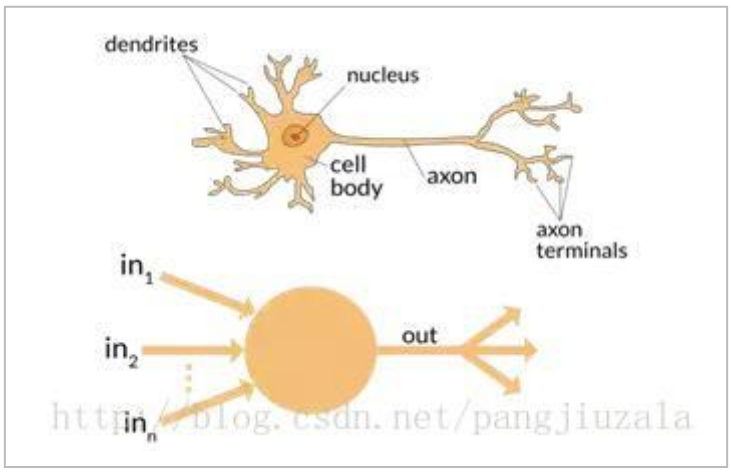
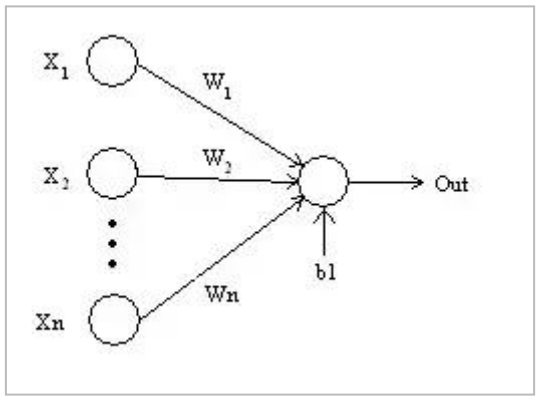
神经网络基础

1）神经元（Neuron）——就像形成我们大脑基本元素的神经元一样，神经元形成神经网络的基本结构。想象一下，当我们得到新信息时我们该怎么做。当我们获取信息时，我们一般会处理它，然后生成一个输出。类似地，在神经网络的情况下，神经元接收输入，处理它并产生输出，而这个输出被发送到其他神经元用于进一步处理，或者作为最终输出进行输出。



2）权重（Weights）——当输入进入神经元时，它会乘以一个权重。例如，如果一个神经元有两个输入，则每个输入将具有分配给它的一个关联权重。我们随机初始化权重，并在模型训练过程中更新这些权重。训练后的神经网络对其输入赋予较高的权重，这是它认为与不那么重要的输入相比更为重要的输入。为零的权重则表示特定的特征是微不足道的。

让我们假设输入为 a，并且与其相关联的权重为 W1，那么在通过节点之后，输入变为 a * W1



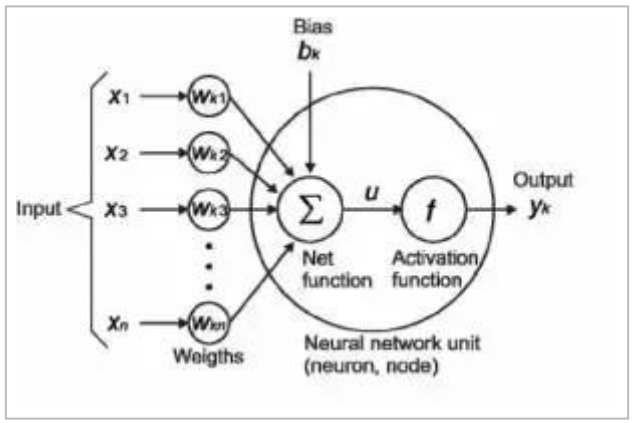
3）偏差（Bias）——除了权重之外，另一个被应用于输入的线性分量被称为偏差。它被加到权重与输入相乘的结果中。基本上添加偏差的目的是来改变权重与输入相乘所得结果的范围的。添加偏差后，结果将看起来像 a* W1 + 偏差。这是输入变换的最终线性分量。

4）激活函数（Activation Function）——一旦将线性分量应用于输入，将会需要应用一个非线性函数。这通过将激活函数应用于线性组合来完成。激活函数将输入信号转换为输出信号。应用激活函数后的输出看起来像 f（a * W1 + b），其中 f（ ）就是激活函数。

在下图中，我们将 “n” 个输入给定为 X1 到 Xn 而与其相应的权重为 Wk1 到 Wkn。我们有一个给定值为 bk 的偏差。权重首先乘以与其对应的输入，然后与偏差加在一起。而这个值叫做 u。

