The Chinese University of Hong Kong

FINAL YEAR PROJECT REPORT (TERM 2)

Applying Reinforcement Learning to “Play” Horse Racing

Author:

OR Ka Yui

Supervisor:

Prof. Michael R. LYU

LYU2003

Faculty of Engineering

Department of Computer Science and Engineering

April 21, 2021

THE CHINESE UNIVERSITY OF HONG KONG

Abstract

Faculty of Engineering

Department of Computer Science and Engineering

BSc degree in Computer Science

Reinforcement Learning in Horse Racing

by CHEUNG Kam Fung, OR Ka Yui

Reinforcement Learning has become popular since the appearance of Alphago. It has been applied to multiple areas in the world and now we would like to study and apply reinforcement learning in horse racing, which is the most famous gambling event in Hong Kong. In this report, we collect data from Hong Kong Jockey Club and Hong Kong Observatory. After analysing them, we first use XGBoost regressor to build a model to predict the horse racing result. Then, we will construct the Deep Q Network model using the result of XGBoost and conclude our work. We also use more advanced models, such as PPO and A2C.

Acknowledgements

We truly appreciate our supervisor Professor Micheal R. Lyu and advisor Mr. Edward Yau for guiding and supporting this project.

Also we would like to express our special thanks to the previous project studying reinforcement learning and horse racing. They give us a good reference to our project “Reinforcement learning in horse racing”.

Content Page

Chapter 1 Introduction

1.1 Overview

The topic of this final year project is trying to learn how to bet on horse racing with rein, throughout this report we will demonstrate the work done during the first semester. This chapter offers a brief overview of this final year project and introduction to the topic. Moreover, it provides related work and previous approaches on the horse racing predictions. In the end, it introduces the difficulties in predicting horse racing results and the coming objective in the second semester.

1.2 Motivation

Nowadays, artificial intelligence has become the most popular technique in the world. As one of the areas of artificial intelligence, machine learning is applied to many different uses in the world, and many softwares and hardwares have born because of machine learning. For example, many CPUs and GPUs have claimed that they are more powerful in machine learning. The most popular electric car company in the world, Tesla, has also developed the Autopilot AI using deep neural networks [1]. Google created the best AI chess player Alphago which beat all human chess players [2]. The trend of machine learning attracts us to study about machine learning and apply machine learning to one of the most popular events in Hong Kong, horse racing. The gambling of horse racing can bring lots of money to us. This attracts us to study horse racing.

Before this project, there are the other projects to apple machine learning in horse racing. For example, LYU1603 Final Year Project used Tensorflow to predict the

horse racing result [3]. LYU1703 Final Year Project used a deep neural network to predict the result [4]. LYU1805 Final Year Project used deep probabilistic programming [5]. All of them used supervised learning to predict the result. Instead of supervised learning, we are going to use reinforcement learning.

Reinforcement learning becomes famous in these years due to Alphago who has used reinforcement learning. Before that, people have used reinforcement learning to play different games. However, no one has ever used reinforcement learning to predict the result of horse racing or teach the AI to bet. That is why we would like to apply reinforcement learning in horse racing.

1.3 Backgound

1.3.1 Horse Racing

Horse racing has been popular for many years in Hong Kong. This event started in the 1840s when Hong Kong was not yet transferred to China [6]. With the complete and strict law of gambling, Hong Kong citizens love betting in horse racing. In 2019/2020, Hong Kong Jockey Club has recorded the total amount bet of horse racing by customers is around 121.0 billion [7].

There are multiple components of horse racing including racecourse, system, horse, jockey, trainer and wager [8].

1.3.1.1 Basic of Horse Racing

There are two main racecourses in Hong Kong. They are Sha Tin Racecourse and Happy Valley.

	Mainly Host	Number of races per race meeting	Track	Max. Number of starters
Sha Tin Racecourse	Sunday day races	10	Turf of All-Weather	14
Happy Valley Racecourse	Wednesday night races	8	Turf	12

The system of horse racing includes season, distance, class, rating, weight and draw. The race season is from September to July with 99 race meetings. Distance can be classified as short, middle and long.

Short	1000M	1200M	1400M
Middle	1600M	1650M	1800M
Long	2000M	2200M	2400M

The horses are classified from Class 1 to Class 5 where Class 1 is the highest class and Class 5 is the lowest. A new horse starts with 52 rating. Around 5 to 7 points will be added to the rating for each win. First 4 in the race will add points while losing in the race will deduct points.

Rating	Class
>=85	1
>=80 & <=100	2
>=60 & <=80	3
>=40 & <=60	4
>=0 & <=60	5

There are handicaps which means runners carry different weights to equalise their chances of winning. The higher rating horse carries more weight. Also, with more additional weight, the horse will get more rating if it wins.

The draw refers to the horse's starting position in the starting gate. The smaller number of the draw means the closer to the inside rail. The horse may get a different advantage from the draw depending on the horse's characteristics.

The jockey rides the horses according to the trainers' instruction. The trainer trains and cares for the horses and manages operation. They all get licenses from Hong Kong Jockey Club.

1.3.1.2 Bet

Odds decides how much you get if you win. Usually, higher odds mean lower chance of winning. For example, if the win odds is 1.5 and you win with $10, you will get the dividend $15 back including the cost $10. There are two types of odds which are Win odds and Place odds. Now, the betting type of horse racing:

Simple bet:

Type	Condition of winning
Win	Pick the winner
Place	Pick any one of the first three horses
Quinella	Pick the first and second horses in any order
Quinella Place	Pick any two of the first three horses in any order
Banker	Pick a banker which must be in all possible combinations and three other selections to pair with the banker. The Quinella combination gives a dividend.
Multiple	Pick 4 horses and there will be 6 combinations. Each Quinella Place combination gives a dividend.
3 Pick 1	The horses are in 3 groups as {1st,2nd}, {3rd,4th,5th}, others. The dividend is given by any runner from the chosen group wins. The odds are special.

Exotic Bets (Single-race):

Type	Condition of winning
Forecast	Pick the first and second horses in correct order
Trio	Pick the first three horses in any order
Tierce	Pick the first three horses in correct order
First 4	Pick the first four horses in any order

Qurtet	Pick the first four horses in correct order

Exotic Bets (Multiple-race):

Type	Condition of winning
Double Trio	Pick the first three horses in any order in each of the two nominated races
Triple Trio	Pick the first three horses in any order in each of the three nominated races
Double	Pick the winner in each of the two nominated races
Treble	Pick the winner in each of the three nominated races (Only available in the last 3 races of each race meeting)
Six Up	Pick the winner in each of the six nominated races (Only available in the last 6 races of each race meeting)
All Up	Placing bets on more than 1 race. They can only be simple bets' combinations. It can be choosing n races with m combinations. The odds will be multiplied by each other.

1.3.2 Reinforcement Learning

Reinforcement learning is one kind of machine learning. It is about how the agent learns to take actions in the environment to get the maximum reward [9]. Reinforcement learning is totally different from the other two kinds of machine learning. Supervised learning trains the agent to learn from the data with labeled input and output. Unsupervised learning trains the agent to learn from the data with

only input and find the pattern. More details of reinforcement learning will be mentioned in the chapter of reinforcement learning.

1.4 Objective

In this project, our objective is to apply reinforcement learning into horse racing. We want to build a model that can place the bet on the winning horse by predicting it. Before the model can have positive profit, we would like the model to gamble like a human being. We will focus on the strategy that the model takes. For example, the model chooses to place a bet on which horses or chooses not to bet.

After training the model similar to human gambling, we would like to push the limit to go beyond human to gain profit in the long term or short term. We want the model to be overwhelming in horse racing. The result will be compared to the other method of machine learning in horse racing.

Chapter 2 Data Preparation

2.1 Data Collection

There are companies selling historical data from 2000 to 2020. But due to our limited budget, we collect our dataset from the Hong Kong Jockey Club official website [10] and Hong Kong Observatory official website [11].

2.1.1 Information of Horses

The information such as weight, sex, color, dim, etc. are collected. The dataset contains these features from 2771 horses.

2.1.2 Historical Horse Racing Records

All the horse racing records from 2010 to 2020 are collected. Each horse participates in horse racing for an average of 3 to 4 years. Therefore, the horse racing records between 2014 to 2019 will be mainly used in our model as we want to label the horse in our model. The dataset contains 37,755 records and 3,080 races from 2014 to 2019. And 2019 to 2020 will be our testing set with 810 races and 9859 records.

2.1.3 Historical Weather Data

The average degree, pressure and humidity from 2010 to 2020 are collected. The data between 2014 to 2019 will be mainly used for our xgboost regressor and deep neural network regressor, as we only use the horse racing record from 2014 to 2020 to train our model. The dataset contains 37,755 records for each race from 2014 to 2019. And 2019 to 2020 will be our testing set with 9859 records.

2.2 Data Description

2.2.1 Race data

The following tables show the feature values of racing records obtained from the HKJC official website. There are around 3890 race records from 2014 to 2020. (End of the 2019.)

Features	Description	Type	Values
race_date	The date of the race	Index	/
race_no	The numer of a race in a day	Index	/
race_index	Unique id of the race	Index	/
location	Location of the race	Categorical	HV, ST
class	Class of the horses	Categorical	Class 1 to 5, Group 1 to 3
race_length	Distance of the race	Categorical	1000, 1200, 1400, 1600, 1650, 1800, 2000, 2200, 2400
course	Track of the race	Categorical	A, A+3, B, B+2, C, C+3
draw	Draw of the horse in a race	Categorical	14 distinct values
going	Condition of the track	Categorical	FAST, SLOW, WET FAST, WET SLOW, FIRM, GOOD TO FIRM, GOOD, GOOD TO YIELDING, YIELDING,
horse_id	Unique id of the horse	Categorical	2744 distinct values
jockey_name	Unique id of jockey	Categorical	113 distinct values
trainer_name	Unique id of trainer	Categorical	112 distinct values
actual_weight	Weight added to the horse	Real value	/
declared_horse_weight	Weight of the horse	Real value	/
win_odds	The odds of betting the horse	Real value	/
place	The final place of the horse in a race	Categorical	14 distinct values
finish_time_sec	Finishing time of the horse in a race	Real value	(Seconds)

Figure 1. Race data

2.2.2 Horse data

The below tables recorded the data of the horse. This record includes all horse data in each race. This means the same horse has different data in different races because these data will change in each race. For example, after the horse may increase its rating, weight from the previous race.

Features	Description	Type	Values
last_actual_weight	The actual weight of last race	Real Value	/
last_declared_horse_weight	The last weight in last race	Real Value	/
diff_actual_weight	Difference actual weight between present race and last race	Real Value	/
diff_declared_horse_weight	Difference declared weight between present race and last race	Real Value	/
country	the country of the horse	Categorical	US,AUS,etc.
age	The age of the horse	Real Value	/
colour	The colour of the horse	Categorical	Bay, Chestnut, etc.
sex	The sex of the horse	Categorical	Gelding
import_type	The import type of the horse	Categorical	PP,PPG
sire_name	The name of the horse's sire	Categorical	Acclamation, Patagan,etc.
last_plc	The place in the last race	Index	/
last_rating	The rating in the last race	Index	/
rating	The rating now	Index	/

Figure 2. Horse data

2.2.3 Weather data

The below table is the weather data in each race. There is 3890 weather data which is every day of the race. Compared to the previous data (LYU1703), the feature values of the weather are significantly fewer. It is because we believe that some

weather data will not help but only make noise to the model. For example, the moon phase of the race day is supposed not to affect the result of the horse because the race court gets a lot of lighting.

Features	Description	Type	Values
mean_degree	The mean of the temperature of the race day	Real Value	/
mean_humidity	The mean of the humidity of the race day	Real Value	/
mean_pressure	The mean of the air pressure of the race dat	Real Value	/

Figure 3. Weather data

2.2.4 Additional data

Features	Description	Type	Values
total_first_count	The total count of first place	Real Value	/
total_second_count	The total count of second place	Real Value	/
total_third_count	The total count of the third place	Real Value	/
total_race_count	The total count of joined race	Real Value	/

Figure 4. Additional data

We believe that if a horse wins a lot, then it is likely to win the next race. Therefore, we add these features to increase the accuracy of our prediction. We have also analyzed the correlation between these features in the next sub-chapter.

2.3 Data Analysis

In this part, we are going to find the relationship between different features values based on the previous project analysing the data. We will classify the data into continuous data and categorical data [12].

2.3.1 Continuous data

The below table shows the correlation between two continuous values. Here, we will pick some significant values and analyze them.

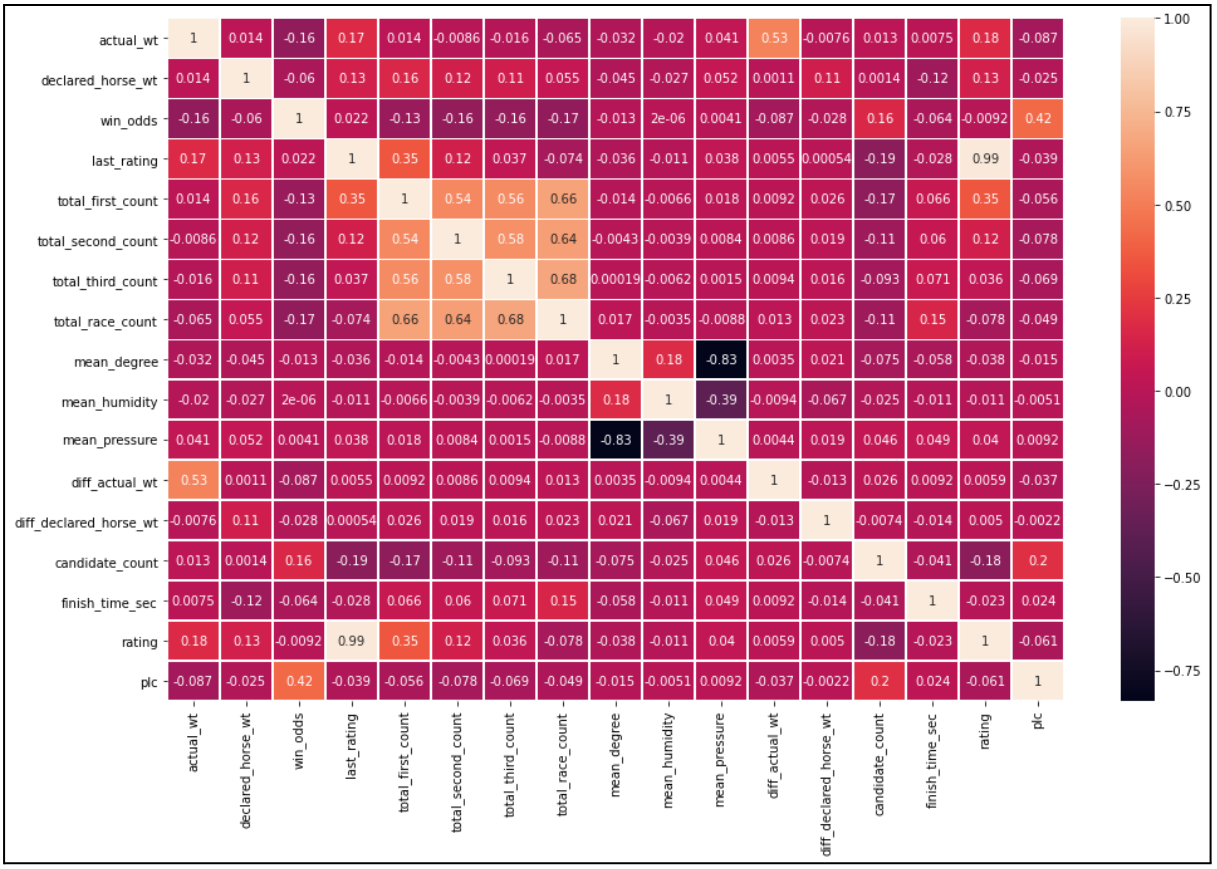

Figure 5. Corr. matrix of Continuous data

The correlation between the count of the first place and second place is 0.54, which is pretty high, and it implies that if a horse wins a lot before, then it is likely to win more. The correlation between the count of the first place and third place and the count of the second place and third place are similar.

The correlation between win odds and the place is 0.42. Supposing that the lower win odds get higher place and this correlation has proved that. The place is the index which is 1 is the highest and the correlation makes sense.

The rating is also related to the count of first place, A horse wins more, then the rating will increase. And there is an interesting correlation that the actual weight of a horse is positively correlated to the horse's rating with 0.18.

2.3.2 Categorical data

The below table shows the association between two categorical values. Here, we will pick some significant, meaningful values and analyze them. The association means the probability of getting the other data if we have data.(Forecast) This is called "Theil's U" or "Uncertainty coefficient". For example, if we have the horse_id, we can forecast the country, colour, sex, import type and sire name. This is certain because a horse can only have one country, one colour, one sex, one import type and one sire name. However, we can still find some interesting associations.

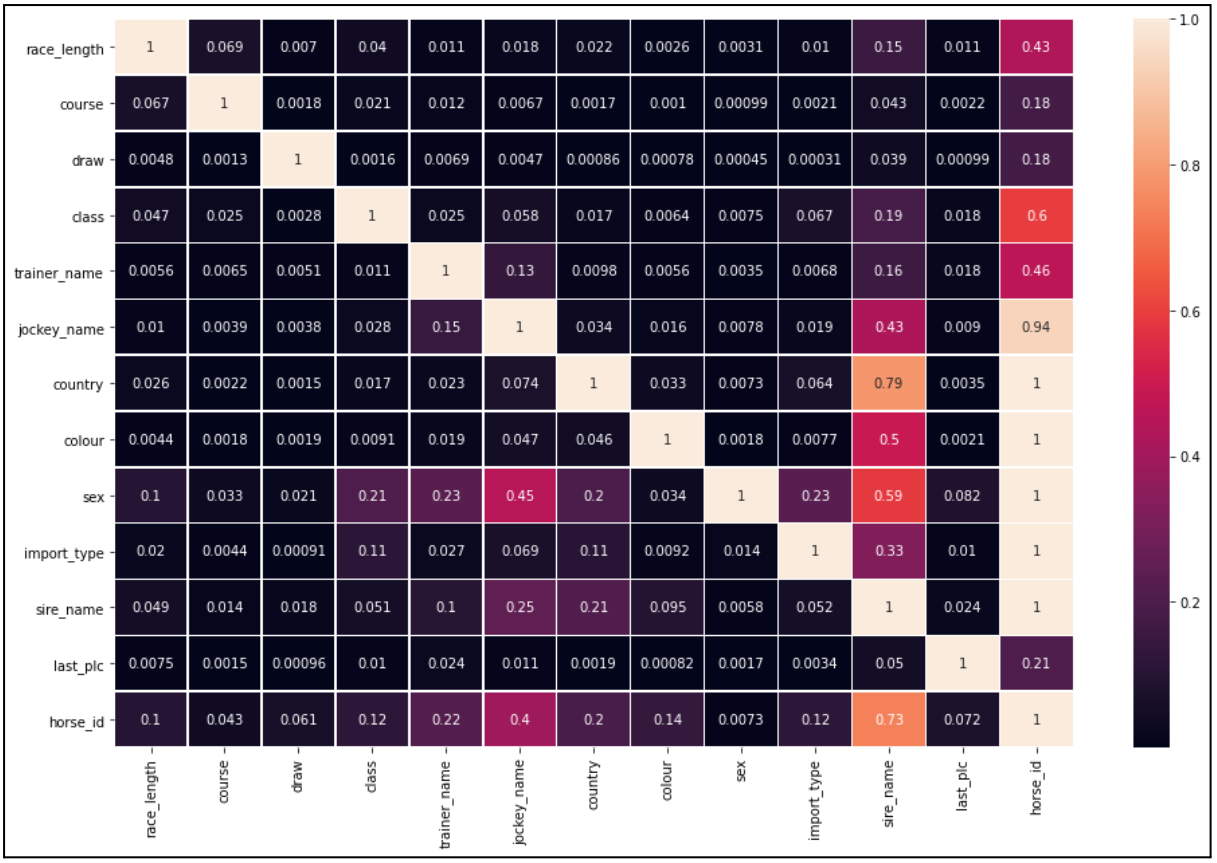

Figure 6. Theil's U of Continuous data

The association between the jockey name and horses' id is 0.94. This means most of the time the horse will only be rode by one jockey. The other association is the class and the race length. These mean the horse will usually train for a specific race length instead of changing it at a high frequency. The horse will stay in one class instead of changing in.

2.3.3 Continuous data vs categorical data

The below table shows the correlation ratio between numerical values and categorical values.The correlation ratio will tell the probability of knowing the category if we have a number. Here, we will pick some significant values and analyze them.

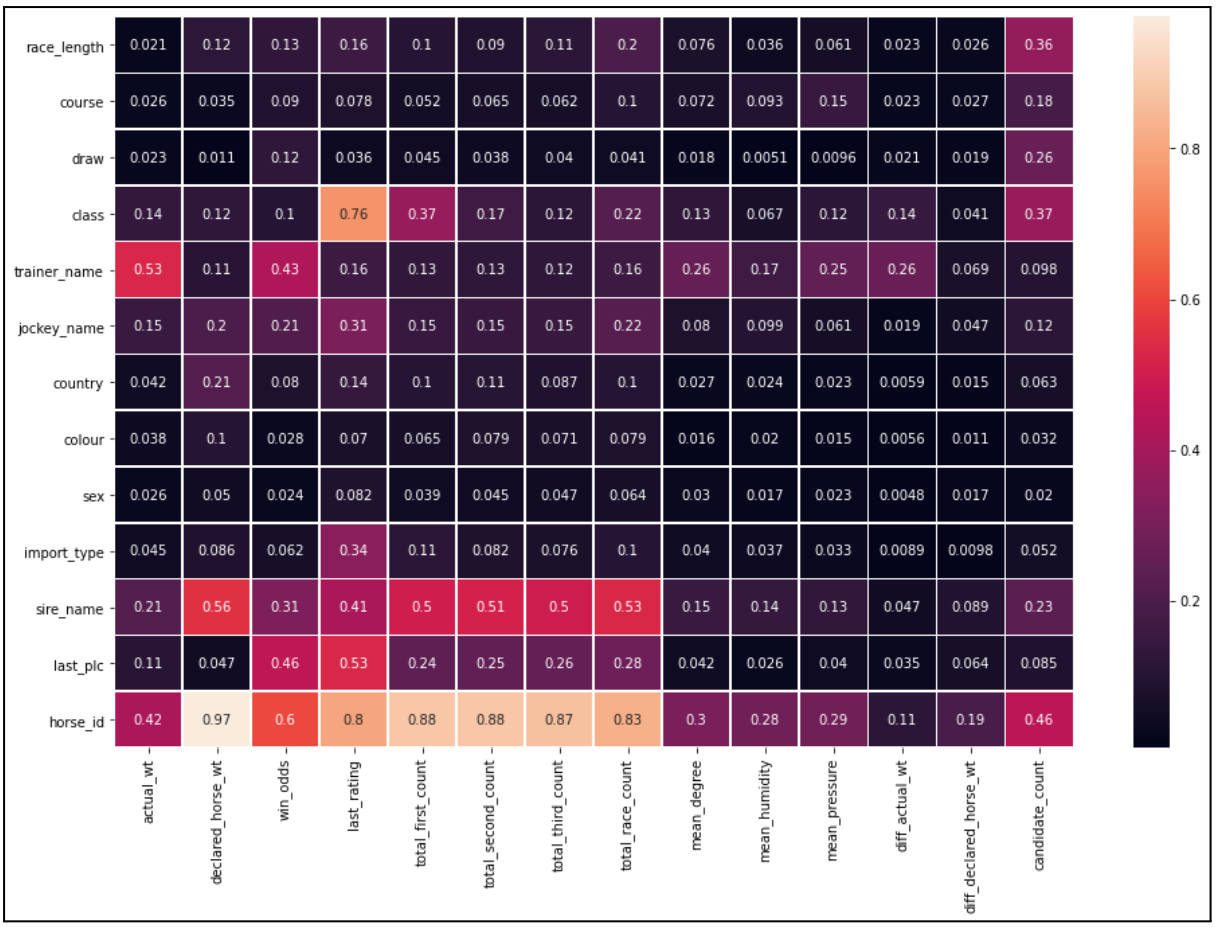

Figure 7. Correlation ratio between Continuous data and Categorical data

For example, if we have the last rating of the horse, in 76% we will know what class the horse belongs to. In this graph, we shall see if we have numerical data, we will easily know which horse it is compared to the other data. This proves that most of the horses have unique data instead of the same data but different horses.

2.3.4 Conclusion of analysis

In machine learning, if there are highly correlated features, It harms the performance of the model which is called multicollinearity. So we may need to exclude those features with high correlation. Luckily, there are no features with more than 0.8 correlation between them, So we will use all of these feature values.

2.4 Data preprocess

We normalize all of the continuous data using Z-score normalization other than max-min normalization as we think that there are outliers between the horses. We want a normalization to handle those outliers. We will use one hot encoding on the categorical data.

Chapter 3 XGBoost

3.1 Description

XGBoost was developed by Tianqi Chen in 2014 as a scalable end-to-end tree boosting system [13]. The reason for using XGBoost in this project is there are a lot of feature values like horses' data. Multivariable regression can be effectively done by XGBoost. In this chapter, the principle of XGBoost will be simply introduced and mainly introduced how to apply XGBoost into our model.

3.1.1 Decision Tree

Decision Tree is the basis of the tree boosting system. Here is a decision tree from the paper "XGBoost: A Scalable Tree Boosting System" written by Tianqi Chen. Decision tree classifies the data by asking the input some questions. The tree below is classifying the input by asking whether the input's age and gender. Every node (split) in the tree represents the features of input. The leaf of the tree is the category. There is a prediction score below the category. Decision trees can handle some missing feature in some input which benefits a lot in our case like setting missing in yes or no.

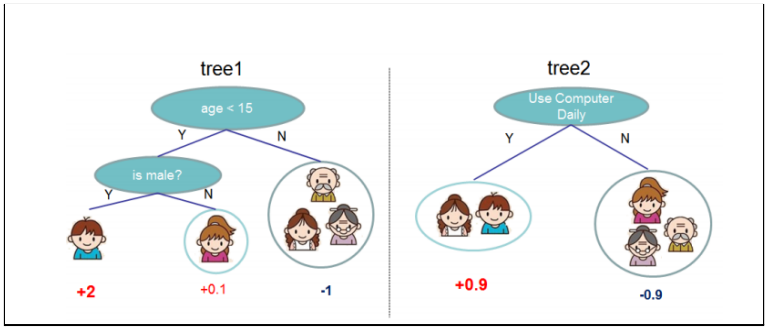

Figure 8. Example of decision tree[1]

[1]A figure in the paper
"XGBoost" URL:https://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf

In our case, the leaf is the predicted weight of every feature value and the predicted weight of each feature value will be used to calculate the predicted finish time. We will use the predicted finish time to predict the finished position of each horse.

However, decision trees usually cause overfitting problems which means training data's accuracy will be very high, while the testing data cannot fit the model, especially there are a lot of feature values causing a lot of noise.

3.1.2 Gradient Boosting Decision Tree

As mentioned in the last part, just the decision tree is difficult to provide a good model to fit the testing data. Boosting is combining weak learners to become a strong learner [14]. Weak learners mean which prediction is not accurate. Strong learners mean their prediction is accurate.

Tree boosting means there are a lot of decision trees with low accuracy, and then combined to a boosted tree with higher accuracy [16]. This is also called a tree ensemble. There is a boy in the figure above. If two trees combine together, the function of boy will become: f(boy) = (+2) + (+0.9) = 2.9

If there are many trees, gradient boosting will be used. The prediction of the i-th tree [15]:

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i)$$

Formula 1. The prediction of the i-th tree

Therefore, the formula of the t-th $\hat{y}_i^t = \hat{y}_i^{t-1} + f_t(x_i)$. $f_t(x_i)$ is the prediction where is the upcoming t-th tree and $\hat{y}_i^{t-1}$ is the prediction which is combined [15]. These two formulas will be used in XGBoost.

From the formula, the boosting will be done step by step. That is why it is called the gradient boosting decision tree.

3.1.3 XGBoost (Extreme Gradient Boosting)

XGBoost is based on gradient boosting decision trees. Here, the derivation of the learning objective is done by Corey Wade and Kevin Glynn from “Hands-On Gradient Boosting with XGBoost and scikit-learn” since this derivation is better for understanding.

The learning objective for t-th boosted tree:

$$obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_l^t\right) + \sum_{i=1}^{t} \Omega\left(f_t\right)$$

Formula 2. The learning objective for t-th boosted tree

The first part of the function is called the loss function which is the mean squared error for regression. is the target value in the i-th row and $\hat{y}_l$ is the prediction in the i-th row. The sum of the difference between all rows will be the error.
Here, the second part is called regularization term which smooths the final learnt weights to avoid overfitting by penalizing the complexity of the model [13].

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Formula 3. Regularization term

W gives the corresponding leaves and T is the number of trees. γ and λ are the penalty. If the penalty is zero, the learning objective will be the same as gradient boosting decision tree's objective.

Then, Chen used second-order approximation to optimize the learning objective which is from Taylor's formula. He calculated the first order and second order gradient by $g_i = \partial_{\hat{y}_i t-1} l\left(y_i, \hat{y}_i^{t-1}\right)$and $h_i = \partial^2_{\hat{y}_i} t_{-1} l\left(y_i, \hat{y}_i^{t-1}\right)$, we can get:

$$obj^{(t)} = \sum_{i=1}^{n} g_i w_{q(x_i)} + \frac{1}{2} h_i w^2_{q(x_i)} + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Formula 4. The learning objective for t-th boosted tree

Let $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$ and the optimal weight of a leaf j:

$$w_j = -\frac{G_j}{H_j + \lambda}$$

Formula 5. The optimal weight

The final objective function:

$$obj^{(t)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

Formula 6. Final objective function

This function can measure the quality of a tree by how well the model fits the data [13] The smaller score means the better structure.

In boosting, everytime the best score is calculated by best weight. However, it is impossible to find all possible tree structures. Therefore, exact greedy algorithm is used to find the best tree [13]:

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

Formula 7. Exact greedy algorithm

From depth = 0, starting from a single leaf and adding branches to the tree iteratively. [13] γ and λ are the penalty.
The first term in the Eqn is the score of the left child tree. The second term is the score of the right child tree. The third term is the score of not being a branch of the tree. Therefore, a loop is required to find out the best tree.

