构建两层神经网络分类器

毛柯宇

April 4, 2022

1 简介

根据作业的要求，本次实验将仅通过 numpy 作为工具，从头开始构建一个两层的神经网络。实验中一共涉及到三个代码文件，分别将实现：

1. "m_nn.py": 神经网络类的构建。在这个类中我们将主要实现：神经网络的参数初始化，损失函数与梯度的计算，训练过程（训练过程会包括 SGD、学习率下降等）、预测与模型的保存。
2. "find_para.ipynb": 文件用于寻找最优的学习率、隐藏层大小与正则化强度的组合。我们将使用交叉验证的方法，分别取一系列的值进行比较。我们会将比较的过程可视化，并将最后选取的最佳模型的权重参数可视化并将最佳模型保存。
3. "load_and_test": 载入最佳模型，直接用最佳模型进行测试并输出最终的测试精度。

相应文件对应的链接：

1. github repo 链接：

   https://github.com/mskmei/-construct-a-2-layer-neural-network-by-numpy.git

2. 模型网盘下载地址：

   https://drive.google.com/file/d/17uzZ3Ol1OWh_3fgQH37WAYOutVTdMCRB/view?usp=sharing

2 原理

2.1 网络结构与前向传播

本次实验中我们手动构建的为 bp 神经网络，基于反向传播实现的神经网络，网络为两层的神经网络。简单来讲网络的结构可以表述为: 输入层、隐藏层与最终输出。大致的示意图如下：

在具体的实验上，我们先实现前项传播。我们先将大小为 n 个 (x,x) 的图片做 reshape 变为 (n,x*x) 的向量（或者 (x*x,n)），这里的 n 可以是我们自己设定的 batch size。进而我们用 x 乘以大小为 (x*x,hidden layer) 大小的权重矩阵 w1 并加上偏置值 b1, 我们对上述值使用第一层的激活函数 ReLu 进行处理得到隐藏层的向量 h, 大小为 (n, hidden layer)。这一部分的表达式可以写作：$h = ReLu(x * w1 + b1)$。其中 ReLu 函数的函数式为：

$$ReLu(x) = \max(x, a)$$

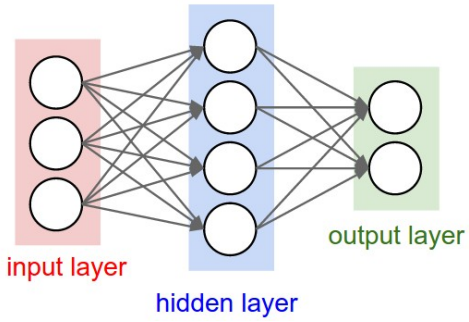


Figure 1: neural network

进一步，我们在利用大小为 (hidden layer, output class) 的权重矩阵 w2，以及偏置值 b2，最后得到 $output = h * w2 + b2$ 作为输出。由于我们做的是多分类问题，我们希望输出关于不同类别的预测概率，因此，我们使用了函数 softmax 作为工具，将 output 转换为一个概率矩阵，每一行为一张图片经过前向传播与 softmax 处理后的向量，而每一列则是这张图片对应每一个类别的概率（这与大多数习惯不同，因为我们将图片 reshape 后的向量做成了行而不是列）。

我们不妨假设 output 第 i 行为 $(z_1, z_2, \cdots, z_m)$,softmax 的公式为：

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}}$$

这样一来，每一行的和都为 1，且每一列的的值都相应地转化我对应类别的概率。在实际的运算中，为了防止 output 的值过大造成的上溢出，我们选择利用 softmax 函数的一个性质：

$$softmax(x) = softmax(x + c)$$

让所有的行都减去该行最大的一个值，不会影响最后的输出，同时这样的行为也防止了上溢出的发生。

2.2 交叉熵损失函数与 L2 正则化

在本次实验中，我们使用交叉熵作为我们的损失函数，使用交叉熵作为损失函数在进行梯度下降计算的时候可以避免出现梯度消失，是多分类问题中很常见的损失函数。在多分类问题中，交叉熵函数的函数表达式可以写作：

$$L = \frac{1}{N}\sum_i L_i = -\frac{1}{N}\sum_i \sum_{j=1}^{m} y_{ij} \log(p_{ic})$$