U =ΣW* X+ b

激活函数被应用于 u，即 f(u)，并且我们会从神经元接收最终输出，如 yk = f（u）。

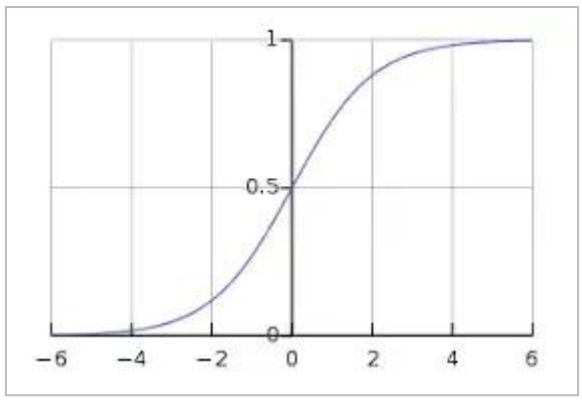


常用的激活函数

最常用的激活函数就是 Sigmoid，ReLU 和 softmax

a) Sigmoid——最常用的激活函数之一是 Sigmoid，它被定义为：

$$sigmoid(x) = 1/(1+e^{-x})$$

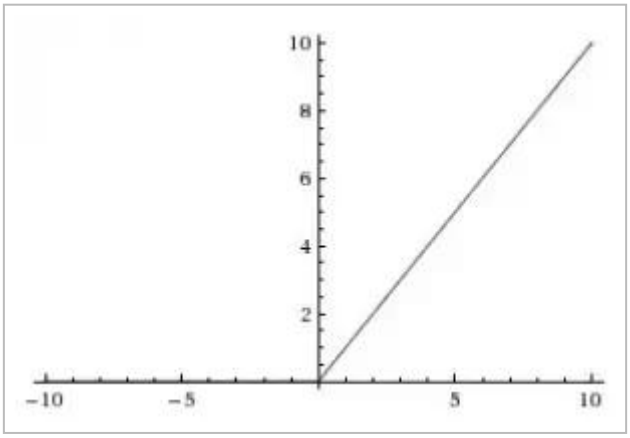


Sigmoid 变换产生一个值为 0 到 1 之间更平滑的范围。我们可能需要观察在输入值略有变化时输出值中发生的变化。光滑的曲线使我们能够做到这一点，因此优于阶跃函数。

b）ReLU（整流线性单位）——与 Sigmoid 函数不同的是，最近的网络更喜欢使用 ReLu 激活函数来处理隐藏层。该函数定义为：

$$f(x) = \max(x, 0).$$

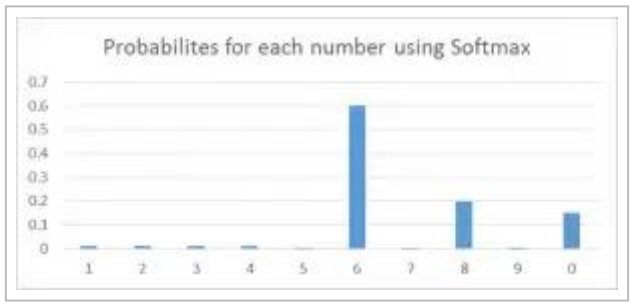
当 X>0 时，函数的输出值为 X；当 X<=0 时，输出值为 0。函数图如下图所示：



使用 ReLU 函数的最主要的好处是对于大于 0 的所有输入来说，它都有一个不变的导数值。常数导数值有助于网络训练进行得更快。

c） Softmax——Softmax 激活函数通常用于输出层，用于分类问题。它与 sigmoid 函数是很类似的，唯一的区别就是输出被归一化为总和为 1。Sigmoid 函数将发挥作用以防我们有一个二进制输出，但是如果我们有一个多类分类问题，softmax 函数使为每个类分配值这种操作变得相当简单，而这可以将其解释为概率。

以这种方式来操作的话，我们很容易看到——假设你正在尝试识别一个可能看起来像 8 的 6。该函数将为每个数字分配值如下。我们可以很容易地看出，最高概率被分配给 6，而下一个最高概率分配给 8，依此类推……

