# Trick Takers

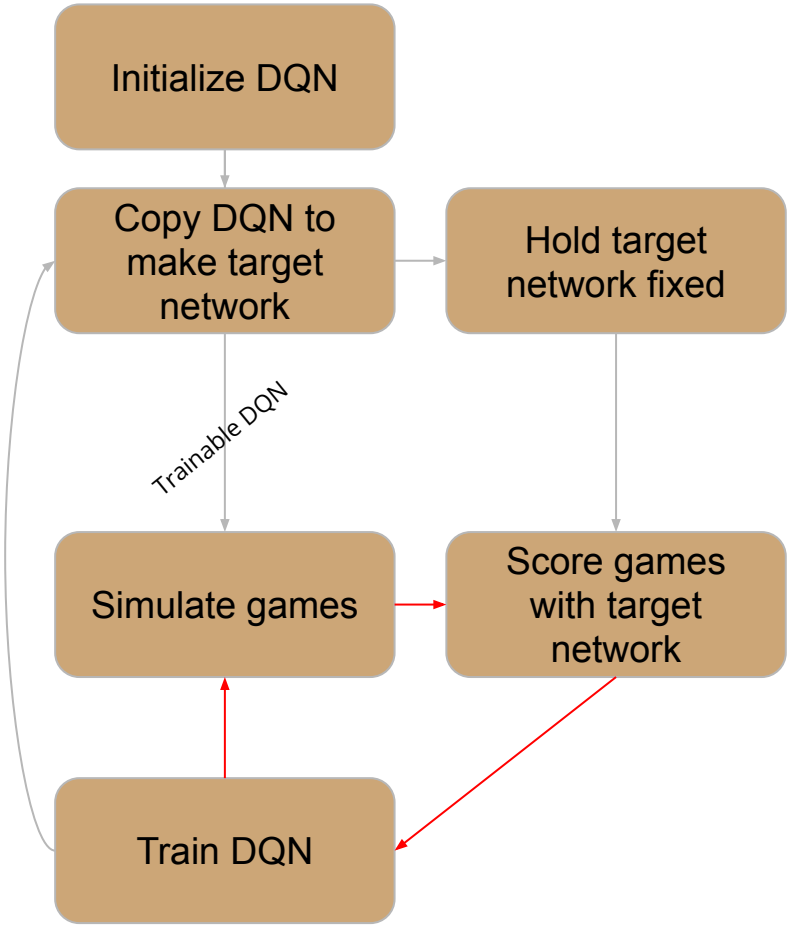
Shin Kim, Juergen Kritschgau, Sixuan Lou,
Edward Varvak, Yizhen Zhao

# Objective

- Train agents to play trick taking card games, such as Spades
- Model Spades as a sequence of decisions leading to a win or loss
- Apply a deep Q-network to optimize game strategy

