

16-720 Homework 5

Chendi Lin

November 28, 2018

Problem 1.1. Prove that softmax is invariant to translation, that is

$$\text{softmax}(x) = \text{softmax}(x + c)$$

Solution 1.1. For each index i in a vector x , we have

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (1)$$

$$\text{softmax}(x_i + c) = \frac{e^{x_i + c}}{\sum_j e^{x_j + c}} \quad (2)$$

$$= \frac{e^{x_i} e^c}{\sum_j e^{x_j} e^c} \quad (3)$$

$$= \frac{e^{x_i} e^c}{e^c \sum_j e^{x_j}} \quad (4)$$

$$= \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (5)$$

$$= \text{softmax}(x_i) \quad (6)$$

Thus, $\text{softmax}(x) = \text{softmax}(x + c)$.

When $c = -\max x_i$, $x + c$ is always less or equal to 0. This prevents the explosion of exponential so that e^{x_i} and $\sum_j e^{x_j}$ will not result in overflow.

Problem 1.2. Softmax can be written as a three step processes, with $s_i = e^{x_i}$, $S = \sum s_i$, and $\text{softmax}(x_i) = \frac{1}{S} s_i$.

Solution 1.2. Since $\text{softmax}(x_i) = \frac{1}{S} s_i = \frac{s_i}{\sum s_i}$, and $s_i = e^{x_i}$, each element in $\text{softmax}(x)$ should be in the range of $(0, 1)$. The sum of all elements is $\sum \frac{s_i}{\sum s_i} = 1$.

One could say that "softmax takes an arbitrary real valued vector x and turns it into a probability distribution".

$s_i = e^{x_i}$ calculates the outcome frequency x_i in the exponential form. $S = \sum s_i$ calculates the total outcome frequency. $\frac{1}{S} s_i$ normalized the frequency of each x_i and outputs the probability.

Problem 1.3. Show that multi-layer neural networks without a non-linear activation function are equivalent to linear regression.

Solution 1.3. When going through a layer of neural network without a non-linear activation function, it is the same as apply a linear function to x such that

$$x_{i+1} = W_i x_i + b_i \quad (7)$$

When applying multi-layer neural networks, we have

$$y = W_n x_n + b_n \quad (8)$$

$$= W_n (W_{n-1} x_{n-1} + b_{n-1}) + b_n \quad (9)$$

$$= W_n W_{n-1} x_{n-1} + W_n b_{n-1} + b_n \quad (10)$$

$$= W' x_{n-1} + b' \quad (11)$$

$$= W'(W_{n-2} x_{n-2} + b_{n-2}) + b' \quad (12)$$

$$\dots\dots \quad (13)$$

$$= Wx + b \quad (14)$$

which is the same as solving a linear regression problem

Problem 1.4. Given the sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$, derive the gradient of the sigmoid function and show that it can be written as a function of $\sigma(x)$

Solution 1.4.

$$\nabla(\sigma(x)) = \frac{d\sigma(x)}{dx} \quad (15)$$

$$= \frac{e^{-x}}{(1 + e^{-x})^2} \quad (16)$$

$$= \frac{1}{1 + e^{-x}} \frac{e^{-x}}{1 + e^{-x}} \quad (17)$$

$$= \frac{1}{1 + e^{-x}} \left(1 - \frac{1}{1 + e^{-x}}\right) \quad (18)$$

$$= \sigma(x)(1 - \sigma(x)) \quad (19)$$

Problem 1.5.

$$J = W^T x + b \quad (20)$$

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial W} \quad (21)$$

$$= x \delta^T \quad (22)$$

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial x} \quad (23)$$

$$= W \delta \quad (24)$$

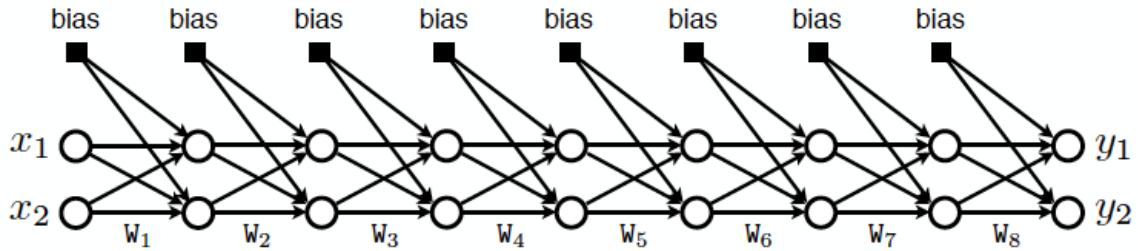
$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial b} \quad (25)$$

$$= \delta \quad (26)$$

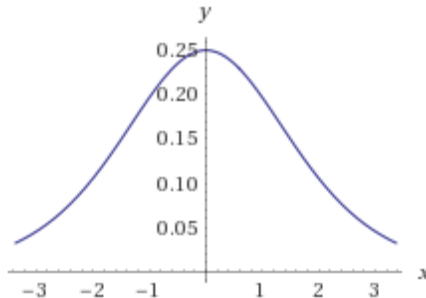
$$(27)$$

Problem 1.6. When the neural network applies the elementwise activation function (such as sigmoid), the gradient of the activation function scales the backpropagation update. This is directly from the chain rule, $\frac{d}{dx}f(g(x)) = f'(g(x))g'(x)$.

Solution 1.6. 1. The multi-layer neural networks flow chart attached here was borrowed from lecture notes.



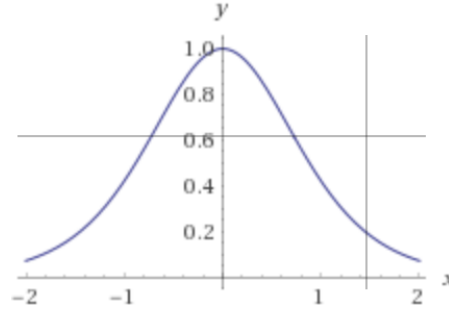
The plot of the derivative of sigmoid is shown here



From the figure we can see that, the maximum of the derivative of sigmoid function is 0.25. When the activation function was applied to the layers, and the derivative was taken, the entries in x will drop quickly. If it is used for many layers, it might lead to a “vanishing gradient” problem.

2. The output range of sigmoid is $(0, 1)$. Since $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} = 1 - \frac{2e^{-2x}}{1+e^{-2x}}$, the output range of $\tanh(x)$ is $(-1, 1)$. Both functions have the similar shapes and are differentiable. $\tanh(x)$ is preferred because when x is positive, $\tanh(x)$ will be positive, and when x is negative, $\tanh(x)$ will be negative.

3.



The plot above shows the derivative of $\tanh(x)$. As we can see here, the range is from 0 to 1. So when we take derivatives in several layers, it is less likely to see the vanishing gradient problem.

4. Since

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (28)$$

$$2\sigma(x) - 1 = \frac{2}{1 + e^{-x}} - \frac{1 + e^{-x}}{1 + e^{-x}} \quad (29)$$

$$= \frac{1 - e^{-x}}{1 + e^{-x}} \quad (30)$$

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \quad (31)$$

we have

$$\tanh(x) = 2\sigma(2x) - 1 \quad (32)$$

Problem 2.1.1. Why is it not a good idea to initialize a network with all zeros? If you imagine that every layer has weights and biases, what can a zero-initialized network output after training?

Solution 2.1.1. If the network is initialized with all zeros, all the outputs generated from the network will be zero, and the generated probability vector will include all the same entries. After backwards propagation and gradient descent, the weights will be updated at the same time, so the layers will still do the same computations.

