# TensorFlow 基础使用方法

冯浩哲

医疗数据兴趣小组

2017 年 3 月 28 日

# 内容梗概

- TensorFlow 的基本语法与数据读入规范
- 基础神经网络与卷积神经网络的搭建实例
- 参考文献

# TensorFlow 的基本语法与数据读入规范

## 勾股定理

直角三角形的斜边的平方等于两直角边的平方和。可以用符号语言表述为：设直角三角形 ABC，其中 $\angle C = 90^\circ$ 则有

$$AB^2 = BC^2 + AC^2 \tag{1}$$

- Placeholder
  - Placeholder 是用于输入层的数据格式，基本使用格式如下：

```
import tensorflow as tf
image = tf.placeholder(tf.float32, [None, 28,28,3])
#或者
image = tf.placeholder(tf.float32, [None, 784])
```

  - 输入层数据使用的是 TensorFlow 的格式，因此一般不能直接用 python 的赋值语句进行赋值。如果想给 x 赋值的话，需要用字典：

```
{image:data}
```

  - 这里有个细节是 data 的输入格式，data 的格式可以是 list 格式，也可以是 numpy 下的 array 格式。

# TensorFlow 的基本语法与数据读入规范

- Placeholder
  - 总结一下图片读入，我们把一张图片转换成一个矩阵，然后再把这个矩阵拉直成 1 维向量，放在一个文件里面。
  - n 张图片就是 n 条向量，然后用 python 的 open 等命令从文件里一行一行读数据，这些数据放在一个二维列表里，然后用字典赋值给用 placeholder 定义的输入层变量。
  - 一般 placeholder 的定义用的是：

```
image = tf.placeholder(tf.float32, [None, 784])
#然后再用reshape重塑大小
x_image = tf.reshape(image, [-1,28,28,1])
```

  - reshape 命令中第一个-1 表示我们要读多少张图片，-1 是缺省值。第二和第三个分别是图片的 width 和 height，第四个是图片的颜色数目，黑白为 1，RGB 为 3。

# TensorFlow 的基本语法与数据读入规范

- Variable
  - Variable 是用于中间层的输入格式，在神经网络中一般用于定义每一层神经网络的输入权重结构，基本使用格式如下：

```
W = tf.Variable(tf.zeros([784, 10]))#设置一层有10个神经元，每个神经元有784个输入权重的神经网络
b = tf.Variable(tf.zeros([10]))#设置神经元的截距，10个神经元有10个截距
```

- Session
  - Session 是程序运行过程的语句，它指导了程序下一步要做什么。使用格式如下：

```
sess=tf.Session()#这表明以后用sess变量来执行步骤，语句为sess.run()
sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels})
#我自己简单理解为 sess.run(要计算的东西,feed_dict=用于计算的数据)
```

# TensorFlow 的基本语法与数据读入规范

- Variable
  - Variable 是用于中间层的输入格式，在神经网络中一般用于定义每一层神经网络的输入权重结构，基本使用格式如下：

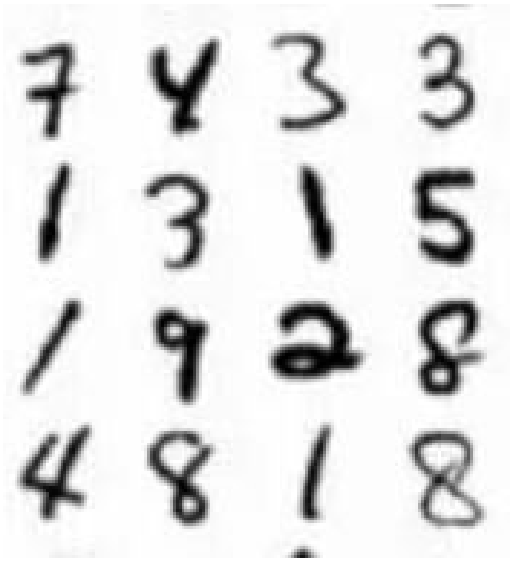
```
W = tf.Variable(tf.zeros([784, 10]))#设置一层有10个神经元，每个神经元有784个输入权重的神经网络
b = tf.Variable(tf.zeros([10]))#设置神经元的截距，10个神经元有10个截距
```

