Udacity ML Part 4 TensorFlow

**Hello, Tensor World!**

让我们来分析一下你刚才跑的 Hello World 的代码。代码如下：

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

*# Create TensorFlow object called hello\_constant*

hello\_constant = tf.constant('Hello World!')

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

*# Run the tf.constant operation in the session*

output = sess.run(hello\_constant)

print(output)

**Tensor**

在 TensorFlow 中，数据不是以整数，浮点数或者字符串形式存在的。这些值被封装在一个叫做 tensor 的对象中。在hello\_constant = tf.constant('Hello World!') 代码中，hello\_constant是一个 0 维度的字符串 tensor，tensors 还有很多不同大小：

*# A is a 0-dimensional int32 tensor*

A = tf.constant(1234)

*# B is a 1-dimensional int32 tensor*

B = tf.constant([123,456,789])

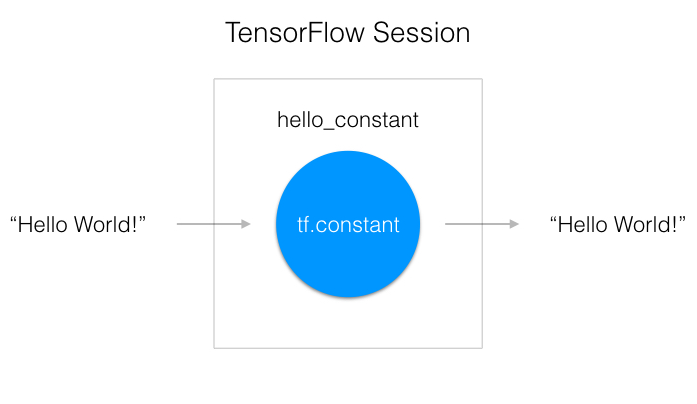
*# C is a 2-dimensional int32 tensor*

C = tf.constant([ [123,456,789], [222,333,444] ])

[**tf.constant()**](https://www.tensorflow.org/api_docs/python/tf/constant) 是你在这课中多个 TensorFlow 运算之一。**[tf.constant()](https://www.tensorflow.org/api_docs/python/tf/constant" \t "_blank)** 返回的 tensor 是一个常量 tensor，因为这个 tensor 的值不会变。

**Session**

TensorFlow 的 api 构建在 computational graph 的概念上，它是一种对数学运算过程进行可视化的一种方法。让我们把你刚才运行的 TensorFlow 的代码变成一个图：



如上图所示，一个 "TensorFlow Session" 是用来运行图的环境。这个 session 负责分配 GPU(s) 和／或 CPU(s) 包括远程计算机的运算。让我们看看怎么使用它：

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

output = sess.run(hello\_constant)

这段代码已经从上一行创建了一个 tensor hello\_constant。下一行是在session里对 tensor 求值。

这段代码用 **[tf.Session](https://www.tensorflow.org/api_docs/python/tf/Session" \t "_blank)** 创建了一个sess的 session 实例。 **[sess.run()](https://www.tensorflow.org/api_docs/python/tf/Session" \l "run" \t "_blank)** 函数对 tensor 求值，并返回结果。

# 输入

在最后一部分中，你向 session 传入一个 tensor 并返回结果。如果你想用一个非常量 non-constant 该怎么办？这就是[**tf.placeholder()**](https://www.tensorflow.org/api_docs/python/tf/placeholder) 和 feed\_dict 派上用场的时候了。这一节将向你讲解向 TensorFlow 传输数据的基本知识。

## tf.placeholder()

很遗憾，你不能把数据赋值到 x 在把它传给 TensorFlow。因为后面你需要你的 TensorFlow 模型对不同的数据集采取不同的参数。这时你需要 **[tf.placeholder()](https://www.tensorflow.org/api_docs/python/tf/placeholder" \t "_blank)**！

数据经过 **[tf.session.run()](https://www.tensorflow.org/api_docs/python/tf/Session" \l "run" \t "_blank)** 函数得到的值，由 **[tf.placeholder()](https://www.tensorflow.org/api_docs/python/tf/placeholder" \t "_blank)** 返回成一个 tensor，这样你可以在 session 开始跑之前，设置输入。

## Session’s feed\_dict

x = tf.placeholder(tf.string)

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

output = sess.run(x, feed\_dict={x: 'Hello World'})

用 **[tf.session.run()](https://www.tensorflow.org/api_docs/python/tf/Session" \l "run" \t "_blank)** 里 feed\_dict 参数设置占位 tensor。上面的例子显示 tensor x 被设置成字符串 "Hello, world"。如下所示，也可以用 feed\_dict 设置多个 tensor。

x = tf.placeholder(tf.string)

y = tf.placeholder(tf.int32)

z = tf.placeholder(tf.float32)

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

output = sess.run(x, feed\_dict={x: 'Test String', y: 123, z: 45.67})

**注意：**

如果传入 feed\_dict的数据与 tensor 类型不符，就无法被正确处理，你会得到 “ValueError: invalid literal for...”。

# TensorFlow 数学

得到输入很棒，但是现在你需要把它用起来。你将用每个人都懂的常用的数学运算，加、减、乘、除来处理 tensor。（还有更多的运算，你可以看[**文档**](https://www.tensorflow.org/api_docs/python/math_ops/)）。

## 加法

x = tf.add(5, 2) *# 7*

从加法开始， **[tf.add()](https://www.tensorflow.org/api_guides/python/math_ops" \t "_blank)** 完成的工作与你期望的一样。它把两个数字，两个 tensor，返回他们的和。

