Feature extraction through LOCOCODE

Neural Computation 11(3):679–714, 1999

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

Abstract

"*Low-co*mplexity *co*ding and *de*coding" (LOCOCODE) is a novel approach to sensory coding and unsupervised learning. Unlike previous methods it explicitly takes into account the information-theoretic complexity of the code generator: it computes *lococodes* that (1) convey information about the input data and (2) can be computed and decoded by low-complexity mappings. We implement LOCOCODE by training autoassociators with *Flat Minimum Search*, a recent, general method for discovering low-complexity neural nets. It turns out that this approach can unmix an unknown number of independent data sources by extracting a minimal number of low-complexity features necessary for representing the data. Experiments show: unlike codes obtained with standard autoencoders, lococodes are based on feature detectors, never unstructured, usually sparse, sometimes factorial or local (depending on statistical properties of the data). Although LOCOCODE is not explicitly designed to enforce sparse or factorial codes, it extracts optimal codes for difficult versions of the "bars" benchmark problem, whereas ICA and PCA do not. It produces familiar, biologically plausible feature detectors when applied to real world images, and codes with fewer bits per pixel than ICA and PCA. Unlike ICA it does not need to know the number of independent sources. As a preprocessor for a vowel recognition benchmark problem it sets the stage for excellent classification performance. Our results reveil an interesting, previously ignored connection between two important fields: regularizer research, and ICA-related research. They may represent a first step towards unification of regularization and unsupervised learning.

1 INTRODUCTION

What is the goal of sensory coding? There is no generally agreed-upon answer to Field's (1994) question yet. Several information-theoretic objective functions (OFs) have been proposed to evaluate the quality of sensory codes. Most OFs focus on statistical properties of the code components (such as mutual information) — we refer to them as code component-oriented OFs, or COCOFs. Some COCOFs explicitly favor near-factorial, minimally redundant codes of the input data (see, e.g., Watanabe 1985, Barlow et al. 1989, Linsker 1988, Schmidhuber 1992, Schmidhuber and Prelinger 1993, Schraudolph and Sejnowski 1993, Redlich 1993, Deco and Parra 1994). Such codes can be advantageous for (1) data compression, (2) speeding up subsequent gradient descent learning (e.g., Becker 1991), (3) simplifying subsequent Bayes classifiers (e.g., Schmidhuber et al. 1996). Other approaches favor local codes, e.g., Rumelhart and Zipser (1986), Barrow (1987), Kohonen (1988). They can help to achieve (1) minimal crosstalk, (2) subsequent gradient descent speed-ups, (3) facilitation of post training analysis, (4) simultaneous representation of different data items. Recently there also has been much work on COCOFs encouraging biologically plausible sparse distributed codes, e.g., Field (1987), Barlow (1983), Mozer (1991), Földiák (1990), Földiák and Young (1995), Palm (1992), Zemel and Hinton (1994), Field (1994), Saund (1994), Dayan and Zemel (1995), Li (1995), Olshausen and Field (1996), Zemel (1993), Hinton and Ghahramani (1997). Sparse codes share certain advantages of both local and dense codes.

But what about coding costs? COCOFs express desirable properties of the code itself, while neglecting the costs of constructing the code from the data. For instance, coding input data without redundancy may be very expensive in terms of information bits required to describe the code-generating network, which may need many finely tuned free parameters. In fact, the most compact code of the possible environmental inputs would be the "true" probabilistic causal model corresponding to our universe's most basic physical laws. Generating this code and using it for dealing with everyday events, however, would be extremely inefficient.

A previous argument for ignoring coding costs (e.g., Zemel 1993, Zemel and Hinton 1994, Hinton and Zemel 1994), based on the principle of minimum description length (MDL, e.g., Solomonoff 1964, Wallace and Boulton 1968, Rissanen 1978), focuses on hypothetical costs of transmitting the data from some sender to a receiver — how many bits are necessary to enable the receiver to reconstruct the data? It goes more or less like this: *"Total transmission cost is the number of bits required to describe (1) the data's code, (2) the reconstruction error and (3) the decoding procedure. Since all input exemplars are encoded/decoded by the same mapping, the coding/decoding costs are negligible because they occur only once."*

We doubt, however, that sensory coding's sole objective should be to transform data into a compact code that is cheaply transmittable across some ideal, abstract channel. We believe that one of sensory coding's objectives should be to reduce the cost of code *generation* through data transformations in *existing* channels (e.g., synapses etc.)[1]. Without denying the usefulness of certain COCOFs, we postulate that an important scarce resource is the bits required to describe the mappings that generate and process the codes — after all, it is these mappings that need to be implemented, given some limited hardware.

Lococodes. For such reasons we shift the point of view and focus on the information-theoretic costs of code-generation (compare Pajunen (1998) for recent related work). We will present a novel approach to unsupervised learning called "*lo*w-*co*mplexity *co*ding and *de*coding" (LOCOCODE — see also Hochreiter and Schmidhuber 1997b, 1997c, 1998). Without assuming particular goals such as data compression, simplifying subsequent classification, etc., but in the MDL spirit, LOCOCODE generates so-called *lococodes* that (1) convey information about the input data, (2) can be computed from the data by a low-complexity mapping (LCM), (3) can be decoded by a LCM. By minimizing coding/decoding costs LOCOCODE will yield efficient, robust, noise-tolerant mappings for processing inputs and codes.

Lococodes through FMS. To implement LOCOCODE we apply *Flat Minimum Search* (FMS, Hochreiter and Schmidhuber 1997a) to an autoassociator (AA) whose hidden layer activations represent the code. FMS is a general, gradient-based method for finding networks that can be described with few bits of information.

Coding each input via few simple component functions (CFs). A CF is the function determining the activation of a code component in response to a given input. The analysis in Section 3 will show that FMS-based LOCOCODE tries to reproduce the current input by using as few code components as possible, each computed by a separate low-complexity CF (implementable, e.g., by a subnetwork with few low-precision weights).

This reflects a basic assumption, namely, that the true input "causes" (e.g., Hinton et al. 1995, Dayan and Zemel 1995, Ghahramani 1995) are indeed few and simple. Training sets whose elements are all describable by few features will result in *sparse* codes, where sparseness does not necessarily mean that there are "few active code components" but that "few code components contribute to reproducing the input". This can make a difference in the nonlinear case, where the *absence* of a particular HU activation may imply *presence* of a particular feature, and where sparseness may mean that for each input only few HUs are simultaneously *non*-active: our generalized view of sparse codes allows for noninformative activation values other than zero. (But LOCOCODE does prune code components that are always inactive or always active.)

We will see that LOCOCODE encourages noise-tolerant feature detectors reminiscent of those

[1] Note that the mammalian visual cortex rarely just transmits data without also transforming it.

observed in the mammalian visual cortex. Inputs that are mixtures of a few regular features, such as edges in images, can be described well in a sparse fashion (only code components corresponding to present features contribute to coding the input). In contrast to previous approaches, however, sparseness is not viewed as an *a priori* good thing, and is not enforced explicitly, but only if the input data indeed is naturally describable by a sparse code. Some lococodes are not only sparse but also factorial, depending on whether the input is decomposable into factorial features. Likewise lococodes may deviate from sparseness towards locality if each input exhibits a single characteristic feature. Then the code will not be factorial (because knowledge of the component representing the characteristic feature implies knowledge of all others), but it will still be natural because it represents the true cause in a fashion that makes reconstruction (and other types of further processing) simple.

Outline. An FMS-review will follow in Section 2. Section 3 will analyze the beneficial effects of FMS' error terms in the context of autoencoding. The remainder of our paper will be devoted to empirical justifications of Lococode. Experiments in Section 4.2 will show that all three "good" kinds of code discussed in previous work (namely local, sparse, factorial) can be natural lococodes. In Section 4.3 Lococode will extract optimal sparse codes reflecting the independent features of random horizontal and vertical (noisy) bars, while ICA and PCA won't. In Section 4.4 Lococode will generate plausible sparse codes (based on well-known on-center-off-surround and other appropriate feature detectors) of real world images. Section 4.5 will finally use Lococode as a preprocessor for a standard, overfitting back-propagation (BP) speech data classifier. Surprisingly, this combination achieves excellent generalization performance. We conclude that the speech data's lococode already conveys the "essential", noise-free information, already in a form useful for further processing and classification. Section 5 will discuss our findings.

2 FLAT MINIMUM SEARCH: REVIEW

FMS Overview. FMS is a general method for finding low complexity-networks with high generalization capability. FMS finds a large region in weight space such that each weight vector from that region has *similar* small error. Such regions are called "flat minima". In MDL terminology, few bits of information are required to pick a weight vector in a "flat" minimum (corresponding to a low complexity-network) — the weights may be given with low precision. In contrast, weights in a "sharp" minimum require a high-precision specification. As a natural by-product of net complexity reduction, FMS automatically prunes weights and units, and reduces output sensitivity with respect to remaining weights and units. Previous FMS applications focused on supervised learning (Hochreiter and Schmidhuber 1995, 1997a): FMS led to better stock market prediction results than "weight decay" and "optimal brain surgeon" (Hassibi and Stork 1993). In this paper, however, we will use it for unsupervised coding only.

Architecture. We use a 3-layer feedforward net. Each layer is fully connected to the next. Let O, H, I denote index sets for output, hidden, input units, respectively. Let $|.|$ denote the number of elements in a set. For $l \in O \cup H$, the activation y^l of unit l is $y^l = f_l(s_l)$, where $s_l = \sum_m w_{lm} y^m$ is the net input of unit l ($m \in H$ for $l \in O$ and $m \in I$ for $l \in H$), w_{lm} denotes the weight on the connection from unit m to unit l, f_l denotes unit l's activation function, and for $m \in I$, y^m denotes the m-th component of an input vector. $W = |(O \times H) \cup (H \times I)|$ is the number of weights.

Algorithm. FMS' objective function E features an unconventional error term:

$$B = \sum_{i,j \in O \times H \cup H \times I} \log \sum_{k \in O} \left(\frac{\partial y^k}{\partial w_{ij}} \right)^2 + W \log \sum_{k \in O} \left(\sum_{i,j \in O \times H \cup H \times I} \frac{\left| \frac{\partial y^k}{\partial w_{ij}} \right|}{\sqrt{\sum_{k \in O} \left(\frac{\partial y^k}{\partial w_{ij}} \right)^2}} \right)^2 .$$

$E = E_q + \lambda B$ is minimized by gradient descent, where E_q is the training set mean squared error (MSE), and λ a positive "regularizer constant" scaling B's influence. Defining λ corresponds to choosing a tolerable error level (there is no *a priori* "optimal" way of doing so). B measures the weight precision (number of bits needed to describe all weights in the net). Reducing B without increasing E_q means removing weight precision without increasing MSE. Given a constant number of output units, all of this can be done efficiently, namely, with standard BP's order of computational complexity. For details see Hochreiter and Schmidhuber's article (1997a) or their home pages. For even more general, algorithmic methods reducing net complexity see Schmidhuber (1997a).

3 EFFECTS OF THE ADDITIONAL TERM B

Where does B come from? To discover flat minima FMS searches for large axis-aligned hypercuboids (boxes) in weight space such that weight vectors within the box yield similar network behavior. Boxes satisfy two flatness conditions, FC1 and FC2. FC1 enforces "tolerable" output variation in response to weight vector perturbations, i.e., near-flatness of the error surface around the current weight vector (in all weight space directions). Among the boxes satisfying FC1, FC2 selects a unique one with minimal net output variance. B is the negative logarithm of this box's volume (ignoring constant terms that have no effect on the gradient descent algorithm). Hence B is the number of bits (save a constant) required to describe the current net function, which does not change significantly by changing weights within the box. The box edge length determines the required weight precision. See Hochreiter and Schmidhuber (1997a) for details of B's derivation.

3.1 First term of B favors sparseness and simple CFs.

Simple component functions (CFs). The term

$$T1 \ := \sum_{i,j \in O \times H \cup H \times I} \log \sum_{k \in O} \left(\frac{\partial y^k}{\partial w_{ij}} \right)^2$$

reduces output sensitivity with respect to weights (and, therefore, units). $T1$ is responsible for pruning weights (and, therefore, units). The chain rule allows for rewriting

$$\frac{\partial y^k}{\partial w_{ij}} = \frac{\partial y^k}{\partial y^i} \frac{\partial y^i}{\partial w_{ij}} = \frac{\partial y^k}{\partial y^i} \ f_i'(s_i) \ y^j ,$$

where $f_i'(s_i)$ is the derivative of the activation function of unit i with activation y^i. If unit j's activation y^j decreases towards zero then for all i the $\frac{\partial y^k}{\partial w_{ij}}$ will decrease. If the first order derivative $f_i'(s_i)$ of unit i decreases towards zero then for all j $\frac{\partial y^k}{\partial w_{ij}}$ will decrease. Note that $f_i'(s_i)$ and y^j are independent of k and can be placed outside of the sum $\sum_{k \in O}$ in $T1$. We obtain:

$$\begin{aligned} T1 \ &= \sum_{i,j \in O \times H \cup H \times I} \left(2 \ \log f_i'(s_i) \ + \ 2 \ \log y^j \ + \ \log \sum_{k \in O} \left(\frac{\partial y^k}{\partial y^i} \right)^2 \right) = \\ & 2 \sum_{i \in O \cup H} \text{fan-in}(i) \log f_i'(s_i) \ + \ 2 \sum_{j \in H \cup I} \text{fan-out}(j) \log y^j \ + \\ & \sum_{i \in O \cup H} \text{fan-in}(i) \log \sum_{k \in O} \left(\frac{\partial y^k}{\partial y^i} \right)^2 , \end{aligned}$$

where fan-in(i) (fan-out(i)) denotes the number of incoming (outgoing) weights of unit i.

