import tensorflow as tf

import numpy as np

from numpy import random

import matplotlib.pyplot as plt

from IPython import display

import pandas

% matplotlib inline

train\_batch\_size = 173

valid\_batch\_size = 173

time\_size = 5

f\_af\_train = np.empty([train\_batch\_size,time\_size,2],dtype = float)

risk\_train = np.empty([train\_batch\_size,time\_size,1],dtype = int)

f\_af\_valid = np.empty([valid\_batch\_size,time\_size,2],dtype = float)

risk\_valid = np.empty([valid\_batch\_size,time\_size,1],dtype = int)

f\_train = pandas.read\_csv("D:/rnn\_test3.csv", usecols=[0], dtype=float,header = 0)

af\_train = pandas.read\_csv("D:/rnn\_test3.csv", usecols=[1], dtype=float,header = 0)

risk\_train\_input = pandas.read\_csv("D:/rnn\_test3.csv", usecols=[2], dtype=int,header = 0)

for i in range(train\_batch\_size):

f\_af\_train[i, :, 0] = f\_train.transpose()[i]

f\_af\_train[i, :, 1] = af\_train.transpose()[i]

risk\_train[i, :, 0] = risk\_train\_input.transpose()[i]

f\_valid = pandas.read\_csv("D:/rnn\_val3.csv", usecols=[0], dtype=float,header = 0)

af\_valid = pandas.read\_csv("D:/rnn\_val3.csv", usecols=[1], dtype=float,header = 0)

risk\_valid\_input = pandas.read\_csv("D:/rnn\_val3.csv", usecols=[2], dtype=int,header = 0)

for i in range(valid\_batch\_size):

f\_af\_valid[i, :, 0] = f\_valid.transpose()[i]

f\_af\_valid[i, :, 1] = af\_valid.transpose()[i]

risk\_valid[i, :, 0] = risk\_valid\_input.transpose()[i]

import tensorflow as tf

class GRU:

def \_\_init\_\_(self, input\_dimensions, hidden\_size, dtype=tf.float64):

self.input\_dimensions = input\_dimensions

self.hidden\_size = hidden\_size

self.Wr = tf.Variable(tf.truncated\_normal(dtype=dtype, shape=(self.input\_dimensions, self.hidden\_size), mean=0, stddev=0.01), name='Wr')

self.Wz = tf.Variable(tf.truncated\_normal(dtype=dtype, shape=(self.input\_dimensions, self.hidden\_size), mean=0, stddev=0.01), name='Wz')

self.Wh = tf.Variable(tf.truncated\_normal(dtype=dtype, shape=(self.input\_dimensions, self.hidden\_size), mean=0, stddev=0.01), name='Wh')

self.Ur = tf.Variable(tf.truncated\_normal(dtype=dtype, shape=(self.hidden\_size, self.hidden\_size), mean=0, stddev=0.01), name='Ur')

self.Uz = tf.Variable(tf.truncated\_normal(dtype=dtype, shape=(self.hidden\_size, self.hidden\_size), mean=0, stddev=0.01), name='Uz')

self.Uh = tf.Variable(tf.truncated\_normal(dtype=dtype, shape=(self.hidden\_size, self.hidden\_size), mean=0, stddev=0.01), name='Uh')

self.br = tf.Variable(tf.truncated\_normal(dtype=dtype, shape=(self.hidden\_size,), mean=0, stddev=0.01), name='br')

self.bz = tf.Variable(tf.truncated\_normal(dtype=dtype, shape=(self.hidden\_size,), mean=0, stddev=0.01), name='bz')

self.bh = tf.Variable(tf.truncated\_normal(dtype=dtype, shape=(self.hidden\_size,), mean=0, stddev=0.01), name='bh')

self.input\_layer = tf.placeholder(dtype=tf.float64, shape=(None, None, input\_dimensions), name='input')

self.x\_t = tf.transpose(self.input\_layer, [1, 0, 2], name='x\_t')

self.h\_0 = tf.matmul(self.x\_t[0, :, :], tf.zeros(dtype=tf.float64, shape=(input\_dimensions, hidden\_size)), name='h\_0')

