Міністерство освіти і науки України

Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Програмування інтелектуальних інформаційних систем

**ЗВІТ**

До лабораторної роботи №6.

Тема: Reinforcement Heroes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Виконав**  **студент** |  | ІТ-92, Бондаренко Дмитро Сергійович |  |  |
|  |  | (№ групи, прізвище, ім’я, по батькові ) |  |  |
|  |  |  |  |  |
| **Прийняв** |  | ас. Баришич Л.М. |  |  |
|  |  | (посада, прізвище, ім’я, по батькові ) |  |  |

Київ 2021

# **Завдання лабораторної роботи**

Завданням даної роботи є застосувати бібліотеку DQN до нашої гри. В результаті, гра має грати сама у себе - проте поведінкові зміни відбуватимуться саме на основі візуальної зміни стану на екрані.

# **Опис використаних технологій**

У цій роботі буде використана мова програмування LISP, оскільки ця мова гарно підходять для такого роду задач, а також бібліотека Tensorflow, для проектування нейронних мереж.

# **Опис програмного коду**

Робота базується на грі, яку ми написали у минулих роботах. До роботи були додані 2 нові файли: DqnNet.py, де ми реалізували нейронну мережу а також медод для тренування моделі, та pacmanAgents.py, де ми описали агента, який базується на на нейронній мережі.

Розберемо файл DqnNet.py. У цьому файлі є представлена така іерархія коду:

Класс DqnNet – у ньому відбувається побудова нейронної мережі.

Метод modelTrain - тренування моделі

Розберемо файл DqnNet.py. У цьому файлі є представлена така іерархія коду:

Клас PacmanDQN – агент для DQN.

Метод getNextMove – отримуємо наступну дію для Pacman.

Метод observationStep – система нагород та коректування моделі.

Метод end – метод для закінчення гри.

Метод observationFunction – метод для продовження гри.

Метод log – логування інформації про ітерації.

Метод modelTrain – тренування моделі.

Метод getOneHot – створення списків векторів з 1 значеннями за індексом дії в списку.

Метод matricesMerge – злиття матриць.

Метод matricesStatesGet – отримання станів матриць.

Метод matrixFoodGet – отримання матриці їжі.

Метод matrixCapsulesGet – отримання матриці капсул сили.

Метод matrixPacmanGet – отримання матриці із PacMan.

Метод matrixWallGet – отримання матриці стін.

Метод matrixGhostGet – отримання матриці привидів.

Метод matrixScaredGhostGet – отримання матриці з наляканими привидами.

Метод getAction – отримання наступної дії.

Метод initialStateRegister – ініціалізація первинного реєстру.

tf.compat.v1.disable\_eager\_execution()  
  
  
class DqnNet:  
 # Побудова нейронної мережі  
 def \_\_init\_\_(self, params):  
 self.params = params  
 self.network\_name = 'dqn\_network'  
 self.sess = tf.compat.v1.Session()  
 self.x = tf.compat.v1.placeholder('float',  
 [None, params['width'],  
 params['height'], 6],  
 name=self.network\_name + '\_x')  
 self.q\_t = tf.compat.v1.placeholder('float',  
 [None],  
 name=self.network\_name + '\_q\_t')  
 self.actions = tf.compat.v1.placeholder("float",  
 [None, 4],  
 name=self.network\_name + '\_actions')  
 self.rewards = tf.compat.v1.placeholder("float",  
 [None],  
 name=self.network\_name + '\_rewards')  
 self.terminals = tf.compat.v1.placeholder("float",  
 [None],  
 name=self.network\_name + '\_terminals')  
  