上式中，N 为我们处理的图片数量（或是自行设置的 batch size），m 为类别的数量，y_{ij} 表示样本 i 的真实类别为 j 时取 1，其余时候均取 0，p_{ij} 表示预测 i 属于 j 类的概率，这个概率将由 softmax 处理后的概率矩阵得到。交叉熵的示意图如下：

为了有效地防止过拟合，我们打算对损失函数添加惩罚项。在本次实验中我们采用了 L2 正则化，处理之后的损失函数表达式可以表达为：

$$L(w1, w2, b1, b2) = -\frac{1}{N}\sum_i \sum_{j=1}^{m} y_{ij} \log(p_{ic}) + \frac{\lambda}{2}||w1||_F^2 + \frac{\lambda}{2}||w2||_F^2$$

其中，λ 是我们可以进行调整的正则化强度，$||\cdot||_F$ 是矩阵的 frobenius 范数，再对其进行平方操作即可得到矩阵所有元素的平方和。至此我们完成了损失函数与 L2 正则化的实现。

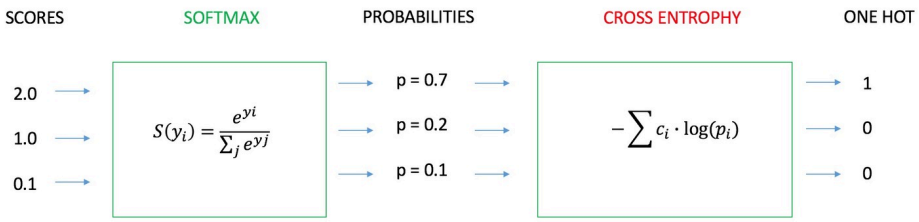


Figure 2: softmax 结合交叉熵示意图

2.3 反向传播

在 bp 神经网络中，反向传播是实现权重矩阵梯度下降的重要一环。我们希望通过反向传播求得损失函数对 w1,w2，b1，b2 的导数，从而可以结合学习率进行梯度下降。

首先对不含正则项的损失函数，我们有一个重要的结论。首先我们设 Z 为未经过 softmax 的输出矩阵，A 为经过了 softmax 的输出矩阵，L 为不含正则项的损失函数。再定义标签矩阵 $Y := y_i j$，yij 表示样本 i 的真实类别为 j 时取 1，其余时候均取 0。我们可以求得:

$$\frac{\partial L}{\partial Z} = \frac{1}{N}(A - Y)$$

有了这个结论之后我们可以很容易地进行反向求导运算（注意添加 L2 正则项），我们不妨将 $\frac{\partial L}{\partial Z}$ 的结果定义为矩阵 P, 将最初的输入矩阵定义为 X，求得:

$$dw_2 = h^T * P + \lambda w_2$$

$$dh = P * w_2^T \& dh[where \quad h < 0] = 0$$

$$dw1 = X^T * dh + \lambda w_1$$

除此以外，db1 与 db2 分别为应该为 dh 与 P，但是我们还需要利用求和工具是其矩阵的形状与本身的形状一致。上述的具体实现过程请见文件 “m_nn.py”

2.4 随机梯度下降与学习率下降策略

随机梯度下降中，我们每次选取 batch size 个数据进行处理，也就是选取 batch size 行的数据进行处理，选取是随机可重复的。对选取的数据，我们通过上述过程求得的梯度向量进行梯度下降，比如 $w_2 new = w_2 - learning \quad rate * dw2$。从而实现了随机梯度下降。每当执行完一个 epoch，我们利用一个固定参数乘以学习率，实现学习率下降策略。

