16-720 Computer Vision
Spring 2020

Homework 5

Zongwen Mu, Andrew ID: zongwenm

1 Theory

Q1.1

For each x_i in vector $\mathbf{x}$, we have:

$$softmax\left(x_i\right) = \frac{\mathrm{e}^{x_i}}{\sum_j \mathrm{e}^{x_j}} \tag{1}$$

$$softmax\left(x_i + c\right) = \frac{\mathrm{e}^{x_i+c}}{\sum_j \mathrm{e}^{x_j+c}} \tag{2}$$

$$= \frac{\mathrm{e}^{x_i}\mathrm{e}^{c}}{\sum_j \mathrm{e}^{x_j}\mathrm{e}^{c}} \tag{3}$$

$$= \frac{\mathrm{e}^{x_i}}{\sum_j \mathrm{e}^{x_j}} \tag{4}$$

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

When $c = -\max x_i$, $x+c$ is always equal or less than 0. This prevents explosion of exponential, so that the result won't overflow.

Q1.2

The range for each element is $(0, 1)$, the sum of all elements is 1;

Softmax takes an arbitrary real valued vector $\mathbf{x}$ andand turns it into a probability distribution.

Calculating $s_i = \mathrm{e}^{x_i}$ is the outcome frequency in exponential form, $\mathbf{S} = \sum s_i$ calculates the total frequency, and dividing each s_i by $\mathbf{S}$ normalizes the frequency of each x_i and gets the probabilty.

Q1.3

When passing through fully connect layers, we have:

$$y_i = W_i x_i + b_i \tag{5}$$

Therefore, when applying multi-layer, we have:

$$
\begin{aligned}
y_n &= W_n x_n + b_n && (6)\\
&= W_n \left(W_{n-1} x_{n-1} + b_{n-1}\right) + b_n && (7)\\
&= W_n W_{n-1} x_{n-1} + W_n b_{n-1} + b_n && (8)\\
&= W' x_{n-1} + b' && (9)\\
&\cdots && (10)\\
&= Wx + b && (11)
\end{aligned}
$$

which is same as linear regression problem.

Q1.4

Derivative of $\sigma(x)$:

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}\sigma(x) &= \frac{\mathrm{d}}{\mathrm{d}x}\left(1+\mathrm{e}^{-x}\right)^{-1} && (12)\\
&= \frac{\mathrm{e}^{-x}}{\left(1+\mathrm{e}^{-x}\right)^2} && (13)\\
&= \frac{1}{1+\mathrm{e}^{-x}}\frac{1+\mathrm{e}^{-x}-1}{1+\mathrm{e}^{-x}} && (14)\\
&= \sigma(x)\left[1-\sigma(x)\right] && (15)
\end{aligned}
$$

Q1.5

$$
\begin{aligned}
y &= Wx + b && (16)\\
\frac{\partial J}{\partial W} &= \frac{\partial J}{\partial y}\frac{\partial y}{\partial W} && (17)\\
&= \delta x^{\mathrm{T}} && (18)\\
\frac{\partial J}{\partial x} &= \frac{\partial J}{\partial y}\frac{\partial y}{\partial x} && (19)\\
&= W^{\mathrm{T}}\delta && (20)\\
\frac{\partial J}{\partial b} &= \frac{\partial J}{\partial y}\frac{\partial y}{\partial b} && (21)\\
&= \delta && (22)
\end{aligned}
$$

Q1.6

1. The derivative of sigmoid function is range in $(0, 0.25)$, which is rather small. Therefore, when applied in multiple layers, it may cause "gradient vanish".

2. The output range of sigmoid function is $(0, 1)$, while the output range of tanh function is $(-1, 1)$. Tanh function is preferred because it could reach the negative part when x is negative.

3. The derivative of tanh function has a range of $(0, 1)$, therefore, the gradient would drop slower and is less likely to cause gradient vanish compared with sigmoid function.

4.

$$\sigma(x) = \frac{1}{1+\mathrm{e}^{-x}} \tag{23}$$

$$\frac{1-\mathrm{e}^{-x}}{1+\mathrm{e}^{-x}} = 2\sigma(x) - 1 \tag{24}$$

$$tanh(x) = \frac{1-\mathrm{e}^{-2x}}{1+\mathrm{e}^{-2x}} \tag{25}$$

$$= 2\sigma(2x) - 1 \tag{26}$$

2 Implement a Fully Connected Network

2.1 Network Initialization

Q2.1.1

Since we need to multiply the inputs of each layer by the weights, if the network is initialized with all zeros, the output from the network will all be zero, and the probabilities would all be the same.

