# Report – Assignment 3

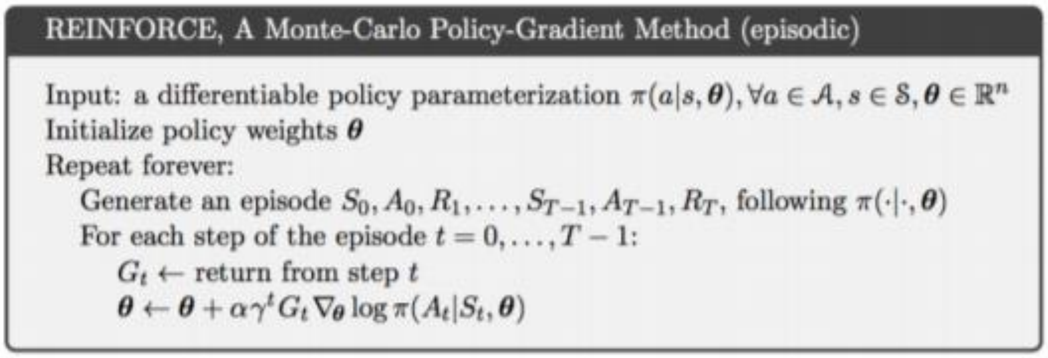
**Name: Sandesh Kumar Srivastava**

**UBIT: Sandeshk**

## Task – Implement REINFORCE

REINFORCE is a Monte-Carlo Policy Gradient algorithm where we update the policy parameters directly instead of value parameters. The parameters are updated following stochastic gradient ascent based on the return obtained for each episode.

Policy parameters start with almost equal values for each action giving it a random behavior. Using this policy, we generate an episode and based on the rewards collected for this episode we update the policy parameters gradually. Since the change in parameters is directly proportional to the rewards collected, it tends to increase the probability of good actions and decrease the probability of bad actions.



```
REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Initialize policy weights $\boldsymbol{\theta}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G_t \leftarrow$ return from step $t$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t,\boldsymbol{\theta})$
```

### Environment:

I have used Open AI Gym's **CartPole-v1** environment for checking REINFORCE algorithm performance. Some of the details of this environment are as follows:

a) Observation/State: It is of the form of Box (4) where the four values correspond to cart position, cart velocity, pole angle and pole angular velocity.
b) Actions: It is of the form of Discrete (2) and consists of Action set (A) = {0,1} where 0 means push cart to left and 1 means push cart to right
c) Reward: It is 1 for each step till the game terminates. Reward set ® = {1}
d) Episode termination: Episode terminates when pole angle is more than 12 degrees or cart reaches the edge of display (cart position more than 2.4) or episode length is more than 500.
e) Starting state: It is randomly selected from all possible states.

## Implementation:

I have defined following two classes:

1. `ReinforceAgent` : It is the main agent class where all the REINFORCE algorithm is implemented. It consists of following methods:
   a) `__init__()`: It is used to initialize class variables and define hyperparameters.
   b) `build_model()`: It is used to define a shallow neural network.
   c) `step()`: It is used to select action to be taken for a given state.
   d) `modify_policy_weights()`: It is used to calculate the gradient for the policy parameters update and train the neural network. This is the method where I am calculating the below equation:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\gamma^t G_t \frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}$$

2. `Environment`: It is the class used to define the CartPole-v1 environment and consists of following methods:
   a) `__init__()`: It is used to instantiate environment and define some parameters like number of episodes.
   b) `train()`: It is the main execution method where we generate episodes and call relevant agent methods. I am also storing the state, action, reward and theta(probability distribution) values for each step here.

## Results:

We capture the rewards collected by the agent for each episode and observe that the agent starts poorly initially but as the agent learns the rewards collected increase and often reach the maximum score of 500 towards the end.

