16-720 Computer Vision: Homework 5

Yanjia Duan

November 2020

Q1.1

For each index i in a vector x,

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

$$softmax(x_i + c) = \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} = \frac{e^{x_i}e^c}{\sum_j e^{x_j}e^c} = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

$$\Rightarrow softmax(x_i) = softmax(x_i + c)$$

Therefore, $softmax(x) = softmax(x + c)$.

If we use $c = -\max x_i$, the softmax result keeps the same and e^{x_i+c} will be in range (0, 1] which could avoid numerical overflow. If $c = 0$, e^{x_i+c} will be in range (0, $+\infty$) and could cause numerical overflow.

Q1.2

- The range of each softmax element is (0, 1], and the sum over all elements is 1.
- Softmax takes an arbitrary real valued vector x and turns it into a probability distribution.
- Step 1: Map all elements to positive values.
  Step 2: Calculate the normalization factor for all elements.
  Step 3: Normalize each element to (0, 1] to get a probability distribution.

Q1.3

Suppose that a multi-layer neural network has two hidden layers, where the first hidden layer is M_1-dimensional and the second hidden layer is M_2-dimensional. The input $x \in \mathbb{R}^N$. Suppose that the weights and biases for the i^{th} hidden layer are w_i and b_i. The linear activation functions for the i^{th} hidden layer are $f_i(x) = c_i x + d_i$. Therefore, $w_1 \in \mathbb{R}^{M_1 \times N}$, $b_1 \in \mathbb{R}^{M_1}$, $w_2 \in \mathbb{R}^{M_2 \times M_1}$, $b_2 \in \mathbb{R}^{M_2}$.

The output from hidden layer 1 is

$$\begin{aligned} y_1 = f_1(w_1 x + b_1) &= c_1(w_1 x + b_1) + d_1 \\ &= c_1 w_1 x + c_1 b_1 + d_1 \end{aligned}$$

which is a linear function of x. The output from hidden layer 2 is

$$\begin{aligned} y_2 = f_2(w_2 y_1 + b_2) &= f_2(w_2(c_1 w_1 x + c_1 b_1 + d_1) + b_2) \\ &= c_2(w_2(c_1 w_1 x + c_1 b_1 + d_1) + b_2) + d_2 \\ &= c_2 w_2 c_1 w_1 x + (c_2 w_2 c_1 b_1 + c_2 w_2 d_1 + c_2 b_2 + d_2) \end{aligned}$$

which is a linear function of x. The same result holds for neural networks of more than two layers. Therefore, multi-layer neural networks without a non-linear activation function are equivalent to linear regression.

Q1.4

The gradient of the sigmoid function can be written as a function of $\sigma(x)$:

$$\begin{aligned}
\sigma(x) &= \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \\
\frac{d\sigma(x)}{dx} &= \frac{e^x(e^x+1) - e^x \cdot e^x}{(e^x+1)^2} \\
&= \frac{e^x}{e^x+1} - (\frac{e^x}{e^x+1})^2 \\
&= \frac{e^x}{e^x+1}(1 - \frac{e^x}{e^x+1}) \\
&= \sigma(x)(1-\sigma(x))
\end{aligned}$$

Q1.5

$$\because \qquad \frac{\partial J}{\partial W} \in \mathbb{R}^{k\times d} \qquad \frac{\partial J}{\partial x} \in \mathbb{R}^{d\times 1} \qquad \frac{\partial J}{\partial b} \in \mathbb{R}^{k\times 1}$$

$$\frac{\partial y}{\partial W} = \begin{bmatrix} \frac{\partial y_1}{\partial W_{11}} & \cdots & \frac{\partial y_k}{\partial W_{k1}} \\ \frac{\partial y_1}{\partial W_{12}} & \cdots & \frac{\partial y_k}{\partial W_{k2}} \\ \vdots & & \vdots \\ \frac{\partial y_1}{\partial W_{1d}} & \cdots & \frac{\partial y_k}{\partial W_{kd}} \end{bmatrix}_{d\times k} = \begin{bmatrix} x_1 & \cdots & x_1 \\ x_2 & \cdots & x_2 \\ \vdots & & \vdots \\ x_d & \cdots & x_d \end{bmatrix}_{d\times k}$$

$$\frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_k}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial y_1}{\partial x_d} & \cdots & \frac{\partial y_k}{\partial x_d} \end{bmatrix}_{d\times k} = \begin{bmatrix} W_{11} & \cdots & W_{k1} \\ \vdots & & \vdots \\ W_{1d} & \cdots & W_{kd} \end{bmatrix}_{d\times k} = W^T$$

$$\frac{\partial y}{\partial b} = \begin{bmatrix} \frac{\partial y_1}{\partial b_1} & \cdots & \frac{\partial y_k}{\partial b_1} \\ \vdots & & \vdots \\ \frac{\partial y_1}{\partial b_k} & \cdots & \frac{\partial y_k}{\partial b_k} \end{bmatrix}_{k\times k} = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{bmatrix}_{k\times k}$$

$$\therefore \qquad \frac{\partial J}{\partial W} = \frac{\partial J}{\partial y} * \frac{\partial y}{\partial W}^T = \delta x^T$$

$$\frac{\partial J}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial J}{\partial y} = W^T\delta$$

$$\frac{\partial J}{\partial b} = \frac{\partial y}{\partial b}\frac{\partial J}{\partial y} = \delta$$

Q1.6

1. Since the output of $\sigma(x)$ is less than 1 as Figure 1 shows, deeper layers have smaller outputs. During backprop, when passing through a sigmoid activation function, the derivative needs to multiply the derivative of sigmoid $\sigma(x)(1-\sigma(x))$ which is very small. So the gradient of deeper layers are closer to zero. This is the "vanishing gradient" problem.

