**Homework 4**

2018324133 김태우

Python과 numpy, tensorflow 라이브러리를 사용하였다.

|  |
| --- |
| import numpy as np  import tensorflow as tf  import random  # Build Network  # Input is (position, velocity)  x = tf.placeholder(tf.float32, [None, 2])  # Reward of current state  R = tf.placeholder(tf.float32, [None, 1])  # Action Mask  A = tf.placeholder(tf.float32, [None, 3])  # Qvalue of next state  nQ = tf.placeholder(tf.float32, [None, 1])  # Hidden Layer with 16  w1 = tf.Variable(tf.random\_normal([2, 16], stddev=0.1))  f1 = tf.matmul(x, w1)  # Final Output  w2 = tf.Variable(tf.random\_normal([16, 3], stddev=0.1))  Q = tf.matmul(f1, w2) |

우선 상수들과 문제의 해결을 위한 Tensorflow 네트워크를 정의하였다.

Input은 현재 position과 velocity를 받고, 16개의 unit을 가진 hidden layer를 통과하여 최종 Q value를 산출한다.

가능한 Action은 좌우로 accel 및 현 속도 유지의 3개를 사용하였다.

A는 action mask로서 사용된 action만 1, 나머지는 0의 값을 갖는 one-hot vector이다. 이 vector를 output Q vector에 곱함으로서 사용된 action의 Q만을 학습할 수 있다.

|  |
| --- |
| desQ = tf.reduce\_sum(R + 0.95 \* nQ)  loss = tf.square(tf.reduce\_sum(tf.multiply(Q, A)) - desQ)  train = tf.train.GradientDescentOptimizer(0.001).minimize(loss) |

그리고 학습을 위한 loss function을 정의한다.

이상적인 Q를 Reward + 0.95 \* (Next Q)로 정의한 후 이상적인 Q와의 차이를 loss로 하여 back propagation으로 weight을 학습한다.

|  |
| --- |
| sess = tf.Session()  init = tf.global\_variables\_initializer()  sess.run(init) |

Tensorflow session을 만들고 initialize한다.

|  |
| --- |
| player = np.zeros((1, 2))  nextplayer = np.zeros((1, 2))  # Run Episode  for iteration in range(1, 5001) :  #Reset Player  player[0][0] = random.random() \* 0.2 - 0.6  player[0][1] = 0.  ... |

이제 초기값을 정의한 후 5000회의 experiance를 for loop로 수행한다.

각 experiance에서 player의 초기값은 position : [-0.6, -0.4], velocity : 0.0 이다.

|  |
| --- |
| ...  step = 0  avgloss = 0.  while step < 2000 and player[0][0] < 0.5 :  # Use epsilon-greedy  r = random.random()  if r < 0.1 :  action = random.randint(0, 2)  else :  res = sess.run(Q, {x : player})  action = np.argmax(res[0])  ... |

이제 매 step을 수행한다.

우선 각 experiance의 최대 step 수는 2000회로 제한하고, position이 0.5에 도달하면 종료하도록 한다.

action은 epsilon-greedy 방식으로 10% 확률로 랜덤 값을, 90% 확률로 argmax Q를 선택하였다.

|  |
| --- |
| ...  # Get Next State  nextplayer[0][0] = player[0][0] + player[0][1]  if nextplayer[0][0] < -1.2 :  nextplayer[0][0] = -1.2  nextplayer[0][1] = player[0][1] + 0.001 \* (action-1) - 0.0025 \  \* np.cos(3 \* player[0][0])  if nextplayer[0][1] < -0.07 :  nextplayer[0][1] = -0.07  elif nextplayer[0][1] > 0.07 :  nextplayer[0][1] = 0.07    # Get Next Q Value  nextQ = sess.run(Q, {x : nextplayer})    # Get Action Mask  ActionMask = np.zeros((1, 3))  ActionMask[0][action] = 1.    # Get Reward  if nextplayer[0][0] >= 0.5 :  reward = 0.  else :  reward = -1.  # Update Q Value  \_, res = sess.run([train, loss], {x : player, R : np.array([[reward]]), \  A : ActionMask, nQ : np.array([[np.max(nextQ[0])]])})  avgloss += res    # Prepare Next Step  player[0][0] = nextplayer[0][0]  player[0][1] = nextplayer[0][1]  step += 1 |