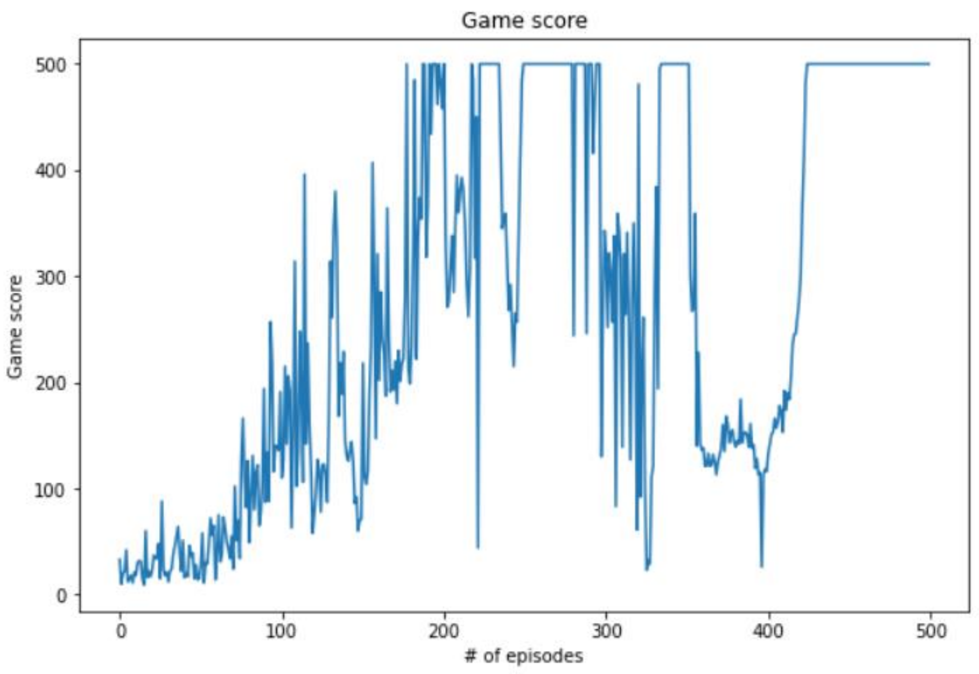
Hyper-parameters used with the algorithm are listed here:

```
        self.memory = deque(maxlen=1000)
        self.gamma = 0.99
        self.alpha = 0.0001
        self.number_of_episodes = 500
        self.C = 1
        self.reward_average_episodes = 10
```

Results are captured with following graphs:

