# 球磨机故障诊断的软件系统设计

当前，大数据分析最流行的编程语言便是Python。Python是一种简单易学，功能强大的编程语言，它有高效率的高层数据结构，简单而有效地实现面向对象编程。Python简洁的语法和对动态输入的支持，再加上解释性语言的本质，使得它在大多数平台上的许多领域都是一个理想的脚本语言，特别适用于快速的应用程序开发。（来源：python官方文档）

而深度学习则需要深度学习相关库的支持。我们选择使用由谷歌开源的TensorFlow。TensorFlow™ 是一个采用数据流图（data flow graphs），用于数值计算的开源软件库。节点（Nodes）在图中表示数学操作，图中的线（edges）则表示在节点间相互联系的多维数据数组，即张量（tensor）。它灵活的架构让你可以在多种平台上展开计算，例如台式计算机中的一个或多个CPU（或GPU），服务器，移动设备等等。TensorFlow 最初由Google大脑小组（隶属于Google机器智能研究机构）的研究员和工程师们开发出来，用于机器学习和深度神经网络方面的研究，但这个系统的通用性使其也可广泛用于其他计算领域。（来源：http://www.tensorfly.cn/）

# 软件结构：

数据预处理：

首先将数据从电子表格中读取，进行数据筛选与清洗。之后使用t-sne算法进行数据降维，将数据分成训练集与测试集，以便进行模型检验。

数据的读取函数：

def xlsread(xls):

fname = xls

bk = xlrd.open\_workbook(fname)

shxrange = range(bk.nsheets)

try:

sh = bk.sheet\_by\_name("Sheet1")

except:

print('no sheet in %s named Sheet1' % fname)

nrows = sh.nrows

ncols = sh.ncols

print('(nrows %d, ncols %d)' % (nrows, ncols))

cell\_value = sh.cell\_value(1, 1)

row\_list = []

for i in range(0, nrows):

row\_data = sh.row\_values(i)

row\_list.append(row\_data)

return row\_list

数据分为测试集和训练集，并归一化

X\_train, Y\_train, X\_test, Y\_test = train\_test\_split(X, Y)

scaler = preprocessing.StandardScaler().fit(X\_train)

X\_train\_minmax = scaler.transform(X\_train)

scaler = preprocessing.StandardScaler().fit(X\_test)

X\_test\_minmax = scaler.transform(X\_test)

X\_TRAIN = X\_train\_minmax

Y\_TRAIN = Y\_train.astype(np.float32)

X\_TEST = X\_test\_minmax

Y\_TEST = Y\_test.astype(np.float32)

使用T-SNE降维的函数

def tsne(data\_test, data\_label, title=None, unbalanced=True, method='tsne'):

label\_n = argmax(data\_label, axis=1)

models = {

'tsne': TSNE(n\_iter=5000),

'pca': PCA()}

model = models[method.lower()]

tsne\_transformed = model.fit\_transform(data\_test, label\_n)

plot\_embedding(tsne\_transformed, label\_n, title if title else 't-sne projection', unbalanced)

return tsne\_transformed

## 神经网络结构的建立：

使用TensorFlow的API建立神经网络流图，定义边（tensor）和节点（运算）

### 2.1 AUTOENCODER的建立和训练：

定义自编码网络，包括网络的结构、参数等，网络结构为24-150-8-150-24：

x = tf.placeholder(tf.float32, [None, 24])

y\_ = tf.placeholder(tf.float32, [None, 8])

W1 = tf.Variable(tf.random\_normal([24, 150]))

b1 = tf.Variable(tf.zeros([150]))

y1 = tf.nn.sigmoid(tf.matmul(x, W1) + b1)

W2 = tf.Variable(tf.random\_normal([150, 8]))

b2 = tf.Variable(tf.zeros([8]))

y2 = tf.nn.sigmoid(tf.matmul(y1, W2) + b2)

W3 = W1 = tf.Variable(tf.random\_normal([8, 150]))

b3 = tf.Variable(tf.zeros([150]))

y3 = tf.nn.sigmoid(tf.matmul(y2, W3) + b3)

