綠色作圖/橘色tensorboard/CNN/RNN/autocoder非監督

Import tensorflow as tf

Import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input\_data 引進MNIST數字資料庫

from sklearn.datasets import load\_digits

from sklearn.datasets import train\_test\_split

from sklearn.datasets import LabelBinarizer

1. 取得資料或自行定義資料(X\_data,Y\_data)

矩陣乘法：tf.matmul(matrix1,matrix2)

加法(A+B)：tf.add(A,B)

加載變數(A=B)：tf.assign(A,B)

mnist = input\_data.read\_data\_sets(‘MNIST\_data’,one\_hot=True)

-28x28的方格資料

For sklearn:

digits = load\_digits()

-8x8的資料方格

X = digits.data

Y = digits.target

Y = LabelBinarizer().fit\_transform(Y)

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size = .3)

For MNIST:

Training\_iters = 100000 跌代次數

Batch\_size = 128 分包大小

N\_inputs = 28 (28X28矩陣一行之數量

N\_steps = 28 (28X28矩陣之總共行數

N\_hidden\_units=128 (neurons in hidden layer 可以自己設定

N\_classes = 10 (因0-9十種結果

For MNiST:

Learning\_rate = 0.01

Training\_epochs = 5 訓練批數

Batch\_size = 256

Display\_step = 1 每幾步展示一次

Example\_to\_show = 10 舉10種範例來演示

N\_input = 784 (img shape:28\*28)

N\_hidden\_1 = 256

N\_hidden\_2 = 128

1. 定義參數(Weights,Biases,Xs,Ys)

設定placeholder(形式,size)

-with tf.name\_scope(‘input’):

keep\_prob = tf.placeholder(tf.float32) 設定dropout(保持的概率)

Xs = tf.placeholder(tf.float32,[None,1],name = ‘x\_input’)

Ys = tf.placeholder(tf.float32, [None,1] ,name = ‘y\_input’)

-None=不規定幾個樣本 1=每個樣本的大小，依資料而定

-[None,64] [None,10]

-[None,784] [None,10]

-[None,n\_steps,n\_inputs][None,n\_classes]

-[None,n\_input]

X\_image = tf.reshape(xs,[-1,28,28,1]) [維度不管,長,寬,高]

設定添加層

-def add\_layer(inputs,in\_size,out\_size,n\_layer,activation\_function):

Layer\_name = ‘layer%s’ %n\_layer

with tf.name\_scope(‘Layer\_name’):

with tf.name\_scope(‘Weights’):

Weights = tf.Variable(放入變數型態,name=’W’)

Tf.histogram\_summary(layer\_name+’/weights’,Weights) 記錄用

with tf.name\_scope(‘Biases’):

Biases = tf.Variable(放入變數型態,name=B’)

Tf.histogram\_summary(layer\_name+’/biases’,biases) 記錄用

with tf.name\_scope(‘Wx\_plus\_b’):

Wx\_plus\_b = tf.matmul(input,Weights)+Biases

Wx\_plus\_b = tf.nn.dropout(Wx\_plus\_b,keep\_prob) 設定dropout

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

Tf.histogram\_summary(layer\_name+’/outputs, outputs) 記錄用

Return outputs

第一層(input,l1)

-l1 = add\_layer(Xs,1,10, n\_layer = 1,activation\_function= tf.nn.relu)

第二層(l1,prediction)

-prediction = add\_layer(l1,10,1, n\_layer = 2,activation\_function= None)

CNN定義變數

-def weight\_variable(shape):

initial = tf.truncated\_normal(shape,stddev = 0.1)

return tr.Variable(initial)

-def bias\_variable(shape):

initial = tf.canstant(0.1,shape =shape)

return tf.variable(initial)

-def conv2d(x,W):

Return tf.nn.conv2d(x,W, strides = [1,1,1,1],padding = ‘SAME’)

