

Continuous Optimization of Radial Basis Function Centers

Matt McDaniel

Abstract

Radial basis functions are a useful tool in both high dimensional interpolation and function approximation. They provide arbitrarily complex function representations, which can also be very easy for humans to interpret. However, the task of selecting RBF centers is non-trivial leading much of the literature to use approximate or heuristic approaches, often resulting in less than optimal solutions. In this paper we present an analytical approach to calculating the gradients of RBF networks with respect to function centers, which shows favorable results in comparison to alternate RBF techniques as well as multilayer perceptrons and support vector machines.

Background

A radial basis function network is an artificial neural network that uses radial basis functions as activation functions. Each hidden node takes the input vector multiplied by a set of weights and calculates the distance from that node's center. This distance is fed into another function to calculate the strength of activation at that node. Using a Gaussian radial basis function is very common. This results in node activations for inputs that are “near” the node center. There are 3 types of parameters that can be optimized: the input weights (equivalent to the widths of the RBF functions), the center locations of the RBF functions, and the weights mapping the RBF activations to the final output.

The most common approach for function approximation is to normalize the input beforehand and then used fixed input weights set to a fraction of the distance between the farthest 2 training points. Then the function centers are selected either as a random subset of the training data or taken as the cluster centers outputted by an algorithm such as k-means clustering. Once the radial basis functions are fixed, calculating the output weights is a linear least squares problem.

Radial basis functions can also be normalized, that is all RBF activation results scaled so that they sum to one. Then RBF networks can be viewed more as a method for interpolating between a set of input-output pairs than as a type of artificial neural network. A function center can be interpreted as an example input with it's output weights being the corresponding example output. The mathematics remain unchanged, but this view leads to a much more natural interpretation of resulting models.

Motivation

This project began as an extension to using nearest neighbor searches for example-based super resolution, also know as image hallucination. In the process of image hallucination a database is built mapping low resolution image patches to high resolution patches. Then during application low resolution patches are extracted from an image, a nearest neighbor search is performed to find one or more best matching high resolution patches and the low resolution patch in the image is replaced with a higher resolution patch.

It takes a large number of raw image patches to build a good model, and image patches are often noisy requiring many neighbors to be drawn and combined. Thus a reasonable approach is to attempt to create a smaller database of “representative” patches, which helps to guard against noise and greatly improves efficiency. One method for doing this is to use k-k-means clusters of the input data, and then interpolate between them using normalized radial basis functions. This is mathematically equivalent to a radial basis function network.

However, in the case of image hallucination there are hard boundaries in the output set that do not appear as strongly when clustering solely on the input data. There may be two patches which appear quite similar in their low resolution versions, but in the higher resolution version one is an edge, while the other is a local texture. What we would like is a selection of representative samples that takes into account patterns in the output as well as in the input. More generally, we know in advance the

model of how our samples will be interpolated, so why not select samples so that that model minimizes the error over our training data?

Approach

So, how do we find representative samples such that interpolating between them gives the best approximation to our data? First we look at the general form of minimizing error under a linear mapping following a change of basis. Let L be our learned library divided into L_I , the input library, and L_O the output library. The change of basis performed is given as the coefficient function c of the input library and the input vector x , with the desired result being the vector y . We seek to perform standard gradient based optimization techniques to achieve a locally optimal library. Since library entries are interchangeable there cannot be a unique globally optimal solution, and we know the function is not convex.

The first significant issue arises in the requirement of the derivative of the coefficient function with respect to the input library. The coefficient library is a vector-valued function, which must be derived with respect to a matrix, resulting in an order 3 tensor. However, this problem can be avoided by only considering single jacobian slices taken with respect to one function center at a time. This general form is still quite expensive to calculate, but as we will show, we can often take advantage of structure in the derivative of the coefficient function to greatly improve performance.

Next we consider a non-normalized set of radial basis functions as our coefficient function. In this case the jacobian slices of interest turn out to be diagonal matrices. Without normalization the location of a function center only affects the activation of that function. This also means we only need to consider that function's corresponding output library entry when calculating the gradient of the input library entry. Thus the final derivative requires only one RBF evaluation, one dot product, and one scalar vector multiplication per library entry per data point.