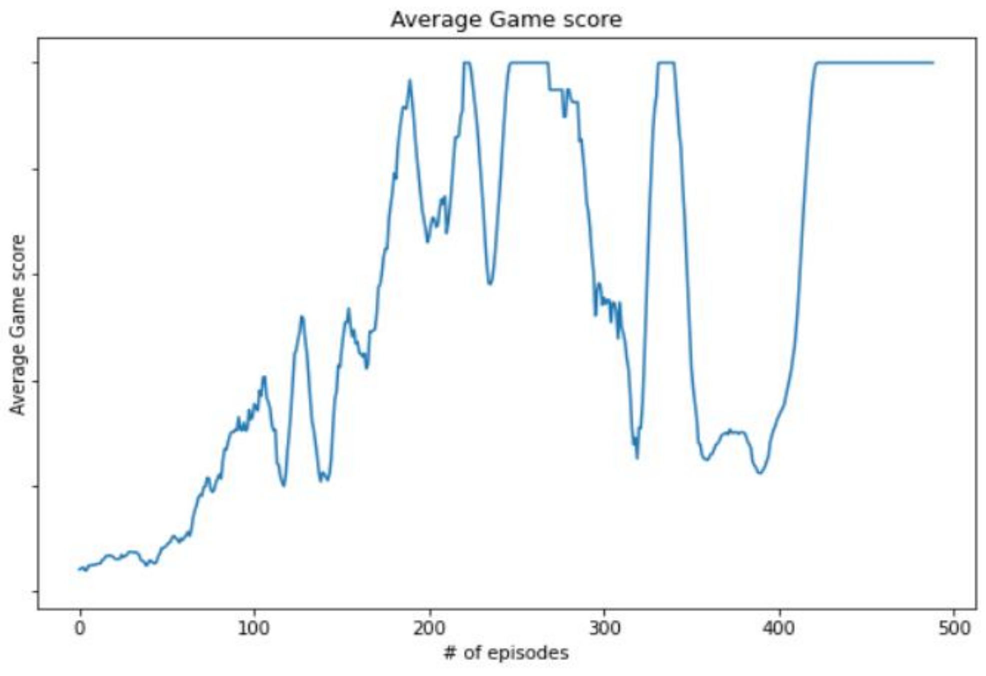
1. **Game score graph**: It captures the reward collected by the agent for different episodes.

```
Game score:  [33.0, 10.0, 20.0, 22.0, 42.0, 12.0, 15.0, 18.0, 11.0, 21.0, 18.0, 30.0,
32.0, 31.0, 15.0, 9.0, 60.0, 16.0, 22.0, 17.0, 23.0, 37.0, 34.0, 34.0, 48.0, 15.0,
88.0, 24.0, 18.0, 21.0, 12.0, 22.0, 24.0, 38.0, 46.0, 55.0, 64.0, 39.0, 22.0, 51.0,
16.0, 20.0, 17.0, 46.0, 36.0, 39.0, 15.0, 28.0, 14.0, 15.0, 28.0, 58.0, 11.0, 30.0,
29.0, 48.0, 72.0, 56.0, 65.0, 14.0, 41.0, 75.0, 31.0, 38.0, 73.0, 63.0, 50.0, 43.0,
34.0, 55.0, 24.0, 102.0, 51.0, 70.0, 34.0, 126.0, 166.0, 108.0, 82.0, 126.0, 49.0,
93.0, 131.0, 80.0, 115.0, 122.0, 65.0, 77.0, 105.0, 194.0, 87.0, 134.0, 88.0, 257.0,
218.0, 116.0, 140.0, 140.0, 136.0, 191.0, 110.0, 127.0, 215.0, 142.0, 206.0, 192.0,
63.0, 157.0, 314.0, 102.0, 135.0, 248.0, 189.0, 106.0, 396.0, 142.0, 237.0, 163.0,
118.0, 58.0, 76.0, 97.0, 127.0, 117.0, 78.0, 122.0, 123.0, 113.0, 87.0, 169.0, 314.0,
261.0, 345.0, 380.0, 329.0, 168.0, 218.0, 189.0, 229.0, 145.0, 131.0, 126.0, 136.0,
144.0, 124.0, 86.0, 92.0, 60.0, 70.0, 71.0, 218.0, 111.0, 104.0, 116.0, 187.0, 232.0,
407.0, 297.0, 147.0, 321.0, 202.0, 285.0, 242.0, 227.0, 187.0, 364.0, 243.0, 191.0,
211.0, 193.0, 220.0, 180.0, 230.0, 201.0, 217.0, 223.0, 283.0, 500.0, 213.0, 199.0,
235.0, 326.0, 485.0, 222.0, 335.0, 374.0, 354.0, 500.0, 500.0, 318.0, 366.0, 500.0,
