Computer Vision HM5

Andrew ID: beileiz@andrew.cmu.edu

November 23, 2019

1 Theory

1.1 Prove softmax is invariant to translation

$$\begin{aligned} softmax(x_i + c) &= \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} \\ &= \frac{e^c e^{x_i}}{e^c \sum_j e^{x_j+c}} \\ &= \frac{e^{x_i}}{\sum_j e^{x_j+c}} \end{aligned} \tag{1}$$

Often we use $c = -max(x_i)$. This is because it bring the range of the numerator to $[e^{-\infty}, e^0]$, which is [0, 1], thus takes an arbitrary real valued vector x ad turns it into a probability meaning

1.2 The meaning of softmax

- The range of each element is [0, 1], The sum of each element is 1
- One could say that softmax takes an arbitrary real valued vector x ad turns it into a probability
- S_i represent for the amount one event occurs, S represent for the amount all events occurs, $\frac{S_i}{S}$ represent for the probability of one event

1.3 The function of nonlinear activation

Create a neural network without activation function, we have the initial layer,

$$z_i = W_i x + b_i \tag{2}$$

$$a_i = z_i \tag{3}$$

$$z_{i+1} = W_{i+1} \times x + b_{i+1} \tag{4}$$

$$a_{i+1} = z_{i+1} \tag{5}$$

from equation (2) (3) (4) (5), we can get

$$z_{i+1} = (W_{i+1} \times W_i)X + (W_{i+1} \times b_i + b_{i+1}) \tag{6}$$

That is to say, a multi-layer neural network without a non-linear activation function is equivalent to linear regression

1.4 Rewrite the derivative of sigmoid activation function

$$\sigma'(x) = \sigma(x)(1-\sigma(x)) \tag{7}$$

1.5 Jacobians for Backward Propagation

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{d} X_j W_{1j} + b_1 \\ \sum_{j=1}^{d} X_j W_{2j} + b_2 \\ \vdots \\ \sum_{j=1}^{d} X_j W_{kj} + b_k \end{bmatrix} \tag{8}$$

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial activation(y_i)} \frac{\partial activation(y_i)}{\partial y_i} \frac{\partial y_i}{\partial w_{ij}} \tag{9}$$

$$\frac{\partial J}{\partial x_j} = \frac{\partial J}{\partial activation(y_i)} \frac{\partial activation(y_i)}{\partial y_i} \frac{\partial y_i}{\partial x_j} \tag{10}$$

$$\frac{\partial J}{\partial b_i} = \frac{\partial J}{\partial activation(y_i)} \frac{\partial activation(y_i)}{\partial y_i} \frac{\partial y_i}{\partial b_i} \tag{11}$$

$$\frac{\partial J}{\partial W_{ij}} = \begin{bmatrix} \frac{\partial J}{\partial y_1}\frac{\partial y_1}{\partial W_{11}} & \frac{\partial J}{\partial y_2}\frac{\partial y_2}{\partial W_{12}} & \cdots & \frac{\partial J}{\partial y_d}\frac{\partial y_d}{\partial W_{1d}} \\ \frac{\partial J}{\partial y_1}\frac{\partial y_1}{\partial W_{21}} & \frac{\partial J}{\partial y_2}\frac{\partial y_2}{\partial W_{22}} & \cdots & \frac{\partial J}{\partial y_d}\frac{\partial y_d}{\partial W_{2d}} \\ \vdots & & & \\ \frac{\partial J}{\partial y_1}\frac{\partial y_1}{\partial W_{k1}} & \frac{\partial J}{\partial y_2}\frac{\partial y_2}{\partial W_{k2}} & \cdots & \frac{\partial J}{\partial y_d}\frac{\partial y_d}{\partial W_{kd}} \end{bmatrix} = \begin{bmatrix} \frac{\partial J}{\partial y_1}x_1 & \frac{\partial J}{\partial y_1}x_1 & \cdots & \frac{\partial J}{\partial y_1}x_1 \\ \frac{\partial J}{\partial y_2}x_2 & \frac{\partial J}{\partial y_2}x_2 & \cdots & \frac{\partial J}{\partial y_2}x_2 \\ \vdots & & & \\ \frac{\partial J}{\partial y_d}x_d & \frac{\partial J}{\partial y_d}x_d & \cdots & \frac{\partial J}{\partial y_d}x_d \end{bmatrix} \tag{12}$$

$$\frac{\partial J}{\partial W} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial y_1} & \frac{\partial J}{\partial y_2} & \cdots & \frac{\partial J}{\partial y_k} \end{bmatrix} = X(\frac{\partial J}{\partial y})' \tag{13}$$

