one-step-actor-critic-mountain-car

March 29, 2023

[2]:
```
'''
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.record_video import RecordVideo
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
```

[14]:
```
class ActorCriticModel(tf.keras.Model):
    """
    Defining policy and value networkss
    """
    def __init__(self, action_size, n_hidden1=400, n_hidden2=400):
        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
```

0.0.1 Agent Class

###**Task 2a:** Write code to compute δ_t inside the Agent.learn() function

[15]:
```
class Agent:
    """
    Agent class
    """
    def __init__(self, action_size, lr=0.00001, gamma=0.99, seed = 85):
        self.gamma = gamma
        self.ac_model = ActorCriticModel(action_size=action_size)
        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     ## Complete this
            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))
```

0.0.2 Train the Network

[17]:
```
env = gym.make('MountainCar-v0', max_episode_steps = 500)

#Initializing Agent
agent = Agent(lr=1e-5, action_size=env.action_space.n)
#Number of episodes
episodes = 300
tf.compat.v1.reset_default_graph()

reward_list = []
average_reward_list = []
begin_time = datetime.datetime.now()
```

```
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)

    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:
        avg_100 =  np.mean(reward_list[-100:])
        if avg_100 > -110.0:
            print('Stopped at Episode ',ep-100)
            break

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 -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 -500.000000
Episode  100 Reward -500.000000 Average Reward -500.000000
Episode  110 Reward -500.000000 Average Reward -500.000000
Episode  120 Reward -500.000000 Average Reward -500.000000
Episode  130 Reward -500.000000 Average Reward -500.000000
Episode  140 Reward -500.000000 Average Reward -500.000000
Episode  150 Reward -500.000000 Average Reward -500.000000
Episode  160 Reward -500.000000 Average Reward -500.000000
Episode  170 Reward -500.000000 Average Reward -500.000000
Episode  180 Reward -500.000000 Average Reward -500.000000
Episode  190 Reward -500.000000 Average Reward -500.000000
```

```
Episode  200 Reward -500.000000 Average Reward -500.000000
Episode  210 Reward -500.000000 Average Reward -500.000000
Episode  220 Reward -500.000000 Average Reward -500.000000
Episode  230 Reward -500.000000 Average Reward -500.000000
Episode  240 Reward -500.000000 Average Reward -500.000000
Episode  250 Reward -500.000000 Average Reward -500.000000
Episode  260 Reward -500.000000 Average Reward -500.000000
Episode  270 Reward -500.000000 Average Reward -500.000000
Episode  280 Reward -500.000000 Average Reward -500.000000
Episode  290 Reward -500.000000 Average Reward -500.000000
Episode  300 Reward -500.000000 Average Reward -500.000000
0:23:14.337485
```

[18]:
```
import matplotlib.pyplot as plt
plt.plot(reward_list)
```

[18]:
```
[<matplotlib.lines.Line2D at 0x7faab0d111c0>]
```