Chess Assignment

Shiran Sun
shiran.sun@uzh.ch

I. Introduction

In this assignment, I firstly introduced two reinforcement learning algorithms, Q-learning and SARSA, and described the difference between them. Then I implemented a Q-learning ANN to solve a chess problem, observed the learning curve of reward and number of moves, and adjusted some parameters to see the change of learning curve. Finally, I applied another Deep SARSA network to this problem and observed the results.

The link to my GitHub project is https://github.com/sunshiran171250501/intro-rl-assignment.

II. Methods

A. Q-Learning

Q-learning is a reinforcement learning algorithm for Markov decision process that learns the optimal policy by taking immediate and future rewards into account. Q-learning does not require knowledge of the environment model. It holds a Q-table of state-action value, observes rewards, and updates the Q-table after each step.

Q-learning is an off-policy learning algorithm. It takes the maximum Q value of the next state as the future reward to update the current Q value. The future reward is scaled down by a discount factor γ, and sum up with the immediate reward to form the estimation. The Q value is updated by adding the error of estimation multiplies the learning rate η. The update function of Q-learning is as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta[R + \gamma \max_A Q(S_{t+1}, A) - Q(S_t, A_t)] \quad (1)$$

B. SARSA

State–Action–Reward–State–Action (SARSA) algorithm is another model-free reinforcement learning algorithm. Like Q-learning, SARSA also maintains a Q-table and learns while interacting with the environment. However, when updating Q value, SARSA calculates the next Q value by applying the same policy to choose the next action, instead of choosing the maximum Q value of next state. This makes it an on-policy algorithm.

The update function of SARSA is as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta[R + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (2)$$

C. Differences between the Two Methods

The strategy to estimate future reward is different between Q-learning and SARSA. As described above, Q-learning is an off-policy algorithm, while SARSA is an on-policy algorithm. The estimation policy of Q-learning is always greedy, while the esitmation and action policy of SARSA are the same. The action SARSA uses to update Q value will be actually taken in the next step.

As a result, Q learning always learns the optimal path. SARSA will otherwise choose a safer path, not the optimal one.

III. Results

A. Q-Learning

To solve the checkmate with a king and queen problem, I implemented a Q-learning algorithm with artificial neural network (ANN). The network consists of an input layer, an output layer, and a hidden layer of 200 neurons. All the layers are fully connected, and the activation function is ReLU for each layer.

The environment is a $N \times N$ chess board with $N = 4$. The state is presented by a $3N^2 + 10$ dimensional vector, which contains the location of the player's king, queen and the opponent's king, the 8 degree of freedom of the opponent's king, and whether the opponent's king is threatened, all of the features one-hot encoded. The action space is $8N$, for there are $8(N-1)$ possible moves of queen, and 8 possibilities of king. The action vector is also one-hot encoded with illegal moves masked by zeros. Therefore, the input and size of the network is 58, and the output size is 32.

Apply the ε-greedy policy to select actions with $\varepsilon = 0.2$ at the beginning, and ε is declined by factor $\beta = 0.0005$ after each step. The learning rate $\eta = 0.0035$, and discount factor $\gamma = 0.85$.

After $N = 100000$ episodes, the results are shown in the following plots. The reward per game is plotted in 1, and the number of moves per game is plotted in 2. I applied exponential moving average to make the plots smoother. The reward and number of moves increases at the beginning, then they both falls to some degree. After that, the reward keep increases, while the number of moves climbs to a local maximum and falls.

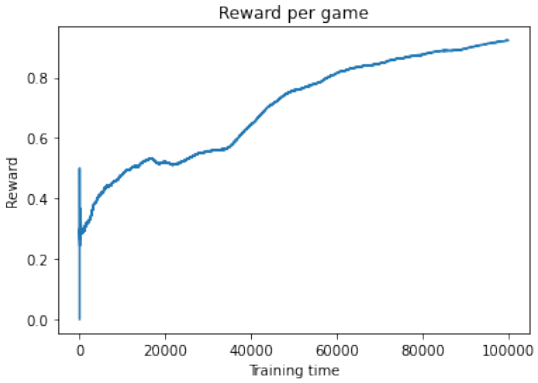

Fig. 1. Reward per game vs training time.

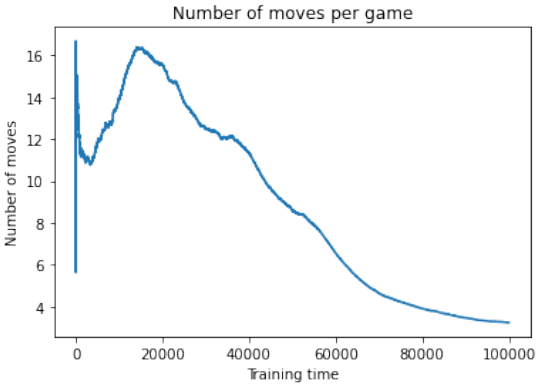

Fig. 2. Number of moves per game vs training time.

