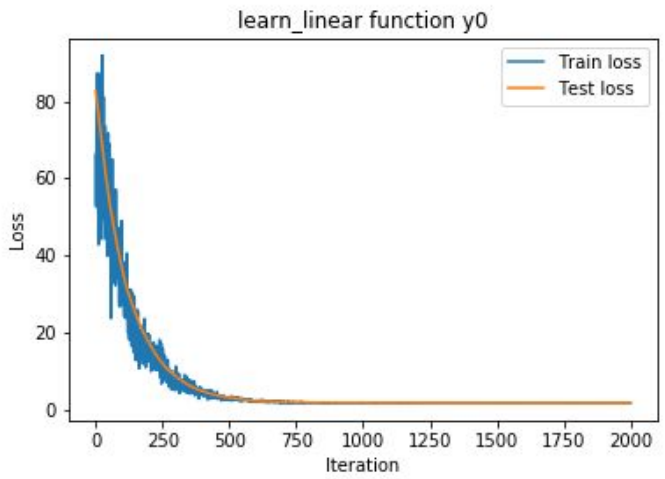
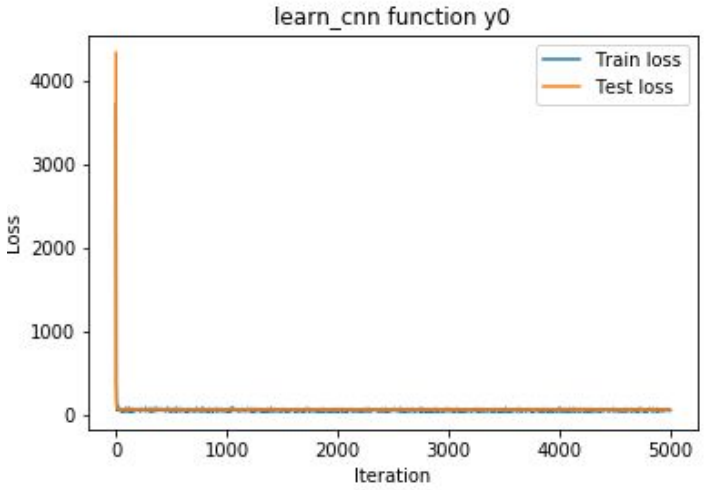
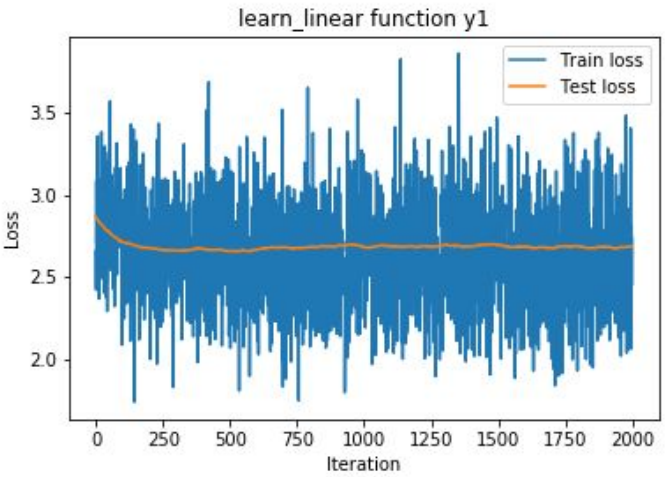
Convnets - ex3

