感知器

一个感知器接收几个二进制输入，$x_1, x_2, \ldots$,并产生一个二进制输出：

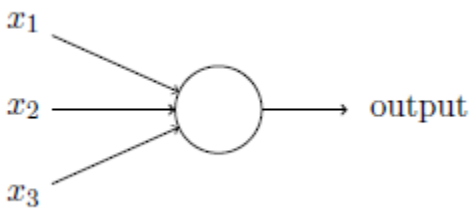

Figure 1

如图 1 所示感知器有三个输入，x_1, x_2, x_3。通常可以有更多或更少的输入。引入权重了计算输出，权重$w_1, w_2, \ldots$,表示相应输入对于输出重要性，神经元输出 0 或 1 由总和$\sum_j w_j x_j$小于或大于一个阈值决定。即

$$\text{output} = \begin{cases} 0 & if \ \sum_j w_j x_j \leq threshold \\ 1 & if \ \sum_j w_j x_j \geq threshold \end{cases} \qquad (1)$$

随着权重和阈值的变化，你可以得到不同的决策模型。

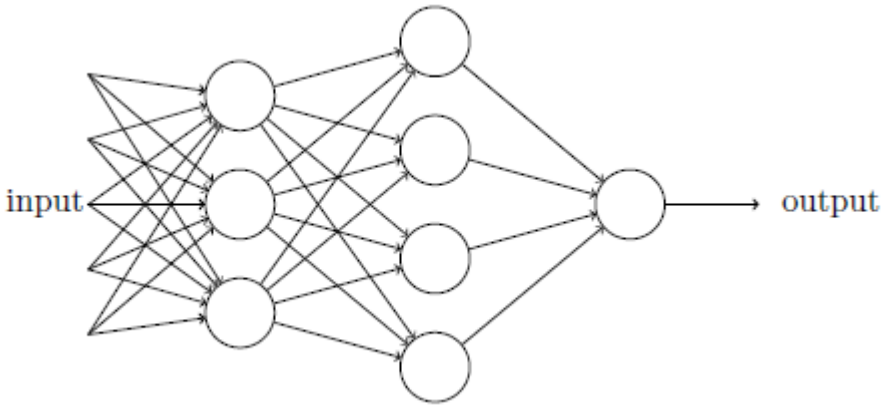

Figure 2

如图 2 所示，第一层感知器通过权衡输入做出三个简单的决策，二第二层感知器可以比第一层做出更复杂和抽象的决策。以这种方式，一个多层的感知器络可以从事复杂巧妙的决策。

简化感知器的数学描述$\sum_j w_j x_j$改写为点积。$w \cdot x = \sum_j w_j x_j$，$w$和$x$对应权重和输入向量。把阈值用偏置$b = -threshold$代替。

我们可以设计学习算法，能够自动调整人工神经元的权重和偏置。这种调整

可以响应外部的刺激，而不需要程序员的直接干预。

S 型神经元

如果对权重（或者偏置）的微小的改动真的能够仅仅引起输出的微小变化，（神经网络迭代的基础）那我们可以利用这一事实来修改权重和偏置，让我们的网络能够表现得像我们想要的那样。但我们用感知器时，单个感知器的一个权重或偏置都可能引起感知器输出的完全翻转。

S 元神经元（如图 1）与感知器类似，但是修改权重和偏置的微小改动只会引起输出的微小变化。S 型神经元的输入可以对应任何实数，对应每个输入有权重$x_1, x_2, \ldots$,和一个总偏置b_0。输出为$\sigma(w \cdot x + b)$，这里σ被称为 S 形函数

