CMU Computer Vision HW3

Jiaqi Geng

October 2020

1 Theory

1.1

Each element of $x + c$ can be represented as $x_i + c$
Therefore, $softmax(x_i + c) = \frac{e^{x_i+c}}{\sum_j e^{x_j+c}}$
$= \frac{e^{x_i} \cdot e^c}{\sum_j e^{x_j} \cdot e^c} = \frac{e^{x_i} \cdot e^c}{e^c \cdot \sum_j e^{x_j}} = \frac{e^{x_i}}{\sum_j e^{x_j}} = softmax(x_i)$

Some elements of x can be very large, causing overflowing.
Elements of x can be all very small,
causing every e^{x_i} to be 0, and leading to a zero denominator.
We subtract the maximum element from all elements of x,
then each $x_i - c$ becomes non-positive, solving the overflowing issue,
and we would have $x_{max} - c = 0$, where x_{max} is the maximum element,
so $e^{x_{max}} = 1$, solving the zero denominator problem.

1.2

Each element of $softmax$ is between 0 and 1.
The sum of all elements is 1.

$softmax$ takes an arbitrary real valued vector x
and turns it into a probability distribution.

Each element of x can be any arbitrary real values (negative, 0, or positive),
the first step e^{x_i} can turn all of them into positive values.
The second step computes the sum of these positive values for normalization.
The third step normalizes each element to be between 0 and 1.

1.3

Let $y_1 = W_1 x_1 + b_1$
$y_2 = W_2 y_1 + b_2$
Then, $y_2 = W_2(W_1 x_1 + b_1) = (W_2 W_1) x_1 + W_2 b_1 + b_2$
Let $W_3 = W_2 W_1, b_3 = W_2 b_1 + b_2$
We now have $y_2 = W_3 x_1 + b_3$

Thus, multi-layer neural networks without a non-linear activation function are equivalent to linear regression.

1.4

Sigmoid function: $\sigma(x) = (1 + e^{-x})^{-1}$
The gradient of the sigmoid function
$= -(1 + e^{-x})^{-2} \cdot e^{-x} \cdot (-1)$
$= (1 + e^{-x})^{-2} \cdot e^{-x}$
$= (1 + e^{-x})^{-1} \cdot (1 + e^{-x})^{-1} \cdot e^{-x}$
$= \frac{1}{1+e^{-x}} \cdot \frac{1+e^{-x}-1}{1+e^{-x}}$
$= \frac{1}{1+e^{-x}} \cdot (1 - \frac{1}{1+e^{-x}})$
$= \sigma(x) \cdot (1 - \sigma(x))$

1.5

Since $y_j = \Sigma_{i=1}^{d} x_i W_{ji} + b_j$
$\frac{\partial y_j}{\partial W_{ji}} = x_i, \frac{\partial y_j}{\partial x_i} = W_{ji}$

Then, we can find
$\frac{\partial J}{\partial W_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial W_{ji}} = \delta_j x_i$
$\frac{\partial J}{\partial x_i} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial x_i} = \delta_j W_{ji}$
$\frac{\partial J}{\partial b_j} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial b_j} = \delta_j$

Thus,
$\frac{\partial J}{\partial W} = \delta x^T$
$\frac{\partial J}{\partial x} = W^T \delta$
$\frac{\partial J}{\partial b} = \delta$

1.6

For simplicity, I indexed the last two dimensions of W with offsets from the center of the kernels.

$y_{d,i,j} = \Sigma_c \Sigma_p \Sigma_q (W_{d,c,p,q} X_{c,i+p,j+q}) + b_d$

We can see that the same $W_{d,c,p,q}$ will be matrix multiplied by $X_{c,i+p,j+q}$ for all i and j.
Therefore, we fixed d, c, p, q and we sum all the possible X terms (times corresponding δ) to get

$\frac{\partial J}{\partial W_{d,c,p,q}} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial W_{d,c,p,q}} = \Sigma_i \Sigma_j \delta_{d,i,j} X_{c,i+p,j+q}$

We can see that the same $X_{c,k,l}$ will be matrix multiplied by $W_{d,c,k-i,l-j}$ for all i and j and d.
Therefore, we fixed c, k, l and we sum all the possible W terms (times corresponding δ) to get

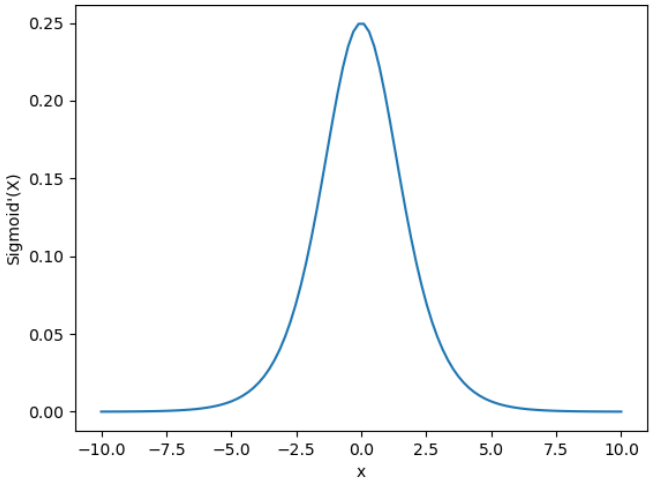
$\frac{\partial J}{\partial X_{c,k,l}} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial X_{c,k,l}} = \Sigma_d \Sigma_i \Sigma_j \delta_{d,i,j} W_{d,c,k-i,l-j}$

Finally, to calculate $\frac{\partial J}{\partial b_d}$, we follow the same logic to fix d,
and sum over all corresponding δ to get

$\frac{\partial J}{\partial b_d} = \frac{\partial J}{\partial y_d} \frac{\partial y_d}{\partial b_d} = \Sigma_c \Sigma_i \Sigma_j \delta_{d,i,j}$

1.7

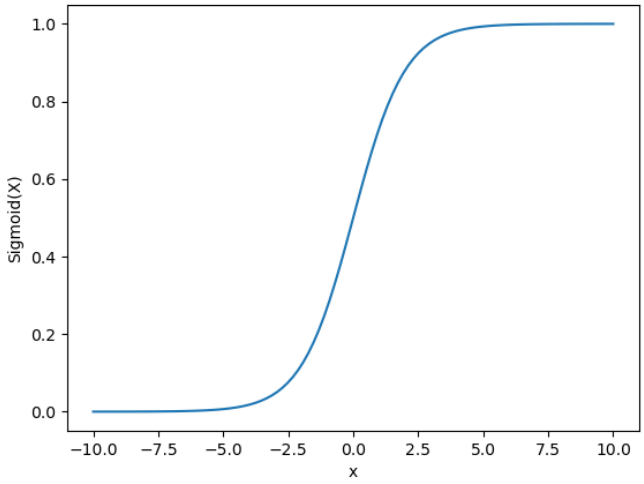
(a)

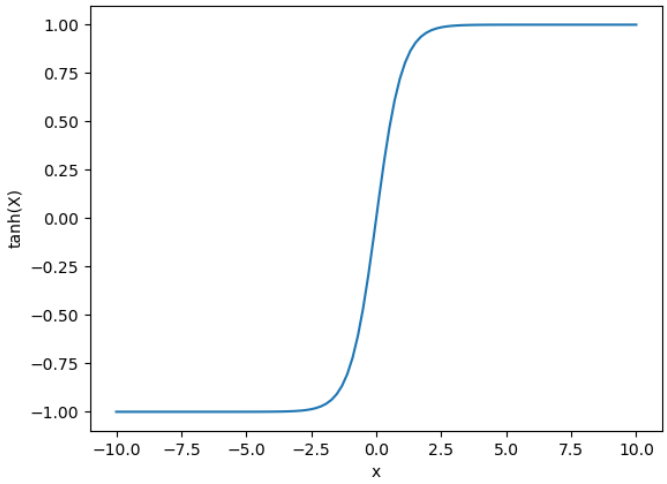


We can see that if the input values are very large or very small,
then the derivatives of sigmoid function will be close to 0.

Also, the maximum of sigmoid's derivative is 0.25.
When we do back propagation through many layers,
the gradients will become very small very quickly (0.25^n).

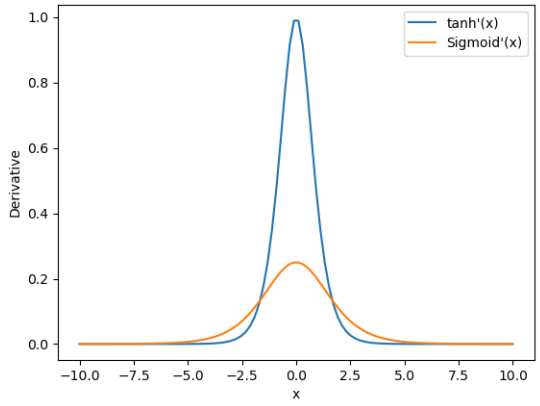
(b)





The output range of sigmoid is 0 to 1.
The output range of tanh is -1 to 1.
Since the output range increases for $\tanh$,
$\tanh(x)$ has larger derivative for x values around the center,
which might alleviate the vanishing gradient problem.

(c)



The $\tanh$ function has a less of a vanishing gradient problem because
it has larger derivative for x values around the center.
When we do back propagation through many layers,
the gradients will tend to have larger values than sigmoid (1^n instead of 0.25^n)

(d)

$\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
We will first try to eliminate the e^{-2x} term in the numerator.

$$
\begin{aligned}
&= \tfrac{1-e^{-2x}}{1+e^{-2x}} + \tfrac{1+e^{-2x}}{1+e^{-2x}} - 1 \\
&= \tfrac{2}{1+e^{-2x}} - 1 \\
&= 2\sigma(2x) - 1
\end{aligned}
$$

2 Implement a Fully Connected Network

2.1.1

If we initialize our network with all zeros,
depending on the activation function we use,
we might have zero or very small gradients,
leading to no or very small updates for weights and biases.

Also, since all weights and biases are initialized to be the same,
all neurons in the hidden layers will be updated in the exact same ways,
which means the model cannot learn anything useful.

The outputs of this network
can be the same or very close before and after training.

2.1.2

See the code for implementation.

2.1.3

We initialized with random numbers
so that neurons in the hidden layers will be updated in different ways.

According to the paper,
The standard initialization will give us variance dependent on the layer size.
We scale the initialization to maintain variance for layers with different sizes.

2.2.1

See the code for implementation.

2.2.2

See the code for implementation.

2.2.3

See the code for implementation.

2.3.1

See the code for implementation.

2.4.1

See the code for implementation

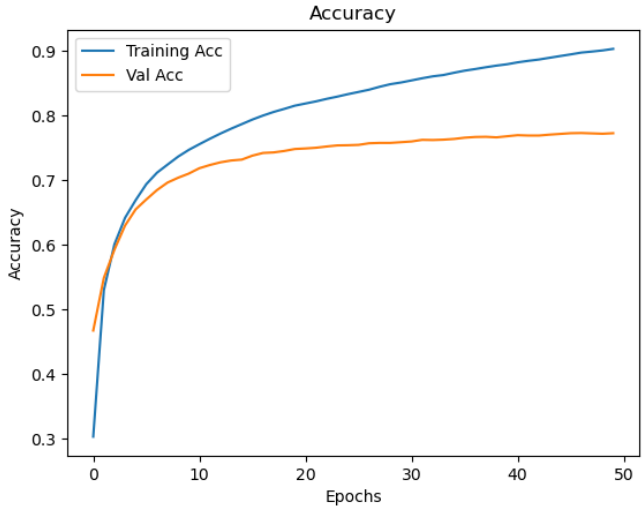
2.5.1

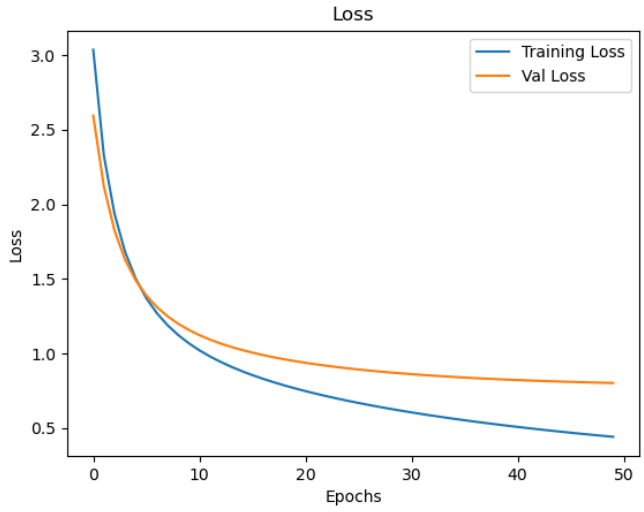
grad_Woutput: 6.69e-09
grad_boutput: 3.02e-11
grad_Wlayer1: 1.53e-07
grad_blayer1: 2.26e-09
total: 1.62e-07

See the code for implementation

3 Training Models

3.1.1

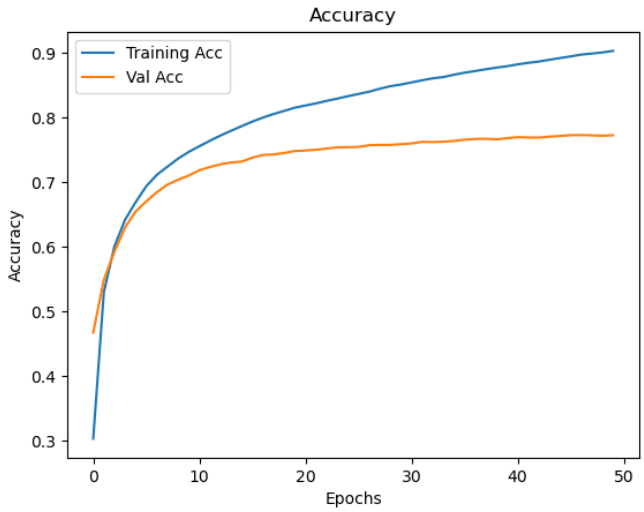


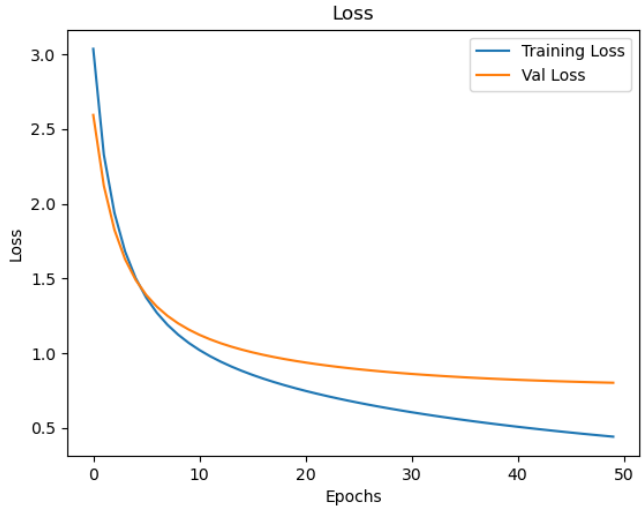


Final validation Acc: 0.7727777777777778

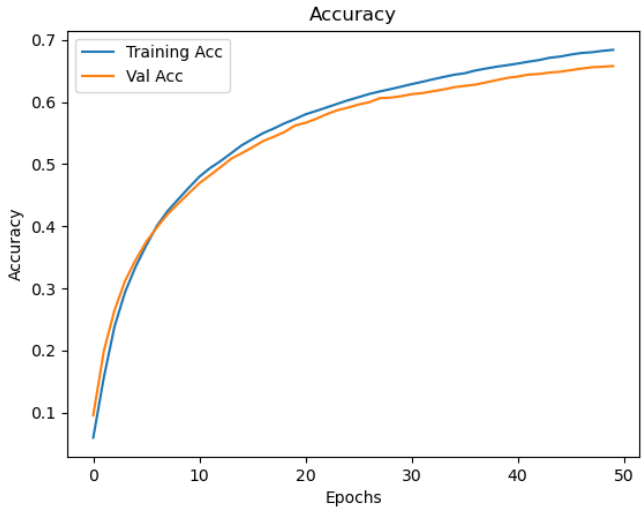
3.1.2

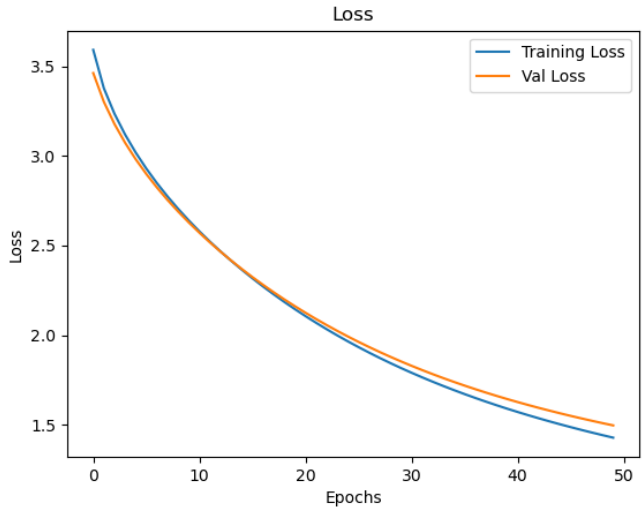
Learning rate: 0.001



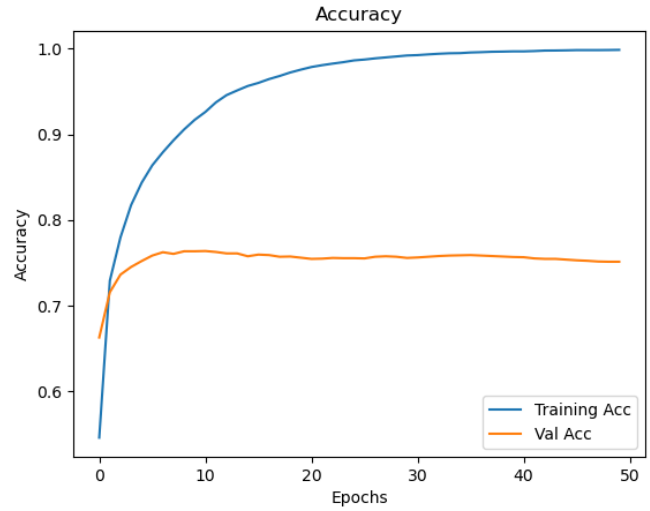


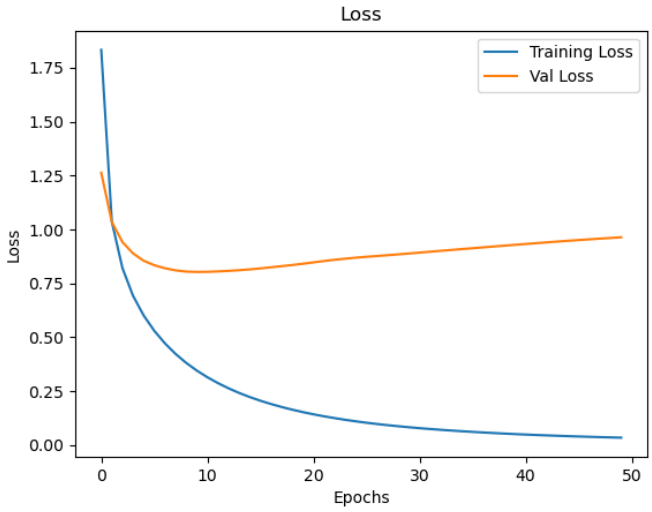
Learning rate: 0.0001





Learning rate: 0.01

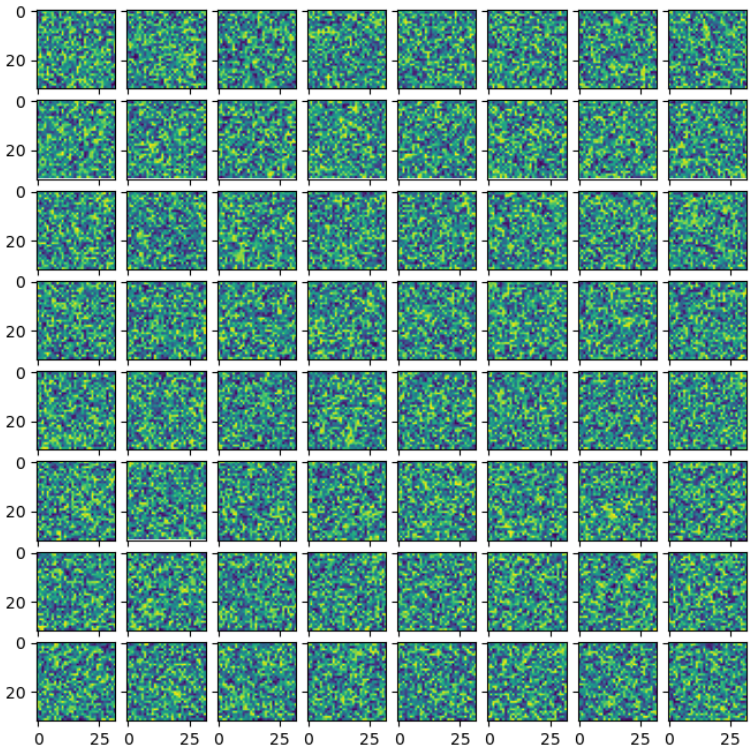




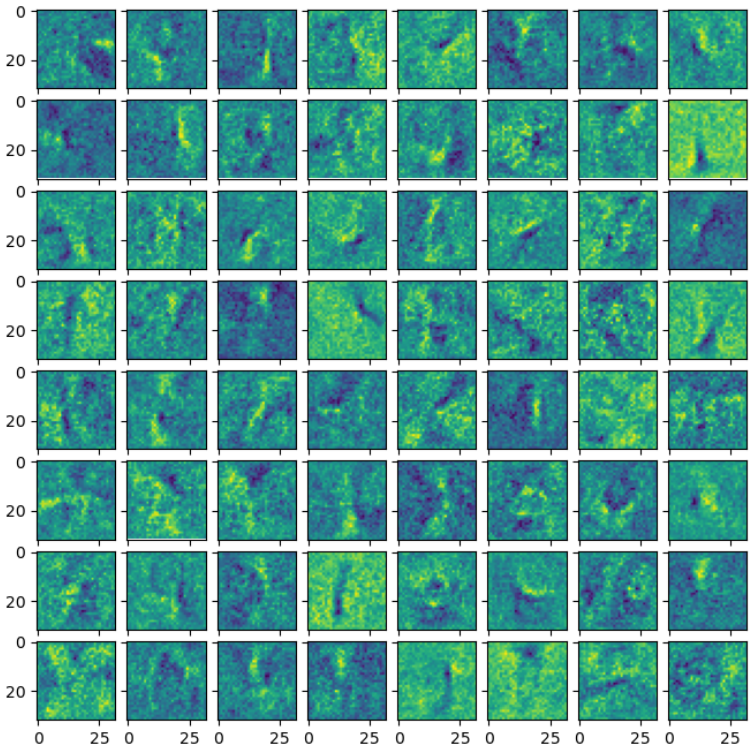
Best test accuracy (when learning rate is 0.001): 0.785
When learning rate is 0.01, the model seems to overfit.
The loss of validation starts to increase after around 10 epoches.
When learning rate is 0.0001, the model accuracy increases very slowly.
When the learning rate is 0.001,
we get the best accuracy over the same number of epoches.

