IEOR 4575 Project Report

Integer Programming : Learning to Cut

-Arunit Maity (am5689)

Introduction

In this project I aimed to solve the Integer Programming problem by learning to choose optimal gomory cuts (cutting plane problem) using reinforcement learning. Different hyperparameters and models were tested and the most optimum model was trained and its results have been showcased. This project was completed as part of the Reinforcement Learning course (IEOR 4575) at Columbia University.

Algorithm Design

The premise of the problem is as follows - constraints and cuts are fed into the policy network. A probability distribution is obtained for the current cuts and an action (sampling a cut) is made based on it. The steps involved in the solution are -

Min-Max Scaling

We have constraints and available cuts for each state. After we normalize the matrices using a min-max scaler, we feed the concatenated vectors into the policy network.

Policy Gradient

Policy gradient methods are reinforcement learning approaches that use gradient descent to optimize parametrized policies in terms of expected return (long-term cumulative reward). After every trajectory, the discounted rewards are calculated and then using policy gradients, we look to update the parameters within the network.

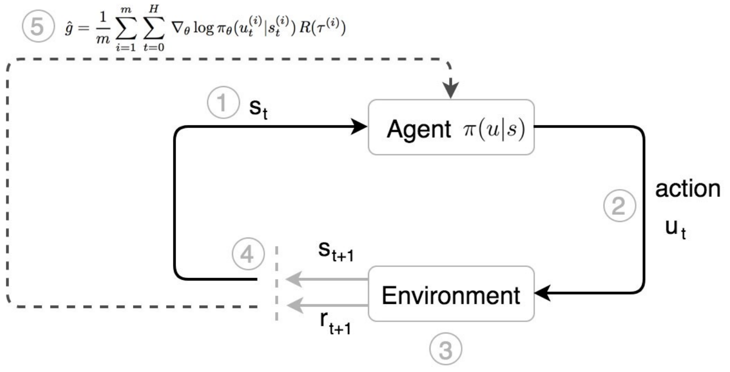

5
$\hat{g} = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\,R(\tau^{(i)})$
1
s_t
Agent $\pi(u|s)$
2
action
u_t
4
s_{t+1}
Environment
r_{t+1}
3

Fig 1 : Typical Policy Gradient RL Framework

Neural Networks

I am using an LSTM based artificial neural network with attention for the purpose of Q-Value prediction. The LSTM network has 1 layer and creates an embedding (vector of size 128 for easy configuration and size 256 for hard configuration). Using this architecture makes sense as we need to account for varying dimensions (Fig 2). We get an output of two embeddings (one for A i.e constraint matrix and one for D i.e gomory cuts matrix) which are then each fed into a fully connected dense layer (128 neurons for easy configuration and 256 neurons for hard configuration). Attention mechanism is then implemented by subjecting the outputs of the dense layers to a sigmoid activation function followed by multiplication of the outputs. This is then used to calculate the score. We get a probability distribution by applying the softmax function on the scores.

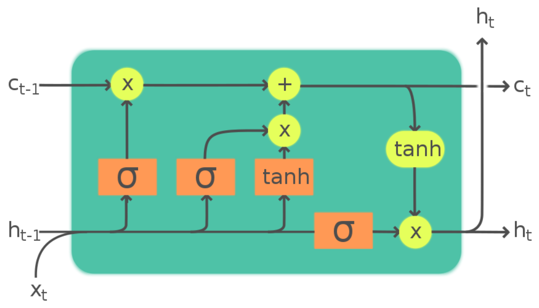


Fig 2 : LSTM Layer

Exploration technique

I also incorporated the concept of exploration using an epsilon-greedy approach. The epsilon value was exponentially decayed using the following formula:

$$epsilon = min(e^{-0.05*episode}, 0.1)$$

Pseudo-code of the Algorithm

```
For every iteration **do:**
    Initialize the state:
        While d != True:
            Normalize the constraints and cuts
            Based on the chosen policy, generate the action
            Decide to use computed action or random action based on epsilon
            Record the next state, reward obtained and whether done or not
```

