Implementation

Linear Regression Model

Basis Function Used

Gaussian Basis Function

$\boldsymbol{\phi}_j(x) = exp\{- (x-\mu_j)^2 / 2s^2\}$

Division of Dataset

40% Training Set

10% Validation Set

50% Testing Set

Choice of Parameters

Each input vector consists of 46 features.

Selecting μ

For each feature vector μ is calculated by taking the mean of all 46 features.

muvec.dat contains the mean for each of 15211 feature vectors.

Following are the calculations for μ for each feature vector:

$d=46$

$\mu=\sum_{i=1}^{d}(x_i)/d$

For each basis function $\boldsymbol{\phi}_{j,}$

$\mu_j = \mu, for\ j=1$

$\mu_j = \mu_{j-1} + (\mu/M)\ , for\ j=2\ to\ M.$

Selecting *s*

For each feature vector *s* is calculated by computing the standard deviation of all 46 features.

sigmavec.dat contains the standard deviation for each of 15211 feature vectors.

Following are the calculations for *s* for each feature vector:

s = standard deviation of 46 features.

$s^2 = \sum_{i=1}^{d} (x_i - \mu_i)\ \hat{}2/d$

For each basis function $\boldsymbol{\phi}_j$,

$s_j = s$, *for* $j=1$

$s_j = s_{j-1} + (s/M)$, *for* $j=2$ *to* M

Selecting Model Complexity *(M)*

Below are the calculation done to decide *M*,

For $M=1$ *to* 50 the below parameters were calculated,

$\boldsymbol{W_{ML}} = (\boldsymbol{\phi}^T \boldsymbol{\phi})^{-1}\ \boldsymbol{\phi}^T \boldsymbol{t}$

$\boldsymbol{W_{ML}}$ = Maximum Likelihood Parameters for $y(x,w)$

$\boldsymbol{t}$ **=** target values of training set.

Using the function, $y(x,w) = \sum_{j=0}^{M-1} (w_j\ \phi_j(x)) = W^T \boldsymbol{\phi}\,(x)$, the predicted values are calculated.

After getting the predicted values *Erms* for both training set and validation set was calculated using the below formulae,

$E_D(w) = 1/2 * \sum_{n=1}^{N} (t_{n-}\ W'\phi(x_n\ ))^2$

$Erms = \sqrt[2]{2E(w)/N}$

The following graph for different values of *M* against the *Erms* for training set and *Erms* for validation set was plotted.