To conclude, XGBoost would find out the best G and H in the final objective function by ensembling trees. Then using an exact greedy algorithm to find the best branch adding to the tree. Here is the full algorithm [13] :

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Algo 1. Exact Greedy Algorithm

XGBoost uses shrinkage to avoid overfitting. Shrinkage adds weight after each step of tree boosting [13]. The concept of shrinkage is similar to learning rate.

3.2 Benefit

There are some benefits of choosing XGBoost. Block compression and sharding improves the speed of reading data from the disk. Cache-aware access provides faster access to cache for calculation.

The column block for parallel learning can reduce the time of sorting the data.

The data will be put into blocks and parallel computing can be used to reduce the time consumed. Multiple cores of the cpu can be used.Since this project's data is very large, handling the data and combining multiple trees at the same time can effectively reduce time. Therefore, using XGBoost can improve the performance of regression.

Here is the data provided by Chen in “XGBoost: A Scalable Tree Boosting System”:

Method	Time per Tree (sec)	Test AUC
XGBoost	0.6841	0.8304
XGBoost (colsample=0.5)	0.6401	0.8245
scikit-learn	28.51	0.8302
R.gbm	1.032	0.6224

XGBoost’s performance is much better than other exact greedy methods like scikit-learn and R.gbm.

3.3 XGBoost in Horse Racing

3.3.1 Progress

First, the target is to predict the horse’s finishing time and find out the ranking of the horses and gamble by the result. Therefore, we will use XGBoost to find out which feature value affects the finishing time the most. We assume that we do not know which feature value so we will put all of them into the algorithm.

3.3.1.1 Data

The training data will be the data from 2014 to 2018. The testing data will be the data from 2019.

The configuration will be the race date, season, race index, horse id, race number, class, win odds, total race counted. These are used to group up the data and most of them will not affect the finishing time.

The labeled input is the rating of the horse and the place of the horse in the race. These two data are labeled since we think they affect the finishing time the most. The labeled output will be the finishing time in seconds which is the prediction needed.

For the feature values, there are two kinds of feature values, continuous features value and categorical features. Continuous features mean the features can be represented in number. In our case, they include actual weight, declared horse weight, win odds, last rating, total count in first place, total count in second place, total count in third place, total count of race taken, mean degree of the day, mean humidity of the day, mean pressure of the day, different between actual weight and mean weight of the race, different between declared weight and mean weight of the race, and the number of candidates in the race.

Categorical features are transferred to true and false. For example, there are multiple race lengths like 1000M, 1200M, 1400M, 2000M, 2200M. They will be transferred to true and false by {0,1} where 0 represents false and 1 represents true. The node of the tree will be (Categorical features < 0.5) to split. The categorical features include the length of the race, the course, the draw, the class, the name of trainer, the name of the jockey, country of the horse, colour of the horse, sex of the horse, the import type, sire name, last place and horse id.

Here is part of the training data and part of the testing data:

Excluding the label, we have 37755 x 3963 training data.

	df_date	race_date	season	race_index	horse_id	race_no	class	win_odds_	total_race_count_	finish_time_sec	rating	plc	actual_wt	declared_horse_wt	win_odds	l
0	2014-01-01	2014/01/01	13/14	286	HK_2008_K364	1	Class 5	13.0	74	83.47	23.0	2	-1.432731	-0.457444	-0.416353	
1	2014-01-01	2014/01/01	13/14	286	HK_2012_P077	1	Class 5	2.9	9	83.89	40.0	3	1.593909	-0.760515	-0.603499	
2	2014-01-01	2014/01/01	13/14	286	HK_2009_L155	1	Class 5	16.0	39	83.97	33.0	4	0.160238	1.297181	-0.360765	
3	2014-01-01	2014/01/01	13/14	286	HK_2010_M188	1	Class 5	12.0	22	84.09	37.0	5	1.116019	0.802696	-0.434882	
4	2014-01-01	2014/01/01	13/14	286	HK_2011_N158	1	Class 5	23.0	10	84.22	33.0	6	0.160238	-0.266030	-0.231059	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
37750	2018-12-29	2018/12/29	18/19	300	HK_2016_A030	10	Class 2	27.0	22	81.93	85.0	4	-0.476950	0.100846	-0.156941	
37751	2018-12-29	2018/12/29	18/19	300	HK_2018_C143	10	Class 2	368.0	2	83.34	83.0	11	-0.476950	-1.127391	6.161578	
37752	2018-12-29	2018/12/29	18/19	300	HK_2018_C054	10	Class 2	24.0	1	81.75	81.0	3	-0.795543	-0.345786	-0.212529	
37753	2018-12-29	2018/12/29	18/19	300	HK_2017_B257	10	Class 2	1.6	7	81.62	89.0	2	0.478831	0.435820	-0.627588	
37754	2018-12-29	2018/12/29	18/19	300	HK_2017_B309	10	Class 2	33.0	1	83.51	82.0	14	-0.636246	1.759763	-0.045765	

37755 rows × 3975 columns

Figure 9. Part of the training data

Excluding the label, we have 9859 x 3963 data testing data.

	df_date	race_date	season	race_index	horse_id	race_no	class	win_odds_	total_race_count_	finish_time_sec	rating	plc	actual_wt	declared_horse_wt	win_odds
0	2019-01-01	2019/01/01	18/19	301	HK_2016_A344	1	Class 4	2.8	10	94.61	54.0	1	1.275316	1.600252	-0.605352
1	2019-01-01	2019/01/01	18/19	301	HK_2015_V400	1	Class 4	7.0	18	95.10	44.0	2	-0.317653	0.627233	-0.527529
2	2019-01-01	2019/01/01	18/19	301	HK_2017_B006	1	Class 4	11.0	14	95.22	41.0	3	-1.114137	-0.250079	-0.453411
3	2019-01-01	2019/01/01	18/19	301	HK_2016_A120	1	Class 4	17.0	20	95.25	53.0	4	1.116019	-1.159293	-0.342235
4	2019-01-01	2019/01/01	18/19	301	HK_2016_A087	1	Class 4	3.5	13	95.44	41.0	5	-0.795543	0.005139	-0.592382
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9854	2019-12-29	2019/12/29	19/20	293	HK_2017_B161	10	Class 3	10.0	17	100.30	72.0	4	0.638128	0.308211	-0.471941
9855	2019-12-29	2019/12/29	19/20	293	HK_2016_A193	10	Class 3	13.0	45	100.82	63.0	10	-0.795543	-0.824320	-0.416353
9856	2019-12-29	2019/12/29	19/20	293	HK_2018_C197	10	Class 3	3.9	12	100.15	64.0	3	-0.636246	0.547478	-0.584970
9857	2019-12-29	2019/12/29	19/20	293	HK_2017_B203	10	Class 3	25.0	25	100.09	70.0	1	0.319535	-1.015733	-0.194000
9858	2019-12-29	2019/12/29	19/20	293	HK_2014_T098	10	Class 3	12.0	69	100.13	61.0	2	-0.795543	1.504545	-0.434882

9859 rows × 3975 columns

Figure 10. Part of the testing data

3.3.1.2 Training

The best component of hyperparameters is found out by function grid_search:

- The objective is ‘reg:squarederror’ which is the loss function of the learning objective.

- The learning_rate which is the shrinkage is set to 0.05.
- The max_depth which the maximum depth of the tree is set to 5. This value can prevent overfitting since higher depth may cause the model to learn relations specific to the training data.
- The n_estimators is set to 230. This is the number of boosted trees trained.
- The random_state is set to 42. This can prevent too many combinations since there are 230 estimators. This will avoid taking too long of searching for the best tree.

For the other hyperparameters, they are set to the default value provided by XGBoost. Final setting:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.05, max_delta_step=0, max_depth=5,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=230, n_jobs=0, num_parallel_tree=1, random_state=42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

Figure 11. Setting of XGBoost

Figures below are the first tree and last tree generated.

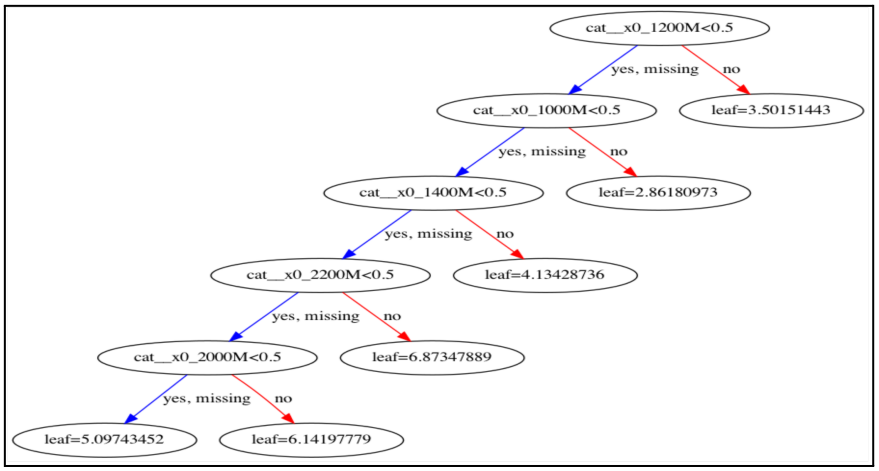

Figure 12. The first tree generated

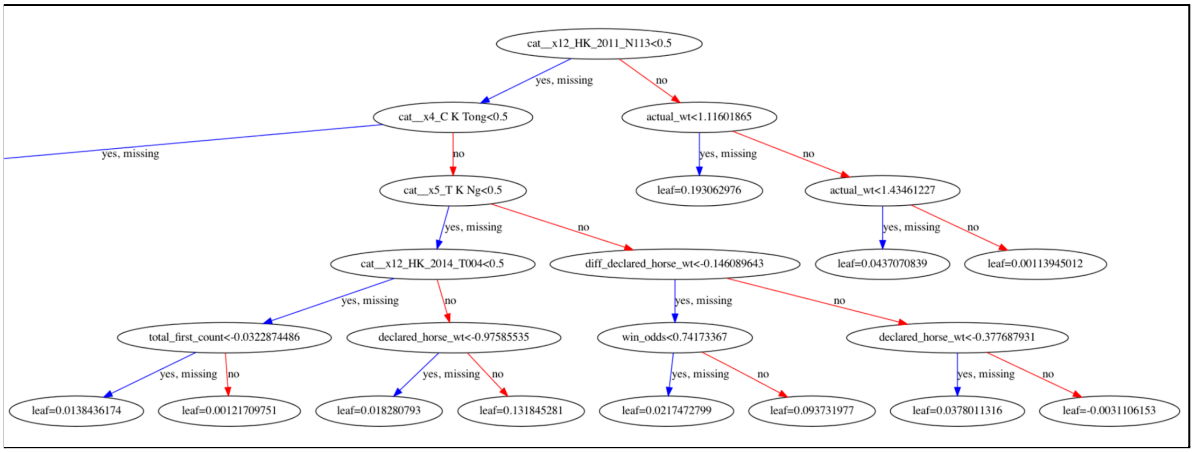

Figure 13. Part of the last tree generated

Then, using the trained model to predict the finishing time of the horses in 2019. Sort the finishing position in ascending order by the predicted finishing time and grouped by each race. Here is one of the predicted race:

	0	df_date	race_date	season	race_index	horse_id	race_no	class	win_odds_	total_race_count_	finish_time_sec	rating	plc
9856	99.330414	2019-12-29	2019/12/29	19/20	293	HK_2018_C197	10	Class 3	3.9	12	100.15	64.0	3
9849	99.410896	2019-12-29	2019/12/29	19/20	293	HK_2018_C443	10	Class 3	5.1	4	100.69	72.0	8
9851	99.426056	2019-12-29	2019/12/29	19/20	293	HK_2017_B023	10	Class 3	5.8	25	100.83	63.0	11
9845	99.607193	2019-12-29	2019/12/29	19/20	293	HK_2017_B189	10	Class 3	8.9	35	100.49	77.0	5
9858	99.632103	2019-12-29	2019/12/29	19/20	293	HK_2014_T098	10	Class 3	12.0	69	100.13	61.0	2
9854	99.680313	2019-12-29	2019/12/29	19/20	293	HK_2017_B161	10	Class 3	10.0	17	100.30	72.0	4
9855	99.829094	2019-12-29	2019/12/29	19/20	293	HK_2016_A193	10	Class 3	13.0	45	100.82	63.0	10
9852	99.874359	2019-12-29	2019/12/29	19/20	293	HK_2017_B317	10	Class 3	16.0	21	101.25	70.0	12
9847	99.891357	2019-12-29	2019/12/29	19/20	293	HK_2017_B353	10	Class 3	15.0	18	100.51	67.0	6
9857	100.045280	2019-12-29	2019/12/29	19/20	293	HK_2017_B203	10	Class 3	25.0	25	100.09	70.0	1
9848	100.065971	2019-12-29	2019/12/29	19/20	293	HK_2015_V338	10	Class 3	33.0	41	100.51	77.0	7
9850	100.530182	2019-12-29	2019/12/29	19/20	293	HK_2016_A127	10	Class 3	69.0	19	102.16	76.0	14
9853	100.536362	2019-12-29	2019/12/29	19/20	293	HK_2017_B330	10	Class 3	74.0	14	101.71	78.0	13
9846	100.768089	2019-12-29	2019/12/29	19/20	293	HK_2018_C489	10	Class 3	102.0	3	100.69	69.0	9

Figure 14. One of the predicted race[2]

3.3.2 Result

The result of XGBoost is satisfying. First, the first tree is meaningful:

[2] The column “0” is the predicted time.

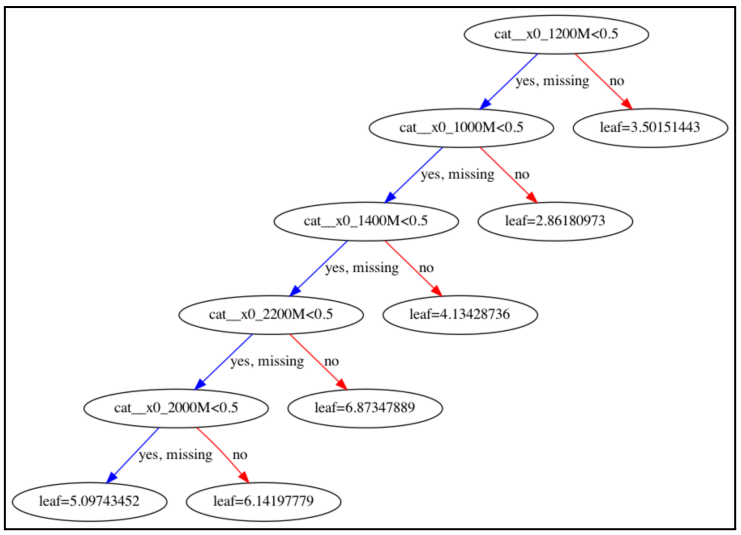

Figure 15. One of the trees generated

The model notices that the length of race determines the finishing time since 1000M's finishing time of a horse is obviously smaller than 1400M. The last tree is more complex than the first tree. That means the gradient boosting is working.

R2 score is 0.9974 This score means how the prediction data fits the actual data. 1 means the same as the actual data. 0 is completely different from the actual data. The R2 score is near to 1 so the prediction is working well.

The accuracy of predicting the first place is 30.3704% and the accuracy of predicting the first, second and third place is 7.1605%. The accuracy compared to the previous project is acceptable.

3.3.3 Simulation

For simplification, we choose only 1 betting type 'win', each time we bet 100 dollars to the first place (Win) according to the predicted data. Then find the result from the actual data. If we win, the cash balance will be added by 10* win_odds - 10. If we lose, the cash balance will be minus by 10.

All the graphs below are about the cash balance. Y-axis represents the cash balance. X-axis represents the race order by date.

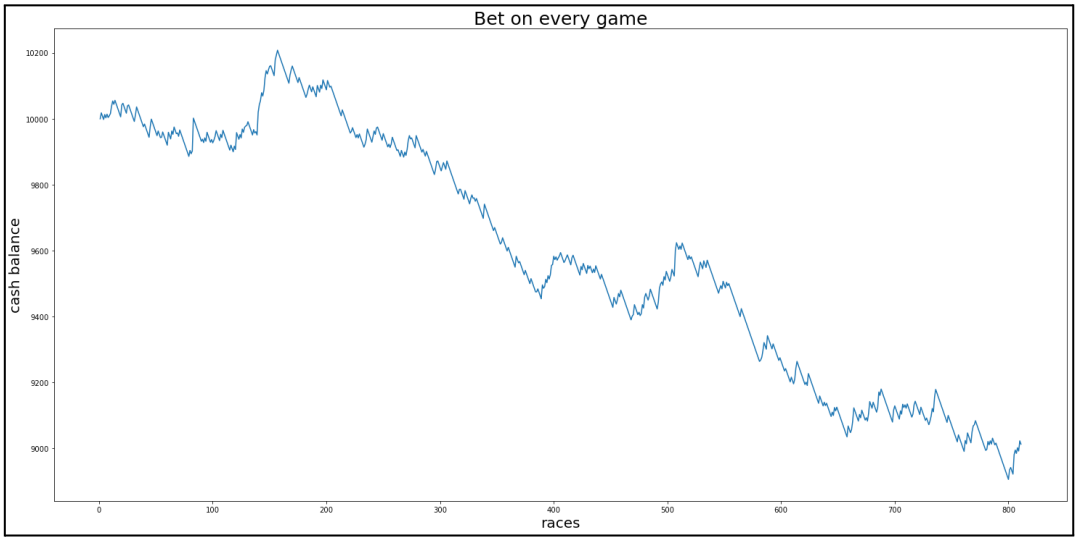

Figure 16. Cash balance when betting on every game

It is intuitive that if we just simply bet on every game, we will definitely have a negative gain. Where the last cash balance is 9013, we lose 1000 eventually.

So we try to find some criteria that we can obtain a positive gain which is ‘how many times the horse participated in horse racing before’

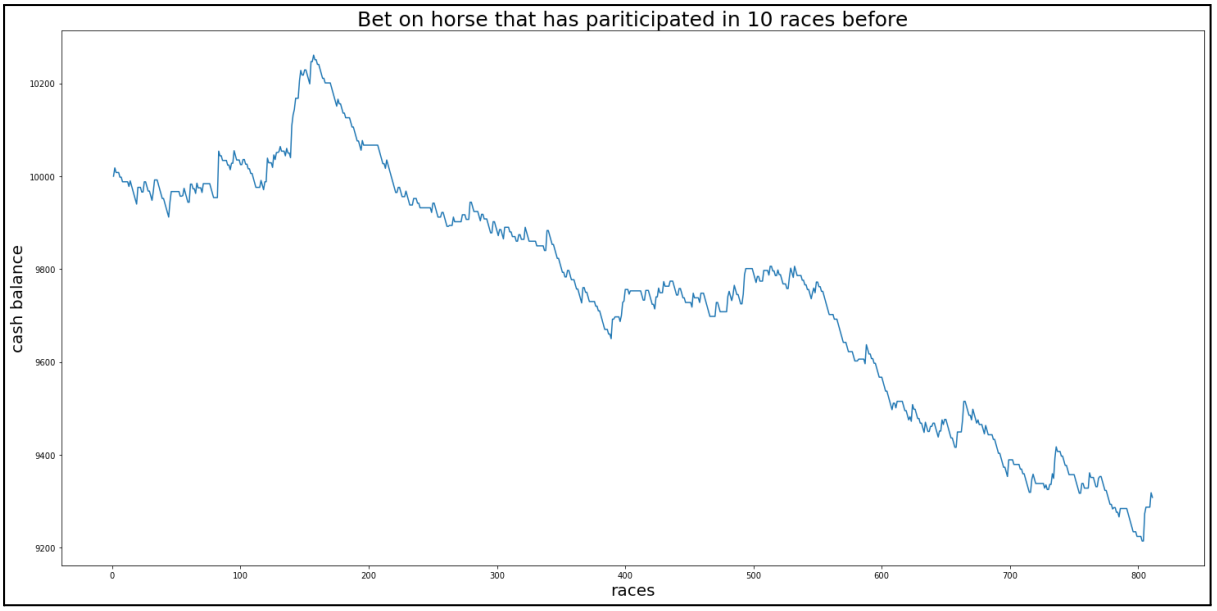

Figure 17. Cash balance when betting based on criteria: 10 races

If we increase the limitation, the cash balance is higher than before. The last cash balance is 9308.

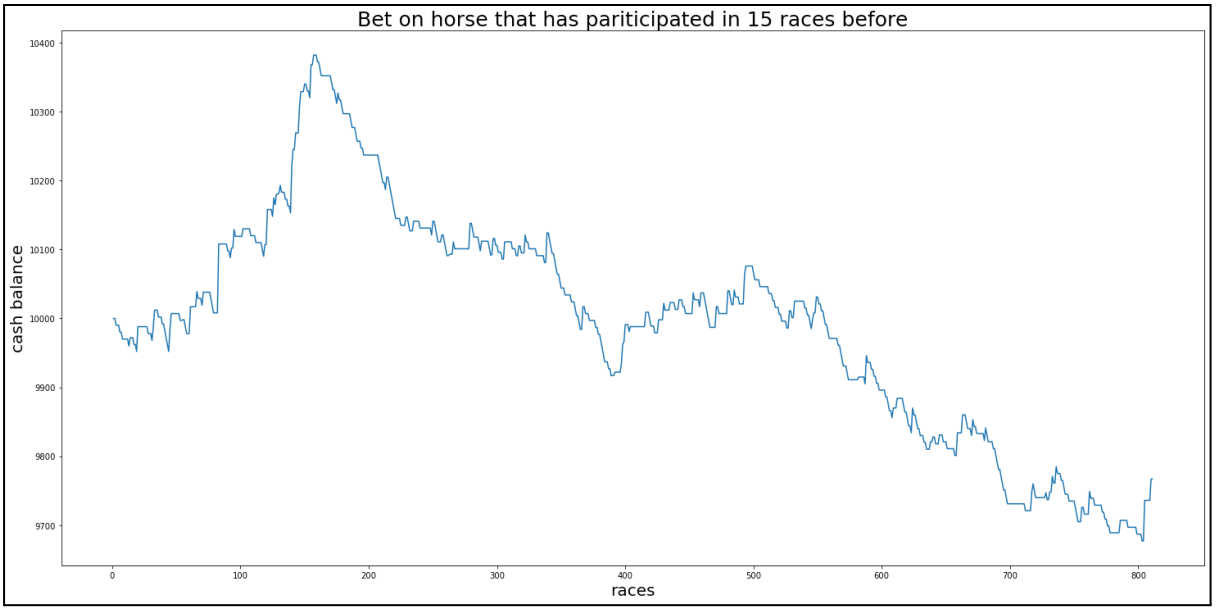

Figure 18. Cash balance when betting based on criteria: 15 races

Keep adding the limitation by 5. The last cash balance is now 9767.

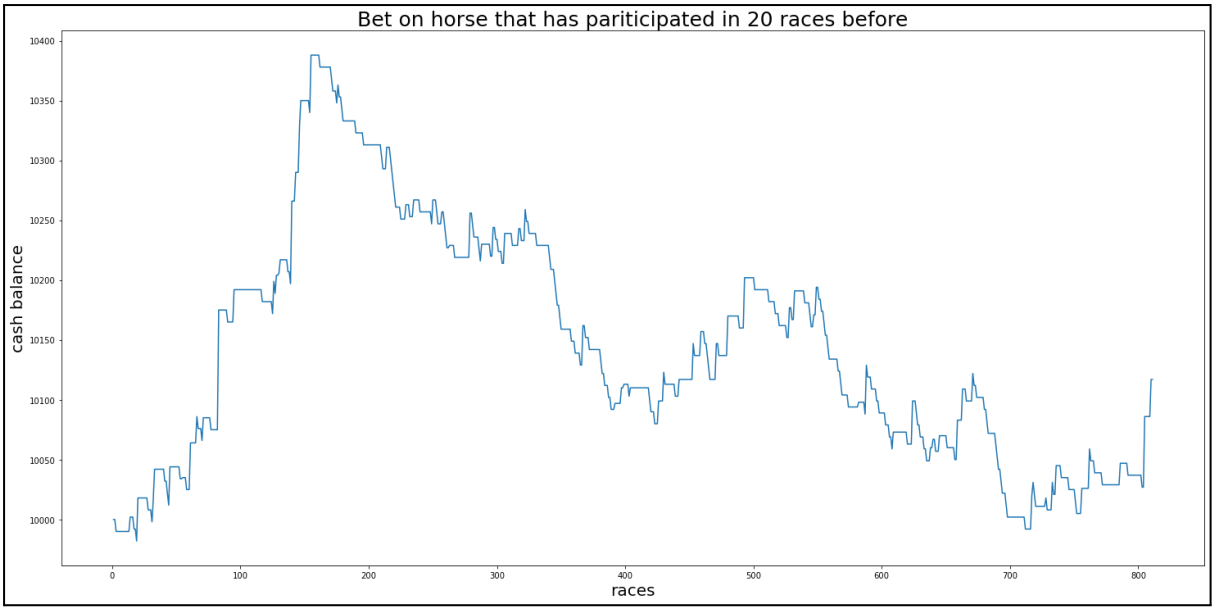

Figure 19. Cash balance when betting based on criteria: 20 races

Now the limitation is set to 20. The last cash balance is positive at 10117 and most of the time the cash balance is above 10000.

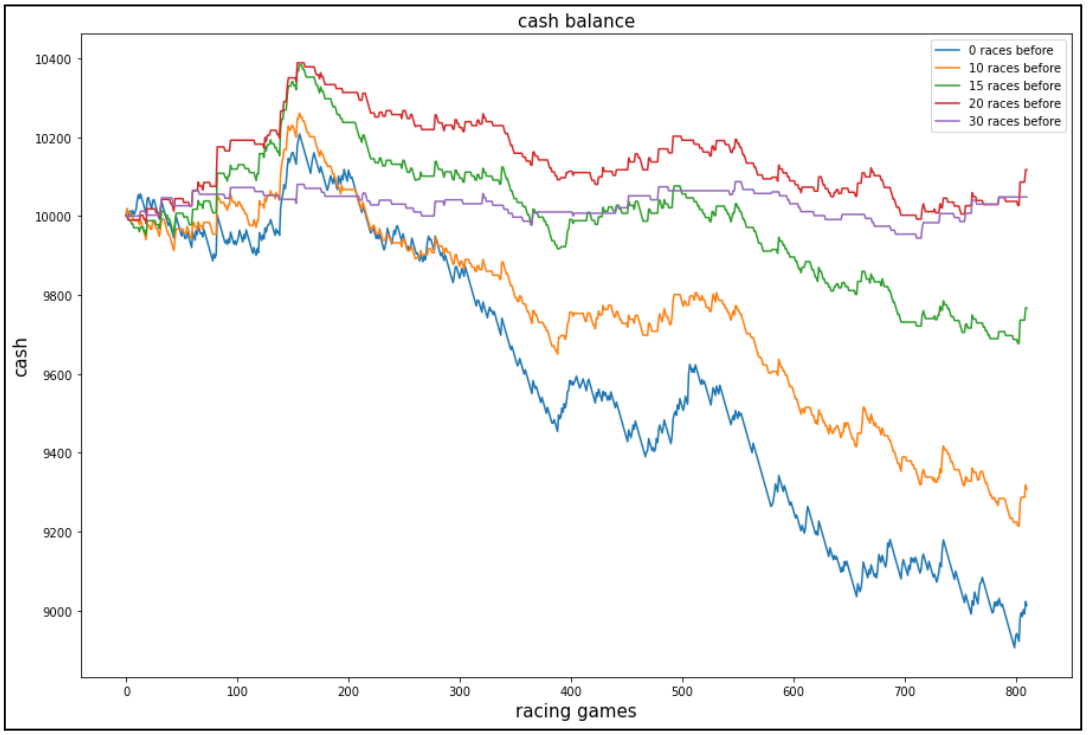

Figure 20. Cash balance when betting based on different criteria

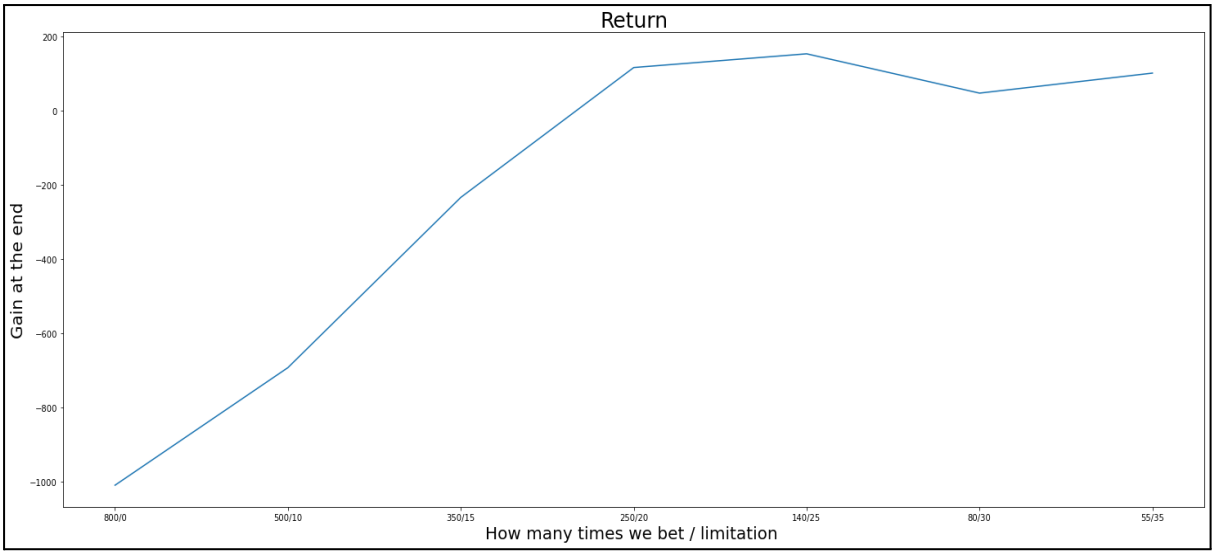

Figure 21. Return of betting based on different criteria

As shown above, the criteria of how many races the horse has participated is effective to decide whether to bet or not.

3.3.4 Conclusion and Analysis

3.3.4.1 Analysis of the results

- There are positive correlation between the ‘participation experience of horses’ to the ‘win rate’
    - The performance of the horses are more stable
    - These experienced horses are old and most of them belong to class 3,4,5.
        - The competition in these classes are not that keen

- Although we can use a criteria to decide whether to bet or not, but the return may not be stable and maximized, and the return is not high, which may not be suitable to use in real life, it can also be treated as a classification problem whether to bet this not or.

