REINFORCEMENT LEARNING
CS6700

Programming Assignment - 2

Course Instructor - Balaraman Ravindran

Submitted by :

Biswajit Rout CE22M042

Shubham Singh MA22C043

Dueling-DQN on Cartpole Environment

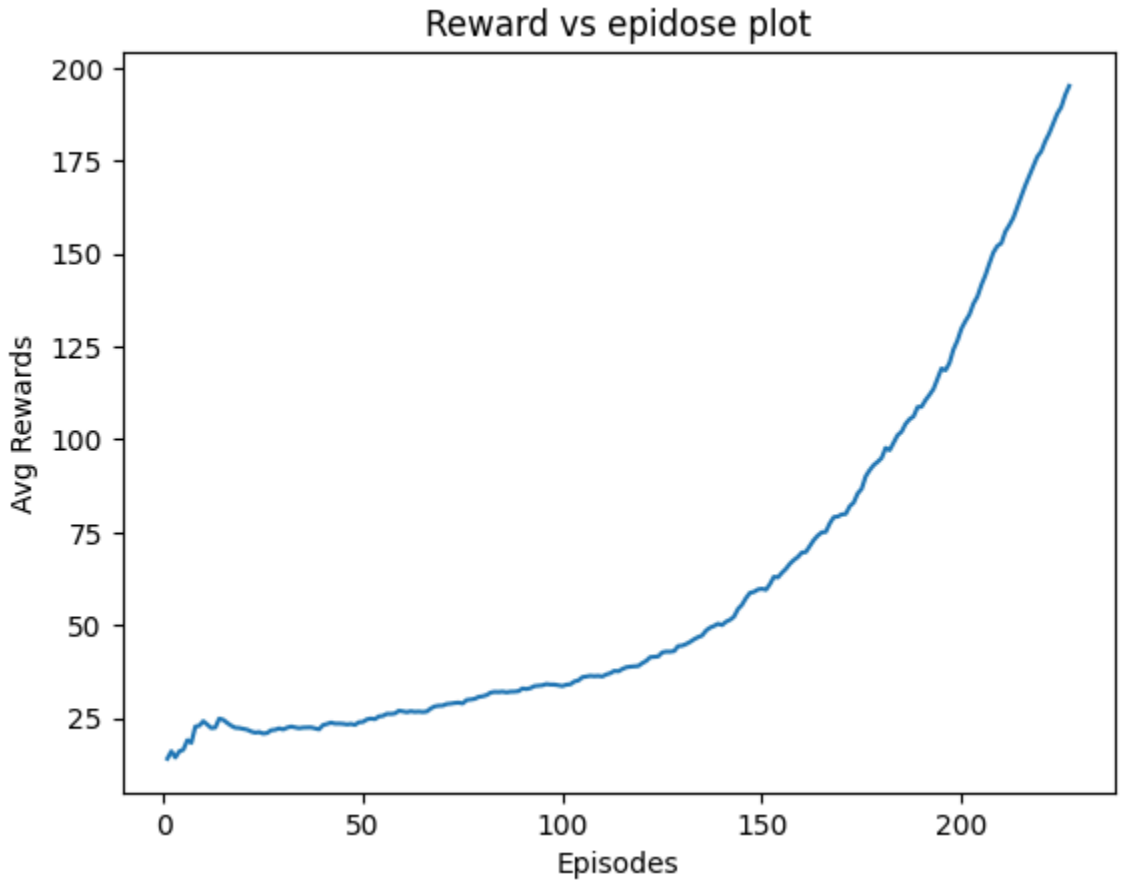
hyperparameter tuning on type -1:

- We performed a bunch of hyperparameters to solve the environment for reward 195.0 from the last 100 actions on average, some of them are listed below.

Hyperparameter set-1 :

Buffer Size = 1e5
Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

```
Environment solved in 227 episodes!     Average Score: 195.22
```

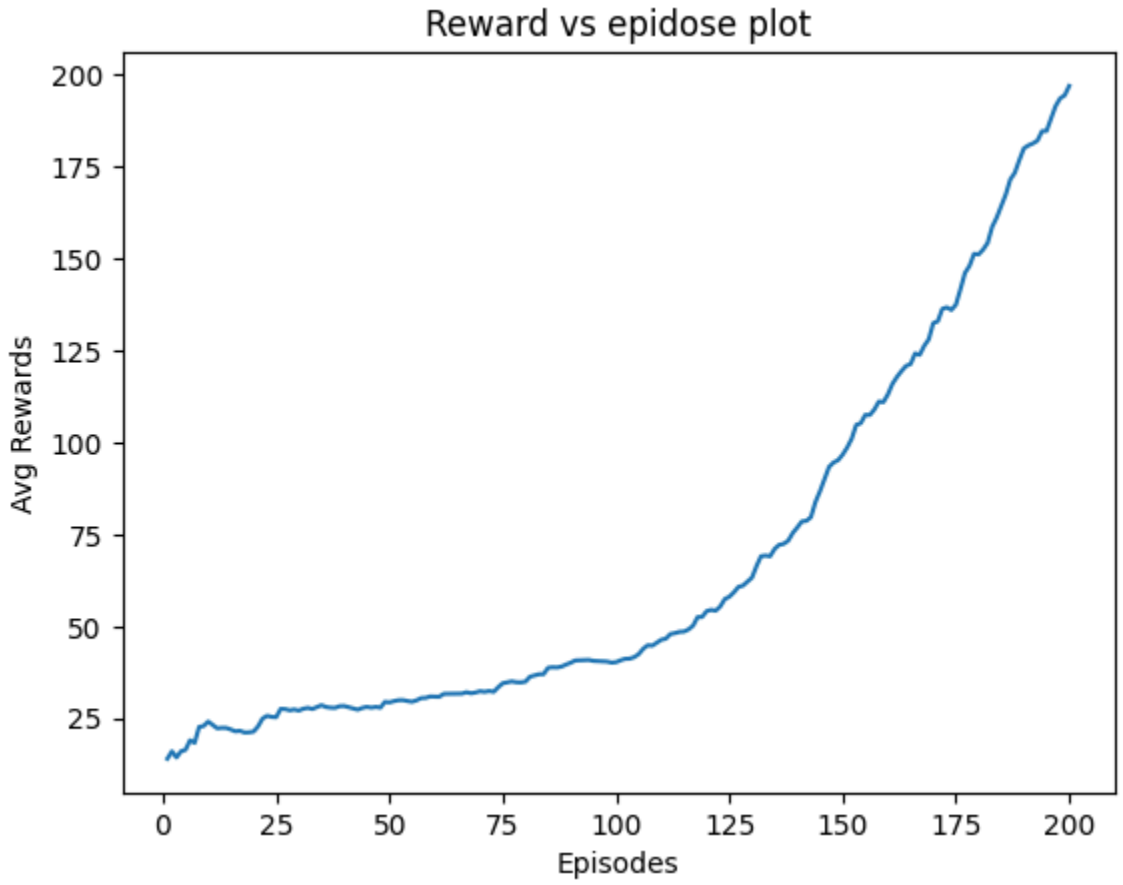


Hyperparameter set-2 :

- We tried to increase the learning rate, resulting in reducing the number of steps to solve the environment.

Buffer Size = 1e5
Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-3
Update every = 20

```
Environment solved in 200 episodes!     Average Score: 196.94
```

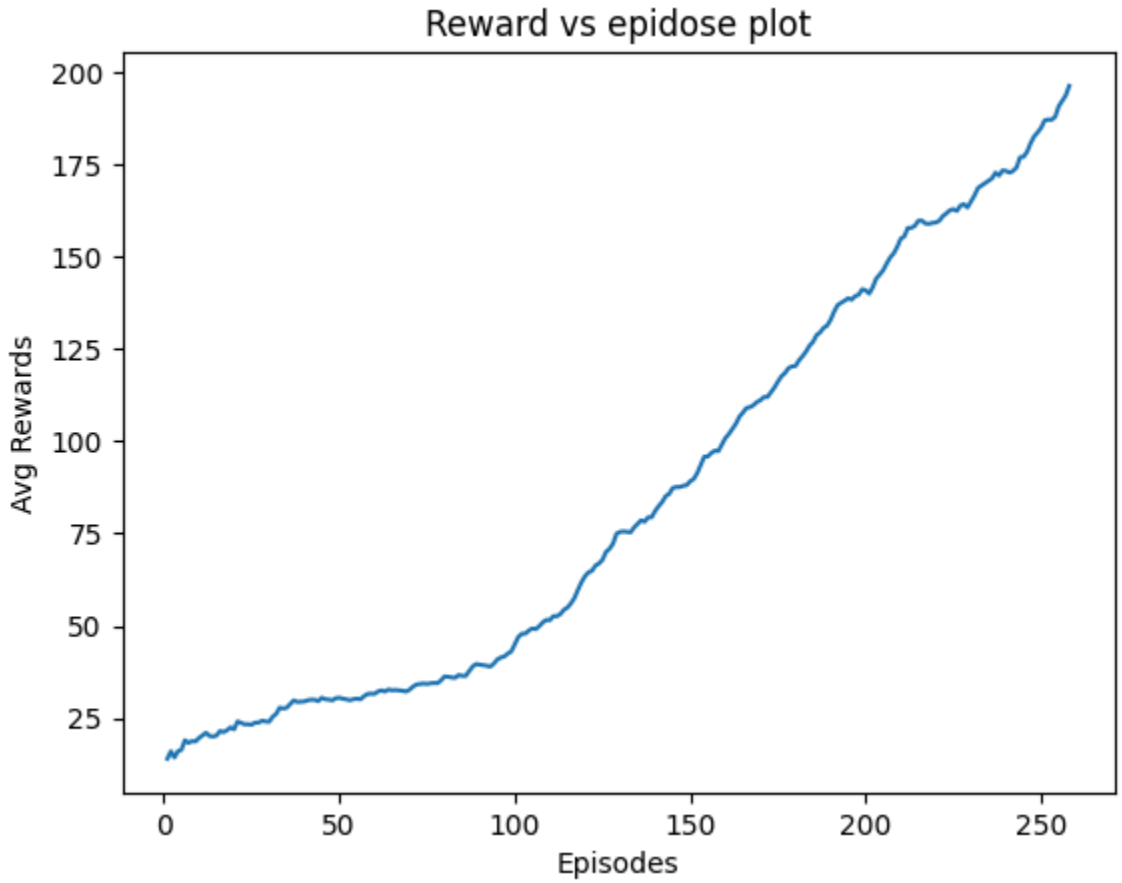


Hyperparameter set-3 :

- We reduced the batch size, but it increased the steps to solve the environment.

