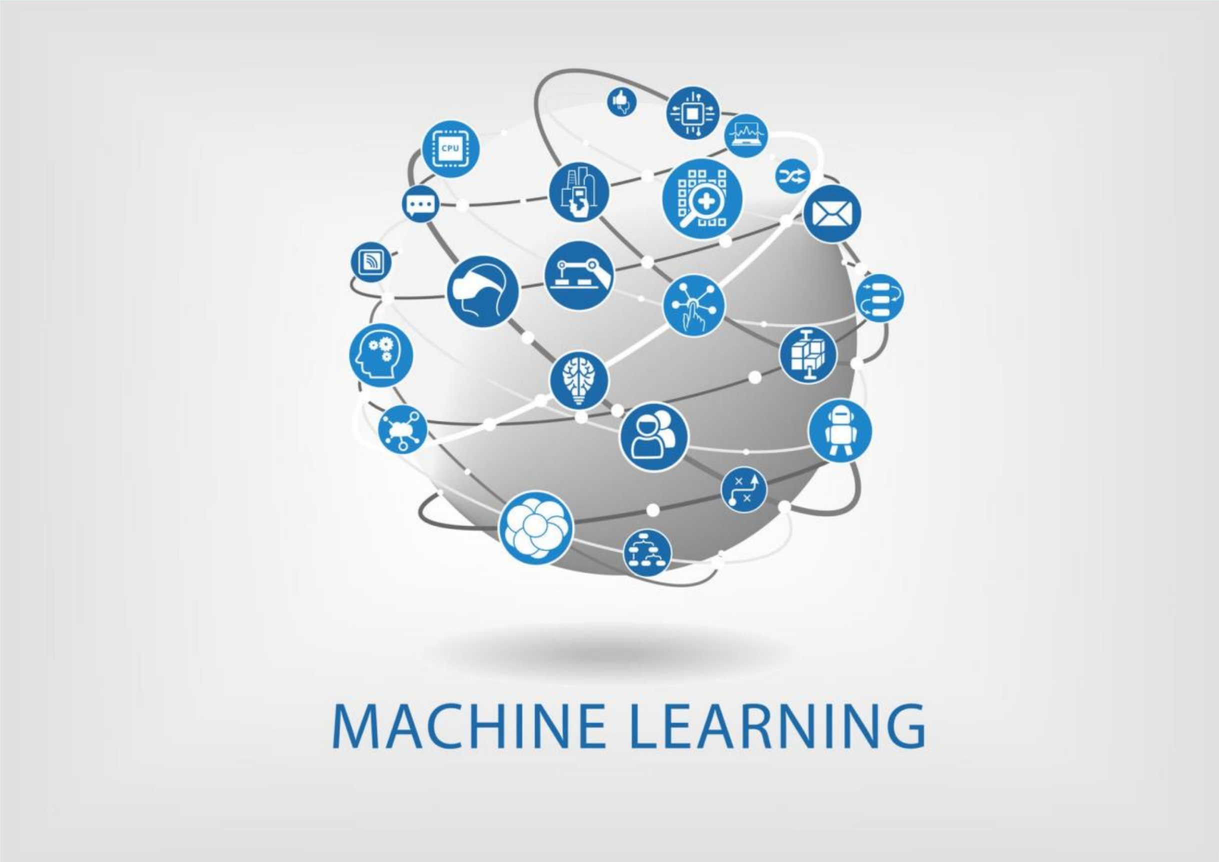
25th Feb. 2020.



NGOMBA LITOMBE – SN1199003

TAOFEEK A.O. YUSUF – SN1199006

Question 1

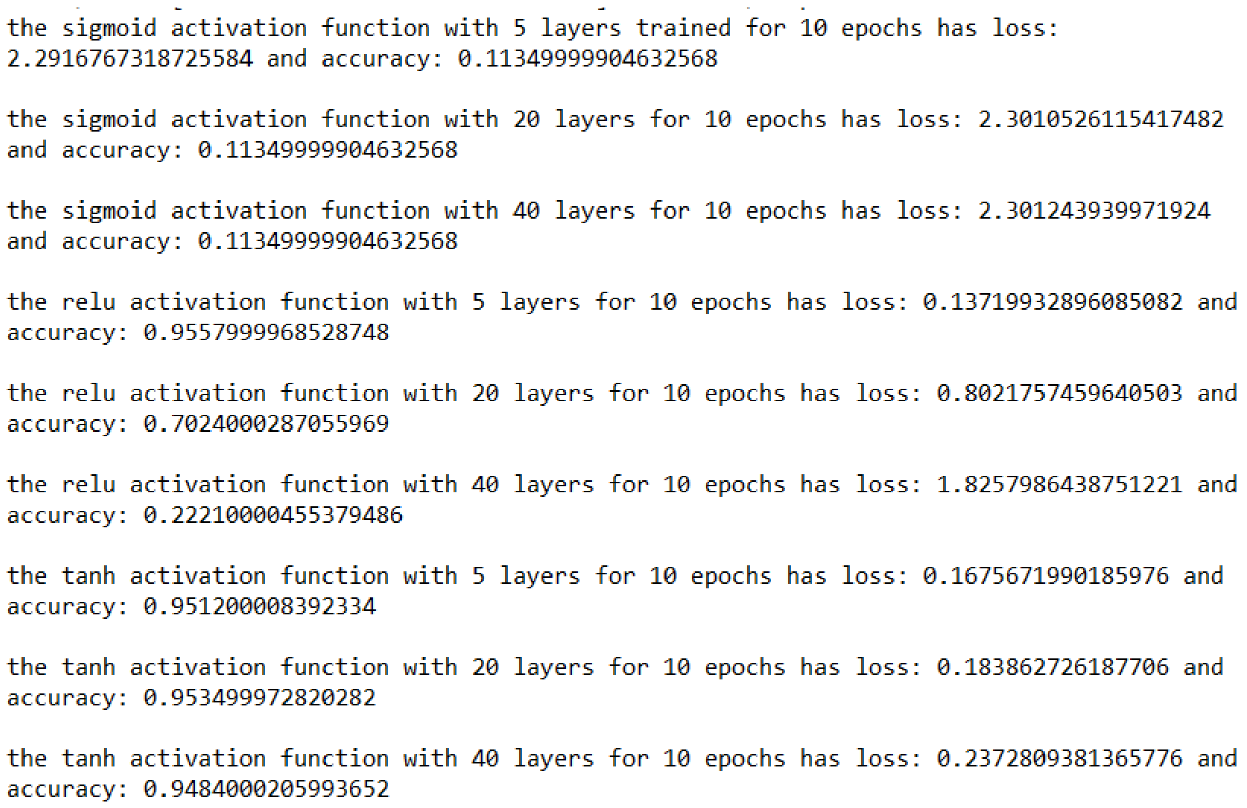
1a) Dataset

In this section, we worked with the MNIST dataset which is a database of handwritten digits that has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. The 60,000-pattern training set contained examples from approximately 250 writers.

1a)Inside hardcopy report

1b) Observations

The hyperbolic tangent performed better all round for the three different number of layers and the sigmoid function performed very poorly for the different number of layers, while the ReLU function performed very well for number of layers = 5, but as the number of layers increased to 20, the performance dropped and finally at, number of layers = 40, performed very poorly as shown in the result summary in figure 1 and table 1 below, respectively.



```
the sigmoid activation function with 5 layers trained for 10 epochs has loss:
2.2916767318725584 and accuracy: 0.11349999904632568

the sigmoid activation function with 20 layers for 10 epochs has loss: 2.3010526115417482
and accuracy: 0.11349999904632568

the sigmoid activation function with 40 layers for 10 epochs has loss: 2.301243939971924
and accuracy: 0.11349999904632568

the relu activation function with 5 layers for 10 epochs has loss: 0.13719932896085082 and
accuracy: 0.9557999968528748

the relu activation function with 20 layers for 10 epochs has loss: 0.8021757459640503 and
accuracy: 0.7024000287055969

the relu activation function with 40 layers for 10 epochs has loss: 1.8257986438751221 and
accuracy: 0.22210000455379486

the tanh activation function with 5 layers for 10 epochs has loss: 0.167567199018597 6 and
accuracy: 0.951200008392334

the tanh activation function with 20 layers for 10 epochs has loss: 0.183862726187706 and
accuracy: 0.953499972820282

the tanh activation function with 40 layers for 10 epochs has loss: 0.2372809381365776 and
accuracy: 0.9484000205993652
```

Figure 1: Training Results

	ReLU	**Tanh**	**sigmoid**
5 layers	0.95	0.95	0.11
20 layers	0.70	0.95	0.11
40 layers	0.22	0.94	0.11

