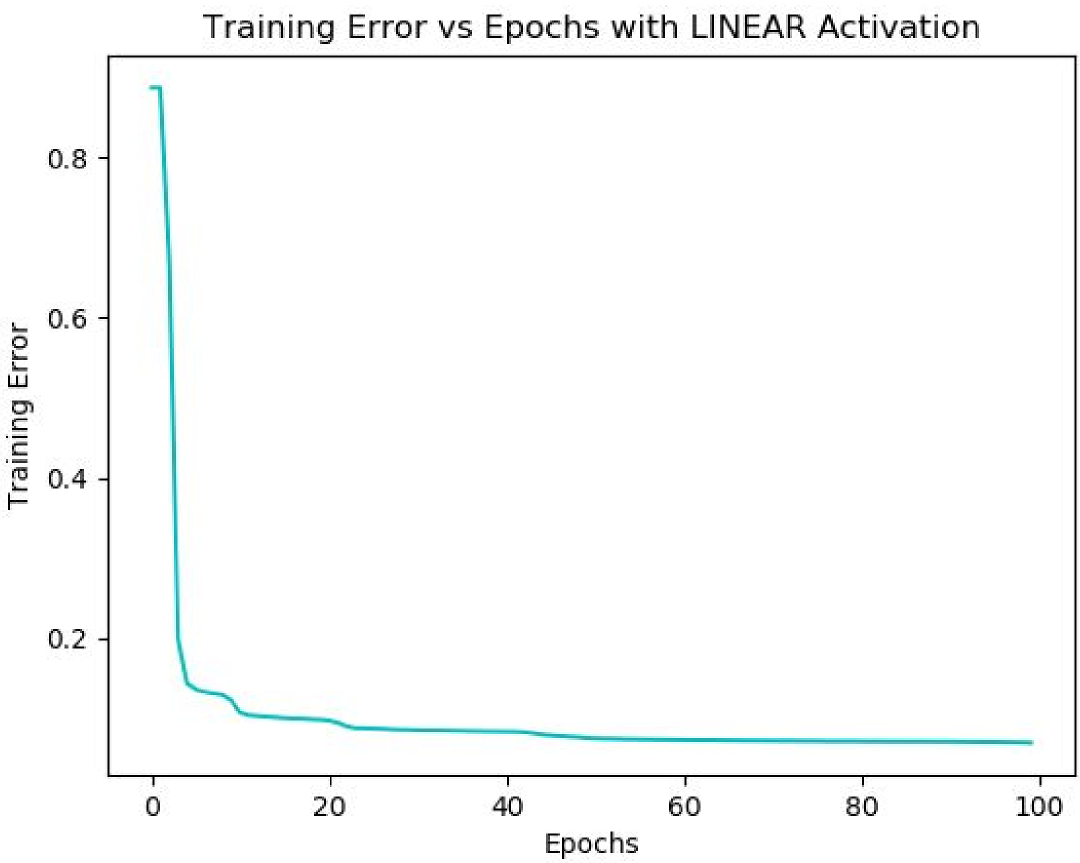
ML Assignment 3 Report

Avi Garg, 2017223

Question1.

