In Proceedings of International Joint Conference on Neural Networks (IJCNN'99) -- Washington, DC, paper number #491, 1999.

Hebbian learning and competition in the Neural Abstraction Pyramid

Sven Behnke
Free University of Berlin, Institute of Computer Science
Takustr. 9, 14195 Berlin, Germany
behnke@inf.fu-berlin.de

Abstract

The recently introduced Neural Abstraction Pyramid is a hierarchical neural architecture for image interpretation that is inspired by the principles of information processing found in the visual cortex. In this paper we present an unsupervised learning algorithm for it's connectivity based on Hebbian weight updates and competition. The algorithm yields a sequence of feature detectors that produce increasingly abstract representations of the image content. These representations are distributed, sparse, and facilitate the interpretation of the image.

We apply the algorithm to a dataset of handwritten digits, starting from local contrast detectors. The emerging feature detectors correspond to step edges, lines, strokes, curves, and digit shapes. They can be used to reliably classify the digits.

1 Introduction

One of the most important problems in pattern recognition is the extraction of meaningful features from the input signals. To compute symbolic information, such as the class of the observed object, it is necessary to aggregate characteristic aspects of the observation in a feature vector that is presented to a classification system.

A variety of feature extraction methods exist, e.g. for the problem of handwritten digit recognition [9]. Some methods use the normalized pixel image as input for a powerful statistical or neural classifier [2]. Others use features having a certain degree of abstraction, such as moments [8] or coefficients of the KL-transformation [6]. The most abstract features are extracted by methods that use the digit's structure for recognition [1]. All these features usually need specific tuning towards the task at hand. Therefore, the transfer to other applications is difficult. For this reason it would be desirable to learn abstract features from example images.

One way to extract features is to construct image pyramids [4]. This multiscale representation quickly reduces the size of the image and can be used to aggregate meaningful information in a single feature. Two-dimensional wavelets [2] are more general, since they usually compute four features per level. The recently introduced Neural Abstraction Pyramid approach to image interpretation [3] offers an even more powerful way for image representation: the number of features increases as the spatial resolution decreases.

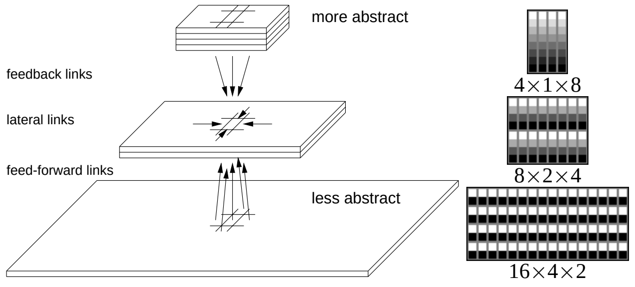

Figure 1: Sketch of the Neural Abstraction Pyramid.

In this paper we introduce an unsupervised learning algorithm for the connections which determine the features detected in the pyramid. It is based on Hebbian weight updates and competition and yields a sequence of more and more abstract representations as their spatial resolution decreases, their diversity increases and they become increasingly sparse. The paper is organized as follows: In the next section we give a brief summary of the Neural Abstraction Pyramid architecture and algorithms. The unsupervised weight construction is described in Section 3. In Section 4 this algorithm is applied to a dataset of handwritten digits. The paper concludes with a discussion of some experimental results and gives an outlook to future work.

2 Neural Abstraction Pyramid

The Neural Abstraction Pyramid [3] is a hierarchical architecture for iterative image interpretation that is based on image pyramids [4] and cellular neural networks [5]. It is inspired by the information processing principles found in the visual cortex. Algorithms for this architecture are defined in terms of local interactions of processing elements that utilize horizontal as well as vertical feedback loops. The goal is to transform a given image into a sequence of increasingly abstract representations. The main features of the architecture are:

- *Pyramidal shape:* Layers of *columns* are arranged vertically to form a pyramid (see Fig. 1). Each column con-

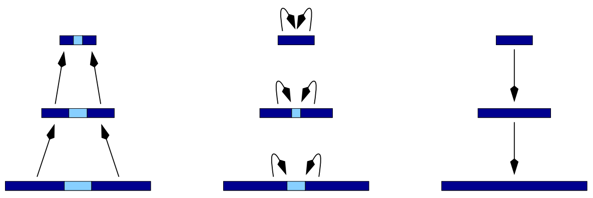

Figure 2: Iterative image interpretation.

sists of a set of neural processing elements (*nodes*) with overlapping receptive fields. The number of nodes per column increases and the number of columns per layer decreases towards the top of the pyramid.

