Aayush Gupta | 2017125

Machine Learning

Assignment 3

As I have used Google colab hence only .ipynb files have been submitted.

Question1

Test accuracy for implemented ReLu: 0.9811

Test accuracy for implemented Sigmoid: 0.8954

Test accuracy for implemented Linear: 0.9007

Test accuracy for implemented Tanh: 0.9665

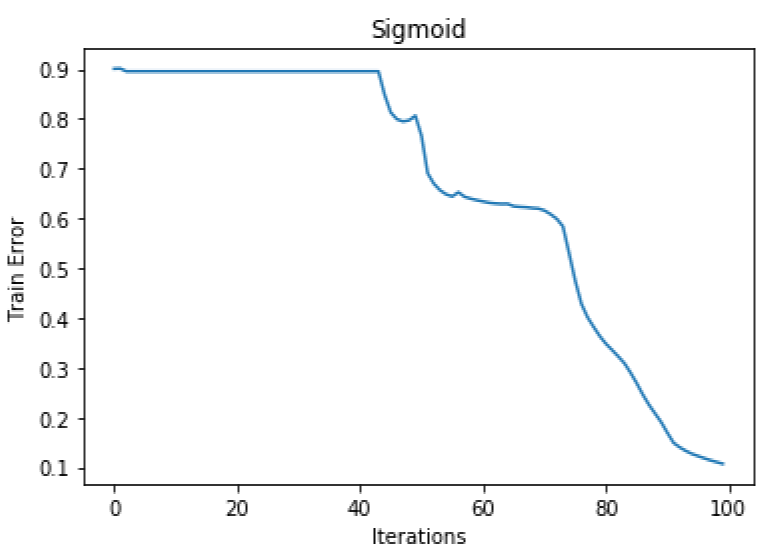
Training accuracy for ReLu: 1.0

Training accuracy for Sigmoid: 0.8934333

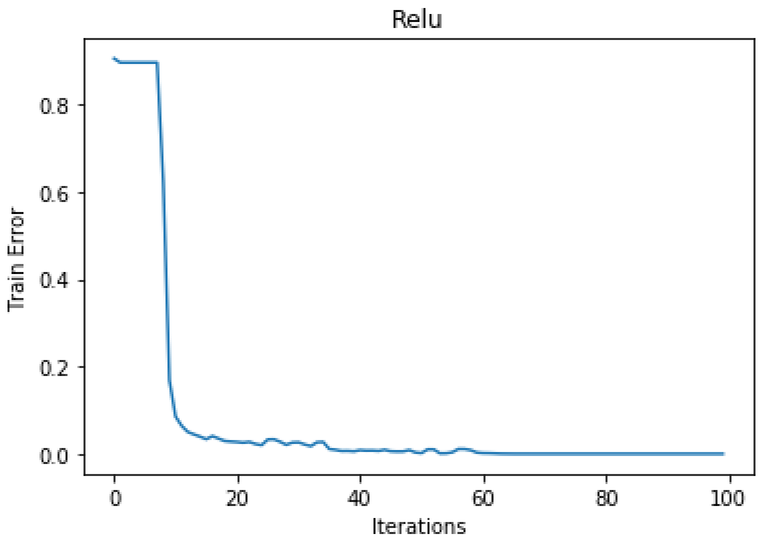
Training accuracy for Linear: 0.90596

Training accuracy for Tanh:0.99518333

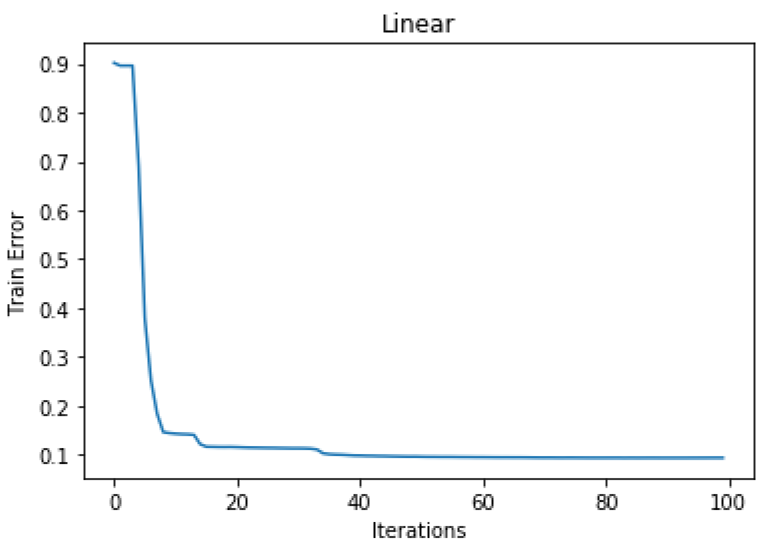
Plot for Sigmoid:



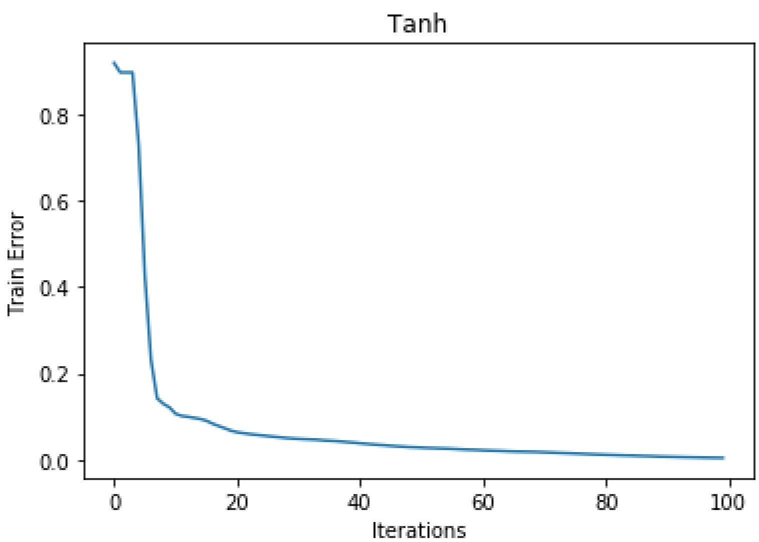
Plot for ReLu:



Plot for Linear:



Plot for Tanh:



Sklearn ReLu accuracy on training set: 0.9979166666666667

Sklearn ReLu accuracy on test set: 0.981

Sklearn linear accuracy on training set: 0.93005

Sklearn linear accuracy on test set: 0.919

Sklearn Sigmoid accuracy on training set: 0.9760666666666666

Sklearn sigmoid accuracy on test set: 0.9648

Sklearn tanh accuracy on training set: 1.0

Sklearn tanh accuracy on test set: 0.9753

Observation

There is not a huge difference between the Sklearn models and self-implemented models. The difference varies from model to model. This difference arises from the fact that the Sklearn by default uses Adam's optimizer while we are using Stochastic Gradient Descent. If we continue the SGD till convergence, the accuracies would be similar.

Question2

I have used a convolution layer with **kernel_size = 5, stride = 1** and padding = 2 followed by BatchNorm2d with num_features = 16 which is then followed by a ReLu layer followed by **Max pool**. Again convolution with **kernel_size = 5** is taken followed by BatchNorm2d with num_features = 32 which is then followed by a ReLu layer followed by **Max pool**. The 2nd layer has 32*7*7 neurons which are then fully connected with the output layer of 10 neurons. Hence there are 5 hidden layers - 2 CONV, 2 POOL, and 1 FC.

Accuracy on training set: 0.9701

Accuracy on test set: 0.8987

Confusion matrix of the training set:

```
[[5720    1   15   28    2    0  234    0    0    0]
 [   0 5997    0    3    0    0    0    0    0    0]
 [  31    0 5612   17  101    0  239    0    0    0]
 [  16    1    4 5884   26    0   69    0    0    0]
 [   2    0   75   31 5632    0  260    0    0    0]
 [   0    0    0    0    0 5999    0    1    0    0]
 [  71    0   31   24   22    0 5852    0    0    0]
 [   0    0    0    0    0    0    0 5997    0    3]
 [   1    0    0    0    0    0    2    0 5997    0]
 [   0    0    0    0    0    1    0   28    0 5971]]
```

Confusion Matrix of testing set:

```
[[821   0  19  17   2   2 136   0   3   0]
 [  1 986   0   8   2   0   0   0   3   0]
 [ 15   0 817   8  54   0 104   0   2   0]
 [ 22   3  10 890  34   0  39   0   2   0]
 [  3   0  50  20 821   0 104   0   2   0]
 [  0   0   0   0   0 981   0  16   1   2]
```