- Session
  - Session 是程序运行过程的语句，它指导了程序下一步要做什么。使用格式如下：

```
sess=tf.Session()#这表明以后用sess变量来执行步骤，语句为sess.run()
sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels})
#我自己简单理解为 sess.run(要计算的东西,feed_dict=用于计算的数据)
```

# 神经网络的初步构建

- 我们尝试在 tensorflow 上建立一个简单的神经网络来进行如下数据的手写识别

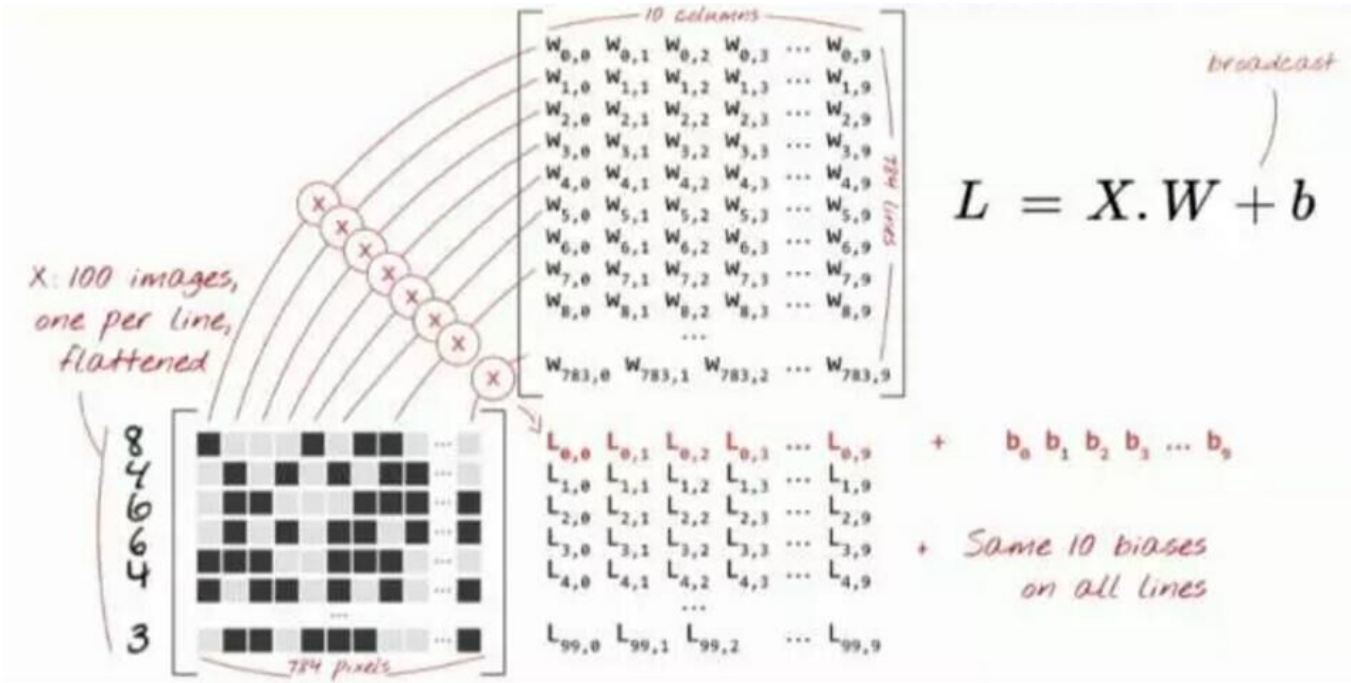


- 每一个手写数字是一个 $28 * 28 * 1$ 的矩阵，一共有 50000 个数据。这批数据是 TensorFlow 平台自带的数据，在程序开头添加两行代码可以自动下载：

