# 深度学习

## 深度学习基础

### 1.1机器学习和深度学习

机器学习：“计算机程序可以在给定某种类别的任务 T 和性能度量 P 下学习经验 E ，如果其在任务 T 中的性能恰好可以用 P 度量，则随着经验 E 而提高。”让我们根据前边的解释来定义我们的预测系统：我们的任务是确定可能产生飓风的气象条件。性能P是在系统所有给定的条件下有多少次正确预测飓风。经验E是我们的系统的迭代次数。

深度学习：深度学习是一种特殊机器学习，通过学习将世界使用嵌套的概念层次来表示并实现巨大的功能和灵活性，其中每个概念都定义为与简单概念相关联，而更为抽象的表示则以较不抽象的方式来计算。

机器学习和深度学习的对比：深度学习与传统的机器学习最主要的区别在于随着数据规模的增加其性能也不断增长。当数据很少时，深度学习算法的性能并不好。这是因为深度学习算法需要大量的数据来完美地理解它。另一方面，在这种情况下，传统的机器学习算法使用制定的规则，性能会比较好。传统机器学会将问题分解为两步：物体检测和物体识别。首先，使用一个边界框检测算法扫描整张图片找到可能的是物体的区域；然后使用物体识别算法(例如 SVM 结合 HOG )对上一步检测出来的物体进行识别。相反，深度学习会直接将输入数据进行运算得到输出结果。例如可以直接将图片传给 YOLO 网络(一种深度学习算法)，YOLO 网络会给出图片中的物体和名称。

### 1.2 深度学习简介



深度学习将所需的复杂映射分解为一系列嵌套的简单映射（每个由模型的不同层描述）

来解决这一难题。输入展示在可见层 (visible layer)，这样命名的原因是因为它包含我们能观察到的变量。然后是一系列从图像中提取越来越多抽象特征的隐藏层 (hidden layer)。因为它们的值不在数据中给出，所以将这些层称为 ‘‘隐藏”; 模型必须确定哪些概念有利于解释观察数据中的关系。

一种是在深度概率模型中使用的方法，将描述概念如何彼此相关的图的深度视为模型深度。一种将计算图的深度视为模型深度。



从线性模型（线性加权重）到连接机制（当网络将大量简单计算单元连接在一起时可以实现智能行为）。连接机制其中一个概念是分布式表示，这一想法是系统每个的输入应该由许多特征表示的，并且每个特征应参与许多可能输入的表示。连接机制运动的另一个重要成就是反向传播算法的成功运用（训练具有内部表示的深度神经网络）和普及 (Rumelhart et al., 1986c; LeCun, 1987)。这个算法虽然曾黯然失色不再流行，但截至写书之时，仍是训练深度模型的主要方法。神经网络研究的第三次浪潮始于 2006 年的突破。名为深度信念网络的神经网络可以使用一种称为贪心逐层训练的策略进行有效地训练。

### 1.3 深度学习数学基础

#### 1.3.1 线性代数

##### 1.3.1.1 标量，向量，矩阵和张量

标量 (scalar)：一个标量就是一个单独的数，不同于线性代数中大多数概念会涉及到多个数。我们用斜体表示标量。标量通常赋予小写的变量名称。当我们介绍标量时，会明确它们是哪种类型的数。比如，在定义实数标量时，我们可能会说‘让 s ∈ R 表示一条线的斜率’。在定义自然数标量时，我们可能会说‘让 n ∈ N 表示元素的数目’。

向量 (vector)：一个向量是一列数。这些数是有序排列的。通过次序中的索引，我们可以确定每个单独的数。通常我们赋予向量粗体的小写变量名称，比如 x。向量中的元素可以通过带脚标的斜体表示。向量 x 的第一个元素是 x1，第二个元素是 x2，等等。有时我们需要指定向量中某个集合的元素。在这种情况下，我们定义一个包含这些索引的集合，然后将该集合写在脚标处。比如，指定 x1，x3 和 x6，我们定义集合 S = {1, 3, 6} xS。比如 x\_1 表示 x 中除 x1 外的所有元素，x\_S 表示 x 中除 x1，x3，x6 外所有元素构成的向量。

矩阵 (matrix)：矩阵是二维数组，其中的每一个元素被两个索引而非一个所确定。我们通常会赋予矩阵粗体的大写变量名称，比如 A。

[ A1,1 A1,2 ]

[ A2,1 A2,2 ]

张量 (tensor)：在某些情况下，我们会讨论不只两维坐标的数组。一般地，一组数组中的元素分布在若干维坐标的规则网格中，我们将其称之为张量。我们使用这种字体 A 来表示张量 “A’’。张量 A 中坐标为 (i, j, k) 的元素记作 Ai,j,k。

注意：

1.转置 (transpose) 是矩阵的重要操作之一。矩阵的转置是以对角线为轴的镜像，这条从左上角到右下角的对角线被称为主对角线 (main diagonal)。。我们将矩阵 A 的转置表示为 A⊤，定义如下(A⊤)i,j = Aj,i.



即可以这么理解，按照主对角线进行镜像。

2.向量可以看作是只有一列的矩阵。对应地，向量的转置可以看作是只有一行的矩阵。有时，我们将向量表示成行矩阵的转置，写在行中，然后使用转置将其变为标准的列向量，比如 x = [x1, x2, x3]⊤.

3.标量可以看作是只有一个元素的矩阵。因此，标量的转置等于它本身，a = a⊤。

4.只要矩阵的形状一样，我们可以把两个矩阵相加。两个矩阵相加是指对应位置的元素相加，比如 C = A + B，其中 Ci,j = Ai,j + Bi,j。

5.标量和矩阵相乘，或是和矩阵相加时，我们将其与矩阵的每个元素相乘或相加，比如 D = a · B + c，其中 Di,j = a · Bi,j + c。在深度学习中，我们也使用一些不那么常规的符号。我们允许矩阵和向量相加，产生另一个矩阵：C = A + b，其中 Ci,j = Ai,j + bj。换言之，向量 b 和矩阵 A 的每一行相加。这个速记方法使我们无需在加法操作前定义复制向量 b 到矩阵的每一行。这种隐式地复制向量 b 到很多位置的方式，被称为广播 (broadcasting)。

