Technical Report

Dept. for Pattern Recognition and Image Processing
Institute for Automation
Technical University of Vienna
Treitlstr. 3/1832
A-1040 Vienna AUSTRIA
Phone: +43 (222) 58801-8161
Fax: +43 (222) 569697
E-mail: bis@prip.tuwien.ac.at

PRIP-TR-7 June 24, 1993

Neural Networks versus Image Pyramids

Horst Bischof, Walter G. Kropatsch

Abstract

Neural networks and image pyramids are massively parallel processing structures. In this paper we exploit the similarities as well as the differences between these structures. The general goal is to exchange knowledge between these two fields. After introducing the basic concepts of neural networks and image pyramids we give a translation table of the vocabulary used in image pyramids and those used in neural networks. In the following sections we compare neural networks and image pyramids in detail. We show how a modified Hopfield network can be used for irregular decimation. We examine the type of knowledge stored and the processing performed by pyramids and neural networks. In the case of numerical information, so called "numerical pyramids" are rather similar to neural networks. But also for "symbolic pyramids" we show how to implement them by neural networks. In particular we present a neural implementation of the $2 \times 2/2$ curve pyramid. We derive some general rules for implementing symbolic pyramids by neural networks. Finally we briefly discuss the role of learning in image pyramids.

1 Introduction

Artificial neural networks (henceforth called neural networks) are characterized by massive parallelism and the ability of learning. Their features make neural networks interesting for pattern recognition and computer vision.

Though neural networks have success in many domains e.g. [51, 47, 5, 45], for complex problems, such as vision, the current approach of using fully connected (three-layer) neural networks has severe deficiencies. As Le Cun [35] has stated "Expecting good performance without any a priori knowledge, releying exclusively on learning is wishful thinking". One way to incorporate a priori knowledge is to specify a proper topology of the network. The question we are concerened in this report, is which neural network topology is suited for computer vision.

Image pyramids have shown to be an efficient data and processing structure for digital images in a variety of vision applications [36, 49]. Therefore we would like to exploit the similarities as well as the differences of neural networks and image pyramids. The general goal of this report is to exchange knowledge between the fields of neural networks and image pyramids. We would like to emphazise that though both types are massive parallel processing structures, there has been no attempt to view these systems in a common framework.

The structure of this report is as follows: In sections 2 and 3 we briefly introduce neural networks and image pyramids in order to define the basic concepts. We give a translation table of the vocabulary used in image pyramids and those used in neural networks. In the following sections we compare neural networks and image pyramids in detail. In section 4 we consider the structure of pyramids (regular and irregular). We derive a Hopfield network that is able to decimate a level of an irregular pyramid. In section 5 we examine the types of information stored in the cells of a pyramid (numeric and symbolic). And in section 6 we examine the processing done by the cells. In particular, networks simulating the curve reduction process of the $2 \times 2/2$ curve pyramid are presented. Finally we give some conclusions and an outlook for further research. Especially the role of learning will be discussed.

2 Neural Networks

There are a variety of different neural network models (e.g. [52, 12]), but there is a common structure to all of them which we describe in this section. A more detailed model can be found in [1]. A neural network consists of a number of highly interconnected processing elements called *units*. The network is characterized by the interconnection scheme, which we call the *topology*, the types of *units*, the processing performed by them, and the *weights* of the connections.

2.1 Network Topology

The topology of the network can be described by a directed Graph $G = \langle U, E \rangle$; U is the set of units and $E \subseteq U \times U$ is the set of arcs between the units. The set of units U is partitioned into three subsets I, H, O (corresponding to input, hidden and output units) such that $I \cup H \cup O = U, (I \cup O) \cap H = \{\}, I, O \neq \{\}$. We call a network *fully connected* if $E = U \times U$. Each arc $\langle i, j \rangle \in E$ has a *weight* $w_{ij} \in \Re$. We call an arc with the associated weight a *connection.*

Some common topologies are:

- *2–layer network (perceptron)*: The network has only input and output units ($H = \{\}$). Arcs exist only between input and output units i.e. $E \subseteq (I \times O)$.
- *Multi-layer network (multi-layer perceptron, feed-forward network)*: The hidden units are arranged in layers $H = H_1 \cup \cdots \cup H_n$ and $H_i \cap H_j = \{\}$ if $i \neq j$. Arcs exist only between adjacent layers i.e. $E \subseteq (I \times H_1) \cup (H_1 \times H_2) \times \cdots \times (H_n \times O)$. If $E = (I \times H_1) \cup (H_1 \times H_2) \times \cdots \times (H_n \times O)$, such a network is called *fully connected feed forward network.*
- *Fully connected network (Hopfield network)*: There exist arcs between all units in the network, i.e. $E = U \times U$. Sometimes self-connections are not allowed, i.e. $E = U \times U - \{\langle i, i \rangle | \ i \in U\}$.
- *Recurrent network*: The Graph G has cycles; This means that there exists at least one path with $\{\langle i_1, i_2 \rangle, \langle i_2, i_3 \rangle, ..., \langle i_n, i_1 \rangle\} \subseteq E$.

2.2 Units

Processing of information occurs in the units. A unit i has a *state vector* $s_i \in S \subset \Re^N$ which describes the internal state of the unit and an *output value* $o_i \in \Re$ which is sent to other units. The main task of a unit is to compute a new state and a new output value, using the incoming signals (output values of the connected units), the weights of the connections and the own state vector. Formally this process can be stated as the application of an *update function* f_i . Let us call the application of the function f_i, *update* of unit i.

The other task of the unit is to change the weight vector in order to adapt its behavior. Formally this is stated in a learning function l_i. Note that for the update and the learning function only information locally available at the unit is used. The units can therefore operate in parallel and independent of one another.

2.3 Representation

It is important to distinguish between two kinds of representation in a neural network. One is called *local* representation and the other is called *distributed* representation. The information to be represented in a neural network can be manyfold, e.g. characters of an

alphabet, greyvalues of pixels, etc.. Let us call the piece of information we are interested in an *item*. We call a representation local if an item is represented by one unit i.e. this unit is activated when the item is present and not activated when the item is not present. In the case of a distributed representation we have a group of units which represent one item, i.e. one item is represented by many units and each unit participates in representing many items.

A local representation is usually much easier to interpret, but a distributed representation is often more economical in terms of units and more robust against noise [22].

3 Image Pyramids

Image pyramids have shown to be efficient data and processing structures for digital images in a variety of vision applications. An image pyramid is a stack of images with exponentially decreasing resolutions [55]. The bottom level of the pyramid is the original image. In the simplest case each successive level of the pyramid is obtained from the previous level by a filtering operation followed by a sampling operator [17]. More general functions can be used to yield the desired reduction. We therefore call them *reduction functions*.

Many image processing algorithms run on this hierarchical structure in $O(\log n)$ parallel processing steps (n is the image diameter), whereas they need $O(n^2)$ steps without the use of pyramids. Image pyramids are closely related to the concept of scale space [60], in which the scale is introduced as an additional continuous dimension (e.g. by smoothing). A pyramid is then a logarithmically sampled version of scale space. Also wavelet transformations are closely related to image pyramids. For more details on these similarities see [32].

There are three important properties that characterize a pyramid:

1. Structure: e.g. neighbors, father–son relations between levels
2. Contents of a cell: e.g. pixel, edge, or more
3. Processing performed by the cells: e.g. filtering

We will now discuss these properties briefly with respect to image pyramids. In the subsequent sections of this paper we will analyze these properties in more detail with respect to neural networks.

3.1 Structure

The structure of a pyramid is determined by the neighbor relations within the levels of the pyramid and by the father–son relations between adjacent levels. We distinguish between

- regular structures and
- irregular structures

depending on whether the structural relations are the same for all pyramid cells (except on the boundary) or whether they may vary from cell to cell.

3.1.1 Regular Pyramids

Two terms describe the structure of a regular pyramid: *reduction factor* and the *reduction window*. The reduction factor r determines the rate by which the number of cells decrease from level to level. The reduction window (typically a square $n \times n$) associates to every cell in a higher level (called father) a set of cells in the level directly below (called sons). The cells which are neighbors on the same level are called brothers (sisters). The usual notation for describing the structure of a regular pyramid is $n \times n/r$. For example in the classical $2 \times 2/4$ pyramid a window of 2×2 cells forms a new cell of the next lower resolution. Since there is no overlap in this pyramid the number of cells decrease from level to level by a factor of 4.

