omegathello

An Othello agent utilizing deep learning

By Johan Karlberg & Erik Månsson

The goal

To explore the usage of deep learning for playing turn-based games, AlphaGo style.

Why Othello?

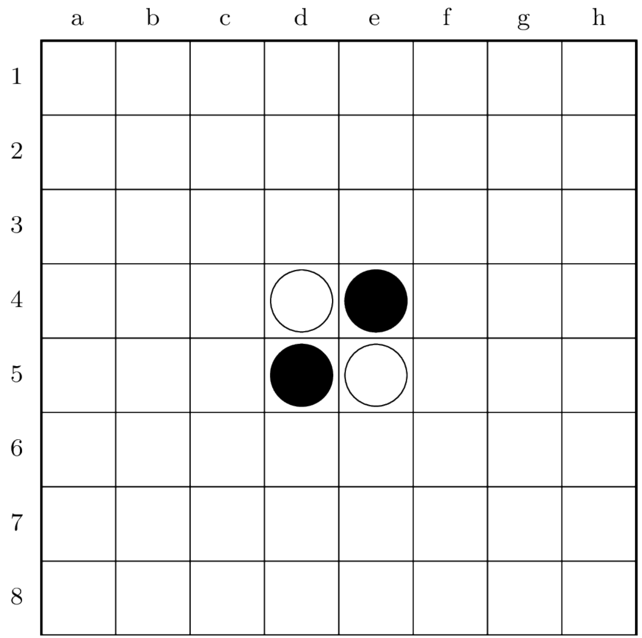
In a nutshell - simplicity:

- The game only has a single piece type (in contrast to Chess)
- A move does not depend on previous moves (as in Go)
- Rather simple implementation of the game itself
- Has a clear and simple objective

But at the same time:

- Complex enough to be competitive

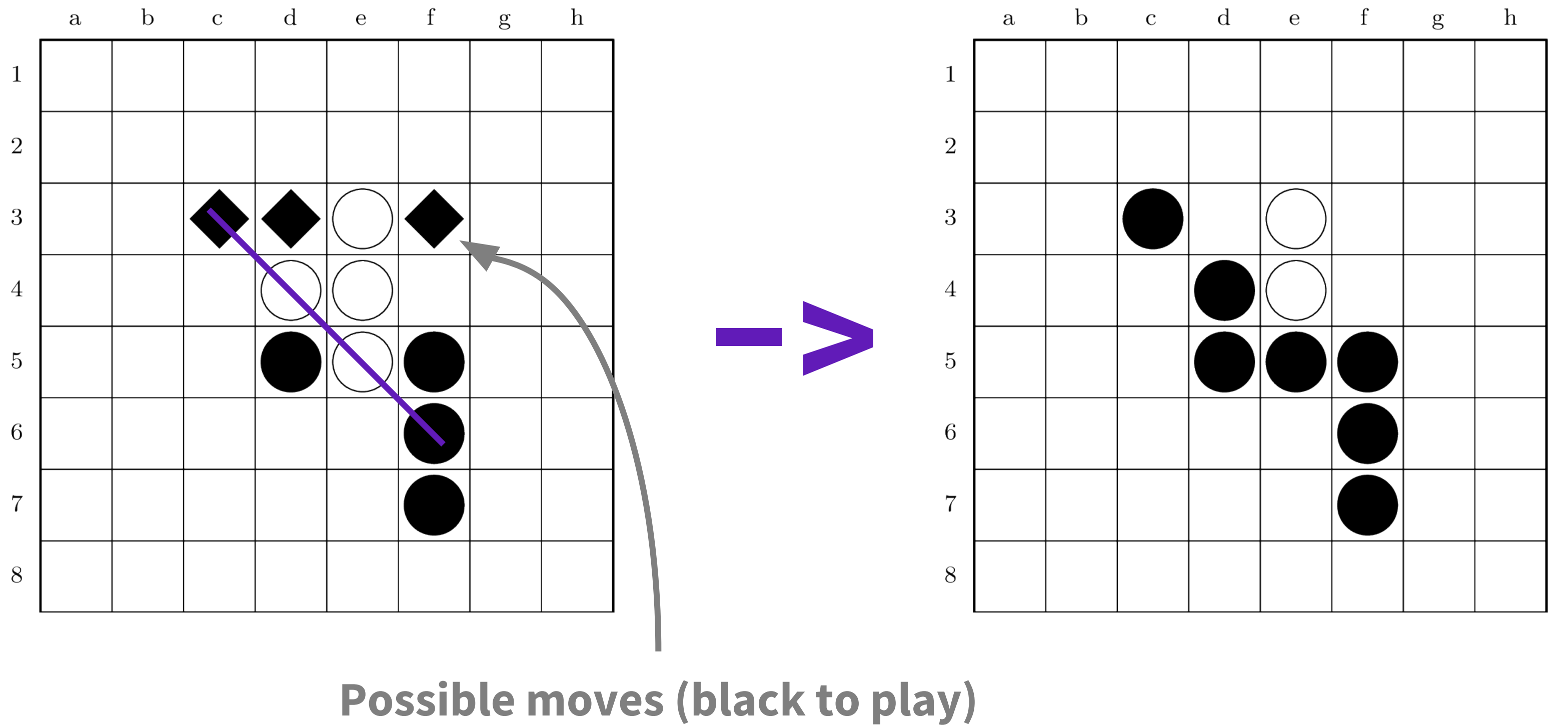
The rules of Othello



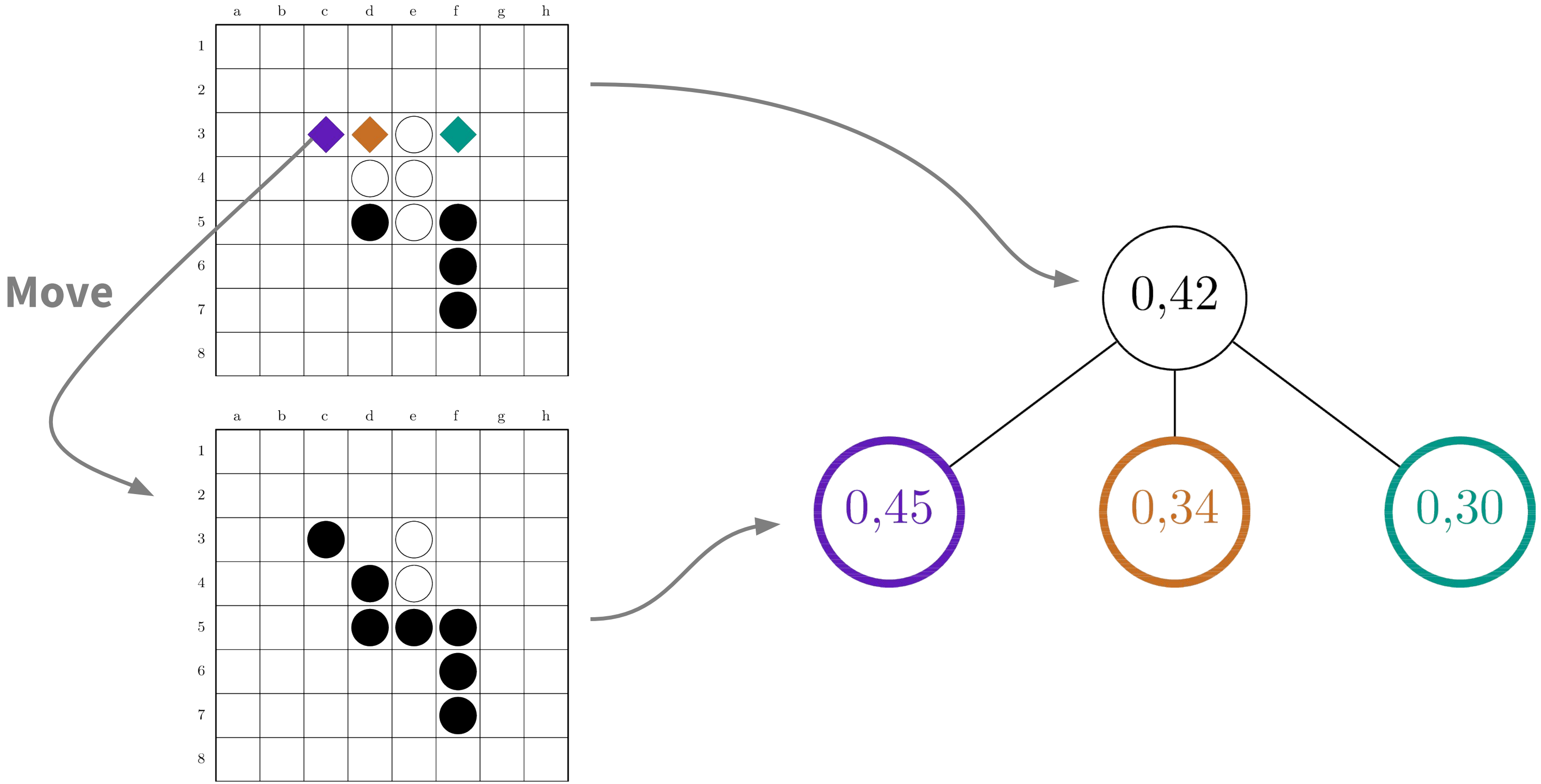
Starting position

- Turn based
- Black moves first
- If no move can be made, the play passes back to the other player
- When no more moves can be made, the player with the most disks wins

The rules of Othello



The game tree



0.45?

Evaluation

An evaluation of a given game state is an estimation of which player currently has the better position.

The simplest possible heuristic is taking the score
“score” = “number of black disks” - “number of white disks”

Creating better heuristics requires deep knowledge about the game.