## 减法和乘法

这是减法和乘法的例子：

x = tf.subtract(10, 4) *# 6*

y = tf.multiply(2, 5) *# 10*

x tensor 求值结果是 6，因为 10 - 4 = 6。y tensor 求值结果是 10，因为 2 \* 5 = 10。是不是很简单！

## 类型转换

为了让特定运算能运行，有时会对类型进行转换。例如，你尝试下列代码，会报错：

tf.subtract(tf.constant(2.0),tf.constant(1)) # Fails with ValueError: Tensor conversion requested dtype float32 for Tensor with dtype int32:

只是因为常量 1 是整数，但是常量 2.0 是浮点数 subtract 需要他们能相符。

在这种情况下，你可以让数据都是同一类型，或者强制转换一个值到另一个类型。这里，我们可以把 2.0 转换成整数再相减，这样就能得出正确的结果：

tf.subtract(tf.cast(tf.constant(2.0), tf.int32), tf.constant(1)) # 1

# TensorFlow 里的线性函数

神经网络中最常见的运算，就是计算输入，权重和偏差的线性组合。回忆一下，我们可以把线性操作的输入写成：

https://d17h27t6h515a5.cloudfront.net/topher/2017/February/58a4d8b3_linear-equation/linear-equation.gif

这里 **W** 是连接两层的权重矩阵。输出 **y** ，输入 **x**, 偏差 **b** 全部都是向量。

## TensorFlow 里的权重和偏差

训练神经网络的目的是更新权重和偏差来更好的预测目标。为了使用权重和偏差，你需要一个能修改的 Tensor 。这就排除了[**tf.placeholder()**](https://www.tensorflow.org/api_docs/python/tf/placeholder) 和 **[tf.constant()](https://www.tensorflow.org/api_docs/python/tf/constant" \t "_blank)** 因为他们的 Tensors 不能改变。这里就需要 **[tf.Variable](https://www.tensorflow.org/api_docs/python/tf/Variable" \t "_blank)** 了。

### tf.Variable()

x = tf.Variable(5)

[**tf.Variable**](https://www.tensorflow.org/api_docs/python/tf/Variable) 类创建一个 tensor，它的初始值可以被改变，就像普通的 Python 变量一样。tensor 把它的状态存在 session里，所以你必须手动初始化它的状态。你用[**tf.global\_variables\_initializer()**](https://www.tensorflow.org/api_docs/python/tf/global_variables_initializer) 来初始化所有可变 tensors。

##### 初始化

init = tf.global\_variables\_initializer()

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

sess.run(init)

[**tf.global\_variables\_initializer()**](https://www.tensorflow.org/api_docs/python/tf/global_variables_initializer) 会返回一个操作，它会从graph中初始化所有的 TensorFlow 变量。你可以通过 session 来呼叫这个操作来初始化所有上面的变量。用 **[tf.Variable](https://www.tensorflow.org/api_docs/python/tf/Variable" \t "_blank)** 类可以让我们改变权重和偏差，但还是要选择一个初始值。

从正态分布中初始化权重是个好习惯。随机化权重可以避免每次训练时候模型卡在同一个地方。在下节学习梯度下降的时候，你会学到更多。

类似地，从正态分布中选择权重可以避免任意一个权重与其他权重相比有压倒性的特性。你可以用[**tf.truncated\_normal()**](https://www.tensorflow.org/api_docs/python/tf/truncated_normal) 函数从一个正态分布中产生随机数。

### tf.truncated\_normal()

n\_features = 120

n\_labels = 5

weights = tf.Variable(tf.truncated\_normal((n\_features, n\_labels)))

[**tf.truncated\_normal()**](https://www.tensorflow.org/api_docs/python/tf/truncated_normal) 返回一个 tensor，它的随机值取自一个正态分布，并且它们的取值会在这个正态分布平均值的两个标准差之内。

因为权重已经被随机化来帮助模型不被卡住，你不需要再把偏差随机化了。让我们简单地把偏差设为 0。

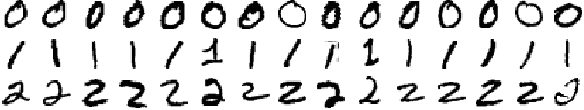
### tf.zeros()

n\_labels = 5

bias = tf.Variable(tf.zeros(n\_labels))

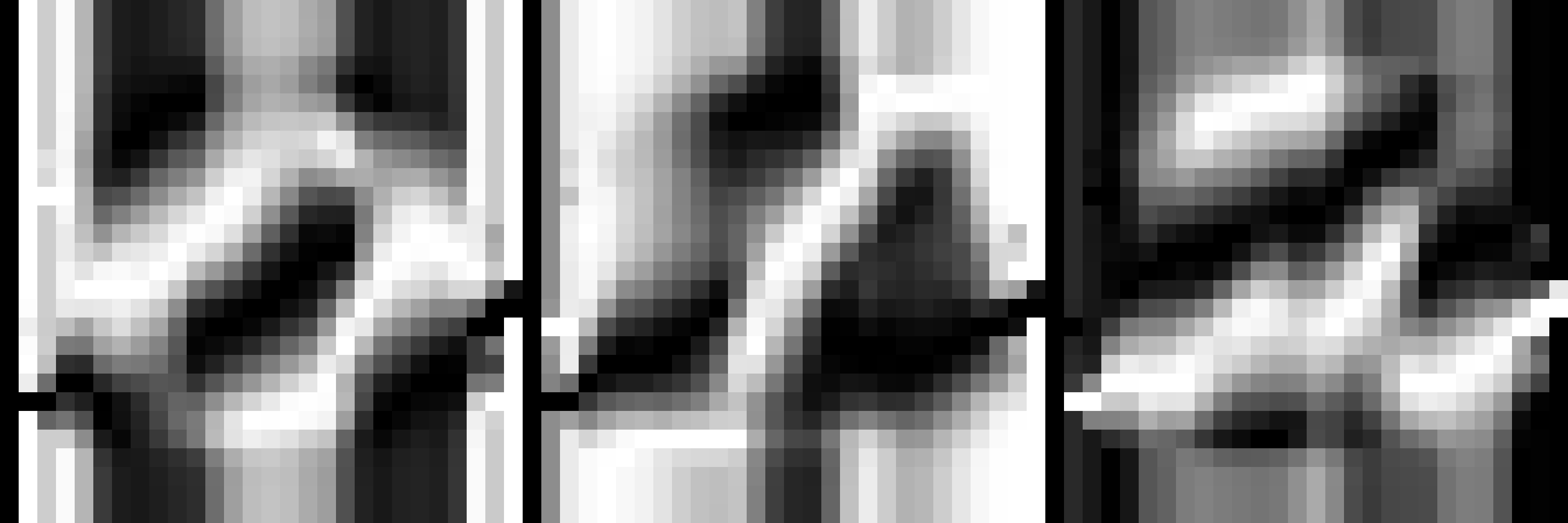
[**tf.zeros()**](https://www.tensorflow.org/api_docs/python/tf/zeros) function 返回一个都是 0 的 tensor。