3.1.2 Irregular pyramids

In irregular pyramids the regularity constraint of regular pyramids is relaxed. These pyramids operate on a general graph structure instead of the regular neighborhood graph as in the case of regular pyramids. There are two ways to construct an irregular pyramid:

1. Parallel graph contraction [50]
2. Decimation of the neighborhood graph [39]

The main purpose for the introduction of irregular pyramids was the rigid behavior (e.g. shift variance) of regular structures [6]. Irregular pyramids offer greater flexiblility [41] for the price of less efficient access.

3.2 Contents of a cell

One can consider the contents of a pyramidal cell as a model of the region which it represents. In the simplest case a cell stores only one (grey) value. We call such pyramids *grey level* pyramids. In more complicated cases several parameters of general models are stored in a cell [19]. But the basic property that numerical values are stored in a cell remains. Subsequently we will call these pyramids *numerical pyramids*.

Besides numerical values it is also possible to store symbolic information in a cell [29]. In this case we have a finite number of symbols, and a cell stores these symbols or relations among them. We call such a pyramid *symbolic pyramid*.

3.3 Processing by a pyramid cell

The main property of processing in a pyramid is that it occurs only local, i.e. every cell computes from the contents of the sons, the brothers, and/or the parents a new value and transmits it to one or more cells of its pyramidal neighborhood. In the bottom–up construction phase input comes from the sons but for some algorithms the flow of information is also in the top–down direction [15].

Table 1: Translation table

Image Pyramids	Neural Networks
cell	unit
level	layer
structure	topology
contents of cell	activation of unit
bottom-up reduction	activation function
parameters of reduction function	weight of arc

The type of operations performed by the cells depends of course on the type of the cell's contents. For grey-level pyramids linear filters e.g. Gaussian are commonly used. But also other non–linear filters have some interesting properties, e.g. minimum and maximum filter or filters based on mathematical morphology. In the case of symbolic pyramids other types of reduction functions have to be used. For example [29] introduced curve relations and a reduction algorithm based on the transitive closure of curve relations. In general a finite state machine [24] may be used to perform a symbolic reduction.

3.4 Translation of terminology

We have introduced the basic concepts of pyramids and neural networks. Since the two research areas have introduced a different terminology we summarize in Table 1 the equivalent notions of the important concepts in these two fields. This should help one being familiar in one of the two fields to translate his knowledge in the other field. In the sequel we will use the vocabulary for image pyramids when talking about image pyramids and that of neural networks when talking about neural networks. But one should keep in mind that the words can be often used interchangeably.

4 Structure

The structure of regular and irregular pyramids can be described by horizontal and vertical graphs. Each level i of a pyramid can be described by a neighborhood graph $G_i = \langle V_i, A_i \rangle$. Where the set of vertices V_i corresponds to the pixels of level i, and $A_i \subseteq V_i \times V_i$ are the neighborhood relations of the pixels. Two vertices $p, q \in V_i$ are connected in G_i if they are neighbors in the structure.

Definition 1
The neighborhood of vertex $p \in V_i$ is defined by $\Gamma(p) := \{p\} \cup \{q \in V_i | (p,q) \in A_i\}$.

The structure is *regular* if a well defined neighborhood relation holds for all vertices (except for the boundary).

The vertical structure (i.e. the connectivity between the levels) can also be described by a (bipartite) graph: $R_i = \langle (V_i \cup V_{i+1}), L_i \subseteq (V_i \times V_{i+1}) \rangle$. The receptive field (i.e. the set of all sons) of a cell $q \in V_{i+1}$ is defined as: $RF(q) := \{p | \langle p, q \rangle \in L_i\}$. In a similar manner the projective field of a cell $p \in V_i$ (i.e. the set of all fathers) can be defined $PF(p) := \{q | \langle p, q \rangle \in L_i\}$.

Any pyramid with n levels can be described by n neighborhood graphs and n–1 vertical graphs. In the case of regular pyramids we need not store all these graphs because information is given implicitely by the the term $n \times n/r$.

From these considerations it is clear that one can built for any pyramid structure an equivalent neural network topology. Therefore all knowledge about the structure of pyramids can be transfered to neural networks. Also the results of shift variance of regular pyramids [6] hold for equivalent neural networks. Indeed we were able to proof that any rigid locally connected neural network structure has shift variance problems [4].

4.1 Irregular Pyramids

In this section we will show how to use neural networks for construction of irregular pyramids by decimation. Decimation divides the cells in a pyramid level into two categories: cells that survive form the cells of the next level and cells that do not appear at reduced levels (non survivor). Peter Meer [39] has given two rules which should be fullfilled by the decimation process. His rules are:

Definition 2

1. *Two neighbors at level i cannot survive both;*

2. *a non survivor must be a neighbor of a survivor.*

We call a decimation which satisfies these rules a *valid decimation.* In [39] it was shown how a decimation can be computed in parallel by a stochastic algorithm. It is also worth noting that the rules 1 and 2 are equivalent to saying that the vertices V_{i+1} of the Graph G_{i+1} on level $i+1$ define a maximum independet (vertex) set (MIS) of the graph $G_i = \langle V_i, A_i \rangle$.

The algorithm for stochastic decimation proceeds in following major steps (for more details see [34]):

1. Assign uniformly distributed random numbers to the cells.

2. Select local maxima as surviving cells.

3. Fill holes, i.e. repeat step 2 as long as there are non-surviving cells which have no surviving neighbor.

4. Every non surviving cell selects a father. This construction also defines the receptive fields.

5. Construct the neighborhood graph of the new level; Two surviving vertices become neighbors if they have vertices in their receptive fields which are neighbors in the level below.

6. Repeat steps 1 - 6 with new level until only a single vertex is in the receptive field.

This basic algorithm can be modified in order to take into account the contents of a cell ending up with an adaptive pyramid [27]. This has been used for image segmentation or connected component analysis in logarithmic time complexity.

4.2 Decimation by Hopfield networks

In the following we will show that we can replace the steps 1 – 3 with a modified Hopfield network which works on the neighborhood graph $G = \langle V, A \rangle$[1].Moreover we show that the formulation as a Hopfield network is more general than the stochastic decimation, and it naturally includes the concept of the adaptive pyramid.

Let us introduce the notion of a survival state of a cell:

Definition 3 *The survival state of a cell $p \in V$ is a function*
$s : V \mapsto \{0, 1\}$ *with* $s(p) = \begin{cases} 1 & \text{if cell } p \text{ survives} \\ 0 & \text{otherwise} \end{cases}$

Let us further introduce the following energy function:

$$E = \sum_{\langle i,j \rangle \in A} s(i)s(j) - \sum_{k \in V} s(k) \tag{1}$$

Now the following theorem holds:

Theorem 1 *The energy function E from eq. (1) obtains a local minimum, E_{min}, if and only if the assignment of surviving and non–surviving cells, $s(p)$ is a valid decimation (i.e. satisfying the rules 1 and 2 of definition 2) or, equivalently, forms a maximum independent vertex set of G.*

Proof.
(a) E_{min} is a local minimum of $E \Rightarrow s(p)$ is a valid Decimation.
Assume $E = E_{min}$ is a local minimum but $\{p \in V | s(p) = 1\}$ is not a valid decimation, then at least one of the rules 1 or 2 must be violated.

Case a.1: rule 1 does not hold:

$\Rightarrow \exists\ p, q \in V$ such that $\langle p, q \rangle \in A$ and (by Def. 3) $s(p) = s(q) = 1$. Changing $s(p)$ to 0 can affect only those terms in equation (1) where $s(p)$ occurs. We can write E as

$$E = \sum_{\langle p,n \rangle \in A} s(p)s(n) + \sum_{\langle n,p \rangle \in A} s(n)s(p) + \sum_{\substack{\langle i,j \rangle \in A \\ i,j \neq p}} s(i)s(j) - s(p) - \sum_{k \neq p \in V} s(k)$$

[1] We skip the subindices for level i because the algorithm works only on one level

because E is symmetric. The fourth term increases the energy by 1 if we change $s(p)$ to 0. But the first and second term decreases the energy at least by 2 because $s(p)s(q)$ changes from 1 to 0 in both terms. Following inequality holds: $E(s(p) = 0, s(q) = 1) \leq E_{min} + 1 - 2 = E_{min} - 1 < E_{min}$ and this is a contradiction that E_{min} is a local minimum.

Case a.2: rule 2 does not hold

$\Rightarrow \exists p \in V$ such that $s(p) = 0$ and $\forall q \in \Gamma(p) : \ s(q) = 0$. Again changing $s(p)$ to one yields $E(s(p) = 1) = E_{min} - 1 < E_{min}$ and this is a contradiction to the fact that E_{min} is a local minimum.

