Neural Network Classification

Siwen Yan
A53093655
siy043@ucsd.edu

Xiyun Liu
A53099348
xil429@ucsd.edu

Abstract

Multi-layer Neural Networks is a good method to do classification. In this assignment, handwritten digits from Yann LeCun's MNIST Database are classified using multi-layer neural networks with softmax outputs. By implementing the traditional two-layer neural networks, we got a 96.133% accuracy on the test dataset. After applying the tricks including shuffling the data, using mini-batches, do normalization, use momentum and input weights initialization, we finally got 99.535% accuracy on the test dataset. Network topology is also investigated in this assignment.

1 Introduction

Multi-layer neural networks is one of a good method to do classification. Softmax regression with cross-entropy error is used as the objective function in the single-layer neural network. In two-layer neural network, a hidden layer between the input and output is added with the sigmoid activation function.

In this assignment, we first implemented the classical two-layer neural network, and then added some tricks to improve the performance. Additionally, network topology is experimented and the performance is compared and analyzed.

2 Methods

2.1 Two-layer neural network

Our goal is to classify the handwritten digits from the MNIST dataset. Softmax activation function is used at the output level and logistic activation function is used at the hidden layer.

To eliminate the influence of the size of the data set, we can get the average of the softmax cross-entropy loss function:

$$E = -\frac{1}{N}\sum_{n}\sum_{k=1}^{c} t_k^n ln y_k^n$$

For hidden layer, we first use the logistic

$$y_j = 1/(1 + exp(a_j)) \tag{1}$$

This has the nice feature that $dy/da = y(1y)$. In the part "Add Tricks of the Trade", we will use tanh function to improve the model.

For now, first compute the update rule for each layer's weights. From the Bishop part in this homework assignment, we have derive the expression for δ_j and δ_k using the definition of δ as $\delta_i = -\frac{\partial E}{\partial a_i}$. Then we get the update rule for each weight

$$w_{jk} = w_{jk} + \eta(t_k - y_k)z_j \tag{2}$$
$$w_{ij} = w_{ij} + \eta g(a_j)(1 - g(a_j))\Sigma_k(t_k - y_k)w_{jk}x_i \tag{3}$$

where w_{jk} is the weights from hidden layer to output layer and w_{ij} is the weights from input to hidden layer.

Vectorize computation can make the update faster in our program. Let w_{HO} be a $j \times k$ metric represents the weights from hidden units to output, $W_{input\ to\ hidden}$ be a $i \times j$ metric represents the weights from input to hidden units, $\vec{x}$ is the input column vector , $\vec{t}$ is the k target column vector, $\vec{y}$ is the k predict column vector, $\vec{G}$ is the element-wise product of $g(a_j)$ and $1 - g(a_j)$, then

$$w_{HO} = w_{HO} + \eta[(\vec{t} - \vec{y}) \cdot \vec{z}^T]^T \tag{4}$$
$$w_{IH} = W_{IH} + \eta\vec{x} \cdot (G * (\vec{t} - \vec{y}) \cdot w_{HO}^T) \tag{5}$$

where * means element-wise product

2.2 Gradient check

To check the correctness of the calculation of the gradients, we could compute the slope with respect to one weight using the numerical approximation

$$\frac{d}{dw}E^n(w) \simeq \frac{E^n(w + \epsilon) - E^n(w - \epsilon)}{2\epsilon} \tag{6}$$

where ϵ is small constant, e.g., 10^{-2}. Compare the gradient computed using numerical approximation with the one computed as in backpropagation. The difference of the gradients should be within big-O of ϵ^2, so if you used 10^{-2}, your gradients should agree within 10^{-4}.

2.3 Tricks of the Trade

There are a few tricks that can be added to improve the model performance. First data is shuffled, and then use mini-batch stochastic gradient descent. Stick with the normalization we have already done in the previous part. Converge could be faster if the average of each input variable over the training set is close to zero. For the output layer, stick with the softmax activation function, since the sigmoid function is symmtric about the origin. However, for the hidden layer, logistic function whose outputs are always positive and so have a positive mean. Therefore, we use

$$f(x) = 1.7159\ tanh(\frac{2}{3}x) \tag{7}$$

as the hidden layer activation function.
The input weights to each unit is initialized under Gaussian distribution with 0 mean and 1/sqrt(fan-in) standard deviation, where the fan-in is the number of inputs to the unit.
Momentum is also used here. If the weight change is written as $\Delta w(t)$, the weight change at the next step should be $\Delta w(t+1) = \mu\Delta w(t) - \eta\frac{\partial E}{\partial w}$. In this assignment, we set μ as 0.9.

3 Results

3.1 Gradient check

The gradient check applies on each weight of each layer. We use for loops to go through each one weight among w_{ij} and w_{jk}, change it with ϵ, calculate the numerical approximation of the gradient and compare with the one computed in backpropagation. Here, only one input and output pattern is used and it is at the first iteration. ϵ is set to 0.01. The mean and maximum difference over all weights are shown in table 1.

	mean of difference	maximum difference
w_{ij}	1.191637 e-08	3.315017 e-07
w_{jk}	1.596774 e-07	1.602811 e-06

Table 1: Gradient difference

The table shows that the gradient computed using numerical approximation and the one computed as in backpropagation agree within 10^{-6} which is ϵ^3, which prove that the gradient computed in backpropagation is correct.

3.2 Two-layer Neural Network Performance

First do normalization on the data. Since the pixel values in the digit images are in the range [0..255], first divide by 255 and then subtract the mean over all of the pixels in each image.

The training set from MNIST is seperated into two part, one training set with 50000 training images and one validation set with 10000 images. The weights are initialized under Gaussian distribution with mean of 0 and variance of 0.1. Early stopping is used by tracking the error of validation set. The learning rate is "annealed" by reducing it over time. Experiment is need to find the proper initial learning rate and metaparameter T. The initial learning rate is set to 3 and T set to 500.

Since we found it is really hard to reach overfitting to early stopping it based on several experiment, we just terminate the program after training through the whole dataset 1000 times.

The percent correct classification on training set, validation set and test set are shown in table 2 and figure 1.

iteration	training	validation	test
100	0.91362	0.9191	0.91453
200	0.93022	0.9347	0.93096
300	0.94044	0.9437	0.94098
400	0.9462	0.9483	0.94655
500	0.95028	0.9518	0.95053
600	0.9535	0.9546	0.95368
700	0.95592	0.9555	0.95585
800	0.9581	0.9575	0.958
900	0.95996	0.9591	0.95982
1000	0.9617	0.9595	0.96133

Table 2: Percent correctness over training

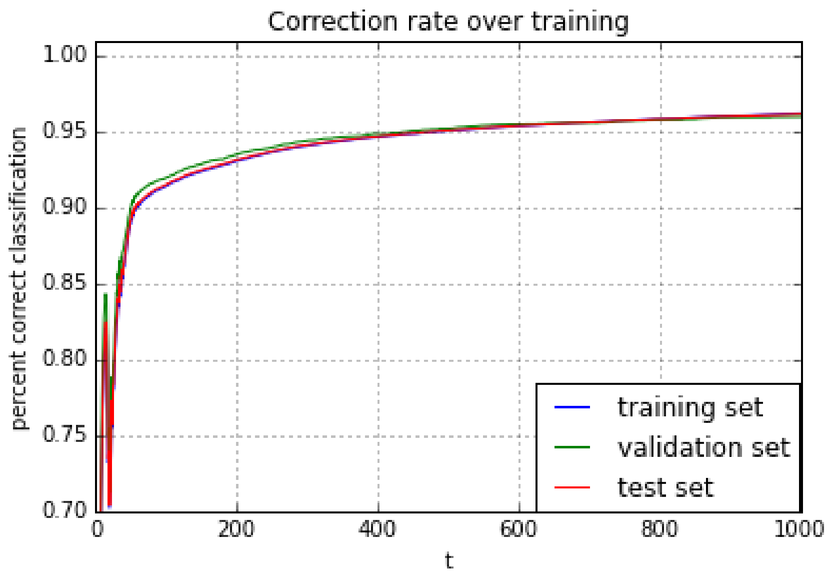

Figure 1: Percent correctness over training

3.3 Adding the Tricks of the Trade

First, all the tricks are applied except mini-batch stochastic gradient descent, which means we use whole batch to train in each iteration. The percent accuracy is shown below. The initial learning rate is 1, and T = 700. The final test correction rate is 99.535%. Also, the percent correct classification over training on training set, validation set and test set are shown in table 3 and figure 2.

Using 10000-batch stochastic gradient descent, after several experiment, we set the initial learning rate to 0.2 and T is 50, then got the following result. It converges at 182 and the final correction rate is 0.86255. The percent correct classification on training set, validation set and test set are shown in table 4 and figure 3.

Using 5000-batch stochastic gradient descent, after several experiment, we set the initial learning rate to 0.03 and T is 8, then got the following result. It converge at 135 final correction rate is 0.73192. The percent correct classification on training set, validation set and test set are shown in table 5 and figure 4.

iteration	test
10	0.873666666667
20	0.911383333333
30	0.930683333333
40	0.943116666667
50	0.95125
60	0.95695
70	0.961166666667
80	0.9647
90	0.967433333333

iteration	training	validation	test
100	0.97118	0.9645	0.970066666667
200	0.9862	0.9694	0.9834
300	0.99332	0.9716	0.9897
400	0.99672	0.9721	0.992616666667
500	0.99828	0.9727	0.994016666667
600	0.99902	0.973	0.994683333333
700	0.9994	0.973	0.995
800	0.99966	0.9735	0.9953
900	0.99976	0.9734	0.995366666667
1000	0.99984	0.9729	0.99535

Table 3: Percent correctness over training on whole batch with tricks

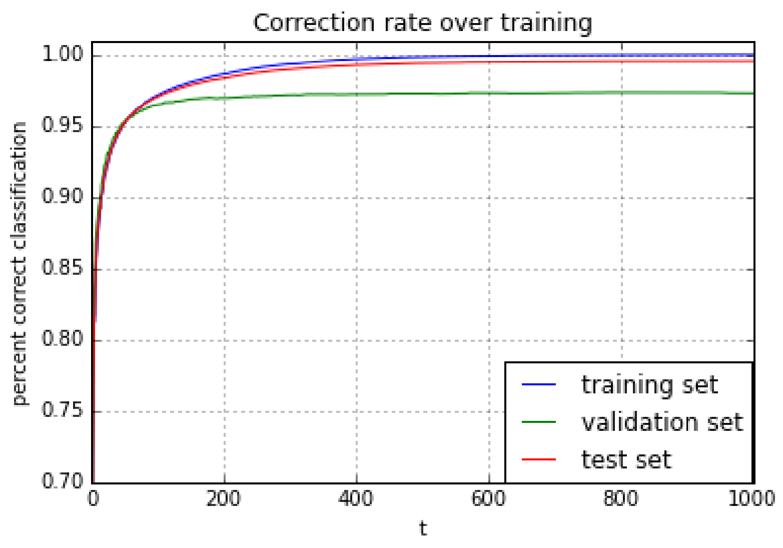

Figure 2: Percent correctness over training on whole batch with tricks

iteration	training	validation	test
10	0.82214	0.8185	0.821533333333
30	0.83138	0.8246	0.83025
50	0.84026	0.8328	0.839016666667
70	0.85026	0.8407	0.848666666667
90	0.85772	0.8492	0.8563
110	0.86232	0.8535	0.86085
130	0.86426	0.854	0.86255
150	0.86426	0.8542	0.862583333333
170	0.86418	0.854	0.862483333333

Table 4: Percent correctness over training using mini-batch size of 10000 with other tricks

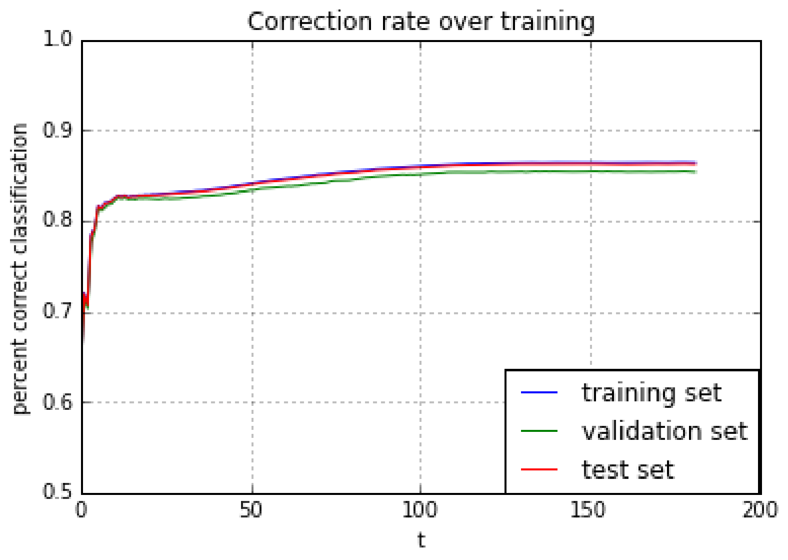

Figure 3: Percent correctness over training using mini-batch size of 10000 with other tricks

iteration	training	validation	test
10	0.69	0.6909	0.69015
30	0.7247	0.7224	0.724316666667
50	0.7309	0.7275	0.730333333333
70	0.7324	0.7292	0.731866666667
90	0.73266	0.7308	0.73235
110	0.73296	0.7305	0.73255
130	0.73254	0.7301	0.732133333333

Table 5: Percent correctness over training using mini-batch size of 5000 with other tricks

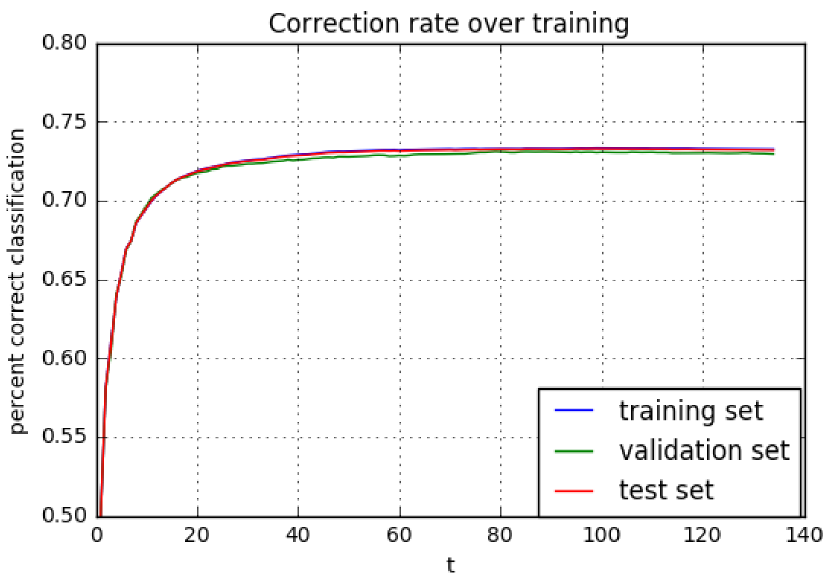

Figure 4: Percent correctness over training using mini-batch size of 5000 with other tricks

3.4 Experiment with Network Topology

Change the number of hidden layers from one to two with the same size of 30. Implement this new architecture and the percent correct classification on training set, validation set and test set are shown in table 6 and figure 5.

Then, we change the number of hidden layers. Keep using the initial learning rate of 1 and T of 700, first we decreased the hidden layer units from 60 to 30. The percent correct classification on training set, validation set and test set are shown in table 7 and figure 6.

Similar, doubling the hidden layer unit from 60 to 120 and the percent correct classification over training are shown in table 8 and figure 7.

iteration	training	validation	test
100	0.95554	0.9457	0.9539
200	0.96692	0.9513	0.964316666667
300	0.97224	0.9528	0.969
400	0.97562	0.9541	0.972033333333
500	0.97824	0.9551	0.974383333333
600	0.98042	0.9557	0.9763
700	0.98216	0.9567	0.977916666667
800	0.98342	0.9564	0.978916666667
900	0.98452	0.9566	0.979866666667
1000	0.98544	0.9568	0.980666666667

Table 6: Percent correctness over training using Three-layer Neural Network

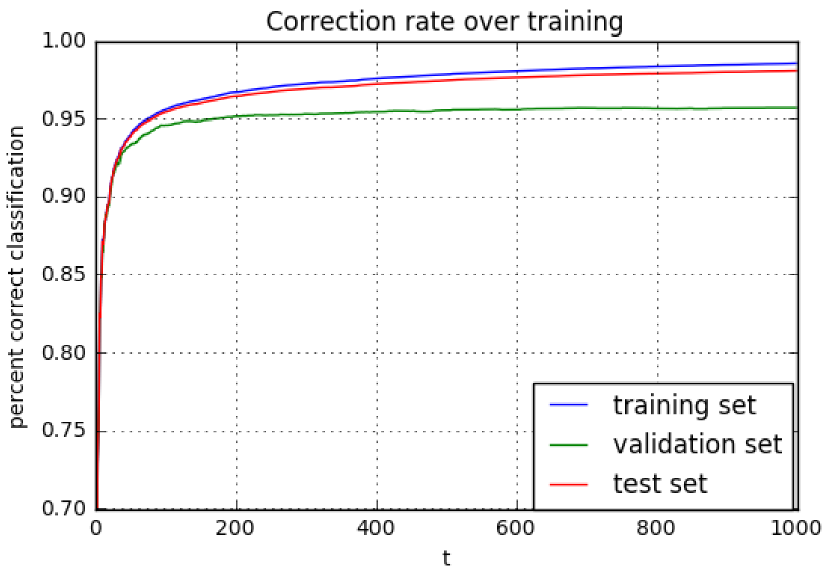

Figure 5: Percent correctness over training using Three-layer Neural Network

iteration	test
10	0.857083333333
20	0.899933333333
30	0.918733333333
40	0.92945
50	0.936966666667
60	0.942183333333
70	0.946416666667
80	0.949883333333
90	0.95236666

iteration	training	validation	test
100	0.9552	0.9504	0.9544
200	0.9692	0.9576	0.967266666667
300	0.97706	0.9592	0.974083333333
400	0.98158	0.9599	0.977966666667
500	0.98538	0.9607	0.981266666667
600	0.98816	0.96	0.983466666667
700	0.99012	0.9599	0.985083333333
800	0.99166	0.9589	0.9862
900	0.9927	0.9591	0.9871
1000	0.99374	0.9591	0.987966666667

Table 7: Percent correctness over training with 30 unit

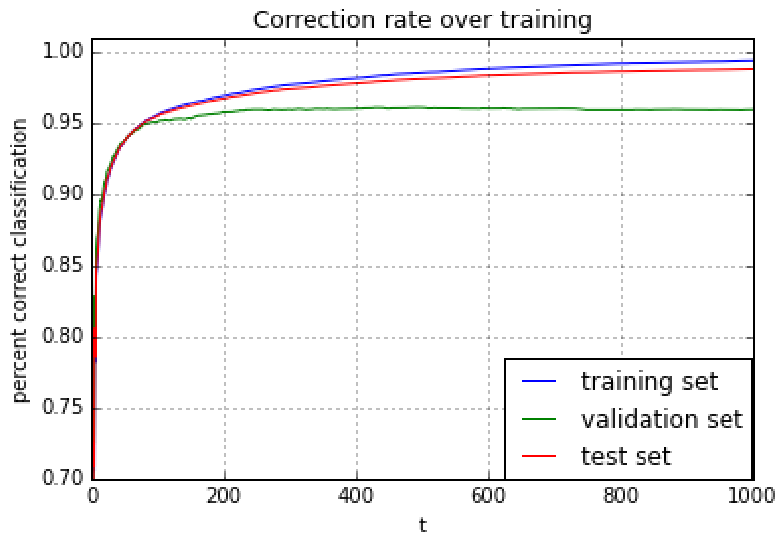

Figure 6: Percent correctness over training with 30 unit

iteration	test
10	0.633933333333
20	0.862633333333
30	0.9024
40	0.936616666667
50	0.951666666667
60	0.959616666667
70	0.965383333333
80	0.969933333333
90	0.973116666667

iteration	training	validation	test
100	0.97782	0.9618	0.97515
200	0.99186	0.9682	0.987916666667
300	0.99654	0.9698	0.992083333333
400	0.99836	0.9703	0.993683333333
500	0.99924	0.9704	0.994433333333
600	0.99958	0.9703	0.9947
700	0.99982	0.971	0.995016666667
800	0.99992	0.9711	0.995116666667
900	0.99996	0.9711	0.99515
1000	0.99998	0.9714	0.995216666667

Table 8: Percent correctness over training with 120 unit

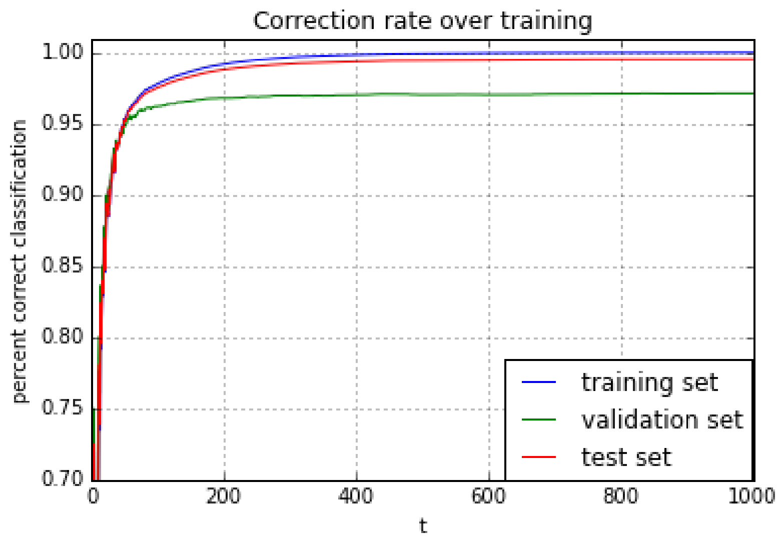

Figure 7: Percent correctness over training with 120 unit

4 Discussion

4.1 Add the Tricks of the Trade

We first add all the tricks expect stochastic gradient descent. Using whole batch, we get a test error rate of 0.465% which is pretty good and actually lower than other results quoted on the MNIST web site. Compared figure 2 with figure 1, we could tell that the tricks works well and do improve the two-layer neural network's accuracy from 96.133% to 99.535% on MNIST dataset.

However, if we use mini-batch stochastic gradient descent, the accuracy is decreased by comparing figure 3 & 4 with figure 2.

4.2 Network Topology

When halving the the hidden layer nodes, the increasing of correction rate is much slower than using 60 units, and the training set accuracy and the test set accuracy after 1000 iteration both decrease. Noted that from the figure we could see that there is a drop on the validation accuracy's curve at 643 iteration, but we didn't terminate the program since we want to compare it with the previous best figure.

When we doubling the hidden layer nodes to 120, the time of each iteration increases.During training, its accuracy increases to 0.994 a little faster than using 60 units, however, the final accuracy is a little lower than using 60 hidden units.

Comparing the above graphs, we could see that know the the performance for the training accuracy and the testing accuracy remain similar with single hidden layer.

5 Conclusion

We classified handwritten digits from Yann LeCun's MNIST Database using multi-layer neural networks with softmax outputs. The traditional two-layer neural networks got a 96.133% accuracy on the test dataset. After doing several experiment on applying the tricks including shuffling the data, using mini-batches, do normalization, use momentum and input weights initialization, we finally got 99.535% accuracy on the test dataset. Increasing the hidden layer from one to two, we got similar results, but using longer time. By halving and doubling the number of hidden units, we found that 60 hidden units are the most suitable one for this problem.

6 Contributions

We work through all the problems together, doing pair programming and adjusting parameters together.

References

[1] LeCun, Yann A., et al. (2012) Efficient backprop." Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48.

7 Appendix

All the code included in the next few pages.