

## 2. 神经网络的工作过程

### 2.1 前向传播

在前面的学习中，我们介绍了神经网络的基本结构，还有神经元的计算方式，本节开始我们继续深入讲解神经网络的工作过程。在前面我们接触过了一种简单的神经网络结构，叫做全连接神经网络，同时，这种神经元从输入层开始，接受前一级输入，并输出到后一级，直至最终输出层，由于数据是一层一层从输入至输出传播的，也叫作前馈神经网络。

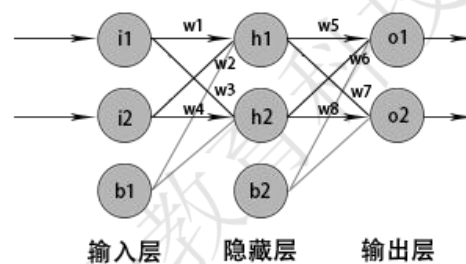


图 6 某神经网络例子

接下来我们看图 6 所示的神经网络，这是一个简单的神经网络结构， $i_1$  与  $i_2$  分别是两个输入，隐藏层有两个神经元节点  $h_1$  与  $h_2$ ，偏置项  $b_1$  与  $b_2$ ，输出层也有 2 个神经元节点  $o_1$  与  $o_2$ ，那么前向传播过程中，隐藏层神经元节点线性部分计算有  $l_{h1} = w_1 \times i_1 + w_2 \times i_2 + b_1$ ，假设激活函数为 sigmoid，那么第一个隐藏层节点输出为  $out_{h1} = \frac{1}{1+e^{-l_{h1}}}$ ，同理  $out_{h2} = \frac{1}{1+e^{-l_{h2}}}$ ，输出层  $o_1$  节点假设激活函数为 sigmoid（这里使用激活函数的作用往往是根据特定的需求，比如多分类问题需要用到 softmax 函数），那么  $l_{o1} = w_5 \times out_{h1} + w_6 \times out_{h2} + b_2$ ，同理  $l_{o2} = w_7 \times out_{h1} + w_8 \times out_{h2} + b_2$ ，这样整个网络的前向传播输出  $out_{o1} = \frac{1}{1+e^{-l_{o1}}}$ 、 $out_{o2} = \frac{1}{1+e^{-l_{o2}}}$ 。

这时如果将我们的参数都赋值常数带入进去，然后设定一组真实的数据（其中包含输入  $i_1$  与  $i_2$ ，也包含一个与之对应的输出  $out$ ），输入之后会计算得到一个计算结果，

然后将计算结果与我们设定的真实结果  $out$  做比较, 会出现差距, 那么损失函数我们上一章已经提到了, 假设损失函数是

$$E_{total} = \frac{1}{n} \sum_{i=1}^n (target - output)^2$$

其中  $target$  代表数据样本的标签, 是理应输出的值,  $output$  代表神经网络计算的输出, 那么这两者之间的差值就能代表当前神经网络计算的误差, 这里取其平方再加和除  $n$  的意义是, 实际输出与我们期望的输出差距越大, 代价越高, 然后取其均方差, 则有  $E_{o1} = \frac{1}{2}(target - out_{o1})^2$ ,  $E_{o2} = \frac{1}{2}(target - out_{o2})^2$ ,  $E_{total} = E_{o1} + E_{o2}$ , 有了这个损失函数我们就可以学习接下来的反向传播了!

## 2.2 反向传播

现在我们来讲一下反向传播, 顾名思义, 反向传播算法是利用损失函数进而从输出到输入方向传播达到调整参数的目的, 它的存在主要是解决深层(多个隐藏层)神经网络的参数更新问题, 反向传播算法就是梯度下降应用了链式法则, 梯度下降这一概念我们下一小节会讲到, 链式法则是微积分中的求导法则, 用于复合函数的求导, 在神经网络中只要有了隐藏层, 那么隐藏层的权重参数与损失函数会构成复合函数, 因此使用链式法则解决复合函数求导问题达到调整权重参数的目的。

具体链式法则的计算方式大家可以查阅高数资料了解。可能在这里出现了一些难理解的专有名词, 那么举个例子来帮助我们更直观的理解, 例如输入是小红, 隐藏层是小蓝, 输出层是小黄, 根据神经网络的计算方式, 最终会产生损失函数, 那么利用梯度下降算法, 可以将损失函数的反馈直接反馈到输出层小黄, 进而调整输出层小黄前面的参数矩阵, 但是隐藏层小蓝不能得直接反馈而不能优化前面的参数矩阵, 这时候利用链式法则, 将小蓝与损失函数形成复合函数从而让误差反馈也能调整隐藏层小蓝前面的权重矩阵。最后迭代几次后最终误差会降低到最小。梯度下降具体是做什么的我们先不用管, 知道它是利用损失函数进而调整权重参数最终使损失函数尽量小就可以了。

接下来我们将上一小节中的神经网络拿过来, 并将输入、输出、权值、偏置都赋予实数, 来演示一下反向传播的过程, 具体实数赋值如下:

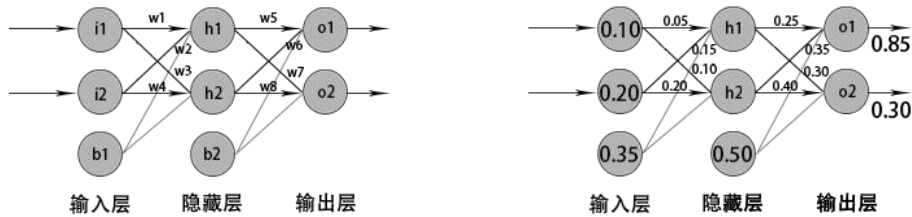


图 7 赋值后的神经网络

图 7 中，左侧是原来我们搭建的神经网络，右侧是将输入赋值 0.10 与 0.20，隐藏层权重为 0.05、0.15、0.10、0.20，偏置为 0.35，输出层权重为 0.25、0.35、0.30、0.40，偏置为 0.50，最终赋值输出为 0.85、0.30，注意这个输出是我们预设好的标签，就是我们希望输入 0.10 与 0.20 后，通过神经网络计算，最终输出 0.85 与 0.30。通过上一节的前向传播，我们得到了前向传播中各个节点的计算公式，将实数带入公式我们可以得到：

$$\text{隐藏层第一个节点线性部分: } l_{h1} = 0.05 \times 0.10 + 0.15 \times 0.20 + 0.35 = 0.385$$

$$\text{隐藏层第二个节点线性部分: } l_{h2} = 0.10 \times 0.10 + 0.20 \times 0.20 + 0.35 = 0.400$$

$$\text{隐藏层第一个节点输出: } \text{out}_{h1} = \frac{1}{1 + e^{-0.385}} = 0.595078474$$

$$\text{隐藏层第二个节点输出: } \text{out}_{h2} = \frac{1}{1 + e^{-0.4}} = 0.598687660$$

同理：输出层线性部分：

$$l_{o1} = 0.25 \times 0.595078474 + 0.35 \times 0.598687660 + 0.50 = 0.8583102995$$

$$l_{o2} = 0.30 \times 0.595078474 + 0.40 \times 0.598687660 + 0.50 = 0.9179986062$$

输出层输出：

$$\text{out}_{o1} = \frac{1}{1 + e^{-0.8583102995}} = 0.702307507$$

$$\text{out}_{o2} = \frac{1}{1 + e^{-0.9179986062}} = 0.714634132$$

损失函数计算结果：

$$E_{o1} = \sum \frac{1}{2} (0.85 - 0.702307507)^2 = 0.010906536$$

$$E_{o2} = \sum \frac{1}{2} (0.30 - 0.714634132)^2 = 0.085960731$$

$$E_{total} = 0.010906536 + 0.085960731 = 0.096867268$$



$$\frac{\partial out_{o1}}{\partial l_{o1}} = out_{o1} \times (1 - out_{o1}) = 0.702307507 \times (1 - 0.702307507) = 0.209071672$$

接着往下计算  $\frac{\partial l_{o1}}{\partial w5}$ :

$$l_{o1} = w5 \times out_{h1} + w6 \times out_{h2} + b2$$

$$\frac{\partial l_{o1}}{\partial w5} = 1 \times out_{h1} \times w5^{1-1} + 0 + 0 = 0.595078474$$

最后三项相乘得出  $\frac{\partial E_{total}}{\partial w5}$ :

$$\frac{\partial E_{total}}{\partial w5} = -0.147692493 \times 0.209071672 \times 0.595078474 = -0.018375021$$

到这之后我们可以根据梯度下降的算法来更新  $w5$  权重了:

$$w_5^+ = w_5 - \eta \frac{\partial E_{total}}{\partial w5} = 0.25 - 1 \times (-0.018375021) = 0.268375021$$

上述的式子是梯度下降的算法,  $\eta$  是学习率, 就是说梯度下降的步幅, 是由工程师凭借经验或者测试而确定的数值, 在这我们先认识一下这些名词, 那么  $w_5$  就被更新为  $w_5^+$ 。同理我们也可以将  $w6, w7, w8$  更新完成。当我们使用更新完之后的参数再带入神经网络去计算时, 会发现最终的输出与真实的输出已经更接近了。这就是梯度下降算法的作用, 是不是迫不及待的想了解梯度下降了? 让我们先把隐藏层的权重是如何通过反向传播算法更新的理解清楚。

对于隐藏层更新权重时, 例如更新  $w1$ , 使用的方法基本与前文更新  $w5$  是一致的, 从  $out_{o1}, l_{o1}, w5$  三处分别求偏导再求其乘积, 但有区别的是, 更新  $w1$  时候, 损失函数传递方式变为下图 9 所示:

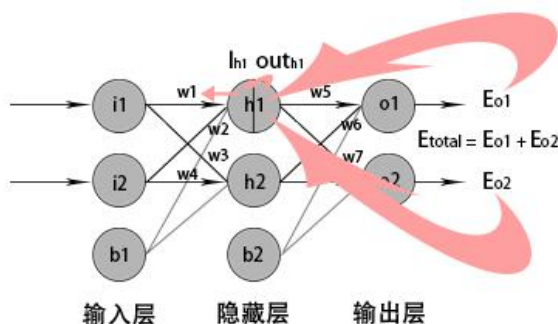


图 9 损失函数传递方式变化

那么就有损失函数对  $w1$  权重求偏导:

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial l_{h1}} \times \frac{\partial l_{h1}}{\partial w1}$$

由上图中的传递方式可以看到,  $out_{h1}$  会接受有  $E_{o1}$ , 与  $E_{o2}$  两个地方传来的误差损失。

那么  $\frac{\partial E_{total}}{\partial out_{h1}}$  则可以拆为:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

继续往下计算  $\frac{\partial E_{o1}}{\partial out_{h1}}$ , 主要是利用链式法则将隐藏层与输出层链接起来计算:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial l_{o1}} \times \frac{\partial l_{o1}}{\partial out_{h1}}$$

那么可以利用输出层的计算结果 ( $\frac{\partial E_{o1}}{\partial out_{o1}}$  与  $\frac{\partial E_{total}}{\partial out_{o1}}$  结果相等):

$$\frac{\partial E_{o1}}{\partial l_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial l_{o1}} = -0.147692493 \times 0.209071672 = -0.030878316$$

接着:

$$l_{o1} = w5 \times out_{h1} + w6 \times out_{h2} + b2$$

$$\frac{\partial l_{o1}}{\partial out_{h1}} = w5 = 0.25$$

最终:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial l_{o1}} \times \frac{\partial l_{o1}}{\partial out_{h1}} = -0.030878316 \times 0.25 = -0.007719579$$

同样我们可以计算出:

$$\frac{\partial E_{o2}}{\partial out_{h1}} = 0.025367174$$

那么:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = -0.007719579 + 0.025367174 = 0.017647595$$

在这之后我们计算第二部分  $\frac{\partial out_{h1}}{\partial l_{h1}}$ :

$$out_{h1} = \frac{1}{1 + e^{-l_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial l_{h1}} = out_{h1}(1 - out_{h1}) = 0.595078474 \times (1 - 0.595078474) = 0.240960084$$

然后计算第三部分  $\frac{\partial l_{h1}}{\partial w1}$ :

$$l_{h1} = w1 \times i1 + w2 \times i2 + b1$$

$$\frac{\partial l_{h1}}{\partial w1} = i1 = 0.1$$

那么三项相乘:

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial l_{h1}} \times \frac{\partial l_{h1}}{\partial w1}$$

$$\frac{\partial E_{total}}{\partial w1} = 0.017647595 \times 0.240960084 \times 0.1 = 0.000425237$$

最后梯度下降算法更新  $w1$ :

$$w_1^+ = w_1 - \eta \frac{\partial E_{total}}{\partial w1} = 0.05 - 1 \times 0.000425237 = 0.049574763$$

同理也可以将  $w2$ 、 $w3$ 、 $w4$  更新完成，最后将更新完的所有参数全部带入重新计算，得出新的输出，然后再根据损失函数、反向传播、梯度下降不停的去迭代更新参数，最终理论上输出的结果会无限接近真实结果，那么这一组参数就会变得非常有价值，上文中的理论推导都是基于简单的实数，并且网络结构简单，目的是让大家理解神经网络的工作过程，真实的环境中都是使用矩阵批量计算的，并且已经有很多深度学习工具帮我们实现了计算过程，我们学会调用对应的函数就能便捷的实现计算过程。

## 2.3 训练神经网络

### 2.3.1 梯度下降算法

在前文中，神经网络的工作过程相信大家已经读之后已经有了一个认识，里面我们有几处提到了梯度下降，这个算法是优化权重参数的关键，并且目前主流的神经网络训练模型基本都使用梯度下降或改进后的梯度下降作为优化算法。这一小节我们就揭开梯度下降算法的神秘面纱。

说到梯度下降，不得不说的就是下山问题，梯度下降的思想其实就是一个下山的过程，假设我们爬到了山顶，然后感觉很累想尽快下山休息。但是这个时候起雾了，山上都是浓雾，能见度很低。那么我们下山的路径是无法确定的，这时候我们要怎么下山呢？聪明的你肯定能想到利用周围能看见的环境去找下山的路径，这个时候梯度下降的思想就能体现了，我们首先要找到周围最陡峭的地方，然后朝着地势往下的方向走，走完一段距离发现最陡峭的方向变了，那么我们调整一下方向接着沿地势往下的方位走。这样不断的进行下去，理论上最后一定会到达山脚，如图 10 所示。



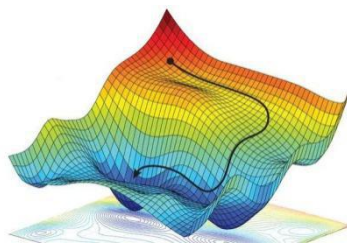


图 10 梯度下降图示

假设这座山有的坡特别平缓，那么几个方向的陡峭程度很接近，我们无法用肉眼测量出最陡峭的，假如恰好我们带了一个测量仪，然后每走一段距离我们就会测量一次，这时候就会出现一个问题，如果我们每走几步就测量一次，那么可能走到明天我们都下不去山，但是我们如果减少测量次数，那么可能就会偏离下山的最快的路径。这里我们就会需要一个合适的测量频率，保证我们下山路径足够快，然后又不需要太多时间去测量。

这个例子对应到神经网络训练过程中的梯度下降算法是怎么样的呢？学完前面的内容都知道，神经网络前向传播会计算出一个输出，输出与真实的数据标签输出对比会产生一个误差，然后根据误差会设定一个损失函数，那么损失函数与之前的权重能形成一个复杂的函数，这个函数就是在刚才举例中的那一座山，那么山脚对应损失函数的最小值。最快的下山方式就是在当下位置找最陡峭的地方然后沿着这个方向往下走，对应到上述复杂函数中，就是根据当前的权重参数找到梯度的方向，然后朝梯度的反向走，这样函数值就能下降最快，因为梯度方向是函数变化最快的方向。

因此，我们不断的使用这种方法，反复的求取梯度，函数就能达到相对的最小值。为什么是相对而不是绝对呢？我们下山过程中可能会遇到这种情况，根据地势往下走，不小心进入了一个很深的洼地，这时四周都是地势向上，人是很聪明的，我们可以爬出来继续找个其它地势向下的方向，但是计算机可没有那么聪明，它可能会深陷其中无论怎么求梯度都爬不出来了。这就是函数可能只能够达到局部的最小值，而不是全局的最小值。

下面再来讲讲梯度下降的梯度是什么。函数在某一点的梯度是这样一个向量，它的方向与取得最大方向导数的方向一致，而它的模为方向导数的最大值。在多个变量函数中，梯度是一个向量，向量有方向，梯度的方向就指出了函数在给定点的上升最快的方向。这就意味着我们需要到达山底，那么在我们下山测量中，梯度就告诉我们下山的方向。梯度的方向是函数在给定点上升最快的方向，那么反方向就是函数在给定点下降最



快的方向，所以我们只要沿着梯度的反方向一直走，就能走到局部的最低点！对于梯度的理解可能不太容易，我们需要搜索一些梯度与导数的资料，然后再去看一部分拟牛顿法的资料来进行深刻的体会理解。

从前文中我们可以知道，因为神经网络里有众多的权重参数，所以我们常常需要处理的是多元复合函数，要想知道某一个权重对损失函数的影响，那么求其偏导数即可，因此对于这个权重，我们可以确定向量的方向，求其值即可，这里涉及到了微分导数及偏导数的高数知识，如果忘记可以去查资料理解一下。要想理解梯度下降的数学原理必须要明白导数微分的概念。下面我们给出梯度下降的描述方程：

$$x_{n+1} = x_n - \eta \frac{\partial f(x,y)}{\partial x}$$

在这解释一下上式，例如原函数为  $z=f(x,y)=x^2+y^2$ ，那么上述公式是一次迭代更新  $x$  求解的过程。 $x_{n+1}$  是更新后的  $x$ ， $x_n$  是当前的  $x$ ， $\frac{\partial f(x,y)}{\partial x}$  代表在  $f(x,y)$  函数中对  $x$  求偏导， $\eta$  代表学习率。学习率看起来很陌生，不要着急，下一节我们就详细的介绍它了，还记得下山问题吗？我们说要找一个合适的测量频率，保证我们下山又快，测量次数又少。那么这个学习率就是影响我们测量频率的因素。可以将其理解为梯度下降过程中的步长。

### 2.3.2 学习率

在我们训练神经网络模型的时候，学习率是里面很重要的一个超参数，它决定着我们的损失函数能否收敛到最小值，还有需要多长时间才能收敛到最小值。一个合适的学习率能够让我们的损失函数在合适的时间内收敛到局部最小值。

在梯度下降算法里，例如一次权重更新中， $w_{n+1} = w_n - \eta \frac{\partial f(w)}{\partial w}$  为我们梯度下降的方程，假设它是一元函数，如果学习率设置的过小时，函数收敛过程如图 11 所示：

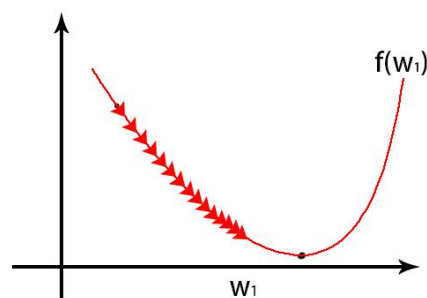


图 11 学习率过小时函数收敛过程

如果学习率过大的时候，函数收敛过程如图 12 所示：

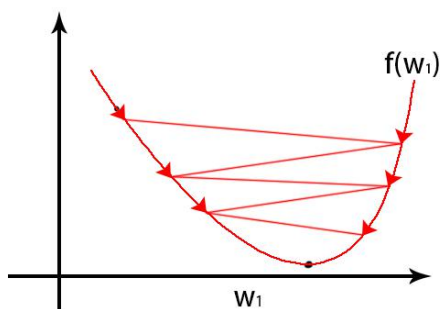


图 12 学习率过大时函数收敛过程

可以看出来，当学习率设定太小时，收敛过程将变得十分缓慢并且可能不收敛。反而学习率设置的过大时，梯度可能会在最小值附近来回震荡，甚至可能无法收敛。学习率有一些优化方法，它可以是固定的也可以是有衰减的，对于优化方法，我们后面再详细讲解。

### 2.3.3 损失函数

这一小节我们着重来介绍神经网络中的损失函数，以及与它关系很紧密的代价函数和目标函数。

损失函数 (loss function) 是用来评估模型的预测结果与真实结果不一致程度的函数。学完前几节之后，我们已经知道了神经网络的工作过程，在前向传播计算完之后会得出一个输出，这个输出就是模型的预测值，真实值是在准备数据时就定义好的，再来复习一遍，比如说 1000 张猫的图片，那么图片像素矩阵就是我们的输入，我们命名这些图为猫 1、猫 2...，这个就是手动打上去的标签，但是计算机是不能理解的，这样就需要将名字为猫的图片也转化为一个输出值矩阵，矩阵中以猫为名字的图片我们定义为 1，我们就知道将猫图片的像素矩阵输入之后经过计算得出的结果我们去和 1 比较，如果计算的预测值和真实值 1 不同，接着进行反向传播、梯度下降优化等等训练神经网络，这样我们最终会得到一个猫图片像素矩阵与数字 1 的映射关系。这个过程中，每次计算的值是预测值，真实值就是 1，是我们定义好的。

损失函数一个非负实值函数，通常会经过绝对值或者平方的方式让其变为非负，因为如果第一次计算差距是负数，第二次是正数，那么就会抵消了，变为没有误差，那肯

定是不合适的。损失函数越小，模型就越健壮。

在很多文章中代价函数与损失函数基本差不多，代价函数一般是说将所有样本的损失函数进行加和平均。在神经网络进行优化算法的时候会加入一些正则化项，是为了解决过拟合问题，让模型的泛化能力变强，体现在算法中就是减少更新权重的程度，让神经网络学习更小的权重。那么目标函数就是指损失函数与正则化项的和，人工智能虽然有一段历史，但是深度学习总的来说是一门比较新的学科，各种术语没有完全确定下来。我们读到一些名词的时候，一定要根据所在的场景里去理解。

### 2.3.4 训练数据

在训练神经网络的过程中，最重要的就是数据这一块，我们所做的所有算法都是为数据服务的，但是在深度学习中我们最容易忽视，也很容易出错的问题就是训练数据的规范了。我们训练的所有数据在输入到模型中的时候都要进行一些规范化。当我们使用一些预训练好的模型时，要知道这些模型本身就是通过规范后的数据训练的，注意在将自己的数据投入模型之前要首先对数据进行规范化处理。

下面我们来讲一下常用的训练数据预处理 的技巧。

#### ✓ 标准化：

数据标准化是指数据的各维度减均值除以标准差，这是最常用的标准化方法。

公式： $\frac{x_i - \mu}{\sigma}$  其中  $\mu$  指的是样本的均值， $\sigma$  指的是样本的标准差。

#### ✓ 归一化：

数据归一化是指数据减去对应维度的最小值除以维度最大值减去维度最小值，这样做可以将数值压缩到 $[0, 1]$ 的区间。

公式： $\frac{x_i - \min(x)}{\max(x) - \min(x)}$

#### ✓ 白化：

白化的目的是去除输入数据的冗余信息。

例如：训练数据是图像，由于图像中相邻像素之间具有很强的相关性，因此输入是冗余的。白化的目的就是降低输入的冗余性。

输入数据集，经过白化处理后，生成的新数据集满足两个条件：一是特征相关性较低；

二是特征具有相同的方差。

北京课工场教育科技有限公司