434.0, 500.0, 500.0, 500.0, 462.0, 500.0, 484.0, 458.0, 500.0, 339.0, 271.0, 276.0,
310.0, 338.0, 285.0, 349.0, 395.0, 360.0, 380.0, 393.0, 380.0, 348.0, 295.0, 262.0,
306.0, 500.0, 485.0, 317.0, 450.0, 44.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 423.0, 345.0, 355.0, 359.0, 310.0,
268.0, 292.0, 253.0, 215.0, 265.0, 257.0, 331.0, 399.0, 484.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 244.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
246.0, 500.0, 500.0, 500.0, 416.0, 465.0, 500.0, 500.0, 500.0, 130.0, 264.0, 343.0,
317.0, 252.0, 322.0, 302.0, 257.0, 338.0, 83.0, 359.0, 344.0, 313.0, 139.0, 321.0,
264.0, 341.0, 254.0, 127.0, 274.0, 350.0, 261.0, 61.0, 481.0, 92.0, 202.0, 261.0, 99.0,
23.0, 32.0, 29.0, 111.0, 121.0, 289.0, 384.0, 194.0, 495.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 300.0, 267.0, 270.0, 359.0, 140.0, 228.0, 143.0, 136.0, 138.0, 121.0,
121.0, 133.0, 121.0, 122.0, 132.0, 125.0, 113.0, 125.0, 131.0, 139.0, 160.0, 135.0,
168.0, 161.0, 143.0, 153.0, 155.0, 143.0, 139.0, 145.0, 142.0, 184.0, 143.0, 152.0,
153.0, 150.0, 139.0, 161.0, 138.0, 141.0, 119.0, 128.0, 113.0, 115.0, 26.0, 112.0,
118.0, 116.0, 133.0, 142.0, 151.0, 153.0, 166.0, 157.0, 163.0, 178.0, 173.0, 153.0,
192.0, 174.0, 190.0, 184.0, 204.0, 235.0, 245.0, 246.0, 261.0, 277.0, 301.0, 365.0,
409.0, 481.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0]
```

