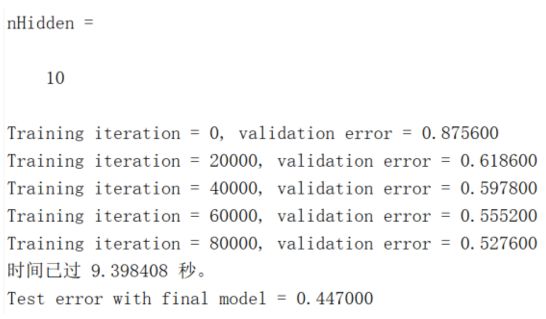
Project-1 of “Neural Network and Deep Learning”

Yixiong Chen   16307110231

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

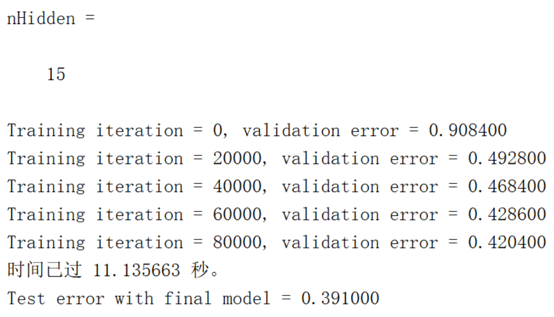


```
nHidden =

    10

Training iteration = 0, validation error = 0.875600
Training iteration = 20000, validation error = 0.618600
Training iteration = 40000, validation error = 0.597800
Training iteration = 60000, validation error = 0.555200
Training iteration = 80000, validation error = 0.527600
时间已过 9.398408 秒。
Test error with final model = 0.447000
```

Fig.1 nHidden = 10

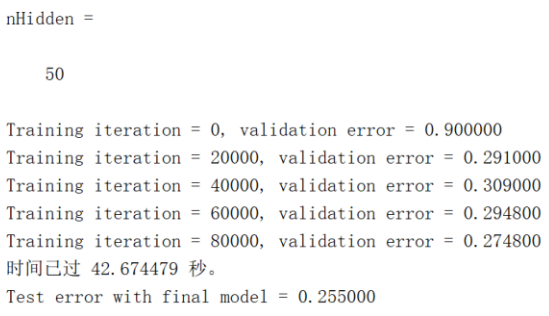


```
nHidden =

    15

Training iteration = 0, validation error = 0.908400
Training iteration = 20000, validation error = 0.492800
Training iteration = 40000, validation error = 0.468400
Training iteration = 60000, validation error = 0.428600
Training iteration = 80000, validation error = 0.420400
时间已过 11.135663 秒。
Test error with final model = 0.391000
```

Fig.2 nHidden = 15

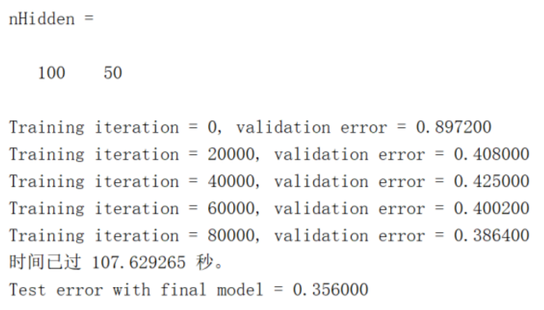


```
nHidden =

    50

Training iteration = 0, validation error = 0.900000
Training iteration = 20000, validation error = 0.291000
Training iteration = 40000, validation error = 0.309000
Training iteration = 60000, validation error = 0.294800
Training iteration = 80000, validation error = 0.274800
时间已过 42.674479 秒。
Test error with final model = 0.255000
```

Fig.3 nHidden = 50



```
nHidden =

   100    50

Training iteration = 0, validation error = 0.897200
Training iteration = 20000, validation error = 0.408000
Training iteration = 40000, validation error = 0.425000
Training iteration = 60000, validation error = 0.400200
Training iteration = 80000, validation error = 0.386400
时间已过 107.629265 秒。
Test error with final model = 0.356000
```