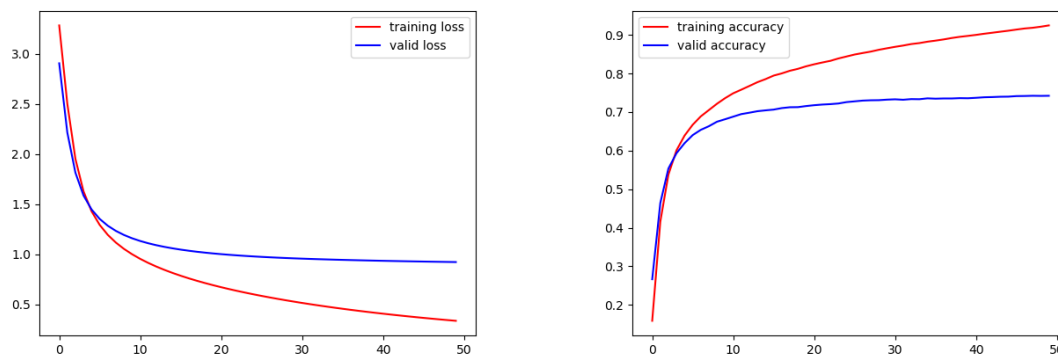
Problem 2.1.3. Why do we initialize with random numbers? Why do we scale the initialization depending on layer size?

Solution 2.1.3. Initializing the network randomly is to prevent the symmetry of the matrix, so that we can avoid the same computations from the layers.

Scaling the initialization based on the layer size can help keep the variance around the desired values when doing forward propagation and backward propagation in the network.

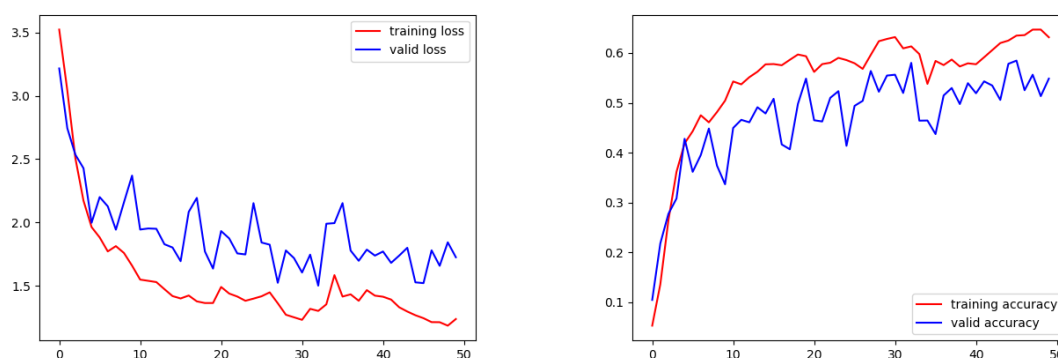
Problem 3.1.1. Train a network from scratch. Use a single hidden layer with 64 hidden units, and train for at least 30 epochs

Solution 3.1.1. With default settings, $batch_size = 64$, $learning_rate = 3e - 3$, the result was the best. Since when plotting total loss, the loss of valid data will be insignificant comparing to the loss of the training data in the same figure, we average the total loss by the number of samples. The plots are shown below.



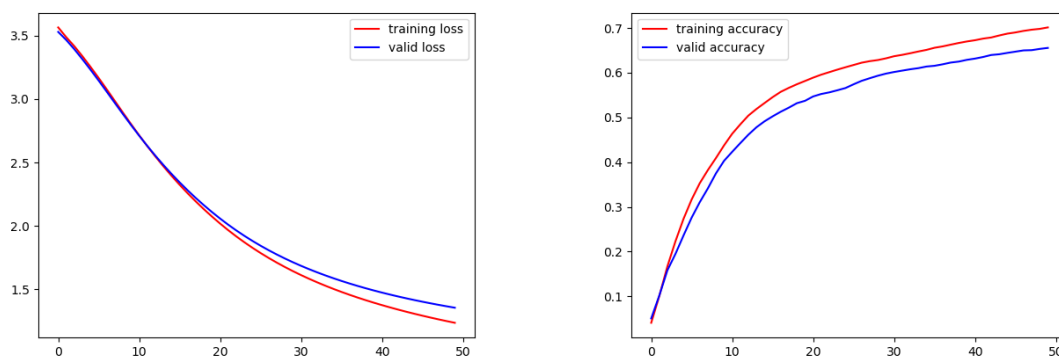
Problem 3.1.2 . Use your modified training script to train three networks, one with your best learning rate, one with 10 times that learning rate and one with one tenth that learning rate. Include all 4 plots in your writeup. Comment on how the learning rates affect the training, and report the final accuracy of the best network on the test set.

Solution 3.1.2. With all the same settings, and $learning_rate = 3e - 2$, results were like



The losses were higher than the previous case while the accuracy was lower. The curves were also more bumpy because the step size was too large.

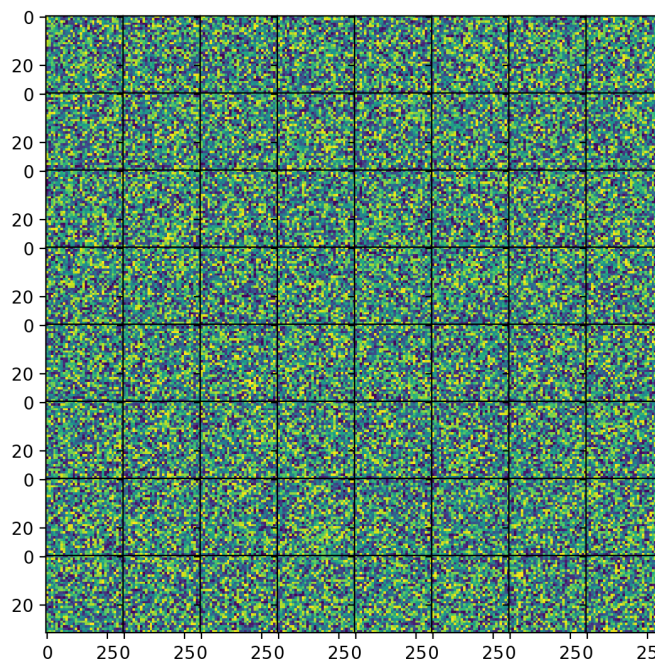
When $learning_rate = 3e - 4$, results were like



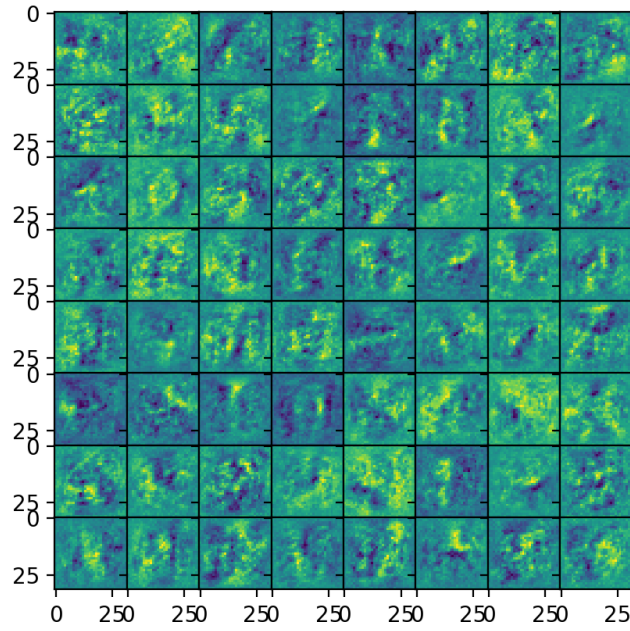
The curves were as smooth as those shown in Q3.1.1, but because the step size was too small, it did not converge to the optimum within the maximum number of iterations (50) that we set.

