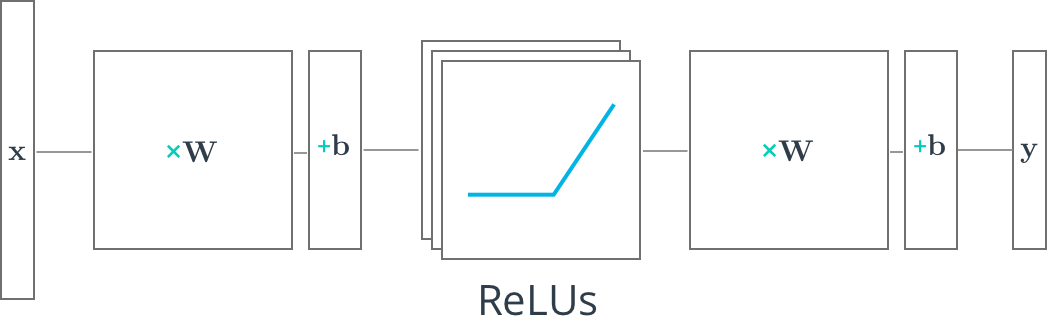
Udacity ML part 4 深度神经网络



**多层神经网络**

在本课程中，你会学到如何用 TensorFlow 构建多层神经网络。之前你应该了解，在网络里面添加一个隐藏层，可以让它构建更复杂的模型。而且，在隐藏层用非线性激活函数可以让它对非线性函数建模。

一个常用的非线性函数叫 **[ReLU（rectified linear unit）](https://en.wikipedia.org/wiki/Rectifier_(neural_networks" \t "_blank)**)。ReLU 函数对所有负的输入，返回 0；所有 *x*>0 的输入，返回 *x*。

接下来你会看到如何在 TensorFlow 里实现一个 ReLU 隐藏层。

# TensorFlow ReLUs

TensorFlow 提供了 ReLU 函数 **[tf.nn.relu()](https://www.tensorflow.org/api_docs/python/tf/nn/relu" \t "_blank)**，如下所示：

*# 隐藏层用 ReLU 作为激活函数*

hidden\_layer = tf.add(tf.matmul(features, hidden\_weights), hidden\_biases)

hidden\_layer = tf.nn.relu(hidden\_layer)

output = tf.add(tf.matmul(hidden\_layer, output\_weights), output\_biases)

上面的代码把[**tf.nn.relu()**](https://www.tensorflow.org/api_docs/python/tf/nn/relu) 放到隐藏层，就像开关一样把负权重关掉了。添加像输出层这样额外的层在激活函数后，就把模型变成了非线性函数。这个非线性的特征使得网络可以解决更复杂的问题。

# TensorFlow 中的深度神经网络

你已经学过了如何用 TensorFlow 构建一个逻辑分类器。现在你会学到如何用逻辑分类器来构建一个深度神经网络。

## 详细指导

接下来我们看看如何用 TensorFlow 来构建一个分类器来对 MNIST 数字进行分类。如果你要在自己电脑上跑这个代码，文件在[**这儿**](https://d17h27t6h515a5.cloudfront.net/topher/2017/February/58a61a3a_multilayer-perceptron/multilayer-perceptron.zip)。你可以在[**Aymeric Damien 的 GitHub repository**](https://github.com/aymericdamien/TensorFlow-Examples)里找到更多的 TensorFlow 的例子。

## 代码

### TensorFlow MNIST

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

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

你可以使用 TensorFlow 提供的 MNIST 数据集，他把分批和独热码都帮你处理好了。

### 参数学习

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

*# 参数*

learning\_rate = 0.001

training\_epochs = 20

batch\_size = 128 *# 如果没有足够内存，可以降低 batch size*

display\_step = 1

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

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

这里的关注点是多层神经网络的架构，不是调参，所以这里直接给你了学习的参数。

### 隐藏层参数

n\_hidden\_layer = 256 *# 层特征数量*

n\_hidden\_layer 决定了神经网络隐藏层的大小。也被称作层的宽度。

### 权重和偏置项

*# 层权重和偏置项的储存*

weights = {

'hidden\_layer': tf.Variable(tf.random\_normal([n\_input, n\_hidden\_layer])),

'out': tf.Variable(tf.random\_normal([n\_hidden\_layer, n\_classes]))