Table 1: Training Result Summary

1c) Evaluation

RESULTS FOR MAXIMUM GRADIENT VS LAYER DEPTH FOR THE DIFFERENT NUMBER OF LAYERS

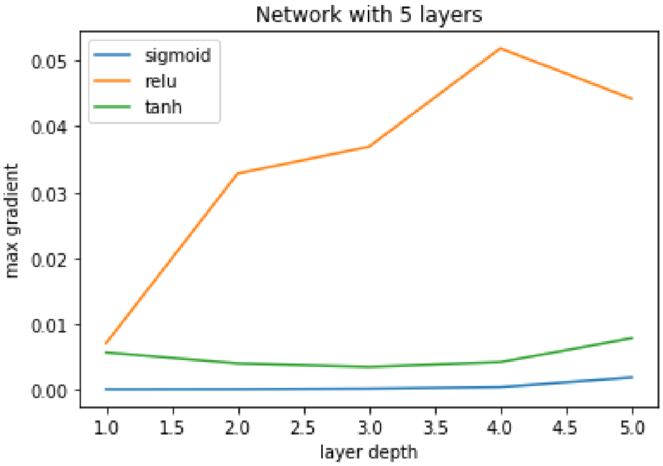


Figure 2: Max Gradient vs. 5 Layer Depth

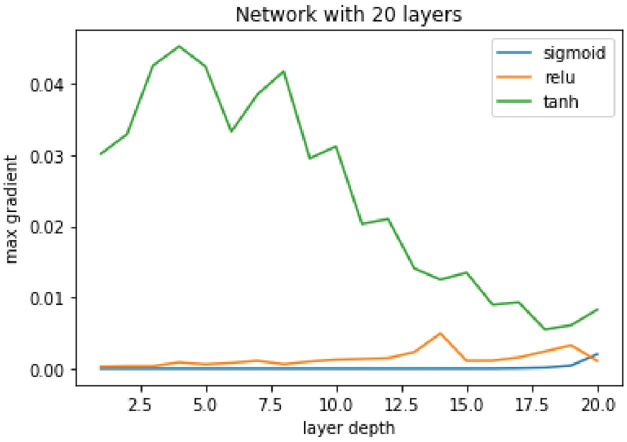


Figure 3: Max Gradient vs. 20 Layer Depth

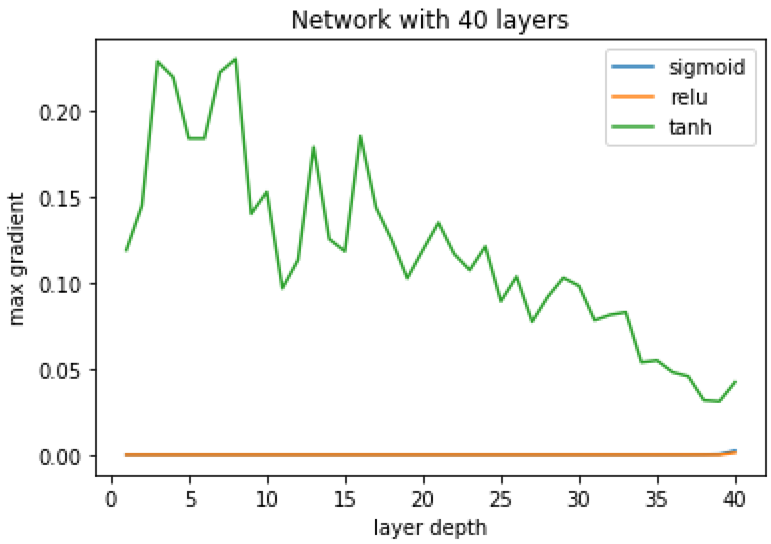
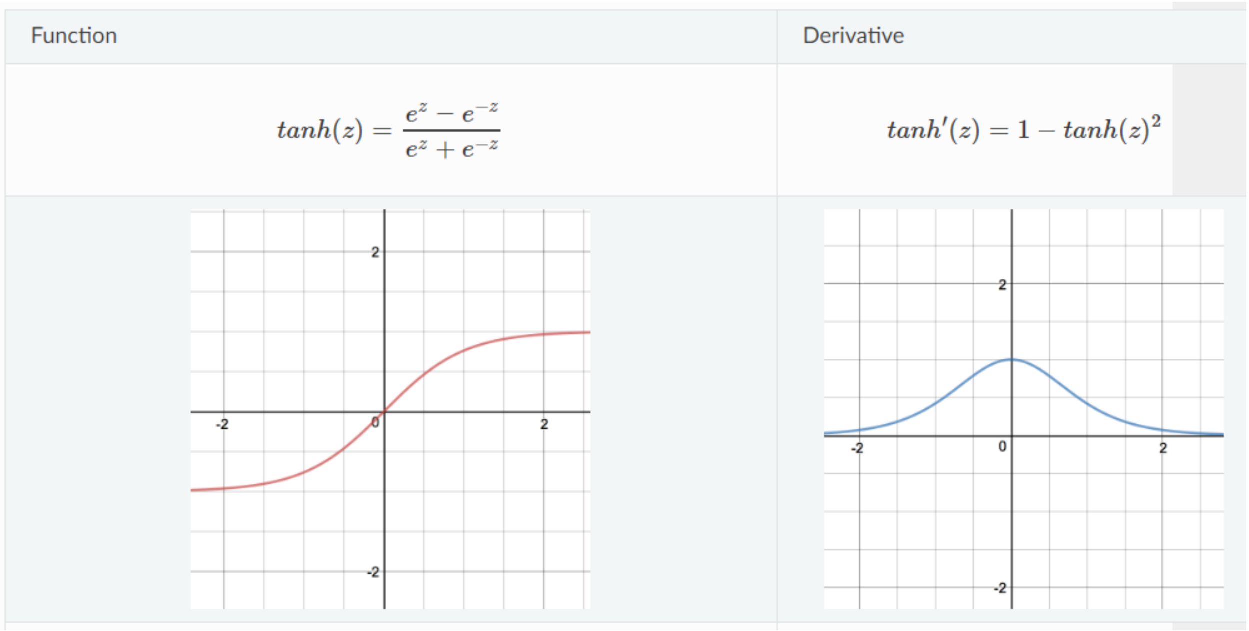


Figure 4: Max Gradient vs. 40 Layer Depth

Some Background Information on Activation Functions and their Ranges

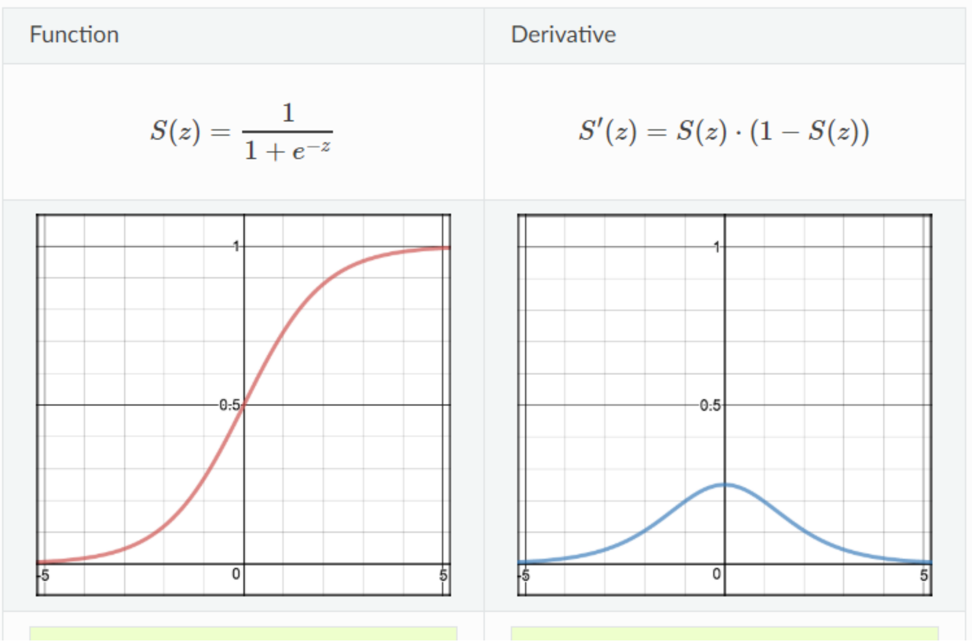
Hyperbolic Tangent



Function	Derivative
$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$tanh'(z) = 1 - tanh(z)^2$

