Chess game rating

by Niko Schmidt

Chess game rating

- What?
  - Evaluation, how well both players played a game (ELO rating)
- How?
  - Deep Learning
  - Input: Party formular

    Output: rating for both players $y \in IR^2$

Why?

- Sometimes hard to see, which games were good or bad
  - Dependent on the oponent‘s strength and daily performance, the opening, …
  - A won game could be still bad played and a lost game good played

What can it be used for?

- See, which games need first improvement
  - Maybe look for similarities in your n worst games e.g.
    - Colour played
    - openings
    - ...
  - Save time (only try to find mistakes in games, in which you performed bad and not in which you played really good)

    (In statistics we should converge to our rating, but get some bad games(?))

Dataset?

- https://database.lichess.org/
  - lichess_db_standard_rated_2013-01.pgn

Architecture?

- Chess games have a different number of moves

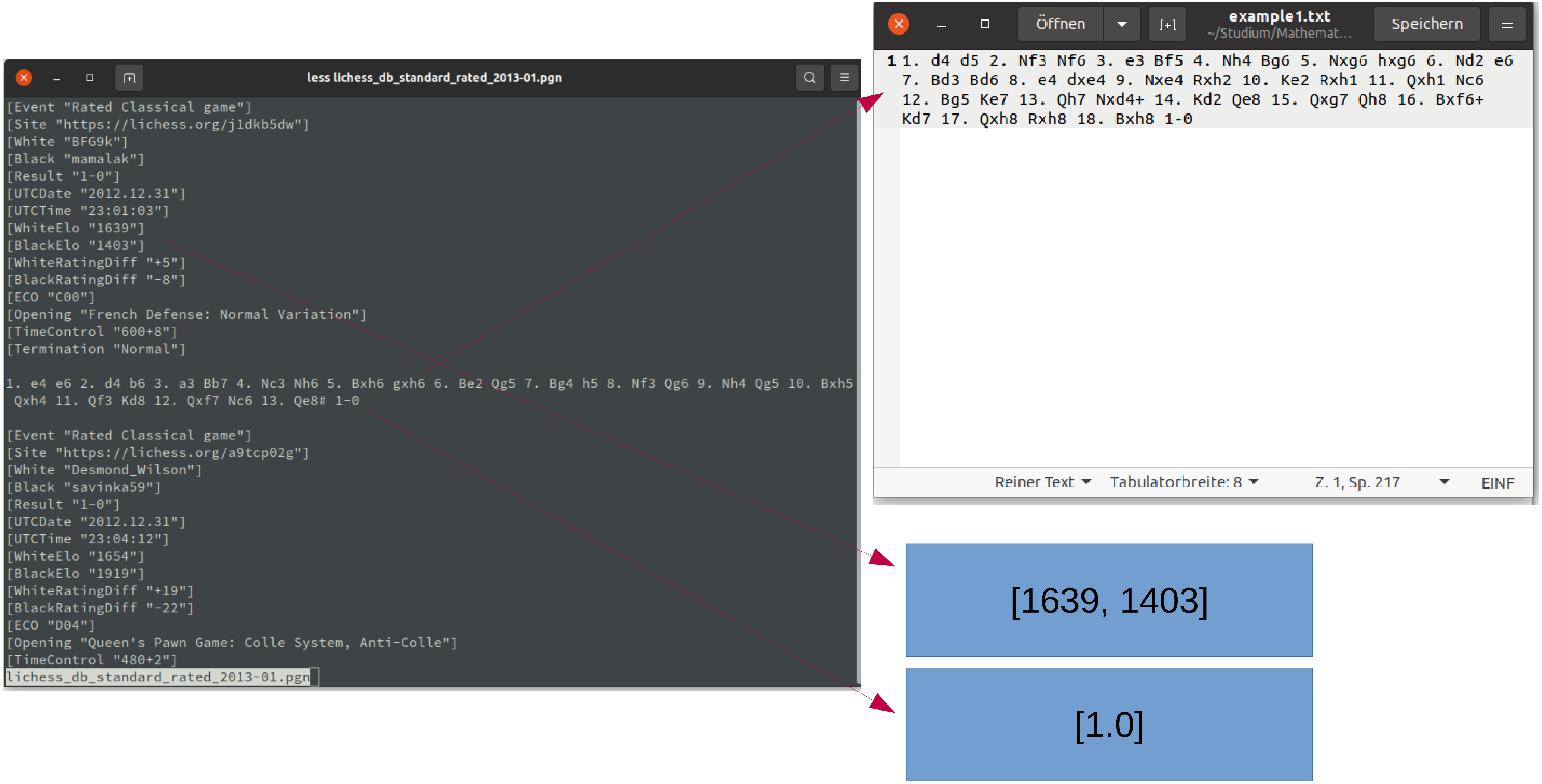
  => RNN networks would be fine.
- But for simplicity just a deep neural network

