华中科技大学

计算机科学与技术学院

《计算机视觉导论》实验报告

基于前馈神经网络的分类任务设计

专　　业:	计算机科学与技术（图灵班）
班　　级:	图灵 2301 班
学　　号:	U202314607
姓　　名:	向恩泽
成　　绩:	
指导教师:	刘　康

完成日期：2025 年 10 月 20 日

目录

I: 任务要求

设计一个前馈神经网络，对一组数据实现分类任务。

下载 dataset.csv 数据集，其中包含四类二维高斯数据和它们的标签。设计至少含有一层隐藏层的前馈神经网络来预测二维高斯样本 $(\mathrm{data}_1, \mathrm{data}_2)$ 所属的分类 label. 这个数据集需要先进行随机排序，然后选取 90% 用于训练，剩下的 10% 用于测试。

II: 环境介绍

本次实验使用 Python 3.10 版本进行开发，并使用 Pytorch 2.8 + cu128 作为深度学习框架。具体环境信息如下：

环境名	具体信息
CPU	AMD Ryzen 9 9955HX3D 16-Core Processor*
GPU	NVIDIA GeForce RTX 5070ti Laptop 12G
DRAM	32 GB
操作系统	Windows 11 + WSL2 Ubuntu 22.04 LTS
Python 版本	Python 3.10.12
Pytorch 版本	Pytorch 2.8 + cu128

* AMD Ryzen 9 9955HX3D 默认包含两个 CCD（CCD0 和 CCD1），每个 CCD 提供 8 核心（16 线程），总计 16 核心（32 线程）。由于环境设置，CCD1 已被禁用，仅 CCD0 在工作。CCD0 几乎等效于 Ryzen 7 9800X3D，提供 8 核心（16 线程）和 96 MB 的 3D V-Cache。

III: 数据集简介与预处理

观察 dataset.csv 可知，点被分为了四类（label 分别为 1,2,3,4），每类 1000 个点，总共 N = 4000。每样本为二维向量 $(\mathrm{data}_1, \mathrm{data}_2)$.

首先，需要对数据进行分割。考虑需要从 .csv 文件中加载数据，为了代码的模块化和可读性，我的代码中实现了一个 class CSVLoader 类来完成从 dataset.csv 中加载数据并进行训练集、测试集分割的任务。打乱时直接使用 numpy 提供的 numpy.random.permutation 即可。

并且，为了训练方便，需要对 label 进行变形，如果直接使用原 label 的话，label 的 shape 为 (N,)，而对分类任务，表现较好的交叉熵损失函数 `nn.CrossEntropyLoss()` 需要的 label shape 为 (N, C)，其中 C 为类别数目，因此需要将 label 变形为 (N, C) 形式。该变形特别简单，只需要将 label 变为 one-hot 形式即可（将 label 进行独热编码）：

```
# import torch.functional as F
def to_onehot(label: torch.Tensor) -> torch.Tensor:
    label = label.to(dtype=torch.int64) - 1 # [1, 4] -> [0, 3]
    label = F.one_hot(label)
    label = label.to(dtype=torch.float32)
    return label
```

该函数只需要将形如 (N,) 的 label 传入，即可返回形如 (N, C) 的 one-hot 形式 label.

最后，对数据集按比例进行 train/validate 划分即可。需要注意的是，为了避免随机化情况下导致每种 label 在训练集和测试集中的分布不均匀，我在划分数据集时，先对每个 label 的数据分别进行划分，然后再将各个 label 的训练集和测试集合并，最终得到完整的训练集和测试集。具体细节请见 `CSVLoader` 类的代码。

由于标准化处理并不是 Loader 的逻辑，因此我将其单独放在了模型训练代码中进行处理。标准化处理的代码如下，该代码只对 feature 进行标准化处理（均值为 0，标准差为 1）：

```
    def normalize(feature: torch.Tensor):
        # used for normalize feature data
        mean = feature.mean(dim=0, keepdim=True)
        std = feature.std(dim=0, keepdim=True)
        feature = (feature - mean) / std
        return feature
```

IV: 模型设计

我设计的网络结构是十分经典的 Embedding + MLP Head 结构，接下来分为 Embedding, MLP 和 Head 三个部分对我的模型进行介绍。

