Homework 5: Neural Networks for Recognition

Yasser Corzo

November 2023

Q1.1

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

Q1.2

- The range of each element is between 0 and 1. The sum of all the elements is 1.
- One could also say that softmax takes an arbitrary real valued vector x and turns it into a probability distribution.
- When performing softmax, we turn every number in the vector $\boldsymbol{x}$ into a positive number by taking the exponent and then sum over all the exponents, making the probability equal to 1. We divide by this sum to give the probability.

Q1.3

WLOG assume there are n layers in our NN (hence n-2 hidden layers) where

$$W_i \in \mathbb{R}^{K\text{x}D}, b_i \in \mathbb{R}^{K\text{x}1}, x \in \mathbb{R}^{D\text{x}1}$$

$$h_1 = W_1x + b_1$$

$$h_2 = W_2 * f_1(W_1x + b_1) + b_2 = W_2 * f_1(h_1) + b_2$$

$$h_3 = W_3 * f_2(h_2) + b_3$$

$$\vdots$$

$$h_{n-1} = W_{n-1} * f_{n-2}(h_{n-2}) + b_{n-1}$$

By this pattern, $h_{n-2} = W_{n-2} * f_{n-3}(W_{n-3} * f_{n-4}(\dots(W_2 * f_1(W_1x + b_1) + b_2)\dots) + b_{n-3}) + b_{n-2}$
Since there is no non-linear activation function, $f_n(W_nx + b_n) = W_nx + b_n$, $\forall n \in \mathbb{Z}$, which is linear.
Hence $h_{n-2} = W_{n-2}(W_{n-3}(\dots(W_2(W_1x + b_1) + b_2)\dots) + b_{n-3}) + b_{n-2}$
Hence $h_{n-1} = W_{n-1}(W_{n-2}(\dots(W_2(W_1x + b_1) + b_2)\dots) + b_{n-2}) + b_{n-1}$
h_{n-1} is composed of multiple linear functions that are the inputs of other linear functions, hence h_{n-1} is linear. We can write h_{n-1} as follows:

$$h_{n-1} = W_{n-1} * X + b_{n-1}$$

$$y = f_{n-1}(h_{n-1}) + b_n = h_{n-1} + b_n = W_{n-1} * X + b_{n-1} + b_n$$

Let $\beta_1 = W_{n-1}$ & $b_{n-1} + b_n = \beta_0$

$$y = \beta_1 * X + \beta_0$$

. Therefore a linear regression by definition.

Q1.4

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

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

$$= \frac{e^{-x}+1}{(e^{-x}+1)^2} - \frac{1}{(e^{-x}+1)^2} \tag{3}$$

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

Q1.5

We know
$$\frac{\partial J}{\partial y} = \delta \in \mathbb{R}^{kx1}, W \in \mathbb{R}^{kxd}, x \in \mathbb{R}^{dx1}$$
and that
$$y_i = \sum_{j=1}^{d} x_j W_{ij} + b_i$$

First take the derivatives of the scalars:

$$\frac{\partial y_i}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_{j=1}^{d} x_j W_{ij} + b_i = W_{ij} \tag{5}$$

$$\frac{\partial y_i}{\partial b_j} = \frac{\partial}{\partial b_j} \sum_{j=1}^{d} x_j W_{ij} + b_i = 1 \tag{6}$$

$$\frac{\partial y_i}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \sum_{j=1}^{d} x_j W_{ij} + b_i = x_j \tag{7}$$

Since we know that $\frac{\partial J}{\partial W} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial W}$, $\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x}$, $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial b}$, as a result of chain rule:

$$\frac{\partial J}{\partial x} = \sum_{i=1}^{k} \frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial x} = W * \delta \tag{8}$$

$$\frac{\partial J}{\partial b} = \sum_{i=1}^{k} \frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial b} = \delta \tag{9}$$

