18 Face Recognition by Dynamic Link Matching[1]

L. Wiskott, C. von der Malsburg and A. Weitzenfeld[2]

We present here a biologically motivated system for invariant and robust recognition of objects from camera images. It originally arose from a homework assignment for a course of neural network self-organization at USC, and in a way it can be seen as a serious test of NSL's maturity as a (neural) simulation tool. Formulated as a large system of coupled non-linear differential equations comprising altogether approximately 3 million variables, its development required extensive series of experiments and continuous graphical monitoring of large sets of variables. Not only did NSL support this process, requiring just minor extensions, but it now makes our system directly accessible to students and colleagues for close inspection and for further development.

Our model is based on the principles of temporal feature binding and dynamic link matching. Objects are stored in the form of two-dimensional aspects. These are competitively matched against current images. During the matching process, complete matrices of dynamic links between the image and all models are refined by a process of rapid self-organization, the final state connecting only corresponding points in image and object models. As a data format for representing images we use local sets ("jets") of Gabor-based wavelets. We have tested the performance of our system by having it recognize human faces against databases of more than one hundred images. The system is invariant with respect to retinal position, and it is robust with respect to head rotation, scale, facial deformation, and illumination.

18.1 Introduction

For the theoretical biologist, the greatest challenge posed by the brain is its tremendous power to generalize from one situation to others. This ability is probably most concretely epitomized in terms of invariant object recognition—the capability of the visual system to pick up the image of an object and recognize that object later in spite of variations in retinal location (as well as other important changes such as size, orientation, changed perspective and background, deformation, illumination, and noise). This capability has been demonstrated by flashing the image of novel objects briefly at one foveal position, upon which subjects were able to recognize the objects in a different foveal position (and under rotation in depth) (B & Gerhardstein 1993).

The conceptual grandfather of many of the neural models of invariant object recognition is Rosenblatt's four-layer perceptron (Rosenblatt 1961). Its first layer is the sensory or retinal surface. Its second layer contains detectors of local features (that is, small patterns) in the input layers. Each one of these is characterized by a feature type and a position x. The third layer contains position-invariant feature detectors, each of which is characterized by a feature type and is to respond to the appearance of its feature type anywhere on the input layer. It is enabled to do so by a full set of connections from all of the cells of the same feature type in the second layer. Thus, the appearance of a pattern in any position of the input layer leads to the activation of the same set of cells in the third layer. Layer four now contains linear decision units which detect the appearance of certain sets of active cells in the third layer and thus of certain objects imaged into the input layer. A decision unit contains an implicit model of an object in the form of a weighted list of third-layer features to be present or absent.

The four-layer perceptron has to contend with the difficulty that a set of feature types has to be found on the basis of which the presence or absence of a given pattern becomes linearly separable on the basis of the un-ordered feature lists displayed by the third layer.

If the feature types employed are too indistinct, there is the danger that different patterns lead to identical third-layer activity, just because the only difference between the patterns is a different spatial arrangement of their features. The danger can be reduced or avoided with the help of feature types of sufficient complexity. However, this is a problematic route itself, since highly complex features are either very numerous (and therefore costly to install) or they are very specific to a given pattern domain (and have to be laboriously trained or hand-designed into the system and limit the system's applicability to the pattern domain). The difficulty arises from the fact that on the way from layer two to layer three position information is discarded for each feature individually (as is required by the condition of position invariance), so that also information on relative position of the features is lost (which creates the potential confusion).

In the study presented here we are solving the indicated problem using a double strategy. Firstly, we employ highly complex features which are constructed during presentation of individual patterns (and which are stored individually for each pattern later to be recognized), and secondly, we employ a data format and a pattern matching procedure (between our equivalent of Rosenblatt's layers two and three) which represent and preserve relative position information for features.

The features we employ are constructed from image data in a two-step process. First, elementary features in the form of Gabor-based wavelets of a number of scales and a number of orientations are extracted from the image (Daugman 1988), giving a set of response values for each point of the image, then the vector of those response values for a given point are treated as a complex feature, which we call a jet. Jets are extracted from an array of sample points in the image (the approach is described in detail in (Lades et al. 1993)).

Our system is explicit in its representation of analogs for layers two and three, which we call "image domain" and "model domain", respectively. The image domain is an array of (16x17) nodes, each node being labeled by a jet when an image is presented. The model domain is actually a composite of a large number (more than one hundred in some of our simulations) of layers ("models") composed of arrays of (10x10) nodes. To store the image of an object (e.g., a human face) a new model is created in the model domain and its nodes are labeled by copying an array of jets from the appropriate part of the image domain.

To recognize an object, the system attempts to competitively match all stored object models against the jet array in the image domain, a process which we call "Dynamic Link Matching." The winning model is identified as the object recognized. The two domains are coupled by a full matrix of connections between nodes, which is initialized with similarity values between image jets and model jets. (This can be seen as our version of Rosenblatt's feature-preserving connections.) The matching process is formulated in terms of dynamical activity variables for the image and model layers (forming localized blobs of activity in both domains), for the momentary strengths of connections between the domains (we assume that synaptic weights change rapidly and reversibly during the recognition process), and for the relative recognition status of each model. The matching process enforces the condition that neighboring nodes in the image layer link up with neighboring nodes in a model layer. In this way the system suppresses the feature rearrangement ambiguity of the Rosenblatt scheme.

Our model cannot be implemented (at least not in any obvious way) in conventional neural networks. Its implementation is, however, easily possible if two particular features are assumed to be realized in the nervous system, temporal feature binding and rapid reversible synaptic plasticity. Both features have been proposed as fundamental components of neural architecture in (von der Malsburg 1981). Temporal feature binding has in the mean time been widely discussed in the neuroscience literature and has received some experimental basis (König and Engel 1995). Although rapid synaptic weight changes have been discussed (Crick 1982) and reported in the literature (Zucker 1989), the quasi-Hebbian control and the time course for rapid reversible plasticity which is implied and required here must still wait for experimental validation.

18.2 Model Description

Principle of Dynamic Link Matching

In Dynamic Link Matching (DLM), the image and all models are represented by layers of neurons, which are labeled by jets as local features (see figure 18.1). Jets are vectors of Gabor wavelet components (see Lades et al. 1993; Wiskott et al. 1997) and a robust description of the local gray value distribution. The initial connectivity is all-to-all with synaptic weights depending on the similarities between the jets. In each layer, neural activity dynamics generates one small moving blob of activity (the blob can be interpreted as covert attention scanning the image or model). If a model is similar in feature distribution to the image, its initial connectivity matrix contains a strong regular component, connecting corresponding points (which by definition have high feature similarity), plus noise in the form of accidental similarities. Hence the blobs in the image and that model tend to align and synchronize in the sense of simultaneously activating, and thus generating correlations, between corresponding regions. These correlations are used, in a process of rapid reversible synaptic plasticity, to restructure the connectivity matrix. The mapping implicit in the signal correlations is more regularly structured than the connectivity itself, and correlation-controlled plasticity thus improves the connectivity matrix. Iteration of this game rapidly leads to a neighborhood preserving one-to-one mapping connecting neurons with similar features, thus providing translation invariance as well as robustness against distortions.

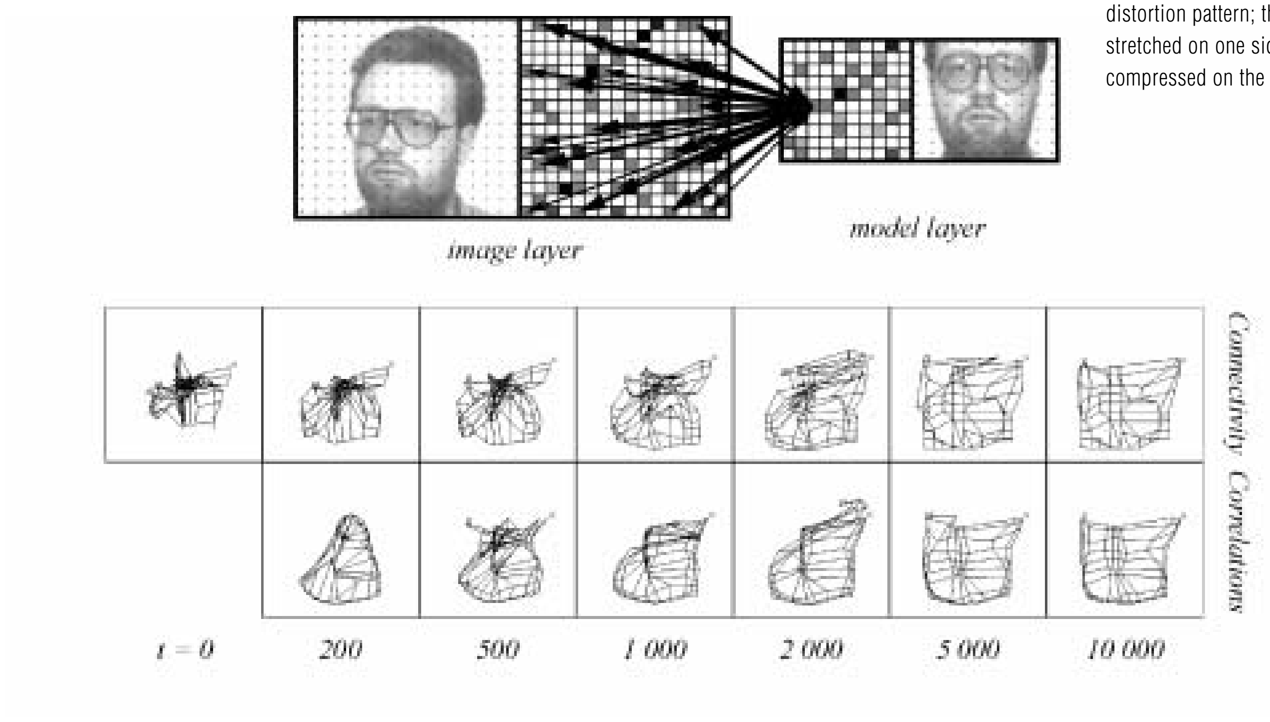


Figure 18.1
DLM between image and model. The nodes are indicated by black dots, and their local features are symbolized by different textures. The synaptic weights of the initial all-to-all connectivity are indicated by arrows of different line widths. The net displays below show how correlations and connectivity co-develop in time. The image layer serves as a canvas on which the model layer is drawn as a net. Each node corresponds to a model neuron, neighboring neurons are connected by an edge. The nodes are located at the centers of gravity of the projective field of the model neurons, considering synaptic weights as physical mass. In order to favor strong links, the masses are taken to the power of three. The correlations are displayed in the same way, using averaged correlations instead of synaptic weights. It can be seen that the correlations develop faster and are cleaner than the connectivity. The rotation in depth causes a typical distortion pattern; the mapping is stretched on one side and compressed on the other.