W4 = tf.Variable(tf.random\_normal([150, 24]))

b4 = tf.Variable(tf.zeros([24]))

y4 = tf.nn.sigmoid(tf.matmul(y3, W4) + b4)

y\_true = x

# Define cost and optimizer, minimize the squared error

cost = tf.reduce\_mean(tf.pow(y\_true - y4, 2))

optimizer = tf.train.RMSPropOptimizer(1e-3).minimize(cost)

# Initializing the variables

init = tf.initialize\_all\_variables()

训练自编码网络,最大迭代周期为1000,学习率为0.001

sess = tf.InteractiveSession()

sess.run(init)

for epoch in range(1000):

# Loop over all batches

for i in range(9):

fd = {x: X\_TRAIN[:(i + 1) \* 100, :]}

\_, c = sess.run([optimizer, cost], feed\_dict=fd)

# Display logs per epoch step

if epoch % 10 == 0:

print("Epoch:", '%04d' % (epoch + 1),

"cost=", "{:.9f}".format(c))

print("AutoEncoder Optimization Finished!")

使用自编码训练的结果建立神经网络分类器模型,网络结构为24-150-8-8

W5 = tf.Variable(tf.random\_normal([8, 8]))

b5 = tf.Variable(tf.zeros([8]))

y\_pred = tf.nn.softmax(tf.matmul(y2, W5) + b5)

cross\_entropy = - \

tf.reduce\_sum(y\_ \* tf.log(tf.clip\_by\_value(y\_pred, 1e-6, 1.0)))

train\_step = tf.train.AdamOptimizer(1e-3).minimize(cross\_entropy)

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

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

# Initializing the variables

init = tf.initialize\_all\_variables()

载入自编码训练得到的网络模型，并使用数据对神经网络分类器调优，最大迭代周期为7000，学习率为0.001

# Launch the graph

saver = tf.train.Saver()

sess = tf.InteractiveSession()

sess.run(init)

accu = 0

ff = [1e+9] \* 2

error = []

precision = []

for step in range(7000):

# Loop over all batches

for j in range(9):

feed\_dict = {x: X\_TRAIN[:(j + 1) \* 100], y\_: Y\_TRAIN[:(j + 1) \* 100]}

\_, f = sess.run([train\_step, cross\_entropy], feed\_dict=feed\_dict)

# Display logs per epoch step

if step % 100 == 0:

print("Step:", '%04d' % (step + 1),

"cross\_entropy=", "{:.9f}".format(f))

#print(sess.run(accuracy, feed\_dict={x: X, y\_: Y}))

g = sess.run(accuracy, feed\_dict={x: X\_TEST, y\_: Y\_TEST})

print("accuracy\_rate=", "%3f" % g)

if g >= accu:

accu = g

saver.save(sess, 'ckpt/epoch\_d%d\_model\_accuracy\_%3f' % (step, g))

print('saved')

ff.append(f)

error.append(f)

precision.append(g)

ff.pop(0)

if (ff[0] - ff[1]) < -10:

break

if f < 10 \* .0000000001:

break

y\_p = sess.run(y\_pred, feed\_dict={x: X\_TEST, y\_: Y\_TEST})

y\_p = np.round(y\_p)

print("Optimization Finished!")

## GRU网络的建立

Tensor的创建与GRU网络的建立方法如下，其中遗忘门偏置为1.0,层数为1,隐藏层单元为100维：

# tf Graph input

x = tf.placeholder(tf.float32, [None, n\_steps, n\_inputs])

y = tf.placeholder(tf.float32, [None, n\_classes])

# Define weights

weights = {

# (24 ,100)

'in': tf.Variable(tf.random\_normal([n\_inputs, n\_hidden\_units])),

# (100 ,8)

'out': tf.Variable(tf.random\_normal([n\_hidden\_units, n\_classes]))

}

biases = {

# (100, )

'in': tf.Variable(tf.constant(0.1, shape=[n\_hidden\_units, ])),

# (8, )

'out': tf.Variable(tf.constant(0.1, shape=[n\_classes, ]))

}

def GRU(X, weights, biases):

# hidden layer for input to cell

# transpose the inputs shape from

# X ==> (100 batch \* 1 steps, 24inputs)

X = tf.reshape(X, [-1, n\_inputs])

# into hidden

# X\_in = (28 batch \* 1 steps, 128 hidden)

X\_in = tf.matmul(X, weights['in']) + biases['in']