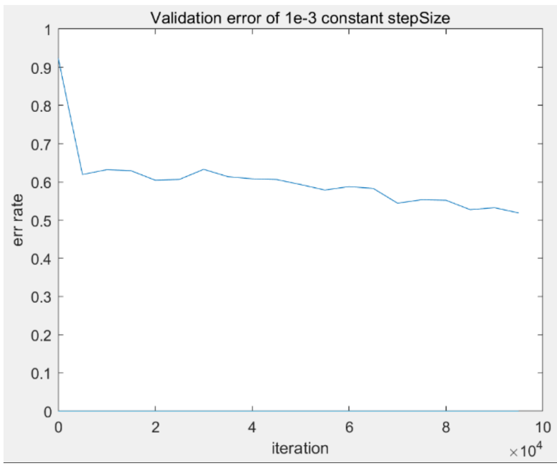
Fig.4 nHidden = 100, 50

运行代码文件“question1_more_nHiddens.m”并修改其中的 23-26 行可以得到不同隐藏层数的网络的训练结果。从以上四张截图中，我们可以发现当隐藏层只有一层的时候，增加这一层的神经元数量可以在增加网络参数、训练时间的同时增强神经网络的表示能力，使得在测试集上的错误率达到更小值。然而，如果加深神经网络到两层隐藏层，不仅网络训练的时间大大延长了，而且相比更少一层的网络准确率也有少许下降。仅根据实验结果推测这是由于网络的表示能力加强后已经超过了这个数据集的需求，在训练集上过拟合严重，导致测试集上的准确率反而有所下降。此外，也有可能是由于更深层的网络参数更多，更加难以训练导致。同时，简单的标准差为 1 的参数初始化也为更深的网络训练增加了难度，因为梯度消失或者梯度爆炸更容易发生。

2. Change the training procedure by modifying the sequence of step-sizes or using different step-sizes for different variables. That momentum uses the update

$$w^{t+1} = w^t - \alpha_t \nabla f(w^t) + \beta_t (w^t - w^{t-1})$$

where α_t is the learning rate (step size) and β_t is the momentum strength. A common value of βt is a constant 0.9.



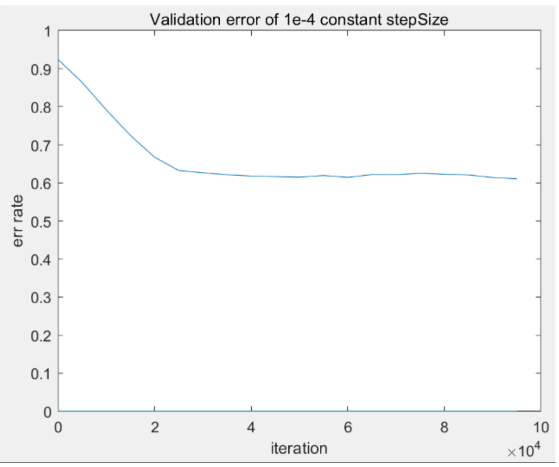
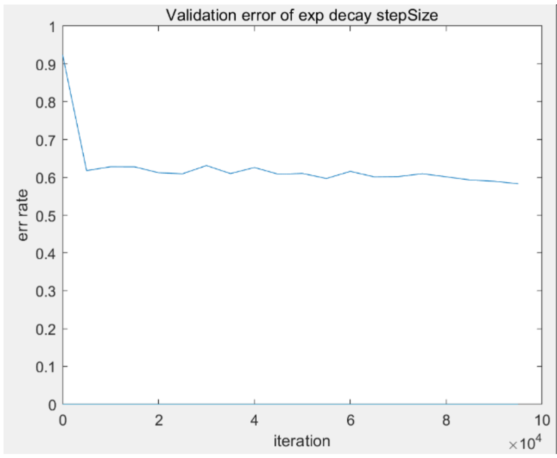


Fig.5 step size = 1e-3　　　　Fig.6 step size = 1e-4



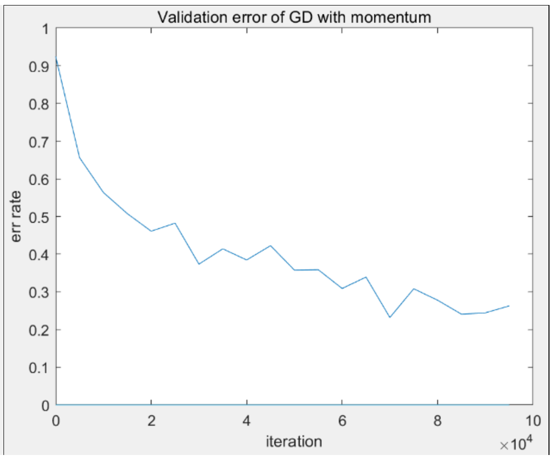


Fig.7 lr exponentially decay (1e-3)　　　　Fig.8 GD with momentum (1e-3)