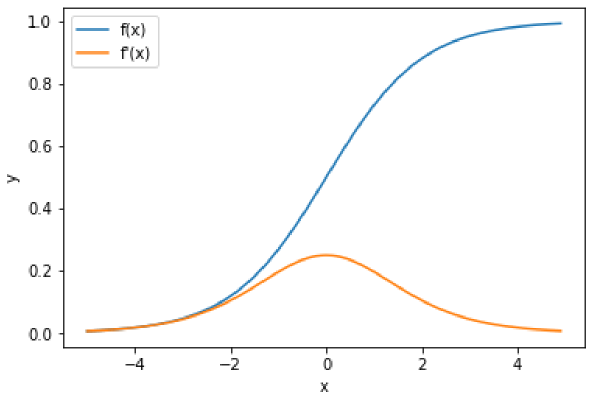
$$\frac{\partial J}{\partial X} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_k}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_k}{\partial x_2} \\ \vdots & & & \\ \frac{\partial y_1}{\partial x_d} & \frac{\partial y_2}{\partial x_d} & \cdots & \frac{\partial y_k}{\partial x_d} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial y_1} \\ \frac{\partial J}{\partial y_2} \\ \vdots \\ \frac{\partial J}{\partial y_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial J}{\partial y_1}w_{11} + \frac{\partial J}{\partial y_2}w_{21} & \cdots + \frac{\partial J}{\partial y_k}w_{k1} \\ \frac{\partial J}{\partial y_1}w_{12} + \frac{\partial J}{\partial y_2}w_{22} & \cdots + \frac{\partial J}{\partial y_k}w_{k2} \\ \vdots & \\ \frac{\partial J}{\partial y_1}w_{1d} + \frac{\partial J}{\partial y_2}w_{2d} & \cdots + \frac{\partial J}{\partial y_k}w_{kd} \end{bmatrix} \tag{14}$$

$$\frac{\partial J}{\partial X} = \begin{bmatrix} \frac{\partial J}{\partial y_1} & \frac{\partial J}{\partial y_2} & \cdots & \frac{\partial J}{\partial y_d} \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} & w_{31} & \cdots & w_{k1} \\ w_{12} & w_{22} & w_{32} & \cdots & w_{k2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{1d} & w_{2d} & w_{3d} & \cdots & w_{kd} \end{bmatrix} = (\frac{\partial J}{\partial y})'W' \tag{15}$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial y} \tag{16}$$

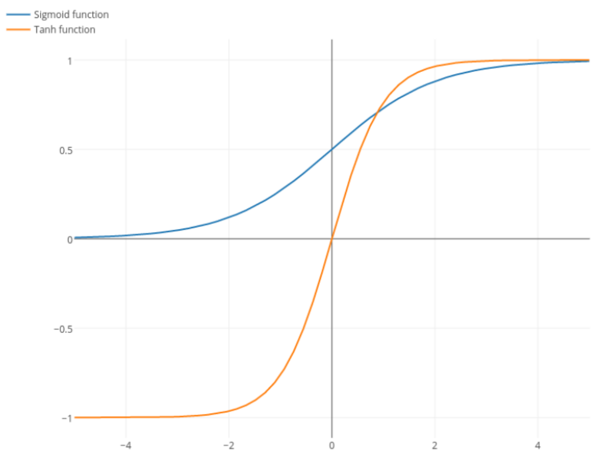
1.6 Compare tanh and sigmoid activation function

1.6.1 Vanishing gradient problem of sigmoid function



From the above figure we can see it's easy for f'(x) to go near 0 The vanishing gradient problem comes about in deep neural networks when the f terms are all outputting values go near 0. When we multiply lots of numbers near 0 together, we end up with a vanishing product, which leads to a very small value and hence practically no learning of the weight values the predictive power of the neural network then platueus.

