Fully Supervised Training of Gaussian Radial Basis Function Networks in WEKA

Eibe Frank
Department of Computer Science
University of Waikato

July 2, 2014

1 Introduction

Radial basis function networks are a type of feedforward network with a long history in machine learning. In spite of this, there is relatively little literature on how to train them so that accurate predictions are obtained. A common strategy is to train the hidden layer of the network using k-means clustering and the output layer using supervised learning. However, Wettschereck and Dietterich [2] found that supervised training of hidden layer parameters can improve predictive performance. They investigated learning center locations, local variances of the basis functions, and attribute weights, in a supervised manner.

This document discusses supervised training of Gaussian radial basis function networks in the WEKA machine learning software. More specifically, we discuss the `RBFClassifier` and `RBFRegressor` classes available as part of the `RBFNetwork` package for WEKA 3.7 and consider (a) learning of center locations and one global variance parameter, (b) learning of center locations and one local variance parameter per basis function, and (c) learning center locations with per-attribute local variance parameters. We also consider learning attribute weights jointly with other parameters.

2 Models and Optimization Methods

We tackle both classification and regression problems. `RBFClassifier` trains a model for classification problems, `RBFRegressor` trains a regression model. In both cases, penalised squared error, with a quadratic penalty on the non-bias weights in the output layer, is used as the loss function to find the network parameters. As the two methods share a substantial amount of code, this common code has been extracted out into a super class called `RBFModel`.

The Gaussian radial basis function model learned by the `RBFRegressor` class is

$$f(x_1, x_2, ..., x_m) = g(w_0 + \sum_{i=1}^{b} w_i \exp(-\sum_{j=1}^{m} \frac{a_j^2 (x_j - c_{i,j})^2}{2\sigma_{i,j}^2})), \quad (1)$$

where $x_1, x_2, ..., x_m$ is the vector of attribute values for the instance concerned, $g(.)$ is the activation function, b is the number of basis functions, w_i is the weight for each basis function, a_j^2 is the weight of the jth attribute, and $c_{i,.}$ and $\sigma_{i,.}^2$ are the basis function centers and variances respectively. The activation function $g(.)$ is the identify function in the regression case.

In the classification case, for `RBFClassifier`, there is one output unit per class value, and the model learned for the lth output unit (i.e., class value) is:

$$f_l(x_1, x_2, ..., x_m) = g(w_{l,0} + \sum_{i=1}^{b} w_{l,i} \exp(-\sum_{j=1}^{m} \frac{a_j^2 (x_j - c_{i,j})^2}{2\sigma_{i,j}^2})), \quad (2)$$

where the activation function $g(.)$ is the logistic function.

The above models are the most complex ones that the implementations can learn, with attribute weights as well as attribute-specific per-basis-function variances $\sigma_{i,j}^2$. By choosing appropriate parameter settings, simplifications of this model can be used instead. By default, attribute weights are not used, i.e., all a_j^2 are fixed at 1, and there is just one global variance parameter, i.e., $\sigma_{i,j}^2 = \sigma_{global}^2$. Instead of using one global variance parameter or $m \times b$ variance parameters, it is also possible to use a different variance parameter per basis function that is the same for all attributes, i.e., $\sigma_{i,j}^2 = \sigma_i^2$. In this case, there are only b variance parameters.

Appropriate settings for the parameters $w_{(l,)i}$, a_j^2, $c_{i,j}$, and $\sigma_{i,j}^2$ are found by identifying a local minimum of the penalised squared error on the training data. In the regression case, this error function is:

$$L_{SSE} = (\frac{1}{2} \sum_{i=1}^{n} (y_i - f(\vec{x}_i))^2) + (\lambda \sum_{i=1}^{b} w_i^2), \quad (3)$$

where y_i is the actual target value for training instance $\vec{x}_i$, and the first sum ranges over all n training instances. The parameter λ is the "ridge" parameter that determines the size of the penalty on the weights and can be specified by the user to control overfitting.

