一起读懂传说中的经典：受限玻尔兹曼机

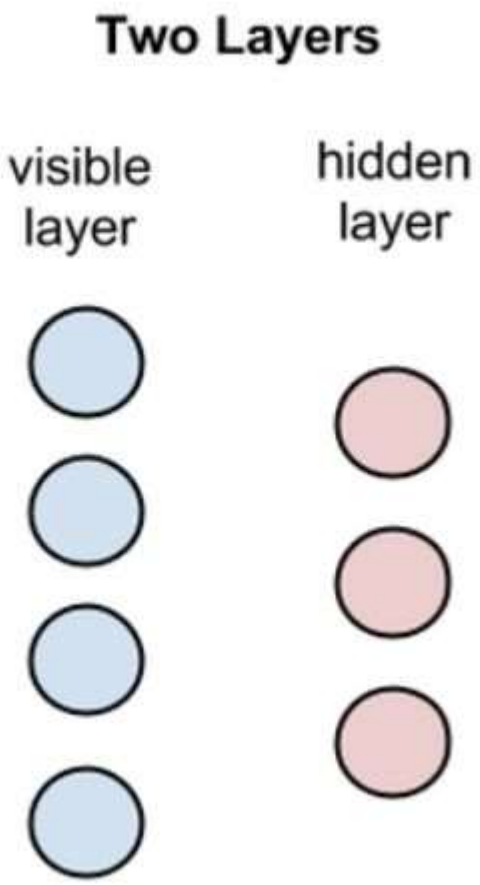
尽管性能没有流行的生成模型好，但受限玻尔兹曼机还是很多读者都希望了解的内容。这不仅是因为深度学习的复兴很大程度上是以它为前锋，同时它那种逐层训练与重构的思想也非常有意思。本文介绍了什么是受限玻尔兹曼机，以及它的基本原理，并以非常简单的语言描述了它的训练过程。虽然本文不能给出具体的实现，但这些基本概念还是很有意思的。

定义 & 结构

受限玻尔兹曼机（RBM，Restricted Boltzmann machine）由多伦多大学的 Geoff Hinton 等人提出，它是一种可以用于降维、分类、回归、协同过滤、特征学习以及主题建模的算法。更多关于如何部署诸如 RBM 这样的神经网络的具体例子，请参阅 deeplearning4j 关于深度学习用例的内容。

本文将从受限玻尔兹曼机的关系和历史重要性出发，首先讨论什么是 RBM。随后，我们会使用图表和浅显的语言来描述它们的运行原理。

RBM 是两层神经网络，这些浅层神经网络是 DBN（深度信念网络）的构建块。RBM 的第一层被称为可见层或者输入层，它的第二层叫做隐藏层。



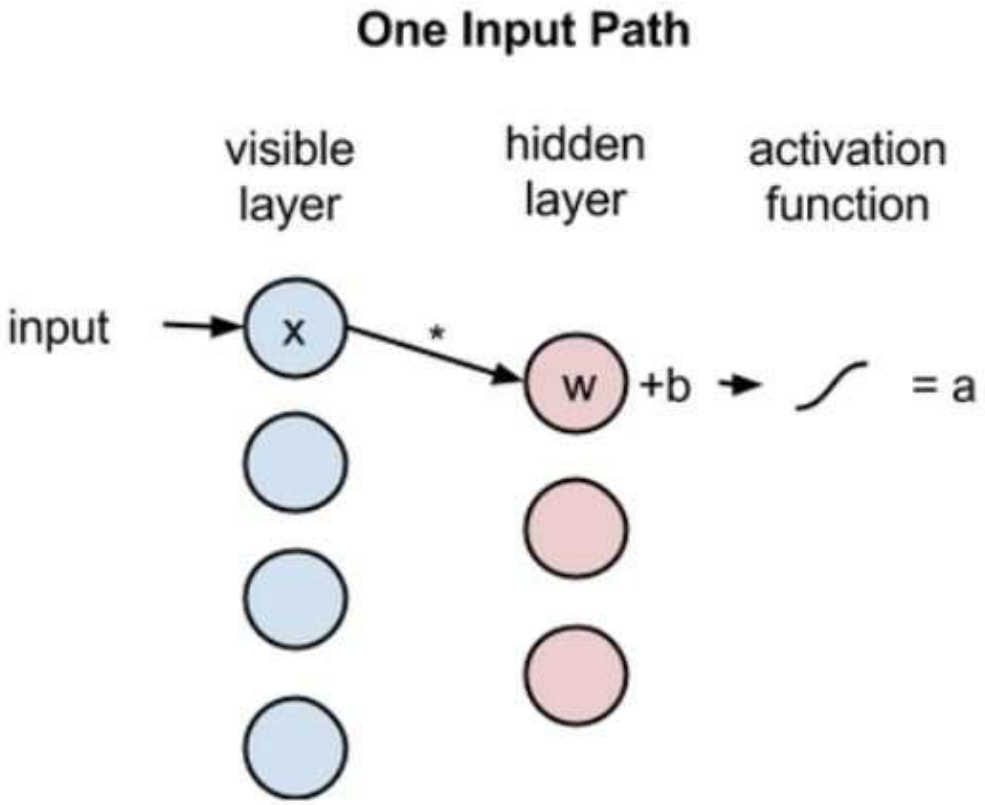
上图中的每个圆圈代表一个类似于神经元的节点，这些节点通常是产生计算的地方。相邻层之间是相连的，但是同层之间的节点是不相连的。

