南京大学 人工智能学院
School of Artificial Intelligence, Nanjing University

Tencent 腾讯

智能应用建模

第三部分：神经网络模型及TensorFlow实现

章宗长

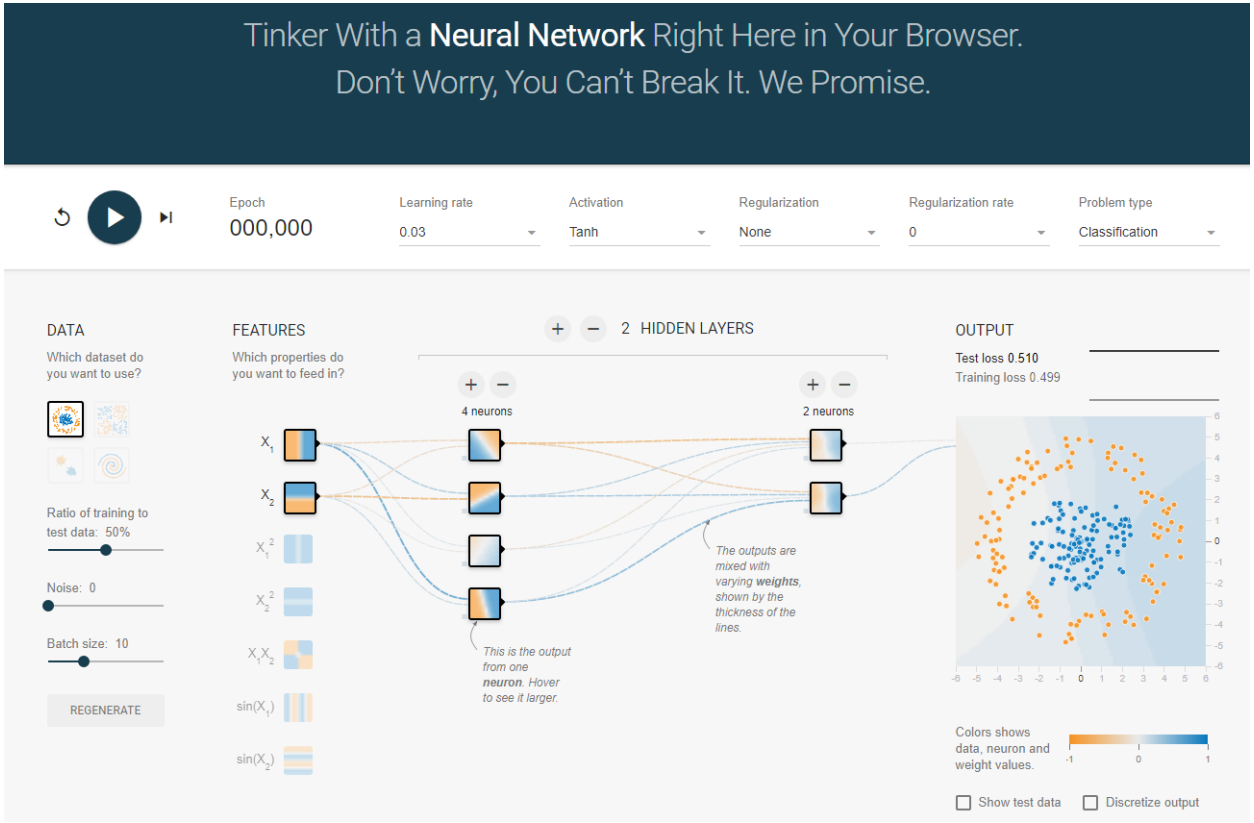
2019年7月3日

内容安排

- TensorFlow游乐场及神经网络
- 前向传播算法
- 神经网络参数与TensorFlow变量
- 通过TensorFlow训练神经网络
- 完整神经网络样例程序

