Programming Assignment 2 Summary Report

For this assignment, an RBF network was trained to learn the function $h(x) = 0.5 + 0.4\sin(2\pi x)$. The RBF uses the k-means algorithm to learn gaussian centers, then LMS to learn the weights of each gaussian. The source program can be tested by running: “python main.py”

Training Data

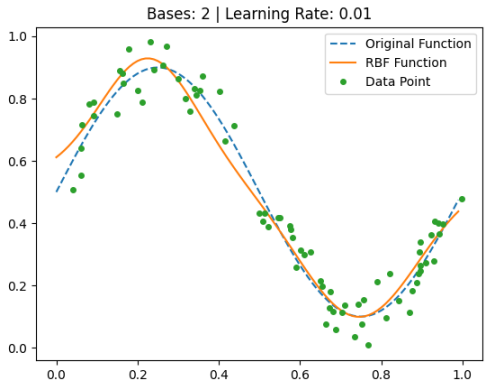
Data was generated in “data_gen.py”, by randomly sampling x values, plugging them into h(x), then adding random noise to create the labels.

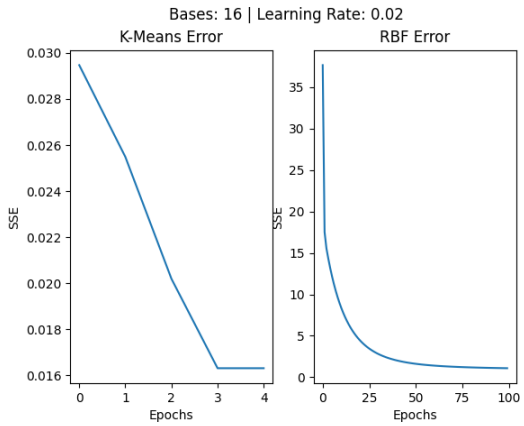
K-Means

The k-means algorithm was implemented as a class with a *train* method which generates the converged gaussian bases and variances. Bases are initialized as random points from the input data, and the algorithm runs until convergence. Every case that was tested converged within 4 epochs. A sample chart displaying the error function over time is shown below; all charts are attached along with this submission in the “plots/diff-variance” folder.

RBF

The RBF algorithm was also implemented with a *train* and *predict* method, and would be initialized with the bases and variances calculated from the k-means implementation. I also initialized weights for each base and a bias, which was optimized with the LMS algorithm. A sample chart displaying the error function is also shown below. Also displayed is a sample graph comparing the data points, original function, and RBF approximation. All charts are attached along with this submission in the “plots/diff-variance” folder.





Results

The network is very dependent on the number of bases. The RBF approximation for 2 bases, for example, is much closer to the target function compared to any other number of bases. This means that models with a higher number of bases tend to overfit this sine function. It also is a good application of Occam's razor - the simplest model that is consistent with the data uses 2 bases, so it is the model we should use.

Learning rate also has an effect on the convergence in the RBF networks. Even though we only trained each network for 100 epochs, the networks that used a higher training rate saw sharper drops in error, and resulted in a better approximation of the original function.

Simplified RBF Network

To simplify the RBF network, the implementation was modified so that all clusters had the same variance. This resulted in the higher number of bases actually performing better. This is because a low number of bases likely has a large d_{max} value, resulting in gaussian functions with large variance. On the other hand, a higher number of bases are more likely to have a smaller variance allowing for more fine tuning of the gaussian weights.

In addition, while in the first approach the function approximations varied very largely and often overfit, this approach maintained a good approximation throughout. It seems that the first method with a variance for each cluster works better for low numbers of bases, and the same variance approach allows you to use a higher number of bases. Results for the simplified RBF network are stored in the “plots/same-variance” folder.