CS5542 Big Data Apps and Analytics

In Class Programming –10 Report
(Jongkook Son)

Project Overview:

Implementing Deep Q-Learning in Python using Keras & OpenAI Gym
Design a Deep Q learning Network (DQN), using Keras & OpenAI Gym , for cartpole game and visualize your results.

Requirements/Task(s):

1) Designing a DQN for cartpole game in python using Keras & OpenAI Gym (70 points)

2) Visualization of DQN cartpole game (10 points)

3) overall code quality (10 points)

4) Pdf Report quality, video explanation (10 points)

What I learned in ICP:

I could have learned how to use the OpenAI gym and keras to implement Deep q learning. In here, We used a case named CartPole. And I could have learned the basic structure of deep q learning and the difference between deep q and q table. Also learned how to implement deep q learning process with python.

ICP description what was the task you were performing and Screen shots that shows the successful execution of each required step of your code

Problem analys

Slightly place the pole on the black cart.
The goal is to move the black cart from side to side so that the stick does not fall down.
(Just a little wrong move and the stick will fall...)

CartPole's environment consists of four types of states and two actions.

Action
In CartPole, there are two actions. And each action is numbered.

Status (State)
The state S of CartPole is an array of four real values.
If one of the four status values changes by 0.0000001,
The state changes.
in other words, State is infinite! It is impossible to make a Q table..

Reward
+1 is compensated for each step. The termination step is also compensated for +1.

DQN modeling

The idea is simple. It is intended to predict the Q-value for the entire state with the sampled state and Q-value data. And we know that the Neural Network is very good at forecasting!

Construct a four-layer neural network as follows

```
# DQN Modeling
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

num_state = env.observation_space.shape[0]
num_action = env.action_space.n


model = Sequential()
# Hidden layer with 512 nodes
model.add(Dense(512, input_dim= num_state, kernel_initializer='he_uniform', activation='relu'))
# Hidden layer with 256 nodes
model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
# Hidden layer with 64 nodes
model.add(Dense(64, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(num_action, activation='linear'))
model.compile(loss='mse', optimizer="adam")
```

Modeled as Keras (Modeling)
Hidden layer definition
Defines the number of hidden layers. We use a 3-hidden layer here.
Defines the number of hidden layer units. We use 512 units, 256 units, 64 here.
Define an active function. We use the ReLU here.
output layer definition
Defines the number of output layer units. Use two units here because it is a Q-value for each action.
Define an active function. Use Linear.
Define the Loss function. Use mean squared error.
Define Optimizer. We use Adam here.

Learning process

```
import random
from collections import deque
from tqdm import tqdm

num_episode = 300
memory = deque(maxlen=2000)

# Hyper parameter
epsilon = 0.3
gamma = 0.95
batch_size = 32

# DQN Learning
for episode in tqdm(range(num_episode)):
    state = env.reset()
    done = False
    while not done:
        if np.random.uniform() < epsilon:
            action = env.action_space.sample()
        else:
            q_value = model.predict(state.reshape(1, num_state))
            action = np.argmax(q_value[0])
        next_state, reward, done, info = env.step(action)
        # Memory
        memory.append((state, action, reward, next_state, done))

        state = next_state

    # Replay
    if len(memory) > batch_size:
        mini_batch = random.sample(memory, batch_size)
        for state, action, reward, next_state, done in mini_batch:
            if done:
                target = reward
            else:
                target = reward + gamma * (np.max(model.predict(next_state.reshape(1, num_state))[0]))
            q_value = model.predict(state.reshape(1, num_state))
            q_value[0][action] = target
            model.fit(state.reshape(1, num_state), q_value, epochs=1, verbose=0)

env.close()
```

```
100%|██████████| 300/300 [17:26<00:00,  3.49s/it]
```

Agent (Agent)
Collect data while exploring in a greedy manner.

Memory (Memory)
Data obtained from Q-Learning (st,at,rt+1,st+1) are stored in memory deck for use in deep learning.

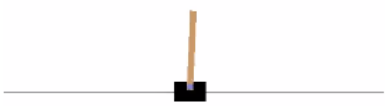
memory=[(s0,a0,r1,s1),(s1,a1,r2,s2),⋯]

Replay
At the end of one episode, data stored in memory is sampled randomly (random) and mini-batch and trained in the DQN model.

Visualize the result

```
from gym.wrappers import Monitor

import base64
from IPython.display import HTML
from IPython import display as ipythondisplay

from pyvirtualdisplay import Display
display = Display(visible=0, size=(1400, 900))
display.start()

def show_video(file_infix):
    with open('./video/openaigym.video.%s.video000000.mp4' % file_infix, 'r+b') as f:
        video = f.read()
        encoded = base64.b64encode(video)
        ipythondisplay.display(HTML(data='''<video alt="Trained CartPole" autoplay
                loop style="height: 200px;">
                <source src="data:video/mp4;base64,{0}" type="video/mp4" />
                 </video>'''.format(encoded.decode('ascii'))))

def wrap_env(env):
    env = Monitor(env, './video', force=True)
    return env
```

```
import gym
import numpy as np

num_state = env.observation_space.shape[0]
env = wrap_env(gym.make('CartPole-v1'))
state = env.reset()
done = False

mem = []

while not done:
    state = np.array(state).reshape(1, num_state)
    q_value = model.predict(state)
    mem.append(q_value[0])
    action = np.argmax(q_value[0])
    state, reward, done, info = env.step(action)

file_infix = env.file_infix
env.close()

show_video(file_infix)
```



Use stored models to predict optimal Q-value and action in any state. Let's save CartPole's movements as a video to see if they work well.
Code for storing learned CartPole tricks on video

.

Conclusion:

This ICP shows that cartpole model can be implemented with gym and keras module. Instead of using Q table in the class, We use Deep Q learning because unlike taxi in here state is infinite, we can not hold every state value into table so we have no choice but to use deep q learning. We used a neural network and predict a Q value here. And as You can see the visualization result, the overall result is quite good.

Challenges that I faced:

The most difficult challenge that I faced was to implement deep q learning process with code. So I get some help from the other people's work

Video link

https://www.youtube.com/watch?v=c0DqDb-yoVk