my_REINFORCE

March 12, 2022

1 CE 40719: Deep Learning

1.1 HW6-Q5: REINFORCE (with baseline)

Full name:

STD-ID:

In this notebook, you are going to implement REINFORCE algorithm on `CartPole-v0` and compare it to the case with a baseline. To know more about this, please refer to Sutton&Barto, 13.3-13.4.

[1]:
```
%%bash
pip install gym pyvirtualdisplay > /dev/null 2>&1
apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1
```

[2]:
```
import gym
import torch
from torch import nn
import torch.nn.functional as F
import numpy as np
from tqdm import tqdm
from gym.wrappers import Monitor
import glob
import io
import base64
from IPython.display import HTML
from pyvirtualdisplay import Display
from IPython import display as ipythondisplay
import matplotlib.pyplot as plt
import warnings


warnings.filterwarnings('ignore')
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

2 Auxiliary methods

You can use the following methods to display demos and results of your code.

```
[3]: root = '/content/video'
display = Display(visible=0, size=(200, 150))
display.start()

def show_video(path=root):
    mp4list = glob.glob(f'{path}/*.mp4')
    if len(mp4list) > 0:
        mp4 = mp4list[0]
        video = io.open(mp4, 'r+b').read()
        encoded = base64.b64encode(video)
        ipythondisplay.display(HTML(data='''<video alt="test" autoplay
                    loop controls style="height: 250px;">
                    <source src="data:video/mp4;base64,{0}" type="video/mp4" />
                    </video>'''.format(encoded.decode('ascii'))))
    else:
        print("Could not find video")

def wrap_env(env, path=root):
    return Monitor(env, path, force=True)

def plot_curves(curves, title, smooth=True, w_size=50):
    """
    This method plots series specified in `curves['series']`
    inside the same figure.

    - curves: a dictionary, dict(curves=a list of lists, labels=a list of␣
 ↪strings);
    - title: figure's title;
    - smooth: whether to take a moving average over each series;
    - w_size: size of the moving average window;

    Notice: Series must have the same length.
    """
    series, labels = curves['series'], curves['labels']
    N = len(series[0])
    assert all([len(s) == N for s in series])
    x = list(range(N))
    for s, label in zip(series, labels):
        window = np.ones(w_size)/w_size
        s_smooth = np.convolve(s, window, mode='same')
        y = s_smooth[w_size:N-w_size] if smooth else s
        plt.plot(x[w_size:N-w_size], y, label=label)
    plt.legend()
    plt.title(title)
    plt.show()
```

3 CartPole-v0

You can see specifications of `CartPole-v0` in the following cell.

[4]:
```
env_id = 'CartPole-v0'
env = gym.make(env_id)
spec = gym.spec(env_id)

print(f"Action Space: {env.action_space}")
print(f"Observation Space: {env.observation_space}")
print(f"Max Episode Steps: {spec.max_episode_steps}")
print(f"Nondeterministic: {spec.nondeterministic}")
print(f"Reward Range: {env.reward_range}")
print(f"Reward Threshold: {spec.reward_threshold}\n")
```

```
Action Space: Discrete(2)
Observation Space: Box(-3.4028234663852886e+38, 3.4028234663852886e+38, (4,),
float32)
Max Episode Steps: 200
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: 195.0
```

Here you can see a demo of the completely random policy.

[5]:
```
env = gym.make('CartPole-v0')
env = wrap_env(env)
state = env.reset()
done = False
while not done:
    action = env.action_space.sample()
    state, reward, done, _ = env.step(action)
env.close()
show_video()
```

```
<IPython.core.display.HTML object>
```

4 Method

You can either use `Net` to create baseline and policy networks, or use any other custom architecture.

```
[6]: class Net(nn.Module):
         def __init__(self, input_dim, hidden_dim, outdim_policy, outdim_baseline):
             super(Net, self).__init__()
             self.shared = nn.Linear(input_dim, hidden_dim)
             self.policy = nn.Linear(hidden_dim, outdim_policy)
             self.baseline = nn.Linear(hidden_dim, outdim_baseline)

         def forward(self, x):
             x = self.shared(x)
             x = F.relu(x)
             p = F.softmax(self.policy(x))
             b = self.baseline(x)
             return p, b
```

```
[12]: class REINFORCECartPole():
          def __init__(self, use_baseline=False, GAMMA=None, lr=None):
              self.env_id = 'CartPole-v0'
              self.env = gym.make(self.env_id)
              self.use_baseline = use_baseline
              self.GAMMA = GAMMA
              ############################ ToDo (1 points) #########################
              # Define your network, optimizer, and criterion.
              ######################################################################


          def generate_episode(self, video=False, train=True):
              trajectory = []
              ############################ ToDo (2 points) #########################
              # Generate a trajectory from the current policy. This method may be
              # used at training and evaluation time. Also you can record the demo
              # of the trajectory to display later.
              ######################################################################

              return trajectory


          def select_action(self, state, train=True):
              ############################ ToDo (4 points) #########################
              # Select action based on `state`. At training time, you should sample
              # from the policy distribution, but at test time, you need to takes
              # the best possible action.
              ######################################################################
              pass

          def train(self, n_episodes, n_eval_episodes=15):
              ############################ ToDo (10 points) ########################
              # Train your networks in the following loop. At the end of each
```

```
        # episode, evaluate your networks on `n_eval_episodes` episodes and
        # store average total return of them in `TRs`. You are going to plot
        # these TRs later.
        ######################################################################
        TRs = []
        for i in tqdm(range(n_episodes)):
            pass
        return TRs

    def evaluate(self, n_episodes):
        ############################ ToDo (2 points) #########################
        # Evaluate your networks on `n_episodes` episodes and return the
        # average **undiscounted** total return.
        ######################################################################
        pass

    def show_demo(self):
        ############################ ToDo (1 points) #########################
        # Display demo of one episode based on the current policy.
        ######################################################################
        pass
```

5 Results & conclusion

[13]:
```
# First you need to choose appropriate input values.
n = ...
lr = ...
GAMMA = ...
kwargs = dict(GAMMA=GAMMA, lr=lr)
```

[ ]:
```
reinforce = REINFORCECartPole(**kwargs)
returns_reinforce = reinforce.train(n)
reinforce.show_demo()
```

[ ]:
```
kwargs['use_baseline'] = True
reinforce_b = REINFORCECartPole(**kwargs)
returns_reinforce_b = reinforce_b.train(n)
reinforce_b.show_demo()
```

[ ]:
```
############################ ToDo (1 points) #########################
# Plot total return curves for both methods in the same figure.
######################################################################
```

Question: (4 points)

- Interpret your results. What is the difference between REINFORCE with baseline and without baseline?
- What is the difference between REINFORCE with baseline and Actor-Critic methods?