11 Receptive Fields[1]

F. Morán, J. C. Chacón, M. A. Andrade and A. Weitzenfeld

11.1 Introduction

The visual nervous system in higher mammals shows a high degree of organization in which different selectivity properties are found. However, the necessary quantity of information for specifying that connectivity is much higher than the information stored in the genetic code (von der Malsburg 1987). Some organizing processes have been found which could explain this fact.

In an early stage of the development of the mammal embryo, the nervous fibers coming from the ganglion cells grow from retina to brain establishing connections into the visual cortex. Once a primary gross connection is reached, a self-organizing process, dependent on neural activity, takes place and the connections are pruned, which gives functional characteristics to the visual system (Linsker 1990; Singer 1987; Stryker 1986; von der Malsburg and Singer 1988). This mechanism permits several superposed mappings to appear in the visual cortex (Fregnac and Imbert 1984; Orban 1984; Tootell et al 1981).

The receptive field is a characteristic organization of the visual system (Hubel and Wiesel 1963; Orban 1984). A receptive field of a neuron is the compact region of the visual space that affects the activity of that neuron. A well-known example is the on-off or Mexican-hat shaped receptive field, with circular symmetry. If the center area is stimulated, an activatory response is produced in the target neuron, whereas the stimulation in the neighborhood produces inhibition.

Based on neurophysiological knowledge, some models that explain visual cortex organization during development have been proposed (Erwin et al 1995). Most of them use neural network architectures and activity dependent rules (Linsker 1986; Miller et al 1989; von der Malsburg 1990).

The self-organizing network presented in this chapter shows how *diffusion of synaptic activity*, *competitive synaptic growth* and *synaptic evolution* can explain the development of variable sized on-off receptive fields in a developmental stage of a mammal embryo prior to visual experience. Synaptic activity is driven by either *activity correlation* or *activity anti-correlation* (i.e., Hebbian (Hebb 1949) or anti-Hebbian (Carlson 1990; Földiak 1990) learning rules, respectively).

11.2 Model Description

The model presented in this chapter is based on the classical work of von der Malsburg (Häussler and von der Malsburg 1983), that has been previously proposed elsewhere (Andrade and Moran 1996). The architecture of the network is schematically represented in figure 11.1.

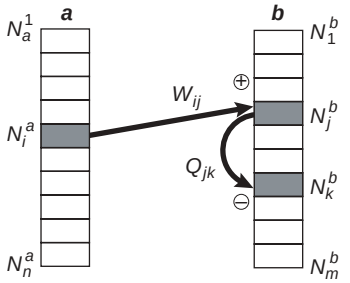

Figure 11.1
The model consists of two neuron layers: (1) an input layer *a* (corresponding to LGN - Lateral Geniculate Nucleus) and (2) an output layer *b* (corresponding to Primary Visual Cortex). These two layers are fully interconnected by excitatory connections, $W_{ij} > 0$ ($i = 1,...,n$; $j = 1,...,m$), while the output layer is fully interconnected by lateral inhibitory connections, $Q_{jk} > 0$, ($j,k = 1,...,m$).

The evolution of the system connectivity starts with an initial random state that will converge into a final state by processing the following differential equations:

$$\frac{dW_{ij}(t)}{dt} = \alpha + \beta W_{ij}(t)(F_{ij}^{a}(t) - \gamma W_{ij}^{2}(t))$$
$$\frac{dQ_{jk}(t)}{dt} = \alpha + \beta Q_{jk}(t)(F_{jk}^{b}(t) - \gamma Q_{jk}^{2}(t)) \quad (11.1)$$

where α is a positive constant that accounts for the generation of new synaptic connections, and parameter β accounts for the rate of change of the established connections.

The terms, $W_{ij}(t)F_{ij}^{a}(t)$ and $Q_{jk}(t)F_{jk}^{b}(t)$, represent a *growth factor* whose value depends on the temporal correlation of the signals connecting the neurons. The growth factor describes the increase (or decrease) of a particular synapse depending on the *global* state of all the network synapses. F_{ij}^{a} and F_{jk}^{b} are the growth factors of synaptic weights *W* and *Q*, respectively. Growth factor F_{ij}^{a} uses a Hebbian rule, since it increases the value of an excitatory connection when correlation grows, as described in previous models for the development of retinotopic connectivity (Häussler and von der Malsburg 1983) and ocular domains (Andrade and Moran 1996). However, growth factor F_{jk}^{b} one uses an anti-Hebbian rule (Carlson 1990; Földiak 1990), since the inhibitory connection is increased.

