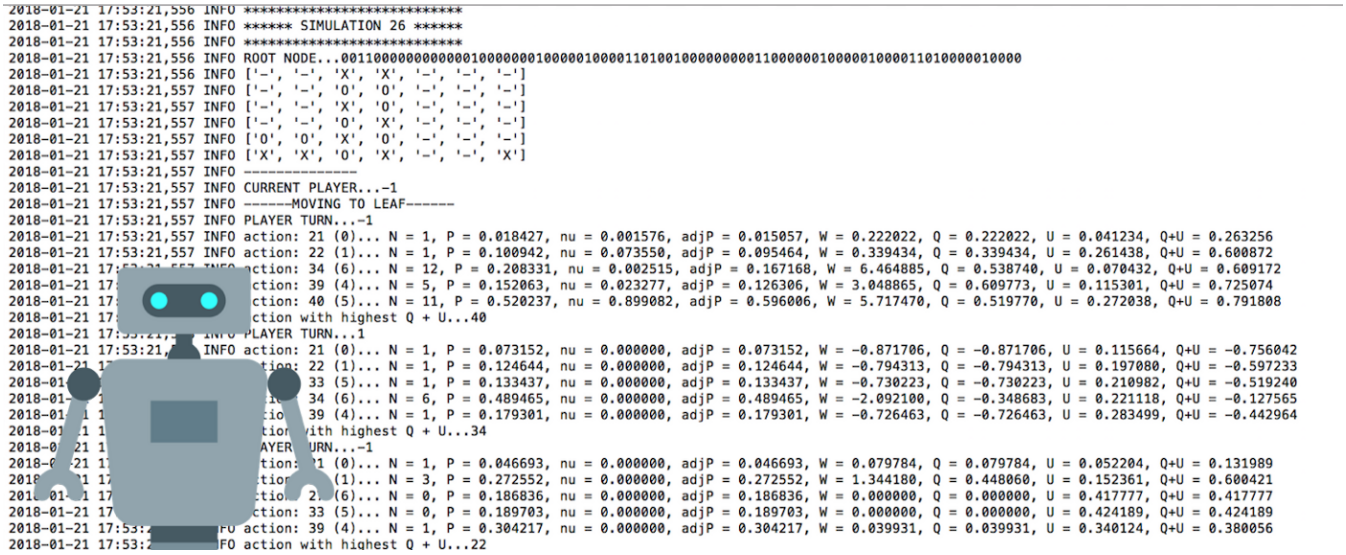



# How to build your own AlphaZero AI using Python and Keras

Teach a machine to learn Connect4 strategy through self-play and deep learning

David Foster Jan 26, 2018 · 11 min read

## Update! (2nd December 2019)

I've just released a series on MuZero — AlphaZero's younger and cooler brother. Check it out 👇

How to Build Your Own MuZero Using Python (Part 1/3)

How to Build Your Own MuZero Using Python (Part 2/3)

How to Build Your Own MuZero Using Python (Part 3/3)

In this article I'll attempt to cover three things:

1. Two reasons why AlphaZero is a massive step forward for Artificial Intelligence
2. How you can build a replica of the AlphaZero methodology to play the game Connect4
3. How you can adapt the code to plug in other games

