ELL409 - Assignment 3

Neural Networks

E Spandana 2020EE10489
Ekansh Singh 2020EE10490
Rashee Agarwal 2020EE10539

09/11/2022

Contents

1 Neural Networks

- Neural networks work on the lines of the way the human brain function, involving layers of neurons connected together with transmit the information and make the combinations
- Neural networks essentially are layers of computations involving linear operation on the input layer and application of an activation function
- A deeper neural network has more number of layers with more number of nodes and is capable of more complex analysis
- In order to create a neural network, a lot of different parameters are to be chosen. The number of layers, the number of neurons/nodes per layer, no of iterations, the type of regularization and many more factors.
- Neural networks find a much wider application in field with complex inputs such as image recognition which is performed using Convolutional Neural Networks and Natural Language Processing which uses Recurrent Neural Network

The basic structure of neural network can be represented as:

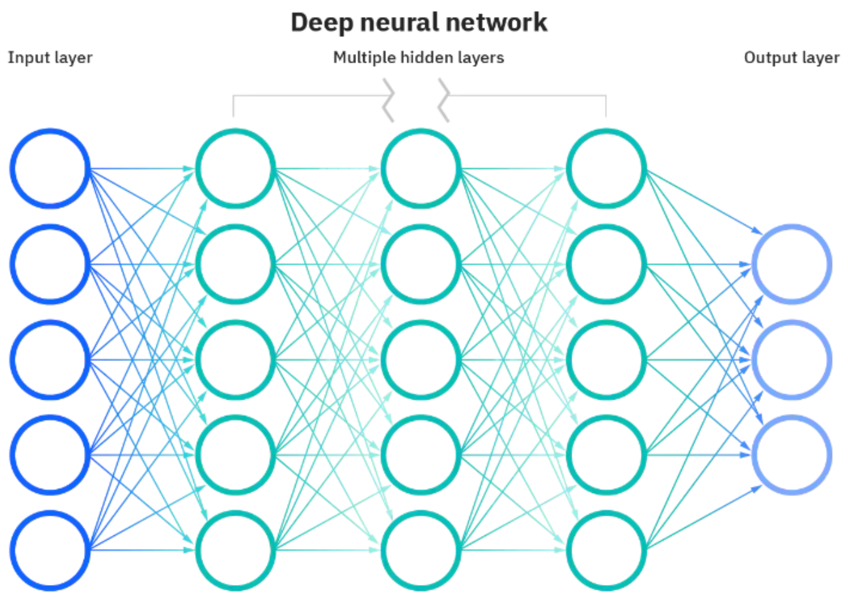


Figure 1: Depecting a neural network

2 Overview

In this assignment, we will be creating a neural network model from scratch using three basic steps i.e.

- Creating and testing the neural network
- Parameter Optimisation
- Performance Estimation

3 Creating and testing the Neural Network

In the dataset we are given the input features in the first 10 columns and the last column contains the feature that has to be predicted i.e. if the house price above the median or not? We will use some steps to create the neural network and the same is described in the following portion of the section.

3.1 Importing the data

3.1.1 Aim

We need to import the data as an initial step to start building our network.

3.1.2 Solution

We initially imported the data i.e. the csv file using pandas library in python and later we stored it in the form of numpy arrays

3.2 Normalization

3.2.1 Aim

We need to normalize the given dataset

3.2.2 Solution

For normalizing the data, we make the data mean-centered and then divide it by the standard deviation of the data as this gives a better result.

$$Normalized\ data = \frac{Data - Mean(Data)}{Standard\ Deviation(Data)} \tag{1}$$

We normalize the data to bring all of it to a common scale while maintaining the distribution and well as the ratio in the data.

3.3 Initializing the network parameters

3.3.1 Aim

We need to initialize the network parameters as well as the number of hidden layers and the activation function

3.3.2 Solution

- The activation function that we used in our network is hyperbolic tan i.e. *tanh*
- We did not fix the number of nodes and layers to make it more general, we define an integer array $layers$ whose size basically tells the numbers of layers and the data inside it tells us the number of nodes in each layer.
- In our network we took the size of that array as 6 and the elements in the array as 10, 32, 64, 128, 256 and 1. The initial number of nodes is 10 as the number of features of the dataset are 10 and 1 representing the number of output features.

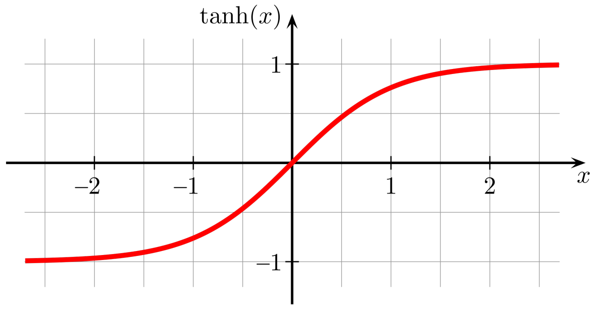


Figure 2: Graph for hyperbolic tan

3.4 Initialising weight and bias of the network

3.4.1 Aim

We need to initialise the weight and bias of our network.

3.4.2 Solution

We initialised the weight parameters randomly using the inbuilt random function in numpy library. The bias parameters were all set to zeroes.

