Learning in the Multiple Class Random Neural Network

Erol Gelenbe and Khaled Hussain
School of Electrical Engineering & Computer Science
University of Central Florida
Orlando, FL 32816
{erol, khaled}@cs.ucf.edu

September 2, 2002

Abstract

Learning is one of the most important useful features of artificial neural networks. In engineering applications, neural network learning is used widely to capture relationships between sets of data when input-output examples are available and a mathematical representation of the relationship is not available in advance. Networks which have "learned" are then capable of "generalization". Spiked recurrent neural networks with "multiple classes" of signals have been recently introduced (Gelenbe and Fourneau 1999), as an extension of the recurrent spiked random neural network introduced by Gelenbe (1989, 1993). These new networks can represent interconnected neurons which simultaneously process multiple streams of data such as the color information of images, or networks which simultaneously process streams of data from multiple sensors. This paper introduces a learning algorithm which applies both to recurrent and feedforward multiple signal class random neural networks (MCRNN). It is based on gradient descent optimization of a cost function. The algorithm exploits the analytical properties of the MCRNN and requires the solution of a system of nC linear and nC non-linear equations (where C is the number of signal classes and n is the number of neurons) each time the network learns a new input-output pair. Thus the algorithm is of $O([nC]^3)$ complexity for the recurrent case, and $O([nC]^2)$ for a feedforward MCRNN. Finally we apply this learning algorithm to color texture modeling (learning), based on learning the weights of a recurrent network directly from the color texture image. The same trained recurrent network is then used to generate a synthetic texture that imitates the original. This approach is illustrated with various synthetic and natural textures.

Keywords: Learning, Recurrent networks, Neural networks, Random Neural Network (RNN), Multiple Class RNN, Color Image Textures.

1 INTRODUCTION

The human brain is a massive, parallel, natural computer composed of some 10^{11} neurons. Each biological neuron is a complex analog processor. Computational neurobiologists are interested in very detailed computer models of neurons, while the artificial neural network community is more interested in the general properties of simplified and tractable artificial neural nets. Thus relatively simple models are used in neural computation. Artificial neural networks have gained popularity due to their success in diverse applications. They are capable of capturing underlying numerical or logical relationships on the basis of examples which are given to them. Many artificial neural network models have been developed and "backpropagation" [34, 38] is

a well-known algorithm used to train feedforward multilayer perceptron networks. A recurrent network on the other hand, has an underlying signal flow graph which contains cycles. Although learning in recurrent networks is less well known, extensions of backpropagation to recurrent networks has been discussed by several authors [35, 36, 7, 37, 18].

The RNN [15, 16, 18] is a a spiked recurrent stochastic model which differs substantially from standard models [2, 7, 35, 36, 37, 38]. It possesses attractive analytical properties [18, 23] such as "product form" and the existence and uniqueness of the network's steady-state solution. It has been used to solve engineering problems from diverse areas such as image and video compression [11, 20], pattern recognition [6], texture modeling [4, 5], associative memory [17], and combinatorial optimization [22, 25, 26]. The RNN represents more closely the manner in which signals are transmitted in many biological neural networks where they travel as spikes [32], rather than as fixed analog signals.

The RNN is easy to simulate, since each neuron is represented by a counter, which counts inhibitory spikes downwards (-1) and excitatory spikes upwards (+1), and can therefore lead to a simple hardware implementation [1]. Furthermore, the RNN model represents neuron potential as an integer, rather than as a binary variable, which leads to a more detailed state representation. For instance, in the RNN the "average potential" of individual neurons (which takes values in $[0, \infty]$) may also be used as the significant output variable of a neuron, rather than just the binary neuron state (on or off), or the value of neuron state represented by a real number in the interval [0,1].

In [19] we use a "single class" RNN to classify grey level textures in Magnetic Resonance (MR) images of the human brain. The purpose of that work was to measure accurately the size of different brain areas in a human subject by being able to discriminate between these areas via their texture as represented in a MR image. The approach consists of learning the weights of a recurrent RNN which is associated with the texture of a particular area of the brain (e.g. grey matter), and then uses the trained network to discriminate between pixels for that area and other pixels in the MR image. The technique presented in [19] is used to discriminate between different brain areas with different texture types with a high degree of accuracy.

The MCRNN model was proposed by Gelenbe and Fourneau [21] as a generalization of the RNN model, in order to develop a mathematical framework to represent networks which simultaneously process information of different types. Thus, spikes belonging to different "classes" may represent different frequencies in a sound-processing network, or different colors in an image-processing network, or the inputs from different sensors in a multi-sensory system. Because this is a spiked model based on the rates at which spikes arrive into the network and on the rate with which spikes are exchanged between neurons, the network inputs themselves are spike trains. In the MCRNN only excitatory spikes belong to multiple signal classes, while inhibitory spikes will belong to a single class.

This paper presents a learning algorithm which applies both to recurrent and feedforward MCRNNs. It is based on gradient descent optimization of a cost function. The algorithm exploits the analytical properties of the MCRNN and requires the solution of a system of nC linear and nC non-linear equations (where C is the number of signal classes and n is the number of neurons) each time the network learns a new input-output pair. Thus the algorithm is of $O([nC]^3)$ complexity for the recurrent case, and $O([nC]^2)$ for a feedforward MCRNN. This algorithm differs from the RNN learning algorithm in a significant manner

In Section 4 we apply this learning algorithm to color texture modeling (learning). This example is chosen because it provides a visible representation of how the learning algorithm can be applied. The MCRNN learning algorithm takes into account the interactions between

the different colors in the texture, and this could not have been handled directly by using our previous "single class" RNN algorithm which could have been used to learn the texture for each color separately. based on learning the weights of a recurrent network directly from the color texture image. The same trained recurrent network is then used to generate a synthetic texture that imitates the original. This approach is illustrated with various synthetic and natural textures.

The organization of this paper is as follows. Section 2 describes the MCRNN; its learning algorithm is presented in Section 3. An example application to the learning of color image textures is given in Section 4.

2 The Mathematical Model

Consider a multiple class random neural model as presented in [21], consisting of n neurons. The state of the $i-th$ neuron is represented by the vector $\underline{k}_i = (k_{i1}, ..., k_{iC})$, where k_{ic} is defined as the potential of class c at neuron i. Note that each k_{ic} is non-negative.

The total potential of neuron i is simply the sum of potentials of each class and is denoted by $k_i = \sum_{c=1}^{C} k_{ic}$. The state of the network as a whole is the vector $\underline{k} = (\underline{k}_1, \underline{k}_2, \ ... \ , \underline{k}_n)$.

All spikes in this model arrive at random time instants to neurons. When an excitatory spike arrives, it changes the state of the neuron in a deterministic manner which is simply determined by the "class" of the spike. When an inhibitory spike arrives, it first selects a class at random, and then changes the state of the neuron deterministically. Excitatory spikes are class dependent, while inhibitory spikes are not class dependent. Although it would have been nice to have inhibitory class dependent spikes, in our earlier work [21] we were unable to extend the analytical treatement detailed in *Theorem 1* to the more general case of negative spikes belonging to multiple classes. State transitions in the model occur because of the following events:

1. Excitatory or inhibitory spikes from the "outside world" (exogenous spike arrival) arrive to a neuron,
2. A neuron fires and sends excitatory or inhibitory spikes to some other neuron,
3. A neuron fires and sends spikes to the "outside world" rather than to some other neuron.

These events are detailed below in the same order:

1. A neuron i in the network can receive exogenous (i.e. external) excitatory signals of class c in independent Poisson arrival streams of rate $\Lambda(i, c)$. When it receives an excitatory signal of class c, the potential of class c of neuron i denoted (k_{ic}) will be increased by one and take the new value $(k_{ic}) + 1$. A neuron i may also receive exogenous inhibitory spikes in the form of a Poisson arrival process of rate $\lambda(i)$. Note that these inhibitory spikes do not have class designation.

2. When an excitatory spike of class c arrives from neuron i to neuron j in class ξ, the potential k_{ic} of class c at neuron i will be decreased by one becoming $k_{ic} - 1$, while the potential of class ξ of neuron j will be increased by one to $k_{j\xi} + 1$. When an inhibitory spike arrives from neuron i in class c to neuron j, if k_j is positive then it will become $k_j - 1$, and the potential of some

class ξ at neuron j will be reduced by 1 with probability $\frac{k_{j\xi}}{k_j}$, becoming $k_{j\xi} - 1$. Note that if $k_{ic} = 0$ and $k_i > 0$, the class c potential will not be affected, but some other potential at this neuron may be affected with the corresponding probability.

