Project 1

written by 奚宏达 22307130464

一、实验目标

本实验是课程《神经网络与数据挖掘》的作业，旨在在原有的框架下搭建MLP与RNN的几个重要部件，并尝试训练出一个可以基于MNIST数据集进行手写数字识别的程序。

二、基本模型

1. Multi-Layer Perceptron（多层感知机）

① 整体架构：

MLP依次包含线性层、激活层（Relu）、线性层共3个层，最后一个线性层用于将结果映射到10个标签上。

② 线性层：

线性层正向传播输入X大小为[batch_size, in_dim]，输出$output = X * W + b$的大小为[batch_size, out_dim]。反向传播中输入的大小为[batch_size, out_dim]，输出$output = grad * W^T$的大小为[batch_size, in_dim]，W的梯度由链式传播可得$\frac{\nabla L}{\nabla W} = \frac{\nabla L}{\nabla output}\frac{\nabla output}{\nabla W} = grad^T * X$，而b的梯度显然就是grad。由于这里的grad是包含很多batch的，因此b的梯度需要grad按行求和得到，不然也会形状不匹配。

线性层起到对输入数据进行加权求和，并加上一个偏置项，从而进行特征变换的作用。而W就是要学习的参数。

③ 激活层：

前向传播中，Relu函数替换掉输入X中小于零的元素为零；反向传播中，根据前向传播中X小于零的元素的位置将grad的对应位置置零。

激活层的作用是引入非线性因素，使得神经网络能够学习复杂的非线性映射。Relu函数也有不容易产生梯度消失的优势。

2. Convolutional Neural Network（卷积神经网络）

① 整体架构：

CNN依次包含卷积层、激活层、池化层、展开层、线性层、激活层、线性层，最后一个线性层用于将结果映射到10个标签上。

② 卷积层：

卷积层前向传播通过公式$H' = \frac{H+2*padding-kernel_size}{stride}$与$W' = \frac{W+2*padding-kernel_size}{stride}$计算输出图像的长与高，再使用kernel对原图像（可以按要求是否加padding）做卷积，加上偏置项后即是输出。卷积层反向传播过程中，卷积核的梯度可以由前向传播公式推出为
$\frac{\nabla L}{\nabla W} = \sum_{b=1}^{batch}\sum_{i=1}^{H'}\sum_{j=1}^{W'} grads[b,k,i,j] \cdot input[b,:,h_{start}:h_{end},w_{start}:w_{end}]$,偏置项的梯度则可以由

$\frac{\nabla L}{\nabla b} = \sum_{b=1}^{batch} \sum_{i=1}^{H'} \sum_{j=1}^{W'} grads[b,k,i,j]$，最后的输出由输入窗口和输出梯度的逐元素乘积求和得到，计算公式为$\frac{\nabla L}{\nabla X} = \sum_{k=1}^{channels_{out}} W[k,c,h\prime,w\prime] \times grads[b,k,i,j]$

卷积层的主要作用是从输入数据中提取局部特征。通过卷积操作，卷积层能够有效捕捉输入图像（或其他类型的输入数据）中的局部模式，如边缘、角点、纹理、形状等。卷积核在输入图像上滑动，并对每个局部区域执行加权求和操作，这样它就能够检测到特定的特征。

③池化层：

池化层类似于图像处理中的下采样，最大池化指的是对于每个小窗口取其中的最大值作为代替整个窗口的值放入新图像，平均池化则是取小窗口的平均值。最大池化反向传播中会根据前向传播记录的最大值位置将反向的输入放上去，平均池化则是将输入的每个值按照在前向传播的对应窗口每一个位置都放上去。

池化层通过缩小特征图的尺寸，显著减少后续卷积层和全连接层所需处理的数据量，从而提高训练速度和推理效率。同时在图像发生小幅移动时也有很好的平移不变性，使得特征图在一定程度上不受位置变化的影响。

3. Loss Function（损失函数）

本实验使用的损失函数为交叉损失，通过累加正确标签label对应的-ln(p[label])计算可得。在有softmax情况下，反向传播中则是输出每次预测的每个结果$\frac{P_i - label}{batch}$计算而得。在没有softmax的情况下，反向传播则是计算每个正确类别的位置的梯度$\frac{-1}{P_i}$。

三、实验探索

问题一：形状不匹配

```
def forward(self, X):
    """
    input: [batch_size, in_dim]
    out: [batch_size, out_dim]
    """
    self.input = X
    output = np.dot(X, self.params['W']) + self.params['b']
    return output
```

本实验最常遇到的问题是形状不匹配，如这里在计算矩阵乘积时最开始写的是np.dot(self.params['W'],X)。这时候才意识到在函数中标明输入与输出大小的好处，由X的大小为[batch_size,in_dim]，self.params['W']为[in_dim,out_dim]，可知应该是X在前。

