a2-full-rl-return-cartpole-acrobot

March 29, 2023

[1]:
```
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
import tensorflow as tf
from IPython import display as ipythondisplay
from PIL import Image
import tensorflow_probability as tfp
```

[2]:
```
class ActorCriticModel(tf.keras.Model):
    """
    Defining policy and value networkss
    """
    def __init__(self,␣
 ↪action_size,num_layers=2,num_neurons_each_layer=[1024,512]):
        super(ActorCriticModel, self).__init__()
        self.num_layers  =num_layers
        self.neurons_each_layer = num_neurons_each_layer
        self.linears = [tf.keras.layers.Dense(self.neurons_each_layer[0],␣
 ↪activation=tf.nn.relu)]
        self.linears.extend([tf.keras.layers.Dense(self.neurons_each_layer[i],␣
 ↪activation=tf.nn.relu) for i in range(1, self.num_layers)])

        #Output Layer for policy
        self.pi_out = tf.keras.layers.Dense(action_size, activation=tf.nn.
 ↪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
        """
        h = None
        for i in range(self.num_layers+1):
          if i == 0:
             h = tf.nn.relu(self.linears[0](state))
          elif i < self.num_layers:
             h = tf.nn.relu(self.linears[i](h))
          else:
             return self.pi_out(h), self.v_out(h)
```

```
/usr/local/lib/python3.9/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[3]:
```
class Agent:
    """Agent( action_size=env.action_space.n, num_layers=num_layers,␣
↪num_neurons_each_layer=num_neurons_each_layer, lr=LR,seed = seed)
    Agent class
    """
    def __init__(self, action_size, num_layers, num_neurons_each_layer, lr=0.
↪001, gamma=0.99, seed = 85):
        self.gamma = gamma
        self.ac_model =␣
↪ActorCriticModel(action_size=action_size,num_layers=num_layers,num_neurons_each_layer=num_n
        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))

    def learn_full_return(self, states, actions, rewards, next_states, dones,T):
      with tf.GradientTape(persistent=True) as tape:
          delta_ts = np.empty(T)
          pis = []
          for t in range(0,T):
            delta_t = 0
            for t_ in range(t,T):
              delta_t += (((self.gamma)**(t_-t))*rewards[t_+1] if t_+1<␣
↪len(rewards) else 0)
```

```
            pi, V_s = self.ac_model(states[t])
            V_s = tf.squeeze(V_s)
            pi, V_s = self.ac_model(states[t])
            delta_t = delta_t - V_s
            delta_ts[t] = (delta_t)
            pis.append(pi)

          loss_a = 0
          loss_c = 0
          for i in range(T):
            loss_a += self.actor_loss(actions[i], pis[i], delta_ts[i])
            loss_c +=self.critic_loss(delta_ts[i])
          loss_total = loss_a + loss_c

      gradient = tape.gradient(loss_total, self.ac_model.trainable_variables,␣
 ↪unconnected_gradients=tf.UnconnectedGradients.ZERO)
      self.ac_model.optimizer.apply_gradients(zip(gradient, self.ac_model.
 ↪trainable_variables))
```

[4]:
```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)


def Actor_critic_full_return(num_layers, num_neurons_each_layer,␣
 ↪LR,num_episodes,seed):
  env = gym.make('MountainCar-v0')

  #Initializing Agent
  agent = Agent( action_size=env.action_space.n, num_layers=num_layers,␣
 ↪num_neurons_each_layer=num_neurons_each_layer, lr=LR,seed = seed)
  #Number of episodes
  episodes = num_episodes
  tf.compat.v1.reset_default_graph()

  reward_list = []
  average_reward_list = []
  steps = []
  begin_time = datetime.datetime.now()
  ep_solved = num_episodes
  for ep in range(1, episodes + 1):
      state = env.reset().reshape(1,-1)
      done = False
      ep_rew = 0
      T = 0
      states,actions,next_states,rewards,dones = [],[],[],[0],[None]
      while not done:
```

```
            T+=1
            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
            states.append(state)
            actions.append(action)
            next_states.append(next_state)
            rewards.append(reward)
            dones.append(done)
            state = next_state ##Updating State
        agent.learn_full_return(states, actions, rewards, next_states, dones,T)
        reward_list.append(ep_rew)
        steps.append(T)
        if ep % 10 == 0:
            avg_rew = np.mean(reward_list[-10:])
            average_reward_list.append(avg_rew)
            print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' %␣
    ↪avg_rew)

        if ep>1000 and np.std(reward_list[-500:])*100/np.mean(reward_list[-500:])␣
    ↪< 5:
            ep_solved = num_episodes
            break