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad (2)$$

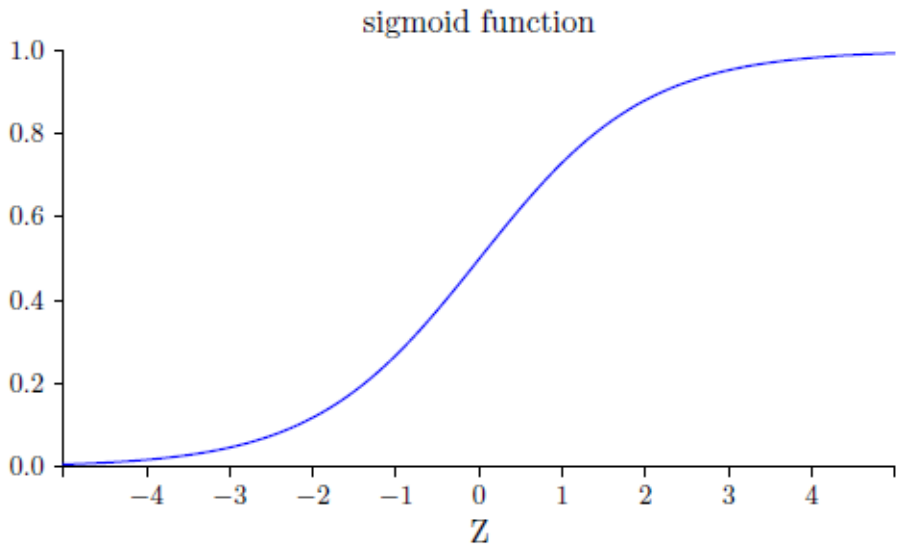

Figure 3

σ函数（逻辑函数）的代数形式如图 3 所示。这样我们可以得到一个平滑的感知器，σ的平滑意味着权重和偏置的微小变化，即Δw_j和Δb，会产生一个输出变化$\Delta output$，可以近似的表示为