问题二：bug难找

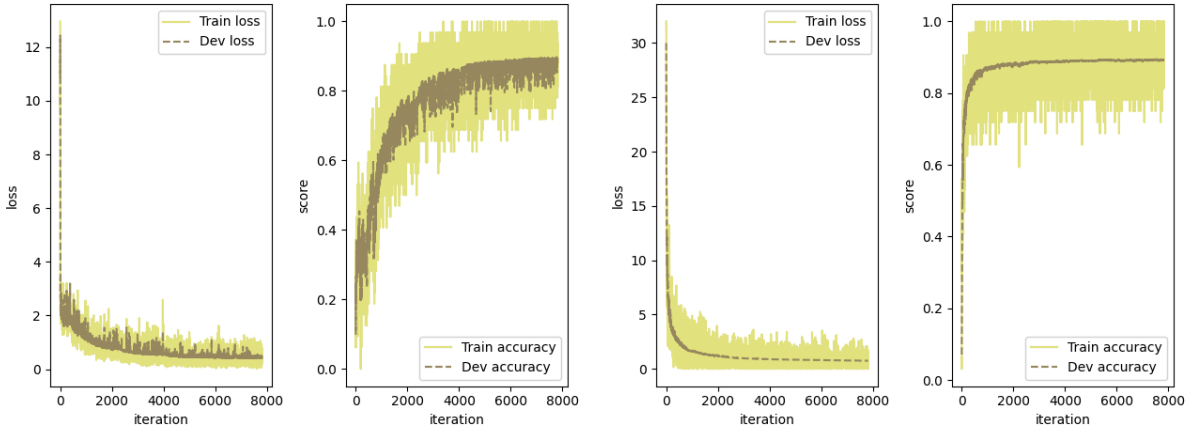
```
<mynn.op.Linear object at 0x00000201E447E0C0>
[-3.34713379e+28  1.13858186e+19  3.55770910e+19  2.68093620e+19
 -1.00789027e+27]
```

如图是在实验中输出的异常梯度，然而实际问题是出在交叉损失反向传播中没对预测结果加softmax使用，在实验中闷头找了半天才找到。后面在实验中加了许多checkpoint以及参数的输出，如在runner中的许多注释起来的输出，可以帮助我便捷地找到bug出错的位置。

四、MLP模型优化

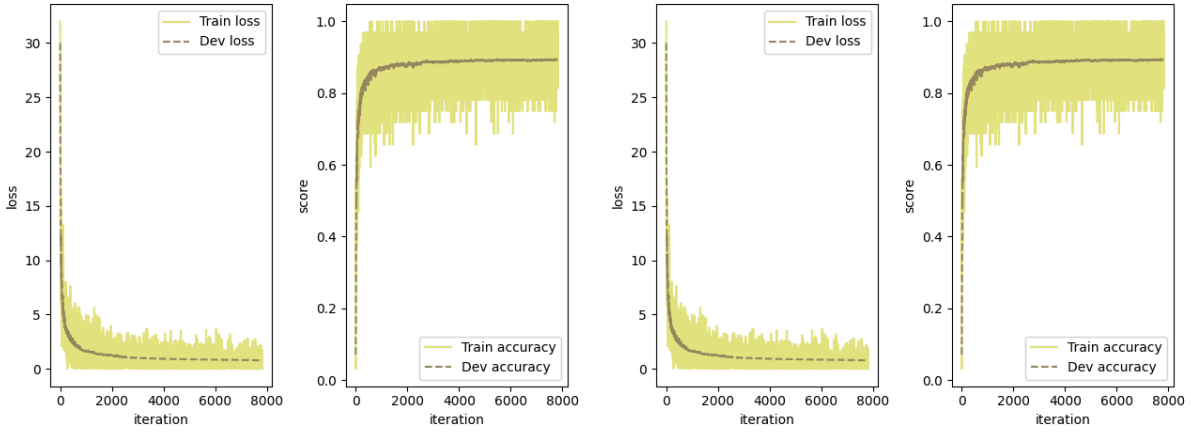
1. 使用平均梯度

在交叉损失计算过程中，最初并没有使用平均梯度的方式传播梯度（没有将结果除batch_size）,训练过程如左下图所示，可以发现loss与score震荡十分严重，推测原因是grad没有取平均导致其基础数值就很大，使得每次梯度更新都移动了过长的距离，让loss与score不能很好地向最优值移动。在取平均梯度后效果明显改善，数值稳定性增强了很多，曲线也更为平滑。



2. 增加L2正则

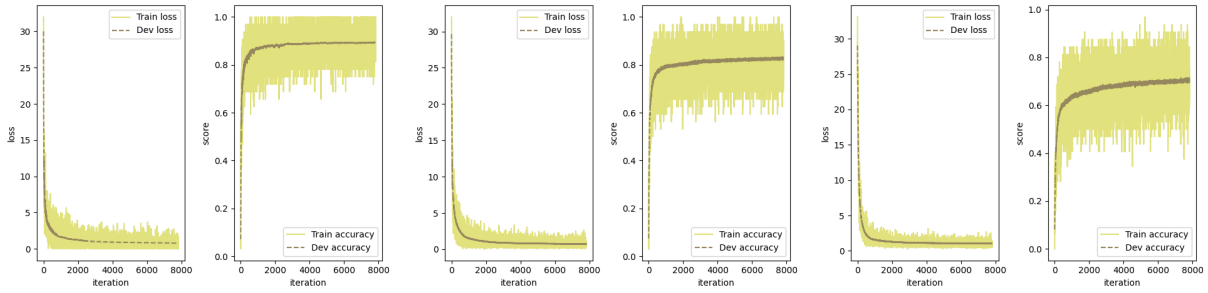
以下两图中左图是未加L2正则的训练过程，右图是加了L2正则的，可以发现两者区别不是很大。而实际在使用test_model.py检验时发现右图的准确率比左图高了一个百分点，因此正则化对于防止过拟合、提高准确率是有帮助的。