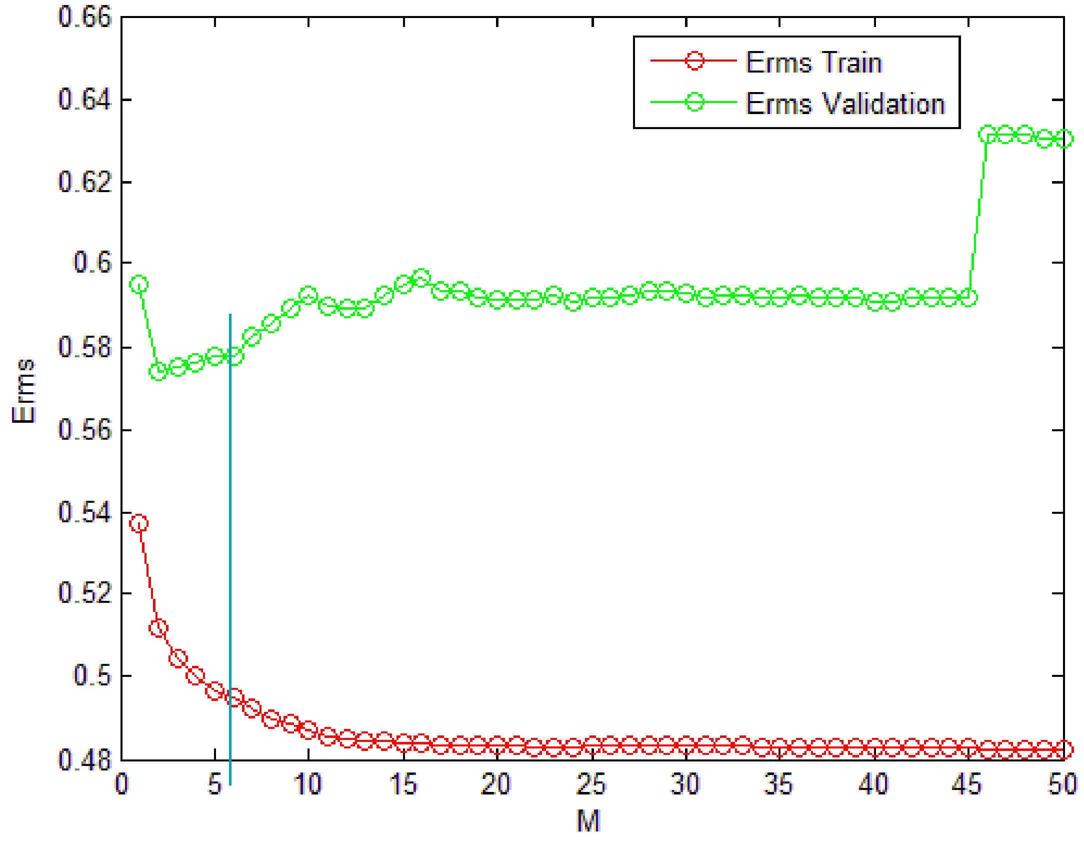


Fig 1: ***Erms*** **v\s** ***M***

From the graph it can be observed that *Erms* for Training and Validation substantially decreases as *M* increases. After *M=6* (blue vertical line) the *Erms* for training keep decreasing but *Erms* for Validation start increasing. Hence selected *M=6* for this model.

For M=6,

Erms train= 0.4951

Erms Validation=0.5777

Selecting Regularization Parameter λ to avoid over-fitting

To select λ, validation set was used and *Erms* of validation for different values of λ was plotted.