3. Finally, a neuron i may fire a spike of class c which then leaves the network (i.e. it does not go to some other neuron). In this case, only the potential k_{ic} of class c at neuron i will be decreased by one becoming $k_{ic} - 1$ while no other neuron's potential will be affected by the event.

Every neuron whose potential is positive ($k_i > 0$) has exponentially distributed random firing times depending on the signal class. The rate at which some neuron i will fire and emit a spike of class c is $r(i,c)\frac{k_{ic}}{k_i}$, where $\frac{k_{ic}}{k_i}$ is the probability that firing will result in the emission of a spike of class c due to the relative strength of the class c potential, and $r(i,c)$ is the firing rate of neuron i for class c. If $k_i = 0$ then each $k_{ic} = 0$, so that no firing will occur. When neuron i fires a class c spike, the spike travels to some other neuron, or it leaves the network.

We represent the movement of spikes using probabilistic transition matrices. We denote by $P^+(i,c;j,\xi)$ the probability that the spike goes from neuron i in class c to neuron j in class ξ, and that it becomes an *excitatory* spike at neuron j. Notice that the + sign is used in the subscript of $P^+(i,c;j,\xi)$ to denote excitation. On the other hand, $P^-(i,c;j)$ denotes the probability that the spike goes from neuron i in class c to neuron j, and that it becomes an *inhibitory* spike at neuron j. Thus inhibitory spikes do not have a "class" designation, while excitatory spikes can belong to different classes. This is consistent with our earlier statement that only excitatory spikes belong to multiple signal classes, while inhibitory spikes will belong to a single class.

Since all spikes need to be accounted for, for any (i,c) we will have:

$$\sum_{(j,\xi)} P^+(i,c;j,\xi) + \sum_j P^-(i,c;j) + d(i,c) = 1, \tag{1}$$

where $d(i,c)$ is the probability that neuron i fires a spike of class c which is then directed outside the network rather to some other neuron. In the sequel we will deal with the "network weights" which are defined as:

$$\begin{aligned} w^+(i,c;j,\xi) &= r(i,c)P^+(i,c;j,\xi), \\ w^-(i,c;j) &= r(i,c)P^-(i,c;j), \end{aligned}$$

and we designate the corresponding weight matrices by W^+ and W^-. These weights correspond to rates of arrival of excitatory and inhibitory spikes from some neuron i to some other neuron ξ.

Following [21], we use the following notation to describe the states of the model:

$$\begin{aligned} \underline{k} + e_{jc} &= (\underline{k}_1, ..., (k_{j1}, ..., k_{jc}+1, ..., k_{jC}), ..\underline{k}_n) \\ \underline{k} + e_{ic} &= (\underline{k}_1, ..., (k_{i1}, ..., k_{ic}+1, ..., k_{iC}), ..\underline{k}_n) \\ \underline{k} + e_{ic} - e_{j\xi} &= (\underline{k}_1, ..., (k_{i1}, ..., k_{ic}+1, ..., k_{iC}), .., (k_{j1}, ..., k_{j\xi}-1, ..., k_{jC}), ...\underline{k}_n) \\ \underline{k} + e_{ic} + e_{j\xi} &= (\underline{k}_1, ..., (k_{i1}, ..., k_{ic}+1, ..., k_{iC}), .., (k_{j1}, ..., k_{j\xi}+1, ..., k_{jC}), ...\underline{k}_n) \end{aligned}$$

Let $p(\underline{k},t)$ denote the probability that at time t the network is in state $\underline{k}$. From the preceding discussion, the global time dependent behaviour of the model (see [21]) can be described by a system of Chapman-Kolmogorov equations (see for instance Feller (1971) [14]) which the probability distribution $p(\underline{k},t)$ must satisfy the following equations:

$$\frac{dp(\underline{k},t)}{dt} = -p(\underline{k},t)\sum_{(i,c)}\left[\Lambda(i,c)+(\lambda(i)+r(i,c))(\frac{k_{ic}}{h_i})\right]$$
$$+\sum_{(i,c)}\{p(\underline{k}+e_{ic},t)r(i,c)(\frac{k_{ic}+1}{k_i+1}d(i,c))$$
$$+p((\underline{k}-e_{ic},t)\Lambda(i,c)1[k_{ic}>0]+p(\underline{k}+e_{ic},t)\lambda(i)(\frac{k_{ic}+1}{k_i+1})$$
$$+\sum_{(j,\xi)}(p(\underline{k}+e_{ic}-e_{j\xi},t)r(i,c)(\frac{k_{ic}+1}{k_i+1})P^+(i,c;j,\xi)1[k_{j\xi}>0]$$
$$+p(\underline{k}+e_{ic},t)r(i,c)(\frac{k_{ic}+1}{k_i+1})P^-(i,c;j)1[k_j=0]$$
$$+p(\underline{k}+e_{ic}+e_{j\xi},t)r(i,c)(\frac{k_{ic}+1}{k_i+1})(\frac{k_jc+1}{k_j+1})P^-(i,c;j))\} \quad (2)$$

2.1 Analytical Results for Steady-State

The Chapman-Kolmogorov equations (2) for the network model are a countably infinite system of ordinary differential equations, with one equation for each state $\underline{k}$ of the network. Since the value of the potential of any neuron is unbounded, we have an infinite number of equations. Fortunately, the network has an elegant stationary (steady-state) solution, as summarized below. Let

$$p(\underline{k}) = \lim_{t\to\infty} p(\underline{k},t) \quad (3)$$

be the stationary probability distribution of network state, if it exists. From [21] we have:

Theorem 1 (Gelenbe and Fourneau 1999) Consider the following system of equations for $\lambda^+(i,c)$, $\lambda^-(i)$, $1\leq i\leq n$, $1\leq c\leq C$, which represent the <u>total</u> average arrival rates of positive and negative signals to each neuron:

$$\lambda^+(i,c) = \sum_{(j,\xi)} q_{j\xi}r(j,\xi)p^+(j,\xi;i,c) + \Lambda(i,c), \quad (4)$$

$$\lambda^-(i) = \sum_{(j,\xi)} q_{j\xi}r(j,\xi)p^-(j,\xi;i) + \lambda(i), \quad (5)$$

and consider the following quantities:

$$q_{ic} = \frac{\lambda^+(i,c)}{r(i,c)+\lambda^-(i)} \quad (6)$$

If the $q_{ic} < 1$ for all $i = 1,..,n$ and $c = 1,...,C$, then stationary probability distribution $p(\underline{k})$ exists and is given by:

$$p(\underline{k}) = \prod_{i=1}^{n}(k_i!)G_i\prod_{c=1}^{C}\left[\frac{(q_{ic})^{k_{ic}}}{k_{ic}!}\right], \quad (7)$$

where the G_i, $i = 1, \ldots, n$ are appropriate normalizing constants.

Notice that (4), (5), and (6) are a system of non-linear algebraic equations whose solution will yield the quantities $q(i,c)$; once these are obtained all relevant firing rates, moments etc.

related to the model can be obtained. Because the underlying model is described by Chapman-Kolmogorov equations, whenever there is a solution, it is necessarily unique and is therefore given by the above product form formula, as shown in [21].

For notational convenience let us write

$$\begin{aligned} w^+(j,\xi;i,c) &= r(j,\xi)p^+(j,\xi;i,c) \geq 0, \\ w^-(j,\xi;i) &= r(j,\xi)p^-(j,\xi;i) \geq 0, \\ N(i,c) &= \sum_{(j,\xi)} q_{j\xi} w^+(j,\xi;i,c) + \Lambda(i,c), \\ D(i,c) &= r(i,c) + \sum_{(j,\xi)} q_{j\xi} w^-(j,\xi;i) + \lambda(i). \end{aligned}$$

Then (6) becomes

$$q_{ic} = \frac{N(i,c)}{D(i,c)} \tag{8}$$

and

$$r(i,c) = \sum_{(j,\xi)} w^+(i,c;j,\xi) + \sum_j w^-(i,c;j). \tag{9}$$

2.2 Existence and Uniqueness of Network Solutions

As with most neural network models, the signal flow equations (4), (5), and (6), which relate the probability that neuron i is excited to the flow of inhibitory or excitatory signals, and to the neuron firing rates, are non-linear. These equations are essential to the construction of the learning algorithm described above. In this section we recall the necessary and sufficient conditions for the existence of the solutions to these equations as proved in [21].