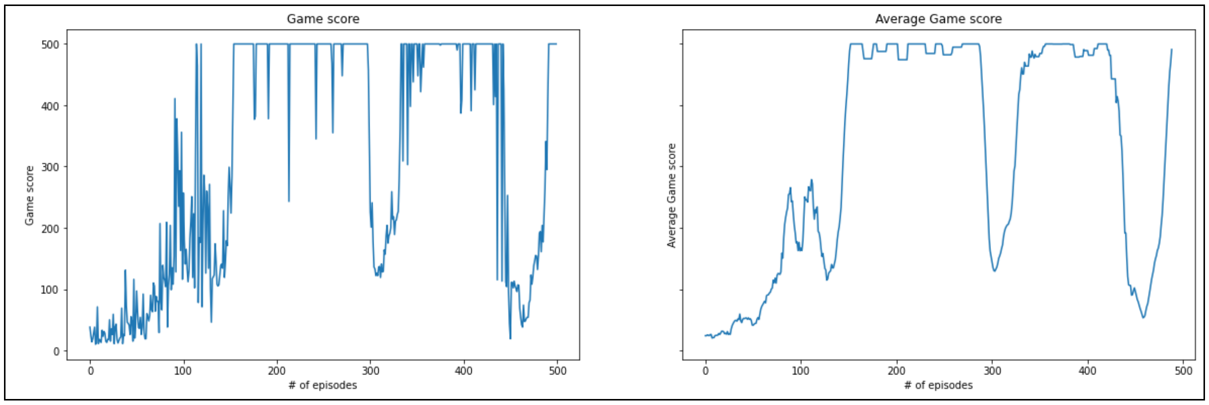


2. **Average game score graph**: It captures the average reward collected by the agent for every `self.reward_average_episodes.`

```
Average Game score:  [20.9, 22.1, 23.0, 20.3, 20.0, 24.5, 24.3, 25.4, 25.0, 25.5, 26.2,
26.4, 26.7, 30.0, 30.6, 33.4, 34.2, 33.8, 34.2, 33.1, 31.6, 30.6, 31.0, 30.8, 34.8, 32.4,
33.9, 34.3, 37.3, 37.7, 37.5, 36.8, 37.6, 36.6, 35.0, 30.1, 29.0, 28.2, 24.6, 25.8, 29.6,
29.0, 27.4, 26.7, 27.6, 33.3, 36.1, 41.2, 41.1, 42.4, 44.1, 46.1, 46.9, 51.3, 52.8, 50.6,
49.3, 46.2, 50.3, 48.6, 51.3, 53.3, 56.5, 52.6, 58.9, 70.5, 77.0, 81.8, 88.9, 91.4, 90.5,
98.5, 99.5, 107.6, 107.2, 97.1, 94.0, 96.3, 103.1, 106.9, 111.0, 106.7, 124.4, 134.7,
134.1, 141.6, 147.9, 151.0, 150.7, 153.0, 152.3, 165.0, 153.5, 152.3, 159.9, 152.2, 153.9,
171.7, 162.8, 165.3, 177.4, 174.8, 171.2, 190.2, 185.2, 202.6, 203.2, 183.6, 179.2, 173.3,
158.2, 152.0, 153.1, 121.3, 119.3, 107.9, 102.9, 99.8, 110.9, 134.7, 151.1, 172.9, 199.2,
224.3, 228.9, 238.4, 246.0, 260.2, 257.8, 239.5, 226.0, 205.1, 181.5, 161.0, 152.8, 140.2,
127.3, 111.4, 104.0, 112.7, 111.2, 108.0, 105.2, 111.5, 126.1, 157.6, 181.3, 189.0, 214.0,
212.4, 229.8, 243.6, 254.7, 254.7, 267.9, 251.5, 240.9, 247.3, 234.5, 236.3, 225.8, 224.6,
222.0, 225.0, 210.9, 214.9, 245.8, 246.0, 246.6, 248.1, 262.7, 288.2, 290.3, 302.1, 317.2,
324.3, 324.3, 353.0, 364.9, 378.0, 395.4, 390.3, 418.1, 434.6, 447.2, 458.0, 458.0, 456.4,
470.4, 483.8, 467.7, 451.4, 429.0, 410.0, 393.8, 376.1, 361.0, 352.1, 342.3, 330.3, 335.7,
346.6, 353.8, 352.3, 344.7, 346.8, 361.9, 370.9, 366.6, 373.6, 338.7, 350.7, 365.9, 386.4,
410.2, 429.6, 429.6, 431.1, 449.4, 454.4, 500.0, 500.0, 500.0, 500.0, 492.3, 476.8, 462.3,
448.2, 429.2, 406.0, 385.2, 360.5, 332.0, 308.5, 291.9, 290.5, 294.9, 307.4, 326.4, 349.6,
370.4, 395.1, 423.6, 447.1, 471.4, 488.3, 498.4, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 474.4, 474.4, 474.4, 474.4, 474.4, 474.4, 474.4, 474.4, 449.0, 449.0,
474.6, 474.6, 466.2, 462.7, 462.7, 462.7, 462.7, 425.7, 427.5, 411.8, 393.5, 368.7, 359.3,
343.0, 318.7, 302.5, 260.8, 283.7, 291.7, 288.7, 270.9, 277.8, 272.0, 275.9, 275.6, 254.5,
273.6, 272.7, 264.4, 239.2, 273.4, 250.5, 244.3, 236.3, 220.8, 210.4, 186.2, 154.1, 139.1,
145.1, 125.9, 155.1, 154.3, 177.7, 217.8, 265.5, 312.3, 359.4, 398.3, 436.2, 457.3, 468.9,
499.5, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 480.0, 456.7, 433.7,
419.6, 383.6, 356.4, 320.7, 284.3, 248.1, 210.2, 192.3, 178.9, 164.0, 140.3, 139.5, 129.2,
126.2, 125.1, 124.4, 126.2, 130.1, 130.3, 135.0, 138.9, 140.0, 142.8, 147.0, 148.8, 149.6,
150.2, 148.4, 153.3, 150.8, 149.9, 150.9, 150.6, 149.0, 150.8, 150.7, 150.3, 148.0, 142.4,
139.4, 135.7, 123.0, 119.2, 117.1, 112.6, 112.1, 112.2, 115.4, 117.9, 123.2, 127.4, 141.1,
147.7, 153.2, 156.9, 162.8, 166.0, 169.9, 173.0, 176.8, 184.6, 192.8, 199.6, 208.4, 220.8,
231.7, 250.8, 272.7, 302.4, 332.0, 358.5, 384.0, 409.4, 433.3, 455.6, 475.5, 489.0, 498.1,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0]
```



