Artificial Intelligence & Its Applications

Laboratory 02

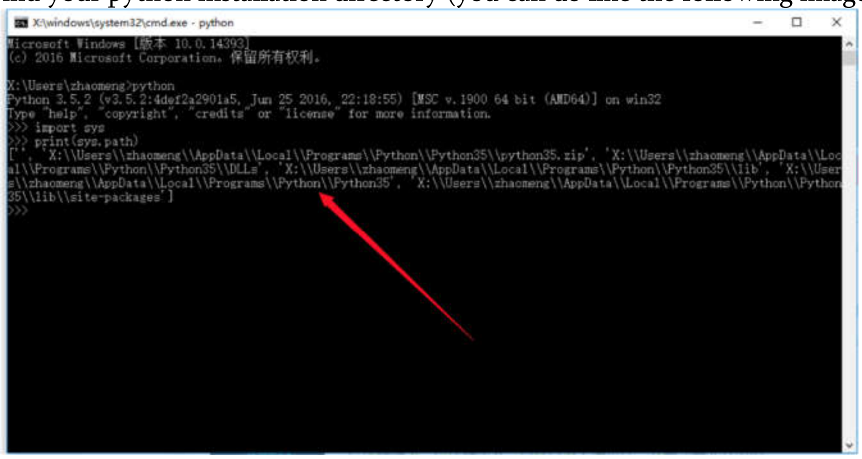
Supplementary Document

Table of Contents

minimax Library

Installation

1. Find your python installation directory (you can do like the following image)



2. Put the minimax folder in the directory Python\Lib\site-packages
   (Python means the directory of your python environment, and it could also be Python36, Python3.8 et al.)

Functions

Build a complete binary tree

Tips: Understand the definition of a complete binary tree if necessary
(https://en.wikipedia.org/wiki/Binary_tree)

```
import minimax
tree = minimax.BiTree()
list_of_leafs = [3, 5, 6, 9]
tree.build_by_list(list_=list_of_leafs, depth=2)
```

Parameters:
- **list_:** The values of leaf nodes (in list format) for the complete binary tree
- **depth:** the depth of the binary tree

Calculate the value of each nodes by minimax

```
tree.fill(max_first=True)
```

Parameters:

- **max_first:** whether the max player plays first

Visualize the graph

```
tree.view_in_graph()
```

Solve game playing problems with Sprague-Grundy (SG) algorithm

Tips: Understand Sprague-Grundy theorem if necessary

https://cp-algorithms.com/game_theory/sprague-grundy-nim.html

```
minimax.SG(sg, son)
```

Parameters:

- **sg:** a list with Boolean values, i.e. 1 or 0. sg[i] = 1 (0) indicates the first-move player wins (loses) when i targets left.
- **son:** all possible states after one move of each state in sg.

Example:

```
sg = [0 for i in range(5)]
     # when we have 4 coins,
     # sg should be initialized with [0, 0, 0, 0, 0]
son = [[],[1-1],[2-1,2-2],[3-1,3-2],[4-1,4-2]]
     # if we can only take 1 or 2 coins one time.
     # son[0] is [] as 0 coins left
minimax.SG(sg, son)
     # sg = [0, 1, 1, 0, 1]
```

gym Library

Installation

command `pip install gym`

Quick Start

Run the following codes and you will see an animation of moving cart and pole.

```
import gym
env = gym.make('CartPole-v0')
for i_episode in range(20):
  observation = env.reset() # restart the environment
  for t in range(100):
    env.render()
    action = env.action_space.sample() # random action
    observation, reward, done, info = env.step(action)
    if done:
      print("Episode finished after {} timesteps".format(t+1))
      break
env.close()
```

The environment's **step** function (in line 8) returns useful information including:

- **observation** (object): an environment-specific object representing your observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game.
- **reward** (float): amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward.
- **done** (boolean): whether it's time to reset the environment again. Most (but not all) tasks are divided up into well-defined episodes, and done being True indicates the episode has terminated. (For example, perhaps the pole tipped too far, or you lost your last life.)
- **info** (dict): diagnostic information useful for debugging. It can sometimes be useful for learning (for example, it might contain the raw probabilities behind the environment's

last state change). However, official evaluations of your agent are not allowed to use this for learning.

https://gym.openai.com/docs/

Tips for Question 3

Refer to `Q3_tips.py`

```
# Code of cartpole-v0 based on q-learning
import numpy as np
import gym
import math
import sys


env = gym.make('CartPole-v0')


def q_index(observation, num_buckets):
    total_index = np.prod(num_buckets)
    x, x_dot, theta, theta_dot = observation
    index = 0

    x_thr = env.env.x_threshold + 1
    x_dot_thr = 100
    theta_thr = math.radians(15)
    theta_dot_thr = math.radians(50)

    x_bins = np.linspace(-x_thr, x_thr, num=num_buckets[0] + 1)
    index += (np.digitize(x, x_bins) - 1) * total_index / num_buckets[0]

    x_dot_bins = np.linspace(-x_dot_thr, x_dot_thr, num=num_buckets[1] + 1)
    index += (np.digitize(x_dot, x_dot_bins) - 1) * total_index / num_buck-
ets[1] / num_buckets[0]

    theta_bins = np.linspace(-theta_thr, theta_thr, num=num_buckets[2] + 1)
    index += (np.digitize(theta, theta_bins) - 1) * total_index / num_buck-
ets[2] / num_buckets[1] / num_buckets[0]

    theta_dot_bins = np.linspace(-theta_dot_thr, theta_dot_thr, num=num_buck-
ets[3] + 1)
    index += (np.digitize(theta_dot, theta_dot_bins) - 1) * total_index /
num_buckets[3] / num_buckets[2] / num_buckets[
        1] / num_buckets[0]
    index = total_index-1 if index >= total_index else index

    return np.int(index)



# gamma
epsilon = 1.0
epsilon_decay = 0.998
epsilon_min = 0.1
alpha = 0.1

num_buckets = np.array([1, 1, 8, 4])  # x, x_dot, theta, theta_dot
q_table = np.zeros((np.prod(num_buckets), env.action_space.n))
env._max_episode_steps = 1000

episodes = 1000
for i in range(episodes + 1):
    observation = env.reset()
    for _ in range(200):
        st = q_index(observation, num_buckets)
```

```
        if np.random.rand() < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(q_table[st])
        epsilon *= epsilon_decay
        observation, reward, done, info = env.step(action)
        if abs(observation[3]) > math.radians(50):
            break
        st_new = q_index(observation, num_buckets)
        q_table[st, action] = (1 - alpha) * q_table[st, action] + alpha * (
                reward + gamma * np.amax(q_table[st_new]))
        if done:
            break
    if i % (episodes / 10) == 0:
        print('episode:{}'.format(i))
        print(q_table)

score = 0
observation = env.reset()
for _ in range(500):
    env.render()
    st = q_index(observation, num_buckets)
    action = np.argmax(q_table[st])
    observation, reward, done, info = env.step(action)
    score += reward
    if done:
        break
env.close()
print('Score=',score)
```