ESANN'1998 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 22-23-24 April 1998, D-Facto public., ISBN 2-9600049-8-1, pp. 21-26

An Incremental local Radial Basis Function Network

Anna Esposito†
Maria Marinaro*†, Silvia Scarpetta*, INFM, Unità di Salerno,

Salerno, Italy
† International Institute for Advanced Science Studies
"E.R.Caianiello"
Via Garibaldi, Vietri sul Mare, Salerno, Italy * Università di
Salerno, Dipartimento di Fisica Teorica "E.R.Caianiello"
Via S. Allende, Baronissi, Salerno, Italy

Abstract. In this paper we present a learning algorithm to approximate functions which are piece-wise continuous.
The network is a RBF net in which the normalized functions are restricted to finite domains, and their variances are updated by supervised training. The net architecture grows during the training phase, increasing the number of hidden nodes in order to use a finer resolution where it is needed. The ability of this kind of net to save training time depends on selectively growing the net structure and on the locality of the basis activation functions. Some examples of the net performance are shown.

1. Introduction

The network is designed to rapidly approximate piece-wise continuous real valued functions $f : \boldsymbol{x} \rightarrow y$ from $k-$dimensional input space, $\boldsymbol{x} \in S_{input}$, to one dimensional output space, $y \in S_{out}$. The net is supposed to learn the function from a finite set of examples $\Gamma_p = \{\boldsymbol{x}_i, y_i; i = 1, \ldots, p\}$ where $y_i = f(\boldsymbol{x}_i)$.
The network is a Radial Basis Function Net [2] in which the normalized [1] basis functions are restricted to finite domains. Each hidden node is a prototype of the function in his domain and it is forced to be zero-value outside his domain. The net architecture grows during the training phase, increasing the number of hidden nodes in order to use a finer resolution where it is needed to describe the function f embedded in the data of the training set. The input space is quantized into partially overlapping local domains. An hidden gaussian prototype node is assigned to each domain. The number of hidden prototype nodes, equal to the number of domains, starting by one, is increased, one by one, until a satisfactory approximation of the function f is reached. The centers of the radial basis functions are fixed according to a predetermined rule which make use of both inputs and targets of the training set, while the variances are determined by a supervised optimization technique.

ESANN'1998 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 22-23-24 April 1998, D-Facto public., ISBN 2-9600049-8-1, pp. 21-26

2. The network model

The net has k input nodes, one output node, and two hidden layers. The first hidden layer contains an increasing number of gaussian prototype nodes.
The activation function $\Psi(\boldsymbol{x}, \bar{\boldsymbol{x}}_i, \beta_i)$ of the hidden node h_i is a gaussian function, centered in $\bar{\boldsymbol{x}}_i$, with variance β_i, restricted to the local domain of radius r_i:

$$\Psi(\boldsymbol{x}, \bar{\boldsymbol{x}}_i, \beta_i) = \sigma_{r_i}(\boldsymbol{x}) \exp(-\beta_i(\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^2),$$

$$\sigma_{r_i}(\boldsymbol{x}) = \begin{cases} 1 & \text{if}[(\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^2 - r_i^2] < 0 \\ 0 & \text{otherwise} \end{cases}$$

and β_i are parameters that have to be learned in the training phase.
It's worth to note that use of σ_{r_i} allows us to have well localized activation functions independently from the value of variances β. The second hidden layer contains two nodes, N and D, which are connected to the i^{th} node of the first hidden layer with weights $y_i = f(\boldsymbol{x}_i)$ and 1 respectively.
The two nodes compute the OR functions of the first layer hidden nodes.
The net output is the ratio of N and D nodes outputs. Thus at the N^{th} step we have:

$$\Phi_N(\boldsymbol{x}) = \frac{\sum_i^N y_i \exp(-\beta_i(\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^2)\sigma_{r_i}(\boldsymbol{x})}{\sum_i^N \exp(-\beta_i(\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^2)\sigma_{r_i}(\boldsymbol{x})}$$

Namely the output of the net is a weighted average of the prototype values y_i,

$$\Phi_N(\boldsymbol{x}) = \sum_i w_i(\boldsymbol{x}) y_i$$

. The weight in this average are the probability functions:

$$w_i(\boldsymbol{x}) = \frac{\exp\left(-\beta_i(\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^2\right)\sigma_{r_i}(\boldsymbol{x})}{\sum_i \exp\left(-\beta_i(\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^2\right)\sigma_{r_i}(\boldsymbol{x})}$$

3. The training phase

The training phase is an iterative process which adds a node at each step. Each step is formed of three stages:

a) A new node is chosen and parameters $\bar{\boldsymbol{x}}_i$, y_i are fixed.

b) The radius of the new node domain is settled and the radii of the near domains have to be resettled. So a new domain appear and the near domains become smaller.

c) The adaptive parameters β are changed according to a learning rule.