Linear / Identity Activation Function



Accuracy on the test set with custom model came out to be 0.9193

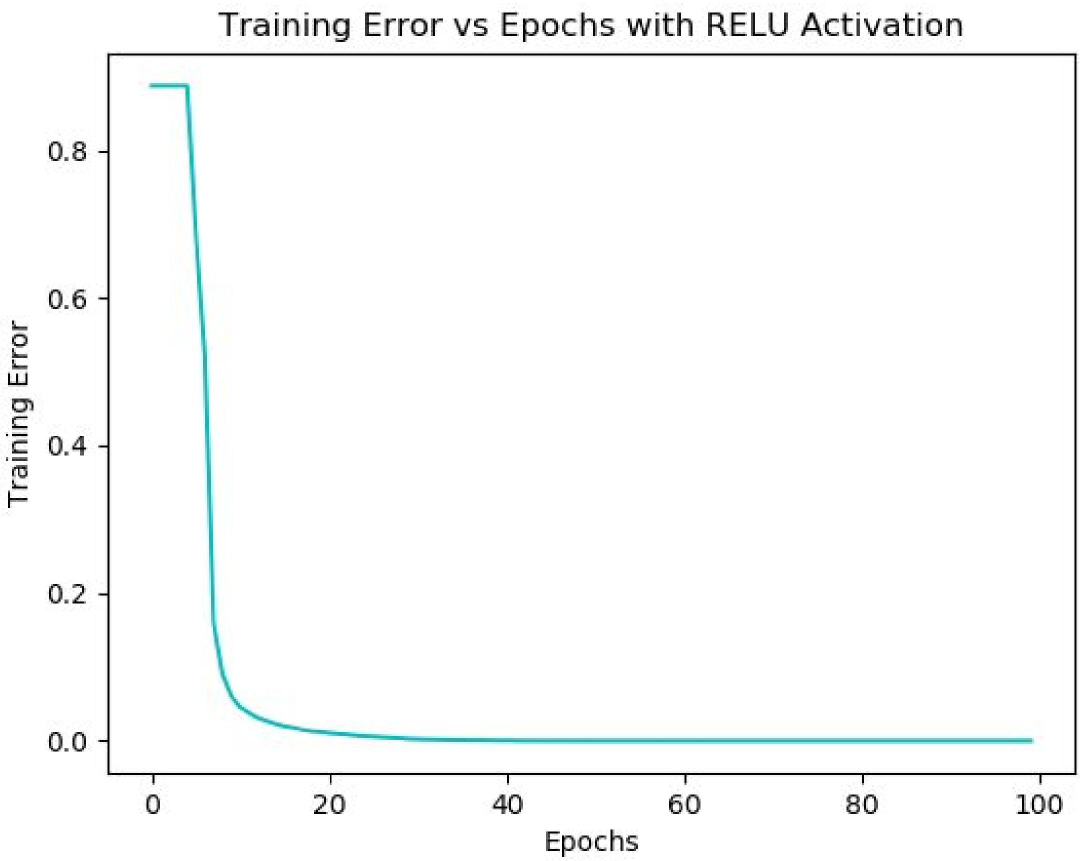
```
Test Accuracy with LINEAR:  0.9193
```

However, Sklearn gave an accuracy of 0.9231 with the same architecture.

```
Test Accuracy for IDENTITY, sklearn 0.9231
```

As the difference is really small i.e. 0.038 and hence insignificant, our model performs comparably to that of Sklearn.

ReLU Activation Function



Accuracy on the test set with custom model came out to be 0.9737

```
Test Accuracy with RELU:  0.9737
```

However, Sklearn gave an accuracy of 0.9795 with the same architecture.

```
Test Accuracy for RELU, sklearn 0.9795
```

As the difference is really small i.e. 0.0058 and hence insignificant, our model performs comparably to that of Sklearn. And it’s the best among all the models.

Sigmoid / Logistic Activation Function

Accuracy on the test set with custom model came out to be 0.9364

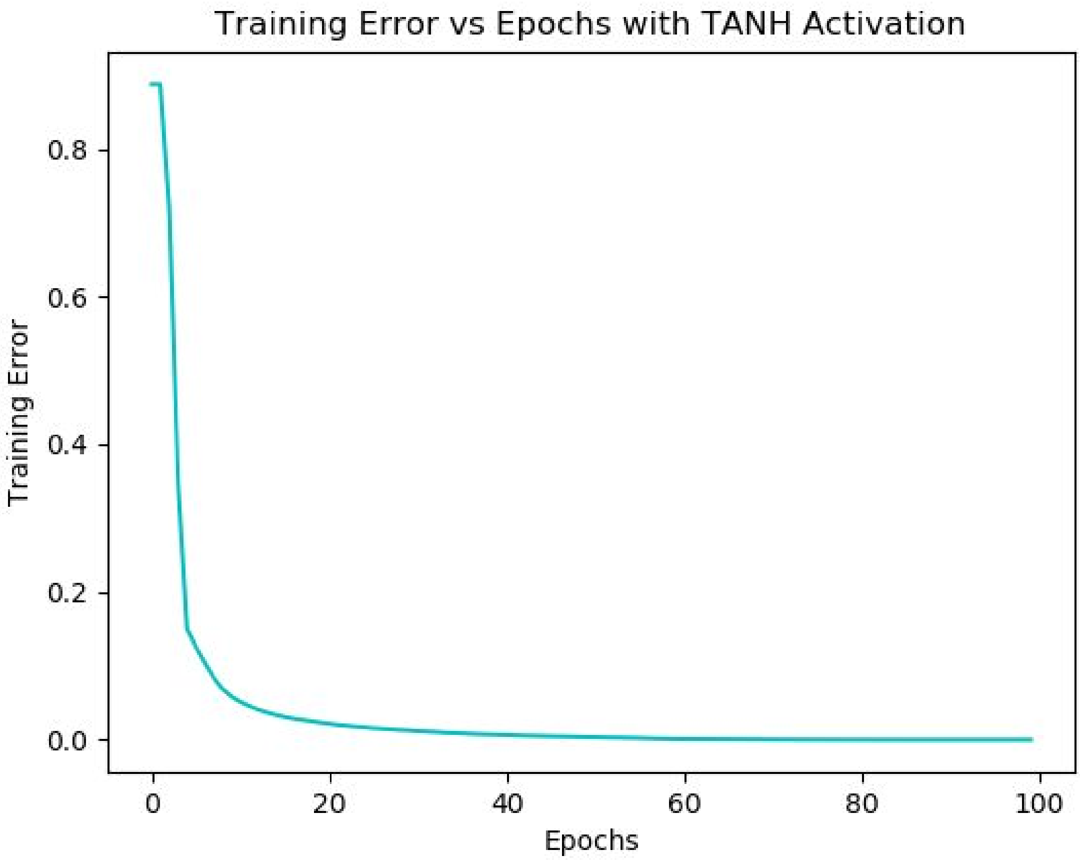
```
Test Accuracy with SIGMOID:  0.9364
```

