From Network: Comput. Neural Syst. **10** (1999) 133-169.

Processing images by semi-linear predictability minimization

Nicol N Schraudolph, Martin Eldracher and Jürgen Schmidhuber
IDSIA, Corso Elvezia 36, CH-6900 Lugano, Switzerland
E-mail: `nic@idsia.ch`, `eldracher.martin@bcg.com` and `juergen@idsia.ch`

Received 30 June 1998, in final form 6 April 1999

Abstract. In the predictability minimization approach (Schmidhuber 1992), input patterns are fed into a system consisting of adaptive, initially unstructured feature detectors. There are also adaptive predictors constantly trying to predict current feature detector outputs from other feature detector outputs. Simultaneously, however, the feature detectors try to become as unpredictable as possible, resulting in a co-evolution of predictors and feature detectors. This paper describes the implementation of a visual processing system trained by semi-linear predictability minimization, and presents many experiments that examine its response to artificial and real-world images. In particular, we observe that under a wide variety of conditions, predictability minimization results in the development of well-known visual feature detectors.

1. Introduction

1.1. Purpose of the Paper

Predictability minimization (PM) has so far mainly been tested on artificial code strings (Lindstädt 1993, Schmidhuber and Prelinger 1993, Schmidhuber 1993, Schmidhuber 1999). First experiments with real world image data have been reported by Schmidhuber, Eldracher and Foltin (1996). The present paper extends that work by examining the effects of different preprocessing of the images, using different input images, having only local receptive fields for predictor units, using different sizes of receptive fields, or receptive fields ordered in columns. It also adds a more detailed study of PM's sensitivity to free parameters.

1.2. Outline of the Paper

Section 2 describes our implementation of semi-linear PM for image processing purposes. After briefly reviewing the PM architecture in general (2.1), we cover the activation of inputs (2.2), code units (2.3), and predictors (2.4) in greater detail, followed by a discussion of the measurements we used to determine system performance (2.5), and a summary of the system's free parameters (2.6).

In Section 3 we present the results of applying these techniques to artificial and real-world images, investigating the effect of input and parameter settings on PM's performance. Having shown that no artifacts are learned from white noise (3.1.1), we first test PM on regular and irregular artificial input patterns (3.1.2, 3.1.3). Next we explore PM's response to the characteristics of photographic input (3.2). While PM is sensitive to how images are pre-processed (3.2.2), we find it quite robust to noise in the input (3.2.3). The section concludes with a look at how sensitive PM is to

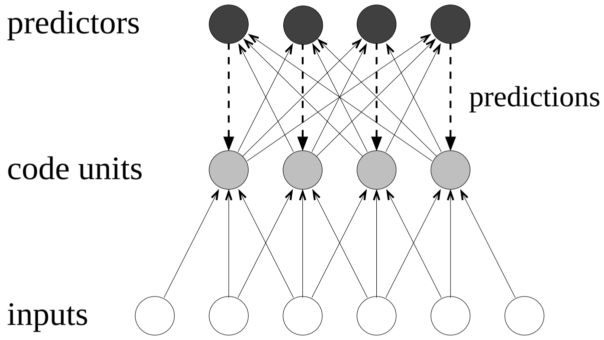

Figure 1. The network architecture for semi-linear PM; see text for details.

the setting of various free parameters, namely learning rate (3.3.1), code unit radius (3.3.2), input pixel offset (3.3.3), exactness parameter (3.3.4), and weight initialization (3.3.5).

Section 4 introduces two major variations of our image processing architecture: local receptive fields for predictor units (4.1), and columnar organization of code units in multiple layers (4.2). Combining these two modifications, we then apply PM to a network with several partially overlapping layers of code units (4.3).

Section 5 presents conclusions and comments prompted by these experimental results. Specifically, we observe a tradeoff between information throughput and binarity (5.1), and discuss the difficulty of measuring the performance of PM (5.2) as well as its dependence on the number of test patterns (5.3) and the choice of learning rates (5.4).

In Section 6 we compare PM to a number of related approaches found in the literature, before concluding the paper with a brief summary of our findings (Section 7).

2. Implementation

2.1. Network Architecture

Figure 1 illustrates our neural network architecture for semi-linear PM: redundant input (white) is coded across code units (grey), which in turn provide input to the predictors (black). Each predictor attempts to predict its code unit, from which it does not receive input. Each code unit, however, tries to escape its prediction (dashed) by representing an environmental property that is independent of those represented by other code units. In this way predictors and code units are caught in an arms race that forces them to co-evolve a recoding of the input that maximizes information throughput and minimizes redundancy.

Predictors and code units are trained by simultaneous gradient descent (resp. ascent) in the squared prediction error, with learning rate `lrp` for predictors, and `lrc` for code units. For reasons of computational simplicity and biological plausibility, no error signals are propagated from the predictors back into the network of code units. Inputs, predictors and code units are all topologically organized on two-dimensional grids, as described in the following.

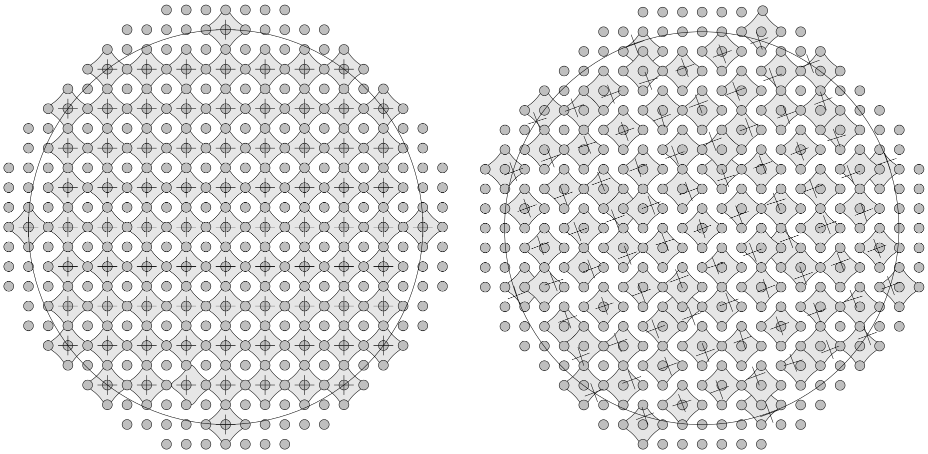

Figure 2. The input area (large circle) positioned on the pixel grid (small disks) of an image at orientations of zero (left) and 20 degrees (right). Crosses indicate the location of input units. The activation of each input unit is the average value of its nearest pixel and the four adjacent ones (shaded areas).

2.2. Input Unit Activation

We apply PM to static, black-and-white images with 2–256 grey levels, ranging in size from 480×480 to 566×702 pixels. These images include white noise, regular and irregular artificial patterns (Figures 5 and 7), and photographs of driving cars (Figure 12; reference image: Figure 15). Each pixel is represented by a grey value between 0.0 (black) and 1.0 (white).

Input is sampled from a circular *input area* whose position and orientation on the image is uniformly random. Within the input area, input units are typically spaced `offset_input_units` $= 2$ pixels apart on a square grid. Unless stated otherwise, the radius of the input area is `radius_input_area` $= 16$ input units, or 32 pixels. Each input unit is deemed to lie on the closest pixel of the image (in terms of Euclidean distance); its activation is the average of this pixel and its four immediate neighbors, as illustrated by the shaded areas in Figure 2.

2.3. Code Unit Activation

Code units are distributed over the input area on a hexagonal grid; they have partially overlapping receptive fields with a radius of `radius_code_units` pixels. If the center of a code unit's receptive field falls outside the input area, the unit is pruned away. The spacing of the hexagonal grid is equal to the receptive field radius, so that the degree of overlap between receptive fields remains constant. The number of code units in the hexagonal grid is therefore determined indirectly by `radius_code_units`, as illustrated in Figure 3. An additional parameter, `number_code_layers`, permits the creation of multiple code units for each receptive field position in the grid.

Each code unit receives input from all input units lying inside its receptive field, plus a constant bias input of 1.0. Its activation is given by the logistic sigmoid function $1/(1+e^{-x})$, where x is the weighted sum of its inputs.

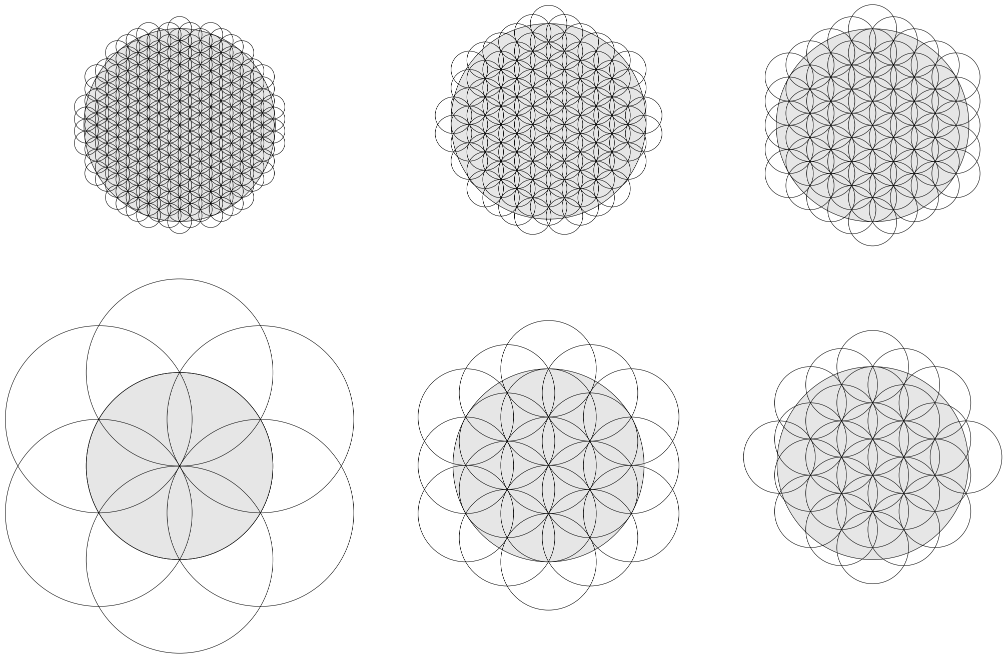

Figure 3. Layout of the receptive fields of code units on the hexagonal grid for (clockwise from upper left) `radius_code_units` = 4, 6, 8 (default), 12, 16, and 32.

2.4. Predictor Unit Activation

Since we are deliberately restricting ourselves to the semi-linear case, all predictor networks consist of just a sigmoid unit with its single layer of weights. Each predictor has a receptive field of radius `radius_pred_units` pixels centered on the position of the code unit it is trying to predict. With the exception of that unit, each predictor receives input from all code units that lie inside its receptive field, plus a constant bias input of 1.0. As with code units, a predictor's activation is given by the logistic sigmoid function of the weighted sum of its inputs. Since local receptive fields for predictors are used only in some of our experiments, we set `radius_pred_units` $= \infty$ by default, so that each predictor receives input from all code units but one.

2.5. Measuring Performance

Every `train_test_interval` training cycles, learning by PM is switched off, and we test the code units' response to `num_pat_test` randomly generated input patterns. For testing purposes, the output of a code unit is taken to be "0" if below $0+$ `exactness`, "1" if above $1-$ `exactness`, and "i" (intermediate) otherwise. Given this classification of code unit activity, we define two performance measures that relate to the network's information compression and transmission properties, respectively: *binarity* and *throughput*.

The binarity of the code is simply the proportion of code unit activity at non-intermediate values. Since a binary output can be represented by a single bit whereas intermediate ("i") values require higher precision, binarity is a measure of information compression.

Table 1. Summary of the free parameters in our experiments.

Parameter	Domain	Default	Range	Section(s)
grey levels	$\mathbb{N}$	16	$2^n, n = 1$–8	3.2.2
noise percentage	$[0, 1]$	0	0.01–0.4	3.2.3
`exactness`	$(0, 0.5]$	0.05	0.15–0.5	3.3.4
`lrp`	$\mathbb{R}^+$	0.7	0.3, 0.5	3.3.1
`lrc`	$\mathbb{R}^+$	0.8 `lrp`	0.24, 0.4	3.3.1
`radius_code_units`	$\mathbb{N}$	8	4–32	3.3.2, 4
`radius_pred_units`	$\mathbb{N}$	∞	4–32	4.1, 4.3
`radius_input_area`	$\mathbb{N}$	16	8–32	3.3.3, 4.1, 4.3
`offset_input_units`	$\mathbb{N}$	2	1	3.3.3, 4.1, 4.3
`number_code_layers`	$\mathbb{N}$	1	7–19	4.2.3
`num_pat_train`	$\mathbb{N}$	10^6	—	—
`train_test_interval`	$\mathbb{N}$	5000	—	—
`num_pat_train_test`	$\mathbb{N}$	1000	—	—
`num_pat_test`	$\mathbb{N}$	5000	—	—

We define the information *throughput* to be $N/$`num_pat_test`, where N is the number of pairwise different output patterns (under the trinary classification) generated in response to the test inputs. Given the huge space of possible inputs, we can safely assume that all `num_pat_test` test patterns are different. Since the space of trinary output patterns is also far greater than `num_pat_test`, we could ideally obtain a different response for each test pattern, *i.e.*, a throughput of 1.0.