Problem 3.1.3 . Visualize the first layer weights that your network learned

Solution 3.1.3. The initialized first layer weights were visualized as



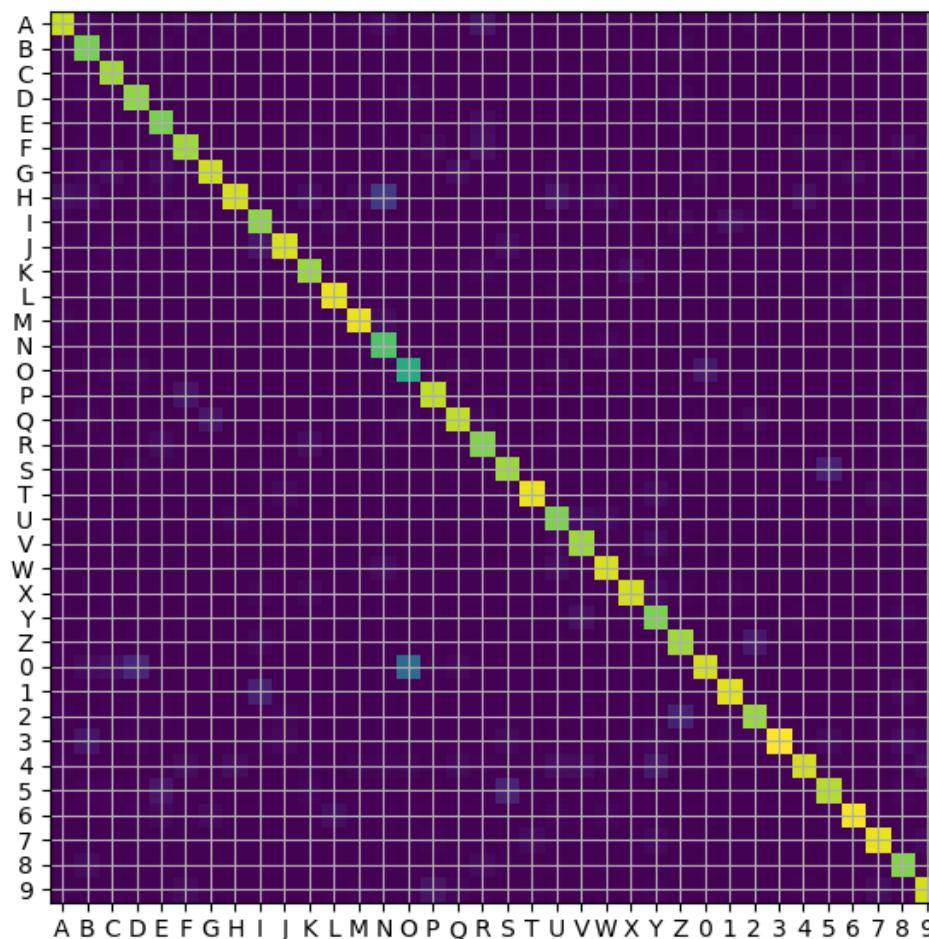
The learned first layer weights were visualized as



Comparing the two figures, it is clear that, since we initialized the layers with random uniform distribution, at first, the visualization of the weights in the first layer was just random noisy points. while after learned in 50 epochs, some more clear patterns, with different shapes can be presented.

Problem 3.1.4. Visualize the confusion matrix for your best model. Comment on the top few pairs of classes that are most commonly confused.

Solution 3.1.4. Presented below is the visualization of the confusion matrix for the best model, with $learning_rate = 3e - 3$.



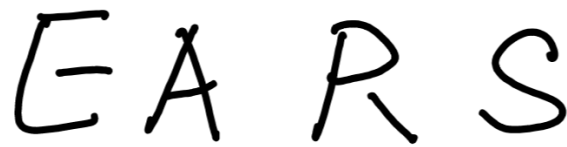
The brighter the grid was, the larger number it was in that grid. Since the main diagonal was the heaviest, we could conclude that the result was pretty good.

From the figure we can see that, the most commonly confused pairs were: ‘O’ and ‘0’, ‘H’ and ‘N’, ‘D’ and ‘0’, which were often misjudged manually also.

Problem 4.1. Q4.1 Theory [2 points] The method outlined above is pretty simplistic, and makes several assumptions. What are two big assumptions that the sample method makes.

Solution 4.1. We assume that:

1. Every letter is fully connected. Since we extract the letters by their connections, if the parts in one single letter were separated, it would be hard for the methods to work. An example is shown below, in which, 'E' and 'R' were badly written because they were not nicely connected.



The image shows the word "EARS" written in a casual, handwritten style. The letters are not connected to each other. Specifically, the 'E' and 'R' are written as separate, isolated shapes, which illustrates the first assumption that every letter must be fully connected for the method to work.

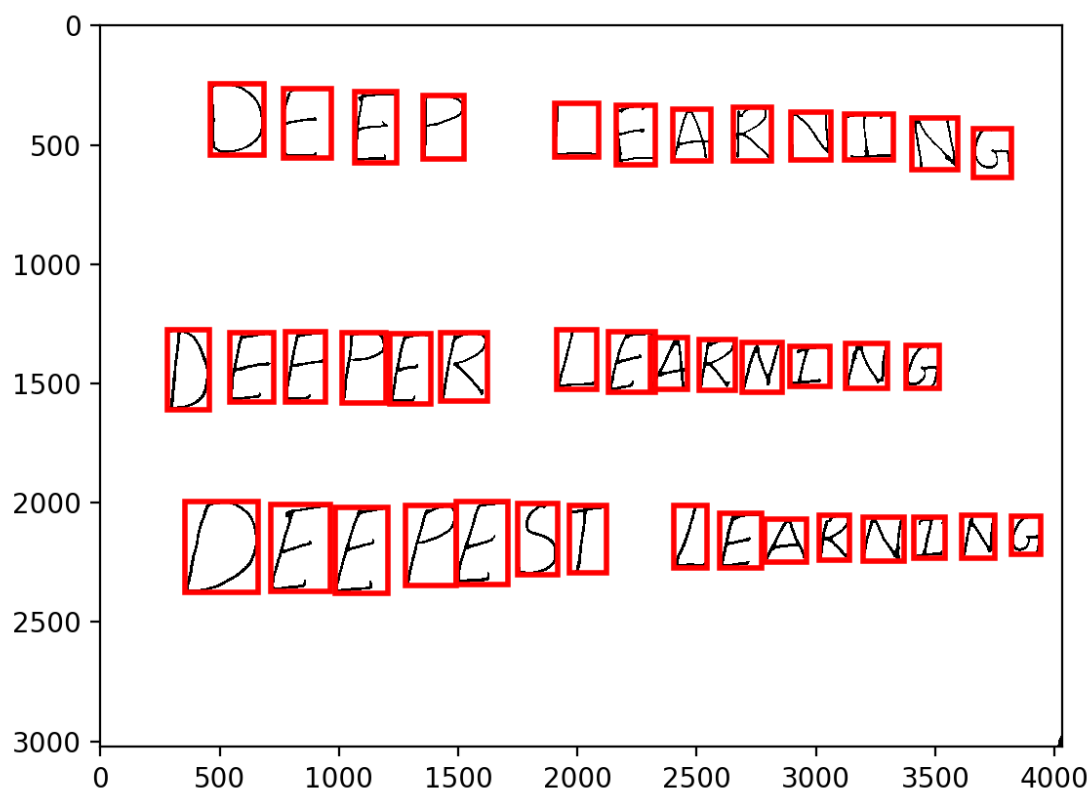
2. Two different letters cannot be connected. Same reason as the previous assumption that, since we extract the letters by their connections, if the parts in two letters were overlapped, the method cannot separated them. An example is shown below.

