Using and Analysis DT and ANN model to predict winner in LoL game

Introduction:

As e-Sport become more and more popular, the game such like League of Legends (LoL) is also the one of the most popular game in the world. By using machine learning algorithms and enormous database, we can build up a model which can predict which team will win the game base on the performance data in the game. In this project, I will use 2 mainly way (that is Decision Tree and Artificial Neural Network) to training two models by using some of my LoL game database and try to use these two models to predict which team will win according to the same attribute database as the training but different data, then finally compare the predict result and actual result to compute this model's precision.

The Decision Tree is a learning algorithm which is supervised learning. It is a prediction model which represent a mapping relationship between object and object's attribute. In the decision tree, some attribute such as GINI can show us the disorder degree in database. During many decision tree algorithms execute, the model is built up with the rule that every step should try their best to cut down the disorder degree in the rest of the database. In my algorithm, it base on several given data to build up a model which every step uses one condition to separate the data into a few piles, and finally find a "tree-like" way to separate each data into several classes.

The Artificial Neural Network is one of the most popular models in machine learning Even in the whole range of Artificial Intelligence, Artificial Neural Network also is one of the most widely used model. Artificial Neural Network is a model that imitate the human neural network to process information that the database gives us. Artificial Neural Network contain many nodes which represent a special output of the given data (we also can think every node as a function). The nodes are in several layers which also means that every layer have several nodes(functions). In such a structure, Artificial Neural Network can solve many problems.

Algorithms:

Pretreatment: I use list type to store the column name which will need to be use in the same place. The “feature_col” is used to store the attribute column whose data will finally be used as the attribute of training set. The “label_col” only contain “winner” which is the goal that need to be predict in the projection.

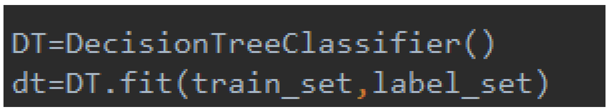
Because there are two big datasets given to us to use. The “new_data.csv” is used to be training set and the “test_set.csv” is used to be testing set. The data in them are stored in “lol” and “lol_test”. The parameter “header” means the row index of the name of the corresponding column.

Decision Tree:

Firstly, I should get the attribute of the training set and the goal of the training set and the goal of the training set. I use the list I build up in the pretreatment to index the data I need.

With the similar way, I get the data of test set and I will use it in the next program.

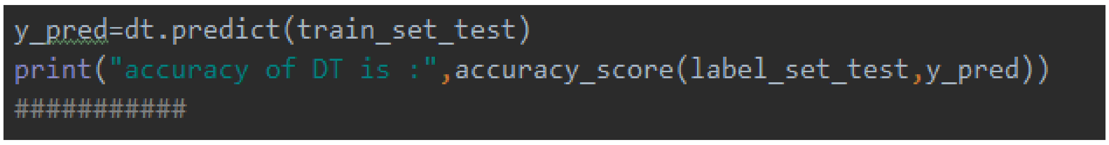
The following steps is to build up the completed Decision Tree model. First, I build up an original Decision Tree model, then using the function “fit()” to train model. After that, the completed Decision Tree model is done. The Decision Tree model will build up a suitable way to separate all attributes into several nodes and use these modes to build up a visual and suitable classification method which can be apply to other dataset with the similar attribute structure.



```
DT=DecisionTreeClassifier()
dt=DT.fit(train_set,label_set)
```

Figure 1

Finally, I use testing set to predict winner and then use “accuracy_score()” to measure accuracy of the model that I just build up.



```
y_pred=dt.predict(train_set_test)
print("accuracy of DT is :",accuracy_score(label_set_test,y_pred))
###########
```

Figure 2

How the Decision Tree work well? By applying greedy approach. In the greedy approach, several measures of node impurity are used such as Gini Index and Entropy. Which is calculated by

$$Gini\ Index = 1 - \sum_{i=0}^{c-1} p_i(t)^2 \text{ and } entropy = -\sum_{i=0}^{c-1} p_i(t) log_2 p_i(t)$$

But, only with the way to measure impurity is not enough, we also need to measure whether the node impurity is lower fast. So, we use "Gain" to measure how the node impurity lower. To measure it, the program will compute node impurity after and before splitting. The "Gain" is the different between them. The program will choose the attribute test condition that produces the highest "Gain". Some example of the program is shown as followed.