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b \qquad (3)$$

神经网络的架构

假设我们有一个如图 4 所示的网络：

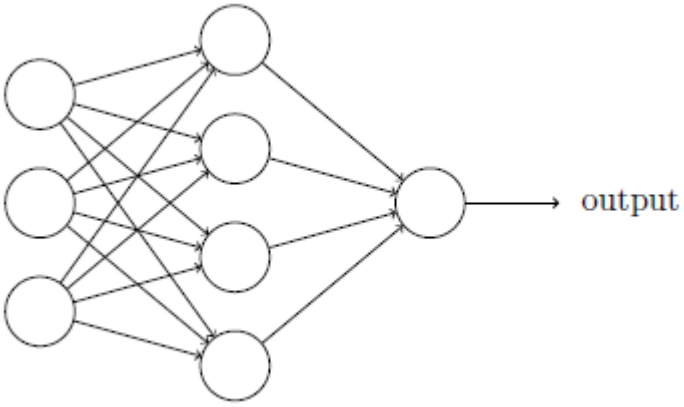

Figure 4

这个网络最左边的成为输入层，最右边的即输出层。中间的层既不是输出也不是输入，被称为隐藏层（可以有多个隐层），如图 6。

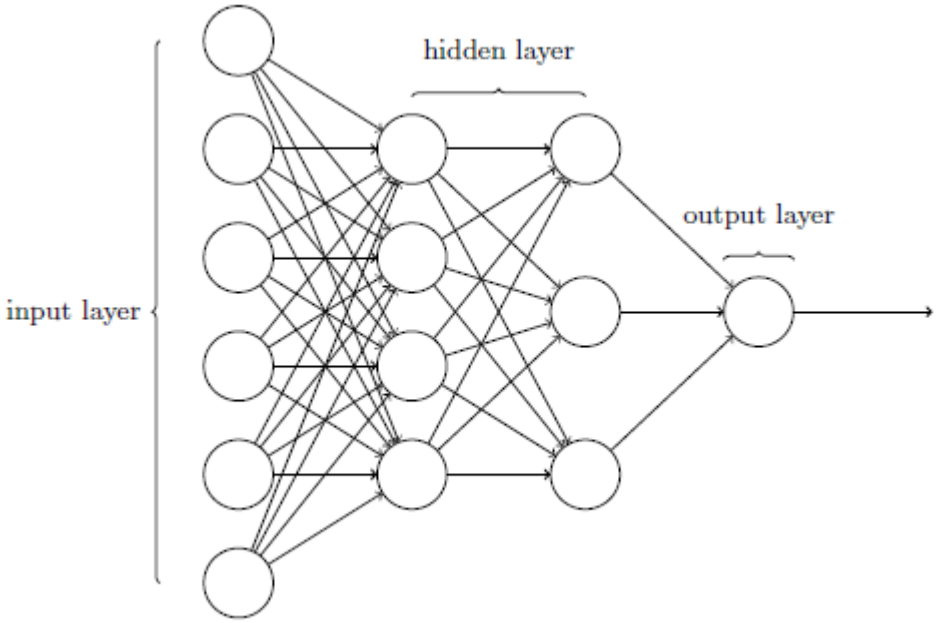

Figure 5

这种多层网络有时被称为多层感知器或者 MLP（但其实是由 S 型神经元构成的）。

像这种以上一层的输出作为下一层的输入的网络被称为前馈神经网络，网络中没有回路，信息总是向前传播。有反馈回路的神经网络模型被称为递归神经网络。

梯度下降算法

我们希望有一个算法，能够找到权重和偏置，使网络所有的输出$y(x)$，能够拟合训练输入x，定义代价函数：

$$\mathbf{C}(\boldsymbol{w},\boldsymbol{b}) = \frac{\mathbf{1}}{\mathbf{2}\boldsymbol{n}}\sum_{\boldsymbol{x}}\|\boldsymbol{y}(\boldsymbol{x}) - \boldsymbol{a}\|^{\mathbf{2}} \qquad (4)$$

这里w表示网络中权重的集合，b是所有的偏置，n是训练数据的个数，a表示当输入为x时的输入向量。

我们将 C 称为二次代价函数，也没成为均方误差或者 MSE。我们训练算法的目的是最小化 C。

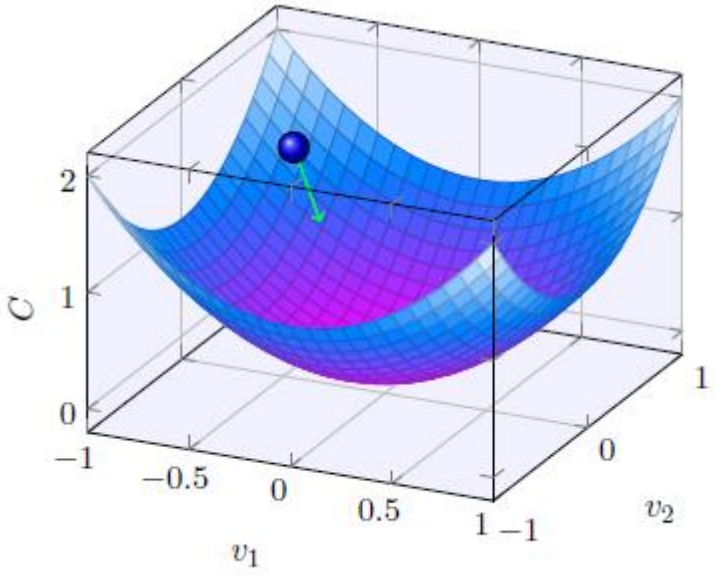

Figure 6

如图 6 所示，我们要求一个函数的最小值，想象一个小球向下滚动直到谷底。假设C是只有$v1$和$v2$两个变量的函数，当小球在$v1$和$v2$方向分别移动一个很小的量时即$\Delta v1$和$\Delta v2$，微积分得到C的变化

$$\Delta C \approx \frac{\partial C}{\partial v1}\Delta v1 + \frac{\partial C}{\partial v2}\Delta v2 \qquad (5)$$

我们要找到一种选择$\Delta v1$和$\Delta v2$的方法使ΔC为负，即让小球向下滚动。定义$\Delta v \equiv (\Delta v1, \Delta v2)^T$，也定义$C$的梯度为偏导数的向量$(\frac{\partial C}{\partial v1}, \frac{\partial C}{\partial v2})^T$。

$$\text{即梯度向量}\nabla C = (\frac{\partial C}{\partial v1}, \frac{\partial C}{\partial v2})^T \qquad (6)$$

$$\text{重写公式：}\Delta C \approx \nabla C \cdot \Delta v \qquad (7)$$

我们要使ΔC为负数，假设$\Delta v = -\eta\nabla C$。其中η为一个很小的正数（称为学习速率）。$\Delta C \approx -\eta\nabla C \cdot \nabla C = -\eta\|\nabla C\|$，所以就保证了$\Delta C < 0$。

$$v \to v' = v - \eta \nabla C \qquad (8)$$

总结梯度下降算法的工作原理就是反复计算梯度∇C，然后沿着相反的方向移动。

神经网络中学习的思想就是利梯度下降算法去寻找能使得方程（4）的代价取得最小值的权重w_k和偏置b_l，所以用权重w_k和偏置b_l来代替v，梯度∇C则有相应的分量$\partial C/\partial w_k$和$\partial C/\partial b_l$。用分量重新规则：

