# TensorFlow学习

## TensorFlow的执行时的基本对象及其具体理解：

分为Tensor、Variable和operation三大类，他们共同构成执行时的graph，执行可以执行graph中的一个子节点，子节点依照DAG顺序执行期上游全部父节点。

示例：下面讲解概念时有具体示例

## 常量操作和变量、另外也理解一下执行时的基本对象

**import**tensorflow**as**tf

m1 **=**tf.constant([[3, 3]]) *# 常量operation，生成一个Tensor*

m2 **=**tf.constant([[1], [2]]) *# 常量operation，生成一个Tensor*

product **=**tf.matmul(m1, m2) *# 加法operation，输入两个Tensor，生成一个Tensor*

​

**with**tf.Session() **as** session: *# 利用Session执行上述代码中生成的DAG*

res**=**session.run(product)

print(res)

[[9]]

In [2]:

**import**tensorflow**as**tf

state **=**tf.Variable([0, 0], "counter") *# 变量*

new\_value**=**tf.add(state, [1, 2]) *# 加法operation 生成一个Tensor*

upstate **=**tf.assign(state, new\_value) *# 赋值operation 生成一个Tensor*

​

init**=**tf.global\_variables\_initializer() *# 变量初始化*

​

**with**tf.Session() **as** session:

session.run(init) *# 变量初始化*

**for** \_ **in**range(10):

session.run(upstate) *# 执行更新赋值也就是“+=”10次*

print(session.run(state)) *# 查看每次执行的效果*

[1 2]

[2 4]

[3 6]

[4 8]

[ 5 10]

[ 6 12]

[ 7 14]

[ 8 16]

[ 9 18]

[10 20]

## Feed和fetch

*# fetch*

**import**tensorflow**as**tf

m1 **=**tf.constant([[3, 3]]) *# 常量operation，生成一个Tensor*

m2 **=**tf.constant([[1], [2]]) *# 常量operation，生成一个Tensor*

m3 **=**tf.constant([3])

product **=**tf.matmul(m1, m2) *# 加法operation，输入两个Tensor，生成一个Tensor*

add**=**tf.add(product, m3)

​

**with**tf.Session() **as** session: *# 利用Session执行上述代码中生成的DAG*

res **=**session.run([product, add]) *# 同一个run中运行多个operation，operation之间可以有依赖关系*

print(res)

[array([[9]]), array([[12]])]

[array([[9]]), array([[12]])]

*# feed*

**import**tensorflow**as**tf

input1 **=**tf.placeholder(tf.float32)

input2 **=**tf.placeholder(tf.float32)

add**=**tf.add(input1, input2)

​

**with**tf.Session() **as** session: *# 利用Session执行上述代码中生成的DAG*

res **=**session.run(add, feed\_dict**=** {input1: [32.1], input2: 78.2}) *# 同一个run中运行多个operation，operation之间可以有依赖关系*

print(res)

[110.299995]

## Tensorflow进行机器学习的一个简单案例——梯度下降求解线性回归模型

*# 构造一个线性回归的数据集*

**import**numpy**as**np

x\_data**=**np.random.rand(1000)

y\_data**=**x\_data**\***2.0**+**1.5

​

*# 利用tensorflow中的梯度下降法训练该线性回归*

**import**tensorflow**as**tf

​

*# 定义线性回归的两个变量：系数和常数项*

k **=**tf.Variable(0.0)

b **=**tf.Variable(0.0)

y **=**x\_data**\*** k **+** b

​

*# 梯度下降需要：学习率、损失函数 --梯度下降会自动根据损失函数求导，这一点很方便*

loss**=**tf.reduce\_mean(tf.square(y\_data**-** y))

optimizer**=**tf.train.GradientDescentOptimizer(0.2)

train**=**optimizer.minimize(loss)

​

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

​

*# 运行梯度下降4000次并打印每次的结果和对应的损失函数*

**with**tf.Session() **as** session:

session.run(init)

**for** step **in**range(4000):

session.run(train)

print(session.run([k, b, loss]))

运行结果：

[0.5802862, 1.0091515, 1.6426241]

[0.87479293, 1.4959297, 0.4374238]

[1.0294542, 1.7277478, 0.14771365]

[1.1155472, 1.8351988, 0.076466516]