-strides為跨度，[]第1,4位要為1，2,3位為x,y軸跨度

-paddind為掃描的廣度

-def max\_pool\_2x2(x):

Return tf.nn.max\_pool(x,ksize = [1,2,2,1],strides=[1,2,2,1],padding=’SAME’)

Conv1

W\_conv1 = weight\_variable([5,5,1,32]) [patch長, patch寬, in size, out size]

B\_conv1 = bias\_variable([32])

H\_conv1 = tf.nn.relu(Conv2d(x\_image,W\_conv1)+b\_conv1)

- relu先進行非線性化

-output 28x28x32

H\_pool1 = max\_pool\_2x2(H\_conv1)

-output 14x14x32

Conv2

W\_conv2 = weight\_variable([5,5,32,64]) [patch長, patch寬, in size, out size]

B\_conv2 = bias\_variable([32])

H\_conv2 = tf.nn.relu(Conv2d(x\_image,W\_conv1)+b\_conv1) -output 14x14x64

H\_pool2 = max\_pool\_2x2(H\_conv2) -output 7x7x64

Func1

W\_fc1 = weight\_variable([7\*7\*64,1024])

B\_fc1 = bias\_variable([1024])

H\_pool2\_flat = tf.reshape(h\_pool2,[-1,7\*7\*64])

-[n\_samples,7,7,64]>>>[n\_sample,7\*7\*64]

H\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

H\_fc1\_drop = tf.nn.dropout(h\_fc1,keep\_prob)

Func2

W\_fc2 = weight\_variable([1024,10])

B\_fc2 = bias\_variable([10])

prediction = tf.nn.softmax(tf.matmul(H\_fc1\_drop, W\_fc2) + b\_fc2)

RNN定義變數

Weights={

‘in’:tf.Variable(tf.random\_normal([n\_inputs,n\_hidden\_units])), 28>>128

‘out’: tf.Variable(tf.random\_normal([n\_hidden\_units,n\_classes])) 128>>10

}

Biases={

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

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

}

Def RNN(X,weights,biases):

決定input

X = tf.reshpae(X,[-1,n\_inputs]) X(128batch,28steps,28uinputs)>>(128\*28,28inputs)

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

X\_in = tf.reshape(X\_in,[-1,n\_steps,n\_hidden\_units]) ->>(128batch,28steps,28uinputs)

##############################################################

決定cell

Lstm\_cell=tf.nn.rnn\_cell.BasicalLSTMCell(n\_hidden\_units,

forget\_bias=1.0, 剛開始不要被忘掉

state\_is\_tuple=True)

- lstm cell is divided into two parts (c\_state,m\_state)-主線和分線

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

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

-若time\_state(這邊是n\_steps)不是在第一個time\_major就是Fasle

##############################################################

決定output

Results=tf.matmul(state[1],weights[‘out’])+biases[‘out’]

Return results

Autocoder定義變數

Weights = {

‘encoder\_h1’:tf.Variable(tf.random\_normal([n\_input,n\_hidden\_1])),

‘encoder\_h2’:tf.Variable(tf.random\_normal([n\_hidden\_1,n\_hidden\_2])),

‘decoder\_h1’:tf.Variable(tf.random\_normal([n\_hidden\_2,n\_hidden\_1])),

‘decoder\_h2’:tf.Variable(tf.random\_normal([n\_hidden\_1,n\_input])),

}

Biases = {

‘encoder\_b1’:tf.Variable(tf.random\_normal([n\_hidden\_1])),

‘encoder\_b2’:tf.Variable(tf.random\_normal([n\_hidden\_2])),

‘decoder\_b1’:tf.Variable(tf.random\_normal([n\_hidden\_1])),

‘decoder\_b2’:tf.Variable(tf.random\_normal([n\_input])),

}

Def encoder(x): -tf.nn.sigmoid的輸出值介於1~0

Layer\_1 = tf.nn.sigmoid(tf.add(tf.matmul(x,weight[‘encoder\_h1’]),biases[‘encoder\_b1’]))

