Generalized Softmax Networks for Non-linear Component Extraction

Jörg Lücke and Maneesh Sahani

Gatsby Computational Neuroscience Unit, UCL, London WC1N 3AR, UK

Abstract. We develop a probabilistic interpretation of non-linear component extraction in neural networks that activate their hidden units according to a softmax-like mechanism. On the basis of a generative model that combines hidden causes using the $\max$-function, we show how the extraction of input components in such networks can be interpreted as maximum likelihood parameter optimization. A simple and neurally plausible Hebbian Δ-rule is derived. For approximately-optimal learning, the activity of the hidden neural units is described by a generalized softmax function and the classical softmax is recovered for very sparse input. We use the bars benchmark test to numerically verify our analytical results and to show competitiveness of the derived learning algorithms.

1 Introduction

Neural network models that can be applied to unsupervised data classification, i.e. clustering, have been well understood in terms of generative models. For mixture models [1], the learning rule was shown to involve the softmax (see, e.g., [2] for an overview) as activation function for the hidden units. Other than clustering, recent neural network models have been shown to be competitive in extracting input components. These networks, e.g. [3,4,5], also use softmax-like mechanisms but a probabilistic understanding in terms of multiple causes models has not yet been developed. In this paper we will show how and under which conditions learning in feed-forward networks with Hebbian plasticity can be understood as maximization of the data likelihood under a non-linear generative model.

2 The Neural Network Model

Consider the network of Fig. 1 which consists of D input units with values $y_1, \ldots, y_D$ and H hidden units with values $g_1, \ldots, g_H$. An input $\boldsymbol{y}$ to the system is represented by activities of input units. Via connections between the input and hidden layer that are parameterized by (W_{id}) the input unit activities determine the activities of the hidden units, $g_i = g_i(\boldsymbol{y}, W)$. The parameters (W_{id}) will be called weights.

Given a set of input patterns we want to adjust the weights W such that the activities of the hidden units appropriately represent the input patterns. Learning will thus depend on the distribution of input patterns. A standard approach in neural network modeling is to use a Δ-rule with divisive normalization. In the case of Hebbian learning the Δ-rule reads:

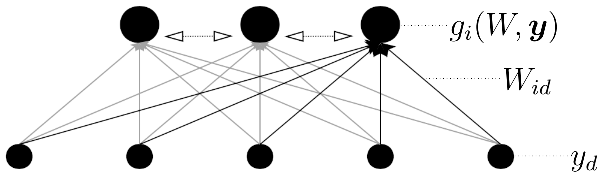


Fig. 1. Architecture of a two layer neural network. Input is represented by values y_1 to y_D of D input units (small black circles). Depending on the input and on the parameters W the activities of the hidden units g_1 to g_H (big black circles) are computed. The dotted horizontal arrows symbolize lateral information exchange that is in general required to compute the functions g_1 to g_H. After the g_i are computed the parameters (W_{id}) are modified using a Δ-rule.

$$\Delta W_{id} = \epsilon\, g_i(\boldsymbol{y}, W)\, y_d \quad \text{and} \quad W_{id}^{\text{new}} = C \frac{W'_{id} + \Delta W_{id}}{\sum_{d'} (W'_{id'} + \Delta W_{id'})}, \tag{1}$$

where W'_{id} denote the old values. Divisive normalization (right-hand-side equation) prevents the weights from growing infinitely and represents the most commonly used constraint for learning in artificial neural networks. The learning rule (1) can directly be associated with Hebbian learning between neural units g_1 to g_H and input units y_1 to y_D. The dotted horizontal arrows in Fig. 1 symbolize lateral information transfer that is generally required to compute the activities $g_i(\boldsymbol{y}, W)$. Neural networks that fit into the framework of Fig. 1 and use a learning rule like (1) or similar are frequently found in the literature and their application domains typically lie in the fields of clustering and component extraction [2,3,4,5,6]. Especially since generative models are used to study unsupervised learning, the class of neural network models that perform clustering could be embedded into a probabilistic framework. Learning in these networks can be regarded as optimization of the data likelihood under a generative model. This relationship in turn determines the function $g_i(\boldsymbol{y}, W)$ in (1) which has to be used for optimal learning. For clustering tasks the related generative models are mixture models and the function $g_i(\boldsymbol{y}, W)$ is the softmax operation, see e.g. [1,2]. A limit case of the softmax is a winner-take-all mechanisms which is likewise frequently applied to clustering tasks [2]. For networks such as [3,4] that extract input components, the situation is different. Linear systems such as independent (ICA) or principal component analysis (PCA) do not seem to represent the right correlates. ICA and PCA have been shown to fail in non-linear tasks, see [7], in which neural network models consistent with the above framework, e.g. [3,4,5], succeed in extracting the elementary input components. The learning dynamics used in these networks are manifold and comparison with other component extraction systems is usually only possible on the basis of benchmark tests. A probabilistic understanding similar to that for mixture models has not yet been developed. Furthermore, some network models perform clustering if data is generated by single causes, and component extraction if input consists of combinations of components, e.g. [3,4]. From the generative point of view this seems contradictory.

