**深度神经网络训练**

09

深度神经网络由于其目标函数严重非凸，模型的训练变得困难。本章我们将讨论深度模型的一些训练技巧，包括：神经网络的梯度计算方法、改进网络中梯度消失的激活函数、梯度下降优化算法中自适应的参数更新方向和学习率、模型参数初始化方法、以及克服过拟合的方法。并通过案例，学习DNN、CNN和RNN等常见深度神经网络的应用。

第8章我们介绍了常见的神经网络结构，本章我们讨论如何训练一个深度神经网络。对一个复杂问题，我们可能需要训练更深的网络（如50层），每层神经元的数目可能数百个，有数十万个连接。这会使得模型的训练相当困难。

深度网络的训练一般采用梯度下降算法。9.1 节讨论深度网络梯度的计算方法：反向传播。9.2节介绍深度网络中常用的激活函数，以克服深度模型中的梯度消失问题。神经网络的目标函数是非凸，训练中可能回陷入局部极小，庞大网络的训练也非常缓慢。9.3节讨论随机梯度下降的各种改进版本，包括自适应学习率和动量技术来加速网络训练。9.4节介绍模型参数初始化方法。深度网络的另一个问题是网络的参数众多，模型有过拟合的风险，9.5节讨论一些克服过拟合的方案，包括正则、数据增广和Dropout等。9.6节给出了DNN、CNN和RNN在MNIST数据集上的应用。

## **标题2**  9.1 梯度计算：反向传播

一元函数在处的梯度为函数在点处的导数。多元函数在点处的梯度为函数对各元素的偏导数组成的向量，记为：

|  |  |
| --- | --- |
|  | （9-1） |

神经网络层数较深，从输入到输出为复合多元函数。复合函数的梯度计算需要用到微积分中的链式法则，反向传播是一种高效的链式法则实现方法。

### **9.1.1** 微积分中的链式法则

#### 1．实数对实数求梯度的链式法则

令是实数，和是从实数映射到实数的函数，且，则根据那么链式法则：

|  |  |
| --- | --- |
|  | （9-2） |

#### 2．向量对向量求导

令为维向量，为维向量，是从到的映射，则向量对向量的偏导数组成雅可比矩阵（Jacobian Matrix）：

|  |  |
| --- | --- |
|  | （9-3） |

例：，则

|  |
| --- |
|  |

#### 3．实数对向量求梯度的链式法则

令，为向量，为实数，是从到的映射，是从到的映射。如果且，那么

|  |  |
| --- | --- |
|  | （9-4） |

或写成向量形式：

|  |  |
| --- | --- |
|  | （9-5） |

即梯度为雅可比矩阵和梯度相乘得到。

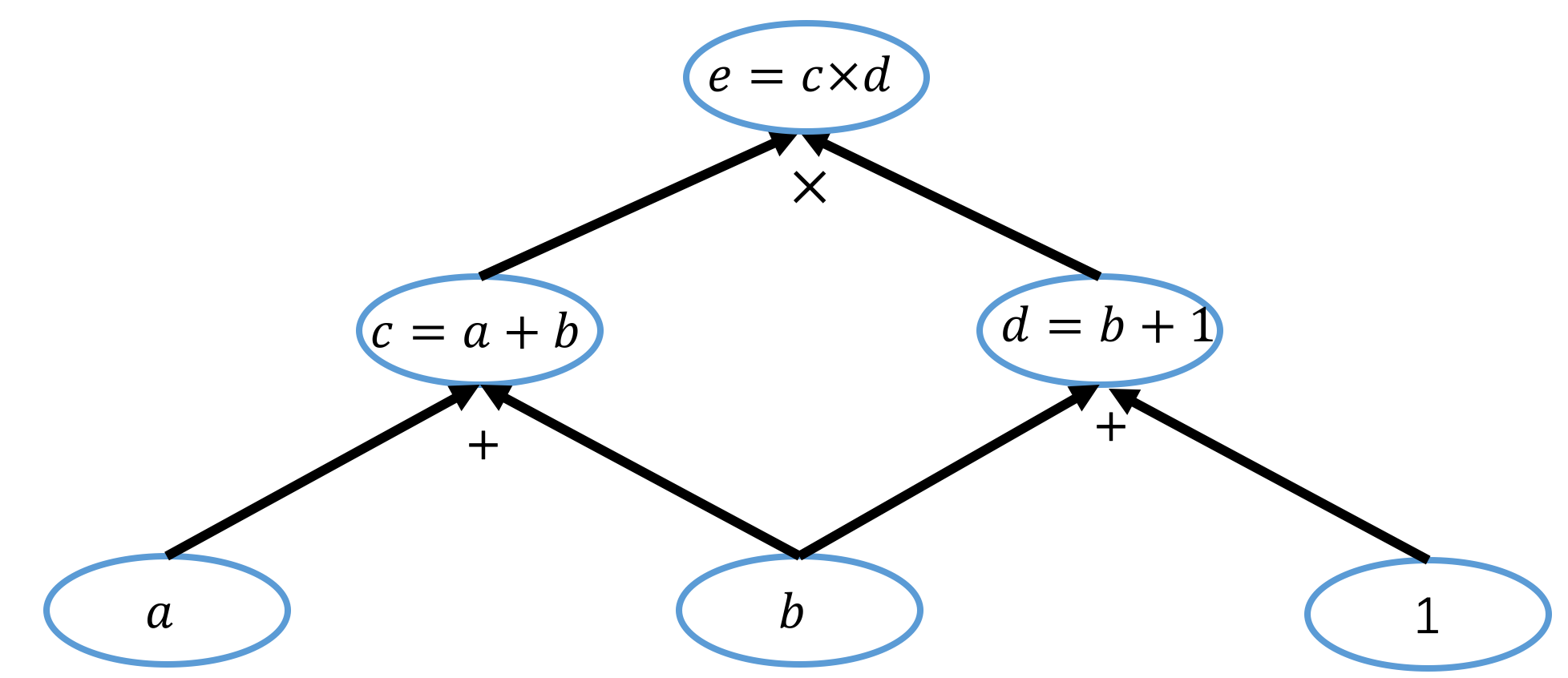
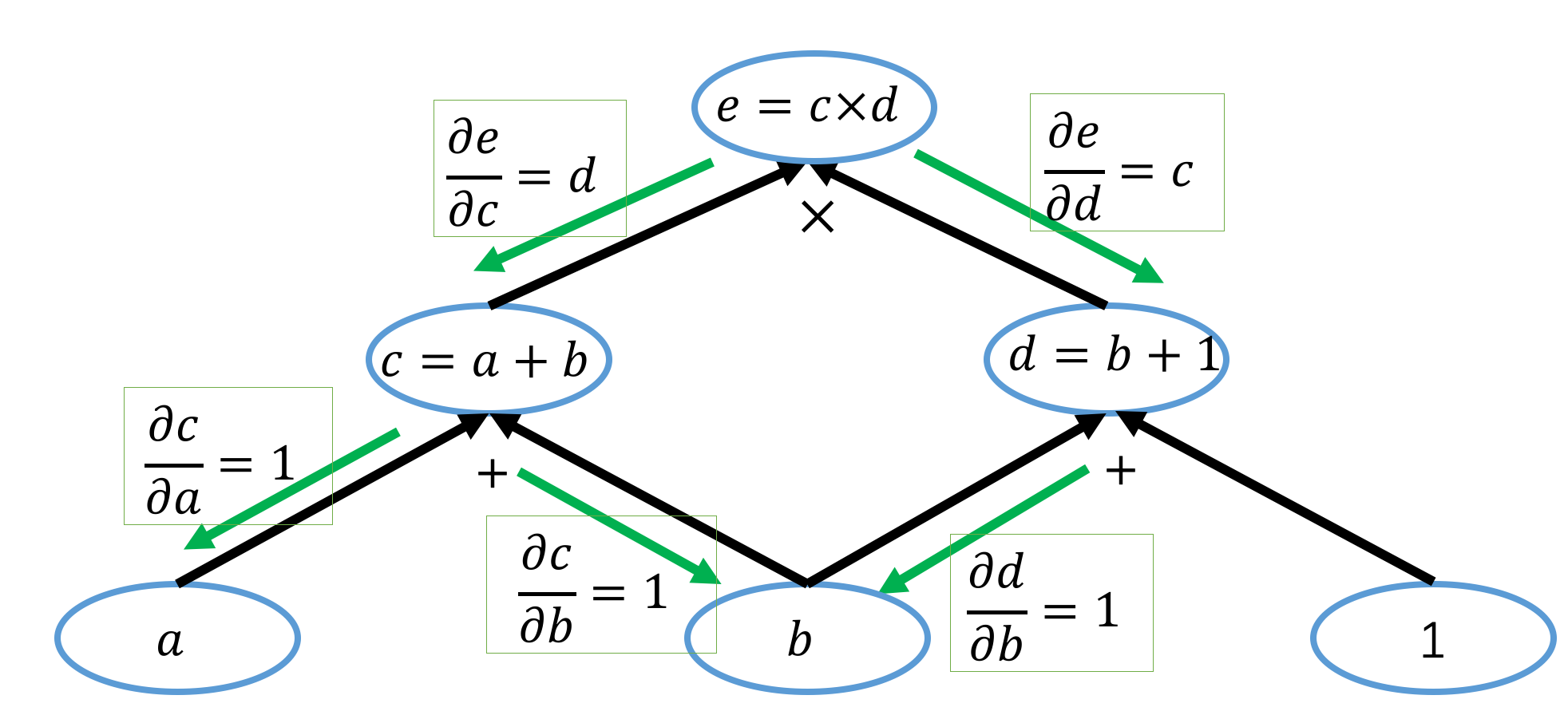
我们还可以将反向传播算法应用于任意维度的张量。从概念上讲，这与向量的反向传播完全相同，唯一的区别是如何将数字排列成网格以形成张量。我们可以想象成在我们运行反向传播之前，将每个张量变平为一个向量，计算一个向量值梯度，然后将该梯度重新构造成一个张量。

