**什麼是Tensorflow?**

TensorFlow 是 Google透過使用資料流 (flow) 圖像，來進行數值演算的新一代開源機器學習工具。

**Tensorflow安裝**

MacOS/Linux安裝

用pip安裝(需先確認電腦中已安裝了pip。若電腦中已安裝了python3.X，因為pip已經自帶在python的模組裡，因此pip也已安裝了)

CPU版本

開啟terminal

#如果是安裝python2.X

$pip install tensorflow

#如果是安裝python3.X

$pip3 install tensorflow

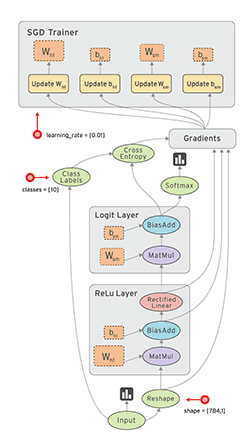
Windows安裝

支持python3.5(64bit)版本

開啟command

c:\>pip install tensorflow

**Tensorflow數據流圖**

****

**實例**

import tensorflow as tf

import numpy as np

#create data

x\_data=np.random.rand(100).astype(np.float32) #生成100個隨機數列，在tensorflow中大部分的數據的type是float32的形式

y\_data=x\_data\*0.1+0.3

#create tensorflow structure start

Weights=tf.Variable(tf.random\_uniform([1],-1.0,1.0)) #tf.random\_uniform(結構,左範圍,右範圍)，初始值是-1~1的數

biases=tf.Variable(tf.zeros([1])) #初始值是0

y=Weights\*x\_data+biases

loss=tf.reduce\_mean(tf.square(y-y\_data)) #誤差，預測的y和實際的y\_data的差別

optimizer=tf.train.GradientDescentOptimizer(0.5) #利用optimizer減少誤差，GradientDescentOptimizer(學習效率)，學習效率<1

train=optimizer.minimize(loss)

init=tf.initiallize\_all\_variables() #初始化變量

#create tensorflow structure end

sess=tf.Session()

sess.run(init) #very important

for step in range(201):

session.run(train)

if step%20==0:

print(step,sess.run(Weights),sess.run(biases))

\*Session

Tensorflow是基於圖架構進行運算的深度學習框架，Session是圖和執行者之間的媒介，首先透過Session來啟動圖，而Session.run()是用來進行操作的，Session再使用完過後需要透過close來釋放資源，或是透過with as的方式來讓他自動釋放。

import tensorflow as tf

matrix1=tf.constant([[3,3]]) #Constant就是不可變的常數

matrix2=tf.constant([[2],[2]])

product=tf.matmul(matrix1,matrix2)

#method 1

session=tf.Session()

result=sess.run(product)

print(result)

sess.close()

#method 2

with tf.Session() as sess:

result2= sess.run(product)

print(result2)

\*Variable

將值宣告賦值給變數（Variables）讓使用者能夠動態地進行相同的計算來得到不同的結果，在TensorFlow中是以tf.Variable()來完成。

在TensorFlow的觀念之中，宣告變數張量並不如Python那麼單純，它需要兩個步驟：

1.宣告變數張量的初始值、類型與外觀 2.初始化變數張量

import tensorflow as tf

state=tf.Variable(0,name='counter')

#print(state.name) #print出來的結果為counter:0

one=tf.constant(1)

new\_value=tf.add(state,one)

update=tf.assign(state,new\_value) #可以透過tf.assign()賦予不同的值，值得注意的地方是對變數張量重新賦值這件事對tensorflow來說也算是一個運算，必須在宣告之後放入Session中執行，否則重新賦值並不會有作用。

init=tf.initialize\_all\_variables() #must have if define variable

with tf.Session() as sess:

sess.run(init)

for \_ in range(3):

sess.run(update)

print(sess.run(state))

\*Placeholder

我們可以將它想成是一個佔有長度卻沒有初始值的None，差異在於None不需要將資料類型事先定義，但是Placeholder必須事先定義好之後要輸入的資料類型與外觀。

import tensorflow as tf

input1=tf.placeholder(tf.float32) #先定義好之後要輸入的資料類型，tf.placeholder(dtype,shape=None,name=None)

input2=tf.placeholder(tf.float32)

output=tf.mul(input1,input2)

with tf.Session() as sess:

print(sess.run(output,feed\_dict={input1:[7.],input2:[2.]})) #將資料以python dict餵進(feed)Placeholder之中，print出來的結果為[14.]

