Hinton神经网络与机器学习 1. 绪论

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/11/12/UTNN-1-Introduction/

为什么需要机器学习?

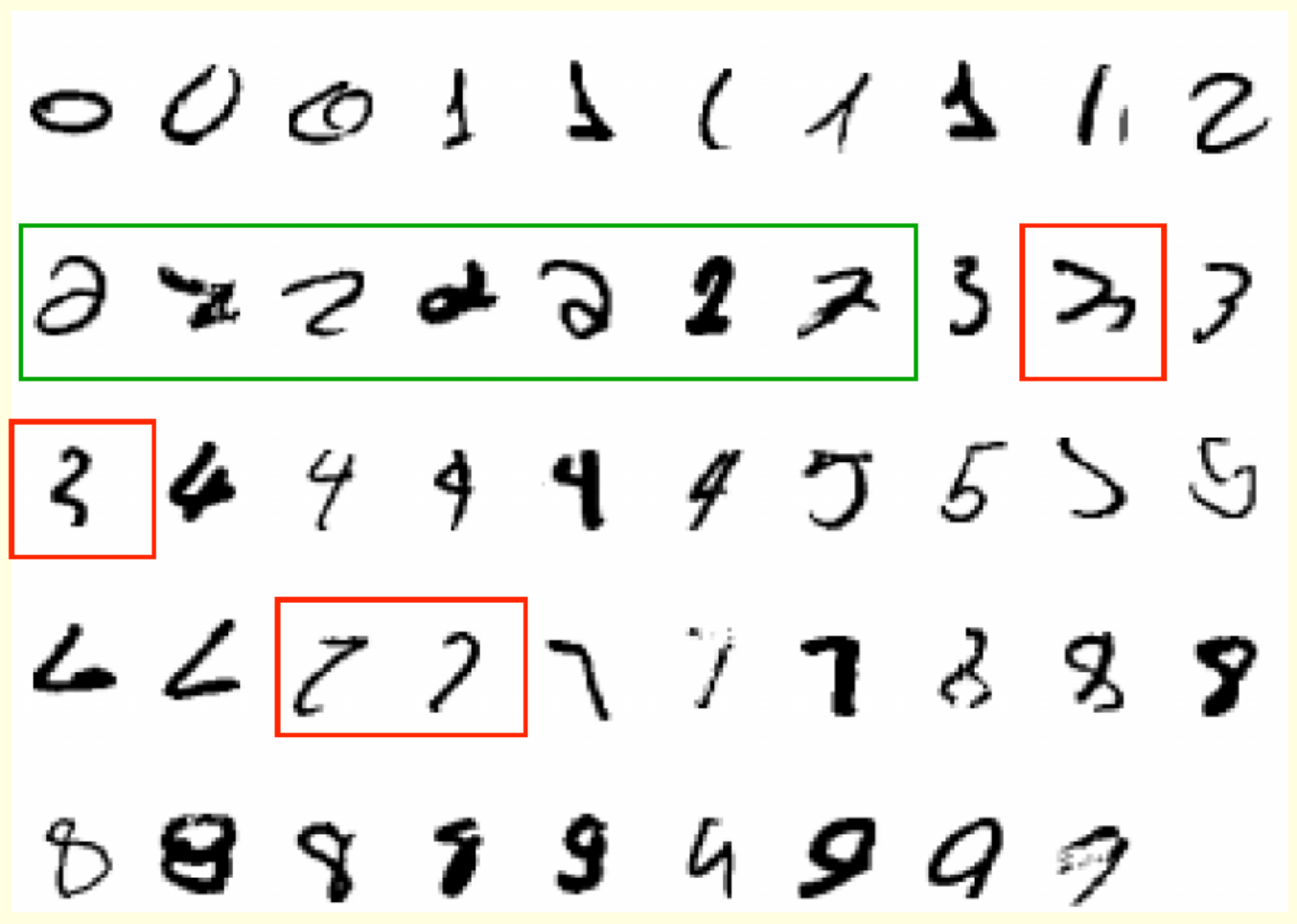
对于一些问题，例如在复杂环境里，如何在新的光源条件下识别三维物体，编写程序去解决这些问题本来就是非常困难的：我们也不知道我们的大脑到底是如何运作来识别它的，即便我们知道，写出来的程序可能也十分复杂。对于另外一些问题，例如判断某笔信用卡交易有多大概率是欺诈性的，可能无法给出简单有效的规则。要解决这样的问题，需要将很多种比较弱的规则组合起来。而且，所谓“道高一尺魔高一丈”，犯罪分子也会根据系统的判别手段，调整欺诈的方式，这就使得判别系统/程序也得随之变化

这些问题可以使用机器学习这一技术来解决：开发人员不再需要为每个特殊的任务编写程序，而是收集很多样本，这些样本针对给定的输入，会相应标记一个正确的输出。机器学习算法会接受这些样本，产生一个程序来完成指定的任务——这个产生的程序与传统的人工手写的程序不同，可能只包含一些数字。如果收集的数据是正确的，那么对于新来的，未在输入数据里出现的例子，程序也能正确处理；如果数据产生了变化（就像信用卡欺诈的例子里那样），开发人员也只需要使用新的数据进行训练，来对程序进行变化。在现阶段，使用机器做大量计算所花的钱比雇人为某个任务写代码的花费是要少一些的

典型的可以使用机器学习来解决的问题包括：

- 模式识别，例如识别物体、识别脸孔或识别所说的单词
- 异常识别，例如识别信用卡交易中的异常交易序列、在核电站识别传感器的异常读数。注意解决这样的问题有时甚至不需要明确地说明哪些值是正常的，哪些是异常的，而是完全由算法去发掘
- 预测，例如预测将来某一天股票的价格或者外汇汇率，以及用户对某部电影的评分

在这诸多问题中，本课将挑选一个作为一个典型样例贯穿始终。类比于遗传学中的工作，由于人们对果蝇的习性了解很多，而且果蝇繁殖速度很快，因此遗传学经常使用果蝇作为研究对象。本课所选的“果蝇”是MNIST数据库，这里面包含了很多人手写的数字。MNIST是公开的，可以被随意下载，而且使用MNIST作为训练数据可以很快训练一个中等大小的神经网络，此外，我们也了解其它很多机器学习算法在MNIST上的效果，因此MNIST是一个很好的例子



MNIST数据集的一个例子

上图给出了MNIST数据集的一个片段，所有的手写数字已经按照进行了归类排序。也就是说，图中的所有数字先是所有0，再是所有1，以此类推，一直到9。以图中用绿框框起来的手写的2为例，如果被告知“这就是手写数字”，那么人们大概是能看出来这些数字是2的，但是并不能用一两句话说清楚这些图像为什么就是2。图中给出的这些数字相互之间形状差异很大，即便是做一些简单的变化（例如旋转）也很难让任意两个数字重叠在一起，因此使用模板来做判定是基本不可能的——这也就是为什么这个任务是一个很好的机器学习任务的原因

在MNIST之上，还有一个更大的数据集ImageNet。该数据集提供了130万张高分辨率的图片，算法的目标是将每张图片分到1000个分类中正确的那一个。2010年时，最好的系统的top 5 choices错误率为25%。2012年时，使用深度学习训练处的模型做到了20%。到了2015年，最好的算法在这项指标上已经可以达到3.8%，而多个深度神经网络的组合模型甚至可以达到3.08%（http://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/）

此外，语音识别也是深度学习非常适用的一个场合。通常来讲，语音识别系统可以细分为三个阶段：

- 预处理阶段。在这个阶段，声波会被转化为一些声学系数组成的向量。通常情况下，每十毫秒的声波会被转化为一组向量
- 声学建模阶段。在这个阶段，会使用相邻的若干向量进行推测，猜测这些向量对应了什么音素
- 解码阶段。在这个阶段，将上一阶段做出的预测进行组合，拟合出的模型将输出说话人所说的内容文本

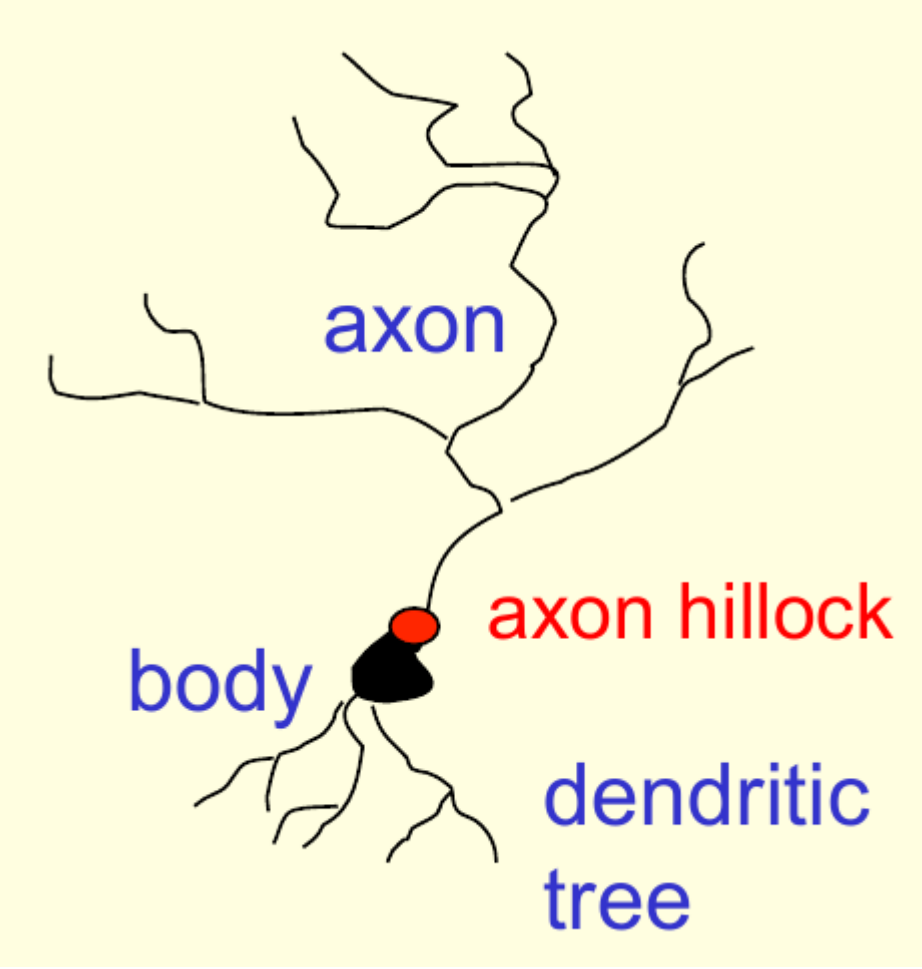
深度学习模型在这样的任务上取得的成绩是斐然的。对于TIMIT数据集，2012年之前最好的模型错误率是24.4%，而且需要对多个模型进行组合。2012年George Dahl和Abdel-rahman Mohamed的8层神经网络将错误率降低到了20.7%。这样的成绩也引起了微软研究院邓力的关注。对于其它语音识别任务，深度学习模型也能够取得比传统GMM模型更好的效果，而且训练时间更短

神经网络是什么?

前面提到的各种大放异彩的深度学习模型，被称之为“深度”的原因是其本质是叠加了很多层的神经网络。神经网络模型，其初始思想就是对生物的神经系统如何工作做一个模拟，所以这一节主要来对生物的神经系统做一个概述。了解生物神经元是如何计算的，有以下几点原因

- 可以了解大脑真正是如何工作的。大脑的工作原理非常复杂，而且当它被取出来时也就不工作了，因此需要使用计算机进行模拟
- 可以了解由神经元和它们之间自适应连接所启发的并行计算方式。这种计算方式与常见的串行计算很不相同。对于大脑很擅长的工作（例如视觉），使用这种计算方式可能能达到非常好的效果；但是对大脑不擅长的工作（例如计算23x71）就不适用了
- 大脑的工作原理可以启发研究人员发明一些新颖的机器学习算法来解决实际问题。尽管这些算法的原理可能并不同于大脑真正的工作原理，但是仍然会是有用的

神经系统的基本组成元素是皮层神经元（cortical neuron）。神经元由一个细胞体（cell body）和一根轴突（axon）组成，其中轴突末端会分叉。细胞体还会分出若干树突（dendritic），来从其它神经元获得输入。轴突与树突通常在突触（synapse）处交互，轴突中产生的冲动（spike）会导致电离子注入到突触后神经元，当足够的电离子流入以后，轴丘（axio hillock）会在突触产生流出的冲动，来对细胞膜（cell membrane）去极化（depolarize）。下图给出了神经元的一个示意图



神经元示意图

冲动随着轴突流动到达突触时，会导致突触小泡（vesicle）释放高浓度的化学传递物质。传递物质会沿着突触间隙（synaptic cleft）扩散至突触后神经元的细胞膜，吸附到膜中的受体（receptor），造成细胞膜形状的改变，产生若干小孔，使得特殊的离子可以进出。突触的功效性还可以随着突触小泡的数量和受体的数量变化。总体来看，突触是比较慢的，但是与RAM相比仍然有其优势：它们体积很小，功耗很低，而且可以根据局部的信号适应环境。但是它们具体是遵从何种规则来做出变化的呢?

每个神经元会从其它神经元那里获取到输入，它们之间通过冲动来进行交互。神经元的输入功效由突触的权重来控制，这些权重可以是正的，也可以是负的，而且这些权重是自适应的，因此整个网络可以适应不同的环境，完成不同的计算任务，例如辨识物体、理解语言、制定计划、控制身体等等。人身上大概有一千亿个神经元，每个神经元有一万个权重。这大量的权重可以在非常短时间内影响计算。

此外，神经元是模块化的，大脑皮层的不同部分有不同的功能，对大脑皮层不同区域的损伤会造成不同的后果，而正常人完成不同的任务时血液也会流向不同的区域。但是，皮层看上去又是相似的，如果在婴儿时期大脑受到了破坏，那么其它部分可能会承担被破坏部分的功能。例如如果在婴儿时期负责视觉的皮层遭到了破坏，大脑可能会把视觉任务重新分配给听觉皮层，在这样的刺激下，听觉皮层中会产生出视觉神经元，因此实际上有可能存在一个通用的学习方法。也就是说，皮层在形成的时候其用途是广泛的，但是随着经验的积累，不同区域的皮层会变得只处理某个特别的任务。如果使用类似的原理来计算，则可以兼顾速度、并发度和灵活性。传统的计算机通过存储顺序指令来获得这样的灵活性，但是这需要一个非常快的中央处理器来执行很长的计算序列

一些简单的神经元模型

本节介绍一些基本的神经元模型。尽管这些模型与真正的神经元工作原理相比简单得似乎不值一提，但是它们的能力已经足可以搭建起来一个神经网络模型，来解决一些有趣的机器学习问题

为了了解复杂的现象，一般总是要对目标做理想化才能建模。理想化的过程可以去掉繁复的枝叶：这些盘根错节的东西对理解主要原理是不必要的。理清主干以后，就可以使用数学手段，将目标系统与其它已经了解过的系统做类比，来理解主要原理。主要原理理解了以后，就可以方便地一点一点增加模型的复杂度，来让模型与现实更加贴近。当然，有时候去理解一些已经知道是错误的模型也是值得的，只要不要忘记这是一个错误的模型就行。例如，神经网络模型中神经元传递的通常是实数值而不是冲动，但是实践证明这种做法是有效的

神经网络中最简单的神经元是**线性神经元**。这种模型非常简单，但是计算能力有限。这种神经元的输出y由输入x_i的加权和再加上偏置b得到，即

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

其中w_i是第i个输入x_i的权重

在此基础上，McCullock-Pitts在1943年提出了**二元阈值神经元**，其第一步还是要算出输入的加权和，然后，要将加权和与某个阈值作比较，如果超出该阈值，则发射一个冲动信号。原作者认为，冲动信号就像是某个谓词的真值，因此这里每个神经元就是将输入的真值做组合，输出自己所代表谓词的真值。二元阈值神经元有两种表示形式。一种是纯加权和的形式，并设定阈值为θ。即

$$z = \sum_i x_i w_i, \;\; y = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

如果将阈值看作是负的偏置，即$\theta = -b$，就可以写出与线性神经元类似的偏置+权重的形式，即

$$z = b + \sum_i x_i w_i, \;\; y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Hinton将上述两种神经元进行了组合，得到了**ReLU神经元**（有时也称为线性阈值神经元）。这种方法仍然会计算输入的线性加权和，但是输出是所有输入的一个非线性函数。具体形式为

$$z = b + \sum_i x_i w_i, \;\; y = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

本课中最常用的神经元是**sigmoid神经元**，~~该神经元也是实践中最常用的神经元~~，它给出的是输出是实数的，且图像平滑有界。这种神经元通常使用logistic函数，即

$$z=b+\sum_i x_i w_i,\ \ y=\frac{1}{1+e^{-z}}$$

~~Logistic函数的好处是其导数的形式很漂亮，使学习过程变得容易~~。具体道理在第三讲会讲解

最后一种神经元是随机二元神经元。它使用的表达式与sigmoid神经元相同，不过把logistic函数的输出看作是短时间内产生冲动的概率。即，它的输出还是0或者1，但是是以概率$P=1/(1+e^{-z})$输出1。类似的技巧可以用在ReLU上，此时神经元产生冲动的概率服从泊松分布

笔者注

（以下注来自知乎用户杜客的回答和罗浩.ZJU的回答）

上面使用删除线删除的内容是Hinton在视频中提到，但是现在不适用的内容。近几年，在深度学习中，ReLU神经元的使用要比sigmoid的使用多很多。记Sigmoid函数为$\sigma(x)$，即

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

其导数（梯度）满足性质

$$\sigma'(x)=\sigma(x)(1-\sigma(x))$$

可知当$x=0$时，sigmoid函数的梯度达到最大值，仅为0.25。而当x很大或者很小时，该函数会接近1或者-1，此时称发生了**饱和**（saturation）现象（事实上，有$\sigma(5)\approx 0.9933$）。在这种情况下，函数的梯度值接近0。当使用深度网络或者RNN时，由反向传播的理论，会出现多个梯度的连乘。而多个接近于0的数连成会快速逼近0，造成**梯度消失**，使得几乎就有没有信号通过神经元传到权重再到数据了。而且，如果初始化的矩阵随机得不好，权重过大，那么饱和会快速出现，模型基本学不到什么东西

对于ReLU而言，当$x>0$时，其激活函数x的导数为1，连乘不会逼近0，避免了梯度消失。此外，对于输入值为负的神经元，其梯度为0，不会参与训练，保证了神经网络的稀疏性，可以避免过拟合。最后，ReLU梯度的计算更加简单，只需要写一个if-else语句，而sigmoid函数的导数涉及到了浮点数计算，实际上还是麻烦的

在近年的实际应用中，另一种**tanh神经元**也受到了一定程度的重视。形式为

$$z=b+\sum_i x_i w_i,\ \ y=\tanh(z)$$

事实上，$\tanh$函数可以由sigmoid函数经过一个简单变换得到。其与原始sigmoid函数相比最大的好处是其是*零中心*的（函数图像在0点中心对称），不过并没有解决在输入太大或太小时造成的梯度消失问题

关于学习的一个简单样例

本节将给出一个关于机器学习的例子，尽管使用的网络结构比较简单，但是它已经可以做一些手写数字识别的工作。这个网络只有两层：

- 输入层神经元表示原始图像的像素密度
- 输出层指出图像表示的是哪个数字

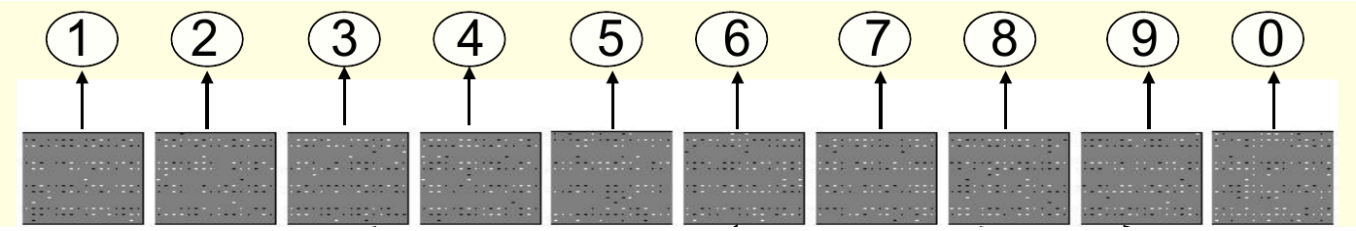
图像的每一个像素点，如果被染色，就相当于有了投票的权利。每个被染色的像素点可以给几种不同的数字投票，得票最多的数字获胜——因此，这个过程就好像是输出的若干个类别互相比赛

前面提到，神经网络中的参数其实就是各个输入所对应的权重，因此为了展示训练过程，一个重要的问题是如何把权重表示出来。直观的做法是写明每个像素点为每个类别贡献了多少权重，但是这样需要写成千上万行数字，因此退而求其次，可以使用颜色来代表权重的大小和正负。对于某个给定的类别（这里是数字），如果一个像素点的颜色越浅，则代表该点贡献了一个越大的正权重；如果颜色越深，则代表该点贡献了一个越小的负权重

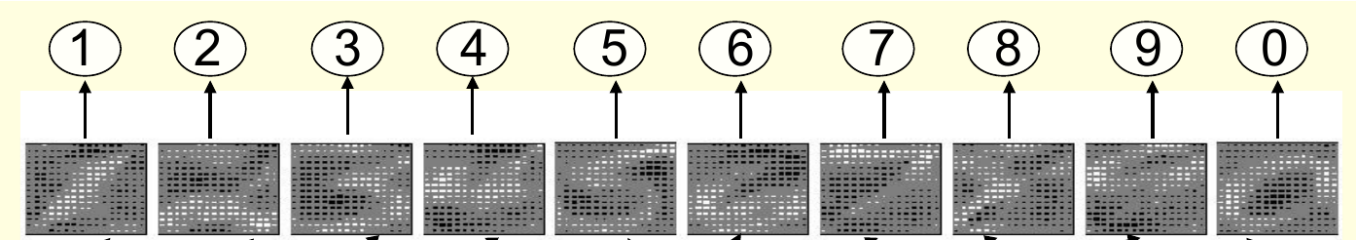
学习的方法设定为，首先，将所有像素点对所有类别贡献的权重随机初始化一个小值，然后对每个给定的手写数字图像中被染色的像素点：

- 增大其对正确分类的权重
- 减小其对网络所猜测的类别的权重

这种做法可以达到一个微妙的平衡：如果神经网络给出了正确的预测，那么该像素点对正确类别的权重贡献先增后减，相互抵消；但是如果给出了错误的预测，那么该像素点对正确类别的权重增大，对错误类别的权重减小。下图给出了刚刚初始化时和使用若干图片训练后的权重示意

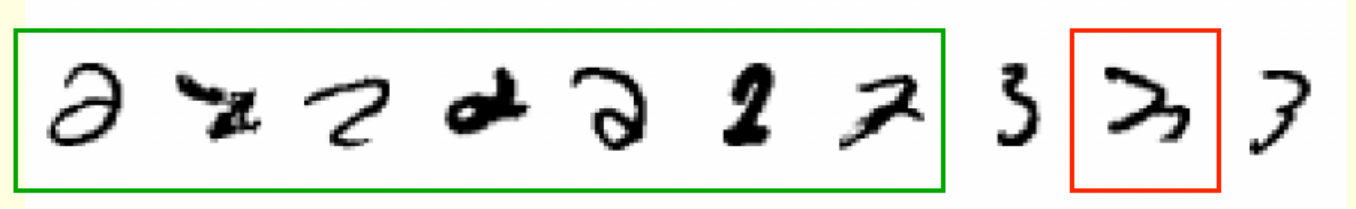


网络对每个数字的初始权重



网络训练后的最终权重

尽管网络已经展现出了学习能力，而且效果还不错，但是它仍然有不足。本质上，它学到的其实是每个数字的一个手写模板。当来到一个新的输入时，对这个输入，用学到的10个模板挨个匹配，哪个匹配得最吻合，就认为写的是哪个数字。但是这个模板并不能完美匹配所有手写的数字图像。例如，对于下面的例子，无法找到一个模板，使得其可以辨识出所有2，同时又不会误把3当成2。因此，还是需要更复杂的网络模型


简单网络难以区分的例子。绿框里的2和红框里的3使用上述算法无法区分

学习的三种类型

机器学习任务总体来讲可以分为三种类型

有监督学习

有监督学习的形式是，给定一个输入向量$\mathbf{x}$，学习预测一个输出t。根据输出的形式，有监督学习又可以细分为两个子类别

- 回归：目标输出是一个实数，或者实数向量。例如预测之后6个月的股票价格、明天中午的气温等
- 分类：目标输出是一个类别标签，例如手写数字识别

进行有监督学习的典型过程通常是，首先选择一类模型$y = f(\mathbf{x}; \mathbf{W})$，它接受输入向量$\mathbf{x}$，使用相关参数$\mathbf{W}$，得到最后的输出$y$。模型这里也可以看作是一种映射，是输入到输出经由$\mathbf{W}$作用的映射。因此，学习的核心就是调整这些参数$\mathbf{W}$，使得模型输出$y$与期望输出$t$之间的差距尽可能小。对于回归问题，这个差距的衡量方法通常是$\frac{1}{2}(y-t)^2$ 。而对分类问题，存在更合理也更有效的衡量方式，之后会讲到

强化学习

强化学习的输出是一个动作，或者一串动作序列。这里也存在监督性的信号，不过只是不定时出现的一些奖励（或惩罚）。因此，强化学习的目标是选择最合适的动作来最大化将来可能获得的奖励的期望和——通常情况下，对每个奖励会设置一个跟时间相关的衰减因子，这样越后面的奖励加权后的重要性越小，使得模型不用考虑太远的将来

强化学习是一件比较难的事情，因为奖励通常都不会马上得到，所以模型并不知道是否走在了正确的道路上，而且奖励只是一个标量，本身提供的信息也有限

强化学习难以学到成百万的参数，通常是几十到几千个。但是其它两种学习形式不会有这样的限制

无监督学习

无监督学习的目的是要发现输入数据一个良好的内部表现形式。在将近40年的时间里，无监督学习在机器学习这个圈子里基本是处于被忽视的状态，一些广泛使用的关于“机器学习”的定义甚至实际上把这种问题排除掉了，而很多研究者认为只有聚类才是无监督学习唯一的表现形式

造成这种现象的原因之一是，很难一句两句话说清楚无监督学习的目标究竟是什么。大致来看，可以列举出如下几类目标

- 为输入数据创造一种内部表示形式，用于之后的有监督学习或强化学习
- 为输入数据提供一种紧凑、低维的表示方法，例如PCA
- 以学到的特征这一形式，为输入提供一种经济的高维表示方法
- 找到输入中一些合理的类簇

本次课程的前一半主要讲授有监督学习问题，后一半主要讲授无监督学习问题，而强化学习不在本次课大纲之内（笔者注：不过近几年深度强化学习也受到了越来越多的重视）

Hinton神经网络与机器学习 2. 感知机

- **本文作者：** Tingxun Shi

- **本文链接：** http://txshi-mt.com/2017/11/18/UTNN-2-the-Perceptron-Learning-Procedure/

神经网络体系结构的主要类型

本课程主要介绍的内容是各种各样的神经网络，而这些神经网络根据神经元连接的方式大致可以划分成三种主要的体系结构

第一种结构是**前馈神经网络**（feed-forward neural networks）。~~这种结构是迄今为止实践中应用最多的结构~~，其第一层是输入，最后一层是输出，中间所有层都称为隐藏层。如果隐藏层层数大于1，那么得到的网络就称为“深度”神经网络。使用时，数据从输入层流入，沿着一个方向通过隐藏层，直到输出层。由于网络中各个神经元都使用了不同的非线性函数做激活函数（见上一讲中的“神经元模型”一节），来对输入进行处理，因此前一层原本相似的数据经过激活函数变换以后，在下一层可能就会变得不同；反之，原来不同的地方可能就会相似。典型的例子是语音识别中，需要不同说话人说的相同内容变得相似，同一说话人说的不同内容变得相异，因此非线性激活函数就起到了这样的作用

第二种结构是**循环神经网络**（recurrent neural networks, RNN）。这种结构比传统前馈神经网络更强大：对于某一个隐藏层神经元，其不一定非要连向下一层神经元，而是也可以连向同层中的其它神经元，即这种网络的连接图中可以有一个有向的环路。这意味着从某个神经元开始，沿着连接方向游走，最后可能会回到出发点。RNN可以表现出很复杂的动态性，因此比较难训练，但是目前研究人员已经找到了很多突破方法

RNN更像生物体中神经网络的结构，同时，它更适用于对序列建模。此时，隐藏层单元之间也会有连接，从表现形式上看，隐藏层之间的连接就像是一个随时间展开的，非常深的网络。此时每个时刻隐藏单元的状态都会决定下一个时刻隐藏单元的状态，只不过隐藏单元之间使用的权重矩阵是相同的，而深度前馈网络中各个隐藏层之间的权重矩阵是不同的。RNN具有在隐藏单元中长时间记忆信息的能力，在自然语言处理领域中用得非常多。例如Ilya Sutskever在ICML 2011上的论文*Generating Text with Recurrent Neural Networks*就使用了RNN来预测给定字符序列后的下一个字符。该工作使用英文维基中的文章作为训练材料，得到的模型可以产生出基本符合英语语法的段落

第三种结构是**对称连接网络**（symmetrically connected networks）。在RNN中，隐藏层神经元之间可以有一串沿时间线向前的连接，也可以有一串向后的连接，但是这两串连接使用的权重矩阵是不同的。对称连接网络也是有两串连接，但是这些连接是对称的，也就是它们的权重矩阵相同。没有隐藏层单元的对称连接网络称为Hopfield网，而带有隐藏层单元的称为玻尔兹曼机（Boltzmann machine）。对称连接网络比RNN更好分析，因为它们在能力上受到了一些限制，而且更符合能量函数。对称连接网络不能对环路建模

初代神经网络：感知机

（本章从本节开始的内容也可参考林轩田老师课程的笔记。需要注意的是，林课中标签集合$\mathcal{Y}=\{-1,1\}$，而本课使用的标签集合$\mathcal{Y}=\{0,1\}$）

本章的主角是感知机这种模型，它在20世纪60年代初期曾经红极一时，但是随着Minsky和Papert证明了它们能力上的限制以后就迅速衰落了下去

对于统计的模式识别问题，有一套比较标准的解决方案：首先，把原始输入向量转化成特征向量。这个阶段不涉及任何“学习”的内容，由开发人员通过直觉来手写代码完成。得到的特征可能有用，也可能没用，如果没用的话，再设法去产生新的特征来验证，这样最后肯定能够得到一组能用的特征向量。真正的学习阶段出现在确定各个特征的权重时。由于最后得到的是一个标量，因此每个特征的权重代表了该特征在多大程度上支持或者反对“输入数据属于要识别的模式”这个命题。确定权重以后，将所有特征加权求和，如果最后得到的值超过某个权重，就认为输入是目标类别的正例，反之则认为是负例。感知机是统计模式识别系统的一个例子，有多种变种，但是被Rosenblatt称为α-感知机的模型大体遵循了这个过程：使用人工从输入获得的激活向量，学习出一个权重，将权重与激活向量求内积得到一个值，根据这个值与阈值的比较结果判定样本的类别

具体说来，感知机使用的是一个二元阈值神经元模型，这个模型在前一章曾经提到过。其表达形式如下：

$$z=b+\sum_i x_i w_i,\ y=\begin{cases}1 & \text{if } z\geq 0\\ 0 & \text{otherwise}\end{cases}$$

由于偏置项可以看作是阈值的负数，为了不去为偏置项设置一个单独的学习策略，可以将输入做一个修改：在输入向量中增加一个维度，这个维度上的值永远是1，那么学到的权重向量在该维度上就是偏置值

对输入进行处理（加入常数项）后，就可以使用如下方法训练感知机：

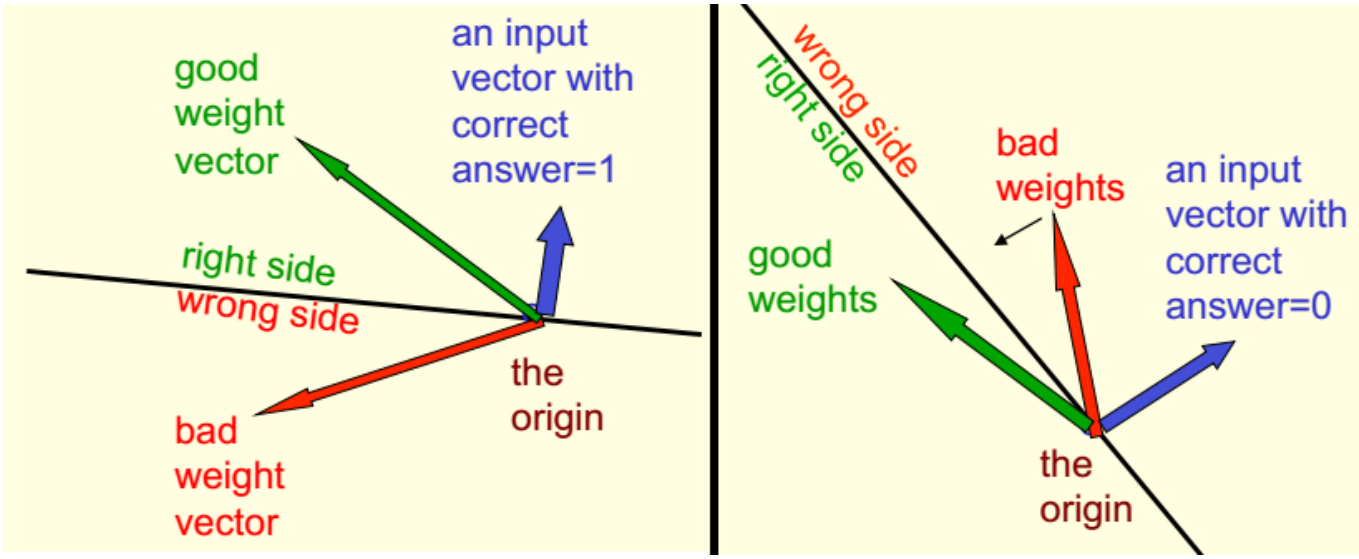
> **感知机训练算法**
>
> 随机初始化权重向量$\mathbf{w}^{(0)}$，使用任意方法遍历全部样本，对每一个样本$\mathbf{x}_i$观察输出，如果输出是正确的，略过，否则
>
> - 如果真实标签是1而输出为0，则$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{x}_i$
> - 如果真实标签是0而输出为1，则$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \mathbf{x}_i$
>
> 重复上述过程，直到遍历过训练集所有样本而权重没有发生改变时，算法终止

感知机算法的一个很好的性质是，如果存在一个权重向量可以将所有样本正确分类，那么算法一定会终止且找出这个向量。然而，它的局限性也在于，如果对给定的训练样本不存在这样一个向量，算法就不能正常工作。事实上，这个向量存在与否，决定于使用了什么特征，而对很多问题很难找到良好的特征

感知机的几何意义

本节将在高维空间里讨论感知机的几何意义，同时要涉及到一些关于超平面的概念。尽管听上去比较复杂抽象，但是若把一个14维空间看成一个3维空间，超平面看成是一个2维平面，问题就好理解一些

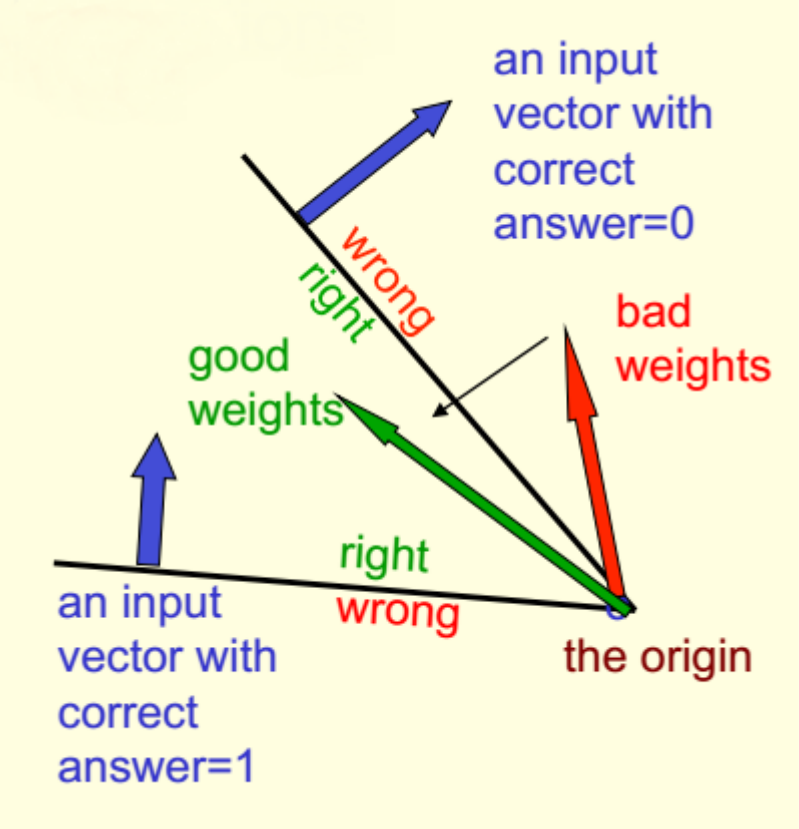
首先来看权重值空间。这个空间里的每一个维度都对应权重向量的一个分量，而空间中的每个点都代表了某个可能的权重。假设沿用上一节的做法，不单独考虑阈值，那么每条输入数据就可以表示为一个穿过原点的超平面，权重在该超平面的某一侧。下图给出了一个示例



感知机权重向量与训练样本超平面的关系。左边为输入向量为正例的情况，右边为输入向量为负例的情况

在图中，每个训练数据以蓝色的向量表示，其定义了一个超平面（以黑线表示），注意超平面与该训练数据所代表的向量总是正交的（即该训练数据是其定义的超平面的法向量）。对于上图左，输入样本为正例。如果权重向量与输入向量指向同样的方向（超平面的同一侧，上图左的绿色向量），此时两个向量的夹角是个锐角，内积的结果是正值。由于不考虑阈值，因此内积为正代表权重向量给出了正确的判断，即这个权重向量是好的。相反，对于上图左中的红色向量，其与输入向量的夹角是钝角，内积结果为负，意味着其将输入样本判定为负，所以是不好的向量。对于上图右，输入样本是负例，因此需要好的向量与输入向量夹角为钝角，指向相反的方向（即权重向量和输入向量指向超平面的两侧）

综上所述，对于任意给定的输入向量，只要某个权重向量指向了正确的方向，那么就能得到正确的答案。进一步地，如果在同一张图里同时画出正例和负例样本，会怎么样呢？

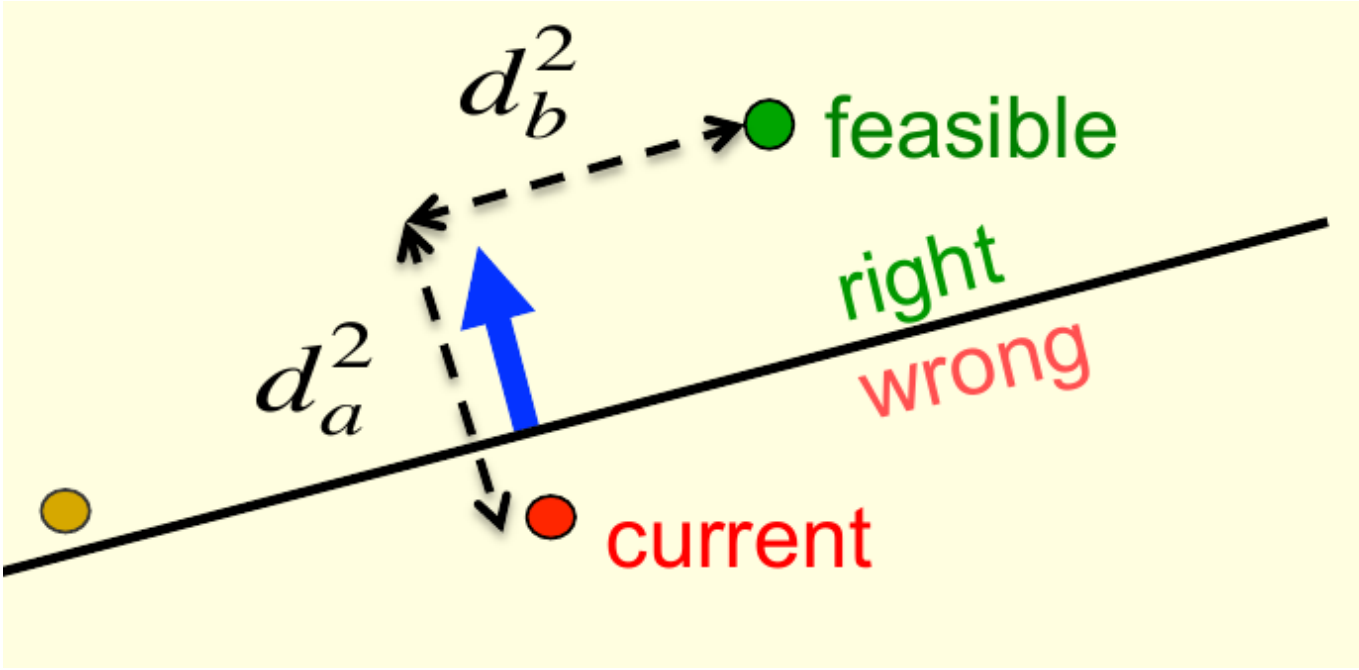


感知机向量应该在某个锥体中

上图中，正例样本和负例样本确定的两个超平面又定义了一个锥体（cone），所有落在这个锥体中的权重向量对这两个样本都能给出正确的结果。这个例子可以容易地扩展到多个样本的情况——当然，有可能对于某个样本集，不存在可以将所有样本都正确分类的向量。但是如果这样的权重向量存在，它必然会存在于某个锥体中。此外，另一个需要注意的事实是，对于任意两个满足条件的权重向量（即这两个向量都在合适的锥体中），它们的均值也会满足最优条件。这意味着这个问题是凸的，而凸学习问题（凸优化问题）往往容易解决

为什么感知机算法能够工作

本节将（从几何的角度）证明为什么感知机算法能最终落入最优解的凸锥中（更理论的证明可以参看林轩田老师的课程笔记）。假设存在一个权重向量能够对训练集的所有样本正确分类，并称该向量为可行向量，以及当前向量错分了某个样本，记为错误向量，如下图所示



讨论的初始状态

上图中，绿色的点代表理想的可行向量，红色的点代表当前的权重向量。由感知机算法的权重更新规则，当算法错分某个样例时，会让权重向量更靠近可行向量一些，因此可以得出当前权重向量与可行向量点的距离平方，其可以进一步分解为某个沿着超平面方向向量的长度平方与沿着超平面法向量方向的向量的长度平方的和，按照图中的记法，为$d_a^2 + d_b^2$。更新权重以后，权重沿着蓝色箭头的方向移动了一点，因此d_a变小，d_b不变，权重可以向着可行向量逼近

但是有些情况是个例外：如果可行向量对应于图中金色的点，输入向量太大，那么当前权重移动就会过多，离可行向量更远。因此我们想定义更可行的权重向量，它不仅能分对所有训练样本，还能保证每个训练样本离它有一定距离（保持一定的边界），这个距离要至少等于定义了超平面的那个输入向量的长度，这样，就能保证感知机算法犯错时权重向量总是靠近可行向量了。因此在有限次犯错后，权重向量肯定会落在可行区间，**如果这个区间存在**

（感觉更严谨的证明还是得看林轩田老师课的笔记）

感知机算法的局限性

感知机的局限性主要来自于应用者所使用的特征。如果用的是对的特征，那几乎什么都能做；但是如果用的是错的特征，就会受到很大的限制

假设现在的数据集输入全都是0-1数组，那么最简单的做法是为数组的每种情况都创建一个特征，这样，对n位的数组，创建2^n个特征可以涵盖所有情形，理论上不会受到任何限制。但是实际上，这种方法很难泛化，因为看到的训练集很可能只是可能出现的所有情况的一个子集，如果测试时遇到了新不在这个子集里的特征组合，那么查表法就无法工作（回忆林课的例子，假设现在有3个特征位，一共有8种组合。即便我们知道了其中5种组合分别对应于正例还是负例，也无法确定剩下3种组合应该如何判定）

如果要使用感知机算法，对这类数据学习权重来判断正负例，事情也会不太简单。事实上，对某些非常简单的例子，感知机算法都无能为力。考虑如下问题：假设输入数据维度为2，每个维度的取值范围是0或者1，假设输入数据为

