CSE 584: Homework 2

Abstract: The code is to implement deep RL using python, keras, tensorflow and gym. Primary purpose is to learn to solve cartpole environment with help of rewards from the actions. Here we train a RL agent to balance a pole on cart. We build a keras DL network to serve as function approximator for the Q-values. We use DQN agent from keras-rl library and use Boltzmann Q Policy which is an example of policy-based approach. Then train the agent and let it learn from its own action and rewards. Evaluate the trained model and implement saving model so it can we used in future.

I have put the comments in the code in orange color in a text box, below the code explained by it. Example is given below.

-

In [1]:

```
!pip install tensorflow==2.3.0 pip
!install gym
!pip install keras
!pip install keras-rl2
```

Installing dependencies:
Tensorflow - used for building and training neural networks
Gym - toolkit for developing and comparing reinforcement learning algorithms
Keras - neural networks API, written in Python which runs on tensorflow
Keras-rl2 – integrate RL with keras

0. Install Dependencies

In [1]:
```
!pip install tensorflow==2.3.0 pip
!install gym
!pip install keras
!pip install keras-rl2
```

Installing dependencies:
Tensorflow - used for building and training neural networks
Gym - toolkit for developing and comparing reinforcement learning algorithms
Keras - neural networks API, written in Python which runs on tensorflow
Keras-rl2 – integrate RL with keras

1. Test Random Environment with OpenAI Gym

In [2]:
```
import gym
import random
```

In [12]:
```
env = gym.make('CartPole-v0')
states = env.observation_space.shape[0]
actions = env.action_space.n
```

1. Create an instance of Cartpole env which is a classic problem where we have to balance pole on cart by applying force to cart.
2. States gives us number of observations in env
3. Action gives us possible number of actions in env
4. We have 2 actions which is left or right.
5. When i ran the code the number of states were 4.

In [13]:
```
actions
```

Out[13]: 2

In [14]:
```
episodes = 10
for episode in range(1, episodes+1): state =
    env.reset()
    done = False
    score = 0

    while not done: env.render()
        action = random.choice([0,1])
        n_state, reward, done, info = env.step(action)
        score+=reward
    print('Episode:{} Score:{}'.format(episode, score))
```

```
Episode:1 Score:28.0
Episode:2 Score:18.0
Episode:3 Score:11.0
Episode:4 Score:13.0
Episode:5 Score:11.0
Episode:6 Score:16.0
Episode:7 Score:11.0
Episode:8 Score:15.0
Episode:9 Score:14.0
Episode:10 Score:30.0
```

1. First render env.
2. Then take random steps.
3. Apply action to env.
4. We get current state, reward, whether we passed or failed and extra information.
5. We calculate the score where it is added by one if we take correct step and pole doesn't fall
6. If done is not true which means we have not failed or pass yet.
7. We run this 10 times and check the score.
8. The score is maximum of 30

2. Create a Deep Learning Model with Keras

In [7]:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
```

Imported tensorflow keras dependencies which are required to define model. We are using Adam optimizer

In [8]:

```
def build_model(states, actions): model =
        Sequential()
        model.add(Flatten(input_shape=(1,states)))
        model.add(Dense(24, activation='relu'))
        model.add(Dense(24, activation='relu'))
        model.add(Dense(actions, activation='linear'))
        return model
```

We pass 2 arguments which are states and action in this function which are the values we get from env parameters.

1. Create a Sequential object.
2. Add a flatten layer and we create a shape of the states in the env.
3. Then Add 2 Dense node for Deep Learning Model with relu activation function.
4. And at last Dense node has an action.
5. We pass states at top and pass action at the last layer.

In [20]:

```
model = build_model(states, actions)
```

In [21]:

```
model.summary()
```

```
Model: "sequential_1"

Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 4)                 0

dense_3 (Dense)              (None, 24)                120

dense_4 (Dense)              (None, 24)                600

dense_5 (Dense)              (None, 2)                 50
=================================================================
Total params: 770
Trainable params: 770
Non-trainable params: 0
```

3. Build Agent with Keras-RL

In [5]:
```
from rl.agents import DQNAgent
from rl.policy import BoltzmannQPolicy
from rl.memory import SequentialMemory
```

Install keras rl dependencies.

1. We import DQN agent of kerasRL env
2. We use policy-based reinforcement learning and use BoltzmannQpolicy.
3. Sequential Memory used for memory for DQNagent

In [6]:
```
def build_agent(model, actions):
    policy = BoltzmannQPolicy()
    memory = SequentialMemory(limit=50000, window_length=1)
    dqn = DQNAgent(model=model, memory=memory, policy=policy,
    nb_actions=actions, nb_steps_warmup=10, target_model_update=1e-2)
    return dqn
```

1. We pass model which is specified above and different action in env.
2. We setup policy, memory and dqn agent.
3. BoltzmannQpolicy - Probability distribution based on Q-values to choose actions, balancing exploration and exploitation.
4. Sequential Memory - Initializes a memory buffer with a limit of 50,000 entries and a window length of 1(single step update).
5. DQNAgent – We pass the model, memory, policy, env actions. Additional parameters include nb_steps_warmup (The number of steps before training begins, allowing the agent to populate its memory.) and target_model_update(The rate at which the target model is updated with weights from the main model.)

In [25]:
```
dqn = build_agent(model, actions)
dqn.compile(Adam(lr=1e-3), metrics=['mae'])
dqn.fit(env, nb_steps=50000, visualize=False, verbose=1)
```