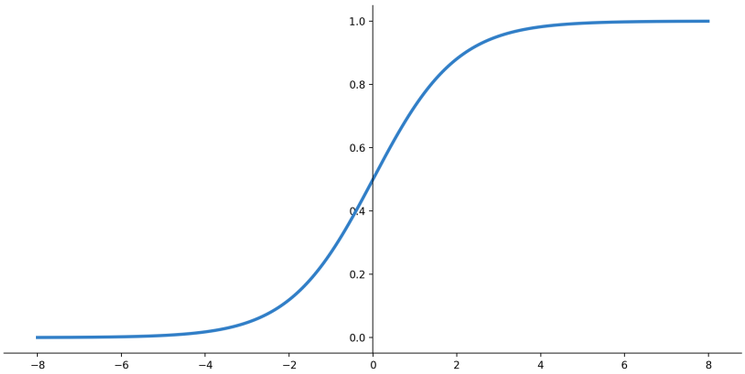


Figure 1: Sigmoid activation function.

2. As shown in Figure 2, the output range of $\tanh(x)$ is $(-1, 1)$, of sigmoid is $(0, 1)$.

Tanh is preferred because tanh learns faster than sigmoid and sigmoid is easy to get stuck. Since tanh is zero-mean and its slope around zero is steeper, if the input value is around zero, tanh will have larger gradient than sigmoid and will learn faster. If the input is strongly negative, sigmoid will be almost zero while tanh is negative, therefore tanh is less likely to get stuck.

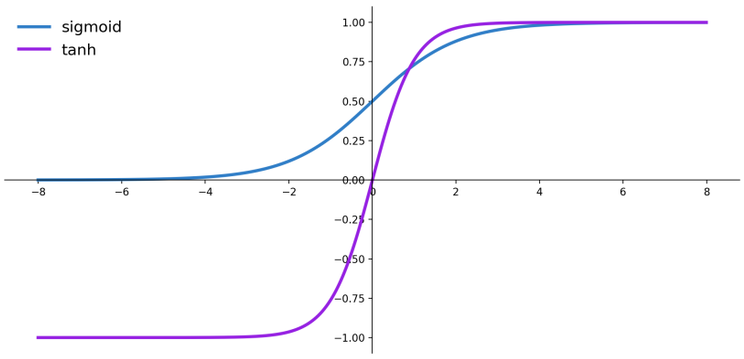


Figure 2: Sigmoid and tanh functions.

3. As shown in Figure 3, the gradient of tanh around zero is larger than sigmoid. During backprop, when passing through a tanh activation function, the whole derivative multiplies a larger tanh derivative. Gradients for deeper neural networks are larger than that of sigmoid. Therefore, $\tanh(x)$ has less of a vanishing gradient problem.

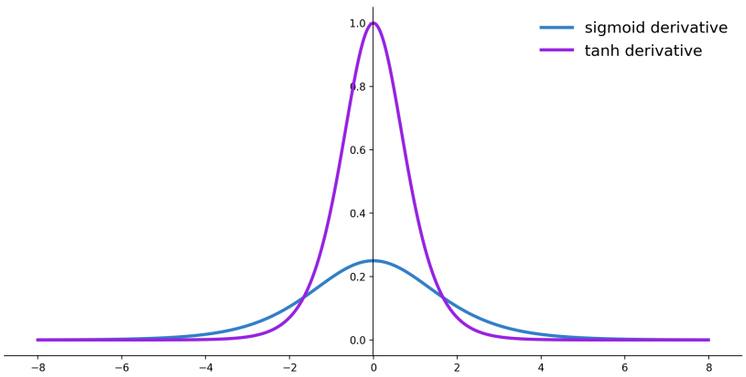


Figure 3: Derivative of $\sigma(x)$ and $\tanh(x)$.