\*激勵函數Activation Function

在類神經網路中使用激勵函數，主要是利用非線性方程式，解決非線性問題，若不使用激勵函數，類神經網路即是以線性的方式組合運算，因為隱藏層以及輸出層皆是將上層之結果輸入，並以線性組合計算，作為這一層的輸出，使得輸出與輸入只存在著線性關係，而現實中，所有問題皆屬於非線性問題，因此，若無使用非線性之激勵函數，則類神經網路訓練出之模型便失去意義。

1.激勵函數需選擇可微分之函數，因為在誤差反向傳遞(Back Propagation)運算時，需要進行一次微分計算。

2.在深度學習中，當隱藏層之層數過多時，激勵函數不可隨意選擇，因為會造成梯度消失(Vanishing Gradient)以及梯度爆炸(Exploding gradients)等問題。

常見的激勵函數的選擇有sigmoid，tanh，ReLU，實用上最常使用ReLU。

\*例子-def add\_layer()

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

def add\_layer(inputs,in\_size,out\_size,activation\_function=None): #add\_layer(輸入值,輸入的大小,輸出的大小,激勵函數)

Weights=tf.Variable(tf.random\_normal([in\_size,out\_size])) #在生成初始參數時，隨機變量(normal distribution)會比全部為0要好很多，所以這裡的Weights為一個in\_size行，out\_size列的隨機變量矩陣。

biases=tf.Variable(tf.zeros([1,out\_size])+0.1) #在機器學習中，biases推薦的初始值不為零，所以+0.1

Wx\_plus\_b=tf.matmul(inputs,Weights)+biases

if activation\_function is None: #當activation\_function為None時(非線性函數)，輸出就是當前的預測值Wx\_plus\_b，不為None時，就會把Wx\_plus\_b傳到activation\_function()函數中得到輸出。

outputs=Wx\_plus\_b

else:

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

return outputs

\*建造神經網路+結果可視化

#Make up some real data

x\_data=np.linspace(-1,1,300)[:,np.newaxis] #定義輸入資料，為一個值域在-1到+1之間的300個數據，為300\*1的形式。numpy.linspace(start, stop, num=50)在指定區間內返回均勻間隔的數字，np.newaxis的功能是增加一個新的維度，ex:[1 2 3]透過[:,np.newaxis]->[[1][2][3]]，[1 2 3]透過[np.newaxis,:]->[[1 2 3]]

noise=np.random.normal(0,0.05,x\_data.shape) #然後定義一個與x\_data形式一樣的噪音點，使得我們進行訓練的資料更像是真實的資料

y\_data=np.squre(x\_data)-0.5+noise #定義y\_data,假設為x\_data的平方減去噪音點

#define placeholder for inputs to network

xs=tf.placeholder(tf.float32,[None,1]) #定義placeholder, 引數中none表示輸入多少個樣本都可以，x\_data的屬性為1，所以輸出也為1

ys=tf.placeholder(tf.float32,[None,1])

l1=add\_layer(x\_data,1,10,activation\_function=tf.nn.relu) #我們定義一個簡單的神經網路，輸入層->隱藏層->輸出層，隱藏層我們假設只給10個神經元，輸入層是有多少個屬性就有多少個神經元，我們的x\_data只有一個屬性，所以只有一個神經元，輸出層與輸入層是一樣的，輸入層有多少個神經元輸出層就有多少個。

prediction=add\_layer(l1,10,1,activation\_function=None) #l1=add\_layer(x\_data,輸入層,隱藏層,activation\_function=tf.nn.relu)，prediction=add\_layer(l1,隱藏層,輸出層,activation\_function=None)

