Week3：实现全连接神经网络

一、实验目的

构造一个三层的全连接神经网络（即包含两个隐含层），在 MNIST 数据集上完成手写 0-9 数字的识别。本次实现的网络不借助 pytorch 框架的函数，需要大家在已有代码（文件夹内）的基础上，自己实现三层全连接神经网络。实验借此回顾理论课中关于全连接神经网络、损失函数、优化器、反向传播等内容，同时使大家对 pytorch 框架的部分基础实现与模型训练过程有初步理解，便于之后直接用框架训练模型(可在完成实验后阅读文件夹内的 html 文件了解更详细内容)。

二、实验内容

1. 使用 tensor 进行运算，实现网络的前向传播和反向传播（不使用 torch.nn 搭建网络，不使用 backward 方法进行反向传播），完成三层全连接神经网络在 MNIST 数据集上的训练和测试。
2. 损失函数方面，需要实现交叉熵损失函数（不使用 torch.nn.CrossEntropyLoss）；优化器方面，需要实现带动量的 SGD 优化器（不使用 torch.optim.SGD）。
3. 请提交一份简短的实验报告，说明神经网络的实现过程以及模型在数据集上的表现。代码应有适量注释，并与报告一起提交。

说明：

（1） 需要设计部分网络的结构，比如两层隐含层的神经元数，激活函数等；

（2） 全连接层的参数初始化无需自己实现，可直接调用函数；

（3） 对类的设计没有具体要求，在代码注释或报告中简要说明即可；

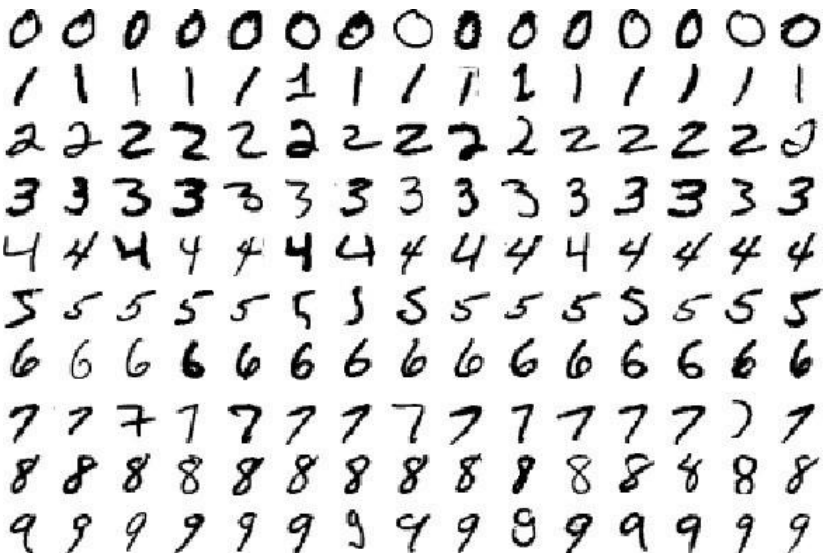
三、相关介绍

这一部分将简要介绍本次实验的相关内容，包括 MNIST 数据集、训练流程简介、损失函数、全连接神经网络、优化器五个方面。

1、MNIST 数据集

MNIST 是图像分类任务常见的数据集。它包含了 0-9 共十个类别的手写数字，总共有 70000 张图片，其中训练集有 60000 张图片，测试集有 10000 张图

片，图像分辨率均为 28×28。MNIST 数据集的部分图片如下图所示。



2、 训练流程简介

开始训练前，需要完成网络结构的搭建、超参数的设置（如学习率、迭代次数等）、数据预处理、损失函数和优化器的设置等。

训练开始时，我们会根据设置好的迭代次数，让网络在数据集上多次训练。对于每一次迭代，我们通常采取 mini-batch gradient descent 的方式，将训练集划分为多个 mini-batch，网络根据每个 mini-batch 进行参数更新。网络训练的流程可以大致表示为：

i. 当 迭代次数未完时 循环

a) 当 没有遍历数据集所有 mini-batch 时 循环

1) 网络输入当前批次的数据，得到输出

2) 通过损失函数计算输出与目标之间的损失

3) 反向传播得到网络梯度

4) 网络根据梯度进行一次更新

5) 其他操作，如一些信息的输出、数据保存等

上面是对流程的大致概括，后续实验会对网络的训练流程进行更加详细的介绍，大家现在只需要对流程有大致了解即可。这些步骤都可以通过 pytorch 框架较为方便地进行实现，本次实验的内容就是不调用已有函数，完成最内层循环的 1）3）（实验内容 1）与 2）4）（实验内容 2）。

