DQN

```
'''
Installing packages for rendering the game on Colab
'''

!pip install gym pyvirtualdisplay > /dev/null 2>&1
!apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1
!apt-get update > /dev/null 2>&1
!apt-get install cmake > /dev/null 2>&1
!pip install --upgrade setuptools 2>&1
!pip install ez_setup > /dev/null 2>&1
!pip install gym[atari] > /dev/null 2>&1
!pip install git+https://github.com/tensorflow/docs > /dev/null 2>&1
```

```
     Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (61.2.0)
```

```
!pip install tensorflow-gpu
```

```
import numpy as np
from scipy.special import softmax
import random
import torch
import torch.nn as nn
import torch.nn.functional as F
from collections import namedtuple, deque
import torch.optim as optim
import datetime
import gym
from gym.wrappers import Monitor
import glob
import io
import base64
import matplotlib.pyplot as plt
from IPython.display import HTML
from pyvirtualdisplay import Display
import tensorflow as tf
from IPython import display as ipythondisplay
from PIL import Image
import tensorflow_probability as tfp
```

Show code

Show code

Show code

Show code

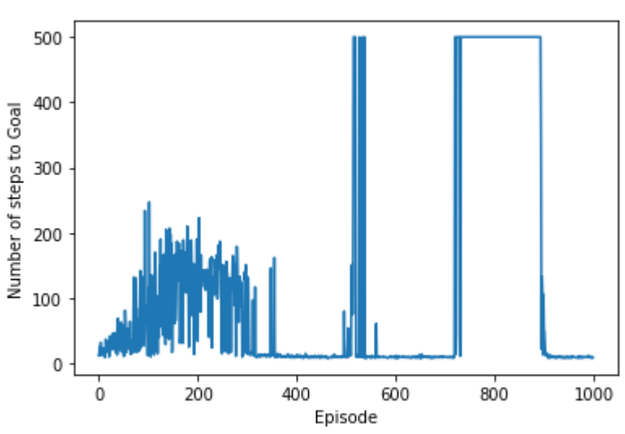
CartPole-v1

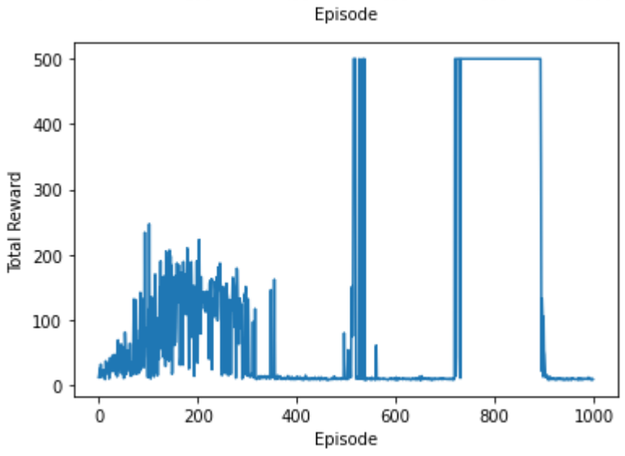
```
env = gym.make('CartPole-v1')
env.seed(0)

state_shape = env.observation_space.shape[0]
action_shape = env.action_space.n
no_of_actions = env.action_space.n

print(state_shape)
print(no_of_actions)
print(env.action_space.sample())
print("----")

state = env.reset()
''' This returns the initial state (when environment is reset) '''

print(state)
print("----")

action = env.action_space.sample()

print(action)
print("----")

next_state, reward, done, info = env.step(action)
''' env.step is used to calculate new state and obtain reward based on old state and action taken  '''

print(next_state)
print(reward)
print(done)
print(info)
print("----")
```

```
4
2
1
----
[-0.04456399  0.04653909  0.01326909 -0.02099827]
----
1
----
[-0.04363321  0.24146826  0.01284913 -0.30946528]
1.0
False
{}
----
```

Variation 1

```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.99
LR = 5e-4
UPDATE_EVERY = 20
EPISODES = 1000
LIM = 1
FC1 = 128
FC2 = 64


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```






```
avg_reward_list = []
idx = -1
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  temp = np.mean(reward_list[i:i + 100])
  avg_reward_list.append(temp)
  if idx == -1 and temp > 475:
    idx = i


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Episodes taken to solve the environment: ', idx + 100)
```

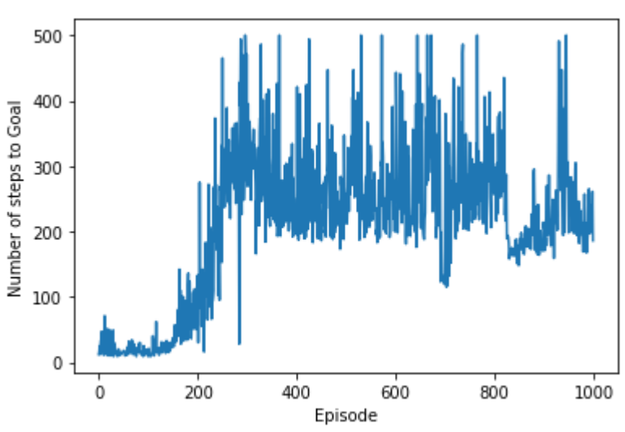
```
Episodes taken to solve the environment:  818
```

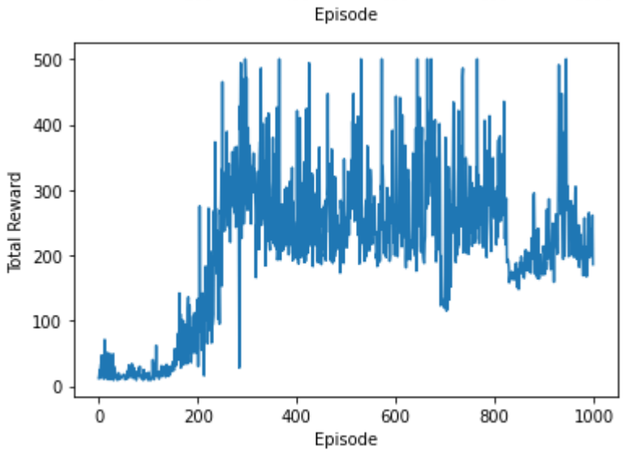

```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  121699.48500000002
```

Variation 2

```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.95
LR = 5e-5
UPDATE_EVERY = 50
EPISODES = 1000
LIM = 0.7
FC1 = 128
FC2 = 128


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```






```
avg_reward_list = []
idx = -1
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  temp = np.mean(reward_list[i:i + 100])
  avg_reward_list.append(temp)
  if idx == -1 and temp > 475:
    idx = i


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Episodes taken to solve the environment: ', idx + 100)

     Episodes taken to solve the environment:  99
```

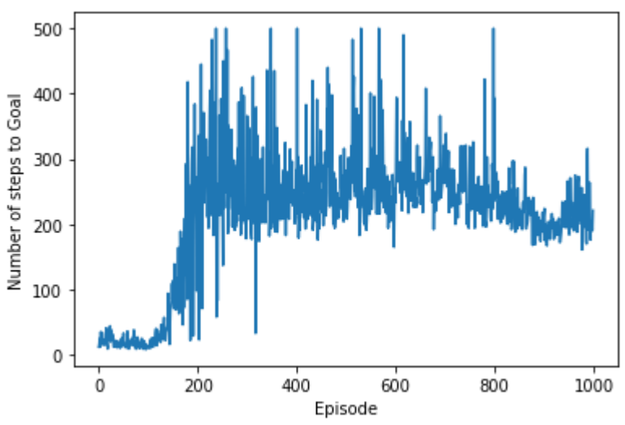


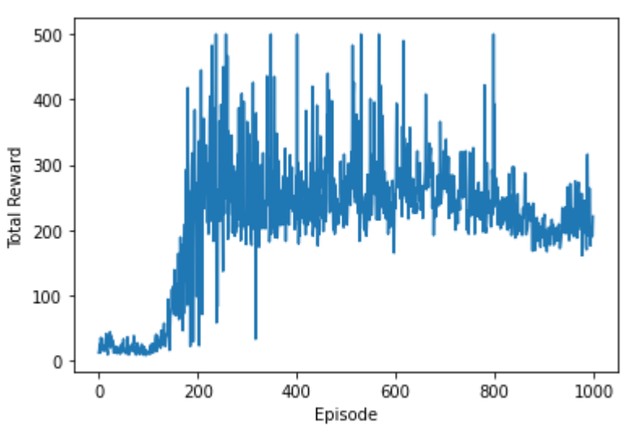
```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)

     Area under the curve:  197394.805
```

Variation 3

```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 96
GAMMA = 0.95
LR = 5e-5
UPDATE_EVERY = 50
EPISODES = 1000
LIM = 0.7
FC1 = 128
FC2 = 128

# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)
```

```
plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```






```
avg_reward_list = []
idx = -1
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  temp = np.mean(reward_list[i:i + 100])
  avg_reward_list.append(temp)
  if idx == -1 and temp > 475:
    idx = i


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Episodes taken to solve the environment: ', idx + 100)
```

```
Episodes taken to solve the environment:  99
```


```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

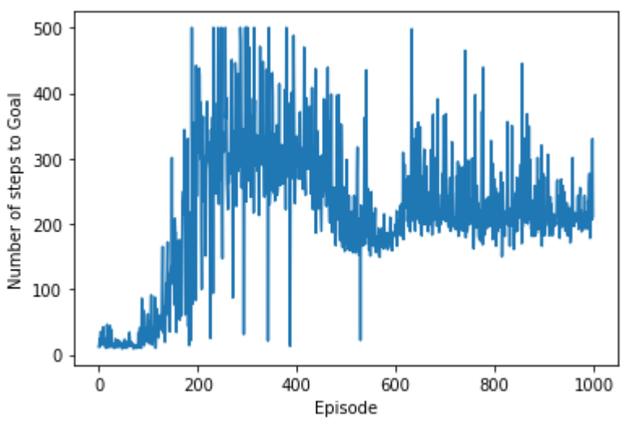
```
Area under the curve:  201565.0
```

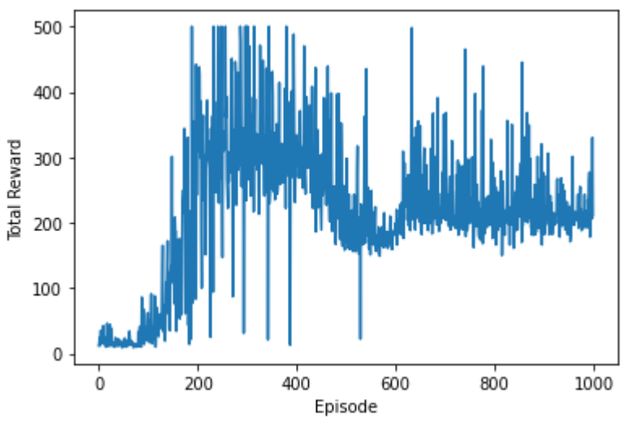
Variation 4

```
'''
```

```
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 96
GAMMA = 0.95
LR = 5e-5
UPDATE_EVERY = 50
EPISODES = 1000
LIM = 0.7
FC1 = 200
FC2 = 128

# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)
```

```
plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```






```
avg_reward_list = []
idx = -1
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  temp = np.mean(reward_list[i:i + 100])
  avg_reward_list.append(temp)
  if idx == -1 and temp > 475:
    idx = i


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Episodes taken to solve the environment: ', idx + 100)
```

```
Episodes taken to solve the environment:  99
```

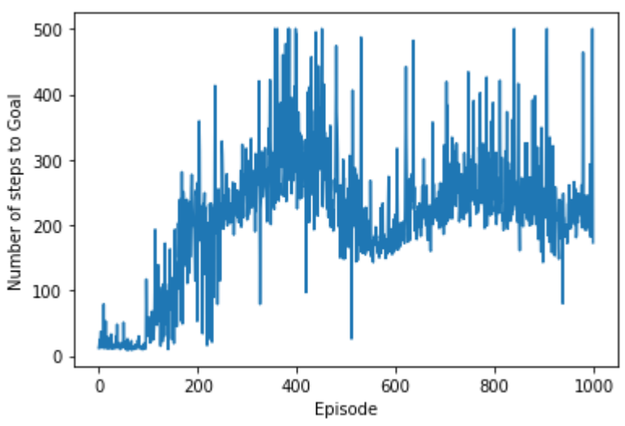

```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

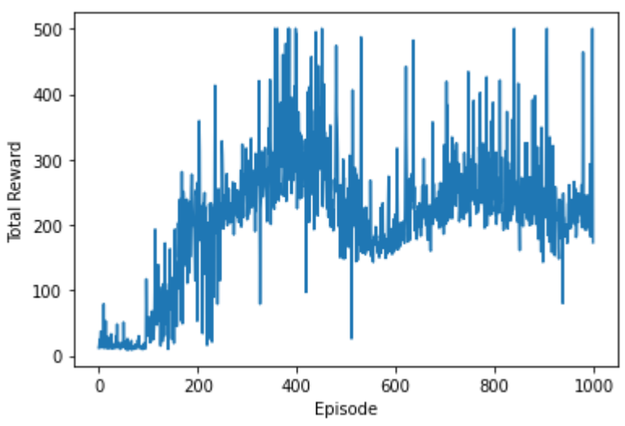
```
Area under the curve:  204873.025
```

Variation 5

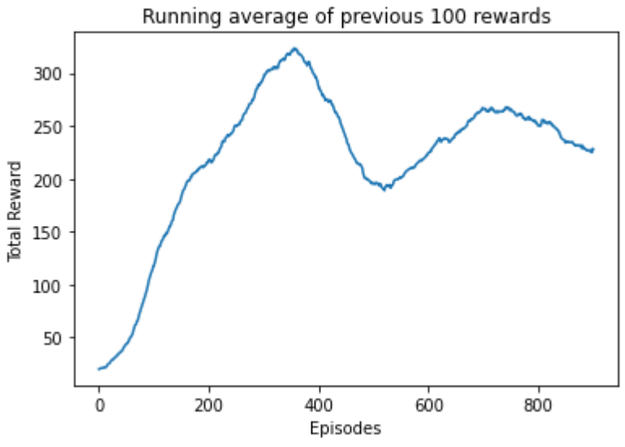
```
'''
Hyper parameters
```

```
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 110
GAMMA = 0.95
LR = 5e-5
UPDATE_EVERY = 50
EPISODES = 1000
LIM = 0.7
FC1 = 200
FC2 = 128

# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```






```
avg_reward_list = []
idx = -1
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  temp = np.mean(reward_list[i:i + 100])
  avg_reward_list.append(temp)
  if idx == -1 and temp > 475:
    idx = i


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Episodes taken to solve the environment: ', idx + 100)

     Episodes taken to solve the environment:  99
```




```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)

     Area under the curve:  198089.61500000002
```

Acrobot-v1

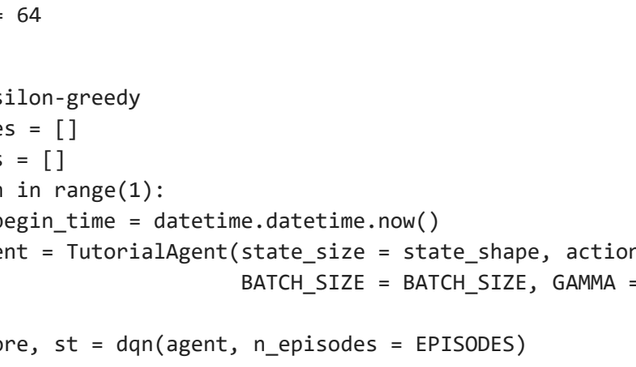
```
env = gym.make('Acrobot-v1')
env.seed(0)
```

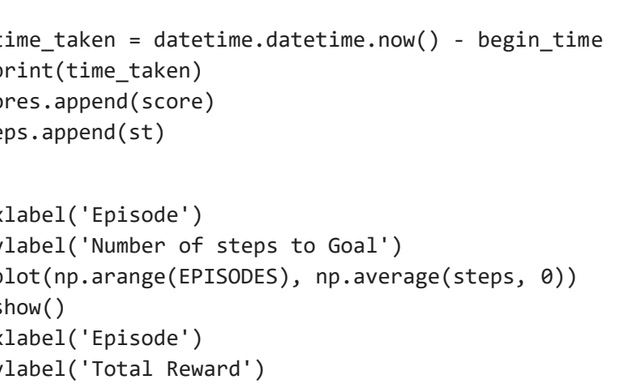
```
state_shape = env.observation_space.shape[0]
action_shape = env.action_space.n
no_of_actions = env.action_space.n

print(state_shape)
print(no_of_actions)
print(env.action_space.sample())
print("----")

state = env.reset()
''' This returns the initial state (when environment is reset) '''

print(state)
print("----")

action = env.action_space.sample()

print(action)
print("----")

next_state, reward, done, info = env.step(action)
''' env.step is used to calculate new state and obtain reward based on old state and action taken  '''

print(next_state)
print(reward)
print(done)
print(info)
print("----")
```

```
6
3
2
----
[ 0.99603073 -0.08901003  0.99567135  0.09294385  0.02653819 -0.04199653]
----
0
----
[ 0.99829918 -0.05829878  0.99945086  0.03313578  0.27308215 -0.54190945]
-1.0
False
{}
----
```

Variation 1

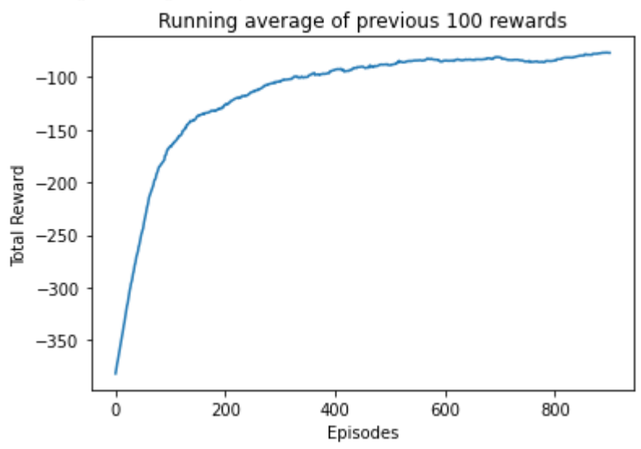
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.99
LR = 5e-4
UPDATE_EVERY = 20
EPISODES = 1000
LIM = 1
FC1 = 128
FC2 = 64


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```