3 参数设定与交叉验证

在实验中，隐藏层大小，学习率与正则化强度为三个最为重要的参数，需要我们进行谨慎的设置。我们分别设置了三个列表装载这三个参数的数个取值，然后使用三个 for 循环实现交叉验证，每一个参数对对应的模型均使用训练—验证集数据训练 20 个 epoch，按照验证过程的精确度大小进行排序，并展示此过程。参数取值的具体取值请见文件”find_para.ipynb”, 最终我们选取的参数为 hidden size = 200, learning rate = 0.001, $\lambda = 0.4$。我们仅仅展示出上述交叉验证输出结果的前五组与最后五组的对应的参数值与最终算得的验证过程精确度:

```
hidden layer's size:50, learning rate:0.010000, regularization:1.000000, validation accuracy:0.106500
hidden layer's size:100, learning rate:0.010000, regularization:0.400000, validation accuracy:0.106500
hidden layer's size:50, learning rate:0.010000, regularization:0.200000, validation accuracy:0.107000
hidden layer's size:50, learning rate:0.010000, regularization:0.600000, validation accuracy:0.107000
hidden layer's size:50, learning rate:0.010000, regularization:0.800000, validation accuracy:0.107000
```

Figure 3: 精确度后五组

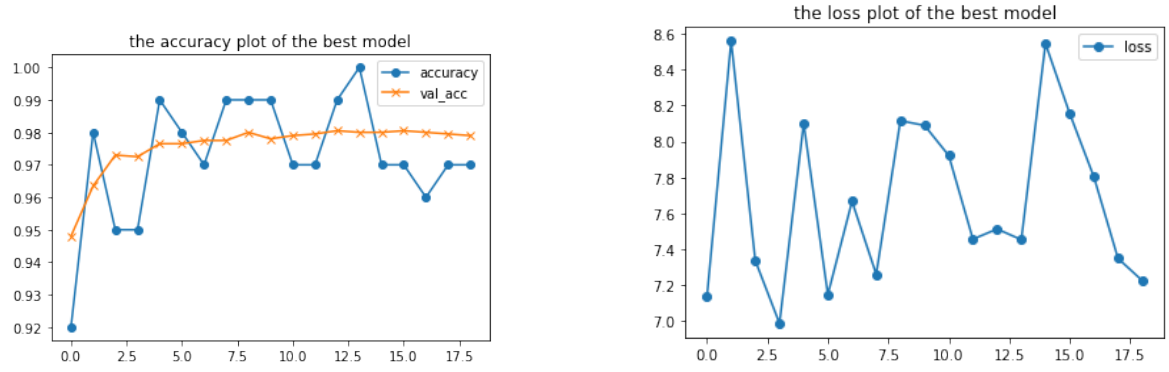
```
hidden layer's size:150, learning rate:0.001000, regularization:0.400000, validation accuracy:0.980500
hidden layer's size:200, learning rate:0.001000, regularization:0.200000, validation accuracy:0.980500
hidden layer's size:250, learning rate:0.001000, regularization:0.200000, validation accuracy:0.980500
hidden layer's size:250, learning rate:0.001000, regularization:0.600000, validation accuracy:0.980500
hidden layer's size:200, learning rate:0.001000, regularization:0.400000, validation accuracy:0.981000
```

Figure 4: 精确度前五组

4 可视化

4.1 最优模型训练与验证过程的可视化

训练与验证过程的精确度与损失可视化：



(a) accuracy (b) loss

Figure 5: 训练与验证过程的精确度与验证过程的损失

可以发现，虽然训练过程的精确度并不稳定，但是验证过程的精确度成非常稳定地上升状态。由于验证过程的数据并没有被模型学习，说明模型在训练的过程中拟合的效果渐入佳境。即使训练过程的精确度不稳定，其数值基本稳定在百分之九十五以上，而且有比较明显的上升趋势。也说明了这个模型拟合效果较好（当然最终的判定需要交给测试集）

