Data generator (C++)

#include <iostream>

#include <cstdio>

#include <cstring>

#include <ctime>

#include <random>

#include <algorithm>

#define MAXN 10005

#define ri register int

#define int ll

typedef long long ll;

const int N = 5000;

using namespace std;

int curt[MAXN], newt[MAXN];

int curh[MAXN], newh[MAXN];

int cura[MAXN], newa[MAXN];

int curp[MAXN], newp[MAXN];

int people[MAXN];

int sum = 0;

inline int read\_int()

{

register int ret = 0, f = 1; register char c = getchar();

while(c < '0' || c > '9') {if(c == '-') f = -1; c = getchar();}

while(c >= '0' && c <= '9') {ret = (ret << 1) + (ret << 3) + int(c - 48); c = getchar();}

return ret \* f;

}

inline void file()

{

freopen("H.csv", "r", stdin);

freopen("testdata.in", "w", stdout);

}

signed main()

{

//Vancouver

file();

srand(time(NULL));

default\_random\_engine e1, e2;

normal\_distribution <double> H(75, 5);

normal\_distribution <double> T(24, 0.3);

for(ri i = 1; i <= N; i++)

{

cin >> curh[i];

if(curh[i] < 70)

newh[i] = 70;

else

newh[i] = 59;

}

fclose(stdin);

freopen("T.csv", "r", stdin);

for(ri i = 1; i <= N; i++)

{

double tmp;

cin >> tmp;

tmp -= 272.15;

curt[i] = int(tmp + 0.5);

sum += curt[i];

}

ri cnt = 0;

for(ri i = 1; i <= N; i++)

{

if(curt[i] \* N < sum)

newt[i] = 30;

else

newt[i] = 20;

}

fclose(stdin);

default\_random\_engine e3, e4;

normal\_distribution <double> S(50, 10);

normal\_distribution <double> L(300, 20);

for(ri i = 1; i <= N; i++)

{

ri opt = rand() % 20;

if(opt == 0)

curp[i] = int(L(e3) + 0.5);

else

curp[i] = int(S(e4) + 0.5);

newp[i] = curp[i] >= 100 ? 1 : 0;

}

for(ri i = 1; i <= N; i++)

cura[i] = rand() % 5 == 0 ? 3 : rand() % 2, newa[i] = cura[i] >= 2 ? 1 : 0;

puts("4 4");

for(ri i = 1; i <= N; i++)

printf("%lld %lld %lld %lld %lld %lld %lld %lld\n", curt[i], curh[i], cura[i], curp[i], newt[i], newh[i], newa[i], newp[i]);

return 0;

}

Back Propagation neuron network (Python)

import tensorflow as tf

from time import time

import numpy as np

from random import randint

from os import system

#from draw import acc\_draw

HLAYERS = [100, 100]

BATCH\_SIZE\_MIN = 20

BATCH\_RATIO = 0.015

TRAINING\_STEPS = 10000

TRAINSET\_RATIO = 0.8

LEARNING\_RATE\_BASE = 0.003

LEARNING\_RATE\_DECAY = 0.8

REGULARIZATION\_RATE = 0.0001

MOVING\_AVERAGE\_DECAY = 0.99

FILE = 'testdata.in'

INDEX = [0, 1]

NAME = 'test-{mode}-{t}'.format(mode=INDEX, t=time())

def get\_data(fil, index):

with open(fil, "r") as f:

raw = f.read().split("\n")

print(raw)

data\_raw = [[eval(y) for y in x.split(' ')] for x in raw[:-1]]

print(data\_raw)

numx = data\_raw[0][0]

print(numx)

y\_ = [x[index+numx] for x in data\_raw[1:]]

print(y\_)

yd = max(y\_) - min(y\_)

ymin = min(y\_)

dataset = [[tuple(x[0:numx]), tuple([int(i==(x[index+numx]-ymin)) for i in range(yd+1)])] for x in data\_raw[1:]]

print(dataset)

return dataset #return get\_dataset(dataset, len(dataset))

def get\_dataset(dataset, num):

out = []

while not len(out) == num:

temp = randint(0, len(dataset)-1)

if not temp in out:

out.append(temp)

return [dataset[x] for x in out]

def inference(input\_tensor, avg\_class, weights):

if None == avg\_class:

'''

layer1 = tf.nn.relu(tf.matmul(input\_tensor, weights1) + biases1)

return tf.matmul(layer1, weights2) + biases2

'''

output\_tensor = tf.nn.relu(tf.matmul(input\_tensor, weights[0][0]) + weights[1][0])

for i in range(1, len(weights) - 1):

output\_tensor = tf.nn.relu(tf.matmul(output\_tensor, weights[0][i]) + weights[1][i])

return tf.matmul(output\_tensor, weights[0][-1]) + weights[1][-1]

else:

'''

layer1 = tf.nn.relu(tf.matmul(input\_tensor, avg\_class.average(weights1)) + avg\_class.average(biases1))

return tf.matmul(layer1, avg\_class.average(weights2)) + avg\_class.average(biases2)

'''

output\_tensor = tf.nn.relu(tf.matmul(input\_tensor, avg\_class.average(weights[0][0])) + avg\_class.average(weights[1][0]))

for i in range(1, len(weights) - 1):

output\_tensor = tf.nn.relu(tf.matmul(output\_tensor, avg\_class.average(weights[0][i])) + avg\_class.average(weights[1][i]))

return tf.matmul(output\_tensor, avg\_class.average(weights[0][-1])) + avg\_class.average(weights[1][-1])

def zfy\_bp(dataset, name):

