英文标注：

p3

prescanned 预先规划的

item 这里指可迭代对象中元素数目

spot 注意到

as we covered in... 在...中提到的

p6

anticipated 预期

p7

handy 方便的

augmentation 扩充

data augmentation 数据增强

intuition 直觉

hence 因此

unitary 单位

deviation 偏差

manipulate 操纵

fleet 舰队，一队

shot 镜头

frame 相框

arbitrarily 随意的，武断的

categorica 明确的

differentiable 可微的

tricky 难办，不好处理

slope 坡度，斜率

saturates 饱和

underlying 潜在的，根本的

compensate for 补偿

stochastic 随机

shuffled data 对数据打乱重新排序

At a minimum 最少，至少

Python的MRO：

MRO 方法解析顺序：对于支持继承的编程语言来说，其方法（属性）可能定义在当前类，也可能来自于基类，所以在方法调用时就需要对当前类和基类进行搜索以确定方法所在的位置。而搜索的顺序就是所谓的「方法解析顺序」（Method Resolution Order，或MRO）。

C++中使用虚函数，虚拟基类等方式来确定使用哪种方法。

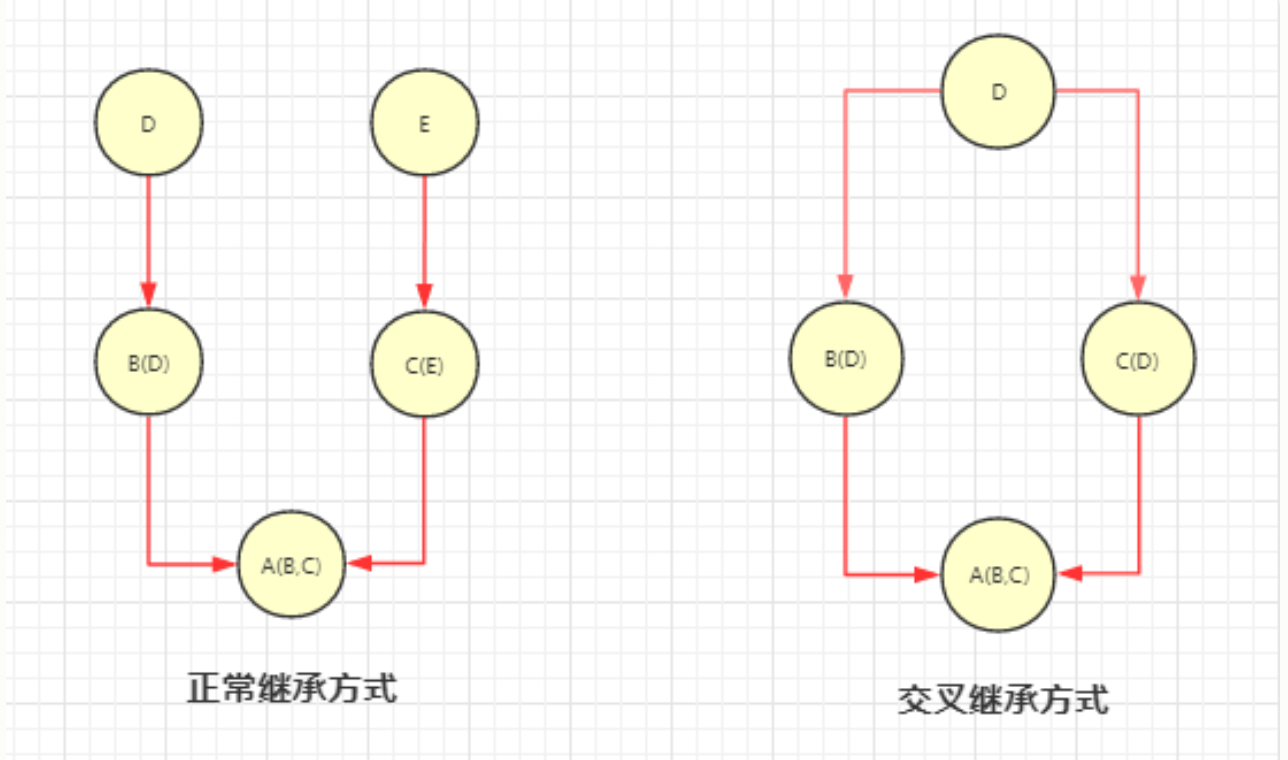
Python支持多继承所以MRO比较复杂。

不同版本的Python的MRO方法不同，

经典类：DFS深度优先搜索（Python2.2以前的版本）

新式类：BFS广度优先搜索（Python2.2中提出，在与经典类共存的情况下，是否继承object是他们的区分方式）

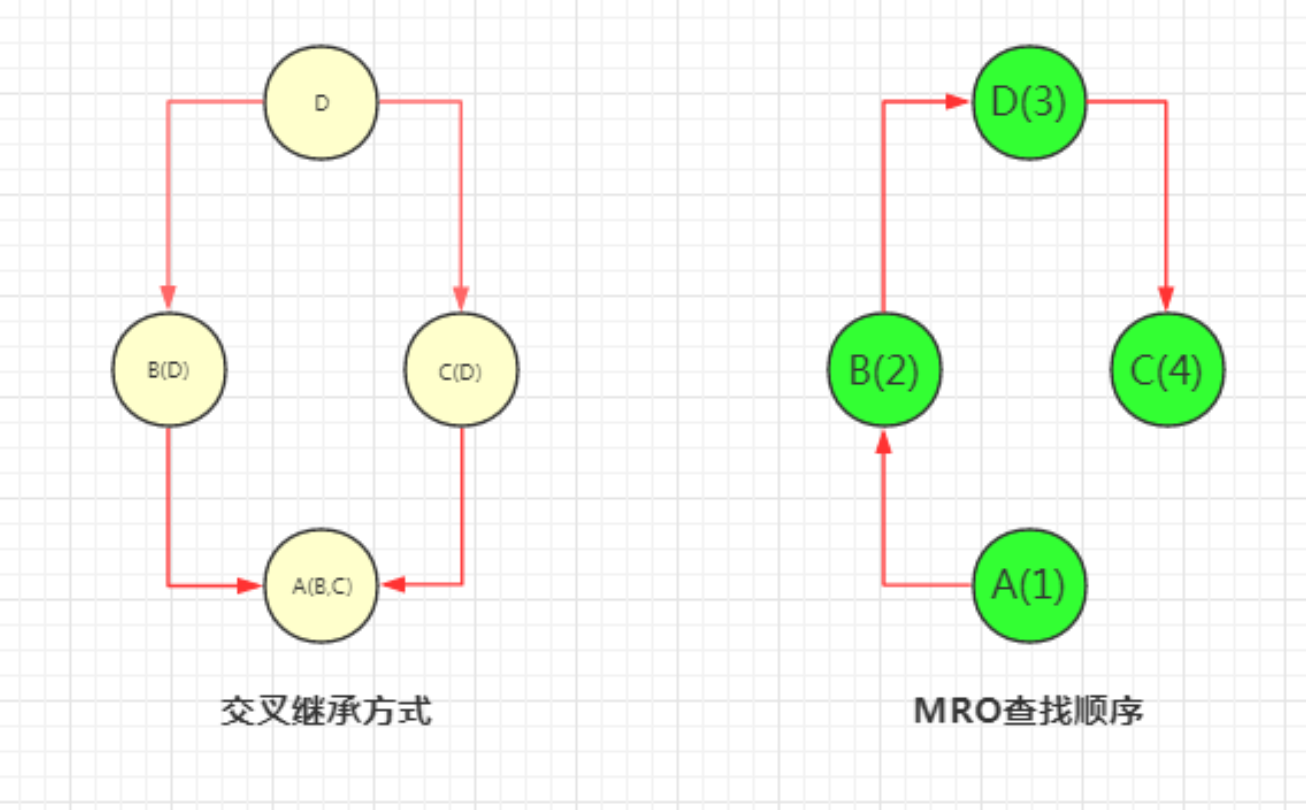
**新式类C3算法：Python2.3提出（也是现在Python3唯一支持的方式）**



在经典类中，没有\_\_mro\_\_属性可以去查看MRO的顺序，但是，可以使用inspect模块中getmro方法

import inspect

inspect.getmro(类名)



P12

（）为了能够像第六章第三节中讨论的那样学习任意函数我们需要至少一个非线性的隐藏层，否则他就只是一个线性模型。隐层特征通过学习能够表达输入之间的关系。这些输入是通过权重矩阵进行编码的。因此这个模型可能学习到比较向量的第176和208两个元素，但是这不是一个提前注意到的关注点，因为在结构上是很难注意到的，实际上他们分别是第6行第16个元素和第七行第16个元素因此他们是相邻的。