$T1$ makes (1) unit activations decrease to zero in proportion to their fan-outs, (2) first-order derivatives of activation functions decrease to zero in proportion to their fan-ins, and (3) the influence of units on the output decrease to zero in proportion to the unit's fan-in. For a detailed analysis see Hochreiter and Schmidhuber (1997a). $T1$ is the reason why low-complexity (or simple) CFs are preferred.

Sparseness. Point (1) above favors sparse hidden unit activations (here: few active components); point (2) favors non-informative hidden unit activations hardly affected by small input changes. Point (3) favors sparse hidden unit activations in the sense that "few hidden units contribute to producing the output". In particular, sigmoid hidden units with activation function $\frac{1}{1+\exp(-x)}$ favor near-zero activations.

3.2 Second term favors few, separated, common component functions.

The term

$$T2 \quad := \quad W \log \sum_{k \in O} \left(\sum_{i,j \in O \times H \cup H \times I} \frac{\left| \frac{\partial y^k}{\partial w_{ij}} \right|}{\sqrt{\sum_{k \in O} \left(\frac{\partial y^k}{\partial w_{ij}} \right)^2}} \right)^2 ,$$

punishes units with similar influence on the output. We reformulate it:

$$T2 \quad = \quad W \log \left(\sum_{i,j \in O \times H \cup H \times I} \quad \sum_{u,v \in O \times H \cup H \times I} \frac{\sum_{k \in O} \left| \frac{\partial y^k}{\partial w_{ij}} \right| \left| \frac{\partial y^k}{\partial w_{uv}} \right|}{\sqrt{\sum_{k \in O} \left(\frac{\partial y^k}{\partial w_{ij}} \right)^2} \sqrt{\sum_{k \in O} \left(\frac{\partial y^k}{\partial w_{uv}} \right)^2}} \right) .$$

Using

$$\frac{\partial y^k}{\partial w_{ij}} = \frac{\partial y^k}{\partial y^i} \frac{\partial y^i}{\partial w_{ij}} ,$$

this can be rewritten as

$$T2 \quad = \quad W \log \left(\sum_{i,j \in O \times H \cup H \times I} \quad \sum_{u,v \in O \times H \cup H \times I} \frac{\sum_{k \in O} \left| \frac{\partial y^k}{\partial y^i} \right| \left| \frac{\partial y^k}{\partial y^u} \right|}{\sqrt{\sum_{k \in O} \left(\frac{\partial y^k}{\partial y^i} \right)^2} \sqrt{\sum_{k \in O} \left(\frac{\partial y^k}{\partial y^u} \right)^2}} \right) .$$

For $i \in O$

$$\frac{\left| \frac{\partial y^k}{\partial y^i} \right|}{\sqrt{\sum_{k \in O} \left(\frac{\partial y^k}{\partial y^i} \right)^2}} = 1$$

holds. We obtain

$$T2 \quad = \quad W \log \left(|O| \; |O \times H|^2 + |I|^2 \sum_{k \in O} \sum_{i \in H} \sum_{u \in H} \frac{\left| \frac{\partial y^k}{\partial y^i} \right| \left| \frac{\partial y^k}{\partial y^u} \right|}{\sqrt{\sum_{k \in O} \left(\frac{\partial y^k}{\partial y^i} \right)^2} \sqrt{\sum_{k \in O} \left(\frac{\partial y^k}{\partial y^u} \right)^2}} \right) .$$

We observe: (1) an output unit that is very sensitive with respect to two given hidden units will heavily contribute to $T2$ (compare the numerator in the last term in the brackets of $T2$). (2) This large contribution can be reduced by making both hidden units have large impact on other output units (see denominator in the last term in the brackets of $T2$).

Choice of component functions (CFs). FMS tries to figure out a way of using (1) as few CFs as possible for each output unit (this leads to separation of CFs), while simultaneously (2) using the same CFs for as many output units as possible (common CFs).

SPECIAL CASE: LINEAR OUTPUT ACTIVATION.

Since our targets will usually be in the linear range of a sigmoid output activation function, let us consider the linear case in more detail. Suppose all output units k use the same linear activation function $f_k(x) = Cx$ (where C is a real-valued constant). Then $\frac{\partial y^k}{\partial y^i} = Cw_{ki}$ for hidden unit i. We obtain

$$T2 \quad = \quad W \log \left(|O| \; |O \times H|^2 + |I|^2 \sum_{i \in H} \sum_{u \in H} \frac{\sum_{k \in O} |w_{ki}| \; |w_{ku}|}{||W_i|| \;\; ||W_u||} \right) ,$$

where W_i denotes the outgoing weight vector of unit i with $[W_i]_k := w_{ki}$, $||.||$ the Euclidean vector norm $||x|| = \sqrt{\sum_i x_i^2}$, and $[.]_k$ the kth component of a vector.

Few component functions preferred. We observe that hidden units whose outgoing weight vectors have near-zero weights yield small contributions to $T2$, that is, the number of CFs will get minimized.

Common component functions preferred. Outgoing weight vectors of hidden units are encouraged to have a large effect on the output (see denominator in the last term in the brackets of $T2$). This implies preference of CFs that can be used for generating many or all output components.

CF separation — few relevant CFs per output unit. On the other hand, two hidden units whose outgoing weight vectors do not solely consist of near-zero weights are encouraged to influence the output in different ways by not representing the same input feature (see numerator in the last term in the brackets of $T2$). In fact, FMS punishes not only outgoing weight vectors with same or opposite directions but also vectors obtained by flipping the signs of the weights (multiple reflections from hyperplanes trough the origin and orthogonal to one axis). Hence two units performing redundant tasks, such as both activating some output unit, or one activating it and the other de-activating it, will cause large contributions to $T2$. This encourages separation of CFs and use of few CFs per output unit.

3.3 Low-Complexity Autoassociators

Given some data set, FMS can be used to find a low-complexity autoassociator (AA) whose hidden layer activations code the individual training exemplars. The AA can be split into two modules: one for coding, one for decoding.

Previous autoassociators (AAs). Backprop-trained AAs *without* a narrow hidden bottleneck ("bottleneck" refers to a hidden layer containing fewer units than other layers) typically produce redundant, continuous-valued codes and unstructured weight patterns. Baldi and Hornik (1989) studied linear AAs *with* a hidden layer bottleneck and found that their codes are orthogonal projections onto the subspace spanned by the first principal eigenvectors of a covariance matrix associated with the training patterns. They showed that the mean squared error (MSE) surface has an unique minimum. Nonlinear codes have been obtained by nonlinear bottleneck AAs with more than 3 (e.g., 5) layers, e.g., Kramer (1991), Oja (1991) or DeMers and Cottrell (1993). None of these methods produces sparse, factorial or local codes — instead they produce first principal components or their nonlinear equivalents ("principal manifolds"). We will see that FMS-based AAs yield quite different results.

FMS-based AAs. According to subsections 3.1 and 3.2, because of the low-complexity *coding* aspect the codes tend to (C1) be binary for sigmoid units with activation function $f_i(x) = \frac{1}{1+\exp(-x)}$ ($f'_i(s_i)$ is small for y^i near 0 or 1), (C2) require few separated code components or hidden units (HUs), and (C3) use simple component functions. Because of the low-complexity *de*coding part, codes also tend to (D1) have many HUs near zero and, therefore, be sparsely (or even locally) distributed, (D2) have code components conveying information useful for generating as many output activations as possible. (C1), (C2) and (D2) encourage minimally redundant, binary codes. (C3), (D1) and (D2), however, encourage sparse distributed (local) codes. (C1) – (C3) and (D1) – (D2) lead to codes with simply computable code components (C1, C3) that convey a lot of information (D2), and with as few active code components as possible (C2, D1). *Collectively this makes code components represent simple input features.*

4 EXPERIMENTS

Outline. Section 4.1 provides an overview of the experimental conditions. Section 4.2 uses simple artificial tasks to show how various lococode types (factorial, local, sparse, feature detector-based) depend on input/output properties. The visual coding experiments are divided into two sections: Section 4.3 deals with artificial bars, Section 4.4 with real world images. In Section 4.3 the "true" causes of the input data are known, and we show that LOCOCODE learns to represent them optimally (PCA and ICA do not). In Section 4.4 it generates plausible feature detectors. Finally, in Section 4.5 LOCOCODE is used as a preprocessor for speech data fed into standard backpropagation classifier. This provokes significant performance improvement.

4.1 Experimental Conditions

In all our experiments we associate input data with itself, using an FMS-trained 3-layer autoassociator (AA). Unless stated otherwise we use 700,000 training exemplars, sigmoid hidden units (HUs) with activation function (AF) $\frac{1}{1+\exp(-x)}$, sigmoid output units with AF $\frac{2}{1+\exp(-x)} - 1$, noninput units with an additional bias input, normal weights initialized in $[-0.1, 0.1]$, bias hidden weights with -1.0, λ with 0.5. The HU AFs do make sparseness better recognizable, but the output AFs are fairly arbitrary — linear AFs or those of the HUs will do as well. Targets are scaled to $[-0.7, 0.7]$, except for Task 2.2. Target scaling (1) prevents tiny first order derivatives of output units (which may cause floating point overflows), and (2) allows for proving that the FMS algorithm makes the Hessian entries of output units $\frac{\partial^2 y^k}{\partial w_{ij} \partial w_{uv}}$ decrease where the weight precisions $|\delta w_{ij}|$ or $|\delta w_{uv}|$ increase (Hochreiter and Schmidhuber 1997a).

Parameters and other details.

- learning rate: conventional learning rate for error term E (just like backprop's).
- λ: a positive "regularizer" (hyperparameter) scaling B's influence. λ is computed heuristically as described by Hochreiter and Schmidhuber (1997a).
- $\Delta\lambda$: a value used for updating λ during learning. It represents the absolute change of λ after each epoch.
- E_{tol}: the tolerable mean squared error (MSE) on the training set. It is used for dynamically computing λ, and for deciding when to switch phases in 2-phase learning.
- 2-phase learning speeds up the algorithm: phase 1 is conventional backprop, phase 2 is FMS. We start with phase 1 and switch to phase 2 once $E_a < E_{tol}$, where E_a is the average epoch error. We switch back to phase 1 once $E_a > \gamma \; E_{tol}$. We finish in phase 2. The experimental sections will indicate 2-phase learning by mentioning values of γ.
- Pruning of weights and units: we judge a weight w_{ij} as being pruned if its required precision ($|\delta w_{ij}|$ in Hochreiter and Schmidhuber 1997a) for each input is 100 times lower (corresponding to 2 decimal digits) than the highest precision of the other weights for the same input. A unit is considered pruned if all incoming weights are pruned except for the bias weight, or if all outgoing weights are pruned.

For more details see Hochreiter and Schmidhuber (1997a) or their home pages.

Comparison. In sections 4.3 and 4.4 we compare LOCOCODE to simple variants of "independent component analysis" (ICA, e.g., Jutten and Herault 1991, Cardoso and Souloumiac 1993, Molgedey and Schuster 1994, Comon 1994, Bell and Sejnowski 1995, Amari et al. 1996, Nadal and Parga 1997) and "principal component analysis" (PCA, e.g., Oja 1989). ICA is realized by Cardoso's (1993) JADE (Joint Approximate Diagonalization of Eigen-matrices) algorithm (we used the Matlab JADE version obtained via FTP from sig.enst.fr). JADE is based on whitening and subsequent joint diagonalization of 4th-order cumulant matrices. For PCA and ICA, 1,000 (3,000) training exemplars are used in case of 5×5 (7×7) input fields.

Information content. To measure the information conveyed by the various codes obtained in sections 4.3 and 4.4 we train a standard backprop net on the training set used for code generation. Its inputs are the code components; its task is to reconstruct the original input (for all tasks except for "noisy bars" the original input is scaled such that all input components are in $[-1.0, 1.0]$). The net has as many biased sigmoid hidden units with activation function (AF) $\frac{1}{1+\exp(-x)}$ as there are biased sigmoid output units with AF $\frac{2}{1+\exp(-x)} - 1$. We train it for 5,000 epochs without caring for overfitting. The training set consists of 500 fixed exemplars in the case of 5×5 input fields (bars) and of 5000 in the case of 7×7 input fields (real world images). The test set consists of 500 off-training set exemplars (in the case of real world images we use a separate test image). The average MSE on the test set is used to determine the reconstruction error.

Coding efficiency — discrete codes. Coding efficiency is measured by the average number of bits needed to code a test set input pixel. The code components are scaled to the interval $[0, 1]$ partitioned into 100 discrete intervals — this results in 100 possible discrete values. Assuming independence of the code components we estimate the probability of each discrete code value by Monte Carlo sampling on the training set. To obtain the bits per pixels (Shannon's optimal value) on the test set we divide the sum of the negative logarithms of all discrete code component probabilities (averaged over the test set) by the number of input components.