3. 增加dropout层

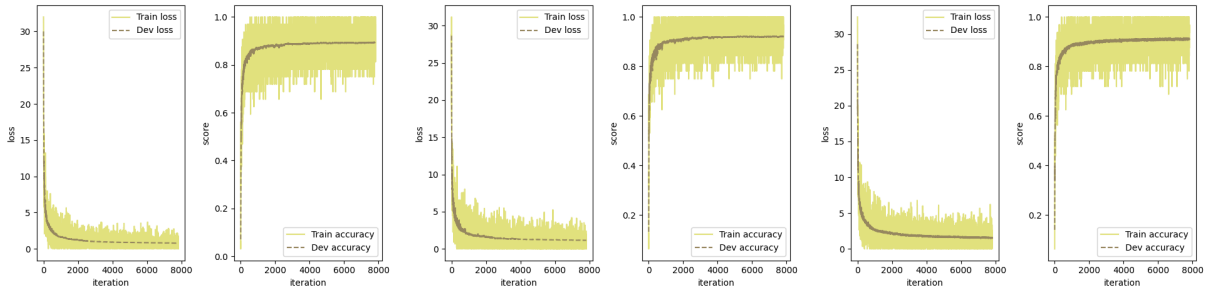
以下两图中左图为未加dropout层的训练过程，中图是加了drop率为0.2的dropout层，右图是加了drop率为0.5的。可以发现加了dropout层后准确率收敛速度变慢了，这是因为每次训练并不是所有节点都接受到了训练，需要相对较长的时间来完成训练。加了dropout层后准确率有所降低是因为训练时用的是部分节点，而最后使用模型的时候是所有节点，会有出入。同时可以发现加了dropout层后的loss收敛是更快的，且变化幅度更小，说明了dropout确实起到

一定防止过拟合的作用。



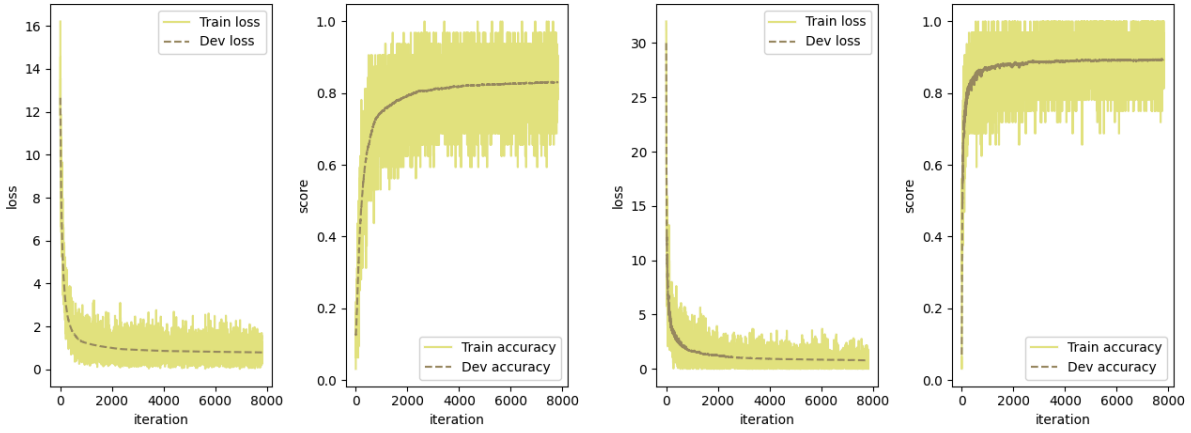
4. 改变隐藏层的单元数

以下从左往右分别是隐藏单元为128、256、512的mlp网络，他们的准确度分别是0.9042、0.9201、0.9382，可以发现在相同隐藏层个数下隐藏单元的增加可以提高模型的表现，推测原因可能是隐藏单元起到类似于降维的作用，隐藏单元过小会忽略掉一些特征，进而导致模型准确率有所降低。



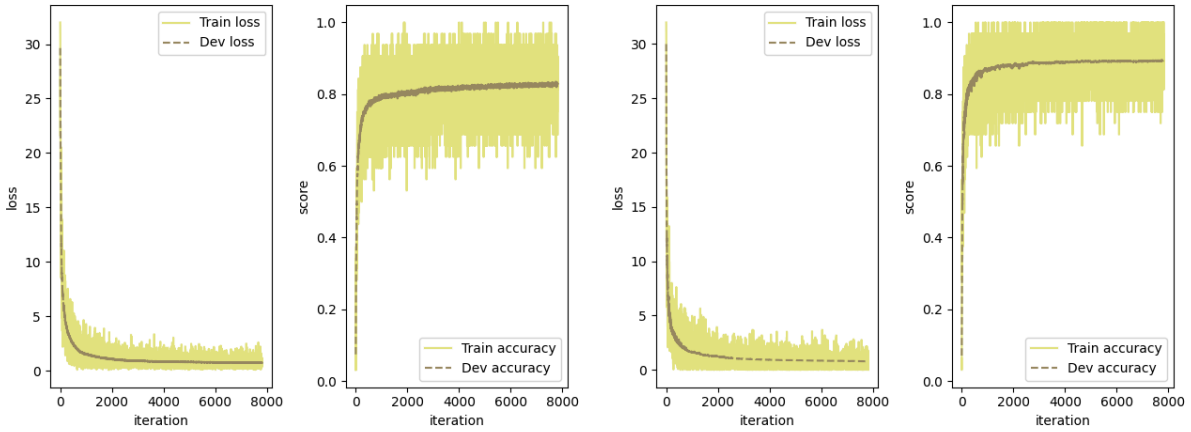
5. 改变隐藏层的个数

以下从左往右分别是单层线性层（768->10）与双层线性层（768->128->10）组成的mlp网络，可以发现双层线性层可以拥有更好准确率且loss下降更快、准确率收敛更快，这是因为双层线性层中先将一些像素整合成特征再将用特征去预测，而单层只是将像素用于预测、更为笨拙。同时注意到使用单层线性层训练速度很快，这是由于参数少导致。



