Lab #2: RBF Network

March 1st, 2011
Andrew D Yates
CSE 779, WI 2011

Table of Contents

Abstract

A Radial Basis Function Network, or RBF network, is an artificial neural network that uses a linear combination of radial basis activation functions to generalize arbitrary functions. In this lab, a collection of RBF networks using a variety of parameters are tested to approximate a single control function. Then, we systematically modify learning rate, number of basis, and Gaussian width function parameters of this network to test how network performance is related to these network parameters.

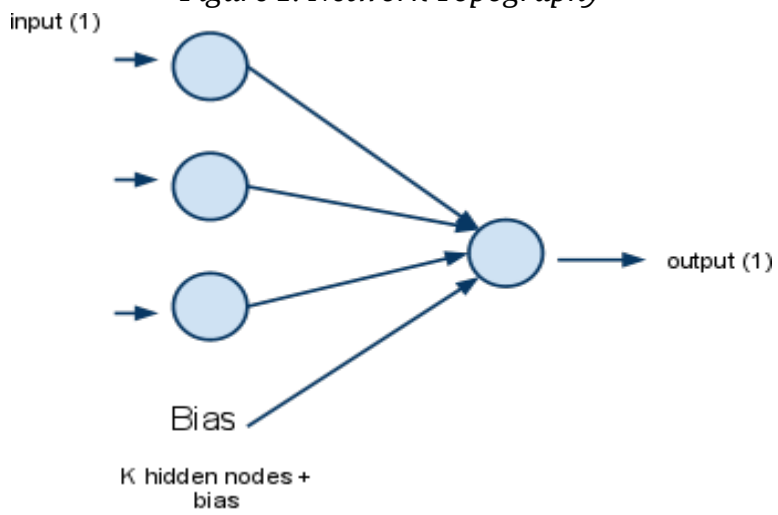
Method

The neural network in this lab is defined as follows:

- 1 input node
- 1 hidden layer of K radial basis function nodes
- 1 output node with one output bit
- Activation function $\phi = exp(-1/2[?][?]$ with $a = 1$
- $\phi = exp(-1/2[?][?]^2 \| c - x \|)$ where $[?][?]$ is the Gaussian width and c is the Gaussian center of neuron j.

- Network topography as in Figure 1
- All weights initialized to random numbers between -0.5 and 0.5

Figure 1: Network Topography



This network was programmed using the Python programming language on OSX and executed in the operating system terminal. Graphing was performed using the *matplotlib* Python library. See the attached source code in the appendix as Appendix C: Source Code.

The function to be approximated was:
$h(x) = 0.5 + 0.4sin(2\pi x)$
This function was sampled for 75 data points sampled randomly from a uniform distribution in the interval [0, 1]. To model actual measurements, uniform noise in the interval [-0.1, 0.1] was added to the function output to generate the desired function output used in network training. Also, another 75 data points were sampled without added noise to be used as a generalization validation sample.

To generate results, the neural network was trained over the epoch of 75 data points for 100 epochs. At this point, training ended, and the training handle generated a collection of statistical results including most notably a plot of the results, the network's mean training error, and the network's mean generalization error. The training handle then tested and compiled results for 12 samples approximated by 3 networks for each of the 20 permutations of the parameters:

eta = (0.01, 0.02)
K = (5, 10, 15, 20, 25)
f_width(variance, shared)

where f_width is the Gaussian width generation function such that:

variance computes σ^2 as the variance of each cluster

shared computes σ^2 as the square of $d_{max}/\sqrt{2K}$ for all clusters

To record the results and to monitor real time network training, the network training program regularly printed network statistics to STDOUT. Also, plots of training results were regularly saved to disk. After completion of the testing handle, the console output and set of graphs were analyzed to generate conclusions about testing performance as it relates to network parameters.

Results

Generally, the RBF network best generalized the desired output function for about K=10 bases using the *shared* Gaussian width function. Because the RBN network is local to the input and quickly converges, changes in eta had little significance in reducing generalization error, although a greater eta tended to decrease testing error, especially when K was relatively large. Further, while adding additional bases beyond 10 tended to slightly decrease training error, generalization error tended to increase or did not improve, indicating network over-training. This was especially evident in the case of the variance Gaussian width functions.

On visual inspection of plotted network approximations, the networks using shared Gaussian width functions were smooth, while the networks using the variance Gaussian width function oscillated wildly around the ideal output function with the frequency of oscillation positively correlated with K. Clearly, the Gaussian widths of the variance Gaussian computed widths were too narrow for this function and did not sufficiently overlap to generate acceptable approximations.