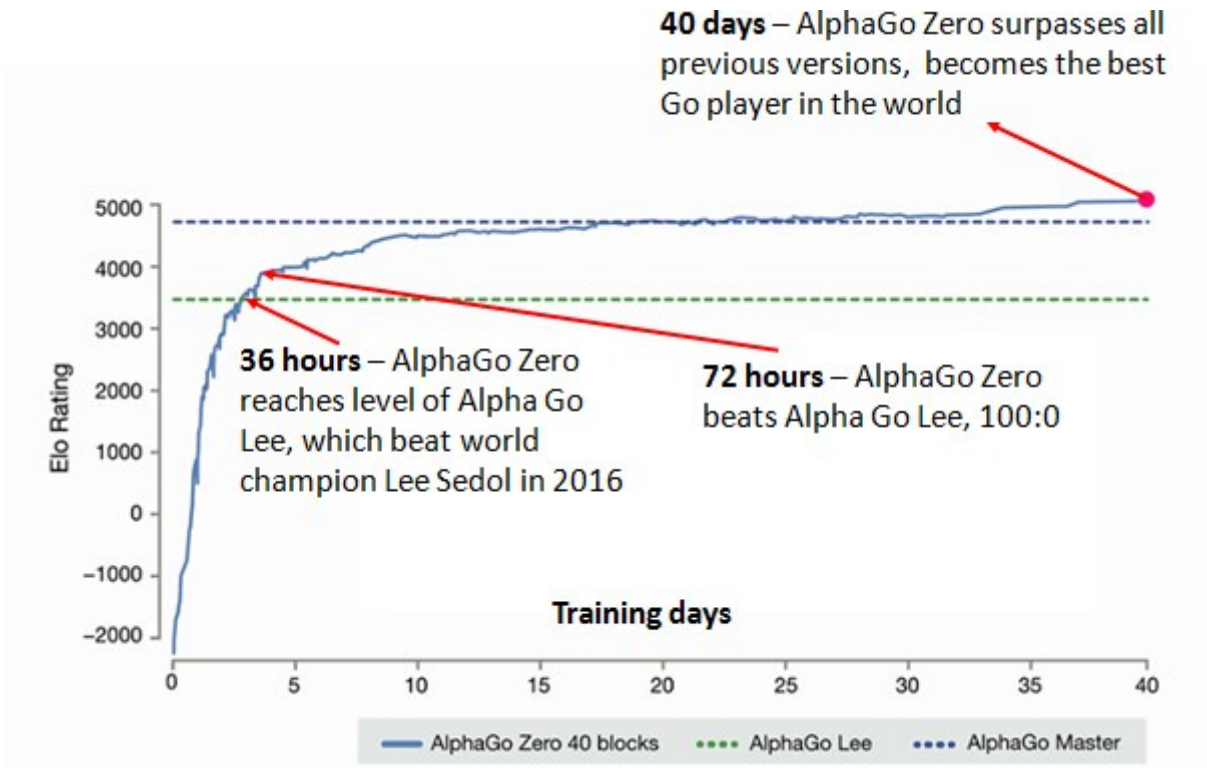
## AlphaGo → AlphaGo Zero → AlphaZero

In March 2016, Deepmind's AlphaGo beat 18 times world champion Go player Lee Sedol 4–1 in a series watched by over 200 million people. A machine had learnt a super-human strategy for playing Go, a feat previously thought impossible, or at the very least, at least a decade away from being accomplished.


Match 3 of AlphaGo vs Lee Sedol

This in itself, was a remarkable achievement. However, on 18th October 2017, DeepMind took a giant leap further.

it had done so by learning solely through self-play, starting 'tabula rasa' (blank state) and gradually finding strategies that would beat previous incarnations of itself. No longer was a database of human expert games required to build a super-human AI .




A graph from '**Mastering the Game of Go without Human Knowledge**'

A mere 48 days later, on 5th December 2017, DeepMind released another paper **'Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm'** showing how AlphaGo Zero could be adapted to beat the world-champion programs StockFish and Elmo at chess and shogi. The entire learning process, from being shown the games for the first time, to becoming the best computer program in the world, had taken under 24 hours.

With this, AlphaZero was born — the general algorithm for getting good at something, quickly, without any prior knowledge of human expert strategy.

There are two amazing things about this achievement:

## 1. AlphaZero requires zero human expertise as input

(the game state is fully known to both players at all times) because no prior expertise is required beyond the rules of the game.

This is how it was possible for DeepMind to publish the chess and shogi papers only 48 days after the original AlphaGo Zero paper. Quite literally, all that needed to change was the input file that describes the mechanics of the game and to tweak the hyper-parameters relating to the neural network and Monte Carlo tree search.

## 2. The algorithm is ridiculously elegant

If AlphaZero used super-complex algorithms that only a handful of people in the world understood, it would still be an incredible achievement. What makes it extraordinary is that a lot of the ideas in the paper are actually far less complex than previous versions. At its heart, lies the following beautifully simple mantra for learning:

> *Mentally play through possible future scenarios, giving priority to promising paths, whilst also considering how others are most likely to react to your actions and continuing to explore the unknown.*
>
> *After reaching a state that is unfamiliar, evaluate how favourable you believe the position to be and cascade the score back through previous positions in the mental pathway that led to this point.*
>
> *After you've finished thinking about future possibilities, take the action that you've explored the most.*
>
> *At the end of the game, go back and evaluate where you misjudged the value of the future positions and update your understanding accordingly.*

Doesn't that sound a lot like how you learn to play games? When you play a bad move, it's either because you misjudged the future value of resulting positions, or you misjudged the likelihood that your opponent would play a certain move, so didn't think to explore that possibility. These are exactly the two aspects of gameplay that AlphaZero is trained to learn.

### How to build your own AlphaZero

the code. There's also a great article here that explains how AlphaZero works in more detail.

## The code

Clone this Git repository, which contains the code I'll be referencing.

To start the learning process, run the top two panels in the **run.ipynb** Jupyter notebook. Once it's built up enough game positions to fill its memory the neural network will begin training. Through additional self-play and training, it will gradually get better at predicting the game value and next moves from any position, resulting in better decision making and smarter overall play.

We'll now have a look at the code in more detail, and show some results that demonstrate the AI getting stronger over time.

N.B — This is my own understanding of how AlphaZero works based on the information available in the papers referenced above. If any of the below is incorrect, apologies and I'll endeavour to correct it!

## Connect4

The game that our algorithm will learn to play is Connect4 (or Four In A Row). Not quite as complex as Go… but there are still 4,531,985,219,092 game positions in total.

Connect4

The game rules are straightforward. Players take it in turns to enter a piece of their colour in the top of any available column. The first player to get four of their colour in a row — each vertically, horizontally or diagonally, wins. If the entire grid is filled without a four-in-a-row being created, the game is drawn.

Here's a summary of the key files that make up the codebase:

### game.py

This file contains the game rules for Connect4.

Each squares is allocated a number from 0 to 41, as follows:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 32 | 33 | 34 |

Action squares for Connect4

The game.py file gives the logic behind moving from one game state to another, given a chosen action. For example, given the empty board and action 38, the **takeAction** method return a new game state, with the starting player's piece at the bottom of the centre column.

You can replace the game.py file with any game file that conforms to the same API and the algorithm will in principal, learn strategy through self play, based on the rules you have given it.

### run.ipynb

This contains the code that starts the learning process. It loads the game rules and then iterates through the main loop of the algorithm, which consist of three stages:

1. **Self-play**
2. **Retraining the Neural Network**
3. **Evaluating the Neural Network**

There are two agents involved in this loop, the **best_player** and the **current_player**.

The best_player contains the best performing neural network and is used to generate the self play memories. The current_player then retrains its neural network on these memories and is then pitched against the best_player. If it wins, the neural network inside the best_player is switched for the neural network inside the current_player, and the loop starts again.

