**Project 1**

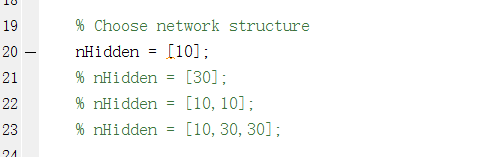
赵心怡 19307110452

2022/4/7

**Task1： Change the network structure: the vector nHidden specifies the number of hidden units in each layer.**

nHidden 表示了每个layer的hidden unit。 ‘nHidden = [n]’表示了单层的神经网络， [n1,n2,n3]代表了三层的神经网络，以此类推。

nhidden的长度代表了神经网络的深度，因此我们可以通过修改这个参数来改变网络的结构。



|  |  |
| --- | --- |
| n-hidden=[10]  test-error=0.513000 | n-hidden=[30]  test error = **0.313000** |
|  |  |
| n-hidden=[10,10]  test error: 0.59300 | n-hidden = [10,30,30]  test error=0.48900 |
|  |  |

从结果上可以看出n-hidden 单层的神经元增加会减少test error，使模型更加准确，运算时间成比例增加。

层数layer的增加不一定能让test error降低，同时它的运行速度减慢，因此当单层的神经元为30的时候模型表现最好。

##### Task2: Change the training procedure by modifying the sequence of step-sizes or using different step-sizes for different variables.

##### 

step-size的大小可以影响决定什么时候模型函数能达到局部收敛已经什么时候停止。

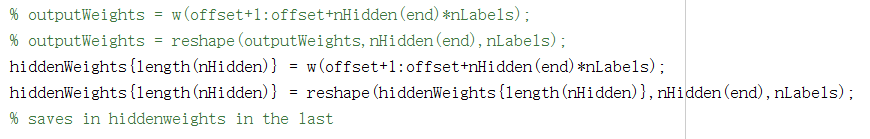
|  |  |
| --- | --- |
| step-size = 1e-4  test-error=0.542000 | step-size=1e-3  test-error = 0.514000 |
|  |  |
| step-size =1e-2  test-error = **0.397000** | step-size=1e-1  test-error = 0.943000 |
|  |  |

从图中可以发现当stepsize设置过大会导致走的过快而不收敛，而步长过小会导致收敛速度很慢，由上结果可以看出这四个取值中的最优的梯度是1e-2

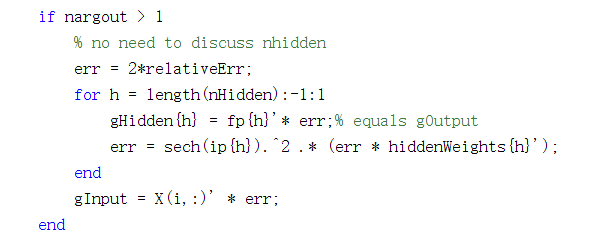
##### Task3: You could vectorize evaluating the loss function (e.g., try to express as much as possible in terms of matrix operations), to allow you to do more training iterations in a reasonable amount of time.

我们用矩阵的形式来尽可能表示损失函数。主要修改的地方如下

1. 删了outputweights, 存在 hiddenweights最后



1. 修改了gInput的表示，在这里不再需要讨论nHidden的情况。删除了gOutput, 存在gHidden的最后



测试了在nHidden=[30], step-size=1e-2的情况下前后两种函数的耗时:

|  |  |
| --- | --- |
| Original time | 13.840808s |
| Matrix time | **6.725201s** |

在nHidden=[10, 10], step-size=1e-2的情况下：

|  |  |
| --- | --- |
| Original time | 57.829088s |
| Matrix time | **6.653249** |

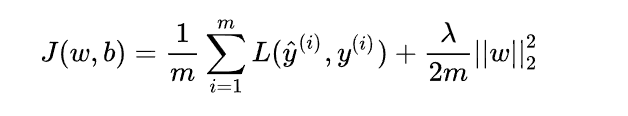
从结果上可以看出，将函数转换成矩阵形式可以显著加快运算速度，当层数多的时候矩阵函数的优化效果更明显。

因为在nhidden=[30]的时候省的时间是2倍，故将迭代次数改为20 0000次

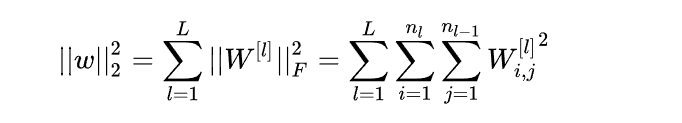
##### Task4: Add l2 regularization (or l1-regularization) of the weights to your loss function. For neural networks this is called weight decay. An alternate form of regularization that is sometimes used is early stopping, which is stopping training when the error on a validation set stops decreasing.