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
#数据下载到当前工作目录下的叫做MNIST_data的文件夹下
```

# 神经网络的初步构建

- 我们想要构建一个这样结构的神经网络，将每一个手写数字的图像展平，输入其行向量值，输出一个 1*10 的行向量，这个行向量表明了在神经网络分类器的作用下该手写数字的分类结果



# 神经网络的初步构建

- 输出层用的函数为 Softmax 函数，即对某个图像获得的线性和 $l = (l_0, l_1, l_2, ..., l_9)$, 做如下运算：

$$y = softmax(l)$$

$$y_i = \frac{e^{l_i}}{\Sigma_{i=0}^{9} e^{l_i}}$$

- 我们所得到的 y 就是分类结果，y 是一个行向量，其每一个元素都在 0-1 之间，最大值对应的列数代表分类结果。比如 $y = (0.01, 0.02, 0.03, 0.04, 0.9, 0, 0, 0, 0, 0)$ 对应的最大值是 4 列，那么分类结果就是 4

- 计算训练结果 $y$ 与真值 $y'$ 的交叉熵：

$$E = -\Sigma_{i=0}^{9} y_i' * ln(y_i)$$

- 我们的目标是最小化 E(当然最后得到的 E 是一组向量，每一个数值为一张图片的交叉熵)

# 神经网络的初步构建

- 输出层用的函数为 Softmax 函数，即对某个图像获得的线性和 $l = (l_0, l_1, l_2, ..., l_9)$, 做如下运算：

$$y = softmax(l)$$

$$y_i = \frac{e^{l_i}}{\Sigma_{i=0}^{9} e^{l_i}}$$

- 我们所得到的 y 就是分类结果，y 是一个行向量，其每一个元素都在 0-1 之间，最大值对应的列数代表分类结果。比如 $y = (0.01, 0.02, 0.03, 0.04, 0.9, 0, 0, 0, 0, 0)$ 对应的最大值是 4 列，那么分类结果就是 4

- 计算训练结果 $y$ 与真值 $y'$ 的交叉熵：

$$E = -\Sigma_{i=0}^{9} y_i' * ln(y_i)$$

- 我们的目标是最小化 E(当然最后得到的 E 是一组向量，每一个数值为一张图片的交叉熵)

# 神经网络的初步构建

- 代码如下：

```
import tensorflow as tf
import types
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

#定义变量，x为输入图像，y为图像标签
x = tf.placeholder(tf.float32, [None, 784])#实例中使用的数据类型是已经展开为1*
                                            784的数据
y_ = tf.placeholder(tf.float32, [None, 10])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

#定义神经网络的运算，这里使用softmax函数
y = tf.nn.softmax(tf.matmul(x, W) + b)

#定义交叉熵
cross_entropy = -tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1])

#训练准确度计算
is_correct=tf.equal(tf.argmax(y,1),tf.argmax(y_,1))
accuracy=tf.reduce_mean(tf.cast(is_correct,tf.float32))#将bool值转换为float32位

#定义迭代方法，这里用的是梯度下降法，函数使用方法是前面0.003是学习率，后面是目标
                                            函数

train_step=tf.train.GradientDescentOptimizer(0.003).minimize(cross_entropy)
```

# 神经网络的初步构建

```
#变量初始化
init=tf.global_variables_initializer()

#开始流程
sess=tf.Session()
batch_X,batch_Y=mnist.train.next_batch(100)#从训练集中一次获取100个数据

#开始训练过程

for i in range(1000):
    batch_X,batch_Y=mnist.train.next_batch(100)

    train_data={x:batch_X,y_:batch_Y}

    sess.run(train_step,feed_dict=train_data)