4.2 EXPERIMENT 1: local, sparse, factorial codes — feature detectors

The following five experiments demonstrate effects of various input representations, data distributions, and architectures according to Table 1. The data always consists of 8 input vectors. Code units are initialized with a negative bias of -2.0.

Constant Parameters. $\Delta\lambda = 1.0$, $\gamma = 2.0$ (2-phase learning).

Experiment 1.1: We use uniformly distributed inputs and 500,000 training examples. *Parameters:* learning rate: 0.1, the "tolerable error" $E_{tol} = 0.1$, *Architecture:* (8-5-8) (8 input units, 5 HUs, 8 output units).

Results: factorial codes. In 7 out of 10 trials, FMS effectively pruned 2 HUs, and produced a *factorial binary code* with statistically independent code components. In 2 trials FMS pruned 2 HUs and produced an almost binary code — with one trinary unit taking on values of 0.0, 0.5, 1.0. In one trial FMS produced a binary code with only one HU being pruned away. Obviously, under certain constraints on the input data, FMS has a strong tendency towards the compact, nonredundant codes advocated by numerous researchers.

Experiment 1.2: See Table 1 for differences to Experiment 1.1. We use 200,000 training examples and more HUs to make clear that in this case fewer units are pruned.

Results: local codes. 10 trials were conducted. FMS always produced a binary code. In 7 trials, only 1 HU was pruned, in the remaining trials 2 HUs. Unlike with standard BP, *almost all inputs almost always were coded in an entirely local manner*, i.e., only one HU was switched on, the others switched off. Recall that local codes were also advocated by many researchers – but they are precisely "the opposite" of the factorial codes from the previous experiment. How can LOCOCODE justify such different codes? How to explain this apparent discrepancy?

Explanation. The reason is: with the different input representation, the additional HUs do not necessarily result in much more additional complexity of the mappings for coding and decoding. The zero-valued inputs allow for low weight precision (low coding complexity) for connections leading to HUs (similarly for connections leading to output units). In contrast to Experiment 1.1 it is possible to describe the i-th possible input by the following feature: "the i-th input component does not equal zero". It can be implemented by a low-complexity component function. This contrasts the features in Experiment 1.1, where there are only 5 hidden units and no zero input components: there it is better to code with as few code components as possible, which yields a factorial code.

Experiment 1.3: like Experiment 1.2 but with *one-dimensional* input. *Parameters:* learning rate: 0.1, $E_{tol} = 0.00004$.

Results: feature detectors. 10 trials were conducted. FMS always produced the following code: one binary HU making a distinction between input values less than 0.5 and input values greater than 0.5, 2 HUs with continuous values, one of which is zero (or one) whenever the binary unit is on, while the other is zero (one) otherwise. All remaining HUs adopt constant values of either 1.0 or 0.0, thus being essentially pruned away. The binary unit serves as a binary *feature detector*, grouping the inputs into 2 classes.

Lococode recognizes the causes. The data of Experiment 1.3 may be viewed as being generated as follows: (1) first choose with uniform probability a value from $\{0.0, 0.75\}$; then (2) choose one from $\{0.05, 0.1, 0.15, 0.2\}$; then (3) add the two values. The first cause of the data is recognized perfectly but the second is divided among two code components, due to the non-linearity of the output unit: adding to 0 is different from adding to 0.75 (consider the first order derivatives).

Experiment 1.4: like Experiment 1.1 but with nonuniformly distributed inputs. *Parameters:* learning rate: 0.005, $E_{tol} = 0.01$.

Results: sparse codes. In 4 out of 10 trials, FMS found a binary code (no HUs pruned). In 3 trials: a binary code with one HU pruned. In one trial: a code with one HU removed, and a trinary unit adopting values of 0.0, 0.5, 1.0. In 2 trials: a code with one pruned HU and 2 trinary HUs. Obviously, with this set-up, FMS prefers codes known as *sparse distributed representations.* Inputs with higher probability are coded by fewer active code components than inputs with lower probability. Typically, inputs with probability $\frac{1}{4}$ lead to one active code component, inputs with probability $\frac{1}{8}$ to two, and others to three.

Explanation. Why is the result different from Experiment 1.1's? To achieve equal error contributions to all inputs, the weights for coding/decoding highly probable inputs have to be given with higher precision than the weights for coding/decoding inputs with low probability: the input distribution from Experiment 1.1 will result in a more complex network. The next experiment will make this effect even more pronounced.

Experiment 1.5: like Experiment 1.4, but with architecture (8-8-8).

Results: sparse codes. In 10 trials, FMS always produced binary codes. In 2 trials only 1 HU was pruned. In 1 trial 3 units were pruned. In 7 trials 2 units were pruned. Unlike with standard BP, *almost all inputs almost always were coded in a sparse, distributed manner:* typically, 2 HUs were switched on, the others switched off, and most HUs responded to exactly 2 different input patterns. The mean probability of a unit being switched on was 0.28, and the probabilities of different HUs being switched on tended to be equal.

Table 1 provides an overview over Experiments 1.1 — 1.5.

Conclusion. FMS always finds codes quite different from standard BP's rather unstructured ones. It tends to discover and represent the underlying causes. Usually the resulting lococode is sparse and based on informative feature detectors. Depending on properties of the data it may become factorial or local. This suggests that LOCOCODE may represent a general principle of unsupervised learning subsuming previous, COCOF-based approaches.

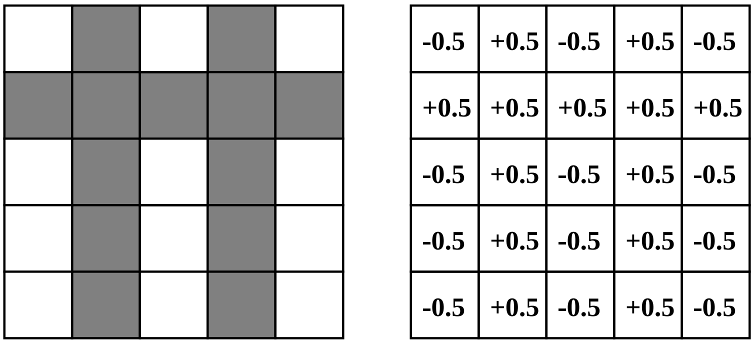
Feature-based lococodes automatically take into account input/output properties (binary?, local?, input probabilities?, noise?, number of zero input components?).

4.3 EXPERIMENT 2: Independent Bars

Task 2.1 — adapted from Dayan and Zemel (1995), see also Földiák (1990), Zemel (1993), Saund (1995), but more difficult (compare M. Baumgartner's 1996 diploma thesis). The input is a 5×5 pixel grid with horizontal and vertical bars at random, independent positions. See Figure 1 for an example. The task is to extract the independent features (the bars). According to Dayan and Zemel (1995), even a simpler variant (where vertical and horizontal bars may not be mixed in the same input) is not trivial:

Exp.	input	input values	input	architecture	code	result
	coding		distribution		components	
1.1	local	0.2, 0.8	uniform	8-5-8	3	factorial code
1.2	local	0.0, 1.0	uniform	8-8-8	7	local code
1.3	dense	0.05, 0.1, 0.15, 0.2, 0.8, 0.85, 0.9, 0.95	uniform	1-8-1	3	feature detectors
1.4	local	0.2, 0.8	$\frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}$	8-5-8	4	sparse code
1.5	local	0.2, 0.8	$\frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}$	8-8-8	6	sparse code

Table 1: *Overview over experiments 1.1 – 1.5: type of input coding, possible values of input components, distribution of the 8 input vectors, architecture in the form "input-hidden-output" units, nature of the resulting lococode (which mainly depends on the nature of the input data).*



-0.5	+0.5	-0.5	+0.5	-0.5
+0.5	+0.5	+0.5	+0.5	+0.5
-0.5	+0.5	-0.5	+0.5	-0.5
-0.5	+0.5	-0.5	+0.5	-0.5
-0.5	+0.5	-0.5	+0.5	-0.5

Figure 1: *Task 2.1: example of partly overlapping bars. The 2nd and the 4th vertical bar and the 2nd horizontal bar are switched on simultaneously. Left: the corresponding input values.*

> "Although it might seem like a toy problem, the 5 × 5 bar task with only 10 hidden units turns out to be quite hard for all the algorithms we discuss. The coding cost of making an error in one bar goes up linearly with the size of the grid, so at least one aspect of the problem gets *easier* with large grids."

We will see that even difficult variants of this task are not hard for Lococode.

Training and testing. Each of the 10 possible bars appears with probability $\frac{1}{5}$. In contrast to Dayan and Zemel's set-up (1995) we allow for bar type mixing. This makes the task harder (Dayan and Zemel 1995, p. 570). To test Lococode's ability to reduce redundancy, we use many more HUs (namely 25) than the required minimum of 10. Dayan and Zemel report that an AA trained without FMS (and more than 10 HUs) "consistently failed". This result has been confirmed by Baumgartner (1996).

For each of the 25 pixels there is an input unit. Input units that see a pixel of a bar take on activation 0.5, others -0.5. See Figure 1 for an example. Following Dayan and Zemel (1995), the net is trained on 500 randomly generated patterns (there may be pattern repetitions). Learning is stopped after 5,000 epochs. We say that a pattern is processed correctly if the absolute error of all output units is below 0.3.

Details. *Parameters:* learning rate: 1.0, $E_{tol} = 0.16$, $\Delta\lambda = 0.001$. *Architecture:* (25-25-25).

Results: factorial (but sparse) codes. Training MSE is 0.11 (average over 10 trials). The net generalizes well: only one of the test patterns is not processed correctly. 15 of the 25 HUs

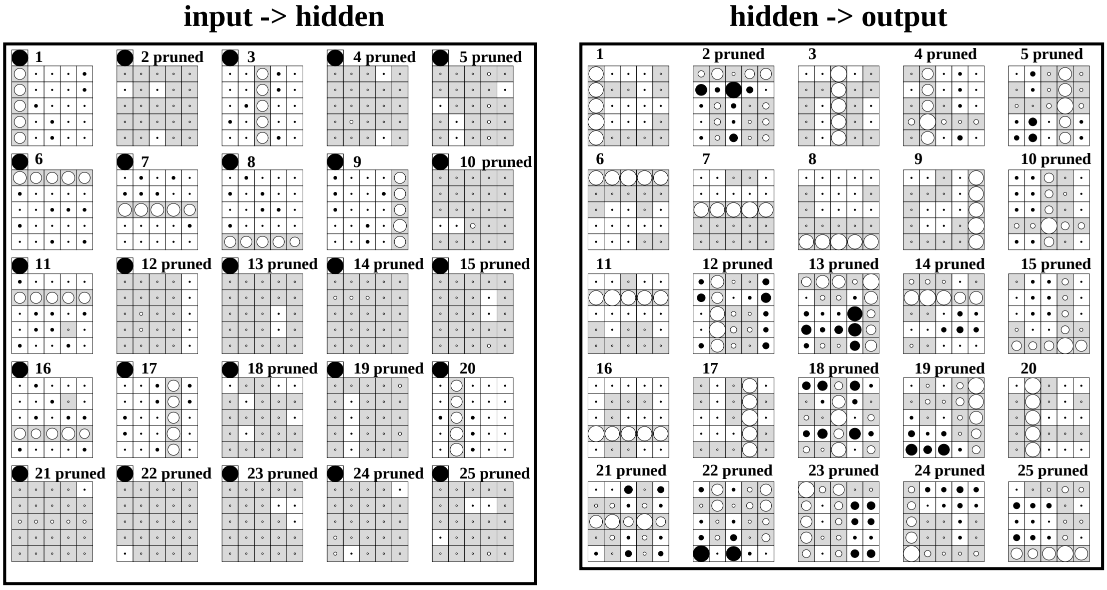


Figure 2: *Task 2.1 (independent bars). Left:* LOCOCODE*'s input-to-hidden weights. Right: hidden-to-output weights. See text for visualization details.*

are indeed automatically pruned. All remaining HUs are binary: LOCOCODE finds an optimal factorial code which exactly mirrors the pattern generation process. Since the expected number of bars per input is 2, the code is also sparse.

For each of the 25 HUs, Figure 2 (left) shows a 5 × 5 square depicting 25 typical post-training weights on connections from 25 inputs (right: to 25 outputs). White (black) circles on gray (white) background are positive (negative) weights. The circle radius is proportional to the weight's absolute value. Figure 2 (left) also shows the bias weights (on top of the squares' upper left corners). The circle representing some HU's maximal absolute weight has maximal possible radius (circles representing other weights are scaled accordingly).

Backprop fails. For comparison we run this task with conventional BP with 25, 15 and 10 HUs. With 25 (15, 10) HUs the reconstruction error is 0.19 (0.24, 0.31). Backprop does not prune any units; the resulting weight patterns are highly unstructured, and the underlying input statistics are not discovered.

PCA and ICA. We tried both 10 and 15 components. Figure 3 shows results. PCA produces an unstructured and dense code, ICA-10 an almost sparse code where some sources are recognizable but not separated. ICA-15 finds a dense code and no sources. ICA/PCA codes with 10 components convey the same information as 10-component lococodes. The higher reconstruction errors for PCA-15 and ICA-15 are due to overfitting (the backprop net over-specializes on the training set).

LOCOCODE can exploit the advantages of sigmoid output functions and is applicable to nonlinear signal mixtures. PCA and ICA, however, are limited to linear source superpositions. Since we allow for mixing of vertical and horizontal bars, the bars do not add linearly, thus exemplifying a major characteristic of real visual inputs. This contributes to making the task hard for PCA and ICA.

