Neural Network and Deep Learning Project1

姜海天 19307110022

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

In this problem, I switched different structures of network. The structure of network is specified by the variable `nHidden`. I adjusted `nHidden` to be [10], [20], [100], [200], [10 10], [10 10 10], [128 128 64]. The results are shown as follows. From the results we can see that more parameters gives rise to better performance, but simply stacking linear layer does not profoundly enhance the performance.

```
nHidden is     10
......
Training iteration = 95000, validation error = 0.462400
Test error with final model = 0.474000

nHidden is     20
......
Training iteration = 95000, validation error = 0.359200
Test error with final model = 0.354000

nHidden is    100
......
Training iteration = 95000, validation error = 0.228000
Test error with final model = 0.234000

nHidden is    200
......
Training iteration = 95000, validation error = 0.212000
Test error with final model = 0.190000

nHidden is     10    10
......
Training iteration = 95000, validation error = 0.549800
Test error with final model = 0.516000

nHidden is     10    10    10
......
Training iteration = 95000, validation error = 0.666000
Test error with final model = 0.595000

nHidden is    128   128    64
......
Training iteration = 95000, validation error = 0.372400
Test error with final model = 0.400000
```

Problem 2 Change the training procedure by modifying the sequence of step-sizes or using different step-sizes for different variables. Also try momentum.

First I change different `stepSize`. From the result we can see that large step size may not converge, while too small step size will lead to slow convergence.

```
stepSize is     1
......
Training iteration = 95000, validation error = 0.901600
Test error with final model = 0.905000

stepSize is     1e-1
......
Training iteration = 95000, validation error = 0.879400
Test error with final model = 0.889000

stepSize is     1e-2
......
Training iteration = 95000, validation error = 0.257400
Test error with final model = 0.268000

stepSize is     1e-3
......
Test error with final model = 0.546000

stepSize is     1e-4
......
Training iteration = 95000, validation error = 0.639600
Test error with final model = 0.660000
```

Now add momentum. We can find that $w^{t+1} - w^t = -\alpha_t \nabla f\left(w^t\right) + \beta_t \left(w^t - w^{t-1}\right)$, so we need a variable to store the value of $w^t - w^{t-1}$. By using the same `nHidden` as 10, we can see that the convergence and performance are way better than its vanilla SGD counterpart.

```
nHidden is     10
stepSize is    1e-3
......
Training iteration = 95000, validation error = 0.250400
Test error with final model = 0.200000
```

Problem 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.

Other than changing the for-loop in back propagation into matrix multiplication, I also merged the output layer with hidden layer to make the code more neat.

Result:

```
>> tic; example_neuralNetwork; toc;
......
Elapsed time is 14.035144 seconds.

>> tic; vectorizedNeuralNetwork; toc;
......
Elapsed time is 6.290832 seconds.
```

Problem 4 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.

Result of regularization:

```
Training iteration = 0, validation error = 0.875400
Training iteration = 5000, validation error = 0.622200
Training iteration = 10000, validation error = 0.616600
Training iteration = 15000, validation error = 0.604400
Training iteration = 20000, validation error = 0.610400
Training iteration = 25000, validation error = 0.609000
Training iteration = 30000, validation error = 0.601600
Training iteration = 35000, validation error = 0.613400
Training iteration = 40000, validation error = 0.603400
Training iteration = 45000, validation error = 0.601200
Training iteration = 50000, validation error = 0.594000
Training iteration = 55000, validation error = 0.592400
Training iteration = 60000, validation error = 0.583800
Training iteration = 65000, validation error = 0.613600
Training iteration = 70000, validation error = 0.630200
Training iteration = 75000, validation error = 0.625400
Training iteration = 80000, validation error = 0.621600
Training iteration = 85000, validation error = 0.618200
Training iteration = 90000, validation error = 0.594400
Training iteration = 95000, validation error = 0.600000
Test error with final model = 0.564000
```

Result of early stopping:

```
Training iteration = 0, validation error = 0.907800
Training iteration = 5000, validation error = 0.561000
Training iteration = 10000, validation error = 0.559600
Training iteration = 15000, validation error = 0.559600
Endure 1 times
Training iteration = 20000, validation error = 0.563800
Endure 2 times
Training iteration = 25000, validation error = 0.557200
Training iteration = 30000, validation error = 0.549600
Training iteration = 35000, validation error = 0.548400
Training iteration = 40000, validation error = 0.564600
Endure 3 times
Training iteration = 45000, validation error = 0.555000
Endure 4 times
Training iteration = 50000, validation error = 0.560000
Endure 5 times
Training iteration = 55000, validation error = 0.552200
Endure 6 times
Early stopping
Test error with final model = 0.516000
```

We can see that both of these two methods reduced overfitting on training data.

Problem 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 $p(y_i) = \exp(z_i) / \sum_{j=1}^{J} \exp(z_j)$. You can replace squared error with the negative log-likelihood of the true label under this loss, $-\log p(y_l)$

derivative derivation:

$$-\log p(y_l) = -\log\left(\exp(z_l) / \sum_{j=1}^{J} \exp(z_j)\right) = -z_l + \log \sum \exp(z_j)$$

$$\frac{d}{dz_i} \log \sum \exp(z_j) = \exp(z_i) / \sum \exp(z_j) = y_i$$

Still using the `MLPClassificationPredict` because softmax only change the scale of each component without reordering.

Result:

```
Training iteration = 0, validation error = 0.929000
Training iteration = 5000, validation error = 0.873400
Training iteration = 10000, validation error = 0.776800
Training iteration = 15000, validation error = 0.701600
```

```
Training iteration = 20000, validation error = 0.659200
Training iteration = 25000, validation error = 0.639400
Training iteration = 30000, validation error = 0.620600
Training iteration = 35000, validation error = 0.612600
Training iteration = 40000, validation error = 0.597600
Training iteration = 45000, validation error = 0.594400
Training iteration = 50000, validation error = 0.588400
Training iteration = 55000, validation error = 0.586200
Training iteration = 60000, validation error = 0.584000
Training iteration = 65000, validation error = 0.581600
Training iteration = 70000, validation error = 0.575200
Training iteration = 75000, validation error = 0.568800
Training iteration = 80000, validation error = 0.568000
Training iteration = 85000, validation error = 0.567800
Training iteration = 90000, validation error = 0.574200
Training iteration = 95000, validation error = 0.571400
Test error with final model = 0.555000
```

Problem 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.

Still use softmax as output activation and negative log likelihood as loss function because this is a classification problem.

Result: adding bias term in hidden layer enhance the performance

```
Training iteration = 0, validation error = 0.873600
Training iteration = 5000, validation error = 0.786600
Training iteration = 10000, validation error = 0.697800
Training iteration = 15000, validation error = 0.628400
Training iteration = 20000, validation error = 0.582600
Training iteration = 25000, validation error = 0.547800
Training iteration = 30000, validation error = 0.512000
Training iteration = 35000, validation error = 0.487200
Training iteration = 40000, validation error = 0.465800
Training iteration = 45000, validation error = 0.442800
Training iteration = 50000, validation error = 0.432800
Training iteration = 55000, validation error = 0.416600
Training iteration = 60000, validation error = 0.405200
Training iteration = 65000, validation error = 0.396000
Training iteration = 70000, validation error = 0.388800
Training iteration = 75000, validation error = 0.375600
Training iteration = 80000, validation error = 0.368400
Training iteration = 85000, validation error = 0.356200
Training iteration = 90000, validation error = 0.348000
Training iteration = 95000, validation error = 0.336600
Test error with final model = 0.329000
```

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

Keep using softmax and NLL loss.

Result: Got better performance due to less overfitting training data.

```
Training iteration = 0, validation error = 0.838000
Training iteration = 5000, validation error = 0.754200
Training iteration = 10000, validation error = 0.684200
Training iteration = 15000, validation error = 0.617800
Training iteration = 20000, validation error = 0.567200
Training iteration = 25000, validation error = 0.527000
Training iteration = 30000, validation error = 0.495800
Training iteration = 35000, validation error = 0.468800
Training iteration = 40000, validation error = 0.450800
Training iteration = 45000, validation error = 0.435400
Training iteration = 50000, validation error = 0.420400
Training iteration = 55000, validation error = 0.409400
Training iteration = 60000, validation error = 0.397200
Training iteration = 65000, validation error = 0.386600
Training iteration = 70000, validation error = 0.372200
Training iteration = 75000, validation error = 0.357200
Training iteration = 80000, validation error = 0.347400
Training iteration = 85000, validation error = 0.341800
Training iteration = 90000, validation error = 0.333200
Training iteration = 95000, validation error = 0.324200
Test error with final model = 0.294000
```

Problem 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)

Use MSE loss now because it has closed-form solution.