Figure 5: Hyperbolic Tangent Function and Derivative

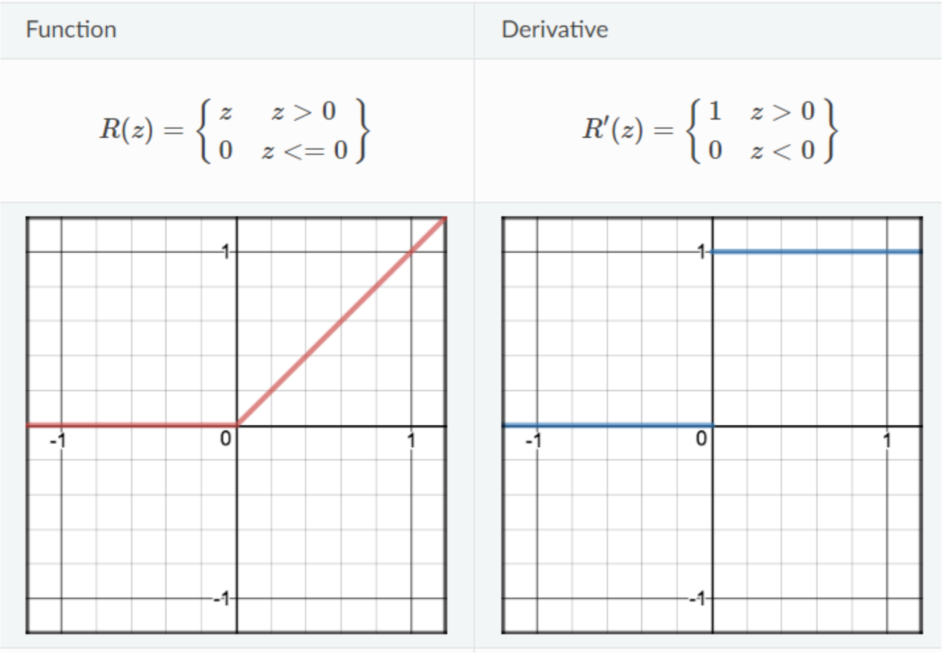
Sigmoid



Function	Derivative
$S(z) = \frac{1}{1 + e^{-z}}$	$S'(z) = S(z) \cdot (1 - S(z))$

Figure 6: Sigmoid Function and Derivative

ReLU



Function	Derivative
$R(z) = \begin{Bmatrix} z & z > 0 \\ 0 & z <= 0 \end{Bmatrix}$	$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ 0 & z < 0 \end{Bmatrix}$

Figure 7: ReLU Function and Derivative

EXPLANATION OF OBSERVATION IN 1B BASED ON GRAPHS IN 1C

Sigmoid

Across the three layers, we can see that the curve of the sigmoid is constant, that is, the network is not learning anything from the dataset (little or no weight updates are made). This is a manifestation of the vanishing gradient problem experienced by the sigmoid function as the number of layers increases, since the range of its derivative is quite low (0 – 1/4) as shown above.

This information alone is insufficient for us to make this conclusion, so we went ahead to check the performance of the sigmoid function on the test set with just two layers. It had an accuracy of 91% reaffirming our claim of the presence of the vanishing gradient problem. Thus, the poor performance of the sigmoid function can be attributed to the vanishing gradient problem.

ReLU

We expected ReLU to perform better than the hyperbolic tangent considering the fact that it does not suffer from the problem of the vanishing gradient but it isn't the case here. From the figure 2, it is quite obvious that ReLU does not suffer from the vanishing gradient problem. In figure 3, it shows that the network still learns but at a slower pace. To this effect, we can only think of a situation where the weights are gradually being set to zero. In figure 4, we experience the problem of dead neurons which are unable to learn as a result of weight updates that makes it never to activate on any data point again. The poor performance of ReLU at layers = 40 was as a result of the dead neurons problem as seen in figure 4.

Tanh

The hyperbolic tangent also suffers from the vanishing gradient problem but at a slower rate than the sigmoid function. Apart from this it also has the advantage of being zero centered, thus, enhancing optimization, which can be seen by the undulating nature of the graphs above in 1c. We believe this is the reason the hyperbolic tangent has been able to provide a good result despite the increase in network depth and its vanishing gradient problem.

1d) LeCun Activation Function

To put things in perspective, while comparing the LeCun activation function with the hyperbolic tangent, the LeCun function and its derivative was plotted as shown in the figures below.

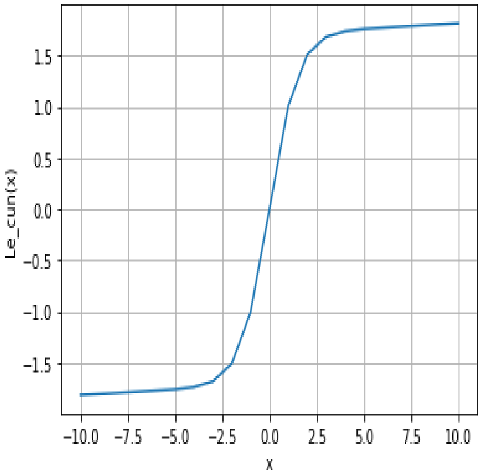


Figure 8: LeCun Function

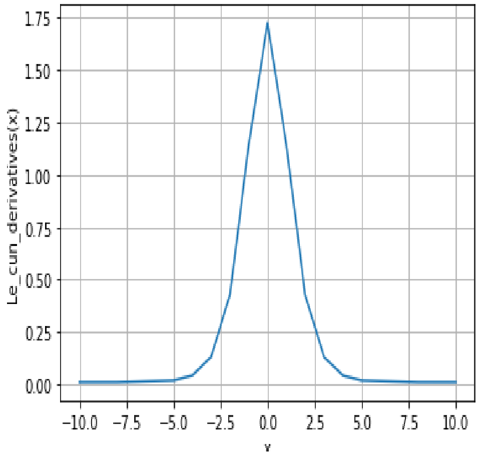


Figure 9: LeCun Derivative

GRAPHS OF MAXIMUM GRADIENT VS LAYER DEPTH

Considering the graph of the gradient functions of the LeCun and Tanh activation functions discussed earlier, it is expected as seen below that for any set of randomly initialized weights, the LeCun activation function would have a higher gradient than the hyperbolic tangent. Take the case of zero for instance, where Tanh has gradient 1 and LeCun has gradient of about 1.72.