- If we just focus on the first place, it is nearly impossible for us to have a positive return, although we have a criteria to decide whether to bet or not.

- This prediction with 30% accuracy may help us in Reinforcement learning by ordering the input horses.
    - The place we predicted, will be used as the order of the horse we input to the reinforcement learning model.

- We learn how complicated horse racing betting is through Building this model.
    - Highly accurate prediction is not enough, we need a strategy to bet.

3.3.4.2 Conclusion

To conclude, The accuracy of the XGBoost is pretty good. However, it is not enough for us to just have a nice accuracy, it is more important that if we can find some betting strategies that help us to obtain positive gain. We will use the XGboost prediction to help construct the reinforcement learning. And find a better strategy to bet.

Chapter 4 Reinforcement Learning

4.1 Introduction

Reinforcement learning is a kind of machine learning that learns by interacting with the environment. The learner is not taught what actions to be taken but learns from the actions [9]. In our case, we do not want to teach the agent how to bet on horse racing. Instead, we want the agent to learn by himself and find the best way to bet.

4.1.1 Algorithm

The basic algorithm of reinforcement learning is shown in the figure below:

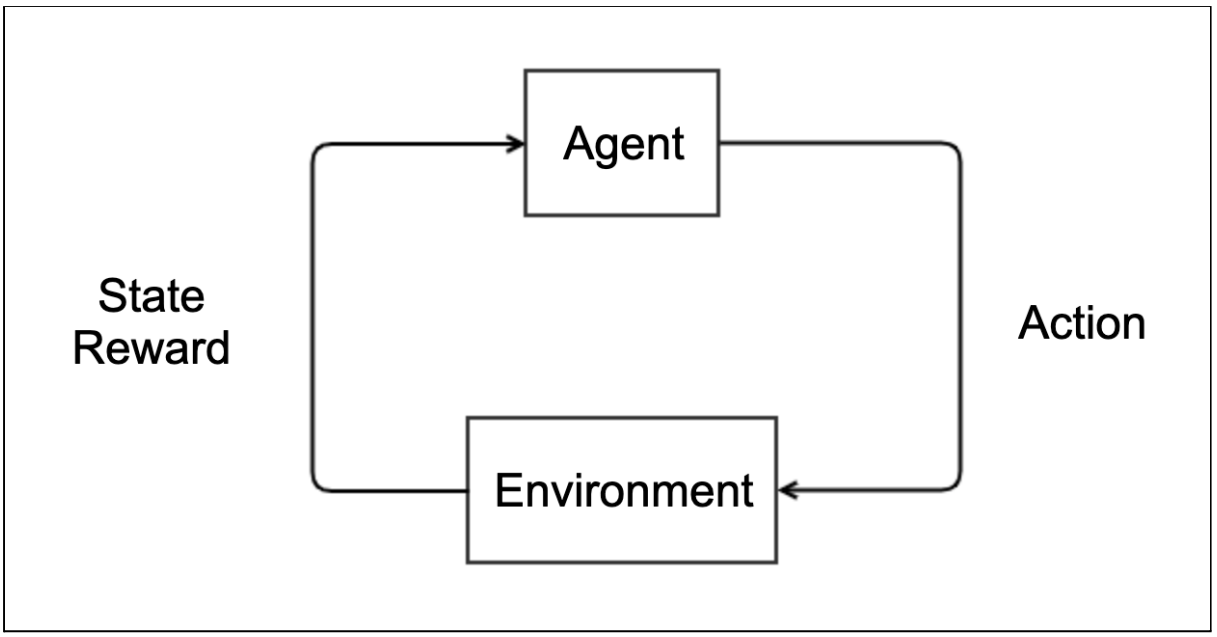

Figure 22. algorithm of RL

The agent which is trained interacts with the environment by action. The action will be based on different policies due to different RL algorithms. After the action, the environment will return a new state and reward to the agent. The agent will know its performance by the reward. Then the agent can update the policy of action by the reward.

4.1.1.1 Markov Decision Process

Formally, reinforcement learning problems can be modeled as Markov Decision Process:

- M = {S,A,O,T,Ɛ,r}
- S : State space containing a set of states (s) that the agent can be.
- A : Action space containing a set of actions (a) that the agent can make.
- O : Observation space containing observations (o) that the environment gives based on the agent's state
- T : Transition operator which is the probability of choosing the future state based on the current state ($p(S_{t+1}|S_t)$)
- Ɛ : Emission probability which is the probability of getting the observation based the state ($p(O_t|s_t)$)
- r : Reward function which is the reward from the action on the state. This can tell us which actions on which states are better ($r(s_t,a_t)$)

4.1.1.2 Goal

The goal of reinforcement learning can be written as:

$$\theta^{\star} = \arg\max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[\sum_{t} r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Formula 8. Reinforcement learning

In words, the goal of reinforcement learning is to find the best policy $\theta^{\star}$ which can bring the best expected total reward:

$$E_{\tau \sim p_{\theta}(\tau)} \left[\sum_{t} r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Formula 9. Best expected total reward

4.1.2 Environment

There are many kinds of environments based on the requirement. There are discrete environments and continuous environments which means finite states and infinite states. More kinds of environments are episodic or non-episodic, single agent or multi agents.

In this project, there will be only one agent, finite states since we only train the agent to learn how to bet. For the environment, it will be an episodic environment which means the current action will not affect the next action. It is partially observable since the agent will not know which horse wins until it finishes its action.

4.1.3 Different Reinforcement Learning Algorithm

Due to different situations in reinforcement learning, Based on the basic algorithm, there are other types of reinforcement algorithms including policy gradients, value-based, actor-critic, model based.

- Policy gradient is a policy-based algorithm. The agent will take an action based on the policy. The policy is different actions with their probability.(Ex. Left 50%, Right 50%) The final reward of each episode will improve the policy.

- Value based means the agent will take an action based on the reward after an action. Most likely, the reward will be given immediately after the action is taken.

- Actor-critic combines policy gradient and value-based. Critic is a value based network while actor is the policy gradient.

- Model based requires a base policy (model). The agent will act based on a built model and improve the model based on the reward.

4.1.4 Objective

In this project, we are using a value-based algorithm called deep Q-learning. The reason for us to choose value based is value based algorithm is more similar to the situation of gambling. We will place a bet on a horse because we believe that that horse will give us reward (money). Same reason, the agent will place a bet on a horse because the horse will give the agent reward. We hope to find out the best policy to get the best profit from horse racing.

4.2 Q-learning

Q learning is a value based method which learns from the value. Although we are not using Q learning in horse racing, it is the basis of deep Q-learning. The reason for not using Q learning will be mentioned at the end of this part.

4.2.1 Q function in a table

Definition of Q function: Q(s,a) For s = state, a = action, The value of Q(s,a) will be updated after each step of the episode.

If we group all the Q functions together, it can be represented as a Q table:

State \ Action	a1	a2
s1	value1	value2
s2	value3	value4

4.2.2 Algorithm

The algorithm of Q learning is to update the Q table and based on the Q table to choose the best action [17].

1. Initialize Q function Q(s,a) to some random values

2. Take an action from a state using epsilon-greedy policy from Q function

3. Observe the reward and the new state

4. Update the Q table by :

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma maxQ(s'a') - Q(s,a))$$

Formula 10. Q-learning

5. Repeat step 2 to step 4 until terminal state

Epsilon greedy policy means that the action will be picked up by the best value. There is a parameter called Epsilon which decides the probability of choosing. Using the table above as the example, let epsilon be 0.9 and value1 > value2. If the agent is in the state s1, there will be 90% of choosing action a1 and 10% of choosing a random action.

For the equation:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma maxQ(s'a') - Q(s,a))$$

Formula 11. Q-learning

- α is the learning rate to decide how many differences need to be learned. This is useful for convergence.

- r is the reward obtained from the action.

- γ is the discount of the future reward. The discount is to decrease the effect of the future reward to the current reward so the agent can learn to take the other action not just because of the future reward. We want a balance between the future reward and the current reward.

MaxQ(s',a') means the next state's best value of the Q function. This is the future reward.

The whole equation is for updating the Q table by calculating the difference between the actual value and the estimated value. The difference will be minimized to build the best Q table to find the best result for best efficiency.

4.3 Deep Q-Learning with MLP policy

In this section, we apply reinforcement learning in horse racing. In order to update the$Q(s, a)$function, we need to iterate through all the state. However, in real life, the state space is too complex for the agent to go through all of the states. So, we need an approximate q-value for those similar states. In deep Q-Learning, we use a neural network to approximate the$Q(s, a)$function, by combining the Q-learning and deep neural network.

4.3.1 Deep Q-Network

Deep Q-network Architecture:

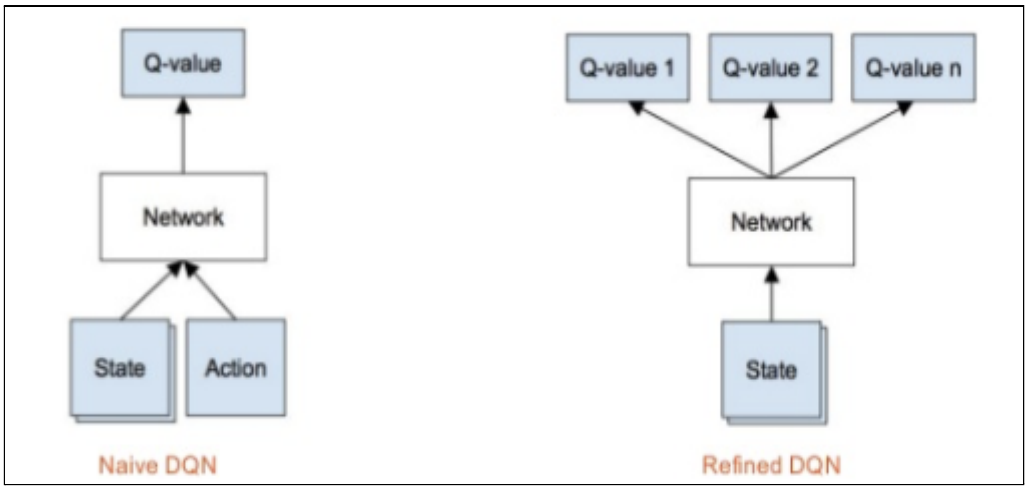

Figure 25. Deep Q-network Architecture

In this project we will choose the Refined DQN.

4.3.1.1 DQN Loss function

$$E_{s,a,r,s'}[((r+\gamma \max_{a'} Q(s',a';\theta')) - Q(s,a;\theta))^2]$$

Formula 12. DQN Loss function

The above loss function is used to find the approximate q-value in the network[19], If the θ'is a constant, we can use gradient descent on the loss function respect to θ.

4.3.1.2 Moving target problem

$(r+\gamma \max_{a'} Q(s',a';\theta'))$ is the target network, and $Q(s,a;\theta)$ is the prediction. However, these two terms are correlated, so if we change θfrequently, the θ' will change frequently too, our network is training for a moving target which will lead to diverge.

So one of the solution from DeepMind[2] states that θ' is updated with a long enough interval so that we can avoid it from diverge.

4.3.1.3 Experience Replay

Q-learning always uses the reward of the last state to train the value network, but the optimization techniques required that the data is independent and identically distributed sampling, so we store an amount of the latest step into the replay buffer, it performs the gradient descent on a sample minibatch. So the correlation between the data can be canceled out [2].

Chapter 5 Applying DQN in horse racing

In this semester, we try a simple approach, directly giving all the feature values of all the candidates in a single game, to see if the agent can bet on a ‘winning’ horse. Several betting types of horse racing are mentioned in the first chapter, However, due to simplicity, we concern only one betting type ‘A bet to win’ (if the horse wins the first place, you win) in this semester.

5.1 Construction

We use the Stable Baselines3 and OpenAI Gym[9] to construct our DQN reinforcement learning.

5.1.1 Environment

We have 10000 cash balance, and if the agent wins, we give him 10 * win odds cash. If he loses, we take 10 cash from him. That means the agent can only bet 10 dollars.

5.1.2 Observation Space

As mentioned before, the feature values of all the horses and the cash balance are the observation space to the agent. Also the win odds of the horses are given. Since there are horse racing games which less than 14 horses participate in, we would set all the feature values of those invalid horses to be -99.

5.1.3 Action Space

We simply want the agent to choose whether to bet and which to bet. So there will be a total 15 actions {‘1’, ‘2’, … ‘14’, ‘15’} that an agent can choose, where action

{‘1’, ‘2’, … ‘14’} represent to bet on the number 1, 2, … ,14 horse respectively. And action {‘15’} represents not to bet.

5.1.4 Step

After the agent goes through one games, the agent will receive first two major informations

1. The new game state : feature values of all 14 horses of the next game and the cash balance after the last game.
2. The reward obtained based on the last action.
3. Game over or not.
    a. Once the agent loses more than 1000 dollars where the cash balance < 9000, then it is game over.
    b. Once it goes through all the racing games, it is game over too.

5.1.5 Reset

In order to train the agent to learn which horse is nice to bet, other than recognizing the winning horses in each state, the state jumps to random racing games everytime we reset, and the cumulative reward will be reset.

5.1.6 Termination state

In horse racing games, we said it is game over when the agent loses more than 1000 dollars or when it goes through all the games.

5.2 Reward Function and discount factor

Reward for the action guides the agent to learn. It is the only feedback that the agent will receive, It is extremely important for the reinforcement learning algorithm to

perform well. A badly constructed reward function will lead to an unwanted behaviour of the agent. Also a good reward function can let the agent learn faster.

In a horse racing game, the basic idea for the reward function is simple, when it wins, it receives a large reward. otherwise, it receives a penalty. Intuitively, the agent will not bet on any racing game if we only give reward or penalty on these two situations, as it will not receive any penalty. So we will give a penalty which is less than the ‘bet but lose’ situation.

5.2.1 Idea of Reward Function

Just simply give penalty or reward based on the change of cash balance, here is one of the example to illustrate:

- $R(bet\ and\ win) = C_1 * \Delta Cash\ Balance\ ,\ where\ C_1 > 0$
- $R(bet\ but\ lose) = C_2 * \Delta Cash\ Balance\ ,\ where\ C_2 < 0$
- $R(\ do\ not\ bet\ ) = C_3 * win\ odds\ of\ the\ true\ first\ place,\ where\ C_2 < C_3 < 0$

Recall that there are invalid horses in the input, so we have to deal with the case when the agent chooses to bet on a invalid horse. There are few approaches [20]:

1. Ignore it
    a. $R(bet\ on\ invalid\ horse) = 0$
2. Same penalty as ‘do no bet’
    a. $R(bet\ on\ invalid\ horse) = R(do\ not\ bet)$
3. Large penalty
    a. $R(bet\ on\ invalid\ horse) = C_4 < C_3$

A paper [20] claims that giving a penalty to the agent is one of the common ways, and masking is one of the methods to do so. However, We will not mask ‘invalid horse’ to ‘valid horse’ so we directly give it a reward which same as ‘do not bet’

We believe that if we choose the first option, it is not reasonable as it becomes ‘do not bet’ without any penalty. Therefore, the agent may always choose this action as there is no penalty.

For the third option, the penalty of ‘bet on a invalid horse’ is large, the agent will tend to bet on those small number horses to avoid getting a large penalty, it probably will just choose the first five actions: ‘bet on horse 1’, ‘bet on horse 2’ … ‘bet on horse 5’ . Since there are at least 5 horses in every race, these actions will never be invalid action.

5.2.2 Discount factor

The formula below shows the total future reward G on the current step t is based on all the rewards multiply γ in all the future steps:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Formula 13. Total future reward with discount factor

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

Formula 14. Total future reward without discount factor

Where the γ is the discount factor, the agent will ‘treat’ the future reward more important if γ is closer to 1 [24].

Learning how to bet on horse racing games one by one is an episodic environment, each separated horse racing game will end in the same way. Rewards are given according to the result, which are ‘bet and win’, ‘bet but lose’ and ‘dont bet’.

In a racing game, the agent is supposed to learn how to win all the bets, the future bet is as important as the current bet. Also, T is a finite number as in our environment, so $\gamma = 1$ which is the second way, the agent should treat every game at the same level.

5.3 Cash balance using DQN

The results below are the cash balance of playing horse racing games including 5 - 14 candidates.

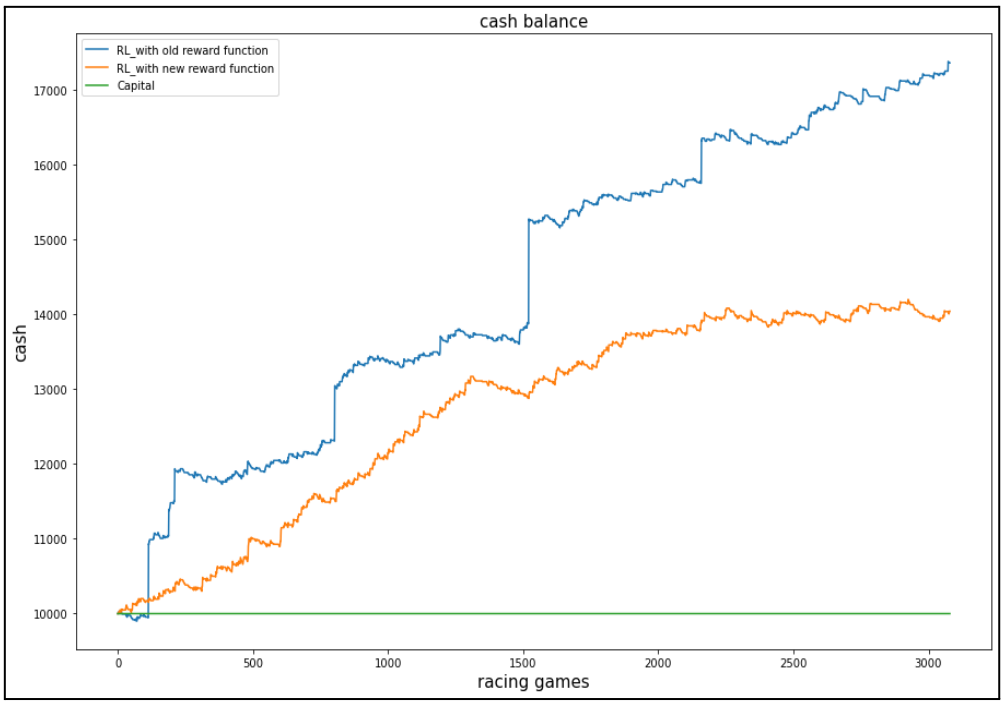

Figure 26. Cash balance between the agent with new and old reward function in training set

Figure 27. Cash balance between the agent with new and old reward function in testing set

We got a positive gain from last semester, however, the result is not good enough and the behaviour of the agent is not good.

5.3.1 Problem we encounter

Since we include all the racing games, there is invalid movement, which is the agent betting on the invalid horses. The invalid horses may hinder the training of the model, so we decide to train the model based on a specific type of horse racing game. And we also would like to see how the number of candidates in a racing game affect the performance of the model.

6 Twelve candidates horse racing game

6.1 Data preparation

We decided to train the model by using the games from 2014 to 2020 because the careers of the horses are usually 4~6 years. However, we have to get enough data to train a new model just to ‘play’ the horse racing game with a specific number of candidates.

6.2 Expand dataset

Based on the data we used last semester, containing only 1332 twelve horses racing games for training, and 343 racing for testing. Which is obviously not enough for training. Therefore, we enlarge our dataset from 2014-2020 to 2011-2020, we obtain 1922 twelve horses racing games for training, and 915 for testing.

Since the career of the horses are usually 4~6 years, we have to sacrifice the advantage of labelling the horse. Then the shape of the input data is (12 * 1160) which is 12 horses and 1160 features per horse.

6.3 New Environment

The shape of input is changed, so as the following field.

6.3.1 Observation space

We have the 12 horses and 1160 columns for each horse, so the observation space is (12 * 1160)

6.3.2 Action space

We have 12 horses, and one more option is ‘not bet’, so there are 13 discrete actions, ‘0’, ‘1’, ‘2’ … ‘12’ which represents ‘bet on horse 1’, ‘bet on horse 2’... ‘not to bet’.

6.3.3 Terminated state

The game is over when the agent has less than $10 cash balance which is different from the last model.

6.3.4 Reward function

Based on the previous experience,

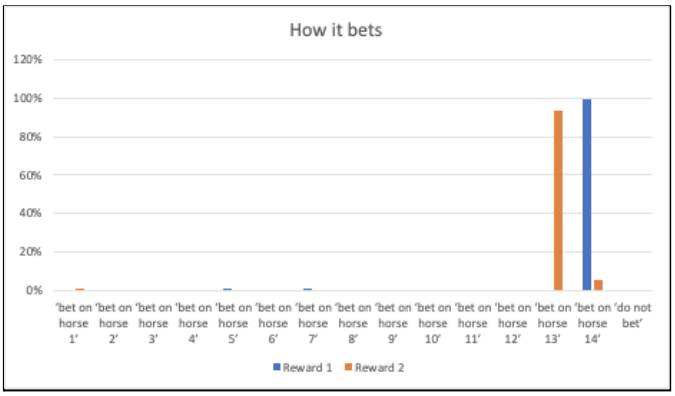

Figure 28. Comparison of ‘how agent bet’ between new and old reward function in testing set

- $R(bet\ and\ win) = C_1 * \Delta Cash\ Balance\ ,\ where\ C_1 > 0$
- $R(bet\ but\ lose) = C_2 * \Delta Cash\ Balance\ ,\ where\ C_2 < 0$
- $R(\ do\ not\ bet\ ) = C_3 * win\ odds\ of\ the\ true\ first\ place,\ where\ C_2 < C_3 < 0$

We construct in this way to prevent the agent from always choosing ‘do not bet’ and try to bet more. However, resulting in the situation that the agent doesn’t choose ‘do

not bet', so we change the reward function this time. By enlarging the reward of 'bet and win' to encourage to bet and win more, and reduce the penalty of 'do not bet'.

However, it is still hard to reshape the reward function, since the training time is time-consuming.

6.3.5 DQN with MLP policy

There are two hidden layers with 25 neurons each and using the tanh activation function, 13 neurons at the output layer.

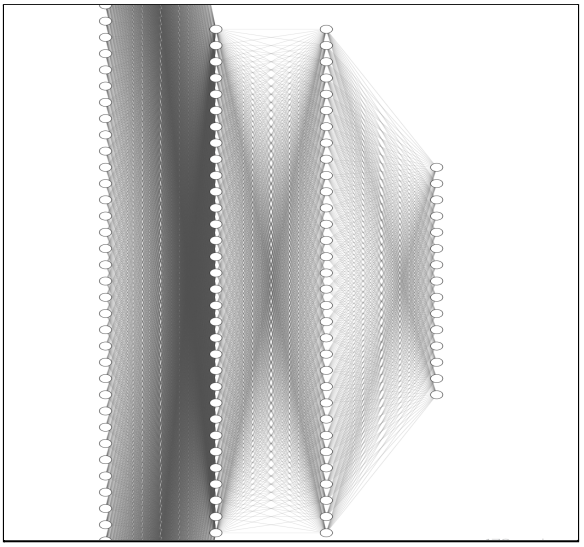

Figure 29. Network structure

6.3.6 Input order

Since the input order matters, we want to provide more information for the model, so we ordered the horses by using XGBoost. The first horse in the observation space is the fastest horse we predicted using XGBoost.

6.4 Result

6.4.1 Convergency

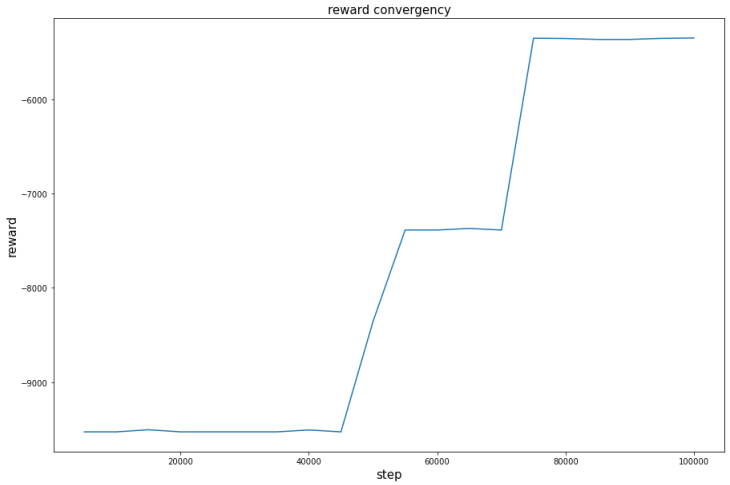

Figure 30. Convergence of DQN

The model is converged after training for 100000 timestamps.

6.4.2 Actions

6.4.2.1 Training set action count

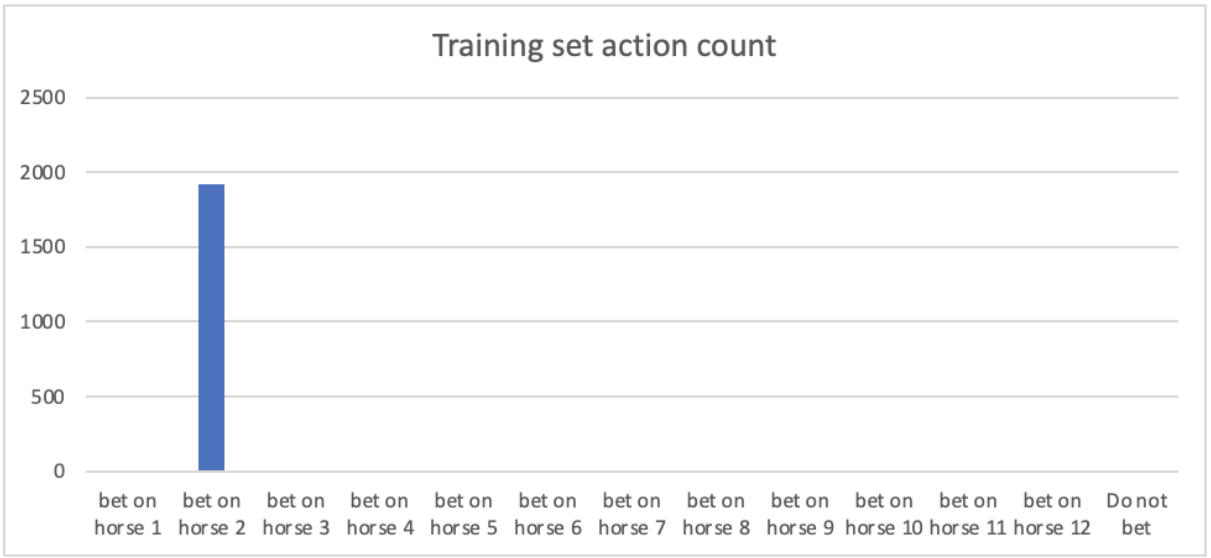

Figure 31. Action count in Training set

There are 1922 bets on horse 2, which is the second fast horse we predicted by using XGBoost. The agent only bet on the horse 2. This behaviour is the same as before.

6.4.2.2 Testing set action count

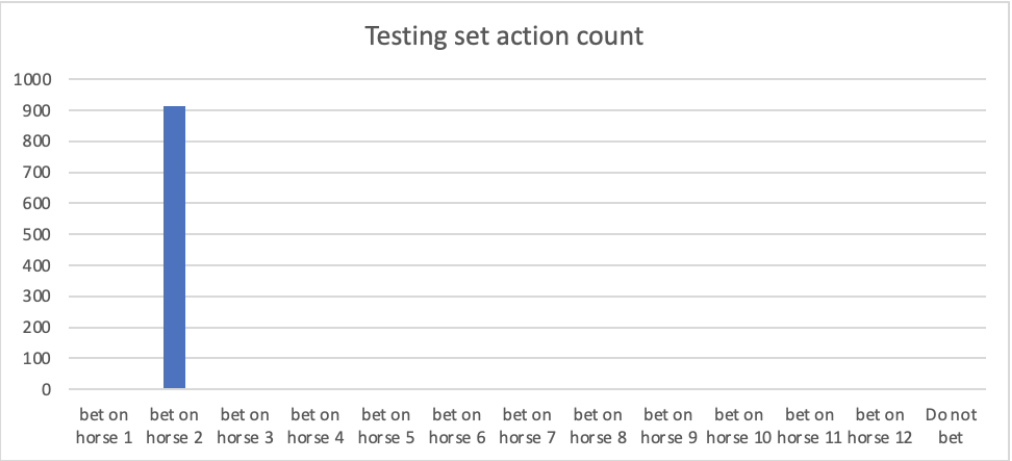

Figure 32. Action count in Testing set

Obviously, the agent performs the same behaviour in the testing set. The agent only bet on horse 2.

6.4.3 Win rate

6.4.3.1 Training set win ratio

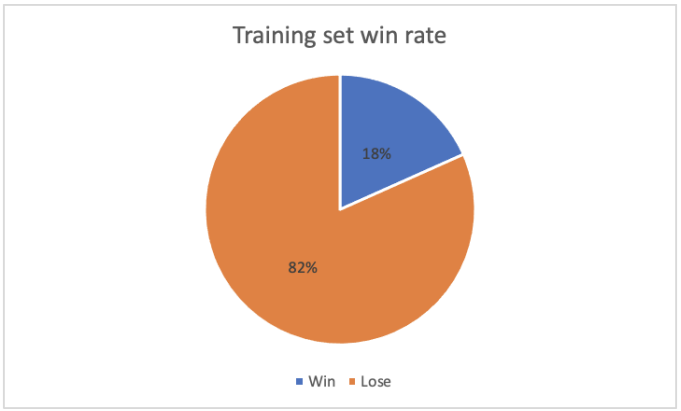

Figure 33. Action count in Testing set

The agent has an around 18% win rate. It wins 352 racing games out of 1922 racing games.

