Available online at www.sciencedirect.com

SCIENCE DIRECT®

NEUROCOMPUTING

Neurocomputing 69 (2006) 1183–1186

www.elsevier.com/locate/neucom

A self-organizing map with homeostatic synaptic scaling

Thomas J. Sullivan[a,*], Virginia R. de Sa[b]

[a]*Department of Electrical and Computer Engineering, University of California, San Diego, USA*
[b]*Department of Cognitive Science, University of California, San Diego, USA*

Available online 7 February 2006

Abstract

Hebbian learning has been a staple of neural-network models for many years. It is well known that the most straight-forward implementations of this popular learning rule lead to unconstrained weight growth. A newly discovered property of cortical neurons is that they try to maintain a preset average firing rate [G.G. Turrigiano, S.B. Nelson, Homeostatic plasticity in the developing nervous system, Nat. Rev. Neurosci. 5 (2004) 97–107]. We use this property to control the Hebbian learning process in a self-organizing map network. In this article, the practicality of this type of learning rule is expanded by deriving a scaling equation for the learning rates for various network architectures.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Self-organizing map; Homeostasis; Weight normalization; Synaptic scaling

1. Introduction

Self-organizing maps (SOM) [2] have been a useful tool in the field of neural networks and have been proposed in various forms as models of cortical development [3,2]. In these models Hebbian learning is used to strengthen associations between input stimuli and active output neurons, but unchecked, this can result in unconstrained weight growth. To counteract this problem, typically weight normalization is employed: after each learning iteration all the weights are divided by the sum of the magnitude of the weights coming into each neuron. While this sounds within the realm of biological possibility, and is obviously helpful in keeping Hebbian learning in check, little evidence from the experimental literature is available to support it. A more plausible mechanism for controlling the Hebbian process has recently emerged. Turrigiano and others [5] have shown that neurons in the cortex actively maintain an average firing rate by scaling their incoming weights. This homeostatic synaptic scaling mechanism is an interesting candidate to constrain weight growth.

In our SOM, the commonly used weight normalization is exchanged for a more biologically realistic homeostatic weight scaling. The new mechanism is capable of keeping Hebbian learning in check and leads to the usual self-organization. This type of mechanism can maintain firing rates in the face of large-scale neuron proliferation and die-off [4]. The main focus of this article is the issue of setting the learning parameters for networks of various size with various input data sets. This practical matter is crucial for generating models without being bogged down in parameter tweaking.

2. SOM with homeostatic synaptic scaling

The SOM model is trained with a series of episodes in which randomly selected input vectors are presented. At each step, the input vector, $\vec{x}$, is first multiplied by the feedforward weights, W_{FF}. In order to get the SOM effect, this feedforward activity is then multiplied by a set of lateral connections, W_{lat}. After the output activity is set, the feedforward weights are updated with a Hebbian learning rule. In the following, w_{ij} are individual elements of the feedforward weight matrix W_{FF} (the subscript FF is dropped for convenience), and $\tilde{w}_{ij}$ is a weight after Hebbian

*Corresponding author.
E-mail address: tom@sullivan.to (T.J. Sullivan).

0925-2312/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2005.12.071

learning, but before weight normalization.

$$\vec{y} = f[W_{\text{lat}}(W_{\text{FF}}\vec{x})], \quad \tilde{w}_{ij}^t = w_{ij}^t + \alpha x_j^t y_i^t. \tag{1}$$

Here $f[a] = \max(0, a)$ and W_{lat} is preset to a Mexican hat shape. α is the Hebbian learning rate, x_j is the presynaptic activity and y_i is the postsynaptic activity. Since each update is positive, there is nothing to limit the growth of the weights. Normally, a weight normalization is used that is based on the sum of the magnitude of the weights coming into each neuron. In our case, we normalize the weights with a value based on the recent activity of the neuron

$$w_{ij}^{t+1} = \frac{\tilde{w}_{ij}^t}{\text{ActivityNorm}_i^t},$$

$$\text{ActivityNorm}_i^t = 1 + \beta_{\text{N}}\left(\frac{A_{\text{avg},i}^t - A_{\text{target}}}{A_{\text{target}}}\right). \tag{2}$$

Here, A_{target} is the internally defined activity level for the neurons, $A_{\text{avg},i}$ is a running average of recent activity for each neuron, i, and β_{N} is the homeostatic adaptation rate.

The above rules lead to self-organized maps. A small model was created to illustrate typical operation. This network consists of 100 inputs and 10 output neurons arranged in a one-dimensional (1-D) string (a ring arrangement was used to avoid edge effects). The input vector activities are specified by a 1-D gaussian shape (standard deviation, σ, of $N/30 = 3.3$ units). The input gaussian is centered on one of the input units, selected at random. Plots of typical network behavior are shown on the left side of Fig. 1. The top-left plot shows that the average activity of the output neurons converge to the target activity level. The bottom-left plot shows an input–output map that has formed after training has ended.

Two measures of map quality were created to measure performance of a trained network. First, the number of discontinuities in a given input–output map (as shown in Fig. 1) were counted and subtracted from the number of output units. This measure is called the discontinuity test and a smooth map that utilizes all the outputs will have a

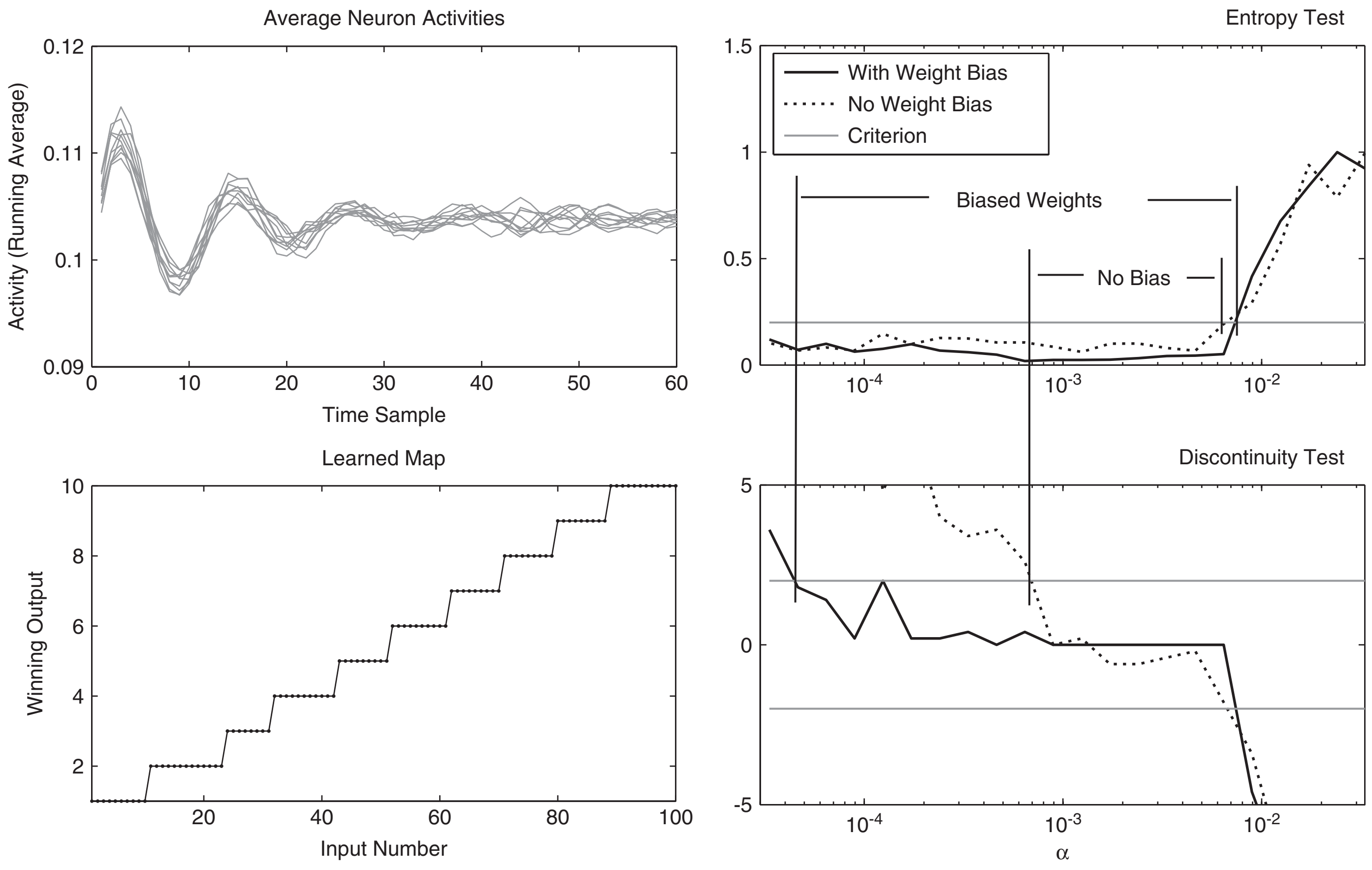

Fig. 1. Typical behavior. 100 inputs, 10 outputs, $\alpha = 8.3 \times 10^{-4}$, $\beta_{\text{N}} = 3.3 \times 10^{-4}$, and $A_{\text{target}} = 0.1$. (Left top) The average neuron activities are driven to the target value over time. (Left bottom) Every possible input was presented to the network, one at a time. The winning output neuron (the one with the highest output rate) was then recorded for each input. The input number is shown on the x-axis, and the corresponding winning output is plotted. This is a good map since similar inputs (inputs whose center is located on neighboring input units) correspond to nearby winning output units. Both the inputs and outputs are arranged in a ring configuration to eliminate edge effects. (Right) Determination of valid α parameter ranges. Two different performance measures, the entropy test and the discontinuity test, are used (see text). The range of valid parameters is determined by setting criteria on these measures. The network with initial weight biases has a wider valid range.

discontinuity test score of 0. The second performance measure, the entropy test, used a large number of randomly selected input data vectors. For each input example, the output unit with the highest activation was called the winner. For the input data set, the winning probability of each output unit was computed. The entropy over these output probabilities was then computed and subtracted from the highest possible entropy (all outputs winning an equal number of times). The best score for this test is 0. By these measures, good performance is shown over a wide range of Hebbian learning rate (α) on the right of Fig. 1. We can set cut-off criteria to define the useful range of α. In this case we use 0.2 times the maximum entropy for the entropy test and 0.2 times the number of output units for the discontinuity test. In the normal case, the initial weights of the model are selected uniformly at random from a range between 0 and 1. In a second case, the weights were biased according to location, with nearby neurons getting bigger weights. This is modelled after structured initial connectivity set up in the cortex during development through chemical gradients. From the figure we see that the network with the biased initial weights converges for a wider range of α values.

3. Learning rate parameter scaling

While creating networks of different sizes, we noticed that different learning rate parameters were required for good performance. This was also true when different magnitudes of input vectors were used. In order to make this homeostatic technique practical, these types of architecture-specific dependencies should be removed. We do not propose that in the natural world, each network must compute its proper learning rate. Rather, for networks of a given size, finding a good learning rate could be a problem left for evolution over a long period of time. We address it now to increase the practicality of our algorithm.

Learning rate parameter scaling has been taken up before by Bednar and others [1]. The relevant parameters in our model are somewhat different (for example A_{target} and the magnitude of the input vectors), so new equations had to be derived. Additionally, we introduce an intuitive method for validating the effectiveness of the derived scaling equation. This is particularly important since approximations are used in the derivation.

For various networks, we would like the rate of change in the system due to the Hebbian update to be constant, regardless of the number of inputs (N) and outputs (M) or the magnitude of the input vectors ($\|\vec{x}\|_1$, which denotes the L1 norm). Through a series of rough approximations, we can see how the effective learning rate depends on these factors. Noting that the input values x_j, output values y_i, and weight values w_{ij} are all positive, we can start by approximating the change in the system for each iteration as the average relative change in weights for all the output units.

$$\frac{\text{Change in System}}{\text{Iteration}} = \frac{\sum_{i=1}^{M} \mathrm{Avg}_j\left(\frac{\Delta w_{ij}}{w_{ij}}\right)}{\text{Iteration}} \approx \frac{M \frac{\mathrm{Avg}_{ij}(\Delta w_{ij})}{\mathrm{Avg}_{ij}(w_{ij})}}{\text{Iteration}}. \tag{3}$$

From the learning rules we can (roughly) approximate the average weight change and average weight value per iteration by using $\mathrm{Avg}(x_j)$ and $\mathrm{Avg}(y_i)$ in place of x_j and y_i, respectively:

$$\mathop{\mathrm{Avg}}_{ij}(\Delta w_{ij}) \approx \alpha\, \mathrm{Avg}(x_j)\, \mathrm{Avg}(y_i),$$

and

$$\mathop{\mathrm{Avg}}_{ij}(w_{ij}) \approx \frac{\mathrm{Avg}(y_i)}{N \mathrm{Avg}(x_j)}, \tag{4}$$

$$\frac{\text{Change in System}}{\text{Iteration}} \approx \alpha MN\, \mathrm{Avg}(x_j)^2 \approx \alpha MN \left(\frac{\|\vec{x}\|_1}{N}\right)^2 = \frac{\alpha M \|\vec{x}\|_1^2}{N}. \tag{5}$$

The above equation tells us that in order to achieve a constant rate of system change per iteration, we would need to scale α by M, N, and $\|\vec{x}\|_1^2$. Also, we should notice that various problems with different complexities might present a different number of data examples to learn. We can call the number of data examples for a given problem an epoch, and strive to keep constant the "Change in System" per epoch, rather than per iteration. Assuming there are K data examples per epoch for a given problem (and thus K iterations per epoch) we can incorporate this term into our equation. We now set the "Change in System" per epoch to a constant value and solve for α. For convenience, we will call the constant $1/\alpha_k$

$$\alpha = \frac{N}{\alpha_k K M \|\vec{x}\|_1^2}. \tag{6}$$

The learning rate is now split into a factor that can be chosen by the user (α_k), and one that scales with the architecture of the network. Practically speaking, we will choose an α_k, calculate α, and use α as the learning rate of our network. Doing this should mean that the networks learn and converge at similar rates, regardless of architecture.

To demonstrate the usefulness of this learning rate parameter scaling, several simulations were performed for networks of various architectures (with K constant). The same network as above was used as a baseline: it has 100 inputs, 10 outputs, $A_{\text{target}} = 0.1$, and $\|\vec{x}\|_1 = 1$. The range of valid learning rate parameter, α was measured as illustrated previously in Fig. 1. This range is shown in the left panel of Fig. 2 on the second line from the bottom. The previously discussed case of biased weights is shown on the bottom line. The baseline architecture was changed in various ways to generate other networks (one change per network), and the resulting ranges of valid learning rate parameter are shown on the left panel. The changes were (1) an increase in the number of outputs to 40 (third from

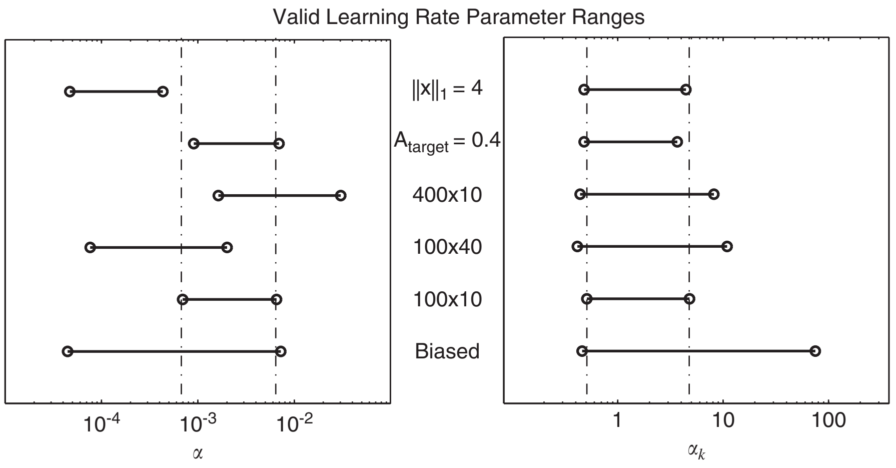

Fig. 2. Valid learning rate parameter ranges. For various networks, the valid range of α values is shown on the left. On the right, the corresponding α_k values are shown. See text for descriptions of network architectures.

bottom), (2) an increase in the number of inputs to 400 (third from top), (3) an increase in A_{target} to 0.4 (second from top), and (4) an increase in $\|\vec{x}\|_1$ to 4 (top). Using the scaling equation, the corresponding α_k values are computed and plotted in the right panel. The ranges of valid α_k line up very well, making the job of changing network architecture much easier. A user merely needs to set a good α_k learning rate parameter. When the architecture changes, the parameter will likely not need to be changed. This eliminates the need for many iterations looking for a valid learning rate.

In this work, we have used a homeostatic synaptic scaling rule in place of the standard weight normalization within an SOM algorithm. This biologically plausible mechanism still results in a proper map. Here, the usefulness of this type of mechanism is expanded by deriving a scaling equation for the learning rate for various network architectures.

Acknowledgments

This material is based upon work supported by the National Science Foundation under NSF Career Grant no. 0133996 and was also supported by NSF IGERT Grant #DGE-0333451 to GW Cottrell.

References

[1] J.A. Bednar, A. Kelkar, R. Miikkulainen, Scaling self-organizing maps to model large cortical networks, Neuroinformatics 2 (3) (2004) 275–302.

[2] T. Kohonen, Self-Organizing Maps, third ed., Springer, Berlin, NY, 2001.

[3] J. Sirosh, R. Miikkulainen, Topographic receptive fields and patterned lateral interaction in a self-organizing model of the primary visual cortex, Neural Comput. 9 (3) (1997) 577–594.

[4] T.J. Sullivan, V.R. de Sa, Homeostatic synaptic scaling in self-organizing maps, in: M. Cottrell (Ed.), Workshop on Self-Organizing Maps, Paris, 2005.

[5] G.G. Turrigiano, S.B. Nelson, Homeostatic plasticity in the developing nervous system, Nat. Rev. Neurosci. 5 (2004) 97–107.

Thomas J. Sullivan studied electrical engineering at the University of Arizona and graduated in 1996. Afterwards, he worked as an analog circuit designer for Burr–Brown Corporation and Gain Technology in Tucson, Arizona. He is currently a Ph.D. student at the University of California, San Diego. His research interests include computational models of the brain, computer vision systems, and circuit design.

Virginia R. de Sa received her Ph.D. in Computer Science in 1994 from the University of Rochester. She then performed postdoctoral work in Computer Science at the University of Toronto on an NSERC fellowship and then at the University of California at San Francisco in Theoretical Neurobiology as a Sloan fellow. She joined the Department of Cognitive Science at the University of California at San Diego in 2001. She is a member of the interdisciplinary programs in Neuroscience, Computational Neuroscience, and Cognitive Science. Her current research combines theoretical and experimental approaches to investigate vision and learning in humans and machines. Specific interests include studying the computational properties of early visual neurons, cortical feedback, and multi-sensory integration.