loss=tf.reduce\_mean(tf.reduce\_sum(tf.square(y\_data-prediction),reduction\_indices=[1])) #計算預測值與真實值的差異

train\_step=tf.train.GradientDescentOptimizer(0.1).minimize(loss)

init=tf.initialize\_all\_variables()

sess=tf.Session()

sess.run(init)

fig=plt.figure() #figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True)，figure(圖的名稱,圖的大小ex=(4,3),參數指定繪圖對象的分辨率(即每英吋有多少個像素),背景顏色,邊框顏色,是否顯示邊框)

ax=fig.add\_subplot(1,1,1) #add\_subplot(1,1,1)表示1x1的網格，第1個子圖，add\_subplot(A,B,C)表示AxB的網格，第C個子圖。

ax.scatter(x\_data,y\_data) #ax.scatter(x, y, z, c = 'r', marker = '^')表示產生散點圖，c表示顏色，marker表示點的形式(o是圓形的點，^是三角形)

plt.ion() #開啟交互模式->連續顯示圖，plt.ioff()->關閉交互模式

plt.show()

for i in range(1000):

sess.run(train\_step,feed\_dict={xs:x\_data,ys:y\_data})

if i%50==0:

print(sess.run(loss,feed\_dict={xs:x\_data,ys:y\_data}))

try:

ax.lines.remove(lines[0])

except Exception:

pass

prediction\_value=sess.run(prediction,feed\_dict={xs:x\_data})

lines=ax.plot(x\_data,prediction\_value,'r-',lw=5) #紅色，寬度為5

plt.pause(0.1) #暫停0.1s

\*神經網路學習的優化(speed up training)

梯度下降法(gradient descent，GD)

梯度下降法是一種不斷去更新參數找「解」的方法，所以一定要先隨機產生一組初始參數的「解」，然後根據這組隨機產生的「解」開始算此「解」的梯度方向大小，然後將這個「解」去減去梯度方向，公式如下: (t是第幾次更新參數，γ是學習率(Learning rate)，一次要更新多少，就是由學習率來控制的)

參考:https://medium.com/@chih.sheng.huang821/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E5%9F%BA%E7%A4%8E%E6%95%B8%E5%AD%B8-%E4%BA%8C-%E6%A2%AF%E5%BA%A6%E4%B8%8B%E9%99%8D%E6%B3%95-gradient-descent-406e1fd001f

http://ruder.io/optimizing-gradient-descent/index.html#adam

隨機梯度下降法(stochastic Gradient Descent,SGD)

在更新參數的時候，GD我們是一次用全部訓練集的數據去計算損失函數的梯度就更新一次參數。SGD就是一次跑一個樣本或是小批次(mini-batch)樣本然後算出一次梯度或是小批次梯度的平均後就更新一次，那這個樣本或是小批次的樣本是隨機抽取的，所以才會稱為隨機梯度下降法。

SGD缺點:在當下的問題如果學習率太大，容易造成參數更新呈現鋸齒狀的更新，這是很沒有效率的路徑。

Momentum

公式: (t是第幾次更新參數，γ是學習率(Learning rate)，m是momentum項(一般設定為0.9))，主要是用在計算參數更新方向前會考慮前一次參數更新的方向(v(t-1))，如果當下梯度方向和歷史參數更新的方向一致，則會增強這個方向的梯度，若當下梯度方向和歷史參數更新的方向不一致，則梯度會衰退。然後每一次對梯度作方向微調。這樣可以增加學習上的穩定性(梯度不更新太快)，這樣可以學習的更快，並且有擺脫局部最佳解的能力。

AdaGrad

SGD和momentum在更新參數時，都是用同一個學習率(γ)，Adagrad算法則是在學習過程中對學習率不斷的調整，這種技巧叫做「學習率衰減(Learning rate decay)」。通常在神經網路學習，一開始會用大的學習率，接著在變小的學習率。大的學習率可以較快走到最佳值或是跳出局部極值，但越後面到要找到極值就需要小的學習率。Adagrad則是針對每個參數客制化的值，這邊假設 g\_t,i為第t次第i個參數的梯度，(ε是平滑項，主要避免分母為0的問題，一般設定為1e-7。Gt這邊定義是一個對角矩陣，對角線每一個元素是相對應每一個參數梯度的平方和。)