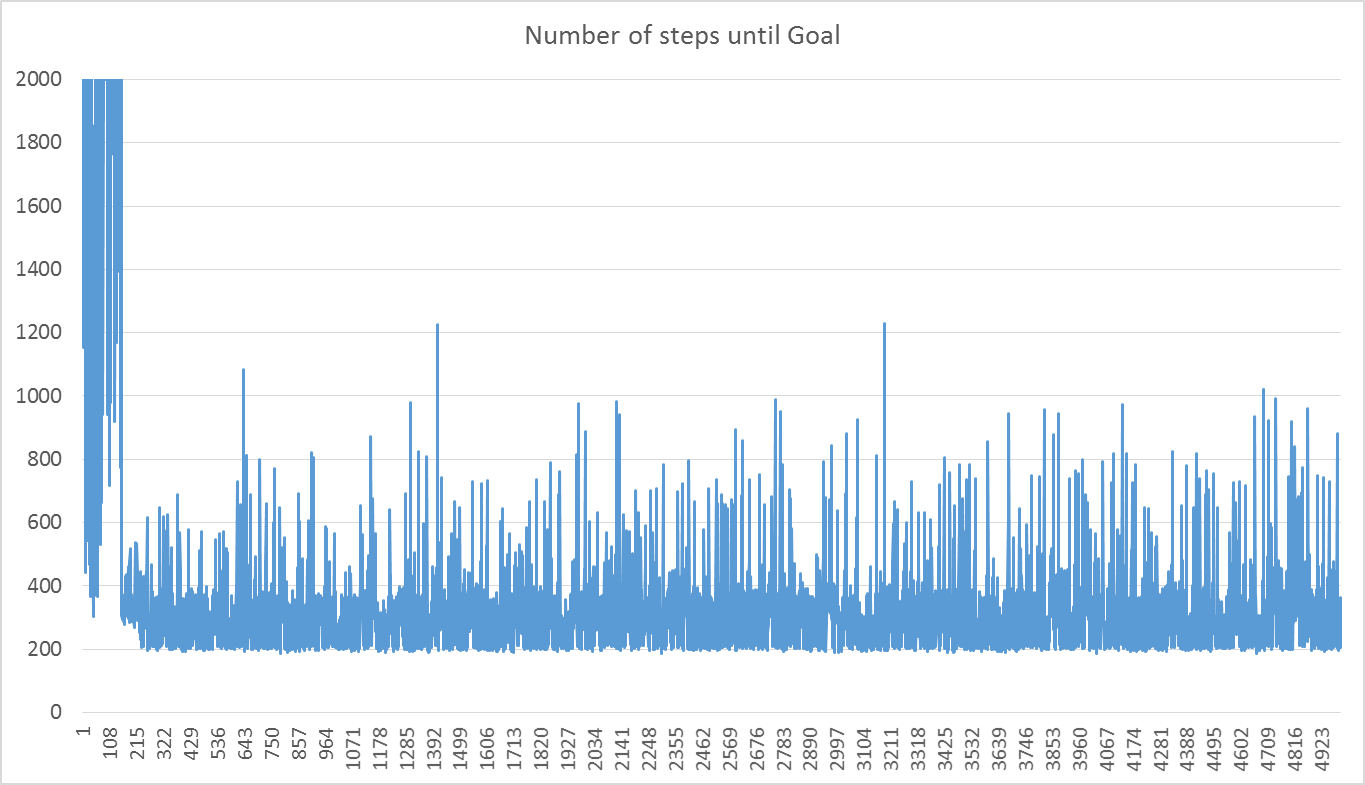
다음으로 next state는 action을 바탕으로 교재에 소개된 공식을 써서 산출한다.

position은 [-1.2, )로, velocity는 [-0.07, 0.07]로 교재와 같이 Clipping하였다.

nextQ는 next state를 넣어서 나온 Q의 maximum 값을 사용하고, reward는 골인 지점에 도달하였으면 0, 아니면 -1을 사용한 뒤, train을 호출하여 back propagation을 수행한다.

