CSE-574 Artificial Neural Networks

Anand anand6@buffalo.edu
Jing Chen jchen445@buffalo.edu

Handwritten Digit Classification:

We use MNIST dataset which has train and test data with labels of digits from 0 to 9. We divide the dataset into train, validation and test datasets. As the inputs are images it has many features, in order to eliminate few of the features we used elements which has same value for all the observations are removed there by reducing the feature space. And each value of feature varies from 0 to 255 it's the pixel intensities in the image.

Feed Forward:

A mechanism to test the given input and with initial weights we pass thru layers using activation function and then derive the result at the output layer.

Back Propagation:

After feed forward we calculate the error and we calculate gradient of weight contributing to the error and modify the weights accordingly at the learning rate.

We proceed this with maximum number of iterations and then find global minimum which is the optimal value of weights for the given neural network. Here we use gradient descent implementing functions and pass it to the *minimize* an optimize function in scipy library.

Regularization in Neural Networks:

Regularization parameter in the model we build is lambda which controls the overfitting of data by penalizing the weights there by increasing the accuracy of the test data. Lambda parameter is complex parameter allows to control bias and variance trade off. For choosing optimal lambda we have to train and test the neural network and see which lambda value range results in better accuracy of validation and test data.

Finding optimal lambda:

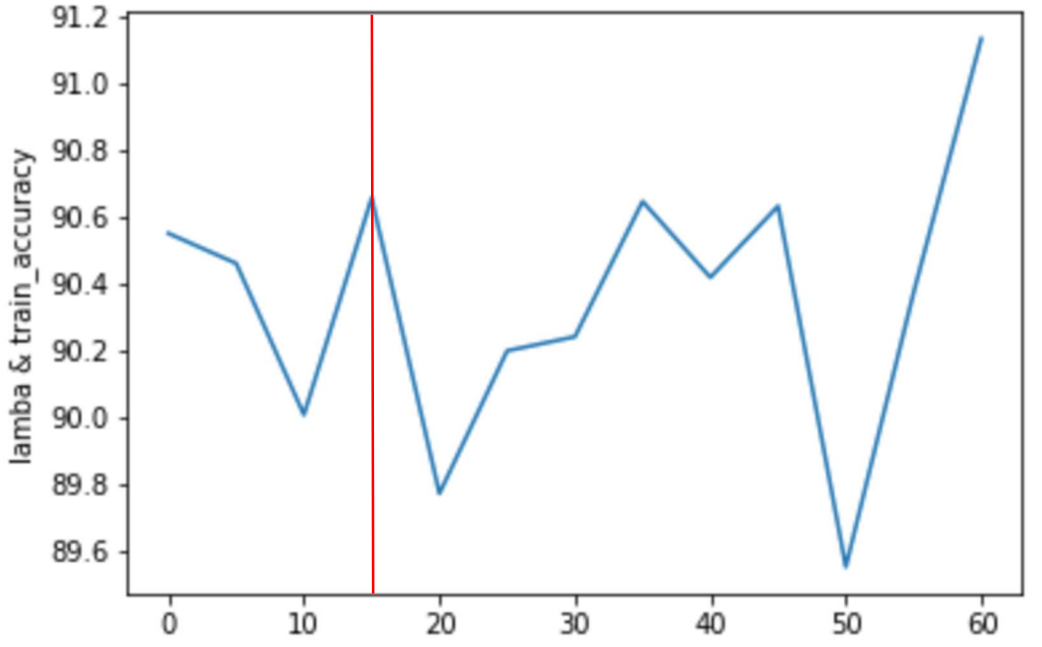
Neural networks are complex and the accuracy of test data depends on the different factors number of hidden layers and number of hidden units and complex parameter lambda. More hidden layers may result in overfitting of data and less hidden layers may result in underfitting of data. Overfitting of data can be controlled using complex parameter lambda. So now we have to simulate the neural network for different values and find the optimal value for lambda and number of hidden layers.