##### 1.3.1.2 矩阵和向量相乘

两个矩阵 A 和 B 的矩阵乘积 (matrixproduct) 是第三个矩阵 C。为了使乘法定义良好，矩阵 A 的列数必须和矩阵 B 的行数相等。如果矩阵 A 的形状是 m × n，矩阵 B 的形状是 n × p，那么矩阵 C 的形状是 m × p。矩阵乘积可以作用于两个或多个并在一起的矩阵，C = AB，该乘法操作定义为Ci,j =∑Ai,k Bk,j .需要注意的是，两个两个矩阵的标准乘积不是指两个矩阵中对应元素的乘积。不过，那样的矩阵操作确实是存在的，被称为元素对应乘积 或者哈达玛乘积A◎B。两个相同维数的向量 x 和 y 的点积 (dot product) 可看作是矩阵乘积 x⊤y。我们可以把矩阵乘积 C = AB 中计算 Ci,j 的步骤看作是 A 的第 i 行和 B 的第 j 列之间的点积。矩阵支持分配A（B+C）=AB+AC 结合律ABC=ACB 不支持结合律。两个向量的点积 (dot product) 满足交换律x⊤y = y⊤x（x⊤y =（x⊤y)⊤= y⊤x）。矩阵乘积的转置有着简单的形式：(AB)⊤ = B⊤A⊤。

##### 1.3.1.3 单位矩阵和逆矩阵

单位矩阵：I3

[ 1 0 0 ]

[ 0 1 0]

[ 0 0 1]

A\*A- =I3 A乘以A逆矩阵=单位矩阵 Ax=b A A-X = b A- x=b A-

确定 Ax = b 是否有解相当于确定向量 b 是否在 A 列向量的生成子空间中。这

个特殊的生成子空间被称为 A 的列空间 (column space) 或者 A 的值域 (range)。

##### 1.3.1.4 范数

在机器学习中，我们经常使用被称为范数 (norm) 的函数衡量向量大小。范数（包括 L

p 范数）是将向量映射到非负值的函数。直观上来说，向量 x 的范数是衡量从原点到点 x 的距离。

#### 1.3.2 概率和信息论

几乎所有的活动都需要能够在不确定性存在时进行推理。不确定性有三种可能的来源：1. 被建模系统内在的随机性。2. 不完全观测。3.不完全建模。

随机变量，概率分布，边缘概率，条件概率。概率分布：Multinoulli 分布和Bernoulli分布和高斯分布(正太分布)，指数分布，经验分布和混合分布。





信息论是应用数学的一个分支，主要研究的是对一个信号能够提供信息的多少进行量化。信息论的基本想法是一个不太可能的事件居然发生了，要比一个非常可能的事件发生，能提供更多的信息。消息说：‘‘今天早上太阳升起’’ 信息量是如此之少以至于没有必要发送，但一条消息说：‘‘今天早上有日食’’ 信息量就很丰富。我们想要通过这种基本想法来量化信息。特别地，

• 非常可能发生的事件信息量要比较少，并且极端情况下，确保能够发生的事件应该没有信息量。

• 更不可能发生的事件要具有更高的信息量。

• 独立事件应具有增量的信息。例如，投掷的硬币两次正面朝上传递的信息量，应该是投掷一次硬币正面朝上的信息量的两倍。

为了满足上述三个性质，我们定义一个事件 x = x 的自信息 (self-information)为

I(x) = = log P(x). (3.48)

在本书中，我们总是用 log 来表示自然对数，底数为 e。因此我们定义的 I(x) 单位是奈特 (nats)。一奈特是以 1e 的概率观测到一个事件时获得的信息量。其他的材料中使用底数为 2 的对数，单位是比特 (bit) 或者香农 (shannons)；通过比特度量的信息只是通过奈特度量信息的常数倍。

#### 1.3.3 数值计算

机器学习算法通常需要大量的数值计算。这通常是指通过迭代地更新解来解决数学问题的算法，而不是解析地提供正确解的符号表达。常见的操作包括优化（找到最小化或最大化函数值的参数）和线性方程组的求解。

条件数表明函数相对于输入的微小变化而变化的快慢程度。输入被轻微扰动而迅速改变的函数对于科学计算来说是可能是有问题的，因为输入中的舍入误差可能导致输出的巨大变化。

我们把要最小化或最大化的函数称为目标函数 (objective function) 或准则 (criterion)。当我们对其进行最小化时，我们也把它称为代价函数 (cost function)、损失函数 (loss function) 或误差函数 (error function)。

我们会探讨两种统计学的主要方法：频率估计和贝叶斯推断。大部分机器学习算法可以分成监督学习和无监督学习两类；我们将探讨不同的分类，并为每类提供一些简单的机器学习算法作为示例。大部分深度学习算法都基于随机梯度下降求解。我们将介绍如何组合不同的算法部分，例如优化算法、代价函数、模型和数据集，来建立一个机器学习算法。

### 1.4 机器学习基础

**任务，T**

通常机器学习任务定义为机器学习系统该如何处理样本 (example)。样本是指我们从某些希望机器学习系统处理的对象或事件中收集到的已经量化的特征 (feature)的集合。

机器学习可以解决很多类型的任务。一些非常常见的机器学习任务列举如下：

• 分类：在这类任务中，计算机程序需要指定某些输入属于 k 类中的哪一类。

• 输入缺失分类：当输入向量的每个度量不被保证的时候，分类问题将会更有挑战。

• 回归：这类任务中，计算机程序会给定输入预测数值。为了解决这个问题，学

习算法会

• 转录：这类任务中，机器学习系统观测一些相对非结构化表示的数据，并转录信息为离散的文本形式。

• 机器翻译：在机器翻译任务中，输入是一种语言的符号序列，计算机程序必须将其转化成另一种语言的符号序列。

• 结构化输出：结构化输出任务涉及到输出是不同元素之间重要关系的向量（或者是含多个值的其他数据结构）的任务。这是一个很大的范畴，包括上面转录任务和翻译任务在内的很多其他任务。例如语法分析——映射自然语言句子到语法结构树，并标记树的节点为动词，名词，副词等等。参考Collobert2011) 应用深度学习到语法分析。另一个例子是图像的像素级分割，将每一个像素分配到特定类别。

• 异常检测：这类任务中，计算机程序在一组事件或对象中筛选，并标记不正常

或非典型的个体。

• 合成和采样：这类任务中，机器学习程序生成一些和训练数据相似的新样本。波形。这是一类结构化输出任务，但是多了每个输入并非只有一个正确输出的条件，我们明确希望输出有很大的偏差，使结果看上去更加自然和真实。

