CSCI 447 Project 2: Neural Networks Fall 2015

Brandon Fenton
Department of Mathematical Sciences
Montana State University
Bozeman, MT 59717
Email: brandon.fenton@gmail.com

Jenna Lipscomb
Department of Computer Science
Montana State University
Bozeman, MT 59717
Email: jennalynnlipscomb@gmail.com

John Sherrill
Department of Mathematical Sciences
Montana State University
Bozeman, MT 59717-2400
Email: prof.sherrill@gmail.com

***Abstract*—Two machine learning frameworks, a radial basis function network and a feedforward neural network, were created to approximate the generalized Rosenbrock function for a variable number of dimensions. An iterative k-fold cross validation process was implemented for both networks to tune model parameters and the resultant models were tested on a final large-scale test set to evaluate performance and thus compare the two different frameworks in the context of function approximation for this particular example. It was hypothesized that the feedforward neural network would possibly be able to model the algebraic representation of the Rosenbrock function and would thus outperform the radial basis function network. This hypothesis was refuted by evidence from the model evaluation process indicating the the radial basis function performs better in this context. The radial basis function network was trained by a recursive least squares implementation that was hypothesized to be significantly faster than the backpropogation algorithm used in training the feedforward neural network. This was confirmed in some of the different configurations for testing the networks.**

I. Introduction

The generalized N-dimensional Rosenbrock function may be expressed as

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$$

$$\text{where} \quad \mathbf{x} = [x_1, \ldots, x_N] \in \mathbb{R}^N.$$

This function was approximated for N = 2, 3, 4, 5, and 6 with each dimension constrained on the interval -1.5 to 1.5. That is, the domain was a 2, 3, 4, 5, and 6 dimensional hypercube $\mathbf{x} \in [-1.5, 1.5]^N$, $N \in \{2, 3, 4, 5, 6\}$.

These functions were to be approximated using a feedforward neural network and radial basis function network. The broad goal of this exercise was two fold: 1) comparing the rate of convergence to an exceptable level of error between the two frameworks and 2) comparing final model performance.

II. Network and Algorithm Descriptions

Two neural networks with different training algorithms were implemented. A feedforward neural (FFN) network framework was created that utilizes stochastic gradient descent via the now famous backpropogation algorithm [1]. A radial basis function (RBF) network framework was also created that utilizes a k-means clustering algorithm for initializing model parameters (to be described in more detail below) and recursive least squares method for iteratively updating the model.

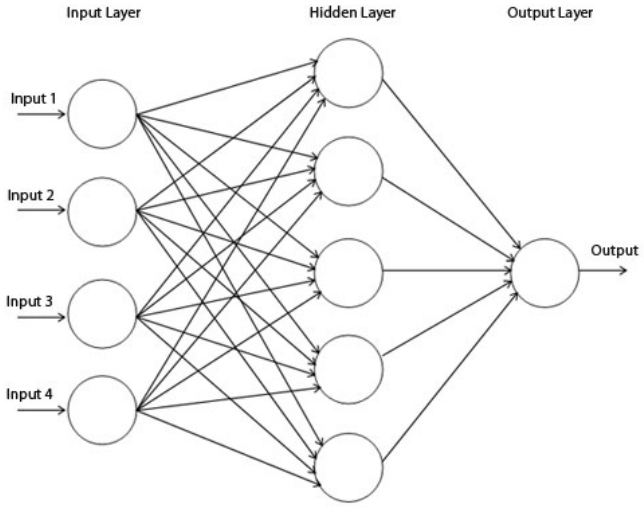

Fig. 1. Example Neural Network

A. Feedforward Neural Network

A standard FFN network consists of a sequence of l layers, each consisting of n_i different "nodes". The first layer is called the input layer and unsurprising takes as input the d_{in} different attributes of the data to be modeled. Thus, $n_1 = d_{in}$. The last layer is called the output layer and also unsurprising provides the output of the network, i.e. the predicted value for the attribute to be predicted. Thus, if the dimensionality of the output space is d_{out}, then $n_l = d_{out}$. The other $l - 2$ layers inbetween the input and output layers are referred to as hidden layers. Each node in hidden layer i is connected with all nodes in layer $i-1$ and $i+1$. Lastly, the edges in the graph described are all unidirectional, such that if one begins at an input node, one nescissarily ends at an output done. A simple 3 layer network is presented visually in Fig. 1 where $n_1 = 4$, $n_2 = 5$, and $n_3 = 1$.