For a probabilistic analysis of networks (1), a generative model will be defined whose parameter update rules for likelihood maximization can be related to the delta rule in

(1). Before we define a generative model in the next section let us express the on-line update rule (1) in terms of a batch mode update for a set of N input patterns $\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(N)}$. By repeatedly applying (1) we get for slowly changing W:

$$W_{id} \approx \frac{W'_{id} + \epsilon \sum_n g_i(W, \boldsymbol{y}^{(n)})\, y_d^{(n)}}{\frac{1}{C} \sum_{d'} \left(W'_{id'} + \epsilon \sum_n g_i(W, \boldsymbol{y}^{(n)})\, y_{d'}^{(n)} \right)} \approx C \frac{\sum_n g_i(W, \boldsymbol{y}^{(n)})\, y_d^{(n)}}{\sum_{d'} \sum_n g_i(W, \boldsymbol{y}^{(n)})\, y_{d'}^{(n)}} . \quad (2)$$

The smaller the change of W the better the Δ-rule (1) is approximating (2).

3 A Non-linear Generative Model

Consider a generative model with D input units $y_1, \ldots, y_D$ and H hidden units $s_1, \ldots, s_h$. The model is parameterized by $\Theta = (\pi, W)$. The scalar π parameterizes the distributions of the hidden units which are assumed to be independent and Bernoulli distributed:

$$p(\boldsymbol{s} \,|\, \pi) = \prod_h p(s_h \,|\, \pi), \text{ where } p(s_h \,|\, \pi) = \pi^{s_h} (1 - \pi)^{1 - s_h} . \quad (3)$$

For given $\boldsymbol{s}$ we take the activities of the input units to be independent. The parameters W determine the influence of a particular $\boldsymbol{s}$ on the distributions of the input units. The values of y_d we take to be independent given $\boldsymbol{s}$ and distributed according to a Poisson distributions:

$$p(\boldsymbol{y} \,|\, \boldsymbol{s}, W) = \prod_d p(y_d \,|\, \overline{W}_d(\boldsymbol{s}, W)), \text{ where } p(y_d \,|\, w) = e^{-w} \frac{w^{y_d}}{y_d!} . \quad (4)$$

The Poisson distribution is a natural choice for non-negative and noisy input. E.g., if we take the causes to be objects with gray values W_{id}, all input patterns are non-negative. Furthermore, in the context of neural networks and Hebbian learning (1), a representation of non-negative values is straightforward. For the function $\overline{W}_d$ in (4) the most common choice is a linear superposition of the weights multiplied by the activities of the hidden units, $\overline{W}_d(\boldsymbol{s}, W) = \sum_h s_h W_{hd}$. Algebraically, this kind of superposition is advantageous for the computation of derivatives of $p(\boldsymbol{y} \,|\, \boldsymbol{s}, W)$ as required for optimization methods. However, for many applications linear superposition is difficult to motivate. E.g., for visual input it does not seem natural to assume linearity because occlusion is not modeled appropriately. For instance, if the causes are objects with the same gray value, linear superposition results in an error in the region where the objects overlap. Alternatively, the maximum function would represent the right model in this case and seems to be a better approximation for occlusion-like scenarios in general. Neural network models that fit into the framework (1) have been shown to be competitive in occlusion-like scenarios [3,4,5]. For our purposes the max-operation therefore seems to be preferable to linear superposition and we use as function $\overline{W}_d$ in (4):