self.h\_t\_transposed = tf.scan(self.forward\_pass, self.x\_t, initializer=self.h\_0, name='h\_t\_transposed')

self.h\_t = tf.transpose(self.h\_t\_transposed, [1, 0, 2], name='h\_t')

def forward\_pass(self, h\_tm1, x\_t):

z\_t = tf.sigmoid(tf.matmul(x\_t, self.Wz) + tf.matmul(h\_tm1, self.Uz) + self.bz)

r\_t = tf.sigmoid(tf.matmul(x\_t, self.Wr) + tf.matmul(h\_tm1, self.Ur) + self.br)

h\_proposal = tf.tanh(tf.matmul(x\_t, self.Wh) + tf.matmul(tf.multiply(r\_t, h\_tm1), self.Uh) + self.bh)

h\_t = tf.multiply(1 - z\_t, h\_tm1) + tf.multiply(z\_t, h\_proposal)

return h\_t

input\_dimensions = 2

hidden\_size = 5

session = tf.Session()

gru = GRU(input\_dimensions, hidden\_size)

W\_output = tf.Variable(tf.truncated\_normal(dtype=tf.float64, shape=(hidden\_size, 1), mean=0, stddev=0.01))

b\_output = tf.Variable(tf.truncated\_normal(dtype=tf.float64, shape=(1,), mean=0, stddev=0.01))

output = tf.map\_fn(lambda h\_t: tf.matmul(h\_t, W\_output) + b\_output, gru.h\_t)

expected\_output = tf.placeholder(dtype=tf.float64, shape=(train\_batch\_size, time\_size, 1), name='expected\_output')

loss = tf.reduce\_sum(0.5 \* tf.pow(output - expected\_output, 2)) / float(valid\_batch\_size)

train\_step = tf.train.AdamOptimizer().minimize(loss)

init\_variables = tf.global\_variables\_initializer()

session.run(init\_variables)

train\_losses = []

validation\_losses = []

for epoch in range(16001):

\_,train\_loss = session.run([train\_step, loss], feed\_dict={gru.input\_layer: f\_af\_train, expected\_output: risk\_train})

validation\_loss = session.run(loss, feed\_dict={gru.input\_layer: f\_af\_valid, expected\_output: risk\_valid})

train\_losses += [train\_loss]

validation\_losses += [validation\_loss]

if epoch % 1000 == 0:

plt.plot(train\_losses, '-b', label='Train loss')

plt.plot(validation\_losses, '-r', label='Validation loss')

plt.legend(loc=0)

plt.title('Loss')

plt.xlabel('Iteration')

plt.ylabel('Loss')

plt.show()

print('Iteration: %d, train loss: %.4f, test loss: %.4f' % (epoch, train\_loss, validation\_loss))

predicate\_data\_size = 57

f\_pre = pandas.read\_csv("D:/rnn\_pre3-faf.csv", usecols=[0], dtype=float,header = 0)

af\_pre = pandas.read\_csv("D:/rnn\_pre3-faf.csv", usecols=[1], dtype=float,header = 0)

file\_saved = open("D:/rnn\_predicated.csv", "w")

for i in range(predicate\_data\_size):

X\_custom = np.empty([predicate\_data\_size,5,2],dtype = float)

a = f\_pre.transpose()[i]

b = af\_pre.transpose()[i]

bitstring\_length = 20

X\_custom[0, :, 0] = a

X\_custom[0, :, 1] = b

y\_predicted = session.run(output, feed\_dict={gru.input\_layer: X\_custom})

y\_predicted\_0 = y\_predicted[0][0]

print(y\_predicted\_0)

if (y\_predicted\_0 < 0.5):

y\_predicted\_0 = 0

elif y\_predicted\_0 >= 0.5 and y\_predicted\_0 < 1.5:

y\_predicted\_0 = 1

elif y\_predicted\_0 >= 1.5 and y\_predicted\_0 < 2.5:

y\_predicted\_0 = 2

else :

y\_predicted\_0 = 3

file\_saved.write(str(int(y\_predicted\_0)))

file\_saved.write('\n')

file\_saved.close()