```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))

### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -77.07
```

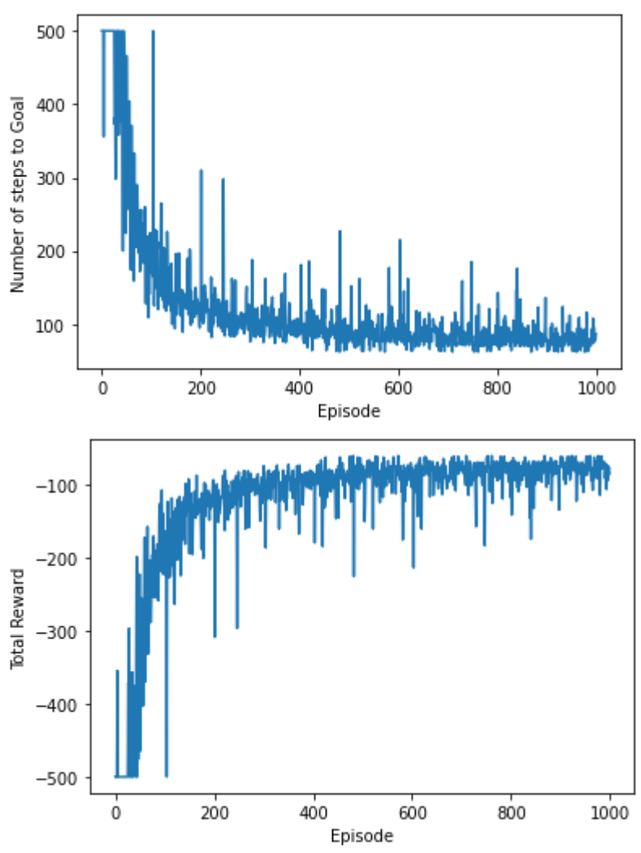



```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -103064.02000000002
```

Variation 2

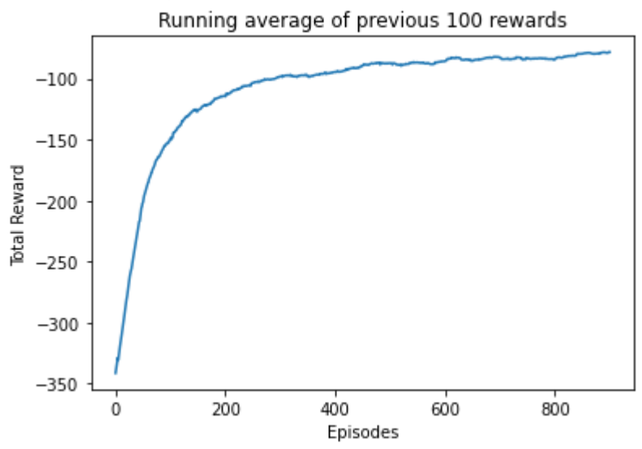
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.99
LR = 5e-4
UPDATE_EVERY = 50
EPISODES = 1000
LIM = 1
FC1 = 128
FC2 = 128


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```




```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))
```

```
### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -78.42
```

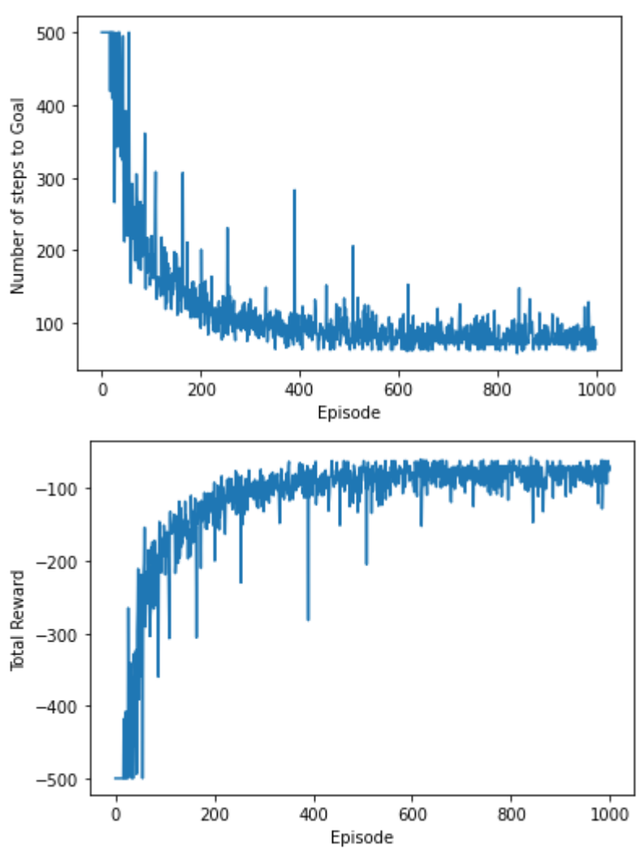



```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -97394.57
```

Variation 3

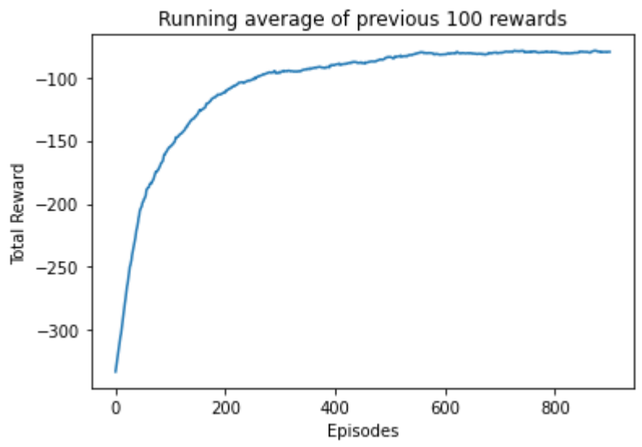
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 100
GAMMA = 0.99
LR = 5e-4
UPDATE_EVERY = 50
EPISODES = 1000
LIM = 1
FC1 = 128
FC2 = 128


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```




```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))


### Plot of total reward vs episode

plt.plot(avg_reward_list)
```

```
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -79.91
```

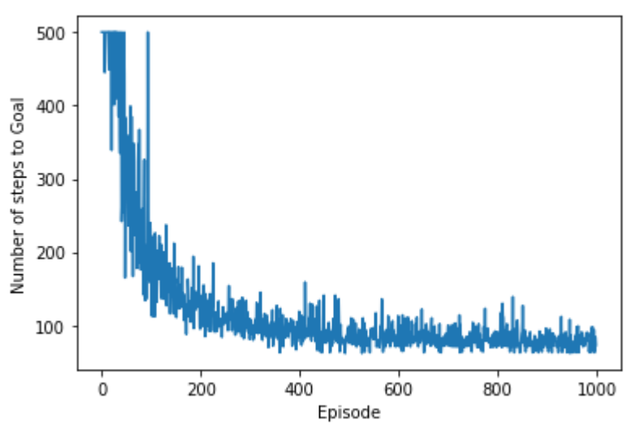


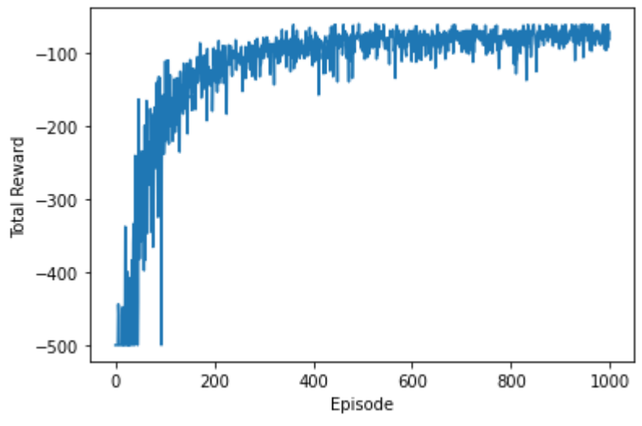

```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -95281.095
```

Variation 4

```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 100
GAMMA = 0.99
LR = 5e-4
UPDATE_EVERY = 50
EPISODES = 1000
LIM = 1
FC1 = 256
FC2 = 128


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)
  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```

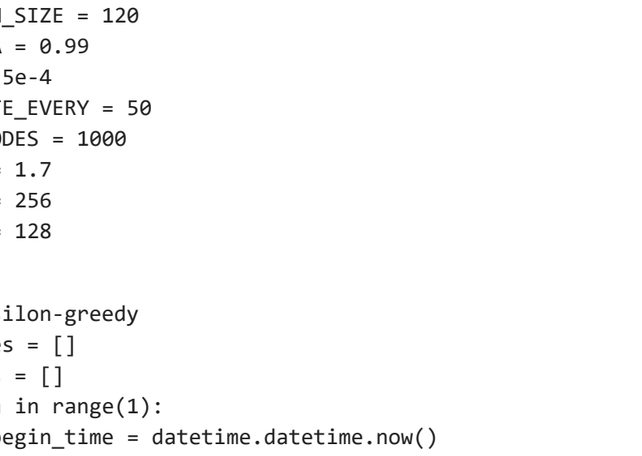





```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
```

```
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -77.51
```



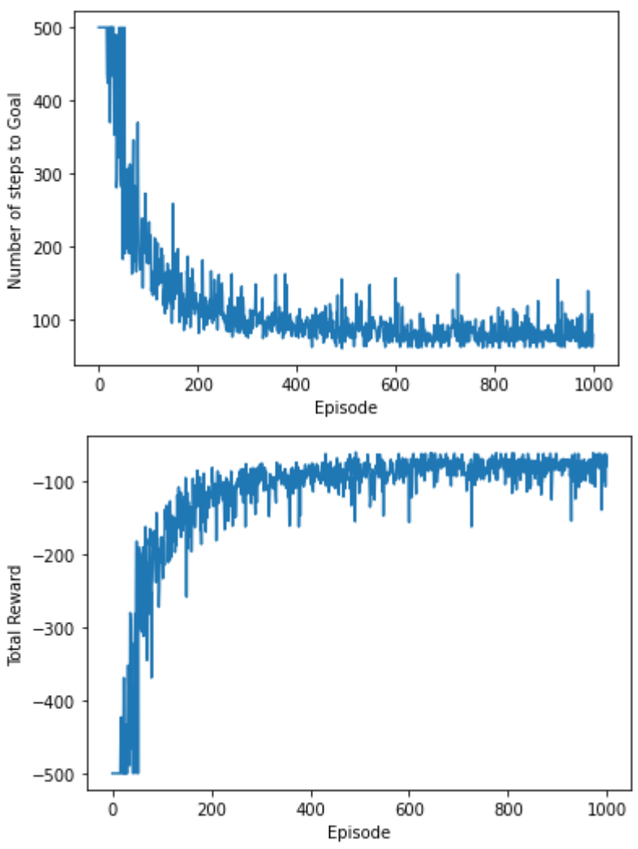

```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -95378.13500000001
```

Variation - 5

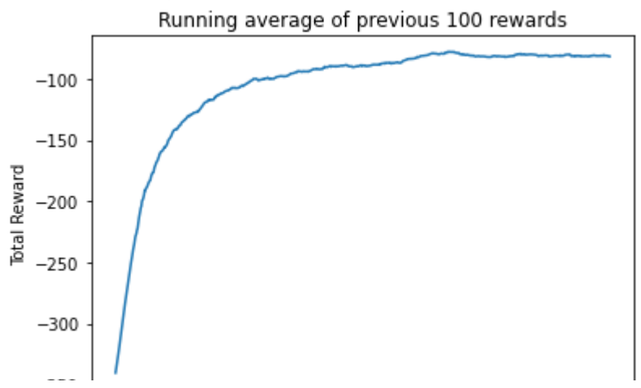
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 120
GAMMA = 0.99
LR = 5e-4
UPDATE_EVERY = 50
EPISODES = 1000
LIM = 1.7
FC1 = 256
FC2 = 128


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)
```

```
plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```




```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -81.54
```




```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -95216.11
```

MountainCar-v0

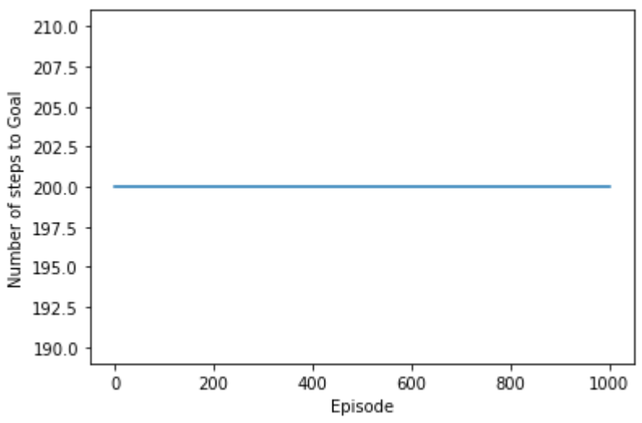
```
env = gym.make('MountainCar-v0')
env.seed(0)
state_shape = env.observation_space.shape[0]
action_shape = env.action_space.n
no_of_actions = env.action_space.n

print(state_shape)
print(no_of_actions)
print(env.action_space.sample())
print("----")

state = env.reset()
''' This returns the initial state (when environment is reset) '''

print(state)
print("----")

action = env.action_space.sample()

print(action)
print("----")

next_state, reward, done, info = env.step(action)
''' env.step is used to calculate new state and obtain reward based on old state and action taken  '''

print(next_state)
print(reward)
print(done)
print(info)
print("----")
```

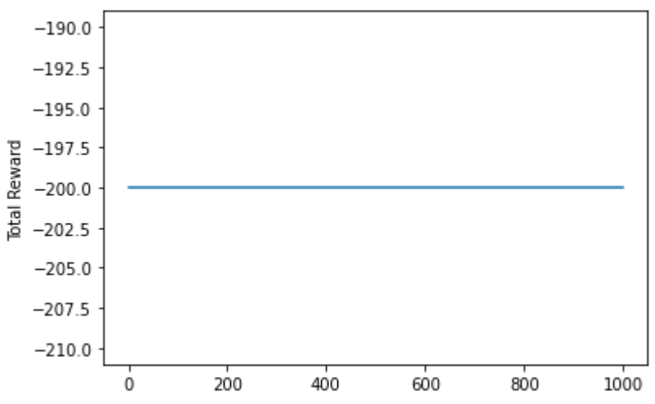
```
2
3
1
----
[-0.58912799  0.        ]
----
1
----
[-5.88639679e-01  4.88309600e-04]
-1.0
False
{}
----
```

Variation 1

```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.99
LR = 5e-4
UPDATE_EVERY = 20
EPISODES = 1000
LIM = 1
FC1 = 128
FC2 = 64
```

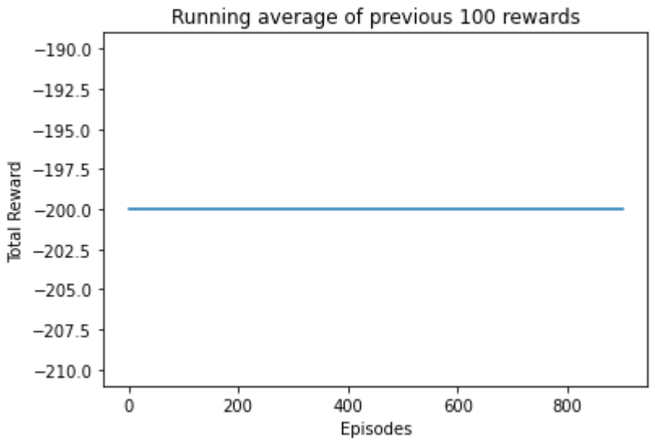
```
# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```





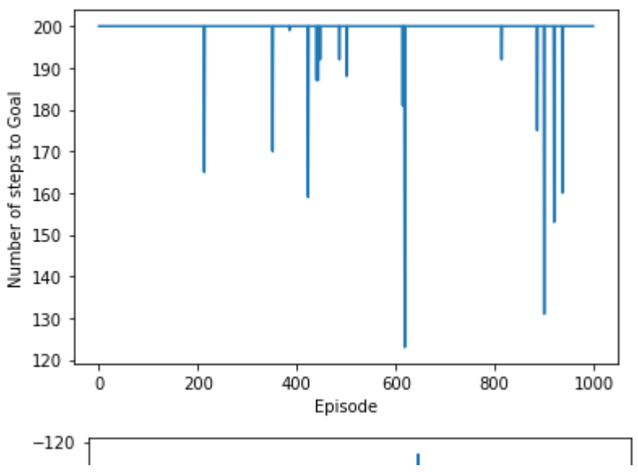

```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -200.0
```



Variation 2

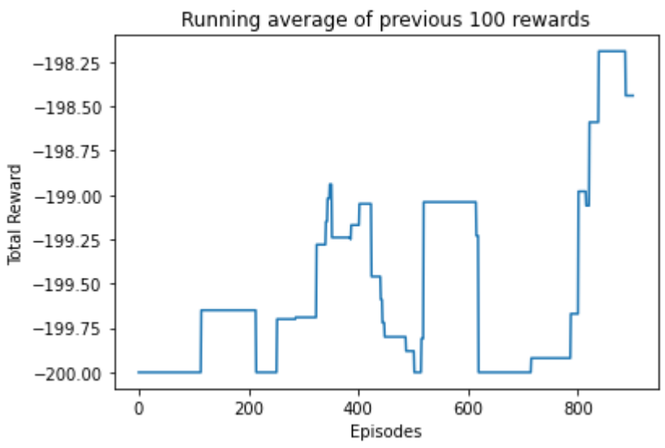
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 32
GAMMA = 0.80
LR = 2e-2
UPDATE_EVERY = 1
EPISODES = 1000
LIM = 1
FC1 = 256
FC2 = 256


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```




```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))
```

```
### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -198.44
```




```
#Make environment
total_rew = simulate_episode_dqn(env, wrap = True, render = True, video = True, log = False)
print("Cumulative reward after episode termination", total_rew)
```