Supervised learning

We don’t have deep knowledge about the game.

We do however have a lot of data from tournaments.

Let’s learn from them using supervised learning on deep neural networks.

A classification problem

More formally, given a game state consisting of

- The board
- The player to move

We want to classify whether

1) Black is winning, or
2) White is winning

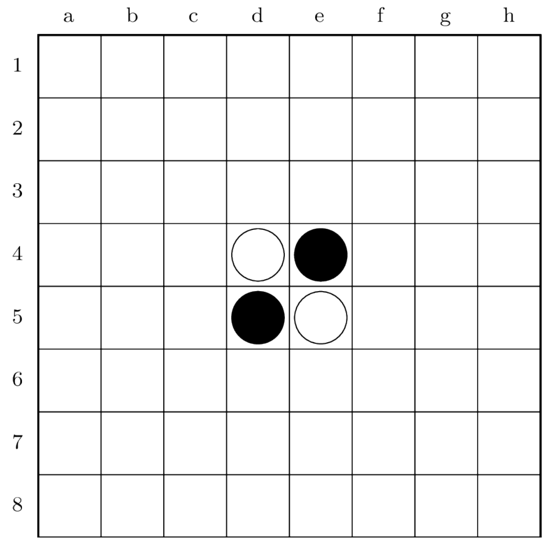
Finding a solution

We need to find suitable...

- Input shape and format
- Labelling & error function
- Data points
- Network structure

Also, we don’t have a lot of computing power, so we need to keep it **efficient**.