B. Changing Parameters of Q-Learning

Next, I did experiments changing the speed β of the decaying trend of ε and the discount factor γ, and see how the average reward and number of moves change.

Increasing the discount factor γ to 0.95, the results are shown in 3 and 4. The reward becomes higher at the end. The number of moves reaches a higher maximum in the learning process.

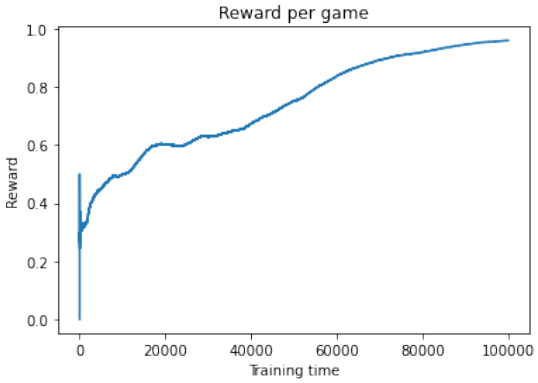

Fig. 3. Reward per game vs training time, γ=0.95.

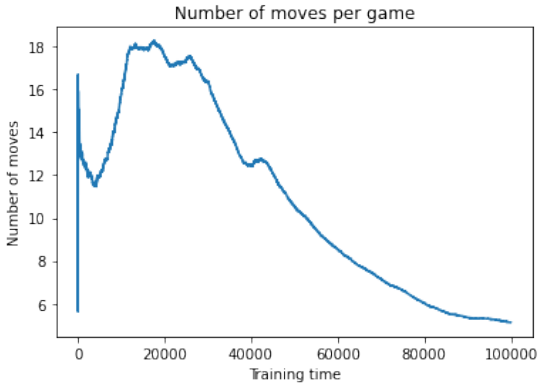

Fig. 4. Number of moves per game vs training time, γ=0.95.

Reducing the discount factor γ to 0.75, the results are shown in 5 and 6. The second crest of number of moves becomes lower.

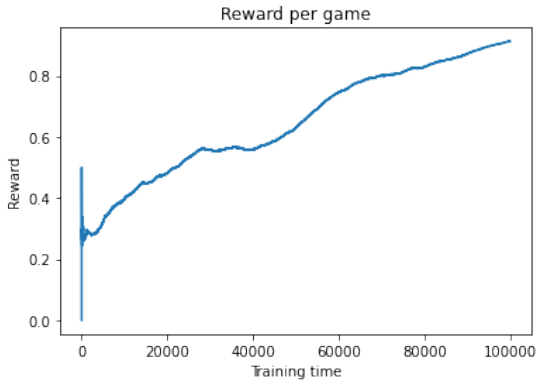

Fig. 5. Reward per game vs training time, γ=0.75.

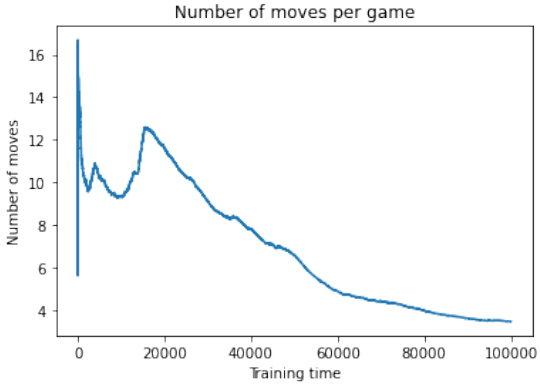

Fig. 6. Number of moves per game vs training time, γ=0.95.

Increasing the speed β of the decaying trend of ε to 0.005, the results are shown in 7 and 8. The final reward becomes higher. The maximal and final number of moves are higher.

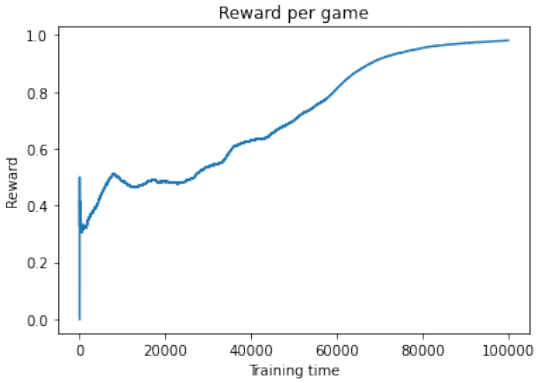

Fig. 7. Reward per game vs training time, β=0.005.

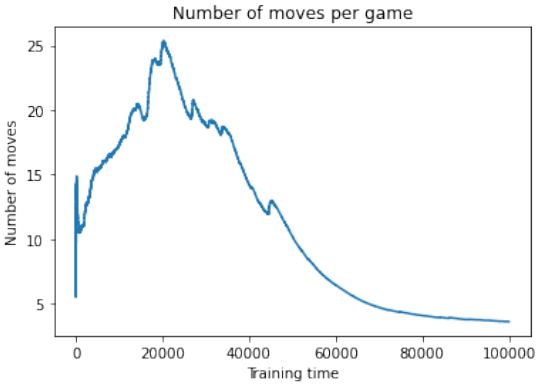

Fig. 8. Number of moves per game vs training time, β=0.005.

