# SE 4050 – Deep Learning

$4^{th}$ Year second semester

Lab Assignment 3

Lasal Sandeepa Hettiarachchi

IT19132310

B.Sc. (Hons) in Information Technology Specializing in Software Engineering

Submitted to

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

# TABLE OF CONTENTS

# 1. MODEL-BASED ALGORITHMS (VALUE ITERATION APPROACH)

Model-based algorithms require a model of the environment in that it incoperates the knowledge of the transition process from one state to another.

## 1.1 valueIterationAgents.py

```
    # Write value iteration code here
        "*** YOUR CODE HERE ***"

        for _ in range(iterations):
            current_values = self.values.copy()  #Getting a copy of the
            states = self.mdp.getStates()
            for state in states:  # iterating through each state
                if self.mdp.isTerminal(state):
                    continue
                # get value for best possible action for changing state
                actions = self.mdp.getPossibleActions(state)
                val_arr = []
                for a in actions:
                    val_arr.append(self.getQValue(state, a))
                best_value = max(val_arr)
                current_values[state] = best_value

            self.values = current_values
```

```
def computeQValueFromValues(self, state, action):
    """
     Compute the Q-value of action in state from the
     value function stored in self.values.
    """
    "*** YOUR CODE HERE ***"
    qValue = 0

    # go through every possible outcome of the action
    t_state_probs = self.mdp.getTransitionStatesAndProbs(state, action)
    for nextState, probability in t_state_probs:

        # add reward & future reward (=V) * probability of the outcome
```

```
            reward = self.mdp.getReward(state, action, nextState)
            discount = self.discount
            val = self.values[nextState]
            qValue = qValue + probability * (reward + discount * val)

        return qValue
        util.raiseNotDefined()


    def computeActionFromValues(self, state):
        """
          The policy is the best action in the given state
          according to the values currently stored in self.values.

          You may break ties any way you see fit.  Note that if
          there are no legal actions, which is the case at the
          terminal state, you should return None.
        """
        "*** YOUR CODE HERE ***"
        # retreive the best possible action for the state
        policies = util.Counter()
        pos_actions = self.mdp.getPossibleActions(state)
        for action in pos_actions:

            # how good is an action = q-value (which considers all possible outcomes)
            policies[action] = self.getQValue(state, action)

        # return the best action, e.g. 'north'
        return policies.argMax()
        util.raiseNotDefined()
```

## 1.2 analysis.py

```
def question2():
    answerDiscount = 0.9
    answerNoise = 0
    return answerDiscount, answerNoise

def question3a():
```

```
    answerDiscount = 0.1
    answerNoise = 0
    answerLivingReward = 0.7  # give a negative penalty for moving
    return answerDiscount, answerNoise, answerLivingReward

def question3b():
    answerDiscount = 0.1
    answerNoise = 0.1
    answerLivingReward = 0.7
    return answerDiscount, answerNoise, answerLivingReward

def question3c():
    answerDiscount = 0.5
    answerNoise = 0
    answerLivingReward = 0.5
    return answerDiscount, answerNoise, answerLivingReward

def question3d():
    answerDiscount = 0.9
    answerNoise = 0.1
    answerLivingReward = 0.2
    return answerDiscount, answerNoise, answerLivingReward

def question3e():
    answerDiscount = 0
    answerNoise = 0
    answerLivingReward = 0
    return answerDiscount, answerNoise, answerLivingReward

def question6():
    answerEpsilon = None
    answerLearningRate = None
    # return answerEpsilon, answerLearningRate
    # If not possible, return 'NOT POSSIBLE'
    return 'NOT POSSIBLE'

if __name__ == '__main__':
    print 'Answers to analysis questions:'
    import analysis
    for q in [q for q in dir(analysis) if q.startswith('question')]:
```

```
        response = getattr(analysis, q)()
        print '  Question %s:\t%s' % (q, str(response))
```

## 2. MODEL-FREE ALGORITHMS (Q-LEARNING)

In reinforcement learning, a model-free algorithm is an algorithm which does not use the transition probability distribution associated

