Final Exam, Machine Learning, August 2022 Semester, AIT

Happy Thursday! This is the final exam for Machine Learning in the August 2022 semester.

This exam is 2.5 hours long. Once the exam starts, you will have exactly 2.5 hours to finish your work and upload your notebook to Google Classroom.

Please fill in this notebook with your code and short answers. Be sure to put all of your code in the cells marked with

```
# YOUR CODE GOES HERE
```

and please put your answers to the short answer questions exactly where you see the remark

You answer goes here.

Be complete and precise in your answers! Be sure to answer the question that's being asked. Don't dump random information in the hope that it'll give you partial credit. I give generous partial credit, but I will deduct points for answers that are not on point.

Also beware that if I discover any cheating, I will give you a 0 for the entire exam, or worse, and you will likely fail the class. Just don't do it!

OK, that's all for the advice. Relax, take a deep breath, and good luck!

Question 1 (10 points)

Consider the following code for the toy cliff walking problem in reinforcement learning:

In [1]:
```
import numpy as np

# Environment receives a current state and action and returns a new state and
 reward

ROWS = 4
COLS = 12
START = (3, 0)
GOAL = (3, 11)

A = ['U', 'D', 'L', 'R']

def env(s, a):
    row = s[0]
    col = s[1]
    if a == 'U':
        row -= 1
    elif a == 'D':
        row += 1
    elif a == 'L':
        col -= 1
    elif a == 'R':
        col += 1
    else:
        raise Exception("Invalid action '" + a + "'")

    if row < 0:
        row = 0
    if row >= ROWS:
        row = ROWS-1
    if col < 0:
        col = 0
    if col >= COLS:
        col = COLS-1

    s = (row, col)

    if row < ROWS-1 or s == START:
        return s, -1.0, False

    if s == GOAL:
        return GOAL, 0.0, True

    return START, -100.0, True

# Set up Q table

def init_q():
    Q = {}
    for row in range(ROWS):
        for col in range(COLS):
            Q[(row, col)] = np.array([ 0.0 for i in range(len(A))])
    return Q

# Epsilon greedy policy
```

```python
epsilon = 0.1

def eps_greedy(Q, s):
    if np.random.uniform() < epsilon:
        return np.random.choice(len(A))
    return np.argmax(Q[s])

# Example of how to use the functions

Q = init_q()
s = START
a = eps_greedy(Q, s)
ns, r, terminal = env(s, A[a])

print('Action', A[a], 'in state', s, 'gives new state', ns, 'reward', r, 'and
 terminal status', terminal)
```

```
Action U in state (3, 0) gives new state (2, 0) reward -1.0 and terminal stat
us False
```

In the space below, fill in the implementation of the Q-learning algorithm. After running Q-learning to convergence, output the resulting policy as a mapping from states to optimal actions.

In [2]:
```
# Number of episodes to run

def q_learn(episodes, gamma, alpha):
    Q = init_q()
    for episode in range(episodes):
        s = START
        terminal = False
        while not terminal:
            a = eps_greedy(Q, s)
            ns, r, terminal = env(s, A[a])
            # Q-learning update
            # YOUR CODE HERE to implement the Q table update for Q learning
            raise Exception('Not implemented yet')
            s = ns
    return Q

episodes = 10000
gamma = 0.9
learn_rate = 0.1
Q = q_learn(episodes, gamma, learn_rate)

def print_q_max(Q):
    print('-------------------------')
    for row in range(ROWS):
        for col in range(COLS):
            if (row, col) == GOAL:
                print('|G', end='')
            elif row == ROWS-1 and col > 0:
                print('|X', end='')
            else:
                print('|%s' % A[Q[(row, col)].argmax()], end='')
        print('|')
        print('-------------------------')


# Print out the optimal policy learned by Q-learning
# YOUR CODE HERE to print out the optimal policy
raise Exception('Not implemented yet')
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-2-f94ca9ed79fe> in <module>
     18 gamma = 0.9
     19 learn_rate = 0.1
---> 20 Q = q_learn(episodes, gamma, learn_rate)
     21 
     22 def print_q_max(Q):

<ipython-input-2-f94ca9ed79fe> in q_learn(episodes, gamma, alpha)
     11             # Q-learning update
     12             # YOUR CODE HERE to implement the Q table update for Q le
arning
---> 13             raise Exception('Not implemented yet')
     14             s = ns
     15     return Q

Exception: Not implemented yet
```

Question 2 (10 points)

Write a function `sarsa_learn` similar to the `q_learn` function in the previous question that implements the SARSA reinforcement learning algorithm for the cliff walking problem.

After running SARSA to convergence, output the optimal policy according to the resulting Q table.

In [3]:
```
# YOUR CODE HERE
raise Exception('Not implemented yet')
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-3-10a4b505b245> in <module>
      1 # YOUR CODE HERE
----> 2 raise Exception('Not implemented yet')

Exception: Not implemented yet
```

Question 3 (10 points)

Explain why the optimal policies in Question 1 and Question 2 are different, even though the environment is the same.

Your answer here.

Question 4 (10 points)

The Gaussian mixture model for clustering is a probabilistic model with negative log likelihood as the cost function.

The k-means clustering method uses a simpler cost function.

In the space below, explain the differences between the two cost functions.

Your answer here.

Question 5 (10 points)

We saw that the cost function for linear regression can be derived as a negative log likelihood cost function when we assume Gaussian errors for the targets (i.e., $y \sim \mathcal{N}(\theta^T \mathbf{x}, \sigma^2)$).

Is it possible for the k-means cost function to be derived in a similar way based on some probabilistic model for the data distribution? What about the cluster membership?

Your answer here.

Question 6 (30 points)

In PyTorch, implement the neural network given at the Tensorflow Playground with these parameters:

https://playground.tensorflow.org/#activation=relu&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=15&networkShape=4,2&seed=0.40825&showTestData=fa
(https://playground.tensorflow.org/#activation=relu&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=15&networkShape=4,2&seed=0.40825&showTestData=fa

(The network has two ReLU hidden layers with 4 and 2 units, respectively.)

Demonstrate that it can solve the 2D annulus classification problem.

To solve this problem, you should generate a synthetic training set as we have done in lab in the past and show that your PyTorch neural network can learn to distinguish the two classes using a plot. Note that there should be some noise/overlap between the two class distributions.

In [5]:
```
# YOUR CODE HERE
```

Question 7 (10 points)

Explain how we should set up the hyperparameters of a SVM (kernel, c, other parameters) to solve the problem in Question 6. You don't have to write any code! Just explain how you would set up the hyperparamters.

Your explanation here.

Question 8 (10 points)

Write a numpy function to transform the $\mathbf{x}$ data from Question 6 to a new 2D repesentation that would enable a logistic regression model to obtain good accuracy.

Demonstrate that your function transforms the training set from Question 6 effectively (plot the transformed data).

In [ ]:
```
# YOUR CODE HERE
```