Let us write the equations for $\lambda^+(i,c)$, $\lambda^-(i)$, $1 \leq i \leq n$, $1 \leq c \leq C$, in the following manner:

$$\begin{aligned} \lambda^+(i,c) &= \sum_{(j,\xi)} P^+(j,\xi;i,c) r(j,\xi) \frac{\lambda^+(j,\xi)}{r(j,\xi) + \lambda^-(j)} + \Lambda(i,c) \\ \lambda^-(i) &= \sum_{(j,\xi)} P^-(j,\xi;i) r(j,\xi) \frac{\lambda^+(j,\xi)}{r(j,\xi) + \lambda^-(j)} + \lambda(i) \end{aligned} \tag{10}$$

Now define the following vectors:

- $\underline{\lambda}^+$ with elements $\lambda^+(i,c)$,
- $\underline{\lambda}^-$ with elements $\lambda^-(i)$,
- $\underline{\Lambda}$ with elements $\Lambda(i,c)$,
- $\underline{\lambda}$ with elements $\lambda(i)$,

and let F be the diagonal matrix with elements $f_{ic} = \frac{r(i,c)}{r(i,c)+\lambda^-(i)} \leq 1$. Then (10) may be written as:

$$\underline{\lambda}^+(I - FP^+) = \underline{\Lambda}, \tag{11}$$

$$\underline{\lambda}^- = \underline{\lambda}^+ FP^- + \underline{\lambda} \tag{12}$$

In Gelenbe and Fourneau 1999 [21] it is shown that quations (11), (12) always have an unique solution. If this solution results in a fixed-point $\underline{y}^*$ such that $\lambda^+(i,c) \geq [r(i) + \lambda^-(i)]$, then the stationary solution for class c of neuron i does not exist and we set $q_{ic}(\underline{y}^*) = 1$. If we obtain $\lambda^+(i,c) < [r(i) + \lambda^-(i)]$, then we set

$$q_{ic}(\underline{y}^*) = \frac{\lambda^+(i,c)}{r(i) + \lambda^-(i)}. \tag{13}$$

The stationary solution $p(\underline{k}) > 0$ for all $\underline{\mathrm{k}}$ of the network exists if $q_{ic}(\underline{y}^*) < 1$ for c=1, ..., C and i=1, ..., n.

3 The Learning Algorithm

The central purpose of this paper is to present a gradient based algorithm for selecting the weights of a recurrent MCRNN, so that we may select a set of weights W so as to best "match" a given set of K input-output pairs in a manner which is mathematically meaningful.

The $k-th$ input will be the arrival rates of excitatory and inhibitory signals to the n neurons for each class of signals (Λ_k, λ_k). Recall that in the MCRNN only excitatory spikes belong to multiple signal classes, while inhibitory spikes will belong to a single class. Thus the $k-th$ input is:

$$\begin{aligned} \Lambda_k &= (\Lambda_{11}(k), ..., \Lambda_{C1}(k), ..., \Lambda_{1n}(k), ..., \Lambda_{Cn}(k)) \\ \lambda_k &= (\lambda_1(k), ..., \lambda_n(k)), \end{aligned}$$

The corresponding desired outputs will be specified in terms of neurons and classes. Thus the $k-th$ desired output is a vector:

$$y_k = (y_{11}(k), ..., y_{1C}(k), ..., y_{n1}(k), ..., y_{nC}(k),$$

where each element $y_{ic}(k)$ is a real number. The network weights are then selected to approximate the desired output vector y_k for the $k-th$ input $\iota_k = (\Lambda_k, \lambda_k)$ so as to minimize a "cost" or "error" function which we denote by E_k. Let $f_{ic}(x)$ be some function of the real variable x, defined for $x \in [0,1)$. Assume that it is differentiable. The cost function we consider is of the form:

$$E_k = \frac{1}{2} \sum_{(i,c)} a_{ic}(f_{ic}(q_{ic}^k) - y_{ic}(k))^2, \tag{14}$$

where the $a_{ic} \geq 0$ are constants we use to provide greater or lesser importance to the different neurons and classes in the cost function, and $q_{ic}(k)$ is the stationary state of neuron i in class c when the $k-th$ input $\iota_k = (\Lambda_k, \lambda_k)$ is applied to the network. Note that any neuron in the recurrent network may contribute to the cost function, so that we do not limit ourselves to a particular set of output neurons. Obviously, we can limit ourselves to certain output neurons in the cost function simply by setting certain coeffients a_{ic} to zero. Another point that needs to be made is that the quadratic form of E_k in (14) is not needed; any differentiable form can also be used, and the extension to that case is straightforward.

Our algorithm lets the network learn both the $nC \times nC$ excitation weight matrix $W_k^+ = \{w_k^+(i,c;j,\xi)\}$ and $nC \times n$ inhibition weight matrix $W_k^- = w_k^+\{(i,c;j)\}$ by computing for each input $\iota_k = (\Lambda_k, \lambda_k)$, a new value W_k^+ and W_k^- of the weight matrices, using gradient descent.

Clearly, because of the manner in which this model is defined where these weights represent firing rates, we only have non-negative values of the weights.

The weights are obtained via gradient descent which updates them to seek a local minimum of the cost function for each successive input, similar to that in the earlier algorithm for the random neural network model [18]. Note that the algorithm we present is for a general recurrent network, differing from what is usually found in the literature [38]. The multiple class structure also requires a distinct mathematical derivation leading to significant differences which need to be detailed if the algorithm is to be correctly implemented.

Let $\eta > 0$ be the learning rate which determines by how much we change the weights at each step of the algorithm. τ will denote the "time" step of the algorithm. Then the gradient algorithm to calculate the successive values of the excitatory weights applied to a cost term E_k yields:

$$w^+_\tau(u,d;v,e) = w^+_{\tau-1}(u,d;v,e) - \eta \sum_{ic} a_{ic}(f_{ic}(q_{ic}) - y_{ic}(k)) \frac{\partial f}{\partial q_{ic}} \frac{\partial q_{ic}}{\partial w^+}|_{[w^+ = w^+_{\tau-1}(u,d;v,e); q_{ic} = q^{\tau-1}_{ic}(k)]} \quad (15)$$

$$w^-_\tau(u,d;v) = w^-_{\tau-1}(u,d;v) - \eta \sum_{ic} a_{ic}(f_{ic}(q_{ic}) - y_{ic}(k)) \frac{\partial f}{\partial q_{ic}} \frac{\partial q_{ic}}{\partial w^-}|_{[w^- = w^-_{\tau-1}(u,d;v); q_{ic} = q^{\tau-1}_{ic}(k)]} \quad (16)$$

where $q^{\tau-1}_{ic}(k)$ is the solution of (6) when the weight values are $\{w^+_{\tau-1}(i,c;j,\xi)\}$ and $\{w^-_{\tau-1}(i,c;j)\}$, and the inputs are $\iota_k = (\Lambda_k, \lambda_k)$. Notice that in the above expressions:

- $q^0_{ic}(k)$ is first calculated using the input ι_k, and we set the values $w^+(u,d;v,e) = w^+_{\tau-1}(u,d;v,e)$, and $w^-(u,d;v) = w^-_{\tau-1}(u,d;v)$, in (8) from the last step of the gradient descent iteration for input ι_{k-1}; thereafter it is recalculated to obtain the successive $q^\tau_{ic}(k)$ from the input ι_k and the new values of the successive weights obtained from the gradient descent algorithm.
- The $\frac{\partial f}{\partial q_{ic}} \frac{\partial q_{ic}}{\partial w^+}$ are evaluated at the values:

$$w^+ = w^+_{\tau-1}(u,d;v,e),$$
$$q_{ic} = q^{\tau-1}_{ic}(k),$$

  and the $\frac{\partial f}{\partial q_{ic}} \frac{\partial q_{ic}}{\partial w^-}$ are evaluated at:

$$w^- = w^-_{\tau-1}(u,d;v),$$
$$q_{ic} = q^{\tau-1}_{ic}(k).$$

Let $1[X]$ be the indicator function which takes the value one if X is a true statement while it takes the value 0 if X is false. In order to proceed with the computation of the partial derivatives, we notice that:

$$\begin{aligned} \frac{\partial q_{ic}}{\partial w^+(u,d;v,e)} &= \sum_{j\xi} \frac{\partial q_{ic}}{\partial w^+(u,d;v,e)} \frac{[w^+(j,\xi;i,c) - q_{ic} w^-(j,\xi;i)]}{D(i,c)} \\ &\quad -1[(u,d) \equiv (i,c)] \frac{q_{ic}}{D(i,c)} \\ &\quad +1[w^+(u,d;v,e) \equiv w^+(u,d;i,c)] \frac{q_{ud}}{D(i,c)} \end{aligned} \quad (17)$$

$$\frac{\partial q_{ic}}{\partial w^-(u,d;v)} = \sum_{j\xi} \frac{\partial q_{ic}}{\partial w^-(u,d;v)} \frac{[w^+(j,\xi;i,c) - q_{ic} w^-(j,\xi;i)]}{D(i,c)}$$

$$-1[(u,d) \equiv (i,c)]\frac{q_{ic}}{D(i,c)}$$
$$-1[w^-(u,d;v) \equiv w^-(u,d;i)]\frac{q_{ic}q_{ud}}{D(i,c)} \tag{18}$$