On that theme, by far, the most critical parameter to generalization and training success was application of the shared Gaussian width function as opposed to the variance Gaussian width function. Next, the most important parameter was to choose a “medium” K number of bases. Values of about K => 5 and K <= 20 best generalized the function, and values of about K=10 also best generalized the shape of the function on visual inspection. Given the application of the shared Gaussian width function, approximations of K < 10 tended to not curve enough to best fit the ideal function, while approximations of K > 10 tended to curve too much to best fit the ideal function. Further, approximations of K => 20 tended to over-fit the noisy training sample of the function rather than generally approximate the ideal function as intended.

Finally, specific training and generalization performance was dependant on the randomized features of the training set and the stochastic network fitting algorithm. Occasionally, a cluster of outliers or a unfortunately selected set radial bases would train an unintended feature into the network approximation which was not represented in

the ideal function.

See “Featured Plots” for notable examples of plots mentioned above.

Conclusion

RBF networks can be trained to approximate functions, but they require tuning and cross-validation to achieve the best general performance because model selection is as important if not more important than training algorithm performance. First, a reduction in training error does not necessarily reduce generalization error and may increase generalization error. For this reason, and because RBF networks tend to converge after a relatively few epochs as compared to back-propagation neural networks, increasing eta has little significance on generalization performance as compared to changing other model parameters. Second, for best generalization, the number of bases should be “high enough but not too high” which can be determined using cross-validation over a range for K values of some fraction of the sample size. In this case, K=10 for N=75 was judged to be a favorable model parameter. Finally, the most important parameter in network performance was the selection of the *shared* Gaussian width function over the *variance* Gaussian width function. Not only was the shared function easier to compute and easier to about which to mathematically reason, but also the shared Gaussian width function helped ensure that individual Gaussian units would not be too narrow (as in this case) or too wide (as could be conceived for other functions). A shared, regular Gaussian width most drastically improved network performance in this case.

Featured Plots

Figure 2: Best “Variance Weight”

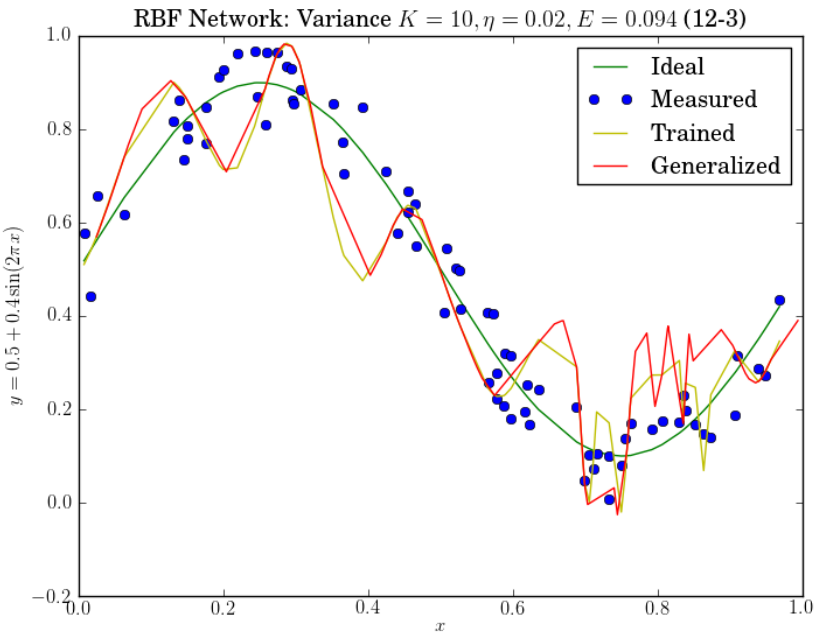


Figure 3: Best “Shared Weight”

