# Course Project

Moon, Hyosik

August 9, 2021

## 1 Required

- **Main objective of the analysis**
  We will optimize the 'CartPole-v0' problem with three different models such as 'RandomForestRegressor, AdaBoostRegressor, ExtraTreesRegressor'. The goal of the problem is to prevent a pole attached to a cart from falling over by increasing and reducing the cart's velocity. Whenever it successes to mainatain the pole, it gets a socre as a reward. We win the game, if we maintain the pole for 200 times.

- **Brief description of the data set**
  Because this is reinforcement problem we need to produce our own data set. In order to get the data set, I implemented the game 10000 times. We can check the data in the figure 1. Four observations represent 'Cart Position', 'Cart Velocity', 'Pole Angle', and 'Pole Angular velocity' respectively.

| | obs0 | obs1 | obs2 | obs3 | action | reward | episode | tot_reward |
|---|---|---|---|---|---|---|---|---|
| **0** | -0.009359 | 0.047097 | -0.022057 | 0.015836 | 0 | 1.0 | 0 | 21.0 |
| **1** | -0.008417 | -0.147701 | -0.021740 | 0.301479 | 0 | 1.0 | 0 | 21.0 |
| **2** | -0.011371 | -0.342507 | -0.015711 | 0.587227 | 1 | 1.0 | 0 | 21.0 |
| **3** | -0.018221 | -0.147168 | -0.003966 | 0.289637 | 1 | 1.0 | 0 | 21.0 |
| **4** | -0.021164 | 0.048010 | 0.001827 | -0.004294 | 0 | 1.0 | 0 | 21.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **21746** | 0.094145 | -0.354251 | -0.199186 | 0.136198 | 1 | 1.0 | 999 | 25.0 |
| **21747** | 0.087060 | -0.156916 | -0.196462 | -0.212125 | 0 | 1.0 | 999 | 25.0 |
| **21748** | 0.083921 | -0.348766 | -0.200704 | 0.012725 | 1 | 1.0 | 999 | 25.0 |
| **21749** | 0.076946 | -0.151417 | -0.200450 | -0.335969 | 1 | 1.0 | 999 | 25.0 |
| **21750** | 0.073918 | 0.045909 | -0.207169 | -0.684570 | 0 | 1.0 | 999 | 25.0 |

21751 rows × 8 columns

Figure 1: Data set information

- **Brief summary of data exploration** In order to train a model, we need the pole's position, velocity, angle, angular velocity, and actions as x variable and reward as y value. So I made a dataframe and extract x and y from the dataframe in the figure 1.

- **Summary of training at least three models** I implemented the game with three different prediction models which are 'RandomForestRegressor, AdaBoostRegressor, and ExtraTreesRegressor'.

    1. RandomForestRegressor. It showed the relatively good result. The average step maintaining the pole was 157, with a perfect score of 30 out of 100 trials. (Figure 2)

```
In [41]: r_memory_df.groupby("episode").reward.sum().mean()
Out[41]: 157.18

In [119]: (r_memory_df.groupby("episode").reward.sum() >= 200).value_counts()
Out[119]: False    70
          True     30
          Name: reward, dtype: int64
```

Figure 2: RandomForestRegressor

2. AdaBoostRegressor. The result was terrable. Because in the case of AdaBoost it modifies the weights sequencially with the wrongly predicted data set. Since our data set consists of randomly chosen episodes, the AdaBoost model might be inappropriate to train the this data. (Figure 3)

```
In [118]: a_memory_df.groupby("episode").reward.sum().mean()
Out[118]: 9.35

In [120]: (a_memory_df.groupby("episode").reward.sum() >= 200).value_counts()
Out[120]: False    100
          Name: reward, dtype: int64
```

Figure 3: AdaBoostRegressor

3. ExtraTreesRegressor. It showed the relatively good result. The average step maintaining the pole was 138, with a perfect score of 17 out of 100 trials. (Figure 4)

```
In [41]: r_memory_df.groupby("episode").reward.sum().mean()
Out[41]: 157.18

In [119]: (r_memory_df.groupby("episode").reward.sum() >= 200).value_counts()
Out[119]: False    70
          True     30
          Name: reward, dtype: int64
```