A code with high throughput and binarity may enable a subsequent classifier to achieve good performance. It is not clear, however, whether a binary output is generally preferable to a continuous one for subsequent visual processing. Although this question is beyond the scope of this paper, some comments on it can be found in Section 5.2.

2.6. Free Parameters

Table 1 lists the free parameters in our implementation of PM, giving their name, natural domain, default value, the explored range of non-default values, and the section(s) in which the effect of this parameter is studied.

3. Experiments with Input and Parameter Settings

3.1. Artificial Input Patterns

We now explore PM's response to the nature of its input, and the setting of its free parameters. We first test PM on various regular and irregular artificial input patterns.

3.1.1. White Noise Our first experiment used a 256 grey level image with pixels set to uniformly random intensity values. Figure 4 shows the receptive fields of the code units that PM evolved after 10^6 representations of this white noise pattern at the default parameter settings (see Table 1). Since the calculation of input unit activations (Section 2.2) introduces correlations between neighboring units, there is

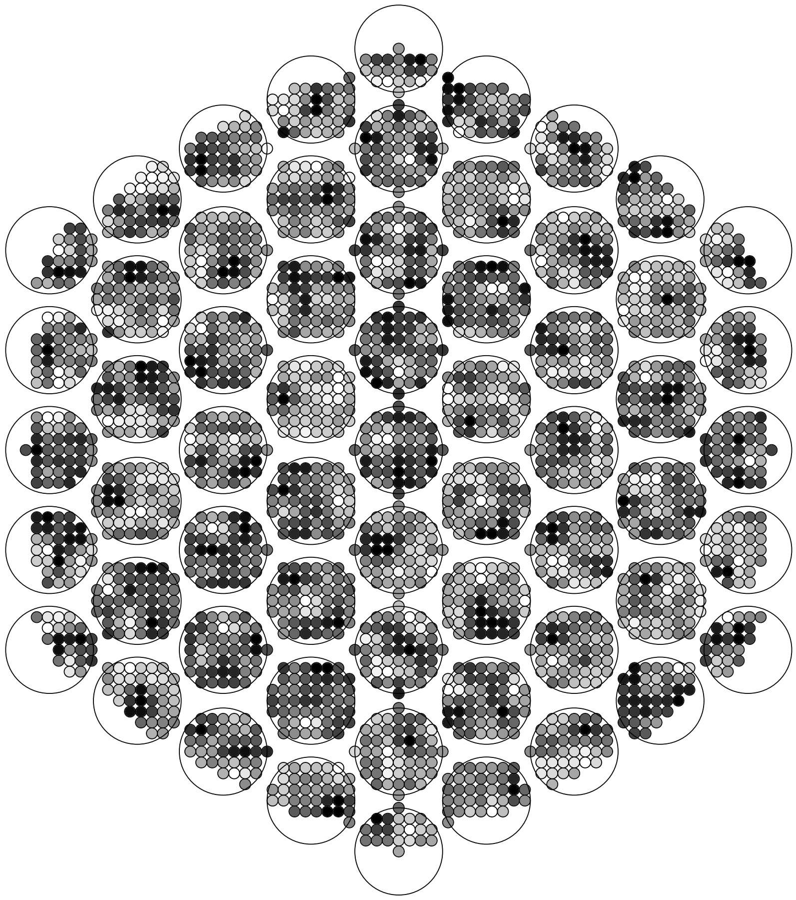

Figure 4. Receptive fields of code units after 10^6 presentations of a white noise pattern. The display of incoming weights (small discs) is scaled for each code unit (large circle) to range from black (most negative) to white (most positive). There is some smoothness but no spatial organization.

some smoothness in the receptive fields. Other than that, however, they appear spatially rather disorganized, as is to be expected.

Nonetheless PM still improved our performance measures: at the end of training, we measured a binarity of 90.8%, and the throughput had risen to 1.0. Thus despite of the absence of structure, the system achieved near-optimal performance according to its learning rule. Similar results were obtained for a white noise image with 16 grey levels.

3.1.2. Tile Patterns In order to observe what kind of feature detectors are developed for various kinds of input, we applied PM (with the default parameter settings) to a set of eight tile patterns.

Figure 5 shows the eight tile patterns we used to train PM; they fall into three groups with respect to our results. For "tile1" PM had considerable difficulty in achieving a high throughput, although it did succeed eventually. Judging from the large swings in throughput evident in Figure 6, it appears that the default learning

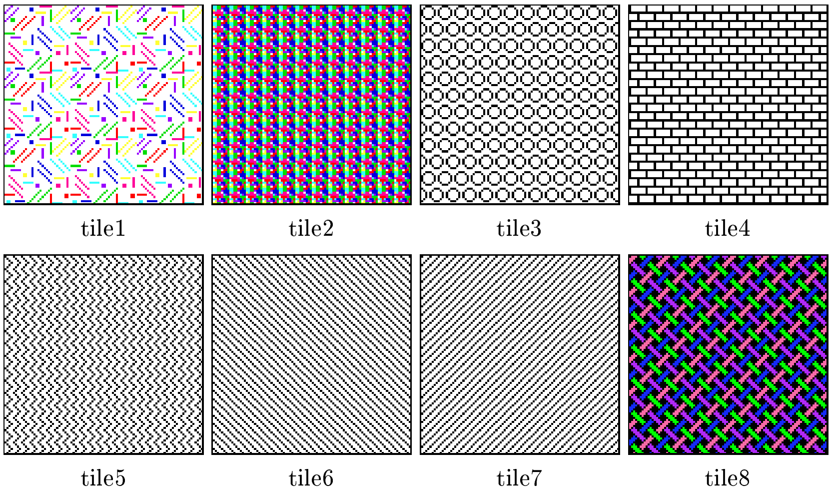

Figure 5. Tile patterns used to test PM.

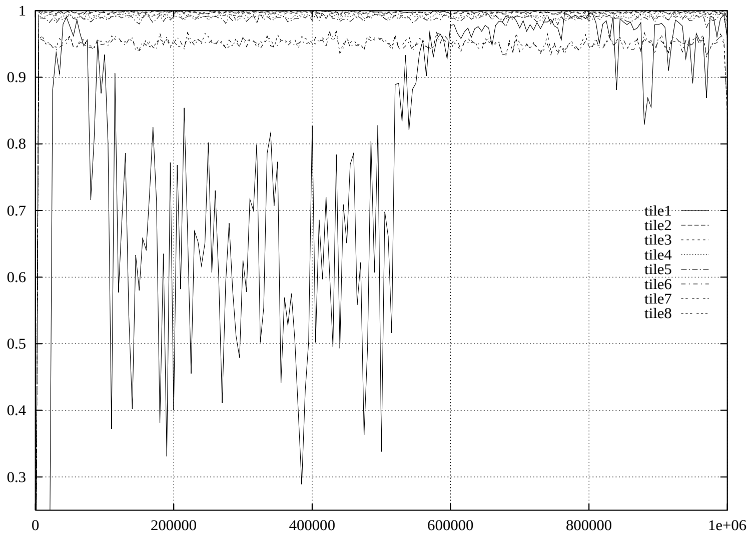

Figure 6. Throughput during training on the patterns shown in Figure 5.

rate may be too high for this pattern. The binarity of the code units (see Table 2) also remained far lower for "tile1" than any of the other tile patterns.

The second group consists of the patterns "tile6" and "tile7", for which PM only reached a throughput of about 90%. This may be due to the fact that these patterns contain little structure, and only at a very fine scale. By contrast, in the third group (all remaining patterns) throughput quickly rose close to 100%. With the exception of "tile2", binarity reached around 95% for all patterns in the second and third group.

3.1.3. Artificial Images For the next set of experiments we constructed the six artificial images shown in Figure 7. We found that the default learning rates were

Table 2. Binarity of the code after training on the patterns shown in Figure 5.

pattern:	tile1	tile2	tile3	tile4	tile5	tile6	tile7	tile8
binarity:	80.4%	91.9%	95.0%	95.3%	94.6%	94.5%	94.3%	95.3%

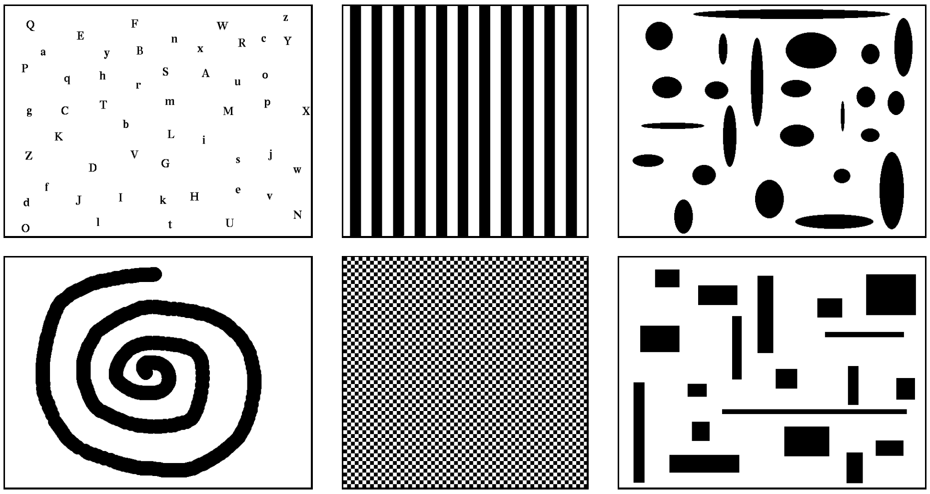

Figure 7. Artificial images used to test PM. From left to right: "letters", "bars", and "ellipses" in the upper row, "spiral", "checkers", and "squares" in the lower.

Table 3. Binarity of the code after training on the images shown in Figure 7.

image:	letters	bars	ellipses	spiral	checkers	squares
binarity:	95.5%	99.9%	97.6%	98.6%	92.4%	99.1%

too high for these images, and lowered them to `lrp` = 0.05 and `lrc` = 0.04. Other than that, we used the default parameter settings given in Table 1. The slow decline in throughput that can be seen in Figure 8 for some of the images indicates that the learning rates may still be a little too high. Well-structured receptive fields formed nonetheless, and so we refrained from repeating the experiment at even lower learning rates.

Interestingly the image "checkers" resulted in the highest throughput but the lowest binarity (shown in Table 3) — it appears that there is a tradeoff between these two performance criteria. The receptive fields formed by PM for this image (Figure 9) show sparseness but no obvious spatial organization: they typically contain only few weights of large magnitude in apparently arbitrary locations. For the image "letters", on the other hand, topologically well-organized receptive fields were formed (Figure 10), though they are not easily interpreted either. Moreover, they did not result in high throughput.

All other images produce receptive fields with center-surround structure, as shown in Figure 11 for the image "squares". In contrast to *e.g.* Section 3.3 the transition between center and surround is very sharp — a result of the abrupt black/white transitions in these images.

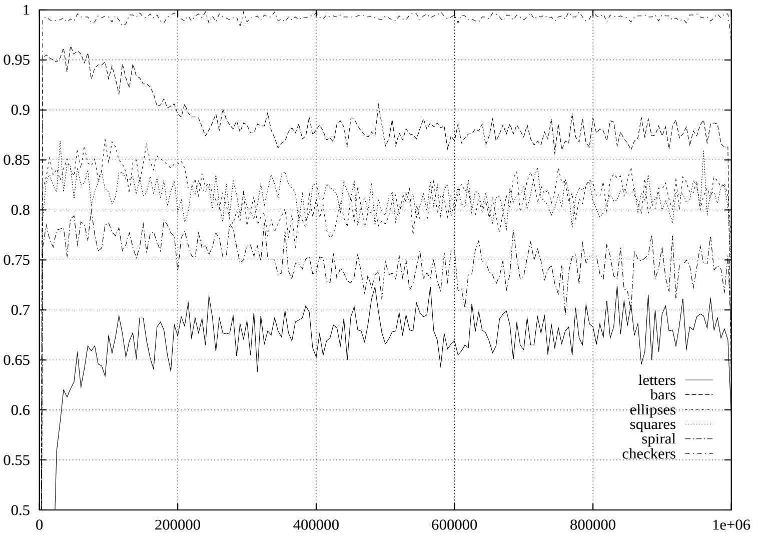

Figure 8. Throughput during training on the images shown in Figure 7.

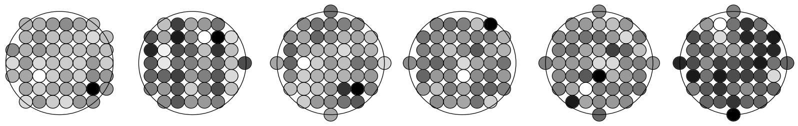

Figure 9. Receptive fields formed for the image "checkers". Throughput is very high (see Figure 8) even though the distribution of weights is sparse, and lacks spatial organization.