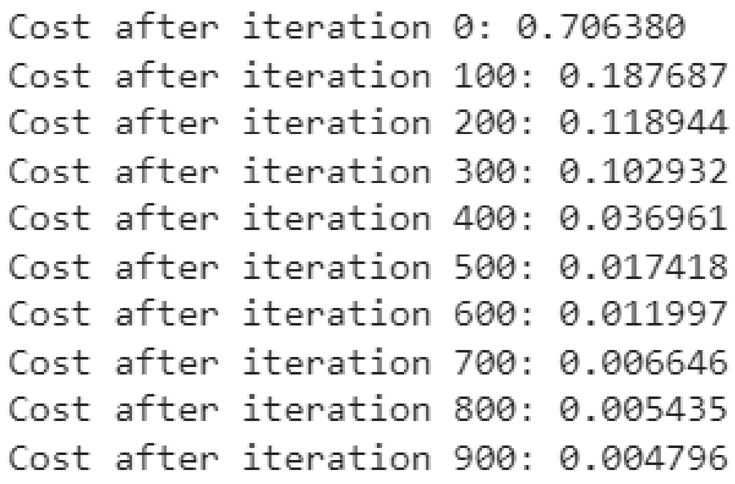
3.5 Training the network using Training data and Testing Targets

3.5.1 Aim

Using the above methods we need to build a neural network

3.5.2 Solution

We split our data set into training, validation and testing sets in the ratio 72:18:10 respectively and use the above mentioned methods and build our first version of the neural network.



```
Cost after iteration 0: 0.706380
Cost after iteration 100: 0.187687
Cost after iteration 200: 0.118944
Cost after iteration 300: 0.102932
Cost after iteration 400: 0.036961
Cost after iteration 500: 0.017418
Cost after iteration 600: 0.011997
Cost after iteration 700: 0.006646
Cost after iteration 800: 0.005435
Cost after iteration 900: 0.004796
```

Figure 3: Cost Updation with the iterations

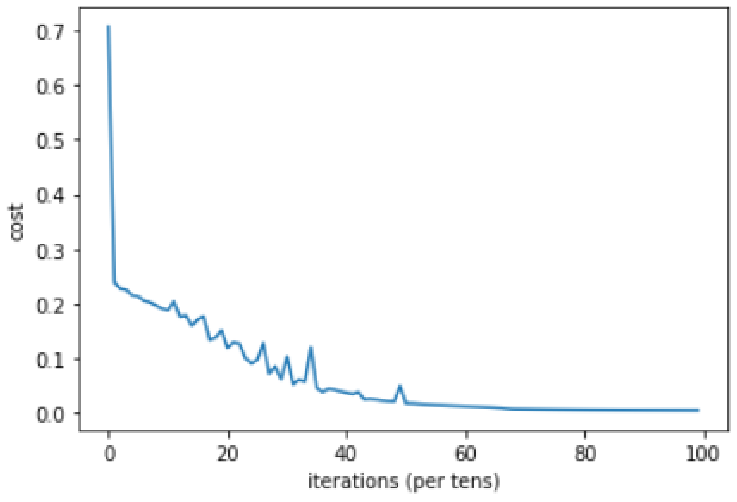
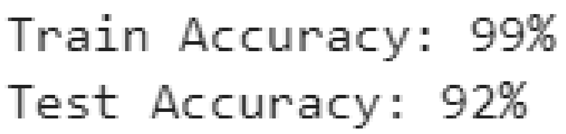


Figure 4: Plot for cost with the iterations



```
Train Accuracy: 99%
Test Accuracy: 92%
```

Figure 5: Training and Testing accuracy

4 Parameter Optimisation

An important process in neural networks is parameter optimization. First, select a performance measure which will be used to compare the performance of the different parameters on the validation set. We used batch gradient descent with momentum and then optimized the parameters and judged the network efficiency in term of percent accuracy.

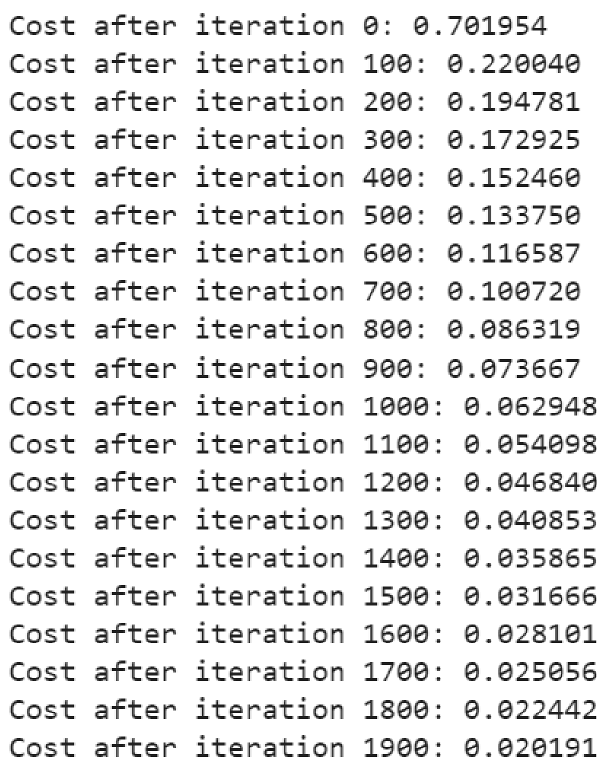
4.1 Selection of network architecture and activation function

4.1.1 Aim

First select a network architecture and an activation function that you believe is a reasonable starting point. Explain your motivation for selecting this architecture and activation function. Define a stopping criterion, a weight initialisation method and a momentum and learning rate update schedule

