LECTURE 9

In today’s class we focus on deep q learning/network, the underlying principle of a Deep Q Network is very similar to the Q Learning algorithm. It starts with arbitrary Q-value estimates and explores the environment using the ε-greedy policy. And at its core, it uses the same notion of dual actions, a current action with a current Q-value and a target action with a target Q-value, for its update logic to improve its Q-value estimates.

Then we learn from the con-artist story that how exponential rise is too sudden and too fast and we build upon that example to demonstrate how in real life if we have too many states , q learning with q table won’t be an optimal solution and that’s why we have to use neural networks to estimate the q values. We saw an example of open-ai game that had more than thousands of state spaces and they has used neural network to train there model. Also in real world we have continuous state spaces so it is impossible to discretize states and then calculate q value like we were doing in q learning.

Then to understand DQN better we start with understanding what even Neural Network is with the help of digit classification algorithm. In the context of digit classification, the structure of the neural network typically involves an input layer, a hidden layer, and an output layer. The input layer contains neurons that correspond to the features of the digits we want to classify, such as pixel values. The hidden layer contains neurons that process the input and transform it into a representation that is useful for the classification task. The output layer contains neurons that correspond to the different possible categories of digits, such as the numbers 0-9, and the network uses the values of these neurons to make predictions about which category the input digit belongs to.

To make things more clear we learnt how to train our neural network ,Training a neural network typically involves the following steps:

- **Initialization**: Initialize the weights and biases of the neural network to small random values.

- **Forward Propagation:** Feed the input data forward through the network, computing the output at each layer until the final output is produced. This process is also known as the feedforward process.

- **Calculation of Error:** Calculate the error between the predicted output and the actual output, using a chosen loss function. The error is a measure of how far off the predictions are from the true values.

- **Backward Propagation**: Propagate the error backwards through the network using the backpropagation algorithm, which involves computing the gradient of the loss function with respect to the weights and biases at each layer.

- **Update Weights**: Use the gradients computed in step 4 to update the weights and biases at each layer, in a direction that reduces the error.

- **Repeat steps 2-5**: Continue to iterate through steps 2-5, adjusting the weights and biases at each iteration, until the error on the training data is minimized to an acceptable level or until some stopping criterion is met.

By repeating these steps, the neural network gradually learns to map inputs to outputs, improving its accuracy over time.

Now that we have a brief idea about basics, we actually start with the Algorithm.
The DQN (Deep Q-Network) architecture consists of two neural networks: the Q network and the Target network, as well as a component called Experience Replay. The Q network is responsible for learning and producing the optimal state-action value, while the Target network is used to estimate the target values for updating the Q network. Experience Replay is the component that interacts with the environment, generating data that is used to train the Q network by storing and replaying

experiences from the past. Together, these components allow the DQN to learn from past experiences and improve its performance over time.

Now the question is why do we need experience relay or targeted network?

The use of Experience Replay in DQN is necessary because training with single samples can result in too much variance, and training with sequential actions can lead to catastrophic forgetting. Experience Replay stores past actions and observations in a memory and randomly selects a batch for training to ensure diversity.

The Target Network in DQN is a second neural network that is used to ensure the stability of the Target Q values. The Q Network is trained to predict the Predicted Q values, but its weights are updated at each time step, causing the Target Q values to fluctuate. The Target Network solves this issue by not getting trained and remaining stable for a short period, after which its weights are copied over from the Q Network. This results in more stable training.

DQN IN DEPTH

- **Initialization-** Initialize the Q Network with random weights and copy them to the Target Network.
- **Execution-** The Experience Replay begins the training data production phase with the first time step and uses the Q Network to choose a -greedy action. While interacting with the environment to produce a training sample, the Q Network assumes the role of the agent. During this phase, no DQN training is conducted. All possible actions that can be made based on the present condition have their Q-values predicted by the Q Network. These Q-values are used to choose a -greedy action. Experience Replay executes the ε-greedy action and receives the next state and reward.
- **Storing**- It stores the results in the replay data. Each such result is a sample observation which will later be used as training data.

- **Random Selection-** We now start the phase to train the DQN. Select a training batch of random samples from the replay data as input for both networks.
- **Training-** The Q network predicts Q values for all actions that can be taken from the state. From the output Q values, select the one for the sample action. This is the Predicted Q Value.

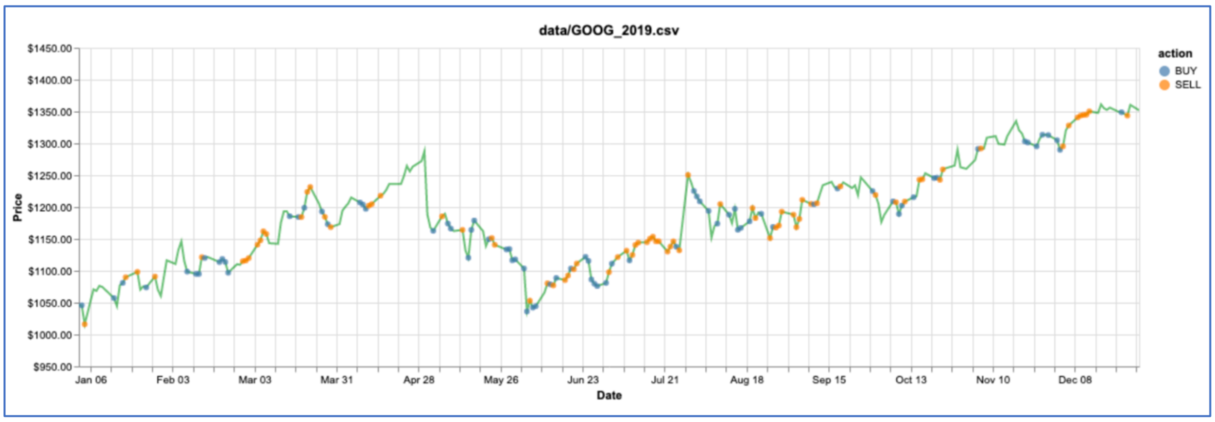
  The Target network receives the sample's subsequent state as input. The Target network determines the highest of these Q values for each action that can be taken from the upcoming state.Predict the Q values for all actions using the upcoming state as the input. The maximum of the Q-values is chosen by the target network.The Target Network's output and the sample's reward make up the Target Q Value.

- **Loss Computation-** Compute the Mean Squared Error loss using the difference between the Target Q Value and the Predicted Q Value.
- **Updating-** Back-propagate the loss and update the weights of the Q Network using gradient descent. The Target network is not trained and remains fixed, so no Loss is computed, and back-propagation is not done. This completes the processing for this time-step. The Target network is not trained so no Loss is computed, and back-propagation is not done.

During the following time-step, the procedure is repeated. The Target network's weights have not been altered, however the Q network's have. This prevents us from chasing a moving goal by allowing the target Q values to stay stationary for a while while the Q network learns to forecast Q values that are more accurately. Copy the Q network weights to the Target network after T time steps. As a result, the Target network can obtain the enhanced weights and forecast Q values with greater accuracy. Processing carries on as usual.The Target network and the Q network weights are once more equal.

SIMULATIONS

For the simulation part we saw a stock trading bot trained on google stocks for the period 2010-2017 and then we saw its profit in the year 2018 and 2019 and the most shocking thing was even though google had lost 11% of its shares in 2018 but the algorithm was able to make $800 profits.



The shortcoming of this algorithm was that it couldn't perform for the years where there was no pattern , like the covid year and it lost $50 .