However, Sklearn gave an accuracy of 0.9511 with the same architecture.

```
Test Accuracy for LOGISTIC, sklearn 0.9511
```

As the difference is really small i.e. 0.0147 and hence insignificant, our model performs comparably to that of Sklearn.

Tanh Activation Function



Accuracy on the test set with custom model came out to be 0.9713

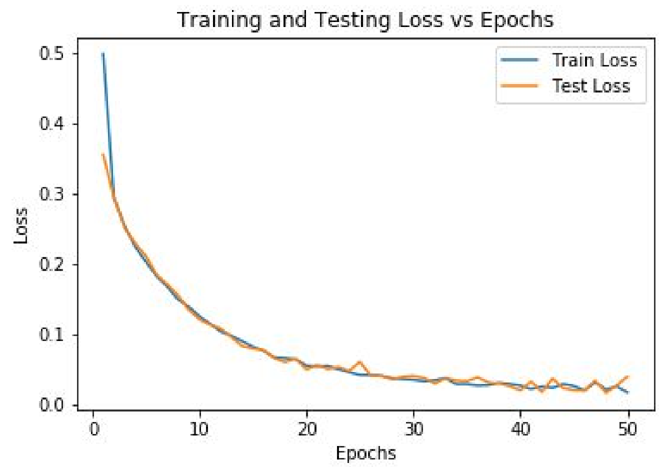
```
Test Accuracy with TANH:  0.9713
```

However, Sklearn gave an accuracy of 0.9801 with the same architecture.

```
Test Accuracy for TANH, sklearn 0.9801
```

As the difference is really small i.e. 0.0088 and hence insignificant, our model performs comparably to that of Sklearn.

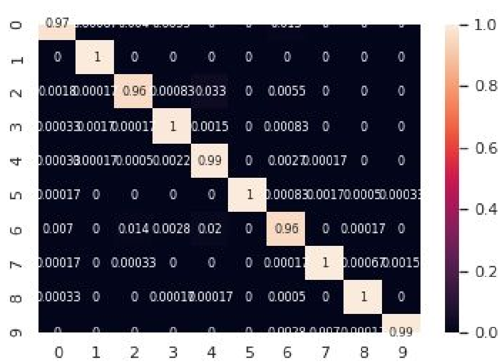
Question2.
Clearly as can be seen that Loss decreases per epoch, therefore, the gradient descent is working correctly.

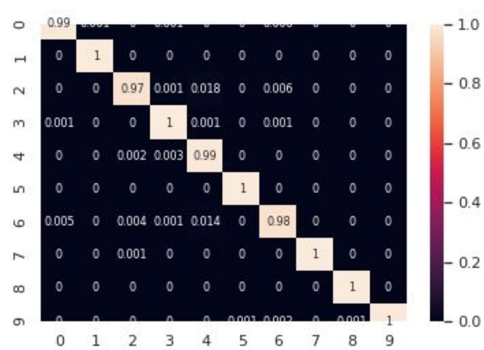



```
Training Accuracy: 0.9862
Testing Accuracy: 0.993
```