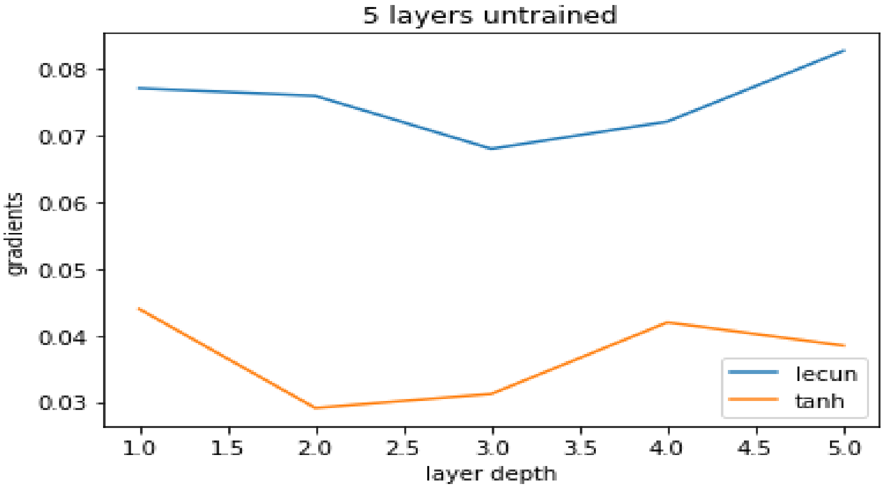


Figure 10: LeCun vs. Tanh for 5-Layer Depth

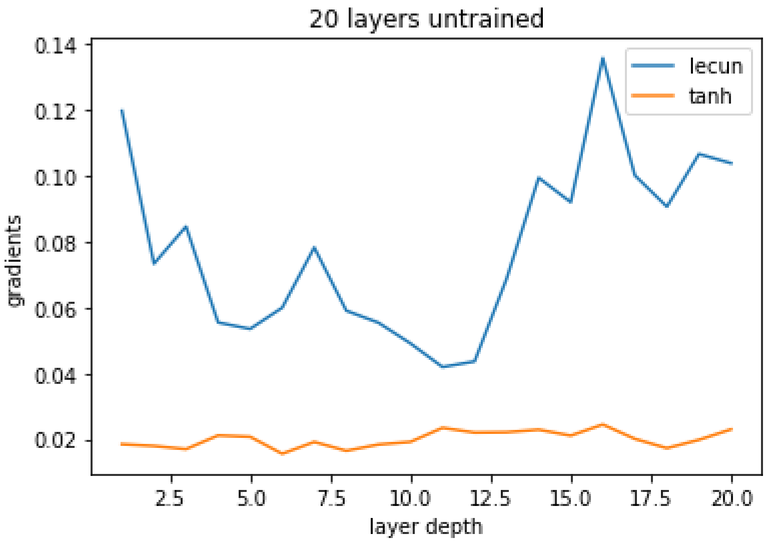


Figure 11: LeCun vs. Tanh for 20-Layer Depth

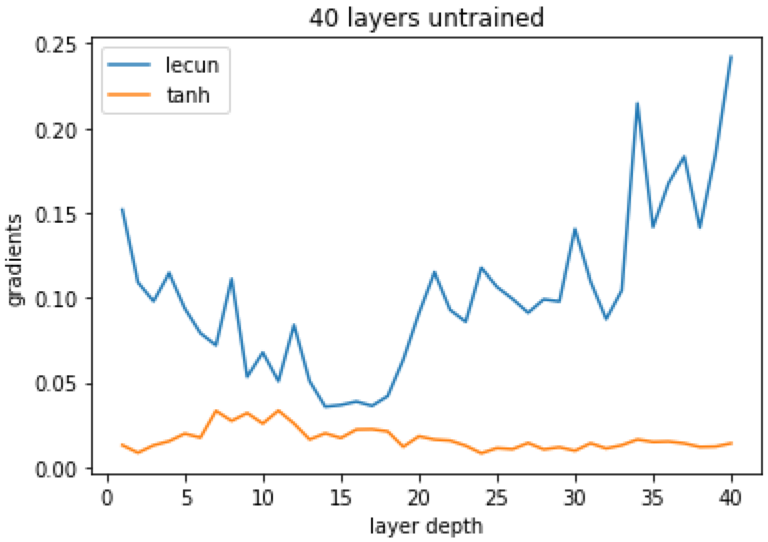


Figure 12: LeCun vs. Tanh for 40-Layer Depth

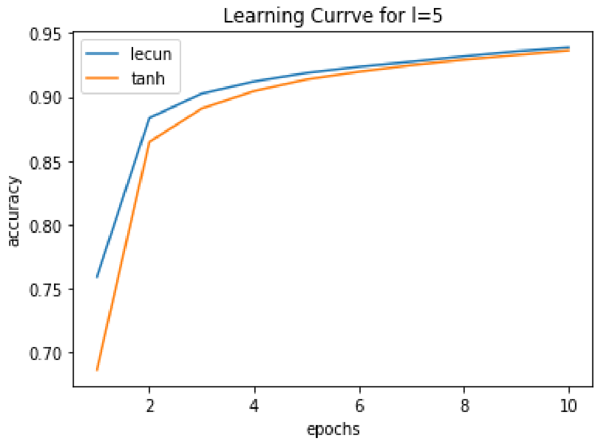


Figure 13: Learning Curve vs. Accuracy of LeCun vs. Tanh for 5-Layer Depth

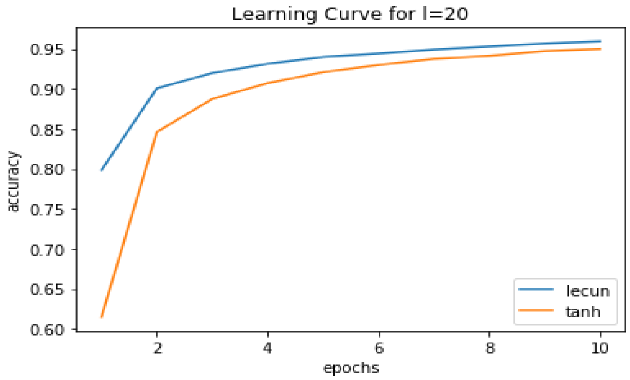


Figure 14: Learning Curve vs. Accuracy of LeCun vs. Tanh for 20-Layer Depth

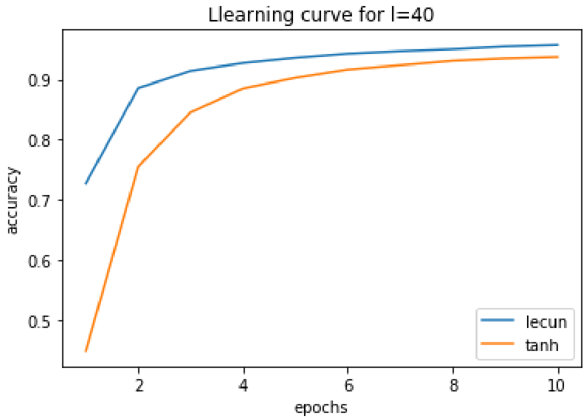


Figure 15: Learning Curve vs. Accuracy of LeCun vs. Tanh for 40-Layer Depth

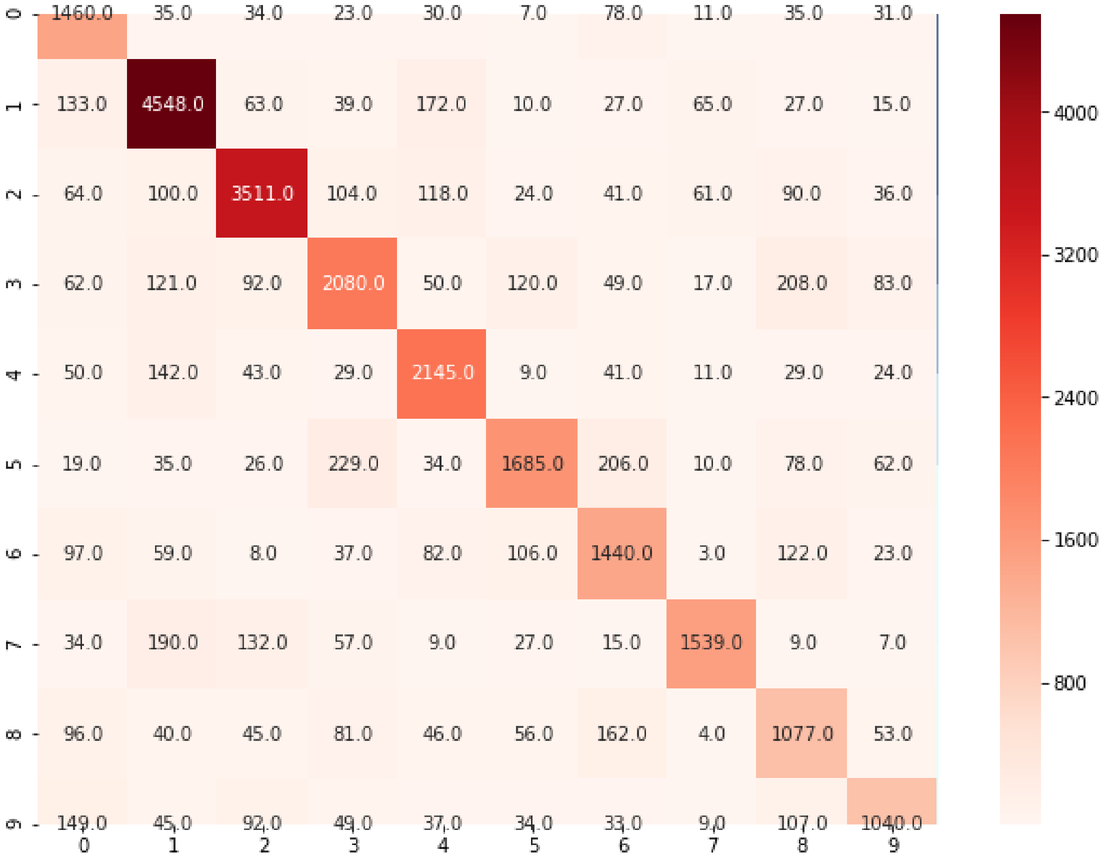
Since the range of movement of the gradients for the LeCun function is wider, than that of the hyperbolic tangent, this shows that the learning is faster with LeCun.

Question 2

2a) About Dataset

SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST and comes from a real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

Confusion Matrix