Let $q = (q_{11}, ..., q_{1C}, q_{21}, ..., q_{2C}, ..., q_{n1}, ..., q_{nC})$, and define the nCxnC matrix:

$$W = \frac{[w^+(j,\xi;i,c) - q_{ic}w^-(j,\xi;i)]}{D(i,c)}, \quad for\ i,j = 1,...,n\ and\ c,\xi = 1,...,C. \tag{19}$$

We can now write the vector equations:

$$\frac{\partial q}{\partial w^+(u,d;v,e)} = \frac{\partial q}{\partial w^+(u,d;v,e)}W + \gamma^+(u,d;v,e)q(u,d)$$

$$\frac{\partial q}{\partial w^-(u,d;v)} = \frac{\partial q}{\partial w^-(u,d;v)}W + \gamma^-(u,d;v)q(u,d)$$

where the elements of the nC-vectors

$$\gamma^+(u,d;v,e) =$$
$$(\gamma^+_{11}(u,d;v,e), ..., \gamma^+_{1C}(u,d;v,e), \gamma^+_{21}(u,d;v,e), ..., \gamma^+_{2C}(u,d;v,e), ..., \gamma^+_{n1}(u,d;v,e), ..., \gamma^+_{nC}(u,d;v,e)),$$
$$\gamma^-(u,d;v) =$$
$$(\gamma^-_{11}(u,d;v), ..., \gamma^-_{1C}(u,d;v), \gamma^-_{21}(u,d;v), ..., \gamma^-_{2C}(u,d;v), ..., \gamma^-_{n1}(u,d;v), ..., \gamma^-_{nC}(u,d;v))$$

are:

$$\begin{aligned}
\gamma^+_{ic}(u,d;v,e) &= -\frac{1}{D(i,c)} \quad if (u,d) = (i,c), (v,e) \neq (i,c), \\
&= \frac{1}{D(i,c)} \quad if (u,d) \neq (i,c), (v,e) = (i,c), \\
&= 0 \quad otherwise
\end{aligned} \tag{20}$$

$$\begin{aligned}
\gamma^-_{ic}(u,d;v) &= -\frac{1+q_{ic}}{D(i,c)} \quad if (u,d) = (i,c), v = i, \\
&= -\frac{1}{D(i,c)} \quad if (u,d) = (i,c), v \neq i, \\
&= -\frac{q_{ic}}{D(i,c)} \quad if (u,d) \neq (i,c), v = i, \\
&= 0 \quad otherwise
\end{aligned} \tag{21}$$

This leads to the following simplified representation:

$$\begin{aligned}
\frac{\partial q}{\partial w^+(u,d;v,e)} &= \gamma^+(u,d;v,e)q(u,d)[I-W]^{-1}, \\
\frac{\partial q}{\partial w^-(u,d;v)} &= \gamma^-(u,d;v)q(u,d)[I-W]^{-1}.
\end{aligned} \tag{22}$$

where I denotes the nC by nC identity matrix. Thus the main computational effort in solving (22) is simply to obtain $[I-W]^{-1}$, which can be done in time complexity $O((nc)^3)$. We now have all the building blocks to specify the learning algorithm for the network:

- Initiate the matrices W_0^+ and W_0^- in some appropriate manner. This initiation will be made at random (among non-negative matrices) if no better information is available; in some cases it may be possible to choose these initial values by using a Hebbian rule, or some other rule based on some direct approach.
- Choose a value of η to be used in the gradient descent algorithm (15) and (16).

1. For each successive value of k, starting with $k = 1$ proceed as follows. Set the input values to $\iota_k = (\Lambda_k, \lambda_k)$.
2. Solve the system of non-linear equations (8) with these values. The solution can be obtained with a fixed-point iteration as in [18].
3. Solve the system of linear equations (22) using the results of (2).
4. Using equations (15), (16), and the results of (2) and (3), update the matrices W_k^+ and W_k^-.
5. Repeat Steps (2), (3), and (4) until the change in the cost function or in the new values of the weights is smaller than some predetermined value.

This section shows that the learning algorithm exploits the analytical properties of the MCRNN, so that each gradient descent step is reduced to solving a system of nC linear eaquations (where C is the number of signal classes and n is the number of neurons), which is of complexity $O([nC]^3)$ for the recurrent network. The nC non-linear equations are solved using a fixed-point iteration, with each iteration being of complexity $O([nC]^2)$. Thus the complexity of the algorithm is of $O([nC]^3)$ and reduces to $O([nC]^2)$ for a feedforward network. In the following section we present a significant application to learning and generating image textures.

4 Application of the MCRNN to Learning and Generating Color Textures

Although there is no exact and generally accepted definition of texture, texture analysis is considered to be one of the most important subjects in image processing and computer vision. The task of extracting texture features is crucial for numerous applications, and if one could model and quantify the process by which the human recognizes texture, one could construct a highly successful recognition system.

The but texture feature extraction algorithms are often grouped into four classes: statistical [27], structural [39], spectral [9], and model based [12]. Statistical approaches yield characterizations of textures as smooth, coarse, grainy, and so on using relationship between intensity values of pixels; measures include the entropy, contrast, and correlation based on the gray level co-occurrence matrix. Structural algorithms are based on the image primitives; they regard primitive as the forming element to generate repeating patterns. Spectral techniques are based on properties of the Fourier spectrum and are used primarily to detect global periodicity in the image. Model based texture analysis methods are based on the construction of an image model that can be used not only to describe the texture, but also to synthesize it.

Methods for modeling textures and extracting texture features can be applied in three broad categories of problems: texture segmentation, texture classification, and texture synthesis. Recently, many artificial neural network (NN) architectures for texture analysis have been proposed (for example, see [8, 41, 40]). Typically, these architectures either apply a supervised NN model

for texture classification or an unsupervised NN model for texture segmentation. However very little attention has been devoted to training a NN on a certain texture and then using the trained network to generate a synthetic texture. Markov random fields (MRFs) have been used extensively to texture synthesis because they are able to capture the local (spatial) contextual information in a texture, and can generate a similar texture using the extracted parameters. However, the MRF based operations performed during texture generation are computationally very costly.

In this Section we apply the MCRNN learning algorithm to the problem of color texture modeling (learning), and then use the learned MCRNN weights to generate or synthesize a textuer which is similar to the one that was used for learning. The approach is based on learning the weights of a recurrent MCRNN directly from the color texture image. The same trained recurrent network is then used to generate a synthetic texture that imitates the original one. The proposed texture learning technique is efficient and its computation time is small. Texture generation is also fast. We have tested our method with different synthetic and natural textures. The experimental results show that the MCRNN can efficiently model a large category of color homogeneous microtextures. Finally, statistical features extracted from the co-occurrence matrix of the original and the MCRNN based texture are used to evaluate the quality of fit of our approach.

4.1 MCRNN Based Feature Extraction from Color Textures

In this section, we describe the algorithm we use for extracting the features of a given image texture through training the MCRNN, and encoding these features as weight matrices of the network. The resulting weights are then be used for generating textures that have similar characteristics to the initial texture. The topology of the proposed random network for color texture generation is shown in Figure 1. A rectangular grid of MCRNN neurons is set up so that each pixel in the image corresponds to a neuron in the grid which we denote by $N(i,j)$ for a particular pixel position (i,j). Excitatory and inhibitory connections are assumed to exist only between immediate neighboring neurons as shown in Figure 1. The classes of signals in the MCRNN that we use correspond simply to one of the three colors Red, Green, and Blue. We assume that only the neighboring neurons on the rectangular grid communicate directly; furthermore we assume spatial homogeneity in the texture. Therefore we simplify the notation for the weights connecting the neuron in position (x,y) to the neuron in position (i,j) as follows:

$$\begin{aligned} w^{+}(i-x, j-y, \xi; c) &= w^{+}((x,y), \xi; (i,j), c) \\ w^{-}(i-x, j-y, \xi) &= w^{-}((x,y), \xi; (i,j)) \end{aligned} \tag{23}$$

Since we only deal with locally connected neurons, the values of $(i-x)$ and $(j-y)$ will be limited to $\{0, +1, -1\}$.

We associate the color intensity value function $f(x,y,c)$ to a pixel (x,y). and normalize it so that $0 \leq f(x,y,c) \leq 1$; it is then assigned to the corresponding neuron (x,y) so that $f(x,y,c) = q_{(x,y),c}$.