#采用测试集数据看看训练成果
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

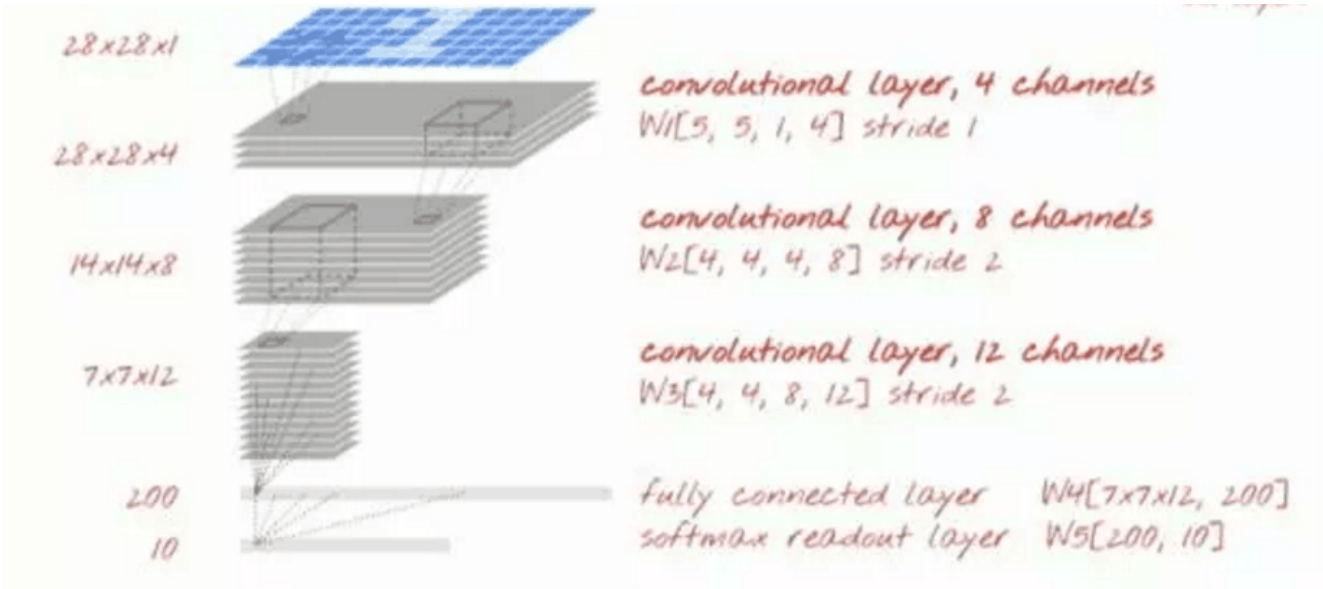
# 增加神经网络的层数示例

- 仍然用刚才的数据，我们来演示以下如何增加神经网络的层数. 这里我们将神经网络变为 5 层，第一层有 200 个神经元，第二层有 100 个，第三层有 60 个，第四层有 30 个，第 5 层有 10 个，开始 4 层神经网络的输出函数依次为 Sigmoid,Relu,Sigmoid,Relu，最后一层的神经网络用 Softmax 函数进行输出

```
W1 = tf.Variable(tf.truncated_normal([784,200],stddev=0.01))
b1 = tf.Variable(tf.zeros([200]))
W2 = tf.Variable(tf.truncated_normal([200,100],stddev=0.01))
b2 = tf.Variable(tf.zeros([100]))
W3 = tf.Variable(tf.truncated_normal([100,60],stddev=0.01))
b3 = tf.Variable(tf.zeros([60]))
W4 = tf.Variable(tf.truncated_normal([60,30],stddev=0.01))
b4 = tf.Variable(tf.zeros([30]))
W5 = tf.Variable(tf.truncated_normal([30,10],stddev=0.01))
b5 = tf.Variable(tf.zeros([10]))
y1 = tf.nn.sigmoid(tf.matmul(x, W1) + b1)
y2 = tf.nn.relu(tf.matmul(y1, W2) + b2)
y3 = tf.nn.sigmoid(tf.matmul(y2, W3) + b3)
y4 = tf.nn.relu(tf.matmul(y3, W4) + b4)
y = tf.nn.softmax(tf.matmul(y4, W5) + b5)
```