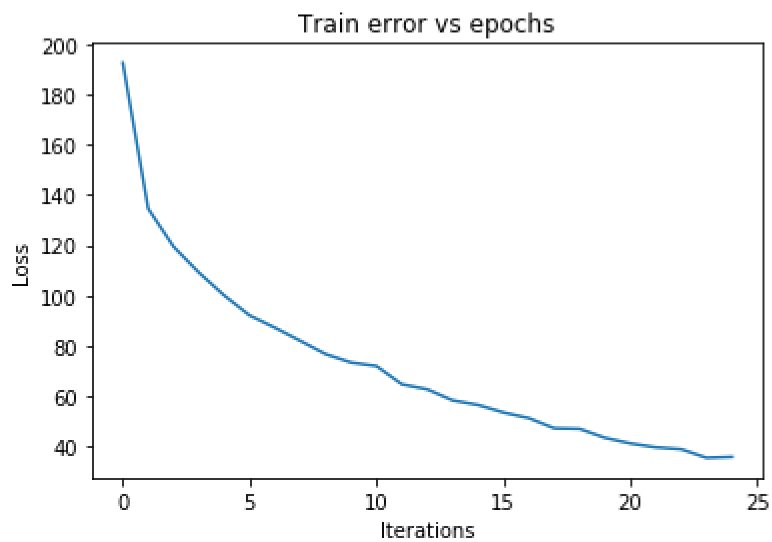
[89   1 37 22 40   0 807   0   4   0]

[0   0   0   0   0   6   0 979   0 15]

[2   0   0   5   4   2   5   2 979   1]

[0   0   0   0   0   9   1 28   0 962]]

The Training Loss Vs Epochs Curve:



The Test Loss Vs Epochs Curve:

The output of the last FC layer have been extracted from the CNN and fed directly into Kernelized SVM. The RBF is chosen bacause the RBF kernel trasnforms the input to infinite dimension in order to make it separable.

<u>Using RBF kernel SVM on the feature vector:</u>

accuracy on training = 0.98425

Accuracy on testing = 0.9086

Confusion matrix of the training set:

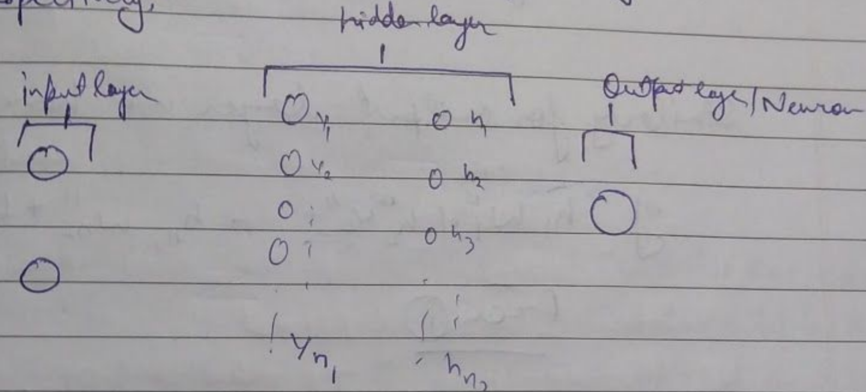
```
[[5851    0   22   22    1    0  104    0    0    0]
 [   0 5997    0    3    0    0    0    0    0    0]
 [  21    0 5814   13  112    0   40    0    0    0]
 [  23    1    4 5940   22    0   10    0    0    0]
 [   4    0   90   29 5821    0   56    0    0    0]
 [   0    0    0    0    0 5999    0    1    0    0]
 [ 152    0   91   37   65    0 5654    0    1    0]
 [   0    0    0    0    0    1    0 5990    0    9]
 [   1    0    0    1    0    0    0    0 5998    0]
 [   1    0    0    0    0    1    0    7    0 5991]]
```

Confusion Matrix of testing set:

```
[[854   0  18  19   2   1 101   0   5   0]
 [  2 984   2   9   0   0   2   0   1   0]
 [ 15   0 859  11  54   0  61   0   0   0]
 [ 14   2   8 907  33   0  34   0   2   0]
 [  1   0  56  26 863   0  53   0   1   0]
 [  0   0   0   0   0 977   0  20   0   3]
 [102   1  62  36  66   0 725   0   8   0]
 [  0   0   0   0   0   9   0 975   0  16]
 [  4   0   4   3   1   3   4   2 979   0]
 [  0   0   0   0   0   8   1  28   0 963]]
```

The accuracy obtained is almost similar. Generally, if we feed a pixel image to SVM after flattening we should expect a bad accuracy. But in this case, the input features are coming from the FC layer of CNN, and hence some features have been extracted by the convolutional layers. So, the data is already corrected for particular features. Then SVM is able to classify them much better than is used to. This makes it a little better model than CNN itself.

(3) Let us assume a general neural network with 2 hidden layers with say n_1 and n_2 neurons respectively.



$\cancel{y_1 = w_{11} i_1 + w_{12} i_2 + \dots + w_{n_1}}$

$$y_1 = w_{11} i_1 + w_{21} i_2 + b_1$$

For general neuron: $y_j = w_{j1} i_1 + w_{j2} i_2 + b_j$

where w_{ji} is weight of j neuron in hidden layer 1 and i is input neuron.

