Analysis of Neural Networks in Function Approximation

Mitch Baker MITCHT.BAKER@GMAIL.COM
Gianforte School of Computing
357 Barnard Hall
Montana State University
Bozeman, MT 59717

James Durtka JDURTKA@GMAIL.COM
Department of Electrical and Computer Engineering
Montana State University
P.O. Box 173780 Bozeman, MT 59717-3780

Hongchuan Wang HONGCHUAN91@GMAIL.COM
Department of Electrical and Computer Engineering
Montana State University
P.O. Box 173780 Bozeman, MT 59717-3780

Editor:

Abstract

In this paper, the ability of Several different configurations of multilayer perceptron (feed-forward) and radial basis function neural networks to work as function approximators is researched, experimented upon, and analyzed. Data is collected on accuracy (both on training and cross-validated sets) and training times for different networks and different datasets. Convergence behavior is explored using the sequences of RMSEs generated during successive epochs for each network and dataset.

Keywords: Neural Network, backpropagation, Radial Basis Function, convergence

Problem Statement

The two neural networks being explored are the feed-forward neural network with backpropagation and the radial basis function neural network. Both of these networks are described in further detail later in this paper. Both of these networks have been proven to be universal function approximators. In order to test this feature of the networks, we will be training the networks to learn the Rosenbrock function (Rosenbrock (1960)), as seen below in figure 1, for varying dimensions. In our case, we will be testing the Rosenbrock function for the values of $n \in 2, 3, 4, 5, 6$ for the two networks implimented: the Multi-layer Perceptron and the Radial Basis Function Network.

$$(\mathbf{x}) = f(x_1, x_2, ..., x_n) = \sum_{i=1}^{n-1} [(1 - x_i^2) + 100(x_{i+1} - x_i^2)^2] \tag{1}$$

Aside from some general concerns about the curse of dimensionality, we hypothesize that there will be very distinct differences in performance between the different networks, both

in terms of accuracy and convergence.

For the MLP networks, we expect to achieve higher accuracy with more nodes and/or more layers, but with longer convergence times, because the network is learning more features. Number of layers are expected to mitigate number of nodes; for instance, two hidden layers, with 10 and 100 nodes respectively, may outperform a single hidden layer with 1,000 nodes - if not for accuracy, then certainly for convergence.

For RBF networks, we again expect more nodes to produce better accuracy but slower convergence.

It is expected that networks trained on higher dimensional datasets become exponentially worse in terms of accuracy (since it isn't practical to scale the data up exponentially). Further, a similar hypothesis might be formed regarding the training time for numbers of layers: more epochs to converge.

Algorithms Implemented

The problem to be addressed is training neural networks to approximate a non-linear, higher-dimensional function, namely the Rosenbrock function. For the sake of effectiveness, we restrict our attention to the hypercube bounded by [-1.5, +1.5] in each dimension for the Rosenbrock function in dimensions $d \in 2, 3, 4, 5, 6$. We are then interested in exploring various approaches to building and training neural networks which may be used to approximate this function. Two basic types of network are employed for this: ordinary feedforward (multi-layer perceptron) networks and radial-basis function networks.

Feed-Forward Neural Network with BackPropagation

Multi-layer Perceptron (MLP) networks are trained entirely using backpropagation (via gradient descent) (Rumelhart and Williams (1986)). Using an object-oriented approach, a neural network is implemented which is designed to take arbitrary inputs and map them to arbitrary outputs, using arbitrary hidden layers with arbitrary number of nodes in each, and arbitrary activation functions throughout the network. Backpropagation is implemented at the top level of abstraction, and applied layer-by-layer using the fast matrix computations available from the Numerical Python (NumPy) library. Backpropagation is implemented with the momentum idea. Further helper functions were developed for mini-batch gradient descent and cross-validation, although only the latter was used during actual training. The routines are optimized for correctness, rather than speed.

k-Fold Cross-validation is implemented (5 folds were used in practice). Additionally, the program holds out 20% at the very beginning for evaluation at the very end.