 layer\_name = 'convolutional1'  
 size = 3  
 channels = 6  
 filters = 16  
 stride = 1  
 self.w1 = tf.Variable(tf.compat.v1.random\_normal([size, size, channels, filters], stddev=0.01),  
 name=self.network\_name + '\_' + layer\_name + '\_weights')  
 self.b1 = tf.Variable(tf.constant(0.1, shape=[filters]),  
 name=self.network\_name + '\_' + layer\_name + '\_biases')  
 self.c1 = tf.nn.conv2d(self.x,  
 self.w1,  
 strides=[1, stride, stride, 1],  
 padding='SAME',  
 name=self.network\_name + '\_' + layer\_name + '\_convs')  
 self.o1 = tf.nn.relu(tf.add(self.c1, self.b1),  
 name=self.network\_name + '\_' + layer\_name + '\_activations')  
  
 layer\_name = 'convolutional2'  
 size = 3  
 channels = 16  
 filters = 32  
 stride = 1  
 self.w2 = tf.Variable(tf.compat.v1.random\_normal([size, size, channels, filters], stddev=0.01),  
 name=self.network\_name + '\_' + layer\_name + '\_weights')  
 self.b2 = tf.Variable(tf.constant(0.1, shape=[filters]),  
 name=self.network\_name + '\_' + layer\_name + '\_biases')  
 self.c2 = tf.nn.conv2d(self.o1,  
 self.w2,  
 strides=[1, stride, stride, 1],  
 padding='SAME',  
 name=self.network\_name + '\_' + layer\_name + '\_convs')  
 self.o2 = tf.nn.relu(tf.add(self.c2, self.b2),  
 name=self.network\_name + '\_' + layer\_name + '\_activations')  
 o2\_shape = self.o2.get\_shape().as\_list()  
  
 layer\_name = 'fully\_connected1'  
 hidden = 256  
 dim = o2\_shape[1] \* o2\_shape[2] \* o2\_shape[3]  
 self.o2\_flat = tf.reshape(self.o2,  
 [-1, dim],  
 name=self.network\_name + '\_' + layer\_name + '\_input\_flat')  
 self.w3 = tf.Variable(tf.compat.v1.random\_normal([dim, hidden], stddev=0.01),  
 name=self.network\_name + '\_' + layer\_name + '\_weights')  
 self.b3 = tf.Variable(tf.constant(0.1, shape=[hidden]),  
 name=self.network\_name + '\_' + layer\_name + '\_biases')  
 self.ip3 = tf.add(tf.matmul(self.o2\_flat, self.w3), self.b3,  
 name=self.network\_name + '\_' + layer\_name + '\_ips')  
 self.o3 = tf.nn.relu(self.ip3,  
 name=self.network\_name + '\_' + layer\_name + '\_activations')  
  
 layer\_name = 'fully\_connected2'  
 hidden = 4  
 dim = 256  
 self.w4 = tf.Variable(tf.compat.v1.random\_normal([dim, hidden], stddev=0.01),  
 name=self.network\_name + '\_' + layer\_name + '\_weights')  
 self.b4 = tf.Variable(tf.constant(0.1, shape=[hidden]),  
 name=self.network\_name + '\_' + layer\_name + '\_biases')  
 self.y = tf.add(tf.matmul(self.o3, self.w4),  
 self.b4,  
 name=self.network\_name + '\_' + layer\_name + '\_outputs')  
 self.discount = tf.constant(self.params['discount'])  
 self.yj = tf.add(self.rewards,  
 tf.multiply(1.0 - self.terminals, tf.multiply(self.discount, self.q\_t)))  
 self.Q\_pred = tf.compat.v1.reduce\_sum(tf.multiply(self.y, self.actions),  
 reduction\_indices=1)  
 self.cost = tf.compat.v1.reduce\_sum(tf.pow(tf.subtract(self.yj, self.Q\_pred), 2))  
  
 self.global\_step = tf.Variable(0,  
 name='global\_step',  
 trainable=False)  
  
 self.optimizer = tf.compat.v1.train.AdamOptimizer(self.params['lr']).minimize(self.cost,  
 global\_step=self.global\_step)  
 self.sess.run(tf.compat.v1.global\_variables\_initializer())  
  