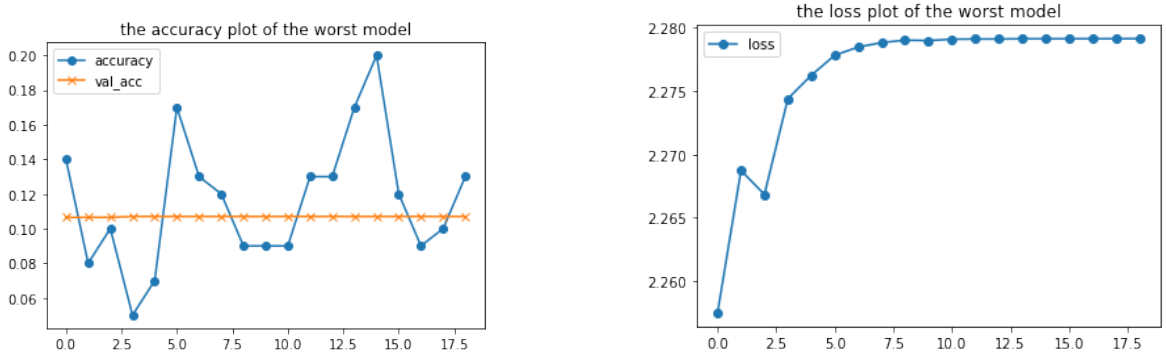
而 loss 虽然波动较大，但是整体上呈现下降趋势，且波动范围并不大。在 loss 方面的优化还有待进一步的探究。为了更直观的说明这个最优模型的优势我们除了最优模型还将展示最差模型与中等的一个模型的训练与验证过程的结果。

4.2 对比试验 1：最差模型训练与验证过程的可视化

最差模型的参数：隐藏层大小：50，学习率：0.01, 正则化强度：1

最差模型的训练与验证过程的精确度与损失可视化：

首先观测训练过程。可以发现随着训练过程的进行，训练过程的精确度始终在 0.1 的附近上下浮动，说明训练的效果极差，拟合效果不明显。与之相比，最优模型的波动虽然存在，但是最



(a) accuracy (b) loss

Figure 6: **最差模型**训练与验证过程的精确度与验证过程的损失

低不会低于 0.95，且基本稳定在 0.97 附近，说明这两者之间有很大的差异，这可能是由于学习率设置过高造成的。

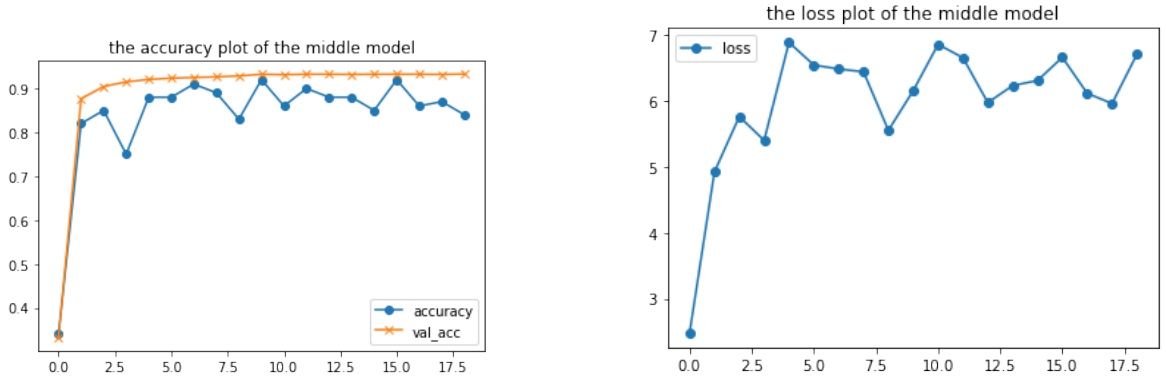
再观测验证过程。首先验证过程的准确度为一条水平实验，说明验证集的正确率从来就没有任何提高，我们认为很有可能出现了过拟合现象，这可能和正则化强度的设置有关。而验证过程的 loss 不降反增大，相比之下最优模型虽然略有浮动，但是总体上没有如此明显的上升趋势。这样 loss 递增趋势进一步说明了这个模型的不合理。