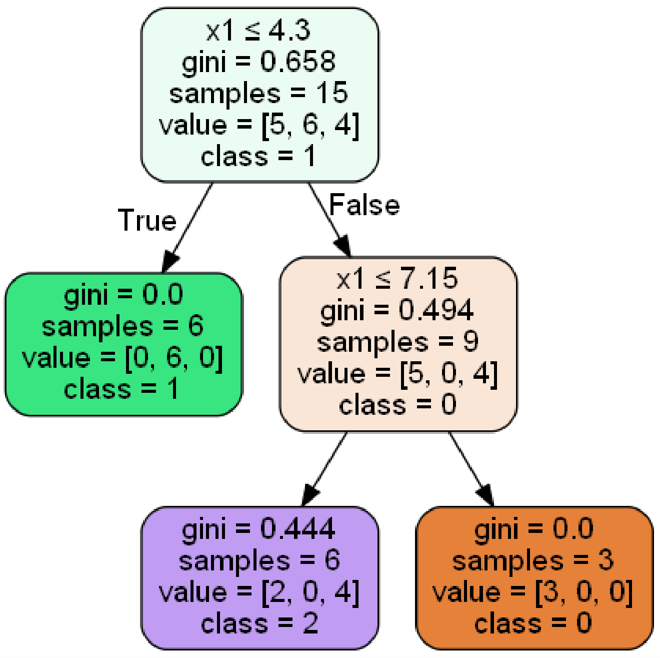
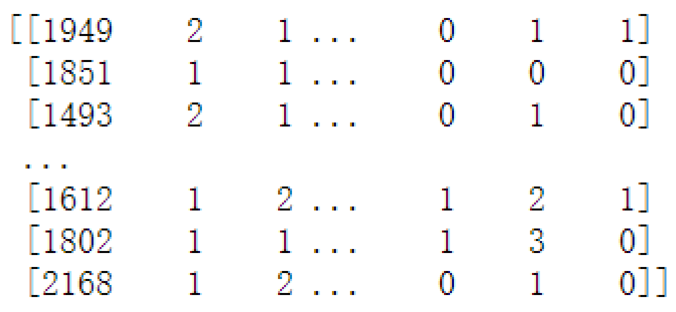


Figure 3

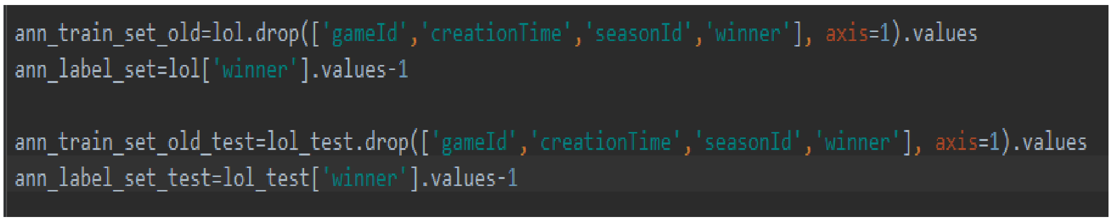
Artificial Neural Network: To get a training set for Artificial Neural Network, we should also select attribute data and label data from original dataset that we get by using pandas. Do not like the way used in building up Decision Tree model, I use "drop()" and ".values" to get the data columns I want. The "drop()" can represent a new dataset that do not have the columns which is mention as parameter in the "drop()". The ".values" is a convenient way to catch the data in the given dataset. Just like the figure8.



```
[[1949    2    1 ...    0    1    1]
 [1851    1    1 ...    0    0    0]
 [1493    2    1 ...    0    1    0]
 ...
 [1612    1    2 ...    1    2    1]
 [1802    1    1 ...    1    3    0]
 [2168    1    2 ...    0    1    0]]
```

Figure 4

Besides, I also have a different pretreatment. Because the element in result of Artificial Neural Network model is start with 0 and the label data in the original dataset is 1 and 2 and when we compare the result and the true label both “0” in the result and “2” in the original dataset will not have match element, it will make a big error when we compute the accuracy of the ANN model. To solve this problem, I let each element in the label column subtract with 1. Then the element will match and the error will reduce.