Buffer Size = 1e5
Batch Size =128
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

```
Environment solved in 258 episodes!     Average Score: 196.27
```

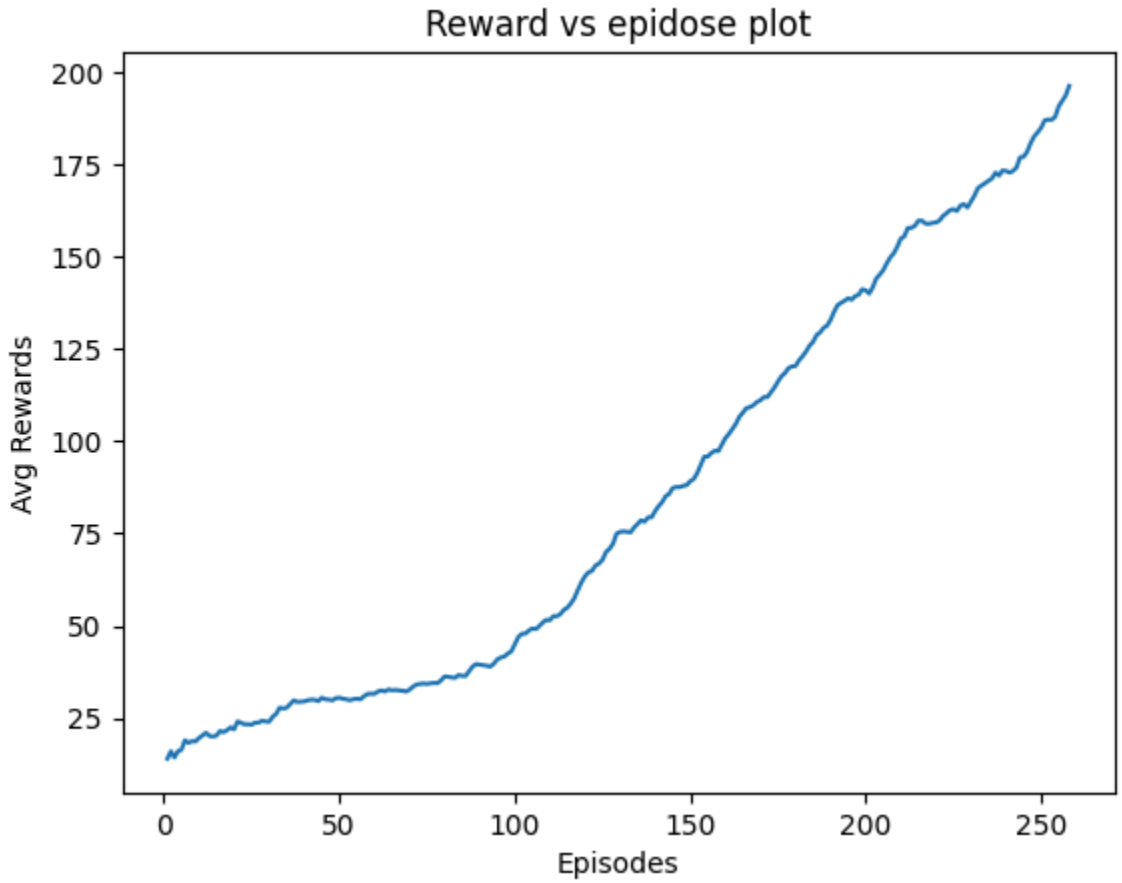


Hyperparameter set-4 :

- We increased Buffer Size, but that doesn't seem to affect the learning.

Buffer Size = 1e6
Batch Size =128
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

```
Environment solved in 258 episodes!     Average Score: 196.27
```



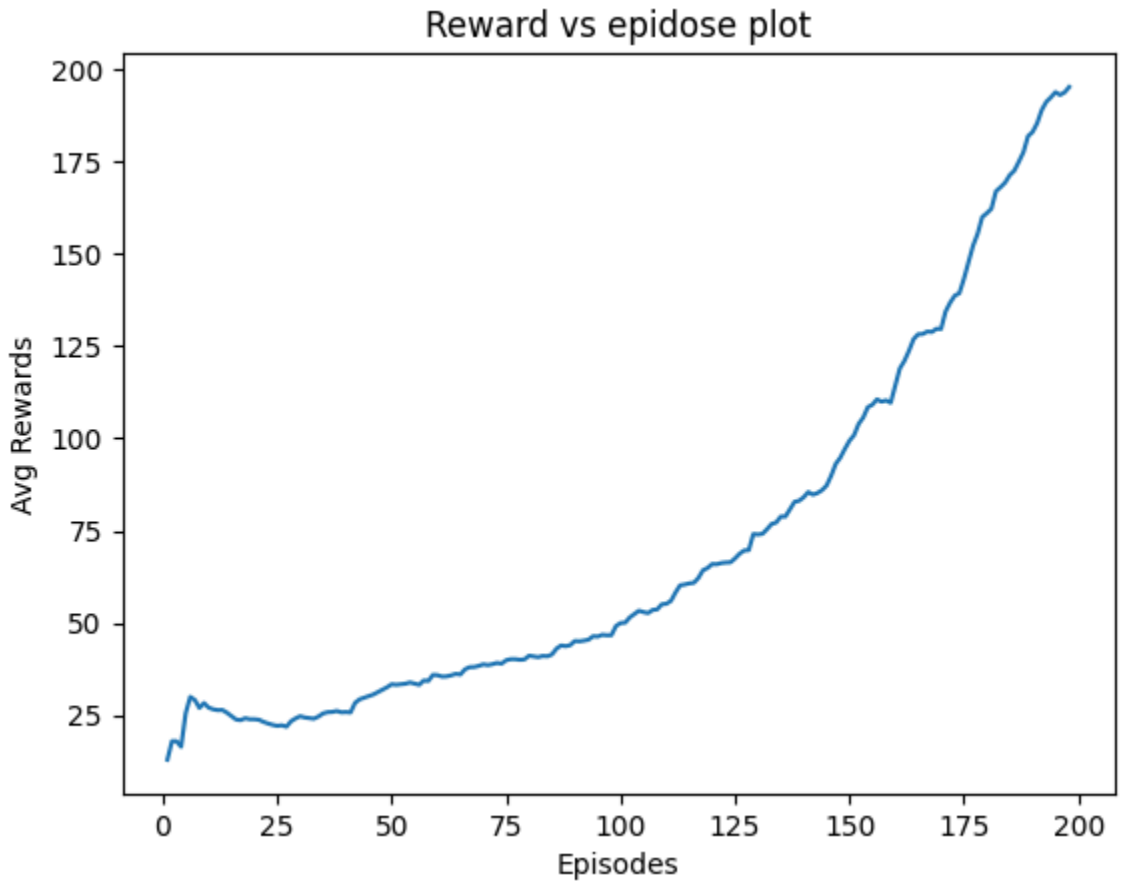
hyperparameter tuning on type -2:

- We performed a bunch of hyperparameters, some of them are listed below.

Hyperparameter set-1 :

Buffer Size = 1e5
Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

```
Environment solved in 198 episodes!     Average Score: 195.14
```

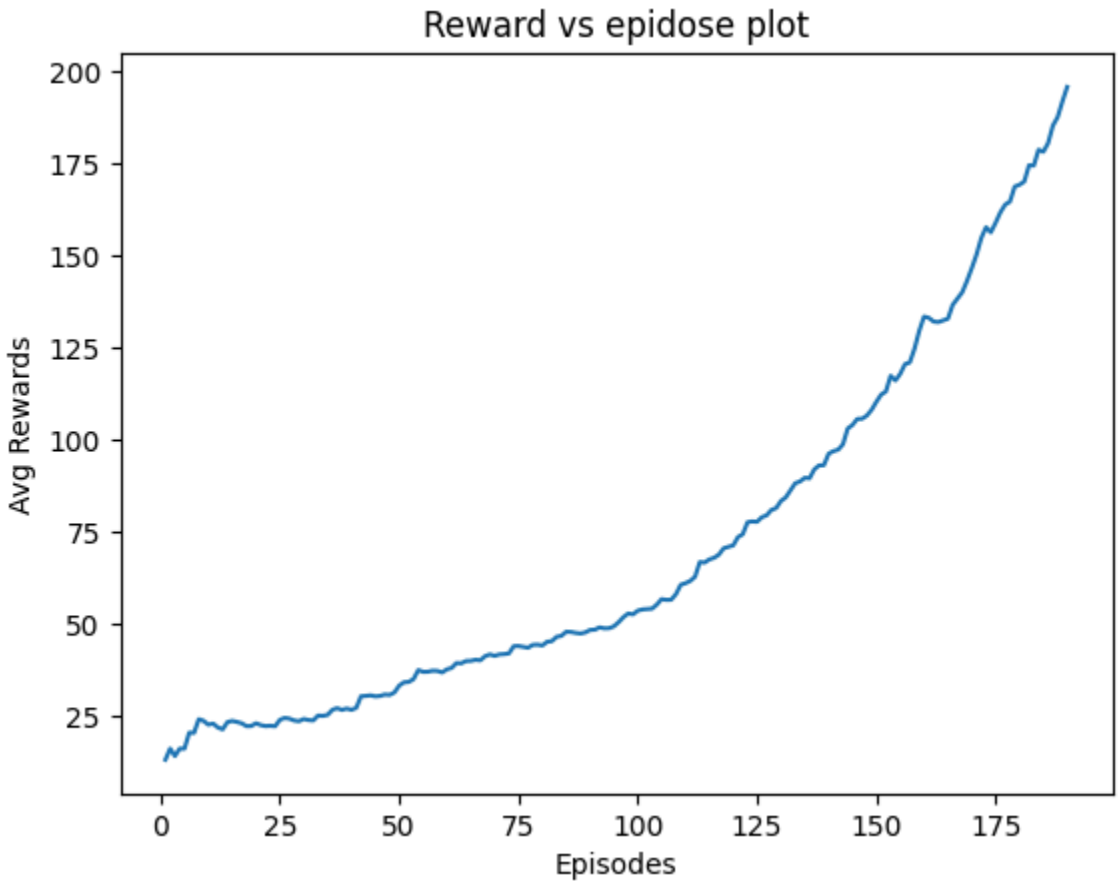


Hyperparameter set-2 :

- We tried to increase the learning rate, resulting in reducing the number of steps to solve the environment.