In the classification case, assuming k classes, the error function becomes:

$$L_{SSE} = (\frac{1}{2} \sum_{i=1}^{n} \sum_{l=1}^{k} (y_{i,l} - f_l(\vec{x}_i))^2) + (\lambda \sum_{l=1}^{k} \sum_{i=1}^{b} w_{l,i}^2), \quad (4)$$

where $y_{i,l} = 0.99$ if instance $\vec{x}_i$ has the lth class value, and $y_{i,l} = 0.01$ otherwise. The values 0.99 and 0.01 are used instead of 1.0 and 0.0 to aid the optimisation process. Additionally, the `RBFClassifier` implementation divides L_{SSE} by

n, the number of training instances, as this was found empirically to improve convergence with the optimisation methods employed. This was not necessary in the regression case.

The gradients of the error functions with respect to the network parameters consist of the corresponding partial derivatives, which can be found using standard calculus. Calculating the partial derivatives involves error backpropagation in the same manner as in multilayer perceptrons.

In the implementations, two gradient-based methods are available for optimising the parameters so that the error functions are minimised: the quasi-Newton method using BFGS updates implemented in WEKA's `Optimization` class, and a non-linear conjugate gradient descent method, implemented as a new `ConjugateGradientDescentOptimization` subclass of the `Optimization` class. This new class implements the hybrid conjugate gradient descent method specified by Equation 1.25 in Dai and Yuan [1]. This method has been shown to be globally convergent when used with a weak Wolfe line search such as the one implemented in the `Optimization class`.

Before training begins, all numeric attributes in the data are normalized to the $[0, 1]$ interval, including the target attribute in the regression case (which is projected back into the original space when predictions are made). Missing values are replaced by the mean (for numeric attributes) or mode (for nominal ones). Constant attributes are removed and nominal attributes are binarised. The same steps are performed for new instances when predictions are to be made.

Another important aspect of the learning process concerns the initialisation of the network parameters. In the classification case, the initial weights of the output layer are sampled from $\mathcal{N}(0, 0.1)$. In the regression case, these initial weighs are sampled from $U(-0.25, 0.25)$. These sampling strategies were determined empirically based on the well-known heuristic of choosing small, randomly distributed initial weights.

A more complex process is used to initialise the hidden unit centers and variances. More specifically, as k-means is frequently used to train the hidden layer of an RBF network in an unsupervised manner, and because it is very fast, WEKA's k-means implementation `SimpleKMeans` is used to establish initial hidden unit centers, i.e., the cluster centers found by k-means are used to initialise the $c_{i,j}$. Moroever, the initial value of all variance parameter(s) $\sigma_{i,j}^2$ in the network is set to the maximum squared Euclidean distance between any pair of cluster centers. In this manner, is is ensured that the initial value of the variance parameter(s) is not too small. This strategy was found empirically to yield a robust learning process in practice. Note that, when attribute weights a_j^2 are used, these are initially set to 1.0.

To speed up the learning process on multi-core computers, the calculation of the error function and its gradient in the implementations is parallelised by employing a user-specified number of threads. The user can also set the number of threads available in the thread pool. The parallel implementation is achieved by splitting the data into equal-size chunks and calculating the error function and gradients separately for each chunk. These are subsequently aggregated.

To further improve speed of the classification method, an approximate version of the logistic function is used instead of the exact expression. Moreover, in both the classification and the regression method, if any delta value occuring in backpropagation—i.e., when calculating the gradient—is smaller in absolute terms than a user-specified threshold (the default is 10^{-6}), then the delta value is treated as zero and corresponding further calculations are skipped.

3 Conclusions

Given the excellent performance of kernel-based classifiers such as support vector machines, which are often applied with an RBF kernel that corresponds to a Gaussian basis function with a fixed global variance, it is likely that per-unit variance parameters—and even more so—per-attribute per-unit variance parameters as provided by the RBF network implementations discussed here, become less important as larger numbers of hidden units are used. Nevertheless, work on attribute selection for support vector machines indicates that attribute weights may be useful for downweighting less relevant attributes even in the case when many basis functions are employed. Moreover, in some applications, it may be important to have very compact predictors with few basis functions, and these can be learned using the implementations of RBF networks presented in this document.

References

[1] Y.H. Dai and Y. Yuan. An efficient hybrid conjugate gradient method for unconstrained optimization. *Annals of Operations Research*, 103:33–47, 2001.

[2] Dietrich Wettschereck and Thomas Dietterich. Improving the performance of radial basis function networks by learning center locations. In *NIPS*, volume 4, pages 1133–1140, 1991.