The result was as follows,

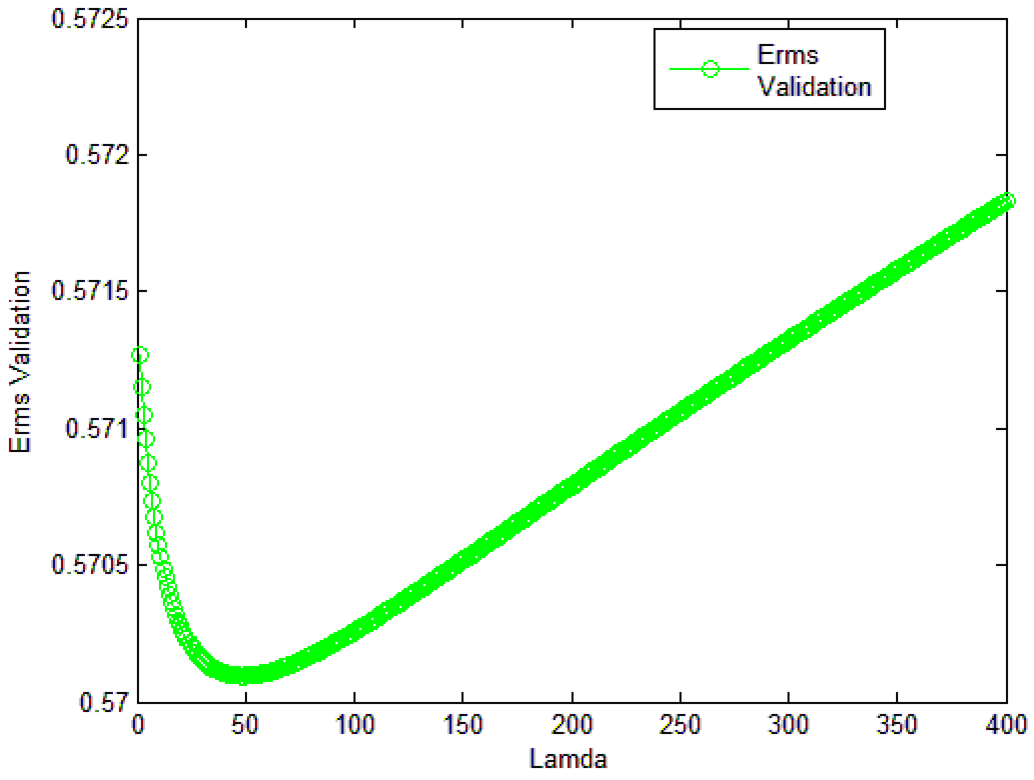


Fig 2: ***Erms Validation*** **v\s** λ

From the above graph it can be observed that the Erms value for validation set decreased with a good rate till $\lambda=50$, after which there was exponential rise in the Erms value .Hence $\lambda=50$ was selected for this model.

Thus after using $\lambda=50$ the parameters are calculated by using the below formula,

$$W_{ML} = (\lambda I + \phi^T \phi)^{-1} \phi^T t$$

The *Erms* for validation after regularization

Erms Train =0.5083

Erms Validation=0.5701

Erms for Predicted values of Testset

For Testset the predicted values were calculated after the regularization.

Below result was observed,

Erms Test=0.5946

Neural Network model for regression

For implementing Neural Network model on the dataset, the 'nntool' of MATLAB was used.

The dataset was divided as below,

40% Training Set

10% Validation Set

50% Testing Set

Training the Model

Levenberg-Marquardt back propagation is used to train the model.

```
net.trainFcn = 'trainlm';
```

`'trainlm'` is a network training function that updates weight and bias states according to Levenberg-Marquardt optimization.

The Levenberg–Marquardt algorithm blends the steepest descent method and the Gauss–Newton algorithm. Fortunately, it inherits the speed advantage of the Gauss–Newton algorithm and the stability of the steepest descent method. It's more robust than the Gauss–Newton algorithm, because in many cases it can converge well even if the error surface is much more complex than the quadratic situation. Although the Levenberg–Marquardt algorithm tends to be a bit slower than Gauss–Newton algorithm (in convergent situation), it converges much faster than the steepest descent method.

The basic idea of the Levenberg–Marquardt algorithm is that it performs a combined training process: around the area with complex curvature, the Levenberg–Marquardt algorithm switches to the steepest descent algorithm, until the local curvature is proper to make a quadratic approximation; then it approximately becomes the Gauss–Newton algorithm, which can speed up the convergence significantly.

Selecting number of Hidden Neurons (*hiddenLayerSize*)

The Training and Validation of dataset was carried out on the model with different number of hidden neurons. The *Erms* for Training and Validation was plotted against the number of Hidden Neurons. Following were the observations,