• 缺失值填补：这类任务中，机器学习算法给定一个新样本x ∈ Rn，x 中某些元

素 xi 缺失。算法必须填补这些缺失值。

• 去噪：这类任务中，机器学习算法的输入是，由未知破坏过程从干净样本x ∈ Rn得到的污染样本x˜ ∈ Rn。算法根据污染后的样本x˜ 预测干净的样本x，或者更一般地预测条件概率分布 P(x | x˜)。

• 密度估计或概率分布律函数估计：在密度估计问题中，机器学习算法学习函数pmodel : R

n → R，其中 pmodel(x) 可以解释成样本采样空间的概率密度函数（如果 x 是连续的）或者概率分布律函数（如果 x 是离散的）。

**性能度量，P**

为了评估机器学习算法的能力，我们必须设计其性能的定量度量。通常性能度量P 是特定于系统执行的任务 T 而言的。对于诸如分类，缺失输入分类和转录任务，我们通常度量模型的准确率 (accuracy)。准确率是指该模型输出正确结果的样本比例。我们也可以通过错误率 (errorrate) 得到相同的信息。错误率是指该模型输出错误结果的样本比例。

**经验，E**

根据学习过程中的不同经验，机器学习算法可以大致分类为无监督 (unsupervised) 和监督 (supervised)。本书中的大部分学习算法可以理解成在整个数据集 (dataset) 上获取经验。数据集是指很多样本组成的集合。有时我们也将样本称为数据点 (data point)。

无监督学习算法 (unsupervised learning algorithm) 训练含有很多特征的数据集，然后学习出这个数据集上有用的结构性质。在深度学习中，我们通常要学习生成数据集的整个概率分布，显式地，比如密度估计，或是隐式地，比如合成或去噪。还有一些其他类型的无监督学习任务，例如聚类，将数据集分成相似样本的集合。

监督学习算法 (supervised learning algorithm) 训练含有很多特征的数据集，不过数据集中的样本都有一个标签 (label) 或目标 (target)。例如，Iris数据集注明了每个鸢尾花卉样本属于什么品种。监督学习算法通过研究 Iris数据集，学习如何根据测量结果将样本划分到三个不同品种。

大部分机器学习算法简单地训练于一个数据集上。数据集可以用很多不同方式来表示。在所有的情况下，数据集都是样本的集合，而样本是特征的集合。表示数据集的常用方法是设计矩阵 (design matrix)。设计矩阵的每一行包含一个不同的样本。每一列对应不同的特征。

通常情况下，当我们训练机器学习模型时，我们可以访问训练集，在训练集上计算一些度量误差，被称为训练误差 (training error)，并且我们会降低训练误差。目前为止，我们讨论的是一个简单的优化问题。机器学习和优化不同的地方在于，我们也希望泛化误差 (generalization error)，也被称为测试误差 (test error)，很低。泛化误差被定义为新输入的误差期望。这里，期望取值自我们期望系统在现实中从输入分布中采样得到的不同可能值。

在这个过程中，测试误差期望会大于或等于训练误差期望。以下是决定机器学习算法效果是否好的因素：

1. 降低训练误差

2. 缩小训练误差和测试误差的差距

这两个因素对应机器学习的两个主要挑战：欠拟合 (underfitting) 和过拟合 (overfitting)。欠拟合发生于模型不能在训练集上获得足够低的误差。过拟合发生于训练误差和和测试误差之间的差距太大。

点估计也可以指输入和目标变量之间关系的估计。我们将这类点估计称为函数估计。

### 1.5 激活函数

1. 什么是激活函数

在神经元中，输入的 inputs 通过加权，求和后，还被作用了一个函数，这个函数就是激活函数 Activation Function。

2. 为什么要用

如果不用激励函数，每一层输出都是上层输入的线性函数，无论神经网络有多少层，输出都是输入的线性组合。如果使用的话，激活函数给神经元引入了非线性因素，使得神经网络可以任意逼近任何非线性函数，这样神经网络就可以应用到众多的非线性模型中。

3. 都有什么激活函数

(1) sigmoid函数

sigmoid函数也叫 Logistic 函数，用于隐层神经元输出，取值范围为(0,1)，它可以将一个实数映射到(0,1)的区间，可以用来做二分类。在特征相差比较复杂或是相差不是特别大时效果比较好。

sigmoid缺点：

激活函数计算量大，反向传播求误差梯度时，求导涉及除法

反向传播时，很容易就会出现梯度消失的情况，从而无法完成深层网络的训练

Sigmoids函数饱和且kill掉梯度。

Sigmoids函数收敛缓慢。

为何会出现梯度消失：

反向传播算法中，要对激活函数求导，导数从 0 开始很快就又趋近于 0 了，易造成“梯度消失”现象。

1. Tanh函数

也称为双切正切函数，取值范围为[-1,1]。tanh在特征相差明显时的效果会很好，在循环过程中会不断扩大特征效果。与 sigmoid 的区别是，tanh 是 0 均值的，因此实际应用中 tanh 会比 sigmoid 更好。

1. ReLU

Rectified Linear Unit(ReLU) - 用于隐层神经元输出。

ELU特点：

输入信号 <0 时，输出都是0，>0 的情况下，输出等于输入

ReLU 的优点：

Krizhevsky et al. 发现使用 ReLU 得到的 SGD 的收敛速度会比 sigmoid/tanh 快很多

ReLU 的缺点：

训练的时候很”脆弱”，很容易就”die”了。例如，一个非常大的梯度流过一个 ReLU 神经元，更新过参数之后，这个神经元再也不会对任何数据有激活现象了，那么这个神经元的梯度就永远都会是 0.如果 learning rate 很大，那么很有可能网络中的 40% 的神经元都”dead”了。

1. softmax函数

Softmax - 用于多分类神经网络输出

就是如果某一个 zj 大过其他 z, 那这个映射的分量就逼近于 1,其他就逼近于 0，主要应用就是多分类。

为什么要取指数，第一个原因是要模拟 max 的行为，所以要让大的更大。第二个原因是需要一个可导的函数。

1. sigmoid ，ReLU， softmax 的比较

Sigmoid 和 ReLU 比较：

sigmoid 的梯度消失问题，ReLU 的导数就不存在这样的问题，对比sigmoid类函数主要变化是：

1）单侧抑制

2）相对宽阔的兴奋边界

3）稀疏激活性。

Sigmoid 和 Softmax 区别：

sigmoid将一个real value映射到（0,1）的区间，用来做二分类。而 softmax 把一个 k 维的real value向量（a1,a2,a3,a4….）映射成一个（b1,b2,b3,b4….）其中 bi 是一个 0～1 的常数，输出神经元之和为 1.0，所以相当于概率值，然后可以根据 bi 的概率大小来进行多分类的任务。