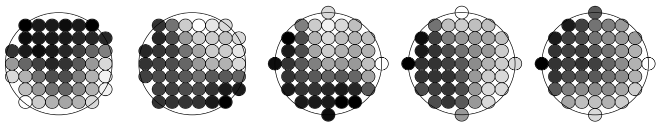

Figure 10. Receptive fields formed for the image "letters". They are spatially well-organized but do not produce a high throughput (see Figure 8).

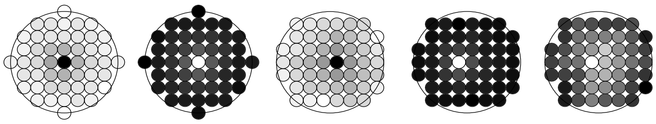

Figure 11. Receptive fields formed for the image "squares". Sharp center-surround structures like these were also formed for "bars", "ellipses", and "spiral".

Table 4. Binarity of the code after training on the photos shown in Figure 12.

photo:	reference	1	2	3	4	5	6
binarity:	61.8%	55.7%	87.3%	76.5%	60.2%	58.8%	59.1%

Most of the time the center of the feature detector coincidences with the center of the receptive field. Due to the partial overlap between receptive fields (*cf.* Figure 3), only input structure at the very center of a code unit's receptive field is not readily visible to any of the neighboring code units, and thus to the unit's own predictor. It is therefore natural that in trying to escape their predictors, code units should pay particular attention to the center of their receptive fields.

3.1.4. Conclusions PM generally produced well-structured receptive fields in response to our structured input patterns. With one notable exception, receptive fields were topologically organized, predominantly into center-surround feature detectors. Throughput and binarity of the code, our two performance measures for PM, were usually quite high. However, it appears that PM can sometimes achieve this through means other than spatial organization of the code units' receptive fields.

3.2. Characteristics of Photographic Input

Henceforth we apply PM to photographic input; specifically, a set of images collected in the context of an autonomous road navigation task. In this section we explore how the characteristics of such photographic input affect PM's performance.

3.2.1. Different Photos In order to assess the dependence of our results on the particular input image, we compared the behaviour of PM with default parameter settings on the six photographs shown in Figure 12 with that on our reference image (Figure 15). All images have been reduced to 16 grey levels.

The throughput on these photos is shown in Figure 13. A high throughput was reached quickly on all images except for our reference photo. It appears that our choice of reference image was not the most fortunate — we suspect that the large area of flat gray it sports in the foreground may make it impossible to achieve a throughput much above 90% in this setting.

The binarity of the trained code units (Table 4) varied widely from image to image, without any obvious correlation to the throughput. The receptive fields always had a well-developed center-surround structure, with no noticeable difference between input images. Those for photo #5 are given in Figure 14 as an example.

3.2.2. Quantization Level In this section we look at the effect of preprocessing a photograph by quantizing it to a given number of grey levels. While images with many grey levels contain more information overall, quantization emphasizes edges in the image that assist in the formation of structured feature detectors. We used our default parameter set, and tested the reference photo at 4, 8, 16, 32, 64, 128, and 256 grey levels (Figure 15).

The evolution of throughput during training on these images is shown in Figure 16. We note the following two major effects:

Figure 12. Six different input photos used to test PM, all reduced to 16 grey levels. See Figure 15 for the reference image.

Table 5. Binarity of the code for the reference photo *vs.* level of quantization.

grey levels:	4	8	16	32	64	128	256
binarity:	20.3%	52.8%	62.8%	54.3%	52.1%	54.8%	38.0%

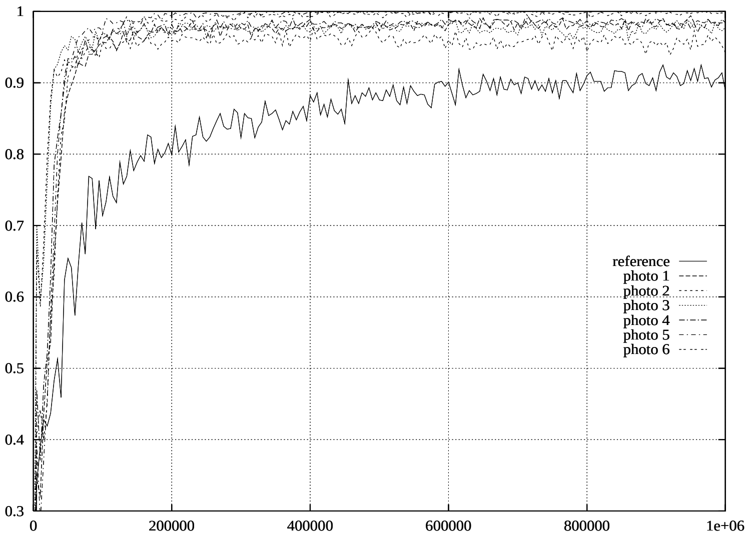

Figure 13. Throughput during training on the photos shown in Figure 12.

- The level of throughput reached asymptotically decreased sharply for heavily quantized images: from 100% for 32 or more grey levels, to 90% for 16, 60% for 8, and only 40% for 4 grey levels.
- For lightly quantized images, the time taken to reach 100% throughput depends on the level of quantization — the 256 grey level image in particular posed considerable difficulty at first.

Clearly the level of quantization has a major influence on PM's ability to rapidly develop a high-throughput code: while some mild quantization emphasizes image features that seem to help PM evolve useful receptive fields quickly, too much of it reduces the amount of information in the image to the point where high throughput is no longer possible. In our case, a quantization to 32 or 64 grey levels appears optimal.

The binarity after a million training patterns (Table 5) presents a similar picture: from as low as 20% for 4 grey levels, it improved drastically with decreasing quantization, only to suffer a gradual decline for images with more than 16 grey levels. We believe this decline to be a transient effect due to a slower development of binarity for images with higher information content. Figure 17 shows that the binarity reached after 5 million training patterns for the 256 grey level image was indeed higher than any of those reported in Table 5.

Regardless of throughput and binarity, center-surround feature detectors were predominant at all quantization levels. Only the border elements, having less input, occasionally did not develop well-organized receptive fields, especially for the heavily quantized images. Other than that, there were no noticeable differences between the receptive fields formed for different levels of quantization.

3.2.3. Level of Noise In this section we examine the robustness of PM to input noise by subjecting it (with default parameter settings) to noisy versions of our reference photo (Figure 15). The noise level is given as a proportion of corrupted pixels. When a pixel is corrupted by noise, we set it to an arbitrary of our 16 gray levels, thus

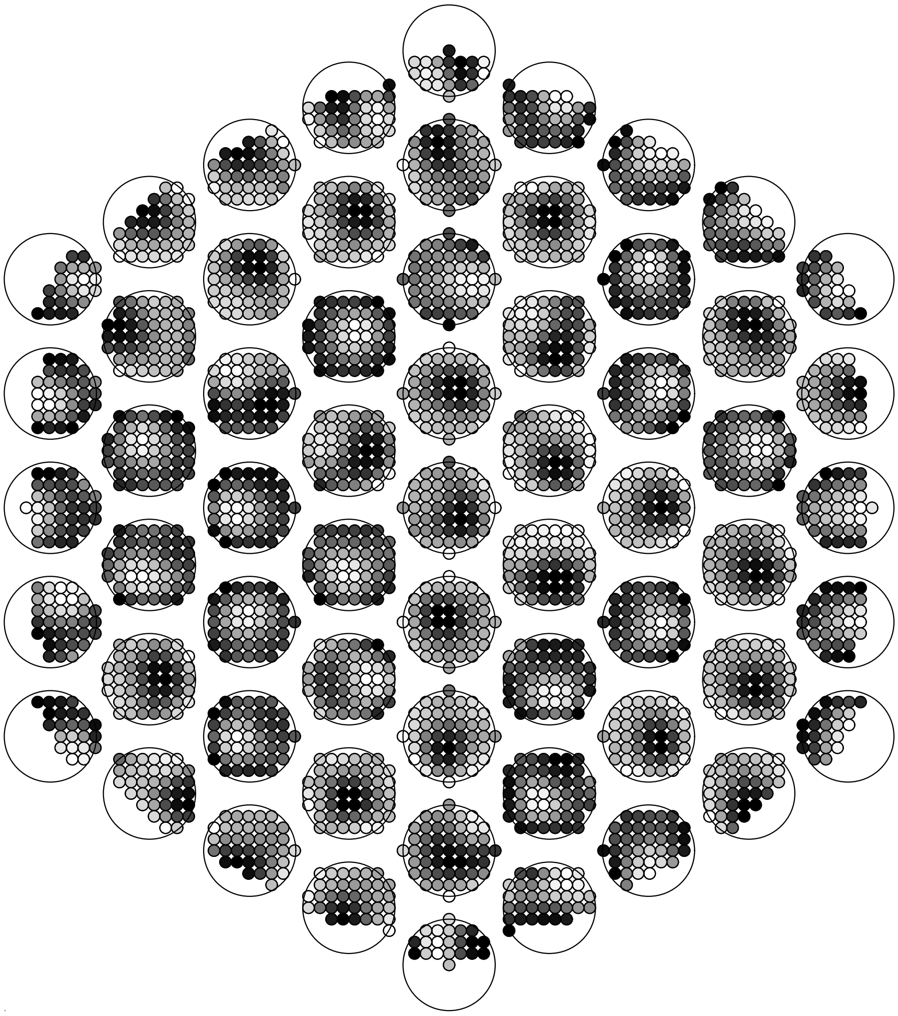

Figure 14. Receptive fields for photo #5 from Figure 12: center-surround structure predominates.

destroying all information it carried. Figure 18 illustrates how the image degrades under this severe form of noise.

Figure 19 shows the evolution of throughput during training — please note the scale of the x-axis. There is a pronounced difference here between the reference photo and its noisy variants: even a small level of noise obviously permits PM to reach maximal throughput far more rapidly. The binarity of the code (Figure 20) and the speed with which it develops both also rise with increasing noise level, albeit in a gentler, more linear fashion. The noise also shortens the initial transient of elevated binarity visible at low noise levels, to the point of dampening it out completely. As usual, binarity is much slower to develop than throughput, even at the highest noise levels.

The receptive fields formed with noise are very similar to those for the noise-free case; Figure 21 shows some examples for 5, 20, and 40 percent of pixels corrupted.

Figure 15. The reference photo quantized to (clockwise from upper left) 4, 16 (default), 64, and 256 grey levels.

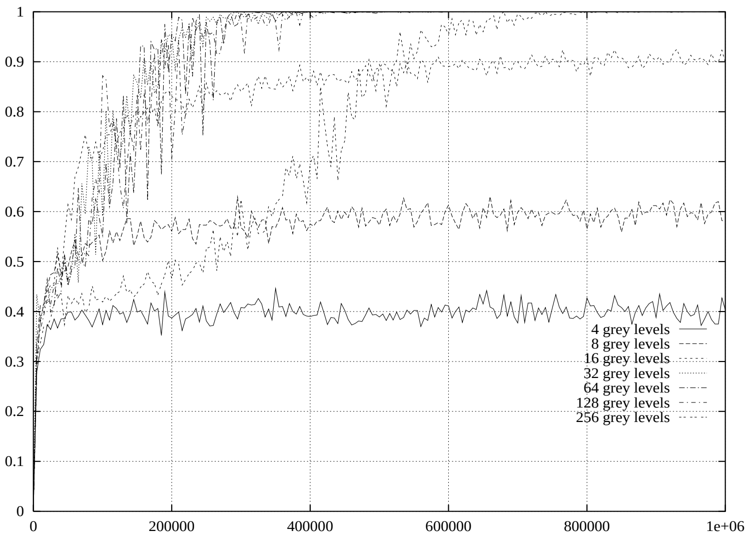

Figure 16. Throughput for the reference photo at different levels of quantization.

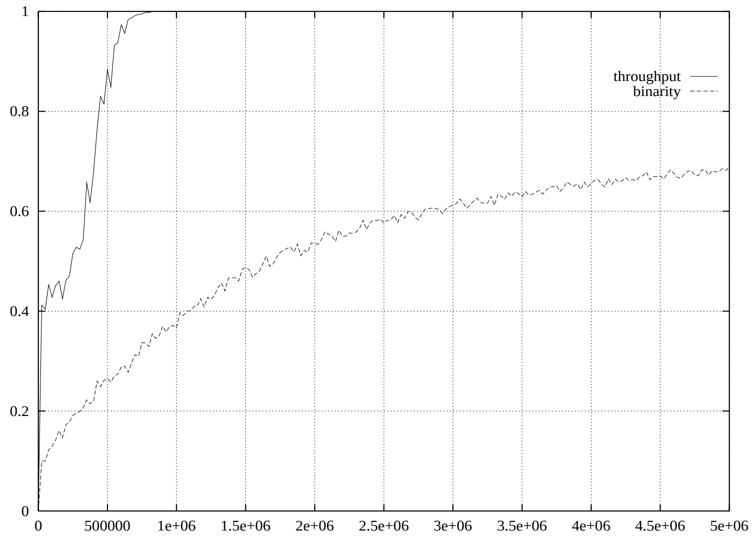

Figure 17. Throughput and binarity for the reference photo at 256 grey levels over an extended run of 5 million training patterns. Binarity keeps increasing gradually.

