1 Chess quantification evaluation

This text is a documentation of the chess quantification evaluation project. It also includes thoughts that changed the way to reach the goal.

1.1 Quantification in chess

The aim is to predict the quality of a chess match with an ELO-rating. Averaged over a lot of games a chess player should get his ELO- or DWZ-rating. With the information, that one game has much less rating than the ELO of the player, the player knows, that that game was probably bad and this game should be improved.

1.2 Preprocessing

1.2.1 Initial thoughts

As an input there should be the game notation encoded to numbers. Photos of the board positions are not useful, as photos are quite much data with all their pixels and one only needs to know, which figure is in which box.
There are three possible encodings, that could come into mind:

- Transform notations from the game formular into numbers, for example $1.e4$ for moving the white pawn in the first move from $e2$ to $e4$, could be tranformed to 54 for $a = 1$, $b = 2$,...,$e = 5$ and as the second digid use the written number. Additionally the figures with names could also get an encoding, such that we would write for e the number 154 (for Pawn=1) and adapted for example for the king $Ke2$ is the number 652. The question would be, if that information can represent the complexity of the game well enough. Also it is the question, if further analysing can be done. For example after getting the information, that one played in a game bad, it would be really helpful to get more detailed information on the sort of the error.

- For that one could think of representing every move of the party formular, by the actual board. Meaning we have $A \in \mathbb{Z}^{8\times 8}$ with A_{ij} has the information, which figure is on field (i, j). One could encode black figures from -6 to -1, 0 for no figure and 1 to 6 for white figures. There could be the problem, that in the neural network the figure is transforming into another figure by reducing the value of A_{ij}.

- Now the final idea is from $https://erikbern.com/2014/11/29/deep-learning-for-chess.html$:
For every move t of the chessboard we have a matrix $A^{(t)} \in \mathbb{N}^{8\times 8\times 12}$. For all $i \in \{1, ..., 12\}$ the matrix $A^{(t)}_{\cdot,\cdot,i}$ represents one sort of figure for one color. For example we have $A^{(t)}_{k,l,i} = 0$ for all, but one positions: Only on $e1$ the white king is at begin of the game.

1.2.2 Preparing of the data set

In a first step the data sets are prepared. Meaning: We write for every match the party notation, that only includes the moves into one file. (With 100 games we get with that 100 files.) We further write one file for the results and one file for the ELOs of both players. This work is done with *read.py*.

1.2.3 The training set

First it seems easy to get a suitable training set. One can just download a dataset from $https://database.lichess.org/$. But the question is, what can we do with this data set?
We have to transform the sparse notation of the party formular into positions in our matrices. On the game notation it is only given, where the figure moves, but not from where it comes, because this is in most cases clear from the end position. Only if it is not clear, some parts of the start position are also given. That brings in all complexity of the chess game for our implementation. Formaly we do not want to check, if the chess move is a legal move, but we have to figure out, which of the figures moved there. Only the king is unique. For all other figures there could be more figures of the same type, that could move to the same position, at least later in the game.
Hence we have to implement the possible moves for every figure and also have to control, if there are figures between the old and the new position and hence the move of that figure is not allowed.
To get a control, if the moves were correct, there is a check implemented: As already said, it is easy to set the figure on the new position, but difficult to remove the figure from the old position. That is used: The program prints out „error“, if the number of figures on the board increases.
At the actual state of implementation there are 199 games tested and 16 times one figure not removed from the old position. Part of that situation happens, when there are 2 figures of the same sort, that one first think, they could move to the same position, but if one of the figures move, the king would be in chess. Hence the formular does not spend the information, where the figure comes from and it is only one chosen, which could be maybe the wrong figure. In the $200th$ game of the dataset there is an error, that crashed the model. To be more accurate that should be fixed, but to get a result in acceptable time, that problems are not fixed and that games are ignored.

1.2.4 Exact result for training and test set

First it seems really easy to get the exact results for the training and test set: One just uses the ELO number of the player for the training set as the exact result, meaning the strength one player had in a match. But a normal distribution seems to be a suitable approximation of a chess player. Here is μ the expectation and we do not know σ, which depends on the continuity of the player. But also the performance of the player in one game could depend on the oponent. For simplicity let us only consider white. Then we have a function y with:

$$y(N(\mu_{white}, \sigma^2_{white}), N(\mu_{black}, \sigma^2_{black})) \in (0, \infty).$$

Here $N(\mu, \sigma^2)$ denotes the normal distribution with expectation μ and variance σ^2. For simplicity we assume, that we just have:

$$y(N(\mu_{white}, \sigma^2_{white})).$$

If we use that as a prediction, we already expect to make an error by just using the expectation as the prediction of the game quality, as in the test set the players do not just play exactly their ELO rating every game. We can also calculate the predicted error by the formular:

$$\int_{-\infty}^{\infty} |x - \mu| f(x|\mu, \sigma^2)\, dx,$$

whith the density function:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

1.1. Lemma
The expected error is:

$$\int_{-\infty}^{\infty} |x - \mu| f(x|\mu, \sigma^2)\, dx = \frac{\sqrt{2\sigma^2}}{\pi}.$$

proof:
It holds with subsitution with $u(x) = (x - \mu)^2$, $u'(x) = 2(x - \mu)$:

$$\begin{aligned}
\int_{-\infty}^{\infty} |x - \mu| f(x|\mu, \sigma^2)\, dx &= \int_{-\infty}^{\mu} (\mu - x) \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}\, dx + \int_{-\infty}^{\mu} (x - \mu) \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}\, dx \\
&= \int_{\infty}^{0} \left(-\frac{1}{2}\right) \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-\frac{u}{2\sigma^2}}\, du + \int_{0}^{\infty} \frac{1}{2} \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-\frac{u}{2\sigma^2}}\, du \\
&= \int_{0}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-\frac{u}{2\sigma^2}}\, du \\
&= \left[-\frac{1}{\sqrt{2\pi\sigma^2}\frac{1}{2\sigma^2}} \mathrm{e}^{-\frac{\mu}{2\sigma^2}} \right]_{0}^{\infty} \\
&= \frac{2\sigma^2}{\sqrt{2\pi\sigma^2}} = \sqrt{\frac{2\sigma^2}{\pi}}.
\end{aligned}$$

□

That seems to be a plausible lower bound for our efforts, if we just take the expectation. But we know, that the expectiation is true for a lot of games. Hence we should take more games, for example 50, and take the mean ELO as the strength of the game. Then we get the problem, how we can do this:
Is it important to somehow presort that training set? Meaning can we just have a match

with white and ELO 2000 and after that white with ELO 1000? That would be far away from the normal distribution. Hence it seems plausible, that we have to sort the data and for one training example y with for example 50 games white should have the same ELO. The same thoughts can be done regarding black: Also in all that games it could be good, if black has always the same ELO.
Due to that thoughts there should be much more preprocessing to sort this data as wanted and that seems to be computationaly expensive. The computation exists, because we then need really much training data to put into the neural network.

1.2.5 Adapting the aim

As the first aim seems to be far to complex, we just try to do another aim: We want to predict with an RNN after ten moves, if white wins or loses the game. By that aim we still use the architecture and parts of the preprocessing of the bigger aim. Hence one could maybe adapt the project to the real aim.