# 实验报告

**在 Fashion-MNIST 数据集上复现神经网络的学习流程**

学号：1951510　　姓名：姜文渊

Tongji University

更新：2022 年 4 月 13 日

本次实验的代码托管在 GitHub 上，见`https://github.com/jwyjohn/IntroAI_Assignment02`，故具体代码不再附在文中。

## 1　实验背景

**实验目的**　前几次课上介绍了神经网络训练的详细的数学原理和常用深度学习框架的使用方法，本次实验企图在老师推荐的 Fashion-MNIST 数据集上，实操复现神经网络的学习流程，以期熟悉训练神经网络的大体流程，并加深对深度学习的理解。

**实验概览**　本文大致分为以下几个部分：

1. 实验原理的简介
2. 实验环境配置
3. 数据集的准备
4. 全连接神经网络结构的搭建
5. 训练与测试
6. 性能比较与分析
7. 实验小结

## 2　实验原理

实验原理部分主要列举全连接神经网络的前向传播和反向传播的矩阵形式的公式，而关于全连接神经网络（多层感知机）的简介，以及公式和原理的推导，在课程中已经较为完备，这里就不再赘述了。

**前向传播过程**

$$\begin{cases} \boldsymbol{z}^{(l)} = \boldsymbol{w}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b} \\ \boldsymbol{a}^{(l)} = \sigma(\boldsymbol{z}^{(l)}) \end{cases} \tag{1}$$

**计算输出层误差**

$$\boldsymbol{\delta}^{(L)} = \nabla_{\boldsymbol{a}^{(L)}} \boldsymbol{C}(\boldsymbol{\theta}) \odot \boldsymbol{\sigma}'(\boldsymbol{z}^{(L)}) \tag{2}$$

**计算反向传播误差**

$$\boldsymbol{\delta}^{(l)} = \left((\boldsymbol{w}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}\right) \odot \sigma'(\boldsymbol{z}^{(l)}) \tag{3}$$

**计算并且更新权重**

$$\begin{cases} w_{jk}^{(l)} := w_{jk}^{(l)} - \alpha \dfrac{\partial C(\theta)}{\partial w_{jk}^{(l)}} \\ b_{j}^{(l)} := b_{j}^{(l)} - \alpha \dfrac{\partial C(\theta)}{\partial b_{j}^{(l)}} \end{cases} \tag{4}$$

# 3 实验环境

**硬件环境** 本次实验使用的是 x86 兼容机。具体配置如下：

- CPU: i7-11800H 8C16T @2.40GHz, Max TDP 35W
- RAM: 8Gx2 DDR4 2400MHz with XMP
- GPU: RTX 3060 Laptop, Max TDP 65W

**深度学习框架** 本次实验使用的深度学习框架为PyTorch，与课上使用的Keras不同。Pytorch是由Facebook的人工智能研究小组开发的深度学习框架。与 Keras 一样，它也抽象出了深层网络编程的许多混乱部分。就高级和低级代码风格而言，Pytorch 介于 Keras 和 TensorFlow 之间。比起 Keras 具有更大的灵活性和控制能力，但同时又不必进行任何复杂的声明式编程。[1]

选择使用 PyTorch 的原因除了其被广泛应用于深度学习相关的研究中，更多的考量在于，框架本身只是算法的载体，通过使用于课上不同的框架实现相同的神经网络的训练于测试流程，可以更好地熟悉深度学习的原理以及现代深度学习的框架。

**集成开发环境** 为方便 PyTorch 的实时调试，笔者采用的开发环境为 Jupyter Notebook。事实上，任何 Python 的开发环境都是合理的。

# 4 实验过程

## 4.1 环境配置

由于环境配置不是本次实验的重点，故这里只大致说明环境配置的流程，具体笔者使用的环境详见开头处笔者的 GitHub 库中的`requirements.txt`。

1. 安装**非最新版本的**显卡驱动
2. 从 TUNA 镜像站下载 Miniconda 安装包并安装
3. 配置 conda 的镜像站和 pypi 的镜像站
4. 使用 conda 创建 PyTorch GPU 的环境，并安装对应的软件包

## 4.2 数据集准备

本次实验使用的数据集为 Fashion-MNIST[2]，其除了内容与经典的 MNIST 数据集不同外，其它参数都保持与 MNIST 数据集相同。例如，图像均为`28x28`的`8bit`灰度图，分类均为 10 类。

**数据集下载** 将`https://github.com/zalandoresearch/fashion-mnist/tree/master/data/fashion`中到本地，其中的四个`*.gz`文件解压后即为数据集。值得注意的是，在国内使用 PyTorch 中的`torch.utils.data.DataLoader`和`torchvision.datasets.FashionMNIST`加载该数据集时，PyTorch 会自动从网上下载这些文件。因为国内互联网的种种特性，下载大多数时候会失败，此时终止该处程序的执行，将前面下载好的四个`*.gz`拷贝到四个`raw`目录下，再次运行时即可成功载入数据集。

