#### [用Tensorflow做手写字识别](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100" \l "collapseEXP100)

**1**[**实验介绍**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne795)

**1. 实验内容**

    为了更好的理解Neural Network，本文使用Tensorflow实现一个最简单的神经网络，然后使用MNIST数据集进行测试，同时使用Tensorboard对训练过程进行可视化。

**2. 实验目标**

    通过本实验掌握如何使用Tensorflow创建神经网络，如何将创建的神经网络应用在MNIST数据集中，如何使用Tensorboard对训练过程进行可视化。

**3. 实验知识点**

* Tensorflow创建神经网络
* Tensorboard可视化

**4. 实验环境**

* python 3.6.5
* numpy 1.14.5
* matplotlib 2.2.2
* sklearn 0.19.1
* pandas 0.23.1
* **2**[**准备工作**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne796)

**1. 项目位置**

项目目录位于~/AI\_Trial/practical\_training/tf\_hand\_writting，该目录为当前实验的工作目录。

**2. 数据集位置**

手写字数据集位于/usr/local/AI\_Trial\_Data/tf\_hand\_writing下，目录MNIST\_DATA便是手写字数据集。

**3**[**实验步骤：前言**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne797)

为了更好的理解Neural Network，本实验使用Tensorflow实现一个最简单的神经网络，然后使用MNIST数据集进行测试。同时使用Tensorboard对训练过程进行可视化。进行本实验之前，希望你已经具备机器学习和[深度学习](http://cuijiahua.com/blog/tag/%e6%b7%b1%e5%ba%a6%e5%ad%a6%e4%b9%a0/)基础。

**4**[**实验步骤：MNIST数据集简介**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne798)

当我们学习新的编程语言时，通常第一个程序就是打印输出著名的“Hello World!”。在深度学习中，MNIST数据集就相当于Hello World。

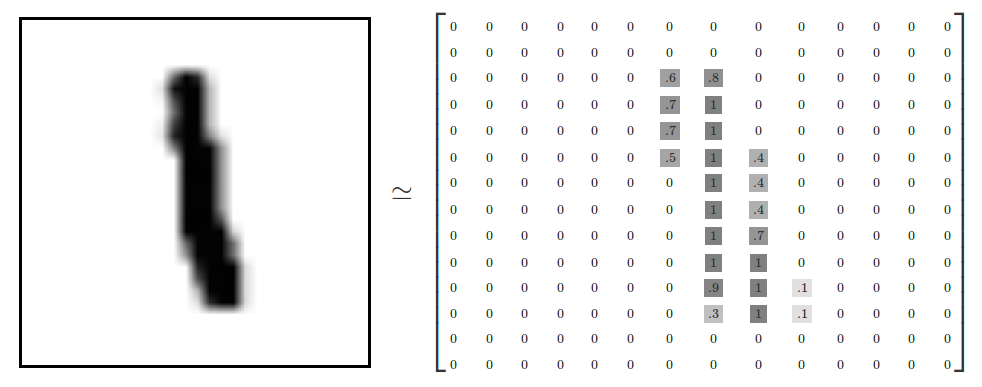
MNIST是一个简单的计算机视觉数据集，它包含手写数字的图像集：

[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_1.png)

数据集：

* train-images-idx3-ubyte 训练数据图像 (60,000)
* train-labels-idx1-ubyte 训练数据label
* t10k-images-idx3-ubyte 测试数据图像 (10,000)
* t10k-labels-idx1-ubyte 测试数据label

每张图像是28 \* 28像素：

[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_2.png)

我们的任务是使用上面数据训练一个可以准确识别手写数字的神经网络模型，并使用Tensorflow对训练过程各个参数的变化进行可视化。

**5**[**实验步骤：【Tensorboard简介】**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne799)

本实验中要使用到Tensorboard，先让我们看看它究竟是用来干什么的。

当使用Tensorflow训练大量深层的神经网络时，我们希望去跟踪神经网络的整个训练过程中的信息，比如迭代的过程中每一层参数是如何变化与分布的，比如每次循环参数更新后模型在测试集与训练集上的准确率是如何的，比如损失值的变化情况，等等。如果能在训练的过程中将一些信息加以记录并可视化得表现出来，是不是对我们探索模型有更深的帮助与理解呢？

Tensorflow官方推出了可视化工具Tensorboard，可以帮助我们实现以上功能，它可以将模型训练过程中的各种数据汇总起来存在自定义的路径与日志文件中，然后在指定的web端可视化地展现这些信息。

**6**[**实验步骤：【Tensorboard简介】-Tensorboard的数据形式**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne800)

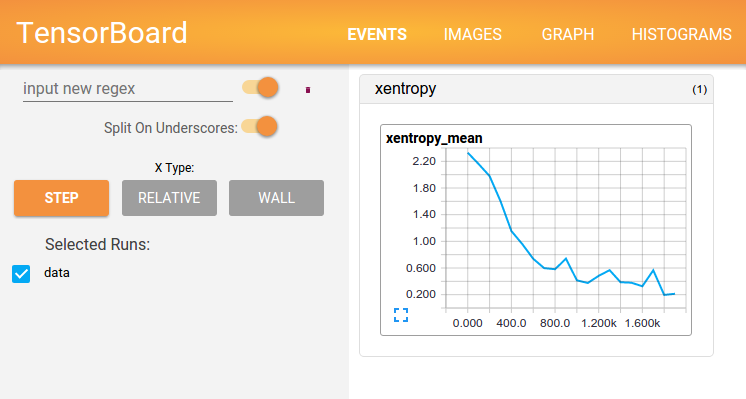
Tensorboard可以记录与展示以下数据形式：

* 标量Scalars
* 图片Images
* 音频Audio
* 计算图Graph
* 数据分布Distribution
* 直方图Histograms
* 嵌入向量Embeddings

**7**[**实验步骤：【Tensorboard简介】-Tensorboard的可视化过程简介**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne801)

TensorBoard 涉及到的运算，通常是在训练庞大的深度神经网络中出现的复杂而又难以理解的运算。为了更方便 TensorFlow 程序的理解、调试与优化，TensorFlow配备了叫做 TensorBoard 的可视化工具。你可以用 TensorBoard 来展现你的 TensorFlow 图像，绘制图像生成的定量指标图以及附加数据。

当 TensorBoard 设置完成后，它应该是这样子的：



## 数据序列化

TensorBoard 通过读取 TensorFlow 的事件文件来运行。TensorFlow 的事件文件包括了你会在 TensorFlow 运行中涉及到的主要数据。下面是 TensorBoard 中汇总数据（Summary data）的大体生命周期。

首先，创建你想汇总数据的 TensorFlow 图，然后再选择你想在哪个节点进行汇总(summary)操作。

比如，假设你正在训练一个卷积神经网络，用于识别 MNISt 标签。你可能希望记录学习速度(learning rate)的如何变化，以及目标函数如何变化。通过向节点附加scalar\_summary操作来分别输出学习速度和期望误差。然后你可以给每个 scalary\_summary 分配一个有意义的 标签，比如 'learning rate' 和 'loss function'。

或者你还希望显示一个特殊层中激活的分布，或者梯度权重的分布。可以通过分别附加 histogram\_summary 运算来收集权重变量和梯度输出。

所有可用的 summary 操作详细信息，可以查看summary\_operation文档。

在TensorFlow中，所有的操作只有当你执行，或者另一个操作依赖于它的输出时才会运行。我们刚才创建的这些节点（summary nodes）都围绕着你的图像：没有任何操作依赖于它们的结果。因此，为了生成汇总信息，我们需要运行所有这些节点。这样的手动工作是很乏味的，因此可以使用tf.merge\_all\_summaries来将他们合并为一个操作。然后你可以执行合并命令，它会依据特点步骤将所有数据生成一个序列化的Summary protobuf对象。最后，为了将汇总数据写入磁盘，需要将汇总的protobuf对象传递给tf.train.Summarywriter。

SummaryWriter 的构造函数中包含了参数 logdir。这个 logdir 非常重要，所有事件都会写到它所指的目录下。此外，SummaryWriter 中还包含了一个可选择的参数 GraphDef。如果输入了该参数，那么 TensorBoard 也会显示你的图像。

现在已经修改了你的图，也有了 SummaryWriter，现在就可以运行你的神经网络了！如果你愿意的话，你可以每一步执行一次合并汇总，这样你会得到一大堆训练数据。这很有可能超过了你想要的数据量。你也可以每一百步执行一次合并汇总，或者如下面代码里示范的这样。

merged\_summary\_op = tf.merge\_all\_summaries()  
summary\_writer = tf.train.SummaryWriter('/tmp/mnist\_logs', sess.graph)  
total\_step = 0  
while training:  
  total\_step += 1  
  session.run(training\_op)  
  if total\_step % 100 == 0:  
    summary\_str = session.run(merged\_summary\_op)  
    summary\_writer.add\_summary(summary\_str, total\_step)

现在已经准备好用 TensorBoard 来可视化这些数据了。

## 启动TensorBoard

输入下面的指令来启动TensorBoard

python tensorflow/tensorboard/tensorboard.py --logdir=path/to/log-directory

这里的参数 logdir 指向 SummaryWriter 序列化数据的存储路径。如果logdir目录的子目录中包含另一次运行时的数据，那么 TensorBoard 会展示所有运行的数据。一旦 TensorBoard 开始运行，你可以通过在浏览器中输入 localhost:6006 来查看 TensorBoard。

如果你已经通过pip安装了 TensorBoard，你可以通过执行更为简单地命令来访问 TensorBoard

tensorboard --logdir=/path/to/log-directory

进入 TensorBoard 的界面时，你会在右上角看到导航选项卡，每一个选项卡将展现一组可视化的序列化数据集 。对于你查看的每一个选项卡，如果 TensorBoard 中没有数据与这个选项卡相关的话，则会显示一条提示信息指示你如何序列化相关数据。