high variance也就是overfitting的时候，我们可以使用regularization来处理。

正则化函数可以表示为：

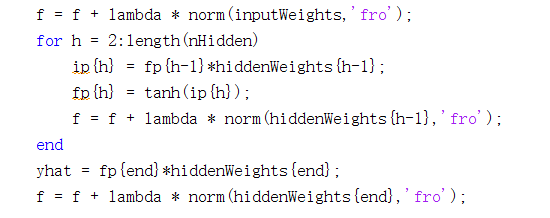


L2正则损失部分是神经网络中每一层的权重矩阵 的Frobenius范数的平方和，假设第l层的权重矩阵维度是 ，计算结果如图：



我们可以通过修改的值检验随着的变化测试误差能否变小。

修改部分代码：





不同lambda下的测试结果：

|  |  |
| --- | --- |
| λ | Test error |
| 1e-4 | 0.150000 |
| 1e-3 | **0.082000** |
| 1e-2 | 0.111000 |
| 1e-1 | 0.315000 |

从结果看出过小正则化效果不明显，过大模型可能比较简单，有欠拟合的风险。在的时候模型表现最好，因此我们选择1e-3.

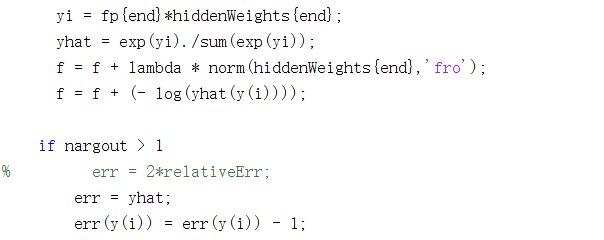
##### Task5: Instead of using the squared error, use a softmax (multinomial logistic) layer at the end of the network so that the 10 outputs can be interpreted as probabilities of each class.

Softmax函数是：

将损失函数对y求导

修改的部分代码：





然后修改对应的predict中的函数



修改前后的结果比较：

|  |  |  |  |
| --- | --- | --- | --- |
|  | MLP | | Softmax |
| Time | 13.051451 | | 12.370367 |
| TEST error | 0.077000 | | **0.062000** |
| figure | |  |  |

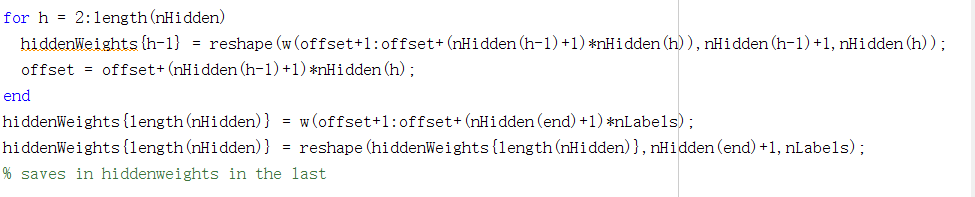
结果中可以看出用softmax函数的结果test error 更小，而且从图像上看下降速度更快。

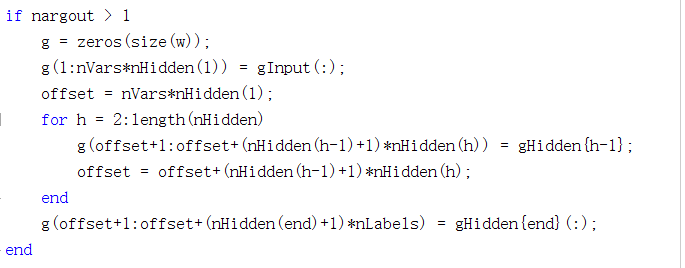
##### Task6: Instead of just having a bias variable at the beginning, make one of the hidden units in each layer a constant, so that each layer has a bias.

在每一个隐藏层中加一个单元作为偏移量。

偏移量参数的更新公式如下:

向隐藏层添加bias的部分代码：





修改前后的结果比较

|  |  |  |  |
| --- | --- | --- | --- |
|  | | without bias | bias |
| TEST error | 0.062000 | | **0.060000** |
| figure |  | |  |

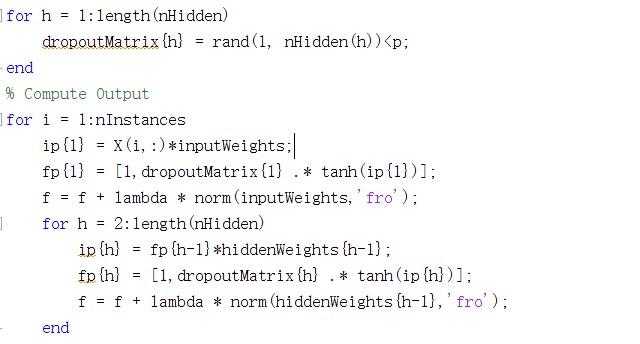
结果可以看出增加bias后test error的变小程度不大，经过多次测试几乎没有影响，我们猜测原因是我们的神经网络只有一层。

在后面的代码中依然保留bias。

##### Task7: Implement “dropout", in which hidden units are dropped out with probability p during training. A common choice is p = 0.5.

dropout 正则化可以减少过度拟合和提高深度神经网络的泛化能力。在训练期间，一些层输出被随机忽略或“丢弃”。 因为 dropout 下的层的输出是随机子采样的，所以在训练期间具有降低容量或细化网络的效果。当使用 dropout 时，可能需要更广泛的网络，例如更多节点。

我们在softmax算法的基础上增加了dropoutMatrix





修改前后的结果比较

|  |  |  |  |
| --- | --- | --- | --- |
|  | | without DROPOUT | DROPOUT |
| TEST error | **0.060000** | | 0.065000 |
| figure |  | |  |

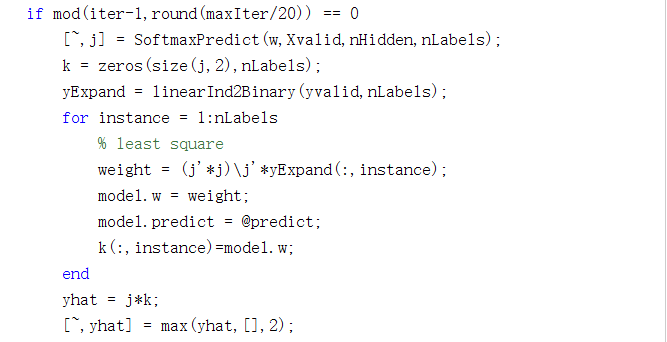
从结果上看dropout可能会对提升test error但是变化不太明显。但dropout可以有效缓解过拟合问题。 为了在本实验中得到最好的测试误差，我们选择不添加dropout。 关于dropout的代码可以在task7文件夹中

##### Task8: You can do ‘fine-tuning’ of the last layer.

在已经拥有训练好的模型的情况下利用微调可以降低训练成本。

确定除最后一层外的所有层的参数，并将最后一层的参数作为凸优化问题求解。

在Sample代码中已经存在线性回归模型linearsquare basis, 我们重新用他的显式解X’\*X\X’Y来简化代码。 添加“微调”的代码如下:

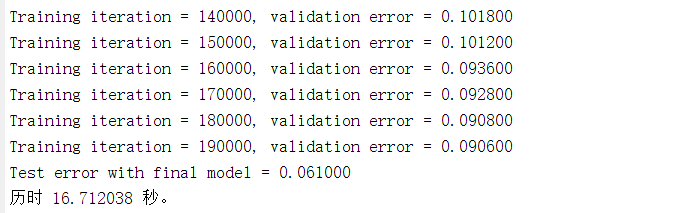


Predict部分



修改前后的结果比较

|  |  |  |  |
| --- | --- | --- | --- |
|  | | without Finetuning | finetuning |
| TEST error | **0.060000** | | 0.061000 |
| figure |  | |  |

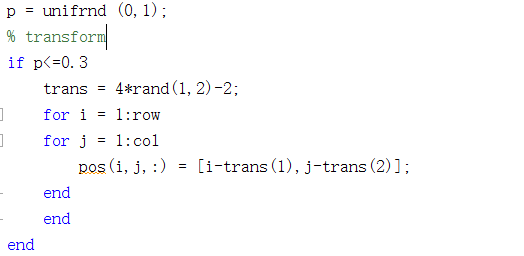


从上面的结果可以看出，虽然添加了“微调”，但测试误差并没有显著降低，甚至在验证中发现会比没有微调的结果要高一些。 然而，通过“微调”可以在实验开始时获得更好的验证误差。 考虑到测试误差没有明显变化，本实验没有使用“微调”。微调的代码保存在task8文件夹中。