1.6.2 tanh vs sigmoid



From above figure we can see the output range of sigmoid function is [0, 1], and the output range of tanh function is [-1,1]. When prefer later because the convergence is quicker because the derivative is bigger

1.6.3 Why tanh has less vanishing gradient problem

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \leq 0.25$$
$$tanh'(x) = 1 - tanh^2(x) \leq 1$$

1.6.4 tanh is a scaled and shifted version of the sigmoid

$$tanh(x) = 2\sigma(2x) - 1$$

2 Implement a Fully Connected Network

2.1 Network Initialization

2.1.1 zero initialization

Because it turn every hidden layer to 0 no matter what the input is, the output layer will be 0

2.1.2 Weight Initialization Code

2.1.3 Weight Scaling

We choose different/random value of initialized weight because we want our neural network to perform symmetry-breaking.
By scacling the weight according to the layer size, the weighted sum of inputs at a hidden layer will not take a large value and reduces the chances of the exploding gradient problem.

3 Training Models

3.1 Training model code

3.2 Comparing different learning rates

From Figure 1 2 3 we can see when $learningrate = 3e-3$ the model get its best performance. When learning rate is ten times smaller ($learningrate = 3e-4$), the model converge very slowly. With limited epoch, it reach a lower accuracy. When the learning rate is ten times smaller ($learningrate = 3e-1$), the model oscillate around the global minimum and fail to converge.

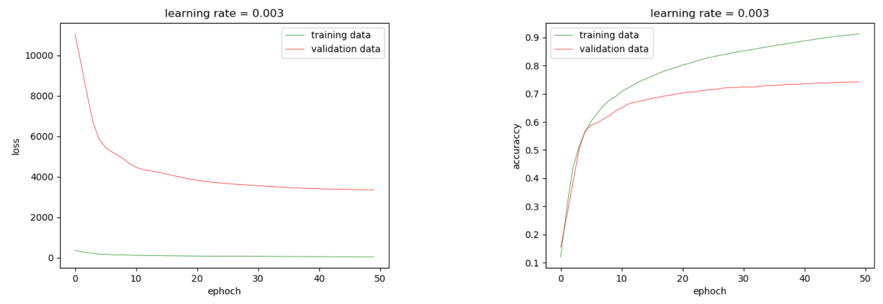


Figure 1: loss and accuracy when learning rate = 3e-3

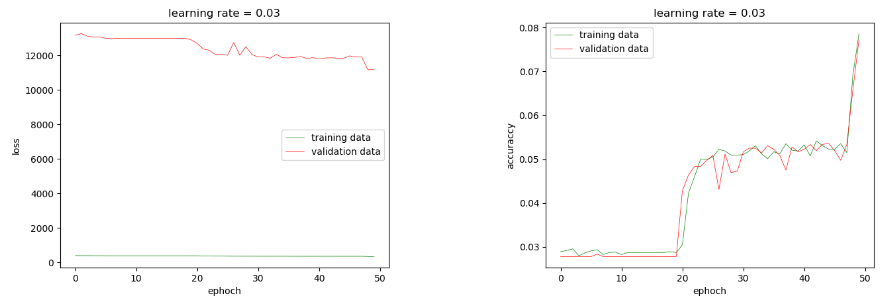


Figure 2: loss and accuracy when learning rate = 3e-2

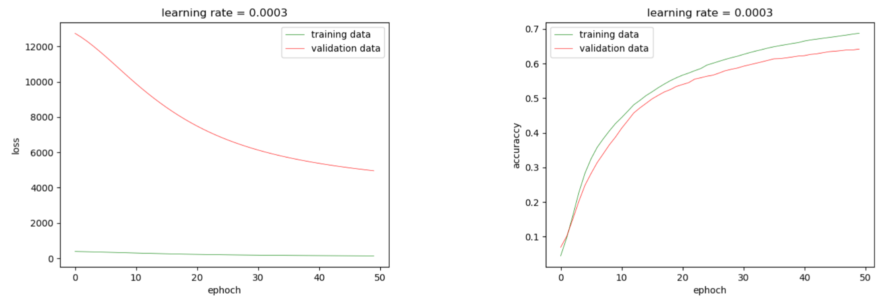


Figure 3: loss and accuracy when learning rate = 3e-4

3.3 Weight Visualization

From Figure 4 and 5, we can see that before training, the weight of every image pixel in every node is just random noise. After training, every node put different ”focus” on different pixel, forming some continuous shape in the visualization.