Task 2.2 (noisy bars). Like Task 2.1 except for additional noise: bar intensities vary in $[0.1, 0.5]$; input units that see a pixel of a bar are activated correspondingly (recall the constant intensity 0.5 in Task 2.1), others adopt activation -0.5. We also add Gaussian noise with variance 0.05 and mean 0 to each pixel. Figure 4 shows some training exemplars generated in this way. The task is adapted from Hinton et al. (1995) and Hinton and Ghahramani (1997) but more difficult because vertical and horizontal bars may be mixed in the same input.

Details. Training, testing, coding and learning are as in Task 2.1, except that $E_{tol} = 2.5$ and

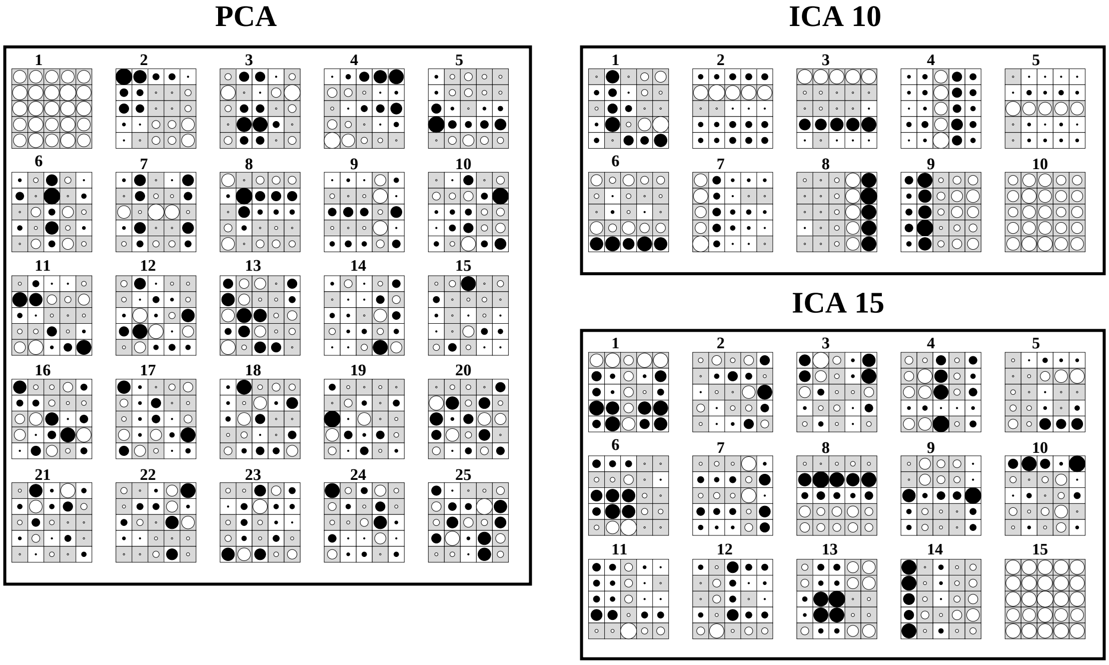


Figure 3: *Task 2.1 (independent bars). PCA and ICA: weights to code components (ICA with 10 and 15 components). ICA-10 does make some sources recognizable, but does not achieve lococode quality.*

$\Delta\lambda = 0.01$. E_{tol} is set to 2 times the expected minimal squared error: $E_{tol} = 2$ (number of inputs) $\sigma^2 = 2 * 25 * 0.05 = 2.5$. To achieve consistency with Task 2.1, the target pixel value is 1.4 times the input pixel value (compare Task 2.1: $0.7 = 1.4 * 0.5$). All other learning parameters are like in Task 2.1.

Results. Training MSE is 2.5 (averaged over 10 trials); the net generalizes well. 15 of the 25 HUs are pruned away. Again Lococode extracts an optimal (factorial) code which exactly mirrors the pattern generation process. Due to the bar intensity variations the remaining HUs are not binary as in Task 2.1. Figure 5 depicts typical weights to and from HUs.

PCA and ICA. Figure 6 shows results comparable to those of Task 2.1. PCA codes and ICA-15 codes are unstructured and dense. ICA-10 codes, however, are almost sparse — some sources are recognizable. They are not separated though. We observe that PCA/ICA codes with 10 components convey as much information as 10-component lococodes. The lower reconstruction error for PCA-15 and ICA-15 is due to information about the current noise conveyed by the additional code components (we reconstruct noisy inputs).

Conclusion. Lococode solves a hard variant of the standard "bars" problem. It discovers the underlying statistics and extracts the essential, statistically independent features, even in presence of noise. Standard BP AAs accomplish none of these feats (Dayan and Zemel, 1995) — this has been confirmed by additional experiments conducted by ourselves. ICA and PCA also fail to extract the true input causes and the optimal features.

Lococode achieves success solely by reducing information-theoretic (de)coding costs. Unlike previous approaches, it does not depend on explicit terms enforcing independence (e.g., Schmidhuber 1992), zero mutual information among code components (e.g., Linsker 1988, Deco and Parra 1994), or sparseness (e.g., Field 1994, Zemel and Hinton 1994, Olshausen and Field 1996, Zemel 1993, Hinton and Ghahramani 1997).

Lococode vs. ICA. Like recent simple methods for "independent component analysis" (ICA, e.g., Cardoso and Souloumiac 1993, Bell and Sejnowski 1995, Amari et al. 1996) Lococode untangles mixtures of independent data sources. Unlike these methods, however, it does not need

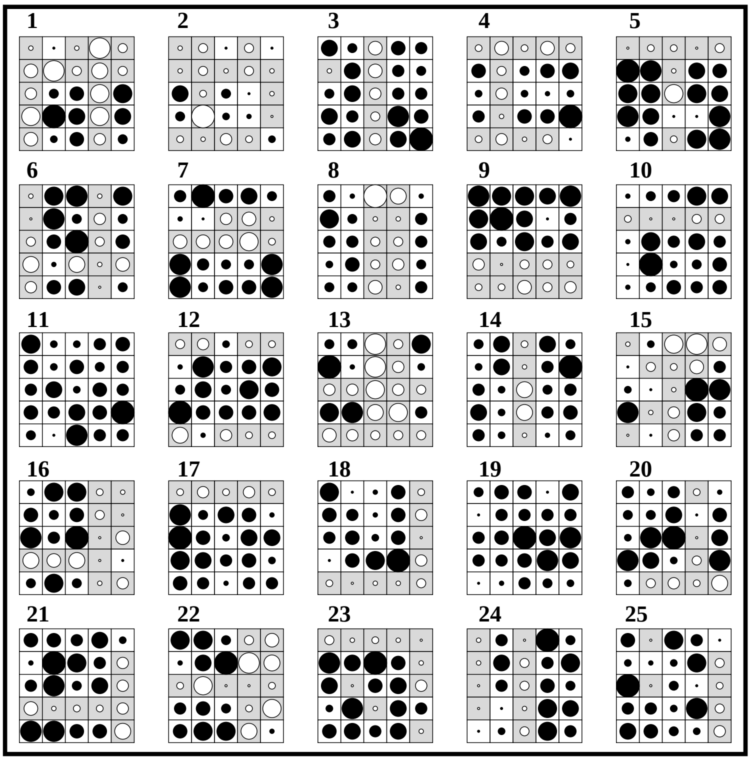


Figure 4: *Task 2.2 — noisy bars examples: 25* 5×5 *training inputs, depicted similarly to the weights in previous figures.*

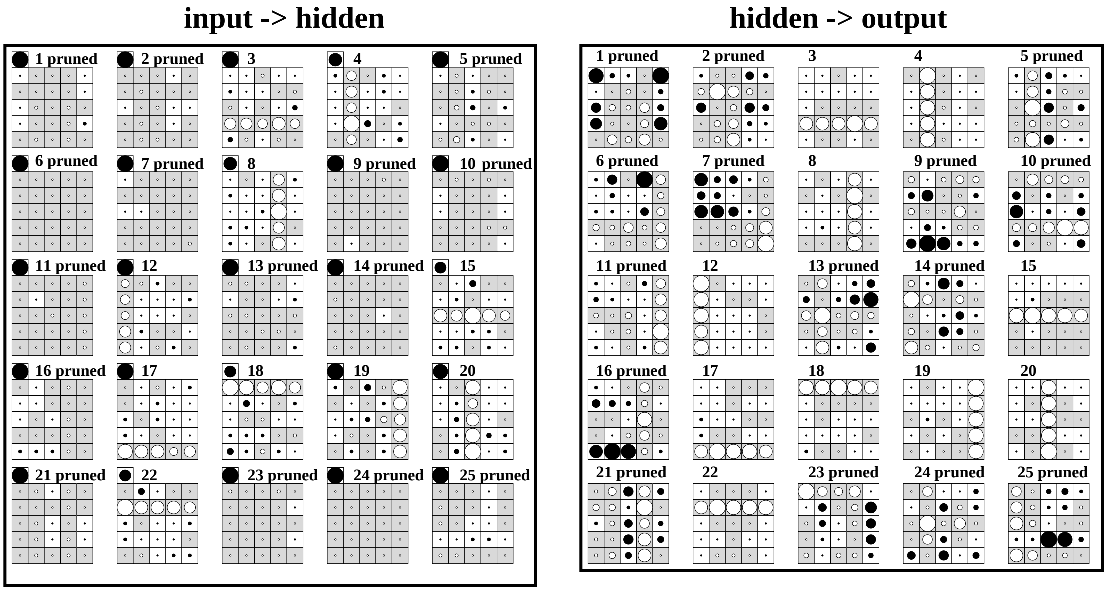


Figure 5: *Task 2.2 (independent noisy bars). Left:* Lococode*'s input-to-hidden weights. Right: hidden-to-output weights.*

to know in advance the number of such sources — like "predictability minimization" (a nonlinear ICA approach — Schmidhuber 1992), it simply prunes away superfluous code components.

In many visual coding applications few sources determine the value of a given output (input)

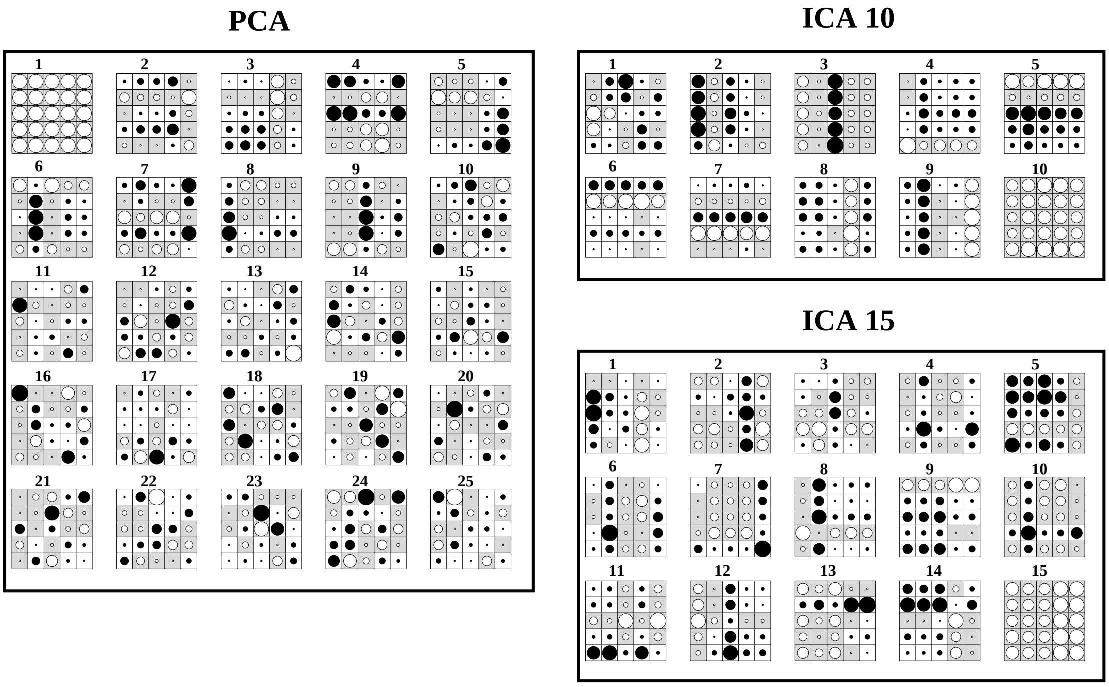


Figure 6: *Task 2.2 (independent noisy bars). PCA and ICA: weights to code components (ICA with 10 and 15 components). Only ICA-10 codes extract a few sources, but they do not achieve the quality of lococodes.*

component, and the sources are easily computable from the input. Here Lococode outperforms simple ICA because it minimizes the number of low-complexity sources responsible for each output component. It may be less useful for discovering input causes that can only be represented by high-complexity input transformations, or for discovering many features (causes) collectively determining single input components (as, e.g., in acoustic signal separation). In such cases ICA does not suffer from the fact that each source influences each input component and none is computable by a low-complexity function.

4.4 EXPERIMENT 3: More Realistic Visual Data

Task 3.1. As in Experiment 2 the goal is to extract features from visual data. The input data is more realistic though — we use the aerial shot of a village.