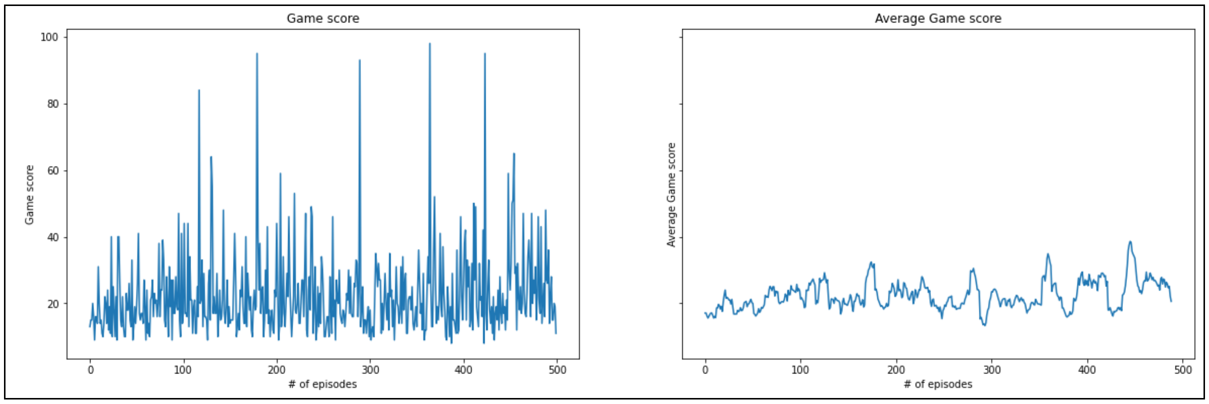
For the initial set of hyperparameters, below is a capture of both graphs. Since REINFORCE algorithm has high variance, the same is reflected in the below graphs.



## Effect of other parameters on performance of REINFORCE:

In this section, I have tried to show the effect of modifying some of the hyperparameters on the game score and average score.

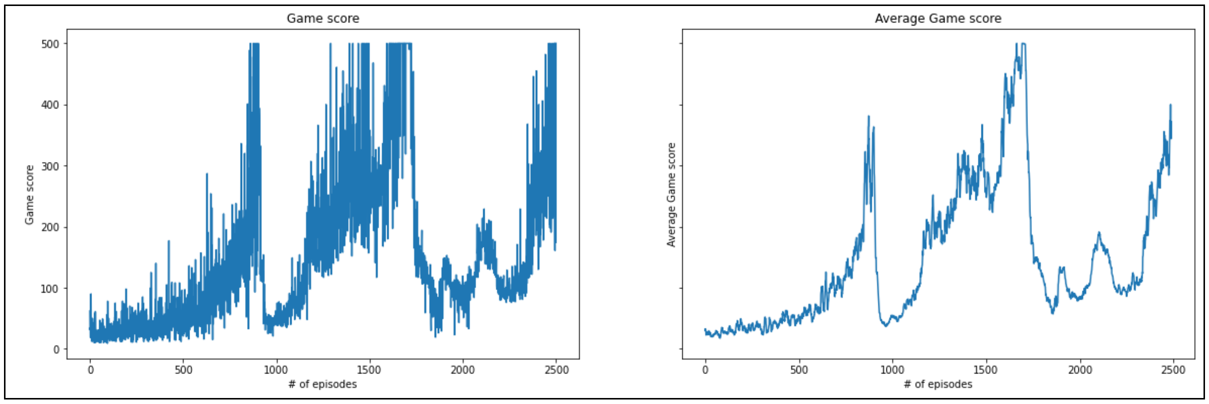
1. `self.C = 5`



Keeping other parameters constant, if I only change the value of C from 1 to 5, the algorithm runs relatively faster as the policy parameters are updated less often. However, the score is very less and it never reaches the maximum score of 500.