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k} \qquad (9)$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l} \qquad (10)$$

随机梯度下降

通过随机选取小量训练输入样本来计算∇C_x，进而估算∇C。通过计算少量样本的平均值我们可以快速得到一个对于实际梯度∇C的很好的估算，这有助于加速梯度下降，进而加速学习过程。

随机梯度下降通过随机选取小量的m个训练输入来工作。我们将这些随机的训练输入标记为$X_1, X_2, \ldots, X_m$，并把它们称为一个小批量数据（mini-batch）。

假设w_k 和b_l 表示我们神经网络中权重和偏置。随即梯度下降通过随机地选取并训练输入的小批量数据来工作:

$$w_k \to w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{x_j}}{\partial W_k} \qquad (11)$$

$$b_l \to b_l' = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{x_j}}{\partial b_l} \qquad (12)$$

然后我们再挑选另一随机选定的小批量数据去训练。直到我们用完了所有的训练输入，这被称为完成了一个训练迭代期（epoch）。然后我们就会开始一个新的训练迭代期。

反向传播算法

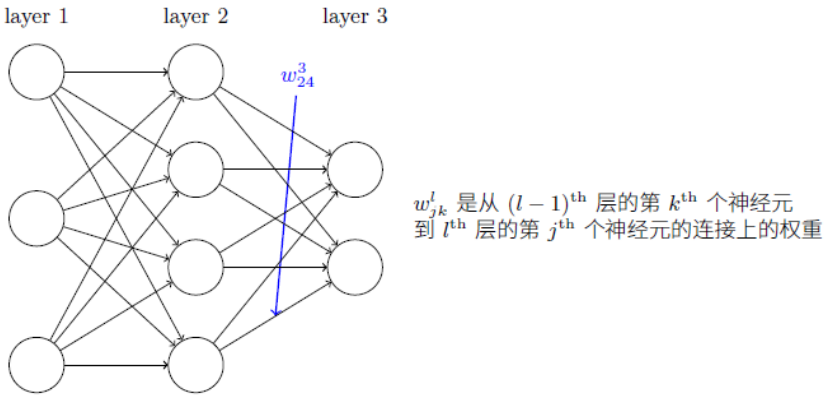

Figure 7

l^{th} 层第j^{th}个的激活值a_j^l和$(l-1)^{th}$层的激活值通过方程关联：

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \qquad (13)$$

我们对每一层l都一个权重矩阵w^l，对每一层l，定义一个偏置向量b^l。

$$a^l = \sigma(w^l a^{l-1} + b^l) \qquad (14)$$

计算a^l的过程中，我们计算了中间量$z^l = w^l a^{l-1} + b^l$。我们称z^l为l层神经元的带权输入。

反向传播算法

1、输入x：为输入层设置对应的激活值a^1

2、前向传播：对每个$l = 2,3,\dots,L$计算相应的$z^l = w^l a^{l-1} + b^l$和$a^l = \sigma(z^l)$

3、输出层误差δ^L：计算向量$\delta^L = \nabla_a C \odot \sigma'(z^l)$

（$\nabla_a C$定义为一个向量，其元素为偏导数$\partial C/\partial a_j^L$，。举个例子，在二次代价函数方程（4）时，我们有$\nabla_a C = (a^L - y)$

$\sigma'(z^l)$为$\partial a_k^L/\partial z_j^L$,实际上是激活函数例如$\sigma(z) = \frac{1}{1+e^{-z}}$的偏导数）

4、反向传播误差：对每个$l = L-1, L-2, \dots, 2$，计算$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$

5、输出：代价函数的梯度由$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$和$\frac{\partial C}{\partial b_j^l} = \delta_j^l$

特别地，给定一个大小为m 的小批量数据，下面的算法在这个小批量数据的基础上应用一步梯度下降学习算法:

1、输入训练样本的集合

2、对每个训练样本x：设置相应的输入激活$a^{x,1}$，并执行下面的步骤：

前向传播：对于每$l = 2,3,\dots,L$计算$z^{x,l} = w^l a^{x,l-1} + b^l$和$a^{x,l} = \sigma(z^{x,l})$