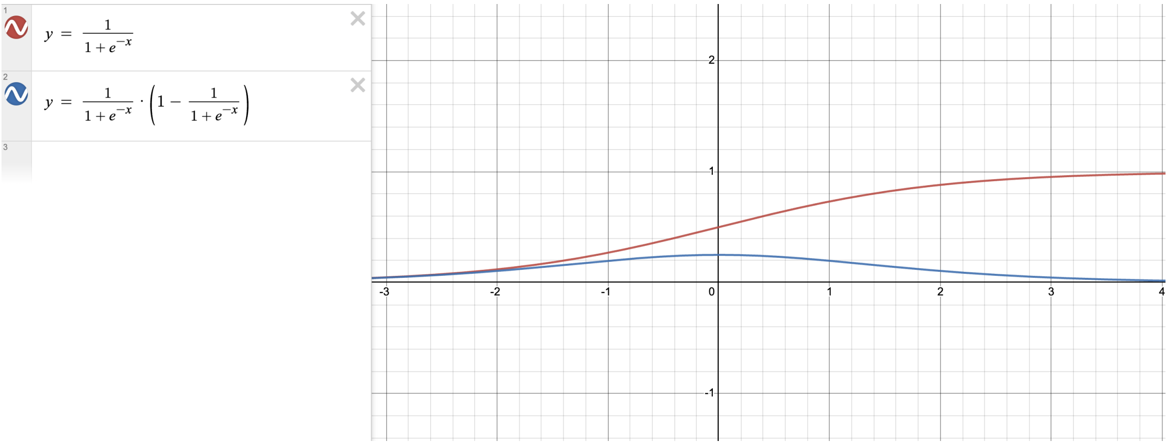
$$\frac{\partial J}{\partial W} = \sum_{i=1}^{k} \frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial W} = x * \delta^T \tag{10}$$

Q1.6

1. If we have multiple layers with sigmoid activation functions, derivative of the output for backpropagation will look like:

   $$\frac{\partial y}{\partial x} = \sigma^{'}(\sigma(h_n))\sigma^{'}(h_n)\sigma^{'}(h_{n-1})\ldots\sigma^{'}(h_2)\sigma^{'}(x)$$

   We know, from Q1.4, that $\sigma^{'}(x) = \sigma(x)(1-\sigma(x))$

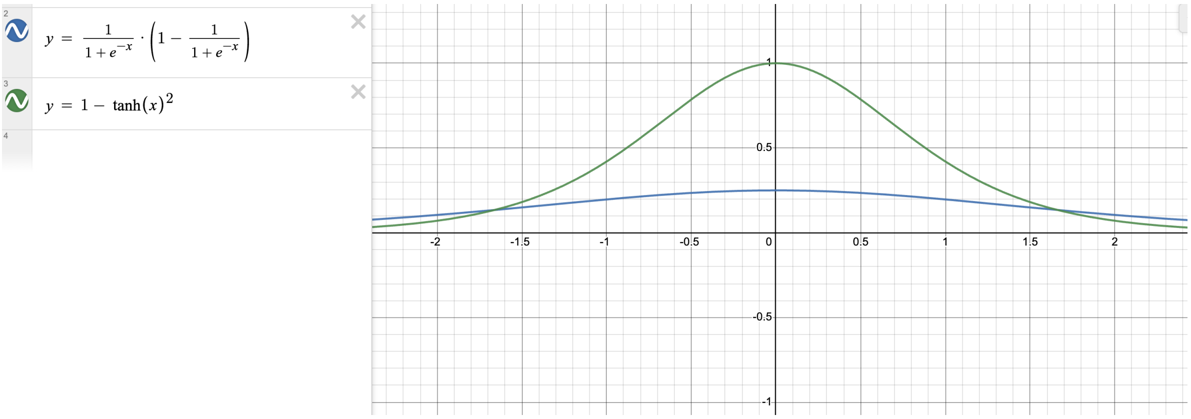
   $y = \frac{1}{1+e^{-x}}$

   $y = \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right)$

   Based on the above graph, the derivative of sigmoid will have output values close to 0. Hence, because of chain rule, derivative of sigmoid will be multiplied multiple times, hence the gradient will be close to 0 when performing backpropagation. Hence the weights will be barely updated, thus causing a ”vanishing” gradient.