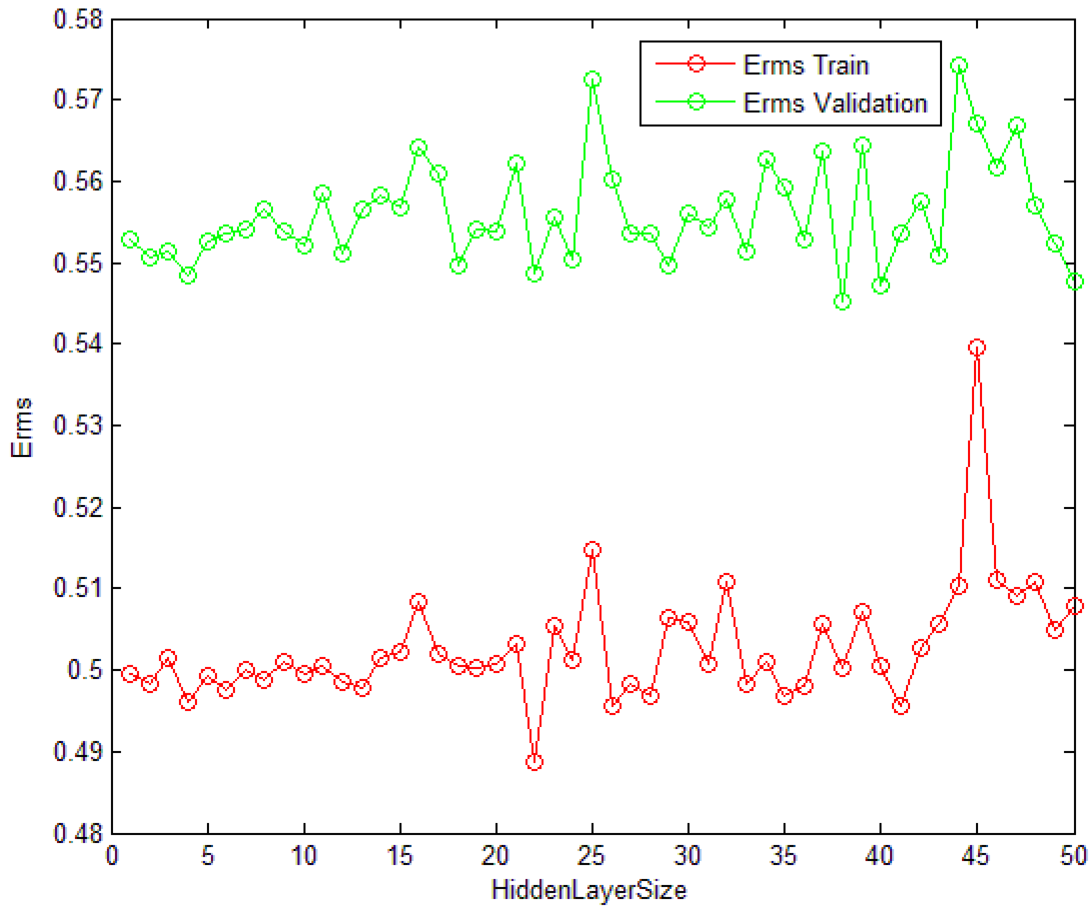


Fig 3: ***Erms*** **v\s** ***HiddenLayerSize***

From the above figure it is observed that for 22 hidden layers the *Erms* for training is minimum. Hence for this model *hiddenLayerSize=22.*

The observed values for *Erms* are as follows:

Erms Train =0.4928

Erms Validation=0.5615

Regularization

To avoid over fitting regularization parameter can be set in 'nntool' by setting 'net.performParam.regularization'. The following observation was made for deciding the regularization parameter.