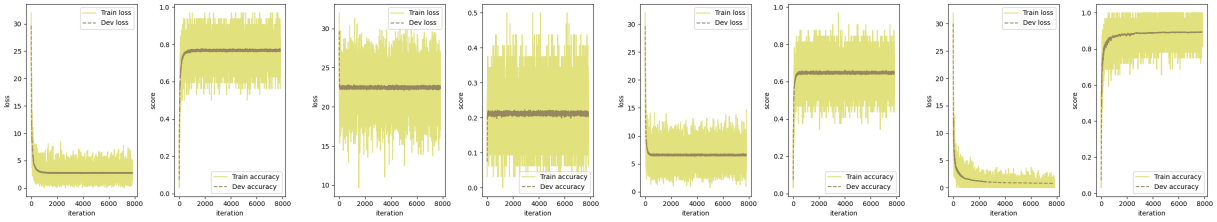
6. 不同优化函数

以下从左往右分别是相同条件下momentGD与SGD优化方法的训练过程，其中左边的准确率为0.87，而右边的为0.90。导致这个的原因是momentGD引入动量，因此在收敛到最优点的过程中会有前面的移动的影响，准确率略有下降。



7. 不同的learning rate schedule

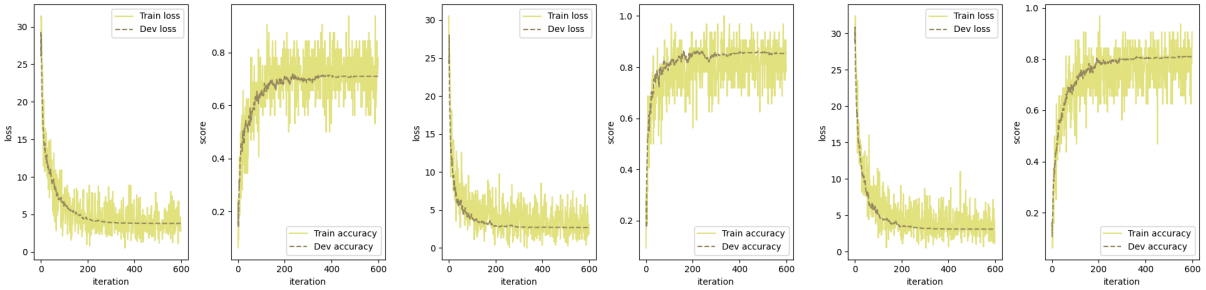
以下从左往右分别是StepLR、gamma=0.9的ExpLR、gamma=0.99的ExpLR、MultistepLR。可以发现StepLR与ExpLR的效果差强人意，这是因为他们并没有关注到loss下降的趋势来调整。而MultistepLR可以让我们根据loss有没有在震荡、无法继续下降来有针对地降低learning rate，缺点是需要多次调整，每次确定一个Milestone后要从头继续训练。