- *Analog representation:* Each layer describes an image in a two-dimensional representation where the level of abstraction increases with height, while spatial resolution decreases. The bottom layer stores the given image (a signal). Subsymbolic representations are present in intermediate layers, while the highest layers contain almost symbolic descriptions of the image content. These representations consist of *quantities* that have an *activity value* from a finite interval for each column.

- *Local interaction:* Each node is connected to some nodes from its neighborhood via directed *weighted links*. The shared weights of all nodes in a layer that represent the same quantity are described by a common *template*. The links can be classified as:
  - *feed-forward links:* perform feature extraction,
  - *lateral links:* for consistent interpretation,
  - *feedback links:* provide interpretation hypotheses.

- *Discrete time computation:* The update of a node's value for time step t depends only on the input values at $(t-1)$. It can be done in a predefined order or priority driven.

Image interpretation works *iteratively,* as sketched in Fig. 2. First, the given image is fed into the bottom layer. In the course of computation the activities spread upwards via feed-forward links at locations where little ambiguity exists. These partial results provide, via lateral and feedback links, a larger context for the interpretation of the more ambiguous image parts. This refines the output and, after a few iterations, the interpretation becomes stable.

3 Unsupervised learning

In order to make the Neural Abstraction Pyramid approach to image interpretation work, we need a sequence of increasingly abstract models of the potential image content. In [3] we designed such models manually. We derived templates for foreground/background, oriented edges, and oriented lines for the binarization of handwritten digits following a Gestalt approach. Although the manual design for such a task is possible, it becomes difficult for problems with more levels of abstraction, since the number of quantities per layer and the number of potential weights per node increase exponentially when going up in the pyramid. Therefore, we need an automatic procedure that learns the models from a given set of example images. Since we don't know exactly, how such models should look like, learning has to be unsupervised. The emerging representations should have following properties:

- *Completeness:* All interesting features of the input image should be represented.

- *Sparseness:* The value of a quantity should be zero at most positions and high at only a few positions.

- *Fairness:* All quantities of a layer should contribute approximately equally to the representation.

In the following we present an unsupervised learning algorithm that constructs templates with the desired properties. It is based on Hebbian weight updates and competition. We assume that appropriate templates for the bottom layer are given. Training works iteratively. Using the topmost representation on layer $(z-1)$ as input it constructs templates for the next layer z. For each layer, first the feed-forward weights are constructed, then the feedback weights to the previous layer are added, and finally the lateral weights are trained. Since the number of layers is logarithmic in the image size, only a few steps are needed to construct the entire hierarchy.

3.1 Feed-Forward Weights

3.1.1 Initialization

We use Σ-Units as neural processing nodes that compute the weighted sum of their inputs and apply a nonlinear output function. The update of a nodes value $v_{x,y,z,q}$ at column (x,y) in layer z for quantity q is done as follows:

$$v_{x,y,z,q}^{t+1} = \sigma\left[\sum_{j\in\mathcal{L}(i)} \mathcal{W}(j)\; v^{t}_{\mathcal{X}(j,x),\mathcal{Y}(j,y),\mathcal{Z}(j,z),\mathcal{Q}(j)} + \mathcal{B}(i)\right].$$

