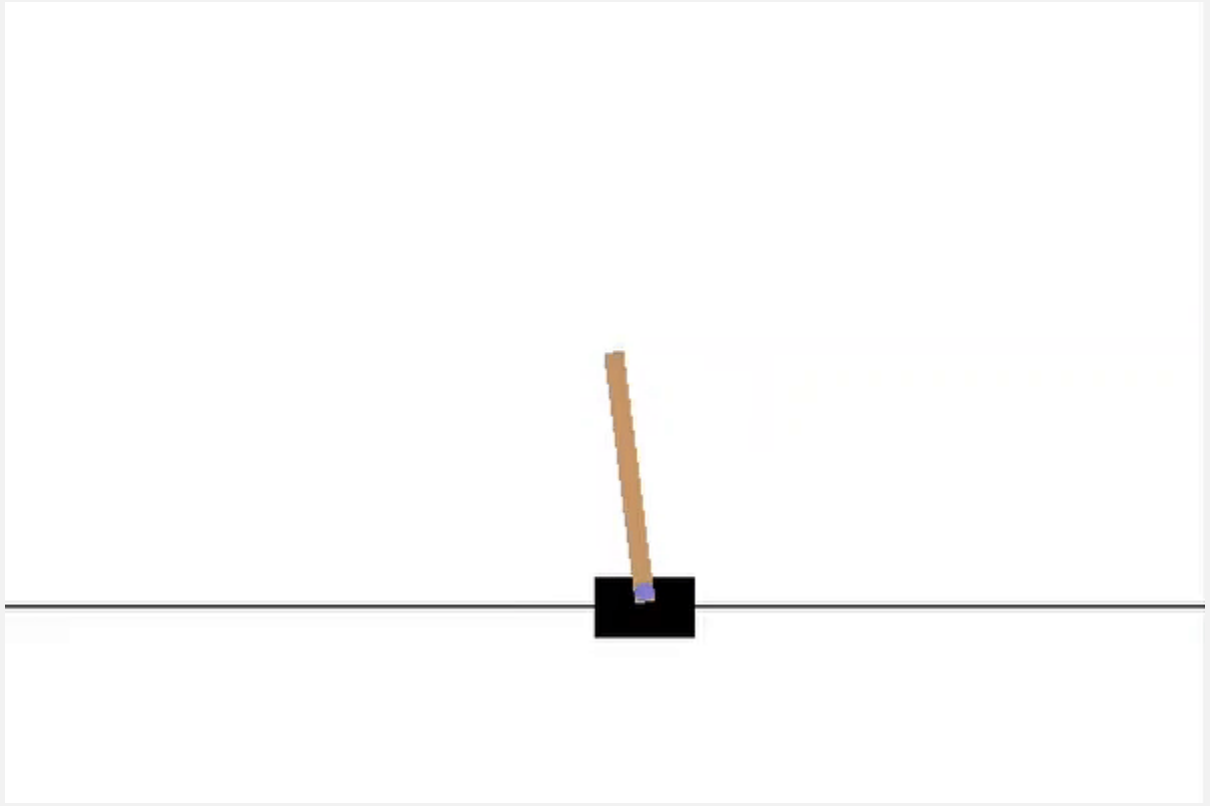
Cart-Pole DQN

이은후

소개



마찰이 없는 트랙에 cart가 있고, 카트에는 막대기(pole)이 연결되어 있음

목표: 막대기가 넘어지지 않도록 하는 것.
(카트에 0(좌), +1(우)의 힘을 가해 조종할 수 있음)

조건

- state (매 스텝마다 관측할 수 있는 환경정보)

  -cart의 위치

  -cart의 속력

  -pole의 각도(수직선으로부터)

  -pole의 각속도

- action

  : cart에 0, +1의 힘을 주어 좌,우로 조종

조건

- 종료조건

  -pole의 각도가 -12~+12를 넘어갈 경우

  -cart가 중심으로부터 2.4이상 벗어날 경우 (-2.4~+2.4)

- reward

  :매 스텝마다 막대기가 잘 서있으면 +1의 보상

환경(env)의 반환값

: 관찰, 보상, done, info (4개)

- 관찰(observation): 환경에 대한 관찰을 나타내는 객체.
- 보상(reward): 이전의 행동을 통해 얻어지는 보상의 양. 크기는 환경에 따라 달라지나,
  궁극적 목표는 언제나 보상의 총량을 높이는 것.
- done: (Boolean) 환경을 reset 해야 할 지 나타내는 진리값.
  (done=True이면 에피소드 종료를 나타냄)
- info: (dict) 디버깅에 활용하는 진단 정보