3.1.3

Weights right after initialization



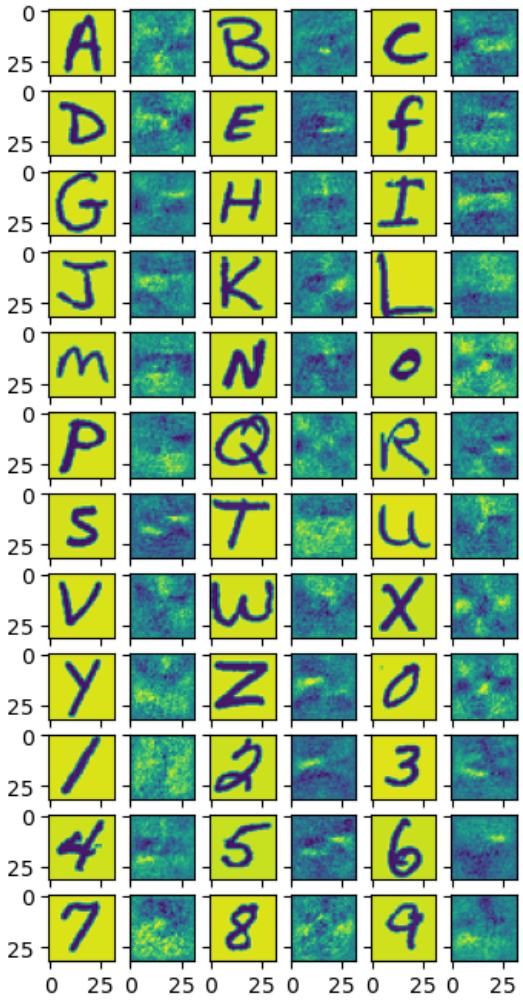
Weights after training



When we visualize the first layer weights right after initialization,
we could not see any patterns.
This makes sense since the weights are initilized randomly.

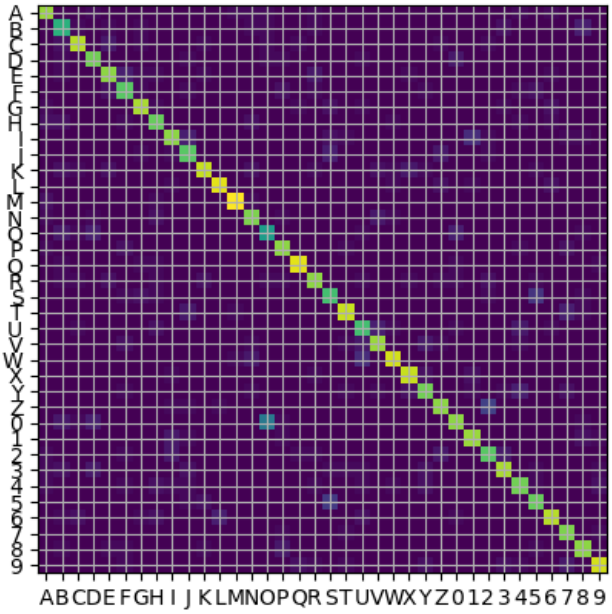
After training, the weights of the first layers become more intreptable.
We can tell some patterns, such as edges in different directions.

3.1.4



In the previous visualization,
we can only see some simple patterns like edges in different directions.
In this visualization,
we can see that the outputs resemble the shapes of actual letters or numbers.

3.1.5



We can clearly see that 0 and O, Y and 2, S and 5, are most commonly confused. This makes sense since they look very similar in shape.

4 Extract Text from Images

4.1

Assumption 1:
Different characters won't be connected.
Assumption 2:
All components of a single characters must be connected.

Example 1:

These two characters are considered as one character since they are connected.

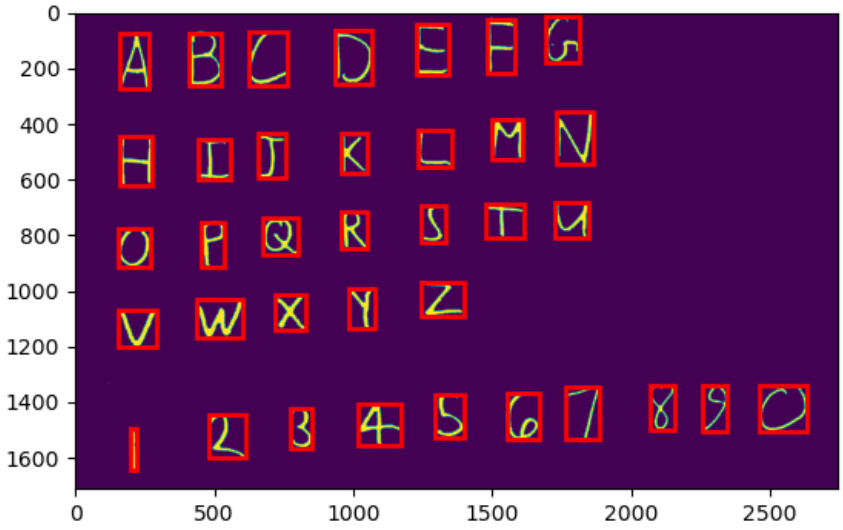
Example 2:

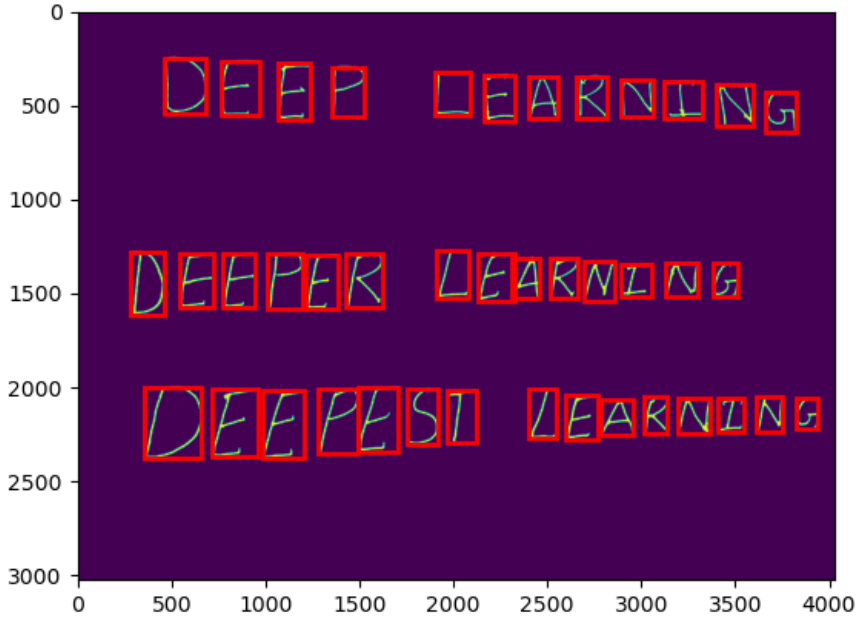
The horizontal line in the middle
is not considered as part of the letter E because it is disconnected.

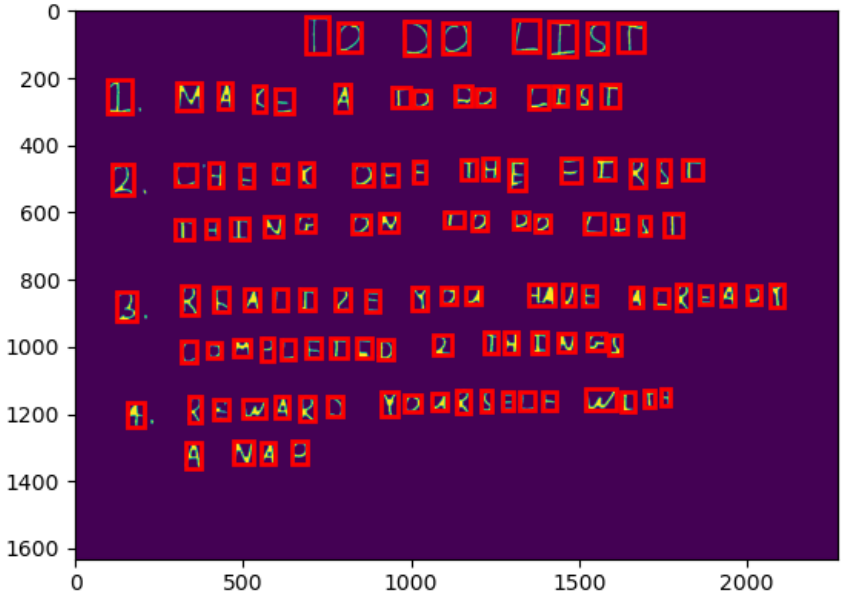
4.2

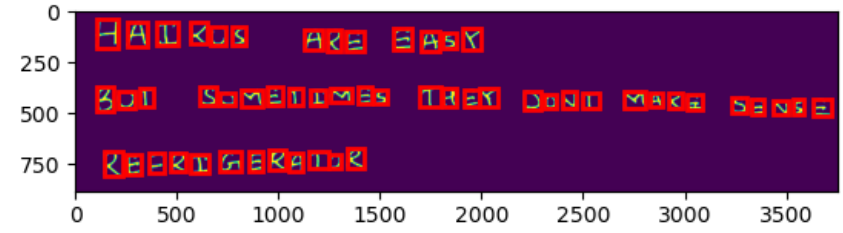
See the code for implementation.