$$\overline{W}_d(\boldsymbol{s}, W) := \max_h \{ s_h W_{hd} \} . \quad (5)$$

Using the equality $\overline{W}_d(\boldsymbol{s},W) = \lim_{\rho\to\infty} \overline{W}^{\rho}_d(\boldsymbol{s},W) = \lim_{\rho\to\infty} (\sum_h (s_h W_{hd})^{\rho})^{\frac{1}{\rho}}$ we define:

$$\mathcal{A}_{id}(\boldsymbol{s},W) := \lim_{\rho\to\infty} \left(\frac{\partial}{\partial W_{id}} \overline{W}^{\rho}_d(\boldsymbol{s},W)\right) = \lim_{\rho\to\infty} \frac{s_i\,(W_{id})^{\rho}}{\sum_h s_h (W_{hd})^{\rho}}\,, \tag{6}$$

$$\text{which implies: } \mathcal{A}_{id}(\boldsymbol{s},W)\, f(\overline{W}_d(\boldsymbol{s},W)) = \mathcal{A}_{id}(\boldsymbol{s},W)\, f(W_{id})\,. \tag{7}$$

f can be any well-behaved function. Equation (7) holds because both sides are zero whenever $\overline{W}_d(\boldsymbol{s},W) \neq W_{id}$.

4 Maximum Likelihood

Given a set of data $Y = \{\boldsymbol{y}^{(n)}\}_{n=1,\ldots,N}$ we want to find the set of parameters W that maximizes the likelihood of the data under the above generative model. Finding the optimal π is straightforward but for the purposes of this paper we will concentrate on the difficult part of optimizing the parameters W. Following the EM procedure, this amounts to maximizing the lower bound $Q(W,W')$:

$$Q(W,W') = \textstyle\sum_{n=1}^{N} \sum_{\boldsymbol{s}} \, p(\boldsymbol{s}\,|\,\boldsymbol{y}^{(n)},\Theta')\, \log\left(p(\boldsymbol{y}^{(n)}\,|\,\boldsymbol{s},W)\, p(\boldsymbol{s}\,|\,W)\right)\,, \tag{8}$$

where Θ' are the old parameters and where $\sum_{\boldsymbol{s}}$ is the sum over all binary vectors of length H. Using Bayes' rule the posterior probabilities are given by:

$$p(\boldsymbol{s}\,|\,\boldsymbol{y}^{(n)},\Theta') = \frac{p(\boldsymbol{s}\,|\,\pi)\, p(\boldsymbol{y}^{(n)}\,|\,\boldsymbol{s},W')}{\sum_{\tilde{\boldsymbol{s}}} p(\tilde{\boldsymbol{s}}\,|\,\pi)\, p(\boldsymbol{y}^{(n)}\,|\,\tilde{\boldsymbol{s}},W')} \quad \text{(E-step)}\,, \tag{9}$$

where $p(\boldsymbol{s}\,|\,\pi)$ and $p(\boldsymbol{y}^{(n)}\,|\,\boldsymbol{s},W')$ are given by (3) and (4), respectively. To account for divisive normalization we use constraint optimization of $Q(W)$:

$$\tfrac{\partial}{\partial W_{id'}}\, Q(W,W') \;+\; \textstyle\sum_{h=1}^{H} \lambda_h \, \tfrac{\partial}{\partial W_{id'}}\, G_h(W) \overset{!}{=} 0\,, \tag{10}$$

where $G_h(W) := \sum_{d=1}^{D} W_{hd} - C = 0$ is a constraint that keeps the sum over the weights for each hidden unit h constant. If we insert (8) with (4) into (10), we obtain by using (6) and (7):

$$\sum_n \sum_{\boldsymbol{s}} \, p(\boldsymbol{s}\,|\,\boldsymbol{y}^{(n)},\Theta')\, \mathcal{A}_{id'}(\boldsymbol{s},W)\, \frac{y^{(n)}_{d'} - W_{id'}}{W_{id'}} + \lambda_i = 0\,. \tag{11}$$