Figure 18. Degraded versions of the reference photo, with (clockwise from upper left) 1, 5, 20, and 40 % of pixels corrupted.

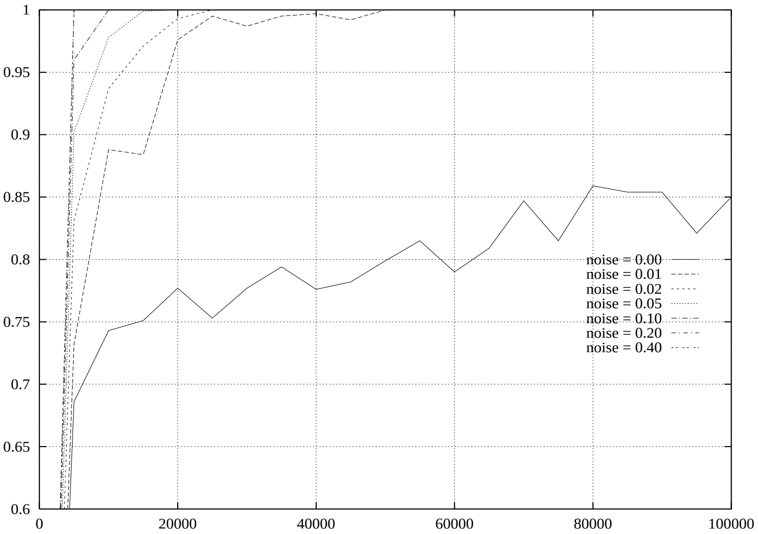

Figure 19. Throughput for the reference photo with different proportions of corrupted pixels.

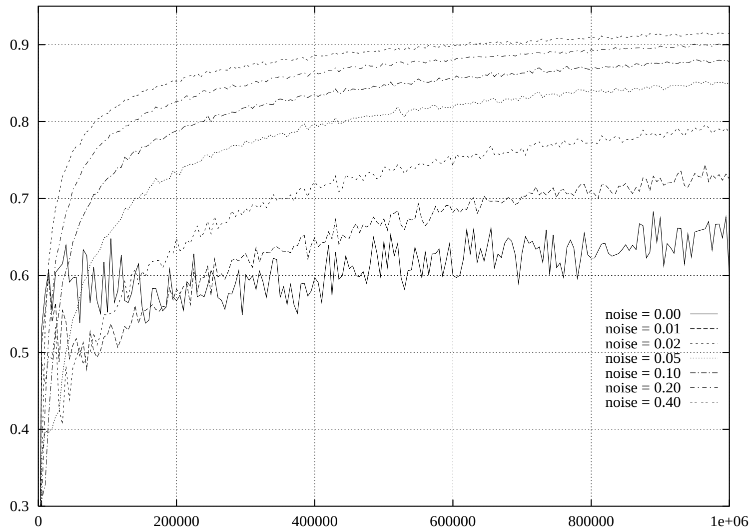

Figure 20. Binarity of the code for the reference photo with different proportions of corrupted pixels.

While the center-surround structures at high noise levels appear slightly less organized, the effect is barely discernible. We conclude that PM is highly robust to severe image degradation.

3.3. Setting of Free Parameters

We now explore the sensitivity of PM to the setting of a number of its free parameters.

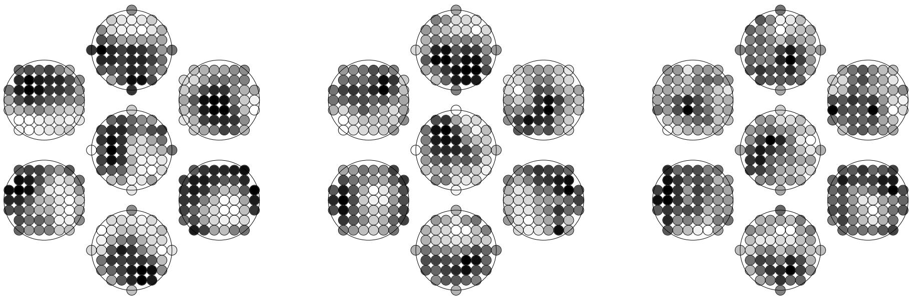

Figure 21. Receptive fields for reference photo with 5% (left), 20% (center), and 40% (right) of pixels corrupted.

Table 6. Binarity of the code after training for different learning rates.

`lrp`:	0.30	0.50	0.70
`lrc`:	0.24	0.40	0.56
binarity:	46.5%	57.5%	85.0%

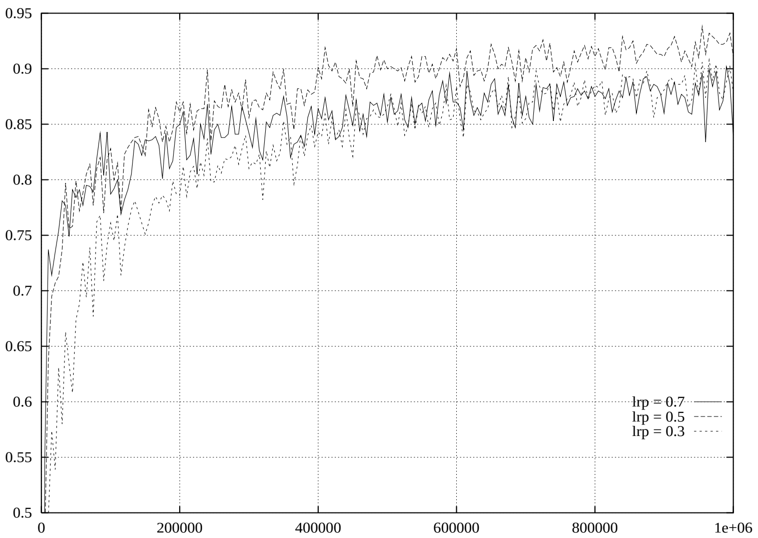

Figure 22. Throughput during training at different learning rates.

3.3.1. Learning Rates In this set of experiments the learning rates `lrc` and `lrp` were varied but kept in the ratio `lrc`/`lrp` = 0.8. A strong correlation between learning rate and binarity after training was evident (Table 6). Figure 22 shows the evolution of throughput during training. While relatively high values were established quickly at all learning rates, asymptotic performance was best for `lrp` = 0.5.

3.3.2. Code Unit Radius For the next set of experiments we fixed `lrp` = 0.7 but changed the receptive field radius for the code units. With the relative overlap fixed, throughput is clearly influenced by the receptive field radius (Figure 23): the highest

Table 7. Binarity of the code after training for various receptive field radii.

code unit radius:	6	7	8	10	12	16
binarity:	94.8%	93.8%	85.0%	75.0%	77.6%	81.9%

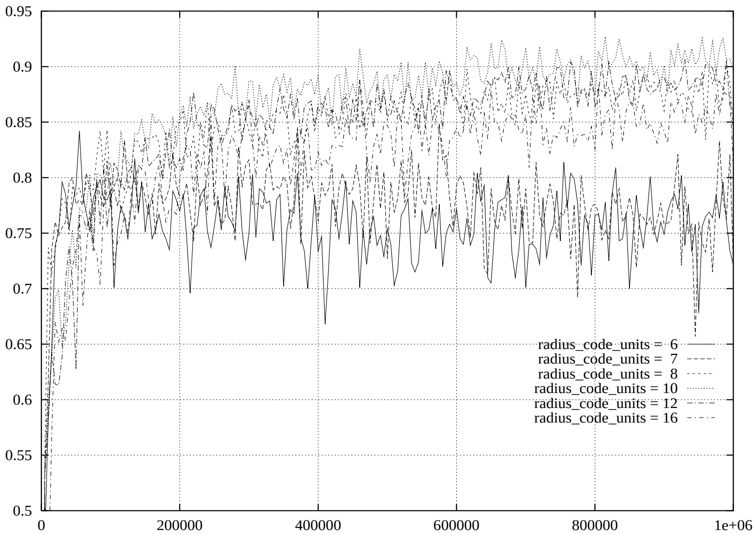

Figure 23. Throughput during training for various receptive field radii.

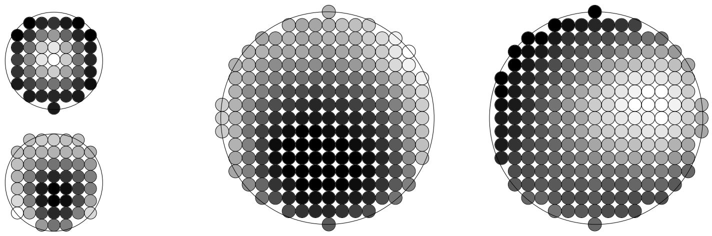

Figure 24. Typical center-surround detectors formed for (left) `lrp` = 0.3, `radius_code_units` = 8; (center and right) `lrp` = 0.7, `radius_code_units` = 16.

rate was achieved for `radius_code_units` = 10, with nearby values (8, 12, and 16) close behind. The smallest receptive fields (`radius_code_units` = 6, 7) resulted in inferior throughput.

The binarity of the produced code (Table 7) shows just the inverse picture: the better the throughput, the worse the binarity, and vice versa. We have already observed this phenomenon in Section 3.1.3; it appears that in such cases the best codes (in terms of information throughput) rely on the use of intermediate activation values. The receptive fields formed by PM here are smooth center-surround structures; they are not necessarily centered with respect to the receptive field. A few typical examples are presented in Figure 24.

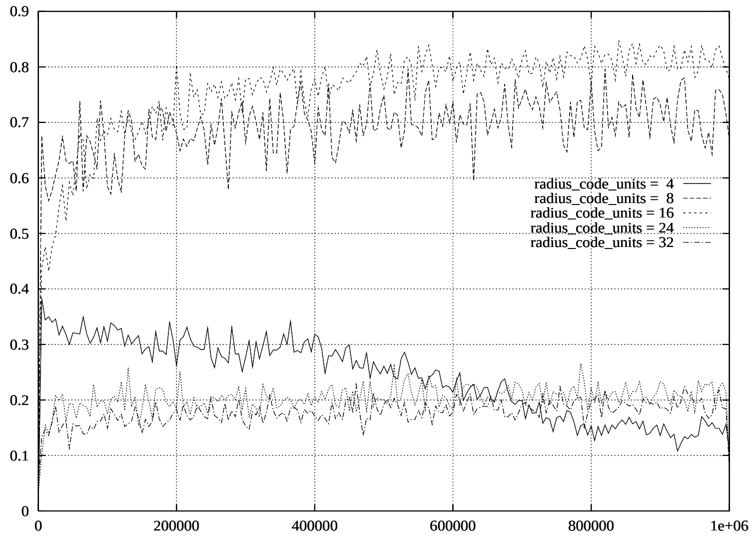

Figure 25. Throughput during training for various receptive field radii, with `radius_input_area` = 32 and `offset_input_units` = 1.

Table 8. Binarity of the code after training for various receptive field radii, with `radius_input_area` = 32 and `offset_input_units` = 1. Compare with Table7.

code unit radius:	4	8	16	24	32
binarity:	99.9%	96.8%	91.3%	97.0%	96.9%

3.3.3. Offset of Input Pixels For the next set of experiments we again varied `radius_code_units`. This time, however, we used the non-standard parameter values `radius_input_area` = 32 and `offset_input_units` = 1. With these settings the input area covers the same area on the image as usual, but with its input units now lying densely in the image pixmap. Thus neighboring input units share more mutual information than before, but the number of inputs to a code unit at a given receptive field size is four times as high.

The throughput (Figure 25) was generally lower, and again showed a strong dependence on receptive field radius: as before, the `radius_code_units` = 8–16 range fared best, with larger or smaller values leading to much inferior results. The slow decline in throughput for `radius_code_units` = 4 is indicative of too large a learning rate. The layout of receptive fields (Figure 3) reveals that in this case each predictor receives input from a large number of code units; to keep the system from diverging, the learning rate for predictors should therefore be reduced accordingly.

The binarity of the produced code (Table 8) was generally higher than before, and again inversely related to the throughput. For `radius_code_units` = 4 it was nearly 100%, and closer inspection revealed most units permanently stuck at zero or one regardless of the input. This explains the low throughput, and was caused by the predictors' learning rate `lrp` being too high.