4.3


0
200
400
600
800
1000
1200
1400
1600
0
500
1000
1500
2000
2500


0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500
4000





4.4

02_letters.jpg
2BCDEFG
HIJKLMN
0PQR5TU
VWXYZ
1Z3FS6789D

04_deep.jpg
DEEP LEAR3ING
DEBBZR LBAKNING
DEEA8S2 LEARQING

01_list.jpg
T0 D0 LIST
I MAKE A TD DD LIST
2 CHRCK 0FF 7HE FIRST
THING QN T0 D0 LIST
S R2ALIZE Y0U MVE ALR6ADT
E0MPLFTLD Z THINGS
F RFWARD Y0URSELF WITH
A NAP

03_haiku.jpg
HAIKUS ARE EASY
BUT SQMETIMES THEY DDNT MAK2 SENQE
REFRIGERAT0R

5 Image Compression with Autoencoders

5.1.1

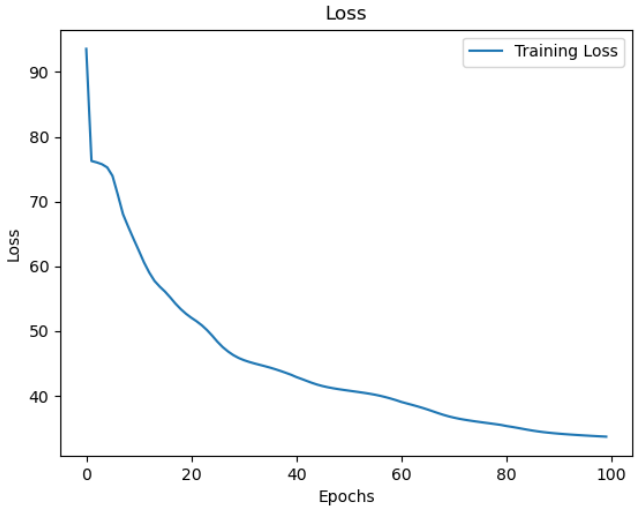
See the code for implementation.

5.1.2

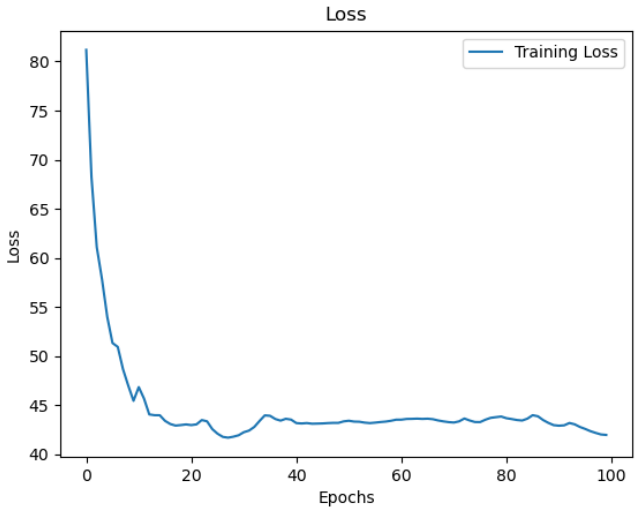
See the code for implementation.

5.2

Without momentum:

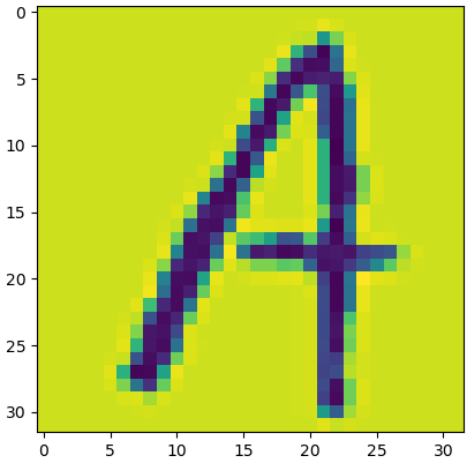


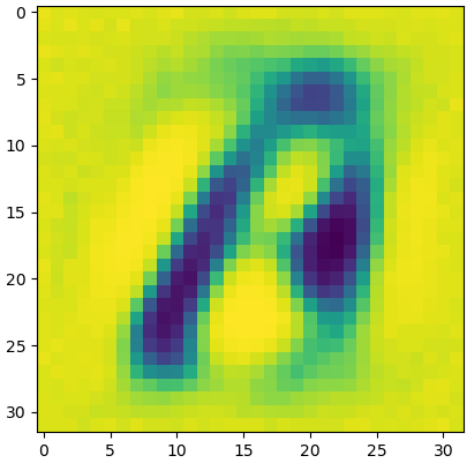
With momentum:

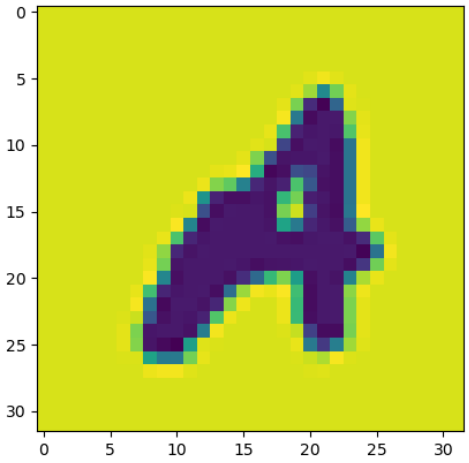


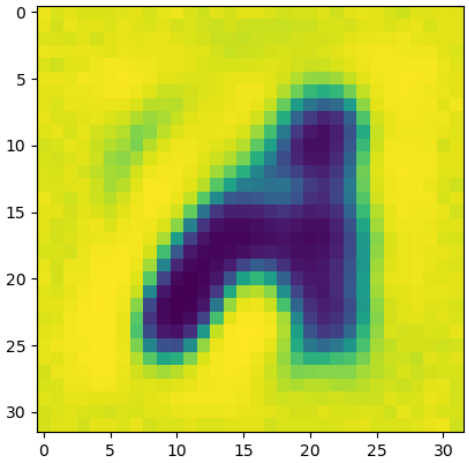
We can see the momentum version converges a lot faster.
As epoches increase, the loss values oscillate.