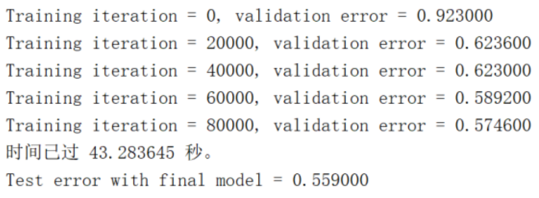
运行代码文件 “question2_diff_stepsize.m”，修改第 35、36 行可以设置梯度下降的模式，在训练十万步后，部分实验结果如上图。

使用常数学习率时，我们发现步长为 1e-1, 1e-2, 1e-3, 1e-4 时，最终达到的测试错误率分别为 85.3%, 25.8%, 48.7%, 61.2%，步长为 1e-2 时达到了最低错误率。当步长太小时，训练速度过慢，并且更可能陷入局部最小值无法跳出，导致最终准确率不高；步长太大时，训练一步可能对参数的改变过多，使得局部的梯度特征无法对下降方向起指导作用，最终导致算法无法进行有效的梯度下降。

在指数下降学习率的梯度下降算法中，以 3e-2, 1e-2, 1e-3 为初始学习率，并令学习率以指数方式在 100000 步后下降到一开始的 1/10，最后它们分别达到了 30.8%, 32.2%, 57.5%测试错误率。使用指数下降学习率能够使得一开始学习率较大，更快地学得粗略特征，而学习率渐渐下降后，有利于函数值渐渐逼近局部最优值而不会以大步幅来回震荡。

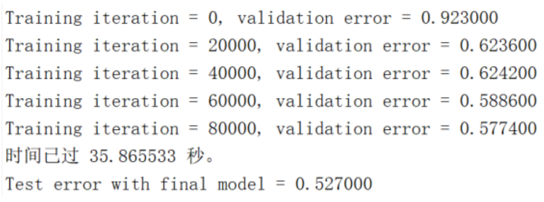
在动量梯度下降算法中，将学习率设置为 3e-2, 1e-2, 3e-3, 1e-3, 3e-4, 1e-4，最终达到 90.5%, 80.1%, 40.6%, 24.7%, 33.1%, 50.9%的测试错误率。我们发现，同样是 1e-3 的学习率，有动量的梯度下降算法不论是最终的准确率还是下降速度都达到了最佳性能。

3. 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.



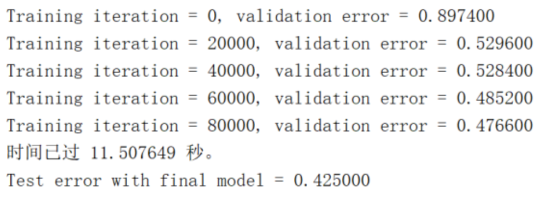
```
Training iteration = 0, validation error = 0.923000
Training iteration = 20000, validation error = 0.623600
Training iteration = 40000, validation error = 0.623000
Training iteration = 60000, validation error = 0.589200
Training iteration = 80000, validation error = 0.574600
时间已过 43.283645 秒。
Test error with final model = 0.559000
```

Fig.9 original loss, nHidden=[15, 10]



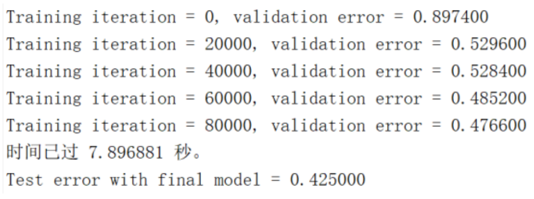
```
Training iteration = 0, validation error = 0.923000
Training iteration = 20000, validation error = 0.623600
Training iteration = 40000, validation error = 0.624200
Training iteration = 60000, validation error = 0.588600
Training iteration = 80000, validation error = 0.577400
时间已过 35.865533 秒。
Test error with final model = 0.527000
```

Fig.10 accelerated loss, nHidden=[15, 10]



```
Training iteration = 0, validation error = 0.897400
Training iteration = 20000, validation error = 0.529600
Training iteration = 40000, validation error = 0.528400
Training iteration = 60000, validation error = 0.485200
Training iteration = 80000, validation error = 0.476600
时间已过 11.507649 秒。
Test error with final model = 0.425000
```

Fig.11 original loss, nHidden=[15]



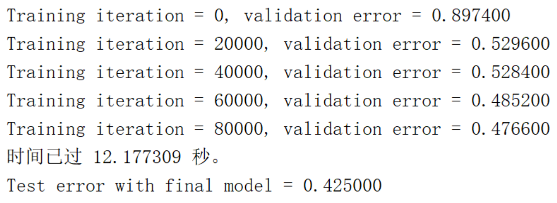
```
Training iteration = 0, validation error = 0.897400
Training iteration = 20000, validation error = 0.529600
Training iteration = 40000, validation error = 0.528400
Training iteration = 60000, validation error = 0.485200
Training iteration = 80000, validation error = 0.476600
时间已过 7.896881 秒。
Test error with final model = 0.425000
```

Fig.12 accelerated loss, nHidden=[15]

运行代码文件“question3_main.m”，修改第 36、37 行可以选择是否使用向量化的 MLP，在训练十万步后，部分实验结果如上图。对应的求梯度的代码在文件“question3_accelereated_MLP.m”中。我们发现，在向量化了 MLP 的前向以及反向传播以后，对于 nHidden=[15, 10]的两层隐藏层的网络而言，速度提升了 20.6%，对于 nHidden=[15]的单隐层神经网络而言，速度提升了 45.6%。

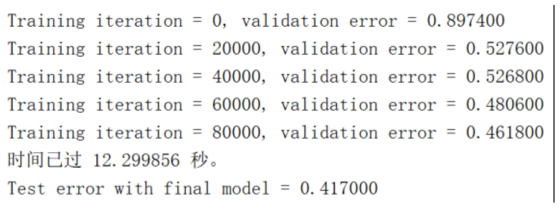
推测向量化的 MLP 计算速度得到极大飞跃的原因是，相比于循环计算而言，向量化计算能够更大程度地利用 Intel 处理器针对 matlab 的并行性支持。在并行的条件下，CPU 的利用效率更高，计算速度也更快。

4. Add l_2 regularization (or l_1-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.



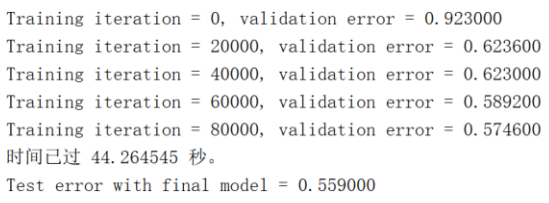
```
Training iteration = 0, validation error = 0.897400
Training iteration = 20000, validation error = 0.529600
Training iteration = 40000, validation error = 0.528400
Training iteration = 60000, validation error = 0.485200
Training iteration = 80000, validation error = 0.476600
时间已过 12.177309 秒。
Test error with final model = 0.425000
```

Fig.13 original loss, nHidden=[15]



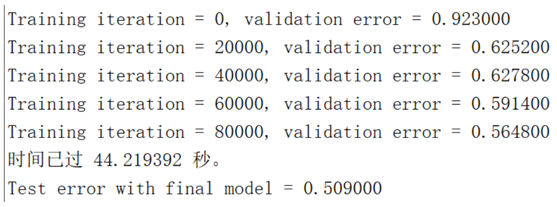
```
Training iteration = 0, validation error = 0.897400
Training iteration = 20000, validation error = 0.527600
Training iteration = 40000, validation error = 0.526800
Training iteration = 60000, validation error = 0.480600
Training iteration = 80000, validation error = 0.461800
时间已过 12.299856 秒。
Test error with final model = 0.417000
```

Fig.14 l2-reg loss, nHidden=[15]



```
Training iteration = 0, validation error = 0.923000
Training iteration = 20000, validation error = 0.623600
Training iteration = 40000, validation error = 0.623000
Training iteration = 60000, validation error = 0.589200
Training iteration = 80000, validation error = 0.574600
时间已过 44.264545 秒。
Test error with final model = 0.559000
```

Fig.15 original loss, nHidden=[15, 10]



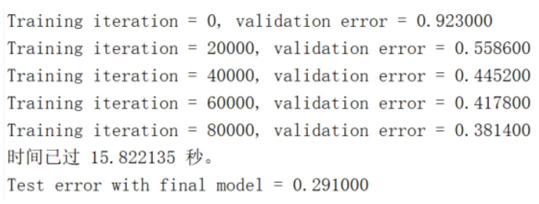
```
Training iteration = 0, validation error = 0.923000
Training iteration = 20000, validation error = 0.625200
Training iteration = 40000, validation error = 0.627800
Training iteration = 60000, validation error = 0.591400
Training iteration = 80000, validation error = 0.564800
时间已过 44.219392 秒。
Test error with final model = 0.509000
```

Fig.16 l2-reg loss, nHidden=[15, 10]

运行代码文件“question4_main.m”，在训练十万步后，部分实验结果如上图。修改该文件的第 36、37 行可以选择是否使用 l2 正则化，对应的 MLP 代码在文件“question4_regularized _MLP.m”中。我们发现，当加上了 l2 正则项以后，模型由于受到了正则化的效果，权重会受

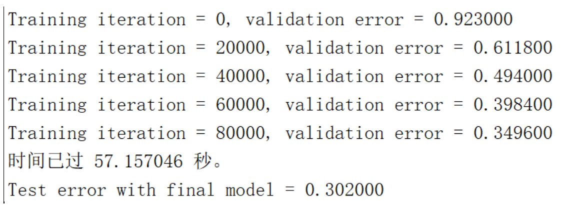
到梯度下降更新的影响而减小，这使得模型的泛化能力增强。不论是在单隐层神经网络还是在双隐层神经网络的情况下，最终的测试错误率都有一定的下降。

5. 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. Recall that the softmax function is $p(y_i) = \frac{exp(z_i)}{\sum_{j=1}^{J} exp(z_j)}$. You can replace squared error with the negative log-likelihood of the true label under this loss, $-logp(y_i)$.



```
Training iteration = 0, validation error = 0.923000
Training iteration = 20000, validation error = 0.558600
Training iteration = 40000, validation error = 0.445200
Training iteration = 60000, validation error = 0.417800
Training iteration = 80000, validation error = 0.381400
时间已过 15.822135 秒。
Test error with final model = 0.291000
```

Fig.17 without softmax layer

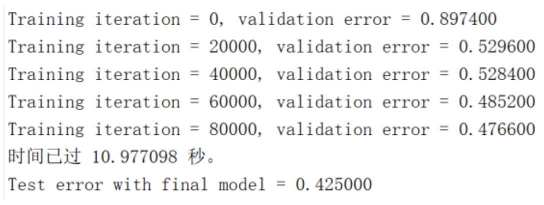


```
Training iteration = 0, validation error = 0.923000
Training iteration = 20000, validation error = 0.611800
Training iteration = 40000, validation error = 0.494000
Training iteration = 60000, validation error = 0.398400
Training iteration = 80000, validation error = 0.349600
时间已过 57.157046 秒。
Test error with final model = 0.302000
```

Fig.18 with softmax layer

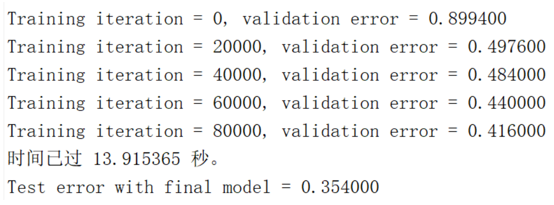
运行代码文件"question5_main.m"，修改第 36、37 行可以选择是否使用 softmax 层，在训练十万步后，部分实验结果如上图。对应的 MLP 代码在文件"question5_softmax _MLP.m"中。我们发现，如果在训练步长为 1e-2，nHidden=[15 10]的情况下，使用 softmax + log loss 的组合，训练速度会大大降低，并且准确率也会有小幅度下降。这是因为平方损失的求导比负对数损失容易很多，并且对于 softmax 函数而言，梯度经过这一层会消耗额外的计算量。由于任务比较简单，使用 softmax 函数做分类任务并不会起到更好的效果，反而使得计算更加复杂了，增大的网络容量可能使得模型的过拟合更严重。

6. 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.



```
Training iteration = 0, validation error = 0.897400
Training iteration = 20000, validation error = 0.529600
Training iteration = 40000, validation error = 0.528400
Training iteration = 60000, validation error = 0.485200
Training iteration = 80000, validation error = 0.476600
时间已过 10.977098 秒。
Test error with final model = 0.425000
```

Fig.19 without bias

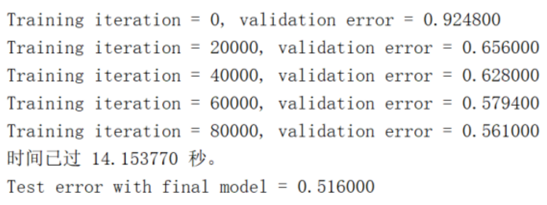


```
Training iteration = 0, validation error = 0.899400
Training iteration = 20000, validation error = 0.497600
Training iteration = 40000, validation error = 0.484000
Training iteration = 60000, validation error = 0.440000
Training iteration = 80000, validation error = 0.416000
时间已过 13.915365 秒。
Test error with final model = 0.354000
```

Fig.20 with bias

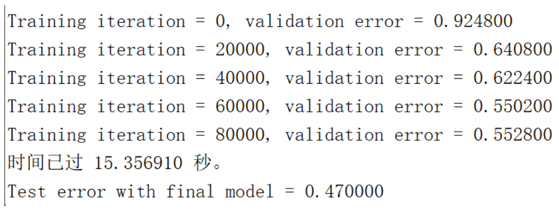
运行代码文件"question6_main.m"，修改第 40 行可以选择是否使用偏置项，在训练十万步后，部分实验结果如上图。对应的 MLP 代码在文件"question6_ MLP_with_bias.m"中。运行结果如图 19、图 20 所示，这两张图分别展示了 nHidden=[15]时隐藏层加偏置项与不加偏置项的训练过程。我们发现，当训练囊括了偏置项时，训练得到的验证集误差显著要比不加偏置项低。这一方面是由于参数的增加的确增加了模型容量，另一方面是由于偏置项使得模型一层的表示能力更加丰富，它们都使得训练更加有效。当然，由于包含了更多参数需要更新，模型训练时间也更长了。

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



```
Training iteration = 0, validation error = 0.924800
Training iteration = 20000, validation error = 0.656000
Training iteration = 40000, validation error = 0.628000
Training iteration = 60000, validation error = 0.579400
Training iteration = 80000, validation error = 0.561000
时间已过 14.153770 秒。
Test error with final model = 0.516000
```

Fig.21 without dropout



```
Training iteration = 0, validation error = 0.924800
Training iteration = 20000, validation error = 0.640800
Training iteration = 40000, validation error = 0.622400
Training iteration = 60000, validation error = 0.550200
Training iteration = 80000, validation error = 0.552800
时间已过 15.356910 秒。
Test error with final model = 0.470000
```

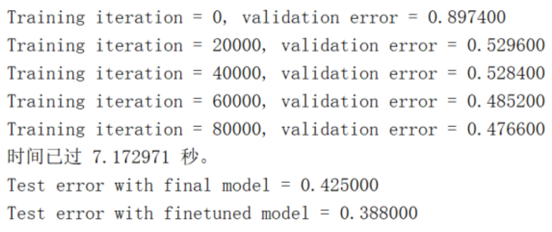
Fig.22 with dropout, p=0.1 to drop

运行代码文件 “question7_main.m” ，修改第 36, 37 行可以选择是否使用偏置项，在训练十万步后，部分实验结果如上图。对应的 MLP 代码在文件 “question7_ MLP_with_dropout.m” 中。运行结果如图 21、图 22 所示，这两张图分别展示了 nHidden=[20 10]时隐藏层加偏置项与不加偏置项的训练过程。我们发现在 0.1 的舍弃概率下，模型是有一定提升的。然而，经过实验我们发现，当舍弃概率从 0.1 开始增大时，模型最终的泛化误差反而开始上升。结果如下表所示：

P	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
测试误差	0.470	0.560	0.640	0.700	0.699	0.713	0.674	0.772	0.812

从这个表格我们发现，当 p=0.2 时，使用 dropout 的模型性能就已经开始不如不使用 dropout 的模型了。经过观察模型的梯度，我们发现，当模型拥有两层隐藏层时，神经元饱和的问题就已经很严重了，底层神经元的梯度比顶层神经元小若干个数量级。在加入了 dropout 以后，神经元的饱和情况更是雪上加霜，除了失活的神经元以外几乎全部饱和。这个问题似乎是由不恰当的权值初始化引起的，越深的神经网络的梯度消失越严重。

8. You can do ‘fine-tuning’ of the last layer. Fix the parameters of all the layers except the last one, and solve for the parameters of the last layer exactly as a convex optimization problem. E.g., treat the input to the last layer as the features and use techniques from earlier in the course (this is particularly fast if you use the squared error, since it has a closed-form solution).



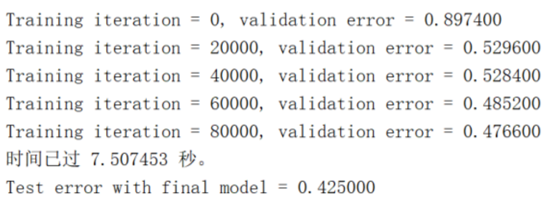
```
Training iteration = 0, validation error = 0.897400
Training iteration = 20000, validation error = 0.529600
Training iteration = 40000, validation error = 0.528400
Training iteration = 60000, validation error = 0.485200
Training iteration = 80000, validation error = 0.476600
时间已过 7.172971 秒。
Test error with final model = 0.425000
Test error with finetuned model = 0.388000
```

