Project-1 of "Neural Network and Deep Learning"

Yanwei Fu

April 6, 2020

Abstract

(1) This is the first project of our course. The deadline is 5:00pm, April 30, 2020. Please send the report to dlcourse.xcm@gmail. (Chengming Xu)

(2) The goal of your write-up is to document the experiments you've done and your main findings. So be sure to explain the results. The report can be written by Word or Latex. Generate a single pdf file of your mini-projects and turned in along with your code. package your code and a copy of the write-up pdf document into a zip or tar.gz file and named as Project1-*your-student-id*_your_name.[zip|tar.gz]. Only include functions and scripts that you modified. Also put the names and Student ID in your paper.

(3) About the deadline and penalty. In general, you should submit the paper according to the deadline of each mini-project. The late submission is also acceptable; however, you will be penalized 10% of scores for each week's delay.

(4) **We provide the MATLAB sample codes, as the general reference; but you can use any other program languages, e.g., Python, C/C++, R, and Lua. Note that you can not invoke any additional deep learning functions.**

1 Neural Network

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. Your task in this question is to modify this training procedure architecture to optimize performance.

Report the best test error you are able to achieve on this dataset, and report the modifications you made to achieve this. Please refer to previous instruction of writing the report.

1.1 Hint

Below are additional features you could try to incorporate into your neural network to improve performance (the options are approximately in order of increasing difficulty). You do not have to implement all of these, the modifications you make to try to improve performance are up to you and you can even try things that are not on the question list in Sec. 1.2. But, let's stick with neural networks models and only using one neural network (no ensembles).

1.2 Questions (10 points each question)

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

done 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

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

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

done 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.

done 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.

done 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

$$p\left(y_i\right) = \frac{\exp\left(z_i\right)}{\sum_{j=1}^{J}\exp\left(z_j\right)}$$

you can replace squared error with the negative log-likelihood of the true label under this loss, $-\log p\left(y_i\right)$

done 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.

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

done 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).

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

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.