Machine Learning for Computer Vision

Exercise 12

Kodai Matsuoka, Yuyan Li

24.07.2017

1 Results

1.1 Distribution of rewards

Change of distribution of reward and threshold is shown in figure 1 and 2.
As iterations are repeated, the distribution moves towards positive direction. The distribution makes no more improve after 15 iteration.

1.2 How the algorithm performance changes if we change n_samples and percentile

We used 200, 500 and 1000 n_samples. For each case, we changed percentile from 10% to 90% . The result is shown in Figure3. When n_samples = 200, mean reward was never above 0. If number of samples are enough many (500, 1000), we got satisfying result most of the time. However percetile should be chosen between 20% and 80%.

1.3 Tune the algorithm & Bonus

We achieved both positive mean reward and 25% of samples above +9.0 with 1000 n_samples using 50% samples for learning. In this setting, the mean reward was +3.9, and 28% of samples were above +9.0.

2 Code

```
#XVFB will be launched if you run on a server
import os
if type(os.environ.get("DISPLAY")) is not str or len(os.environ
    .get("DISPLAY"))==0:
```

Figure 1: Change of distribution of reward, 300 n_samples, 50 percentile

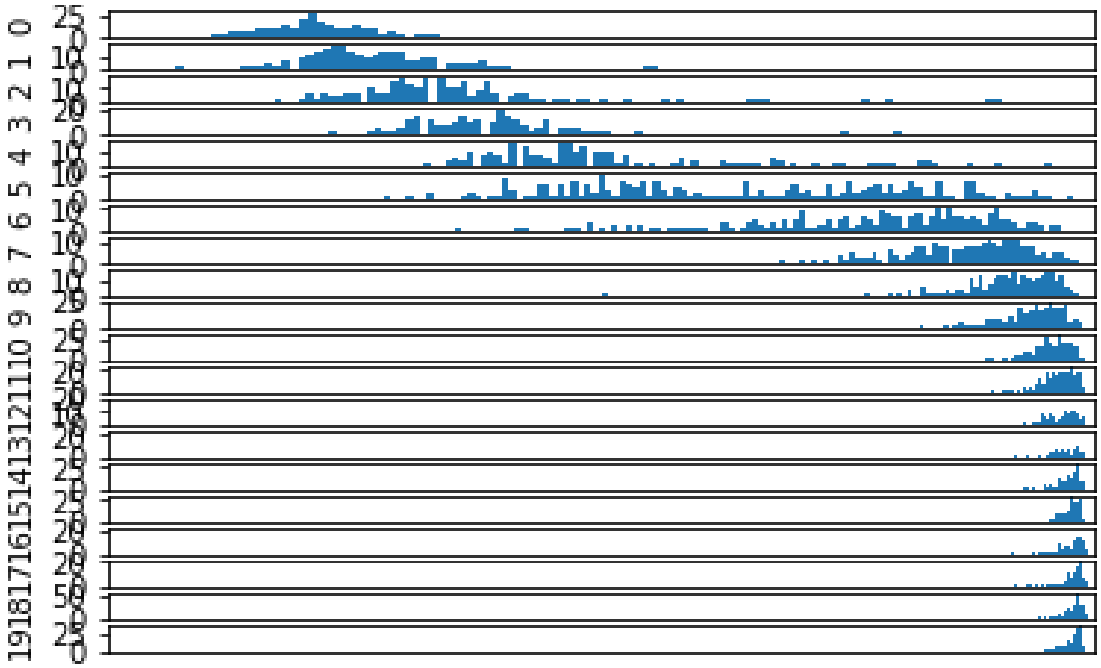


Figure 2: Chanege of threshold, 300 n_samples, 50 percentile

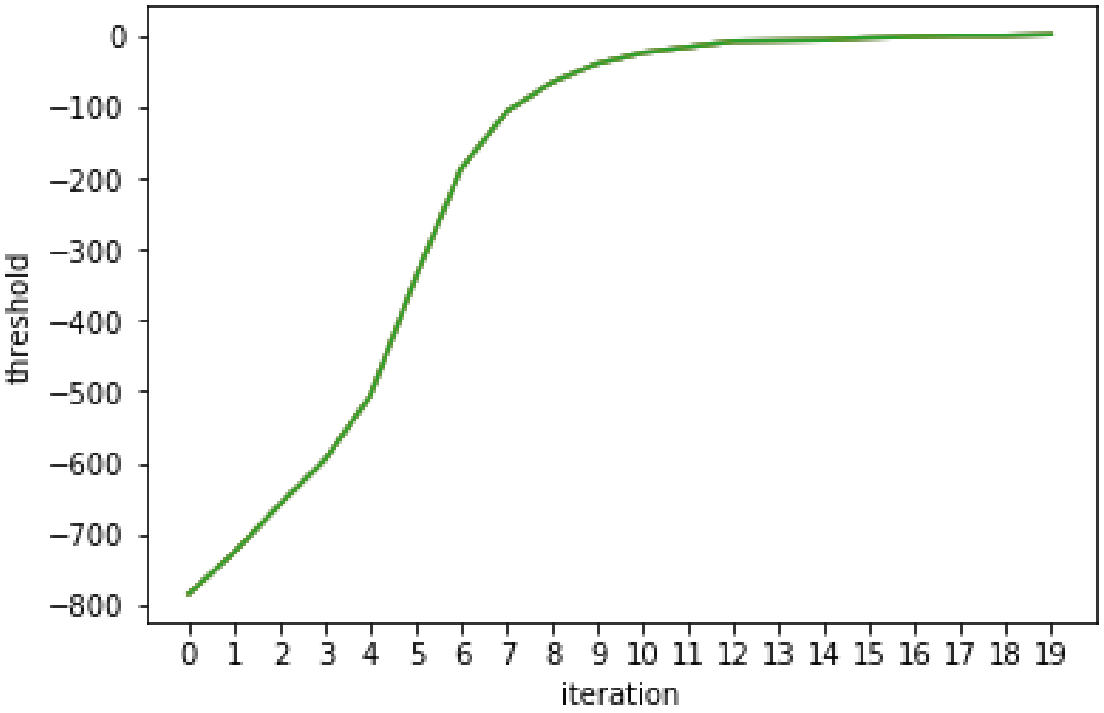
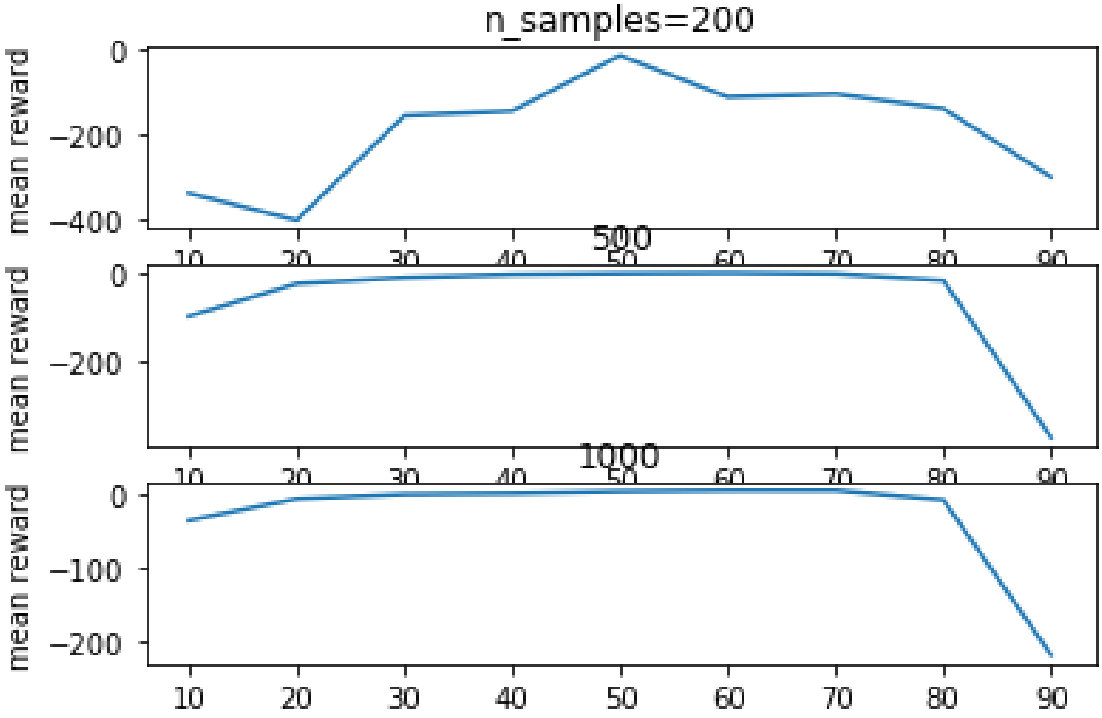


Figure 3: The relation between percentile and mean reward with defferent n_samples




```
    !bash ../xvfb start
    %env DISPLAY=:1

import gym
import numpy as np, pandas as pd
import matplotlib.pyplot as plt

env = gym.make("Taxi-v2")

def generate_session(t_max=10**4):
    """
    Play game until end or for t_max ticks.
    returns: list of states, list of actions and sum of rewards
    """
    states,actions = [],[]
    total_reward = 0.

    s = env.reset()

    for t in range(t_max):