Excluding the edges contected to the input layer, for each edge there is an associated weight. Nodes take as input the

weighted sum of outputs from their upsteam nodes plus a bias term specific to each node (a node in layer i is considered upstream from nodes in layers $j > i$ and a node in layer i is considered downstream from nodes in layers $j < i$). In the implementation provided, weights and biases are initialized to random values. The output value of a node is the value of a network-wide specified activation function of the input to the node. A typical activation function used is the logistic function or the hyperbolic tangent function. For this project, both the logistic function and a linear activation function were implemented with the choice left up to the user. If the linear activation function is chosen a slope must be supplied.

1) Training: Training for FFN networks is generally accomplished by a task often referred to as "backpropogation". In general terms, the error between predicted output and target output is propogated back through the network to provide information to be able to perform a gradient descent through the parameter space (the space of all possible weights and bias values).

B. Radial Basis Function Network

An RBF network has the same topology as a FFN network except there is only one hidden layer. RBF networks are also unidirection. The graphic in Fig. 1 also represents an RBF network topoloy.

The defining difference between the two networks is that RBF networks use *radial basis functions* for activation functions. A radial basis function, $\phi(\mathbf{x}, \mathbf{c})$, is a function solely of the distance between points $\mathbf{x}$ and $\mathbf{c}$: $||\mathbf{x}-\mathbf{c}||$. A typical choice for the basis function (and the one chosen for the provided implementation) is a Gaussian basis function:

$$\phi(\mathbf{x}) = e^{-\beta||\mathbf{x}-\mathbf{c}||^2} \tag{1}$$

where β is a user-specified, tunable parameter. For each node i in the hidden layer (referred to as "Gaussian Nodes" or simply "Gaussians") the centers of the basis function, $\mathbf{c}_i$ are different.

The only edges in the network that have associated weights w_i are the edges connecting the hidden layer to the output layer. While not necessary, a bias term b is included in the authors implementation. Thus, the predicted output o, given input $\mathbf{x}$, of an RBF network with k Gaussians and one dimensional output is

$$o = \sum_{i=1}^{k} w_i e^{-\beta||\mathbf{x}-\mathbf{c_i}||^2} + b.$$

1) Training: The training process for RBF networks is customarily divided into two parts. The first step is choosing appropriate centers for the radial basis functions of the Gaussian nodes. Often this is performed by using the k-means clustering algorithm. Such an algorithm is used by the authors for this step.

Once the Gaussian centers have been chosen, the model weights and bias may be learned. This may be accomplished by a variety of methods. The two most common being gradient descent and ordinary least-squares minimization. Given a training data set $D = \{\mathbf{x}_j\}_{j=}^{N}$, there is a closed form expression given by a generalized inverse of an appropriately organized matrix of the training data. Computation of matrix inverses is, however, computationally laborious and is not well suited for very large RBF networks.

Recursive least squares provides a more efficient method that allows for continual learning and updating of the weights and bias. Given one training point, an update to the weights and bias can be expressed in terms of scalar and matrix multiplication (technically a matrix inverse is computed but is an inverse of a 1 by 1 matrix (division)). Thus, the only step at which a matrix inverse is required to compute is at the initialization step when starting values of weights and bias are chosen. While it is possible to randomly assign weights, this would be providing irremovable error into parameter estimation that would be impossible to remove. The method the authors chose was as follows: given an RBF network with k Gaussians, compute the least-squares minimizing weights and bias for k training points (using the generalized inverse). Then iteratively update the weights and bias by recursive least squares.

III. Experimental Approach

The process used for tuning the tunable parameters for the two frameworks was accomplished by random sampling from the tuning parameter space. For the feedforward network this was $T = \{(a, m) : a \in (0, 1), m \in (0, 1)\}$ where a represents the learning rate and m, the momentum. For the RBF network this was $T = \{(\lambda, \beta_{tune}) : \lambda \in (0, .01), \beta_{tune} \in (0, .1)\}$. After reasonable values for the tuning parameters were found, networks were trained and tested using 10-fold cross-validation.