### 2.1 qlearningAgents.py

```
# qlearningAgents.py
# ------------------
# Licensing Information:  You are free to use or extend these projects for
# educational purposes provided that (1) you do not distribute or publish
# solutions, (2) you retain this notice, and (3) you provide clear
# attribution to UC Berkeley, including a link to http://ai.berkeley.edu.
#
# Attribution Information: The Pacman AI projects were developed at UC Berkeley.
# The core projects and autograders were primarily created by John DeNero
# (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
# Student side autograding was added by Brad Miller, Nick Hay, and
# Pieter Abbeel (pabbeel@cs.berkeley.edu).

from game import *
from learningAgents import ReinforcementAgent
from featureExtractors import *

import random,util,math

class QLearningAgent(ReinforcementAgent):
    """
      Q-Learning Agent

      Functions you should fill in:
        - computeValueFromQValues
        - computeActionFromQValues
        - getQValue
        - getAction
        - update

      Instance variables you have access to
```

```
        - self.epsilon (exploration prob)
        - self.alpha (learning rate)
        - self.discount (discount rate)

      Functions you should use
        - self.getLegalActions(state)
          which returns legal actions for a state
    """
    def __init__(self, **args):
        "You can initialize Q-values here..."
        ReinforcementAgent.__init__(self, **args)

        "*** YOUR CODE HERE ***"
        self.values = util.Counter()

    def getQValue(self, state, action):
        """
          Returns Q(state,action)
          Should return 0.0 if we have never seen a state
          or the Q node value otherwise
        """
        "*** YOUR CODE HERE ***"
        return self.values[(state, action)]
        util.raiseNotDefined()

    def computeValueFromQValues(self, state):
        """
          Returns max_action Q(state,action)
          where the max is over legal actions.  Note that if
          there are no legal actions, which is the case at the
          terminal state, you should return a value of 0.0.
        """
        "*** YOUR CODE HERE ***"
        maxQ = float('-inf')
        for action in self.getLegalActions(state):
            maxQ = max(maxQ, self.getQValue(state, action))
        return maxQ if maxQ != float('-inf') else 0.0
        util.raiseNotDefined()

    def computeActionFromQValues(self, state):
```

```
        """
          Compute the best action to take in a state.  Note that if there
          are no legal actions, which is the case at the terminal state,
          you should return None.
        """
        "*** YOUR CODE HERE ***"
        if len(self.getLegalActions(state)) == 0:
            return None

        bestQ = self.computeValueFromQValues(state)
        bestActions = []
        for action in self.getLegalActions(state):
            if bestQ == self.getQValue(state, action):
                bestActions.append(action)

        return random.choice(bestActions)
        util.raiseNotDefined()

    def getAction(self, state):
        """
          Compute the action to take in the current state.  With
          probability self.epsilon, we should take a random action and
          take the best policy action otherwise.  Note that if there are
          no legal actions, which is the case at the terminal state, you
          should choose None as the action.

          HINT: You might want to use util.flipCoin(prob)
          HINT: To pick randomly from a list, use random.choice(list)
        """
        # Pick Action
        legalActions = self.getLegalActions(state)
        action = None
        "*** YOUR CODE HERE ***"
        if util.flipCoin(self.epsilon):
            action = random.choice(legalActions)
        else:
            action = self.computeActionFromQValues(state)

        return action
        util.raiseNotDefined()
```

```
    def update(self, state, action, nextState, reward):
        """
          The parent class calls this to observe a
          state = action => nextState and reward transition.
          You should do your Q-Value update here

          NOTE: You should never call this function,
          it will be called on your behalf
        """
        "*** YOUR CODE HERE ***"
        oldValue = self.values[(state, action)]
        newValue = reward + (self.discount * self.computeValueFromQValues(nextState))

        self.values[(state, action)] = (1 - self.alpha) * oldValue + self.alpha * newValue
        #util.raiseNotDefined()

    def getPolicy(self, state):
        return self.computeActionFromQValues(state)

    def getValue(self, state):
        return self.computeValueFromQValues(state)

class PacmanQAgent(QLearningAgent):
    "Exactly the same as QLearningAgent, but with different default parameters"

    def __init__(self, epsilon=0.05,gamma=0.8,alpha=0.2, numTraining=0, **args):
        """
        These default parameters can be changed from the pacman.py command line.
        For example, to change the exploration rate, try:
            python pacman.py -p PacmanQLearningAgent -a epsilon=0.1

        alpha    - learning rate
        epsilon  - exploration rate
        gamma    - discount factor
        numTraining - number of training episodes, i.e. no learning after these many episodes
        """
        args['epsilon'] = epsilon
        args['gamma'] = gamma
        args['alpha'] = alpha
```

```
        args['numTraining'] = numTraining
        self.index = 0  # This is always Pacman
        QLearningAgent.__init__(self, **args)

    def getAction(self, state):
        """
        Simply calls the getAction method of QLearningAgent and then
        informs parent of action for Pacman.  Do not change or remove this
        method.
        """
        action = QLearningAgent.getAction(self,state)
        self.doAction(state,action)
        return action

class ApproximateQAgent(PacmanQAgent):
    """
       ApproximateQLearningAgent

       You should only have to overwrite getQValue
       and update.  All other QLearningAgent functions
       should work as is.
    """
    def __init__(self, extractor='IdentityExtractor', **args):
        self.featExtractor = util.lookup(extractor, globals())()
        PacmanQAgent.__init__(self, **args)
        self.weights = util.Counter()

    def getWeights(self):
        return self.weights

    def getQValue(self, state, action):
        """
          Should return Q(state,action) = w * featureVector
          where * is the dotProduct operator
        """
        "*** YOUR CODE HERE ***"
        features = self.featExtractor.getFeatures(state, action)

        sum = 0
        for feature, value in features.iteritems():
```

```
      sum += self.weights[feature] * value
    return
    util.raiseNotDefined()

  def update(self, state, action, nextState, reward):
    """
      Should update your weights based on transition
    """
    "*** YOUR CODE HERE ***"
    newValue = reward + self.discount * self.computeValueFromQValues(nextState)
    oldValue = self.getQValue(state, action)
    difference = newValue - oldValue

    features = self.featExtractor.getFeatures(state, action)
    for feature, value in features.iteritems():
     self.weights[feature] += self.alpha * difference * features[feature]
    #util.raiseNotDefined()

  def final(self, state):
    "Called at the end of each game."
    # call the super-class final method
    PacmanQAgent.final(self, state)

    # did we finish training?
    if self.episodesSoFar == self.numTraining:
      # you might want to print your weights here for debugging
      "*** YOUR CODE HERE ***"
      pass