**8**[**实验步骤：【手写数字识别】**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne802)

现在，我们使用最基础的手写数字识别。

**9**[**实验步骤：【手写数字识别】-准备数据集、定义超参数等准备工作**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne803)

（1）首先是导入需要使用的包：

import tensorflow as tf  
from tensorflow.examples.tutorials.mnist import input\_data  
import os

（2）定义超参数

如果你问，这个超参数为啥要这样设定，如何选择最优的超参数？这个问题此处先不讨论，超参数的选择在机器学习建模中最常用的方法就是“交叉验证法”。而现在假设我们已经获得了最优的超参数，设置学利率为0.001，dropout的保留节点比例为0.9，最大循环次数为1000。

另外，还要设置两个路径，第一个是数据下载下来存放的地方，一个是summary输出保存的地方。

max\_steps = 1000  # 最大迭代次数  
learning\_rate = 0.001   # 学习率  
dropout = 0.9   # dropout时随机保留神经元的比例  
data\_dir = '/usr/local/AI\_Trial\_Data/tf\_hand\_writing/MNIST\_DATA'   # 样本数据存储的路径  
log\_dir = './MNIST\_LOG'    # 输出日志保存的路径

（3）GPU设置

这里使用GPU进行训练，如果使用cpu，可以略过此步。如果使用GPU建议进行设置。

os.environ["CUDA\_VISIBLE\_DEVICES"] = "0"  
config = tf.ConfigProto(allow\_soft\_placement = True)  
gpu\_options = tf.GPUOptions(per\_process\_gpu\_memory\_fraction = 0.33)  
config.gpu\_options.allow\_growth = True

上述代码的意思是使用GPU设备0，最多给GPU分配总共内存的百分之33，并且允许GPU按需申请内存。也就是说，假设一个程序使用一块GPU内存百分之10就够了，如果我们没有指定allow\_growth=True，那么程序会直接占用GPU内存的百分之33，因为这个是我们给它分配的。如果我们连0.33，也就是GPU内存的百分之33都没有指定，那么程序会直接占用整个GPU设备0。为了充分利用资源，特别是大家共享同一个服务器时，指定这些参数就很有必要了。

（4）下载数据下载数据是直接调用了[tensorflow](http://cuijiahua.com/blog/tag/tensorflow/)提供的函数read\_data\_sets，输入两个参数，第一个是下载到数据存储的路径，第二个one\_hot表示是否要将类别标签进行独热编码。它首先回去找制定目录下有没有这个数据文件，没有的话才去下载，有的话就直接读取。所以第一次执行这个命令，速度会比较慢，因为没有数据集，需要进行下载。在实验目录中，已经存在了该数据集，所以不用再去互联网上下载。

# 获取数据集，并采用采用one\_hot编码  
mnist = input\_data.read\_data\_sets(data\_dir,one\_hot = True)

**10**[**实验步骤：【手写数字识别】-数据处理**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne804)

（1）创建[tensorflow](http://cuijiahua.com/blog/tag/tensorflow/)默认会话：

sess = tf.InteractiveSession(config = config)

为了使设置的GPU参数生效，我们需要在创建会话的时候传入这个config参数。

（2）创建输入数据的占位符，分别创建特征数据x，标签数据y\_

在tf.placeholder()函数中传入了3个参数，第一个是定义数据类型为float32；第二个是数据的大小，特征数据是大小784的向量，标签数据是大小为10的向量，None表示不确定大小，后续可以传入任何数量的样本；第3个参数是这个占位符的名称。

with tf.name\_scope('input'):  
    x = tf.placeholder(tf.float32, [None, 784], name='x-input')  
    y\_ = tf.placeholder(tf.float32, [None, 10], name='y-input')

mnist下载好的数据集是很多个大小为1\*784的向量，就是已经对28\*28的图片进行了向量化处理。

（3）使用tf.summary.image保存图像信息

前面也说了，特征数据其实就是图像的像素数据拉升成一个1\*784的向量，现在如果想在tensorboard上还原出输入的特征数据对应的图片，就需要将拉升的向量转变成28 \* 28 \* 1的原始像素了，于是可以用tf.reshape()直接重新调整特征数据的维度：

将输入的数据转换成[28 \* 28 \* 1]的shape，存储成另一个tensor，命名为image\_shaped\_input。  
为了能使图片在tensorbord上展示出来，使用tf.summary.image将图片数据汇总给tensorbord。  
tf.summary.image（）中传入的第一个参数是命名，第二个是图片数据，第三个是最多展示的张数，此处为10张。

# 保存图像信息  
with tf.name\_scope('input\_reshape'):  
    image\_shaped\_input = tf.reshape(x, [-1, 28, 28, 1])  
    tf.summary.image('input', image\_shaped\_input, 10)

**11**[**实验步骤：【手写数字识别】-初始化参数并保存参数信息到summary**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne805)

（1）初始化参数w和b

在构建神经网络模型中，每一层中都需要去初始化参数w,b,为了使代码简介美观，最好将初始化参数的过程封装成方法function。 创建初始化权重w的方法，生成大小等于传入的shape参数，标准差为0.1，遵循正态分布的随机数，并且将它转换成tensorflow中的variable返回。

# 初始化权重参数  
def weight\_variable(shape):  
    initial = tf.truncated\_normal(shape, stddev = 0.1)  
    return tf.Variable(initial)

创建初始换偏执项b的方法，生成大小为传入参数shape的常数0.1，并将其转换成tensorflow的variable并返回。

# 初始化偏执参数  
def bias\_variable(shape):  
    initial = tf.constant(0.1, shape = shape)  
    return tf.Variable(initial)

（2）记录训练过程参数变化

我们知道，在训练的过程在参数是不断地在改变和优化的，我们往往想知道每次迭代后参数都做了哪些变化，可以将参数的信息展现在tenorbord上，因此我们专门写一个方法来收录每次的参数信息。

# 绘制参数变化  
def variable\_summaries(var):  
    with tf.name\_scope('summaries'):  
        # 计算参数的均值，并使用tf.summary.scaler记录  
        mean = tf.reduce\_mean(var)  
        tf.summary.scalar('mean', mean)  
  
        # 计算参数的标准差  
        with tf.name\_scope('stddev'):  
            stddev = tf.sqrt(tf.reduce\_mean(tf.square(var - mean)))  
        # 使用tf.summary.scaler记录记录下标准差，最大值，最小值  
        tf.summary.scalar('stddev', stddev)  
        tf.summary.scalar('max', tf.reduce\_max(var))  
        tf.summary.scalar('min', tf.reduce\_min(var))  
        # 用直方图记录参数的分布  
        tf.summary.histogram('histogram', var)

**12**[**实验步骤：【手写数字识别】-构建神经网络层**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne806)

（1）创建第一层隐藏层

  创建一个构建隐藏层的方法,输入的参数有：

* input\_tensor：特征数据
* input\_dim：输入数据的维度大小
* output\_dim：输出数据的维度大小(=隐层神经元个数）
* layer\_name：命名空间
* act=tf.nn.relu：激活函数（默认是relu)

# 构建神经网络  
def nn\_layer(input\_tensor, input\_dim, output\_dim, layer\_name, act=tf.nn.relu):  
    # 设置命名空间  
    with tf.name\_scope(layer\_name):  
        # 调用之前的方法初始化权重w，并且调用参数信息的记录方法，记录w的信息  
        with tf.name\_scope('weights'):  
            weights = weight\_variable([input\_dim, output\_dim])  
            variable\_summaries(weights)  
        # 调用之前的方法初始化权重b，并且调用参数信息的记录方法，记录b的信息  
        with tf.name\_scope('biases'):  
            biases = bias\_variable([output\_dim])  
            variable\_summaries(biases)  
        # 执行wx+b的线性计算，并且用直方图记录下来  
        with tf.name\_scope('linear\_compute'):  
            preactivate = tf.matmul(input\_tensor, weights) + biases  
            tf.summary.histogram('linear', preactivate)  
        # 将线性输出经过激励函数，并将输出也用直方图记录下来  
        activations = act(preactivate, name='activation')  
        tf.summary.histogram('activations', activations)  
    # 返回激励层的最终输出  
    return activations

调用隐层创建函数创建一个隐藏层：输入的维度是特征的维度784，隐藏层的神经元个数是500，也就是输出的维度。

hidden1 = nn\_layer(x, 784, 500, 'layer1')

（2）创建一个dropout层

随机关闭掉hidden1的一些神经元，并记录keep\_prob，减少保存参数，防止过拟合。

# 创建dropout层  
with tf.name\_scope('dropout'):  
    keep\_prob = tf.placeholder(tf.float32)  
    tf.summary.scalar('dropout\_keep\_probability', keep\_prob)  
    dropped = tf.nn.dropout(hidden1, keep\_prob)

（3）创建一个输出层

输入的维度是上一层的输出:500,输出的维度是分类的类别种类：10，激活函数设置为全等映射identity。（暂且先别使用softmax，会放在之后的损失函数中一起计算）

**13**[**实验步骤：【手写数字识别】-创造损失函数**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne807)

使用tf.nn.softmax\_cross\_entropy\_with\_logits来计算softmax并计算交叉熵损失,并且求均值作为最终的损失值。

# 创建损失函数  
with tf.name\_scope('loss'):  
    # 计算交叉熵损失（每个样本都会有一个损失）  
    diff = tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_, logits=y)  
    with tf.name\_scope('total'):  
        # 计算所有样本交叉熵损失的均值  
        cross\_entropy = tf.reduce\_mean(diff)  
    tf.summary.scalar('loss', cross\_entropy)

**14**[**实验步骤：【手写数字识别】-训练**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne808)

首先，使用AdamOptimizer优化器训练模型，最小化交叉熵损失。

# 使用AdamOptimizer优化器训练模型，最小化交叉熵损失  
with tf.name\_scope('train'):  
    train\_step = tf.train.AdamOptimizer(learning\_rate).minimize(cross\_entropy)

