ASSIGNMENT- SECOND YEAR STUDENTS

--SHIRSHAKK PURKAYASTHA

I.I.S.E.R. Bhopal

NOTE:

"I am facing troubles in converting the dictionary of state movements described in the gym environment library by the (action_size)."

"The problem lies with converting the discrete datatype of action_size into integer."

DEEP Q-NETWORK

DEEP Q NETWORK CODE(with outputs):

```
import gym
import numpy as np
"This code has to be rerun in another session as gym library does not allow registering 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"
]
 import gym

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:

```
GFFFF
FFFFF
FFFFG
FFFFF
FGFFG
/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()
```

```
env.reset()

env.render()



print("Action Space {}".format(env.action_space))

print("State Space {}".format(env.observation_space))
```

OUTPUT:

```
GFFFF
FFFFF
FFFFG
FFFFF
FGFFG
Action Space Discrete(4)
State Space Discrete(25)
/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()
```

```
from keras.models import Sequential

from keras.layers import Dense, Activation, Flatten, Embedding, Reshape

from keras.optimizers import Adam
```

OUTPUT: Using TensorFlow backend.

```
env.reset()

env.step(env.action_space.sample())[0]

action_size = env.action_space

print(action_size)

state_size = env.observation_space

print(state_size)

print(type(action_size))
```

OUTPUT:

```
Discrete(4)
Discrete(25)
<class 'gym.spaces.discrete.Discrete'>/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()
```

"I am facing troubles in converting the dictionary of state movements described in the gym environment library by the (action_size)"

"The problem lies with converting the discrete datatype of action_size into integer"

```
"HIDDEN LAYER 1"
```

```
model=Sequential()
model.add(Embedding(500,4, input_length=1))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
print(model.summary())
model.add(Dense(n, activation='linear'))
```

```
"HIDDEN LAYER 2"
model = Sequential()
model.add(Embedding(500, 4, input_length=1))
model.add(Reshape((4,)))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(action_size, activation='relu'))
print(model.summary())
```

OUTPUT:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 1, 6)              3000
_________________________________________________________________
reshape_2 (Reshape)          (None, 6)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                350
_________________________________________________________________
dense_2 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_3 (Dense)              (None, 50)                2550
=================================================================
Total params: 8,450
Trainable params: 8,450
Non-trainable params: 0
_________________________________________________________________
None
```

```
!pip install tensorflow==2.0.0-beta1
#print(tf.__version__)
import tensorflow as tf
from rl.agents.dqn import DQNAgent
from rl.policy import EpsGreedyQPolicy
from rl.memory import SequentialMemory
```

```
memory = SequentialMemory(limit=50000, window_length=1)

policy = EpsGreedyQPolicy()

dqn = DQNAgent(model=model, nb_actions=action_size, memory=memory, nb_steps_warmup
=500, target_model_update=1e-2, policy=policy)

dqn.compile(Adam(lr=1e-3), metrics=['mae'])

dqn.fit(env, nb_steps=1000000, visualize=False, verbose=1, nb_max_episode_steps=99
, log_interval=100000)

dqn.test(env, nb_episodes=5, visualize=True, nb_max_episode_steps=99)

dqn.save_weights('dqn_{}_weights.h5f'.format("Taxi-v2"), overwrite=True)
```

```
#FLATTENNING THE MODEL

model = Sequential()

model.add(Flatten(input_shape=(1,) + env.observation_space.shape))

model.add(Dense(16))

model.add(Activation('relu'))

model.add(Dense(nb_actions))

model.add(Activation('linear'))

print(model.summary())
```

****