Department of Computer Science and Engineering (Data Science)

Subject: Reinforcement Learning (DJ19DSC502)

AY: 2022 - 23

Experiment 3

REINFORCE ALGORITHM

Aim: Implement REINFORCE algorithm on CartPole/Lunar Ladder

Theory:

A policy is defined as the probability distribution of actions given a state. The objective of the policy is to maximize the "Expected reward".

The policy is usually a Neural Network that takes the state as input and generates a probability distribution across action space as output.

REINFORCE Algorithm

REINFORCE belongs to a special class of Reinforcement Learning algorithms called Policy Gradient algorithms. The objective is to maximize the "expected" reward when following a policy π.

$$\text{Expected Reward } \mathbf{G}(\theta) = \sum_{i=0}^{n} \text{Probability of action}_k \text{ at state}_i * \text{discounted reward}_{(k,i)}$$

$$\theta = \text{Policy}$$

$$n = \text{Number of steps in the episode}$$

$$k = \text{index of action}$$

Here the discounted reward is the sum of all the rewards the agent receives in that future discounted by a factor Gamma.

$$\text{Discounted Reward at } t, R(t) = r(t) + \gamma * R(t+1) = r(t) + \gamma * r(t+1) + \gamma^2 * r(t+2) + \ldots + \gamma^{(T-t)} * r(T)$$

$$\gamma = \text{Discount Factor, usually } 0.9$$

$$T = \text{Terminal state's time step}$$

The discounted reward at any stage is the reward it receives at the next step + a discounted sum of all rewards the agent receives in the future.

The REINFORCE gradient can be given as follows:

$$\nabla \mathbb{E}_{\pi_\theta}\left[r(\tau)\right] = \mathbb{E}_{\pi_\theta}\left[\left(\sum_{t=1}^{T} G_t \nabla \log \pi_\theta(a_t|s_t)\right)\right]$$

Where G_t represents the discounted reward

For practical implementation, let us consider the policy as NN. The agent samples from these probabilities provided by the policy and selects an action to perform in the environment. At the end of an episode, we know the total rewards the agent can get if it follows that policy. We backpropagate the reward through the path the agent took to estimate the "Expected reward" at each state for a given policy.

Algorithm:

Algorithm 2.1 REINFORCE algorithm

Initialize learning rate α
Initialize weights θ of a policy network π_θ
for $episode = 0, \ldots, MAX\_EPISODE$ **do**
  Sample a trajectory $\tau = s_0, a_0, r_0, \ldots, s_T, a_T, r_T$
  Set $\nabla_\theta J(\pi_\theta) = 0$
  for $t = 0, \ldots, T$ **do**
    $R_t(\tau) = \sum_{t'=t}^{T} \gamma^{t'-t} r'_t$ ← $R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t r_t$
    $\nabla_\theta J(\pi_\theta) = \nabla_\theta J(\pi_\theta) + R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$
  end for
  $\theta = \theta + \alpha \nabla_\theta J(\pi_\theta)$
end for

The steps involved in the implementation of REINFORCE would be as follows:

1. *Initialize a Random Policy (a NN that takes the state as input and returns the probability of actions)*
2. *Use the policy to play N steps of the game — record action probabilities-from policy, reward-from environment, action — sampled by agent*
3. *Calculate the discounted reward for each step by backpropagation*
4. *Calculate expected reward G*
5. *Adjust weights of Policy (back-propagate error in NN) to increase G*
6. *Repeat from 2*

CartPole Environment:

A CartPole-v0 is a simple playground provided by OpenAI to train and test Reinforcement Learning algorithms. The agent is the cart, controlled by two possible actions +1, -1 pointing on moving left or right.

The reward +1 is given at every timestep if the pole remains upright. The goal is to prevent the pole from falling over (maximize total reward). After 100 consecutive timesteps and an average reward of

195, the problem is considered solved.

The episode ends when the pole is more than 15 degrees from vertical or the cart moves more than 2.4 units from the center.

For more information on the environment, visit CartPole

Lab Assignment To Do:

1) Explore the CartPole environment.
2) Train a policy using REINFORCE algorithm.
3) Visualize the trained policy in the CartPole environment.