[1.1678884, 1.8820568, 0.057430845]

[1.203488, 1.8994639, 0.0509492]

………

[1.9999962, 1.5000021, 1.2278321e-12]

[1.9999963, 1.500002, 1.1365984e-12]

[1.9999964, 1.500002, 1.0793286e-12]

[1.9999965, 1.5000019, 9.928555e-13]

[1.9999967, 1.5000019, 9.46713e-13]

[1.9999968, 1.5000018, 8.66649e-13]

[1.9999969, 1.5000018, 8.3117866e-13]

[1.9999969, 1.5000017, 7.964047e-13]

[1.999997, 1.5000017, 7.523795e-13]

[1.9999971, 1.5000017, 7.226504e-13]

[1.9999971, 1.5000015, 6.7916517e-13]

[1.9999973, 1.5000015, 6.461818e-13]

[1.9999973, 1.5000015, 6.461818e-13]

[1.9999973, 1.5000015, 6.461818e-13]

[1.9999973, 1.5000015, 6.461818e-13]

[1.9999973, 1.5000015, 6.461818e-13]

最后稳定在6.461818e-13处

## 二层神经网络进行回归训练

**import** tensorflow **as** tf

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

​

*# 创建数据：二次曲线的分布 + 噪声*

*# ----*

*# 创建从-0.5到0.5的等差数列，长度为200*

line\_data **=** np.linspace(**-**0.5, 0.5, 200)

*# 为创建的数组再增加一个轴*

x\_data **=** line\_data[:, np.newaxis]

noise **=** np.random.normal(0, 0.02, x\_data.shape)

​

y\_data **=** np.square(x\_data) **+** noise

​

*# 看下数据的类型和shape 注意python中float位数变换会导致数组变长或变短，需要astype*

print(noise.shape, noise.dtype)

print(x\_data.shape)

print(y\_data.shape, y\_data.dtype)

​

*# 定义两个placeholder用来feed数据，分别代表x和y --1列*

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

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

​

*# ----*

*# 构建神经网络进行训练*

*# 假定：*

*# --一个隐含层L1*

*# --十个神经元*

*# ----*

​

*# 定义第一层的weight和bias, 都是1\*10的矩阵, 初始化分别为正态随机和0.0*

weight\_1 **=** tf.Variable(tf.random\_normal([1, 10]))

bias\_1 **=** tf.Variable(tf.zeros([1, 10]))

a\_L1 **=** tf.matmul(x, weight\_1) **+** bias\_1

z\_L1 **=** tf.nn.tanh(a\_L1) *# 这里用tanh更好一些*

​

*# 定义第二层的weight和bias，10\*1的矩阵和1\*1的矩阵，初始化分别为正态随机和0.0*

weight\_2 **=** tf.Variable(tf.random\_normal([10, 1]))

bias\_2 **=** tf.Variable(tf.zeros([1, 1]))

a\_L2 **=** tf.matmul(z\_L1, weight\_2) **+** bias\_2

y\_head **=** tf.nn.tanh(a\_L2) *# 这里也是用tanh更好一些*

*# 定义二次损失函数并依据梯度下降法进行训练 -- 这样梯度下降的train就变成了x和y的函数*

loss **=** tf.reduce\_mean(tf.square(y **-** y\_head))

optimizer **=** tf.train.GradientDescentOptimizer(0.1)

train **=** optimizer.minimize(loss)

​

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

​

*# 运行梯度下降4000次并打印每次的结果和对应的损失函数*

**with** tf.Session() **as** session:

session.run(init)

i **=** 0

**for** step **in** range(2000):

session.run(train, feed\_dict**=** {x: x\_data, y: y\_data}) *# 此处是最小化*

ls **=** session.run(loss, feed\_dict**=** {x: x\_data, y: y\_data})

*# 注意这里不是ls = session.run(loss), 因为loss也是DAG, 也是x和y的函数，因此同样需要feed，*

*#tf中常量和运行结果可以直接打印，其他都要run后打印*

**if**(i **%** 20 **==** 0):

print(step, ls) *# 每隔二十次打印损失，观察收敛情况*

i **+=** 1

*# 训练的神经网络模型中的两层的w和b*

print("weight\_1", session.run(weight\_1))

print("bias\_1", session.run(bias\_1))

print("weight\_2", session.run(weight\_2))

print("bias\_2", session.run(bias\_2))