}

biases = {

'hidden\_layer': tf.Variable(tf.random\_normal([n\_hidden\_layer])),

'out': tf.Variable(tf.random\_normal([n\_classes]))

}

深度神经网络有多个层，每个层有自己的权重和偏置项。'hidden\_layer'的权重和偏置项只对隐藏层， 'out'的权重和偏置项只对输出层。如果神经网络比这更深，那每一层都有权重和偏置项。

### 输入

*# tf Graph input*

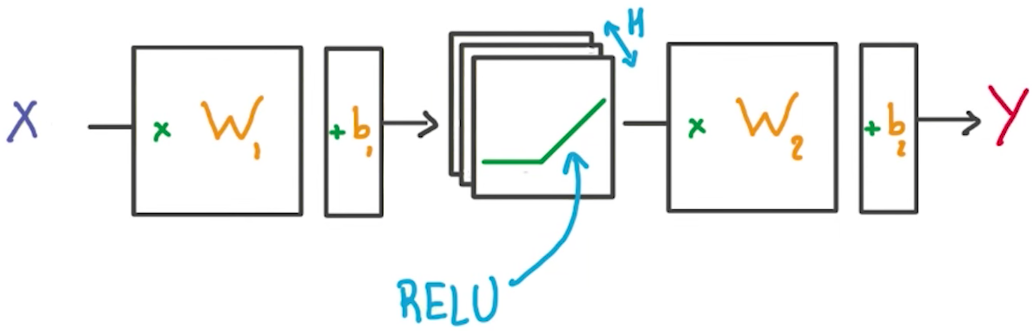
x = tf.placeholder("float", [**None**, 28, 28, 1])

y = tf.placeholder("float", [**None**, n\_classes])

x\_flat = tf.reshape(x, [-1, n\_input])

MNIST 数据集是由 28px 乘 28px 单**[通道](https://en.wikipedia.org/wiki/Channel_(digital_image%29" \t "_blank)**图片组成。tf.reshape()函数把 28px 乘 28px 的矩阵换成了 784px by 1px 的向量x。

### 多层感知器



*# ReLU作为隐藏层激活函数*

layer\_1 = tf.add(tf.matmul(x\_flat, weights['hidden\_layer']),\

biases['hidden\_layer'])

layer\_1 = tf.nn.relu(layer\_1)

*# 输出层的线性激活函数*

logits = tf.add(tf.matmul(layer\_1, weights['out']), biases['out'])

你之前已经见过 tf.add(tf.matmul(x\_flat, weights['hidden\_layer']), biases['hidden\_layer'])，也就是 xw + b。把线性函数与ReLU组合在一起，给你一个两层网络。

### 优化器 Optimizer

*# Define loss and optimizer*

cost = tf.reduce\_mean(\

tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logits, labels=y))

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

.minimize(cost)

这跟 Intro to TensorFlow lab 里用到的优化技巧一样。

### Session

*# Initializing the variables*

init = tf.global\_variables\_initializer()

*# Launch the graph*

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

sess.run(init)

*# Training cycle*

**for** epoch **in** range(training\_epochs):

total\_batch = int(mnist.train.num\_examples/batch\_size)

*# Loop over all batches*

**for** i **in** range(total\_batch):

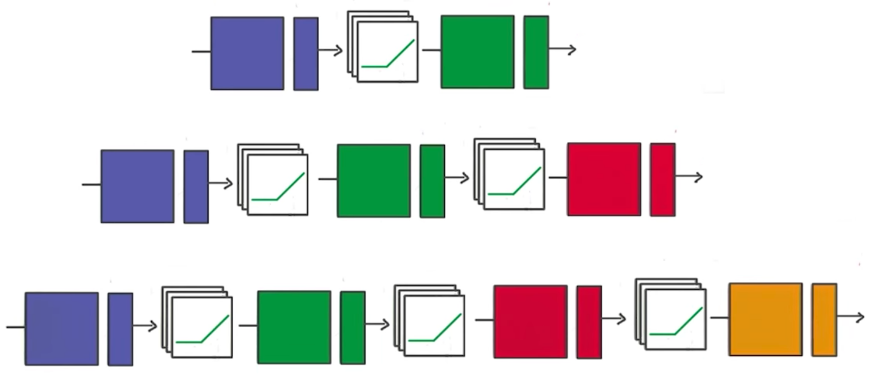
batch\_x, batch\_y = mnist.train.next\_batch(batch\_size)