### agent.py

This contains the Agent class (a player in the game). Each player is initialised with its own neural network and Monte Carlo Search Tree.

The **simulate** method runs the Monte Carlo Tree Search process. Specifically, the agent moves to a leaf node of the tree, evaluates the node with its neural network and then backfills the value of the node up through the tree.

to enact the move.

The **replay** method retrains the neural network, using memories from previous games.

## model.py


A sample of the residual convolutional network build using Keras

This file contains the Residual_CNN class, which defines how to build an instance of the neural network.

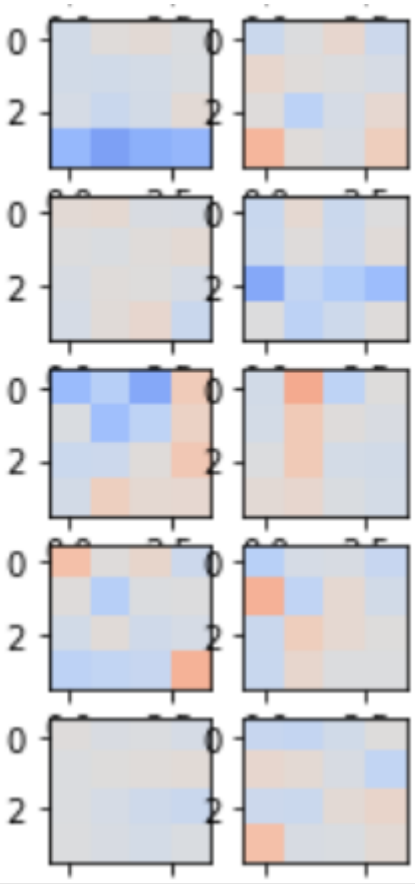
It uses a condensed version of the neural network architecture in the AlphaGoZero paper — i.e. a convolutional layer, followed by many residual layers, then splitting into a value and policy head.

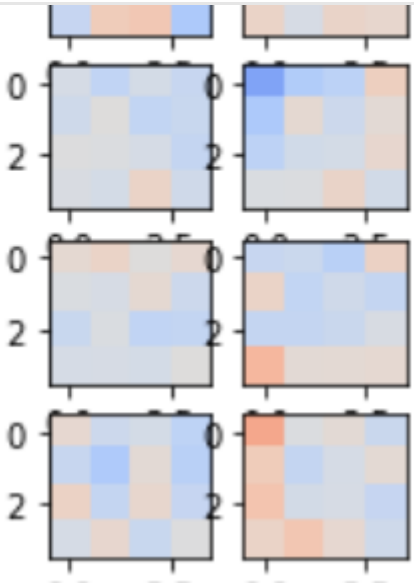
The depth and number of convolutional filters can be specified in the config file.

The Keras library is used to build the network, with a backend of Tensorflow.

To view individual convolutional filters and densely connected layers in the neural network, run the following inside the the run.ipynb notebook:

```
current_player.model.viewLayers()
```





Convolutional filters from the neural network

## MCTS.py

This contains the Node, Edge and MCTS classes, that constitute a Monte Carlo Search Tree.

The MCTS class contains the **moveToLeaf** and **backFill** methods previously mentioned, and instances of the Edge class store the statistics about each potential move.

## config.py

This is where you set the key parameters that influence the algorithm.

```
#### CONFIG FOR Connect 4

#### SELF PLAY
EPISODES = 75
MCTS_SIMS = 100
MEMORY_SIZE = 90000
TURNS_UNTIL_TAU0 = 10 # turn on which it starts playing deterministically
CPUCT = 1
EPSILON = 0.2
ALPHA = 0.8


#### RETRAINING
BATCH_SIZE = 256
EPOCHS = 1
REG_CONST = 0.0001
LEARNING_RATE = 0.1
MOMENTUM = 0.9
TRAINING_LOOPS = 10
```

```
    , {'filters':75, 'kernel_size': (4,4)}
    , {'filters':75, 'kernel_size': (4,4)}
    , {'filters':75, 'kernel_size': (4,4)}
    , {'filters':75, 'kernel_size': (4,4)}
    ]

#### EVALUATION
EVAL_EPISODES = 20
SCORING_THRESHOLD = 1.3
```

Adjusting these variables will affect that running time, neural network accuracy and overall success of the algorithm. The above parameters produce a high quality Connect4 player, but take a long time to do so. To speed the algorithm up, try the following parameters instead.

```
EPISODES = 25
MCTS_SIMS = 50
MEMORY_SIZE = 30000
```

### funcs.py

Contains the **playMatches** and **playMatchesBetweenVersions** functions that play matches between two agents.

To play against your creation, run the following code (it's also in the run.ipynb notebook)

```
from game import Game
from funcs import playMatchesBetweenVersions
import loggers as lg

env = Game()
playMatchesBetweenVersions(
env
, 1  # the run version number where the computer player is located
, -1 # the version number of the first player (-1 for human)
, 12 # the version number of the second player (-1 for human)
, 10 # how many games to play
, lg.logger_tourney # where to log the game to
, 0  # which player to go first - 0 for random
)
```

the root directory.

To restart the algorithm from this checkpoint later, transfer the run folder to the run_archive folder, attaching a run number to the folder name. Then, enter the run number, model version number and memory version number into the initialise.py file, corresponding to the location of the relevant files in the run_archive folder. Running the algorithm as usual will then start from this checkpoint.

### memory.py

An instance of the Memory class stores the memories of previous games, that the algorithm uses to retrain the neural network of the current_player.