五、CNN模型优化

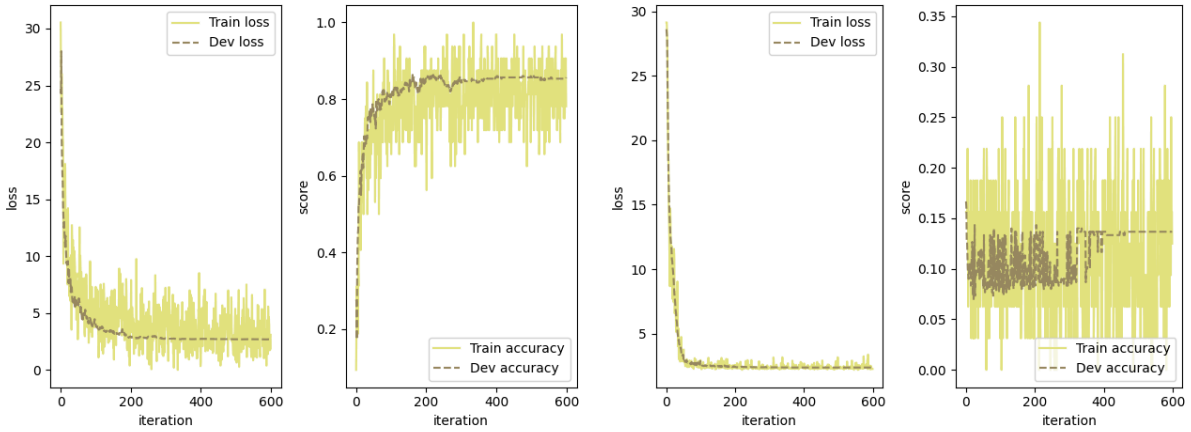
1. 改变卷积的输出通道数

以下三图分别是输出通道为6,8,12的训练过程，他们的准确度分别为0.73、0.82、0.80，可以确定使用8个输出通道最为合适。



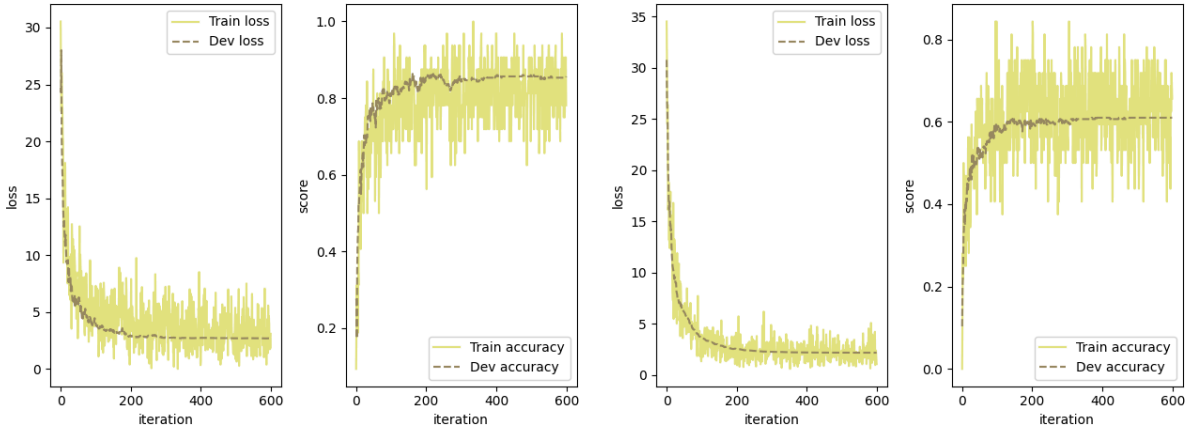
2. 改变模型线性层隐藏单元的数量

以下分别是CNN网络中线性层隐藏单元为128和256的训练过程，其中隐藏单元为128的表现更好，推测原因是隐藏单元数为128更接近预测需要的特征数量，256则增加了许多无用的特征导致过拟合，这也可以从256对应的loss值较小且随iteration波动不大发现。



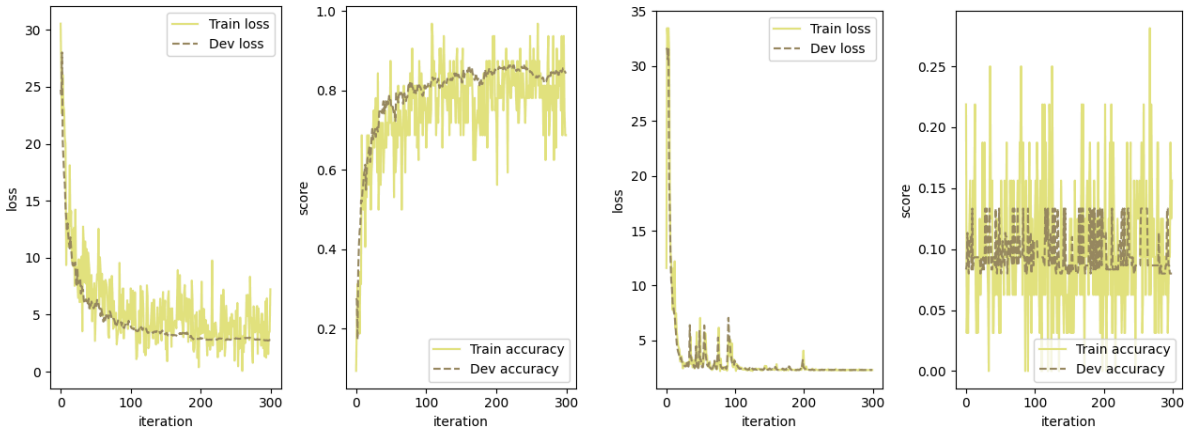
3. 改变池化层的大小

以下从左到右分别是池化层大小为2与大小为3的过程比较，可以发现池化层大小为2更好，推测原因是本身图像是28*28，经过卷积后大小更小了，这时候用太大的池化反而会消除图像必要的特征，导致难以去判断数字。同时，我们从池化层大小为3的dev loss的变化幅度不大可以更清晰地了解池化可以增加稳定性（平移不变性）。