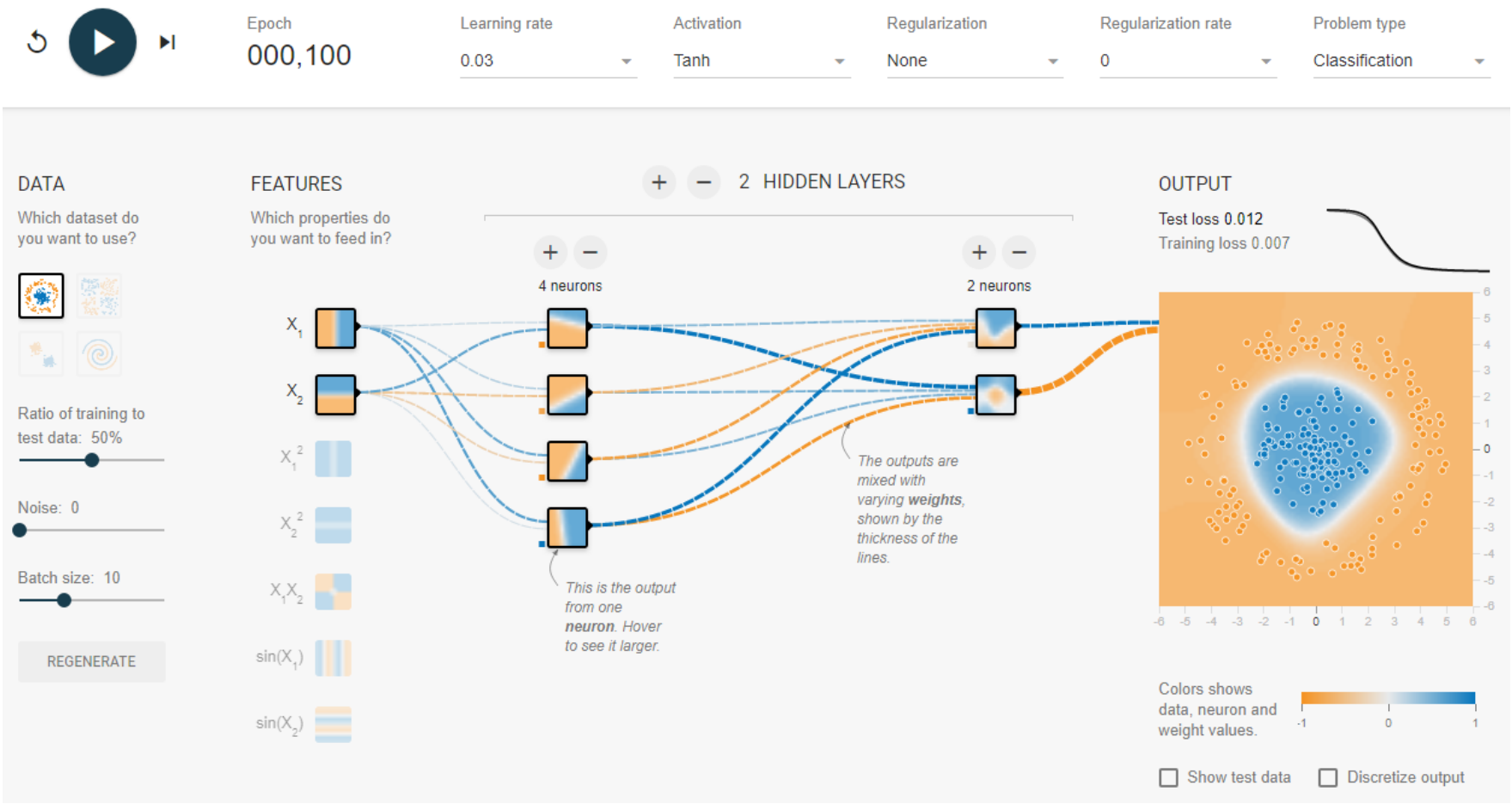
TensorFlow游乐场及神经网络

http://playground.tensorflow.org



TensorFlow游乐场界面截图

TensorFlow游乐场及神经网络（续）



TensorFlow游乐场训练100轮之后的截图

TensorFlow游乐场及神经网络（续）

使用神经网络解决分类问题主要可以分为以下4个步骤：

第一步：提取问题中实体的特征向量作为神经网络的输入

第二步：定义神经网络的结构，并定义如何从神经网络的输入得到输出，即神经网络的前向传播算法

第三步：训练神经网络，通过训练数据来调整神经网络中参数的取值

第四步：使用训练好的神经网络来预测未知的数据

内容安排

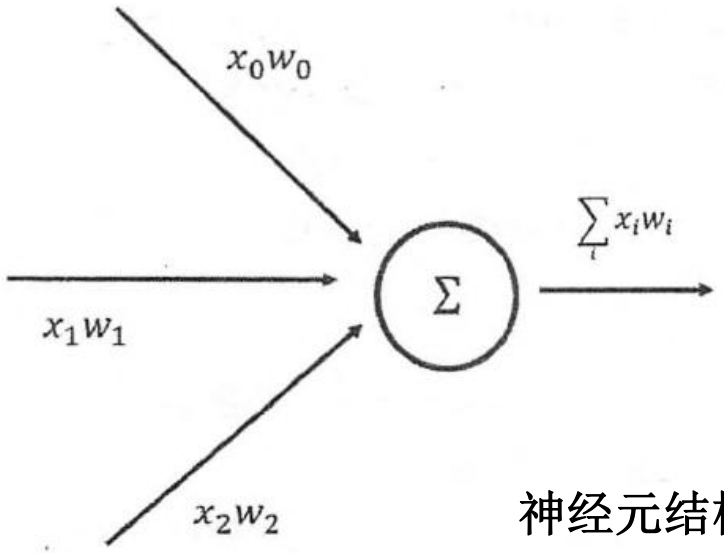
- TensorFlow游乐场及神经网络
- **前向传播算法**
- 神经网络参数与TensorFlow变量
- 通过TensorFlow训练神经网络
- 完整神经网络样例程序