```
Cumulative reward after episode termination -199.0
```

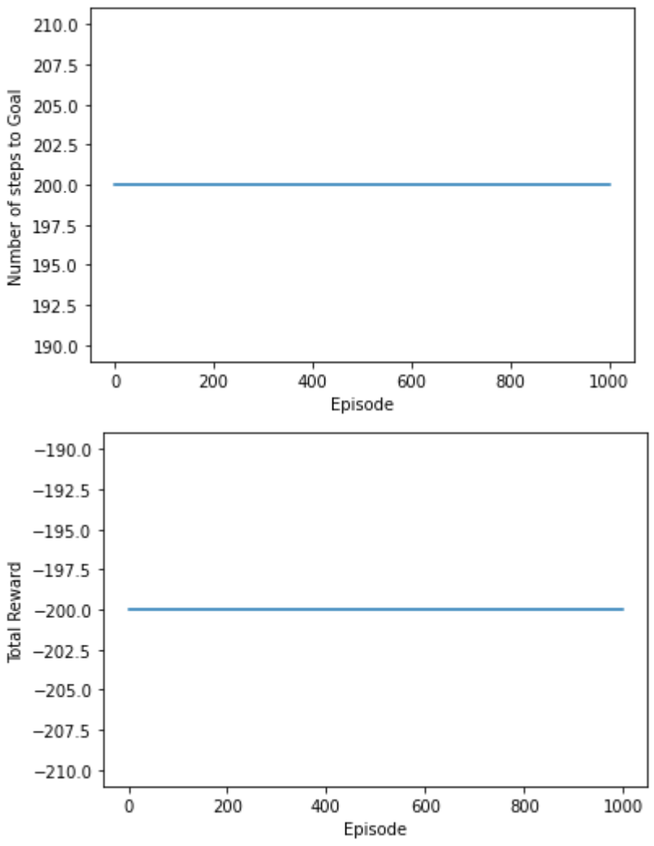
```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -179578.98000000004
```

Variation 3

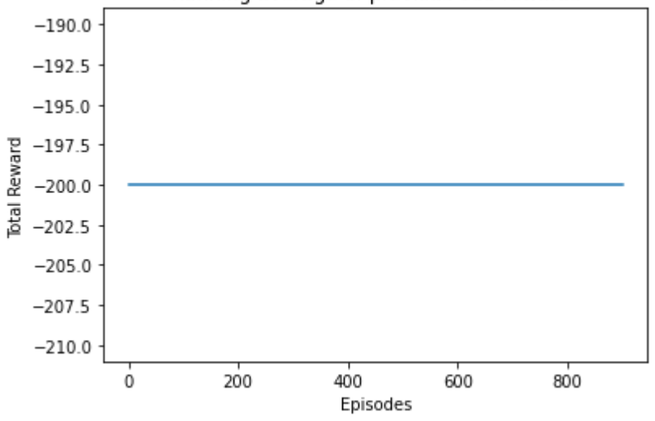
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 32
GAMMA = 0.2
LR = 2e-2
UPDATE_EVERY = 1
EPISODES = 1000
LIM = 1
FC1 = 256
FC2 = 256


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
```

```
  scores.append(score)
  steps.append(st)

plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```




```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -200.0
```




```
#Make environment
total_rew = simulate_episode_dqn(env, wrap = True, render = True, video = True, log = False)
print("Cumulative reward after episode termination", total_rew)
```

```
Cumulative reward after episode termination -199.0
```

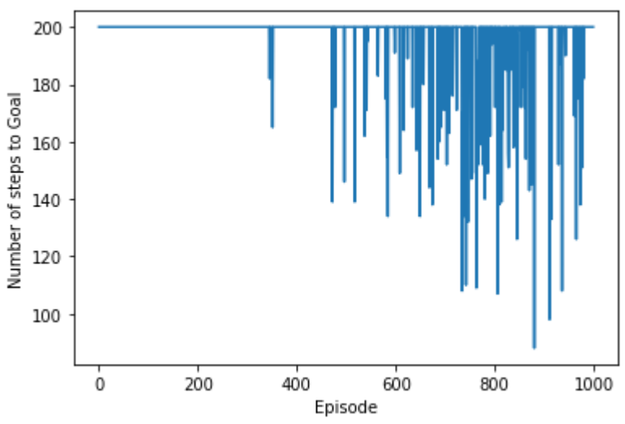
```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

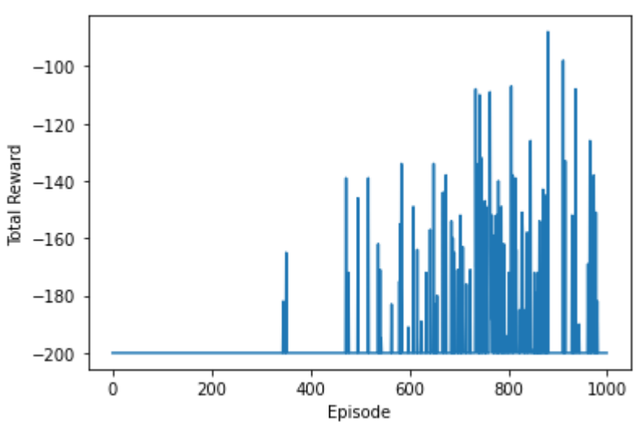
```
Area under the curve:  -180000.0
```

Variation 4

```
'''
Hyper parameters
'''
```

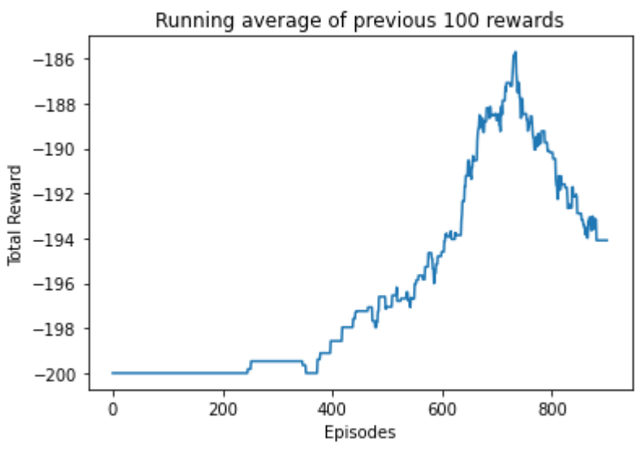
```
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 32
GAMMA = 0.9
LR = 2e-3
UPDATE_EVERY = 20
EPISODES = 1000
LIM = 2
FC1 = 128
FC2 = 128

# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```






```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -194.09
```



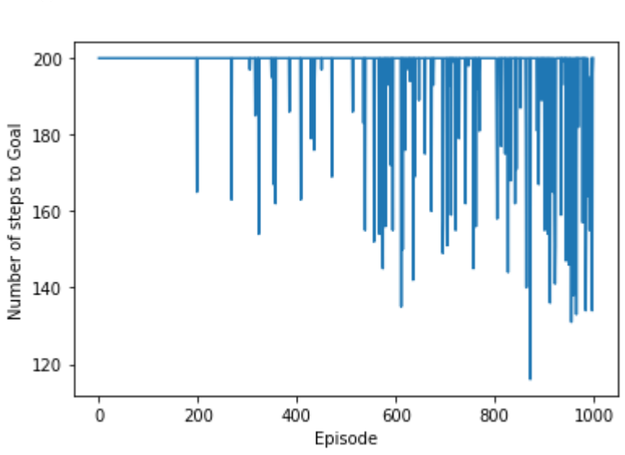

```
#Make environment
total_rew = simulate_episode_dqn(env, wrap = True, render = True, video = True, log = False)
print("Cumulative reward after episode termination", total_rew)
```

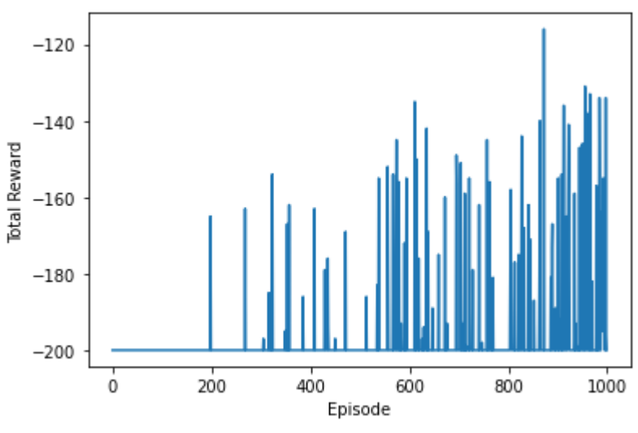
```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
     Area under the curve:  -176531.745
```

Variation 5

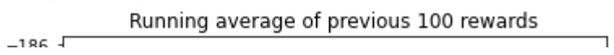
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.9
LR = 2e-3
UPDATE_EVERY = 40
EPISODES = 1000
LIM = 2
FC1 = 128
FC2 = 128


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```






```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -186.33
```




```
#Make environment
total_rew = simulate_episode_dqn(env, wrap = True, render = True, video = True, log = False)
print("Cumulative reward after episode termination", total_rew)
```

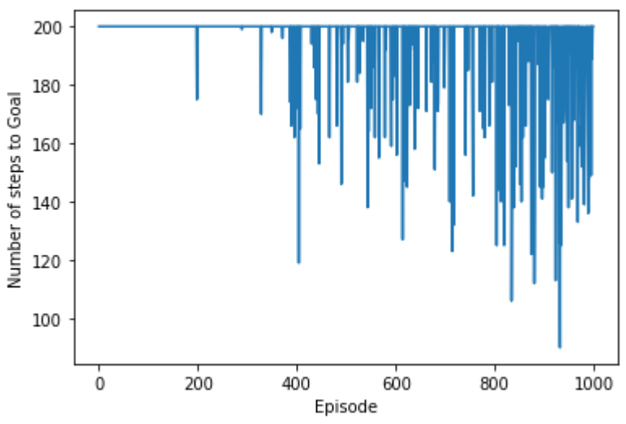
```
Cumulative reward after episode termination -199.0
```

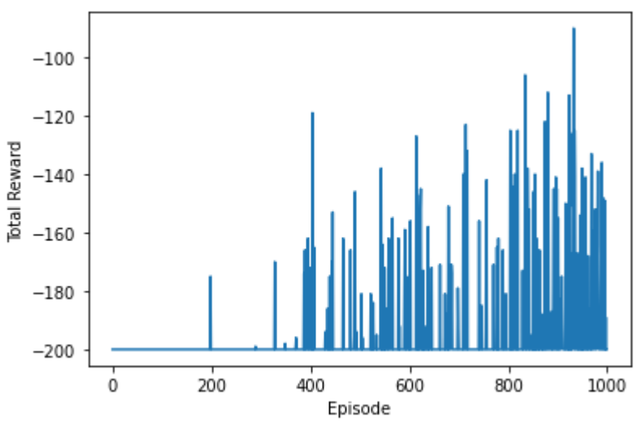
```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -177314.635
```

Variation 6

```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.9
LR = 2e-3
UPDATE_EVERY = 40
EPISODES = 1000
LIM = 2
FC1 = 256
FC2 = 128


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```

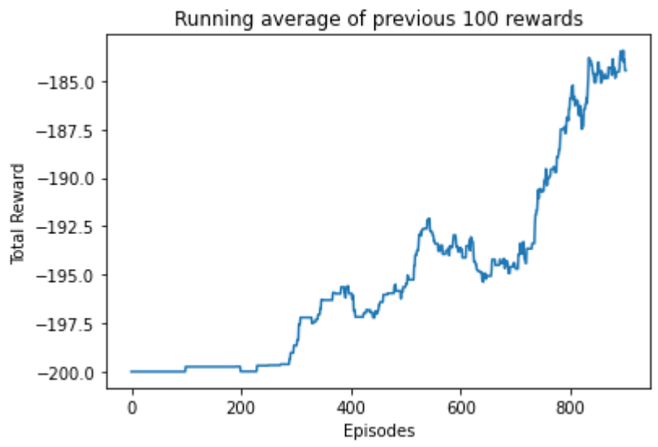





```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))
```

```
### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -184.47
```




```
#Make environment
total_rew = simulate_episode_dqn(env, wrap = True, render = True, video = True, log = False)
print("Cumulative reward after episode termination", total_rew)
```

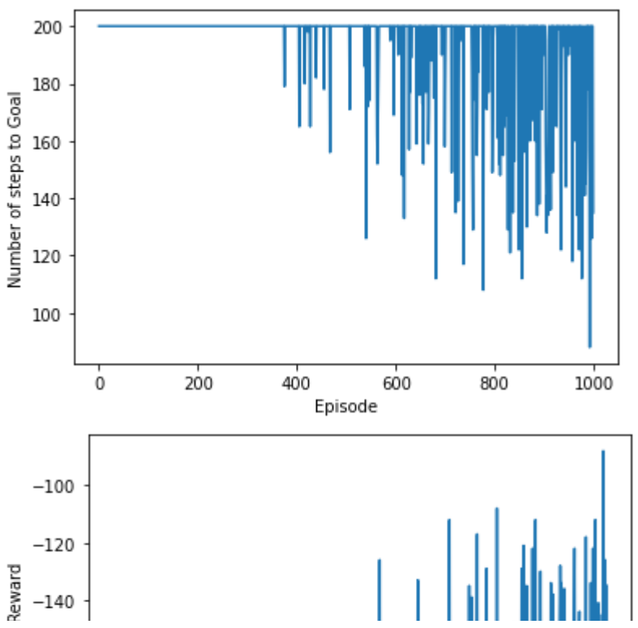
```
Cumulative reward after episode termination -199.0
```

```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -175640.935
```

Variation 7

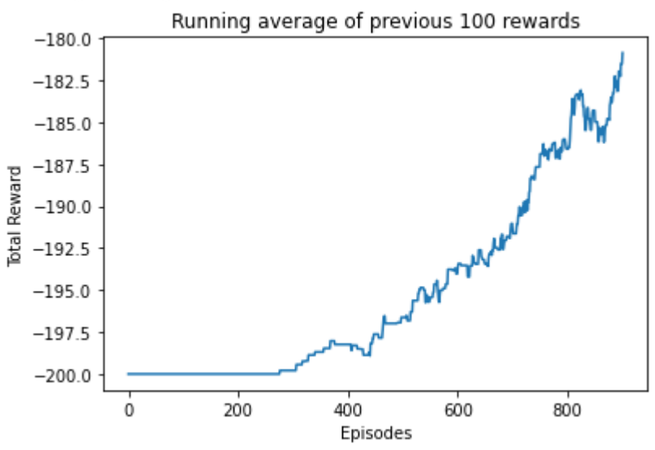
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.9
LR = 2e-4
UPDATE_EVERY = 50
EPISODES = 1000
LIM = 2
FC1 = 256
FC2 = 128


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```




```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))
```

```
### Plot of total reward vs episode
plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -180.88
```




```
#Make environment
total_rew = simulate_episode_dqn(env, wrap = True, render = True, video = True, log = False)
print("Cumulative reward after episode termination", total_rew)
```

```
Cumulative reward after episode termination -199.0
```

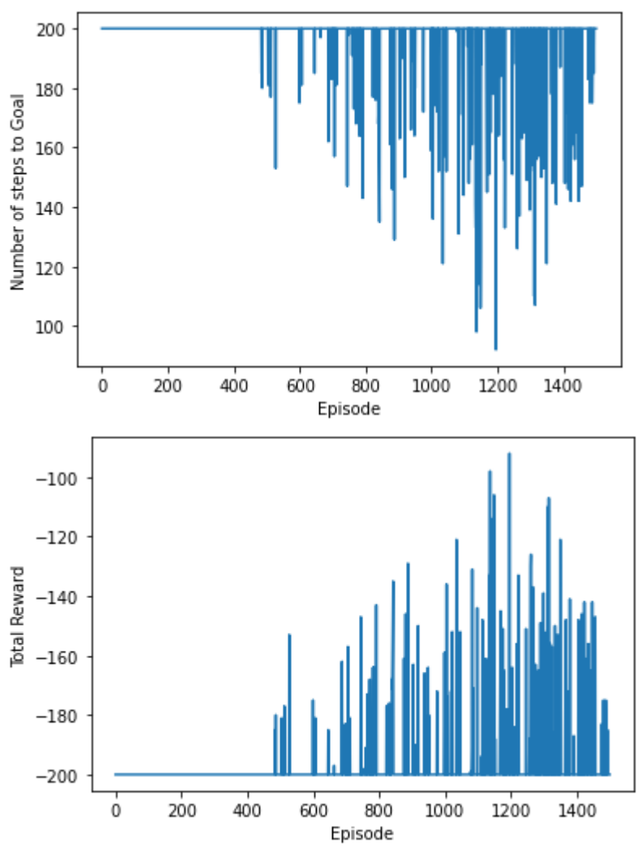
```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -175689.19
```

Variation 8

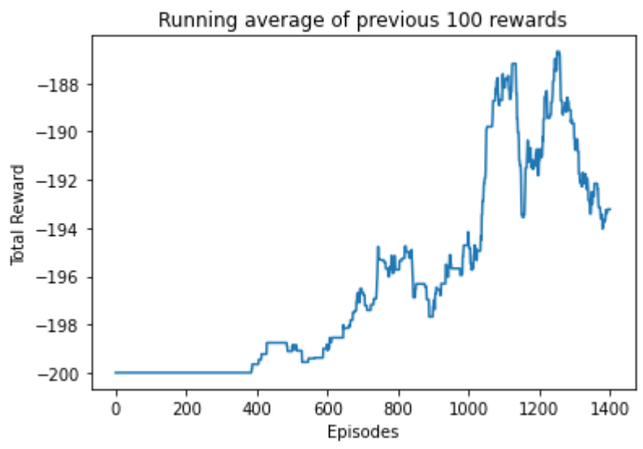
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.9
LR = 2e-4
UPDATE_EVERY = 50
EPISODES = 1500
LIM = 2
FC1 = 256
FC2 = 128


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES, eps_decay = 0.99)
```

```
  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```




```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -193.23
```




```
#Make environment
total_rew = simulate_episode_dqn(env, wrap = True, render = True, video = True, log = False)
print("Cumulative reward after episode termination", total_rew)
```

```
Cumulative reward after episode termination -199.0
```