```

# 3. AUTOGRADER TOOL RESULTS

## 3.1 Question 1

```
(py2_venv) (base) lasalhettiarachchi@Lasals-MacBook-Pro reinforcement % python2 autograder.py -q q1

Starting on 10-31 at 4:37:35

Question q1
===========

*** PASS: test_cases/q1/1-tinygrid.test
*** PASS: test_cases/q1/2-tinygrid-noisy.test
*** PASS: test_cases/q1/3-bridge.test
*** PASS: test_cases/q1/4-discountgrid.test

### Question q1: 6/6 ###


Finished at 4:37:35

Provisional grades
==================
Question q1: 6/6
------------------
Total: 6/6

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

(py2_venv) (base) lasalhettiarachchi@Lasals-MacBook-Pro reinforcement %
```

## 3.2 Question 2

```
(py2_venv) (base) lasalhettiarachchi@Lasals-MacBook-Pro reinforcement % python2 autograder.py -q q2

Starting on 10-31 at 4:38:19

Question q2
===========

*** PASS: test_cases/q2/1-bridge-grid.test

### Question q2: 1/1 ###


Finished at 4:38:19

Provisional grades
==================
Question q2: 1/1
------------------
Total: 1/1

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

## 3.3 Question 3

```
(py2_venv) (base) lasalhettiarachchi@Lasals-MacBook-Pro reinforcement % python2 autograder.py -q q3

Starting on 10-31 at 4:38:54

Question q3
===========

*** PASS: test_cases/q3/1-question-3.1.test
*** PASS: test_cases/q3/2-question-3.2.test
*** PASS: test_cases/q3/3-question-3.3.test
*** PASS: test_cases/q3/4-question-3.4.test
*** PASS: test_cases/q3/5-question-3.5.test

### Question q3: 5/5 ###


Finished at 4:38:55

Provisional grades
==================
Question q3: 5/5
------------------
Total: 5/5