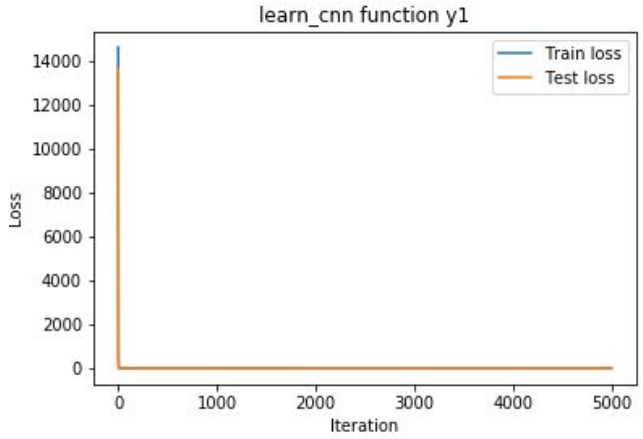
Id : 305248791

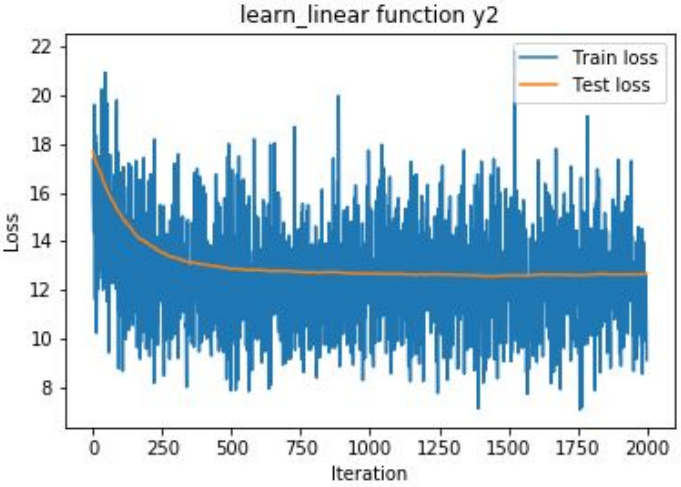
Toy Convnet:

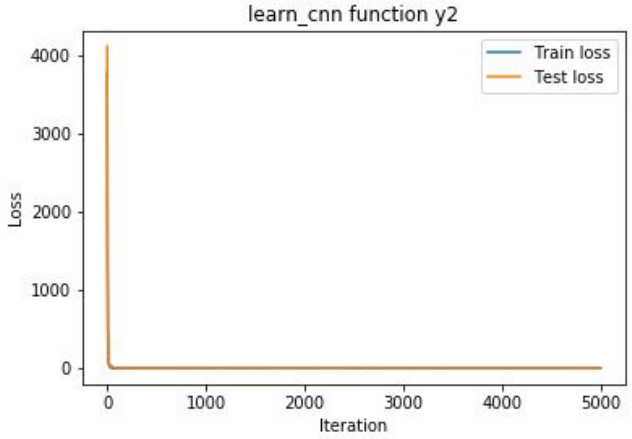






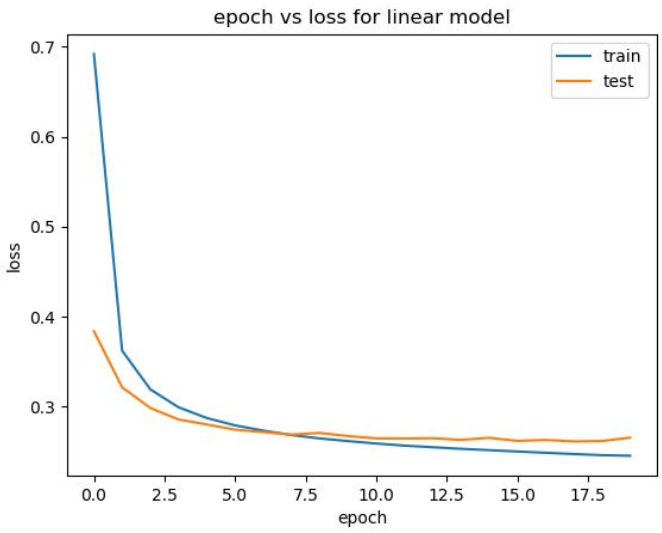


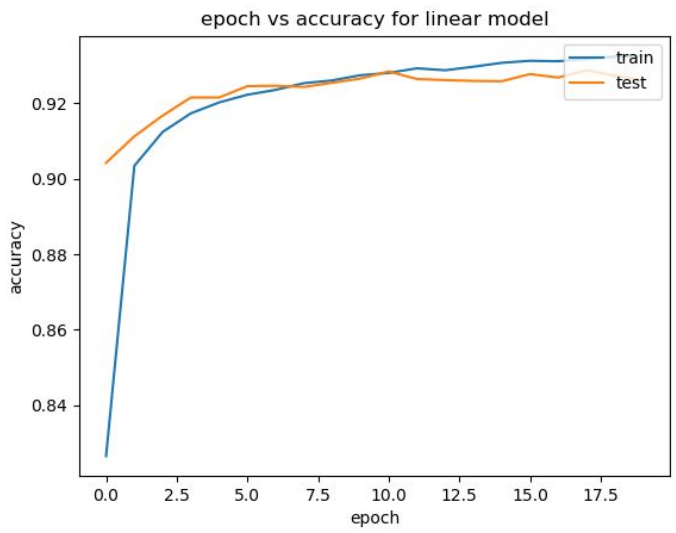




Learn cnn was able to learn all three functions well with fast convergence of the loss to zero loss. Learn cnn was better on all three functions and we can say that it is equal to the linear model on the linear function y0 were also there linear model achieved zero loss but more slowly.
The learn linear model was able to learn well only the linear function y0 (naturally) were the loss converges to zero, on both train data and test data.
The loss of the linear model reduces on the other two functions (y1,y2) and also converges only on test data but to values not close to zero.

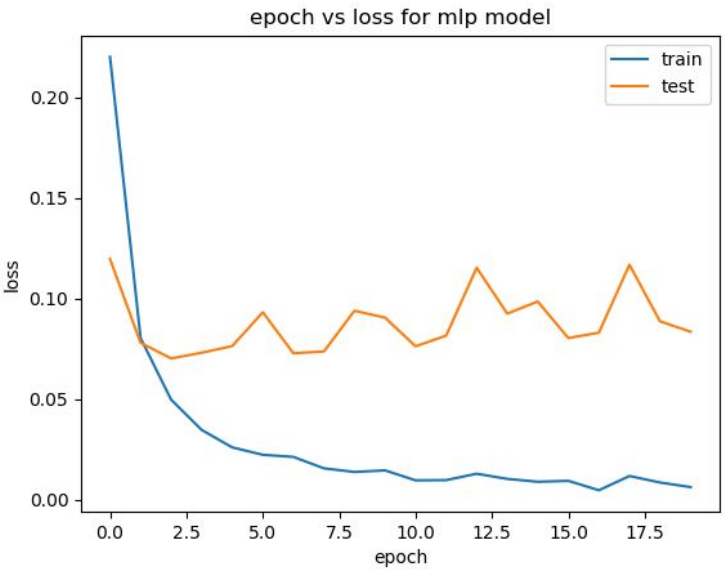
<u>MNIST classifiers :</u>
<u>Linear Model:</u>

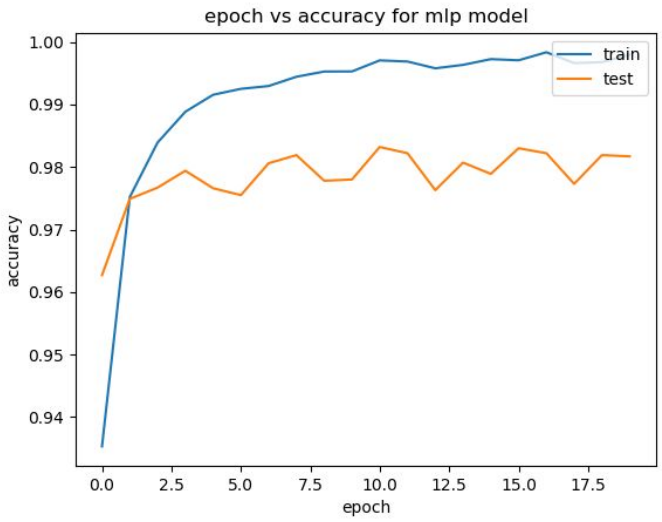




The model was able to generalize because it achieved good results in term of accuracy and loss on the test data, relatively to its simplicity it achieved good result with convergence of the loss to values close to zero and convergence of the accuracy to 92%+ on test and train data.