A. Tuning

Searching a grid for all possible tuning parameters proved to be impractical given the slow speed of the learning process of the various algorithms. A less intensive approach was taken that randomly sampled from the tuning parameter space. There is theoretical evidence supporting this approach [2] suggesting that while optimal parameters may not be found, with a sufficiently large sample of possible tuning parameters, *near optimal* tuning parameters will almost certainly be found.

1) FFN Network: After testing a random selection of possible learning rates and momentums, it was found that the quickest rates of convergence were found with a learning rate of $a = 0.05$ and momentum of $m = 0$. Results are provided in Table I.

The topology used for the feedforward network was as follows: input layer with n_{in} nodes, a hidden layer with $2n_{in}$ nodes, a single node for the output layer.

2) RBF Network: After testing a random selection of possible tolerance for the k-means clustering algorithm, a value of $\lambda = 0.01$ was chosen. The normalization constant, β, used in the radial basis functions was considered dependent on the dimensionality n_{in} of the input $D = \{\mathbf{x}_i\}_i$ and the number

TABLE I
FEEDFORWARD TUNING RESULTS

Dimensions	Learning Rate	Momentum
2	0.05	0
3	0.05	0
4	0.05	0
5	0.05	0
6	0.05	0

of Gaussian nodes n_G such that the β used in equation 1 is given by

$$\beta = \frac{(n_G)^{\beta_{tune}}}{\max_{i \neq j} ||\mathbf{x}_i - \mathbf{x}_j||}$$

where β_{tune} is the user-specified tuning parameter. Results are provided in Table II.

TABLE II
RBF TUNING RESULTS

Dimensions	β_{tune}	Clustering Tolerance
2	0.1	0.1
3	0.1	0.1
4	0.1	0.1
5	0.1	0.1
6	0.1	0.1

Given that the dimensionality of the input was n_{in}, the authors selected $2n_{in}$ as the number of Gaussian nodes to include in the single hidden layer.

IV. RESULTS

And extented listing of model performance in terms of root mean squared error (RMS), absolute error (AE), and correlation between target and predicted values are prodived in Tables III and IV. For each dimension, the measures of error are averages across the 10 different folds used in the 10-fold cross-validation.

TABLE III
FEEDFORWARD MODEL PERFORMANCE

Dimensions	RMS	AE	Correlation
2	217.5	153.0	0.273
3	339.8	249.8	0.313
4	509.2	390.1	0.318
5	508.4	391.6	0.312
6	555.9	432.0	0.355

V. CONCLUSION

As can be seen in the tables above, the RBF network framework performed significantly better than the FFN network in all three measures chosen for analysis. In particular, it was found that the predicted values from the RBF network were much more highly correlated with the target values. These results are consistent across all dimensions of the problem.

TABLE IV
RBF MODEL PERFORMANCE

Dimensions	RMS	AE	Correlation
2	161.8	113.7	0.705
3	221.2	165.9	0.778
4	259.4	196.2	0.814
5	284.1	219.0	0.838
6	307.3	239.0	0.845

Thus, the hypothesis that the FFN network would be able to model the algebraic properties of the Rosenbrock function (its polynomial nature) is not confirmed by these experiments. It is possible that with better choices of tuning parameters, more appropriate models could have been fit to the data. The authors hypothesize that a large sample from the tuning parameter space would have resulted in finding better tuning parameters and thus would have resulted in building models with much smaller errors.

One potential explination for the difference in performance is the more efficient training method used in the RBF network. For the two highest dimension cases (n=5, 6) some of the FFN networks being trained did not converge. That is, no local optima were found for the weights and biases. Naturally, this would not result in a satisfactory average level of error by any measure.

REFERENCES

[1] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations Volume 1: Foundations, MIT Press, Cambridge, MA.

[2] Bergstra, J., Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research. Volume 13. 281 - 305.

[3] Fenton, B., Sherrill, J. (2015). Github repository for project: https://github.com/joncheryl/csci447.