### **9.1.2** 计算图和反向传播

计算图是计算代数中的一个基础处理方法，通过有向图来表示给定数学表达式，并可以根据图的特点快速方便地对表达式中的变量进行求导。神经网络的本质就是一个多层复合函数，因此也可以通过一个图来表示其表达式。计算图中，结点表示变量，边表示简单运算。

**例：**表达式 

计算图如图9-2（a）所示。图中为叶子结点，无需再往下计算导数，为常数结点。为非叶子结点。

（a） （b）

图9-1 表达式的计算图

前向计算完成表达式的计算。

如给定，按箭头方向前向计算得到：

利用计算图，很方便用反向计算来计算导数（如图9-1（b）所示）。

首先我们计算每条边（有直接关系的变量）的导数：

然后根据微积分链式法则，按箭头的反方向计算非直接变量之间的导数：

### **9.1.3** **DNN**的反向传播算法

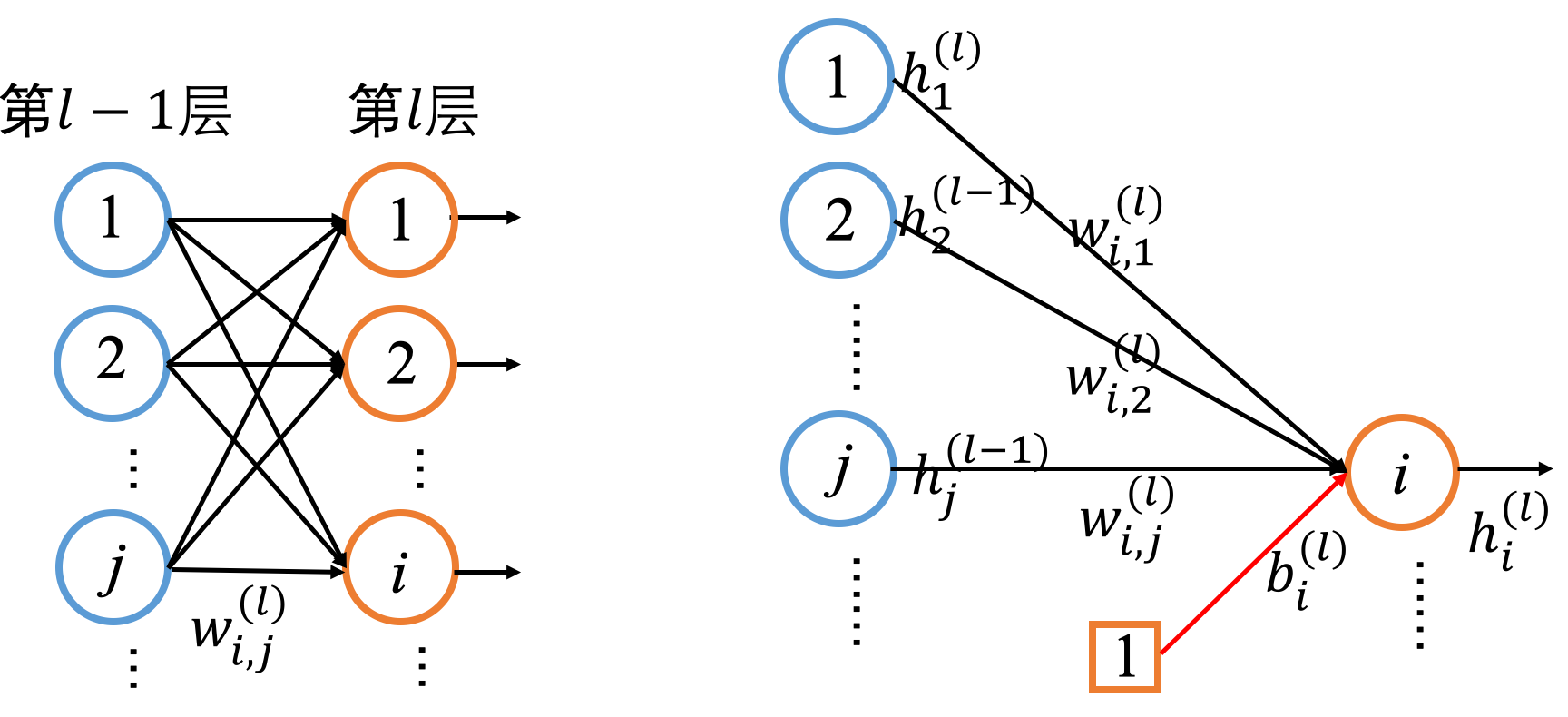
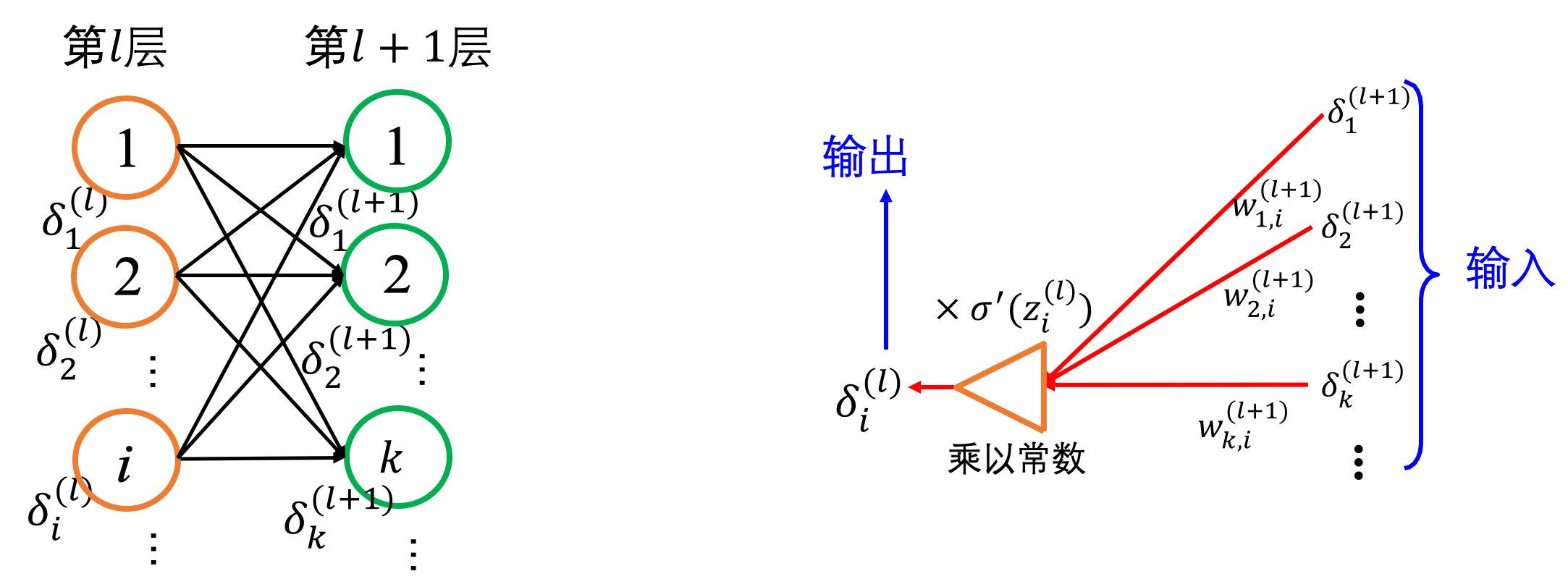
在训练DNN模型时，目标函数形式为：

