神经网络的入门、搭建与应用

作者：殷和义
邮箱：chinayinheyi@163.com
2018 年 4 月 28 日

注：1. 以下内容仅供大家学习参考，禁止用于商业目的，否则后果自负。

2. 部分内容来自本人的学术论文，禁止抄袭用在其它地方。

3. 如有错误，欢迎指正。

目录

第一部分：神经网络入门

1 什么是神经网络

人工神经网络（Artificial Neural Network，ANN）是 20 世纪 80 年代以来在人工智能领域兴起的研究热点。最早的神经网络起源于 1943 年生理学家 McCulloch 和数学家 Pitts 提出的第一个网络模型——M-P 模型，后来经过几十年的发展，已经取得了巨大的成就。神经网络的强大之处在于它卓越的学习能力，文献[1]和[2]已经证明一个足够大的三层的神经网络就可以无限接近地拟合任意复杂函数。

[1] Cybenko G. Approximation by superpositions of a sigmoidal function[J]. Mathematics of control, signals and systems, 1989, 2(4): 303-314.

[2] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators[J]. Neural networks, 1989, 2(5): 359-366.

2 神经网络的结构

首先介绍单个神经元的结构图，如图 2-1 所示。

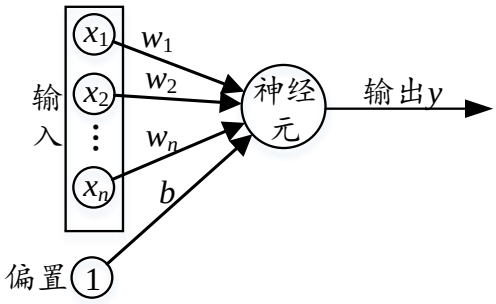

图 2-1 神经元的结构图

输出 y 可以表示为：

$$y = f\left(x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots + x_n \cdot w_n + b\right) \tag{2-1}$$

式中，$f(\ )$表示激活函数。

利用单一的神经单元，我们可以搭建一个多层的深度神经网络。例如，图 1-2 给出了一个三层的前馈神经网络（没有画出偏置）。

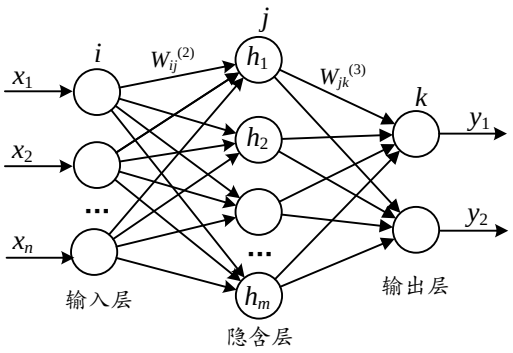

图 2-2 前馈神经网络

在多层的深度神经网络中，通常把第一层称作输入层，最后一层称为输出层，中间的称为隐含层。

在图 2-2 的三层神经网络中，给定输入 $\boldsymbol{X}$ 时，隐含层第 i 个单元和输出层第 j 个单元分别计算：

$$h_i = f(w_{1i}^{(2)}x_1 + w_{2i}^{(2)}x_2 + \cdots + w_{ni}^{(2)}x_n + b_i^{(2)}) \tag{2-2}$$

$$y_j = f(w_{1j}^{(3)}h_1 + w_{2j}^{(3)}h_2 + \cdots + w_{mj}^{(3)}h_m + b_j^{(3)}) \tag{2-3}$$

式中，$f(\,)$ 表示激活函数；$w_{ij}^{(l)}$ 表示从 l-1 层第 i 个神经元至 l 层第 j 个神经元的连接权值；b_i 表示可训练的偏置。