6.4.3.2 Testing set win ratio

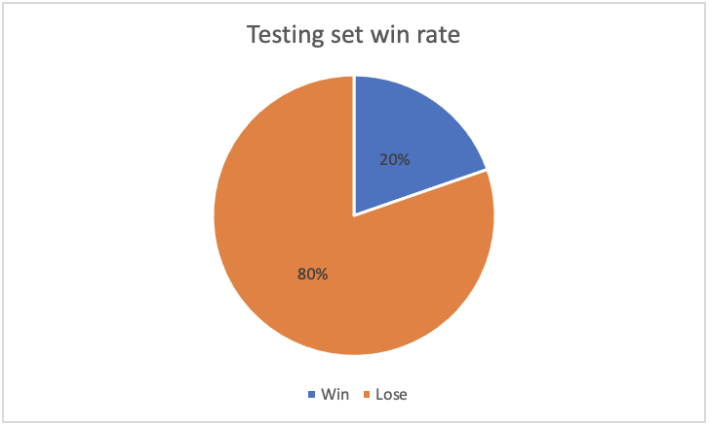

Figure 34. Action count in Testing set

The agent has around a 20% win rate in the testing set. It wins 180 racing games out of 915 racing games.

6.4.4 Cash balance

6.4.4.1 Training set cash balance

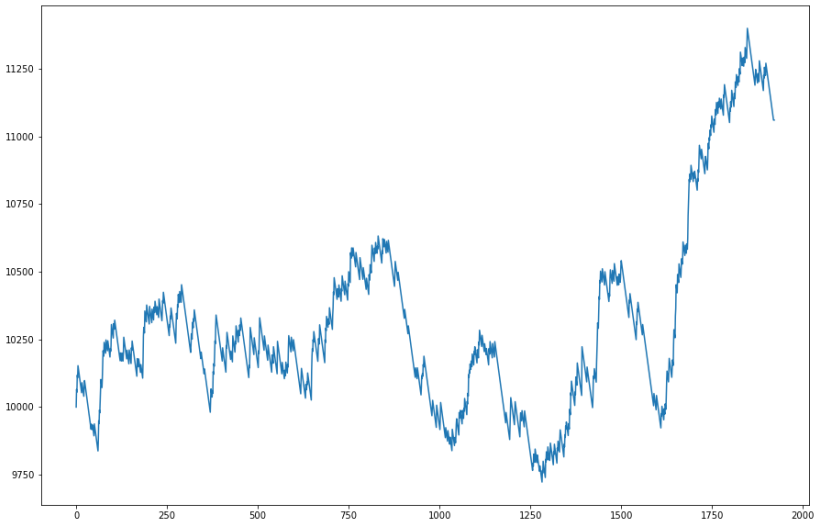

Figure 35. cash balance in training set

The agent wins $1060 at the end. The maximum cash balance is $11399, and the minimum is $9723. The largest win in a single racing game is $180.

6.4.4.2 Testing set cash balance

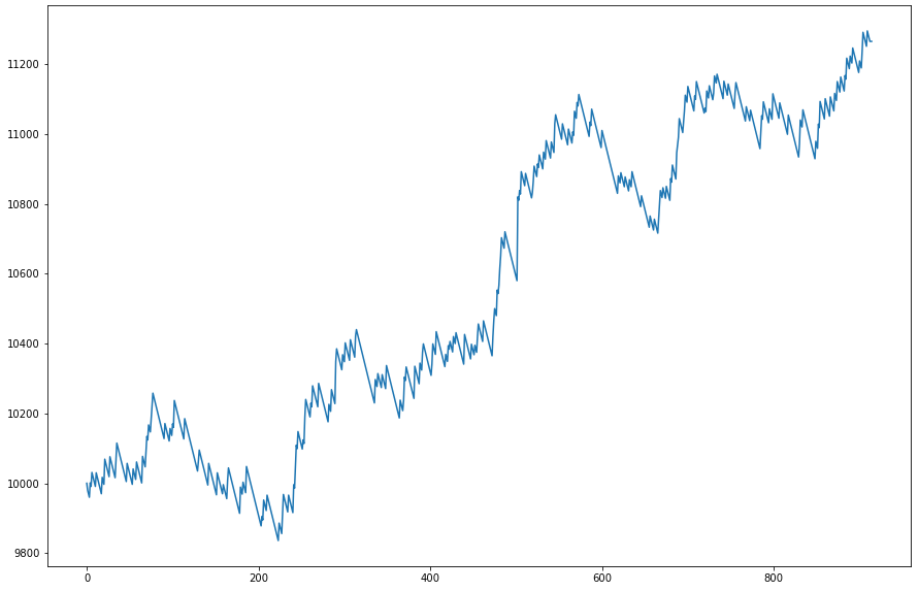

Figure 36. cash balance in testing set

The agent wins $1265 at the end. The maximum cash balance is $11295, and the minimum is $9836. The largest win in a single racing game is $240.

6.5 Result analysis

It is easy to see that DQN plays safe, its main bet on the horse 2, which is the second fastest. Although the betting action is simple, it is more profitable than other strategies.

However, since I have trained so many times, and reshape the reward function again and again, I think that it is too time-consuming, so we decide to find another method instead struggling in improving the performance of DQN.

6.6 Conclusion

The model is not performing very well, the model seems like betting on the position of the horses rather than betting on the horses. The results are close to the model betting on all kinds of racing games, which is not what we want to see.

To improve this problem, we can have better feature engineering, reshape the reward function again and again to figure out the best reward function, or even try to construct a better environment. However, it is too time consuming.

In order to have better results, we decided to use more advanced models A2C and PPO.

Chapter 7 Apply PPO in horse racing

7.1 Policy gradient

PPO is a policy gradient algorithm in Reinforcement Learning.

Policy gradient method is focusing on modeling and optimizing the policy directly by calculating the policy function $\pi(a \mid s)$ which is also called as actor, where π is a probability distribution on action a under state$s$. The most possible 'right' action will be chosen.

7.1.1 Basic policy gradient

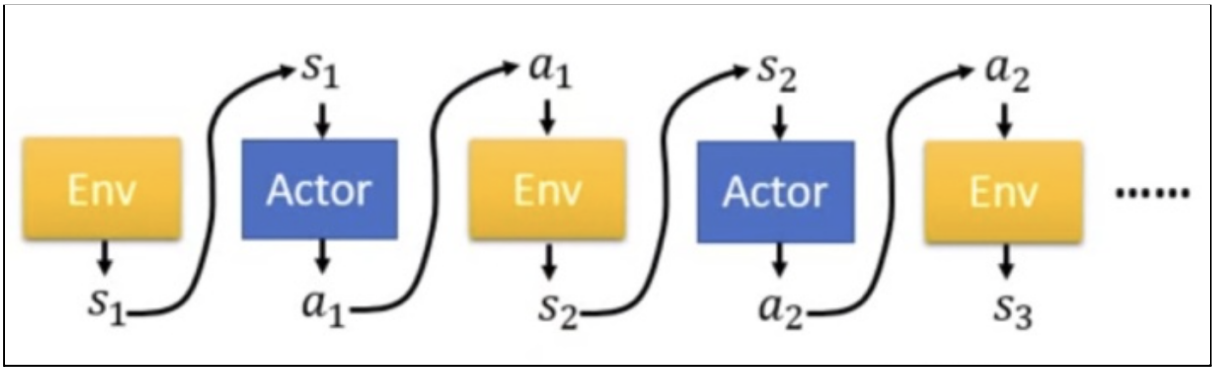

Figure 37. Flow of the policy gradient

Trajectory$\tau = \{s1, a1, r1, s2, a2, r2, ..., sT, aT, rT\}$ is the chain of policy gradient, to treat it as a probability

$$P(\tau|\theta) = p(s1)p(a1|s1,\theta)p(r1,s2|s1,a1)p(a2|s2,\theta)p(r2,s3|s2,a2)...$$
$$= p(s1)\Pi(p(a_t|s_t,\theta)p(r_t, s_{t+1}|\, s_t,\, a_t)).$$

Formula 15. The probability form of τ

In this formula, only the second part $p(a_t|s_t, \theta)$ is controlled by our actor π_θ, and we can also receive the reward from the action.

The expected reward is $R_\theta = \sum R(\tau)p_\theta(\tau) = E_{\tau\sim p_\theta(\tau)}[R(\tau)]$ by assuming $R(\tau) = \sum r_t$. Then our mission is to maximize the expected value by gradient ascent: $\nabla R_\theta = \sum R(\tau)\nabla p_\theta(\tau) = \sum R(\tau)p_\theta(\tau)\nabla log p_\theta(\tau)$.

Formula 16. break down the gradient of Reward

We have $P(\tau|\theta) = p(s1)\Pi(p(a_t|s_t, \theta)p(r_t, s_{t+1}|\, s_t, a_t))$, by taking log and ignore

Formula 17. The probability form of τ

the terms not related to θ.

We obtain the following equation: $\nabla log P(\tau|\theta) = \sum \nabla log p\ (a_t|s_t, \theta)$.

Formula 18. The probability form of τ

Then we combine these equations, we have what we want:

$$\nabla R_\theta = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{Tn} R(\tau^n)\nabla log\, p\ (a_t^n|s_t^n, \theta) = E[R(\tau)\nabla log p_\theta(\tau)]$$

Formula 19. The expected reward

It makes the use of reward to increase the probability of choosing the 'right' action. if $R(\tau^n)$ is large positive number, then ∇R_θ is larger, vice versa, then we can maximize

the probability that brings us the largest expected reward. One thing we need to be careful is that the Reward is always positive, and if some action is not chosen, then the probability of those chosen actions will always increase, so we may need to subtract a constant to make it negative. We can replace the reward $R(\tau^n)$as an advantage function or other meaningful function such as advantage function.

7.1.2 On-policy to Off-policy

It is obvious that we have to update our policy after each of the transaction, we have to sample it again and again, which causes an extremely inefficient training, so we do an important sampling $E_{x\sim p}[f(x)] = E_{x\sim q}[f(x)\frac{p(x)}{q(x)}]$ to achieve off policy then we

Formula 20. importance sampling

can make use of another distribution which called actors to get those sample data. Then we have $\bigtriangledown J^{\theta'}(\theta) = E_{\pi_{\theta'}}[\frac{\pi_\theta(s,a)}{\pi_{\theta'}(s,a)}R(s,a)\bigtriangledown log\pi_{\theta'}(s,a)]$.

Formula 21. The objective function of Off-policy[26]

We use the θ' actor to interact with the environment to sample the data, then we use it to update our actual actor θ. Since actor θ' is not updated, so that the sample data can be used again and again. The importance weight $\frac{P_\theta(a_t|s_t)}{P_{\theta'}(a_t|s_t)}$ will adjust expected reward. However, it may cause overfitting easily once the probability is too far away.

7.2 Why PPO

As we discussed that the policy is updated based on maximizing the expected reward, however, the update of the policy is too sensitive as the importance weight may be large, it may cause a bad performance of the model. So to limit the update of the new policy is necessary, which is the key value of PPO.

7.2.1 Trust region

It is introduced in TRPO, $\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right]$ which is subjected to the

Formula 22. The expected reward[27]

Kullback–Leibler divergence, the constraint is $\hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta.$ [27], combine the constraint inside the equation we have $\underset{\theta}{\text{maximize}}\, \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta\,\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]\right]$,

Formula 23. The objective function of TRPO[27]

then we can avoid the expected reward being too large.

7.2.2 Clipped PPO

The CLIP-PPO: $L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$,where $r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$.

Formula 24. The objective function of CLIP [26]

Instead of using the KL penalty, it create a lower bound/upper bound on $r_t(\theta)\hat{A}_t$ with $1-\epsilon, 1+\epsilon$ [26], which can prevent the new policy from being too far away from the old policy. We want to keep $r_t(\theta)$ close to 1 which means that the policy is updated slowly. We can see that the KL divergence of the new policy using clipped update is lower than other[26].

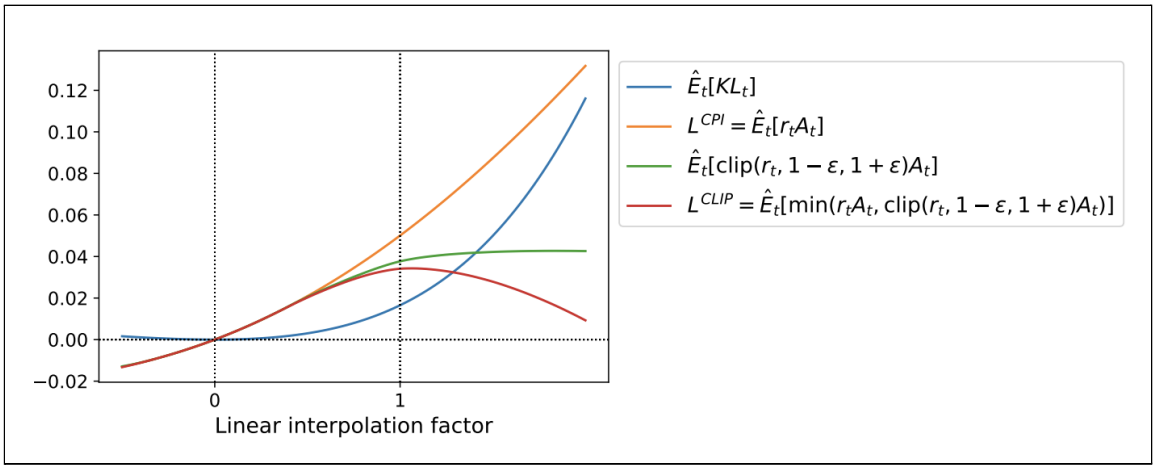

Figure 38,KL divergence [26]

7.2 Construction

We use the Stable Baselines3 and OpenAI Gym[9] to construct our PPO reinforcement learning. Moreover This chapter is focused on the 12 candidates horse racing games.

7.2.1 Environment

We have 10000 cash balance, and if the agent wins, we give him 10 * win odds cash. If he loses, we take 10 cash from him. That means the agent can only bet 10 dollars.

7.2.2 Action Space

We have 12 horses, and one more option is ‘not bet’, so there are 13 discrete actions, ‘0’, ‘1’, ‘2’ … ‘12’ which represents ‘bet on horse 1’, ‘bet on horse 2’... ‘not to bet’.

7.2.3 Step

After the agent goes through one games, the agent will receive first two major informations

4. The new game state : feature values of all 12 horses of the next game and the cash balance after the last game.
5. The reward obtained based on the last action.
6. Game over or not.
    a. Once the agent loses more than 9990 dollars where the cash balance < 10, then it is game over.
    b. Once it goes through all the racing games, it is game over too.

7.2.4 Reset

In order to train the agent to learn which horse is nice to bet, other than recognizing the winning horses in each state, the state jumps to random racing games everytime we reset, and the cumulative reward will be reset.

7.2.5 Termination state

In horse racing games, we said it is game over when the agent loses more than 9990 dollars or when it goes through all the games.

7.2.6 Reward function

Based on the previous experience, We will use the same way to construct our reward function.

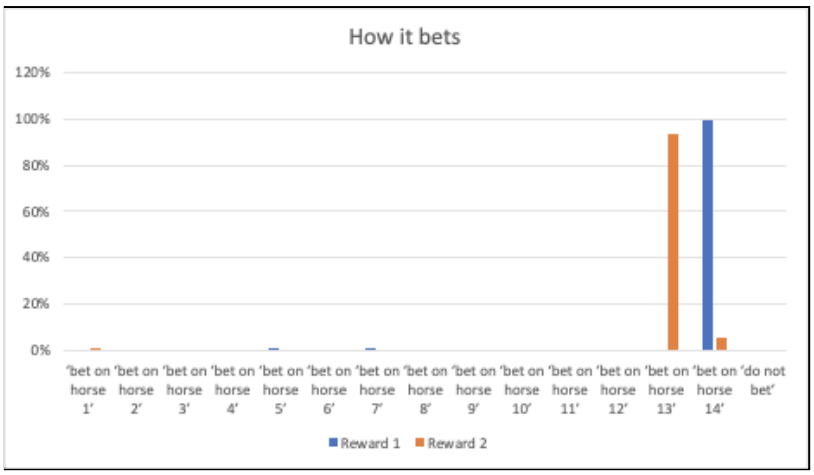

Figure 39. Comparison of 'how agent bet' between new and old reward function in testing set

- $R(bet\ and\ win) = C_1 * \Delta Cash\ Balance\ ,\ where\ C_1 > 0$
- $R(bet\ but\ lose) = C_2 * \Delta Cash\ Balance\ ,\ where\ C_2 < 0$
- $R(\ do\ not\ bet\ ) = C_3 * win\ odds\ of\ the\ true\ first\ place,\ where\ C_2 < C_3 < 0$

We construct in this way to prevent the agent from always choosing 'do not bet' and try to bet more. However, resulting in the situation that the agent doesn't choose 'do not bet', so we change the reward function this time. By enlarging the reward of 'bet and win' to encourage to bet and win more, and reduce the penalty of 'do not bet'.

However, it is still hard to reshape the reward function, since the training time is time-consuming.

7.2.7 DQN with MLP policy

There are two hidden layers with 25 neurons each and using the tanh activation function, 13 neurons at the output layer.

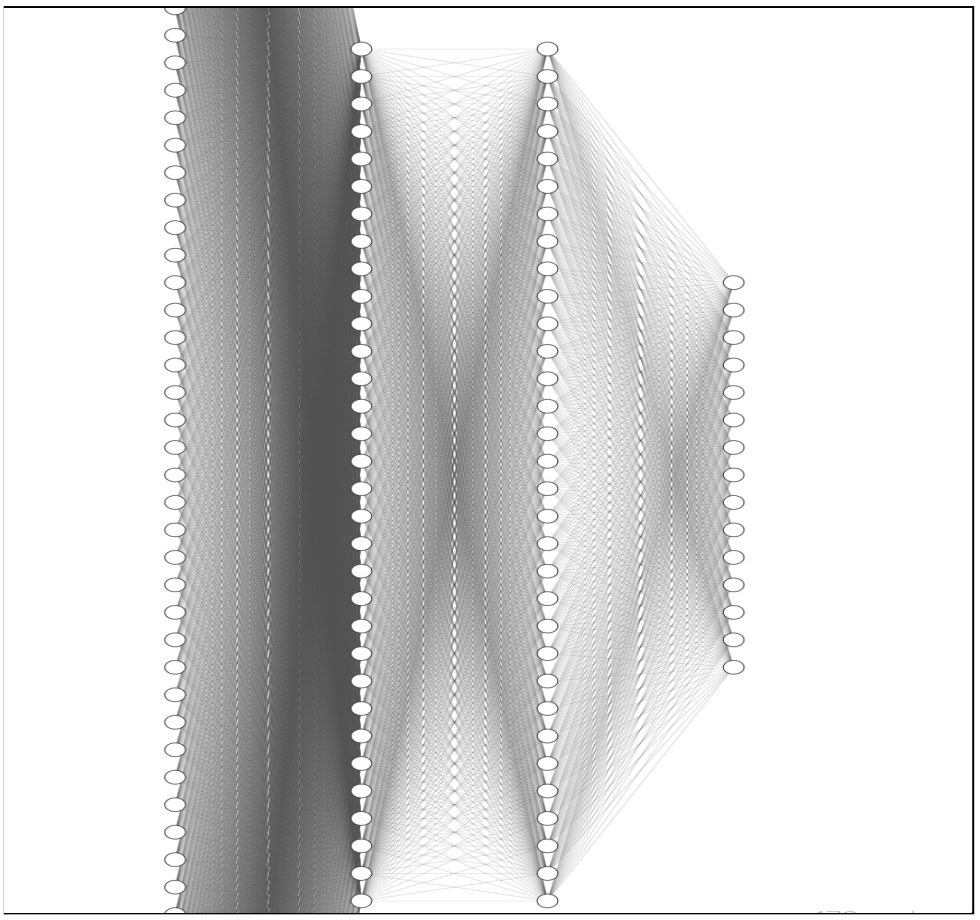

Figure 40. Network structure

7.2.8 Input order

Since the input order matters, we want to provide more information for the model, so we ordered the horses by using XGBoost, it means that the horse with the smaller number is faster than the horse with larger number.

7.2.9 Hyperparameter

we set gamma as 1, the clip-range is 0.2.

7.3 Result

7.3.1 Convergency

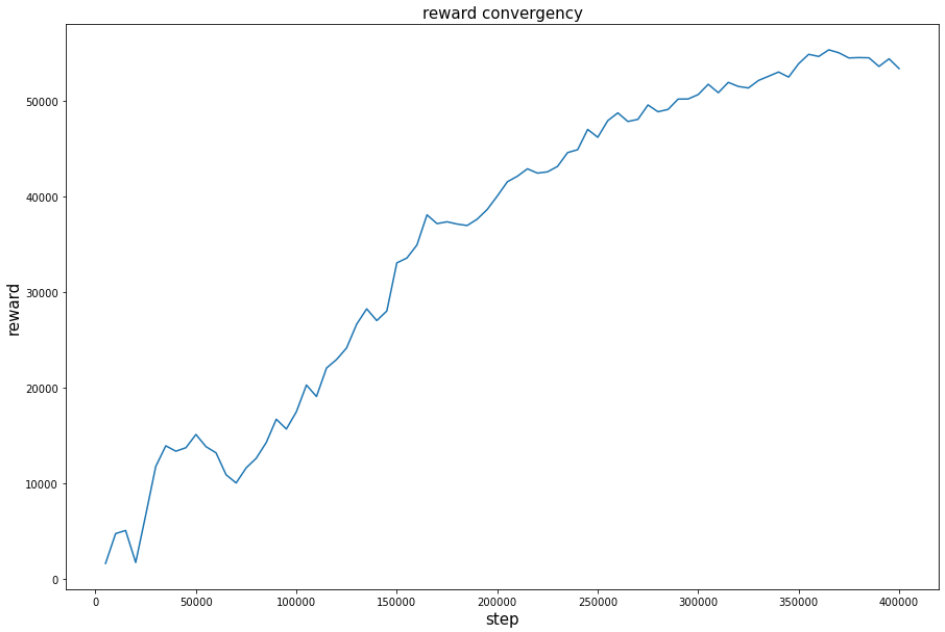

Figure 41. Reward convergence

The reward converged after around 400000 timestep.

7.3.2 Action count

7.3.2.1 Traning set

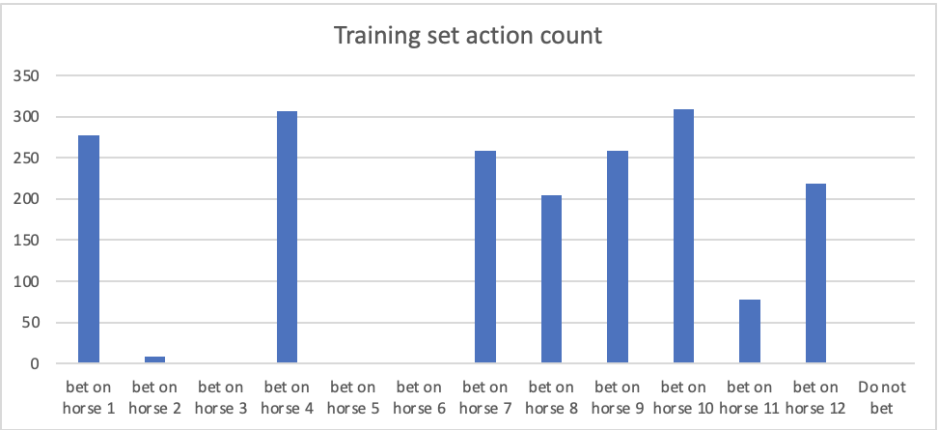

Figure 42, training set action count

The agent tends to bet on the large number horses which are predicted as slower using XGBoost but with a larger win odds. As before, the agent never chooses not to bet.[3]

7.3.2.2 Testing set

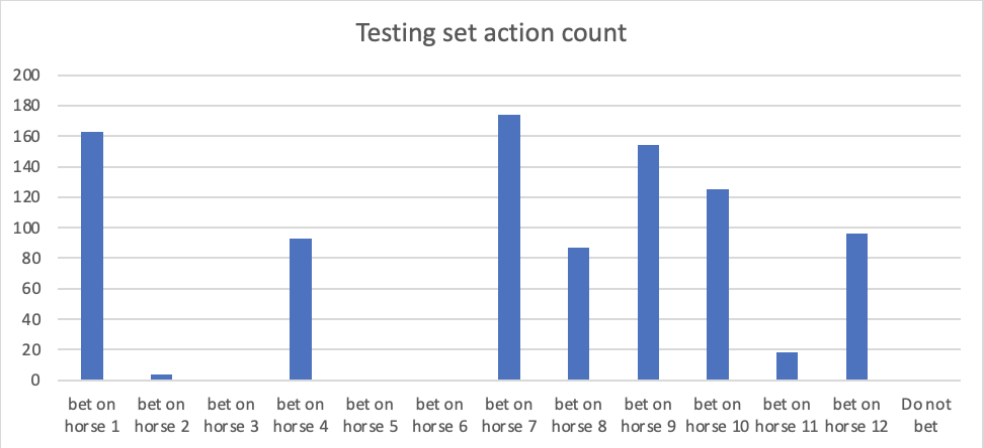

Figure 43, testing set action count

The agent tends to bet on the large number horses which are predicted as slower using XGBoost but with a larger win odds. The choice is similar to the training set, except horse 7 is the most popular choice in the testing set while horse 10 is the most popular one in the training set.[4]

[3] The number is shown in appendix
[4] The number is shown in appendix

7.3.3 Win rate

7.3.3.1 Traning set

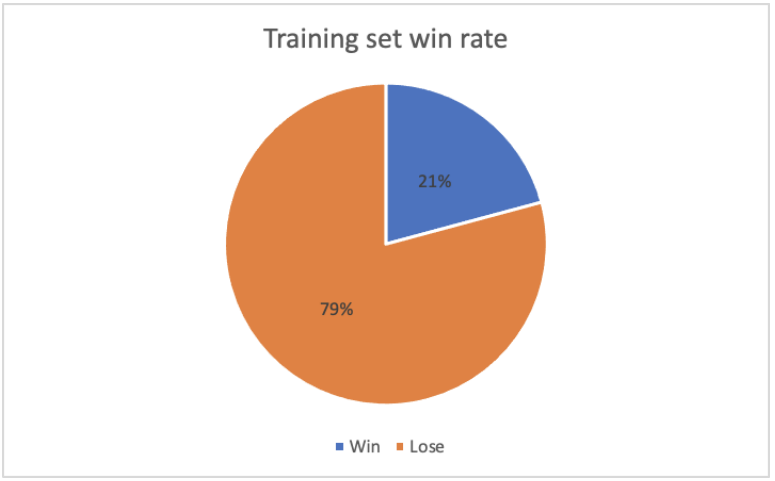

Figure 44, training set win rate

The agent wins 400 out of 1922 games, which is around a 20% win rate.

7.3.3.2 Testing set

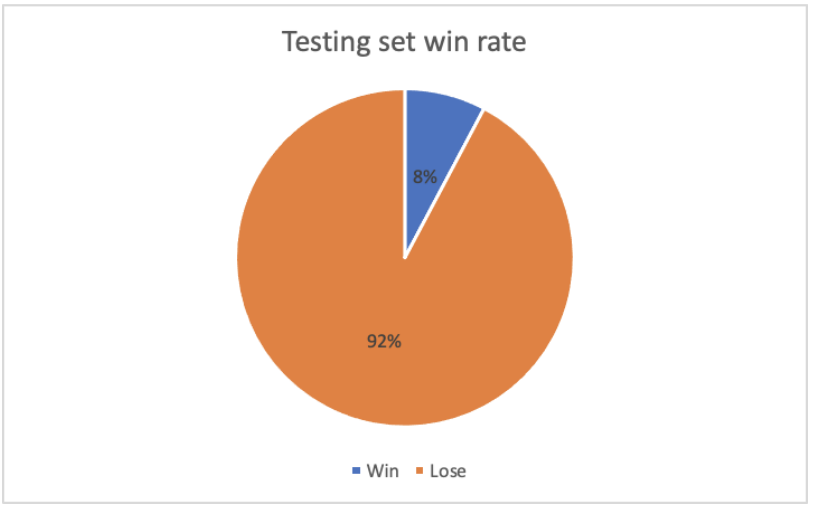

Figure 45, testing set win rate

The agent wins 71 out of 915 games, which is around 8% win rate, the performance is much worse than in the training set.

7.3.4 Cash balance

7.3.4.1 Training set

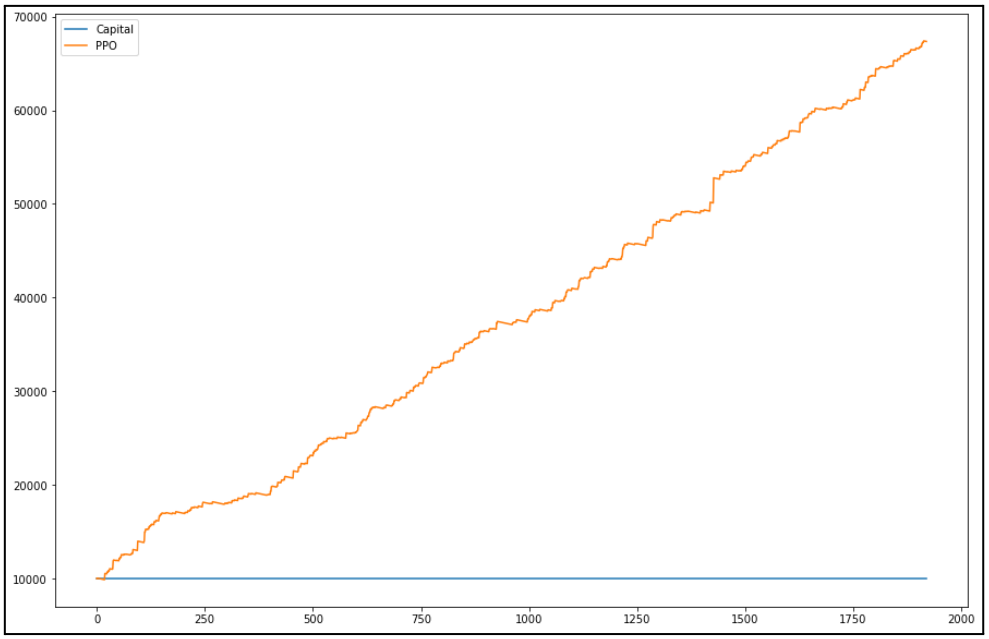

Figure 46, training set cash balance

The agent has 67,356 cash balances at the end. The minimum cash balance is 9877. The largest gain is 2680, overall, the model is well trained for the training data.

