学习率

选择合适的学习率对于神经网络训练的成功至关重要，包括对于训练卷积神经网络（CNN）分类MNIST数据集。学习率决定了模型在每次迭代中参数更新的幅度。如果学习率过高，模型可能会在最优解附近震荡而无法收敛；如果学习率过低，训练过程可能会过于缓慢，需要更多时间才能达到满意的结果。

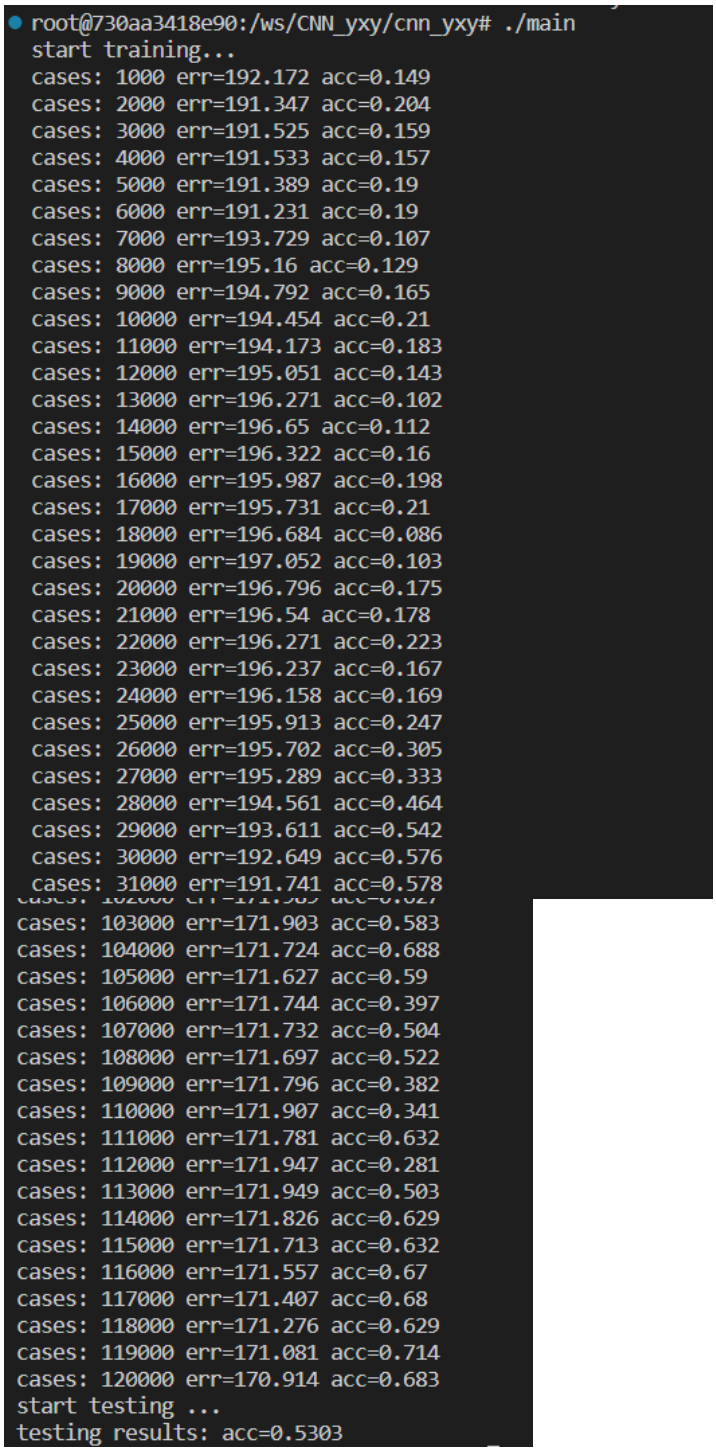
对于MNIST这类相对简单的数据集，初始学习率通常可以设置在0.01到0.001之间。然而，最佳学习率往往需要根据具体的模型架构，优化器选择，及其它训练参数等进行微调。

在实践中，一种常见的方法是使用学习率衰减策略，比如步长衰减、指数衰减或余弦退火等。这意味着开始时可以设定一个较大的学习率，如0.01，然后随着训练的进行逐步降低学习率。这样可以帮助模型在初始阶段快速收敛，而在后期更精细地调整权重。