*# Run optimization op (backprop) and cost op (to get loss value)*

sess.run(optimizer, feed\_dict={x: batch\_x, y: batch\_y})

TensorFlow 中的 MNIST 库提供了分批接收数据的能力。调用mnist.train.next\_batch()函数返回训练数据的一个子集。

## 深度神经网络



就是这样！从一层到两层很简单。向网络中添加更多层，可以让你解决更复杂的问题。在下面的视频中，你将了解改变层的数量会对你的网络有怎样的影响。

# 保存和读取 TensorFlow 模型

训练一个模型的时间很长。但是你一旦关闭了 TensorFlow session，你所有训练的权重和偏置项都丢失了。如果你计划在之后重新使用这个模型，你需要重新训练！

幸运的是，TensorFlow 可以让你通过一个叫 **[tf.train.Saver](https://www.tensorflow.org/api_docs/python/tf/train/Saver" \t "_blank)** 的类把你的进程保存下来。这个类可以把任何[**tf.Variable**](https://www.tensorflow.org/api_docs/python/tf/Variable) 存到你的文件系统。

## 保存变量

让我们通过一个简单地例子来保存 weights 和 bias Tensors。第一个例子你只是存两个变量，后面会教你如何把一个实际模型的所有权重保存下来。

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

*# The file path to save the data*

*# 文件保存路径*

save\_file = './model.ckpt'

*# Two Tensor Variables: weights and bias*

*# 两个 Tensor 变量：权重和偏置项*

weights = tf.Variable(tf.truncated\_normal([2, 3]))

bias = tf.Variable(tf.truncated\_normal([3]))

*# Class used to save and/or restore Tensor Variables*

*# 用来存取 Tensor 变量的类*

saver = tf.train.Saver()

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

*# Initialize all the Variables*

*# 初始化所有变量*

sess.run(tf.global\_variables\_initializer())

*# Show the values of weights and bias*

*# 显示变量和权重*

print('Weights:')

print(sess.run(weights))

print('Bias:')

print(sess.run(bias))

*# Save the model*

*# 保存模型*

saver.save(sess, save\_file)

Weights:

[[-0.97990924 1.03016174 0.74119264]

[-0.82581609 -0.07361362 -0.86653847]]

Bias:

[ 1.62978125 -0.37812829 0.64723819]

weights 和 bias Tensors 用 **[tf.truncated\_normal()](https://www.tensorflow.org/api_docs/python/tf/truncated_normal" \t "_blank)** 函数设定了随机值。用 **[tf.train.Saver.save()](https://www.tensorflow.org/api_docs/python/tf/train/Saver" \l "save" \t "_blank)** 函数把这些值被保存在save\_file 位置，命名为 "model.ckpt"，（".ckpt" 扩展名表示"checkpoint"）。

如果你使用 TensorFlow 0.11.0RC1 或者更新版，一个叫做 "model.ckpt.meta" 的文件也会生成。它包含了 TensorFlow graph。

## 加载变量

现在这些变量已经存好了，让我们把它们加载到新模型里。

*# Remove the previous weights and bias*

*# 移除之前的权重和偏置项*

tf.reset\_default\_graph()

*# Two Variables: weights and bias*

*# 两个变量：权重和偏置项*

weights = tf.Variable(tf.truncated\_normal([2, 3]))

bias = tf.Variable(tf.truncated\_normal([3]))

*# Class used to save and/or restore Tensor Variables*

*# 用来存取 Tensor 变量的类*

saver = tf.train.Saver()