3 激活函数

在第 2 节，提到了两次激活函数，到底什么是激活函数呢？说通俗一点，就是为了增强神经网络的表示能力，常常需要引入非线性，这个工作就交给了激活函数来完成。

3.1 常见的激活函数

Sigmoid 函数、Tanh 函数和 ReLU 函数是人工神经网络常用的激活函数，它们的表示式和函数曲线如下所示：

$$\text{Sigmoid}(x) = \frac{1}{1+\mathrm{e}^{-x}} \tag{3-1}$$

$$\text{Tanh}(x) = \frac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}+\mathrm{e}^{-x}} \tag{3-2}$$

$$\mathrm{ReLU}(x)=\begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{3-3}$$

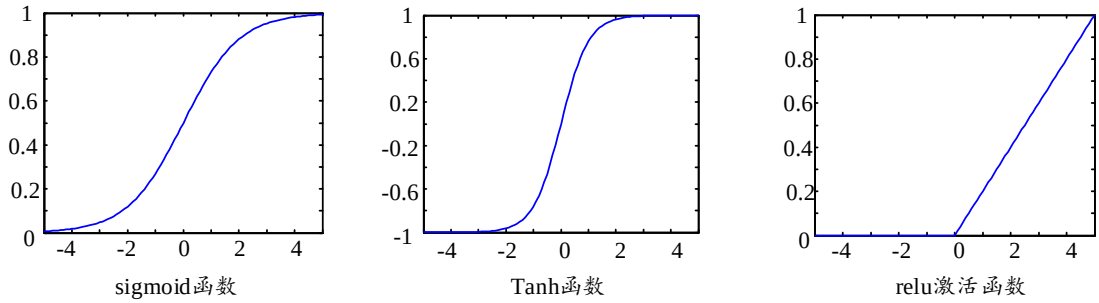

图 3-1 三种激活函数的曲线图

3.2 激活函数的过饱和问题

对于 sigmoid 激活函数，计算它的导数：

$$f(x)=\frac{1}{1+\mathrm{e}^{-x}} \tag{3-4}$$

$$f'(x)=\frac{\mathrm{e}^{-x}}{(1+\mathrm{e}^{-x})^2}=f(x)\cdot[1-f(x)] \tag{3-5}$$

sigmoid 激活函数的导数的曲线如图 3-2 所示。

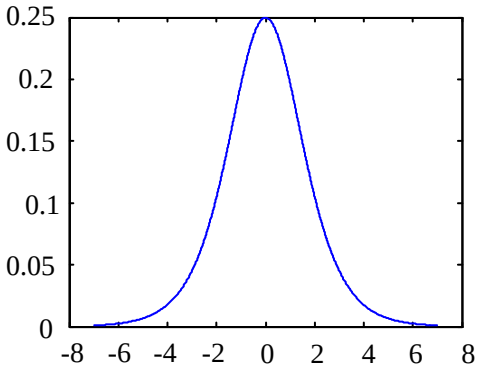

图 3-2 sigmoid 函数的导数

从 sigmoid 激活函数的导数曲线可以看出，当激活函数的输入值的绝对值过大时，它的导数接近于 0，这就是 sigmoid 激活函数的过饱和现象。它会导致梯度在反向传播过程（后续讲的 BP 算法）中逐渐消失，从而使网络的权值与偏置不能被训练，导致神经网络不收敛。

对于 tanh 激活函数，计算它的导数：

$$f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{3-6}$$

$$f'(x)=\frac{\left(e^{x}+e^{-x}\right)^{2}-\left(e^{x}-e^{-x}\right)^{2}}{\left(e^{x}+e^{-x}\right)^{2}}=1-\left[f(x)\right]^{2} \tag{3-7}$$

tanh 激活活函数导数的曲线如图 3-3 所示，它的饱和问题与 sigmoid 激活函数的道理相同。

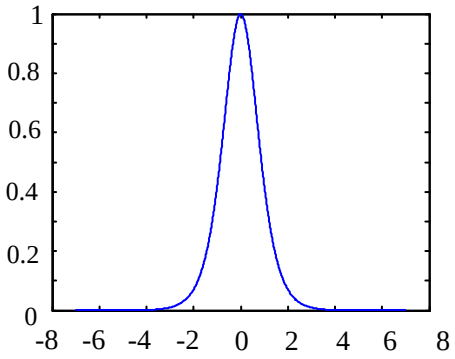

图 3-3 tanh 函数的导数

对于 Relu 激活函数，计算它的导数：

$$f(x)=\begin{cases}x & x\geq 0\\ 0 & x<0\end{cases} \tag{3-8}$$

$$f'(x)=\begin{cases}1 & x\geq 0\\ 0 & x<0\end{cases} \tag{3-9}$$

从公式 3-9 中，可以看出，当 x>0 时，Relu 激活函数的导娄为常数，因此 Relu 激活函数并不会出现过饱和现象，因此它也常常被用于深层神经网络中，来解决网络训练过程中的梯度消失现象。

3.3 小结

对于三种激活函数，由于 Sigmoid 函数非零均值的特性，它的实际表现中性能最差，大多数情况下 Tanh 激活函数要优于 Sigmoid 函数。Sigmoid 激活函数与 tanh 激活函数都会出现过饱和现象，使网络在训练过程中出现梯度消失或爆炸现象。ReLU 激活函数可以有效避免过饱和现象，在深层神经网络中具有良好的性能，原因在于 ReLU 函数的导数为常数，可以有效杜绝网络训练过程中的梯度消失现象。

4 损失函数

我们通过一个具体的例子来引出损失函数，这个例子也是后面我们将仿真的例子，就是 1-10 个数字的分类问题。假设有 60000 张手写字体，每一张为 28*28 的黑白图片，共有 10 类，即 0，1，2，……，9。每一个手写字体样本可以表示为1×784 的一维向量，样本标签可以表示为1×10 的一维向量，例如当样本标签为 7 时，可以表示为[0,0,0,0,0,0,0,1,0,0]。

现在想用如图 1-5 的多层神经网络实现对手写字体的分类任务，怎么办？

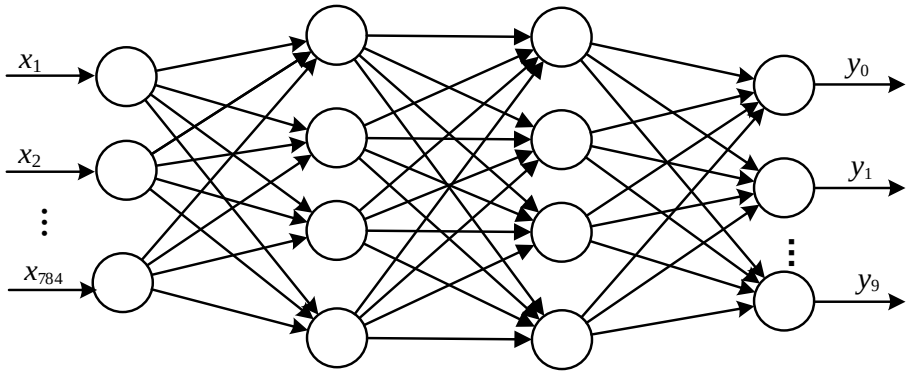

图 1-5 多层神经网络

Fig.1-5 multi-layer neural network

把图 1-5 的多层神经网络表示为以权值 w 和偏置 b 为参数的函数 $h_{w,b}(x)$，给定一个训练样本（$\boldsymbol{x}, \boldsymbol{y}$），当网络的输入 $\boldsymbol{x}$ 时，期望输出为 $\boldsymbol{y}$，而实际输出为 $h_{w,b}(\boldsymbol{x})$。因此，我们需要调整权值 w 和偏置 b（调整权值 w 和偏置 b 的过程就是网络训练的过程），使得网络的实际输出 $h_{w,b}(\boldsymbol{x})$ 无限接近期望输出 $\boldsymbol{y}$，在这里，使用**损失函数**来表示网络实际输出与期望输出的误差值。

注：加粗的变量表示向量。

4.1 平方差损失函数

平方差损失函数，使得网络的期望输出与实际输出的差的平方和最小，公式表示为：

$$J(\,)=\frac{1}{2}\left\|h_{w,b}(\boldsymbol{x})-\boldsymbol{y}\right\|^2 \qquad (4\text{-}1)$$

上式中，1/2 为系数项，加这个系数的原因在于求平和项的导数的时候会产生一个 2 的系数，正好相约为 1，其实不加也没事。‖ ‖表示向量 2 范数，只所以没有用（）代替‖ ‖

是因为 $h_{w,b}(\boldsymbol{x})-\boldsymbol{y}$ 表示是一个向量。$\|\ \|^2$ 表示了对向量内的每一项求平方后加和。

4.2 交叉熵损失函数

在二分类问题上，常常使用交叉熵损失函数。对于二分类问题，神经网络的输出用一个神经元 y 表示即可，训练样本的标签要么为 0，要么为 1。对于一个训练样本（$\boldsymbol{x}$, y），交叉熵损失函数表示为：

$$J = y\log\left[h_{w,b}(\boldsymbol{x})\right]+(1-y)\log\left[1-h_{w,b}(\boldsymbol{x})\right] \tag{4-2}$$

4.3 softmax 损失函数

在多分类问题时，常常会使用 softmax 损失函数，在很多文献中，也常常把 softmax 层单独看作一层网络结构。首先说明什么是 softmax 层，再讲 softmax 损失函数。

当给定一个输入 $\boldsymbol{y}$ 时，

$$\boldsymbol{y}=\left[y_1,y_2,y_3,\cdots,y_n\right] \tag{4-3}$$

经过 softmax 层之后，输出为 $\overline{\boldsymbol{y}}$ 且它的各项和为 1：

$$\overline{\boldsymbol{y}}=\left[\frac{\mathrm{e}^{y_1}}{\sum_i \mathrm{e}^{y_i}},\frac{\mathrm{e}^{y_2}}{\sum_i \mathrm{e}^{y_i}},\frac{\mathrm{e}^{y_3}}{\sum_i \mathrm{e}^{y_i}},\cdots,\frac{\mathrm{e}^{y_n}}{\sum_i \mathrm{e}^{y_i}}\right] \tag{4-4}$$

因此，softmax 层本质上是把输入 $\boldsymbol{y}$ 转化为概率输出 $\overline{\boldsymbol{y}}$。

在此基础上，引入 softmax 损失函数，对于输入训练样本 $\boldsymbol{x}$，当它的标签为 T 时，sfotmax 损失函数表示为：

$$J=-\sum_{j=1}^{k}\mathrm{I}\{\mathrm{T}=j\}\log\frac{\mathrm{e}^{y_j}}{\sum_{\mathrm{i}=1}^{k}\mathrm{e}^{y_i}} \tag{4-5}$$

式中，k 为种类数目，y_i 表示输出层第 i 个节点的输出值，I{ T = j}表示若 j 与 T 相等则为 1，否则为 0。

4.4 小结

常用的损失函数就这么几种，还有一种信息增益损失函数，想了解的话，可以百度

或 google 一下。

5 反向传播算法

由 1.4 节我们知道，神经网络的训练过程就是使损失函数达到最小值（趋于 0）的过程。因此，可以利用梯度下降法调整权值 w 与偏置 b，使损失函数的值达到最小。关键问题是如何求得权值 w 与偏置 b 的梯度呢？这就是反向传播(Backpropagation, BP)算法要做的事情。

美国学者 Rumelhart、Hinton 和 Williams 等人于 20 世纪 80 年代提出了反向传播（Backpropagation, BP）算法，用于解决多层前馈神经网络的训练问题。下面基于多层的前馈神经网络，推导 BP 算法更新网络权值的过程，推导过程中的相关符号表示如下：

- 损失函数记作：$J(\ )$
- 激活函数记作：$f(\ )$
- 第 l 层的神经元的数目记作：$N^{(l)}$。
- 第 l 层第 j 个节点的输入值记作：$I_j^{(l)}$，它的输出记作：$O_j^{(l)}$
- 从第 l-1 层第 i 个神经元到第 l 层第 j 个神经元的连接权值记作 $w_{ij}^{(l)}$，偏置记作 $b_j^{(l)}$。

网络的正向传播

设单个输入样本为 (x, y)，初始化网络中权值 W 与偏置 b，并假设神经网络共有 L 层。根据图 2-2 可知：

第 1 层：第 j 个神经元输入与输出分别表示为：

$$\mathrm{I}_j^{(1)} = \sum_{i=1}^{N^{(0)}} w_{ij}^{(1)} \bullet x_i + b_j^{(1)} \tag{5-1}$$

$$O_j^{(1)} = f(\mathrm{I}_j^{(1)}) \tag{5-2}$$

第 2 层：第 j 个神经元的输入与输出可以表示为：

$$\mathrm{I}_j^{(2)} = \sum_{i=1}^{N^{(1)}} w_{ij}^{(2)} \bullet O_i^{(1)} + b_j^{(2)} \tag{5-3}$$

$$O_j^{(2)} = f(\mathrm{I}_j^{(2)}) \tag{5-4}$$

数据逐层传递，依次类推……，

第 L 层：第 j 个神经元的输入与输出为：

$$\mathrm{I}_j^{(L)} = \sum_{i=1}^{N^{(L-1)}} w_{ij}^{(L)} \bullet O_i^{(L-1)} + b_j^{(L)} \tag{5-5}$$

$$O_j^{(L)} = \mathrm{I}_j^{(L)} \tag{5-6}$$

注：在公式（5-6）中，当使用 softmax 损失函数时，最后一层不需要激活函数（加了激活函数也行，不过是多余的），所以输入与输出相等。若使用平方差损失函数，需要加一个激活函数哦。

Loss 层：求得 loss 层中的损失函数：

$$J(\,) = J\left(O_1^{(L)}, O_2^{(L)}, \cdots\cdots, O_{N^{(L)}}^{(L)}, y\right) \tag{5-7}$$

式中，$O_1^{(L)}, O_2^{(L)}, \cdots\cdots$ 表示第 L 层的输出值，y 表示样本的标签。

误差的反向传播

Loss 层：根据我们选择的具体损失函数，可以求出 J 对第 L 层中第 j 个神经元的输出值 $O_j^{(L)}$ 的梯度：$\dfrac{\partial J}{\partial O_j^{(L)}}$。

第 L 层：我们已知 $w_{ij}^{(L)}$、$b_j^{(L)}$、$I_j^{(L)}$、$O_j^{(L)}$ 和 $\dfrac{\partial J}{\partial O_j^{(L)}}$，所以可以求得：

第 L 层的权值梯度：

$$\frac{\partial J}{\partial w_{ij}^{(L)}} = \frac{\partial J}{\partial O_j^{(L)}} \cdot \frac{\partial O_j^{(L)}}{\partial I_j^{(L)}} \cdot \frac{\partial I_j^{(L)}}{\partial w_{ij}^{(L)}} = \frac{\partial J}{\partial O_j^{(L)}} \cdot f'(I_j^{(L)}) \cdot O_i^{(L-1)} \tag{5-8}$$

第 L 层的偏置梯度：

$$\frac{\partial J}{\partial b_j^{(L)}} = \frac{\partial J}{\partial O_j^{(L)}} \cdot \frac{\partial O_j^{(L)}}{\partial I_j^{(L)}} \cdot \frac{\partial I_j^{(L)}}{\partial b_j^{(L)}} = \frac{\partial J}{\partial O_j^{(L)}} \cdot f'(I_j^{(L)}) \cdot 1 = \frac{\partial J}{\partial O_j^{(L)}} \cdot f'(I_j^{(L)}) \tag{5-9}$$

第 L-1 层的第 i 个神经元的输出值 $O_i^{(L\text{-}1)}$ 的梯度：

$$\frac{\partial J}{\partial O_i^{(L-1)}} = \sum_j^{N_L} \left[\frac{\partial J}{\partial O_j^{(L)}} \bullet \frac{\partial O_j^{(L)}}{\partial I_j^{(L)}} \bullet \frac{\partial I_j^{(L)}}{\partial O_i^{(L-1)}} \right] = \sum_j^{N_L} \left[\frac{\partial J}{\partial O_j^{(L)}} \bullet f'(I_j^{(L)}) \bullet w_{ij}^{(L)} \right] \tag{5-10}$$

第 L-1 层：我们已知 $w_{ij}^{(L)}$、$b_j^{(L)}$、$I_j^{(L)}$、$O_j^{(L)}$ 和 $\dfrac{\partial J}{\partial O_j^{(L)}}$，所以可以求得：

第 L-1 层的权值梯度为：

$$\frac{\partial J}{\partial w_{ij}^{(L-1)}}=\frac{\partial J}{\partial O_j^{(L-1)}}\cdot\frac{\partial O_j^{(L-1)}}{\partial I_j^{(L-1)}}\cdot\frac{\partial I_j^{(L-1)}}{\partial w_{ij}^{(L-1)}}=\frac{\partial J}{\partial O_j^{(L-1)}}\cdot f'(I_j^{(L-1)})\cdot O_i^{(L-2)} \tag{5-11}$$

第 L-1 层的偏置梯度：

$$\frac{\partial J}{\partial b_j^{(L-1)}}=\frac{\partial J}{\partial O_j^{(L-1)}}\cdot\frac{\partial O_j^{(L-1)}}{\partial I_j^{(L-1)}}\cdot\frac{\partial I_j^{(L-1)}}{\partial b_j^{(L-1)}}=\frac{\partial J}{\partial O_j^{(L-1)}}\cdot f'(I_j^{(L-1)})\cdot 1=\frac{\partial J}{\partial O_j^{(L-1)}}\cdot f'(I_j^{(L-1)}) \tag{5-12}$$

第 L-2 层的第 i 个神经元的输出值 $O_i^{(L-1)}$ 的梯度：

$$\frac{\partial J}{\partial O_i^{(L-2)}}=\sum_j^{N_L}\left[\frac{\partial J}{\partial O_j^{(L-1)}}\bullet\frac{\partial O_j^{(L-1)}}{\partial I_j^{(L-1)}}\bullet\frac{\partial I_j^{(L-1)}}{\partial O_i^{(L-2)}}\right]=\sum_j^{N_L}\left[\frac{\partial J}{\partial O_j^{(L-1)}}\bullet f'(I_j^{(L-1)})\bullet w_{ij}^{(L-1)}\right] \tag{5-13}$$

数据逐层传递，依次类推……，最后求得第 1 层权值与偏置的梯度。最后，我们利用 BP 算法求出了每一层神经网络中的权值与偏置的梯度。

权值与偏置的更新

神经网络的训练过程，就是对公式（2-4）求最优解的过程。根据梯度下降法的原理，在网络的每一次迭代过程中，按照如下公式对参数 $\boldsymbol{W}$ 与 $\boldsymbol{b}$ 进行更新。

$$w_{ij}^{(l)}=w_{ij}^{(l)}-\frac{\partial}{\partial w_{ij}^{(l)}}J(W,b) \tag{5-14}$$

$$b_{ij}^{(l)}=b_{ij}^{(l)}-\frac{\partial}{\partial b_{ij}^{(l)}}J(W,b) \tag{5-15}$$

$$\frac{\partial}{\partial w_{ij}^{(l)}}J(W,b)=\left[\frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_{ij}^{(l)}}J(W,b,x^{(i)},y^{(i)})\right]+\lambda w_{ij}^{(l)} \tag{5-16}$$

$$\frac{\partial}{\partial b_{ij}^{(l)}}J(W,b)=\frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b_{ij}^{(l)}}J(W,b,x^{(i)},y^{(i)}) \tag{5-17}$$