Equations (11) can be solved for the Lagrange multipliers by summing over d' and by using $\sum_d W_{hd} = C$. Applying some straightforward algebra, we obtain a set of non-linear equations for the parameter update:

$$W_{id} = \frac{\displaystyle\sum_n \langle \mathcal{A}_{id}(\boldsymbol{s},W)\rangle\, y^{(n)}_d}{\displaystyle\frac{1}{C}\sum_{n,d'} \langle \mathcal{A}_{id}(\boldsymbol{s},W)\rangle\, y^{(n)}_{d'} - \sum_n \left((\sum_{d'} \langle \mathcal{A}_{id'}(\boldsymbol{s},W)\rangle\, \frac{W_{id'}}{C}) - \langle \mathcal{A}_{id}(\boldsymbol{s},W)\rangle \right)}\,, \tag{12}$$

$$\text{where } \langle \mathcal{A}_{id}(\boldsymbol{s},W)\rangle = \textstyle\sum_{\boldsymbol{s}} \, p(\boldsymbol{s}\,|\,\boldsymbol{y}^{(n)},\Theta')\, \mathcal{A}_{id}(\boldsymbol{s},W). \tag{13}$$

Equation (12) can be simplified if we assume that the hidden causes are relatively homogeneously covering the input space. More precisely, if we parameterize the actually generating causes by W^{gen}, let us assume that all $W^{\text{gen}}_{id} > 0$ of cause i can be covered by the same number of $W^{\text{gen}}_{cd} \geq W^{\text{gen}}_{id}$:

$$W^{\text{gen}}_{id} > 0 \Rightarrow \textstyle\sum_{c \neq i} \mathcal{H}(W^{\text{gen}}_{cd} - W^{\text{gen}}_{id}) \approx b_i \,, \tag{14}$$

where $\mathcal{H}$ is the Heaviside function. b_i is the number of causes that can cover cause i. Fig. 2C,D illustrate this condition. For both examples condition (14) is fulfilled. E.g. for Fig. 2D $b_i = 0$ for all horizontal causes and $b_i = 1$ for the vertically oriented cause. If the model parameters W satisfy condition (14), which can be expected at least close to the maximum likelihood solution, we obtain with (12) and after rearranging terms:

$$W_{id} = C \, \frac{\sum_n \langle \mathcal{A}_{id}(\boldsymbol{s}, W) \rangle \; y^{(n)}_d}{\sum_{d'} \sum_n \langle \mathcal{A}_{id'}(\boldsymbol{s}, W) \rangle \; y^{(n)}_{d'}} \quad \text{(M-step)}. \tag{15}$$

Equation (15) can be used as a fixed point equation. Inserting the old values of W on the right-hand-side to obtain new values of W, approaches the right solution in numerical simulations.

5 E-Step Approximation

The M-step (15) together with expectations (13) represent an optimization algorithm for the generative model (3) to (5). However, for practical applications we are still confronted with the problem of an exponential cost for computing the sufficient statistics (13). For our model let us assume that most inputs have been generated by just few hidden causes. To approximate for sparse input we first group terms in (13) according to the number of active hidden units:

$$\langle \mathcal{A}_{id}(\boldsymbol{s}, W) \rangle \; = \; \sum_{\boldsymbol{s}} p(\boldsymbol{s} \,|\, \boldsymbol{y}^{(n)}, \Theta') \, \mathcal{A}_{id}(\boldsymbol{s}, W) \tag{16}$$

$$= \sum_{h} p(\boldsymbol{s}_h \,|\, \boldsymbol{y}^{(n)}, \Theta') \, \mathcal{A}_{id}(\boldsymbol{s}_h, W) + \sum_{\substack{a,\, b \\ a < b}} p(\boldsymbol{s}_{ab} \,|\, \boldsymbol{y}^{(n)}, \Theta') \, \mathcal{A}_{id}(\boldsymbol{s}_{ab}, W) + \sum_{\substack{a,\, b,\, c \\ a < b < c}} \ldots \,, \tag{17}$$

where $\boldsymbol{s}_h := (0, \ldots, 0, 1, 0, \ldots, 0)$ 1 at h's position,
$\boldsymbol{s}_{ab} := (0, \ldots, 0, 1, 0, \ldots, 0, 1, 0, \ldots, 0)$ 1 at a's and b's position $a \neq b$,

and analogously for $\boldsymbol{s}_{abc}$ etc. Note that $\mathcal{A}_{id}(\boldsymbol{0}, W) = 0$ because of (6). Each of the posterior probabilities $p(\boldsymbol{s} \,|\, \boldsymbol{y}^{(n)}, \Theta')$ above implicitly contains a similar sum over $\boldsymbol{s}$ for normalization whose terms may be grouped in the same way:

$$p(\boldsymbol{s} \,|\, \boldsymbol{y}^{(n)}, \Theta') \; = \; \frac{p(\boldsymbol{s}, \boldsymbol{y}^{(n)} \,|\, \Theta')}{\sum_{a} p(\boldsymbol{s}_a, \boldsymbol{y}^{(n)} \,|\, \Theta') + \sum_{\substack{a,\, b \\ a < b}} p(\boldsymbol{s}_{ab}, \boldsymbol{y}^{(n)} \,|\, \Theta') + \sum_{\substack{a,\, b,\, c \\ a < b < c}} \ldots} \,, \tag{18}$$

where we have used that fact that the update rule (15) remains unchanged if we remove all inputs equal to zero, i.e., all inputs with $\sum_d y_d^{(n)} = 0$. Combining (16) and (18) yields:

$$\langle \mathcal{A}_{id}(\boldsymbol{s}, W) \rangle = \qquad (19)$$
$$\frac{\sum_a p(\boldsymbol{s}_a, \boldsymbol{y}^{(n)} \,|\, \Theta')\, \mathcal{A}_{id}(\boldsymbol{s}_a, W) + \sum_{\substack{a,b \\ a<b}} p(\boldsymbol{s}_{ab}, \boldsymbol{y}^{(n)} \,|\, \Theta')\, \mathcal{A}_{id}(\boldsymbol{s}_{ab}, W) + \ldots}{\sum_a p(\boldsymbol{s}_a, \boldsymbol{y}^{(n)} \,|\, \Theta') + \sum_{\substack{a,b \\ a<b}} p(\boldsymbol{s}_{ab}, \boldsymbol{y}^{(n)} \,|\, \Theta') + \ldots}.$$

If we assume that the significant posterior probability mass will concentrate on vectors, $\boldsymbol{s}$, with only a limited number of non-zero entries, the expanded sums in both numerator and denominator of (19) may be truncated without significant loss. We will use an approximation of $\langle \mathcal{A}_{id}(\boldsymbol{s}, W) \rangle$ that corresponds to a truncation of (19) after terms of order two in the denominator and first order terms in the numerator. As numerical simulations show, this is the approximation of lowest possible order which still results in a system that can be used for component extraction. After inserting (3) to (5) into (19) and by using $\mathcal{A}_{id}(\boldsymbol{s}_h, W) = \delta_{ih}$ the approximation reads:

$$\langle \mathcal{A}_{id}(\boldsymbol{s}, W) \rangle \approx \frac{\exp(I_i^{(n)})}{\sum_h \exp(I_h^{(n)}) + \frac{\overline{\pi}}{2} \sum_{\substack{a,\,b \\ a \neq b}} \exp(I_{ab}^{(n)})}, \quad \overline{\pi} := \left(\frac{\pi}{1-\pi}\right)^{\beta}, \qquad (20)$$

$$I_i^{(n)} = \beta \sum_d \Big(\log(W_{id})\, y_d^{(n)} - W_{id}\Big), \;\; I_{ab}^{(n)} = \beta \sum_d \Big(\log(\tilde{W}_d^{ab})\, y_d^{(n)} - \tilde{W}_d^{ab}\Big) \qquad (21)$$

where $\tilde{W}_d^{ab} = \max(W_{ad}, W_{bd})$.The parameter β has been introduced to implement an annealing procedure [8] in later simulations. The M-step (15) together with the approximate E-step (20) is a learning algorithm that we will refer to as *approximate EM*.

If we now compare (15) with (2), we find that a neural network (1) can be used for parameter optimization by choosing $g_i(W, \boldsymbol{y}^{(n)}) = \langle \mathcal{A}_{id}(\boldsymbol{s}, W) \rangle$. Note that in general g_i would have to depend on d. However, if we use approximation (20), $\langle \mathcal{A}_{id}(\boldsymbol{s}, W) \rangle$ only depends on i. For the neural network this implies that the change of the weight W_{id} only depends on the activities of the pre- and post-synaptic units:

$$\Delta W_{id} = \epsilon\, g_i\, y_d \quad \text{with} \quad g_i = \frac{\exp(I_i)}{\sum_h \exp(I_h) + \frac{\overline{\pi}}{2} \sum_{\substack{a,\,b \\ a \neq b}} \exp(I_{ab})} \qquad (22)$$

and divisive normalization ($\sum_d W_{id} = C$). Note that the function g_i in (22) is similar to the classical softmax [2] but has an additional term in the denominator. For very sparse input, i.e. $\overline{\pi} \ll 1$, the classical softmax is recovered.