4.1.2 Solution

- Here, we choose to use tanh as the activation function as the output of the tanh function is zero-centered. Hence, we can map the output values as strongly positive, negative, or neutral. The value of tanh lies between [-1,1]. So if we use this function in hidden layers, the mean of the hidden layer comes out to be 0 or very close to 0, which helps in centering and learning the next layer much easier.
- We used a fixed stopping criterion here i.e. 1000 epochs because that led to significant reduction in the error as shown.



```
Cost after iteration 0: 0.701954
Cost after iteration 100: 0.220040
Cost after iteration 200: 0.194781
Cost after iteration 300: 0.172925
Cost after iteration 400: 0.152460
Cost after iteration 500: 0.133750
Cost after iteration 600: 0.116587
Cost after iteration 700: 0.100720
Cost after iteration 800: 0.086319
Cost after iteration 900: 0.073667
Cost after iteration 1000: 0.062948
Cost after iteration 1100: 0.054098
Cost after iteration 1200: 0.046840
Cost after iteration 1300: 0.040853
Cost after iteration 1400: 0.035865
Cost after iteration 1500: 0.031666
Cost after iteration 1600: 0.028101
Cost after iteration 1700: 0.025056
Cost after iteration 1800: 0.022442
Cost after iteration 1900: 0.020191
```

Figure 6: Cost with iterations on applying the momentum method

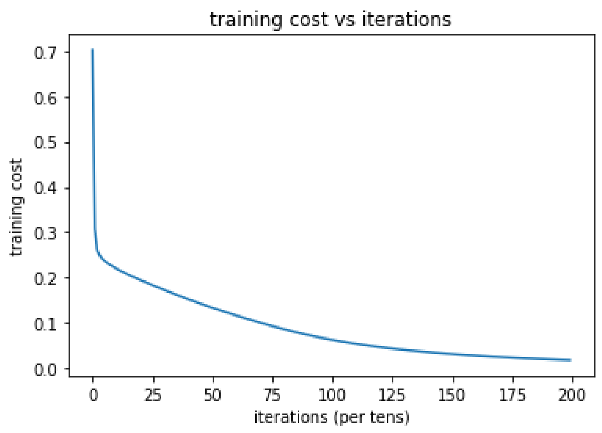


Figure 7: Plot of Cost and iterations with the momentum method

- We experimented with both random and He initialization and found out that He initialization turned out to be giving a way more higher accuracy.
  In He initialization, while randomly declaring the W matrices those values are multiplied with a factor of

$$\sqrt{\frac{1}{sizeoflayer^{[l-1]}}}$$

- The learning rate update schedule will be such that for every 15 iterations we would update the learning by multiplying it with a constant k where

$$0 < k < 1$$

4.2 Optimizing the initial learning rate (disable regularization)

4.2.1 Aim

Optimize the initial learning rate (disable regularization). Explain how you found a good initial value for learning rate. Save the plot of the training and validation loss with iterations.

4.2.2 Solution

We took some values between 0.01 and 4 and observe the accuracy for each of the learning rates and plot the same to get the best initial learning rate i.e 1.2 here and then we update it in the further steps.