前向传播算法

神经网络可以将输入的特征向量经过层层推导得到最后的输出，并通过这些输出解决分类或者回归问题

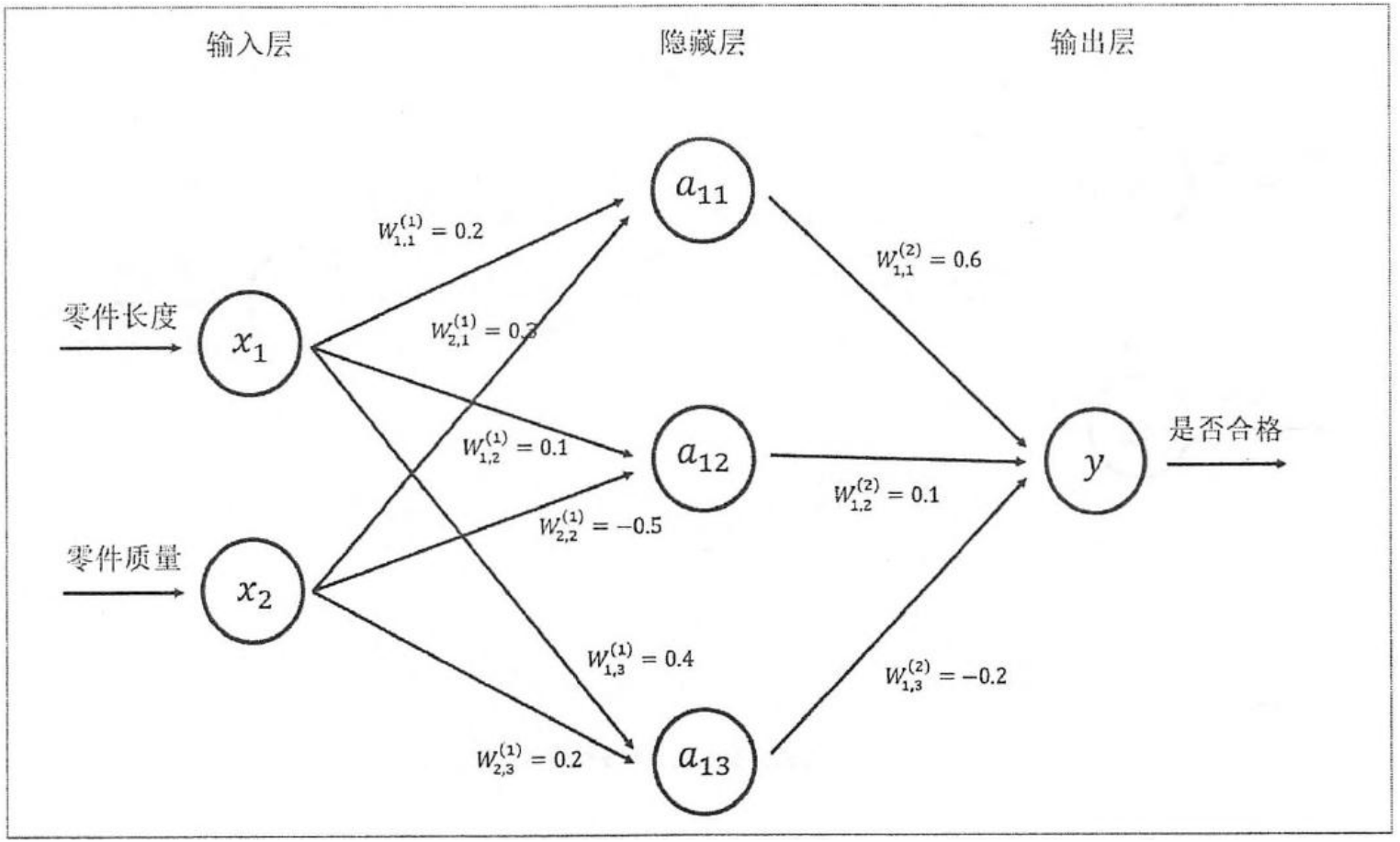
神经网络的输出是通过前向传播算法得到的

介绍最简单的全连接网络结构的前向传播算法



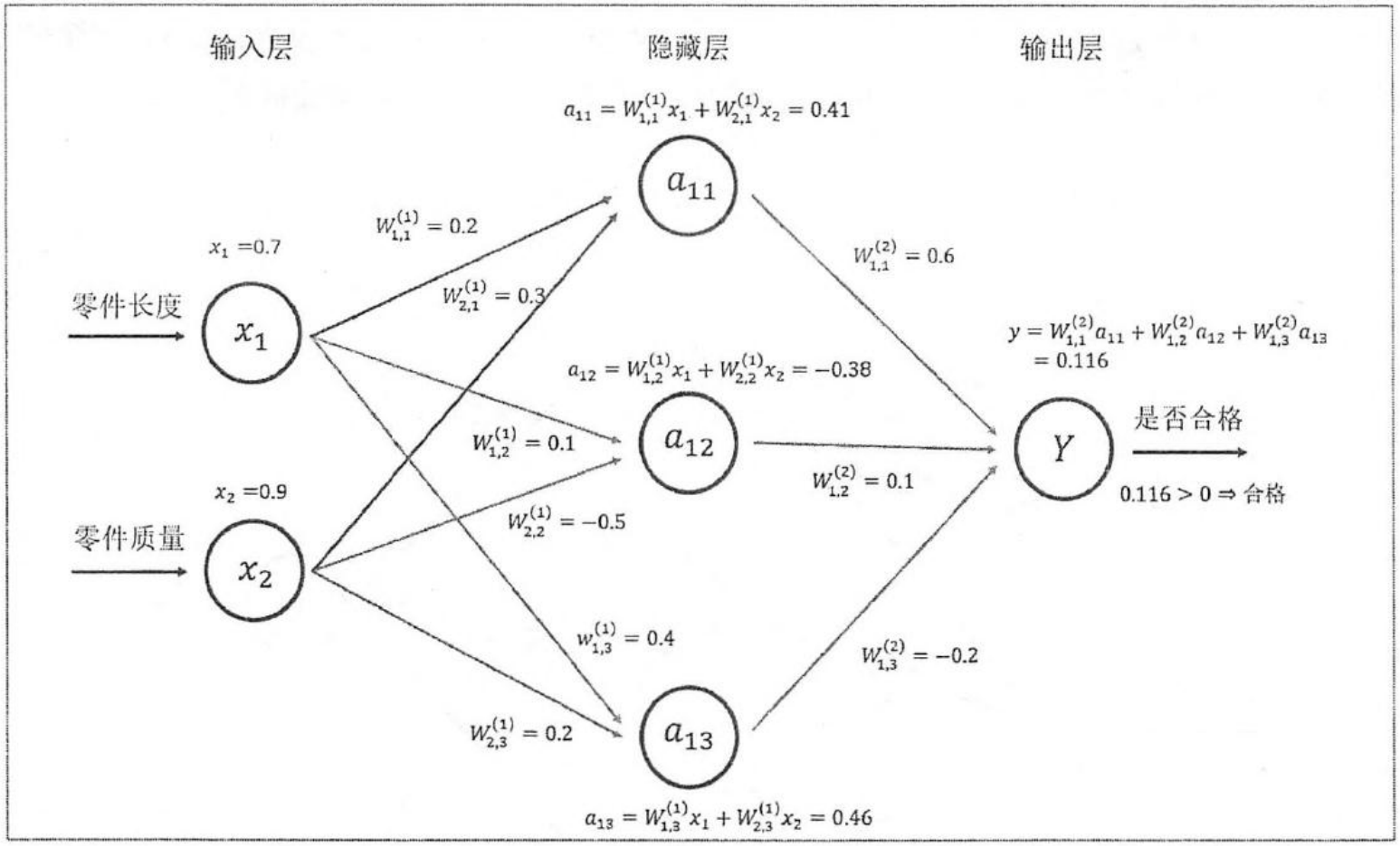
神经元结构示意图

前向传播算法（续）



判断零件是否合格的三层神经网络结构图

前向传播算法（续）



神经网络前向传播算法示意图

前向传播算法（续）

将输入x_1，x_2组织成一个1x2的矩阵$x=[x_1, x_2]$，而$W^{(1)}$组织成一个2x3的矩阵：

$$W^{(1)}=\begin{bmatrix} W_{1,1}^{(1)} & W_{1,2}^{(1)} & W_{1,3}^{(1)} \\ W_{2,1}^{(1)} & W_{2,2}^{(1)} & W_{2,3}^{(1)} \end{bmatrix}$$

通过矩阵乘法得到隐藏层三个节点所组成的向量取值：

$$a^{(1)}=[a_{11},a_{12},a_{13}]=xW^{(1)}=[x_1,x_2]\begin{bmatrix} W_{1,1}^{(1)} & W_{1,2}^{(1)} & W_{1,3}^{(1)} \\ W_{2,1}^{(1)} & W_{2,2}^{(1)} & W_{2,3}^{(1)} \end{bmatrix}$$

$$=[W_{1,1}^{(1)}x_1+W_{2,1}^{(1)}x_2, W_{1,2}^{(1)}x_1+W_{2,2}^{(1)}x_2, W_{1,3}^{(1)}x_1+W_{2,3}^{(1)}x_2]$$

前向传播算法（续）

类似的，输出层可以表示为：

$$[y]=a^{(1)}W^{(2)}=[a_{11},a_{12},a_{13}]\begin{bmatrix}W_{1,1}^{(2)}\\W_{2,1}^{(2)}\\W_{3,1}^{(2)}\end{bmatrix}=[W_{1,1}^{(2)}a_{11}+W_{1,2}^{(2)}a_{12}+W_{1,3}^{(2)}a_{13}]$$

以下TensorFlow程序实现了该神经网络的前向传播过程：

```
a = tf.matmul(x, w1)
y = tf.matmul(a, w2)
```