For recognition purposes, DLM has to be applied in parallel to many models. The best fitting model, i.e. the model most similar to the image, will finally have the strongest connections to the image and will have attracted the greatest share of blob activity. A simple integrating winner-take-all mechanism detects the correct model (see figure 18.2).

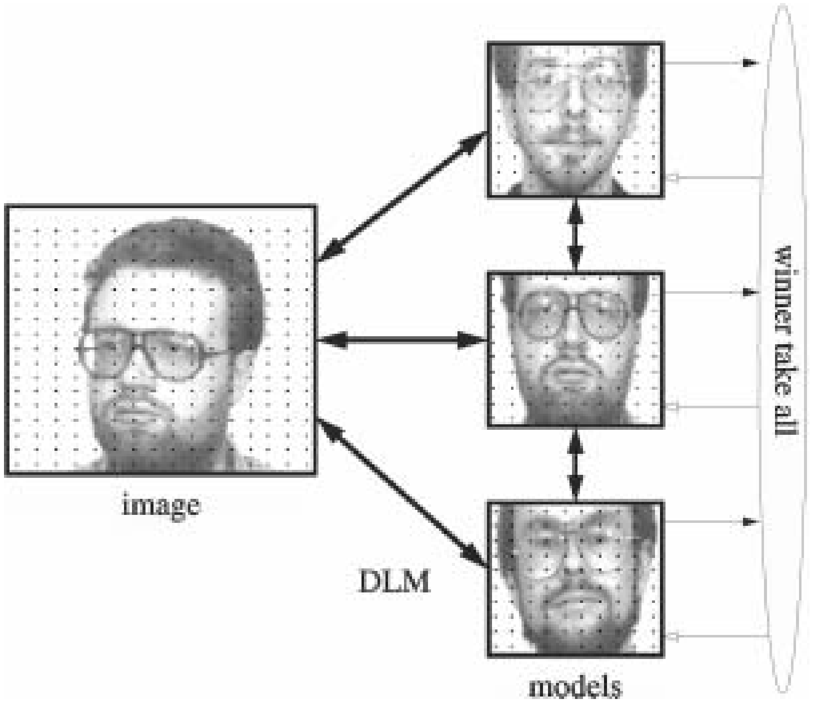


Figure 18.2
Architecture of the DLM face recognition system. Image and models are represented as neural layers of local features, as indicated by the black dots. DLM establishes a regular one-to-one mapping between the initially all-to-all connected layers, connecting corresponding neurons. Thus, DLM provides translation invariance and robustness against distortion. Once the correct mappings are found, a simple winner-take-all mechanism can detect the model that is most active and most similar to the image.

The equations of the system are given in table 18.1; the respective symbols are listed in table 18.2. In the following sections we will explain the system step by step: blob formation, blob mobilization, interaction between two layers, link dynamics, attention dynamics, and recognition dynamics.

Layer dynamics:

$$h_i^p(t_0) = 0$$

$$\dot{h}_i^p = -h_i^p + \sum_{i'} g_{i-i'} \max_{p'}\left(\sigma\left(h_{i'}^{p'}\right)\right) - \beta_h \sum_{i'} \sigma\left(h_{i'}^p\right) - \kappa_{hs} s_i^p \tag{18.1}$$

$$+ \kappa_{hh} \max_{qj}\left(W_{ij}^{pq} \sigma\left(h_j^q\right)\right) + \kappa_{ha}\left(\sigma\left(a_i^p\right) - \beta_{ac}\right) - \beta_\theta \Theta\left(r_\theta - r^p\right)$$

$$s_i^p(t_0) = 0$$

$$\dot{s}_i^p = \lambda_\pm\left(h_i^p - s_i^p\right) \tag{18.2}$$

$$g_{i-i'} = \exp\left(-\frac{(i-i')^2}{2\sigma_g^2}\right) \tag{18.3}$$

$$\sigma(h) = \begin{cases} 0 & h \le 0 \\ \sqrt{h/\rho} & 0 < h < \rho \\ 1 & h \ge \rho \end{cases} \tag{18.4}$$

Attention dynamics:

$$a_i^p(t_0) = \alpha_N$$

$$\dot{a}_i^p = \lambda_a\left(-a_i^p + \sum_{i'} g_{i-i'} \sigma\left(a_{i'}^p\right) - \beta_a \sum_{i'} \sigma\left(a_{i'}^p\right) + \kappa_{ah} \max_{p'}\left(\sigma\left(h_i^{p'}\right)\right)\right) \tag{18.5}$$

Link dynamics:

$$W_{ij}^{pq}(t_0) = S_{ij}^{pq} = \max\left(S_\phi\left(J_i^p, J_j^q\right), \alpha_S\right)$$

$$\dot{W}_{ij}^{pq}(t) = \lambda_W\left(\sigma\left(h_i^p\right)\sigma\left(h_j^q\right) - \Theta\left(\max_{j'}\left(W_{ij'}^{pq} / S_{ij'}^{pq}\right) - 1\right)\right) W_{ij}^{pq} \tag{18.6}$$

Recognition dynamics:

$$r^p(t_0) = 1$$

$$\dot{r}^p(t) = \lambda_r r^p\left(F^p - \max_{p'}\left(r^{p'} F^{p'}\right)\right) \tag{18.7}$$

$$F^p(t) = \sum_i \sigma\left(h_i^p\right)$$

Table 18.1
Formulas of the DLM face recognition system

Variables:	
h	internal state of layer neurons
s	delayed self-inhibition
a	attention
W	synaptic weights between neurons of two layers
r	Recognition variable
F	total activity of each neuron (fitness)
Indices:	
$(p;p' ; q;q')$	layer indices, -1 indicates image layer, 1,...,M indicate model layers
$=(-1; -1; 1,...,M; 1,...,M)$	if equations describe image layer dynamics
$=(1,...,M; 1,...,M; -1; -1)$	if equations describe model layer dynamics
$(i; i'; j; j')$	two-dimensional indices for individual neurons in layers $(p; p'; q; q')$ respectively
Functions:	
$g_{i-i'}$	Gaussian interaction kernel
$\sigma(h)$	nonlinear squashing function
$\Theta(\cdot)$	heavy side function
$S_\phi(J,J')$	similarity between feature jets J and J'
Parameters:	
$\beta_h = 0.2$	strength of global inhibition
$\beta_a = 0.02$	attention blob global inhibition strength
$\beta_{ac} = 1$	global inhibition strength compensating for attention blob
$\beta_\theta = \infty$	model supression global inhibition strength
$\kappa_{hs} = 1$	self inhibition strength
$\kappa_{hh} = 1.2$	image and model layers interaction strength
$\kappa_{ha} = 0.7$	attention blob effect on running blob
$\kappa_{ah} = 3$	running blob effect on attention blob
$\lambda_\pm$	delayed self-inhibition decay constant
$= \lambda_+ = 0.2$	if h-$s > 0$
$= \lambda_- = 0.004$	if h-$s \leq 0$
$\lambda_a = 0.3$	attention dynamics time constant
$\lambda_W = 0.05$	link dynamics time constant
$\lambda_r = 0.02$	recognition dynamics time constant
$\alpha_N = 0.1$	attention blob initialization constant
$\alpha_S = 0.1$	minimal weight
$\rho = 2$	squashing function slope radius
$\sigma_g = 1$	excitatory interaction kernel Gauss width
$r_\theta = 0.5$	model suppression threshold

Table 18.2
Variables and parameters of the DLM face recognition system.

Blob Formation

Blob formation on a layer of neurons can easily be achieved by local excitation and global inhibition (consider equations 18.1, 18.3, and 18.4 with $\kappa_{hs} = \kappa_{hh} = \kappa_{ha} = \beta_\theta = 0$; cf. also Amari 1977). Local excitation is conveyed by the Gaussian interaction kernel *g* and generates clusters of activity. Global inhibition, controlled by β_h, lets the clusters compete against each other. The strongest one will finally suppress all others and grow to an equilibrium size determined by the strength of global inhibition.

Blob Mobilization

Generating a running blob can be achieved by delayed self-inhibition *s*, which drives the blob away from its current location to a neighboring one, where the blob generates new self-inhibition. This mechanism produces a continuously moving blob (consider equations 18.1 and 18.2 with $\kappa_{hh} = \kappa_{ha} = \beta_\theta = 0$; see also figure 18.3). In addition, the self-inhibition serves as a memory and repels the blob from regions recently visited. The driving force and the recollection time as to where the blob has been can be independently controlled by the time constants λ_+ and λ_-, respectively.

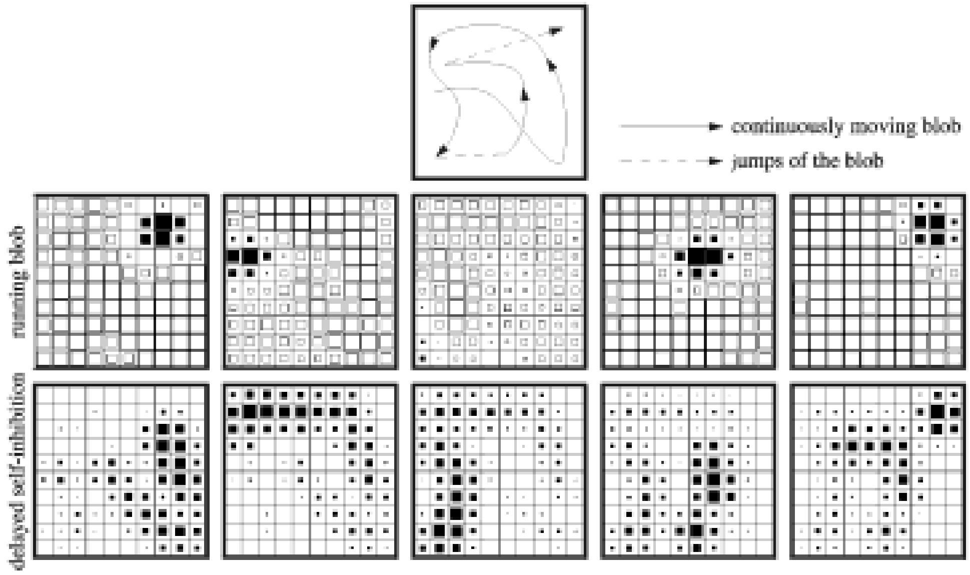


Figure 18.3
A sequence of layer states. The activity blob h shown in the middle row has a size of approximately six active nodes and moves continuously over the whole layer. Its course is shown in the upper diagram. The delayed self-inhibition s, shown in the bottom row, follows the running blob and drives it forward. One can see the self-inhibitory tail, which repels the blob from regions just visited. Sometimes the blob runs into a trap (cf. column three) and has no way to escape from the self-inhibition. It then disappears and reappears again somewhere else on the layer. (The temporal increment between two successive frames is 20 time units.)

