CPS 제어 및 분석 알고리즘 특론

- Reinforcement Learning #3 -

부산대학교

김호원

부산대
정보보호 및 IoT 연구실,
사물인터넷연구센터

2018.10

I. Contextual Bandit

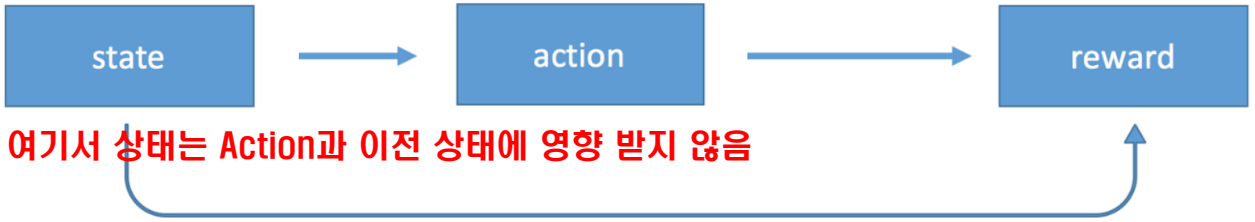
https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-1-5-contextual-bandits-bff01d1aad9c

Bandits Problem

bandit problem

action만이 보상에 영향 줌

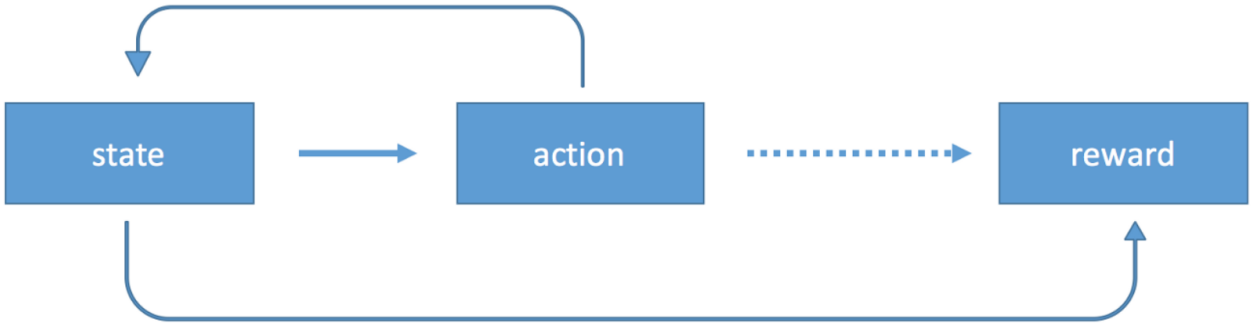
Multi-armed Bandit



Contextual Bandit

여기서 상태는 Action과 이전 상태에 영향 받지 않음

상태와 action이 보상에 영향 줌



Full RL Problem

Action이 상태에 영향을 주며, 상태는 이전 상태에 영향 받음.
또한, 상태는 보상에 영향 줌, 이때 보상은 바로 주어지지 않고 지연됨

Bandits Problem

Multi-armed one bandit problem

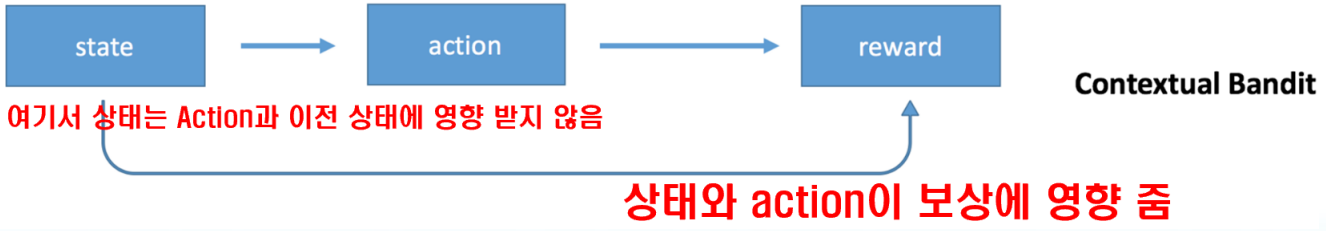
- 하나의 **machine(bandit)** 에 여러 개의 손잡이**(arm)** 있다고 가정
- **Agent**는 하나의 손잡이를 당겨서, **+1** 혹은 **-1**의 보상을 받음
- **Agent**가 **positive** 보상을 받도록 손잡이를 선택하는 방법을 배우는 것이 목적임
- 이전 예에서는, 오직 한 개의 상태만 존재하므로 환경을 무시한 **agent** 설계 가능했음
  - **In previous example, we can design an agent that completely ignores the state of the environment, since for all intents and purposes, there is only ever a single, unchanging state**

action만이 보상에 영향 줌

Contextual Bandit

Contextual Bandits

- 여기선 **state** 개념 도입함
- **State**은 환경을 **description** 함 → 환경 정보를 얻기 때문에, **agent**는 이 정보를 활용하여 **action** 취함
- 하나의 **machine(bandit)**이 아니라 여러 개의 **bandit(3 bandits and each has 4 arms)**으로 확장됨
- **상태를 통해 어떤 bandit을 다루고 있는지 알며, agent**의 목표는 하나가 아닌 여러 개의 **bandit** 에 대해 최선의 **action**을 학습하는 것임
- 각 **bandit**은 각 손잡이에 대해 잠재적 보상이 다르므로, **agent** 는 환경의 상태에 기반을 두고 취할 액션의 조건을 학습할 필요 있음
- **NN**는 환경 상태를 입력받아 액션을 출력함
- 정책(π)은 **NN**의 출력**(one bandit** 에선 **NN**의 **weight** 였음)