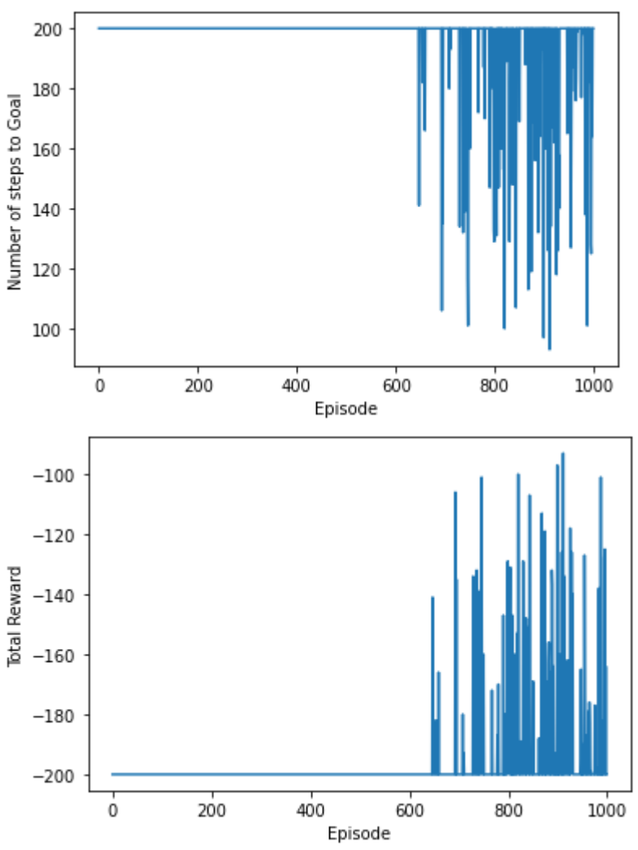
```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -274747.04500000004
```

Variation 9

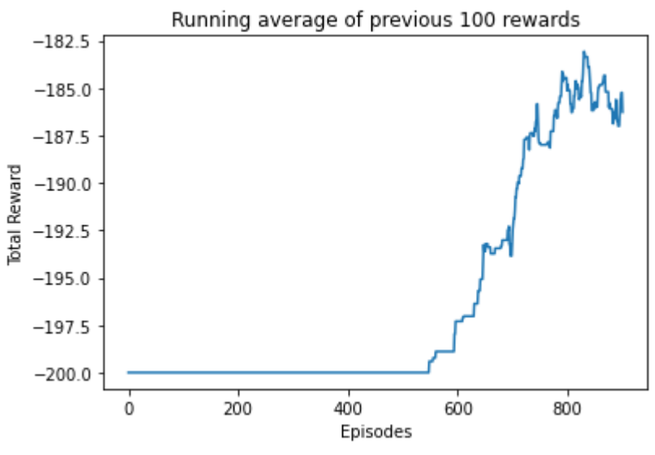
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 96
GAMMA = 0.9
LR = 2e-4
UPDATE_EVERY = 50
EPISODES = 1000
LIM = 2
FC1 = 256
FC2 = 128

# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)
```

```
plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```




```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -186.27
```



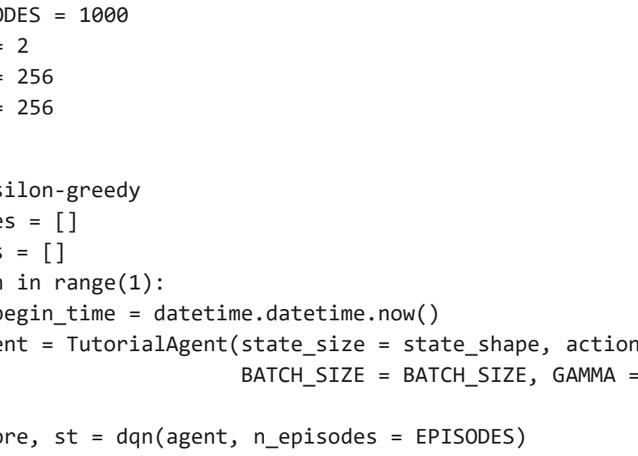

```
#Make environment
total_rew = simulate_episode_dqn(env, wrap = True, render = True, video = True, log = False)
print("Cumulative reward after episode termination", total_rew)
```

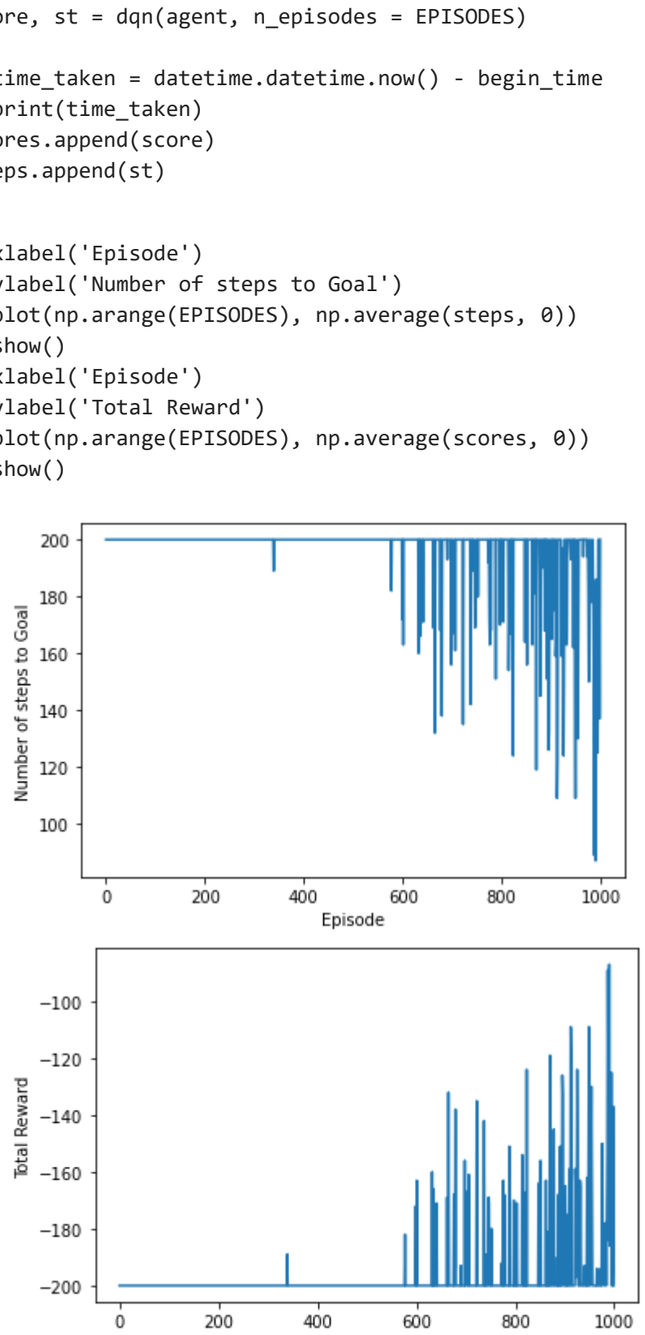
```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -176709.20500000002
```

Variation 10

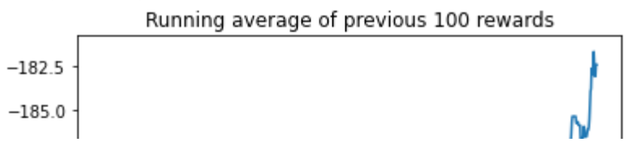
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 96
GAMMA = 0.9
LR = 2e-4
UPDATE_EVERY = 50
EPISODES = 1000
LIM = 2
FC1 = 256
FC2 = 256


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```






```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -182.43
```




```
#Make environment
total_rew = simulate_episode_dqn(env, wrap = True, render = True, video = True, log = False)
print("Cumulative reward after episode termination", total_rew)
```

```
Cumulative reward after episode termination -199.0
```

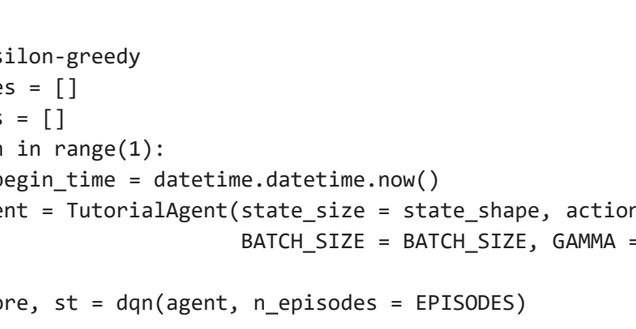
```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

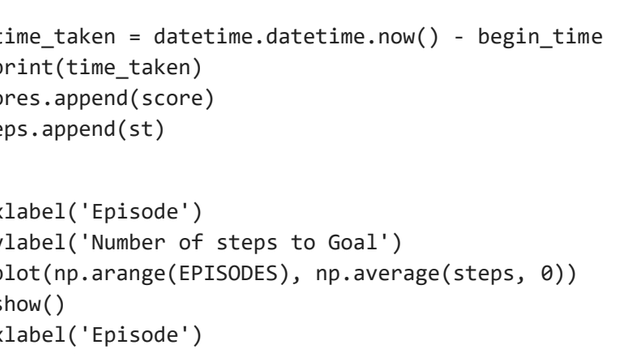
```
Area under the curve:  -177469.275
```

Variation 11

```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 96
GAMMA = 0.9
LR = 2e-4
UPDATE_EVERY = 80
EPISODES = 1000
LIM = 3
FC1 = 256
FC2 = 256
```

```
# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```






```
avg_reward_list = []
```

```
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))


### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -198.17
```



```
#Make environment
total_rew = simulate_episode_dqn(env, wrap = True, render = True, video = True, log = False)
print("Cumulative reward after episode termination", total_rew)
```

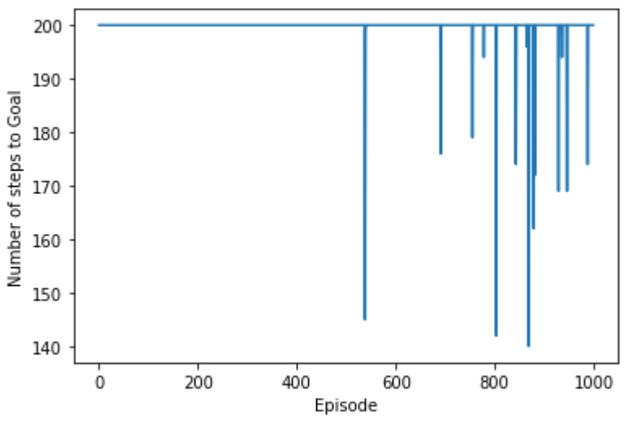
```
Cumulative reward after episode termination -199.0
```

```
# Compute the area using the composite trapezoidal rule.
area = np.trapz(avg_reward_list)
print("Area under the curve: ", area)
```

```
Area under the curve:  -179551.35499999998
```

Variation 12

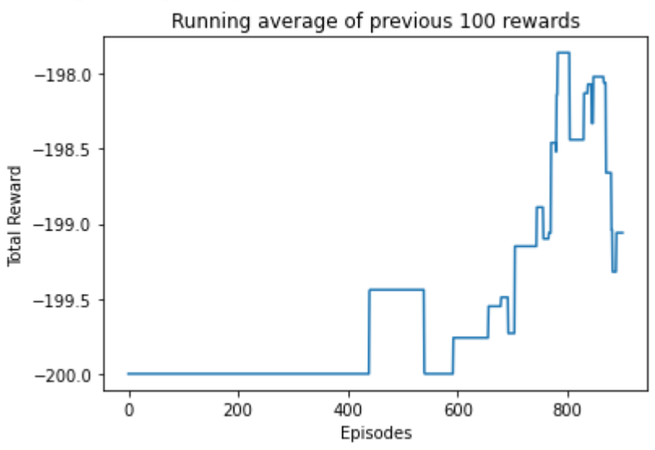
```
'''
Hyper parameters
'''
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.75
LR = 1e-2
UPDATE_EVERY = 10
EPISODES = 1000
LIM = 1
FC1 = 128
FC2 = 64


# Epsilon-greedy
scores = []
steps = []
for n in range(1):
  # begin_time = datetime.datetime.now()
  agent = TutorialAgent(state_size = state_shape, action_size = action_shape, fc1 = FC1, fc2 = FC2, BUFFER_SIZE = BUFFER_SIZE,
                        BATCH_SIZE = BATCH_SIZE, GAMMA = GAMMA, LR = LR, UPDATE_EVERY = UPDATE_EVERY, lim = LIM, seed = 0)

  score, st = dqn(agent, n_episodes = EPISODES)

  # time_taken = datetime.datetime.now() - begin_time
  # print(time_taken)
  scores.append(score)
  steps.append(st)


plt.xlabel('Episode')
plt.ylabel('Number of steps to Goal')
plt.plot(np.arange(EPISODES), np.average(steps, 0))
plt.show()
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.plot(np.arange(EPISODES), np.average(scores, 0))
plt.show()
```




```
avg_reward_list = []
reward_list = np.average(scores, 0)
for i in range(len(reward_list) - 99):
  avg_reward_list.append(np.mean(reward_list[i:i + 100]))
```

```
### Plot of total reward vs episode

plt.plot(avg_reward_list)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Running average of previous 100 rewards')
print('Running average of previous 100 rewards: ', avg_reward_list[-1])
```

```
Running average of previous 100 rewards:  -199.06
```




```
#Make environment
total_rew = simulate_episode_dqn(env, wrap = True, render = True, video = True, log = False)
print("Cumulative reward after episode termination", total_rew)
```

```
Cumulative reward after episode termination -199.0
```

```
'''
Installing packages for rendering the game on Colab
'''
```

```
!pip install gym pyvirtualdisplay > /dev/null 2>&1
!apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1
!apt-get update > /dev/null 2>&1
!apt-get install cmake > /dev/null 2>&1
!pip install --upgrade setuptools 2>&1
!pip install ez_setup > /dev/null 2>&1
!pip install gym[atari] > /dev/null 2>&1
!pip install git+https://github.com/tensorflow/docs > /dev/null 2>&1
```

```
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (61.2.0)
```

```
'''
A bunch of imports, you don't have to worry about these
'''

import numpy as np
import random
import torch
import torch.nn as nn
import torch.nn.functional as F
from collections import namedtuple, deque
import torch.optim as optim
import datetime
import gym
from gym.wrappers import Monitor
import glob
```

```
import io
import base64
import matplotlib.pyplot as plt
from IPython.display import HTML
from pyvirtualdisplay import Display
import tensorflow as tf
from IPython import display as ipythondisplay
from PIL import Image
import tensorflow_probability as tfp
import math
from scipy.special import softmax


import numpy as np
def tolerant_mean(arrs):
    lens = [len(i) for i in arrs]
    arr = np.ma.empty((np.max(lens),len(arrs)))
    arr.mask = True
    for idx, l in enumerate(arrs):
        arr[:len(l),idx] = l
    return arr.mean(axis = -1), arr.var(axis=-1)
```

Actor-Critic

```
class ActorCriticModel(tf.keras.Model):
"""
Defining policy and value networkss
"""
def __init__(self, action_size, n_hidden1=1024, n_hidden2=512):
    super(ActorCriticModel, self).__init__()

    #Hidden Layer 1
    self.fc1 = tf.keras.layers.Dense(n_hidden1, activation='relu')
    #Hidden Layer 2
    self.fc2 = tf.keras.layers.Dense(n_hidden2, activation='relu')

    #Output Layer for policy
    self.pi_out = tf.keras.layers.Dense(action_size, activation='softmax')
    #Output Layer for state-value
    self.v_out = tf.keras.layers.Dense(1)

def call(self, state):
    """
    Computes policy distribution and state-value for a given state
    """
    layer1 = self.fc1(state)
    layer2 = self.fc2(layer1)

    pi = self.pi_out(layer2)
    v = self.v_out(layer2)

    return pi, v
```

One-Step Return Agent

```
class Agent:
    """
    Agent class
    """
    def __init__(self, action_size, lr=0.001, gamma=0.99, seed = 85, n_hidden1=1024, n_hidden2=512):
        self.gamma = gamma
        self.ac_model = ActorCriticModel(action_size=action_size, n_hidden1=n_hidden1, n_hidden2=n_hidden2)
        self.ac_model.compile(tf.keras.optimizers.Adam(learning_rate=lr))
        np.random.seed(seed)

    def sample_action(self, state):
        """
        Given a state, compute the policy distribution over all actions and sample one action
        """
        pi,_ = self.ac_model(state)

        action_probabilities = tfp.distributions.Categorical(probs=pi)
        sample = action_probabilities.sample()

        return int(sample.numpy()[0])

    def actor_loss(self, action, pi, delta):
        """
        Compute Actor Loss
        """
        return -tf.math.log(pi[0,action]) * delta

    def critic_loss(self,delta):
        """
        Critic loss aims to minimize TD error
        """
        return delta**2

    @tf.function
    def learn(self, state, action, reward, next_state, done):
        """
        For a given transition (s,a,s',r) update the paramters by computing the
        gradient of the total loss
        """
        with tf.GradientTape(persistent=True) as tape:
            pi, V_s = self.ac_model(state)
            _, V_s_next = self.ac_model(next_state)

            V_s = tf.squeeze(V_s)
            V_s_next = tf.squeeze(V_s_next)


            #### TO DO: Write the equation for delta (TD error)
            ## Write code below
            delta = reward+self.gamma*V_s_next-V_s

            loss_a = self.actor_loss(action, pi, delta)
            loss_c =self.critic_loss(delta)
            loss_total = loss_a + loss_c
```

```
        gradient = tape.gradient(loss_total, self.ac_model.trainable_variables)
        self.ac_model.optimizer.apply_gradients(zip(gradient, self.ac_model.trainable_variables))
```