Compute the discounted reward

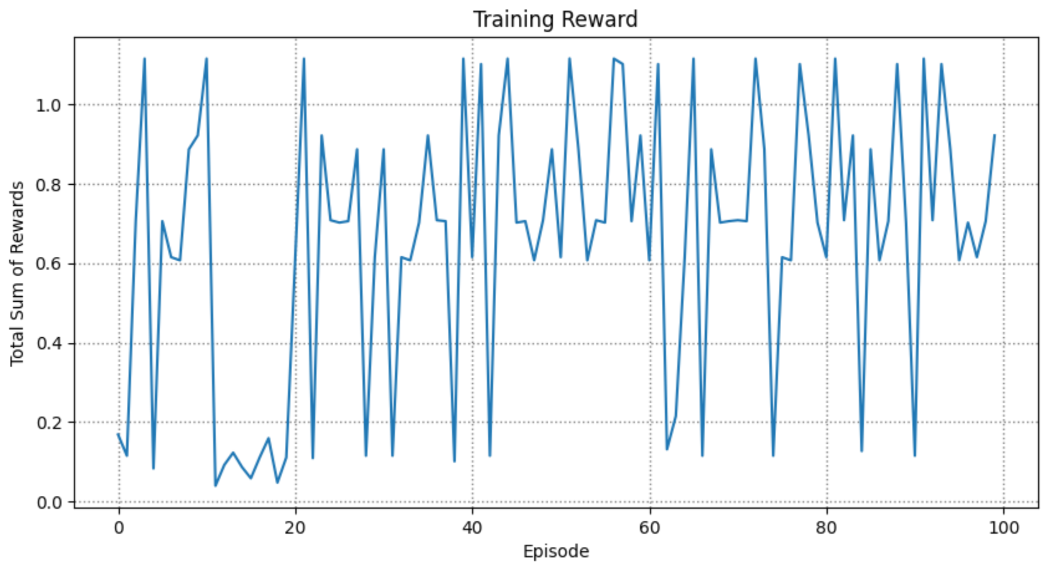
Train the ANN model and update weights accordingly

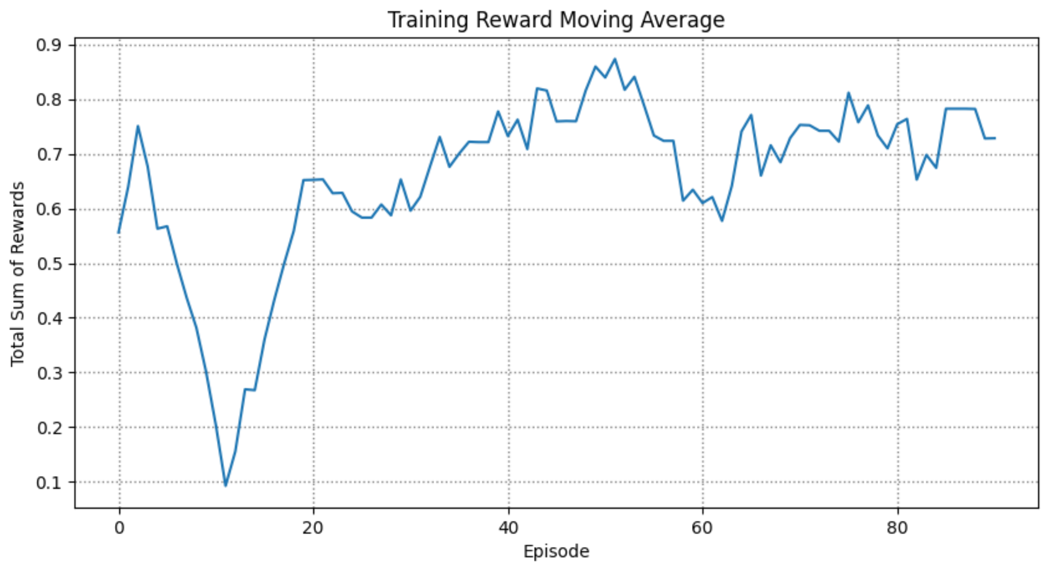
Print relevant statistics and record training time