### loss.py

This file contains a custom loss function, that masks predictions from illegal moves before passing to the cross entropy loss function.

### settings.py

The locations of the run and run_archive folders.

### loggers.py

Log files are saved to the **log** folder inside the run folder.

To turn on logging, set the values of the logger_disabled variables to False inside this file.

Viewing the log files will help you to understand how the algorithm works and see inside its 'mind'. For example, here is a sample from the logger.mcts file.

```
2018-01-21 17:53:21,556 INFO **************************
2018-01-21 17:53:21,556 INFO ****** SIMULATION 26 ******
2018-01-21 17:53:21,556 INFO **************************
2018-01-21 17:53:21,556 INFO ROOT NODE...00110000000000010000000100000100001110100100000000011000000100000100001101000010000
2018-01-21 17:53:21,556 INFO ['-', '-', 'X', 'X', '-', '-', '-']
2018-01-21 17:53:21,557 INFO ['-', '-', 'O', 'O', '-', '-', '-']
2018-01-21 17:53:21,557 INFO ['-', '-', 'X', 'O', '-', '-', '-']
2018-01-21 17:53:21,557 INFO ['-', '-', 'O', 'X', '-', '-', '-']
2018-01-21 17:53:21,557 INFO ['O', 'O', 'X', 'O', '-', '-', '-']
2018-01-21 17:53:21,557 INFO ['X', 'X', 'O', 'X', '-', '-', 'X']
2018-01-21 17:53:21,557 INFO --------------
2018-01-21 17:53:21,557 INFO CURRENT PLAYER...-1
2018-01-21 17:53:21,557 INFO ------MOVING TO LEAF------
2018-01-21 17:53:21,557 INFO PLAYER TURN...-1
2018-01-21 17:53:21,557 INFO action: 21 (0)... N = 1, P = 0.018427, nu = 0.001576, adjP = 0.015057, W = 0.222022, Q = 0.222022, U = 0.041234, Q+U = 0.263256
2018-01-21 17:53:21,557 INFO action: 22 (1)... N = 1, P = 0.100942, nu = 0.073550, adjP = 0.095464, W = 0.339434, Q = 0.339434, U = 0.261438, Q+U = 0.600872
2018-01-21 17:53:21,557 INFO action: 34 (6)... N = 12, P = 0.208331, nu = 0.002515, adjP = 0.167168, W = 6.464885, Q = 0.538740, U = 0.070432, Q+U = 0.609172
2018-01-21 17:53:21,557 INFO action: 39 (4)... N = 5, P = 0.152063, nu = 0.023277, adjP = 0.126306, W = 3.048865, Q = 0.609773, U = 0.115301, Q+U = 0.725074
2018-01-21 17:53:21,558 INFO action: 40 (5)... N = 11, P = 0.520237, nu = 0.899082, adjP = 0.596006, W = 5.717470, Q = 0.519770, U = 0.272038, Q+U = 0.791808
2018-01-21 17:53:21,558 INFO action with highest Q + U...40
2018-01-21 17:53:21,558 INFO PLAYER TURN...1
2018-01-21 17:53:21,558 INFO action: 21 (0)... N = 1, P = 0.073152, nu = 0.000000, adjP = 0.073152, W = -0.871706, Q = -0.871706, U = 0.115664, Q+U = -0.756042
2018-01-21 17:53:21,558 INFO action: 22 (1)... N = 1, P = 0.124644, nu = 0.000000, adjP = 0.124644, W = -0.794313, Q = -0.794313, U = 0.197080, Q+U = -0.597233
2018-01-21 17:53:21,558 INFO action: 33 (5)... N = 1, P = 0.133437, nu = 0.000000, adjP = 0.133437, W = -0.730223, Q = -0.730223, U = 0.210982, Q+U = -0.519240
2018-01-21 17:53:21,558 INFO action: 34 (6)... N = 6, P = 0.489465, nu = 0.000000, adjP = 0.489465, W = -2.092108, Q = -0.348683, U = 0.221118, Q+U = -0.127565
2018-01-21 17:53:21,558 INFO action: 39 (4)... N = 1, P = 0.179301, nu = 0.000000, adjP = 0.179301, W = -0.726463, Q = -0.726463, U = 0.283499, Q+U = -0.442964
2018-01-21 17:53:21,558 INFO action with highest Q + U...34
2018-01-21 17:53:21,559 INFO PLAYER TURN...-1
2018-01-21 17:53:21,559 INFO action: 21 (0)... N = 1, P = 0.046693, nu = 0.000000, adjP = 0.046693, W = 0.079784, Q = 0.079784, U = 0.052204, Q+U = 0.131989
2018-01-21 17:53:21,559 INFO action: 22 (1)... N = 3, P = 0.272552, nu = 0.000000, adjP = 0.272552, W = 1.344180, Q = 0.448060, U = 0.152361, Q+U = 0.600421
2018-01-21 17:53:21,559 INFO action: 27 (6)... N = 0, P = 0.186836, nu = 0.000000, adjP = 0.186836, W = 0.000000, Q = 0.000000, U = 0.417777, Q+U = 0.417777