The template $i = \mathcal{T}(z,q)$ is associated with quantity q at layer z. $\mathcal{L}(i)$ is the set of links of that template and $\mathcal{B}(i)$ is the template bias. $(\mathcal{X}(j,x),\mathcal{Y}(j,y),\mathcal{Z}(j,z),\mathcal{Q}(j))$ describe location and quantity of the input value for link j, and $\mathcal{W}(j)$ is the link weight. The output function $\sigma(.)$ is here a saturation function that limits the values to the interval $[0,1]$. We set the bias values $\mathcal{B}(i)$ to zero. All feed-forward inputs of a node come from the corresponding position in the layer $(z-1)$ directly below the node. Every link is either excitatory or inhibitory. We decided to use specific excitatory links that have the sum $\mathcal{E}(i)$ and unspecific inhibition via a single inhibitory link $\mathcal{I}(i)$. The excitatory connections originate from all nodes of a $K \times K$ window of columns around the position of the node in layer $(z-1)$. The weights of the excitatory links are initialized unspecifically: larger positive weights are used in the center and weaker weights are used towards the periphery of the receptive field window. The weights have a random component and are normalized to a sum of $\mathcal{E}(i) = 2.0$. The unspecific inhibitory inputs $\mathcal{I}(i)$ don't come directly from the nodes of the below layer, but are aggregated via two intermediate steps. First, the smoothed sum $s_{x,y,(z-1)}$ of the values $v_{x,y,(z-1),q}$ of all quantities q is computed for each column (x,y) in layer $(z-1)$. Each node in the same column has the weight of α_0 in that sum and nodes from neighboring columns contribute with the weight α_1. Then, $s_{x,y,(z-1)}$ is subsampled by two in each direction and called $\hat{s}_{x,y,(z-1)}$. The inhibitory connection of a node in layer z originates from the corresponding node of this quantity and is initialized with the weight $\mathcal{I}(i) = 0$. The ratio between the sum of the excitatory links $\mathcal{E}(i)$ and the inhibitory link $\mathcal{I}(i)$ determines how specific a node will react to incoming stimuli. If the inhibition is small compared to the excitation, the node will react unspecifically to many stimuli that partially match its excitatory links. In contrast, if the inhibition is strong, the node will be sharply tuned to the stimuli that exactly match its weights.

The number of nodes per column is chosen to be two times the corresponding number from the layer below and the number of columns reduces by a factor of four. This reduces the number of nodes per layer by a factor of two with each step. In addition to the templates that describe the quantities of the new layer, we insert a template that computes the smoothed sum of all quantities $s_{x,y,z}$ as described above.

3.1.2 Hebbian weight update

Hebbian weight modifications [7] are used to make the excitatory weights specific. The idea is to change the template of the most active node so that it becomes more specific to the current input. This means, it will answer stronger to the same stimulus and answer weaker to other stimuli.

For each training step an image is chosen randomly. It is loaded into the bottom layer of the pyramid and the values of all nodes are computed in the appropriate order for a small number of iterations. We apply the following learning rules for all positions in which the smoothed and subsampled sum of the inputs $\hat{s}_{x,y,(z-1)}$ and the smoothed sum of the outputs $s_{x,y,z}$ is different from zero. For these columns (x,y) we find the most active quantity q_{max} and the quantity q_{sec} with the second highest value. We change the excitatory weights $\mathcal{W}(j)$ of the winning template $i = \mathcal{T}(z, q_{max})$ as follows:

$$\begin{aligned} \Delta\mathcal{W}(j) &= \eta_z \, v_{in} \, v_{out}, \\ \text{with} \quad v_{in} &= \mathcal{H}(j) \, v_{\mathcal{X}(j,x),\mathcal{Y}(j,y),z-1,\mathcal{Q}(j)}, \\ v_{out} &= v_{x,y,z,q_{max}} - v_{x,y,z,q_{sec}}. \end{aligned}$$

A weight $\mathcal{W}(j)$ is increased by an amount that is proportional to the product of scaled input activity and the amount that the activity of the winner exceeds the one of the second best quantity. The learning rate η_z increases with height, e.g. $\eta_z = 0.001 M_z$, where M_z is the number of quantities in layer z. The scaling factor $\mathcal{H}(j)$ used for the input activity is one in the center of the window and descends to zero towards the periphery. It enforces a centered response of the template. The Hebbian term makes the weights larger. To prevent unlimited weight growth, the sum of the excitatory weights $\mathcal{E}(i)$ is kept constant by scaling down all weights $\mathcal{W}(j)$ with a common factor. The net effect is that the weights receiving strong input are increased and the other weights are decreased.

3.1.3 Competition

The above normalization of the sum of excitatory weights is a form of competition. The excitatory weights $\mathcal{W}(j)$ of a template i compete to have a large portion of its weight sum $\mathcal{E}(i)$. We need also some sort of competition between the M_z templates of layer z to enforce the given constraints.