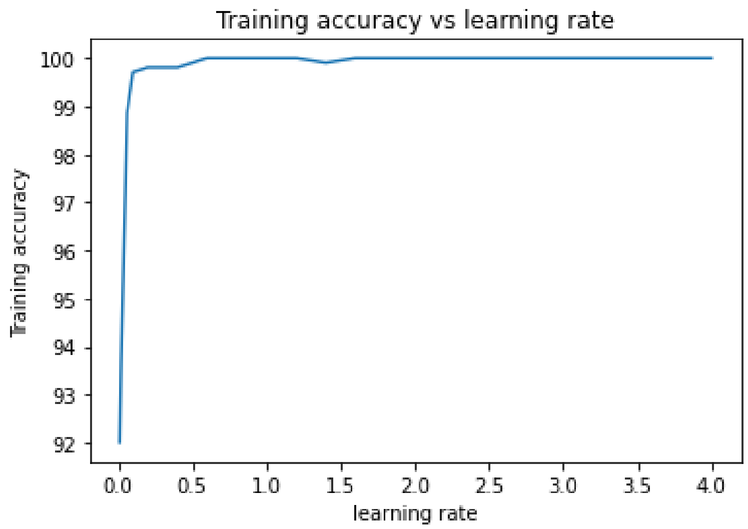


Figure 8: Accuracy vs the Initial Learning Rate

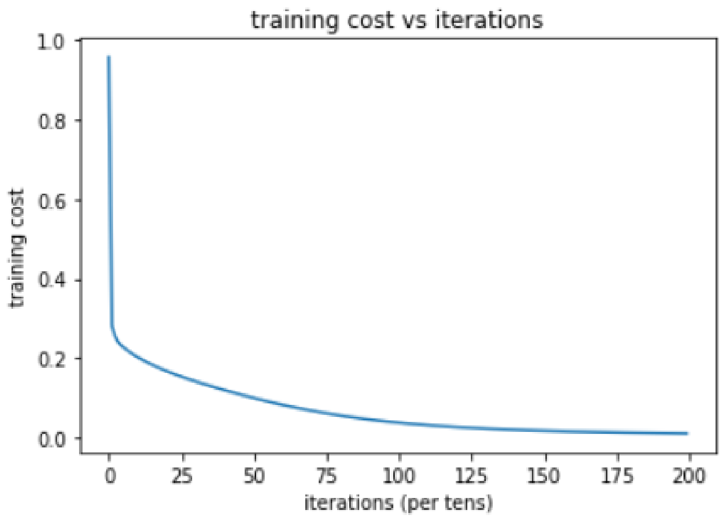


Figure 9: Training loss vs iterations

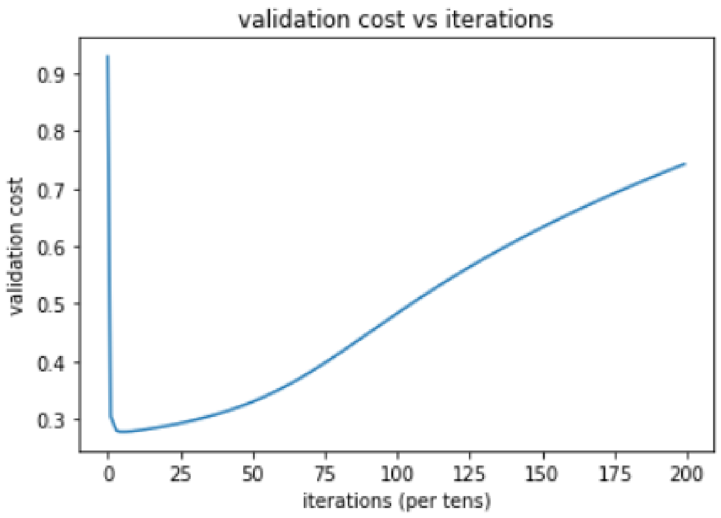


Figure 10: Validation loss vs iterations

4.3 Optimizing the learning rate update schedule

4.3.1 Aim

Optimize the learning rate update schedule. Include a table where you present the results on the validation set of the different learning rate schedules you have tested.

4.3.2 Solution

As mentioned earlier, we will have a constant k where $0 < k < 1$ and on every 15 iterations the learning rate is updated by multiplying with the constant k.
We tested out the same on the validation set for the best initial rate that was found out to be 1.2 in the last section.

Learning Rate Parameter	Accuracy
0.60	92.01
0.65	90.87
0.70	92.77
0.75	91.63
0.80	91.25
0.85	90.87
0.90	92.39
0.95	90.11
0.98	90.87
0.99	90.87
0.999	91.25

Table 1: Learning Rate Update Parameters and Accuracy

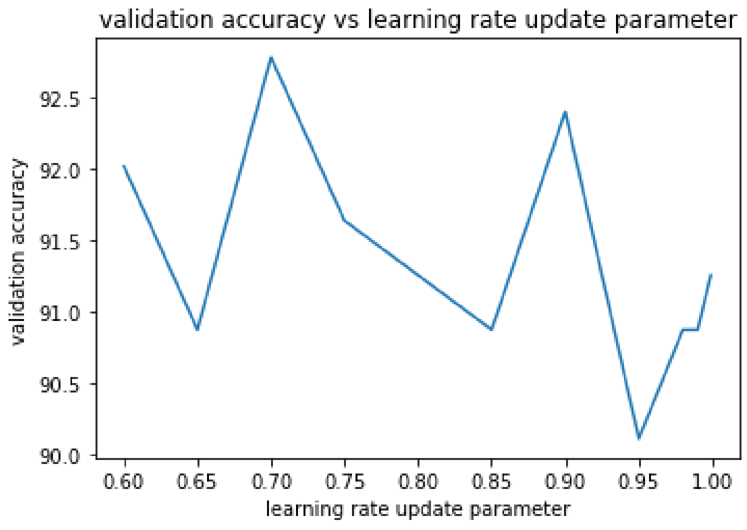


Figure 11: Accuracy vs the Learning Rate Update Parameter

We can see from the learning rate update parameter vs the accuracy that the accuracy is maximum for

$$k \approx 0.71$$

4.4 Using Dropout

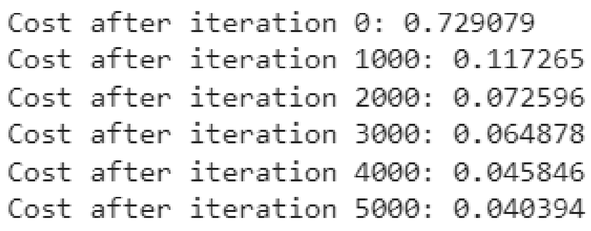
4.4.1 Aim

Use dropout and report if there is any improvement in the validation performance. Explain what changes you have made to the network and/or training procedure when you use dropout.

4.4.2 Solution

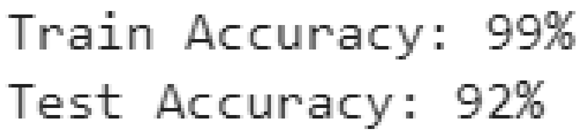
- Dropout is used to drop some of the nodes in the neural network, which do not have a significant contribution to the prediction, and thus this technique is useful to prevent the model from over fitting the dataset
- So when dropout was used with momentum along with he initialization we attained an almost same accuracy as the previous model though it required significantly higher number of iterations
- In the forward propagation step, which so ever node led to a contribution less than a defined threshold is dropped, and an equivalent change in back propagation algorithm was done
- Therefore, in the code the probability of keeping a node is given by *keepprob* and this parameter is set to 0.88, thus each node is dropped with a probability of 12%

Now we put forth some results when dropout is applied to our model



```
Cost after iteration 0: 0.729079
Cost after iteration 1000: 0.117265
Cost after iteration 2000: 0.072596
Cost after iteration 3000: 0.064878
Cost after iteration 4000: 0.045846
Cost after iteration 5000: 0.040394
```

Figure 12: Cost with iterations on applying the dropout method



```
Train Accuracy: 99%
Test Accuracy: 92%
```