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

*# Load the weights and bias*

*# 加载权重和偏置项*

saver.restore(sess, save\_file)

*# Show the values of weights and bias*

*# 显示权重和偏置项*

print('Weight:')

print(sess.run(weights))

print('Bias:')

print(sess.run(bias))

Weights:

[[-0.97990924 1.03016174 0.74119264]

[-0.82581609 -0.07361362 -0.86653847]]

Bias:

[ 1.62978125 -0.37812829 0.64723819]

你注意到，你依然需要在 Python 中创建 weights 和 bias。**[tf.train.Saver.restore()](https://www.tensorflow.org/api_docs/python/tf/train/Saver" \l "restore" \t "_blank)** 函数把保存的数据加载到weights 和 bias 当中。

因为 **[tf.train.Saver.restore()](https://www.tensorflow.org/api_docs/python/tf/train/Saver" \l "restore" \t "_blank)** 设定了 TensorFlow 变量，这里你不需要调用[**tf.global\_variables\_initializer()**](https://www.tensorflow.org/api_docs/python/tf/global_variables_initializer)了。

## 保存一个训练好的模型

让我们看看如何训练一个模型并保存它的权重。

从一个模型开始：

*# Remove previous Tensors and Operations*

*# 移除之前的 Tensors 和运算*

tf.reset\_default\_graph()

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

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

learning\_rate = 0.001

n\_input = 784 *# MNIST 数据输入 (图片尺寸: 28\*28)*

n\_classes = 10 *# MNIST 总计类别 (数字 0-9)*

*# Import MNIST data*

*# 加载 MNIST 数据*

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

*# Features and Labels*

*# 特征和标签*

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

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

*# Weights & bias*

*# 权重和偏置项*

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*

*# 定义损失函数和优化器*

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*

*# 计算准确率*

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

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

让我们训练模型并保存权重：

**import** math

save\_file = './train\_model.ckpt'

batch\_size = 128

n\_epochs = 100

saver = tf.train.Saver()

*# Launch the graph*

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

sess.run(tf.global\_variables\_initializer())

*# Training cycle*

**for** epoch **in** range(n\_epochs):

total\_batch = math.ceil(mnist.train.num\_examples / batch\_size)

*# Loop over all batches*

**for** i **in** range(total\_batch):

batch\_features, batch\_labels = mnist.train.next\_batch(batch\_size)

sess.run(

optimizer,

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

*# Print status for every 10 epochs*

**if** epoch % 10 == 0:

valid\_accuracy = sess.run(

accuracy,

feed\_dict={

features: mnist.validation.images,

labels: mnist.validation.labels})

print('Epoch {:<3} - Validation Accuracy: {}'.format(

epoch,

valid\_accuracy))

*# Save the model*

saver.save(sess, save\_file)

print('Trained Model Saved.')

Epoch 0 - Validation Accuracy: 0.06859999895095825

Epoch 10 - Validation Accuracy: 0.20239999890327454

Epoch 20 - Validation Accuracy: 0.36980000138282776

Epoch 30 - Validation Accuracy: 0.48820000886917114

Epoch 40 - Validation Accuracy: 0.5601999759674072

Epoch 50 - Validation Accuracy: 0.6097999811172485

Epoch 60 - Validation Accuracy: 0.6425999999046326

Epoch 70 - Validation Accuracy: 0.6733999848365784

Epoch 80 - Validation Accuracy: 0.6916000247001648

Epoch 90 - Validation Accuracy: 0.7113999724388123

Trained Model Saved.