        #pick action from policy (at random with probabilities)
```

```
        a = np.random.choice(6, p = policy[s])

        new_s,r,done,info = env.step(a)

        #record prev state, action and add up reward to states,
            actions and total_reward accordingly
        states.append(s)
        actions.append(a)
        total_reward += r

        s = new_s
        if done:
            break
    return states,actions,total_reward,t


n_samples = 100  #sample this many samples
percentile = 50  #take this percent of session with highest
   rewards
smoothing = 0.1  #add this thing to all counts for stability
policy = np.ones((n_states,n_actions))/n_actions #need to reset
    the policy when changing hyperparameters

for i in range(20):

    sessions = [generate_session() for i in range(n_samples)]

    batch_states,batch_actions,batch_rewards,timestep = map(np.
       array,zip(*sessions))

    #batch_states: a list of lists of states in each session
    #batch_actions: a list of lists of actions in each session
    #batch_rewards: a list of floats - total rewards at each
       session

    threshold = np.percentile(batch_rewards, q=100-percentile)

    elite_states = batch_states[batch_rewards>threshold]
    elite_actions = batch_actions[batch_rewards>threshold]
    elite_states, elite_actions = map(np.concatenate,[
       elite_states,elite_actions])

    #count actions from elite states
    elite_counts = np.zeros_like(policy)+smoothing
```

```
    #count all state-action occurences in elite_states and
        elite_actions
    for state,action in zip(elite_states,elite_actions):
        elite_counts[state,action]+=1.

    #improve the policy
    for state in elite_states:
        policy[state] = elite_counts[state] / sum(elite_counts[
            state])

    print("mean reward = %.5f\tthreshold = %.1f"%(np.mean(
        batch_rewards),threshold))
```