The intuition for the learning rule is as follows: The system tries to explain a given input pattern using its current state of the model parameters W. If one hidden unit explains the input better than any combination of two units, this unit is modified. If the input is better explained by a combination of two units, learning is penalized. Eqns. (22) describe an easy and computationally efficient method to maximize the data likelihood under the generative model (3) to (5).

6 Simulations

It remains to be shown that the approximations we have introduced result in parameter updates that indeed approximately maximize the data likelihood. That the likelihood is always increased is only guaranteed if we use the exact M-step (12) and the exact sufficient statistics (13). Although our approximations are derived from the exact M- and E-steps there is no guarantee that they always result in an increase of the likelihood. Furthermore, even in the case of exact EM, learning can converge to local optima. To

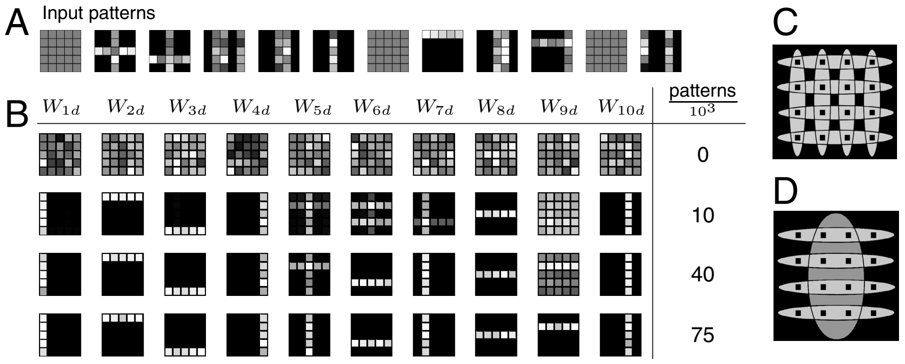
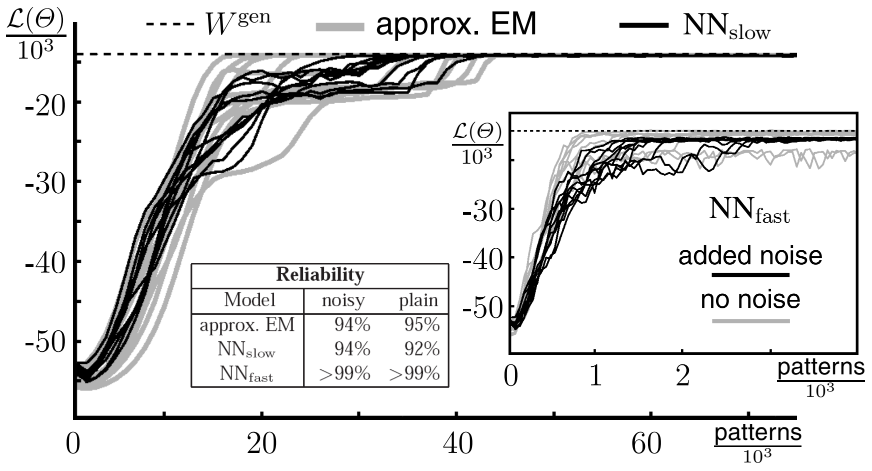


Fig. 2. Bars test with 5x5 pixels, 10 bars, and $\pi = \frac{2}{10}$. **A** 12 input patterns from the set of $N = 500$ input patterns used. **B** Update of the parameters W using the neural network NN_{slow}. **C** Sketch of the hidden causes of a bars test with 8 bars. The black squares symbolize the pixels. The bars test satisfies condition (14). **D** Exemplarily another distribution of hidden causes that satisfies the condition.

