Fishnet: Discriminating AI from Human Chess Players

Cristobal Sciutto (Teammates: Lucas Sato and Sean Roelofs, both enrolled in CS 221)

Motivation

In 1997, Deep Blue defeated Gary Kasparov. Finally, after a long history of computer chess starting in the 60s, artificial intelligence had reached its first public milestone. Despite the initial cynicism, modern chess AIs (e.g. Google's Deep Mind) are well superior to modern Grandmasters. Nevertheless, their mechanics are still "dry". Magnus Carlsen, the current World Champion, remarks that when you play the AIs, not only will you inevitably lose, you will also be bored. In this context, it makes sense to develop a kind of chess Turing test. Can the AI play in a way considered "human?" When the way the machine plays is in fact indistinguishable from a human we will no longer be able to criticize style. The definition of that Turing test is the task at hand for this project.

Dataset

The data we have been working with takes the form of chess games' descriptions. We are concerned with PGN (Portable Game Notation) files - a standardized form of representing a chess game. Here is an example PGN file:

```
05> [White "lichess AI level 5"]
06> [Black "romaaaan"]
07> [Result "1-0"]
18>  1.  d4  c6  2.  Nc3  d5  3.  Nf3  Nf6  4.
     Ne5  Bf5  5.  h3  h6  6.  Rb1  b5  7.  b4
     a5  8.  e4  axb4  9.  exf5  bxc3  10.
     Be2  Rxa2  11.  Rb3  Ra1  12.  Nf3  Ne4
     13.  Ne5  f6  14.  Bh5+  g6  15.  Bxg6#
     1-0
```

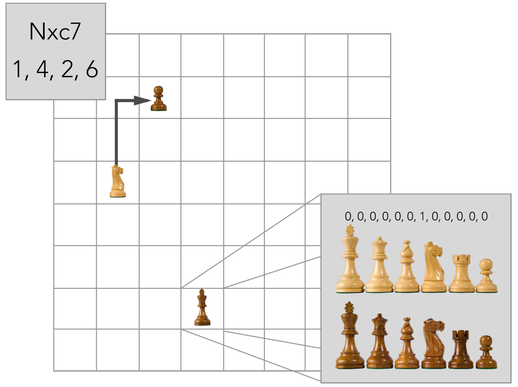
We began on a 2878 game dataset between human and the Stockfish level 5 AI, from the Lichess.org server. This served as our validation dataset for initial experimentation. From there we expanded to 84,033 games between any Human and AI, from the FICS server.

Methods

We approach the problem as a binary classification task on the players of a given game. We output y=0 if the AI is White, and y=1 if Black.

To perform the classification, four central methods were employed, of increasing complexity: Logistic Regression, SVMs, ANNs and RNN. The former established the baseline for performance.

Feature Representations



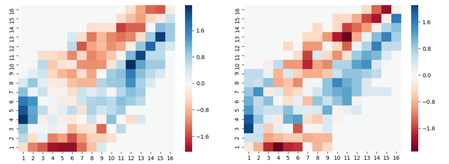
For the features, we focused on sparse feature-sets from which the algorithm can infer.

Origin and destination coordinates of moves: e.g. (4, 1, 4, 3) correspond to the move 1.e4, i.e. moving a pawn from e2 to e4. All moves are transformed into coordinates and flattened.

Board-state vector: each board-state is represented as a 64 x 6 x 2 n-array, corresponding to whether or not a player's piece is present in a given square. These boardstates are either compressed

Results

Experiment 1: # of pieces remaining



FICS Coeff. **Lichess Coeff.**

x: white pieces left,
y: black pieces left
positive: black is AI

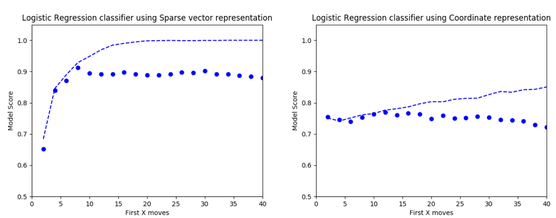
	Lichess	FICS
Log-Reg	0.69	0.66

Experiment 2: controlled squares

	Lichess	FICS
Log-Reg	0.72	0.66
SVM	0.59	0.51

Experiment 3: move coordinates and sparse vector representation

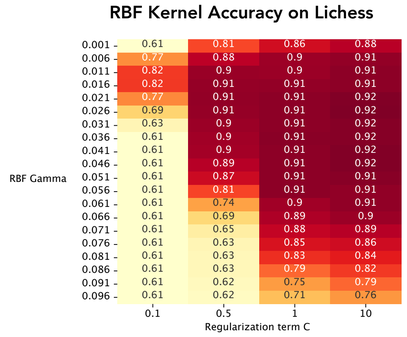
Lichess	Coord.	Sparse
Log-Reg	0.835, 0.741	1.0, 0.885
Linear	0.847, 0.739	1.0, 0.882
RBF (y=0.7)	1.0, 0.615	1.0, 0.615
Poly (deg=3)	0.994, 0.773	0.567, 0.619



We were severly overfitting with the SVMs. Furthermore, we needed to now if our techniques extrapolated to different AIs.

Interlude: increased dataset, caching, hyperparameter validation, and PCA

Experiment 4: SVMs with PCA



Best Results	Log-Reg	RBF g=0.04, C=10
FICS	0.811	0.900

Experiment 5: ANN, RNN

With our 80k data points, we were confident that a neural-network approach would be fruitful. For the implementation, we used the Keras framework. Below are our best results.

	Lichess	FICS
ANN (Layers: 210)	0.904	-
ANN (Layers: 200, 40)	0.890	0.882
ANN (Layers: 60, 50, 12)	-	0.906
LSTM (Outputs: 10)	0.904	0.860
LSTM (Outputs: 25)	0.915	-

Future Steps

We seemed to have reached an upper bound of 90% accuracy. The next step seems to be go back and look for more fruitful feature representations.

References

Deep Mind: eureka.eu.com/innovation/deep-mind-chess/
Python-Chess: github.com/niklasf/python-chess