7.3.4.2 Testing set

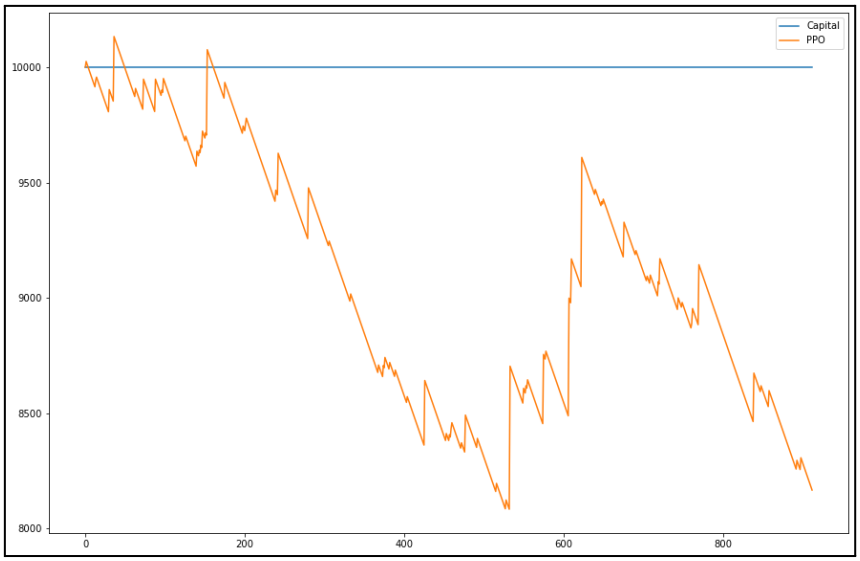

Figure 47, testing set cash balance

The agent has 8,167 cash balances at the end. The minimum cash balance is 8084. The largest gain is 620$. On average, the agent performs poorly, it loses most of the time.

7.4 Results analysis

The behaviour of the model tends to bet on those horses with higher win odds instead of higher chance to win. Since higher win rate doesn’t imply higher return. This is a reasonable behaviour.

This time we don't tune any parameter, and just train it long enough. The performance of the model performs so well in the training set, however, it performs poorly in the testing set. Intuitively, It may be caused by the overfitting or the setting is not tuned well.

We would like to achieve higher returns, not a higher win rate.

We may have two simple approaches to improve the model:

1. Early stop the training
2. Decreased the clip range by 0.05[26]
3. Increase the reward of winning, reward more if win more

7.5 Improvement

The setup will be the same as before, since the model performs well in the training data.

Based on the previous chap, the following option is made:

1. Early stop the training
2. Decreased the clip range by 0.05[26]
3. Increase the reward of winning, reward more if win more

7.5.1 Convergence

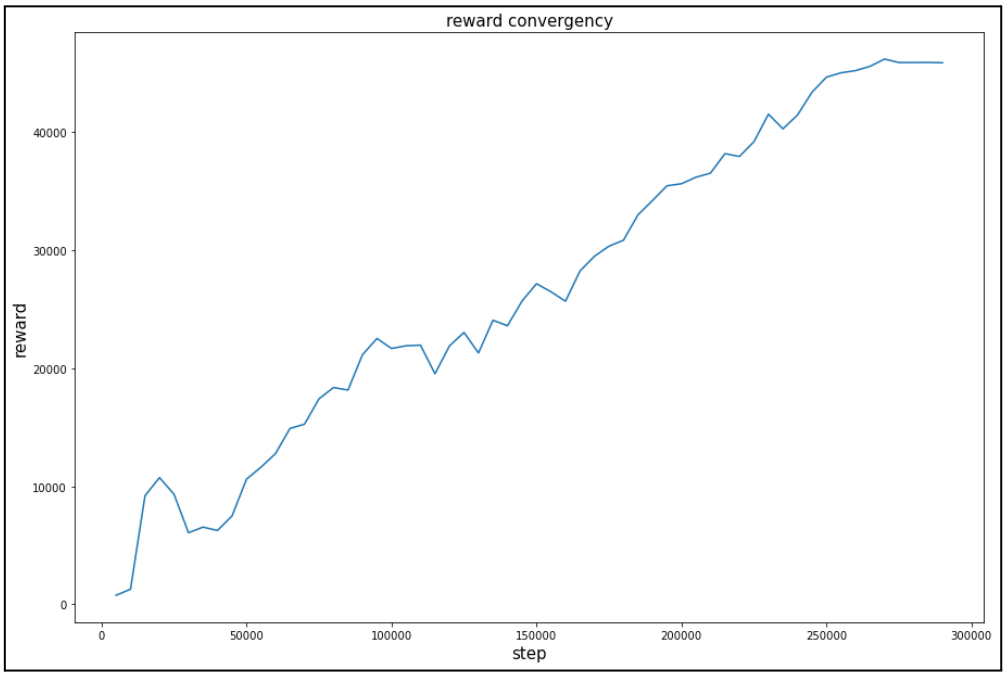

Figure 48, reward convergence

Since I want to stop the training early to avoid overfitting, I stop the training on the 300,000 step.

7.5.2 Action count

7.5.2.1 Traning set

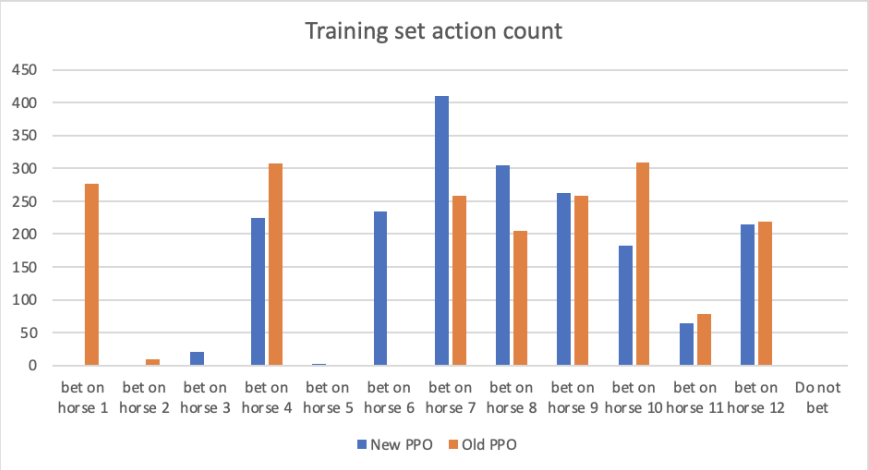

Figure 49, training set action count

The agent tends to bet on the large number horses which are predicted as slower using XGBoost but with a larger win odds. In this model, this behaviour is more obvious. As you can see, the new PPO never bet on horse 1 but it bet on horse 6.

7.5.2.2 Testing set

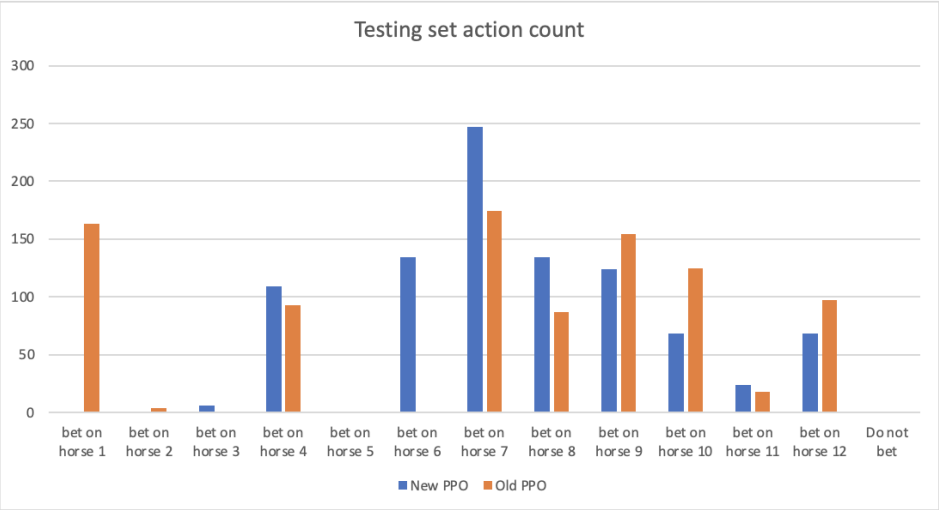

Figure 50, testing set action count

The same situation occurs in testing sets, it never chooses not to bet and tends to bet on horses with large numbers which are the slower with higher win odds. The new model chose ‘bet on horse 6’ rather than choose ‘bet on horse 1’.

7.5.3 Win ratio

7.5.3.1 Traning set

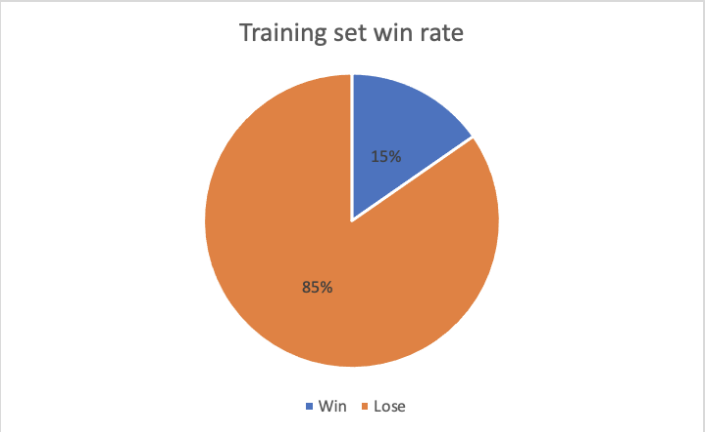

Figure 51, training set win rate

The agent wins 294 out of 1922 games, which is around a 15% win rate. Which is lower than the last model by 5%. It is expected as we stop the training early and we increase the reward of winning a large return.

7.5.3.2 Testing set

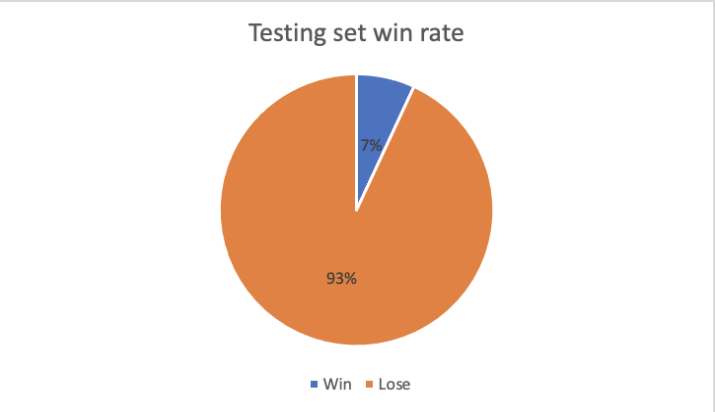

Figure 52, testing set action count

The agent wins 63 out of 915 games, which is around a 7% win rate. Which is lower than the last model by 1%. It is expected as we stop the training early and we increase the reward of winning a large return. The win rate doesn't really matter as we only care about the return.

7.5.4 Cash balance

7.5.4.1 Training set

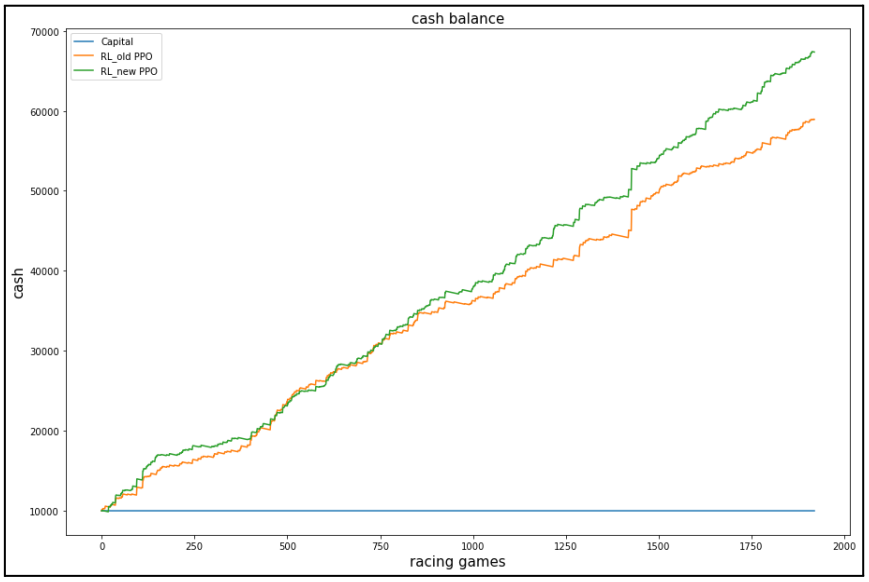

Figure 53, training set cash balance

The orange one is our new model. The agent has 58,938 cash balances at the end. The minimum cash balance is 10000. The largest gain is 2680$. The model is also well trained.

7.5.4.2 Testing set

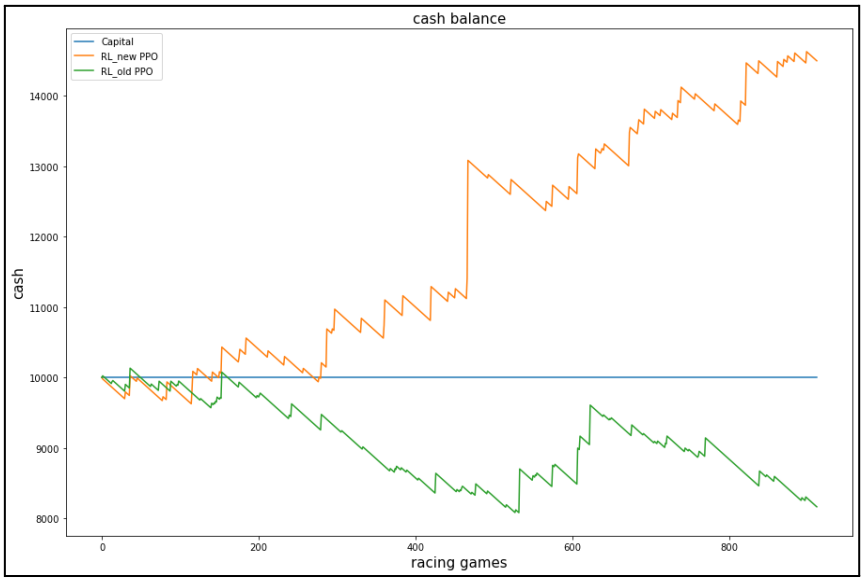

Figure 54, testing set cash balance

The orange one is our new model. The agent has 14,492 cash balances at the end. The minimum cash balance is 9628. The largest gain is 1670$. which is larger than the previous model around $6k.

7.5.5 Results analysis

After tuning the clip range and reshaping a bit of the reward function, we can obtain better results in the testing set.

Although the win rate is only 7%, the winning amount is large, so although we lose 852 racing games, which is a loss of $8520, we win $13012 at total, so we can have a positive return of 14,492.

The behaviour of the agent is to bet on slower horses predicted with a higher win odds, which can bring us a large return as you can see in the testing set.

Comparing to the previous model, this new model does not bet on horse 1 and bet more on horse 7, which may have a higher return. The behaviour of ‘higher risk higher return’ is more obvious, which is caused by reshaping of the reward function.

7.6 DQN and PPO

7.6.1 Comparsion

Two different approaches in reinforcement learning. If we consider the win rate, the DQN model is better than the PPO model. However, it is obvious that the performance of the PPO is better than the DQN by comparing the return.

7.6.2 Performance and strategy

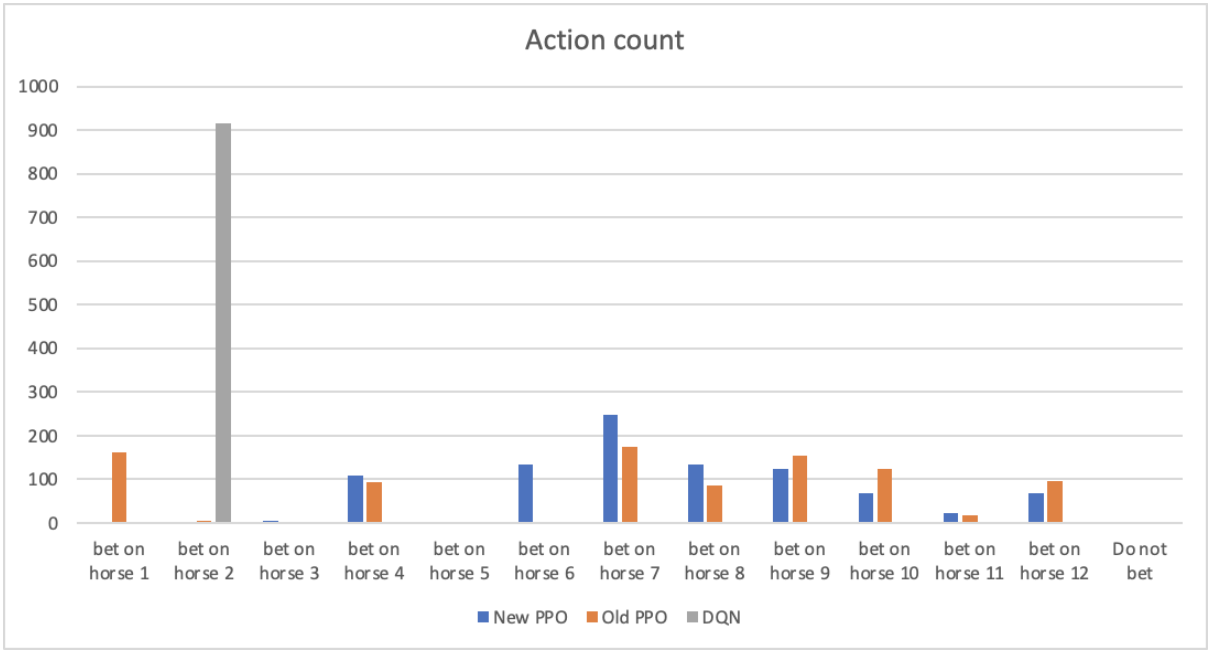

Figure 55, comparison of action count

The win rate of DQN is the largest, having 20% of win rate, which is because the DQN tends to bet on safer horses. The number of the horse is the input order of the

horse, we ordered it by XGBoost prediction, the smaller the number, the faster the horse is predicted. As we can see, the PPO can achieve a strategy that bet on higher return horses. However, we think that betting is too simple by using DQN, we think that playing horse racing games is too complicated for DQN.

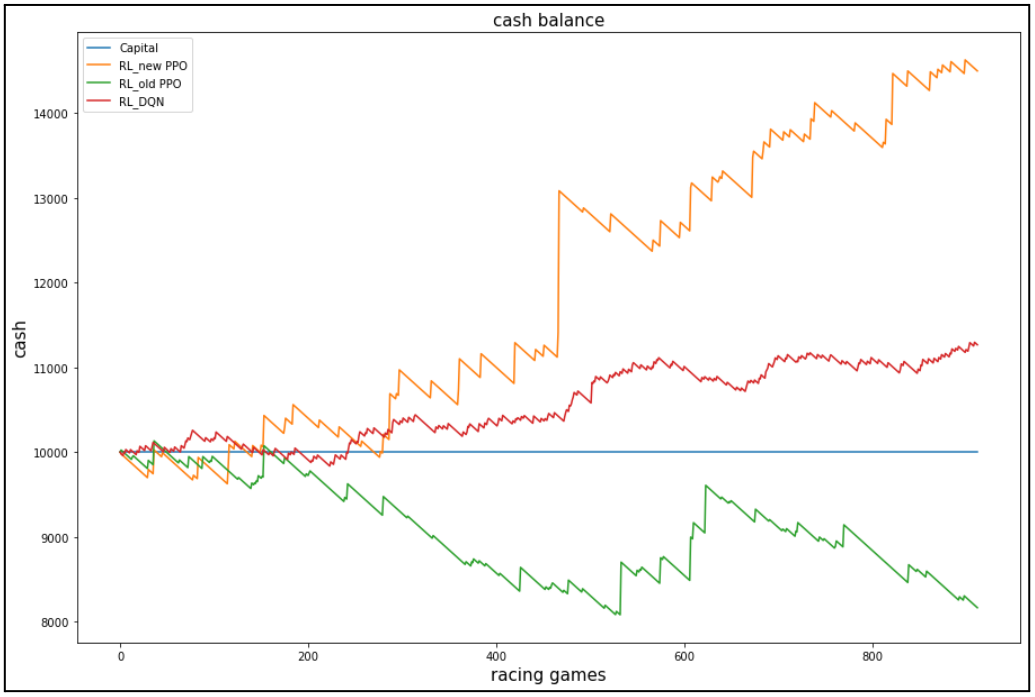

Figure 56, comparison of cash balance

In this chart, we can see that the DQN model plays more safer than PPO, unlike PPO having many ‘jump’s, DQN has nearly none big jumps. The new PPO has more return than the DQN by 3k. It is because the strategy is different, so the shape of the curve and the performance is different.

7.7 Overall conclusion

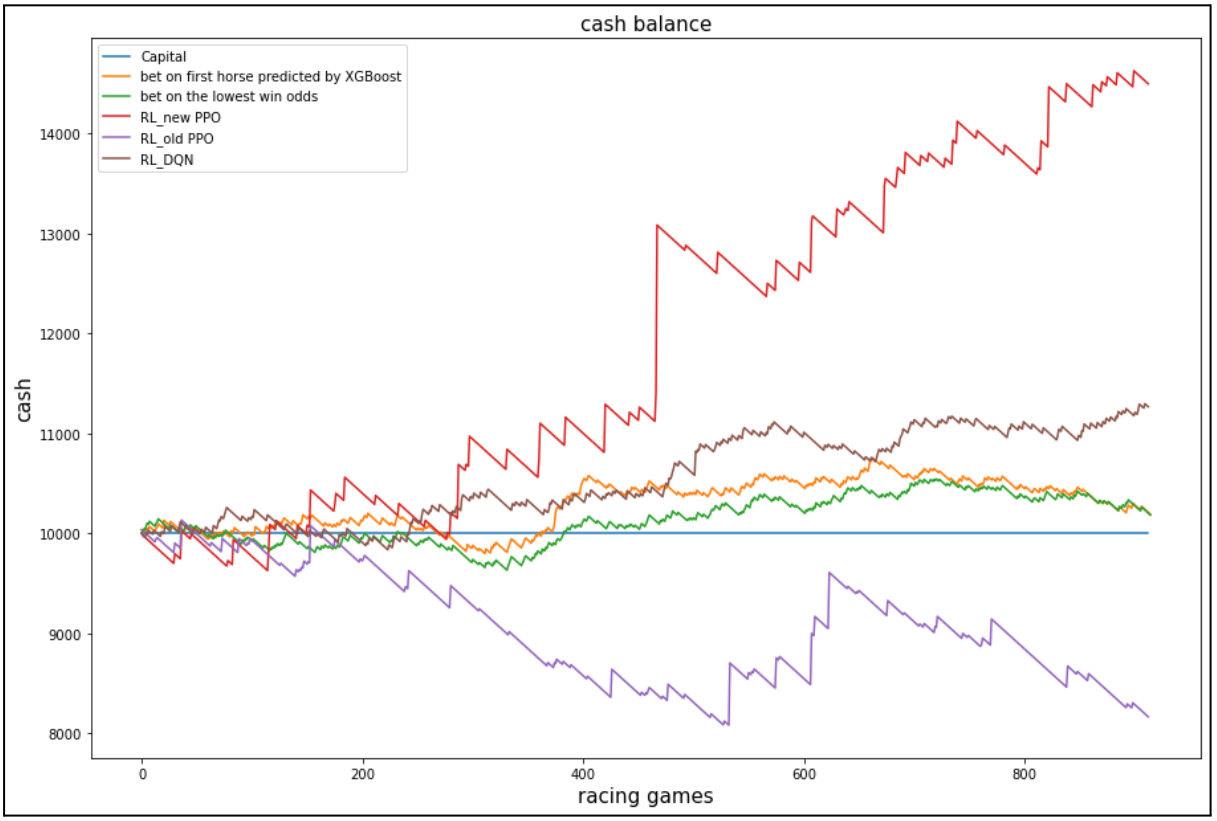

Figure 57, comparison of cash balance of all strategies

The performance of DQN and new PPO is better than other common strategies such as betting on the lowest win odds horse, or bet on the first horse predicted by XGBoost.

The performance of PPO is better than DQN since we do not care about the win rate but the return, we have 4,492 net gains. But we could have a better set up for the PPO to have a better performance and to adjust the strategy.

Chapter 8 Applying A2C in horse racing

In this chapter, A2C will be used to train the agent to learn how to bet horse racing. There will be two main betting types which are “win” and “place”. These two betting types have been introduced in the first chapter.

8.1 Advantage Actor Critic (A2C)

In this section, we are going to introduce the advantage actor critic (A2C). A2C is one of the most common reinforcement learning methods introduced by Google.

Before talking about A2C, the basics of A2C will be introduced and that is actor critic. Also, actor critic is based on Q learning and policy gradient. Therefore, let’s start from policy gradient.

8.1.1 Policy Gradient

Policy gradient is quite similar to Q-learning. In Q-learning, the Q table will record the state and the relative action. Therefore Q-learning will directly output an action regarding to which state the agent is at. What policy gradient does is to output the probability of an action. The training is to maximize the probability of the correct action to the agent, so the agent can choose that action to gain the maximum reward.

$$\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$$

Formula 25 Policy gradient

This is the function of policy gradient. The middle component is very important to the whole algorithm. It means that if the probability of the action chosen is low and the reward The last component is the reward function which means after the action is

chosen under a state, the reward is given (which can be positive or negative). The update function is shown below:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Formula 26 Update function

The probability will be increased or decreased due to the reward. The whole process will be:

1. The agent choose the action based on the probability from the policy
2. Calculate the formula 24
3. Update the policy

However, policy gradient will not be used since policy gradient is too simple to deal with the horse racing problem. Now, a better and more complicated algorithm will be introduced and it is called actor critic.

8.1.2 Actor Critic

Actor critic is a combination of Q learning and policy gradient. The actor critic is the basic of many advanced algorithms including advantage actor critic and proximal policy gradient. Actor is the policy gradient while the critic is the Q learning. To compare with policy gradient, the reward function will be changed. In policy gradient, the reward is directly obtained from the environment, but in actor critic, it is calculated from the critic which is a temporal learning.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - V(\mathbf{s}_{i,t})\right)$$

Formula 25 Policy gradient with Q-learning

This is the function of an actor critic algorithm. This is very similar to the one in policy gradient but the last part is different. This is the reward from the critic.

$$Q(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta} \left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t \right]$$

Formula 26 Q-learning Update Function

$$V(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q(\mathbf{s}_t, \mathbf{a}_t)]$$

Formula 27 Function of all expected Q-table's output

There are three components from this function. The first part is the reward from the Q learning. The second and third parts are the difference of the reward calculated by the Q-learning and it is called temporal difference. The full process is:

1. The agent choose the action based on the probability from the policy
2. Find the reward based on the Q learning and the expected value from previous.
3. Calculate function formula 25
4. Update the policy by update function

To conclude, this is the basic theorem of actor critic. People improved actor critic by using different Q learning and also changing the update function of the policy gradient.

8.1.3 Advantage Actor Critic

Advantage actor critic (A2C) is introduced by Google DeepMind.[28] This is an updated version from actor critic.

The algorithm is different from actor critic. In A2C, we directly use the deep Q network as the critic side. The most important is the change of the algorithm.

$$\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$$

Formula 28 Policy gradient with advantage function

The last part is changed and this is called an advantage function.

$$\hat{A}^{\pi}(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \hat{V}_{\phi}^{\pi}(\mathbf{s}_i') - \hat{V}_{\phi}^{\pi}(\mathbf{s}_i)$$

Formula 29 Advantage function

The advantage function is to calculate the value of the state. In actor critic, the last part of the function is calculating the reward of the action. In A2C, it calculates the reward of the action and the value of the state. In horse racing, this can be used to calculate the state which is all feature values of the horses. It will be mentioned in the forward chapter.

The last difference of A2C is to set up multi workers to train the model and use multi-threads to boost up the training. Each worker and the global network is an actor critic network. For each update, each worker will run a full process of the algorithm then update the experience to the global network. The coordinator will wait for all workers to update the experience in each step. This can highly increase the efficiency of the training. The example with 3 workers is shown below.

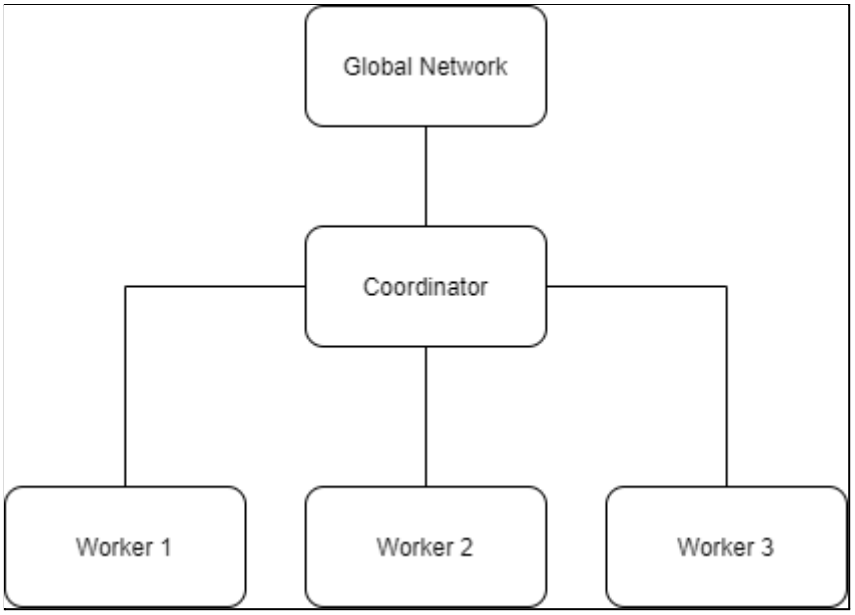

Figure 58 The network of A2C

The multi-threads in the CPU can be used. The following graph shows the CPU usage while training:

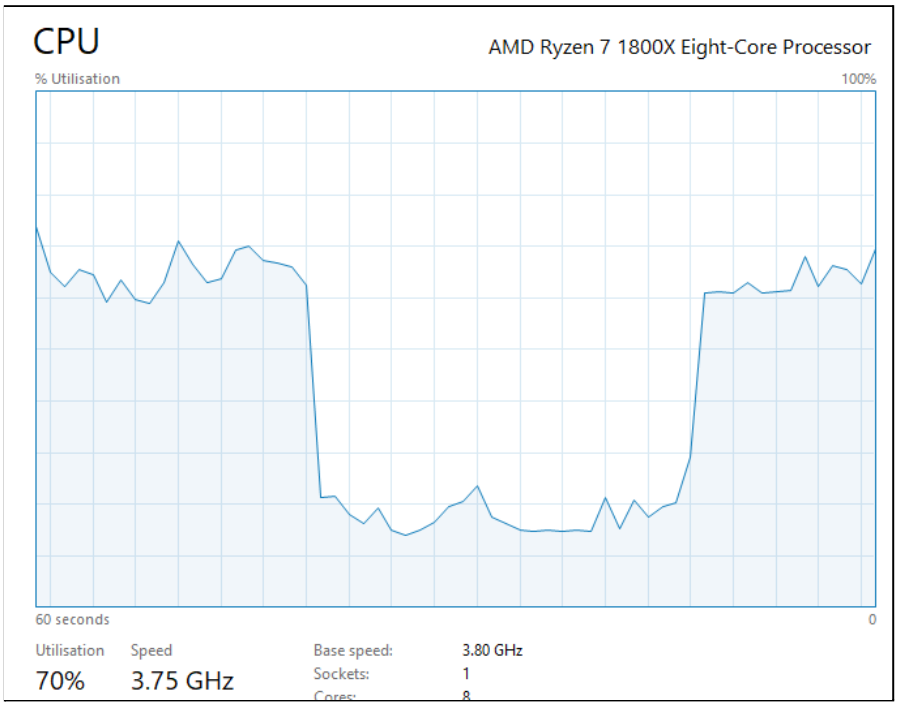

Figure 59 CPU Usage

8.2 Construction

Stable Baselines3 and OpenAI Gym[9] will be used to construct the A2C reinforcement learning.

