REINFORCEMENT LEARNING

ASSIGNMENT: POLICY GRADIENT (REINFORCE)

SARTHAK CHAKRABORTY (16CS30044)

---

1. **LEARNING ALGORITHM:**

   REINFORCE is a Monte Carlo policy gradient method that updates the policy step-by-step by generating episodic tasks. The objective is to learn a policy that maximizes the cumulative future reward to be received starting from any given time t until the terminal time T. Since this is a maximization problem, we optimize the policy by taking the gradient ascent with the partial derivative of the objective with respect to the policy parameter theta. The policy function is parameterized by a neural network. The algorithm calculates the discounted rewards from one episode and updates the policy parameter.

   Normalizing the reward by subtracting its mean from the rewards is a useful trick applied to reduce the variance and help the agent learn the policy faster. REINFORCE is an on-policy algorithm, that is, it samples from the policy that the algorithm updates.

2. **NETWORK ARCHITECTURE:**

   The neural network that I have used for finding the policy given a state included 2 layers. The input layer was of 37 neurons, the state space size while the output layer had 4 neurons, the size of the action space. The 1st hidden layer consisted of 32 neurons and the second hidden layer consisted of 16 neurons.

   The activation function for the hidden layers is ReLU. The final values in the output layer was then fed through a softmax function to get the probabilities. Thus the architecture in short was:

   A. *Input Layer*: 37 neurons
   B. *Output Layer*: 4 neurons
   C. *Number of Hidden Layers*: 2
   D. *Hidden Layer*: 32 neurons, 16 neurons
   E. *Activation Function*: ReLU
   F. *Output Layer Function*: Softmax

3. RESULTS:

The algorithm converged to an average reward of around 12. The reward is averaged for the last 100 episodes(implemented using a deque). It took around 8000 episodes to reach a value of 12 after which the average reward oscillated around 12. During the training phase, the increase in average reward was small.

```
Episode: 50     Avg. Reward: 0.48
Episode: 100    Avg. Reward: 0.5
Episode: 150    Avg. Reward: 0.52
Episode: 200    Avg. Reward: 0.5
Episode: 250    Avg. Reward: 0.6
Episode: 300    Avg. Reward: 0.6
Episode: 350    Avg. Reward: 0.68
Episode: 400    Avg. Reward: 0.76
Episode: 450    Avg. Reward: 0.8
Episode: 500    Avg. Reward: 0.82
Episode: 550    Avg. Reward: 0.88
Episode: 600    Avg. Reward: 1.3
Episode: 650    Avg. Reward: 0.92
Episode: 700    Avg. Reward: 1.0
Episode: 750    Avg. Reward: 1.18
Episode: 800    Avg. Reward: 1.32
Episode: 850    Avg. Reward: 1.18
Episode: 900    Avg. Reward: 1.68
Episode: 950    Avg. Reward: 1.58
```

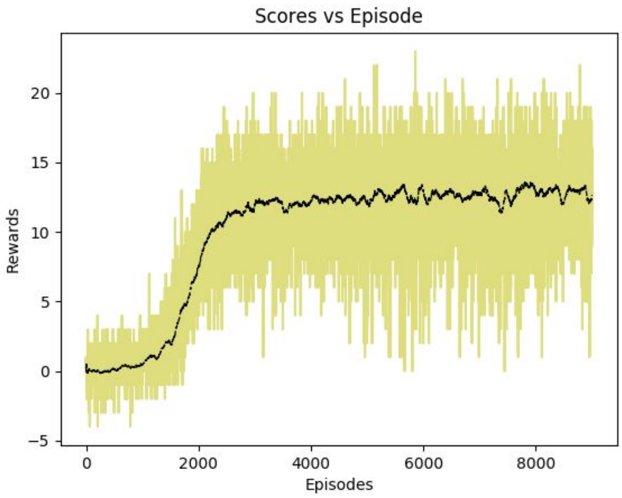
During saturation, the average rewards were as follows:

```
Episode: 8000   Avg. Reward: 10.72
Episode: 8050   Avg. Reward: 12.56
Episode: 8100   Avg. Reward: 11.5
Episode: 8150   Avg. Reward: 11.14
Episode: 8200   Avg. Reward: 11.38
Episode: 8250   Avg. Reward: 11.64
Episode: 8300   Avg. Reward: 12.02
Episode: 8350   Avg. Reward: 11.52
Episode: 8400   Avg. Reward: 11.12
Episode: 8450   Avg. Reward: 12.56
Episode: 8500   Avg. Reward: 10.74
Episode: 8550   Avg. Reward: 11.98
Episode: 8600   Avg. Reward: 11.34
Episode: 8650   Avg. Reward: 11.3
Episode: 8700   Avg. Reward: 12.34
Episode: 8750   Avg. Reward: 12.18
Episode: 8800   Avg. Reward: 12.2
```

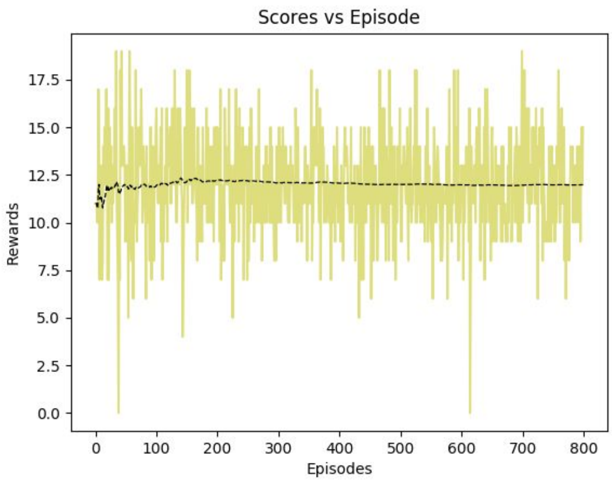
The stopping criterion for the environment was when the average reward exceeded 13.

```
Environment solved!     Average Score: 13.80
```

The training curve is as follows:



With the saved model parameter, when the policy was evaluated for 800 episodes, it yielded a graph which is shown below:



GAMMA used for the above experiment was **1.0**. I have also attached the model parameters that was trained in the above process and it is saved as filename `model_param`.

The zip file submitted has a file `Training_phase_saved_model.ipynb` that has the code when the model was trained.

4. FUTURE WORK

As it can be seen that, the reward the agent can accumulate becomes saturated after reached an average reward of 12. The same was the condition even when the network architecture was changed. Hence, it shows that an on-policy solution does not suit this environment. Thus we need an off-policy solution. A very good candidate to solve this environment is to use Deep Q-Learning Network(DQN) algorithm. It is an off-policy value gradient algorithm that samples from a behavioral policy but updates a target policy. This ensures that the agent can choose from actions that will never be sampled if we had followed the target policy.