ASSIGNMENT- SECOND YEAR STUDENTS

--SHIRSHAKK PURKAYASTHA

I.I.S.E.R. Bhopal

DEEP Q-NETWORK

DEEP Q NETWORK CODE (with outputs):

```
import numpy as np
import gym
import random

from keras.layers import Dense
from keras.models import Sequential
from gym.envs.registration import register, spec
from collections import deque
from pandas import DataFrame, Series
```

OUTPUT:

```
Using TensorFlow backend.
```

```
EPISODES = 2048
EPSILON = 0.95
EPSILON_DECAY = 0.95
EPSILON_MIN = 0.2
LEARNING_RATE = 0.01
GAMMA = 0.9
BATCH_SIZE = 32 #can be customized to 64, better fit with 32 obtained

ACTION_LEFT = 0
ACTION_DOWN = 1
ACTION_RIGHT = 2
ACTION_UP = 3
ACTION_DEFAULT = None
ACTION_TEXT = {
```

```
    ACTION_LEFT: 'left',
    ACTION_DOWN: 'down',
    ACTION_RIGHT: 'right',
    ACTION_UP: 'up'
}
"This code has to be rerun in another session as gym library does not allow regist
ering any custom environment twice"
from gym.envs.registration import register
register(
    id='Deterministic-4x4-FrozenLake-v0', #name given to this new environment
    entry_point='gym.envs.toy_text.frozen_lake:FrozenLakeEnv', #env entry point
    kwargs={'map_name': '4x4', 'is_slippery': False} #argument passed to the env
)
env = gym.make('Deterministic-4x4-FrozenLake-v0') # load the environment
my_desk = [
    "GFFFF",
    "FFFFF",
    "FFFFG",
    "FFFFF",
    "FGFFG"
]
class CustomizedFrozenLake(gym.envs.toy_text.frozen_lake.FrozenLakeEnv):
    def __init__(self, **kwargs):
        super(CustomizedFrozenLake, self).__init__(**kwargs)

        for state in range(self.nS): #for all states
            for action in range(self.nA): #for all actions
                my_transitions = []
                for (prob, next_state, _, is_terminal) in self.P[state][action]:
                    row = next_state // self.ncol
                    col = next_state - row * self.ncol
                    tile_type = self.desc[row, col]
                    if tile_type == b'F':
                        reward = -1
```

```
                    elif tile_type == b'G':
                        reward = 10
                    #else:
                        #reward = 0

                    my_transitions.append((prob, next_state, reward, is_terminal))
                self.P[state][action] = my_transitions


from gym.envs.registration import register


register(
    id='Stochastic-5x5-FrozenLake-v0',
    entry_point='gym.envs.toy_text.frozen_lake:FrozenLakeEnv',
    kwargs={'desc': my_desk, 'is_slippery': False})
env = gym.make('Stochastic-5x5-FrozenLake-v0')
env.render()
```