(b) $\{p|s(p) = 1\}$ defines a valid Decimation $\Rightarrow E = E_{min}$ is a local minimum

We have to show that by changing only one state of a cell $r \in V$ we get a higher energy value.

b.1: $s(r) = 0 \Rightarrow \exists q \in \Gamma(r) : \ s(q) = 1 \Rightarrow E(s(r) = 1) = E_{min} - 1 + 2 = E_{min} + 1 > E_{min}$

b.2: $s(r) = 1 \Rightarrow \forall q \in \Gamma(p) : \ s(q) = 0 \Rightarrow E(s(r) = 0) = E_{min} + 1 > E_{min}$

From b.1 and b.2 we conclude that E_{min} is a local minimum of E. qed.

Given the energy function in eq.(1) we can now define a Hopfield network operating on the neighborhood graph which minimizes this energy function. In [25, 26] Hopfield has described a network of fully connected units operating asynchroniously which is governed by the following energy function:

$$E = -\frac{1}{2} \sum_i \sum_j w_{ij} s(i) s(j) - \sum_i I_i s(i) + \sum_i U_i s(i) \tag{2}$$

where $w_{ij} \in \Re$ is the weight between unit i and j, I_i is the external Input and U_i is the threshold of unit i.

Hopfield proved that if the weights are symmetric (i.e. $w_{ij} = w_{ji}$) and the units update asynchroniously, the network will settle in a local minimum of E. If we now set in equation 2

$$w_{ij} = \begin{cases} -2 & \langle i, j \rangle \in A \\ 0 & \text{otherwise} \end{cases}$$

and $I_i = 1$ and $U_i = 0$ for all i we get equation 1.

The resulting Hopfield network operates on the neighborhood graph G and computes valid decimations (according to definition 2). The update procedure of the cells is as follows:

$$s(p) = \begin{cases} 1 & \text{if } 1 - 2 \sum\limits_{q, \langle q,p \rangle \in A} s(q) > 0 \\ 0 & \text{otherwise} \end{cases}$$

The initial state of the network can be choosen at random.Having established this relationship we can now apply all the theory available for Hopfield networks. For example the convergence time is of interest because in the algorithm of Meer [39] the convergence takes $O(|V|)$ steps in the worst case (when the random number generator produces a ramp function on every iteration). In [58] it was proven that a Hopfield network with negative

weights (if weights are 0 the connection is not present) and a sequential update algorithm (i.e. at a time only one unit is updated) converges in the worst case in $2|V|$ steps.

But one should note that by a parallel update scheme and a connectivity graph which is far away from full connectivity one could design a parallel algorithm which converges much faster, because all units which are not connected can be updated in parallel without altering the convergence properties.

4.3 Adaptive Pyramids

The energy function in eq.(2) has many parameters which can be changed in order to influence the decimation: The weights w_{ij} between two cells express the constraint on the states of these cells. If w_{ij} is negative these two cells should not be both on, on the other hand a positive weight forces the two cells to be both on. One should note as long as $w_{ij} > -I_i$ the behavior of the network is not changed. This can be seen easily from the proof of Theorem 1. The external input I_i (and the threshold U_i) can force a single cell to survive or not if set properly (e.g. if $I_i > \sum_{j \in \Gamma(i)} |w_{ji}|$ the cell i survives).

We can use these observations to build adaptive pyramids [41, 27]. To determine the connected components of a binary image we can set the weights according to

$$w_{ij} = \begin{cases} -2 & i \text{ and } j \text{ are neighbors and they have the same grey value} \\ 0 & \text{otherwise} \end{cases}$$

In this case the survivors and non-survivors are computed only within a homogeneous region (with identical pixel values). This procedure is equivalent to Montanvert's algorithm [41], which applied the stochastic decimation only within a homogeneous region.

This scheme can be easily generalized to grey-level image segmentation where the weights of the corresponding Hopfield network are set according to the difference in greyvalue of the pixels;

i.e. $w_{ij} = -f(d_{ij})$ and $d_{ij} = |g_i - g_j|$

where g_i and g_j are the greyvalues of pixel i and j, and f is a suitable function, e.g. linear, logarithmic or some step function as shown in Fig. 1.

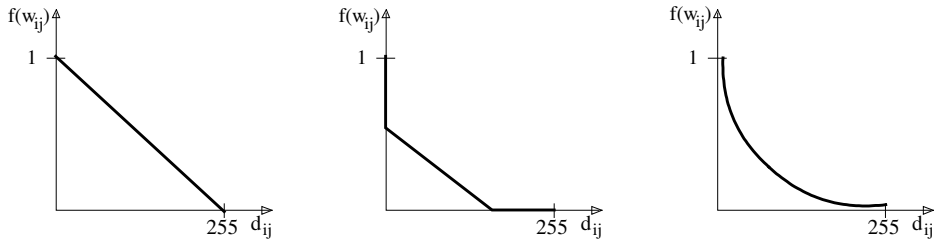


Figure 1: Different functions for setting the weights of a Hopfield network

From these considerations one can see that the decimation by using Hopfield networks has the advantage that it naturally includes the concept of the adaptive pyramid.

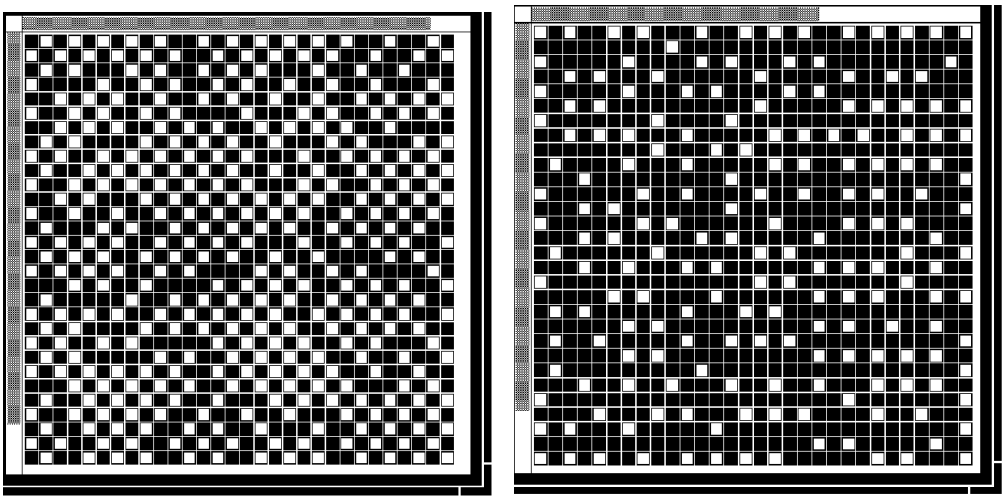
4.4 Experiments

In order to demonstrate the theoretical results we have performed several experiments. For the simulations we have used a small image with 30×30 pixels. We show the decimation only for the base level of the pyramid.

Fig. 2 depictes a decimation for a 4-connected (left) and an 8-connected (right) neighborhood graph. The decimations are all valid, as is expected from Theorem 1. In the case of 4-neighborhood one can see that regular patterns, which are the dominant structure, are broken by line-like structures. This resembles somehow the structure of the human retina, where one can see similar patterns [33]. The network converges on the average in 8 asynchronous update steps. The maximum number during our simulations (100 runs) were 12 update steps.

Fig. 3 shows the decimation for a binary image with 8-neighborhood where we have used a line from left to right at rows 13-15. The weights of the units on the line to the units of the background are set to zero. The Hopfield network treats the components of the image independently, i.e. the line and the background are decimated independent of one another.

These expermints have shown that the decimation by Hopfield networks is an alternative to the algorithm of Meer. In the future we will integrate this decimation in a pyramidal algorithm, and build whole pyramids. The main goal is image segmentation. An interesting question in this respect is the difference between the segmentations of our algorithm to the one of Montanvert [41].