 # Тренування моделі  
 def modelTrain(self, bat\_s, bat\_a, bat\_t, bat\_n, bat\_r):  
 feed\_dict = {self.x: bat\_n,  
 self.q\_t: np.zeros(bat\_n.shape[0]),  
 self.actions: bat\_a,  
 self.terminals: bat\_t,  
 self.rewards: bat\_r}  
 q\_t = self.sess.run(self.y, feed\_dict=feed\_dict)  
 q\_t = np.amax(q\_t, axis=1)  
 feed\_dict = {self.x: bat\_s,  
 self.q\_t: q\_t,  
 self.actions: bat\_a,  
 self.terminals: bat\_t,  
 self.rewards: bat\_r}  
 \_, cnt, cost = self.sess.run([self.optimizer,  
 self.global\_step,  
 self.cost],  
 feed\_dict=feed\_dict)  
 return cnt, cost

settings = {  
 'epsilon': 1.0,  
 'epsilon\_final': 0.1,  
 'epsilon\_step': 10000,  
 'train\_iterations': 5000,  
 'batch\_size': 32,  
 'replay\_mem\_size': 100000,  
 'discount': 0.95,  
 'lr': .0002  
}  
  
  
# Отримати шлях у вигляді числового значення  
def getValue(direction):  
 if direction == Directions.NORTH:  
 return 0.  
 elif direction == Directions.EAST:  
 return 1.  
 elif direction == Directions.SOUTH:  
 return 2.  
 else:  
 return 3.  
  
  
# Отримати шлях у вигляді строки  
def getDirection(value):  
 if value == 0.:  
 return Directions.NORTH  
 elif value == 1.:  
 return Directions.EAST  
 elif value == 2.:  
 return Directions.SOUTH  
 else:  
 return Directions.WEST  
  
  
# Агент для DQN  
class PacmanDQN(game.Agent):  
 def \_\_init\_\_(self, args):  
 super().\_\_init\_\_()  
 self.params = settings  
 self.params['width'] = args['width']  
 self.params['height'] = args['height']  
 self.params['num\_training'] = args['numTraining']  
 self.dqn\_net = DqnNet(self.params)  
 self.general\_record\_time = time.strftime("%a %d %b %Y %H %M %S", time.localtime())  
 self.Q\_global = []  
 self.cost\_disp = 0  
 self.cnt = self.dqn\_net.sess.run(self.dqn\_net.global\_step)  
 self.local\_cnt = 0  
 self.iteration\_number = 0  
 self.last\_score = 0  
 self.start\_time = time.time()  
 self.last\_reward = 0.  
 self.replay\_memory = deque()  
 self.last\_scores = deque()  
  
 # Отримуємо наступну дію для Pacman  
 def getNextMove(self):  
 if np.random.rand() > self.params['epsilon']:  
 self.Q\_pred = self.dqn\_net.sess.run(  
 self.dqn\_net.y,  
 feed\_dict={self.dqn\_net.x: np.reshape(self.current\_state,  
 (1, self.params['width'], self.params['height'], 6)),  
 self.dqn\_net.q\_t: np.zeros(1),  
 self.dqn\_net.actions: np.zeros((1, 4)),  
 self.dqn\_net.terminals: np.zeros(1),  
 self.dqn\_net.rewards: np.zeros(1)})[0]  
  
 self.Q\_global.append(max(self.Q\_pred))  
 a\_winner = np.argwhere(self.Q\_pred == np.amax(self.Q\_pred))  
  
 if len(a\_winner) > 1:  
 move = getDirection(  
 a\_winner[np.random.randint(0, len(a\_winner))][0])  
 else:  
 move = getDirection(  
 a\_winner[0][0])  
 else:  
 move = getDirection(np.random.randint(0, 4))  
 self.last\_action = getValue(move)  
 return move  
  