OUTPUT:

```
GSFFF
FFFFF
FFFFG
FFFFF
FGFFG
```

```
class DQNAgent():
    def __init__(self):
        self.env = self._build_env()
        self.nb_status = self.env.observation_space.n
        self.nb_action = self.env.action_space.n
        self.memory = deque(maxlen=2048)
        self.model = self._build_model()

    def _build_env(self): #Customized env setup
        frozen_lake = 'Stochastic-5x5-FrozenLake-v0'
        try:
            spec(frozen_lake)
        except:
            register(id='Stochastic-5x5-FrozenLake-v0',
                     entry_point='gym.envs.toy_text.frozen_lake:FrozenLakeEnv',
                     kwargs={'desc': my_desc, 'is_slippery': False})
        return gym.make(frozen_lake)

    def episode(self):
        status = self.env.reset()

        while True:
            action = self._choose_action(status)
            next_status, reward, done, info = self.env.step(action)
```

```
            self.memory.append((status, action, reward, next_status, done))
            status = next_status

            if done:
                break

    def _choose_action(self, status, choose_best = False, return_probs = False):
        global EPSILON

        if_explore = False
        if choose_best:
            if_explore = False
        else:
            if_explore = np.random.uniform() < EPSILON

        action = ACTION_DEFAULT
        if if_explore:
            # exploration
            action = np.random.choice(self.nb_action)
        else:
            # exploitation
            reward_pred = self.model.predict(self._one_hot_status(status))[0]
            action = np.argmax(reward_pred)

        if EPSILON > EPSILON_MIN:
            EPSILON *= EPSILON_DECAY

        return action if not return_probs else (action, reward_pred)

    def replay(self):
        if len(self.memory) < BATCH_SIZE:
            return

        batches = random.sample(self.memory, BATCH_SIZE)
        X = []
        y = []
        for status, action, reward, next_status, done in batches:
            actual_reward = reward

            if not done:
                next_reward_pred = self.model.predict(
self._one_hot_status(next_status))
                actual_reward += GAMMA * np.max(next_reward_pred[0])

            one_hot_status = self._one_hot_status(status)
            reward_pred = self.model.predict(one_hot_status)
            reward_pred[0][action] = actual_reward

            X.append(one_hot_status[0])
            y.append(reward_pred[0])

        self.model.train_on_batch(DataFrame(X), DataFrame(y))
        # self.model.fit(X, y, epochs=1, verbose=0)

    def demo(self):
        print("\n------------- DEMO ----------------")
        decisions = []
        rewards = []
        for status in range(self.nb_status):
```

```
            best_action, reward = self._choose_action(status, choose_best=True,
return_probs=True)
            decisions.append(best_action)
            rewards.append(reward)

        for i in range(self.nb_status):
            text = ''
            if i==1:
                text = 'START'
            elif i in (0,14,21,24):
                text = 'GOAL'
            else:
                text = ACTION_TEXT[decisions[i]]

            print("{0:^7}".format(text), end='')

            if (i + 1) % 5 == 0:
                print('\n')

        print('LEFT\t\tDOWN\t\tRIGHT\t\tUP')
        for r in rewards:
            print([i for i in r])

    def _one_hot_status(self, status):
        one_hot_status = np.zeros(self.nb_status)
        one_hot_status[status] = 1
        one_hot_status = np.expand_dims(one_hot_status, axis=0)
        return one_hot_status

    def _build_model(self):
        model = Sequential()
        model.add(Dense(16, input_dim=self.nb_status, activation='relu'))
        model.add(Dense(16, activation='relu'))
        model.add(Dense(self.nb_action, activation='linear'))

        model.compile(loss='mse', optimizer='adadelta')
        model.summary()

        return model

def main():
    agent = DQNAgent()

    for i in range(EPISODES):
        agent.episode()
        agent.replay()

        if (i+1) % 512 == 0:
            agent.demo()

if __name__ == '__main__':
    main()
    print('\nDone')
```