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

## 3.4 Question 4

```
(py2_venv) (base) lasalhettiarachchi@Lasals-MacBook-Pro reinforcement % python2 autograder.py -q q4

Starting on 10-31 at 4:39:55

Question q4
===========

*** PASS: test_cases/q4/1-tinygrid.test
*** PASS: test_cases/q4/2-tinygrid-noisy.test
*** PASS: test_cases/q4/3-bridge.test
*** PASS: test_cases/q4/4-discountgrid.test

### Question q4: 5/5 ###


Finished at 4:39:55

Provisional grades
==================
Question q4: 5/5
------------------
Total: 5/5

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

## 3.5 Question 5

```
(py2_venv) (base) lasalhettiarachchi@Lasals-MacBook-Pro reinforcement % python2 autograder.py -q q5

Starting on 10-31 at 4:40:23

Question q5
===========

*** PASS: test_cases/q5/1-tinygrid.test
*** PASS: test_cases/q5/2-tinygrid-noisy.test
*** PASS: test_cases/q5/3-bridge.test
*** PASS: test_cases/q5/4-discountgrid.test

### Question q5: 3/3 ###


Finished at 4:40:25

Provisional grades
==================
Question q5: 3/3
------------------
Total: 3/3

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

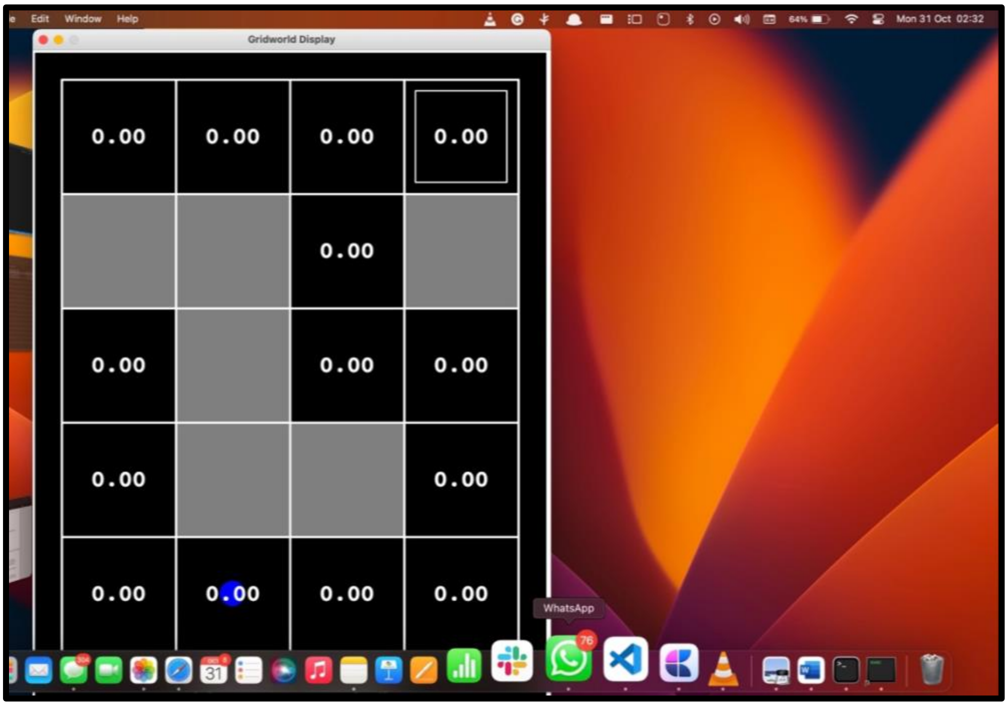
## 3.6 Question 6

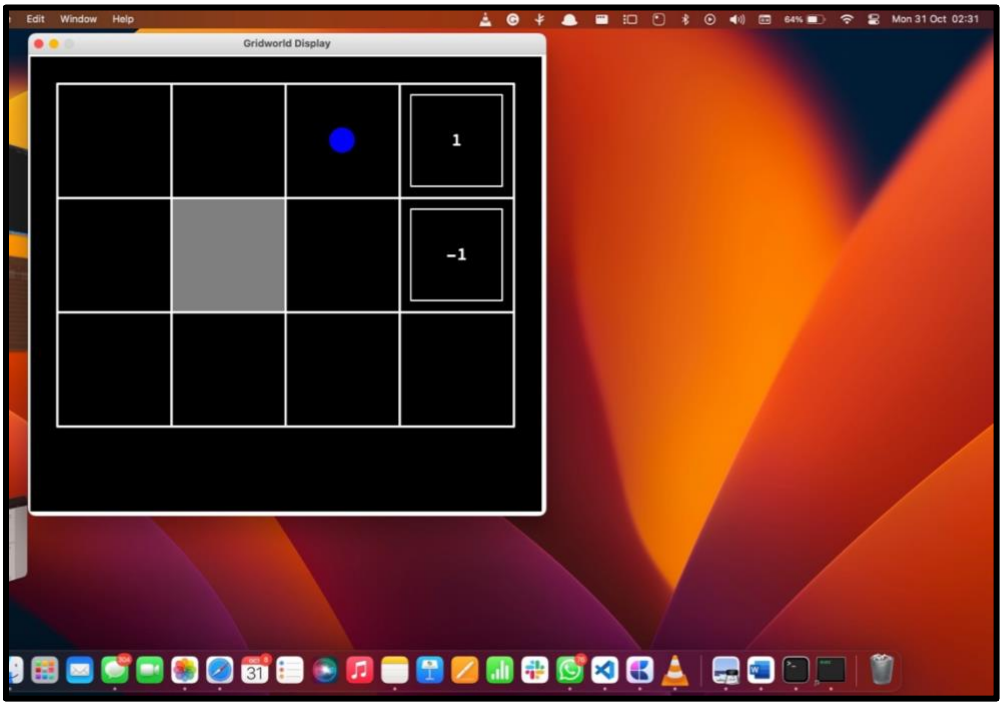
```
● (py2_venv) (base) lasalhettiarachchi@Lasals-MacBook-Pro reinforcement % python2 autograder.py -q q6

  Starting on 10-31 at 4:40:58

  Question q6
  ===========

  *** PASS: test_cases/q6/grade-agent.test

  ### Question q6: 1/1 ###


  Finished at 4:40:58

  Provisional grades
  ==================
  Question q6: 1/1
  ------------------
  Total: 1/1

  Your grades are NOT yet registered.  To register your grades, make sure