在下一个模型中我们将更直观地看到正则化强度与学习率对模型的显著影响。

4.3 对比试验 2：一个中等精确度模型训练与验证过程的可视化

中间模型的参数：隐藏层大小：50，学习率：0.0001，正则化强度：0.4

中间模型的训练与验证过程的精确度与损失可视化：



(a) accuracy (b) loss

Figure 7: **中间模型**训练与验证过程的精确度与验证过程的损失

首先与最优模型相比，中建模型已经初具雏形。其训练过程与验证过程的的精确度已经呈现稳步上升趋势，loss 虽然在开头有一个突增，但是在之后逐渐稳定。不足之处也十分明显，比如训练与验证的精确度只能在 0.9 处徘徊，难以进一步上升。

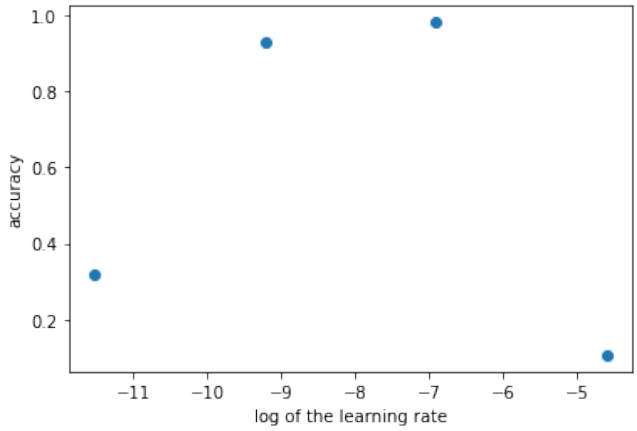
当我们将其与最差模型相比时，我们很容易发现其参数与最差模型相比只改变了学习率与正

则化强度两个参数。学习与正则化强度的下调非常明显地优化了模型。我们不难验证我们在最差模型的分析做出的关于训练过程与验证过程异常的假说。

经过两个模型的分析，我们进一步肯定了学习率与正则化强度过大对模型造成的巨大负面影响，并肯定了我们最优模型的拟合能力。

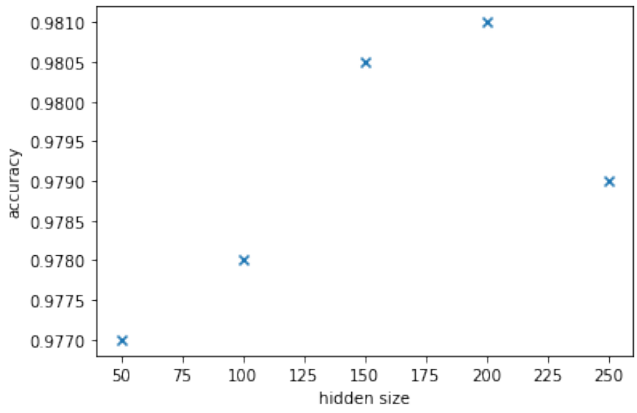
4.4 学习率，隐藏层与正则化强度的比较研究

首先我们决定一定程度上基于最优参数对交叉验证过程进行可视化。对于学习率，正则化强度与隐藏层大小三个参数，我们分别控制两者为上一步中选出的最优参数组合的数值，可视化剩余一个参数变化对于验证过程准确度的影响。图像经过散点图进行呈现。由于时间等因素我们是测试了较少量的参数（但是参数范围的选取均基于了前人经验），所以只能一定程度反映各个参数与验证过程精确度的关系。首先探讨学习率与验证过程准确度的关系。因为学习率取值具有不同的数量级，为了横坐标分布均匀，我们选取了 log(learning rate) 作为横坐标：