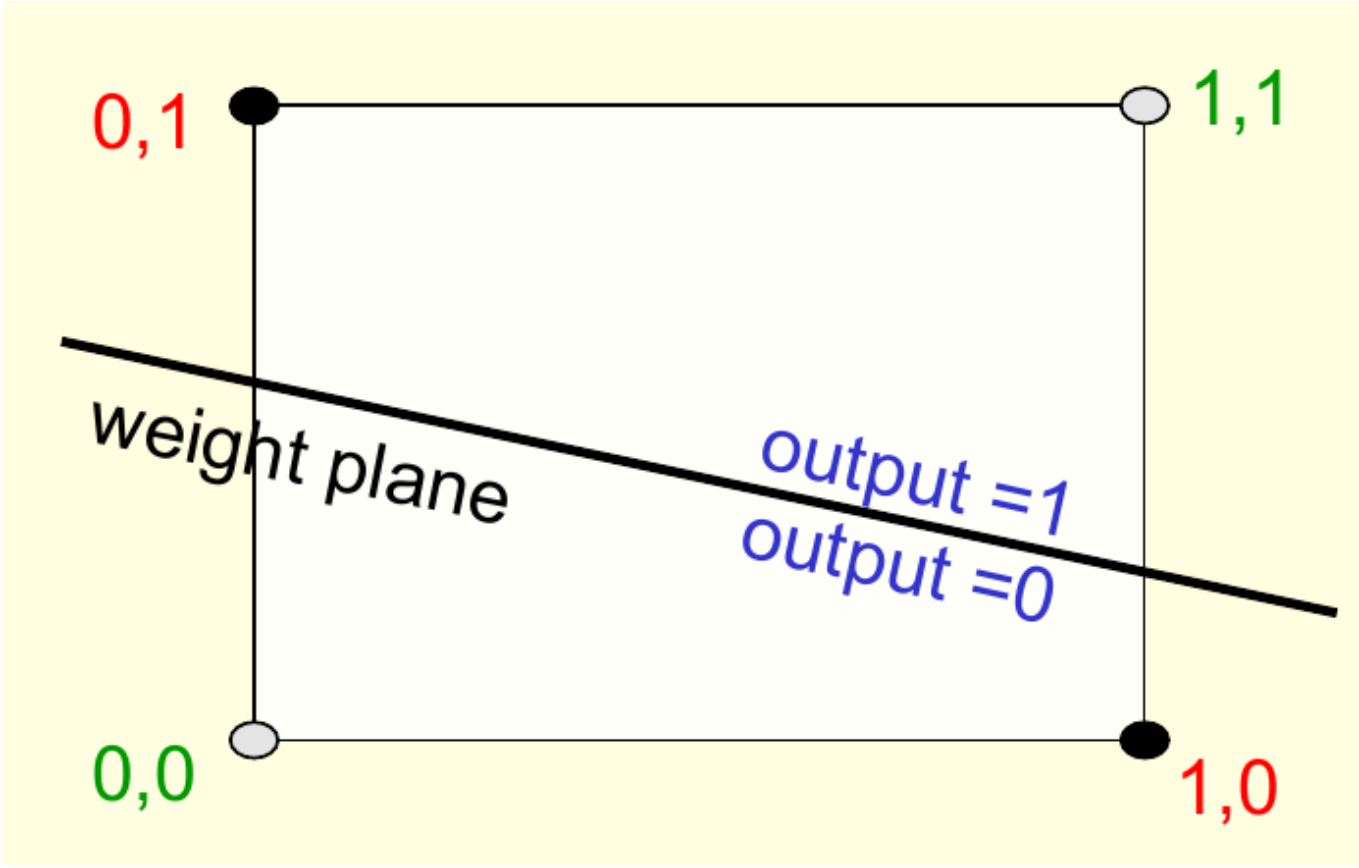
$$(1,1) \to 1; \quad (0,0) \to 1$$
$$(1,0) \to 0; \quad (0,1) \to 0$$

即，如果两元素相同，则为正例；两元素相反，则为负例。对于这样的例子，感知机不能学习到权重。从代数的角度看，求解这个例子相当于求解如下四个不等式

$$\begin{aligned} w_1 + w_2 &\geq \theta, & 0 &\geq \theta \\ w_1 &< \theta, & w_2 &< \theta \end{aligned}$$

这四个不等式显然是不能同时成立的：由第二个不等式，θ是非正数，又由第三个和第四个不等式，w_1, w_2是小于θ的负数，它们的和肯定是个更小的负数，$w_1 + w_2 < \min(w_1, w_2) < \theta$，与$w_1 + w_2 \geq \theta$是矛盾的

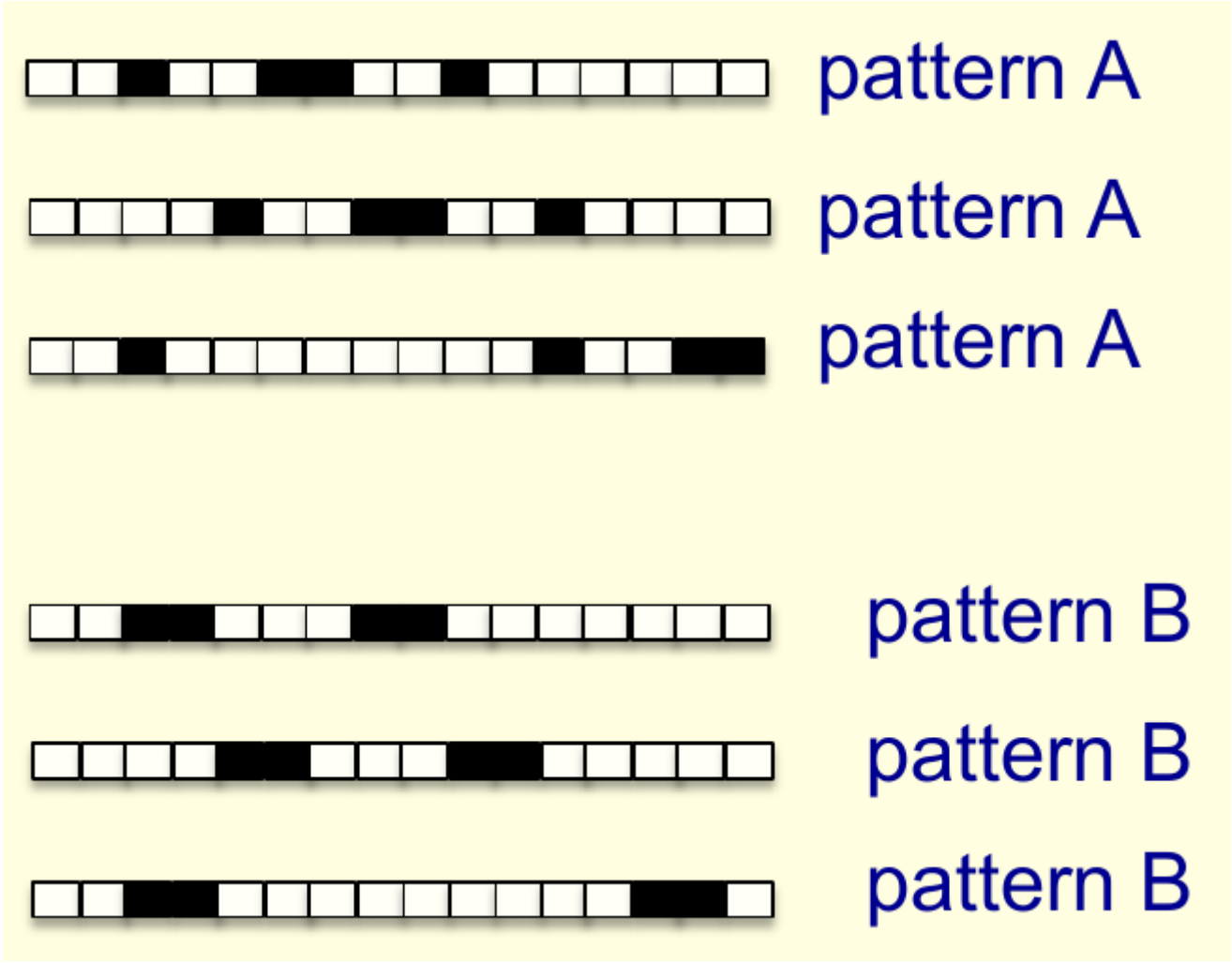
从几何的角度看，这个问题更为直观。对于上述问题，建立一个坐标系，横纵轴对应输入向量的两个维度，则这个空间中的每个点都可以对应于一个数据点。权重向量则在这个空间中定义了一个分离超平面，该平面垂直于权重向量指向的方向。此外，这里还考虑进了截距（偏置）的存在，因此分离超平面不应非得通过原点。对于前面给出的例子，可以做出下图



线性不可分的情况，感知机无法求解出权重

可以看到，对于这样的四个数据点，无法做出一条直线使得所有灰色的点在线的一侧，同时保证所有黑色的点在线的另一侧。对于这种无法使用超平面分离的例子，我们称其为**线性不可分的**。

对于其它例子，感知机也有不能判别的情况。考虑下面这个模式识别的例子：假设有一个长度为L像素的线段，其中一些像素点被染成了黑色。染色的方法遵循某些模式，如图所示



感知机不能完成的一种模式识别任务案例

对于上图，模式A为1黑2白2黑2白1黑，模式B为2黑3白2黑。注意这里把这个线段看作是首尾连接的，因此模式A和B的第三种情况也都遵守各自的规则。而且，例子中两个模式虽然不同，但是其中黑色像素点的数量相同，都为4。假设只使用像素点是否为黑作为特征，能否用感知机区分这两种模式？答案是不能。因为对模式A，有四个点被染色（激活），感知机得到的得分是$4\sum w_i$；而对模式B，也有四个点被激活，感知机得到的得分也是$4\sum w_i$。因此在这种情况下，感知机无法区分A和B。Minsky等人的著作也就是从这里入手对感知机进行批判的，因为模式识别就是要保证算法能够在模式经过变换以后仍然能够判断出模式是否相同，而这个实验证明感知机不能直接学习到“变换”。为了处理这些变换操作，感知机需要使用多个特征来意识到子模式的变换所带来的信息，而这种特征需要人手动编码来提供，并不是通过学习过程学到。在20世纪70年代，当时人们的结论是，感知机以及衍生出的神经网络都是不怎么有用的。而现在回头看，更正确的结论是，只有到神经网络能够学到如何检测特征，那才是真正的强大。不过人们花了将近20年来思考如何用神经网络检测特征

从这一讲可以得知，没有隐藏层单元的神经网络（即感知机）能力非常有限。而且，即便加入隐藏层单元，如果在这些单元只进行线性变化，意义也不大，因为最后还是一个线性模型，这种模型只有在我们手动喂进了正确的特征时才有用。真正需要的模型是多层含有自适应非线性隐藏单元的模型，但是这样问题就变成了我们应该如何训练这样的网络。事实上，学到隐藏单元的权重就是在学习特征，而这也就是真正的难点：没有人能直接告诉我们隐藏层应该干什么，它们什么时候应该活跃，什么时候不该活跃。没人能告诉我们特征选取器应该是什么样时，如何学习隐藏单元的权重，进而把隐藏单元转化成真正需要的特征选取器呢？

Frank Rosenblatt在20年代60年代初期编写了一本伟大的著作《神经动力学原理》，在这本书中，他描述了很多种不同的感知机模型。1969年，Minsky和Papert出版了《感知机》这本书，分析了感知机的能力和局限。然而，很多人却将本书中提出的局限扩大到了所有神经网络模型，甚至是整个人工智能领域。但实际上，感知机在今天仍然广泛应用于使用非常大特征向量的应用中

Hinton神经网络与机器学习 3. 反向传播

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/11/21/UTNN-3-the-Backpropagation-Learning-Procedure/

学习线性神经元的权重

本节首先来看使用线性神经元的神经网络如何学习权重。这个结构看起来像普通感知机，但是其实学习过程是不一样的。感知机的学习过程是调整权重，让权重向量向更好的权重向量靠近，而线性神经元虽然也是在调整权重，但是目的是让输出向目标输出靠近

感知机算法中保证的一个基本情况在神经网络里不会出现：在感知机中，两个有效权重的平均仍然是个有效权重；但是对神经网络，并不一定会是这样。因此，神经网络的学习方法与感知机的学习方法不同——对于多层神经网络，它甚至就不该被叫做是“多层感知机”。训练神经网络时，是通过观察实际输出值是否靠近目标输出来判断网络的学习能力是否在增强。这种方法对验证非凸优化问题时也适用，不过不适用于感知机算法。对于感知机算法，即便权重向量在靠近正确的权重，输出可能也会远离正确的输出

多层神经网络里最简单的情况是使用线性函数做神经元。假设权重向量为$\mathbf{w}$，输入向量为$\mathbf{x}$，那么神经元的预测值就是

$$y = \sum_i w_i x_i = \mathbf{w}^{\mathsf{T}} \mathbf{x}$$

由于此时观察的是预测值与实际值的距离，因此这里使用两者之间差的平方（平方误差）来衡量。这个问题是有解析解的，但是神经网络并不准备用这种方法求解权重：一方面，从科学角度，生物神经元的工作方式应该不是求解一个符号方程；另一方面，从工程角度，希望找到一个更普适的做法，来让它也适用于多层非线性神经元的情况。解析解法只能适用于使用平方误差、线性激活的情况，而迭代解法尽管效率低一些，但更容易适用于更复杂的系统

具体的迭代解法为，首先定义误差函数：

$$E = \frac{1}{2} \sum_{n \in \text{training}} (t^n - y^n)^2$$

将误差函数对权重求导

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_n \frac{\partial y^n}{\partial w_i} \frac{dE^n}{dy^n} = -\sum_n x_i^n (t^n - y^n)$$

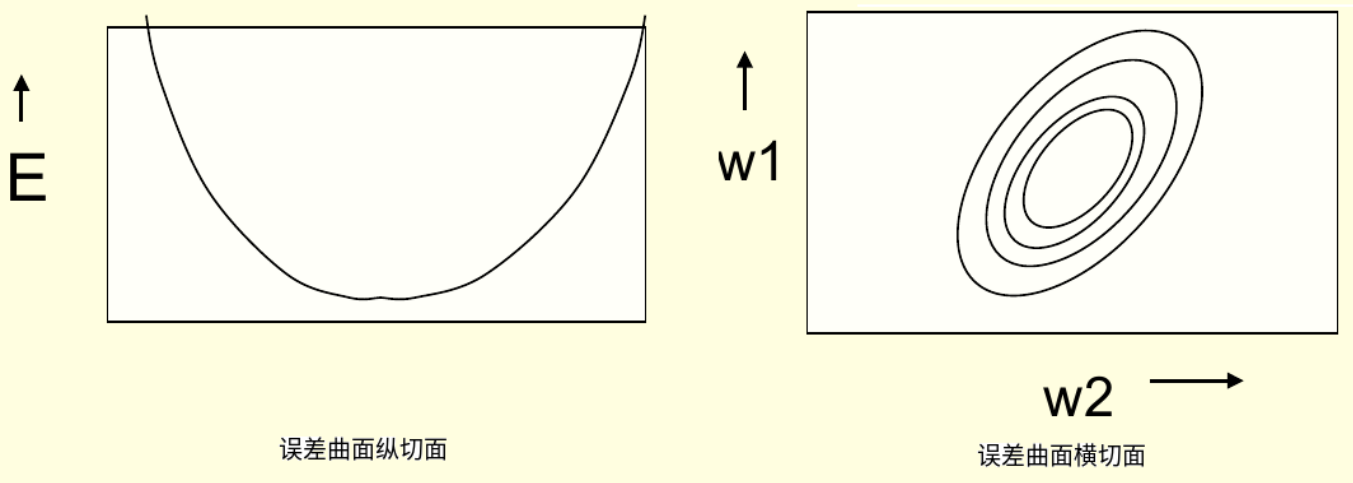
因此“delta rule”，也就是权重的更新规则为

$$\Delta w_i = -\epsilon \frac{\partial E}{\partial w_i} = \sum_n \epsilon x_i^n (t^n - y^n)$$

现在的问题是，这种迭代式算法（称为**梯度下降**）最后能得到正确答案吗？也许最后得到的不是最佳答案，但是如果把学习率ϵ调小，则可以接近最佳答案。另一个问题是这种算法的效率如何。需要注意的是，如果输入向量的两个维度高度相关，则学习效率会比较差。如果使用在线的权重更新规则，也就是每遇到一个样本就更新一次权重，那么这跟感知机的权重更新规则很像。不过，此时变化量大小同时由残差大小$t^n - y^n$和学习率决定，因此选择一个正确的学习率是件比较麻烦的事儿：如果学习率太大，系统会不稳定；如果学习率太小，收敛又会比较慢

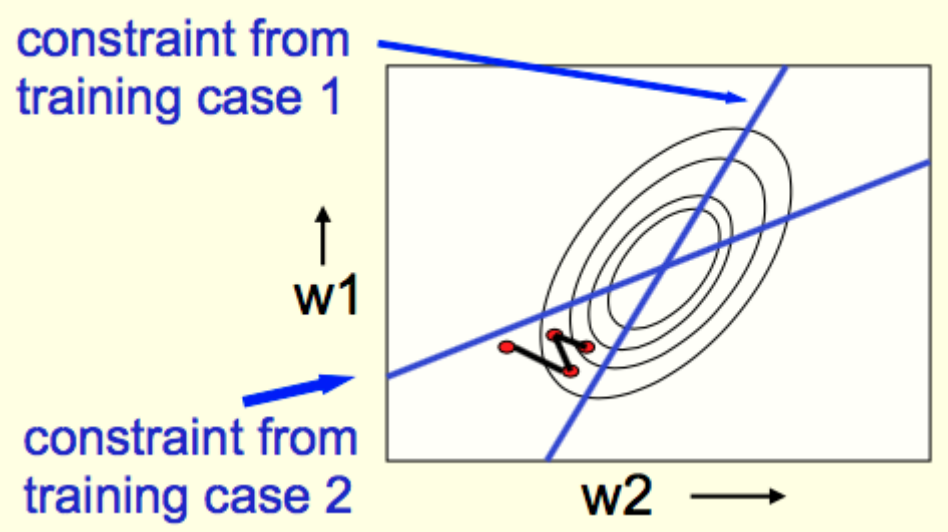
误差曲面

可以通过误差曲面（error surface）来理解线性神经元学习过程中发生了什么。考虑最简单的情况，假设权重只有两个维度，使用线性神经元和平方误差函数，那么以这两个权重为水平坐标，误差值为纵坐标，可以画出一个误差曲面。这个曲面的纵切面都是一个抛物线，横切面总是一个椭圆。更复杂一些。当权重不太大的时候，误差曲面会是光滑的，但是可能仍然存在许多局部最小值点



误差曲面示意图

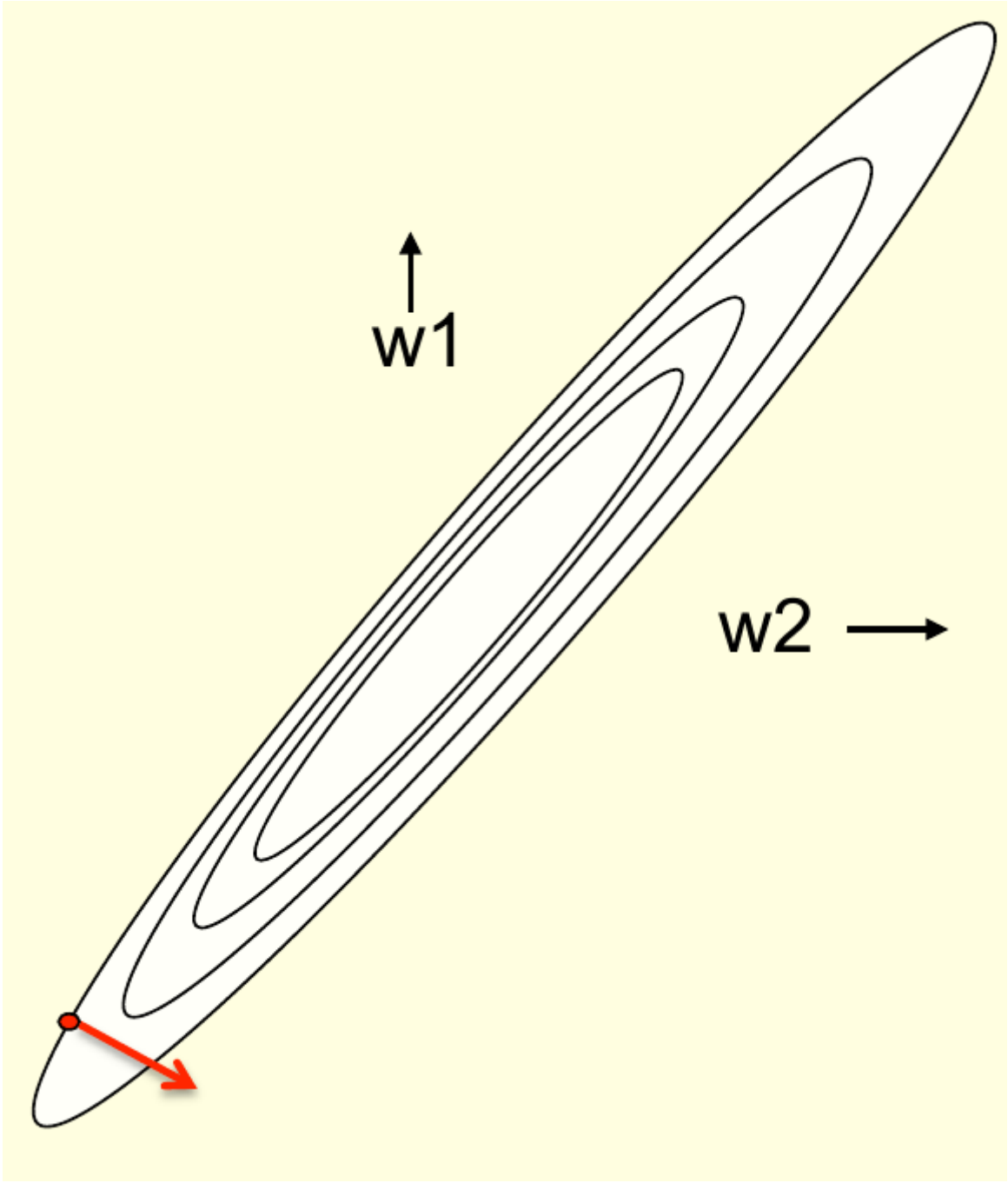
通过误差曲面，可以知道梯度下降每一步其实就是计算误差函数对权重的导数，如果将权重沿着导数方向变化，那就是在误差曲面上沿着最陡的方向下降了一步。如果从上往下看这个误差曲面，就可以画出一些椭圆形的等高线。如果使用批处理的梯度下降法，就可以使得权重沿着正确的角度向中间逼近，但是每次更新权重都需要把所有$x_i^n(t^n - y^n)$加起来。也可以使用在线学习的方法做，即每看过一个测试数据就更新一次权重，这时更新权重时所走的角度就会受到训练数据的限制



在线梯度下降受到训练数据的限制

如上图所示，第一次更新权重是根据第一条数据，因此移动方向垂直于第一条数据对应的蓝线，第二次更新权重时根据第二条数据，移动方向垂直于第二条数据对应的蓝线，以此类推，循环往复。权重也会曲折地向着两条蓝线的交点前进

考虑一种极端情况，就是输入数据限制线几乎平行，等高线是特别长的椭圆的时候。如下图所示



梯度下降的最差情况

此时，梯度下降的每个方向基本都垂直于向着极值点的方向，每次迭代时迈出的那一步都只向极值点靠近很小的一点，因此训练时间就会特别长

学习Logistic神经元的权重

为了将线性神经元的学习规则扩展到由非线性神经元组成的多层网络的学习规则，需要做两点

首先，需要将学习规则扩展到单个非线性神经元，这里选用的非线性激活函数是Logistic函数。这个函数的输出是一个实数值，图像平滑，而且在实数集上有界。神经元的激活函数如下：

$$z = b + \sum_i x_i w_i,\ y = \frac{1}{1 + e^{-z}}$$

这里y就是一个Logistic函数，可以简记为$y = \sigma(z)$。可以容易地推导出对$z \in \mathbb{R}, y \in (-1, 1)$。Logistic函数的另一个性质是求导比较容易，即有

$$y = \sigma(z) \Rightarrow \frac{dy}{dz} = y(1-y)$$

接下来就是求误差函数对权重的导数，有

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial y^n}{\partial w_i}\frac{\partial E}{\partial y^n} = \sum_n \frac{\partial z}{\partial w_i}\frac{dy}{dz}\frac{\partial E}{\partial y^n} = -\sum_n x_i^n y^n(1-y^n)(t^n-y^n)$$

反向传播算法

在得到前面的结论以后，就可以来到核心问题：如何学习多层特征，也就是隐藏层的权重。这种学习方法称为反向传播算法，于20世纪80年代提出，是那个年代人工智能领域一个最主要的成果，也引起了人们对神经网络井喷式的关注

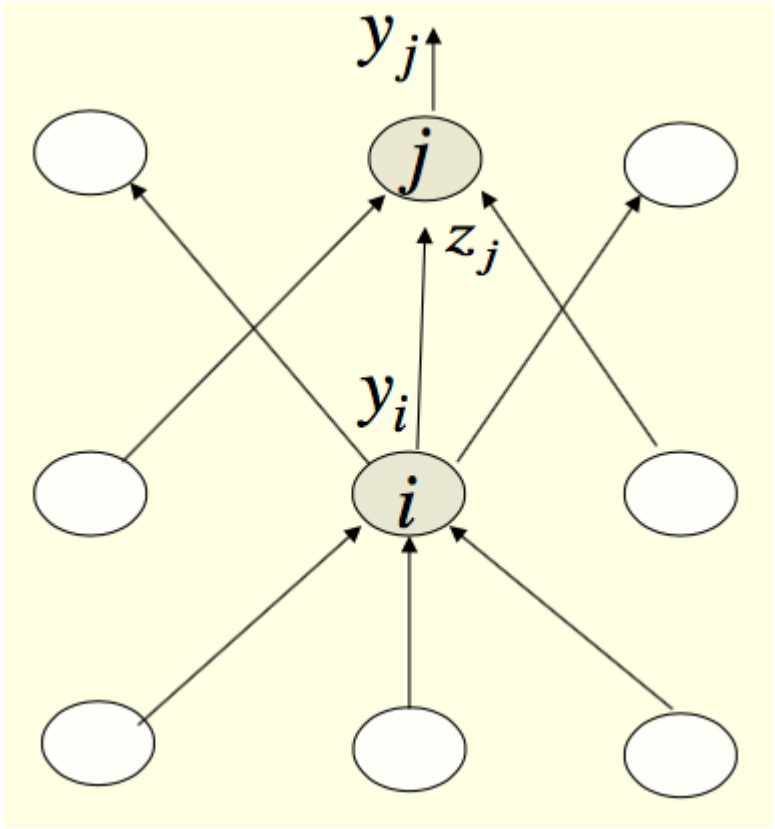
如果使用的神经网络没有隐藏层，那么其建模能力非常有限。如果像感知机那样手工加一层硬编码的特征，网络的能力是更强大了，但是为新任务设计特征又成了一个难点，而且需要开发人员手动解决

一种直接的，有点像强化学习的想法是，随机对权重做一些调整，观察对模型性能造成了什么影响。如果影响是正面的，就把这个权重保存起来。这种做法的问题是效率太低了，可能要正向传播计算很多次才能有效调整一次权重。而且判断权重效果好不好不能只看一条训练数据，需要很多训练数据才能下结论。而且当训练接近结束的时候，权重的调整不能太大，否则几乎肯定会把事情搞砸，因为此时权重的大小已经接近理想。与反向传播相比，这种做法慢的倍数大概相当于网络中权重的数量，通常会是几百万倍！

那么，可不可以并行随机调整所有权重，然后观察结果呢？也不行，因为这样还是要尝试很多遍，而每一次都是对所有权重的不同随机干扰，而随机调整权重为的是看在某个维度上的效果，那么所有在其它维度上的修改实际上都是噪声。另一种做法是随机修改隐藏层单元的激活值（activity），而不是修改权重，因为隐藏层单元的数量比权重数量少，因此这样做可能会效率高一些，但是仍然不如反向传播算法（慢的倍数变成了神经元的数量）

反向传播算法背后的思想是，我们不知道隐藏层单元应该做什么，它们被称作“隐藏层”就是因为没人能解释它们的状态应该是什么，但是对某条给定的数据，当修改某个隐藏层单元的激活值时，可以计算误差变化得多快。因此，我们不再把看隐藏层单元的激活值，而是看误差对隐藏激活值的偏导数。由于每个隐藏单元都会影响许多不同的输出单元，因此如果有很多输出单元的话，就会对总体误差造成很多不同的影响，这些影响可以被有效地组合起来，所以就可以同时有效计算所有隐藏单元的误差导数。一旦得到了误差导数，就可以知道对某个训练数据，修改隐藏单元的激活值以后误差变化多快，因此就可以知道误差对隐藏激活值的导数，进而知道误差对指向某隐藏单元的权重的导数

因此，反向传播算法的大致方法就是，首先计算每个输出单元的输出与目标值之间的差值，将这个差值转化为误差导数。然后，利用后一层得到的误差导数，计算本层的误差（相对于激活值的）导数，再使用这个导数计算误差相对于输入权重的导数。注意由于隐藏层节点通常连向多个输出层节点，因此求导时需要将它们都加起来。如下给出了一个例子



反向传播示意

上图中，节点j是输出层节点，i是它前面的隐藏层节点，输出误差仍然使用平方误差，因此正向传播时满足

$$z_j = \sum_i w_{ij} y_i, \quad y_j = \sigma(z_j), \quad E = \frac{1}{2} \sum_{j \in \text{output}} (t_j - y_j)^2$$

对应的反向传播就有

$$\begin{aligned}
\frac{\partial E}{\partial y_j} &= -(t_j - y_j) \\
\frac{\partial E}{\partial z_j} &= \frac{dy_j}{dz_j} \cdot \frac{\partial E}{\partial y_j} = y_j(1 - y_j) \cdot \frac{\partial E}{\partial y_j} \\
\frac{\partial E}{\partial y_i} &= \sum_j \frac{dz_j}{dy_i} \cdot \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \cdot \frac{\partial E}{\partial z_j} \\
\frac{\partial E}{\partial w_{ij}} &= \frac{\partial z_j}{\partial w_{ij}} \cdot \frac{\partial E}{\partial z_j} = y_i \cdot \frac{\partial E}{\partial z_j}
\end{aligned}$$

如何使用反向传播算法计算出来的导数

推导出如何计算误差关于隐藏层权重的导数，是使神经网络有效学习的关键要素。不过即便这个问题解决了，还有一些其它的问题需要关注。只有把这些问题都解决了，才能得到一个完整的、正确的学习过程

其中一些问题是优化问题。也就是，如何使用在单个数据上的误差导数来得到一组好的权重？这部分内容将在第六章讲授。另一个问题是如何保证学习到的权重可以很好地泛化推广。也就是说，如何保证它们在那些没有用来训练的数据上也表现良好？这部分内容将在第七章讲授。不过，这里将会给出一些概述

优化问题

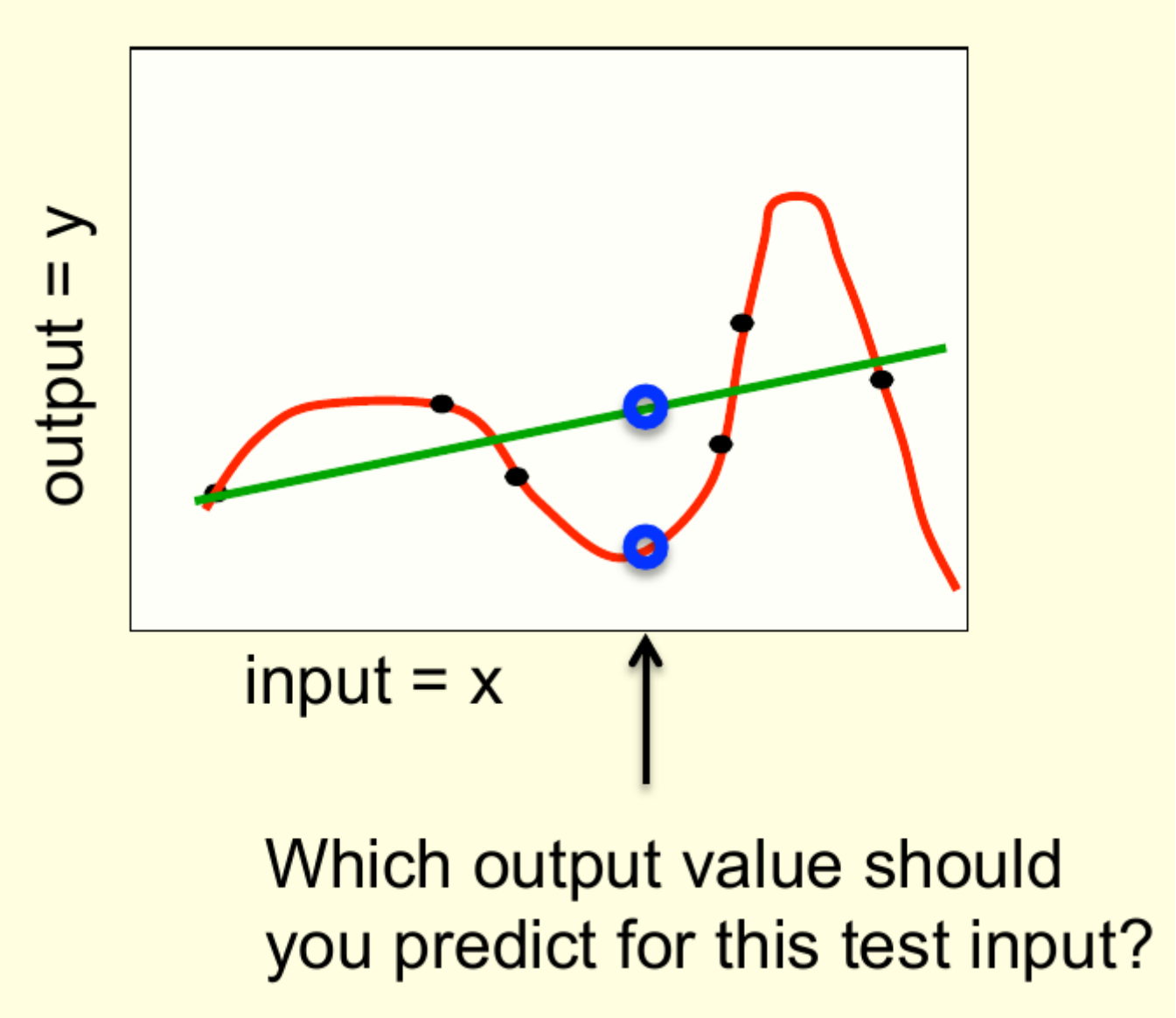
所谓优化问题，实际上关心的是如何使用权重导数。第一个问题是，更新权重的频率怎样合适？可以每看到一条数据就更新一次，此时是使用该条数据的信息通过反向传播计算误差导数，然后对权重微调。很明显，这样会导致权重波动比较大，因为对每条数据得到的误差导数都是不同的。不过通常来讲，如果对权重的修改足够小，那么还是会走在正确的方向上。一种看似不错的修改是使用完整的一批数据做训练，把所有数据上的误差导数加在一起，然后在这个方向上走一步。但是通常来讲，初始化的权重效果都不太好，而所使用的训练集可能非常大，我们并不想在所有数据上走一遍来修改某些已经可以预料到效果很差的权重。更合理的做法是，在能理性推断应该把权重往哪个方向上修改之前，先看少部分训练数据，直到训练快结束时，才看更多的训练数据。这就带来了一种折中的方法——小批量学习法（mini-batch learning）：随机挑选一部分训练样本，确定大致的方向，让权重的波动不太大。这种学习方法是最常用的

另一个问题是，权重更新多少比较合适。一种方法是手动设置一批学习率，将导数与之相乘，修改权重。更好的方法是让学习率自适应，如果误差值上下抖动不收敛，就减小学习率；如果每次权重掉得相对比较大，就增大学习率。甚至，可以为网络中的每个连接都设置一个学习率，这样有些权重就能学得快些，有些能学得慢些。再进一步，都不一定选择最陡峭的下降方向！回忆前面那个被拉长的椭圆型的误差表面，如果每一步都沿着最陡的方向走，其实是垂直于了应该下降的方向，反而不是好事。在很多学习问题中，尤其是学习接近结束时，这是一个典型问题。所以，理性的选择时选择一个别的更好的方向，不过想找到这个方向实际比较困难

泛化问题

第二个问题是，网络在它没看到过的数据上表现能有多好。问题原因是，尽管训练数据包含了从输入到输出上的一些规律信息，但是也含有两类噪声：

- 第一类噪声是目标值可能不可靠。这种情况一般不用太担心
- 第二类噪声是取样误差。也就是说，输入和输出之间存在的规律联系实际上可能是偶然造成的，因为采样时采到了一些特殊的样例——尤其是样本比较小的时候。例如，在教学生什么是多边形时，不好的做法是只拿出正方形和长方形，说“这就是多边形”。这些的确是多边形，但是对于之前没接触过这一概念的人来说，他们不会意识到多边形可以有三条边也可以有六条边，也不会意识到多边形的几个角不一定非得都是直角。如果你拿出的是三角形和六边形，那么学生又不会意识到多边形不一定非得是凸的，内角不一定非得是60度的倍数。总之，不管如何仔细地选择样本，对于样本的任意有限集合，总会有一些偶然的规律出现。而训练模型时，无法告知机器这个规律是偶然的还是必然的，所以模型对这两种规律都会去学习。而当模型足够大足够有力的时候，它会很好地拟合取样误差，造成灾难性的后果，泛化能力极差



过拟合的例子

上图给出了一个例子。对于给出的这六个点，可以使用一个自由度为2的模型（绿色直线）来拟合，它只能穿过一个点；也可以使用一个自由度为6或者更高的模型来拟合（红色直线），它可以完美地穿过所有点。但是如果新的输入数据是箭头指向的输入值，应该用哪个模型预测的结果呢？能很好拟合*大量*数据的模型是令人信服的，而复杂模型在小数据集上表现良好是稀松平常的。对于目前的例子，大多数人会选择绿线上的蓝点作为预测值。但是如果给出十倍的数据，这些数据都离红线很近，大家就会信任红线了

这种对数据拟合过于完美，以至于很好拟合了取样误差的现象，叫做**过拟合**。这里，简单介绍一下神经网络中用来防止过拟合的方法，但是更细致的讨论留待之后（大部分在第七章）

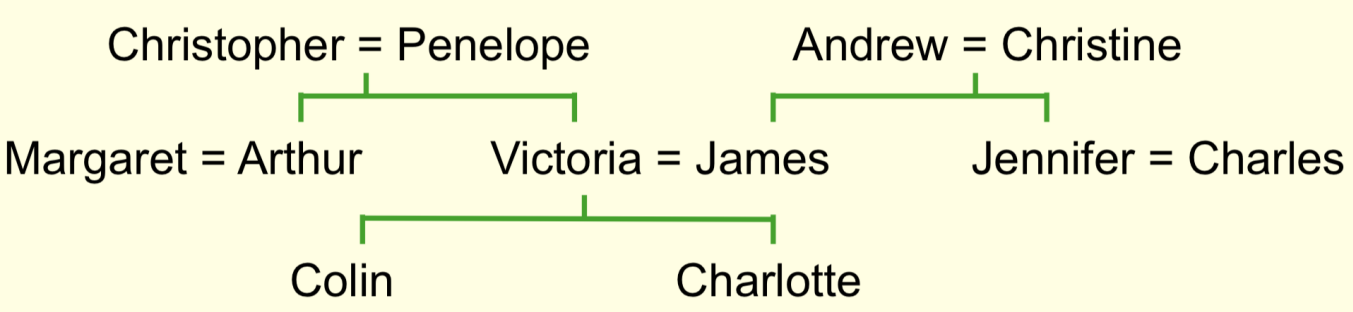
- 权值衰减（weight decay）：试图将神经网络的权重稳定在一个小值上，甚至上大多数权重都为0。这样做是为了让模型简单
- 权值共享（weight sharing）：让很多权重保持取相同值，也可以让模型简单。你不必须要知道这个值应该是什么，实际上这个值是学出来的。不过要保证很多权重都是这个值
- 提早停止（early stopping）：选定一个伪测试集，观察网络在这个数据集上的性能。如果性能下降，停止训练
- 模型平均（model averaging）：训练很多不同的网络，将它们平均起来，以减少误差
- 贝叶斯拟合（Bayesian fitting）：模型平均一种更漂亮的做法
- 神经元丢弃（dropout）：通过在训练时随机关闭一些隐藏单元来使模型更加鲁棒
- 生成预训练（generative pre-training）：一种更复杂的方法，将在本课将结束时介绍

Hinton神经网络与机器学习 4. 神经语言模型与词向量

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/12/22/UTNN-4-Learning-Feature-Vectors-for-Words/

学习预测下一个单词

本讲将介绍反向传播算法的一个有趣的应用，即可以用它来学出对单词意义的一种特征表示。这种做法可以追溯到20世纪80年代，那时人们使用神经网络来理解家族树（family tree）中所包含的信息。下图给出了家族树的一个示意图



家族树的示意图

家族树中包含了大量的人名，同时可以定义一些谓词来表示这些人之间的关系，包括父子关系、夫妻关系、叔侄关系/姑侄关系等等。其中，在给定了某些关系的前提下，一些关系可以推导出来。例如：

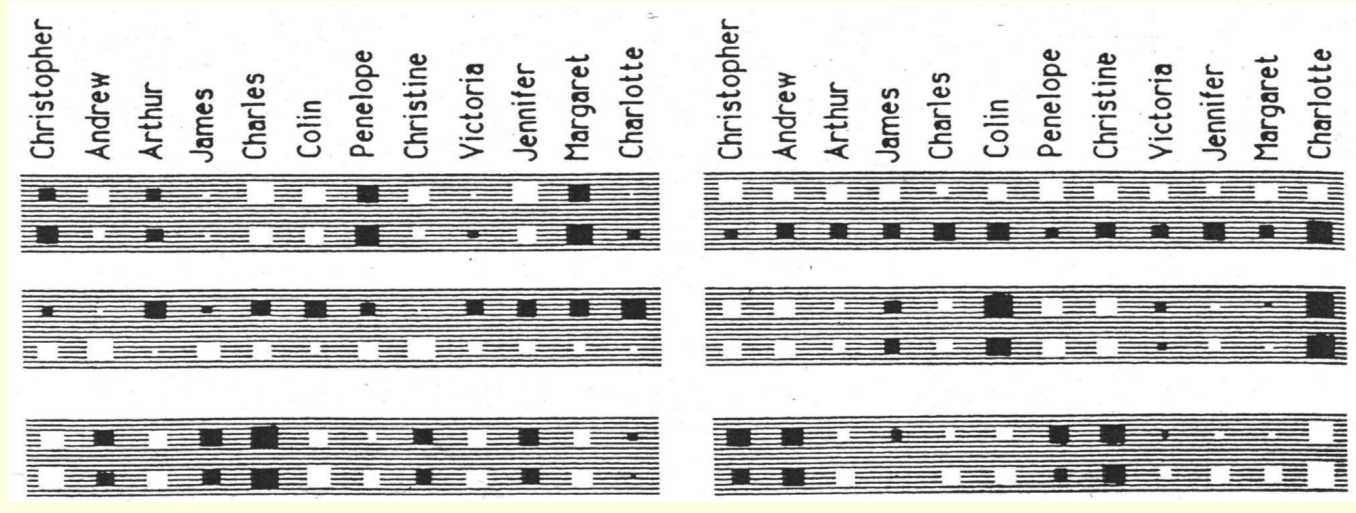
$$(\text{colin}\ has\text{-}father\ \text{james}) \wedge (\text{colin}\ has\text{-}mother\ \text{victoria}) \rightarrow (\text{james}\ has\text{-}wife\ \text{victoria})$$

因此，具体的学习问题就是，给定大量表示家族树中信息的元组，学习出这些谓词之间的关系。最简单的办法是手写这些规则，但是这样会使得搜索空间变得很大，因此研究人员试图让神经网络在一个由实数权重组成的连续空间中搜索并捕捉信息。如果送给神经网络一个人名和一个关系名称，它能够给出对应的人名，那么就说这个神经网络学成了。例如，对上面的例子，神经网络的输入如果是James和wife，它应该能输出Victoria