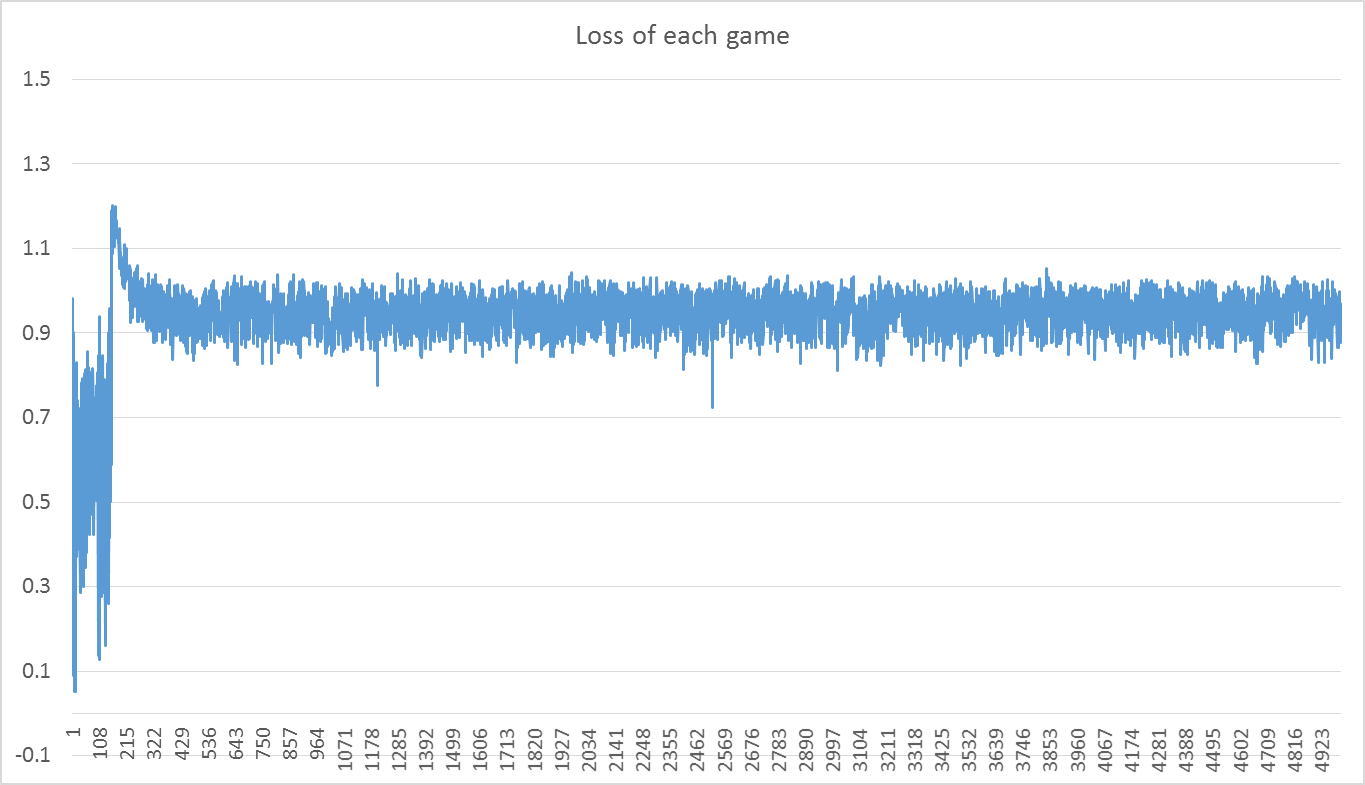
결과

매 게임의 Goal에 도달할 때까지 걸린 step 수이다. (2000은 도달하지 못하고 종료된 경우이다)



처음에는 Goal에 도달하지 못하다가 약 150게임부터 도달하기 시작해서 최저 약 200회 정도의 step에 Goal에 도달하는 것을 확인할 수 있다.