Adagrad缺點是在訓練中後段時，有可能因為分母累積越來越大(因為是從第1次梯度到第t次梯度的和)導致梯度趨近於0，如果有設定early stop的，會使得訓練提前結束。early stop:在訓練中計算模型的表現開始下降的時候就會停止訓練。

RMSProp

RMSProp和Adagrad一樣是自適應的方法，但Adagrad的分母是從第1次梯度到第t次梯度的和，所以和可能過大，而RMSprop則是算對應的平均值，因此可以緩解Adagrad學習率下降過快的問題。

公式:E[]在統計上就是取期望值，所以是取g\_i^2的期望值，白話說就是他的平均數。ρ是過去t-1時間的梯度平均數的權重，一般建議設成0.9。

Adam

Momentum是「計算參數更新方向前會考慮前一次參數更新的方向」， RMSprop則是「在學習率上依據梯度的大小對學習率進行加強或是衰減」。Adam則是兩者合併加強版本(Momentum+RMSprop+各自做偏差的修正)。

mt和vt分別是梯度的一階動差函數和二階動差函數(非去中心化)。因為mt和vt初始設定是全為0的向量，Adam的作者發現算法偏量很容易區近於0，因此他們提出修正項，去消除這些偏量

Adam更新的準則: (adam2)(建議預設值β1=0.9, β2=0.999, ε=10^(-8)。)

\*優化器optimizer

參考:https://www.tensorflow.org/api\_docs/python/tf/train

https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/518746/

\*可視化

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

\* def add\_layer(inputs,in\_size,out\_size,n\_layer,activation\_function=None):

\* layer\_name='layer%s'%n\_layer

\* with tf.name\_scope(layer\_name):

\* with tf.name\_scope('weight'):

\* Weights=tf.Variable(tf.random\_normal([in\_size,out\_size]),name='W')

\* tf.histogram\_summary(layer\_name+'/weights',Weights)

\* with tf.name\_scope('biases'):

\* biases=tf.Variable(tf.zeros([1,out\_size])+0.1,name='b')

\* tf.histogram\_summary(layer\_name+'/biases',biases)

\* with tf.name\_scope('Wx\_plus\_b'):

Wx\_plus\_b=tf.matmul(inputs,Weights)+biases

if activation\_function is None:

outputs=Wx\_plus\_b

else:

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

\* tf.histogram\_summary(layer\_name+'/outputs',outputs)

return outputs

#Make up some real data

x\_data=np.linspace(-1,1,300)[:,np.newaxis]

noise=np.random.normal(0,0.05,x\_data.shape)

y\_data=np.squre(x\_data)-0.5+noise

#define placeholder for inputs to network

\* with tf.name\_scope('inputs'):

\* xs=tf.placeholder(tf.float32,[None,1],name='x\_input')

\* ys=tf.placeholder(tf.float32,[None,1],name='x\_input')

\* l1=add\_layer(x\_data,1,10,n\_layer=1,activation\_function=tf.nn.relu)

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

\* with tf.name\_scope('loss'):

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

\* tf.scalar\_summary('loss',loss)

\* with tf.name\_scope('train'):

train\_step=tf.train.GradientDescentOptimizer(0.1).minimize(loss)

init=tf.initialize\_all\_variables()

sess=tf.Session()

\* merged=tf.merge\_all\_summaries()

\* writer=tf.train.SummaryWriter("logs/",sess.graph)

sess.run(init)

fig=plt.figure()

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

ax.scatter(x\_data,y\_data)

plt.ion()

plt.show()

for i in range(1000):

sess.run(train\_step,feed\_dict={xs:x\_data,ys:y\_data})

if i%50==0:

\* result=sess.run(merged,feed\_dict={xs:x\_data,ys:y\_data}))

\* writer.add\_summary(result,i)

存檔後開啟terminal，移動到檔案目錄後輸入tensorboard --logdir='logs/'

複製網址，在網址列上貼上搜尋