2. sigmoid:$[0, 1]$ tanh:$[-1, 1]$ We might prefer tanh because of the range of values outputted is larger than that of sigmoid, hence the gradient will be larger, avoiding the vanishing gradient issue.

3. As shown in the graph below, tanh has a much larger gradient compared to sigmoid as convergence is more definitive than sigmoid, hence suffers less from the ”vanishing gradient” problem.



2 $y = \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right)$

3 $y = 1 - \tanh(x)^2$

4

4.

$$\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} = \frac{1-e^{-2x}}{1+e^{-2x}} * \frac{e^x}{e^x} \tag{11}$$

$$= \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^x + e^{-x} - 2e^{-x}}{e^x + e^{-x}} \tag{12}$$

$$= 1 - 2\frac{e^{-x}}{e^x + e^{-x}} = 1 - \frac{2}{e^{2x}+1} \tag{13}$$

$$= 1 - 2(\sigma(-2x)) = 1 - 2(1 - \sigma(2x)) \tag{14}$$

$$= 2\sigma(2x) - 1 \tag{15}$$

Q2.1.1

If we initialize all weights as 0s, this might cause gradients to also become zero during back propagation. Parameters will never be updated.

Q2.1.2

Implemented in python/nn.py.

Q2.1.3

We initialize with random numbers because when weights in a neural net are initialized with the same value, each perceptron in the neural net learns the same features during training, thus have the same weights. Thus the neural net is unable to generalize over complex patterns. By initializing with random values, different features are learned during training, thus the model is able to learn complex patterns. Initializing with random values also avoids the "vanishing gradient" problem because if all the weights start are initialized with the same value, the gradients for each weight will be the same during backpropagation, and the weights will be updated uniformly, thus increasing the likelihood that convergence will be reached at a local minimum.

We scale the initialization depending on layer size because the variance of the gradients across the layers during backpropagation are roughly the same across the layers. Having gradients of very different magnitudes at different layers can cause slower training, which occurs when weights have the same value where gradients at first may be uniform, but as training progresses they diverge from each other (with larger gradients in the lower layers).

Q2.2.1

Implemented in python/nn.py.

Q2.2.2

Implemented in python/nn.py.

Q2.2.3

Implemented in python/nn.py.

Q2.3.1

Implemented in python/nn.py.

Q2.4

Implemented in python/nn.py.

Q2.5

Implemented in python/run q2.py.