然后，计算准确率,并用tf.summary.scalar记录准确率。

# 计算准确率  
with tf.name\_scope('accuracy'):  
    with tf.name\_scope('correct\_prediction'):  
        # 分别将预测和真实的标签中取出最大值的索引，弱相同则返回1(true),不同则返回0(false)  
        correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1))  
    with tf.name\_scope('accuracy'):  
        # 求均值即为准确率  
        accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

**15**[**实验步骤：【手写数字识别】-送入数据集**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne809)

feed\_dict用于获取数据，如果是train==true，也就是进行训练的时候，就从mnist.train中获取一个batch大小为100样本，并且设置dropout值为0.9。如果是不是train==false,则获取minist.test的测试数据，并且设置dropout为1，即保留所有神经元开启。

同时，每隔10步，进行一次测试，并打印一次测试数据集的准确率，然后将测试数据集的各种summary信息写进日志中。 其余的时候，都是在进行训练，将训练集的summary信息并写到日志中。

# summaries合并  
merged = tf.summary.merge\_all()  
# 写到指定的磁盘路径中  
train\_writer = tf.summary.FileWriter(log\_dir + '/train', sess.graph)  
test\_writer = tf.summary.FileWriter(log\_dir + '/test')  
   
# 运行初始化所有变量  
tf.global\_variables\_initializer().run()  
   
def feed\_dict(train):  
    if train:  
        xs, ys = mnist.train.next\_batch(100)  
        k = dropout  
    else:  
        xs, ys = mnist.test.images, mnist.test.labels  
        k = 1.0  
    return {x: xs, y\_: ys, keep\_prob: k}  
   
for i in range(max\_steps):  
    if i % 10 == 0:  # 记录测试集的summary与accuracy  
        summary, acc = sess.run([merged, accuracy], feed\_dict=feed\_dict(False))  
        test\_writer.add\_summary(summary, i)  
        print('Accuracy at step %s: %s' % (i, acc))  
    else:  # 记录训练集的summary  
        summary, \_ = sess.run([merged, train\_step], feed\_dict=feed\_dict(True))  
        train\_writer.add\_summary(summary, i)  
   
train\_writer.close()  
test\_writer.close()

整体程序浏览：

import tensorflow as tf  
from tensorflow.examples.tutorials.mnist import input\_data  
import os  
  
os.environ["CUDA\_VISIBLE\_DEVICES"] = "0"  
config = tf.ConfigProto(allow\_soft\_placement = True)  
gpu\_options = tf.GPUOptions(per\_process\_gpu\_memory\_fraction = 0.33)  
config.gpu\_options.allow\_growth = True  
  
  
max\_steps = 1000  # 最大迭代次数  
learning\_rate = 0.001   # 学习率  
dropout = 0.9   # dropout时随机保留神经元的比例  
data\_dir = '/usr/local/AI\_Trial\_Data/tf\_hand\_writing/MNIST\_DATA'   # 样本数据存储的路径  
log\_dir = './MNIST\_LOG'    # 输出日志保存的路径  
  
# 获取数据集，并采用采用one\_hot热编码  
mnist = input\_data.read\_data\_sets(data\_dir,one\_hot = True)  
  
sess = tf.InteractiveSession(config = config)  
  
with tf.name\_scope('input'):  
    x = tf.placeholder(tf.float32, [None, 784], name='x-input')  
    y\_ = tf.placeholder(tf.float32, [None, 10], name='y-input')  
  
# 保存图像信息  
with tf.name\_scope('input\_reshape'):  
    image\_shaped\_input = tf.reshape(x, [-1, 28, 28, 1])  
    tf.summary.image('input', image\_shaped\_input, 10)  
  
# 初始化权重参数  
def weight\_variable(shape):  
    initial = tf.truncated\_normal(shape, stddev = 0.1)  
    return tf.Variable(initial)  
  
# 初始化偏执参数  
def bias\_variable(shape):  
    initial = tf.constant(0.1, shape = shape)  
    return tf.Variable(initial)  
  
# 绘制参数变化  
def variable\_summaries(var):  
    with tf.name\_scope('summaries'):  
        # 计算参数的均值，并使用tf.summary.scaler记录  
        mean = tf.reduce\_mean(var)  
        tf.summary.scalar('mean', mean)  
  
        # 计算参数的标准差  
        with tf.name\_scope('stddev'):  
            stddev = tf.sqrt(tf.reduce\_mean(tf.square(var - mean)))  
        # 使用tf.summary.scaler记录记录下标准差，最大值，最小值  
        tf.summary.scalar('stddev', stddev)  
        tf.summary.scalar('max', tf.reduce\_max(var))  
        tf.summary.scalar('min', tf.reduce\_min(var))  
        # 用直方图记录参数的分布  
        tf.summary.histogram('histogram', var)  
  
# 构建神经网络  
def nn\_layer(input\_tensor, input\_dim, output\_dim, layer\_name, act=tf.nn.relu):  
    # 设置命名空间  
    with tf.name\_scope(layer\_name):  
        # 调用之前的方法初始化权重w，并且调用参数信息的记录方法，记录w的信息  
        with tf.name\_scope('weights'):  
            weights = weight\_variable([input\_dim, output\_dim])  
            variable\_summaries(weights)  
        # 调用之前的方法初始化权重b，并且调用参数信息的记录方法，记录b的信息  
        with tf.name\_scope('biases'):  
            biases = bias\_variable([output\_dim])  
            variable\_summaries(biases)  
        # 执行wx+b的线性计算，并且用直方图记录下来  
        with tf.name\_scope('linear\_compute'):  
            preactivate = tf.matmul(input\_tensor, weights) + biases  
            tf.summary.histogram('linear', preactivate)  
        # 将线性输出经过激励函数，并将输出也用直方图记录下来  
        activations = act(preactivate, name='activation')  
        tf.summary.histogram('activations', activations)  
    # 返回激励层的最终输出  
    return activations  
          
hidden1 = nn\_layer(x, 784, 500, 'layer1')  
  
# 创建dropout层  
with tf.name\_scope('dropout'):  
    keep\_prob = tf.placeholder(tf.float32)  
    tf.summary.scalar('dropout\_keep\_probability', keep\_prob)  
    dropped = tf.nn.dropout(hidden1, keep\_prob)  
      
      
y = nn\_layer(dropped, 500, 10, 'layer2', act=tf.identity)  
  
# 创建损失函数  
with tf.name\_scope('loss'):  
    # 计算交叉熵损失（每个样本都会有一个损失）  
    diff = tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_, logits=y)  
    with tf.name\_scope('total'):  
        # 计算所有样本交叉熵损失的均值  
        cross\_entropy = tf.reduce\_mean(diff)  
    tf.summary.scalar('loss', cross\_entropy)  
      
      
# 使用AdamOptimizer优化器训练模型，最小化交叉熵损失  
with tf.name\_scope('train'):  
    train\_step = tf.train.AdamOptimizer(learning\_rate).minimize(cross\_entropy)  
  
# 计算准确率  
with tf.name\_scope('accuracy'):  
    with tf.name\_scope('correct\_prediction'):  
        # 分别将预测和真实的标签中取出最大值的索引，弱相同则返回1(true),不同则返回0(false)  
        correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1))  
    with tf.name\_scope('accuracy'):  
        # 求均值即为准确率  
        accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  
          
  
tf.summary.scalar('accuracy', accuracy)  
  
# summaries合并  
merged = tf.summary.merge\_all()  
# 写到指定的磁盘路径中  
train\_writer = tf.summary.FileWriter(log\_dir + '/train', sess.graph)  
test\_writer = tf.summary.FileWriter(log\_dir + '/test')  
  
# 运行初始化所有变量  
tf.global\_variables\_initializer().run()  
  
def feed\_dict(train):  
    """Make a TensorFlow feed\_dict: maps data onto Tensor placeholders."""  
    if train:  
        xs, ys = mnist.train.next\_batch(100)  
        k = dropout  
    else:  
        xs, ys = mnist.test.images, mnist.test.labels  
        k = 1.0  
    return {x: xs, y\_: ys, keep\_prob: k}  
  
for i in range(max\_steps):  
    if i % 10 == 0:  # 记录测试集的summary与accuracy  
        summary, acc = sess.run([merged, accuracy], feed\_dict=feed\_dict(False))  
        test\_writer.add\_summary(summary, i)  
        print('Accuracy at step %s: %s' % (i, acc))  
    else:  # 记录训练集的summary  
        summary, \_ = sess.run([merged, train\_step], feed\_dict=feed\_dict(True))  
        train\_writer.add\_summary(summary, i)  
  
train\_writer.close()  
test\_writer.close()