3.3.4. Exactness Parameter In this set of experiments we examined the influence of the `exactness` parameter — which determines when we consider a code unit output

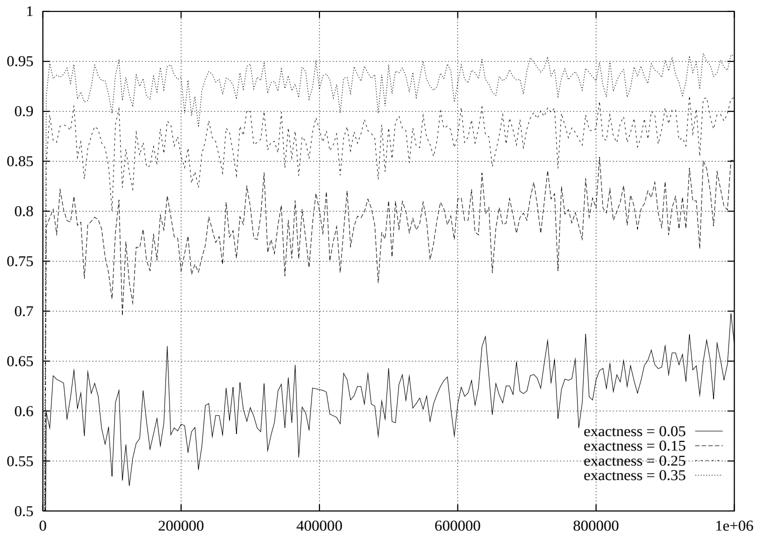

Figure 26. Binarity of the code during training for different values of `exactness`.

to be binary — on our two performance measures, binarity and throughput. We chose `radius_pred_units` = 32 with local receptive fields for the predictor units (see Section 4.1).

Figure 26 shows that as one would expect, the level of binarity measured rises in direct correspondence to the relaxation of the `exactness` parameter. For `exactness` = 0.5, one would of course automatically consider all units binary, and hence always find a binarity of 100%. The effect on throughput is less blatant — while Figure 27 shows a consistent decrease in throughput with increasing `exactness`, the change is very slight until one reaches `exactness` = 0.5. At that extreme, the available space for coding input patterns (as far as the throughput measurement is concerned) shrinks from 3^n to 2^n: since intermediate values are no longer acknowledged as such, a negative impact on throughput is to be expected.

3.3.5. Weight Initialization Here we examine to what extent the weight initialization influences the result of learning. We ran 9 experiments at the default parameter settings but starting from different random initial conditions. Figure 28 shows that the variation in throughput due to initial conditions was smaller than that due to the stochastic nature of the learning process itself. This also indicates that thanks to this stochasticity, semi-linear PM is not likely to get stuck in a local minimum.

3.3.6. Conclusions For code units trained on our reference photo, center-surround receptive field structures are generally formed. The optimal receptive field size appears to be around `radius_code_units` = 10. At this setting, PM develops codes with high throughput but relatively low binarity, indicating that intermediate activation values are utilized to some extent. Our throughput measure is robust with respect to the choice of (moderate) values for the `exactness` parameter, whereas the measured level of binarity is heavily affected by it.

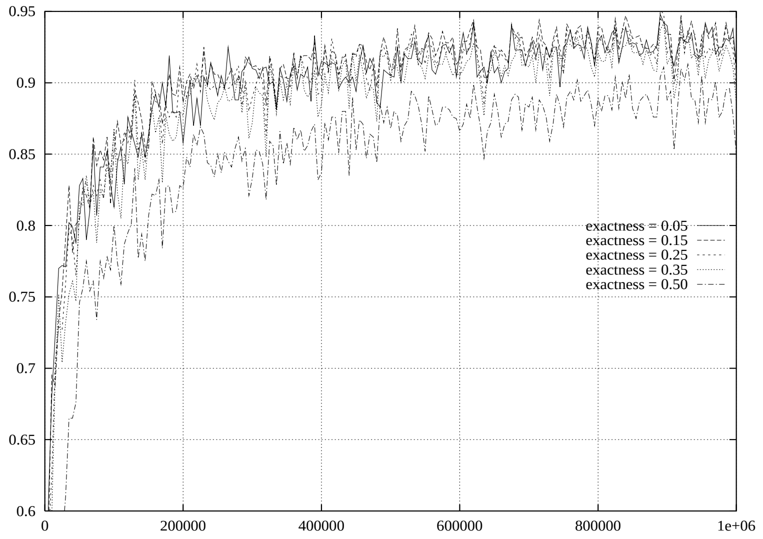

Figure 27. Throughput during training for different values of `exactness`.

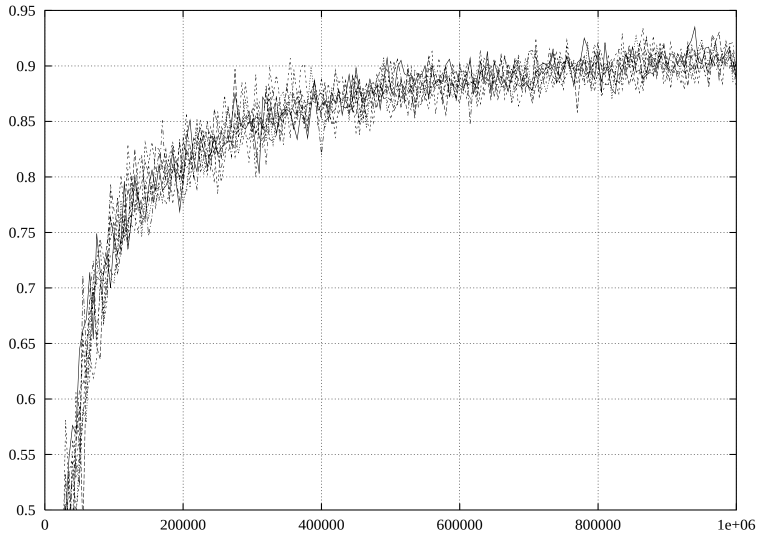

Figure 28. Throughput during training for different random initial weights.

4. Experiments with Architectural Variations

4.1. Local Receptive Fields for Predictors

Up to now, predictor units had access to all code units, except for the one they were learning to predict. In this section we introduce local receptive fields for the predictors. This not only increases the locality of the entire system, but also saves much memory and computing time due to the drastically reduced number of weights. Predictors now receive input only from code units that lie inside their circular receptive field with radius `radius_pred_units`, centered on the position of the code unit they are to

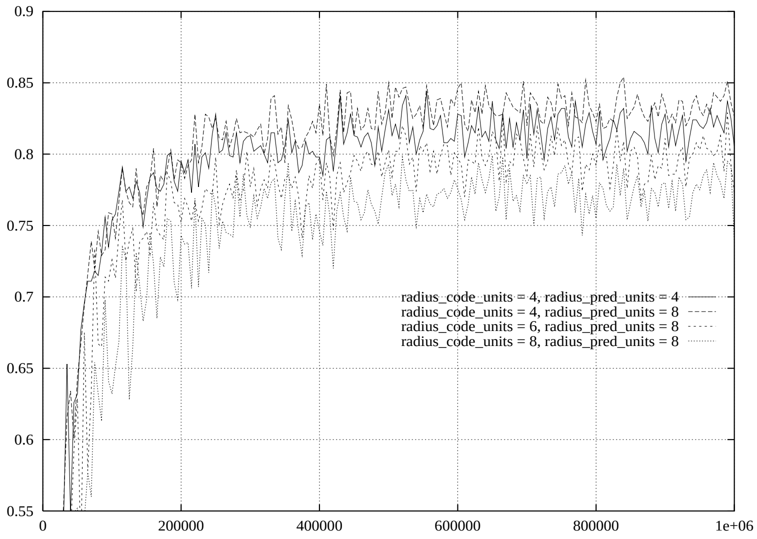

Figure 29. Throughput for predictors with local receptive fields.

Table 9. Binarity for predictors with local receptive fields.

code unit radius:	4	4	6	8
predictor radius:	4	8	8	8
binarity:	82.7%	64.4%	81.5%	85.6%

predict.

4.1.1. Varying Code and Predictor Unit Radius For this set of experiments we fixed `radius_input_area` = 16 and `offset_input_units` = 1, and varied `radius_code_units` and `radius_pred_units`. (Recall that `radius_code_units` not only affects the receptive field size of code units, but also determines the number of code and predictor units in the network.) The throughput resulting from four different combinations of these two parameters is shown in Figure 29. The smaller receptive fields yield better throughput, especially the combination of `radius_code_units` = 4 with `radius_pred_units` = 8. As usual, the binarity (Table 9) is inversely related to the throughput.

4.1.2. Smaller Input Area We now repeated the experiments with a smaller input area, setting `radius_input_area` = 8 while keeping `offset_input_units` = 1. Figure 30 shows that while the general trend — better performance with small receptive fields — remains the same, the differences are far more pronounced now. It becomes clear that for best performance, the system should be biased in favor of the predictors by giving them larger receptive fields than the code units. Again the binarity (Table 10) is inversely related to the throughput.

The receptive fields exhibit well-developed center-surround structure, with the occasional exception at the very edge of the input area. Despite of their significant difference in throughput, the feature detectors found with a predictor unit radius of 4

Table 10. Binarity with local receptive fields and `radius_input_area` = 8.

code unit radius:	4	4	8
predictor radius:	4	8	8
binarity:	88.4%	74.3%	86.7%

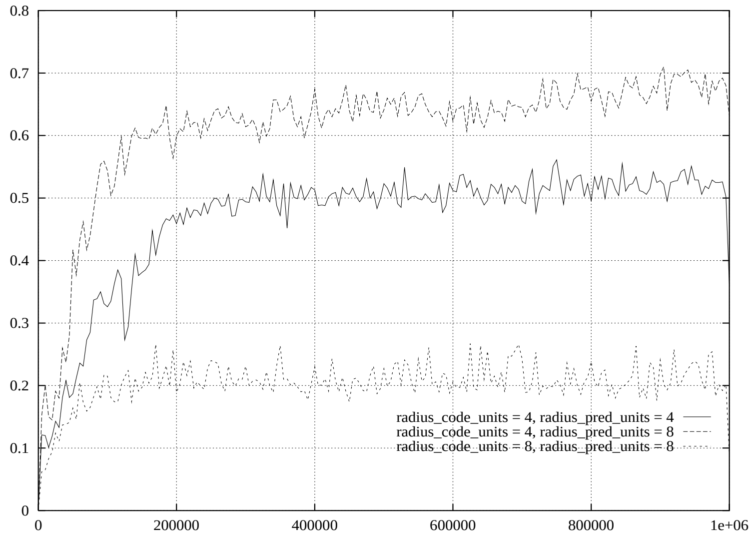

Figure 30. Throughput with local receptive fields and `radius_input_area` = 8.

and 8, respectively, appear qualitatively quite similar, as illustrated in Figure 31.

4.1.3. Larger Offset of Input Pixels We also performed experiments with `radius_input_area` = 16 and `offset_input_units` = 2, *i.e.*, a larger input area that is sampled more sparsely, so that the number of inputs to each code unit remains the same. Figure 32 shows the evolution of throughput for the five combinations of code unit and predictor radii that we tested.

With `radius_code_units` = `radius_pred_units` = 4, a gradual decline in throughput can be observed; this is the typical signature of too high a learning rate (*cf.* Figure 25). This is confirmed by the binarity (Table 11), which at near 100% indicates that learning has all but ceased in this case. At the other end of the spectrum, the combination `radius_code_units` = `radius_pred_units` = 16 exhibits a lower *asymptotic* level of throughput, reinforcing our previous finding that too large a `radius_code_units` is detrimental to throughput.

Although the intermediate receptive field sizes all yield comparable throughput at the end of our experiment, note that for the combination `radius_code_units` = 4, `radius_pred_units` = 8 it was much slower to develop. The exceptionally low binarity in this case suggests that given more training time, the throughput of this combination would continue to improve, to asymptotically outperform all others.

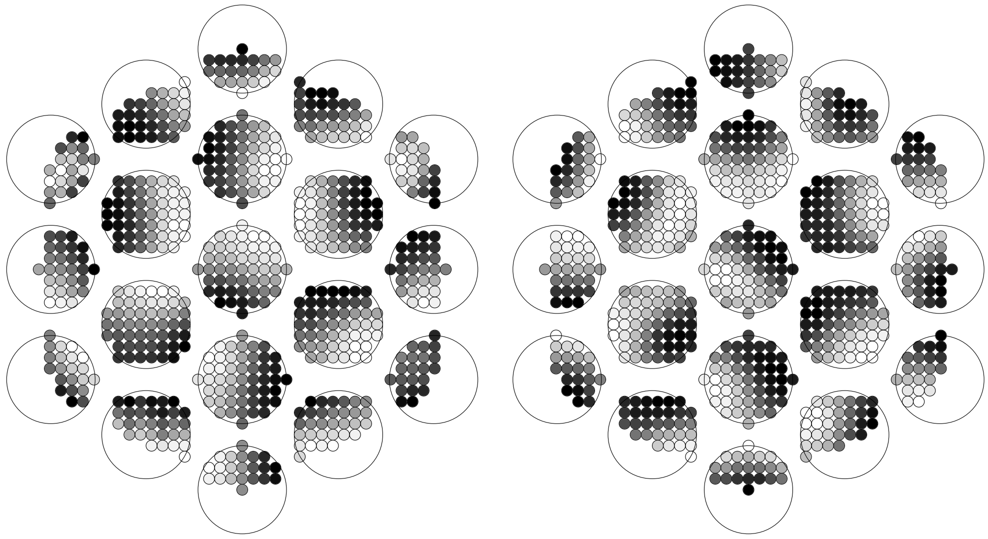

Figure 31. Feature detectors for predictors with local receptive fields, `radius_input_area` = 8, and `radius_code_units` = 4. No clear difference between `radius_pred_units` = 4 (left) and `radius_pred_units` = 8 (right) is evident.