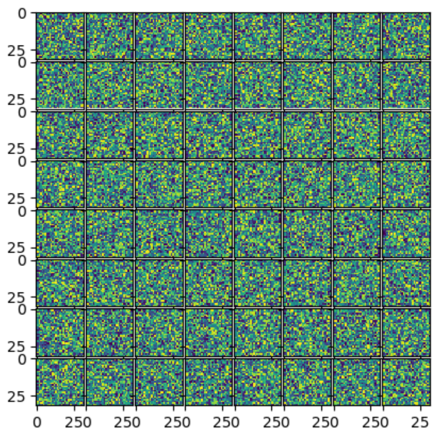


Figure 4: original weights

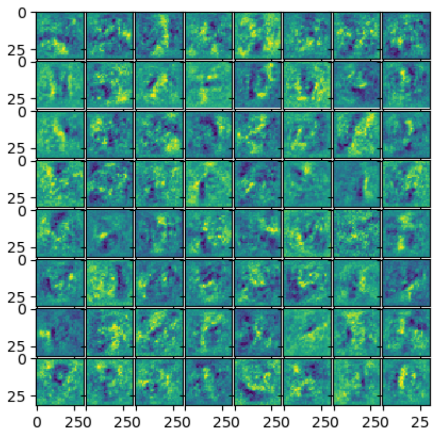


Figure 5: trained weights

3.4 Confusion Matrix

From Figure 6, we can see the most confused pair is character ”O” and number ”0”, which is also confused to us. Other confused pairs are ”s” and ”5”, ”z” and ”2” which have familiar shapes.

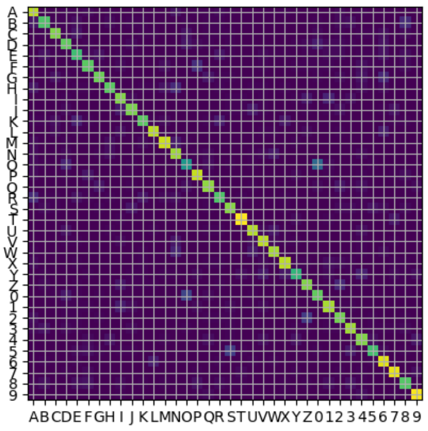


Figure 6: prediction confusion matrix

4 Extract Text from Images

4.1 Draw bounding box

The method outlined is based on two assumptions:

1. The stroke of a single character is continuous
2. The gray value of all texts exceed certain threshold and the gray value of background is less than it

I suppose the texts in Figure 7 may be misjudged since their strokes are not continuous. But in practice, this problem may be easily solved by dilation

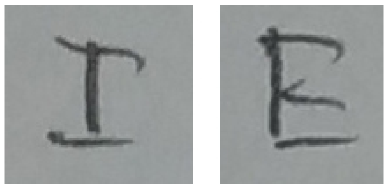
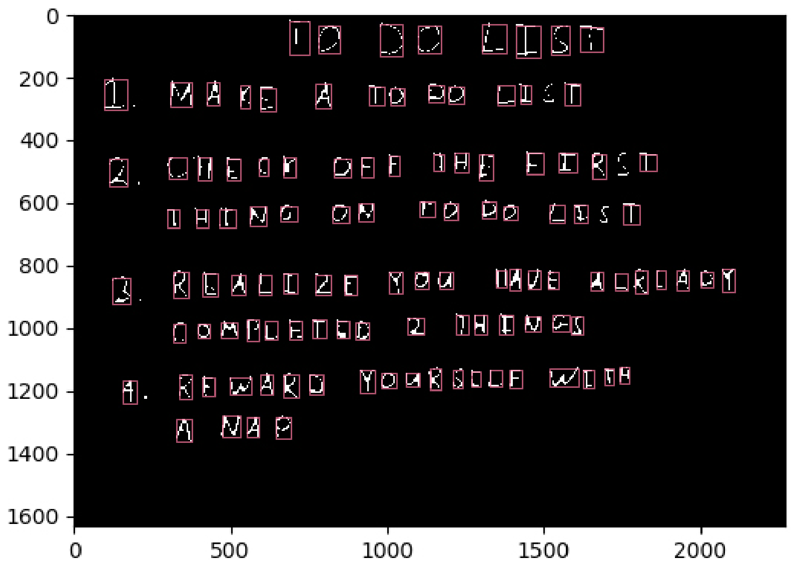
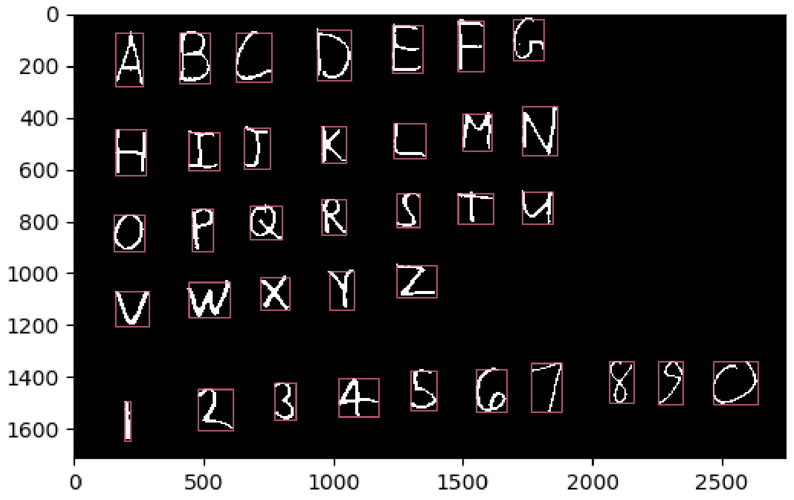