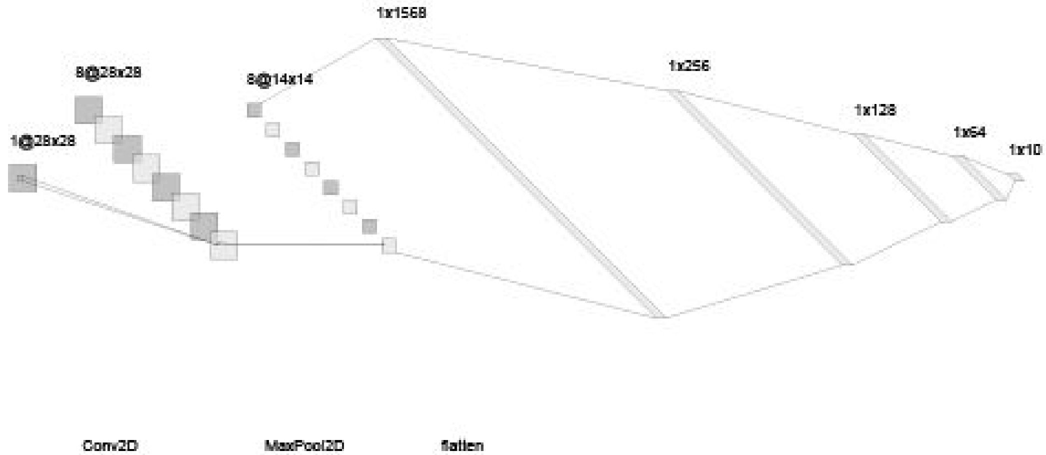
The model gives really accurate predictions after 50 epochs on a batch size of 100 as can be seen from the accuracy.

The confusion matrix for the train set is shown on the left and test set on the right. Clearly diagonal elements are heavy as compared to other entries, therefore, the model performs well.





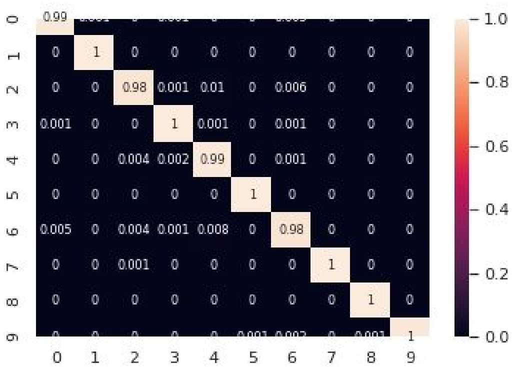
Model architecture is shown below, with filter size of 5, and 8 filters. MaxPool2D is used for pooling and the last layer is a fully connected layer.

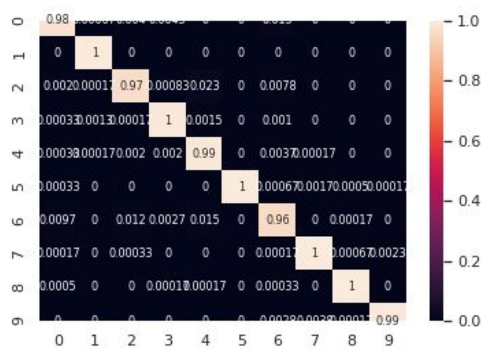


SVM was trained with ‘rbf’ kernel giving very good accuracy on both train and test set.

```
Train Accuracy SVM:  0.9876833333333334
Test Accuracy SVM:  0.9943
```

Confusion matrix for test set is on the left and the train set is on right, clearly diagonal elements are heavy as compared to other entries, therefore, the model performs well.



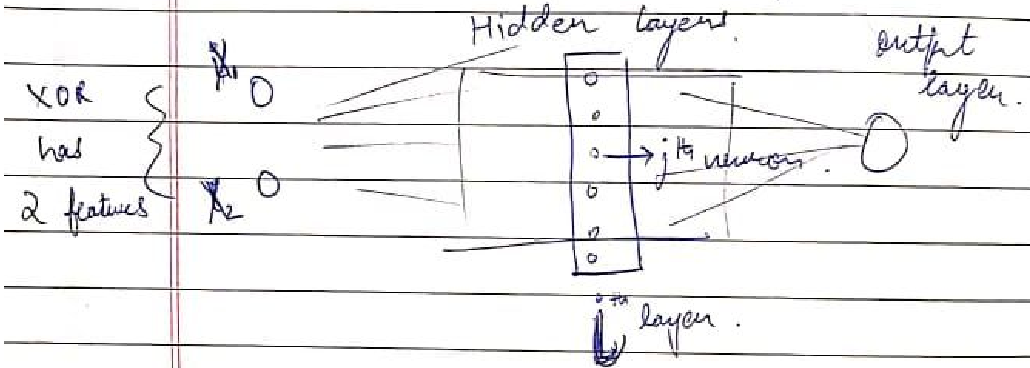


On the basis of the findings, we can see that SVM performs slightly better on the given dataset as it’s a classification problem and with ‘rbf’ kernel we mapped our input to a higher dimension to make it separable by a hyperplane.

Question3.

Q3. NO, XOR can't be classified using a neural net that has a linear activation function, reason being, it's equivalent to single layer perceptron.