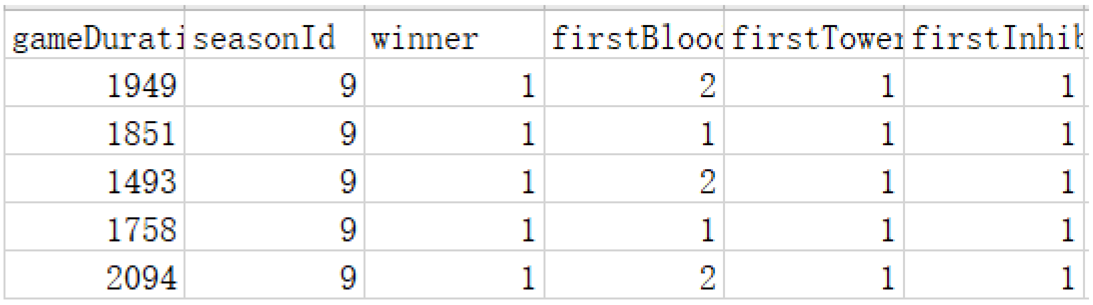


```
ann_train_set_old=lol.drop(['gameId','creationTime','seasonId','winner'], axis=1).values
ann_label_set=lol['winner'].values-1

ann_train_set_old_test=lol_test.drop(['gameId','creationTime','seasonId','winner'], axis=1).values
ann_label_set_test=lol_test['winner'].values-1
```

Figure 5

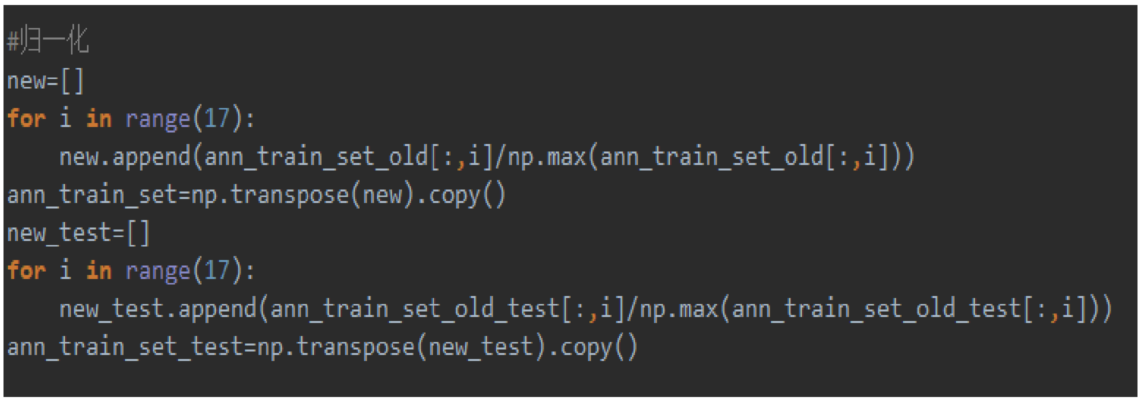
Before we apply these datasets to the Artificial Neural Network model, we need to normalize the attribute in the training set. There are some attributes too large to be train as attribute just as show in the figure 10.



gameDurati	seasonId	winner	firstBloo	firstTowe	firstInhi
1949	9	1	2	1	1
1851	9	1	1	1	1
1493	9	1	2	1	1
1758	9	1	1	1	1
2094	9	1	2	1	1

Figure 6

if we apply these attributes to the ANN model, they will attract the whole model with incomparable strength, which will increase the error and lower other attributes’ strength accordingly. To make every attribute have the same power in the ANN model, we should normalize each attribute to be in the same interval. I try to let each element in one column to be divide by the biggest element in the column. It is not enough if we just do this, because when we try to assign the normalized element to the old dataset, the element will be rounded by the system. I use another way, just make a new empty array, and collect each element that have be normalized. Because the element in the new array is arranged in column and I need it to be arranged in row, I copy the transpose of that new array and rename as “ann_train_set”. Then I will just use it be the final training set which will be apply to the Artificial Neural Network model. For the testing set, the similar way is also should be used.



```
#归一化
new=[]
for i in range(17):
    new.append(ann_train_set_old[:,i]/np.max(ann_train_set_old[:,i]))
ann_train_set=np.transpose(new).copy()
new_test=[]
for i in range(17):
    new_test.append(ann_train_set_old_test[:,i]/np.max(ann_train_set_old_test[:,i]))
ann_train_set_test=np.transpose(new_test).copy()
```