## 加载训练好的模型

让我们从磁盘中加载权重和偏置项，验证测试集准确率

saver = tf.train.Saver()

*# Launch the graph*

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

saver.restore(sess, save\_file)

test\_accuracy = sess.run(

accuracy,

feed\_dict={features: mnist.test.images, labels: mnist.test.labels})

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

Test Accuracy: 0.7229999899864197

就是这样！你现在知道如何保存再加载一个 TensorFlow 的训练模型了。下一章节让我们看看如何把权重和偏置项加载到已经修改的模型中。

# 把权重和偏置项加载到新模型中

很多时候你想调整，或者说“微调”一个你已经训练并保存了的模型。但是，把保存的变量直接加载到已经修改过的模型会产生错误。让我们看看如何解决这个问题。

## 命名报错

TensorFlow 对 Tensor 和计算使用一个叫 name 的字符串辨识器，如果名称没有给，TensorFlow 会自动创建一个。TensorFlow 会把第一个节点命名为 <Type>，把后续的命名为<Type>\_<number>。让我们看看这对加载一个有不同顺序权重和偏置项的模型有哪些影响：

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

*# Remove the previous weights and bias*

*# 移除先前的权重和偏置项*

tf.reset\_default\_graph()

save\_file = 'model.ckpt'

*# Two Tensor Variables: weights and bias*

*# 两个 Tensor 变量：权重和偏置项*

weights = tf.Variable(tf.truncated\_normal([2, 3]))

bias = tf.Variable(tf.truncated\_normal([3]))

saver = tf.train.Saver()

*# Print the name of Weights and Bias*

*# 打印权重和偏置项的名字*

print('Save Weights: {}'.format(weights.name))

print('Save Bias: {}'.format(bias.name))

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

sess.run(tf.global\_variables\_initializer())

saver.save(sess, save\_file)

*# Remove the previous weights and bias*

*# 移除之前的权重和偏置项*

tf.reset\_default\_graph()

*# Two Variables: weights and bias*

*# 两个变量：权重和偏置项*

bias = tf.Variable(tf.truncated\_normal([3]))

weights = tf.Variable(tf.truncated\_normal([2, 3]))

saver = tf.train.Saver()

*# Print the name of Weights and Bias*

*# 打印权重和偏置项的名字*

print('Load Weights: {}'.format(weights.name))

print('Load Bias: {}'.format(bias.name))

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

*# Load the weights and bias - ERROR*

*# 加载权重和偏置项 - 报错*

saver.restore(sess, save\_file)

上述代码会有下列输出

Save Weights: Variable:0

Save Bias: Variable\_1:0

Load Weights: Variable\_1:0

Load Bias: Variable:0

...

InvalidArgumentError (see above for traceback): Assign requires shapes of both tensors to match.

...

你注意到，weights 和 bias 的 name 属性与你保存的模型不同。这是为什么代码报“Assign requires shapes of both tensors to match”这个错误。saver.restore(sess, save\_file) 代码试图把权重数据加载到bias里，把偏置项数据加载到 weights里。

与其让 TensorFlow 来设定 name 属性，不如让我们来手动设定：

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

tf.reset\_default\_graph()

save\_file = 'model.ckpt'

*# Two Tensor Variables: weights and bias*

weights = tf.Variable(tf.truncated\_normal([2, 3]), name='weights\_0')

bias = tf.Variable(tf.truncated\_normal([3]), name='bias\_0')

saver = tf.train.Saver()

*# Print the name of Weights and Bias*

print('Save Weights: {}'.format(weights.name))

print('Save Bias: {}'.format(bias.name))

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

sess.run(tf.global\_variables\_initializer())

saver.save(sess, save\_file)

*# Remove the previous weights and bias*

tf.reset\_default\_graph()

*# Two Variables: weights and bias*

bias = tf.Variable(tf.truncated\_normal([3]), name='bias\_0')

weights = tf.Variable(tf.truncated\_normal([2, 3]) ,name='weights\_0')

saver = tf.train.Saver()

*# Print the name of Weights and Bias*

print('Load Weights: {}'.format(weights.name))

print('Load Bias: {}'.format(bias.name))

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

*# Load the weights and bias - No Error*

saver.restore(sess, save\_file)

print('Loaded Weights and Bias successfully.')

