Artificial Intelligence Project 3
Neural Networks

Class 1, Team 6

January 10, 2015

Contents

1 Team Members

Name	Student ID	Job
Fan Ziyao	12330081	Team leader, implementation
Chen Yingcong	12330049	Implementation
Chen Xiongtao	12330040	Modeling, implementation
Huang Long	12330132	Implementation
Zhang Qiuyi	12330402	Implementation, documentation

2 Problem Description

The dataset (including the training set and the test set) contains clean data generated from 8×8 images of handwritten digits. Each record contains the sampled data of size $8 \times 8 = 64$ (each point ranging in $[0, 16]$) and its label (range $[0, 9]$).

Given this dataset, the goal is to build a neural network with the training set and evaluate its precision by classifying the records in the test set (that is, to predict its label based on its sampled data).

3 Algorithm and Implementation

For this project we implemented a multilayer perceptron(MLP) with the logistic function

$$y(v_i) = (1 + e^{-v_i})^{-1}$$

as the activation function.

3.1 Building the Neural Network

First we initialize the neural network with random weights ranging between $[-0.25, 0.25]$ (it is just an empirical value). Then, we use back propagation for training the network. Here instead of running the backpropagation on the training data sequantially, we randomly choose records from the training data to propagate. The algorithm is discribed in Algorithm 1.

The back propagation, as shown in Algorithm 2, will be run until a certain number of epochs is reached.

Algorithm 1 Training the Network

```
function TRAINING($examples$, $epochs$)
    Initiailize $network$ with random weights between $[-0.25, 0.25]$
    for $k = 1 \to epochs$ do
        Randomly choose a record $(\mathbf{x}, \mathbf{y})$ from $examples$
        BACK-PROPAGATE($\mathbf{x}$, $\mathbf{y}$, $network$)
    end for
end function
```

Algorithm 2 Back Propagation

```
function BACK-PROPAGATE($\mathbf{x}$, $\mathbf{y}$, $network$)            ▷ $\mathbf{x}$ is the input
    $w \leftarrow network.weights$            ▷ $\mathbf{y}$ is the vector transformed from the label
    for each node $i$ in the input layer do
        $a_i \leftarrow x_i$
    end for
    for $l = 2 \to L$ do            ▷ $L$ is the number of layers in the network
        for each node $j$ in layer $l$ do
            $in_j \leftarrow \sum_i w_{i,j} a_i$
            $a_j \leftarrow g(in_j)$            ▷ $g$ is the activation fucntion
        end for
    end for
    for each node $j$ in the otput layer do
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$            ▷ $\Delta$ is a local vector of erros indexed by network node
    end for
    for $l = L - 1 \to 1$ do
        for each node $i$ in layer $l$ do
            $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$            ▷ propagate backwords
        end for
    end for
    for each weight $w_{i,j}$ do
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$            ▷ gradient descent rule, $\alpha$ is the learning rate
    end for
    $network.weights \leftarrow w$
end function
```

3.2 Classification

The algorithm for classifying an observation is described in Algorithm 3. Note that the output is a vector, its value needs to be further interpreted.

Algorithm 3 Classification

```
function CLASSIFY($network$, $observation$)
    $w \leftarrow network.weights$
    $a \leftarrow observation$
    for $l = 2 \to L$ do            ▷ $L$ is the number of layers in the network
        for each node $j$ in layer $l$ do
            $in_j \leftarrow \sum_i w_{i,j} a_i$
            $a_j \leftarrow g(in_j)$            ▷ $g$ is the activation fucntion
        end for
    end for
    return $a$
end function
```

4 Experiment Result

For this project, we set the number of layers in the neural network to 3 (i.e. one input layer, one hidden layer, one output layer), and the learning rate α is set to 0.2. Before building the neural network, we first transform the labels into vectors. Every elements in the vector is first set to 0, then the element with the same index as the label will be set to 1. For example, when the label is 0, we transform it into a vector:

$$[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

Then, the input layer consists of vectors of size 64, and the output layer consists of vectors of size 10. The output vectors contains the probabilities for each label, here we choose the label with the largest probability as the final result.

We evaluate the implementation from two different perspectives. First, we examine the relationship between its precision and the size of the training data (i.e. the learning curve), as shown in Figure 1. The number of hidden units is set to 50, and we stop the iteration after $50,000$ epochs. When the size of training data is larger than 1000, the precision stays higher than 90%. Generally the precision increases with the size of training data, but there are signs of overfitting.

Then, we examine the relationship between the total errors produced by the neural network and the number of epochs. Here we use the full training data set and set the number of hidden units to 50. As shown in Figure 2, The number of errors decreases significantly when the epochs grows large enough. When the number of epochs becomes larger than $20,000$, the number of errors stays at about 100 (given that the size of the test set is about $1,700$, this implies that the precision stays over 90%).