2. To check the behavior further, I increased the number of episodes to 2500 and captured the graph below.

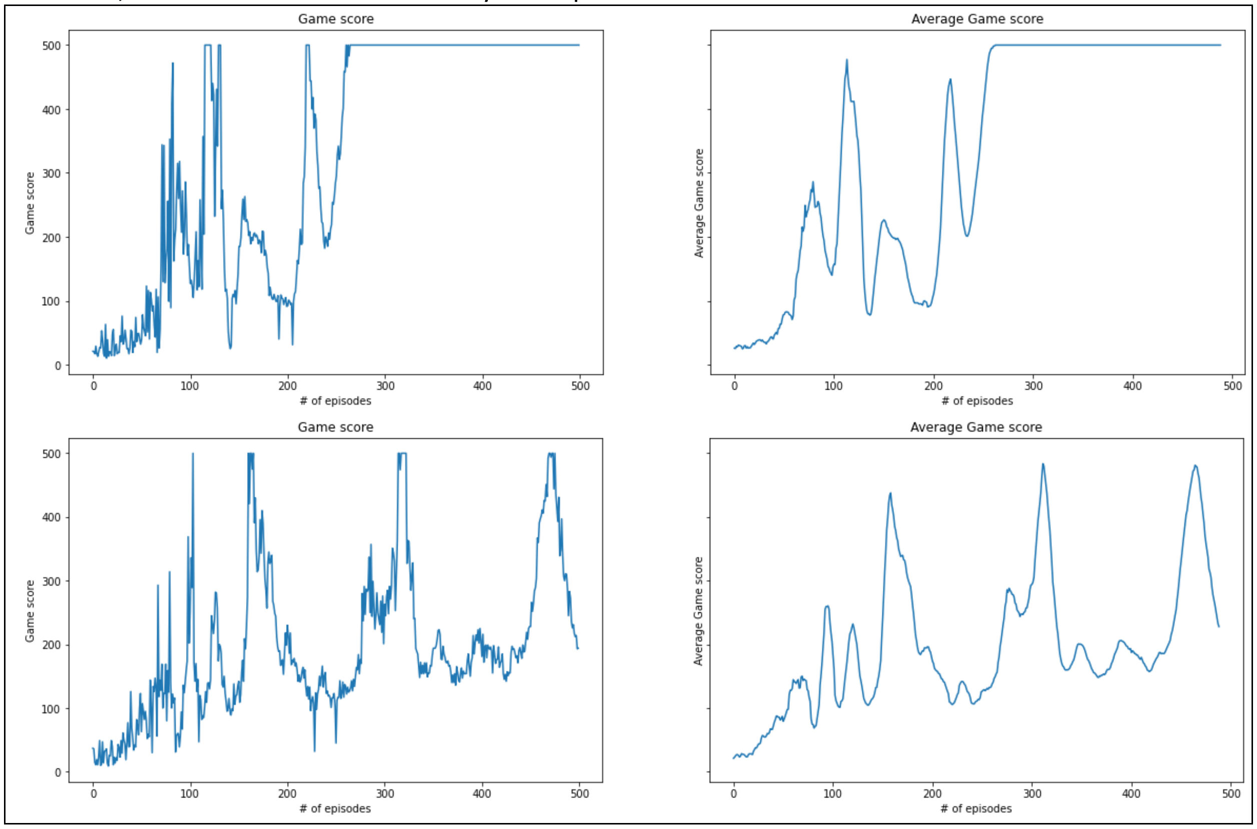
```
self.C = 5
self.number_of_episodes = 2500
```



As we can see that the performance increases as compared to (1) but it takes a large number of episodes to reach that. Even then, the performance is not very stable.

```
3. self.gamma = 0.7
```