Q3.1

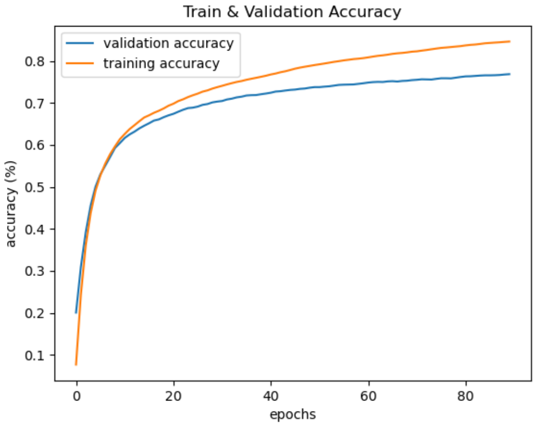


Figure 1: Train & Validation Accuracy Plot

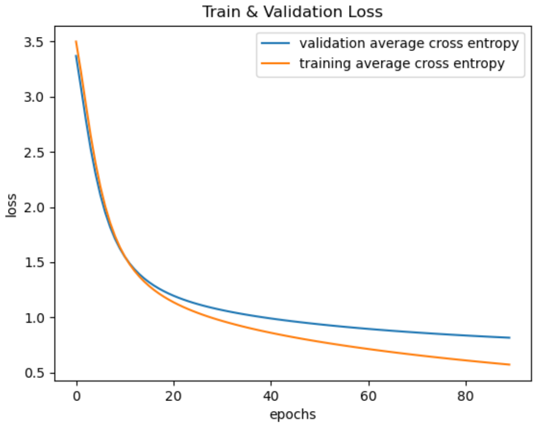
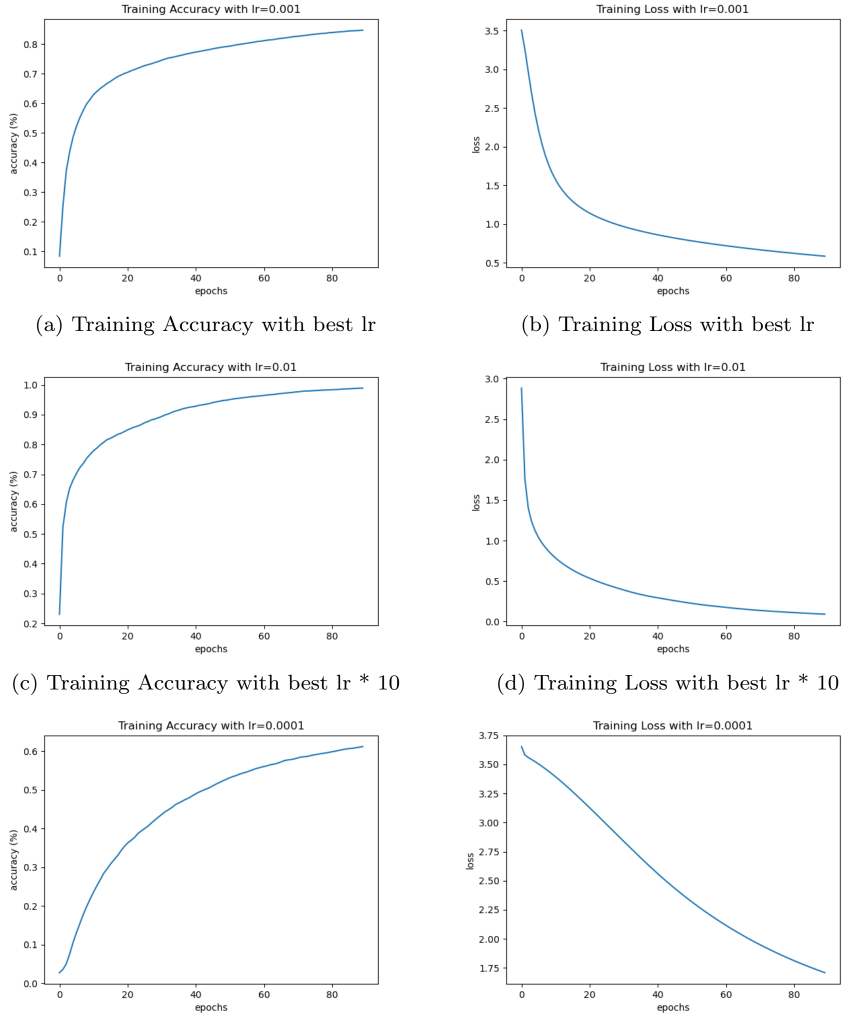


Figure 2: Train & Validation Loss Plot

Q3.2



(a) Training Accuracy with best lr

(b) Training Loss with best lr

(c) Training Accuracy with best lr * 10

(d) Training Loss with best lr * 10

(e) Training Accuracy with best lr * 0.1

(f) Training Loss with best lr * 0.1

The larger the training, the larger the difference is in updating the weights of the model. Hence, convergence is reached much quicker. With a large learning rate, training accuracy increases at a drastic rate for the first few epochs, then plateaus sooner and at a high accuracy, hence the logarithmic shape of the curve. In terms of loss, training loss decreases drastically for the first few epochs, then plateaus at a low loss, hence the exponential decay shape of the curve. On the other hand, with a small learning rate, there is a smaller difference in updating the weights of the model, hence convergence takes longer. Training accuracy