 # Система нагород та коректування моделі  
 def observationStep(self, state):  
 if self.last\_action is not None:  
 self.last\_state = np.copy(self.current\_state)  
 self.current\_state = self.matricesStatesGet(state)  
 self.current\_score = state.getScore()  
 reward = self.current\_score - self.last\_score  
 self.last\_score = self.current\_score  
  
 # Система нагороди  
 if reward > 20:  
 self.last\_reward = 50  
 elif reward > 0:  
 self.last\_reward = 10  
 elif reward < -10:  
 self.last\_reward = -500  
 self.won = False  
 elif reward < 0:  
 self.last\_reward = -1  
  
 if self.terminal and self.won:  
 self.last\_reward = 100.  
 self.ep\_rew += self.last\_reward  
  
 experience = (self.last\_state, float(self.last\_reward), self.last\_action, self.current\_state, self.terminal)  
 self.replay\_memory.append(experience)  
 if len(self.replay\_memory) > self.params['replay\_mem\_size']:  
 self.replay\_memory.popleft()  
  
 self.modelTrain()  
  
 self.local\_cnt += 1  
 self.frame += 1  
 self.params['epsilon'] = max(self.params['epsilon\_final'],  
 1.00 - float(self.cnt) / float(self.params['epsilon\_step']))  
  
 # Метод для закінчення гри  
 def end(self, state):  
 self.ep\_rew += self.last\_reward  
 self.terminal = True  
 self.observationStep(state)  
 self.log()  
  
 # Метод для продовження гри  
 def observationFunction(self, state):  
 self.terminal = False  
 self.observationStep(state)  
 return state  
  
 # Логування інформації про ітерації  
 def log(self):  
 log\_file = open('./logs/' + str(self.general\_record\_time) +  
 '-iteration\_num-' + str(self.params['num\_training']) + '.log', 'a')  
 log\_file.write("# %4d ; steps: %5d ; time\_elapsed: %4f ; points: %12f ; epsilon: %10f " %  
 (self.iteration\_number,  
 self.local\_cnt,  
 time.time() - self.start\_time,  
 self.ep\_rew,  
 self.params['epsilon']))  
 log\_file.write("; Q: %10f ; won: %r \n" % (max(self.Q\_global,  
 default=float('nan')),  
 self.won))  
 sys.stdout.write("# %4d ; steps: %5d ; time\_elapsed: %4f ; points: %12f ; epsilon: %10f " %  
 (self.iteration\_number,  
 self.local\_cnt,  
 time.time() - self.start\_time,  
 self.ep\_rew,  
 self.params['epsilon']))  
 sys.stdout.write("; Q: %10f ; isWon: %r \n" % ((max(self.Q\_global,  
 default=float('nan')),  
 self.won)))  
 sys.stdout.flush()  
  
 # Тренування моделі  
 def modelTrain(self):  
 if self.local\_cnt > self.params['train\_iterations']:  
 batch = random.sample(self.replay\_memory,  
 self.params['batch\_size'])  
  
 batch\_next\_states = []  
 batch\_terminal\_state = []  
 batch\_states = []  
 batch\_rewards = []  
 batch\_actions = []  
  
 for i in batch:  
 batch\_states.append(i[0])  
 batch\_rewards.append(i[1])  
 batch\_actions.append(i[2])  
 batch\_next\_states.append(i[3])  
 batch\_terminal\_state.append(i[4])  
 batch\_states = np.array(batch\_states)  
 batch\_rewards = np.array(batch\_rewards)  
 batch\_next\_states = np.array(batch\_next\_states)  
 batch\_terminal\_state = np.array(batch\_terminal\_state)  
 batch\_actions = self.getOneHot(np.array(batch\_actions))  
  
 self.cnt, self.cost\_disp = self.dqn\_net.modelTrain(  
 batch\_states,  
 batch\_actions,  
 batch\_terminal\_state,  
 batch\_next\_states,  
 batch\_rewards)  
  