也就是说，不存在层内通信，这就是 RBM 中的限制所在。每一个节点都是处理输入数据的单元，每个节点通过随机决定是否传递输入。随机意味着「随机判断」，这里修改输入的参数都是随机初始化的。

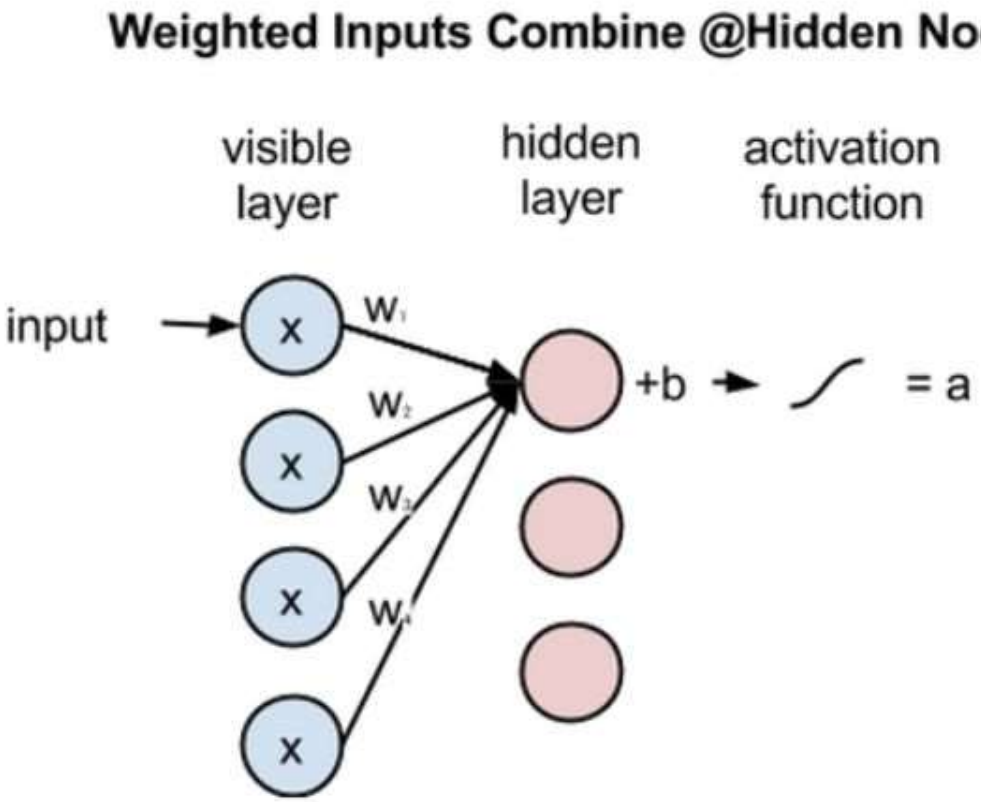
每个输入单元以数据集样本中的低级特征作为输入。例如，对于一个由灰度图组成的数据集，每个输入节点都会接收图像中的一个像素值。MNIST 数据集有 784 个像素点，所以处理它们的神经网络必须有 784 个输入节点。

现在让我们跟随单像素穿过这两层网络。在隐藏层的节点 1，x 和一个权重相乘，然后再加上一个偏置项。这两个运算的结果可作为非线性激活函数的输入，在给定输入 x 时激活函数能给出这个节点的输出，或者信号通过它之后的强度。这里其实和我们常见的神经网络是一样的过程。

```
activation f((weight w * input x) + bias b ) = output a
```