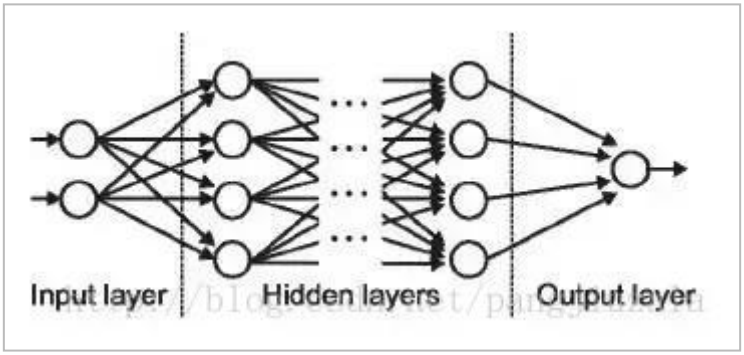


5）神经网络（Neural Network）——神经网络构成了深度学习的支柱。神经网络的目标是找到一个未知函数的近似值。它由相互联系的神经元形成。这些神经元具有权重和在网络训练期间根据错误来进行更新的偏差。激活函数将非线性变换置于线性组合，而这个线性组合稍后会生成输出。激活的神经元的组合会给出输出值。

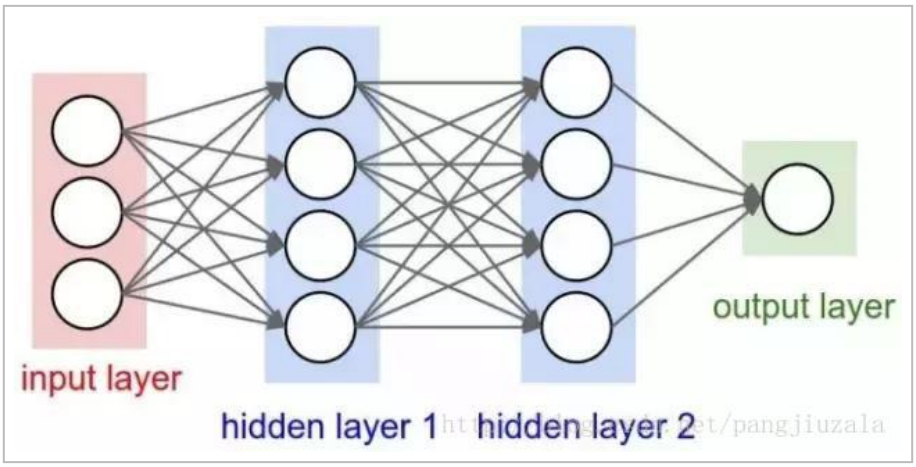
一个很好的神经网络定义——

“神经网络由许多相互关联的概念化的人造神经元组成，它们之间传递相互数据，并且具有根据网络” 经验 “调整的相关权重。神经元具有激活阈值，如果通过其相关权重的组合和传递给他们的数据满足这个阈值的话，其将被解雇; 发射神经元的组合导致 “学习”。

6）输入 / 输出 / 隐藏层（Input / Output / Hidden Layer）——正如它们名字所代表的那样，输入层是接收输入那一层，本质上是网络的第一层。而输出层是生成输出的那一层，也可以说是网络的最终层。处理层是网络中的隐藏层。这些隐藏层是对传入数据执行特定任务并将其生成的输出传递到下一层的那些层。输入和输出层是我们可见的，而中间层则是隐藏的。



7）MLP（多层感知器）——单个神经元将无法执行高度复杂的任务。因此，我们使用堆栈的神经元来生成我们所需要的输出。在最简单的网络中，我们将有一个输入层、一个隐藏层和一个输出层。每个层都有多个神经元，并且每个层中的所有神经元都连接到下一层的所有神经元。这些网络也可以被称为完全连接的网络。



8）正向传播（Forward Propagation）——正向传播是指输入通过隐藏层到输出层的运动。在正向传播中，信息沿着一个单一方向前进。输入层将输入提供给隐藏层，然后生成输出。这过程中是没有反向运动的。

9）成本函数（Cost Function）——当我们建立一个网络时，网络试图将输出预测得尽可能靠近实际值。我们使用成本／损失函数来衡量网络的准确性。而成本或损失函数会在发生错误时尝试惩罚网络。