4.2 Color Texture Modeling (Learning) Procedure

We start with an $N \times H$ image containing an artificial or natural color texture, and we select in the image all $(2n+1) \times (2n+1)$ square windows, where each pixel in a window is associated

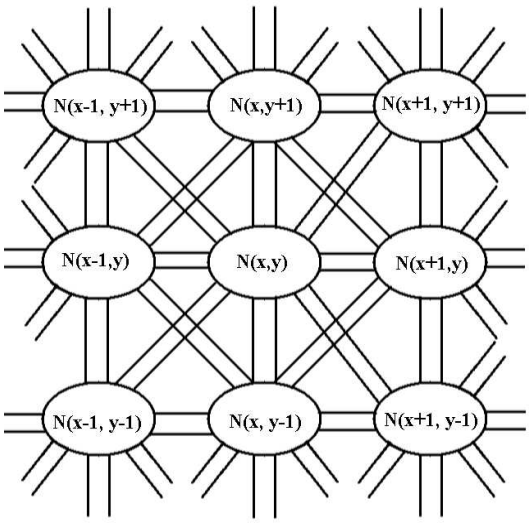

Figure 1: Topology of the random neural network used for texture modeling and generation [5].

with a neuron in the MCRNN. The MCRNN texture model associates a neuron $N(x, y)$ to each point (x, y) in the plane as shown in Figure 1.

Assuming that the texture is spatially homogenous, we will train the MCRNN over *all such, possibly overlapping, windows* in the image, and obtain a set of weights which are representative of the texture. The weights going from the neurons at the edge of the network to the neurons inside the network and the weights which are internal to the network will be considered. Since the $(2n+1) \times (2n+1)$ windows are used to "learn" the weights of a neural network of the same size, the neurons at the edges of the network will only have neighbouring neurons on the edges or inside the network, and not beyond the edges. Thus there will be only be weights going from the edge neurons to the neighbouring edge or internal neurons.

Let K be the total number of overlapping windows in the image; these constitute the K training patterns. Thus each individual input pattern is a set of $(2n+1) \times (2n+1)$ pixels, with red, blue and green color intensities given for each of these pixels. The corresponding output pattern is exactly the same as the input pattern. At each training step we adjust the MCRNN weights so that the sum-squared difference between the input pattern and the state of each neuron in the MCRNN is minimized.

The procedure for texture learning uses the algorithm described earlier, and is summarized as follows:

- Initialize the weight matrices W_0^+ and W_0^- to random values between 0 and 1.
- Set the inhibitory rates $\lambda^{(k)}$ to zero.
- Set the excitatory input rates $\Lambda^{(k)}$ for the MCRNN to the normalized pixel values in the texture. These normalized pixel values are selected as indicated previously, so that for each pixel position (x, y) we have $\Lambda^k(x, y)_c = f(x, y, c)$.
- Set the desired outputs $y^{(k)}$ to the same normalized pixel values in the image so that $y^k(x, y, c) = f(x, y, c)$.
- Solve the nonlinear system of equations (8) to obtain the neuron states $q_{(x,y),c}$.

- Use the MCRNN Learning Algorithm to obtain the network parameters which minimize the cost function:

$$E_k = \frac{1}{2} \sum_{(x,y),c} (q^k_{(x,y),c} - y^k_{(x,y),c})^2. \tag{24}$$

The block diagram in Figure 2 summarizes the algorithm that is used.

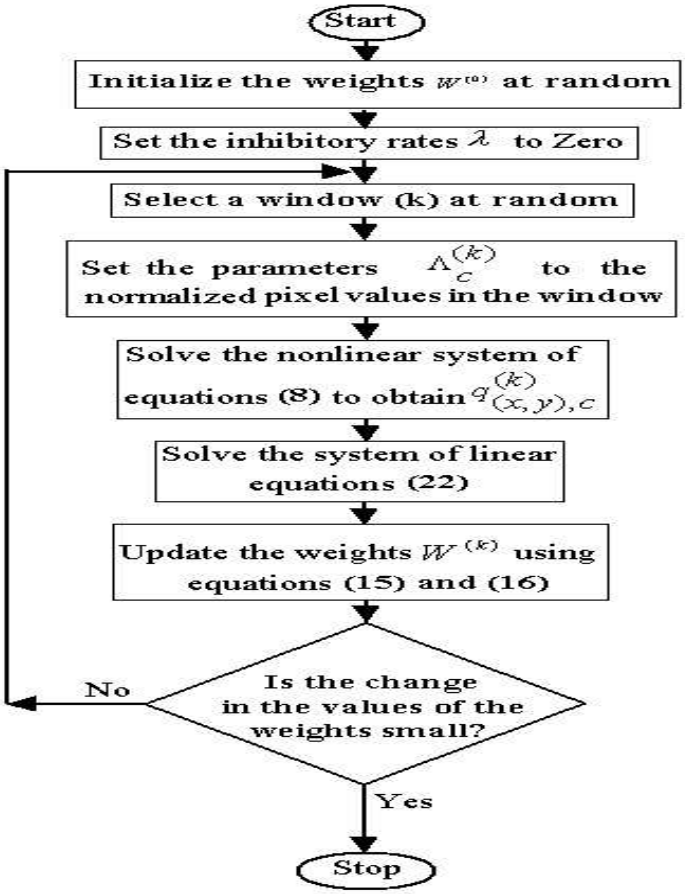

Figure 2: Block diagram of the MCRNN learning algorithm applied to the color texture images

4.3 Texture Synthesis (Generation) Procedure

The approach in this section is similar to the texture generation approach presented by Atalay and Gelenbe [5]; however there is one important difference. In the approach we present here we explicitly use the distinct average values in the original texture for each different color as we initialize the generation algorithm. In our earlier work [5] we only made use of the average overally intensity in initializing the algorithm.

We use the same set of equations (4) and (6) to generate a texture using the MCRNN. We therefore assume that he weights of the neural network have been obtained using the learning algorithm as described in the previous sections. However we still need to select the external parameters $\Lambda_{i,c}$ and λ_i in equations (4) and (6). We interpret the variables i and j in these equations as pixel positions in the plane, and "class" corresponds to color. Thus each $q_{i,c}$ is interpreted as the expected intensity of color c at position $i = (x, y)$.

Since we assume that the texture is homogeneous, the probabilities $q_{i,c}$ at *different positions* i and for the *same color* c should be identical; therefore we will drop the dependency on i and denote it simply by by q_c. Equations (4), (6) then become:

$$q_c = \frac{\sum_{(u,v),\xi} q_\xi w^+((u,v),\xi;,c) + \Lambda_c}{r_c + \sum_{(u,v),\xi} q_\xi w^-((u,v),\xi) + \lambda} \tag{25}$$

where we have used the notation for weights between neighboring in (23) with the assumption of spatial homogeneity, and have dropped the dependency on i in the external parameters $\Lambda_{i,c}$ and λ_i. This simplifies to:

$$q_c = \frac{\sum_\xi q_\xi \alpha_{\xi,c} + \Lambda_c}{r_c + \sum_\xi q_\xi \beta_{\xi,c} + \lambda}, \tag{26}$$

for each color c, where

$$\alpha_{\xi,c} = \sum_{(u,v)} w^+((u,v),\xi;c),$$

$$\beta_{\xi,c} = \sum_{(u,v)} w^-((u,v),\xi),$$

$$r_c = \sum_{(u,v,\xi)} w^+((u,v),c;\xi) + \sum_{(u,v)} w^-((u,v),c)$$

are computed from the weights which have been previously obtained by the learning algorithm.

The two remaining unknowns in the system of equations (26) are now the Λ_c and λ, i.e. $C+1$ unknowns if there are C colors. Since we only have C equations (26), we will arbitrarily set $\lambda = 0$ so that the only external signals into the network used for generating the texture will be the excitatory color signals Λ_c. In order to compute the Λ_c we will use the color intensities $f(x,y,c)$ in the original texture image at each position (x,y), and simply compute their average value over the region of interest whose size is M pixels, normalized to the number L of color intensity levels which are being used:

$$f_c = \frac{1}{LM} \sum_{x,y} f(x,y,c), \tag{27}$$

so that $0 \leq f(c) \leq 1$. Then we we simply set each $q_c = f_c$ in equations (26), which directly yields the unkowns Λ_c.

