Project 2 – Radial Basis Function Networks

Daniel Marchese

Introduction

The purpose of this project was to implement a radial basis function network from scratch to perform function approximation. The approximated function was fixed to a sine wave in the interval [0, 1]. The network was trained using the LMS algorithm to approximate the function using training data with artificially generated noise.

Project Layout

The project is implemented as a simple Python package. The logic for driving the network can be found in *driver.py*. Utilities such as the k-means implementation can be found in the *util* folder. The actual implementation for the RBF Network can be found in the *networks* folder.

Implementation

The program was implemented in pure Python 2.7 consisting of one class and several utility functions inside the driver. The class is implemented in the networks package as RBFNetwork. The actual logic for driving the function network exists in this class, which contains a feed method for retrieving output for a given input, and a train method which will train the network on the given input and desired output. Construction of the network takes a list of Gaussian centers and their accompanying variance.

Additionally, K-Means was implemented for finding the centers of the Gaussians. The implementation follows the classic design pattern for K-Means by constantly updating the picked centroids until they do not change between two iterations. Once the centroids were picked, there were two methods for calculating the corresponding variances. The first method was to base the variance on the points that were closest to each centroid using equation 1 below:

$$\sigma_j^2 = \frac{1}{\|C_j\|} \sum_{i \in C_j} \|\boldsymbol{u}_j - \boldsymbol{x}_i\|^2 \quad \textbf{(1)}$$

where σ_j^2 is the variance for centroid j, $\|C_j\|$ is the size of the cluster in question, $\boldsymbol{u}_j$ is the location of the centroid for the cluster in question, and $\boldsymbol{x}_i$ is the input vector.

The second method was to base the variance on the overall placements of the clusters using equation 2 below:

$$\sigma^2 = \frac{d_{max}^2}{2K} \quad \textbf{(2)}$$

where σ^2 is the variance used for all Gaussians in the RBF network, d_{max} is the maximum distance between Gaussian centers, and K is K value used in the K-Means algorithm.

Results

Overall, I found the results to be rather surprising. For each trial with different numbers of bases, η value, and calculation method for σ^2, I graphed the results and calculated the mean-

squared error. The MSE values are shown below in table 1, the graphs can be found at the end of the report in figures 1 and 2. (the first set is with variance calculated for each Gaussian center, the second set with a fixed variance value).

	$K = 2$	$K = 4$	$K = 7$	$K = 11$	$K = 16$
Varying σ^2, $\eta = 0.01$	0.0062	0.0135	0.0205	0.0112	0.0134
Varying σ^2, $\eta = 0.02$	0.0048	0.0154	0.0132	0.0112	0.0073
Constant σ^2, $\eta = 0.01$	0.0131	0.0094	0.0137	0.0058	0.0035
Constant σ^2, $\eta = 0.02$	0.0076	0.0079	0.0061	0.0044	0.0032

Table 1: Mean-Squared-Error values with varying variance calculation, learning rate, and number of Gaussian centers.

The mean-squared-error for the ground truth function on the given input points was 0.0033.

Discussion

There are 3 main variables in this lab that dramatically change the results: the method of choosing σ^2, the chosen value of K, and the chosen value of the learning rate: η.

When varying the method of choosing σ^2, there is a very noticeable difference in the shape of the final approximated function. With a constant variance for all of the Gaussian centers, the graphs are much smoother in appearance, and tend to match the dataset with higher fidelity. With a unique variance chosen for each Gaussian center, the resulting approximation tends to be much more jagged for increasing K. However, it should also be noted that the choice in η did have a noticeable effect on the eventual accuracy of the network after 100 epochs with the constant sigma value. The difference was much less noticeable with the varying variances.

When varying the number of bases used in the approximation, the overall accuracy depends on the choice of variance for each Gaussian. With a unique variance per Gaussian, the over-fitting problem becomes very clear with $K > 2$. The problem is much less apparent with constant choice of σ^2.

I was most surprised by the striking difference between approximations when changing the calculation method for variance on each basis function. However, it makes sense since the larger variance makes wider bases, which provides more ways for the different functions to add together in a way that best fits our test data.