Full Returns Agent

```
import math
class AgentFullReturn:
    """
    Agent class
    """
    def __init__(self, action_size, lr=0.001, gamma=0.99, seed = 85, n_hidden1=1024, n_hidden2=512):
        self.gamma = gamma
        self.ac_model = ActorCriticModel(action_size=action_size)
        self.ac_model.compile(tf.keras.optimizers.Adam(learning_rate=lr), run_eagerly = True)
        np.random.seed(seed)

    def sample_action(self, state):
        """
        Given a state, compute the policy distribution over all actions and sample one action
        """
        pi,_ = self.ac_model(state)

        action_probabilities = tfp.distributions.Categorical(probs=pi)
        sample = action_probabilities.sample()

        return int(sample.numpy()[0])

    def actor_loss(self, actions, pis, deltas):
        """
        Compute Actor Loss
        """
        l = tf.convert_to_tensor(0.)
        actionsn = tf.convert_to_tensor(actions)
        for i in tf.range(tf.shape(actionsn)[0]):
          l = l -tf.math.log(pis[i][0][actions[i]]) * deltas[i]

        # tf.math.multiply(pis, deltas)

        return l

    def critic_loss(self,deltas):
        """
        Critic loss aims to minimize TD error
        """
        return tf.math.reduce_sum(tf.math.square(deltas))

    @tf.function
    def learn(self, state_trajectories, actions, reward_count):
        """
        For a given transition (s,a,s',r) update the paramters by computing the
        gradient of the total loss
        """
        deltas = tf.TensorArray(dtype=tf.float32, size=0, dynamic_size=True)
        pis = tf.TensorArray(dtype=tf.float32, size=0, dynamic_size=True)
        with tf.GradientTape(persistent=True) as tape:
          for i in tf.range(tf.shape(state_trajectories)[0]):
            delta = tf.convert_to_tensor(0.)
            state = state_trajectories[i]
            pi, V_s = self.ac_model(state)
            V_s = tf.squeeze(V_s)
            for j in tf.range(i,tf.shape(state_trajectories)[0]):
              delta = delta + tf.math.pow(self.gamma,tf.cast(j-i,tf.float32))*reward_count[j]

              #### TO DO: Write the equation for delta (TD error)
              ## Write code below

            delta = delta - V_s

            deltas = deltas.write(i, delta)
            pis = pis.write(i, pi)

          deltas = deltas.stack()
          pis = pis.stack()
          # print(tf.shape(deltas))
          loss_a = self.actor_loss(actions, pis, deltas)
          loss_c = self.critic_loss(deltas)
          loss_total = loss_a + loss_c

        gradient = tape.gradient(loss_total, self.ac_model.trainable_variables)
        self.ac_model.optimizer.apply_gradients(zip(gradient, self.ac_model.trainable_variables))
```

n-step Returns Agent

```
import math
class AgentNStep:
    """
    Agent class
    """
    def __init__(self, action_size, lr=0.001, gamma=0.99, seed = 85, n_hidden1=1024, n_hidden2=512):
        self.gamma = gamma
        self.ac_model = ActorCriticModel(action_size=action_size)
        self.ac_model.compile(tf.keras.optimizers.Adam(learning_rate=lr), run_eagerly = True)
        np.random.seed(seed)

    def sample_action(self, state):
        """
        Given a state, compute the policy distribution over all actions and sample one action
        """
        pi,_ = self.ac_model(state)

        action_probabilities = tfp.distributions.Categorical(probs=pi)
        sample = action_probabilities.sample()

        return int(sample.numpy()[0])

    def actor_loss(self, actions, pis, deltas):
        """
        Compute Actor Loss
```

```
    """
    l = tf.convert_to_tensor(0.)
    actionsn = tf.convert_to_tensor(actions)
    for i in tf.range(tf.shape(actionsn)[0]):
      l = l -tf.math.log(pis[i][0][actions[i]]) * deltas[i]

    # tf.math.multiply(pis, deltas)

    return l

def critic_loss(self,deltas):
    """
    Critic loss aims to minimize TD error
    """
    return tf.math.reduce_sum(tf.math.square(deltas))

@tf.function
def learn(self, state_trajectories, actions, reward_count):
    """
    For a given transition (s,a,s',r) update the paramters by computing the
    gradient of the total loss
    """
    n=0
    deltas = tf.TensorArray(dtype=tf.float32, size=0, dynamic_size=True)
    pis = tf.TensorArray(dtype=tf.float32, size=0, dynamic_size=True)
    with tf.GradientTape(persistent=True) as tape:
      for i in tf.range(tf.shape(state_trajectories)[0]):
        delta = tf.convert_to_tensor(0.)
        state = state_trajectories[i]
        pi, V_s = self.ac_model(state)
        _, V_s_n = self.ac_model(state_trajectories[i+n])
        V_s = tf.squeeze(V_s)
        V_s_n = tf.squeeze(V_s_n)
        for j in tf.range(i,n+i+1):
          delta = delta + tf.math.pow(self.gamma,tf.cast(j-i,tf.float32))*reward_count[j]

          #### TO DO: Write the equation for delta (TD error)
          ## Write code below

        delta = delta + tf.math.pow(self.gamma,tf.cast(n,tf.float32))*V_s_n - V_s

        deltas = deltas.write(i, delta)
        pis = pis.write(i, pi)

      deltas = deltas.stack()
      pis = pis.stack()
      # print(tf.shape(deltas))
      loss_a = self.actor_loss(actions, pis, deltas)
      loss_c = self.critic_loss(deltas)
      loss_total = loss_a + loss_c

    gradient = tape.gradient(loss_total, self.ac_model.trainable_variables)
    self.ac_model.optimizer.apply_gradients(zip(gradient, self.ac_model.trainable_variables))
```

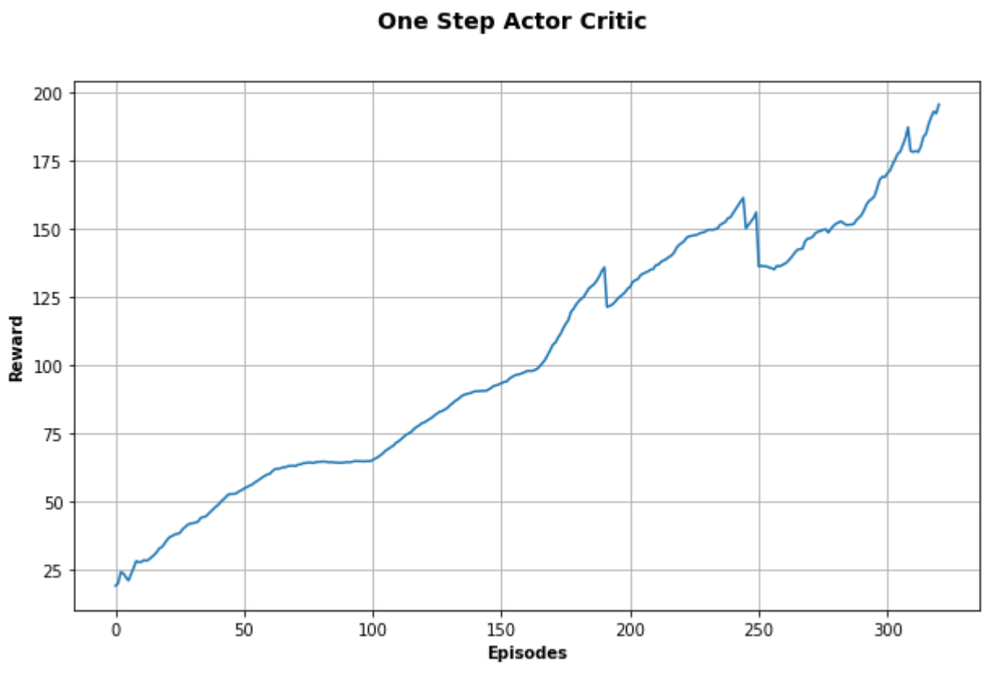
Cartpole-v1 environment

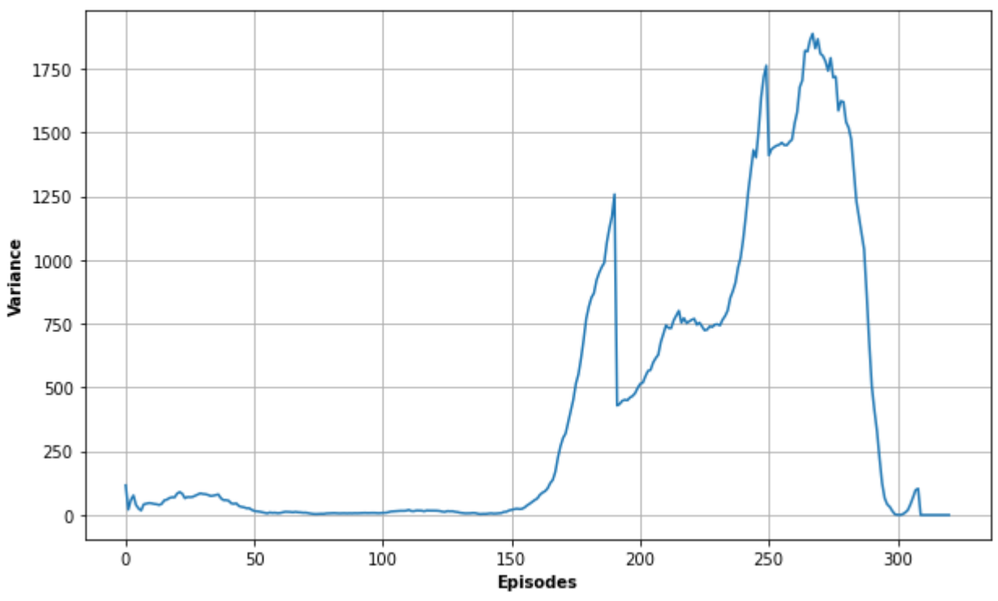
One-Step Return

```
env = gym.make('CartPole-v1')

#Initializing Agent
#Number of episodes
episodes = 1800

average_reward_list_list=[]
total_reward_list=[]
begin_time = datetime.datetime.now()

for i in range(5):
  average_reward_list=[]
  reward_list=[]
  agent = Agent(lr=1e-4, action_size=env.action_space.n)
  tf.compat.v1.reset_default_graph()
  for ep in range(1, episodes + 1):
      state = env.reset().reshape(1,-1)
      done = False
      ep_rew = 0
      while not done:
          action = agent.sample_action(state) ##Sample Action
          next_state, reward, done, info = env.step(action) ##Take action
          next_state = next_state.reshape(1,-1)
          ep_rew += reward  ##Updating episode reward
          agent.learn(state, action, reward, next_state, done) ##Update Parameters
          state = next_state ##Updating State
      reward_list.append(ep_rew)
      average_reward_list.append(np.mean(reward_list[-100:]))

      if ep % 10 == 0:
          avg_rew = np.mean(reward_list[-10:])
          print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' % avg_rew)

      if ep>100 and ep % 100:
          avg_100 =  np.mean(reward_list[-100:])
          if avg_100 > 195.0:
              print('Stopped at Episode ',ep-100)
              break
  total_reward_list.append(np.sum(reward_list))
  average_reward_list_list.append(average_reward_list)
average_reward_list=tolerant_mean(average_reward_list_list)[0]
variance_reward_list=tolerant_mean(average_reward_list_list)[1]
time_taken = datetime.datetime.now() - begin_time
print(time_taken)

     Episode  10 Reward 18.000000 Average Reward 22.700000
     Episode  20 Reward 74.000000 Average Reward 61.500000
     Episode  30 Reward 100.000000 Average Reward 78.100000
     Episode  40 Reward 100.000000 Average Reward 68.900000
     Episode  50 Reward 41.000000 Average Reward 68.100000
     Episode  60 Reward 81.000000 Average Reward 90.500000
     Episode  70 Reward 56.000000 Average Reward 87.000000
     Episode  80 Reward 92.000000 Average Reward 73.700000
```

```
Episode  90 Reward 108.000000 Average Reward 55.200000
Episode  100 Reward 54.000000 Average Reward 61.200000
Episode  110 Reward 230.000000 Average Reward 103.200000
Episode  120 Reward 122.000000 Average Reward 120.300000
Episode  130 Reward 99.000000 Average Reward 108.300000
Episode  140 Reward 144.000000 Average Reward 114.900000
Episode  150 Reward 109.000000 Average Reward 106.000000
Episode  160 Reward 93.000000 Average Reward 109.900000
Episode  170 Reward 109.000000 Average Reward 109.300000
Episode  180 Reward 47.000000 Average Reward 111.600000
Episode  190 Reward 72.000000 Average Reward 71.400000
Episode  200 Reward 45.000000 Average Reward 63.300000
Episode  210 Reward 58.000000 Average Reward 64.200000
Episode  220 Reward 114.000000 Average Reward 136.000000
Episode  230 Reward 72.000000 Average Reward 133.800000
Episode  240 Reward 81.000000 Average Reward 95.300000
Episode  250 Reward 263.000000 Average Reward 90.100000
Episode  260 Reward 123.000000 Average Reward 108.900000
Episode  270 Reward 171.000000 Average Reward 158.800000
Episode  280 Reward 166.000000 Average Reward 187.400000
Episode  290 Reward 500.000000 Average Reward 244.400000
Episode  300 Reward 268.000000 Average Reward 456.000000
Stopped at Episode  209
Episode  10 Reward 23.000000 Average Reward 29.400000
Episode  20 Reward 95.000000 Average Reward 43.500000
Episode  30 Reward 36.000000 Average Reward 57.000000
Episode  40 Reward 75.000000 Average Reward 61.300000
Episode  50 Reward 52.000000 Average Reward 65.900000
Episode  60 Reward 152.000000 Average Reward 86.600000
Episode  70 Reward 113.000000 Average Reward 72.800000
Episode  80 Reward 52.000000 Average Reward 72.700000
Episode  90 Reward 36.000000 Average Reward 55.900000
Episode  100 Reward 42.000000 Average Reward 58.600000
Episode  110 Reward 128.000000 Average Reward 70.700000
Episode  120 Reward 199.000000 Average Reward 126.800000
Episode  130 Reward 116.000000 Average Reward 147.500000
Episode  140 Reward 103.000000 Average Reward 119.500000
Episode  150 Reward 104.000000 Average Reward 111.100000
Episode  160 Reward 141.000000 Average Reward 128.100000
Episode  170 Reward 500.000000 Average Reward 284.700000
Episode  180 Reward 301.000000 Average Reward 411.200000
Episode  190 Reward 54.000000 Average Reward 79.300000
Episode  200 Reward 106.000000 Average Reward 94.300000
Episode  210 Reward 126.000000 Average Reward 164.200000
Episode  220 Reward 144.000000 Average Reward 177.200000
Episode  230 Reward 87.000000 Average Reward 125.000000
Episode  240 Reward 190.000000 Average Reward 196.200000
Episode  250 Reward 426.000000 Average Reward 311.800000
Stopped at Episode  150
Episode  10 Reward 18.000000 Average Reward 23.400000
```

```
### Plot of total reward vs episode
## Write Code Below
plt.figure(figsize=(10,6))
plt.plot(average_reward_list)
plt.suptitle("One Step Actor Critic", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Reward", fontweight = 'bold')
plt.grid(True)
plt.show()
plt.figure(figsize=(10,6))
plt.plot(variance_reward_list)
plt.suptitle("Variance of Reward across 5 runs", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Variance", fontweight = 'bold')
plt.grid(True)
plt.show()
print("Number of episodes for each run: 209, 150, 145, 221 ,91")
```





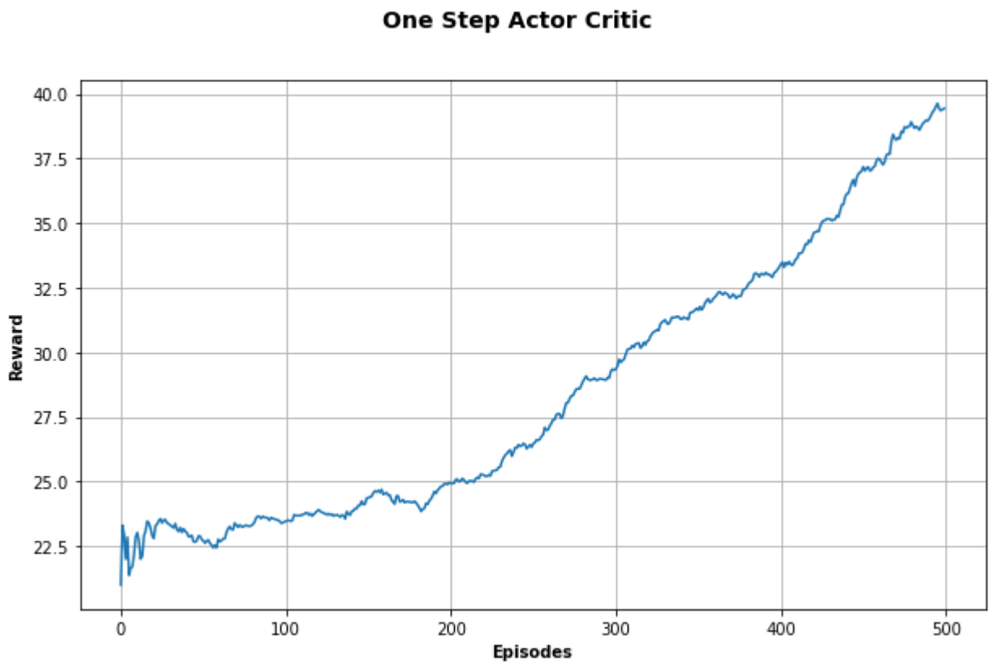

```
Number of episodes for each run: 209, 150, 145, 221 ,91
```