|  |  |
| --- | --- |
|  | （9-6） |

其中为样本数，注意这里我们用表示样本索引（之前我们大多用表示样本索引，神经网络部分中我们用表示每层神经元的索引）。为损失函数（如回归问题可用L2损失、分类任务可用交叉熵损失），为正则项（如L2正则、L1正则）。在神经网络训练中，我们通常采用小批量（Mini-Batch）方式训练，这样为每批次的样本数目。下面的描述中，为了书写简洁，我们讨论只有一个样本的情况，多个样本对每个样本的损失函数值求和即可。

DNN的计算过程是从输入层，经过多个隐含层，传播到输出层。我们称之为正向传播，如图9-2（a）所示。给定输入，对全连接神经网络，正向传播的计算过程为：

|  |  |
| --- | --- |
| ，  ，  … …  ，  ，  … …  ，  ， | （9-5） |

（a）前向计算过程 （b）反向传播过程

图9-2 DNN的前向计算和反向传播

采用梯度下降法，参数的更新公式为：

|  |  |
| --- | --- |
|  | （9-8） |

其中为学习率。

为了更清楚地看到每个元素的影响，我们下面按元素展开进行细节讨论。

根据微积分链式法则，偏导可以写成两个因式相乘：

|  |  |
| --- | --- |
|  | （9-9） |

根据前向计算过程，第二项为

|  |  |
| --- | --- |
|  | （9-10） |

第一项记为，当时，得到

|  |  |
| --- | --- |
|  | （9-11） |

当时，得到的方向传播递推公式：

|  |  |
| --- | --- |
|  | （9-12） |

写成矩阵形式为：

|  |  |
| --- | --- |
|  | （9-13） |

其中表示按元素乘。

之间的关系看起来也类似一个网络，表示为图9-2（b）。所以

|  |  |
| --- | --- |
|  | （9-14） |

类似的，可得到参数的更新公式：

|  |  |
| --- | --- |
|  | （9-15） |

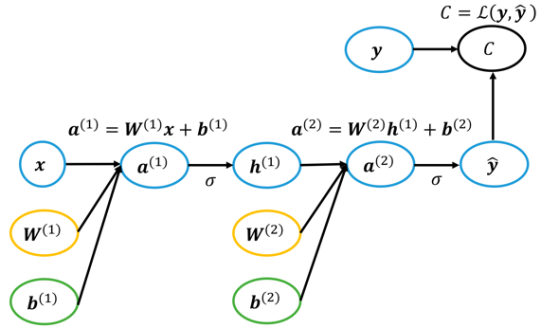
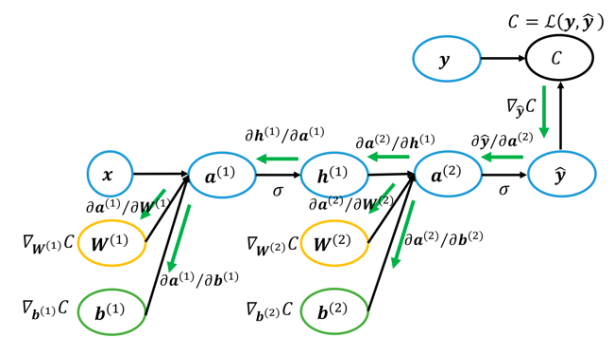
或者写成矩阵形式：

|  |  |
| --- | --- |
|  | （9-16） |

由于根据链式规则计算梯度时，误差从网络的输出层经过隐含层逐层传播，与前向计算方向相反，神经网络的梯度下降法被称为反向传播（Back-Propagation，BP）算法。

### **9.1.4** **DNN**的计算图

图9-3给出了有2个隐含层的DNN的计算图，这里目标函数我们暂时只考虑损失函数，没包括正则项。

（a） （b）

图9-3 有2个隐含层的DNN的计算图

（a）中给出了前向计算过程：

|  |  |
| --- | --- |
| ， | （9-17） |

要计算损失函数对每个变量的梯度，我们从输出层往前，反向计算每条边对应的偏导数。

第一条边，和我们具体任务的损失函数有关。如对回归问题的L2损失，标签一般为标量：，则。如果是分类任务，一般将原始类别标签用独热编码为向量，如类分类中一个样本的类别为，编码成向量，，所以

|  |
| --- |
|  |

第二条边为向量对向量的雅可比矩阵：

|  |  |
| --- | --- |
| 。 | （9-18） |

接下来根据计算：

|  |  |
| --- | --- |
|  | （9-19） |

为向量对矩阵求导，令分别为第2个隐含层和第一个隐含层的神经元的数目，我们可以将矩阵扁平成一个向量，然后再计算向量对对应向量的雅可比矩阵，雅可比矩阵矩阵的大小，我们用下标表示雅可比矩阵的行索引，表示雅可比矩阵的列索引。我们将展开成标量形式：

|  |  |
| --- | --- |
| 。 | （9-20） |

当时，，

当时，，

所以

|  |
| --- |
| 。 |

（9-21）

类似地，可得到：

|  |  |
| --- | --- |
|  | |
|  | （9-22） |

最后，我们得到各个参数的梯度：

|  |  |
| --- | --- |
|  | （9-23） |

结果同直接利用链式法则相同。

### **9.1.5** **CNN**的反向传播算法

CNN中包含卷积层和池化层，下面我们分别对池化层和卷积层反向传播算法展开讨论。

#### 池化层

在前向传播时，池化层一般我们会用最大池化或者平均池化，池化的区域大小已知，池化后分辨率降低。现在我们反过来，要从低分辨率后的误差，还原高分辨率中较大区域对应的误差。

在反向传播时，首先会把的所有子矩阵矩阵大小还原成池化之前的大小。如果是最大池化，则把的所有子矩阵的各个池化局域的值放在之前做前向传播算法得到最大值的位置。如果是平均池化，则把的所有子矩阵的各个池化局域的值取平均后放在还原后的子矩阵位置。池化层的反向传播我们记为：，完成池化误差矩阵放大与误差重新分配的操作。

下面我们用一个例子来说明这个过程。假设池化区域大小是。的第个子矩阵为：

|  |  |
| --- | --- |
|  | （9-27） |

由于池化区域为，我们先将做还原，即变成：

|  |  |
| --- | --- |
|  | （9-28） |

如果是最大池化，假设之前在前向传播时记录的最大值位置分别是左上，右下，右上，左下，则转换后的矩阵为：

|  |  |
| --- | --- |
| 。 | （9-24） |

如果是平均池化，则进行平均：转换后的矩阵为：

|  |  |
| --- | --- |
| 。 | （9-25） |

#### 卷积层的反向传播

一个卷积层可以有多个卷积核（通道），各个卷积核的处理方法是完全相同且独立，为了书写简单，我们下面提到卷积核都是指卷积层中若干卷积核中的一个。对二维图像，DNN中我们将图像展开成一维向量，也是向量；在CNN中，是三维张量。