**此外，还可以在载入数据时对数据集进行 Normalization 和数据集增广，从而使得后面的训练获得更好的效果。**

**数据集划分** 一般来说，训练神经网络需要自行划分数据集，但 Fashion-MNIST 已经将训练集和测试集划分，故无需我们自己动手划分了。

### 4.3 神经网络结构搭建

和 Keras 类似，在 PyTorch 中，我们可以通过继承`nn.Module`类，将`self.model`设为一个拥有多层的`nn.Sequential`模型，并保证每一层的输入输出尺寸相匹配，即可完成模型的创建。例如创建一个多层的使用`LeakyReLU`作为激活函数的模型，可由下面十几行代码完成构建。

```
class fcNet(nn.Module):
    def __init__(self):
        super(fcNet, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(784, 200),
            nn.LeakyReLU(inplace=True),
            nn.Linear(200, 200),
            nn.LeakyReLU(inplace=True),
            nn.Linear(200, 10),
            nn.Sigmoid(),
            nn.LogSoftmax()
        )
    def forward(self, x):
        x = self.model(x)
        return x
```

### 4.4 训练与测试

**参数设置** 得益于 Python 灵活的语法和 Jupyter 灵活的调试方法，我们的一些超参数，例如`batch_size`，`learning_rate`，`momentum`等，可以根据训练的效果随时调整。

**训练过程** 训练前需要先指定优化器，例如`SGD`、`Adam`等。由于训练是在 GPU 上进行，故而需要将模型和数据利用`.to(device)`方法存放到显存中进行训练。受限于 RTX3060 Laptop 的显存大小，模型的规模和`batch_size`都不能过大。长时间挂机训练时，注意兼容机的散热和供电，并需要配置好操作系统使之不会进入休眠状态。为了防止训练意外停止，每一个 epoch 之后，训练后的网络都会进行一次保存，且训练的 loss 和 accuracy 都会记录在 log 中备查。

```
def train(epoch):
    network.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        data = data.view(-1, 28*28)
        output = network(data.to(device))
        loss = F.nll_loss(output, target.to(device)) # 计算loss
        loss.backward() # 反向传播
        optimizer.step() # 更新权值
        ..... # logging与保存模型
```

## 5 性能分析与比较

我们在 Fashion-MNIST 的任务中，主要关注的有以下几点指标：1）模型的准确率；2）训练模型所花费的 epoch 数；3）模型的规模。

除了训练的全连接神经网络外，笔者也尝试了其它的一些网络结构，例如各类卷积神经网络等。相比于全连接神经网络，卷积神经网络在测试集上达到一定 accuracy 所需的 epoch 更少，且其参数也远小于相似性能的全连接神经网络。

此外，通过查阅 Fashion-MNIST 相关的内容可知，引入一些类似 dropout 和 batch normalization 的技巧[3]，对数据集进行随机翻转、旋转等数据集增广操作，也可以提高神经网络的准确率，且对训练所需的 epoch 数量影响不大。[4]

# 6 实验总结

通过本次实验，笔者对课上讲授的神经网络的训练过程进行了实操，除了加深了理解外，对于实操中需要注意的细节也具有了深刻的认识，下面列举一些遇到的印象深刻的“坑”：

1. 使用`torchvision.datasets.FashionMNIST`下载数据集太慢，故改用先下载后替换的方式“骗”过 PyTorch，从而成功加载数据集。
2. 输入数据和第一层的 shape 不匹配，需将输入数据 reshape 后传入神经网络。
3. 两层间的 shape 不匹配。
4. 激活函数和 Loss function 选择不当，导致训练几个 epoch 后数值变为 inf 或 nan。
5. 过深的全连接神经网络因为 fp32 的精度问题导致梯度消失，从而无法改进 accuracy。
6. 训练的模型或者 batch 过大，导致显存不足而报错。

通过在实践中查阅各类资料，亲自踩坑并解决，才真正理解理论和实践之间的鸿沟，并且验证了课上讲过的理论内容的正确性。

此外，通过对不同的网络结构进行各类性能的比较，一方面笔者了解到 Fashion-MNIST 数据集的挑战之大，另一方面也说明了目前的深度学习领域仍有较大的研究空间。此外，一些模型虽然准确率表现平庸，但是凭借其极少的参数量，或者是极小的推理所需算力，也被认为是优秀的神经网络结构，且有其在边缘计算设备上应用的价值；这也说明了对于深度学习作为一门有实践意义的具体科学，其研究不仅会追求准确率，一些其它因素也是值得我们考量的。

# 参考文献


[1] 人工智能遇见磐创. 深度学习框架 Keras 与 Pytorch 对比[EB/OL]. 2022. https://www.jianshu.com/p/f0b36e82df9c.

[2] XIAO H, RASUL K, VOLLGRAF R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms[A]. 2017.

[3] ZALANDORESEARCH. Fashion-mnist benchmark[EB/OL]. 2022. https://github.com/zalandoresearch/fashion-mnist#:~:text=Rust-,Benchmark,-We%20built%20an.

[4] MESHKINI K, PLATOS J, GHASSEMAIN H. An analysis of convolutional neural network for fashion images classification (fashion-mnist)[C]//International Conference on Intelligent Information Technologies for Industry. Springer, 2019: 85-95.


# 互评

**互评人：1950787 杨鑫 给分：100% （8/8）** 短评：

该该同学详细的汇报了在 Fashion-MNIST 数据集上复现神经网络的学习流程的过程，分别从实验原理、环境配置、数据集的准备、神经网络结构的搭建、训练与测试、性能比较以及结果分析等方面对整个实验流程做了详尽的分析和总结。体现了神经网络训练的核心过程，实验流程阐释清晰，实验总结也十分完善，我认为是一篇非常优秀的报告。