Fig.23 the improvement of finetuning

运行代码文件 “question8_finetune.m” ，在训练十万步后，实验结果如上图。对应的预测倒数第二层网络输出的代码在文件 “question8_ finetune_predict.m”中。

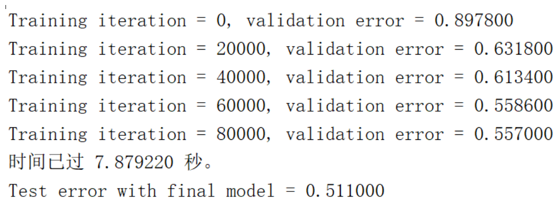
运行的结果如图所示，这张图展示了当 nHidden=[15]时在直接训练网络后就进行预测以及再经过 finetuning 后进行预测的测试集错误率。我们发现在进行了最后一层权重的 finetuning 后，模型性能立竿见影地明显提升了。尤其如果选用的是平方损失，直接进行显式求解并不会消耗多大的计算量，却可以得到巨大的提升。

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



```
Training iteration = 0, validation error = 0.897400
Training iteration = 20000, validation error = 0.529600
Training iteration = 40000, validation error = 0.528400
Training iteration = 60000, validation error = 0.485200
Training iteration = 80000, validation error = 0.476600
时间已过 7.507453 秒。
Test error with final model = 0.425000
```

Fig.24 without data augmentation



```
Training iteration = 0, validation error = 0.897800
Training iteration = 20000, validation error = 0.631800
Training iteration = 40000, validation error = 0.613400
Training iteration = 60000, validation error = 0.558600
Training iteration = 80000, validation error = 0.557000
时间已过 7.879220 秒。
Test error with final model = 0.511000
```

Fig.25 with data augmentation

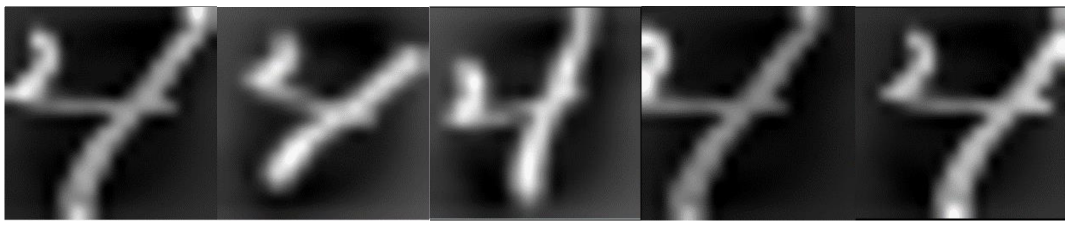


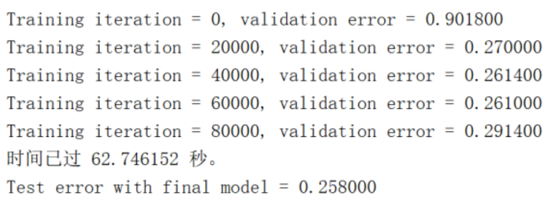
Fig.26-Fig.30 different data augmentation methods

先运行“question9_transforms.m”，这个程序会生成数据增强后的数据集。第二步运行“question9_data_augmentation.m”, 在第 5 行可以选择使用的是原数据集还是增强后的数据集。

如图 24、25 所示，这是在 nHidden=[15]下训练十万步后的结果。我们惊奇地发现：没有数据增强的情况下训练出来的模型性能是要好于有数据增强的情况的。在 nHidden=[50]以及 nHidden=[15 10]的情况下分别实验，无数据增强的结果分别为 0.26, 0.527，有数据增强的结果分别为 0.373, 0.546，同样得出了相似的结论。经过一系列查询和思考，这个结果可能是由于训练集与验证集、测试集太过于相似，这导致数据增强后的数据样本分别反而有了偏差。这使得最终模型的测试结果恶化。

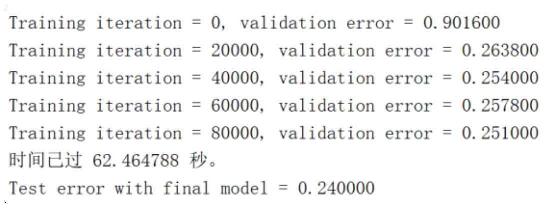
在运行了以上两个程序后，可以经过“question9_vis.m”进行数据可视化。如图 26-30 所示，这五张图片分别是原图、顺时针旋转 15 度的图像、逆时针旋转 15 度的图像、左平移 2 个像素的图像、右平移两个像素的图像。

10. 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.



```
Training iteration = 0, validation error = 0.901800
Training iteration = 20000, validation error = 0.270000
Training iteration = 40000, validation error = 0.261400
Training iteration = 60000, validation error = 0.261000
Training iteration = 80000, validation error = 0.291400
时间已过 62.746152 秒。
Test error with final model = 0.258000
```

Fig.31 convolutional net



```
Training iteration = 0, validation error = 0.901600
Training iteration = 20000, validation error = 0.263800
Training iteration = 40000, validation error = 0.254000
Training iteration = 60000, validation error = 0.257800
Training iteration = 80000, validation error = 0.251000
时间已过 62.464788 秒。
Test error with final model = 0.240000
```

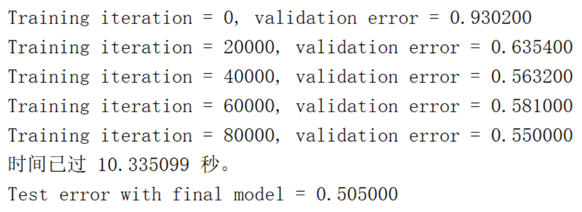
Fig.32 fc net with nHidden=[80]

运行代码文件 “question10_main.m”，在训练十万步后，实验结果如上图。对应的网络定义以及反向传播在代码文件“question10_ConvNet.m”中，inference 代码为“question10_Conv_Predict.m”。

如图 31、32 所示，设定卷积网络只含有一层 3x3 卷积层，在将与原图尺寸相同的特征图展开后连接有一层全连接层，而全连接神经网络包含一层含有 80 个神经元的全连接层。在相同的计算时长下，卷积神经网络最终性能并没有超过全连接神经网络。经过实验，将此

设定下的 3x3 卷积改为 5x5 卷积后，测试集错误率为 25.7%，准确率相差不大，训练时间为 61 秒，同样十分接近。

然而如果从参数量上来看，卷积神经网络仅有$3 \times 3 + 257 \times 10 = 2579$个参数，而全连接网络包含$257 \times 80 + 80 \times 10 = 21360$个参数，远远多于卷积网络。在全连接网络参数量与卷积网络接近的情况下，这个网络只能有 10 个隐藏神经元。在 10 个隐藏层神经元的全连接网络的设定下运行程序，结果如图 33 所示。



```
Training iteration = 0, validation error = 0.930200
Training iteration = 20000, validation error = 0.635400
Training iteration = 40000, validation error = 0.563200
Training iteration = 60000, validation error = 0.581000
Training iteration = 80000, validation error = 0.550000
时间已过 10.335099 秒。
Test error with final model = 0.505000
```

Fig.33 fc net with nHidden=[10]

我们发现如果两种网络参数量接近，则卷积神经网络的性能远远好于全连接神经网络。这表明卷积层在图像任务上有着非常强大的编码能力，能够将局部特征非常好地提取出来，使得网络表示能力大大强于全连接神经网络。

11. 由于前 10 题为了逐一展示各种操作对网络训练的影响，每一题中并不涉及除了该种调整以外的其他优化。最后，我们通过设置 Xavier 参数初始化方法、利用带动量的梯度下降方法、增加网络的层数为 8 个卷积核，全连接隐藏层的神经元数量分别为(128, 128, 64, 32, 32)，得到了一个性能较为强劲的模型。具体代码见 “FinalNet.m”，运行结果如图 34 所示。最终测试集上预测准确率达到了 96.7%，如果再往模型上加上几层卷积层，每一层考虑偏置项，往损失函数中加上正则项，准确率可能会更高。

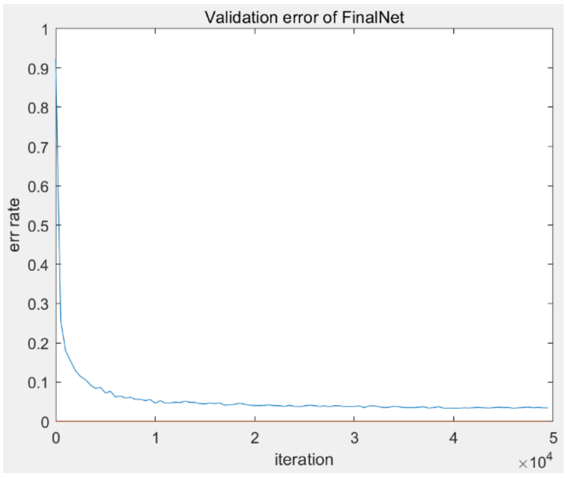


Fig.34 validation error of FinalNet