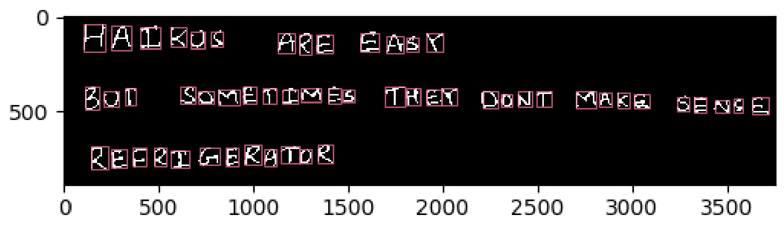
Figure 7: possibly misjudged text

4.2 Bounding box code

4.3 Bounding box result







4.4 Text recognition result

In the recognition, I padding the original bounding box with 20px both in height and width, then resize them to (32*32) image. I found by padding the image, making it looks more like the training set in Q3, the correctness increased. If I have time, I should try more different padding value.

4.4.1 To do list

```
T0 D0 LIST
I MAKE A TO DO LIST
Z CHFEK DFF 7HE FIR4T THING QN TO D0 CI4T
3 R2ALIZE Y0U HVE ALRQADY C0MPLETLD 2THINFS
5 REWARO Y0UR 6ELF WETA A NAP
```

4.4.2 ABCDEFG

```
ABCDEFG
HIJKLMN
QFQRSTU
VWXYZ
8Z34SG7X9D
```

4.4.3 Haikus

```
HAIKUS ARE EA5X
BUT SOMETIMES TAEY DOWT MAK6 SENQE
REFRIGERAT0R
```

4.4.4 Deep learning

```
DYHF CCARMIXG
D4FTFK LHAKNIRG
5FCFFST LEARRIMG
```

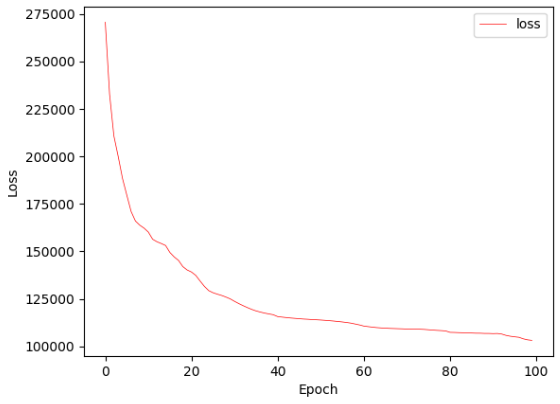
5 Image Compression with Autoencoder

5.1 Build the autoencoder code

5.2 Train the auto encoder code

5.2.1 Loss over epoch

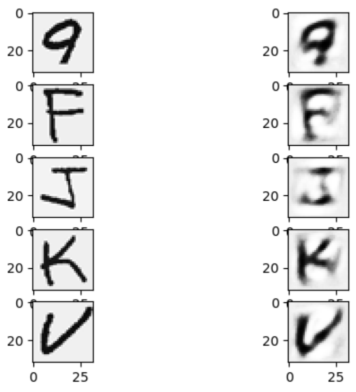
The loss at first drop very fast, then slower, then keep horizontal.