(a) (b)

Figure 2: Decimations of graphs with 4-neighborhood (a) and 8-neighborhood (b)

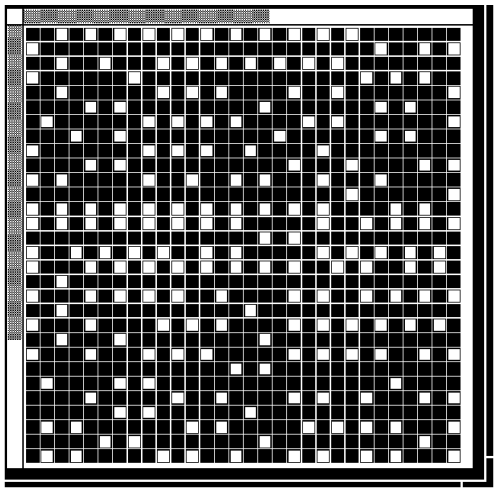


Figure 3: Decimation of a binary image

5 Contents

We have seen in section 3.2 that we can distinguish pyramids with numeric and symbolic contents in the cells. Whereas neural networks store only numeric information in the units. Therefore cells of a numeric pyramid are from the viewpoint of contents rather similar. We will show in section 6 that also the processing performed by the cells and units respectively is not very different. The case of symbolic information needs further investigation.

5.1 Symbolic Content of a cell

Let $\Sigma = \{\sigma_1, \sigma_2, \cdots \sigma_N\}$ be a finite set of symbols, and $R \subseteq \Sigma \times \Sigma$ a binary relation between these symbols. A cell in a symbolic pyramid stores symbols and/or relations between these symbols. Our main concern in this section is on how we can represent this information by a neural network. We are therefore confronted with a representation problem.

Since we have a finite number of symbols (and relations) we can code a symbol by the activation of a unit e.g. $\sigma_1 = 0, \sigma_2 = 0.1$ etc. This representation is very "unnatural" for neural networks and we would give up some essential characteristics of neural networks. We therefore have to use several units to store symbolic information. The general idea is to replace a cell of the pyramid by a small neural network. We have two possibilities:

1. local representation
2. distributed representation

5.1.1 Local Representation

If we have N symbols we need N units to represent these symbols. Every symbol has a designated unit which becomes active when the symbol is present. If we have to represent relations among the symbols we need $O(N^2)$ units to represent all possible relations. When only one symmetric relation at a time needs to be stored N units are sufficient (i.e. if both unit σ_1 and unit σ_2 are active the relation $\sigma_1 R \sigma_2$ is represented). But if more relations need to be stored we are facing the so called binding problem [13]. We need therefore a designated unit for each relation $\sigma_i R \sigma_j$ which is active when the relation is present. The problem even becomes more severe when several different relations or n-ary relations need to be represented.

5.1.2 Distributed Representation

Distributed representations are more economical in terms of units and are also more robust than local representations. The general idea is that more than one unit is active when representing one item and a single unit participates in representing more than one item. Many possibilities for distributed representation have been proposed [22, 16, 46, 44, 9, 10]. Many of them are suited for representing numeric information [16, 9], others have been proposed for symbolic information [46, 44].

Recently Pollack [46] has introduced the concept of "Recursive Distributed Representations". He has shown that he can represent binary trees and arbitrary list structures in this framework. The general idea is very simple. An autoassociative network like the one in Fig. 4a is used.

This network is trained to reproduce the input at the output units through a narrow channel of hidden units (half the number of hidden units than input units). The activation of the hidden units is then the distributed representation. This network can now be used iteratively to produce complicated data structures.

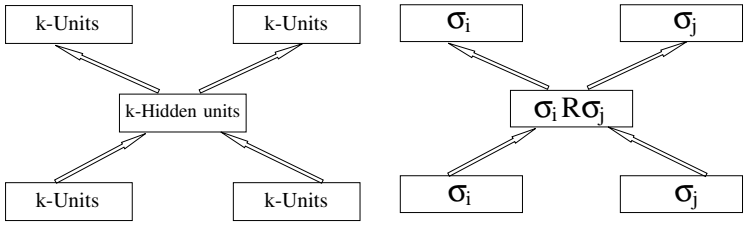


Figure 4: (a) Autoassociative Network used by Pollack to encode symbolic structures (b) encoding curve relations

For our purpose we can use this scheme to represent relations with only $O(N)$ units (N is the number of symbols to represent). Assume we have a local representation of the individual symbols. We can now represent the relation $\sigma_i R \sigma_j$ with N units like in Fig. 4b. Since for $N \neq 3$, $2^N \geq N^2$ we can represent all relations with this scheme and have also the possibility to convert it back into a local representation. Moreover we can use constraints on the weights for special relations, e.g. if the weights from the first set of input units are identical to the second set of input units the representation of the relation is symmetric i.e. $\sigma_i R \sigma_j = \sigma_j R \sigma_i$.

We have performed a simple experiment which shows the distributed code which is developed in the hidden units. We use 4 relations, represented locally at the input and output units. The network has 8 input units, 4 hidden units, and 8 output units. The weights of the inputs units $1-4$ to the hidden units are constrained to be identical to those of the input units $5-8$ to the hidden units (e.g. $w_{11} = w_{51}$). The network is trained with back-propagation [52] to autoassociate the input with the output. This taskes about 100 presentations of training patterns till the network has converged. In table 2 we show the activations of the hidden and output units for certain inputs. One can see that the network has completely learned the task and that the hidden units have developed a distributed

Table 2: Distributed representation of relations developed by an autoassociative network

input	hidden	output
0 0 0 1 0 0 0 1	1 0 0 1	0 0 0 1 0 0 0 1
0 0 1 0 0 0 0 1	1 0.15 0 0.2	0 0 1 0 0 0 0 1
0 1 0 0 0 0 0 1	0 0 0 1	0 1 0 0 0 0 0 1
1 0 0 0 0 0 0 1	1 0 1 1	1 0 0 0 0 0 0 1
....		
1 0 0 0 1 0 0 0	1 0 1 1	1 0 0 0 1 0 0 0

representation of the relation. We will discuss in section 6.2 how this code can be used to process symbolic information.

6 Processing

The result of a reduction function of a pyramid depends on the contents of the cell, there is no use of performing symbolic reduction on grey-level pyramids and vice versa. We will therefore consider processing of numerical and symbolic information separately.

6.1 Numeric Information

Three types of filters are commonly used as reduction functions in numeric pyramids:

1. Linear filters
2. Non-linear filters
3. Morphological filters

6.1.1 Linear reduction functions

The most commonly used reduction function in a pyramid is a convolution. This operation is also used by units in neural networks (though some other models exist [11, 8]), when computing the weighted sum of inputs and connection weights.

In pyramids all cells usually perform the same reduction function (same kernel) whereas in neural networks different units have different weights. But there are some models (e.g. weight sharing [35]) which force units to develop the same weights.

Another differnce is that after computing the weighted sum, units usally use a nonlinearity (typically a sigmoid function $f(x) = \frac{1}{(1+e^{-x})}$), before sending their output to other units. In image processing terms, the application of this activation function corresponds to

a non-linear contrast enhancement. Let us call such pyramids which use a sigmoid contrast stretch *sigmoid pyramids*.

In order to study the effect of this operation for pyramids we have conducted a set of simple experiments. We have used a $n \times n/4$ Gaussian pyramid allowing only integer values in the range $[0 \ldots 255]$ for the greyvalues. Since we want to process greyvalues in the range $[0 \ldots 255]$ we have to adapt the sigmoid to that range. In particular we have used following sigmoid function:

$$f(x) = \frac{255}{1 + e^{-\frac{14x}{255}+7}} \tag{3}$$

In oder to control the steepness of the function an additional parameter β is introduced:

$$f(x) = \frac{255a(\beta)}{1 + e^{\beta(-\frac{14x}{255}+7)}} - b(\beta) \tag{4}$$

$a(\beta)$ and $b(\beta)$ are choosen such that $f(0) \sim 0$ and $f(255) \sim 255$, i.e.: $a(\beta) = \frac{1+e^{7\beta}}{e^{7\beta}-1}$ and $b(\beta) = \frac{255}{e^{7\beta}-1}$

For $0 < \beta < 1$ $f(x)$ is flat and for $\beta > 1$ $f(x)$ gets steeper. In the limit $\beta \to 0$ $f(x)$ approaches a linear function and for $\beta \to \infty$ $f(x)$ approaches a step function. In Figure 5 you can see $f(x)$ for various values of β.