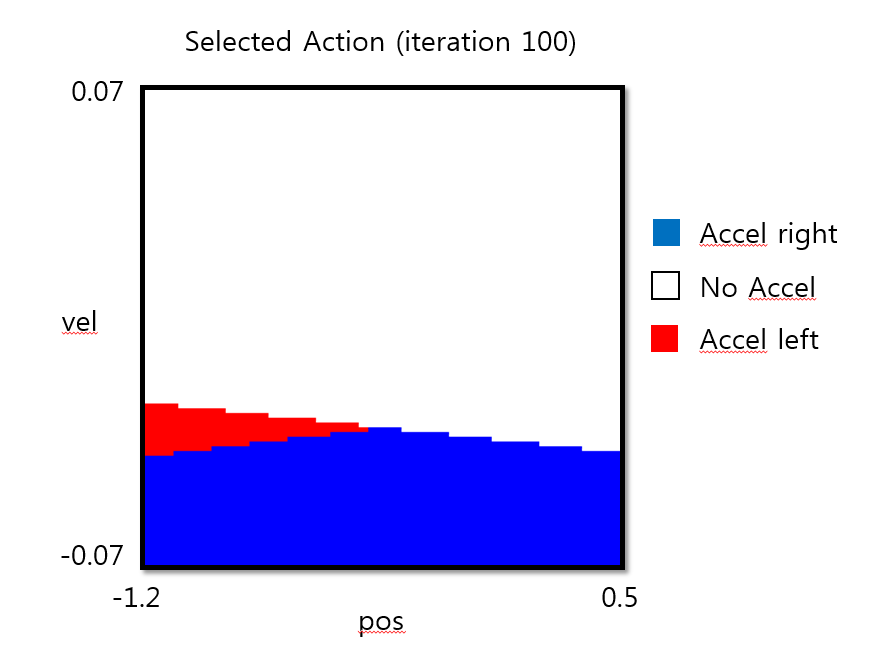
다음으로 각 게임에서의 training의 평균 loss (err²)이다.



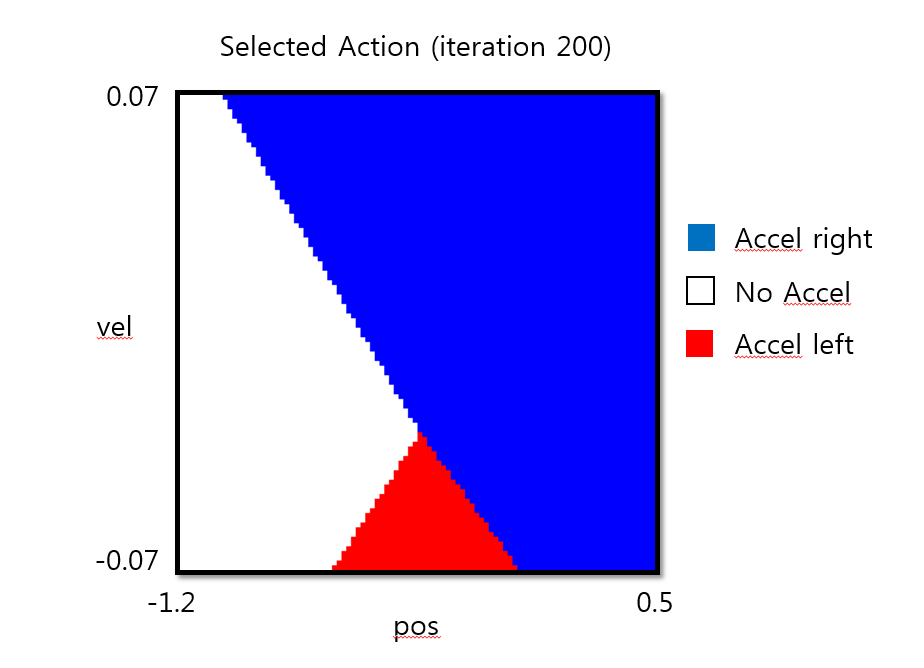
처음엔 Goal에 도달하지 못할 때는 유의미한 reward가 없어 (reward는 Goal에 도달했을 때에만 0이다.) 크게 요동치다가 Goal에 도달하기 시작한 후로 약 0.9 정도를 유지한 것을 알 수 있다.