接下来，让我们看一下多个输入单元是如何结合在一个隐藏节点的。每个 x 乘以一个独立的权重，然后相加后再加一个偏置项，最后将结果传递到激活函数来产生输出。

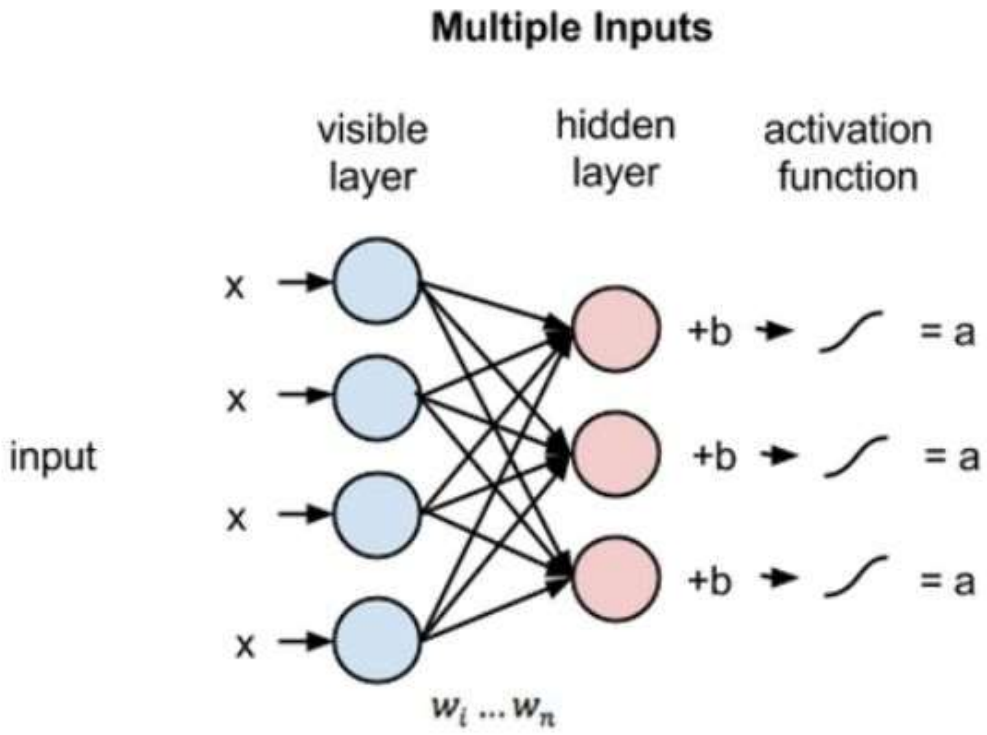


因为所有可见（或输入）节点的输入都被传递到所有的隐藏节点了，所以 RBM 可以被定义为对称二分图（symmetrical bipartite graph）。

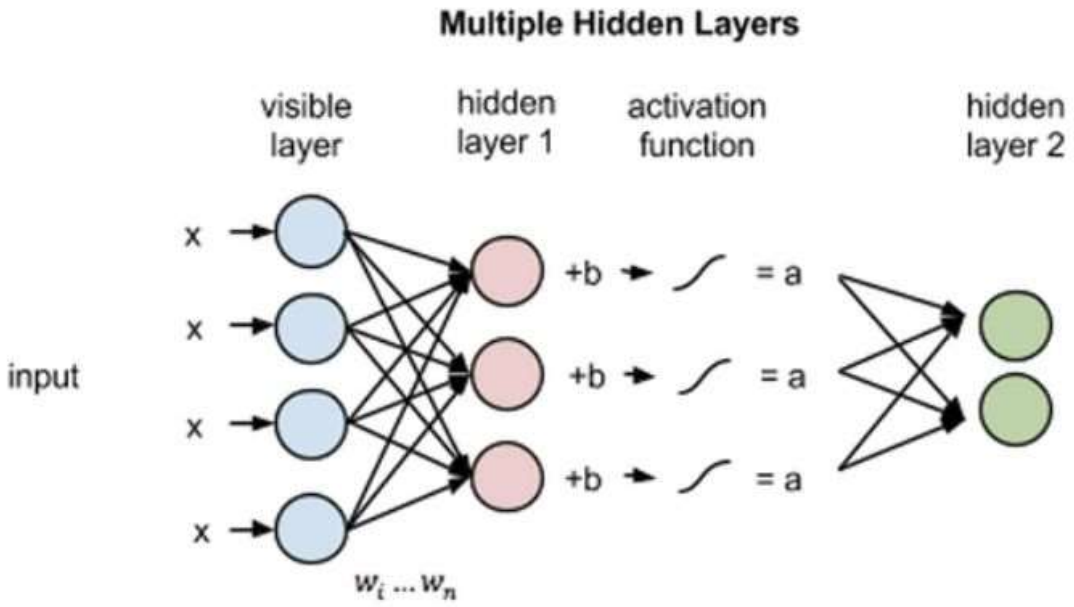
对称意味着每个可见节点都与一个隐藏节点相连（如下所示）。二分则意味着它具有两部分，或者两层。图是一个数学术语，指的是由节点和边组成的网络。

在每一个隐藏节点，每个输入 x 都与对应的权重 w 相乘。也就是说，一个输入 x 会拥有 12 个权重（4 个输入节点×3 个输出节点）。两层之间的权重总会形成一个矩阵，矩阵的行数等于输入节点的个数，列数等于输出节点的个数。