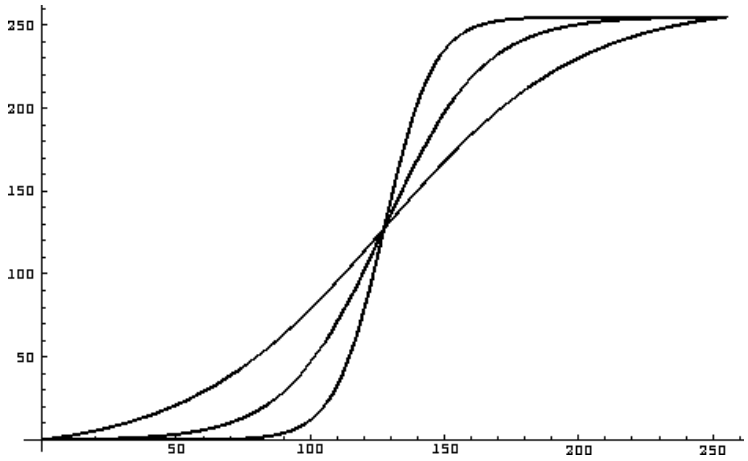


Figure 5: The sigmoid function $f(x)$ for $\beta = 2, 1, 0.5$

The first experiment shows the effect of boundary preservation and noise elimination. We have generated a black test image of size 256×256 containing a white square of size 50×50 (see Fig. 6 (left)), and then we have added uniform random noise in the range $[-100 \ldots 100]$ to this image (see Fig. 6 (right)). We have build 4 levels of a $5 \times 5/4$ Gaussian pyramid on these two images (Fig. 7 for displying purpose the images of the pyramids are

zoomed to be of identical size, no interpolation was used). In the case without noise one can see that the edges get blurred. In the case with noise the noise is gradually decreased from level to level. Then we have constructed the same $5 \times 5/4$ pyramid, but with the use of the sigmoid function, where we set $\beta = 2$ (Fig. 7). From this simple expermiment one can see that the sigmoid pyramid effectively eliminates the noise as well as it preserves the edges of the square.

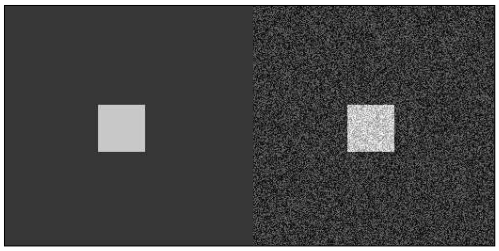


Figure 6: The test image with (right) and without (left) noise

In the second experiment we have used a $3 \times 3/4$ Gaussian pyramid on a real image. Fig. 8 shows the Gaussian and the sigmoid pyramid with $\beta = 0.4$. The Gaussian pyramid produces a blurred version of the image whereas the sigmoid pyramid tends to produce a binary image. This effect can also be seen from the histograms (see Fig. 9). This effect of the sigmoid pyramid is useful for image segmentation and threshold selection. For example based on the histogram of Fig. 9 a threshold for detecting the road can be easily found. This is not the case for the Gaussian pyramid.

A critical parameter of the sigmoid pyramid is β. β controls the steepness of the sigmoid function as well as the "degree" of nonlinearity. If β is choosen too large we have saturation effects; i.e. the whole images gets black or white. The choice of β depends on the image content and the goal of processing. For binary images β can be choosen large, this is also the case when the goal is to segment the image, in other cases β should be choosen rather small. The choice of β can also be controlled by some adaptive procedure similar to adaptive threshold selection [53].

6.1.2 Non-linear filters

We have already seen in the previous section how a linear pyramid can be augmented by nonlinear processing from neural networks. But also other non-linear filters (e.g. minimum or maximum) have been used as reduction functions for pyramids [7]. Local maxima (minima) detection can also be performed by neural networks [43] by a mechanism called lateral inhibition or Winner Takes All (WTA). Most of the proposed schemes have the disadvantage that the value of the maximum is not preserved. Recently Tsotsos [56] has introduced an algorithm which preserves the value of the maximum. This algorithm can be used for building maximum (minimum) pyramids.

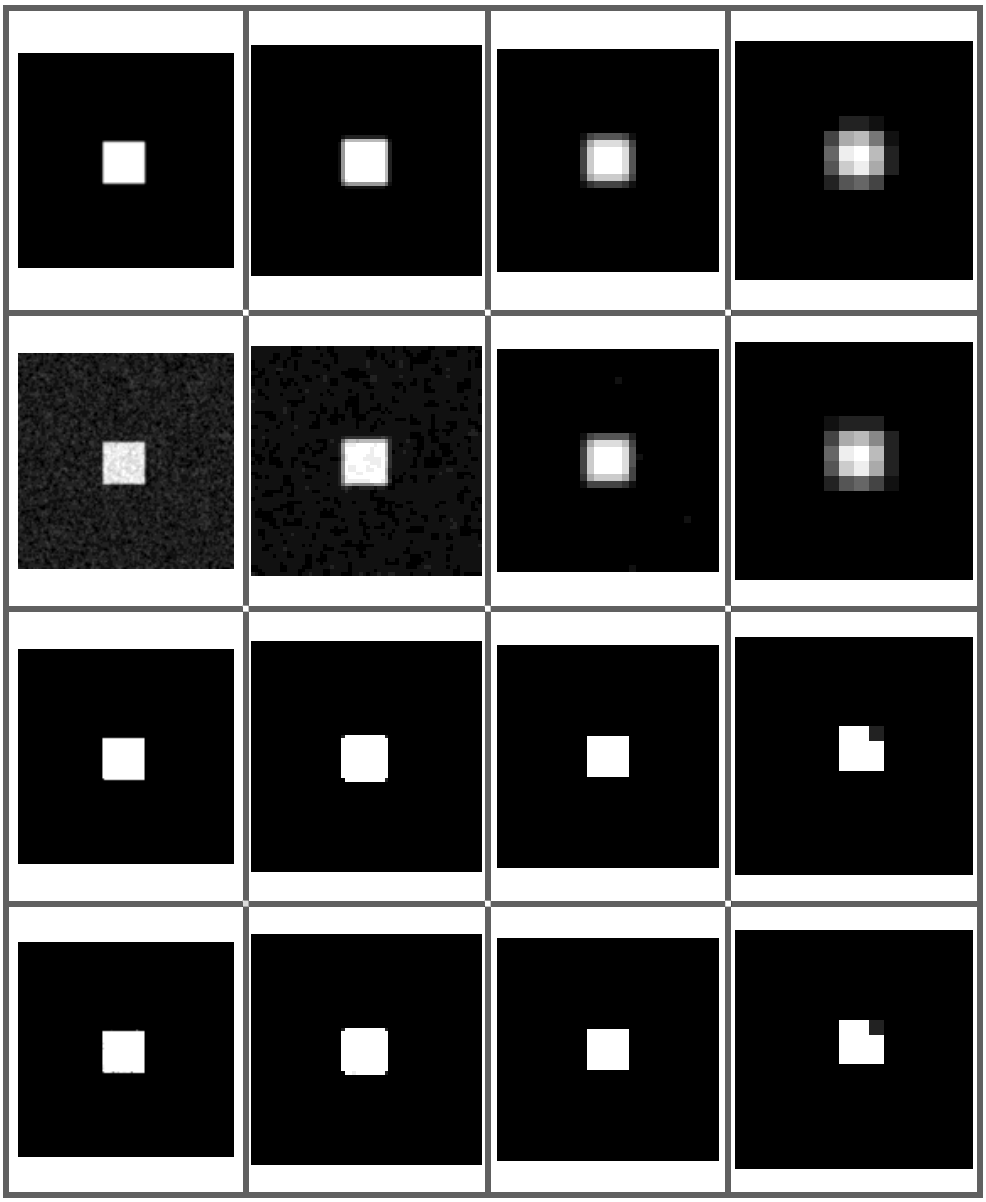


Figure 7: Gaussian (upper two rows) and Sigmoid (lower two rows) Pyramids of the test images

