Introduction to Reinforcement Learning

Install TensorFlow and OpenAI Gym for Python3 if you have not done so already:

```
$ sudo apt-get update
$ sudo apt-get install python3-pip
$ pip3 install tensorflow
$ cd gym
$ pip3 install -e .
$ pip3 install -e '.[atari]'
$ pip3 install -e '.[box2d]'
$ pip3 install -e '.[classic_control]'
$ pip3 install pybullet
```

The above assumes that you have already cloned the Gym git repo and installed required packages. Refer to VM Setup document if you have not done this already.

Additionally, install numpy and matplotlib:

```
$ pip3 install numpy
$ pip3 install matplotlib
```

TensorFlow

Implement a linear regression model in TensorFlow, given the following dataset:

```
W, b = 0.5, 1.4
X = np.linspace(0,100, num=100)
y = np.random.normal(loc=W * X + b, scale=2.0, size=len(X))
```

Pseudocode:

```
Import necessary libraries
Create dataset
Create placeholders for input and output
Create variables for model parameters (weight and bias)
Define the linear computation (weight * x + bias)
Define the mean squared error
Define the optimisation operation: opt = tf.train.AdamOptimizer(0.4).minimize(loss)
Create TF session and initialise variables
Loop to train the parameters (210 iterations):
        Run the optimiser and get the loss
        Print loss and model parameters (weight and bias) every 40 iterations
Print the final model parameters (weight and bias)
Plot the original dataset and the linear model
```

Use TensorBoard to plot the MSE loss over time and monitor the weight and bias of the linear regression model during training (create a histogram for each epoch / training iteration).

OpenAI Gym

Manually play the FrozenLake game (Gym environment "FrozenLake-v0") by reading the action to be taken from the user (keyboard input).

The following are the available actions:

- 0: move left
- 1: move down
- 2: move right
- 3: move up

When rendered, the environment will look as follows:

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

S: start point
G: goal
F: frozen (can traverse)
H: hole (cannot traverse)

The goal is to move the agent from S to G without falling in the holes. Movement is uncertain and a reward of +1 is assigned if the end goal is reached.

Print the action and observation spaces and note the type and dimensions.

Print the new state after each action. Do you understand what information is encoded in the state? Note also the uncertain movement, i.e. you may not end up moving in the direction intended.

Repeat for the "Taxi-v3" Gym environment. The state information is a little more complicated for this environment (500 states), so don’t print the state after each action here.

The goal of this game is to pick up a passenger and drop them at a precise location (in the minimum possible time).

The rewards/penalties are as follows:

- +20 for successful drop-off.
- -10 for illegal pickup or drop-off.
- -1 for every timestep.

The action space is as follows:

- 1: move south
- 2: move north
- 3: move east
- 4: move west
- 5: pickup
- 6: drop-off

The : symbol represents an empty location. The | symbol represents a wall that the taxi can't travel through. R,G,Y,B are four potential pickup/dropoff points. The pickup point (passenger location) is identified by the light blue color and the dropoff point by magenta. The taxi is represented by a yellow rectangle.

Print the cumulative reward at each timestep. Make some illegal moves to familiarise yourself with the environment and rewards/penalties.

Dynamic Programming

Implement policy iteration and value iteration DP algorithms. In your implementations, record the total number of iterations required to achieve convergence to the optimal V-function.

Apply your algorithms to the FrozenLake and Taxi Gym environments. Once training is complete, run 100 episodes to evaluate the performance of your agent (as indicated by the total cumulative reward).

Begin with the following parameters:

- FrozenLake: discount factor = 0.99, V-function convergence threshold = 0.0001
- Taxi: discount factor = 0.9, V-function convergence threshold = 0.01

Vary these parameters and note any changes in training time (number of iterations before convergence) and resulting performance.

Run another 5 episodes for each environment. With each action, render the environment (to visualise) and add a 1s delay.

For the FrozenLake environment:

- Print the resulting policy and V-function (reshape to 4x4 to better visualise)
- Record every V-function estimate and plot for each state. Note that the V-function estimates converge monotonically to the optimal values.