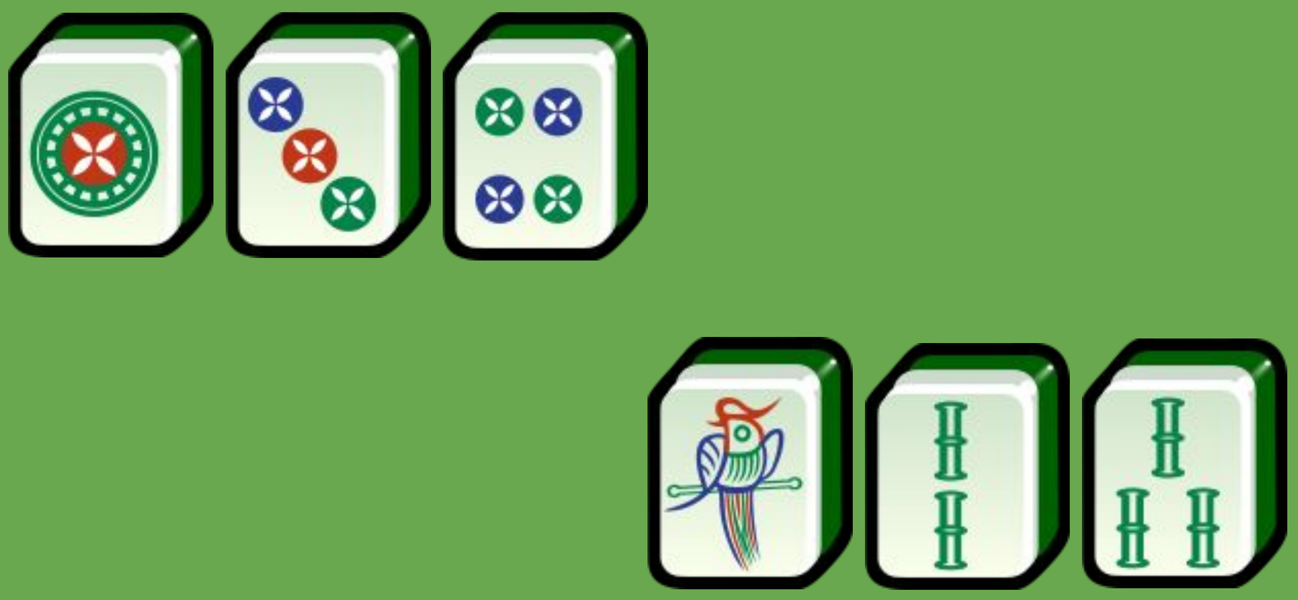
# AI Mahjong



CS221 Fall 2019 | Matthew Trost, Lisa Liao, Joe Wang

## Introduction

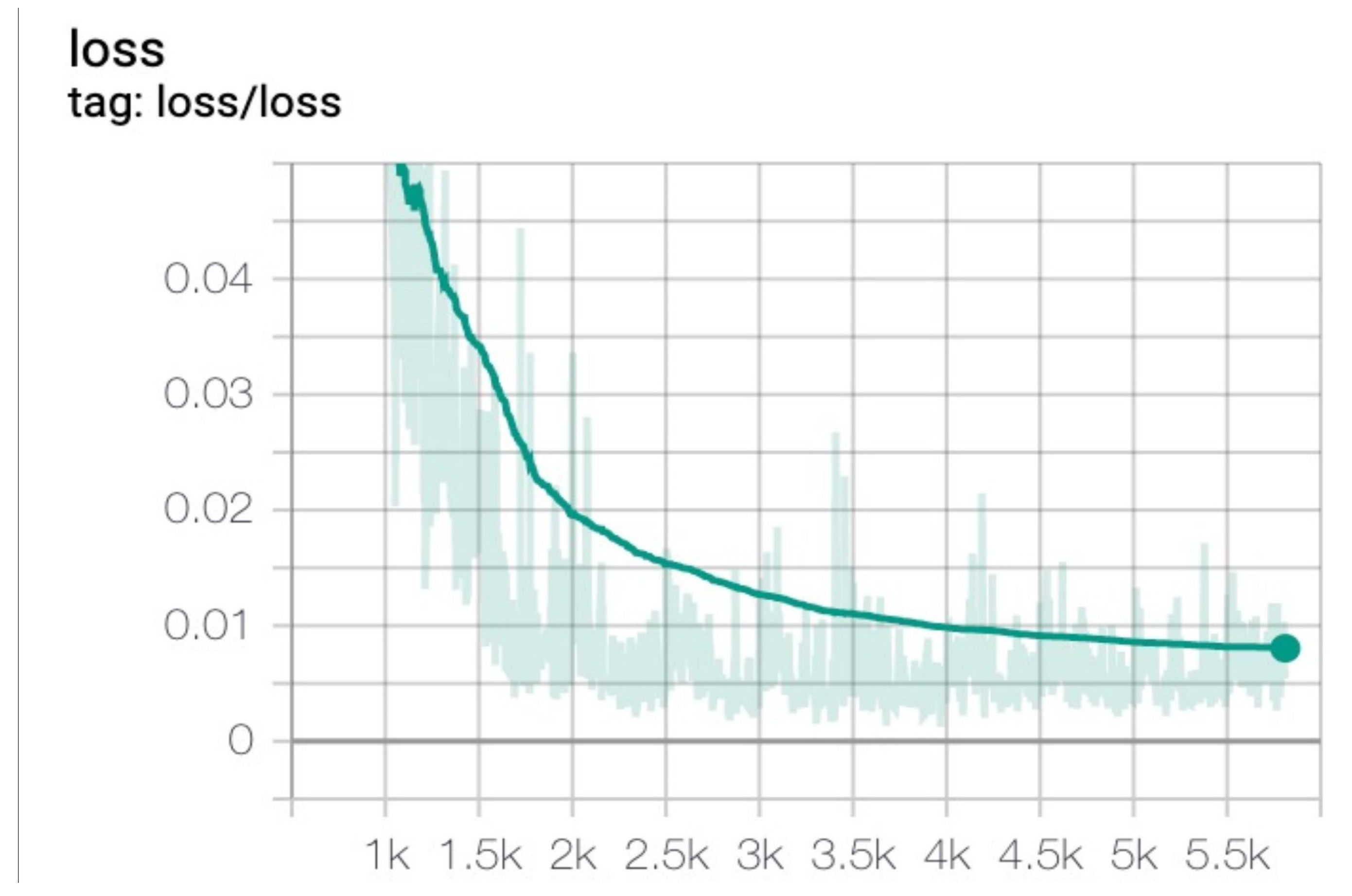
Our project goal is to create a Mahjong-playing agent that utilizes Q-learning to intelligently make decisions in playing against its opponents.

**Game rules:**
Everyone starts with 13 tiles. On each turn, you draw one tile from the deck and either:
1) win (have 4 groups of 3 and 1 pair).
2) discard 1 tile.

**Input:** 13 tiles in a Mahjong hand, plus 1 drawn tile (randomly served up by the deck).

**Output:** 1 tile to discard on each turn of each player/agent.

**Loss over time for Deep Q-Learning training:**



## Methods (Model and Algorithms)

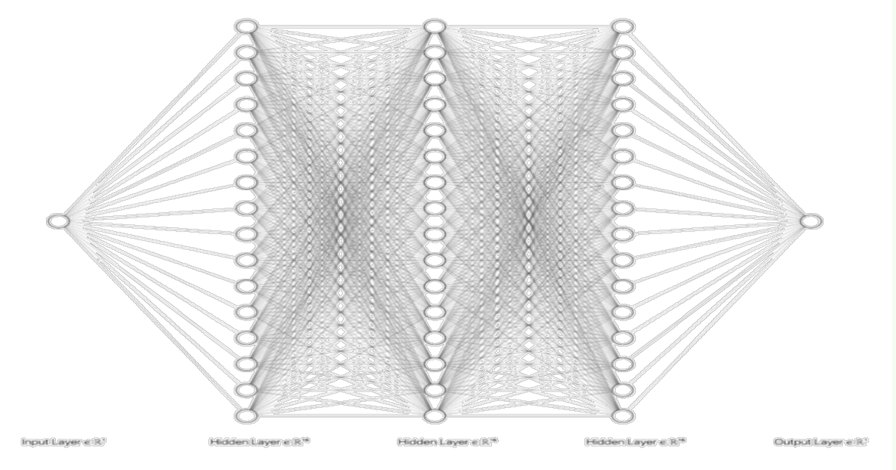
**Model:** Our model of the Mahjong game environment contains 1) the game engine itself which is implemented through a customized OpenAI Gym environment. 2) the agents that play the game, which are trained based on various reinforcement learning algorithms.

**State Feature Vector (for Q-Learning):** For each of 14 tiles in a player's hand, the state vector includes information about number of duplicates in hand and number of discarded duplicates for the nearest four (-2,-1,0,+1,+2) adjacent tiles including current tile. The complete state vector is (14 X 5 X 2). These features tell the system our human knowledge about what patterns are significant.

**Action:** 1 tile in the hand (1-14) to discard.

**Deep Q-learning Neural Network Hyperparameters:**

- learning policy: MultiLayer Perceptron (MLP)
- gamma=0.99
- learning_rate=0.0005
- batch_size=32



## Discussion

Since we are training a game-playing agent without labeled datasets, we need to use a Reinforcement Learning algorithm. However, tabular Q-Learning would not work for mahjong because with 34 unique tile types and 14 tiles in each hand, the number of game states is upper-bounded by $34^{14}$, which is far too many to tabulate.

Instead, we started with a linear function approximator, which dotted a feature vector with learned weights. Unfortunately, this was not sufficiently expressive. Our initial feature vector only indicated the number of each tile type in our hand, so it could not express that 3 of one tile is in general far more than 3 times as valuable than 1 of that tile.

Furthermore, we noticed that all of our games were ending in draws; i.e. no agent was able to acquire a winning hand before all tiles were drawn. A certain number of draws is expected because in our modified version of mahjong agents are not allowed to "steal" opponents' discarded tiles, making it harder to gather the tiles they need. However, a 100% draw rate indicates that the agent is not playing intelligently. We changed our vector and replaced the simplistic linear function with a neural network. This Deep Q-Learning algorithm only draws 10% of the time.

## Results

We set up our agent to play against 3 randomly-discarding "baseline" agents. For the first 100 or so games there were no winners; all tiles were drawn before a player achieved a winning hand. Suddenly, however, our agent began winning around 90% of the time, and it has remained at 90%, with the other 10% being draws. The graph above shows the loss over time as the Deep Q-Learning agent trained.

## Conclusion

Overall, we have built a simple agent that can play a modified version of Mahjong independently. Our agent already reliably beats random players, so we will test it against ourselves to better evaluate its current ability level. To achieve further performance gains, we want to implement a version of the agent that learns via Double Deep Q-learning so our agent can learn more accurately through the training process.