Sarvayga Singh

60009200030 - K1

Experiment 3 - : Implement REINFORCE algorithm on CartPole/Lunar Ladder

In [42]:
```
!pip install gymnasium[box2d]
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gymnasium[box2d] in /usr/local/lib/python3.9/dist-packages (0.27.1)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.9/dist-packages (from gymnasium[box2d]) (2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.9/dist-packages (from gymnasium[box2d]) (4.5.0)
Requirement already satisfied: importlib-metadata>=4.8.0 in /usr/local/lib/python3.9/dist-packages (from gymnasium[box2d]) (6.0.0)
Requirement already satisfied: jax-jumpy>=0.2.0 in /usr/local/lib/python3.9/dist-packages (from gymnasium[box2d]) (1.0.0)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.9/dist-packages (from gymnasium[box2d]) (1.22.4)
Requirement already satisfied: gymnasium-notices>=0.0.1 in /usr/local/lib/python3.9/dist-packages (from gymnasium[box2d]) (0.0.1)
Collecting pygame==2.1.3.dev8
  Downloading pygame-2.1.3.dev8-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.7/13.7 MB 67.5 MB/s eta 0:00:00
Collecting box2d-py==2.3.5
  Downloading box2d-py-2.3.5.tar.gz (374 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 374.4/374.4 KB 31.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting swig==4.*
  Downloading swig-4.1.1-py2.py3-none-manylinux_2_5_x86_64.manylinux1_x86_64.whl (1.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 85.9 MB/s eta 0:00:00
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-packages (from importlib-metadata>=4.8.0->gymnasium[box2d]) (3.15.0)
Building wheels for collected packages: box2d-py
  error: subprocess-exited-with-error

  × python setup.py bdist_wheel did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
  Building wheel for box2d-py (setup.py) ... error
  ERROR: Failed building wheel for box2d-py
  Running setup.py clean for box2d-py
Failed to build box2d-py
Installing collected packages: swig, box2d-py, pygame
  Running setup.py install for box2d-py ... done
  DEPRECATION: box2d-py was installed using the legacy 'setup.py install' method, because a wheel could not be built for it. A possible replacement is to fix the wheel build issue reported above. Discussion can be found at https://github.com/pypa/pip/issues/8368
Successfully installed box2d-py-2.3.5 pygame-2.1.3.dev8 swig-4.1.1
```

In [25]:
```
import gym
import numpy as np
from collections import deque
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (16, 10)

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.distributions import Categorical
torch.manual_seed(0)

import base64, io

# For visualization
from gym.wrappers.monitoring import video_recorder
from IPython.display import HTML
from IPython import display
import glob
```

In [27]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device
```

Out[27]:
```
device(type='cuda', index=0)
```

In [26]:
```
env = gym.make('CartPole-v0')
env.seed(0)

print('observation space:', env.observation_space)
print('action space:', env.action_space)
```