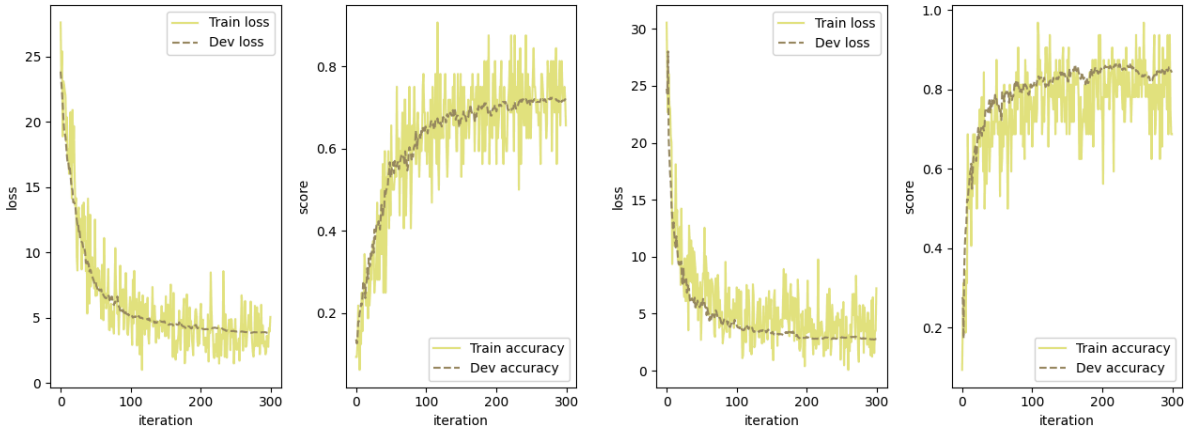
4. 改变卷积层的个数

以下分别是单层卷积层与双层卷积层的训练过程，同样也可以发现双层卷积层的dev loss变化率过小，这说明产生了过拟合，准确率远不如单层卷积层。



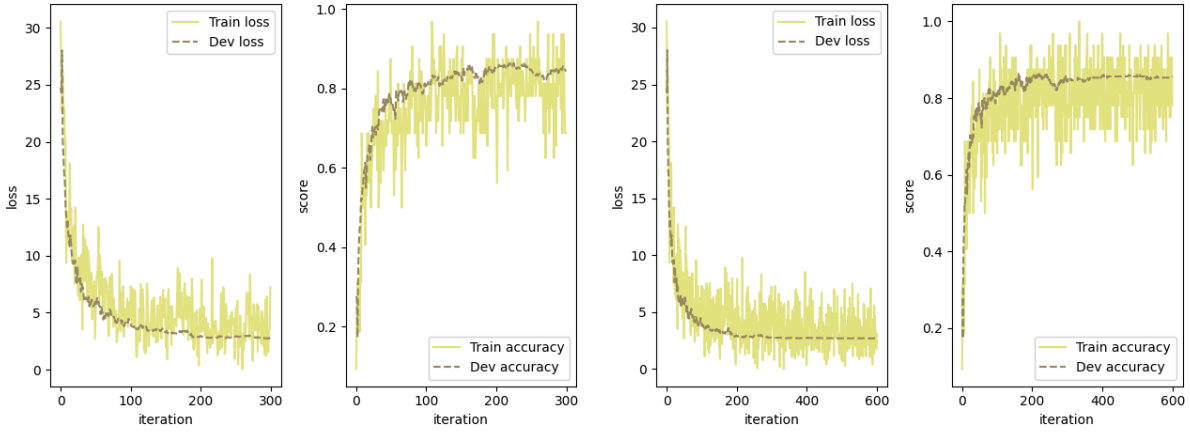
5. 改变线性层层数

以下分别是单层线性层与双层线性层的训练过程，可以发现双层线性层的效果更好，这是因为它有效地将卷积层的输出整合到更有辨别力的特征，而不是直接将所有卷积层的输出直接用于判断。



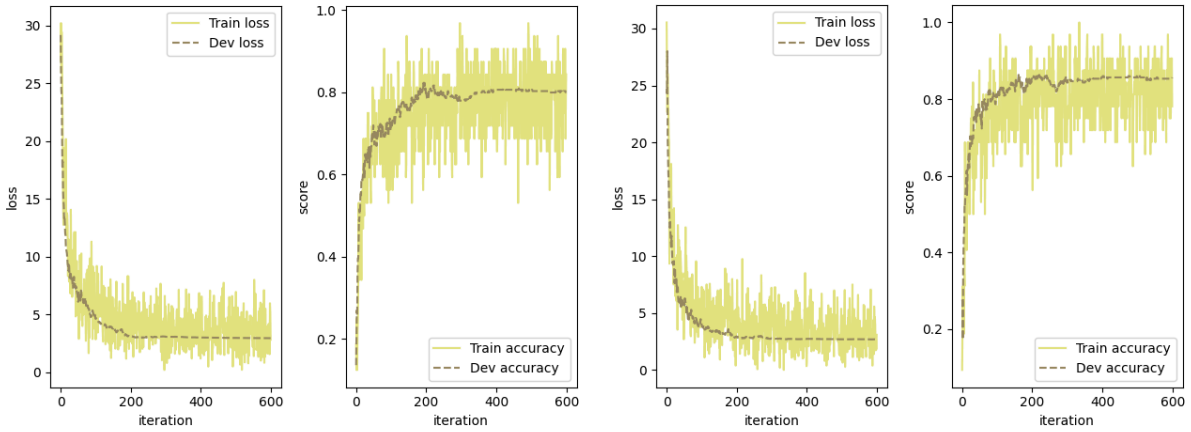
6. 改变epoch次数

以下分别是epoch为1与epoch为2的过程对比，其中epoch为2最后的结果完成了收敛但只比epoch为1的准确度高了一点点，效果一般，但是时间消耗确实epoch为1的两倍，不太值得。



7. 改变训练集

以下分别是相同训练次数下只从300个数据集里抽取数据训练与从50000个数据集里抽取数据训练的比较。可以发现从50000个数据集中抽取数据训练的效果更好，推测原因是50000个中数据的变异性更大，让模型可以更好地理解、提取数字真正的特征。



六、 整体优化

1. 充分利用numpy在矩阵计算的快速性

如卷积神经网络在卷积层计算正向传播与反向传播的输出时，原本需要4个for循环遍历W的所有维度，但是这里采取矩阵运算的方式即可以只使用三个维度就可以完成运算，实验中极大提高了程序运行效率。