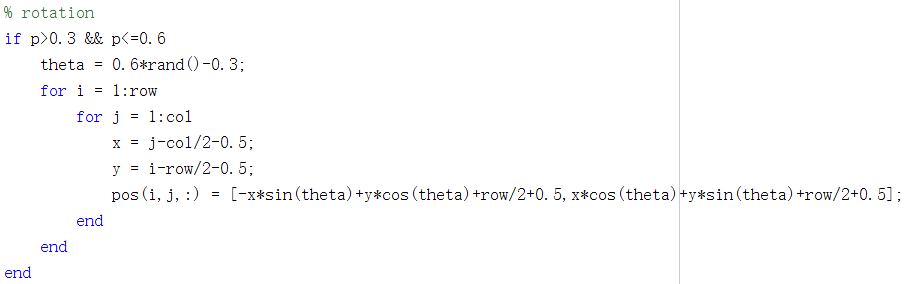
##### Task9: You can artificially create more training examples, by applying small transformations (translations, rotations, resizing, etc.) to the original images.

在Matlab中平移、旋转、翻转图像矩阵的操作都可以通过函数来完成。 具体代码如下:

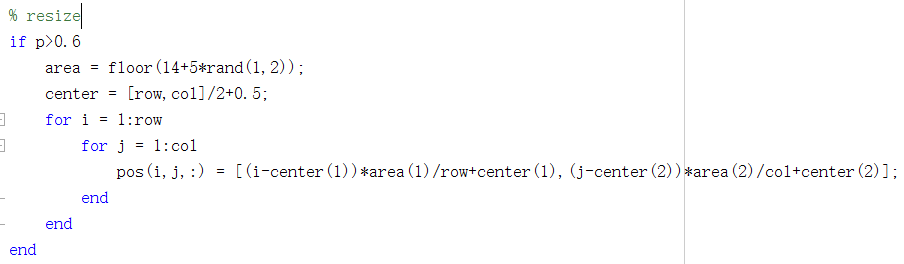
Transform:



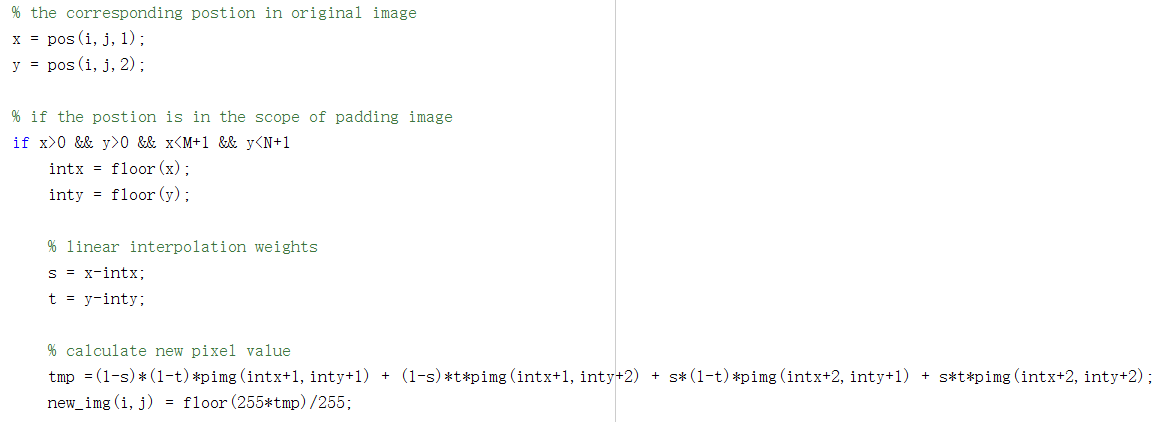
Rotation



Resize



我们将处理后的点的坐标保存在pos矩阵中然后用bilinear\_interpolation插值到新图像中

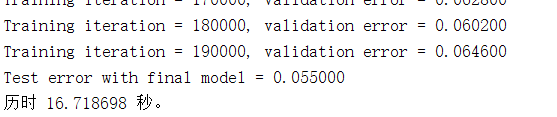


处理后的数据保存在transformed\_digits.mat文件中

我们用随机转换处理后的训练和验证数据对模型进行训练

修改前后的结果比较：

|  |  |  |  |
| --- | --- | --- | --- |
|  | | without transform | transform |
| TEST error | 0.060000 | | 0.055000 |
| figure |  | |  |
|  |  | |  |



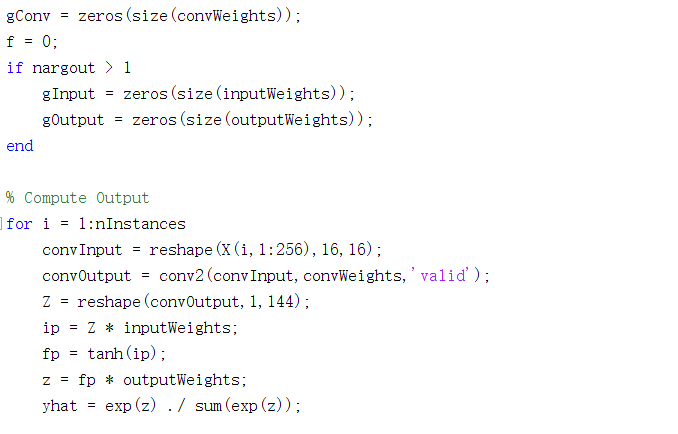
从结果上看验证误差的变化不大但是测试误差有明显的下降。这意味着该模型可以更好地防止过拟合。我们选取处理后的data来训练模型。

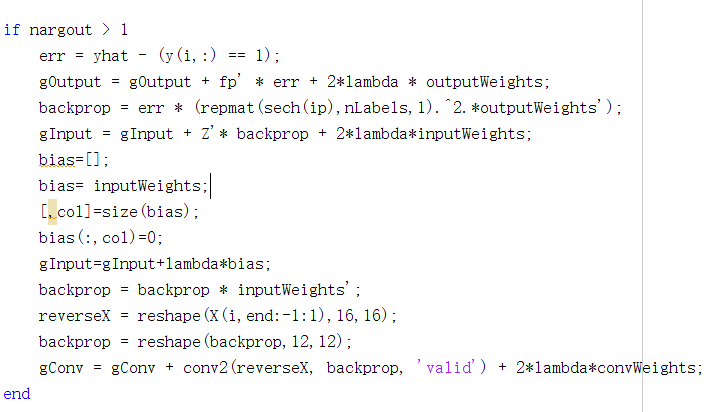
##### Task10: Replace the first layer of the network with a 2D convolutional layer. You will need to reshape the USPS images back to their original 16 by 16 format. The Matlab conv2 function implements 2D convolutions. Filters of size 5 by 5 are a common choice.

卷积层损失函数的偏导可以用下式表示，我们设kernel\_size=5, batch\_size=3

同时我们采用了transformed\_data来训练模型。nHidden设为[100], lambda=1e-3, stepsize=1e-2.

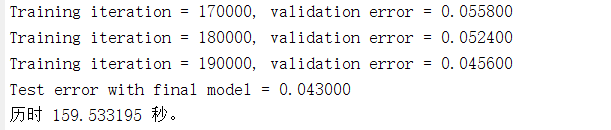
部分代码如下：

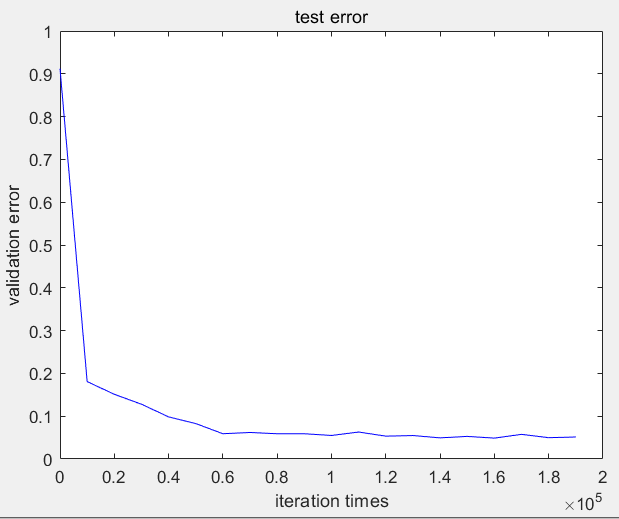




卷积神经网络的结果：

TEST error： 0.043000





从结果看出卷积神经网络运算速度会慢一些，但通过卷积神经网络可以达到更低的测试误差。

##### 总结：

通过我们的优化，最终的测试误差可以降低到0.043000.