increases over time but at a smaller rate compared to training with a larger learning rate, evident in the training accuracy plot with learning rate of 0.0001 where there's barely any plateau in the accuracy. In terms of training loss, it does decrease but linearly (i.e. at a smaller rate than with a larger learning rate), unlike with a larger learning rate where the curve of the plot of training loss was in an exponential decay fashion.
When running test dataset on this model, final accuracy is 0.7727777777777778.

Q3.3

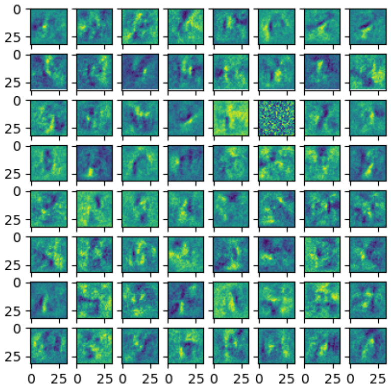


Figure 3: First Layer Weights Visual

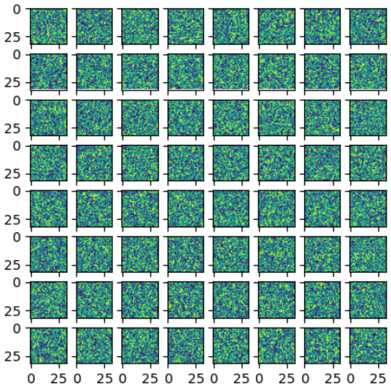


Figure 4: network weights immediately after initialization

Compared to the network weights immediately after initialization, the first layer weights are not randomly distributed. There are some patterns present here.

Q3.4

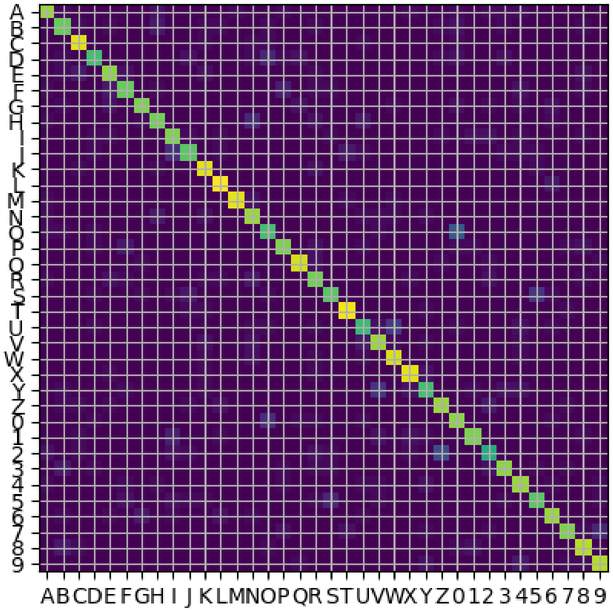


Figure 5: Confusion Matrix of Test set

The top few pairs of classes that are most commonly confused are (0, O), (5, S), (2, Z), and (Y, V). This could be because these pairs look the same, especially when handwritten. Hence it would be harder to distinguish from the other.

Q4.1

Two big assumptions the sample method makes is that the handwritten letters are uppercase & in English and no two letters are connected.