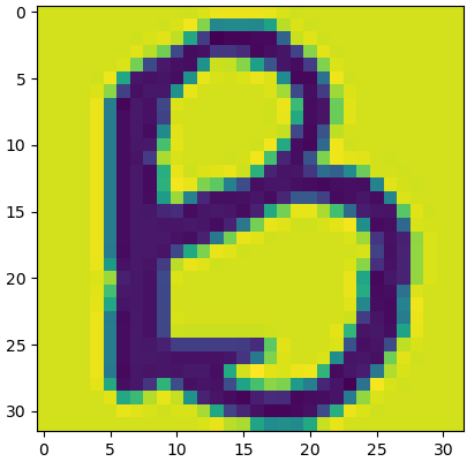
5.3.1

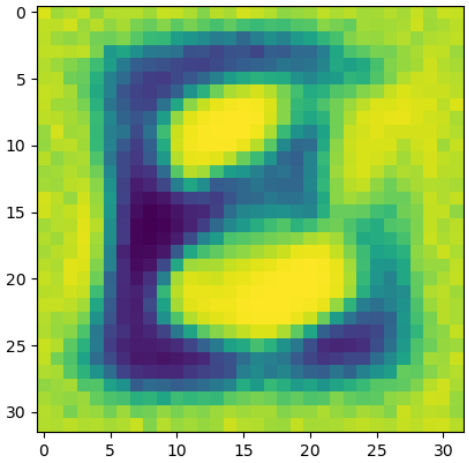


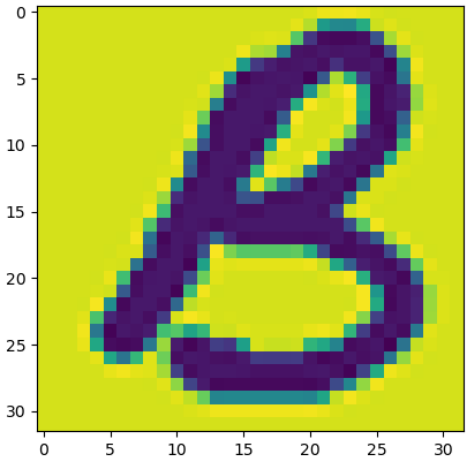


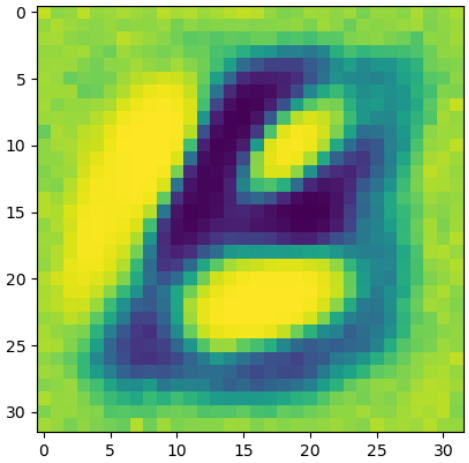

0
5
10
15
20
25
30
0
5
10
15
20
25
30

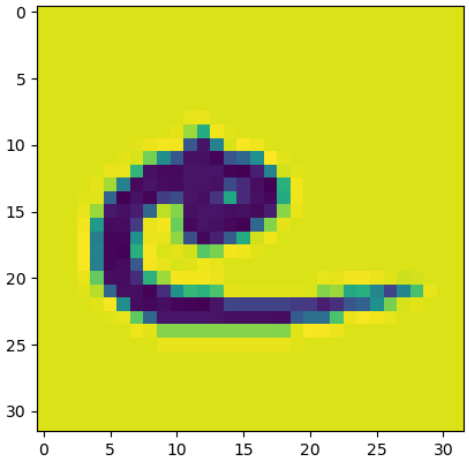

0
5
10
15
20
25
30
0
5
10
15
20
25
30

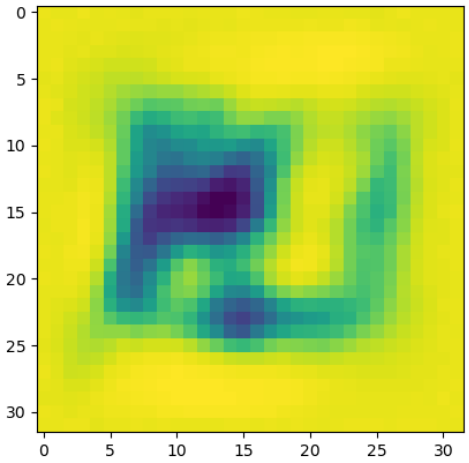

0
5
10
15
20
25
30
0
5
10
15
20
25
30

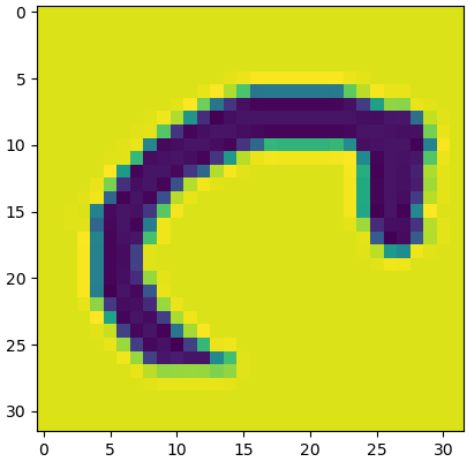

0
5
10
15
20
25
30
0
5
10
15
20
25
30

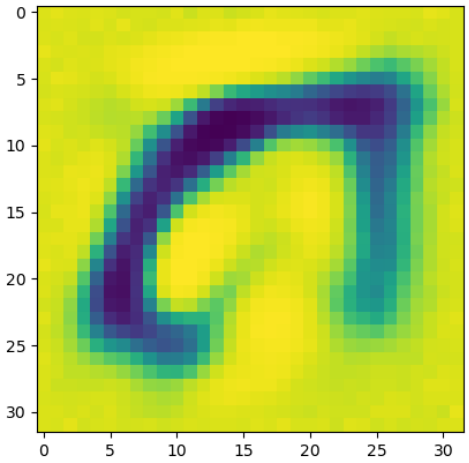

0
5
10
15
20
25
30
0
5
10
15
20
25
30

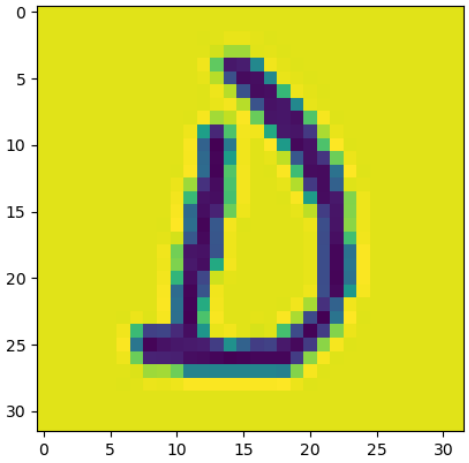

0
5
10
15
20
25
30
0
5
10
15
20
25
30

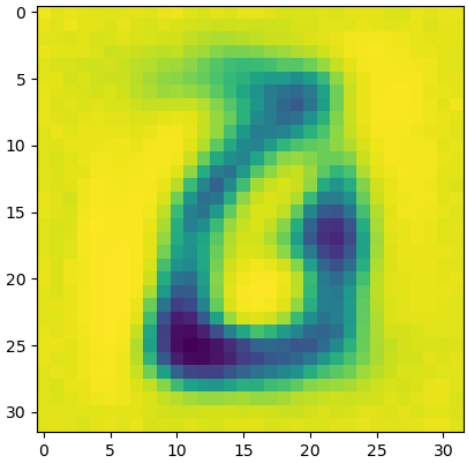

0
5
10
15
20
25
30
0
5
10
15
20
25
30

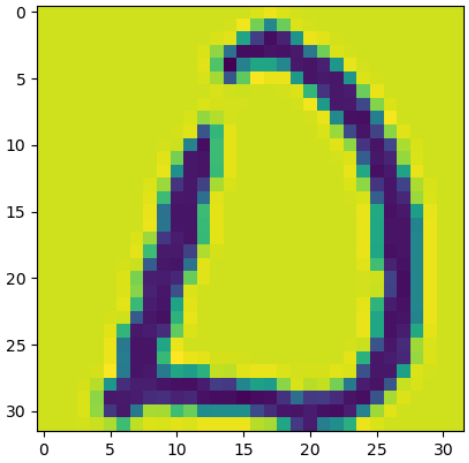

0
5
10
15
20
25
30
0
5
10
15
20
25
30

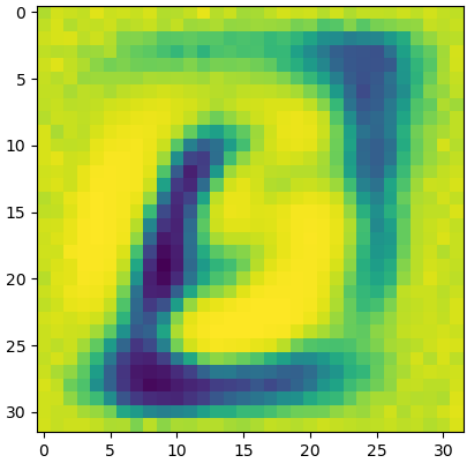

0
5
10
15
20
25
30
0
5
10
15
20
25
30

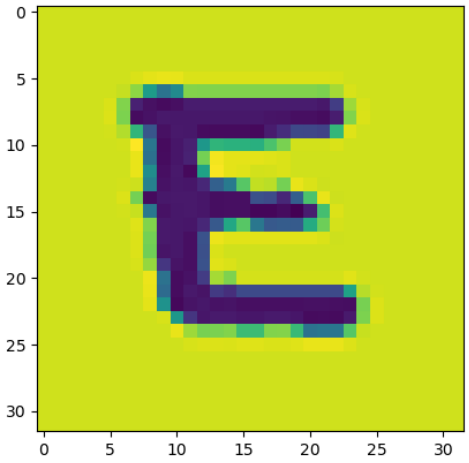

0
5
10
15
20
25
30
0
5
10
15
20
25
30

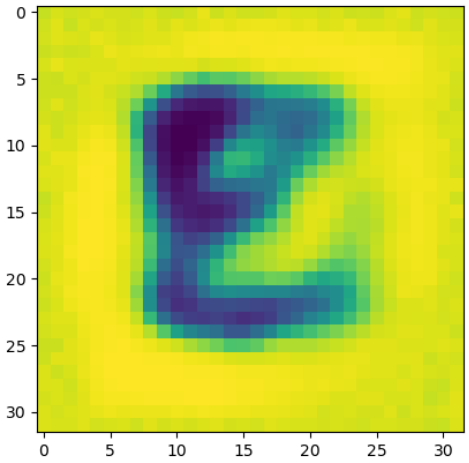

0
5
10
15
20
25
30
0
5
10
15
20
25
30

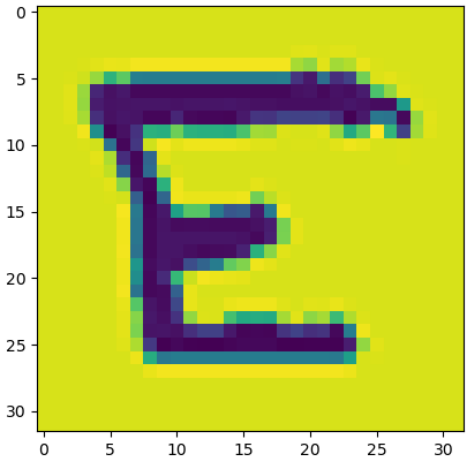

0
5
10
15
20
25
30
0
5
10
15
20
25
30

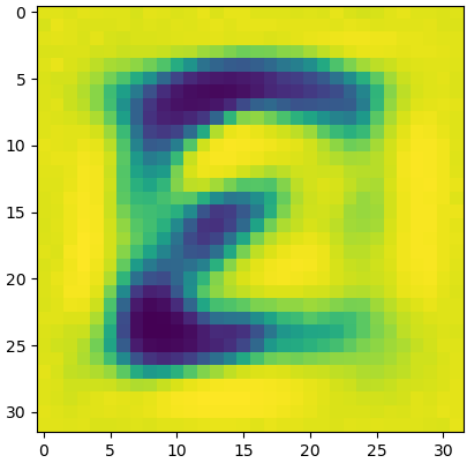

0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30





I think the reconstructed images lose some details compared to the originals, but the reconstructed images preserve the general shapes.
Note: I used the non-momentum version to generate these images.

