HW2-1

这个代码整体上是一个基于Pytorch的音素分类任务实现。

代码的核心功能：

1. **数据加载与预处理**：

   首先从.npy 文件加载 TIMIT 语音数据集，然后把数据集划分为训练集和验证集。创建 PyTorch 的 Dataset 和 DataLoader 对象。

2. **模型架构**：

   代码构建了一个四层的全连接神经网络，其结构为：429 输入 → 1024 隐藏层 → 512 隐藏层 → 128 隐藏层 → 39 输出（对应 39 种音素类别）。采用 Sigmoid 作为激活函数。

3. **训练流程**：

   运用 Adam 优化器和交叉熵损失函数，并且训练 20 个轮次（epochs），其中每轮都会计算训练集和验证集的准确率与损失值，最后保存验证集准确率最高的模型。

4. **测试流程**：

   加载保存的最佳模型，然后对测试数据进行预测，最后将预测结果写入 CSV 文件。

这是第一次运行代码的结果：

```
saving model with acc 0.703
[013/020] Train Acc: 0.759404 Loss: 0.741234 | Val Acc: 0.700456 loss: 0.945627
[014/020] Train Acc: 0.764574 Loss: 0.723574 | Val Acc: 0.702159 loss: 0.942118
[015/020] Train Acc: 0.769472 Loss: 0.707325 | Val Acc: 0.704427 loss: 0.936154
saving model with acc 0.704
[016/020] Train Acc: 0.773688 Loss: 0.691314 | Val Acc: 0.701732 loss: 0.945713
[017/020] Train Acc: 0.778676 Loss: 0.676633 | Val Acc: 0.701586 loss: 0.953081
[018/020] Train Acc: 0.783107 Loss: 0.662425 | Val Acc: 0.699659 loss: 0.963290
[019/020] Train Acc: 0.786397 Loss: 0.649180 | Val Acc: 0.700086 loss: 0.957681
[020/020] Train Acc: 0.790644 Loss: 0.636623 | Val Acc: 0.699732 loss: 0.964269

Process finished with exit code 0
```

改batch大小为16，然后获得的结果为：

```
[008/020] Train Acc: 0.758744 Loss: 0.735630 | Val Acc: 0.703476 loss: 0.929885
[009/020] Train Acc: 0.767662 Loss: 0.706983 | Val Acc: 0.703883 loss: 0.928784
saving model with acc 0.704
[010/020] Train Acc: 0.775463 Loss: 0.680141 | Val Acc: 0.703216 loss: 0.943100
[011/020] Train Acc: 0.782889 Loss: 0.656170 | Val Acc: 0.700565 loss: 0.952324
[012/020] Train Acc: 0.790131 Loss: 0.633150 | Val Acc: 0.703094 loss: 0.963501
[013/020] Train Acc: 0.796688 Loss: 0.611609 | Val Acc: 0.699159 loss: 0.979420
[014/020] Train Acc: 0.802591 Loss: 0.591415 | Val Acc: 0.699716 loss: 0.987717
[015/020] Train Acc: 0.809046 Loss: 0.572342 | Val Acc: 0.700057 loss: 0.986309
[016/020] Train Acc: 0.814368 Loss: 0.553536 | Val Acc: 0.695894 loss: 1.019265
[017/020] Train Acc: 0.820493 Loss: 0.535859 | Val Acc: 0.698317 loss: 1.011478
[018/020] Train Acc: 0.825414 Loss: 0.519137 | Val Acc: 0.695476 loss: 1.042427
[019/020] Train Acc: 0.830811 Loss: 0.502845 | Val Acc: 0.691114 loss: 1.057155
[020/020] Train Acc: 0.835826 Loss: 0.486575 | Val Acc: 0.692049 loss: 1.058787
```