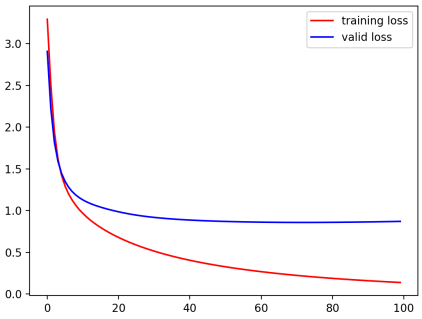
Q2.1.3

Initializing the network with random numbers can avoid getting the same computations from each layer. Scaling the initialization depending on layer size could help keep the variance around desired values when doing forward and backwards propagation.

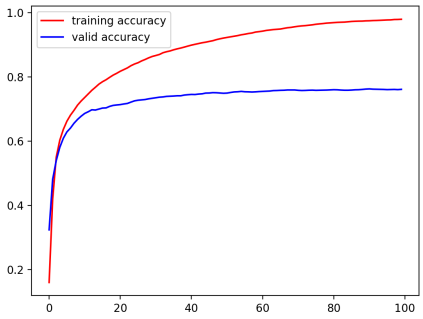
3 Training Models

Q3.1

With learning rate modified to $3e{-}3$, batch size set to 32, after 100 iterations, the valid accuracy is 76.25%. Since the valid loss is quite insignificant compared with training loss, so I averaged training loss by the number of samples, the loss and accuracy figures are shown below:



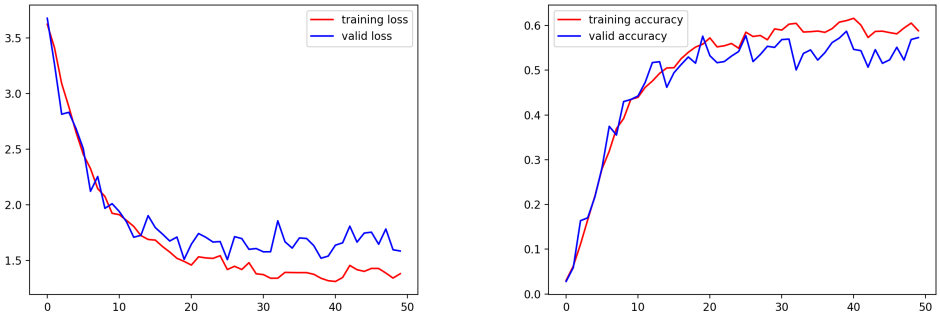
(a) Loss Figure



(b) Accuracy Figure

Figure 1: Loss and Accuracy Figures

Q3.2 With learning rate $3e-2$, the figures are look like:

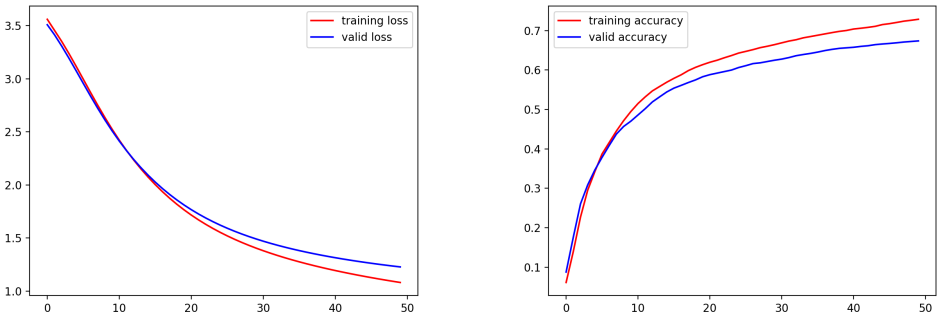


(a) Loss Figure (b) Accuracy Figure

Figure 2: Loss and Accuracy Figures

The losses are higher and the accuracy dropped, and there is also oscillation occurred.

