## RL problem formation

### state variables

The state variables here consists of:

S&P market data(“open”,”high”, “low”, “close”, “adj close”, “price”), stock market data(“open”,”high”, “low”, “close”, “adj close”, “price”), volume ratio between stock and S&P, 7 day stock volatility, stock returns. The data were standardized and price series were all transformed by a sigmoid function before feeding to the agent.

### training

For each day, the agent observes the data for the previous day. The agent then decides the action for that day, either buy or sell or do nothing. For every consecutive 28 days, the agent clears all positions. If the agent’s action is sell, the instant reward is defined as:

At the end of each trading day, the agent stores instant reward, state variables and actions into a memory list. In the last trading day we enforce a clear position.

Here is an very important point which is the trade-off between exploration and exploitation. The epsilon in agent.py is called the exploration rate, which is set to 1 at first. At the beginning of the training, we do not know anything about the values in Q table. We need to randomly pick an action to act because we don’t know how to select the best action.

With more and more actions are taken and more and more rewards are recorded, we are gathering the knowledge of selecting the best action at each step. With the increase of number of training episodes, we need to decrease the eplison, which means that the agent become more confident at choosing the best action instead of randomly select an action. So the value of epsilon will decrease gradually. self.epsilon \*= self.epsilon\_decay means that we need to have a decay factor to discount the epsilon. Sometimes we need to set a minimum epsilon value to make exploration possible.

### agent

The commission rate is set to 0.97, gamma decay factor is set to 0.95. The action model is specified as a three layer neural network with 64, 32 and 8 units in each layer. Here the agent use Adam as optimizer and “MSE” as loss function.

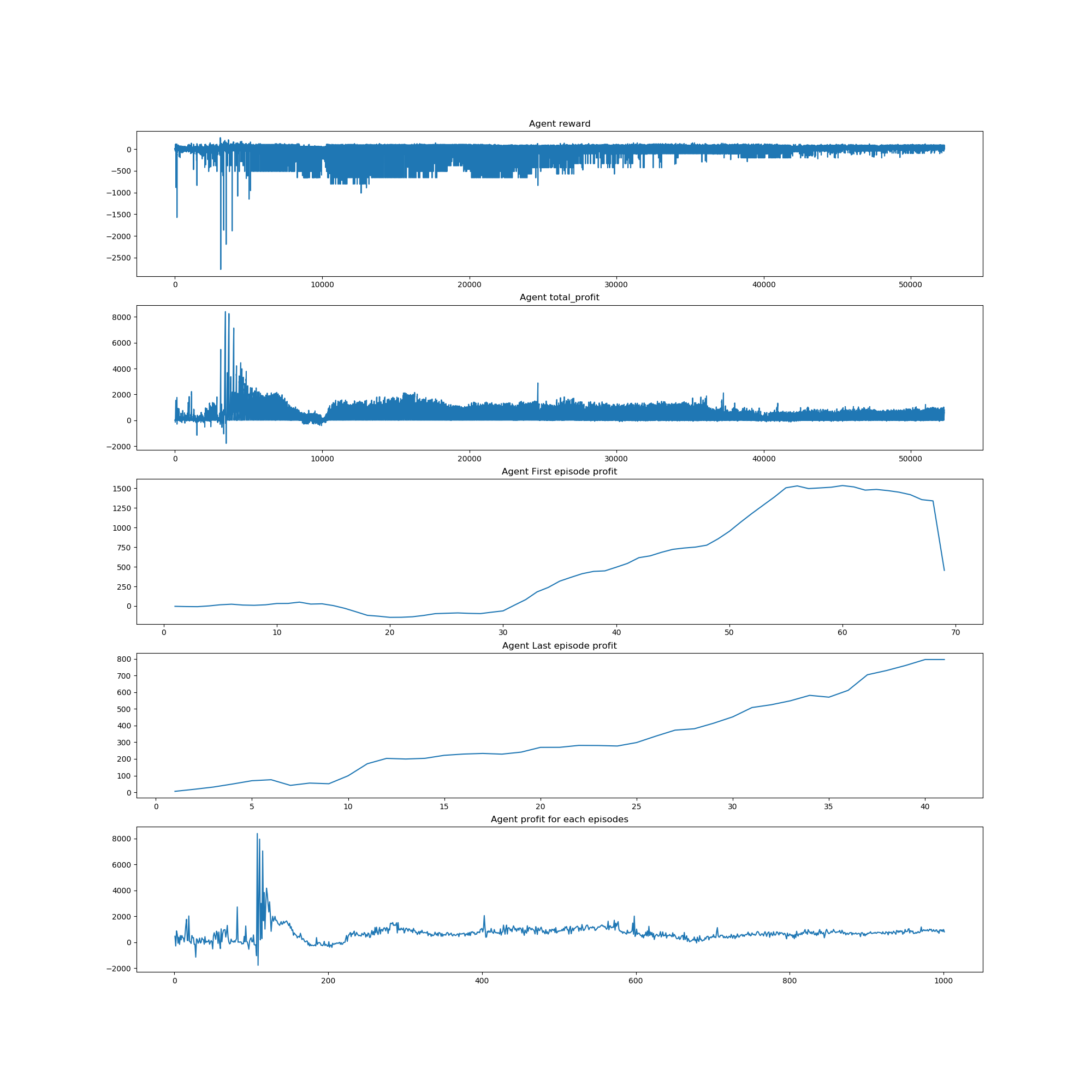
At first iteration the agent randomly decides the action. Then the agent performs “experience replay”:

The agent has a batch size of 32. Looking back into the history memory, the agent choose the latest 32 records of memory, and iterately update the target using each record. The target is specified as:

The sample generated by Q-learning is correlated with each other, if we could randomly choose a certain number of experiences from our memory pool, the probability of autocorrelation will be reduced greatly.

That is, looking back the agent knows what should have been choosen as the best action in each day. Then we construct a new piece of training set to fit for our model. The input is the historical state variables at that time, and the output is a 3 by 1 vector. For the actions that was chosen, the target value is observed and used. For other two actions that didn’t happen, we use model’s prediction to approximate. For each day, we perform such experience replay to adjust our neural network model to learn the environment.

## Experiental Results



We train our model for 1000 episodes on Google using historical data from 2017 to 2018. We can see in the first episode the agent behaves randomly and although it earns some money in the end, it didn’t adjust for the force sell. But in the last episode, the agent has adopt that and the profit curve looks much more stable. We then use Google historical data from 2018 to 2019 as out-of-sample data to test, the agent gains a net profit of $216.6 while the stock price goes from $1006 to $1173.

Here we have no budget constraint, if we fix the total money available we’ll be able to calculate the return of our agents and have stock return as benchmark to tell whether our agent is beating the market or not.