Buffer Size = 1e5
Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-3
Update every = 20

```
Environment solved in 190 episodes!     Average Score: 195.82
```

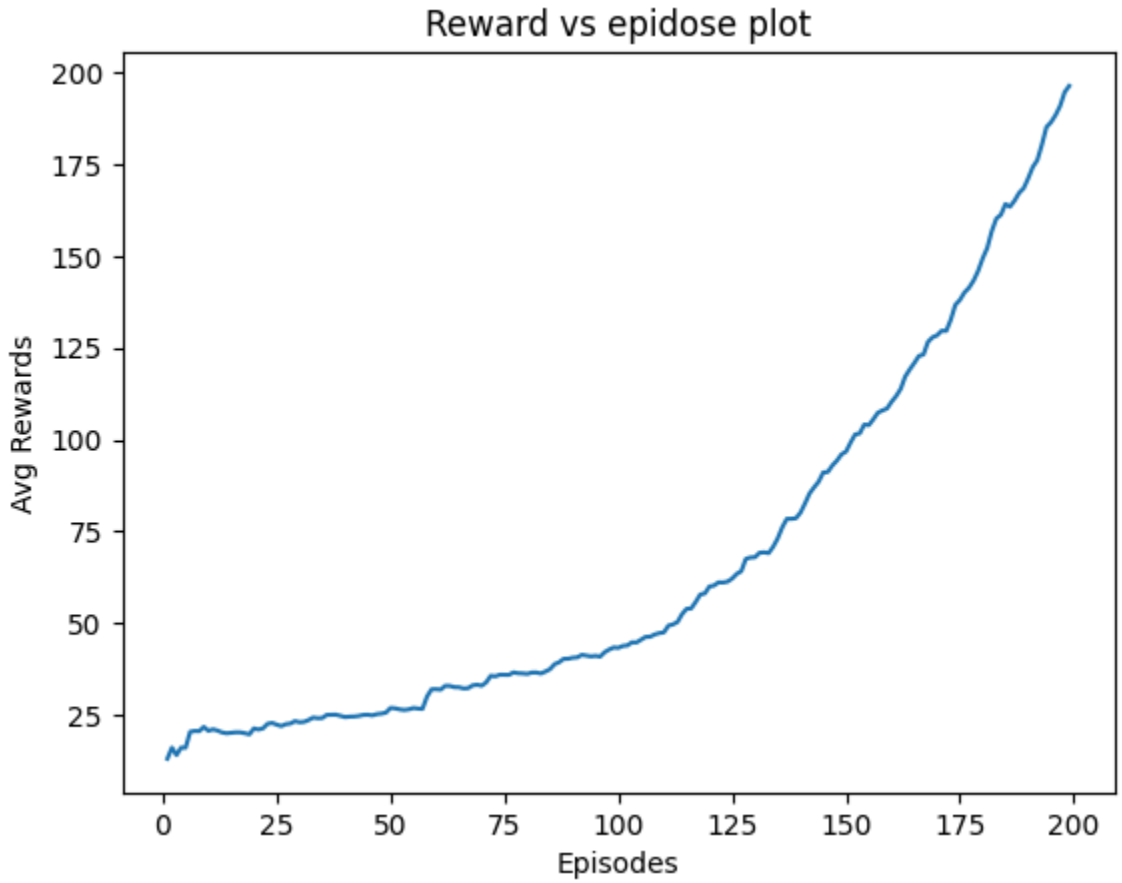


Hyperparameter set-3 :

- We reduced the batch size, but it increased the steps to solve the environment.

Buffer Size = 1e5
Batch Size =128
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

```
Environment solved in 199 episodes!     Average Score: 196.44
```

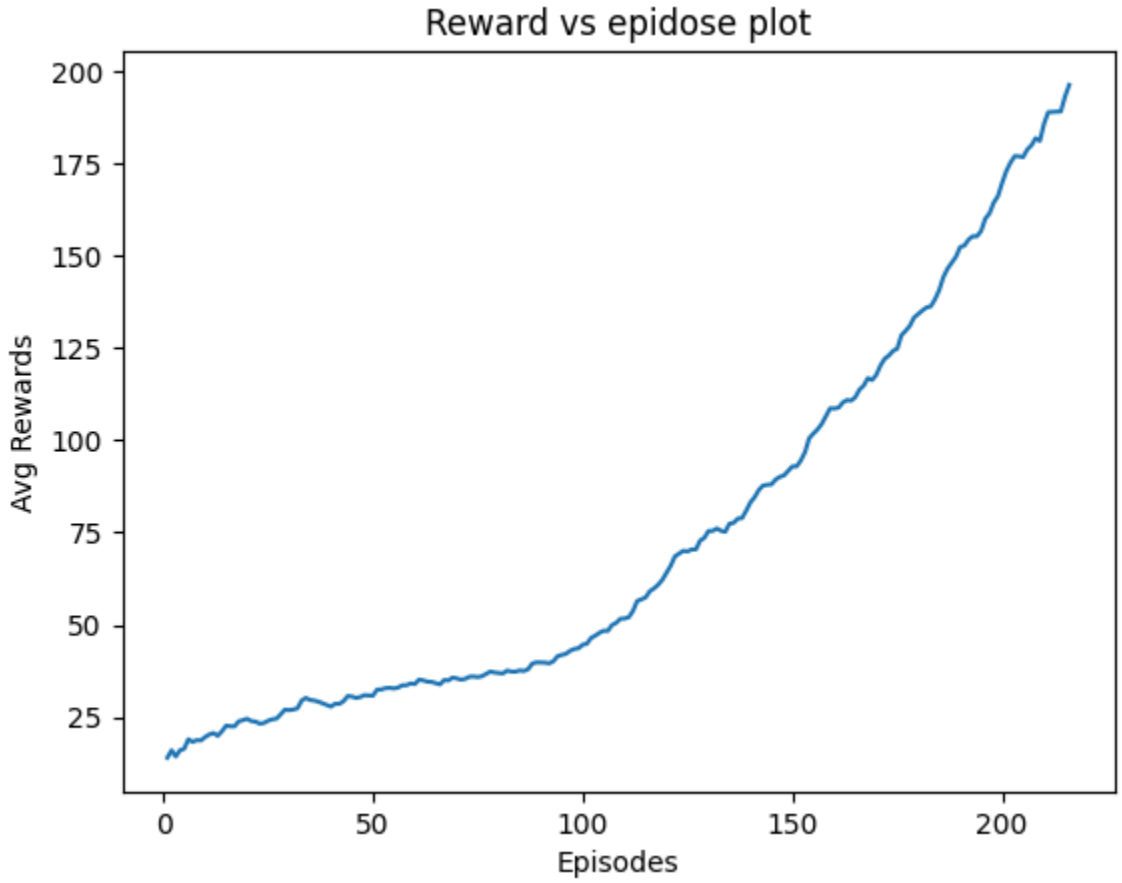


Hyperparameter set-4 :

- We increased Buffer Size, but that doesn't seem to affect the learning, but steps are increased to solve the environment.

Buffer Size = 1e6
Batch Size =128
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

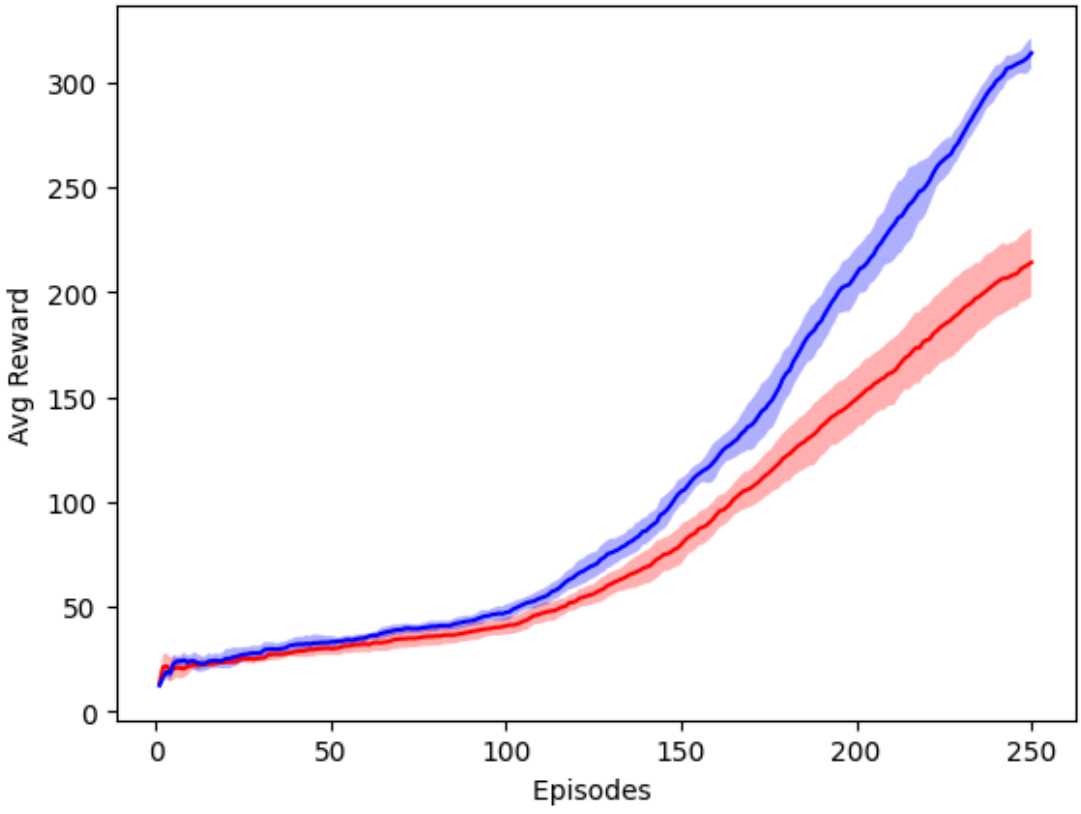
```
Environment solved in 216 episodes!     Average Score: 196.18
```



Comparison between Type-1 and Type-2 :

Observations

- With an optimal set of parameters we tried to compare the both Type-1 (average advantages subtraction) and Type-2 (maximum advantages subtraction) equations mentioned in the question.
- We set episodes to 250 max as all the variants of the hyperparameter are able to solve the environment within 250 episodes.
- Type -2 out-performs the Type-1, the reward collected by Type-1 is around 200 points at the end of 250 episodes while Type-2 manages to collect more than 300 points. This is because of the nature of the equation itself as it is advantages over max action.
- Type-2 variance is low as maximum advantages subtraction gives more sharper q value estimation and consistency. That's why Type-2 graphs have low variance and sharp raising than Type-1.
- Type-1 tends to produce smoother Q-value estimations than Type-2, as the difference between the advantage of each action and the average is more evenly distributed across actions.