Finally, we consider the normalized radial basis function case. Unfortunately, since every coefficient requires division by the activations of all basis functions this derivative is completely dense. However, the gradients of each coefficient with respect to moving a library entry are all linearly dependent. When we increase the activation of one function we decrease the activation of all the other functions by that same amount in proportion to how activated they were. We can take advantage of the rank 1 nature of our coefficient derivatives by calculating their vector factors W_j and V_j , and then making use of the associativity of the final formula

$$\begin{aligned} L_O * c(L_I, x) &= y \\ F(L) &= \sum_i (L_O c(L_I, x_i) - y_i)^2 \\ \frac{\partial F}{\partial L_O} &= 2(L_O c - y) c^T \\ \frac{\partial F}{\partial L_{I,j}} &= [2 L_O \frac{\partial c}{\partial L_{I,j}}]^T (L_O c - d_o) \end{aligned}$$

$$\begin{aligned} \text{if } c_j &= s((L_{I,j} - x)^2) \text{ then} \\ \frac{\partial c_j}{\partial L_{I,j}} &= 2 s'((L_{I,j} - x)^2) (L_{I,j} - x) \\ \frac{\partial c_k}{\partial L_{I,j}} &= 0 \text{ for } j \neq k \\ \text{thus} \\ \frac{\partial F}{\partial L_{I,j}} &= \frac{\partial c_j}{\partial L_{I,j}} L_{O,j}^T (L_O c - y) \end{aligned}$$

$$\begin{aligned} \text{if } c_j &= \frac{s((L_{I,j} - x)^2)}{\sum_i s((L_{I,i} - x)^2)} \text{ then} \\ \text{Let } a_j &= s((L_{I,j} - x)^2), b_j = \sum_{i \neq j} s((L_{I,i} - x)^2) \\ \frac{\partial c_j}{\partial L_{I,j}} &= \frac{2 b_j (L_{I,j} - x) s'((L_{I,j} - x)^2)}{(a_j + b_j)^2} \\ \frac{\partial c_k}{\partial L_{I,j}} &= \frac{-2 a_k (L_{I,j} - x) s'((L_{I,j} - x)^2)}{(a_j + b_j)^2} \text{ for } j \neq k \\ \text{Let } V_j &= \frac{(L_{I,j} - x) s'((L_{I,j} - x)^2)}{[\sum_i s((L_{I,i} - x)^2)]^2} \\ \text{Let } W_{j,k} &= \begin{cases} -2 s'((L_{I,k} - x)^2) & k \neq j \\ -\sum_{i \neq j} W_{j,i} & k = j \end{cases} \\ \text{thus} \\ \frac{\partial c}{\partial L_{I,j}} &= W_j V_j^T \\ \frac{\partial F}{\partial L_{I,j}} &= 2 V_j W_j^T L_O^T (L_O c - y) \end{aligned}$$

rather than ever directly forming the jacobian.

We can precalculate the affect of changing coefficients on the final output by multiplying the transpose of the output library by the error vector once for each data point. Then when we need to multiply this by the derivative of the coefficient function we first multiply W_j transpose which is equivalent to a dot product. Then we multiply the resulting scalar by V_j avoiding any matrix multiplications in the innermost loop. Thus we have one potentially large matrix multiply per data point with one dot product and one scalar vector multiplication per library entry per data point.

Hypothesis

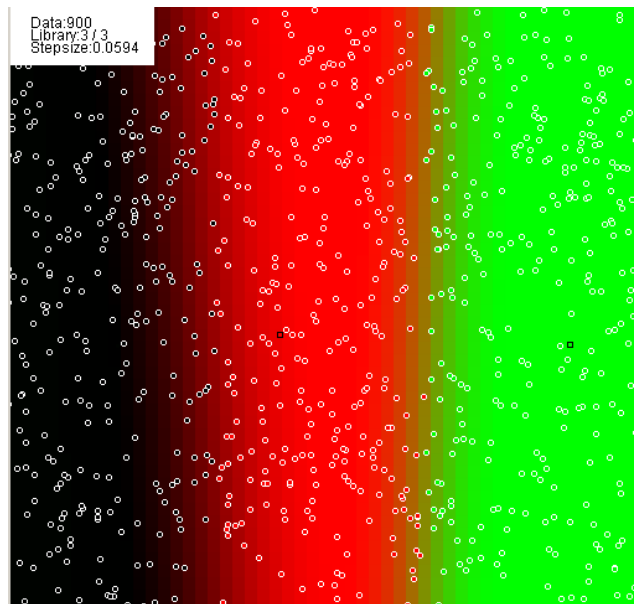
Since my algorithm begins with library entries calculated by using the standard clustering approach, and then performs optimization it will almost surely reduce error on a training set in comparison to RBF algorithms which do not perform the final optimization step. Since this step merely replaces the inductive bias implied by clustering I do not expect it to generalize any worse than that approach, and so I would expect my algorithm to perform at least slightly better than classical RBF center selecting approaches on almost all data.