Architecture in more details

- Model in general and other DNN stuff from bakery sales prediction
  - Adapted to our problem – for example
    - Number of layers
    - Number of units
    - Activation functions

How to start

- Transform data from lichess into
  - Game notation
  - Result
  - ELO numbers
- Problem:
  - File too big to open directly



```
[Event "Rated Classical game"]
[Site "https://lichess.org/j1dkb5dw"]
[White "BFG9k"]
[Black "mamalak"]
[Result "1-0"]
[UTCDate "2012.12.31"]
[UTCTime "23:01:03"]
[WhiteElo "1639"]
[BlackElo "1403"]
[WhiteRatingDiff "+5"]
[BlackRatingDiff "-8"]
[ECO "C00"]
[Opening "French Defense: Normal Variation"]
[TimeControl "600+8"]
[Termination "Normal"]

1. e4 e6 2. d4 b6 3. a3 Bb7 4. Nc3 Nh6 5. Bxh6 gxh6 6. Be2 Qg5 7. Bg4 h5 8. Nf3 Qg6 9. Nh4 Qg5 10. Bxh5
 Qxh4 11. Qf3 Kd8 12. Qxf7 Nc6 13. Qe8# 1-0

[Event "Rated Classical game"]
[Site "https://lichess.org/a9tcp02g"]
[White "Desmond_Wilson"]
[Black "savinka59"]
[Result "1-0"]
[UTCDate "2012.12.31"]
[UTCTime "23:04:12"]
[WhiteElo "1654"]
[BlackElo "1919"]
[WhiteRatingDiff "+19"]
[BlackRatingDiff "-22"]
[ECO "D04"]
[Opening "Queen's Pawn Game: Colle System, Anti-Colle"]
[TimeControl "480+2"]
lichess_db_standard_rated_2013-01.pgn
```

```
1. d4 d5 2. Nf3 Nf6 3. e3 Bf5 4. Nh4 Bg6 5. Nxg6 hxg6 6. Nd2 e6
7. Bd3 Bd6 8. e4 dxe4 9. Nxe4 Rxh2 10. Ke2 Rxh1 11. Qxh1 Nc6
12. Bg5 Ke7 13. Qh7 Nxd4+ 14. Kd2 Qe8 15. Qxg7 Qh8 16. Bxf6+
Kd7 17. Qxh8 Rxh8 18. Bxh8 1-0
```

[1639, 1403]

[1.0]

Tranform chess notation into numbers

- For every position we have (8,8,12)-format
  - (8,8) is the chess board and the 12 is for the different sorts of figures and the different colors
    - For example A[i,j,0]= 1 if the white king is in (i,j) and A[i,j,0]=0 if the white king is not there

Training a neural network

- First two steps are done in seperate files and localy to be independent of Google Colaboratory etc
- Now uploading in Google Colaboratory
  - Problem: Format (8,8,12,number_of_moves,number_of_games) takes a lot of memory
  - Transforming it into (8,8,number_of_moves,number_of_games) and back in Google Colaboratory

Problem with solutions for training

Chess player with rating μ_1 and μ_2. They play:

$$x_1 \in \mathcal{N}(\mu_1, \sigma_1^2)\ ,\ x_2 \in \mathcal{N}(\mu_2, \sigma_2^2)$$

with $\mathcal{N}(\mu, \sigma)$ standard normaldistribution with expectancy μ and variance σ^2. The mean expected error is