Layer\_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer\_1,weight[‘encoder\_h2’]),

biases[‘encoder\_b2’]))

return layer\_2

Def decoder(x):

Layer\_1 = tf.nn.sigmoid(tf.add(tf.matmul(x,weight[‘decoder\_h1’]),biases[‘decoder\_b1’]))

Layer\_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer\_1,weight[‘decoder\_h2’]),

biases[‘decoder\_b2’]))

return layer\_2

變數型態：

\*隨機變數：tf.random\_normal([in\_size,out\_size])

隨機數列([維度],從-1,到1)：tf.random\_uniform([1],-1.0,1.0)

\*零矩陣([維度])：tf.zeros([1,out\_size]+0.1)

激勵函數(activation\_function)

-google：tensorflow activation

-最常用：tf.nn.relu

-classification：tf.nn.softmax

-如果要避免NONE：tf.nn.tanh

Classification定義準確度函式

-def compute\_accuracy(v\_xs,v\_ys):

global prediction

y\_pre = sess.run(prediction,feed\_dict={xs:v\_xs})

correct\_prediction = tf.equal(tf.argmax(y\_pre,1),tf.argmax(v\_ys,1))

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

result = sess.run(accuracy,feed\_dict={xs:v\_xs,ys:v\_ys})

return result

三、定義訓練方法

定義loss的值

-with tf.name\_scope(‘loss’):

loss = tf.reduce\_mean(tf.reduce\_sum(tf.square(Ys-prediction),reduction\_indices=[1]))

(實際-預測)平方後 求和(要定義哪一軸總和)後 再平均

Tf.scalar\_summary(‘loss’,loss) 記錄用，出現在event

定義cross\_entropy的值

-用於classification，用法 = loss

-cross\_entropy= tf.reduce\_mean(-tf.reduce\_sum(Ys\*tf.log(prediction),reduction\_indices=[1]))

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

定義Autoencoder的cost值

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

-y\_true = X

-y\_pred = decoder(encoder(X))

定義優化器減少誤差之方法(學習效率，通常小於1)

- with tf.name\_scope(‘train’):

optimizer = tf.train.GradientDescentOptiomizer(0.5).minimizer(loss)

-共七種，參考加速神經網絡訓練

-CNN建議使用tf.train.AdamOptimizer(1e-4) 0.0001

-RNN 建議使用tf.train.AdamOptimizer(1e-4) 0.0001

-Autoencoder 建議使用tf.train.AdamOptimizer(1e-2) 0.01

四、初始化結構(很重要)

Init = tf.global\_variables\_initializer()

五、Session操作控制

開始操作定義：With tf.Session() as sess:

操作初始化：Sess.run(init)

操作訓練

-for I in range(步數):

batch\_xs, batch\_ys = mnist.train.next\_batch(batch\_size) 每一次都提取小部份訓練(SGD)

sess.run(train,feed\_dic{Xs:X\_data,Ys:Y\_data})

-feed\_dic{}可替換成batch\_xs, batch\_ys

-或再加上keep\_prob:0.4 設定dropout(保持0.4的資料)

-step = 0

While step\*batch\_size<training\_iters:

batch\_xs, batch\_ys = mnist.train.next\_batch(batch\_size)

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

sess.run(train,feed\_dic{Xs:X\_data,Ys:Y\_data})

step +=1

-total\_batch = int(mnist.train.num\_examples/batch\_size)

For epoch in range(training\_epochs):

For I in range(total\_batch):

Batch\_xs,batch\_ys = mnist.train.next\_batch(batch\_size) max(x)=1 min(x)=0

If epoch % display\_step = 0:

Print(‘epoch:’,’%04d’ % (epoch+1), ‘cost=’,’{:.9f}’.format(c))

操作變數：Sess.run(變數, ,feed\_dic{Xs: batch\_xs,Ys: batch\_ys })