# X\_in ==> (20 batch, 1 steps, 128 hidden)

X\_in = tf.reshape(X\_in, [-1, n\_steps, n\_hidden\_units])

# cell

# basic LSTM Cell.

lstm\_cell = tf.nn.rnn\_cell.GRU(n\_hidden\_units, forget\_bias=1.0)

# lstm cell is divided into two parts (c\_state, h\_state)

\_init\_state = lstm\_cell.zero\_state(batch\_size, dtype=tf.float32)

outputs, final\_state = tf.nn.dynamic\_rnn(lstm\_cell, X\_in, initial\_state=\_init\_state, time\_major=False)

# hidden layer for output as the final results

outputs = tf.unpack(tf.transpose(outputs, [1, 0, 2])) # states is the last outputs

results = tf.matmul(outputs[-1], weights['out']) + biases['out']

return results

GRU网络的训练如下，其中学习率= 0.001，最大迭代周期= 50000，训练批量= 100：

pred = GRU(x, weights, biases)

cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(pred, y))

train\_op = tf.train.AdamOptimizer(lr).minimize(cost)

correct\_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))

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

init = tf.initialize\_all\_variables()

with tf.Session() as sess:

sess.run(init)

for step in range(50000):

# Loop over all batches

for j in range(8):

batch\_xs = X\_train[:(j + 1) \* 100]

batch\_xs = batch\_xs.reshape([-1, n\_steps, n\_inputs])

batch\_ys = Y\_train[:(j + 1) \* 100]

feed\_dict = {x: batch\_xs, y: batch\_ys}

\_, f = sess.run([train\_op, cost], feed\_dict=feed\_dict)

if step % 100 == 0:

print("Step:", '%04d' % (step + 1),

"cost=", "{:.9f}".format(f))

# print(sess.run(accuracy, feed\_dict={x: batch\_xs,y: batch\_ys,})

Test\_data = X\_test.reshape((-1, n\_steps, n\_input))

Test\_label = Y\_test

print("Testing Accuracy:", sess.run(accuracy, feed\_dict={x: Test\_data, y: Test\_label}))

根据网络训练的结果，计算误差，绘制误差曲线，对误差以及训练误差进行简要分析，寻找最优网络结构等超参数

# 绘制误差曲线

t = np.arange(100)

fig = plt.figure(num=1, figsize=(10, 10))

ax1 = fig.add\_subplot(1, 2, 1)

l0 = ax1.plot(t, class\_output0, color='red', linewidth=1.0, marker='o')

plt.xlim((-1, 100))

plt.ylim((0, 9))

plt.xlabel('samples data')

plt.ylabel('fault diagnosis classifications')

ax1.set\_title('original curve graph')

ax2 = fig.add\_subplot(1, 2, 2)

l3 = ax2.plot(t, net\_output0, color='b', linewidth=1.0, marker='\*')

plt.xlim((-1, 100))

plt.ylim((0, 9))

plt.xlabel('samples data')

plt.ylabel('fault diagnosis classifications')

ax2.set\_title('recognition curve graph')

fig = plt.figure(num=2, figsize=(10, 10))

plt.plot(t, class\_output0, label='actual fault classification', color='m', linewidth=2.0, marker='\*')

plt.plot(t, net\_output0, label='recognintion fault classification', color='g', linewidth=1.0, marker='o')

plt.xlabel('samples data')

plt.ylabel('fault classification')

plt.xlim((-1, 100))

plt.ylim((0, 9))

plt.title('recognition data classification curve graph')

plt.legend()

# 误差变化曲线

s = np.arange(len(error))

fig = plt.figure(num=3, figsize=(10, 10))

ax4 = fig.add\_subplot(1, 1, 1)

plt.plot(s, error, 'c-\*')

plt.xlim((-1, len(error)))

plt.ylim((100, 3000))

plt.xlabel('steps')

plt.ylabel('cross entropy')

ax4.set\_title('error curve graph')

# 准确率曲线

r = np.arange(len(precision))

fig = plt.figure(num=4, figsize=(10, 10))

ax5 = fig.add\_subplot(1, 1, 1)

plt.plot(r, precision, 'k-o')

plt.xlim((-1, len(precision)))

plt.ylim((0, 1))

plt.xlabel('steps')

plt.ylabel('precision rate')

ax5.set\_title('precision rate curve graph')

plt.show()