注：公式（5-16）中，权值的更新加入了权值衰减系数，后续为讲。

到此为止，权值更新完毕，BP 算法推导完成。

6 正则化方法

过拟合现象一直是人工神经网络训练过程中需要解决的重要问题，尤其是当网络规

模很大，包含大量的权值与偏置参数时，过拟合化问题更容易出现。过拟合化是指在网络训练过程中，随着迭代次数的增加，网络的训练误差继续下降，而测试样本的识别率不再增加甚至出现下降的现象。网络的过拟合问题会导致网络的泛化能力下降，使网络对未知的测试样本的识别能力变差。

6.1 dropout 机制

抑制过拟合问题的常用手段包括 dropout 机制和权值的正则化机制。dropout 机制是指在网络的每一次训练过程中，随机地屏蔽掉每一层的全连接层中一定比例神经元，从而促使整个网络的神经元都会被均匀地训练，提高网络的泛化能力。对于 dropout 机制，也可以换一个角度思考：由于在每一次的训练过程中都是随机地屏蔽掉神经元，相当于每一次的训练都是面对不同的神经网络，因此 dropout 机制是平均了许多不同神经网络的识别性能，从而起到抑制过拟合的作用。

图 6-1 给出了 dropout 机制的示意图。

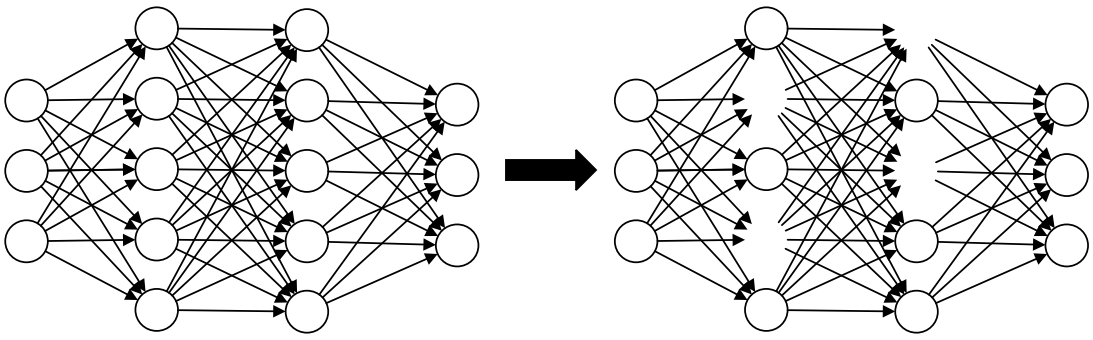

图 6-1 dropout 机制

6.2 权值正则化

而权值的正则化机制就是限制权值的大小，常见的正则化机制有 L1 正则化和 L2 正则化，公式分别如下所示：

$$J = J_0 + \frac{\lambda}{2} \cdot \sum_w |w| \tag{3-7}$$

$$J = J_0 + \frac{\lambda}{2} \cdot \sum_w w^2 \tag{3-8}$$

式中，J_0 为原始的损失函数，J 表示加入正则化机制后的损失函数，w 表示网络的权值，

λ表示权重衰减系数。

6 权值初始化方法

常见的网络权值的初始化方法包括随机初始化法、服从正态分布的初始化方法以及文献[1]提出的 Xavier 初始化方法等，Xavier 方法效果最好，它的表达式为：

$$W \sim U\left[-\frac{\sqrt{6}}{n_j+n_{j+1}}, \frac{\sqrt{6}}{n_j+n_{j+1}}\right] \tag{6-1}$$

式中，W 为权值；U 表示服从均匀分布；n_j 、n_{j+1} 为待初始化神经元的扇入、扇出系数。

如果想了解更多 Xavier 初始化的方法，可以参考文献[1]或查看本人之前写过的一篇博文 http://www.cnblogs.com/yinheyi/p/6411713.html，对文献[1]进行了大致的讲解。

第二部分：python 代码的实现神经网络框架

第三部分：使用多层神经网络识别手写字体