Details. Figure 7 shows two images with 150 × 150 pixels, each taking on one of 256 gray levels. 7×7 pixels subsections from the left hand side (right hand side) image are randomly chosen as training inputs (test inputs), where gray levels are scaled to input activations in $[-0.5, 0.5]$. Training stop: after 150,000 training examples. *Parameters:* learning rate: 1.0, $E_{tol} = 3.0$, $\Delta\lambda = 0.05$. *Architecture:* (49-25-49). $E_{tol} = 3.0$,

Image structure. The image is mostly dark except for certain white regions. In a preprocessing stage we map pixel values above 119 to 255 (white) and pixel values below 120 to 9 different gray values. The largest reconstruction errors will be due to absent information about white pixels. Our receptive fields are too small to capture structures such as lines (streets).

Results: sparse codes, on-center-off-surrounds. 6 trials led to similar results (6 trials seem sufficient due to tiny variance). Only 9 to 11 HUs survive. They indeed reflect the structure of the image (compare the preprocessing stage): (1) Informative white spots are captured by on-center-off-surround HUs. (2) Since the image is mostly dark (this also causes the off-surround effect), all output units are negatively biased. (3) Since most bright spots are connected (most white pixels are surrounded by white pixels), output/input units near an active output/input unit

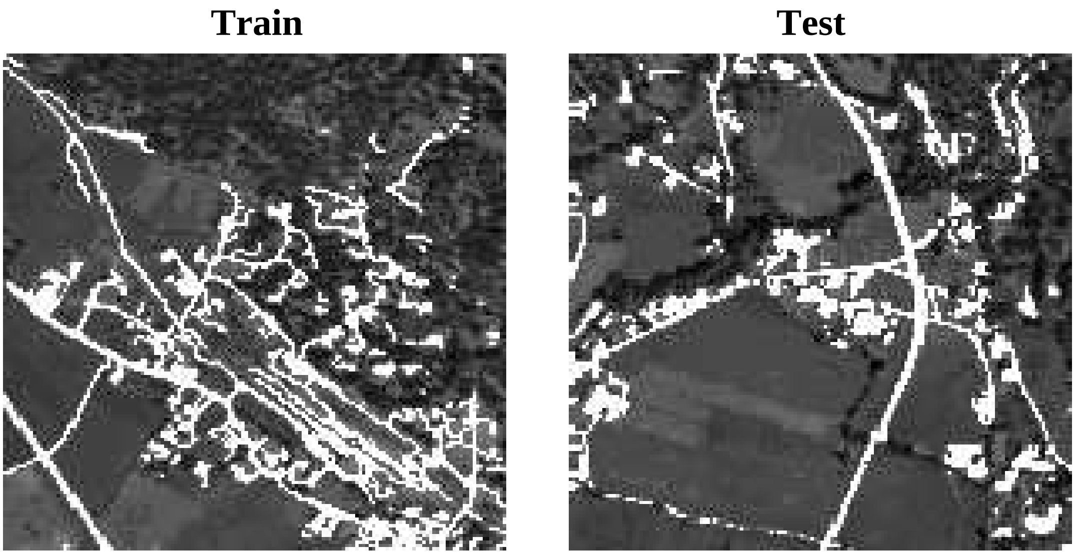


Figure 7: *Task 3.1 — village image. Image sections used for training (left) and testing (right).*

tend to be active, too (positive weight strength decreases as one moves away from the center). (4) The entire input is covered by on-centers of surviving units — all white regions in the input will be detected. (5) The code is sparse: few surviving white-spot-detectors are active simultaneously because most inputs are mostly dark. Figure 8 depicts typical weights on connections to and from HUs (output units are negatively biased). 10 units survive.

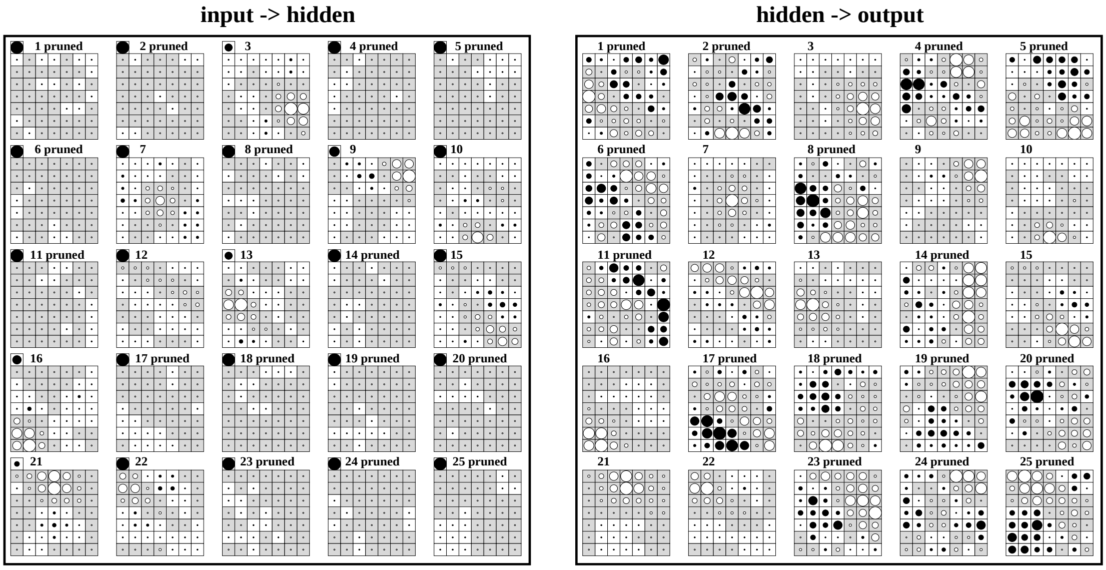


Figure 8: *Task 3.1 (village). Left:* Lococode*'s input-to-hidden weights. Right: hidden-to-output weights. Most units are essentially pruned away.*

PCA and ICA. Figure 9 shows results for PCA and ICA. PCA-10 codes and ICA-10 codes are about as informative as 10 component lococodes (ICA-10 a bit more and PCA-10 less). PCA-15 codes convey no more information: Lococode and ICA suit the image structure better. Because there is no significant difference between subsequent PCA eigenvalues after the 8th, Lococode did find an appropriate number of code components.

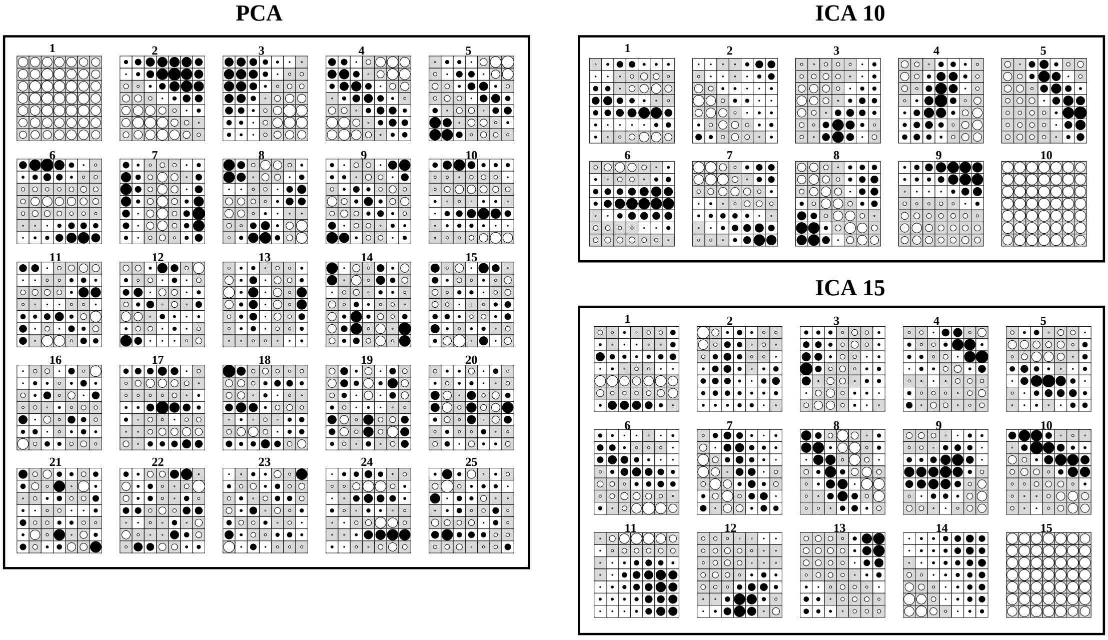


Figure 9: *Task 3.1 (village). PCA and ICA (with 10 and 15 components): weights to code components.*

Figure 10 depicts the reconstructed test image codes with code components mapped to 100 intervals. Reconstruction is limited to 147 × 147 pixels of the image covered by 21 × 21 input fields of size 7 × 7 (the 3 remaining stripes of pixels on the right and lower border are black). Code efficiency and reconstruction error averaged over the test image are given in Table 2. The bits required for coding the 147 × 147 section of the test image are: Lococode: 14,108, ICA-10: 16,255, PCA-10: 16,312 and ICA-15: 23,897.

Task 3.2. Like Task 3.1, but the inputs stem from a 150 × 150 pixels section of an image of wood cells (Figure 11: left: training image, right: test image). $E_{tol} = 1.0$, $\Delta\lambda = 0.01$. Training stop: after 250,000 training examples. All other parameters are like in Task 3.1.

Image structure. The image consists of elliptic cells of various sizes. Cell interiors are bright; cell borders dark.

Results. 4 trials led to similar results (4 trials seem sufficient due to tiny variance). Bias weights to HUs are negative. To activate some HU, its input must match the structure of the incoming weights to cancel the inhibitory bias. 9 to 11 units survive. They are obvious feature detectors and can be characterized by the positions of the centers of their on-center-off-surround structures relative to the input field. They are specialized on detecting the following cases: the on-center is north, south, west, east, northeast, northwest, southeast, southwest of a cell, or centered on a cell, or between cells. Hence the entire input is covered by position-specialized on-centers.

Figure 12 depicts typical weights on connections to and from HUs. Typical feature detectors: unit 20 detects a southeastern cell; unit 21 western and eastern cells; unit 23 cells in the northwest and southeast corners.

PCA and ICA. Figure 13 shows results for PCA and ICA. PCA-11 codes and ICA-11 are about as informative as the 11 component lococode (ICA-11 a little less and PCA-11 more). It seems that both Lococode and ICA detect relevant sources: the positions of the cell interiors (and cell borders) relative to the input field. Gaps in the PCA eigenvalues occur between the 10th and the 11th, and between the 15th and the 16th. Lococode essentially found the first gap.

Task 3.3. Like Task 3.1 — but now we use images of *striped* piece of wood. See Figure 14. $E_{tol} = 0.1$. Training stop: after 300,000 training examples. All other parameters are like in Task 3.1.

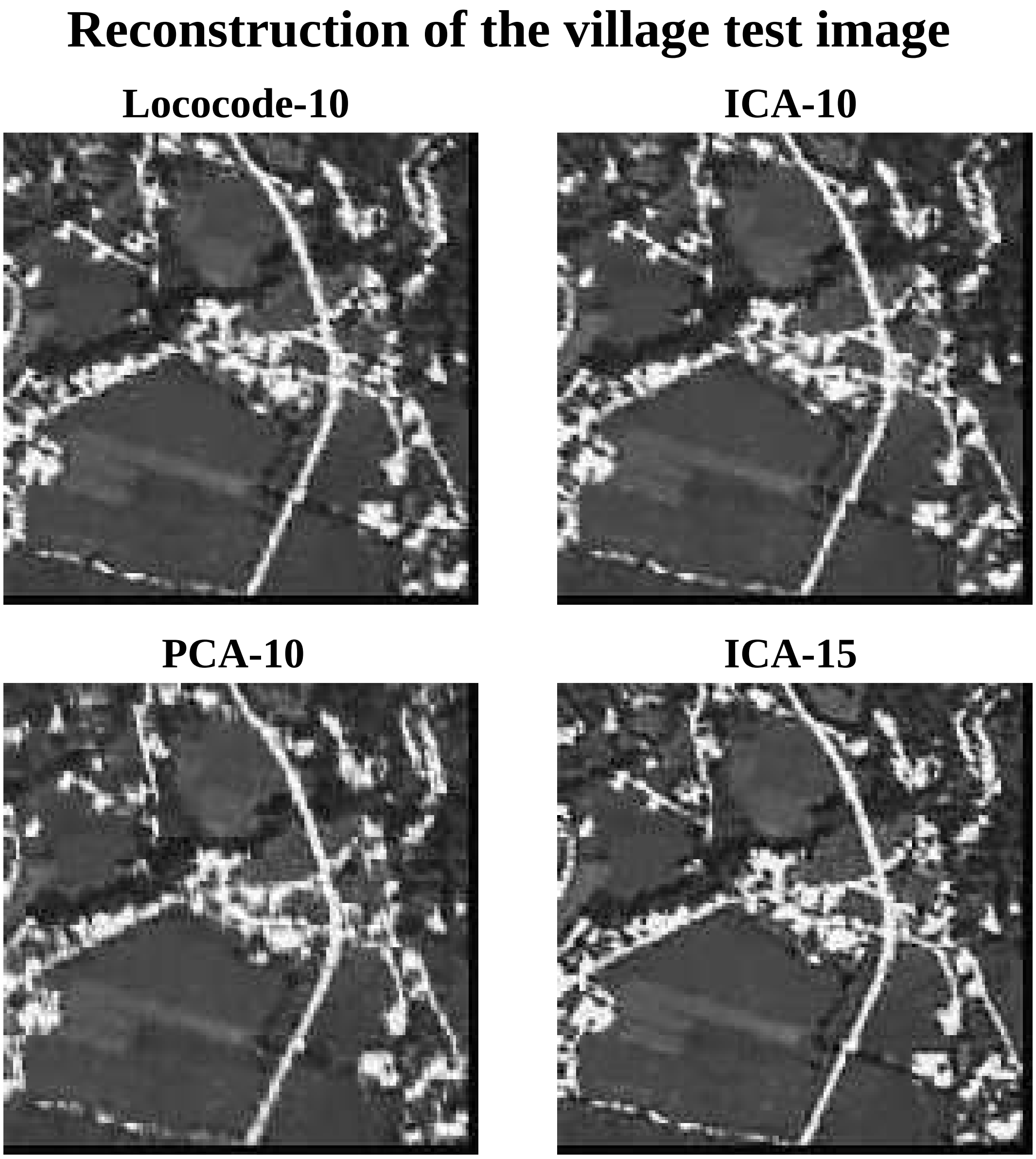


Figure 10: *Task 3.1 (village).* 147 × 147 *pixels of test images reconstructed by* Lococode, *ICA-10, PCA-10 and ICA-15. Code components are mapped to 100 discrete intervals. The second best method (ICA-10) requires 15 % more bits than* Lococode.