每个隐藏节点会接收 4 个与对应权重相乘的输入。这些乘积的和再一次与偏置相加，并将结果馈送到激活函数中以作为隐藏单元的输出。



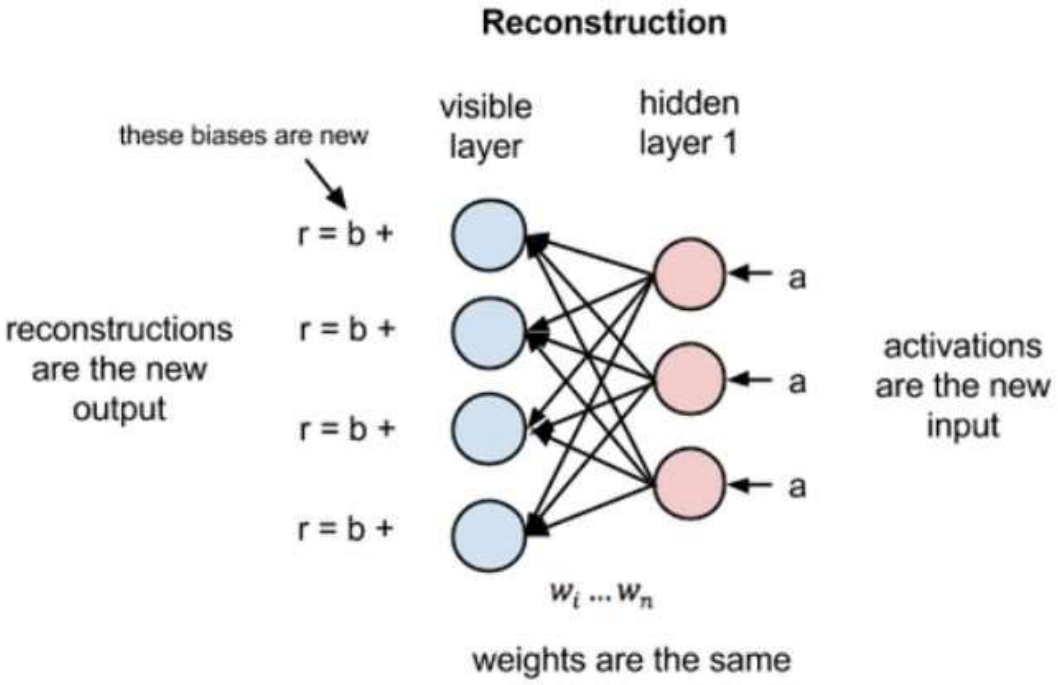
如果这两层是更深网络的一部分，那么第一个隐藏层的输出会被传递到第二个隐藏层作为输入，从这里开始就可以有很多隐藏层，直到它们增加到最终的分类层。对于简单的前馈网络，RBM 节点起着自编码器的作用，除此之外，别无其它。



重建（Reconstruction）

但是在本文关于 RBM 的介绍中，我们会集中讨论它们如何以一种无监督的方式通过自身来重建数据，这使得在不涉及更深层网络的情况下，可见层和第一个隐藏层之间会存在数次前向和反向传播。

在重建阶段，第一个隐藏层的激活状态变成了反向传递过程中的输入。它们与每个连接边相同的权重相乘，就像 x 在前向传递的过程中随着权重调节一样。这些乘积的和在每个可见节点处又与可见层的偏置项相加，这些运算的输出就是一次重建，也就是对原始输入的一个逼近。这可以通过下图表达：



因为 RBM 的权重是随机初始化的，所以，重建结果和原始输入的差距通常会比较大。你可以将 r 和输入值之间的差值看做重建误差，然后这个误差会沿着 RBM 的权重反向传播，以一个迭代学习的过程不断反向传播，直到达到某个误差最小值。

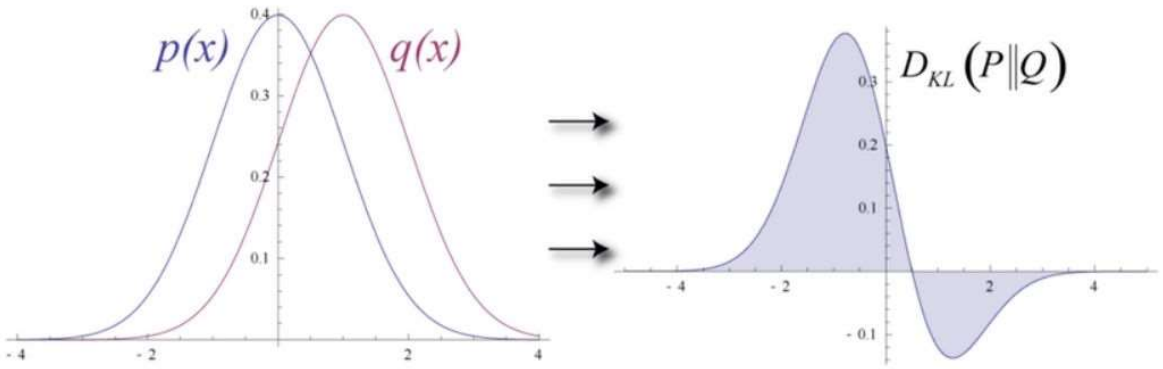
正如你所看到的，在前向传递过程中，给定权重的情况下 RBM 会使用输入来预测节点的激活值，或者输出的概率 x:p(a|x; w)。

但是在反向传播的过程中，当激活值作为输入并输出原始数据的重建或者预测时，RBM 尝试在给定激活值 a 的情况下估计输入 x 的概率，它具有与前向传递过程中相同的权重参数。这第二个阶段可以被表达为 p(x|a; w)。