The decaying terms $W_{ij}(t)\gamma W^{2}_{ij}(t)$ and $Q_{jk}(t)\gamma Q^{2}_{jk}(t)$ are cubic weight terms multiplied by a constant γ controlling their respective contribution. These terms account for the *individual* growth restrictions for each synaptic connection.

The growth factors are a function of the neurons activity correlation (Andrade and Moran 1997; Häussler and von der Malsburg 1983):

$$F_{ij}^{a}(t) \propto < A_{i}^{a}(t), A_{j}^{b}(t) > t$$
$$F_{jk}^{b}(t) \propto < A_{j}^{b}(t), A_{k}^{b}(t) > t \quad (11.2)$$

where $A_{i}^{a}(t)$ represents the activity of the input layer neurons N_{i}^{a} (for i = 1,...,n), and $A_{j}^{b}(t)$ represents the activity of the output layer neurons N_{j}^{b} (for j = 1,...,m).

Initially, the only source of activity is spontaneous activity from the photoreceptors (layer *a* in this model), since the visual system does not receive any coherent visual signal from the environment. Therefore, the spontaneous and non-correlated activity $f_i(t)$ will be the only source of activity in layer *a*. Moreover, due to the lateral propagation of activity in that layer, the resulting neuron output will depend on its own activity as well as on its neighboring neurons, modulated by a cushioning diffusion term,

$$D_{ij}^{a} = G^{a}(|i - j|) \quad (11.3)$$

where G^a is a function of the distance between the neurons assumed to be Gaussian:

$$G^{a}(x) = \frac{h_a}{s_a}\exp\left(-(x/s_a)^2/2\right)$$
$$G^{b}(x) = \frac{h_b}{s_b}\exp\left(-(x/s_b)^2/2\right) \quad (11.4)$$

where s_a and s_b are positive constants describing the Gaussian width, and h_a and h_b indicate the surface under the curve, with $x = |i - j|$.

In other words, the output activity of the neuron is given by:

$$A_{i}^{a}(t) = \sum_{k=1}^{n} f_k(t) D_{ki}^{a} \quad (11.5)$$

This activity is transmitted to the output layer by means of the excitatory connections W_{ij}. In output layer b, the signal received by each neuron is modified by two different effects: lateral excitatory diffusion of the signal and lateral inhibitory transmission of the signal, Q_{jk}. The two combined effects lead to the following expression describing activity in the output neurons:

$$A_j^b(t) = \sum_{q=1}^{n}\sum_{p=1}^{n}\sum_{o=1}^{m} f_q^a(t) D_{qp}^a W_{po}(t)\left(D_{kj}^b - \sum_{l=1}^{m} D_{ol}^b Q_{lj}(t)\right) \tag{11.6}$$

Since the input layer activity is spontaneous, there is no correlation between the activity of two neurons, that is to say:

$$< f_i(t), f_j(t) > t = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases} \tag{11.7}$$

Or what amounts to:

$$< f_i(t), f_j(t) > t = \delta_{ij} \tag{11.8}$$

where δ is Kronecker's delta. Taking this into account, we can rewrite expression (11.1) as follows (for a more detailed description see (Andrade and Moran 1997)):

$$\begin{aligned} \frac{dW_{ij}(t)}{dt} &= \alpha + \beta W_{ij}(t)\left[\sum_{q=1}^{n} D_{qi}^a E_{qj}(t) - \gamma W_{ij}^2(t)\right] \\ \frac{dQ_{jk}(t)}{dt} &= \alpha + \beta Q_{jk}(t)\left[\sum_{q=1}^{n} E_{qj}(t) E_{qk}(t) - \gamma Q_{ij}^2(t)\right] \end{aligned} \tag{11.9}$$

where

$$E_{ij}(t) = \sum_{k=1}^{n} D_{ik}^a \sum_{p=1}^{m} W_{kp}(t)\left(D_{pj}^b - \sum_{l=1}^{m} D_{pl}^b Q_{lj}(t)\right) \tag{11.10}$$

It should be noted that a side effect of these correlation functions is that all spontaneous activity has been *explicitly* eliminated from the two equations. Among other considerations, this results in a much easier numeric integration.