Layer Interaction and Synchronization

In the same way as the running blob is repelled by its self-inhibitory tail, it can also be attracted by excitatory input from another layer, as conveyed by the connection matrix *W* (consider equation 18.1 with $\kappa_{ha} = \beta_\theta = 0$). Imagine two layers of the same size mutually connected by the identity matrix, i.e. each neuron in one layer is connected only with the one corresponding neuron in the other layer having the same index value. The input then is a copy of the blob of the other layer. This favors alignment between the blobs, because then they can cooperate and stabilize each other. This synchronization principle hold also in the presence of the noisy connection matrices generated by real image data (see figure 18.4). (The reason why we use the maximum function instead of the usual sum will be discussed later on)

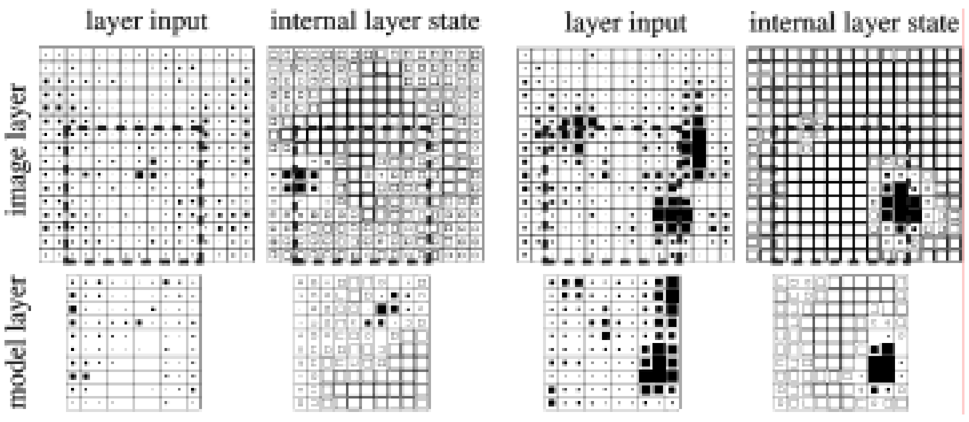


Figure 18.4 Synchronization between two running blobs. Layer input as well as the internal layer state *h* is shown at an early stage, in which the blobs of two layers are not yet aligned, left, and at a later state, right, when they are aligned. The two layers are of different size, and the region in Layer 1 which correctly maps to Layer 2 is indicated by a square defined by the dashed line. In the early non-aligned case one can see that the blobs are smaller and not at the location of maximal input. The locations of maximal input indicate where the actual corresponding neurons of the blob of the other layer are. In the aligned case the blobs are larger and at the locations of high layer input.

Link Dynamics

Links are initialized by the similarity S_ϕ between the jets J of connected nodes (see Wiskott 1995), with a guaranteed minimal synaptic weight of α_S Then, they become cleaned up and structured on the basis of correlations between pairs of neurons (consider equation 18.6; see also figure 18.1). The correlations, defined as $\sigma\left(h_i^p\right)\sigma\left(h_j^q\right)$, result from the layer synchronization described in the previous section. The link dynamics typically consists of a growth term and a normalization term. The former lets the weights grow according to the correlation between the connected neurons. The latter prevents the links from growing infinitely and induces competition such that only one link per neuron survives, suppressing all others.

Attention Dynamics

The alignment between the running blobs depends very much on the constraints, i.e. on the size and format of the layer on which they are running. This causes a problem, since the image and the models have different sizes. We have therefore introduced an attention blob *a* which restricts the movement of the running blob on the image layer to a region of about the same size as that of the model layers (consider equations 18.1 and 18.5 with $\beta_\theta = 0$). The basic dynamics of the attention blob is the same as for the running blob, except there is no self-inhibition. The model layers also have the same attention blob to keep the conditions for their running blobs similar to that in the image layer (only one attention blob is effectively used for all models for computational efficiency). This is important for the alignment. The attention blob restricts the region for the running blob via the term

$$\kappa_{ha}\left(\sigma\left(a_i^p\right) - \beta_{ac}\right) \tag{18.8}$$

with the excitatory blob

$$\sigma\left(a_i^p\right) \tag{18.9}$$

compensating the constant inhibition β_{ac}. The attention blob on the other hand gets excitatory input

$$\kappa_{ha}\left(\sigma\left(h_i^p\right)\right) \tag{18.10}$$

from the running blob and can thus be shifted into a region where input is especially large and favors activity. The attention blob therefore automatically aligns with the actual face position (see figure 18.5). The attention blob layer could be initialized based on pre-attentive segmentation cues, such as texture or color. However, we use a flat initialization and leave the alignment of the attention blob to an initial synchronization phase based purely on the similarity values of the image jets with the model jets in the gallery.

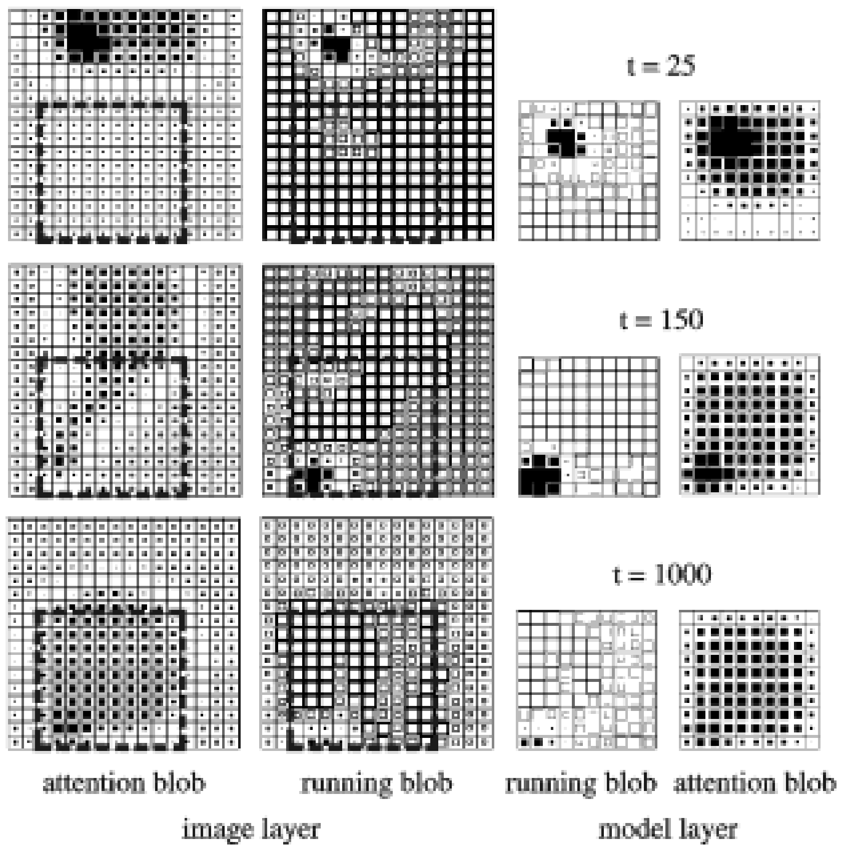


Figure 18.5
Function of the attention blob, using an extreme example of an initial attention blob manually misplaced for demonstration. At *t*=150 the two running blobs ran synchronously for a while, and the attention blob has a long tail. The blobs then lost alignment again. From *t*=500 on, the running blobs remained synchronous, and eventually the attention blob aligned with the correct face position, indicated by a square made of dashed lines. The attention blob moves slowly compared to the small running blob, as it is not driven by self-inhibition. Without an attention blob the two running blobs may synchronize sooner, but the alignment will never become stable.

Recognition Dynamics

We have derived a winner-take-all mechanism from Eigen's (1978) evolution equation and applied it to detect the best model and suppress all others (See equations 18.1 and 18.7). Each model cooperates with the image depending on its similarity. The most similar model cooperates most successfully and is the most active one. We consider the total activity of the model layer p as fitness F^p. The layer with the highest fitness suppresses all others (as can easily be seen if the F^p are assumed to be constant in time and the recognition variables r^p are initialized to 1). When a recognition variable r^p drops below the suppression threshold r_θ, the activity of layer p is suppressed by the term

$$-\beta_\theta \Theta\left(r_\theta - r^p\right) \tag{18.11}$$

Bidirectional Connections

The connectivity between two layers is bidirectional and not unidirectional as in the previous system (Konen and Vorbrüggen 1993). This is necessary for two reasons: Firstly, by this means the running blobs of the two connected layers can more easily align. With unidirectional connections one blob would systematically run behind the other. Secondly, connections in both directions are necessary for a recognition system. The connections from model to image layer are necessary to allow the models to move the attention blob in the image into a region which fits the models well. The connections from the image to the model layers are necessary to provide a discrimination cue as to which model best fits the image. Otherwise, each model would exhibit the same level of activity.

Blob Alignment in the Model Domain

Since faces have a common general structure, it is advantageous to align the blobs in the model domain to insure that they are always at the same position in the faces, either all at the left eye or all at the chin etc. This is achieved by connections between the layers, expressed by the term

$$+\sum_{i'} g_{i-i'} \max_{p'} \sigma\left(h_{i'}^{p'}\right) \tag{18.12}$$

instead of

$$+\sum_{i'} g_{i-i'} \sigma\left(h_{i'}^{p}\right) \tag{18.13}$$

in equation 18.1. If the model blobs were to run independently, the image layer would get input from all face parts at the same time, and the blob there would have a hard time to align with a model blob, and it would be uncertain whether it would be the correct one. The cooperation between the models and the image would depend more on accidental alignment than on the similarity between the models and the image, and it would then be likely that the wrong model was picked up as the recognition result. One alternative is to let the models inhibit each other such that only one model would have a blob at a time. The models then would share time to match onto the image, and the best fitting one would get most of the time. This would probably be the appropriate setup if the models were of different structure, as is the case for arbitrary objects.

Maximum Versus Sum Neurons

The model neurons used here use the maximum over all input signals instead of their sum. The reason is that the sum would mix up many different signals, while only one can be correct, i.e. the total input would be the result of one correct signal mixed with many distractions. Hence the signal-to-noise ratio would be low. We have observed an example where even a model identical to the image was not picked as the correct one, because the sum over all the accidental input signals favored a completely different-looking person. For that reason we introduced the maximum input function, which is reasonable since the correct signal is likely to be the strongest one. The maximum rule has the additional advantage that the dynamic range of the input into a single cell does not vary much when the connectivity develops, whereas the signal sum would decrease significantly during synaptic re-organization and let the blobs loose their alignment.

18.3 Model Implementation

The **DlmModel** is made of a top level **Dlm** module as shown in figure 18.6. These sub-modules are related to the image/object domain (layer 1), model domain (layer 2) or their interconnection.