设计的时候，网络的输入层对人名和亲属关系都进行了独热编码。对每条数据， 只有两个神经元被激活，分别对应一个人名和一个关系。而输出是24个神经元，对每条数据只有一个神经元被激活。使用独热编码的好处是，对这24个向量组成的集合，其任何子集中的向量都是线性无关的，而且不需要对每个人名有任何先验知识

接下来，对人名部分，会有6个神经元与之相连。因为这一层的神经元数小于输入层神经元数目，因此每个神经元不可能只对应于一个人名。这样，神经网络需要把每个人名表示成这6个神经元的某种行为模式。换句话说，这一层实际上可以学出来每个人名的分布式编码（输入的独热编码则对应地被称为局部编码）

学习到的结果如下图所示



人名的分布式表示（向量表示）示意图

这里一共有6个灰色块，对应着6个神经单元。每一个灰色块里包含了24个大小不一的黑白色块，其中大小表示了权重的绝对值大小，黑色表示负值，白色表示正值。每个灰色块里的第一行对应于英语人名，第二行对应于意大利语人名。图中已经给出了各个英语人名分别对应于第几个色块

从图中可以发现一些学习到的有趣事实

- 右上角的灰色块中，英语人名的权重都是正的，意大利语人名的权重都是负的（第一行都是白的，第二行都是黑的），这说明对于给定的名字，神经网络可以判断其是属于哪种语言。因此对给定英语的输入，它断不会给出意大利语的输出
- 第二行右边灰色块中，有四个大的白块，对应于英语的Christopher, Andrwe, Penelope和Christine（以及意大利语的相应人名）；而两个大的黑块对应于英语的Colin和Charlotte（以及意大利语的相应人名）。这里值的符号可以理解为，大的正值意味着人的辈分高，而小的负值（绝对值大的负值）意味着人的辈分低，中坚一代则是在0附近波动。即这个神经元反映了辈分信息
- 第三行左边的灰色块中，Andrew、James、Charles、Christine和Jennifer（甚至包括Charlotte）都是负值，对应于家族树中，这些人都来自树的右侧。也就是说，模型可以学出给定的人来自树的哪一支

令人兴奋的是，以上信息都没有被人编码在数据里，而是由神经网络根据给定数据自己学出，它能自动判定在这个领域里哪些特征是好的特征。当然，这些特征起作用的先决条件是其它信息（例如亲属关系）使用相似的表现形式，以及后面的隐含层能够学到如何使用这些特征预测其它特征

另一种观察神经网络如何工作（以及对其工作进行评估）的方法是，对已有的112条关系随机选出108条来训练，然后看神经网络在剩下4条数据上的表现。这项在20世纪80年代进行的实验最后能得到50%到75%的准确率，对于一项24分类的任务来说，这个结果已经足够可喜可贺，而且如果训练集更大，效果应该会更好

如今，我们有更大的数据库，里面可能有百万条的关系数据，每条数据都可以写成$A\ R\ B$的形式（即A与B存在关系R）。对于这样的数据，可以训练两种网络

- 一种网络是前面介绍的原型的延展，也是接受A和R做输入来预测B，这样可以找到数据库中不太可能出现的元组，即对已有数据进行纠错
- 另一种类似，但是是接受A、R和B做输入，并为这些输入打标记，0为不可能，1为可能，以此来预测某个三元组为真的概率

对家族树例子的简明神经科学解释

本节可能会引起科研人员的兴趣，但是估计不合工程人员的胃口。因此如果你是工程师，可以跳过本节:)

在计算机科学圈子里，有一项争论估计持续了将近一百年，就是概念应该是用一个特征向量来表示，还是用其与其他概念之间的关系所表示

- 特征学派认为，概念实际上就是语义特征组成的一个大集合。这样的定义有助于解释概念之间的相似度，而且对机器学习很方便
- 结构主义理论认为概念的意义寓藏于其与其它概念的关系之中，因此概念知识最好表示为关系图，而不是向量

Hinton认为这两派都错了，因为它们认为这两者是你死我活的，但实际关系并不是这样，比如神经网络就是使用语义特征向量来实现关系图。前面那一小节已经证明，神经网络可以在不给定推导规则的前提下，根据给定的人名和关系，通过一遍正向传播的计算就可以找出对应的人名，其原理是许多概率特征互相影响。这种特征一般被称作“微特征”，来强调说它们并不是显式的，有意识的特征。就像大脑一样，大脑有上百万个神经元，神经元之间还有上百万个交互，因此有的时候人可以“扫一眼”就得出结果。这中间可能没有有意识的思维过程，但是这也是神经元之间通过交互大量计算得到的结果。有时候，人可能会使用一些外在的规律来做一些有意识的、细致的推导，但是有时候也会凭直觉，下意识的思维来做事情

有很多人在考虑如何使用神经网络实现关系图的时候，通常假设一个神经元代表关系图中的一个概念，神经元之间的连接表示一个二元关系。不过这种想法一般不能实用，因为现实中“关系”可能有很多种形式，而神经网络中神经元的连接只有大小，没有类型之分，而且这样也不方便表示三元关系。真正应该怎么使用神经网络来实现关系图还没有一个公认很好的解决方案，但是很大可能是用多个神经元表达某一个概念，而且每个神经元会用来参与表示不同的概念。这种表示法就称为概念的分布表示，也就是说，概念和神经元之间的关系是多对多的

softmax输出函数

softmax输出函数会让神经网络的输出之和变为1，因此可以表示输入样本属于多个互斥离散集合的概率分布。那么为什么不用之前提到的平方误差呢？假设期望输出为1，实际输出为十亿分之一，那么梯度将不会被反向传播来让logistic单元发生变化，因为此时进入了激活函数的饱和区，梯度近似为0。即这样非常大的误差实际上基本不会修改网络中的权重。另外，如果要设计的输出是某个样本属于不同标签（标签间互斥）的概率，那么实际上我们知道输出总和应该是1，这个先验知识应该告知给神经网络。因此，可以修改输出函数来传递这一特征，即使用softmax函数。这个函数可以看作是max函数的软连续（soft continous）版本：记第i个神经元的输出为z_i（称为分对数logit），其经过softmax函数转换得到的输出不只依赖于z_i本身，而是依赖于这一组中所有的z，即

$$y_i = \frac{e^{zi}}{\sum_{j\in\text{group}} e^{zj}}$$

显然，y_i满足如下两个性质：

- $0 < y_i \leq 1$
- $\sum_i y_i = 1$

因此y_i就可以看作是输入属于类别i的概率值。其对原始输入z_i的梯度为

$$\begin{aligned}\frac{\partial y_i}{\partial z_i} &= \frac{e^{zi}\sum_j e^{zj} - e^{zi}\cdot e^{zi}}{(\sum_j e^{zj})^2} \\ &= \frac{e^{zi}}{\sum_j e^{zj}} - \left(\frac{e^{zi}}{\sum_j e^{zj}}\right)^2 \\ &= y_i - y_i^2 = y_i(1-y_i)\end{aligned}$$

现在的问题是，如果使用softmax函数做输出函数，对应的代价函数应该怎么选择？答案是像往常一样，最合适的代价函数是正确答案的负对数概率，也就是说，要最大化得到正确答案的概率的对数值。因此，如果目标值的某一位为1，其余位为0，则只需要将所有可能答案相加，让错误的答案与0相乘，让正确的答案与1相乘，就能得到正确答案的负对数概率。得到的这个代价函数称为**交叉熵代价函数**，假设目标值为$\mathbf{t}$，在第j个维度的分量为t_j，则交叉熵代价函数C可以写为

$$C = -\sum_j t_j \log y_j$$

交叉熵代价函数的一个优美性质是当期望目标值为1，输出接近0时，梯度非常大。使用链式求导，有

$$\frac{\partial C}{\partial z_i} = \sum_j \frac{\partial C}{\partial y_j}\frac{\partial y_j}{\partial z_i} = y_i - t_i$$

尽管$\partial y_j / \partial z_i$可能很平，但是$\partial C / \partial y_j$会很陡，一定程度上减轻梯度饱和的影响

附：向量化求交叉熵代价函数梯度的过程

（这也是CS224n的一次作业）

假设期望的输出为$\boldsymbol{y}$，实际输出为$\boldsymbol{t}$，softmax的输入为$\boldsymbol{z}$，则有

$$\begin{aligned}\boldsymbol{y} &= \mathrm{softmax}(\boldsymbol{z}) \\ C(\boldsymbol{t}, \boldsymbol{y}) &= -\sum_i t_i \log(y_i)\end{aligned}$$

由于$\boldsymbol{t}$是独热编码，不失一般性，假设第k个分量为1，其余分量均为0，则有

$$t_i = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{elsewhere} \end{cases}$$

因此展开交叉熵代价函数为

$$\begin{aligned}-\sum_i t_i \log(y_i) &= -t_k \log(y_k) = -\log(y_k) \\ &= -\log \frac{e^{zk}}{\sum_i e^{zi}} = -z_k + \log \sum_i e^{zi}\end{aligned}$$

所以

$$\begin{aligned}\frac{\partial C(\boldsymbol{t}, \boldsymbol{y})}{\partial \boldsymbol{z}} &= \begin{cases} -1 + \frac{e^{t_i}}{\sum_i e^{t_i}} & \text{if } i = k \\ \frac{e^{t_i}}{\sum_i e^{t_i}} & \text{elsewhere} \end{cases} \\ &= \boldsymbol{y} - \boldsymbol{t}\end{aligned}$$

神经概率语言模型

利用前面提取姓名特征的方法，可以得到单词的特征向量表示，得到的向量称为**词向量**。词向量在语音识别领域非常有用（笔者注：现在词向量基本已经成了NLP各个问题的基石，不再被语音识别所独享）：如果背景音比较嘈杂，那么很难单纯从原始的声音输入辨别音素。对于同样的声音信号，可能有多个不同的单词都能与之很好对应。例如，如果背景音嘈杂，就很难分辨beach和speech这两个词。人能做到这一点是无意识的，但是同样的方法不能用在机器上。因此，语音识别的研究人员需要根据给定的上文来判断下一个单词更可能是哪个，即建立一个**语言模型**

传统的语言模型是使用N元语法，通常N为3。即，采集大量语料，统计所有三元词组出现的频率，然后根据这些频率信息利用相对概率来猜在给定了前两个单词的情况下第三个单词会是什么。例如，假设前面两个词已知是a和b，判断下一个词是c还是d，则有

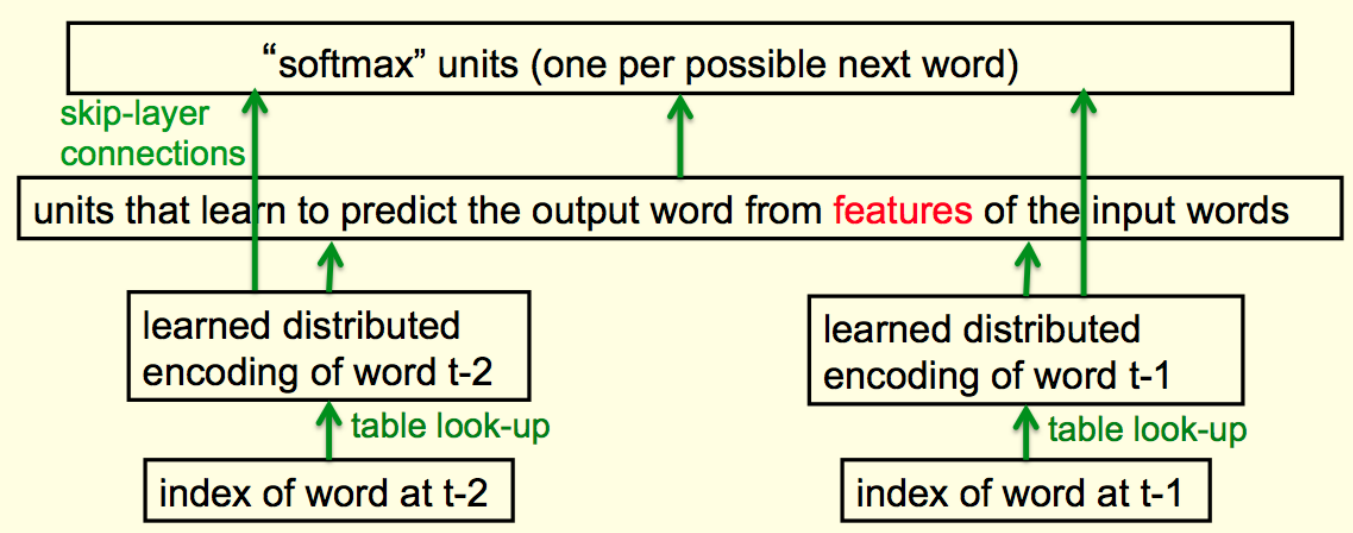
$$\frac{p(w_3 = \mathrm{c} | w_2 = \mathrm{b}, w_1 = \mathrm{a})}{p(w_3 = \mathrm{d} | w_2 = \mathrm{b}, w_1 = \mathrm{a})} = \frac{\mathrm{count(abc)}}{\mathrm{count(abd)}}$$

三元语法的问题是对上文的信息利用有限，它不能利用更多的上文进行推断。其原因一方面在于，使用类似的方法，上文越丰富，需要存储的频率数据越多，可能会压爆内存；另一方面是，词组空间变大会使得大部分词组出现的次数接近于0，因此概率趋近于0。有时，如果三元组不存在或者概率太小，则需要回退到二元语法，考察二元组出现的频率。也就是说，使用这种方法，大部分情况下说“ab后面接c的概率是0”只是因为某个词组出现的频率为0，而这种做法是不正确的

此外，三元语法难以捕获一些更深层的信息。例如，假设看过“the cat got squashed in the yard on Friday”，那么理应可以对“the dog got flattened in the garden on Monday”这样的句子中的词做出不错的预测。但是N元语法做不到这点，因为仅凭统计学上的信息难以得出dog-cat，squashed-flattened，yard-garden，Friday-Monday这样的关系

为了突破N元语法的种种限制，一种解决思路是把单词转化成包含语义和语法特征的向量，然后使用前面若干词的特征来预测后面单词的特征。使用这种特征表示法可以把更大的上文利用起来，例如可以使用前十个词的特征

Bengio等人提出了一种神经网络结构来预测下一个单词，这个结构如图所示



Bengio等人提出的神经网络，用来预测下一个单词

网络的最底层可以看作是单词索引，也可以看作是一组神经元，只是每时每刻都只有一个神经元被激活。这个被激活的神经元会决定接下来这个隐藏层的行为模式，也就是其所对应的单词的分布式表示（即这个单词的词向量）。由于索引和词向量一一对应，因此整个过程等价于查表，但是这个词向量可以通过学习过程来修改

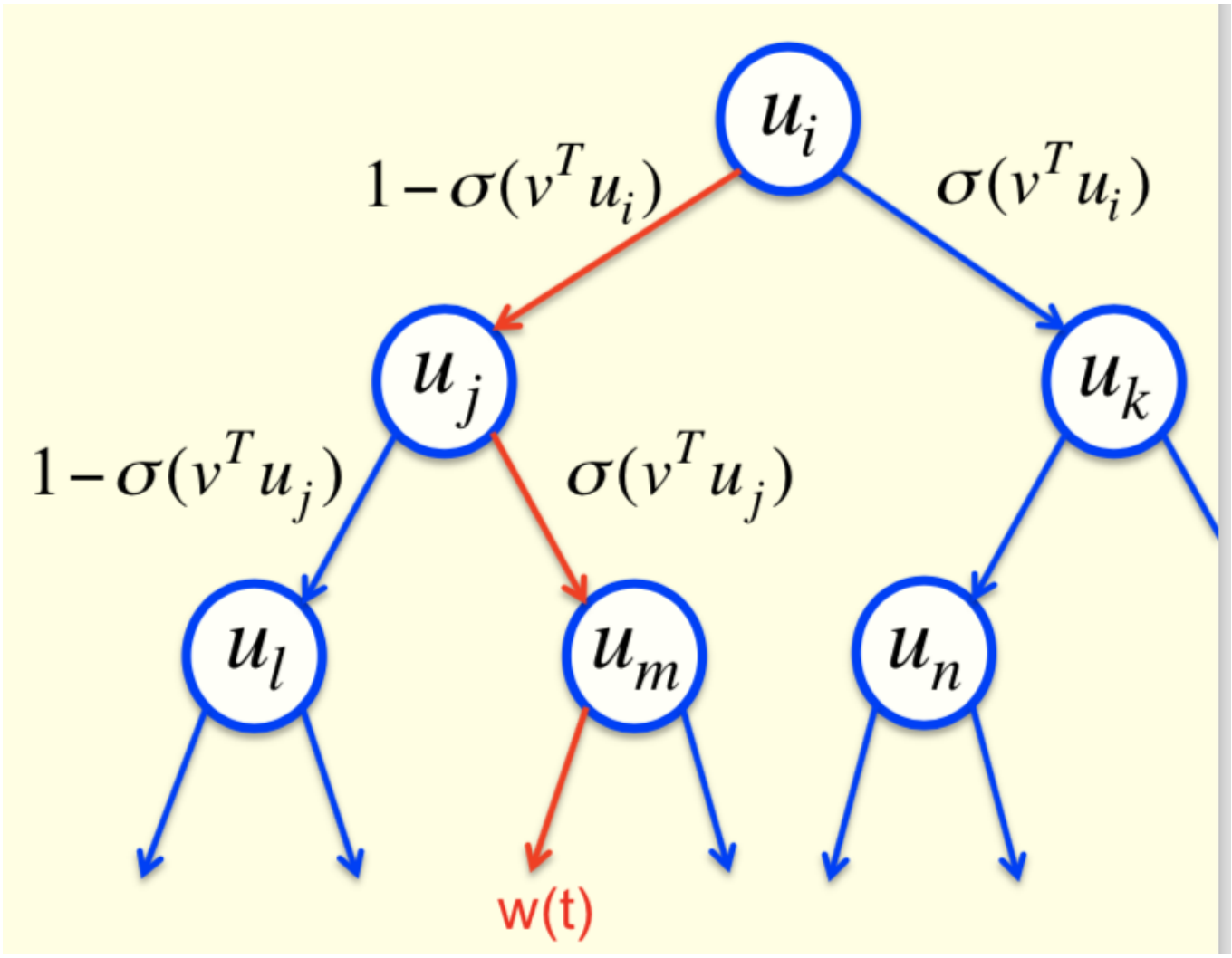
有了词向量，就可以通过另一个隐藏层和softmax层来得出所有单词出现在下一个位置上的概率。不过，更好的做法是是直接用学到的词向量连接到softmax层，因为词向量本身就包含了足够的信息来预测下一个单词

这个结构的问题是最后的输出层很大，需要处理大概十万个单词（因为这种情况下通常一个词的不同形态对应不同单词，例如cat和cats被看做是两个单词），因此整个网络中权重就更多，容易过拟合。如果我们不像Google那样有大量的数据，就需要试着减小隐藏层单元的数量。但是，这样就难以算出这十万个单词的准确概率，会导致大量单词的概率非常小，而这些小概率通常会比较相关。例如，语音识别工具要判断两个罕见词哪个更应该出现，即便两者出现的概率都不大。因此，需要一种方法来处理这样大量的输出

处理神经概率语言模型的大量输出

为了避免在输出层设置十万个输出单元，一种比较有效的方法是使用序列结构。也就是说，前面还是上文单词，但是最后放置要输出单词的一个候选词，这样，神经网络的输出是这个候选词的分对数得分。这种做法的问题是当候选词有多个词的时候需要把神经网络跑好几遍。在得到所有可能候选词的得分以后，可以把这些得分单独送进softmax函数，得到每个单词的概率。使用这个预测概率和目标概率（实际通常是一个某维度为1其余为0的向量）可以计算两者的交叉熵，利用这个误差函数可以反向传播梯度更新权重。如果候选词集合比较小（例如先用N元语法筛一遍），这种做法还是很有效的

另一种做法是使用二叉树，树的叶子节点是所有单词。使用前面单词的上文信息可以得到一个预测向量$\boldsymbol{v}$，然后将$\boldsymbol{v}$与树中的每个内部节点学到的向量$\boldsymbol{u}_i$做内积，对得到的内积计算一次logistic函数，则走向右子树的概率为$\sigma(\boldsymbol{v}^\mathsf{T}\boldsymbol{u}_i)$，走向左子树的概率为$1-\sigma(\boldsymbol{v}^\mathsf{T}\boldsymbol{u}_i)$。这样沿着某条确定的路径走下去就可以找到期望的单词。具体做的时候，只需要把上文中单词的词向量加起来就能得到预测向量$\boldsymbol{v}$。接下来，对于某个已知的期望单词，要做的就是让从根节点走向这个单词的路径概率尽可能大。例如下图中就要让红色的路径（$i \to j \to m$）的概率尽可能大



使用树结构帮助预测单词并减少输出层大小

因此，为了让选中正确目标单词的概率最大化，就需要试着最大化选中通向正确单词路径的对数概率和。因此在学习时，只需要关注正确路径上的节点。相比于原来$O(N)$的复杂度，这种做法在学习时的复杂度降低到了$O(\log N)$。对每个节点，由于知道下一个单词是什么，因此知道正确的路径。而且，通过将预测向量与每个节点的学习向量求内急，可以知道选择每条路径的概率。因此，可以求得误差函数关于$\boldsymbol{u}_i$和$\boldsymbol{v}$的梯度，所以训练起来会非常快。不过这种做法的问题是测试时仍然很慢，因为此时需要考虑很多单词的概率，所以不能只专注于一条路径

Collobert和Weston提出了一种更简单的学习词向量的方案。他们做的不是预测下一个单词，而是只是学出一些词向量，然后将这些词向量用在一些不同的NLP任务中。此外，他们不只使用上文信息，还是用下文信息，因此实际使用的是一个长度为11的词窗口，其中有5个词出现在上文，5个出现在下文。窗口中间位置他们选择放一个正确的单词（的确出现在这个语境里）或者放一个随机挑选的单词。训练出来的神经网络应该对正确的单词输出高分，对随机的单词输出低分。训练预料均来自于wiki，大概有6亿条数据。得到的词向量使用t-sne降维到二维空间并做出图以后可以展现出令人吃惊的效果，语义上相近的词在空间中也聚集在一起。这意味着上下文信息实际上能够透露很多关于单词意义的信息，事实上，有些人认为这也是人类学习词语意义的主要方式。例如，假如你从未听说过scrommed这个单词，但是看到she scrommed him with the frying pan这句话，应该也能猜出来这是什么意思

（后面具体例子略，毕竟现在词向量已经为人所熟知了:)）

参考文献

Collobert, Ronan, and Jason Weston. “A unified architecture for natural language processing: Deep neural networks with multitask learning.” *Proceedings of the 25th international conference on Machine learning (ICML)*. ACM, 2008. (pp. 160-167)

Mnih, Andriy, and Geoffrey E. Hinton. “A scalable hierarchical distributed language model.” *Advances in neural information processing systems (NIPS)*. 2009. (pp. 1081-1088)

Hinton神经网络与机器学习 5. 卷积神经网络

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/12/23/UTNN-5-Object-Recognition-with-Neural-Nets/

为何物体识别很困难?

尽管人类在计算能力上能被计算机甩开好几百条街，但是有一项任务做起来却是得心应手，比计算机高到不知道哪里去了。这项任务就是物体识别——人类是如此擅长这件事情，甚至不知道做起来有什么难点。但是同样的事情，机器做起来可谓是困难重重

- 难以从图片中把要识别的物体分割出来（segment out）。在现实生活中，我们可以在移动中寻找线索，也可以通过肉眼得到三维的立体线索，但是这些线索在静态图片中都不能用。因此，仅凭图片，很难分清哪些东西属于同一个物体。此外，目标物体的某一部分可能隐藏在其他物体的后面，因此难以看到其全貌
- 其次，像素密度受光的影响非常大，这种影响程度甚至等同于物体本身对像素密度的影响程度。假设某种黑色表面的物体置于明亮光中，其像素密度可能比置于暗光下的白色物体的像素密度还要大。而要识别物体本质上是对数字的操作，这些数字就是像素的密度
- 物体的形态还会变化，有同样名称的两个物体其形状可能不同。例如同样是手写数字2，有的人在底部会写成横，有人会绕个弧
- 有些物体类别的定义是由功能决定（affordance，功能决定性），而不是由外观决定的。例如椅子实际上是一种坐具，因此太师椅、藤椅等等都可以算作是椅子，虽然它们的形状不同
- 观察物体的角度可以不同。在现实生活中，人类可以根据不同角度的观察结果辨识出一个三维物体，但是同一个物体观察的视角不同形状也会不同，输入的信息（在物体识别问题中是像素点）就会发生比较大的变化，使得传统机器学习方法难以处理。举个例子，假设现在要做一个简单的机器学习模型，输入数据的两列应该是病人的年龄和体重，但是有的人填数据的时候填错了，“年龄”一栏填的是体重，“体重”一栏填的是年龄，那就必须在训练模型之前先把这种混乱的数据处理掉。这种现象称为“维度跳跃”（dimension hopping）

如何使算法不对视角敏感

被观察的物体会因视角不同导致形状发生变化，使得同一个物体送进算法的数据（像素）是不同的，这是计算机视觉面临的一个主要难题，而且现在（2012年）暂时还没有广泛被接受的解法。目前主流的做法分为四种

- 使用冗余的，表示不变性的特征
- 把要识别的物体用一个框框住，然后使用正则化的像素
- 使用重复特征，然后做池化
- 使用相对于镜头或屏幕不变的部分

第三种方法也称为**卷积神经网络**（Convolutional Neural Nets, CNN），将在第三节介绍。第四种方法将在本课程后面部分介绍。本节主要讲述第一、二种方法

使用不变性特征

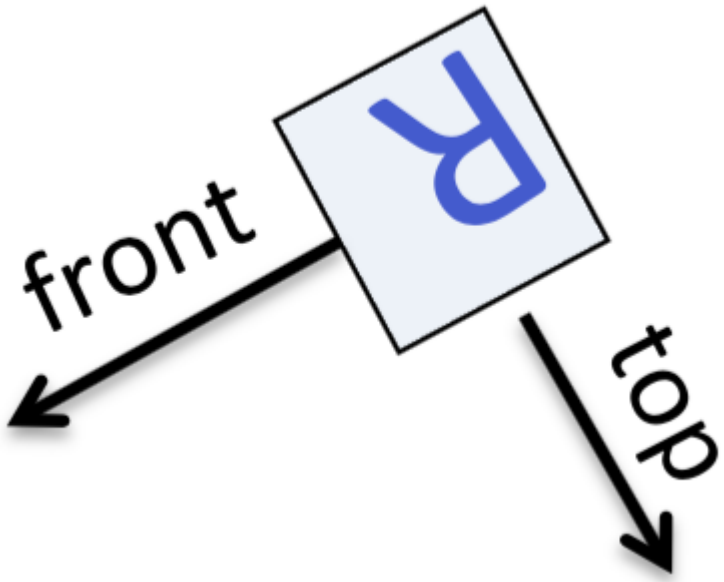
这种方法的意思是，图像中有一些特殊的特征，即便是在图像被变换（翻转、平移或缩放）后这些特征本身也是不变的。因此应该提取大量的、冗余的这种特征。例如，假设图像中某两条平行线中间有一个红点，这种特征肯定不会因图像变化而变化

有了这些特征以后，接下来的问题是如何把它们组合起来。实际上的解决方法很简单，就是不需要组合，不需要直接表示特征之间的关系，因为这些关系会被其他特征所捕捉。因此，实际上只需要把这一堆特征送进模型，因为模型之间有交叠和冗余，因此其中某个特征会说明其他两个特征是怎么关联的

不过，使用不变性特征做物体识别时，最重要的一点是，不要把来自不同物体的几部分组合起来

合理归一化

第二种方法称为合理归一化（judicious normalization）。例如，下面这个R经过上下颠倒以后在周围加了个框。然后，就可以在框上标明顶部（top）和前部（front）



颠倒的R，加上框

这样一来，就可以说R在框的后部有一个竖线，在顶部有一个弯，等等。无论视角怎么变，如果这个框画对了，这些位置特征相对于这个框来说就是不变的，物体的同一个部分总会有相同的归一化像素。需要说明的是，这里“框”是一个抽象概念，实际使用时不一定非得是长方形

但是这种方法的难度在于，很难选择这个框，原因如下

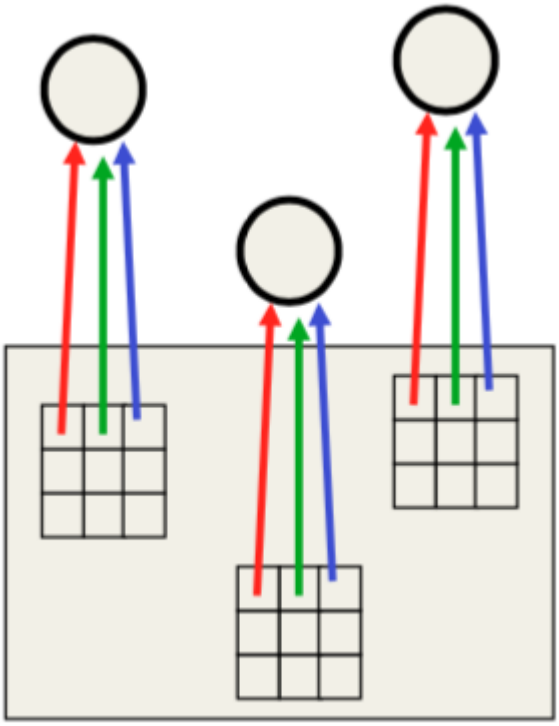
- 分割物体时，可能会有误差
- 物体的一部分可能会被其它东西挡住
- 物体的朝向有时候会与我们经常看到的朝向不同。例如上面这个R就是倒着的。而且，在标记上面这个边框哪里是上面，哪里是前面时，用到了一些先验知识。如果现在要做的是手写字母识别，有个人写了一个蹩脚的小写d，然后在上面画蛇添足加了一笔，那么再画框的时候前后和上下与上面的例子就是相反的了。这意味着，要想正确画框，就要先认出图像的形状；要想认出正确的形状，就要先画对框。这就成了一个鸡生蛋蛋生鸡的问题

接着上面的最后一条扩展一下，一些神经科学家认为人们看到*任何*方向不对的物体时都会在心中有个旋转的过程，但是Hinton认为这纯属无稽之谈：人们一眼看上去就知道是R，而且是上下颠倒了，只有这样才能知道怎么把这个字母正过来。真正要用到心算的时候是判断一个物体的“手性”（即是否镜像）时

因此，可以引出一种称为“蛮力正则化”的方法，即在训练时使用正规的，正确分割的物体，因此可以在训练识别器的时候使用合理归一来把物体框起来。测试时，对零散的物体，使用各种可能的框去框物体。这种方法在2012年时是一种比较受欢迎的方法，因为用它来检测未分割图片中角度比较正的物体（例如人脸和房子）比较好用。进一步地，尝试所有可能的框时，可以使用一种粗糙的网格法。这样识别器就可以处理一些变化特征，例如位置的变化和缩放信息

用于手写数字识别的卷积神经网络

卷积神经网络的灵感来自于重复特征。由于物体总是移动，其表现形式（像素点）也不同，因此如果某个特征检测器在图片的一部分起作用，它在别处可能也能用上。因此，可以在所有不同的位置构建很多相同特征检测器的不同拷贝



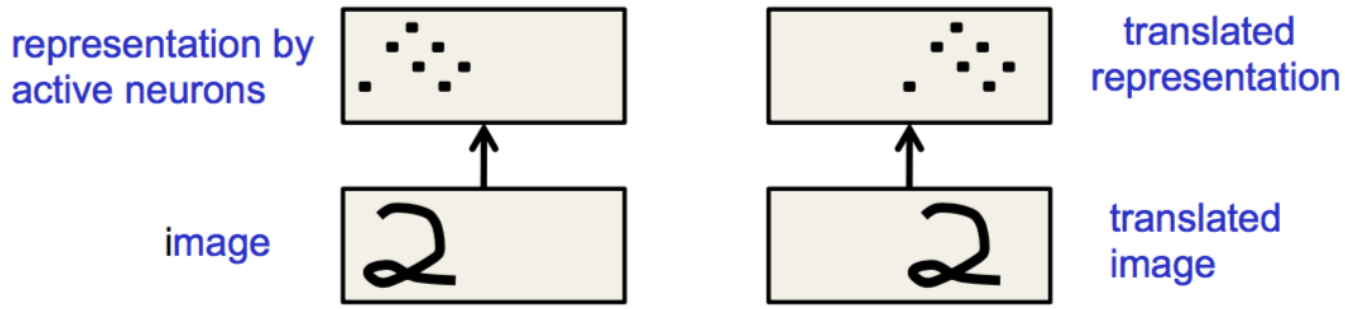
重复的特征检测器

上图给出了三个特征检测器，三者都是互相重复的。每个检测器都有9个权重，对应于9个像素点，而且对每个检测器，它们的权重都相同。例如，对于三个引出红箭头的格子，它们使用的权重都一样。这种跨区域的重复

性可以极大减少要学习的参数数量。记检测器的权重为w，检测器检测的区域为x，则x通常被称作**输入**，w通常被称作**核函数**，它们的输出$x * w$有时被称作**特征映射**

上面这种特征检测器可以看做是图片中某个局部区域的一种映射，也可以看作是一种特征类型。为了识别物体，肯定要训练多种类型的特征，这样，图片的每个部分可以被不同类型的特征所表示。重复特征也易于通过反向传播来学习，实际上，对反向传播算法稍加改动就能加入这种权重之间的线性约束关系。具体做法是，先按照往常一样计算梯度，然后修改梯度使得它们满足所受的约束。因此，如果在权重更新之前它们满足线性约束，在这样更新以后约束仍然存在。例如，假设约束条件是$w_1 = w_2$，同时这两者在权重更新前就相等，那么为了保证权重更新后两者依然相等，就需要使得$\Delta w_1 = \Delta w_2$。为此，可以计算$\frac{\partial E}{\partial w_1}$和$\frac{\partial E}{\partial w_2}$，然后令$\Delta w_1 = \Delta w_2 = \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2}$，就可以保持这样的约束条件

需要注意的是，重复的特征并不会让被激活的神经元有平移不变性，只不过会让神经元有等变（equivariant）的行为，可以看下图的例子

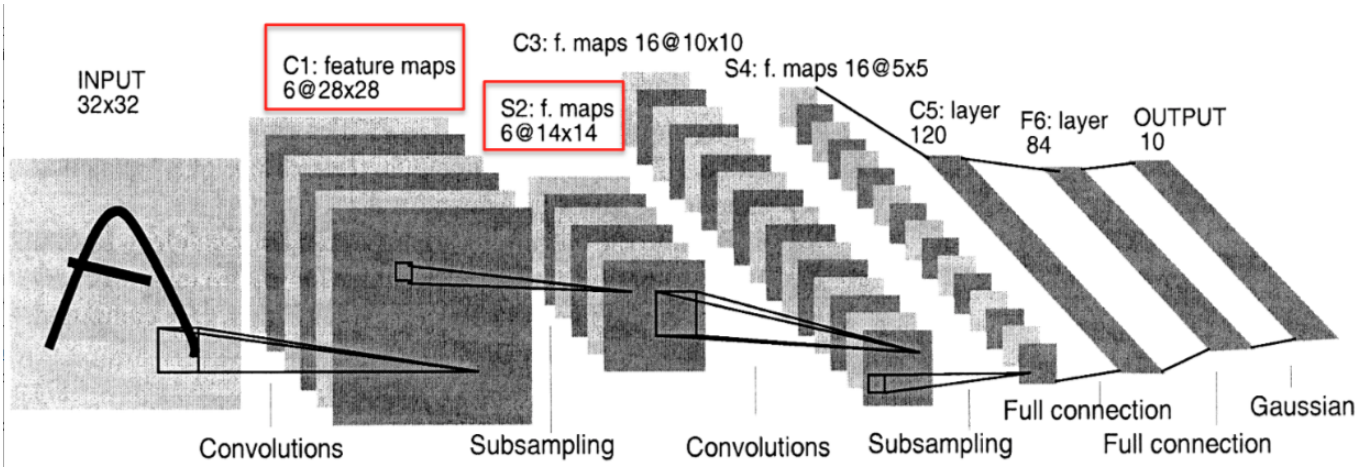


重复特征使神经元有等变行为

很显然，在这两种情况下，手写数字的位置是不一样的，激活的神经元的位置也是不一样的，但是被激活的神经元展现出来的模式相同。CNN得到的真正的“不变性”是知识的不变性：如果某个特征在训练时对某个区域有用，那么这个特征的检测器在测试时对所有的区域都可用。也就是说，等变的是神经元的行为，不变的是权重

如果想让神经网络对神经元行为取得某些不变性，那么就需要对重复特征检测器的结果做**池化**（pooling），例如对四个邻居检测器的结果求平均，将得到的这个平均值送入下一层，可以实现一小部分平移不变性。这样做的好处是可以减少下一层接受的输入数量，所以可以产生更多映射，在下一层学习更多特征。不过通常情况下做的池化操作是求最大值（这种池化操作称为最大池）。这样做的问题是经过几层池化可能会丢失物体准确的位置信息，例如，假设要做的任务是检测出有没有人脸，那么经过池化以后大致能看到哪里是鼻子，哪里是嘴，根据粗略的位置，可以做出判断。但是如果要做的任务是检测出这是谁的脸，就不好办了。因为池化操作使得眼睛跟眼睛，鼻子和嘴唇之间的精确距离信息都丢掉了

Yann LeCun团队设计的LeNet是卷积神经网络的一个成功应用，它具有很多个隐藏层，每一层有很多重复单元的映射，而且每一个隐藏层都会对周围重复单元的结果做池化，再将池化结果送入下一层。下图给出了LeNet5的一个例子，其中“子采样层”（subsampling）就对应于现在说的池化层。更多的结果可以参看Yann LeCun的



需要注意的是，在设计LeNet时，里面其实也使用到了人类的一些先验知识，例如局部连通性、权重限制、以及选出何时的神经元激活函数等等。另一种方法是使用先验知识构造大量训练数据。当训练数据的数据量变大以后，尽管训练时间会变长，但是优化方法可以帮助网络找到一些聪明的，人类之前想不到甚至没法完全理解的处理方式。将这两种方法（权重共享/池化 + 人造数据）可以通过类似蛮力法的方法比较好地解决手写数字识别的问题。瑞士的一个研究团队可以达到只出现35个错误的成绩，此外，他们还通过训练多个模型然后将模型进行组合的方法，将错误数量压低到25个

这里引出了另一个问题：如果测试集有10000条样本，模型A判断错40条，模型B判断错30条，我们能确定地说B比A好吗？不一定。来看一下下面两个McNemar检验矩阵

	model 1 wrong	model 1 right
model 2 wrong	29	1
model 2 right	11	9959

	model 1 wrong	model 1 right
model 2 wrong	15	15
model 2 right	25	9945

两种情况下的McNemar检验矩阵

对于McNemar检验，着重要看的是红色的两个数据。对于第一种情况，“模型B正确但模型A错误”与“模型A正确但模型B错误”的比例达到了惊人的11比1，这个结果实在是太显著了，因此可以肯定地说B更好。但是对于第二种情况，两者的比例仅为5比3，这样的数据就不显著了，因此没有足够的置信度说B更好

（笔者注：实际上，McNemar检验的计算逻辑要稍微复杂一些。假设模型A正确但模型B错误的测试数据数量为b，模型B正确但模型A错误的测试数据数量为c，McNemar检验先计算的是$(b-c)^2/(b+c)$，然后把这个值看作是自由度为1的χ^2分布的值，通过这个值可以看p值。对于上图第二种情况，对应的p值大约为0.1x，不能说有显著性）

用于物体识别的卷积神经网络

尽管卷积神经网络在MNIST手写数字识别上已经取得了很好的效果，但是要想用在实际任务，从一张包含了大量杂乱物体的彩色图片中识别出来目标物体，还是比较难的。这些难点在于

- 要识别的物体有很多种。即便只需要识别一千个类别，目标类别数量也是原来的100倍
- 像素更高。原来是28 x 28的黑白图片，现在要识别的至少是256 x 256的彩色图片
- 真实场景下需要处理的图片是三维实体的二维影像，所以会丢失很多信息

- 手写数字中虽然有笔画交叠（因此需要算法把数字分割开），但是不会出现物体遮挡，在同一个场景里也不会出现多个不同种类的物体，也不会出现光照变化

类似于MNIST，有一个称为ImageNet的数据库可以用来检测物体识别算法的准确程度。图库中有120万张高分辨率的训练图片，他们被人类手工标注成了1000个类别。但是，手工标注的结果并不总是十分可靠，有的图片里有两个不同类别的物体，但是标签只标注了其中一个。因此，ImageNet分类竞赛允许系统对给定图片给出5条猜测，如果人类给出的标记落在这5条猜测中，就算是正确。除此之外，ImageNet竞赛还有一个“定位任务”，因为现有算法都是使用特征“袋”方法（类比于NLP领域中的词袋模型），因此尽管算法对给定图片知道它符合什么特征，也知道里面包含什么物体，但是并不知道目标究竟在哪里。因此，定位任务要求算法不仅能找到目标物体，还要将其框出来。如果给出的框与正确的框的交叠部分能达到50%，就算合格

目前（2012年），计算机视觉系统使用的多是复杂的多阶段系统，这些系统的初始阶段一般是使用是某些数据手动调优少许参数，末尾阶段一般是某种机器学习算法。但是整个系统并不是一个“连贯的”，端到端的学习系统，不能像反向传播那样，让早期特征检测器所使用的参数根据其在最后判断类别时的表现进行自动调整

ImageNet竞赛的一个里程碑式的CNN体系结构是Alex Krizhevsky（Алекс Крижевский）在NIPS2012上提出的网络结构（现在称为AlexNet）。它包括7个隐藏层（不算最大池化层），其中前面几层都是卷积层，这样可以降低计算量。最后两层才是全连接层，这里包含了网络的大部分参数（层与层之间大概有1600万参数），用来寻找前面卷积层提取的局部特征的最优组合

AlexNet在每个隐藏层中使用的激活函数都是ReLU，它们比logistic激活函数训练起来更快，而且表示性更强。此外，它还使用了一种称为“竞争归一化”（competitive normalization）的方法：对某个区域，如果其邻接区域的神经元非常活跃，那么就抑制该区域神经元的活跃程度

AlexNet使用了如下两种方法来提升模型的泛化能力，避免过拟合

- 训练时，在256 x 256的图像上随机选取一个224 x 224的子图，这样可以得到大量数据。另外，对每张图像，还做了一个左右镜像，这种左右的映射不会实际改变物体的外观（除了图像里面是手写笔迹这种情况）。但是，Alex没有使用上下镜像，因为重力是一个很关键的因素。在测试时，它取测试图像四角上224x224的图像和中间同样大小的图像，然后对这五张图像都做一个左右翻转，用总共这十张图片得到是个测试结果，将这些结果组合得到最后的结果
- 使用了一种称为dropout的新型正则化方法。这种方法的细节将在后面的章节中介绍，其基本思路是每使用一条训练数据时在每一层随机丢弃一般隐藏层神经元，以避免这些神经元之间过强的合作。合作可以很好地拟合训练数据，但是如果测试数据的分布显著不同，那么这些合作就会导致过拟合

此外，AlexNet的成功还得益于其使用GPU进行计算，因为GPU能够高效完成矩阵乘法，到存储器的带宽也很高。随着GPU和CPU技术的发展，以及数据集规模的疯狂增长，深度网络将成为物体识别的不二之选（嗯，现在看来已经成为了现实）

除去ImageNet，还有一些其它应用领域也值得关注。例如，Vlad Mnih使用局部感知野（local field）和多层ReLU单元来从卫星图像上抽取道路，不过他没有用卷积操作

Hinton神经网络与机器学习 6. 神经网络的优化

- **本文作者：** Tingxun Shi

- **本文链接：** http://txshi-mt.com/2017/12/27/UTNN-6-Optimization-How-to-Make-the-Learning-Go-Faster/

Mini-batch梯度下降概论

本章主要介绍一些关于神经网络优化的方法。但是在介绍优化方法之前先回顾一下关于误差函数/代价函数的内容。如第三章所示，给定某个线性神经元，可以画出其误差表面：其纵轴是误差，横轴是权重值。如果使用的是平方误差，则误差表面的横切面应该是一个椭圆，纵切面是一个双曲线。对多层非线性网络来说，其误差表面更加复杂，但是权重不大时，仍然是光滑的，局部也可以用二次曲线逼近。如果使用full-batch梯度下降（即梯度下降时使用所有样本），单步来看，应该沿着向下滑的方向才能最大地缩减误差。但是问题是，这个下滑方向并不指向我们想要让代价函数减小的方向。当椭圆特别长时，实际上，梯度下降的方向几乎正交于理想方向。这个时候就比较尴尬了：算法在我们不想走的方向上迈出了一大步，却在想要的方向上迈出了很微小的一步。尽管上面的这个例子是在线性系统中所展示的，但是在非线性的多层网络里，情况不会有什么变化。所以，当使用的学习率很大时，学习就会出现问题，误差会沿着很弯曲的表面来回碰撞。当学习率太大的时候，实际上算法就不收敛了。因此，我们的目标是，沿着梯度小而稳定的方向快速下降，沿着梯度大而不稳定的方向慢速下降