Results

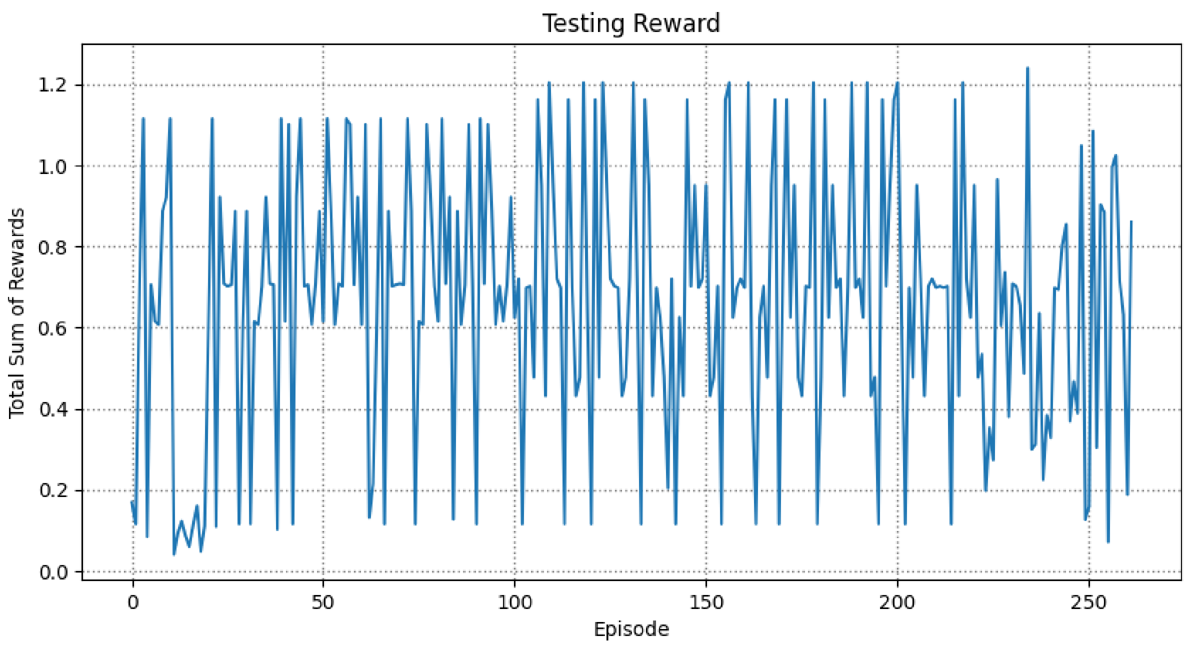
The training reward and training reward moving average over all the training iterations have been plotted below for both the configurations. I have also created new random testing instances and have plotted the testing reward and testing reward moving average for both the configurations as well.