To fulfill the fairness constraint, the winning frequency of all templates should be about the same. We increase the strength of a node's inhibitory weight $\mathcal{I}(i)$ each time it wins the competition, and decrease it otherwise. This makes templates that win above average less active and more specific. Consequently, these templates will not win too frequently. On the other hand, templates that win less often become more active and less specific and therefore win now more often. The change is done by adding a small constant δ_f to $\mathcal{I}(i)$ such that the net effect for a template that has an average winning frequency is zero, e.g. $\delta_f^{winning} = -\eta_f$, $\delta_f^{not\_winning} = -\delta_f^{winning}/(M_z - 1)$. If this makes $\mathcal{I}(i)$ positive, we add its weight to $\mathcal{E}(i)$ and set $\mathcal{I}(i)$ to zero.

To achieve a complete representation, we force the templates to react to all significant stimuli by controlling the smoothed sum $s_{x,y,z}$ of the quantities in layer z to be equal to the smoothed and subsampled sum $\hat{s}_{x,y,z-1}$ of the input quantities from layer $(z-1)$: $\Delta\mathcal{I}(i) = \Delta\mathcal{E}(i) = (\eta_c/M_z) \, v_{x,y,z,q}(\hat{s}_{x,y,(z-1)} - s_{x,y,z})$. If the activity of the templates is too low, we increase the excitatory weights of the active templates and disinhibit them at the same time.

To enforce sparseness, we make the average winning activity of a template large, e.g. $\mathcal{V} = 0.75$: $\Delta\mathcal{E}(i) = \eta_s(\mathcal{V} - v_{x,y,z,q_{max}})$. If the activity of the winner is too small, its excitatory weights are scaled up. Otherwise they are decreased.

The efficacy of the constraint enforcing rules can be controlled by the factors η. One useful choice could be: $\eta_f = \eta_c = \eta_s = 0.1\eta_z$. The rules are designed such that their net effect goes to zero, if the produced representation has the desired properties. Then the templates become stable and the training can be terminated. The number of training images needed to determine the weights of the templates for a layer increases with the height of that layer, since the number of examples per image decreases and the number of weights per layer increases. Because the emerging representations are sparse, most of the weights will be zero after training and can be pruned away without loss. This significantly speeds up the computation and saves memory.

3.2 Feedback Weights

After the feed-forward weights of the topmost layer z have been constructed, the feedback weights to the previous layer $(z-1)$ are added to the pyramid. Each node receives specific excitatory feedback from all quantities in the upper layer that correspond to its position and unspecific inhibitory feedback from the smoothed sum of these quantities. The excitatory weights are initialized with a constant value. To make them specific, the pyramid is updated for some iterations and the local maxima of the activities in the layer $(z-1)$ are determined. The weights of the winning nodes q_{max} are changed using a Hebbian term:

$$\Delta\mathcal{W}(j) = \eta_b \, v_{\mathcal{X}(j,x),\mathcal{Y}(j,y),z,\mathcal{Q}(j)} \, v_{x,y,z-1,q_{max}}.$$

The learning rate η_b decreases with the number of updates. To prevent unlimited weight growth, the sum of the excitatory weights is normalized. The inhibitory weight is chosen such that there is neither oscillation nor unlimited increase of activity, but a stable intermediate state after few iterations.

3.3 Lateral Weights

The final step in the design of a new layer is the construction of its lateral weights. Again, we use specific excitation and unspecific inhibition. The set of excitatory lateral weights of a node in layer z originate from all nodes of its $L \times L$ neighborhood. The inhibitory lateral weight comes from the smoothed sum of these quantities. After the pyramid has been updated for some iterations, the nodes q_{max} with locally maximal activities are found and updated using the Hebb rule:

$$\Delta\mathcal{W}(j) = \eta_l \, v_{\mathcal{X}(j,x),\mathcal{Y}(j,y),z,\mathcal{Q}(j)} \, v_{x,y,z,q_{max}}.$$

As for the feedback weights, η_l decreases, the sum of the excitatory weights is kept constant, and the inhibitory weight is chosen to balance the average effect of the excitation.

4 Experimental Results

To illustrate the properties of the described unsupervised learning algorithm, we apply it to a dataset of handwritten digits that have been extracted from German ZIP-codes. The available digits were partitioned as follows: 44619 digits constitute the training set, 5379 digits were used for testing the system performance and to stop training, and 6313 digits were used for final validation.

Since the digits show a high degree of variance, some preprocessing steps are necessary prior the presentation to the pyramid. Preprocessing consists of binarization, size and slant normalization. The images are scaled to 24×24 pixel and centered into the 32×32 bottom layer of the pyramid.

