저는 신경망 알고리즘을 이용한, 딥러닝 분류 학습을 이용했습니다.

에피소드를 무작위로 수행후, 이중 점수가 나은 에피소드를 골라 저장해서 단순한 시퀀셜 모델을 만들어 학습했습니다.

이외에

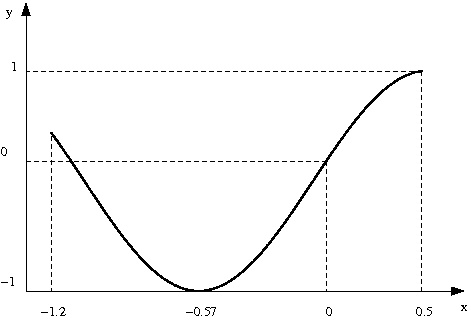
https://gist.github.com/gkhayes/3d154e0505e31d6367be22ed3da2e955

<https://skettee.github.io/post/q_network/>

<http://jinicoding.net/openai-gym%EC%9C%BC%EB%A1%9C-q-table-%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-%EB%A7%8C%EB%93%A4%EA%B8%B0-2/>

등도 살펴보았으나, Q-모델의 경우 학습이 잘 안되서 가장 기초적인 방법을 이용했습니다.

mountaincar 문제의 그레프는 다음과 같습니다.



이때 리워드를 주는 x값을 여러 번 바꿔봤으나, 0.2가 가장 학습결과가 좋은 데이터셋이 나왔습니다.

**기본적인 gym의 mountain car 동작**

*#https://davinci-ai.tistory.com/33 참조하였습니다.*

**import** gym

**import** tensorflow **as** tf

**import** random

**import** numpy **as** np

env **=** gym**.**make('MountainCar-v0') *# env = gym에서 제공하는 환경으로, 언덕이며 왼쪽과 오른쪽을 반복하여 가속도를 만들어야 오를 수 있음.*

print(env**.**observation\_space) *# 관찰공간으로, agent가 환경을 볼 수 있는 범위이다.*

print(env**.**observation\_space**.**low) *# x축 좌표의 최솟값, 최소 속도*

print(env**.**observation\_space)

print(env**.**observation\_space**.**high) *# x축 좌표의 최댓값, 최대 속도*

print(env**.**\_max\_episode\_steps) # 최대 스탭의 수

print(env**.**action\_space) *# agent가 할 수 있는 행동의 경우의 수, discrete는 이산적, 정수로 나눌수 있음을 뜻한다.*

*# agent : car, action : left, stop, right이고, reward는 200이하의 step에서 time step마다 -1, 깃발(x축 0.5지점)에 도착하는 것(종료 조건, 최대 보상 조건)이 목표임*

env **=** gym**.**make('MountainCar-v0') *# 환경 생성*

env**.**reset() *# 환경 초기화, 안해주면 render 되지 않음*

action **=** env**.**action\_space**.**sample() *# 랜덤한 액션 수행, 0,1,2중 하나 랜덤으로 리턴.*

print(action) *# 액션*

print(env**.**step(action)) *# 수행 결과. 환경, 보상, ep 종료여부*

env**.**render() *# 실행결과 화면으로 출력*

env**.**close() *# 안해주면 render에서 멈춤*

*## 여기까지가 기본적인 명령어*

**ep의 수행**

In [177]:

env **=** gym**.**make('MountainCar-v0') *# 환경 생성*

env**.**reset() *# 환경 초기화, 안해주면 render 되지 않음*

state **=** 0 *# state 변수*

score **=** 0

suc **=** **False**

**while** **True**:

action **=** env**.**action\_space**.**sample()

obs, reward, done, info **=** env**.**step(action)

print(obs)

score **=** score **+** reward

state **=** state **+** 1

**if** done:

**break**

env**.**render()

env**.**close() *# 안해주면 render에서 멈춤*

**if** score **>** **-**200:

suc **=** **True**

print(score)

print(state)

기본적으로, env.step가 깃발에 도달 또는 200회를 다 했는지를 반환한다는 점을 이용, 반복문으로 반복합니다.