我们在运行网络时的目标是提高我们的预测精度并减少误差，从而最大限度地降低成本。最优化的输出是那些成本或损失函数值最小的输出。

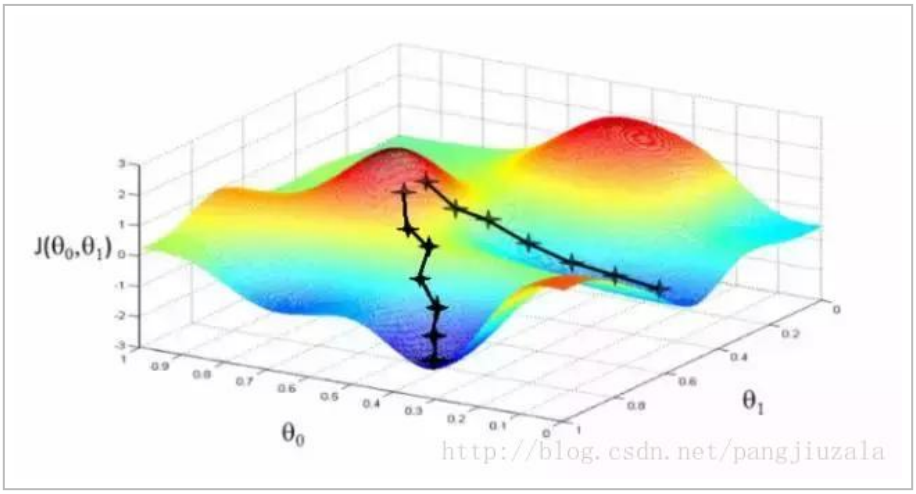
如果我将成本函数定义为均方误差，则可以写为：

C= 1/m ∑(y–a)^2，

其中 m 是训练输入的数量，a 是预测值，y 是该特定示例的实际值。

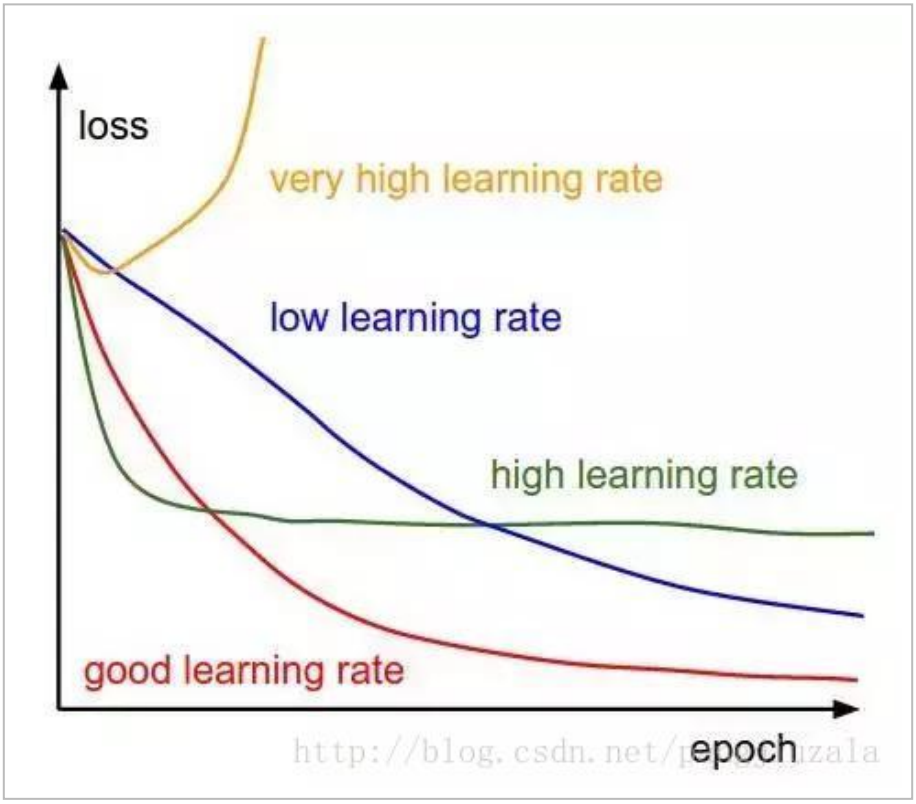
学习过程围绕最小化成本来进行。

10）梯度下降（Gradient Descent）——梯度下降是一种最小化成本的优化算法。要直观地想一想，在爬山的时候，你应该会采取小步骤，一步一步走下来，而不是一下子跳下来。因此，我们所做的就是，如果我们从一个点 x 开始，我们向下移动一点，即Δh，并将我们的位置更新为 x-Δh，并且我们继续保持一致，直到达到底部。考虑最低成本点。



在数学上，为了找到函数的局部最小值，我们通常采取与函数梯度的负数成比例的步长。

11）学习率（Learning Rate）——学习率被定义为每次迭代中成本函数中最小化的量。简单来说，我们下降到成本函数的最小值的速率是学习率。我们应该非常仔细地选择学习率，因为它不应该是非常大的，以至于最佳解决方案被错过，也不应该非常低，以至于网络需要融合。