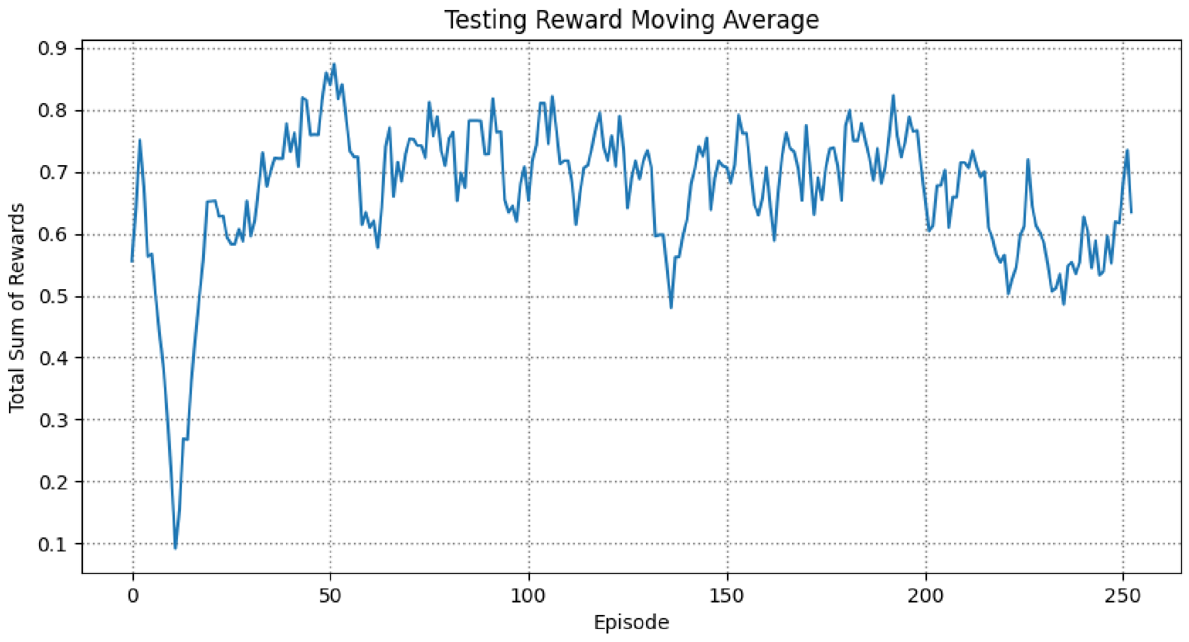
Easy Configuration [Training] -



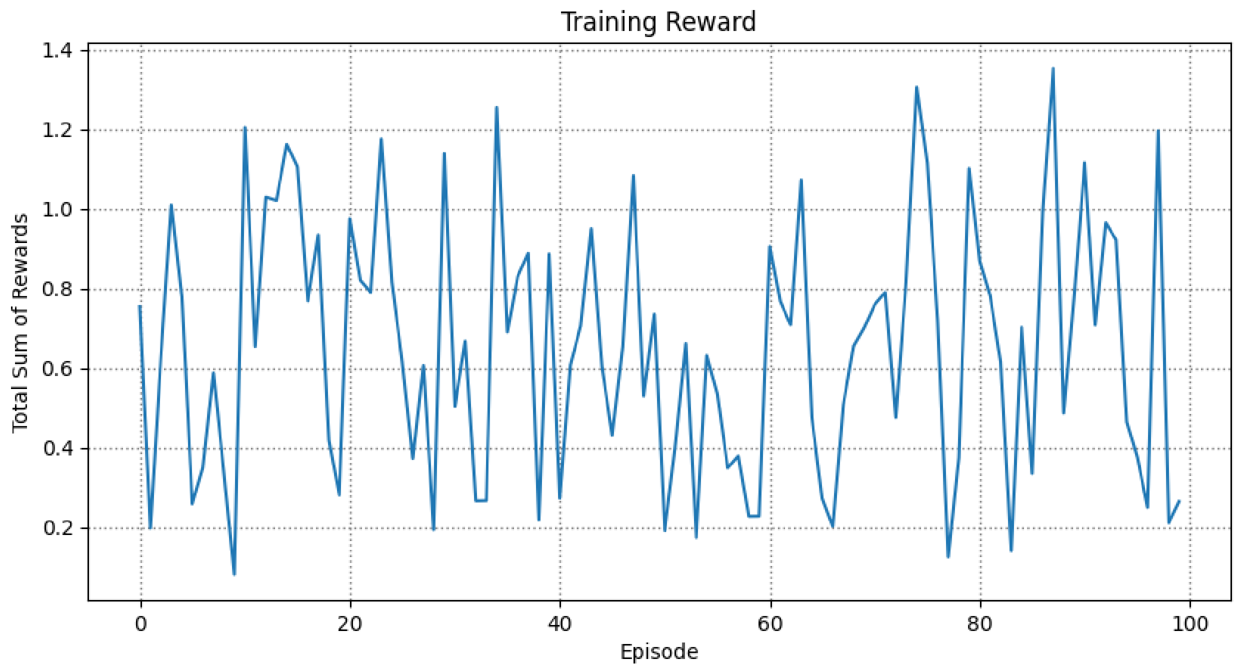


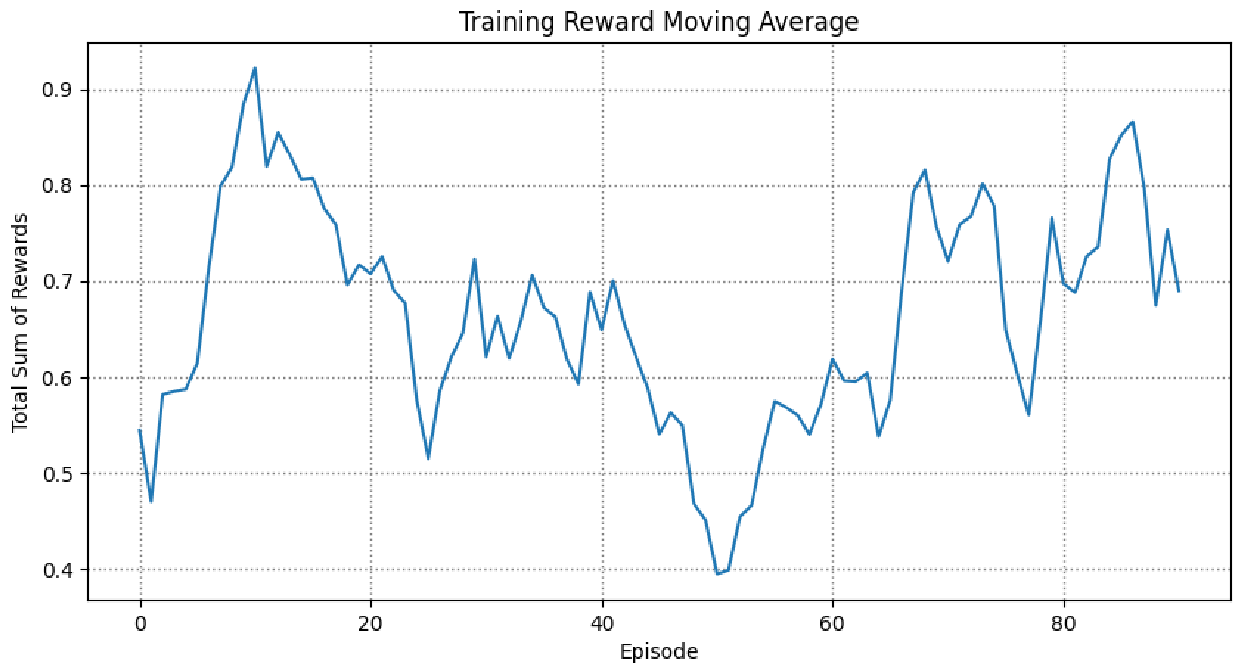