Figure 7

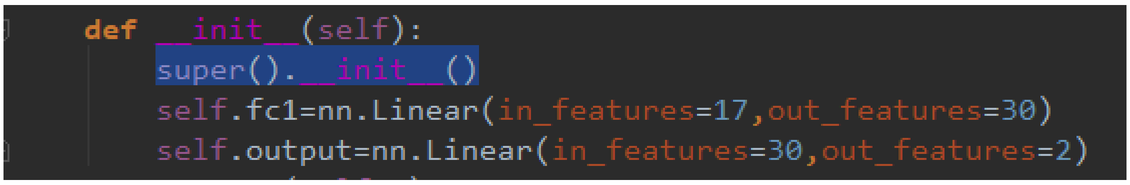
After I do some pretreatment for the data that need to be used in the Artificial Neural Network model, I also transform all of the array into N-dim tensor, which attribute should be float tensor and the label should be long tensor.

Then, I start to build up the Artificial Neural Network model. First I build up a class called “ANN” with the parameter “nn.Module”. The goal of this operation is that I want to inherit from a pre-define module with some other operate to build up my self-define Artificial Neural Network model. Expect inheritance, I must define some parameter in the ANN model.

There are two part in the class. The first one is “def __init__ (self)”, which is to define each layer’s attribute. In this part, I firstly use “super (). __init__ ()” to call constructor in the parent class. Then define the first layer: using “nn.linear” to define it and the parameter in it is “in_features=17” and “ out_features=30”, because there are 17 kinds of attribute in the attribute dataset, the ANN will need 17 input data and the “out_features” is not that important. There will be 2 layers in my ANN model. Then I define the second layer also use “nn.linear ()” this function means this layer will add a linear transformation to the incoming data which is

$$y = xA^T + b$$

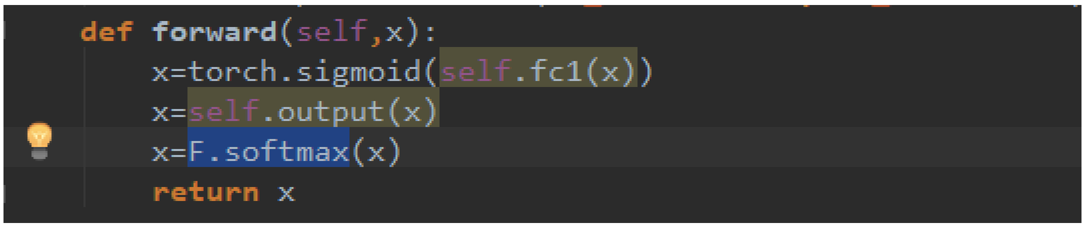
because the last layer I define " out_features=30 ", I need to define the "in_feature" in this layer to be 30, too. What' more, the winner will only appear in one of the two teams, so I just set " out_features=2".



```
def __init__(self):
    super().__init__()
    self.fc1=nn.Linear(in_features=17,out_features=30)
    self.output=nn.Linear(in_features=30,out_features=2)
```

Figure 8

The second part of the class is "def forward (self, x):", which is to define the connection relationship between each layer. The parameter "x" is a formal parameter which is represent a N-dim tensor. The process in this part is as followed. Because we just define the first layer "fc1" in last part so the x will be used as income feature and the whole expression "self. fc1(x)' will represent the output feature of this layer. Between two layers we need an activation function to add nonlinear factors to improve the expressive ability and solve nonlinear classification problem. I select a popular activation function which is sigmoid function. So, the output of the first layer should be the input of the function "torch. sigmoid", this function needs the input data to be the type tensor, that is why we transfer all data into type tensor. After being transform in the sigmoid function the output of it will be used as the incoming feature of the second layer. Besides, the output will need to be transform in the "F.softmax()" which can scale each element into the interval (0,1) and with the sum of them is 1. Finally, return the final answer and the structure of the Artificial Neural Network is done.