**16**[**实验步骤：【手写数字识别】-运行程序**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne810)

将上述代码保存到Python文件中，例如保存为run.py，输入以下命令运行整个程序：

python run.py train

在程序中定义的summary node就会将要记录的信息全部保存在指定的logdir路径中了，训练的记录会存一份文件，测试的记录会存一份文件。

运行程序，等待10分钟左右训练结束。

运行效果如下所示：

Accuracy at step 0: 0.1048  
Accuracy at step 10: 0.6867  
Accuracy at step 20: 0.808  
Accuracy at step 30: 0.8607  
Accuracy at step 40: 0.8859  
Accuracy at step 50: 0.8984  
Accuracy at step 60: 0.9029  
Accuracy at step 70: 0.9082  
Accuracy at step 80: 0.9095  
Accuracy at step 90: 0.9169  
Accuracy at step 100: 0.9164  
Accuracy at step 110: 0.9209  
Accuracy at step 120: 0.9145  
Accuracy at step 130: 0.9275  
Accuracy at step 140: 0.9274  
Accuracy at step 150: 0.9328  
Accuracy at step 160: 0.9338  
Accuracy at step 170: 0.932  
Accuracy at step 180: 0.9317  
Accuracy at step 190: 0.9355  
Accuracy at step 200: 0.9372  
Accuracy at step 210: 0.9412  
Accuracy at step 220: 0.9373  
Accuracy at step 230: 0.938  
Accuracy at step 240: 0.941  
Accuracy at step 250: 0.9446  
Accuracy at step 260: 0.9453  
Accuracy at step 270: 0.9435  
Accuracy at step 280: 0.9455  
Accuracy at step 290: 0.9435  
Accuracy at step 300: 0.9472  
Accuracy at step 310: 0.9477  
Accuracy at step 320: 0.9484  
Accuracy at step 330: 0.9504  
Accuracy at step 340: 0.9464  
Accuracy at step 350: 0.9472  
Accuracy at step 360: 0.9478  
Accuracy at step 370: 0.9472  
Accuracy at step 380: 0.9517  
Accuracy at step 390: 0.9538  
Accuracy at step 400: 0.9506  
Accuracy at step 410: 0.9532  
Accuracy at step 420: 0.9556  
Accuracy at step 430: 0.9538  
Accuracy at step 440: 0.9549  
Accuracy at step 450: 0.9534  
Accuracy at step 460: 0.9557  
Accuracy at step 470: 0.9523  
Accuracy at step 480: 0.9566  
Accuracy at step 490: 0.9543  
Accuracy at step 500: 0.9534  
Accuracy at step 510: 0.9566  
Accuracy at step 520: 0.9572  
Accuracy at step 530: 0.9579  
Accuracy at step 540: 0.9579  
Accuracy at step 550: 0.9538  
Accuracy at step 560: 0.9607  
Accuracy at step 570: 0.9559  
Accuracy at step 580: 0.9572  
Accuracy at step 590: 0.956  
Accuracy at step 600: 0.9619  
Accuracy at step 610: 0.9587  
Accuracy at step 620: 0.9589  
Accuracy at step 630: 0.9607  
Accuracy at step 640: 0.9637  
Accuracy at step 650: 0.9609  
Accuracy at step 660: 0.9622  
Accuracy at step 670: 0.9638  
Accuracy at step 680: 0.9626  
Accuracy at step 690: 0.9584  
Accuracy at step 700: 0.9609  
Accuracy at step 710: 0.9606  
Accuracy at step 720: 0.9596  
Accuracy at step 730: 0.9616  
Accuracy at step 740: 0.9625  
Accuracy at step 750: 0.9641  
Accuracy at step 760: 0.9644  
Accuracy at step 770: 0.9643  
Accuracy at step 780: 0.9625  
Accuracy at step 790: 0.9619  
Accuracy at step 800: 0.9638  
Accuracy at step 810: 0.9639  
Accuracy at step 820: 0.964  
Accuracy at step 830: 0.9636  
Accuracy at step 840: 0.9642  
Accuracy at step 850: 0.966  
Accuracy at step 860: 0.9667  
Accuracy at step 870: 0.9656  
Accuracy at step 880: 0.9637  
Accuracy at step 890: 0.9666  
Accuracy at step 900: 0.9652  
Accuracy at step 910: 0.9674  
Accuracy at step 920: 0.9678  
Accuracy at step 930: 0.9686  
Accuracy at step 940: 0.9684  
Accuracy at step 950: 0.968  
Accuracy at step 960: 0.9659  
Accuracy at step 970: 0.9669  
Accuracy at step 980: 0.9664  
Accuracy at step 990: 0.9676

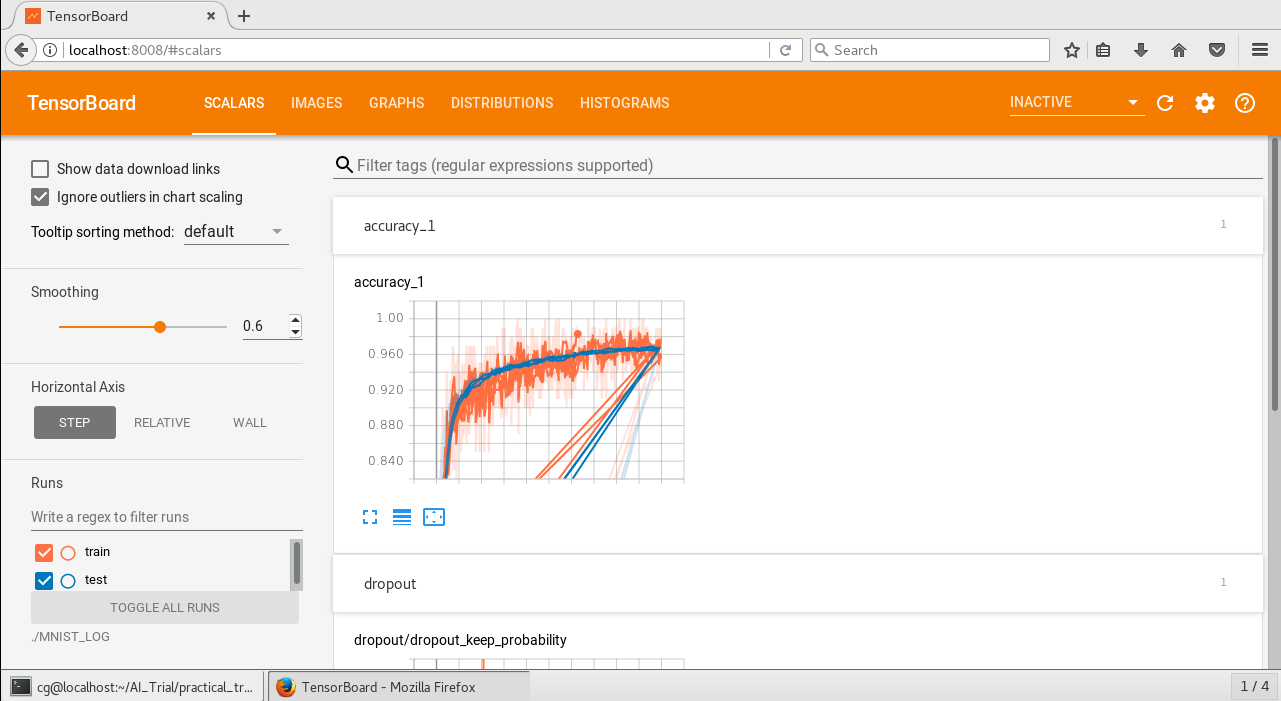
可以看到，随着迭代次数的增加，准确率也在提高。

与此同时，在运行的时候，我们就可以打开Tensorboard查看训练状态。使用如下指令：

tensorboard --logdir=./MNIST\_LOG --port=8008

上述指令logdir指定了存储log的路径，在程序里设置的路径。port指定了查看端口，此处设为8008。

运行上述指令后，我们就可以在浏览器查看Tensorboard了。



如果是远程登陆，可以在地址栏输入服务器IP地址加端口号，例如：192.168.176.167:8008，如果是本地登陆，在地址栏输入localhost:8008即可。

于是我们可以从这个web端看到所有程序中定义的可视化信息了。

**17**[**实验步骤：【Tensorboard Web端解释】**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne811)

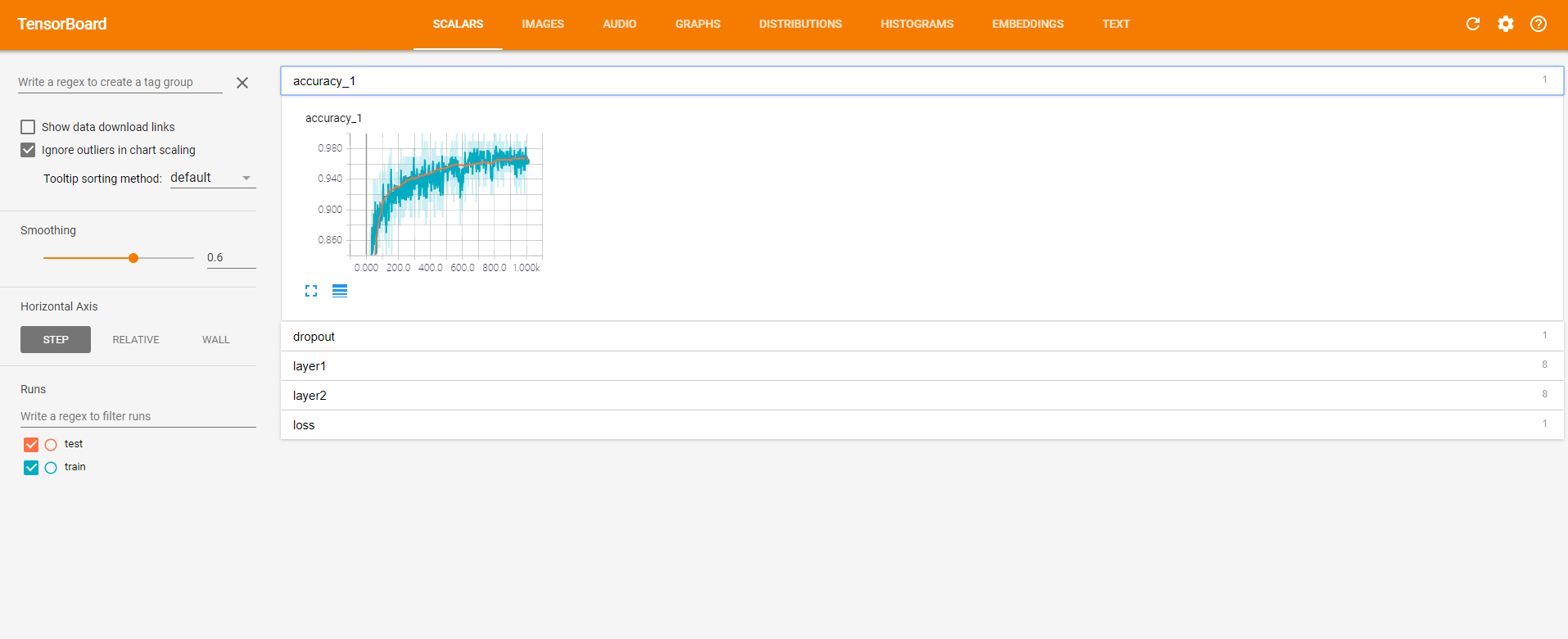
看到最上面橙色一栏的菜单，分别有7个栏目，都一一对应着我们程序中定义信息的类型。