在讲述如何达到这一目标之前，再看一点关于随机梯度下降（Stochastic Gradient Descent, SGD）的原理和动机。如果你有一个高度冗余的数据集，那么在该数据集前一半计算出来的梯度与用该数据集后一半计算出来的梯度应该是相同的，所以用整个数据集计算梯度就是一种对时间的浪费，不如用前一半数据计算出来梯度，更新权重，然后用另一半数据计算新的梯度。最极端的情况，可以对每条训练数据计算一次梯度并更新权重，这种做法称为**在线学习**。当然，通常也不使用如此极端的方法，而是每次使用比较小量的一批数据，通常是10个、100个甚至100个样本。这样做有两个好处

- 更新权重时，计算量更少
- 可以并行计算多条数据的梯度。如果使用GPU计算，效率会更高（GPU善于计算类似矩阵乘法这样的操作）

做mini-batch SGD时，有一点需要注意，就是尽量不要让一个mini-batch中的数据只属于一个类别。推而广之，假设要解决的是一个多分类问题，要尽量让一个mini-batch中的数据类别分布均匀

因此，学习问题基本上可以分为两类

- 全梯度下降（full-batch GD），使用所有训练数据计算梯度。这种做法在优化问题里研究得比较透彻，因此也有一些加速学习的方法，例如共轭梯度法（conjugate gradient）对非线性问题就很有效。不过，多层神经网络与传统优化问题研究的对象有所区别，因此使用传统方法时一般要做比较大的修改
- mini-batch学习，适用于训练集冗余且比较大的情况

最后，来看一个基本的mini-batch梯度下降求解线性问题的做法。首先，猜测一个初始的学习率。如果误差变大或者剧烈摇摆，就降低它；如果误差降低太慢，但是仍然在持续下降，就提高它。通常可以写一段代码来使学习率自适应学习过程，但是有一个实际经验，就是学习即将结束的时候降低学习率总是没错的，因为使用mini-batch SGD会造成梯度的波动，进而使得权重发生波动。而训练要结束的时候总是希望结果能平稳一些，因此降低学习率有助于得到一个平稳的权重。降低学习率的另一个好的时机是当误差不再平稳下降时，而对误差进行观测的一个有效方法是观察学习过程在训练集上的效果

关于mini-batch梯度下降的一些技巧

权重的初始化

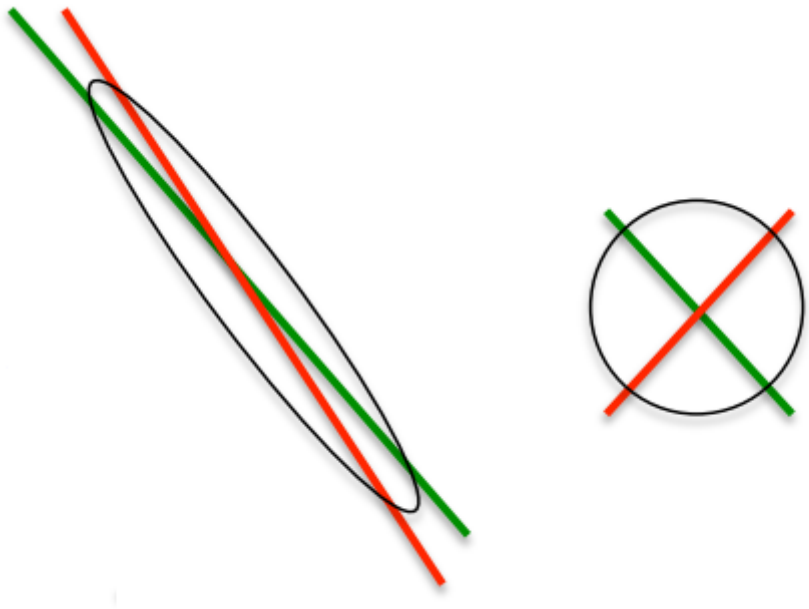
如果两个神经元的权重相同，偏置相同，那么给定相同的输入，这两个神经元不会有不同的。因此一般是用小的随机值来初始化权重，打破这种“对称性”。所以如果神经网络的隐藏层单元扇入数（fan in，可以理解为它可以从多少个上游单元接收输入）特别大，用大的权重会导致饱和，这时就应该使用更小的权重——更好的原则是，让初始权重的大小正比于扇入数的平方根

输入平移

将输入做平移（shift）会对神经网络的学习速率造成令人意想不到的重要影响。所谓输入平移，就是在输入值的每个维度上加上一个常数。最好的平移方法是在所有训练数据上平移一个均值，这样使得输入的均值为0。来看一个例子：假设有如下两组数据，输入输出分别为

$$\begin{aligned} \boldsymbol{x}_1 &= [101, 101] \qquad y_1 = 2 \\ \boldsymbol{x}_2 &= [101, 99] \qquad y_2 = 0 \end{aligned}$$

假设第一个例子的代表色为绿色，第二个例子的代表色为红色，下图左给出了这种情况下的误差表面。可以大致理解为，当只有第一个例子时，权重$\boldsymbol{w}$需要满足$\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_1 = 2$，因此$\boldsymbol{w}$的两个分量之间呈线性关系，线的斜率是$-101/101 = -1$。而只有第二个例子时，权重$\boldsymbol{w}$需要满足$\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_2 = 0$，因此$\boldsymbol{w}$的两个分量之间呈线性关系，线的斜率是$-101/99$。这两条线基本上是平行的，因此当将两者组合起来时，会得到一个被拉长的椭圆



移动输入以后对误差表面的影响。左图是移动前的误差表面，非常长的椭圆梯度下降会非常缓慢；而右图是移动后的结果，正圆的误差表面上，梯度下降的方向总是指向圆心

如果对$\boldsymbol{x}_i$的每个维度都减去100，那么就会得到一个截然不同的误差表面，如上图右所示，它是一个正圆形。这个时候，做梯度下降就容易多了

如果考虑的不是输入，而是隐藏层的计算，那么类似地，让隐藏单元使用双曲正切函数$\tanh$会是一个比较明智的做法，因为这样可以保证激活值是一个在-1和1之间的数，且均值大致为0，这会使下一层的学习变快一点。计算上，$\tanh$也不慢，而且其与之前熟知的sigmoid函数σ存在如下的一个关系

$$\tanh(x) = 2\sigma(x) - 1$$

因此，总体来看，$\tanh$比sigmoid函数会更好用一点

（后面Hinton还提到了sigmoid函数的一点优势，但是我没看得十分明白。其大概意思是说，如果输入是比较小的负值，那么sigmoid的结果会是0。如果输入更小，则sigmoid的输出还是0，这样可以过滤掉一些比较极端的输入）

输入缩放

另一种比较有效果的做法是对输入进行缩放（scale）。前面提到过，输入平移的最好效果是让输入样本的均值为0，那么缩放的最好效果是让输入样本的方差为1。假设输入样本的第一个维度范围非常小（假设取值范围是$[-0.1, 0.1]$），第二个维度范围非常大（例如$[-10, 10]$），那么误差表面沿第一个维度的曲率就会特别大，

因为一点细小的改变就会让输出变化很大；而误差表面沿第二个维度的曲率会特别小，因为同样微小的改变几乎不会影响输出。重新缩放以后，就不会有这样的问题

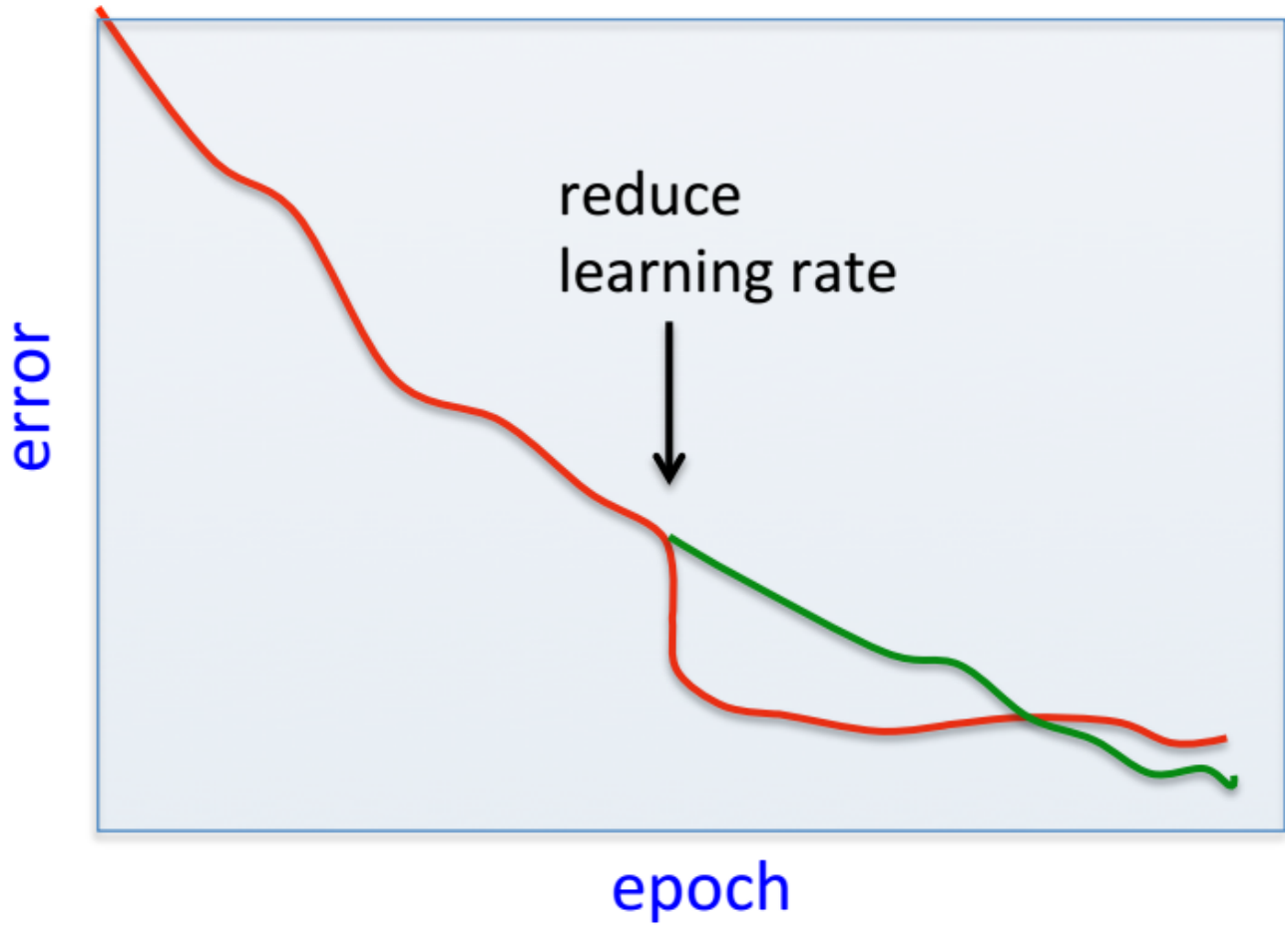
去相关性

无论是对输入数据做平移还是缩放，这种做法都是简单易于实现的。接下来来看一种更复杂的方法，但是这种方法效果更好，因为它能保证给出一个圆形的误差表面（至少对线性神经元而言）。这种做法是对输入向量的各个维度**去相关性**（decorrelate）。去相关性的具体实现有很多种，比较常见的一种是先对输入数据做主成分分析（PCA），把特征值最小的几个维度去掉（这样做也可以达到维度缩减的目的），然后对剩下的维度除以各自对应特征值的平方根。这样，得到的误差表面肯定是正圆形，梯度的方向指向最小值

其它常见问题

- 关于学习率：如果给的初始学习率特别特别大，那么会导致隐藏层状态要么是恒为开的（一直被激活），要么是恒为关的（一直不被激活）。因为这会导致与该隐藏单元关联的权重或是特别大的正数，或是特别小的负数，因此其状态不再依赖于输入，也不会受到输出的误差带来的影响——单元一直处于导数基本为0的“高原”状态，学习自然会停止 在之前，人们对神经网络的担忧总是认为，神经网络最后会陷在一个局部最小值上。因此，他们会把神经网络停止学习而误差很高的情况归结于神经网络会在一个很差的局部最小值上收敛。这是不对的，实际上，这更可能意味着学习陷入到了一个高原区，走不出去
- 使用神经网络解决分类问题时，一般都是用平方误差或者交叉熵误差，因此最好的预测策略通常是让输出单元的输出正比于某个“出现次数”。按照社区助教回复举的例子，如果现在要使用神经网络去学习一些训练语料，然后在给定某些字母的前提下，让它预测下一个字母最有可能是什么。由于神经网络的权重是随机初始化的，因此最开始得到的隐藏层特征对输出完全没有帮助，此时最好的输出策略是让每个字母的输出概率正比于这个字母在训练语料中出现的概率，也就是完全没用上输入信息。神经网络会很快找到这个策略，因此前面的误差会下降得非常快。要改进这个策略，网络需要通过隐藏层从输入层得到足够有用的、可感知到的信息，而如果初始化的权重比较小，这个过程可能是比较长的。所以在实践中可能观察到的现象是，学习进展很快，一会儿误差就不再下降了，看上去达到了某个局部最优点。但是实际上，到达的是另一个平原而已 所以，要辩证地看待课程前面提到的“学习临近结束时要降低学习率”这条经验，也要小心不要过早降低学习率。降低学习率会减少由不同mini-batch的不同梯度带来的误差的随机波动，所以看上去学习过程可以很快收敛，但是实际上会减慢之后的学习过程。下图给出了一个示例情况，其中红线

是提前降低学习率的学习曲线，可以看到其最后效果要差于正常调整学习率的情况（绿线）



加快学习的四种方法

本节前面所讲的是如何让学习的效果更好，这里提供四种方法，它们都是为了让模型学习更快而设计的

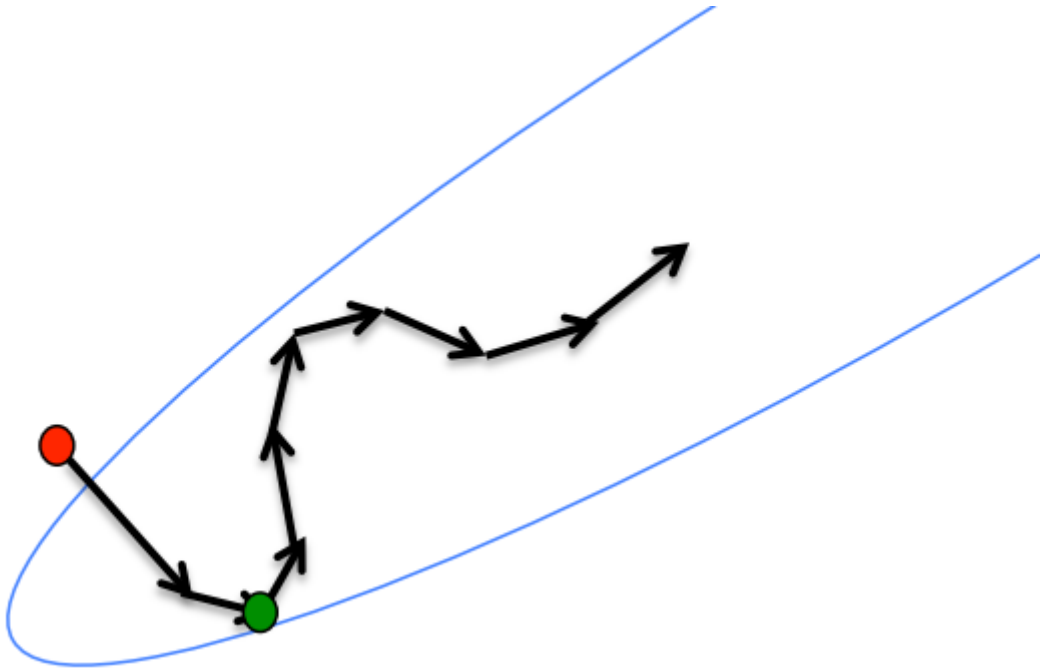
1. 动量法，它不使用梯度来修改权重“位置”，也就是说，如果把权重想象为在误差表面上的一个球，标准梯度下降是修改球的位置，修改量是梯度与学习率的乘积。动量法使用梯度来加速球的前进，也就是说，其使用梯度修改球的速度，由速度去修改球的位置。尽管看上去效果相同，但是其原理是有本质不同的，因为球可以有“动量”（momentum），即它可以记住上一个梯度
2. 为每个参数设置一个单独的，自适应学习率，然后根据经验测量来慢慢调整学习率。如果梯度的符号一直在变，就减小学习率；否则就增大
3. rmsprop，将学习率除以最近梯度的滑动范数平均值。可以看作是上面方法的mini-batch版本
4. 借鉴full-batch学习中的方法，利用曲率信息，并修改算法使其适用于神经网络和mini-batch学习（本课不讲授）

动量法

在使用梯度下降训练神经网络时，可以使用动量法（momentum）来有效提升学习速度。这一方法适用于full-batch学习，也适用于mini-batch的学习。实际上，将mini-batch梯度下降与动量法结合可能是大规模神经网络最常见的学习方法

假设在误差表面上放置一个球，球在水平面的位置表示当前的权重向量。由于其最开始是静止的，因此它会先沿着最陡的方向（梯度方向）向下滑，但是当速度起来以后，动量会驱使它沿着之前的方向前进。显然，我们想让球最后停留在误差表面的一个最低点上，所以希望能减少球的能量，因此能引入一些粘性是最好不过的——也就是说，要在每次迭代中都缓慢降低求得速度

动量法的作用就是在高曲率的地方阻止小球的振动，可以看一下下图的例子



动量法示意图

小球从红点出发，经过两步以后会达到绿点位置。此时，两点处的梯度范数基本相同，但是方向相反。因此纵向于“峡谷”的梯度已经被抵消掉了，但是沿着峡谷方向的梯度还在，因此使用梯度法，可以保留该方向的速度，继续前进，最后达到最小值点

动量法的数学公式比较简单

$$\boldsymbol{v} = \alpha \boldsymbol{v}(t-1) - \epsilon \frac{\partial E}{\partial \boldsymbol{w}}(t)$$

前一项是对上一个mini-batch的速度向量做些微衰减得到的结果，这里α一般是一个小于1的数，例如0.9，起到了加入粘性的作用。后一项是当前梯度造成的影响，使得我们可以一定程度地沿梯度下降。接着，t时刻权重的改变量就是前面求出来的速度向量，也可以表示为一个有关于$t-1$时刻权重改变量的式子，即

$$\begin{aligned}\Delta \boldsymbol{w}(t) &= \boldsymbol{v}(t) \\ &= \alpha \boldsymbol{v}(t-1) - \epsilon \frac{\partial E}{\partial \boldsymbol{w}}(t) \\ &= \alpha \Delta \boldsymbol{w}(t-1) - \epsilon \frac{\partial E}{\partial \boldsymbol{w}}(t)\end{aligned}$$

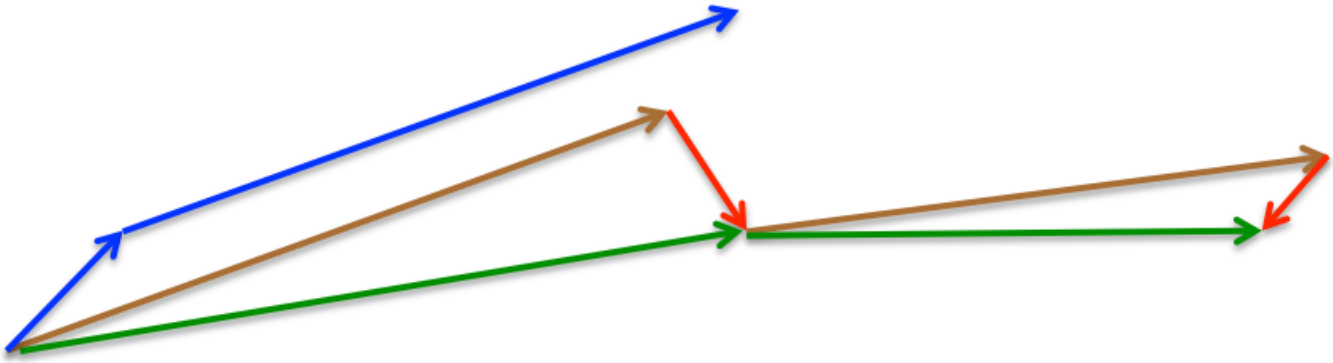
动量法的行为很直观。在一个误差表面上，球会不断从梯度获得前进的速度，这个速度值与动量项（实际上是粘性）所减小的速度值相平衡。由于速度不断累积，因此最后小球会达到一个最终速度。如果动量项接近于1，小球的前进速度会远快于普通梯度下降法所给的速度，其最终速度的计算公式为

$$\boldsymbol{v}(\infty) = \frac{1}{1-\alpha}\left(-\epsilon \frac{\partial E}{\partial \boldsymbol{w}}\right)$$

因此，如果$\alpha = 0.99$，动量法的下降速度比普通梯度下降法要快100倍

在学习的开始阶段，如果初始的随机权重太大，那么梯度就会很大，这时不宜使用大的动量，设成0.5比较好。当大的梯度消失时，说明学习进入到了一个正常的阶段，可能正在某个长峡谷里摸索，此时可以平滑地提高动量值。相比于直接使用大学习率的方法，使用小学习率+大动量可以达到一个实际上更大的学习速率，因为前者会导致权重纵向于峡谷方向剧烈振动

标准的动量法是先计算当前位置的梯度，然后算出当前梯度与之前梯度的组合，把这个组合当做前进方向迈出很大一步，即前进方向总是累加的梯度方向。最近（2012年），Ilya Sutskever发现了一种更好的动量法，这种方法借鉴了Nesterov在优化凸函数时的方法，即先沿着之前累加梯度的方向跳出一大步，然后测量落脚点的梯度，进行修正。两种方法比较相似，其不同点在于，标准方法是先加上当前的梯度，然后跳出去赌一把；在Nesterov方法里，是先跳出去，然后再校正。下图给出了这两种方法的图示



Nesterov法与标准动量法的比较

图中

- 蓝色的向量为标准动量法的前进方向
- 棕色的向量是Nesterov法跳跃（前进一大步）的方向
- 红色的向量是校正方向
- 绿色的向量是累加梯度的方向

上图中这些向量有一些几何意义需要点明一下。首先，所有绿色向量都是棕色向量与红色向量的加和。其次，第二个棕色向量的方向与第一个绿色向量的方向一致，长度是绿色向量的长度乘以动量项（粘滞系数α）。最后，长的蓝色向量与第一个棕色向量平行

为每个连接设置自适应学习率

本节介绍的方法是20世纪80年代晚期Robbie Jacobs提出来的，后来经过了很多其他研究人员的改进。其核心思想是，神经网络中的每个连接都应该有其自己的自适应学习率。在更新每个连接的权重时，需要观察更新的结果，然后根据这些结果经验性地设置对应的学习率。所以如果某个权重梯度的符号总是变来变去，就降低学习率，反而就增加

为什么这种做法是一种不错的做法呢？先看一下问题。由于神经网络比较深，连接数比较多，因此不同神经元之间连接对应的权重变化会很大，尤其是不同层之间的权重差别会更大。假设初始化的时候使用了小的权重，那么在前面几层的梯度会比后面几层的要小。另一个重要的因素是隐藏单元的扇入数有差别。当同时改变多个不同的输入权重来修正某个相同的错误时，扇入数决定了“超调”（overshoot）量：可能某个单元之前没有得到足够的输入，但是权重一次更新以后，反而它接受了太多输入——显然，如果扇入数大，这种状况的影响就更大

因此，可以先手动设置一个全局的学习率，然后对每个权重，乘以一个局部的收益（gain）。具体做法是，首先，对每个权重w_{ij}，初始化的收益g_{ij}都为1，因此最开始每个权重的改变量为

$$\Delta w_{ij} = -\epsilon g_{ij} \frac{\partial E}{\partial w_{ij}}$$

接下来就是如何自适应地调整g_{ij}。如果权重所使用的梯度没有改变符号，就使用“加性增、乘性减”的原则增加g_{ij}，否则减小g_{ij}。具体的更新方式为

$$
\begin{aligned}
&\text{if} && \left(\frac{\partial E}{\partial w_{ij}}(t)\cdot\frac{\partial E}{\partial w_{ij}}(t-1)\right)>0\\
&\text{then} && g_{ij}=g_{ij}(t-1)+0.05\\
&\text{else} && g_{ij}=g_{ij}(t-1)\cdot 0.95
\end{aligned}
$$

这样，如果梯度发生了波动，大的收益值就会快速下降

一个有趣的问题是，如果梯度都是随机的会怎么样。这会导致g_{ij}增减的概率相等，因为前一时刻梯度与当前时刻梯度有一半的概率是同号的，一半概率是异号的。由于g_{ij}的初始值为1，增加0.05和缩减为原来的95%概率相等，因此最后它会在1附近震荡

有许多种让自适应学习率表现更好的方法

- 可以把收益值限定在某个范围内，例如[0.1, 10]或者[0.01, 100]等。收益值不应该太大，因为这样权重会进入一个不稳定状态，收益值也不会很快降下来，所有权重就都被搞崩了
- 使用full-batch学习，或者做mini-batch学习时一次使用更多的样本。因为自适应学习率本质上是为full-batch学习设计的，这样可以保证梯度符号的改变并不是因为mini batch的采样出现问题，而是真的因为梯度跑到了峡谷的另一端
- 将自适应学习率与动量法相结合。更进阶的方法是，看当前权重梯度的符号与当前权重速度的符号是否相同。需要注意的是，只有误差表面近似为正圆时，自适应学习率才能取得比较显著的效果，但是动量法没有这个问题。即便是误差表面是狭长的椭圆形状，动量法也能推动梯度快速向最优点移动

Rmsprop

本节主要介绍rmsprop这种优化方法。不过在此之前，先看一下这个算法的基础——rprop算法。Rprop算法的理论基础是不同权重的梯度大小会非常不同，而且它们会在学习的过程中变化，因此很难选择一个唯一的，全局的学习率。如果做的是full batch学习，那么仅仅使用梯度的符号就可以处理这种梯度的多样性，这样，所有权重的增量是一致的。这种做法有助于让小梯度逃离平原，因为即便梯度很微小，通过这种方法前进的步子也可以很大。需要注意的是，调大学习率不能达到同样的效果，因为这样会使梯度大的权重变得太大

Rprop将使用梯度符号的思想和为每个权重单独确定步长的思想结合了起来，也就是，当决定要对权重做多大改变时，不再看梯度的大小，而只看梯度的符号。但是，步长是随着时间的变化而自适应调整的。一般来讲，如果过去两次梯度的符号相同，则步长变为原来的1.2倍；否则，步长以一个更大的系数衰减（例如变成原来的一半）。此外，通常还要对步长的大小做出限制，Mike Shuster的建议是把它限定在$[10^{-6},50]$之间，不过还是应该具体问题具体分析。例如，如果输入太小，那可能需要比较大的权重才能起效果。不过如果处理的不是这个问题，用50作为上限可能有点大，应该降得更多一些

那么rprop为什么不适用于mini-batch的学习呢？因为它的思想背离了SGD的核心思想。SGD的核心思想是，当学习率比较小的时候，梯度是过去连续几个mini-batch的梯度的均值。假设某个权重在9个mini batch中的梯度是0.01， 第十个mini batch的梯度是-0.09，那么权重最好不动。但是rprop不能保证这样的性质，因为它只关注连续两次梯度的符号是否相同，因此会连增8次，然后再狠狠下降一次，最后导致权重变得特别大。所以问题来了，有没有可能把rprop的鲁棒性、从多个batch获取信息的高效性，和SGD正确组合mini batch梯度的洞见性组合起来呢?

这种方法就是rmsprop法，可以看作是rprop的mini-batch版本，它会把梯度再除以梯度的大小。因为mini batch rprop的问题是总是为每个mini batch的梯度除以一个不同的值，所以为了解决这个问题，可以让临近几个mini batch梯度的除数大致相等。要达到这个目的，可以为每个权重维护一个梯度平方的滑动均值，即

$$
\text{MeanSquare}(w,t)=\alpha\cdot\text{MeanSquare}(w,t-1)+\beta\cdot\left(\frac{\partial E}{\partial w}(t)\right)^2
$$

（在讲义中，设置$\alpha = 0.9, \beta = 0.1$。不过Hinton说明这只是示意值。但是两者加和是否必须为1课程中并没有提）。得到这个值以后，可以把梯度除以$\sqrt{\text{MeanSquare}(w,t)}$，这也是“rms”的来源（rooted mean square）。注意这里没有为每个连接都各自单独设定一个自适应的学习速率，而是对每个连接都采用相同的策略

在rmsprop的基础上，可以做一些进一步的扩展，例如：

- 将其与标准的动量法相结合。不过Hinton组当年做的实验发现引入动量法带来的改进比较有限，需要进一步观察
- 将其与Nesterov动量法相结合。Ilya Sutskever的实验发现这样做效果不错，尤其是用最近几次梯度的rms去除校正项（而不是跳跃项）时
- 将其与分连接自适应学习率法相结合，这样的算法更像rprop，不过需要进一步的研究
- 其它方法，可以参看Yann LeCun组*No More Pesky Learning Rates* (ICML 2013, pp. 343 - 351)

神经网络学习方法小结

所以，总体来看，如果数据集比较小，例如只有不到10000条数据，或者是不太冗余的更大一些的数据集，可以考虑使用full batch的方法。对于这类问题，一些从优化理论领域扩展来的方法，例如非线性共轭梯度法、LBFGS法都比较适用，而且这些方法都有比较成熟的实现。或者也可以使用自适应学习率法或rprop，它们算是专为神经网络所设计

如果数据集有很多冗余，那么一定要使用mini batch法，否则会造成极大浪费。在这种情况下，首先试用带有动量的SGD，但是不要为每个权重自适应学习率。接下来可以尝试rmsprop，或者其改进版本。最后，可以尝试Yann LeCun的最新方法，他是优化狂魔，所以他们组的工作是比较值得跟进的

需要注意的是，神经网络没有一种普适通用的学习方法，原因有两点。其一，**神经网络的结构千差万别。**那些特别深的网络，尤其是体系结构中存在狭窄瓶颈的网络，非常难以优化，它们对小的梯度非常敏感。循环神经网络（RNN）又是另一种情况，它们需要注意过去一段时间之前发生的事情，同时基于此来修改权重，所以不需要特别准确的优化，因为需要在过拟合之前停止优化过程。其次，**神经网络处理的任务千差万别。**一些任务需要非常精确的权重，但是有些就不需要。另外，有些任务可能需要处理罕见情况（例如处理罕见词），但是对于其他任务就没有这个需求。因此，如何训练神经网络并不存在定数，优化方法有很多，不可能符合每个人的口味，但是总有一种是比较合适的

Hinton神经网络与机器学习 7. 循环神经网络 I

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2018/01/07/UTNN-7-Recurrent-Neural-Networks/

对序列建模

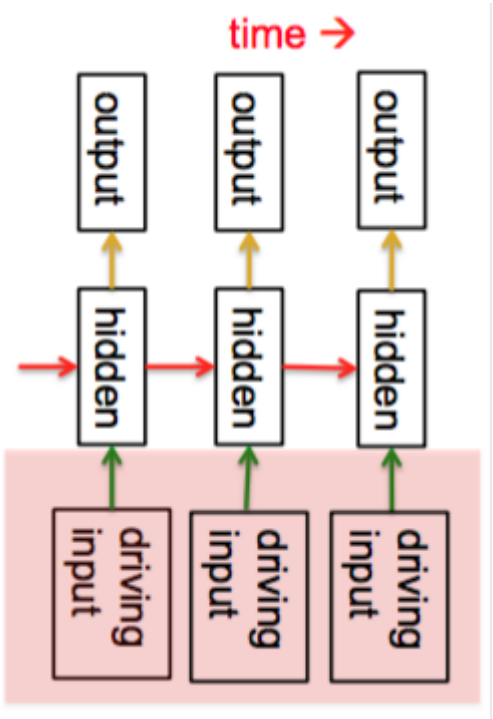
当使用机器学习方法对序列建模时，通常是要把一个序列转化为另一个序列。例如，想把英语单词序列转化为法语单词序列（机器翻译），或者是将声音信号转化为单词的文本序列（语音识别）。有的时候，并没有一个单独的目标序列，在这种情况下，可以通过试着预测输入序列的下一个项（item）来获取一个教学信号，因此此时目标输出序列就是一步之后的输入序列——对于时间序列来说，这种预测是很正常的。不过对于图像则不然，预测下一个像素点是什么感觉应该是没有什么意义

当预测序列中的下一个项时，实际上有监督学习和无监督学习的界限已经变得模糊了：预测下一项内容时，使用的方法是有监督学习方法，但是并不需要独立的教学信号，因此这里又存在一些无监督的味道

在介绍RNN之前，先看一些其它对序列建模的模型。其中一种不需要额外内存的方法叫**自回归模型**，其原理是使用前面一些项的值做一些加权平均，来预测接下来的这一项。对这种算法稍加修改，加入隐藏单元，可以得到一个前馈神经网络，例如Bengio的第一个语言模型就使用了这样的原理

其他一些模型可以让其隐藏状态有一些内部的动态变化，这样这些隐藏状态可以根据自身内部的动态性演化，并产生观察结果。通过该机制，模型可以在隐藏状态里存储信息，并保留比较长的一段时间。如果隐藏状态的动态性是带有噪声的，而且它通过隐藏状态生成输出的过程也是带有噪声的，那么通过观察这种生成模型的输出是不可能推断其具体的隐藏状态的，只能推测在所有可能隐藏状态向量的概率分布，而这个过程通常比较难。不过对某两种特殊的隐藏状态模型，它们的隐藏状态分布是可计算的

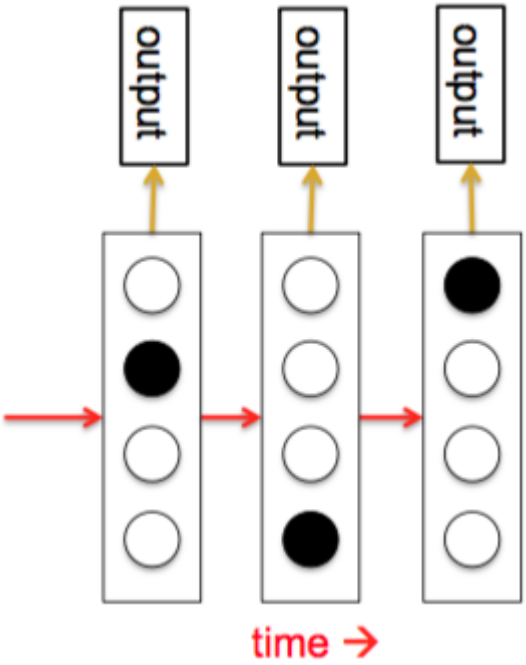
第一种模型是**线性动态系统**（linear dynamical systems），在工程界得到了广泛应用。下图给出了该模型的示意图



线性动态系统示意图

这种生成模型的隐藏状态都是实数，有线性动态性（由红色箭头表示），这种动态性带有高斯噪声，所以隐藏状态会按照一定概率演化。模型的底部是驱动输入，可以直接影响隐藏单元，而隐藏单元会决定输出。因此为了预测系统的下一个输出，就需要能推断隐藏状态。这里隐藏状态可以被推断的关键原因是因为假设了动态性是高斯噪声，满足高斯分布，而高斯分布的一个特点就是对高斯分布做线性变换的结果还是一个高斯分布，因此给定观察结果（输出）时隐藏状态的分布也是高斯分布（实际上，是一个完全协方差高斯分布，即协方差矩阵元素都不为0）。对于这类问题，可以用**卡尔曼滤波**（Kalman filtering）来解决

第二种模型是**隐马尔可夫模型**（hidden Markov model, HMM），它使用的是离散分布，基于离散数学，因此广受计算机科学家喜爱。下图给出了HMM的示意图



HMM示意图

在HMM中，隐藏状态是N个可能值中的一个，状态之间的转移符合某种概率分布，可以组成一个概率矩阵M，称为转移矩阵。例如，从状态1转移到状态2的概率是M_{12}。而且，输出模型也是随机的，因此系统所在的状态不能完全决定其所产生的输出，也就是说，内部状态“隐藏”在这个概率矩阵的背后。为了预测HMM的下一个输出，需要推测其可能位于哪个隐藏状态，对此可以使用一种动态规划算法，通过观测值可以推出隐藏状态的概率分布矩阵。得到概率分布矩阵以后，就可以得到HMM算法。这种算法在2012年左右是语音识别算法的首选（也是输入法引擎算法的首选）

HMM有一个非常基本的局限性：它每次产生数据的时候，都会选择从N个隐藏状态中选择一个。所以每个隐藏状态中储存的时间信息最多有$\log N$位。假设现在要使用HMM生成一句话，已经生成了前一半，要生成后一半，则它对前一半的记忆是通过其在N个隐藏状态中的哪一个来体现，所以只有信息中的$\log N$位。为了让句子的后一半与前一半相协调，则后面的句子的语法要与前面匹配（性、数、格一致），语义要与前面匹配，语调、口音、音色等等都需要匹配。所以如果需要使用HMM来生成一个句子，那么隐藏状态需要覆盖所有这些信息。假设这些信息一共有100位，那么HMM就需要2^{100}个状态！

因此就需要循环神经网络（RNN）来解决这个问题了，它可以更有效地“记住”信息。其有两个特点：

- 分布式隐藏状态。也就是，同一时刻可以激活多个不同的单元，因此它可以同时记住几件事情
- 非线性。隐藏状态不再需要以线性方式演化，因此系统可以更加复杂。在有足够神经元和时间的情况下，它几乎可以计算任何事情

除此以外，线性动态系统和HMM都是随机模型，有一些噪声。但是模型真的需要这样吗？实际上，这两个随机模型在隐藏状态上的后验分布都是确定函数，概率分布都是一些确定的数字。而在RNN里，可以认为这些数字构成了RNN的隐藏状态，就像这些简单随机模型中的概率分布一样

RNN的行为可以展现出如下特点：

- 震荡性，因此RNN可以适用于动作控制（motion control）。例如可以求出走路时的步幅
- 可以停留在某个稳定点（point attractor）上，因此可以适用于从某些“记忆”中获取信息（Hinton没有展开这一块，所以不是很明白。论坛里助教也并不理解这个概念）。可以理解为，你有一些大致的想法，大概知道想要获取什么，然后系统就可以停留在某个稳定点上，这些稳定点就对应于你想要获取的信息
- 混沌性（chaotically）。通常混沌性对信息处理来说并不是一件好事，但是在某些环境下，如果你正在与一些更聪明的对手作对，无法智取对方，那么随机出招可能反倒是个好事。达到这种随机性的方法就是让

行为体现出混沌性（这又是什么鬼……）

RNN的关键问题是，计算能力太强，因此难以训练

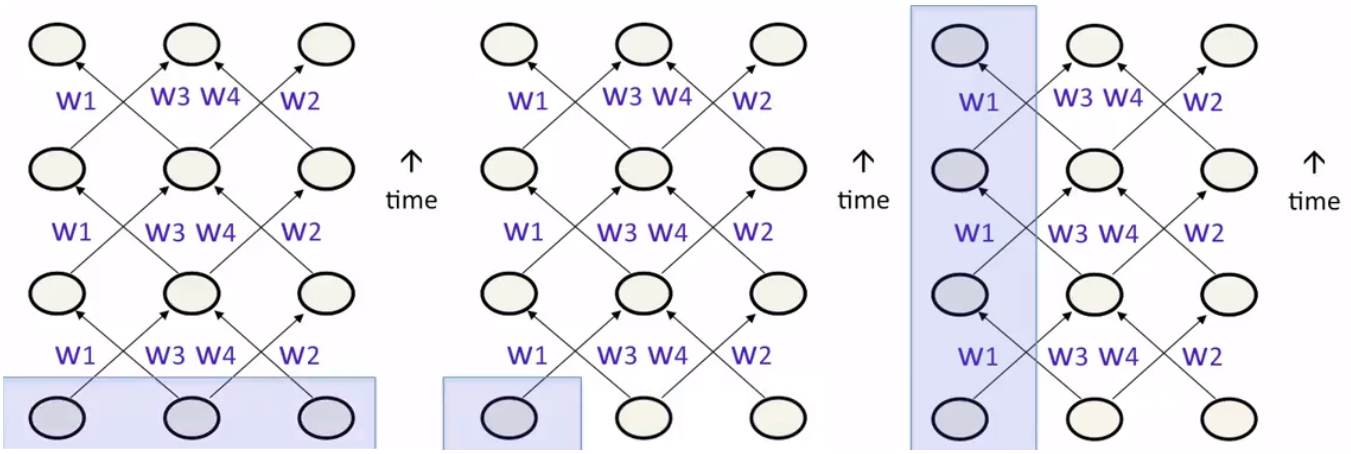
使用反向传播算法训练RNN

假设现在有一个简单的RNN，由三个相互连接的神经元构成。连接之间有一个时间单位的延迟，网络的时钟每次走一个时间单位。要理解如何训练一个RNN，关键问题是要看到RNN的本质还是前馈神经网络，只不过是要将网络按照时间展开

在第0时刻，RNN有一个初始状态，通过同样的权重一点一点得到接下来各个时刻的状态。也就是说，RNN是一个每层权重都限制为相同权重的前向网络。在前面讲解CNN时，我们看到过反向传播如何在存在某些（线性）约束的情况下进行，即先像往常一样计算梯度，然后修改梯度来满足限制条件。对于RNN也是一样的，即先计算$\frac{\partial E}{\partial w_1}$和$\frac{\partial E}{\partial w_2}$，然后对$w_1$和$w_2$都使用$\frac{\partial E}{\partial w_1}+\frac{\partial E}{\partial w_2}$

这样，就可以得到RNN的反向传播算法，不过这一算法更常见地是被称作“随时间推移的反向传播算法”（BackPropagation Through Time，BPTT）。因此， 可以从时间的角度来思考这个算法：正向的一趟计算在每个时间步都对所有隐藏单元的行为做一次计算，然后构建起一个栈；反向的一趟计算在每个时间步计算错误的偏导数，然后一步步把这个栈拆解掉（这也是为什么这个算法被称为是“随时间推移的”的原因）。在反向传播以后，就可以把所有不同时间步对各个权重的偏导数加起来，通过一定的比例变换最终让每层权重相同

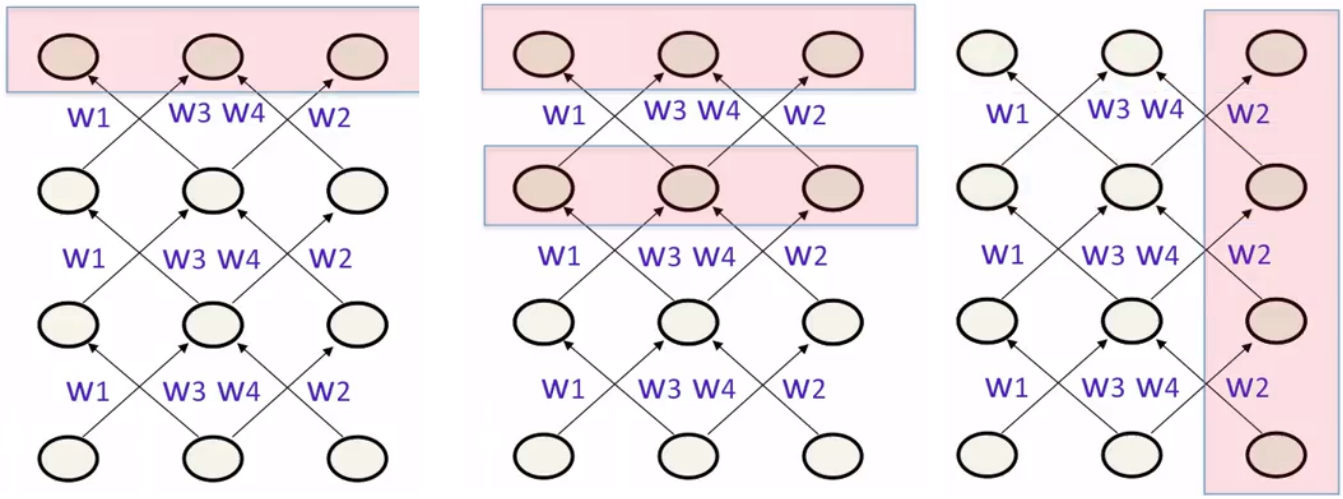
对于RNN，还有一个初始状态的问题。尽管可以将所有初始状态都设定成一个默认值，但是这样网络的效果不会特别好。事实上，也可以像学习权重那样学到网络中的状态，即先对所有非输入节点设定一个随机的初始值，然后每次反向传播也计算误差函数对状态的梯度，使用梯度下降来对状态进行修改。RNN中的初始状态有很多种初始化方法