二分类问题时 sigmoid 和 softmax 是一样的，求的都是 cross entropy loss，而 softmax 可以用于多分类问题

softmax是sigmoid的扩展，因为，当类别数 k＝2 时，softmax 回归退化为 logistic 回归。具体地说，当 k＝2 时，softmax 回归的假设函数为：

利用softmax回归参数冗余的特点，从两个参数向量中都减去向量θ1 ，得到:

最后，用 θ′ 来表示 θ2−θ1，上述公式可以表示为 softmax 回归器预测其中一个类别的概率为

另一个类别概率的为

这与 logistic回归是一致的。

softmax建模使用的分布是多项式分布，而logistic则基于伯努利分布

多个logistic回归通过叠加也同样可以实现多分类的效果，但是 softmax回归进行的多分类，类与类之间是互斥的，即一个输入只能被归为一类；多个logistic回归进行多分类，输出的类别并不是互斥的，即"苹果"这个词语既属于"水果"类也属于"3C"类别。

1. 如何选择

选择的时候，就是根据各个函数的优缺点来配置，例如：

如果使用 ReLU，要小心设置 learning rate，注意不要让网络出现很多 “dead” 神经元，如果不好解决，可以试试 Leaky ReLU、PReLU 或者 Maxout

## 2.Tensorflow

### 2.1 基础

#### 2.1.1 简介

TensorFlow™ 是一个采用数据流图（data flow graphs），用于数值计算的开源软件库。节点（Nodes）在图中表示数学操作，图中的线（edges）则表示在节点间相互联系的多维数据数组，即张量（tensor）。它灵活的架构让你可以在多种平台上展开计算，例如台式计算机中的一个或多个CPU（或GPU），服务器，移动设备等等。TensorFlow 最初由Google大脑小组（隶属于Google机器智能研究机构）的研究员和工程师们开发出来，用于机器学习和深度神经网络方面的研究，但这个系统的通用性使其也可广泛用于其他计算领域。

什么是数据流图（Data Flow Graph）：数据流图用“结点”（nodes）和“线”(edges)的有向图来描述数学计算。“节点” 一般用来表示施加的数学操作，但也可以表示数据输入（feed in）的起点/输出（push out）的终点，或者是读取/写入持久变量（persistent variable）的终点。“线”表示“节点”之间的输入/输出关系。这些数据“线”可以输运“size可动态调整”的多维数据数组，即“张量”（tensor）。张量从图中流过的直观图像是这个工具取名为“Tensorflow”的原因。一旦输入端的所有张量准备好，节点将被分配到各种计算设备完成异步并行地执行运算。



#### 2.1.2 Tensorflow简单介绍

Tensorflow是一种计算图模型，即用图的形式来表示运算过程的一种模型。Tensorflow程序一般分为图的构建和图的执行两个阶段。图的构建阶段也称为图的定义阶段，该过程会在图模型中定义所需的运算，每次运算的的结果以及原始的输入数据都可称为一个节点（operation ，缩写为op）。以下程序构建过程：



“import tensorflow as tf”，是在python中导入tensorflow模块,并另起名为“tf”；接着定义了两个常量op，m1和m2，均为1\*2的矩阵；最后将m1和m2的值作为输入创建一个矩阵加法op，并输出最后的结果result。我们分析最终的输出结果可知，其并没有输出矩阵相加的结果，而是输出了一个包含三个属性的Tensor。

第二个阶段为图的执行阶段，也就是在会话（session）中执行图模型中定义好的运算。我们通过程序2-2来解释图的执行阶段：

描述了图的执行过程，首先通过“tf.session()”启动默认图模型，再调用run()方法启动、运图模型，传入上述参数result，执行矩阵的加法，并打印出相加的结果，最后在任务完成时要记得调用close()方法，关闭会话。

我们还可以利用CPU或GPU等计算资源分布式执行图的运算过程。一般我们无需显示的指定计算资源，Tensorflow可以自动地进行识别，如果检测到我们的GPU环境，会优先的利用GPU环境执行我们的程序。但如果我们的计算机中有多于一个可用的GPU，这就需要我们手动的指派GPU去执行特定的op。tf.device(“/gpu:2”)。

Tensorflow中还提供了默认会话的机制，我们通过调用函数as\_default()生成默认会话。Sseion.as\_default()。交互式环境中经常会使用的InteractiveSession()方法，其创建sess对象后，可以直接输出运算结果。Ss = tf.InteractiveSession()。

首先，图模型的最大好处是节约系统开销，提高资源的利用率，可以更加高效的进行运算。因为我们在图的执行阶段，只需要运行我们需要的op,这样就大大的提高了资源的利用率；其次，这种结构有利于我们提取中间某些节点的结果，方便以后利用中间的节点去进行其它运算；还有就是这种结构对分布式运算更加友好，运算的过程可以分配给多个CPU或是GPU同时进行，提高运算效率；最后，因为图模型把运算分解成了很多个子环节，所以这种结构也让我们的求导变得更加方便。

Tensor（张量）是Tensorflow中最重要的数据结构，用来表示Tensorflow程序中的所有数据。Tensor本是广泛应用在物理、数学领域中的一个物理量。那么在Tensorflow中该如何理解Tensor的概念呢？

实际上，我们可以把Tensor理解成N维矩阵（N维数组）。其中零维张量表示的是一个标量，也就是一个数；一维张量表示的是一个向量，也可以看作是一个一维数组；二维张量表示的是一个矩阵；同理，N维张量也就是N维矩阵。

在计算图模型中，操作间所传递的数据都可以看做是Tensor。



构建图的运算过程输出的结果是一个Tensor，且其主要由三个属性构成：Name、Shape和Type。Name代表的是张量的名字，也是张量的唯一标识符，我们可以在每个op上添加name属性来对节点进行命名，Name的值表示的是该张量来自于第几个输出结果（编号从0开始），上例中的“mul\_3:0”说明是第一个结果的输出。Shape代表的是张量的维度，上例中shape的输出结果(1,1)说明该张量result是一个二维数组，且每个维度数组的长度是1。最后一个属性表示的是张量的类型，每个张量都会有唯一的类型，常见的张量类型如图2-2所示。



Tensorflow中对常量的初始化，不管是对数值、向量还是对矩阵的初始化，都是通过调用constant()函数实现的。因为constant()函数在Tensorflow中的使用非常频繁，经常被用于构建图模型中常量的定义，所以接下来，我们通过程序2-9了解一下constant()的相关属性：



函数constant有五个参数，分别为value，name，dtype，shape和verify\_shape。其中value为必选参数，其它均为可选参数。Value为常量的具体值，可以是一个数字，一维向量或是多维矩阵。Name是常量的名字，用于区别其它常量。Dtype是常量的类型，具体类型可参见图2-2。Shape是指常量的维度，我们可以自行定义常量的维度。

verify\_shape是验证shape是否正确，默认值为关闭状态(False)。也就是说当该参数true状态时，就会检测我们所写的参数shape是否与value的真实shape一致，若不一致就会报TypeError错误。如：上例中的实际shape为(2,0)，若我们将参数中的shape属性改为(2,1)，程序就会报如下错误：

TypeError: Expected Tensor's shape: (2, 1), got (2,).

Tensorflow还提供了一些常见常量的初始化，如：tf.zeros、tf.ones、tf.fill、tf.linspace、tf.range等，均可以快速初始化一些常量。例如：我们想要快速初始化N维全0的矩阵，我们可以利用tf.zeros进行初始化。Tensorflow还可以生成一些随机的张量，方便快速初始化一些随机值。如：tf.random\_normal()、tf.truncated\_normal()、tf.random\_uniform()、tf.random\_shuffle()等。

除了常量constant()，变量variable()也是在Tensorflow中经常会被用到的函数。变量的作用是保存和更新参数。执行图模型时，一定要对变量进行初始化，经过初始化后的变量才能拿来使用。变量的使用包括创建、初始化、保存、加载等操作。



展示了创建变量的多种方式。我们可以把函数variable()理解为构造函数，构造函数的使用需要初始值，而这个初始值是一个任何形状、类型的Tensor。也就是说，我们既可以通过创建数字变量、一维向量、二维矩阵初始化Tensor，也可以使用常量或是随机常量初始化Tensor，来完成变量的创建。当我们完成了变量的创建，接下来，我们要对变量进行初始化。变量在使用前一定要进行初始化，且变量的初始化必须在模型的其它操作运行之前完成。初始化变量的三种方式：初始化全部变量、初始化变量的子集以及初始化单个变量。首先，global\_variables\_initializer()方法是不管全局有多少个变量，全部进行初始化，是最简单也是最常用的一种方式；variables\_initializer()是初始化变量的子集，相比于全部初始化化的方式更加节约内存；Variable()是初始化单个变量，函数的参数便是要初始化的变量内容。通过上述的三种方式，我们便可以实现变量的初始化，放心的使用变量了。



我们经常在训练模型后，希望保存训练的结果，以便下次再使用或是方便日后查看，这时就用到了Tensorflow变量的保存。变量的保存是通过tf.train.Saver()方法创建一个Saver管理器，来保存计算图模型中的所有变量。



我们要注意，我们的存储文件save.ckpt是一个二进制文件，Saver存储器提供了向该二进制文件保存变量和恢复变量的方法。保存变量的方法就是程序中的save()方法，保存的内容是从变量名到tensor值的映射关系。Saver提供了一个内置的计数器自动为checkpoint文件编号。这就支持训练模型在任意步骤多次保存。此外，还可以通过global\_step参数自行对保存文件进行编号，例如：global\_step=2，则保存变量的文件夹为model.ckpt-2。

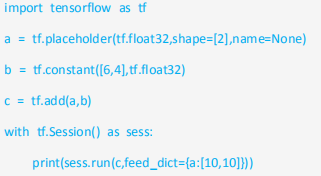
那如何才能恢复变量呢？首先，我们要知道一定要用和保存变量相同的Saver对象来恢复变量。其次，不需要事先对变量进行初始化。

本程序示例中，我们要注意：变量的获取是通过restore()方法，该方法有两个参数，分别是session和获取变量文件的位置。我们还可以通过latest\_checkpoint()方法，获取到该目录下最近一次保存的模型。

以上就是对变量创建、初始化、保存、加载等操作的介绍。此外，还有一些与变量相关的重要函数，如：eval()等。

认识了常量和变量，Tensorflow中还有一个非常重要的常用函数——placeholder。placeholder是一个数据初始化的容器，它与变量最大的不同在于placeholder定义的是一个模板，这样我们就可以session运行阶段，利用feed\_dict的字典结构给placeholder填充具体的内容，而无需每次都提前定义好变量的值，大大提高了代码的利用率。

Placeholder()方法有dtype，shape和name三个参数构成。dtype是必填参数，代表传入value的数据类型；shape是选填参数，代表传入value的维度；name也是选填参数，代表传入value的名字。我们可以把这三个参数看作为形参，在使用时传入具体的常量值。这也是placeholder不同于常量的地方，它不可以直接拿来使用，而是需要用户传递常数值。



Tensorflow中还有一个重要的概念——fetch。Fetch的含义是指可以在一个会话中同时运行多个op。这就方便我们在实际的建模过程中，输出一些中间的op，取回多个tensor。

参考链接：<https://www.cnblogs.com/tensorflownews/p/8671397.html>

### 2.2 入门教程

#### 2.2.1 入门教程学习之前言

参考链接：<https://blog.csdn.net/jdbc/article/details/52402302>

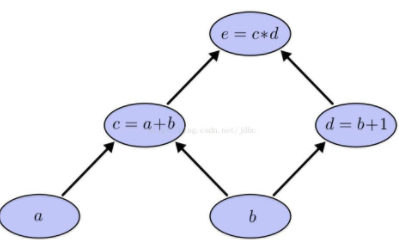
Tensorflow 拥有易用的 python 接口，而且可以部署在一台或多台 cpu , gpu 上，兼容多个平台，包括但不限于安卓/windows/linux 等等平台上，而且拥有 tensorboard这种可视化工具，可以使用 checkpoint 进行实验管理，得益于图计算，它可以进行自动微分计算，拥有庞大的社区，而且很多优秀的项目已经使用 tensorflow 进行开发了。

如果不想去研究 tensorflow 繁杂的API,仅想快速的实现些什么，可以使用其他高层工具。比如 tf.contrib.learn，tf.contrib.slim，Keras 等，它们都提供了高层封装。这里是 tflearn 的github样例集。

实际上编写tensorflow可以总结为两步.

（1）组装一个graph;

（2）使用session去执行graph中的operation。



当需要计算e时就需要计算c与d，而计算c就需要计算a与b，计算d需要计算b。这样就形成了依赖关系。这种有向无环图就叫做计算图，因为对于图中的每一个节点其微分都很容易得出，因此应用链式法则求得一个复杂的表达式的导数就成为可能，所以它会应用在类似tensorflow这种需要应用反向传播算法的框架中。

graph , session , operation , tensor 四个概念的简介如下：

Tensor：类型化的多维数组，图的边；

Operation:执行计算的单元，图的节点；

Graph：一张有边与点的图，其表示了需要进行计算的任务；

Session:称之为会话的上下文，用于执行图。

Graph仅仅定义了所有 operation 与 tensor 流向，没有进行任何计算。而session根据 graph 的定义分配资源，计算 operation，得出结果。既然是图就会有点与边，在图计算中 operation 就是点而 tensor 就是边。Operation 可以是加减乘除等数学运算，也可以是各种各样的优化算法。每个 operation 都会有零个或多个输入，零个或多个输出。 tensor 就是其输入与输出，其可以表示一维二维多维向量或者常量。而且除了Variables指向的 tensor 外所有的 tensor 在流入下一个节点后都不再保存。

import tensorflow as tf

graph = tf.Graph()

with graph.as\_default():

foo = tf.Variable(3,name='foo')

bar = tf.Variable(2,name='bar')

result = foo + bar

initialize = tf.global\_variables\_initializer()

print(result) #Tensor("add:0", shape=(), dtype=int32)

这段代码，首先会载入tensorflow，定义一个graph类，并在这张图上定义了foo与bar的两个变量，最后对这个值求和，并初始化所有变量。其中，Variable是定义变量并赋予初值。让我们看下result（最后1行代码）。后面是输出，可以看到并没有输出实际的结果，由此可见在定义图的时候其实没有进行任何实际的计算。

with tf.Session(graph=graph) as sess:

sess.run(initialize)

res = sess.run(result)

print(res) #

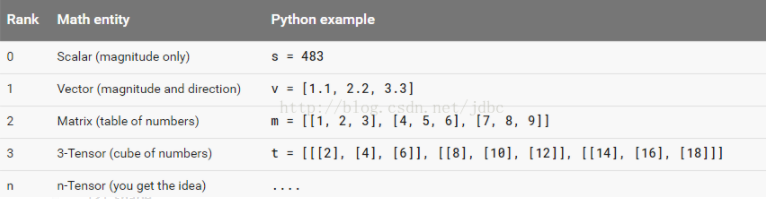
定义了session，并在session中执行了真正的初始化，并且求得result的值并打印出来。可以看到，在session中产生了真正的计算，得出值为5。

Tensorflow的数据结构有着rank,shape,data types的概念，下面来分别讲解。

数据结构

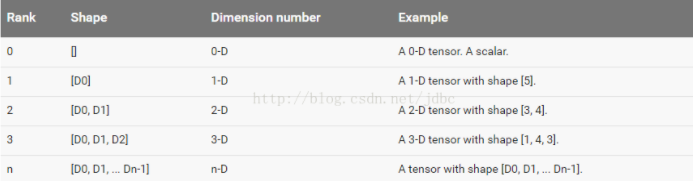
1. Rank

Rank一般是指数据的维度，其与线性代数中的rank不是一个概念。其常用rank举例如下。



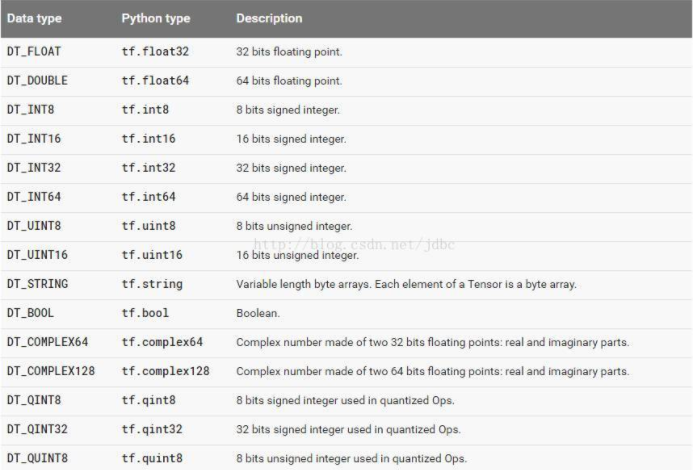
（2）shape

Shape指tensor每个维度数据的个数，可以用python的list/tuple表示。下图表示了rank,shape的关系。



（3）data type

Data type，是指单个数据的类型。常用DT\_FLOAT，也就是32位的浮点数。下图表示了所有的types。



Variables

当训练模型时，需要使用Variables保存与更新参数。Variables会保存在内存当中，所有tensor一旦拥有Variables的指向就不会在session中丢失。其必须明确的初始化而且可以通过Saver保存到磁盘上。Variables可以通过Variables初始化。

weights = tf.Variable(tf.random\_normal([784, 200], stddev=0.35),name="weights")

biases = tf.Variable(tf.zeros([200]), name="biases")

其中，tf.random\_normal是随机生成一个正态分布的tensor，其shape是第一个参数，stddev是其标准差。tf.zeros是生成一个全零的tensor。之后将这个tensor的值赋值给Variable。

实际在其初始化过程中做了很多的操作，比如初始化空间，赋初值（等价于tf.assign），并把Variable添加到graph中等操作。注意在计算前需要初始化所有的Variable。一般会在定义graph时定义global\_variables\_initializer，其会在session运算时初始化所有变量。

直接调用global\_variables\_initializer会初始化所有的Variable，如果仅想初始化部分Variable可以调用tf.variables\_initializer。

Init\_ab = tf.variables\_initializer([a,b],name=”init\_ab”)

Variables可以通过eval显示其值，也可以通过assign进行赋值。Variables支持很多数学运算，具体可以参照官方文档。

值得注意的是Variables与constant的区别。Constant一般是常量，可以被赋值给Variables，constant保存在graph中，如果graph重复载入那么constant也会重复载入，其非常浪费资源，如非必要尽量不使用其保存大量数据。而Variables在每个session中都是单独保存的，甚至可以单独存在一个参数服务器上。可以通过代码观察到constant实际是保存在graph中，具体如下:

const = tf.constant(1.0,name="constant")

print(tf.get\_default\_graph().as\_graph\_def())

另外一个值得注意的地方是尽量每一个变量都明确的命名，这样易于管理命令空间，而且在导入模型的时候不会造成不同模型之间的命名冲突，这样就可以在一张graph中容纳很多个模型。

placeholders与feed\_dict

当我们定义一张graph时，有时候并不知道需要计算的值，比如模型的输入数据，其只有在训练与预测时才会有值。这时就需要placeholder与feed\_dict的帮助。

定义一个placeholder，可以使用tf.placeholder(dtype,shape=None,name=None)函数。

#### 2.2.2 入门教程学习之基本概念

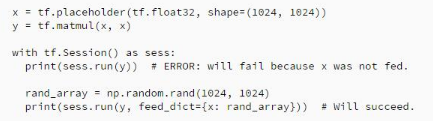
1.占位符

x = tf.placeholder(tf.float32, [None, 10])

y = tf.placeholder("float")

z = tf.placeholder(tf.int32)

占位符表示当OP 操作进行时候传递进来的过程,,也就是 feed 喂养 给予操作，所以 需要外部传递过来，比如训练图像时候X 为图像数据 Y为图像标签，这样的时候 X Y 都是来自训练测试数据，所以前面需要定义2个用来传递的占位符来传递X Y feed 需要传递 python中的 tuple元祖 feed\_dict={x: xdata, y: ydata} ,当执行op操作时候 feed传入替换 ，执行完后销毁。



这里上面 第二个参数 shape 表示约束维度 比如 shape=(1024,1024) 就是约束为1024 行 1024列的二维数组

如果shape=(None,1024) None 是python中的空值 ，表示 可以任何维度行的 1024列的二维数组

1. 变量

w = tf.Variable(0.0, name="weights")

b= tf.Variable (tf.zeros([100]),name="bias")

Variable第一个参数 表示 初始化的值 ，比如w 初始化为0 这样 w 在后面的过程中会一直变化，下次可能获取w时候是w=xxx 某个值，在tensorFlow中这种 可以通过fetch 也就是当需要获得 传递 某个最新的w 时候 可以 session.run([w]) ，这时候 传递的w 就是每次最新变化的最新的w， 如果有时候需要常量 使用k = tf.constant(3.0) #表示定义一个值为3.0的常量

1. Session会话

TensorFlow 通过回话 进行 连接 操作 执行所有的op（操作）形成一个有向图，进行执行运行 图计算 graph compute每一个回话 的 只要有变量存在的情况 网络图 第一个bottom都是 initop 也就是初始化变量

init = tf.initialize\_all\_variables()

with tf.Session() as session:

session.run(init)#1.第一个图的节点op

session.run(googlenet)#2.第二个图的节点op 1-2-n

sess.close()#释放资源

TensorFlow = tensor (多维数组) + flow (graph 图 op)

session回话上下文管理

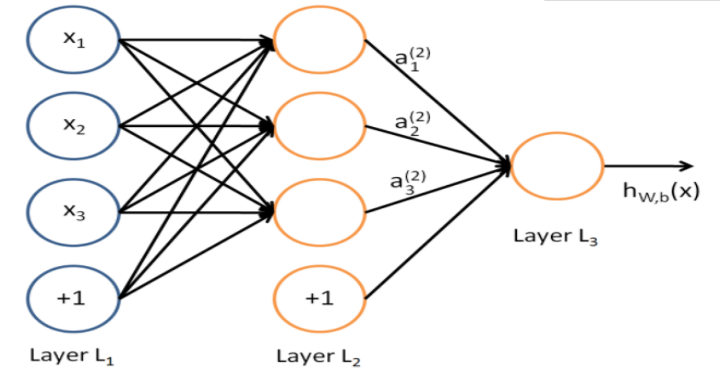
variable == tensor(多维数据变量)

placeholder == 外部传入的参数变量

#### 2.2.3 CNN卷积神经网络的基本定义理解

参考链接：https://my.oschina.net/yilian/blog/661218

首先 上面讲了神经元 Y=WX+B ，通过输入的参数X ===========》Y 深度学习 每一个batch来说 其实就是 多项公式



我们知道 在数学里面 求多项公式 其实 就是 矩阵 W 矩阵 乘与 X 加上 B 矩阵 = Y矩阵 ，矩阵 二元数组在tensorflow 也是一个tensor ndarray , 通常 我们知道 因为relu 收敛效果要比sigmod 与tanh 要好，所以在cnn中常用relu，所以 其实 对于输出o=relu(wx+b)。

1. TensorFlow卷积

比如 下面 是tensorflow卷积定义 relu (W\*X+B) W 矩阵 \* X矩阵 + B矩阵 = W 权重variable变量 \* X (placeholder占位符外部输入)variable变量 + B偏重变量，因为深度学习 会自动 不断地计算loss损失 BP 来调整 w b 所以 w b初始化 可以随便 全部都是0 都行 ，所以 其实 就是 X 以及 Y 对于 X来说其实我们知道 就是 我们图像数据 Y 是图像的标签，但是Y需要转为数学可以计算的值，所以采用one-hot 数组记录 标签的索引就行，比如 xx1 xx2 xx3 相应的y1=[1,0,0] y2=[0 1 0] y3=[0 0 1]

那么 其实 就是 X 图像的像素 通过外部输入 placeholder占位符 Y值 外部输入 通过 placeholder占位符

我们知道 W\*X 矩阵 相乘 必须 符合 MXN NXM =MXM 也就是说W的 列 必须 与 X的行数目相同 这是要注意的，所以上一张shape来规范维度计算 ，下面是一个卷积层 定义 relu(wx+b)  下面是tensorflow来表示relu(wx+b)的公式

其中要注意参数 strides 是卷积滑动的步长 你可以配置更多的系数

def conv2d(x, w, b):

return tf.nn.relu(tf.nn.bias\_add(tf.nn.conv2d(x, w, strides=[1, 1, 1, 1], padding='SAME'),b))