**18**[**实验步骤：【Tensorboard Web端解释】-SCALARS**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne812)

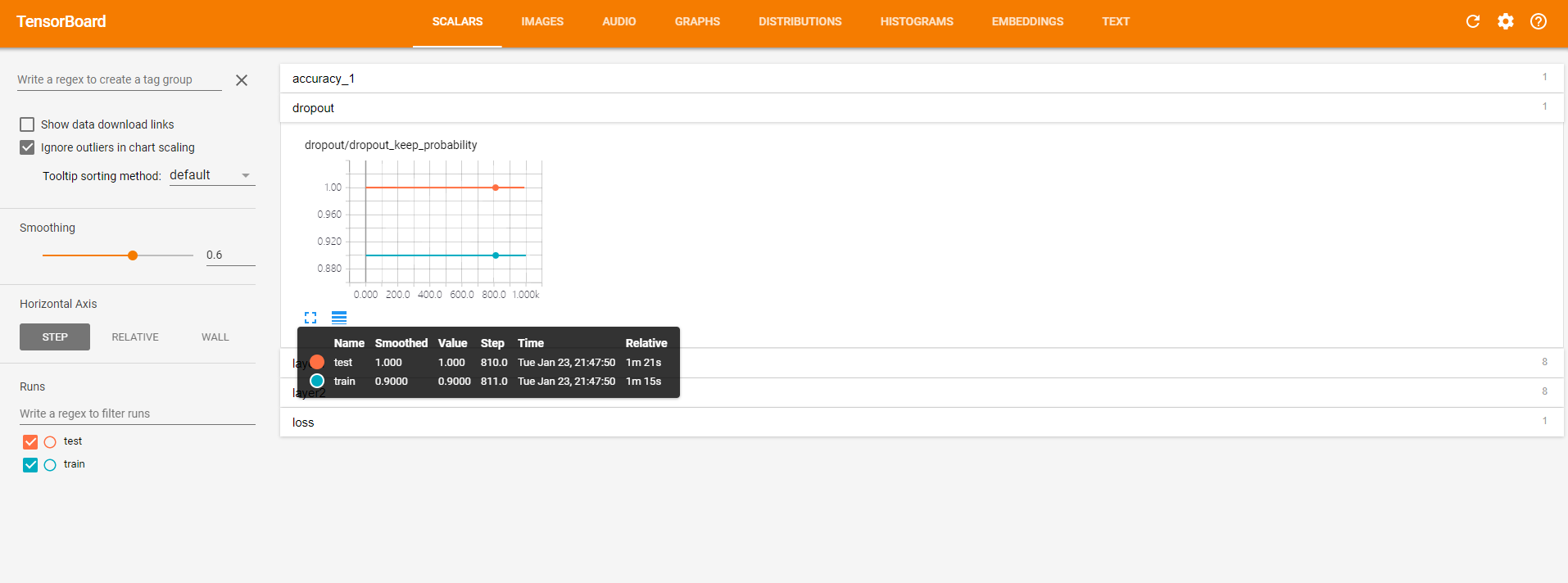
展示的是标量的信息，我程序中用tf.summary.scalars()定义的信息都会在这个窗口。 回顾程序中定义的标量有：准确率accuracy,dropout的保留率，隐藏层中的参数信息，已经交叉熵损失。这些都在SCLARS窗口下显示出来了。

点开accuracy,红线表示test集的结果，蓝线表示train集的结果，可以看到随着循环次数的增加，两者的准确度也在增加，直达1000次时会到达0.967左右。

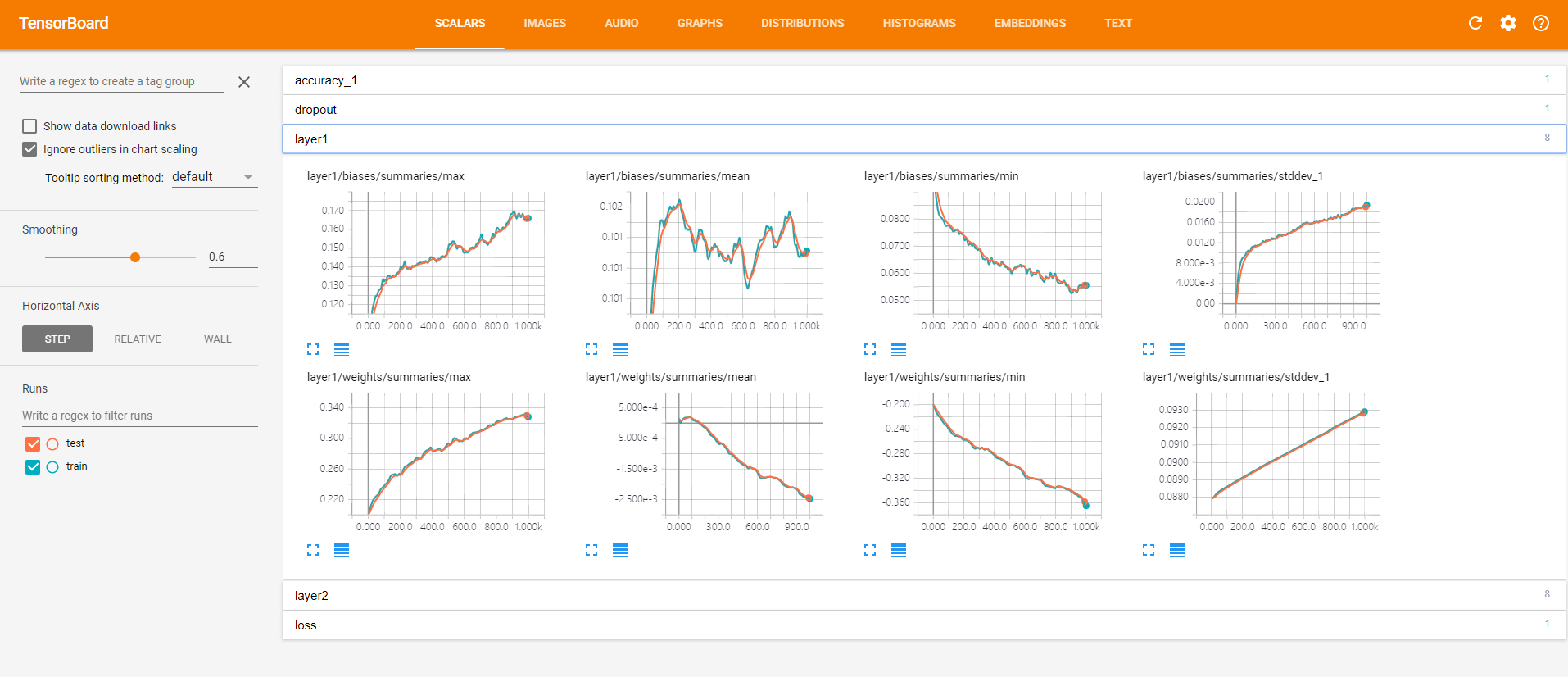
蓝线有大幅度震动是因为batch的设置问题，在每个batch里，训练效果好，但是换了一个新batch准确率就会下降，但是整体趋势还是增加的。

[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_6.png)

点开dropout，红线表示的测试集上的保留率始终是1，蓝线始终是0.9。

[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_7.png)

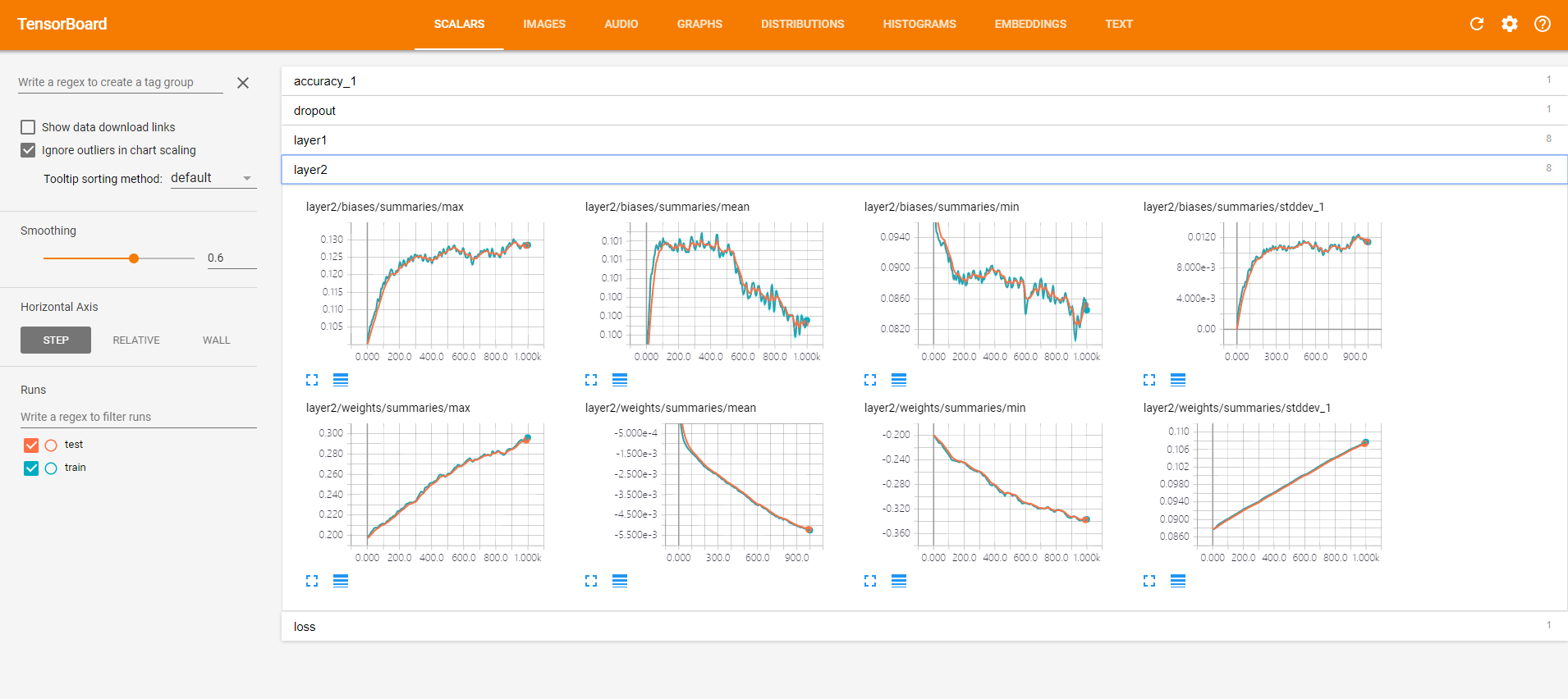
点开layer1，查看第一个隐藏层的参数信息。

[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_8.png)