Red -> Type - 1
Blue -> Type - 2
On Cartpole

Dueling-DQN on Acrobot Environment

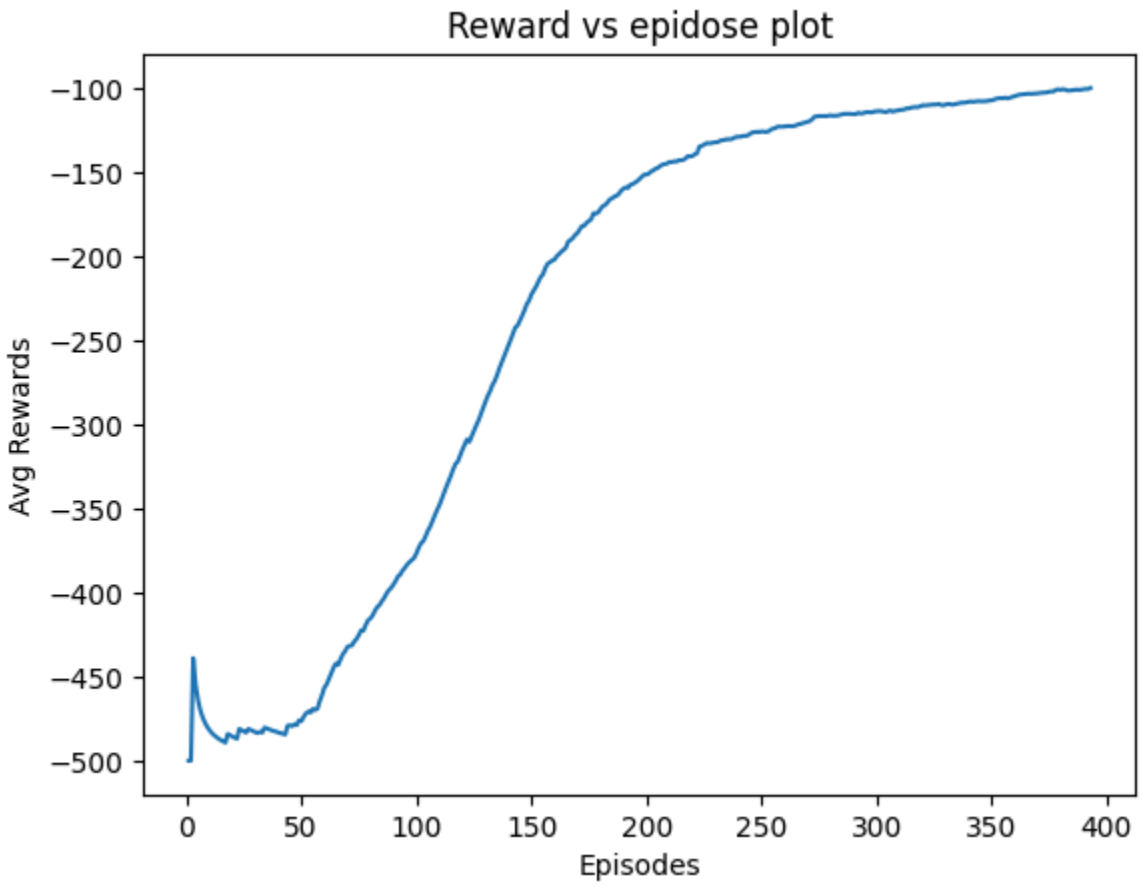
hyperparameter tuning on type -1:

- We performed a bunch of hyperparameters to solve the environment for getting a reward of -100.0 , some of them are listed below.

Hyperparameter set-1 :

Buffer Size = 1e5
Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

```
Environment solved in 393 episodes!  Average Score: -99.63
```

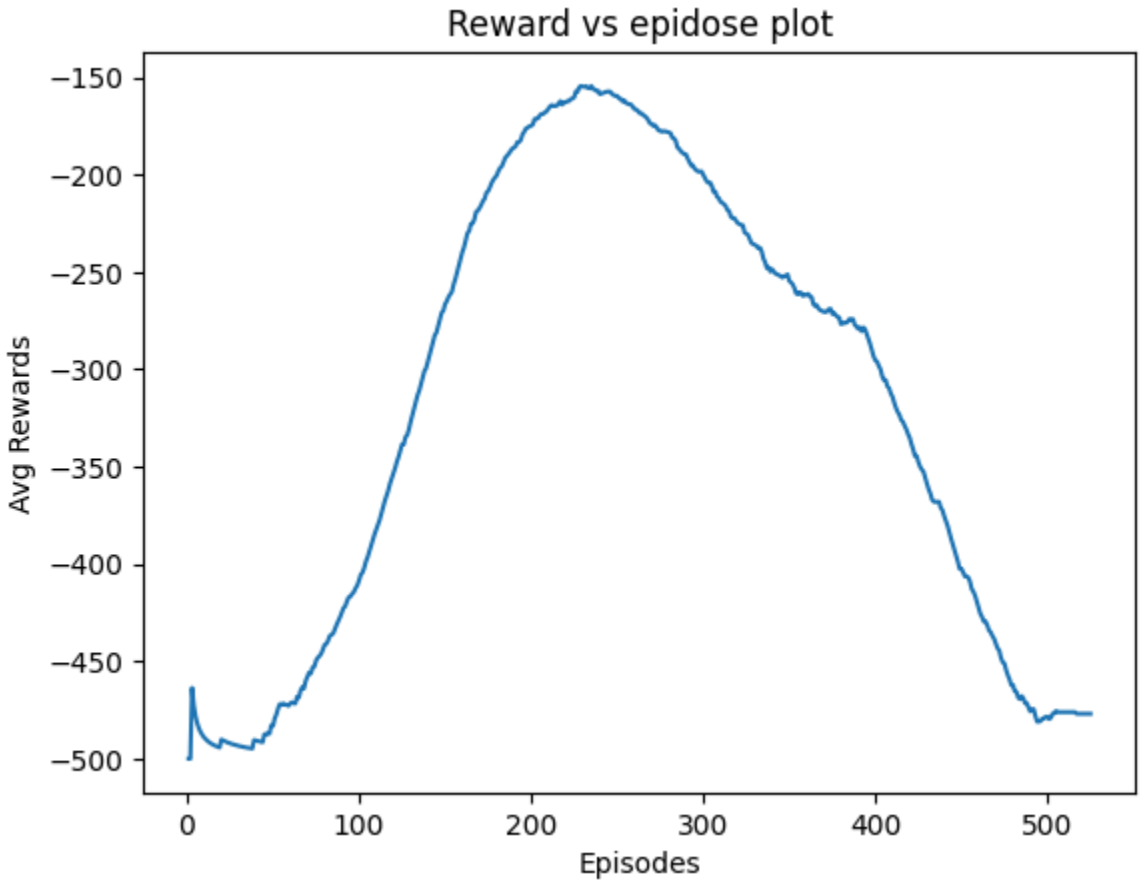


Hyperparameter set-2 :

- We tried to increase the learning rate, resulting in not converging on the environment. This may be due to the very sparse reward of the environment itself.

Buffer Size = 1e5
Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-3
Update every = 20

```
Environment not solved in 525   Average Score: -476.93
```



Hyperparameter set-3 :

- We reduced the batch size, standard deep learning hyperparameter tuning process, able to solve the environment in fewer steps.

Buffer Size = 1e5
Batch Size = 128
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

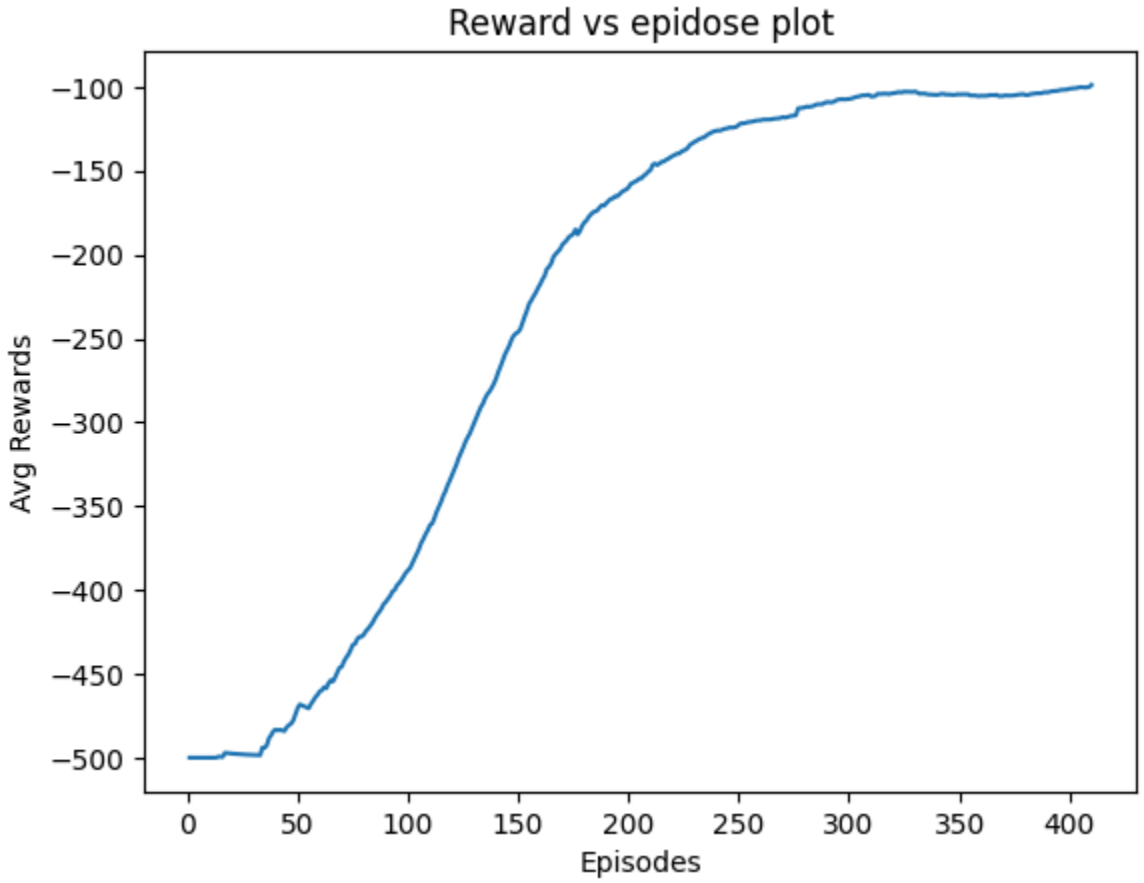
```
Environment  solved in 383 Average Score: -100.0
```