```
Training for 50000 steps ... Interval 1 (0
steps performed)
10000/10000 [==============================] - 49s
5ms/step - rewar d: 1.0000
51 episodes - episode_reward: 193.118 [58.000, 200.000] - loss: 5.3
76 - mae: 39.195 - mean_q: 78.705

Interval 2 (10000 steps performed)
10000/10000 [==============================] - 49s
5ms/step - rewar d: 1.0000
54 episodes - episode_reward: 185.056 [64.000, 200.000] - loss: 7.5
53 - mae: 40.527 - mean_q: 81.044

Interval 3 (20000 steps performed)
10000/10000 [==============================] - 51s
5ms/step - rewar d: 1.0000
52 episodes - episode_reward: 193.462 [33.000, 200.000] - loss: 8.9
35 - mae: 40.588 - mean_q: 81.322
```

```
Interval 4 (30000 steps performed)
10000/10000 [==============================] - 53s
5ms/step - reward: 1.0000
50 episodes - episode_reward: 200.000 [200.000, 200.000] - loss: 1
1.357 - mae: 41.679 - mean_q: 83.338

Interval 5 (40000 steps performed)
10000/10000 [==============================] - 54s
5ms/step – reward: 1.0000
done, took 256.117 seconds
```

Out[25]: <tensorflow.python.keras.callbacks.History at 0x7ff0e46ad650>

1. Built the agent.
2. Compiled it with Adam Optimizer.
3. Chose metric as 'mae' – Mean Absolute Error.
4. Then start the training and it trains on the environment for 50000 steps.
5. Put visualize as False so we can see the env render visually.
6. It took 256.117 seconds and we reached the reward of 200 in 4th interval.

In [27]:

```python
scores = dqn.test(env, nb_episodes=100, visualize=False)
print(np.mean(scores.history['episode_reward']))
```

```
Testing for 100 episodes ...
Episode 1: reward: 200.000, steps: 200
Episode 2: reward: 200.000, steps: 200
Episode 3: reward: 200.000, steps: 200
Episode 4: reward: 200.000, steps: 200
Episode 5: reward: 200.000, steps: 200
Episode 6: reward: 200.000, steps: 200
Episode 7: reward: 200.000, steps: 200
Episode 8: reward: 200.000, steps: 200
Episode 9: reward: 200.000, steps: 200
Episode 10: reward: 200.000, steps: 200
Episode 11: reward: 200.000, steps: 200
Episode 12: reward: 200.000, steps: 200
Episode 13: reward: 200.000, steps: 200
Episode 14: reward: 200.000, steps: 200
Episode 15: reward: 200.000, steps: 200
Episode 16: reward: 200.000, steps: 200
Episode 17: reward: 200.000, steps: 200
Episode 18: reward: 200.000, steps: 200
Episode 19: reward: 200.000, steps: 200
Episode 20: reward: 200.000, steps: 200
Episode 21: reward: 200.000, steps: 200
Episode 22: reward: 200.000, steps: 200
Episode 23: reward: 200.000, steps: 200
Episode 24: reward: 200.000, steps: 200
Episode 25: reward: 200.000, steps: 200
Episode 26: reward: 200.000, steps: 200
Episode 27: reward: 200.000, steps: 200
Episode 28: reward: 200.000, steps: 200
Episode 29: reward: 200.000, steps: 200
Episode 30: reward: 200.000, steps: 200
Episode 31: reward: 200.000, steps: 200
Episode 32: reward: 200.000, steps: 200
Episode 33: reward: 200.000, steps: 200
```

Episode 34: reward: 200.000, steps: 200
Episode 35: reward: 200.000, steps: 200
Episode 36: reward: 200.000, steps: 200
Episode 37: reward: 200.000, steps: 200
Episode 38: reward: 200.000, steps: 200
Episode 39: reward: 200.000, steps: 200
Episode 40: reward: 200.000, steps: 200
Episode 41: reward: 200.000, steps: 200
Episode 42: reward: 200.000, steps: 200
Episode 43: reward: 200.000, steps: 200
Episode 44: reward: 200.000, steps: 200
Episode 45: reward: 200.000, steps: 200
Episode 46: reward: 200.000, steps: 200
Episode 47: reward: 200.000, steps: 200
Episode 48: reward: 200.000, steps: 200
Episode 49: reward: 200.000, steps: 200
Episode 50: reward: 200.000, steps: 200
Episode 51: reward: 200.000, steps: 200
Episode 52: reward: 200.000, steps: 200
Episode 53: reward: 200.000, steps: 200
Episode 54: reward: 200.000, steps: 200
Episode 55: reward: 200.000, steps: 200
Episode 56: reward: 200.000, steps: 200
Episode 57: reward: 200.000, steps: 200
Episode 58: reward: 200.000, steps: 200
Episode 59: reward: 200.000, steps: 200
Episode 60: reward: 200.000, steps: 200
Episode 61: reward: 200.000, steps: 200
Episode 62: reward: 200.000, steps: 200
Episode 63: reward: 200.000, steps: 200
Episode 64: reward: 200.000, steps: 200
Episode 65: reward: 200.000, steps: 200
Episode 66: reward: 200.000, steps: 200
Episode 67: reward: 200.000, steps: 200
Episode 68: reward: 200.000, steps: 200
Episode 69: reward: 200.000, steps: 200
Episode 70: reward: 200.000, steps: 200
Episode 71: reward: 200.000, steps: 200
Episode 72: reward: 200.000, steps: 200
Episode 73: reward: 200.000, steps: 200
Episode 74: reward: 200.000, steps: 200
Episode 75: reward: 200.000, steps: 200
Episode 76: reward: 200.000, steps: 200
Episode 77: reward: 200.000, steps: 200
Episode 78: reward: 200.000, steps: 200
Episode 79: reward: 200.000, steps: 200
Episode 80: reward: 200.000, steps: 200
Episode 81: reward: 200.000, steps: 200
Episode 82: reward: 200.000, steps: 200
Episode 83: reward: 200.000, steps: 200
Episode 84: reward: 200.000, steps: 200
Episode 85: reward: 200.000, steps: 200
Episode 86: reward: 200.000, steps: 200
Episode 87: reward: 200.000, steps: 200
Episode 88: reward: 200.000, steps: 200
Episode 89: reward: 200.000, steps: 200
Episode 90: reward: 200.000, steps: 200
Episode 91: reward: 200.000, steps: 200
Episode 92: reward: 200.000, steps: 200
Episode 93: reward: 200.000, steps: 200
Episode 94: reward: 200.000, steps: 200
Episode 95: reward: 200.000, steps: 200