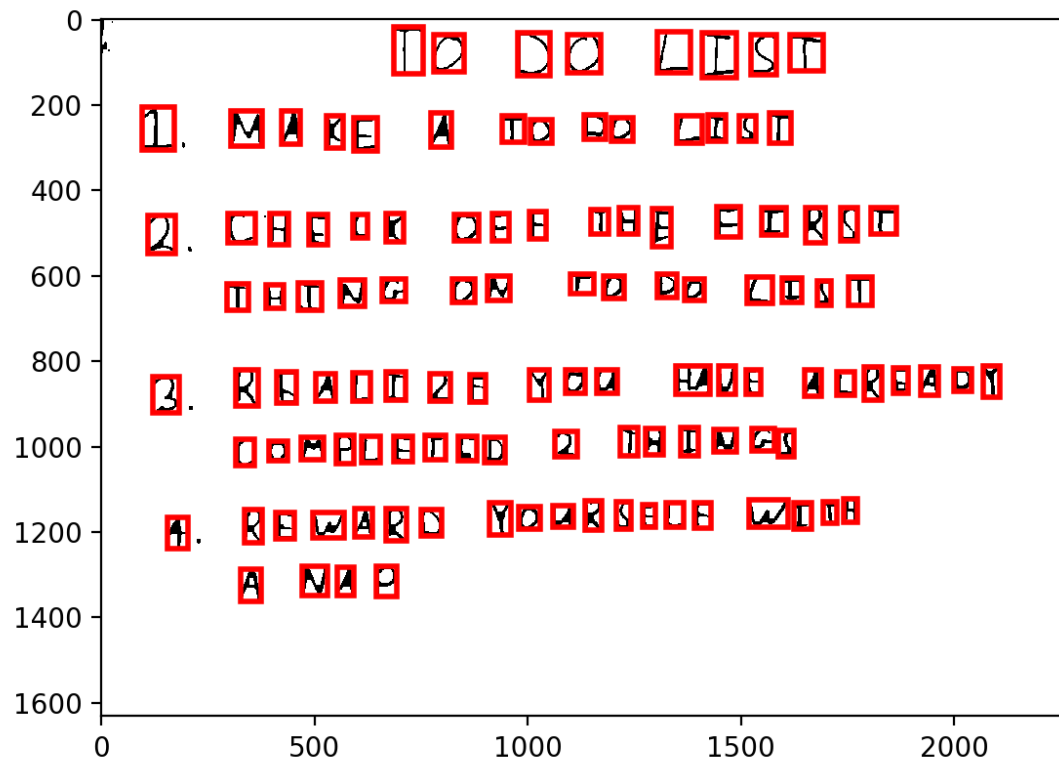


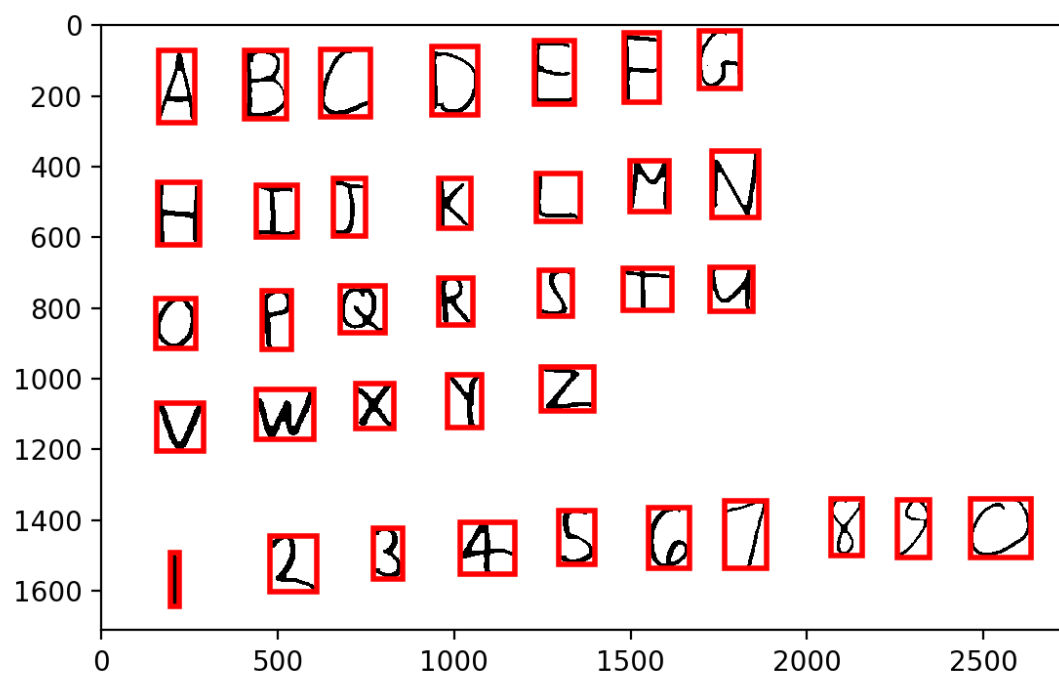
The image shows the phrase "TAKE IT EASY" written in a casual, handwritten style. The letters are not connected to each other. Specifically, the 'E' in "EASY" is written such that its top and bottom strokes are connected to the 'A' on its left, illustrating the second assumption that two different letters cannot be connected for the method to work.

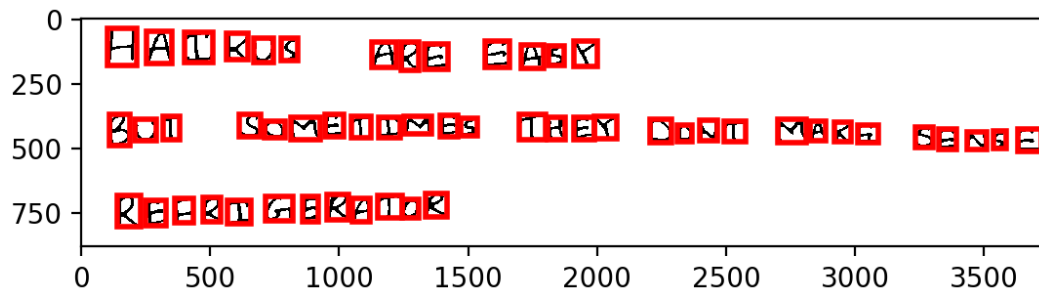
Problem 4.3. Find letters

Solution 4.3. All the letters were found and circled by the red rectangles.









Problem 4.4. Include the extracted text in your writeup.

Solution 4.4. By the trained nn in Q3.2.1 and adequate letter separations, the results were:

1. “deep.jpg”:

DEEP LEARMIXU

DE8FEF LEAR4ING

DEEPEST LEARNING

2. “to_do_list.jpg”

