忆阻器神经网络报告

1. 网络结构

本次实验搭建了一个比较简单的卷积神经网络，网络设计如下：

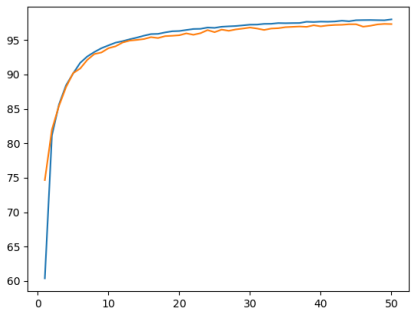
```
def forward(self, x):
    x = F.relu(self.conv1(x))
    x = F.relu(self.conv2(self.pool(x)))
    x = self.pool(x)
    x = x.view(-1, image_size // 4 * image_size // 4 * 8)
    x = F.relu(self.fc1(x))
    x = F.dropout(x, training=self.training, p=0.4)
    x = F.log_softmax(self.fc2(x), dim=1)
    return x
```

图 1: 模型设计

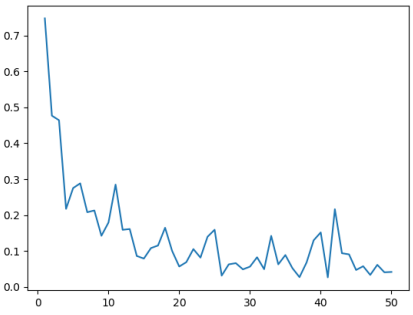
2. 参数对网络的影响

2.1. base 模型

蓝色为 train_acc，红色为 val_acc 结果如下：

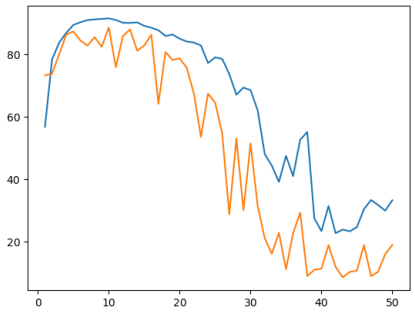


loss 曲线如下：

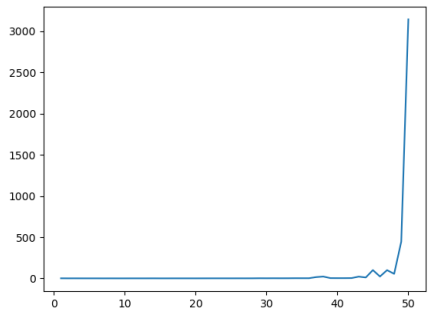


2.2. 忆阻器网络

使用 ppt 种给出的加噪函数模拟忆阻器的效果，guass 的参数取 mean 为 0，std 为 0.3。在每一次 eval 之前调用加噪函数。效果如下：



loss 曲线如下：

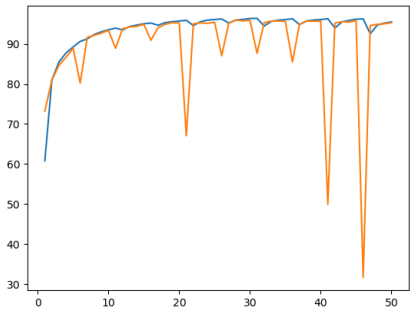


可以发现 acc 在训练次数增长时，会波动性的下降，模型的准确率最终会下降到一个很差的地步，10% 左右的准确率约等于模型崩掉了。

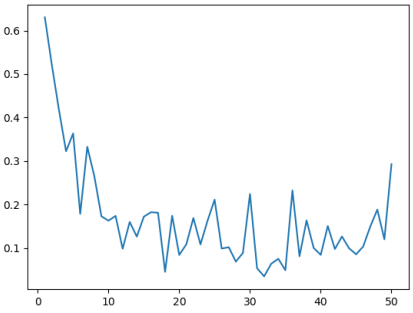
出现这个问题的原因我的想法是这样的。联想到课堂上讲到的忆阻器的记忆功能，在本次问题中，每次给模型加了噪声后，由于记忆的功能，模型会把这个“偏置”记住，但事实上，如果训练的次数足够多，模型能够把参数的噪声消除，但是为什么没有消除呢，我的理解是加噪太频繁，每次的 train 不足以消除噪声的影响，train 完后的 val 却又引入了噪声，类比于忆阻器的记忆功能，网络

不断记住了这个“偏置”，导致模型积累的偏置越来越大，最终模型崩掉。

因此我想到加噪不应该如此频繁，我选择每 5 个 epoch 加一次噪。结果如下：



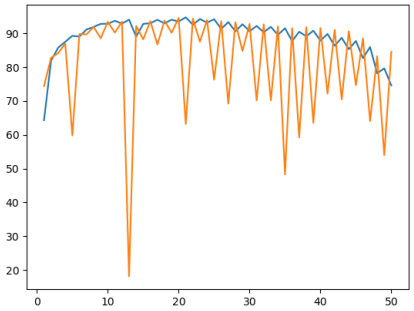
loss 曲线如下：



可以看出，val 的 acc 会在每五个 epoch 后骤降，但是经过几轮的 train 后，模型可以消除噪声的影响，仍然保持较好的准确率，约 95%，但比起 base（97%）还是有所下降。这印证了前面的猜想。