Figure 13: Train and test accuracy of the dropout method

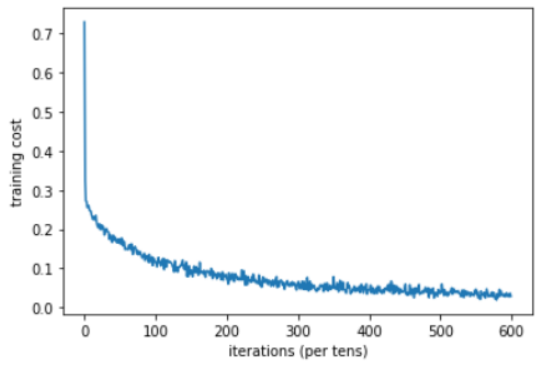


Figure 14: Training cost varying over iterations

Since there wasn't any drastic decrease in test accuracy we can conclude that the model wasn't over fitting the dataset much.

4.5 Regularization and their performance

4.5.1 Aim

Use two other types of regularization (any two you wish) and compare their performance with dropout. Explain which regularization parameters you optimized and present a plot which shows the performance on the validation set as a function of the parameters that needs to be optimized. Also discuss why regularization is needed.

4.5.2 Solution

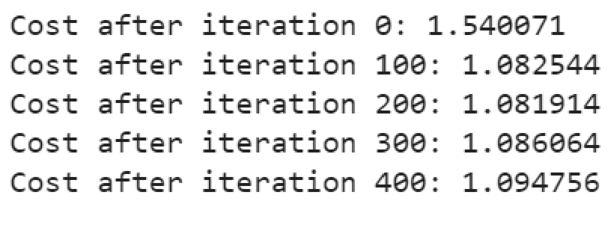
Regularization is needed to prevent the model from over fitting the dataset. This might lead to an extremely good train accuracy but the model might give a poor result on testing dataset, since the model fitted to all the deviation and errors present in train dataset. To prevent such scenario, regularization is done which might have comparatively less train accuracy but will have a better test accuracy and thus will lead to an overall more adapted model. We can use two types of regularization techniques that are Lasso(L1) and Ridge (L2) regularizations. Now we'll show the results we got by applying them on the network.

4.5.3 L1 Regularization

Lasso regression is used not only for reducing over fitting but also for feature selection because this technique can lead to features being neglected for prediction. We have the extra penalty factor λ multiplied with the sum of magnitudes of the weights.

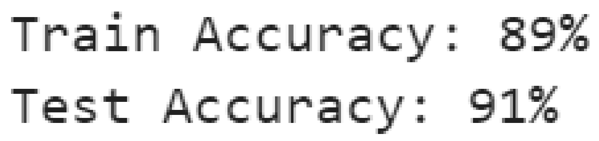
$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n,\mathbf{w}) - t_n)^2 + \lambda\sum_{j=0}^{M}|w_j|$$

In this expression we need to optimize the λ parameter of the regularization.
Now we put forth the results and comparisons we get from the model we formed.



```
Cost after iteration 0: 1.540071
Cost after iteration 100: 1.082544
Cost after iteration 200: 1.081914
Cost after iteration 300: 1.086064
Cost after iteration 400: 1.094756
```

Figure 15: Cost with iterations in L1 Regularization



```
Train Accuracy: 89%
Test Accuracy: 91%
```

Figure 16: Training and Testing Accuracy in L1 Regularization

We see a slight decrease in the accuracy in case of regularization than from the dropout case that could be because we might have tuned some parameters which were fitting very well and got disturbed post regularization.

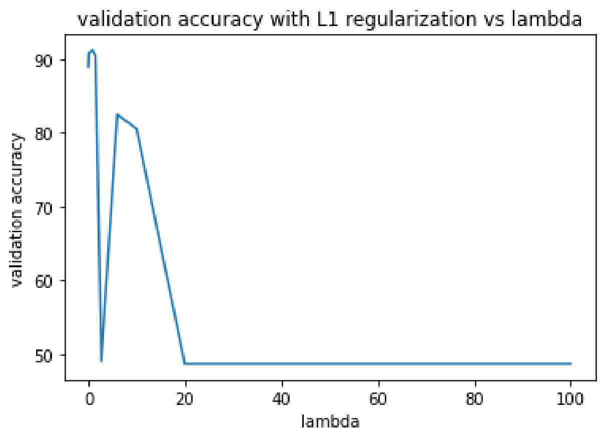


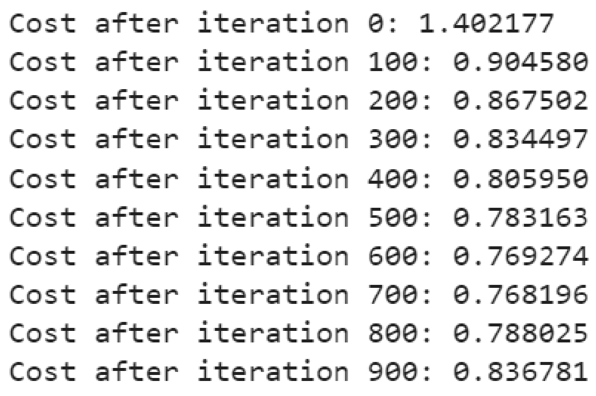
Figure 17: Variation of Validation Accuracy with the Regularization Parameter

4.5.4 L2 Regularization

In ridge regression just as in lasso regression the weights are being altered by adding penalty term, but here we multiply it with the sum of squares of the weights to avoid over fitting. Consequently, here we have a weights dependent term in the gradient of parameters. Higher the value of penalty term, the reduced are the chances of over fitting, but consequently there is a slight decrease in the accuracy.