可以看到没什么进展，甚至loss更大。我考虑到可能是我只调整了batch而没有改loss，同时考虑到20轮次太少，决定将训练轮次epoch翻倍

```
[001/050] Train Acc: 0.525220 Loss: 1.584882 | Val Acc: 0.624301 loss: 1.224388
saving model with acc 0.624
[002/050] Train Acc: 0.647841 Loss: 1.134446 | Val Acc: 0.666031 loss: 1.067446
saving model with acc 0.666
[003/050] Train Acc: 0.686789 Loss: 0.990605 | Val Acc: 0.683853 loss: 1.000047
saving model with acc 0.684
[004/050] Train Acc: 0.709503 Loss: 0.909010 | Val Acc: 0.694646 loss: 0.958071
saving model with acc 0.695
[005/050] Train Acc: 0.726119 Loss: 0.850894 | Val Acc: 0.699179 loss: 0.940308
saving model with acc 0.699
[006/050] Train Acc: 0.738604 Loss: 0.805914 | Val Acc: 0.703590 loss: 0.924496
saving model with acc 0.704
[007/050] Train Acc: 0.749498 Loss: 0.768829 | Val Acc: 0.702562 loss: 0.930125
[008/050] Train Acc: 0.758744 Loss: 0.735630 | Val Acc: 0.703476 loss: 0.929885
[009/050] Train Acc: 0.767662 Loss: 0.706983 | Val Acc: 0.703883 loss: 0.928784
saving model with acc 0.704
[010/050] Train Acc: 0.775463 Loss: 0.680141 | Val Acc: 0.703216 loss: 0.943100
[011/050] Train Acc: 0.782889 Loss: 0.656170 | Val Acc: 0.700565 loss: 0.952324
[012/050] Train Acc: 0.790131 Loss: 0.633150 | Val Acc: 0.703094 loss: 0.963501
[013/050] Train Acc: 0.796688 Loss: 0.611609 | Val Acc: 0.699159 loss: 0.979420
[014/050] Train Acc: 0.802591 Loss: 0.591415 | Val Acc: 0.699716 loss: 0.987717
[015/050] Train Acc: 0.809046 Loss: 0.572342 | Val Acc: 0.700057 loss: 0.986309
[016/050] Train Acc: 0.814368 Loss: 0.553536 | Val Acc: 0.695894 loss: 1.019265
```

但是可以看到，整体是没什么进展的。考虑到是不是这一次应该放大batch。并且考虑到验证集并不需要太多，于是将VAL_RATIO 改为0.01。下图为运行结果：

```
[040/050] Train Acc: 0.739916 Loss: 0.817639 | Val Acc: 0.682358 loss: 0.895482
saving model with acc 0.682
[041/050] Train Acc: 0.741355 Loss: 0.812363 | Val Acc: 0.682602 loss: 0.900371
saving model with acc 0.683
[042/050] Train Acc: 0.742991 Loss: 0.807362 | Val Acc: 0.676992 loss: 0.913897
[043/050] Train Acc: 0.744198 Loss: 0.802445 | Val Acc: 0.679675 loss: 0.908292
[044/050] Train Acc: 0.745487 Loss: 0.797590 | Val Acc: 0.678211 loss: 0.903431
[045/050] Train Acc: 0.746871 Loss: 0.792931 | Val Acc: 0.682520 loss: 0.898968
[046/050] Train Acc: 0.748137 Loss: 0.788367 | Val Acc: 0.682520 loss: 0.892196
[047/050] Train Acc: 0.749370 Loss: 0.784019 | Val Acc: 0.680081 loss: 0.903664
[048/050] Train Acc: 0.750219 Loss: 0.779704 | Val Acc: 0.680650 loss: 0.903076
[049/050] Train Acc: 0.751517 Loss: 0.775569 | Val Acc: 0.683089 loss: 0.889765
saving model with acc 0.683
[050/050] Train Acc: 0.752880 Loss: 0.771166 | Val Acc: 0.683008 loss: 0.891318
```

但是结果明显不好，考虑到第49个epoch还是在更新，所以对参数进一步调整，我将epoch的轮数改为100，并且之前调整batch忘了调整Learning Rate，这次将Learning Rate调整为1e-5.

```
[190/200] Train Acc: 0.719627 Loss: 0.891099 | Val Acc: 0.673333 loss: 0.981327
saving model with acc 0.673
[191/200] Train Acc: 0.719910 Loss: 0.889804 | Val Acc: 0.671382 loss: 0.981308
[192/200] Train Acc: 0.720244 Loss: 0.888513 | Val Acc: 0.673333 loss: 0.978312
[193/200] Train Acc: 0.720668 Loss: 0.887211 | Val Acc: 0.671951 loss: 0.979169
[194/200] Train Acc: 0.720804 Loss: 0.886054 | Val Acc: 0.673496 loss: 0.979731
saving model with acc 0.673
[195/200] Train Acc: 0.721256 Loss: 0.884855 | Val Acc: 0.673171 loss: 0.975997
[196/200] Train Acc: 0.721664 Loss: 0.883505 | Val Acc: 0.671463 loss: 0.978889
[197/200] Train Acc: 0.722126 Loss: 0.882144 | Val Acc: 0.672927 loss: 0.975972
[198/200] Train Acc: 0.722292 Loss: 0.881108 | Val Acc: 0.675935 loss: 0.972522
saving model with acc 0.676
[199/200] Train Acc: 0.722787 Loss: 0.879873 | Val Acc: 0.674390 loss: 0.975749
[200/200] Train Acc: 0.723103 Loss: 0.878656 | Val Acc: 0.672846 loss: 0.975043
```