该模型的具体代码部分在 main.py 中的 class Network 中，可以在代码中找到具体实现。整体使用该类实现网络时，传入参数为：

```
# build network
net = Network(
    in_dim=2,
    embed_dim=8,
    hidden_params=hidden_params,
    num_classes=4,
    device=device
)
```

这样即可获得一个完整的前馈神经网络模型 net 没后续只需要对 net 进行训练和测试即可。由于本次实验是一个分类任务，所以默认选择交叉熵损失函数 nn.CrossEntropyLoss() 作为损失函数，后续将在启动脚本中介绍损失函数设置。

IV.1: Embedding Layer

由于数据 feature 是十分简单的二维向量，因此我设计的 Embedding 层也十分简单，即一个全连接层 nn.Linear()，其中输入维度为 2，输出的嵌入维度为 8，即 nn.Linear(2, 8)。该层的作用是将二维向量映射到一个更高维度的空间中，便于后续的非线性变换和分类。

IV.2: MLP Layer (FCN)

考虑到数据集较为简单，且数据量不大，因此模型设计上不需要特别多的参数（过多的参数，一是本地计算资源有限，参数过多跑不动，二是参数过多容易过拟合，~~三是确实没这个必要~~），因此，实现代码时在代码中内置了三种 FCN 模型，根据参数规模分别区分为 tiny, normal 和 huge 三种，具体参数如下：

```
builtin_model = {
    "tiny": [16, "relu", 8, "relu"],
    "normal": [20, "tanh", 40, "leakyrelu", 20, "leakyrelu"],
    "huge": [64, "tanh", 128, "sigmoid", 384, "leakyrelu", 100, "leakyrelu", 30,
"sigmoid"]
}
```

当然，代码保留有自行设置参数的功能，只需要在启动时传入 `--structure` 参数即可，这一部分将会在启动脚本中详细介绍。

IV.3: Head Layer

模型本质作为一个四分类器，因此我对 Head 层的要求为将 MLP 输出的四维向量转化为对应类别的概率，故而我选择了一个十分简单的 Softmax 层作为 Head 层，即 `nn.Softmax(dim=1)`.

V: 实验分析

V.1: 数据分析

注意：由于数据分析要求以 mini-batch 为单位进行，因此后续分析“一次迭代”均指一个 mini-batch 的训练过程。

由于实验数据集较为简单，且数据量不大，因此超参数设置 epoch 为 2，batch_size 为 32，学习率 learning_rate 初始为 10^{-3}，使用 tensorboard 进行训练时的 Accuracy 和 Loss 监控可视化，每个 epoch 在 batch_size=32 时有 113 次迭代，两个 epoch 共 226 次迭代，对内置的三个模型 huge, normal 和 tiny 分别进行训练，然后获得如下 Tensorboard 可视化结果：

