

数据科学HW2

刘家宁

November 2025

1 实验结果

本次作业的内容是关于对12个被试在62个通道，5个频率上的脑电信号进行情绪分类，其中情绪分为积极，中性和消极。本次实验采用跨被试留一交叉验证，即令每个被试成为测试集，其他所有被试成为训练集进行12次单独的训练，最后根据准确率的平均值和方差来判定模型的好坏与泛化能力。首先给出本次实验的结果：

Model	acc_mean	acc_std
CNN	0.5927	0.1270
LSTM	0.5290	0.1415
AutoEncoder	0.4926	0.1086
MLP	0.5693	0.1158

Table 1: 不同模型在留一交叉验证下的准确率均值和标准差

2 数据预处理

由于留一交叉验证本身就能体现模型的泛化能力，所以这里对于每次训练不划分CV集，只有训练集和测试集。

对于每次训练，保留一个被试作为测试集，其他所有被试成为训练集。

首先对于 62×5 个通道和频段分别计算mean和std，做z-score归一化。这里不选择对每个人分别进行归一化的原因是这样跨被试差异被消除，每个人都是均值为0方差为1的分布，模型无法学习不同被试之间的统计分布，且在训练集和测试集上使用的标准化公式不一致。

对于CNN，我们使用给定的二维矩阵来把原本的62个通道变为 8×9 个通道，并在部分无通道的位置补0。这样的好处是模型现在可以利用空间信息在二维上进行卷积。

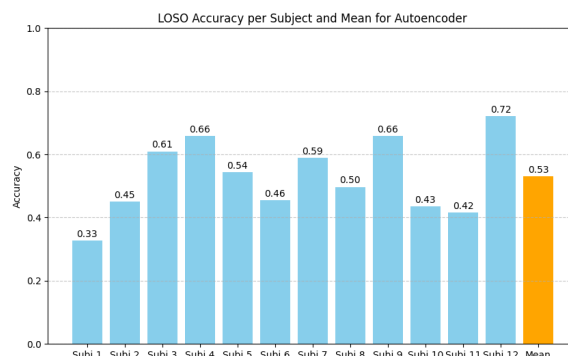


Figure 1: Enter Caption

对于时序模型LSTM，我们先将每个被试的数据切成若干个连续段，然后取大小为 $T = 10$ 的窗口进行分类。这样是为了保证时序模型能获得足够的时序信息，而不仅仅是某一时间点的切片。对于数据增强，我们尝试了三个方式进行数据增强：

- **高斯噪声**：对数据进行0.01倍方差的微扰，这是考虑到标准化采用的是训练集的数据，但测试集的数据在标准化之后未必方差为1。
- **幅值缩放**：考虑到每个人情绪激烈程度不同，我们对幅值进行缩放，倍数在(0.95,1.05)之间。
- **随机dropout**：随机dropout一些通道，尝试泛化模型的学习能力。

3 CNN

在CNN训练中，我们使用的optimizer是Adam，损失函数为WeightedCrossEntropy，后者可以帮我们处理类别不平衡问题。

我们进行了多次尝试，激活函数均为ReLU，以下是第一次尝试：

- conv1: Conv2d(in_channels=5, out_channels=16, kernel_size=3, padding=1)
- bn1: BatchNorm2d(16)
- conv2: Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)
- bn2: BatchNorm2d(32)
- pool: MaxPool2d(kernel_size=2, stride=2, padding=(0,1))
- fc1: Linear(32*4*5, 128)
- dropout: Dropout(0.5)