We use as examples to train the network a set Γ_p of p points $\{t_i, f_i \equiv f(t_i)\}_{i=1,;p}$ where t_i are chosen random in S_{input}.
More in detail, we perform the following operations at the i^{th} step of the

ESANN'1998 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 22-23-24 April 1998, D-Facto public., ISBN 2-9600049-8-1, pp. 21-26

training phase :
• We choose from the training set Γ_p the point $(\boldsymbol{x}_s, f_s)$ such that

$$(f_s - \Phi_{i-1})^2 = \max_{r=1,\ldots,p} (f_r - \Phi_{i-1})^2$$

i.e. the point where the squared error between target and net output is largest. And we add a new node i whose center is fixed equal to $\boldsymbol{x}_s$.
• The weighted connection y_i from the new node to the node N of the second hidden layer is settled to the value $y_i = f_s$, that is the target of the input t_s.
• The radius (or threshold) r_i of the new domain is settled (see next section).
• Once the parameters $\bar{\boldsymbol{x}}_i, y_i$ and r_i of the new node are set, the variances β of all the nodes are trained using a gradient descendent method with momentum, minimizing the sum $E = \sum_{r=1}^{p} (f_r - \Phi_i(t_r))^2$, over all the points of Γ_p, of squared error between network output and the target output:

$$\beta_j(t) = \beta_j(t-1) + \Delta\beta_j(t)$$

$$\Delta\beta_j(t) = -\eta \frac{\partial E}{\partial \beta_j} + \mu \Delta\beta_j(t-1)$$

where η is the learning rate and μ is the momentum gain.
All the points of the training set are presented to the net several times, until β parameters are learned. The learning of parameters β of the activation functions by gradient descendent algorithm represents a non-linear optimization problem which will typically be computationally intensive and may be plagued with a multitude of of local minima and plateau which can usually retard the learning process. But, this is not the case. In fact, by appropriate choose of the thresholds, any given input will only generate a activation in a small fraction of basis functions. Moreover, at each step, the new prototype node we add has only a local influence and the β parameters of the prototype nodes far from the new one, learned in the preceding steps, are still good parameters.
During the β-training the largest squared error between network response $\Phi(t_r)$ and target f_r is stored with the corresponding input and target value (t_s, f_s). This point (t_s, f_s) is the prototype value of the new node that will be added in the next step. If the net output is an enough good approximation of the function which have to be approximated, the training is stopped, otherwise the next $(i+1)^{th}$ step begins and a new node is added.

4. Setting the thresholds

The radius r_i of each new hidden gaussian node i is settled in a different way according to the dimensionality of the input.
In the case of one-dimensional input space the rule is the following. We choose two different radii for each new node i, a left threshold r_i^L and a right threshold r_i^R. We set r_i^L (r_i^R) to the distance between the centre $\bar{\boldsymbol{x}}_i$ and the left (right) next-neighbour domain centre. Also the radii of the two next-neighbour domains have to be resettled with the same rule. In such a way all the domains

ESANN'1998 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 22-23-24 April 1998, D-Facto public., ISBN 2-9600049-8-1, pp. 21-26

are partially overlapping and each point of the input space belongs to two overlapping domains (except only points near the boundary part of the input space).
If the input space is bidimensional we set the radius of the new node in such a way that all domains are partially overlapping with some of the neighbour domains. Let $r1$ the minimum distance between the new node centre $\bar{\boldsymbol{x}}_i = (\bar{x}_{i,1}, \bar{x}_{i,2})$ and the others centers $\bar{\boldsymbol{x}}_j$ such that

$$c = \sum_{k=1}^{2} (\bar{x}_{i,k} - \bar{x}_{j,k}) > 0$$

and let $r2$ the minimum distance between the new node centre and the others node centers such that $c < 0$. Let $rr1$ the distance of the second-neighbour node with $c > 0$ and let $rr2$ the distance of the second-neighbour node with $c < 0$. The radius of the new node is settled to the larger between $r1$ and $r2$ if the number of nodes is small (less then 10) and to the larger between $rr1$ and $rr2$ if there are more then 10 nodes or if the parameter c of all the nodes have the some sign. Also the radii of the other domain have to be resettled with the same rule. In such a way quite each point of the input belongs to, at least, two overlapping domains.