2018-01-21 17:53:21,559 INFO action: 33 (5)... N = 0, P = 0.189703, nu = 0.000000, adjP = 0.189703, W = 0.000000, Q = 0.000000, U = 0.424189, Q+U = 0.424189
2018-01-21 17:53:21,559 INFO action: 39 (4)... N = 1, P = 0.304217, nu = 0.000000, adjP = 0.304217, W = 0.039931, Q = 0.039931, U = 0.340124, Q+U = 0.380056
```

```
2018-01-21 17:53:21,561 INFO DONE...0
2018-01-21 17:53:21,561 INFO ['-', '-', 'X', 'X', '-', '-', '-']
2018-01-21 17:53:21,562 INFO ['-', '-', 'O', 'O', '-', '-', '-']
2018-01-21 17:53:21,562 INFO ['-', '-', 'X', 'O', '-', '-', '-']
2018-01-21 17:53:21,562 INFO ['-', 'O', 'O', 'X', '-', '-', 'X']
2018-01-21 17:53:21,562 INFO ['O', 'O', 'X', 'O', '-', '-', 'X']
2018-01-21 17:53:21,562 INFO ['X', 'X', 'O', 'X', '-', 'O', 'X']
2018-01-21 17:53:21,562 INFO --------------
2018-01-21 17:53:21,562 INFO ------EVALUATING LEAF------
2018-01-21 17:53:21,570 INFO PREDICTED VALUE FOR -1: 0.718004
2018-01-21 17:53:21,571 INFO added node...001100000000000010000000100100100011101001000000001100001010000110000110100000010010...p = 0.057289
2018-01-21 17:53:21,571 INFO added node...001100000000000010000000100100100011101001000000001100000010011000011010000010010...p = 0.877674
2018-01-21 17:53:21,571 INFO added node...001100000000000010000000100100100011101001000000001100000010001110000110100000010010...p = 0.004717
2018-01-21 17:53:21,571 INFO added node...001100000000000010000000100100100011101001000000001100000010000110000110101000010010...p = 0.015134
2018-01-21 17:53:21,572 INFO added node...001100000000000010000000100100100011101001000000001100000010000110000110100000010110...p = 0.045186
2018-01-21 17:53:21,572 INFO ------DOING BACKFILL------
2018-01-21 17:53:21,572 INFO updating edge with value 0.718004 for player -1... N = 12, W = 6.435474, Q = 0.536290
2018-01-21 17:53:21,572 INFO ['-', '-', 'X', 'X', '-', '-', '-']
2018-01-21 17:53:21,572 INFO ['-', '-', 'O', 'O', '-', '-', '-']
2018-01-21 17:53:21,572 INFO ['-', '-', 'X', 'O', '-', '-', '-']
2018-01-21 17:53:21,572 INFO ['-', 'O', 'O', 'X', '-', '-', 'X']
2018-01-21 17:53:21,572 INFO ['O', 'O', 'X', 'O', '-', '-', 'X']
2018-01-21 17:53:21,572 INFO ['X', 'X', 'O', 'X', '-', 'O', 'X']
2018-01-21 17:53:21,572 INFO --------------
2018-01-21 17:53:21,572 INFO updating edge with value -0.718004 for player 1... N = 7, W = -2.810105, Q = -0.401444
2018-01-21 17:53:21,573 INFO ['-', '-', 'X', 'X', '-', '-', '-']
2018-01-21 17:53:21,573 INFO ['-', '-', 'O', 'O', '-', '-', '-']
2018-01-21 17:53:21,573 INFO ['-', '-', 'X', 'O', '-', '-', '-']
2018-01-21 17:53:21,573 INFO ['-', '-', 'O', 'X', '-', '-', '-']
2018-01-21 17:53:21,573 INFO ['O', 'O', 'X', 'O', '-', '-', 'X']
2018-01-21 17:53:21,573 INFO ['X', 'X', 'O', 'X', '-', 'O', 'X']
2018-01-21 17:53:21,573 INFO --------------
2018-01-21 17:53:21,573 INFO updating edge with value 0.718004 for player -1... N = 4, W = 2.062184, Q = 0.515546
2018-01-21 17:53:21,573 INFO ['-', '-', 'X', 'X', '-', '-', '-']
2018-01-21 17:53:21,573 INFO ['-', '-', 'O', 'O', '-', '-', '-']
2018-01-21 17:53:21,573 INFO ['-', '-', 'X', 'O', '-', '-', '-']
2018-01-21 17:53:21,573 INFO ['-', 'O', 'O', 'X', '-', '-', '-']
2018-01-21 17:53:21,573 INFO ['O', 'O', 'X', 'O', '-', '-', 'X']
2018-01-21 17:53:21,574 INFO ['X', 'X', 'O', 'X', '-', 'O', 'X']
2018-01-21 17:53:21,574 INFO --------------
2018-01-21 17:53:21,574 INFO updating edge with value -0.718004 for player 1... N = 1, W = -0.718004, Q = -0.718004
2018-01-21 17:53:21,574 INFO ['-', '-', 'X', 'X', '-', '-', '-']
2018-01-21 17:53:21,574 INFO ['-', '-', 'O', 'O', '-', '-', '-']
```