## 线性分类练习



A subset of the MNIST dataset

你试着来用 TensorFlow 来对 MNIST 数据集中的手写数字 0, 1和 2进行分类。上图是你训练数据的示意。你会注意到有些 1是在顶部有不同角度的[**serif（衬线体）**](https://en.wikipedia.org/wiki/Serif)。这些相同点和不同点对构建模型的权重会有影响。



左: label为0的权重。中: label是 1的权重。 右: label为2的权重。

上图是每个 label (0, 1和 2) 训练得到的权重。权重显示了他们找到的每个数字的独特性。用MNIST来训练你的权重，完成这个练习。

### 说明

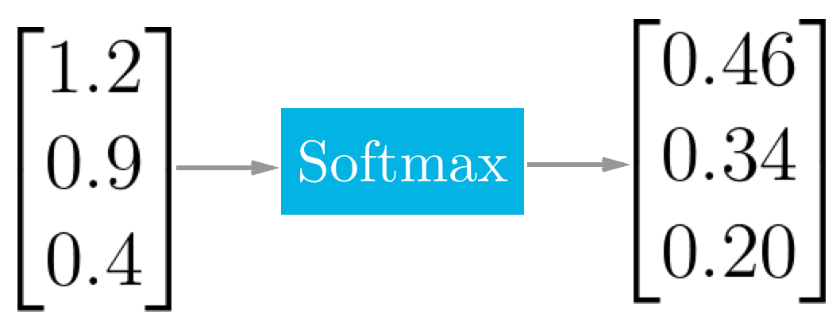
1. 打开 quiz.py
   1. 实现 get\_weights 让它返回一个 **[tf.Variable](https://www.tensorflow.org/api_docs/python/tf/Variable" \t "_blank)** 的权重
   2. 实现 get\_biases 返回一个 **[tf.Variable](https://www.tensorflow.org/api_docs/python/tf/Variable" \t "_blank)** 偏差
   3. 实现 xW + b，线性函数
2. 打开 sandbox.py
   1. 初始化权重

因为xW + b中的 xW是矩阵相乘，所以你要用 **[tf.matmul()](https://www.tensorflow.org/api_docs/python/tf/matmul" \t "_blank)** 函数，而不是 **[tf.multiply()](https://www.tensorflow.org/api_docs/python/tf/multiply" \t "_blank)**。不要忘记矩阵相乘的规则tf.matmul(a,b) 不等于tf.matmul(b,a)。

quiz.py sandbox.py

# TensorFlow Softmax

The softmax 函数可以把它的输入，通常被称为 **logits** 或者 **logit scores**，处理成 0 到 1 之间，并且能够把输出归一化到和为 1。这意味着 softmax 函数与分类的概率分布等价。它是一个网络预测多酚类问题的最佳输出激活函数。



softmax 函数的实际应用示例

## TensorFlow Softmax

当我们用 TensorFlow 来构建一个神经网络时，相应地，它有一个计算 softmax 的函数。

x = tf.nn.softmax([2.0, 1.0, 0.2])

就是这么简单， **[tf.nn.softmax()](https://www.tensorflow.org/api_docs/python/tf/nn/softmax" \t "_blank)** 直接为你实现了 softmax 函数，它输入 logits，返回 softmax 激活函数。

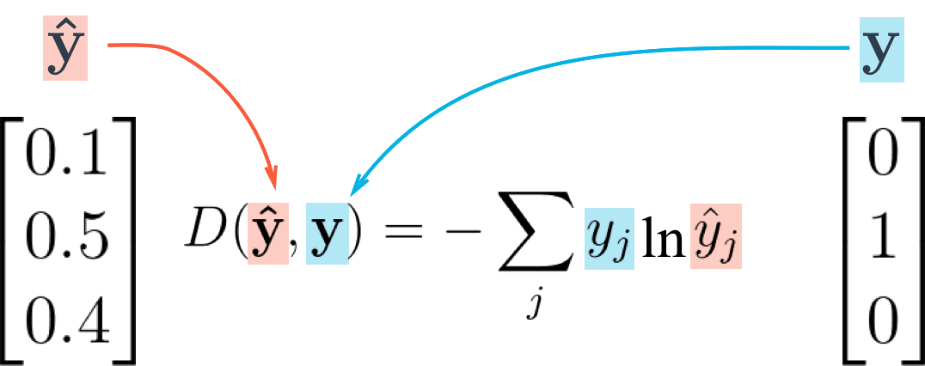
## 练习

在下面使用 softmax 函数返回 logits 的 softmax。

Softmax.py

**TensorFlow 中的交叉熵（Cross Entropy）**

与 softmax 一样，TensorFlow 也有一个函数可以方便地帮我们实现交叉熵。



Cross entropy loss function 交叉熵损失函数

让我们把你从视频当中学到的知识，在 TensorFlow 中来创建一个交叉熵函数。创建一个交叉熵函数，你需要用到这两个新的函数：

* [**tf.reduce\_sum()**](https://www.tensorflow.org/api_docs/python/tf/reduce_sum)
* [**tf.log()**](https://www.tensorflow.org/api_docs/python/tf/log)

