Executive Summary for Trick Takers

(Deep Learning Bootcamp Summer 2024, The Erdos Institute)

Team Members :
Shin Kim, Juergen Kritschgau, Sixuan Lou, Edward Varvak, Yizhen Zhao

Overview :

In reinforcement learning problems, the agent learns how to maximize a numerical reward signal through direct interactions with the environment and without relying on a complete model of the environment. In fact, agents using model free methods learn from raw experience and without any inferences about how the environment will behave. An important model free method is the use of the Q-Learning algorithm to approximate the optimal action value function. However, it can be impractical to estimate the optimal action value function for every possible state-action pair. Deep Q-Learning uses a neural network trained with a variant of Q-Learning as a nonlinear function approximator of the optimal action value function. Our objective is to use Deep Q-Learning to train an agent to make legal moves and/or win tricks while playing the card game Spades (without bidding).

Methodology and Results :

Creating the Environment :
We created an environment that can randomly initialize and play through games, keep track of the states, and stop the game when the agent plays an illegal move. At each stage in the game, the game state contains information about the cards in the current player's hand, the tricks each player took, the cards that have been played and the turns in which they were played, and the cards that have not been played. Each card is one-hot encoded as a vector of length 52 and the game state is a vector of length 3016. When the agent plays a card, the other players randomly select a legal move on their respective turns.

We allowed the agent to change the environment in one of two ways. The agent feeds the input vector of length 3016 to the neural network and receives a vector of length 52 consisting of the action values. Then, the agent either chooses an action with the maximum action value or chooses a legal action with the maximum action value. In the former case, the agent may choose an illegal move and the game terminates at the next state if the agent does so. In the latter case, the agent always makes legal moves and only needs to learn how to maximize the number of tricks taken.

Architecture of the Neural Network :
The network consists of two hidden linear layers each of which has 128 nodes and is followed by a rectifier activation. The output is a vector of length 52 and each element of the vector is the estimated value of the action value function when playing the card of the corresponding index given the state of the game specified by the input vector.

Implementing Replay Memory and Optimizing Parameters :
The replay memory is a list of 13 double ended queues (or deques) each of length 10000. An element in the i-th deque is a tuple, called a transition, consisting of the state of the particular game at the agent's i-th turn, the action that the agent took at that state, the reward the agent received for taking the action, and the state that followed the agent's action. The agent selects an action based on an epsilon-greedy policy with a

decaying epsilon. Each time the agent selects an action, the corresponding transition tuple is appended onto the appropriate deque. We use the optimizer AdamW to update the parameters. The optimizer uses a batch of 100 transitions that are randomly sampled from the deques that are sufficiently filled. This ensures that the transitions that the agent uses to learn are not serially correlated. Additionally, the stratification of the replay memory described above helps the agent learn from different stages of the game. In particular, this prevents the agent from only sampling transitions corresponding to the first few turns of the games which take up most of the replay memory in the early stages of training.

Optimizing Hyperparameters :
One of the more sensitive hyperparameters was the discount rate (gamma). The gamma value was especially sensitive when the agent selected actions without regard to whether or not the action is legal. For gamma values close to 1, the values of the estimated action value functions tend to blow up in magnitude resulting in poor agent performance. For gamma values close to 0, the agent was able to learn the rules of the game very well resulting in an average duration of around 11 turns for 100 simulated games. To reach the balance between learning to play legal moves and looking further into the future to maximize the value function, we decided a discount rate around 0.3 to be optimal.

Results :
In the case when the agent selected actions based on the network output alone, the agent was able to play for 8.5 turns on average in 100 simulated games. When the agent only selected legal actions, the agent was able to take 3.7 tricks on average in 1000 simulated games. To evaluate this performance, we implemented a one turn greedy algorithm where the agent chooses the card with the highest chance of winning the trick at each turn. This baseline model was able to take 3.8 tricks on average in 1000 simulated games.

Future Directions :

- So far, the agent played against other players that select a random legal move when it is their turn. It would be interesting to see how the agent learns when we allow for the other players to select actions based on a trained neural network.
- Given that the inputs to the neural network consist entirely of one hot encoded information, we opted to use linear layers. We also kept the network depth fixed to keep the training time relatively short. We can experiment with the network architecture to find one that optimizes agent performance.

References : https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html