Hyperparameter set-4 :

- We increased Buffer size to see the effect, but that doesn't have much effect on solving the environment.

Buffer Size = 1e6
Batch Size = 128
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

```
Environment  solved in 410.     Average Score: -98.89
```



hyperparameter tuning on type -2:

- We performed a bunch of hyperparameters to solve the environment for getting a reward of -100.0 , some of them are listed below.

Hyperparameter set-1 :

Buffer Size = 1e5
Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

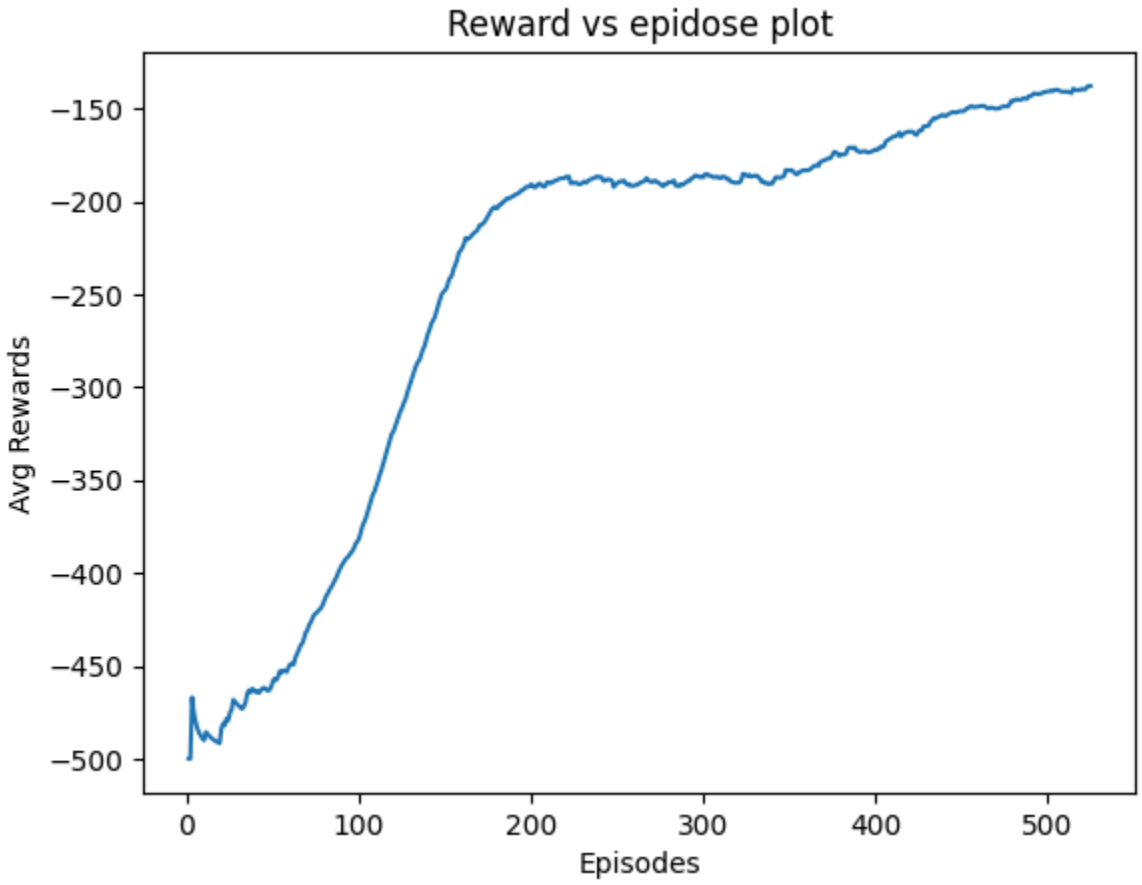
```
Environment solved in  690  episodes! Average Score: -99.96
```

Hyperparameter set-2 :

- We increased the learning rate that resulted in solving the environment in fewer steps.

Buffer Size = 1e5
Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-3
Update every = 20

```
Environment not  solved in  525  episodes! Average Score: -137.860
```



Hyperparameter set-3 :

- We have reduced the batch size but here that doesn't help much in solving environment.

Buffer Size = 1e5
Batch Size = 128
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

```
Environment not  solved in  600  episodes! Average Score: -113.89
```

Hyperparameter set-4 :

- We increased Buffer size to see the effect, but that doesn't have much effect on solving the environment.

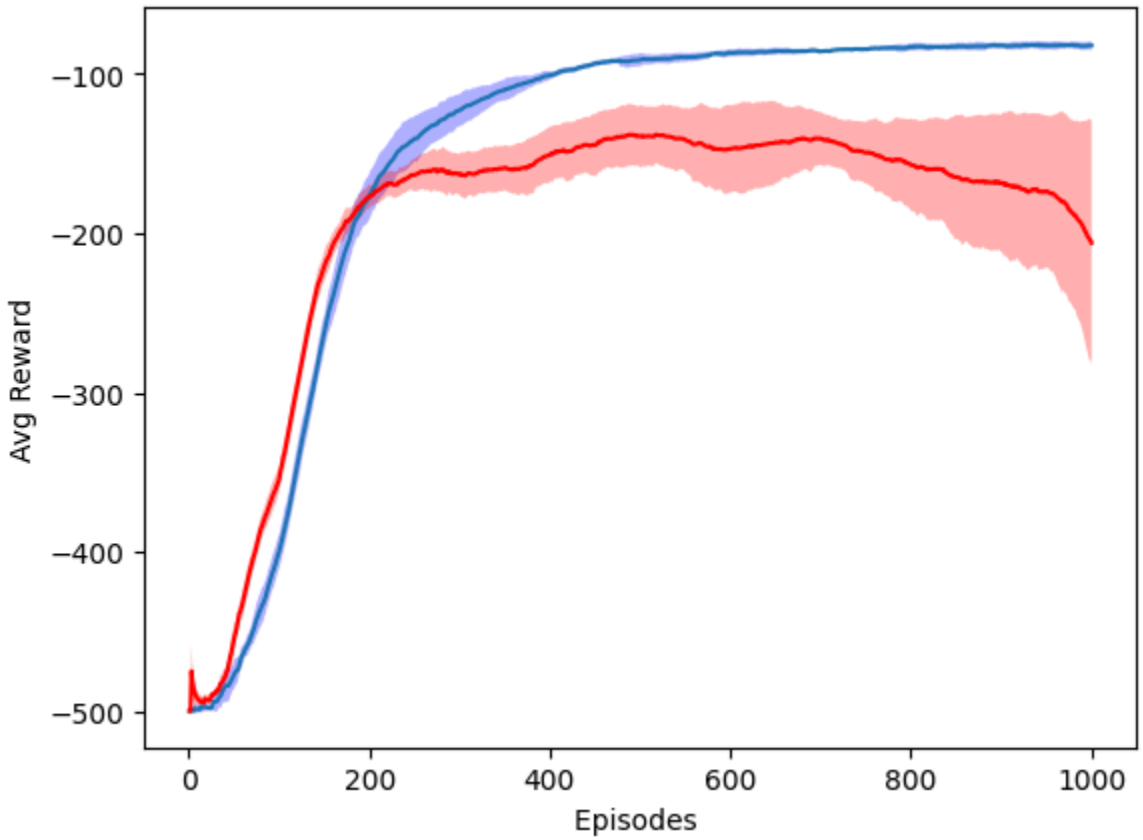
Buffer Size = 1e6
Batch Size = 128
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

```
Environment not  solved in  600  episodes! Average Score: -114.41
```

Comparison between Type-1 and Type-2 :

Observations :

- With an optimal set of parameters we tried to compare the both Type-1 (average advantages subtraction) and Type-2 (maximum advantages subtraction) equations mentioned in the question.
- We set episodes to 1000 max as all the variants of the hyperparameter are able to solve the environment moreover by 600 or 700 episodes.
- Type -2 out-performs the Type-1 in reward collection consistently.
- Type-2 variance is low as maximum advantages subtraction gives more sharper q value estimation and consistency. Type-1 is having higher variance than usual as it's a sparse environment to train.



Red -> Type - 1
Blue -> Type - 2
On Acrobot

Monte-Carlo Reinforce, Cartpole Environment

hyperparameter tuning on w/o baseline :

- We performed a bunch of hyperparameters to solve the environment for reward 195.0, some of them are listed below.