7.2.3

在第六章中，神经网络才产生了一个定量预测的温度作为输出。我们可以在这里做相似的事情。使我们的网络输出一个标量值（n\_not=1），如果值为0.0代表飞机，1.0代表鸟。使用他们作为MSELoss（均方损失函数）的目标函数。通过这样做我们能够将这个问题变为一个回归问题。然而，再仔细看看，我们在处理在本质上有点不同的问题。

我们需要意识到这是一个分类问题：它要么是鸟或者一个飞机。如我们在第四章学到的，当我们需要表达一个分类变量我们需要将变量转化为独热编码表达，例如[1,0]对于飞机，[0,1]对于鸟（顺序是任意的）。如果我们分为了10个类，这样还是有效的。在整个CIFAR10数据集上时，我们使用一个长度为10的向量。

在理想的情况下，神经网络应当对于飞机输出torch.tensor([1.0, 0.0])对于鸟输出torch.tensor([0.0, 1.0])实际上因为我们的分类器并不是完美的，我们可以预测到神经网络的输出应当是在两者之间。在这次实验中关键是意识到我们可以将我们的输出解释为可能性，输出向量的第一个数是飞机的可能性，第二个数是鸟的可能性。

以概率的形式来处理这个问题会给我们网络的输出提出一些额外的限制。输出的每个元素的值必须在0~1之间，所有元素的和为1.0.我们确信两者其中之一一定会发生。这听起来像是一个严格的约束，以可微的形式来对数值向量进行变换达到这个要求。但是有一个很好的方法来完成它，且它是可微的，叫做softmax函数。

7.2.4将输出表示维概率

Softmax是以一个使用一个向量作为参数然后输出另一个相同维度向量的函数，使用这个函数产生的向量能够满足我们上述的为了表示概率的限制。

在Softmax中我们使用向量的每一个元素来进行幂运算，以e为底数，以某一元素为指数计算每个元素对应的值，然后求和。对每个求出来的值除以他们的和。

在模型的最后加入一个softmax层来产生概率。

在这里为了调用这个模型，我们需要调整输入的维度。Torch的模型的层要求第0维是batch的数量所以我们需要将我们的3\*32\*32的图片变为3072的一维张量然后在第0维添加一个新的维度。我们在第三章学习了这项操作。

然后我们得到了预测的概率，但是现在还没到兴奋的时间。我们线性层的权重和偏移还没有进行学习。他们现在还是由pytorch在-1到1之间随机初始化的。有趣的是，我们还可以看到对于输出的grad\_fn这是反向计算时使用的提示，在我们进行反向传播时我们需要用到他。

·另外，我们知道输出概率对应与那个标签，但是我们的网络是不知道的，第一个概率对应飞机还是鸟。这是这个网络并不清楚这一点。在经过反向传播的训练之后，使用损失函数赋予这两个数意义。在训练之后我们将通过计算输出中最大的概率来得到其对应的标签。方便的是当我们给torch.max提供一个维度参数之后，他能返回一个对应维度上最大的索引值，在这个位置上取得最大值。我们需要一个取得概率向量的最大值，所有维度取1。如果维度取1，取得的是一个bactch中所有图片的鸟和飞机概率的最大值。

模型告诉我们这是一个鸟。很幸运，但是我们已经将我们的模型改为概率模型来对图片进行分类。我们现在已经使用一个输入图片跑了我们的模型，现在可以开始训练了。像伤两章中做的那样，我们需要在训练中设置一个损失函数，目标是优化参数使其最小。

7.25 分类的损失函数

我们刚提到损失函数赋予了概率意义。在第五章和第六章，我们使用了MSE作为损失函数。我们能够继续使用MSE作为损失函数并且使我们的输出概率在0~1之间，但是经过仔细思考，我真正感兴趣的并不是产生的标签值。返回去查看求参数最大值的操作时，我们曾经提取了预测的类的索引，我们真正注意的时第一个类的可能性更大反之亦然。也就是说，我们想要去惩罚错误的分类而不是煞费苦心地惩罚不想0.0或者1.0的结果。

在这个例子中我们想要最大化的是正确类的输出概率，out是sotfmax层的输出，class\_index是一个对应的字典。对应正确类别的概率的叫做在给定模型参数和数据的likelihood。也就是说我们想要一个损失函数当likelihood很小时具有很高的惩罚，这样可以使模型选择其他的类别。当正确选项的likelihood大于其他选项时，损失函数很小，我们的目的别不是将正确类别的概率提高到1.