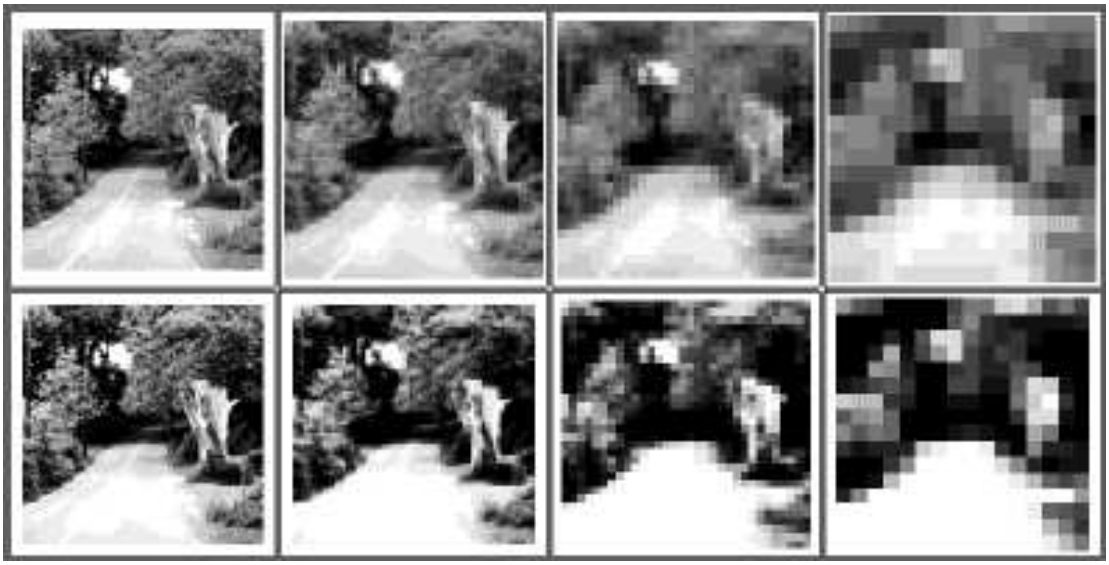


Figure 8: Gaussian and Sigmoid Pyramids on the road image

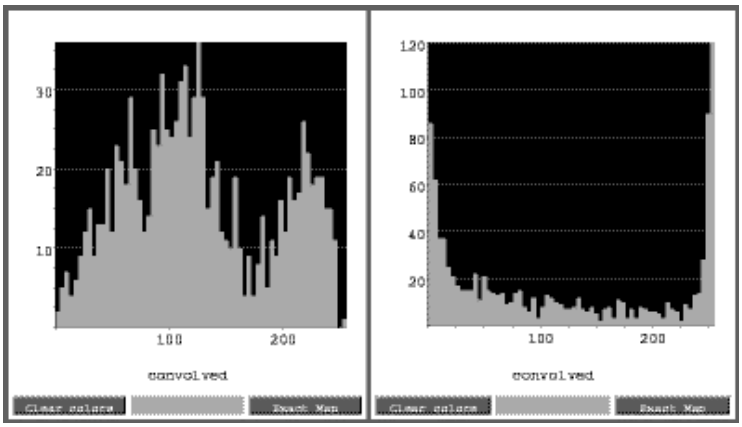


Figure 9: Histograms of level 3 of the Gaussian (left) and Sigmoid Pyramid (right)

6.1.3 Morphological filters

Haralick et.al [18] have introduced the concept of a morphological pyramid, which is built by morphological operations [54]. It has also been shown that certain types of neural networks (iconic neural networks) can implement morphological operations [57]. Morphological neural networks have been proposed which share principles from neural networks and mathematical morphology [48].

6.1.4 Summary

From the considerations of the three previous sections we can conclude that numerical pyramids and neural networks share many similarities. Indeed most operations performed by pyramids can also be implemeneted by neural networks. Taking into account also section 4 (Structure) and 5 (Contents) we can conclude that neural networks can be used to implement any numerical pyramid. Moreover we have seen that pyramids can be augmented by functions from neural networks. These results will give us also a firm basis for studying learning algorithms for pyramids, which will be subject of a further research.

6.2 Symbolic Information

As we have seen in section 5.1, in order to represent symbolic information we have to give up the one-to-one correspondence between cells and units; we have to replace a cell of a pyramid by a small neural network. We will now extend this concept to describe processing of symbolic information by neural networks. We will start by designing a neural network for the $2\times2/2$ curve pyramid [29]. Finally we present some broader view of neural networks within the concept of symbolic pyramids.

6.2.1 $2 \times 2/2$ Curve pyramid

The $2 \times 2/2$ curve pyramid was introduced in [28]. The basic idea is that linear structures of images are represented by curve relations. A cell of the pyramid is considered as an observation window through which the curve is observed. A single curve intersects this window only twice. Only the intersection sides (N, E, S, W) are stored in the cell (i.e. a curve relation). We denote a curve relation by AB, where $A, B \in \{N, E, S, W, F\}$ (F is the special end code when the curve ends in a cell).

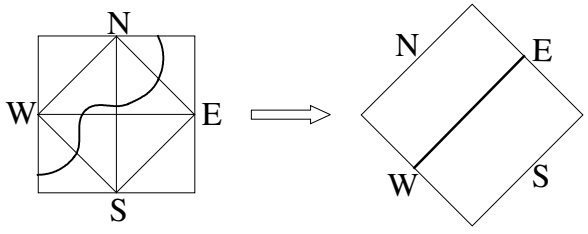


Figure 10: A reduction steps of the $2 \times 2/2$ Curve Pyramid

The basic routines of building the next level of the pyramid are (Fig. 10, Fig. 11):

1. Split - subdivision of the cells contents by introducing a diagonal.
2. Transitive closure - the curve relations of the four son cells are merged by computing the transitive closure of all relations (i.e. $AB, BC \Rightarrow AC$).
3. Merge - the curve relations of the new cell are selected.

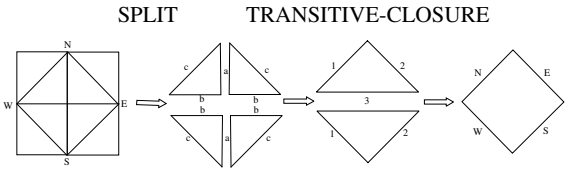


Figure 11: Splitting the Curve Relation by the diagonal

The $2 \times 2/2$ curve pyramid has several interesting properties like the length reduction property, "structural" noise filtering etc. which are described in [30, 31].

6.3 Neural Network curve pyramid

The $2 \times 2/2$ curve pyramid is an ideal test case to study neural network implementations of symbolic pyramids and to illustrate the concept of replacing a cell of a pyramid by a neural network. In order to simplify the discussion we will not describe end codes F and U-turns, but they can be easily included.

The representation of the relation is as follows. Each cell has four output units (N,E,S,W), which have connections to the cells in the next higher level. With this scheme only one relation per cell can be stored unambiguously. In this case exactly two units are activated (i.e. output value = 1) and two units are not activated (i.e. output value = 0). For example if the units N and S are activated a North-South relation is represented. Inside a cell we represent the relations by a local representation as described in section 5.1 (i.e. one cell per possible relation), since we have to represent multiple relations in order to compute the transitive closure.

We have to describe 3 steps in terms of neural networks:

1. Split
2. Transitive Closure
3. Merge

6.3.1 Split

The first operation to perform is to split the curve relations by the diagonal as shown in Fig. 11. This can be done by a neural network like the one in Fig.12.

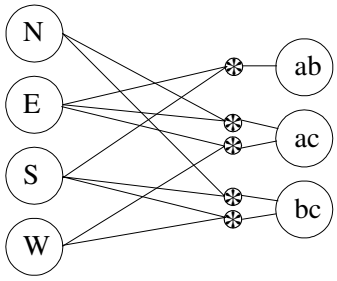


Figure 12: Neural Network Implementation of the splitting operation

The small circles with the star inside indicate multiplicative connections, i.e. only if all inputs are activated a value of one is passed to the units. One should note that three units are sufficient to represent the possible relations in the triangle because the relation is symmetric. The operation depicted in Fig. 12 is done for all four sons of a cell.

6.3.2 Transitive closure

At the next step we have to connect the curve segments in the four triangles by computing the transitive closure of the curve relations. We will first merge the upper and the lower two triangles in parallel, and then merge the resulting two triangles to the final square, like it is shown in Fig 11.

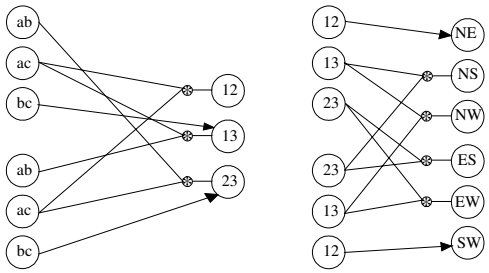


Figure 13: Neural Network Implementation of transitive closure operation. left: for two triangles to another traingle, right: two triangles to a square

The networks for merging two triangles to another triangle and merging two triangles to a square are shown in Fig. 13 (small circles with stars indicate multiplicative connections, small arrows indicate connections with a weight of 1). It is easy to verify that these circuits compute the transitive closure.

6.3.3 Merge

The third operation simply converts the relation code used for processing the transitive closure to the code used by the output units. This can be done by the network shown in Fig. 14.