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

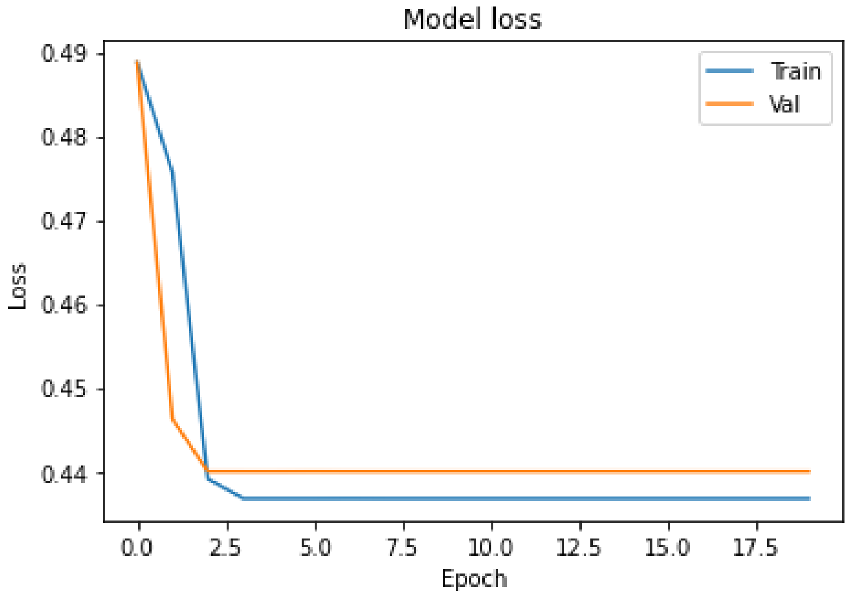
with density f.

- More games necessary for lower error.

Simplifying the problem

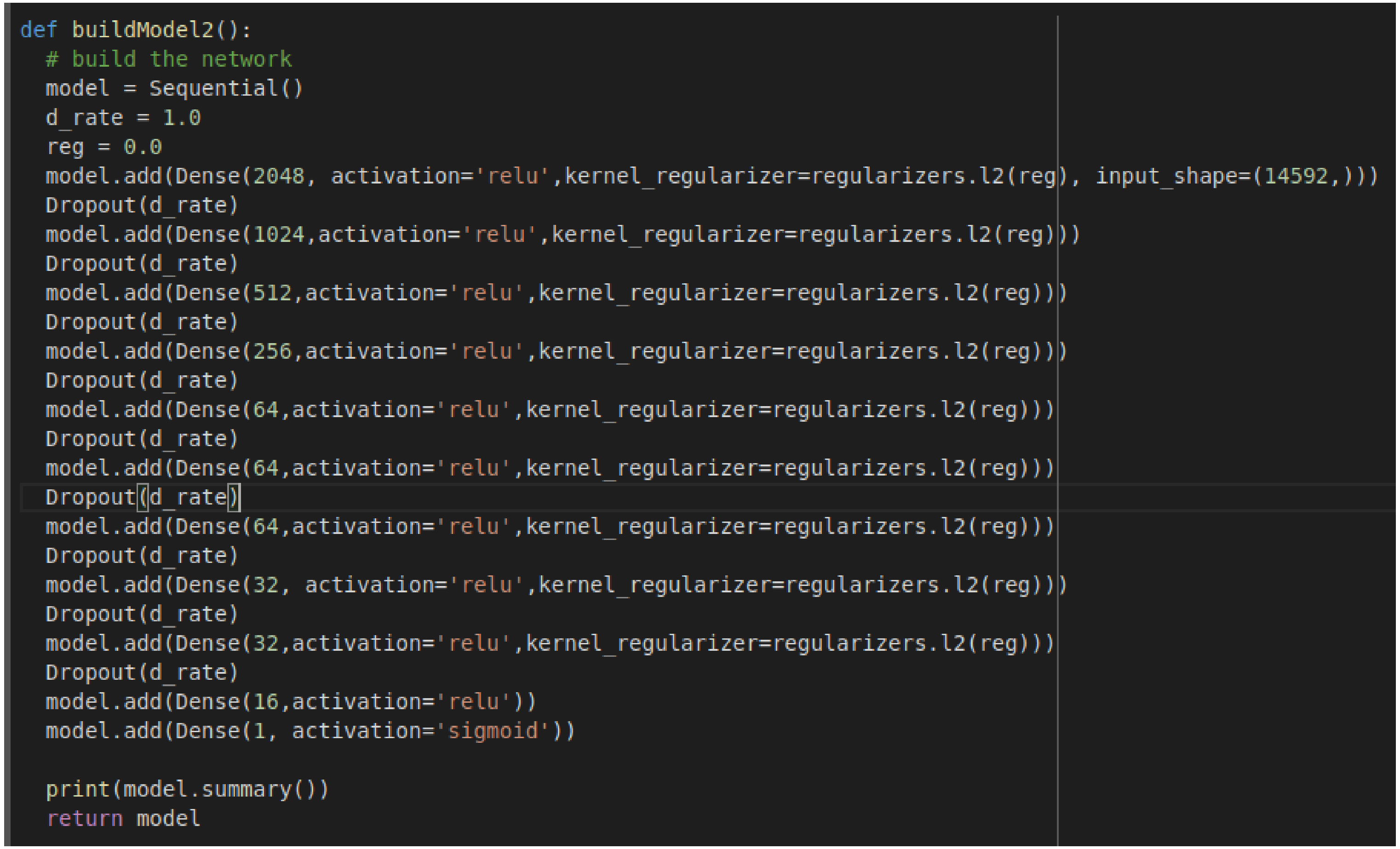
- Simplify the problem, to test the things done until now.
- Just take first 19 moves and predict, who wins.
  - Compare with real results
  - Hard to say if the model is good (data quality [mostly fast games, online, ...])
- Take mean absolut error, because mean squared would prefer draws