An application program was developed around the classes created to implement MLP and RBF networks. This application is designed to allow for training of all networks over all

datasets as a batch job. Exploration of network parameters (e.g. number of hidden layers, number of nodes, number of RBFs) takes place within the application program, while a batch file is used to feed each dataset into the program via commandline, one at a time, so that each configuration of neural net is trained on each dataset, producing data about accuracy and training time for each of the algorithms applied to each dataset.

Radial Basis Function Neural Network

Following a commonly used approach, Radial Basis Function (RBF) networks (Broomhead and Lowe (1988)) are trained by use of the k-means clustering algorithm to cluster data, followed by backpropagation to the single output layer (in this application, this is just a single node with weights from the RBFs). Thus, the majority of the training time is in the application of k-means clustering, not the network itself.

The RBF centroids are initialized using a grid-sampling concept. A number is chosen which becomes half the number of RBF nodes per dimension. In this way, the proper number of centroids are distributed in a grid across the problem domain. The numbers used are 4, 6, and 8. Another curse of dimensionality discovered with this approach is that, for higher dimensions, the grid approach produces very large numbers of RBFs, which makes training time an issue. In the interest of timeliness, for higher dimensions (4, 5, and 6), a 3d grid is used (with the other dimensions of the centroids set to 0), which makes training faster with additional sacrifice to accuracy (in fact, this in some sense means that the solution is invalid for higher dimensions). For this reason, we expect the MLP networks to perform much more well in these dimensions.

For the parameter σ_i, the heuristic (described in Benoudjit and Verleysen (2003), and discussed in the design document) is used. The same value for σ is used throughout the RBF network.

Experimental Design

Although the goal is to approximate a real, continuous function, it is necessary to discretize for the purpose of numerical computation. This implies producing a dataset for each of the five dimensions we wish to approximate the Rosenbrock function over. Recall that we selected the domain of [-1.5, +1.5] in each dimension. Further, we now sample the function within this domain. Two different datasets are produced per function: one using random sampling, and one using grid sampling. These datasets can be visualized in the $n = 2$ case using 3D models as seen in figure 1 and figure 2. Among other things, we are interested to see what effect sampling methodology has on the networks' ability to learn the function.

As an approach to combating the so-called "Curse of Dimensionality," progressively larger datasets were generated for larger dimensions d. Unfortunately, as the datasets became very large, time to train became a significant constraining factor and we were forced to downsample some of the larger datasets. This was done using random sampling, so

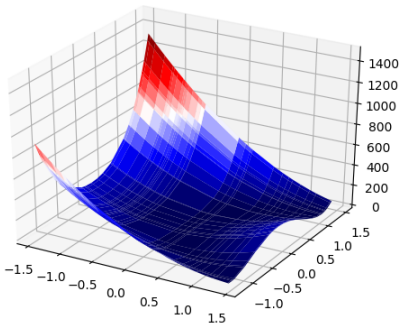

Figure 1: 3D model produced by randomly sampling the 3d Rosenbrock function

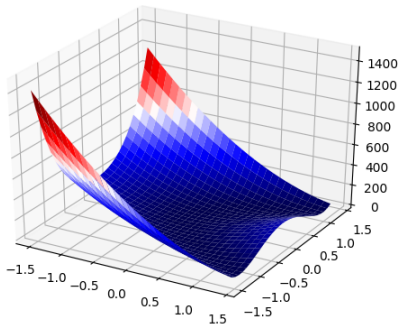

Figure 2: 3D model produced by grid sampling the 3d Rosenbrock function

although there are still two datasets (produced via grid sampling and random sampling, respectively), they have both been down-sampled randomly.

Given a sampled dataset, we holdout 20% of the data for evaluation at the end, and then perform 10-fold cross-validation during training.

Finally, we vary several parameters of the neural networks we are testing.

For RBF networks, each network is trained with a parametrically determined number of radial basis functions. This is because we want to do a good job of sampling the space, but in higher dimensions we need fewer RBFs for practical purposes.