**Reduce Sum**

x = tf.reduce\_sum([1, 2, 3, 4, 5]) *# 15*

[**tf.reduce\_sum()**](https://www.tensorflow.org/api_docs/python/tf/reduce_sum) 函数输入一个序列，返回他们的和

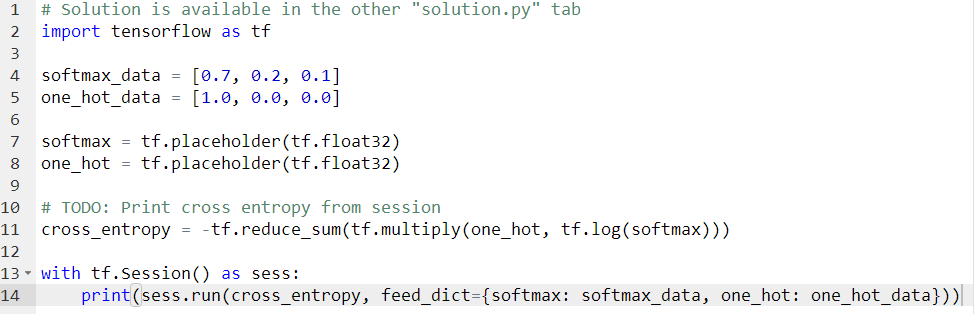
**Natural Log**

x = tf.log(100) *# 4.60517*

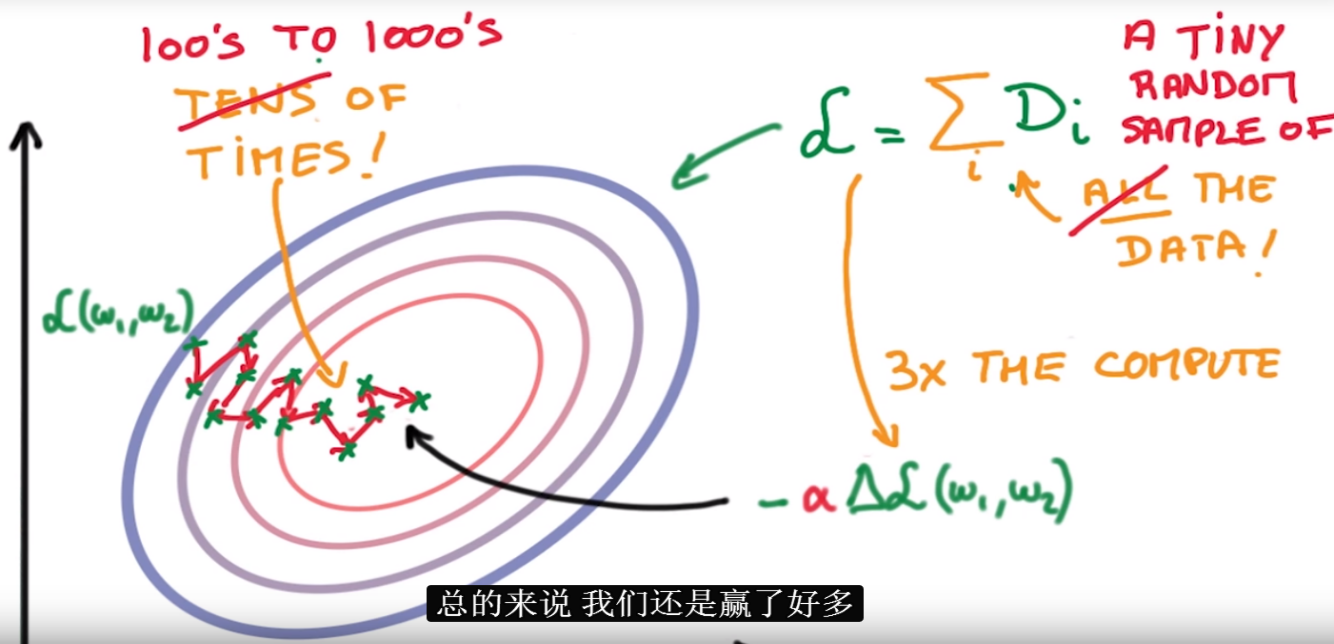
[**tf.log()**](https://www.tensorflow.org/api_docs/python/tf/log) 所做跟你所想的一样，它返回所输入值的自然对数。