Assume a "three color" (Red, Green, Blue) texture. Clearly, the network weights could also be fixed according to some external criteria to generate synthetic textures with predefined characteristics [5], or they can be obtained from the learning algorithm applied to a given texture image as discussed in Section 4.2. The procedure for texture generation can now be summarized in the following steps:

1. Set the parameters Λ_c as described above, and set $\lambda = 0$, and fix the weights W to the desired values.

2. Generate at random three values between 0 and 1 for each pixel (x,y) and assign them to the variables $q_0(x,y,red), q_0(x,y,green), q_0(x,y,blue)$. The synthesized color texture will be obtained by computing $q_K(x,y,c)$ for each pixel position (x,y) as follows, where K is an index for the maximum number of iterations of the numerical algorithm described below.

3. Set $k = 0$

4. **For each** (x, y) in the pixel space that the texture will fill, **compute**:

$$\lambda^+_{k+1}(x, y, c) = \sum_{l=x-1}^{x+1} \sum_{m=y-1}^{y+1} \sum_{\xi} q_k(l, m, \xi) w^+(x - l, y - m, \xi; c) + \Lambda_c,$$

$$\lambda^-_{k+1}(x, y) = \sum_{l=x-1}^{x+1} \sum_{m=y-1}^{y+1} \sum_{\xi} q_k(l, m, \xi) w^-(x - l, y - m, \xi),$$

5. **Calculate**:

$$q_{k+1}(x, y, c) = \frac{\lambda^+_{k+1}(x, y, c)}{r_c + \lambda^-_{k+1}(x, y)} \tag{28}$$

6. **Update** $k \leftarrow k + 1$

7. **if** $k \leq (K - 1)$ **goto** Step 4, **else** end.

Now from the values $q_K(x, y, c)$ remaining at the end of the procedure, we simply compute the corresponding pixel values by setting $f(x, y, c) = L\ q_K(x, y, c)$.

4.4 Experimental Results with Synthetic Color Textures

The major challenge of learning and reproducing textures is to capture and reproduce their features without actually producing a "copy" of the original texture. In this section we illustrate the power of the algorithms we have developed by using them on both synthetic and natural textures.

We will first illustrate the texture learning and generation procedure by considering examples of purely synthetic textures generated by the MCRNN using various settings of the weights. We will then consider a set of real image textures selected for their diverse natural characteristics. All the textures we present here have 256 levels for each of the three fundamental colors. Each image we present is of size $128 \times 128 \times 3$ bytes.

In Figure 3, (a), (c), (e), (g), (i) and (k) we present a set of synthetic textures. These have been generated by the MCRNN texture synthesis procedure with fixed sets of weights. Then, the texture learning procedure is applied to each of these texture images, and the corresponding sets of weights are learned. Finally, the resulting learned weights are used in generating each of the corresponding images in Figure 3 (b), (d), (f), (h), (j).

The visual comparison of the two sets of images reveals strong similarities between the original and the "reproduced" (after learning) synthetic textures, and it can be seen that these textures are indeed far from being identical copies. Thus they differ in detail while being very similar, as they should be from our intuitive notion of textures.

However it is important to compare the original textures and the generated textures using a quantitative approach. We will do this based on metrics obtained from the well-known co-occurrence matrix. We will be using *second-order* statistics in these comparisons, and one may wonder whether we should use higher order statistics. However experts on visual perception of textures [30, 31] indicate that textures which have very close values of first and second order statistics are visually indistinguishable; i.e. if their second order statistics are similar but higher

order statistics are different, they cannot be distinguished from the point of view of human visual perception [30, 31].

In [31] it is said that (quote): " ... the conjecture that no *texture pairs can be discriminated if they agree in their second-order statistics* seems to hold for a surprisingly large class of textures. Although a few rather weak counter-examples have been found, the conjecture can still be maintained with only minor modifications." (end quote). Thus we will compare the textures learned and then generated by the MCRNN with the original textures using only second order statistics.

As an example, the texture in Figure 3 (a) is generated using the following parameters; when the parameters are not specified they are set to zero:

$w^+(0,1,R,R) = 1$, $w^+(0,-1,R,R) = 1$, $w^+(1,1,G,G) = 1$, $w^+(-1,-1,G,G) = 1$,
$w^+(-1,1,G,G) = 5$, $w^+(1,-1,G,G) = 5$, $w^+(1,0,B,B) = 1$, $w^+(-1,0,B,B) = 1$,
$w^+(1,1,B,B) = 4$, $w^+(-1,-1,B,B) = 4$, $w^+(-1,1,B,B) = 4$, $w^+(1,-1,B,B) = 4$,
$w^+(1,1,R,B) = 2$, $w^+(-1,-1,R,B) = 2$, $w^+(-1,1,R,B) = 2$, $w^+(1,-1,R,B) = 2$,
$w^+(1,0,G,B) = 1$, $w^+(-1,0,G,B) = 1$, $w^+(0,1,G,B) = 2$, $w^+(0,-1,G,B) = 2$,
$w^-(1,0,R) = 2$, $w^-(-1,0,R) = 2$, $w^-(-1,1,R) = 1$, $w^-(1,-1,R) = 1$,
$w^-(1,0,G) = 2$, $w^-(-1,0,G) = 2$, $w^-(1,1,G) = 0.5$, $w^-(-1,-1,G) = 0.5$, $w^-(0,1,G) = 5$,
$w^-(0,-1,G) = 5$,
$w^-(-1,1,G) = 1$, $w^-(1,-1,G) = 1$, $w^-(1,0,B) = 6$, $w^-(-1,0,B) = 6$,
$w^-(1,1,B) = 3$, $w^-(-1,-1,B) = 3$, $w^-(0,1,B) = 0.5$, $w^-(0,-1,B) = 0.5$,
$w^-(-1,1,B) = 1$, $w^-(1,-1,B) = 1$,

Once the image texture is generated by the MCRNN, the texture learning algorithm is applied to the synthetic texture image. The "learned" weights are obtained using the MCRNN learning algorithm with 2000 gradient descent iterations for each weight. Finally the MCRNN is used once again as a texture generator to obtain the texture shown in Figure 3 (b).

A well-known statistical feature that is often used to characterize textures is the co-occurrence matrix [28, 29] which can be used to represent similarity or dissimilarity between two given textures. Identical co-occurrence matrices are an indication of strong similarity between textures.

In order to quantitatively evaluate the similarity between the original texture, and the texture which is generated using the MCRNN with parameters which have been learned from the original texture, we will use a "co-occurrence matrix" P. Note again that we limit our comparisons to second-order statistics as suggested in [30, 31].

Note that the color levels of pixels in the images we deal with can take one of 2^{24} values. Let i and j be color pixel values, while (k,l) and (m,n) denote pixel locations in the image. $f(k,l)$ denotes the pixel value at location (k,l) and it includes color information since it can take one of 2^24 values. The elements $P(i,j)$ of the co-occurrence matrix P are defined by:

$$P(i,j) = \sum_{all\ k,l} \{[f(k,l) = i\ \&\ f(k,l+1) = j]\ OR\ [f(k,l) = j\ \&\ f(k,l+1) = i]\} \qquad (29)$$

so that each $P(i,j)$ simply counts the number of adjacent pixels in the y axis which have neighboring values i followed by j or vice-versa. Note that this co-occurrence matrix does not consider the effect of the x direction so that it is only useful if the texture is homogeneous along the x-axis. Since P is very large, it cannot be used readily to compare textures directly.

Therefore we define a certain number of properties of P. These are:

$$Energy = \sum_{i,j}[P(i,j)]^2,$$

$$Contrast = \sum_{i,j}(i-j)^2 P(i,j)$$

$$Entropy = \sum_{i,j} P(i,j) log P(i,j)$$

$$Homogeneity = \sum_{i,j} \frac{P(i,j)}{1+|i-j|}$$

These four properties of the original synthetic textures of this section, and of the corresponding learned and then generated textures shown in Figure 3, are listed for comparison in Table 1. The very small difference between the numerical data for the original and generated provides quantitative confirmation of the effectiveness of the MCRNN learning and generation algorithm.

Table 1: Statistical features extracted from the co-occurrence matrix for textures of Figure 3

Figure	Description	Energy	Contrast	Entropy	Homogeneity
2(a)	specified	5510801	77706402	95169	16349
2(b)	learned	5249483	80420832	94678	16134
2(c)	specified	1108741	51607808	56970	15888
2(d)	learned	1104047	53147734	56300	15838
2(e)	specified	10327819	12398773	119302	28405
2(f)	learned	9241897	13270037	115089	28450
2(g)	specified	100821229	7339544	190592	35305
2(h)	learned	80766097	7386374	185923	34558
2(i)	specified	161171	194922033	18363	5869
2(j)	learned	162763	197518167	18480	5755
2(k)	specified	529091	35986526	49026	11896
2(l)	learned	535347	36737689	49489	11907

4.5 Modeling Natural Textures

We now apply the texture learning procedure to some natural texture images. For each of the textures, the MCRNN weights are first obtained using the learning algorithm. In the training phase, we have observed that it is not necessary to use all the overlapping 3×3 windows as training patterns. For each training window we have used 2000 iterations of the learning algorithm.