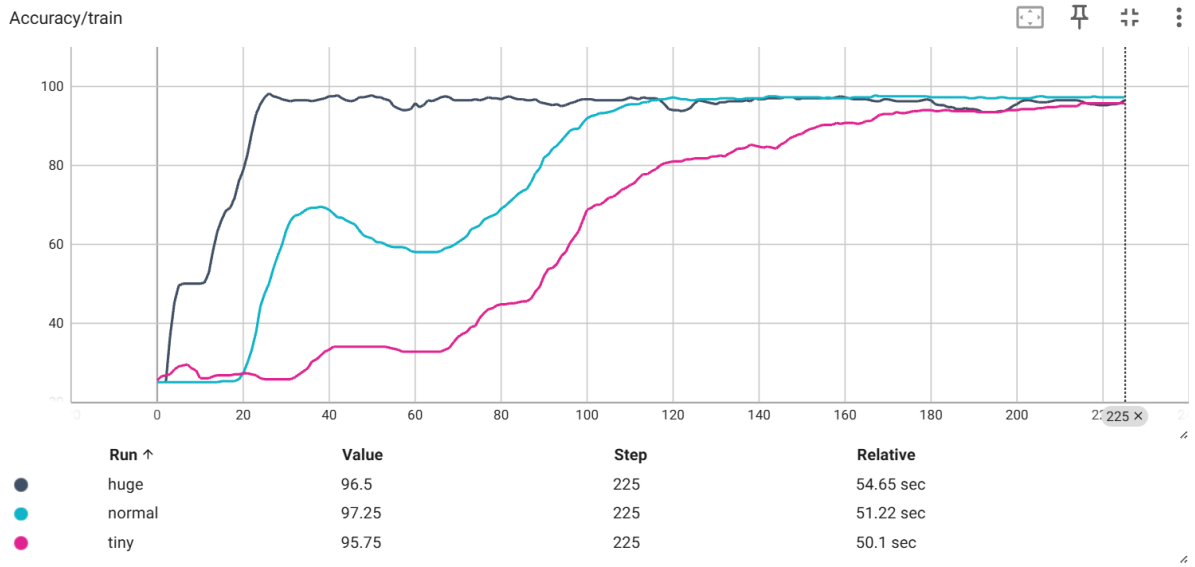

图 1　Accuracy/mini-batch Iteration 变化图（smooth=0）

可以观察到，huge 模型 Accuracy 在训练过程中提升最快，经具体数据观察，在第 26 次迭代时，huge 模型就达到了 98.25% 的准确率，而同一迭代训练次数的 normal 模型仅

达到了 50%，tiny 模型更是仅有 25.75%（约等于乱猜）。说明模型规模扩大，模型的表达能力和学习能力越强，在相同训练次数下，模型的性能提升越快。

但是，模型扩大并不意味着全是正收益，注意 图 1 中，大约在 Step=122 和 Step=195 的位置 huge 模型的准确率出现明显下降，即 huge 模型出现过拟合情况，模型在训练集上学习得过于充分，导致其泛化能力下降，从而在后续的训练中出现性能下降的现象，甚至比起同一迭代次数的 normal 模型更差。

Loss/mini-batch Iteration 变化图其实与 Accuracy/mini-batch Iteration 变化图呈现出类似的趋势：

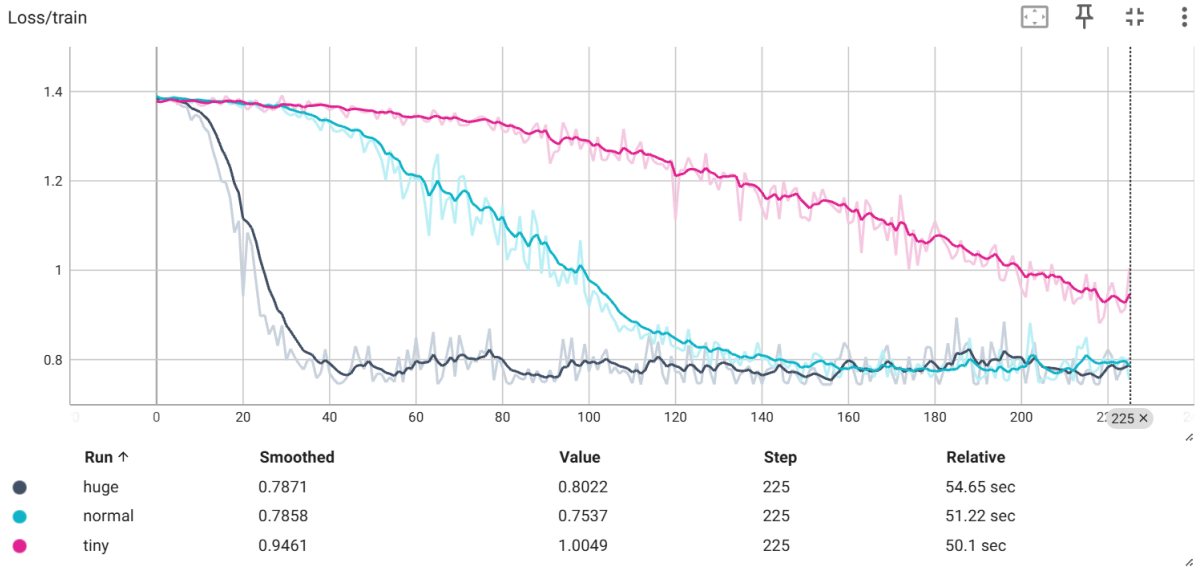

图 2　Loss/mini-batch Iteration 变化图（smooth=0.75）

Loss 的变化与上述 Accuracy 变化趋势类似，huge 模型在训练初期 Loss 降低最快，但在训练后期出现震荡，说明模型出现过拟合现象，而 normal 和 tiny 模型则表现出较为平稳的 Loss 下降趋势。但由于 tiny 模型较小，模型表达能力有限，Loss 下降速率较慢，因此在给定的较少迭代次数内，其 Loss 并未达到 normal/huge 模型的水平。