六、可視化

fig = plt.figure

ax = fig.add\_subplot(1,1,1) 定義連續性的作圖(編號)

ax.scatter(x\_data,y\_data) 點的形式作圖

plt.ion() 不暫停

plt.show()

連續動作：(放入操作訓練的迴圈中)

try:

ax.lines.remove(lines[0]) 去除上一個線段

except Exception:

pass

lines = ax.plot(x\_data,prediction\_value,’r-’,lw = 5) 畫線圖(x座標,y座標,線段顏色,線的粗度)

plt.pause(0.1) 暫停0.1秒

七、神經網絡可視化

Writer = tf.summary.FileWriter(‘logs/’,sess,graph)

使用CMD，轉到放置logs資料夾的頁面，輸入 tensorboard --logdir=’logs/’

Merged = tf.merge\_all\_summarier() 打包記錄用可視化函式

連續動作：(放入操作訓練的迴圈中)

Result = sess.run(merged, feed\_dic{Xs:X\_data,Ys:Y\_data})

- feed\_dic{}可替換成batch\_xs, batch\_ys

-可加上keep\_prob:1 設定dropout(為保留全部結果顯示，必要是1)

Writer.add\_summary(result,i)

八、儲存和讀取變數

儲存

Tf.Variable(變數型態,dtype=tf.float32,name=’名稱’)

Saver = tf.train.Saver()

save\_path = saver.save(sess,’資料夾/檔案名稱.ckpt’)

-在sess.run(init)之後加上

讀取

Tf.Variable(變數型態,dtype = tf.float32,name = ‘名稱’)

Saver = tf.train.Saver()

Saver.restore(sess, ’資料夾/檔案名稱.ckpt’)

九、scope命名方式

with tf.name\_scope(‘train’):

var2 = tf.Variable(name='var2', initial\_value=[2], dtype=tf.float32)

-所有名為var2的變數會被區分成var2、var2\_1、var2\_2等等

with tf.variable\_scope("a\_variable\_scope") as scope:

initializer = tf.constant\_initializer(value=3)

var3 = tf.get\_variable(name='var3', shape=[1], dtype=tf.float32, initializer=initializer)

scope.reuse\_variables() -宣告變數名稱一樣的將被取代

var3\_reuse = tf.get\_variable(name='var3',)

-所有名為var3的變數會被後一個取代

def \_weight\_variable(shape, name='weights'):

initializer = tf.random\_normal\_initializer(mean=0., stddev=0.5, )

return tf.get\_variable(shape=shape, initializer=initializer, name=name)

十、Batch normalization批標準化 (需要再看清楚)

使用時機：標準化初始資料、標準化每層資料(通常加在add\_layer裡，在定義完Wx\_plus\_b後)

Fc\_mean,fc\_var = tf.nn.moments(Wx\_plus\_b,axes = [0])

-取得Wx\_plus\_b之資料平均和資料標準差

-如果是要求初始資料標準化，把Wx\_plus\_b改成Xs即可

定義scale,shift,epsilon,ema

Scale = tf.Variable(tf.ones([out\_size]))

Shift = tf.Variable(tf.zeros([out\_size]))

Epsilon = 0.001

Ema = tf.train.ExponentialMovingAverage(decay = 0.5)

Def mean\_var\_with\_update():

Ema\_apply\_op = ema.apply([fc\_mean,fc\_var])

With tf.control\_dependencies([ema\_apply\_op]):

Return tf.identity(fc\_mean),tf.identity(fc\_var)

Mean,var = mean\_var\_with\_update()

Wx\_plus\_b = tf.nn.batch\_normalization(Wx\_plus\_b,mean,var,shift,scale,epsilon)

-等同：

Wx\_plus\_b = (Wx\_plus\_b-fc\_mean)/tf.sqrt(fc\_var+0.001)

Wx\_plus\_b = Wx\_plus\_b\*scale+shift