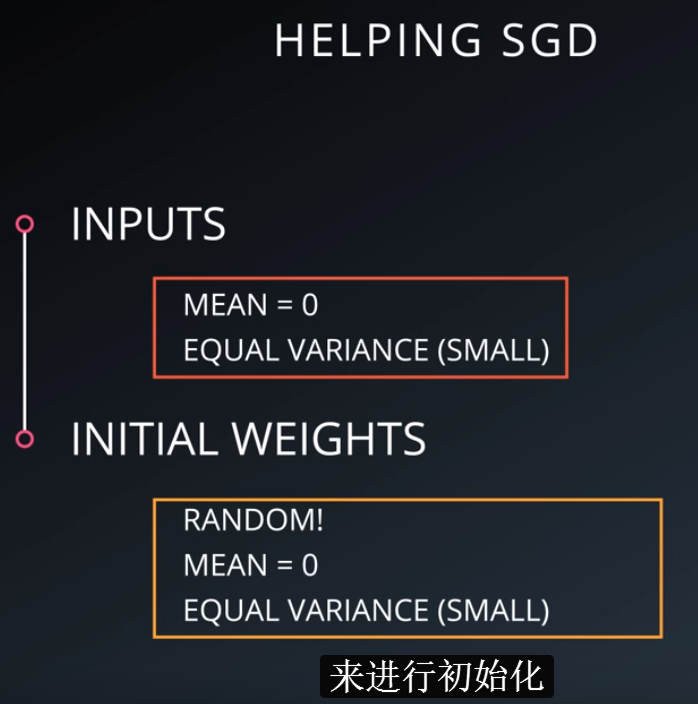
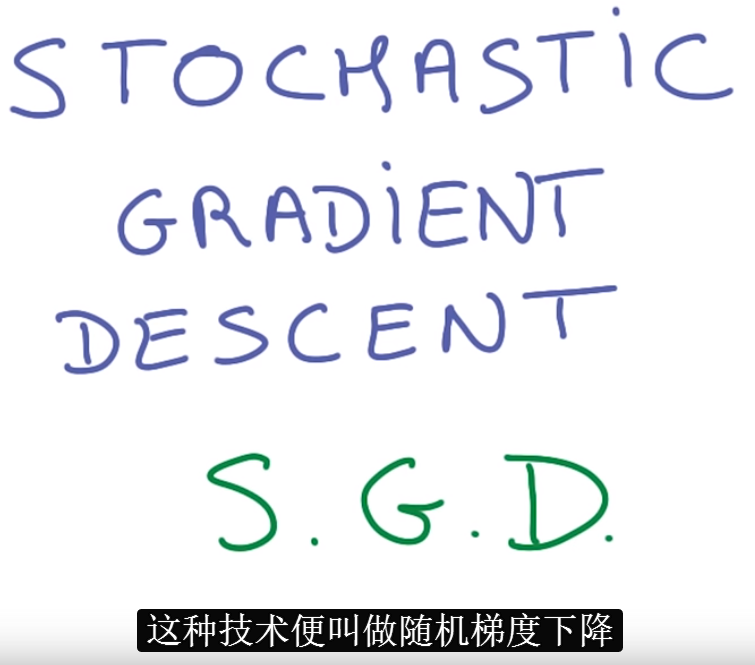
**练习**

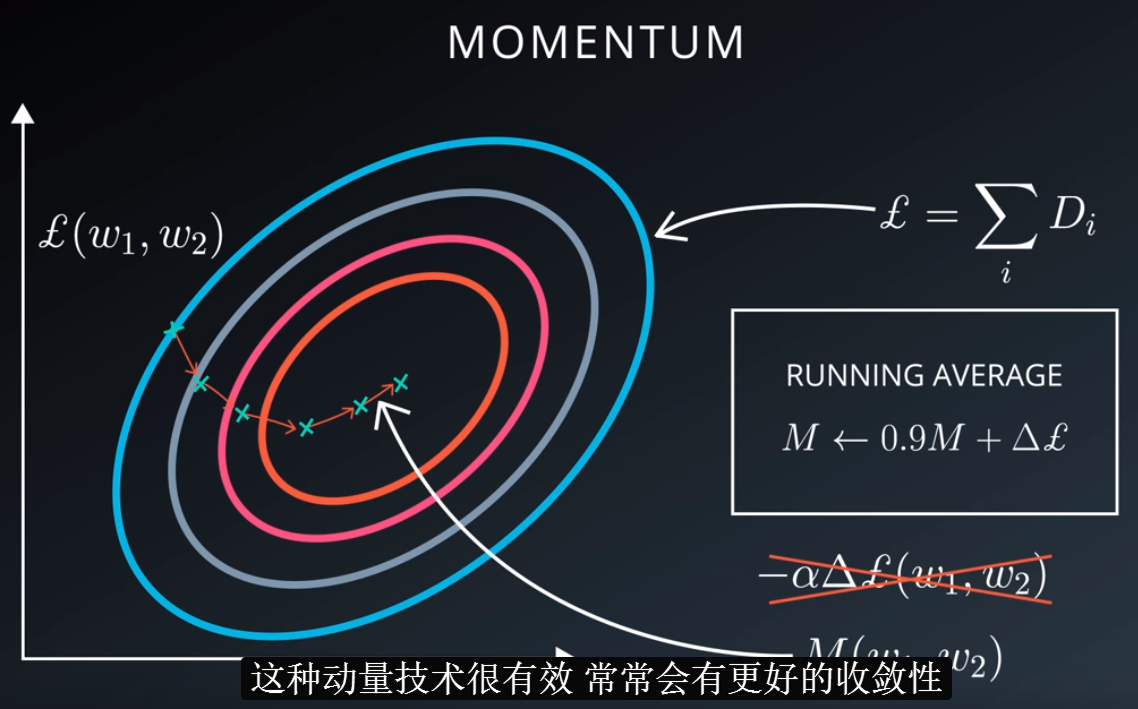
用softmax\_data 和 one\_hot\_encod\_label打印交叉熵