Output from the logger.mcts file

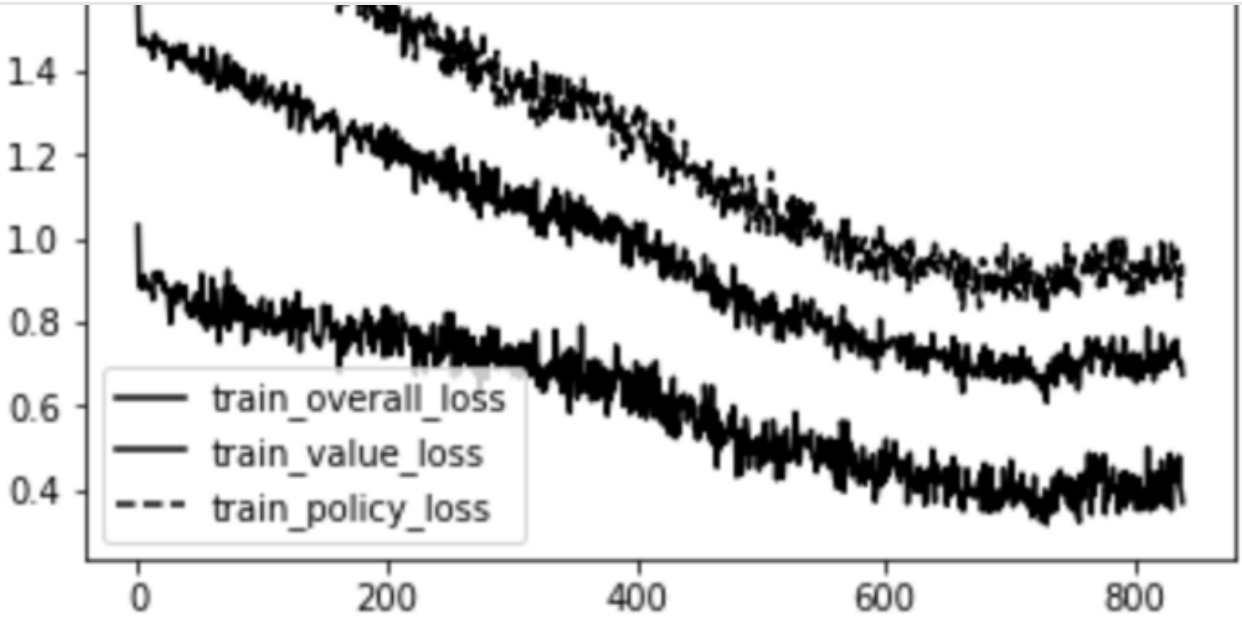
Equally from the logger.tourney file, you can see the probabilities attached to each move, during the evaluation phase:

```
2018-01-21 22:37:27,826 INFO ====================
2018-01-21 22:37:27,827 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-21 22:37:27,827 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-21 22:37:27,827 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-21 22:37:27,827 INFO ['-', '-', '-', 'X', 'O', '-', '-']
2018-01-21 22:37:27,827 INFO ['-', '-', '-', 'X', 'O', 'X', '-']
2018-01-21 22:37:27,827 INFO ['O', 'X', 'X', 'O', 'X', 'O', '-']
2018-01-21 22:37:27,827 INFO --------------
2018-01-21 22:37:30,619 INFO action: 18
2018-01-21 22:37:30,619 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-21 22:37:30,619 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-21 22:37:30,619 INFO ['----', '----', '----', '0.11', '0.44', '----', '----']
2018-01-21 22:37:30,620 INFO ['----', '----', '----', '----', '----', '0.10', '----']
2018-01-21 22:37:30,620 INFO ['0.08', '0.10', '0.09', '----', '----', '----', '----']
2018-01-21 22:37:30,620 INFO ['----', '----', '----', '----', '----', '----', '0.09']
2018-01-21 22:37:30,620 INFO MCTS perceived value for O: 0.130000
2018-01-21 22:37:30,621 INFO NN perceived value for O: 0.040000
```

Output from the logger.tourney file

## Results

Training over a couple of days produces the following chart of loss against mini-batch iteration number:




Loss against mini-batch iteration number

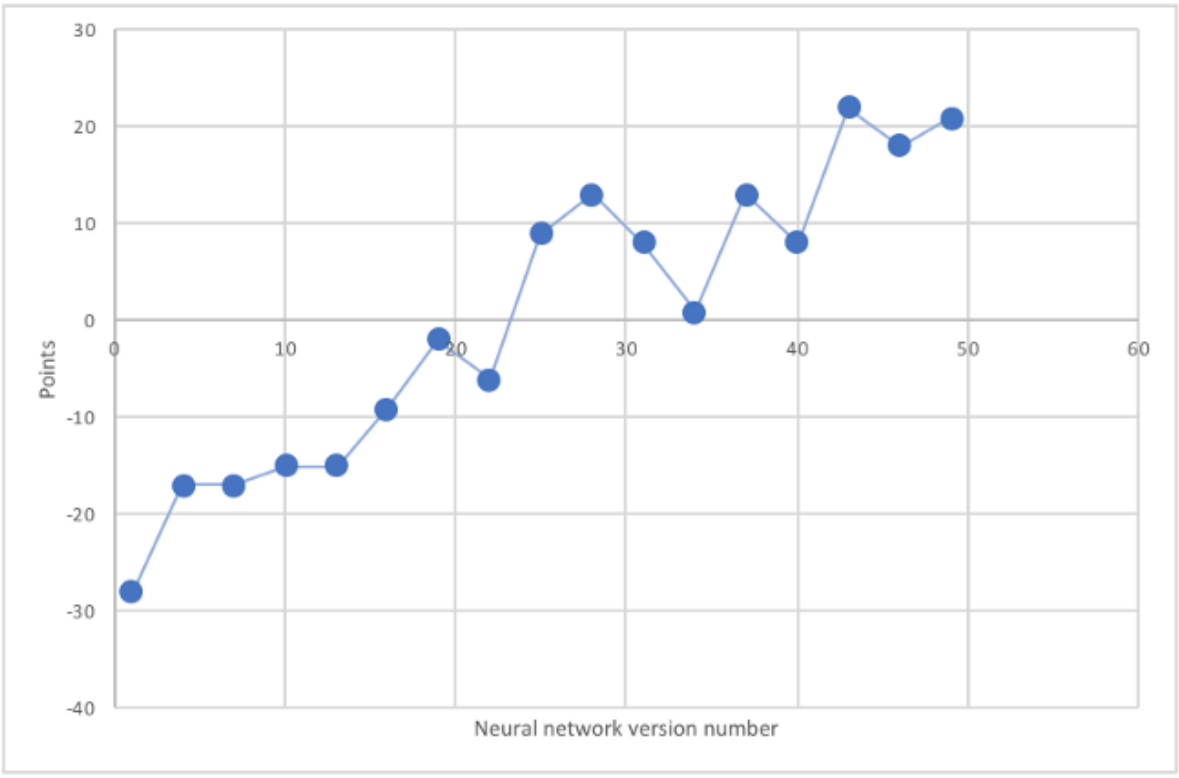
The top line is the error in the policy head (the cross entropy of the MCTS move probabilities, against the output from the neural network). The bottom line is the error in the value head (the mean squared error between the actual game value and the neural network predict of the value). The middle line is an average of the two.

