import numpy as np

import pandas as pd

from subprocess import check\_output

print(check\_output(["ls", "../input"]).decode("utf8"))

import time

import copy

import numpy as np

import pandas as pd

import chainer

import chainer.functions as F

import chainer.links as L

from plotly import tools

from plotly.graph\_objs import \*

from plotly.offline import init\_notebook\_mode, iplot, iplot\_mpl

init\_notebook\_mode()

data = pd.read\_csv('../input/Data/Stocks/goog.us.txt')

data['Date'] = pd.to\_datetime(data['Date'])

data = data.set\_index('Date')

print(data.index.min(), data.index.max())

data.head()

date\_split = '2016-01-01'

train = data[:date\_split]

test = data[date\_split:]

len(train), len(test)

def plot\_train\_test(train, test, date\_split):

data = [

Candlestick(x=train.index, open=train['Open'], high=train['High'], low=train['Low'], close=train['Close'], name='train'),

Candlestick(x=test.index, open=test['Open'], high=test['High'], low=test['Low'], close=test['Close'], name='test')

]

layout = {

'shapes': [

{'x0': date\_split, 'x1': date\_split, 'y0': 0, 'y1': 1, 'xref': 'x', 'yref': 'paper', 'line': {'color': 'rgb(0,0,0)', 'width': 1}}

],

'annotations': [

{'x': date\_split, 'y': 1.0, 'xref': 'x', 'yref': 'paper', 'showarrow': False, 'xanchor': 'left', 'text': ' test data'},

{'x': date\_split, 'y': 1.0, 'xref': 'x', 'yref': 'paper', 'showarrow': False, 'xanchor': 'right', 'text': 'train data '}

]

}

figure = Figure(data=data, layout=layout)

iplot(figure)

plot\_train\_test(train, test, date\_split)

class Environment1:

def \_\_init\_\_(self, data, history\_t=90):

self.data = data

self.history\_t = history\_t

self.reset()

def reset(self):

self.t = 0

self.done = False

self.profits = 0

self.positions = []

self.position\_value = 0

self.history = [0 for \_ in range(self.history\_t)]

return [self.position\_value] + self.history # obs

def step(self, act):

reward = 0

# act = 0: stay, 1: buy, 2: sell

if act == 1:

self.positions.append(self.data.iloc[self.t, :]['Close'])

elif act == 2: # sell

if len(self.positions) == 0:

reward = -1

else:

profits = 0

for p in self.positions:

profits += (self.data.iloc[self.t, :]['Close'] - p)

reward += profits

self.profits += profits

self.positions = []

# set next time

self.t += 1

self.position\_value = 0

for p in self.positions:

self.position\_value += (self.data.iloc[self.t, :]['Close'] - p)

self.history.pop(0)

self.history.append(self.data.iloc[self.t, :]['Close'] - self.data.iloc[(self.t-1), :]['Close'])

# clipping reward

if reward > 0:

reward = 1

elif reward < 0:

reward = -1

return [self.position\_value] + self.history, reward, self.done

env = Environment1(train)

print(env.reset())

for \_ in range(3):

pact = np.random.randint(3)

print(env.step(pact))

def plot\_loss\_reward(total\_losses, total\_rewards):

figure = tools.make\_subplots(rows=1, cols=2, subplot\_titles=('loss', 'reward'), print\_grid=False)

figure.append\_trace(Scatter(y=total\_losses, mode='lines', line=dict(color='skyblue')), 1, 1)

figure.append\_trace(Scatter(y=total\_rewards, mode='lines', line=dict(color='orange')), 1, 2)

figure['layout']['xaxis1'].update(title='epoch')

figure['layout']['xaxis2'].update(title='epoch')

figure['layout'].update(height=400, width=900, showlegend=False)

iplot(figure)

def plot\_train\_test\_by\_q(train\_env, test\_env, Q, algorithm\_name):