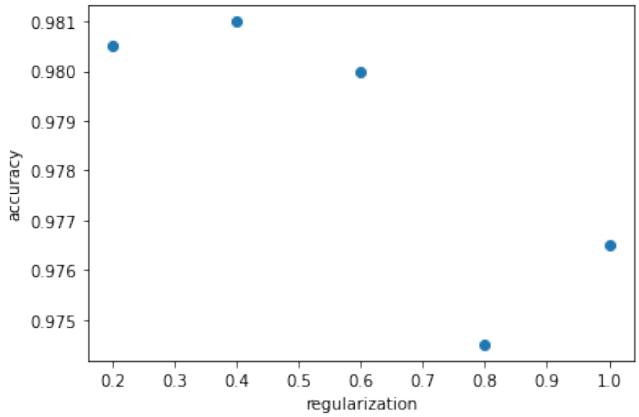
学习率从 1e-5 增加到到 1e-2，验证过程精确度呈现大致先增后减，大致在 0.001 处取得极大值（进一步可以设置梯度进行探究）。学习率在过小或者过大时验证过程精确度都呈现了**非常明显**地下滑，说明学习率的取值应当适中在 0.0001 与 0.001 之间。

其次是隐藏层大小与验证过程精确度：



与学习率相似，随着隐藏层数增大，验证过程精确度也呈现了先增后减的趋势，并大致在 200 层时取得精确度极大值。隐藏层的大小同样也不宜过大过小，合适的范围大致在区间（175,225）之间。但是，隐藏层的改变对精确度的影响远远小于学习率的影响（在学习率合适的条件下）。

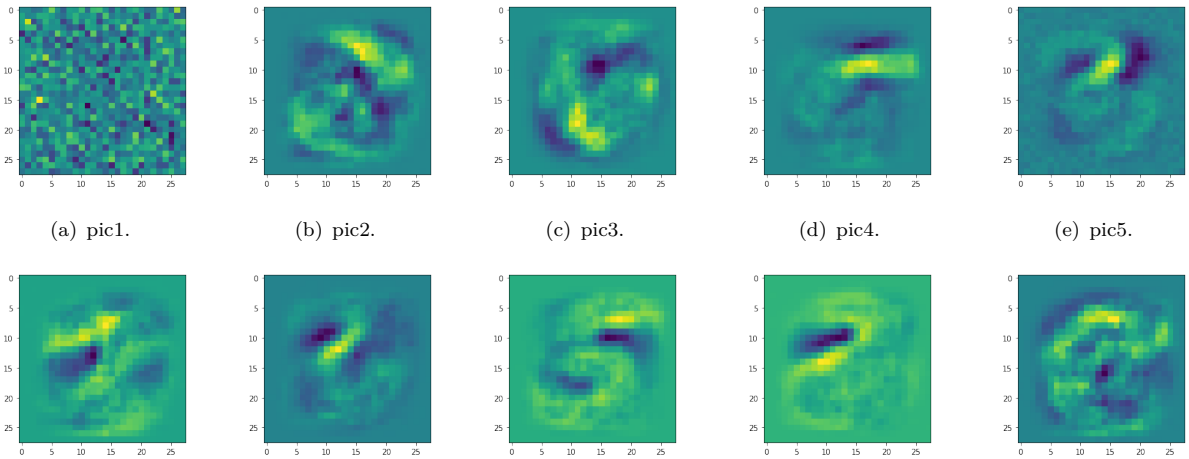
最后是正则化强度与验证过程精确度：



随着正则化强度的变化，精确度也出现一定的浮动。不过在学习率与隐藏层层数合适的条件下其波动幅度很小，以至于最小并没有跌出 0.97。不过我们仍然选取较优者（即正则化强度为 0.4 时）作为最优模型的选项。