12）反向传播（Backpropagation）——当我们定义神经网络时，我们为我们的节点分配随机权重和偏差值。一旦我们收到单次迭代的输出，我们就可以计算出网络的错误。然后将该错误与成本函数的梯度一起反馈给网络以更新网络的权重。 最后更新这些权重，以便减少后续迭代中的错误。使用成本函数的梯度的权重的更新被称为反向传播。

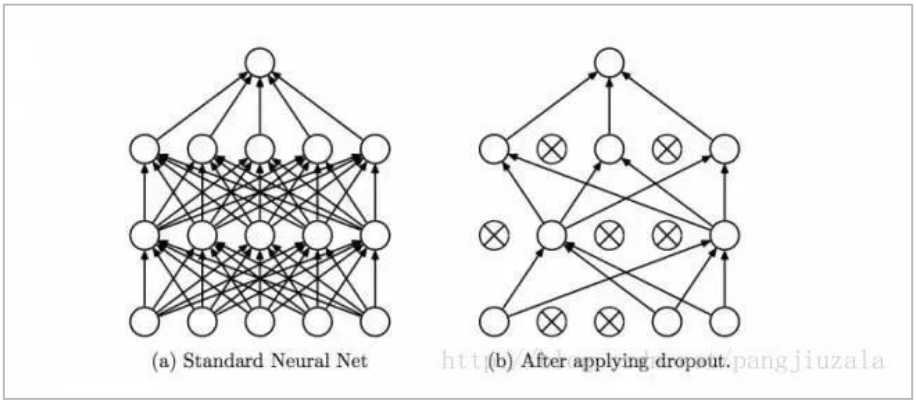
在反向传播中，网络的运动是向后的，错误随着梯度从外层通过隐藏层流回，权重被更新。

13）批次（Batches）——在训练神经网络的同时，不用一次发送整个输入，我们将输入分成几个随机大小相等的块。与整个数据集一次性馈送到网络时建立的模型相比，批量训练数据使得模型更加广义化。

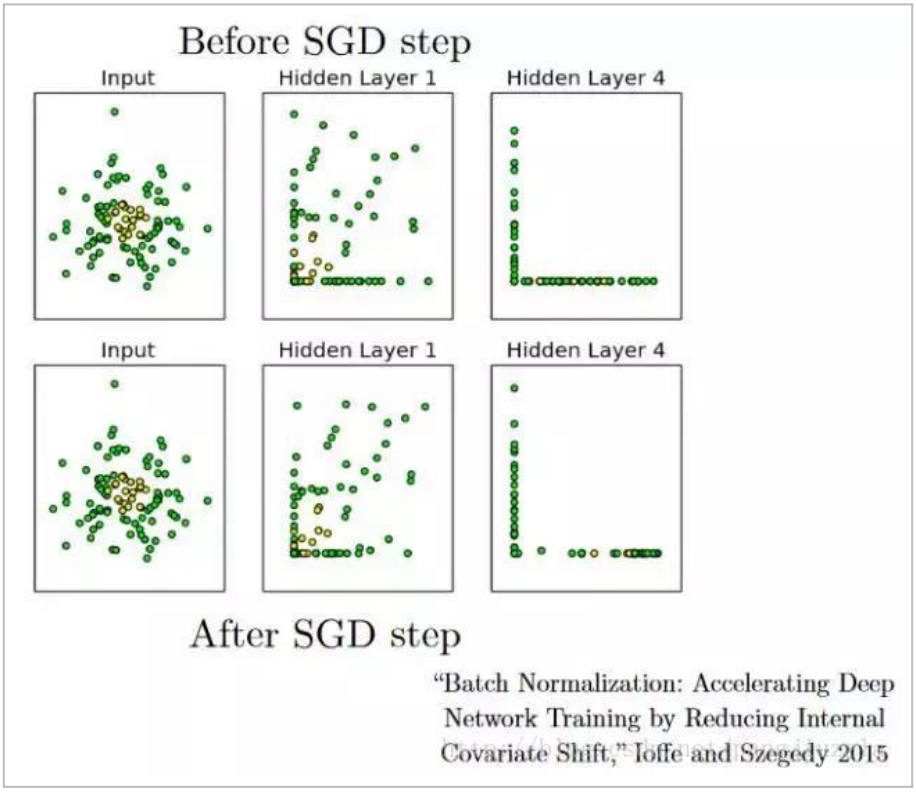
14）周期（Epochs）——周期被定义为向前和向后传播中所有批次的单次训练迭代。这意味着 1 个周期是整个输入数据的单次向前和向后传递。