Results



Test set:
error guessing draw:
0.492
error corrected predictions:
0.440

Architecture



```
def buildModel2():
  # build the network
  model = Sequential()
  d_rate = 1.0
  reg = 0.0
  model.add(Dense(2048, activation='relu',kernel_regularizer=regularizers.l2(reg), input_shape=(14592,)))
  Dropout(d_rate)
  model.add(Dense(1024,activation='relu',kernel_regularizer=regularizers.l2(reg)))
  Dropout(d_rate)
  model.add(Dense(512,activation='relu',kernel_regularizer=regularizers.l2(reg)))
  Dropout(d_rate)
  model.add(Dense(256,activation='relu',kernel_regularizer=regularizers.l2(reg)))
  Dropout(d_rate)
  model.add(Dense(64,activation='relu',kernel_regularizer=regularizers.l2(reg)))
  Dropout(d_rate)
  model.add(Dense(64,activation='relu',kernel_regularizer=regularizers.l2(reg)))
  Dropout(d_rate)
  model.add(Dense(64,activation='relu',kernel_regularizer=regularizers.l2(reg)))
  Dropout(d_rate)
  model.add(Dense(32, activation='relu',kernel_regularizer=regularizers.l2(reg)))
  Dropout(d_rate)
  model.add(Dense(32,activation='relu',kernel_regularizer=regularizers.l2(reg)))
  Dropout(d_rate)
  model.add(Dense(16,activation='relu'))
  model.add(Dense(1, activation='sigmoid'))

  print(model.summary())
  return model
```

Further work to do - preprocessing

- The following data preprocesses are wrong:
  - If there are two figures that could move to one position, but one is not allowed, because one is bounded due to king, a new figure is added
  - If analysis is on and in the chess notation is something like „{„, the program crashes. Due to that there is a if-statement to ignore games
  - Due to both there are parts with only zeros in data, because list append did not work

Further work to do – deep neural network

- Try the original wanted task
- Try an RNN and not put in the whole game at once
- Adapt the DNN in a way, that the error reduces earlier and not after around 80 epochs.
- Include a softmax, such that we directly decide between won, draw or lost and do not have to preprocess it.