Testing the model with different lambda with hidden units = 50:

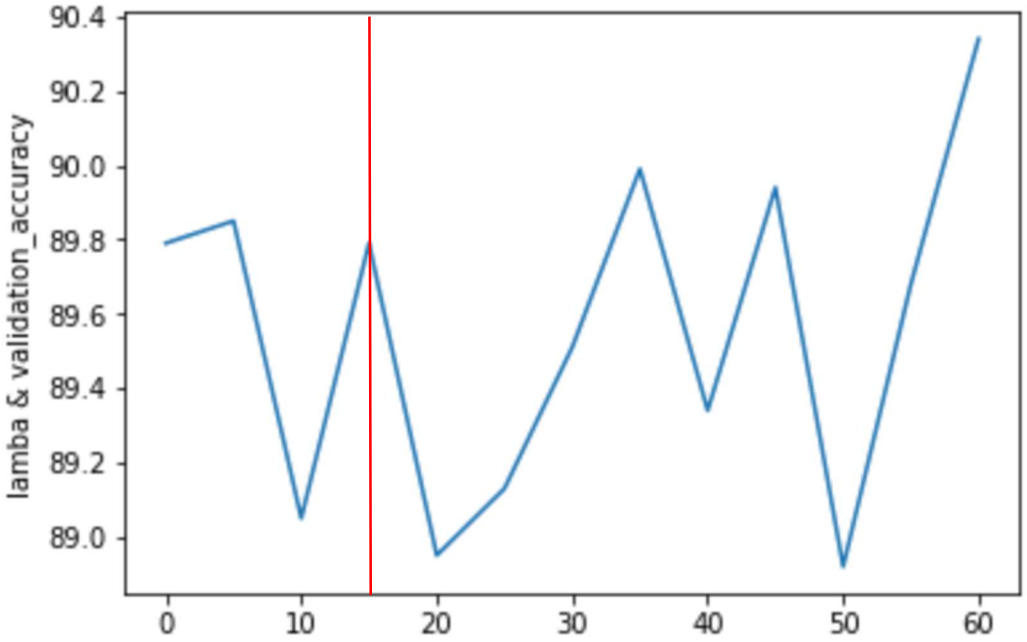
With lambda = 0 hidden nodes = 50:
Training set Accuracy: 90.538%
Validation set Accuracy: 89.91%
Test set Accuracy: 90.29%

Varying lambda from 0 to 60 with increment of 5:

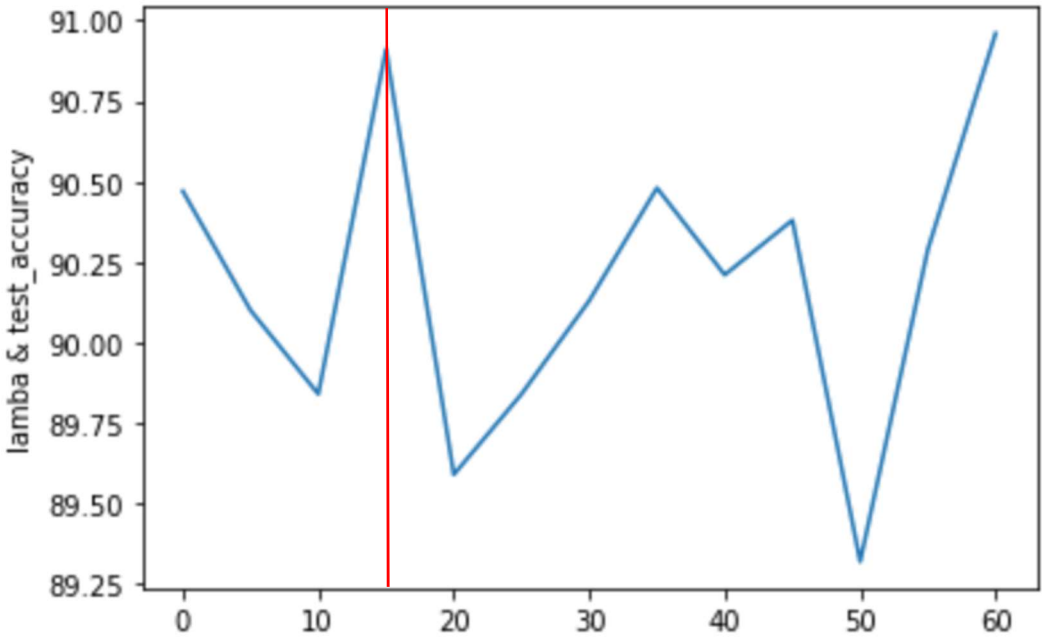
Training Data:



Validation Data:



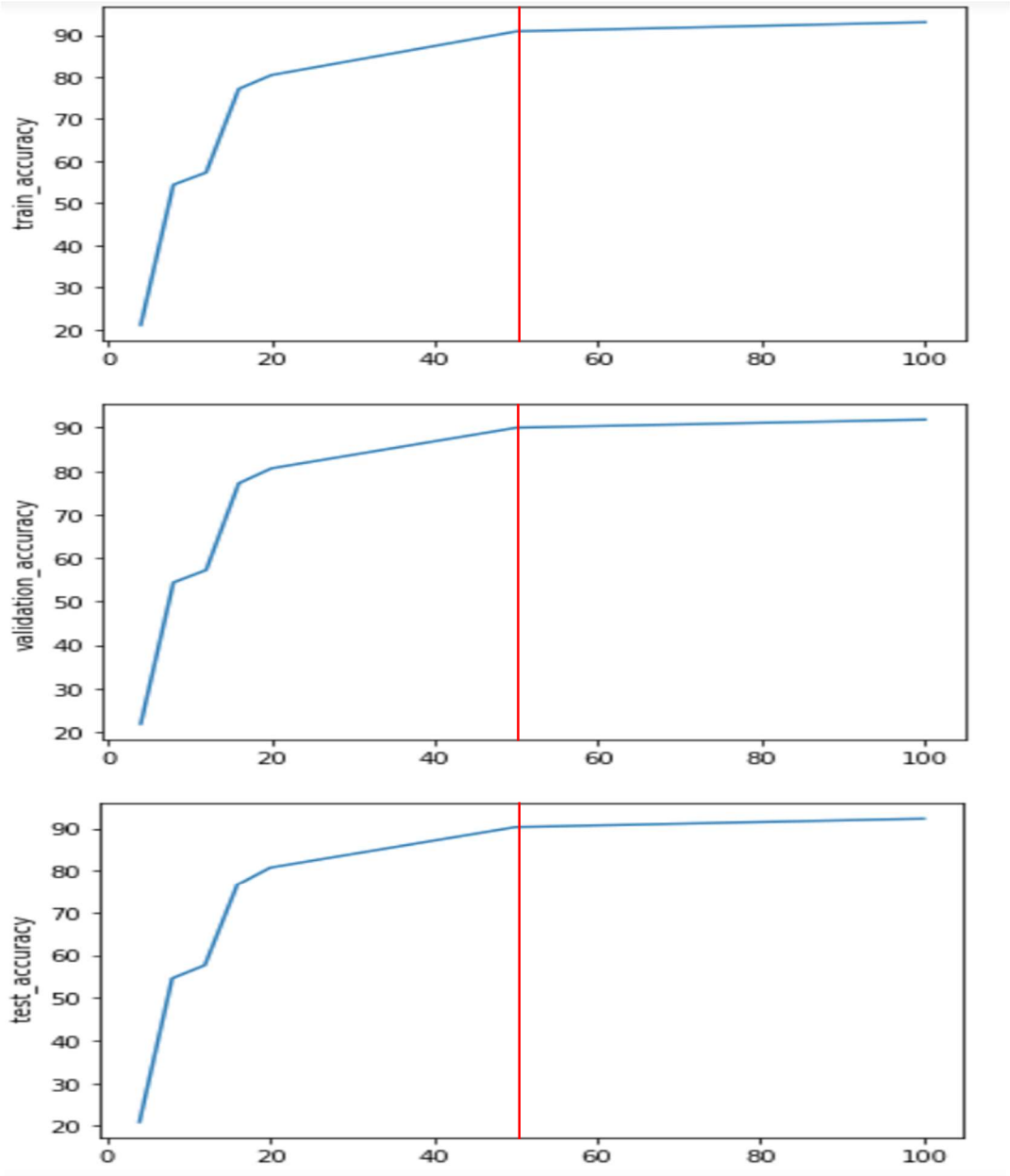
Test Data:



By analyzing all the three plots, we choose lambda based on test and validation data as lambda parameter is used to control the overfitting of training data and increase accuracy in testing data. Here in the plots we observe that validation and test data have greater accuracy when lambda is 15. So, we choose the lambda as 15 which is optimal for the current model.

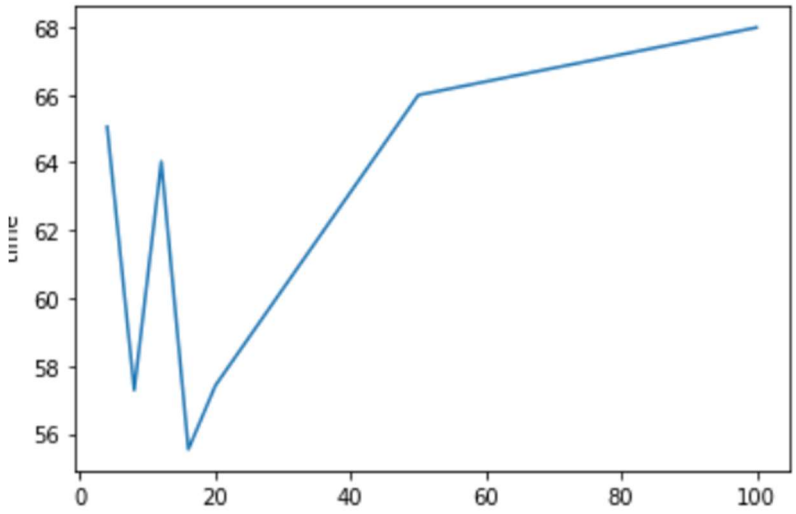
Finding optimal number of hidden nodes:

Lambda is set to 15 which is the optimal value for the model
Now we test the model with different number of hidden units = [4, 8, 12, 16, 20, 50, 100]



From the plots we observe that the model does not improve much after increasing the hidden nodes more than 50. Its rate of accuracy gets merely constant after 50 hidden nodes. So, with this we conclude 50 is the optimal number of hidden nodes for the given model.

Training time for different number of hidden nodes:



From the plot we can infer that the training time varies with number of hidden nodes as the time taken for training increases exponentially with increase in number of hidden nodes. Considering the time taken for training and accuracy, we choose the number of hidden nodes as 50 and lambda as 15.

Conclusion:

We choose optimal lambda as 15 and number of hidden nodes as 50.

Accuracy of classification on Handwritten Digits:

Training set Accuracy: 90.834%
Validation set Accuracy: 89.99%
Test set Accuracy: 90.28%

Accuracy of classification on CelebA:

Training set Accuracy: 86.17%
Validation set Accuracy :85.21%
Test set Accuracy: 86.11%

TensorFlow for Deep Neural Networks:

Tensor Flow

Type	Hidden Layers	Hidden Nodes per layer	Time Taken to train(in seconds)	Test Accuracy
Single Layer NN	1	256	106	86.11
Deep NN	2	256	132	81.22
Deep NN	3	256	133	80.01
Deep NN	5	256	169	74.49
Deep NN	7	256	195	73.84

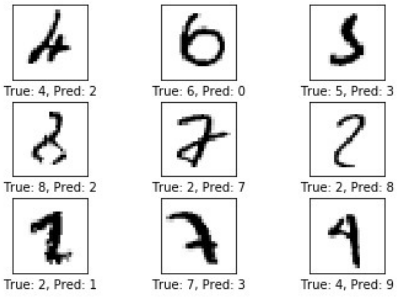
TensorFlow library helps to build deep neural networks easily and also it learns the weights at learning rate using batch there by making the process faster. But adding more layers and increasing the depth of neural networks increases learning of all features well there by resulting in overfitting . Such deep layered neural networks tend to behave well on training data but not on test data.