사실 본 trainig에서는 reward를 goal에 도달할 때까지 모두 -1을 주었기 때문에 게임 도중의 reward 변화가 없어 Q값이 수렴하기 쉽지 않다. reward를 좀더 유의미한 의미를 갖도록 개선하면 더 빠르고 정확하게 수렴할 수 있을 것으로 예상된다.

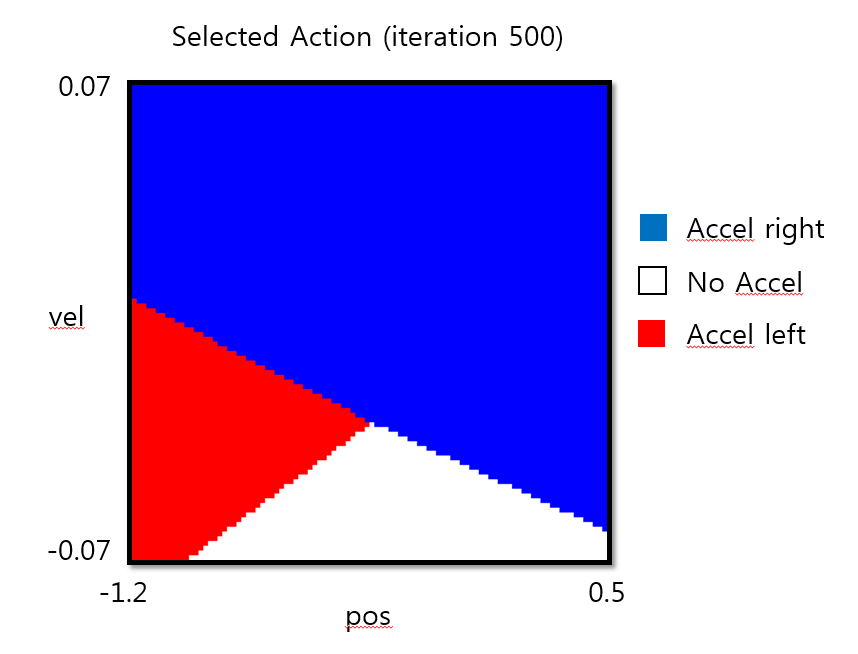
training의 진행에 따른 policy action과 Q value의 변화는 다음과 같다.



