dw2的知识准备

设计一个神经网络库

在第一次作业已经实现了tensor类和他所需要的算子。这一次需要从高层讲述一些神经网络库的核心组件。首先是module类，它可以帮助我们组装模型。里面有eval，train等方法。其中最重要的方法是parameters，他返回该层的参数。每个Module对象可以递归包含另一个Module对象作为自己的成员变量，表现模块由若干子模块递归组成。例如可以定义一个包含两条ScaleAdd路径的模块。

随后作业中定义了线性层，用Parameter类来定义权重和偏置项，这里parameter就是可训练的变量（即可更新）

归一化：假设对ReLU神经网络的权重使用分布N(0,c/n)进行初始化，其中n是输入维度，c≠2，（没有选择最正确的方案。当c=2时激活值的方差最稳定）。如前面的课程所示，至少在使用普通SGD优化器时，对MNIST数据集，50层的神经网络无法训练：c=1时梯度的ℓ2范数非常小，激活值随着层数增多变小，模型训不起来；c=3时梯度的ℓ2范数非常大，激活值随着层数增多暴增，激活值快速达到NaN。所以，需要再次提醒，初始化对神经网络非常重要——事实上，它比你想象得还要敏感：如果c取1.7或2.3，仍然会出现刚才所说的梯度太大或梯度太小现象，导致模型训练的效果有偏差。也就是说，初始化带来的影响会贯彻模型训练的始终。实验表明，神经网络训练收敛以后，收敛得到的权重与初始权重相比不会改变太多，而不同的初始化会导致激活结果在层与层之间变化很大。然而，神经网络中每个“层”几乎可以做任何计算，所以是不是可以引入一个层来稳定激活值的方差，抵消不好的初始化带来的影响？这就是归一化层提出的动机。最简单的想法是对一层输出的所有激活值归一化（均值为0，方差为1），这种方法也称为层归一化（layer normalization，LN）

文本, 信件

描述已自动生成

LN在深度学习中非常常见。目前火爆的Transformer就是使用这种方法做归一化，但是对普通全连接神经网络，则加了LN以后很难训练到一个较低的损失值。其原因是LN是要把所有样本激活值的均值和方差都进行归一化，但是这种指标本身可能是很好的特征。现在假设我们不再只考虑一条样本，而是考虑一个小批次数据Z，其每一行表示之前的一个样本：

文本

描述已自动生成

LN等价于对矩阵的行做归一化，那么自然地，另一种方法是对矩阵的列做归一化——这种方法称为批次归一化（batch normalization，BN）。这种方法保留了不同样本激活值在范数、数值、大小方面的特征，因此也得到了广泛应用。但是，这种情况下，使得样本间的数据相互依赖，实际上感觉会很奇怪。一个改进方案是在每个层为所有特征维护一个均值/方差的滑动平均值，在测试时使用这个统计特征做归一化

文本, 信件

描述已自动生成

正则化：典型的深层网络（即便是两层网络）都是过参数化（overparameterized）模型，即其包含的参数数量多于训练样本数量。在一些假设条件下，这意味着这些网络可以完全拟合数据（0训练误差），在传统机器学习或者统计学上称为模型过拟合，即其泛化能力比较弱。正则化和归一化在某种程度上有相似之处，都是在层上加一些功能来帮助有效优化模型或者提升模型效果。典型的深层网络（即便是两层网络）都是过参数化（overparameterized）模型，即其包含的参数数量多于训练样本数量。在一些假设条件下，这意味着这些网络可以完全拟合数据（0训练误差），在传统机器学习或者统计学上称为模型过拟合，即其泛化能力比较弱。