4.
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2e^x - (e^x + e^{-x})}{e^x + e^{-x}} = \frac{2e^x}{e^x + e^{-x}} - 1 = \frac{2}{1+e^{-2x}} - 1 = 2\sigma(2x) - 1$$

Q2.1.1

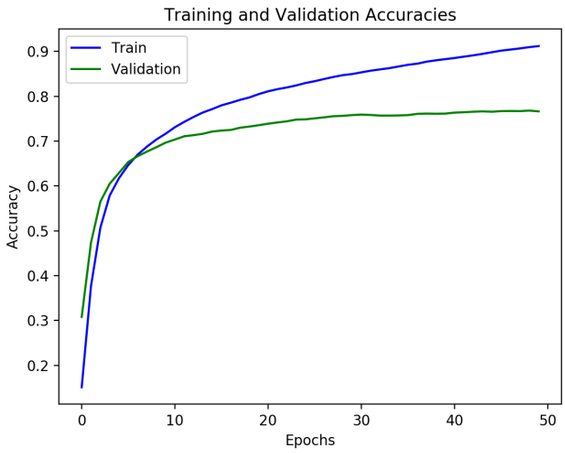
If we initialize a network with all zeros, during forward propagation, each hidden unit will generate the same output, which is zero. During back propagation, the derivatives for all parameters are the same since the outputs of hidden units are the same. The network will update all parameters in the same way. Therefore, after training, the network will output same numbers for each output unit.

Q2.1.3

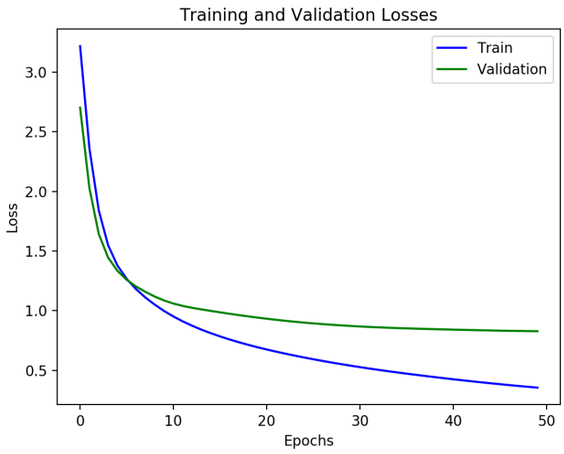
Initializing weights with random numbers will give each weight a different value and stochastic gradient descent will have different gradients for each weight, so that each weight are updated differently. There will be better chance for the model to go down the loss surface and find a local minimum. If initializing weights with same numbers, each weight has the same gradient and are updated exactly in the same way. The model will have little chances to find a local minimum.

We scale the initialization depending on layer size in order to make the variance of the activation function of each layer similar. So no weights start at saturating points, and outputs of activation functions will be around zero which make the gradient large. It could avoid gradient vanishing problem and the learning will become faster.

Q3.1



(a) Train and validation accuracy over 50 epochs.



(b) Train and validation losses over 50 epochs.