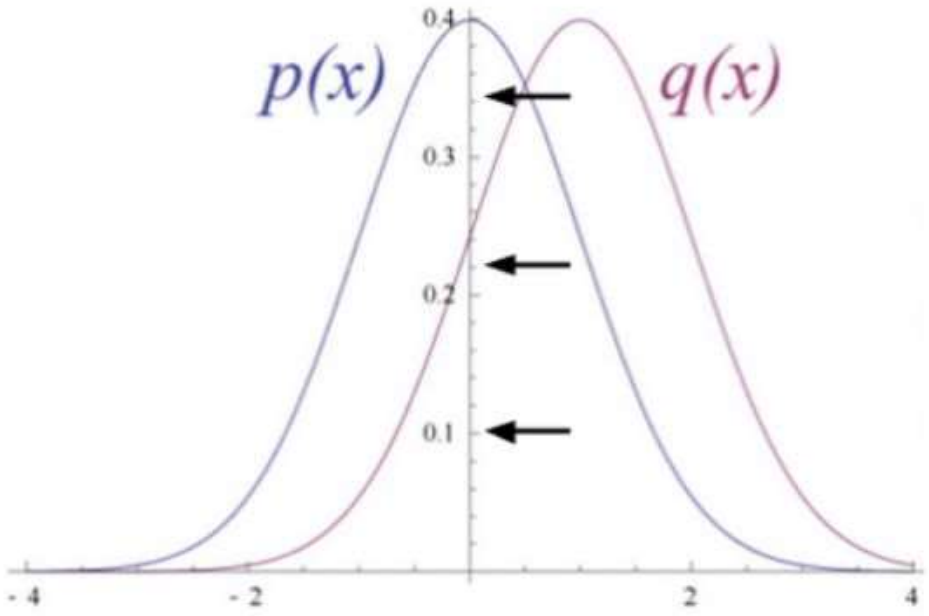
这两个概率估计将共同得到关于输入 x 和激活值 a 的联合概率分布，或者 p(x, a)。重建与回归有所不同，也不同于分类。回归基于很多输入来估计一个连续值，分类预测出离散的标签以应用在给定的输入样本上，而重建是在预测原始输入的概率分布。

这种重建被称之为生成学习，它必须跟由分类器执行的判别学习区分开来。判别学习将输入映射到标签上，有效地在数据点与样本之间绘制条件概率。若假设 RBM 的输入数据和重建结果是不同形状的正态曲线，它们只有部分重叠。

为了衡量输入数据的预测概率分布和真实分布之间的距离，RBM 使用 KL 散度来度量两个分布的相似性。KL 散度测量的是两条曲线的非重叠区域或者说发散区域，RBM 的优化算法尝试最小化这些区域，所以当共享权重与第一个隐藏层的激活值相乘时就可以得出原始输入的近似。图的左边是一组输入的概率分布 p 及其重构分布 q，图的右侧是它们的差的积分。