- fc2: Linear(128, 3)
- learning_rate = 1e-3, weight_decay = 1e-4, epoch = 20

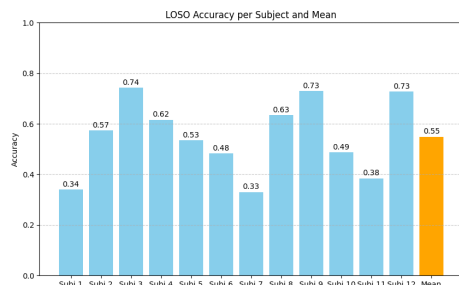


Figure 2: 第一次尝试的训练结果

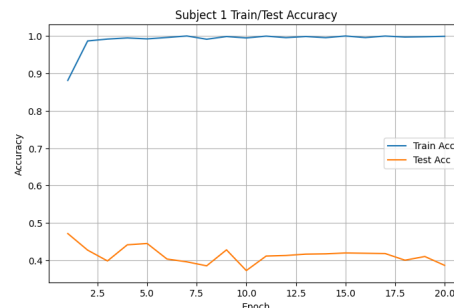


Figure 3: 在第二位受试上的结果

但是这样训练起来比较慢，而且我们注意到它在不到5个epoch之后迅速就拟合了，所以我们减少了参数的大小和learning rate来进行第二次尝试：

- conv1: Conv2d(in_channels, 16, kernel.size=3, padding=1)
- bn1: BatchNorm2d(16)
- pool: MaxPool2d(kernel.size=2, stride=2, padding=(0,1))
- fc1: Linear(16 * 4 * 5, 64)
- dropout: Dropout(0.5)
- fc2: Linear(64, 3)
- learning_rate = 1e-4, weight_decay = 1e-4, epoch = 10

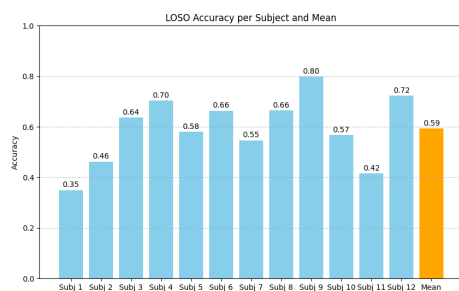


Figure 4: 第二次尝试的训练结果

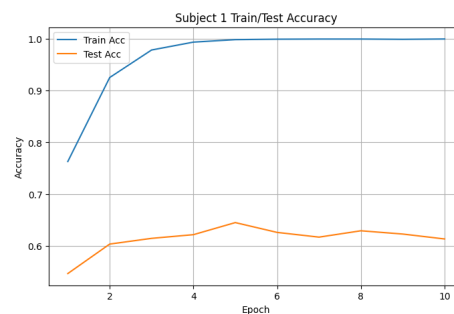


Figure 5: 在第二位受试上的结果

我们可以看到，虽然参数量减少了，但是整体正确率反而上升了，同时train acc的曲线也更加平缓，不像第一次来回震荡，整体正确率上升了。同时我们发现被试之间差距比较明显，有些被试

的数据分布明显与他人有显著的不同，我们进行了多次实验，但是模型在1号被试上总是正确率较低，在其他受试上均至少在一次实验中有超过0.5的成绩。

4 LSTM

LSTM其实就是升级版的RNN，比起RNN来更加不容易产生梯度爆炸/消失。以下是LSTM在时间窗口不同（ $T = 10, 20, 30$ ）的时候的结果：

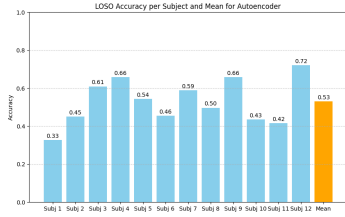


Figure 6: $T=10$ 的结果

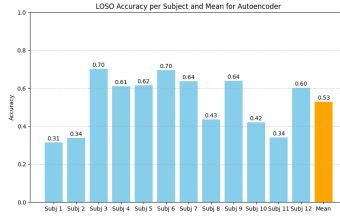


Figure 7: $T=20$ 的结果

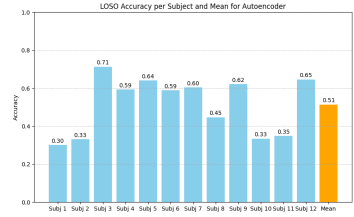


Figure 8: $T=30$ 的结果

其中在 $T=10$ 和 $T=20$ 时表现较好，但是感觉这里LSTM没有体现出优势，正确率还不如只能看1帧的CNN高。

5 Autoencoder

Autoencoder是通过一个虚假的任务（重建原本的内容）来训练一个对原本输入的编码器。在第一步训练完成后，我们冻结autoencoder的参数，然后将编码器的输出喂给classifier来分类。为了简单期间，这里我们的classifier就是一个包含全连接和dropout层的简单MLP。

这里考虑到如果autoencoder仍然采用全连接层去做，可能本质上和一个MLP没有什么区别，所以最后我们选择了用卷积层做，但是其实效果仍然不是很好，原因应该是我autoencoder的结构不够好。我们最后采用的方案是这样的：

- conv1: Conv2d(in_channels, 16, kernel_size=3, padding=1)
- ReLU()
- pool1: MaxPool2d(2, 2)
- conv2: Conv2d(16, 32, kernel_size=3, padding=1)
- ReLU()
- pool2: MaxPool2d(2, 2)
- learning_rate = 1e-3, weight_decay = 1e-4, epoch = 10