Contextual Bandit

Example code: Contextual Bandits

- 4개의 **arm**을 갖는 3개의 **bandit** 사용
- 각 **arm**은 서로 다른 성공 확률 가짐. 이에 최선의 결과를 얻기 위해 다른 **action** 필요
  - The pullBandit function generates a random number from a normal distribution with a mean of 0.
  - The lower the bandit number, the more likely a positive reward will be returned.
  - We want our agent to learn to always choose the bandit-arm that will most often give a positive reward, depending on the Bandit presented.

Contextual Bandit - code

```
class contextual_bandit():
    def __init__(self):
        self.state = 0
        #List out our bandits. Currently arms 4, 2, and 1 (respectively) are the most optimal.
        self.bandits = np.array([[0.2,0,-0.0,-5],[0.1,-5,1,0.25],[-5,5,5,5]])
        self.num_bandits = self.bandits.shape[0]  # shape은 차원return → 3개 bandits
        self.num_actions = self.bandits.shape[1]  # 4개의 arms on each bandit

    def getBandit(self):
        self.state = np.random.randint(0,len(self.bandits))
                # 3개의 bandit 중에서 하나 선택?
        return self.state

    def pullArm(self,action):
        bandit = self.bandits[self.state, action]
        result = np.random.randn(1)  #Get a random number. Normal distribution 갖는 난수1개 생성
        if result > bandit:
            #return a positive reward.
            return 1
        else:
            #return a negative reward.
            return -1
```

Contextual Bandit - code

```
class agent():
def __init__(self, lr, s_size,a_size):
        #These lines established the feed-forward part of the network. The agent takes a state and produces an action.
        self.state_in= tf.placeholder(shape=[1],dtype=tf.int32)
        state_in_OH = slim.one_hot_encoding(self.state_in,s_size)
        output = slim.fully_connected(state_in_OH,a_size,\
        biases_initializer=None,activation_fn=tf.nn.sigmoid,weights_initializ
er=tf.ones_initializer())
        self.output = tf.reshape(output,[-1])
        self.chosen_action = tf.argmax(self.output,0)

        #The next six lines establish the training proceedure. We feed the reward and chosen action into the network
        #to compute the loss, and use it to update the network.
        self.reward_holder = tf.placeholder(shape=[1],dtype=tf.float32)
        self.action_holder = tf.placeholder(shape=[1],dtype=tf.int32)
       self.responsible_weight = tf.slice(self.output,self.action_holder,[1])
        self.loss = -(tf.log(self.responsible_weight)*self.reward_holder)
        optimizer = tf.train.GradientDescentOptimizer(learning_rate=lr)
        self.update = optimizer.minimize(self.loss)
```

Contextual Bandit - code

Training the Agent

- We will train our agent by getting a state from the environment, take an action, and recieve a reward.
- Using these three things, we can know how to properly update our network in order to more often choose actions given states that will yield the highest rewards over time.

```
tf.reset_default_graph()
cBandit = contextual_bandit() #Load the bandits.
myAgent =
agent(lr=0.001,s_size=cBandit.num_bandits,a_size=cBandit.num_actions)
#Load the agent.

weights = tf.trainable_variables()[0]     #The weights we will evaluate to look into the network.
total_episodes = 10,000    #Set total number of episodes to train agent on.
total_reward = np.zeros([cBandit.num_bandits,cBandit.num_actions])     #Set
scoreboard for bandits to 0.
e = 0.1 #Set the chance of taking a random action.
init = tf.initialize_all_variables()
```

Contextual Bandit - code

Training the Agent

```
# Launch the tensorflow graph
with tf.Session() as sess:
sess.run(init)
i = 0
while i < total_episodes:
    s = cBandit.getBandit() #Get a state from the environment.
    #Choose either a random action or one from our network.
    if np.random.rand(1) < e:
        action = np.random.randint(cBandit.num_actions)
    else:
        action = sess.run(myAgent.chosen_action,  \
                 feed_dict={myAgent.state_in:[s]})

    reward = cBandit.pullArm(action) #Get our reward for taking an action given a bandit.
    #Update the network.
    feed_dict={myAgent.reward_holder:[reward],
               myAgent.action_holder:[action],myAgent.state_in:[s]}
    _,ww = sess.run([myAgent.update,weights], feed_dict=feed_dict)
#Update our running tally of scores.
    total_reward[s,action] += reward
```

Contextual Bandit - code

Training the Agent

```
if i % 500 == 0:

print "Mean reward for each of the " + str(cBandit.num_bandits) + " bandits: "
+ str(np.mean(total_reward,axis=1))
i+=1

for a in range(cBandit.num_bandits):
print "The agent thinks action " + str(np.argmax(ww[a])+1) + " for bandit " +
str(a+1) + " is the most promising....“

if np.argmax(ww[a]) == np.argmin(cBandit.bandits[a]):
print "...and it was right!"
else:
print "...and it was wrong!"
```

감사합니다

Q & A