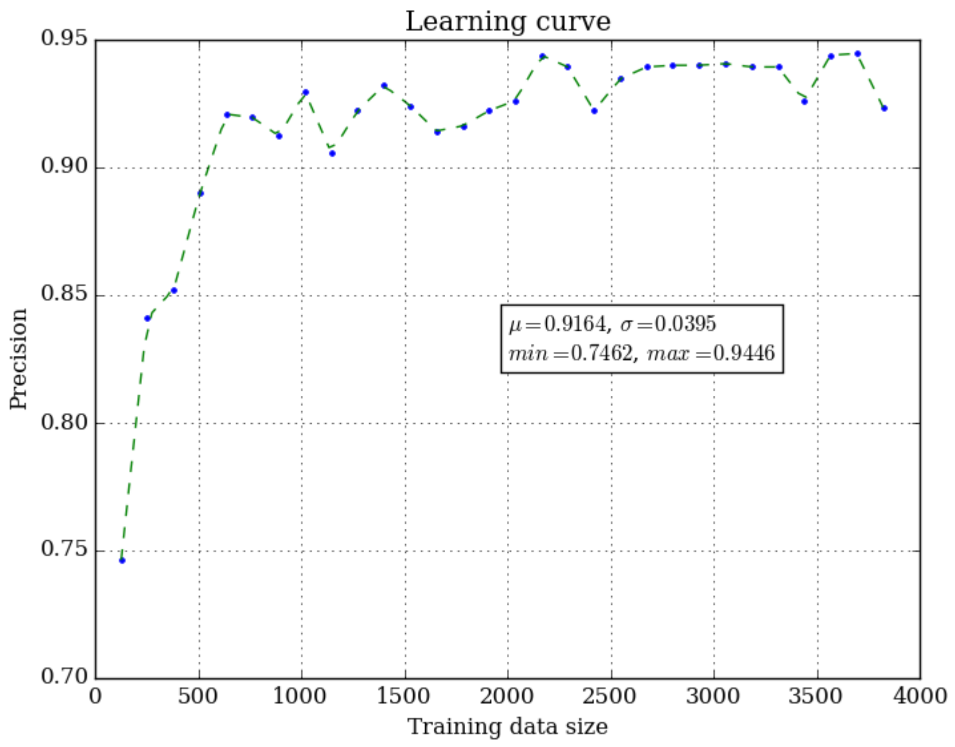

Figure 1: Learning Curve

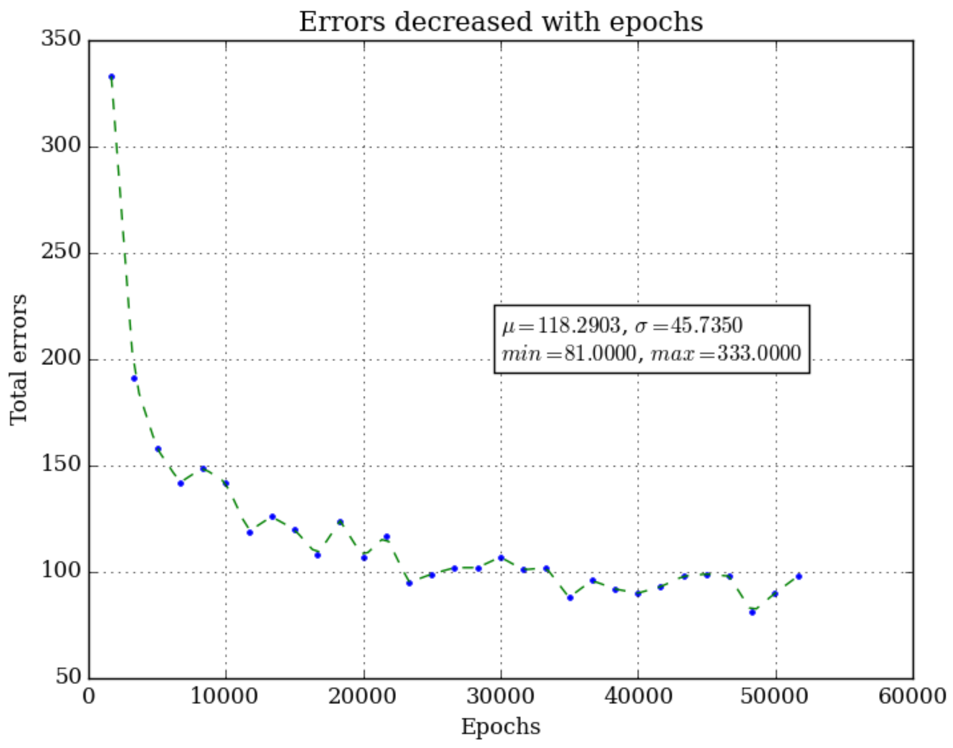

Figure 2: Relationship between total errors and epochs