offer no benefit and makes overfitting more often happen. Although modern computer scientists tend to use the dropout method which is a simpler and more suitable technique to handle the proposed overfitted problem, we will still follow the Giraffe in our implementation.

Also, we use the ReLU as the activation function following the Giraffe.

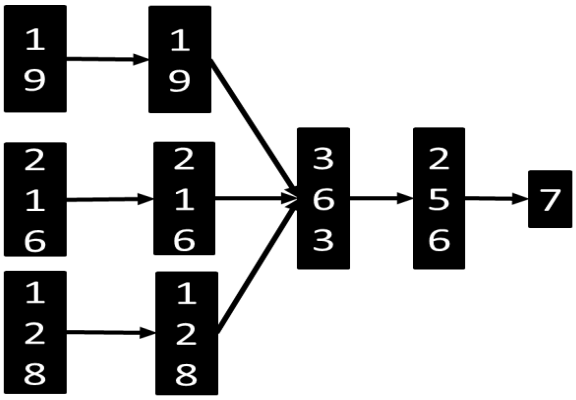

Figure 5 Architecture of Hand-craft-feature Fully Connected NN

We use the following hyperparameters to train the NN:

Optimizer	Adam
Initial LR	0.001
Loss function	Cross Entropy Loss
Scheduler	Exponential 0.98
Training epoch	60

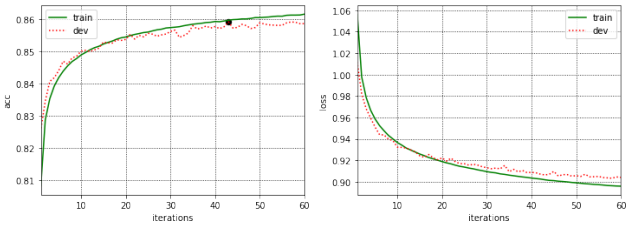

Figure 5 Result of Hand-craft-feature NN

We get 86.11% top-2 accuracy in the test set.

3.3 AUTOENCODER NN

3.3.1 FEN-to-vector Conversion

It is easy to notice that the 773-dimension vector we gain previously is quite sparse as most of the pieces will only exist in one or two squares on the chessboard. Such a high-dimension and sparse vector is hard for the NN to propagate and therefore may hard its performance. So, we propose an autoencoder [2] to try to extract some high-level features before the task in order to meet the dimension reduction purpose.

3.3.2 Training of Autoencoder (AE)

Unsupervised training will be applied to the AE for reducing the above 773 dimensions sparse vector to the 100 dimensions vector.

The AE consists of an encoder part and a decoder part. We find out that using hyperbolic tangent instead of ReLU as our activation function will enhance the ability of the AE. And we use the sigmoid function in the output layer to restrict the output in the range of 0 to 1.

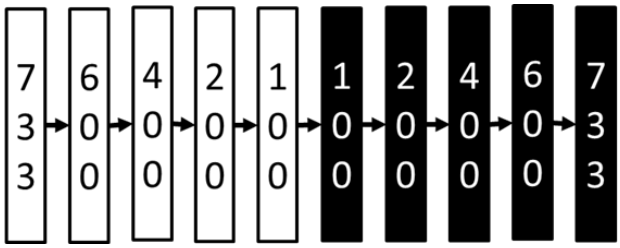

Figure 6 Architecture of AE

We use the following hyperparameters to train the autoencoder:

Optimizer	Adam
Initial LR	0.001
Loss function	Cross Entropy Loss
Scheduler	Cosine Annealing (T_max = 4, eta_min =

	1e-5)
Training epoch	60

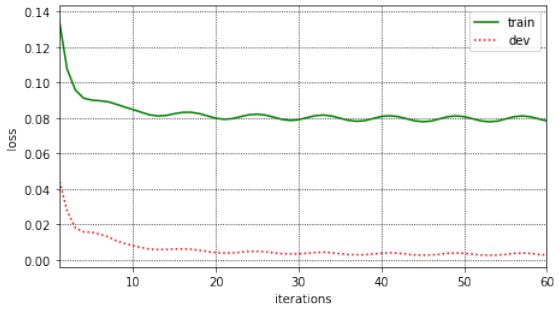

Figure 7 Result of AE

The minimum loss we gain is 0.0025, which is an acceptable value for the following tasks.