Q3.2

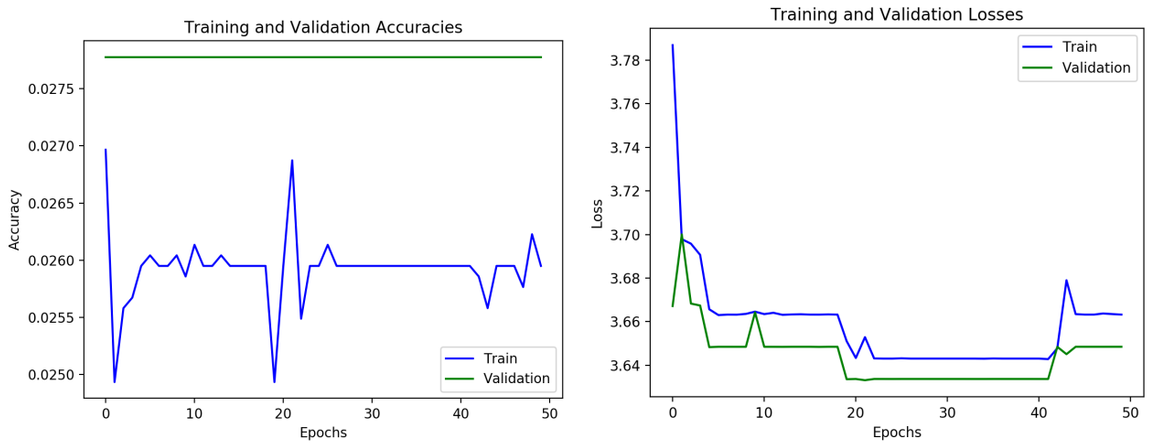


Figure 5: Train/validation accuracy and loss with learning rate 0.05

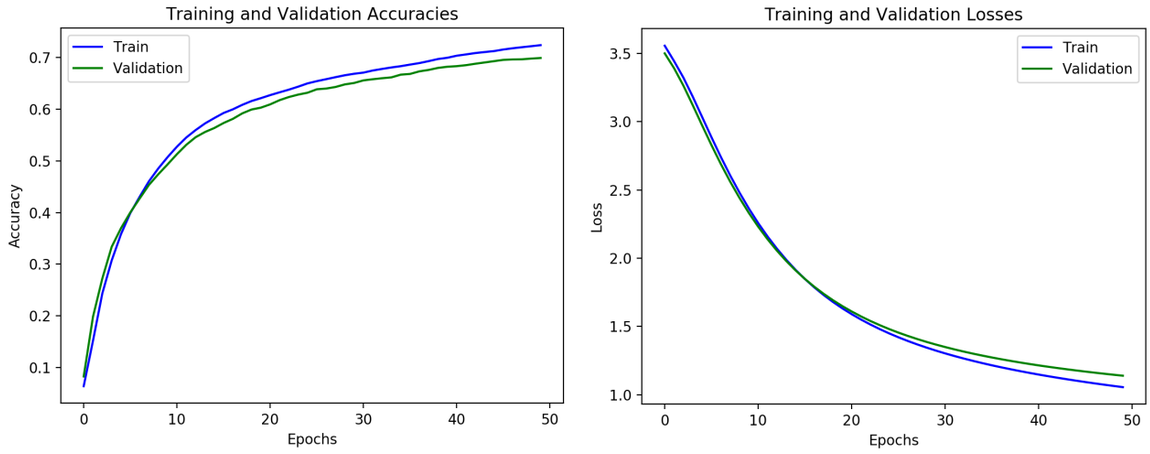
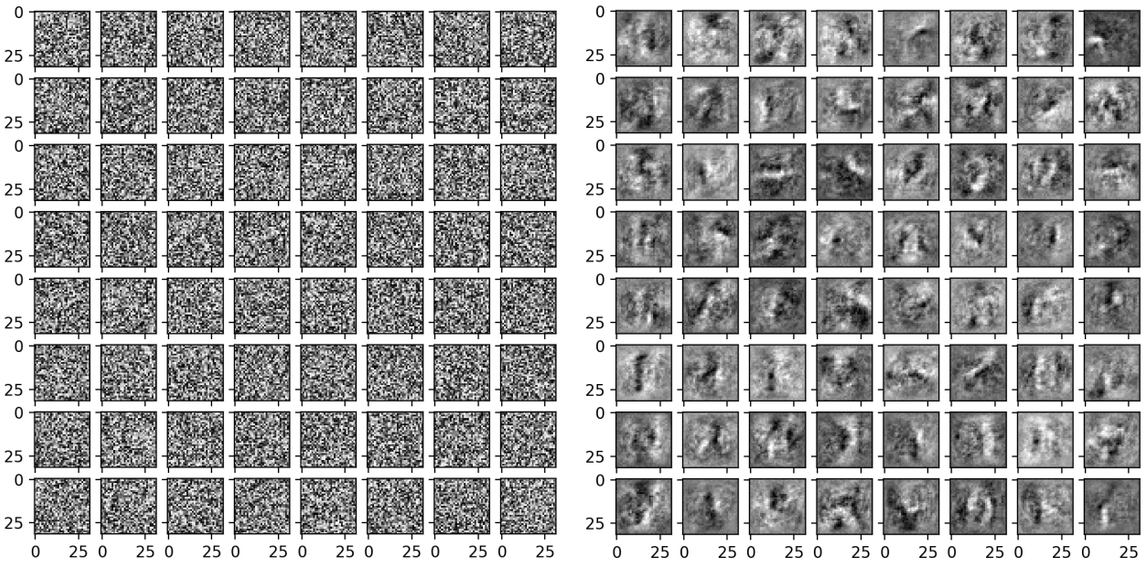


Figure 6: Train/validation accuracy and loss with learning rate 0.0005

In Figure 5, the learning rate is too large that the model doesn't converge. Large learning rate makes the train loss decrease fast at the beginning but also overshoots the local minimum. The model is not able to converge. As shown in Figure 6, model with smaller learning rate does learn to improve the accuracy and will converge, but is very slow.

Final test accuracy of the best model: 0.7711.

Q3.3



(a) Before (b) After

Figure 7: Visualization of first layer weights before and after training.

The weights before training are random noise, and after training it's more organized and shows some patterns. Each weight image in the 8×8 grids represents a possible pattern (stroke) among all upper-case letters and digits. For example, there are vertical lines, horizontal lines, dots and circles.