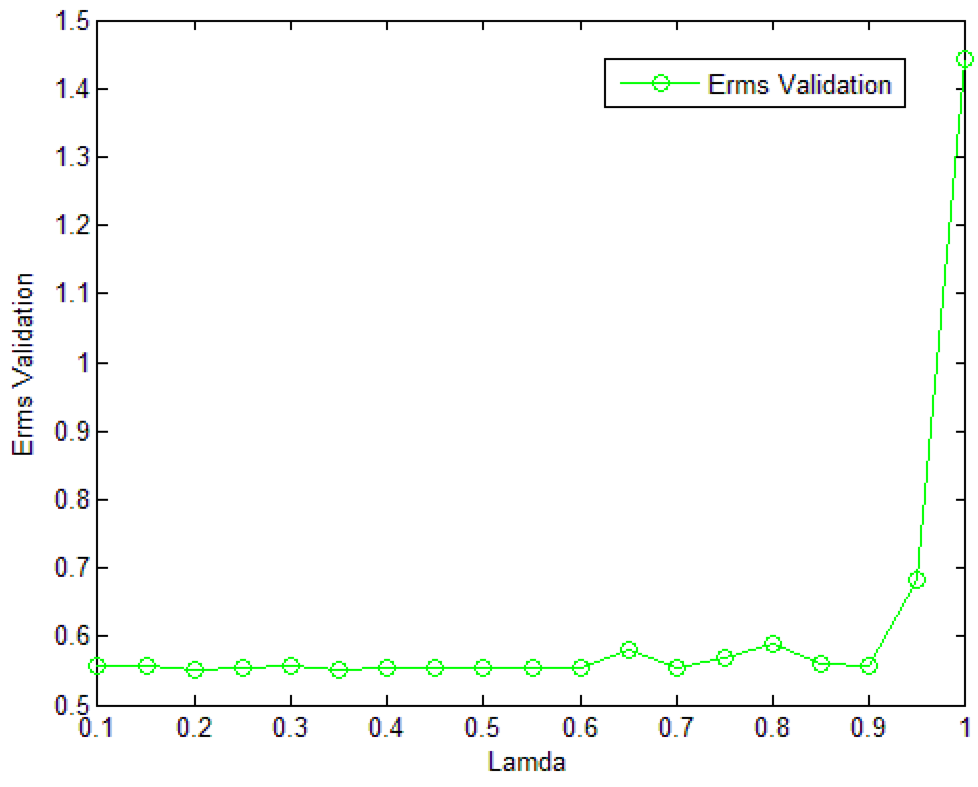


Fig 4: *Erms Validation* **v\s** λ

From the graph it can be observed that *Erms validation* increases exponentially after $\lambda=0.9$.

Erms Validation for $\lambda=0.1$ is the lowest. Hence we chose $\lambda=0.1$ as regularization parameter.

Thus after regularization,

Erms Train =0.5074

Erms validation= 0.5487

Erms for Predicted values of Testset

For Testset the predicted values were calculated after the regularization.