The Neural Abstraction Pyramid is initialized at the lowest level (z=0) with contrast detectors. These have a center-surround type receptive field that looks at the gray values of the input image. There are four different quantities: center-on-off-surround and center-off-on-surround in two scales, representing the fine and coarse details of the foreground and the background, respectively. We construct feed-forward templates for higher layers using the window-size $K = 4$ and the sum weights $\alpha_0 = 2/M_z$, and $\alpha_1 = 0.5\,\alpha_0$.
The following representations emerge (compare to Table 1):

- *Edges:* Vertical, horizontal, and diagonal step edges are detected at layer one. Figure 3 shows the activities of the Edge-templates and their weights that receive input from the contrast detectors. In addition, the best stimuli from the training set are displayed for each edge template.

- *Lines:* At layer two short line segments with 16 different

z	name	quantities	columns	nodes	size
5	digits	128	1×1	128	32×32
4	curves	64	2×2	256	16×16
3	strokes	32	4×4	512	8×8
2	lines	16	8×8	1024	4×4
1	edges	8	16×16	2048	2×2
0	contrasts	4	32×32	4096	1×1

Table 1: Hierarchy of emerging representations.

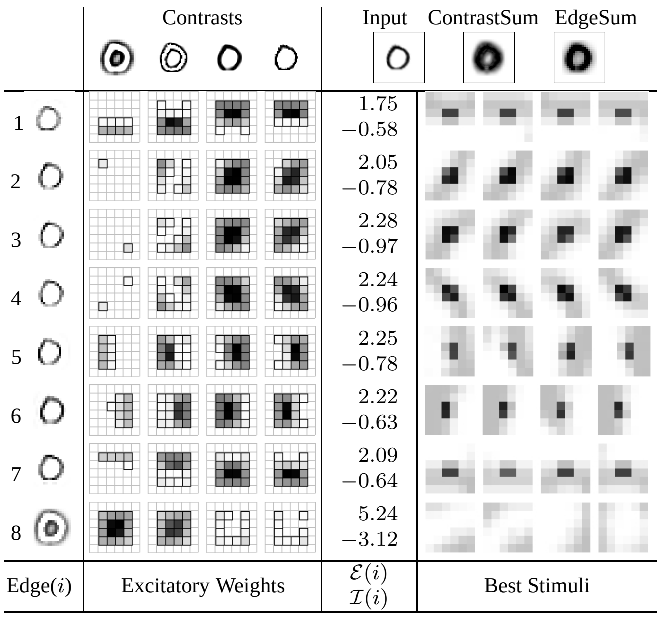

Figure 3: Edge templates. Shown are: activity of the quantities for some digit (Input ”0”), excitatory weights to Contrasts, excitation sum $\mathcal{E}(i)$ and inhibition $\mathcal{I}(i)$, stimuli that caused the highest winning activity.

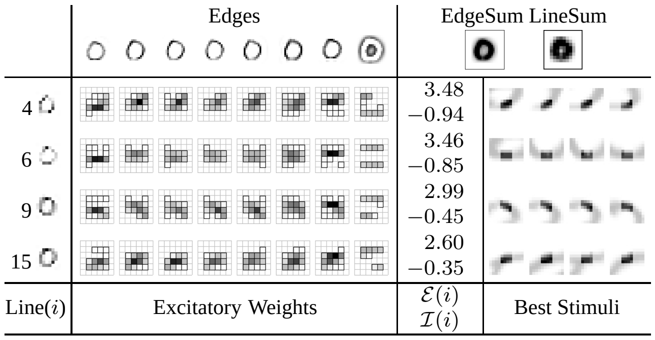

Figure 4: Line templates. Four horizontal templates have been chosen.

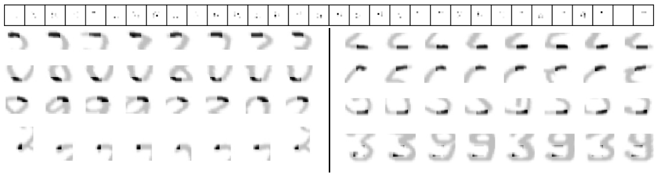

Figure 5: Stroke templates. Shown are the eight best stimuli of eight templates that detect horizontal lines.

orientations are detected (see Figure 4).