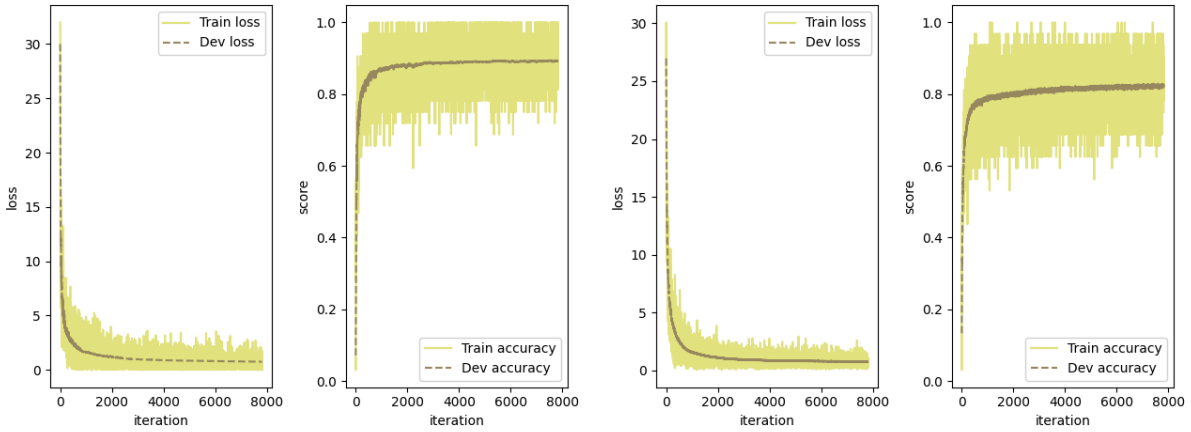
```
for i in range(out_H):
    for j in range(out_W):
        h_start = i * self.stride
        h_end = h_start + self.kernel_size
        w_start = j * self.stride
        w_end = w_start + self.kernel_size

        # Extract the current window
        window = X_padded[:, :, h_start:h_end, w_start:w_end]

        # Compute gradients for weights and bias
        for k in range(self.out_channels):
            # Gradient for weights
            dW[k] += np.sum(window * grads[:, k, i, j].reshape(-1, 1, 1, 1), axis=0)

            # Gradient for bias
            db[k] += np.sum(grads[:, k, i, j])

            # Gradient for input
            dX_padded[:, :, h_start:h_end, w_start:w_end] += self.params['W'][k] * grads[:, k, i, j].reshape(-1, 1, 1, 1)
```

2. 避免数据溢出

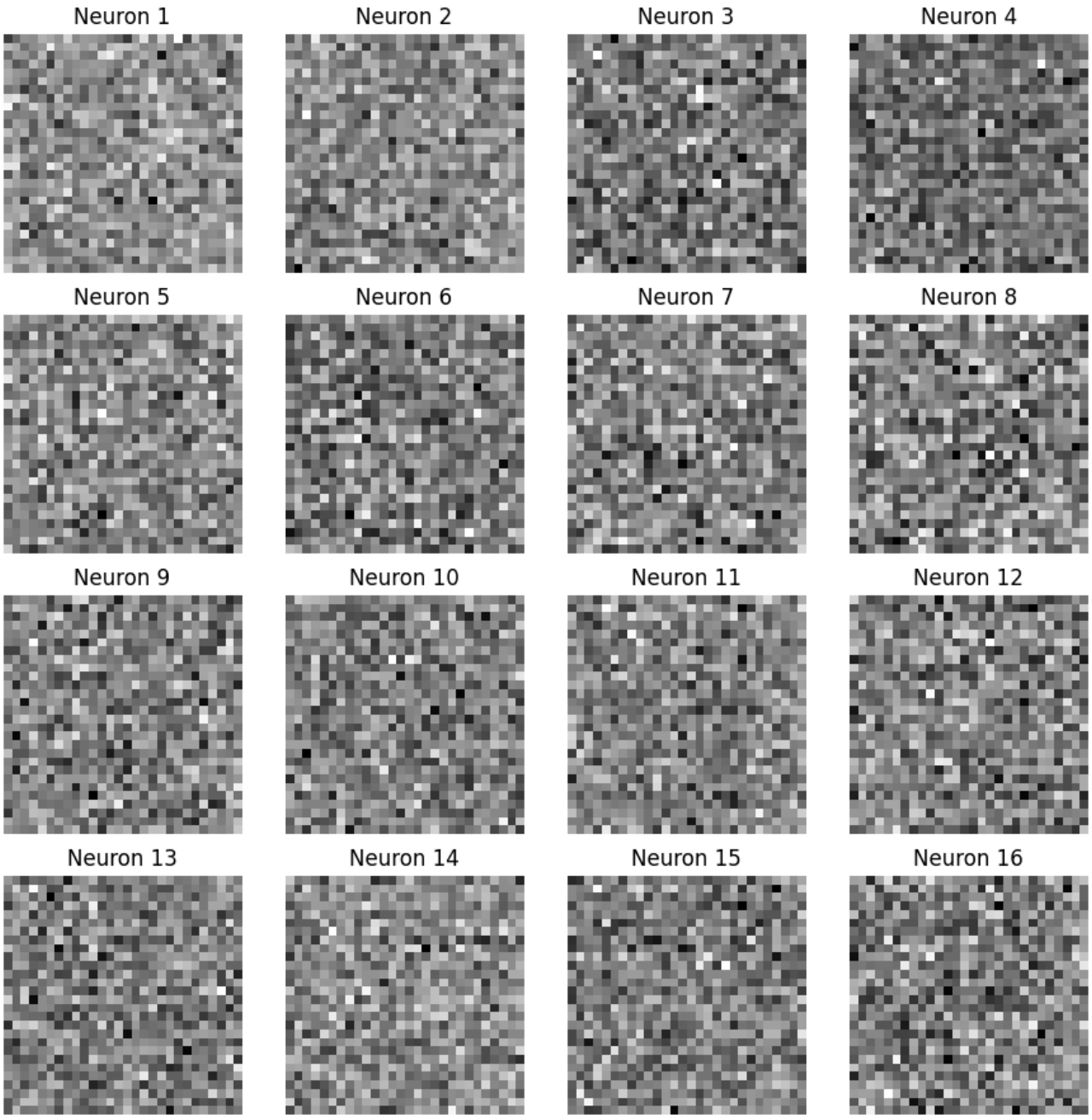
交叉损失层在计算损失时会用到np.log，这时候log里面为0的话会导致损失无穷大，无法继续接下来的计算。因此这里人工加入了极小值1e-15，避免了这种情况。