其中tf.matmul实现了矩阵乘法的功能

内容安排

- TensorFlow游乐场及神经网络
- 前向传播算法
- **神经网络参数与TensorFlow变量**
- 通过TensorFlow训练神经网络
- 完整神经网络样例程序

变量的声明

TensorFlow如何组织、保存以及使用神经网络中的参数？

变量的声明函数（tf.Variable）：保存和更新神经网络中的参数

声明一个2x3的矩阵变量的方法：

```
weights = tf.Variable(tf.random_normal([2, 3], stddev=2))
```

变量的初始化

TensorFlow随机生成函数

函数名称	随机数分布	主要参数
tf.random_normal	正太分布	平均值、标准差、取值类型
tf.truncated_normal	正太分布，但如果随机出来的值偏离平均值超过 2 个标准差，那么这个数将会被重新随机	平均值、标准差、取值类型
tf.random_uniform	平均分布	最小、最大取值，取值类型
tf.random_gamma	Gamma 分布	形状参数 alpha、尺度参数 beta、取值类型

TensorFlow常数生成函数

函数名称	功能	样例
tf.zeros	产生全 0 的数组	tf.zeros([2, 3], int32) -> [[0, 0, 0], [0, 0, 0]]
tf.ones	产生全 1 的数组	tf.ones([2, 3], int32) -> [[1, 1, 1], [1, 1, 1]]
tf.fill	产生一个全部为给定数字的数组	tf.fill([2, 3], 9) -> [[9, 9, 9], [9, 9, 9]]
tf.constant	产生一个给定值的常量	tf.constant([1, 2, 3]) -> [1,2,3]

变量的初始化（续）

在神经网络中，偏置项（bias）通常用常数来设置初始值：

```
biases = tf.Variable(tf.zeros([3]))
```

TensorFlow也支持通过其他变量的初始值来初始化新的变量：

```
w2 = tf.Variable(weights.initialized_value())
w3 = tf.Variable(weights.initialized_value() * 2.0)
```

样例

样例：通过变量实现神经网络的参数并实现前向传播的过程

```
import tensorflow as tf

#声明w1、w2两个变量
w1 = tf.Variable(tf.random_normal((2, 3), stddev=1, seed=1))
w2 = tf.Variable(tf.random_normal((3, 1), stddev=1, seed=1))

#暂时将输入的特征向量定义为一个常量
x = tf.constant([[0.7, 0.9]])

#通过前向传播算法获得神经网络的输出
a = tf.matmul(x, w1)
y = tf.matmul(a, w2)
```

样例（续）

样例（续）：声明一个会话，并通过会话计算结果

```
sess = tf.Session()

sess.run(w1.initializer)  #初始化w1
sess.run(w2.initializer)  #初始化w2

print(sess.run(y))
sess.close()
```

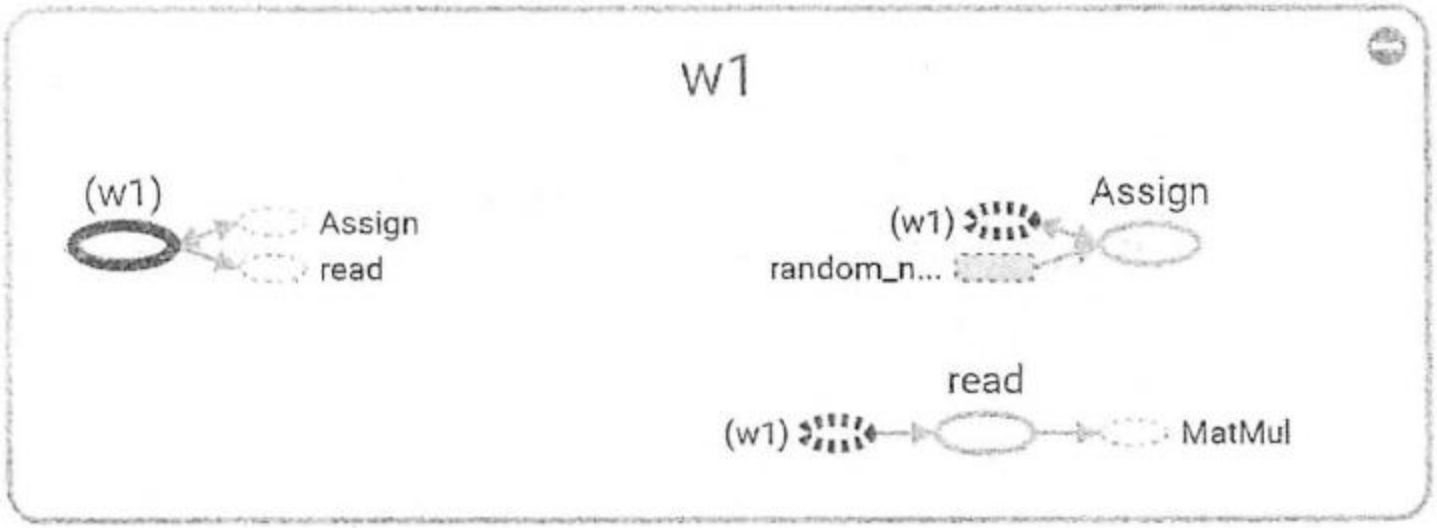
通过tf.global_variables_initializer函数实现初始化所有变量的过程

```
init_op = tf.global_variables_initializer()
sess.run(init_op)
```