和DNN类似，我们首先考虑相邻两层之间的递推关系：

|  |  |
| --- | --- |
|  | （9-26） |

所以需要确定。和之间为卷积运算关系：

|  |  |
| --- | --- |
| 。 | （9-27） |

我们通过一个例子来体会卷积操作，进而推出其反向传播算法。

例：假设我们要处理如下卷积操作：

|  |
| --- |
| 。 |

该卷积操作可分解为下列等式：

*，*

*，*

*，*

*，*

所以，

同理，得到

，

，

，

，

，

，

。

将所有的式子都写出来，就会发现，我们可以用一个卷积运算来计算：

|  |  |
| --- | --- |
| 。 | （9-28） |

上面式子其实和DNN的类似，区别在于卷积求导时，卷积核被旋转了180度（）。翻转180度的意思是上下翻转一次，接着左右翻转一次。而在DNN中这里是矩阵的转置。

已经知道当前层的误差项，参考之前的计算，可以得到：

，

，

，

。

跟一样，我们可以用矩阵卷积的形式表示：

，

这样就得到了

|  |  |
| --- | --- |
| 。 | （9-29） |

对于，稍微有些特殊，因为是高维张量，而只是一个向量，不能像DNN那样直接和相等。通常的做法是将的各个子矩阵的项分别求和，得到一个误差向量，即为的梯度公式：

|  |  |
| --- | --- |
| 。 | （9-30） |

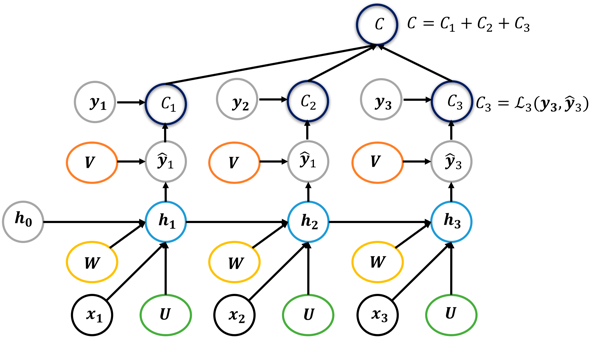
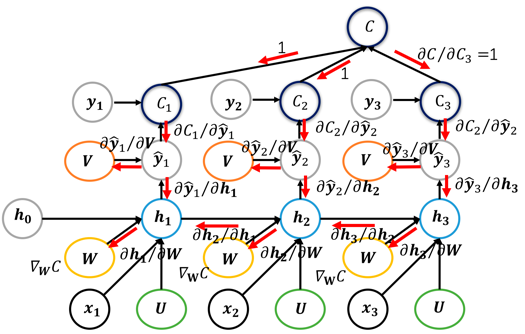
### **9.1.6** 循环神经网络的反向传播算法

RNN的正向传播依次按照时间的顺序计算，反向传播从最后一个时刻将累积的残差传递回来，跟普通神经网络的梯度计算本质上并没有不同。但是由于加入了时间顺序，计算的方式有所不同，称为基于时间的反向传播（Back Propagation Through Time，BPTT）算法。RNN中，不同时刻的参数是共享的，所以反向传播时我们更新的是相同的参数。

循环神经网络的输入是一整个序列，代表时刻的隐含状态，代表时刻网络的输出，为输入层到隐藏层直接的权重，为隐藏层到隐藏层的权重，为隐藏层到输出层的权重，分别为隐含层和输出层的激活函数。这里为了书写简洁，我们忽略偏置项。图9-4给出了一个时间长度为3的RNN的计算图。

RNN的前向计算为：

|  |  |
| --- | --- |
| *，*  *。* | （9-31） |

（a） （b）

图9-4 序列长度为3的RNN的计算图

对于RNN，序列的每个位置都有损失函数，因此最终的损失。

我们从计算图的输出层往前推进一层，这一层边对应的偏导数为：

|  |  |
| --- | --- |
|  | （9-32） |

再往前推进一层，从：

|  |  |
| --- | --- |
|  | （9-33） |

具体值与损失函数有关。

再往前推进一层，根据 ，。这里我们引入中间变量 ，因为在图中我们为了简化图，将激活函数和矩阵与向量乘合并成了一个结点。因为这是一个全连接层，所以我们直接将前面DNN的结论拿来代入即可：

|  |  |
| --- | --- |
|  | （9-34） |
|  | （9-35） |

其中为输出层结点数目，为输出层结点数目，为的矩阵：

。

到此为止，我们可以计算的梯度：

|  |  |
| --- | --- |
|  | （9-36） |

再往前推进一层，根据，。这里我们引入中间变量，这也是一个全连接层，我们直接将前面DNN的结论拿来代入：

|  |  |
| --- | --- |
|  | （9-37） |

注意同时具有和两个后续结点，因此，它的梯度由下式计算：

|  |  |
| --- | --- |
|  | （9-38） |

当时，只一个后续结点有，因此，它的梯度由下式计算：

|  |  |
| --- | --- |
|  | （9-39） |

再继续反向计算：，这里同样用到中间变量：

|  |  |
| --- | --- |
|  | （9-40） |

其中为的矩阵：

。

|  |  |
| --- | --- |
|  | （9-41） |

其中为的矩阵：

。

最后我们得到的梯度：

|  |  |
| --- | --- |
|  | （9-42） |
|  | （9-43） |

如果序列很长，即很大时，RNN模型在反向传播时会出现梯度消失或梯度爆炸，即高层若出现梯度较小或较大时，反向传播到低层时，梯度几乎为0或无穷大。为了更清楚地看到这一点，定义序列索引位置的隐藏状态的梯度为：

|  |
| --- |
|  |

我们像DNN一样从递推。

当时，

|  |
| --- |
|  |

*当*时，

从上面式子可以看出，计算涉及多次连乘，越大，连乘次数越多。所以当序列长度很大，展开的网络就越深，梯度计算中涉及的乘法次数越多。如果这些数值大于1，多次相乘的数值会无穷大，我们称之为梯度爆炸（Gradient Explode）；反之，如果单个数值小于1，多次相乘的数值会无穷大，我们称之为梯度消失（Gradient Vanish）。上面我们讨论的是简单RNN，LSTM和GRU就是为了克服RNN中的梯度消失或梯度爆炸做的改进。LSTM的梯度计算我们就不详细展开，读者可以自行练习。

最后安慰一下我们的读者，如果你已经被这一小节的梯度计算搞晕了也没关系，现在流行的深度学习平台都可以根据我们定义的前向运算，自动进行反向传播计算。

## **标题2**  9.2 激活函数

激活函数的主要作用是为神经网络提供非线性建模能力。如果没有激活函数，即使多个神经元连接也只能进行线性映射。只有加入了激活函数之后，深度神经网络才具备了分层的非线性映射学习能力。激活函数在模型训练中参与梯度的反向传播计算，所以激活函数的导数也会影响训练的收敛。

激活函数应该具有的性质：

• 可微性：当优化方法是基于梯度的时候，这个性质是必须的。

• 单调性：当激活函数是单调的时候，单层网络能够保证是凸函数。

• 非饱和性：饱和指的是在某些区间梯度接近于零，使得参数无法继续更新。

• 输出值的范围：有限的输出范围使得网络对于一些比较大的输入也会比较稳定（但可能会导致梯度消失，或者限制神经元表达能力）。当激活函数的输出是无限的时候（如ReLU），模型的训练会更加高效，不过在这种情况下，一般需要更小的学习率。

• 归一化（Normalization）：归一化的主要思想是使样本分布自动归一化到零均值、单位方差的分布（注意在前面几章我们用的术语是标准化，这里为了和神经网络领域的文献保持一致采用归一化表述），从而稳定训练。

图9-5给出了深度学习中常用的激活函数及其导数的图形，接下来我们结合图形来理解不同激活函数的特点。

|  |  |  |
| --- | --- | --- |
|  |  |  |
| （a）Sigmoid | （b）tanh | （c）ReLU |
|  |  |  |
| （d）Leaky\_ReLU | （e）ELU | （f）SELU |

图9-5 深度网络中常用的激活函数及其导数

### **Sigmoid**

函数将输入值变换到0和1之间：

|  |  |
| --- | --- |
|  | （9-44） |

函数在定义域内处处可导。但的导数最大值为0.25，这意味着反向传播时，返回网络的误差将会在每一层收缩至少75%，从而会导致梯度消失问题。且函数两侧导数逐渐趋近于0。这种导数趋近于0的性质我们称之为软饱和（与之对应的的硬饱和指导数等于0）。因此一旦输入落入饱和区，激活函数的导数就会变得接近于0，导致向前面层传递的梯度变得非常小。这种现象被称为梯度消失。一般来说， 网络在 5 层之内就会产生梯度消失现象。最近一些新的优化方法有效缓解梯度消失，例如逐层批正规化、Xavier权重初始化等。此时，网络参数很难得到有效训练。

另外sigmoid输出的均值并不为0，这会导致经过激活函数之后的输出，在作为后面一层的输入的时候均值非0，这个时候如果输入进入下一层神经元的时候全是正的，那么在更新参数时永远都是正梯度。例如，下一层神经元的输入是，参数是和，那么输出为，这个时候。所以如果是0均值的数据，那么梯度就会有正有负。不过这个问题并不是很严重，因为一般神经网络在训练的时候都是按批次进行训练的，可以在一定程度上缓解这个问题。