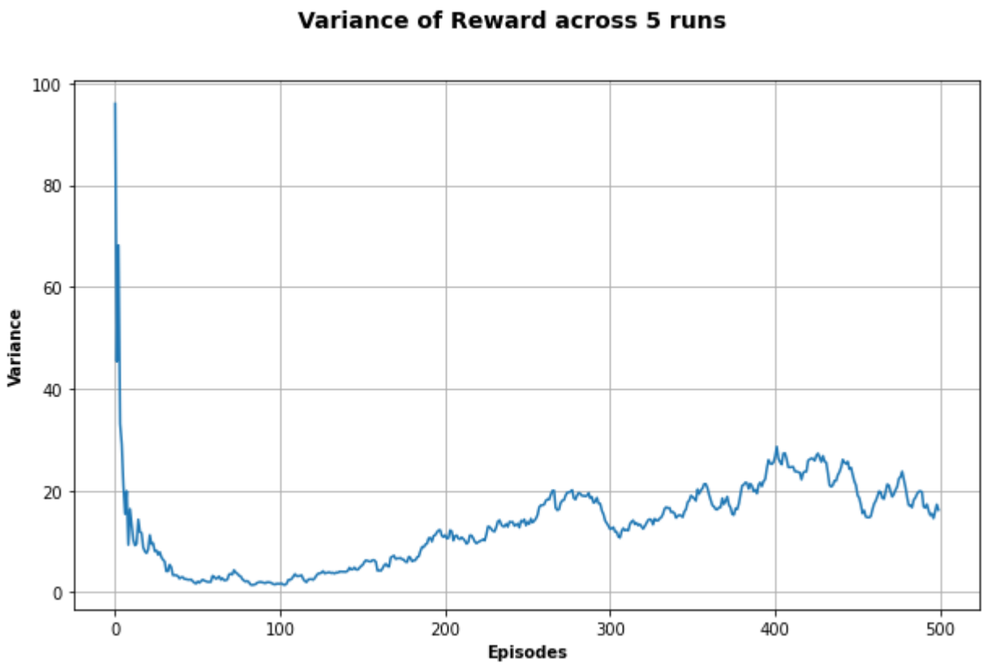
▾ Full Returns

```
env = gym.make('CartPole-v1')

#Initializing Agent
#Number of episodes
episodes = 500

average_reward_list_list=[]
total_reward_list=[]
begin_time = datetime.datetime.now()

for i in range(5):
  average_reward_list=[]
  reward_list=[]
  agent = AgentFullReturn(lr=1e-4, action_size=env.action_space.n)
  tf.compat.v1.reset_default_graph()
  for ep in range(1, episodes + 1):
      state = env.reset().reshape(1,-1)
      done = False
      ep_rew = 0
      state_trajectories = []
      actions = []
      reward_count = []
      iter = 0
      while not done and iter < 1000:
          action = agent.sample_action(state) ##Sample Action
          next_state, reward, done, info = env.step(action) ##Take action
          next_state = next_state.reshape(1,-1)
          ep_rew += reward  ##Updating episode reward
          state_trajectories.append(state)
          actions.append(action)
          reward_count.append(reward)
          #
          state = next_state ##Updating State
          iter+=1


      reward_list.append(ep_rew)
      average_reward_list.append(np.mean(reward_list[-100:]))

      state_trajectories = tf.stack(state_trajectories)
      actions = tf.stack(actions)
      reward_count = tf.stack(reward_count)
      agent.learn(state_trajectories, actions, reward_count) ##Update Parameters

      if ep % 10 == 0:
          avg_rew = np.mean(reward_list[-10:])
          print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' % avg_rew)

      if ep>100 and ep % 100:
          avg_100 =  np.mean(reward_list[-100:])
          if avg_100 > 195.0:
              print('Stopped at Episode ',ep-100)
              break
  total_reward_list.append(np.sum(reward_list))
  average_reward_list_list.append(average_reward_list)
average_reward_list=tolerant_mean(average_reward_list_list)[0]
variance_reward_list=tolerant_mean(average_reward_list_list)[1]
time_taken = datetime.datetime.now() - begin_time
print(time_taken)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function AgentFullReturn.learn at 0x7f28a1cb39e0> triggered tf.function retracing. Tracing is expensive and the excessive number of trac
WARNING:tensorflow:6 out of the last 6 calls to <function AgentFullReturn.learn at 0x7f28a1cb39e0> triggered tf.function retracing. Tracing is expensive and the excessive number of trac
Episode  10 Reward 20.000000 Average Reward 20.600000
Episode  20 Reward 31.000000 Average Reward 27.300000
Episode  30 Reward 27.000000 Average Reward 26.000000
Episode  40 Reward 18.000000 Average Reward 25.700000
Episode  50 Reward 16.000000 Average Reward 22.000000
Episode  60 Reward 24.000000 Average Reward 25.200000
Episode  70 Reward 95.000000 Average Reward 30.700000
Episode  80 Reward 19.000000 Average Reward 22.500000
Episode  90 Reward 25.000000 Average Reward 28.000000
Episode  100 Reward 17.000000 Average Reward 18.400000
Episode  110 Reward 20.000000 Average Reward 16.400000
Episode  120 Reward 26.000000 Average Reward 28.800000
Episode  130 Reward 12.000000 Average Reward 19.900000
Episode  140 Reward 16.000000 Average Reward 20.400000
Episode  150 Reward 40.000000 Average Reward 24.000000
Episode  160 Reward 9.000000 Average Reward 26.600000
Episode  170 Reward 28.000000 Average Reward 26.800000
Episode  180 Reward 12.000000 Average Reward 16.500000
Episode  190 Reward 40.000000 Average Reward 29.800000
Episode  200 Reward 20.000000 Average Reward 26.500000
Episode  210 Reward 12.000000 Average Reward 31.400000
Episode  220 Reward 20.000000 Average Reward 23.000000
Episode  230 Reward 22.000000 Average Reward 23.300000
Episode  240 Reward 24.000000 Average Reward 39.000000
Episode  250 Reward 24.000000 Average Reward 24.400000
Episode  260 Reward 29.000000 Average Reward 32.500000
Episode  270 Reward 17.000000 Average Reward 34.700000
Episode  280 Reward 23.000000 Average Reward 25.800000
Episode  290 Reward 20.000000 Average Reward 29.000000
Episode  300 Reward 21.000000 Average Reward 27.500000
Episode  310 Reward 22.000000 Average Reward 25.600000
Episode  320 Reward 49.000000 Average Reward 27.000000
Episode  330 Reward 31.000000 Average Reward 25.300000
Episode  340 Reward 15.000000 Average Reward 24.100000
Episode  350 Reward 12.000000 Average Reward 31.200000
Episode  360 Reward 34.000000 Average Reward 28.900000
Episode  370 Reward 24.000000 Average Reward 34.300000
Episode  380 Reward 31.000000 Average Reward 25.500000
Episode  390 Reward 31.000000 Average Reward 25.700000
Episode  400 Reward 34.000000 Average Reward 24.700000
Episode  410 Reward 37.000000 Average Reward 29.500000
Episode  420 Reward 69.000000 Average Reward 33.000000
Episode  430 Reward 16.000000 Average Reward 28.500000
Episode  440 Reward 17.000000 Average Reward 34.000000
Episode  450 Reward 47.000000 Average Reward 32.100000
Episode  460 Reward 29.000000 Average Reward 44.300000
Episode  470 Reward 19.000000 Average Reward 35.900000
Episode  480 Reward 75.000000 Average Reward 28.800000
Episode  490 Reward 40.000000 Average Reward 32.600000
Episode  500 Reward 62.000000 Average Reward 37.500000
Episode  10 Reward 62.000000 Average Reward 29.400000
Episode  20 Reward 15.000000 Average Reward 15.500000
Episode  30 Reward 20.000000 Average Reward 31.900000
Episode  40 Reward 22.000000 Average Reward 16.500000
Episode  50 Reward 12.000000 Average Reward 20.700000
```

```
### Plot of total reward vs episode
## Write Code Below
plt.figure(figsize=(10,6))
plt.plot(average_reward_list)
plt.suptitle("One Step Actor Critic", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Reward", fontweight = 'bold')
plt.grid(True)
plt.show()
plt.figure(figsize=(10,6))
plt.plot(variance_reward_list)
plt.suptitle("Variance of Reward across 5 runs", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Variance", fontweight = 'bold')
plt.grid(True)
plt.show()
```





n-step returns

```
env = gym.make('CartPole-v1')

#Initializing Agent
#Number of episodes
episodes = 500

average_reward_list_list=[]
total_reward_list=[]
begin_time = datetime.datetime.now()

for i in range(5):
  average_reward_list=[]
  reward_list=[]
  agent = AgentNStep(lr=1e-4, action_size=env.action_space.n)
  tf.compat.v1.reset_default_graph()
  for ep in range(1, episodes + 1):
      state = env.reset().reshape(1,-1)
      done = False
      ep_rew = 0
      state_trajectories = []
      actions = []
reward_count = []
iter = 0
while not done and iter < 1000:
    action = agent.sample_action(state) ##Sample Action
    next_state, reward, done, info = env.step(action) ##Take action
    next_state = next_state.reshape(1,-1)
    ep_rew += reward  ##Updating episode reward
    state_trajectories.append(state)
    actions.append(action)
    reward_count.append(reward)
    #
    state = next_state ##Updating State
    iter+=1


reward_list.append(ep_rew)
average_reward_list.append(np.mean(reward_list[-100:]))

state_trajectories = tf.stack(state_trajectories)
actions = tf.stack(actions)
reward_count = tf.stack(reward_count)
agent.learn(state_trajectories, actions, reward_count) ##Update Parameters

if ep % 10 == 0:
    avg_rew = np.mean(reward_list[-10:])
    print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' % avg_rew)
```

```
      if ep>100 and ep % 100:
          avg_100 =  np.mean(reward_list[-100:])
          if avg_100 > 195.0:
              print('Stopped at Episode ',ep-100)
              break
  total_reward_list.append(np.sum(reward_list))
  average_reward_list_list.append(average_reward_list)
average_reward_list=tolerant_mean(average_reward_list_list)[0]
variance_reward_list=tolerant_mean(average_reward_list_list)[1]
time_taken = datetime.datetime.now() - begin_time
print(time_taken)
```

```
     Episode  10 Reward 59.000000 Average Reward 25.300000
     Episode  20 Reward 54.000000 Average Reward 27.000000
     Episode  30 Reward 10.000000 Average Reward 17.700000
     Episode  40 Reward 27.000000 Average Reward 25.900000
     Episode  50 Reward 40.000000 Average Reward 25.100000
     Episode  60 Reward 42.000000 Average Reward 27.200000
     Episode  70 Reward 23.000000 Average Reward 23.100000
     Episode  80 Reward 25.000000 Average Reward 19.800000
     Episode  90 Reward 8.000000 Average Reward 19.300000
     Episode  100 Reward 15.000000 Average Reward 16.400000
     Episode  110 Reward 10.000000 Average Reward 20.600000
     Episode  120 Reward 16.000000 Average Reward 19.100000
     Episode  130 Reward 28.000000 Average Reward 22.000000
     Episode  140 Reward 19.000000 Average Reward 19.700000
     Episode  150 Reward 12.000000 Average Reward 18.000000
     Episode  160 Reward 28.000000 Average Reward 17.100000
     Episode  170 Reward 39.000000 Average Reward 19.800000
     Episode  180 Reward 32.000000 Average Reward 23.200000
     Episode  190 Reward 13.000000 Average Reward 18.500000
     Episode  200 Reward 14.000000 Average Reward 14.700000
     Episode  210 Reward 12.000000 Average Reward 13.300000
     Episode  220 Reward 9.000000 Average Reward 21.000000
     Episode  230 Reward 20.000000 Average Reward 16.100000
     Episode  240 Reward 22.000000 Average Reward 17.300000
     Episode  250 Reward 14.000000 Average Reward 26.300000
     Episode  260 Reward 26.000000 Average Reward 16.300000
     Episode  270 Reward 9.000000 Average Reward 17.400000
     Episode  280 Reward 20.000000 Average Reward 16.800000
     Episode  290 Reward 12.000000 Average Reward 13.700000
     Episode  300 Reward 18.000000 Average Reward 17.100000
     Episode  310 Reward 21.000000 Average Reward 17.300000
     Episode  320 Reward 13.000000 Average Reward 15.000000
     Episode  330 Reward 9.000000 Average Reward 16.900000
     Episode  340 Reward 22.000000 Average Reward 14.800000
     Episode  350 Reward 12.000000 Average Reward 16.600000
     Episode  360 Reward 9.000000 Average Reward 13.700000
     Episode  370 Reward 11.000000 Average Reward 13.200000
     Episode  380 Reward 9.000000 Average Reward 17.000000
     Episode  390 Reward 26.000000 Average Reward 17.100000
     Episode  400 Reward 11.000000 Average Reward 15.300000
     Episode  410 Reward 22.000000 Average Reward 15.700000
     Episode  420 Reward 19.000000 Average Reward 16.400000
     Episode  430 Reward 15.000000 Average Reward 14.800000
     Episode  440 Reward 17.000000 Average Reward 13.400000
     Episode  450 Reward 14.000000 Average Reward 13.900000
     Episode  460 Reward 11.000000 Average Reward 17.500000
     Episode  470 Reward 14.000000 Average Reward 14.100000
     Episode  480 Reward 11.000000 Average Reward 14.200000
     Episode  490 Reward 8.000000 Average Reward 12.100000
     Episode  500 Reward 9.000000 Average Reward 16.200000
     Episode  10 Reward 13.000000 Average Reward 19.100000
     Episode  20 Reward 17.000000 Average Reward 21.400000
     Episode  30 Reward 10.000000 Average Reward 17.100000
     Episode  40 Reward 29.000000 Average Reward 25.200000
     Episode  50 Reward 14.000000 Average Reward 24.300000
     Episode  60 Reward 14.000000 Average Reward 26.200000
     Episode  70 Reward 20.000000 Average Reward 29.000000
     Episode  80 Reward 13.000000 Average Reward 25.900000
```

```
### Plot of total reward vs episode
## Write Code Below
plt.figure(figsize=(10,6))
plt.plot(average_reward_list)
plt.suptitle("One Step Actor Critic", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Reward", fontweight = 'bold')
plt.grid(True)
plt.show()
plt.figure(figsize=(10,6))
plt.plot(variance_reward_list)
plt.suptitle("Variance of Reward across 5 runs", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Variance", fontweight = 'bold')
plt.grid(True)
plt.show()
```

Acrobot-v1

One-step returns

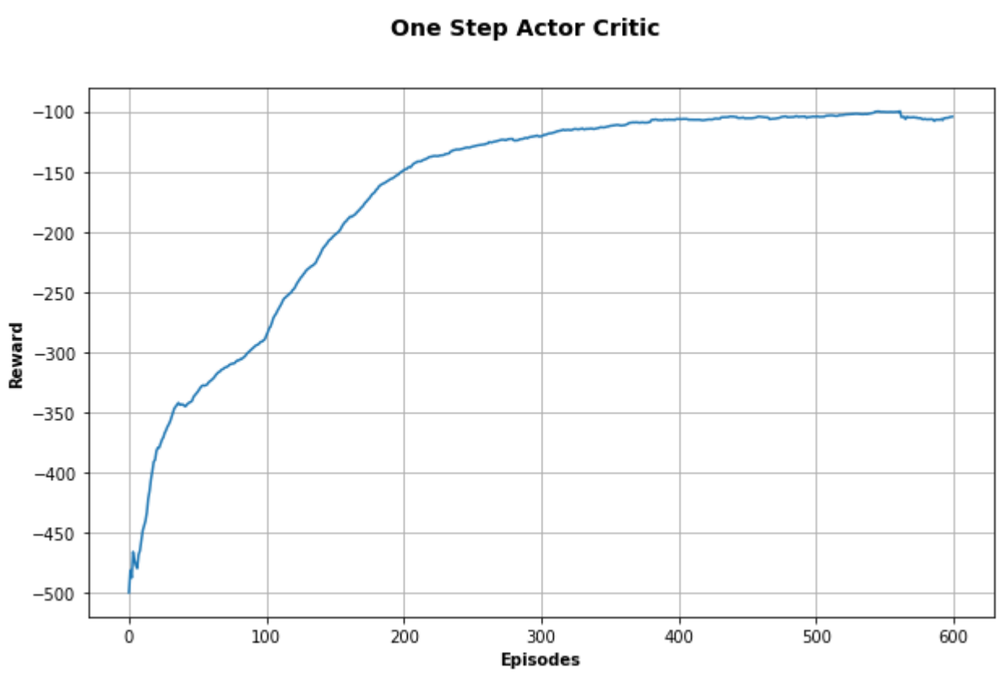

```
env = gym.make('Acrobot-v1')

#Initializing Agent
#Number of episodes
episodes = 600
```