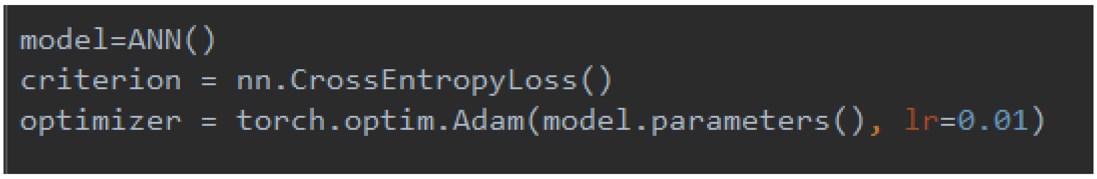


```
def forward(self,x):
    x=torch.sigmoid(self.fc1(x))
    x=self.output(x)
    x=F.softmax(x)
    return x
```

Figure 9

ANN that we just define is just a class so we need to define an object in this class. Then I need to declare two more function before I start iterate.

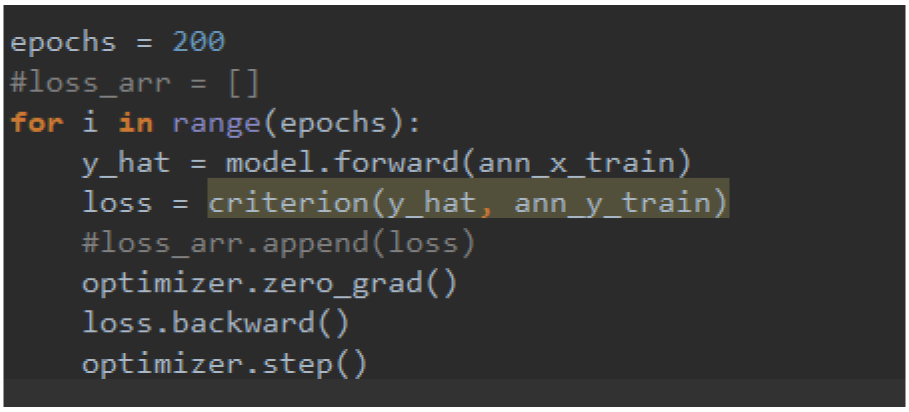
“nn.CrossEntropyLoss()” is a function that can be used to calculate cross entropy loss and “torch. Optim .Adam()” is a powerful function that can optimize the parameter in the ANN model. To use both, I need to declare a criterion and optimizer. I also reset some of the parameter in it, such as reset learning rate to be 0.01.



```
model=ANN()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

Figure 10

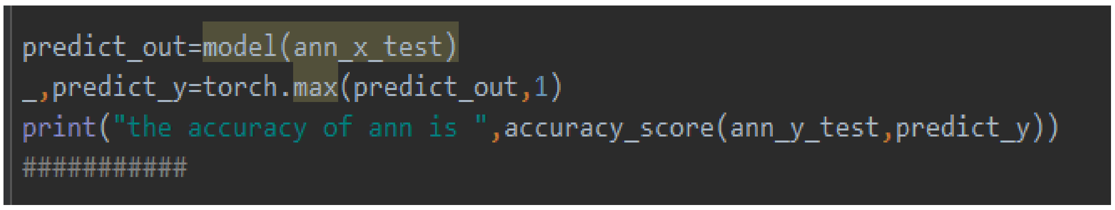
The iteration is as followed. In a for loop and using the function “forward()” to transform the training dataset and compute the loss by using criterion and then optimize the model and iterate it by 200 times.



```
epochs = 200
#loss_arr = []
for i in range(epochs):
    y_hat = model.forward(ann_x_train)
    loss = criterion(y_hat, ann_y_train)
    #loss_arr.append(loss)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Figure 11

My Artificial Neural Network model is done, then I will try to measure its accuracy. First try to predict the output for the testing set. But the output is not the final answer, because it just give us the probability for this example is in one label, so I need to use “torch.max()” to find it, the first parameter is the input dataset and the second parameter is to decide index whether the max in one column or in one row, so we choose 1 to find max element in one row. And we just need what the label is, so we only need to find the index of the max element. Finally, using “accuracy_score()” to calculate the accuracy of the ANN model.



```
predict_out=model(ann_x_test)
_,predict_y=torch.max(predict_out,1)
print("the accuracy of ann is ",accuracy_score(ann_y_test,predict_y))
###########
```

Figure 12

Ensemble: ensemble is a special way to improve classifier's improvement. I design two way to ensemble both Decision Tree and Artificial Neural Network. The first way is mean vote: I add two model's result probability maps to be a new one and using it to vote a new result. The second one is max vote, which I will select the highest predict probability from two model to one example, and this will be the final prediction toward this example.

