INTRODUCTION:

Othello is a two-player turn-based board game played on an eight-by-eight grid, much like chess. As such we have a discrete time and discrete state-space setting.
Also like chess, we have black and white pieces, but in Othello, black has the initiative.
From the starting state (point to screen), black has four legal moves - the ones that successfully trap a white piece in between black pieces.
Then white has three legal responses, and so on and so on...
And whoever controls the most pieces at the end wins.

CHALLENGES:

1. Up to this day, 8x8 Othello remains unsolved. How do we assess the outcomes when we don't know what outcomes to look for? Ideally, we would have it play against expert humans but this was not an option for a project of this size.
   Our solution to this was to avoid it by reducing the setting to 6x6 Othello, where it is known that the white pieces have an advantage. As such if we see a lot of wins for the white side, this indicates that we are at least on the right track.
   Furthermore, this also reduces the state-space size from 10^28 to 10^12, which is still a large number, but it greatly lowered the time needed to train the model.
   We also tried 4x4 Othello, but this was too easy to brute-force and as such 6x6 turned out to be a happy middle-ground.

2. Another big challenge is how to balance exploitation/exploration such that we can learn meaningful patterns in a reasonable amount of time without having to go through the whole tree of possible games. Since rewards are so sparse in tasks such as these, we would like the exploration to be selective in such a way that it reaches rewards more often.
   To handle both of these problems, we use the Monte Carlo tree search algorithm in combination with a policy network, which we will talk more about in a couple of slides.

3. Lastly, we have to deal with the agent/opponent setting. How do we set up the reward structure? Should we have to separate policies - one for black and one for white?
   This is solved through self-play and a minimax algorithm, which allows for both sides to play against the same reward structure, thereby requiring only one policy, and one set of values. So, essentially, we are not training the agent to play as black or white. We are training it to play Othello against itself.

NEURAL NETWORKS:

Value network:

For the value network, we input a board position to a ResNet inspired architecture with two residual blocks that incorporates convolutions, batch normalization, ReLU activations, and skip connections. Each convolution output was held at a constant 6x6x64 before it, at the end was flattened and sent to a linear output function, predicting the value.

Policy network:

The architecture for the policy network was simply a fully connected feedforward network with a flattened board position as input, and a probability distribution over the legal actions from that board position as output.

Training the networks:

Since it was difficult to find 6x6 Othello experience, the initial training of the value network was based on randomly simulated games. This value network was then used in combination with the MCTS to generate new games, from which we trained the policy network. Here are the initial training curves for the networks.

Training cycle:

Now, with the networks in hand, we can define a semi-online training cycle for the agent. We simulate games from the MCTS with the initial networks, save the games in a replay buffer, and after a set amount of episodes, send this replay buffer into training of the neural networks, which then get sent back to the MCTS where a new tree is built.

Here are some of the training curves when we performed the training...