CMSC 25400 Machine Learning
Homework 6

Deqing Fu

November 30, 2017

1 Description of Code

1.1 Neural Network

I use the Neural Network with 1 hidden layer $\mathbf{x}_{\text{hidden}} \in \mathbb{R}^{N_{\text{hidden}}+1}$ and the input layer $\mathbf{x}_{\text{input}} \in \mathbb{R}^{N_{\text{input}}+1}$, where the addition 1 is the bias neuron. And the output layer is $\mathbf{x}_{\text{output}} \in \mathbb{R}^{N_{\text{output}}}$. Now we denote, $\mathbf{x}_0 = \mathbf{x}_{\text{input}}, \mathbf{x}_1 = \mathbf{x}_{\text{hidden}}, \mathbf{x}_2 = \mathbf{x}_{\text{output}}$.
And I use the activation function, the Sigmoid function

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

For the holdout set, we choose the the size of the set being $\frac{N_{\text{train}}}{5}$, where N_{train} is the size of the training set.

1.2 Data Structure

Here I talk about the weight matrices. There are 2 weight matrices. $W_1 \in \mathbb{R}^{(N_{\text{input}}+1)\times(N_{\text{hidden}}+1)}$ is the matrix that transforms the input layer to the hidden layer and $W_2 \in \mathbb{R}^{(N_{\text{hidden}}+1)\times(N_{\text{output}})}$ is the matrix that transforms the hidden layer to the output layer. More specifically,

$$W_1 = \begin{pmatrix} \mathbf{w}_1^{(1)} & \mathbf{w}_2^{(1)} & \cdots & \mathbf{w}_{N_{\text{hidden}}+1}^{(1)} \end{pmatrix}$$

where $\mathbf{w}_i^{(1)} \in \mathbb{R}^{N_{\text{input}}+1}$ are column vectors.

$$W_2 = \begin{pmatrix} \mathbf{w}_1^{(2)} & \mathbf{w}_2^{(2)} & \cdots & \mathbf{w}_{N_{\text{output}}}^{(2)} \end{pmatrix}$$

where $\mathbf{w}_i^{(2)} \in \mathbb{R}^{N_{\text{hidden}}+1}$ are column vectors.

As we know that for neuron y_i in the next layer and $\mathbf{x}$ being a column vector, the previous layer, and W_m is the corresponding weight matrix, we have

$$y_i = \phi(\mathbf{x}^T \mathbf{w}_i^{(m)}) = \phi(\mathbf{x}^T W_{*,i})$$

Hence,

$$\mathbf{x}_{k+1}^T = \phi(\mathbf{x}_k^T W_m)$$

To optimize redundant transpositions, we initialize each layer vector to be a row vector. Then we have

$$\mathbf{x}_1 = \phi(\mathbf{x}_0 W_1)$$
$$\mathbf{x}_2 = \phi(\mathbf{x}_1 W_2)$$

1.3 Loss Function

I use the squared error loss function, where $\mathbf{x} \in \mathbb{R}^{N_{\text{output}}}$ is the output of the output layer.

$$\varepsilon = ||\mathbf{x} - \mathbf{e}_{\mathbf{y}}|| = \sum_{i=1}^{N_{\text{output}}} (x_{\tau(i)} - y_i)^2$$

1.4 Stochastic Gradient Descent

We know the general gradient descent mechanism is:

$$w_{s\to t} \leftarrow w_{s\to t} - \eta \frac{\partial \varepsilon}{\partial w_{s\to t}}$$

Moreover,

$$\eta \frac{\partial \varepsilon}{\partial w_{s\to t}} = \frac{\partial \varepsilon}{\partial a_t} \frac{\partial a_t}{\partial w_{s\to t}} = \delta_t x_s$$

where x_s is the s neuron of the previous layer and a_t is the value of the t neuron before activation of this layer. As we choose the loss function as $\varepsilon = \sum_{i=1}^{N_{\text{output}}} (x_{\tau(i)} - y_i)^2$, then,

$$\delta_t = \frac{\partial \varepsilon}{\partial a_t} = 2(x_t - y_i)\phi'(a_t)$$