11.3 Model Architecture

The model is implemented by a **Recfield** instantiated by the **RecfieldModel** at the top level. The **Recfield** module contains four modules, **LayerA**, **LayerB**, **ConnectW** and **ConnectQ** as shown in figure 11.2.

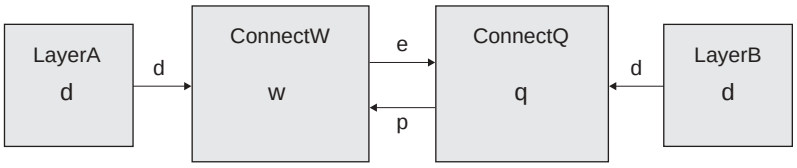

Figure 11.2
The Recfield module is composed of LayerA module sending data *d* to the excitatory connection module ConnectW and LayerB module send data *d* to inhibitory connection module ConnectQ. Module *ConnectW* send data *e* to *ConnectQ*. *ConnectQ* module sends data *p* to *ConnectW*.

The **Recfield** module instantiates its four submodules as follows (note the different parameters passed to each),

```
nslModule Recfield (int x1, int y1, int x2, int y2)
{
    private LayerA a(x1, y1);
    private LayerB b(x2, y2);

    private ConnectW w(x1, y1, x2, y2);
    private ConnectQ q(x1, y1, x2, y2);
}
```

In our model all parameters passed to **Recfield**, *x1*, *y1*, *x2* and *y2*, have a size of 5.

LayerA Module

LayerA module defines the diffused activation *da* with the help of a gaussian distribution function,

```
nslModule LayerA (int x1, int y1)
{
    public NslDoutFloat4 d(x1, y1, x1, y1);  // (da), to w

    private NslFloat0 s(); // gaussian spread (sa)
    private NslFloat0 h(); // gaussian height (ha)
}
```

The **initRun** method computes *d* value. Note that a user defined external function is applied,

```
public void initRun()
{
    nslGaussian(d,h,s);
}
```

The gaussian function is defined as a library (it will also be used by **LayerB**).

```
private void nslGaussian(NslFloat4 g, NslFloat0 h, NslFloat0 s)
{
    int      i, j, k, l;      // loops
    float    dist,dx,dy;
   int x1 = g.getRows();
   int y1 = g.getCols();
    for (i = 0; i < x1; i++)
      for (j = 0; j < y1; j++)
        for (k = 0; k < x1; k++)
          for (l = 0; l < y1; l++) {
    dx = nslAbs(i - k);
    if (dx > (x1 / 2))
    dx = x1 - dx;
    dy = nslAbs(j - l);
    if (dy > (y1 / 2))
    dy = y1 - dy;
    dist = nslSqrt (dx*dx + dy*dy);
    g[i][j][k][l] = (h/s)*nslExp(-nslPow((dist/s),2)/2) ;
          }
}
```

ConnectW Module

The excitatory connection module **ConnectW** is defined as follows

```
nslModule ConnectW (int x1, int y1, int x2, int y2) {

    public NslDinFloat4 d(x1, y1, x1, y1);  // from a
    public NslDinFloat4 p(x2, y2, x2, y2);  // from q
    public NslDoutFloat4 e(x1, x2, y1, y2); // to q

    private NslFloat4 w(x1, x2, y1, y2);

    private NslFloat0 maxinitval(); // max weight val
    private NslFloat0 seed(); // random seed

    private NslFloat0 alpha();// get weights out from zero
    private NslFloat0 beta();// integration parameter for act
    private NslFloat0 gamma();// cubic decay term for w
}
```

Parameters *x1*, *y1*, *x2* and *y2* are assigned to local attributes so they can later be used by local methods cycling on every array element. Weights are initialized by a random function

```
public void initWeights(float randa)
{
    int   i, j, k, l;         // loops

    for (i = 0; i < _x1; i++)
      for (j = 0; j < _x2; j++)
        for (k = 0; k < _y1; k++)
          for (l = 0; l < _y2; l++)
           w[j][i][l][k] = maxinitval*randa;
}
```

Function *nslNormRand*() is shown as follows

```
private int nslNormRand(NslFloat0 seed)
{
    // calculation of the random maximum value
    int j, max_rand = 0;
    for (int i = 0; i < 1000; i++) { // max number of iterations ?
      j = nslRand();
      if (j > max_rand)
        max_rand = j;
    }
    // random seed
    nslRand(seed);

    return nslRand()/max_rand; // normalization to 1
}
```