8.2.1 Environment

The previous setting will remain so the initial balance will be 10000. In each betting, if the agent wins, he will get 10 * odds (win odds / place odds). If the agent losses, he will lose 10.

8.2.2 Observation Space

Using the previous setting, the observation space will be all features values of all the horses and the cash balance will also be included. We want the agent to learn how the feature values are useful or not. Since some of the horse racing games have less than 14 horses participated in, the invalid horses' feature values will be -99 as separating and the agent has to learn which is an invalid horse and which is a real horse.

8.2.3 Action Space

There will be 15 actions including choosing 14 horses (some of them may be invalid) and the last action will be choosing not to bet. {'1', … '15'} will be the action space. {'1',...,'14'} is choosing the horse to bet and {'15'} means not to bet.

8.2.4 Step

In each game, the agent will choose the action from the action based on the A2C function.

After each game, the agent will receive informations, including:

1. The new game states which are feature values of the next game's horses and the cash balance.
2. The reward obtained from the action

The environment will also check whether the game is over. The game is over when it goes through all the racing games. For example, if all testing cases have been run over, the game will be over. Also, for better training, if the cash balance is lower than 9000, the game will be over.

8.2.5 Reset & Terminate

In order to train the agent to learn which horse is better to bet by analysis instead of recognizing the winning horse in each game, the state will choose a racing game by random every time we reset.

For the terminate state, as mentioned in the last part, the game will be terminated when the cash balance is lower than 9000 or all games have been run over.

8.2.6 Reward & Penalty

The reward given is to guide the agent how to learn when the penalty (negative reward) is also given when the agent did something we do not want. The reward function may be changed to train a better agent. The initial thought of the reward function is:

- $R(bet\ and\ win) = C_1 * \Delta Cash\ Balance\ ,\ where\ C_1 > 0$
- $R(bet\ but\ lose) = C_2 * \Delta Cash\ Balance\ ,\ where\ C_2 < 0$
- $R(\ do\ not\ bet\ ) = C_3 * win\ odds\ of\ the\ true\ first\ place,\ where\ C_2 < C_3 < 0$
- $R(bet\ on\ invalid\ horse) = R(do\ not\ bet)$

This is based on the previous result from DQN. However, we may change the reward function to have a better result.

8.2.7 Discount Factor

Based on the previous result, there will not be any discount vector since betting on horse racing games is an episodic environment and each separated horse racing game will end in the same way. There will be no discount factor although the original algorithm has it in policy gradient. The agent will treat every game at the same level. Therefore gamma $\gamma = 1$will be used in all training.

8.2.8 Setting

There will be some important settings to be used in the training and testing.

- Gamma means the discount factor which will be 1 as mentioned
- Learning rate determines the step size in each iteration
- n_steps means the number of steps to run per update
- total_timesteps means the total number of steps in one training_step
- training_step means the total number of rounds for a complete training

8.2.9 Policy

Multilayer Perceptron (MLP) will be used as the Deep Q network in A2C. There will be two layers with 64 nodes and 1 hidden layer to process the Q network.

8.3 Betting on “Win”

In this section, we will start to train the agent to bet on horse racing with betting type “win”. “Win” means choosing the horse which is the first place.

8.3.1 Training

This part will include how we implement models and update them to have a better training result.

8.3.1.1 First Model

Now, we construct our first model to try to learn how to bet the winning horse with A2C. Since it is the first model, the result may not be satisfying.

8.3.1.1.1 Setting

The first model constructed uses the following setting:

The parameter:

- *learning_rate = 0.0007*
- *gamma = 1*
- *total_timesteps = 5000*
- *training_step = 20*

The reward function:

- *R(bet_win) = 10 * win_odds*
- *R(not_bet) = -10 * (win_odds from the winning horse)*
- *R(Invalid_bet) = -10 * (win_odds from the winning horse)*
- *R(bet_but_lose) = -10*

Here, we use the learning rate as 0.0007 to let the agent try the other strategy to find the best strategy in the whole training. Gamma will be 1 since we want there will be no discount factor and the agent can treat every game as independent to the others.

8.3.1.1.2 Result

The result is shown down below:

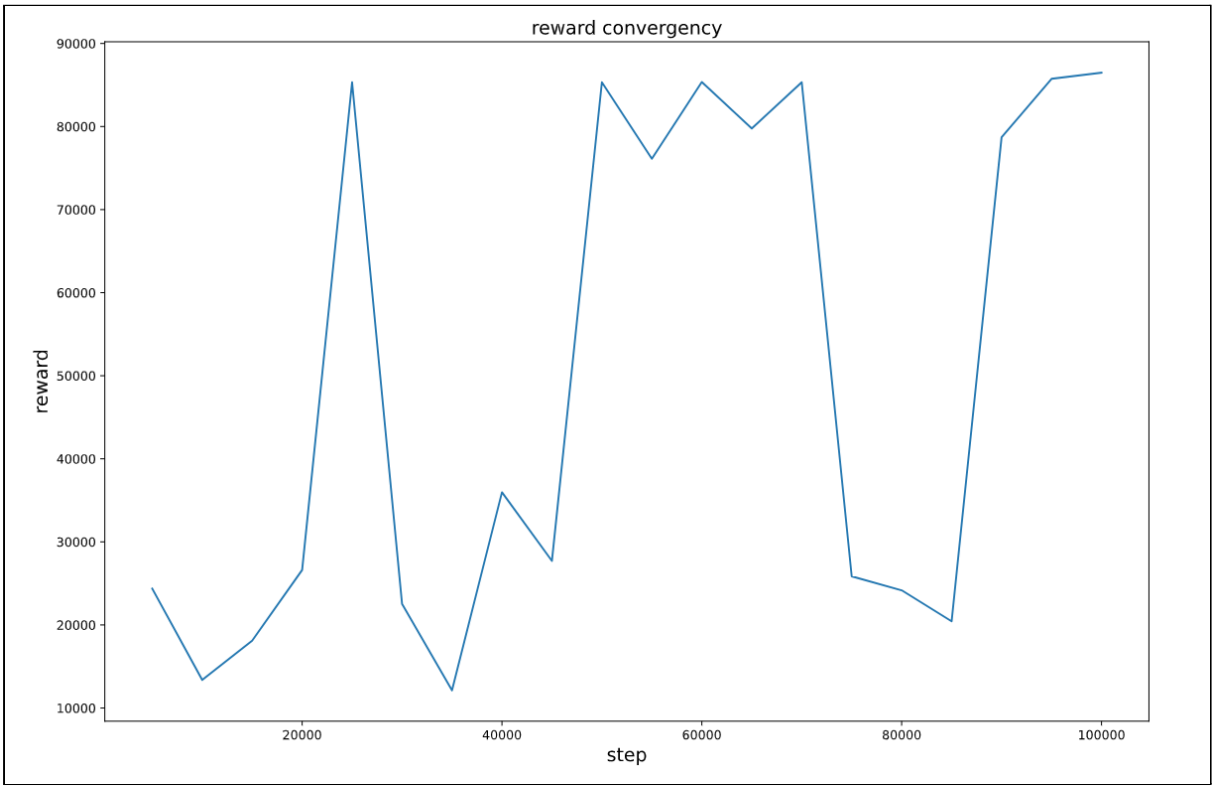

Figure 60 reward convergency

It shows that the reward cannot be converged which means it has not yet found the best and stable strategy. We can see that when it is about 25000 steps, it suddenly goes up and later on, it falls down. Also at the last steps of the whole training, the graph did not show any convergence since there is no flat line and end due to the end of the training.

The training result is also very bad too. The below graph shows after the training, how the agent plays betting in horse racing with training data. X axis represents the games and y axis represents the balance.

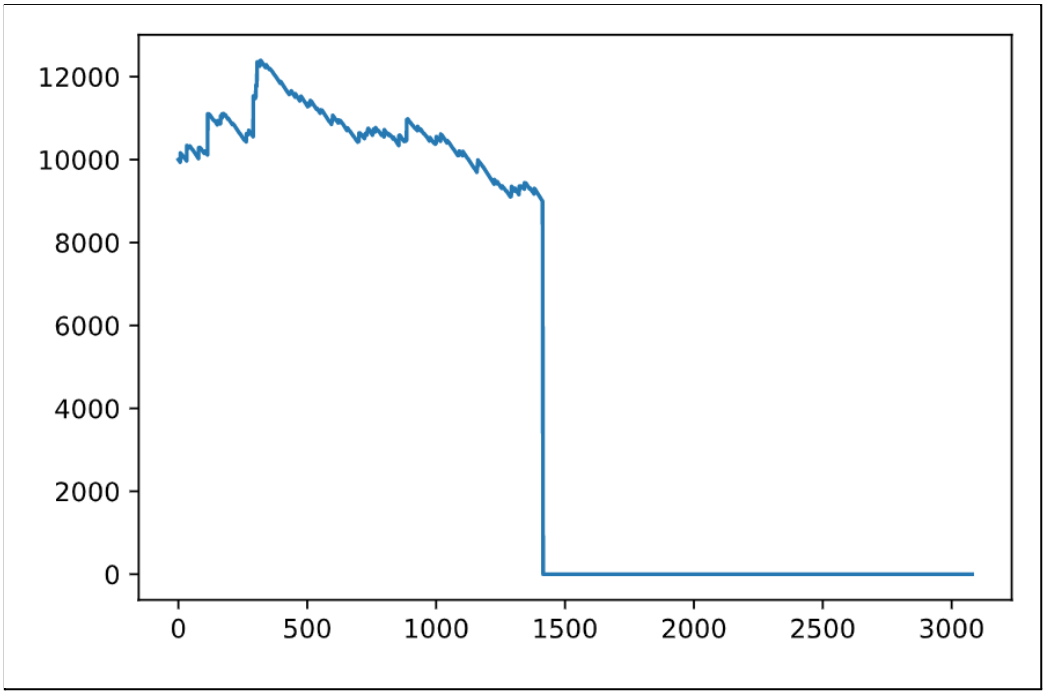

Figure 61 Training Result

It seems that even though the agent bet in horse racing with training data, the training result is very bad. We can see the training stops at around 1400 games since the balance has dropped under 9000. The agent has a great gain at around 500 games and his balance once went to 12000. However, since most of the time the agent is losing, the final result is very bad.

8.3.1.1.3 Analysis

To conclude this failure, the first reason is the training steps are not enough. The convergency shows this. Therefore, for future models, the training steps will be further increased. Also, the agent seems to choose some horse with high win odds since higher the win odds will give him more rewards. This reward function may be changed for a better result since we do not want the agent to only bet on some horses with high odds.

8.3.1.2 Second Model

We have seen how the first model failed, so now we are going to construct the second model which will improve the first model by fixing the problem we met.

8.3.1.2.1 Setting

Since the last model may have the problem of lacking steps for training. This model will increase the training steps a lot. Also, the reward function will be changed to prevent the agent from betting on the horse with high win odds.

The parameter:

- *learning_rate = 0.0007*
- *gamma = 1*
- *total_timesteps = 5000*
- *training_step = 20*

The reward function:

- *R(bet_win) = 1000*
- *R(not_bet) = -10 * (win_odds from the winning horse)*
- *R(Invalid_bet) = -10 * (win_odds from the winning horse)*
- *R(bet_but_lose) = -10*

8.3.1.2.2 Result

The result is shown below:

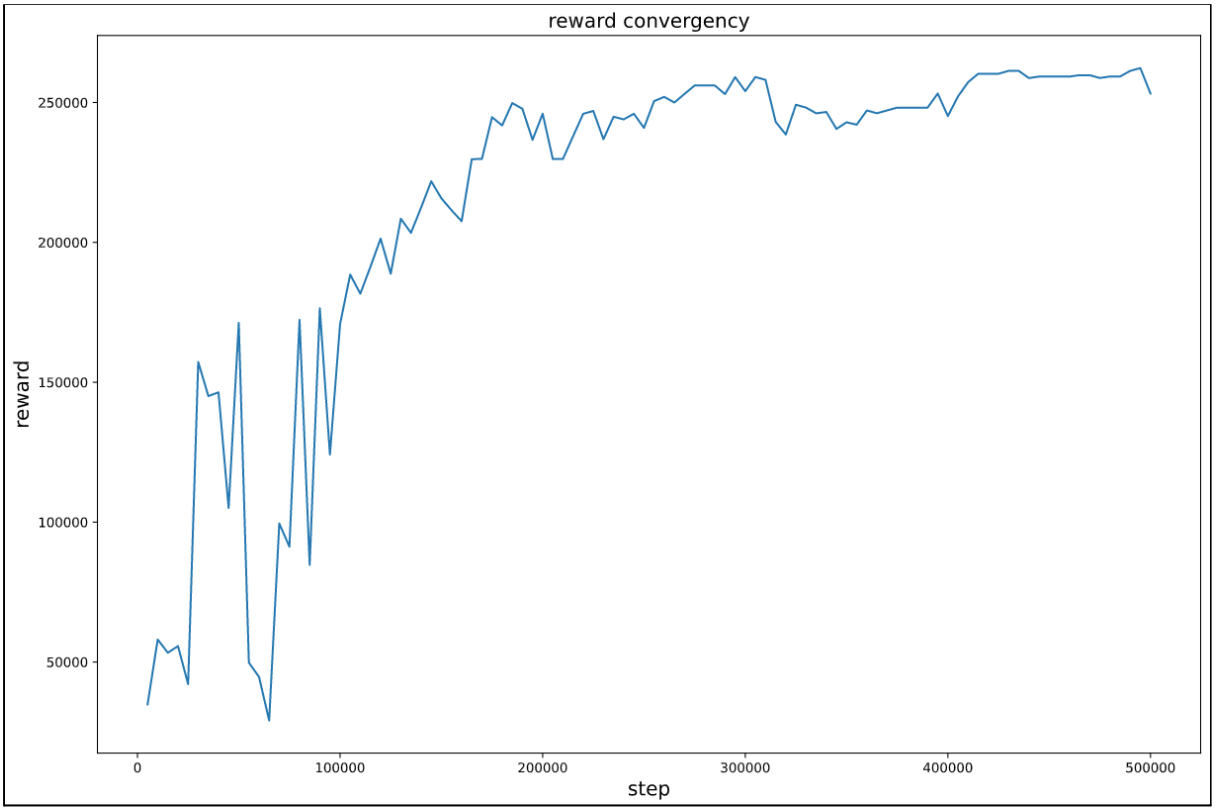

Figure 62 Reward Convergency

The reward is not stable until around 200000 steps. This proves that our thought is right since the training step is not enough. The reward is stable after 200000 steps of training. We can assume that the agent has found a stable strategy. Also, the first 100000 steps shows that the reward cannot converge and this may mean that the agent is trying to develop his own strategy.

The action count is shown below:

Figure 63 Training Set Action Count

The graph shows that there is 1 time choosing 1st horse, 241 times choosing 4th horse, 478 choosing the 9th horse, 30 times choosing the 11th horse, 3 times choosing the 13th horse, 201 times choosing the 14th horse and 2125 times choosing not to bet. There are no invalid actions.

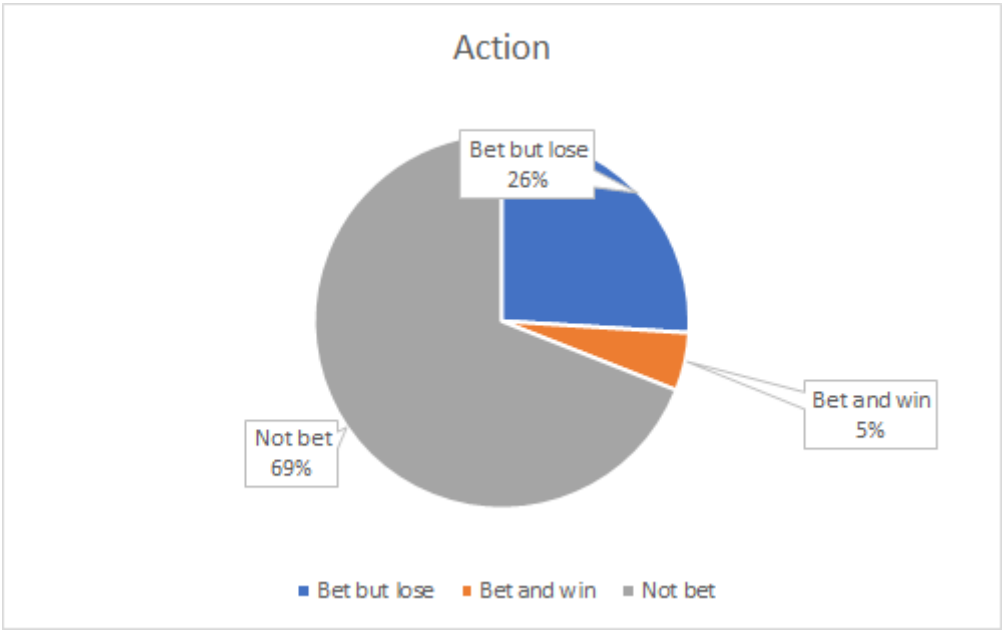

Figure 64 Action

This graph shows the actions' ratio. There are 69% "not bet", 26% "bet but lose" and 5% "bet and win".

The win ratio is shown below:

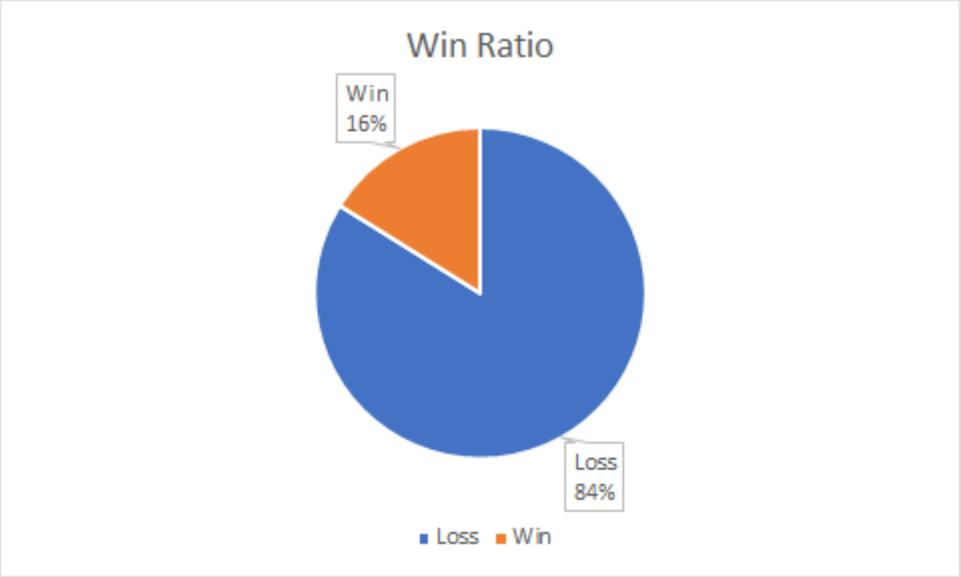

Figure 65 Win ratio

In all betting, there are 16% the agent wins the betting and 84% the agent loses the betting.

The following graph shows the result of the balance list:

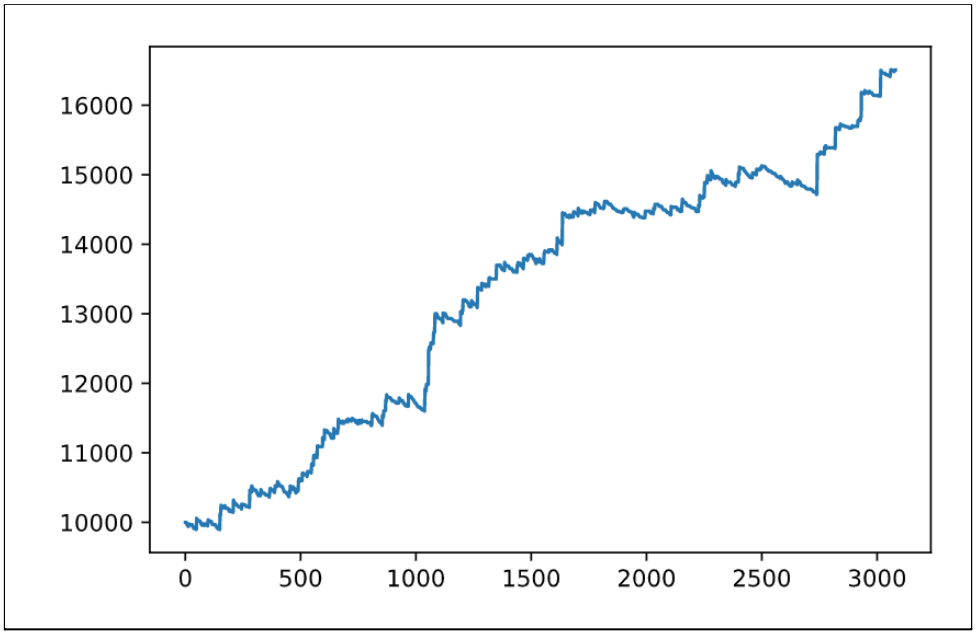

Figure 66 Balance list

The training result is very good since it has a lot of training steps. Also, the reward function can encourage the agent to learn to bet by the feature values of the horse.

8.3.1.1.3 Analysis

To conclude this model, the graph of reward convergency has proven that there is actually lack of training in the first model. We need to have a lot of training steps which can let the agent construct a stable strategy. Moreover, the training shows that the agent seems not to bet most of the time. We will test this model in the coming chapter. However, we still need one more training to prove that this model is not constructed due to luck but the model can actually be trained well in this situation.

8.3.1.3 Third Model

This model is built since we want to prove that the training of the second model is correct and it is not done by luck.

8.3.1.3.1 Setting

The setting of the third model will remain the same since we want to prove the correctness of the second model. Please refer to the second model to see the full setting.

8.3.1.3.2 Result

The result is shown below:

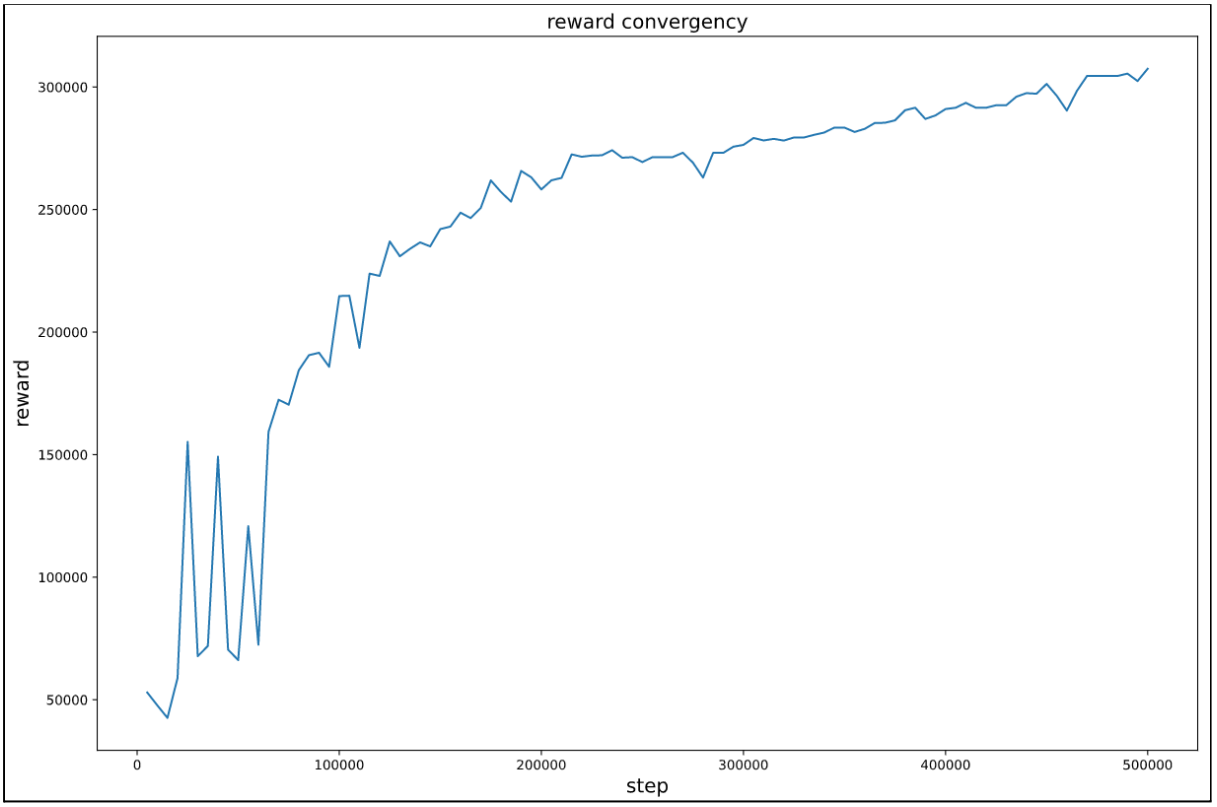

Figure 67 Reward Convergency

From the graph, we can see the reward converges at around 200000 steps. We can say that the agent is able to develop a stable strategy to bet.

8.3.1.2.3 Analysis

To conclude the result of the third model, we can assume that the construction of the second model is correct and the setting will be used. Before testing the second model, we would like to train a model with a lesser penalty of not betting and the setting similar to this model.

8.3.1.4 Fourth Model

This model is used to train the model with a lesser penalty of not betting. We would like the agent to figure out when not to bet to prevent the high risk or low confidence betting.

8.3.1.4.1 Setting

The setting of the third model will remain the same except the reward function since we want the agent to learn when to bet or not to bet.

The parameter:

- *learning_rate = 0.0007*
- *gamma = 1*
- *total_timesteps = 5000*
- *training_step = 20*

The reward function:

- *R(bet_win) = 1000*
- *R(not_bet) = 0*
- *R(Invalid_bet) = -10 * (win_odds from the winning horse)*
- *R(bet_but_lose) = -10*

8.3.1.4.2 Result

The result is shown below:

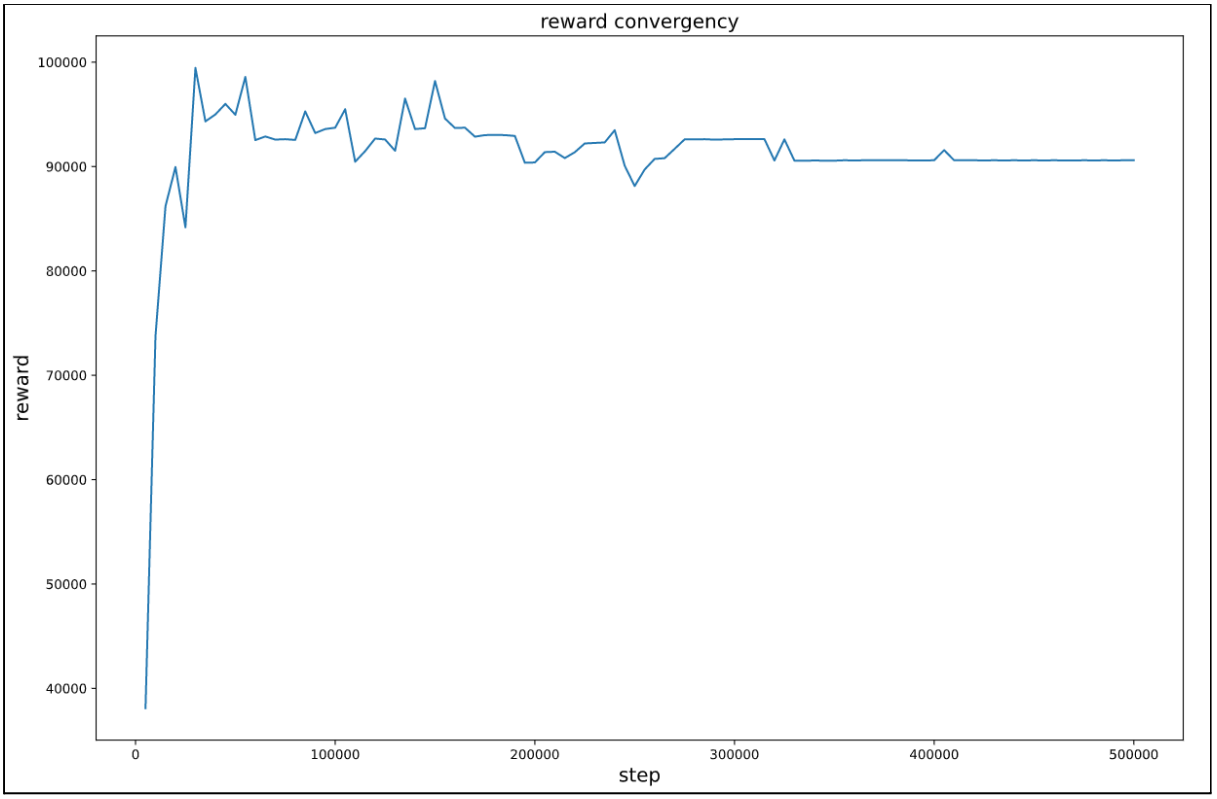

Figure 68 Reward Convergency

The reward converges at around 300000 steps. To compare to the graph of the second model, we can see the final reward is much lower since the agent has not bet many times.