4.5 最优模型权重矩阵的可视化

除此之外，我们考虑展示每一层的权重矩阵参数。对于第一层，我们可以将 w1 的重新 reshape 为容易展示的形式，我们使用的数据集为 mnist，图片大小为 28*28，于是我们可以将 w1 的第一个维度重新 reshape 为（28,28）的矩阵形式，进而我们分别可视化其前 10 层的图像（迫于空间原因只展示前 10 层）：

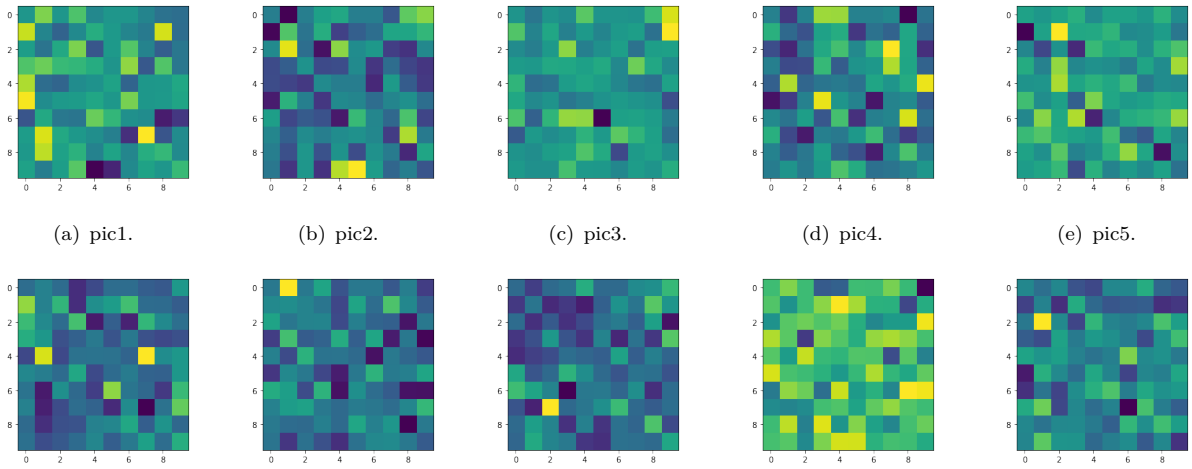


(a) pic1. (b) pic2. (c) pic3. (d) pic4. (e) pic5.

(f) pic6. (g) pic7. (h) pic8. (i) pic9. (j) pic10.

Figure 8: 前十层权重矩阵 w1 参数可视化

对于 w2，由于本实验中 w2 形状为 (200,10)，不能通过上述过程进行展示，也无法降维构造 rgb 形式的图片。因此我们选择将其 reshape 为较为规范的形式 (100,20) 然后再进行上述方法的可视化（这是因为我们选择层数是 10 的倍数，而 output 正好是 10，则一定可以被 100 整除），每一层 reshape 为 (10,10)，打印前 10 层：



(a) pic1. (b) pic2. (c) pic3. (d) pic4. (e) pic5.

(f) pic6. (g) pic7. (h) pic8. (i) pic9. (j) pic10.

Figure 9: 前十层权重矩阵 w2 参数可视化

5 测试结果与结论

最后我们在 “load_and_test.ipynb” 文件中，加载我们训练好的模型，直接用来对测试集（测试集独立于训练集与验证集）进行测试，得到最终的测试准确度：0.9659999999999099。

测试过程的准确度显然是低于训练过程的，这是因为在训练与验证过程中，我们通过训练集对参数进行调整，并且通过验证过程的准确度选择模型为作为我们的最优模型。然而测试集数据与训练与验证集均没有关系，即测试集数据对我们的训练的内容与选取的标准而言是完全陌生的，从理论上来讲一般测试的精确度会一定程度低于训练与验证过程的精确度。

但是其值距离训练与验证过程的精确度不超过百分之二，且达到了百分之九十五以上，这足以说明该模型没有出现明显的过拟合现象，并且可以认为我们手动构建的神经网络模型在取相对最优参数时的预测较为准确！