Easy Configuration [Testing] -



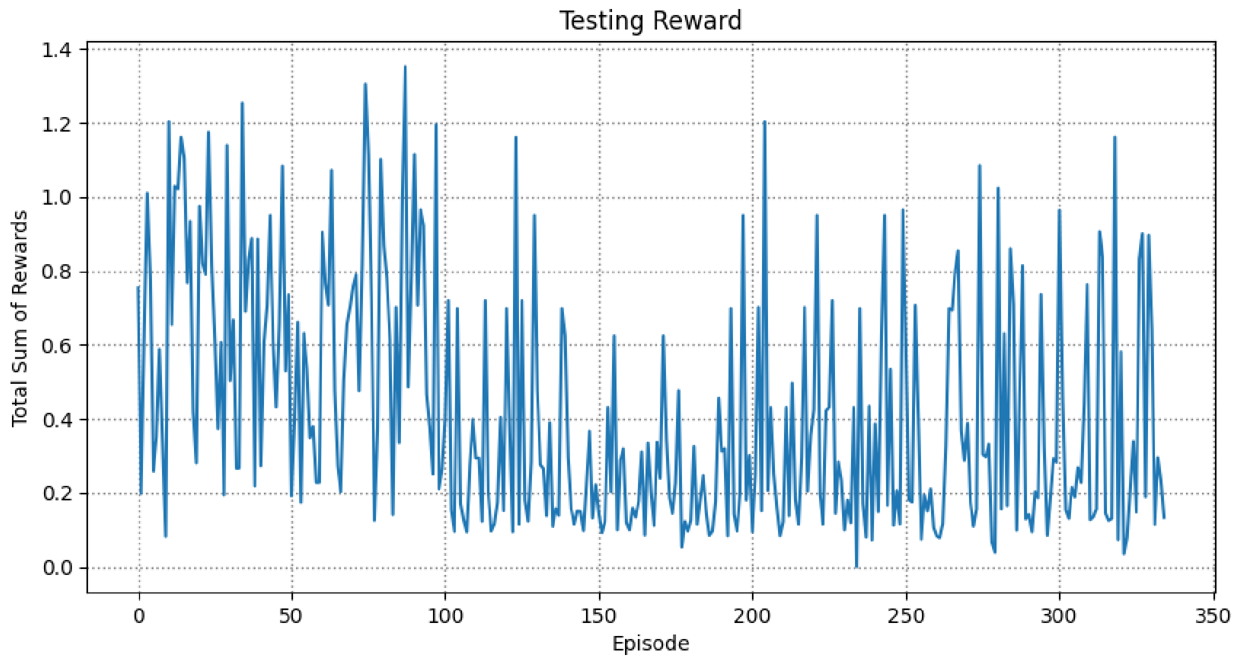


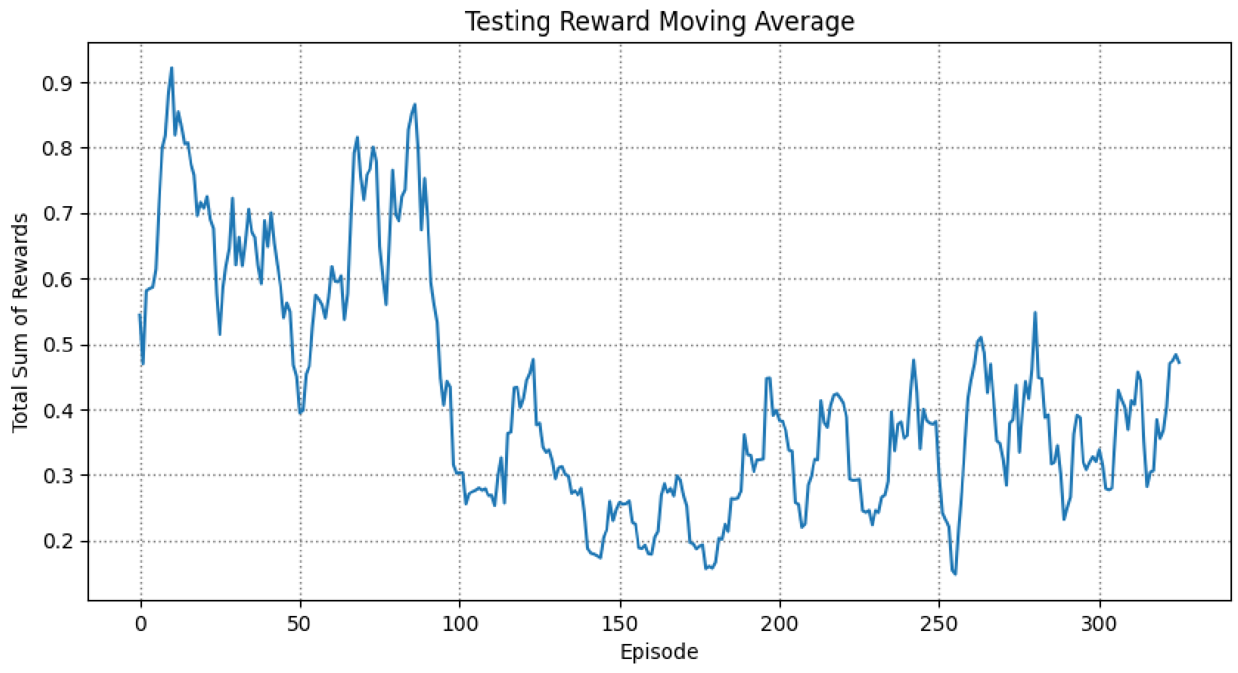
Hard Configuration [Training] -





Hard Configuration [Testing] -


Testing Reward
Total Sum of Rewards
Episode


Testing Reward Moving Average
Total Sum of Rewards
Episode