### **Tanh**

不是以零为中心的，更好的选择是一个函数。（双曲正切）函数可以将输入值变换到-1和1之间：

|  |  |
| --- | --- |
|  | （9-45） |

虽然函数的形状和函数的形状很像，但函数的输出值在坐标系的原点上对称，中心为零，因此使用激活函数收敛会更快，减轻消失梯度的现象。

### **ReLU**

整流线性单元（Rectified Linear Units，ReLU）函数是现在最常用的激活函数，提供了一个很简单的非线性变换：

|  |  |
| --- | --- |
|  | （9-46） |

ReLU计算量小（不涉及除法），计算成本低。另外ReLU使得一部分神经元的输出为0，造成网络稀疏，并且减少了参数之间的相互依存关系，缓解了过拟合问题。当输入为正数时，ReLU函数的导数为1，解决了梯度消失问题，收敛速度远快于和。

尽管输入为0时ReLU函数不可导，我们仍取此处的导数为0。然而，当输入为负数时，ReLU函数的导数为0，硬饱和。由于零值梯度而无法更新其权重，使得它们对于剩下的训练阶段沉默。这种现象被称为神经元死亡。为了缓解神经元死亡，一种方案是参数初始化时谨慎（如采用Xavier初始化），另外也可以将学习率设置得小一些，使得参数更新不要太大，或使用Adagrad等自动调节学习率的算法。ReLU还经常被“诟病”的一个问题是输出具有偏移现象，即输出均值恒大于零。偏移现象和神经元死亡会共同影响网络的收敛性。

### **LeakyReLU**

LeakyReLU是对ReLU的改进，主要是当时，会有一个很小的正梯度，具有非饱和行，减轻了神经元现象死亡。LeakyReLU的函数表达式为：

|  |  |
| --- | --- |
|  | （9-47） |

其中是一个很小的常数。参数也可以通过学习得到，称为参数化修正线性单元（Parameteric Rectified Linear Unit，PReLU）。

### **ELU**

指数线性单元（Exponential Linear Units，ELU）融合了sigmoid和ReLU，具有左侧软饱性， 其函数表达式为：

|  |  |
| --- | --- |
|  | （9-48） |

ELU继承了LeakyReLU的优点，左侧软饱和性质使得ELU对输入变化或噪声更鲁棒。但ELU包含指数运算，运算量大。

### **SELU**

缩放指数线性单元（Scaled Exponential Linear Units，SELU）为

|  |  |
| --- | --- |
|  | （9-49） |

其中参数、的值通过推导得到：

，

。

经过SELU激活函数后，样本分布自动归一化到0均值和单位方差。SELU不存在死区，输入大于零时，激活输出对输入进行了放大（导数为），但存在饱和区（负无穷时， 趋于）。

### **MaxOut**

MaxOut是深度学习网络中的一层网络，就像池化层、卷积层一样。我们可以把MaxOut看成是网络的激活函数层，假设激活函数层的输入特征向量为：，MaxOut层每个神经元的计算公式为：

|  |  |
| --- | --- |
|  | （9-50） |

其中。如果参数，网络就变成普通的DNN。所以相当于在传统的DNN中，在第层到第层，参数只有一组，分别为二维矩阵和一维向量。在MaxOut网络中，我们在这一层同时训练组参数，然后选择值最大的作为下一层神经元的输入，这样函数即充当了激活函数。

Maxout可视为一个可学习的分段线性函数。由于任何一个凸函数都可以由线性分段函数进行逼近，Maxout可以拟合任意的的凸函数。

但MaxOut明显增加了网络的计算量，使得原本只需要1组参数，变成需要在组中挑1组。

### 如何选择选择激活函数？

在实际应用中，我们选择激活函数的一般原则：

1．首选ReLU，速度快，但要注意学习速率；

2．如果ReLU效果欠佳，尝试使用LeakyReLU、ELU或MaxOut等变种。

3．可以尝试使用tanh。

4．Sigmoid和tanh在RNN（LSTM、注意力机制等）结构中作为门控或者概率值。其它情况下，减少Sigmoid的使用。

## **标题2**  9.3 深度学习中的优化算法

深度学习中最常用的优化算法是（一阶）梯度下降法。对为目标函数，假设其参数为**，**令目标函数对参数的梯度，梯度下降法中参数更新公式为：

|  |
| --- |
|  |

其中为学习率。除了计算梯度，梯度下降算法中还需要设置学习率。学习率是梯度下降算法中重要的超参数。如果学习率设置得太小，收敛非常缓慢；而太大的学习率则会阻碍收敛，导致损失函数在最优点附近震荡甚至发散。

如果将学习率设置成海森矩阵的逆矩阵，得到（二阶）牛顿法：

|  |
| --- |
|  |

其中参数的二阶偏导组成的海森矩阵。牛顿法收敛速度快，但海森矩阵求逆矩阵的时间复杂度高（），不适合大数据。并且神经网络的目标函数通常严重非凸，这种情况下牛顿法的收敛性难以保证。即使是凸优化，也只有在迭代点离全局最优很近时，牛顿法才会体现出收敛快的优势。

深度学习中，梯度下降算法可能会在下述情况中遇到困难：

1．深度学习的目标函数非凸，可能存在多个局部极小值，如图9-6所示为ResNet56目标函数。而梯度下降只能找到局部极值，不能保证找到全局最优值；

2．可能会陷入峡谷地带。峡谷类似一个带有坡度的狭长小道，左右两侧是“峭壁”。在峡谷中，准确的梯度方向应该沿着坡的方向向下，但粗糙的梯度估计使其稍有偏离就撞向两侧的峭壁，然后在两个峭壁间来回**震荡**。

3．可能会陷入鞍点。对形似马鞍状的目标函数，一个方向两头翘（从这个方向看是极小值），另一个方向两头垂（从该方向看是极大值），而中间区域近似平地，即为鞍点。鞍点的梯度为0，因此一旦优化过程中不慎落入鞍点，优化很可能就会停滞。

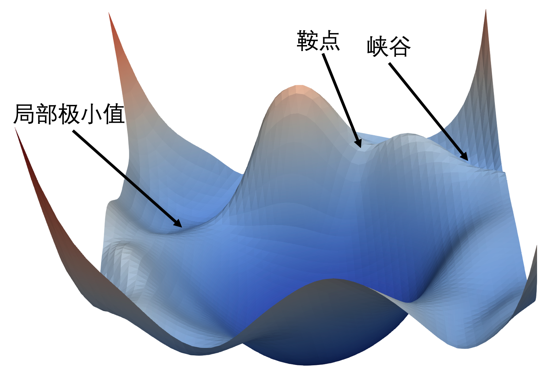


图9-6 ResNet56的目标函数[15]，目标函数包括多个局部极小值、**峡谷**地带和鞍点

随机梯度下降，或者小批量梯度下降中，每个批次用来计算计算损失函数的梯度的样本是随机选择的， 这意味着在某一特定点，每个批次得到的梯度实际上可能指向与所有样本损失函数的梯度略有不同。也就是说，尽管所有样本损失函数的梯度可能把参数推向一个局部极小值，或困在一个鞍点，但是这种随机的不同梯度有可能帮助我们避开这些情况。

梯度下降其实是在让参数朝负梯度方向走一步，步长为学习率。所以一是要找好方向，二是确定步长。为了克服基础梯度下降法的一些，研究者们提出了动量法，对梯度方向进行调整；也提出了自适应的学习率，对步长进行调整。

### **9.3.1** 动量法

动量法模拟物理世界中的惯性，参数的移动量不仅与梯度有关，还与上一时刻的移动量有关：

|  |  |
| --- | --- |
|  | （9-51） |

其中为动量因子，通常设为0.5、0.9、0.99。一般开始训练时小一些，后面大一些。当然也可以和学习率一样在训练时自适应调整。一般初始值是一个较小的值，随后会慢慢变大。动量算法引入了变量 ，相当于速度，即参数在参数空间移动的方向和速率。速度为之前所有梯度的加权和（负梯度的指数衰减平均），超参数决定了之前梯度的贡献衰减的多快：

|  |  |
| --- | --- |
|  | （9-52） |

在下降初期，相邻两次的移动方向相似，所以动量的引入会增大移动量，加速收敛过程。在下降中后期，局部最小值所在的吸引盆数量较多，一旦陷进吸引盆当中，梯度，但是前后两次更新方向基本相同。动量使得更新幅度增大，协助跃出吸引盆。在遇到峡谷时，如果学习率不合适使得两次更新方向基本相反，在原地“震荡”，动量的因子使得更新幅度减小，减弱震荡现象。