	0	1	2	3	4	5	6	7	8	9
0	1460.0	35.0	34.0	23.0	30.0	7.0	78.0	11.0	35.0	31.0
1	133.0	4548.0	63.0	39.0	172.0	10.0	27.0	65.0	27.0	15.0
2	64.0	100.0	3511.0	104.0	118.0	24.0	41.0	61.0	90.0	36.0
3	62.0	121.0	92.0	2080.0	50.0	120.0	49.0	17.0	208.0	83.0
4	50.0	142.0	43.0	29.0	2145.0	9.0	41.0	11.0	29.0	24.0
5	19.0	35.0	26.0	229.0	34.0	1685.0	206.0	10.0	78.0	62.0
6	97.0	59.0	8.0	37.0	82.0	106.0	1440.0	3.0	122.0	23.0
7	34.0	190.0	132.0	57.0	9.0	27.0	15.0	1539.0	9.0	7.0
8	96.0	40.0	45.0	81.0	46.0	56.0	162.0	4.0	1077.0	53.0
9	149.0	45.0	92.0	49.0	37.0	34.0	33.0	9.0	107.0	1040.0

Figure 16: Confusion Matrix for the Test Dataset

Test size = 26, 032 - **Number of correct Predictions** = 20, 525 - **Accuracy** = 79%

2b) Model Plot Output

From each class, an image was selected at random and the output from all the convolutional filters in the network displayed. In our code, the display is automatically made for all the random selected images in the ten classes but for demonstrative purposes in this report, we shall make use of just one.