RNN的状态初始化

- 上图左给出的方法是先指定所有单元的初始状态。如果把RNN看作权重受限的前向网络，这种做法比较自然
- 上图中给出的方法是指定某几个单元的初始状态
- 上图右给出的方法是对某几个单元指定它们在各个时间步的初始状态。如果要解决的问题是对顺序数据的建模，那么这种方法更自然

类似地，RNN的学习目标也有很多种选择方法

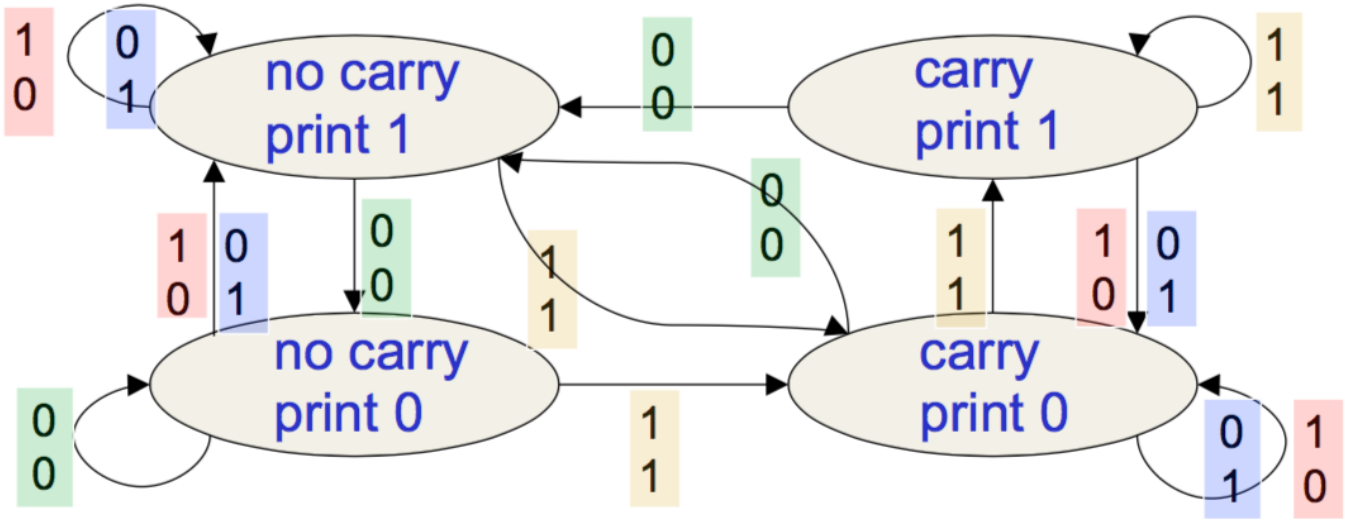


RNN的学习目标选择

- 上图左给出的方法是选择所有单元的最后状态。如果把RNN看作权重受限的前向网络，这种做法比较自然
- 上图中给出的方法是选择所有单元最后几个时间步的状态。这样做有助于学习到某些“稳定点”（attractor）
- 上图右给出的方法是指定某些单元为输出单元，输出这些单元在每个时间步的最后状态。如果要提供连续输出，这种方法比较自然

示例：训练一个简单的RNN

本节我们来考虑这样一个问题：如何训练一个神经网络，使得其输入为两个二进制数字，输出它们的和（也是二进制表示）。这个问题是可以用普通的前馈神经网络解决的，但是做起来比较麻烦：首先，需要预先确定加数、被加数以及和的最大位数。此外，更重要的是，对两个输入不同位上的做法是不能泛化的。确切说，对于两个输入，前面的处理逻辑和后面的处理逻辑使用不同的权重，因此两者的逻辑不能自动泛化。这意味着，使用前馈神经网络不能优雅地解决二进制加法问题



解决二进制加法的有限状态自动机

上图给出了一个解决二进制加法问题的有限状态自动机。系统在每个时刻都会处于图中的某个状态，然后接受下一列的两个数，根据输入决定状态的转移，进入到某个状态时则做相应的动作。假设现在自动机在右上角的状态，它会输出1，同时打开进位器。下一栏的两个数都是1，那么它会进入到相同的状态，输出1。但是如果下一栏的两个数一个是1一个是0，则它还会打开进位器，但是进入到右下角的状态，输出0；如果下一栏的两个数都是0，那么它关闭进位器，进入到左上角的状态，输出1。这意味着解决二进制加法的RNN需要有两个输入单元和一个输出单元，对应于在每个时刻接受两个输入位产生一个输出位

需要注意的是，网络的输出对应于两个时间步之前的输出。要延迟两个时间步的原因是，网络需要一个时间步来根据输入更新隐藏单元，另一个时间步来从隐藏状态产生输出。因此网络最少只需要三个隐藏单元，这三个单元是双向全连接的，每个方向的权重不同。这样的结构使得网络可以根据时刻t的隐藏状态推断出时刻$t+1$的隐藏状态。从输入单元到隐藏单元是普通的前馈模式，这样隐藏状态可以获取输入；类似地，从隐藏单元到输出单元也是普通的前馈模式，以产生最终输入

由于RNN能够完成二进制加法操作，因此其实际上这三个隐藏单元学到了四种不同的模式，对应于前面有限状态自动机的四个节点。注意有限状态机节点对应的不是RNN隐藏节点，而是RNN隐藏节点的活动向量（笔者注：这个名词，对应英文activity vector，似乎很少在其他课程或者文献中见到。但是Hinton前几天发表的Capsule Network也用了这个概念，不知道是不是他自己创造的）——在每个时刻，有限状态自动机只能处于某个确定的状态，而RNN的隐藏单元也只能有一个确定的活动向量。因此RNN可以模拟有限状态自动机，但是表示能力是指数级地增强：有N个隐藏单元的RNN可以有2^N个可能的二进制活动节点（不过只有N^2个权重）。因此如果输入流有两个独立事件并行发生，则有限状态自动机需要将自己的状态数平方，但是RNN只需要将自己的隐藏单元数翻倍

为什么训练RNN很困难

RNN训练的难点在于，其正向传播过程和反向传播过程是有差别的。前向传播的过程中使用的是“挤压”函数（squashing function），例如logistic函数，来防止活动向量爆炸。但是反向传播是线性的，由前向传播时的结果决定，因此会面临线性系统面临的问题：如果权重都很小，那么梯度就会小，连乘以后会变得更小。那么，由于RNN是按时间展开的，所以这意味着经过几个时间步的传递，比较早的节点就得不到什么错误更新的信息了。同样的道理，如果权重都很大，那么梯度就会爆炸。不过，对于传统的前馈神经网络，除非这个网络特别深，否则不会出现这么糟糕的情况

上面说的这两种情况，分别对应于梯度消失和梯度爆炸两种情况。小心地初始化权重可以有效地解决这个问题，不过即使这样也很难让当前的输入输出利用其很多个时间步之前的信息，也就是说，普通的RNN结构很难处理长距离的依赖关系

假设系统中有两个稳定点，要通过RNN学出这两个点，那么，如果初始状态在这两个点附近，则最后的结果肯定是停留在对应的稳定点上，最终状态对初始状态改变量的导数接近0，出现了梯度消失。但是如果初始状态在两个点的交界处，那么无论往哪边偏一点，都会陷入到对应的点上，不同的偏差可能会导致截然不同的结果，也就是出现梯度爆炸

为了解决RNN难以学习长距离依赖的问题，大概有四种解决方法

- 使用一种叫做长短期记忆（Long-Short Term Memory, LSTM）的方法。这个方法在下一节介绍，简单来说，原理是修改神经网络的体系结构
- 使用更好的，可以处理非常小梯度的优化器。这个方法在下一章介绍
- 小心的初始化输入层到隐藏层的权重，非常小心地初始化隐藏层之间，以及隐藏层到输出层的权重，保证隐藏状态是一个弱耦合振子的大资源库。对于这样的网络，送进一个输入序列以后，会产生长时间的“回响”，通过回响来记住在输入序列里发生了什么。这种网络称为回声状态网络（Echo State Network, ESN）。它学到的只是隐藏层到输出层的连接
- 使用动量以及ESN里用到的初始化方法，学习网络中的所有连接

对RNN的改进

长短期记忆（LSTM）

个人感觉，Hinton这一块讲得不是特别好，所以视频被我跳过了。具体内容可以参考这篇闻名遐迩的博客。之后的博客里我可能也会再做一些介绍

Hinton神经网络与机器学习 8. 循环神经网络 II

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2018/01/16/UTNN-8-Recurrent-Neural-Networks-2/

无Hessian矩阵（Hessian-Free, HF）优化法及其应用

HF优化原理

假设损失函数是一个二次曲线（曲率恒定），确切说，是一个碗口向上的抛物线，那么在这个曲线上选定方向，向前一直前进，误差在重新变大之前会减少多少？这个值取决于梯度和曲率的比值，所以如果选定的方向是好的方向，这个比值应该高。也就是说，梯度小的地方曲率也小

之前在讲神经网络优化的时候，曾经提到过，如果选最陡的方向做梯度下降，那么有可能这个方向并没有指向我们真正想去的方向。不过，如果误差表面的横截面是一个正圆形，那么梯度的方向就可以用了。**牛顿法**的思路就是使用一个线性变换将椭圆形的横截面转变成圆形，即将梯度乘以曲率矩阵（Hessian矩阵）的逆：

$$\Delta \mathbf{w} = -\epsilon \mathbf{H}(\mathbf{w})^{-1} \frac{dE}{d\mathbf{w}}$$

如果损失函数的纵截面是二次曲线，ϵ选择合适，使用牛顿法可以直达靶心！但是这里的问题是，如果权重特别多，达到百万量级，那么Hessian矩阵里面会有万亿个项。因为曲率矩阵里的每一个元素指明的是，当我们沿着某方向移动时，梯度在任一方向的变化量，即

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_n} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_n \partial w_1} & \frac{\partial^2 E}{\partial w_n \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_n^2} \end{bmatrix}$$

如果损失函数的横截面是正圆，那么Hessian矩阵中非对角线元素都是0，即沿着该方向移动时，梯度在其它方向上不会变化，因此梯度方向就是真正想去的方向。但是如果损失函数的横截面是椭圆，Hessian矩阵中非对角线元素就不是0，沿着该方向变化来修改权重的某个分量，梯度在其它方向上的分量也会改变。因此如果同时修改所有权重，那么其它权重的变化会导致梯度在权重的第一个分量上的变化，使得事情可能会变得更糟（其它权重的变化导致梯度的第一分量符号被逆转，走向相反方向）。权重维度越大，就应该越小心

由于大的神经网络中Hessian矩阵的项太多，很难求逆，因此需要解决这个问题。LeCun组提出的一个方案是只关注对角线上的前若干个元素，但是这些元素太少了，如果这么做基本上要忽略掉矩阵中其它所有元素。另一种做法是使用各种方法近似Hessian矩阵，降低它的秩，例如HF法、LBFGS法等等。在HF法中，我们对Hessian矩阵做一个近似，然后假设这个近似是对的，使用共轭梯度法（conjugate gradient）来减小误差，接着再做一次近似和优化，如此往复。如果在RNN中使用这种方法，一般要加一个正则化，防止优化方法将隐藏层状态改变太多（因为RNN一般是按时间展开，如果较早时间步中的隐藏状态发生了巨大变化，会连锁影响后面所有的隐藏状态。因此一般可以加状态改变量的平方作为正则项）

共轭梯度法的核心思想是，先找到某个方向的最小值，一直前进，然后再找新的方向。寻找新的方向时，保证新的方向与前一个方向“共轭”，这样就不会破坏已经取得的前面所有方向的梯度。假设这个二次曲面是在N维空间上的二次曲面，共轭梯度法保证在N步之后可以找到最小点，因为它的原理就是每一步都在一个独立的（尽管不是正交的）方向达到该方向上梯度为0的点。一个更好的性质是，在远小于N步之后，算法就能非常接近最小值点。此外，共轭梯度在非二次曲面上效果也不错，也适用于mini batch的学习

HF优化器的原理就是，先使用一个二次表面逼近原始表面，然后再在这个近似的二次表面上使用共轭梯度法

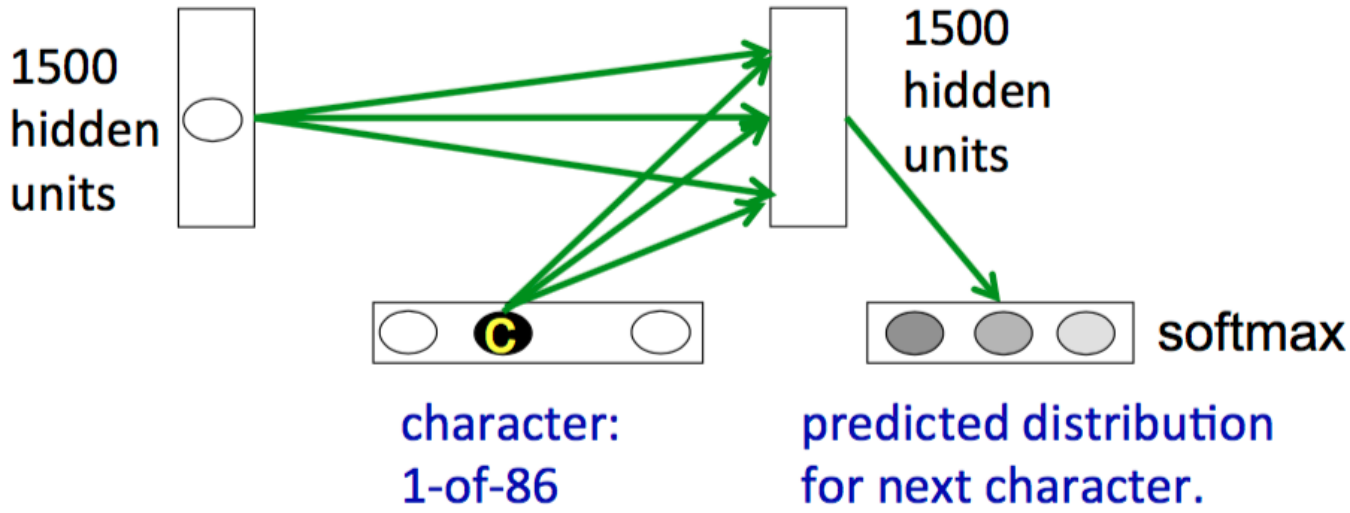
使用HF的字符模型

字符模型

通常情况下，讲到神经网络语言模型，提到的都是单词级别的语言模型，也就是把一句话看作是由单词组成的字符串。但是，就把这些句子看成是由字母组成的字符串也是有好处的。假设现在想通过“阅读”维基百科的所有文章来构建一个模型，如果需要将wiki里的所有文本都预处理成单词，那么就会比较麻烦，包括以下几个方面

- 词素（morpheme）。词素是表达词意的最小单位，但是具体一个单词里哪些部分可以算作是词素，这个是见仁见智的
- sub-word。有的时候，词的一部分可能反映了某种意思。例如，英语里以sn-开头的词都多少与嘴唇和鼻子有关，例如sneeze, snarl等等。这里是否包含了一些规律?
- 例如New York这样的词，大多数情况下是希望把它看做是一个词，但是有的时候，例如谈论New York minster roof的时候，应该把它看作是两个词
- 最后，对于黏着语，这类语言的单词通常都附着了多个语法成分，因此可以做出来特别长的单词。例如芬兰语的ymmärtämättömyydelläänsäkään对应了英语的“even with his/her/their lack of understanding”。这个词在字典里没有，而且日常生活中也很少用到

因此设计一个RNN来建立一个字符模型是有必要的，即使用T时刻的隐藏状态和输入来决定$T+1$时刻的隐藏状态，然后通过一个softmax来产生下一个可能的字符。假设输入字符表有86个， 隐藏层有一层，里面有1500个隐藏单元，那么下面给出了这个RNN的示意图



比较自然的RNN网络用来做字符级语言模型建模

但是一般并不会使用这种体系结构（这里存疑，因为感觉现在常见的用的的确是这种结构）。为什么呢？由于理论上任何前缀后面都可以跟86个字母中的任意一个，因此最后可以组成一个很大的树，如图所示。每当接受一个字符输入时，都会沿着这棵树向下走向一个新的节点。对于长度为N的词，如果用的是统计的方法，那么就需要存储整个的这棵大树，每条边实际上存储了给定前面所有前缀的情况下，接下来的字母的概率。在RNN

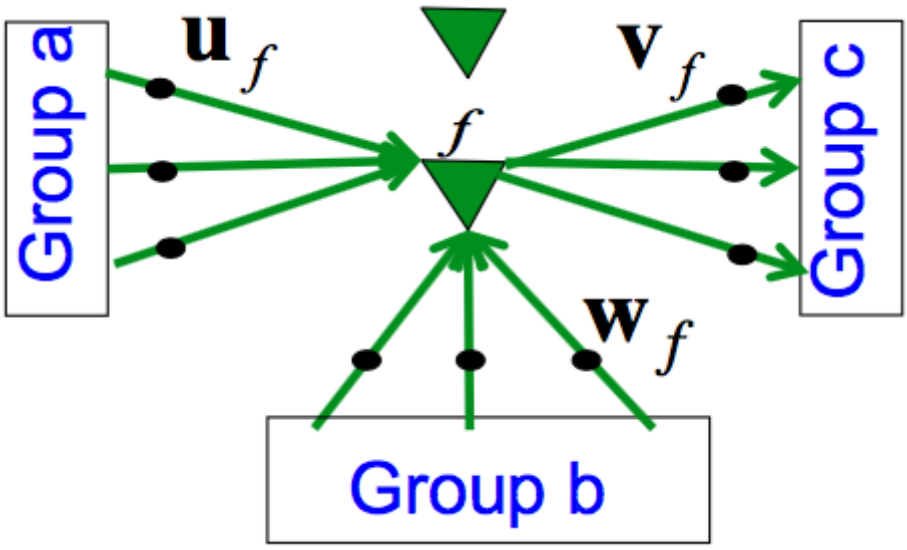
里，这样的前缀信息被隐藏状态向量来表示，也就是说，每个输入都会把当前的隐藏状态转化为一个新的隐藏状态

使用神经网络的隐层（隐式地）实现这棵树有一个好处，就是可以共享很多结构。例如当前面的输入为f, i, x时，已经可以判断出来正在输入的可能是一个动词，而这时如果再输入i，后面跟着的很可能会是n和g。这种“前面是动词，输入i，后面可能是ng”的知识可以共享给所有其他动词。需要再着重提一下的是，是当前的隐藏状态和当前的输入**共同**决定了接下里的输出。如果隐藏状态里没有动词信息，那么输入i并不一定期望后面是ng。

乘性连接（multiplicative connection）

基于前面的介绍，现在要解决的问题是，如何使用当前的输入字符来确定从当前隐藏状态到下一个隐藏状态的权重矩阵。对于原始的做法，实际上要把所有86个字符都试一遍，来找出一个1500 x 1500的矩阵，这样会有很多参数（86 x 1500 x 1500），容易过拟合。如果非要使用这种方法，也需要数据量很大的时候才行（但是我个人感觉这个条件已经可以很容易地满足了）

前面提到，对不同的单词，可以有一些通用的“特征”。那么一个思路就是，使用这样的特征来减少参数的数量，同时仍然实现这种“乘性连接”的功能。即，对着86个字符训练不同的转移矩阵，但是让这些矩阵共享参数。在这里，引入“因子”（factor）的概念，并假设因子的输入是两个组a和b，输出是一个组c。如下所示



因子示意图

因子会先对其两个输入组各自计算加权和$\mathbf{a}^\mathsf{T}\mathbf{u}_f$以及$\mathbf{b}^\mathsf{T}\mathbf{w}_f$，然后，将这两个积再相乘，可以得到一个标量。用这个标量去缩放输出权重$\mathbf{v}_f$可以得到组c的输入，即

$$\mathbf{c}_f = (\mathbf{b}^\mathsf{T}\mathbf{w}_f)(\mathbf{a}^\mathsf{T}\mathbf{u}_f)\mathbf{v}_f$$

沿用之前的例子，这时，因子从上一个隐藏状态接收到1500个参数，从字母表接收到86个参数，产生给下一个隐藏状态1500个参数，这样，因子总共只需要3086个参数

这个过程也可以换一个角度理解，即每个因子实际上定义了一个非常简单的转换矩阵，其秩为1。上面的等式等价于下式

$$\mathbf{c}_f = (\mathbf{b}^\mathsf{T}\mathbf{w}_f)(\mathbf{u}_f\mathbf{v}_f^\mathsf{T})\mathbf{a}$$

这里外积$\mathbf{u}_f\mathbf{v}_f^\mathsf{T}$就是秩为1的矩阵

最后，将所有因子的$\mathbf{c}_f$加起来，就得到了送入c的转换矩阵

$$\mathbf{c} = \left(\sum_f (\mathbf{b}^\mathsf{T} \mathbf{w}_f)(\mathbf{u}_f \mathbf{v}_f^\mathsf{T}) \right) \mathbf{a}$$

也就是说，每个因子都给出了一个秩为1的矩阵，而字母表$\mathbf{b}$中的每个字母决定了这些矩阵的权重

使用HF优化器预测下一个字符

Ilya Sutskever从英文维基挑出了500万个长度为100个字符的句子，从第11个字符开始预测。训练在一块GPU上持续进行了大约一个月，最后得到的模型效果还不错，可能是到2012年为止效果最好的单神经网络模型（除非将多个模型组合起来才能击败他）。这个模型的一个显著特点是，可以匹配长距离的括号和引号

具体生成字符串的做法是，以默认的隐藏状态开始，给出一个句子。模型随着读入这个句子的字符时会更新隐藏状态。读完这个句子以后，开始预测接下来的字符。对于接下来的字符，模型会产生一个概率分布，然后我们随机从这个分布中取一个字符，告诉它这个字符就是我们要的，让它接着预测（也就是说默认它预测的字符总是对的），直到我们想结束这个过程为止

观察这个模型产生的文本，可以发现有这么一些特点：

- 尽管有的词搭配很怪，例如Opus Paul at Rome，但是考虑到Opus, Paul和Rome之间存在关系，还是可以理解的
- 不会产生特别长的句法结构，每一个句号后面新的句子都会开始一个新的主题
- 很少产生英语单词表里没有的词。即便是产生了，这个词也很像英语单词，例如ephemerable
- 尽管有时候会产生不匹配的括号，但是它也会匹配长距离的括号和引号

另一个有趣的实验是，给一个自造词，看看会有什么结果。Hinton选择了thrunge这个词，它不存在于词典里，但是看着像是一个动词。如果前面给的是Sheila，那么对于Sheila thrunge模型会预测接下来的字母是s，这意味着它能发现主语是单数，前面有点像动词。如果前面给People，那么对于People thrunge模型预测接下来的字符是空格，说明它能掌握people是复数这个特征。如果前面给Shiela, 并将t大写，那么对于“Shiela, Thrunge”它会补成“Shiela, Thrungelini del Rey”，即一个人名——而且这个人名还不错

经过实验观察，可以肯定的是，模型能够学到以下知识

- 能够知道什么是词，长什么样。它产生的词绝大部分都是英语单词，而且知道句首字母大写，产生出的日期格式也没什么问题。即便是产生了虚构的，非英语的单词，也很难看出来。它可以产生专有名词，也知道日期和数字应该出现的语境
- 能够产生匹配的括号和引号，而且某种意义上还会对括号和引号记数。例如，如果给出的字符没有括号/引号，它后面基本不会给出右括号/引号。如果给出的字符有左括号/引号，那么在20个字符以内基本就会生成对应的符号
- 能够掌握某些语法规则，不过很难说具体的语法规则是什么，看上去像是很多比较弱的句法关联。例如，在产生“柏拉图”这个词以后不久，就会产生“维特根斯坦”。模型大概知道卷心菜和蔬菜之间存在关系，但是并不知道具体是什么关系。当然，这种特点是可以理解的，人类也会存在。假设现在玩一个游戏，提问者问完问题以后回答者必须马上喊出答案，那么在听到问题“奶牛喝什么”以后大部分人的意识都是喊出“牛奶”。实际上奶牛是不喝牛奶的，但是人们听到“奶牛”和“喝”以后都会想到牛奶，即便逻辑上不通

Mikolov组使用了类似的技术来预测下一个单词会是什么单词。总体而言，RNN不需要太多数据就能达到其他模型能够达到的效果，而且随着数据集的增大，RNN的效果提升得更快

回声状态网络

回声状态网络（Echo State Network, ESN）使用一种更聪明的技巧来让学习RNN变得更容易。它把网络中的链接初始化为一个大的，耦合的振子（oscillator）库，并将输入转化为这些振子的状态，通过这些状态来预测输出。ESN学到的唯一东西是如何将振子与输出相关联，而不用学习隐藏层之间，以及隐藏层与输出层之间的链接。不过，ESN需要大量的隐藏状态

更具体地说，ESN随机初始化所有的隐藏状态并将其固定，然后训练这些隐藏状态如何去影响输出。这种做法与感知机的思路有点像，它最后训练的就是从大量随机隐藏状态到输出的线性模型。此外，这种方法与支持向量机SVM也有异曲同工之妙。但是，这些随机的权重必须设置得非常精妙，才会避免出现梯度消失/梯度爆炸的现象。做法是保证隐藏层到隐藏层的权重满足如下性质：在每个迭代之后，活动向量的长度保持不变。即对于某个线性系统而言，其矩阵的谱半径（spectral radius）为1，也就是隐藏层到隐藏层的权重矩阵的最大特征值为1。在非线性系统中，如果权重被设置在一个合适范围之内，则输入能够在循环状态中回响很久

另一种重要的做法是使用稀疏连通性，即对隐藏层到隐藏层的权重矩阵，不去设置大量中等大小的权重，而是只保留少数几个很大的权重，其它的权重都设为0。这样，可以得到很多松耦合的振子，使得信息可以在网络的一部分中停留一段时间，不会很快被传播到其它部分。对于这种方法，需要小心选择输入层到隐藏层的连接数，要既保证能够得到松耦合的振子，还不会清除掉振子中包含的关于最近历史的信息

（中间的实验略）

ESN的好处是训练起来很快，因为只需要拟合一个线性模型，而且对一维时间序列的建模能力非常好，可以预测未来很长一段时间的数据。但是它不适合对高维数据建模（例如声音帧、视频帧等）。为了建模这样的数据，它需要的隐藏层单元数比RNN所需要的多很多。最近（2012年），Ilya Sutskever发现，使用ESN用到的初始化方法来初始化RNN，并使用带有动量的rmsprop优化方法，能很有效地训练RNN

Hinton神经网络与机器学习 9. 提高神经网络的泛化能力

- **本文作者：** Tingxun Shi

- **本文链接：** http://txshi-mt.com/2018/01/20/UTNN-9-Ways-to-Make-Neural-Networks-Generalize-Better/

提高网络泛化能力的若干方法

概论

当训练网络用的数据量比较小，而网络又特别复杂时，就容易发生过拟合现象。其原因是，尽管数据集的输入和输出之间存在着某种规律性的关系，但是任何有限的训练集都会包含采样误差，也就是说，训练集中的某些规律可能是偶然的，只是因为选中了某些特殊的训练样例才会出现。因此，拟合模型时，并不能确定所有规律都是真实的规律，也不能保证如果重新对样本抽样，这些规律是否还能存在。因此，模型同时拟合了必然规律和偶然规律。如果模型参数太多，拟合得“太好”，就会很好地拟合训练集中的偶然误差，不好泛化，因此需要一些正则化方法来防止过拟合

解决过拟合的最好方法，也是最简单的方法，就是获得更多数据。第二种方法是精巧地去限制网络的能力，使其有足够的能力拟合必然规律，同时难以拟合偶然规律。这种方法比较困难，不过也有若干种做法。第三种方法在下一讲介绍，就是讲多种不同的模型平均起来。如果这些模型形式不同，犯的错误也不同，那么三个臭皮匠赛过诸葛亮，可以用训练数据的不同子集分别训练多个模型，这种方法称为装袋法（bagging）。第四种方法是贝叶斯方法，即只是用一个神经网络结构，但是使用不同组权重来分别预测，最后对预测结果求平均。这三节主要是看第二种方法

单论限制模型能力，也有若干种做法。其一是从网络结构下手。如果能缩小隐藏层层数和每层的单元数，就能控制网络中连接的个数，即参数数量。其二是使用比较小的权重做初始化，因此它需要非常长的时间才会过拟合，然后提早停止训练使其没有过拟合的机会。这种做法基于的假设是网络先找到必然规律。第三种方法非常常见，即对隐藏层层数和每层单元数不做限制，而是使用惩罚项或者权重的平方/绝对值做约束，来惩罚过大的权重。最后，可以通过往权重中或者激活中加入噪声来控制模型能力。通常情况下，会把以上这些方法组合起来使用

验证集

上述这些方法通常都会用到一些超参数，例如隐藏层层数、隐藏节点数、权重惩罚项的大小等等。很多人可能会通过在测试集上实验来选取一组最好的超参数，但是这种做法是非常错误的，因为按照某个测试集精细调整过的模型不一定在新的测试集上好用。也就是说，你会被假象蒙蔽。这里举一个极端情况：假设测试数据完全是随机的，答案不依赖于输入，或者不能被输入预测（例如某些金融领域的问题）。如果选择了某个在测试集上表现很好的模型，这个模型比瞎猜的效果好得多，你以为发现了宇宙的真谛，可以走上人生巅峰了，但是实际上用另一个测试集测试，这个模型基本还是会跟瞎猜的效果半斤八两

正确的做法是将整个数据集分成三个部分：**训练数据**用来训练模型，**验证数据**不用来训练，而是用来决定如何设置超参数。也就是说，使用验证数据来检查模型效果。做完这些以后，使用最好的超参数训练一边模型，然后在**测试数据**上看模型的最终效果。因此，测试数据只能用一次，目的是给出网络效果的无偏估计

有一种方法能更好地估计模型效果，就是让数据轮流做验证集。具体做法是，还是保留一份测试集来做最终的无偏估计，不过将剩下的数据分成N等份，每次用$N-1$份数据训练，第N份数据做验证，然后做一个轮转。例如，将数据集分成5份，第一次用第1 2 3 4四组数据训练，第5组数据验证，第二次用2 3 4 5四组数据训练，第1组数据验证，以此类推。这样可以得到N个估计。这种方法叫做N折交叉验证。注意这N个估计彼此并不独立

提前停止法

如果要在一个小电脑上训练一个大模型，没有时间用不同超参数训练好几个模型，那么有一种方法可以有效防止过拟合。首先，用一些小的权重来初始化模型。随着模型的训练，这些权重会增长。此时，观察模型在验证集上的效果，当它的效果变差时，结束训练。注意如果使用的评估指标是错误率而不是平方误差，那么这个指标可能会在验证集上浮动，因此最保险的做法是确定模型效果变差时再回到没有变差的那个点

这种方法能控制模型能力的原因是权重比较小的模型通常能力不足，而权重没有时间变得特别大。那么为什么权重小会导致网络能力不足呢？考虑一个有若干输入单元、若干隐藏单元和若干输出单元的模型。当权重很小时，如果隐藏单元是logistic单元，输入接近0，那么logistic的值会在线性区域的中间部分，即此时logistic单元的表现跟线性单元很像。也就是说，权重很小的时候，整个网络实际上像是一个线性网络，将输入直接映射为输出。随着网络的学习，权重变大，达到logistic单元的非线性区域，这些参数才真正被用上了。从另一个角度讲，这意味着可用的参数随着权重变大而变多，因此提前停止实际上限制了真正可用的参数数量，让网络在训练集上的表现和在验证集上的表现几乎相等——这种现象可以理解为，网络学习到了必然规律，还没开始学习偶然规律

限制权重大小

权重惩罚

通过限制权重大小来控制模型的复杂程度，标准做法是引入惩罚项，惩罚大的权重。其隐含的假设是，小权重的网络比大权重的网络简单

通常使用L2权重惩罚项来限制网络权重，也就是限制权重的平方。这种方法在某些神经网络的文献当中称为权重衰减，因为对惩罚项求导的结果像是把权重强推到零。引入L2惩罚项以后，代价函数变为

$$C = E + \frac{\lambda}{2}\sum_i w_i^2$$

将其对w_i求偏导，有

$$\frac{\partial C}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda w_i$$
$$\text{when } \frac{\partial C}{\partial w_i} = 0, w_i = -\frac{1}{\lambda}\frac{\partial E}{\partial w_i}$$

因此，在达到代价函数的最小值时，只有当该权重有比较大的误差导数时，这个权重才会比较大。因此加入L2惩罚项以后不会有那种值很大又什么都不做的权重存在。这种做法的好处是，首先，极大提高了网络的泛化能力，因为如果出现了大值而无用的权重，网络可能会用它来拟合偶然规律（取样误差）。其次，这样会使模型变得平滑，使输出的变化速度慢于输入的变化速度。假设网络接受两个相似的输入，有L2正则项的那个会倾向让两个输入的权重各为$w/2$，而没有正则项的会让一个输入的权重为w另一个为0。如果现在将其中某个输入替换为一个完全不同的值，前者的变化会比后者小

除去L2正则项，L1正则项也是一个常用的方法，它所惩罚的是权重的绝对值。使用L1正则项会使大部分权重变为0（L2正则项会让大部分权重变得很小，但不会完全为0）。或者，可以使用一种更极端的惩罚函数，使得代价函数的梯度在权重很大时反而很小，这样它会把小的权重都淘汰掉，只剩下很少几个大的权重

权重限制

要在权重上做文章来避免过拟合，除了对权重加以惩罚以外，还可以对权重施加一些限制条件。也就是说，对隐藏层单元和输出层单元的输入权重，对其范数设置一个封顶的值。如果权重的范数超过了这个限定值，就把每个分量除以限定值，做放缩。这种方法与权重惩罚相比有如下几点好处

- 首先，可以人为设置一个比较合理的上限
- 其次，可以避免隐藏单元在零点附近震荡，此时它的权重都非常微小，起不到什么效果
- 最后，可以防止权重爆炸

需要注意的是，当某个单元达到了限定值，它所有权重的有效惩罚取决于大的梯度。所以如果某些输入权重有很大的梯度，就会试图增大自己的范数，把其它权重的范数变小。也就是说，如果把权重限制也看做是一种惩罚项，它做的事情是让大权重的值变得合理，而让小权重更小。这种做法比设定一个固定的惩罚值来将不相关权重推到0要有效。实际上，这种做法就像是让某些限制条件满足的拉格朗日乘子

使用噪声做正则项

假设现在有一个简单的线性神经网络，代价函数是平方误差，网络结构是输入单元直接连接到输出单元。现在，往输入添加高斯噪声，即输入变为$x_i + N(0, \sigma_i^2)$。经由权重w_i的处理，最后的输出也会带有高斯噪声，即输出变为$y_j + N(0, w_i^2\sigma_i^2)$。也就是说，额外增加的这个方差实际上使得平方误差项又增加了一个随机项，该随机项满足均值为0，方差为σ_i^2的高斯分布。因此最小化新的平方误差时，也会最小化$N(0, w_i^2\sigma_i^2)$这一项，这一项的作用可以看作是给w_i增加了一个系数为σ_i^2的L2惩罚项。具体推导如下

$$y^{\text{noisy}} = \sum_i w_i(x_i + \epsilon_i) = \sum_i w_i x_i + \sum_i w_i \epsilon_i \ \ (\epsilon_i \sim N(0, \sigma_i^2))$$

$$\begin{aligned}
\mathbb{E}\left[(y^{\text{noisy}} - t)^2\right] &= \mathbb{E}\left[\left(y + \sum_i w_i \epsilon_i - t\right)^2\right] = \mathbb{E}\left[\left((y-t) + \sum_i w_i \epsilon_i\right)^2\right] \\
&= (y-t)^2 + \mathbb{E}\left[2(y-t)\sum_i w_i \epsilon_i\right] + \mathbb{E}\left[\left(\sum_i w_i \epsilon_i\right)^2\right] \\
&= (y-t)^2 + 2(y-t)\sum_i w_i \mathbb{E}[\epsilon_i] + \mathbb{E}\left[\sum_i w_i^2 \epsilon_i^2\right] + \mathbb{E}\left[\sum_{i \neq j} w_i w_j \epsilon_i \epsilon_j\right]
\end{aligned}$$

由于$\epsilon_i \sim N(0, \sigma_i^2)$，且对$i \neq j$，$\epsilon_i$和$\epsilon_j$相互独立，因此$\mathbb{E}[\epsilon_i] = 0$，$\mathbb{E}[\epsilon_i \epsilon_j] = \mathbb{E}[\epsilon_i]\mathbb{E}[\epsilon_j] = 0$。又$\mathrm{Var}[\epsilon_i] = \sigma_i^2 = \mathbb{E}[\epsilon_i^2] - (\mathbb{E}[\epsilon_i])^2 = \mathbb{E}[\epsilon_i^2]$，因此上式最后可以化简为

$$\mathbb{E}\left[(y^{\text{noisy}} - t)^2\right] = (y-t)^2 + \sum_i w_i^2 \sigma_i^2$$

对于更复杂的网络，可以往权重中加入高斯噪声来限制网络的能力。这种做法不再等价于使用L2惩罚项，但是效果却更好，尤其是对于RNN更是这样

也可以在激活中使用噪声，将其作为一种正则化手段。假设使用反向传播来训练一个多层神经网络，网络的隐藏单元使用logistic函数做激活函数。在做正向传播时，动一些手脚：让隐藏单元随机输出0或者1（输出1的概率就是sigmoid函数的输出），但是反向传播的时候使用的是通过正常正向传播得出的，确定的实数。这种方法会让网络在训练集上的表现变差，而且训练过程变慢。但是如果用在测试集上，效果会有显著提高

贝叶斯方法简介

本节是对贝叶斯方法的一个简介。贝叶斯方法的原理是，不再考虑模型参数最可能的设置，而是先考虑参数的所有可能设置，然后对观察到的数据，判断每种设置的概率。贝叶斯理论的思考框架假设对任何事件都有一个先验概率分布，数据提供一个似然项，将先验分布和似然结合就可以得 到一个后验分布。其中，似然项更倾向使数据更有可能成为观测到的样子的参数设置，可以不符合先验分布。如果能得到足够的数据，不管先验分布与似然有多大的偏离，似然都会“克服”先验分布。如果数据足够，真相必然会水落石出，也就是说，即便先验是错的，最后也能得到正确的假设

来看一个关于掷硬币的例子。假设有一枚硬币，我们对其内部构造一无所知，只知道投掷它会以概率p得到正面，概率$1-p$得到反面。现在投掷了100次，得到53个正面，那么参数p应该是多少？频率学派通过最大似然法会说$p = 0.53$。注意这个数计算的过程并不是特别显然。将“扔100次硬币得到53个正面”这个事件记为D，则这个事件发生的概率$P(D) = p^{53}(1-p)^{47}$。令$dP(D)/dp = 0$，可以得出$p = 0.53$

不过，用最大似然法决定模型参数存在一些问题。假设我们只投了一次硬币，得到一个正面，那我们并不能说这个硬币掷出正面的概率是1。事实上，更可靠地猜测应该是0.5，但是如何证明呢？或者更重要的是，是否存在一个唯一且合理的答案？

我们知之不多，没有太多数据，因此难以确保p究竟应该是多少。所以我们应该拒绝给出一个单个的答案，而是给出所有可能答案的整个概率分布。如果先验的“信念”是硬币有一半机会掷出正面，那么$p = 0.5$是很有可能的，$p = 1$是很不可能的。这里，先把先验设为所有不同的p出现的机会都相等，然后，观察到硬币掷出了一个正面，所以对每个可能的先验p，乘以以这个参数为基础，掷出正面的概率。例如，对$p = 1$，那么它预测只会掷出正面，因此对这个参数乘以1。对$p = 0$的先验乘以0，对$p = 0.5$的先验乘以0.5，以此类推。这样可以得到一个未归一化的后验分布。最后，对整个线下面积归一化，可以得到投掷第一次硬币以后参数的概率

密度。接下来，再做一次实验，假设投出了反面，重复做上述过程，可以得到一个新的概率密度。经过剩余的98次投递，得到的是一个峰值在0.53左右的一个图像

所有以上这些过程都可以用贝叶斯定理来描述。令参数W和数据集D的联合概率为$P(D,W)$（对有监督学习问题，数据集包含了目标值）。这个值可以用两种方式计算

$$P(D)P(W|D) = P(D,W) = P(W)P(D|W)$$

将等式两边同时除以$P(D)$，有

$$P(W|D) = \frac{P(W)P(D|W)}{P(D)}$$

贝叶斯定理告诉我们，W给定数据D的后验分布，等于W的先验分布与给定W的值观察到已知数据D的概率的乘积，再使用$P(D)$归一化。而数据D的概率可以表示为

$$P(D) = \int_W P(W)P(D|W)$$

由于$P(D)$是对所有可能的W求积分，因此$P(D)$不依赖于具体的W的值。不过$P(W)$和$P(D|W)$依赖具体的W

权重衰减的贝叶斯角度解释

如果要使用完整的贝叶斯方法，就要计算模型每种可能参数组合的后验概率。不过有一种简化方法，就是找到一组参数设置，它既能满足对参数的先验信念，也能符合所观测到的数据。这种方法称为最大后验（Maximum A Posteriori, MAP）学习，它可以很好地解释使用权重衰减法时真正发生了什么

使用有监督的最大似然学习方法来最小化平方误差的时候，找到的权重向量需要使残差（目标值与网络预测值之差）的平方最小化。这个任务目标等价于要找到一个权重向量，使之最大化正确答案的概率密度的对数值。为了说明这样的等价性，假设正确答案是通过往神经网络的输出中添加高斯噪声来产生，那么在这种情况下得到正确答案的概率是多少？这个问题可以这么描述。假设网络的输出$y_c = f(\text{input}_c, W)$，由于真实的目标值是输出添加噪声的结果，因此条件概率密度可以写为

$$P(t_c|y_c) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{(t_c - y_c)^2}{2\sigma^2}\right\}$$

对等式左右两边取对数，有

$$-\log P(t_c|y_c) = k + \frac{(t_c - y_c)^2}{2\sigma^2}, k = \frac{1}{\sqrt{2\pi}\sigma}$$

因此，如果代价函数是得到正确答案的概率的负对数，那么代价函数的形式相当于是在最小化两者距离的平方

接下来来看一下解决这个问题的贝叶斯方法。前面说过，要使用贝叶斯方法，就要找到所有可能权重对应的全部后验分布——对于神经网络来讲，这么做有点难。虽然可以使用蒙特卡罗法来近似这个分布，但是有一种更简单的做法，就是找到可能性最大的权重向量。尽管可以先随机初始化一个W，然后通过让$P(W|D)$变大来调整W，但是这种做法容易陷入局部最小值。更好的方法是在对数范畴内解决这个问题，即若要最小化代价函数，就计算概率的负对数值。OK，现在的目标是最大化不同训练样例产生目标值的概率的乘积，假设各个样例c的输出误差是独立的，则有

$$P(D|W) = \prod_c P(t_c|W) = \prod_c P(t_c|f(\text{input}_c, W))$$

由于对数函数是单调的，不改变原函数的单调性，因此可以两边取对数（计算概率的乘积容易发生下溢）

$$\log P(D|W) = \sum_c \log P(t_c|W)$$

同样的方法可以用到贝叶斯定理里。定义代价函数为$-\log P(W|D)$，则有

$$\text{Cost} = -\log P(W|D) = -\log P(W) - \log P(D|W) + \log P(D)$$

由于$\log P(D)$与W无关，因此优化W时可以不考虑这项。此外，$-\log P(W)$，也就是权重的负对数先验，只取决于W。由前面的说明，最大化权重的对数概率相当于最小化某个距离的平方，因此如果假设权重的先验满足一个期望为0的高斯分布，那么最小化权重的平方等价于最大化权重的对数概率。这个概率密度可以写为

$$P(W) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{w^2}{2\sigma_w^2}\right\}$$
$$\Rightarrow -\log P(W) = \frac{w^2}{2\sigma_W^2} + k$$

这样，就可以得到一种对权重衰减（或称权重惩罚）的解释。假设真实目标值增加了期望为0，方差为σ_D^2的高斯噪声，权重的先验满足一个期望为0，方差为σ_W^2的高斯分布，那么记代价函数为C^*，有

$$C^* = \frac{1}{2\sigma_D^2}\sum_c (y_c - t_c)^2 + \frac{1}{2\sigma_W^2}\sum_i w_i^2$$

等式两边同时乘以$2\sigma_D^2$，有

$$C = E + \frac{\sigma_D^2}{\sigma_W^2}\sum_i w_i^2$$

这里E就是通常优化神经网络时用到的平方误差，第二项的σ_D^2/σ_W^2就是权重的惩罚项。这意味着，如果前面提到的先验概率分布都成立，那么惩罚项就不再是一个超参数，不是一个随意的值

MacKay的权重惩罚项设定法

MacKay在20世纪90年代曾经提出过一种不需要验证集的权重惩罚项设定法。在学到可以最小化平方误差的模型以后，可以通过残差误差的方差来找到输出方差的最佳值。对于权重的先验，由于其满足高斯分布，可以先猜测方差是多少，然后使用一种称作经验贝叶斯的方法来调整，即就使用学到的权重的方差。具体内容可以参考下述论文

MacKay, D. J. (1995). Probable networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, *6*(3), 469-505.