从物理角度来看，负梯度代表力，推动粒子沿着目标函数表面下坡的方向移动。而可以看做是惯性，最终收敛到局部极小点。

涅斯捷罗夫动量法（Nesterov Accelerated Gradient, NAG）将梯度计算放在对参数施加当前速度之后，这样算法有了对前方环境预判的能力：

|  |  |
| --- | --- |
|  | （9-53） |

图9-7给出了例2-1广告数据集上梯度下降法、动量法和NAG的比较。可以看出，动量法确实可以加快收敛，尤其前几次迭代参数更新量大，迭代次数比梯度下降法少一半。在图9-7（b）中，虽然后面阶段搜索范围越过了最佳位置（学习率过大），这时两次更新方向相反，动量法会使得更新幅度减小，再慢慢回到最佳位置。在第二行的例子中，目标函数在竖直方向比在水平方向的斜率的绝对值更大，梯度下降法中参数在竖直方向比在水平方向移动幅度更大，在长轴上呈“之”字形反复跳跃，缓慢向最小值逼近。动量法中参数在竖直方向上的移动更加平滑，且在水平方向上更快逼近最优解，因为此时竖直方向的当前梯度与之前的梯度方向相反相互抵消，移动的幅度小。NAG由于提前预知了目标函数的信息，相当于多考虑了目标函数的二阶导数信息，类似牛顿法的思想，因此搜索路径更合理，收敛速度更快。

|  |  |  |
| --- | --- | --- |
|  |  |  |
| （a）梯度下降法 | （b）动量法 | （c）NAG |
|  |  |  |
| （d）梯度下降法 | （e）动量法 | （f）NAG |

图9-7 梯度下降法、动量法和NAG优化技术的比较

### **9.3.2** 自适应学习率

学习率是梯度下降法中重要的超参数。当不同参数的梯度度值有较大差别时，需要选择足够小的学习率使得自变量在梯度值较大的维度上不发散（如图9-7第二行），但这样会导致自变量在梯度值较小的维度上迭代过慢。所以所有参数在所有训练阶段都设置同一个的学习率不是一个明智的选择，因此研究者们提出了多种学习率自适应调整策略。

#### AdaGrad算法

AdaGrad为模型的每个参数独立设置学习率，每个参数的学习率反比于其历史梯度平方和的平方根。随着优化过程的进行，对于已经下降很多的变量，则减缓学习率；对于还没怎么下降的变量，则保持一个较大的学习率。从而减缓陡峭区域的下降过程、加速平坦区域的过程。令，，AdaGrad的参数更新公式为：

|  |  |
| --- | --- |
|  | （9-54） |

其中表示向量按元素乘，为对向量按元素求平方根，初始学速率一般设置为0.01，通常取很小的数，如。

AdaGrad算法具有一些令人满意的理论性质。但从训练开始就积累梯度平方会导致有效学习率过早和过量减小，AdaGrad算法在迭代后期由于学习率过小，可能较难找到一个有用的解。

#### RMSProp

为了缓解AdaGrad算法中学习率衰减过快的问题，RMSprop算法改梯度累积为指数衰减的移动平均（类似动量法），以丢弃遥远的过去历史：

|  |  |
| --- | --- |
|  | （9-55） |

RMSProp被证明是一种有效且实用的深度神经网络优化算法。

#### Adam

Adam（ADaptive Moments）算法将动量和RMSprop结合：

|  |  |
| --- | --- |
|  | （9-56） |

所以Adam算法同时获得了AdaGrad和RMSProp算法的优点。在实际应用中，Adam方法效果良好。当然，这里的动量也可以换成涅斯捷罗夫动量，得到Nadam。

在实际应用中，Adam通常是一个很好的选择。Adam的收敛速度比梯度下降快，但最终收敛的结果并没有梯度下降好，主要是后期Adam的学习率太低，影响收敛。学习率下降的动量法通常表现不错，不过需要仔细调整学习率。建议训练前期采用Adam，享受Adam快速收敛的优势；后期切换到梯度下降，慢慢寻找最优解。

## **标题2**  9.4 权重初始化

梯度下降法等迭代优化算法还需要设置一个参数的初始值。对简单的机器学习模型，如Logistic回归，简单的将模型参数初始化0或较小的随机数即可。然而对于深度学习而言，由于目标函数非凸，层次深，如何选择参数初始值便成为一个值得探讨的问题。

深度网络模型的偏置参数通常设置为。模型权重的初始化对于网络的训练很重要，不好的初始化参数会导致梯度传播问题，降低训练速度；而好的初始化参数能够加速收敛，并且更可能找到较优解。

### 全**0**

对神经网络，不能将所有权重都初始化为0或相同的值。以带一个隐含层，一个输出结点的DNN为例，假设隐藏层结点使用相同的激活函数。如果将每个隐藏单元的参数都初始化为相等的值，由于网络中神经元的更新机制完全相同和网络的对称性，在正向传播时每个隐藏单元将根据相同的输入计算出相同的值，并传递至输出层。在反向传播中，每个隐藏单元的参数梯度值相等。因此，这些参数在使用基于梯度的优化算法迭代后值依然相等。之后的迭代也是如此。在这种情况下，无论隐藏层的结点有多少，隐藏层本质上只有1个结点在发挥作用。因此我们通常对神经网络模型的权重参数，进行随机初始化。

### 随机数初始化

一种可选的方案是将权重初始化为较小的随机数，如高斯分布或均匀分布抽样。但随着网络层数的增加，神经网络各层输出值分布的方差随着层数的增大而增大，隐藏层的输入的方差过大，会在经过激活函数时落入饱和区，过早地出现梯度消失。

这些方差的变化可以根据前向计算推导。前向传播为（忽略偏置项）：

|  |
| --- |
|  |

其中为激活函数。计算的方差为：

其中第二行、第三行假设各个变量互相独立，第三行到第四行是假设数据和权重都是中心化的，即，。

对第一个隐含层，输入，假设输入各维的方差相等，记为，每个权重的方差也相等，记为，则。如果激活函数为线性函数，则。

类似，对第层，。如果总是大于1，那么随着层数越深，方差会越来越大，最后导致溢出；反过来，如果乘积小于1，那么随着层数越深，方差就会越来越小，就容易导致数据差异小而不易产生有力的梯度。

### **Xavier**初始化

Xavier初始化的基本思想是保持各层的激活值和梯度在传播过程中方差保持一致。为了问题的简便，Xavier初始化的推导过程是基于线性函数的，但在一些非线性神经元中（激活函数取或）也很有效，因为当很小时，他们的乘积之和也较小，和在0附件的表现与线性相似，梯度接近1。

令输入的方差，如果第一层输出的方差与输入的方差都保持一致，则第1层输出的方差为：

|  |
| --- |
|  |

得到。

继续前向传播，对第层，得到，其中表示该层输入的数目（前一层神经元的数目）。

类似的，反向传播时，。如果也保持梯度方差保持一致，即

|  |
| --- |
|  |

得到：，其中表示该层输出的数目（本层神经元的数目）。

显然当且仅当时，才能保证正向和反向的方差一致。当二者不一致时，可以综合一下，使用下式：

|  |  |
| --- | --- |
|  | （9-57） |

均匀分布的方差为，因此根据方差可反过来得到的分布边界：

|  |  |
| --- | --- |
|  | （9-58） |

Xavier初始化假设激活函数为线性，所以对采用激活函数的网络有效，但对采用ReLU激活函数的网络无能能力。因为当输入小于0时，ReLU激活函数输出为0，不满足Xavier初始化激活函数近似线性的假设。下面讨论的He初始化就是针对ReLU激活函数，对Xavier初始化的改进。