환경(env)의 반환값

[observation, action]

- 각 에피소드를 시작할 때 첫번째 관찰한 값
  state = env.reset()
  return값은 [카트 위치, 카트 속도, 막대기 각도, 막대기 회전율]

```
First observation: [0.02762716 0.01787037 0.01697009 0.02417966]
```

- action = env.action_space.sample()
  action은 항상 0 또는 1값을 반환함

```
Action: 1
```

환경(env)의 반환값

[step]

- step = env.step(action)
  : action을 선택했을 때 (observation, reward, done, info)반환

```
First observation: [0.02762716 0.01787037 0.01697009 0.02417966]
Action: 1
Step: (array([ 0.02798457,  0.2127449 ,  0.01745368, -0.26310107]), 1.0, False, {})
```

observation: [카트 위치, 카트 속도, 막대기 각도, 막대기 회전율]
reward: 1
done = False (에피소드가 종료되지 않았다는 의미)

코드

- Neural Network 모델 생성

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(24, input_dim=4, activation=tf.nn.relu),
    tf.keras.layers.Dense(24, activation=tf.nn.relu),
    tf.keras.layers.Dense(2, activation='linear')
])
```

4개의 입력
2개의 은닉층 (24개의 노드)
2개의 출력

- 총 1000회 에피소드

```
for i in range(1000):

    state = env.reset()
    state = np.reshape(state, [1, 4])
    eps = 1 / (i / 50 + 10)
```

eps = epsilon
에피소드 진행될수록 exploration보다 exploitation 비율이 높아짐

코드

- 한 에피소드는 200 time-step으로 이루어짐

```
for t in range(200):

    # Inference: e-greedy
    if np.random.rand() < eps:
        action = np.random.randint(0, 2)
    else:
        predict = model.predict(state)
        action = np.argmax(predict)

    next_state, reward, done, _ = env.step(action)
    next_state = np.reshape(next_state, [1, 4])

    memory.append((state, action, reward, next_state, done))
    state = next_state

    if done or t == 199:
        print('Episode', i, 'Score', t + 1)
        score.append(t + 1)
        break
```

E-greedy에 따라 선택

선택한 action에 따라 다음 상태, 보상, done값이 변화함.
next_state는 다시 네트워크에 입력하기 위한 형태로 변환

200 time-step동안 게임이 진행되거나 그 전에 종료될 경우,
점수 저장 후 해당 에피소드 종료

코드

- 훈련 진행

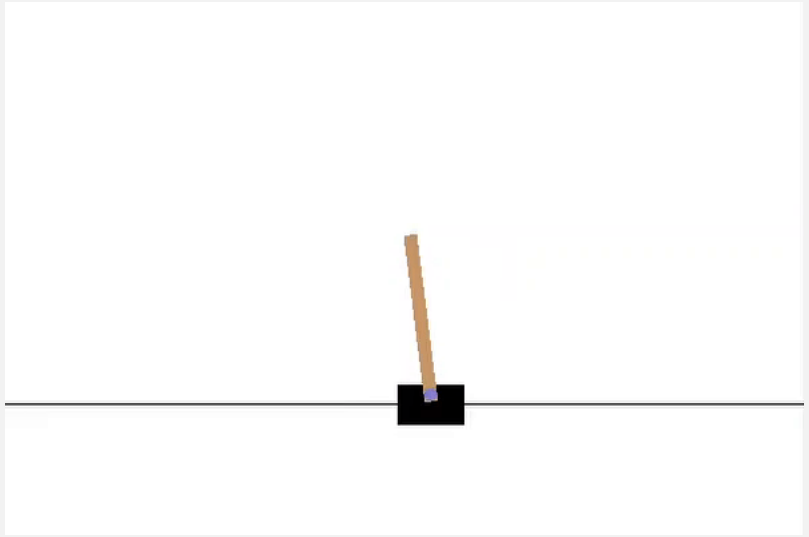
```
# Training
if i > 10:    에피소드를 10회 이상 진행하는 경우:
    minibatch = random.sample(memory, 16)