With learning rate $3e-4$, the figures are look like:



(a) Loss Figure (b) Accuracy Figure

Figure 3: Loss and Accuracy Figures

The curves are smooth but since the step is too small, the network didn't converge to the optimum within the max iteration number 100.

Q3.3

The visualization of the initialized first layer weights were shown below:

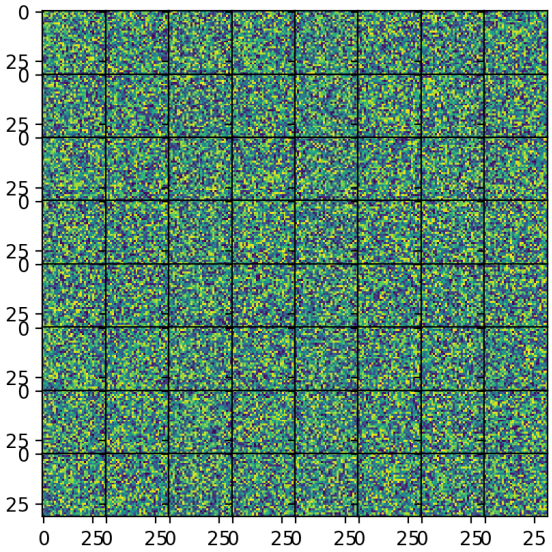


Figure 4: Initialized Weights

The visualization of learned first layer weights were shown below:

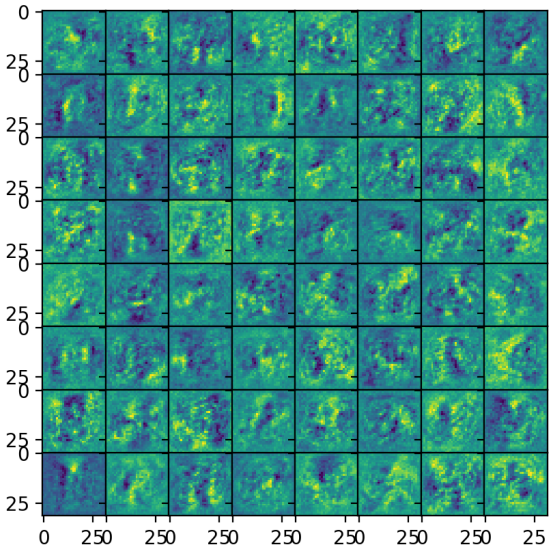


Figure 5: Trained Weights

Comparing these two figures, we could find that the initialized weights look like random noise since we initialize the weights with random uniform distribution. After training, the learned weights are more clear patterns.

Q3.4

The visualized confusion matrix is shown below:

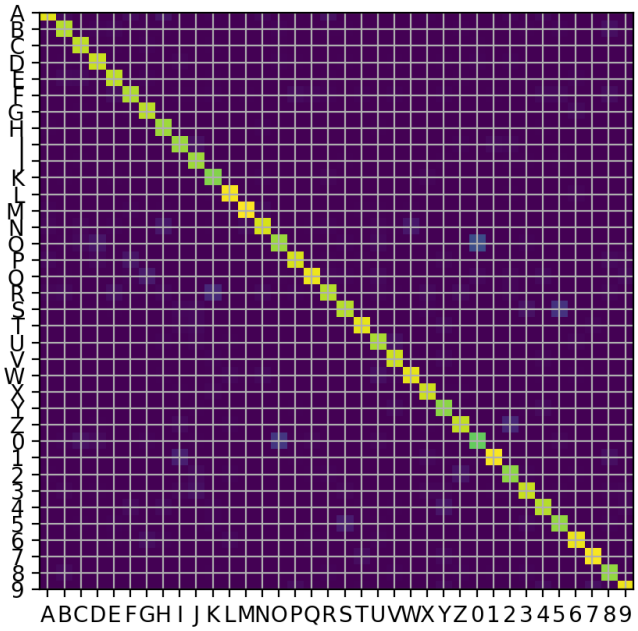


Figure 6: Confusion Matrix

From the figure we can tell the most commonly confused pairs are: ‘O‘ and ‘0‘, ‘s‘ and ‘5‘, which have similar shape that would often be misjudged easily.