变量的声明（续）

变量和张量的关系：变量的声明函数tf.Variable是一个运算。这个运算的输出结果是一个张量。**变量只是一种特殊的张量**

操作tf.Variable在TensorFlow中的底层实现



神经网络前向传播样例中变量w1相关部分的计算图可视化

集合

在一个计算图中，可以通过集合（collection）来管理不同类别的资源

通过tf.add_to_collection函数可以将资源加入一个或多个集合中，然后通过tf.get_collection获取一个集合里面的所有资源。这里的资源可以是**张量**、**变量**等

TensorFlow中维护的集合列表

集合名称	集合内容	使用场景
tf.GraphKeys.VARIABLES	所有变量	持久化 TensorFlow 模型
tf.GraphKeys.TRAINABLE_VARIABLES	可学习的变量（一般指神经网络中的参数）	模型训练、生成模型可视化内容
tf.GraphKeys.SUMMARIES	日志生成相关的张量	TensorFlow 计算可视化
tf.GraphKeys.QUEUE_RUNNERS	处理输入的 QueueRunner	输入处理
tf.GraphKeys.MOVING_AVERAGE_VARIABLES	所有计算了滑动平均值的变量	计算变量的滑动平均值

需要优化的参数

通过tf.global_variables函数可以拿到当前计算图上所有的变量

当构建机器学习模型时，比如神经网络，可以通过变量声明函数中的trainable参数来区分需要优化的参数和其他参数

如果声明变量时参数trainable为True，那么这个变量将会被加入GraphKeys.TRAINABLE_VARIABLES集合

可以通过tf.trainable_variables函数得到所有需要优化的参数

神经网络优化算法会将GraphKeys.TRAINABLE_VARIABLES集合中的变量作为默认的优化对象

变量的属性

维度和类型是变量最重要的两个属性

一个变量在构建之后，它的类型就不能再改变了

以下代码会报出类型不匹配的错误

```
w1 = tf.Variable(tf.random_normal((2, 3), stddev=1, name="w1"))
w2 = tf.Variable(tf.random_normal((2, 3), dtype=tf.float64,
stddev=1, name="w2"))

w1.assign(w2)

'''
程序将报错：
TypeError: Input 'value' of 'Assign' Op has type float64 that does
not match type float32 of argument 'ref'.
'''
```

变量的属性（续）

维度在程序运行中是有可能改变的，但是需要通过设置参数validate_shape=False

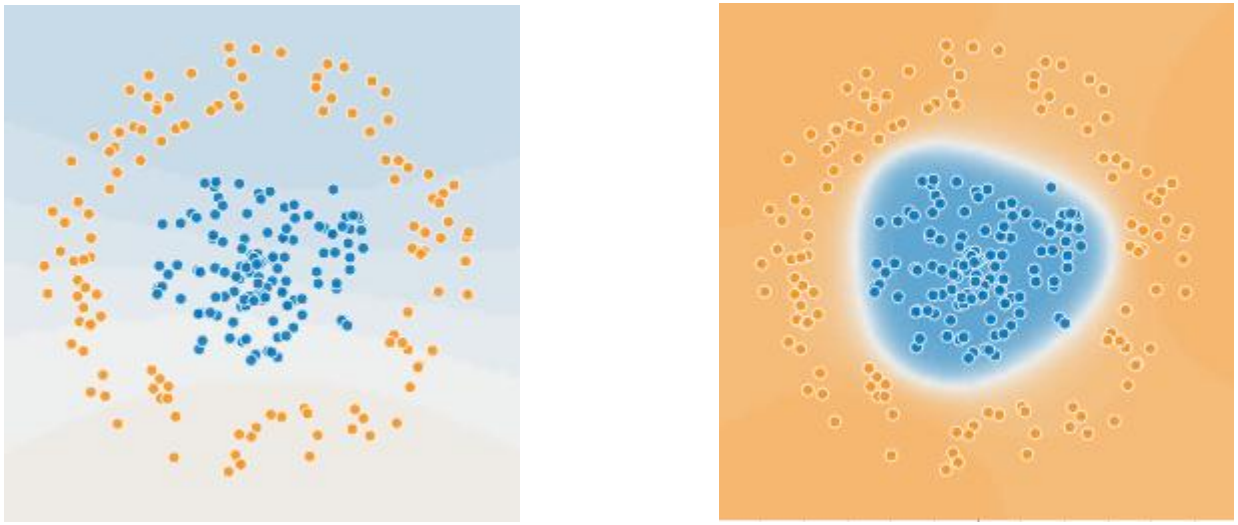
```
w1 = tf.Variable(tf.random_normal((2, 3), stddev=1, name="w1"))
w2 = tf.Variable(tf.random_normal((2, 2), stddev=1, name="w2"))

tf.assign(w1, w2)
'''
程序将报错：
ValueError: Dimension 1 in both shapes must be equal, but are 3 and 2 for 'Assign' (op: 'Assign') with input shapes: [2,3], [2,2].
'''

#这一句可以被成功执行
tf.assign(w1, w2, validate_shape=False)
```