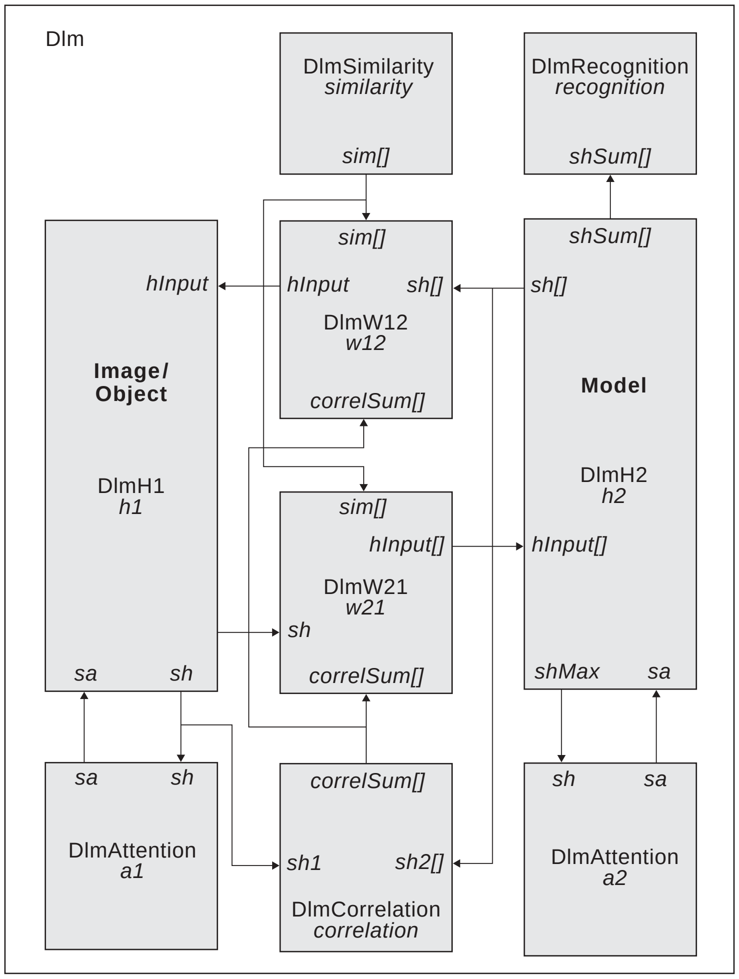


Figure 18.6
The **Dlm** module includes image or object domain modules (layer 1) and model domain (layer 2). The object domain comprises *h1* and *a1* modules, the model domain comprises *h2*, *a2*, *recognition* and *similarity*, while the interconnection modules are implemented by *w12*, *w21* and *correlation* modules. Note that some ports in the submodules have a "[]" (brackets) ending; these represent port arrays instead of the usual single value ports. The connections between such ports are actually multiple ones.

An important concern of this model is how to implement the fact that the model layer manipulates multiple faces, the ones stored in the database, as opposed to the image/ object layer representing a single one to be compared against. We had the choice to create multiple model, recognition, attention, connection and correlation modules corresponding each to a single face transformation. This would have incremented the total number of modules in the system together with its complexity. Instead, we chose to have single model layer modules representing each multiple face transformations. To make this possible we implemented port arrays instead of single ports in each of these modules when appropriate (see the "[]" (brackets) port array notation in the figure). For example, the *sim* input port array in many of the modules, such as in *w12*, is defined as a **NslDinFloat4** array of size *gallerySizeMax* to make it really a 5-dimensional array,

```
public NslDinFloat4 sim()[gallerySizeMax];
```

Note that in this case we use dynamic memory allocation since no instantiation parameters were given above. For example, in the *w12* module, the *sim* port array is assigned memory space as follows (see Appendix I for further details),

```
for (int i=0; i<gallerySizeMax;i++)
   sim[i].nslMemAlloc(i2max,j2max,i1Rmax,j1Rmax);
```

Thus, module interconnections for port array interconnections require a "for loop" style format as follows,

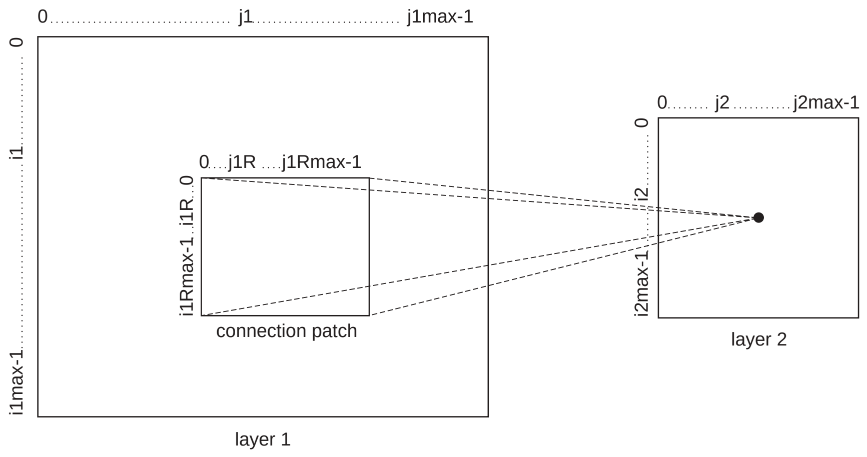
```
   for (int i=0; i<gallerySizeMax; i++)
      nslConnect(similarity.sim[i],w12.sim[i]);
```

Note that *gallerySizeMax* represents the maximum number of gallery faces used from the database for comparison purposes.

Similarity Module

The **Similarity** module performs the initial DLM model processing. The DLM database, implemented as special text and binary files (see Appendix I for a detailed description of text file manipulation and Appendix III for NSLC extensions to binary files), consists of objects and models in the form of stored graphs with the precomputed Gabor-wavelet transform coefficients. For the models they are taken from a grid of 10 x 10 nodes centered on the faces. For the objects the grids cover the whole image plane with 16 x 17 nodes. From these stored graphs a subgraph can be selected. From the 16 x 17 graph for example a 12 x 12 subgraph will automatically be selected if the size of layer 1 is 12 x 12. In addition one can choose the location of the subgraph by *Si1offset* and *Sj2offset* offsets given as integer numbers behind the model names in the gallery files.

Since the object and model layers vary in size a mapping must be created to match elements in both. This is achieved by using a connection patch in layer 1 corresponding to a single cell in layer 2, as shown in figure 18.7.



0 .... j1 .... j1max-1
0 .... i1 .... i1max-1
0....j1R j1Rmax-1
0 .. i1R .. i1Rmax-1
connection patch
layer 1
0........ j2 .... j2max-1
0 .... i2 .... i2max-1
layer 2

Figure 18.7
Matrix representation of layers 1 and 2, and the connectivity patch from a cell in layer 2 to layer 1.

The size of the patches in layer 1 is always *i1Rmax* x *j1Rmax*, but their position depends on the position of the corresponding cell in layer 2 (layer 2 can be as large as layer 1). If we consider only one dimension, for example the *i1*-index, the patches would have *i1max*-*i1Rmax*+1 different offsets varying in range from 0 to *i1max*-*i1Rmax*). These offsets should be equally distributed depending on the different positions in range from 0 to *i2max*-1 in layer 2. Since *i2*/(*i2max*-1) lies in the range [0..1], (*i1max*-*i1Rmax*)*i2/(*i2max*-1) covers the correct range [0,..,(*i1max*-*i1Rmax*)]. In order to round it to the closest integer we define an index function named *i1Index* returning (*i1max*-*i1Rmax*)**i2*/(*i2max*-1)+0.5. For example, given a layer 1 of size 7 and a patch size of 4, there are 4 different offsets, from 0 to 3.

In terms of actual computation, the **initSys** method initializes the *sim* array (order of five) corresponding to the module's output by computing the similarity between the original *DlmGImage* and the different *DlmGModel* library images, both read from external files (not shown here). For all models taken from the gallery (1 to *gallerySize*), for every element (*i2*, *j2*) in layer 2 (*i2max* and *j2max* are the 2-dimension sizes of the model layer), for every element (*i1R*, *j1R*) in the connection patch and for every offset element (*i1*, *j1*) in layer 1 (*i1Rmax* and *j1Rmax* represent the 2-dimension sizes of the patch while *i1max* and *j1max* represent the 2-dimension sizes of the object layer), compute the similarity as follows,

```
for (int model=1; model<=gallerySize; model++) {
  for (int i2=0; i2<i2max; i2++)
    for (int j2=0; j2<j2max; j2++)
      for (int i1R=0,int i1=i1Index
        (i2,0,frame,i1max,i1Rmax,i2max);
          i1R<i1Rmax; i1R++, i1++)
            for (int j1R=0,int j1=j1Index
              (j2,0,frame,j1max,j1Rmax,j2max);
                j1R<j1Rmax; j1R++, j1++) {
                  sim[model][i2][j2][i1R][j1R] =
                    nslMax(alpha_s,
                      similarity(DlmGImage,i1,j1,DlmGModel,
                        i2,j2));
             }
}
```

(Note that `model=0` represents the average face model.) There are a few aspects to note in the above code. The *i1Index* (and *j2Index*) functions include a *frame* variable. The *frame*, whose elements are not connected to layers 2, is put around layer 1, the object layer, in order to give the attention blob space to move around the border of layer 1. If *i1max* and *j1max* are equal in size as *i2max* and *j2max* then no attention blob is required and the frame not necessary. The actual *i1Index* function (analogous to the *j2Index* function) is as follows,

```
int i1Index(int i2, int i1R, int frame,int i1max,int i1Rmax,
    int i2max){
        return i1R + frame + (i1max-2*frame-i1Rmax)*i2/
            (i2max-1)+0.5;
}
```

Note also in the similarity equation, that the value of *sim*, for each model, is assigned as the maximum value between parameter *alpha_s* and the resulting similarity value. This is done to restrict minimum values in *sim*.

Once the similarity computation has been completed, the average layer values (*model*=0) are set the maximum of all models.

```
for (int i2=0; i2<i2max; i2++)
    for (int j2=0; j2<j2max; j2++)
        for (int i1R=0; i1R<i1Rmax; i1R++)
            for (int j1R=0; j1R<j1Rmax; j1R++) {
            float s = 0;
            for (int model=1; model<=gallerySize; model++)
                s = nslMax(s,sim[model][i2][j2][i1R][j1R]);
            sim[0][i2][j2][i1R][j1R] = s;
        }
```

H Module

To take advantage of common functionality between the image and layer models, a *supermodule* **H** is defined containing aspects common to both **H1** and **H2**. At the structure level the two submodules **H1** and **H2** share a number of variables having the exact same dimension, corresponding to previously defined equation symbols, as shown in table 18.3 (we omit all scalar parameters from the table).