结果依然不好，怀疑是学习率太低了。我怀疑是我之前调的太乱了，也有可能第一次是在自己的电脑上跑的，我决定重新测试一下。测试结果是没问题的，还是0.704。于是我尝试将batch增大，其在第六个轮次就达到了0.704.感觉不清楚参数还能怎么改。

```
[003/100] Train Acc: 0.716529 Loss: 0.876784 | Val Acc: 0.695260 loss: 0.945635
saving model with acc 0.695
[004/100] Train Acc: 0.735505 Loss: 0.810143 | Val Acc: 0.702602 loss: 0.927290
saving model with acc 0.703
[005/100] Train Acc: 0.749727 Loss: 0.758935 | Val Acc: 0.701204 loss: 0.933358
[006/100] Train Acc: 0.763031 Loss: 0.715390 | Val Acc: 0.703566 loss: 0.930530
saving model with acc 0.704
[007/100] Train Acc: 0.774084 Loss: 0.678229 | Val Acc: 0.697521 loss: 0.961021
[008/100] Train Acc: 0.783792 Loss: 0.645022 | Val Acc: 0.701387 loss: 0.959118
[009/100] Train Acc: 0.792969 Loss: 0.615648 | Val Acc: 0.695167 loss: 0.990075
[010/100] Train Acc: 0.802192 Loss: 0.586600 | Val Acc: 0.696821 loss: 0.997833
[011/100] Train Acc: 0.810427 Loss: 0.561071 | Val Acc: 0.692638 loss: 1.021846
[012/100] Train Acc: 0.817765 Loss: 0.535903 | Val Acc: 0.693167 loss: 1.038383
[013/100] Train Acc: 0.825310 Loss: 0.513986 | Val Acc: 0.692130 loss: 1.057622
```

然后我尝试将sigmoid激活函数改为ReLU。最后得到结果为：

```
[003/100] Train Acc: 0.716432 Loss: 0.871280 | Val Acc: 0.689195 loss: 0.960753
saving model with acc 0.689
[004/100] Train Acc: 0.734259 Loss: 0.808872 | Val Acc: 0.694073 loss: 0.950658
saving model with acc 0.694
[005/100] Train Acc: 0.748320 Loss: 0.760577 | Val Acc: 0.699049 loss: 0.943637
saving model with acc 0.699
[006/100] Train Acc: 0.759954 Loss: 0.720589 | Val Acc: 0.701704 loss: 0.940250
saving model with acc 0.702
[007/100] Train Acc: 0.770012 Loss: 0.687290 | Val Acc: 0.698118 loss: 0.961375
[008/100] Train Acc: 0.779230 Loss: 0.656535 | Val Acc: 0.699135 loss: 0.967799
[009/100] Train Acc: 0.787737 Loss: 0.628911 | Val Acc: 0.695403 loss: 0.985193
[010/100] Train Acc: 0.795819 Loss: 0.603132 | Val Acc: 0.694155 loss: 1.005560
[011/100] Train Acc: 0.803058 Loss: 0.579555 | Val Acc: 0.693024 loss: 1.016315
[012/100] Train Acc: 0.809320 Loss: 0.558764 | Val Acc: 0.694122 loss: 1.031247
[013/100] Train Acc: 0.815645 Loss: 0.538678 | Val Acc: 0.694114 loss: 1.048455
```

可以看到ReLU的表现要比sigmoid好一点。

然后我想到进一步调整参数，可以看到在第6轮就获得了最佳结果，于是我将Learning调为0.0001，batch改为512，最终得到的结果有明显的进步：

```
saving model with acc 0.710
[012/100] Train Acc: 0.757331 Loss: 0.741528 | Val Acc: 0.712383 loss: 0.895947
saving model with acc 0.712
[013/100] Train Acc: 0.762761 Loss: 0.723792 | Val Acc: 0.710631 loss: 0.899155
[014/100] Train Acc: 0.767350 Loss: 0.707805 | Val Acc: 0.710127 loss: 0.902422
[015/100] Train Acc: 0.772413 Loss: 0.691438 | Val Acc: 0.712904 loss: 0.896851
saving model with acc 0.713
[016/100] Train Acc: 0.777279 Loss: 0.675958 | Val Acc: 0.710196 loss: 0.904296
[017/100] Train Acc: 0.781568 Loss: 0.661551 | Val Acc: 0.711948 loss: 0.904137
[018/100] Train Acc: 0.785980 Loss: 0.647382 | Val Acc: 0.709790 loss: 0.912864
```