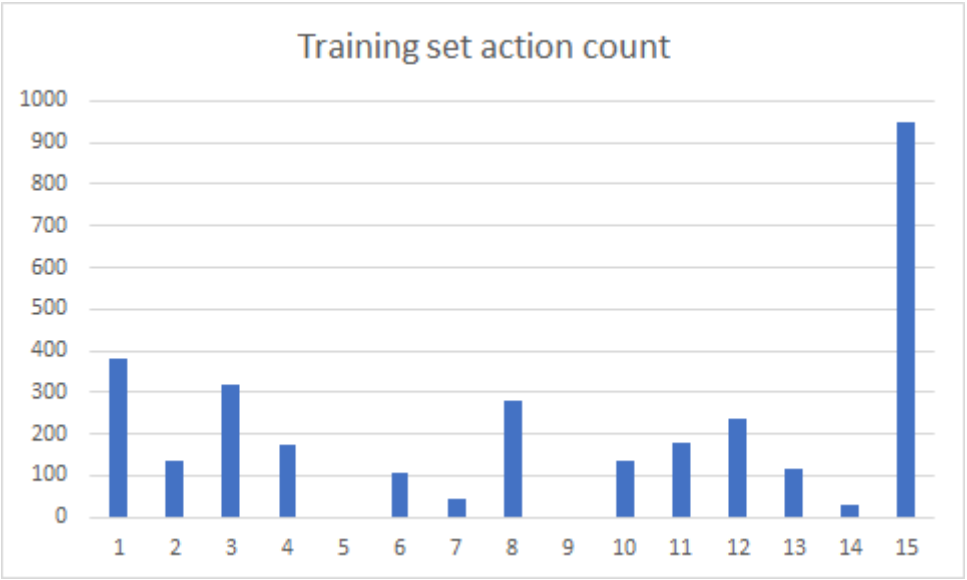

Figure 69 Training set action count

This graph shows the action count of the training set. They are widely distributed in different choices from 1 to 14 except “5” and “9”. There are a total 950 times choosing not to bet.

There are totally 2056 invalid actions and they are all counted as “not to bet”:

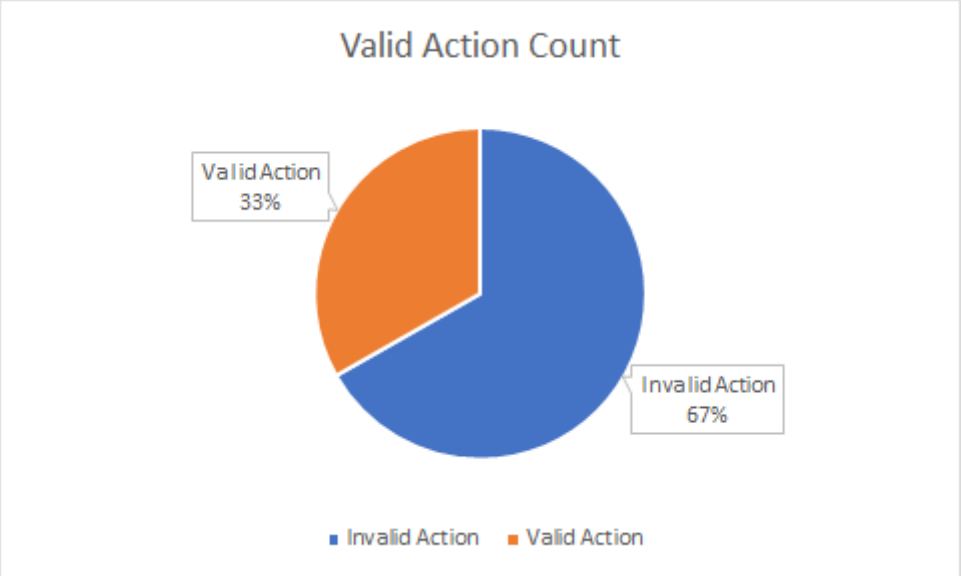

Figure 70 Valid Action Count

The graph of the win ratio:

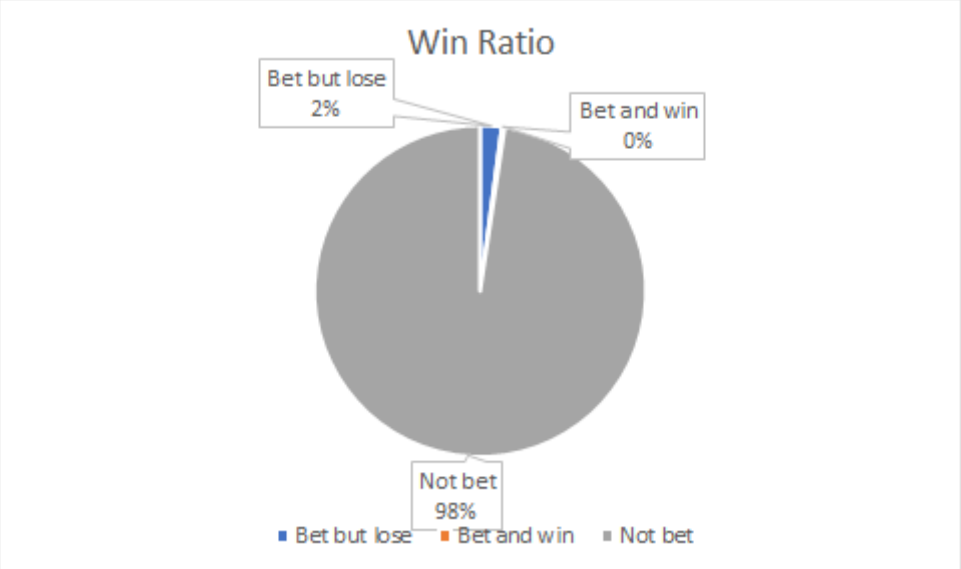

Figure 71 Win Ratio

The 0% in the graph is actually 0.3%. “Bet but lose” is 2%. “Not bet” is 98%.

8.3.1.4.3 Analysis

This result is very bad since the agent cannot see which action is valid or not (2056 invalid actions). Also, the agent does not like to bet and the agent chooses so many invalid actions to make “not to bet”. This is not a good result we want since the agent is not willing to bet when we want him to choose which to bet. The ratio of betting is too low. Even when the agent bet, the agent cannot bet on the winning horse. To conclude, too high a ratio of invalid action and “not to bet” Therefore, we would say that this model is a failure.

8.3.1.5 Conclusion

Since that the second model has been proven to be correct and the other models do not have a good result in the training. We want to test the second model to see whether it runs well in the testing data.

8.3.2 Testing

In this part, we are going to test the model with the testing data which is different from the training data. We would like to see whether there is any overfitting or underfitting.

The model (which is the second model in the training part) will be tested with the testing data which are the racing games in 2019. This model will greatly encourage the agent to choose a winning horse.

8.3.2.1 Actions

The action list is shown below:

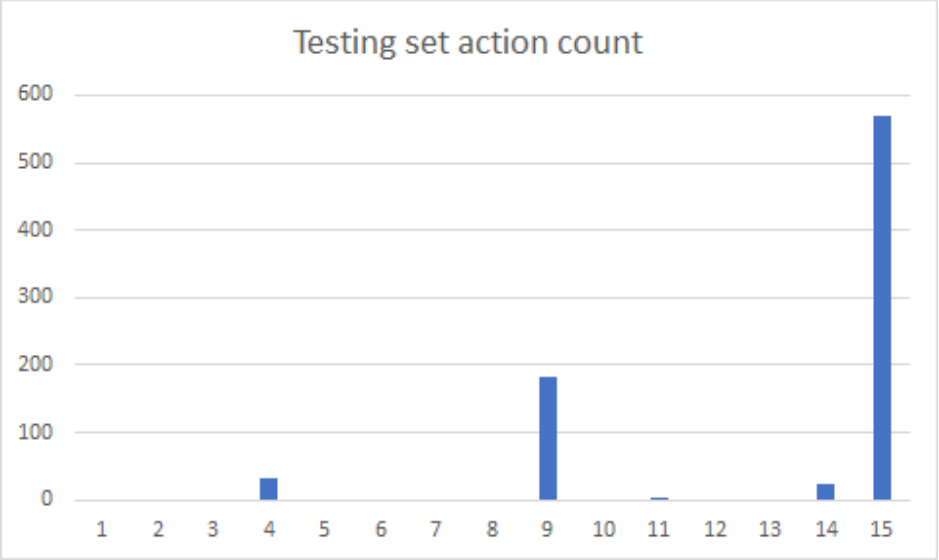

Figure 72 Testing set action count

In this graph, 1 to 14 means that choosing the horse from 1 to 14 and 15 means not to bet. There are 33 times choosing the 4th horse, 182 times choosing the 9th horse, 2 times choosing the 11th horse, 22 times choosing the 14th horse and 571 times choosing not to bet. There is no invalid action (choosing the invalid horse)

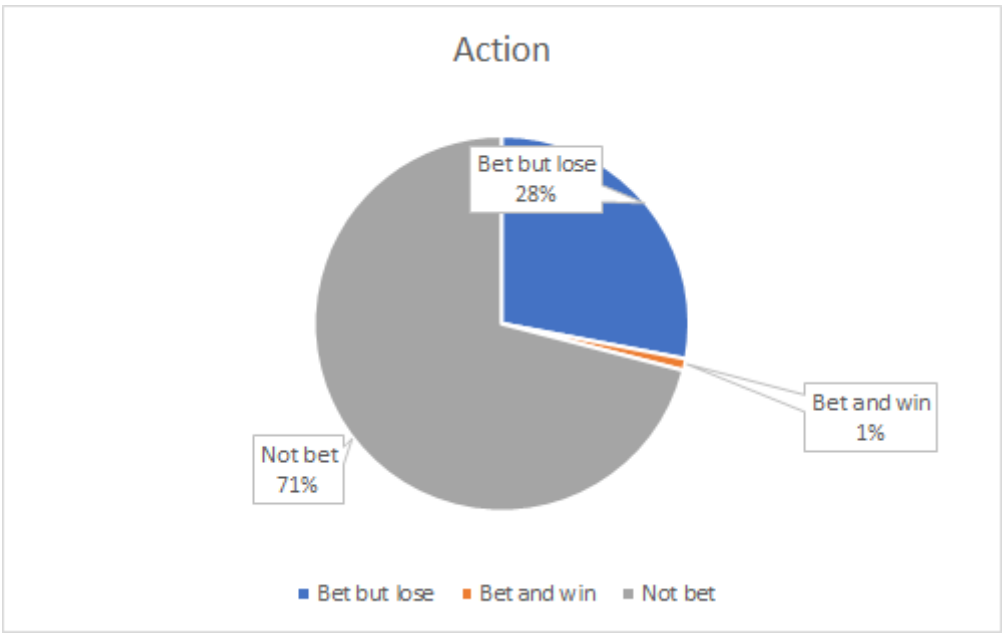

Figure 73 Action

This graph shows the ratio of betting or not. 71% of them did not bet, 1% of them bet and win and 28% of them bet and lose.

8.3.2.2 Win Ratio

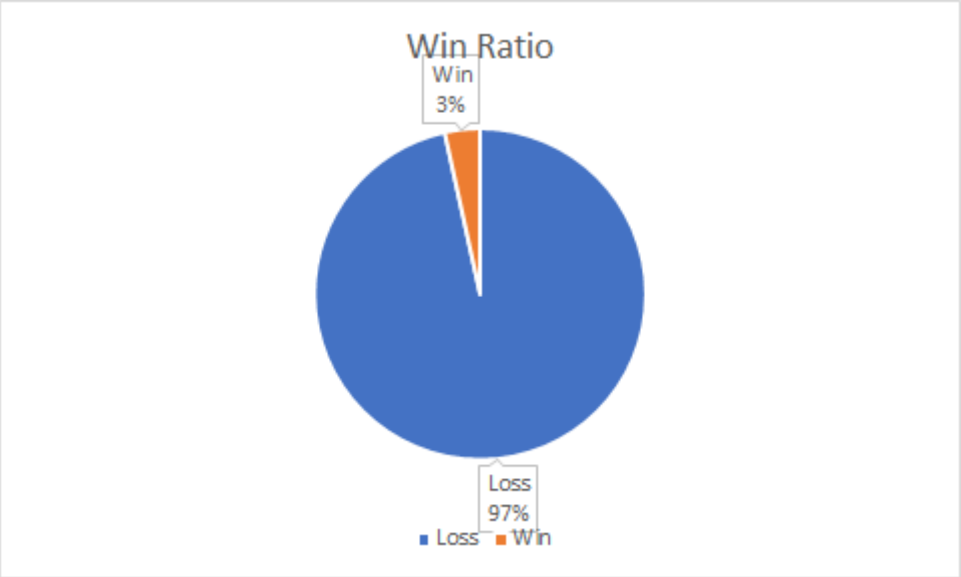

Figure 74 Win Ratio

This graph shows the ratio of betting or not. 97% of all betting is losing and only 3 % of all betting is winning.

8.3.2.2 Cash balance

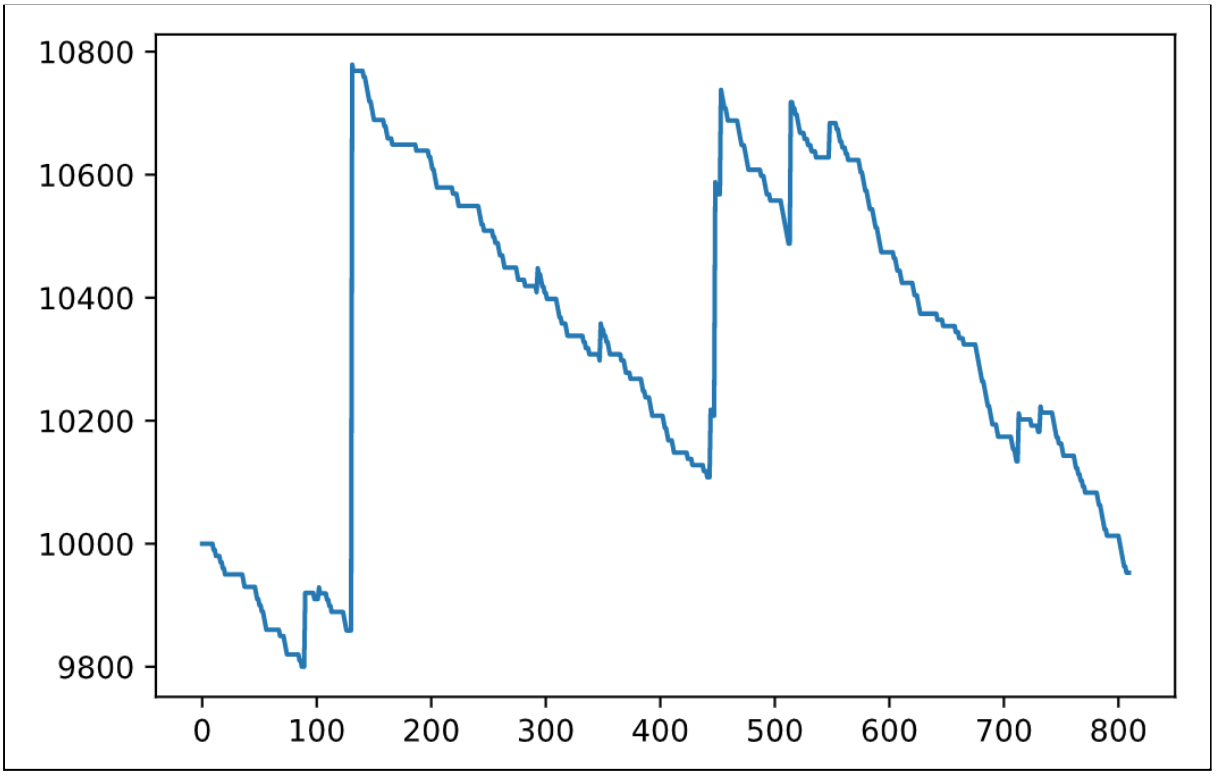

Figure 75 Cash Balance

The initial balance is 10000 and the last balance is 9953. It gets a slight loss at the end. Most of the time the agent is losing. The balance rapidly increased at around the 150th game, 450th game.

8.3.2.2 Analysis

Most of the time, the agent does not want to bet. Even when it bets, most of the time it will bet wrongly or it will bet on some horses with high win odds so we can see that there are some vertical lines in the cash balance. That is why the balance suddenly goes up. Also, the agent's action focuses on only a few horses which are "4", "9", "11" and "14". Maybe the training affects the strategy of the agent.

8.3.3 Conclusion

The model is not performing well as we want the agent to learn how to bet on the winning horse by understanding the feature values of the horses, but in the testing, the agent chose a few horses only and most of the time he did not bet. However, the agent seems to be able to find a stable strategy to bet which is some high win odds horse even though he lost a lot. The agent can learn when to bet and when not to bet. Therefore, although the agent's balance is negative, A2C has been proven to be able to train the agent to bet by having a stable strategy.

In the next part, we are going to use another betting type "Place" to see whether A2C can perform better on training the model.

8.4 Betting on “Place”

In this section, we will start to train the agent to bet on horse racing with betting type “place”. “Place” means choosing the horse which is in the top three.

8.4.1 Training

8.4.1.1 Setting

The first model will use the parameters of the previous model of “win” since it has a fair result.

The parameter:

- *learning_rate = 0.0007*
- *gamma = 1*
- *total_timesteps = 5000*
- *training_step = 20*

The reward function:

- *R(bet_win) = 1000*
- *R(not_bet) = -10 * (win_odds from the winning horse)*
- *R(Invalid_bet) = -10 * (win_odds from the winning horse)*
- *R(bet_but_lose) = -10*

Here, we use the learning rate as 0.0007 to let the agent try the other strategy to find the best strategy in the whole training. Gamma will be 1 since we want there will be no discount factor and the agent can treat every game as independent to the others. The reward function will be used to encourage the agent to choose the top three horses.

8.4.1.2 Result

After the training, the result is shown below:

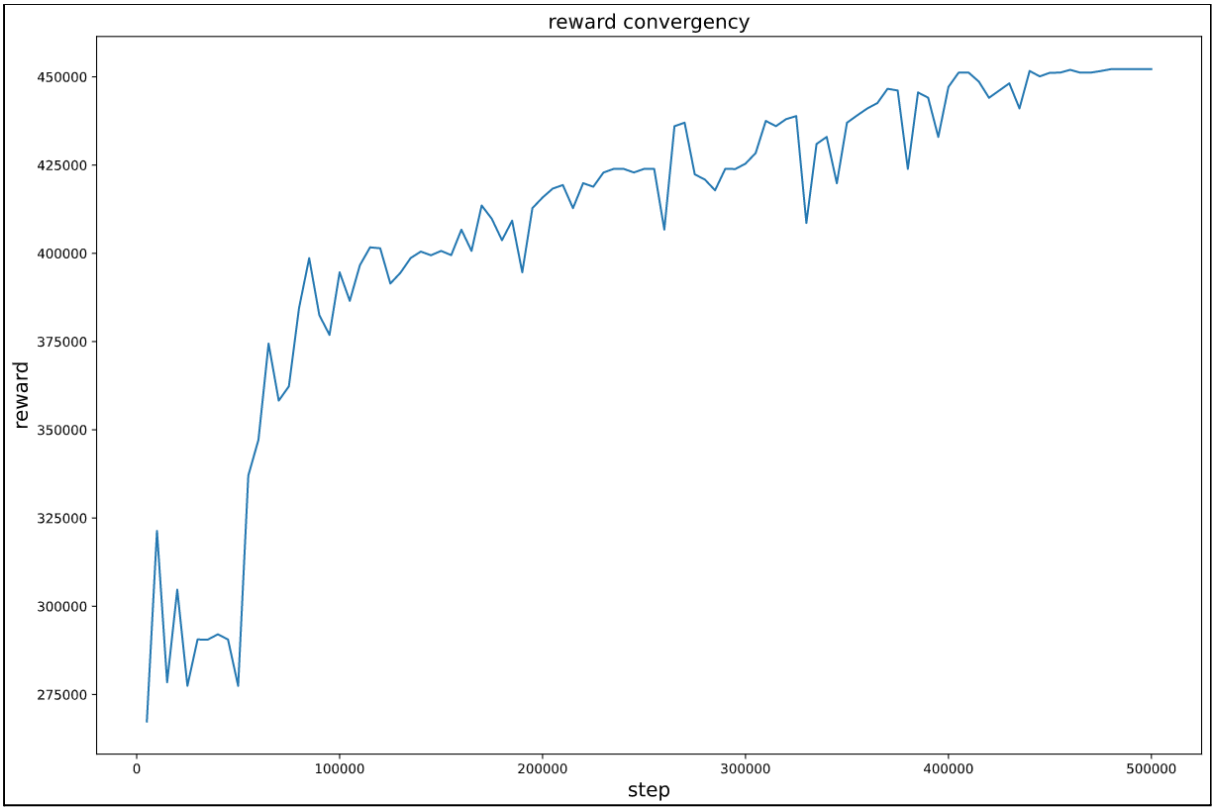

Figure 76 Reward Convergency

The reward converges at around 400000 steps. We can see the agent find a stable strategy at around 400000 steps.

This graph shows the action counts. The actions counts are distributed in different horses. However, the count of not betting is low.

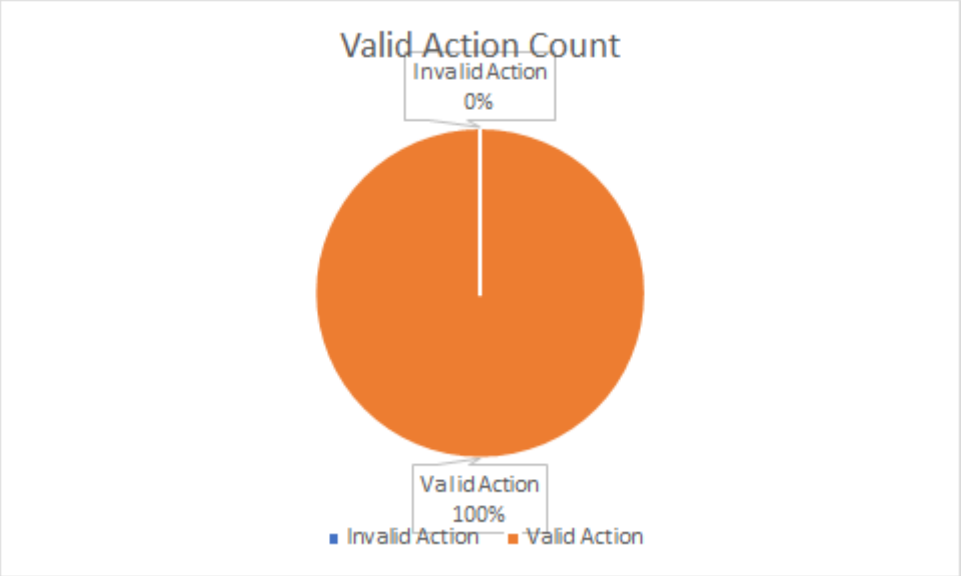

Figure 77 Valid Action Count

There are no invalid actions.

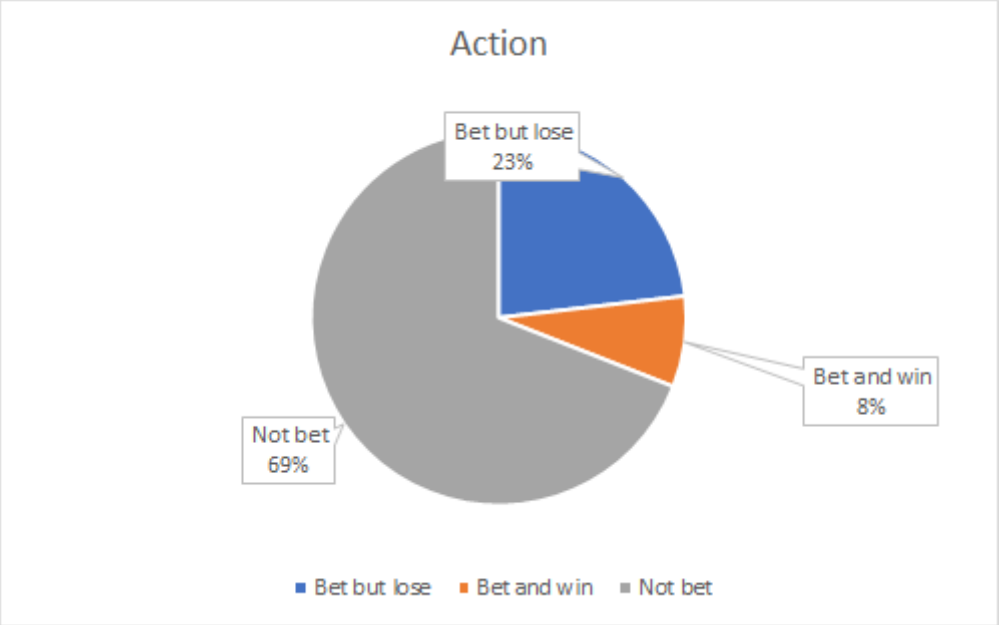

Figure 78 Action

This graph shows the action of bet and win, not to bet and bet but lose. There is 8% bet and win, 23% bet but lose and 69% not bet.

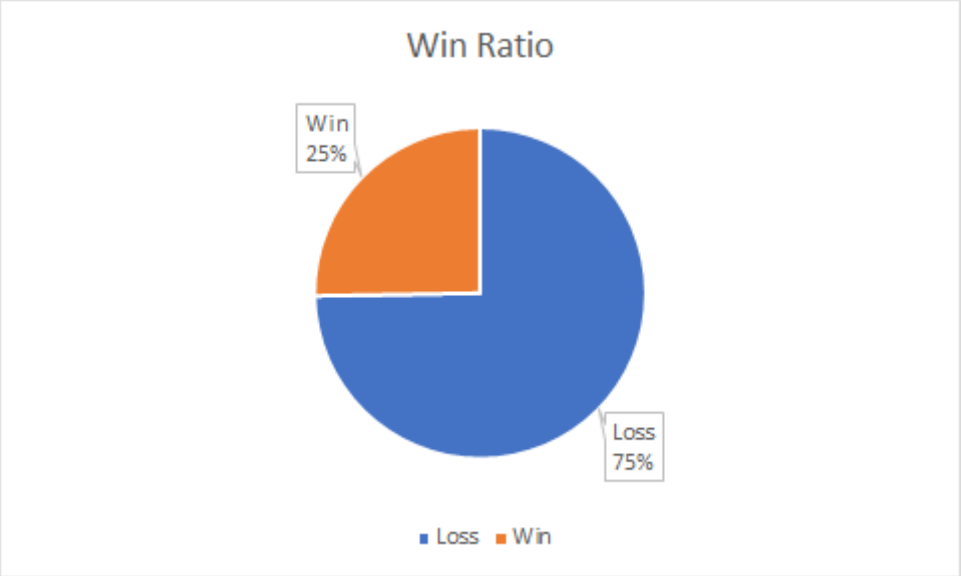

Figure 79 Win Ratio

This graph shows the win ratio in the training. 25% of the betting wins and 75% of the betting loss.

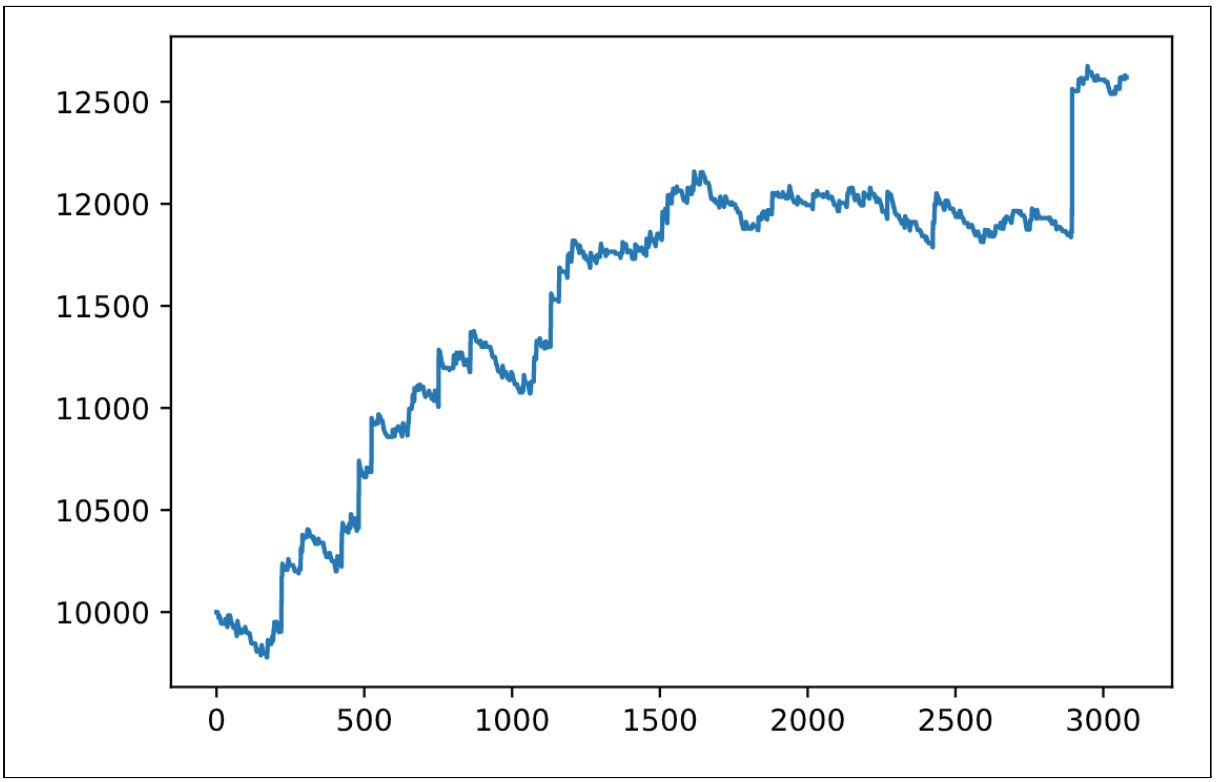

Figure 80 Cash Balance

This graph shows the cash balance of the training result. The initial balance is 10000 and the last balance is 12619. It gains at the end and the gain is larger than the loss so the slope of the graph is increasing.