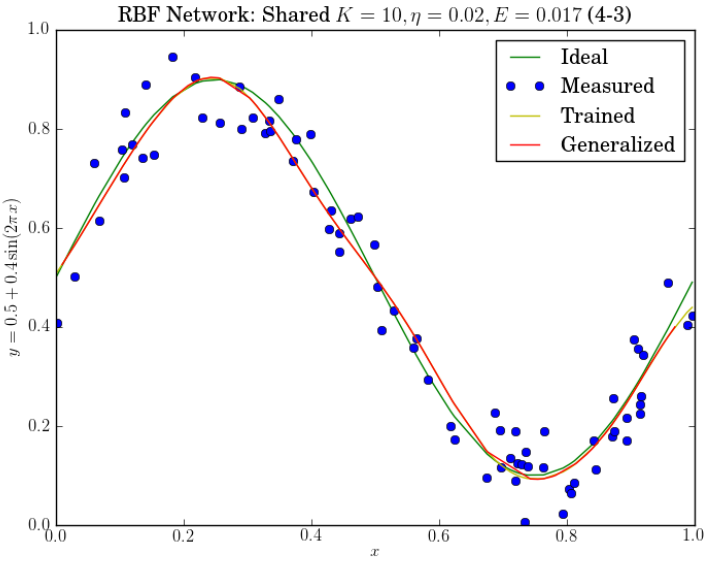


Figure 4: Terrible performance, low K

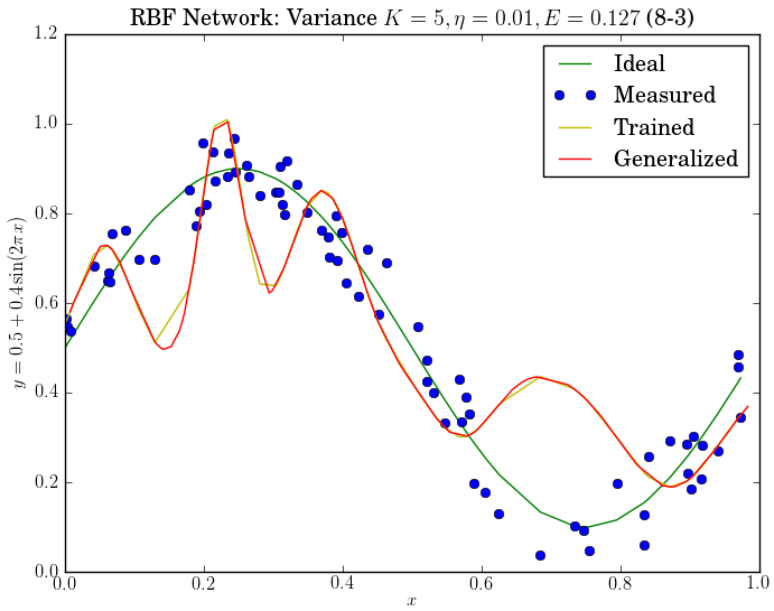


Figure 5: Terrible performance, high K

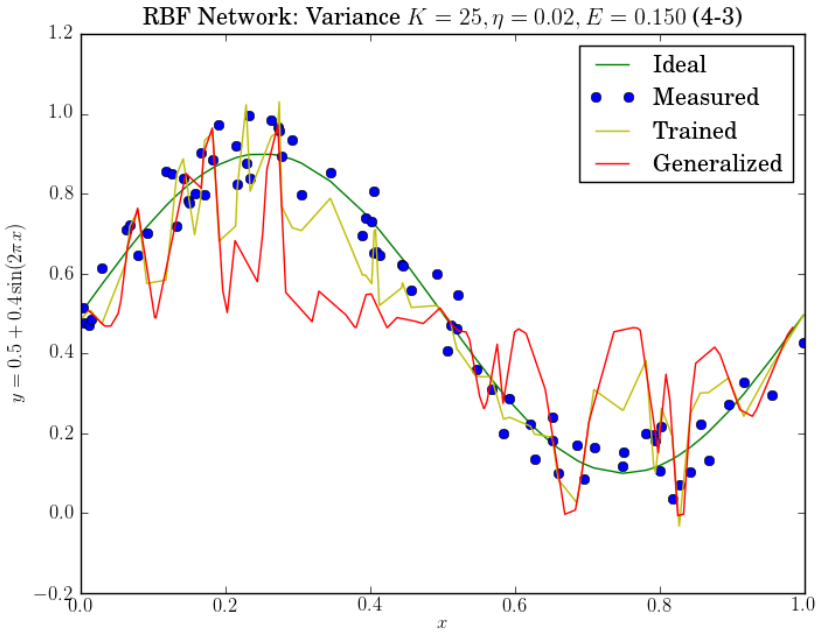


Figure 6: Underfit, low K

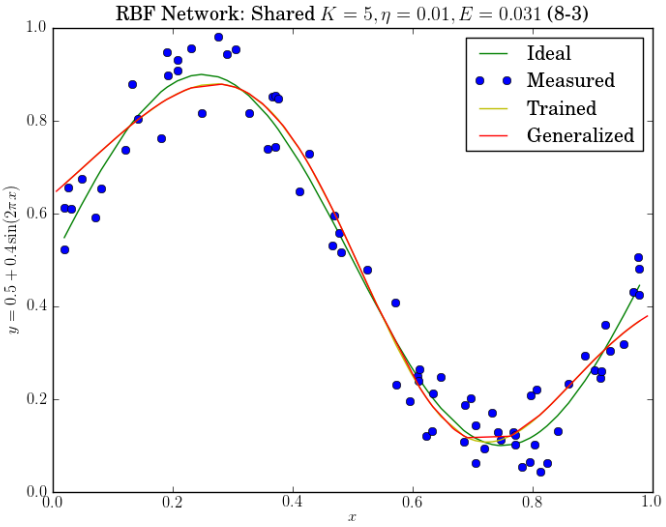


Figure 7: Overfit, high K

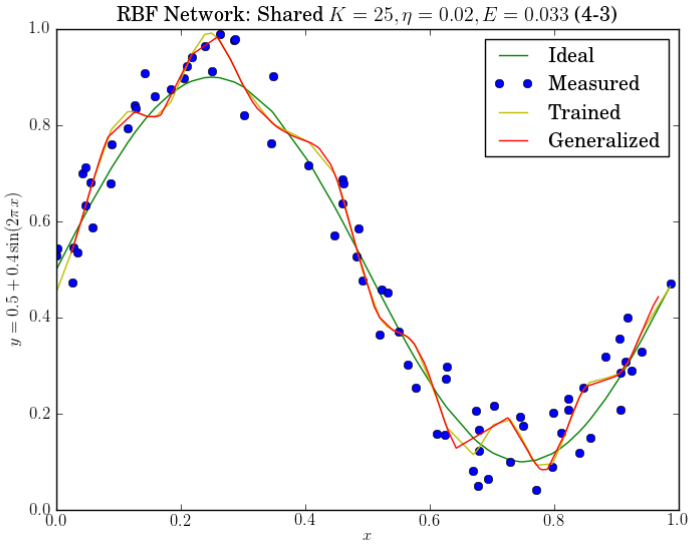
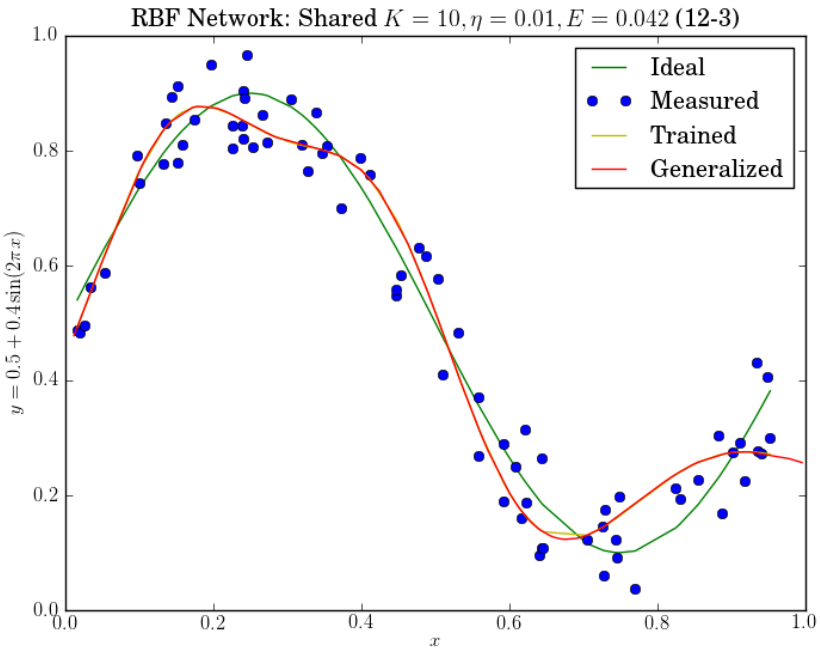


Figure 8: Sampling outliers and stochastic properties add superfluous feature to estimate for otherwise favorable parameters.



Attachments

The following are included as attachments:

1. **Attachment A**: Source Code: "rbn.py"
2. **Attachment B**: List of Testing Results: "formatted results.txt"
3. **Attachment C:** Program Console Output: "rbn console output.txt"
4. **Attachment D**: Collection of Network Plot Images