When putting all the subnetworks together we get a network for performing a reduction step for a single cell (see Fig. 15). One can see that this network is hierarchically structured and consists of many identical subnetworks. In order to handle U-turns and end-codes correctly a few additional units are required. One should note that this network is a designed solution (no learning is required), and performs the same operations as the $2 \times 2/2$ curve pyramid. But it nicely demonstrates that the operations of a symbolic pyramid can be performed by a neural network.

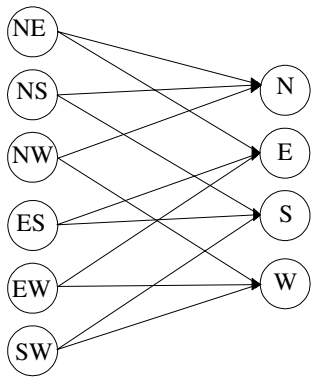


Figure 14: Neural Network for Merging the curve Codes

6.4 Distributed curve pyramid

In section 5.1.2 we have seen how we can learn a distributed code for relations by an autoassociative network. However we need also to process these relations. This cannot be done the same way we did it for a local representation of the relations, where we have designed a network by hand. Therefore we have to employ some learning algorithm. A simple way would be to enumerate all possible combinations of relations, and training of a suitable network by a supervised learning method (e.g. back-propagation). The result would implement the desired reduction function with a distributed representation. With

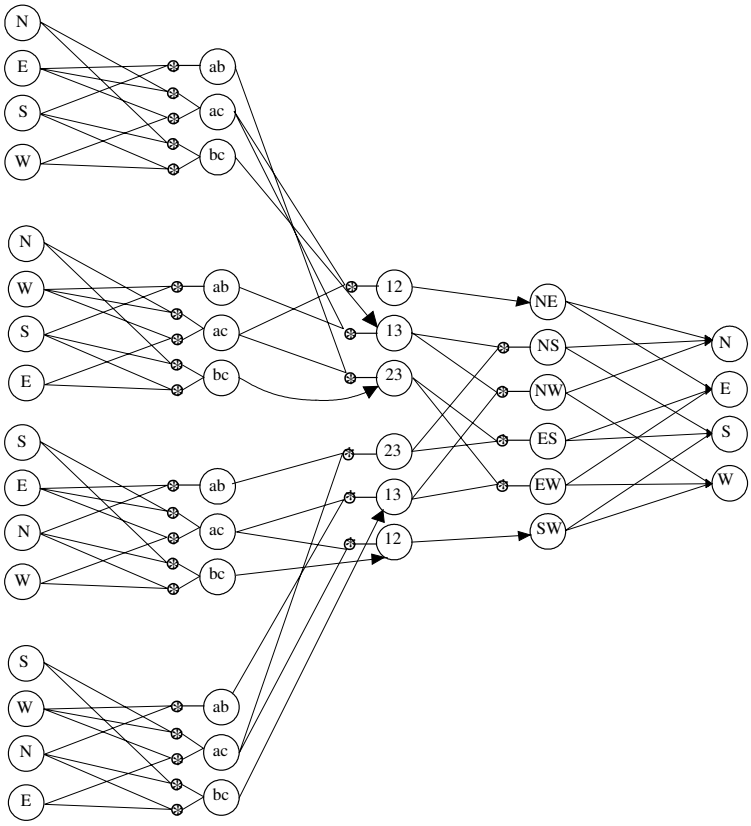


Figure 15: Neural Network for a reduction step in the $2 \times 2/2$ curve pyramid

this approach no new representations (e.g. new curve primitives) will arise.

For these reasons we would like to employ some unsupervised learning method which has the freedom to form also new representations. A possible architecture for this goal is similar to the autoassociative network we have used for learning the distributed curve relations.

However with the unsupervised method we are facing the fundamental problem that the code developed is not necessarily the same as that used as an input. Therefore we have to train all levels of the pyramid and use different reduction functions at these levels. The other problem is that we cannot decode the distributed code into a local one for interpretation purposes.

We hope that these problems listed above can be circumvented by special autoassociative networks where we are employing constraints on the weights, and an iterative training scheme; i.e. start with some distributed code; learn the reduction step; take the newly formed code as input and apply it again to the same network; repeat this process until some code has stabilized. In order to check these ideas several experiments and a theoretical analysis have to be performed.

6.5 General neural network scheme for symbolic pyramids

Though the scheme with the distributed representation should be more general than the network with local representation, it is not general enough for all kinds of symbolic pyramids. Replacing a cell of a pyramid by a small neural network gives us enough freedom to perform more general computations. In order to process symbolic information we need a network which can act as a finite state machine.

In general these small networks can be considered as grouping small image parts to objects (or parts of objects). Certain constraints have to be satisfied by this grouping process (e.g. the parts meet at a certain angle). Since the grouping is done hierarchically the "combinatorial explosion" of checking all possible combinations can be avoided. Moreover we can use learning algorithms to alter this grouping process. From this considerations we can describe the required properties of the networks.

1. *Recurrent network* - each state of the network can be associated with a symbol.
2. *Constraint satisfaction* - the network must converge to a stable state which fullfills as much constraints as possible in order to arrive at a "good" interpretation of the scene.
3. *Distributed representation* - as we have seen in section 5.1 an efficient representation of symbolic structures has to be distributed.
4. *Unsupervised Learning* - if learning should be performed at the small networks it has to be unsupervised in order to avoid the problem of teaching each cell of the pyramid separately [14, 59].

There are some network models which fullfill the properties 1 - 3, e.g. Hopfield network [25, 26] recurrent Jordan network [40], Boltzmann machine [23]. But all these models use a supervised learning scheme. Some research has to be done to find an ideal combination for our purposes. Especially the learning problem is still a major obstacle to perform these tasks.

7 Conclusions and Outlook

The goal of this report was to identify the similarities of image pyramids and neural networks. We have considered the structure of pyramids and the topology of neural networks: the type of information stored in the cells and units; and the kind of processing performed by them. In all of the above cases we have found considerable similarities between neural networks and image pyramids. Though the major goal was to compare existing methods we have also introduced several new ideas and methods:

1. Hopfield networks for constructing irregular pyramids
2. Distributed representations of curve relations

3. Sigmoid pyramids

4. Neural curve pyramid

5. Replacement of a pyramidal cell by a small neural network

As a major open problem we have identified suitable learning algorithms for the architectures intoduced.

Learning has to a great extend been excluded from this report. But one of the main motivations for identifing the similarities between neural networks and image pyramids is to employ learning algorithms on pyramids. An ideal learning algorithm for pyramids has to be unsupervised in order to avoid the problem of long learning times and bad scaling characteristics [21]. A reasonable principle for learning in such pyramidal architectures is the *infomax principle* proposed by Linsker [38, 37]. It states that the cells of each level should preserve as much information as possible. This principle is tightly connected to Hebbian-Learning [20]. We have recently proposed an algorithm [2] based on a modified Hebb-rule of Oja [42], which is able to learn the weights of a numerical pyramid. In a forthcomming paper [3] we will exploit the use of learning algorithms in pyramids in more detail. We would like to emphazize that this extends the capabilities of image pyramids aiming at a common framework for cellular processing structures.

References

[1] Horst Bischof. Modular, hierarchical, and geometrical neural networks. Technical Report PRIP-TR-9, Dept. for Pattern Recognition and Image Processing, TU Wien, 1991.

[2] Horst Bischof. Neural networks and image pyramids. In Horst Bischof and Walter Kropatsch, editors, *Pattern Recognition 1992*, volume 62 of *OCG-Schriftenreihe*, pages 249–260. Oldenbourg, 1992.

[3] Horst Bischof. Learning pyramids. accepted at Neural Networks and Artificial Intelligence at Substance Identification Technologies, 1993.

[4] Horst Bischof and Axel Pinz. The invariance problem for hierarchical neural networks. In Chen Su-Shing, editor, *Neural and Stochastic Methods in Image and Signal Processing*, volume SPIE Vol. 1766, pages 118 – 129. SPIE, 1992.

[5] Horst Bischof, Werner Schneider, and Axel Pinz. Multispectral classification of landsat-images using neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 30(3):482–490, 1992.

[6] M. Bister, J. Cornelis, and Azriel Rosenfeld. A critical view of pyramid segmentation algorithms. *Pattern Recognition Letters*, Vol. 11(No. 9):pp. 605–617, September 1990.

[7] Ronald P. Blanford and Steven L. Tanimoto. Bright spot detection in pyramids. *Computer Vision, Graphics, and Image Processing*, Vol. 43(No. 2):pp.133–149, August 1988.

[8] Giles Lee C. and Maxwell Tom. Learning, invariance, and generalization in high-order neural networks. *Applied Optics*, 26(23):4972–4978, 1987.