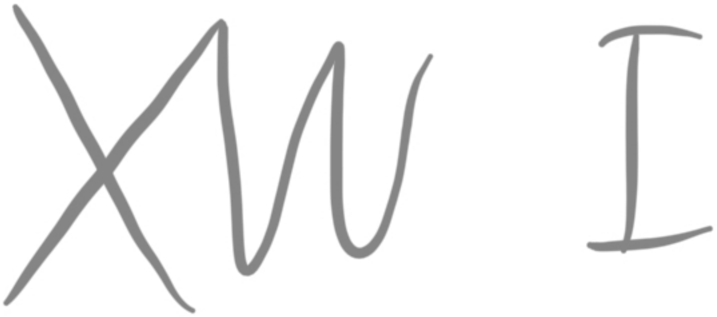


Figure 6: Example of two letters connected (X & W)

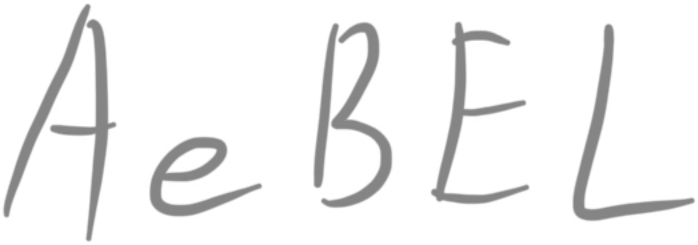
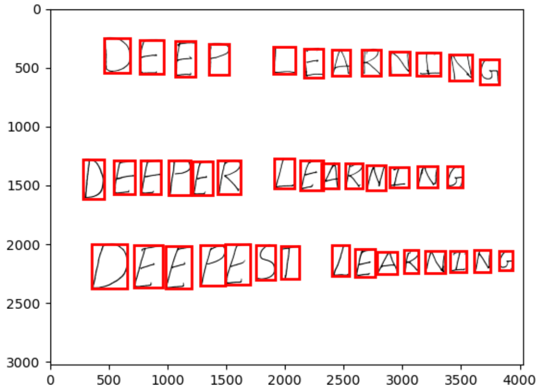


Figure 7: Example of lowercase letter

Q4.2

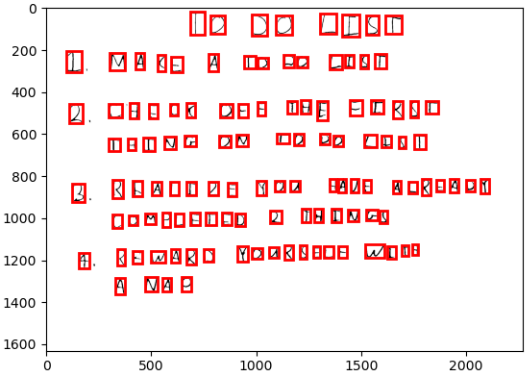
Implemented in python/q4.py

Q4.3



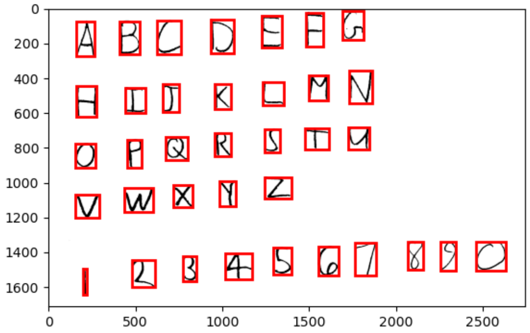
DEEP LEARNING
DEEPER LEARNING
DEEPEST LEARNING
0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500
4000

(a)



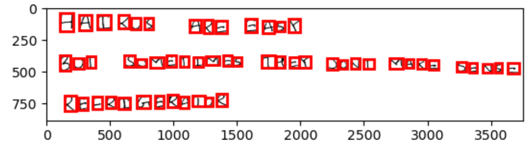
0
200
400
600
800
1000
1200
1400
1600
0
500
1000
1500
2000

(b)



A B C D E F G
H I J K L M N
O P Q R S T U
V W X Y Z
1 2 3 4 5 6 7 8 9 0
0
200
400
600
800
1000
1200
1400
1600
0
500
1000
1500
2000
2500

(c)



HAIKUS ARE EASY
BUT SOMETIMES THEY DONT MAKE SENSE
REFRIGERATOR
0
250
500
750
0
500
1000
1500
2000
2500
3000
3500

(d)