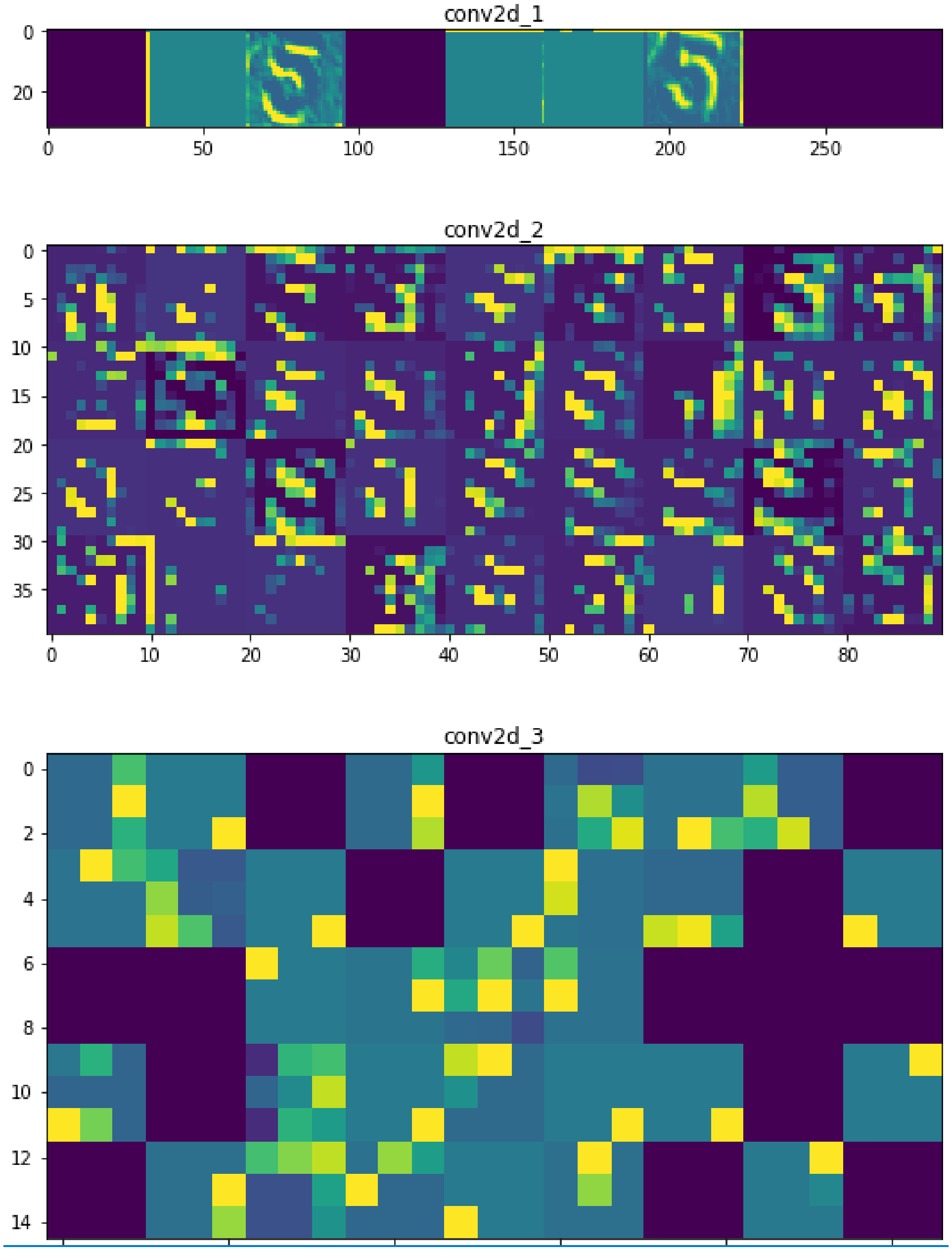


Figure 17: Convolution Filter

Each row is made up of 9 cells and each cell represents the output from one particular neuron. The first convolutional layer has 9 neurons, the second layer has 9x4 and the last layer has 49 neurons (we maintained a constant 9 rows to make the grid, therefore some cells in the last figure do not represent neurons).

In the first convolutional layer, most of the features of the image is retained although, a few picked up nothing and the filters are left blank.

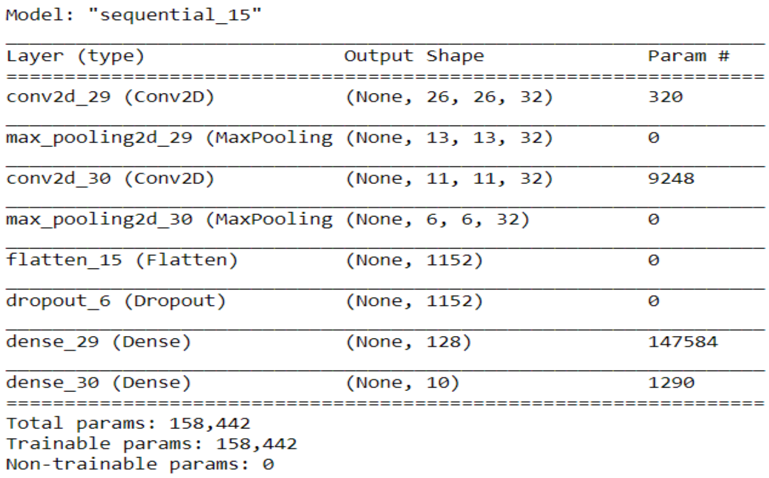
As we go deeper in the layers, the activations become increasingly abstract and less visually interpretable. They begin to encode higher-level concepts such as curves, corners and edges. Higher presentations carry increasingly less information about the visual contents of the image, and increasingly more information related to the class of the image.