The **initRun** method simply initializes the weights by a normalized random function

```
public void initRun()
{
    int randa = nslNormRand(seed);
    initWeights(randa);
}
```

Function *convGauss*() is shown as follows

```
private void convGauss()
{
   int             i, j, k, l, m, n, o, p;       /* loops */
   float           sum, sum2;

   /* convolutions gaussian 1 w pp (from L2) */
   for (i = 0; i < x1; i++)
     for (j = 0; j < y1; j++)
       for (k = 0; k < x2; k++)
         for (l = 0; l < y2; l++) {
          sum = 0;
       for (m = 0; m < x1; m++)
          for (n = 0; n < y1; n++) {
       sum2 = 0;
       for (o = 0; o < x2; o++)
       for (p = 0; p < y2; p++)
       sum2=sum2+w[o][m][p][n]*pp[o][p][k][l];
          sum = sum + d[i][j][m][n] * sum2;
          }
          e[k][i][l][j] = sum;
         }
}
```

Function *modifyWeights*() is shown as follows

```
private void modifyWeights()
{
   int             i, j, k, l, m, n, o, p;       // loops
   float           sum, sum2;

   // weight modification

   for (i = 0; i < x1; i++)
     for (j = 0; j < y1; j++)
       for (k = 0; k < x2; k++)
         for (l = 0; l < y2; l++) {
          sum = 0;
       for (m = 0; m < x1; m++)
          for (n = 0; n < y1; n++)
       sum = sum + d[m][n][i][j] * e[k][m][l][n];
          w[k][i][l][j] = alpha + w[k][i][l][j] *
       (1 + beta*(sum - gamma * w[k][i][l][j]
       * w[k][i][l][j]));
       if (w[k][i][l][j] < 0)
          nslPrint("no");
         }
}
```

The **simRun** method processes the differential equation defining the weight activity

```
public void simRun()
{
   // convGauss()
   e = d * (w * p);
   // modifyWeights()
   nslDiff(w,1.0,alpha + beta*w(d*e - gamma*(w^w)); // eq
      (11.9)
}
```

LayerB Module

LayerB module defines only diffused activation *db* with the help of a gaussian distribution function,

```
nslModule LayerB (int x2, int y2)
{
   public NslDoutFloat4 d(x2, y2, x2, y2); // (db) to Q
   private NslFloat0 s();  // gaussian spread (sb)
   private NslFloat0 h();    // gaussian height (hb)
}
```

The **initRun** method computes *d* value. Note that a user defined external function is applied,

```
public void initRun()
{
   nslGaussian(d,delta,sp);
}
```

ConnectQ Module

The inhibitory connection module **ConnectQ** is defined as follows

```
nslModule LayerB (int x1, int y1, int x2, int y2)
{
   public NslDinFloat4 e(x1, x2, y1, y2);  // from w
   public NslDinFloat4 d(x2, y2, x2, y2);  // from b (db)
   public NslDoutFloat4 p(x2, y2, x2, y2); // to w

   private NslFloat4 q(x1, y1, x2, y2);//x2*y2,x2*y2 inhib(q)
   private NslFloat0 alpha(); // get weights out from zero
   private NslFloat0 beta();// integration parameter for act
   private NslFloat0 gamma();  // cubic decay term for w
}
```

The **initRun** method simply initializes the weights to zero

```
public void initRun()
{
   q = 0;
}
```

Function *convGauss*() is shown as follows

```
private void convGauss()
{
   int             i, j, k, l, m, n, o, p;       // loops
   float           sum, sum2;

   // convolutions q gaussian 2

   for (i = 0; i < x2; i++)
     for (j = 0; j < y2; j++)
       for (k = 0; k < x2; k++)
         for (l = 0; l < y2; l++) {
       sum = 0;
       for (m = 0; m < x2; m++)
          for (n = 0; n < y2; n++)
   sum = sum + d[i][j][m][n] * q[m][n][k][l];
       p[i][j][k][l] = d[i][j][k][l] - sum;
         }
}
```