Q3.4

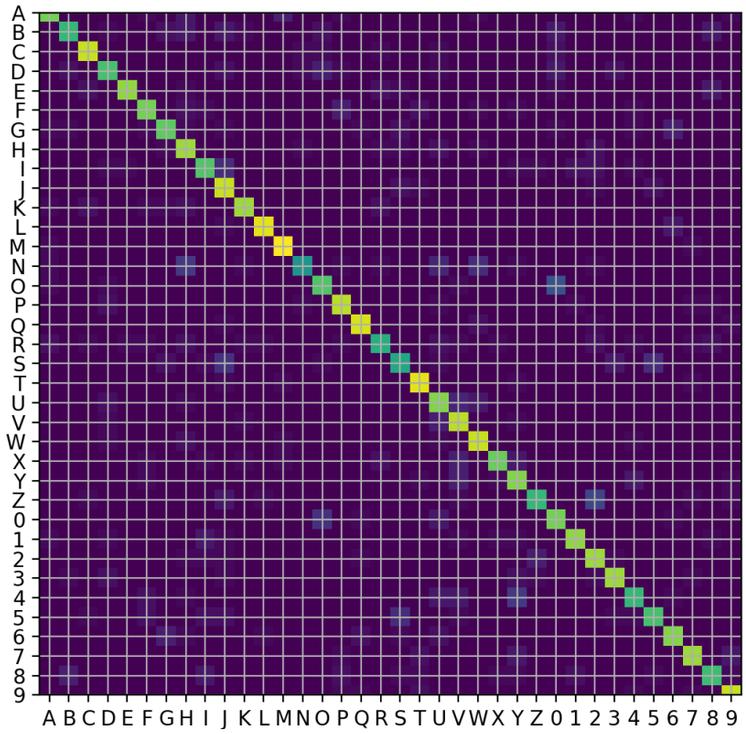


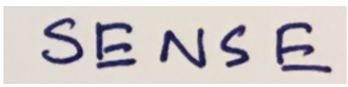
Figure 8: Confusion matrix on test set.

Some most commonly confused pairs are O and 0, Z and 2, 5 and S, whose values in the confusion matrix of both orders are high. These pairs look similar in their strokes. Since their patterns and features are similar, the neural network is easy to misclassify them.

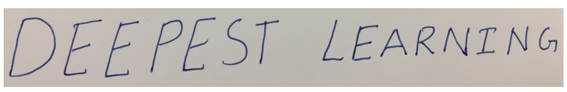
Q4.1

Two big assumptions:
(1) All strokes of a character are connected. There are no connections between characters.
(2) Letters are organized in lines. Letters in a same line have similar sizes.

Some examples that may fail:



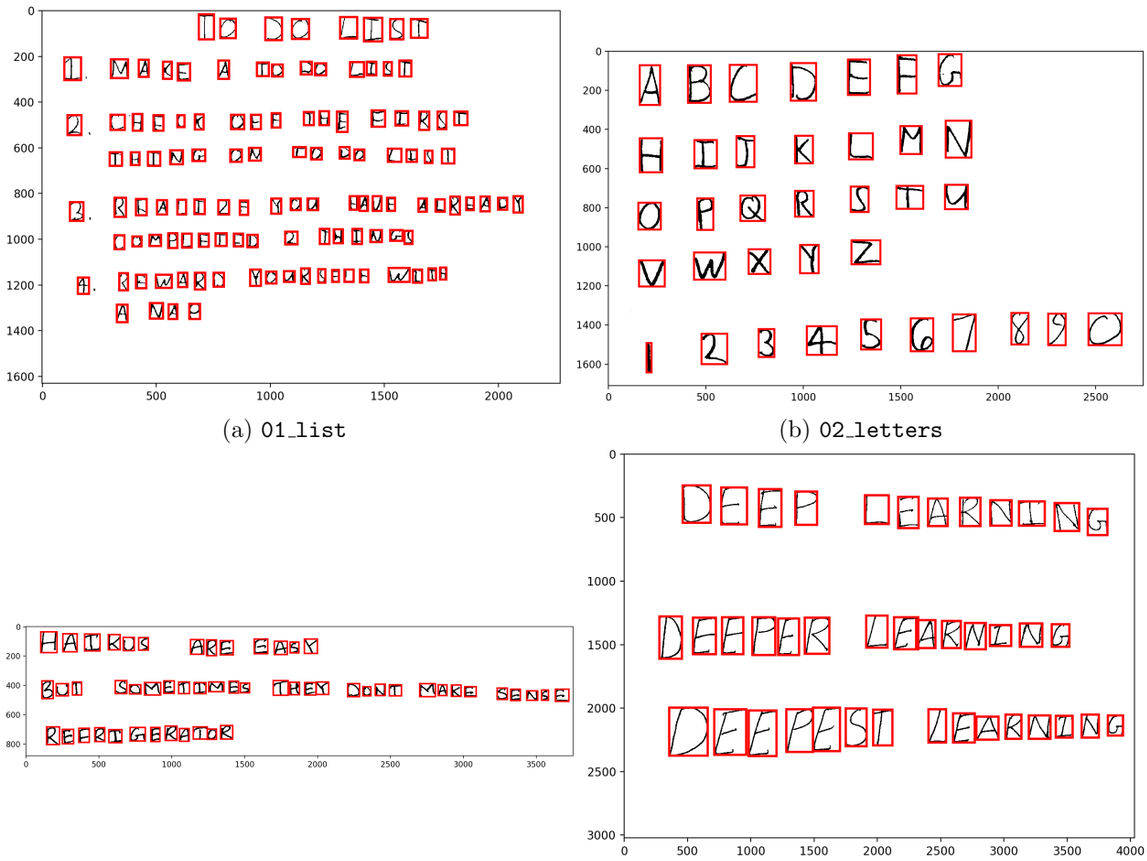
(a) Letter strokes not connected



(b) Letters not on same line

In example (a), strokes of letter E are not connected and the algorithm may put bounding boxes around the horizontal stroke and vertical stroke separately. In example (b), from left to right, sizes of letters keep shrinking. The algorithm may not determine them to be on the same line and group them together.