Image structure. The image consists of dark vertical stripes on a brighter background.

Results. 4 trials led to similar results Only 3 to 5 of the 25 HUs survive and become obvious feature detectors, now of a different kind: they detect whether their receptive field covers a dark stripe to the left, to the right, or in the middle.

Figure 15 depicts typical weights on connections to and from HUs. Example feature detectors: unit 6 detects a dark stripe to the left, unit 11 a dark stripe in the middle, unit 15 dark stripes left and right, unit 25 a dark stripe to the right.

PCA and ICA. See Figure 16. PCA-4 codes and ICA-4 codes are about as informative as 4-component lococodes. Component structures of PCA/ICA codes and lococodes are very similar:

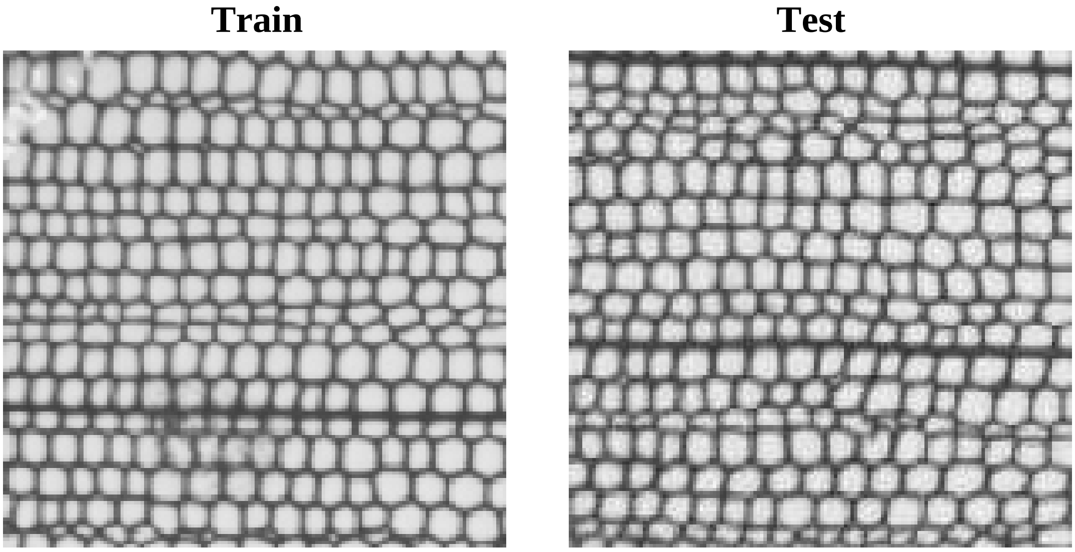


Figure 11: *Task 3.2 — wood cells. Image sections used for training (left) and testing (right).*

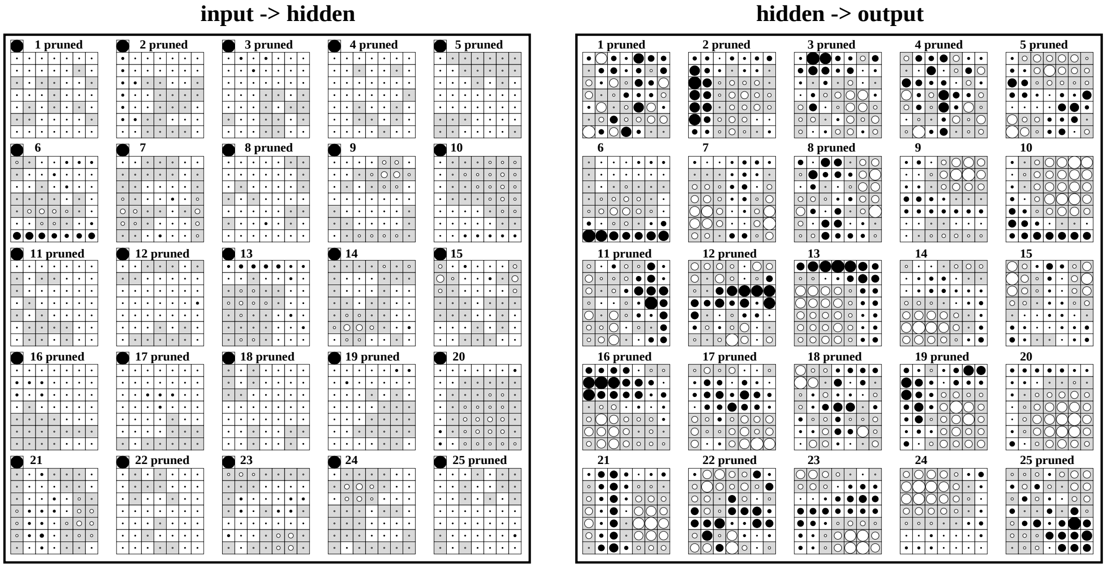


Figure 12: *Task 3.2 (cells). Left:* Lococode*'s input-to-hidden weights. 11 units survive.*

all detect the positions of dark stripes relative to the input field. Gaps in the PCA eigenvalues occur between 3rd and 4th, 4th and 5th, 5th and 6th. Lococode automatically extracts about 4 relevant components.

4.4.1 Overview over experiments 2 and 3

Table 2 shows that most lococodes and some ICA codes are sparse, while most PCA codes are dense. Assuming that each visual input consists of many components collectively describable by few input features, Lococode seems preferable.

Conclusion. Unlike standard BP-trained AAs, FMS-trained AAs generate highly structured sensory codes. FMS automatically prunes superfluous units. PCA experiments indicate that the remaining code units suit the various coding tasks well. Taking into account statistical prop-

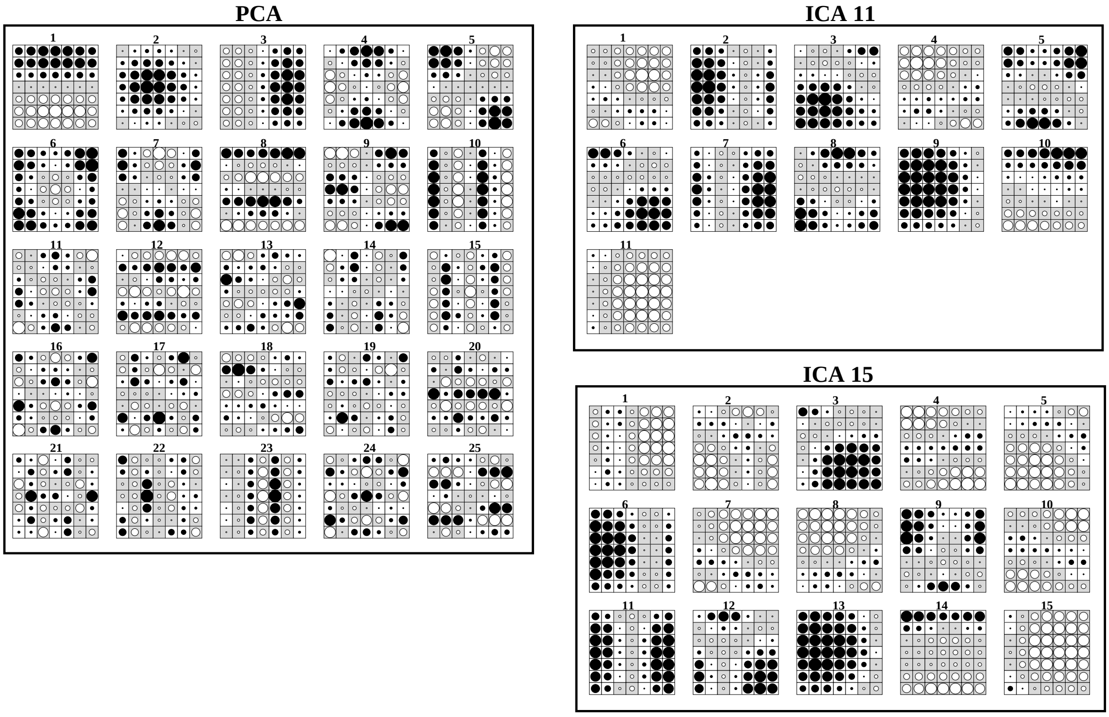


Figure 13: *Task 3.2 (cells). PCA and ICA (with 11 and 15 components): weights to code components.*

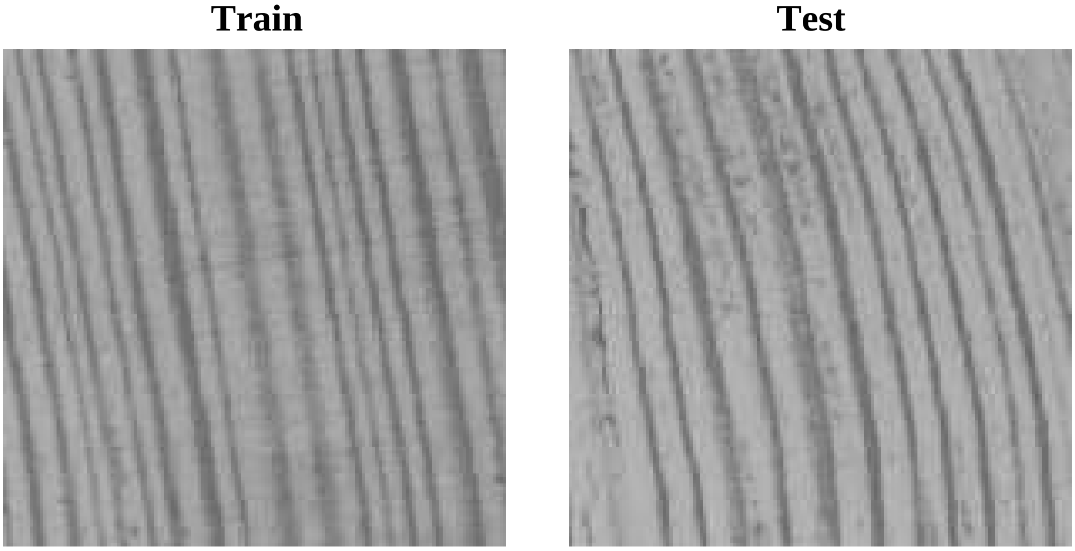


Figure 14: *Task 3.3 — striped wood. Image sections used for training (left) and testing (right).*

erties of the visual input data, Lococode generates appropriate feature detectors such as the familiar on-center-off-surround and bar detectors. It also produces biologically plausible sparse codes (standard AAs do not). FMS's objective function, however, does *not* contain explicit terms enforcing such codes (this contrasts previous methods, e.g., Olshausen and Field 1996).