```
observation space: Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38], (4,), float3
2)
action space: Discrete(2)
```

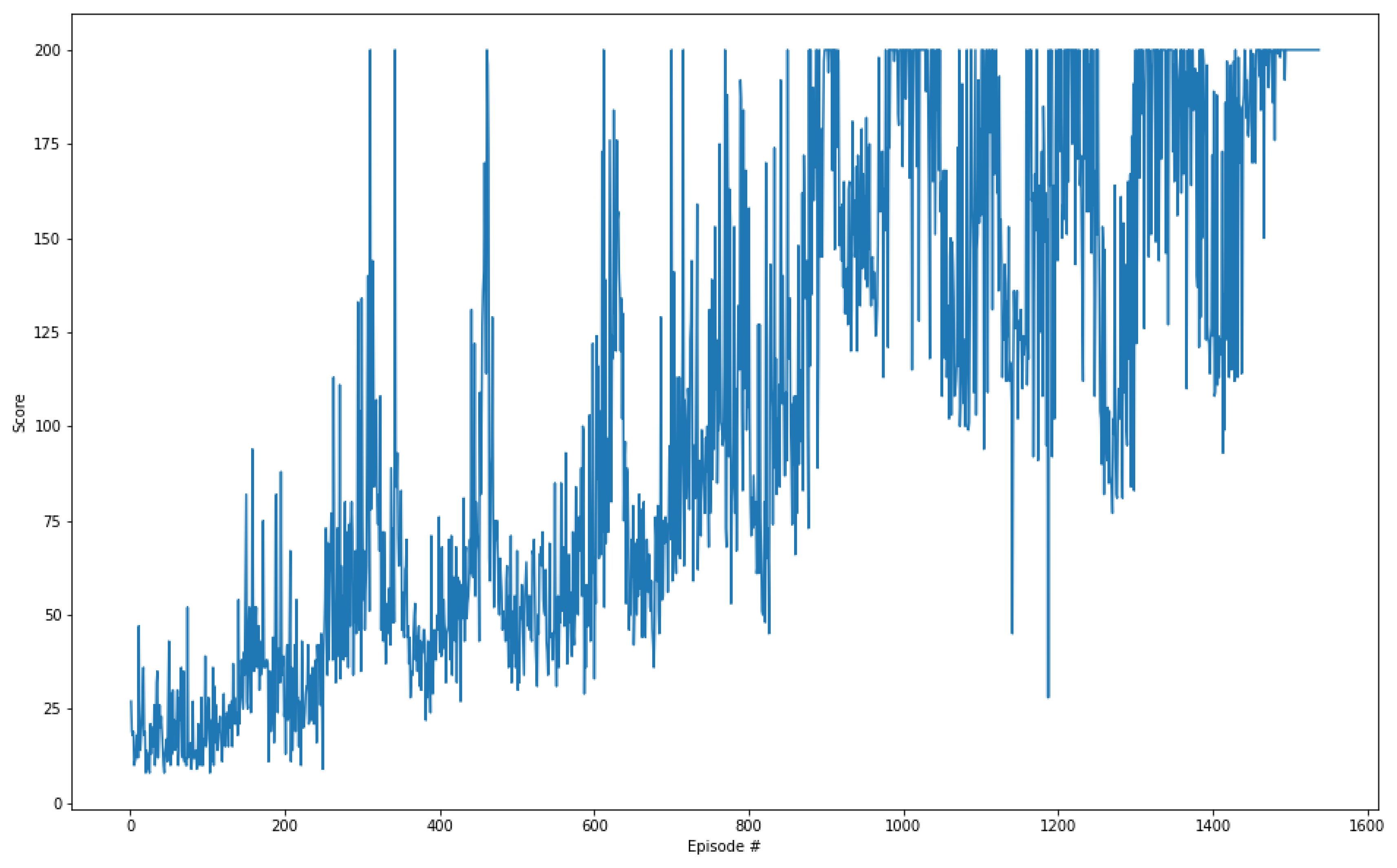
In [28]:
```
class Policy(nn.Module):
    def __init__(self, state_size=4, action_size=2, hidden_size=32):
        super(Policy, self).__init__()
        self.fc1 = nn.Linear(state_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, action_size)

    def forward(self, state):
        x = F.relu(self.fc1(state))
        x = self.fc2(x)
        # we just consider 1 dimensional probability of action
        return F.softmax(x, dim=1)

    def act(self, state):
        state = torch.from_numpy(state).float().unsqueeze(0).to(device)
        probs = self.forward(state).cpu()
        model = Categorical(probs)
        action = model.sample()
        return action.item(), model.log_prob(action)
```

In [29]:
```
def reinforce(policy, optimizer, n_episodes=1000, max_t=1000, gamma=1.0, print_every=100):
    scores_deque = deque(maxlen=100)
    scores = []
    for e in range(1, n_episodes):
        saved_log_probs = []
        rewards = []
        state = env.reset()
        # Collect trajectory
        for t in range(max_t):
            # Sample the action from current policy
            action, log_prob = policy.act(state)
            saved_log_probs.append(log_prob)
            state, reward, done, _ = env.step(action)
            rewards.append(reward)
            if done:
                break
        # Calculate total expected reward
        scores_deque.append(sum(rewards))
        scores.append(sum(rewards))