  to follow your instructor's guidelines to receive credit on your project.
```

## 3.7 Question 7

```
Average Score: 500.2
Scores:        495.0, 499.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 495.0, 503.0, 503.0, 503.0, 495.0, 495.0, 503.0, 499.0, 499.0,
495.0, 495.0, 499.0, 503.0, 499.0, 495.0, 503.0, 499.0, 495.0, 503.0, 503.0, 503.0, 499.0, 503.0, 495.0, 503.0, 499.0, 503.0, 503.0, 503.0, 5
03.0, 499.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 499.0, 503.0, 499.0, 503.0, 503.0, 495.0, 503.0, 499.0, 503.0, 499.0, 495.0, 503.0, 49
9.0, 495.0, 495.0, 499.0, 495.0, 499.0, 499.0, 499.0, 499.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 499.0, 495.0, 503.0, 503.0, 495.0, 495
.0, 499.0, 503.0, 495.0, 503.0, 503.0, 495.0, 495.0, 503.0, 495.0, 503.0, 495.0, 503.0, 503.0, 499.0, 499.0, 495.0, 503.0, 499.0, 503.0, 503.
0, 503.0
Win Rate:      100/100 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, W
in, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Wi
n, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q7/grade-agent.test (1 of 1 points)
***     Grading agent using command:  python pacman.py -p PacmanQAgent -x 2000 -n 2100 -l smallGrid -q -f --fixRandomSeed
***     100 wins (1 of 1 points)
***         Grading scheme:
***          < 70:  0 points
***         >= 70:  1 points

### Question q7: 1/1 ###


Finished at 4:41:56

Provisional grades
==================
Question q7: 1/1
------------------
Total: 1/1

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

# 4. APPENDIX


Edit Window Help
Mon 31 Oct 02:32
Gridworld Display
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
WhatsApp


Edit Window Help
Mon 31 Oct 02:31
Gridworld Display
1
-1