为了抵消这种情况，统计中的常见方法是加入一些正则化。不正式地讲，正则化限制了函数类别的复杂度来确保模型可以更好地泛化。深度学习中有两类正则化手段

* 隐式正则化：使用模型架构或算法本身的性质，并没有有意控制模型复杂度。使用某个权重初始化网络，并使用SGD优化就是一种隐式正则化手段，因为此时我们没有在“所有神经网络”上进行优化，而是对能用SGD优化的网络进行优化。由于神经网络训练后权重离初始点变化不大，我们也没有搜索整个参数空间，而是只在初始化权重的一个邻接范围内寻找解
* 显式正则化：则是带着正则化网络的目的有意修改网络架构和训练过程

对网络权重进行正则化的最常见手段是\*\*ℓ2\*\*正则化，也称为权重衰减。经典情况下，模型参数的范数大小通常是模型复杂度的一个合理表现（一种非正式的解释是，当权重很小时，函数必须特别平滑才能拟合数据），所以可以优化网络的同时，让参数尽量小。常见方法是在损失函数上加一个正则项，通常是ℓ2正则项，表现形式如下

徽标

描述已自动生成

这才是机器学习更合适的优化目标，因为实际使用时通常都会对权重进行限制。对于上述优化目标，对应的梯度下降更新规则为

文本

描述已自动生成

由于α和λ都是正数，因此1−αλ也是正数，即每次更新时Wi都乘上了一个小于1的系数，缩小了，所以称为权重衰减算法。具体实现时，惯例是在优化器中实现这一正则化方法，而非在损失函数中实现。然而，在深度学习中，网络参数权重的大小和模型复杂度的关系并不明确，存在权重大小不同，但是训练收敛后损失近似的情况，所以，权重真的需要做衰减吗？这实际上是一个见仁见智的问题

另一种要在这里提及的正则化手段称为dropout。前面提到的ℓ2正则化是明确地对网络权重做正则化，dropout则是对激活值做正则化：在训练的过程中，随机将网络的一部分激活值置为0。

注意为了保证方差的稳定性这里要除以一个1−p做放大。Dropout通常只用在训练过程，在测试时不使用。Dropout看上去也是一种奇怪的解决方案，它会极大修改拟合出的函数的行为。一些人认为其背后的思想是让网络在缺少某些激活值的时候也变得鲁棒，但是由于模型在推理时不需要使用dropout，所以这个解释有些勉强。**本课认为，dropout是一种随机逼近，就像SGD是随机逼近整个数据集的梯度**

截止到目前，已经讲了很多种使神经网络的优化变得容易的方法，包括不同学习率的选择、是否使用动量、权重初始化选择策略、归一化、正则化等等。初学者很容易考虑使用网格搜索来找出最优组合，但这对深度学习是不可行的，因为整体代价太高了。以BN举例，在原始论文中，作者认为BN奏效的原因是减少了内部协变量偏移（internal covariate shift），认为BN和正则化之间存在关系。然而这种方法奏效的真实原因一直存在争议：NeurIPS 2018的文章How does BN help Optimization认为实际上BN让优化表面更光滑，另一篇文章Gradient Descent on Neural Networks Typically Occurs at the Edge of Stability则否认这个观点，认为表面的光滑不很重要。这个例子可以看出，尽管有个技术大家都在用，但是其原理仍然不清楚。另外，近期工作表明，BN对解决分布偏移问题能有较大帮助，可以说这种古老、简单的方法又有新的领域能够焕发第二春——这也是深度学习的神奇所在。

硬件加速，GPU编程

为了让程序能够运行，还要编写对应的宿主代码（CPU端侧代码）来运行核。具体地，在CPU端要新建三个数组，然后进行CUDA allocate在显存中分配空间，并将数据从内存拷贝至显存。接下来，计算为了执行程序需要多少个块，再调用GPU核（参数为GPU指针）。具体计算逻辑在GPU上执行结束以后，将结果从显存拷贝至内存，然后释放显存空间。下面代码给出了上述流程的示意（这段代码仅为示意使用。cudaMemcpy操作在内存和显存之间搬运数据比较耗时，会成为性能瓶颈，因此实际开发中会让数据在显存中呆尽可能长的时间——所以写torch程序时，尽量不要频繁调用.numpy()操作）