T0 J0 LIST

I MA KE A T0 D0 LIST

2 CH2 CK 0F F YHE FIKST

THING 0N T0 D0 LIST

3 RFALI Z E Y0U HVE ALR2ADY

COMPLFTLD 2 YHINGS

F RFWARD Y0URSELFWITH

A NAF

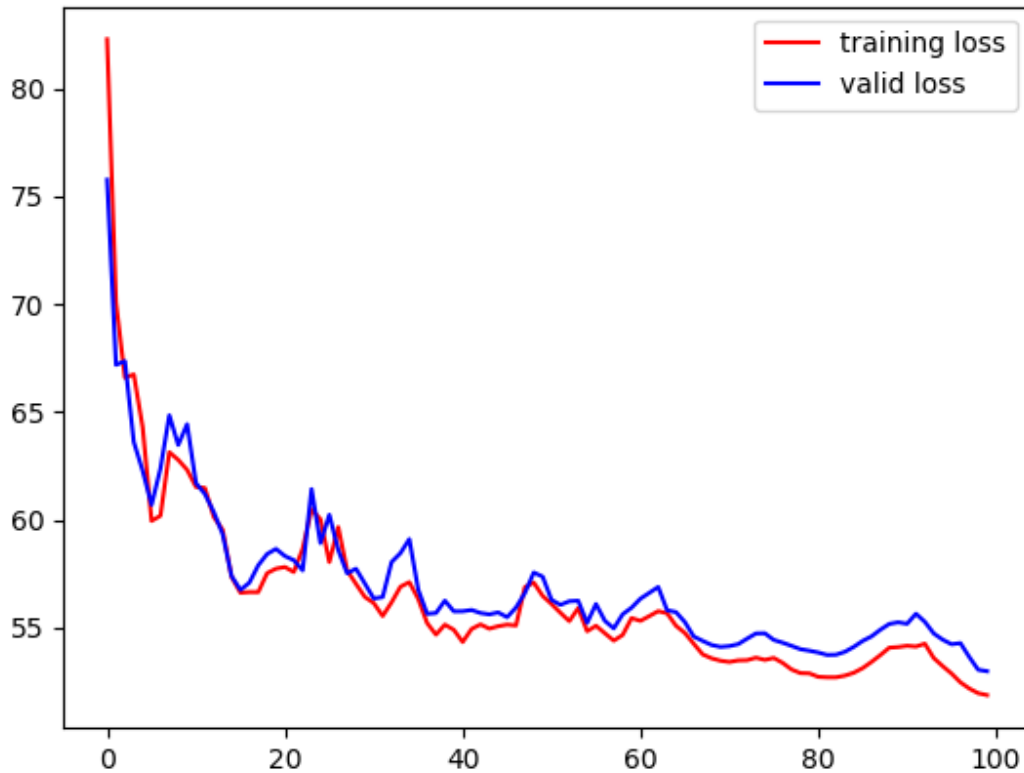
3. "letters.jpg":
A B C D E F G
H Z J K L M N
0 P R R S T 4
V W X Y Z
1 X 3 F S 6 7 X 70

4. "haiku.jpg":
HAIKOS ARR EASX
BOT S4MRT2MRF TRFY DF4T MAKR FR4FR
REFRXARRAT0R

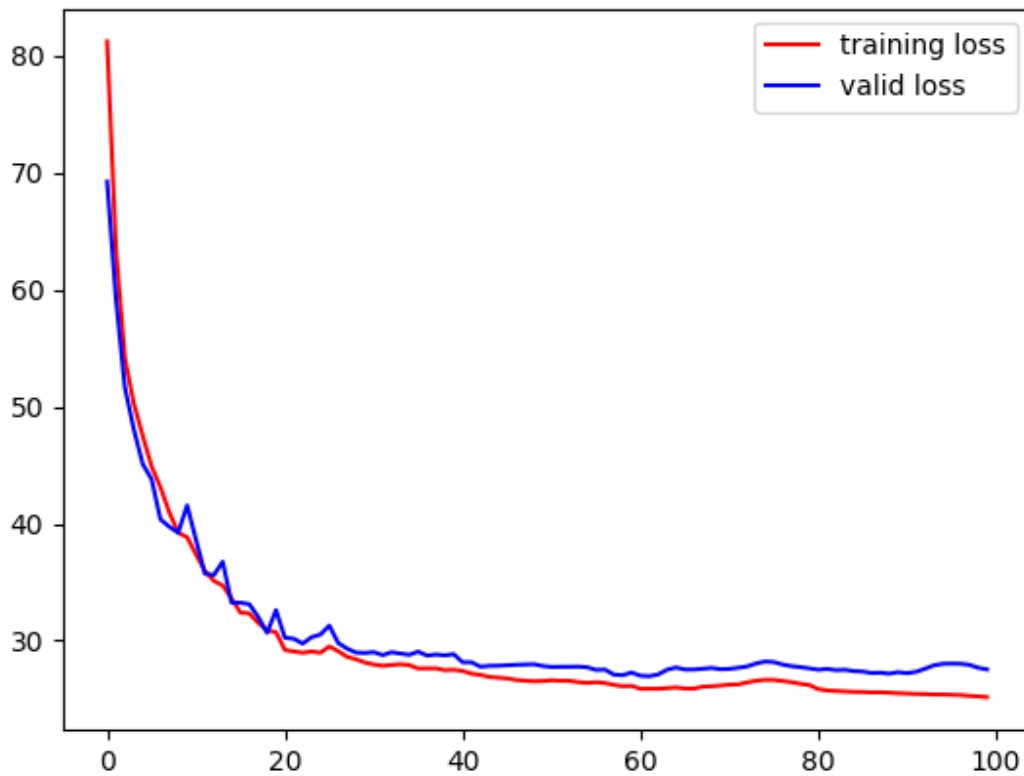
We can see that, overall, the results were pretty good, except for "to_do_list.jpg". It was mainly because that, the letters were too slim in that picture. Thus, after dilation and resizing, it became harder for the network to recognize the letters. For "to_do_list.jpg", different padding and dilation were applied.

Problem 5.2. Using the provided default settings, train the network for 100 epochs. What do you observe in the plotted training loss curve as it progresses?

Solution 5.2. With the default settings, picking the batch size as 36, and learning rate as $3e - 5$, the average loss over all the training samples and valid samples were plotted as below



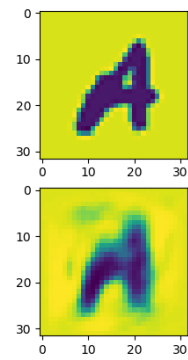
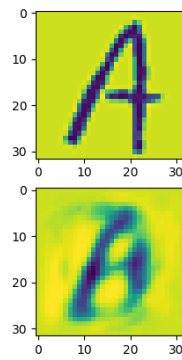
Overall, the loss was decreased in each epoch, with bumpy behaviors. If we changed the hidden layer size to be 64, we have



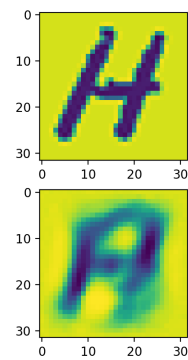
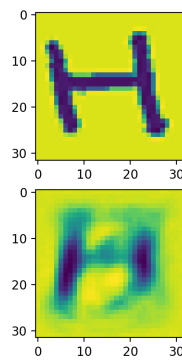
Here, the curves were smoother, and the losses were lower for both the training data and valid data.

Problem 5.3.1. Now let's evaluate how well the autoencoder has been trained. Select 5 classes from the total 36 classes in your dataset and for each selected class include in your report 2 validation images and their reconstruction. What differences do you observe that exist in the reconstructed validation images, compared to the original ones?

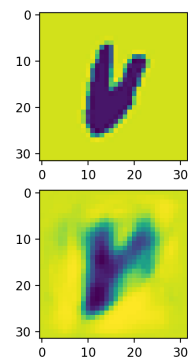
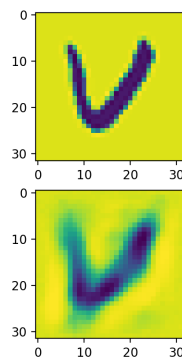
Solution 5.3.1. class “A”:



class “H”:



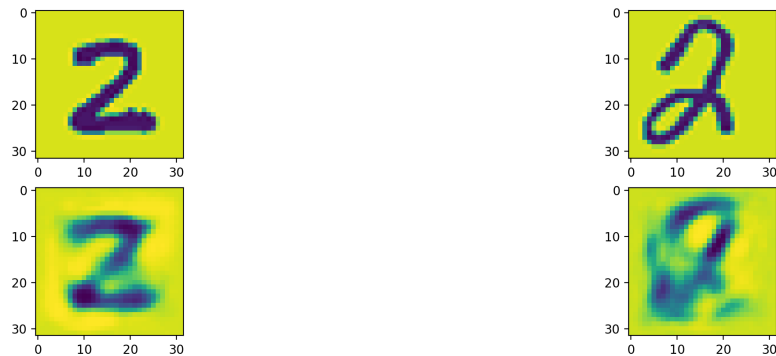
class “v”:



class “0”:



class “2”:



Compared to the original images, the basic shapes of the letters and digits have been reconstructed. However, it is impossible to fully represent the originally images with Autoencoders, so the reconstructed valid images looked more blurred and more interpolations existed between the edges of the letters and the background.