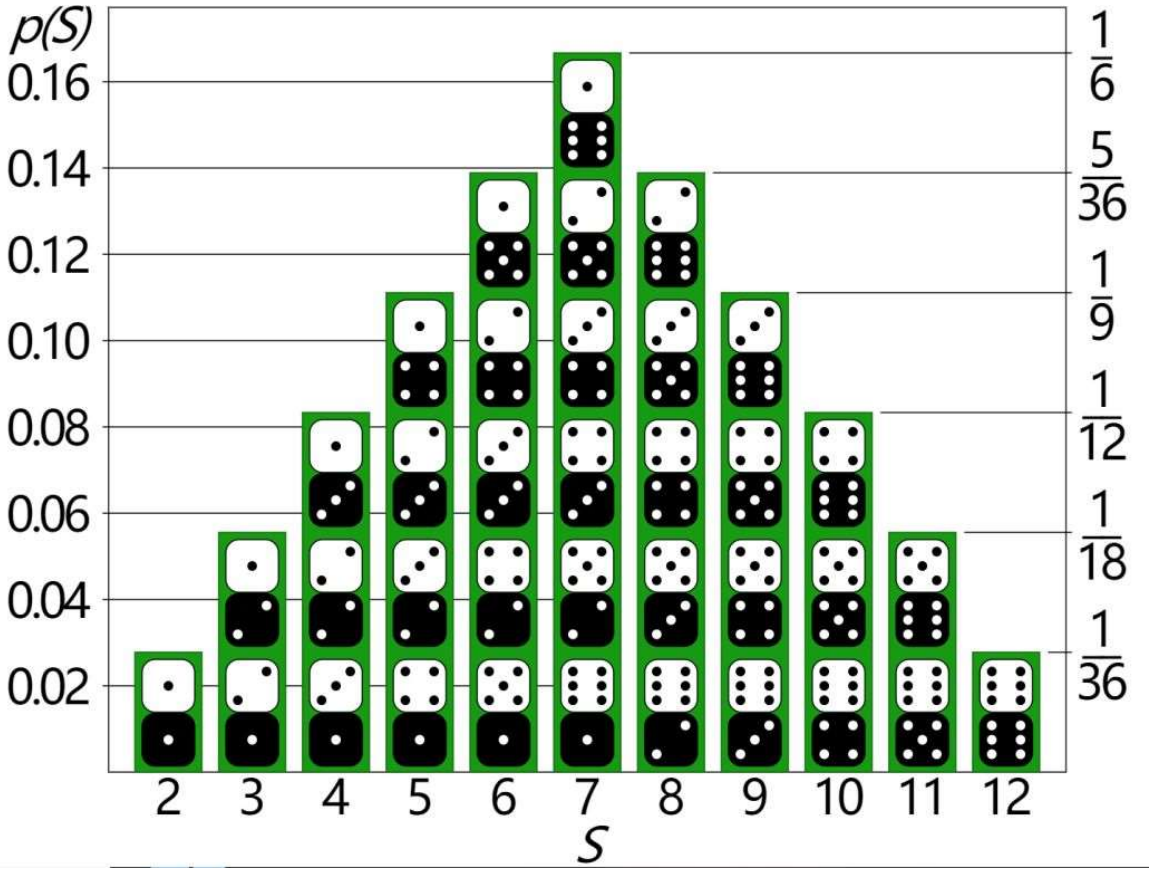


迭代地根据它们产生的误差来调节权重，RBM 学会了逼近原始数据。你可以说权重在慢慢地反映输入数据的结构，并通过隐藏层的激活值进行编码，学习过程就像两个概率分布在逐步重合。



概率分布

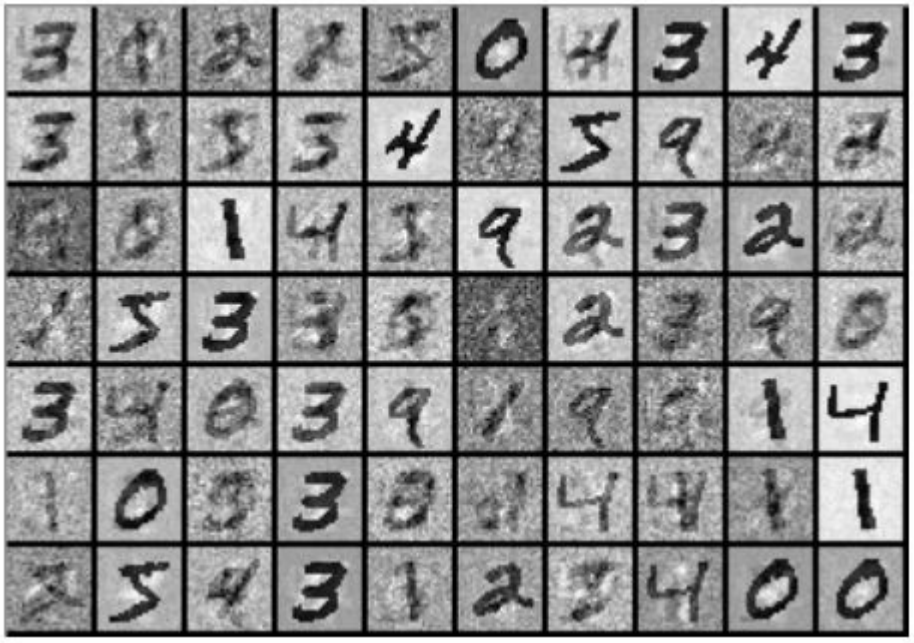
让我们来讨论一下概率分布。如果你在掷两个骰子，所有结果的概率分布如下：



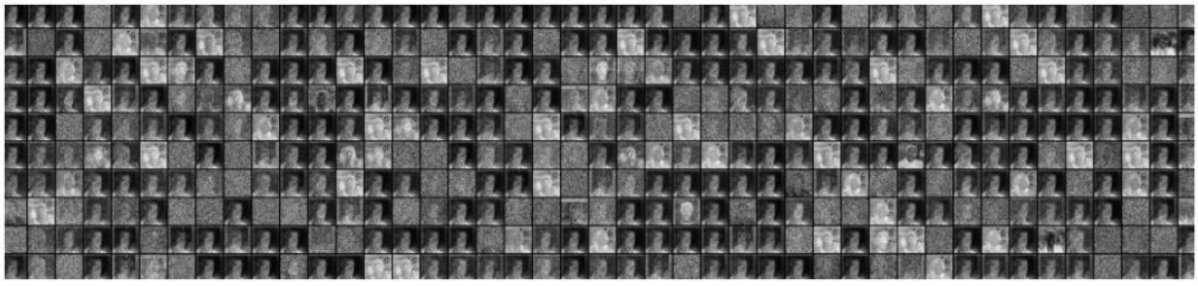
也就是说，和为 7 的结果是最有可能出现的，因为相比于 2 到 12 等其它结果，有更多的抛掷组合可以得到 7 这个结果（3+4,1+6,2+5）。