The experiments show that equally-sized PCA codes, ICA codes, and lococodes convey ap-

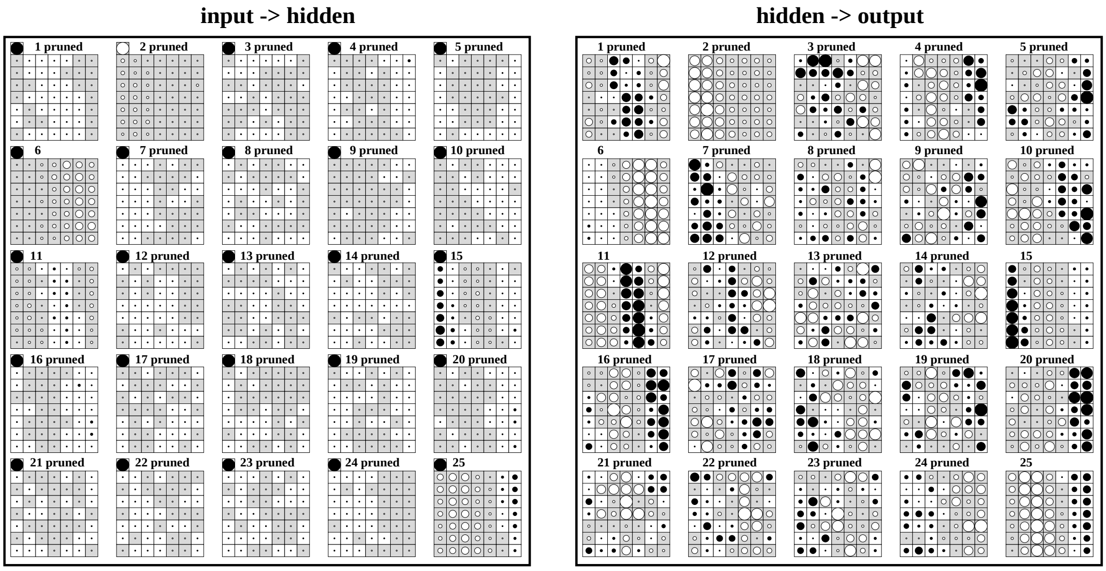


Figure 15: *Task 3.3 (stripes). Left:* Lococode*'s input-to-hidden weights. 4 units survive.*

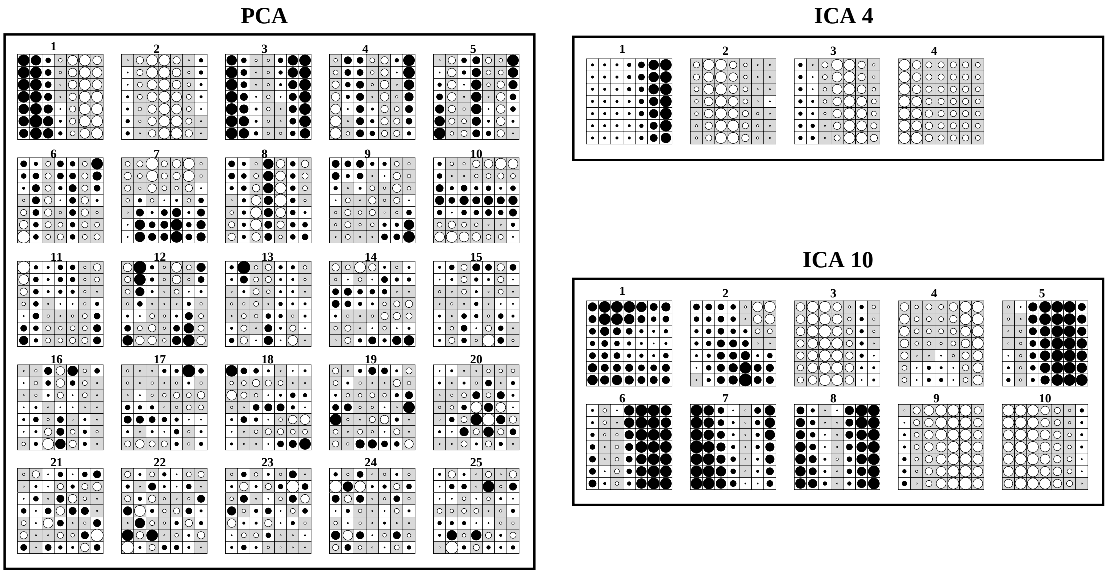


Figure 16: *Task 3.3 (stripes). PCA and ICA (with 11 and 15 components).*

proximately the same information. Lococode, however, codes with fewer bits per pixel. Unlike PCA and ICA, it determines the code size automatically. Some of the feature detectors obtained by Lococode are similar to those found by ICA. In cases where we *know* the true input causes, however, Lococode does find them whereas ICA does not.

4.5 EXPERIMENT 4: vowel recognition

Lococodes cannot only be justified by reference to previous ideas on what's a "desirable" code. Next we will show that they can help to achieve superior generalization performance on a standard supervised learning benchmark problem. This section's focus on speech data also illustrates Lococodes's versatility: its applicability is not limited to visual data.

Exp.	input field	method	# code comp.	reconst. error	code type	code efficency – reconst.
bars	5×5	LOC	10	0.08	sparse (factorial)	1.22 – 0.09
bars	5×5	ICA	10	0.08	almost sparse	1.44 – 0.09
bars	5×5	PCA	10	0.09	dense	1.43 – 0.09
bars	5×5	ICA	15	0.09	dense	2.19 – 0.10
bars	5×5	PCA	15	0.16	dense	2.06 – 0.16
noisy bars	5×5	LOC	10	1.05	sparse (factorial)	1.37 – 1.06
noisy bars	5×5	ICA	10	1.02	almost sparse	1.68 – 1.03
noisy bars	5×5	PCA	10	1.03	dense	1.66 – 1.04
noisy bars	5×5	ICA	15	0.71	dense	2.50 – 0.73
noisy bars	5×5	PCA	15	0.72	dense	2.47 – 0.72
village image	7×7	LOC	10	8.29	sparse	0.69 – 8.29
village image	7×7	ICA	10	7.90	dense	0.80 – 7.91
village image	7×7	PCA	10	9.21	dense	0.80 – 9.22
village image	7×7	ICA	15	6.57	dense	1.20 – 6.58
village image	7×7	PCA	15	8.03	dense	1.19 – 8.04
wood cell image	7×7	LOC	11	0.84	sparse	0.96 – 0.86
wood cell image	7×7	ICA	11	0.87	sparse	0.98 – 0.89
wood cell image	7×7	PCA	11	0.72	almost sparse	0.96 – 0.73
wood cell image	7×7	ICA	15	0.36	sparse	1.32 – 0.39
wood cell image	7×7	PCA	15	0.33	dense	1.28 – 0.34
wood piece image	7×7	LOC	4	0.83	almost sparse	0.39 – 0.84
wood piece image	7×7	ICA	4	0.86	almost sparse	0.40 – 0.87
wood piece image	7×7	PCA	4	0.83	almost sparse	0.40 – 0.84
wood piece image	7×7	ICA	10	0.72	almost sparse	1.00 – 0.76
wood piece image	7×7	PCA	10	0.53	almost sparse	0.91 – 0.54

Table 2: *Overview over experiments 2 and 3: name of experiment, input field size, coding method, number of relevant code components (code size), reconstruction error, nature of code observed on the test set. PCA's and ICA's code sizes are prewired.* Lococode*'s, however, are found automatically. The final column shows coding efficiency measured in bits per pixels (for code components mapped to 100 discrete intervals) and reconstruction error (for this discrete code).* Lococode *exhibits superior coding efficiency.*

Task. We recognize vowels, using vowel data from Scott Fahlman's CMU benchmark collection (see also Robinson 1989). There are 11 vowels and 15 speakers. Each speaker spoke each vowel 6 times. Data from the first 8 speakers is used for training. The other data is used for testing. This means 528 frames for training and 462 frames for testing. Each frame consists of 10 input components obtained by low pass filtering at 4.7kHz, digitized to 12 bits with a 10 kHz sampling rate. A twelfth order linear predictive analysis was carried out on six 512 sample Hamming-windowed segments from the steady part of the vowel. The reflection coefficients were used to calculate 10 log area parameters, providing the 10 dimensional input space.

Coding. The training data is coded using an FMS AA. Architecture: (10-30-10). The input components are linearly scaled in [-1,1]. The AA is trained with 10^7 pattern presentations. Then its weights are frozen.

Classification. From now on, the vowel codes across all nonconstant HUs are used as inputs for a conventional supervised BP classifier, which is trained to recognize the vowels from the code. The classifier's architecture is $((30 - c)$-11-11), where c is the number pruned HUs in the AA. The hidden and output units are sigmoid with activation function $\frac{2}{1+\exp(-x)} - 1$, and receive an additional bias input. The classifier is trained with another 10^7 pattern presentations.

Parameters. AA net: learning rate: 0.02, $E_{tol} = 0.015$, $\Delta\lambda = 0.2$, $\gamma = 2.0$. Backprop classifier: learning rate: 0.002.

Overfitting. We confirm Robinson's results: the classifier tends to overfit when trained by simple BP — during learning, the test error rate first decreases and then increases again.

Comparison. We compare: *(1) Various neural nets* (see Table 1). *(2) Nearest neighbor:* classifies an item as belonging to the class of the closest example in the training set (using Euclidean distance). *(3) LDA:* linear discriminant analysis. *(4) Softmax:* observation assigned to class with best fit value. *(5) QAD:* quadratic discriminant analysis (observations are classified as belonging to the class with closest centroid, using Mahalanobis distance based on the class-specific covariance matrix). *(6) CART:* classification and regression tree (coordinate splits and default input parameter values are used). *(7) FDA/BRUTO:* flexible discriminant analysis using additive models with adaptive selection of terms and splines smoothing parameters. BRUTO provides a set of basis functions for better class separation. *(8) Softmax/BRUTO:* best fit value for classification using BRUTO. *(9) FDA/MARS:* FDA using multivariate adaptive regression splines. MARS builds a basis expansion for better class separation. *(10) Softmax/MARS:* best fit value for classification using MARS. *(11)* Lococode*/Backprop:* "unsupervised" codes generated by Lococode with FMS, fed into a conventional, overfitting BP classifier.

	Technique	nr. hidden	error rates	
		units	training	test
(1.1)	Single-layer perceptron	–	–	0.67
(1.2.1)	Multi-layer perceptron	88	–	0.49
(1.2.2)	Multi-layer perceptron	22	–	0.55
(1.2.3)	Multi-layer perceptron	11	–	0.56
(1.3.1)	Modified Kanerva Model	528	–	0.50
(1.3.2)	Modified Kanerva Model	88	–	0.57
(1.4.1)	Radial Basis Function	528	–	0.47
(1.4.2)	Radial Basis Function	88	–	0.52
(1.5.1)	Gaussian node network	528	–	0.45
(1.5.2)	Gaussian node network	88	–	0.47
(1.5.3)	Gaussian node network	22	–	0.46
(1.5.4)	Gaussian node network	11	–	0.53
(1.6.1)	Square node network	88	–	0.45
(1.6.2)	Square node network	22	–	0.49
(1.6.3)	Square node network	11	–	0.50
(2)	Nearest neighbor	–	–	0.44
(3)	LDA	–	0.32	0.56
(4)	Softmax	–	0.48	0.67
(5)	QDA	–	0.01	0.53
(6.1)	CART	–	0.05	0.56
(6.2)	CART (linear comb. splits)	–	0.05	0.54
(7)	FDA / BRUTO	–	0.06	0.44
(8)	Softmax / BRUTO	–	0.11	0.50
(9.1)	FDA / MARS (degree 1)	–	0.09	0.45
(9.2)	FDA / MARS (degree 2)	–	0.02	0.42
(10.1)	Softmax / MARS (degree 1)	–	0.14	0.48
(10.2)	Softmax / MARS (degree 2)	–	0.10	0.50
(11)	Lococode / Backprop	30/11	0.05	0.42

Table 3: *Vowel recognition task: generalization performance of different methods. Surprisingly, FMS-generated lococodes fed into a conventional, overfitting backprop classifier led to excellent results. See text for details.*

Results. See Table 3. FMS generates 3 different lococodes. Each is fed into 10 BP classifiers with different weight initializations: the table entry for "Lococode/Backprop" represents the mean of 30 trials. The results for neural nets and nearest neighbor are taken from Robinson (1989). The other results (except for Lococode's) are taken from Hastie et al. (1993). Our method led to excellent generalization results. The error rates after BP learning vary between 39 and 45 %.

Backprop fed with LOCOCODE code sometimes goes down to 38 % error rate, but due to overfitting, the error rate increases again (of course, test set performance may not influence the training procedure). Given that BP by itself is a very naive approach it seems quite surprising that excellent generalization performance can be obtained just by feeding BP with *nongoal-specific* lococodes.

Typical feature detectors. The number of pruned HUs (with constant activation) varies between 5 and 10. 2 to 5 HUs become binary, and 4 to 7 trinary. With all codes we observed: apparently, certain HUs become feature detectors for speaker identification. Another HU's activation is near 1.0 for the words "heed" and "hid" ("i" sounds). Another HU's activation has high values for the words "hod", "hoard", "hood" and "who'd" ("o"-words) and low but nonzero values for "hard" and "heard". LOCOCODE supports feature detection.

Why no sparse code? The real-valued input components cannot be described precisely by the activations of the few feature detectors generated by LOCOCODE. Additional real-valued HUs are necessary for representing the missing information.

Better results with additional information. Hastie et al. also obtained additional, even slightly better results with an FDA/MARS variant: down to 39 % average error rate. It should be mentioned, however, that their data was subject to goal-directed preprocessing with splines, such that there were many clearly defined classes. Furthermore, to determine the input dimension, Hastie et al. used a special kind of generalized cross-validation error, where one constant was obtained by unspecified "simulation studies". Hastie and Tibshirani (1996) also obtained an average error rate of 38 % with discriminant adaptive nearest neighbor classification. About the same error rate was obtained by Flake (1998) with RBF networks and hybrid architectures. Also, recent experiments (mostly conducted during the time this paper has been under review) showed that even better results can be obtained by using additional context information to improve classification performance, e.g., Turney (1993), Herrmann (1997), and Tenenbaum and Freeman (1997). For an overview see Schraudolph (1998). It will be interesting to combine these methods with LOCOCODE.

Conclusion. Although we made no attempt at preventing classifier overfitting, we achieved excellent results. From this we conclude that the lococodes fed into the classifier already conveyed the "essential", almost noise-free information necessary for excellent classification. We are led to believe that LOCOCODE is a promising method for data preprocessing.