5.3.2

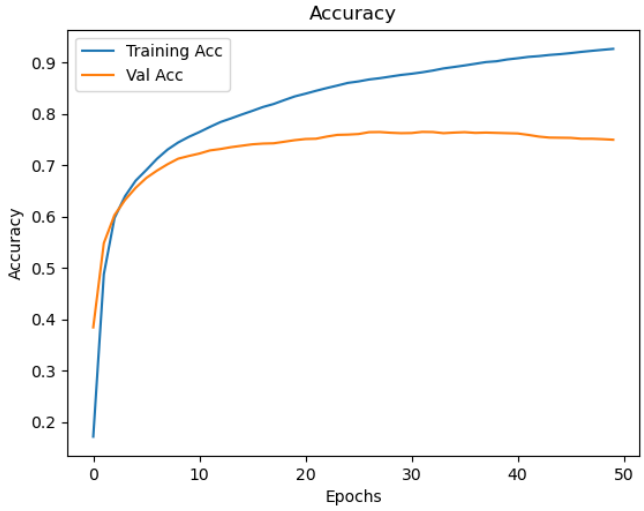
I did not do this part

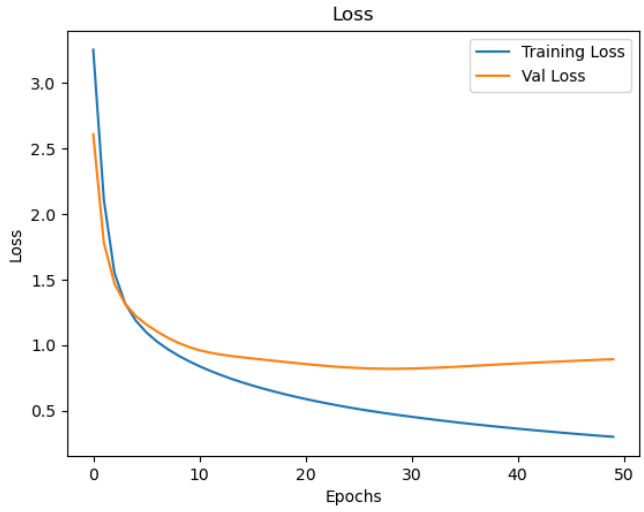
6 Comparing against PCA

I did not do this section

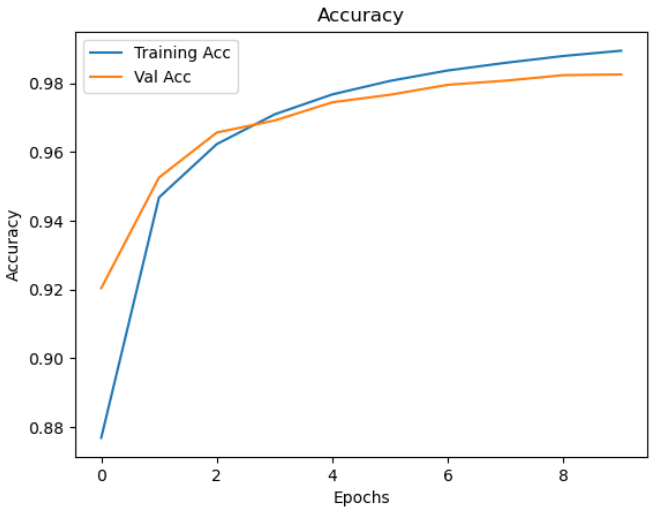
7 PyTorch

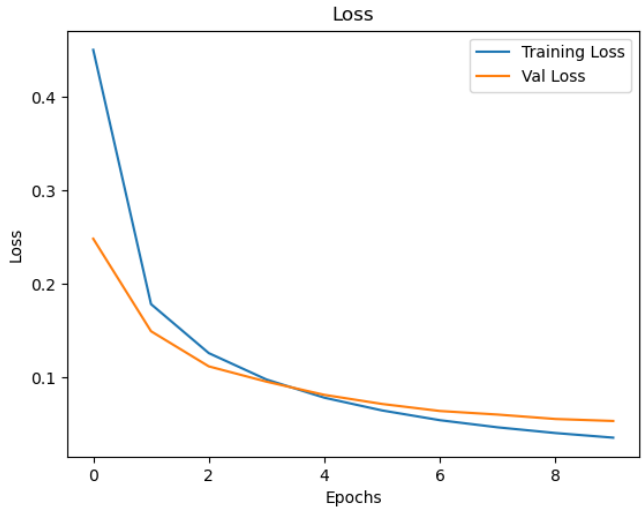
7.1.1



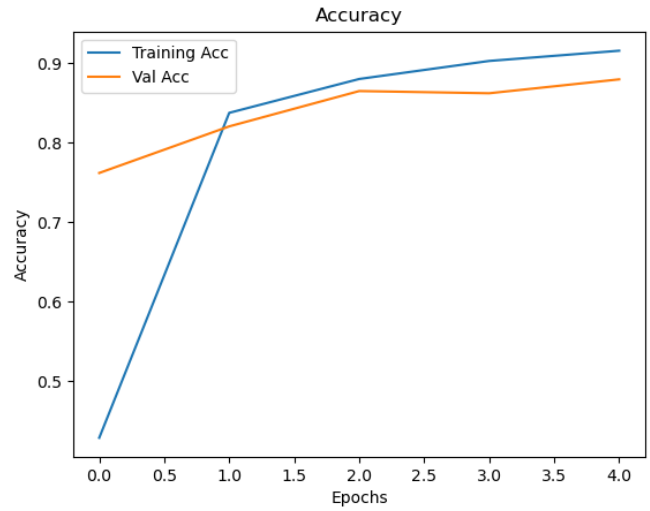


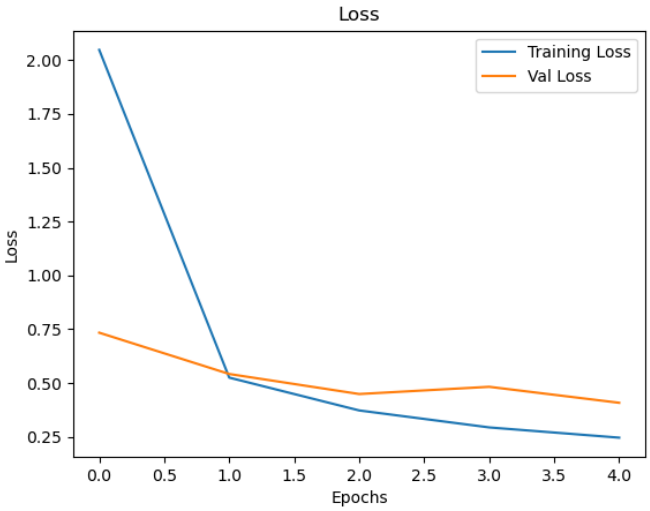
7.1.2


Accuracy
Training Acc
Val Acc
Accuracy
0.98
0.96
0.94
0.92
0.90
0.88
0
2
4
6
8
Epochs


Loss
Training Loss
Val Loss
Loss
0.4
0.3
0.2
0.1
0
2
4
6
8
Epochs

7.1.3


Accuracy
Training Acc
Val Acc
Accuracy
0.9
0.8
0.7
0.6
0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epochs


Loss
Training Loss
Val Loss
Loss
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epochs

7.1.4

My Result:

02_letters.jpg
H8CDEF6
HIJKLMN
0PQRSTY
VWXYZ
1234S678gQ
Accuracy: 0.7777777777777778

04_deep.jpg
DEEP LEARNING
DEERER hgARNING
DEERZSJ LEARNING
Accuracy: 0.8536585365853658

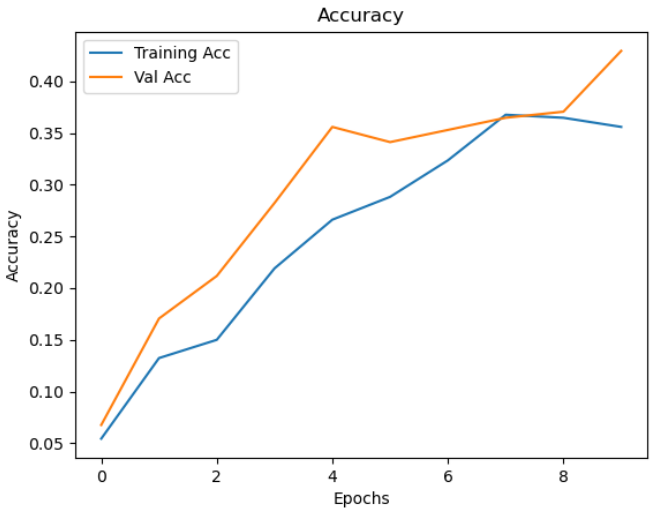
01_list.jpg
TQ DO LIST
1 MAKE A TO DO LIST