Save Weights: weights\_0:0

Save Bias: bias\_0:0

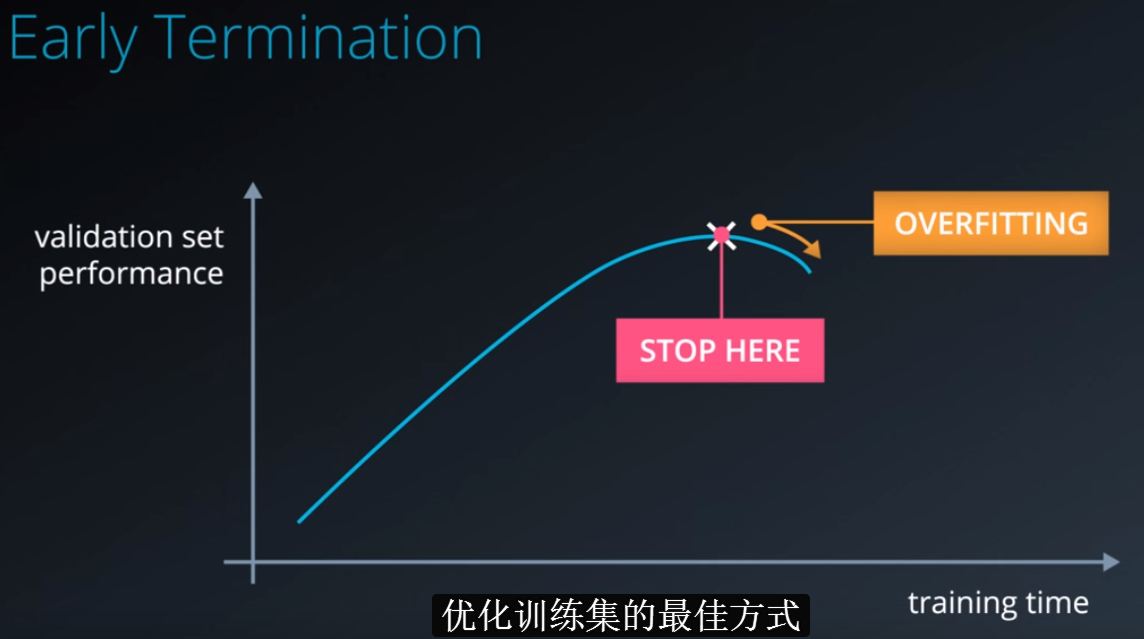
Load Weights: weights\_0:0

Load Bias: bias\_0:0

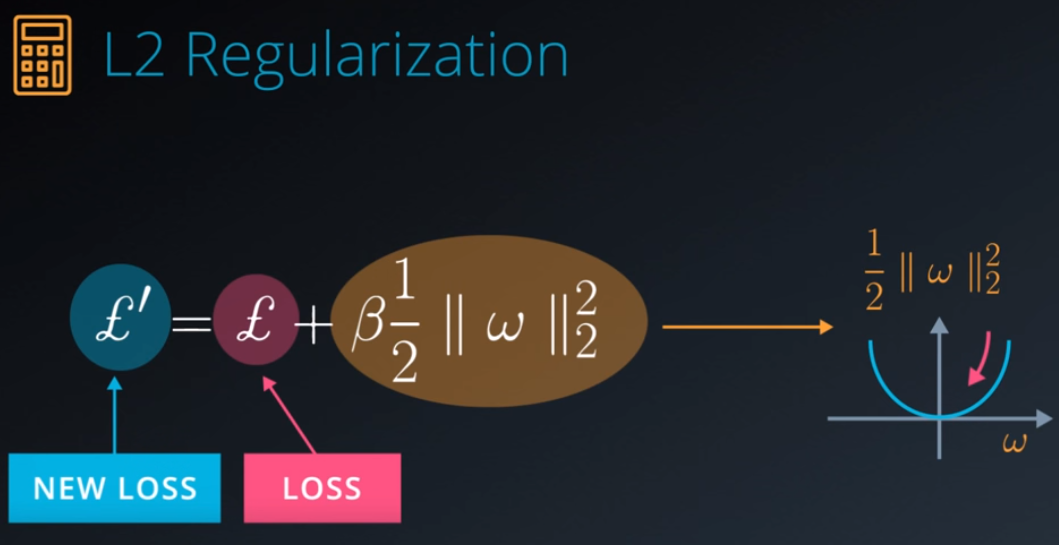
Loaded Weights and Bias successfully.