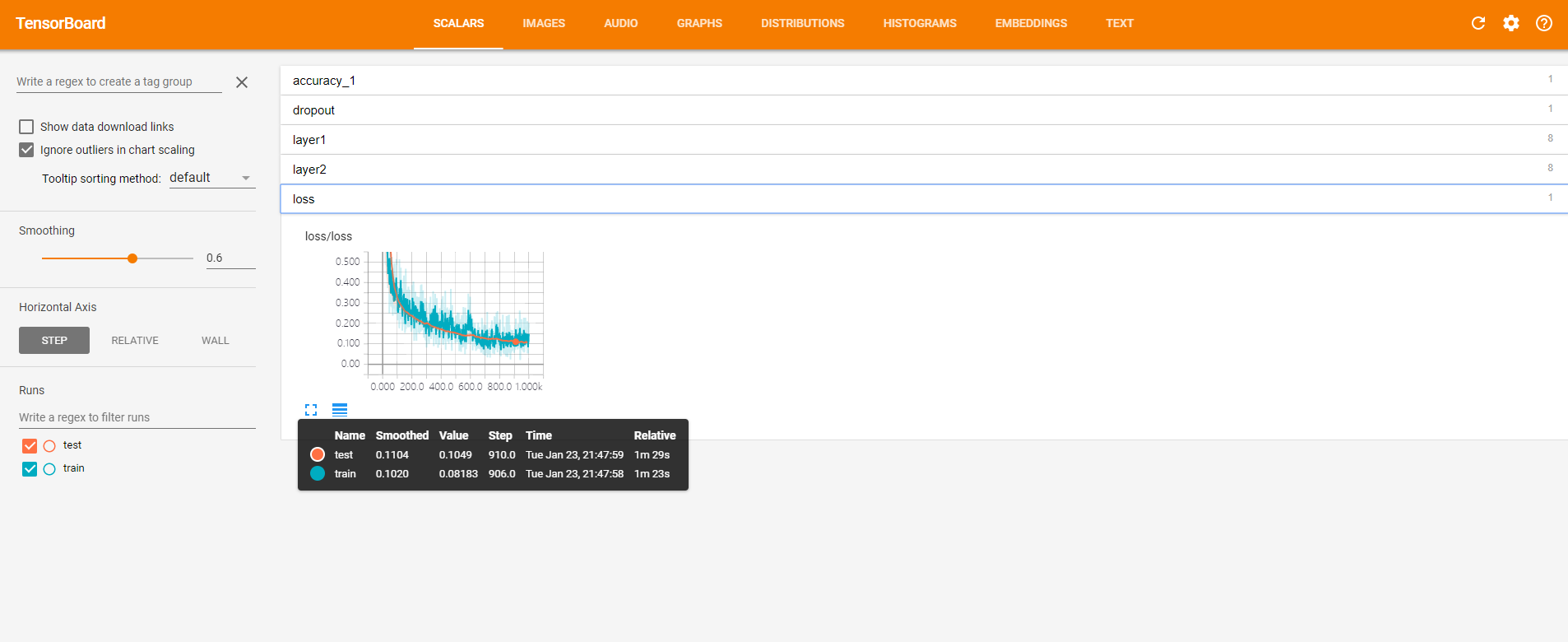
以上，第一排是偏执项b的信息，随着迭代的加深，最大值越来越大，最小值越来越小，与此同时，也伴随着方差越来越大，这样的情况是我们愿意看到的，神经元之间的参数差异越来越大。因为理想的情况下每个神经元都应该去关注不同的特征，所以他们的参数也应有所不同。

第二排是权值w的信息，同理，最大值，最小值，标准差也都有与b相同的趋势，神经元之间的差异越来越明显。w的均值初始化的时候是0，随着迭代其绝对值也越来越大。

点开layer2，查看第二层的参数信息。

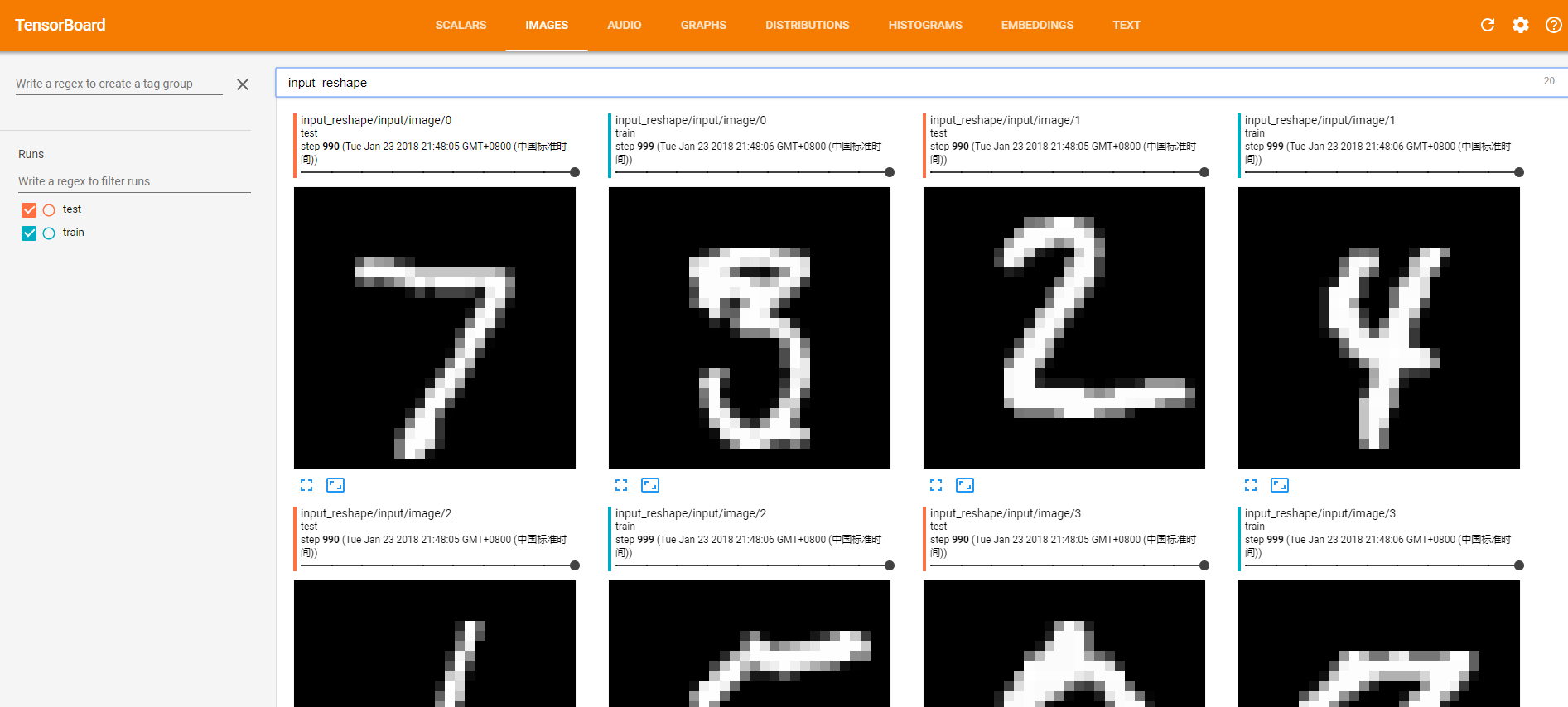
[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_9.png)

点开loss，可见损失的降低趋势。

[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_10.png)

**19**[**实验步骤：【Tensorboard Web端解释】-IMAGES**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne813)

在程序中我们设置了一处保存了图像信息，就是在转变了输入特征的shape，然后记录到了image中，于是在tensorflow中就会还原出原始的图片了：

[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_11.png)