*# 预测结果*

predict **=** session.run(y\_head, feed\_dict**=** {x: x\_data}) *# 这里y\_head即预测结果，注意同样是一个DAG，是x的函数，因此需要feed*

​

*# 利用matplot展示执行效果*

plt.figure()

plt.scatter(x\_data, y\_data) *# 训练数据用散点图展示*

plt.plot(x\_data, predict, 'r-', lw **=** 3) *# 预测值用曲线展示，r表示红色，-表示实现，lw为曲线粗细*

plt.show()

​

​

(200, 1) float64

(200, 1)

(200, 1) float64

0 0.3595666

20 0.0062810197

40 0.005968654

60 0.0057035936

80 0.005450571

100 0.0052087354

120 0.0049773413

140 0.00475574

## 多分类逻辑回归进行手写数字识别

*# 数据集介绍*

*# MNIST数据集，100k的训练数据，10k的预测数据，数据由tensorflow中的examples.tutorials.mnist读取*

*# 数据集介绍：：Yann LeCun's website*

*# 由28\*28的像素组成输入特征，输出特征为0-9的数字*

​

*# 可调节参数：*

*# --------*

*# batch\_size, initial\_weight,二次损失函数,learning\_rate,epoch\_n*

*# --------*

​

**import** tensorflow **as** tf

**from** tensorflow.examples.tutorials.mnist **import** input\_data

​

mnist **=** input\_data.read\_data\_sets("MNIST\_data", one\_hot **=** **True**)

​

*# mini\_batch的大小*

batch\_size **=** 100

batch\_n **=** mnist.train.num\_examples **//** batch\_size

​

*# # 定义两个placeholder用来feed数据，分别代表x和y --784列和10列(one-hot)*

x **=** tf.placeholder(tf.float32, [**None**, 784])

y **=** tf.placeholder(tf.float32, [**None**, 10])

​

*# # ----*

*# # 构建多分类回归*

​

*# # 定义weight和bias，初始化分别为正态随机和0.0*

initial\_weight **=** tf.random\_normal([784, 10])

weight **=** tf.Variable(initial\_weight)

bias **=** tf.Variable(tf.zeros([10]))

a **=** tf.matmul(x, weight) **+** bias

y\_head **=** tf.nn.softmax(a)

​

*# # 定义二次损失函数并依据梯度下降法进行训练 -- 这样梯度下降的train就变成了x和y的函数*

learning\_rate **=** 0.1

loss **=** tf.reduce\_mean(tf.square(y **-** y\_head))

optimizer **=** tf.train.GradientDescentOptimizer(learning\_rate)

train **=** optimizer.minimize(loss)

​

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

​

correct\_prediction **=** tf.equal(tf.argmax(y, 1), tf.argmax(y\_head, 1)) *# tf.argmax找到x中等于1的最大的id*

correction **=** tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32)) *# tf.cast 转换类型，将bool转为float，从而求得准确率*

​

*# 迭代500次，进行mini\_batch梯度下降*

epoch\_n **=** 500

**with** tf.Session() **as** session:

session.run(init)

**for** step **in** range(epoch\_n):

**for** batch **in** range(batch\_n):

batch\_x, batch\_y **=** mnist.train.next\_batch(batch\_size)

session.run(train, feed\_dict**=** {x: batch\_x, y: batch\_y}) *# 此处是最小化*

corr **=** session.run(correction, feed\_dict**=** {x: mnist.test.images, y: mnist.test.labels}) *# 基于测试集对准确率进行测试*

print("in iteration " **+** str(step) **+** "the accuracy is : " **+** str(corr)) *# 打印准确率*

*# 这里看似有问题，其实没问题，因为图没变，DAG对输入的batch依次执行梯度下降法，*

*# 并执行epoch\_n个周期，权重会更新epoch\_n \* batch\_n次*

​

Extracting MNIST\_data\train-images-idx3-ubyte.gz

Extracting MNIST\_data\train-labels-idx1-ubyte.gz

Extracting MNIST\_data\t10k-images-idx3-ubyte.gz

Extracting MNIST\_data\t10k-labels-idx1-ubyte.gz

in iteration 0the accuracy is : 0.1661

in iteration 1the accuracy is : 0.1918