Problem 5.3.2. PSNR

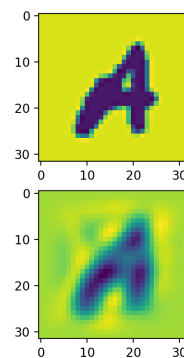
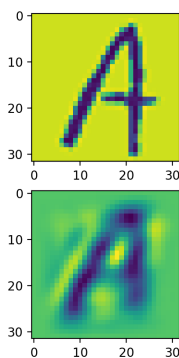
Solution 5.3.2. With the default settings, picking the batch size as 36, and learning rate as $3e - 5$, the PSNR from the sutoencoder across all validation images was 13.16. However, if we changed the hidden layer size to be 64, the PSNR was 16.63324331619196.

Problem 6.1. What is the size of your projection matrix? What is its rank?

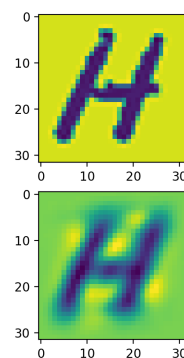
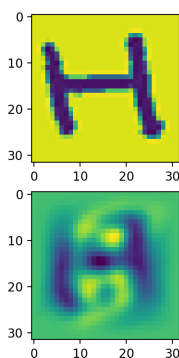
Solution 6.1. Since originally our data was a flattened vector from 32×32 images, it has 1024 elements. To extract the first 32 elements only, we need our projection matrix to be 1024×32 . its rank is 32.

Problem 6.2. Use the classes you selected in Q5.3.1, and for each of these 5 classes, include in your report 2 test images and their reconstruction. You may use test labels to help you find the corresponding classes. What differences do you observe that exist in the reconstructed test images, compared to the original ones? How do they compare to the ones reconstructed from your autoencoder?

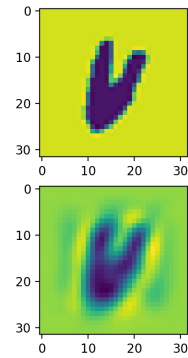
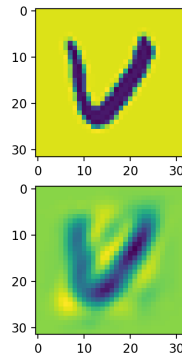
Solution 6.2. For comparison, the same images were used as the ones in Problem 5.3.1.
class “A”:



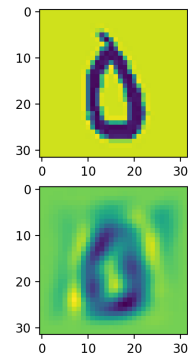
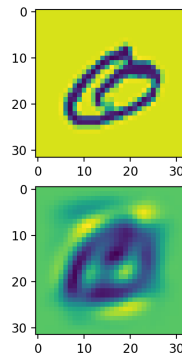
class “H”:



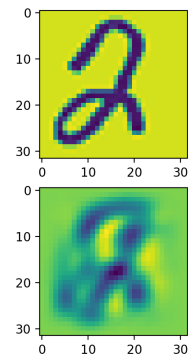
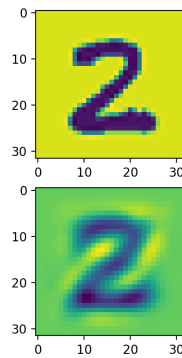
class “v”:



class “0”:



class “2”:



The shapes of the letters and digits were nicely reconstructed. The reconstruction images were still blur comparing to the original ones, but they looked more clear and sharp comparing to the results from autoencoder. The values of the background were very different from the original ones, but that is because by PCA, we threw away the useless information.

Problem 6.3. Report the average PSNR you get from PCA. Is it better than your autoencoder? Why?

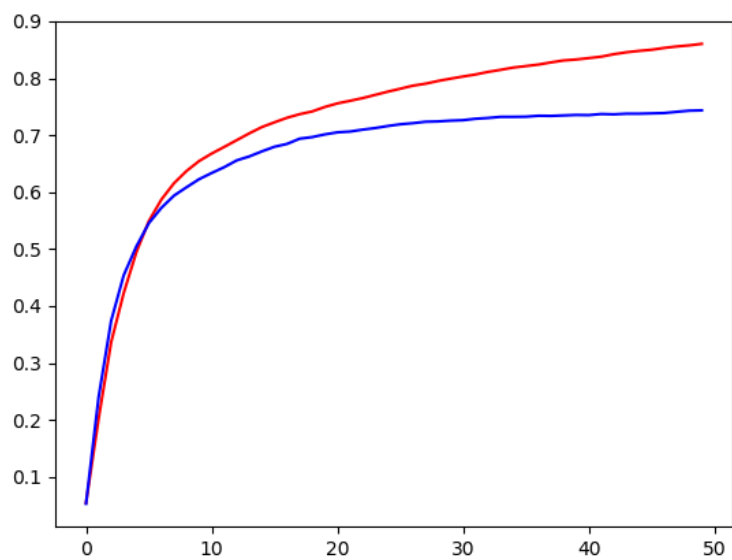
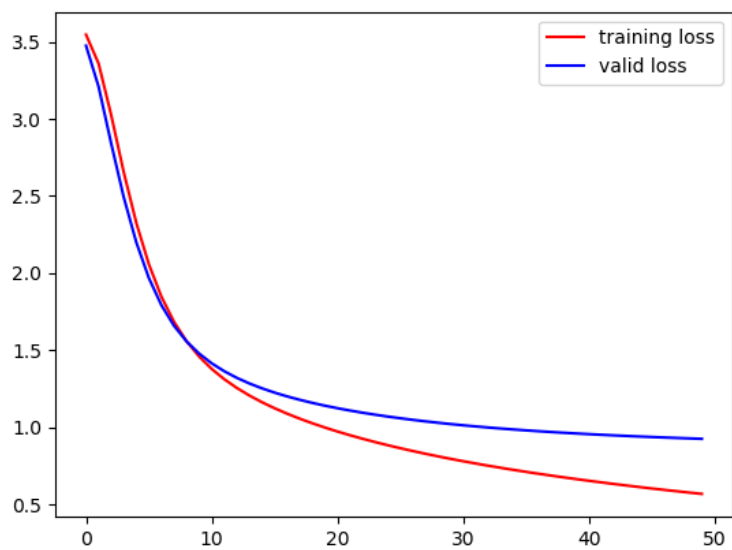
Solution 6.3. The average PSNR from PCA was 16.283913993444422. Since the higher PSNR is, the better the performance is, PCA outperformed autoencoder with default settings, while was slightly beaten by autoencoder when hidden layer size was 64.

Usually autoencoder should perform better than PCA because it includes more parameters and the non-linearity provides more information of the images, while PCA has only one linear operator in total. However, in this specific case, PCA performed better than autoencoder with default settings because the images contained a lot of redundant information such as the background. Throwing them away as in PCA would not destroy the necessary information to distinguish the letters and digits.