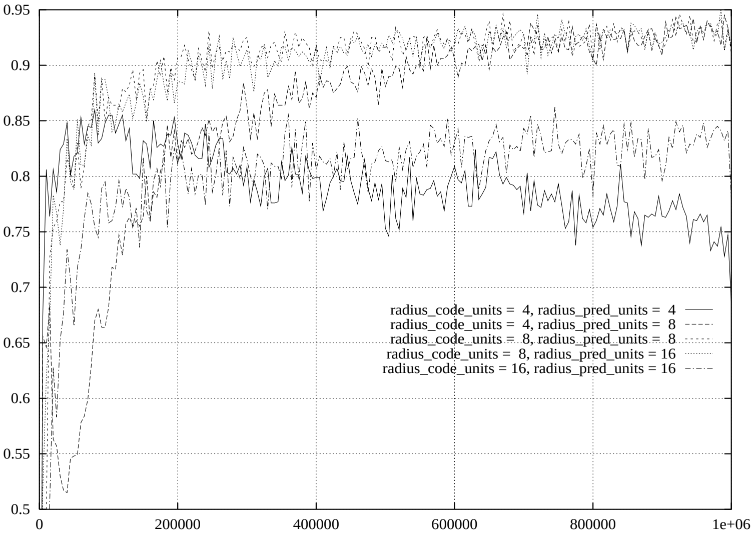

Figure 32. Throughput for predictors with local receptive fields, using `radius_input_area` = 16 and `offset_input_units` = 2.

Table 11. Binarity for predictors with local receptive fields, using `radius_input_area` = 16 and `offset_input_units` = 2.

code unit radius	4	4	8	8	16
predictor radius	4	8	8	16	16
binarity	98.6%	45.4%	79.9%	67.7%	86.6%

Table 12. Binarity for a column of seven code units *vs.* receptive field radius.

code unit radius:	4	8	16	32
binarity:	85.8%	84.7%	91.8%	98.4%

4.2. Columnar Arrangement of Code Units

Since it is well-known that feature detectors in the visual cortex of mammals are arranged in columns, we also performed experiments where all receptive fields fully overlap, *i.e.*, all code units are arranged in a "column" at the same input position. Now each code unit must try to extract a statistically independent feature from exactly the same input as all its peers. Since its predictor now has access to code units that receive identical input, the code unit's task is far more difficult under this arrangement.

4.2.1. Seven Layers, Varying Receptive Field Size In the first set of experiments we used `number_code_layers` = 7 layers‡ of code units, each comprising a single code unit and its predictor. We used `offset_input_units` = 1, `radius_input_area` = 16, and varied `radius_code_units` between 4 and 32. Since we only have one code unit per layer, this setup also varies the *effective* input area.

Table 12 shows that the binarity of the resulting code is at a high 85% for code unit radii of 4 and 8, then rises further with increasing receptive field size. As in most of our experiments, the throughput (Figure 33) exhibits the inverse behaviour: after a marked improvement from radius 4 to 8, it declines again for larger values. It thus appears that there is an optimal receptive field size for high information throughput on our reference image at around `radius_code_units` = 8.

Even at the best receptive field size, however, the throughput appears rather small though compared to some of our other experiments. It must be kept in mind though that only 128 different patterns can be represented with a binary code across seven units; for a three-valued code this number is 2187. Since we use `num_pat_train_test` = 1000 patterns to test for throughput, it should come as no surprise that a near-binary code cannot produce a distinct output for each of these test patterns. In fact, the system appears quite adept at using a small proportion of non-binary (intermediate) activation values to raise throughput above 0.128, the maximum that a purely binary code could achieve.

The receptive fields of the trained system are typically smooth and spatially well-organized. Smaller receptive fields generally have center-surround structure; larger ones often combine two or more center-surrounds into more complex, oriented feature detectors. Figure 34 shows an intermediate situation, with center-surrounds present both individually and doubled up into oriented bar shapes.

4.2.2. Nineteen Layers, Varying Receptive Field Size This set of experiments used the same parameter settings as the preceding one, except that now there are 19 layers§. Extracting 19 independent features from the same input is a much harder

‡ That is, seven code units per receptive field location. Note that our (biologically inspired) notion of "layer" differs from the term's common usage in neural network research.

§ The numbers 7 and 19 are not quite arbitrary. Although this is irrelevant for the experiments, they were chosen because they produce fully populated hexagons in our receptive field diagrams.

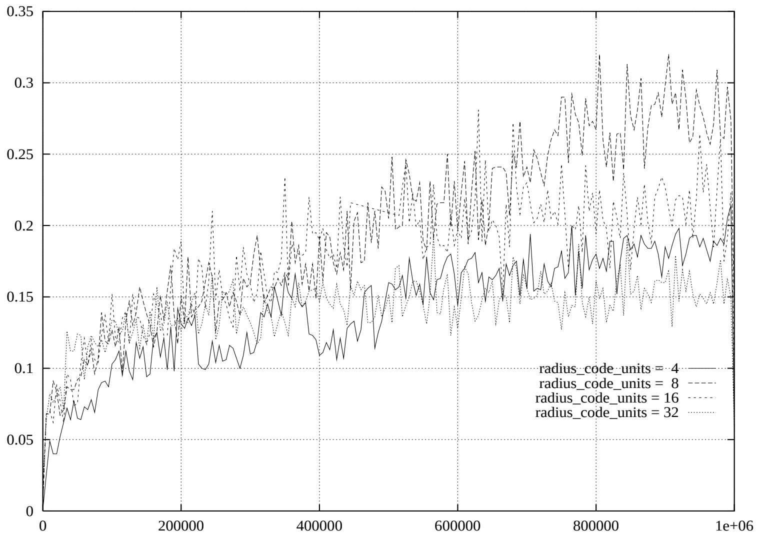

Figure 33. Throughput for a column of seven code units *vs.* receptive field radius.

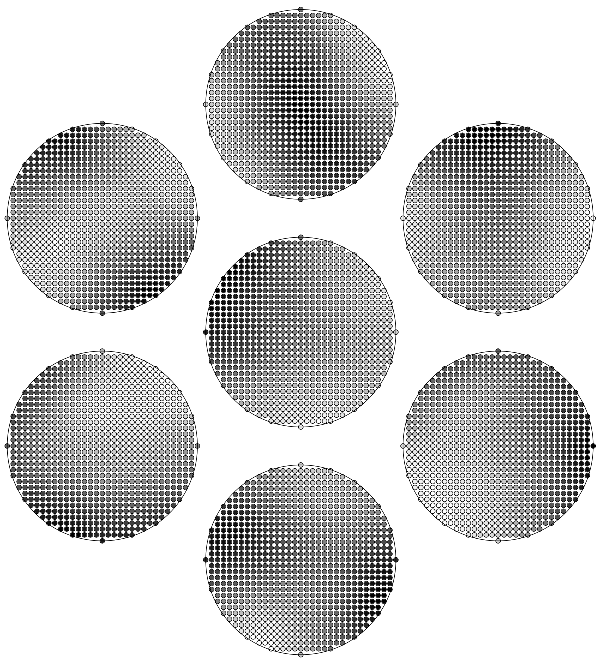

Figure 34. The receptive fields learned by a column of seven code units with `radius_code_units` $= 16$. Both center-surround and bar-shaped feature detectors are formed.

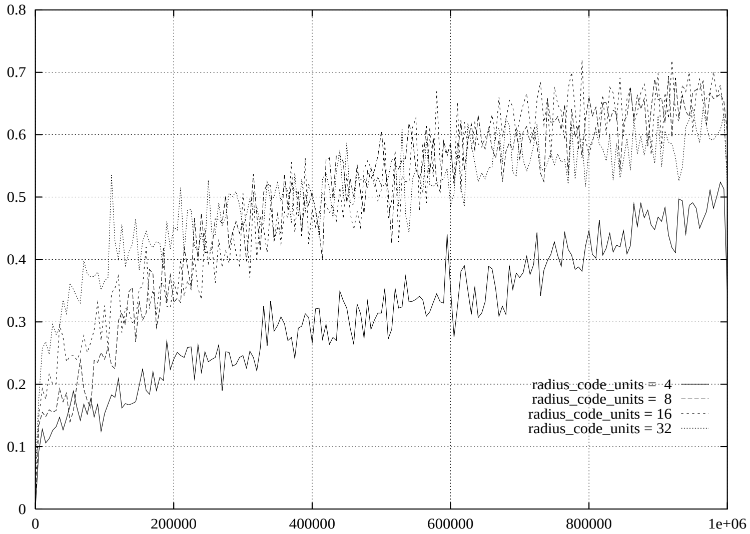

Figure 35. Throughput for a column of 19 code units *vs.* receptive field radius.

Table 13. Binarity for a column of 19 code units *vs.* receptive field radius.

code unit radius:	4	8	16	32
binarity:	84.0%	81.8%	94.7%	98.5%

Table 14. Binarity for a column of code units *vs.* the number of layers, with `radius_code_units` = 16.

code layers:	7	10	13	16	19
binarity:	91.8%	94.2%	95.7%	96.1%	94.8%

task; after `num_pat_train` = 1,000,000 training patterns the throughput (Figure 35) is still growing.

As before, throughput is quite low for `radius_code_units` = 4 — it appears that such a small receptive field simply does not provide enough information about the image to readily permit the extraction of many independent features. By contrast, there is almost no decline in throughput for large receptive field sizes here. Since more code units are now available to produce distinct output patterns, the throughput also reaches much higher values now.

The binarity of the code units — shown in Table 13 — as well as the receptive fields they develop (see Figure 36 for an example) are very similar to those obtained for 7 layers.

4.2.3. Varying Number of Layers In the third set of experiments we kept `radius_code_units` = 16 and `offset_input_units` = 1 but varied the number of layers between 7 and 19 in increments of 3. In Figure 37 it can be observed that throughput improves with increasing number of layers. Up to `number_code_layers` = 16 the improvement with each increment is greater than what could be expected

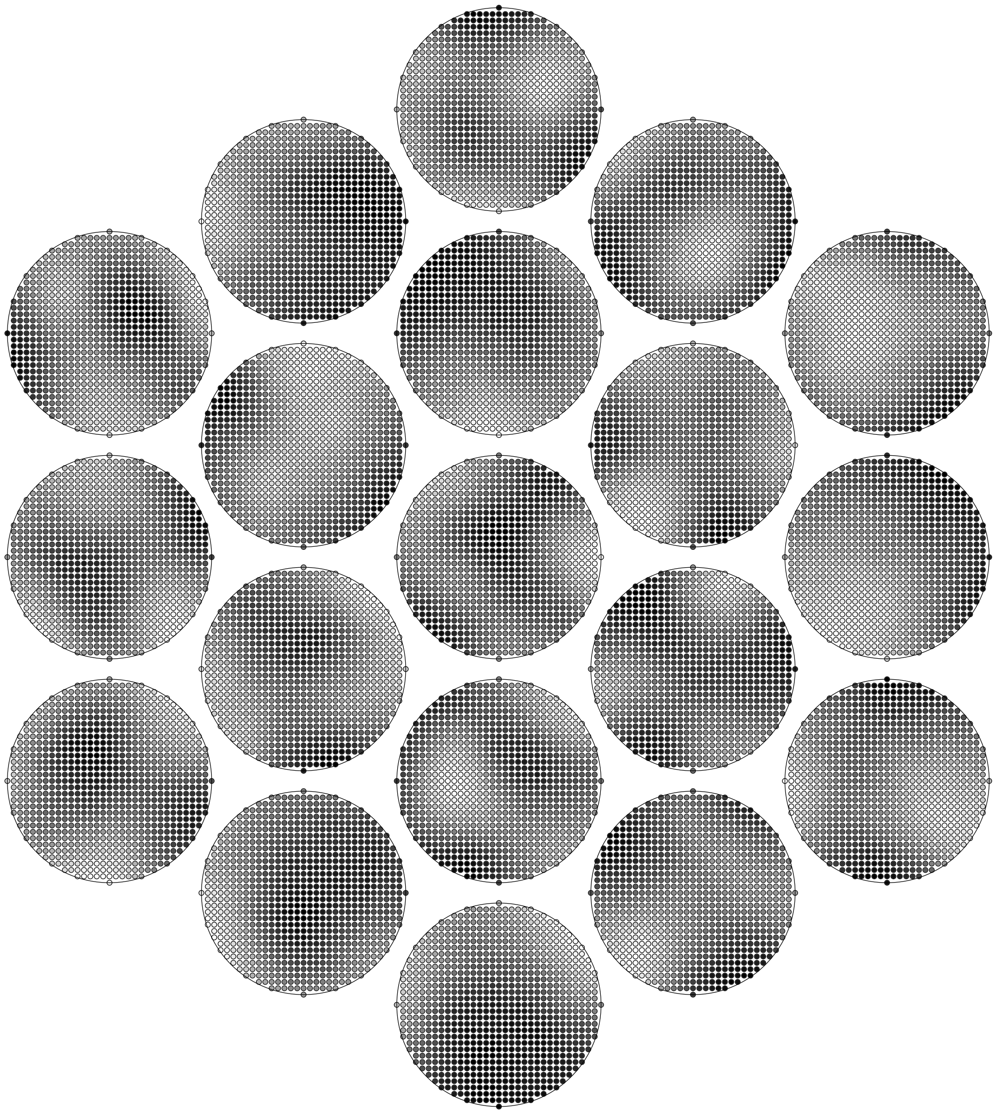

Figure 36. The receptive fields learned by a column of 19 code units with `radius_code_units` = 16. Often several center-surround detectors combine to form more complex, oriented structures.

simply due to the greater length of output codes; from 16 to 19 layers, however, it is smaller. This suggests that in our limited training time, the 19-layer system did not learn to utilize its three extra code units for the extraction of additional information (as compared to the 16-layer system) from the reference image.