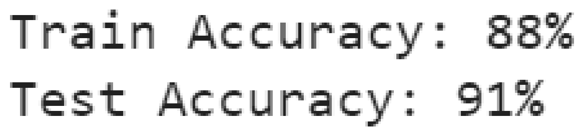
$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, \mathbf{w}) - t_n)^2 + \lambda \sum_{j=0}^{M} w_j^2$$

In this expression we need to optimize the λ parameter of the regularization.
Now we put forth the results and comparisons we get from the model we formed.



```
Cost after iteration 0: 1.402177
Cost after iteration 100: 0.904580
Cost after iteration 200: 0.867502
Cost after iteration 300: 0.834497
Cost after iteration 400: 0.805950
Cost after iteration 500: 0.783163
Cost after iteration 600: 0.769274
Cost after iteration 700: 0.768196
Cost after iteration 800: 0.788025
Cost after iteration 900: 0.836781
```

Figure 18: Cost with iterations in L2 Regularization



```
Train Accuracy: 88%
Test Accuracy: 91%
```

Figure 19: Training and Testing Accuracy in L2 Regularization

We see a slight decrease in the accuracy in case of regularization than from the dropout case that could be because we might have tuned some parameters which were fitting very well and got disturbed post regularization.

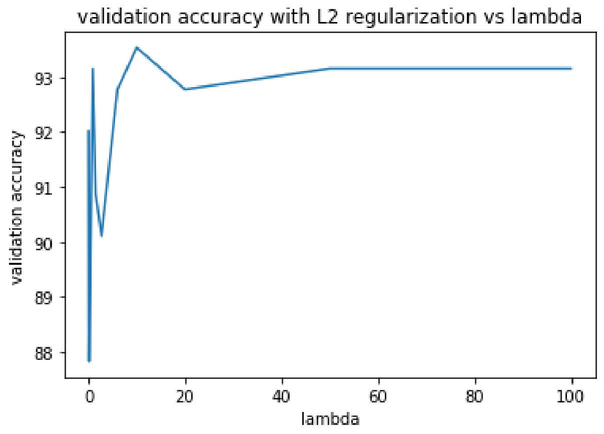


Figure 20: Variation of Validation Accuracy with the Regularization Parameter

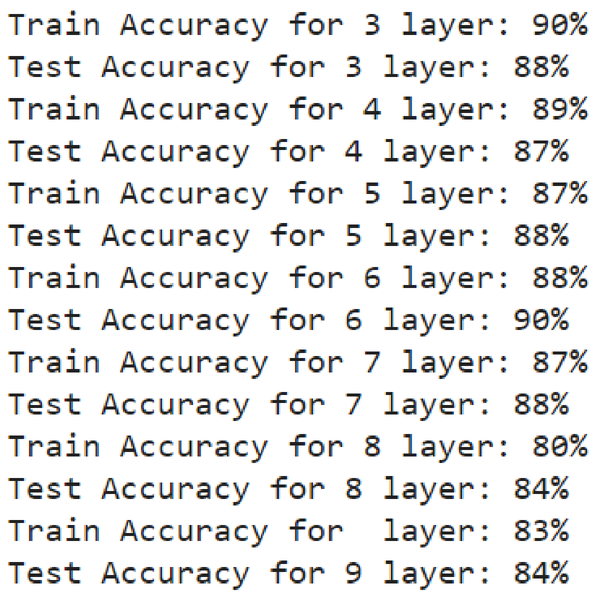
4.6 Optimizing the topology of the network

4.6.1 Aim

Optimize the topology of the network, i.e. the number of hidden layers and the number of neurons in each hidden layer, the size of your input, and the number of neurons in the output layer. Include a plot which shows the performance on the validation set as a function of the number of layers. Include another plot which shows the performance on the validation set as function of the hidden layers size.

4.6.2 Solution

We considered many network models and found train and test accuracies for all of them and on completing them we get that the highest accuracy if for 6 layers for the model we chose i.e with nodes 10 32 64 128 256 1.



```
Train Accuracy for 3 layer: 90%
Test Accuracy for 3 layer: 88%
Train Accuracy for 4 layer: 89%
Test Accuracy for 4 layer: 87%
Train Accuracy for 5 layer: 87%
Test Accuracy for 5 layer: 88%
Train Accuracy for 6 layer: 88%
Test Accuracy for 6 layer: 90%
Train Accuracy for 7 layer: 87%
Test Accuracy for 7 layer: 88%
Train Accuracy for 8 layer: 80%
Test Accuracy for 8 layer: 84%
Train Accuracy for  layer: 83%
Test Accuracy for 9 layer: 84%
```

Figure 21: Accuracies for different layer sizes

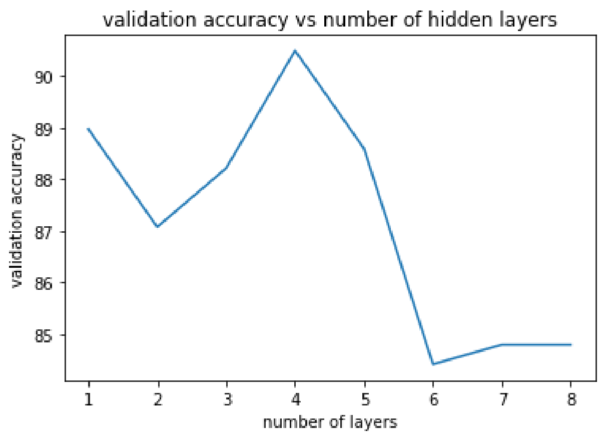


Figure 22: Accuracy for Different Number of Hidden Layers

When varying number of neurons in final hidden layer we got the following trend.