From the results we can infer that, as we increase the number of hidden layers the accuracy of test data decreases. We can control this overfitting by adding regularization parameter using *tf.keras.regularizers.Regularizer* which applies penalties on the weights of layers during optimization by which we can achieve the improve test accuracy in deep neural networks. Other way is having more data for deeper neural networks.

Tensorflow – Convolutional Neural Networks:

We will use CNN over MNIST dataset and see how it behaves:
Size of:
- Training-set: 55000
- Test-set : 10000
- Validation-set: 5000

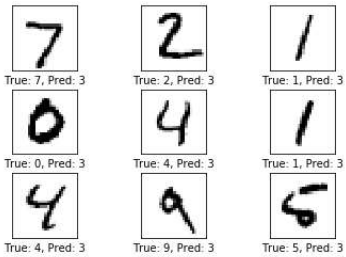
Accuracy on Test-Set: 9.5% (946 / 10000)
Example errors:



Confusion Matrix:

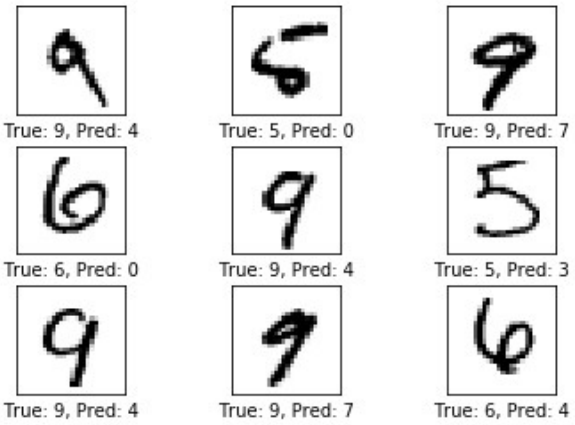
```
Confusion Matrix:
[[   0    0    0  974    0    0    0    6    0    0]
 [   0    0    0 1135    0    0    0    0    0    0]
 [   2    0    0 1021    1    0    0    8    0    0]
 [   3    0    0  944    0    0    0   63    0    0]
 [   7    0    0  891    1    0    0   83    0    0]
 [   0    0    0  877    0    0    0   15    0    0]
 [   1    0    0  954    1    0    0    2    0    0]
 [   0    0    0 1027    0    0    0    1    0    0]
 [   1    0    0  968    2    0    0    3    0    0]
 [   0    0    0 1006    1    0    0    2    0    0]]
```

Number of iterations: 1
Optimization Iteration: 1, Training Accuracy: 6.2%
Time usage: 0:00:01
Accuracy on Test-Set: 9.0% (900 / 10000)



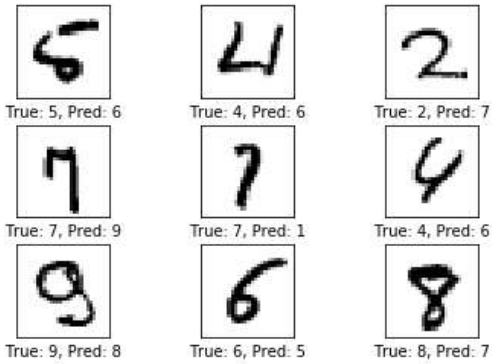

```
Confusion Matrix:
[[   0    0    0  980    0    0    0    0    0    0]
 [   0    0    0 1134    0    0    0    0    0    1]
 [  14    0    0 1003    2    0    0    2    0   11]
 [  23    0    0  888    1    0    0    3    0   95]
 [  29    0    0  914   12    0    0   26    0    1]
 [   3    0    0  880    3    0    0    2    0    4]
 [   3    0    0  948    6    0    0    0    0    1]
 [   0    0    0 1026    1    0    0    0    0    1]
 [   2    0    0  966    4    0    0    1    0    1]
 [   2    0    0 1001    6    0    0    0    0    0]]
```

