BU.330.775 Machine Learning: Design and Deployment

**Lab 7. Reinforcement Learning using Gymnasium**

Learning Goal: practice simple reinforcement learning techniques to balance the pole using the Gymnasium package

1. First, we import packages and configure matplotlib's global styles, set the font size, axis label size, legend and tick label font size. We will use animation today, and specify the animation format as `jshtml` to display the animation in Jupyter Notebook.

import sys

import matplotlib.animation

import matplotlib.pyplot as plt

plt.rc('font', size=14)

plt.rc('axes', labelsize=14, titlesize=14)

plt.rc('legend', fontsize=14)

plt.rc('xtick', labelsize=10)

plt.rc('ytick', labelsize=10)

plt.rc('animation', html='jshtml')

1. Reinforcement learning is typically computationally expensive. So, let’s use GPU if there is one. Go to Runtime > Change runtime and select a GPU hardware.
2. Let's install the Gymnasium library, which provides many environments for Reinforcement Learning. We'll also install the extra libraries needed for classic control environments (including CartPole, which we will use shortly), as well as for Box2D and Atari environments, which are needed for the exercises. By running the following scripts, you accept the Atari ROM license.

%pip install -q -U gymnasium swig

%pip install -q -U gymnasium[classic\_control,box2d,atari,accept-rom-license]

1. Import Gym and make a new CartPole environment. The CartPole (version 1) is a very simple environment composed of a cart that can move left or right, and pole placed vertically on top of it. The agent must move the cart left or right to keep the pole upright.

import gymnasium as gym

env = gym.make("CartPole-v1", render\_mode="rgb\_array")

1. Let's initialize the environment by calling its reset() method. This returns an observation, as well as a dictionary that may contain extra information. In the case of the CartPole, each observation is a 1D NumPy array composed of 4 floats: they represent the cart's horizontal position, its velocity, the angle of the pole (0 = vertical), and the pole’s angular velocity.

obs, info = env.reset(seed=42)

print(obs, info)

1. An environment can be visualized by calling its render() method. Then, we define a `plot\_environment` function that renders and displays an image of the Gymnasium environment. The function creates a graphics window, obtains the environment image through `env.render()`, and displays it in an image without coordinate axes. After calling this function, the output shows an image of the cart and pole in the CartPole environment.

img = env.render()

# creates a little function to render and plot an environment

def plot\_environment(env, figsize=(5, 4)):

plt.figure(figsize=figsize)

img = env.render()

plt.imshow(img)

plt.axis("off")

return img

plot\_environment(env)

plt.show()

1. Let's see how to interact with an environment. Your agent will need to select an action from an "action space" (the set of possible actions). There are two possible actions: accelerate towards the left (0) or towards the right (1).

env.action\_space

1. Since the pole is leaning toward the right (obs[2] > 0), let's accelerate the cart toward the right.

action = 1 # accelerate right

obs, reward, done, truncated, info = env.step(action)

obs

1. the cart is now moving toward the right (obs[1] > 0). The pole is still tilted toward the right (obs[2] > 0), but its angular velocity is now negative (obs[3] < 0). Display the new environment.

plot\_environment(env)

plt.show()

1. The environment also tells the agent how much reward it got during the last step, whether the game is over, whether the environment is interrupted, and some extra information. For example, in some games it may indicate how many lives the agent has.

Since the goal is to keep the pole upright for as long as possible, by default, a reward of +1 is given for every step taken, including the termination step. The default reward threshold is 500 for v1.

print(f'reward: {reward}')

print(f'done: {done}')

print(f'truncated: {truncated}')

print(f'info: {info}')

1. Now we will define a *policy* for that. This is the strategy that the agent will use to select an action at each step. It can use all the past actions and observations to decide what to do.

Let's hard code a simple strategy: if the pole is tilting to the left, then push the cart to the left, and vice versa. We will try it for 500 different episodes. For every episode, we will take 200 steps of actions.

def basic\_policy(obs):

angle = obs[2]

return 0 if angle < 0 else 1

totals = []

for episode in range(500):

episode\_rewards = 0

obs, info = env.reset(seed=episode)

for step in range(200):

action = basic\_policy(obs)

obs, reward, done, truncated, info = env.step(action)

episode\_rewards += reward

if done or truncated:

break

totals.append(episode\_rewards)

1. Compute the performance of these 500 episodes using rewards, and its minimum, and maximum values.

import numpy as np

totals=np.array(totals)

totals, min(totals), max(totals)

1. As expected, this strategy is a bit too basic: the best it did was to keep the pole up for only 63 steps. Let's visualize one episode.

def update\_scene(num, frames, patch):

patch.set\_data(frames[num])

return patch,

def plot\_animation(frames, repeat=False, interval=40):

fig = plt.figure()

patch = plt.imshow(frames[0])

plt.axis('off')

anim = matplotlib.animation.FuncAnimation(

fig, update\_scene, fargs=(frames, patch),

frames=len(frames), repeat=repeat, interval=interval)

plt.close()

return anim

def show\_one\_episode(policy, n\_max\_steps, seed):

frames = []

env = gym.make("CartPole-v1", render\_mode="rgb\_array")

obs, info = env.reset(seed=seed)

for step in range(n\_max\_steps):

frames.append(env.render())

action = policy(obs)

obs, reward, done, truncated, info = env.step(action)

if done or truncated:

break

env.close()

return plot\_animation(frames)

1. Now call the function with 100 max steps and episode (seed) 333.

show\_one\_episode(basic\_policy, 100, 333)

**Feel free to try different episode. This is the end of Lab 7. There is no homework.**