Q4.3



(a) `01_list` (b) `02_letters`

(c) `03_haiku` (d) `04_deep`

Figure 10: Example images with bounding boxes around letters.

Q4.4

01_list.jpg:

```
TODO LI5T
I MA1E A TODO LI5T
2 CHFCK OFE TH5 FIRST
  THING ON TO OO LI5T
3 RIALIZE YOU HAVE ALREADX
  COMPLEICD 2 THINGS
A REWARD YOURSELF WITH
  A NAP
```

Accuracy: 0.7913043478260869

02_letters.jpg:

```
1 B C D E F 6
H I I K L M N
O P Q R 5 T V
V W X Y Z
1 X 3 4 S 6 7 X 9 0
```

Accuracy: 0.75

03_haiku.jpg:

```
HAIKUS ARE EASY
BUT SCMETIMES TREX DDNT MAKE SENGE
REFRIGERATOR
```

Accuracy: 0.8888888888888888

04_deep.jpg:

```
DEEF LEARMIN6
DEEBER LEARKING
OEEAE5I LEARNING
```

Accuracy: 0.7804878048780488

Q5.2

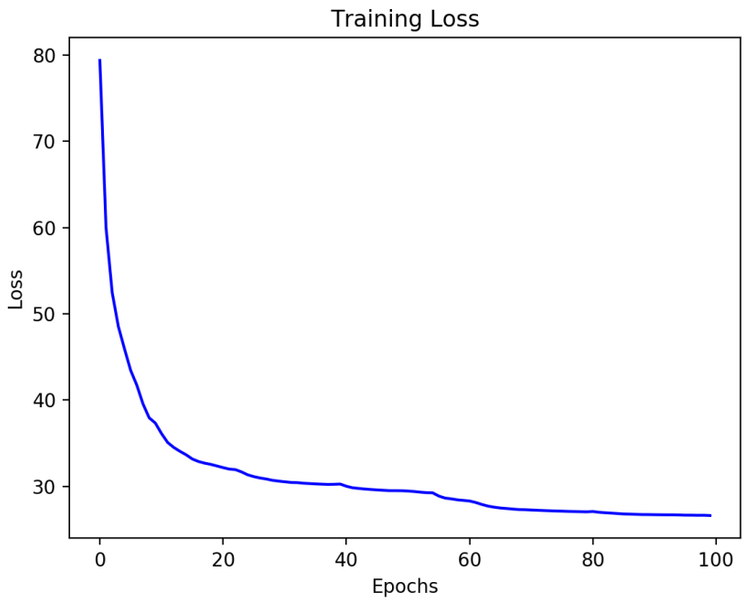


Figure 11: Training loss curve under default parameter settings.

With momentum terms, the training loss drops rapidly from around first 10 epochs, then drops slower and almost goes plateau. Since the learning rate gets multiplied by 0.9 every 20 epochs, there is a little training loss decrease at epoch 20, 40, 60 and 80 in the above Figure. The reason is that when learning rate decreases, the step becomes smaller and the model could go down a small local minimum in the loss surface that was previously overshot by a large learning rate.