OUTPUT:

```
/usr/local/lib/python3.6/dist-packages/gym/envs/toy_text/frozen_lake.py:112:
RuntimeWarning: invalid value encountered in true_divide
  isd /= isd.sum()
```

```
/usr/local/lib/python3.6/dist-packages/gym/envs/toy_text/discrete.py:13:
RuntimeWarning: invalid value encountered in greater
  return (csprob_n > np_random.rand()).argmax()

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                416
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 68
=================================================================
Total params: 756
Trainable params: 756
Non-trainable params: 0
_________________________________________________________________

------------- DEMO ----------------
 GOAL   START  left   left   down

 left   right  down   right  down

 right   up    down   right  GOAL

  up    down   right  left   left

 down   GOAL   left   right  GOAL

LEFT             DOWN             RIGHT            UP
[0.0, -3.7252903e-09, -9.313226e-10, -4.0745363e-10]
[0.026883956, 0.027926492, -0.2081756, 0.05548093]
[0.16299412, -0.042435415, -0.024069825, -0.029029256]
[-0.026142433, -0.23784776, -0.088704094, -0.10066654]
[-0.33659855, 0.025706047, -0.14406803, -0.14200369]
[0.12137655, 0.060025744, 0.06183045, -0.09932299]
[-0.18018717, -0.12429759, -0.010431042, -0.27416897]
[-0.09432528, -0.04692413, -0.095963344, -0.07032873]
[0.08967779, -0.042142354, 0.12336747, -0.1239941]
[-0.07959615, 0.024529729, -0.04175861, -0.0247763]
[0.062016807, 0.02022862, 0.13188526, -0.09485021]
[-0.13127577, -0.00862902, -0.01202275, 0.023150962]
[0.01305115, 0.0760047, -0.04525051, 0.0034894133]
[0.029764276, -0.05809837, 0.10379493, 0.0028660449]
[-0.023524888, -0.02000624, -0.11762434, -0.06505621]
[0.0074093547, -0.04352007, 0.04105658, 0.042052124]
[-0.3079422, -0.009171013, -0.1923568, -0.2293536]
[0.0052046347, 0.05247982, 0.074958146, -0.11156538]
[0.11399228, -0.17130932, -0.06638776, -0.0668566]
[0.1130384, -0.02585829, -0.17499678, 0.037038647]
[-0.016249675, 0.020873848, -0.029830104, -0.042138215]
[-0.03306373, -0.024689697, -0.08251397, -0.03340717]
[0.06681705, -0.07714953, -0.09243506, -0.10808613]
[0.02140005, -0.027815484, 0.03517409, -0.116329335]
[-0.3423802, 0.017804835, -0.14311832, -0.17612892]
```

```
------------- DEMO ----------------
 GOAL   START  left   left   down

 left   right  down   right  down

 right   up    down   right  GOAL

  up    down   right  left   left

 down   GOAL   left   right  GOAL

LEFT             DOWN             RIGHT            UP
[9.313226e-10, 0.0, 1.8626451e-09, -2.910383e-11]
[0.026883963, 0.027926493, -0.20817558, 0.055480935]
[0.16299413, -0.042435415, -0.024069823, -0.029029248]
[-0.026142415, -0.23784776, -0.088704094, -0.10066652]
[-0.33659852, 0.025706047, -0.14406802, -0.14200367]
[0.121376574, 0.060025763, 0.06183046, -0.09932299]
[-0.18018715, -0.12429756, -0.010431021, -0.27416894]
[-0.094325274, -0.04692413, -0.095963344, -0.07032872]
[0.089677796, -0.042142354, 0.12336747, -0.12399409]
[-0.07959615, 0.024529746, -0.041758597, -0.024776299]
[0.062016804, 0.020228634, 0.13188526, -0.094850205]
[-0.13127576, -0.008629009, -0.0120227495, 0.02315097]
[0.0130511485, 0.0760047, -0.045250505, 0.003489408]
[0.029764265, -0.058098376, 0.10379491, 0.0028660516]
[-0.023524888, -0.02000624, -0.11762435, -0.065056205]
[0.007409349, -0.04352005, 0.04105658, 0.04205212]
[-0.3079422, -0.009170983, -0.19235677, -0.2293536]
[0.005204642, 0.05247983, 0.07495814, -0.11156538]
[0.1139923, -0.17130932, -0.066387765, -0.0668566]