Symbol	Variable Name	Variable Type	Description
$G_i^p = \sum_{i'} g_{i-i'} \max_{p'}\left(\sigma\left(h_{i'}^{p'}\right)\right)$	`hTransE`	`NslFloat2`	Gaussian lateral interaction
$\sigma\left(a_i^p\right)$	`sa`	`NslDinFloat2`	Input received from the attention module
$h_i^p - s_i^p$	`d`	`NslFloat2`	Delayed self inhibition argument

Table 18.3 Symbol and variable relationship defined in **H** and common to both **H1** and **H2** in dimension.

Additionally, the following method implements equation (18.3) and is used to initialize the Gaussian variable *g* used in both submodules,

```
protected void initGauss(){
   for (int k = 0; k < gSize; k++) {
      float x = (float) (k-gSigma/2);
      g[k] = nslExp(-x*x/(2*gSigma*gSigma);
   }
}
```

Additionally, the following method performs the convolution described in the second hand side element of equation (18.1) performed by both submodules,

```
protected NslFloat2 gaussConvolved(NslFloat1 g,NslFloat2 sh){
   return nslConvZero(nslFillRows(g,1),nslConvZero
      (nslFillCols(g,1),sh));
}
```

The above function generates a two dimensional array as result from this convolution. It first convolves *sh* against a matrix whose columns are replications (*nslFillCols*) of the gaussian vector *g*. The result of this convolution is applied to a matrix whose rows are replications (*nslFillRows*) of the gaussian vector *g*. We use a convolution function treating elements beyond the matrix border as zeroes (*nslConvZero*).

H1 Module

The image layer implementation defines a number of variables corresponding to previously defined equation symbols, as shown in table 18.4.

Table 18.4 Symbol and variable relationship defined in **H1**.

Symbol	Variable Name	Variable Type	Description
$I_i^p = \max_{qj}\left(W_{ij}^{pq}\sigma\left(h_j^q\right)\right)$	`hInput`	`NslDinFloat2`	Input received from *hInput* from *w12* layer
$\sigma\left(h_i^p\right)$	`sh`	`NslDoutFloat2`	Layer output
h_i^p	`h`	`NslFloat2`	Layer activity
s_i^p	`s`	`NslFloat2`	Delayed self inhibition

The **simRun** main computation, describing layer dynamics on *h* and *s* in module *h1*, is as follows:

1. Calculate the Gaussian lateral interaction *hTransE* in the image layer,

$$G_i^p = \sum_{i'} g_{i-i'} \max_{p'}\left(\sigma\left(h_{i'}^{p'}\right)\right) \tag{18.14}$$

corresponding to the second term of the right hand size of equation (18.1) where *g* has been computed in the **initRun** method during initialization,

```
hTransE = gaussConvolved(g,sh);
```

The maximum operation is not necessary since there is only one image layer *p*.

2. Integrate the differential equation for the image layer in correspondence to equation (18.1),

$$\dot{h}_i^p = -h_i^p + \sum_{i'} g_{i-i'} \max_{p'}\left(\sigma\left(h_{i'}^{p'}\right)\right) - \beta_h \sum_{i'} \sigma\left(h_{i'}^p\right) - \kappa_{hs} s_i^p$$
$$+ \kappa_{hh} \max_{qj}\left(W_{ij}^{pq}\sigma\left(h_j^q\right)\right) + \kappa_{ha}\left(\sigma\left(a_i^p\right) - \beta_{ac}\right) - \beta_\theta \Theta\left(r_\theta - r^p\right)$$

The equation is implemented as follows,

```
nslDiff(h,1.,- h + hTransE - beta_h*nslSum(sh) - kappa_hs*s
   + kappa_hh*hInput + kappa_ha*(sa-beta_ac));
```

Note that *nslSum*(*sh*) corresponds to the following expression,

$$\sum_{i'} \sigma\left(h_{i'}^p\right)$$

Additionally notice that the strong inhibition term

$$-\beta_\theta \Theta\left(r_\theta - r^q\right)$$

is ineffective in the image layer since there are no competing image layers.

3. Compute the image layer output *sh* from current activity *h* as given by equation (18.4),

```
computeOutputFunc(sh,h,rho);
```

4. Integrate the differential equation (equation 18.2) given by,

$$\dot{s}_i^p = \lambda_\pm\left(h_i^p - s_i^p\right)$$

with *lambda* depending on the sign of the difference of *h* and *s*,

```
d = h - s;
for (i1=0; i1<i1max; i1++)
   for (j1=0; j1<j1max; j1++)
      if (d[i1][j1]>0)
         d[i1][j1] = d[i1][j1]*lambda_p;
      else
         d[i1][j1] = d[i1][j1]*lambda_m;
nslDiff(s,1.,d);
```

H2 Module

The image layer implementation defines a number of variables corresponding to previously defined equation symbols, as shown in table 18.5. Notice that these variables differ from those defined in module **H1** in that an additional dimension has been added in the form of an array (having its size correspond *gallerySizeMax*).

Symbol	Variable Name	Variable Type	Description
$I_j^q = \max_{pi}\left(W_{ji}^{qp}\sigma\left(h_i^p\right)\right)$	`hInput`	`NslDinFloat2[]`	Input received from *hInput* from *w12* layer
$\sigma\left(h_j^q\right)$	`sh`	`NslDoutFloat2[]`	Layer output
$\sum_{j'}\sigma\left(h_{j'}^q\right)$	`shSum`	`NslDoutFloat0[]`	Sum over *sh*
$\max_{q'}\left(\sigma\left(h_j^{q'}\right)\right)$	`shMax`	`NslDoutFloat2[]`	Maximum activity of layer 2
h_j^q	`h`	`NslFloat2[]`	Layer activity
s_j^q	`s`	`NslFloat2[]`	Delayed self inhibition

Table 18.5
Symbol and variable relationship defined in **H2**. Notice that all types have an additional dimension specified by the "[]" array symbol.

The **simRun** main computation describing layer dynamics on *h* and *s* in module *h2*, is quite similar to module *h1*, being as follows:

5. Calculate the Gaussian lateral interaction *hTransE* in the model layer,

$$G_j^q = \sum_{j'} g_{j-j'} \max_{q'}\left(\sigma\left(h_{j'}^{q'}\right)\right)$$

Since the maximum operation is equal for all models it needs to be calculated only once.

```
hTransE = gaussConvolved(g,shMax,rho);
```

Notice that we have an additional variable *shMax* in **H2**.

6. Integrate the differential equation for the model layer in correspondence to equation (18.1),

$$\dot{h}_j^q = -h_j^q + \sum_{j'} g_{j-j'} \max_{q'}\left(\sigma\left(h_{j'}^{q'}\right)\right) - \beta_h \sum_{j'}\sigma\left(h_{j''}^q\right) - \kappa_{hs} s_j^q$$
$$+ \kappa_{hh} \max_{pi}\left(W_{ji}^{qp}\sigma\left(h_i^p\right)\right) + \kappa_{ha}\left(\sigma\left(a_j^q\right) - \beta_{ac}\right) - \beta_\theta \Theta\left(r_\theta - r^q\right)$$

The equation is implemented as follows and applied to each model,

```
nslDiff(h[model],1.0, - h[model] + hTransE
    - beta_h*shSum[model] - kappa_hs*s[model]
    + kappa_hh*hInput[model] + kappa_ha*(sa-beta_ac));
```

Notice that the strong inhibition term

$$-\beta_\theta \Theta\left(r_\theta - r^q\right)$$

is simulated simply by skipping over those layers that have too low recognition values r^q (skip statement in the source code), else continue processing the model layer.

Also notice that the output sum *shSum* is computed for each model as follows,

```
shSum[model] = nslSum(sh[model]);
```

7. Compute for each model layer output *sh* from current activity *h* as described in equation (18.4),

```
computeOutputFunc(sh[model],h[model],rho);
```

8. Integrate for each model the differential equation (equation 18.2) given by,

$$\dot{s}_j^q = \lambda_\pm \left(h_j^q - s_j^q \right)$$

with *lambda* depending on the sign of the difference of *h* and *s*,

```
d = h[model] - s[model];
for (i2=0; i2<i2max; i2++)
   for (j2=0; j2<j2max; j2++)
      if (d[i2][j2]>0)
         d[i2][j2] = d[i2][j2]*lambda_p;
      else
         d[i2][j2] = d[i2][j2]*lambda_m;
nslDiff(s[model],1.,d);
```

9. Compute the model layer output *shMax*

$$\max_{q'} \left(\sigma \left(h_j^{q'} \right) \right)$$

implemented by,

```
for (i2=0; i2<i2max; i2++)
   for (j2=0; j2<j2max; j2++) {
      shMax[i2][j2] = 0;
      for (int model=1; model<=gallerSizeMax; model++)
         shMax[i2][j2] = nslMax(shMax[i2][j2],sh[model]
            [i2][j2]);
   }
```

Attention Module

We take advantage of the similarity between the attention modules for image and model layers to define a single one instantiated twice, respectively. The **Attention** module defines equation symbols and variable names, as shown in table 18.5.

Symbol	Variable Name	Variable Type	Description
a_i^p	a	NslFloat2	Attention layer activity
$\sigma\left(a_i^p\right)$	sa	NslDoutFloat2	Attention layer output
$A_i^p = \sum_{i'} g_{i-i'} \sigma\left(a_{i'}^p\right)$	aTransE	NslFloat2	Gaussian lateral interaction
$\sigma\left(h_i^p\right)$ or $\max_{p'}\left(\sigma\left(h_i^{p'}\right)\right)$	sh	NslDinFloat2	Layer input received from the image or model (max) layer, respectively.

Table 18.5
Symbol and variable relationship.

Attention modules compute only if attention is set. The **initRun** method initializes the attention layer (**a1** and **a2**) as follows,

```
a = alpha_N;
sa = computeOutputFunc(a);
```

The **simRun** method describes the attention dynamics for the two layers,

10. Calculate the Gaussian lateral interaction *aTransE* from its previous output *sa*,

$$A_i^p = \sum_{i'} g_{i-i'} \sigma\left(a_{i'}^p\right) \tag{18.15}$$

The implementation is as follows,

```
aTransE = gaussConvolved(g,sa);
```

Due to the maximum operation there is effectively only one attention layer for all models,

11. Integrate differential equation for the attention layer as described in equation (18.5) by obtaining inputs from the image or model layer outputs, respectively,

$$\dot{a}_i^p = \lambda_a\left(-a_i^p + \sum_{i'} g_{i-i'}\sigma\left(a_{i'}^p\right) - \beta_a \sum_{i'} \sigma\left(a_{i'}^p\right) - \kappa_{ah}\sigma\left(h_i^p\right)\right)$$

The equation is implemented as follows,

```
nslDiff(a,1.,lambda_a*(-a+aTransE-beta_a*nslSum(sa)+
   kappa_ah*sh));
```

12. Compute the output function

$$\sigma\left(a_i^p\right)$$

The equation is implemented as follows,

```
computeOutputFunc(sa,a,rho);
```

W Module
To take advantage of common functionality between the image and layer model connectivity, a *supermodule* **W** is defined containing aspects common to both **W12** and **W21**. At the structure level the two submodules **W12** and **W21** share a number of variables having the exact same dimension, corresponding to previously defined equation symbols, as shown in table 18.6 (we omit all scalar parameters from the table).

