Author: Aditya Jain

About : Assignment 4, APS1080 (Introduction to RL)

Topic : Function Approximation and SARSA Control

In [2]:

```
import matplotlib.pyplot as plt
import gym
from IPython import display as ipythondisplay
import numpy as np
import json
import pickle
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import torch
```

In [3]:

```
env = gym.make('MountainCar-v0')

print('Observation Space: ', env.observation_space)
print('Action Space: ', env.action_space)

no_actions = 3
```

```
Observation Space:  Box(-1.2000000476837158, 0.6000000238418579, (2,), float3
2)
Action Space:  Discrete(3)
```

Initialize the Neural Network

In [4]:

```
input_dim  = 2              # position and velocity
no_layers1 = 256
no_layers2 = 128
no_layers3 = 128
out_dim    = no_actions     # no of actions

model = keras.Sequential()
model.add(keras.Input(shape=(input_dim,)))
model.add(layers.Dense(no_layers1, activation="relu"))
model.add(layers.Dense(no_layers2, activation="relu"))
model.add(layers.Dense(no_layers3, activation="relu"))
model.add(layers.Dense(out_dim))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               768

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 128)               16512

 dense_3 (Dense)             (None, 3)                 387

=================================================================
Total params: 50,563
Trainable params: 50,563
```

```
Non-trainable params: 0
_________________________________________________________________
```

Epsilon greedy action function

In [5]:

```
def epsilon_greedy_action(obs, epsilon, no_actions, model):
    '''chooses epsilon greedy action given a policy'''

    if np.random.rand()<epsilon:
        action = np.random.choice(no_actions, p=[1/no_actions, 1/no_actions,
    else:
        obs    = tf.expand_dims(obs, axis=0)
        pred   = model(obs)
        action = np.argmax(pred)

    return action
```

Semi-gradient SARSA Training

In [ ]:

```
episodes    = 0
tot_step    = 0
epsilon     = 0.1
alpha       = 1e-3
optimizer   = keras.optimizers.SGD(learning_rate=alpha)
loss_fn     = keras.losses.MeanSquaredError()
gamma       = 0.9
converged   = False
conv_step   = 0
conv_thr    = 2000      # if loss not decreasing after n epsidoes, then conve
epi_cost    = []
min_loss    = 1e+6

while not converged:
    episodes += 1
    epi_rew  = 0

    cur_obs     = env.reset()
    cur_action  = epsilon_greedy_action(cur_obs, epsilon, no_actions, model)
    done        = False

    while not done:
        next_obs, reward, done, info = env.step(cur_action)
        epi_rew  += reward
        tot_step += 1
        if done:
            with tf.GradientTape() as tape:
                target       = reward
                prediction   = model(tf.expand_dims(cur_obs, axis=0), trainin
                cur_estimate = prediction[0][cur_action]
                loss_value   = loss_fn(tf.expand_dims(target, axis=0), tf.exp
        else:
            next_action = epsilon_greedy_action(next_obs, epsilon, no_actions
            with tf.GradientTape() as tape:
                next_prediction  = model(tf.expand_dims(next_obs, axis=0), tr
                target           = reward + gamma*next_prediction[0][next_act
                cur_prediction   = model(tf.expand_dims(cur_obs, axis=0), tra
                cur_estimate     = cur_prediction[0][cur_action]
                loss_value       = loss_fn(tf.expand_dims(target, axis=0), tf
            cur_obs          = next_obs
            cur_action       = next_action

        experiment.log_metric("Loss per timestep", loss_value, step=tot_step)
        grads = tape.gradient(loss_value, model.trainable_weights)
        optimizer.apply_gradients(zip(grads, model.trainable_weights))

        if loss_value < min_loss:
            min_loss  = loss_value
            conv_step = 0
        else:
            conv_step += 1

        # if conv_step >= conv_thr:
        #     converged = True
        if episodes>400:
            converged = True

    epi_cost.append(epi_rew)
    experiment.log_metric("episode cost", epi_rew, step=episodes)

experiment.end()
```

In [7]:

```
model.save('semigradient_a4.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have ye
t to be built. `model.compile_metrics` will be empty until you train or evalua
te the model.
```

Testing the trained controller

In [8]:

```
def control_performance(env_name, model, no_actions, trials):
    env          = gym.make(env_name)
    reward_list  = []
    steps_list   = []
    reached_goal = 0

    for i in range(trials):
        done           = False
        episode_reward = 0
        obs            = env.reset()
        steps          = 0

        while not done:
            obs                     = tf.expand_dims(obs, axis=0)
            pred                    = model(obs)
            action                  = np.argmax(pred)
            obs, reward, done, info = env.step(action)
            steps                   += 1
            episode_reward          += reward

        if steps<200:
            steps_list.append(steps)
            reached_goal += 1

        reward_list.append(episode_reward)

    return reached_goal, steps_list
```

In [ ]: