Peak Stick RBF Network for Online System Identification

Hossein Mobahi

Robotic Lab.
Dept. of Electrical and Computer Engineering
University of Tehran
Tehran, Iran

Farrokh Janabi-Sharifi
Robotics and Manufacturing Automation Lab
Dept. of Mechanical and Industrial Engineering
Ryerson University
Toronto, Ontario, Canada

Abstract

In many practical problems of online system identification, the distribution of observed samples is uneven. For instance, at points where system is idle or changes slowly, the sample density increases and where system moves quickly, it is reduced. This generally results in performance degradation of learning. We will propose a new algorithm for training RBF networks that is particularly developed for online learning with uneven sample distribution. The basic idea is to find peaks and stick to them. Experiments show a notable improvement in convergence rate, settling of weights and error minimization.

1. Introduction

One of the problems for the control of complex non-linear systems is often the difficulty in the identification of an accurate mathematical model of the system. To address this issue, in recent years, adaptive and online approximation methods are receiving considerable attention [6, 8, 9]. In this context, Artificial Neural Networks are excellent candidate to deal with approximation problems, thanks to their functional approximation capabilities and to the availability of effective learning algorithms.

However, real-time, on-line learning of non-linear mappings requires additional developments in neural learning algorithms. In particular, the approximation algorithm is very important since it can dramatically influence the performance of the controlled system. Therefore, aspects such as mapping accuracy and convergence rate of the learning algorithm should be clearly addressed.

In this paper, we will propose a new learning algorithm named Peak Stick (PS), which is particularly developed for online system identification using RBF networks. The algorithm can robustly deal with a serious problem in on-line learning, the uneven sample distribution problem. Unfortunately this problem is often overlooked and there are only a few works addressing it [1,2].

Among various types of neural networks, we selected RBF network for system identification due to it suitable properties in local specialization, global generalization and smooth map generation capacities [4]. Peak Stick algorithm attempts to localize function's peaks and lock an RBF center there. Therefore, future samples near to that unit will no longer attract it.

This paper is organized as follows. In section 2, we will introduce the uneven distribution problem. Then we will give an overview of RBF networks and their learning. Sections 4 and 5 explain the concepts behind "Peak Stick" algorithm and provide its formal algorithm respectively. Section 6 compares the proposed algorithm with the common RBF learning rule and discusses the improvements. Finally in section 7, we will have conclusion and future works.

2. Uneven Sample Distribution

In a passive* system identification task, generally the input and output variables of the reference system are measured and used unselectively and by the

* The learning system is not allowed to choose its examples

learning system as its training samples. In many practical problems, the density of the observed samples is uneven. For instance, when the system becomes idle or it changes very slowly, the density of samples increases and when it changes quickly, the density is reduced.

Uneven sample distribution usually degrades the learning quality of adaptive systems like neural networks. This is due to the fact that the adjustable parameters of a typical network continually change and therefore, high-density regions of the state space are over-learned by drifting parameters while low-density regions are learned poorly and forgotten quickly.

In case of offline learning, one can store all samples and then make any desired distribution out of it, e.g. uniform distribution. For instance, Bohn [1] has proposed a method for controlling signal distribution by resampling and has applied it to RBF networks. However, in our literature review, we could find a few works that have addressed this problem for the online case.

Ahrns et.al [2] proposed error modulated Kohonen rule to cope with this problem in on-line learning. Instead of reflecting the input probability density, they try to achieve a uniform distribution of local approximation errors. Local error is approximated from instantaneous errors, using an exponential recency-weighted average. To achieve a reasonable approximation of error, the forgetting factor of the average must be inversely proportional to the sampling density. However, sampling density is generally not known a priori and it may also change during system's operation.

In this article we will develop a new learning algorithm to cope with uneven distribution of samples in online learning tasks. Since our learning algorithm is based on RBF networks, we will first have a short review on this type of network.

3. RBF Networks

An RBF is a function whose output is symmetric around an associated center. For example, a Gaussian function can be viewed as an RBF by selecting the Euclidian norm. A Radial Basis Function Network (RBFN) is a 3-layer feed-forward network (Figure 1) in which each hidden unit computes the RBF activation and the output units compute a weighted sum of the hidden-unit activations.

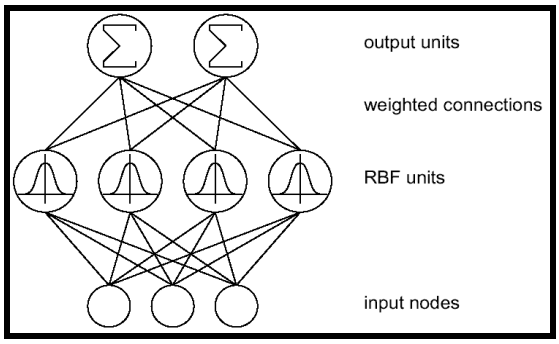

Figure 1. RBF Network

The mapping of the input vector to each output unit using Gaussian RBFs is mathematically shown in equation (1). Here, a nonlinear function *f(***x***)* is approximated by a weighted sum of Gaussians where $\boldsymbol{\mu}_i$, σ_i and w_i denote the center coordinate, width and weight of i-th RBF unit respectively and **x** is the input vector. A detailed overview of RBF networks is available in [5].

RBFNs have been used in different applications in order to model unknown functions [3, 4, 6, 8]. They have suitable properties to be used for function approximation [4] because of their local specialization and global generalization capabilities. In addition, they produce smooth maps, appropriate for control applications, compared with other local methods such as Locally Linear Maps (LLM) [4]. Therefore, we adopt RBFN in our system identification problem.

$$f(\mathbf{x}): R^n \rightarrow R$$
$$y \approx f(\mathbf{x}) = \sum_i w_i \exp\left(\frac{-\left\|\mathbf{x}-\boldsymbol{\mu}_i\right\|^2}{2\sigma_i^2}\right) \quad (1)$$

Since one would like at least one RBF unit to be active in regions of the input space where data is presented, RBF centers should be placed in regions of data concentration. In online simple training, the winning center is updated using standard competitive learning rule. For simple RBFNs, widths of RBFs are generally considered constant. However, output weights are updated using Least Mean Square (LMS) method. Equation (2) shows updates for online training of a simple RBFN, where s is the winning unit and η with ζ are learning rates.

$$s = \arg\min_i \left\|\mathbf{x}-\boldsymbol{\mu}_i\right\|$$
$$\Delta\boldsymbol{\mu}_s = \eta(\mathbf{x}-\boldsymbol{\mu}_s) \quad (2)$$
$$\Delta w_s = \zeta(f(\mathbf{x})-y)\exp\left(\frac{-\left\|\mathbf{x}-\boldsymbol{\mu}_s\right\|^2}{2\sigma^2}\right)$$

It is obvious that these update rules are influenced by sample distribution, i.e., in the long run low-density samples have less impact on parameter adjustment. Therefore the learning performance degrades when sample distribution is uneven. Even more advanced RBFNs that are particularly developed for online learning tasks such as Resource Allocating Networks (RAN) [7] and Dynamic Cell Structures (DCS) [2] suffer from this problem, because their adaptable parameters change continually as well.

4. Peak Stick Learning Rule

A Peak Stick network is an RBFN whose learning algorithm is particularly developed for on-line training of samples with uneven distribution. Peak Stick algorithm attempts to localize function's peaks. Since RBF centers are sensible choices for representing smooth peaks, they are placed at the discovered hills or valleys. Once the largest peak about an RBF unit is found, the unit is frozen there. Therefore future samples near to that unit will no longer attract it.

In order for this algorithm to work, a few assumptions must be made:

1. The mapping to be learned must be smooth enough such that it can be approximated by a reasonable number of RBF functions.
2. We assume that placing an RBF center at a peak location is a fine choice. This is true again if the peaks are RBF-like.
3. Once a peak is found in a region, the RBF in that region is locked there and it will never shorten its height (weight) again. Therefore, if the system to be learned in not time invariant, Peak Stick learning will fail to learn.

The incremental process of peak finding is achieved by restricting parameter adaptation to when the absolute value of the observed sample is larger than RBF's weight. This coagulation prevents drift of the parameters when the actual location of the peak is found. Peak Stick acts in a competitive manner. Therefore each RBF unit is responsible to find the largest peak only within its Vornoi region.

At the beginning, RBF centers are initialized at random spots and their weights are set to very small values. In the main loop, when a sample is acquired, its nearest RBF unit with the same weight sign is selected for adaptation. Adaptation occurs only if the magnitude of the observed output is larger than network's output. Hence, the height/depth of each RBF gradually rises in the correct direction until the largest peak of its local region is found.

Determining misplaced units can make an efficient use of network resources. A misplaced unit barely contributes to reconstruction of the original function. Therefore, a unit that has successively won competition, but its amplitude remaining very small, is possibly misplaced. Misplaced units are moved to other places in the input space.

Another resource management can be achieved by eliminating redundant units. It may occasionally happen that two near RBFs can be combined and replaced by a single RBF, or they cancel out each other. These cases may release one and two units respectively. Therefore, units are checked so that redundant units are eliminated and then implanted again as new seeds (units with very small weights, but with their original signs) in other places.

Recent changes in parameters of a unit mean that such unit has not yet found its accurate place. Combining such a unit with one that has already settled down at its right place corrupts the settled RBF too. Therefore, two near units should be combined only if both have become stable. We call the fluctuation of an RBF unit its temperature and measure it from the recent changes in its parameters.

To keep the algorithm lightweight, no statistics from the past observations is preserved. Therefore, while a redundant or misplaced unit may be found, we cannot suggest an adequate place for it. Perhaps the best zero-th order guess is the region about the current observation, hoping the unit can grow up there soon. Although this substitution may lead to a misplaced unit itself, at least it is located in a valid state where the system may encounter again. Note that we do not know the regions of input space that system may travel prior to observing them.

5. Peak Stick (PS) Algorithm

The formal description of Peak Stick algorithm, suitable for computer-based simulation, is presented as follows.

I. Initialization:

1. Initialize the centers of units at random places and set their weights to $+\varepsilon$ or $-\varepsilon$, where ε is a very small positive number.
2. Initialize the temperatures t of units to T_0.

II. Shifting Misplaced Units:

3. Acquire input vector **x** and output value *y*.
4. Based on a predefined probability of shifting, either perform a shift (as described below) or go to step 9.
5. The set of misplaced units contains units whose weights are below a threshold α.

$$\mathrm{M} = \bigcup_{w_i<\alpha} \{(\boldsymbol{\mu}_i, w_i)\} \tag{3}$$

6. If M is null go to step 9 otherwise randomly choose one of its members, m_j:

$$\mathrm{M} \neq \varnothing \rightarrow m_j = (\boldsymbol{\mu}_j, w_j) \in \mathrm{M} \tag{4}$$

7. Shift the center of unit *j* to **x** and set its weight to ε. The sign of the ε is chosen randomly.

$$\boldsymbol{\mu}_j = \mathbf{x} \tag{5}$$

$$w_j = \pm\varepsilon$$

8. Reset its temperature

$$t_j = T_0 \tag{6}$$

III. Winner Determination

9. The winning unit *s* is defined as the unit with the minimum distance from its centroid to **x**, and having similar output signs.

$$s = \underset{i;w_i y>0}{\operatorname{argmin}} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\| \tag{7}$$

10. If *y* and w_s have different signs, no winner is defined, therefore go to step 3.

IV. Adaptation

11. Adapt (winner's) parameters only if moving toward a larger peak. Here η and ζ are learning rates for center and weight adaptation and γ is a forgetting factor.

$$|y| > |f(\mathbf{x})| \wedge |y| > |w_s| \rightarrow$$

$$\begin{cases} \Delta\boldsymbol{\mu}_s = \eta(\mathbf{x} - \boldsymbol{\mu}_s) \\ \Delta w_s = \zeta(y - f(\mathbf{x}))\exp\left(\dfrac{-\left\|\mathbf{x} - \boldsymbol{\mu}_s\right\|^2}{2\sigma^2}\right) \\ t_s^{new} = (1-\gamma)t_s^{old} + \gamma(\left\|\Delta\boldsymbol{\mu}_s\right\|^2 + \Delta w_s{}^2) \end{cases} \tag{8}$$

V. Filtering

12. Find the nearest unit to *s* and call it *r*.

$$r = \arg\min_i \left\| \boldsymbol{\mu}_s - \boldsymbol{\mu}_i \right\| \tag{9}$$

13. If the distance between centers of *r* and *s* is below a threshold β and their temperatures are colder than *T*, then combine them to one RBF, and shift the other.

$$(\left\|\boldsymbol{\mu}_r - \boldsymbol{\mu}_s\right\| < \beta) \wedge (t_s < T) \wedge (t_r < T) \rightarrow$$

$$\begin{cases} \boldsymbol{\mu}_r = \dfrac{w_r\boldsymbol{\mu}_r + w_s\boldsymbol{\mu}_s}{w_r + w_s} \\ w_r = w_r + w_s \\ \boldsymbol{\mu}_s = \mathbf{x} \\ w_s = \pm\varepsilon \end{cases} \tag{10}$$

14. Go to step 2

6. Experimental Results

To evaluate the performance of PS learning rule, two experiments were carried out; one with Matlab's *peaks* function and another with a gray-scale image. In the first experiment, Mean Squared Error (MSE) was computed and plotted. Although RBF parameters are updated locally and online, for evaluation purpose, error is computed on the whole space at any moment of time. Below, *A* is the evaluation region in the input space. Network parameters and global error are denoted by *Θ(k)* and *E(k)* respectively, in *k-th* iteration.

$$E(k) = \frac{\int_A (f_{\Theta(k)}(A) - y_A)^2 \, dA}{\int_A dA} \tag{11}$$

Matlab's *Peaks* is a function of two variables, obtained by translating and scaling Gaussians in equation (12) as shown in Figure 2. Uneven distribution was modeled by trigonometric functions as shown in equation (13). The shape of this distribution in 1000 iterations is shown in Figure 3. At points where the curve has less change along time axis (vertical), distribution becomes denser.

$$\begin{aligned} z = {} & 3(1-x_1)^2 \exp(-x_1{}^2 - (x_2+1)^2) \\ & -10\left(\frac{x_1}{5} - x_1{}^3 - x_2{}^5\right)\exp(-x_1{}^2 - x_2{}^2) \\ & -\frac{1}{3}\exp(-(x_1+1)^2 - x_2{}^2) \end{aligned} \tag{12}$$

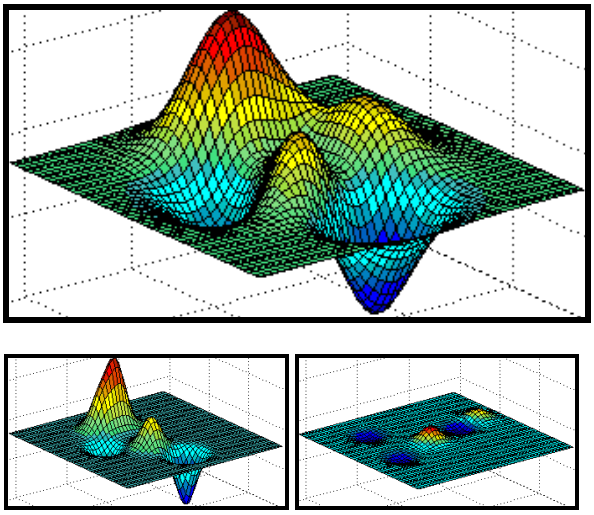

Figure 2. Top: Matlab's Peaks Function, Bottom: PSRBF vs. Simple RBF reconstructions

$$\begin{cases} x_1 = r(t)\cos\theta(t) \\ x_2 = r(t)\sin\theta(t) \\ r(t) = \sin^2(\dfrac{(4\pi+1)t}{1000}) \\ \theta(t) = \dfrac{2\pi t}{1000} \end{cases} \qquad (13)$$

Table 1 summarizes parameters used in PS and Simple RBF learning along with their corresponding errors. Other parameters that remained constant in our experiment were as follows:

- **Common**: η=0.1, ζ=0.1
- **PS**: α=0.1, β=0.1, γ=0.1, P_{shift}=0.01, T_0=5.0, T=1.0

The algorithm was executed five times for each set of parameters in rows of the table and the average of five *MSE(k)*'s were computed. Each execution was realized by 10,000 iterations and error was measured within A=[-3,3]x[-3,3] region of input space. Since the plotted errors for comparing Peak Stick and Simple RBF did not behave much differently, only the first plot is shown in Figure 4. The mean of each error over 10,000 iterations is also available in the table.

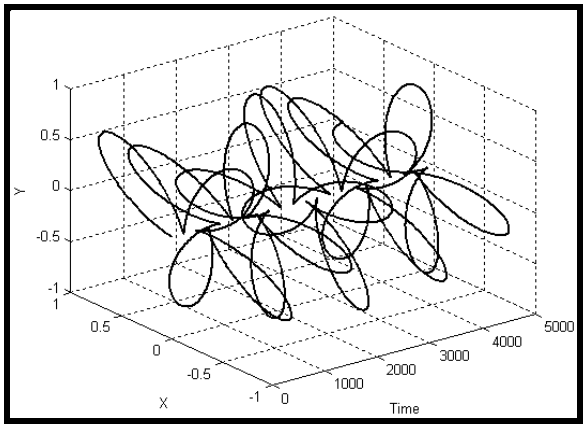

Figure 3. An uneven sample distribution generated by trigonometric functions

Figure 4 indicates that the error of a Simple RBF (which is itself a function of its weights) constantly oscillates with very large ripples, while that of PS-RBF converges quickly. Moreover, it is clear from Table 1 that the average error of PS-RBF is much less than that of Simple RBF. Reconstruction results can be seen in the bottom plots of Figure 2.

In the second experiment a 256x256 gray-scale image was approximated using 100 RBF units with width of 0.003. Gray values were normalized to range from -1 (black) to 1 (white). The same parameters and trigonometric distribution were applied. Again while PS-RBF could converge quickly, Simple RBF oscillated aimlessly and finally had no achievement. Figure 5 shows reconstructed images of networks trained by PS-RBF and Simple RBF algorithms after 5000 iterations. Evolution of networks weights is available as movie files on Internet*.

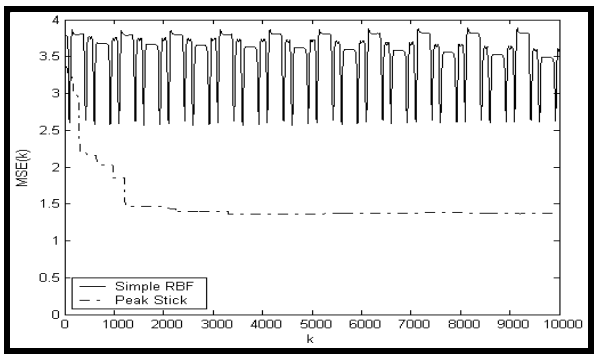

Figure 4 Error Plots of PSRBFN vs Simple RBF

* http://www.digibrain.org/psrbf

Table 1. PS-RBFN vs. Simple RBF

#	Common Parameters		Average Error	
	Units	Initial Width	PS	Simple RBF
1	5	0.1	1.5665	3.5824
2	10	0.1	1.3248	3.5592
3	5	0.5	1.3586	3.6525
4	10	0.5	1.4627	4.3836
5	5	1.0	2.6271	4.5936
6	10	1.0	2.7822	7.2759