Board representation

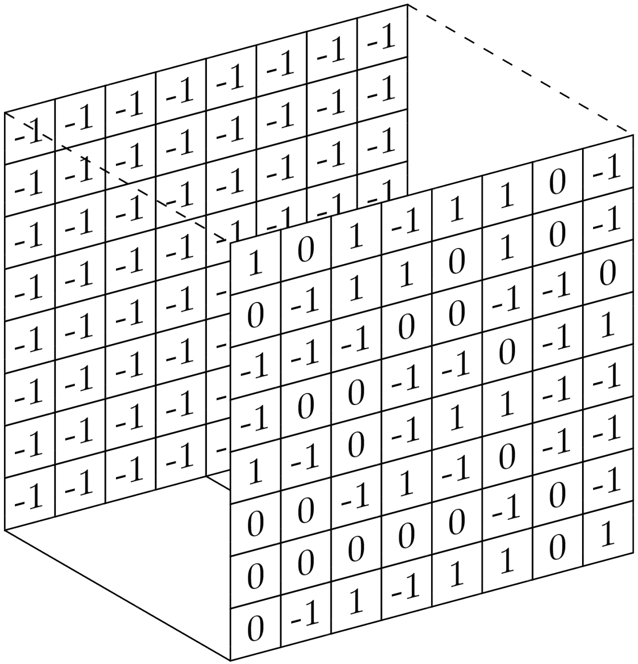


$$=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

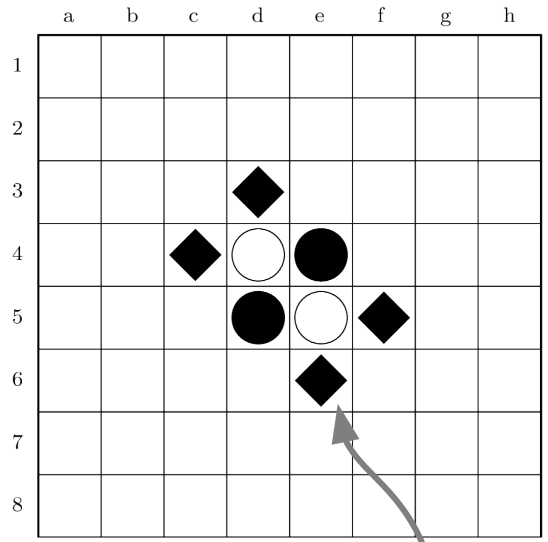
But who’s turn is it?

A second input layer is added to make use of the information of who’s turn it is.