Once the training is complete, a complete network covering the whole image is constructed with the weights obtained for the 3×3 window, and is then used to generate the texture. We compare the original texture and the generated texture by calculating the co-occurrence matrix as in Section 4.4 and the tabulating the same set of four properties. The visual results are presented in Figure 4, while Table 2 summarizes the numerical comparison of the texture

propertise. Again, we observe an excellent visual and numerical similarity between the original and generated color textures.

Table 2: Statistical features extracted from the co-occurrence matrix for textures of Figure 4.

Figure	Description	Energy	Contrast	Entropy	Homogeneity
3(a)	natural	66747545	5381953	146747	18341
3(b)	synthetic	63745447	5436059	145955	18330
3(c)	natural	146670113	1142197	162480	33509
3(d)	synthetic	166515201	1093520	164130	33955
3(e)	natural	5443935	39938879	95299	7820
3(f)	synthetic	5157309	41993308	94591	8015

5 Conclusions and Further Work

In this paper we have first recalled the MCRNN and the main analytical results concerning this model, and in particular the product form solution and the existence-uniqueness results concerning the analytical solution to this non-linear model.

Then we have described a learning algorithm for the recurrent and the feedforward MCRNN. It is based on gradient descent minimization of a cost function. The algorithm exploits the analytical properties of the MCRNN and reduces the gradient descent algorithm to the solution of a system of nC linear equations nC non-linear equations.

To illustrate the use of this network and of its learning algorithm we have applied it to a difficult problem in image analysis: the characterization and generation of color image textures. We have used the learning algorithm to learn the weights of a recurrent network directly from the color texture image. The trained recurrent network has then been used via a relaxation algorithm to generate a synthetic texture that imitates the original. This approach has been illustrated with both synthetic and natural textures, and we have used a variety of standard texture metrics to compare the originals with the equivalent "learned and generated" textures. The metrics used are Energy, Contrast, Entropy and Homogeneity. The differences in the metrics for each texture considered range from at most 6% or so for the Energy, to well under 1% for Homogeneity. This, as well as the visual comparison, illustrates the effectiveness both of the MCRNN and of its learning algorithm.

The work described in this paper raises several new questions, both in terms of theoretical development and of applications. The Random Neural Network and the MCRNN capture spiked behavior in artificial neural networks. A natural extension could allow more realism in the mathematical model by including other effects such as neurotransmitters. The question then is whether large network models can still remain analytically and computationally tractable and how such models can be used to develop new learning algorithms that may be useful in engineering applications. We have recently studied the power of the RNN to carry out function approximation [23]; it would be interesting to see whether the MCRNN can provide us with additional capabilities in this area.

Another interesting question comes up when one considers the texture learning and generation application we have described. The ability of certain animals to control skin appearance

and pigmentation in response to a variety of external stimuli is well known. Would it be possible to model in a fairly detailed fashion the actual circuits that control skin pigmentation based on visual control in specific animals? As the tractable neural network models that we can work with become more powerful and detailed, we can hope that they will lead to even more fascinating questions, and to new opportunities for powerful and realistic applications.

References

[1] A.H. Abdelbaki, E. Gelenbe "Analog hardware implementation of the Random Neural Network model", *International Joint Conference on Neural Networks (2000)*, IEEE Press, July 2000.

[2] D.H. Ackley, G.E. Hinton, T.J. Sejnowski,"A learning algorithm for Boltzman machines," *Cognitive Science*, Vol. 9, pp 147-169, 1985.

[3] L.B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," *Proc. IEEE First International Conf. on Neural Networks*, San Diego, CA Vol. II, pp 609-618, 1987.

[4] V. Atalay and E. Gelenbe, N. Yalabik, "Texture generation with the random neural network model," *International Journal of Pattern Recognition and Artificial Intelligence,* vol. 6, no. 1, pp. 131-141, 1992.

[5] V. Atalay and E. Gelenbe,"Parallel algorithm for colour texture generation using the random neural network model," *International Journal of Pattern Recognition and Artificial Intelligence,* vol. 6, no. 2-3, pp. 437-446, 1992.

[6] H. Bakircoglu, E. Gelenbe,"Random Neural Network Recognition of Shaped Objects in Strong Clutter," *SPIE,* Vol. 3307, pp. 22-28, 1998.

[7] H. Behrens, D. Gawronska, J. Hollatz, B. Schurmann, " Recurrent and feedforward backpropagation: performance studies, " in Kohonen, T. et. al. (ed.), *Artificial Neural Networks*, Vol. II, North-Holland, pp 1511-1514, 1991.

[8] R. Chellappa, B. S. Manjunath, and T. Simchony, "Texture segmentation with neural networks," in *Neural Networks for Signal Processing,* Ed. Bart Kosko, Prentice Hall, 1992.

[9] J. M. Coggins and A. K. Jain, "A spatial filtering approach to texture analysis," *Pattern Recognition Letters*, vol. 3, 1985, pp. 195-203.

[10] F. S. Cohen and D. B. Cooper, "Simple parallel hierarachical and relaxation algorithms for segmentation noncausal Markov random fields," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 9, no.1, (1987).

[11] C. Cramer, E. Gelenbe, H. Bakircioglu, "Low bit rate video compression with neural networks and temporal subsampling," *Proceedings of the IEEE*, Vol.84, No. 10, pp. 1529-1543, 1996.

[12] G. C. Cross and A. K. Jain, "Markov random field texture models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 5, 1983, pp. 25-39.

[13] H. Derin and H. Elliott, "Modeling and segmentation of noisy and textured images using Gibbs random fields," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 9, no. 1, pp. 39-55, (1987).

[14] W. Feller, "An Introduction to Probability Theory and its Applications", Vol. I and II, J. Wiley, 1971.

[15] E. Gelenbe, "Random neural networks with negative and positive signals and product form solution," *Neural Computation,* Vol. 1, No.4, pp. 502-510, 1989.

[16] E. Gelenbe, "Stability of the random neural network model,"*Neural Computation,* Vol. 2, No. 2, pp 239-247, 1990.

[17] E. Gelenbe, "Neural networks advances and applications II," Elsevier, Amsterdam, 1992.

[18] E. Gelenbe, "Learning in the recurrent random neural network," *Neural Computation,* Vol. 5, No. 1, pp. 154-164, 1993.

[19] E. Gelenbe, Y. Feng and K. R. R. Krishnan, "Neural network methods for volumetric magnetic resonance imaging of the human brain," *Proceedings IEEE*, Vol. 84, No. 10, pp. 1488–1496, Oct. 1996.

[20] E. Gelenbe, C. Cramer, M. Sungur, P. Gelenbe, "Traffic and video quality in adaptive neural compression," *Multimedia Systems*, Vol. 4, pp. 357-369, 1996.

[21] E. Gelenbe, J.M. Fourneau, "Random neural networks with multiple classes of signals," *Neural Computation*, Vol. 11, pp. 721-731, 1999.

[22] E. Gelenbe, V. Koubi, F. Pekergin "Dynamical random neural network approach to the Traveling Salesman Problem", *IEEE Symposium on Systems, Man and Cybernetics* 2, pp. 630–635, 1992.

[23] E. Gelenbe, Z.-H. Mao, Y. Da-Li, "Function approximationRandom neural networks with multiple classes of signals," *IEEE Trans. On Neural Networks*, 1999.

[24] E. Gelenbe, K. F. Hussain, H. E. Abdelbaki,"Random neural network texture model," *Proceedings of SPIE*, Jan 2000.

[25] A. Ghanwani "A qualitative comparison of neural network models applied to the vertex covering problem", *ELEKTRIK*, Vol. 2, No. 1, pp. 11–18, 1994.

[26] A. Ghanwani "Neural and delay based heuristics for the Steiner problem in networks", *European Journal of Operations Research*, Vol. 108, pp. 241–265, 1998.

[27] R. M. Haralik, K. Shanmugam, and I. Dinstein, "Textural features for image classification," *IEEE Trans. Syst. Man Cybern.,* vol. 3, 1971, pp. 610-621.

[28] *Computer and Robot Vision*, Addison-Wesley, 1992.

[29] R. M. Haralick and R. Bosley, "Texture features for image classification," *Third ERTS Symposium*, NASA SP-351, pp. 1219-1228, 1973.

[30] B. Julesz, H.L. Frisch, E.N. Gilbert, and L.A. Shepp, "Inability of humans to discrminate between visual textures that agree in second-order statistics – revisited", *Perception* Vol. 2, pp. 391–405, 1973.