*Number of iterations:*99
Time usage: 0:00:05
Accuracy on Test-Set: 63.0% (6297 / 10000)



```
Confusion Matrix:
[[ 949    2    0   24    0    0    1    1    3    0]
 [   0 1114    1   19    0    0    0    0    1    0]
 [ 210   12  560  217   12    0    9   11    1    0]
 [  21   14    4  957    0    0    0   10    3    1]
 [  23   60    6   20  839    0    8   18    4    4]
 [ 133   87    3  494   19   48    0   11   88    9]
 [ 269   63   38   65   17    0  503    1    2    0]
 [  16   82   16   51    3    2    0  845   13    0]
 [  84   79    1  452    7    2    0   12  335    2]
 [  51   71    0   81  373    0    2  250   34  147]]
```

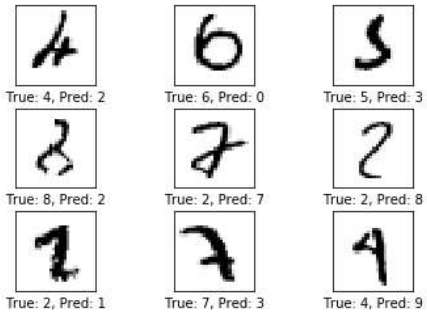
Number of iterations: 900
Time usage: 0:00:48
Accuracy on Test-Set: 93.2% (9317 / 10000)




```
Confusion Matrix:
[[ 957    0    0    1    0    3   13    1    5    0]
 [   0 1110    2    3    0    1    5    0   14    0]
 [  14    2  917   26   13    0   13   15   30    2]
 [   1    3    5  959    0    7    1    9   19    6]
 [   0    4    3    1  906    0   20    1    5   42]
 [   8    3    0   33    4  793   20    1   25    5]
 [  10    3    1    1    9   11  920    1    2    0]
 [   1   13   27    9    4    1    1  924    3   45]
 [   7    3    1   28    6   11    8    8  893    9]
 [  11    5    3   12   17    2    1    9   11  938]]
```

Number of iterations: 9000

Time usage: 0:08:48
Accuracy on Test-Set: 98.9% (9886 / 10000)
Example errors:




```
Confusion Matrix:
[[ 974    0    0    0    0    1    2    1    2    0]
 [   0 1133    1    0    0    0    1    0    0    0]
 [   1    2 1021    1    1    0    0    2    4    0]
 [   1    0    0 1003    0    3    0    1    2    0]
 [   0    1    2    0  972    0    1    0    0    6]
 [   2    0    0    4    0  881    2    0    1    2]
 [   4    2    0    0    1    1  950    0    0    0]
 [   0    3    8    1    0    0    0 1008    1    7]
 [   6    1    2    1    1    1    1    2  956    3]
 [   3    5    0    1    6    1    0    1    4  988]]
```

With 9000 iterations and with time 8 mins we could achieve test accuracy of 98.9% on MNIST data set. Confusion matrix explains clearly how many observations are classified wrongly and the misclassification class as well. Its easily interpretable.

In neural networks we analyze features of whole image and learn parameters and increase depth of hidden layers to learn all the features and predict. But in convolutional neural networks we read the same image using sampling (using filters of specific size with strides and padding) and then apply pooling and these entire layers are then connected to output layer. This is a complex neural network and is called convolutional neural network. So convolutional neural network learns parameters effectively there by with increase in number of iterations on sampling and optimization the test accuracy was nearly 99%. CNN is efficient.