# train

pobs = train\_env.reset()

train\_acts = []

train\_rewards = []

for \_ in range(len(train\_env.data)-1):

pact = Q(np.array(pobs, dtype=np.float32).reshape(1, -1))

pact = np.argmax(pact.data)

train\_acts.append(pact)

obs, reward, done = train\_env.step(pact)

train\_rewards.append(reward)

pobs = obs

train\_profits = train\_env.profits

# test

pobs = test\_env.reset()

test\_acts = []

test\_rewards = []

for \_ in range(len(test\_env.data)-1):

pact = Q(np.array(pobs, dtype=np.float32).reshape(1, -1))

pact = np.argmax(pact.data)

test\_acts.append(pact)

obs, reward, done = test\_env.step(pact)

test\_rewards.append(reward)

pobs = obs

test\_profits = test\_env.profits

# plot

train\_copy = train\_env.data.copy()

test\_copy = test\_env.data.copy()

train\_copy['act'] = train\_acts + [np.nan]

train\_copy['reward'] = train\_rewards + [np.nan]

test\_copy['act'] = test\_acts + [np.nan]

test\_copy['reward'] = test\_rewards + [np.nan]

train0 = train\_copy[train\_copy['act'] == 0]

train1 = train\_copy[train\_copy['act'] == 1]

train2 = train\_copy[train\_copy['act'] == 2]

test0 = test\_copy[test\_copy['act'] == 0]

test1 = test\_copy[test\_copy['act'] == 1]

test2 = test\_copy[test\_copy['act'] == 2]

act\_color0, act\_color1, act\_color2 = 'gray', 'cyan', 'magenta'

data = [

Candlestick(x=train0.index, open=train0['Open'], high=train0['High'], low=train0['Low'], close=train0['Close'], increasing=dict(line=dict(color=act\_color0)), decreasing=dict(line=dict(color=act\_color0))),

Candlestick(x=train1.index, open=train1['Open'], high=train1['High'], low=train1['Low'], close=train1['Close'], increasing=dict(line=dict(color=act\_color1)), decreasing=dict(line=dict(color=act\_color1))),

Candlestick(x=train2.index, open=train2['Open'], high=train2['High'], low=train2['Low'], close=train2['Close'], increasing=dict(line=dict(color=act\_color2)), decreasing=dict(line=dict(color=act\_color2))),

Candlestick(x=test0.index, open=test0['Open'], high=test0['High'], low=test0['Low'], close=test0['Close'], increasing=dict(line=dict(color=act\_color0)), decreasing=dict(line=dict(color=act\_color0))),

Candlestick(x=test1.index, open=test1['Open'], high=test1['High'], low=test1['Low'], close=test1['Close'], increasing=dict(line=dict(color=act\_color1)), decreasing=dict(line=dict(color=act\_color1))),

Candlestick(x=test2.index, open=test2['Open'], high=test2['High'], low=test2['Low'], close=test2['Close'], increasing=dict(line=dict(color=act\_color2)), decreasing=dict(line=dict(color=act\_color2)))

]

title = '{}: train s-reward {}, profits {}, test s-reward {}, profits {}'.format(

algorithm\_name,

int(sum(train\_rewards)),

int(train\_profits),

int(sum(test\_rewards)),

int(test\_profits)

)

layout = {

'title': title,

'showlegend': False,

'shapes': [

{'x0': date\_split, 'x1': date\_split, 'y0': 0, 'y1': 1, 'xref': 'x', 'yref': 'paper', 'line': {'color': 'rgb(0,0,0)', 'width': 1}}

],

'annotations': [

{'x': date\_split, 'y': 1.0, 'xref': 'x', 'yref': 'paper', 'showarrow': False, 'xanchor': 'left', 'text': ' test data'},

{'x': date\_split, 'y': 1.0, 'xref': 'x', 'yref': 'paper', 'showarrow': False, 'xanchor': 'right', 'text': 'train data '}

]

}

figure = Figure(data=data, layout=layout)

iplot(figure)

def train\_dddqn(env):

class Q\_Network(chainer.Chain):

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

super(Q\_Network, self).\_\_init\_\_(

fc1 = L.Linear(input\_size, hidden\_size),

fc2 = L.Linear(hidden\_size, hidden\_size),

fc3 = L.Linear(hidden\_size, hidden\_size//2),

fc4 = L.Linear(hidden\_size, hidden\_size//2),

state\_value = L.Linear(hidden\_size//2, 1),

advantage\_value = L.Linear(hidden\_size//2, output\_size)