Hinton神经网络与机器学习 10. 组合多个神经网络以提高泛化能力

为什么模型组合效果更好

模型组合是一种在拟合必然规律和过拟合采样误差间取得平衡的方法。当训练数据比较有限时，就容易造成过拟合，如果将不同模型的预测结果做平均，则可以降低过拟合风险。对于回归问题，平方误差可以分解为偏差项和方差项。如果模型能力有限，偏差项就会比较大，它表示模型逼近真是函数的能力有多差。如果模型对训练集里的采样误差的把握能力比较强，方差项就比较大——之所以叫“方差”，是因为如果用同样大小、来自同样分布的另一个训练集来训练模型，拟合出来的新模型会完全不一样。**模型平均实际上在做的事情是把方差平均掉**。这样，就可以把那些能力强而方差大的模型利用起来，而这些模型通常都是低偏差的

接下来来分析一下为什么可以得到这个结论：对测试集中的任意一条数据，都可能会存在一个模型，其在该数据上的效果比众多模型的平均效果要好，而不同的模型可能在不同的数据上取得很好的效果。如果对任意一条数据，都有一个模型的效果表现非常突出，那么将所有模型预测结果组合平均以后，这个平均模型在大部分数据上的效果就会比任意一个单体模型强。因此，目标是如何获得一些既有不低准确度，而且还能在预测时产生非常不同结果的小模型

“将网络进行组合可以降低方差”这个结论，也可以从数学角度给出证明。这里要比较的是两种期望平方误差：其一是随机选择一个模型，对所有数据进行预测得到的平方误差，另一是将所有模型平均起来对数据预测得到的误差。假设一共有N个模型，对任一样本，平均模型的预测为

$$\bar{y} = \langle y_i \rangle_i = \frac{1}{N}\sum_{i=1}^{N} y_i$$

其中尖括号$\langle\rangle$表示求期望。假设现在随机选择一个模型做预测，那么得到的平方误差的期望可以写为（这里将讲义里的尖括号展开了）

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N}(t-y_i)^2 &= \frac{1}{N}\sum_{i=1}^{N}\left((t-\bar{y})-(y_i-\bar{y})\right)^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}\left((t-\bar{y})^2+(y_i-\bar{y})^2-2(t-\bar{y})(y_i-\bar{y})\right) \\ &= (t-\bar{y})^2+\frac{1}{N}\sum_{i=1}^{N}(y_i-\bar{y})^2-2(t-\bar{y})\frac{1}{N}\sum_{i=1}^{N}(y_i-\bar{y})\end{aligned}$$

上面等式右边，第二项是模型方差的期望，第三项中，由于各个模型是独立的，其预测结果期望为$\bar{y}$，因此$1/N\sum y_i = \bar{y}$，即第三项为0。第一项则是平均模型的平方误差。这意味着，随机选择模型做预测，平方误差的期望比平均模型的平方误差多，多出来的值等于各模型方差的期望。可以用一个更直观的例子来说明这个事情。假设对某条数据，有两个模型预测，一个预测结果是$\bar{y}-\epsilon$，一个预测结果是$\bar{y}+\epsilon$，那么方差的期望是ϵ^2。出现这样状况的原因是，离目标t越远的预测值会贡献更大的误差，这个误差经过平方计算会被放大。需要注意的是，当噪声不符合高斯分布时，不应该采用平均模型的方法。如果模型预测的是输入属于各个类别的概率（求解分类问题），简单求平均也是一种好的做法。假设有两个模型，一个对正确标签给出的概率是p_i，另一个给出p_j，测量指标是得到正确标签概率的对数值，那么一般会有

$$\log\left(\frac{p_i+p_j}{2}\right) \geq \frac{\log p_i + \log p_j}{2}$$

这是由对数函数的性质决定的

因此，综上所述，现在的问题是怎么让众多模型得到不同的预测结果。可以仅使用某单个学习算法，如果这种算法每次都会陷入到不同的局部极小值，就可以达到期望的效果。也可以训练不同类型的模型，例如决策树、高斯过程模型、SVM等等。对于神经网络，如果想获得不同的结果，可以设置不同的隐藏层数、设置不同的隐

藏单元数、设置不同的隐藏单元类型（不同模型使用不同激活函数）、使用不同的正则化方法（例如一些使用提早停止法、另一些使用L2正则化），以及使用不同的学习算法（例如一些使用full batch梯度下降，另一些使用mini-batch 梯度下降）

使用不同的数据训练模型也可以达到同样的效果。**装袋法**（bagging）使用数据的不同子集来训练不同模型，其中这些子集是使用有放回抽样构造的。这种方法的典型应用是随机森林法（可以参考林轩田老师关于装袋法和随机森林的讲解），不过神经网络用装袋法效率比较堪忧。另一种方法是**boosting**算法，这种算法串行训练一批能力比较差的模型，对每个模型单独设置各个样本的权重（对前一个模型判断对的样本降低权重，对前一个模型判断错的样本提高权重）。这个算法的细节也可以参考林轩田老师的讲义

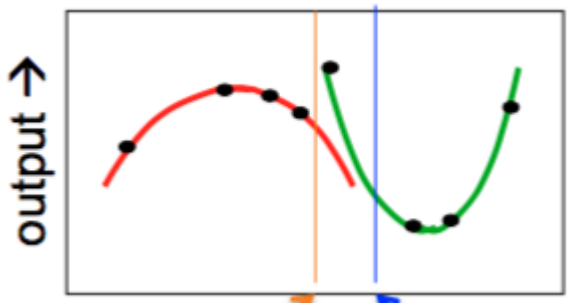
专家组（Mixture of Experts）

专家组方法的核心思想是，把数据集看作是来自不同领域信息的组合，训练的时候，训练多个神经网络，每个网络只专精某一个领域（称为专业化, specialization）。具体说来，每个模型只专注于训练数据的某个子集，不去学习其它数据。这种做法对小数据集不太可行，但是如果数据集非常大，效果会很好

这引入了“局部模型”（local model）和“全局模型”（global model）两个概念。最近邻法是局部模型的一个典型特例，而使用所有数据训练出的多项式模型是全局模型的代表。局部模型训练起来很快，而且对得到的结果做一些拒不平滑可以显著改善模型效果。全局模型一般训练起来很慢，而且可能会不太稳定，因为数据的微小变动可能会导致训练出模型很大的改变，因为所有模型参数都会依赖于全部数据

在这两者中间，有一种概念叫做“多局部模型”（multiple local model），这种模型适用于数据集可以划分为若干区域（regime），每个区域的输入输出关系还不一样的情况。例如，对于金融数据，不同的经济状况可能会是输入输出的映射关系不同，但是你可能并不知道如何划分这些经济状况，这种划分方法本身也在学习内容之列。那么接下来的问题就是，如何将数据进行划分

一种自然的想法将训练数据聚类，每一类对应一个区域。方向是对的，但是此时不能根据输入向量的相似度聚类，而是要根据输入输出的关系聚类。以下图为例，如果将数据按输入输出关系（橙色的线）划分，那么两组数据可以容易地用两条抛物线拟合；如果只按输入（蓝色的线）划分，那么拟合难度就大了



按照不同方法聚类的效果对比

接下来是如何设计代价函数。假设一共有N个模型，每个模型的预测结果为y_i。如果将代价函数设计为

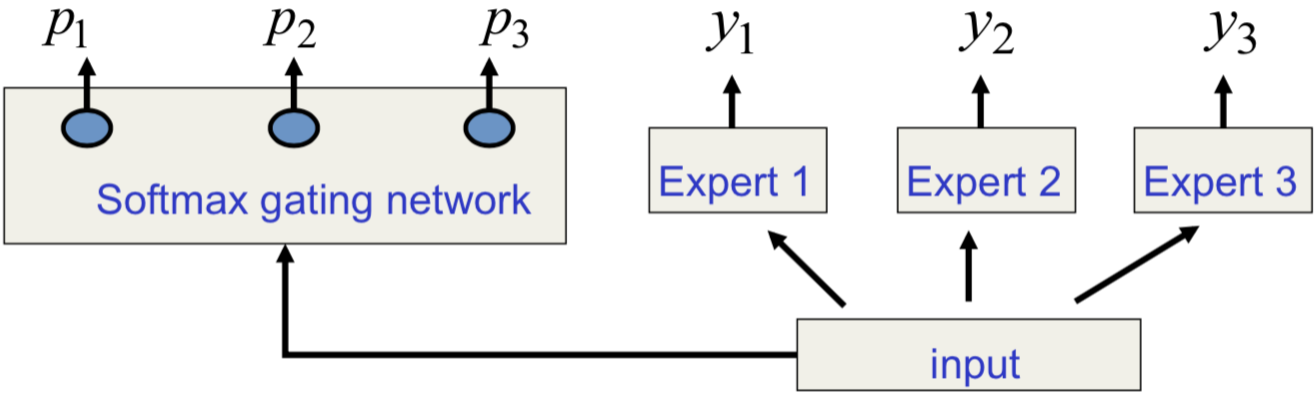
$$E = \left(t - \frac{1}{N} \sum_{i=1}^{N} y_i \right)^2$$

那么这样会鼓励模型之间的协作，因为模型会学着弥补某些模型犯的错误。需要注意的是，这种方法是*在训练的*就去对模型的结果进行平均。假设对某个模型i，没有包含它时，其余模型的平均预测值$\bar{y}_{_i}$离目标t特别远，那么这种方法会使得y_i离t也很远，只不过是反方向的远。例如，假设一共有3个模型，其中两个模型的预测平均值是20，目标值为0，那么第三个模型如果预测是-40的话，所有模型的平均预测值就跟目标值吻合了。但是，可以换一种思路：假设另外两个模型都挂了，第三个模型如果离目标值很近，那么真正预测的时候，对

于这样的数据，直接撇开另外两个模型，只用第三个模型预测不是挺好么？这就催生了另一种目标函数的核心思想，即鼓励模型的专家化。其具体形式为

$$E = \frac{1}{N}\sum_{i=1}^{N} p_i (t - y_i)^2$$

此时实际上是把每个模型单独与目标值比较，然后用另外一个“管理器”来管理每个模型的权重。这个权重可以看作是一种概率，如果只选择第i个模型，那么它的权重/概率就是1，其它都是0。这样最后就会得到若干专家，每个专家都只专精于处理训练集的一小部分，对其它部分一无所知



专家组体系结构

上图给出了专家组体系结构的示意图。对于每个输入，每个专家会给出预测值，每个专家的概率由管理器来控制。管理器可能有多层，但是最后一层肯定是一个softmax层，这样管理器就可以应对任意多个专家。将管理器的输出和专家的输出相结合，就可以得到代价函数的值。具体说来，有

$$E = \sum_i p_i (t - y_i)^2$$

$$p_i = \frac{e^{xi}}{\sum_j e^{xj}}$$

将误差对专家的输出y_i求偏导，可以得到训练每个专家的信号，即

$$\frac{\partial E}{\partial y_i} = p_i (t - y_i)$$

如果该专家对该数据一无所知，那么对应的p_i就会很小，因此这个专家接收到的梯度就会非常小，其内部的权重基本不会受到修改

将误差对管理器的输出求偏导，可以得到训练管理器的信号。实际上，这是在求误差对softmax输入量x_i的偏导数，有

$$\frac{\partial E}{\partial x_i} = p_i ((t - y_i)^2 - E)$$

这样，如果专家i的预测里目标值不远，那么p_i就会上升；反之则会下降

一个更好的代价函数会用到关于混合模型的一些概念，这里不做展开讲解，只做大致介绍。前面提到过最大似然的一种解释，就是做回归时其实网络是在做高斯预测。也就是说，假设每个专家的输出实际上是一个随机值，满足均值为y_i，方差为1的高斯分布，而管理器实际上是对每个高斯分布设定一个缩放比率（称为混合比mixing proportion），要求所有模型的缩放比率加和为1。所有专家的输出经过缩放再加和，得到的预测分布就

不是高斯分布了，现在的问题是要最大化目标值在该分布下的对数概率值。记$p(t^c|\mathrm{MoE})$为给定混合模型时样例c的目标值的概率，则有

$$p(t^c|\mathrm{MoE}) = \sum_i p_i^c \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}(t^c - y_i^c)^2\right\}$$

目标函数就是要最小化

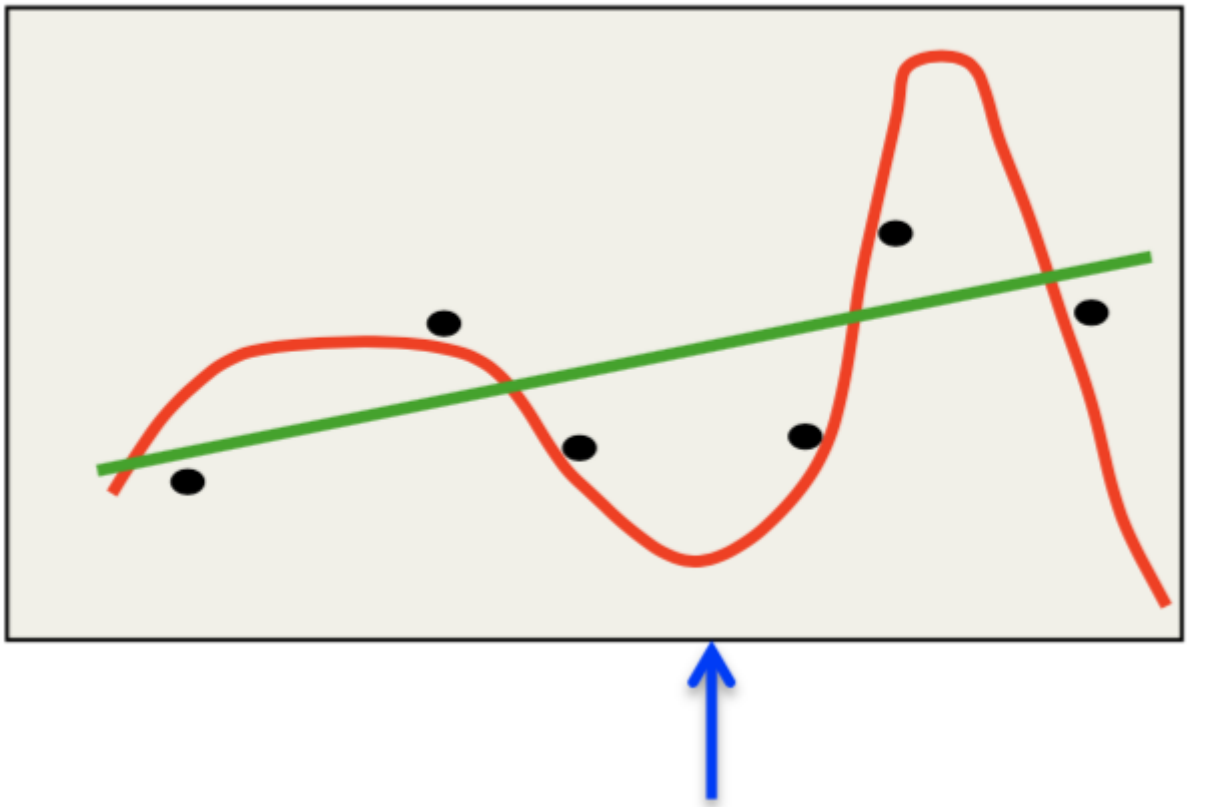
$$-\log p(t^c|\mathrm{MoE})$$

全贝叶斯学习

思想简介

在全贝叶斯学习中，要做的不是寻找一组最好的参数，而是在所有可能的参数上都去尝试，寻找一个完全的后验分布。也就是对每组可能的参数都期望计算一个后验概率密度。这种做法的问题是对于复杂的模型计算量太大，以至于寻找一个完整的后验分布是不太可能的。而且，得到完整的后验分布以后，还要使由各种参数组合得到的模型得出预测结果，然后将结果用后验分布概率做加权平均，这个计算量就更大了。但是全贝叶斯方法的一个诱惑点在于，即便是没有太多数据，复杂模型也可用

前面提到过过拟合的概念。使用比较小的数据集合训练一个复杂模型时，通常会出现这种现象。但是对这种现象发生的原因也可以解释为，是因为没有拿到参数上完整的后验分布。频率学派的观点是数据不够就应该使用简单的模型，这种说法只有当人们假设“拟合模型”就是“找到一组最好的参数”时才适用。如果能找到完整的后验分布，是可以避免过拟合的。当数据比较少的时候，完整的后验分布会给出比较模糊的结果，因为不同参数会给出非常不同的预测结果，所以后验分布也会显著不同。如果数据量变大，那么后验概率就可以只关注少数几组参数设置，预测结果可能会更集中



多组复杂模型的按后验概率加权平均的结果，可以达到更好的效果

上图中给出了一个例子。图中给出了六个点，红线是训练出的五次多项式，而绿线是训练出的一条直线。一般人都会认为是简单模型效果更好，例如蓝色箭头所指的位置红色模型的预测看着就很离谱。但是如果在五次多项式所形成的空间上有一个合理的先验分布，进而得到所有可能的后验分布（例如上图下部红色细线表明了其它可能的五次多项式，但是它们的后验概率值比较低），可以发现如果对蓝色的箭头部分取一个均值，那么预测结果就跟绿色模型的预测结果接近了。从贝叶斯学派的角度解释，采样到多少数据不应该影响你对模型的复杂度的先验信念。或者说，没收集到足够多的数据不应该让你认为事情应该更加简单

要使用神经网络做全贝叶斯学习，一个先决条件是神经网络的参数非常少，而且每个参数的取值也是有限的几个。这样，就可以在参数集上做一个网格搜索，得到所有参数组合对应模型的测试结果（似然）。这样，对每个参数组合，可以通过将它的似然和先验相乘再归一化，得到其后验概率。这种计算过程代价比较大，但是不需要梯度下降，也不需要担心局部最优解。在得到完整的后验分布以后，就可以在测试数据上做测试，此时有

$$p(t_{\text{test}}|\text{input}_{\text{test}}) = \sum_{g\in\text{grid}} p(W_g|D)p(t_{\text{test}}|\text{input}_{\text{test}}, W_g)$$

贝叶斯学习的实际应用

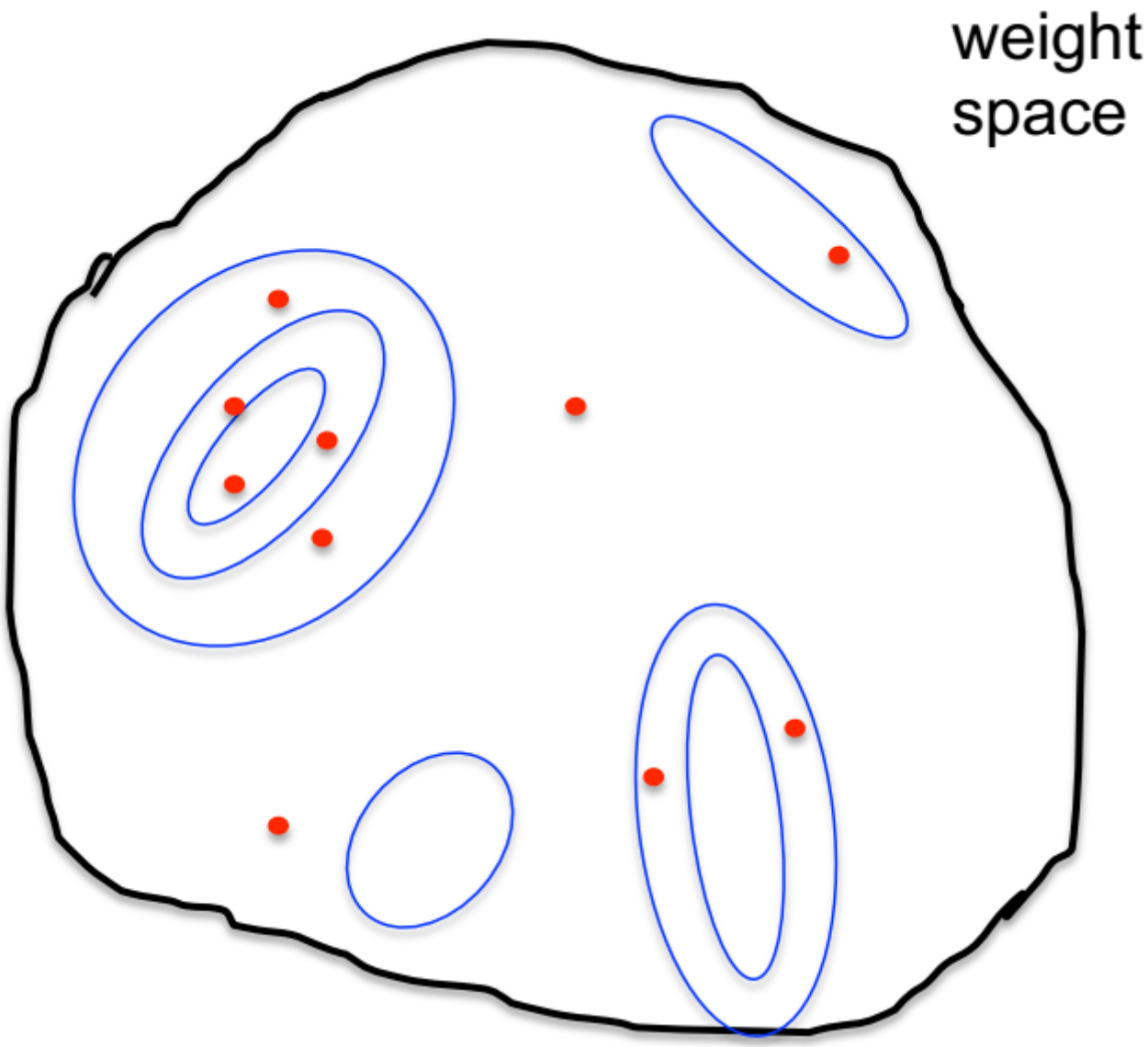
然而，神经网络里一般都有上百万个权重，因此理论上可行的网格法在实际中是不太可行的。但是，在有足够数据时，大部分参数向量都是不太可能适用的，所以只需要考察少部分的权重即可

尽管完全考察权重空间是不太可能的，但是可以退而求其次，根据权重向量的后验概率来对该空间进行抽样。对于下面的等式

$$p(y_{\text{test}}|\text{input}_{\text{test}}, D) = \sum_i p(W_i|D)p(t_{\text{test}}|\text{input}_{\text{test}}, W_i)$$

我们不再考虑所有的W_i，而是根据$p(W_i|D)$，即W_i的后验分布来对W做抽样

在传统的梯度下降算法中，一般是随机将权重初始化为一些比较小的值，然后在损失函数上沿着梯度方向下降，最后的结果无论是到达某个局部最小值点，还是陷入平原，还是因为跑得太慢导致我们没有耐心让它跑下去，其过程都是从某个初始点沿着某条确定路径到达某个终点。现在，引入采样的方法：初始化方法不变，只不过在每次更新权重以后，都随机加入一些高斯噪声，这样，权重向量不再会下降，而是开始随机游走，不过更倾向于损失函数值比较小的区域。下图给出了随机游走10000步以后的权重分布示意图



权重随机游走的分布示意图。最小值比较小的区域被访问的次数最多，其它局部最小点也会有一些访问。高损失值的区域由于比较大，也会被访问几次

如果加入高斯噪声的方法是正确的，并且让权重向量游走足够长时间，那么就可以得到权重空间后验分布的无偏抽样（结合上图，就是红点越多的地方后验分布越高）。这种方法称为**马尔科夫链蒙特卡洛**（Markov Chain Monte Carlo, MCMC）。事实上，有更有效的方法可以得到类似的无偏抽样，也就是说这种方法不需要权重向量游走时间太长

全贝叶斯学习可以用在mini batch SGD中。当在某个随机的mini batch上计算损失函数的梯度时，可以得到一个带有取样噪声的无偏估计。这个取样噪声可以提供给MCMC方法使用

Dropout

Dropout是一种~~新兴的~~防止过拟合的手段（论文发表于2012年，考虑此课也是2012年发布，因此这么说没有问题）。其核心思想是对每条测试用例都训练一个模型，然后在测试时候求这些模型的平均

首先来看一下，当有很多模型时，怎么将它们的输出组合起来。一种方法是将输出的概率求平均，例如模型A对三个类别的概率判断为0.3、0.2和0.5，模型B判断为0.1、0.8、0.1，那么平均以后这三个类别的概率分别为0.2、0.5和0.3。另一种方法是求几何平均值然后做归一化。保持前面两个模型做出的判断不变，此时平均以后未归一化的集合平均值分别为$\sqrt{0.03}, \sqrt{0.16}$和$\sqrt{0.05}$，这三个数的和为0.7968，因此归一化以后组合模型输

出的概率为21.74%、50.2%和28.06%。这里有一点值得玩味：对于求几何平均值的过程，如果某个模型输出的概率比较小，那么相当于它给其它模型的结果投了一张反对票

OK，现在来看一下怎么将输出平均的思想运用起来。假设现在要训练一个只有一个隐藏层的神经网络，它每次接受一个训练数据时，对每个隐藏层神经元以50%的概率将其抛弃掉，然后用剩下的神经元组成的隐藏层来训练这条数据。如果隐藏层有H个神经元，这意味着每次训练时都是从2^H个不同的结构里选择一个训练，而且所有这些结构都共享权重——也就是说，对每个隐藏单元，无论它是否被启用，在不同的结构里都是用相同的权重。这种方法称为**dropout**，可以看做是某种模型平均的方法。由于每个抽样到的模型都是用一个测试用例来训练，因此dropout也可以看做是baggings算法的一种极端情况。此外，所有模型共享权重，意味着每个模型都是被其它模型强烈地正则化。这种正则化方法比L1惩罚项或L2惩罚项强很多：后者是把权重推向0，而前者是把权重推向“正确的权重”

前面大概说明了dropout如何在训练时起作用，那么测试时应该怎么使用呢？取样多个模型然后对其输出求几何平均值太耗时间了，更有效的做法是在训练时保留所有隐藏单元，但是将输出权重减半。这实际上等价于计算所有2^H个模型的输出的几何平均值

（笔者注：讲义里和原始论文里并没有给出这一等价性的证明。在另一篇论文里给出了一些理论证明，但是过程比较复杂，这里不直接引用了）

如果网络中有多个隐藏层，那么可以把每层的dropout率设为0.5，然后在测试时将所有隐藏单元的输出权重减半。这种方法得到的网络称为**均值网络**（mean net）。或者，也可以仍然使用dropout随机丢弃一些节点，生成多个随机模型，然后对这些模型的结果做平均。这种做法的优势是可以在预测结果中引入不确定性。此外，输入层也可以使用dropout，不过要把保留率调高一些（丢弃率调低）。这种做法在Bengio组提出的降噪自动编码器（denoising autoencoder）中得到了应用，成效不错

Dropout机制在AlexNet中得到了应用，而且效果非常好。如果你的深度神经网络出现了明显的过拟合现象，使用dropout可以显著地降低错误率。实际上，任何需要提早结束机制来防止过拟合的网络，如果使用dropout来做过拟合，效果都会更好。如果网络没有过拟合，则应该把它变复杂然后引入dropout

对于dropout，有另一种解释方法，这种方法也是给Hinton以最原始灵感的方法。假设某个隐藏单元“知道”哪些隐藏单元在网络中，它就可以和它们在训练数据上协作。因此，训练时一个隐藏单元学会的更多是如何修正同伙的错误，而这种机制会导致由隐藏单元之间复杂的协作关系。如果数据发生了一些改动，这种协作可能不能很好处理。就好像是如果组织很多人去完成一项庞大、复杂的刺杀计划，这项计划就很有可能会失败，因为总会有一些意想不到的情况出现，也总会有人做出出人意料的举动，导致大家无法配合完成任务。好的做法是把整个计划分解成若干简单的 ，小的计划。这样如果意外发生，很多计划都会失败，但是总有一些团队仍然能成功。Dropout的原理类似，实际上是迫使隐藏单元和其它若干不同单元组成多种组合。这样，一方面每个隐藏单元自己要保证有足够的能力，另一方面，若干协作者一起也可以完成一些比较复杂的任务

Hinton神经网络与机器学习 11. Hopfield网和玻尔兹曼机

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2018/02/04/UTNN-11-Hopfield-Nets-and-Boltzmann-Machines/
-

Hopfield网

本讲开始介绍的神经网络和传统的前馈神经网络不同，它们的性质都是由某个全局的能量函数推演而来，因此称为基于能量的模型。Hopfield网络是其中最简单的一种，由二元阈值的单元组成，单元之间有循环连接。通常来讲，带有循环连接和非线性神经元的网络不太好分析，因为其行为比较难以捉摸：它们可以陷入稳态，可

以振荡，甚至可以陷入混沌（除非你知道无限精度的初始状态，否则无法预测不远的将来的状态）。不过，研究发现如果连接是对称的，那么就存在一个全局的能量函数。整个网络的每个二元配置有一个能量。所谓二元配置指的是，网络中的神经元的可能取值有两种，为每个神经元设置其中一个值。如果为阈值的决策规则设置了正确的能量函数，网络的能量会一直下降。如果持续使用该规则，最后会达到一个能量最小值

因此，能量函数的设置非常重要。某个配置的全局能量是若干局部贡献量的总和，而每个“贡献量”由连接权重和两个神经元的二元状态来决定。也就是说，能量函数的定义为

$$E = -\sum_i s_i b_i - \sum_{i<j} s_i s_j w_{ij}$$

注意能量越小越好，所以这里加了负号来做限制。s_i的取值是0或者1（或者在另一种Hopfield网里是-1或1）。有了二次能量函数的定义，每个单元就可以计算改变自身状态会改变多少能量。为此，需要定义**能量差**（energy gap）的概念，该值实际上就是关闭神经元i和打开神经元i两种情况下能量的变化量，即

$$\text{Energy gap} = \Delta E_i = E(s_i = 0) - E(s_i = 1) = b_i + \sum_j s_j w_{ij}$$

在Hopfield网中，寻找最小能量通常使用一种顺序迭代的方法：从某个随机状态开始，以随机次序每次更新某一个神经元：看它取哪个状态能量能变小，就让它处于哪个状态。注意这里不能使用节点同时决策的方法，因为如果同时并行更新状态，会导致周期为2的振荡出现。如果在并行更新状态的同时还加入随机的等待时间，也就是说，某个节点更新完状态以后不通知其它节点，而是等待随机长度的时间以后再进行下一次更新，那么这种振荡一般就不会出现

Hopfield认为，这种基于能量，能稳定在某个能量最小值点的模型适用于存储记忆力。使用上述的状态决策规则，可以把部分正确的记忆内容扩散到所有节点，清除掉一些错误的或者不全面的记忆。因此，可以在只知道一部分内容的情况下访问某一项内容。Hopfield网的鲁棒性很强，在硬件出现损坏的时候也可以正常工作，而阈值决策规则的作用此时就像是古生物学家从几根骨骼化石复原出一只恐龙一样

Hopfield网的记忆存储规则也比较简单。如果激活值使用1或者-1，那么将连接两单元的权重加上两神经元状态的乘积就可以，即

$$\Delta w_{ij} = s_i s_j$$

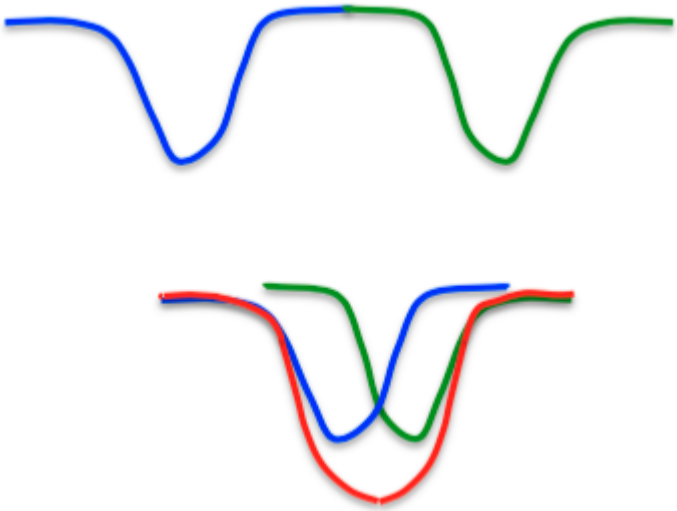
这种更新方法的好处是完全是在线更新，遍历过数据一遍就能得到结果，因为它并不是由误差驱动的，不需要将预测值与真实值比较，然后根据比较结果做微调。但是，这是一把双刃剑，带来的弊端是这种存储方法效率不高。如果状态是0-1值，更新方法略为复杂，但是计算起来还算简单，为

$$\Delta w_{ij} = 4\left(s_i - \frac{1}{2}\right)\left(s_j - \frac{1}{2}\right)$$

处理伪极小值

Hopfield网的存储能力是一个问题。对一个有N个节点，全连接的网使用前面的存储规则，网络的容量大概是$0.15N$的记忆内容。也就是说，如果有N个单元，每个单元有两种取值，在有两个单元的记忆开始出现混淆之前，可以存储的记忆数量大概是$0.15N$。每个记忆都是N个单元的一个随机配置，所以有N比特信息，因此Hopfield网能存储的信息总量是$0.15N^2$比特。也就是说，计算机使用了$0.15N^2$个比特来存储权重，因此这种分布式记忆+局部特征最小化的方法难以使计算机有效利用其存储能力：网络中一共有N^2个权重和偏置，存储M个记忆以后，每个连接权重的取值是$[-M, M]$之间的一个整数（因为假设初始状态是-1或1，每存入一个记忆以后权重增1/减1），因此存储所有权重和偏置理论上需要的比特数是$N^2 \log(2M+1)$

所以是什么限制了Hopfield网的能力呢？每次要记住的配置时，都希望可以创造一个新的能量最小值。如下图所示，对某两个二元配置，理想状态是学到两个最小值点，一个是蓝色的，一个是绿色的。不过实际上，由于两者很近，因此最后会发生混淆，两者合并成一个红色的，更小的最小值点，但是不能对这两个二元配置作区分（这部分参考了知乎问答：关于Hopfield神经网络 ）。这种相邻最小值点的融合现象称为“伪最小值点”（spurious minima）



伪最小值点的示意

Hopfield、Feinstein和Palmer等人提出了一种应对策略：让网络从一个随机初始状态开始，做一些“忘却”（unlearning）操作。也就是说，不管它处于什么状态，反着用前面提到的存储规则。以前面的图为例，假设现在处于红色的峰值点，经过一些“忘却”以后，就可以脱离这个点，得到了两个独立的最小值，进而增大记忆量。这种方法实践中管用，但是发明者没有做出比较好的原理分析。Francis Crick和Graham Micherson认为忘却法与快速眼球移动睡眠（REM睡眠）的过程很像（具体解释略去）。不过，关于忘却法的另一个问题是，应该忘掉多少东西？实际上，忘却的内容也是使用数据拟合模型这一过程的一部分。如果使用最大似然法拟合这个模型，忘却的过程会自然而然地出现，因此关键问题是提出一个合适的代价函数

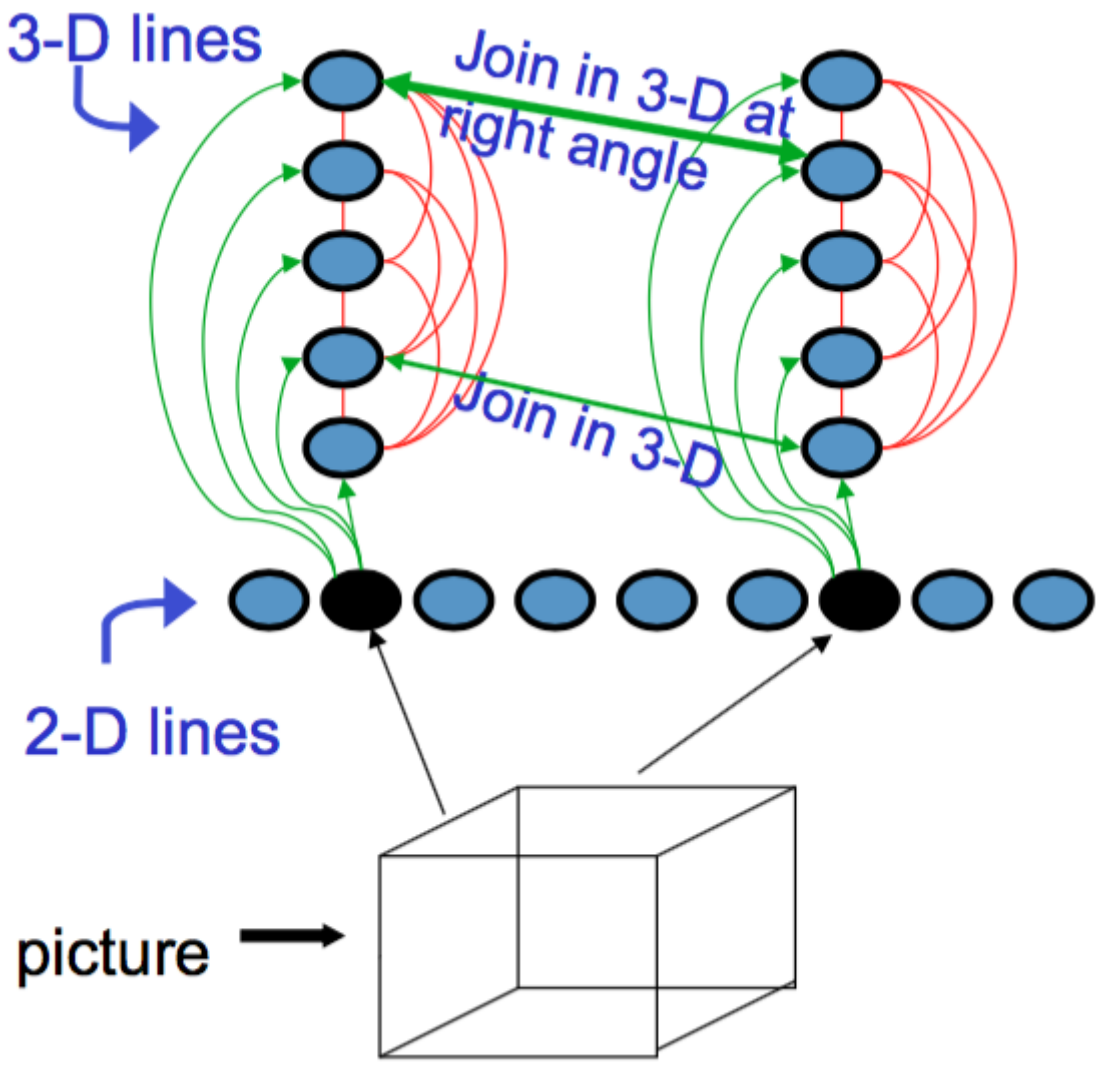
在具体介绍忘却法的代价函数之前，先看一看物理学家是如何增强Hopfield网能力的。Elizabeth Gardner提出了一种方法，就是不像之前那样只遍历一遍数据集就存一个向量，而是循环遍历多次训练集。这样，牺牲的是在线学习能力，得到的是更有效的存储。整个训练过程有点像感知机的收敛过程：让整个网络先进入到某个记忆状态，然后单独来看每个单元，判断给定所有其它单元状态的情况下，该单元是否展现了想要的状态。如果结果为“是”，那么让其输入的权重不变；否则按照收敛过程指定的权重更新规则更新权重。这种方法被统计学家称为“伪似然法”。不过Hopfield网中的权重都是对称的，因此对每个权重需要获得两组梯度然后求均值

带有隐藏单元的Hopfield网

Hopfield网包含了两个核心想法：首先，如果网络中的神经元只有两个可能取值，而且连接是对称的，那么就可以找到一个局部的能量最小值；其次，局部的能量最小值可能对应于某些记忆。不过，有另外一种寻找局部最小值的方法，就是不再让网络存储记忆，而是让它对输入构造一些解释。此时，输入由一些可见单元表示，解释由隐藏单元构成。也就是说，此时对输入的解释是隐藏单元上的一个二元配置，整个系统的能量表示这个解释有多差，因此目标是让隐藏单元进入比较低能量的状态

举个例子，当你看到图片里的一条二维的线，它应该是由什么样的三维物体的边缘所投影出来的？实际上，它可能对应了很多不同的三维物体，因为投影时边缘两端的深度信息都已经丢失了。而且，如果若干物体都放在一起，它们之间存在相互遮挡，那么你能看到的其实也只是一条线

假设现在有若干张图片，每张图片里面有若干条线。可以建立一组神经元，每个神经元对应图中的一条线。由于对任意一张图片，很难包含所有的线，因此对给定的一张图片，只有若干几个神经元能够被激活。这些神经元是输入神经元。然后，对所有三维物体的线或边缘，也对应建立神经元。由于一条二维线可以对应若干三维物体，因此对某个被激活的输入神经元，可以与其所有对应的隐藏神经元建立联系



使用带隐藏单元的Hopfield网学习二维线段与三维物体边缘联系的示意图

上图给出了一个示意图。这里，输入神经元与隐藏神经元之间的绿色连线表示两者存在可能的对应关系，每个连接都共享相同的权重。由于一条二维线段只对应于一种可能的三维物体，因此这些隐藏神经元之间彼此还存在竞争关系，以红线表示

接下来，要建模的是不同输入单元对应的两组隐藏单元之间的关系：图片中两条二维直线相交，通常意味着两个对应的三维物体在交点处有同样的深度，因此可以建立两个隐藏单元之间的联系（上图中连接两隐藏神经元的细绿色箭头）。除此以外，如果三维物体之间不仅相交还相互垂直，那么就给一个比较强的连接。这样，在建立了隐藏单元与输入单元的连接，以及隐藏单元之间的连接以后，就可以对某个输入图像做出解释

如果决定使用低能量的状态来对输入做出解释，相对应的会带来两个问题：首先，如何避免隐藏单元陷入比较差的能量函数的局部最小值？第二个更难的问题是，如何学习出这些权重，尤其是体系结构中没有监督器的情况下？

使用随机单元以改进搜索

Hopfield网做出的决策总是向着降低网络能量这一目标前进（如果不改变单元状态，那么能量保持不变），因此，很难脱离局部最小值。解决方法是加入随机噪声，这样就可以从比较差的极值点离开，尤其是极值点比较

浅的时候效果更明显。这里要加入的不是某个固定数量的噪声，更有效的策略是先加入很多噪声，然后对空间做一个粗略探索，找到总体来讲不错的区域，然后减少噪声。这样一来，噪声很大的时候，就可以翻过比较大的壁垒；减小噪声的时候，网络就开始聚焦于附近的最小值。如果噪声降低得很慢，系统就会陷入一个比较深的最小值，这种方法称为**模拟退火法**

模拟退火法背后的原理是，在物理系统或者使用能量函数的模拟系统中，温度会影响转移的概率：

- 当系统的温度比较高时，粒子向上上升的概率低于向下降的概率，但是低得不多。这样，更容易翻过壁垒，但是即便达到比较深的极小值也不太容易待得住
- 当系统的温度比较低时，粒子移动的概率总体都不大，但是上升的概率远远低于下降的概率。理论上讲，所有粒子最后都会陷入比较深的最小值，但是如果持续在低温环境下运行，粒子也难以从局部最小值走出来