V.2: 实验结论

最明显的结论即模型并非越大越好，过大的模型容易出现过拟合，但另一方面，模型过小也会因为表达能力不足而导致性能不佳。因此，在实际应用中，需要根据具体任务和数据集的复杂度，选择合适规模的模型，以达到最佳的性能表现。

当然，过拟合也可以得到缓解，但是因为数据集十分简单，因此代码实现中没有采用 L2 正则化或者 Dropout 等方法。一些后续工作可以围绕添加 Dropout 与正则化限制模型复杂度展开，从而进一步提升模型的泛化能力。

同时，训练模型时的一些监控手段也是必要的，通过观察模型 Loss 与 Accuracy 的变化趋势，可以及时发现模型过拟合等问题，从而采取例如早停、动态学习率等措施进行调整，减少震荡，提升模型性能。

VI: 启动 / Startup

VI.1: 依赖安装

代码所需依赖保存在 `./requirements.txt` 文件中，可以通过以下命令进行安装：

```
pip install -r ./requirements.txt
```

本实验使用 python venv 虚拟环境进行环境隔离与开发，建议在虚拟环境中安装依赖。

VI.2: 启动脚本

实验代码的启动通过命令行参数进行配置，主要参数包括：

- `--csv_path`：数据集路径，默认为 `./dataset.csv`；
- `--num_epoch`：训练轮数；
- `--batch_size`：每个 mini-batch 的样本数量，默认为 32；
- `--learning_rate`：学习率，默认为 0.001；
- `--criterion`：损失函数类型，支持 `mse` 和 `crossentropy`，默认为 `crossentropy`；
- `--structure`：前馈神经网络的结构，可以选择内置的 `tiny`, `normal`, `huge`，或者自定义结构，例如 `--structure="32,relu,32,relu"`；
- `--log_dir`：Tensorboard 日志保存目录，默认为 `./logs/default`。

本地启动脚本 `launch.sh` 示例：

```
python main.py \
    --csv_path="./dataset.csv" \
    --batch_size=32 \
    --learning_rate=1e-3 \
    --num_epoch=2 \
```

```
    --criterion="crossentropy" \
    --log_dir=$LOG_DIR \
    --structure="normal"
```

然后运行 `./launch.sh` 即可一键开始训练。

VI.3: Tensorboard 可视化

训练代码使用 Tensorboard 进行训练过程的可视化监控。

如果需要使用 Tensorboard 进行可视化，可以在命令行中运行以下命令：

```
tensorboard --logdir=/path/to/tensorboard/logs --port=6006
```

然后在浏览器中打开 `http://localhost:6006` 即可查看训练过程中的各种指标变化情况，例如 Loss/Train 和 Accuracy/Train 等。

VII: 实验代码与数据

实验代码、数据集、训练数据和实验报告均可以在以下 GitHub 仓库中找到：Arextre/CVLabs/lab1.

具体的实验数据通过 Tensorboard 保存在 `./logs` 目录下，如果需要验证检查，可以使用

```
tensorboard --logdir=./logs --port=6006
```

然后在浏览器中打开 `http://localhost:6006` 鼠标选中面板中的曲线查看具体每一轮的 Acc 和 Loss.

附录：提交文件说明

提交的文件结构如下：

```
向恩泽-U202314607-实验报告一
├── dataset.csv
├── launch.sh           # 启动脚本实例
├── logs                # log 文件保存目录
│   ├── huge
│   │   └── ......
│   ├── normal
│   │   └── ......
│   └── tiny
│   │   └── ......
├── main.py             # 程序入口源代码
├── report              # 实验报告源代码
│   ├── img
│   │   └── ......
│   └── report.typ
├── requirements.txt    # 依赖列表
└── utils.py            # 工具函数源代码
└── 实验报告.pdf        # 实验报告 pdf 版本
```

实验所用代码位于 main.py 与 utils.py 中。实验报告源代码位于 report/report.typ 中，实验数据集位于 dataset.csv 中。实验日志文件保存在 logs/ 目录下。