Q5.3.1

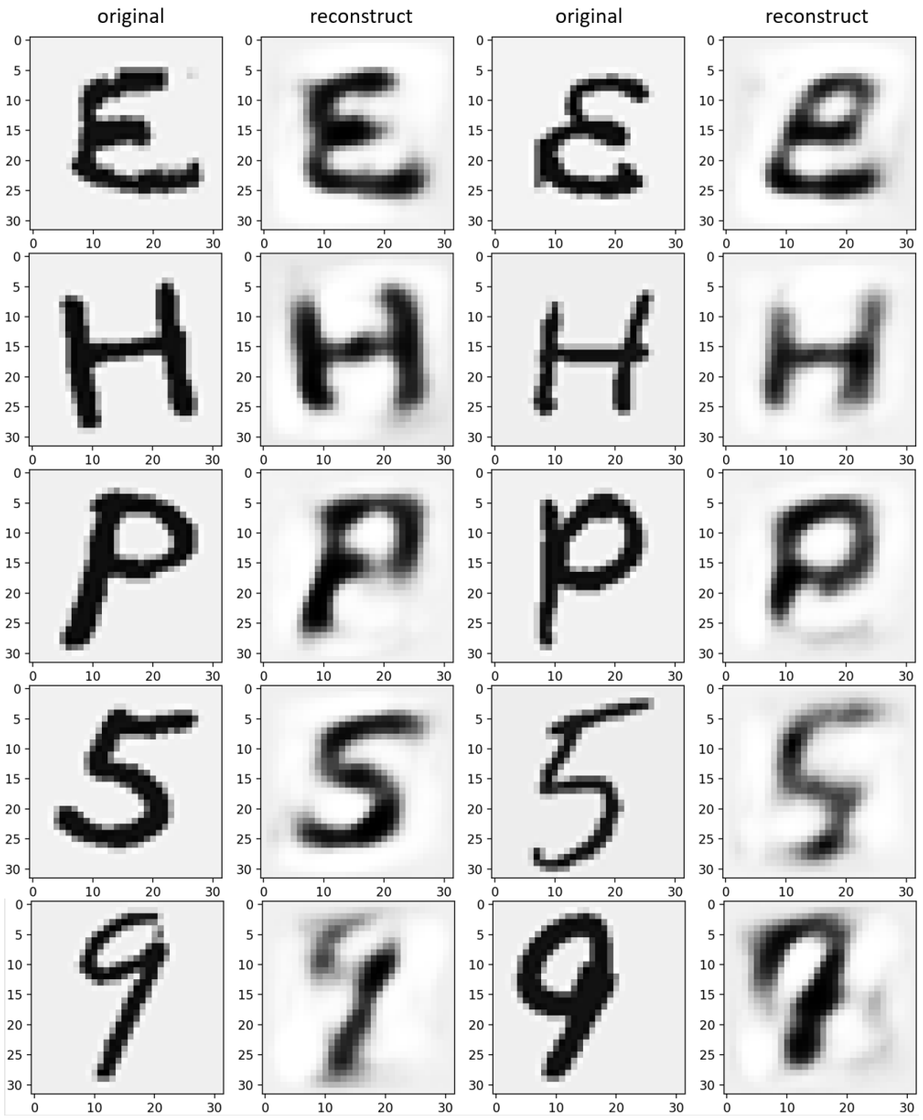


Figure 12: Original validation images and their reconstruction.

The reconstructed images have similar patterns to the original ones but are blurrier. The reason is that during encoding, the dimension of feature representations shrinks from 1024 to 32 which loses some information about the original patterns. When expanding the feature vector dimension back to 1024, there's not enough information to totally reconstruct the original image.

Q5.3.2

Average PSNR on validation set: 15.856.

Q6.1.1

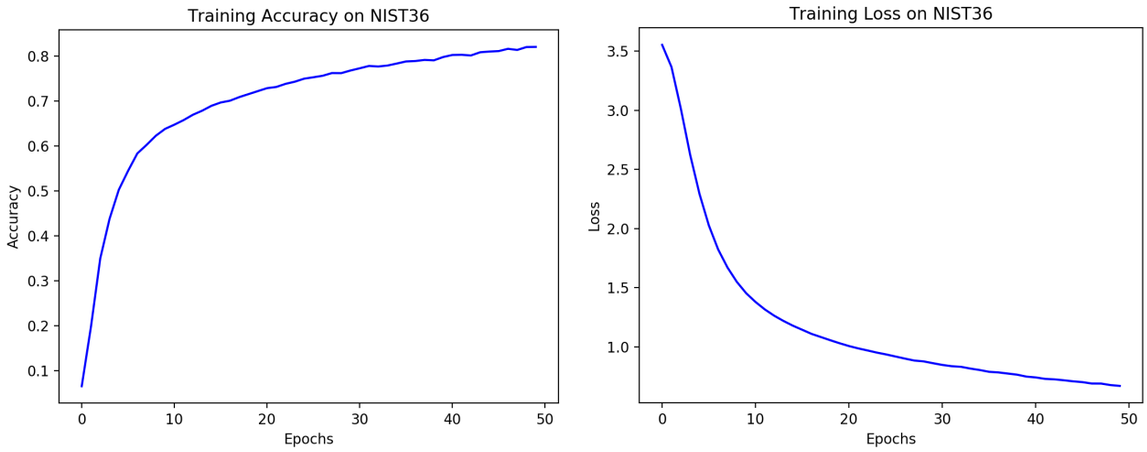


Figure 13: Training accuracy and loss using fully-connected network on NIST36.

Q6.1.2

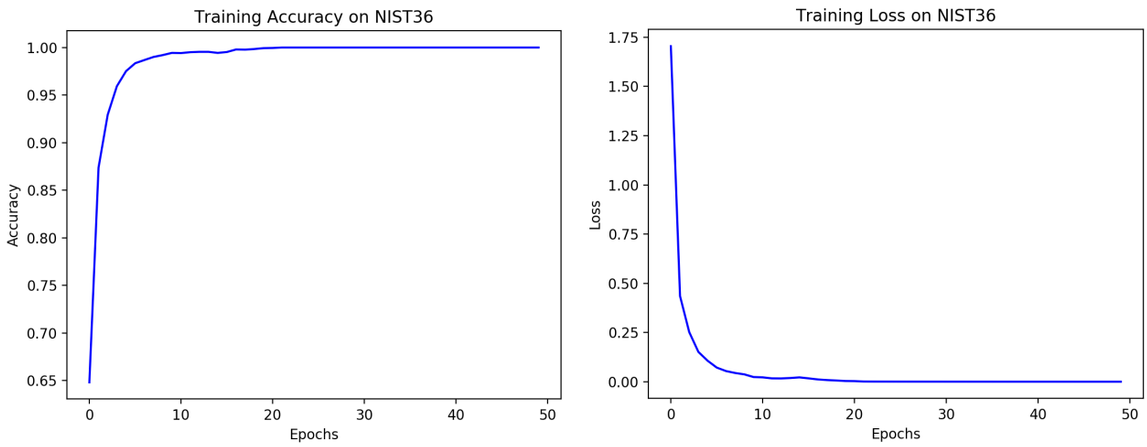


Figure 14: Training accuracy and loss using convolutional neural network on NIST36.