输出误差$\delta^{x,L}$：计算向量$\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$

反向传播误差：对每个$l = L-1, L-2, \dots, 2$计算$\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$

3、梯度下降：对每个$l = L-1, L-2, \dots, 2$根据$w^l \to w^l - \frac{\eta}{m}\sum_x \delta^{x,l}\,(a^{x,l-1})^T$和$b^l \to b^l - \frac{\eta}{m}\sum_x \delta^{x,l}$

交叉熵代价函数

$$C = -\frac{1}{n}\sum_x [y ln a + (1-y)\, ln(1-a)] \qquad (15)$$

将交叉熵看做代价函数有两点原因。第一，它是非负的，$C > 0$。第二，如果对于所有的训练输入x，神经元实际的输出接近目标值，那么交叉熵将接近 0。

1 输出层误差$\delta^L = a^L - y$

2 输出：代价函数的梯度由$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$和$\frac{\partial C}{\partial b_j^l} = \delta_j^l$

其他不变

柔性最大值 softmax

第L层第j个神经元的激活函数为：

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_j^L}} \qquad (16)$$

过度拟合和规范化

1、过度拟合

我们有 training_data（训练集）、test_data（测试集）和 validation_data（验证集）。我们使用 validation_data 来防止过度拟合。。我们在每个迭代期的最后都计算在 validation_data 上的分类准确率。一旦分类准确率已经饱和，就停训练，这个策略被称为提前停止。

为什么用 validation_data 而不是 test_data 来防止过度拟合：因为如果我们的超参数是基于 test_data 的话，可能最终我们会得到过度拟合于 test_data 的超参数，网络性能不能泛化到其他数据集上。我们借助 validation_data 训练超参数，一旦获得想要的超参数，最终使用 test_data 来进行准确率测量。这种寻找好的超参数的.法有时候被称为 hold out 法。

2、规范化

权重衰减（L2 规范化）

规范化的交叉熵：

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2 \qquad (17)$$

其中C_0是前面的代价函数，如二次代价函数或交叉熵代价函数。直觉上，规范化的效果是让网络倾向学习小一点的权重，大的权重只有能够给出代价函数第一项足够的提升时才被允许。规范化可以当做一种寻找小的权重和最小化原始的代价函数之间的折中。这两部分相对的重要性由λ的值来控制：λ越小，越偏向于最小化原始代价函数，反之，倾向于小的权重。对方程（17）求权重和偏置的偏导数：

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b}$$

所以

$$b \to b - \frac{\eta}{m}\sum_x \frac{\partial C_x}{\partial b}$$

$$w \to w - \eta\frac{\partial C_0}{\partial w} - \frac{\eta\lambda}{n} w$$

$$= \left(1 - \frac{\eta\lambda}{n}\right) w - \eta\frac{\partial C_0}{\partial w}$$

因为通过 m 个小批量训练样本来估计：

$$b \to b - \frac{\eta}{m}\sum_x \frac{\partial C_x}{\partial b} \qquad (18)$$

$$w \to \left(1 - \frac{\eta\lambda}{n}\right) w - \frac{\eta}{m}\sum_x \frac{\partial C_x}{\partial w} \qquad (19)$$

```
self.weights = [(1-eta*(lmbda/n))*w-(eta/len(mini_batch))*nw
                for w, nw in zip(self.weights, nabla_w)]
self.biases = [b-(eta/len(mini_batch))*nb
               for b, nb in zip(self.biases, nabla_b)]
```

规范化能减少过度拟合。

L1 规范化：这个方法在未规范化的代价函数上加上一个权重绝对值的和。

$$C = C_0 + \frac{\lambda}{n}\sum_w |w| \qquad (20)$$

对方程（20）求导，我们有：

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} sgn(w)$$

其中$sgn(w)$是w的正负号，即w为正数时为$+1$，w为负数时为-1。对 L1 规范化的网络进行更新的规则：

$$w \to w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta\lambda}{n} sgn(w) \qquad (21)$$