Hyperparameter set-1 :

Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

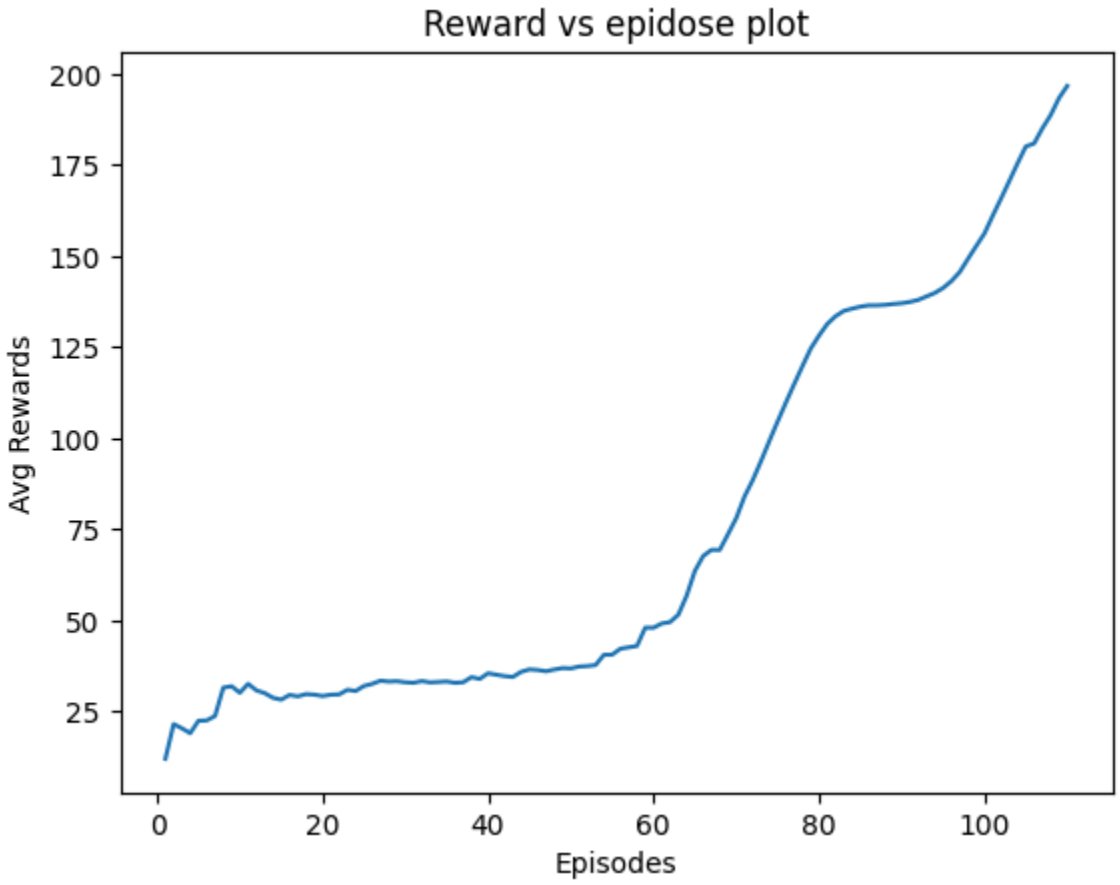
```
Environment solved in 303 episodes!  Average Score: 196.34
```

Hyperparameter set-2 :

- We decreased the learning rate and the environment got solved in fewer steps than previous.

Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-3
Update every = 20

```
Environment  solved in 110 episodes!  Average Score: 196.71
```



Hyperparameter set-3 :

- We decreased the batch size to see the effect on solving the environment.

Batch Size = 128
Gamma = 0.99
Learning Rate = 5e-3
Update every = 20

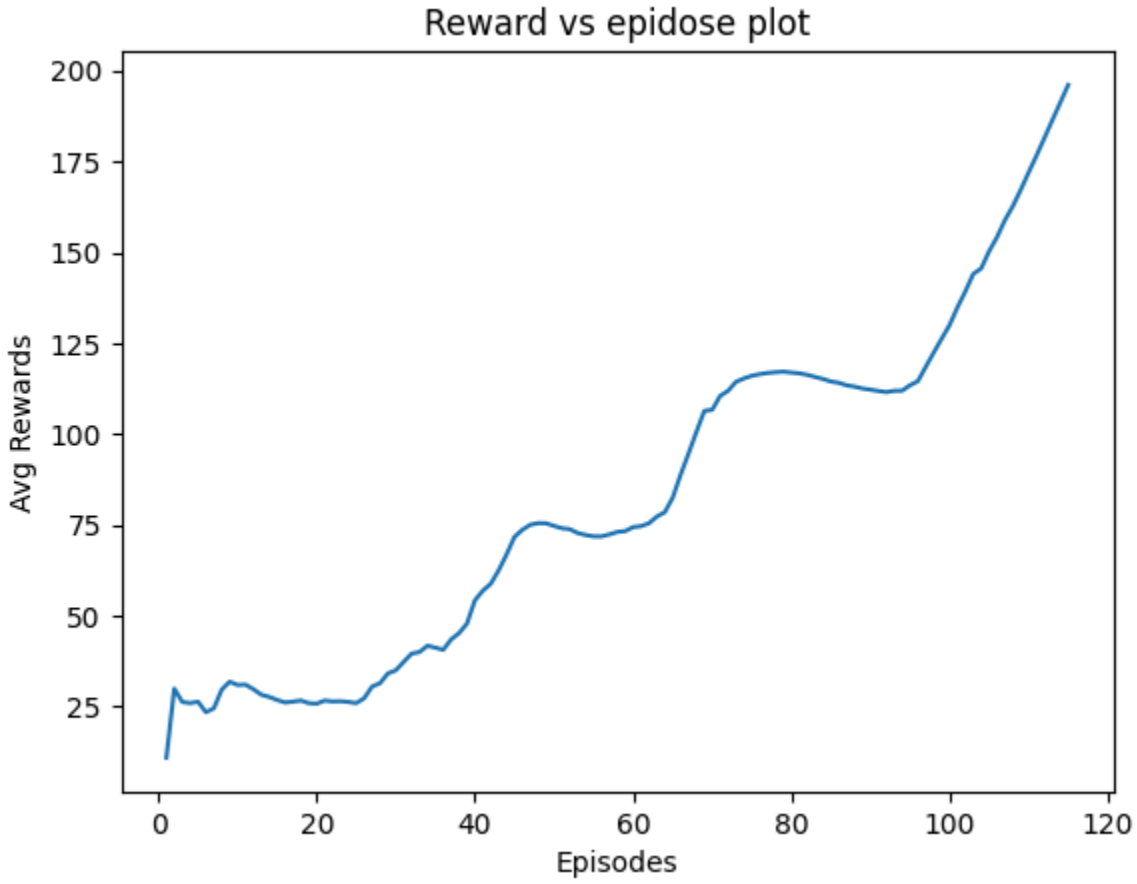
```
Environment  solved in 202 episodes!  Average Score: 196.05
```

Hyperparameter set-4 :

- Further reducing the batch size and increasing learning rate to see the learning changes in the environment.

Batch Size = 64
Gamma = 0.99
Learning Rate = 5e-3
Update every = 20

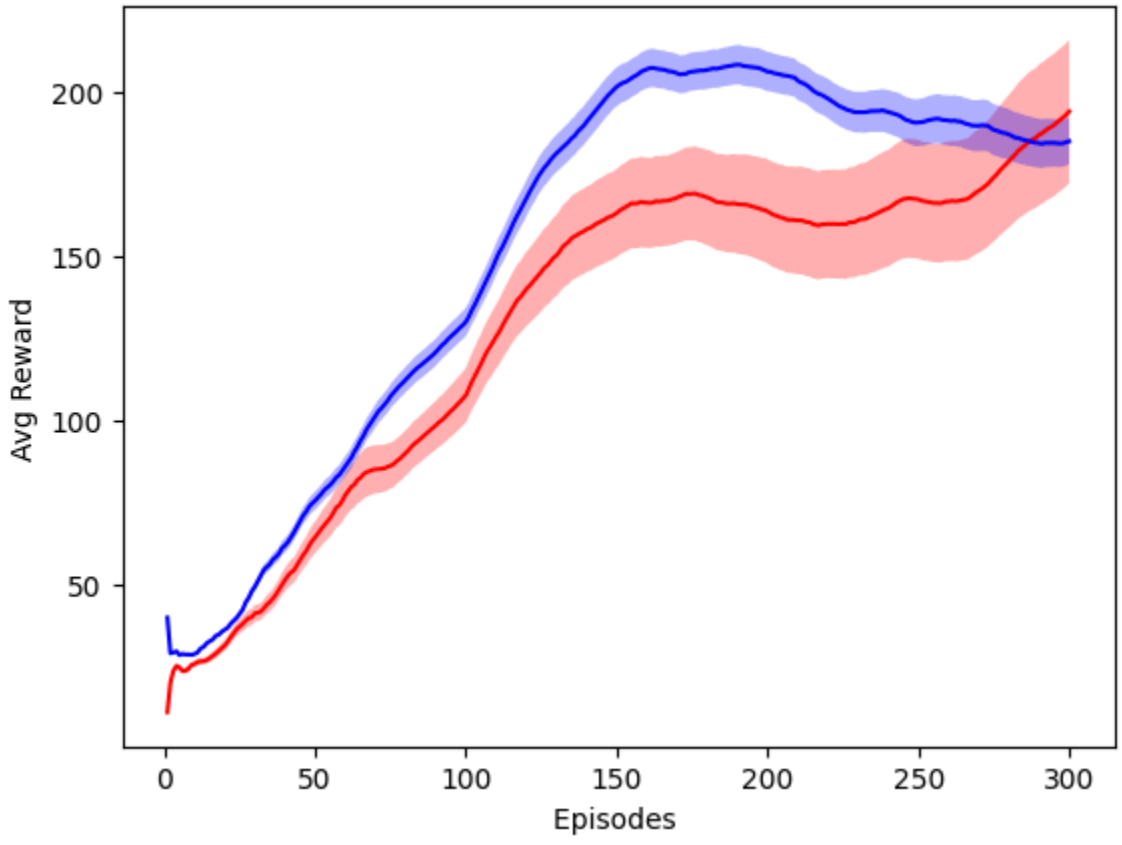
```
Environment solved in 115 episodes!  Average Score: 196.03
```



Comparison between without baseline and with baseline :

Observations :

- With an optimal set of parameters we tried to compare the both without baseline subtraction and with baseline subtraction equations mentioned in the question.
- Reinforce with Baseline has lower variance, leading to more stable and faster convergence.Reinforce with Baseline often converges more quickly and reliably compared to Reinforce without Baseline due to the reduced variance in the updates.
- Reinforce with Baseline is typically more robust to noise and randomness in the environment due to its lower variance.
- Reinforce without Baseline sometimes exhibited more erratic behavior during training, as it is more sensitive to variance in the estimated returns while varying batch sizes.



