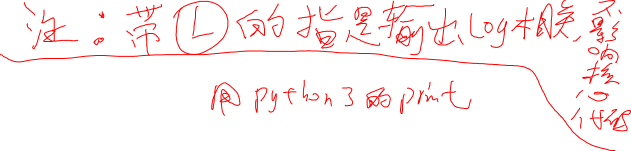
from \_\_future\_\_ import print\_function



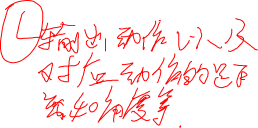
from collections import deque

from hjk\_real\_facing\_people\_webots\_env\_obstacle import WebotsLidarNnEnv

from hjk\_saved\_neural\_qlearning import NeuralQLearner

from hjk\_real\_facing\_people\_webots\_env\_obstacle import actionoutPath

import tensorflow as tf



import numpy as np

import sys

import rospy

import random

import copy



import xml.dom.minidom

import gc

from time import gmtime, strftime

out\_wintimes\_path = "my\_net22/wintimes\_" + strftime("%Y-%m-%d-%H-%M-%S", gmtime()) + ".txt"



out\_test\_wintimes\_path = "my\_net22/test\_wintimes\_" + strftime("%Y-%m-%d-%H-%M-%S", gmtime()) + ".txt"

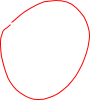


laser\_dim = 16

dst\_dim = 2

facing\_dim = 1

history\_dim = 1



state\_dim = laser\_dim + dst\_dim + facing\_dim + history\_dim



num\_actions = 5

MAX\_STEPS = 150

COLLISION\_THRESHOLD = 0.3

episode\_history = deque(maxlen=100)

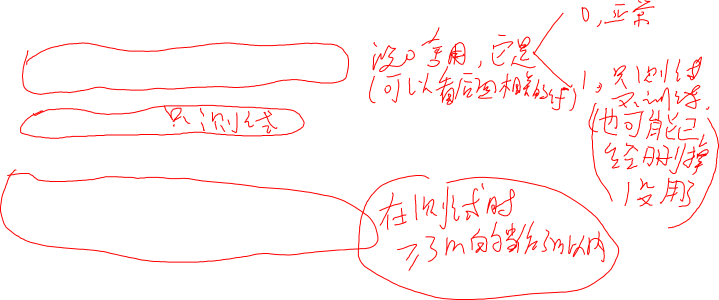
# 0 means to train; 1 means to test for my case ;

py\_function = 0

MAX\_EPISODES = 100000

JUSTTEST = 0

# now when justtest, it will make use. And should use right limit\_sta\_dis\_value and limit\_sta\_dis\_pos



USELIMITSTADIS = 0

LIMITSTADISVALUE = 3 \* 10

LIMITSTADISPOS = laser\_dim # 0~15is laser, 16 is dist

## four layers

num\_layers = 4

num\_neural = [state\_dim, 18, 18, num\_actions]

human\_x = []

human\_y = []

human\_rotation\_z = []

robot\_x = []

robot\_y = []

robot\_rotation\_z = []

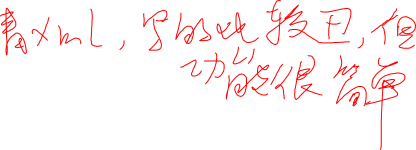


def loadxml():



dom = xml.dom.minidom.parse('src/env/test.xml')

root = dom.documentElement



bb = root.getElementsByTagName('episode')

for i, var in enumerate(bb):

human\_x.append(float(var.getElementsByTagName('human\_x')[0].firstChild.data))

human\_y.append(float(var.getElementsByTagName('human\_y')[0].firstChild.data))

human\_rotation\_z.append(int(var.getElementsByTagName('human\_rotation\_z')[0].firstChild.data))

robot\_x.append(float(var.getElementsByTagName('robot\_x')[0].firstChild.data))

robot\_y.append(float(var.getElementsByTagName('robot\_y')[0].firstChild.data))

robot\_rotation\_z.append(float(var.getElementsByTagName('robot\_rotation\_z')[0].firstChild.data))

def add\_layer(inputs, in\_size, out\_size, w\_name, b\_name, activation\_function=None):

Weights = tf.get\_variable(w\_name, [in\_size, out\_size],

initializer=tf.random\_normal\_initializer(mean=0.0, stddev=0.2))

biases = tf.get\_variable(b\_name, out\_size,

initializer=tf.constant\_initializer(0))

Wx\_plus\_b = tf.matmul(inputs, Weights) + biases

if activation\_function is None:

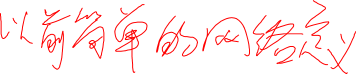
outputs = Wx\_plus\_b

else:

outputs = activation\_function(Wx\_plus\_b)

return outputs

def init\_q\_net(states):



h1 = add\_layer(states, num\_neural[0], num\_neural[1], 'W1', 'b1', activation\_function=tf.nn.relu)

h2 = add\_layer(h1, num\_neural[1], num\_neural[2], 'W2', 'b2', activation\_function=tf.nn.relu)

q = add\_layer(h2, num\_neural[2], num\_neural[3], 'W3', 'b3', activation\_function=None)

return q



def complex\_init\_q\_net(states):

laser = tf.slice(states, [0, 0], [-1, laser\_dim])



goal = tf.slice(states, [0, laser\_dim], [-1, dst\_dim + facing\_dim])

his\_a = tf.slice(states, [0, laser\_dim + dst\_dim + facing\_dim], [-1, history\_dim])



# print(laser)



# print(goal)



# print(his\_a)

laser\_8 = add\_layer(laser, laser\_dim, 8, 'W1', 'b1', activation\_function=tf.nn.relu)

cat\_laser\_8\_his\_a = tf.concat([laser\_8, his\_a], 1)



# print(cat\_laser\_8\_his\_a)

laser\_5 = add\_layer(cat\_laser\_8\_his\_a, 9, 5, 'W2', 'b2', activation\_function=tf.nn.relu)

cat\_laser\_5\_goal = tf.concat([laser\_5, goal], 1)

cat\_r8 = add\_layer(cat\_laser\_5\_goal, 8, 8, 'W3', 'b3', activation\_function=tf.nn.relu)

cat\_5 = add\_layer(cat\_r8, 8, 5, 'W4', 'b4', activation\_function=None)

# h\_8\_and\_1 = tf.concat(1, h\_)

# h2 = add\_layer(h1, num\_neural[1], num\_neural[2], 'W2', 'b2', activation\_function=tf.nn.relu)

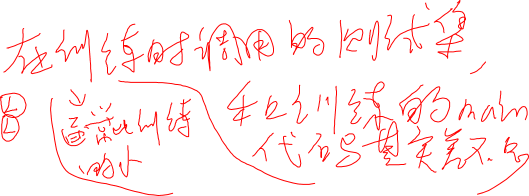
# q = add\_layer(h2, num\_neural[2], num\_neural[3], 'W3', 'b3', activation\_function=None)

return cat\_5

def testtest(path):

env.collision\_threshold = 0.25

xml\_test\_cnt = 0



sum\_steps = 0

l = len(human\_x)

#print('---- ', l)

for i\_episode in xrange(l):

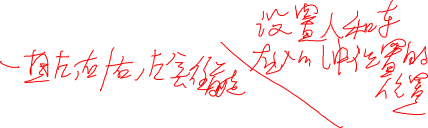
env.my\_case = -1

state = env.reset(1, human\_x[i\_episode], human\_y[i\_episode], human\_rotation\_z[i\_episode], robot\_x[i\_episode],

robot\_y[i\_episode], robot\_rotation\_z[i\_episode])

last\_action = -1

wrongActionTimes = 0



record\_t = -1

for t in xrange(MAX\_STEPS):

record\_t = t

#print("In episode ", i\_episode, ":")

#print('step ' + str(t))

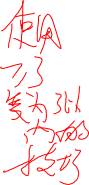
# print("new change : ", state, "sure: ", state[-1])

# print("333333 --- state: ", state[LIMITSTADISPOS], state)

limit\_state = copy.deepcopy(state)

if limit\_state[LIMITSTADISPOS] > LIMITSTADISVALUE:

limit\_state[LIMITSTADISPOS] = LIMITSTADISVALUE - random.random() \* 10



# print("www i don't konw: ", state)

print('tttttttttttttttttttttttttttttttttttttttttttt: ', i\_episode, 'wintimes: ', xml\_test\_cnt)

action = q\_learner.eGreedyAction(limit\_state[np.newaxis, :], False)

if (last\_action == 3 and action == 4) or (last\_action == 4 and action == 3):

wrongActionTimes = wrongActionTimes + 1

if wrongActionTimes == 2:

wrongActionTimes = 0



print('hjk--- ffffff :', action)



action = 0

#print('hjk--action: ', action, 'lastaction', last\_action)

actionout = open(actionoutPath, 'a+')