下面是在训练过程中损失函数的变化。它叫做NLL。表达式是NLL = - sum(log(out\_i[c\_i])),。其中sum使用了N个例子c\_i是第i个例子的正确类别索引，让我们看图7.10.NLL作为预测概率的损失函数。

这个图显示了当输出结果中正确类别的概率很低时，损失函数无限大，当正确类别预测的概率高于0.5时下降的速度较小。NLL使用概率作为输入所以当正确类别的概率上升，错误类别的结果下降。

总结起来，我们分类器的损失函数这样计算：对于在batch中的每一个例子

首先进行前向传播，然后得到最后一个线性层的输出，

计算他们softmax然后取得对应类别的概率

得到在此参数下正确类别的概率。注意我们知道每个例子对应的类，因为我们使用的是一个有监督的数据集

计算他的对数值，然后加上一个符号，再进行求和得到损失

所以在pytorch中，我们应当如何做呢？pytorch有一个NLLLOSS类然而和你想的不一样，他不使用一个概率而是使用一个概率对数的张量作为输入。他之后计算给定的数据的NLL。输入约定背后有一个很好的理由。因为当概率接近0时计算一个概率的对数值是很不好的。变通方法时使用nn.LogSoftmax而不是nn.Softmax

这会使计算在数值上稳定。

损失使用nn.LogSoftmax层的输出作为第一个参数使用一个类标签的张量作为第二个参数，我们现在可以进行测试了。

结束我们对于损失函数的研究，我们现在看一看如何使用交叉熵损失函数优于MSE。在分类问题上，NLLLoss等价与交叉熵函数，在图7.11中，我们看到交叉熵损失的坡度，我们开始时舍弃的MSE方案，更容易饱和。潜在的原因时MSE的斜率太低不能补偿softmax层为错误的预测带来的平坦度。这是MSE不适合作为分类任务损失的原因。

7.26训练分类器

好了，我们现在准备好重新使用在第五章中写的训练代码，然后看看如何进行训练

Scatter\_函数 用一个源张量或者标量以及索引来修改另一个张量，可以用于做独热编码，其中参数dim是使用index进行替换的维度

适当的增加Batch\_Size的优点：

1.通过并行化提高内存利用率。

2.单次epoch的迭代次数减少，提高运行速度。（单次epoch=(全部训练样本/batchsize)/iteration=1）

3.适当的增加Batch\_Size,梯度下降方向准确度增加，训练震动的幅度减小。（看上图便可知晓）

经验总结：

相对于正常数据集，如果Batch\_Size过小，训练数据就会非常难收敛，从而导致underfitting。

增大Batch\_Size,相对处理速度加快。

增大Batch\_Size,所需内存容量增加（epoch的次数需要增加以达到最好的结果）

这里我们发现上面两个矛盾的问题，因为当epoch增加以后同样也会导致耗时增加从而速度下降。因此我们需要寻找最好的Batch\_Size。

一个iteration包括一个batch中数据的前向传播和反向的求导梯度更新。Epoch代表使用数据集训练了多少次。

Stochastic gradient descent

SGD这是对整个数据集估计的梯度的较差的近似值，有助于收敛并防止优化过程在过程中陷入局部极小。在SGD中，每个epoch开始时都要对数据集进行重新排序，有助于确保在小batch上估计的梯度序列代表整个数据中计算的梯度，通过重新排序我们在梯度下降中有效地引入了随机性，减少了停止于局部最优的结果。但也导致我们最好使用较少的学习率。

通常，batchsize是一个恒定的大小，我们需要在训练之前设置，就像学习率一样。这些被称为超参数，以区别于模型的参数。DataLoader的作用是从数据集中小批量进行采样，从而使我们能够灵活地选择不同的采样策略。一种非常常见的策略是每个epoch都进行打乱顺序后进行统一采样。图7.14显示了data loader打乱了它从数据集获得的索引。