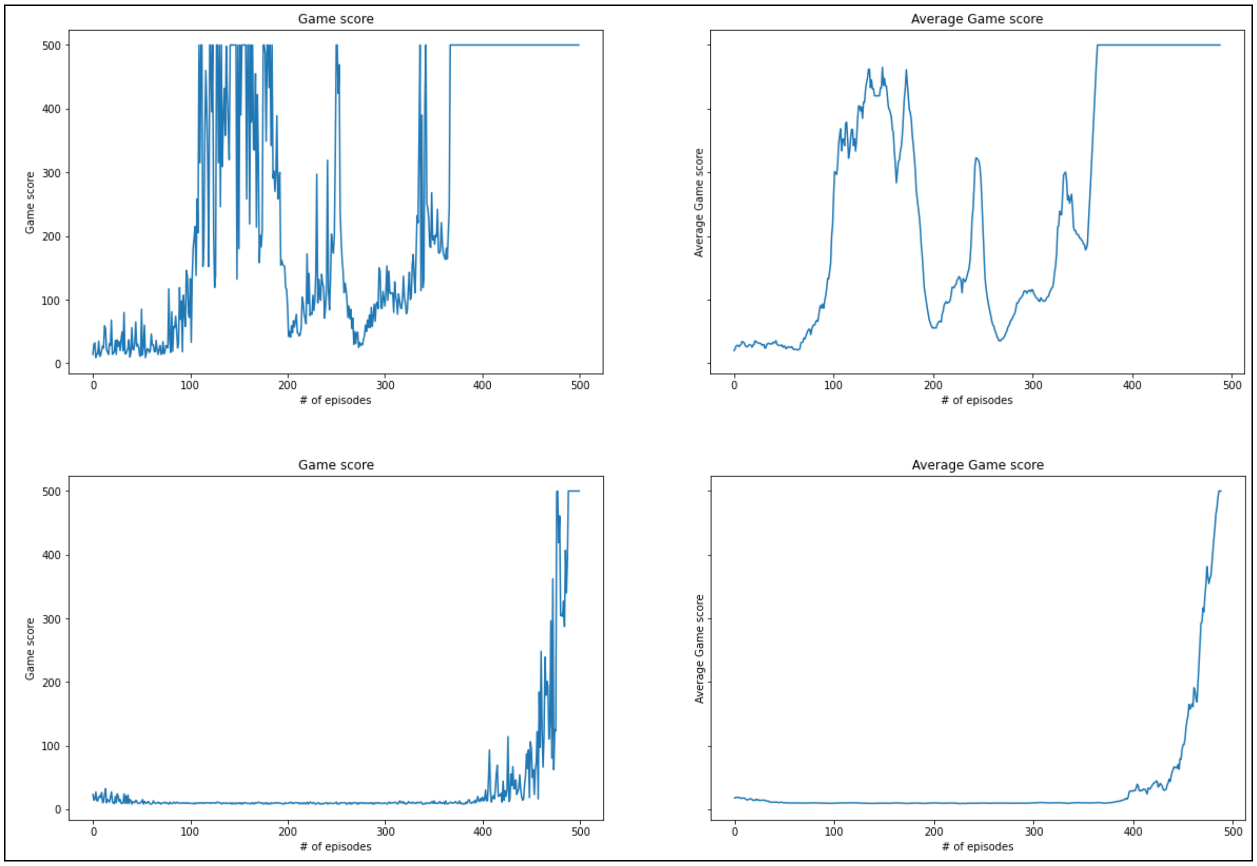
Keeping other parameters constant, I changed the discount factor gamma to 0.7. In this case, the graph varies significantly with each run. Sometimes the rewards collected are very good however for some iterations, the rewards collected are very low. I present results obtained for two such iterations below:



We can explain this behavior by the argument that since REINFORCE has high variance so by keeping gamma low we are giving less importance to future rewards and more importance to current reward which in turn show high variance.

```
4. self.alpha = 0.01
```

Keeping other parameters constant, I tried modifying the learning rate alpha to 0.01 and studied the behavior. As seen in the previous section, the results obtained are very stable. I present results obtained for two such iterations below:



## Conclusion:

I have implemented REINFORCE algorithm and studied the results on CartPole-v1 gym environment. The algorithm is relatively easy to implement and shows good results on the environment. However, the REINFORCE algorithm suffers from high variance which can be clearly seen in the results obtained. I then conclude by experimenting with some hyperparameter to study their effect on the algorithm in general.