Requirements:

In this projection, I have used many packages which can help me build up the model conveniently.

Pandas: pandas is a powerful tool to analysis structured data. It base on numpy and have wide application in data mining and data analysis. Pandas provide an efficient way to get data from Comma-Separated Values file and I use this convenient way twice.

Sklearn.tree. DecisionTreeClassifier: sklearn.tree is a subpackage of sklearn. I just import the subpackage of sklearn.tree which called "DecisionTreeClassifier". As for this subpackage, it has the mainly functions that we needed to build up Decision Tree. I use "DecisionTreeClassifier ()" to build up an initial Decision Tree model. Then I use "fit ()" to let the initial model fit the training set, in this way the model will be completed and can be used in the other dataset. To use the completed model to apply on other dataset, I use "predict ()" to meet the projection's need.

Numpy: numpy is widely used in the scientific compute. For us, it also provides a powerful N-dim array object and many math tools. I use "np.max()" to find the max number of the array, there are many other function can achieve this function but this function can be faster and make my model work efficiently. The "np.transpose()" is used to get the transpose of the N-dim array.

Sklearn.metrics: Sklearn.metrics provides many ways to measure some degree. I also use it to measure whether my model do well in the prediction. We can only import the “accuracy_score()” the first parameter of it is the standard dataset and the second one is the compared dataset, by comparing two dataset to calculate the accuracy.

Torch: pytorch is a tensor library that focusing on deep learning and using GPU and CPU to optimize it. I use “torch.FloatTensor()” to transform the original data to float type tensor, use “torch.LongTensor()” to transform the original data to Long type tensor and also use “torch.sigmoid()” to build up an activation function.

Torch.nn: torch.nn is a useful subpackage in the torch which is used to build up several layers in Artificial Neural Network. The torch.nn contain many functions and we can use it to build up different types of layers. I use “nn.linear()” to build up linear layers.

Torch.nn.functional: this package provide several functions, some of it are useful. I use “Torch.nn.functional.softmax()” which can scale the element in the tensor into the interval (0,1) and sum of the elements is 1.

Result:

Classifier	**Accuracy**	**Time waste**
Decision Tree	**0.9611386379092587**	**0.07739260000000003**
Artificial Neurol Network	**0.9596327601282425**	**3.9985997**
Ensemble mean	**0.9632760128242495**	
Ensemble max	**0.9590012629942679**	

Both of two classifiers have done well and gain a very high accuracy in the testing set. And the mean vote ensemble algorithm makes a little improvement in accuracy, but the max vote ensemble makes a lower accuracy, that is because both two model are all have a very huge accuracy, so there will be some edge data will slightly affect the vote in this ensemble.

As for time waste, Decision Tree is using much less time. With higher accuracy and lower time consuming, Decision Tree is more suitable for this project.

Comparison and discussion:

Experience:

In this projection, I have practiced that how to build up a Decision Tree and an

Artificial Neural Network in the program. Especially for the ANN model, it was the first time that I really build up each layer and connection in the network and define all the parameter to let it work well. This project also gives me a better know toward Artificial Neural Network which I used to think it as a complex theorem. And for algorithms, the project gives me more experience than I think, which is how to make pretreatment to my dataset and let it well suit the need of the model. To achieve it, I have learned that I should make some pretreatment to let my program work well.

Improve:

I think this project have many aspects to improve. First is about algorithm's improvement. I think there will be an algorithm which contain ANN and DT and will learn from two models' advantage to build up a better model. Which I think can using DT to be some layer in the ANN and using more DT to produce some small Decision Tree and using ANN to combine each small tree's result to be a more credible answer. Maybe using ANN to extract feature in the original data, then using Decision Tree to train this feature data. This new model may have more compatibility. The second is about the apply extension, I think this project is suitable for the game live platform such as Bilibili to use it to predict winning rate toward two teams and in this way to bring audiences more fun to watch the game. If this plan is for us to solve it, I will prefer to using FLASK framework to build up a web platform and catch real time game data and analyse them in the real time. Thirdly I think I will increase the complexity in the Artificial Neural Network which may including but not limit to increase the number of layers and design some special function and apply it into some layers.

That is the whole content of this report.