        # Recalculate the total reward applying discounted factor
        discounts = [gamma ** i for i in range(len(rewards) + 1)]
        R = sum([a * b for a,b in zip(discounts, rewards)])

        # Calculate the loss
        policy_loss = []
        for log_prob in saved_log_probs:
            # Note that we are using Gradient Ascent, not Descent. So we need to calculate it with negative rewards.
            policy_loss.append(-log_prob * R)
        # After that, we concatenate whole policy loss in 0th dimension
        policy_loss = torch.cat(policy_loss).sum()

        # Backpropagation
        optimizer.zero_grad()
        policy_loss.backward()
        optimizer.step()

        if e % print_every == 0:
            print('Episode {}\tAverage Score: {:.2f}'.format(e, np.mean(scores_deque)))
        if np.mean(scores_deque) >= 195.0:
            print('Environment solved in {:d} episodes!\tAverage Score: {:.2f}'.format(e - 100, np.mean(scores_deque)))
            break
    return scores
```

In [30]:
```
policy = Policy().to(device)
optimizer = optim.Adam(policy.parameters(), lr=1e-2)
scores = reinforce(policy, optimizer, n_episodes=2000)
```

```
Episode 100	Average Score: 18.38
Exception ignored in: <function VideoRecorder.__del__ at 0x7fb1a9b940d0>
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/gym/wrappers/monitoring/video_recorder.py", line 269, in __del__
    self.close()
  File "/usr/local/lib/python3.9/dist-packages/gym/wrappers/monitoring/video_recorder.py", line 228, in close
    if self.encoder:
AttributeError: 'VideoRecorder' object has no attribute 'encoder'
Episode 200	Average Score: 32.82
Episode 300	Average Score: 45.29
Episode 400	Average Score: 61.26
Episode 500	Average Score: 67.80
Episode 600	Average Score: 54.74
Episode 700	Average Score: 84.57
Episode 800	Average Score: 106.12
Episode 900	Average Score: 117.23
Episode 1000	Average Score: 166.28
Episode 1100	Average Score: 162.56
Episode 1200	Average Score: 148.20
Episode 1300	Average Score: 148.28
Episode 1400	Average Score: 176.78
Episode 1500	Average Score: 176.77
Environment solved in 1437 episodes!	Average Score: 195.46
```

In [31]:
```
# Plot the learning progress
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(1, len(scores)+1), scores)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```



In [37]:
```
# Animate it with Video
def show_video(env_name):
    mp4list = glob.glob('video/*.mp4')
    if len(mp4list) > 0:
        mp4 = 'video/{}.mp4'.format(env_name)
        video = io.open(mp4, 'r+b').read()
        encoded = base64.b64encode(video)
        display.display(HTML(data='''<video alt="test" autoplay
                loop controls style="height: 400px;">
                <source src="data:video/mp4;base64,{0}" type="video/mp4" />
             </video>'''.format(encoded.decode('ascii'))))
    else:
        print("Could not find video")

def show_video_of_model(policy, env_name):
    env = gym.make(env_name)
    vid = video_recorder.VideoRecorder(env, path="video/{}.mp4".format(env_name))
    state = env.reset()
    done = False
    for t in range(1000):
        vid.capture_frame()
        action, _ = policy.act(state)
        next_state, reward, done, _ = env.step(action)
        state = next_state
        if done:
            break
    vid.close()
    # env.close()
```

In [ ]:
```
show_video_of_model(policy, 'CartPole-v0')
```

In [ ]:
```
show_video('CartPole-v0')
```

In [44]:
```
# import gymnasium as gym
# env = gym.make("LunarLander-v2", render_mode="human")
# observation, info = env.reset(seed=42)
# for _ in range(1000):
#    action = env.action_space.sample()  # this is where you would insert your policy
#    observation, reward, terminated, truncated, info = env.step(action)

#    if terminated or truncated:
#       observation, info = env.reset()
# env.close()
```

In [ ]: