第三章 深度学习基础

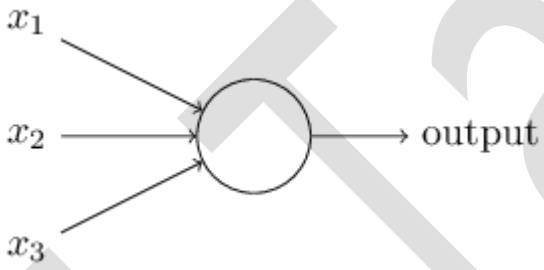
3.1 基本概念

3.1.1 神经网络组成？

为了描述神经网络，我们先从最简单的神经网络说起。

感知机

简单的感知机如下图所示：



其输出 $output = \begin{cases} 0, if \sum_i w_i x_i \le threshold \\ 1, if \sum_i w_i x_i > threshold \end{cases}$。

假如把感知机想象成一个加权投票机制，比如 3 位评委给一个歌手打分，打分分别为 4 分、1 分、-3 分，这 3 位评分的权重分别是 1、3、2，则该歌手最终得分为

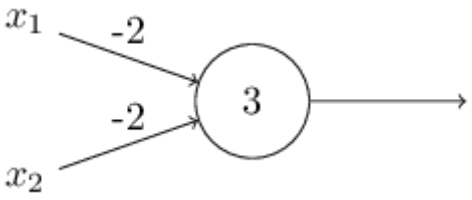
4*1+1*3+（−3）*2=1。按照比赛规则，选取的 threshold 为 3，说明只有歌手的综合评分大于 3 时，才可顺利晋级。对照感知机，该选手被淘汰，因为

$$\sum_i w_i x_i < threshold = 3, output = 0$$

用—b 代替 threshold。输出变为：

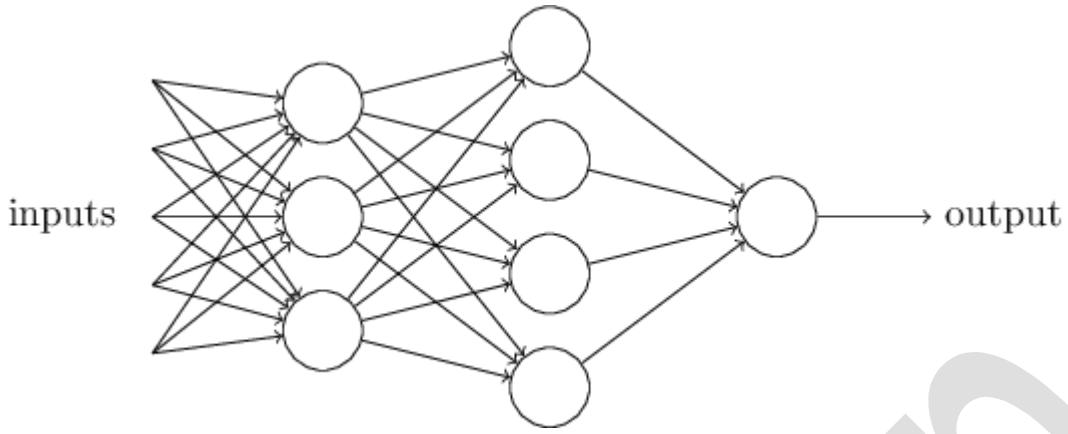
$$output = \begin{cases} 0, if \quad w \cdot x + b \le 0 \\ 1, if \quad w \cdot x + b > 0 \end{cases}$$

设置合适的 w 和 b，一个简单的感知机单元的与非门表示如下：



当输入为 0,1 时，感知机输出为 0*（-2）+1*（-2）+3=1。

复杂一些的感知机由简单的感知机单元组合而成：



Sigmoid 单元

感知机单元的输出只有 0 和 1，实际情况中，更多的输出类别不止 0 和 1，而是[0,1]上的概率值，这时候就需要 sigmoid 函数把任意实数映射到[0,1]上。

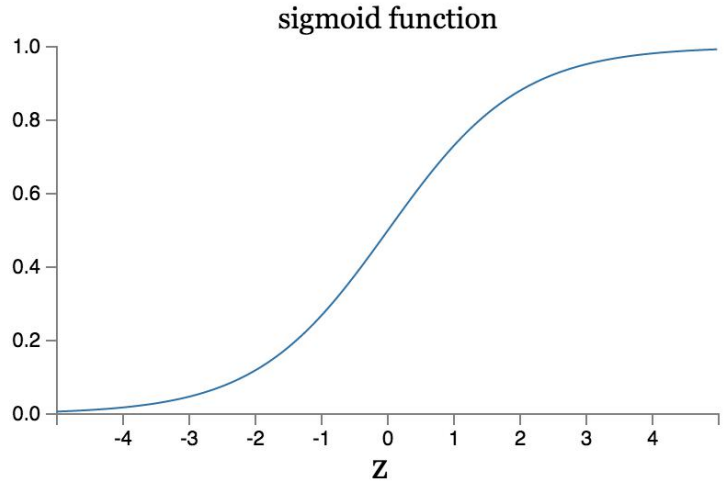
神经元的输入

$$z=\sum_i w_i x_i + b$$

假设神经元的输出采用 sigmoid 激活函数

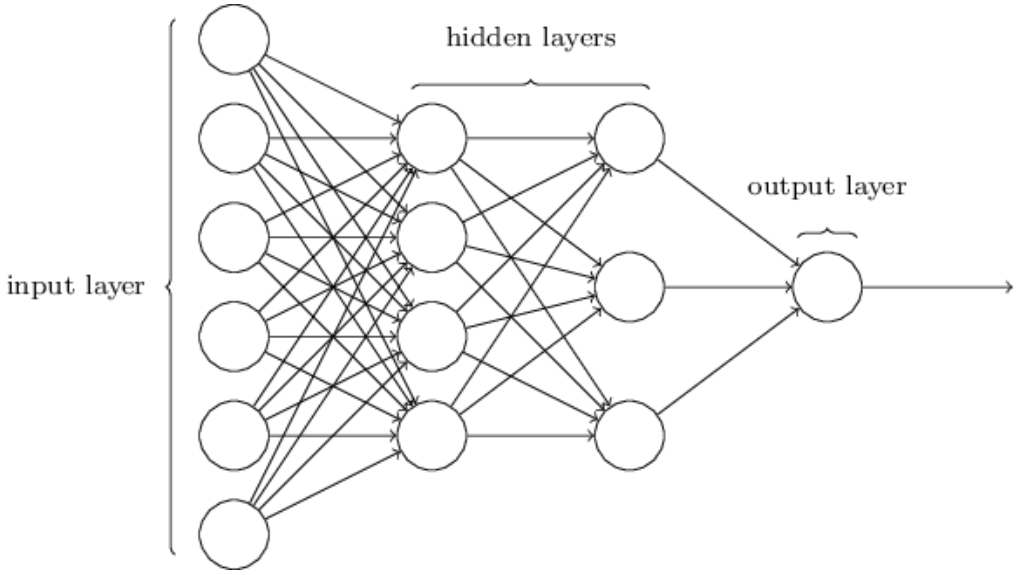
$$\sigma(z)=\frac{1}{1+e^{-z}}$$

sigmoid 激活函数图像如下图所示：

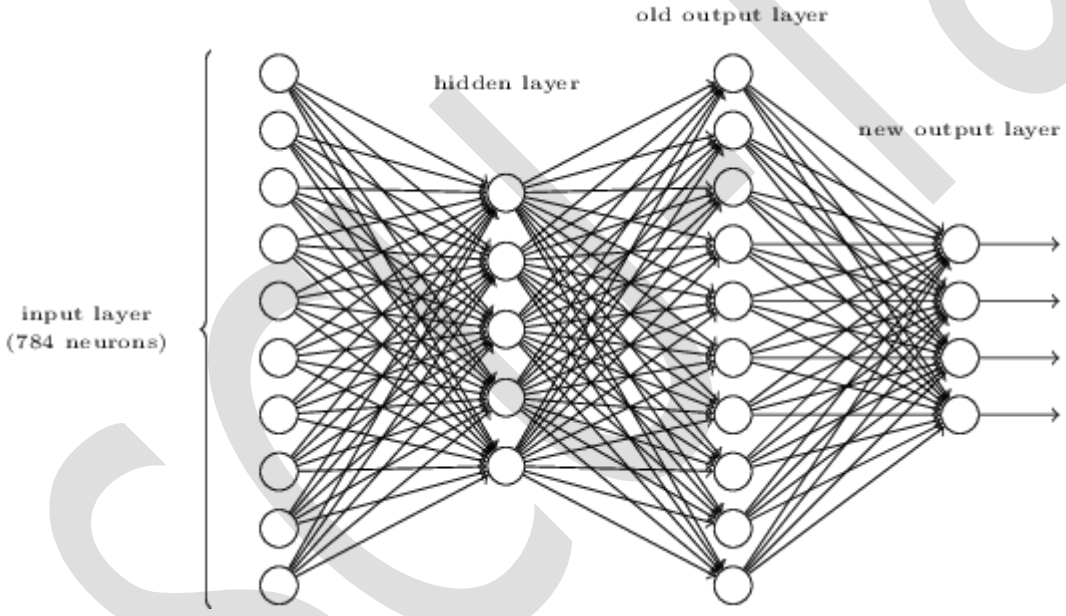


全连接神经网络

即第 i 层的每个神经元和第 i-1 层的每个神经元都有连接。



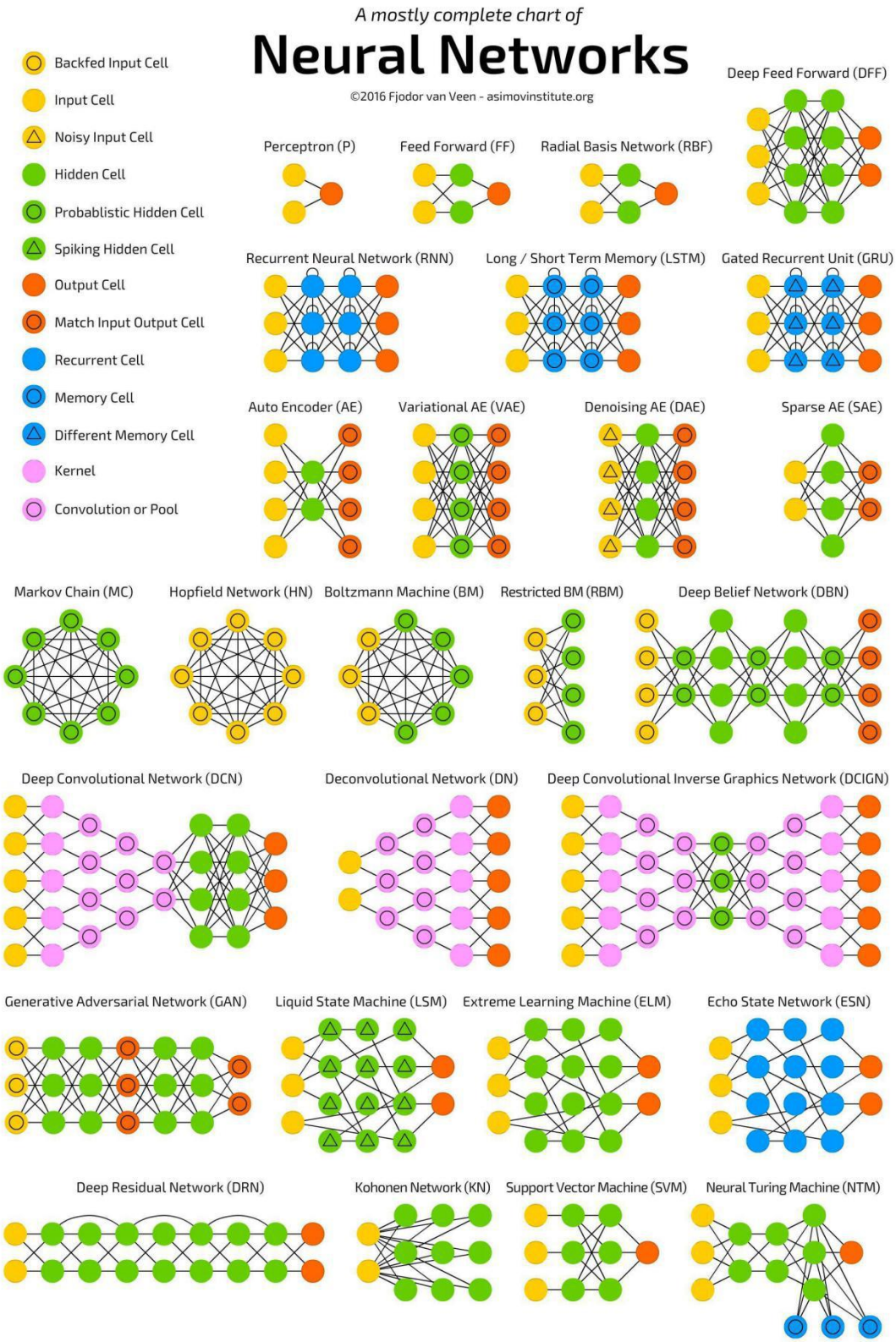
输出层可以不止有 1 个神经元。隐藏层可以只有 1 层，也可以有多层。输出层为多个神经元的神经网络例如下图：



3.1.2 神经网络有哪些常用模型结构？

下表包含了大部分常用的模型

https://blog.csdn.net/nicholas_liu2017/article/details/73694666


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

3.1.3 如何选择深度学习开发平台？

现有的深度学习开源平台主要有 Caffe, Torch, MXNet, CNTK, Theano, TensorFlow, Keras 等。那如何选择一个适合自己的平台呢，下面列出一些衡量做参考。

参考 1：与现有编程平台、技能整合的难易程度

主要是前期积累的开发经验和资源，比如编程语言，前期数据集存储格式等。

参考 2: 与相关机器学习、数据处理生态整合的紧密程度

深度学习研究离不开各种数据处理、可视化、统计推断等软件包。考虑建模之前，是否具有方便的数据预处理工具？建模之后，是否具有方便的工具进行可视化、统计推断、数据分析？

参考 3：对数据量及硬件的要求和支持

深度学习在不同应用场景的数据量是不一样的，这也就导致我们可能需要考虑分布式计算、多 GPU 计算的问题。例如，对计算机图像处理研究的人员往往需要将图像文件和计算任务分部到多台计算机节点上进行执行。当下每个深度学习平台都在快速发展，每个平台对分布式计算等场景的支持也在不断演进。

参考 4：深度学习平台的成熟程度

成熟程度的考量是一个比较主观的考量因素，这些因素可包括：社区的活跃程度；是否容易和开发人员进行交流；当前应用的势头。

参考 5：平台利用是否多样性？

有些平台是专门为深度学习研究和应用进行开发的，有些平台对分布式计算、GPU 等构架都有强大的优化，能否用这些平台/软件做其他事情？比如有些深度学习软件是可以用来求解二次型优化；有些深度学习平台很容易被扩展，被运用在强化学习的应用中。

3.1.4 为什么使用深层表示

1、深度神经网络的多层隐藏层中，前几层能学习一些低层次的简单特征，后几层能把前面简单的特征结合起来，去学习更加复杂的东西。比如刚开始检测到的是边缘信息，而后检测更为细节的信息。

2、深层的网络隐藏单元数量相对较少，隐藏层数目较多，如果浅层的网络想要达到同样的计算结果则需要指数级增长的单元数量才能达到。

3.1.5 为什么深层神经网络难以训练？

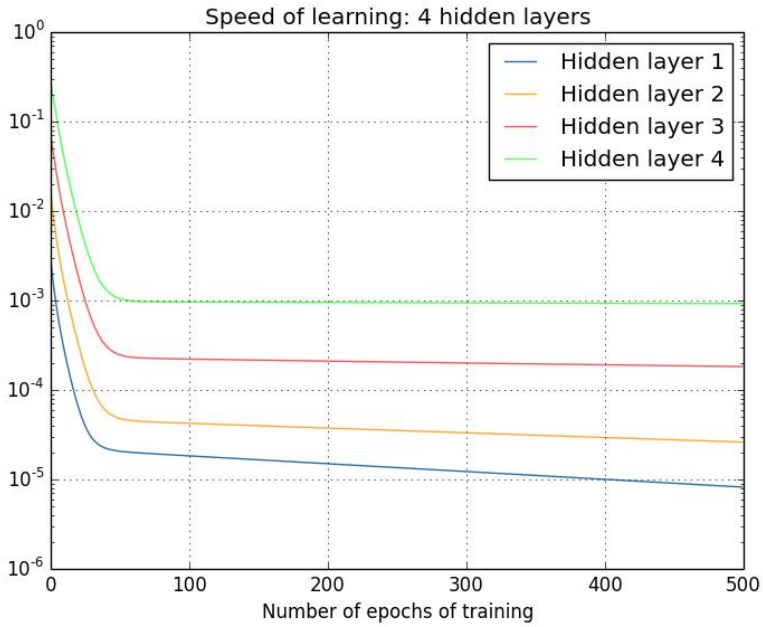
https://blog.csdn.net/BinChasing/article/details/50300069

http://mini.eastday.com/mobile/180116023302833.html

《Why is it hard to train deep neural networks? Degeneracy, not vanishing gradients, is the key》

1、梯度消失

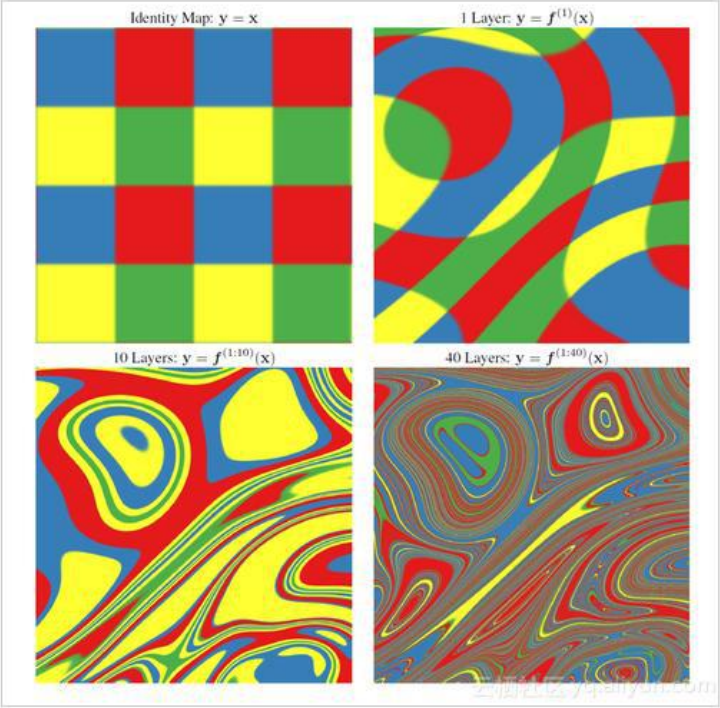
梯度消失是指通过隐藏层从后向前看，梯度会变的越来越小，说明前面层的学习会显著慢于后面层的学习，所以学习会卡住，除非梯度变大。下图是不同隐含层的学习速率。



2、梯度爆炸

3、权重矩阵的退化导致模型的有效自由度减少。参数空间中学习的退化速度减慢，导致减少了模型的有效维数，网络的可用自由度对学习中梯度范数的贡献不均衡，随着相乘矩阵的数量（即网络深度）的增加，矩阵的乘积变得越来越退化。

在有硬饱和边界的非线性网络中（例如 ReLU 网络），随着深度增加，退化过程会变得越来越快。Duvenaud 等人 2014 年的论文里展示了关于该退化过程的可视化：

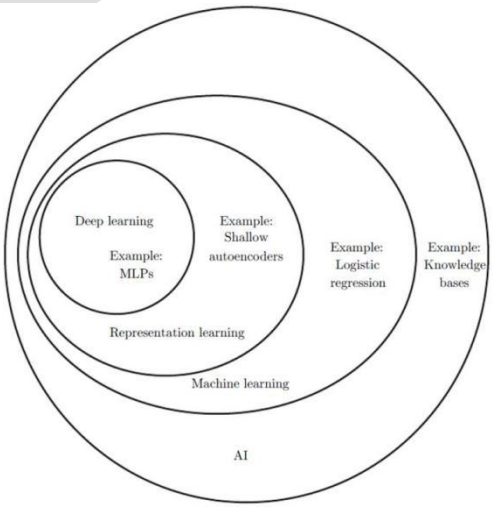


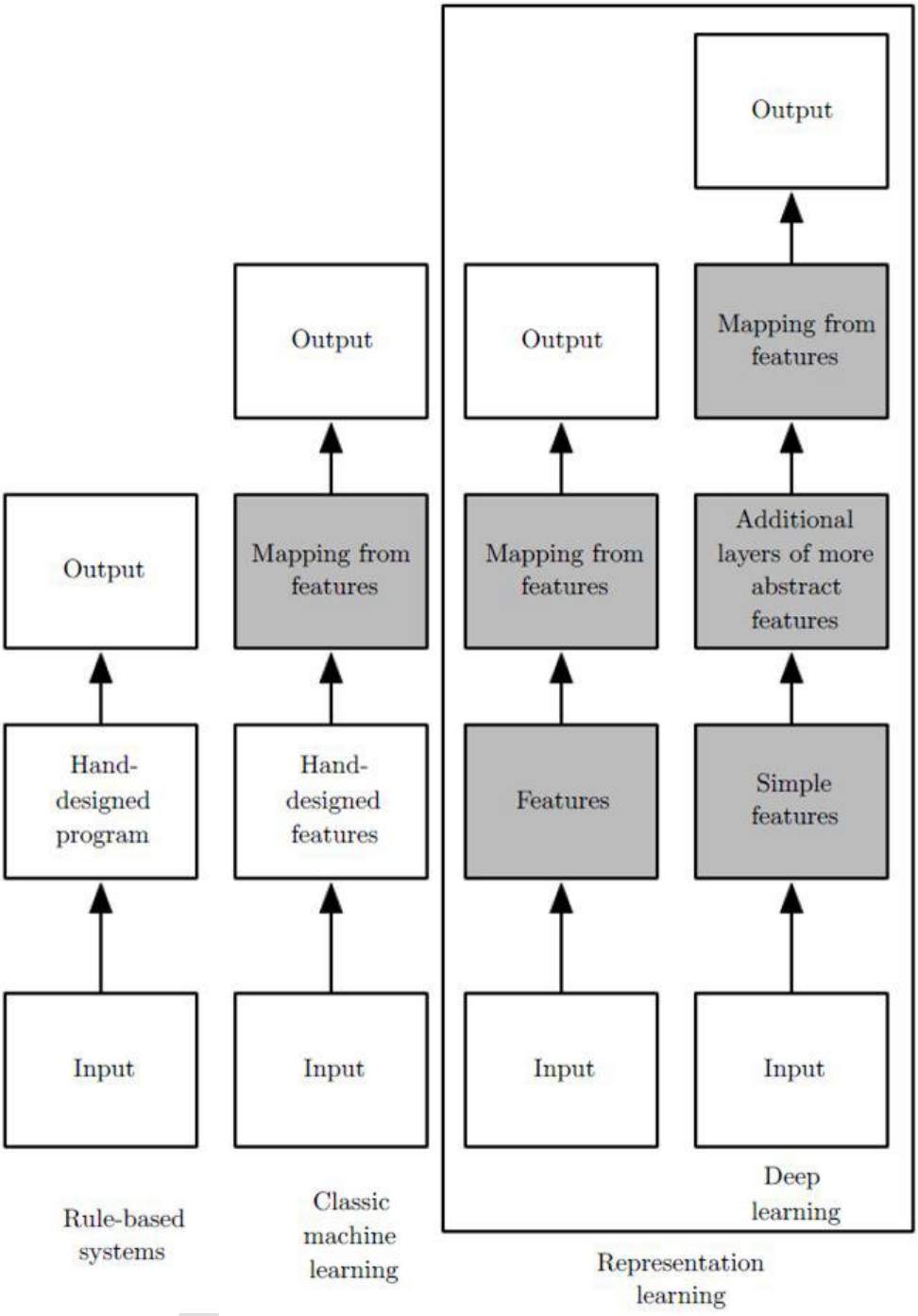
随着深度的增加，输入空间（左上角所示）会在输入空间中的每个点处被扭曲成越来越细的单丝，只有一个与细丝正交的方向影响网络的响应。沿着这个方向，网络实际上对变化变得非常敏感。

3.1.6 深度学习和机器学习有什么不同

机器学习：利用计算机、概率论、统计学等知识，输入数据，让计算机学会新知识。机器学习的过程，就是通过训练数据寻找目标函数。

深度学习是机器学习的一种，现在深度学习比较火爆。在传统机器学习中，手工设计特征对学习效果很重要，但是特征工程非常繁琐。而深度学习能够从大数据中自动学习特征，这也是深度学习在大数据时代受欢迎的一大原因。





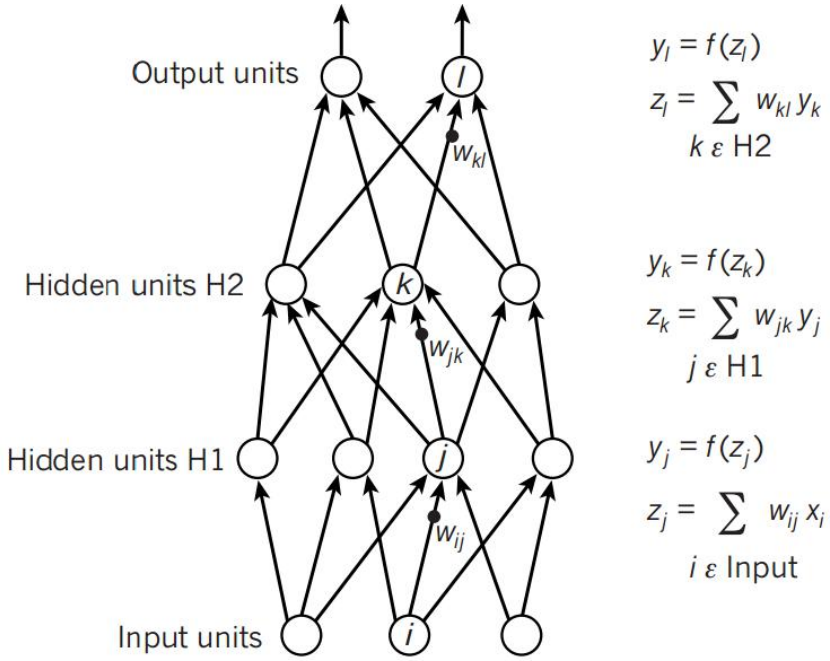
3.2 网络操作与计算

3.2.1 前向传播与反向传播？

https://blog.csdn.net/lhanchao/article/details/51419150

在神经网络的计算中，主要由前向传播(foward propagation，FP)和反向传播(backward propagation，BP)。

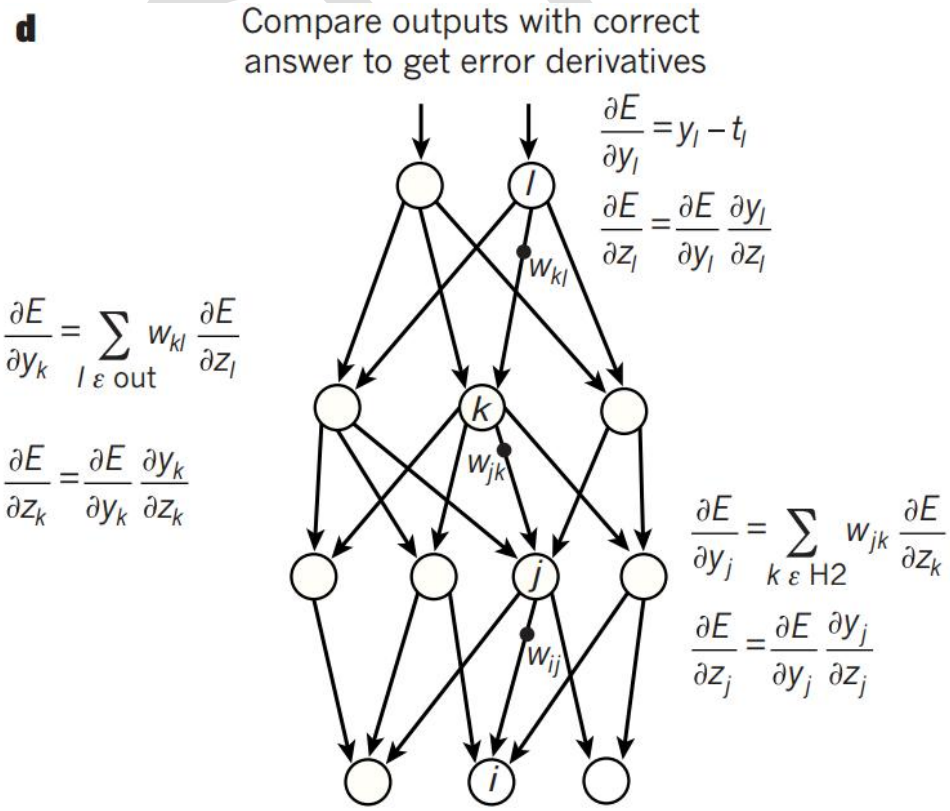
前向传播



假设上一层结点 i,j,k,…等一些结点与本层的结点 w 有连接，那么结点 w 的值怎么算呢？就是通过上一层的 i,j,k 等结点以及对应的连接权值进行加权和运算，最终结果再加上一个偏置项（图中为了简单省略了），最后在通过一个非线性函数（即激活函数），如 ReLu，sigmoid 等函数，最后得到的结果就是本层结点 w 的输出。

最终不断的通过这种方法一层层的运算，得到输出层结果。

反向传播



由于我们前向传播最终得到的结果，以分类为例，最终总是有误差的，那么怎么减少误差

呢，当前应用广泛的一个算法就是梯度下降算法，但是求梯度就要求偏导数，下面以图中字母为例讲解一下。

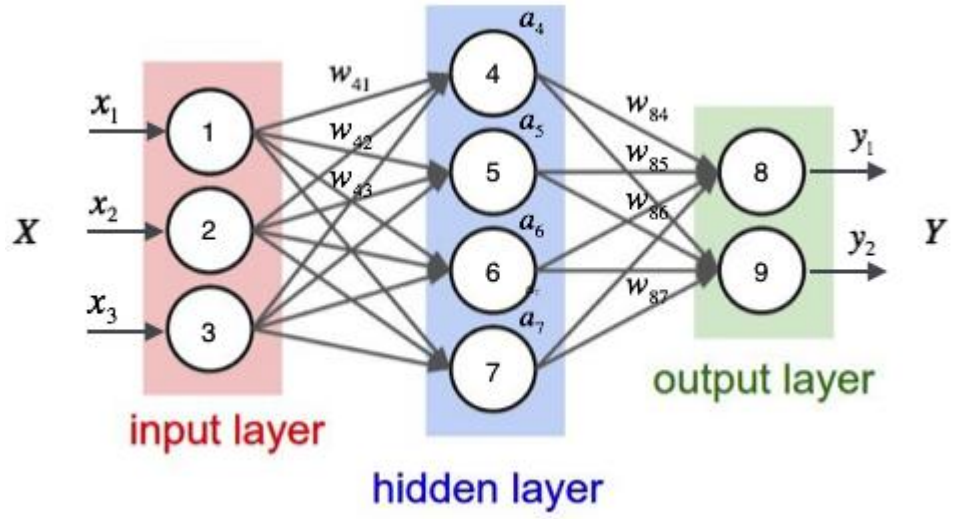
由于我们前向传播最终得到的结果，以分类为例，最终总是有误差的，那么怎么减少误差呢，当前应用广泛的一个算法就是梯度下降算法，但是求梯度就要求偏导数，下面以图中字母为例讲解一下。

设最终总误差为E，对于输出那么E#对于输出结点y_l的偏导数是$y_l - t_l$，其中t_l是真实值$\frac{\partial y_l}{\partial z_l}$是指上面提到的激活函数，$z_l$是上面提到的加权和，那么这一层的$E$对$z_l$的偏导数为$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$。同理，下一层也是这么计算，（只不过$\frac{\partial E}{\partial y_k}$计算方法变了），一直反向传播到输入层，最后有$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$且$\frac{\partial z_j}{\partial x_i} = w_ij$

然后调整这些过程中的权值，再不断进行前向传播和反向传播的过程，最终得到一个比较好的结果

3.2.2 如何计算神经网络的输出？

原文地址：https://www.zybuluo.com/hanbingtao/note/476663



如上图，输入层有三个节点，我们将其依次编号为 1、2、3；隐藏层的 4 个节点，编号依次为 4、5、6、7；最后输出层的两个节点编号为 8、9。比如，隐藏层的节点 4，它和输入层的三个节点 1、2、3 之间都有连接，其连接上的权重分别为是 w_{41}, w_{42}, w_{43}。

为了计算节点 4 的输出值，我们必须先得到其所有上游节点（也就是节点 1、2、3）的输出值。节点 1、2、3 是输入层的节点，所以，他们的输出值就是输入向量本身。按照上图画出的对应关系，可以看到节点 1、2、3 的输出值分别是 x_1, x_2, x_3。

$$a_4 = sigmoid\left(\vec{w}^T \cdot \vec{x}\right) = sigmoid\left(w_{41}x_1 + w_{42}x_2 + w_{43}x_3 + w_{4b}\right)$$

其中 w_{4b} 是节点 4 的偏置项

同样，我们可以继续计算出节点 5、6、7 的输出值 a5,a6,a7。

计算输出层的节点 8 的输出值 y_1：

$$y_1 = sigmoid\left(\vec{w}^T \cdot \vec{a}\right) = sigmoid\left(w_{84}a_4 + w_{85}a_5 + w_{86}a_6 + w_{87}a_7 + w_{8b}\right)$$

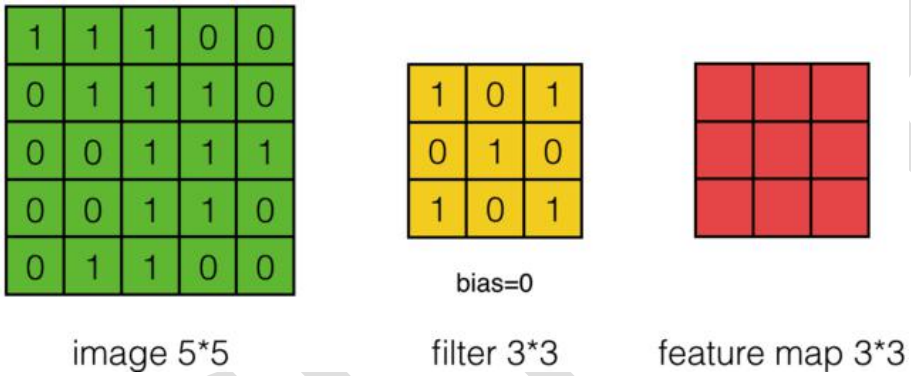
其中 w_{8b} 是节点 8 的偏置项。

同理，我们还可以计算出 y_2。这样输出层所有节点的输出值计算完毕，我们就得到了在输入向量 x_1, x_2, x_3, x_4 时，神经网络的输出向量 y_1， y_2。这里我们也看到，输出向量的维度和输出层神经元个数相同。

3.2.3 如何计算卷积神经网络输出值？

https://www.zybuluo.com/hanbingtao/note/485480

假设有一个 5*5 的图像，使用一个 3*3 的 filter 进行卷积，想得到一个 3*3 的 Feature Map，如下所示：



$x_{i,j}$ 表示图像第 i 行第 j 列元素。$w_{m,n}$ 表示 filter 第 m 行第 n 列权重。w_b 表示 filter 的偏置项。$a_{i,j}$ 表示 feature map 第 i 行第 j 列元素。 f 表示激活函数，这里以 relu 函数为例。

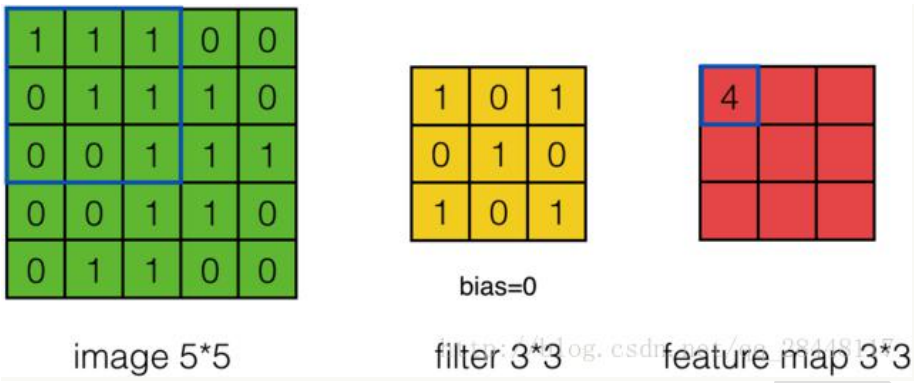
卷积计算公式如下：

$$a_{i,j} = f\left(\sum_{m=0}^{2}\sum_{n=0}^{2} w_{m,n}x_{i+m,j+n} + w_b\right)$$

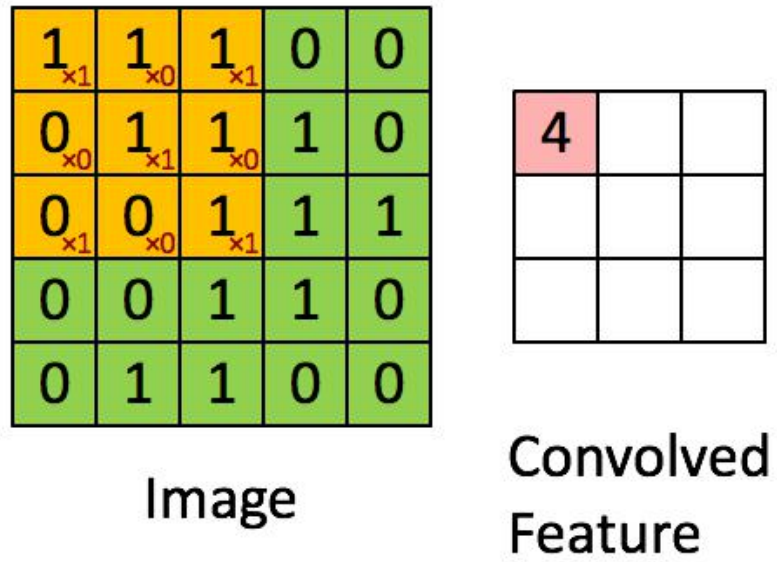
当步长为 1 时，计算 feature map 元素 $a_{0,0}$ 如下：

$$\begin{aligned}
a_{0,0} &= f\left(\sum_{m=0}^{2}\sum_{n=0}^{2} w_{m,n}x_{0+m,0+n} + w_b\right) \\
&= relu\left(w_{0,0}x_{0,0} + w_{0,1}x_{0,1} + w_{0,2}x_{0,2} + w_{1,0}x_{1,0} + w_{1,1}x_{1,1} + w_{1,2}x_{1,2}\right. \\
&\quad \left. + w_{2,0}x_{2,0} + w_{2,1}x_{2,1} + w_{2,2}x_{2,2}\right) \\
&= 1+0+1+0+1+0+0+0+1 \\
&= 4
\end{aligned}$$

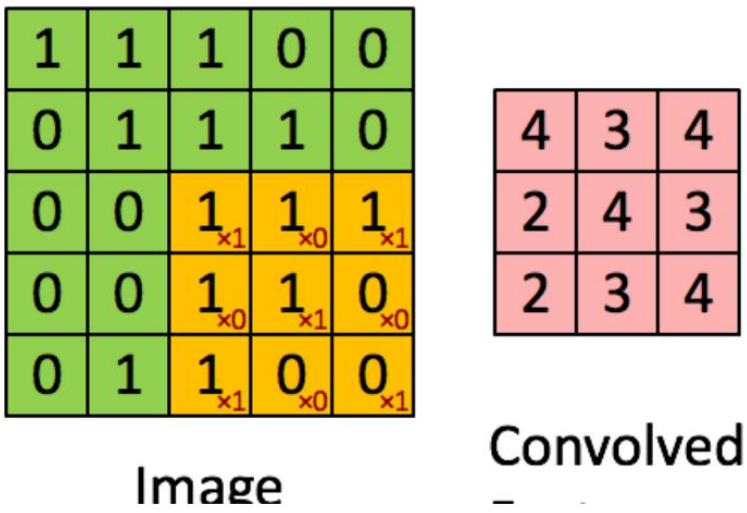
结果如下：



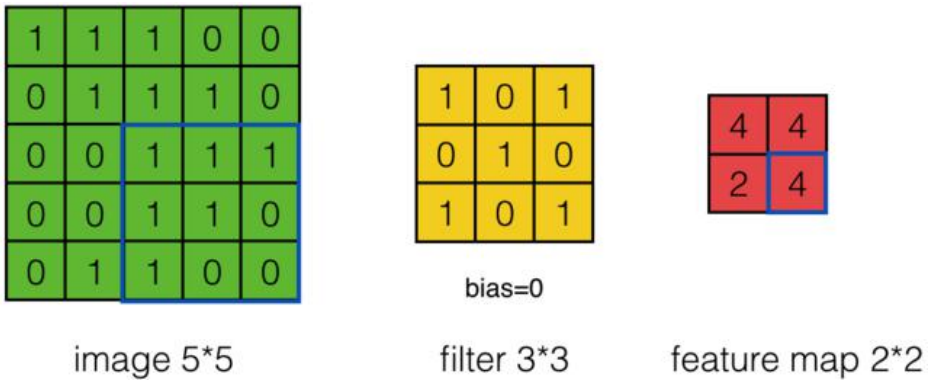
其计算过程图示如下：



以此类推，计算出全部的 Feature Map。



当步幅为 2 时，Feature Map 计算如下



注：图像大小、步幅和卷积后的 Feature Map 大小是有关系的。它们满足下面的关系：

$$W_2 = (W_1 - F + 2P)/S + 1$$
$$H_2 = (H_1 - F + 2P)/S + 1$$

其中，W_2 是卷积后 Feature Map 的宽度；W_1 是卷积前图像的宽度；F 是 filter 的宽度；P 是 Zero Padding 数量，Zero Padding 是指在原始图像周围补几圈 0，如果 P 的值是 1，那么就补 1 圈 0；S 是步幅；H_2 卷积后 Feature Map 的高度；H_1 是卷积前图像的宽度。

举例：假设图像宽度 $W_1 = 5$，filter 宽度 F=3，Zero Padding P=0，步幅 S=2，Z 则

$$W_2 = (W_1 - F + 2P)/S + 1$$
$$= (5 - 3 + 0)/2 + 1$$
$$= 2$$

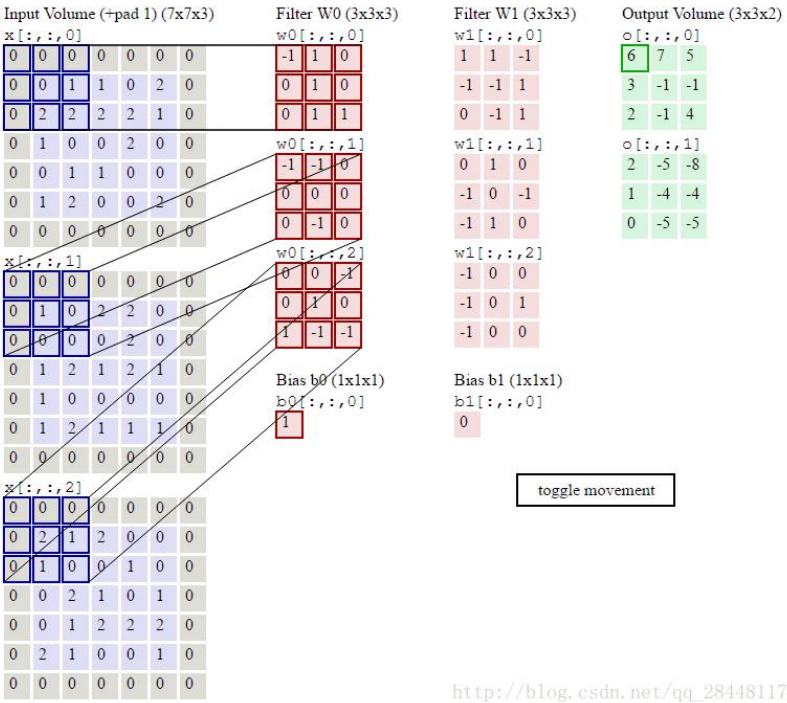
说明 Feature Map 宽度是 2。同样，我们也可以计算出 Feature Map 高度也是 2。

如果卷积前的图像深度为 D，那么相应的 filter 的深度也必须为 D。深度大于 1 的卷积计算公式：

$$a_{i,j} = f\left(\sum_{d=0}^{D-1}\sum_{m=0}^{F-1}\sum_{n=0}^{F-1} w_{d,m,n} x_{d,i+m,j+n} + w_b\right)$$

其中，D 是深度；F 是 filter 的大小；$w_{d,m,n}$ 表示 filter 的第 d 层第 m 行第 n 列权重；$a_{d,i,j}$ 表示 feature map 的第 d 层第 i 行第 j 列像素；其它的符号含义前面**相同**，不再赘述。

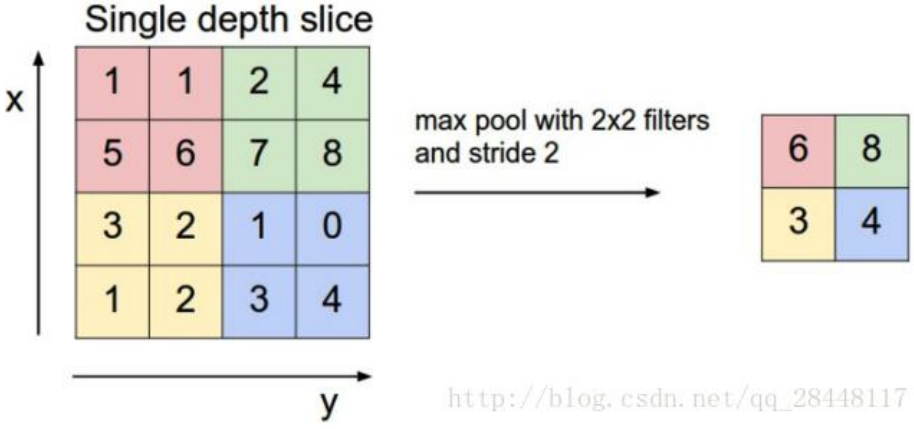
每个卷积层可以有多个 filter。每个 filter 和原始图像进行卷积后，都可以得到一个 Feature Map。卷积后 Feature Map 的深度(个数)和卷积层的 filter 个数是相同的。下面的图示显示了包含两个 filter 的卷积层的计算。7*7*3 输入，经过两个 3*3*3filter 的卷积(步幅为 2)，得到了 3*3*2 的输出。图中的 Zero padding 是 1，也就是在输入元素的周围补了一圈 0。Zero padding 对于图像边缘部分的特征提取是很有帮助的。



以上就是卷积层的计算方法。这里面体现了局部连接和权值共享：每层神经元只和上一层部分神经元相连(卷积计算规则)，且 filter 的权值对于上一层所有神经元都是一样的。对于包含两个 3*3*3 的 fitler 的卷积层来说，其参数数量仅有(3*3*3+1)*2=56 个，且参数数量与上一层神经元个数无关。与全连接神经网络相比，其参数数量大大减少了。

3.2.4 如何计算 Pooling 层输出值输出值？

Pooling 层主要的作用是下采样，通过去掉 Feature Map 中不重要的样本，进一步减少参数数量。Pooling 的方法很多，最常用的是 Max Pooling。Max Pooling 实际上就是在 n*n 的样本中取最大值，作为采样后的样本值。下图是 2*2 max pooling：



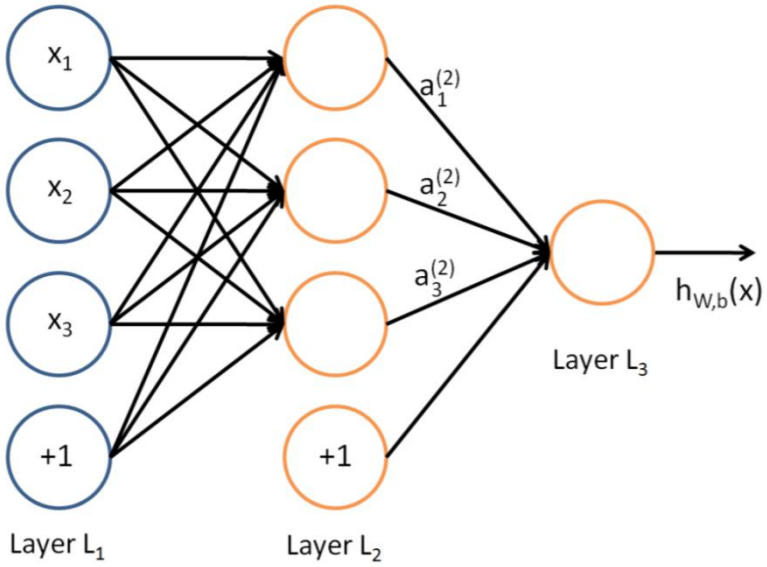
除了 Max Pooing 之外，常用的还有 Mean Pooling——取各样本的平均值。

对于深度为 D 的 Feature Map，各层独立做 Pooling，因此 Pooling 后的深度仍然为 D。

3.2.5 实例理解反向传播

http://www.cnblogs.com/charlotte77/p/5629865.html

一个典型的三层神经网络如下所示：

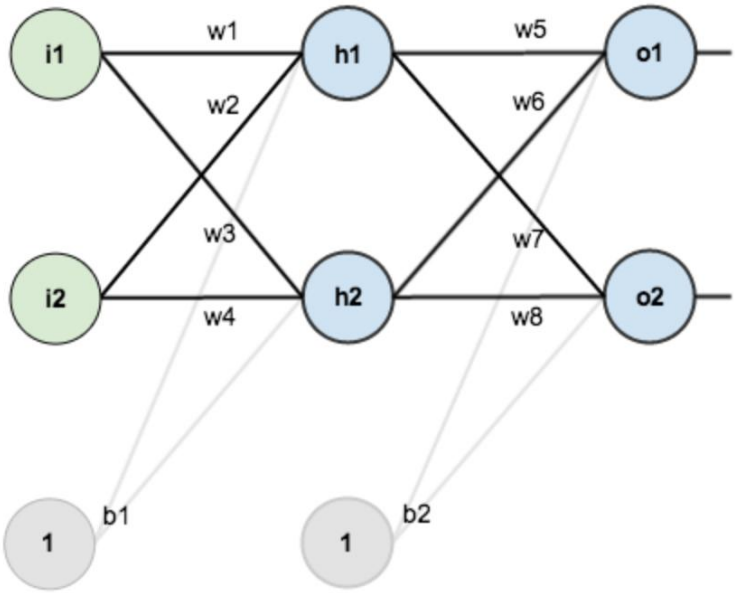


其中 Layer L_1 是输入层，Layer L_2 是隐含层，Layer L_3 是输出层。

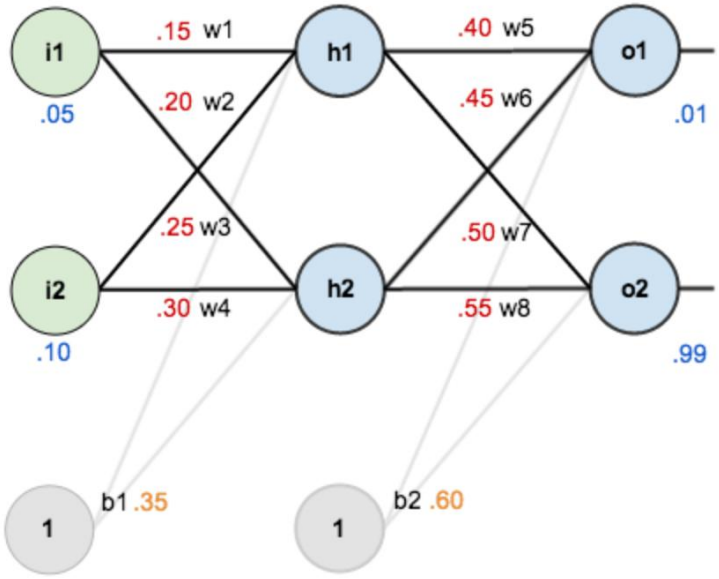
假设输入数据集为 $D=\{x_1,x_2,\cdots,x_n\}$，输出数据集为 $D'=\{y_1,y_2,\cdots,y_n\}$。

如果输入和输出是一样，即为自编码模型。如果原始数据经过映射，会得到不同与输入的输出。

假设有如下的网络层：



输入层包含神经元i_1, i_2，偏置b_1；隐含层包含神经元h_1, h_2，偏置b_2，输出层为o_1, o_2，w_i层与层之间连接的权重，激活函数为 sigmoid 函数。对以上参数取初始值，如下图所示：



其中，输入数据 $i_1 = 0.05$， $i_2 = 0.10$;

输出数据 o1=0.01,o2=0.99;

初始权重 w1=0.15,w2=0.20,w3=0.25,w4=0.30;

w5=0.40,w6=0.45,w7=0.50,w8=0.55

目标：给出输入数据 i1,i2(0.05 和 0.10)，使输出尽可能与原始输出 o1,o2(0.01 和 0.99)接近。

前向传播

1 输入层——>输出层

计算神经元 h1 的输入加权和：

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

神经元 h1 的输出 o1:(此处用到激活函数为 sigmoid 函数)：

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

同理，可计算出神经元 h2 的输出 o2：

$$out_{h2} = 0.596884378$$

2. 隐含层---->输出层：

计算输出层神经元 o1 和 o2 的值：

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

这样前向传播的过程就结束了，我们得到输出值为[0.75136079 , 0.772928465]，与实际值[0.01 , 0.99]相差还很远，现在我们对误差进行反向传播，更新权值，重新计算输出。

反向传播

1. 计算总误差

总误差：(square error)

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

但是有两个输出，所以分别计算 o1 和 o2 的误差，总误差为两者之和：

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

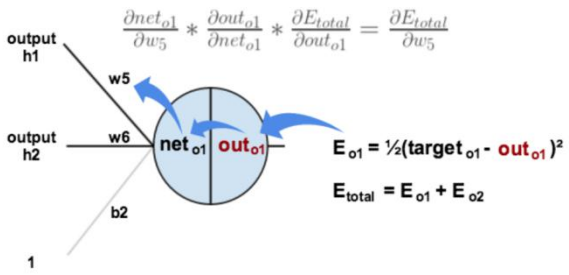
$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

2. 隐含层---->输出层的权值更新：

以权重参数 w5 为例，如果我们想知道 w5 对整体误差产生了多少影响，可以用整体误差对 w5 求偏导求出：（链式法则）

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

下面的图可以更直观的看清楚误差是怎样反向传播的：



http://www.cnblogs.com/charlotte77/p/5629865.html

3.3 超参数

3.3.1 什么是超参数？

超参数:比如算法中的 learning rate （学习率）、iterations(梯度下降法循环的数量)、（隐藏层数目）、（隐藏层单元数目）、choice of activation function（激活函数的选择）都需要根据实际情况来设置，这些数字实际上控制了最后的参数和的值，所以它们被称作超参数。

3.3.2 如何寻找超参数的最优值？

在使用机器学习算法时，总有一些难搞的超参数。例如权重衰减大小，高斯核宽度等等。算法不会设置这些参数，而是需要你去设置它们的值。设置的值对结果产生较大影响。常见设置超参数的做法有：

1、猜测和检查：根据经验或直觉，选择参数，一直迭代。

2、网格搜索：让计算机尝试在一定范围内均匀分布的一组值。

3、随机搜索：让计算机随机挑选一组值。

4、贝叶斯优化：使用贝叶斯优化超参数，会遇到贝叶斯优化算法本身就需要很多的参数的困难。

5、在良好初始猜测的前提下进行局部优化：这就是 MITIE 的方法，它使用 BOBYQA 算法，并有一个精心选择的起始点。由于 BOBYQA 只寻找最近的局部最优解，所以这个方法是否成功很大程度上取决于是否有一个好的起点。在 MITIE 的情况下，我们知道一个好的起点，

但这不是一个普遍的解决方案，因为通常你不会知道好的起点在哪里。从好的方面来说，这种方法非常适合寻找局部最优解。稍后我会再讨论这一点。

6、最新提出的 LIPO 的全局优化方法。这个方法没有参数，而且经验证比随机搜索方法好。

3.3.3 超参数搜索一般过程？

超参数搜索一般过程：

1、将数据集划分成训练集、验证集及测试集。

2、在训练集上根据模型的性能指标对模型参数进行优化。

3、在验证集上根据模型的性能指标对模型的超参数进行搜索。

4、步骤 2 和步骤 3 交替迭代，最终确定模型的参数和超参数，在测试集中验证评价模型的优劣。

其中，搜索过程需要搜索算法，一般有：

网格搜索、随机搜过、启发式智能搜索、贝叶斯搜索。

3.4 激活函数

3.4.1 为什么需要非线性激活函数？

为什么需要激活函数？

1、激活函数对模型学习、理解非常复杂和非线性的函数具有重要作用。

2、激活函数可以引入非线性因素。如果不使用激活函数，则输出信号仅是一个简单的线性函数。线性函数一个一级多项式，线性方程的复杂度有限，从数据中学习复杂函数映射的能力很小。没有激活函数，神经网络将无法学习和模拟其他复杂类型的数据，例如图像、视频、音频、语音等。

3、激活函数可以把当前特征空间通过一定的线性映射转换到另一个空间，让数据能够更好的被分类。

为什么激活函数需要非线性函数？

1、假若网络中全部是线性部件，那么线性的组合还是线性，与单独一个线性分类器无异。这样就做不到用非线性来逼近任意函数。

2、使用非线性激活函数 $f(x)$，以便使网络更加强大，增加它的能力，使它可以学习复杂的事物，复杂的表单数据，以及表示输入输出之间非线性的复杂的任意函数映射。使用非线性激活函数，能够从输入输出之间生成非线性映射。