For MLP networks, each dataset is trained with a network with either 0, 1, or 2 hidden layers. Furthermore, when there are hidden layers, each layer is composed with either 10 or 100 nodes. This means a total of $1+2+4 = 7$ MLP networks are trained in total.

The original experiment design proved too ambitious to complete in a timely manner; originally, the goal had been 10-fold cross-validation with 5,000 epochs per fold. In practice, in order to complete training on most of the networks the specifications were reduced to a mere 100 epochs per fold, with five folds in total.

Experiment results

In some ways, due to the excessively ambitious experimental design which had to be modified midway through training to meet time constraints, the results are less than desired. A flawed approach was taken toward training higher dimensional RBF networks, the 5d case

was never trained, and some of the MLP networks had to be removed from the original design due to being too large. Due to a major oversight during program development, the final weights in each fold were never saved, so although we have a "test" dataset that was held out from the beginning, we have no way to evaluate the results over this test dataset. Finally, each fold was trained for at most 100 epochs, which in many cases was insufficient to show significant improvement.

However, a large amount of data was collected over what was done, which at least to some extent allows us to evaluate the networks we trained in terms of convergence and performance.

Algorithms	**Hidden Layers**	**Number of Nodes**	**RMSE RS**	**RMSE GS**
MLP	0	NA	174.0446	255.9826
	1	10	189.6478	269.8912
	1	100	200.015	286.0538
	2	10,10	191.0418	290.3936
	2	10,100	NA	NA
	2	100, 10	189.0706	288.7300
	2	100, 100	NA	NA
RBF	–	k=144	130.6688	236.2514
	–	k=324	141.9308	256.9854
	–	k=576	151.8992	271.0091

Table 1: Example of averaged results of MLP and RBF for $n = 2$ Rosenbrock function using random sampling, RS, and grid sampling, GS.

A sample of results seen in the 2d case is shown below, in figure 3. Here we see that the RBF networks tend to be more stable, at least in the beginning. Further, it is clear that the smaller RBF networks converge a little faster, as expected. The MLP networks are much more unstable (two of them actually blow up entirely: the 10/100 case and the 100/100 case); however, the potential for better results is clear, but it would seem to require more training time to see an improvement over the RBF networks.

One surprising detail seen in this plot is the performance of the 0-hidden layer MLP network. Initially, it seems to outperform most of the other MLP networks. However, it is also the most stable of the networks, and it flattens out very quickly, indicating that in the long run it will probably not do well.

We also looked a little more closely at the differences between RBF networks of different sizes, as seen in figures 4 and 5 below. Here we averaged the curves for training RMSE and validation RMSE separately, for the 64-node RBF and 216-node RBF cases applied to the grid-sampled 3d dataset. Notice that the y-axis has been truncated to accentuate the curves. At this scale, the differences are not so noticeable, but a big difference is that, while