3.3.3 Training

Then, we will preserve the encoder part as the initialization and connect serval layers to interpret the 100-dimension vector.

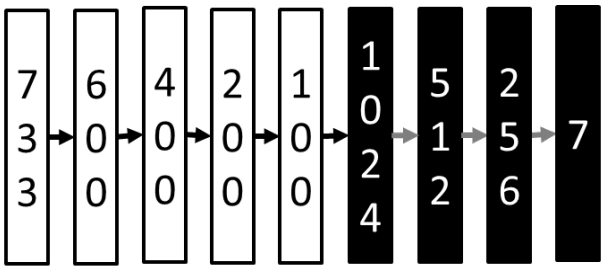

Figure 7 Architecture of AE Classifier

We use the following hyperparameters to train the NN:

Optimizer	Adam
Initial LR	0.001
Loss function	Cross Entropy Loss
Scheduler	Cosine Annealing (T_max = 5, eta_min = 1e-5)
Training epoch	150

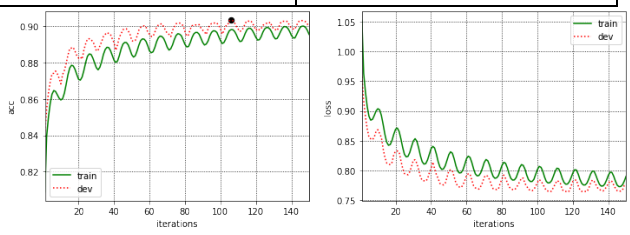

Figure 8 Result of AE Classifier

We get 89.99% top-2 accuracy in the test set.

4 CONCLUSION

In this project, we reasonably use 3 different architectures for the classification tasks. The trivial NN has no adjustment rather than the bitboard representation and no special implementation in the architecture. The two remaining NNs are focus on dimension reduction. The hand-craft-feature NN uses some prior knowledge to extract the features. And, the AE uses unsupervised learning to extract the feature vector before training the classifier.

Both the trivial NN and the AE get 90% top-2 accuracy after training, which is better than the hand-craft-feature NN. It shows that the neural network can learn some features that better than the human understanding of the chess position evaluation task.

5 FUTURE DEVELOPMENT

5.1 UNDERSTANDING NN

One of the criticisms and concerns toward deep learning is that it is still a black box to some extent. Neither loss functions nor optimization functions are mathematically well-defined or even humanly interpretable.

There have been many efforts to try to visualize or mathematically prove the mechanism of the neural network [3]. In this project, the high-level 100-dimension vector given by the autoencoder lacks human understanding. As human players are more familiar with the combination of strategies from the position, such an abstract vector could hardly help human players. So, it is possible to interpret them by observation the correspondence between the strategies and the activated bits of the vectors. The same approach could be applied to CNN in order to understand how the kernels react to the input chess positions.

5.2 OTHER ARCHITECTURES

Along with the prosperity of deep learning, numerous interesting architectures and operators have been proposed and get SOTA performance in specific fields.

With a longer time and more powerful computational resources, we may try to use some other architectures that are applied in other fields like natural language processing (NLP) or computer vision (CV). Although there are no promising theories that such NNs will be also valid in the computer chess evaluation function, it is noticeable that there are some kinds of similarities between the computer chess evaluation and the NLP and CV tasks. For example, CNN was proposed to reduce the extremely large dimension of high-resolution photos in the CV dataset. And,

it is possible to consider the chess position as multi-channel color images of 8x8 pixels in size.

REFERENCES

[1] M. Lai, "Giraffe: Using Deep Reinforcement Learning to Play Chess", Imperial College London, September 2015.

[2] Eli (Omid) David, Nathan S. Netanyahu, and Lior Wolf, “DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess”, International Conference on Artificial Neural Networks (ICANN), Springer LNCS, Vol. 9887, pp. 88–96, Barcelona, Spain, 2016.http://dx.doi.org/10.1007/978-3-319-44781-0_11

[3] Matthew D. Zeilerzeiler, Rob Fergus, “Visualizing and Understanding Convolutional Networks”, 2013.