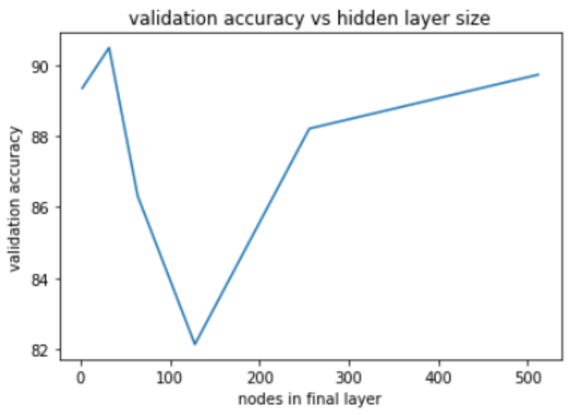


Figure 23: Accuracy for Varying hidden layers size

4.7 Comparing different Activation functions

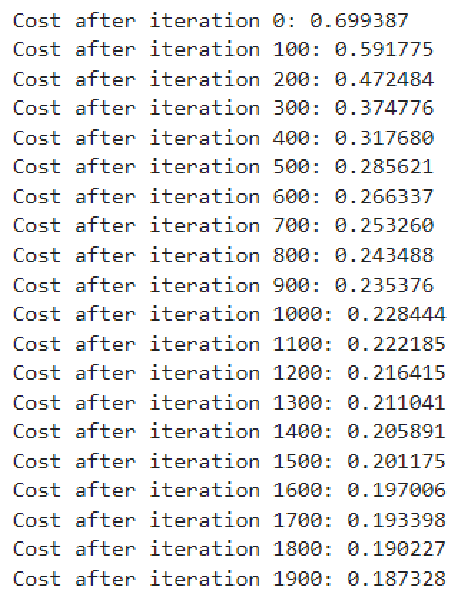
4.7.1 Aim

Train a network using the optimal set of parameters you have found so far and a different activation function than the one you defined in step 1. Comment on the performance on the validation set and discuss if it is any different than the initial activation function you used.

4.7.2 Solution

In this section we used ReLu as the intermediate layers activation function. ReLu is a simpler and more efficient activation function and is used in commonly used in bigger models such as CNN and it deals with the issues such as converging value of output of activation function in case of sigmoid or tanh functions, at higher values of input.

$$ReLu(x) = max(0, x) \tag{2}$$



```
Cost after iteration 0: 0.699387
Cost after iteration 100: 0.591775
Cost after iteration 200: 0.472484
Cost after iteration 300: 0.374776
Cost after iteration 400: 0.317680
Cost after iteration 500: 0.285621
Cost after iteration 600: 0.266337
Cost after iteration 700: 0.253260
Cost after iteration 800: 0.243488
Cost after iteration 900: 0.235376
Cost after iteration 1000: 0.228444
Cost after iteration 1100: 0.222185
Cost after iteration 1200: 0.216415
Cost after iteration 1300: 0.211041
Cost after iteration 1400: 0.205891
Cost after iteration 1500: 0.201175
Cost after iteration 1600: 0.197006
Cost after iteration 1700: 0.193398
Cost after iteration 1800: 0.190227
Cost after iteration 1900: 0.187328
```

Figure 24: Training cost for ReLu activation

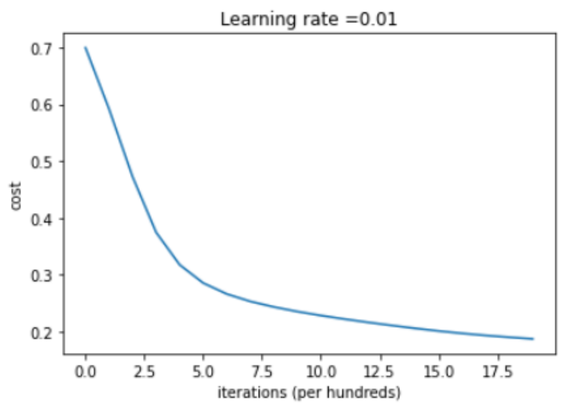
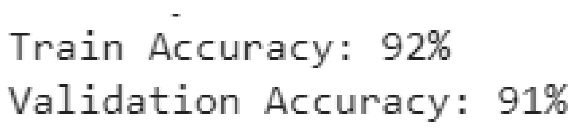


Figure 25: Training cost varying over iterations for ReLu activation



```
Train Accuracy: 92%
Validation Accuracy: 91%
```

Figure 26: Accuracy for ReLu activation

We attained a comparable validation accuracy in both cases of tanh and Relu activation, though training in the latter required slower learning rate and more number of iterations. Also training accuracy attained in case of tanh activation was significantly higher than relu.

4.8 Training the network using optimal set of parameters

4.8.1 Aim

Train a network using the optimal set of parameters and include a figure which shows the training and validation loss and another figure which shows the training and validation prediction error. Present in the same figure the curves you saved in step 2. Comment on their differences.