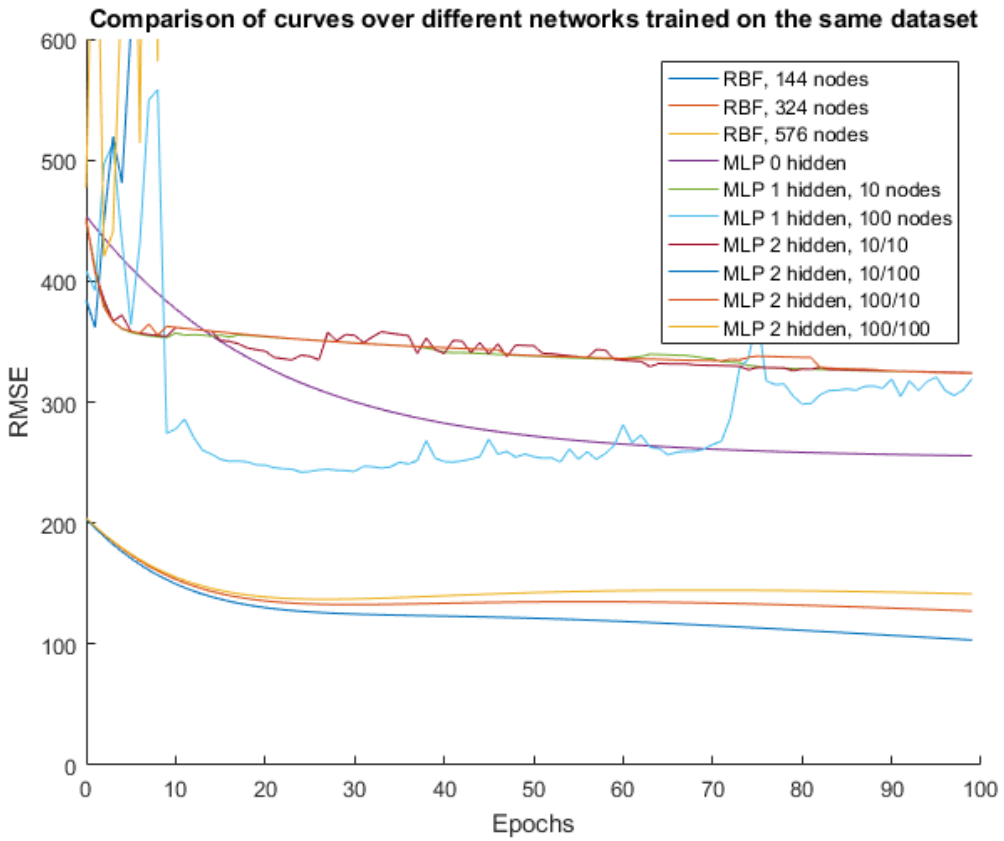

Figure 3: RMSE over the validation set for various networks trained on the 2d (grid-sampled) dataset

the training set error curve between the two is nearly identical, validation error actually decreases much more quickly in the 64-node case. This is indicative of the fact that, again, we are learning less features, and thus less likely to overfit. This demonstrates a clear tradeoff between generalization and forming a good fit to the data. In the long run, we expect that a larger RBF could provide greater accuracy on the training set, but fail to generalize.

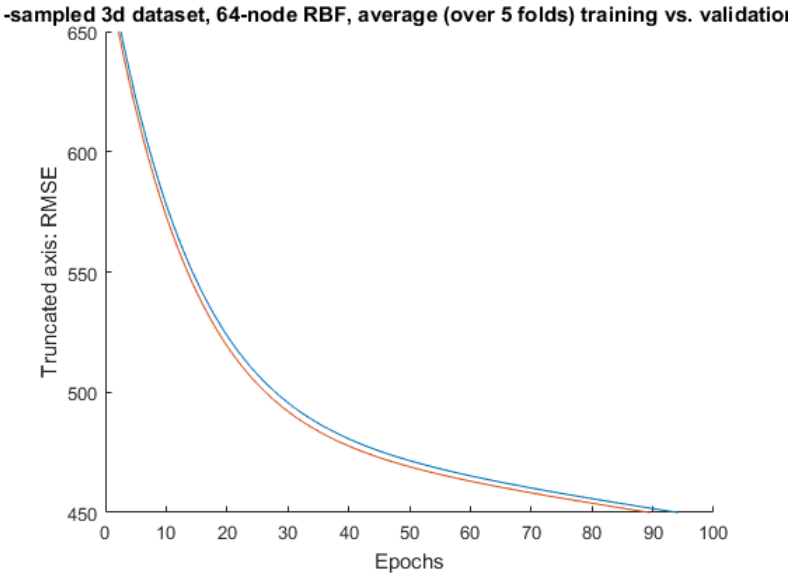

Figure 4: RMSEs (averaged over five folds) over the training and validation sets for the 64-node RBF trained on the 3d (grid-sampled) dataset