Although my algorithm is designed for interpolation-like regression problems I also intend to test it on several classification problems. I will be comparing to support vector machines, random forests, and multilayer perceptrons. Although radial basis function networks are often portrayed as a subset of neural networks they actually share more in common with support vector machines that use RBF kernels than they do with multilayer perceptrons. I expect my algorithm will perform somewhat similarly to a support vector machine as a result, however, I do not know which will perform better. Support vector machines consider kernels with all data points and use a maximum margin classifier with slack to protect against over-fitting. My algorithm uses a significantly smaller number of functions to begin with, so over-fitting is limited by the complexity of the function itself.

Empirical Evaluation

Before testing on any repository data sets I created several small tests to illustrate that my algorithm does actually detect output patterns that are not present in the input data. One such test is pictured below. Data points are distributed randomly over the screen with those in the left third colored black, the middle third colored red, and the right third colored green. The algorithm is given a library of 3 entries and tasked with finding the examples that best map screen location to color. The background color shows the output of the learned function, while the black squares show the examples learned (the left square is not visible).

This example is somewhat contrived, but it illustrates precisely the type of pattern the standard center selection methods cannot detect. A method using k-means clustering for center selection distributes the three examples in a large triangle with no regard for how outputs are distributed. Given a sufficient number of library entries the standard approach will detect a reasonable approximation to the pattern, but using more basis functions decreases function evaluation speed as well as allows much more potential for over-fitting.



I ran three regression tests using popular data-sets from the UCI machine learning repository

comparing my algorithm to the radial basis function network implementation in the Waikato Environment for Knowledge Analysis (WEKA) machine learning tools. Three to five iterations of ten-fold cross validation were used for each test to reduce variance in the final errors calculated. The same radial basis function (Gaussian), number of library entries/hidden nodes and function widths were used between my algorithm and the WEKA algorithm. As expected my algorithm performed better on all three of these tests.

Mean Absolute Error on UCI ML Repository Regression Tests

Algorithm	Forest Fires	White Wine	Red Wine
My Algorithm	18.0600	0.5601	0.4900
WEKA RBF	18.7500	0.6422	0.5710

I also ran four classification tasks comparing my algorithm with the WEKA implementations of multilayer perceptrons, support vector machines and random forests. Random forest outperformed all other algorithms by a significant margin on the wine quality data-sets, however none of the other differences are quite that clear-cut. My algorithm outperformed multilayer perceptrons on 2 of 4 tests, and support vector machines on 3 of 4 tests. The only task for which support vector machines did best was also the only test limited to only two classes (Wine had 6, Iris had 3). This makes some sense as the other algorithms, mine included, extend more naturally to many class problems.

Accuracy on UCI Repository Classification Tests

Algorithm	White Wine	Red White	Iris	Breast Cancer
My Algorithm	53.78%	62.96%	97.06%	96.61%
WEKA MLP	54.76%	59.72%	96.64%	97.18%
WEKA SMO SVM	52.08%	58.04%	95.75%	97.54%
WEKA Random Forest	67.21%	70.36%	94.63%	95.42%

Future Work

While the derivative formulas presented in this paper are complete and a simple gradient descent guarantees convergence to a local minimum there are many improvements that could be made towards the efficiency of the learning algorithm and the final function application. Most obviously more sophisticated optimization techniques could be applied to achieve much faster convergence. While I do not recommend calculating the second derivatives analytically, I expect methods that numerically estimate the effect of the second derivative such as BFGS or its limited memory variant would greatly reduce run-time.

The current version scales linearly in the number of library entries being trained, however this is not strictly necessary for large library sizes. Typically, only a hand full of functions will have non-negligible activations for any one data point, and all other functions can be safely ignored in both the training process and applications. Determining the points with non-negligible activations is very similar to a k-nearest neighbor search, which can be performed very efficiently even in large numbers of dimensions using vantage point trees.

Since the training is geared toward minimizing the sum of squared error with the effect on the gradient calculated separately for each data point. This technique, as with many machine learning techniques, is embarrassingly parallel. Also, like many machine learning algorithms, it is susceptible to getting stuck in local minima as well as over-fitting if given more functions than necessary. Thus it

would be an interesting project to see what improvements could be gained by building ensembles of these models.

References