综合以上两种状况，比较好的组合策略是从高温开始，慢慢降低温度

因此，在Hopfield网中引入噪声的做法是将二元阈值单元转化为二元随机单元，让其做出有偏的随机决策，而噪声的数量由“温度”来控制。提高噪声的数量等价于降低所有配置之间的能量差。具体的形式为

$$p(s_i = 1) = \frac{1}{1 + e^{-\Delta E_i / T}}$$

这个式子与logistic回归中用到的等式类似，只不过能量差ΔE_i缩小了温度数量那么大的倍数。如果温度很高，那么指数项接近于0，分母接近于2，单元被激活的概率大概为0.5。温度下降时，取决于ΔE_i的符号，单元或者是稳定地保持在开的状态，或者是稳定在关的状态。当温度降低到0时，退化为Hopfield网，单元是否被激活由ΔE_i的符号决定，此时随机性褪去，单元退化为原始的二元阈值单元。能量差的定义仍然为

$$\Delta E_i = E(s_i = 0) - E(s_i = 1) = b_i + \sum_j s_j w_{ij}$$

模拟退火是一种有效避免陷入局部最优，以改善搜索效果的方法，对玻尔兹曼机的提出也产生了影响。但是深入了解它会分散对玻尔兹曼机的注意力，因此这里不再详述。从这里开始，每个单元都是温度为1的二元随机单元，是能量差的标准logistic函数

要了解玻尔兹曼机的学习过程，就要理解热平衡的概念。大多数人都认为，到达热平衡时，系统稳定下来，不再发生任何变化。热平衡大概是这个意思，但是这不意味着每个单元的状态都稳定了。实际上，每个单元的状态还会变化，稳定下来的是配置上的概率分布。我们称稳定在某个特定分布上的概率分布为**稳态分布**（stationary distribution），稳态分布由系统的能量函数决定。在稳态分布中，任何配置的概率正比于其负能量。理解热平衡的一种直观方法是，假设很多相同的系统构成了一个巨大的混合体，每个系统都有完全相同的能量函数，比如，有一个巨大的随机Hopfield网组成的集合，每个Hopfield网的权重都相等。在这个混合体中，可以定义某个配置的概率为系统中处于这个配置的系统的比例。然后，可以以任何分布作为初始情况，例如可以让所有节点的配置都相同，或者对每种可能的配置，都有相同数量的网络以该配置为初始配置（这样就是均匀分布），然后使用随机更新规则（选择一个单元，检查能量差，根据能量差随机决定是打开还是关闭，然后选择下一个单元，以此类推），最后会达到一个稳态，即系统中每种配置的比例是固定的（如果连接是对称的，最后肯定会达到这个状态）

对此，可以有一个类比。假设在拉斯维加斯有个巨大的赌场，里面有多于$52!$个发牌者，每个发牌者拿到的牌都是未开封的，由同一家厂商制造，初始顺序按照标准顺序排好。然后，所有发牌者开始随机洗牌，而且不玩那种把牌变回原来顺序的那种把戏。经过少数几次洗牌以后，可能黑桃K还在黑桃Q上面，也就是说，任意牌堆里还会有一些牌没有忘记自己的初始位置。不过，在经过长时间洗牌以后，所有牌堆的所有牌都会被打乱顺序，此时牌的$52!$种排列出现的数量相等，并且每种排列的数量不会随着洗牌的过程而变化

（简而言之，热平衡不是系统内的粒子不动，而是粒子还在动，但是整个系统呈现稳定状态）

玻尔兹曼机如何对数据建模

带有隐藏单元的随机Hopfield网也称为**玻尔兹曼机**，这种模型擅长对二进制数据建模。也就是说，给定一组二进制的训练向量，可以使用隐藏单元来训练一个模型，这个模型可以对任意二进制向量计算一个概率。这样，可以判定其它二进制向量是否来自于相同分布。一个实际的例子是，假设有一个很大的词表，那么每个类型的文档可以使用二进制向量表示（每一位代表该单词是否出现），进而判断测试数据应该属于哪个类别。也可以使用玻尔兹曼机监测复杂系统，检测异常行为。所有这些应用的背后都是贝叶斯定理：假设有若干模型，每个模型对应一个分布，那么可以使用贝叶斯定理计算给定观察数据对应模型的概率，即

$$p(\text{Model}_i|\text{data}) = \frac{p(\text{data}|\text{Model}_i)}{\sum_j p(\text{data}|\text{Model}_j)}$$

使用贝叶斯定理时，需要计算模型产生数据的概率。这种做法有两种，一种是先根据隐藏单元的先验分布生成某些隐变量的状态，然后用隐变量生成二进制数组。这种模型称为因果模型（causal model）。在这种模型里，隐藏单元的先验分布一般是独立的，因此每个单元被激活的概率（如果是二元状态）只取决于各自的偏置量。得到隐藏单元以后，就可以使用隐藏层到输出层的权重和偏置，加上logistic非线性变换得到输出层每个单元被激活的概率。因此，对于因果模型，得到某个向量$\mathbf{v}$的概率是

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{h})p(\mathbf{v}|\mathbf{h})$$

其中$\mathbf{h}$是任意隐藏单元

不过，玻尔兹曼机不是因果模型，而是基于能量的模型（……），一切真理都定义为可见单元和隐藏单元联合配置的能量，而将能量与概率相联系又有两种方法：其一是让$p(\mathbf{v},\mathbf{h}) \propto \exp -E(\mathbf{v},\mathbf{h})$，其二是定义为在多次更新所有随机二元单元，达到热平衡以后，网络得到该联合配置的概率。这两种方法本质是一样的。记$E(\mathbf{v},\mathbf{h})$为可见单元配置为$\mathbf{v}$，隐藏单元配置为$\mathbf{h}$时网络的能量，有

$$-E(\mathbf{v},\mathbf{h}) = \sum_{i\in\mathbf{v}} v_i b_i + \sum_{k\in\mathbf{h}} h_k b_k + \sum_{i<j} v_i v_j w_{ij} + \sum_{i,k} v_i h_k w_{ik} + \sum_{k<l} h_k h_l w_{kl}$$

这里b_i是单元（无论是隐藏单元还是可见单元）i的偏置值，w_{ij}是单元i和j之间的权重。对同在隐藏层/可见层的单元，需要避免重复计数

由于可见层任一配置的概率$p(\mathbf{v},\mathbf{h}) \propto \exp -E(\mathbf{v},\mathbf{h})$，因此通过上述方法得到$-E(\mathbf{v},\mathbf{h})$以后可以使用类似softmax的方法算出概率值

$$p(\mathbf{v},\mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u},\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}}$$

其中分母也被物理学家称为“划分函数”（partition function）

类似地，可以求出可见单元某个配置的概率

$$p(\mathbf{v}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u},\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}}$$

如果玻尔兹曼机对应的网络太大，那么隐藏层+可见层所有单元可能的配置数会非常多，因此计算完整的划分函数值不太现实。此时可以使用MCMC方法来抽样，即从某个随机的全局配置开始，随机选取某个单元，根据

能量差随机改变状态。这样持续下去，等到马尔可夫链达到稳定分布，就得到了模型的一个抽样，这个抽样的概率$p(\mathbf{v},\mathbf{h})$正比于$\exp\{-E(\mathbf{v},\mathbf{h})\}$

另一个问题是，给定一个数据向量，怎么从隐藏配置的后验分布上获得一个样本？由于隐藏配置的集合也是指数量级的，因此仍然使用MCMC方法。与前述过程唯一不同的是，这是要保证可见单元不变，对应于给定的数据向量，只改变隐藏节点的状态。这时，可以理解为每个隐藏配置都是对观察数据的可见配置的一个解释，更好的解释能量更低

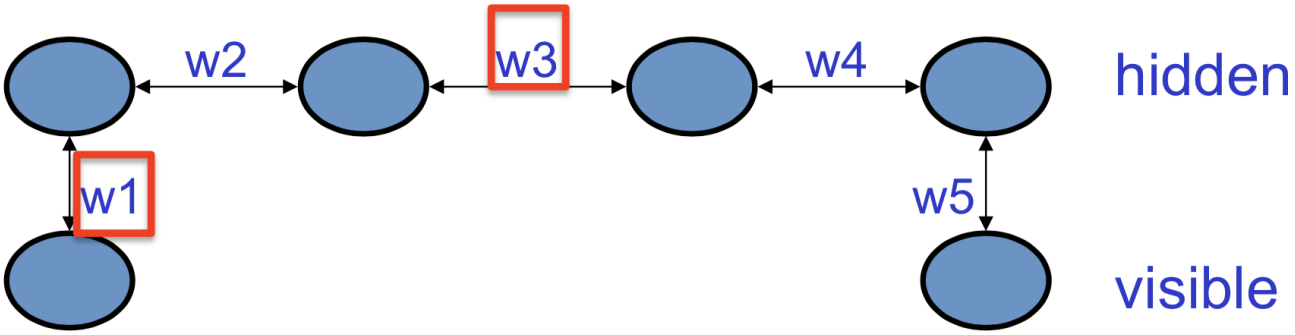
Hinton神经网络与机器学习 12. 受限玻尔兹曼机

（视频和讲义中的第二节“More efficient ways to get the statistics”由于是可选章节，被略去了）

玻尔兹曼机学习

玻尔兹曼机学习算法不是一个有监督学习算法，没有期望输出。给定一个输入向量（实际上更应该是看做输出向量），算法试图构造一个模型，模型会为训练集中每个二进制向量计算一个概率，而学习的目标是让这些概率的乘积最大化，也就是让概率的对数和最大化。此外，这个概率也是通过如下方式得到训练集里所有N条训练数据的概率：令网络在没有外部输入的情况下进入稳定状态，对可见向量做一次抽样，如此重复N次

玻尔兹曼机的学习过程是比较难的，这里以一个简单模型来做一个示例。模型结构如下图所示



一个简单的玻尔兹曼机示意图

这里，四个隐藏单元链式相连，两端各自接一个可见单元。给这个网络的训练集是(0, 1)和(1, 0)，也就是说，我们希望两个可见单元处于各自相反的状态——这就要求所有权重的乘积为负。因为，比如说所有权重都是正的，让w_1为正就会让第一个隐藏单元为正，这会使第二个隐藏单元也为正，以此类推，因此第四个隐藏单元会让另一个可见单元也为正

（这里我试图展开一点来讲这个事情。由于前面学得不是太扎实，因此可能存在偏差。如果我的想法的确有问题，请及时纠正我！

在前一讲，有一个给定可见单元配置$\mathbf{v}$和隐藏单元配置$\mathbf{h}$时整个网络能量的计算公式，为

$$-E(\mathbf{v},\mathbf{h})=\sum_{i\in\mathbf{v}}v_ib_i+\sum_{k\in\mathbf{h}}h_kb_k+\sum_{i<j}v_iv_jw_{ij}+\sum_{i,k}v_ih_kw_{ik}+\sum_{k<l}h_kh_lw_{kl}$$

这里没有截距，可见单元之间也没有权重连接，因此可以简写为

$$-E(\mathbf{v},\mathbf{h})=\sum_{i,k}v_ih_kw_{ik}+\sum_{k<l}h_kh_lw_{kl}$$

为了让这个式子尽可能大，给定v_1和w_{11}都为正，那么h_1应该为正。由于w_{12}也为正，那么h_2为正才能使$h_1h_2w_{12}$为正。这样推导下去，在v_1为正且所有w_{ij}都为正的情况下，v_2也应该为正才能是能量最大化）

如果把其中某个权重变为负数，那么就可以得到两个隐藏单元之间的反相关关系。这意味着学习w_1的时候要知道其他权重，例如要想知道如何修改w_1就要知道w_3的值，因为w_3为正的时候对w_1的修改和w_3为负的时候对w_1的修改是不一样的。不过这种看似复杂的情形实际有一种很简单的学习算法，而且这种算法只需要局部信息。实际上，任意权重对其它所有权重和数据要了解的内容都包含在两个相关性的差里。或者换种方式说，如果将玻尔兹曼机给某个可见向量$\mathbf{v}$的概率的对数值$\log p(\mathbf{v})$，对权重w_{ij}求偏导数，其等于如下两者的差

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}}=\langle s_is_j\rangle_{\mathbf{v}}-\langle s_is_j\rangle_{\text{model}}$$

其中

- $\langle s_is_j\rangle_{\mathbf{v}}$是将可见单元限制为$\mathbf{v}$，当系统进入热平衡状态时，状态乘积的期望值
- $\langle s_is_j\rangle_{\text{model}}$是不对可见单元做限制，当系统进入热平衡状态时，状态乘积的期望值

将第一项命名为$\langle s_is_j\rangle_{\text{data}}$，可以得到

$$\Delta w_{ij}\propto\langle s_is_j\rangle_{\text{data}}-\langle s_is_j\rangle_{\text{model}}$$

第一项说明表示数据时，增加权重的量等比于活动单元状态的乘积；而第二项则用来加以控制，当你对模型分布做采样时，它降低权重的量等比于两个单元一起被激活的频率。可以如此理解：第一项对应于Hopfield网的记忆存储，第二项对应于避免陷入伪最小值点——这条规则其实是在说明“忘却”（unlearning）过程是如何进行的

那么为什么这个偏导数的形式如此简单呢？热平衡是全局配置的概率是能量的指数函数，即$p\propto e^{-E}$。所以当达到热平衡时，可以得到概率对数与能量函数的一个线性关系。由于能量函数与权重也是线性关系，因此权重和概率对数是线性关系，所以有

$$-\frac{\partial E}{\partial w_{ij}}=s_is_j$$

实际上，系统最后稳定到热平衡的过程就是传播权重信息的过程。这里不需要一个显式的反向传播，而是要分成两个阶段：有数据情况下的稳定阶段和无数据情况下的稳定阶段。不过网络在这两个阶段的行为基本类似，只是边界条件不一样（如果用反向传播算法，正向传播的计算过程和反向传播的计算过程完全不同）

为什么要有一个无数据情况下的稳定阶段（这里称为负向阶段）？前面提到这就像Hopfield网避免陷入伪最小值点的忘却过程，不过这里稍作进一步解释：对于某个可见向量，其概率可以写为

$$p(\mathbf{v})=\frac{\sum_{\mathbf{h}}e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u}}\sum_{\mathbf{g}}e^{-E(\mathbf{u},\mathbf{g})}}$$

看分子（正向阶段，注意这里“正向”对应的是英文的“positive”），其实是降低求和项里能量大的项，使得系统在可见单元已经为$\mathbf{v}$的情况下，到达热平衡时可以找到适合$\mathbf{v}$的隐藏向量$\mathbf{h}$，从而使$\mathbf{v}$的能量足够小。调整权重的过程实际上是让分子变大，能量变小的过程。学习的第二个阶段是对分母（划分函数）做同样事情的过程。对于其它能让系统能量比较低的全局配置（其实是可见状态和隐藏状态的组合），算法试图提升它们的能量。也就是说，正向阶段是增大分子，负向阶段是减小分母

为了将这些学习规则跑起来，需要收集统计信息，包括正向统计信息和负向统计信息两种。一种比较基本、效率低下的方法是：

- 在正向阶段，将可见单元设置为数据向量，然后将隐藏单元设为随机的二元状态。每次更新网络中的一个隐藏单元，直到网络达到温度为1的热平衡。然后，对每对相连的单元，抽样求$s_i s_j$的期望。将这个过程应用于训练集里的每条数据，然后求均值
- 在反向阶段，不做任何限制，将所有单元设为随机二元状态，然后每次更新一个单元，直到网络达到温度为1的热平衡。然后，对每对相连的单元，抽样求$s_i s_j$的期望。重复以上过程很多次

现在的问题是，在反向阶段，究竟要重复前述操作多少次（后面大概说了一下，但是最后也没说清需要采样多少次……）

以上介绍的是一种比较低效的收集统计信息的方法。更高效的方法是可选内容，这里不做介绍

受限玻尔兹曼机（RBM）

受限玻尔兹曼机（RBM）对普通玻尔兹曼机施加的限制有两点：一是隐藏层只有一层，二是隐藏单元之间没有连接（所以画出来的图是一个二部图），因此给定可见单元的情况下，可以很容易得到平衡分布（原因是隐藏节点相互独立）。这使得RBM的学习过程和推断过程都很容易——给定可见单元的向量以后，一步就可以达到热平衡。即给定可见单元向量以后，可以计算出每个隐藏节点h_j被激活的概率，进而计算$v_i h_j$的期望值（其中v_i是某个可见节点）。其中h_j被激活的概率为

$$p(h_j = 1) = \frac{1}{1 + \exp\{-\left(b_j + \sum_{i \in \text{vis}} v_i w_{ij}\right)\}}$$

由于该值独立于其他所有隐藏节点，因此可以把这个概率并行求出。如果训练集里有多条数据，可以使用Tieleman在2008年给出的PCD算法。算法仍然分为两个阶段

- 正向阶段：对每条数据，将可见单元设置为给定的向量，计算所有隐藏节点和可见节点对的$v_i h_j$。对所有相连的节点对，$\langle v_i h_j \rangle$是mini-batch上所有数据的平均值
- 负向阶段：准备一些“幻想粒子”（fantasy particle），这些幻想粒子实际上是一些全局的配置。然后，对每个幻想粒子，做几次如下的“交替并行更新”：每次更新权重以后，对幻想粒子也做一点更新，把它们往平衡态拉一点。对每对连接的单元对，求所有幻想粒子的平均$v_i h_j$，得到负统计数据

这个算法能达到不错的效果，而且允许RBM构建密度比较好的模型。不过，还有一种更快的学习算法。在此之前，先看一种效率比较低的学习算法：在最开始的时候，将可见单元设为一条训练数据，称此时为t_0时刻。给定可见单元以后，更新隐藏单元，可以算出t_0时刻状态乘积的期望值，记为$\langle v_i h_j \rangle^0$，与$t_0$对应。得到新的隐藏单元以后，返回来并行更新所有可见单元，得到t_1时刻的可见单元，再并行更新所有隐藏单元，得到$\langle v_i h_j \rangle^1$，从而完成一次重构过程。在漫长的往复更新以后，在第∞时刻，系统达到热平衡，按照上面的记法，可以得到$\langle v_i h_j \rangle^\infty$。此时得到了幻想粒子。接下来学习规则就比较简单了，为

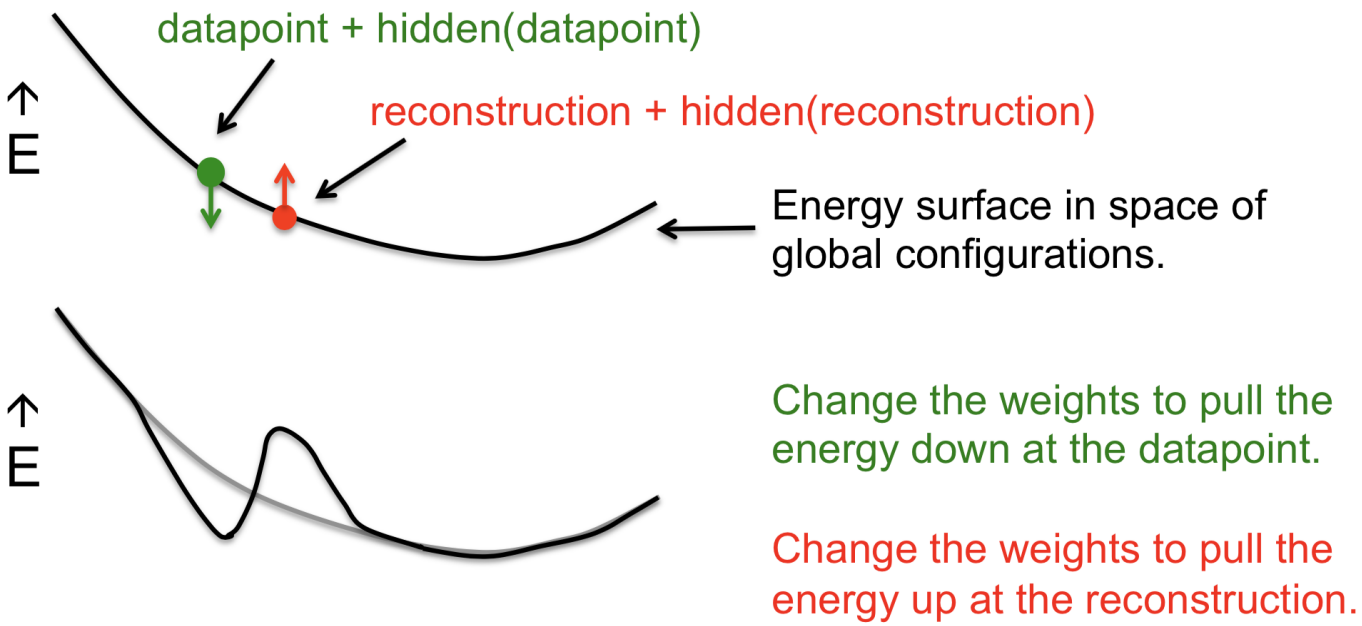
$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty)$$

这个算法的问题是要运行太久的时间才能达到热平衡，得到幻想粒子。令人惊愕的是，实际上只需要走一步（更新两次隐藏单元和一次可见单元），就可以获得需要的统计值，然后使用如下方法更新权重

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1)$$

这种做法肯定不是最大似然法，因为负统计数据是错的，不过这种方法在实践中可用。这是为什么呢？马尔可夫链会以给定数据为起点开始向着平衡分布游走，因此经过少数几步以后就可以看到游走的方向。如果初始权重不好，走到平衡态实际就是浪费时间。我们不用等系统进入平衡态就能知道如何修改权重来防止它走到错误

的方向，要做的就是降低在完整一步之后生成“虚构（confabulation）”的概率，提升数据的概率，这样模型就不会走错了。一旦数据和模型走了完整一步以后达到的区域有相同的分布，学习过程就会停止



对比差异学习算法示意图

这里给出了一个全局配置空间中的能量表面示意图，用以解释前面介绍的“对比差异学习算法”（contrastive divergence learning，又称为对比散度学习）。绿色的点是一个数据点，也就是可见向量和通过随机更新隐藏单元得到的隐藏向量。从这个点开始，让马尔可夫链经过一个完整的更新步骤，可以得到新的可见向量和相对应的隐藏向量，也就是红色的重构点。然后，修改权重会降低数据点的能量，增大重构点的能量，得到一个经过局部拉伸的新表面。可以看到这样实际上是在数据点创建一个能量最小值，同时让远离数据点的部分保持不变

因此，这种取巧做法对远离数据点的部分会失效，我们需要小心那些里数据空间很远，但是模型很喜欢的区域。这些低能量的孔洞可以导致正则项变得很大，而且如果使用上述取巧法无法感知到。如果使用持久粒子（即它们的状态会被记忆，而且在每次更新以后会被多次更新），这些粒子会找到这些孔洞，掉进去，学习过程变成补洞的过程。所以，更平衡的方法是先使用一些小的权重，然后以CD1算法开始（在一次完整的步骤之后就获取负统计数据），过一段时间，变成使用CD3，再过一段时间，使用CD5……以此类推

RBM学习示例

本讲介绍如何使用RBM学习一系列特征，以重构手写数字2的图像。输入图像是16像素 x 16像素的，有50个二值的隐藏单元来学习特征。接收到数据的时候，先使用权重和节点之间的连接激活特征（也就是对每个隐藏神经元，使用随机决策来看它是否应该被激活），然后使用这些特征重构数据（对每个像素，判断是否应该为1），最后使用重构数据重新激活特征。当网络观察原始数据时，增加活跃像素和活跃特征探测器（隐藏单元）之间的权重，降低原始数据的全局配置的能量；当网络观察重构数据时，减小活跃像素和活跃特征探测器之间的权重，增加重构数据的能量。在学习刚刚开始，权重都随机（使用随机权重来打破网络的对称性）的时候，重构的能量几乎肯定比原始数据的要低，因为重构数据是网络在给定隐藏单元激活状态时倾向在可见单元上产生的值，而它肯定会根据能量函数产生低能量的数据。学习的过程就是降低原始数据能量，提高重构数据能量的过程 （后面学习过程和效果略。大致来说，对没见到的图像，网络可以给出一个不错的重构数据。由于网络把什么都看作是数字2，如果给进一个3，它也会试图重构出一个像2的图像，保持顶部形状不变，但是中间弯折的部分会处理出比较模糊的效果）

使用RBM做协同过滤

（协同过滤的背景略）

可以训练一个“语言模型”，也就是把每条数据写成一个三元组组成的字符串，形式为`<用户名 电影名 评分>`。这样一来，推荐系统就是要根据元组的第一、二个元素猜测第三个元素是什么。因此可以将每个用户编码成一个向量特征，每个电影也编码成一个向量特征。将这两个向量特征求内积，就可以得到预测的评分（甚至都不用做softmax）。这种模型实际上称作矩阵分解模型，因为如果把用户向量按行排列，电影向量按列排列，就会得到两个矩阵。如果将这两个矩阵相乘，就会得到所有的得分

另一种方法是使用RBM来解决这个问题，但是看上去不那么直观，需要绕一下。假设把每个用户看成是一条训练数据，表现为由电影评分组成的向量，而对每部电影建立一个可见单元，每个单元有五种可能取值（因此可见单元不再是二元的，而是五路softmax得到的结果）。因此网络里有约18000个可见单元，此外还有约100个隐藏单元，每个单元只有0或1两种取值，还带有一个偏置项。由于用户不可能看过所有电影，因此模型要做的事情是预测缺失值（对用户没看过的电影，猜测用户会打多少分）

这里面的隐忧是，网络里如果有18000个可见单元，又要求每个可见单元都与所有隐藏单元全连接，网络的权重数太多了。但是，由于每个用户点评过的电影都只有少数若干个，因此实际可以对每个用户维护一个不同的RBM。这些不同的RBM的可见单元不同（对应于用户评过分的电影），但是隐藏单元都共享，而且从每个隐藏单元到每部电影的权重也被所有为这部电影打过分的用户共享。训练的时候，先使用前面的CD1，过一段时间改为CD3，然后是CD5和CD9。使用RBM解决协同过滤取得的效果跟矩阵分解的总体效果差不多，但是两者犯的错误很不一样。因此同时训练两个模型，然后对模型求平均可能会收效很大

Hinton神经网络与机器学习 13. 深度信念网络

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2018/02/17/UTNN-13-Deep-Belief-Nets/

反向传播优劣论

反向传播算法在70年代到80年代被不同的研究组独立发现了几次，不过追根溯源可以到60年代。在1969年，Bryson和Ho在控制论领域发明了反向传播算法的线性版本。Paul Werbos意识到可以将这种算法推广到非线性范畴，并在1974年发表的论文中提出了可能是反向传播的第一个正式版本。1981年，Rumelhart、Williams和Hinton在不知道Werbos的工作的情况下也发明了反向传播算法，但是经过实验没有达到预想的效果，所以就先放弃了。1985年，David Parker和Yann LeCun分别又各自发现了这种算法，Hinton也重新进行了尝试，发现效果不错。1986年，Hinton等人发表了一篇论文，并在文中提出了一个令人信服的例子以说明算法的能力

尽管反向传播算法让人们看到了学习多层非线性特征探测器的希望，但是在90年代末，大部分机器学习界的研究人员都放弃了这种算法。尽管它仍然还受心理学家和实际应用（例如构建信用卡欺诈检测模型）的青睐，但是在机器学习领域，大部分人都认为SVM要更好一些。对此有一些比较流行的解释，比如神经网络不能很好地利用多层非线性特征（CNN是个例外）、RNN和深度自动编码器效果不好，等等。而SVM不需要使用者有太多经验，能产生可复现的结果，还有更好的理论支持，因此当红一时

现在来看，90年代末反向传播不受重视的原因有如下几点：当时的计算机太慢，计算能力太弱，有标签的数据集太少。神经网络太小，也没有得到合适的初始化（一般会初始化太小），因此梯度在反向传播的过程中很容易消失。不过为什么对视觉和语音这样的问题神经网络能获得成功呢？关键问题是，机器学习的任务可以分为两种类型，一种更像统计学界人们研究的问题，另一种是人工智能界人们研究的问题：

	统计问题	人工智能问题

	统计问题	人工智能问题
数据维度	低，通常少于100维	比较高
噪声强度	噪声多，如何处理很重要	噪声不是主要问题
数据的结构	不多，可以被简单模型描述	数据中有大量结构，而且比较复杂
主要问题	将真实结构从噪声中剥离。这种问题不适合非贝叶斯神经网络处理，而应该试用SVM或高斯过程	找到一种复杂结构的表示方法。适用于反向传播，因为多层神经网络的计算能力很强

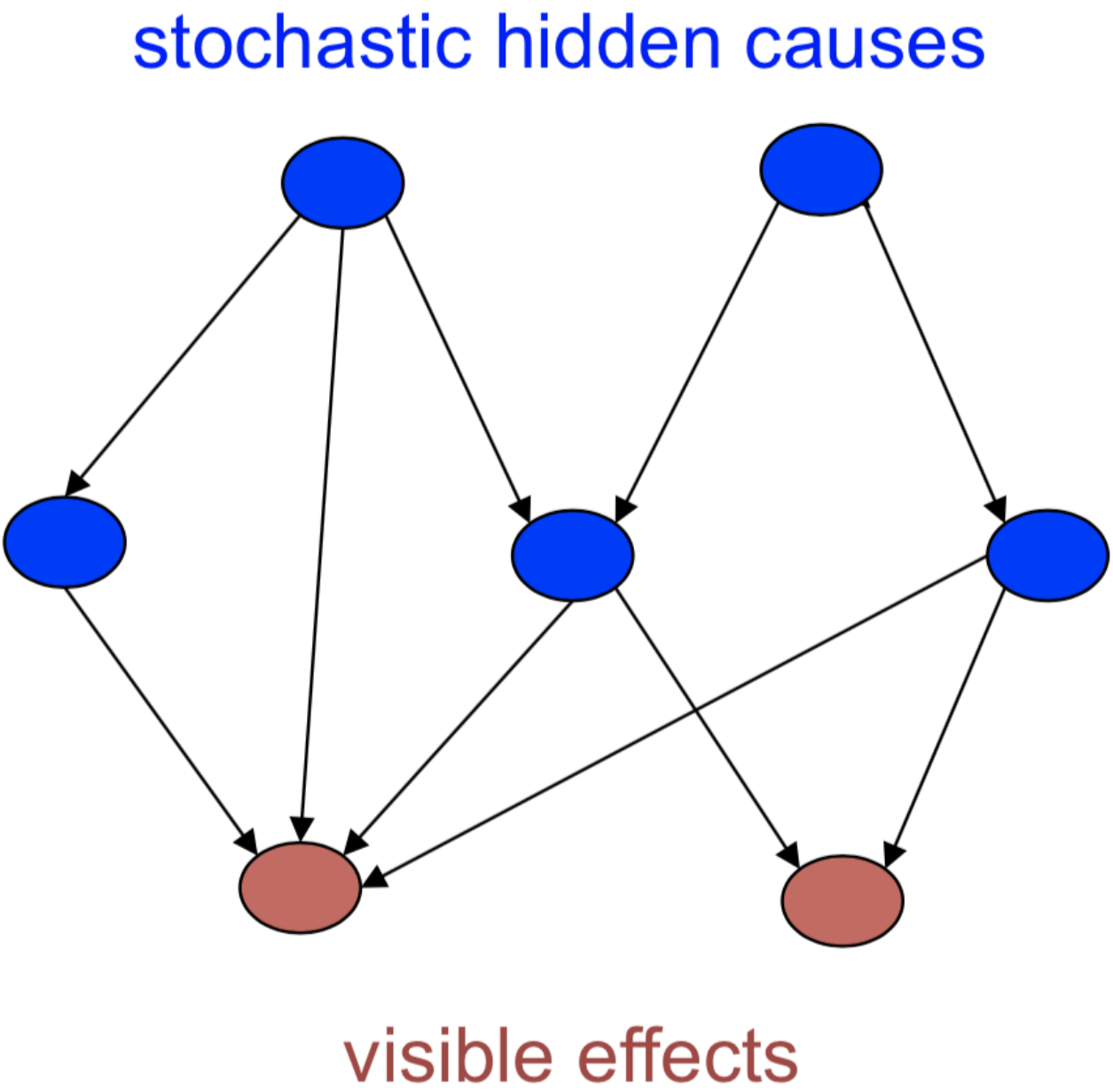
为什么SVM不适合表示复杂结构呢？总体来看，看待SVM有两种不同的视角。第一种观点是，SVM就是感知机的一个转世化身，只不过应用了一种比较精巧的技巧，即核技巧。这种方法的核心思想是将原始输入映射到一个非常大的非线性、非自适应的特征空间。然后，在这个空间中，学习一个权重。由于使用了最大边界分离超平面法，SVM能用一种有效的方法学习权重，还能防止过拟合。第二种观点仍然把SVM看作是感知机的转世化身，但是对其使用的特征有不一样的角度来思考。在这种观点下，训练集中的每个输入向量都可以用来定义一个“特徵”（原文为pheature。Hinton特意使用了一种不同的拼写方法来区别feature（特征））。每个特徵相当于是对测试输入和这个训练输入做了一次全局匹配，也就是说，定义了测试输入与该训练数据有多相似。然后，有一种很聪明的方法来同时学出如何向这些特徵分配权重来做出正确决策，并同时做特征选择。尽管两个看问题的角度不一样，但是总而言之，SVM都是使用了非适应的特征和一个可适应的权重层——这也就是SVM的限制：你不能使用SVM学出多层表示。一件逸闻是1995年，贝尔实验室自适应系统研究组的老大Larry Jackel和SVM的提出者Vladimir Vapnik打赌。前者赌2000年的时候人们会从理论层面上理解为什么大规模神经网络在大规模数据集上能有良好的效果；而后者打赌在2005年没人会使用反向传播和神经网络。现在看来两个人都错了，限制反向传播训练出的大规模神经网络的根本因素是没有快的电脑和大的数据集，而不是理论基础不完善

信念网

Hinton在20世纪90年代放弃反向传播的一个原因是需要太多有标签的数据集，但是现实生活中很难找到这么多有标签的数据集。但是，使用梯度下降学到权重的方法又比较好，不太让人容易放弃，所以接下来的问题就是能否找到另一种目标函数，可以既使用梯度下降又使用无标签的数据呢？这启发我们使用一种称为“生成模型”的模型，其目标函数是对输入数据建模，而不是预测某个标签。这与统计学和人工智能学界发展出来的新方向**图模型**相得益彰，这种模型是将离散的图结构组合进来，使用图来表示变量之间的依赖关系，进而可以根据给定的，其它变量的观察值，推出某个变量取值的概率分布。玻尔兹曼机就是一种图模型，不过是无向图模型。1992年，Radford Neal提出可以在玻尔兹曼机的基础上使用有向图，进而提出了sigmoid信念网的概念

深度网络的第二个问题是训练时间太长。当隐藏层很多时，学习过程特别慢。关于这个问题的原因众说纷纭，但是Hinton认为是因为无法找到一个比较合理的权重初始化方法。此外，反向传播算法可能会陷入不太好的局部极值点。如果初始化的随机权重离最优解太远，那么就更容易出现这样的问题。尽管可以使用更简单的模型，然后使用凸优化求解，但是这是一种妥协，逃避了真实数据的复杂度。克服这些反向传播的限制因素的最好方法是使用无监督学习算法，这样做是为了保留梯度下降和SGD的有效性和简洁性，不过是对输入建模，不再对输入和输出之间的关系建模。此时，权重的作用是最大化生成模型生成给定输入的概率，即最大化$p(x)$而非$p(y|x)$

那么接下来的问题是，应该学习什么样的生成模型呢？是学习基于能量的模型（例如玻尔兹曼机），还是学习由理想化的神经元组成的因果模型呢？还是学习两者的混合模型呢？在讲解因果信念网之前，先来看一下关于人工智能和概率的背景关系：20世纪70年代和80年代早期，人工智能学界的研究人员对概率论有强烈的抵触心态。在那时，如果研究AI的人提到概率，那就会被认为是朽木不可雕，愚不可及，没有摸到AI的门。（中间两段引用略）。进入到20世纪80年代，出现了很多关于专家系统的工作，例如医疗诊断或者矿藏勘探等。这些系统使用了大量的规则，由于要解决实际问题，因此要处理不确定性。由于前面学界共识的影响，大部分AI研究人员仍然不愿意使用概率论作为工具。不过由Pearl、Heckman、Lauritz等人提出的图模型还是拥抱了概率论，并且系统的确比专家系统使用的一个萝卜一个坑的方法要好。具体说来，离散的图可以表示变量之间的依赖关系，而有了图以后，需要做一些计算，来根据给定节点的观察值计算其它节点的期望值。信念网是图模型领域里对某种特殊子图的称呼，这种图是有向无环图，而且连接比较稀疏。在这种情况下，可以有效计算出未观察到的节点的概率。不过这种算法不适合稠密连接的图



信念网示意图

上图给出了一个信念网的示意图。尽管实际可能观察到任何变量，但是这里假设只能观察到叶子节点（没有边流出），即蓝色节点是随机的，隐藏单元；红色的节点是可观察的单元，其中隐藏节点可以看做是可见节点的“因”。当我们观察到变量时，就有两个问题可以解决

- 推断问题：推断隐藏变量的状态（概率分布）
- 学习问题：给定训练集，如何调整变量之间的连接，使得网络更可能生成训练数据？这里对连接的调整包含两项内容：其一是节点之间谁影响谁，其二是这种影响的强度

我们接下来来看一下图模型与神经网络的比较。早期的图模型使用专家来定义（稀疏连接的）图结构和条件概率，因此主要问题是如何正确推断，而不关注学习问题，因为这里隐含了来自于专家的知识。但是对于神经网络，学习永远是核心问题，基本不存在对知识进行硬编码这么一说。尽管CNN里某些基本属性是人工配置的，但是总体来讲，神经网络的知识来源于对训练数据的学习，不是专家的经验。神经网络的目标不是为了提供解释能力，以及稀疏的连接性来让推断变得容易（尽管有神经网络版的信念网）

总体来讲，有两种生成模型可以训练。一种是基于能量的模型，使用对称连接连接器一些二元的随机神经元，这样可以得到玻尔兹曼机。玻尔兹曼机不好训练，但是如果加上对连接性的限制，得到的RBM比较好训练，但是这样只能学习到一个隐藏层，为了训练的容易而放弃了多个隐藏层组成的神经网络的强大能力。另一种是因果模型，是由二元随机神经元组成的有向无环图，也就是sigmoid信念网。1992年，Neal提出了这种模型，并说明其比玻尔兹曼机更容易训练。由于sigmoid信念网的所有神经元都是0-1随机变量，要生成数据，先从最顶层开始，根据偏置值决定这些神经元是0还是1，这样就消除了随机性。得到了第一层的状态，就可以一层层推出下一层的状态，得到一个因果序列，租后就可以得到网络的“信念”，即一个无偏的对可见单元值向量的采样。因此，因果模型比玻尔兹曼机要更容易生成采样结果

（课件中的第三讲“sigmoid信念网的学习”在这次课程中被略去了）

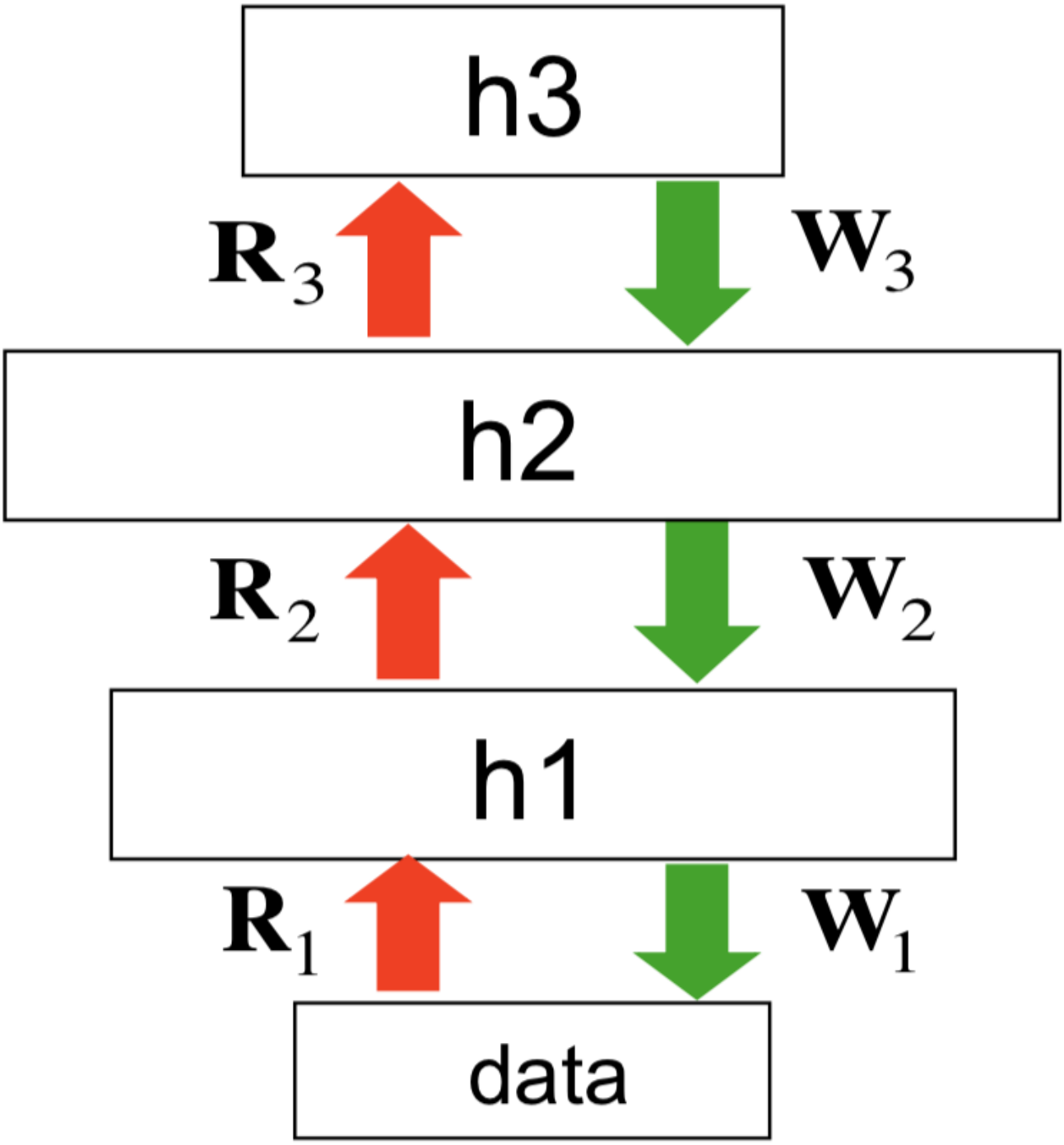
唤醒-睡眠算法

唤醒-睡眠算法（wake-sleep algorithm）是一种有效学习sigmoid信念网的算法，它背后的思想属于一个比较新的机器学习领域——变分学习（variational learning）。这种想法听上去比较疯狂：如果计算正确的后验分布比较难，那么可以算出一些简单的近似值，然后做最大似然学习

给定数据向量的情况下，学习诸如sigmoid信念网这类的复杂模型是比较难的，因为很难从隐藏配置的真是后验分布上取样，不好获得无偏的取样样本。所以一种疯狂的想法是，从某些其它分布取样，然后希望学习过程能正常进行。我们假设隐藏配置上的后验分布能被分解为每个独立隐藏单元分布的连乘，也就是说，假设数据已经给定，每个隐藏层单元都相互独立。这种假设在RBM中是正确的，但是在sigmoid信念网中是错误的

来快速看一下这种“阶乘分布”的例子。假设我们有三个隐藏单元，各自被激活的概率是0.3、0.6和0.8，那么隐藏状态为(1, 0, 1)的概率就是$0.3 \times (1 - 0.6) \times 0.8 = 0.096$。通常情况下，长度为$N$的二值向量的自由度是$2^N$（实际上是$2^{N-1}$，因为概率值相加必须为1），但是阶乘分布的自由度只有N

接下来来看一下分解-睡眠算法。在该算法里，神经网络有两套权重，如图所示



唤醒-睡眠算法示意图

模型本身是一个生成模型，因此绿色箭头是模型本来的权重，称为**生成权重**（generative weights），用来定义数据向量上的概率分布。此外，还有一些额外的权重，称为**识别权重**（recognition weights），以红色箭头表示。这些权重用来获得后验分布的近似值，即用来获得每个隐藏层的阶乘分布以（不是很好地）逼近后验分布。如算法名称所示，算法分成两个阶段：