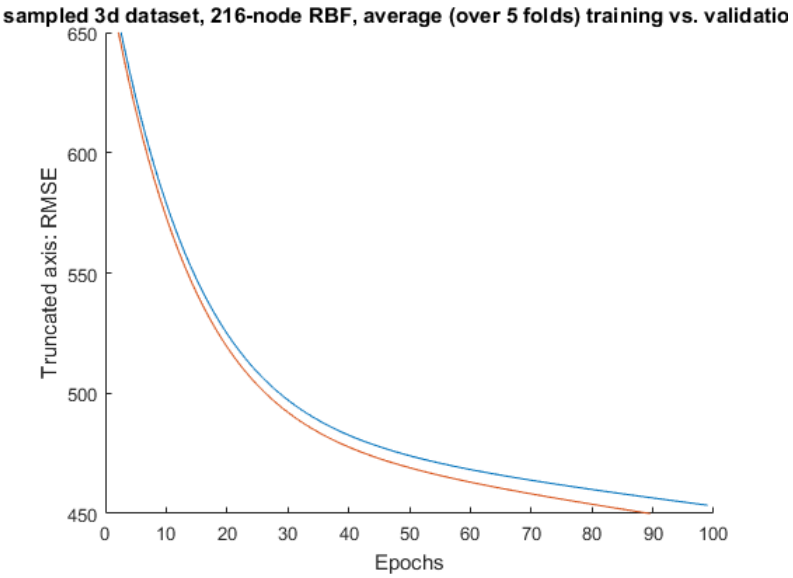

Figure 5: RMSEs (averaged over five folds) over the training and validation sets for the 216-node RBF trained on the 3d (grid-sampled) dataset

A goal proposed in the design document was to compare the grid-sampling and random-sampling strategies. This is something we attempted; however, the results are of limited usefulness due to a flawed sampling methodology, which produced differently sized datasets for each case. The differences in size are radical enough (in fact, nearly double) not to make a grid/random sampling comparison meaningful, but they do provide some insight into the results when the same network is trained over differently sized networks. Comparing figure 5 to figure 4, we see that the network converges in noticeably fewer epochs given more data. Of course, this also increases training time.

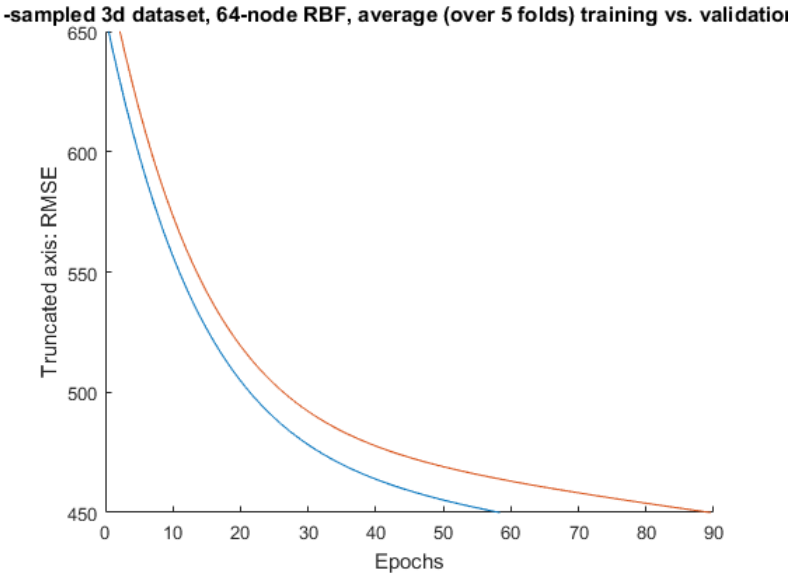

Figure 6: RMSEs (averaged over five folds) over the training and validation sets for the 216-node RBF trained on the 3d (grid-sampled) dataset

As regards performance across different numbers of dimensions, much of our data is inconclusive on this subject because we were never able to test the resulting networks on the held-out test data set (due to failure to save the weights from the networks that were trained). Thus, the only real comment to be made here is on the magnitude of the RMSE over the validation set during training (and even this is somewhat suspect, due to the different sizes of datasets trained). This is shown below, in figures 7 and 8.