 # Створення списків векторів з 1 значеннями за індексом дії в списку  
 def getOneHot(self, actions):  
 actions\_onehot = np.zeros((self.params['batch\_size'], 4))  
 for i in range(len(actions)):  
 actions\_onehot[i][int(actions[i])] = 1  
 return actions\_onehot  
  
 # Злиття матриць  
 def matricesMerge(self, state\_matrices):  
 state\_matrices = np.swapaxes(state\_matrices, 0, 2)  
 total = np.zeros((7, 7))  
 for i in range(len(state\_matrices)):  
 total += (i + 1) \* state\_matrices[i] / 6  
 return total  
  
 # Отримання станів матриць  
 def matricesStatesGet(self, state):  
  
 # Отримання матриці їжі  
 def matrixFoodGet(game\_state):  
 width, height = game\_state.data.layout.width, game\_state.data.layout.height  
 grid = game\_state.data.food  
 matrix = np.zeros((height, width), dtype=np.int8)  
 for i in range(grid.height):  
 for j in range(grid.width):  
 cell = 1 if grid[j][i] else 0  
 matrix[-1 - i][j] = cell  
 return matrix  
  
 # Отримання матриці капсул сили  
 def matrixCapsulesGet(game\_state):  
 width, height = game\_state.data.layout.width, game\_state.data.layout.height  
 capsules = game\_state.data.layout.capsules  
 matrix = np.zeros((height, width), dtype=np.int8)  
 for i in capsules:  
 matrix[-1 - i[1], i[0]] = 1  
 return matrix  
  
 # Отримання матриці із PacMan  
 def matrixPacmanGet(game\_state):  
 width, height = game\_state.data.layout.width, game\_state.data.layout.height  
 matrix = np.zeros((height, width), dtype=np.int8)  
 for agentState in game\_state.data.agentStates:  
 if agentState.isPacman:  
 pos = agentState.configuration.getPosition()  
 cell = 1  
 matrix[-1 - int(pos[1])][int(pos[0])] = cell  
 return matrix  
  
 # Отримання матриці стін  
 def matrixWallGet(game\_state):  
 width, height = game\_state.data.layout.width, game\_state.data.layout.height  
 grid = game\_state.data.layout.walls  
 matrix = np.zeros((height, width), dtype=np.int8)  
 for i in range(grid.height):  
 for j in range(grid.width):  
 cell = 1 if grid[j][i] else 0  
 matrix[-1 - i][j] = cell  
 return matrix  
  
 # Отримання матриці привидів  
 def matrixGhostGet(game\_state):  
 width, height = game\_state.data.layout.width, game\_state.data.layout.height  
 matrix = np.zeros((height, width), dtype=np.int8)  
 for agentState in game\_state.data.agentStates:  
 if not agentState.isPacman:  
 if not agentState.scaredTimer > 0:  
 pos = agentState.configuration.getPosition()  
 cell = 1  
 matrix[-1 - int(pos[1])][int(pos[0])] = cell  
 return matrix  
  
 # Отримання матриці з наляканими привидами  
 def matrixScaredGhostGet(game\_state):  
 width, height = game\_state.data.layout.width, game\_state.data.layout.height  
 matrix = np.zeros((height, width), dtype=np.int8)  
 for agentState in game\_state.data.agentStates:  
 if not agentState.isPacman:  
 if agentState.scaredTimer > 0:  
 pos = agentState.configuration.getPosition()  
 cell = 1  
 matrix[-1 - int(pos[1])][int(pos[0])] = cell  
 return matrix  
  
 width, height = self.params['width'], self.params['height']  
 observation = np.zeros((6, height, width))  
 observation[0] = matrixWallGet(state)  
 observation[1] = matrixPacmanGet(state)  
 observation[2] = matrixGhostGet(state)  
 observation[3] = matrixScaredGhostGet(state)  
 observation[4] = matrixFoodGet(state)  
 observation[5] = matrixCapsulesGet(state)  
 observation = np.swapaxes(observation, 0, 2)  
 return observation  
  