Inspired by AlphaGo



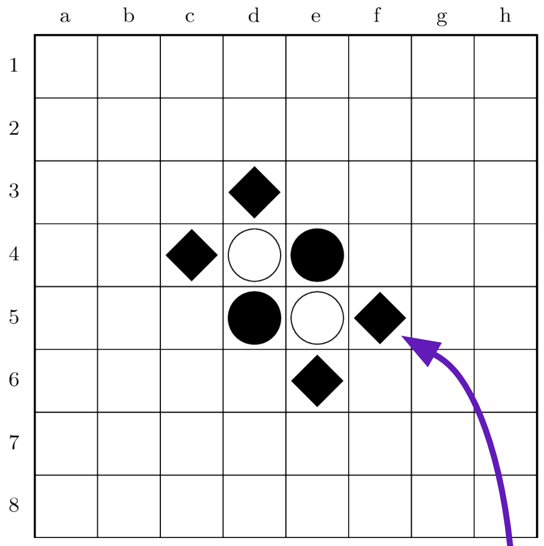
Another thing, symmetry



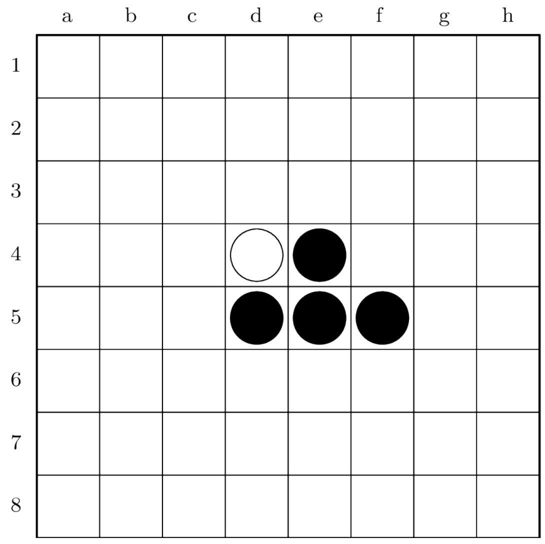
Possible first move

All equally good
=
Unnecessarily complex

Handling symmetry

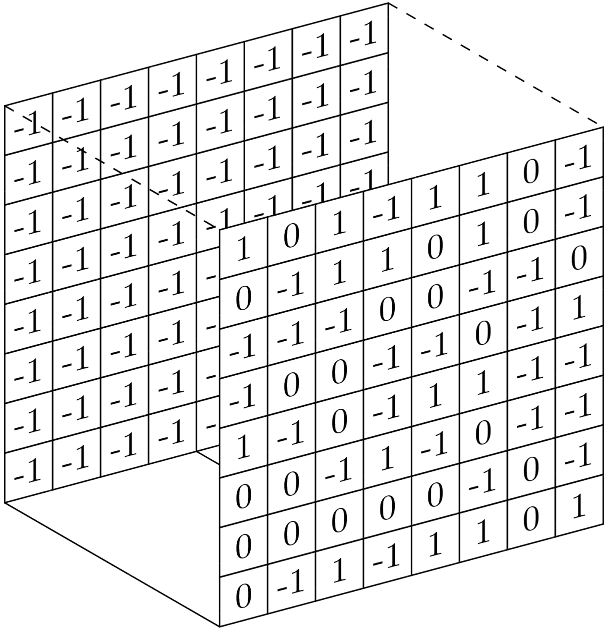


Starting position in practice



Per tournament rules, always assume this move

Labelling


1 0 1 -1 1 1 0 -1
0 -1 1 1 0 1 0 -1
-1 -1 -1 0 0 -1 -1 0
-1 0 0 -1 -1 0 -1 1
1 -1 0 -1 1 1 -1 -1
0 0 -1 1 -1 0 -1 -1
0 0 0 0 0 -1 0 -1
0 -1 1 -1 1 1 0 1

->

?

Labelling

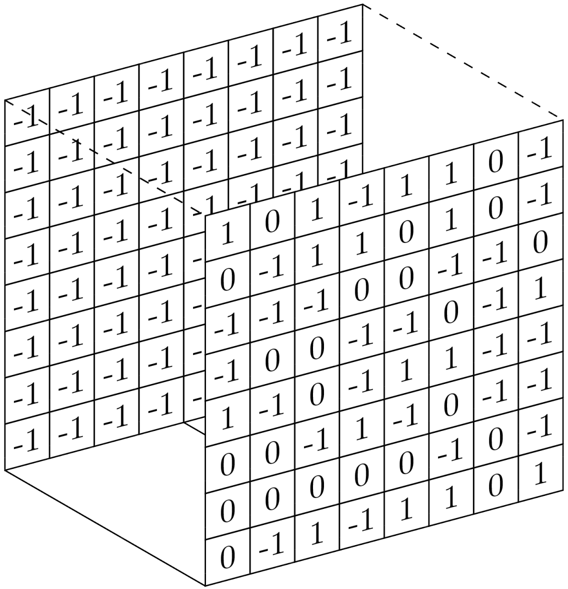
“Classic” classification labelling:

- One-hot
  - [1, 0], if black is winning
  - [0, 1], if white is winning
- Single binary node
  - 1, if black is winning
  - 0, if white is winning

These are typically used with a cross-entropy error function.

Labelling

Instead we went for...



->

$$\begin{cases} 1, & \text{if } \textbf{black} \text{ is winning} \\ -1, & \text{if } \textbf{white} \text{ is winning} \end{cases}$$

Labelling

Why [-1, 1]?

- Uniform with the input format
- Works neatly with the **mean square error** function

Then why mean square error?

- Much more reliable results
- Good metric on loss (accuracy is hard/expensive to measure)

We later discovered that AlphaGo does the same.

The training data

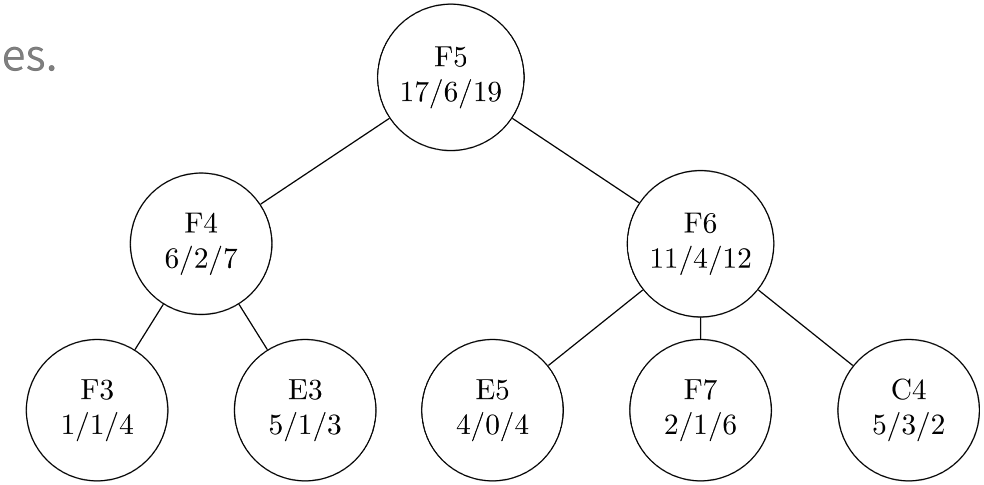
Data from the WTHOR database of tournament games. Currently 122k games.