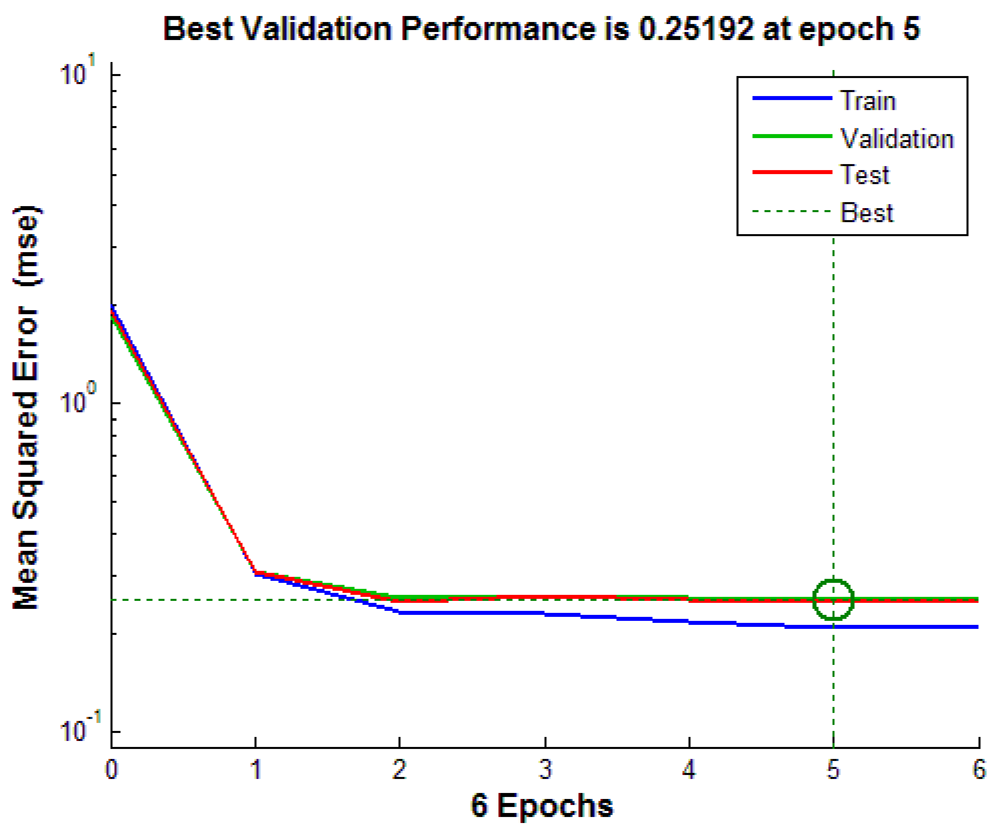
Below result was observed,

Erms Test=0.5074

Other Observations

Number of EPOCH

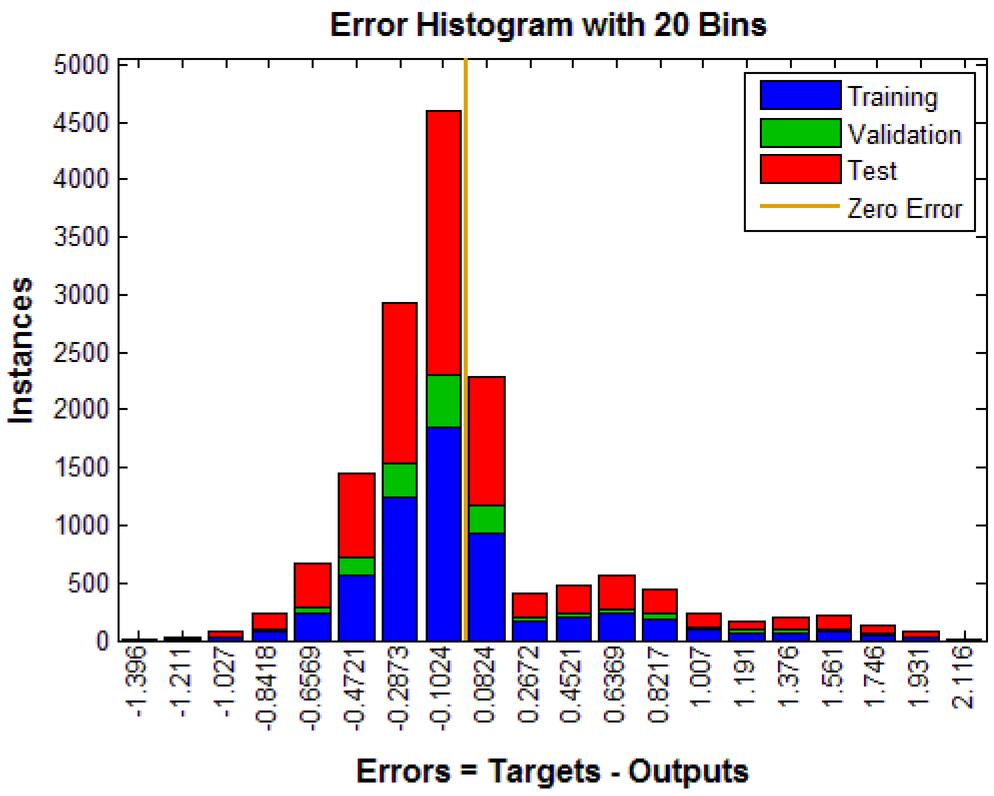
An epoch is a measure of the number of times all of the training vectors are used once to update the weights.



From the graph it can be observed that for six iterations the training vectors were used to update the weights.

For the Fifth epoch best validation performance was observed.

Error Histogram for Neural Network Model:



Conclusion

The regression using the Neural Network Model provide better prediction of the target values for the Testset than the Linear Regression Model.

For Neural Network Model: *Erms Test=0.5074*

For Linear Regression model: *Erms Test=0.5946*

Note: The results for Neural Network Model may vary each time the model is run on the same dataset. This is because 'nntool' sets random weight and bias values each time the Neural Network model is run on the dataset.