Symbol	Variable Name	Variable Type	Description
W_{ij}^{pq}	`w`	`NslFloat4[]`	Connection weights
$C_{ij}^{qp} = \sigma\left(h_i^p\right)\sigma\left(h_j^q\right)$	`correlSum`	`NslDinFloat4[]`	Input from **Correlation** module
S_{ij}^{qp}	`sim`	`NslDinFloat4[]`	Input from **Similarity** module

Table 18.6
Symbol and variable relationship defined in **W** and common to both **W12** and **W21** in dimension. Notice that all types have an additional dimension specified by the "[]" array symbol.

Notice that since the link dynamics for both connection layers is simulated only after every *loops* iterations.

W12 Module
The connectivity layer from the model layer to the image layer is defined by module **W12**. The symbols particular to this layer are shown in table 18.7. In addition this layer inherits all symbols defined in supermodule **W**.

Symbol	Variable Name	Variable Type	Description
$I_i^p = \max_{qj}\left(W_{ji}^{pq}\sigma\left(h_j^q\right)\right)$	hInput	NslDoutFloat2	Output to image layer
$\sigma\left(h_j^q\right)$	`sh`	`NslDinFloat2[]`	Input from model layer
$N_i^p = \min_{qj}\left\{1, S_{ij}^{qp} / W_{ji}^{pq}\right\}$	`normFactor`	`NslFloat2`	Normalization matrix

Table 18.7
Symbol and variable relationship defined in **W12**. Notice the types having an additional dimension specified by the "[]" array symbol.

The **initRun** method computes the initialization values for the connection module given by the following equation (equation 18.6),

$$W_{ji}^{pq}(t_0) = S_{ij}^{qp} = \max\left(S_\phi\left(J_i^p, J_j^q\right), \alpha_S\right)$$

Taking out the nested "for" loops, the equation is implemented as follows,

```
w[model][i1R][j1R][i2][j2] = sim[model][i2][j2][i1R][j1R];
```

Notice how we switch subscripts since **W12** connects layer 2 elements to layer 1 patch elements. Also, the *max* function has already been applied in the similarity module. Additionally, the **initRun** method computes maximum values for the average layer (*model*=0).

```
float stmp = 0;
for (model=1; model<=gallerySize; model++)
   stmp = nslMax(stmp,sim[model][i2][j2][i1R][j1R]);
w[0][i1R][j1R][i2][j2] = stmp;
```

The **simRun** method describes connection dynamics on **W12**. Computation is as follows:

13. Calculate growth of the weight matrix *w* from model layers to image layer according to the first term in differential equation equation (18.6) based on the accumulated correlations and according to the equation

$$\dot{W}_{ji}^{pq}(t) = \lambda_W C_{ij}^{qp} W_{ji}^{pq} \tag{18.16}$$

Since this integration switches scripts around we perform the integration directly as follows,

$$W_{ji}^{pq} = W_{ji}^{pq} + \lambda_W C_{ij}^{qp} W_{ji}^{pq} \Delta t$$

The latter is implemented by the following code,

```
w[model][i1R][j1R][i2][j2] += nslSystem.getSimDelta()
   *w[model][i1R][j1R][i2][j2]*lambda_W
   *correlSum[model][i2][j2][i1R][j1R];
```

Note the "+=" expression directly adding the left hand side variable to the right hand side. The "nslSystem.getSimDelta()" expression returns the system's "delta".

14. Although link dynamics is not simulated as a differential equation but by strict normalization, the outcome is the same. The normalization rule corresponding to the second term in equation (18.6) becomes an explicit and separate normalization rule in the program. The normalization factors by which the weights converging on the image layer need to be multiplied is

$$N_i^p = \min_{qj}\left\{1, S_{ij}^{qp} \big/ W_{ji}^{pq}\right\}$$

Notice that *sim* is symmetric and can be used for both directions. The variable *normFactor* is initialized to 1. Also notice that each model layer needs its own set of normalization factors for *w21* but not for *w12* (although we end up computing one for each anyway).

```
if (w[model][i1R][j1R][i2][j2]>sim[model][i2][j2][i1R][j1R])
  normFactor[i1][j1] = minimum(normFactor[i1][j1],
    sim[model][i2][j2][i1R][j1R]/w[model][i1R][j1R][i2][j2]);
```

15. Normalize the weights going from each model layer to the image layer by the normalization factors,

$$W_{ji}^{pq} = N_i^p W_{ji}^{pq}$$

```
w[model][i1R][j1R][i2][j2] *= normFactor[i1][j1];
```

Note the "`*=`" expression directly multiplying the left hand side variable with the right hand side.

16. Skip computation if inhibition in model layer `sh` is too strong. (There is no competition between links going to different models while there is competition between links converging to the image layer from different models.)
17. Calculate the output *hInput* sent to the image layer calculated as the maximum of the input *sh* from the model layer multiplied by the connection *w* as described in equation (18.1),

$$\max_{pi}\left(W_{ji}^{pq}\sigma\left(h_i^p\right)\right)$$

The implementation is as follows,

```
hInput[i1][j1] = nslMax(hInput[i1][j1],
   w[model][i1R][j1R][i2][j2]*sh[model][i2][j2]);
```

W21 Module

The connectivity layer from the image layer to the model layer is defined by module **W21**. The symbols particular to this layer are shown in table 18.8. In addition this layer inherits all symbols defined in supermodule **W**.

Symbol	Variable Name	Variable Type	Description
$I_j^q = \max_{pi}\left(W_{ij}^{qp}\sigma\left(h_i^p\right)\right)$	`hInput`	`NslDoutFloat2[]`	Output to model layer
$\sigma\left(h_i^p\right)$	`sh`	`NslDinFloat2`	Input from image layer
$N_j^q = \min_{pi}\left\{1, S_{ij}^{qp} \big/ W_{ij}^{qp}\right\}$	`normFactor`	`NslFloat2[]`	Normalization matrix

Table 18.8
Symbol and variable relationship defined in W21. Notice the types having an additional dimension specified by the "[]" array symbol, exactly the opposite from those specified in W12.

The **initRun** method computes the initialization values for the connection module given by the following equation (equation 18.6),

$$W_{ij}^{qp}(t_0) = S_{ij}^{qp} = \max\left(S_\phi\left(J_i^p, J_j^q\right), \alpha_S\right)$$

An important consideration here is that subscripts correspond to those in the similarity output. Taking out the nested "for loops" the equation is implemented as follows,

```
   w[model] = sim[model]
```

Notice how we switch subscripts since **W21** connects layer 1 patch elements to layer 1 elements. Also, the *max* function has already been applied in the similarity module. Additionally, the **initRun** method computes maximum values for the average layer (*model*=0).

```
   float stmp = 0;
   for (model=1; model<=gallerySize; model++)
      stmp = nslMax(stmp,sim[model]);
   w[0] = stmp;
```

The **simRun** method describes connection dynamics on **W21**. Computation is as follows:

18. Calculate growth of the weight matrix *w* from model layers to image layer according to the first term in differential equation equation (18.6) based on the accumulated correlations and according to the equation

$$\dot{W}_{ij}^{qp}(t) = \lambda_W C_{ij}^{qp} W_{ji}^{qp}$$

Since this integration switches scripts around we perform the integration directly as follows,

$$W_{ij}^{qp} = W_{ij}^{qp} + \lambda_W C_{ij}^{qp} W_{ij}^{qp} \Delta t$$

The latter is implemented by the following code,

```
   w[model] += nslSystem.getSimDelta()
      *w[model][i2][j2]*lambda_W*correlSum[model];
```

Note the "+=" expression directly adding the left hand side variable to the right hand side. The "nslSystem.getSimDelta()" expression returns the system's "delta".

19. Although link dynamics is not simulated as a differential equation but by strict normalization, the outcome is the same. The normalization rule corresponding to the second term in equation (18.6) becomes an explicit and separate normalization rule in the program. The normalization factors by which the weights converging on the image layer need to be multiplied is

$$N_j^q = \min_{pi}\left\{1, S_{ij}^{qp} \middle/ W_{ij}^{qp}\right\}$$

Notice that *sim* is symmetric and can be used for both directions. The variable *normFactor* is initialized to 1. Also notice that each model layer needs its own set of normalization factors for *w21* but not for *w12* (although we end up computing one for each anyway).

```
   if (w[model][i2][j2][i1R][j1R]>sim[model][i2][j2][i1R][j1R])
     normFactor[i1][j1] = minimum(normFactor[i1][j1],
       sim[model][i2][j2][i1R][j1R]/w[model][i2][j2][i1R][j1R]);
```

20. Normalize the weights going from each model layer to the image layer by the normalization factors,

$$W_{ij}^{qp} = N_j^q W_{ij}^{qp}$$

```
   w[model][i2][j2][i1R][j1R] *= normFactor[i1][j1];
```

Note the “*=” expression directly multiplying the left hand side variable with the right hand side.

21. Skip computation if inhibition in model layer `sh` is too strong. (There is no competition between links going to different models while there is competition between links converging to the image layer from different models.)

Calculate the output *hInput* sent to the image layer calculated as the maximum of the input *sh* from the model layer multiplied by the connection *w* as described in equation (18.1),

$$\max_{qj}\left(W_{ij}^{qp}\sigma\left(h_j^q\right)\right)$$

The implementation is as follows,

```
   hInput[i1][j1] = nslMax(hInput[i1][j1],
      w[model][i2][j2][i1R][j1R]*sh[model][i2][j2]);
```

Correlation Module
The correlation module between image-model connections and model-layer connections. The correlation symbols and variable names are shown in table 18.9.

Symbol	Variable Name	Variable Name	Description
$\sigma\left(h_i^p\right)$	`Sh1`	`NslDinFloat2`	Image layer input
$\sigma\left(h_j^q\right)$	`Sh2`	`NslDinFloat2[]`	Model layer input
C_{ij}^{qp}	`correlSum`	`NslDoutFloat4[]`	Accumulated correlation

Table 18.9 Symbol and variable relationship.

The **initRun** method initializes variables to zero. The **simRun** method computes the link dynamics on the correlation module.