A tree of states is built where Wins/Ties/Losses are counted.

Label = (Wins - Losses) / (W + T + L)

State used if abs(y) > 0.90.

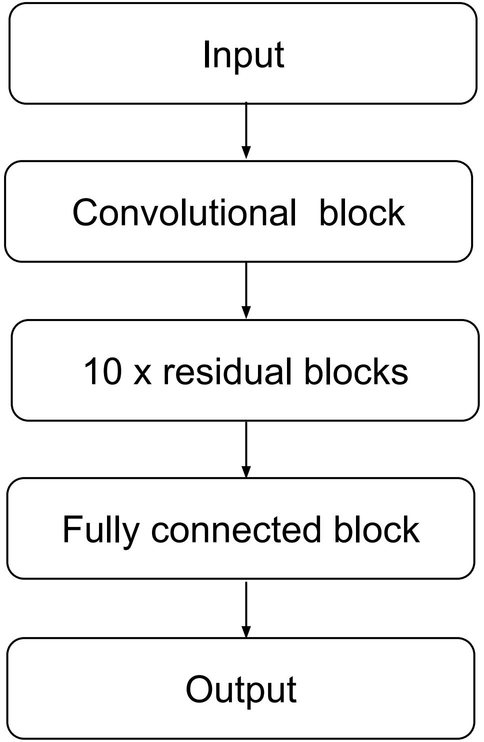
A total of 5.3M unique states, of which 4.8M are used.



The top nodes of a move tree

The network

- A deep residual convolutional neural network
- Residuals mitigates the “vanishing gradient problem” in deep networks
- Same style as AlphaGo, but ~64 times smaller
- Obtained from a lot of trial and error



Implementation

"**Keras** is a high-level neural networks API, written in **Python** and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling **fast experimentation**. Being able to go from idea to result with the least possible delay is key to doing good research."

Implementation

Everything else is done in C and Cython.

Why? Python is **slow**. Cython improves the performance with static typing.

Fun note: for some parts, our C implementation is over **1000x** faster than the Python counterpart.

Putting it all together

Our agent is a minimax search utilizing the trained neural network

To evaluate its performance, we also implemented

- A game simulation function
- Some competitors

Our main competitor

A Static Weight Heuristic Function.

By Sannidhanam and Annamalai at University of Washington.

$$\begin{bmatrix} 4 & -3 & 2 & 2 & 2 & 2 & -3 & 4 \\ -3 & -4 & -1 & -1 & -1 & -1 & -4 & -3 \\ 2 & -1 & 1 & 0 & 0 & 1 & -1 & 2 \\ 2 & -1 & 0 & 1 & 1 & 0 & -1 & 2 \\ 2 & -1 & 0 & 1 & 1 & 0 & -1 & 2 \\ 2 & -1 & 1 & 0 & 0 & 1 & -1 & 2 \\ -3 & -4 & -1 & -1 & -1 & -1 & -4 & -3 \\ 4 & -3 & 2 & 2 & 2 & 2 & -3 & 4 \end{bmatrix}$$

* https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/RUSSIA/Final_Paper.pdf

Results

Analyzing minimax performance, counting SWHF wins:

SWHF

Score

MM depth	1	2	3	4
1	71%	86%	93%	98%
2	59%	82%	91%	96%
3	53%	75%	81%	95%
4	44%	69%	78%	90%

Results

The evaluated agent uses a **minimax depth 1** search, trained for 2 epochs.

Win rate against...

Minimax depth	Random	Score	SWHF
1	83%	69%	50%
2	-	66%	39%
3	-	74%	32%
4	-	36%	24%
5	-	36%	17%

Results

Same thing using a **minimax depth 2** search.

Win rate against...

Minimax depth	Score	SWHF
1	95%	83%
2	92%	77%
3	89%	67%
4	85%	64%
5	80%	54%
6	68%	47%

Shouldn’t it be stronger?

- Supervised learning is only as strong as the data trained on
  - Use more/better data
  - Use reinforcement learning
- Computing power is limited, otherwise we could...
  - Use larger nets
  - Do hyperparameter optimization
- We need more time to explore the subject

Future work

- Monte carlo tree search
- Reinforcement learning
- Generalize to play other games

This slide is intentionally left (almost) blank.

Thanks for listening!
Questions?