MNIST Data Classification with Neural Network

Abstract

A fully-connected neural network was implemented to classify MNIST data of handwritten digits. A total of 4000 images, 400 images for each digit, was used to train this network, and the classification accuracy of this network was tested on 2000 images, 200 images for each digit. This network, the activation functions in hidden layers of which are all sigmoid functions and the output layer of which is the Sofmax function, was trained with stochastic gradient descent, and all the gradients were computed via back propagation. With 2 hidden layers, the first of which consists of 25 activation functions and the second of which consists of 10 activation functions, the classification accuracy on the test set reaches 95.9%. The accuracy on the test set was further improved to 96.8% by reducing the number of units in the first layer to 80% and 97.7% via data augmentation, which increases the training and validation sets by perturbing each entry of the inputs of original data by $\pm 10\%$. Finally, using recurrent network work with long-short-term-memory cells improved the test accuracy to 98.4%.

1 Introduction

One way to reach the learning capability of human beings is to imitate the interconnected structure of natural neuronal networks (Figure 1). Shown in Figure 2, neural networks are interconnected in their hidden layers, which consist of activation functions, and make decisions in their output and loss layers, which correspond to cortices [2]. In this paper, a neural network has been built to classify images of hand-written digits from 0 to 9 (sample image of hand-written 0: Figure 3). The inputs of this neural network is generated as follows:

1. a 28×28 matrix is generated from the grey scale of the images;
2. this matrix is then vectorized into a vector of 1×784.

All the units in the hidden layers of this neural network are sigmoid functions. The m^{th} unit in the l^{th} hidden layer take the form of

$$\frac{1}{1+\exp\left(-\mathbf{w}_{lm}^{\top}\mathbf{z}_{lm}+b_{lm}\right)},$$

for which $\mathbf{w}_{lm}^{\top}$ denotes the unit's weight vector, $\mathbf{z}_{lm}$ the unit's input, and b_{lm} the unit's offset. Note that the number of units in the last hidden layer has to be 10, as there are 10 different labels in this study. All 10 outputs of the last hidden layer, z_j for $j = 1, \ldots, 10$, are sent to the output layer, which uses Softmax function:

$$f(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}},$$

MIT 9.520/6.860: Statistical Learning Theory and Applications (Fall 2016).

which describe the "probabilities" of each output that add up to 1. These "probabilities" are then sent to a cross-entropy loss layer:

$$L = -\sum_{i=1}^{10} y_i \log(f(z)_i),$$

for which y_i uses the label of the data point used for training and L quantifies the cost of misclassification. For example, if the digit is 1, $y_2 = 1$, and all the other y_i's are 0.

The primary goal of this neural network problem is to use the images of hand-written digits and their corresponding labels to optimize weight vectors and offsets of units in hidden layers by minimizing the cross-entropy loss for the given images:

$$\min_{\mathbf{w}'_{lm}s, \mathbf{b}'_{lm}s} L(\mathbf{w}'_{lm}s, \mathbf{b}'_{lm}s, \mathbf{x}, \mathbf{y}), \tag{1}$$

for which $\mathbf{x}$ and $\mathbf{y}$ denote the input and output of given data. The solution to (1), at the t^{th} iteration, is obtained by gradient descent,

$$\mathbf{w}_{lm}^{t+1} = \mathbf{w}_{lm}^{t} - \gamma^t \frac{\partial L}{\partial \mathbf{w}_{lm}}\Big|_{\mathbf{w}_{lm}=\mathbf{w}_{lm}^t}, \tag{2}$$

where γ^t is the step size that can be a function of t. One major challenge is the computation of $\frac{\partial L}{\partial \mathbf{w}_{lm}}$, which can obtained from chain rule or back propagation:

$$\frac{\partial L}{\partial \mathbf{w}_{lm}} = \sum_{i=1}^{k}\sum_{j=1}^{k} \frac{\partial L}{\partial f(z)_i}\frac{\partial f(z)_i}{\partial z_j}\frac{\partial z_j}{\partial \mathbf{w}_{lm}},$$

$$\frac{\partial L}{\partial \mathbf{b}_{lm}} = \sum_{i=1}^{k}\sum_{j=1}^{k} \frac{\partial L}{\partial f(z)_i}\frac{\partial f(z)_i}{\partial z_j}\frac{\partial z_j}{\partial b_{lm}},$$

for which

$$\frac{\partial L}{\partial f(z)_i} = -\frac{y_i}{f(z)_i},$$

$$\frac{\partial f(z)_i}{\partial z_j} = \begin{cases} f(z)_i(1 - f(z)_i) & \text{if } i = j, \\ -f(z)_i f(z)_j & \text{otherwise.} \end{cases}$$

One problem with a neural network with many units in the hidden layers, is the large number of parameters that need to be optimized, which leads to two issues, inability to reach convergence and being stuck at a local minimum. These two issues are first addressed by properly initializing the weight vectors and offsets. For a hidden layers with n inputs and m units, the coefficient matrix of this layer, the columns of which are the weight vectors and offsets of each unit dimension and the dimension of which is therefore $(n + 1) \times m$, is randomly sampled from a Gaussian with zero mean and $1/\sqrt{(n + 1)}$ standard deviation. The advantages of this random sampling are

1. large rate of change initially since the largest rate of change for the sigmoid function $\frac{1}{1+\exp(-t)}$ is at $t = 0$ (zero mean);
2. the summation of the variance for each column of the coefficient matrix is 1, independent of the number of inputs and units ($1/\sqrt{(n + 1)}$ standard deviation);
3. breaking symmetry from random sampling.

Another way to address the issue of being stuck at a local minimum is to calculate $\frac{\partial L}{\partial \mathbf{w}_{lm}}\Big|_{\mathbf{w}_{lm}=\mathbf{w}_{lm}^t}$ one data point for each iteration (referred to as "stochastic gradient descent"). Stochastic gradient descent also cost less computationally. As a result of using stochastic gradient descent, the step size γ^t generally should be a function of t, and the specific function form varies between cases and has been chosen based on trial-and-error, $0.5/(t + 1/500)^{0.505}$.

During the update of coefficient matrices based using 2000 data points (200 data points for each digit), two types of quantities were monitored, the changes in the magnitudes of weight matrices for

all hidden layers and the errors on the training and validation sets, which also contains 2000 data points (200 data points for each digit). Once the changes in the magnitude of weight matrices and the decrease in validation error are no greater than 10^{-6}, the update is stopped (Figure 4).

With the trained weight matrices, the inputs of the test data were sent to the network, the Softmax layer of which outputs the "probability" vector of 1×10. The guess is the label with the largest "probability". For example, if the output is $[0.0, 0.1, 0.3, 0.5, 0.1, 0.0, 0.05, 0.05, 0.0, 0.0]^\top$, the guess for this test data will be 3.

2 Results & discussions

The optimal structure of the hidden layers in the neural network was determined by empirically minimizing the classification accuracy on the test set. As shown in Table 1, the optimal structure of the hidden layers is $[25, 10]$, meaning 25 sigmoid functions for the first hidden layer and 10 sigmoid functions for the second hidden layer. The test accuracy of this structure is 95.9%. Note that increasing the number of hidden layers from 2 to 3 may not improve the test accuracy, as the added number of undetermined parameters may lead to over-fitting.

To further check if there is over-fitting, the number of units in the first hidden layer was reduced by different percentages, and the test accuracies were compared (Figure 5). The optimal number of units in the first layer is 20. The test accuracy is slightly increased to 96.8%.

To increase the robustness of the neural network, both the training and validation sets were enlarged via data augmentation. Specifically, in addition to the original data points, additional data points were generated by changing each entry of the inputs by $\pm 10\%$. With additional data points, the test accuracy was increased to 97.7%.

One more thing explored is to use recurrent neural network with long-short-term-memory cells that remember values for short or long time (Figure 6). With 2 hidden layers of $[25, 10]$, the test accuracy was improved to 98.4%.

3 Future directions

Figure 7 is one example of mis-classifying 9 as 0. The issue is that the tail of this 9 is too small such that it is classified as 0. One way to take into consideration the features of local structures is to use convolutional neural network.

Structure of neural network	Classification accuracy (test)
$[10, 10]$	92.0%
$[25, 10]$	95.9%
$[25, 10, 10]$	94.2%

Table 1: Determination of the optimal structure of hidden layers

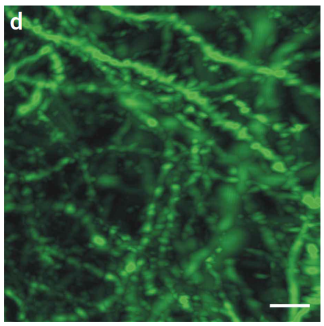

Figure 1: 3D reconstruction of neuronal network in a mouse's dendritic spines [1].

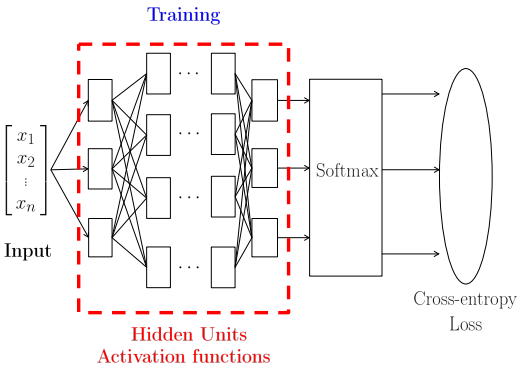

Figure 2: Fully-connected neural net.

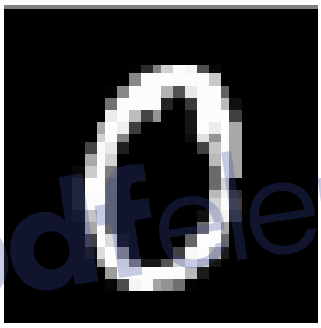

Figure 3: A sample image from MNIST data for hand-written 0.

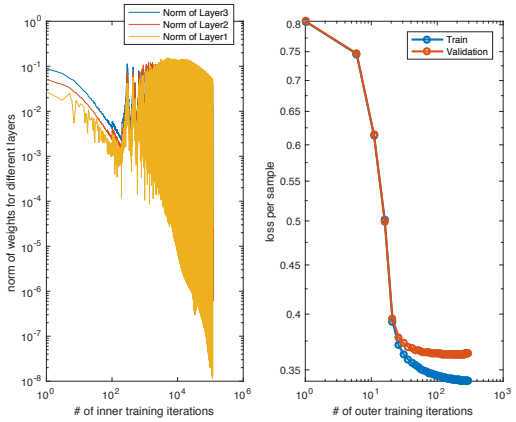

Figure 4: Early stopping: the changes in the magnitude of weight matrices and the decrease in validation error are no greater than 10^{-6}.

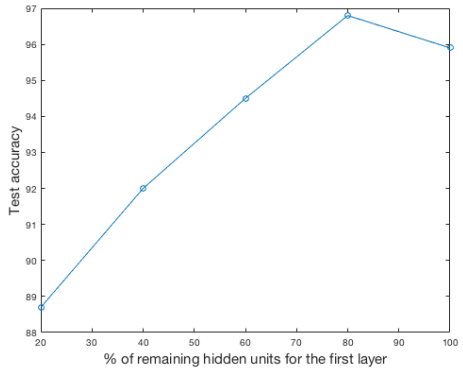

Figure 5: Optimal dropout rate for the first layer of a two-layer neural net: the maximum number of activation units of the first layer is 25; the second layer has 10 activation units.

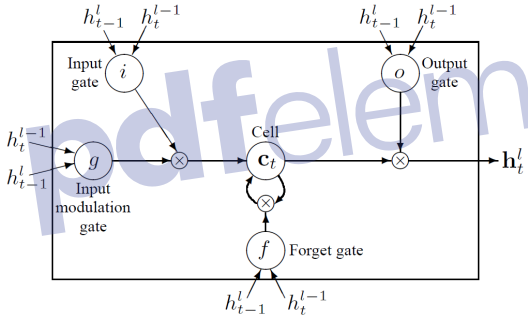

Figure 6: Long-short-term memory cell [3].

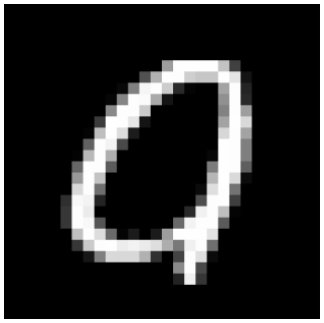

Figure 7: One example of mis-classification for a neural network with hidden layer $[20, 10]$.

References

[1] Dodt, Hans-Ulrich, Ulrich Leischner, Anja Schierloh, Nina Jährling, Christoph Peter Mauch, Katrin Deininger, Jan Michael Deussing, Matthias Eder, Walter Zieglgänsberger, and Klaus Becker. "Ultramicroscopy: three-dimensional visualization of neuronal networks in the whole mouse brain." *Nature Methods* 4, no. 4 (2007): 331-336.

[2] Walton, Mark E., Joseph T. Devlin, and Matthew FS Rushworth. "Interactions between decision making and performance monitoring within prefrontal cortex." Nature Neuroscience 7.11 (2004): 1259-1265.

[3] Zaremba, Wojciech, Ilya Sutskever, and Oriol Vinyals. "Recurrent neural network regularization." arXiv preprint arXiv:1409.2329 (2014).