你可以选择你用来训练网络的周期数量，更多的周期将显示出更高的网络准确性，然而，网络融合也需要更长的时间。另外，你必须注意，如果周期数太高，网络可能会过度拟合。

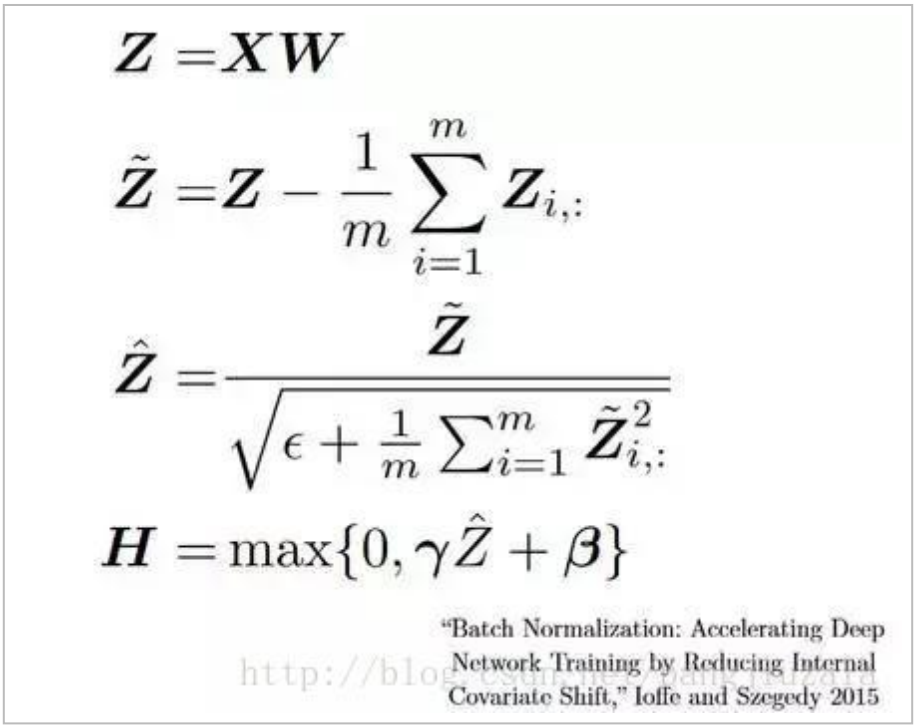
15）丢弃（Dropout）——Dropout 是一种正则化技术，可防止网络过度拟合套。顾名思义，在训练期间，隐藏层中的一定数量的神经元被随机地丢弃。这意味着训练发生在神经网络的不同组合的神经网络的几个架构上。你可以将 Dropout 视为一种综合技术，然后将多个网络的输出用于产生最终输出。



16）批量归一化（Batch Normalization）——作为一个概念，批量归一化可以被认为是我们在河流中设定为特定检查点的水坝。这样做是为了确保数据的分发与希望获得的下一层相同。当我们训练神经网络时，权重在梯度下降的每个步骤之后都会改变，这会改变数据的形状如何发送到下一层。



