Detailed Project Report

Anshumann

11 July 2021

1 Introduction/Aim

The aim of the project was to use Reinforcement Learning based techniques to predict the optimal action to be taken by a Q-Learning agent in stock dealings for a particular company.

The Developed Model uses Deep Q Network (DQN) technique involving Neural Networks to learn the Optimal Q Function and then take the best possible action amongst the various available actions corresponding to a particular State.

2 Implementation details

1. **The DeepQNetwork Class** : It consists of the function that initialises the Deep Neural Network (using the Pytorch Framework) that is used to evaluate the Q values corresponding to various State-Action pairs. It also consists of the **forward** function that evaluates the Q-values corresponding to the State-Action pairs upon being provided with appropriate inputs.

In Particular the NN used consists of Two Fully Connected Layers (except the input layer) , uses the Adam optimizer and MSE as the Loss Function.

2. **The Agent Class**: It consists of the function that initialises the various hyperparameters as well as the Two Deep Neural Networks,the main Network that Evaluates the Q-values and the Target Q network.

The **storeTransition** function fills up the batch memory as the agent progresses and encounters new States and takes new actions. Note that the batch size =32 denotes the size of this batch memory.

The **chooseAction** function returns the action to be taken by the agent based on the epsilon-greedy strategy.

The **learn** function updates the Model Parameters using the State-Action-Next_State-reward tuples from the batch memory via the Bellman Equation. It uses the Q values obtained from the Target Q Network as the Target Values (as used in the Bellman Equation).

3. **The Main Code**: It initialises certain critical values and Parameters to be used later;
3.1 **num_episodes** : It denotes the number of times the agent trains on the same data-set.

3.2 **state_size** : It denotes the size of the State to be input in the Q Learning Model , i.e it denotes that how many previous days' Adj. Close price is to be used as a state vector.

The rest of the values have been used as defined in the problem statement.

3 Working details

The Agent is initialised within the main function with epsilon value as 1 because the agent initially has no idea about the Q values of different State-Action pairs and hence it explores completely, epsilon then decays at a uniform rate with each episode denoted by eps_decay value till it finally saturates to a eps_min value.

For each time step within an episode,the agent outputs an action based on the epsilon-greedy strategy and then based on the values of tr_count and max_tr_count at that time step takes the appropriate action.The **upd_state** function then updates the state according to the action taken and returns the updated values of all the parameters along with the immediate reward for the State-Action pair.

The State , Action , Next_State , Reward Tuple is then stored in the Batch Memory and the agent.learn() function is invoked to update the Model Weights. The Target Network is updated with the weights of the main Q Network at regular time intervals (governed by the steps_upd_target parameter).

The **upd_state** Function : This function is used to update the values of parameters like curr_stocks , tr_count,total_money etc. and also returns the reward and the optimal action (the best action that should have been taken by the agent) corresponding to the State-Action pairs.

The details of how the parameters are updated were already mentioned in the Problem Statement. To calculate the reward value, the function takes the difference between the values of the net_worth that the agent has after taking the action and before taking it. The net_worth is basically the sum of the money

that the agent has on that day and the money value of the stocks that the agent has on **that** day.