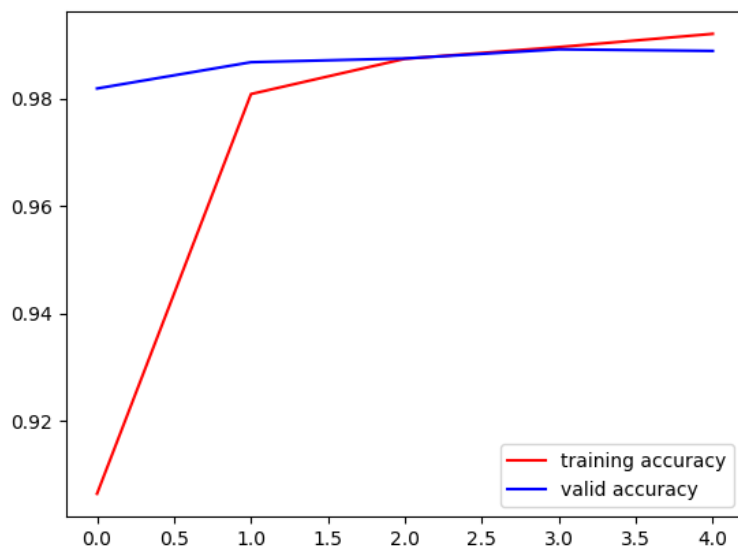
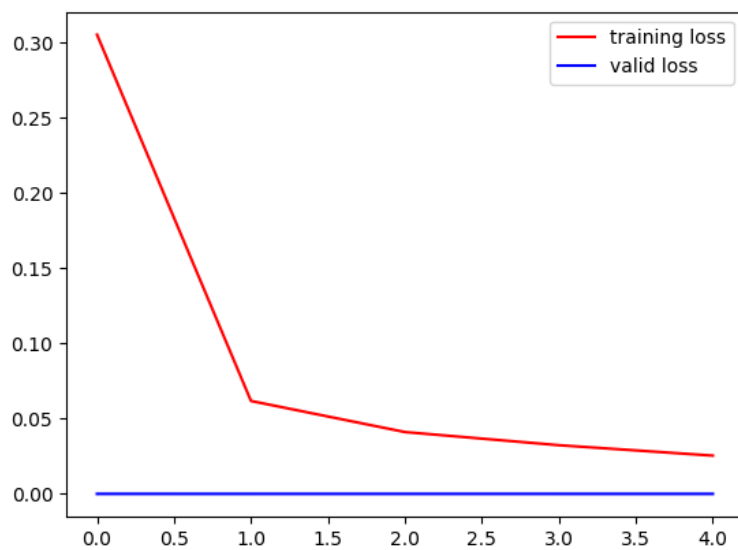
Problem 6.4. Count the number of learned parameters for both your autoencoder and the PCA model. How many parameters do the respective approaches learn? Why is there such a difference in terms of performance?

Solution 6.4. For autoencoder, the number of parameters was $1024 * 32 + 32 + 32 * 32 + 32 + 32 * 32 + 32 + 32 * 1024 + 1024 = 68704$. For PCA, the number of parameters was $1024 * 32 = 32768$. Since autoencoder learns more parameters and includes more information of the images, it should perform better than PCA. However, as we mentioned in the previous question, in this specific case, PCA performed better than autoencoder with default settings because the images contained a lot of redundant information such as the background.

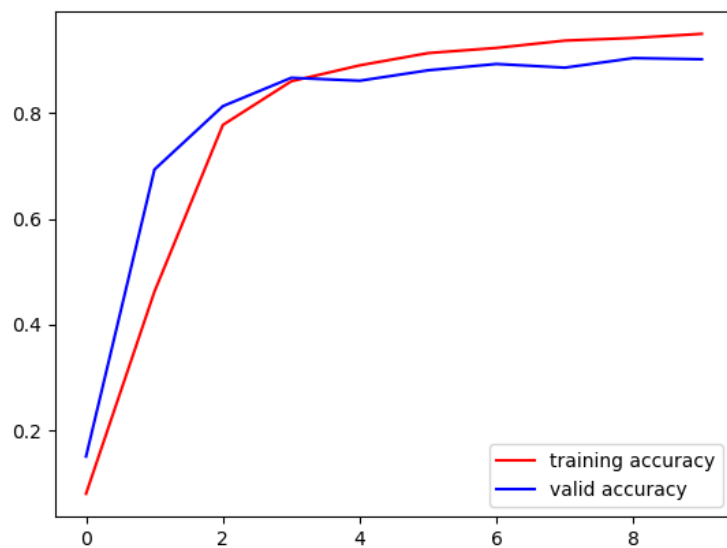
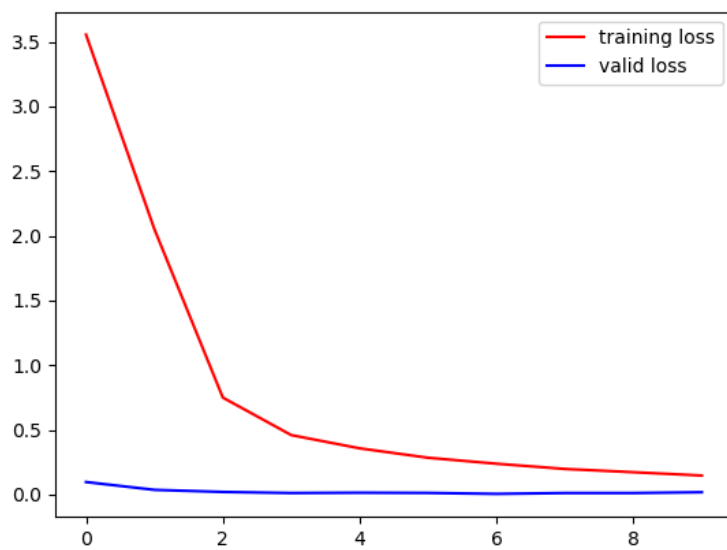
Problem 7.1.1. Re-write and re-train your fully-connected network on NIST36 in PyTorch. Plot training accuracy and loss over time.



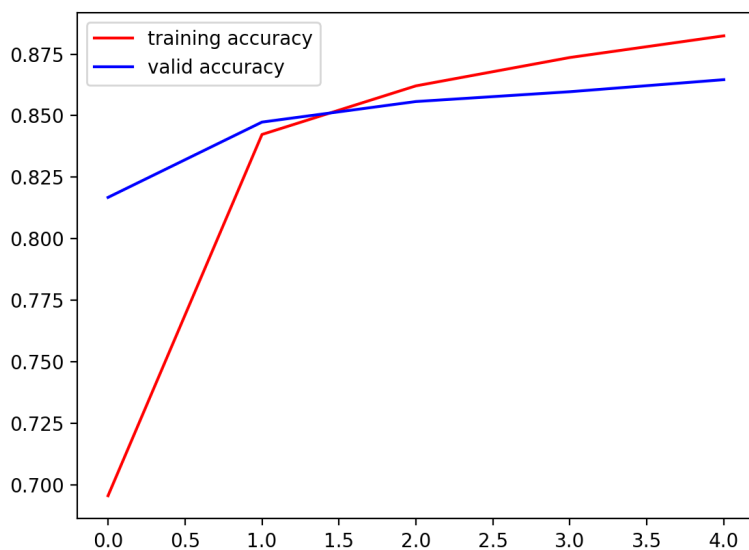
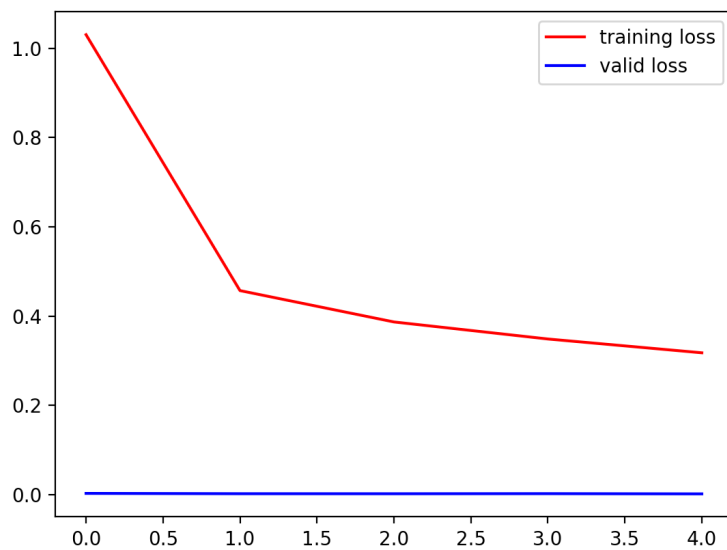
Problem 7.1.2. Train a convolutional neural network with PyTorch on MNIST. Plot training accuracy and loss over time. Since the results were too good, only 5 epochs were applied.