Clearly, the neural network is getting better at predicting the value of each game state and the likely next moves. To show how this results in stronger and stronger play, I ran a league between 17 players, ranging from the 1st iteration of the neural network, up to the 49th. Each pairing played twice, with both players having a chance to play first.

Here are the final standings:

| | player2 | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| player 1 | 1 | 4 | 7 | 10 | 13 | 16 | 19 | 22 | 25 | 28 | 31 | 34 | 37 | 40 | 43 | 46 | 49 | total |
| 1 | | -2 | -2 | -2 | -2 | 0 | -1 | -2 | -2 | -2 | -2 | -2 | -2 | -2 | -2 | -2 | -1 | -28 |
| 4 | 2 | | 0 | 0 | 0 | -2 | -2 | -1 | -2 | 0 | -2 | -2 | -2 | -2 | 0 | -2 | -2 | -17 |
| 7 | 2 | 0 | | -2 | -2 | -1 | 0 | -1 | -2 | -2 | -2 | 1 | -2 | -2 | 0 | -2 | -2 | -17 |
| 10 | 2 | 0 | 2 | | 0 | -1 | -1 | -2 | -1 | -2 | -2 | -2 | -2 | -2 | -2 | 0 | -2 | -15 |
| 13 | 2 | 0 | 2 | 0 | | -1 | -2 | -1 | -1 | -2 | -1 | -2 | -1 | -2 | -2 | -2 | -2 | -15 |
| 16 | 0 | 2 | 1 | 1 | 1 | | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -2 | -2 | -2 | -9 |
| 19 | 1 | 2 | 0 | 1 | 2 | 1 | | 0 | -2 | -2 | 1 | -1 | -1 | 2 | -2 | -2 | -2 | -2 |
| 22 | 2 | 1 | 1 | 2 | 1 | 1 | 0 | | -2 | -2 | -1 | -1 | -1 | -2 | -2 | -2 | -1 | -6 |

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 2 | 2 | 2 | 2 | 1 | 1 | -1 | 1 | -1 | 0 | | 1 | 0 | 0 | 0 | 0 | -2 | 8 |
| 34 | 2 | 2 | -1 | 2 | 2 | 1 | 1 | 1 | 1 | -2 | -1 | | 0 | -1 | -2 | -2 | -2 | 1 |
| 37 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 0 | -1 | 0 | 0 | | -1 | 2 | 1 | 0 | 13 |
| 40 | 2 | 2 | 2 | 2 | 2 | 1 | -2 | 2 | 0 | 0 | 0 | 1 | 1 | | -2 | -2 | -1 | 8 |
| 43 | 2 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 1 | 0 | 0 | 2 | 2 | 2 | | 1 | 2 | 22 |
| 46 | 2 | 2 | 2 | 0 | 2 | 2 | 2 | 2 | 2 | 0 | 0 | 2 | -1 | 2 | -1 | | 0 | 18 |
| 49 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 0 | 1 | -2 | 0 | | 21 |




Clearly, the later versions of the neural network are superior to the earlier versions, winning most of their games. It also appears that the learning hasn't yet saturated — with further training time, the players would continue to get stronger, learning more and more intricate strategies.

As an example, one clear strategy that the neural network has favoured over time is grabbing the centre column early. Observe the difference between the first version of the algorithm and say, the 30th version:

### 1st neural network version

```
2018-01-26 12:20:26,734 INFO EPISODE 2 OF 10
2018-01-26 12:20:26,734 INFO ====================
2018-01-26 12:20:26,735 INFO player1 plays as X
2018-01-26 12:20:26,736 INFO ['-', '-', '-', '-', '-', '-', '-']
```

```
2018-01-26 12:20:26,736 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:26,736 INFO --------------
2018-01-26 12:20:28,083 INFO action: 39
2018-01-26 12:20:28,083 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:28,083 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:28,083 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:28,083 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:28,083 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:28,084 INFO ['0.15', '0.13', '0.14', '0.14', '0.16', '0.12', '0.15']
2018-01-26 12:20:28,084 INFO MCTS perceived value for X: 0.010000
2018-01-26 12:20:28,084 INFO NN perceived value for X: 0.060000
2018-01-26 12:20:28,084 INFO ====================
2018-01-26 12:20:28,084 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:28,084 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:28,084 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:28,084 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:28,084 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:28,085 INFO ['-', '-', '-', '-', 'X', '-', '-']
2018-01-26 12:20:28,085 INFO --------------
2018-01-26 12:20:30,339 INFO action: 37
2018-01-26 12:20:30,339 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:30,339 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:30,339 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:30,339 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:30,340 INFO ['----', '----', '----', '----', '0.13', '----', '----']
2018-01-26 12:20:30,340 INFO ['0.12', '0.12', '0.17', '0.17', '----', '0.14', '0.14']
2018-01-26 12:20:30,340 INFO MCTS perceived value for O: 0.010000
2018-01-26 12:20:30,340 INFO NN perceived value for O: 0.060000
2018-01-26 12:20:30,340 INFO ====================
2018-01-26 12:20:30,341 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:30,341 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:30,341 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:30,341 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:30,341 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:30,341 INFO ['-', '-', 'O', '-', 'X', '-', '-']
2018-01-26 12:20:30,341 INFO --------------
2018-01-26 12:20:31,598 INFO action: 36
2018-01-26 12:20:31,598 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:31,598 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:31,598 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:31,598 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:20:31,598 INFO ['----', '----', '0.14', '----', '0.14', '----', '----']
2018-01-26 12:20:31,598 INFO ['0.14', '0.16', '----', '0.15', '----', '0.15', '0.14']
2018-01-26 12:20:31,599 INFO MCTS perceived value for X: 0.010000
2018-01-26 12:20:31,599 INFO NN perceived value for X: 0.060000
2018-01-26 12:20:31,599 INFO ====================
2018-01-26 12:20:31,599 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:31,599 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:31,599 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:31,599 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:31,599 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:20:31,599 INFO ['-', 'X', 'O', '-', 'X', '-', '-']
```

## 30th neural network version

```
2018-01-26 12:18:21,529 INFO EPISODE 1 OF 10
2018-01-26 12:18:21,529 INFO ====================
2018-01-26 12:18:21,530 INFO player2 plays as X
2018-01-26 12:18:21,530 INFO --------------
2018-01-26 12:18:21,530 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:21,531 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:21,531 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:21,531 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:21,531 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:21,531 INFO ['-', '-', '-', '-', '-', '-', '-']
```

```
2018-01-26 12:18:23,383 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:18:23,383 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:18:23,383 INFO ['0.03', '0.04', '0.04', '0.74', '0.05', '0.07', '0.03']
2018-01-26 12:18:23,383 INFO MCTS perceived value for X: 0.110000
2018-01-26 12:18:23,383 INFO NN perceived value for X: 0.160000
2018-01-26 12:18:23,383 INFO ====================
2018-01-26 12:18:23,384 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:23,384 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:23,384 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:23,384 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:23,384 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:23,384 INFO ['-', '-', '-', 'X', '-', '-', '-']
2018-01-26 12:18:23,384 INFO --------------
2018-01-26 12:18:25,232 INFO action: 31
2018-01-26 12:18:25,232 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:18:25,232 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:18:25,232 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:18:25,232 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:18:25,232 INFO ['----', '----', '----', '0.81', '----', '----', '----']
2018-01-26 12:18:25,232 INFO ['0.03', '0.03', '0.04', '----', '0.04', '0.03', '0.02']
2018-01-26 12:18:25,232 INFO MCTS perceived value for O: -0.070000
2018-01-26 12:18:25,233 INFO NN perceived value for O: 0.020000
2018-01-26 12:18:25,233 INFO ====================
2018-01-26 12:18:25,233 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:25,233 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:25,233 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:25,233 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:25,233 INFO ['-', '-', '-', 'O', '-', '-', '-']
2018-01-26 12:18:25,233 INFO ['-', '-', '-', 'X', '-', '-', '-']
2018-01-26 12:18:25,233 INFO --------------
2018-01-26 12:18:26,719 INFO action: 24
2018-01-26 12:18:26,720 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:18:26,720 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:18:26,720 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:18:26,720 INFO ['----', '----', '----', '0.82', '----', '----', '----']
2018-01-26 12:18:26,720 INFO ['----', '----', '----', '----', '----', '----', '----']
2018-01-26 12:18:26,721 INFO ['0.01', '0.02', '0.06', '----', '0.04', '0.04', '0.01']
2018-01-26 12:18:26,721 INFO MCTS perceived value for X: 0.040000
2018-01-26 12:18:26,721 INFO NN perceived value for X: 0.060000
2018-01-26 12:18:26,721 INFO ====================
2018-01-26 12:18:26,721 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:26,721 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:26,721 INFO ['-', '-', '-', '-', '-', '-', '-']
2018-01-26 12:18:26,721 INFO ['-', '-', '-', 'X', '-', '-', '-']
2018-01-26 12:18:26,722 INFO ['-', '-', '-', 'O', '-', '-', '-']
2018-01-26 12:18:26,722 INFO ['-', '-', '-', 'X', '-', '-', '-']
```

This is a good strategy as many lines require the centre column — claiming this early ensures your opponent cannot take advantage of this. This has been learnt by the neural network, without any human input.

## Learning a different game

There is a game.py file for a game called 'Metasquares' in the **games** folder. This involves placing X and O markers in a grid to try to form squares of different sizes. Larger squares score more points than smaller squares and the player with the most points when the grid is full wins.

## Summary

Hopefully you find this article useful — let me know in the comments below if you find any typos or have questions about anything in the codebase or article and I'll get back to you as soon as possible.


If you would like to learn more about how our company, Applied Data Science develops innovative data science solutions for businesses, feel free to get in touch through our website or directly through LinkedIn.

*… and if you like this, feel free to leave a few hearty claps :)*

Applied Data Science is a London based consultancy that implements end-to-end data science solutions for businesses, delivering measurable value. If you're looking to do more with your data, let's talk.