Similarly

$$h_1 = w'_{11} y_1 + w'_{12} y_2 + \dots w'_{1n_1} y_{n_1} + b'_1$$

$$= w'_{11} \cdot [w_{11} i_1 + w_{21} i_2 + b_1] + \dots + w'_{1n_1} [w_{1n_1} i_1 + w_{2n_1} i_2 + b_{n_1}]$$

$$= i_1 \underbrace{W_1}_{\text{say}} + i_2 \underbrace{W_2}_{\text{say}} + \underbrace{B_1}_{\text{say}} \quad \text{--- } (1)$$

Similarly for output layer we have a form of $h_1 W_1'' + h_2 W_2'' + \dots h_{n_2} W_{n_2}'' + b_{out}$

from (1)

$$\text{Output} = i_1 \left(\sum_{j=1}^{n_2} W'_{j1} W^{o}_{j1} \right) + i_2 \left(\sum_{j=1}^{n} W'_{j1} W_{j2} \right) + \sum_{i=1}^{n_2} b_i$$

$$= W_1 i_1 + W_2 i_2 + B$$

which is equivalent or similar to the output of a single layer perceptron which we already know, can't be used to classify a XOR problem. Hence we can't use a linear activation to classify XOR table.

(4) Different parts which ~~made~~ make up a Deep Convolution Neural Network are:-

1) Convolution Layer
2) Pooling layer
3) Fully connected layers

1) Convolution Layer: This layer is use to extract certain features present in input data which are in form of images. An appropriate kernel or filter is used to extract the features. This filter is then iterated over the complete image to extract those features from the input image. ~~CNN~~ Convolutionary layers are used to extract certain features even before that is passed through the fully connected layers. Hence, they increase the accuracy of a neural network. It also reduces the size of input, therefore reducing computation. Output image has ~~size~~ size $\frac{N-K}{S}+1$

where N is input size, K is kernel size, S is stride.

Convolution is defined as $S(i,j) = (I*K)(i,j)$

$$= \sum_m \sum_n \underset{\text{Input}}{I(m,n)}\ \underset{\text{Kernel}}{K(i-m, j-n)}$$

Date
Page

2) Pooling layer : This layer is used after convolutionary layer. Main aim of this layer is to down sample further the output of convolution layer. It reduces the spatial size thus reducing parameters and computation. Generally a 2×2 pooling kernel with stride 2 is used with Max pooling operation.

1) Max pooling : It takes the max value of a sub-matrix of size of kernel.

2) Averaging pool : It takes average of local submatrix defined by kernel.

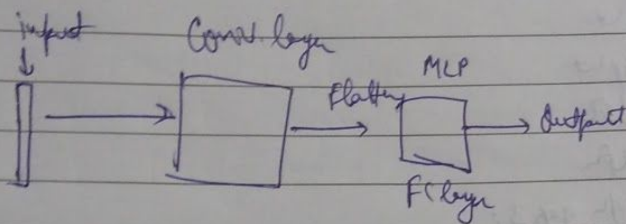
3) Median pooling : It takes the median pixel value of local submatrix of image.

One more advantage of this is that it makes the neural network transitionally invariant which means slight translation of the local features of the image won't affect the classification of the image.

3) Fully Connected layers : The function of

This layer is take the output of convolution or pooling layer and classify the images. The outputs of the above 2 layers are flattened and then fed into fully connected layers and the output give us the ~~output~~ classified output. The FC layer contains an output layer in itself which predicts the class of an image. This part is exactly similar to a traditional neural network

(5) CNN having 2 hidden layers



MLP has 2 FC layers, 1 hidden and 1 output.

```
° flatten (2D matrix) {
    arr = []
    for i in range len(2D matrix):
        for j in range len(2D-matrix[i]):
            arr.append(2D matrix[i][j])

    return arr.

init_kernel(int t):
        ↳ initialize kernel of size txt
init_hidden_layer( ) →
conv_layer(input image):
    kernel = init_kernel(3).
    h = image.height
    w = image.width
    # stride = 1
H = init_hidden_layer(h-3+1, w-3+1).
    sharing(input, H)

Apply_conv_2D(input, kernel size):
```

$$S(i,j) = \sum_m \sum_n I(m,n)\, K(i-m, j-n)$$ [Use for loops]

→ kernel size

pooling (l_1, l_2) :

$H_1 = L_1$. height
$H_2 = L2$. height
$W_1 = L_1$. width
$W_2 = L_2$. width
for i in H_1 with gap 3:
    for j in H_2:
        connect (l_1, l_2, i, j)