然后根据网上的提示，对网络结构进一步进行进行了修改。

将结构修改为如下所示：

```
class Classifier(nn.Module):
def __init__(self):
        super(Classifier, self).__init__()
        self.net= nn.Sequential(
            nn.Linear(429, 2048),# 1nn.LeakyReLU(),
#nn.ReLU(),nn.BatchNorm1d(2048),
            nn.Dropout(0.5),
            nn.Linear(2048, 2048),# 2nn.LeakyReLU(),
#nn.ReLU(),nn.BatchNorm1d(2048),
            nn.Dropout(0.5),
            nn.Linear(2048, 2048),# 2nn.LeakyReLU(),
#nn.ReLU(),nn.BatchNorm1d(2048),
            nn.Dropout(0.5),
            nn.Linear(2048,1024),# 3nn.LeakyReLU(),
#nn.ReLU(),nn.BatchNorm1d(1024),
            nn.Dropout(0.5),
            nn.Linear(1024, 512),# 4#nn.ReLU(),nn.LeakyReLU(),
            nn.BatchNorm1d(512),
            nn.Dropout(0.5),
            nn.Linear(512, 256),# 5#nn.ReLU(),nn.LeakyReLU(),
            nn.BatchNorm1d(256),
            nn.Dropout(0.5),
            nn.Linear(256, 39)
        )
```

```
def forward(self, x):
        x= self.net(x)
return x
```

目前代码跑了一半（已经8个小时了），从当前输出日志可以看到结果明显比之前好：

```
25346.1s    81    [034/100] Train Acc: 0.750810 Loss: 0.771163 | Val Acc: 0.759048 loss: 0.741496
25346.1s    82    saving model with acc 0.759
26083.7s    83    [035/100] Train Acc: 0.751792 Loss: 0.767514 | Val Acc: 0.759292 loss: 0.741294
26083.7s    84    saving model with acc 0.759
26819.3s    85    [036/100] Train Acc: 0.753802 Loss: 0.761662 | Val Acc: 0.760443 loss: 0.738916
26819.3s    86    saving model with acc 0.760
27557.6s    87    [037/100] Train Acc: 0.754653 Loss: 0.757883 | Val Acc: 0.759621 loss: 0.738843
28294.1s    88    [038/100] Train Acc: 0.756141 Loss: 0.752623 | Val Acc: 0.760471 loss: 0.739178
28294.1s    89    saving model with acc 0.760
```

从改变的结构上看，新的代码有如下改动和优势：

1、网络深度和宽度都增加

新结构增加了网络深度（从 3 个隐藏层增加到 6 个隐藏层）和宽度（每层神经元数量更多）。更深更宽的网络具有更强的表达能力，能够学习更复杂的非线性映射关系，可能在复杂任务上表现更好。

2、激活函数改变

将 ReLU 改为 LeakyReLU。LeakyReLU 解决了 ReLU 的 "神经元死亡" 问题，允许负输入时有小梯度通过，有助于缓解梯度消失。

标准 ReLU 的定义是：ReLU(x)=max(0,x)当输入值为负数时，ReLU 会将其置为 0，这可能导致部分神经元在训练过程中永久 “死亡”(即不再对任何输入产生响应)。

而 **LeakyReLU** 通过允许负输入有一个小的非零梯度来解决这个问题：

$$\text{LeakyReLU}(x) = \begin{cases} x, & \text{如果 } x \geq 0 \\ \alpha x, & \text{如果 } x < 0 \end{cases}$$

其中，α 是一个很小的常数（通常为 0.01），称为 “泄漏率”(leakage rate)。

3、增加了批量归一化

每层后添加了 BatchNorm1d。加速训练收敛，增强模型稳定性，减少对初始化的依赖，并具有一定正则化效果。

BatchNorm1d的核心思想是在训练深度神经网络时，每一层的输入分布会随着前层参数的更新而变化，这被称为 **内部协变量偏移**（Internal Covariate Shift）。这种变化会

导致训练变慢，需要更小的学习率和更精细的参数初始化。

BatchNorm 的目标：通过对每一批次数据进行归一化，使每层输入的分布保持稳定，从而加速训练并提高模型泛化能力。

4、增加了 Dropout 正则化

每层后添加了 Dropout (0.5)。通过随机丢弃神经元，减少过拟合，提高模型泛化能力。

5、网络结构设计

采用了金字塔式结构（2048→1024→512→256）。有助于提取不同层次的特征表示，从一般特征到具体特征逐步细化。