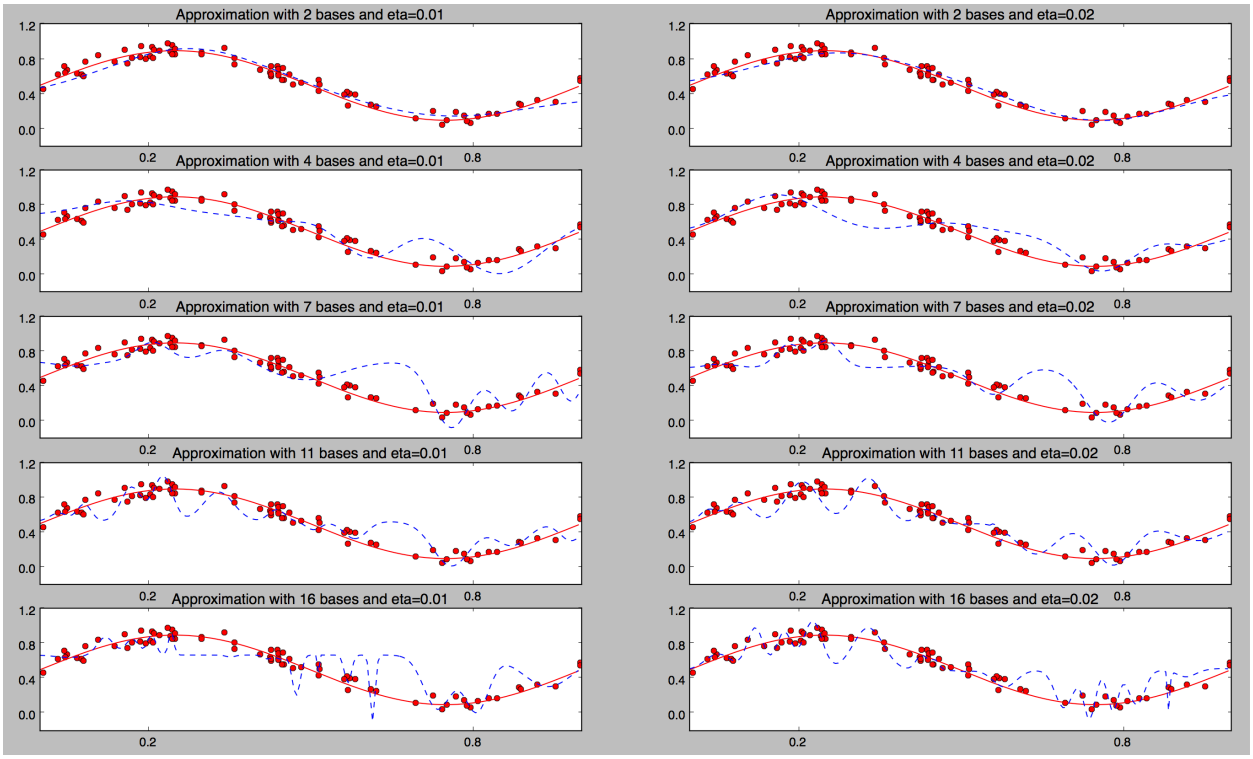

Figure 1: Graphs showing the results of training the network with a varying number of bases using the local method for calculating the variance on each basis.

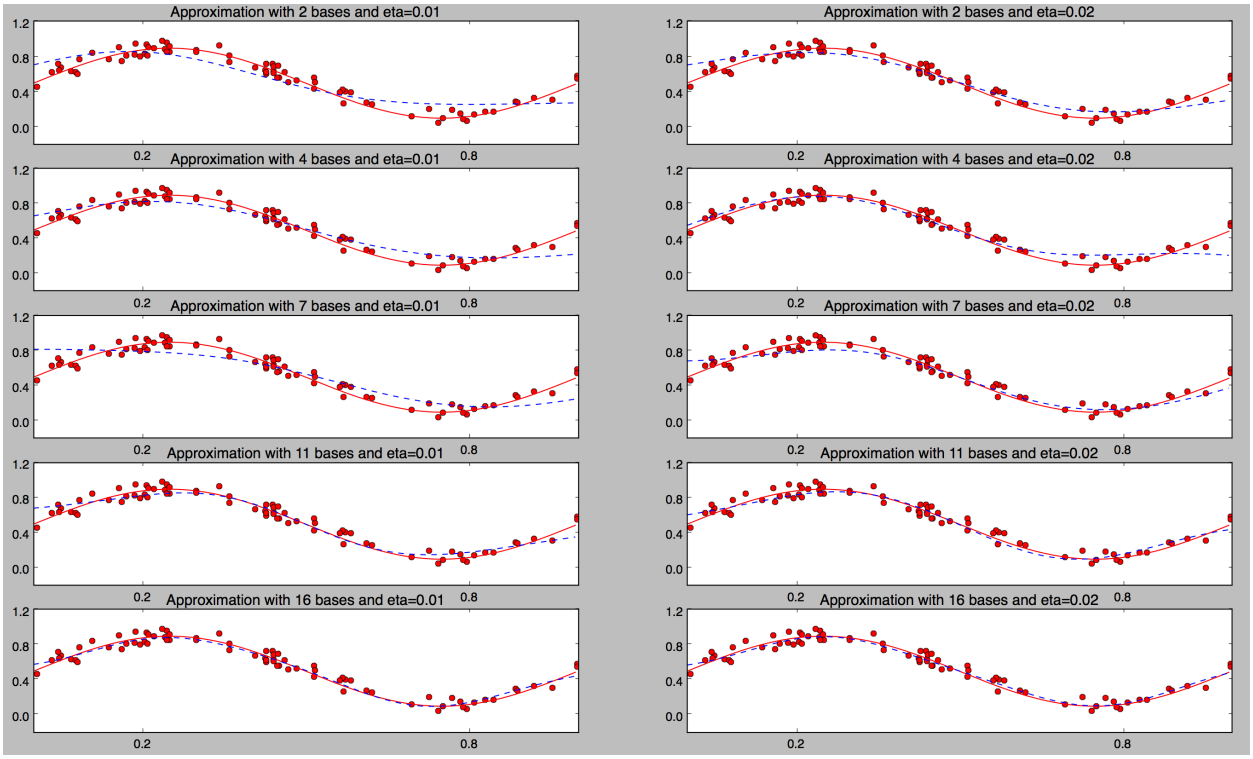

Figure 2: Graphs showing the results of training the network with a varying number of bases using the global method for calculating the variance on each basis.