或者举另一个例子：语言是字母的特定概率分布，因为每一种语言会使用一些字母较多，而另一些较少。在英语中，字母 e、t 以及 a 是最常见的，然而在冰岛语中，最常见的字母是 a、t 和 n。因此尝试使用基于英语的权重集合来重建冰岛语将会导致较大的差异。

同样，图像数据集拥有像素值的唯一概率分布，这取决于数据集中图像的种类。像素值的分布取决于数据集中的图像类别，例如 MNIST：



或者 Faces in the Wild 数据集中标记的头像：



想象一下仅输入狗和大象图片的 RBM，它只有两个输出节点，每个结点对应一种动物。在前向传递的过程中 RBM 会问自己这样的问题：在给定的这些像素下，我应该向哪个节点发送更强的信号呢，大象节点还是狗的节点？在反向传递的过程中 RBM 的问题是：给定一头大象的时候，应该期望那种像素分布？

那就是联合概率分布：给定 a 时 x 的概率以及给定 x 时 a 的概率，可以根据 RBM 两层之间的共享权重而确定。

从某种意义上而言，学习重建的过程就是学习在给定的图像集合下，哪些像素会倾向于同时出现。由深层网络的隐藏层节点所产生的激活状态表现出来的共现现象：例如，「非线性灰色管＋大的、松软的耳朵＋皱纹」可以作为一个分布。

在上面的两幅图像中，你看到了用 Deeplearning4j 实现的 RBM。这些重建代表着 RBM 的激活值所「认为」输入数据看起来的样子，Geoff Hinton 将其称为机器「做梦」。当被呈现在神经网络在训练过程时，这种可视化是非常有用的启发，它让人确信 RBM 确实在学习。如果不是，那么它的超参数应该被调整。

最后一点：你会注意到 RBM 有两个偏置项。这是有别于其它自动编码器的一个方面。隐藏层的偏置项有助于 RBM 在前向传递中获得非零激活值，而可见层的偏置有助于 RBM 学习后向传递中的重建。

多层受限玻尔兹曼机

一旦 RBM 学到了与第一隐藏层激活值有关的输入数据的结构，那么数据就会沿着网络向下传递一层。你的第一个隐藏层就成为了新的可见层或输入层。这一层的激活值会和第二个隐藏层的权重相乘，以产生另一组的激活。

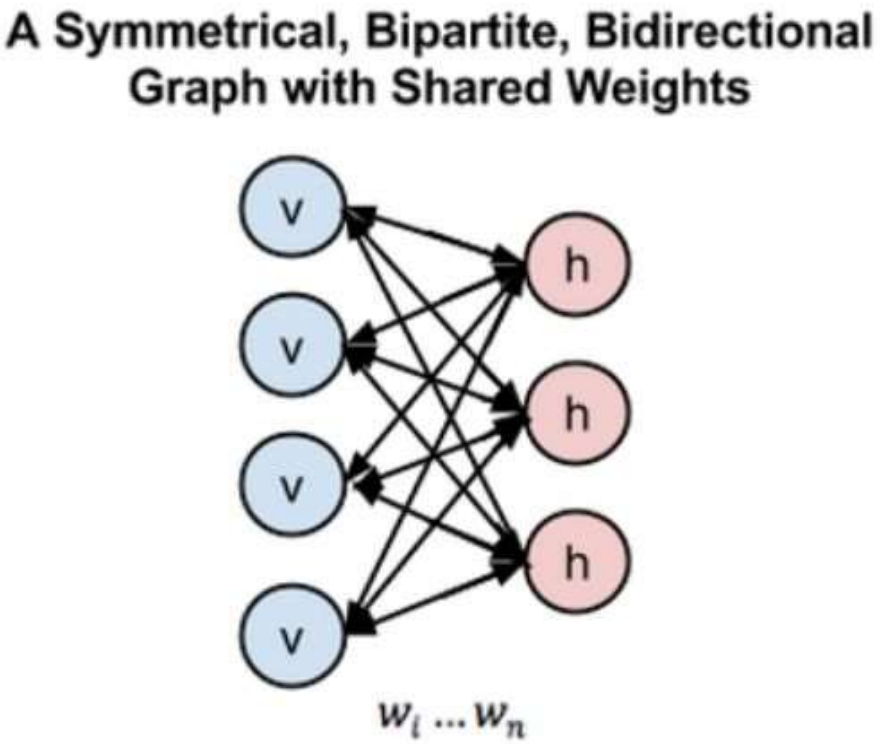
这种通过特征分组创建激活值集合序列，并对特征组进行分组的过程是特征层次结构的基础，通过这个过程，神经网络学到了更复杂的、更抽象的数据表征。

对于每一个新的隐藏层，权重都会通过迭代反复调整，直至该层能够逼近来自于前一层的输

入。这是贪婪的、逐层的、无监督的预训练。它不需要使用标签来改善网络的权重，这意味着我们可以在无标签的数据集上进行训练，而这些数据没有经过人工处理，这是现实中绝大多数的数据。通常，拥有更多数据的算法会产生更准确的结果，这也是深层学习算法崛起的原因之一。