[0.113038406, -0.025858276, -0.17499678, 0.037038643]
[-0.016249662, 0.020873846, -0.029830106, -0.042138208]
[-0.033063725, -0.024689697, -0.08251397, -0.033407174]
[0.066817075, -0.07714952, -0.092435054, -0.10808611]
[0.021400072, -0.027815491, 0.035174113, -0.11632933]
[-0.3423802, 0.01780485, -0.14311829, -0.17612892]
```

```
------------- DEMO ----------------
 GOAL   START  left   left   down

 left   right  down   right  down

 right   up    down   right  GOAL

  up    down   right  left   left

 down   GOAL   left   right  GOAL

LEFT              DOWN              RIGHT             UP
[0.0, 0.0, 0.0, -2.910383e-11]
[0.02688396, 0.02792648, -0.20817558, 0.05548094]
[0.16299413, -0.042435423, -0.02406982, -0.029029245]
[-0.026142422, -0.23784776, -0.088704094, -0.10066652]
[-0.33659852, 0.02570604, -0.14406802, -0.14200367]
[0.12137655, 0.06002574, 0.061830454, -0.09932298]
[-0.18018714, -0.12429759, -0.010431031, -0.27416894]
[-0.09432528, -0.046924137, -0.095963344, -0.07032872]
[0.08967779, -0.042142354, 0.12336747, -0.12399409]
[-0.07959613, 0.02452974, -0.041758608, -0.024776287]
[0.062016826, 0.020228626, 0.13188526, -0.09485019]
[-0.13127576, -0.008629013, -0.012022751, 0.023150973]
[0.013051151, 0.0760047, -0.045250513, 0.0034894098]
[0.029764242, -0.058098413, 0.1037949, 0.0028660644]
[-0.023524888, -0.02000624, -0.11762434, -0.065056205]
[0.0074093705, -0.043520063, 0.041056573, 0.042052127]
[-0.3079422, -0.009171013, -0.19235681, -0.22935359]
[0.005204647, 0.052479826, 0.07495814, -0.111565374]
[0.11399228, -0.17130932, -0.066387765, -0.0668566]
[0.1130384, -0.02585829, -0.17499678, 0.037038647]
[-0.016249668, 0.02087384, -0.029830106, -0.042138208]
[-0.033063725, -0.024689697, -0.08251397, -0.03340717]
[0.06681707, -0.07714953, -0.09243506, -0.10808611]
[0.02140008, -0.027815498, 0.035174113, -0.11632931]
[-0.3423802, 0.017804824, -0.14311829, -0.17612892]
```

```
------------- DEMO ----------------
 GOAL   START  left   left   down

 left   right  down   right  down

 right   up    down   right  GOAL

  up    down   right  left   left

 down   GOAL   left   right  GOAL

LEFT              DOWN              RIGHT             UP
[0.0, 0.0, 0.0, -2.910383e-11]
[0.02688396, 0.02792648, -0.20817558, 0.05548094]
[0.16299413, -0.042435423, -0.02406982, -0.029029245]
[-0.026142422, -0.23784776, -0.088704094, -0.10066652]
[-0.33659852, 0.02570604, -0.14406802, -0.14200367]
[0.12137655, 0.06002574, 0.061830454, -0.09932298]
[-0.18018714, -0.12429759, -0.010431031, -0.27416894]
[-0.09432528, -0.046924137, -0.095963344, -0.07032872]
[0.08967779, -0.042142354, 0.12336747, -0.12399409]
[-0.07959613, 0.02452974, -0.041758608, -0.024776287]
[0.062016826, 0.020228626, 0.13188526, -0.09485019]
[-0.13127576, -0.008629013, -0.012022751, 0.023150973]
[0.013051151, 0.0760047, -0.045250513, 0.0034894098]
[0.029764242, -0.058098413, 0.1037949, 0.0028660644]
[-0.023524888, -0.02000624, -0.11762434, -0.065056205]
[0.0074093705, -0.043520063, 0.041056573, 0.042052127]
[-0.3079422, -0.009171013, -0.19235681, -0.22935359]
[0.005204647, 0.052479826, 0.07495814, -0.111565374]
[0.11399228, -0.17130932, -0.066387765, -0.0668566]
[0.1130384, -0.02585829, -0.17499678, 0.037038647]
[-0.016249668, 0.02087384, -0.029830106, -0.042138208]
[-0.033063725, -0.024689697, -0.08251397, -0.03340717]
[0.06681707, -0.07714953, -0.09243506, -0.10808611]
[0.02140008, -0.027815498, 0.035174113, -0.11632931]
[-0.3423802, 0.017804824, -0.14311829, -0.17612892]

Done
```