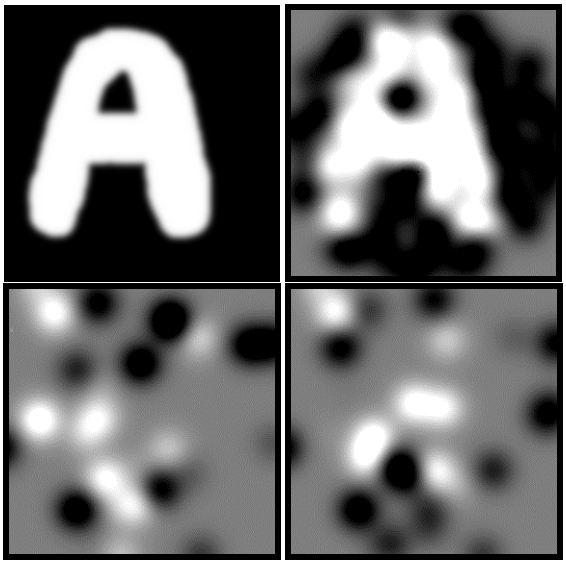

Figure 5. Top-Left: The original image, Top-Right: Reconstructed image using PS-RBFN, Bottom Left and Right: Reconstructed image using Simple RBF with initial and final weights respectively.

7. Conclusion and Future Works

We postulated the problem of uneven sample distribution as a difficulty for a large class of online learning algorithms. The phenomenon results in fluctuation of network's parameters and enormously enlarges the settling time of the network. We then proposed a new learning algorithm (PS) for RBF networks that is particularly developed for coping with this problem. Our experiments confirm that the proposed algorithm converges very quickly to a lower level of error.

Although stability is shown empirically, no mathematical proof is presented. Therefore, stability analysis of PS algorithm is an important path for future research. In addition, for combining two Gaussians and replacing a single one instead simple heuristic rules were adopted. Therefore, another issue for future study is finding the optimal combination and replacement for Gaussians. At last, the algorithm is restricted to learn time invariant systems. Generalizing PS to be applicable to time variant systems remains to be addressed.

8. References

[1] Bohn, C., "An incremental unsupervised learning scheme for function approximation", *Proc. 1997 IEEE Int. Conf. Neural Networks*, pp. 1792-1797, Piscataway, NJ, June 1997.

[2] Ahrns, I., Bruske, J., Sommer, G., "On-line learning with dynamic cell structures", *Proc. 1995 Int. Conf. Artificial Neural Networks: ICANN'95*, Paris, France, pp. 141-146, 1995.

[3] Fritzke, B., "Fast learning with incremental RBF networks", *Neural Processing Letters*, vol. 1, no. 1, pp. 2-5, 1994.

[4] Fritzke, B., "Incremental learning of local linear mappings", In F. Fogelman and P. Gallinari, editors, *Proc. 1995 Int. Conf. Artificial Neural Networks: ICANN'95*, pp. 217-222, Paris, France, 1995.

[5] Ghosh, J., Nag, A., "An Overview of Radial Basis Function Networks", *Radial Basis Function Neural Network Theory and Applications*, R. J. Howlerr and L. C. Jain (Eds), Physica-Verlag., 2000.

[6] Kim B.S., Calise A.J, "Nonlinear flight control using Neural Networks", *AIAA Journal of Guidance, Control, and Dynamics,* vol. 20, no. 1, pp. 26--33, 1997.

[7] Platt J.C., "A resource allocating network for function interpolation", *Neural Computation* 1991, vol. 3, no. 2, pp. 213-225.

[8] Polycarpou M., "Online approximators for nonlinear system identification: A unified approach", *Control and Dynamics Systems Serries,* vol. 7, Neural Networks Systems and Applications (Academic Press, January 1998).

[9] Sanner R.M., Slotine J., "Gaussian Networks for direct adaptive control", *IEEE Trans. Automatic Control,* vol.3, no.6, pp. 837-863, 1992.