因为这些权重早已接近数据的特征，所以在使用深度信念网络进行图像分类的时候，后续的监督学习阶段可以更简单地学习。尽管 RBM 有很多用途，但合适的权重初始化以方便以后的分类是其主要优点之一。从某种程度而言，它们完成了某种类似于反向传播的功能：它们很好地调整了权重，以对数据进行更好的建模。你可以说预训练和反向传播是达到相同目的的可替代方法。

为了在一个图中展示受限玻尔兹曼机，我们需要使用对称二分双向图表示：



对于那些对深入研究 RBM 结构感兴趣的人而言，它们是一种无向图模型，也被称作马尔科夫随机场。

代码实例：Stacked RBMS

GitHub 链接：

https://github.com/deeplearning4j/dl4j-examples/blob/master/dl4j-examples/src/main/java/org/deeplearning4j/examples/unsupervised/deepbelief/DeepAutoEncoderExample.java

参数 & K

变量 k 是运行对比散度（Contrastive Divergence）的次数。对比散度是用来计算梯度（该斜率表示网络权重与其误差之间的关系）的方法，没有这种方法，学习就无法进行。

在上面的例子中，你可以看到如何将 RBM 创建为具有更通用多层配置的层。在每个点处，你会发现一个可以影响深度神经网络结构和性能的额外参数。大多数这些参数都是在这里定义的。

参数初始化（weightInit 或者 weightInitialization）表示放大或者抑制到达每个节点的输入信号的系数的初始值。合适的权重初始化可以节省大量的训练时间，因为训练一个网络只不过是调整系数来传递最佳信号，从而使网络能够准确分类。

激活函数（activationFunction）是一组函数中的一个，用于确定每个节点处的激活阈值，高于阈值的信号可以通过，低于阈值的信号就被阻止。如果一个节点传递了一个信号，则它被「激活」。

优化算法（optimizationAlgo）指神经网络最小化误差或者找到最小误差轨迹的方式，它是一步一步调整参数的。LBFGS 是一种优化算法，它利用二阶导数来计算梯度的斜率，系数将沿着梯度的斜率进行调整。

正则化（regularization）方法（如 L2）有助于防止神经网络中的过拟合。正则化本质上会惩罚较大的系数，因为大系数意味着网络已经学会将结果锁定在几个高权值的输入上了。过强的权重会使网络模型在面对新数据的时候难以泛化。

显元/隐元（VisibleUnit/HiddenUnit）指神经网络的层。显元或者可见层，是输入到达的层，隐元或者隐藏层，是输入被结合成更复杂特征的层。这两种单元都有各自所谓的变换，在这里，可见层是高斯变换，隐藏层是整流线性单元，它们将来自它们对应层的信号映射到新的空间。

损失函数（lossFunction）是测量误差的方法，或者测量网络预测和测试集包含的正确的标签之间差距的方法。我们在这里使用的是 SQUARED_ERROR，它使所有的误差都是正值，因此可以被求和并反向传播。

学习率（learningRate，如 momentum）会影响神经网络在每次迭代中校正误差时调整系数的程度。这两个参数有助于确定网络将梯度降低到局部最优时的步长。较大的学习率会使网络学习得更快，并且可能越过最佳值。较小的学习率可能减慢学习，而且可能是低效的。

连续 RBM

连续 RBM 是受限玻尔兹曼机的一种形式，它通过不同类型的对比散度采样接受连续的输入（也就是比整数切割得更细的数字）。这允许 CRBM 处理图像像素或字数向量这类被归一化到 0 到 1 之间的小数的向量。

应该注意，深度学习网络的每一层都需要四个元素：输入、系数、偏置项以及变换（激活算法）。

输入是数值数据，是一个来自于前面层（或者原始数据）的向量。系数是通过每个节点层的特征的权重。偏置项确保部分节点无论如何都能够被激活。变换是一种额外的算法，它在数据通过每一层以后以一种使梯度（梯度是网络必须学习的）更容易被计算的方式压缩数据。

这些额外算法和它们的组合可以逐层变化。

一种有效的连续 RBM 在可见（或者输入）层上使用高斯变换，在隐藏层上使用整流线性单元（ReLU）变换。这在面部重建中特别有用。对于处理二进制数据的 RBM 而言，只需要进行二进制转换即可。

高斯变换在 RBM 的隐藏层上的表现不好。相反，使用 ReLU 变换能够表示比二进制变换更多的特征，我们在深度置信网络中使用了它。

总结 & 下一步工作

你可以将 RBM 的输出解释为百分比。每次重建的数字不为零，这是 RBM 学习输入的良好指示。

应当指出的是，RBM 并不能生成所有的浅层前馈网络中最稳定、最一致的结果。在很多情况下，密集层自编码器性能较好。事实上，业界正在转向变分自编码器和 GAN 等工具。

下一步，我们将会展示如何实现深度置信网络（https://deeplearning4j.org/deepbeliefnetwork.html），它由许多受限玻尔兹曼机堆叠而成。

原文链接：https://deeplearning4j.org/restrictedboltzmannmachine.html#params