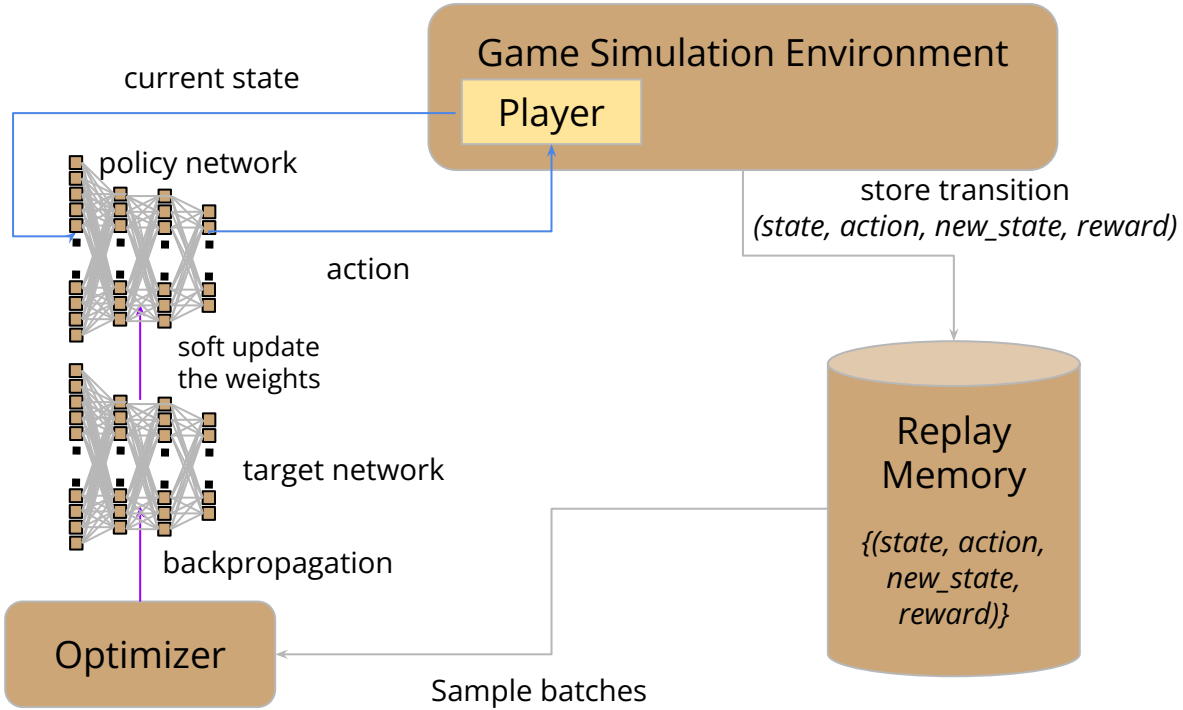
# Deep Q Learning

- DQNs learn to approximate an unknown Q function which computes the optimal discounted cumulative reward from a given state and action
- We know that Q function satisfies the relation Q(s,a) = r +$\gamma$max Q(s',a')
- Training minimizes the error between predicted value Q(s,a), and the target value r + $\gamma$max Q(s',a')



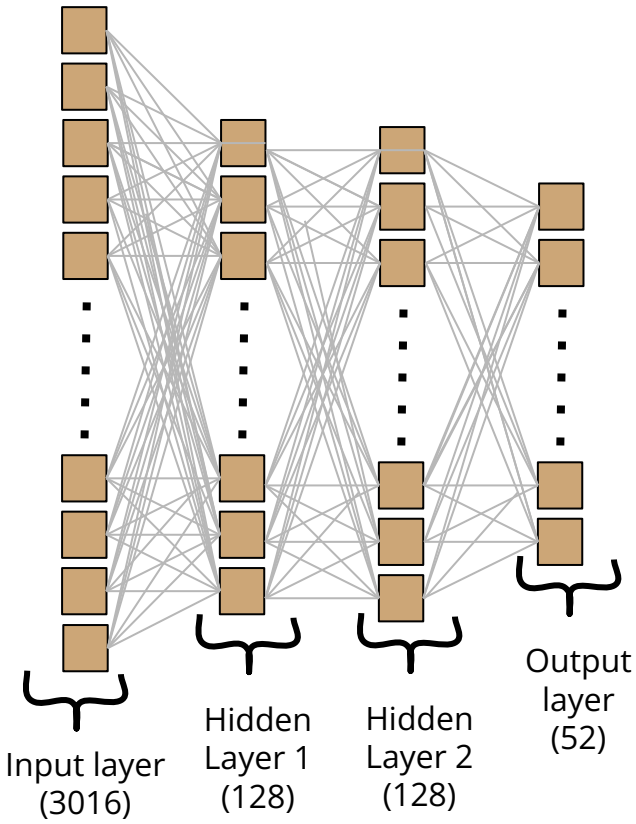
# Learning Framework

- Each episode of the training starts a new game
- Player asks the policy network each turn for an action (ε-greedy)
- The environment computes the next state and reward given the action
- Reward = 1 if the player wins this trick, Reward = 0 otherwise
- Store this transition and perform a backpropagation using the loss of Q-functions



# Network Architecture

- The network takes inputs of length 3016, consisting of the hand (52), the cards played (56 x 52), and the cards not yet seen (52)
- The vector is then passed through two fully connected hidden layers
- The output layer has length 52, with each element denoting the network's estimate for the Q-value of playing the corresponding card



# Hyperparameter Tuning

Recall that the value of a state-action pair

$$Q(s, a) = r + \gamma \max Q(s', a')$$

- When $\gamma$ was set to 1, we found that the estimated Q-function diverged
- With $\gamma = 0$, the network was prone to short-term thinking, and failed to learn beyond a certain threshold

- Through experimentation, we found that the optimal value for $\gamma$ was 0.3

# Results

- The results for the agent without knowledge of which moves are legal.
  - The agent was able to play for 8.5 turns on average before playing an illegal move in 100 simulated games
- The results for the agent which selects legal moves that maximize reward.

```
simulate_with_network('weights/ev_q_function_output_outside_folder.pth', 1000)
```

```
Simulated 1000 games.
Average reward per game: 3.845
```

- The performance of the baseline models.

```
simulate(lambda game : random_agent(game, None), 1000)
```

```
Simulated 1000 games.
Average reward per game: 3.308
```

```
simulate(greedy_policy, 1000)
```

```
Average reward per game: 3.8
```

# Future Directions

- Train agents to play with bidding strategies: Optimize the agent's decision making about how many tricks to bid based on its hand
- Transfer learning to other trick taking games: Apply the knowledge to similar games like Hearts or Bridge
- Scaling to complex game states: Increase the agent's ability to handle more complex game situations
- Further train the agent against itself, to see if we can improve its performance