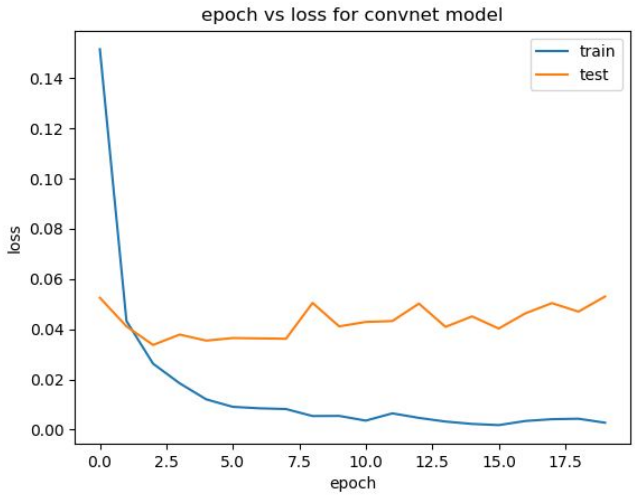
MLP:

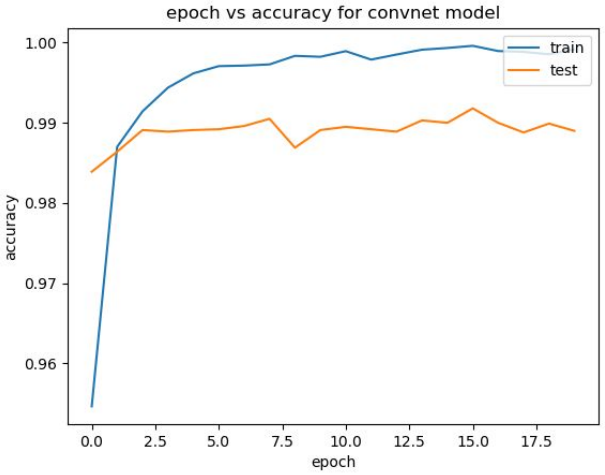




The model consists of two fully-connected layers with “relu” activation, and a last fully connected layer with “softmax” activation. I made these choices following the instructions of what layers and activation we should use for this model and this combination achieved good results in term of loss and accuracy both on train and test data. and it is better than the linear model as we can see from comparing the graphs.

Convnet:



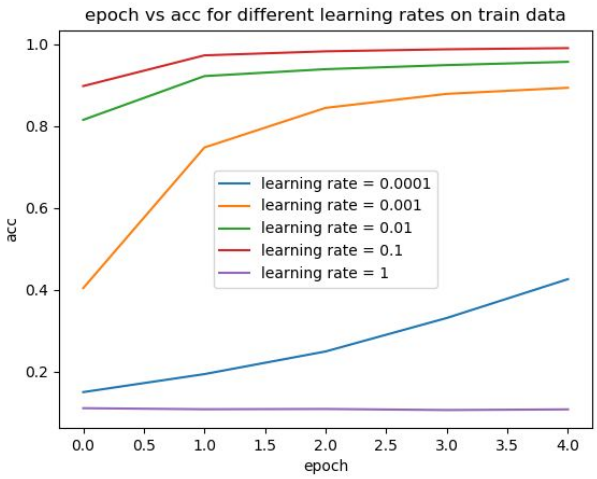


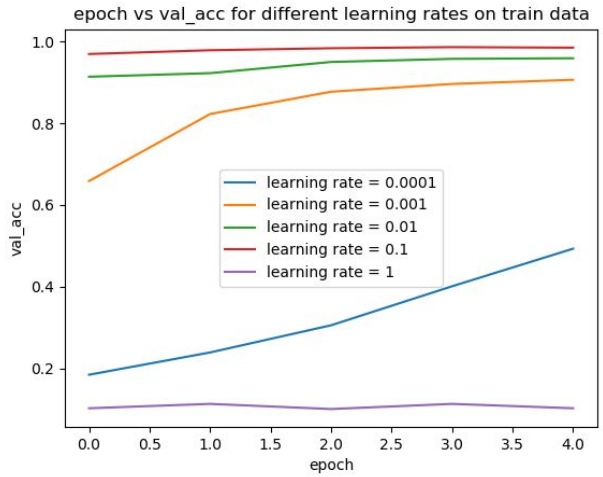
Here I’ve done something similar to the lenet-5 architecture combining convolution layers with maxpooling(for subsampling) and adding two fully-connected layers at the end.
The convnet achieved better results than MLP and the linear model as we can see from the graphs.