# print("??????????????????????????????????????????????????????????????????????????????????????")

print('action: ', action, 'lastaction: ', last\_action, file = actionout)

sys.stdout.flush()



actionout.close()

next\_state, reward, done, \_ = env.step(action)

last\_action = action

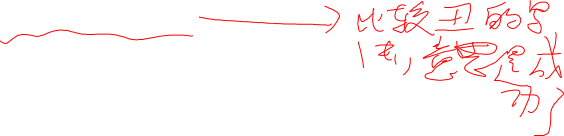
state = next\_state

if done:

if reward >= 500 - 1:

xml\_test\_cnt += 1

sum\_steps += t



out\_test\_wintimes = open(out\_test\_wintimes\_path, 'a+')

print('testround: ', i\_episode, "win!! use steps: ", t, file=out\_test\_wintimes)

out\_test\_wintimes.close()

break

if xml\_test\_cnt == 0:

mean\_steps = 0

else:

mean\_steps = sum\_steps \* 1.0 / xml\_test\_cnt

out\_test\_wintimes = open(out\_test\_wintimes\_path, 'a+')

print('test ', path, "wintimes: ", xml\_test\_cnt, "mean\_steps: ", mean\_steps, file=out\_test\_wintimes)

sys.stdout.flush()

out\_test\_wintimes.close()

env.collision\_threshold = COLLISION\_THRESHOLD

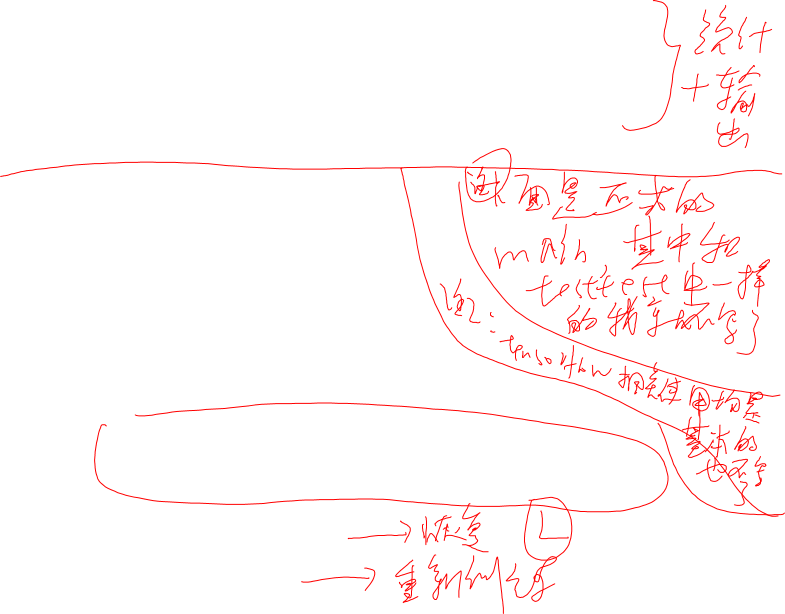
if \_\_name\_\_ == '\_\_main\_\_':

len\_args = len(sys.argv)

path = None

if (len\_args > 1):

path = str(sys.argv[1])



loadxml()

env\_name = 'facing\_people\_webots\_env\_obstacle'

sess = tf.Session()

#### hjk change the learning\_rate to 0.001. nnn.....

optimizer = tf.train.RMSPropOptimizer(learning\_rate=0.004, decay=0.9)

# writer = tf.train.SummaryWriter("/tmp/{}-experiment-1".format(env\_name), graph=sess.graph)

writer = tf.summary.FileWriter("my\_net20/{}-experiment-1".format(env\_name), graph=sess.graph)

if path is not None:

print('resotre net path: ' + path)

else:

print("init")

# restore\_net(sess, path)



q\_learner = NeuralQLearner(sess,



optimizer,

complex\_init\_q\_net,

path,

state\_dim,

num\_actions,

512, # batch\_size=32,

0.5, # init\_exp=0.3, # 0.5, # initial exploration prob

0.1, # final\_exp=0.001, # final exploration prob

# anneal\_steps=10000, # N steps for annealing exploration

200000, # anneal\_steps=2000, # N steps for annealing exploration

10000, # replay\_buffer\_size=10000,

3, # store\_replay\_every=3, # how frequent to store experience

0.9, # discount\_factor=0.9, # discount future rewards

0.01, # target\_update\_rate=0.01,

0.01, # reg\_param=0.01, # regularization constants

5, # max\_gradient=5, # max gradient norms

False, # double\_q\_learning=False,

None, # summary=None,

100 # summary\_every=100

)

# print(sess.run(tf.get\_default\_graph().get\_tensor\_by\_name("q\_network/b3:0")))

# print(sess.run(tf.get\_default\_graph().get\_tensor\_by\_name("target\_network/b3:0")))

env = WebotsLidarNnEnv(laser\_dim, COLLISION\_THRESHOLD)

wintimes = 0

for i\_episode in xrange(MAX\_EPISODES):

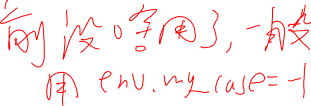
# initialize

if py\_function == 1:

env.my\_case = i\_episode



else:



env.my\_case = -1

state = env.reset()