Interestingly, the binarity of the resulting code (Table 14) also increases up to 16 layers, then declines again slightly. In contrast to the normally observed negative correlation between throughput and binarity, the latter here appears to be indicative of the efficiency of information transfer through the trained system. We suspect that this was facilitated by all code units having identical receptive field parameters in this

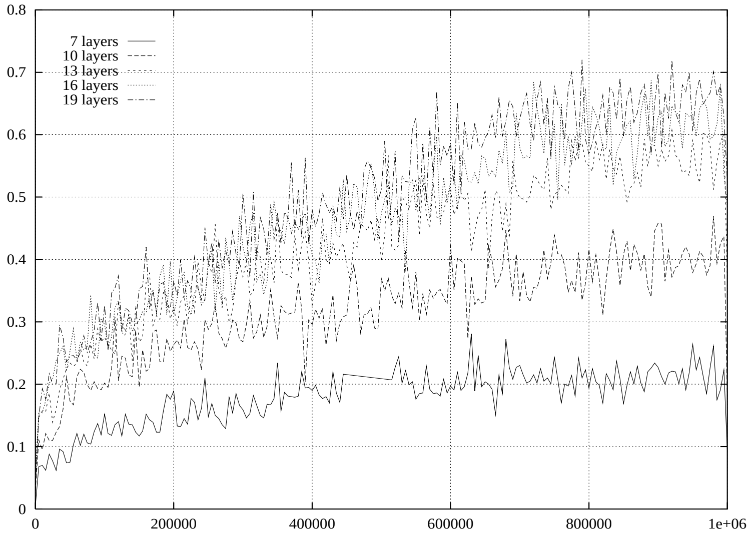

Figure 37. Throughput for a column of code units *vs.* the number of layers, with `radius_ code_units` = 16. The stretch of missing values in the curve for 7 layers is due to a hard disk having filled to capacity during the experiment (also visible, albeit less prominently, in Figure 33). This did not affect the learning process.

Table 15. Binarity for predictors with local receptive fields, using `number_code_layers` = 7, `radius_input_area` = 8, and `offset_input_units` = 1.

code unit radius:	4	8
predictor radius:	4	8
binarity:	79.1%	57.4%

experiment.

The receptive fields formed in this set of experiments feature the same combinations of center-surround detectors found in the preceding two; they smoothly range from single or double occurrence (within the same receptive field) for 7 layers (Figure 34) to predominantly triplets for 19 layers (Figure 36).

4.3. Multiple Columns of Code Units

We now combine two ideas presented previously: in addition to having `number_code_layers` = 7 code units (and predictors) at each receptive field location (*cf.* Section 4.2), we replicate these columns on a hexagonal grid, with local receptive fields for the predictors as well as the code units, as introduced in Section 4.1. Thus predictors now receive input from the other code units in “their” column as well as from those in neighboring columns. Conversely, code units must now try to extract multiple features that are both independent (so as to evade prediction via code units within their column) and local (to evade prediction from neighboring columns).

4.3.1. Varying Code and Predictor Unit Radius The first experiments were performed with `radius_input_area` = 8, `offset_input_units` = 1, and

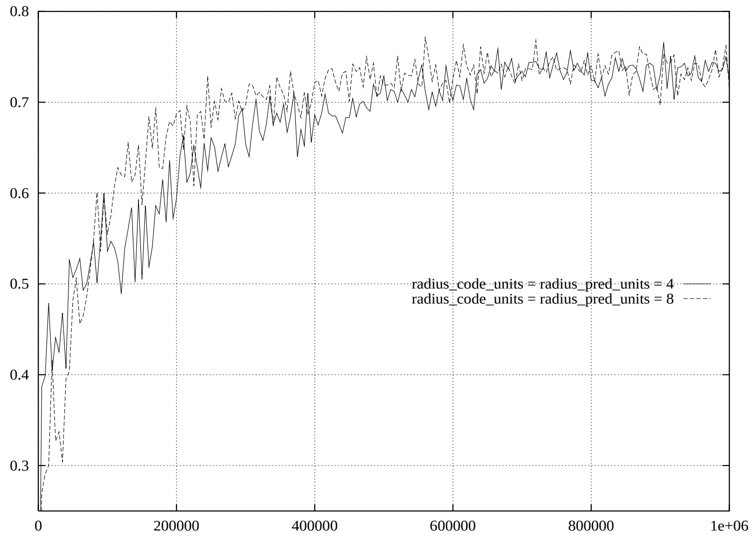

Figure 38. Throughput for predictors with local receptive fields, using `number_code_layers` = 7, `radius_input_area` = 8, and `offset_input_units` = 1.

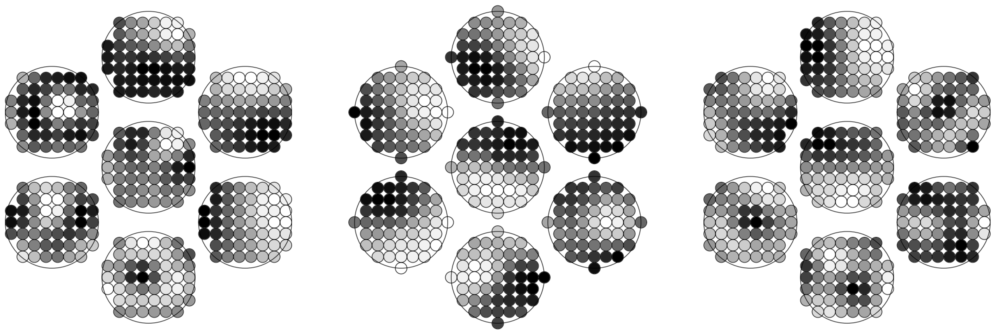

Figure 39. Three columns of feature detectors from the experiment using `radius_code_units` = `radius_pred_units` = 4.

`radius_code_units` = `radius_pred_units` = 4 and 8, respectively. Figure 38 shows that the throughput develops almost identically in both cases, though the smaller receptive fields did produce a more binary code (see Table 15).

Figures 39 and 40 show a few columns of feature detectors from these experiments. It is evident that within each column, code units tend to focus on different aspects of their common input. The difficult requirement for code units to extract multiple independent local features is now no longer met by simple center-surround detectors alone; we can observe oriented bars and even hyperbolic shapes. Despite of this proliferation of complex structure, the receptive fields always appear spatially well-organized.

4.3.2. Larger Offset of Input Pixels As before, we repeated our experiments with a larger but more sparsely sampled input area (`radius_input_area` = 16, `offset_input_units` = 2). Figure 41 shows that learning proceeds very slowly for

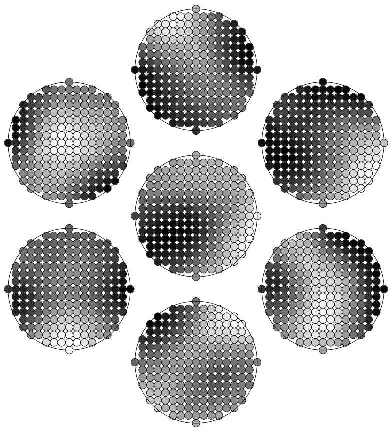

Figure 40. A column of feature detectors from the experiment using `radius_code_units` = `radius_pred_units` = 8.

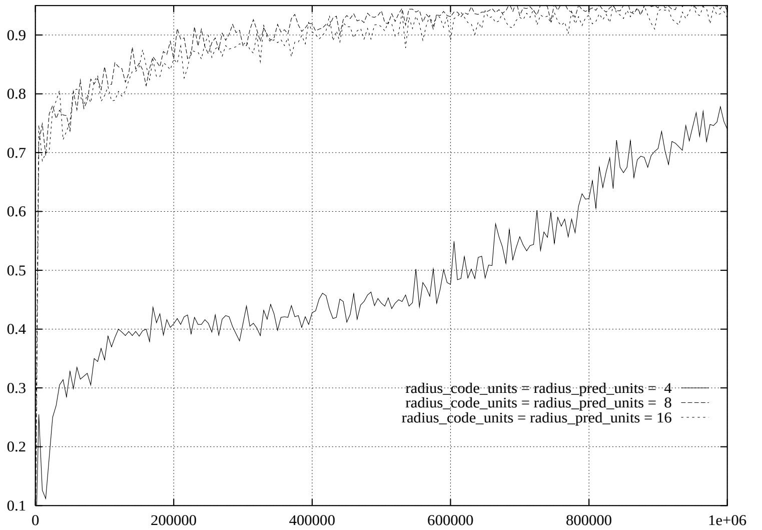

Figure 41. Throughput for predictors with local receptive fields, using `number_code_layers` = 7, `radius_input_area` = 16, `offset_input_units` = 2.

`radius_code_units` = `radius_pred_units` = 4. We attribute this to the fact that with such small receptive fields, the system must attempt to extract 7 independent local features from less than a dozen inputs — surely a rather difficult task. Visual inspection of the feature detectors at the end of this experiment revealed no obvious spatial organization, even though the throughput was beginning to pick up in the second half of training. The extremely low binarity in this case (Table 16) suggests though that given enough time, PM may be able to achieve high throughput even here. For the larger receptive field sizes we obtained good results, similar to those reported above (Section 4.3.1). Receptive fields were always well-structured; we show a full set from one experiment in Figure 42.

Table 16. Binarity for predictors with local receptive fields, using `number_code_layers` = 7, `radius_input_area` = 16, `offset_input_units` = 2.

code unit radius:	4	8	16
predictor radius:	4	8	16
binarity:	5.6%	51.4%	84.5%

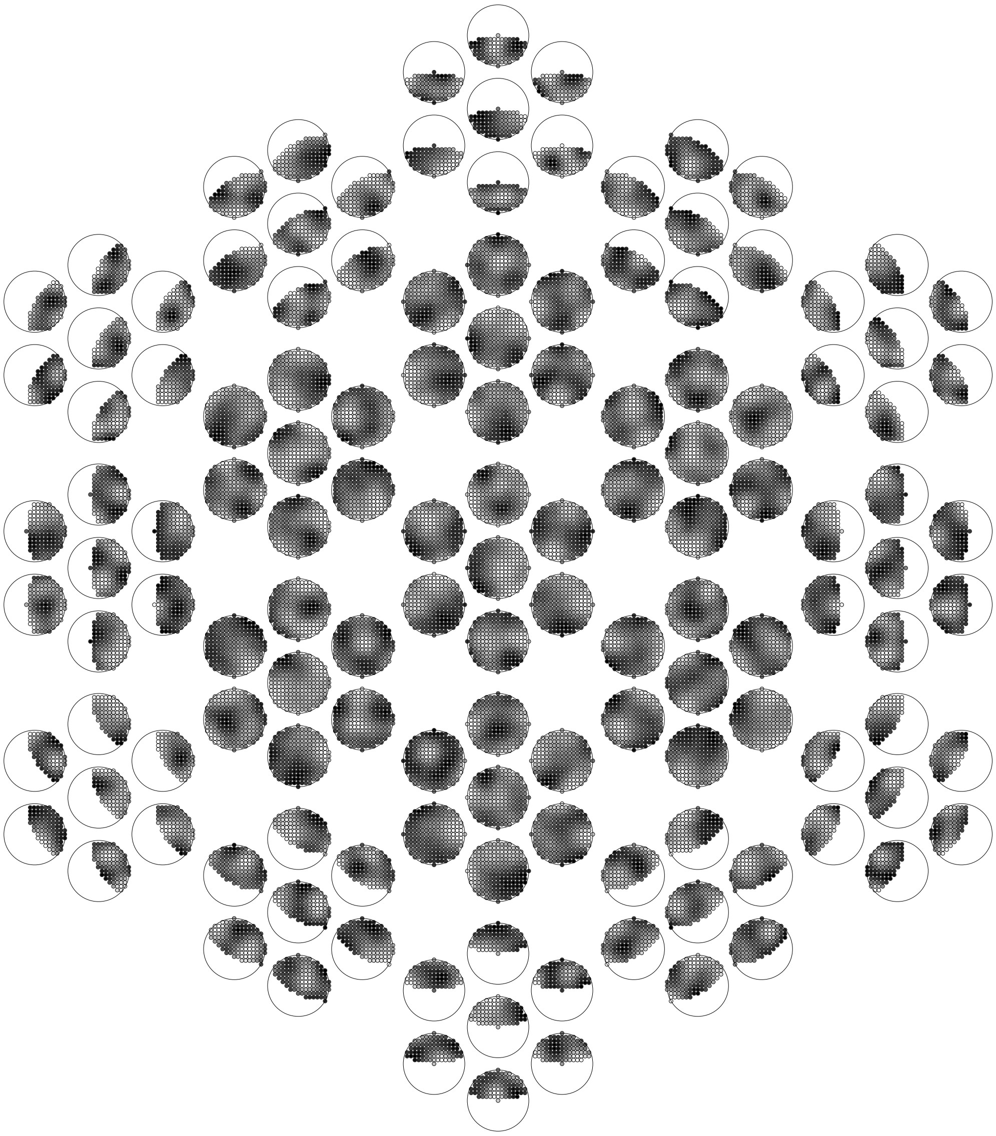

Figure 42. All feature detectors that were formed in the experiment with `radius_code_units` = `radius_pred_units` = 16. Each rosette represents a column of `number_code_layers` = 7 units.