弃权（Dropout）

弃权是一种相当激进的技术。和 L1、L2 规范化不同，弃权技术并不依赖对代价函数的修改。在弃权中，我们改变网络本身。

假设我们有一个训练数据x 和对应的目标输出y。通常我们会通过网络向前传播x，然后反向传播来确定对梯度的贡献。使用弃权技术，我们会随机（临时）地删除网络中一半的隐藏神经元，同时让输入层和输出层的神经元保持不变。在一个小批量的数据上对有关权重和偏置更新，然后重复这个过程。

通过不断重复，我们的网络会学到一个权重和偏置的集合。当然，这些权重和偏置也是在一半的隐藏神经元被弃权的情形下学到的。当我们实际运行整个网络时，是指两倍的隐藏神经元将会被激活。为了补偿这个，我们将从隐藏层神经元的权重减半。

权重初始化

之前的方式就是根据独立高斯随机变量来选择权重和偏置，其被归一化为均值为 0，标准差 1。更好的方法是我们使用均值为 0 标准差为$1/\sqrt{n_{in}}$的高斯随机

分布来初始化权重，而偏置的初始化还用值为 0，标准差 1 的随机高斯分布。

```
self.biases = [np.random.randn(y, 1) for y in self.sizes[1:]]
self.weights = [np.random.randn(y, x)/np.sqrt(x)
                for x, y in zip(self.sizes[:-1], self.sizes[1:])]
```

神经网络的超参数选择

学习速率η，规范化参数λ、$minibatch$大小等等超参数选择的方法。

学习速率：首先，选择 η的阈值的估计。这个估计并不需要太过精确。你可以估计这个值的量级，比如说从 0.01 开始。如果代价在训练的若干回合开始下降，你就可以逐步地尝地尝试$\eta = 0.1,1.0, \ldots$,直到你找到一个 η的值使得在开始若干回合代价就开始震荡或者增加。相反，如果代价在 = 0:01 时就开始震荡或者增加，那就尝试$\eta = 0.001,,0.0001,..$,直到找到。

规范化参数：开始时不包含规范化（$\lambda = 0.0$），确定η的值。使用已经确定的η，使用验证数据来选择好的λ，然后根据验证集的性能按照因子 10 增加或减少其值。改进λ后可以再返回优化η。

迭代期数量：在验证集上计算准确率，使用早停止技术。

小批量数据大小：$minibatch$的大小选择其实是一个相对独立的超参数，可以尝试不同的$minibatch$，选择性能提升最快的$minibatch$

自动技术进行调参：通常技术是网格搜索（grid search）在 James Bergstra 和 Yoshua Bengio 2012 年的论文中给出了综述。

随机梯度下降的变化形式

Hessian（海森矩阵）技术
基于 momentum 的梯度下降

人工神经元的其他模型

tanh神经元

使用双曲正切函数代替了 S 形函数。输入为x，权重向量为w，偏置为b的$tanh$神经元的输出是：

$$tanh(w \cdot x + b) \qquad (22)$$

$tanh$函数的定义为:

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

对于 S 形函数$\sigma(z) = \frac{1+\tanh(z/2)}{2}$。所以$tanh$仅仅是 S 函数的按比例变化版本。

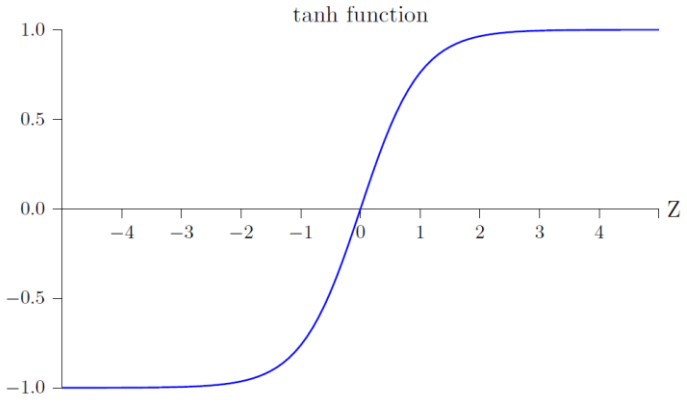

Figure 8

修正线性神经元（RLU）

输入为x，权重向量为w，偏置为b的 RLU 神经元的输出是：

$$\max(0, w \cdot x + b) \qquad (23)$$

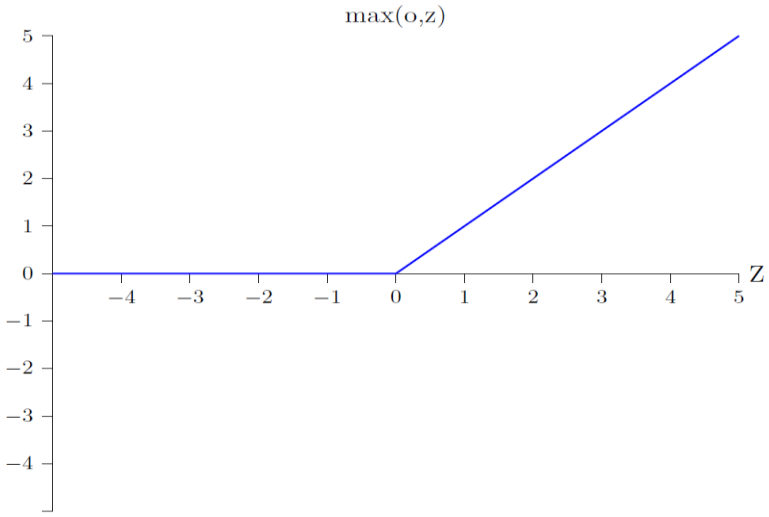

Figure 9

显然，RLU 神经元和sigmoid和tanh都不一样，但是 RLU 也能够计算任何函数。

卷积神经网络

卷积神经网络采用了三种基本概念：局部感受野（local receptive fields），共享权重（shared weight）和混合（pooling）。

局部感受野：在之前看到的全连接层的网络中，输入被描述成纵向排列的神经元。但在一个卷积神经网络中，不会把每个输入像素连接到每个隐藏神经元。确切的说，第一个隐藏层中的每个神经元会连接到输入神经元的一个小区域。