Proof: Take a general neural network that has some hidden layers, now take 2 neurons say x_1, x_2 for input we prove this using induction.



For j^{th} neuron of layer 1, the induced local field i.e v_j,

$$v_j = w_{1j} x_1 + w_{2j} x_2 + b_1$$

$$y_j = \phi(v_j) \qquad \phi \rightarrow \text{linear}$$

$$= m(v_j) + c$$

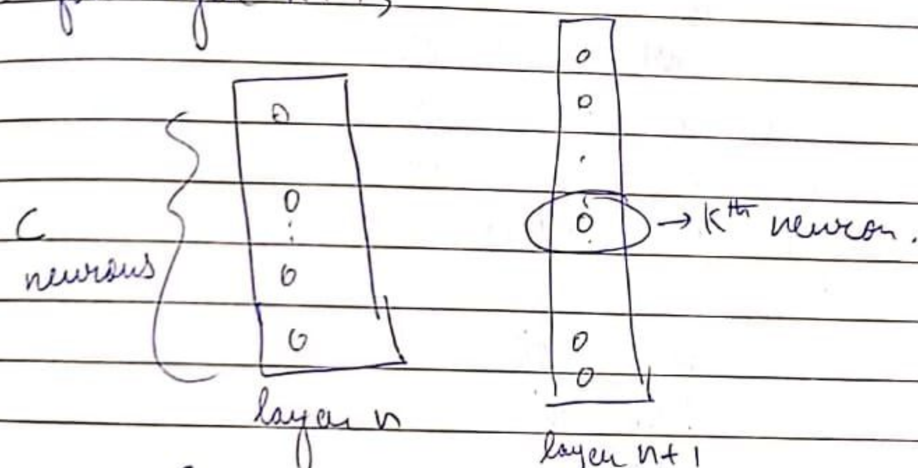
$$= m w_{1j} x_1 + m w_{2j} x_2 + m b_1 + c$$

$$= w_1' x_1 + w_2' x_2 + b_1'$$

i.e linear combination of inputs only.

For some layer $i = n$, assume true.

$\therefore$ for layer $n+1$,



$$V_k = \sum_{i=1}^{c} w_{ik} x_i + b_n,$$

$$\therefore y_k = \phi(v_k) = m(V_k) + c$$

$$= \sum_i m w_{ik} x_i + m b_n + c$$

$$= \sum w_i' x_i + b_n'$$

i.e. a linear combination of input of previous layers.

$\therefore$ By induction we see that output layer is linear combination of input layer. Thus it behaves as a SLP.

Question4.

Q.4.) There are 3 different components in a deep convolution Neural Network, namely,

(i) Convolution Layer:

- Helps in feature detection for an image.
- Uses correct filters / kernels which has learned features already and are hence used to detect the presence of the features in the input image.
- Advantages over a traditional NN, being it extracts some features even before the the input reaches F.C layer.
- It boosts the accuracy of model & helps F.C. layer classify better, reducing dimensions of image using convolution operation which is commutative

(ii) Pooling Layer:

- Applied after every convolution layer, this down samples the image by reducing the spatial size of image.
- It reduces parameters & hence computation time of the model.
- Maxpooling, Minpooling, Meanpooling are some examples of pooling.

- The advantage is that it adds invariancy to the model., tradianal ANN are translationally invariant, the pooling layer of deep CNN helps model to be translationaly invarient i.e. invarient to translation of local features.

(iii) Fully Connected Layer:

- Input is result of conv. layer & pooling layer. to classify the images.
- Result from previos layer are flattend & fed to this layer.
- The layer is similar to that of a traditional NN, however, they have much more information as they have confidence of feature of input.
- It performs better at clasification as due to conv. & pooling layers the no. of parameters are reduced drastically in FC layer without loss of information.

Question5.

```
def create layer(I, O, inp size, outsize, n1, n2):
    # n1 & n2 are layer current & forward layer

    loop i : 0 -> outsize :
      loop j : 0 -> outsize.
        endX <- j+2
        endY <- i+2.
        loop k : i -> endY + 1
          loop l : j -> endX + 1
            connect (n1, n2, I(k,l), O(i,j))

    kernel-size <- inpsize - (outsize - 1)* stride.

    loop i : 0 -> kernel-size
      loop j : 0 -> kernel-size
        loop k : 0 -> inpsize - kernel-size +1
          loop l : 0 -> inpsize - kernel-size+1
            loop m : i -> kernel-size + 1
              loop n : j -> kernel-size +j

                share(n1, n2, I(k,l),
                      O(i, j), I(m,n),
                      O(j, j)
                ).
```