verify that the derived approximate EM and neural networks with Δ-rule (22) are indeed able to extract hidden causes, we use the bars benchmark test for component extraction [3,4,5,6]. In the bars test a bar appears with fixed probability at one of, e.g., 10 positions. To test the algorithms we use generative parameters W^{gen} in the form of $H = 10$ horizontal and vertical bars. Values W^{gen}_{id} that correspond to a bar are set to $W^{\text{gen}}_{id} = 10$, values that correspond to the background are zero, $W^{\text{gen}}_{id} = 0$. We generate $N = 500$ input patterns according to the generative model (3) to (5) with $\pi = \frac{2}{10}$. A subset of the resulting input patterns is displayed in Fig. 2A. The Poisson distribution (4) results in noisy bars. In Fig. 2B the parameters W are displayed in the case of using a neural network (22) with $\epsilon = 0.1$, which we will refer to as NN_{slow}. As can be seen, the system's parameters W converge from randomly initialized values to values similar to those of the generating parameters W^{gen}. In Fig. 2B learning time is measured in terms of the number of patterns that are randomly drawn from a set of $N = 500$. For the same bars test settings, the development of the data likelihood is plotted in Fig. 3 for ten trials with NN_{slow}(back lines). Note that for the likelihood plots we have used the same set of $N = 500$ input patterns for all trials. In all other simulations we use a newly generated set of 500 patterns for each trial. In Fig. 3 it can be observed that the network increases the likelihood up to a value that closely corresponds to the likelihood of the generating weights W^{gen}. The likelihoods of ten trials using approximate EM, (15) and (20), are



Reliability		
Model	noisy	plain
approx. EM	94%	95%
NN_{slow}	94%	92%
NN_{fast}	>99%	>99%

Fig. 3. Development of log-likelihood values during learning for the same bars test as in Fig. 2. Time courses for approximate EM (broad grey lines) and NN_{slow} (black lines) approach the likelhood value of the actually generating weights W^{gen} (dashed line). Note that one EM iteration corresponds to $N = 500$ patterns. The small plot shows log-likelihood values for NN_{fast} (black). Likelihoods for NN_{fast} without Gaussian noise on the weights is plotted for comparison (gray). The table shows the probability of each system to extract all bars in a given trial, i.e., its reliability. Values are given for bars with Poisson noise and without noise.

plotted for comparison (broad gray lines). Note that EM and NN_{slow} essentially behave in the same way as could be expected on the basis of our analytical results. NN_{slow} updates W after every pattern presentation whereas EM uses all $N = 500$ patterns for each update. In Fig. 3, $N = 500$ patterns therefore correspond to one EM iteration. For NN_{slow} the online update introduces stochasticity. This effects learning only marginally, however. Compared to approx. EM, the likelihoods are slightly noisier and convergence times seem to vary less.

For learning we use an annealing proceedure, i.e., we use $\beta < 1$ in (21) and increase β to one in the course of learning. For EM and NN_{slow} we use the same cooling schedule, which remains unchanged throughout this paper. We increase β in (21) by decreasing a 'temperature' T using $\beta = \frac{1}{T}$. We use $T_1 = 18$ for the first 50 iterations and linearly cool to $T = 1$ in the subsequent 100 iterations[1]. For later comparison we introduce another neural network version based on (22). Instead of a cooling schedule we use a fixed temperature $T = 16$ and add Gaussian noise for each weight at each update: $\Delta W_{id} = \epsilon\, g_i\, y_d + \sigma\, \eta$. Using $\epsilon = 1.0$ and $\sigma = 0.02$ we obtain a neural network that requires relatively few pattern presentations for learning ($< 1,000$). We therefore refer to the network as NN_{fast}. Ten time-courses of likelihood values are shown for NN_{fast} in Fig. 3 (small plot, black lines). Because of the additional noise the final likelihood values are somewhat lower than those of the generating weights. If a network with the same

[1] For NN_{slow} this translates to: 25×10^3 pattern presentations with $T_1 = 18$ and linear cooling to $T = 1$ within the next 50×10^3 patterns.

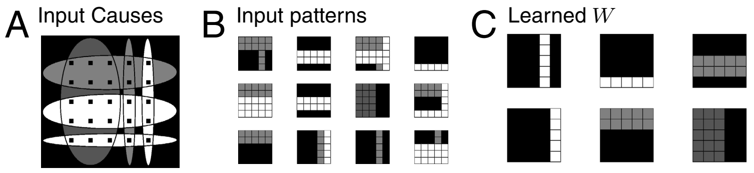


Fig. 4. Bars test with different bar intesities and widths. **A** Scematic visualization of the six bars. **B** 12 examples of the $N = 500$ input patterns used. **C** Learned parameters W using $\mathrm{NN}_{\mathrm{slow}}$ (from black for $W_{id} = 0$ to white for $W_{id} = 10$). Note that $\sum_d W_{id} = 50$ for all weights.

parameters as $\mathrm{NN}_{\mathrm{fast}}$ but without added noise ($\sigma = 0$) is used, likelihood values are higher (small plot, gray lines) but the system converges to local optima relatively often.