5.3 Evaluating the autoencoder

5.3.1 Autoencoder visualization

Compared to the original image, it's more vague, with smooth edge. It contains less but accurate information.



5.3.2 Peak Signal-to-noise Ratio

$$AveragePSNR = 16.121307760985935 \tag{17}$$

6 Comparing with PCA

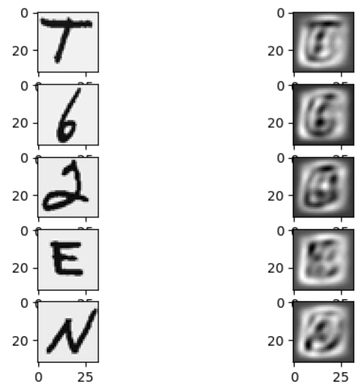
6.1 Projection Matrix

The shape of the projection matrix
= Original pixel number * compressed pixel number
= 1024*32

The rank of the projection matrix = compressed pixel number = 32

6.2 Reconstruction

The reconstructed image is more vague with unclear edges, containing less information. Compare to the auto-encoder images, it's harder to tell.



6.3 Peak Signal-to-noise Ratio

$$PSNR = 16.34680324377696 \tag{18}$$

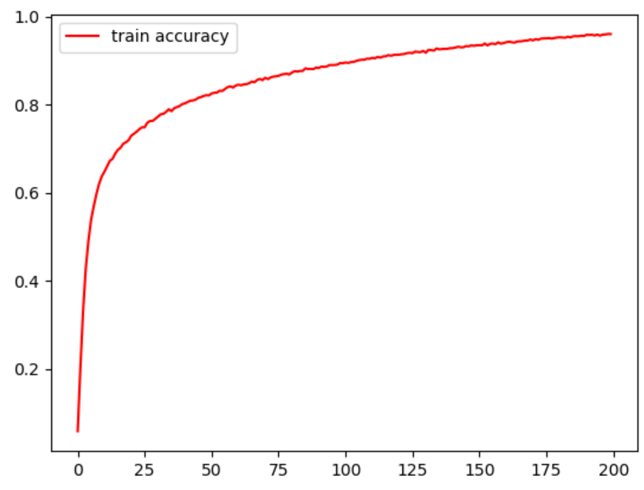
6.4 Learned parameters of PCA and autoencoder

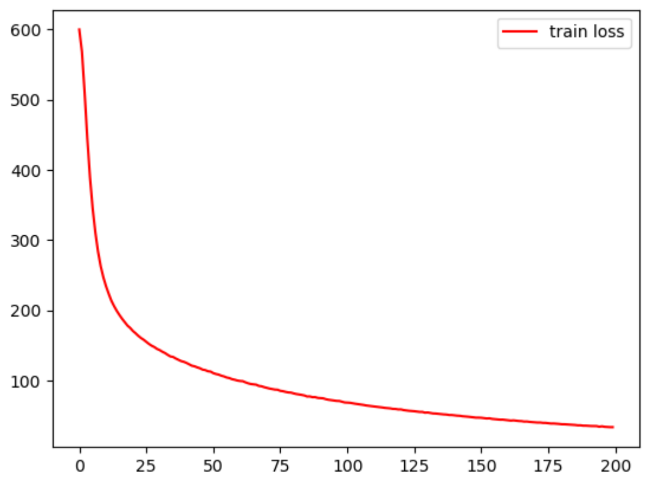
The number of learned parameters in PCA is 1024*1024, while the number of learned parameters of auto-encoder is (1024*32)*(32*32)*(32*32). The performance of auto-encoder is better because it has more parameters and non-linear learning process

