Homework 4

- Kai Huang
- k.huang@pitt.edu

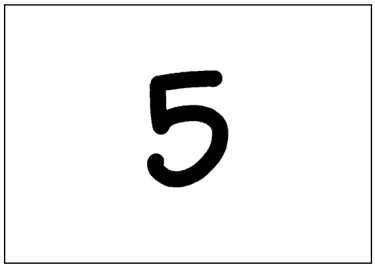
This project uses a artificial neural network with one hidden layer to classify English characters (62 classes). It includes data preprocessing part and classifier implementation. All of the work is done with MATLAB code without any online code or non-built-in packages. The average testing accuarcy is 0.68.

Data Preprocessing

1. Resize and Cropping

The raw data samples are 900 by 1200 images of PNG format. By observing, we can see all the images are black and white, so we can convert them to binary images, i.e., using 1 and 0 to represent white and black pixels respectively. Other than that, each image contains much redundant margin, so we can truncate the region of interest before classification. The truncation algorithm is simply doing exhaustive search for the character pixel boundary. I defined the algorithm in the file `FindBound.m`.

Specifically, in this project, I convert the raw PNG image to a 32 by 32 binary image. One example is shown as follows.

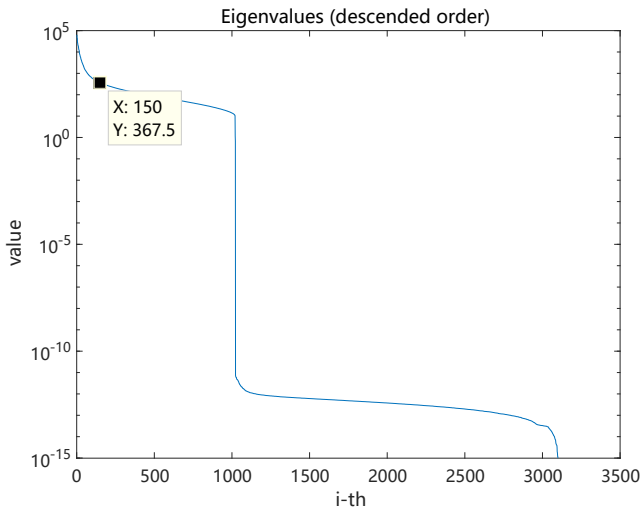


Raw image 900x1200

After preprocessing 32x32

2. Dimensionality Reduction

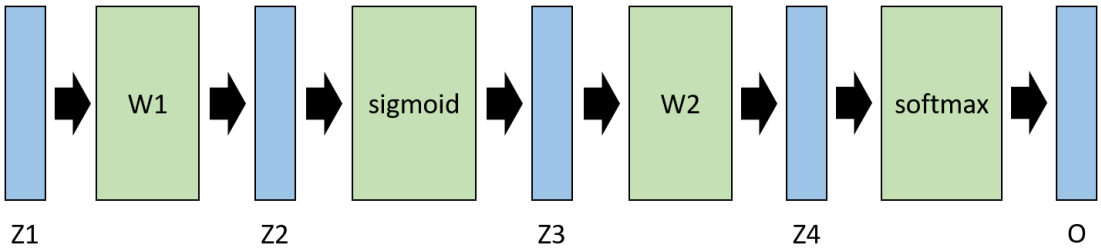
When fed to the neural network, the image should be flattened. 32 by 32 leads to a 1024 by 1 vector, which is relatively long compared with hidden layer dimension. So I simply use PCA for dimensionality reduction.



By observing the eigenvalues above, I choose to reserve the first 150 eigenvalues.

Artificial Neural Network

The ANN contains one input layer, one hidden layer and one output layer. To better explain Back Propagation implementation, I illustrate the network structure as follows.



1. Forward Propagation

I defined forward propagation in file `ThreeLayerNN.m`.

$$\mathbf{z}_2 = W_1^T \mathbf{z}_1$$

$$\mathbf{z}_3 = \text{sigmoid}(\mathbf{z}_2)$$

$$\mathbf{z}_4 = W_2^T \mathbf{z}_3$$

$$\mathbf{o} = \text{softmax}(\mathbf{z}_4)$$

Please noted that $\mathbf{z}_1$ and $\mathbf{z}_3$ should extend one dimension by simply attach constant 1 before doing matrix multiplication.

2. Back Propagation

I defined back propagation in file `BackPropOnce.m`

We use $\mathbf{g}_i$ to denote the gradient of loss with respect to $\mathbf{z}_i$, i.e., $\mathbf{g}_i = \frac{\partial L}{\partial \mathbf{z}_i}$. Then we have following backward propagation. ($\odot$ indicates element-wise multiplication)

$$\mathbf{g}_4 = \mathbf{o} - \mathbf{y}$$

$$\mathbf{g}_3 = W_2 \mathbf{g}_4$$

$$\mathbf{g}_2 = \mathbf{g}_3 \odot \mathbf{z}_3 \odot (1 - \mathbf{z}_3)$$

$$\mathbf{g}_1 = W_1 \mathbf{g}_2$$

To calculate gradient of loss with respect to weight W_1 and W_2, we have following formulas.

$$\frac{\partial L}{\partial W_1} = \mathbf{z}_1 \mathbf{g}_2 \qquad \frac{\partial L}{\partial W_2} = \mathbf{z}_3 \mathbf{g}_4$$

With $\frac{\partial L}{\partial W_1}$ and $\frac{\partial L}{\partial W_2}$ available, we can use gradient descent to update the weights W_1 and W_2. In this project, I use mini-batch method so we take the average of gradients for updating. The update formula is as follows, in which α is learning rate.

$$W \leftarrow W - \alpha \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L_i}{\partial W}$$

Since the dataset is quite small, I add L2 norm to mitigate overfitting problem. Adding L2 norm regularizer in the loss function is equivalent to imposing decay on W in gradient descent update. The formula is shown as follows.

$$W \leftarrow (1 - C)W - \alpha \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L_i}{\partial W}$$

ANN Training and Testing

According to the homework instruction, I use 9/10 of data for training and 1/10 for testing.

The hyperparameters are set as follows.

- $\alpha = 1$
- $C = 1 \times 10^{-3}$
- $\mathrm{batchSize} = 310$
- $\mathrm{numEpoch} = 130$

I run the training and testing for 5 times independently. The accuracies are reported below.

	1	2	3	4	5	Average
Training	0.741	0.742	0.751	0.749	0.751	**0.749**
Testing	0.687	0.652	0.703	0.677	0.687	**0.681**