22. Calculate the input received from the image layer output *sh1* and from the model layer output *sh2* multiplied to compute *correlSum* as described in the following equation,

$$C_{ij}^{qp} = \sigma\left(h_i^p\right)\sigma\left(h_j^q\right) \tag{18.17}$$

Notice that since the link dynamics is simulated only after every *loops* iterations, the correlations are accumulated over time, leading to the += operator. The correlations are symmetric and can be used for the weight matrices from image layer to model layers and vice versa.

```
   correlSum[model][i2][j2][i1R][j1R] +=
      sh1[i1][j1]*sh2[model][i2][j2];
```

23. After the weights have been changed the accumulated correlations need to be reset to zero.

```
correlSum[model] = 0;
```

Recognition Module

The **recognition** module describes the recognition dynamics. If the recognition variable of a model drops below *r_theta*, the model becomes ruled out by a strong inhibition term. In the simulation it is just skipped in order to save cpu-time. The variables and symbols are shown in table 18.10.

Symbol	Variable Name	Variable Type	Description
r^p	rec	NslFloat0[]	Recognition activity
F^p	shSum	NslDinFloat0[]	Recognition sum value from model layers
$R = \max_{p'}\left(r^{p'}F^{p'}\right)$	`recShSumMax`	`float`	Recognition maximum sum value for the model layers

Table 18.10 Symbol and variable relationship.

The **initRun** method initializes the recognition layer (winner-take-all mechanism). The **simRun** method processes the recognition layer (winner-take-all mechanism). Recognition dynamics are

24. Calculate terms *shSum* and *recShSumMax* from equation (18.7),

$$R = \max_{p'}\left(r^{p'}F^{p'}\right)$$

The implementation is as follows,

```
float recShSumMax = 0;
for (model=1; model<=gallerySize; model++)
   recShSumMax = nslMax(recShSumMax,rec[model]*shSum[model]);
```

25. Integrate the recognition dynamics as described in equation (18.7),

$$\dot{r}^p(t) = \lambda_r r^p\left(F^p - \max_{p'}\left(r^{p'}F^{p'}\right)\right)$$

The implementation is as follows,

```
nslDiff(rec[model],1.,lambda_r*rec[model]^(shSum[model] –
   recshSumMax));
```

26. Compute which models to skip. This information is propagated back to other modules requiring this information through port interconnections not shown here.

```
if (model>0 && rec[model] <= r_theta)
   skipModel[model] = 1;
```

18.4 Simulation and Results[3]

Different experiments were carried out using different combinations of layer and patch sizes as shown in table 18.11.

Experiment	Layer and Patch Flags	frame	i1max	j1max	i2max	j2max	i1Rmax	j1Rmax
Blob, layer and link dynamics	*small_layer* = 1 *small_patch* = 0	0	10	10	10	10	10	10
Attention dynamics	*small_layer* = 0 *small_patch* = 0	2	17+4	16+4	10	10	17	16
Recognition dynamics	*small_layer* = 0 *small_patch* = 1	2	17+4	16+4	10	10	8	8

Table 18.11
Size combinations for layers and patches for the different experiments.

The first four experiments: Blob formation, blob mobilization, layer interaction and synchronization and link dynamics use the first combination where flags are set as *small_layer* = 1 and *small_patch* = 0. All modifications are made to the source file.

Blob Formation

Load the simulation file (*gallerySize* = 0, *attention* = 0)

```
   nsl source DLMB
   nsl init
   nsl run
```

and observe how a blob arises.

Restart the simulation with different initial conditions `[Ctrl-C; nsl init;` mouse clicks with left button on layer *h1*; `nsl cont]`.

Vary also β_h, originally set to 0.2 e.g. `[Ctrl-C; nsl set dlm.h1.beta_h 0.1; nsl set dlm.h2.beta_h 0.1; nsl cont]`.

What is a reasonable range for β_h?

Blob Mobilization

Load the simulation file (*gallerySize* = 0, *attention* = 0)

```
   nsl source DLMR
   nsl init
   nsl run
```

and observe how a blob arises and moves over the layer.

Vary λ_+, λ_-, and κ_{hs} (*lambda_p, lambda_m, kappa_hs*), originally set to 0.2, 0.004, and 1, respectively, to e.g. `[Ctrl-C; nsl set dlm.h1.lambda_m 0.001; nsl set dlm.h2.lambda_m 0.001; nsl cont]`.

Why should λ_- be larger for smaller layers? Is the shape of the blob speed-dependent?

Layer Interaction and Synchronization

Load the simulation file (*gallerySize* = 1, *attention* = 0, *workOnAverage* = 0)

```
   nsl source DLMS
   nsl init
   nsl run
```

observe how the two blobs synchronize and align with each other. Try different runs (for each run a new object is selected randomly and some synchronize easier than others) and use different object galleries [edit the file *DLMobjects* and exchange the **pose1* (= 15 degrees rotated faces) block with the **pose2* (= 30 degrees rotated faces) or **pose3* (= different facial expression) block]. Vary κ_{hh} (*kappa_hh*), originally set to 1.2. What happens if κ_{hh} is too large or too small?

Link Dynamics
Load the simulation file (*gallerySize* = 1, *attention* = 0, *workOnAverage* = 0)

```
   nsl source DLMM
   nsl init
   nsl run
```

observe how the connectivity develops in time.

Vary λ_w (*lambda_W*). What happens if λ_w is too large?

Attention Dynamics
Load the simulation file (*gallerySize* = 1, *attention* = 1, *workOnAverage* = 0)

```
   nsl source DLMR
   nsl source DLMA
   nsl init
   nsl run
```

observe how an attention blob arises and restricts the region in which the small blob is allowed to move.

Vary κ_{ah} and κ_{ha} (*kappa_ah, kappa_ha*), originally set to 3 and 0.7, respectively.

Now restart the simulation with

```
   nsl source DLMS
   nsl init
   nsl run
```

and see whether the two blobs on the layers of different size can synchronize without an attention blob.

Then add the attention blob [Ctrl-C; nsl load DLMA; nsl init; nsl run]

and see how the alignment between the blobs can become more stable (notice that for each run a new object is selected randomly, which can be suppressed by

`[nsl set dlm.similarity.ObjectSelectionMode 1]` in which case always the object indicated by `preferredObject` is used; with `[nsl set dlm.similarity. ObjectSelectionMode 3]` objects are selected randomly again).

You can also experiment with the attention blob misplaced in the beginning [Ctrl-C; nsl init; mouse clicks with the left button near the border on layer a1; nsl cont]. Vary κ_{ah} and κ_{ha}.

Recognition Dynamics
Load the simulation file (*gallerySize* = 5, *attention* = 1, *workOnAverage* = 1)

```
   nsl source DLMG
   nsl source DLMA
   nsl init
   nsl run
```

observe the recognition process. In the first 1000 time units only the average layer with index 0 is simulated. The correct model has index 1. Shown are, for all models, the total layer activity, the recognition variable, and the sum over all synaptic weights (cf. also figure). The connectivity and the layer 2 internal state as well as its input is shown only for the currently most active layer. The time, the index of the most active layer, and the values of the recognition parameters are given as usual output. Asterisks indicate layers that have been ruled out.

Data Base

As face database we used galleries of 111 different persons. For most persons there is one neutral frontal view, one frontal view of different facial expression, and two views rotated in depth by 15 and 30 degrees respectively. The neutral frontal views serve as model gallery, and the other three are used as test images for recognition. The models, i.e. the neutral frontal views, are represented by layers of size 10X10. Though the grids are rectangular and regular, i.e. the spacing between the nodes is constant within each dimension, the graphs are scaled horizontally and vertically and are aligned manually: The left eye is always represented by the node in the fourth column from the left and the third row from the top, the mouth lies on the fourth row from the bottom, etc. The *x*- (that is, horizontal) spacing ranges from 6.6 to 9.3 pixels with a mean value of 8.2 and a standard deviation of 0.5. The *y*-spacing ranges from 5.5 to 8.8 pixels with a mean value of 7.3 and a standard deviation of 0.6. An input image of a face to be recognized is represented by a 16X17 layer with an *x*-spacing of 8 pixels and a *y*-spacing of 7 pixels. The image graphs are not aligned, since that would already require recognition. The variations of up to a factor of 1.5 in the *x*- and *y*-spacings must be compensated for by the DLM process.

Technical Aspects

DLM in the form presented here is computationally expensive. We have performed single recognition tasks with the complete system, but for the experiments referred to in table 18.12 we have modified the system in several respects to achieve a reasonable speed. We split up the simulation into two phases. The only purpose of the first phase is to let the attention blob become aligned with the face in the input image. No modification of the connectivity was applied in this phase, and only one average model was simulated. Its connectivity was derived by taking the maximum synaptic weight over all real models for each link:

$$W_{mn}^{a}(t_0) = \max_{pq} W_{mn}^{pq}(t_0)$$

This attention period takes 1000 time steps. Then the complete system, including the attention blob, is simulated, and the individual connection matrices are subjected to DLM. Neurons in the model layers are not connected to all neurons in the image layer, but only to an 8X8 patch. These patches are evenly distributed over the image layer with the same spatial arrangement as the model neurons themselves. This still preserves full translation invariance. Full rotation invariance is lost, but the jets used are not rotation invariant in any case. The link dynamics is not simulated at each time step, but only after 200 simulation steps or 100 time units. During this time a running blob moves about once over all of its layer, and the correlation is integrated continuously. The simulation of the link dynamics is then based on these integrated correlations, and since the blobs have moved over all of the layers, all synaptic weights are modified. For further increase in speed, models which are ruled out by the winner-take-all mechanism are no longer simulated; they are just set to zero and ignored from then on ($\beta_\theta = \infty$). The CPU time needed for the recognition of one face against a gallery of 111 models is approximately 10-15 minutes on a Sun SPARCstation 10-512 with a 50 MHz processor.

In order to avoid border effects, the image layer has a frame with a width of 2 neurons without any features or connections to the model layers. The additional frame of neurons helps the attention blob to move to the border of the image layer. Otherwise, it would have a tendency to stay in the center.

Results

Figure 18.8 shows a sample recognition process using a test face strongly differing in expression from the model. The gallery contains five models. Due to the tight connections between the models, the layer activities show the same variations and differ only little in intensity. This small difference is averaged over time and amplified by the recognition dynamics, which rules out one model after the other until the correct one survives. The example was monitored for 2000 units of simulation time. An attention phase of 1000 time units had been applied before, but is not shown here. We selected a sample run which had exceptional difficulty to decide between models. The sum over the links of the connectivity matrices was even higher for the fourth model than for the correct one. This is a case where the DLM is actually required to stabilize the running blob alignment and recognize the correct model. In some other cases the correct face can be recognized without modifying the connectivity matrix.