- *Strokes:* Larger line segments that have a specific orientation and a specific curvature are detected at layer tree (Figure 5). Detectors for line endings and specific parallel lines emerge also.
- *Curves:* The feature detectors at layer four react to typical large substructures of digits, such as curves, crossings, junctions, etc. (compare to Figure 6).
- *Digits:* The templates at the topmost layer see the entire digit. Consequently, detectors for typical digit shapes emerge (shown in Fig. 7). The left side of the figure displays templates that react best to digits from a specific class. The right side shows templates that are stimulated by digits from multiple classes. They seem to focus on some aspect of the digit, such as the presence of a vertical line or a characteristic curve.

The emerging templates do not represent all possible combinations of substructures, but only the typical ones. The frequent combinations are represented by multiple similar templates with great detail. When going up in the hierarchy of representations, the correlation of template activities with the digit class increases. This is remarkable, since no class information has been presented to the system so far.

We investigate now the usefulness of the feature detectors for digit recognition. First, we construct only two layers of the pyramid that include lateral and feedback weights. For each example we update the activities six times and input the topmost layer to a classifier. Table 2 shows the performance

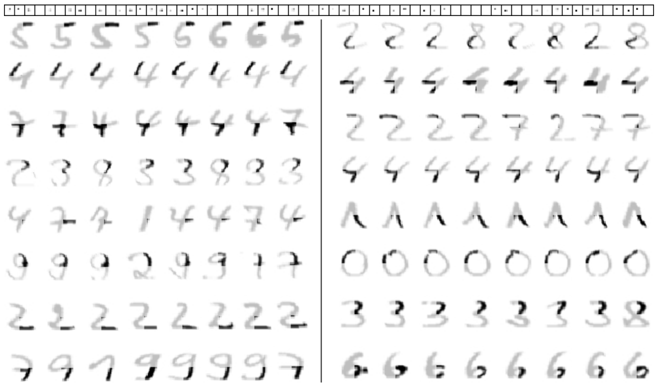

Figure 6: Curve templates. Shown are the eight best stimuli of the 16 first templates.

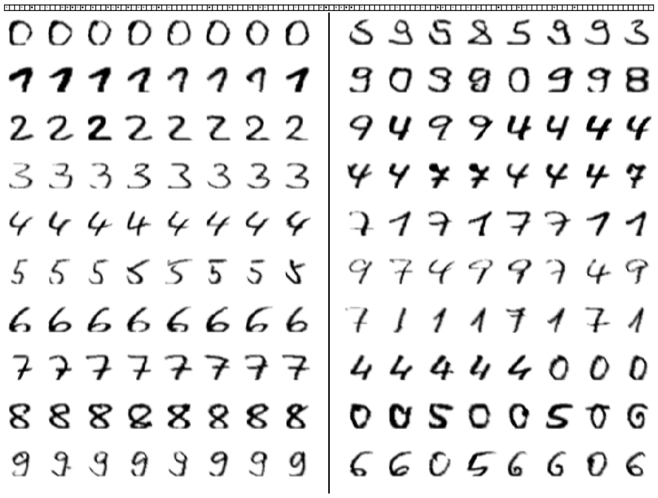

Figure 7: Digit templates. Shown are the eight best stimuli. For the left column templates have been chosen that correspond to a single class. The right column shows templates that focus on some other aspect of a digit.

of a KNN classifier and two feed-forward neural networks (FFNN) that have been trained with backpropagation using the digit's gray values and the extracted lines as features. One can see that the performance of the neural networks is better for the more abstract features. Furthermore, the digits can be classified from the Lines representation by a network without hidden units, while the same network did not converge during training on the Gray representation.

In the second experiment, we feed the top four layers of the feed-forward pyramid into a 1920-128-10 FFNN to classify the digits. After 120 epochs of online-training with a learning rate of $\eta = 0.01$ we observe a zero-reject substitution rate of 1.65% on the test set and of 1.49% on the validation set. Table 3 shows the results for different numbers of hidden units, as well as for a network without hidden units and a KNN classifier. These rates compares favorably to the

features / classifier	Gray TST	Gray VAL	Lines TST	Lines VAL
KNN 15	2.98	2.87	4.53	4.36
FFNN $1024-10$	no convergence		2.04	2.14
FFNN $1024-64-10$	2.49	2.65	1.90	2.04

Table 2: Zero-reject substitution rates of different classifiers.

results published in [1] for the same data set. We can also reject ambiguous digits by looking at the two best classes. The substitution rate drops to 0.55% when 2.52% of the validation set are rejected and to 0.21% for 7.9% rejects. Figure 8 shows the substitution-reject curve of this classifier, compared to the structural classifier and the TDNN classifier from [1]. Clearly, the classifier that used the features extracted by the Neural Abstraction Pyramid performs about as well as the combination of the other two classifiers. The figure also shows the results when the new classifier is combined sequentially [2] with the other two. Now the zero-reject substitution rate drops to 1.17%. The substitution rate can be reduced to 0.30% with 3.60% and to 0.11% with 9.20% rejects. These results are the best we know for this dataset.