Red -> w/o Baseline
Blue -> with Baseline
On Cartpole

Monte-Carlo Reinforce, Acrobot Environment

hyperparameter tuning on w/o baseline :

- We performed a bunch of hyperparameters to solve the environment for reward -100.0, some of them are listed below.

Hyperparameter set-1 :

Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

```
Environment not solved in 700 episodes!    Average Score: -111.0
```

Hyperparameter set-2 :

- We increased the learning rate, decreased batch size and were able to solve the environment.

Batch Size = 32
Gamma = 0.99
Learning Rate = 5e-3
Update every = 20

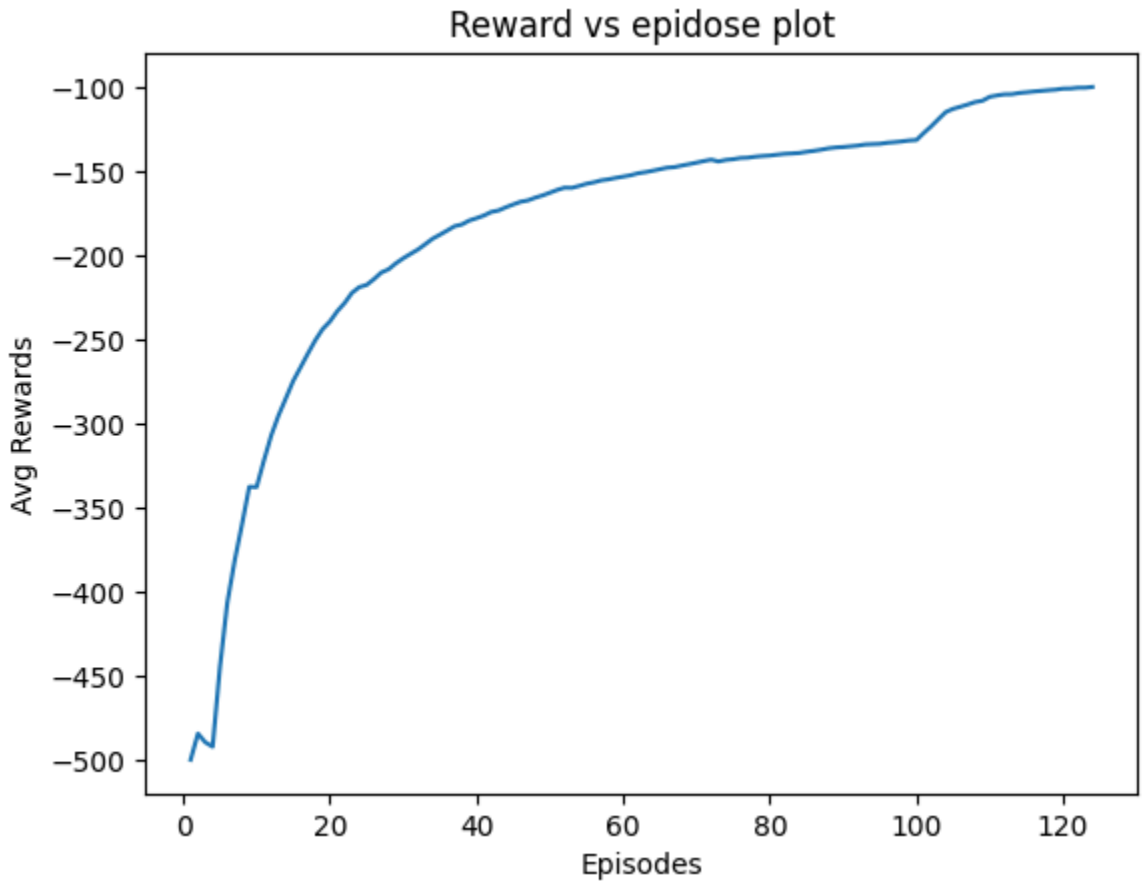
```
Environment  solved in 268 episodes!  Average Score: -100.0
```

Hyperparameter set-3 :

- We have reduced the batch size and it took fewer steps to solve the environment.

Batch Size = 128
Gamma = 0.99
Learning Rate = 5e-3
Update every = 20

```
Environment  solved in 124 episodes!  Average Score: -99.78
```



Hyperparameter set-4 :

Batch Size = 64
Gamma = 0.99
Learning Rate = 5e-3
Update every = 10

```
Environment solved in 257 episodes!  Average Score: -99.78
```

hyperparameter tuning on with baseline :

- We performed a bunch of hyperparameters to solve the environment for reward -100.0, some of them are listed below.

Hyperparameter set-1 :

Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-4
Update every = 20

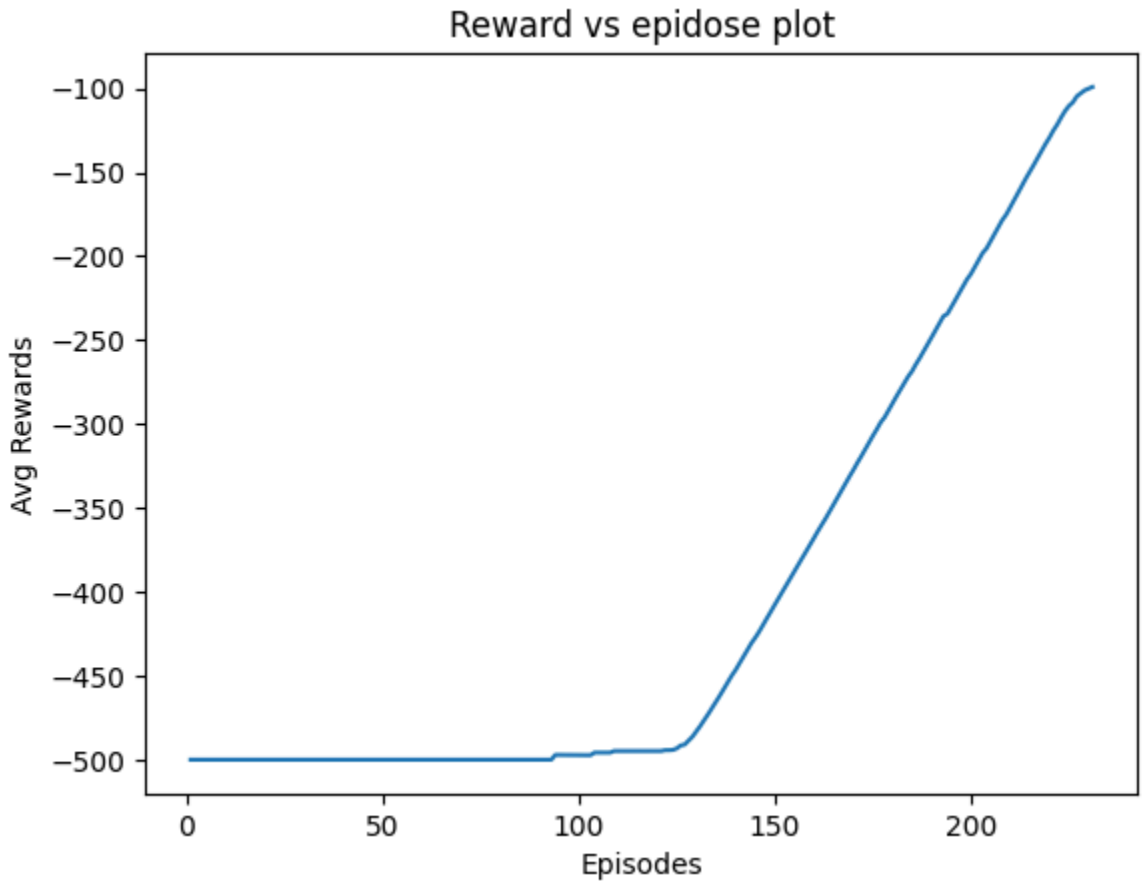
```
Environment not solved in 703 episodes!    Average Score: -106.0
```

Hyperparameter set-2 :

- We increased the learning rate and were able to solve the environment.

Batch Size = 256
Gamma = 0.99
Learning Rate = 5e-3
Update every = 20

```
Environment  solved in 231 episodes!  Average Score: -100.0
```