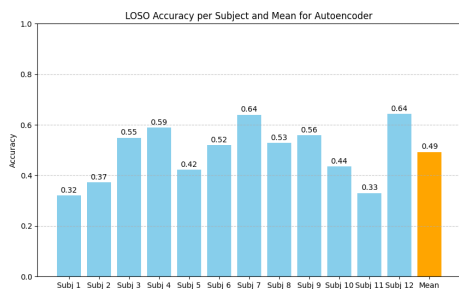


Figure 9: Autoencoder的结果

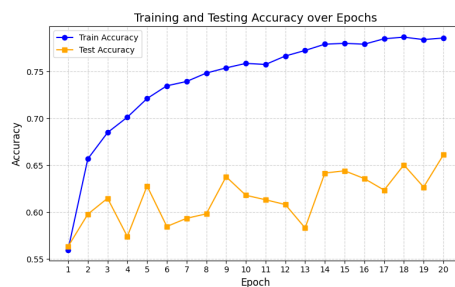


Figure 10: 在第三位受试上的结果

而且我们再次验证了第一个被试是非常特殊的，因为在训练autoencoder的时候只有第一个被试的train loss和test loss差距非常大。

6 MLP

这里我们简单采用了两层隐藏层的简单MLP，结构如下：

- hidden_dim1 = 256, hidden_dim2 = 128
- fc1: Linear(310, hidden_dim1)
- bn1: BatchNorm1d(hidden_dim1)
- fc2: Linear(hidden_dim1, hidden_dim2)
- bn2: BatchNorm1d(hidden_dim2)
- fc3: Linear(hidden_dim2, 3)
- dropout: Dropout(p=0.3)

从结果来看，其实最简单的MLP模型效果反而是不错的，而CNN只比它好0.02。

上次作业中LDA的表现非常不错，所以这次我们试一下先用LDA进行降维然后再丢到MLP中去做分类，但是我发现由于LDA降维的要求是 $\dim \leq \min(n_{features}, n_{classes} - 1)$ ，所以我们只能保留2维的信息，这样其实反而丢掉了深度学习的优势，因为输入的参数太少了，MLP压根就没干啥，信息已经都被LDA压缩完了。这里感觉道理是既然参数量还跑的动，那就可以先都交给深度学习做，没必要自己乱降维，反而丢信息。结果如下：

7 总结

感觉这次整体的结果很不理想，可能是因为没有找到合适的超参数，当然这也和数据集存在坏点（第一位受试者）以及样本量有关。大部分的情况都是很快在training set上收敛了但是在test set上

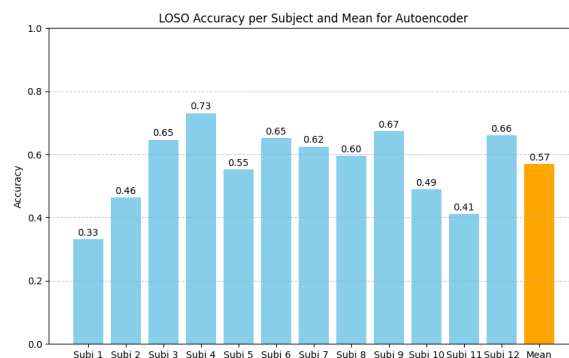


Figure 11: MLP的结果

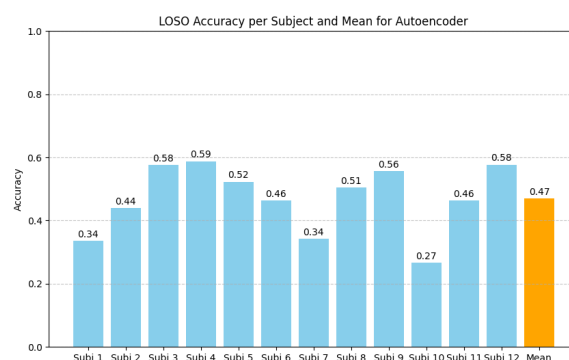


Figure 12: 先LDA再MLP的结果

表现很差。我其实原本比较期待LSTM的效果会比较好，因为他可以多看几帧的信息，但是事实上好像也没有带来特别大的优势。其实这次数据预处理做的比较简单，原本还想做一下PCA或者其他降维方法的，但是感觉跑一次实验太久了就没做。中间有尝试做一些数据增强，但是感觉效果并不显著，感觉还是输入的个人差异过大。