 # Отримання наступної дії  
 def getAction(self, state):  
 move = self.getNextMove()  
 legal = state.getLegalActions(0)  
 if move not in legal:  
 move = Directions.STOP  
 return move  
  
 # Ініціалізація первинного реєстру  
 def initialStateRegister(self, state):  
 self.last\_score = 0  
 self.current\_score = 0  
 self.last\_reward = 0.  
 self.ep\_rew = 0  
 self.last\_state = None  
 self.current\_state = self.matricesStatesGet(state)  
 self.last\_action = None  
 self.terminal = None  
 self.won = True  
 self.Q\_global = []  
 self.delay = 0  
 self.frame = 0  
 self.iteration\_number += 1

# **Скріншоти роботи програмного застосунку**

Робота цього алгоритму базується на тому, що чим більше циклів навчання ми здійснемо, тим краще буде результат. Навчання моделі базується на так званому Q-значенні. Q-значення — це максимальна очікувана винагорода, яку агент може отримати, виконавши задану дію A зі стану S. І чим це значення більше, тим краще у нас навчена модель. Для прикладу, візьмемо перші ітерації, де Q-значення все ще маленьке, але вже існує:

# *387* ; steps: *6070* ; time\_elapsed: *5*.*612979* ; points: *-509.000000* ; epsilon: *0*.*899800* ; Q: *-246.349991* ; won: False   
# *388* ; steps: *6090* ; time\_elapsed: *5*.*691736* ; points: *-497.000000* ; epsilon: *0*.*897900* ; Q: *-245.590500* ; won: False   
# *389* ; steps: *6099* ; time\_elapsed: *5*.*725770* ; points: *-497.000000* ; epsilon: *0*.*897100* ; Q: *-261.480194* ; won: False   
# *390* ; steps: *6109* ; time\_elapsed: *5*.*763013* ; points: *-509.000000* ; epsilon: *0*.*896200* ; Q: *-258.887787* ; won: False   
# *391* ; steps: *6121* ; time\_elapsed: *5*.*809546* ; points: *-500.000000* ; epsilon: *0*.*895100* ; Q: *-254.649307* ; won: False

Як можна побачити, очікувана нагорода низька, а одже, у нас набагато більше шансів програти.

А тепер подивимося на останні значення із логу:

# *5996* ; steps: *331744* ; time\_elapsed: *1589*.*304196* ; points: *-541.000000* ; epsilon: *0*.*100000* ; Q: *156*.*085190* ; won: False   
# *5997* ; steps: *331801* ; time\_elapsed: *1589*.*576263* ; points: *-523.000000* ; epsilon: *0*.*100000* ; Q: *241*.*979477* ; won: False   
# *5998* ; steps: *331839* ; time\_elapsed: *1589*.*757766* ; points: *107*.*000000* ; epsilon: *0*.*100000* ; Q: *197*.*437271* ; won: True   
# *5999* ; steps: *331908* ; time\_elapsed: *1590*.*105445* ; points: *76*.*000000* ; epsilon: *0*.*100000* ; Q: *293*.*975891* ; won: True   
# *6000* ; steps: *332115* ; time\_elapsed: *1591*.*124729* ; points: *-662.000000* ; epsilon: *0*.*100000* ; Q: *171*.*386505* ; won: False

Ми все ще програємо, але шанс перемогти став набагато вище.

Тобто, чим довше ми будемо навчати нашу модель, тим краще буде результат.



Ігрове поле, на якому відбувалося навчання.

**Висновок:** У цій роботі ми навчилися будувати нейронні мережі для RL та DQL, побудували агента, який працює на DQL. А також ми навчили модель, та зрозуміли, як кількість ітерацій для навчання впливают на результат гри моделі.