Project-1 of “Neural Network and Deep Learning”

1. Introduction

In this problem we will investigate handwritten digit classification. The inputs are 16 by 16 grayscale images of handwritten digits (0 through 9), and the goal is to predict the number of the given the image. If you run example\_neuralNetwork it will load this dataset and train a neural network with stochastic gradient descent, printing the validation set error as it goes. To handle the 10 classes, there are 10 output units (using a {−1, 1} encoding of each of the ten labels) and the squared error is used during training.

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

**answer:**

‘nHidden = [n]’ means that [n] represents a single-layer neural network, and [n1,n2,n3... n10] represents a 10-layer neural network. The dimension of the "nHidden" vector is equal to the number of layers of the neural network. By implementing different values, the following results can be obtained,

|  |  |
| --- | --- |
| nHidden = [10] | nHidden = [30] |
| Test error = 0.514000 | **Test error = 0.303000** |
|  |  |
| nHidden = [10,10] | **nHidden = [30,30]** |
| Test error = 0.588000 | **Test error = 0.453000** |
|  |  |

It can be seen from the above results that as the number of single-layer neurons increases, not only the test error gradually decreases, but also the speed of the validation error decreases. At the same time, although the increase in the number of layers can effectively reduce the test error, but the training speed is slow, so we choose A

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

**answer:**

As an important super parameter in deep learning, step-sizes determines whether the objective function converges to the local minimum and when.

The proper step-sizes can make the objective function converge to the local minimum in the proper time. By implementing different values, the following results can be obtained,

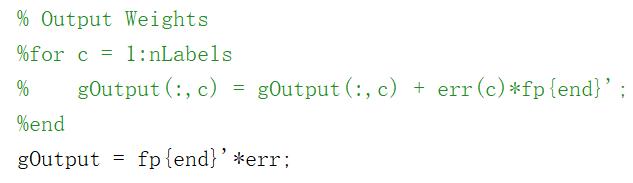
|  |  |
| --- | --- |
| step-sizes = 1e-3 | step-sizes = 1e-2 |
| Test error = 0.302000 | **Test error = 0.240000** |
|  |  |
| step-sizes = 1e-1 | **step-sizes = 3e-2** |
| Test error = 0.905000 | **Test error = 0.834000** |
|  |  |

It can be seen from the above results that step-sizes too high will cause loss exploding, and step-sizes too low will cause loss to drop too slowly. Therefore, the experiment selects step-sizes = 1e-2.

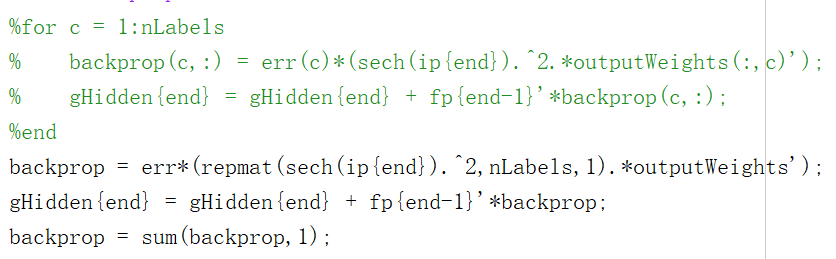
* 1. **Task3:** You could vectorize evaluating the loss function (e.g., try to express as much as possible in terms of matrix operations), to allow you to do more training iterations in a reasonable amount of time.

**answer:**

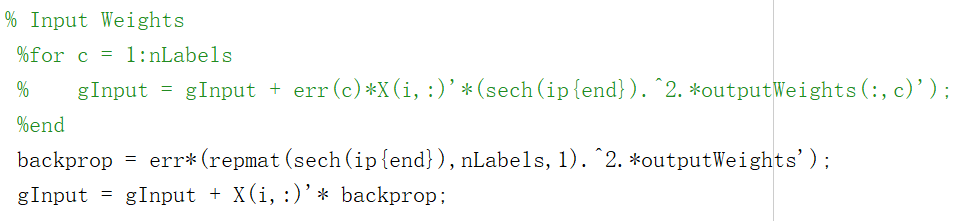
The for loop can be changed to matrix operation, the specific modification process is as follows,



modification 1



modification 2



modification 3

|  |  |
| --- | --- |
| Before modification | After modification |
| 20.261918s | **8.662308s** |

It can be seen from the above results that through express as much as possible in terms of matrix operations, the running time can be shortened by nearly three times, so we modify the training iterations to maxIter = 300000.

* 1. **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.

**answer:**

The regularization formula is as follows,

By implementing different , the following results can be obtained,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0.1 | 0.01 | 0.001 | 0.0001 | 0.00001 |
| Test error | 0.182000 | 0.115000 | 0.076000 | 0.116000 | 0.168000 |

As can be seen from the above results, proper lambda can effectively reduce test error, so select .

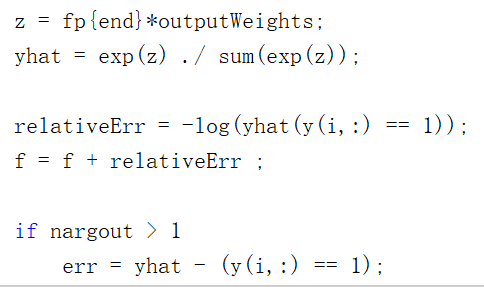
* 1. **Task5:** Instead of using the squared error, use a softmax (multinomial logistic) layer at the end of the network so that the 10 outputs can be interpreted as probabilities of each class.

**answer:**

The softmax function is,

Differentiate loss against y,

The code is,



modification

The result is as follows,

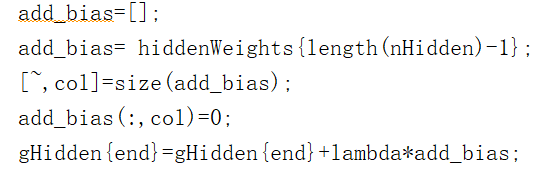
|  |  |  |
| --- | --- | --- |
|  | Before modification | After modification |
| Test error | **0.076000** | **0.052000** |

* 1. **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.

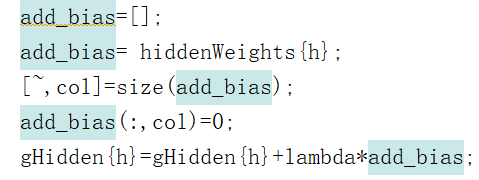
**answer:**

The formula for updating the offset parameter is as follows,

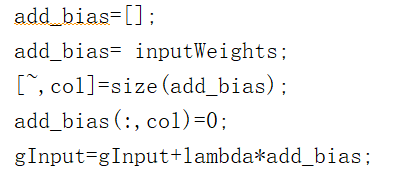
The code to add an offset to the hidden layer is as follows,



modification 1



modification 2



modification 3

The result is as follows,

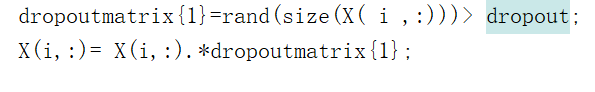
|  |  |  |
| --- | --- | --- |
|  | Before modification | After modification |
| Test error | **0.052000** | **0.046000** |
| After modification |  | |

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

**answer:**

Dropout can be used as a trick for training deep neural networks. In each training batch, by ignoring half of the feature detectors (leaving half of the hidden layer node value as 0), the phenomenon of overfitting can be significantly reduced. This approach can reduce the interaction between feature detectors (hidden nodes). Detector interaction means that some detectors rely on other detectors to function.

The code to add dropout is as follows,



modification 1







medication 2

The result is as follows,

|  |  |  |
| --- | --- | --- |
|  | Before modification | After modification |
| Test error | **0.046000** | **0.097000** |
| After modification |  | |

It can be seen from the above results that dropout will make the test error rise, but dropout can effectively alleviate the problem of overfitting. In order to get the best test error in this experiment, choose not to add dropout. Therefore, the code about dropout is in the “dropout” folder.

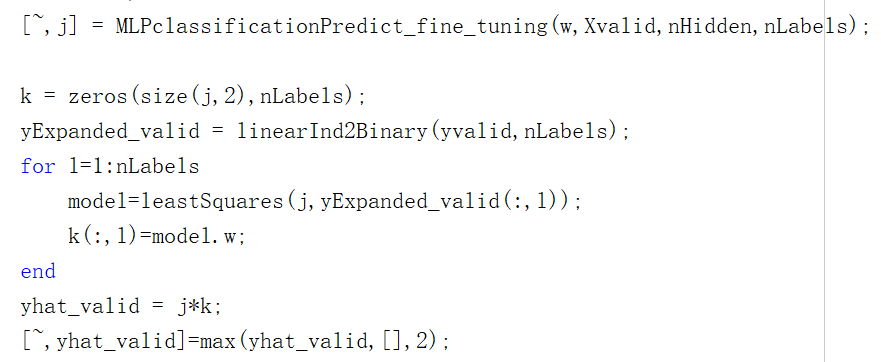
* 1. **Task8:** You can do ‘fine-tuning’ of the last layer.

**answer:**

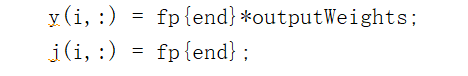
.Fix the parameters of all the layers except the last one, and solve

for the parameters of the last layer exactly as a convex optimization problem. The linear regression model has already existed in Sample code.

The code to add ‘fine-tuning’ is as follows,



modification 1



modification 2

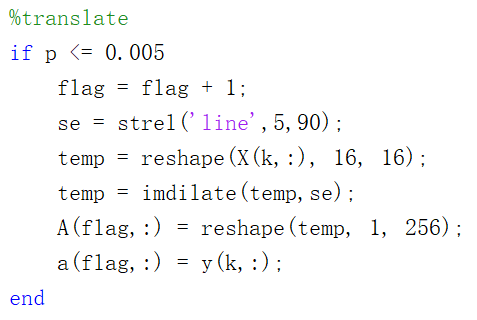
|  |  |  |
| --- | --- | --- |
| LS | Before modification | After modification |
| Test error | **0.048000** | **0.046000** |
| before modification |  | |
| After modification |  | |

It can be seen from the above results that although ‘fine-tuning’ is added, the test error does not decrease significantly. However, a better validation error can be obtained at the beginning of the experiment with ‘fine-tuning’. Considering that there is no obvious change in test error, this experiment did not use ‘fine-tuning’. Therefore, the code about fine-tuning is in the ‘fine-tuning’ folder.

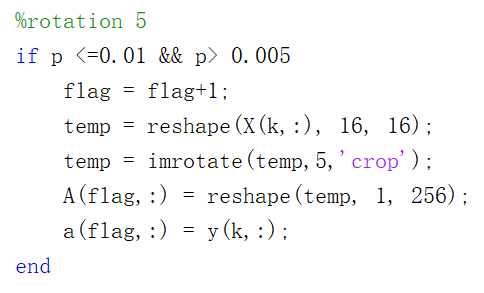
* 1. **Task9:** You can artificially create more training examples, by applying small transformations (translations, rotations, resizing, etc.) to the original images.

**answer:**

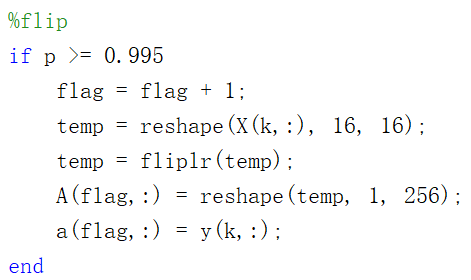
Translate, rotate and flip the picture matrix operation is very simple in the Matlab, which can be completed through the function. The specific operation is as follows,



translate



rotation



flip

In addition to increasing the training samples, I also increased the batch size to 3, which means,



The result is as follows,

|  |  |  |
| --- | --- | --- |
|  | Before modification | After modification |
| Test error | **0.046000** | **0.039000** |
| After modification |  | |

It can be seen that the validation error going up but the test error declining. which means the model is better to prevent overfitting.

* 1. **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.

**answer:**

The partial derivative of the convolution kernel of the loss function with respect to the layer is,

that is,

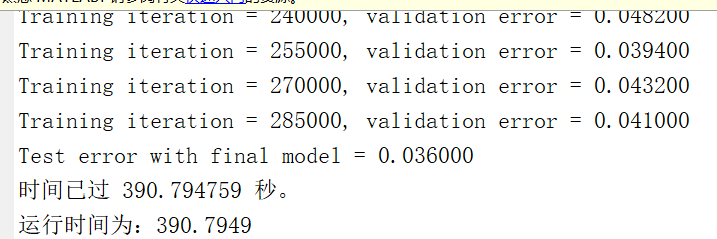
The code about convolution is in the “conv” folder. The result is as follows,

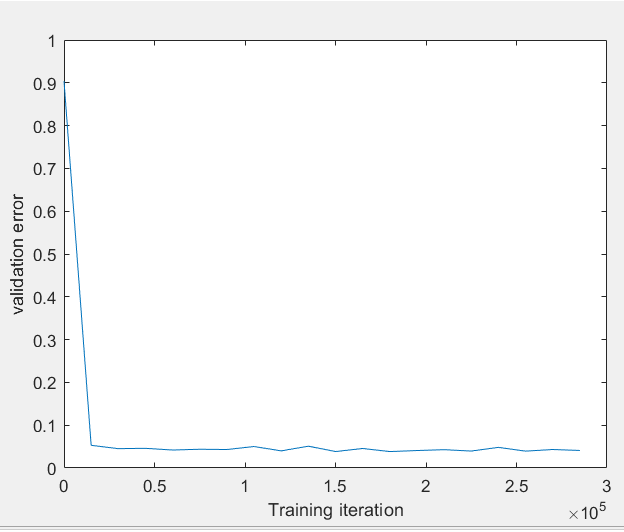
|  |  |  |
| --- | --- | --- |
|  | Before modification | After modification |
| Test error | **0.039000** | **0.036000** |
| After modification |  | |

In addition, at this time, the hidden layer nHidden = [100], stepSize = 4e-3

1. Conclusion

The best error is 0.036,

****

****