但是下一层预期分布类似于之前所看到的分布。 所以我们在将数据发送到下一层之前明确规范化数据。



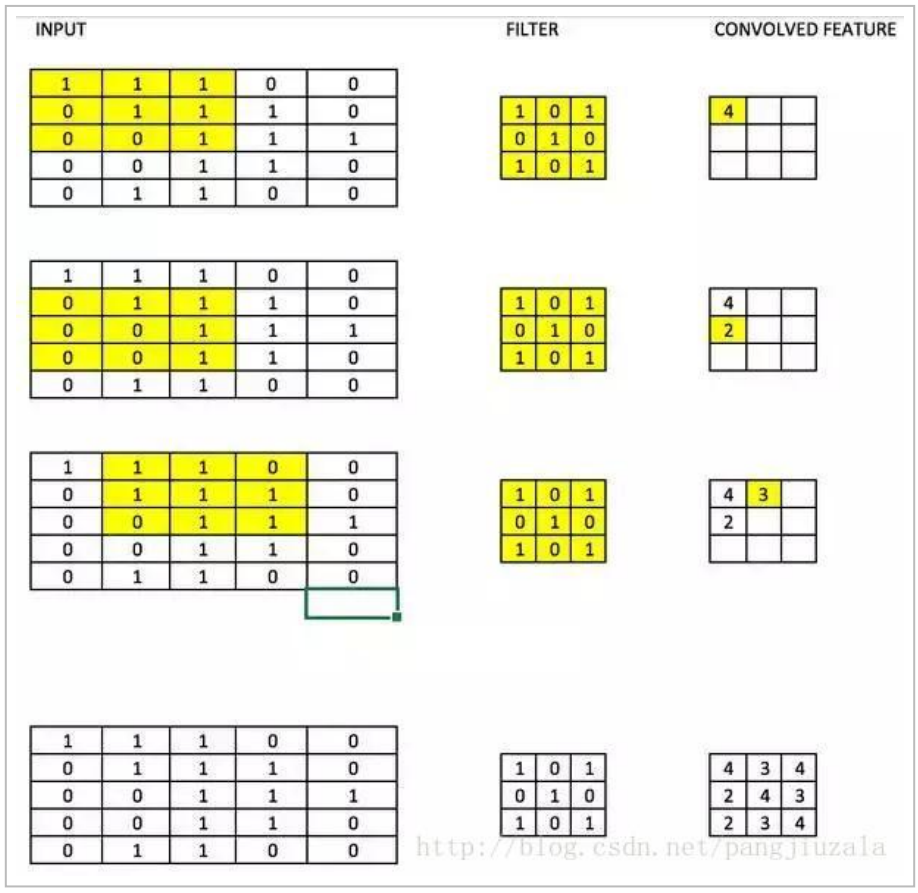
卷积神经网络

17）滤波器（Filters）——CNN 中的滤波器与加权矩阵一样，它与输入图像的一部分相乘以产生一个回旋输出。我们假设有一个大小为 28 * 28 的图像，我们随机分配一个大小为 3 * 3 的滤波器，然后与图像不同的 3 * 3 部分相乘，形成所谓的卷积输出。滤波器尺寸通常小于原始图像尺寸。在成本最小化的反向传播期间，滤波器值被更新为重量值。

参考一下下图，这里 filter 是一个 3 * 3 矩阵：

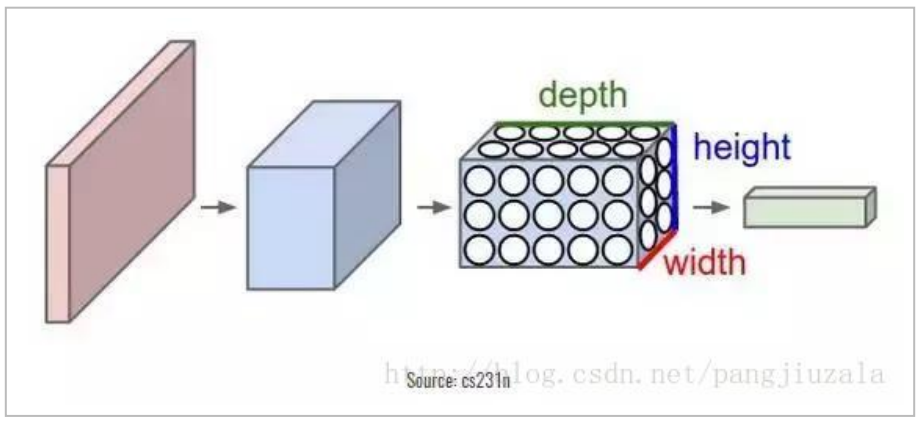
$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

与图像的每个 3 * 3 部分相乘以形成卷积特征。

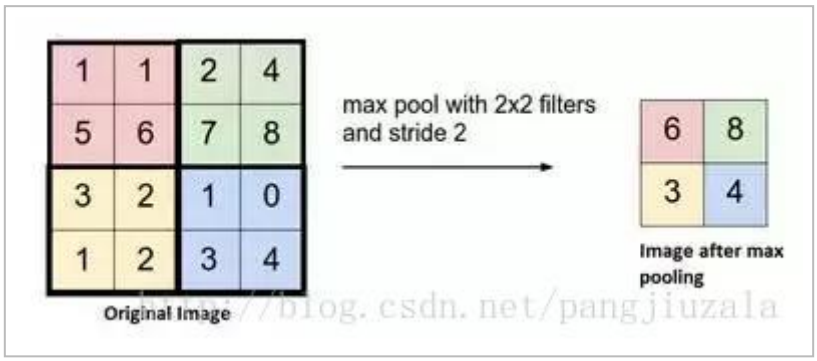


18）卷积神经网络（CNN）——卷积神经网络基本上应用于图像数据。假设我们有一个输入的大小（28 * 28 * 3），如果我们使用正常的神经网络，将有 2352（28 * 28 * 3）参数。并且随着图像的大小增加参数的数量变得非常大。我们 “卷积” 图像以减少参数数量（如上面滤波器定义所示）。当我们将滤波器滑动到输入体积的宽度和高度时，将产生一个二维激活图，给出该滤波器在每个位置的输出。我们将沿深度尺寸堆叠这些激活图，并产生输出量。