Function *modifyWeights*() is shown as follows

```
private void modifyWeights()
{
   int             i, j, k, l, m, n, o, p;       // loops
   float           sum, sum2;

   // weight modification

   for (i = 0; i < x2; i++)
     for (j = 0; j < y2; j++)
       for (k = 0; k < x2; k++)
         for (l = 0; l < y2; l++) {
       sum = 0;
       for (m = 0; m < x1; m++)
          for (n = 0; n < y1; n++)
       sum = sum + e[i][m][j][n] * e[k][m][l][n];
       w[i][j][k][l] = alpha + w[i][j][k][l] * (1 + beta *
          (sum - gamma*q[i][j][k][l] * q[i][j][k][l]));
       if (q[i][j][k][l] < 0)
          nslPrint("no");
         }
}
```

The **simRun** method processes the differential equation defining the weight activity

```
public void simRun()
{
   // convGauss()
   p = d - (d * q);
   // modifyWeights()
   nslDiff(q,1.0,alpha + beta*q(e*e - ga*(q^q)); // eq (11.9)
}
```

11.4 Simulation and Results[1]

The simulation control file contains parameter value assignment. Note how we can assign common values to different module parameters (*alpha*, *beta* and *gamma*)

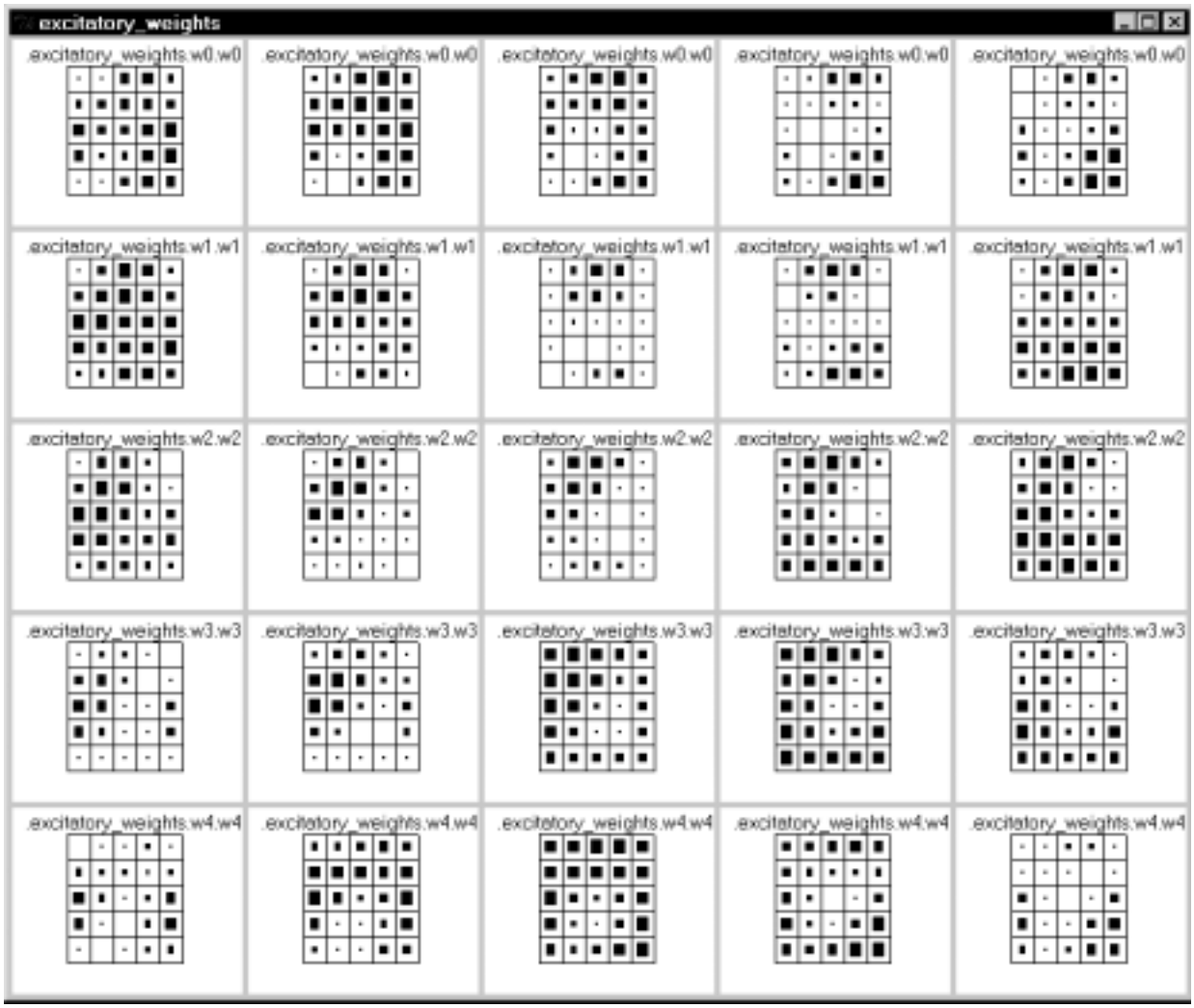

Figure 11.3