4 Extract Text from Images

Q4.1

The assumptions are:

1. Every letter is fully connected.

2. Two different letters are not connected.

Since we extract the letters by finding connected pixels, so if parts of the character are separated or there are several connected characters, it may cause error in detection.

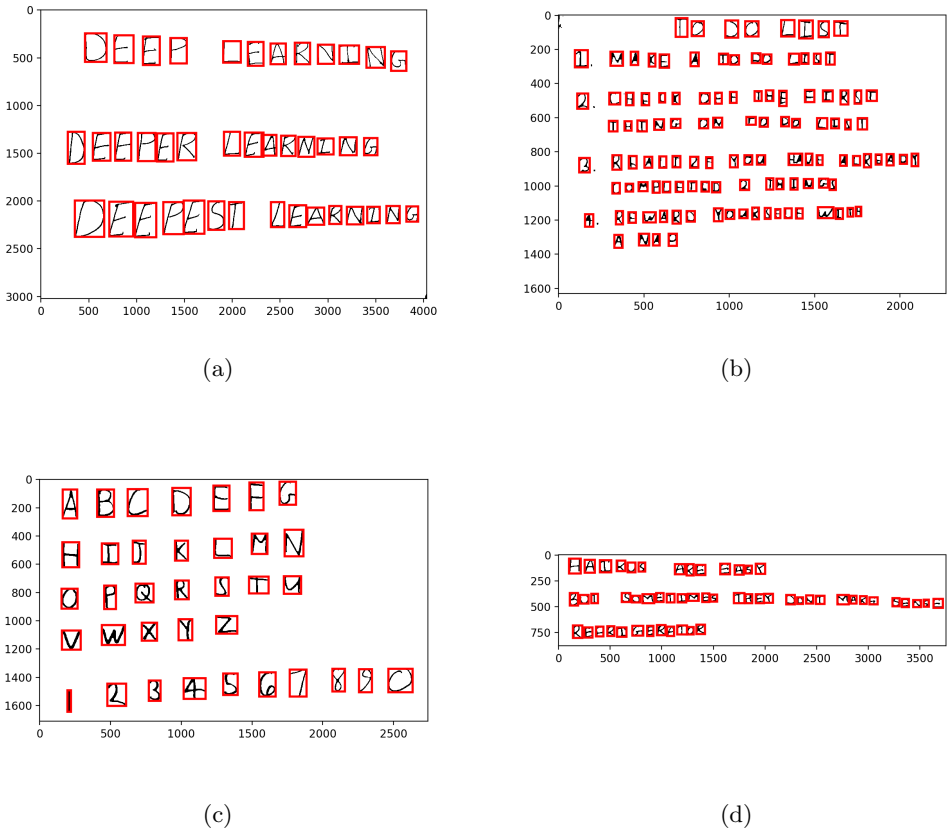
Example images:

(a) Letter not fully connected (b) Connected letters

Figure 7: Examples of Possible Detection Failure

Q4.3

All letters were detected correctly, results are shown below:



(a) (b)

(c) (d)

Figure 8: Detection Results

5 Image Compression with Autoencoders

5.2 Training the Autoencoder

Q5.2

With default settings and batch size 36, learning rate $3e-5$, the loss curves are shown below:

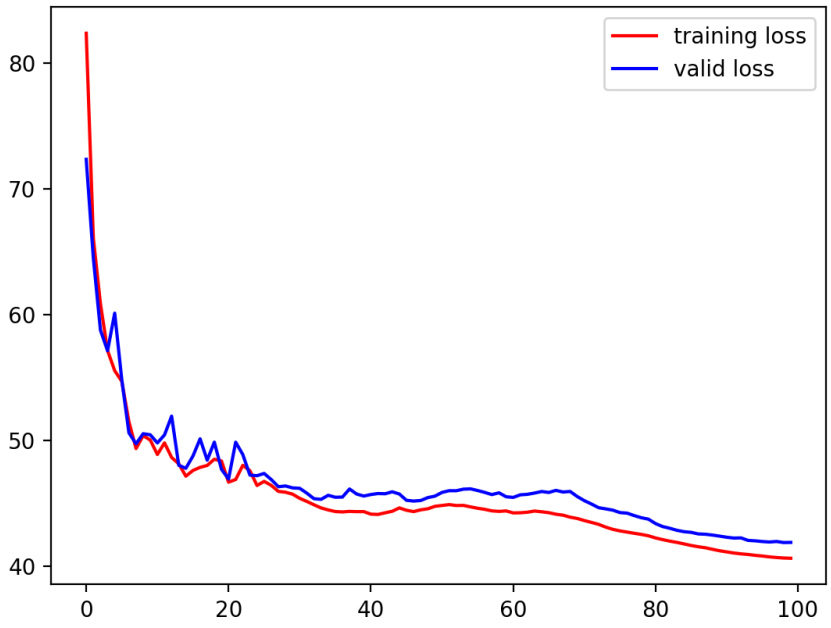


Figure 9: Loss Figure

As training epochs increase, the loss dropped in a vibrating way, but it became smoother at the end of training.

5.3 Evaluating the Autoencoder

Q5.3.1

Class 2:

(a) (b)

Figure 10: Class 2

Class A:

(a) (b)

Figure 11: Class A

Class H:

(a) (b)

Figure 12: Class H

Class O:

(a) (b)

Figure 13: Class O

Class V:

(a) (b)

Figure 14: Class V

We could approximately tell the shapes of reconstructed figures are quite similar to the original ones, but autoencoder is unable to fully reconstruct the figure, so the reconstructed figures are blurred.

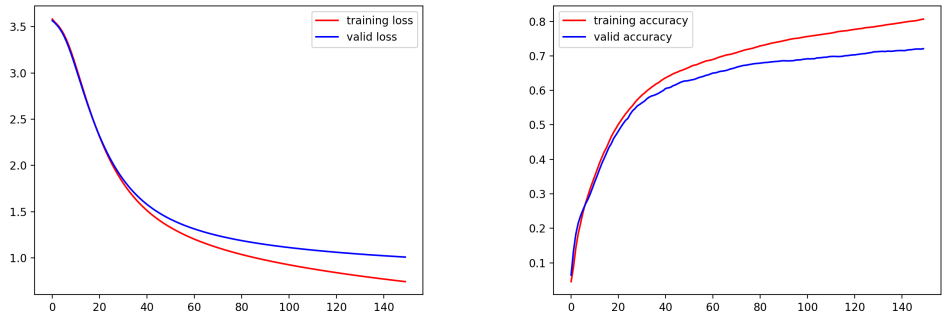
Q5.3.2

With default settings and batch size 36, learning rate $3e{-}5$, the PSNR is 13.52373364094888.

6 PyTorch

6.1 Train a Neural Network in PyTorch

Q6.1.1



(a) Loss Figure (b) Accuracy Figure

Figure 15: Loss and Accuracy Figures

Q6.1.2

The structure of convolutional network is:

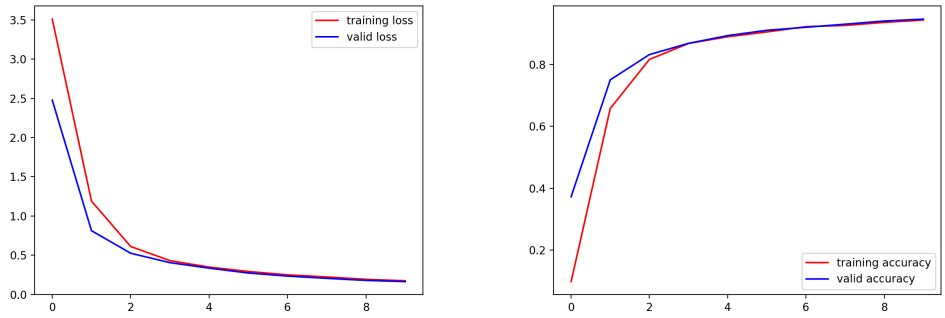
```
class ConvNet(nn.Module):
    def __init__(self, num_classes=36):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 20, 5, 1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(20, 50, 5, 1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.fc1 = nn.Linear(50*5*5, 512)
        self.fc2 = nn.Linear(512, num_classes)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = x.view(-1, 5*5*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return x
```

Figure 16: Network Structure

and the results are shown below:



(a) Loss Figure

(b) Accuracy Figure

Figure 17: Loss and Accuracy Figures

Compared with the fully-connected network, the convolutional network converged faster and reached a significantly higher accuracy, and the loss in convolutional network dropped sharply at the first two iterations.