7 Pytorch

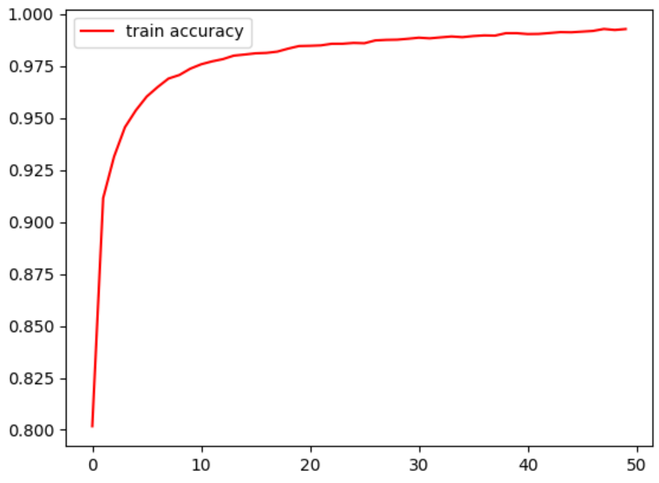
7.1 Train a neural network in PyTorch

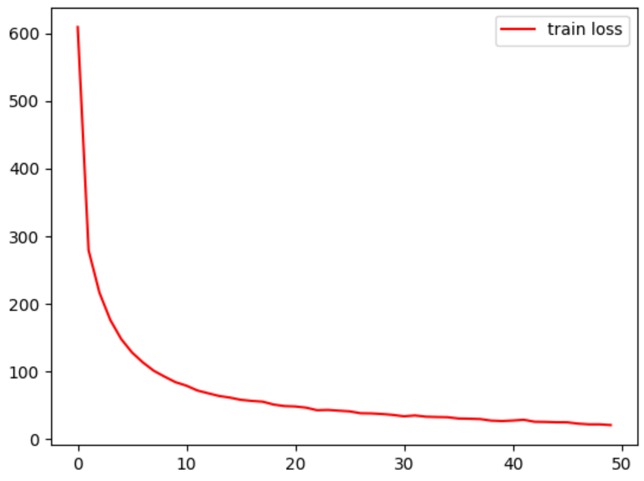
7.1.1 Two layer fully connected neural network on NIST36



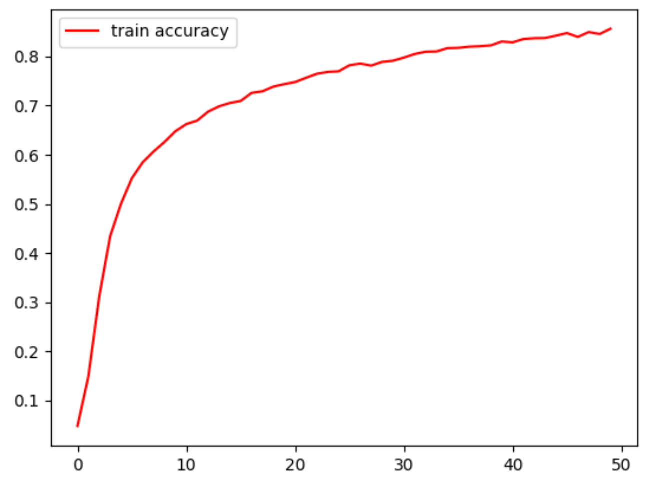


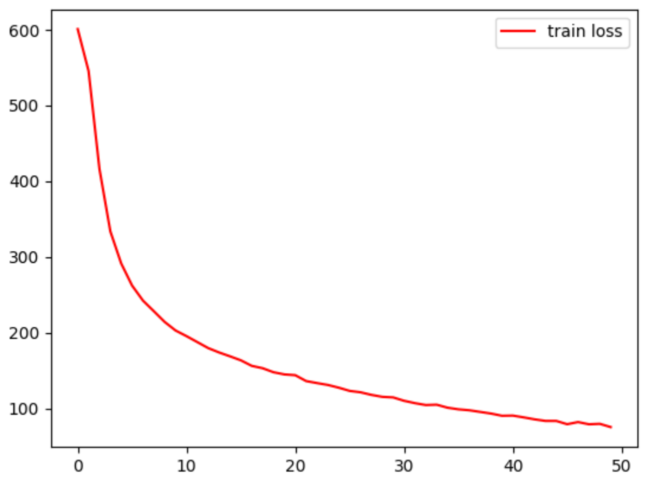
7.1.2 Lenet-5 on MNIST





7.1.3 Lenet-5 on NIST36


train accuracy


train loss