5 CONCLUSION

LOCOCODE, our novel approach to unsupervised learning and sensory coding, does not define code optimality solely by properties of the code itself but takes into account the information-theoretic complexity of the mappings used for coding and decoding. The resulting lococodes typically compromise between conflicting goals. They tend to be sparse and exhibit *low but not minimal* redundancy — if the costs of generating minimal redundancy are too high. Lococodes tend towards binary, informative feature detectors, but occasionally there are trinary or continuous-valued code components (where complexity considerations suggest such alternatives).

A general principle? According to our analysis LOCOCODE essentially attempts at describing single inputs with as few and as simple features as possible. Depending on the statistical properties of the input, this can result in either local, factorial, or sparse codes, although biologically plausible sparseness is the most common case. Unlike the objective functions of previous methods (e.g., Olshausen and Field 1996), however, LOCOCODE's does *not* contain an explicit term enforcing, say, sparse codes — sparseness or factoriality are not viewed as a good things *a priori.* This seems to suggest that LOCOCODE's objective may embody a general principle of unsupervised learning going beyond previous, more specialized ones.

Regularizers and unsupervised learning. Another way of looking at our results is this: there is at least one representative (FMS) of a broad class of algorithms (regularizer algorithms that reduce net complexity) which can do optimal feature extraction as a by-product. This reveils an interesting, previously ignored connection between two important fields (regularizer research and ICA-related research), and may represent a first step towards unification of regularization and unsupervised learning.

Advantages. Lococode is appropriate if single inputs (with many input components) can be described by few features computable by simple functions. Hence, assuming that visual data can be reduced to few simple causes, Lococode is appropriate for visual coding. Unlike simple ICA, Lococode (a) is not inherently limited to the linear case, and (b) does not need *a priori* information about the number of independent data sources. Even when the number of sources is known, however, Lococode can outperform other coding methods. This has been demonstrated by our Lococode implementation based on FMS-trained autoassociators (AAs), which easily solves coding tasks that have been described as hard by other authors, and whose input causes are not perfectly separable by standard AAs, PCA, and ICA. Furthermore, when applied to realistic visual data, Lococode produces familiar on-center-off-surround receptive fields and biologically plausible sparse codes (standard AAs do not). Codes obtained by ICA, PCA and Lococode convey about the same information, as indicated by the reconstruction error. But Lococode's coding efficiency is higher: it needs fewer bits per input pixel. Our experiments also demonstrate the utility of Lococode-based data preprocessing for subsequent classification.

Limitations. FMS' order of computational complexity depends on the number of output units. For typical classification tasks (requiring few output units) it equals standard backprop's. In the AA case, however, the output's dimensionality grows with the input's. That's why large scale FMS-trained AAs seem to require parallel implementation. Furthermore, although Lococode works well for visual inputs, it may be less useful for discovering input causes that can only be represented by high-complexity input transformations, or for discovering many features (causes) collectively determining single input components (as, e.g., in acoustic signal separation, where ICA does not suffer from the fact that each source influences each input component and none is computable by a low-complexity function).

Future work. Encouraged by the familiar lococodes obtained in our experiments with visual data we intend to move on to higher-dimensional inputs and larger receptive fields. This may lead to even more pronounced feature detectors like those observed by Schmidhuber et al. (1996). It will also be interesting to test whether successive Lococode stages, each feeding its code into the next, will lead to complex feature detectors such as those discovered in deeper regions of the mammalian visual cortex. Finally, encouraged by our successful application to vowel classification, we intend to look at more complex pattern recognition tasks.

We also intend to look at alternative Lococode implementations besides FMS-based AAs. Finally we would like to improve our understanding of the relationship between low-complexity codes, low-complexity art (see Schmidhuber, 1997b) and informal notions such as “beauty” and “good art”.

6 ACKNOWLEDGMENTS

We would like to thank Peter Dayan, Manfred Opper, Nic Schraudolph, Rich Zemel, and several anonymous reviewers for helpful discussions and for comments on a draft of this paper. This work was supported by *DFG grant SCHM 942/3-1* and *DFG grant BR 609/10-2* from “Deutsche Forschungsgemeinschaft”. Schmidhuber would also like to acknowledge support from *SNF grant 21-43'417.95 “predictability minimization”*.

References

Amari, S., Cichocki, A., and Yang, H. (1996). A new learning algorithm for blind signal separation. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*, pages 757–763. The MIT Press, Cambridge, MA.

Baldi, P. and Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2:53–58.

Barlow, H. B. (1983). *Understanding natural vision.* Springer-Verlag, Berlin.

Barlow, H. B., Kaushal, T. P., and Mitchison, G. J. (1989). Finding minimum entropy codes. *Neural Computation*, 1(3):412–423.

Barrow, H. G. (1987). Learning receptive fields. In *Proceedings of the IEEE 1st Annual Conference on Neural Networks*, volume IV, pages 115–121. IEEE.

Baumgartner, M. (1996). Bilddatenvorverarbeitung mit neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München.

Becker, S. (1991). Unsupervised learning procedures for neural networks. *International Journal of Neural Systems*, 2(1 & 2):17–33.

Bell, A. J. and Sejnowski, T. J. (1995). An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159.

Cardoso, J.-F. and Souloumiac, A. (1993). Blind beamforming for non Gaussian signals. *IEE Proceedings-F*, 140(6):362–370.

Comon, P. (1994). Independent component analysis – a new concept? *Signal Processing*, 36(3):287–314.

Dayan, P. and Zemel, R. (1995). Competition and multiple cause models. *Neural Computation*, 7:565–579.

Deco, G. and Brauer, W. (1995). Nonlinear higher-order statistical decorrelation by volume-conserving neural architectures. *Neural Networks*, 8(4):525–535.

Deco, G. and Parra, L. (1994). Nonlinear features extraction by unsupervised redundancy reduction with a stochastic neural network. Technical report, Siemens AG, ZFE ST SN 41.

DeMers, D. and Cottrell, G. (1993). Non-linear dimensionality reduction. In S. J. Hanson, J. D. C. and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 580–587. Morgan Kaufmann, San Mateo, CA.

Field, D. J. (1987). Relations between the statistics of natural images and the response properties of cortical cells. *Journal of the Optical Society of America*, 4:2379–2394.

Field, D. J. (1994). What is the goal of sensory coding? *Neural Computation*, 6:559–601.

Flake, G. W. (1998). Square unit augmented, radially extended, multilayer perceptrons. In Orr, G. B. and Müller, K.-R., editors, *Tricks of the Trade.* Springer Verlag, Berlin. To appear in *Lecture Notes in Computer Science.*

Földiák, P. (1990). Forming sparse representations by local anti-hebbian learning. *Biological Cybernetics*, 64:165–170.

Földiák, P. and Young, M. P. (1995). Sparse coding in the primate cortex. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*, pages 895–898. The MIT Press, Cambridge, Massachusetts.

Ghahramani, Z. (1995). Factorial learning and the EM algorithm. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems 7*, pages 617–624. MIT Press, Cambridge MA.

Hassibi, B. and Stork, D. G. (1993). Second order derivatives for network pruning: Optimal brain surgeon. In S. J. Hanson, J. D. C. and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 164–171. San Mateo, CA: Morgan Kaufmann.

Hastie, T. J. and Tibshirani, R. J. (1996). Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6):607–616.

Hastie, T. J., Tibshirani, R. J., and Buja, A. (1993). Flexible discriminant analysis by optimal scoring. Technical report, AT&T Bell Laboratories.

Herrmann, M. (1997). On the merits of topography in neural maps. In Kohonen, T., editor, *Proceedings of the Workshop on Self-Organizing Maps*, pages 112–117. Helsinki University of Technology.

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1158–1161.

Hinton, G. E. and Ghahramani, Z. (1997). Generative models for discovering sparse distributed representations. *Philosophical Transactions of the Royal Society* **B**, 352:1177–1190.

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length and Helmholtz free energy. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 3–10. Morgan Kaufmann, San Mateo, CA.

Hochreiter, S. and Schmidhuber, J. (1995). Simplifying nets by discovering flat minima. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 529–536. MIT Press, Cambridge MA.

Hochreiter, S. and Schmidhuber, J. (1997a). Flat minima. *Neural Computation*, 9(1):1–42.

Hochreiter, S. and Schmidhuber, J. (1997b). Low-complexity coding and decoding. In Wong, K. M., King, I., and Yeung, D., editors, *Theoretical Aspects of Neural Computation (TANC 97), Hong Kong*, pages 297–306. Springer.

Hochreiter, S. and Schmidhuber, J. (1997c). Unsupervised coding with Lococode. In Gerstner, W., Germond, A., Hasler, M., and Nicoud, J.-D., editors, *Proceedings of the International Conference on Artificial Neural Networks, Lausanne, Switzerland*, pages 655–660. Springer.

Hochreiter, S. and Schmidhuber, J. (1998). Lococode versus PCA and ICA. In *Proceedings of the International Conference on Artificial Neural Networks*. To appear.

Jutten, C. and Herault, J. (1991). Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24(1):1–10.

Kohonen, T. (1988). *Self-Organization and Associative Memory*. Springer, second ed.

Kramer, M. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37:233–243.

Li, Z. (1995). A theory of the visual motion coding in the primary visual cortex. *Neural Computation*, 8(4):705–730.

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

Molgedey, L. and Schuster, H. G. (1994). Separation of independent signals using time-delayed correlations. *Phys. Reviews Letters*, 72(23):3634–3637.

Mozer, M. C. (1991). Discovering discrete distributed representations with iterative competitive learning. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 627–634. San Mateo, CA: Morgan Kaufmann.

Nadal, J.-P. and Parga, N. (1997). Redundancy reduction and independent component analysis: Conditions on cumulants and adaptive approaches. *Neural Computation*, 9(7):1421–1456.

Oja, E. (1989). Neural networks, principal components, and subspaces. *International Journal of Neural Systems*, 1(1):61–68.

Oja, E. (1991). Data compression, feature extraction, and autoassociation in feedforward neural networks. In Kohonen, T., Mäkisara, K., Simula, O., and Kangas, J., editors, *Artificial Neural Networks*, volume 1, pages 737–745. Elsevier Science publishers B.V., North-Holland.

Olshausen, B. A. and Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609.

Pajunen, P. (1998). Blind source separation using algorithmic information theory. In Fyfe, C., editor, *Proceedings of Independence and Artificial Neural Networks (I & ANN)*, pages 26–31. ICSC Academic Press.

Palm, G. (1992). On the information storage capacity of local learning rules. *Neural Computation*, 4(2):703–711.

Redlich, A. N. (1993). Redundancy reduction as a strategy for unsupervised learning. *Neural Computation*, 5:289–304.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465–471.

Robinson, A. J. (1989). *Dynamic Error Propagation Networks*. PhD thesis, Trinity Hall and Cambridge University Engineering Department.

Rumelhart, D. E. and Zipser, D. (1986). Feature discovery by competitive learning. In *Parallel Distributed Processing*, pages 151–193. MIT Press.

Saund, E. (1994). Unsupervised learning of mixtures of multiple causes in binary data. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 27–34. Morgan Kaufmann, San Mateo, CA.

Saund, E. (1995). A multiple cause mixture model for unsupervised learning. *Neural Computation*, 7(1):51–71.

Schmidhuber, J. (1992). Learning factorial codes by predictability minimization. *Neural Computation*, 4(6):863–879.

Schmidhuber, J. (1997a). Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873.

Schmidhuber, J. (1997b). Low-complexity art. *Leonardo, Journal of the International Society for the Arts, Sciences, and Technology*, 30(2):97–103.

Schmidhuber, J., Eldracher, M., and Foltin, B. (1996). Semilinear predictability minimization produces well-known feature detectors. *Neural Computation*, 8(4):773–786.

Schmidhuber, J. and Prelinger, D. (1993). Discovering predictable classifications. *Neural Computation*, 5(4):625–635.

Schraudolph, N. N. (1998). On centering neural network weight updates. In Orr, G. B. and Müller, K.-R., editors, *Tricks of the Trade*. Springer Verlag, Berlin. To appear in *Lecture Notes in Computer Science*.

Schraudolph, N. N. and Sejnowski, T. J. (1993). Unsupervised discrimination of clustered data via optimization of binary information gain. In S. J. Hanson, J. D. C. and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 499–506. San Mateo, CA: Morgan Kaufmann.

Solomonoff, R. (1964). A formal theory of inductive inference. Part I. *Information and Control*, 7:1–22.

Tenenbaum, J. B. and Freeman, W. T. (1997). Separating style and content. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9*, pages 662–668. The MIT Press, Cambridge, MA.

Turney, P. D. (1993). Exploiting context when learning to classify. In *Proceedings of the European Conference on Machine Learning*, pages 402–407. ftp://ai.iit.nrc.ca/pub/ksl-papers/NRC-35058.ps.Z.

Wallace, C. S. and Boulton, D. M. (1968). An information theoretic measure for classification. *Computer journal*, 11(2):185–194.

Watanabe, S. (1985). *Pattern Recognition: Human and Mechanical.* Willey, New York.

Zemel, R. S. (1993). *A minimum description length framework for unsupervised learning.* PhD thesis, University of Toronto.

Zemel, R. S. and Hinton, G. E. (1994). Developing population codes by minimizing description length. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 11–18. San Mateo, CA: Morgan Kaufmann.