**반복으로 랜덤한 act를 수행**

env **=** gym**.**make('MountainCar-v0')

env**.**reset() *# 환경 초기화*

scores **=** []

train\_data **=** []

apt\_scores **=** []

req\_score **=** **-**198 # 학습할만한 점수만 기록하기위해 적당한 점수를 준다.

**for** i **in** range(50000):

env**.**reset()

score **=** 0

*# state = 0*

result **=** []

previous\_obs **=** []

**while** **True**:

action **=** env**.**action\_space**.**sample()

obs, reward, done, info **=** env**.**step(action)

**if** len(pre\_obs) **>** 0:

result**.**append([pre\_obs, action])

pre\_obs **=** obs

**if** obs[0] **>** **-**0.2: *# x 좌표에 따라 리워드를 준다.*

reward **=** 1

score **=** score **+** reward

*# state = state + 1*

**if** done:

**break**

*# env.render() # 실행결과 화면으로 출력*

scores**.**append(score)

**if** score **>** req\_score:

apt\_scores**.**append(score)

**for** data **in** result:

train\_data**.**append(data)

env**.**close() *# 안해주면 render에서 멈춤*

print('all of scores mean', np**.**mean(scores))

print('req scores', len(apt\_scores))

print('mean of req scores', np**.**mean(apt\_scores))

print(suc)

적당한 트레인 데이터를 뽑기 위해, obs에 따른 리워드 정책을 바꿔가면서 시도해본 결과, 0.2일 때 가장 적절한 학습 데이터가 나옴을 확인할 수 있었습니다.

**Train**

In [179]:

train\_X **=** np**.**array([i[0] **for** i **in** train\_data])**.**reshape(**-**1, 2) *# 상태를 X*

train\_Y **=** np**.**array([i[1] **for** i **in** train\_data])**.**reshape(**-**1, 1) *# 그에 따른 행동을 Y*

print(train\_X**.**shape)

print(train\_Y**.**shape)

model **=** tf**.**keras**.**Sequential([

tf**.**keras**.**layers**.**Dense(128, input\_shape**=**(2,), activation**=**'relu'),

tf**.**keras**.**layers**.**Dense(32, activation**=**'relu'),

tf**.**keras**.**layers**.**Dense(3, activation**=**'softmax')

])

기본적인 레이어를 이용한 신경망을 만들었습니다.

In [181]:

model**.**compile(

optimizer**=**tf**.**optimizers**.**Adam(),

loss**=**'sparse\_categorical\_crossentropy',

metrics**=**['accuracy']

)

callback **=** tf**.**keras**.**callbacks**.**EarlyStopping(monitor**=**'val\_loss', patience**=**5)

history **=** model**.**fit(train\_X, train\_Y, epochs**=**30, callbacks**=**[callback], batch\_size**=**32, validation\_split**=**0.25)

PLOT

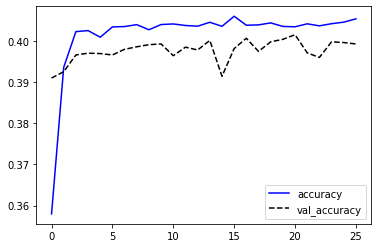
**import** matplotlib.pyplot **as** plt

plt**.**plot(history**.**history['accuracy'], 'b-', label**=**'accuracy')

plt**.**plot(history**.**history['val\_accuracy'], 'k--', label**=**'val\_accuracy')

plt**.**legend()

plt**.**show()



env**.**close()

env**.**reset()

score **=** 0

state **=** 0

previous\_obs **=** []

**while** **True**:

**if** len(previous\_obs) **==** 0:

action **=** env**.**action\_space**.**sample()

**else**:

logit **=** model**.**predict(np**.**expand\_dims(previous\_obs, axis**=**0))[0]

action **=** np**.**argmax(logit)

obs, reward, done, \_ **=** env**.**step(action)

previous\_obs **=** obs

score **=** reward **+** score

state **=** state **+** 1

env**.**render()

**if** done:

**break**

print('score', score)

print('state', state)

env**.**close()

솔직히 이번 학기는 많이 부족함을 느끼고 있습니다.

학교를 안가다보니 물어볼 사람도 없고.. 실력이 미천하다보니 받아주는 연구실도 없는지라.. 어떻게 공부를 해야하는지가 너무 막막합니다.

책을 추천해주시면 감사하겠습니다..