Problem 7.1.3. Train a convolutional neural network with PyTorch on the included NIST36 dataset. Again, since the results were too good, only 10 epochs were applied.



Problem 7.1.4. Train a convolutional neural network with PyTorch on the EMNIST Balanced dataset and evaluate it on the findLetters bounded boxes from the images folder.



The training results were plotted above. Only 8 iterations were applied. By the trained cnn on EMNIST data set with 8 iterations, the results were:

1. “deep.jpg”:
DEEP LEARNING
DEEPER LEARNING
DEEPEST LEARNING

2. “to_do_list.jpg”
T0 D0 LIST
2 MA KE A R0 P0 LXST
2 CHE KK 0F F RHE FIRSR
RHING 0N RO DO LIST
3 RXA2I 2 E YOU MVE ALREABY
COMPLETKD 2 RWINGS
T REWARD YOURSELFWXR
A NAP

3. “letters.jpg”:
A BC D E F G
H I J K L M N
B P A R S T Y
V W X X Z
I 2 3 4 S G 7 8 Y0

4. “haiku.jpg”:
HAQKHF ARE BAAR
BHR SBMARFMHA THAR QHNR MNKA AHNIR
RBFRIQPRMTHR

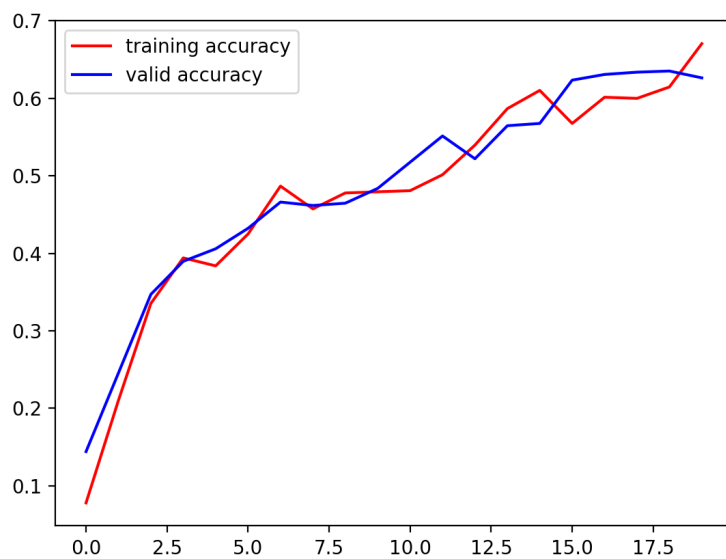
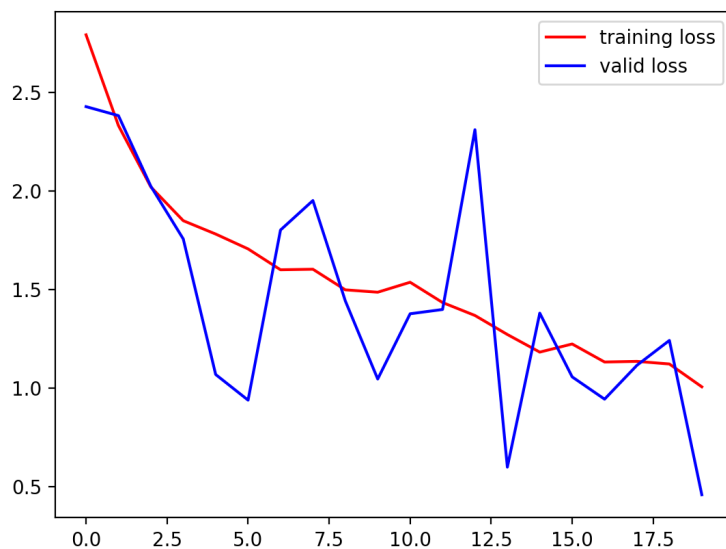
Again, for “to_do_list.jpg”, different padding and dilation were applied. We can see that, overall, the results were better than those from nn.

Problem 7.2. Fine-tune a single layer classifier using pytorch on the flowers 17 (or flowers 102!) dataset using squeezenet1_1, as well as an architecture youve designed yourself.

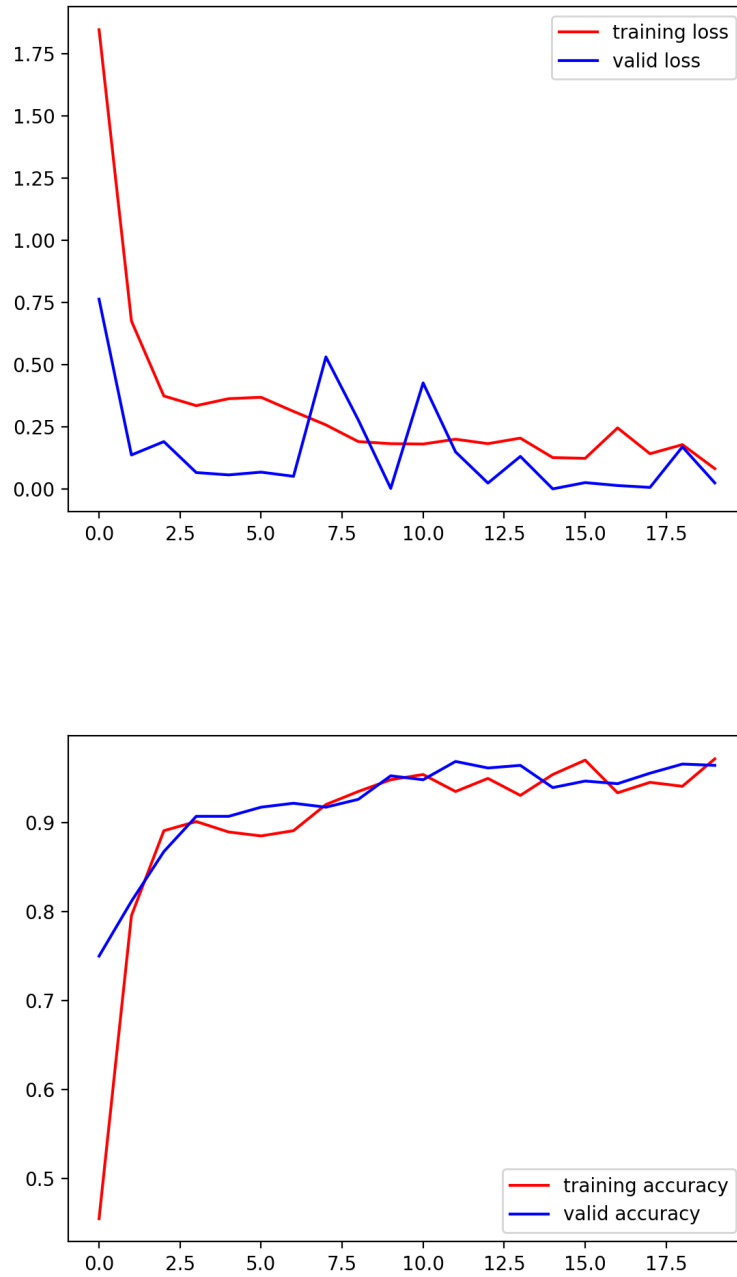
Reference to <https://discuss.pytorch.org/t/fine-tuning-squeezenet/3855/3>.

The learning rate was $1e-3$, and batch size was 16.

The loss and accuracy of my own architecture were shown as



The loss and accuracy of pretrained squeezenet 1.1 were shown as



From the figure above, it is clear that, the training results of pretrained squeezenet were way better than those of our own designed net from scratch. The first iteration from the pretrained net already achieved 45% accuracy, and the accuracy for both the training data and test data reached above 90% at the end. However, after 20 epochs, the accuracy of our net can only achieve around 65%.