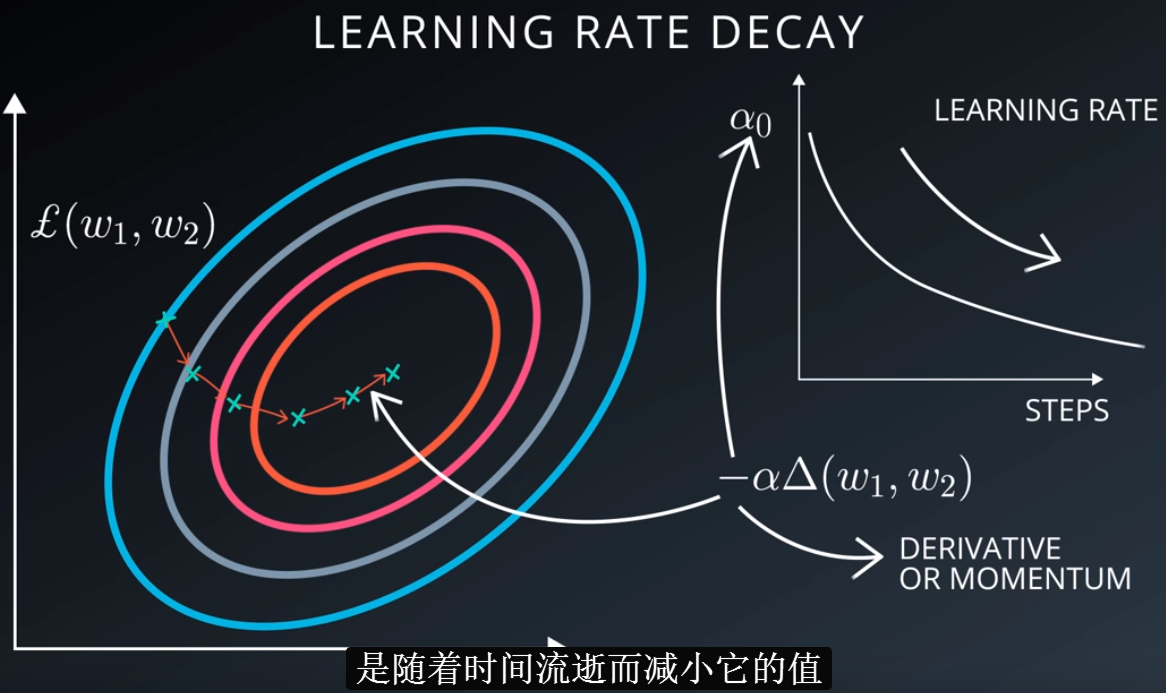


用随机选取的小片段算loss和梯度方向以减少计算量











## Mini-batching

在这一节，你将了解什么是 mini-batching，以及如何在 TensorFlow 里应用它。

Mini-batching是一个一次训练数据集的一小部分，而不是整个训练集的技术。它可以是的内存较小，不能同时训练整个数据集的电脑也可以训练模型。

Mini-batching 从运算角度来说是低效的，因为你不能在所有样本中计算 loss。但是这点小代价也比根本不能跑模型要划算。

它跟随机梯度下降（SGD）和在一起用也很有帮助。办法是在每一代训练之前，对数据进行随机混洗，然后创建 mini-batches ，对每一个 mini-batch，用梯度下降训练网络权重。因为这些 batches 是随机的，你其实是在对的每一个 batch 做随机梯度下降（SGD）。

让我们看看你的机器能否训练出 MNIST 数据集的权重和偏置项。

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

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

n\_input = 784 *# MNIST data input (img shape: 28\*28)*

n\_classes = 10 *# MNIST total classes (0-9 digits)*

*# Import MNIST data*

mnist = input\_data.read\_data\_sets('/datasets/ud730/mnist', one\_hot=**True**)

*# The features are already scaled and the data is shuffled*

train\_features = mnist.train.images

test\_features = mnist.test.images

train\_labels = mnist.train.labels.astype(np.float32)

test\_labels = mnist.test.labels.astype(np.float32)

*# Weights & bias*

weights = tf.Variable(tf.random\_normal([n\_input, n\_classes]))

bias = tf.Variable(tf.random\_normal([n\_classes]))

输入、权重和偏置项总共的内存空间需求是 174 MB，并不是太多。你可以在CPU和GPU上训练整个数据集。

但将来你要用到的数据集可能是以G来衡量，甚至更多。你可以买更多的内存，但是会很贵。例如一个12 GB显存容量的 Titan X GPU 会超过1000美金。所以，为了在你自己机器上跑大模型，你需要学会用 mini-batching。

Let's look at how you implement mini-batching in TensorFlow. 让我们看下如何在 TensorFlow 下实现 mini-batching

## TensorFlow Mini-batching

要使用 mini-batching，你首先要把你的数据集分成 batches。

不幸的是，有时候不可能把数据完全分割成相同数量的 batch。例如有 1000 个数据点，你想每个 batch 有 128 个数据。但是1000 无法被 128 整除。你得到的结果是 7 batch，每个128个数据点，一个 batch 有 104个数据点。(7\*128 + 1\*104 = 1000)

batch里面的数据点数量会不同的情况下，你需要利用 TensorFlow 的**[tf.placeholder()](https://www.tensorflow.org/api_docs/python/tf/placeholder" \t "_blank)** 函数来接收这些不同的 batch。

继续上述例子，如果每个样本有n\_input = 784特征，n\_classes = 10个可能的标签，features的维度应该是[None, n\_input]，labels的维度是 [None, n\_classes]。

*# Features and Labels*

features = tf.placeholder(tf.float32, [**None**, n\_input])

labels = tf.placeholder(tf.float32, [**None**, n\_classes])

None在这里做什么用呢？

None 维度在这里是一个 batch size 的占位符。在运行时，TensorFlow 会接收任何大于 0 的 batch size。

回到之前的例子，这个设置可以让你把features 和 labels给到模型。无论 batch 中包含128，还是104个数据点。

现在你知道了基本概念，让我们学习如何来实现 mini-batching

### 问题三

对features 和 labels实现一个 batches 函数。这个函数返回每个有最大batch\_size数据点的 batch。下面有例子来说明一个示例batches函数的输出是什么。

*# 4 个特征*

example\_features = [

['F11','F12','F13','F14'],

['F21','F22','F23','F24'],

['F31','F32','F33','F34'],

['F41','F42','F43','F44']]

*# 4 个label*

example\_labels = [

['L11','L12'],

['L21','L22'],

['L31','L32'],

['L41','L42']]

example\_batches = batches(3, example\_features, example\_labels)

example\_batches 变量如下：

[

*# 分2 个 batches:*

*# 第一个 batch 的 size 是 3*

*# 第二个 batch 的 size 是 1*

[

*# size 为 3 的第一个Batch*

[

*# 3 个特征样本*

*# 每个样本有四个特征*

['F11', 'F12', 'F13', 'F14'],

['F21', 'F22', 'F23', 'F24'],

['F31', 'F32', 'F33', 'F34']