Figure 4: ExtraTreesRegressor

- **Explanation of your final model** Overall, the RandomForestRegressor showed the best performance. I think that this is because we used closely related features to predict the result which are 'Cart Position', 'Cart Velocity', 'Pole Angle', and 'Pole Angular velocity'. Thus, the result of RandomForest is better than ExtraTreesRegressor (ETR) because ETR trains the model with randomly chosen features not with all features. In order to increase the performance, I trained the final model (RandomForestRegressor) with more data which have 20000 trials. And the final result shows that the average step maintaining the pole was 157, with a perfect score of 35 out of 100 trials. (Figure 5)

```
In [11]: r_memory_df.groupby("episode").reward.sum().mean()

Out[11]: 157.15

In [12]: (r_memory_df.groupby("episode").reward.sum() >= 200).value_counts()

Out[12]: False    65
         True     35
         Name: reward, dtype: int64
```

Figure 5: Final model score

- **Summary Key Findings and Insights** In order to increase the performance, using deep neural network can be another way. Thus I thought about two neural-networks models such as deep neural network, and recurrent neural network for the advanced steps. But in the case of RNN, it seems not to be good to improve the performance because the model is trained based on the episodes which have the same comb_rewards which is not so related to sequential events. So, I abandoned the RNN. Instead of it, I implemented the Deep Neural Network manually with two hidden layers with (5, 64), (64, 1) dimentions. In the figure 6, y-axis shows the error rate. Even though the error rate decreases the total performance was much lower (Average step is 9). I think that this is because when the NN was trained the weights were modified based on individual episode's comb_rewards which has the same value at the same episode. So, in order to improve the model, the comb_rewards need to be changed. To be specific, within the same episode, according to the steps increase, the rewards also need to be increased. After I changed the rewards, I was able to check the improved result like represented in the figure 7.