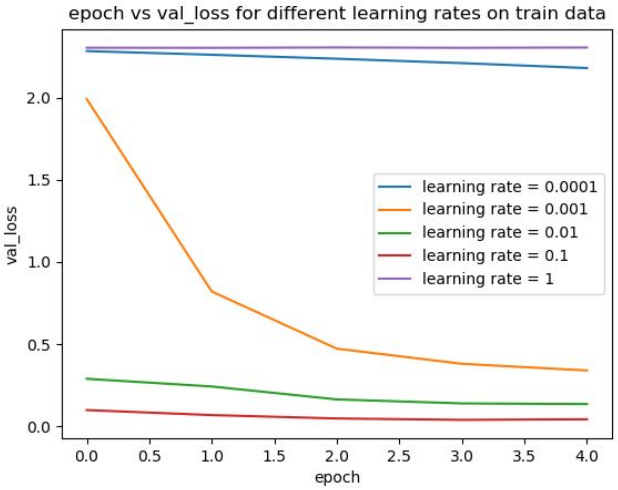
Hyper-Parameter Exploration (Learning rate):

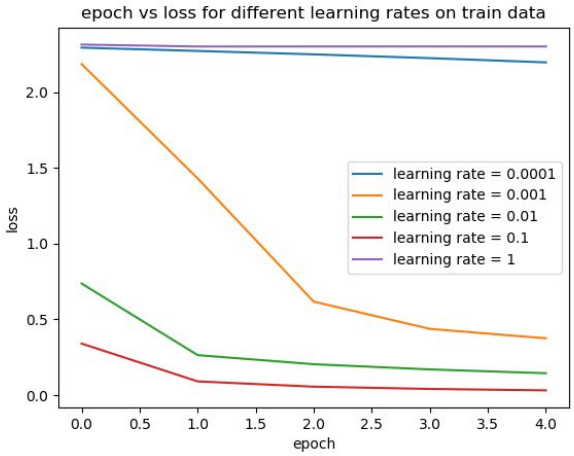
Up until now I used the adam optimizer which is a special variation of gradient descent that automatically figures out the best learning rate throughout the gradient descent process.

In order to check the effect of different learning rates I trained the convnet network using sgd optimizer, with 5 epochs for each learning rate:









As we can see from the graphs above the two learning rates that are problematic are the highest learning rate 1 (purple line) which doesn't converge and the lowest learning rate 0.0001 (blue line) which shows a very slow tendency for convergence. That is because, the size of weight changes is determined by the learning rate, low learning rates mean the model may take a long time training before it gets accurate, high learning rates mean the model may take huge steps around in a field always jumping over the best weights and never getting very accurate. The best learning rate in term of fast convergence and best results with regard to loss and accuracy on train and validation data is 0.1 (red line).

Autoencoders:

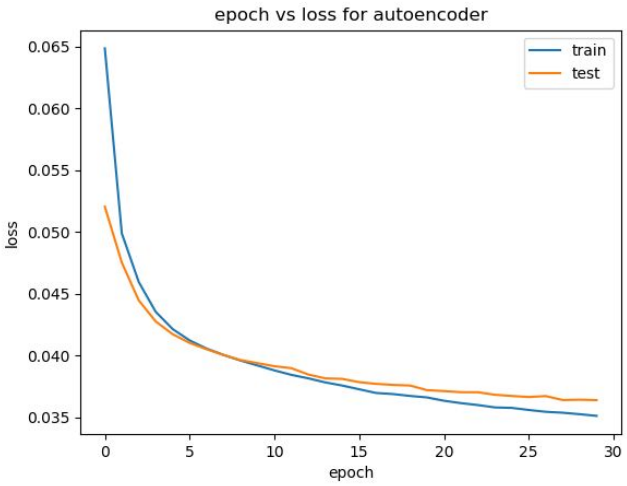
I decided to reduce the dimension by stacking fully connected layers untill I reduced the dimension down to 2 (encoder) and reverse the process until I reached 784 dimension (decoder). At first I thought that deeper neural network would be better but as I turned off some of the layers I achieved better results until I ended up with an autoencoder that performed best and better than the PCA, below is the architecture with layers I started with and later turned off (the turned off layers are commented with #):

```
   x = Flatten()(input_img)
   x = Dense(512, activation='relu')(x)
#    x = Dense(256, activation='relu')(x)
   x = Dense(128, activation='relu')(x)
#    x = Dense(64, activation='relu')(x)
#    x = Dense(32, activation='relu')(x)
#    x = Dense(16, activation='relu')(x)
#    x = Dense(8, activation='relu')(x)
#    x = Dense(4, activation='relu')(x)

   encoded = Dense(2, activation='relu', name='encoder')(x)

   #    x = Dense(4, activation='relu')(encoded)
#    x = Dense(8, activation='relu')(x)
#    x = Dense(16, activation='relu')(encoded)
#    x = Dense(32, activation='relu')(x)
#    x = Dense(64, activation='relu')(encoded)
   x = Dense(128, activation='relu')(encoded)
#    x = Dense(256, activation='relu')(x)
   x = Dense(512, activation='relu')(x)
   decoded = Dense(784, activation='sigmoid')(x)
```

I.e. the architecture I ended up with is:

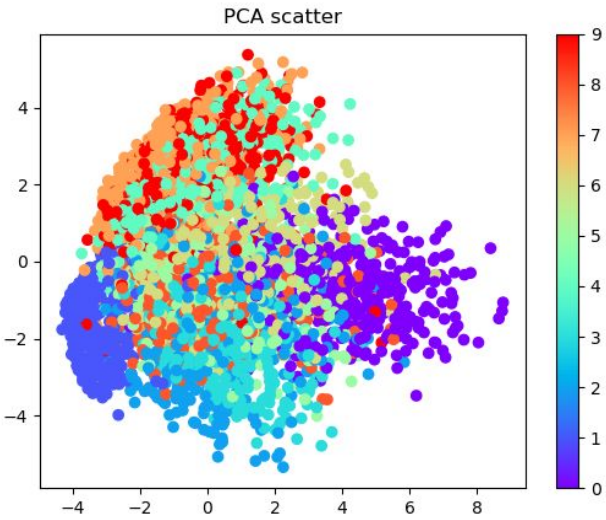
```
  input_img = Input(shape=(28, 28, 1))

   x = Flatten()(input_img)
   x = Dense(512, activation='relu')(x)
   x = Dense(128, activation='relu')(x)

   encoded = Dense(2, activation='relu', name='encoder')(x)

   x = Dense(128, activation='relu')(encoded)
   x = Dense(512, activation='relu')(x)
   decoded = Dense(784, activation='sigmoid')(x)
```

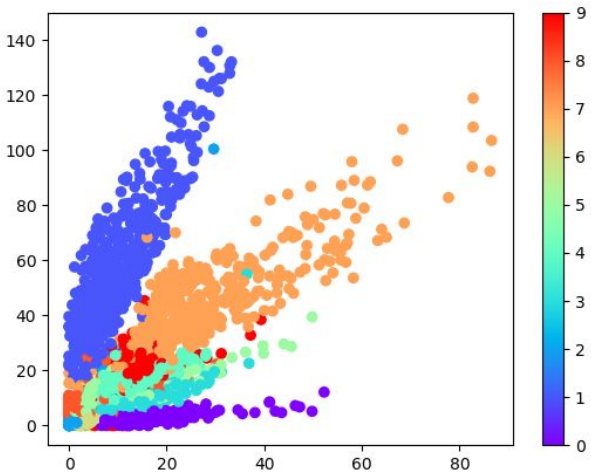
Learning curves for the autoencoder: we can see that the loss decreases as we increase the number of epochs.