Compared to Figure 13 which is trained by fully-connected network, training with convolutional neural network (CNN) converges faster and achieves higher classification accuracy and lower training loss. The reason is that CNN takes into consideration the spatial information of the image, and extracts different levels of abstraction in different convolution layers. Fully-connected network takes an image as a sequence of pixels which lose the spatial information. Therefore, CNN performs better than fully-connected network.

Q6.1.3

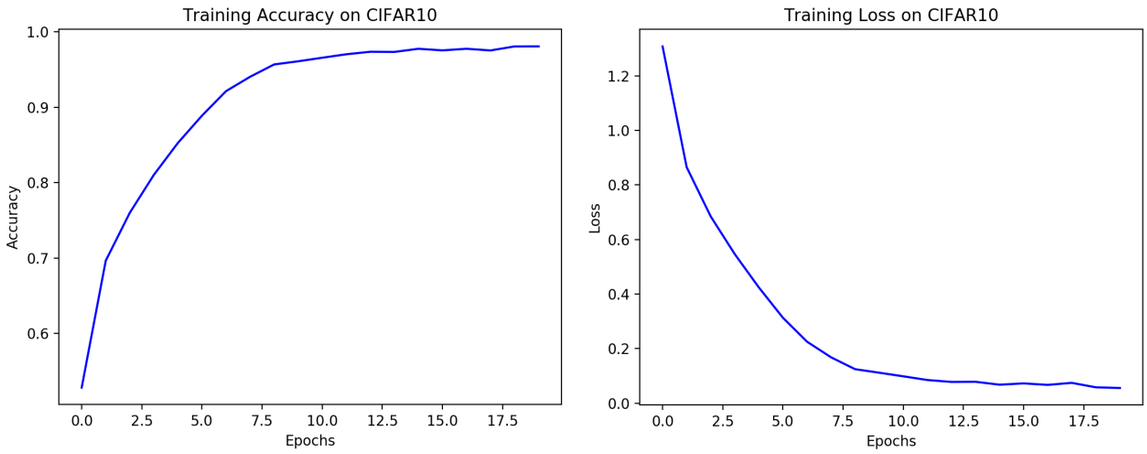


Figure 15: Training accuracy and loss using convolutional neural network on CIFAR10.

Q6.1.4

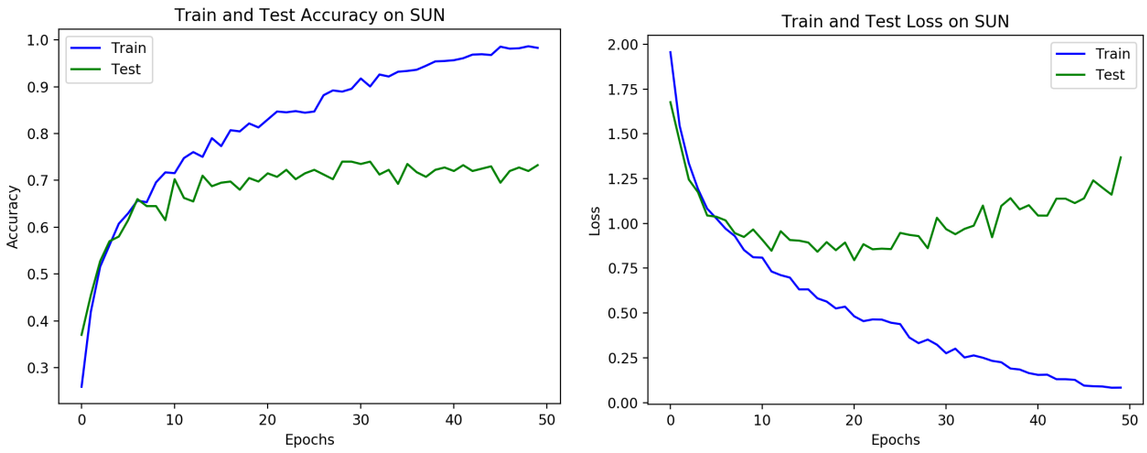


Figure 16: Train and test accuracy and loss using convolutional neural network on SUN database.

The convolutional neural network (CNN) has 74.75% test accuracy, while the best traditional bag-of-words (BoW) approach has 66.75% test accuracy. This is because that the way they choose features are different. BoW runs filters of different kinds and scales on image, clusters filter responses into K groups and uses the count in these K groups as feature vector. The feature vector is kind of hand-crafted because human choose the filter banks, which is limited compared to CNN that automatically learns feature vectors from convolutional results. CNN is also powerful that it learns features of different levels of abstraction in different layers, which could take care of both locality and details. Therefore, CNN outperforms BoW in a large amount.