], [

*# 3 个标签样本*

*# 每个标签有两个label.*

['L11', 'L12'],

['L21', 'L22'],

['L31', 'L32']

]

], [

*# size 为 1 的第二个 Batch*

*# 因为 batch size 是 3。所以四个样品中只有一个在这里。*

[

*# 1 一个样本特征*

['F41', 'F42', 'F43', 'F44']

], [

*# 1 个label*

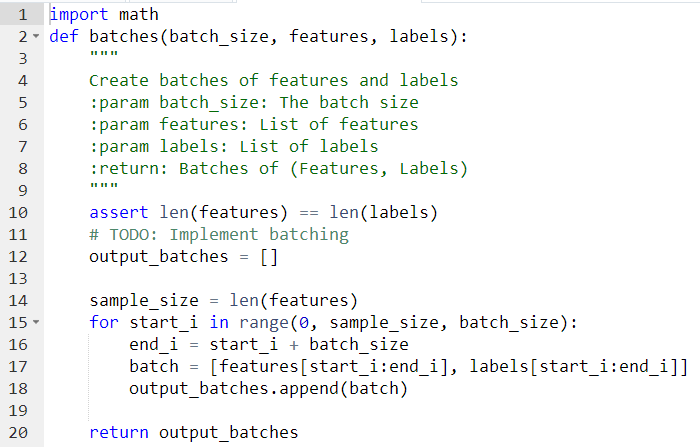
['L41', 'L42']

]

]

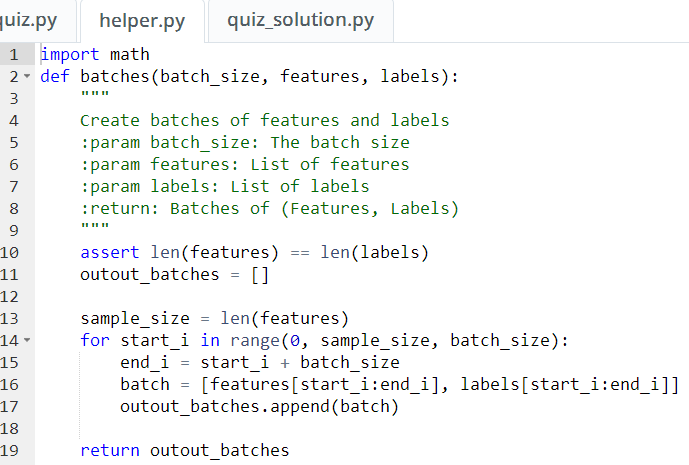
]