内容安排

- TensorFlow游乐场及神经网络
- 前向传播算法
- 神经网络参数与TensorFlow变量
- **通过TensorFlow训练神经网络**
- 完整神经网络样例程序

训练前后的效果对比

只有经过有效训练的神经网络模型才可以真正地解决分类或者回归问题



TensorFlow游乐场训练前和训练后的效果对比

监督学习

使用监督学习的方式来更合理地设置网络参数的取值

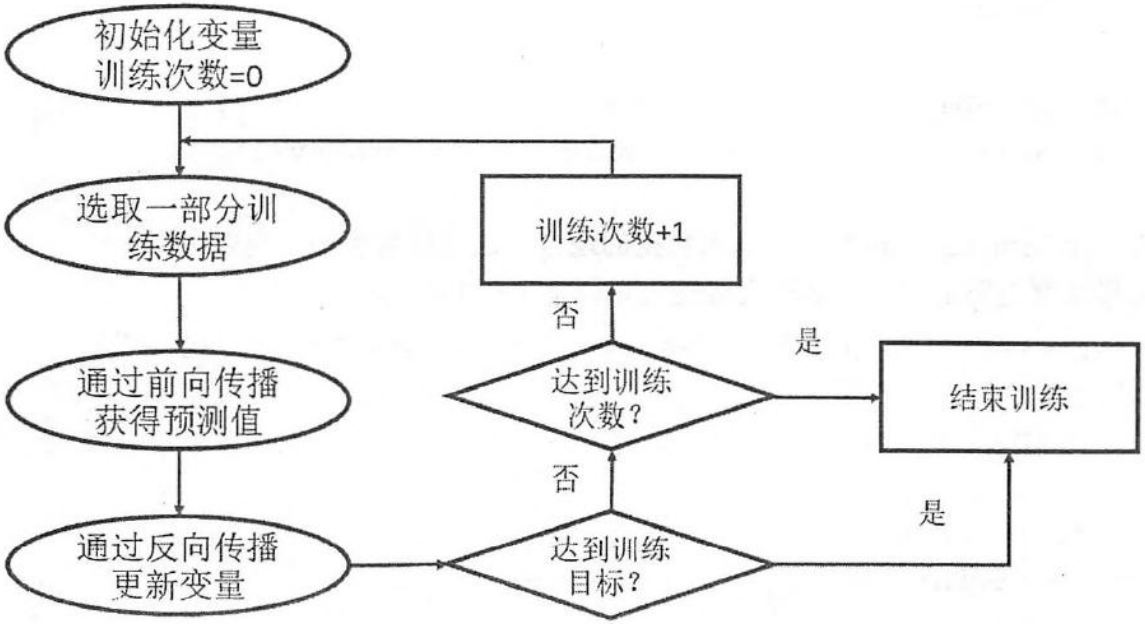
需要有一个标注好的训练数据集

监督学习的思想：在已知答案的标注数据集上，模型给出的预测结果要尽量接近真实的答案

通过调整神经网络中的参数对训练数据进行拟合，可以使得模型对未知的样本提供预测的能力

反向传播算法

反向传播算法实现了一个迭代的过程。在每次迭代中，选取一小部分训练数据（batch），得到预测值。基于预测值和真实值之间的差距，更新网络参数的取值



神经网络反向传播优化流程图

placeholder机制

TensorFlow提供了placeholder机制用于提供输入数据

placeholder相当于定义了一个位置，这个位置中的数据在程序运行时再指定

不需要生成大量常数来提供输入数据，只需要将数据通过placeholder传入TensorFlow计算图

在placeholder定义时，这个位置上的数据类型是需要指定的，且类型是不可以改变的

placeholder中数据的维度信息可以根据提供的数据推导得出，所以不一定要给出

样例

```
import tensorflow as tf

w1 = tf.Variable(tf.random_normal((2, 3), stddev=1, seed=1))
w2 = tf.Variable(tf.random_normal((3, 1), stddev=1, seed=1))

#定义placeholder作为存放输入数据的地方
x = tf.placeholder(tf.float32, shape=(1,2), name="input")

a = tf.matmul(x, w1)
y = tf.matmul(a, w2)

sess = tf.Session()
init_op = tf.global_variables_initializer()
sess.run(init_op)

print(sess.run(y, feed_dict={x: [[0.7, 0.9]]}))
sess.close()
```

输出结果：[[3.957578]]

样例（续）

一次性计算多个样例的前向传播结果

```
x = tf.placeholder(tf.float32, shape=(3,2), name="input")

...

print(sess.run(y, feed_dict={x: [[0.7, 0.9], [0.1, 0.4], [0.5, 0.8]]}))
```