2 CHECK OFF THf FJ8ST
THING ON T0 DO LtST
3 RBALJ2E YOU MVE hLREADY
C0MPLET6D 2 THINGS
t REWARD YOURSELF WITH
A NAP
Accuracy: 0.8771929824561403
Note: For this example, my code misclassified 'Have' to be 'MVE' (missing one character),
to simplify my code for calculating accuracy,
I got rid of the 'H' character in my ground truth,
this won't affect the accuracy too much.

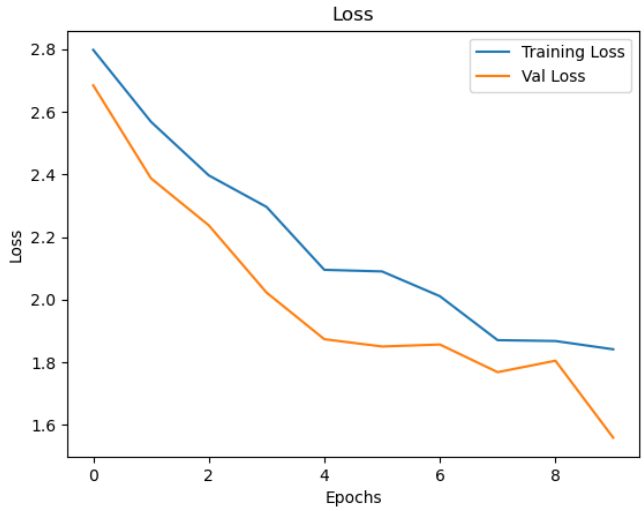
03_haiku.jpg
HAIKUS ARE EASr
8UT SOMETIMES THEr DONT MAKE SENSE
REfRIGERATOR
Accuracy: 0.9259259259259259

7.2.1

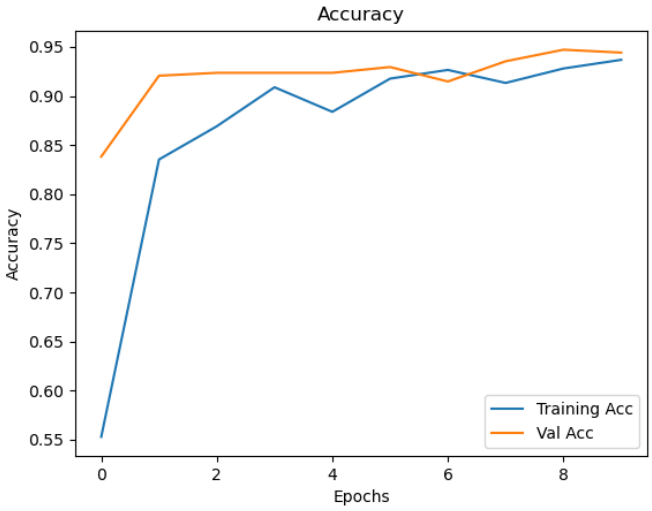
The fine tuned version receives much better performance.

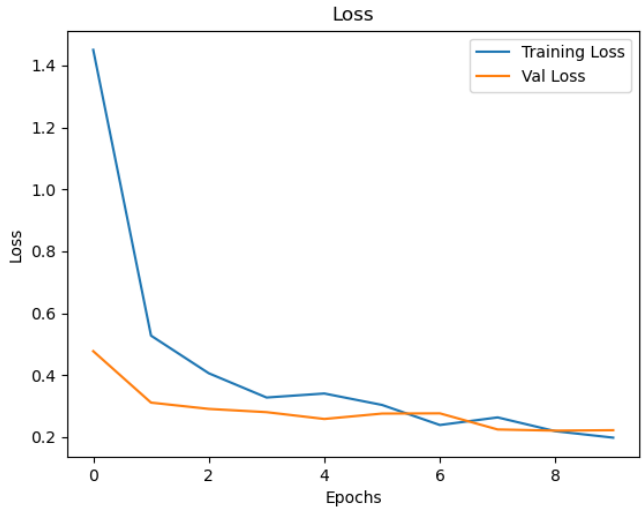
Custom:





Fine Tuned:


Accuracy
Accuracy
Epochs
Training Acc
Val Acc
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
2
4
6
8


Loss
Loss
Epochs
Training Loss
Val Loss
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8

8 Open Study Group Credit

I (Jiaqi Geng) have hosted two online study sessions with Qichen Fu.
Here are the proofs.