总的来说，选择最佳学习率通常需要一些实验和调整。

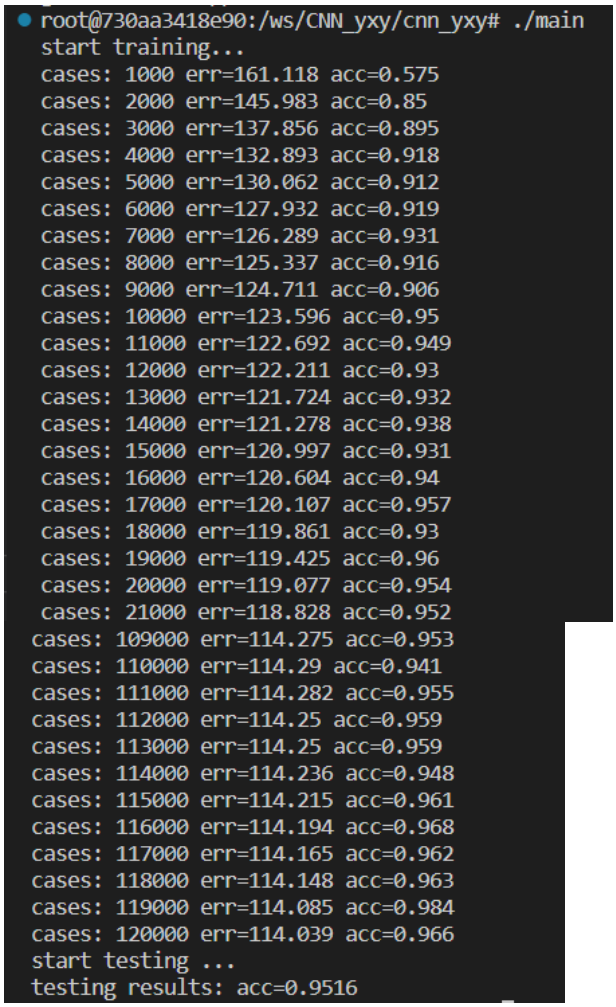
- **LEARNING_RATE=0.1**：
    - **在复杂的网络结构中**：学习率高，大幅震荡明显（err与acc都在震荡），难收敛，效果差

```
//三层卷积层的神经网络结构
M.add_conv( 1, 5, 8, {28, 28, 1} );
M.add_relu( M.output_size() );
M.add_pool( 2, 2, M.output_size() );

M.add_conv( 1, 2, 10, M.output_size() );
M.add_relu( M.output_size() );
M.add_pool( 2, 2, M.output_size() );

M.add_conv( 1, 2, 12, M.output_size() );
M.add_relu( M.output_size() );

M.add_fc( M.output_size(), 10 );
```



- **在简单的网络结构中**：学习率高，收敛速度快，训练后期出现震荡（轻度），测试集上的表现较好

```
//一层卷积层的神经网络结构
M.add_conv( 1, 5, 10, {28, 28, 1} );
M.add_relu( M.output_size() );
M.add_pool( 2, 2, M.output_size() );

M.add_fc( M.output_size(), 10 );
```



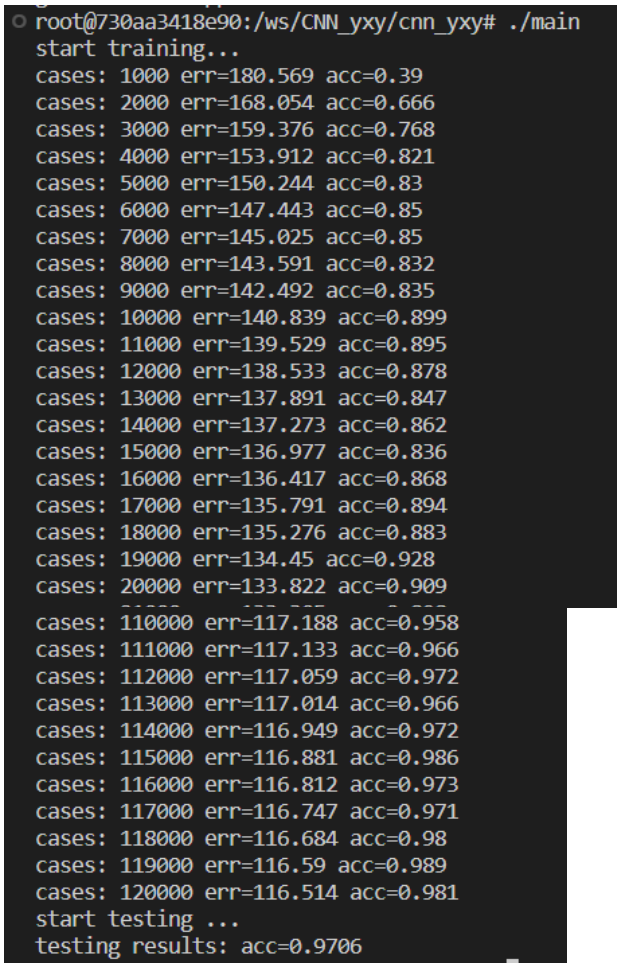
- **LEARNING_RATE=0.01**：

- **复杂网络（三层卷积层）**：没有大幅震荡，err稳步下降，逐步收敛，拟合效果较好，测试集上表现更优

```
root@730aa3418e90:/ws/CNN_yxy/cnn_yxy# ./main
start training...
cases: 1000 err=195.464 acc=0.124
cases: 2000 err=194.228 acc=0.175
cases: 3000 err=193.547 acc=0.188
cases: 4000 err=193.16 acc=0.191
cases: 5000 err=192.755 acc=0.227
cases: 6000 err=192.516 acc=0.234
cases: 7000 err=192.231 acc=0.263
cases: 8000 err=191.976 acc=0.285
cases: 9000 err=191.653 acc=0.307
cases: 10000 err=191.199 acc=0.377
cases: 11000 err=190.712 acc=0.368
cases: 12000 err=190.065 acc=0.447
cases: 13000 err=189.423 acc=0.424
cases: 14000 err=188.708 acc=0.459
cases: 15000 err=187.985 acc=0.444
cases: 16000 err=187.144 acc=0.539
cases: 17000 err=186.381 acc=0.516
cases: 18000 err=185.641 acc=0.53
cases: 19000 err=184.82 acc=0.587
cases: 20000 err=183.962 acc=0.588
cases: 21000 err=183.243 acc=0.578
cases: 22000 err=182.417 acc=0.614
```

```
cases: 106000 err=156.66 acc=0.847
cases: 107000 err=156.445 acc=0.836
cases: 108000 err=156.233 acc=0.835
cases: 109000 err=155.994 acc=0.852
cases: 110000 err=155.812 acc=0.823
cases: 111000 err=155.59 acc=0.848
cases: 112000 err=155.363 acc=0.844
cases: 113000 err=155.173 acc=0.822
cases: 114000 err=154.958 acc=0.848
cases: 115000 err=154.748 acc=0.842
cases: 116000 err=154.518 acc=0.868
cases: 117000 err=154.291 acc=0.871
cases: 118000 err=154.084 acc=0.856
cases: 119000 err=153.784 acc=0.944
cases: 120000 err=153.516 acc=0.905
start testing ...
testing results: acc=0.8191
```