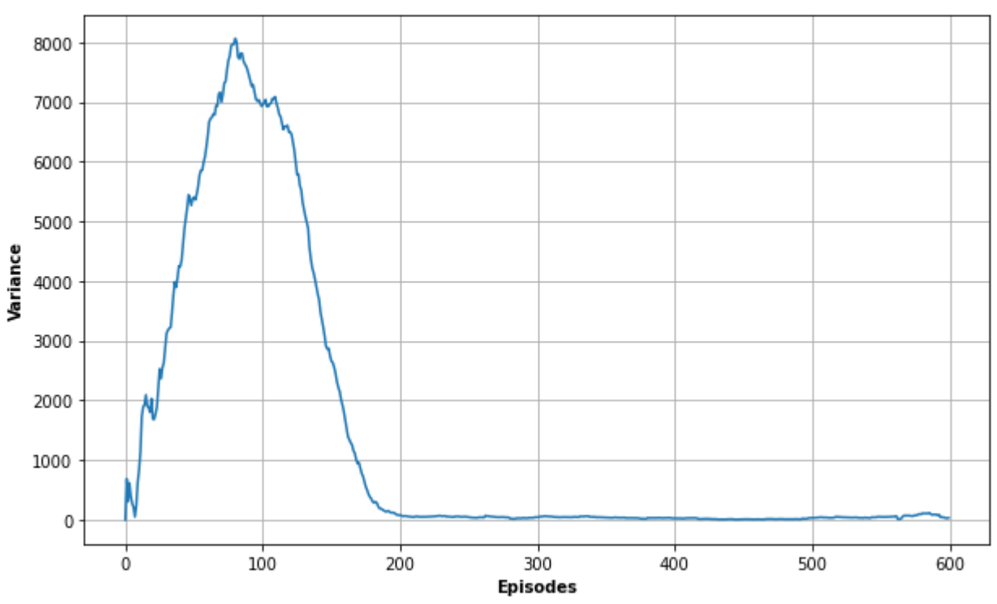
```
average_reward_list_list=[]
total_reward_list=[]
begin_time = datetime.datetime.now()

for i in range(5):
  average_reward_list=[]
  reward_list=[]
  agent = Agent(lr=0.00002, action_size=env.action_space.n, n_hidden1=256, n_hidden2=512)
  tf.compat.v1.reset_default_graph()
  for ep in range(1, episodes + 1):
      state = env.reset().reshape(1,-1)
      done = False
      ep_rew = 0
      while not done:
          action = agent.sample_action(state) ##Sample Action
          next_state, reward, done, info = env.step(action) ##Take action
          next_state = next_state.reshape(1,-1)
          ep_rew += reward  ##Updating episode reward
          agent.learn(state, action, reward, next_state, done) ##Update Parameters
          state = next_state ##Updating State
      reward_list.append(ep_rew)
      average_reward_list.append(np.mean(reward_list[-100:]))

      if ep % 10 == 0:
          avg_rew = np.mean(reward_list[-10:])
          print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' % avg_rew)

      if ep>100 and ep % 100:
          avg_100 =  np.mean(reward_list[-100:])
          if avg_100 > -90.0:
              print('Stopped at Episode ',ep-100)
              break
  total_reward_list.append(np.sum(reward_list))
  average_reward_list_list.append(average_reward_list)
average_reward_list=tolerant_mean(average_reward_list_list)[0]
variance_reward_list=tolerant_mean(average_reward_list_list)[1]
time_taken = datetime.datetime.now() - begin_time
print(time_taken)
```

```
    Episode  10 Reward -416.000000 Average Reward -472.400000
    Episode  20 Reward -485.000000 Average Reward -399.900000
    Episode  30 Reward -500.000000 Average Reward -421.700000
    Episode  40 Reward -500.000000 Average Reward -440.400000
    Episode  50 Reward -436.000000 Average Reward -451.700000
    Episode  60 Reward -475.000000 Average Reward -426.100000
    Episode  70 Reward -399.000000 Average Reward -417.700000
    Episode  80 Reward -354.000000 Average Reward -429.800000
    Episode  90 Reward -281.000000 Average Reward -324.600000
    Episode  100 Reward -252.000000 Average Reward -283.000000
    Episode  110 Reward -243.000000 Average Reward -232.400000
    Episode  120 Reward -187.000000 Average Reward -197.100000
    Episode  130 Reward -128.000000 Average Reward -149.000000
    Episode  140 Reward -142.000000 Average Reward -160.800000
    Episode  150 Reward -146.000000 Average Reward -153.400000
    Episode  160 Reward -135.000000 Average Reward -150.500000
    Episode  170 Reward -123.000000 Average Reward -152.300000
    Episode  180 Reward -129.000000 Average Reward -128.300000
    Episode  190 Reward -113.000000 Average Reward -127.600000
    Episode  200 Reward -111.000000 Average Reward -147.800000
    Episode  210 Reward -138.000000 Average Reward -133.100000
    Episode  220 Reward -120.000000 Average Reward -139.500000
    Episode  230 Reward -129.000000 Average Reward -125.200000
    Episode  240 Reward -123.000000 Average Reward -116.900000
    Episode  250 Reward -103.000000 Average Reward -143.000000
    Episode  260 Reward -128.000000 Average Reward -118.600000
    Episode  270 Reward -99.000000 Average Reward -121.600000
    Episode  280 Reward -99.000000 Average Reward -116.300000
    Episode  290 Reward -132.000000 Average Reward -111.200000
    Episode  300 Reward -139.000000 Average Reward -150.500000
    Episode  310 Reward -73.000000 Average Reward -95.600000
    Episode  320 Reward -87.000000 Average Reward -100.100000
    Episode  330 Reward -90.000000 Average Reward -110.700000
    Episode  340 Reward -92.000000 Average Reward -105.100000
    Episode  350 Reward -76.000000 Average Reward -99.000000
    Episode  360 Reward -96.000000 Average Reward -100.500000
    Episode  370 Reward -91.000000 Average Reward -98.700000
    Episode  380 Reward -101.000000 Average Reward -129.500000
    Episode  390 Reward -106.000000 Average Reward -113.400000
    Episode  400 Reward -87.000000 Average Reward -121.700000
    Episode  410 Reward -111.000000 Average Reward -113.000000
    Episode  420 Reward -112.000000 Average Reward -91.500000
    Episode  430 Reward -88.000000 Average Reward -97.000000
    Episode  440 Reward -102.000000 Average Reward -99.800000
    Episode  450 Reward -85.000000 Average Reward -108.600000
    Episode  460 Reward -104.000000 Average Reward -89.600000
    Episode  470 Reward -107.000000 Average Reward -142.300000
    Episode  480 Reward -120.000000 Average Reward -97.700000
    Episode  490 Reward -88.000000 Average Reward -103.600000
    Episode  500 Reward -174.000000 Average Reward -88.800000
    Episode  510 Reward -83.000000 Average Reward -95.600000
    Episode  520 Reward -98.000000 Average Reward -92.600000
```

```
     Episode  530 Reward -70.000000 Average Reward -89.000000
     Episode  540 Reward -112.000000 Average Reward -98.100000
     Episode  550 Reward -93.000000 Average Reward -109.500000
     Episode  560 Reward -94.000000 Average Reward -101.200000
     Episode  570 Reward -100.000000 Average Reward -88.500000
     Episode  580 Reward -99.000000 Average Reward -97.800000
```

```
### Plot of total reward vs episode
## Write Code Below
average_reward_list=tolerant_mean(average_reward_list_list[0:3])[0]
variance_reward_list=tolerant_mean(average_reward_list_list[0:3])[1]
plt.figure(figsize=(10,6))
plt.plot(average_reward_list)
plt.suptitle("One Step Actor Critic", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Reward", fontweight = 'bold')
plt.grid(True)
plt.show()
plt.figure(figsize=(10,6))
plt.plot(variance_reward_list)
plt.suptitle("Variance of Reward across 5 runs", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Variance", fontweight = 'bold')
plt.grid(True)
plt.show()
print("Number of episodes for each run: 600, 562, 600, 600 ,600")
```






```
Number of episodes for each run: 600, 562, 600, 600 ,600
```

Full Returns

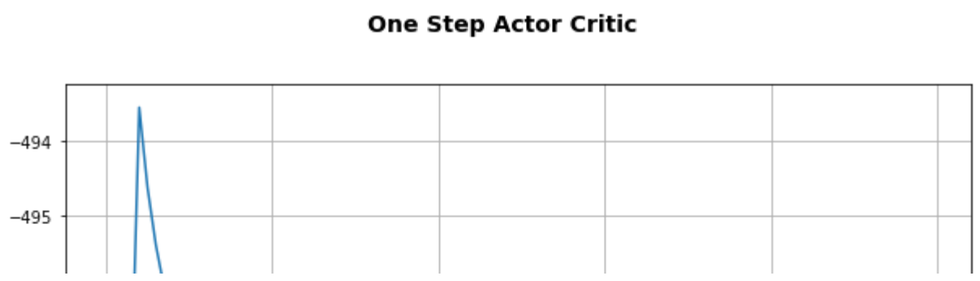
```
env = gym.make('Acrobot-v1')

#Initializing Agent
#Number of episodes
episodes = 100

average_reward_list_list=[]
total_reward_list=[]
begin_time = datetime.datetime.now()
```

```
for i in range(4):
  average_reward_list=[]
  reward_list=[]
  agent = AgentFullReturn(lr=0.00002, action_size=env.action_space.n, n_hidden1=256, n_hidden2=512)
  tf.compat.v1.reset_default_graph()
  for ep in range(1, episodes + 1):
      state = env.reset().reshape(1,-1)
      done = False
      ep_rew = 0
      state_trajectories = []
      actions = []
      reward_count = []
      iter = 0
      while not done and iter < 1000:
          action = agent.sample_action(state) ##Sample Action
          next_state, reward, done, info = env.step(action) ##Take action
          next_state = next_state.reshape(1,-1)
          ep_rew += reward  ##Updating episode reward
          state_trajectories.append(state)
          actions.append(action)
          reward_count.append(reward)
          #
          state = next_state ##Updating State
          iter+=1


      reward_list.append(ep_rew)
      average_reward_list.append(np.mean(reward_list[-100:]))
```

```
      state_trajectories = tf.stack(state_trajectories)
      actions = tf.stack(actions)
      reward_count = tf.stack(reward_count)
      agent.learn(state_trajectories, actions, reward_count) ##Update Parameters

      if ep % 10 == 0:
          avg_rew = np.mean(reward_list[-10:])
          print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' % avg_rew)

      if ep>100 and ep % 100:
          avg_100 =  np.mean(reward_list[-100:])
          if avg_100 > -90.0:
              print('Stopped at Episode ',ep-100)
              break
  total_reward_list.append(np.sum(reward_list))
  average_reward_list_list.append(average_reward_list)
average_reward_list=tolerant_mean(average_reward_list_list)[0]
variance_reward_list=tolerant_mean(average_reward_list_list)[1]
time_taken = datetime.datetime.now() - begin_time
print(time_taken)
```

```
     Episode  10 Reward -500.000000 Average Reward -487.100000
     Episode  20 Reward -500.000000 Average Reward -500.000000
     Episode  30 Reward -500.000000 Average Reward -500.000000
     Episode  40 Reward -500.000000 Average Reward -500.000000
     Episode  50 Reward -500.000000 Average Reward -492.600000
     Episode  60 Reward -500.000000 Average Reward -500.000000
     Episode  70 Reward -500.000000 Average Reward -490.200000
     Episode  80 Reward -500.000000 Average Reward -500.000000
     Episode  90 Reward -500.000000 Average Reward -500.000000
     Episode  100 Reward -500.000000 Average Reward -478.100000
     Episode  10 Reward -500.000000 Average Reward -500.000000
     Episode  20 Reward -500.000000 Average Reward -500.000000
     Episode  30 Reward -500.000000 Average Reward -500.000000
     Episode  40 Reward -414.000000 Average Reward -491.400000
     Episode  50 Reward -500.000000 Average Reward -500.000000
     Episode  60 Reward -500.000000 Average Reward -500.000000
     Episode  70 Reward -500.000000 Average Reward -495.800000
     Episode  80 Reward -500.000000 Average Reward -500.000000
     Episode  90 Reward -500.000000 Average Reward -497.800000
     Episode  100 Reward -500.000000 Average Reward -500.000000
     Episode  10 Reward -500.000000 Average Reward -500.000000
     Episode  20 Reward -500.000000 Average Reward -494.700000
     Episode  30 Reward -500.000000 Average Reward -500.000000
     Episode  40 Reward -500.000000 Average Reward -500.000000
     Episode  50 Reward -500.000000 Average Reward -500.000000
     Episode  60 Reward -500.000000 Average Reward -497.600000
     Episode  70 Reward -500.000000 Average Reward -500.000000
     Episode  80 Reward -500.000000 Average Reward -492.100000
     Episode  90 Reward -406.000000 Average Reward -490.600000
     Episode  100 Reward -500.000000 Average Reward -500.000000
     Episode  10 Reward -500.000000 Average Reward -500.000000
     Episode  20 Reward -500.000000 Average Reward -500.000000
     Episode  30 Reward -500.000000 Average Reward -497.400000
     Episode  40 Reward -500.000000 Average Reward -500.000000
     Episode  50 Reward -500.000000 Average Reward -500.000000
     Episode  60 Reward -500.000000 Average Reward -500.000000
     Episode  70 Reward -500.000000 Average Reward -500.000000
     Episode  80 Reward -500.000000 Average Reward -500.000000
     Episode  90 Reward -500.000000 Average Reward -484.600000
     Episode  100 Reward -500.000000 Average Reward -500.000000
     0:53:30.163109
```

```
### Plot of total reward vs episode
## Write Code Below
plt.figure(figsize=(10,6))
plt.plot(average_reward_list)
plt.suptitle("One Step Actor Critic", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Reward", fontweight = 'bold')
plt.grid(True)
plt.show()
plt.figure(figsize=(10,6))
plt.plot(variance_reward_list)
plt.suptitle("Variance of Reward across 5 runs", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Variance", fontweight = 'bold')
plt.grid(True)
plt.show()
```

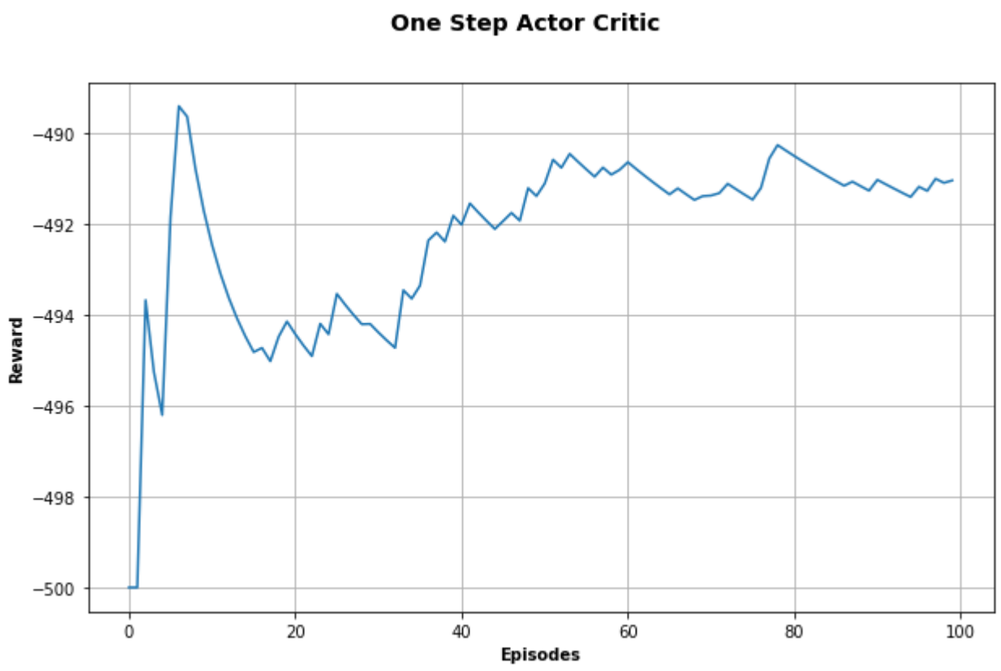


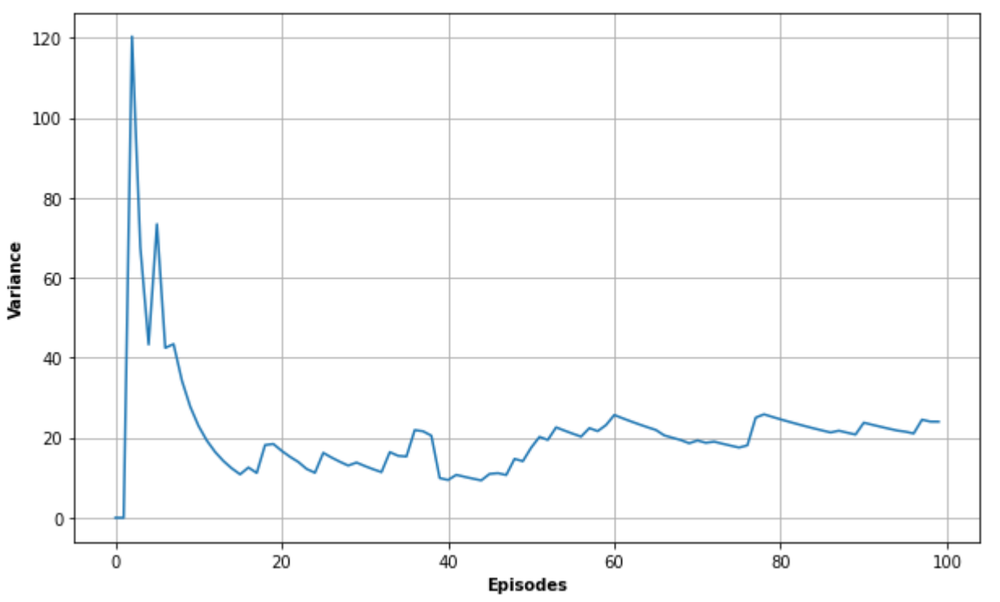
n-step returns


```
env = gym.make('Acrobot-v1')

#Initializing Agent
#Number of episodes
episodes = 100

average_reward_list_list=[]
total_reward_list=[]
begin_time = datetime.datetime.now()

for i in range(4):
  average_reward_list=[]
  reward_list=[]
  agent = AgentNStep(lr=0.00002, action_size=env.action_space.n, n_hidden1=256, n_hidden2=512)
tf.compat.v1.reset_default_graph()
for ep in range(1, episodes + 1):
    state = env.reset().reshape(1,-1)
    done = False
    ep_rew = 0
    state_trajectories = []
    actions = []
    reward_count = []
    iter = 0
    while not done and iter < 1000:
        action = agent.sample_action(state) ##Sample Action
        next_state, reward, done, info = env.step(action) ##Take action
        next_state = next_state.reshape(1,-1)
        ep_rew += reward  ##Updating episode reward
        state_trajectories.append(state)
        actions.append(action)
        reward_count.append(reward)
        #
        state = next_state ##Updating State
        iter+=1


    reward_list.append(ep_rew)
    average_reward_list.append(np.mean(reward_list[-100:]))

    state_trajectories = tf.stack(state_trajectories)
    actions = tf.stack(actions)
    reward_count = tf.stack(reward_count)
    agent.learn(state_trajectories, actions, reward_count) ##Update Parameters

    if ep % 10 == 0:
        avg_rew = np.mean(reward_list[-10:])
        print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' % avg_rew)

    if ep>100 and ep % 100:
        avg_100 =  np.mean(reward_list[-100:])
        if avg_100 > -90.0:
            print('Stopped at Episode ',ep-100)
            break
  total_reward_list.append(np.sum(reward_list))
  average_reward_list_list.append(average_reward_list)
average_reward_list=tolerant_mean(average_reward_list_list)[0]
variance_reward_list=tolerant_mean(average_reward_list_list)[1]
time_taken = datetime.datetime.now() - begin_time
print(time_taken)
```

```
    Episode  10 Reward -500.000000 Average Reward -500.000000
    Episode  20 Reward -500.000000 Average Reward -500.000000
    Episode  30 Reward -500.000000 Average Reward -500.000000
    Episode  40 Reward -378.000000 Average Reward -487.800000
    Episode  50 Reward -500.000000 Average Reward -500.000000
    Episode  60 Reward -500.000000 Average Reward -500.000000
    Episode  70 Reward -500.000000 Average Reward -500.000000
    Episode  80 Reward -500.000000 Average Reward -500.000000
    Episode  90 Reward -500.000000 Average Reward -500.000000
    Episode  100 Reward -500.000000 Average Reward -500.000000
    Episode  10 Reward -500.000000 Average Reward -488.000000
    Episode  20 Reward -451.000000 Average Reward -495.100000
    Episode  30 Reward -500.000000 Average Reward -500.000000
    Episode  40 Reward -500.000000 Average Reward -480.600000
    Episode  50 Reward -500.000000 Average Reward -490.700000
    Episode  60 Reward -500.000000 Average Reward -500.000000
    Episode  70 Reward -442.000000 Average Reward -487.200000
    WARNING:tensorflow:5 out of the last 27 calls to <function AgentNStep.learn at 0x7f289ce025f0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings co
    WARNING:tensorflow:5 out of the last 11 calls to <function AgentNStep.learn at 0x7f289ce025f0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings co
    Episode  80 Reward -500.000000 Average Reward -462.700000
    Episode  90 Reward -500.000000 Average Reward -493.300000
    Episode  100 Reward -443.000000 Average Reward -494.300000
    Episode  10 Reward -500.000000 Average Reward -492.400000
    Episode  20 Reward -500.000000 Average Reward -500.000000
    Episode  30 Reward -476.000000 Average Reward -477.200000
    Episode  40 Reward -500.000000 Average Reward -494.300000
    Episode  50 Reward -500.000000 Average Reward -483.600000
    Episode  60 Reward -438.000000 Average Reward -469.000000
    Episode  70 Reward -500.000000 Average Reward -492.200000
    Episode  80 Reward -500.000000 Average Reward -470.700000
    Episode  90 Reward -500.000000 Average Reward -500.000000
    Episode  100 Reward -500.000000 Average Reward -473.800000
    Episode  10 Reward -500.000000 Average Reward -486.400000
    Episode  20 Reward -500.000000 Average Reward -491.200000
    Episode  30 Reward -500.000000 Average Reward -500.000000
    Episode  40 Reward -500.000000 Average Reward -475.900000
    Episode  50 Reward -500.000000 Average Reward -484.300000
    Episode  60 Reward -500.000000 Average Reward -482.600000
    Episode  70 Reward -500.000000 Average Reward -500.000000
    Episode  80 Reward -500.000000 Average Reward -500.000000
    Episode  90 Reward -500.000000 Average Reward -500.000000
    Episode  100 Reward -500.000000 Average Reward -487.900000
    0:44:00.365325
```

```
### Plot of total reward vs episode
## Write Code Below
plt.figure(figsize=(10,6))
plt.plot(average_reward_list)
plt.suptitle("One Step Actor Critic", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Reward", fontweight = 'bold')
plt.grid(True)
plt.show()
plt.figure(figsize=(10,6))
plt.plot(variance_reward_list)
plt.suptitle("Variance of Reward across 5 runs", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Variance", fontweight = 'bold')
plt.grid(True)
plt.show()
```