输出结果：

```
[[3.957578 ]
 [1.1537654]
 [3.1674924]]
```

神经网络参数的优化

```
#使用sigmoid函数将y转换为0~1之间的数值。转换后代表预测是正样本
#的概率，1-y代表预测是负样本的概率
y = tf.sigmoid(y)

#定义损失函数来刻画预测值与真实值的差距
cross_entropy = -tf.reduce_mean(y_ * tf.log(tf.clip_by_value(y, 1e-10,
1.0)) + (1 - y_) * tf.log(tf.clip_by_value(1-y, 1e-10, 1.0)))

learning_rate = 0.001  #定义学习率

#定义反向传播算法来优化神经网络中的参数
train_step = \
    tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)
…
#对GraphKeys.TRAINABLE_VARIABLES集合中的所有变量进行优化
sess.run(train_step)
```

神经网络参数的优化（续）

目前TensorFlow支持10种不同的优化器，常见的优化方法有三种：

tf.train.GradientDescentOptimizer

tf.train.AdamOptimizer

tf.train.MomentumOptimizer

内容安排

- TensorFlow游乐场及神经网络
- 前向传播算法
- 神经网络参数与TensorFlow变量
- 通过TensorFlow训练神经网络
- **完整神经网络样例程序**

生成模拟数据集

程序的功能：在一个模拟数据集上训练神经网络，以解决二分类问题

```
import tensorflow as tf

#Numpy是一个科学计算的工具包
from numpy.random import RandomState

#通过随机数生成一个模拟数据集
rdm = RandomState(1)
dataset_size = 128
X = rdm.rand(dataset_size, 2)

#定义规则来给出样本的标签。在这里所有x1+x2<1的样本都被认为是正
#样本，而其他为负样本
Y = [[int(x1 + x2 < 1)] for (x1, x2) in X]
```

定义神经网络结构和前向传播过程

```
#定义训练数据batch的大小
batch_size = 8

#定义神经网络的参数
w1 = tf.Variable(tf.random_normal((2, 3), stddev=1, seed=1))
w2 = tf.Variable(tf.random_normal((3, 1), stddev=1, seed=1))

#在shape的一个维度上使用None可以方便使用不同的batch大小
x = tf.placeholder(tf.float32, shape=(None,2), name="x-input")
y_ = tf.placeholder(tf.float32, shape=(None,1), name="y-input")

#定义神经网络前向传播的过程
a = tf.matmul(x, w1)
y = tf.matmul(a, w2)
```

定义损失函数和反向传播算法

```
#定义损失函数和反向传播的算法
y = tf.sigmoid(y)

cross_entropy = -tf.reduce_mean(y_ * tf.log(tf.clip_by_value(y, 1e-10, 1)) + (1 - y_) * tf.log(tf.clip_by_value(1 - y, 1e-10, 1)))

train_step = tf.train.AdamOptimizer(0.001).minimize(cross_entropy)
```

生成会话

```
with tf.Session() as sess:
    init_op = tf.global_variables_initializer()
    sess.run(init_op)

    STEPS = 5000   #设置训练的轮数
    for i in range(STEPS):
        #每次选取batch_size个样本进行训练
        start = (i * batch_size) % dataset_size
        end = min(start + batch_size, dataset_size)

        #通过选取的样本训练神经网络并更新参数
        sess.run(train_step,  feed_dict = {x: X[start:end], y_: Y[start:end]})
        if i % 1000  ==  0:
            #每隔一段时间计算在所有数据上的交叉熵并输出
            total_cross_entropy = sess.run(cross_entropy, \
                                           feed_dict = {x: X, y_: Y})
            print("After %d training steps, cross entropy on all data is %g"
                        %(i, total_cross_entropy))
```

输出结果

```
After 0 training steps, cross entropy on all data is 1.89805
After 1000 training steps, cross entropy on all data is 0.655075
After 2000 training steps, cross entropy on all data is 0.626172
After 3000 training steps, cross entropy on all data is 0.615096
After 4000 training steps, cross entropy on all data is 0.610309
```

```
print sess.run(w1)
print sess.run(w2)
```

在训练之前神经网络的参数值

```
w1 = [[-0.8113182   1.4845988   0.06532937]
 [-2.4427042   0.0992484   0.5912243 ]]
w2 = [[-0.8113182 ] [ 1.4845988 ] [ 0.06532937]]
```

在训练之后神经网络的参数值

```
w1 = [[ 0.02476984  0.5694868   1.6921942 ]
 [-2.1977348  -0.23668921  1.1143897 ]]
w2 = [[-0.45544702] [ 0.4911093 ] [-0.9811033 ]]
```

小结

训练神经网络的过程可以分为以下三个步骤：

1. 定义神经网络的结构和前向传播的输出结果
2. 定义损失函数以及选择反向传播优化算法
3. 生成会话并在训练数据上反复运行反向传播优化算法