```
Episode 96: reward: 200.000, steps: 200
Episode 97: reward: 200.000, steps: 200
Episode 98: reward: 200.000, steps: 200
Episode 99: reward: 200.000, steps: 200
Episode 100: reward: 200.000, steps: 200
200.0
```

In [29]:

```python
_ = dqn.test(env, nb_episodes=15, visualize=True)
```

```
Testing for 15 episodes ...
Episode 1: reward: 200.000, steps: 200
Episode 2: reward: 200.000, steps: 200
Episode 3: reward: 200.000, steps: 200
Episode 4: reward: 200.000, steps: 200
Episode 5: reward: 200.000, steps: 200
Episode 6: reward: 200.000, steps: 200
Episode 7: reward: 200.000, steps: 200
Episode 8: reward: 200.000, steps: 200
Episode 9: reward: 200.000, steps: 200
Episode 10: reward: 200.000, steps: 200
Episode 11: reward: 200.000, steps: 200
Episode 12: reward: 200.000, steps: 200
Episode 13: reward: 200.000, steps: 200
Episode 14: reward: 200.000, steps: 200
Episode 15: reward: 200.000, steps: 200
```

As we can see in test in every episode we get reward as 200

4. Reloading Agent from Memory

In [30]:
```
dqn.save_weights('dqn_weights.h5f', overwrite=True)
```

In [31]:
```
del model
del dqn
del env
```

In [9]:
```
env = gym.make('CartPole-v0')
actions = env.action_space.n
states = env.observation_space.shape[0]
model = build_model(states, actions) dqn =
build_agent(model, actions)
dqn.compile(Adam(lr=1e-3), metrics=['mae'])
```

In [10]:
```
dqn.load_weights('dqn_weights.h5f')
```

In [11]:
```
_ = dqn.test(env, nb_episodes=5, visualize=True)
```

```
Testing for 5 episodes ...
WARNING:tensorflow:From /Users/nicholasrenotte/opt/anaconda3/lib/py
thon3.7/site-packages/tensorflow/python/keras/engine/training_v1.p y:2070:
Model.state_updates (from tensorflow.python.keras.engine.tr aining) is
deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
 Episode  1:  reward:  200.000,  steps:  200
 Episode  2:  reward:  200.000,  steps:  200
 Episode  3:  reward:  200.000,  steps:  200
 Episode  4:  reward:  200.000,  steps:  200
 Episode  5:  reward:  200.000,  steps:  200
```

In [ ]:
```

```

Finally, we just implement to save our dqn model and implement it directly from memory without having to train again and as you can see it gives us reward as 200 just like before.