8.4.1.3 Analysis

This result is very satisfying since the agent is able to find a stable strategy to bet on the top three horses. Also, we accept such a result with no invalid actions. The ratio of betting and not betting is acceptable. The win ratio is around 25% so this is also acceptable since it is not high and overfitting.

8.4.1.4 Conclusion

To conclude the result, we think that this trained model is satisfying and able to be used in the testing. We will test the model later.

8.4.2 Testing

In this part, we are going to test the model with the testing data which is different from the training data. We would like to see whether there is any overfitting or underfitting. Also, we would like to know how the model works well with different real world data.

8.4.2.1 Actions:

The action list is shown below:

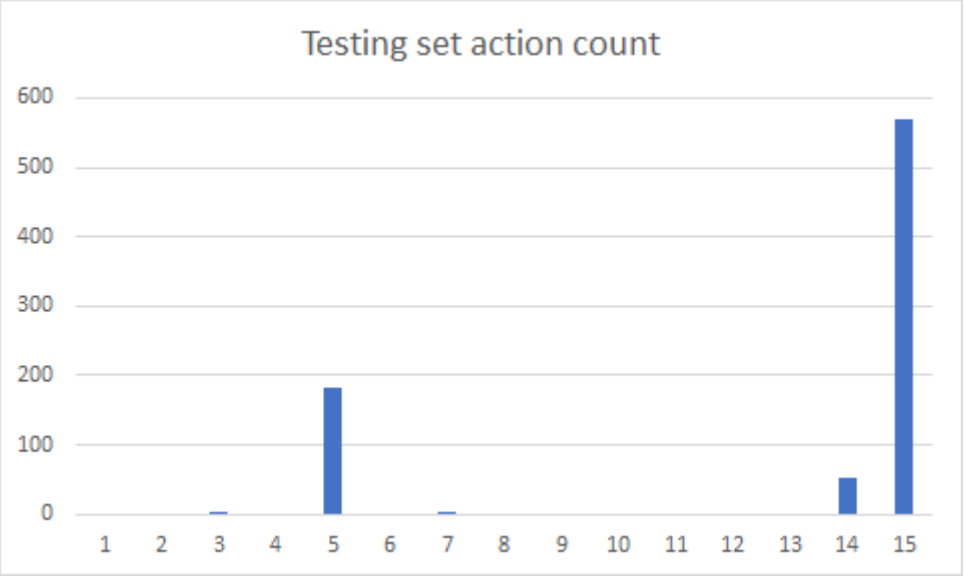

Figure 81 Testing set action count

In this graph, 1 to 14 means that choosing the horse from 1 to 14 and 15 means not to bet. There are 2 times choosing the 3rd horse, 183 times choosing the 5th horse, 3times choosing the 7th horse, 51 times choosing the 14th horse and 570 times not to bet.

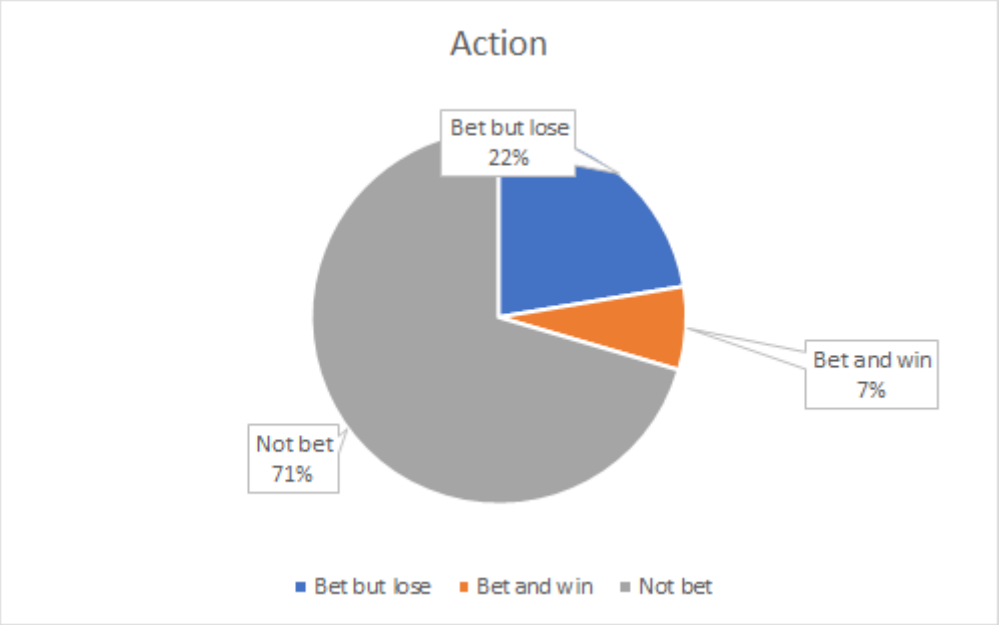

Figure 82 Action

This graph shows the ratio of the action. There are 71% not to bet, 22% bet but lose and 7% bet and win.

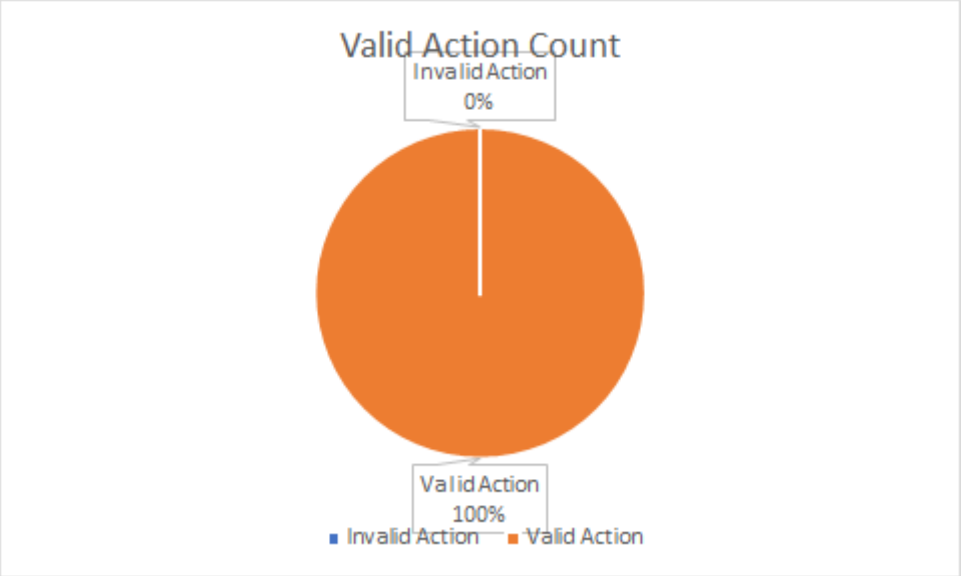

Figure 83 Valid Action Count

There are no invalid actions.

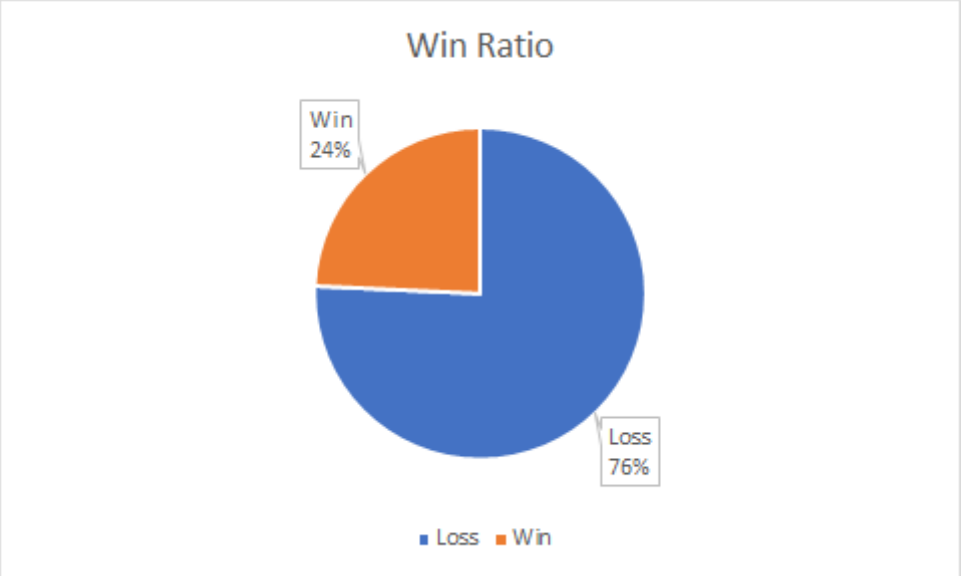

Figure 84 Win Ratio

This graph shows the ratio of betting or not. 76% of all betting is losing and only 24% of all betting is winning.

8.4.3 Conculsion

The model is performing well as the model is able to find a stable strategy to bet on the winning horse in both training and testing. In both training and testing, the model gets around 24% of winning in the betting. Although the ratio of not betting is still high which is around 70% but this is acceptable since the agent does not want to bet on some horses with low confidence or high risk after analysis. This can prove that the agent is able to know when to bet and when not to bet. The final balance of the agent is positive and the slope of the balance list is also increasing. We can say that A2C is able to train the agent to bet in “place”

8.5 Conclusion

To conclude the work of A2C, we can say that the A2C can train a model to bet on horse racing with many feature values. The following graph shows the cash balance on different betting methods with A2C.

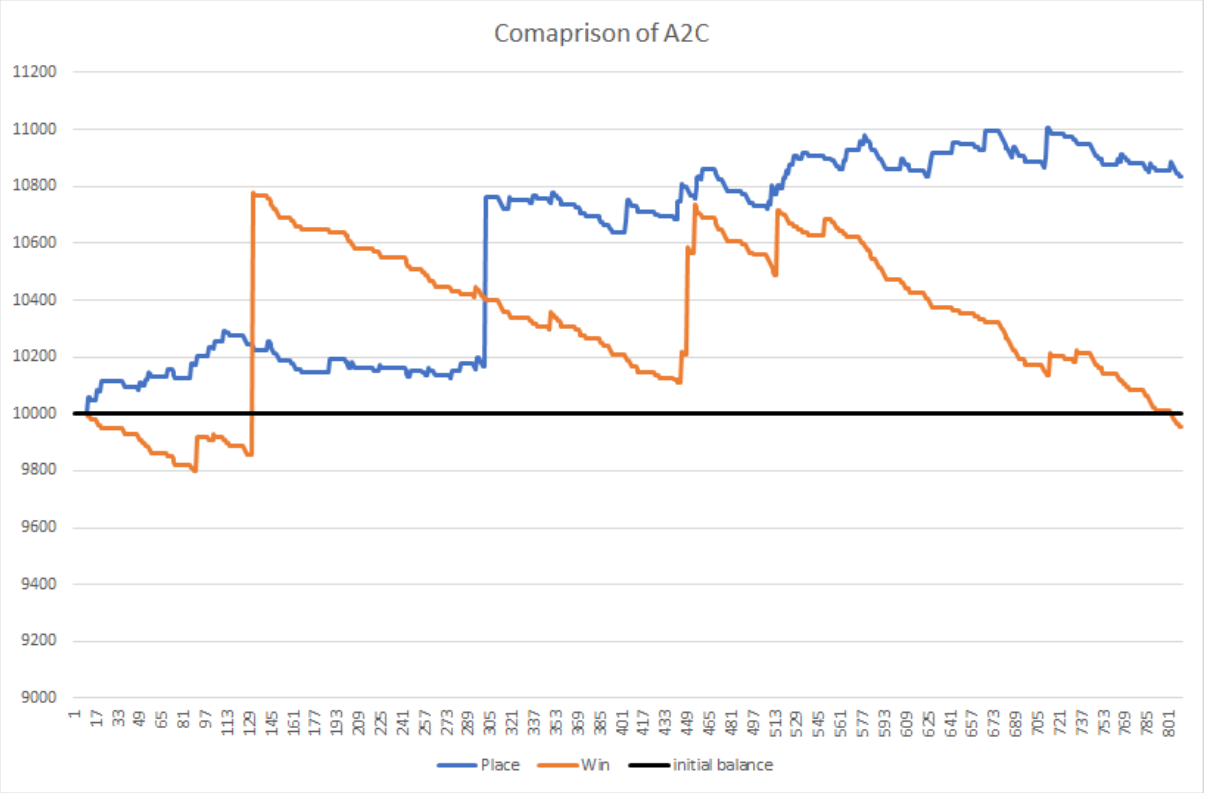

Figure 85 Comparison

We can see that only the betting type “place” can have positive gain >10000. In A2C, it seems that “Win” ‘s result is very bad since it has negative gain. However, actually “Win” still has a positive gain before 800 games. We may say that horse racing still has luck inside so no matter we use such powerful algorithm A2C, there may still be loss. We would like to accept the result and now move on the other algorithm PPO.

Chapter 9 Division of Labour

We have separated our works for division of labour. In the last semester, we share our works from chapter 1 to 5.

Because in the last semester, there have been 6 to 14 horses in a race. My work is to explore more algorithms and more betting types this semester. Also, my work is to explore the way for the agent to learn to recognise the invalid horse.

Therefore, I am in charge of A2C and two different betting types with different numbers of horses in the racing. I mainly work on chapter 8.

Chapter 10 Conclusion

To conclude what we have done in this project, we are successful to build a model with DQN, A2C and PPO. Also, we are able to bet on different betting types like “win” and “place”.

10.1 Conclusion of Work

For DQN and PPO, we have proved that the agent can earn positive gain. We decrease the size of the data with only races with only 12 horses to have a better result. The following graph shows the great and satisfying result.

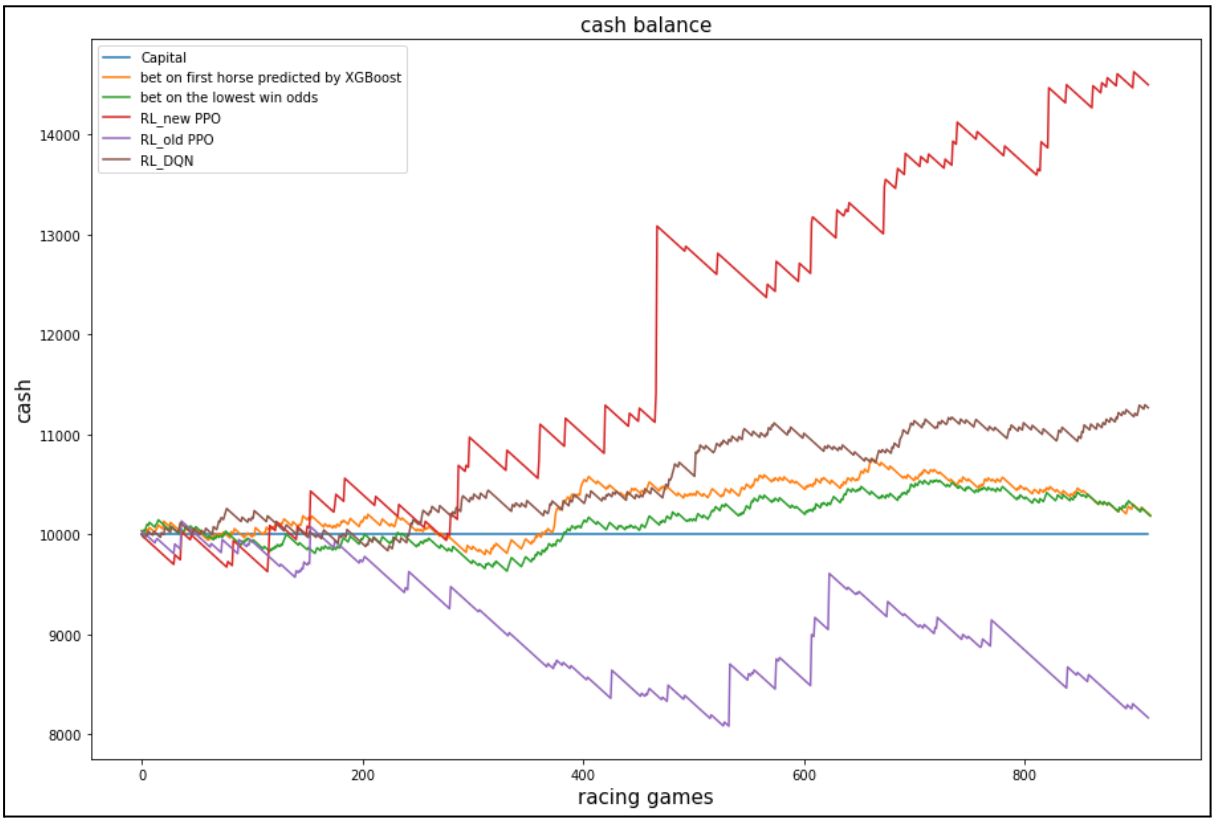

Figure 86 Cash balance

We can see that the new version of DQN and the PPO can have a better result than the previous version of DQN and also the XGBoost.

For A2C with two betting types “win” and “place”, the agent also has a stable strategy and has a positive gain with “place”. Here is the graph of the result:

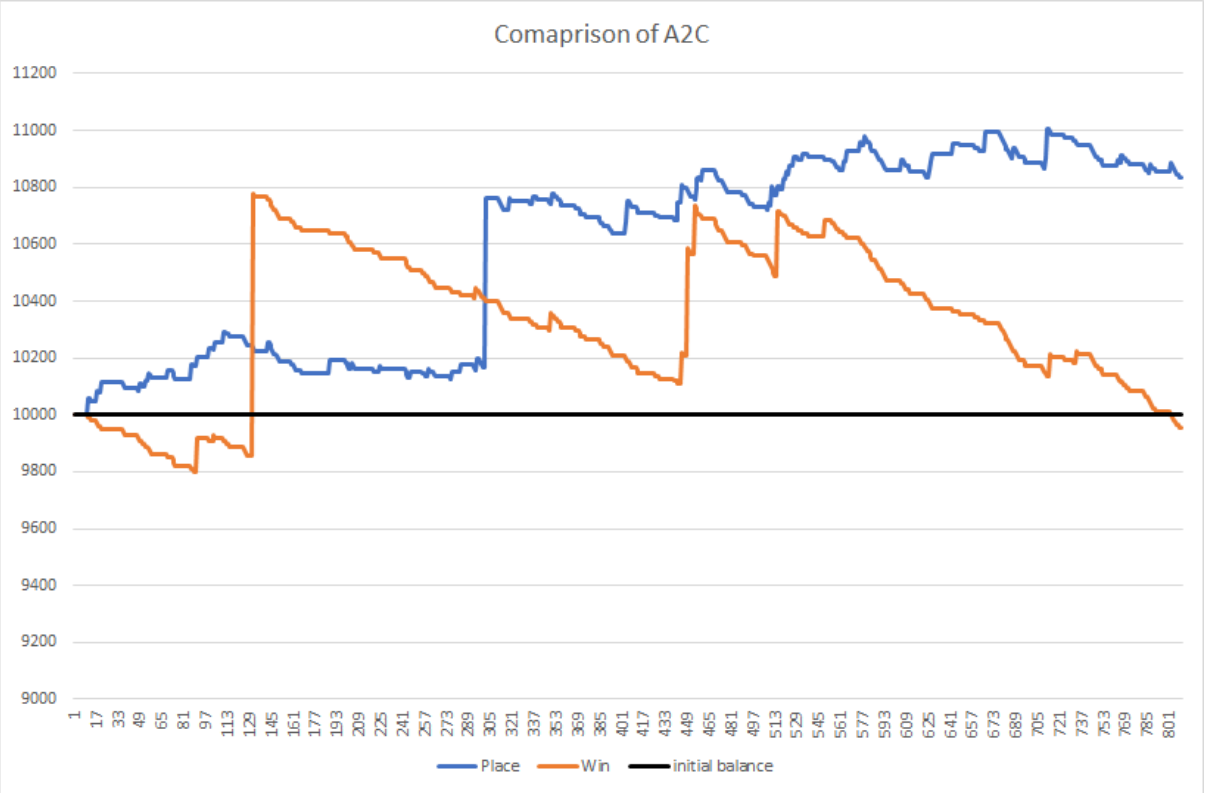

Figure 87 Cash Balance

This graph shows the result of the two A2C models. The “place” one have a positive gain at last while the “win” one does not. Although the “win” one does not have a positive gain. The agent can have a stable strategy and be able to seperate the invalid horses.

10.2 Problem Encountered

There are several problems that we need to deal with and since this kind of application in reinforcement learning is not well developed.

10.2.1 Invalid horse input in Reinforcement Learning

Since the input order of the horses affect how the agent acts. The agent seldom bet on the races with less than 14 horses, as it wants to avoid the ‘invalid bet’. So we need to put more effort on how to deal with the invalid horse problem, maybe from the input format, and focus on a fixed amount of candidate races.

We have tuned the reward function to let the agent learn which horse is an invalid horse. The A2C model proved that the agent can determine which horse is invalid with 0 invalid action count.

Also, we tried to change the horse with only 12 horses to prevent such a case for better training. The result is actually good and the training has become efficient and successful.

10.2.2 Time Consuming

One training requires around 3 hours. With changing the parameter and more training models to compare. The time consuming becomes very large since we have trained a lot of models. The models shown in this report are just some of our trained models. We have already filtered many of them to keep the report clear and tidy.

10.3 Future

This project is just a small step for reinforcement learning applied on horse racing. There are always many ways to improve

10.3.1 More betting types

In the introduction, there are other types like “Quinella” which requires picking multiple horses to be applied to the model. These betting types will be more difficult to train and the accuracy is supposed to be lower but the reward will be much larger compared to “Win” or “Place”. The agent needs to act multiple times to place a bet. The odds’ calculation will also be different.

We have tried to apply it with A2C. However, the result is very bad since we need much more training then “win” and “place”

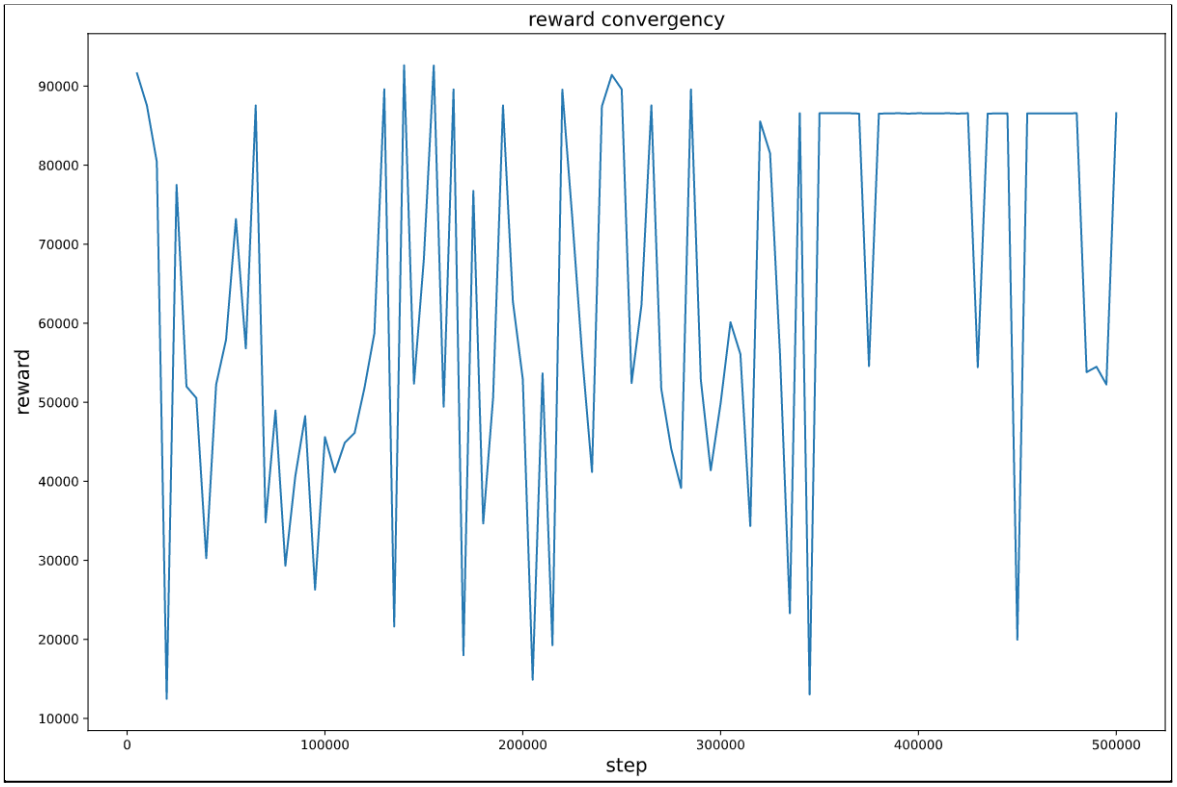

Figure 88 Reward Convergency of A2C “Quinella”

Since the combination of the horse can be 14 * 14 = 196 combinations which are 14 times then “win” and “place”. This requires 14 times of time to train it. One model’s training time is around 3 hours. This means that we need at least 42 hours to complete the training.

10.3.2 More algorithms

Although we have applied DQN, PPO and A2C in our training. There are many other algorithms to use. However, due to the limitation of time, we are not able to study all of them and use all of them.

Reference

[1] Autopilot AI. (n.d.). Retrieved July 22, 2020, from https://www.tesla.com/autopilotAI

[2] AlphaGo: The story so far. (n.d.). Retrieved August 2, 2020, from https://deepmind.com/research/case-studies/alphago-the-story-so-far

[3] Cheng, T., & Lay, M. (2017). *Predicting Horse Racing Result Using TensorFlow* (Rep.).

[4] Yide, L. (2018). *Predicting Horse Racing Result with Machine Learning* (Rep.).

[5] Wong, Y. (2019). *Horse Racing Prediction using Deep Probabilistic Programming with Python and PyTorch* (Rep.).

[6] Wong, K. (2020, November 24). A Brief History of Horse Racing in Hong Kong. Retrieved November 29, 2020, from https://discovery.cathaypacific.com/brief-history-hong-kong-horse-racing/

[7] HKJC. (2020). Performance Highlight 19-20. Retrieved 17 July, 2020, from

https://corporate.hkjc.com/corporate/common/chinese/pdf/report-2019-20/HKJC_AR20_Book_A_Highlights.pdf

[8] HKJC (n.d.). Racing Academy. Retrieved July 29, 2020, from https://entertainment.hkjc.com/entertainment/english/learn-racing/racing-academy.aspx

[9] Ravichandiran, S. (2018). *Hands-on reinforcement learning with Python: Master reinforcement and deep reinforcement learning using OpenAI Gym and TensorFlow*. Birmingham, UK: Packt Publishing.

[10] The Hong Kong Jockey Club. (n.d.). Retrieved July 29, 2020, from https://www.hkjc.com/home/english/index.aspx

[11] Hong Kong Observatory. (n.d.). Retrieved July 29, 2020, from https://www.hko.gov.hk/en/index.html

[12] Zychlinski, S. (2018). [Web log post]. Retrieved from https://towardsdatascience.com/the-search-for-categorical-correlation-a1cf7f1888c9

[13] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794).

[14] Kearns, M. (1988). Thoughts on hypothesis boosting. *Unpublished manuscript*, *45*, 105.

[15] Wade, C. (2020). *Hands-on gradient boosting with XGBoost and scikit-learn: Perform accessible machine learning and extreme gradient boosting with python*. Birmingham: Packt Publishing.

[16] Friedman, J. H., Hastie, T., & Tibshirani, R. J. (1998). *Additive logistic regression: A statistical view of boosting*. Stanford (CA): Stanford University. Division of Biostatistics.

[17] Winder, P. W. P. (2020). *Reinforcement Learning*. Van Duuren Media.

[18] A. Y. Ng, D. Harada, and S. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping”, in ICML, vol. 99, 1999, pp. 278–287.

[19] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529-533.

[20] Huang, S., & Ontañón, S. (2020). A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*.

[21] DeepSenseAI (n.d.). Retrieved September 29, 2020, from https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

[22] Sewak, Mohit (2019) "*Deep Reinforcement Learning: Frontiers of Artificial Intelligence"* Springer Singapore Pte. Limited

[23] Graesser, Laura ; Keng, Wah (2019) "*Foundations of Deep Reinforcement Learning: Theory and Practice in Python"* Addison-Wesley Professional 2019

[24] Dayan, Peter ; Balleine, Bernard W "*Reward, Motivation, and Reinforcement Learning*" Neuron (Cambridge, Mass.), 2002-10, Vol.36 (2), p.285-298

[25] Man Ho, Leung ; (2017) "Applying Modern Reinforcement Learning to Play Video Games" (Rep)

[26] John Schulman ; Filip Wolski ; Prafulla Dhariwal ; Alec Radford ; Oleg Klimov “*Proximal Policy Optimization Algorithms” arXiv:1707.06347 , 2017-Jul*

[27] J. Schulman ; S. Levine ; P. Moritz ; M. I. Jordan ; P. Abbeel. “*Trust region policy optimizatio*n”. In: CoRR, abs/1502.05477 (2015).

[28] Volodymyr Mnih ; Adrià Puigdomènech Badia ; Mehdi Mirza ; Alex Graves ; Timothy P. Lillicrap ; Tim Harley ; David Silver ; Koray Kavukcuoglu “*Asynchronous Methods for Deep Reinforcement Learning” arXiv:1602.01783 [cs.LG]. In: ICML, 2016*

Appendix

There are 3010 races in our testing set :

number of candidates	number of races
14	955
13	193
12	1332
11	256
10	168
9	72
8	51
7	28
6	19
5	6

There are 810 races in our testing set:

number of candidates	number of races
14	240
13	54
12	343

11	73
10	35
9	34
8	14
7	10
6	6
5	1

The action count in training set:

Training set	NEW_PPO	OLD_PPO
bet on horse 1	0	277
bet on horse 2	0	9
bet on horse 3	21	0
bet on horse 4	225	307
bet on horse 5	2	0
bet on horse 6	234	0
bet on horse 7	410	259
bet on horse 8	305	205
bet on horse 9	262	259
bet on horse 10	183	309
bet on horse 11	65	78
bet on horse 12	215	219
Do not bet	0	0

The action count in testing set:

Training set	NEW_PPO	OLD_PPO
bet on horse 1	0	163
bet on horse 2	0	4
bet on horse 3	6	0
bet on horse 4	109	93
bet on horse 5	0	0
bet on horse 6	134	0
bet on horse 7	247	174
bet on horse 8	134	87
bet on horse 9	124	154
bet on horse 10	68	125
bet on horse 11	24	18
bet on horse 12	68	97
Do not bet	0	0