Q4.4

```
DEEP LEARMING
DEBPER LEARHING
DEEPEST LEARNING

TQDQLIST
I NARE A TD DD LIST
2 LH2CK DFF THE FIRST
  THING QN TO DQ LIST
3 RZALIZE YOU HAUE ALR6ADT
  CQMPLFTLD Z THINGS
9 R8WARD YOURSELF WITH
  A NAP

2 B C D E F G
H I J K L M N
Q P Q R S T W
V W X Y Z
1 Z 3 G S 6 7 8 9 Q

HAZKUS ARG GASY
BLT SQMETIMBS TREY DOWT MAKG BGNGE
RBGRIGBRATQR
```

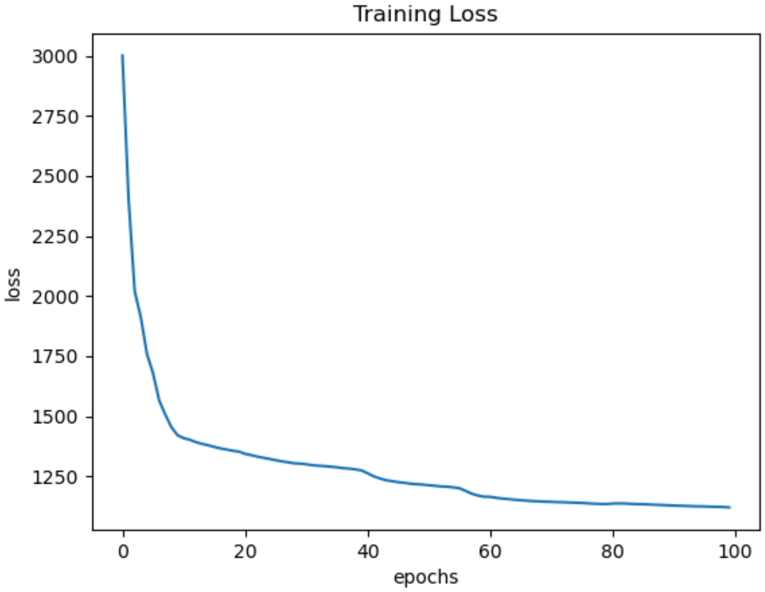
Q5.1.1

Implemented in python/run_q5.py

Q5.1.2

Implemented in python/run_q5.py

Q5.2



In the plot I observed that as epochs increased for the first few iterations, training loss decreased rapidly. This rate of change decreased as more the number of epochs further increased, eventually leading to convergence in training loss.

Q5.3.1

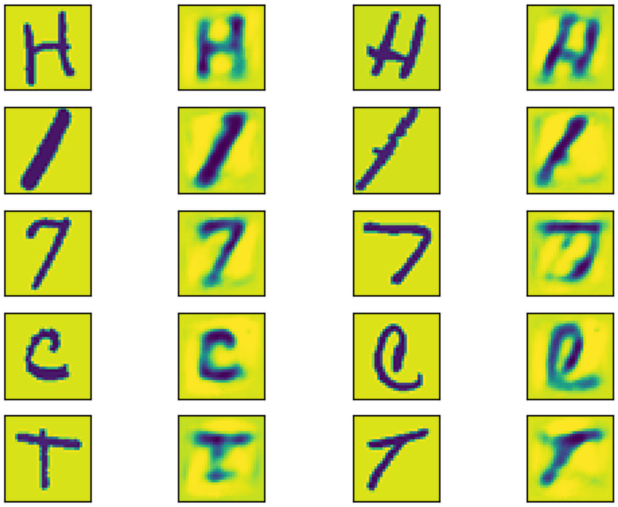


Figure 8: 1st & 3rd columns are validation, 2nd & 4th are reconstructed

Differences that I observed in the reconstructed images, compared to the original ones are that the reconstructed images are much more blurry and noisy than the original images.

Q5.3.2

average PSNR: 15.474077393682263