### **He**初始化

在ReLU网络中，假定每一层有一半的神经元被激活（输入大于0），另一半为0，所以，要保持方差不变，只需要在Xavier初始化的基础上再除以2，得到He初始化（亦被称为MSRA初始化）：

|  |  |
| --- | --- |
|  | （9-58） |

或

|  |  |
| --- | --- |
|  | （9-59） |

### 批量归一化（**Batch Normalization, BN**）

从图9-7中我们知道对输入数据做标准化处理（特征的均值为0、标准差为1），使得各个特征的分布相近，这样更容易训练出有效的模型。对深层网络，我们希望每层网络的输入也有类似性质。批量归一化正是这样一种的方法，巧妙而粗暴地将每层的输出值强行做一次归一化和线性变换，以达到输入输出方差相等的目的，从而使得深度网络的训练变得可行。深度网络训练通常都是以小批量（Mini-Batch）方式进行，因此这里的归一化也是以小批量为单位进行。

对全连接层，批量归一化层置于全连接层中的线性组合和激活函数之间。设全连接层的第输入为，权重参数和偏差参数分别为和，激活函数为。设批量归一化的运算符为，则使用批量归一化的全连接层的输出为，其中批量归一化输入为。

考虑一个由个样本组成的小批量，为了简化归一化操作，BN中归一化对每维特征单独进行：

|  |  |
| --- | --- |
|  | （9-60） |

其中分别表示样本索引和特征索引，是一个很小的常数，保证分母大于0。

经过上述归一化后，每层输入每个特征的分布均值为0，方差为1。但这种归一化操作也会降低神经网络的表达能力，使得底层网络学习到的参数信息丢失。另一方面，通过让每一层的输入分布均值为0，方差为1，会使得输入在经过或激活函数时，容易落在非线性激活函数的线性区域。

为了恢复数据本身的表达能力，BN对归一化后的数据进行线性变换，引入了两个可学习的参数与：

|  |  |
| --- | --- |
|  | （9-61） |

当时，可以实现等价变换并且保留了原始输入特征的分布信息（如果批量归一化无益，学出的模型可以不使用批量归一化）。注意，在归一化过程中减去均值，因此偏置项可以忽略掉或可以被置为0。

对卷积层来说，批量归一化发生在卷积计算之后、应用激活函数之前。如果卷积计算输出多个通道，我们对这些通道的输出分别做批量归一化，且每个通道都拥有独立的拉伸和偏移参数。在单个通道上，设小批量中有个样本，卷积计算输出的高和宽分别为和，我们需要对该通道中个元素同时做批量归一化。对这些元素做标准化计算时，我们使用相同的均值和方差，即该通道中个元素的均值和方差。

使用批量归一化训练时，我们可以将批量大小设得大一点，从而使批量内样本的均值和方差的计算都较为准确。将训练好的模型用于预测时，我们希望模型对于任意输入都有确定的输出。因此，单个样本的输出不应取决于批量归一化所需要的随机小批量中的均值和方差。一种常用的方法是通过移动平均估算整个训练数据集的样本均值和方差，并在预测时使用。可见，和丢弃层一样，批量归一化层在训练模式和预测模式下的计算结果是不一样的。

在BN中，由于我们使用小批量的均值与方差作为对整体训练样本均值与方差的估计，不同批次的均值与方差会有所不同，这为网络的学习过程中增加了随机噪声，这种噪声与丢弃通过关闭部分神经元输出给训练带来噪声类似，在一定程度上对模型起到了正则化的效果。BN的作者验证了网络加入BN层后，可以去掉丢弃层，模型也同样具有很好的泛化效果。

### 预训练

预训练（pre-training）是一种非常有效的神经网络的初始化方法。一种方式是采用非监督的方式，贪心地逐层训练自编码器得到初始权重，然后再做细调（fine-tuning）。

不过这种方式现在已经不常用了，一种更有效的方式是从类似问题中已经训练好的模型入手。一些著名的研究团队公布了许多预训练好的模型，如和。不过在选择预训练模型的时候需要非常仔细，需要考虑新数据集与原始数据集之间的相似度和新数据集的规模。

* 若数据集与预训练模型采用的训练数据集非常相似，且新数据集较小，只要在预训练模型最顶层输出特征上再训练一个线性分类器即可；
* 若新数据集较大，可以使用一个较小的学习速率对预训练模型的最后几个顶层进行调优。如果新数据集与预训练模型采用的训练数据集相差较大但拥有足够多的数据，则需要对网络的多个层进行调优，同样也要使用较小的学习速率。
* 最坏的情况是新数据集不仅较小，而且与原始数据集相差较大，此时基于较为靠前的特征层就使用SVM分类器可能是相对较好的方案。

### 权重初始化建议

• 使用ReLU激活函数（无BN）时，最好选用He初始化方法，将参数初始化为服从高斯分布或者均匀分布的较小随机数。

• BN的使用减少了网络对参数初始值尺度的依赖，此时使用较小的标准差（如0.01）的高斯分布进行初始化即可。

• 借助预训练模型中参数作为新任务参数初始化的方式也是一种简便易行且十分有效的模型参数初始化方法。

## **标题2**  9.5 减弱过拟合策略

深度神经网络通过大量的参数，能拟合各种复杂的数据集。这种独特的能力使其能够在许多复杂任务表现优异。然而模型在学习过程时，如果缺乏控制可能会导致过拟合现象的发生——模型在训练集上表现很好，但对新的测试数据预测时效果不好。本节我们讨论一些控制模型过拟合的方法。

### 数据增广

减少过拟合的一种方式是增加训练样本数量。但增加训练样本数量成本高，一种解决方案是对已有的训练样本进行一些处理，来“制造”出更多的样本，称为数据增广（Data Augmentation）。如对已有的训练图像进行水平翻转、垂直翻转、任意角度旋转、缩放或扩大、添加随机噪声、随机破坏图片的一部分等。当然随着训练样本增多，模型的训练时间也会相应增长。

### 正则

正则通过在目标函数中增加正则项控对模型的复杂施加惩罚，减轻过拟合。同其他机器学习模型一样，深度网络中我们也可以考虑L2正则或L1正则。

在深度学习模型中，加入L2正则后的目标函数为：

|  |  |
| --- | --- |
|  | （9-62） |

与不加正则的目标函数 相比，加入正则项后，梯度下降算法中的计算表达式需要做如下修改：

|  |  |
| --- | --- |
|  | （9-63） |

L2正则也被称做权重衰减（Weight Decay），因为加上正则项，有个增量，在更新的时候，会多减去这个增量，使得比没有正则项的值要小一些：

其中。

### 训练提前停止

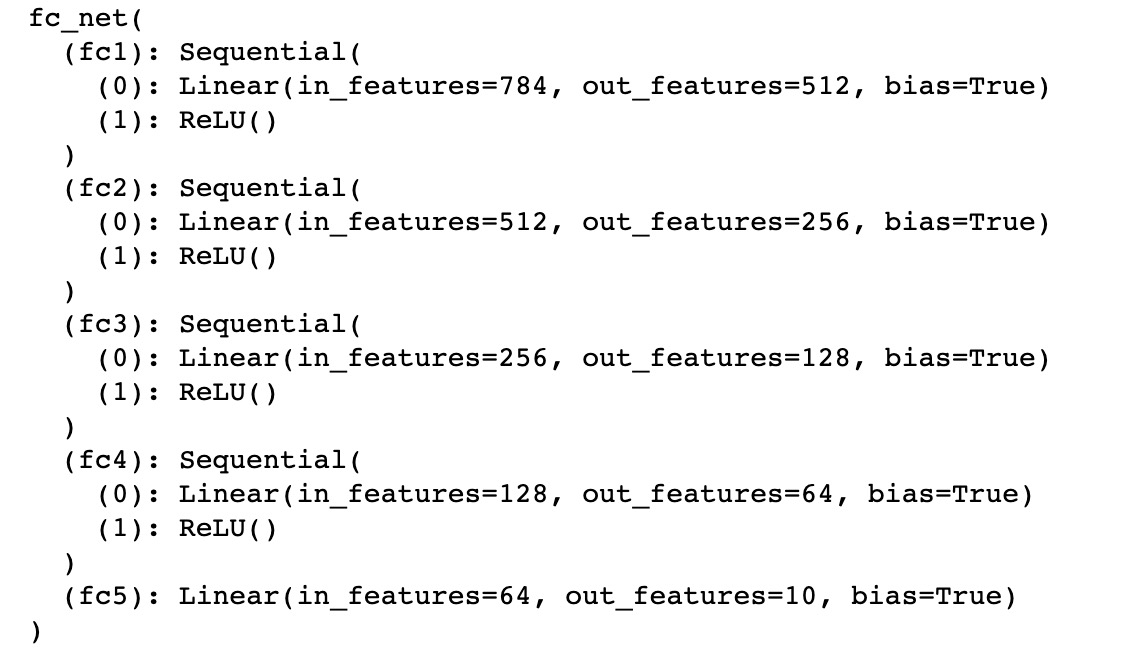
神经网络模型通常采用迭代方法训练，训练误差会随着迭代训练次数增加而单调减小，但在验证集上，通常误差会先减小，之后又增大。即训练次数过多时，模型会对训练样本拟合的越来越好，但是对验证集拟合效果逐渐变差，即发生了过拟合。因此迭代训练次数不是越多越好，可以通过监控训练误差和验证误差随迭代次数的变化趋势，选择合适的迭代次数，提前停止训练。提前停止其实在其他机器学习模型（如决策树、GBDT）的训练中也用到。