- 唤醒阶段。在这个阶段，把数据放在可见层（最底层），然后使用识别权重做一个前向传播。在每个隐藏层，独立为每个隐藏单元做一个随机的二项决策，决策是应该打开还是关闭。所以，前向过程可以得到所有隐藏单元的随机二元状态，我们将它看作是从给定数据的情况下真实后验分布中的一个取样，并用它来做最大似然学习，不过学习的是生成权重
- 睡眠阶段。在这个阶段，做的事情都反了过来：在顶层，生成一个随机向量，根据隐藏单元的先验分布独立生成状态，然后一路走下来，每次生成一层的状态，最后生成模型的一个无偏抽样。拿到这个抽样以

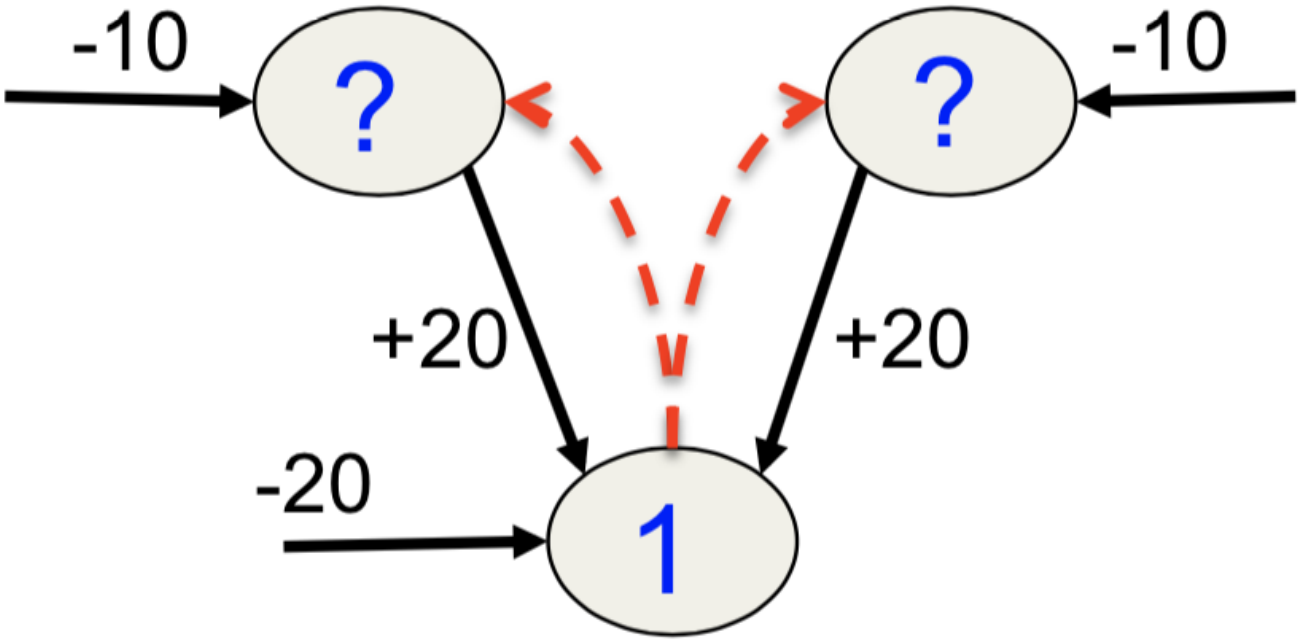
后，就可以看是否可以从数据恢复隐藏状态，也就是说，在这个阶段，使用生成权重做传播计算，学习识别权重

尽管实践上看，如果权重是随机初始化的，那么经历过几次唤醒-睡眠阶段的切换，能学习到一个很好的模型，但是算法仍然有瑕疵

1. 识别权重学的是把生成模型“反过来”。但是在学习的开始阶段，它们是在没有任何数据的空间里学着吧生成模型反过来。因为最开始使用模型生成数据时，权重不好，生成的数据与真实数据相比差别很大。所以这是一种浪费——当然咯，这是个小问题
2. 识别权重并不遵循数据对数概率的梯度，甚至也不遵循该概率变分界（variational bound）上的权重。由于没有遵循正确的梯度，因此会得到错误的模式平均（mode averaging）——这是个严重的问题。模型平均的概念下文会解释
3. 我们知道，由于相消解释作用（explaining away effect），最上面的隐藏层上的真实后验分布远非独立的，但是为了简单起见，我们假设分布独立。不过这种近似估计对中间的隐藏层可能不是一个很差的选择

尽管如此，Karl Friston认为这就是大脑工作的原理，但是Hinton认为应该还有更好的算法

最后来看一下模式平均。不过这里，笔者补充一个来自于原先13.3节中的例子，这个例子在讲模式平均时用到了，而且还可以用来对相消解释作用做说明。所谓相消解释，指的是即便两个隐藏状态（可以看做是事件发生的原因）在先验上是独立的，当我们观察到它们都能影响到的某件事的结果时，这两个原因也变得彼此相关了。例如，无论是卡车撞到房子还是发生地震，房子都会产生剧烈的抖动，而且发生地震和卡车撞击两件事是独立的。但是如果我们发现房子抖动了，而且观测到了地震，那么“发生了卡车撞击”这件事的概率实际上降低了。现在再来看这个例子，如下图所示



关于房屋抖动的信念网络。隐藏节点一个是发生地震，一个是卡车撞击

假设首先运行睡眠阶段的算法，通过模型产生数据，大多数情况下，顶部的两个单元都是关闭的，因为按照先验分布两件事发生的概率不大。因为它俩是关闭的，而且可见单元的偏置是-20，因此可见单元也是被关闭的。假设在很偶然的情况下（大概十亿分之一）两个隐藏单元中的一个被打开（两者被打开的概率相等），那么可见单元有一半的可能性会被打开。换句话说，如果看到可见单元被打开，那么有一半概率是左边的隐藏单元被打开，另一半是右边的被打开，但几乎不可能是两个都被打开或被关闭。那么来看看识别权重会学到什么：一半情况下，可见单元被置为1，两个隐藏单元被激活的概率相等，因此两者的概率都是0.5。最后，识别单元会在隐藏层上学习生成一个对(0.5, 0.5)的阶乘分布，最后会是有1/4的概率得到配置(1, 1)，另有1/4的概率

得到配置(0, 0)，但是这两者在可见单元被激活时都是很不可能的配置。实际上，最好的识别模型是只选择一个模式，例如只打开左边或只打开右边

Hinton神经网络与机器学习 14. 深度信念网络与判别微调

- **本文作者：** Tingxun Shi

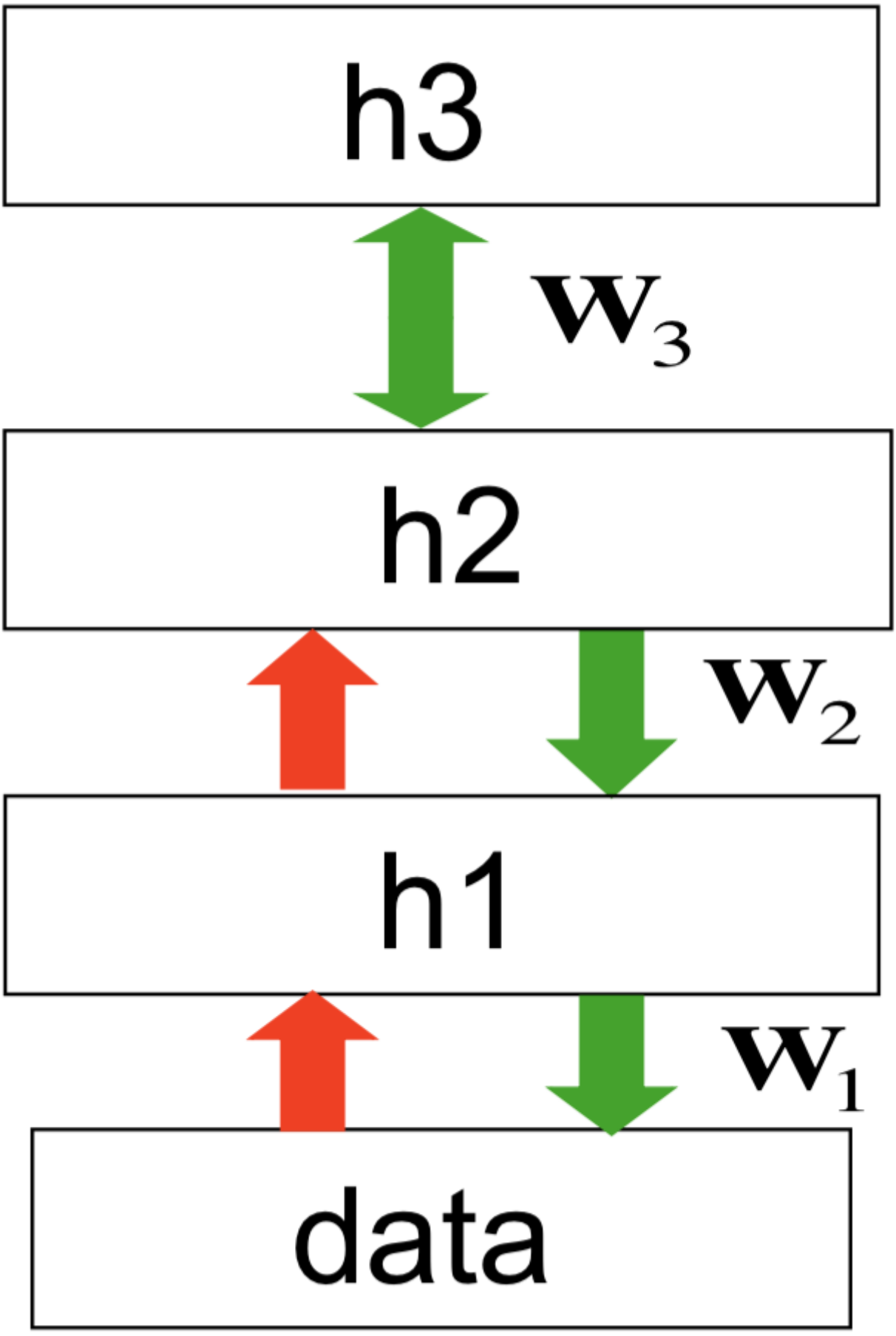
- **本文链接：** http://txshi-mt.com/2018/02/22/UTNN-14-Deep-Neural-Nets-with-Generative-Pre-Training/

通过堆叠RBM学习特征层

本讲开始，先不提sigmoid信念网（SBN），回过头看看受限玻尔兹曼机RBM。前面提到，学习RBM有一种相对来讲比较简单的方法，可以学出一层非线性特征（隐藏状态）。受此启发，可以将这层特征作为新的输入，再学出一个RBM的非线性特征，如此持续下去，最后可以得到一个由若干RBM堆叠起来的深度网络。但是，这实际上不只是一个简单的多层玻尔兹曼机，Hinton的学生G. Y. Tay指出它实际上是一个深度sigmoid信念网络。可以证明，每次这样添加一层特征，实际上是改善了生成训练数据的概率的对数值的变分下界

假设现在可见向量为$\mathbf{v}$，经过学习，可以得到第一层RBM，其隐藏状态为$\mathbf{h}_1$，权重为$\mathbf{W}_1$。然后，将$\mathbf{h}_1$作为输入，学习可以得到第二层RBM，其隐藏状态为$\mathbf{h}_2$，权重为$\mathbf{W}_2$。注意如果$\mathbf{W}_2$的初始值为$\mathbf{W}_1^\mathsf{T}$，且$\mathbf{h}_2$的节点数与$\mathbf{v}$的节点数相同，那么第二个RBM实际上就是$\mathbf{h}_1$一个不错的模型，因为它是第一个RBM翻转而成的，而RBM不太在乎哪个是隐藏单元哪个是可见单元。学到这两个RBM以后，可以将它们组合起来得到一个新的模型，它的上面两层是RBM，也就是第二个隐藏层和第一个隐藏层之间是一个无向模型，连接对称；而底下两层（第一个隐藏层和输入）之间是有向模型，像一个SBN。也就是说，对$\mathbf{v}$和$\mathbf{h}_1$之间的对称连接，舍弃那些从$\mathbf{v}$指向$\mathbf{h}_1$的边。这样就得到了一个深度信念网（DBN）。如果DBN由三个RBM堆叠而成，那么最上面两层（沿用前面的记法，是$\mathbf{h}_3$和$\mathbf{h}_2$）形成一个RBM，下面三层形成两个SBN。

对于这样三个RBM堆叠生成的DBN，生成数据的过程是：首先，在$\mathbf{h}_2$和$\mathbf{h}_3$之间往复运动，达到平衡态（这个过程使用的是交替吉布斯采样方法，即先平行更新所有$\mathbf{h}_3$中的神经元，然后平行更新所有$\mathbf{h}_2$中的神经元，再平行更新所有$\mathbf{h}_3$中的神经元，交替执行这样一个漫长的过程。最后，顶层的RBM实际上定义了$\mathbf{h}_2$的先验分布，当这个RBM达到稳态以后，使用$\mathbf{W}_2$生成$\mathbf{h}_1$，再使用$\mathbf{W}_1$生成数据。整个结构如下图所示



三层DBN

注意这里红色的箭头不是生成模型的一部分，它们是对应$\mathbf{W}_i$的转置，在推断时候用到

这里需要插一句的是，阶乘分布的均值不再是阶乘分布。假设RBM有4个隐藏单元，给定一个可见向量，这4个隐藏单元上的后验分布是阶乘的。假设对输入向量$\mathbf{v}_1$，隐藏单元被打开的概率分别为0.9、0.9、0.1和0.1，对输入向量$\mathbf{v}_2$，隐藏单元被打开的概率分别为0.1、0.1、0.9和0.9。那么这两个平均（聚合）以后的概率为0.5、0.5、0.5和0.5。现在，假设隐藏单元的状态为(1, 1, 0, 0)，如果该状态处于$\mathbf{v}_1$的后验分布中，则$P((1,1,0,0)) = 0.9^4 = 0.6561$（讲义这里好像算错了……）；如果该状态处于$\mathbf{v}_2$的后验分布中，则$P((1,1,0,0)) = 0.1^4 = 0.0001$，在聚合的后验分布下，$P((1,1,0,0)) = (0.6561 + 0.0001)/2 = 0.3281$。但是，如果这个聚合的后验分布也是可分解为阶乘的，那么就应该是$0.5^4 = 0.0625$

接下来回到主线：为什么这种贪婪的学习方法（学一个RBM，再学一个新的RBM来对第一个RBM的隐藏单元的行为模式建模）是可行的呢？先学出来的RBM的权重$\mathbf{W}$实际上定义了五个不同的分布：前两个是$P(\mathbf{v}|\mathbf{h})$和$P(\mathbf{h}|\mathbf{v})$，使用这两个分布可以通过交替马尔可夫链，在给定隐藏状态的条件下更新可见状态，再在给定可见状态的条件下更新隐藏状态。如果这个链运行的时间足够长，就可以获得一个来自于联合分布$P(\mathbf{v},\mathbf{h})$的样本，因此$\mathbf{W}$也定义了联合分布$P(\mathbf{v},\mathbf{h})$（当然，它也通过$-E$的形式更直接地定义了联合分布，但是因为网络里有太多单元，这个值无法计算）。对于这个联合分布，如果忽略$\mathbf{v}$，就得到了$\mathbf{h}$的分布，也就是由RBM定义的$\mathbf{h}$的先验分布。同理，也可以得到RBM定义的$\mathbf{v}$的先验分布。接下来，定义

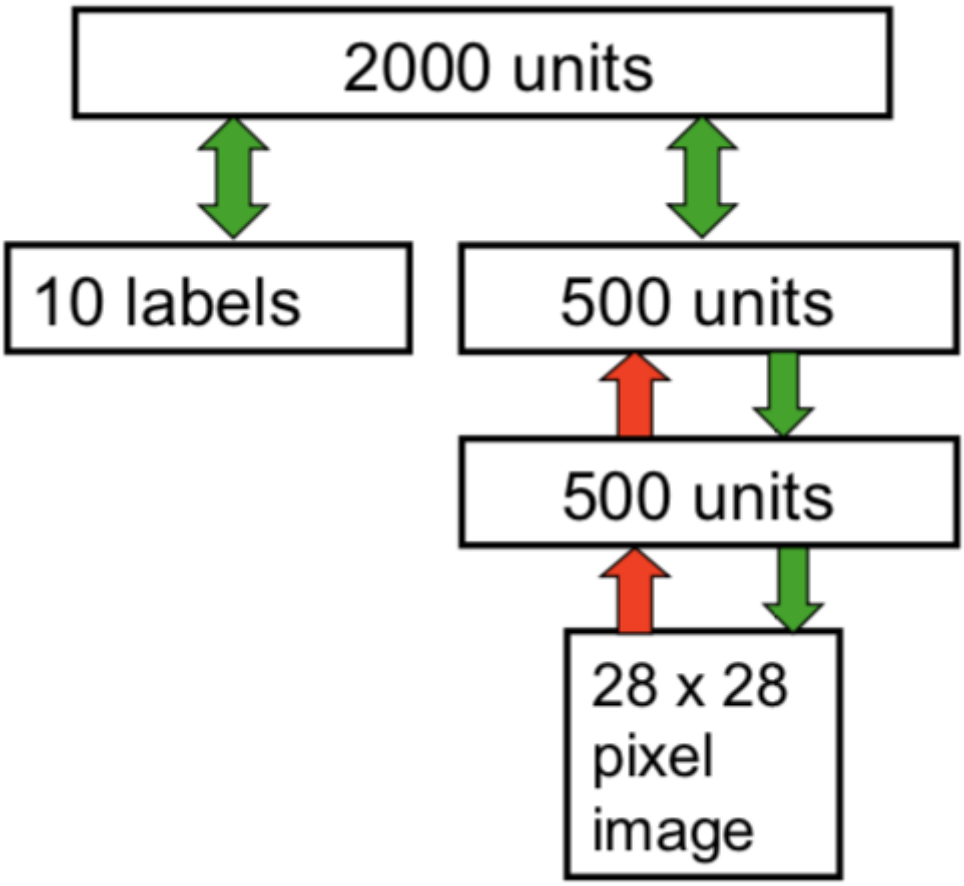
$$P(\mathbf{v}) = \sum_{\mathbf{h}} P(\mathbf{h})P(\mathbf{v}|\mathbf{h})$$

不看求和项里的$P(\mathbf{v}|\mathbf{h})$，只学习一个$P(\mathbf{h})$的更好的模型（能更好拟合聚合后验的先验），就也会改进$P(\mathbf{v})$的模型（这里聚合后验aggregated posterior指对训练集中的所有$\mathbf{v}$求$P(\mathbf{h}|\mathbf{v})$的平均）。所以，真正做的事情是，使用第一个RBM来获得这个聚合后验，然后用第二个RBM来为此聚合后验构建一个更好的模型。假设第二个RBM的初始状态就是把第一个颠倒过来，那么初始模型就是第一个RBM所得到的聚合后验，如果调整权重只能让系统更好

将RBM堆叠得到DBN以后，实际上可以对整个系统使用唤醒-睡眠算法的一个变种继续微调（fine-tune）：

1. 首先，做一个随机的自底向上的计算，调整底层的生成权重，使得底层可以长于重构特征（类似于标准的唤醒-睡眠算法）
2. 在顶层的RBM中，做若干次正向和反向的计算，对RBM的隐藏状态和可见状态做若干次抽样，然后做对比差异学习（第12章中所讲述）
3. 做一个随机的自顶向下的计算，通过底下的SBN生成数据，进而调整顶层的权重，使得顶层可以长于重构特征（即唤醒-睡眠算法的睡眠阶段）

DBN的一个应用是可以对MNIST手写数字和对应标签的联合分布建模，示意图如下所示



用于MNIST手写数字识别的DBN

图中，右侧下方由两层RBM堆叠生成的SBN是无监督学习产生的。将最上层的500个隐藏单元和10个标签（实际上是一个softmax单元）连接起来，送入一个大的，有2000个隐藏单元的隐藏层，训练顶层的RBM来对连接向量建模。顶层的RBM训练好以后，用前面介绍的改进版唤醒-睡眠算法微调整个系统，这样整个系统既可以识别数字，也可以很好地生成样本——这也就是导论章节里demo的来源

通过微调使DBN做判别学习

前面说到，DBN的每一层其实都是学习好的RBM，那么连接层与层之间的权重就可以看作是经过预训练得到的权重的初始值。在前一节里，通过唤醒-睡眠算法的一个变种，微调过的权重可以善于生成数据。在这一节，将使用反向传播算法来微调模型，来使它善于判别数据

为什么使用反向传播来微调预训练模型能收到不错的效果呢？对此的解释可以分成两个方面

- 从最优化的方面，如果网络比较大，而且每一层有自己的局部性，那么每一次学习一层效果会比较好。假设现在解决的是计算机视觉的问题，每一层都有一些局部感受野，那么相隔比较远的几块区域不会有什么相互作用，所以并行学出一个很大的层是很容易的。预训练的时候，先学出一些比较合理的特征检测器，然后再进行反向传播，可以让这些特征检测器有助于做判别，因为此时初始的梯度比随机的更合理，反向传播也不需要再做一个全局的搜索
- 从泛化的角度，使用与训练的网络能更好地降低发生过拟合的风险。因为最终学到的权重中包含的信息大部分来自于对输入向量分布的建模，而输入向量（如果做CV）包含的信息通常比标签包含的信息多很多。由于使用预训练模型已经得到了很多特征检测器，就不用再利用标签中包含的信息从头学习了，这些信息已经足够用来把这些检测器微调到类别边界上的合适位置。这里还有一点值得注意，就是因为反向传播已经不再用来发现新特征，因此甚至不需要特别多的有标签数据，使用无标签的数据已经能够发现好的特征了。尽管在没有标签的时候，由于系统不知道学到什么样的特征会对之后的判别任务有帮助，因此会

学到比较多无用的特征，但是如果某些特征有用，它们肯定比原始输入有用得多（也就是说，使用生成学习学到的特征肯定有那么几个特别有用，比从头训练一个判别特征要好）

（后面都是讲模型效果怎么好，略）

判别微调时发生了什么

首先，根据Bengio组的工作，第一个隐藏层的特征检测器的感受野在微调前后并没有发生太大的变化，但是效果就是对判别类任务有帮助，而且网络越深，帮助越大。此外，没有经过预训练的网络随着训练epoch的增多，权重最后会陷入不同局部最小值，而有预训练的网络的权重最后相对会集中一些

（上面略去了一些对实验比较具体的讲解）

所以为什么预训练很靠谱呢？假设生成图像-标签对的方式是，让一个现实生活中的人去生成一张照片（例如拍照），然后我们自己去给这张照片根据其像素点的关系贴一个标签，不依赖于拍照的人，那么试图学习一个从图像到标签的映射是合理的，因为此时标签直接依赖于图像本身。但是实际上，更合理也更常见的是，让照片的生成者自己去给生成的图像附加一个标签，那么此时标签就跟图像的生成者有关系，不再跟像素点有关了。而且，照片的生成过程一般是，当你看到一只奶牛，你把它照下来，然后说这张图片是奶牛的图片。因此此时照片的拍摄者和图片之间有很高的带宽，标签不再依赖于图像而依赖于拍摄者，而且两者之间的带宽比较低（不知道这张照片是否被上下颠倒了，不知道奶牛的颜色、死活、大小等等）。因此，比较靠谱的做法是试着先顺着高带宽的通路找到那个拍摄者，再由拍摄者决定标签——预训练的过程，就是从表象追溯本原的过程，而微调的过程就是从本原寻找标签的过程

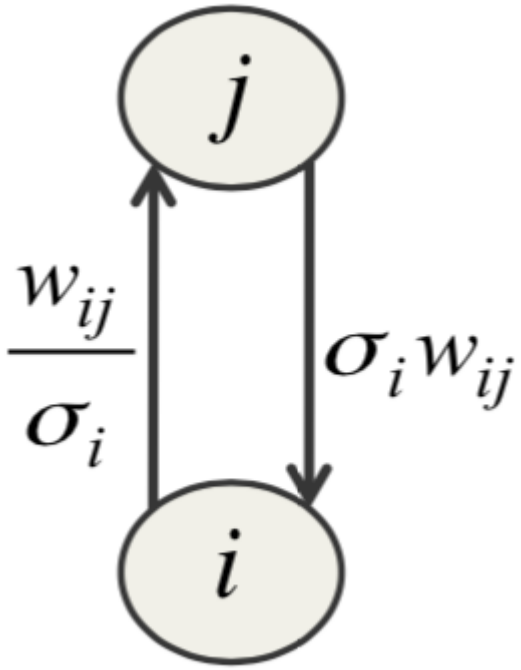
使用RBM对实数值数据建模

对于手写数字，图片上的某个像素点可能只沾上了一点墨水，因此可以用概率很好地建模——沾上了多少墨水，被激活的概率就是多少。但是对于真实的图像来说，这种做法不会特别奏效，因为真实图片中每个点的像素密度几乎总是它邻居的平均值，这种现象不能用logistic单元建模，它不能表示类似于“像素密度很可能是69，但是不可能是71或67”这样的事实，所以需要另一种单元，通常是用带有高斯范数的线性单元。这样，可以把像素点使用高斯变量建模。对此，仍然可以使用交替吉布斯采样法来运行马尔可夫链，来进行对比散度学习，不过需要设置一个更小的学习率。等式如下所示

$$E(\mathbf{v},\mathbf{h})=\sum_{i\in\text{vis}}\frac{(v_i-b_i)^2}{2\sigma_i^2}-\sum_{j\in\text{hid}}b_jh_j-\sum_{i,j}\frac{v_i}{\sigma_i}h_jw_{ij}$$

右边的第一项是一个向下凹的抛物线，防止能量值过大，其最小值点在第i个单元的偏置处取到。第i个单元的值离b_i越远，能量加得越多，呈二次项关系，因此是要让第i个可见单元离b_i尽可能近。第三项表示可见单元和隐藏单元之间的交互项，如果把这一项对v_i求梯度，可以得到一个常量$\sum_j(h_jw_{ij}/\sigma_i)$

能量函数写出来比较容易，也容易求导，但是学习时还是有问题。很多文章都说这种高斯二元RBM很难运转起来，而且也难以学到可见单元比较紧的方差。找到这个问题的原因花了很久，可以参看下图



学习高斯二元RBM的困难

考虑可见单元i对隐藏单元j施加的效果，当单元i的标准差σ_i很小的时候，它会放大自底向上的权重，同时降低自顶向下权重的效果。结果是，或者自顶向上的效果太大，或者自顶向下的结果太小，隐藏单元趋近饱和，要么被永远开启，要么被永远关闭。解决方法是将隐藏节点个数设置得比可见单元数多很多，因此它们之间小的权重也可以有大的自顶向下的效果（人多力量大）

这里，介绍一种单元，称为阶梯sigmoid单元（stepped sigmoid unit）。核心思想史，对每个随机的二元隐藏单元都创造很多拷贝，每个拷贝有相同的权重，也有基本的偏置b，但是对b施加一些固定的偏移量：第一个单元偏移量为-0.5，第二个-1.5，第三个-2.5，以此类推。这样，如果$\frac{w_{ij}}{\sigma_i}$特别小，没有单元被激活，但是随着这个值的变大，被激活的单元数也会线性增多。也就是说，当σ_i变小时，被激活的拷贝单元会变多，使得自顶向下的影响效果不会变小。但是，这种做法代价太昂贵，不过可以做一些快速的近似，有

$$\langle y\rangle = \sum_{n=1}^{\infty}\sigma(x+0.5-n)\approx \log(1+e^x)\approx \max(0,x+\text{noise})$$

这就是前面提到过的ReLU单元$\mathrm{ReLU}(x)=\max(0,x)$，计算起来很快

ReLU的一个性质是，如果参数的偏置为0，那么就会有缩放不变性，这种性质对处理图像很好。也就是说如果对图像$\mathbf{x}$，把所有像素密度乘以一个标量a，那么

$$\mathrm{ReLU}(a\mathbf{x})=a\mathrm{ReLU}(\mathbf{x})$$

但是ReLU并不是完全线性的，即

$$\mathrm{ReLU}(a+b)\neq \mathrm{ReLU}(a)+\mathrm{ReLU}(b)$$

这一性质有点像卷积神经网络展示出来的平移不变性

（14.5节是可选视频，这里跳过了）

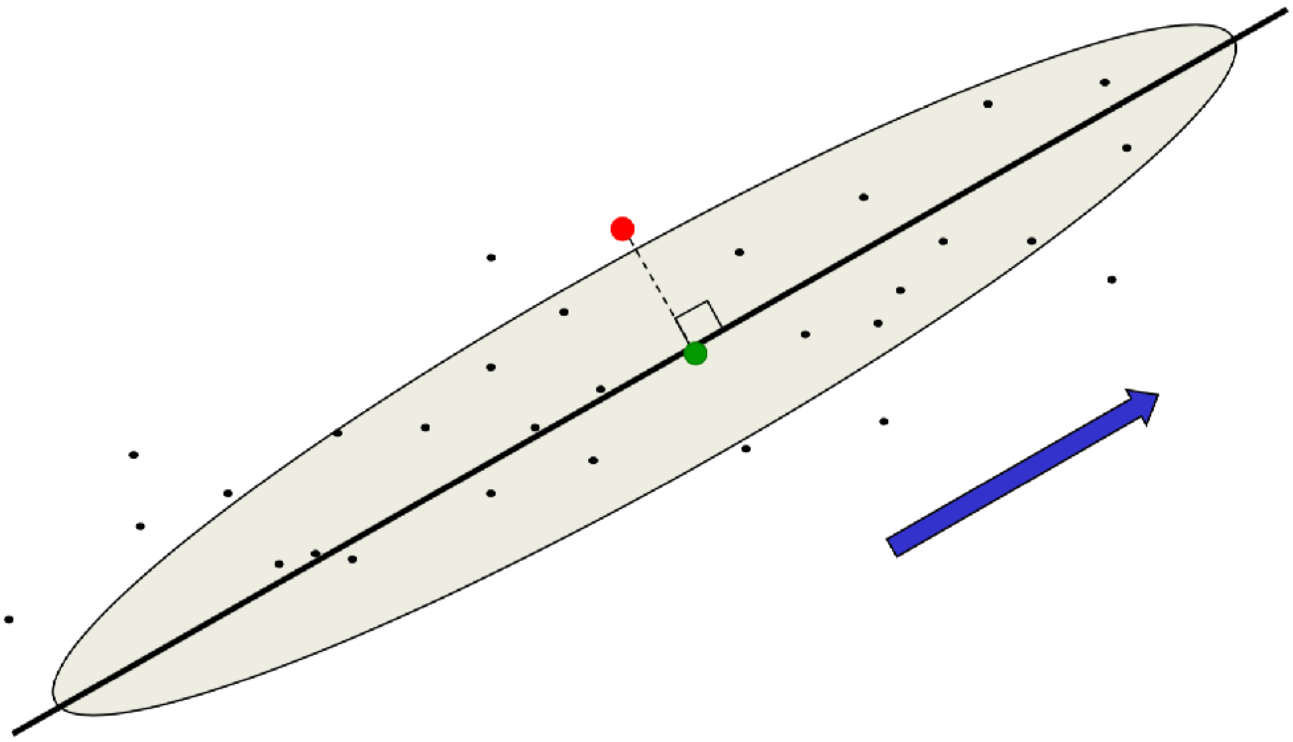
Hinton神经网络与机器学习 15. 自动编码器

- **本文作者：** Tingxun Shi

- **本文链接：** http://txshi-mt.com/2018/02/24/UTNN-15-Autoencoders/
-

从主成分分析到自动编码器

主成分分析（PCA）的思想是，高维数据通常可以使用很低维度的数据编码。假设输入是N维数据，现在想用M $(M < N)$维的数据表示之，方法是选择M个相互正交的方向，且数据在这些方向上有最大的方差，然后略去剩下$N - M$个数据不大的方向。这样，M个主方向就形成了一个低维子空间，可以将原始N维数据表示成它在子空间上的投影。由于舍弃的信息在舍弃的方向上变化不大，所以没有什么损失。要从降维表示的数据重构原有数据，只需要在舍弃的方向上使用所有数据在该方向上的均值，误差就是数据点在未表示方向上的分量与均值的差的平方和，如下图所示



PCA示意图

PCA将二维空间中的数据投影到黑色粗线上，其中红色的点投影后会变成绿色的点。重构误差是红点和绿点之间距离的平方

可以使用反向传播实现PCA，但是效率不高。网络结构是，输入和输出都是N维的，中间隐藏层有M个神经元，然后要减少输出和输入之间的平方误差。此时，中间的M维向量就是降维后的压缩表示。如果隐藏神经元和输出神经元都是线性的，学到的隐藏单元就可以最小化重构误差，就像PCA做的那样。不过隐藏单元不一定是主成分，它们跟主成分在一个线性空间中，但是可能有一些翻转和倾斜，方向不正交，而且M个分量倾向于有相等的方差

使用反向传播实现PCA尽管效率不太高，但是可以对算法进一步泛化。假设在降维表示的前面和后面加入两层非线性，靠近输入的称为编码权重，靠近输出的称为解码权重，那么就可以把数据表示到一个弯曲的面上，而非平面。这种做法像是在用监督学习方法来做无监督学习，得到的模型称为自动编码器

深度自动编码器

深度自动编码器，顾名思义，就是层数更多的自动编码器。它看上去应该比PCA更擅长做维度缩减，因为在两个方向上（输入到低维表示和低维表示到输出）都提供了灵活的映射，而且这种映射可以是非线性的。深度自

动编码器的训练时间应该是线性于输入数据集大小的，而且训练好以后，网络的编码部分应该很快，因为实质上就是矩阵相乘。但是不幸的是，很难使用反向传播来优化自动编码器，因为通常人们都使用很小的初始权重，因此反向传播时梯度会很快消失。不过现在，可以使用无监督的与训练方法逐层生成初始权重，或者像回声状态网络那样合理对权重初始化。2006年，Russ Salakhutdinov和Hinton实现了第一个成功的深度自编码器，他们训练了4个RBM，堆叠起来然后展开，最后使用反向传播做微调，效果很好

用于文档检索的深度自动编码器

文档检索的核心是计算两篇文档之间的相似度。一种比较传统的方法是潜在语义分析（Latent Semantic Analysis, LSA），它的做法是从文档中提取每个单词的数量，然后对得到的向量组成的矩阵做PCA。具体说，首先把文档转换成一个大的词袋，即维护一个很大的数组（向量），单词都转换为ID，数组中单词ID对应的下标是这个单词出现的数量。这种做法会丢失词之间的顺序信息，不过仍然保留了最基本的信息（此外，还要去掉一些跟主题无关的“停用词”，例如the, a, an这种）。假设词表大小为2000，有上百万个文档，那么就会得到上百万个2000维的向量。对于给定的文档，如果逐一比对，计算肯定慢死了。为了提高速度，需要对这个大矩阵降维，例如降到10维。以前的思路是使用PCA，但是既然深度自动编码器的效果比PCA要好一些，就可以使用深度自动编码器来做这个事情。网络分为7层，输入层和输出层（第1和第7层）都是2000维，第2层使用500个神经元（第6层同），第3层使用250个神经元（第5层同），最后第4层成为一个“瓶颈”，只有10个神经元，网络的目的就是要把原始2000维的信息放到这10个神经元里

需要注意的是，词数和像素值还不太一样，所以要把词袋中每个单词的个数除以停用词总个数N，这样每个向量都可以看作是一个概率向量，即该向量在所有维度上的值相加为1。可以这么理解，就是每个词对应的值是随便在文档里取一个非停用词，然后取到了这个值的概率。自动编码器的输出层是一个2000维softmax，输出的值也可以看作是一种概率向量。此外，这里还使用了一个技巧：使用输入向量激活第一个隐藏层时，将所有权重乘以N，来放大这个RBM中自底向上的权重

（后面效果略）

语义哈希

前面提到，可以训练一个深度自动编码器来产生文档的压缩表示，最中间一层的结果就是文档在低维空间的映射（前一节是10维，这一节是30维）。在训练时，先训练若干个RBM，把它们堆叠起来，然后展开，使用编码器权重的转置做相应解码器的权重，然后使用反向传播做微调。微调时，在输入端加入一些高斯噪声，因此自动编码器需要把神经元更倾向于调到恒开或恒关，以抵抗噪声的干扰。在测试的时候，直接把中间编码层神经元的值按照阈值设为0或1，这样，自动编码器会把词袋模型转化成少数几个二进制数，也就是说，学到一些二进制特征

后来，Alex Krizhevsky（似乎就是AlexNet的发明人？）发现其实不需要加入高斯噪声，只需要使用随机二进制单元就可以。这样，在正向阶段，使用logistic的结果随机输出一个0/1值，然后，在反向阶段使用logistic的实数输出来得到光滑的梯度

得到这些短的二进制码后，就可以对每个已知的文档做一个顺序搜索。当查询文档到来时，首先提取这个二进制码，如果这个码不对应于任何已知文档，就和所有已知文档的编码作比较。尽管这种比较操作可以通过位运算很快完成，但是当已知文档数以亿计时，还是有点慢。因此，可以把这些编码看作是内存地址，当查询到来时，直接看查询文档的地址周围有哪些编码好的文档（可以通过翻转较少几位的方法来达到查看周边地址的功能），这些文档就是相似文档。这种方法被Hinton称为“超市搜索法”，就好像你进到一个没去过的超市，想买鲲鱼罐头但不知道怎么走，就可以问导购金枪鱼罐头哪里卖，然后去ta指的地方附近找鲲鱼罐头

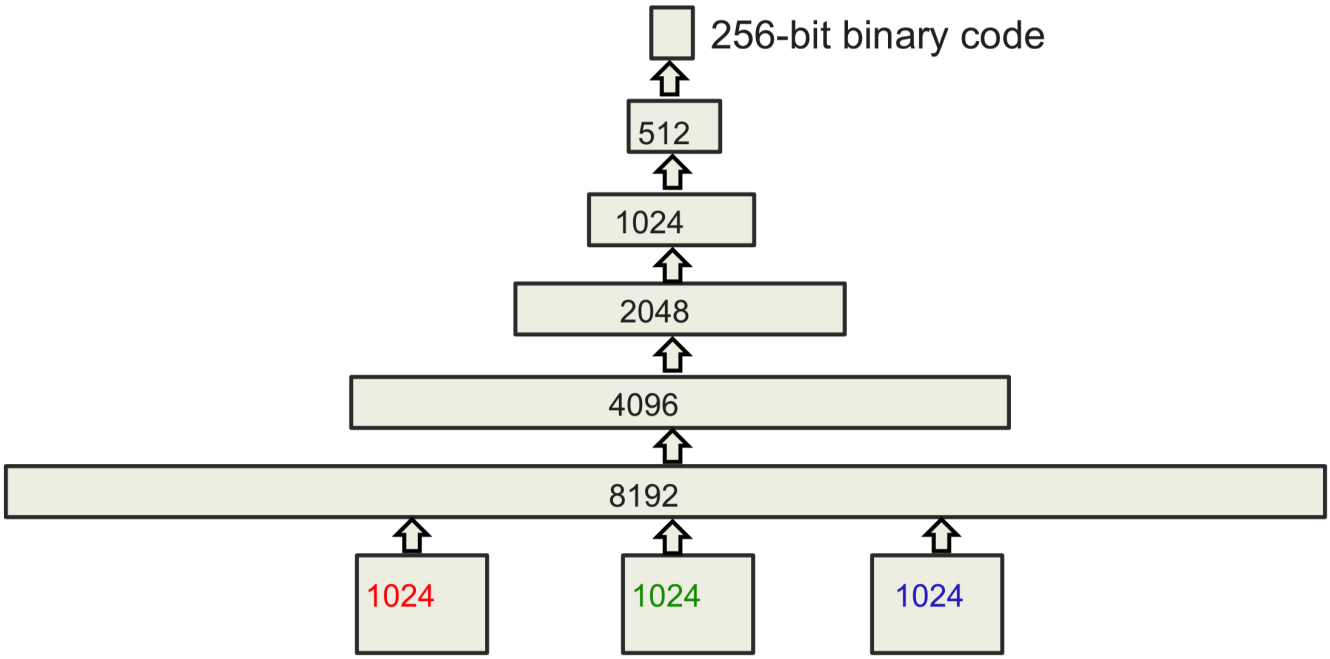
还可以从另一个角度理解语义哈希。大部分快速的检索算法都是先从查询中提取一些关键字，然后找到对应的索引列表。例如Google会维护一个含有罕见词文档的倒排索引，当查询中有对应的罕见词时，马上就可以找到

文档列表，然后将这个列表与其他列表做集合相交的操作来找到满足所有查询的文档。现在，计算机有一种特殊的硬件，称为内存总线（memory bus），它可以在单条指令里对32个特别长的列表求交集，32位编码中每个bit都对应内存中一半的地址。语义哈希实际上就是用机器学习技术来将检索问题映射到计算机擅长做的列表相交问题

为图像检索学习二进制编码

传统的图像检索通常都是利用图像的题注来做，但是为什么不直接使用图像呢？图像包含的信息肯定要比题注多。这里最基本的问题是图像的单个像素对说明图像内容没有太多贡献，~~而从图像中提取出来物体类别太难了~~（本条已过时）。退而求其次，那是否可以从图像中抽取出来一个向量，而让这个向量包含图像内容的信息呢？如果对向量的大小不加限制，那么查找效率和存储都是问题，所以需要像语义哈希那样学习一个简短的二进制编码

这里具体使用的是一个两阶段方法。首先，使用一个短小的二进制编码（大概30位），来通过语义哈希快速缩小查找范围，将目光聚焦到若干靠谱的图像上。然后，使用一个稍长的二进制编码（256位）做一个精确匹配。Alex Krizhevsky设计的网络结构如下所示



Alex Krizhevsky设计的深度自动编码器

其中输入是32x32的图像，分成RGB三个信道，所以输入共有3072维。整个网络有6700万个参数

（中间效果介绍略）

进一步地，我们希望信息提取能对图像的内容更敏感，而不是对像素密度敏感。要达到这样的效果，可以先训练一个巨大的，用来做物体识别的网络（如第五章所讲的网络），然后去这个网络最后一个隐藏层的激活向量，将其看作是图像的表示

（效果介绍略）

用于预训练的浅层自动编码器

说完了深层自动编码器，这里想说一下只有一个隐藏层的浅层自动编码器。当训练RBM时，如果使用的是一个时间步的对比散度训练，那么它会试着重新构造出与训练数据比较像的数据。这种模型看上去像是一个自动编码器，但是由于隐藏单元只允许取0或1的值，因此它其实被正则化得比较狠，因此能力收到了很大限制。如果

使用最大似然法训练RBM，那么得到的模型就完全不像自动编码器了：假设某个像素点是个纯噪声，自动编码器会试图重构它，而使用最大似然训练的RBM会完全忽视这个像素，只使用输入的偏置建模

由于RBM可以看做是一种经过了强正则化得到的自动编码器，也许可以把预训练时用到的RBM替换为自动编码器的堆叠。如果直接这么做的话，预训练的效果不是特别好。但是，如果稍作改进，情况会不太一样。Vincent等人提出了一种**降噪自动编码器**的模型，训练这种模型时，要将输入向量的大部分维度都设为0（不同的输入被置0的维度不同）。这种做法像是在输入层做dropout。由于降噪自动编码器的目标仍然是重构数据，而且输入缺少了很多信息，因此它就会被逼着使用隐藏节点提取输入之间的关联性，重构出被置零的数据。如果将降噪自动编码器堆叠起来，效果会不错，可以跟堆叠RBM相提并论，甚至更好。这种方法唯一的问题是缺少RBM那样优美的差分下界，不过这是理论界要解决的问题

2011年，Rifai等人提出了一种称为收缩自动编码器（contractive autoencoder, CAE）的模型。核心思想史让隐藏单元的激活尽可能**不敏感**于输入（不过不能完全忽略输入，毕竟是要重构它们）。达到这个目标的做法是对每个异常单元对每个输入的梯度的平方做惩罚，因此当输入变化的时候，隐藏单元不会发生太大变化。CAE用来预训练的效果不错，CAE的编码中只有一小部分对输入的变化敏感，而且对不同的输入，隐藏层对应的敏感单元不同，这种表现跟RBM差不多

目前，有很多种做逐层预训练来发现好特征的方法。如果数据集没有足够的标签，那么这种方法（要用在使用标签之前）对后续的判别微调很有帮助——尤其是有海量无标签数据，但是只有少量有标签数据时。但是如果有标签的数据集很大，那么就没必要用预训练来初始化权重了。不过，如果想让网络更大， 那么预训练还是有必要的

（后面这点视频的核心思想是，如果有标签数据集规模与网络规模匹配，那么预训练和正则化就不太必要。但是如果网络规模比数据集规模大很多，预训练和正则化就有必要了）