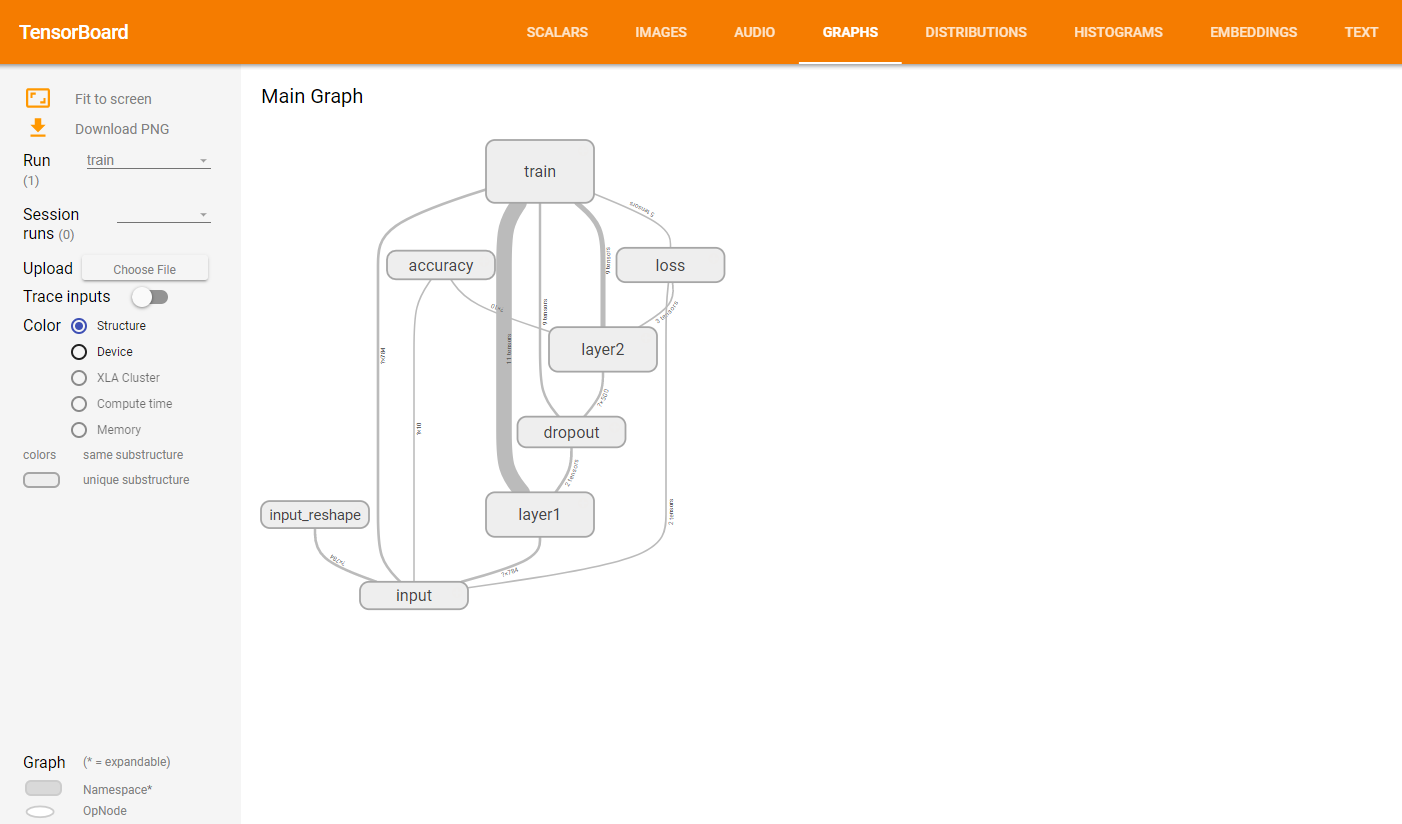
整个窗口总共展现了10张图片（根据代码中的参数10）

**20**[**实验步骤：【Tensorboard Web端解释】-AUDIO**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne814)

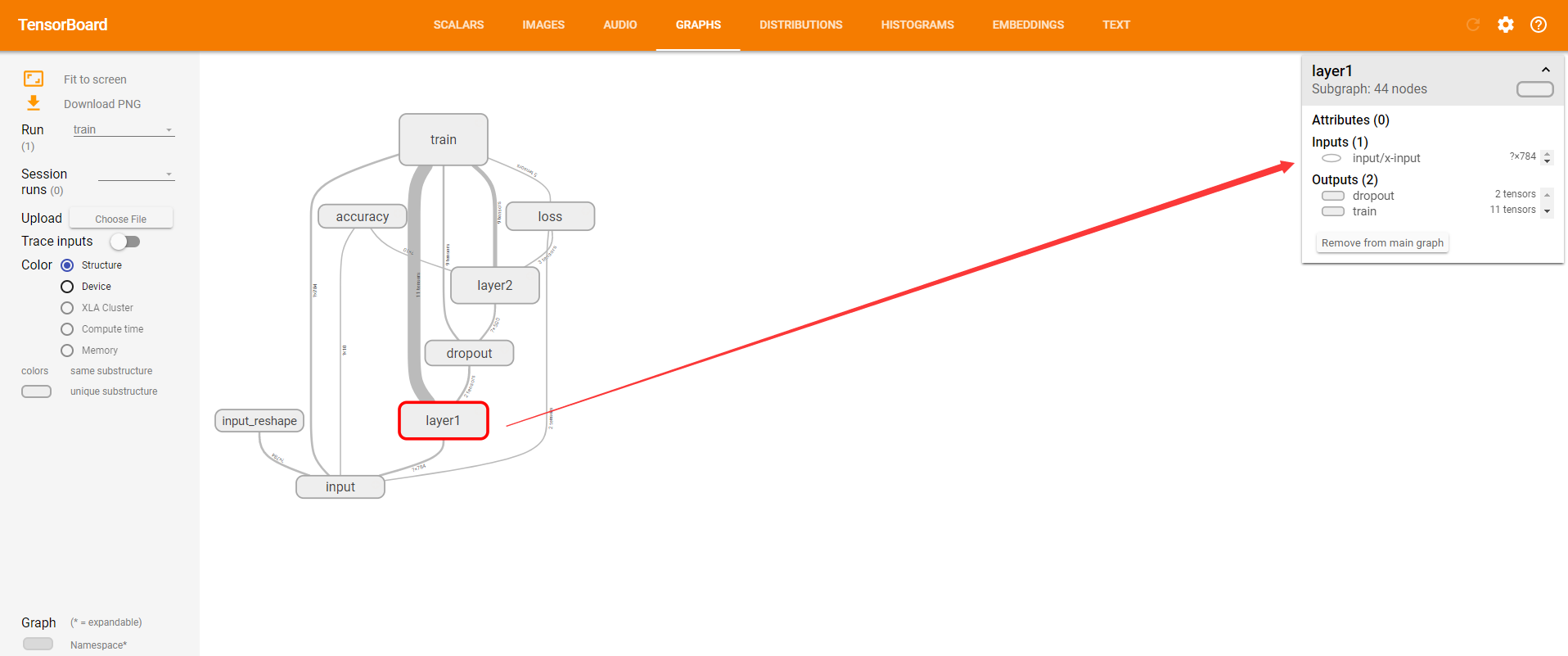
这里展示的是声音的信息，但本实验中没有涉及到声音的。

**21**[**实验步骤：【Tensorboard Web端解释】-GRAPHS**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne815)

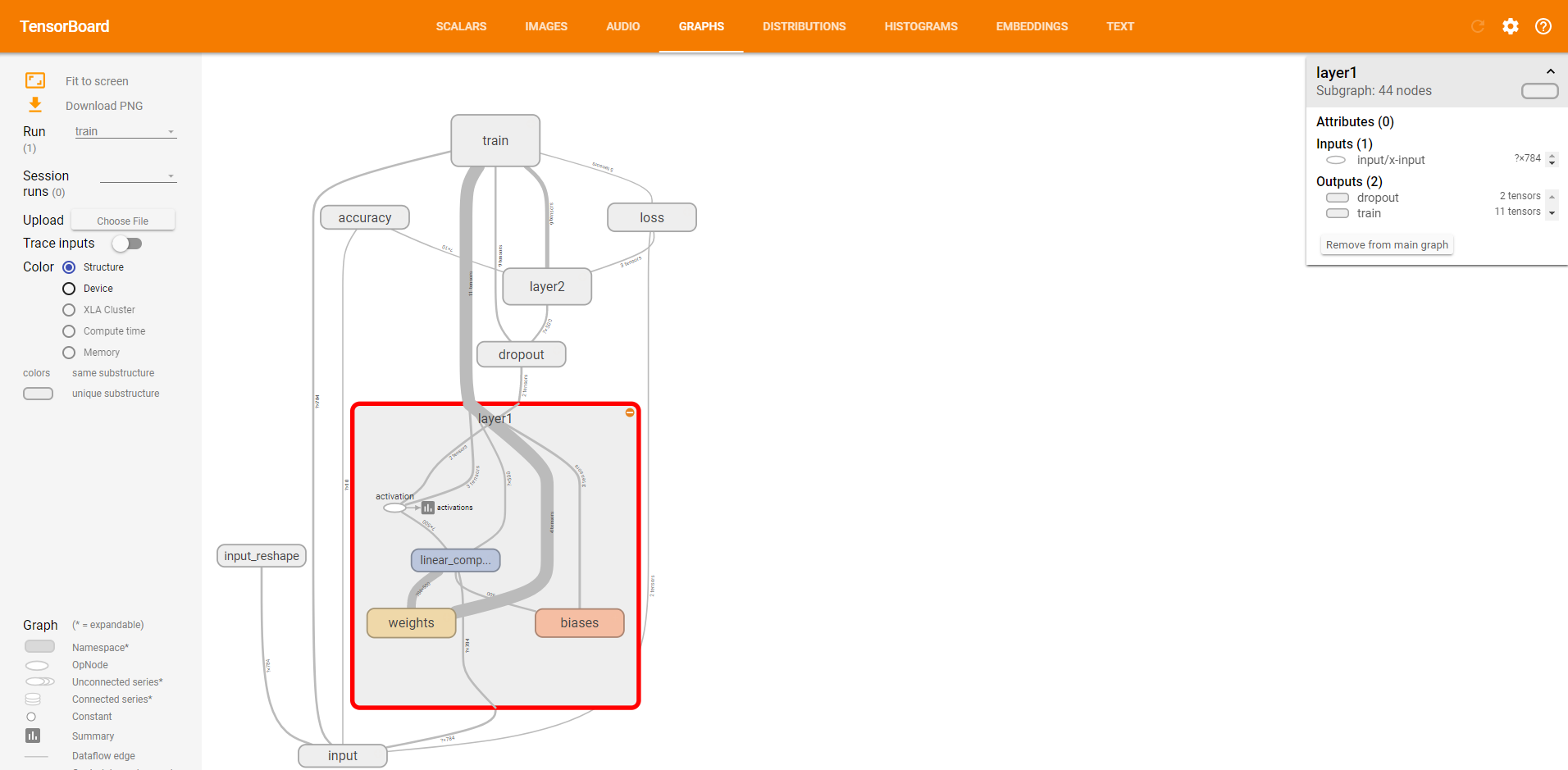
这里展示的是整个训练过程的计算图graph,从中我们可以清晰地看到整个程序的逻辑与过程。

