CSE474/574: Introduction to Machine Learning (Fall 2018)
Project 4: Bonus Part

Game1

PacmanDQN

Deep Reinforcement Learning in Pac-man

Tasks Implemented

Step 0: Import the dependencies

```
!pip install gym
import gym
!apt install cmake libopenmpi-dev zlib1g-dev
#installing dependencies otherwise pip installation will thow error
!pip install stable-baselines
from stable_baselines.common.vec_env import DummyVecEnv
from stable_baselines.deepq.policies import MlpPolicy,CnnPolicy
from stable_baselines import DQN
!apt-get install python-opengl
```

Step 1: Create the environment

Here we'll create the MsPacman-v0 environment, taken from OpenAI Gym.

Step 3: Create the hyperparameters

```
y = .95 #gamma
e = 0.1 #random selection epsilion
num_episodes = 2000
RANDOM_THRESHOLD = 1000 #minimum number of frames to choose random actions for
memory_size = 1000
train_batch_size = 64
env_features = 210*160*3
```

Game2

Step 4: Defining Convolutional Neural Network for DQN

Arooshi Avasthy arooshia@buffalo.edu

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.

Step 5: Implement Q-function

- The central idea in Q-Learning is to recognize or learn the optimal action in every state visited by the system (also called the optimal policy) via trial and error.
- The trial and error mechanism can be implemented within the real-world system (commonly seen in robotics) or within a simulator (commonly seen in management science / industrial engineering).
- The agent chooses an action, obtains feedback for that action, and uses the feedback to update its database.
- In its database, the agent keeps a so-called Q-factor for every state-action pair. When the feedback for selecting an action in a state is positive, the associated Q-factor's value is increased, while if the feedback is negative, the value is decreased.
- The feedback consists of the immediate revenue or reward plus the value of the next state.

Game2
Q* Learning with FrozenLake

The goal of this game is to go from the starting state (S) to the goal state (G) by walking only on frozen tiles (F) and avoid holes (H).

Tasks Implemented:

Step 0: Import the dependencies

We use 3 libraries:

- Numpy for our Qtable
- OpenAI Gym for our FrozenLake Environment
- Random to generate random numbers

Step 1: Create the environment

- Here we'll create the FrozenLake environment.
- OpenAI Gym is a library composed of many environments that we can use to train our agents.
- In our case we choose to use Frozen Lake.

```
env = gym.make("FrozenLake-v0")
```

Step 2: Create the Q-table and initialize it

- Now, we'll create our Q-table, to know how much rows (states) and columns (actions) we need, we need to calculate the action_size and the state_size
- OpenAI Gym provides us a way to do that: env.action_space.n and env.observation_space.n

```
action_size = env.action_space.n
state_size = env.observation_space.n
```

```
qtable = np.zeros((state_size, action_size))
print(qtable)
```

Step 3: Create the hyperparameters

- Here, we'll specify the hyperparameters

```
total_episodes = 15000        # Total episodes
learning_rate = 0.8           # Learning rate
max_steps = 99                # Max steps per episode
gamma = 0.95                  # Discounting rate

# Exploration parameters
epsilon = 1.0                 # Exploration rate
max_epsilon = 1.0             # Exploration probability at start
min_epsilon = 0.01            # Minimum exploration probability
decay_rate = 0.005             # Exponential decay rate for exploration prob
```

Step 4: The Q learning algorithm

- Now we implement the Q learning algorithm:
    - Initialize Q-Values (Q(s,a)) for all state action-pairs.
    - Choose an action(a) in the current world state (s) based on current Q-value estimates Q(s,).

- Take the action (a) and observe the outcome state(s') and reward(r).
  Update $Q(s,a) := Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

References:

[1] https://gym.openai.com/

[2]https://github.com/simoninithomas/Deep_reinforcement_learning_Course/tree/master/Q%20learning/FrozenLake

[3] https://github.com/tychovdo/PacmanDQN/