DATA\_NUM = len(dataset)

INPUT\_NODE = len(dataset[0][0])

OUTPUT\_NODE = len(dataset[0][1])

LAYER = [INPUT\_NODE] + HLAYERS + [OUTPUT\_NODE]

BATCH\_SIZE = min(BATCH\_SIZE\_MIN, BATCH\_RATIO\*DATA\_NUM)

x = tf.placeholder(tf.float32, [None, INPUT\_NODE], name='x-input')

y\_ = tf.placeholder(tf.float32, [None, OUTPUT\_NODE], name='y-input')

'''

weights1 = tf.Variable(tf.truncated\_normal([INPUT\_NODE, LAYER1\_NODE], stddev=0.1))

biases1 = tf.Variable(tf.constant(0.1, shape=[LAYER1\_NODE]))

weights2 = tf.Variable(tf.truncated\_normal([LAYER1\_NODE, OUTPUT\_NODE], stddev=0.1))

biases2 = tf.Variable(tf.constant(0.1, shape=[OUTPUT\_NODE]))

'''

weights = [[], []]

for i in range(len(LAYER) - 1):

weights[0].append(tf.Variable(tf.truncated\_normal([LAYER[i], LAYER[i+1]], stddev=0.1)))

weights[1].append(tf.Variable(tf.constant(1.0, shape=[LAYER[i+1]])))

y = inference(x, None, weights)

global\_step = tf.Variable(0, trainable=False)

variable\_averages = tf.train.ExponentialMovingAverage(MOVING\_AVERAGE\_DECAY, global\_step)

variable\_averages\_op = variable\_averages.apply(tf.trainable\_variables())

average\_y = inference(x, variable\_averages, weights)

cross\_entropy = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(logits=y, labels=tf.argmax(y\_, 1))

cross\_entropy\_mean = tf.reduce\_mean(cross\_entropy)

regularizer = tf.contrib.layers.l2\_regularizer(REGULARIZATION\_RATE)

regularization = 0

for i in range(len(weights)):

regularization = regularization + regularizer(weights[0][i])

loss = cross\_entropy\_mean + regularization

learning\_rate = tf.train.exponential\_decay(LEARNING\_RATE\_BASE, global\_step, DATA\_NUM / BATCH\_SIZE, LEARNING\_RATE\_DECAY)

train\_step = tf.train.GradientDescentOptimizer(learning\_rate).minimize(loss, global\_step=global\_step)

with tf.control\_dependencies([train\_step, variable\_averages\_op]):

train\_op = tf.no\_op(name='train')

correct\_prediction = tf.equal(tf.argmax(average\_y, 1), tf.argmax(y\_, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

saver = tf.train.Saver()

with tf.Session() as sess:

tf.global\_variables\_initializer().run()

testset = dataset[int(DATA\_NUM\*TRAINSET\_RATIO):-1]

trainset = dataset[0:int(DATA\_NUM\*TRAINSET\_RATIO)]

X, Y = [a[0] for a in testset], [a[1] for a in testset]

print(type(X))

test\_feed = {x: X[0:-1], y\_: Y[0:-1]}

for i in range(TRAINING\_STEPS):

validate\_acc = sess.run(accuracy, feed\_dict=test\_feed)

print("after {num} training steps, validation accuracy using average model is {acc}".format(num=i, acc=validate\_acc))

start = (i \* BATCH\_SIZE) % int(DATA\_NUM\*TRAINSET\_RATIO)

end = min(start+BATCH\_SIZE, int(DATA\_NUM\*TRAINSET\_RATIO))

sess.run(train\_op, feed\_dict={x: [a[0] for a in trainset[start:end]], y\_: [a[1] for a in trainset[start:end]]})

if 0 == i % 1000:

saver.save(sess, "{name}/{name}-{stp}".format(name=name, stp=i, t=int(time())))

with open("{name}/{name}.acc".format(name=name), "a+") as f:

f.write("{acc}\n".format(stp=i, t=int(time()), acc=validate\_acc))

def main(argv=None):

for i in INDEX:

#system('mkdir test-{ind}-{t}'.format(ind=i, t=time()))

name = 'test-{mode}-{t}'.format(mode=i, t=time())

zfy\_bp(get\_data(FILE, i), name)

system('python {}'.format('draw.py'))

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

main()

Result Visualization (Python)

from os import listdir

import numpy as np

import matplotlib.pyplot as plt

from time import time

from os import system

def acc\_draw(num):

name = 'figs-{}'.format(int(time()))

dirs = [[x for x in listdir('.') if 'test-{}'.format(i) == x[0:6]] for i in num]

print(dirs)

for x in dirs: x.sort()

f = [open('{name}/{name}.acc'.format(name=dirs[i][-1]), 'r') for i in range(len(dirs)) if not [] == dirs[i]]

raw = [f[i].read().split('\n')[:-1] for i in range(len(dirs))]

print(len(raw))

system('mkdir {}'.format(name))

for i in range(len(raw)):

x = np.array(range(len(raw[i])))

y = np.array([eval(k) for k in raw[i]])

plt.figure(i)

plt.plot(x, y)

plt.xlabel('steps of training')

plt.ylabel('accuracy')

plt.savefig('{name}/BPNN-{num}.jpg'.format(name=name, num=i))

acc\_draw([0, 1, 2, 3])

ARIMA model (MATLAB)