## **标题2**  9.6 深度神经网络应用案例——**MNIST**手写数字识别

我们在MNIST数据集进行深度神经网络练习，MNIST数据集介绍请见5.5.3节。

#### 1．全连接神经网络（DNN）

全连接神经网络由全连接层组成，每个全连接层后面接一个激活函数层进行非线性映射。试验中的网络的结构如下所示：



网络的第一个全连接层的输入维度为784，即为28\*28的黑白图像摊平之后的维度，前四个全联接层后面均连接ReLU激活函数，最后的fc5层将fc4层的64维的输出维度变为10维，输出图像所属每个类别（0-9）的概率。

训练时我们从42,000张训练图像中抽取除20%的图像作为验证集，用于监控训练过程的收敛情况。训练采用小批量的方式进行，每批次中的训练样本的数目为32。每轮训练后训练误差和验证集上的误差如图9-8所示，可以看出在20轮左右训练过程已经收敛。试验中优化器为，损失函数选择交叉熵损失函数（多分类问题）。注意PyTorch中的反向传播函数在进行计算时是将梯度累积起来而不是替换掉，因此在每一个批次开始的时候我们要先将优化器的梯度置零。对每批次的训练数据，网络在得到输出之后，将输出结果以及标签传入到损失函数中并得到该批次的损失函数的值，最后使用对损失函数进行反向传播，并使用对网络参数进行更新。在验证时，我们像训练一样计算损失函数，不同的是，我们不需要将损失函数进行反向传播来改变模型参数，因此在验证时我们使用来将整个训练过程包起来。在训练结束后，我们将模型参数保存起来，便于测试时读取模型。

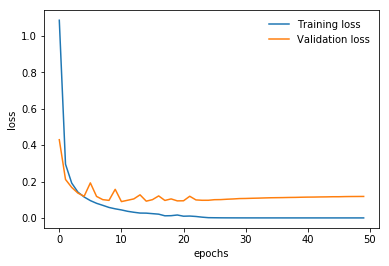


图9-8 DNN在MNIST数据集上的训练误差和验证误差。

#### 2．卷积神经网络CNN

与全连接网络DNN不同，CNN网络主要是由卷积层而非全连接层组成的。试验中的网络结构如下所示：

图片包含 屏幕截图

描述已自动生成

网络中卷积层提取输入图像的特征，并将最后得到的输出特征传入分类器中进行分类。在每一个层之后都有一个层以及激活函数。在最后一个层的激活函数之后，有一个（最大池化层）。在提取特征之后，特征被传入到由3个全连接层组成的分类器中。每一个全连接层都使用了丢弃层来防止过拟合，并在每一个全连接层之后都有一个层与激活函数。与DNN的结构类似，最后一层的全连接层的输出维度为图像的类别数。

CNN的训练损失与验证损失的图像如图9-9所示，可见模型在迭代40轮左右收敛。

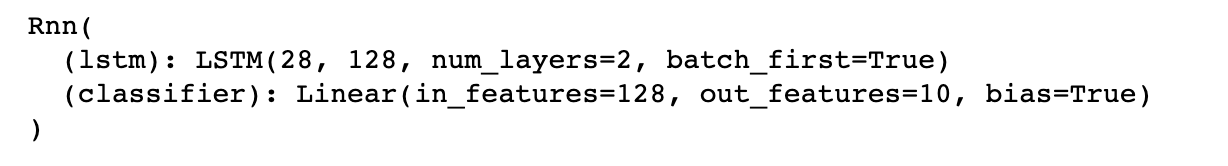
图片包含 屏幕截图, 地图

描述已自动生成

图9-9 CNN在MNIST数据集上的训练误差和验证误差。

#### 3．循环神经网络RNN

我们使用RNN中最具有代表性的LSTM作为主体，并在其之后使用一个全连接层来将LSTM获取的隐藏状态的维度映射到类别数目上。试验中RNN的网络结构如下所示：



其中LSTM的输入维度为28，即图像的每一行作为LSTM的一个时刻的输入。输出维度128为隐藏单元的大小，LSTM的隐藏层的数目。与之前两个网络类似，全连接层的作用是将128维的特征映射到10维，即类别数目上。

LSTM的训练误差以及验证误差与迭代轮数之间的关系如9-10所示。可以看出，模型在训练10轮后基本收敛。

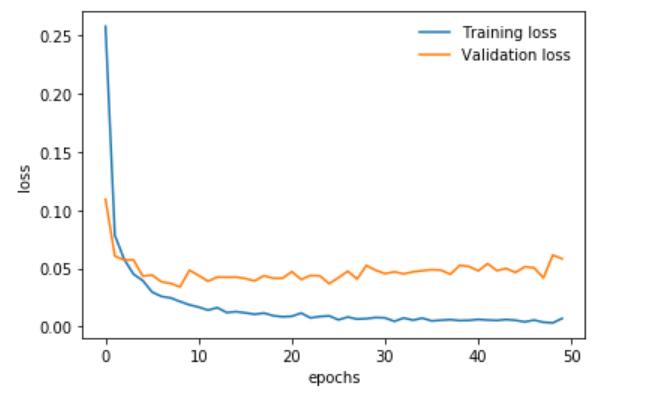


图9-10 LSTM在MNIST数据集上的训练误差和验证误差。

## **标题2**  9.7 小结

训练深度网络一般采用小批量的梯度下降算法。本章我们学习了深度网络的梯度计算方法：反向传播，并对梯度下降中的各个因素进行了分析，包括：权值初始化、自适应学习率、动量法修正梯度方向。深度网络模型复杂，容易过拟合，我们还讨论了一些缓建模型过拟合的技术。

## **标题2**  9.8 练习

1. 假设你建立了一个神经网络，并将权重和偏差初始化为零。以下哪些陈述是正确的？

（A）第一隐藏层中的每个神经元将执行相同的计算。因此，即使在梯度下降的多次迭代之后，层中的每个神经元将计算与其他神经元相同的东西。

（B）第一隐层中的每个神经元在第一次迭代中执行相同的计算。但是在梯度下降的一次迭代之后，他们将学会计算不同的东西。

（C）第一个隐藏层的每个神经元都会计算相同的东西，但不同层的神经元会计算不同的东西。

（D）即使在第一次迭代中，第一个隐藏层的神经元也会彼此执行不同的计算，参数将以自己的方式不断演化。

2. 在神经网络中，下列哪些方法可以防止过拟合？

（A）丢弃

（B）批量归一化

（C）正则

3. 下列技术中，哪些对减少过拟合）有用？

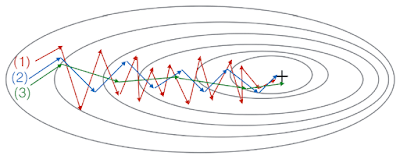
（A）L2正则化

（B）Xavier初始化

（C）丢弃法

（D）数据增广

4. 下图中对目标函数用3种优化方法求解：（a）梯度下降、（b）动量参数的梯度下降和（c）动量参数的梯度下降，请问曲线（1）、（2）和（3）分别对应哪种优化方合？



5. ReLU激活函数有哪些的优缺点。

6. 梯度消失问题是如何产生的？如何解决？

7. 对MNIST数据集，采用5层DNN模型，每个隐含层的结点数目同9.6节，在每层中加入BN层，比较增加BN层后训练的收敛速度和性能。

8. 对MNIST数据集，采用9.6节的CNN模型，但去掉全连接层的丢弃层，比较去掉丢弃层后模型的性能。

9. 对MNIST数据集，采用5层DNN模型，每个隐含层的结点数目同9.6节，在每层中加入BN层，比较不同优化算法对训练收敛速度的影响（不同学习率设置策略、是否带动量）。