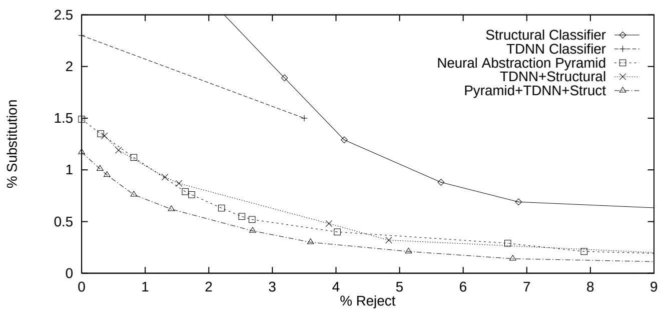

Figure 8: Performance of different digit classifiers.

classifier	TST	VAL
KNN 15	3.59	3.64
FFNN $1920-10$	1.71	1.66
FFNN $1920-32-10$	1.71	1.73
FFNN $1920-64-10$	1.67	1.68
FFNN $1920-128-10$	1.65	1.49

Table 3: Zero-reject substitution rates of different classifiers that input the upper four layers of the feed-forward pyramid.

5 Discussion

This paper presented an unsupervised learning algorithm for the construction of the weights in the Neural Abstraction Pyramid. We applied the algorithm to a dataset of handwritten digits to produce increasingly abstract digit representations. The emerging feature detectors are meaningful and can be interpreted in terms of detected combinations of digit substructures. This leads to a hierarchical image description that is distributed and sparse. When looking at the best stimuli for the feature detectors, one can see that these are not similar in terms of their pixel image, but in terms of their recursive decomposition to substructures. The pyramidal digit representation becomes increasingly invariant against distortions when going up in the hierarchy.

The features extracted facilitate the recognition of the digits. When used as input to an FFNN-classifier the recognition performance observed was very satisfactory. It outperforms any single classifier that has been tested on that dataset and is about as good as the combination of the TDNN and the structural digit recognizer. When combined with these two classifiers, the recognition performance improves further.

We also tested methods for the design of the lateral and vertical connectivity in the pyramid. These links provide a large context to interpret ambiguous image parts at low cost.

We want to investigate these effects in greater detail in the future. Furthermore, we plan to apply the Neural Abstraction Pyramid approach to more complex image interpretation problems like face recognition and object segmentation.

References

[1] S. Behnke, M. Pfister, and R. Rojas. Recognition of handwritten digits using structural information. In *Proceedings ICNN'97*, volume. 3, pages 1391–1396, 1997.

[2] S. Behnke, M. Pfister, and R. Rojas. A study on the combination of classifiers for handwritten digit recognition. In *Proceedings NN'98*, pages 39–46, 1998.

[3] S. Behnke and R. Rojas. Neural abstraction pyramid: A hierarchical image understanding architecture. In *Proceedings IJCNN'98*, volume 2, pages 820–825, 1998.

[4] V. Cantoni and M. Ferretti. *Pyramidal Architectures for Computer Vision*. Advances in computer vision and machine intelligence. Plenum Press, New York, 1994.

[5] L. O. Chua and T. Roska. The CNN paradigm. *Transactions on Circuits and Systems*, 40(3):147–156, 1993.

[6] P. J. Grother and G. T. Candela. Comparison of handprinted digit classifiers. NISTIR 5209, NIST, 1993.

[7] Donald Hebb. *The Organization of Behaviour*. John Wiley, New York, 1949.

[8] H.K. Sardana, M.F. Daemi, and M.K. Ibrahim. Global description of edge patterns using moments. *Pattern Recognition*, 27(1):109–118, 1994.

[9] C.L. Wilson. Evaluation of character recognition systems. In *Neural Networks for Signal Processing III – New York*, pages 485–496. IEEE, 1993.