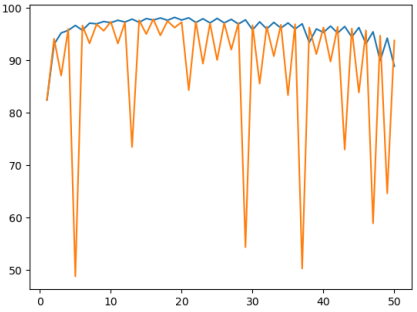
2.3. 学习率对比

采用每 2 个 epoch 后进行一次加噪，作为对照组。lr 为 0.0001 时的 acc 图像如下：



最终 acc 会在 84% 左右。

learning rate 改为 0.001，acc 图像如下：

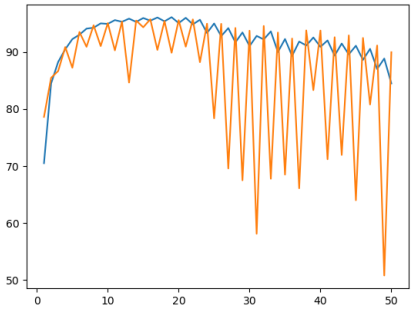


最终 acc 会在 94% 左右。

分析原因是因为 0.0001 作为 learning rate 偏小，训练时迭代的步幅比较小，更新达到极值的速度慢，比起 0.001 效果略差。

2.4. batchsize 对比

learning rate 改为 0.0001，batch size 改为 64，结果如下：

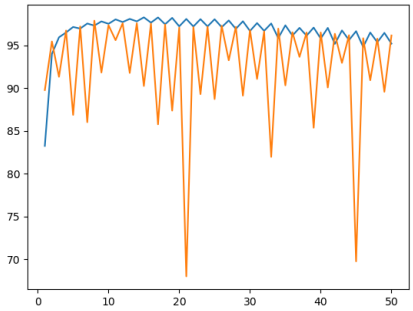


最终 acc 会在 90% 左右。

由于 bs 的增加，在同样的 epoch 的情况下，会使网络的 weights 更新迭代的次数变少，所以需要对 LR 随着 bs 的增加而线性增加，因此在相同的学习率下，batchsize 适当小时效果会更好。当然这和模型也有关。但是 batchsize 大时我们也不能太过于增大学习率，因为这会导致最终的收敛不稳定，精度有所下降。

2.5. 层数的对比

我添加了一层卷积层。因为卷积层增加后，模型参数量更大，因此我增加了学习率到 0.001，之后进行测试。结果如下：



最终 acc 会在 96% 左右，比起两层时的 94%acc，可以看到效果是上升的。这是显而易见的因为加深网络会学习到更加多的深层特征，这样测试的效果肯定会上升。但是大网络带来的问题就是训练太慢和太难。而本任务由于数据集和网络都比较简答，不太需要考虑这个问题。

同时，加深网络可以加强模型的鲁棒性，因此下面的部分对网络的改进里，我将不会从深度方面去考虑。

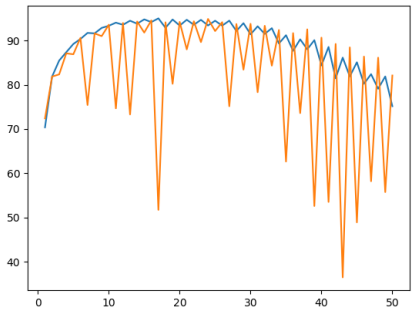
3. 改进

其实观察前面的训练过程可以发现，因为噪声的加入，训练的过程中 acc 会出现很大的波动。ppt 中也指出，这种波动是忆阻器神经网络的需要解决的地方。在本实验中我都使用的是 Adam 优化器，Adam 优化器在面对这样波动性大的数据收敛速度不够快。

想要优化这个结果，我想到的方法是更改优化器，针对这样波动性大的数据，含动量的优化器能够快速优化，因此我认为需要选用带有动量的优化器。在查询后，我使用了这样的两个优化器进行改进。

RMSprop，其思想是，梯度震动较大的项，在下降时，减小其下降速度；对于震动幅度小的项，在下降时，加速其下降速度。

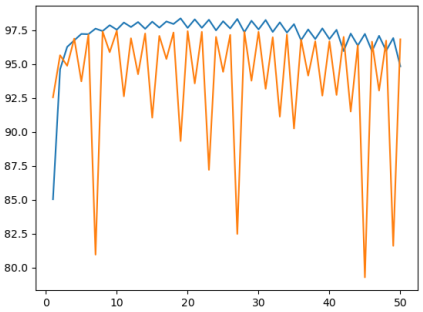
效果如下：



最终 acc 会在 85% 左右。

虽然这种情况可能出现了过拟合的情况，但是在观察中间的准确率变化后，发现最大的 acc 也会有 95%，当然优化器本身也有很多参数，如果仔细调的话应该也可以取得比较好的稳定性。但是这样的作法比较耗时耗力，因此我查询资料找到了 NAdam 优化器。

NAdam，是 Adam 的改进版，其类似于带有 Nesterov 动量项的 Adam，Nadam 对学习率有了更强的约束，同时对梯度的更新也有更直接的影响。一般而言，在想使用带动量的 RMSprop，或者 Adam 的地方，大多可以使用 NAdam 取得更好的效果。因此选用 NAdam。效果如下。



最终 acc 会在 96% 左右。

可以看到这印证了我们的两个猜想：1、动量优化器可以取得更好的效果，因为动量优化器可以更快迭代波动性大的数据。2、NAdam 确实会比 RMSprop 优化器更好。

这两个优化器的代码如下。

```
#optimizer = optim.RMSprop(params=net.parameters(), lr=0.0001, alpha=0.99, eps=1e-08, weight_decay=0, momentum=0, centered=False)
#optimizer = optim.NAdam(params=net.parameters(), lr=0.002, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, momentum_decay=0.004)
```

我还尝试了一些数据集增强的方法，但是效果并没有什么变化。而且作业要求的是对模型的改进，因此我也没有太过多的尝试。