Excitatory connection weight *w* in ConnectW module.

```
nsl set system.simDelta 0.1
nsl set system.simEndTime 50

set alpha 0.00005
set beta 0.001
set gamma 1e4

nsl set recfield.w.alpha $alpha
nsl set recfield.w.beta $beta
nsl set recfield.w.gamma $gamma

nsl set recfield.q.alpha $alpha
nsl set recfield.q.beta $beta
nsl set recfield.q.gamma $gamma

nsl set recfield.a.seed 77
nsl set recfield.a.maxinitval 0.001
nsl set recfield.a.sp 1
nsl set recfield.a.delta 4

nsl set recfield.b.sp 1
nsl set recfield.b.delta 4
```

To simulate the model load “recfield.nsl” and then run it. Three display frames are created containing a display canvas each, for *w*, *q* and *e* respectively. The connection matrices *w* and *q* and the resulting matrix *e* are shown in the figures 11.3 to 11.5. Matrix *e* represents the excitatory/inhibition effect produced on each output layer neuron when the input layer neuron is activated and the signal is transmitted through the network. Therefore, it represents the set of receptive fields corresponding to the cortex neurons.

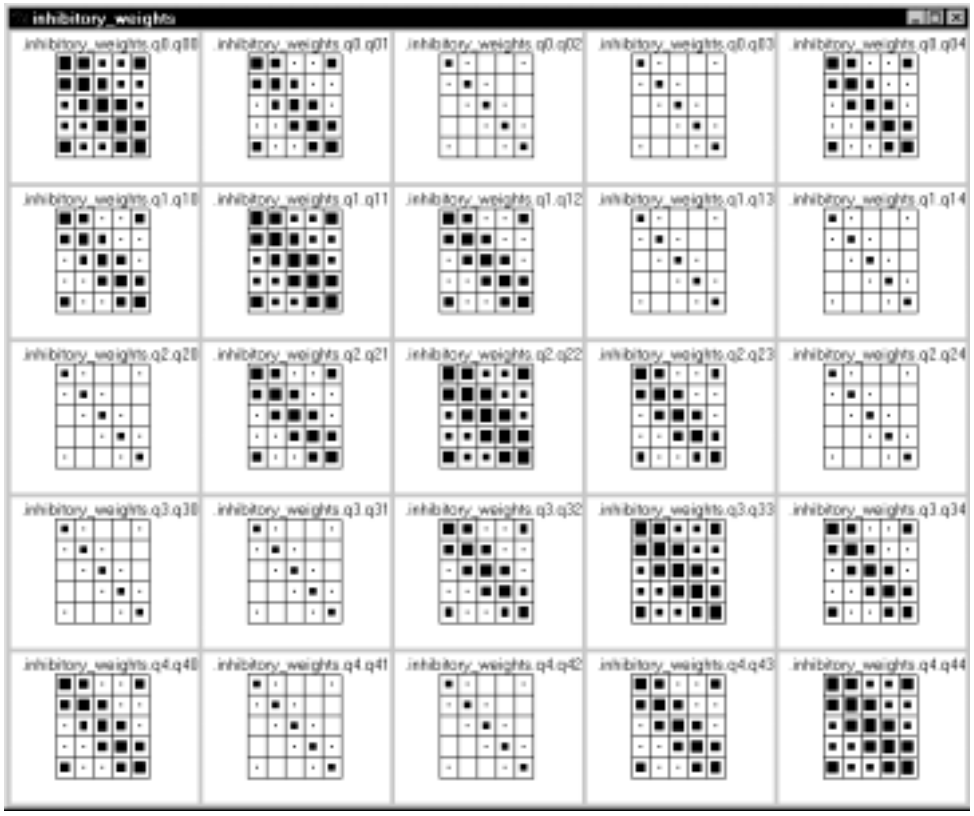

Figure 11.4
Inhibitory connection weight *q* in ConnectQ module.