        if ep % 100:
            avg_100 =  np.mean(reward_list[-100:])
            if avg_100 > -100.0:
                print('Stopped at Episode ',ep-100)
                ep_solved = ep-100
                break
    time_taken = datetime.datetime.now() - begin_time
    print(time_taken)
    return [reward_list,steps,ep_solved]
```

[5]:
```
LR = 1e-5
NUM_NEURONS_EACH_LAYER = [18,36,18]
NUM_LAYERS = 3
NUM_EPISODES = 200
rew,ste,eps = Actor_critic_full_return(NUM_LAYERS, NUM_NEURONS_EACH_LAYER,␣
 ↪LR,NUM_EPISODES,np.random.randint(1,200))
```

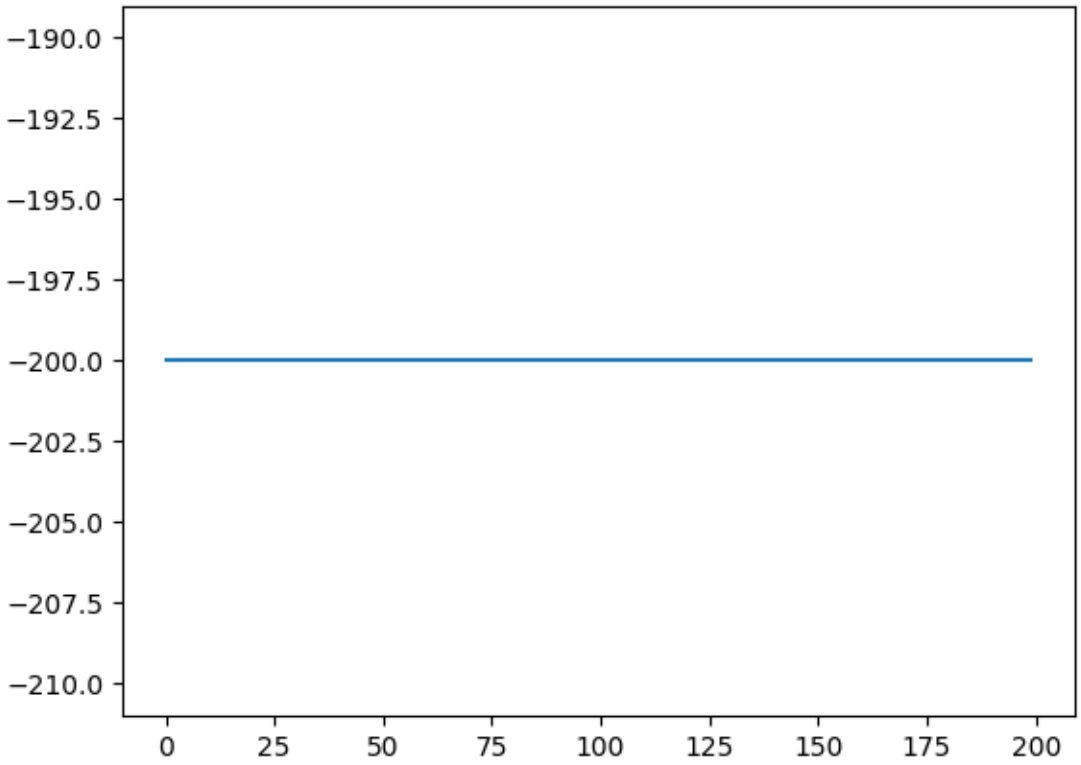
```
/usr/local/lib/python3.9/dist-packages/gym/core.py:317: DeprecationWarning:
WARN: Initializing wrapper in old step API which returns one bool instead
of two. It is recommended to set `new_step_api=True` to use new step API. This
will be the default behaviour in future.
```

```
  deprecation(
/usr/local/lib/python3.9/dist-
packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning:
WARN: Initializing environment in old step API which returns one bool

instead of two. It is recommended to set `new_step_api=True` to use new step

API. This will be the default behaviour in future.
  deprecation(

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  110 Reward -200.000000 Average Reward -200.000000
Episode  120 Reward -200.000000 Average Reward -200.000000
Episode  130 Reward -200.000000 Average Reward -200.000000
Episode  140 Reward -200.000000 Average Reward -200.000000
Episode  150 Reward -200.000000 Average Reward -200.000000
Episode  160 Reward -200.000000 Average Reward -200.000000
Episode  170 Reward -200.000000 Average Reward -200.000000
Episode  180 Reward -200.000000 Average Reward -200.000000
Episode  190 Reward -200.000000 Average Reward -200.000000
Episode  200 Reward -200.000000 Average Reward -200.000000
0:11:20.675324
```

[6]:
```
plt.plot(rew)
```

```
/usr/local/lib/python3.9/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[6]:
```
[<matplotlib.lines.Line2D at 0x7f35149dd040>]
```


−190.0
−192.5
−195.0
−197.5
−200.0
−202.5
−205.0
−207.5
−210.0
0
25
50
75
100
125
150
175
200