# 卷积神经网络构建

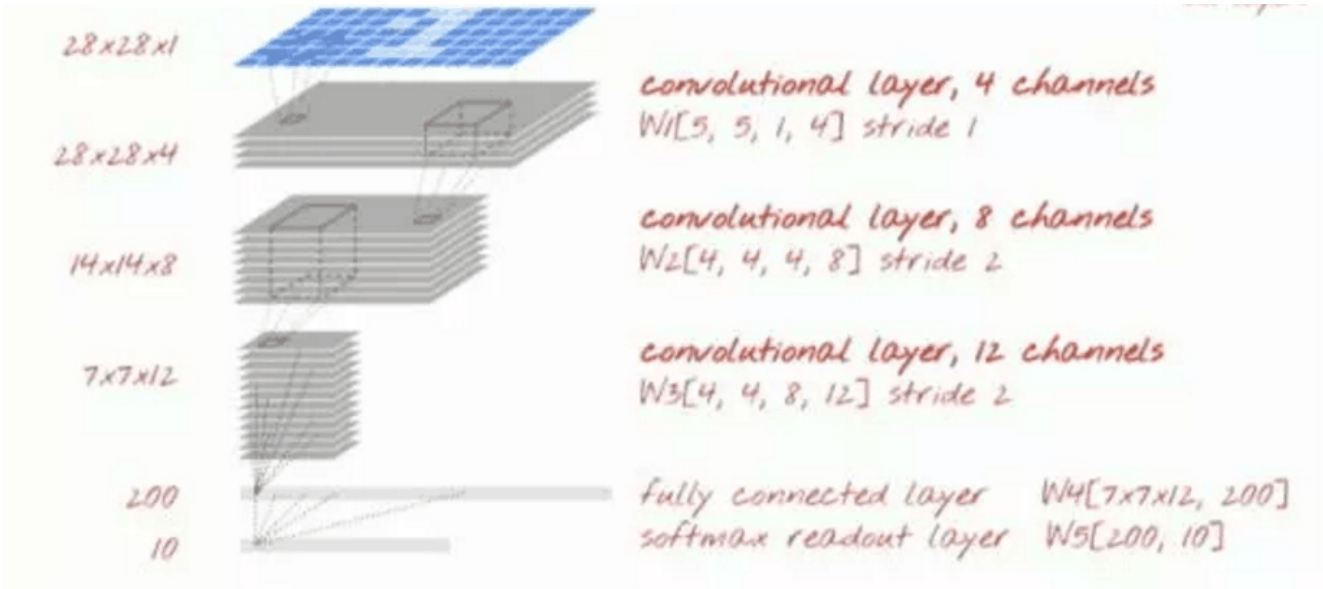
- 用刚才的数据，我们来尝试共构建卷积神经网络，基础结构如下图




- 该卷积网络有 3 个卷积层，如图所示。W 向量表示感受野，步长，输出。第一个卷积层有 5 个神经元，感受野为 5*5，每个神经元的颜色通道为 1，移动步长为 1，因此输出的是 $28 * 28 * 5$ 的数据。第二个卷积层有 8 个神经元，感受野为 4*4，因为上一层有 4 个神经元进行输出，因此颜色通道为 4，步长为 2，输出 $14 * 14 * 8$ 的数据。第三层类似。第四层每个神经元与第三层输出的所有数据全部连接，也就是说感受野是 $7 * 7 * 12$, 一共有 200 个神经元，这样就将图像数据变成了 $1 * 200$ 的列向量，第五个神经元将其转化为我们要的 $1 * 10$ 的分类输出

# 卷积神经网络构建

- 用刚才的数据，我们来尝试共构建卷积神经网络，基础结构如下图




- 该卷积网络有 3 个卷积层，如图所示。W 向量表示感受野，步长，输出。第一个卷积层有 5 个神经元，感受野为 5*5，每个神经元的颜色通道为 1，移动步长为 1，因此输出的是 $28 * 28 * 5$ 的数据。第二个卷积层有 8 个神经元，感受野为 4*4，因为上一层有 4 个神经元进行输出，因此颜色通道为 4，步长为 2，输出 $14 * 14 * 8$ 的数据。第三层类似。第四层每个神经元与第三层输出的所有数据全部连接，也就是说感受野是 $7 * 7 * 12$, 一共有 200 个神经元，这样就将图像数据变成了 $1 * 200$ 的列向量，第五个神经元将其转化为我们要的 $1 * 10$ 的分类输出

# 卷积神经网络构建

- 构建卷积层与构建先前的普通层几乎是一样的，构建格式如下：

```
W1=tf.Variable(tf.truncated_normal([5,5,1,4],stddev=0.01))
B1=tf.Variable(tf.ones([4])/10)
#对比构建普通层
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
#卷积层的变量格式为[width感受野, height感受野, 颜色通道, 神经元个数], 仅仅是将普
                                                通层的第一个
#参数转化为感受野与颜色通道个数而已
```

# 卷积神经网络构建

- 代码与代码分析如下

```
import tensorflow as tf
import types
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])
#图像与标签和普通神经网络一样，仅仅多出了一个reshape的过程
x_image = tf.reshape(x, [-1,28,28,1])

#创建3个卷积层，2个全连接层，这里没有池化层
W1=tf.Variable(tf.truncated_normal([5,5,1,4],stddev=0.01))
B1=tf.Variable(tf.ones([4])/10)#神特么的10，应该是分10类吧，初始化1/10
W2=tf.Variable(tf.truncated_normal([5,5,4,8],stddev=0.01))
B2=tf.Variable(tf.ones([8])/10)
W3=tf.Variable(tf.truncated_normal([4,4,8,12],stddev=0.01))
B3=tf.Variable(tf.ones([12])/10)

#2个全连接层
W4=tf.Variable(tf.truncated_normal([7*7*12,200],stddev=0.01))
B4=tf.Variable(tf.ones([200])/10)
W5=tf.Variable(tf.truncated_normal([200,10],stddev=0.01))
B5=tf.Variable(tf.ones([10])/10)
```