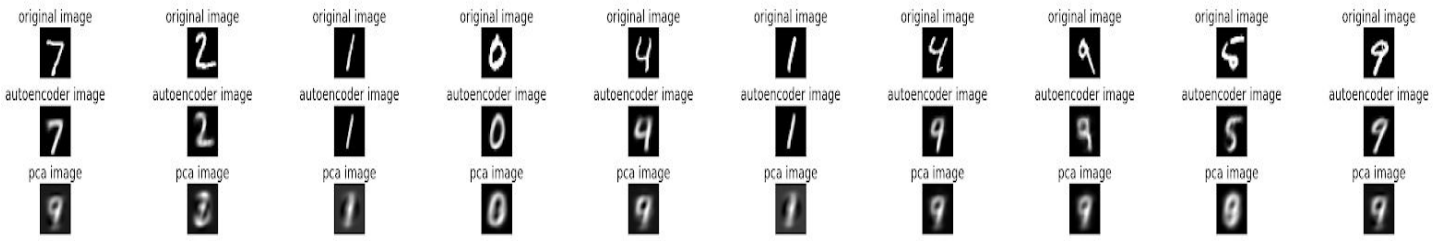


<u>Comparison with PCA:</u>

AUTOENCDOER SCATTER







My autoencoder extracts the structure better because we can see by comparing the scatter plots that in the autoencoder scatter the clusters of the digits are better separated than in the pca scatter. And in the autoencoder scatter the clusters are not stacked on upon the other as much as in the the pca scatter.
Also I added a plot for ten images and compared the decoding of images by my autoencoder and the pca (first row is the original image, second row is the decoded image by my autoencoder , third row is the decoded image by pca). We can see in this plot that the autoencoder successfully decoded 9 out of 10 images and the one mistake was between 9 and 4 which are similar in structure. On the other hand the pca only successfully decoded 5 out of 10 images and not in a very clear manner.

שאלה ד:

בגלל ש t משותף בין כל הקודקודים בשכבה
הוא שקול ל t-b שכבה אפינית עם Relu
רגיל, ולכן גם לו מושלם כוח ביטוי למודל
שלנו.

שאלה 2:

נסמן f:

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

$$(1)\ F'(x) = \frac{d}{dx}\left(\frac{1}{\sigma(x)}\right) = \frac{-\sigma'(x)}{\sigma(x)^2}$$

מצד שני,

$$(2)\ F'(x) = \frac{d}{dx}(1+e^{-x}) = -e^{-x} = 1 - F(x) = 1 - \frac{1}{\sigma(x)}$$

$$= \frac{\sigma(x) - 1}{\sigma(x)}$$

נשים לב (1) ו (2):

$$\frac{-\sigma'(x)}{\sigma(x)^2} = \frac{\sigma(x) - 1}{\sigma(x)}$$

$$\sigma'(x) = (1 - \sigma(x)) \cdot \sigma(x)$$

שאלה 3: באותו דרך נקבל

$$\frac{dl}{du} = 2(p-y)m^T = 2(m^T u - y)\, m^T$$
$$= -2y m^T$$

$$\frac{dl}{dw_j} = \frac{dl}{dp} \cdot \frac{dp}{dm_{ij}} \cdot \frac{dm}{do_{ij}} \cdot \frac{do_{ij}}{dw_j}$$

נסתכל על $\frac{do_{ij}}{dw_j}$,

$$o_{ij}(x_{i-1}, x_i, x_{i+1}, w_j) = \max\{w^T x, 0\} = 0$$

$$\frac{do_{ij}}{dw_j} = \frac{do_{ij}}{dr} \frac{dr}{dw_j} = x_{i-1:i+1} \cdot \begin{cases} 1 & o_{ij} > 0 \\ 0 & o_{ij} = 0 \end{cases} = 0$$

$$\Rightarrow \frac{dl}{dw_1} = \frac{dl}{dw_2} = 0$$

$$w_1 = w_1 - \lambda \frac{dl}{w_1} = 0$$
$$w_2 = w_2 - \lambda \frac{dl}{dw_2} = 0$$

באופטימיזציה בגלל נסתכלים על $\frac{do_{ij}}{dw_j}$ ובגלל שהם יוצא אפס כי $o_{ij} = 0$ כמו שמישהו לפני לכן w_1, w_2 יתקעו על אפס בכל האופטימיזציות. ובאות בגלל.