[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_12.png)

单击某个节点，可以查看属性，输入，输出等信息。

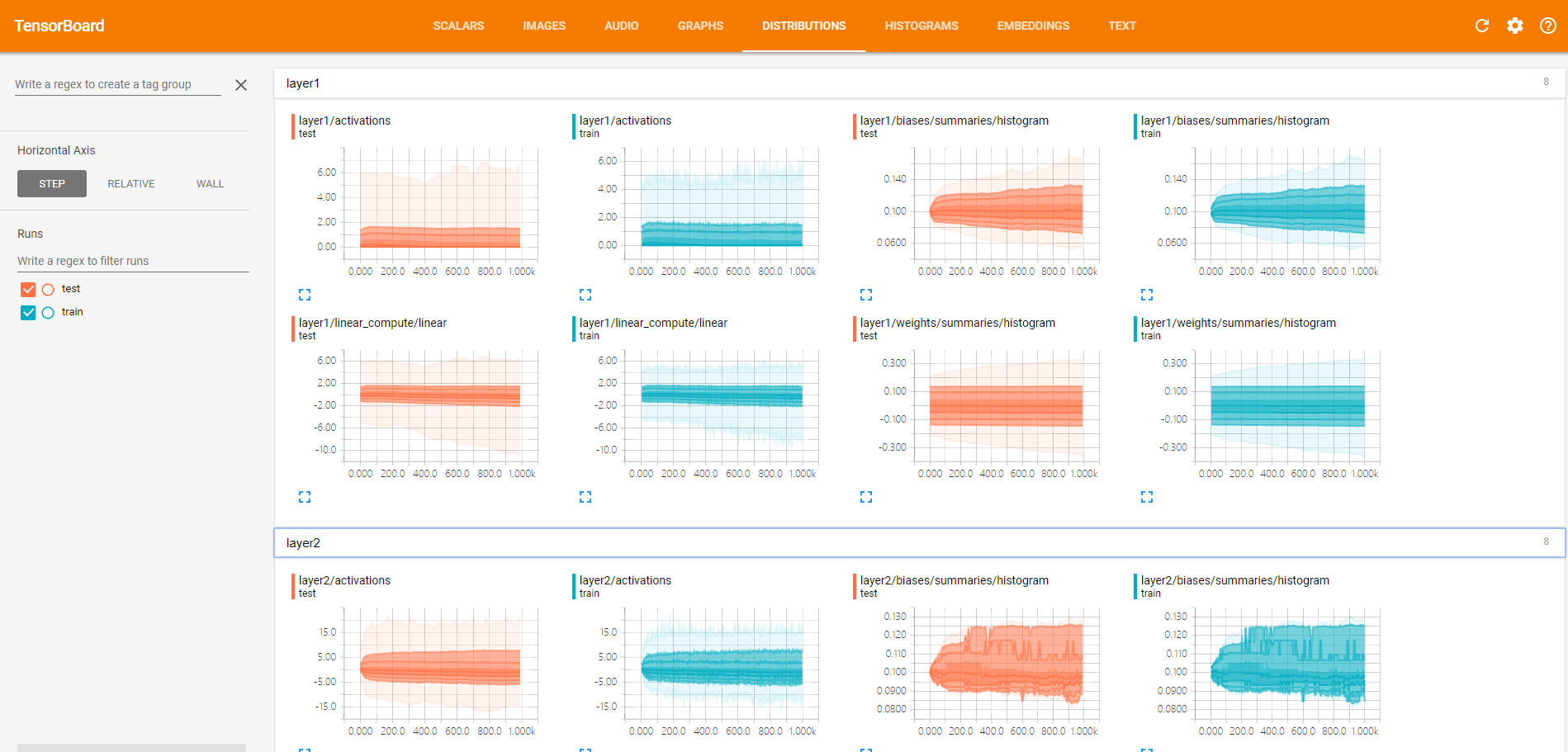
[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_13.png)

单击节点上的“+”字样，可以看到该节点的内部信息。

[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_14.png)

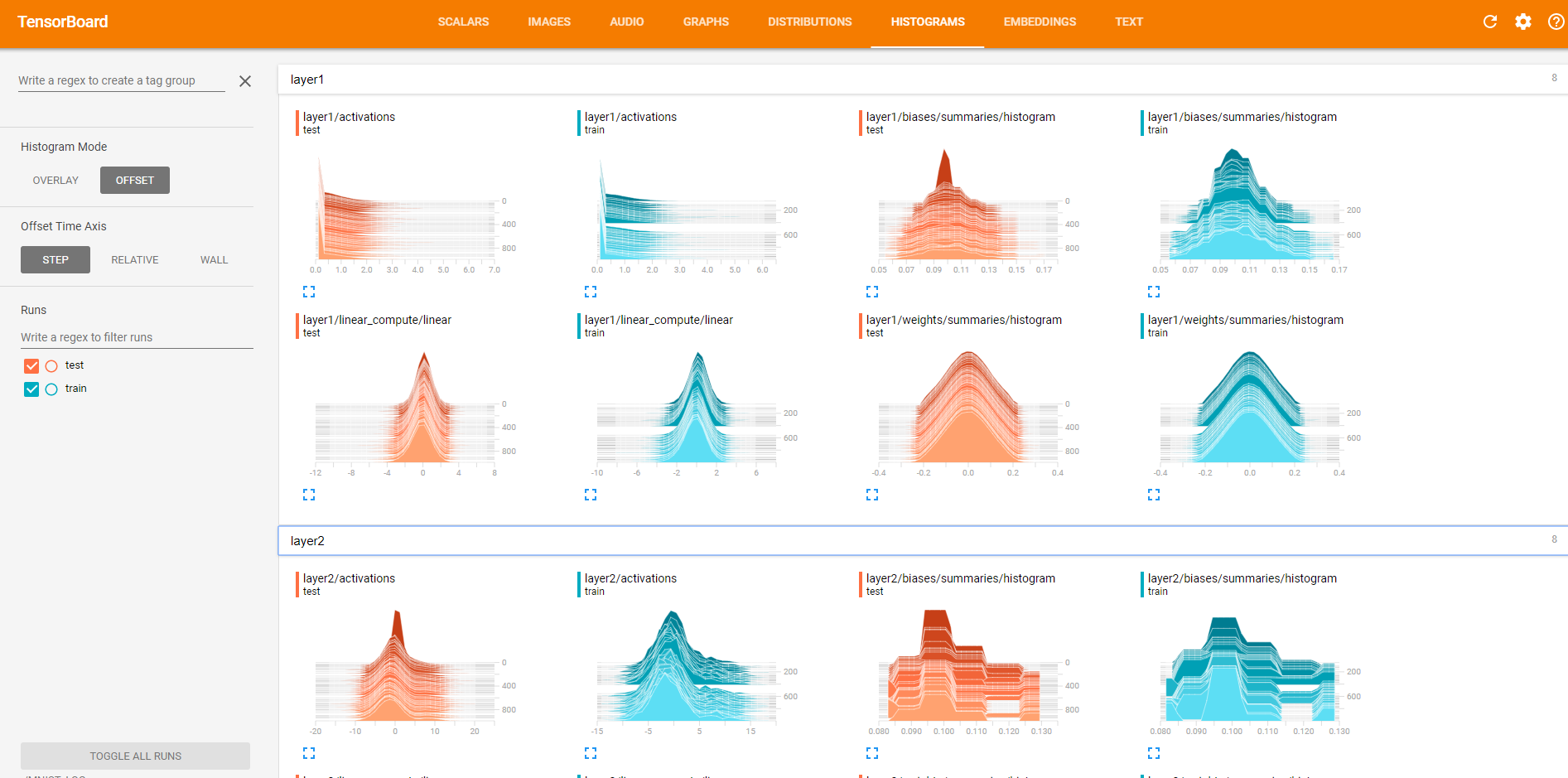
**22**[**实验步骤：【Tensorboard Web端解释】-DISTRIBUTIONS**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne816)

这里查看的是神经元输出的分布，有激活函数之前的分布，激活函数之后的分布等。

[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_15.png)

**23**[**实验步骤：【Tensorboard Web端解释】-HISTOGRAMS**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne817)

也可以看以上数据的直方图：

[](http://cuijiahua.com/wp-content/uploads/2018/01/dl_4_16.png)

**24**[**实验步骤：【Tensorboard Web端解释】-EMBEDDINGS**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne818)

展示的是嵌入向量的可视化效果，本实验中没有使用这个功能。

**25**[**实验总结**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne819)

本实验主要内容是使用手写数字识别的案例来讲解了如何初步使用Tensorflow的可视化工具Tensorboard。

**26**[**参考文献及延伸阅读**](http://demo.cjudge.net:8081/admin/courseAdmin/expAdmin/guides/editExpGuide.jsp?guideID=100#collapseOne820)

**参考资料：**

1. 郑泽宇，梁博文，顾思宇.TensorFlow：实战Google深度学习框架[M]. 电子工业出版社, 2018.

**延伸阅读：**

1. Ian Goodfellow,Yoshua Bengio,Aaron Courville. 深度学习[M]..人民邮电出版社，2017.