# 卷积神经网络构建

- 代码与代码分析如下

```
#进行卷积
Y1=tf.nn.relu(tf.nn.conv2d(x_image,W1,strides=[1,1,1,1],padding='SAME')+B1)
Y2=tf.nn.relu(tf.nn.conv2d(Y1,W2,strides=[1,2,2,1],padding='SAME')+B2)
Y3=tf.nn.relu(tf.nn.conv2d(Y2,W3,strides=[1,2,2,1],padding='SAME')+B3)
YY=tf.reshape(Y3,shape=[-1,7*7*12])#这里-1是预留的数字，会自动进行计算
Y4=tf.nn.relu(tf.matmul(YY,W4)+B4)#也是用relu层，然后输出一个1*200向量
y=tf.nn.softmax(tf.matmul(Y4,W5)+B5)
```

- 卷积层用的函数都是 Relu 函数，最后一层全连接层的输出用了 softmax 函数，从卷积层到全连接层的过渡需要 reshape 一下，神经网络构建完毕后所有代码与前面相同，可以试着自行补全。

- 如果我们像基本网络一样尝试运行这个示例，迭代 1000 次大约需要 5 分钟，而对测试集分类的正确率也仅仅只有 11%。这是因为卷积神经网络的参数太多，层数也很多，因此每次迭代 1 遍耗时很长，同时让每一个权重收敛所需的迭代次数也很大(大约 20000 次迭代后精度会在 80% 左右)。这就是为什么我们需要 GPU 与分布式计算方法来进行网络训练了。

# 卷积神经网络构建

- 代码与代码分析如下

```
#进行卷积
Y1=tf.nn.relu(tf.nn.conv2d(x_image,W1,strides=[1,1,1,1],padding='SAME')+B1)
Y2=tf.nn.relu(tf.nn.conv2d(Y1,W2,strides=[1,2,2,1],padding='SAME')+B2)
Y3=tf.nn.relu(tf.nn.conv2d(Y2,W3,strides=[1,2,2,1],padding='SAME')+B3)
YY=tf.reshape(Y3,shape=[-1,7*7*12])#这里-1是预留的数字，会自动进行计算
Y4=tf.nn.relu(tf.matmul(YY,W4)+B4)#也是用relu层，然后输出一个1*200向量
y=tf.nn.softmax(tf.matmul(Y4,W5)+B5)
```

- 卷积层用的函数都是 Relu 函数，最后一层全连接层的输出用了 softmax 函数，从卷积层到全连接层的过渡需要 reshape 一下，神经网络构建完毕后所有代码与前面相同，可以试着自行补全。

- 如果我们像基本网络一样尝试运行这个示例，迭代 1000 次大约需要 5 分钟，而对测试集分类的正确率也仅仅只有 11%。这是因为卷积神经网络的参数太多，层数也很多，因此每次迭代 1 遍耗时很长，同时让每一个权重收敛所需的迭代次数也很大(大约 20000 次迭代后精度会在 80% 左右)。这就是为什么我们需要 GPU 与分布式计算方法来进行网络训练了。

# 参考书目


- 神经网络在应用科学和工程中的应用：从基本原理到复杂的模式识别 (是基础神经网络的一部字典，结合了大量统计知识)
- Springer Series in Operations Research and Financial Engineering(连续函数优化的理论入门书，神经网络的关键步骤就是对目标函数全局最小值的寻找)
- Identifying and attacking the saddle point problem in high-dimensional non-convex optimization(指出高维优化问题中根本没有那么多局部极值。作者依据统计物理，随机矩阵理论和神经网络理论的分析，以及一些经验分析提出高维非凸优化问题之所以困难，是因为存在大量的鞍点（梯度为零并且 Hessian 矩阵特征值有正有负）而不是局部极值。)

# 参考网页

- 微信公众号“机器之心”有很多翻译好的关于神经网络的文章
- Tensorflow 的官网
- http://cs231n.github.io/convolutional-networks/ CNN 神经网络的直观解释
- http://www.jeyzhang.com/tensorflow-learning-notes-2.html (如何处理过拟合问题，有代码)