你可以看到下面的图，以获得更清晰的印象。



19）池化（Pooling）——通常在卷积层之间定期引入池层。这基本上是为了减少一些参数，并防止过度拟合。最常见的池化类型是使用 MAX 操作的滤波器尺寸（2,2）的池层。它会做的是，它将占用原始图像的每个 4 * 4 矩阵的最大值。



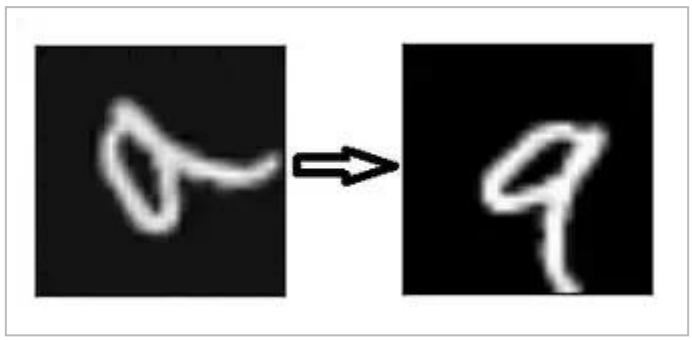
你还可以使用其他操作（如平均池）进行池化，但是最大池数量在实践中表现更好。

20）填充（Padding）——填充是指在图像之间添加额外的零层，以使输出图像的大小与输入相同。这被称为相同的填充。

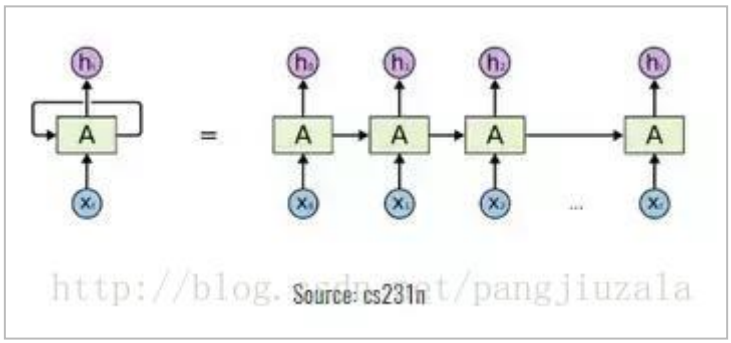
在应用滤波器之后，在相同填充的情况下，卷积层具有等于实际图像的大小。

有效填充是指将图像保持为具有实际或 “有效” 的图像的所有像素。在这种情况下，在应用滤波器之后，输出的长度和宽度的大小在每个卷积层处不断减小。

21）数据增强（Data Augmentation）——数据增强是指从给定数据导出的新数据的添加，这可能被证明对预测有益。例如，如果你使光线变亮，可能更容易在较暗的图像中看到猫，或者例如，数字识别中的 9 可能会稍微倾斜或旋转。在这种情况下，旋转将解决问题并提高我们的模型的准确性。通过旋转或增亮，我们正在提高数据的质量。这被称为数据增强。



循环神经网络



22）循环神经元（Recurrent Neuron）——循环神经元是在 T 时间内将神经元的输出发送回给它。如果你看图，输出将返回输入 t 次。展开的神经元看起来像连接在一起的 t 个不同的神经元。这个神经元的基本优点是它给出了更广义的输出。

23）循环神经网络（RNN）——循环神经网络特别用于顺序数据，其中先前的输出用于预测下一个输出。在这种情况下，网络中有循环。隐藏神经元内的循环使他们能够存储有关前一个单词的信息一段时间，以便能够预测输出。隐藏层的输出在 t 时间戳内再次发送到隐藏层。展开的神经元看起来像上图。只有在完成所有的时间戳后，循环神经元的输出才能进入下一层。发送的输出更广泛，以前的信息保留的时间也较长。

然后根据展开的网络将错误反向传播以更新权重。这被称为通过时间的反向传播（BPTT）。

24）消失梯度问题（Vanishing Gradient Problem）——激活函数的梯度非常小的情况下会出现消失梯度问题。在权重乘以这些低梯度时的反向传播过程中，它们往往变得非常小，并且随着网络进一步深入而 “消失”。这使得神经网络忘记了长距离依赖。这对循环神经网络来说是一个问题，长期依赖对于网络来说是非常重要的。

这可以通过使用不具有小梯度的激活函数 ReLu 来解决。

25）激增梯度问题（Exploding Gradient Problem）——这与消失的梯度问题完全相反，激活函数的梯度过大。在反向传播期间，它使特定节点的权重相对于其他节点的权重非常高，这使

得它们不重要。这可以通过剪切梯度来轻松解决，使其不超过一定值。

全文完