上面我们理解了， 其实就是 W矩阵 \* X矩阵 +B 矩阵 =Y矩阵Y矩阵 好算 ，W矩阵 +B矩阵 已知 可以随意后面要bp调整。那么X矩阵 是图像 也就是说输入的X。

x = tf.placeholder(tf.float32, [None, w\*h])

#w\*h 因为批次训练 数据可以任意所以第一个是None ，第二个图像是个w \* h的图像 ，可以展开得到 w \* h 的数组

我们知道 卷积运算 其实是卷积的个数 M 展开的卷积核长宽维度 K \* 卷积能够提取的数目 K \* 提取图像的维度N =MXN。也就说可以通过 卷积核 让图像 W\*H 转变成 一个卷积的个数的行，N 为提取图像卷积特征的每一行数据的的矩阵还是遵循WX+B 。

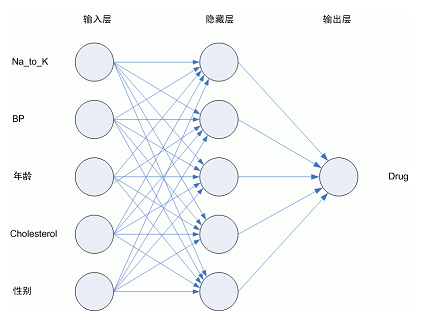
我们定义好W我们可以把整个网络看成wx+b 忽略掉那些relu 等等其实就是一个y=wx+b 我们知道输入是上面的

x = tf.placeholder(tf.float32, [None, w\*h])

y = tf.placeholder(tf.float32, [None, ysize]) #y的数目个数 比如3个类 就是3

x\*W = y 计入batch为50 y为10 那么[50,224\*224] \* W= [50 ,10]

那么W 需要是一个[224\*224，10] 的矩阵 ，也就是说 至少224\*224\*10\*50 个连接



X [None ,w\*h] 对于每一个 w\*h 是一个矩阵 每一层的w 也是一个矩阵 每一层的 b也是一个矩阵 ，每一层的输出y1 也是一个矩阵 y=[w\*h]\*w+b 为了减少系数，我们使用卷积，把它转换成MXN的值 ，这里就是跟全连接层的不同，使用了卷积转换成了一个MXN的卷积特征 而全连接层就是 y=wx+b (这里省略了那些relu(wx+b) tanh(wx+b))

因为卷积层的w是需要 与 w\*h 提取的MXK来做矩阵相乘 所以 他是跟卷积核相关 以及 输入 输出 相关，对于每一张图像。输入为1 输出分解为多个输出用于第二层运算。

wc=tf.Variable(tf.random\_normal([3, 3, 1, 64])

#3 3 分别为3x3大小的卷积核 1位输入数目 因为是第一层所以是1 输出我们配置的64 所以我们知道了 如果下一次卷积wc2=[5,5,64,256] //5x5 是我们配置的卷积核大小，第三位表示输入数目 我们通过上面知道 上面的输出 也就是下一层的输入 所以 就是64 了输出我们定义成256 这个随你喜好，关键要迭代看效果，一般都是某一个v\*v的值

这样我们知道了 wc\*(卷积得到的x) + b =y1下一层输入，我们也能知道 一下层输入 mxnxo 分别为输出O个卷积核特征MXN 我们也能知道b 大小 那个肯定跟O一样，比如上面是64输出，所以b1=tf.Variable(tf.random\_normal([64]))，同样可以知道b2=[256]。

### 2.2 实列

#### 2.2.1 mnist识别实例

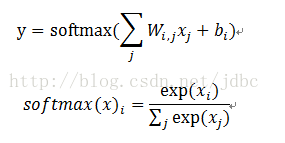
介绍了一些tensorflow基础后，我们用一个完整的例子将这些串起来。首先，需要下载数据集，mnist数据可以在Yann LeCun's website（http://yann.lecun.com/exdb/mnist/）下载到，也可以通过如下两行代码得到。

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

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

该数据集中一共有55000个样本，其中50000用于训练，5000用于验证。每个样本分为X与y两部分，其中X如下图所示，是28\*28的图像，在使用时需要拉伸成28\*28=784维的向量。

用图形可以表示为下图，具体原理这里不再阐述，更多细节参考该链接(https://tech.meituan.com/2015/05/08/intro-to-logistic-regression.html)。



当使用tensorflow进行graph构建时，大体可以分为五部分：

1、为输入X与输出y定义placeholder；

2、定义权重W；

3、定义模型结构；

4、定义损失函数；

5、定义优化算法。

首先导入需要的包，定义X与y的placeholder以及 W,b 的 Variables。其中None表示任意维度，一般是min-batch的 batch size。而 W 定义是 shape 为784,10，rank为2的Variable，b是shape为10，rank为1的Variable。Rank代表维度一三维[]个数。Shape是一维个数[]里面个数。

import tensorflow as tf

x = tf.placeholder(tf.float32, [None, 784])

y\_ = tf.placeholder(tf.float32, [None, 10])

W = tf.Variable(tf.zeros([784, 10]))

b = tf.Variable(tf.zeros([10]))

之后是定义模型。x与W矩阵乘法后与b求和，经过softmax得到y。

y = tf.nn.softmax(tf.matmul(x, W) + b)

求逻辑回归的损失函数，这里使用了cross entropy，其公式可以表示为：



这里的 cross entropy 取了均值。定义了学习步长为0.5，使用了梯度下降算法（GradientDescentOptimizer）最小化损失函数。不要忘记初始化 Variables。

cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(y\_\*tf.log(y),reduction\_indices=[1]))

train\_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross\_entropy)

init = tf.global\_variables\_initializer()

最后，我们的 graph 至此定义完毕，下面就可以进行真正的计算，包括初始化变量，输入数据，并计算损失函数与利用优化算法更新参数。

with tf.Session() as sess:

sess.run(init)

for i in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step, feed\_dict={x: batch\_xs, y\_: batch\_ys})

其中，迭代了1000次，每次输入了100个样本。mnist.train.next\_batch 就是生成下一个 batch 的数据，这里知道它在干什么就可以。那么训练结果如何呢，需要进行评估。这里使用单纯的正确率，正确率是用取最大值索引是否相等的方式，因为正确的 label 最大值为1，而预测的 label 最大值为最大概率。

correct\_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y\_,1))

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

print(sess.run(accuracy, feed\_dict={x: mnist.test.images, y\_: mnist.test.labels}))