    for state, action, reward, next_state, done in minibatch:
        target = reward
        if not done:
            target = reward + dis * np.amax(model.predict(next_state)[0])
        target_outputs = model.predict(state)
        target_outputs[0][action] = target
        model.fit(state, target_outputs, epochs=1, verbose=0)
```

model.fit() 통해 훈련 진행

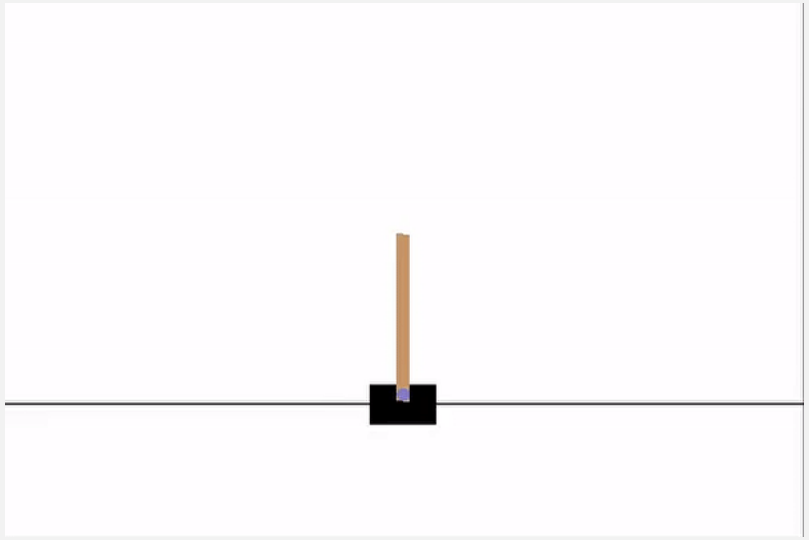
Q value (=target) 설정,
현재 state에서 네트워크가 출력해야 할 값들 설정

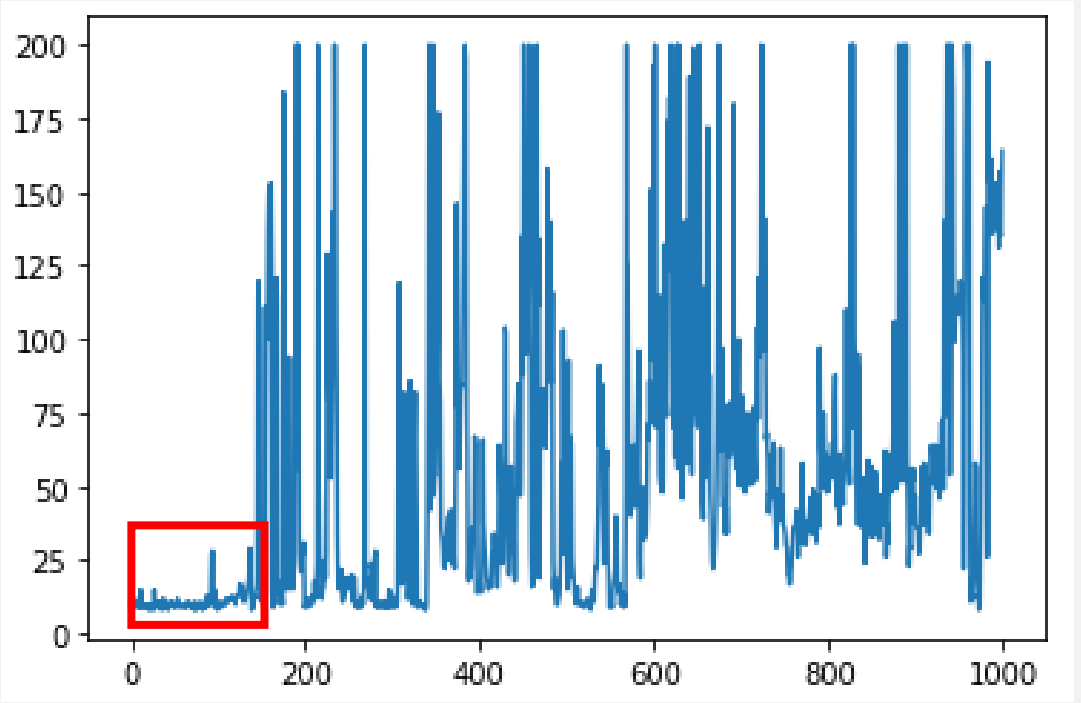
결과

학습 초반



학습 후반





[에피소드에 따른 score]

(score: 200번의 time-step 중 몇 번까지 돌아갔는지.)

학습할수록 실력이 점점 향상됨