3、 损失函数

损失函数用于评估网络的输出与目标之间的差距，损失越小说明网络输出更好地接近目标。常用的损失函数有交叉熵损失函数、均方误差损失函数等。其中，交叉熵损失函数是分类任务中常见的损失函数，也是本实验中需要大家实现的内容。它的表达式为（没设置类别权重时）：

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

Pytorch 实现中，CrossEntropyLoss 的实现是 LogSoftmax 与 NLLLoss 的结合，实现时可以据此完成交叉熵损失函数。相关内容可参考 https://blog.csdn.net/qq_22210253/article/details/85229988。

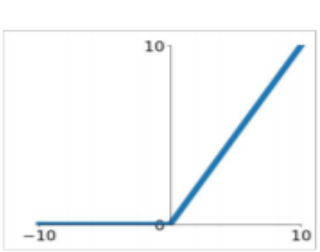
4、 全连接神经网络

这一部分的内容将从三个方面进行介绍，分别是网络结构、前向传播以及反向传播。

（1） 网络结构

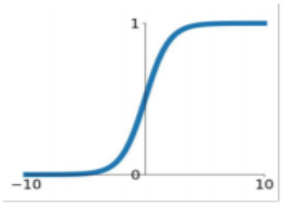
网络的结构为三层全连接神经网络。对于本次任务，网络首层与末层的神经元个数都已确定。对于首层，由于图片分辨率为 28*28，所以神经元个数应为 784；对于末层，由于任务是十分类任务，所以神经元个数应为 10 个。中间两个隐含层的神经元个数大家可以自由设置（如 256 与 128、128 与 128 等）。

另一个选择是激活函数的种类。常用的激活函数有 relu、sigmoid、tanh 等，不同的激活函数可能需要参数初始化时不同的参数初始化。



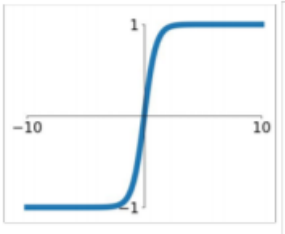
ReLU

$$\max(0, z)$$



Sigmoid

$$g(z) = \frac{1}{1+e^{-z}}$$



tanh

$$\tanh(z)$$

如果选择 relu 作为激活函数，可调用 torch.nn.init 中的 kaiming_uniform_对网络的权重进行初始化（这一部分内容可以参考 torch.nn.Linear 的实现）。如果选择 sigmoid 或 tanh 作为激活函数，可调用 xavier_uniform_进行初始化。相关内容可参考 https://blog.csdn.net/weixin_42147780/article/details/103238195

补充：在 torch.nn.init 中有常用的网络参数初始化方法。

（2） 前向传播

对于某一个全连接层，假设它的权重为 w，且有偏置 b。那么对于给定的输入 x，激活函数 g，该层的输出 y 为 y=g（wx+b）。

本实验的网络由三层全连接层构成，在前向传播时只需将上一层的输出当成本层的输入，根据本层的参数继续计算即可。由于使用 mini-batch 的策略，假设每一批次的图片数量为 bs，那么初始输入就是维度为[bs, 784]的 tensor（代码中已将 1*28*28 维度展开为 784 维），最后经过网络的前向传播，输出维度为[bs, 10]的 tensor。

（3） 反向传播

前向传播得到输出后，根据损失函数计算输出与目标的损失，同时也获得了梯度。梯度从损失函数开始，由网络的最后一层向前传递。每一层需要计算当前层的梯度（之后更新参数需要使用），并将梯度提供给前一层，用于前一层的梯度计算。

5、优化器

优化器可根据梯度信息对网络参数进行更新。本次需要实现的带动量的 SGD 是常用的优化器，动量的引入在一定程度上加速了参数更新过程。带动量的 SGD 的参数更新形式为：

$$\begin{aligned} \boldsymbol{v}_{t+1} &= \gamma \boldsymbol{v}_t + \eta \nabla L(\boldsymbol{\theta}_t) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \boldsymbol{v}_{t+1} \end{aligned}$$

其中，γ 为动量（通常取 0.9），η 为学习率。这部分内容可以参考 https://blog.csdn.net/tsyccnh/article/details/76270707。