As seen above, the last layer is not visually interpretable, at least with the human eye.

Question 3

Since the true test set of this question is unknown to us, we develop this model placing huge importance in the generalization ability of the model than on other metrics.

Broadly speaking, our model composes of two convolutional layers and two dense layers. The exact layout of the model is shown below.



```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_29 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 11, 11, 32)        9248
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 6, 6, 32)          0
_________________________________________________________________
flatten_15 (Flatten)         (None, 1152)              0
_________________________________________________________________
dropout_6 (Dropout)          (None, 1152)              0
_________________________________________________________________
dense_29 (Dense)             (None, 128)               147584
_________________________________________________________________
dense_30 (Dense)             (None, 10)                1290
=================================================================
Total params: 158,442
Trainable params: 158,442
Non-trainable params: 0
```

Figure 18: Network Architecture

In order to boost generalization, which is our primary objective, we performed the following:

Dataset Augmentation

We had two main datasets which we used to make more datasets by applying different data expansion techniques. The two main data are:

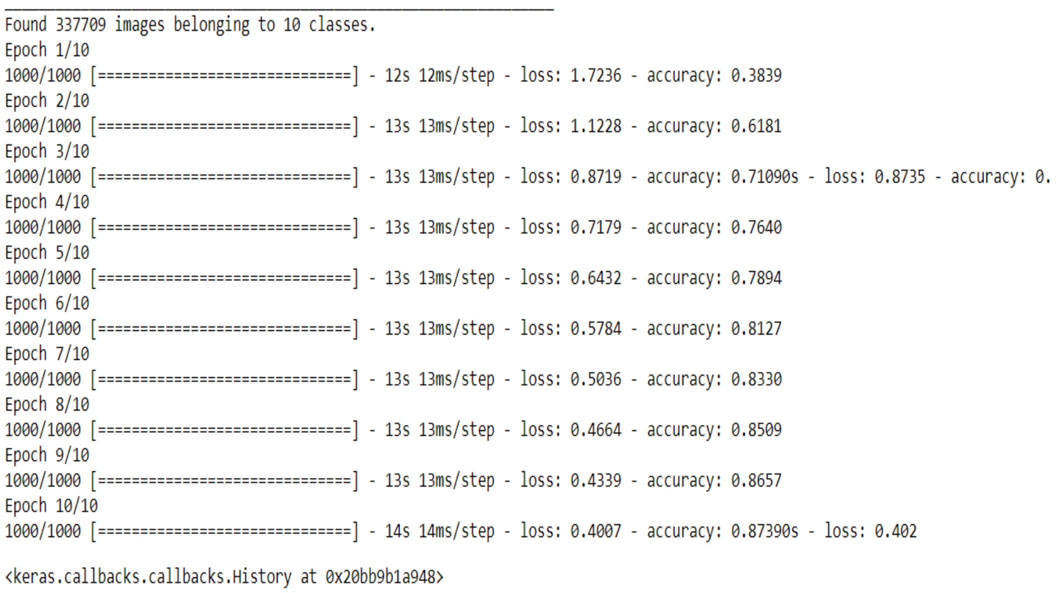
- The MNIST training dataset (60000 observations)
- Sample test set provided to us via announcement on eClass (122 observations)

Based on the above datasets, we managed to generate 3, 377, 709 images divided across the 10 different classes. The following dataset expansion techniques were utilized, in array of:

Horizontal flipping, Vertical flipping, Rotation, Brightness variation, Whitening, Shear intensity, Zoom, Width shift range, Dropout (rate=0.2).

Results

The Network was trained for 20 epochs and we didn't bother much about other schemes like early stopping callback but we didn't want to limit testing to a particular set and also because we hope that the diversity of the training set would play a key role. The following results were obtained during model training operation.



```
_________________________________________________________________
Found 337709 images belonging to 10 classes.
Epoch 1/10
1000/1000 [==============================] - 12s 12ms/step - loss: 1.7236 - accuracy: 0.3839
Epoch 2/10
1000/1000 [==============================] - 13s 13ms/step - loss: 1.1228 - accuracy: 0.6181
Epoch 3/10
1000/1000 [==============================] - 13s 13ms/step - loss: 0.8719 - accuracy: 0.71090s - loss: 0.8735 - accuracy: 0.
Epoch 4/10
1000/1000 [==============================] - 13s 13ms/step - loss: 0.7179 - accuracy: 0.7640
Epoch 5/10
1000/1000 [==============================] - 13s 13ms/step - loss: 0.6432 - accuracy: 0.7894
Epoch 6/10
1000/1000 [==============================] - 13s 13ms/step - loss: 0.5784 - accuracy: 0.8127
Epoch 7/10
1000/1000 [==============================] - 13s 13ms/step - loss: 0.5036 - accuracy: 0.8330
Epoch 8/10
1000/1000 [==============================] - 13s 13ms/step - loss: 0.4664 - accuracy: 0.8509
Epoch 9/10
1000/1000 [==============================] - 13s 13ms/step - loss: 0.4339 - accuracy: 0.8657
Epoch 10/10
1000/1000 [==============================] - 14s 14ms/step - loss: 0.4007 - accuracy: 0.87390s - loss: 0.402

<keras.callbacks.callbacks.History at 0x20bb9b1a948>
```

Figure 19: Model Training Result

After training, we test our model against the 122 images provided to use and also against the MNIST test data. We obtained the following accuracy measurements:

Test Set	**Accuracy**
Provided Testset	85%
MNIST Testset	80%