5. Simulations

In this section we present some results obtained considering one and two dimensional functions.
The first test is a one-dimensional function $f(x) = sin(x) + sin(5x) + sin(3x)$ with $x \in [-0.35, 3.5]$. The output was scaled to lie in the range [0,0.9). Both the training set and the test set consist of 100 input-output pairs generated random in the input-domain.
The growing procedure has been stoppen when the mean squared error is less than 0.001. The net structure has reached 7 hidden gaussian nodes. The mean squared error is 0.0004. Results are shown in fig. 1.B. Our performance is comparable with the one obtained by Vinod and Ghose [5] for growing nonuniform feedforward networks for continuous mappings.
We have test our algorithm also on a function with some discontinuity points,and we have obtained a good performance in this case too. An example is reported in fig. 1.A. The function is

$$f(x) = \begin{cases} sin(x) + sin(5x) + sin(3x) & \text{if } x < 0.4 \text{ or } x > 2.1 \\ 0 & \text{if } 1 < x < 1.5 \\ sin(x) + sin(5x) + sin(3x) - 2.5 & \text{otherwise} \end{cases}$$

The input domain is $x \in [-0.35, 3.5]$.
The training consists of 100 input-output pairs generated random in the input-domain.
The learning process has been stopped when the net structure has reached 12

ESANN'1998 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 22-23-24 April 1998, D-Facto public., ISBN 2-9600049-8-1, pp. 21-26

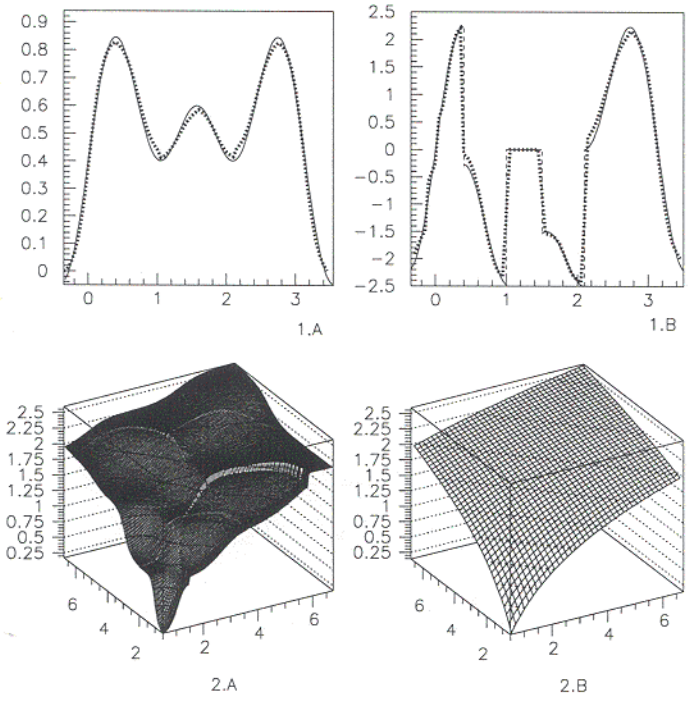

Figure 1: The dotted line is the net output and the thin line is the function to be approximated. (1.A) $f(x) = 0.5 * exp(-x) * sin(6x)$ (1.B) A discontinous function approximation. Figure 2: Approximation of the function $f(x, y) = log(x + y)$ (2.A) the net output (2.B) the target function $f(x, y) = log(x + y)$.

hidden gaussian nodes.
The third function is $f(x_1, x_2) = log(x_1 + x_2)$. The input domain is $x_1 \in [0.5, 3.75], x_2 \in [0.5, 3.75]$ The training set consists of 500 input-output pairs generated random in the input-domain. The results shown in fig. 2 are obtained stopping the learning process when the net structure has reached 11 hidden gaussian nodes.

6. Conclusion

The ability of this kind of network to save training time depends on selectively growing the net structure and on the locality of the basis activation functions. The thresholds of the activation functions turn out to be very useful when one

wants to approximate a function with some discontinuity points.
The error-driven growth is an heuristic attempt to enforce the net to use a finer resolution only where it is useful to describe the function f embedded in the data of the training set.
In the networks where a lattice quantization of the input space is used [4], it's difficult to choose an optimal lattice spacing, and usually a multi-resolution hierarchy becomes needed [3]. If a single uniform scale of resolution is used, there is a trade-off between the ability to generalize and the ability to capture fine detail.
Our approach does not use a lattice quantization of the input space but it performs a partitioning of the input space according to a rule which determine the prototypes's position and the domains's dimension. The rule make use all the information contained in the training set. This leads to placement of a set of prototype vector in input space that isn't necessary lattice-form but it reflects the distribution of the input-target data points.
Finally the possibility to have different values of variances β makes our approach more flexible of other methods previously introduced. Experiments about a kind of network with a similar growing strategy but without thresholds are in progress.

References

[1] H. Schioler and U.Hartmann. Mapping Neural Network Derived from the Parzen Window Estimator. Neural Networks Vol. 5 pp. 903, 1992

[2] C.M.Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.

[3] J. Moody, Fast learning in multi resolution hierarchies. *Advances in Neural Information Processing Systems 1.* Touretzky ed., pp. 29-39, Morgan Kaufmann, SanMateo, CA.

[4] Bulsari *Neural Networks*, Vol. 6, 1993

[5] V. Vinod and S. Ghose, Growing nonuniform feedforward networks for continuous mapping, *Neurocomputing* **10** pp.55, 1996