Moreover,

$$\delta_t = \phi'(a_t) \sum_{u \in \mathcal{O}(t)} w_{t\to u} \delta_u$$

Hence, when vectorize everything, we have $\delta_2 \in \mathbb{R}^{N_{\text{output}}}$ and $\delta_1 \in \mathbb{R}^{N_{\text{output}}+1}$. And they are both row vectors. More specifically, (where $*$ is the operation of element-wise multiplication)

$$\delta_2 = 2\left[\phi'(\mathbf{x}_2) * (\mathbf{x}_2 - \mathbf{e}_{\hat{\mathbf{y}}})\right]$$
$$\delta_1 = \phi'(\mathbf{x}_1) * (W_2 \delta_2^T)^T$$

Hence, the weights are updated as follows:

$$W_2 \leftarrow W_2 - \eta \mathbf{x}_1^T \delta_2$$
$$W_1 \leftarrow W_1 - \eta \mathbf{x}_0^T \delta_1$$

1.5 Pseudocode

Algorithm 1 Training Artificial Neural Network

Let $S = \{(x, y)\}$ be the training set with cardinality N_{train}.
Initialize weight matrices W_1 and W_2.
Let ϕ be the Sigmoid function and can be applied to tensors element-wisely.
for $ep = 1$ to $epoch$ **do**
    for $i = 1$ to N_{train} **do**
        $\mathbf{x}_0 = S[i][x].append([1])$ ▷ Input Layer, and we append a bias neuron
        $\mathbf{x}_1 = \phi(\mathbf{x}_0 W_1)$ ▷ Hidden Layer
        $\mathbf{x}_2 = \phi(\mathbf{x}_1 W_2)$ ▷ Output Layer
        $\mathbf{y} = \mathbf{e}_{S[i][y]}$ ▷ Label of the image
        $\varepsilon = \mathbf{x}_2 - \mathbf{y}$ ▷ Error
        $\delta_2 = 2\left[\phi'(\mathbf{x}_2) * \varepsilon\right]$ ▷ Operation $*$ is the element-wise multiplication
        $\delta_1 = \phi'(\mathbf{x}_1) * (W_2 \delta_2^T)^T$
        $W_2 \leftarrow W_2 - \eta \mathbf{x}_1^T \delta_2$ ▷ Updating Weight
        $W_1 \leftarrow W_1 - \eta \mathbf{x}_0^T \delta_1$ ▷ Updating Weight
    end for
end for

1.6 Decisions and Results

1.6.1 Initialization of Weights

To avoid situations that the first few rounds are wasteful due to the initialization of weights are too large and lead to the saturation of the sigmoid function. I normalize the weights during initialization. i.e. for i, j, $\mathbf{w}_i^{(j)}$ is normalized, that is $\sum_k \left(\mathbf{w}_i^{(j)}\right)_k = 1$.

1.6.2 Parameters and Results

Fixed parameters are

$$\begin{cases} N_{\text{input}} = 784 \\ N_{\text{output}} = 10 \end{cases}$$

After days of experiments (will discuss in the next section), I Choose

$$\begin{cases} N_{\text{hidden}} = 256 \\ \eta = 0.1 \\ epoch = 55 \end{cases}$$

And this will reduce the testing error to be

$$\varepsilon_{test} = 1.9\%$$

2 Performance Study

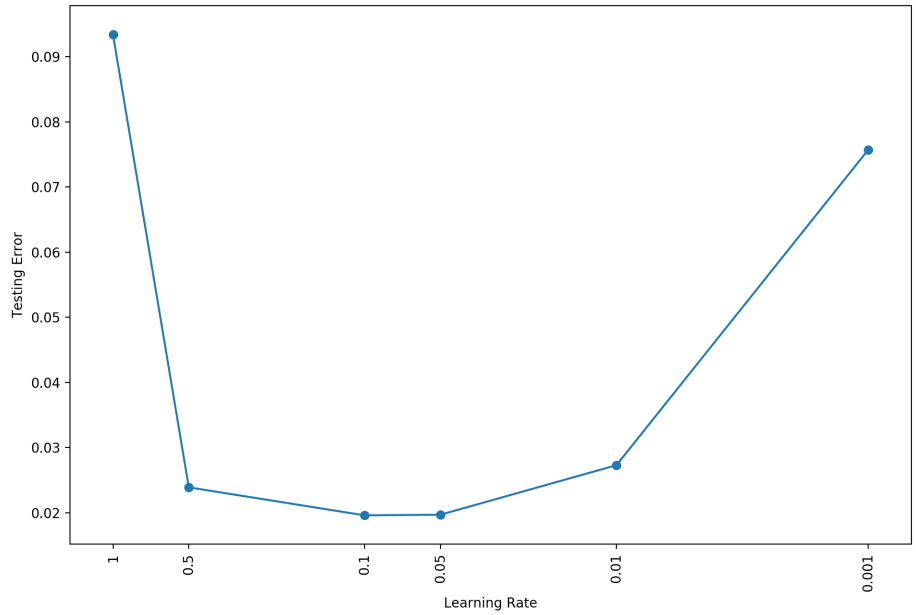
2.1 Learning Rate

Fixing $N_{\text{hidden}} = 256$ and $epoch = 40$

Table 1: Experiments on Learning Rate

Learning Rate	Testing Error
1	9.34%
0.5	2.39%
0.1	1.96%
0.5	1.97%
0.01	2.73%
0.001	7.57%

Figure 1: Experiment Figure on Learning Rate



We can see from either the table or the figure that we'd better choose the Learning Rate:

$$\eta = 0.1$$

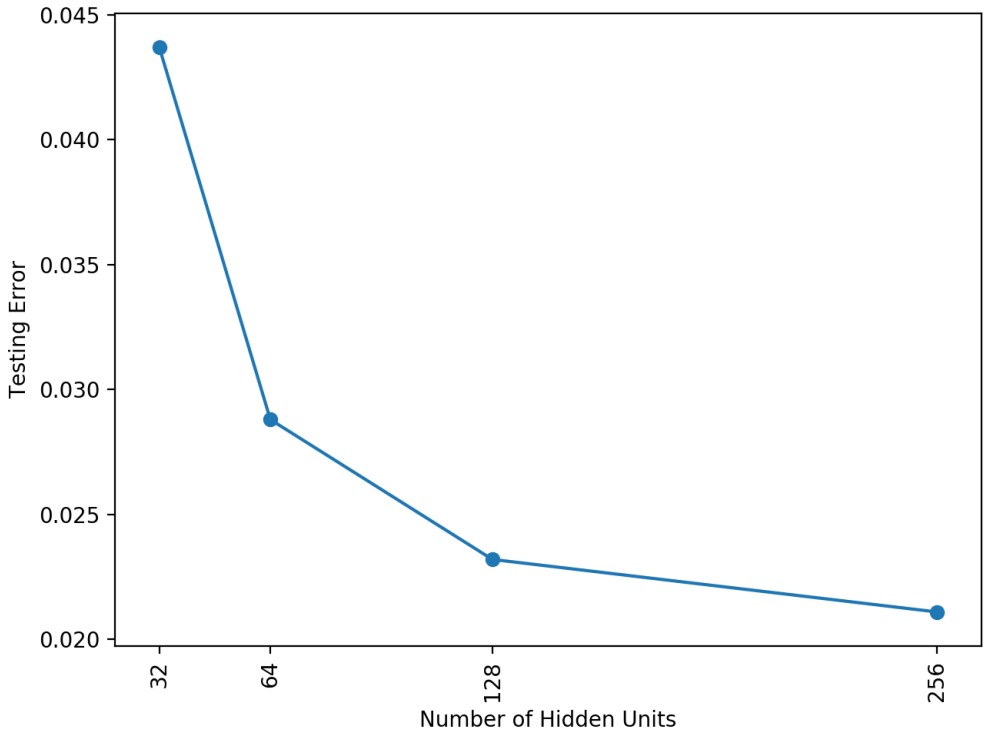
2.2 Number of Hidden Neurons

Fixing *epoch* = 40 and $\eta = 0.1$

Table 2: Experiments on Number of Hidden Neurons

Number of Hidden Neurons	Testing Error
32	4.37%
64	2.88%
128	2.32%
256	2.11%

Figure 2: Experiment Figure on Number of Hidden Neurons



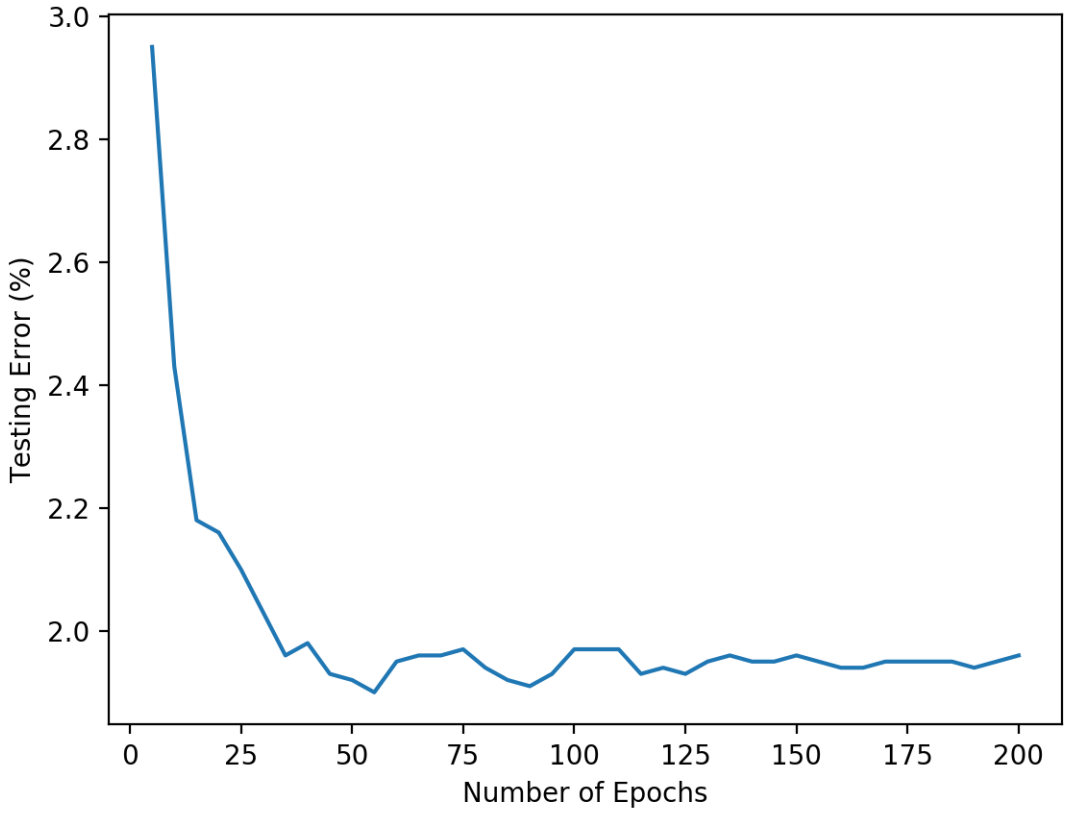
We can see from either the table or the figure that we'd better choose the Number of Hidden Neurons:

$$N_{\text{hidden}} = 256$$

2.3 Epochs

Fixing $N_{\text{hidden}} = 256$ and $\eta = 0.1$. Then after experiments, we have the graph:

Figure 3: Experiment Figure on Number of Epochs



From the experiments, when $\eta = 0.1$, the best number of epochs is 55 where the testing error can achieve as small as 1.9%.

3 Modifications

I modified the algorithm by adding one hidden layer, and now I have two hidden layers. After tuning the parameters, I choose the following parameters:

$$\begin{cases} N_{\text{hidden}_1} = 256 \\ N_{\text{hidden}_2} = 128 \\ N_{\text{input}} = 784 \\ N_{\text{output}} = 10 \\ \eta = 0.1 \\ epoch = 60 \end{cases}$$

And the dimensions of the three weight matrices are:

$$\begin{aligned} W_1 &\in \mathbb{R}^{(N_{\text{input}}+1)\times(N_{\text{hidden}_1}+1)} \\ W_h &\in \mathbb{R}^{(N_{\text{hidden}_1}+1)\times(N_{\text{hidden}_2}+1)} \\ W_2 &\in \mathbb{R}^{(N_{\text{hidden}_2}+1)\times(N_{\text{output}})} \end{aligned}$$

And the pseudocode is modified from the base implementation:

Algorithm 2 Training Artificial Neural Network with 2 Hidden Layers

Let $S = \{(x, y)\}$ be the training set with cardinality N_{train}.
Initialize weight matrices W_1, W_h and W_2.
Let ϕ be the Sigmoid function and can be applied to tensors element-wisely.
for $ep = 1$ to $epoch$ **do**
    for $i = 1$ to N_{train} **do**
        $\mathbf{x}_0 = S[i][x].append([1])$ ▷ Input Layer, and we append a bias neuron
        $\mathbf{x}_{h_1} = \phi(\mathbf{x}_0 W_1)$ ▷ Hidden Layer 1
        $\mathbf{x}_{h_2} = \phi(\mathbf{x}_{h_1} W_h)$ ▷ Hidden Layer 2
        $\mathbf{x}_2 = \phi(\mathbf{x}_{h_2} W_2)$ ▷ Output Layer
        $\mathbf{y} = \mathbf{e}_{S[i][y]}$ ▷ Label of the image
        $\varepsilon = \mathbf{x}_2 - \mathbf{y}$ ▷ Error
        $\delta_2 = 2\left[\phi'(\mathbf{x}_2) * \varepsilon\right]$ ▷ Operation $*$ is the element-wise multiplication
        $\delta_h = \phi'(\mathbf{x}_{h_2}) * (W_2 \delta_2^T)^T$
        $\delta_1 = \phi'(\mathbf{x}_{h_1}) * (W_h \delta_h^T)^T$
        $W_2 \leftarrow W_2 - \eta \mathbf{x}_{h_2}^T \delta_2$ ▷ Updating Weight
        $W_h \leftarrow W_h - \eta \mathbf{x}_{h_1}^T \delta_h$ ▷ Updating Weight
        $W_1 \leftarrow W_1 - \eta \mathbf{x}_0^T \delta_1$ ▷ Updating Weight
    end for
end for

By using 2 hidden layers, I achieved the testing error 1.85%.

4 Submission

There are two python files: the one named *homework6.py* is the base implementation with only one hidden layer, and with the normalization of weights initially and the other one named *homework6_2hidden.py* is the one with two hidden layers. By default, both python files run the training first and testing second but the TA can use other functions in the file such as **experiment**() which I used for finding the optimal parameters for training, and functions such as **predictDigitX**() and **predictDigitX2**(), which are used for predicting the two testing files.
The prediction results of **TestDigitX** and **TestDigitX2** are **PredDigitY** and **PredDigitY2** respectively.

5 Some Comments

In the description of this homework, we are supposed to find the number of epochs "when error on the holdout set starts increasing substantially". However, such phenomenon is hard to detect, among my days of experiments, I only detected two such instances when I set the number of epochs to be extremely large (such as 400). Even though such phenomenon was detected, it was not so obvious at all. Mostly, the holdout error would drop to a certain values and seems to converge for a long time, and fortunately, it would begin increasing for some time (substantially) but most of time it would start decreasing and converge again.

Another thing we are supposed to do is when we conduct experiments of parameters, we keep track of the testing error corresponding to the number of epochs. But as we can see from Figure 3., the testing error tends to converge too (and in my experiments, it mostly converge to 1.95% when using the parameters I chose. Such phenomenon does not conform with my expectation. What I expected was the testing error, like what we should expect about the holdout error, should increase after sometime due to overfitting. But strangely, the model seems not to overfit.

One last observation of doing this homework is that the optimal testing error is very hard to achieve. My best result is 1.9% and such result was achieved after I run four times.

If time permitting, I can do more than the small modification of only adding one hidden layer. I can try to change the activation function from Sigmoid to ReLU or tanh, or use softmax for the final layer. I can play with the loss function such that using the cross entropy loss function. Also, I can test about the number of hidden neurons: what will happen if I increase the number to 512, 1024, or even larger numbers. Last thing I can do further is try to use convolution neural network to train the data to see what would happen.