In figure 11.3, the variations in geometry of the receptive fields can be noted, while in figure 11.4 the inhibitory weights show a more homogeneous shape, that is, each neuron is connected to its neighbors in a circular manner. These differences are more relevant if larger number of neurons are used (i.e., a layer of 8x8 neurons presents neurons with highly different oriented receptive fields)

The result of the joint action of the two weight matrices and the intra-layer lateral diffusion of signal become apparent in the receptive field values showed in figure 11.5. Thus, different receptive fields consist of a compact activation area placed in different positions, and the form of this area is also variable, showing oriented symmetries and different orientations. The size of the receptive fields is variable, too. When observed in detail, there is a spatial continuity between the receptive fields. So, closer neurons tend to coincide in the situation of its positive-area and to have similar geometry, either in orientations or sizes.

11.5 Summary

Through the simulation of this model, the essential characteristics of self-organization have been demonstrated. The kind of resulting connectivity let us to explain how the nervous system in general, and the visual system in particular, can obtain their specific connectivity through self-organizing process based in the system activity and a reduced number of local rules, easily justifiable from a physiological point of view.

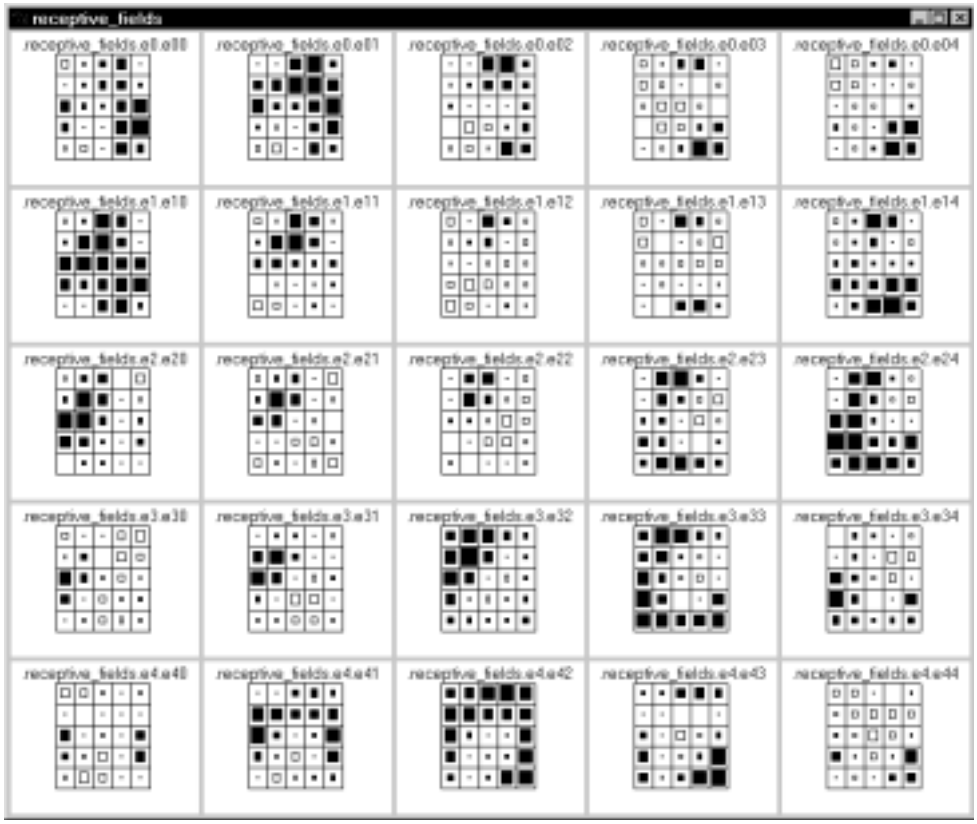

Figure 11.5 Receptive field *e* in ConnectW module.

Notes

1. This work has been supported in part by grants from the DGICYT, MEC, Spain (project no. PB92-0456) and from the CICYT, Spain. (project no. BIO96-0895). This work is part of the Doctoral Thesis of M.A.A. developed under a fellowship from the MEC, Spain. We thank Prof. Ch. von der Malsburg for his support and suggestions.
2. The Receptive Fields model was implemented and tested under NSLC.