[31] B. Julesz, "Experiments in the visual perception of texture", *Scientific American*, Vol. 232, pp. 34–43, 1975.

[32] E.C. Kandel, J.H. Schwartz, "Principles of Neural Science," Elsevier, Amsterdam, 1985.

[33] J.G. Kemeny, J.L. Snell, "Finite Markov Chains," Van Nostrand, Princeton, 1965.

[34] Y. Le Cun, " A learning procedure for asymmetric threshold networks," *Proc. Cognitiva 85*, pp 599-604, 1985.

[35] F.J. Pineda, "Generalization of backpropagation to recurrent and higher order neural networks," in Anderson, D.Z. (ed.), *Neural Information Processing Systems*, American Inst. Phys., p 602, 1987.

[36] F.J. Pineda, "Recurrent backpropagation and the dynamical approach to adaptive neural computation," *Neural Computation*, Vol. 1, No. 2, pp 161-172, 1989.

[37] B.A. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Computation*, Vol. 1, No. 2, pp 263-269, 1989.

[38] D.E. Rumelhart, G.E. Hinton, R.J. Williams, "Learning internal representations by error propagation," in D.E. Rumelhart, J.L. McClelland, and the PDP Research Group " Parallel distributed processing" Vols. I and II, Bradford Books and MIT Press, Cambridge, Mass.(1986).

[39] H. Voorhees and T. Poggio, "Detecting textons and texture boundaries in natural images," in *Proc. first Int. Conf. on Computer Vision,* London, 1987, pp. 250-258.

[40] N. Rao and V. Vemuri, "A neural network architecture for texture segmentation and labeling," *Proceesings of the International Joint Conference on Neural Networks,* Washington, DC, vol. I, pp. 127-133.

[41] A. Visa, "A texture classifier based on neural network principles," *Proceesings of the International Joint Conference on Neural Networks,* San Diego, vol. I, pp. 491-496.

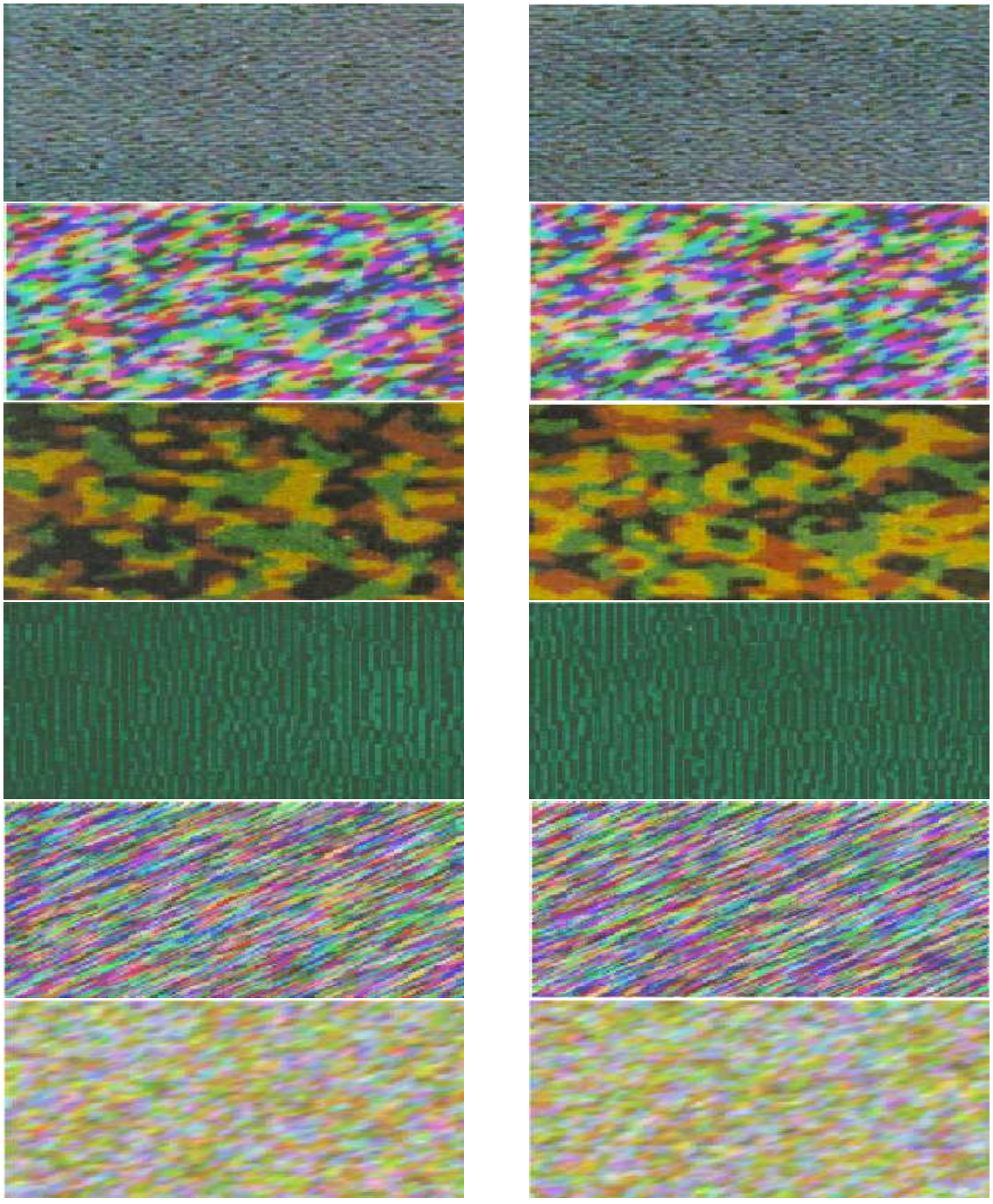

Figure 3: Comparison of synthetic color textures generated with prespecified parameters by the MCRNN (left), and learned by the MCRNN from the left-hand images and then generated by the MCRNN with the learned parameters (right)

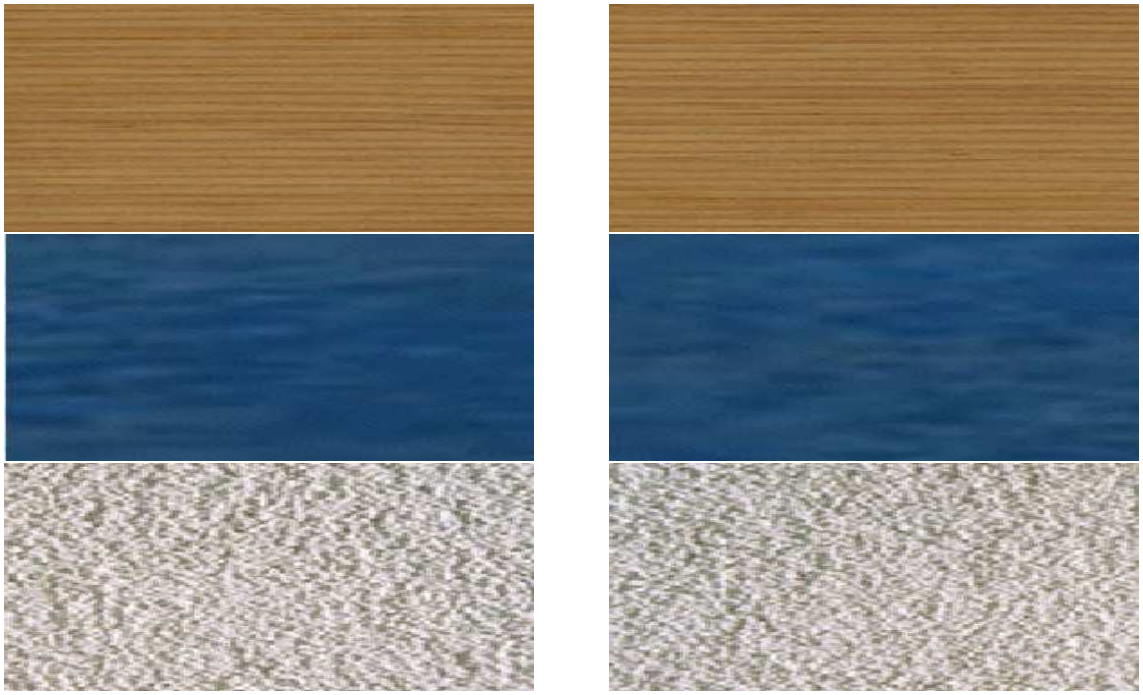

Figure 4: Natural color textures (Left), and corresponding synthetic textures (Right) obtained by learning the parameters and then generating the textures with the MCRNN