5. Comments and Conclusions

5.1. Tradeoff between Throughput and Binarity

Very often we observe a tradeoff between the binarity of the code and the throughput reached. Specifically, if the binarity rises too quickly in the beginning of the learning process, driving all units into saturation, it becomes very difficult to change the system's reaction, because the gradients used in the delta learning rule become very small. Since units start out with random initial weights (*i.e.*, low mutual information with the input), they tend to initially produce similar activation for all inputs, and so the throughput of the system will remain very weak.

There are at least two ways to improve this behaviour. One way would be to use a learning rule that normalizes the gradient in some appropriate manner, *e.g.* by using only the sign of the derivative, or adding a small constant to it. In this way even saturated units could be retrained, and the system would retain some plasticity even in states with very high binarity. A second approach would be to have each code unit try to maximize both its variance and its unpredictability explicitly. In order to avoid additional free parameters there should be an automatic weighting of these two terms. In our experiments it seems generally advantageous to set parameters such that binarity and throughput can both rise to a high level; a good way to achieve this is to let the throughput develop well ahead of the binarity.

5.2. Performance Measures

Another point is how we measure the performance of our systems. Firstly, consider the binarity of the code. The task of a hypothetical subsequent system typically becomes simpler if we provide it with binary input patterns: the search space is discrete, hence comparatively small, and patterns are either identical, or they differ clearly. This helps in particular classical (discrete) learning algorithms and statistical tools such as Bayesian classifiers. However, it is not clear whether high binarity will always help a subsequent *neural* system to perform better. Consider our results for images at different quantization levels (Section 3.2.2), for instance: PM applied to a moderately quantized image (with plenty of information left) outperformed experiments on the same image reduced to fewer grey levels.

The binarity of a code, however, is strongly influenced by the `exactness` parameter (*cf.* Section 2.5): the larger `exactness` is, the higher a binarity we will measure for the same code. The `exactness` also affects our second measurement, the throughput: as observed in Section 4.2, a higher throughput can easily result from having more intermediate ("i") values in the code. The number of activations classified "i", however, is again determined by `exactness`. On the other hand, in Section 3.3.4 we have seen that as long as `exactness` does not become too large, this effect remains negligible.

A third, implicit measure for the quality of our systems is the visual inspection of its receptive fields. Intuitively, we would like to see smooth, topologically well-organized structure in the receptive field, preferably of a simple, easily identified type: center-surround fields, oriented edge detectors, and so forth. On the other hand, why should the code produced by those receptive fields necessarily be better than one for which we do not have such a handy label? In Section 3.1 we measured very high throughput for the "checkers" image even though there was no readily identifiable structure (other than sparseness) in the receptive fields. Since there was no noise in this image, we must conclude that PM has managed to exploit some non-obvious (to us!) structure in it in order to reach this level of performance. Should we disallow such solutions simply because they do not match our preconceived notions? In that case, we should perhaps introduce Kohonen-style interactions between neighboring weights in order to restrict ourselves *a priori* to the smooth receptive fields that we are most familiar with.

5.3. Number of Test Patterns

While we computed the throughput at the end of training using `num_pat_test` = 5000 random patterns, intermediate tests were done with only `num_pat_train_test` = 1000 patterns for performance reasons. A comparison of intermediate and final values reveals that the measured throughput tends to be consistently lower in the final test — raising the question whether we are approaching some upper limit to the number of different codes that a given set of trained feature detectors can produce. It would be interesting to test this hypothesis by measuring the throughput of trained code units for varying numbers of random test patterns.

5.4. Choice of Learning Rates

An appropriate choice of learning rates for predictors and code units is crucial for the system's success. At too large rates, the system does not learn, and the throughput decreases steadily from an initial high. Only in rare cases the system was able to recover from this failure mode. Too small learning rates produce a similar result: the throughput first rises a little, but then starts to decline, never to recover. In practice, however, we have not had much difficulty finding appropriate learning rates, and often found the usable range of values to stretch over up to two orders of magnitude.

A more significant problem is the occasional tendency of PM to develop highly binary units prematurely. Unlike, *e.g.*, in plain backpropagation networks, this can not be prevented simply by using smaller learning rates for the entire system. Our observations of this phenomenon suggest, however, that increasing the *ratio* between the learning rates for predictors and code units may help in such cases. An even more interesting solution would be an automatic adaptation of learning rates (Schraudolph 1999) designed to prevent premature binarity (*cf.* Section 5.1).

6. Related Approaches

Redundancy reduction is widely regarded as an important goal of unsupervised learning — see *e.g.* (Barlow, Kaushal and Mitchison 1989, Atick, Li and Redlich 1992, Schmidhuber 1999); also compare (Linsker 1988, Földiák 1990, Deco and Parra 1997, Miller 1994, Field 1994, Schraudolph 1995). But how can this goal be achieved in a massively parallel, local, efficient, and biologically plausible way? Here we briefly characterize PM's relationship to other methods — see also (Schmidhuber, Eldracher and Foltin 1996):

- Unlike certain inherently sequential methods, *e.g.*, (Rubner and Schulten 1990), PM can be implemented in parallel.
- Unlike *e.g.* (Barrow 1987), there may be many simultaneously active code units (multiple "winners" instead of just one), as long as they represent *different* aspects

of the environment. In other words, we obtain a distributed rather than local encoding.

- Unlike a number of other approaches, there is no need to perform expensive, non-local calculations, such as computing the derivatives of determinants of covariance matrices (Linsker 1988), or inverting a weight matrix (Bell and Sejnowski 1995).
- Unlike *e.g.* (Deco and Parra 1997, Földiák 1990, Rubner and Tavan 1989), and anti-Hebbian systems in general, PM requires neither time-consuming settling phases (due to recurrent connections), nor analytic computation of weight vectors.
- Like Independent Component Analysis (ICA) methods (Jutten and Herault 1991, Cardoso and Souloumiac 1993, Comon 1994, Molgedey and Schuster 1994, Bell and Sejnowski 1995, Amari, Cichocki and Yang 1996, Nadal and Parga 1997, *etc.*) PM learns to create *statistically independent* code components, as opposed to merely *decorrelated* ones. For efficiency reasons, most statistical classifiers (*e.g.*, Bayesian pattern classifiers) assume statistical independence of their input variables.
- Like LOCOCODE (Hochreiter and Schmidhuber 1999), but unlike most ICA methods, PM does not need to know the number of statistically independent sources in the input *a priori*.
- Like LOCOCODE, and the symplectic map approach (Parra, Deco and Miesbach 1996), PM has a potential to discover *non-linear* redundancy in the input data, and to generate appropriate redundancy-free codes. Note that PM predates these two alternatives.
- Like binary information gain optimization (Schraudolph and Sejnowski 1993), but unlike most other "neural" methods (see references above), existing variants of PM create *binary* rather than *continuous* codes. Binary codes facilitate post-training analysis of the code, and correspond to a higher rate of data compression.

7. Summary

We have implemented semi-linear PM in an image-processing system that features sub-sampling and random rotation of image patches, as well as spatially organized, partially overlapping, localized receptive fields for both predictor and code units. Using this software, PM has been applied to a variety of images over a wide range of image preprocessing, network architecture, and learning rate parameters. Center-surround, bar-like, and more complex feature detectors have been observed to form depending on experimental conditions. Information throughput and data compression ("binarity") of the resulting code have been measured and compared in various settings. The software permits independent variation of the degree of overlap for the receptive fields of code and predictor units, and the effect of such variation has been investigated.

Having shown that no artefacts are learned by PM from white noise, we tested it on regular and irregular artificial input patterns first. A real-world photo was then used as input, and two different ways of distributing the code units' receptive fields were explored. We could show that our results are independent of the network's random initialization, and that they are similar for a variety of equally pre-processed photos. While PM is sensitive to how images are pre-processed, we found it quite robust to noise.

In further experiments, we introduced code units with fully overlapping receptive fields, forming columns of feature detectors. We then showed that restricting predictors to have only local receptive fields does not significantly affect results. Combining these two modifications, we then applied PM to a network with several partially overlapping layers of code units. Finally, we studied some of the previously introduced variants for the case where there was no rotation of the input area from which the image was sampled.

Throughput and binarity of the code, the two measurements we have used to see how good a preprocessing stage PM provides for subsequent processing, were generally high at the end of training. Throughput rose very quickly in particular for noisy images (see Section 3.2.3) and those with many grey levels (see Section 3.2.2). Binarity was generally far slower to develop, except for some artificial images (see Section 3.1) where premature binarity hindered the development of a code with high throughput. This tradeoff between binarity and throughput was observed frequently.

Generally PM produced well-structured, topologically smooth receptive fields, even when predictors had only local connectivity (see Section 4.1), or when multiple code units (and predictors) were arranged in columns (see Sections 4.2 and 4.3). The main exceptions were experiments with very small receptive fields, or with random noise as input. However, the receptive fields formed do not always correspond to those known from, say, the mammalian visual system, or classical image processing algorithms; nor do a high throughput and binarity reliably predict such structures.

Acknowledgments

We would like to thank Bernhard Foltin for the help he provided along with his older programs, and our former colleagues at the Technische Universität München (chair of Prof. Dr. Brauer) for the use of their computing cycles to conduct some of the experiments reported here. This work was supported by the Swiss National Science Foundation under SNF grant #2100–045700.95/1.

References

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

Atick J J, Li Z and Redlich A N 1992 Understanding retinal color coding from first principles *Neural Computation* **4**(4) 559–572

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 *Proceedings of the IEEE 1st Annual Conference on Neural Networks* Vol IV IEEE pp 115–121

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

Deco G and Parra L 1997 Non-linear feature extraction by redundancy reduction in an unsupervised stochastic neural network *Neural Networks* **10**(4) 683–691

Field D J 1994 What is the goal of sensory coding? *Neural Computation* **6**(4) 559–601

Földiák P 1990 Forming sparse representations by local anti-Hebbian learning *Biological Cybernetics* **64** 165–170

Hochreiter S and Schmidhuber J 1999 Feature extraction through LOCOCODE *Neural Computation* **11**(3) 679–714

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

Lindstädt S 1993 Comparison of two unsupervised neural network models for redundancy reduction *in* M C Mozer, P Smolensky, D S Touretzky, J L Elman and A S Weigend (eds) *Proceedings of the 1993 Connectionist Models Summer School* Hillsdale, NJ: Erlbaum Associates pp 308–315

Linsker R 1988 Self-organization in a perceptual network *IEEE Computer* **21** 105–117

Miller K D 1994 A model for the development of simple cell receptive fields and the ordered arrangement of orientation columns through activity-dependent competition between on- and off-center inputs *Journal of Neuroscience* **14**(1) 409–441

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

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

Parra L, Deco G and Miesbach S 1996 Statistical independence and novelty detection with information preserving nonlinear maps *Neural Computation* **8**(2) 260–269

Rubner J and Schulten K 1990 Development of feature detectors by self-organization: A network model *Biological Cybernetics* **62** 193–199

Rubner J and Tavan P 1989 A self-organization network for principal-component analysis *Europhysics Letters* **10** 693–698

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

Schmidhuber J H 1993 Netzwerkarchitekturen, Zielfunktionen und Kettenregel Habilitationsschrift, Institut für Informatik, Technische Universität München

Schmidhuber J H 1999 Neural predictors for detecting and removing redundant information *in* H Cruse, J Dean and H Ritter (eds) *Adaptive Behavior and Learning* Kluwer in press

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

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

Schraudolph N N 1995 *Optimization of Entropy with Neural Networks* PhD thesis University of California, San Diego

Schraudolph N N 1999 Local gain adaptation in stochastic gradient descent *Technical Report IDSIA-09-99* IDSIA, Corso Elvezia 36, CH-6900 Lugano, Switzerland

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 Cowan and C L Giles (eds) *Advances in Neural Information Processing Systems* Vol 5 Morgan Kaufmann, San Mateo, CA pp 499–506