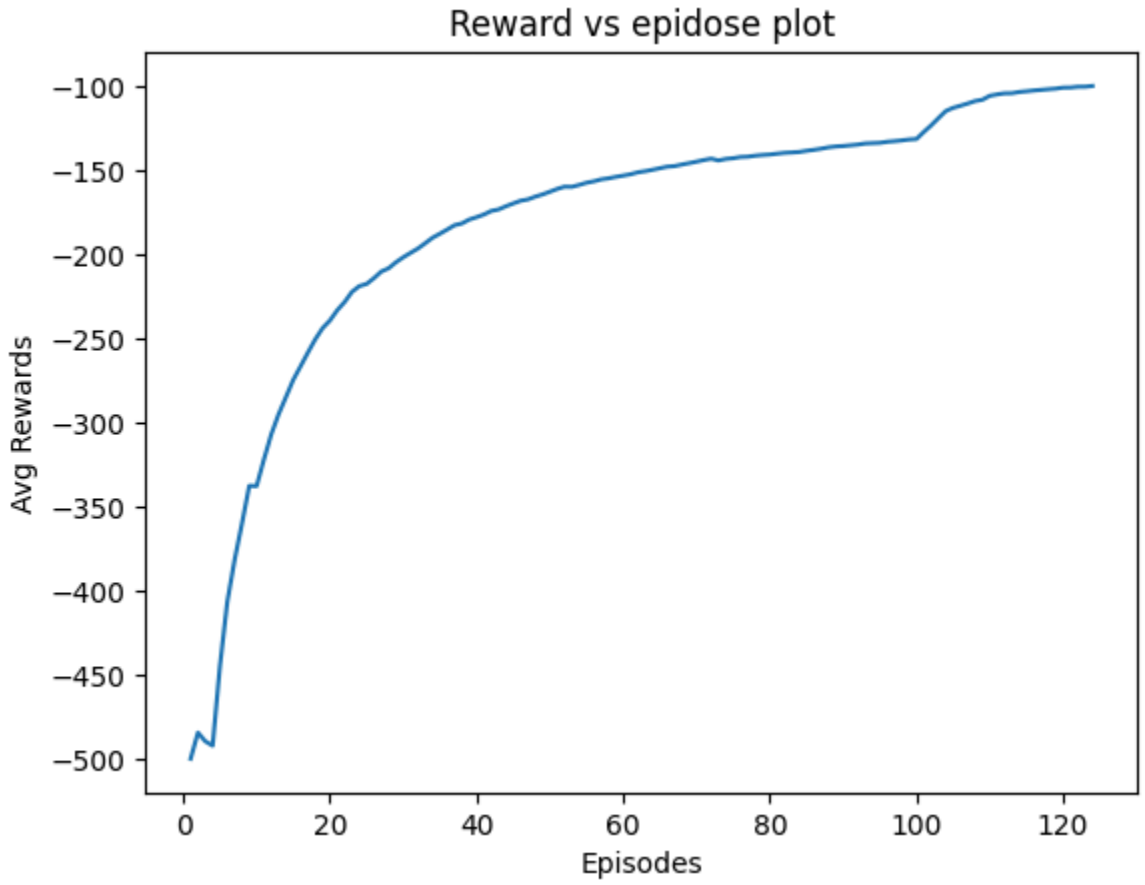


Hyperparameter set-3 :

- We have reduced the batch size and it took fewer steps to solve the environment.

Batch Size = 128
Gamma = 0.99
Learning Rate = 5e-3
Update every = 20

```
Environment  solved in 121 episodes!  Average Score: -99.95
```



Hyperparameter set-4 :

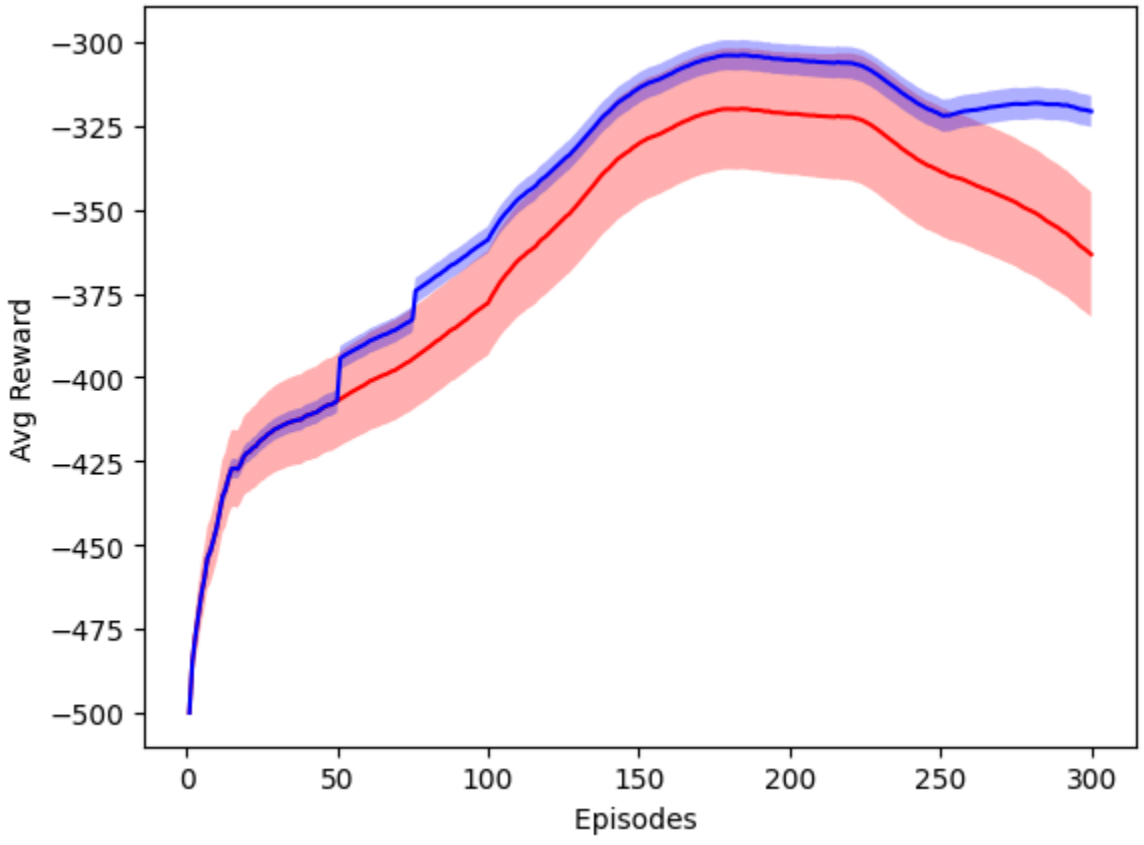
Batch Size = 64
Gamma = 0.99
Learning Rate = 5e-3
Update every = 10

```
Environment solved in 259 episodes!  Average Score: -98.86
```

Comparison between without baseline and with baseline :

Observations :

- With an optimal set of parameters we tried to compare the both without baseline subtraction and with baseline subtraction equations mentioned in the question.
- Reinforce with a baseline exhibits reduced variance, resulting in enhanced stability and quicker convergence. It reliably converges faster than Reinforce without a baseline due to the dampened variance in updates.
- Moreover, Reinforce with a baseline tends to display greater resilience to environmental noise and randomness due to its diminished variance. In contrast, Reinforce without a baseline may occasionally demonstrate erratic training behavior, especially when batch sizes fluctuate, owing to its heightened sensitivity to variance in estimated returns.



Red -> w/o Baseline
Blue -> with Baseline
On Cartpole

Snippets of code -

- We have used a modified version of tutorials code for our case.

Dueling DQN -

Network -

```
class QNetworkDueling(nn.Module):

    def __init__(self, state_size, action_size, seed, fc1_units=128, fc2_units=64):
        """Initialize parameters and build model."""
        super(QNetworkDueling, self).__init__()
        self.seed = torch.manual_seed(seed)

        # Common feature extraction layers
        self.fc1 = nn.Linear(state_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)

        # State value function
        self.fc3_val = nn.Linear(fc2_units, 1)

        # Advantage function
        self.fc3_adv = nn.Linear(fc2_units, action_size)

    def forward(self, state):
        """Build a network that maps state -> action values."""
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))

        # State value calculation
        val = self.fc3_val(x)

        # Advantage value calculation
        adv = self.fc3_adv(x)

        # Combine state value and advantage to get Q-value
        return val + adv - adv.mean()
```

Type -1 code :

```python
def learn(self, experiences, gamma):
    states, actions, rewards, next_states, dones = experiences

    # Get mean predicted Q values (for next states) from target model
    # Q_targets_next = self.qnetwork_target(next_states).detach().max(1)[0].unsqueeze(1)
    Q_targets_next = self.qnetwork_target(next_states).detach().mean(dim=1).unsqueeze(1)

    # Compute Q targets for current states
    Q_targets = rewards + (gamma * Q_targets_next * (1 - dones))

    # Get expected Q values from local model
    Q_expected = self.qnetwork_local(states).gather(1, actions)

    # Compute loss
    loss = F.mse_loss(Q_expected, Q_targets)

    # Minimize the loss
    self.optimizer.zero_grad()
    loss.backward()

    # Gradient Clipping
    for param in self.qnetwork_local.parameters():
        param.grad.data.clamp_(-1, 1)

    self.optimizer.step()
```

Type - 2 code :

```python
def learn(self, experiences, gamma):
    states, actions, rewards, next_states, dones = experiences

    # Get max predicted Q values (for next states) from target model
    Q_targets_next = self.qnetwork_target(next_states).detach().max(1)[0].unsqueeze(1)

    # Compute Q targets for current states
    Q_targets = rewards + (gamma * Q_targets_next * (1 - dones))

    # Get expected Q values from local model
    Q_expected = self.qnetwork_local(states).gather(1, actions)

    # Compute loss
    loss = F.mse_loss(Q_expected, Q_targets)

    # Minimize the loss
    self.optimizer.zero_grad()
    loss.backward()

    # Gradient Clipping
    for param in self.qnetwork_local.parameters():
        param.grad.data.clamp_(-1, 1)

    self.optimizer.step()
```

Monte Carlo Reinforce -

Without Baseline -

```
def run():
  class network_nn(nn.Module):

      def __init__(self, state_size, action_size, seed, fc1_units=128, fc2_units=64):
          """Initialize parameters and build model."""
          super(network_nn, self).__init__()
          self.seed = torch.manual_seed(seed)

          # Common feature extraction layers
          self.fc1 = nn.Linear(state_size, fc1_units)
          self.fc2 = nn.Linear(fc1_units, fc2_units)
          self.fc3 = nn.Linear(fc2_units, action_size)


      def forward(self, state):
          """Build a network that maps state -> action values."""
          x = F.relu(self.fc1(state))
          x = F.relu(self.fc2(x))
          x = self.fc3(x)

          return x
```

With Baseline -

```python
class network_nn(nn.Module):

    def __init__(self, state_size, action_size, seed, fc1_units=128, fc2_units=64):
        """Initialize parameters and build model."""
        super(network_nn, self).__init__()
        self.seed = torch.manual_seed(seed)

        # Common feature extraction layers
        self.fc1 = nn.Linear(state_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        # State value function
        self.fc3_val = nn.Linear(fc2_units, 1)

        # baseline function
        self.fc3_baseline = nn.Linear(fc2_units, action_size)


    def forward(self, state):
        """Build a network that maps state -> action values."""
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))

        # State value calculation
        val = self.fc3_val(x)

        # baseline value calculation
        baseline = self.fc3_baseline(x)

        # Combine state value and baselinee
        return val + baseline - baseline.mean()
```

Github Link - https://github.com/routb68/reinforcement_learning_assignment_2

THE END