- **简单网络（单层卷积层）**：收敛速度比高学习率更慢，拟合效果好。



    - 简单网络中收敛速度略慢，在几种网络中都未出现震荡，在测试集上表现都优于高学习率。

- **LEARNING_RATE=0.001**：
    - 在不同网络结构种都表现为：收敛速度慢，拟合不充分。

```
root@730aa3418e90:/ws/CNN_yxy/cnn_yxy# ./main
start training...
cases: 1000 err=192.699 acc=0.223
cases: 2000 err=191.819 acc=0.303
cases: 3000 err=190.251 acc=0.402
cases: 4000 err=188.361 acc=0.446
cases: 5000 err=185.535 acc=0.573
cases: 6000 err=182.621 acc=0.651
cases: 7000 err=179.389 acc=0.715
cases: 8000 err=176.881 acc=0.696
cases: 9000 err=174.295 acc=0.724
cases: 10000 err=171.564 acc=0.787
cases: 11000 err=169.258 acc=0.782
cases: 12000 err=167.335 acc=0.77
cases: 13000 err=165.764 acc=0.736
cases: 14000 err=164.332 acc=0.737
cases: 15000 err=163.218 acc=0.725
cases: 16000 err=161.885 acc=0.785
cases: 17000 err=160.776 acc=0.818
cases: 18000 err=159.729 acc=0.831
cases: 19000 err=158.481 acc=0.882
cases: 20000 err=157.368 acc=0.852
cases: 21000 err=156.384 acc=0.843
cases: 22000 err=155.332 acc=0.873
cases: 23000 err=154.55 acc=0.838
cases: 24000 err=153.728 acc=0.851
cases: 25000 err=153.06 acc=0.836
cases: 26000 err=152.271 acc=0.873
cases: 27000 err=151.606 acc=0.861
cases: 28000 err=150.942 acc=0.875
cases: 101000 err=134.446 acc=0.895
cases: 102000 err=134.393 acc=0.866
cases: 103000 err=134.349 acc=0.855
cases: 104000 err=134.239 acc=0.911
cases: 105000 err=134.156 acc=0.886
cases: 106000 err=134.093 acc=0.861
cases: 107000 err=134.023 acc=0.872
cases: 108000 err=133.958 acc=0.867
cases: 109000 err=133.863 acc=0.905
cases: 110000 err=133.829 acc=0.856
cases: 111000 err=133.755 acc=0.883
cases: 112000 err=133.659 acc=0.902
cases: 113000 err=133.609 acc=0.854
cases: 114000 err=133.522 acc=0.893
cases: 115000 err=133.449 acc=0.879
cases: 116000 err=133.346 acc=0.912
cases: 117000 err=133.255 acc=0.909
cases: 118000 err=133.171 acc=0.902
cases: 119000 err=133.031 acc=0.942
cases: 120000 err=132.909 acc=0.929
start testing ...
testing results: acc=0.8822
```

权重初值

在神经网络训练中，权重的初始化非常重要。合适的初始化可以帮助网络更快地收敛，并可能避免梯度消失或爆炸的问题。

以下是一些常见的权重初始化方法：

1. **零初始化**：最简单的一种方法是将所有权重初始化为零。但这并不是一个好的选择，因为这会导致所有神经元在反向传播过程中学习到相同的更新，这样的网络等同于一个单一神经元的网络。
2. **随机初始化**：将权重初始化为很小的随机数是常见的做法。这可以打破神经元的对称性，每个神经元可以学习到不同的特征。常见的随机初始化方法有均匀分布和正态分布，通常数值范围设定在-0.01到0.01之间。
3. **Glorot/Xavier 初始化**：这种方法由 Xavier Glorot 和 Yoshua Bengio 提出，它的基本思想是保持输入和输出的方差一致。对于tanh等饱和型激活函数，这种初始化方法效果较好。

4. **He 初始化**：He初始化由Kaiming He等人提出，适用于ReLU及其变种的激活函数。与Xavier初始化不同，He初始化在计算权重初始值时考虑了ReLU的非线性特性。

这些初始化策略在许多深度学习框架中，如TensorFlow和PyTorch，都有内置实现。最终选择哪种初始化方法，取决于具体模型和激活函数的选择。

最后，偏置通常可以初始化为零，这并不会带来和权重全零初始化相同的问题，因为偏置并不会参与到层间的相互作用中。

```
//卷积核的初始化
for ( int i = 0; i < extend_filter; i++ )
    for ( int j = 0; j < extend_filter; j++ )
        for ( int z = 0; z < in_size.z; z++ )
            neww( i, j, z ) = 1.0f / N * rand() / 2147483647.0;
```

N 为上一层的神经元数量，2147483647.0 是能 rand() 出来的最大值。

这个初始化方法的基本思路是为每个权重分配一个在0到`1.0f / N`之间的随机数，其中`N`是输入神经元的数量。这是一个简单的标准化或者说缩放策略，让初始权重的大小与输入神经元的数量成反比。

`rand() / 2147483647.0`产生了一个在[0,1]范围内的均匀分布的随机数，然后将这个随机数乘以`1.0f / N`得到最终的初始权重。这个方法的目标是使所有权重的初始值都较小，且均匀分布。

这种初始化策略与Xavier/Glorot初始化有一些相似之处，Xavier/Glorot初始化也是根据输入神经元的数量调整权重的初始值。但在Xavier/Glorot初始化中，权重的初始值是从一个更复杂的分布中抽取的，目的是使得每个神经元的输出都具有相同的方差。