这个输入图像的区域被称为隐藏神经元的局部感受野。

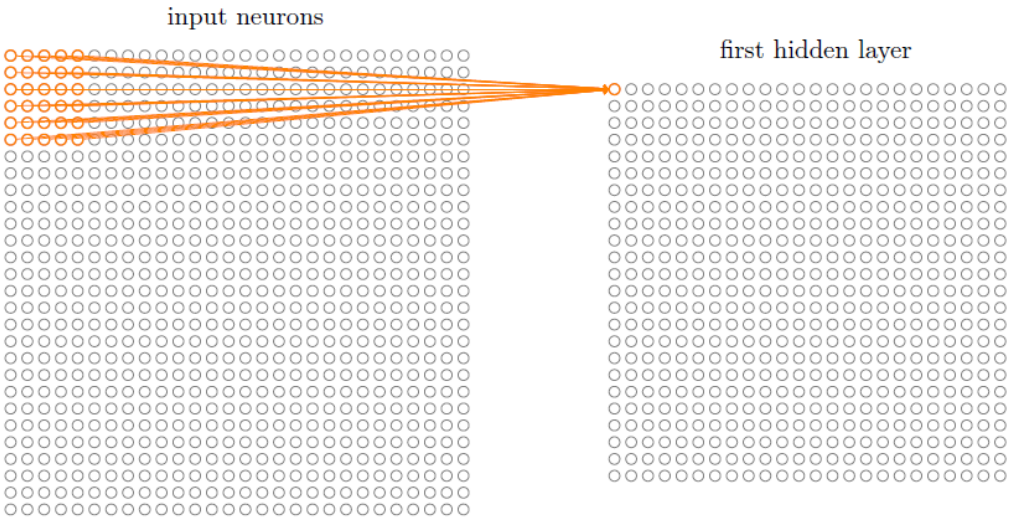

Figure 10

然后我们在整个输入神经元上交叉移动局部感受野。如向右一个神经元移动感受野，连接到第二个隐藏神经元

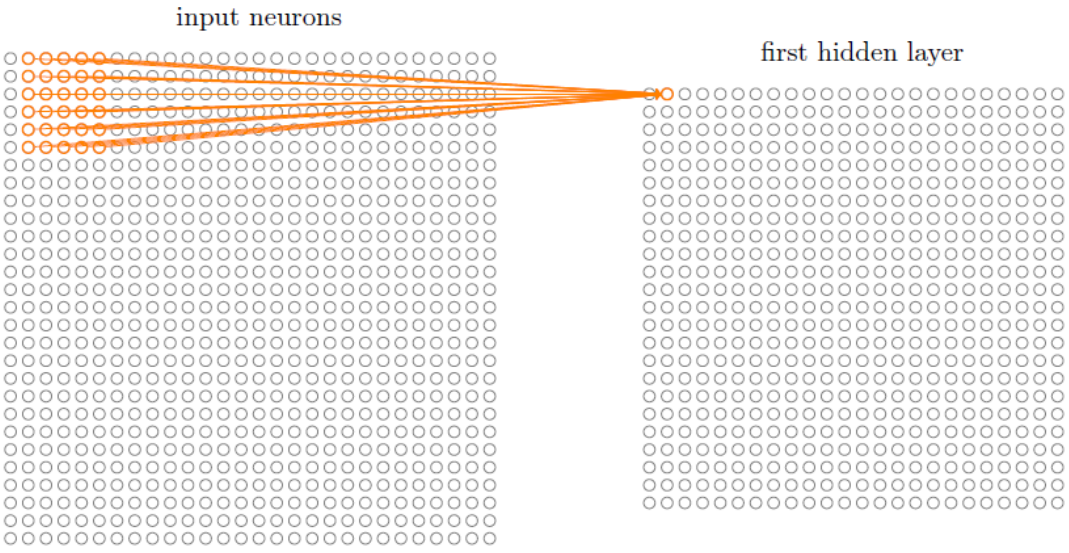

Figure 11

有时候会使用不同的跨距。例如向右或向下移动 2 个单位的神经元。

共享权重和偏置：如上图5×5的感受野，有一个5×5的权重矩阵和偏置b，而隐藏层的每一个神经元所对应的感受野的权重矩阵和偏置都相同。

对于第j, k个隐藏神经元，输出为：

$$\sigma(b + \sum_{l=0}^{4} \sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}) \qquad (24)$$

因此我们有时候把从输入层到隐藏层的映射称为一个特征映射。我们把定义特征映射的权重称为共享权重，偏置称为共享偏置。共享权重和偏置经常被称为一个卷积核或者滤波器。

共享权重和偏置的一个很大的优点是大大减少了参与卷积网络的参数。

混合层：除了卷积层，卷积神经网络还包括混合层（pooling layers）。混合层通常是紧接着在卷层之后使用。它要做的是简化从卷层输出的信息。

一个混合层取得从卷积层输出的每一个特征映射并且为它们准备一个凝缩的特征映射。例如，混合层的每个单元可能概括前一层的一个2×2的区域。常见的混合程序的是最大值混合（max-pooling）。在最大值混合中，一个混合单元简单地输出其2×2输入区域的最大激活值。

卷积层通常包含超过一个特征映射。我们将最大值混合分别应用于每一个特征映射。所以如果有三个特征映射，组合在一起的卷积层和最大值混合层看起来像这样：

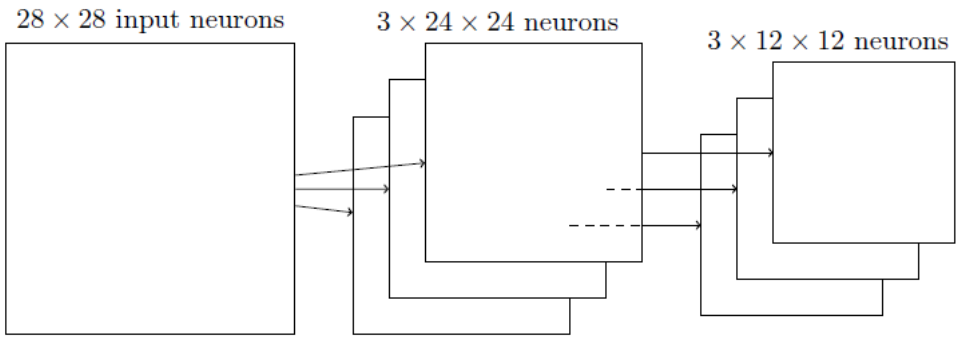

Figure 12

另一种混合技术是 L2 混合，取2×2区域中激活值的平方和的平方根，而不是最大值