Conclusion

In our previous paper, Baker and Wang (2017), we describe the hypothesis tested in this paper. While we did hypothesize for the smaller MLP networks, networks with zero or one hidden layer, to perform well on the lower dimensions, we did not predict that the smaller MLP networks would outperform them as much as they did for the smaller dimensions. This can be seen in our results, table , for the simplistic case of the Rosenbrock function

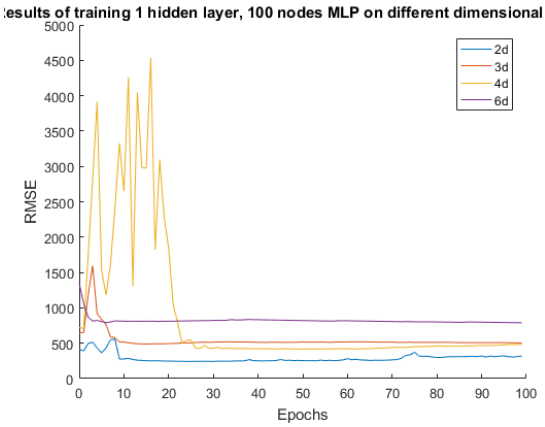

Figure 7: Typical curves of RMSE over different dimensional data trained by the same net.

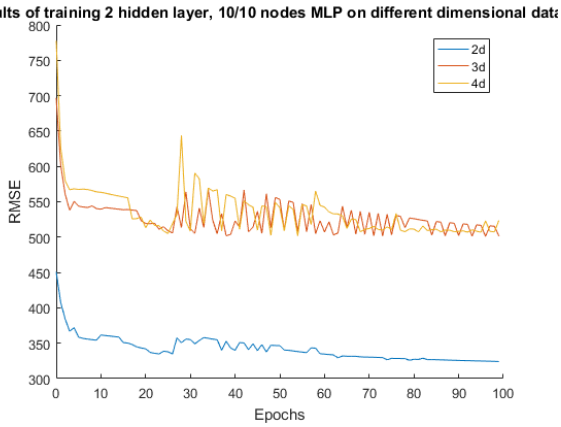

Figure 8: Typical curves of RMSE over different dimensional data trained by the same net.

where $n = 2$. This is most likely comes down to Occam's Razor in that with the information we provided the simplest solution often works best; the more complex networks were not able to perform as well due to the added difficulty to achieve a proper training.

Another one of our hypotheses was that MLPs with fewer nodes per layer would out perform MLPs with similar amounts of nodes but fewer layers due to training the former MLPs should be easier and able to generalize more efficiently. In our results, as seen in table and visualized in figure 3, we can see the roughly similarly for grid sampling and only a small difference in random sampling. This results in the inability to make any significant statements when evaluating the group's hypothesis.

For the RBFs, we hypothesized that the number of basis functions would have an inverse relationship between accuracy and convergence. This proved to be a mostly correct hypothesis as seen in figure 3. When comparing the RBFs against each other we could see that as the number of basis functions increased that as the RBF was trained the higher number epochs would start to slow the speed of convergence if not start to diverge. As stated previously, due to lack of solid data for the higher dimensional cases no meaningful conclusions can be made about the accuracy compared to convergence for said cases.

Overall, this exploration of function approximation using neural networks proved moderately successful. The lower dimensional problems provided enough data to make meaningful conclusions and testing for the implemented algorithms, however much more testing and research could be done for the higher dimensional problems and more complex neural networks.

References

Durtka Baker and Wang. Neural networks in function approximation, 2017.

Benoudjit and Verleysen. On the kernel widths in radial-basis function networks. *Neural Processing Letters*, 18(139), 2003.

Broomhead and Lowe. Radial basis functions, multi-variable function interpolation and adaptive networks, 1988.

Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3), 1960.

Hinton Rumelhart and Williams. Learning represesntations by back-propagating errors. *Nature*, 323(9), 1986.