七、 数据增强

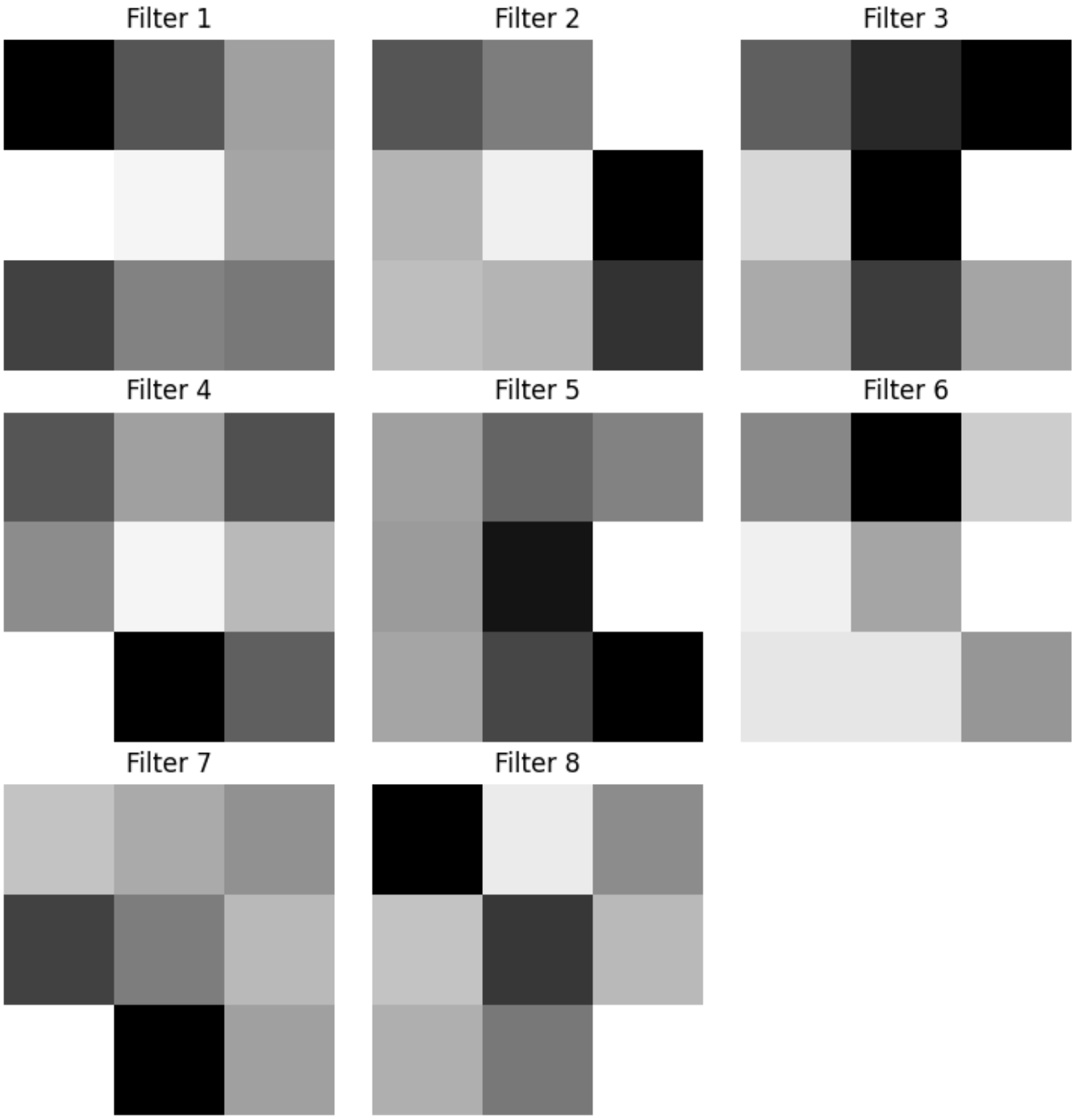
以下分别是使用原始数据和对mlp数据进行数据增强后（平移旋转等）训练的过程，可以发现数据增强导致训练的准确度有些许下降，这是因为旋转等操作可能导致数字label错误（比如6旋转后变成9）。但是模型最后准确度依然很高，这说明数据增强的确可以作为我们缺少数据时扩充数据的一个手段。



八、 权重可视化



以上是MLP的线性层的权重矩阵的部分，可以发现每个权重矩阵都提取了全图各个位置的综合信息用来判断当前数字，



以上是CNN使用的卷积核，可以发现每个卷积核都最关注窗口的一到二个位置（某一到两个像素块特别白还在特别黑，实际是由于两个值一个是最大值，一个是最小值），且可以发现最黑与最白的像素块往往相邻，由于输出kernel后发现最大值一般是正值而最小值为负值，因此黑白色块邻接就是在算这边的梯度，作为特征提供给线性层继续判断。

九、总结

根据多次优化实验后最后得到的最优模型参数已分别保存在提交的程序中，MLP达到了92%的准确率而CNN达到了82%的准确率。