Reducing the speed β of the decaying trend of ε to 0.00005, the results are shown in 9 and 10. The final reward becomes lower, and the second crest of number of moves is lower.

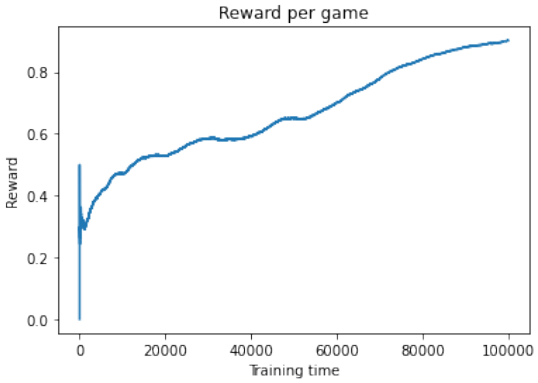

Fig. 9. Reward per game vs training time, β=0.00005.

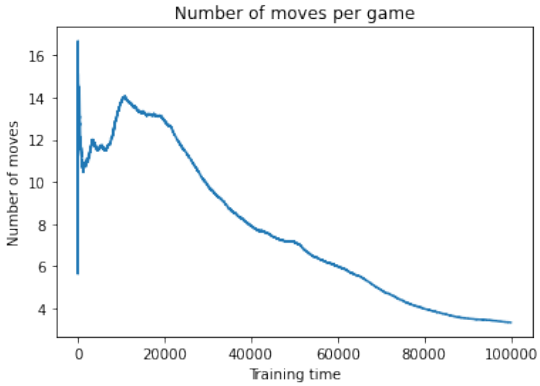

Fig. 10. Number of moves per game vs training time, β=0.00005.

C. Deep SARSA

Then, a Deep SARSA algorithm is implemented to solve the same problem. The Deep Neural Network (DNN) is built by keras using tensorflow backend. I applied 2 dense hidden layers of 64 ad 32 neurons to achieve an acceptable learning speed on my own device, and used a GPU to accelerate the process. The activation function of hidden layers is ReLU.

I also reduced the noise with mini-batches. The size of experience replay buffer is 200, and the batch size is 32. The mini-batches are randomly sampled, and the ε factor declines only after every experience replay process.

After $N = 100000$ episodes, the results are shown in 11 and 12. The learning curve of rewards is smoother than Q-learning with ANN model, and the maximum and final number of moves are also lower.

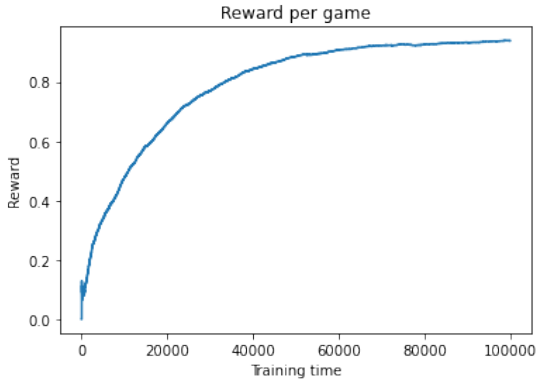

Fig. 11. Reward per game vs training time.

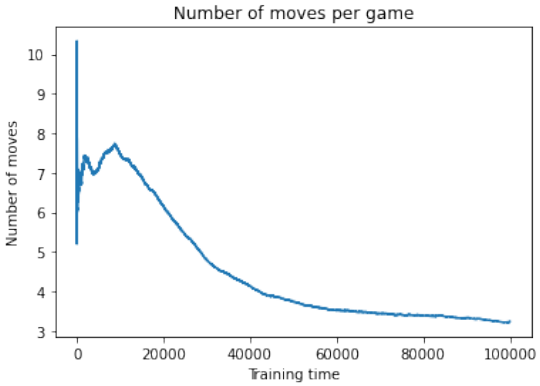

Fig. 12. Number of moves per game vs training time.

IV. Conclusions

There are two different classic reinforcement learning algorithms, the off-policy Q-learning, and the on-policy SARSA algorithm. I implemented a Q-learning algorithm and a deep SARSA network for the chess problem. Both algorithms can solve the problem properly. However, the DNN of SARSA performs better, which takes less moves to finish the game.

Changing the discount factor γ and the speed β of the decaying trend of ε influences the results of Q-learning algorithm. Increasing γ makes the final reward and maximum number of moves higher, while decreasing γ makes the local maximum of number of moves lower. The higher the factor β is, the final rewards, the maximal and final number of moves are higher, and a lower β makes the second crest of numbers of moves lower.