Mountaincar-v0

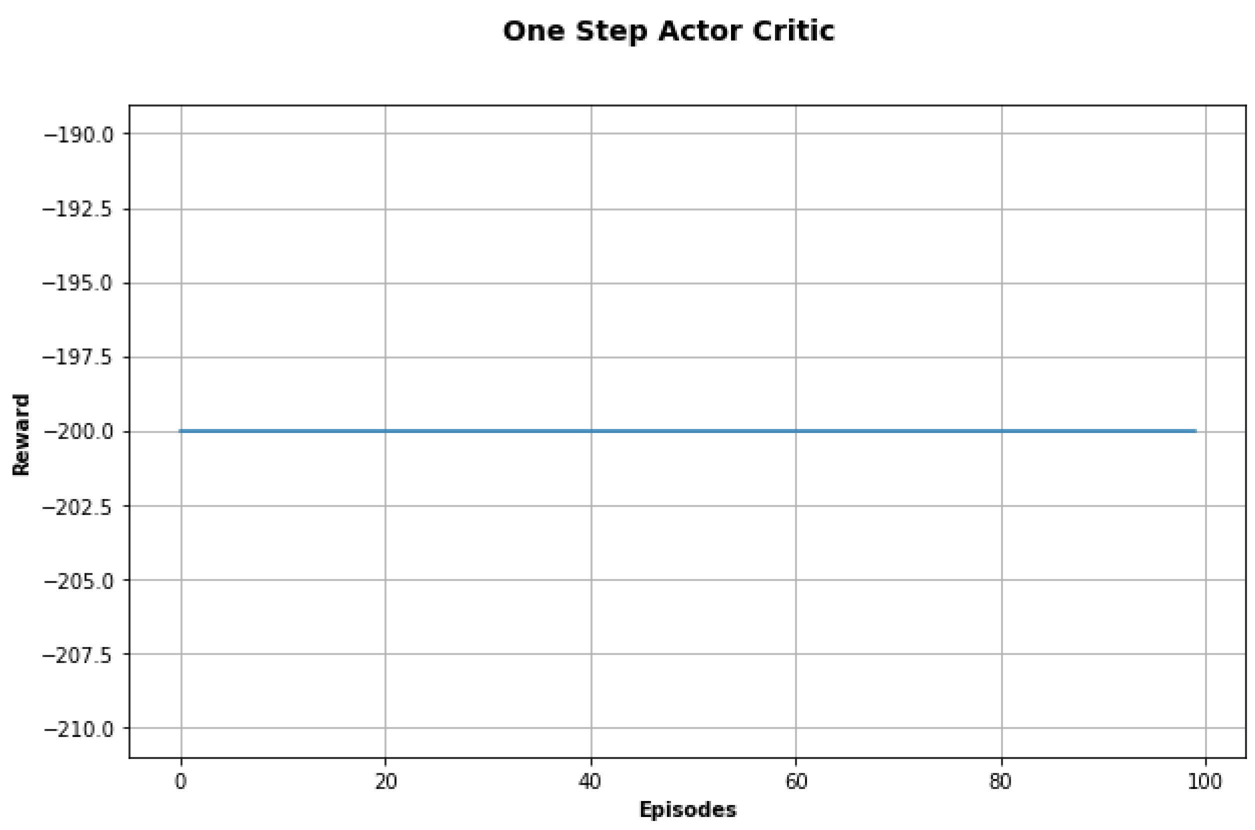
One-step return

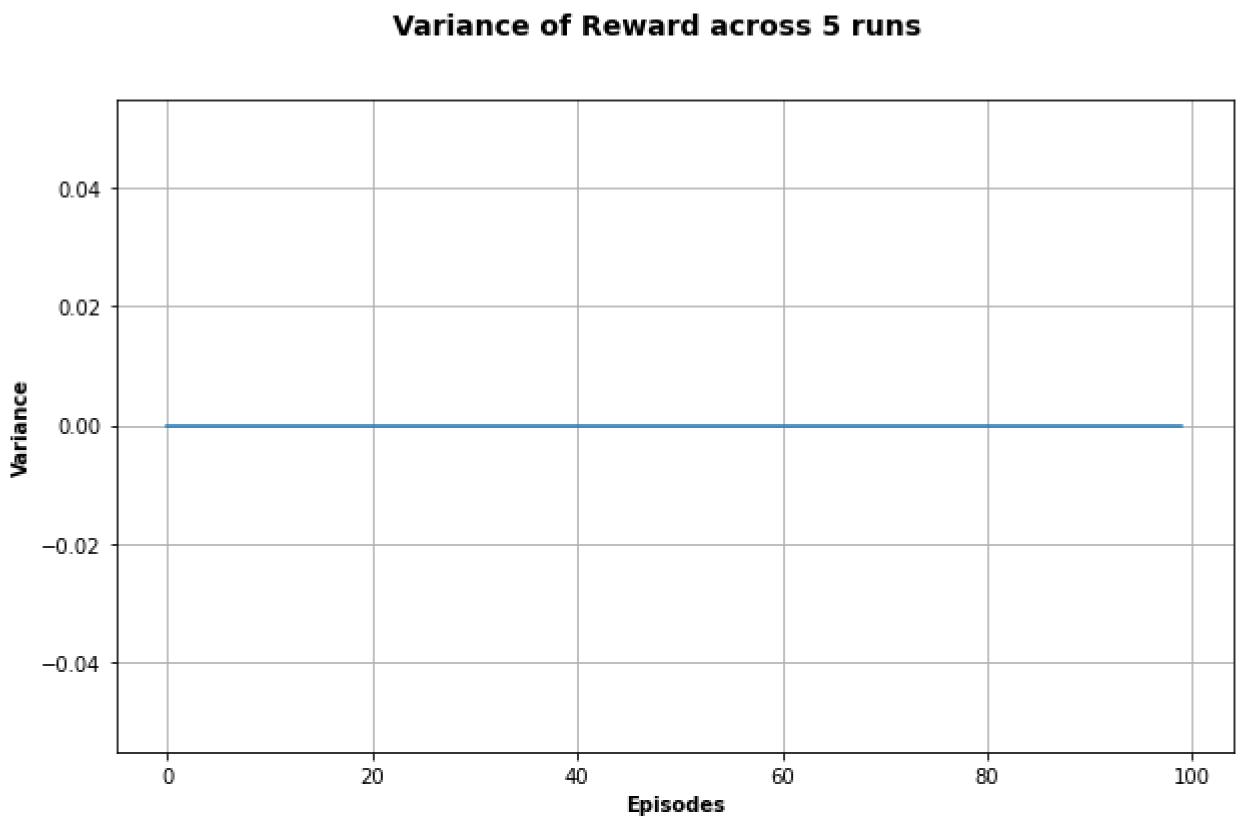
```
env = gym.make('MountainCar-v0')

#Initializing Agent
#Number of episodes
episodes = 100

average_reward_list_list=[]
total_reward_list=[]
begin_time = datetime.datetime.now()

for i in range(4):
  average_reward_list=[]
  reward_list=[]
  agent = Agent(lr=0.00001, action_size=env.action_space.n, n_hidden1=512, n_hidden2=512)
  tf.compat.v1.reset_default_graph()
  for ep in range(1, episodes + 1):
      state = env.reset().reshape(1,-1)
      done = False
      ep_rew = 0
      while not done:
          action = agent.sample_action(state) ##Sample Action
          next_state, reward, done, info = env.step(action) ##Take action
          next_state = next_state.reshape(1,-1)
          ep_rew += reward  ##Updating episode reward
          agent.learn(state, action, reward, next_state, done) ##Update Parameters
          state = next_state ##Updating State
      reward_list.append(ep_rew)
      average_reward_list.append(np.mean(reward_list[-100:]))

      if ep % 10 == 0:
          avg_rew = np.mean(reward_list[-10:])
          print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' % avg_rew)

      if ep>100 and ep % 100:
          avg_100 =  np.mean(reward_list[-100:])
          if avg_100 > -90.0:
              print('Stopped at Episode ',ep-100)
              break
  total_reward_list.append(np.sum(reward_list))
  average_reward_list_list.append(average_reward_list)
average_reward_list=tolerant_mean(average_reward_list_list)[0]
variance_reward_list=tolerant_mean(average_reward_list_list)[1]
```

```
time_taken = datetime.datetime.now() - begin_time
print(time_taken)
```

```
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
0:11:55.498551
```

```
### Plot of total reward vs episode
## Write Code Below
plt.figure(figsize=(10,6))
plt.plot(average_reward_list)
plt.suptitle("One Step Actor Critic", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Reward", fontweight = 'bold')
plt.grid(True)
plt.show()
plt.figure(figsize=(10,6))
plt.plot(variance_reward_list)
plt.suptitle("Variance of Reward across 5 runs", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Variance", fontweight = 'bold')
plt.grid(True)
plt.show()
```





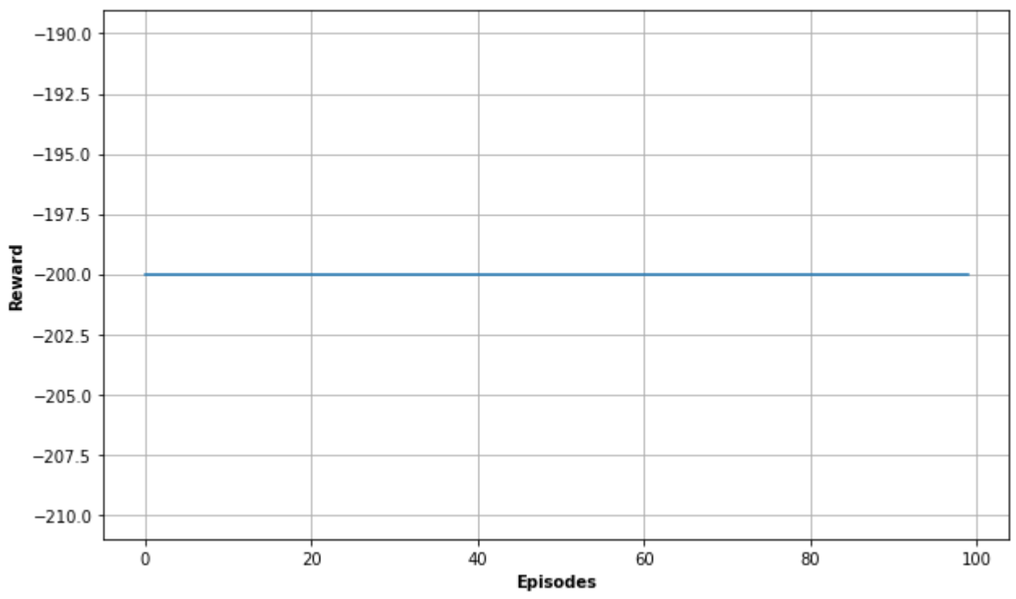
Full Returns

```
env = gym.make('MountainCar-v0')

#Initializing Agent
#Number of episodes
episodes = 100

average_reward_list_list=[]
```

```
total_reward_list=[]
begin_time = datetime.datetime.now()

for i in range(4):
  average_reward_list=[]
  reward_list=[]
  agent = AgentFullReturn(lr=0.00001, action_size=env.action_space.n, n_hidden1=512, n_hidden2=512)
  tf.compat.v1.reset_default_graph()
  for ep in range(1, episodes + 1):
      state = env.reset().reshape(1,-1)
      done = False
      ep_rew = 0
      state_trajectories = []
      actions = []
      reward_count = []
      iter = 0
      while not done and iter < 1000:
          action = agent.sample_action(state) ##Sample Action
          next_state, reward, done, info = env.step(action) ##Take action
          next_state = next_state.reshape(1,-1)
          ep_rew += reward  ##Updating episode reward
          state_trajectories.append(state)
          actions.append(action)
          reward_count.append(reward)
          #
          state = next_state ##Updating State
          iter+=1


      reward_list.append(ep_rew)
      average_reward_list.append(np.mean(reward_list[-100:]))

      state_trajectories = tf.stack(state_trajectories)
      actions = tf.stack(actions)
      reward_count = tf.stack(reward_count)
      agent.learn(state_trajectories, actions, reward_count) ##Update Parameters

      if ep % 10 == 0:
          avg_rew = np.mean(reward_list[-10:])
          print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' % avg_rew)

      if ep>100 and ep % 100:
          avg_100 =  np.mean(reward_list[-100:])
          if avg_100 > -90.0:
              print('Stopped at Episode ',ep-100)
              break
  total_reward_list.append(np.sum(reward_list))
  average_reward_list_list.append(average_reward_list)
average_reward_list=tolerant_mean(average_reward_list_list)[0]
variance_reward_list=tolerant_mean(average_reward_list_list)[1]
time_taken = datetime.datetime.now() - begin_time
print(time_taken)
```

```
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
0:14:04.684317
```

```
### Plot of total reward vs episode
## Write Code Below
plt.figure(figsize=(10,6))
plt.plot(average_reward_list)
plt.suptitle("One Step Actor Critic", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Reward", fontweight = 'bold')
plt.grid(True)
plt.show()
plt.figure(figsize=(10,6))
plt.plot(variance_reward_list)
plt.suptitle("Variance of Reward across 5 runs", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Variance", fontweight = 'bold')
plt.grid(True)
plt.show()
```



n-step returns

```
env = gym.make('MountainCar-v0')

#Initializing Agent
#Number of episodes
episodes = 100

average_reward_list_list=[]
total_reward_list=[]
begin_time = datetime.datetime.now()

for i in range(4):
  average_reward_list=[]
  reward_list=[]
  agent = AgentNStep(lr=0.00001, action_size=env.action_space.n, n_hidden1=512, n_hidden2=512)
tf.compat.v1.reset_default_graph()
for ep in range(1, episodes + 1):
    state = env.reset().reshape(1,-1)
    done = False
    ep_rew = 0
    state_trajectories = []
    actions = []
    reward_count = []
    iter = 0
    while not done and iter < 1000:
        action = agent.sample_action(state) ##Sample Action
        next_state, reward, done, info = env.step(action) ##Take action
        next_state = next_state.reshape(1,-1)
        ep_rew += reward  ##Updating episode reward
        state_trajectories.append(state)
        actions.append(action)
        reward_count.append(reward)
        #
        state = next_state ##Updating State
        iter+=1


    reward_list.append(ep_rew)
    average_reward_list.append(np.mean(reward_list[-100:]))

    state_trajectories = tf.stack(state_trajectories)
    actions = tf.stack(actions)
    reward_count = tf.stack(reward_count)
    agent.learn(state_trajectories, actions, reward_count) ##Update Parameters

    if ep % 10 == 0:
        avg_rew = np.mean(reward_list[-10:])
        print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' % avg_rew)

    if ep>100 and ep % 100:
        avg_100 =  np.mean(reward_list[-100:])
        if avg_100 > -90.0:
            print('Stopped at Episode ',ep-100)
            break
  total_reward_list.append(np.sum(reward_list))
  average_reward_list_list.append(average_reward_list)
average_reward_list=tolerant_mean(average_reward_list_list)[0]
variance_reward_list=tolerant_mean(average_reward_list_list)[1]
time_taken = datetime.datetime.now() - begin_time
print(time_taken)
```

```
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
```

```
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
0:19:21.564483
```

```
### Plot of total reward vs episode
## Write Code Below
plt.figure(figsize=(10,6))
plt.plot(average_reward_list)
plt.suptitle("One Step Actor Critic", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Reward", fontweight = 'bold')
plt.grid(True)
plt.show()
plt.figure(figsize=(10,6))
plt.plot(variance_reward_list)
plt.suptitle("Variance of Reward across 5 runs", fontweight = 'bold', fontsize = 14)
plt.xlabel("Episodes", fontweight = 'bold')
plt.ylabel("Variance", fontweight = 'bold')
plt.grid(True)
plt.show()
```