# print("2222222 --- state: ", state[LIMITSTADISPOS])

total\_rewards = 0

last\_action = -1

wrongActionTimes = 0

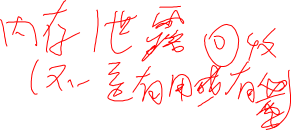
record\_t = -1

for t in xrange(MAX\_STEPS):

gc.collect()



record\_t = t



print("In episode ", i\_episode, ":")

print('step ' + str(t))

# print("new change : ", state, "sure: ", state[-1])

if JUSTTEST == 0:

action = q\_learner.eGreedyAction(state[np.newaxis, :])

else:

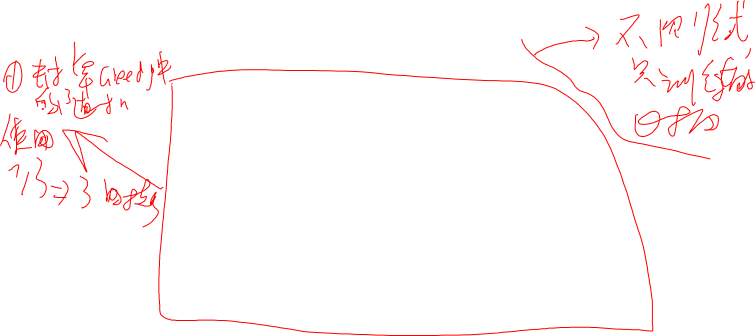
if USELIMITSTADIS == 0:

action = q\_learner.eGreedyAction(state[np.newaxis, :], False)

else:



# print("333333 --- state: ", state[LIMITSTADISPOS], state)



limit\_state = copy.deepcopy(state)

if limit\_state[LIMITSTADISPOS] > LIMITSTADISVALUE:

limit\_state[LIMITSTADISPOS] = LIMITSTADISVALUE - random.random() \* 10

# print("www i don't konw: ", state)

print('hjk--- limit\_state: ', state, limit\_state)

action = q\_learner.eGreedyAction(limit\_state[np.newaxis, :], False)

if (last\_action == 3 and action == 4) or (last\_action == 4 and action == 3):

wrongActionTimes = wrongActionTimes + 1

if wrongActionTimes == 2:

wrongActionTimes = 0

print('hjk--- ffffff :', action)

action = 0

print('hjk--action: ', action, 'lastaction', last\_action)

actionout = open(actionoutPath, 'a+')

# print("??????????????????????????????????????????????????????????????????????????????????????")

print('action: ', action, 'lastaction: ', last\_action, file=actionout)

sys.stdout.flush()

actionout.close()

next\_state, reward, done, \_ = env.step(action)

last\_action = action

total\_rewards += reward

if JUSTTEST == 0:



if state[-1] != -1 and next\_state[-1] != -1:

q\_learner.storeExperience(state, action, reward, next\_state, done)

q\_learner.updateModel(i\_episode)

state = next\_state



if done:

if reward >= 500 - 1:

wintimes += 1

else:

record\_t = -1

break

episode\_history.append(wintimes)

# mean\_rewards = np.mean(episode\_history)

print("Episode {}".format(i\_episode))

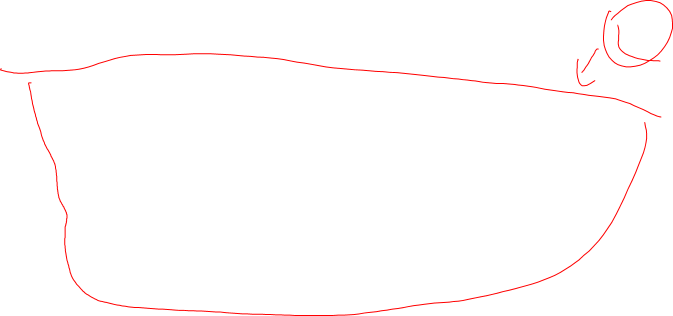
#print("Finished after {} timesteps".format(t + 1))

print("Reward for this episode: {}".format(total\_rewards))

# print("last 99 episodes wintimes: ", episode\_history[-1][0] - episode\_history[0][0])

out\_wintimes = open(out\_wintimes\_path, "a+")

print("Episode: ", i\_episode, " ", q\_learner.exploration, " ", "last 99 episodes wintimes: ",



episode\_history[-1] - episode\_history[0], 'step: ', record\_t,

file=out\_wintimes)

sys.stdout.flush()

out\_wintimes.close()

# print("Average reward for last 100 episodes: {}".format(mean\_rewards))

if JUSTTEST == 0:

if i\_episode >= 200 and i\_episode % 200 == 0:

path = 'my\_net22/' + env\_name + '\_' + str(num\_layers) \

+ 'layers\_' + str(i\_episode + 1) + 'epsiode\_' + \

strftime("%Y-%m-%d-%H-%M-%S", gmtime()) + 'restore\_network\_rerandom'

q\_learner.save\_net(path)



testtest(str(i\_episode + 1))、