)

self.input\_size = input\_size

self.hidden\_size = hidden\_size

self.output\_size = output\_size

def \_\_call\_\_(self, x):

h = F.relu(self.fc1(x))

h = F.relu(self.fc2(h))

hs = F.relu(self.fc3(h))

ha = F.relu(self.fc4(h))

state\_value = self.state\_value(hs)

advantage\_value = self.advantage\_value(ha)

advantage\_mean = (F.sum(advantage\_value, axis=1)/float(self.output\_size)).reshape(-1, 1)

q\_value = F.concat([state\_value for \_ in range(self.output\_size)], axis=1) + (advantage\_value - F.concat([advantage\_mean for \_ in range(self.output\_size)], axis=1))

return q\_value

def reset(self):

self.zerograds()

Q = Q\_Network(input\_size=env.history\_t+1, hidden\_size=100, output\_size=3)

Q\_ast = copy.deepcopy(Q)

optimizer = chainer.optimizers.Adam()

optimizer.setup(Q)

epoch\_num = 50

step\_max = len(env.data)-1

memory\_size = 200

batch\_size = 50

epsilon = 1.0

epsilon\_decrease = 1e-3

epsilon\_min = 0.1

start\_reduce\_epsilon = 200

train\_freq = 10

update\_q\_freq = 20

gamma = 0.97

show\_log\_freq = 5

memory = []

total\_step = 0

total\_rewards = []

total\_losses = []

start = time.time()

for epoch in range(epoch\_num):

pobs = env.reset()

step = 0

done = False

total\_reward = 0

total\_loss = 0

while not done and step < step\_max:

# select act

pact = np.random.randint(3)

if np.random.rand() > epsilon:

pact = Q(np.array(pobs, dtype=np.float32).reshape(1, -1))

pact = np.argmax(pact.data)

# act

obs, reward, done = env.step(pact)

# add memory

memory.append((pobs, pact, reward, obs, done))

if len(memory) > memory\_size:

memory.pop(0)

# train or update q

if len(memory) == memory\_size:

if total\_step % train\_freq == 0:

shuffled\_memory = np.random.permutation(memory)

memory\_idx = range(len(shuffled\_memory))

for i in memory\_idx[::batch\_size]:

batch = np.array(shuffled\_memory[i:i+batch\_size])

b\_pobs = np.array(batch[:, 0].tolist(), dtype=np.float32).reshape(batch\_size, -1)

b\_pact = np.array(batch[:, 1].tolist(), dtype=np.int32)

b\_reward = np.array(batch[:, 2].tolist(), dtype=np.int32)

b\_obs = np.array(batch[:, 3].tolist(), dtype=np.float32).reshape(batch\_size, -1)

b\_done = np.array(batch[:, 4].tolist(), dtype=np.bool)

q = Q(b\_pobs)

indices = np.argmax(q.data, axis=1)

maxqs = Q\_ast(b\_obs).data

target = copy.deepcopy(q.data)

for j in range(batch\_size):

target[j, b\_pact[j]] = b\_reward[j]+gamma\*maxqs[j, indices[j]]\*(not b\_done[j])

Q.reset()

loss = F.mean\_squared\_error(q, target)

total\_loss += loss.data

loss.backward()

optimizer.update()

if total\_step % update\_q\_freq == 0:

Q\_ast = copy.deepcopy(Q)

# epsilon

if epsilon > epsilon\_min and total\_step > start\_reduce\_epsilon:

epsilon -= epsilon\_decrease

# next step

total\_reward += reward

pobs = obs

step += 1

total\_step += 1

total\_rewards.append(total\_reward)

total\_losses.append(total\_loss)

if (epoch+1) % show\_log\_freq == 0:

log\_reward = sum(total\_rewards[((epoch+1)-show\_log\_freq):])/show\_log\_freq

log\_loss = sum(total\_losses[((epoch+1)-show\_log\_freq):])/show\_log\_freq

elapsed\_time = time.time()-start

print('\t'.join(map(str, [epoch+1, epsilon, total\_step, log\_reward, log\_loss, elapsed\_time])))

start = time.time()

return Q, total\_losses, total\_rewards

plot\_loss\_reward(total\_losses, total\_rewards)

plot\_train\_test\_by\_q(Environment1(train), Environment1(test), Q, 'Dueling Double DQN')