图7.14显示了DATASET和DATALOADER是如何将数据集中的数据打乱然后放入一个个batch中的。

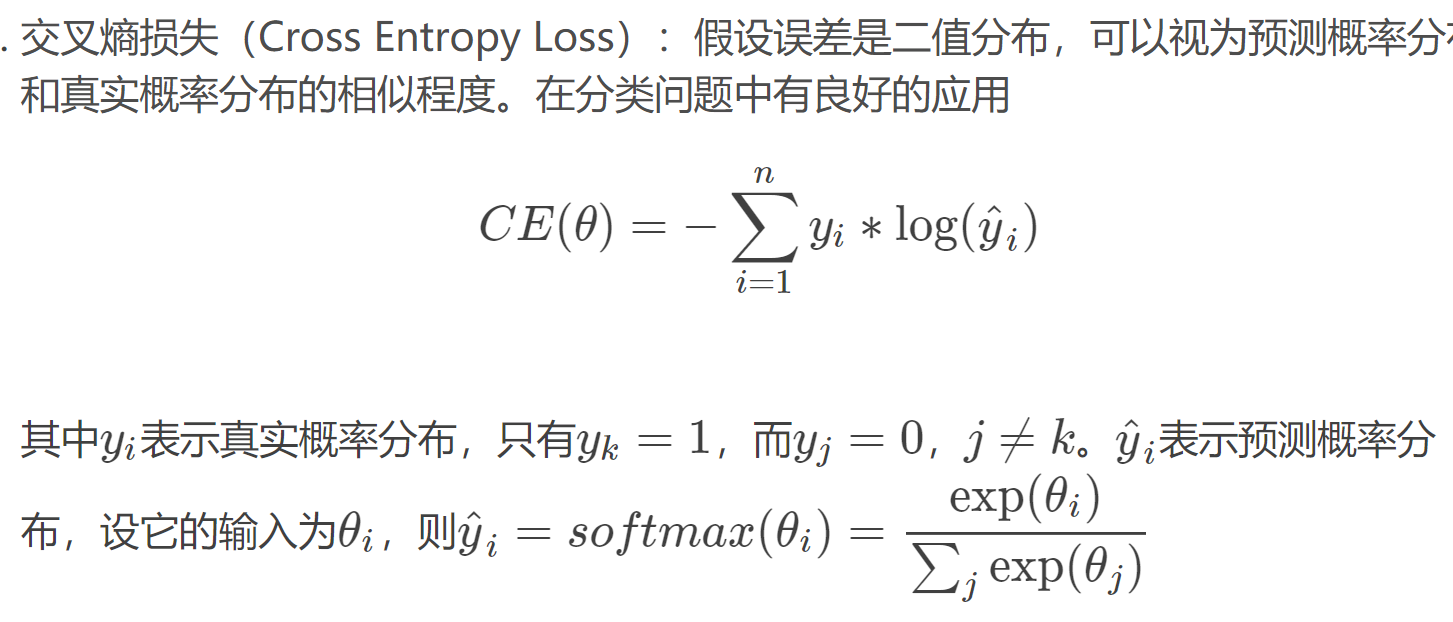
让我们看看这是怎么完成的。至少。DataLoader需要一个数据集batch的大小和一个shuffle的布尔值来表示在每次epoch开始时是否需要对数据进行打乱。

DATALoader是一个可以在循环中使用的类，所以我们可以在新的训练代码的内存循环中直接使用他。

虽然模型很简单层数很少，数据集也较小。但是这两个类中的许多样本可能存在系统性差异（例如背景颜色），这有助于模型基于几个像素区分鸟类和飞机。所以效果还可以。Accuracy: 0.794000

P24

Logsoftmax加nllloss 这两项与交叉熵是等价的。在信息学中的交叉熵根据样本大小进行标准化，这种交叉熵可以解释为目标分布下预测分布的负对数似然。和深度学习中的交叉熵有一定区别。在文献中不要混淆。



训练模型是可以去掉softmax层，将NLLLOSS替换为交叉熵。这更加的方便。

只需一次就可以完成所有操作，唯一的问题是模型的输出不能解释为概率（或对数概率）。我们需要使用softmax来显式地获取输出的概率。

接下来我们修改了模型的结构 ，我们试图将特征的数量逐渐减少到输出端，希望中间层能够在越来越短的中间输出中更好地压缩信息

在训练和在测试集上评估得到了更好的准确率0.802，但效果的提升并不是很大，在训练集上模型的准确率十分完美0.9981。这告诉我们模型发生了过拟合。我们的全连接模型通过记忆训练集来在训练集上识别鸟和飞机，但是在测试集的表现就没有那么好了，尽管我们使用了一个更大的模型。

Pytoch中可以是欧阳那个nn.model模块的parameters()方法来确定一个模型有多少个参数，而且我们使用这个方法把参数传递给优化器，使用numel方法可以得到一个张量中有多少个元素。加起来得到总和。在我们的例子中，参数需要设置requires\_grad为true，我们想要在整个模型上得到的是可训练的参数。