[9] Ballard D.H. Interplolation coding: A representation for numbers in neural models. *Biological Cybernetics*, 57:389–402, 1987.

[10] Hinton Geoffrey E. Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, 46:47–75, 1990.

[11] Girosi F. and Poggio T. Networks and the best approximation property. *Biological Cybernetics*, 63:169–176, 1990.

[12] J.A. Feldman. Connectionist models and their properties. *Cognitive Science*, 6:205–254, 1982.

[13] J.A. Fodor and Z.W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71, 1988.

[14] K. Fukushima, S. Miyake, and T. Ito. *Neocognitron*: a neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-13(Nb. 3):pp.826–834, September/October 1983.

[15] William I. Grosky and Ramesh Jain. A pyramid-based approach to segmentation applied to region matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(No.5):pp.639–650, September 1986.

[16] P.J. Hancock. Data representation in neural nets: an empirical study. In *Proc. of the 1988 Connectionist Models Summer School*, pages 11–20. Morgan Kaufmann, 1988.

[17] Robert M. Haralick and Shapiro Linda G. Glossary of computer vision terms. *Pattern Recognition*, 24:pp.69–93, 1991.

[18] Robert M. Haralick, Charlotte Lin, James S. J. Lee, and Xinhua Zhuang. Multiresolution morphology. In *Proceedings of the First International Conference on Computer Vision*, pages 516–520, London, England, June 1987.

[19] R. L. Hartley. *Multi–Scale Models in Image Analysis.* PhD thesis, University of Maryland, Computer Science Center, 1984.

[20] D.O. Hebb. *The organization of behavior.* Wiley, New York, 1949.

[21] G. Hinton. Connectionist Learning Procedures. *Artificial Intelligence*, 40:185–234, 1989.

[22] G.E. Hinton, J.L. McCelland, and D.E. Rumelhart. Distributed Representations. In McCelland Rumelhart, editor, *Parallel Distributed Processing*, volume 1. MIT Press, 1986.

[23] G.E. Hinton and T.J. Sejnowski. Learning and relearning in boltzmann machines. In McCelland Rumelhart, editor, *Parallel Distributed Processing*, volume 1. MIT Press, 1986.

[24] J.E. Hopcroft and Ullman J.D. *Introduction to Automata theory, languages and computations.* Addison Wesley, 1979.

[25] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *PNAS*, 79:2554–2558, 1982.

[26] J.J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *PNAS*, 82:3088–3092, 1984.

[27] J.M. Jolion and Montanvert A. The adaptive pyramid, a framework for 2D image analysis. *Computer Vision, Graphics, and Image Processing: Image Processing*, Vol. 55(No. 3):pp.339–348, May 1992.

[28] Walter G. Kropatsch. Hierarchical curve representation in a new pyramid scheme. Technical Report TR-1522, University of Maryland, Computer Science Center, June 1985.

[29] Walter G. Kropatsch. A pyramid that grows by powers of 2. *Pattern Recognition Letters*, Vol. 3:pp.315–322, 1985.

[30] Walter G. Kropatsch. Curve representations in multiple resolutions. In *Proc. Eighth International Conference on Pattern Recognition*, pages 1283–1285. IEEE Comp.Soc., 1986.

[31] Walter G. Kropatsch. Elimination von "kleinen" Kurvenstücken in der $2 \times 2/2$ Kurvenpyramide. In E. Paulus, editor, *Mustererkennung 1987*, Informatik Fachberichte 149, pages 156–160. Springer Verlag, 1987.

[32] Walter G. Kropatsch. Image Pyramids and Curves an Overview. Technical Report 2, Dept. for Pattern Recognition and Image Processing, TU-Vienna, March 1991.

[33] Walter G. Kropatsch. Irregular Pyramids. In *Modelling and New Methods in Image Processing and in Geographical Information Systems*, pages 39–50, 1992.

[34] Walter G. Kropatsch and Annick Montanvert. Irregular pyramids. Technical Report PRIP-TR-5, Dept. f. Pattern Recognition and Image processing, TU Wien, 1992.

[35] Y. Le Cun, O. Matan, Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W., Jackel L.D., and Baird H.S. Handwritten ZIP Code Recognition with Multilayer Networks. In *Proc. of the 10.ICPR*, pages 35–40. IEEE Computer Society, 1990.

[36] S. Levialdi. Programming image processing machines. In S. Levialdi and V. Cantoni, editors, *Pyramidal Systems for Image Processing and Computer Vision*, volume F25 of *NATO ASI Series*, pages 311–328. Springer-Verlag Berlin, Heidelberg, 1986.

[37] R. Linsker. From basic network principles to neural architectures (series). *PNAS*, 83:7508–7512,8390–8394,8779–8783, 1986.

[38] R. Linsker. Self-organization in a perceptual network. *IEEE Computer*, 21:105–117, 1988.

[39] Peter Meer. Stochastic image pyramids. *Computer Vision, Graphics, and Image Processing*, Vol. 45(No. 3):pp.269–294, March 1989.

[40] Jordan M.I. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proc. of 8th Annual Conf. of the Cognitive Science Society*, pages 531–546, 1986.

[41] A. Montanvert and Bertolino P. Irregular pyramids for parallel image segmentation. In H. Bischof and W.G. Kropatsch, editors, *Pattern Recognition 1992*, volume OCG-Schriftenreihe 62, pages 13–34. Oldenbourg, 1992.

[42] E. Oja. A simplified neuron model as a priniple component analyzer. *J. Mathematical Biology*, 15:267–273, 1982.

[43] Yon Han Pao. *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley, first edition, 1989.

[44] Smolensky Paul. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46:159–216, 1990.

[45] Axel Pinz and Horst Bischof. Constructing a Neural Network for the Interpretation of the species of Trees in Aerial Photographs. In *Proc. of the 10.ICPR*, pages 755–757. IEEE Computer Society, 1990.

[46] J.B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.

[47] N. Qian and T.J. Sejnowski. Predicting the Secondary Structure of Globular Proteins using Neural Network Models. *Journal of Molecular Biology*, 202:865–884, 1988.

[48] G.X. Ritter, Li D., and Wilson J.N. Image algebra and its relationships to neural networks. In *Proc. of SPIE Tech. Symp. Southeast on Optics, Elec.-Optics, and Sensors*, pages 90–101, Orlando 1989.

[49] Azriel Rosenfeld, editor. *Multiresolution Image Processing and Analysis.* Springer, Berlin, 1984.

[50] Azriel Rosenfeld. Arc colorings, partial path groups, and parallel graph contractions. Technical Report TR-1524, University of Maryland, Computer Science Center, July 1985.

[51] M.W. Roth. Survey of Neural Network Technology for Automatic Target recognition. *IEEE Transaction on Neural Networks*, 1(1):28–43, 1990.

[52] David E. Rumelhart and James A. McClelland. *Parallel Distributed Processing*, volume 1. MIT Press, first edition, 1986.

[53] P.K. Sahoo, S. Soltani, and A.K.C. Wang. A survey of thresholding techniques. *Computer Vision, Graphics, and Image Processing: Image Processing*, Vol. 41:pp.233–260, 1988.

[54] Jean Serra. *Image Analysis and Mathematical Morphology.* Academic Press Inc., New York, 1982.

[55] Steven L. Tanimoto. Paradigms for pyramid machine algorithms. In S. Levialdi and V. Cantoni, editors, *Pyramidal Systems for Image Processing and Computer Vision*, volume F25 of *NATO ASI Series*, pages 173–194. Springer-Verlag Berlin, Heidelberg, 1986.

[56] John K. Tsotsos. Localizing stimuli in a sensory field using an inhibitory attentional beam. Technical Report RBCV-TR-91-37, University of Toronto, October 1991.

[57] S.S. Wilson. Vector morphology and iconic neural networks. *IEEE Trans. on Systems, Man, and Cybernetics*, 19(6):1636–1644, 1989.

[58] Shrivastava Y., Dasgupta S., and Reddy S.M. Guaranteed convergence in a class of hopfield networks. *IEEE Trans. on Neural Networks*, 3(6):951–961, 1992.

[59] Toru Yamaguchi and Walter G. Kropatsch. A vision by neural network or by pyramid. In *Proceedings of the 6th Scandinavian Conference on Image Analysis*, pages 104–111, Oulu, Finland, June 1989.

[60] A. L. Yuille and T. A. Poggio. Scaling theorems for zero crossings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 8(No.1):pp.15–25, January 1986.