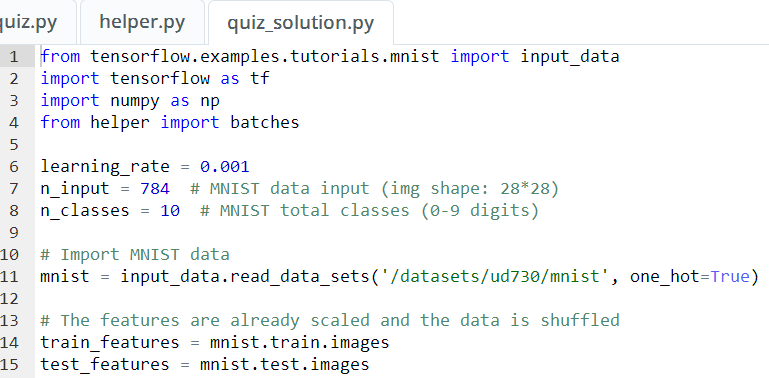
在下面 "quiz.py" 文件里实现 batches 函数

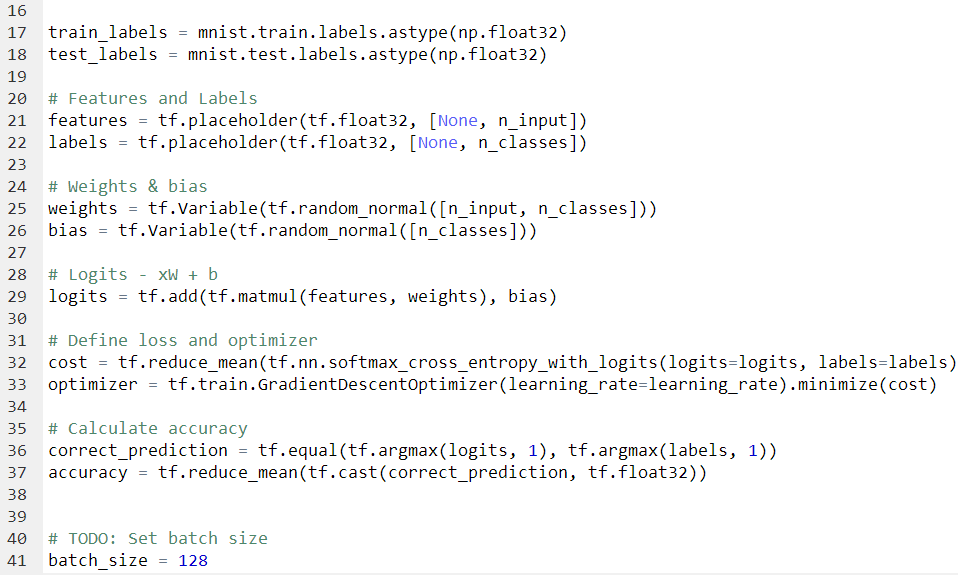


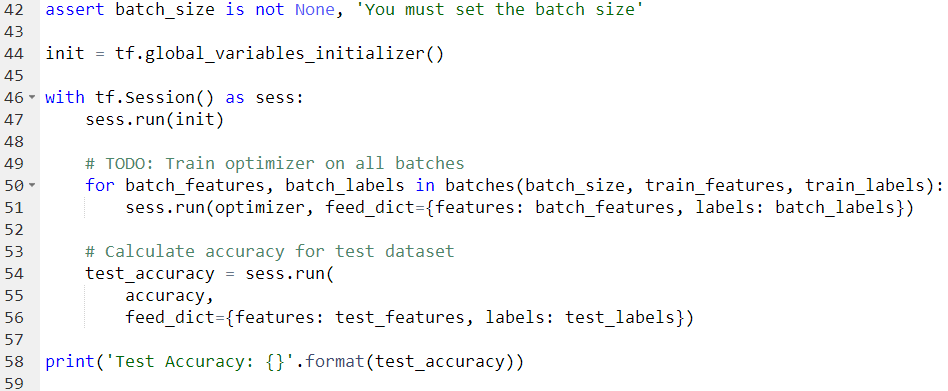
让我们用 mini-batching 来把 MNIST 特征和目标分批给到线性模型。

设定 batch size，用 batches 函数来分配所有数据。建议的 batch size 是128，你也可以根据自己内存大小来改变它。









Test Accuracy: 0.11720000207424164

虽然准确度不高，但是你或许知道训练集不止用来训练一次。你可以用数据集多次训练一个模型。下一章节我们会讨论"epochs"这个话题。

## Epochs （代）

一个 epoch（代）是指整个数据集正向反向训练一次。它被用来提示模型的准确率并且不需要额外数据。本节我们将讲解 TensorFlow 里的 epochs，以及如何选择正确的 epochs。

下面是训练一个模型 10 代的 TensorFlow 代码

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

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

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

**from** helper **import** batches *# Helper function created in Mini-batching section*

**def** **print\_epoch\_stats**(epoch\_i, sess, last\_features, last\_labels):

"""

Print cost and validation accuracy of an epoch

"""

current\_cost = sess.run(

cost,

feed\_dict={features: last\_features, labels: last\_labels})

valid\_accuracy = sess.run(

accuracy,

feed\_dict={features: valid\_features, labels: valid\_labels})

print('Epoch: {:<4} - Cost: {:<8.3} Valid Accuracy: {:<5.3}'.format(

epoch\_i,

current\_cost,

valid\_accuracy))

n\_input = 784 *# MNIST data input (img shape: 28\*28)*

n\_classes = 10 *# MNIST total classes (0-9 digits)*

*# Import MNIST data*

mnist = input\_data.read\_data\_sets('/datasets/ud730/mnist', one\_hot=**True**)

*# The features are already scaled and the data is shuffled*

train\_features = mnist.train.images

valid\_features = mnist.validation.images

test\_features = mnist.test.images

train\_labels = mnist.train.labels.astype(np.float32)

valid\_labels = mnist.validation.labels.astype(np.float32)

test\_labels = mnist.test.labels.astype(np.float32)

*# Features and Labels*

features = tf.placeholder(tf.float32, [**None**, n\_input])

labels = tf.placeholder(tf.float32, [**None**, n\_classes])

*# Weights & bias*

weights = tf.Variable(tf.random\_normal([n\_input, n\_classes]))

bias = tf.Variable(tf.random\_normal([n\_classes]))

*# Logits - xW + b*

logits = tf.add(tf.matmul(features, weights), bias)

*# Define loss and optimizer*

learning\_rate = tf.placeholder(tf.float32)

cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logits, labels=labels))

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=learning\_rate).minimize(cost)

*# Calculate accuracy*

correct\_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(labels, 1))

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

init = tf.global\_variables\_initializer()

batch\_size = 128

epochs = 10

learn\_rate = 0.001

train\_batches = batches(batch\_size, train\_features, train\_labels)

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

sess.run(init)

*# Training cycle*

**for** epoch\_i **in** range(epochs):

*# Loop over all batches*

**for** batch\_features, batch\_labels **in** train\_batches:

train\_feed\_dict = {

features: batch\_features,

labels: batch\_labels,

learning\_rate: learn\_rate}

sess.run(optimizer, feed\_dict=train\_feed\_dict)

*# Print cost and validation accuracy of an epoch*

print\_epoch\_stats(epoch\_i, sess, batch\_features, batch\_labels)

*# Calculate accuracy for test dataset*

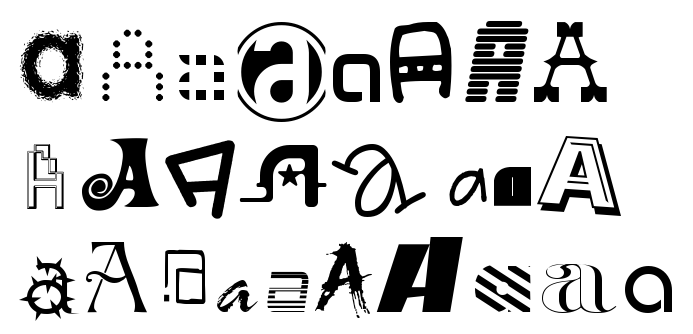
test\_accuracy = sess.run(

accuracy,

feed\_dict={features: test\_features, labels: test\_labels})

print('Test Accuracy: {}'.format(test\_accuracy))

# TensorFlow Neural Network Lab

**[](http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html)**

# TensorFlow Lab

我们为你准备了一个 Jupyter notebook，指引你在 TensorFlow 里创建一个单层的神经网络。你将实现数据归一化，用 TensorFlow 来创建一个网络并训练它。

## 获取 notebook

notebook 和所有相关的文件都可以在我们的 [**GitHub repo**](https://github.com/udacity/deep-learning) 里获得。你可以 clone 这个目录，或者下载 Zip 文件。

用 Git clone：

git clone https://github.com/udacity/deep-learning.git