上面参数个数的计算告诉我们，这种全连接的神经网络并不适合这样像素的图片处理，如果我们使用1024\*1024像素的图片作为输入值，在第一个隐藏成使用1024个参数那么我们在第一层就会有31个参数，使用32位的float存储也需要12GB的内存，甚至我们还没有设计第二层，更别说梯度计算和存储了，这种全连接的方式处理图片不适合大部分的GPU

7.2.7 全连接神经网络的限制

使用图片变成一维的张量，然后使用线性层来进行处理。输入为每一个在我们的RGB图像中的分量，并计算它与每个输出特征的值的线性组合。一方面，我们允许将图像中的任何像素与其他像素相结合

。另一方面，我们没有利用相邻或远处像素的相对位置，而是将图像视为一个大的数字向量，这极大地提高了计算量和成本。

同时，其没有平移不变性。以飞机为例子，在32×32的图像中拍摄到的一架在空中飞行的飞机与蓝色背景下的黑色十字形状非常相似。如图所示的全连接的网络7.15需要了解当像素0,1为暗时，像素1,1也是暗的，这能暗示是一架飞机。图7.16的上半部分对此进行了说明。但是，将同一架飞机移动一个或多个像素，就像图的下半部分一样，像素之间的关系必须从头开始重新学习：这时，飞机很可能在点0,2是暗的，像素1,2是暗的。也就是说，全连接的神经网络不具有平移不变性。这意味着，经过从位置0.0开始的飞机图片训练的网络，将无法识别从位置11开始的完全相同的飞机。位置一旦改变，网络无法适应。

为了解决这一问题，我们必须扩充数据集，也就是说，在训练期间对图像进行平移，这样网络就有机会看到各个位置上的飞机，并且我们需要对整个数据集上的所有图片进行平移。当然我们可以使用torchvision.transforms帮助我们完成平移。但是这中数据增强策略并不是没有代价的，他的代价是隐藏层节点数目必须相应增加，来存储平移相关的信息。

在这章的最后，我们有一个数据集，一个模型，使用训练策略训练了我们的模型。但是由于模型的结构和我们的问题并不匹配，所以相较于我们想要模型得到的一般特征，我们的模型发生了过拟合。

本章中我们创建了一个模型，允许将图像中的每个像素与其他像素关联起来，而不管它们的空间排列。不过一个更合理的假设是理论上说，离得更近的像素之间的关联性要大得多。我们模型的这种设置意味着我们训练的分类器不是平移不变的，所以我们如果我们希望在测试集上做得好，就要使用大量的计算来学习图片平移之后的副本。

解决我们当前的一系列问题的方法是改变我们的模型来使用卷积层。我们将在下一章讨论这意味着什么。

7.3结论

在本章中我们从数据集到模型到选择合适的损失函数以及不同的参数对训练的影响完成地解决了一个简单的图像分类问题。所有的问题都是使用Pytorch提供的通用工具包来解决的，接下来在你使用pytorch的过程中还会经常用到。

但是我们也发现了我们的模型的一个重大的缺陷，我们将二维的图片作为一维数据来进行处理。并且我们没有一个很好的方式来处理我们问题的平移不变性，在下一章我们将学习如何利用二维图片数据的性质来得到更好的结果。

我们可以利用我们现在学到的知识来处理一些数据，而不需要平移不变性。例如，在表格数据或我们在第4章中遇到的时间序列数据上使用它，我们可能已经可以做很多事情了。在某种程度上，我们也可以在适当表示的文本数据上使用它（例如在词袋模型中使用它）。

7.4经验

1 改变损失函数（MSE）

改变损失函数能够影响训练的结果

2能否降低模型的维度能够减少过拟合吗？

降低模型的维度会造成欠拟合

7.5总结

计算机视觉是深度学习最广泛的应用之一。

一些数据集有监督的图像数据集是公开的；其中许多可以通过torchvision导入

Datasets和DataLoaders为加载和采样数据集提供了简单而有效的抽象。

对于分类任务，在网络的输出之后加上softmax函数来生成满足解释为概率的要求的值。然后对正确类别的softmax概率，使用负对数函数处理。这是一个分类问题的理想损失函数。这个损失函数在pytorch中叫做交叉熵。

我们确实可以将图像视为像素值的向量，就像其他任何数字数据一样使用全连接的网络来处理他们。但是这样做会使利用数据中的空间关系变得更加困难。

简单的模型可以使用nn.Sequential.创建