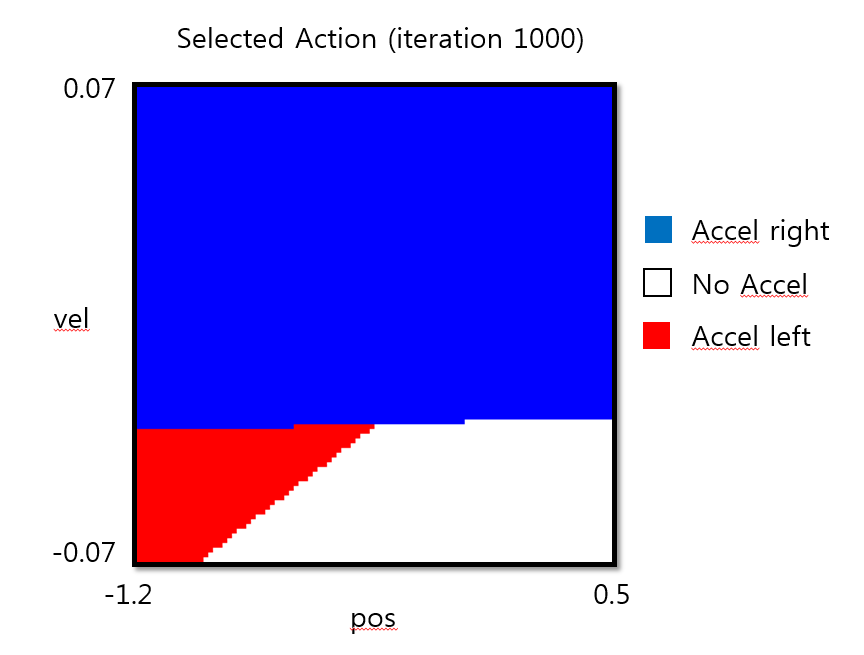
100회에서는 차량이 좌측으로 갈때는 우측으로 엑셀을 줘야 하는 걸 배웠지만 아직 우측으로 가속하는 걸 배우지 못한 상태이다.



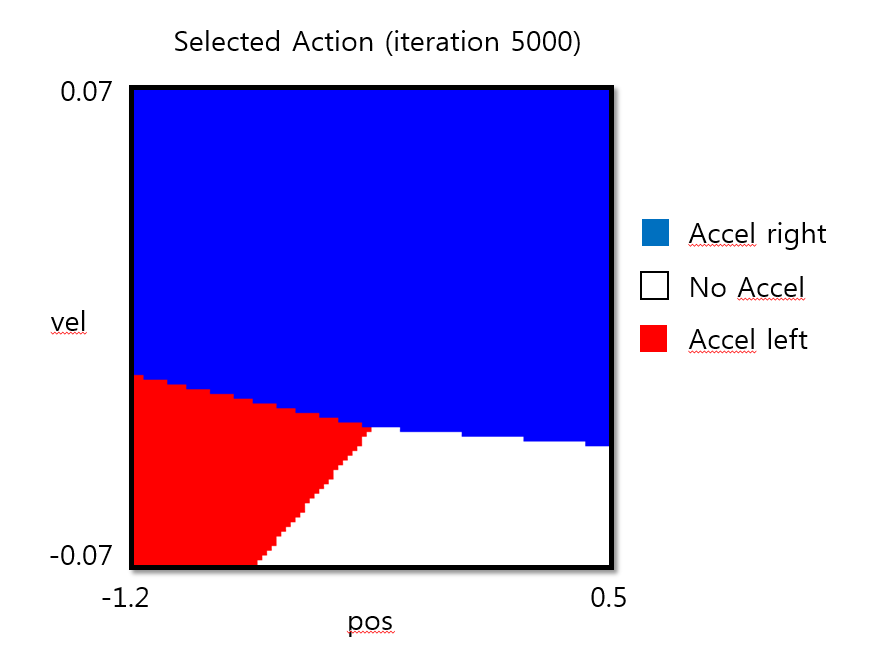
우측으로 가속하기 시작하였다. 아직 왼쪽 끝에서 오른쪽으로 가속하는 걸 배우지 못했다.



모양이 잡혀가는 걸 확인할 수 있다.



대부분의 경우에 우측으로 accel을 한다.



최종 policy이다. 왼쪽에서 좌측으로 달리고 있을 때는 좌측 꼭대기를 거쳐 와야 하므로 좌측으로 가속을 하고, 우측으로 가고 있을 때에는 우측으로 가속을 함을 확인할 수 있다.