Result: Fine-tuning enhanced the performance of the neural network.

```
Training iteration = 0, validation error = 0.912800
Training iteration = 5000, validation error = 0.623400
Training iteration = 10000, validation error = 0.627200
Training iteration = 15000, validation error = 0.603600
Training iteration = 20000, validation error = 0.604400
Training iteration = 25000, validation error = 0.601600
```

```
Training iteration = 30000, validation error = 0.592800
Training iteration = 35000, validation error = 0.594800
Training iteration = 40000, validation error = 0.593400
Training iteration = 45000, validation error = 0.584200
Training iteration = 50000, validation error = 0.560600
Training iteration = 55000, validation error = 0.572000
Training iteration = 60000, validation error = 0.559000
Training iteration = 65000, validation error = 0.554200
Training iteration = 70000, validation error = 0.541400
Training iteration = 75000, validation error = 0.537000
Training iteration = 80000, validation error = 0.548800
Training iteration = 85000, validation error = 0.538200
Training iteration = 90000, validation error = 0.504600
Training iteration = 95000, validation error = 0.510600
Test error = 0.496000
Test error after fine tuning = 0.475000
```

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

Used four methods to augment data: translation, rotation, resizing and adding Gaussian noise.

Result from the example vanilla neural network: the testing error is lower due to better generalization.

```
Training iteration = 0, validation error = 0.913200
Training iteration = 5000, validation error = 0.602800
Training iteration = 10000, validation error = 0.605000
Training iteration = 15000, validation error = 0.582800
Training iteration = 20000, validation error = 0.591200
Training iteration = 25000, validation error = 0.584800
Training iteration = 30000, validation error = 0.566800
Training iteration = 35000, validation error = 0.534600
Training iteration = 40000, validation error = 0.580800
Training iteration = 45000, validation error = 0.548200
Training iteration = 50000, validation error = 0.569000
Training iteration = 55000, validation error = 0.546400
Training iteration = 60000, validation error = 0.543200
Training iteration = 65000, validation error = 0.551800
Training iteration = 70000, validation error = 0.517600
Training iteration = 75000, validation error = 0.519200
Training iteration = 80000, validation error = 0.540600
Training iteration = 85000, validation error = 0.527400
Training iteration = 90000, validation error = 0.521800
Training iteration = 95000, validation error = 0.493000
Test error with final model = 0.442000
```

Problem 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.

Suppose $I * w = F$, where I refers to image, F refers to the feature map, then by simple calculation, we can find that $\frac{\partial L}{\partial w} = I * \frac{\partial L}{\partial F}$. That is the back propagation law for the convolution kernel.

Result on softmax: the performance enhanced compared to no convolution layer.

```
Training iteration = 0, validation error = 0.879800
Training iteration = 5000, validation error = 0.744000
Training iteration = 10000, validation error = 0.627400
Training iteration = 15000, validation error = 0.537000
Training iteration = 20000, validation error = 0.495400
Training iteration = 25000, validation error = 0.467200
Training iteration = 30000, validation error = 0.437600
Training iteration = 35000, validation error = 0.411600
Training iteration = 40000, validation error = 0.413800
Training iteration = 45000, validation error = 0.384600
Training iteration = 50000, validation error = 0.384800
Training iteration = 55000, validation error = 0.379200
Training iteration = 60000, validation error = 0.367600
Training iteration = 65000, validation error = 0.361800
Training iteration = 70000, validation error = 0.367600
Training iteration = 75000, validation error = 0.363600
Training iteration = 80000, validation error = 0.360800
Training iteration = 85000, validation error = 0.348600
Training iteration = 90000, validation error = 0.361000
Training iteration = 95000, validation error = 0.355600
Test error with final model = 0.356000
```

Extension 1 Use ReLU as activation function.

$ReLU(x) = \max(x, 0)$

$ReLU'(x) = 1, x > 0;\ \ 0, x \leq 0$

Result: Using ReLU as activation function for each layer is better than tanh.

```
Training iteration = 0, validation error = 0.931000
Training iteration = 5000, validation error = 0.540000
Training iteration = 10000, validation error = 0.528600
Training iteration = 15000, validation error = 0.474400
Training iteration = 20000, validation error = 0.450600
Training iteration = 25000, validation error = 0.420600
Training iteration = 30000, validation error = 0.435600
Training iteration = 35000, validation error = 0.377600
```

```
Training iteration = 40000, validation error = 0.373400
Training iteration = 45000, validation error = 0.361200
Training iteration = 50000, validation error = 0.321800
Training iteration = 55000, validation error = 0.330800
Training iteration = 60000, validation error = 0.317200
Training iteration = 65000, validation error = 0.316400
Training iteration = 70000, validation error = 0.312200
Training iteration = 75000, validation error = 0.324800
Training iteration = 80000, validation error = 0.281200
Training iteration = 85000, validation error = 0.271000
Training iteration = 90000, validation error = 0.272600
Training iteration = 95000, validation error = 0.267600
Test error with final model = 0.250000
```

Extension 2 Use sigmoid as activation function

$\sigma(x) = \frac{1}{1+e^{-x}}$

$\sigma'(x) = \sigma(x) \cdot \sigma(-x)$

Result: after longer training, get way better result than tanh activation function.

```
Training iteration = 0, validation error = 0.891800
Training iteration = 15000, validation error = 0.782600
Training iteration = 30000, validation error = 0.650200
Training iteration = 45000, validation error = 0.568600
Training iteration = 60000, validation error = 0.511600
Training iteration = 75000, validation error = 0.472200
Training iteration = 90000, validation error = 0.434000
Training iteration = 105000, validation error = 0.405600
Training iteration = 120000, validation error = 0.380200
Training iteration = 135000, validation error = 0.358400
Training iteration = 150000, validation error = 0.341200
Training iteration = 165000, validation error = 0.327200
Training iteration = 180000, validation error = 0.313400
Training iteration = 195000, validation error = 0.300400
Training iteration = 210000, validation error = 0.288000
Training iteration = 225000, validation error = 0.274600
Training iteration = 240000, validation error = 0.264800
Training iteration = 255000, validation error = 0.257600
Training iteration = 270000, validation error = 0.248800
Training iteration = 285000, validation error = 0.246000
Test error with final model = 0.223000
```

Extension 3 Minibatch SGD

Add mini-batch training to the ReLU activated softmax neural network. The batch size if set to 10.

The result is better than last one because of additional information for back propagation.

```
Training iteration = 0, validation error = 0.876400
Training iteration = 5000, validation error = 0.327600
Training iteration = 10000, validation error = 0.232800
Training iteration = 15000, validation error = 0.181000
Training iteration = 20000, validation error = 0.172600
Training iteration = 25000, validation error = 0.171800
Training iteration = 30000, validation error = 0.148400
Training iteration = 35000, validation error = 0.134200
Training iteration = 40000, validation error = 0.126800
Training iteration = 45000, validation error = 0.147600
Training iteration = 50000, validation error = 0.123000
Training iteration = 55000, validation error = 0.131000
Training iteration = 60000, validation error = 0.117800
Training iteration = 65000, validation error = 0.126200
Training iteration = 70000, validation error = 0.121800
Training iteration = 75000, validation error = 0.115800
Training iteration = 80000, validation error = 0.117200
Training iteration = 85000, validation error = 0.107800
Training iteration = 90000, validation error = 0.112600
Training iteration = 95000, validation error = 0.116200
Test error with final model = 0.117000
```

Extension 4 Learning rate Decay

Use step learning rate decay, namely dropping the learning rate by a factor every few steps.

Result: The training is more steady because we can explore more delicately in the loss landscape after adequate turns of training.

```
Training iteration = 0, validation error = 0.893800
Training iteration = 5000, validation error = 0.532600
Training iteration = 10000, validation error = 0.486800
Training iteration = 15000, validation error = 0.508200
Training iteration = 20000, validation error = 0.469600
Training iteration = 25000, validation error = 0.443200
Training iteration = 30000, validation error = 0.404000
Training iteration = 35000, validation error = 0.403000
Training iteration = 40000, validation error = 0.389800
Training iteration = 45000, validation error = 0.381600
Training iteration = 50000, validation error = 0.379600
Training iteration = 55000, validation error = 0.369400
Training iteration = 60000, validation error = 0.361600
```

```
Training iteration = 65000, validation error = 0.365000
Training iteration = 70000, validation error = 0.360400
Training iteration = 75000, validation error = 0.345600
Training iteration = 80000, validation error = 0.341800
Training iteration = 85000, validation error = 0.336800
Training iteration = 90000, validation error = 0.341000
Training iteration = 95000, validation error = 0.336600
Test error with final model = 0.335000
```

Extension 5 All in one

Integrate the above tricks into one whole neural network, and achieved the best performance of over 95% accuracy on test set.

```
Training iteration = 0, validation error = 0.926400
Training iteration = 5000, validation error = 0.099400
Training iteration = 10000, validation error = 0.074800
Training iteration = 15000, validation error = 0.060400
Training iteration = 20000, validation error = 0.056600
Training iteration = 25000, validation error = 0.051800
Training iteration = 30000, validation error = 0.051600
Training iteration = 35000, validation error = 0.048400
Training iteration = 40000, validation error = 0.050800
Training iteration = 45000, validation error = 0.049000
Training iteration = 50000, validation error = 0.050200
Training iteration = 55000, validation error = 0.049800
Training iteration = 60000, validation error = 0.047000
Training iteration = 65000, validation error = 0.047400
Training iteration = 70000, validation error = 0.048800
Training iteration = 75000, validation error = 0.049400
Training iteration = 80000, validation error = 0.047800
Training iteration = 85000, validation error = 0.049200
Training iteration = 90000, validation error = 0.049200
Training iteration = 95000, validation error = 0.048800
Test error with final model = 0.044000
```