4.8.2 Solution

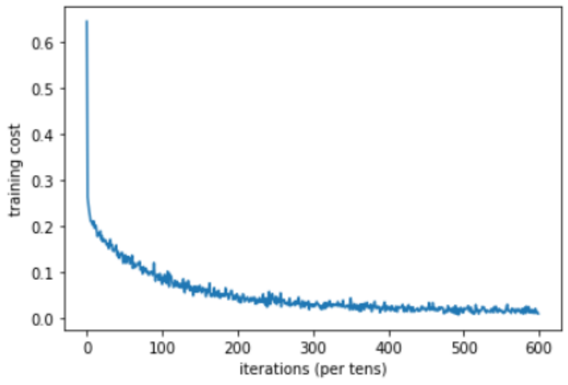


Figure 27: Training loss vs iterations

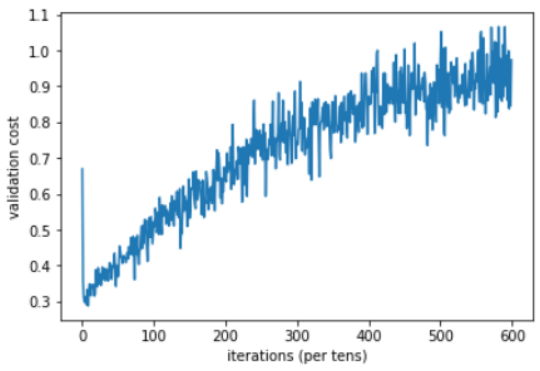


Figure 28: Validation loss vs iterations

In the below graph red plot signifies training loss for the without dropout network, whereas green plot has loss with dropout added network. There is a difference in x scale length of red and green graph since both the models were trained for different number of iterations

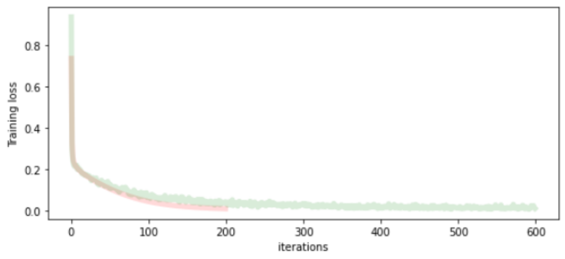


Figure 29: Comparison of training loss

In the below graph red plot signifies validation loss for the without dropout network, whereas green plot has loss with dropout added network. There is a difference in x scale length of red and green graph since both the models were trained for different number of iterations

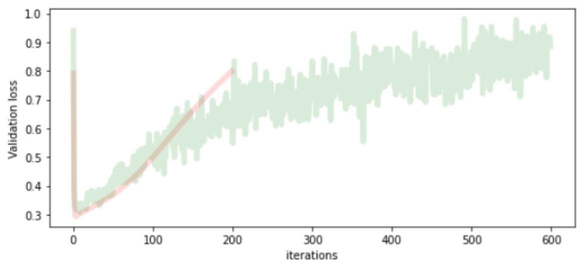


Figure 30: Comparison of Validation loss

Training loss might be less in the red graph, because model is prone to over fitting, and thus models fits the dataset very well. For the validation loss graph, in the long run, green graph has lower validation loss, since it has been regularized by dropout algorithm.

5 Performance Estimation

5.0.1 Aim

Test the performance of the network trained with the optimal set of parameters on the test set and report the confusion matrix.

5.0.2 Solution

On running through various algorithms, varying number of layers and number of neurons in it and altering the various parameters such as learning rate, probability threshold and so on, we reached the conclusion that the model with the layers as 10, 32, 64, 128, 256, 1 is the most efficient one, with a characteristic validation accuracy of 92% and final test accuracy of 91%.

```
Final test Accuracy: 91%
```

On comparison of training cost variation over iterations graph of without dropout and with dropout, later one settled with minor disturbance in cost, which is essential because of the regularizing nature of dropout algorithm. Also, the validation accuracy increased with later iterations thus signifying, model was being prevented from over fitting the dataset.

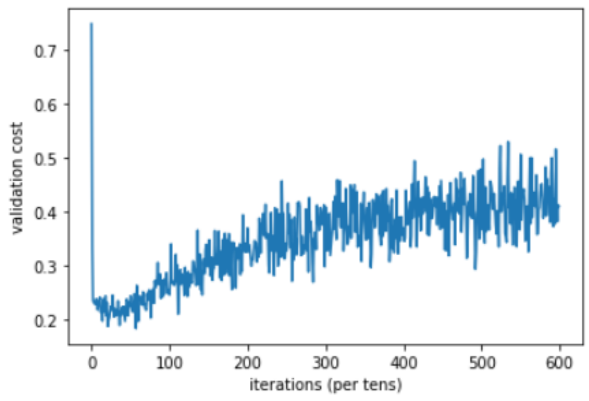
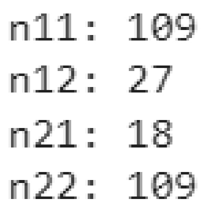


Figure 31: Final Validation Accuracy



```
n11: 109
n12: 27
n21: 18
n22: 109
```

Figure 32: Confusion Matrix values

6 Conclusion

- Having tried various models parameters with varying number of layers, neurons in each layer, initialization function (random and He), dropout, regularization methods (Lasso and Ridge), different activation functions (tanh and relu), we came to conclusion that he initialization with tanh activation on a 4 hidden layers (32, 64, 128, 256) network model added on with momentum and dropout regularization gave the best i.e. 91% test accuracy.
- Also the various in between parameters such as learning rate, learning rate update parameter, update parameter in momentum, neuron dropout probability were systematically tested to set the values that give best accuracy
- This whole training process was done on a training and validation and the final test accuracy was determined on an entirely unused subset of dataset to prevent any accidental bias in prediction

- The final efficiency of model was judged on basis of accuracy as well as confusion matrix and the model proved to be highly efficient in both respects