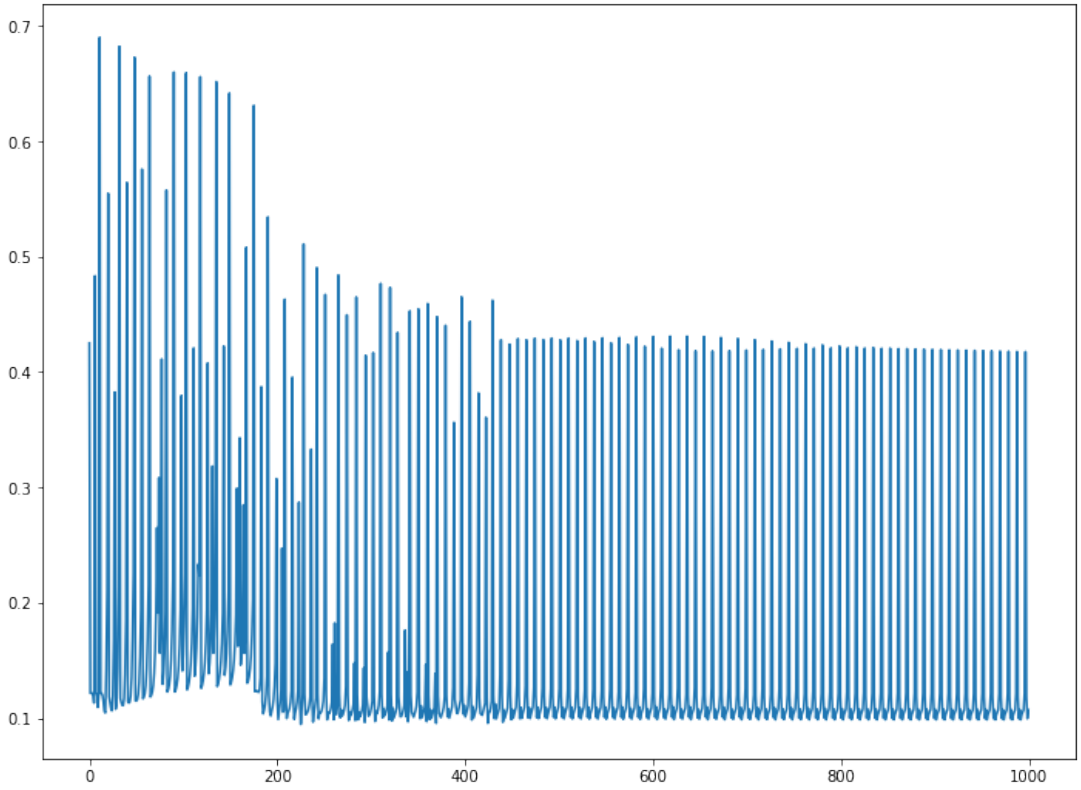


Figure 6: Error of Deep Neural Network model

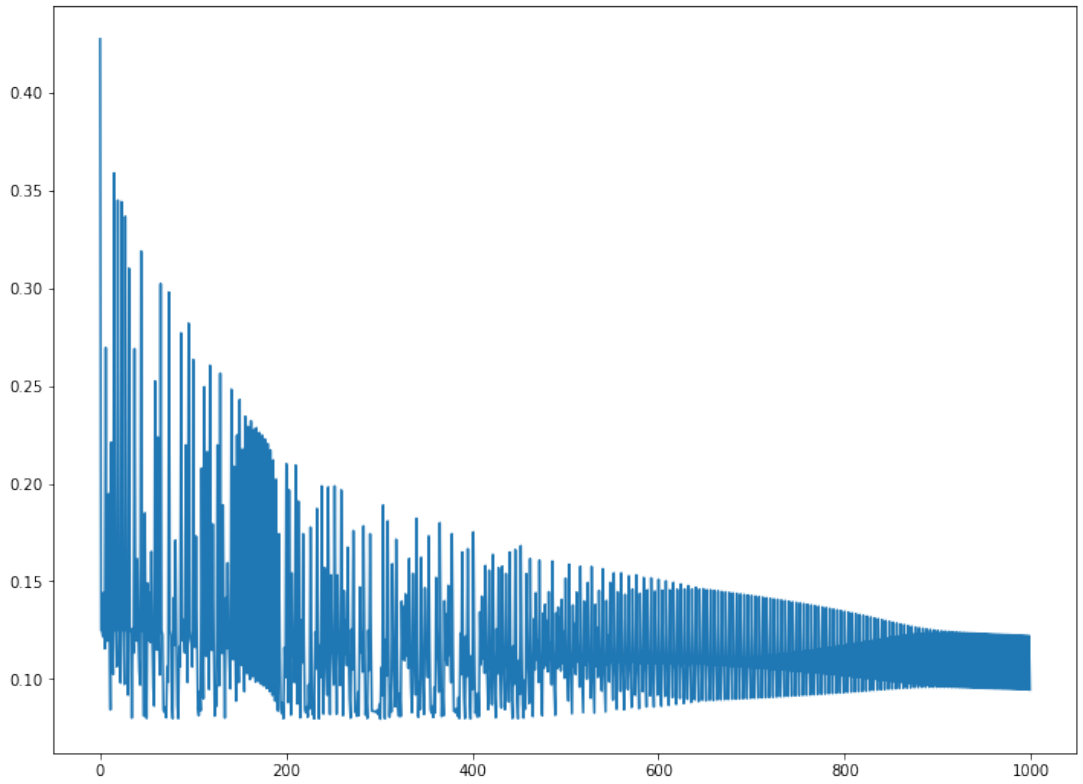


Figure 7: Improved version of DNN

- **Suggestions for next steps** When I checked the final result, even though the prediction error was low, the performance was still bed (Average step is 10). I need to find the reason of the poor performance of the Deep Neural Network model.