这次没问题！Tensor 名称批评正确，数据正确加载。

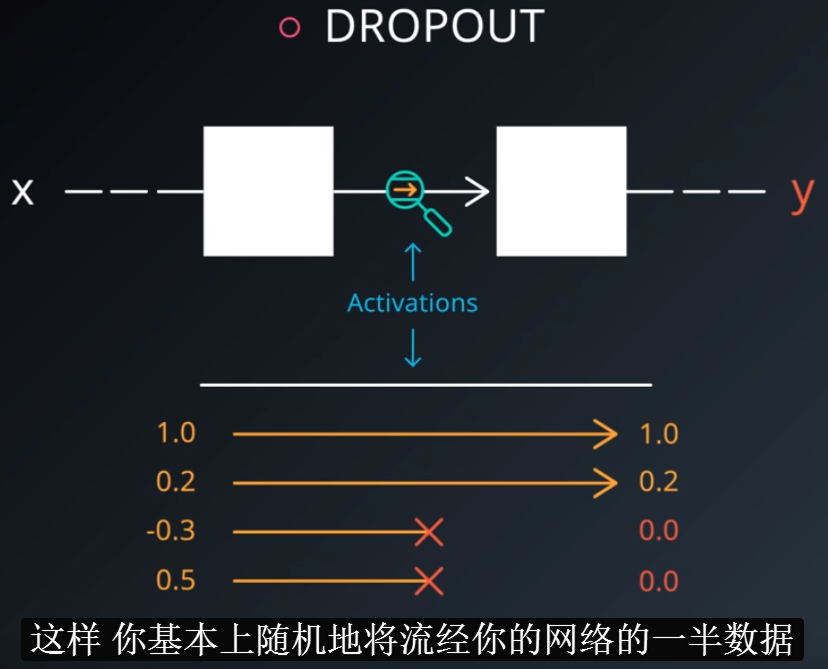
消除过拟合的方法：



另一种方法是正则化



另一种防止过拟合的方式是dropout，由Hinton发明。 如果不想用dropout，也可以通过增大模型解决过拟合。



# TensorFlow Dropout

图 1：来自论文 "Dropout: A Simple Way to Prevent Neural Networks from
Overfitting" (https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf)

图 1：来自论文 "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" ([**https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf**](https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf))

Dropout 是一个降低过拟合的正则化技术。它在网络中暂时的丢弃一些单位([**神经元**](https://en.wikipedia.org/wiki/Artificial_neuron))，以及它们的前后连接，图一是一个 dropout 如何工作的示意图

TensorFlow 提供了一个 **[tf.nn.dropout()](https://www.tensorflow.org/api_docs/python/tf/nn/dropout" \t "_blank)** 函数，你可以用来实现 dropout。

让我们来看一个如何使用 **[tf.nn.dropout()](https://www.tensorflow.org/api_docs/python/tf/nn/dropout" \t "_blank)**的例子。

keep\_prob = tf.placeholder(tf.float32) *# probability to keep units*

hidden\_layer = tf.add(tf.matmul(features, weights[0]), biases[0])

hidden\_layer = tf.nn.relu(hidden\_layer)

hidden\_layer = tf.nn.dropout(hidden\_layer, keep\_prob)

logits = tf.add(tf.matmul(hidden\_layer, weights[1]), biases[1])

上面的代码展示了如何在神经网络中应用dropout

[**tf.nn.dropout()**](https://www.tensorflow.org/api_docs/python/tf/nn/dropout)函数有两个参数：

1. hidden\_layer：你要应用 dropout 的 tensor
2. keep\_prob：任何一个给定单位的留存率（**没有**丢弃的）

keep\_prob 可以让你调整 drop 单位的数量。为了补偿被丢弃的单位，[**tf.nn.dropout()**](https://www.tensorflow.org/api_docs/python/tf/nn/dropout) 把所有保留下来的单位（**没有**丢弃的）乘 1/keep\_prob

在训练时，一个好的keep\_prob初始值是0.5。

在测试时，把 keep\_prob 值设为1.0 ，这样保留所有的单位，最大化模型的能力。