Recognition rates for galleries of 20, 50, and 111 models are given in table 8.12. As is already known from previous work (Lades et al. 1993), recognition of depth-rotated faces is in general less reliable than, for instance, recognition of faces with an altered expression. It is interesting to consider recognition times (measured in arbitrary units). Although they vary significantly, a general tendency is noticeable: Firstly, more difficult tasks take more time, i.e. recognition time is correlated with error rate. This is also known from psychophysical experiments (see for example Bruce et al. 1987; Kalocsai et al. 1994). Secondly, incorrect recognition takes much more time than correct recognition. Recognition time does not depend very much on the size of the gallery.

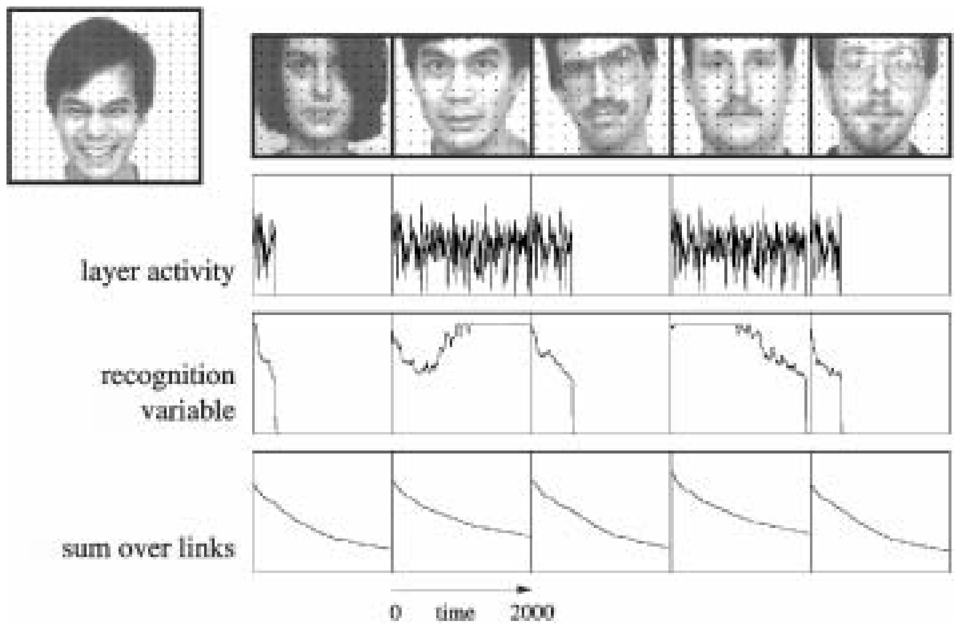


Figure 18.8
DLM recognition: A sample run. The test image is shown on the left, with 16X17 neurons indicated as black dots. The models have 10X10 neurons and are aligned with each other. The corresponding total layer activities, i.e. the sum over all neurons of one model, are shown in the upper graph. The most similar model is usually slightly more active than the others. On that basis the models compete against each other, and eventually the correct one survives, as indicated by the recognition variable. The sum over all links of each connection matrix is shown in the lower graphs. It gives an impression of the extent to which the matrices self-organize before the recognition decision is made.

Gallery Size	Test Images	Correct Recognition Rate		Recognition Time	
		#	%	Correct Recognition	Incorrect Recognition
20	111 rotated faces (15 degrees) 110 rotated faces (30 degrees) 109 frontal views (grimace)	106 93 100	95.5 84.5 91.7	320±490 990±1900 290±530	4110±2860 2480±2580 5610±7480
50	111 rotated faces (15 degrees) 110 rotated faces (30 degrees) 109 frontal views (grimace)	104 83 95	93.7 75.5 87.2	390±500 930±1010 440±1290	3770±2120 2080±1440 4480±6190
111	111 rotated faces (15 degrees) 110 rotated faces (30 degrees) 109 frontal views (grimace)	101 69 92	91.0 62.7 84.4	420±460 1350±3017 380±410	3770±3130 4600±3720 3380±4820

Table 18.12
Recognition results against a gallery of 20, 50, and 111 neutral frontal views. Recognition time (with two iterations of the differential equations per time unit) is the time required until all but one models are ruled out by the winner-take-all mechanism.

18.5 Summary

We routinely use NSL for homework assignments in class as well as for our own research, and we found it appropriate for both. It is very easy to get started with NSL. It provides a reasonable set of basic structures and functions to create a neural model with just a few lines of code. Moreover, when a more complex model requires more than this basic functionality, additional functions and data structures can conveniently be added. As a matter of fact, the model presented here used very little of the NSL-specific algorithms and functions and profited mainly from NSL's graphic display facilities and interactive control structures.

The model presented here deviates in some very fundamental ways from other biological and neural models of vision or of the brain. Foremost among these is its extensive exploitation of rapid reversible synaptic plasticity and temporal feature binding. Since these features, although first presented a decade and a half ago (von der Malsburg 1981), have not received wide acceptance in the community yet, we have expended great effort to demonstrate the functional superiority of the dynamic link architecture over more conventional neural models by using it to solve a real-world problem, object recognition. We are presenting here our best achievement so far in this venture.

The model presented here is closely related to a more technically oriented system (the "algorithmic system" in contrast to the "dynamical system" described here). It has also been developed in our group and is described in (Lades et al. 1993; Wiskott et al. 1997). Essential features are common to the two systems, among them the use of jets composed of Gabor-based wavelet features, and of dynamic links to establish a mapping between the image domain and individual models.

Our model for object recognition is successful in emulating the performance and operational characteristics of our visual system in some important aspects. As in the biological case, the flexible recognition of new objects can be installed simply by showing them once. Our system works with a type of standard feature detector, wavelets, which dominates much of the early visual cortical areae (Jones & Palmer 1987). The sensitivity of our system to changes in the stimulus, as for instance head rotation and change in facial expression, is strongly correlated with that of human subjects (Kalocsai et al. 1994; this study involved a version of our algorithmic system). And, above all, our model is superior in its object discrimination ability to all biologically motivated models known to us, and is at least one of the top competitors among technical systems for face recognition (in a blind test of face recognition against large galleries, performed by the American Army Research Lab, our algorithmic system came out as one of the top competitors). Moreover, our system goes beyond mere recognition of objects, providing the basis for a detailed back-labeling of the image with interpretations in terms of explicit object or pattern models which are linked to the image by dynamic links and temporal feature binding.

In spite of this success, there are still some difficulties and discrepancies. One concern is processing time. The reorganization of the connectivity matrix between the image domain and the model domain requires that the two domains be covered at least twice by the running blob. The speed of this blob is limited by the time taken by signal transmission between the domains and by the temporal resolution with which signal coincidence can be evaluated by dendritic membranes and rapidly plastic synapses. Assuming a characteristic time of a few milliseconds we estimate that our model would need at least one second to create a synaptic mapping. This is much too long compared to the adult's speed of pattern recognition (Subramaniam et al. 1995). We therefore see our system as a model for processes that require the establishment of mappings between the image and object models. This is often the case whenever the absolute or relative placement of parts within a figure is important, and is very likely to be also required when a model for a new object is to be laid down in memory. The actual inspection times required by subjects in such cases are much longer than those required for mere object recognition and can easily be accommodated by our model. We believe that mere recognition can be speeded up by short-cuts. Potential for this we see in two directions, a reduction of the ambiguity of spatial feature arrangement with the help of trained combination-coding features, and a more efficient way (than our running activity blobs) of installing topographically structured synaptic mappings between the image domain and the model domain. A possible scheme for this would be the switching of whole arrays of synapses with the help of specialized control neurons and presynaptic terminals (Anderson & van Essen 1987).

Another as yet weak point of our model is the internal organization of the model domain and the still semi-manual mode in which models are laid down. It is unrealistic to assume completely disjoint models, for several reasons, not the least of which economy in terms of numbers of neurons required. Also, it is unrealistic to see the recognition process as a competition between the dozens of thousands of objects that an adult human may be able to distinguish. Rather, pattern similarities within large object classes should be exploited to give the recognition process hierarchical structure and to support generalization to new objects with familiar traits. The existence of such hierarchies is well supported by neurological observations (Damasio & Damasio 1992) and is implicit in psychophysical results (Biederman 1987) showing that many objects are recognized as simple arrays of shape primitives which are universally applicable. In a system closely related to the one presented here (von der Malsburg & Reiser 1995), a model domain was dynamically constructed as one comprehensive fusion graph containing as sub-graphs models for different objects, and in fact for different aspects of these objects, with different models sharing many nodes. Further research is required in this direction.

Another limitation of the present system is its inability to deal with alterations of size and orientation of the object image beyond a few percent and beyond a few degrees. For this it would be necessary that the connections between the image domain and the model domain linked also features of different size and orientation. Size and orientation invariance has been successfully demonstrated in the context of the algorithmic system (Buhmann et al. 1990; Lades 1995). Direct implementation in the present model would, however, make the DLM process slower and much more difficult or perhaps even impossible, because the system would have to start with a connectivity matrix with many more non-zero entries. The problem may have to be solved with the help of a two-step DLM process, the first step installing an expectation as to size and orientation of the image, specializing the dynamic links accordingly, the second step organizing the match as described here. In many cases, estimates of size and orientation of an object's image can be derived from available cues, one of which being the object's outline as found by a segmentation mechanism.

In the set of simulations presented here we simplified the recognition problem by presenting the objects to be recognized against a homogeneous background. More difficult scenes may require separate segmentation mechanisms which first identify an image region or regions as candidates for recognition (although a version of the algorithmic system was able to recognize known objects in spite of a dense background of other objects and of partial occlusion (Wiskott & von der Malsburg 1993)). Our model is ideally suited to implement image segmentation mechanisms based on temporal feature binding, as proposed in (von der Malsburg 1981), implemented in (von der Malsburg & Buhmann 1992; Vorbrüggen 1995) and supported by experimental data as reviewed in (König & Engel 1995). According to that idea, all neurons activated by a given object synchronize their temporally structured signals to express the fact that they are part of one segment. This coherent signal, suitably identified with our attention variable a_i^p, equation (5), could focus the recognition process on segments.

In summary, we feel that in spite of some remaining difficulties and discrepancies we may have, with our model, a foot in the door to understanding important functional aspects of the human visual system. The environment provided by NSL has proved to be of great help in the development of our system, and we are extremely pleased that with NSL's help we can share our system with students and research groups, both for didactic purposes and as a cutting-edge research tool.

Notes

1. This work has been funded by grants from the German Federal Ministry of Science and Technology (413-5839-01 IN 101 B/9), from AFOSR (F49620-93-1-0109), from the EU (ERBCHRX-CT-930097), and a grant by the Human Frontier Science Program.
2. A. Weitzenfeld developed the NSL3.0 version from the original NSL2.1 model implementation written by L. Wiskott and he contributed Section 18.3 to this chapter.
3. The DLM model was implemented and tested under NSLC.