For comparison with other systems in the literature we will use the same bars test as above but instead of bars with Poisson noise we use non-noisy bars. In the bars test, a very prominent criterion for comparison is the so called *reliablity* of the system [3,4,5,6,9,10]. It is defined as the probability to find all bars in a given trial. If a system tends to converge to local optima, its reliability is low. The table in Fig. 3 summarizes the reliability values for approx. EM, $\mathrm{NN}_{\mathrm{slow}}$, and $\mathrm{NN}_{\mathrm{fast}}$ for bars with and without Poisson noise. Each reliability value was computed on the basis of 100 trials and each trail starts with a different randomly initialized W. For both bars test versions the same algorithms and the same cooling schedule were used (the same fixed T in the case of $\mathrm{NN}_{\mathrm{fast}}$). Using the criterion of reliability, $\mathrm{NN}_{\mathrm{fast}}$ shows the best performance. The reason is the additional Gaussian noise and noise introduced by updating online. The combination of both sources of noise drives the system out of shallow optima. Furthermore, $\mathrm{NN}_{\mathrm{fast}}$ is the fastest system in terms of required pattern presentations. It needs less than $1,000$ patterns to find all bars in the majority of 100 simulations[2]. Note, however, that in some trials learning time is much longer. In terms of likelihood values, approx. EM and $\mathrm{NN}_{\mathrm{slow}}$ are to be preferred. These system also have the advantage of a well defined stopping criterion.

Regarding values of reliability, the systems' performance is very robust if their parameter π in (20) or (22) is chosen very different from the actually generating parameter $\pi^{\mathrm{gen}} = \frac{2}{10}$ (note that it was not neccessary to distinguished between π and π^{gen}, so far). In the beginning of learning the small exponent $\beta \approx 0.05$ in (20) results in $\overline{\pi}$ values close to one for a large range of different π. Close to the end of the cooling schedule, for $\beta \approx 1$, different π values do likewise only marginally effect learning because the second term in the denominator of (20) is either very small or very large compared to the first. In contrast, the likelihoods are relatively sensitive to different values of π. Using an EM approach similar to the one for W, the value for π can, however, be infered.

Finally, the reliabilities of the algorithms are found to be robust w.r.t. violations of assumptions (14) and $\sum_d W_{id}^{\mathrm{gen}} = C$ for the generating weights. An example of a bars test that violates both assumptions is shown in Fig. 4. $\mathrm{NN}_{\mathrm{slow}}$ finds all bars with 96% reliability in this case.

[2] The system is taken to have found all bars if each hidden unit represents a different bar. This is usually already the case for intermediate likelihood values.

7 Discussion

Generative model and neural network approaches have both been suggested for the extraction of non-linear components. In this paper we have analytically shown that Hebbian learning in feed-forward neural networks can correspond to parameter updates in a generative model using approximate EM. The distinguishing feature of this generative model is an explicit non-linearity in the form of a $\max$ operation.The appropriate activation rule for the networks turns out to be the generalized softmax rule (20). The generalization drops back to the classical softmax for very sparse input.

The analytical results were verified in numerical simulations using the bars benchmark test. We found that the derived neural networks increase the likelihood under the generative model and avoid local optima in the great majority of trials. The derived learning algorithm $\mathrm{NN}_{\mathrm{fast}}$ performs best if learning time and system reliability are used for comparison. Using a high learning rate and additional noise the network model $\mathrm{NN}_{\mathrm{fast}}$ avoids all local optima and needs few pattern presentations for learning. The same is reported, e.g., for the networks [3,4] which both fit into the framework considered here (Eqn. 1 and Fig. 1). Furthermore, the generalized softmax (20) offers an explanation of why networks such as [3] and [4] can also be successfully applied to clustering tasks. In contrast to many neural networks, algorithms that are based on generative models are by definition well interpretable probabilistically. They have, however, been criticized [3,4,7] for frequently failing to extract the true causes. E.g., for the same bars test as used here, the models [9] and [10] find all bars in just 27% and 69% (even though bar overlap is excluded) of trials, respectively. Instead of treating them as separate or competing approaches, we have in this paper shown that generative and neural network approaches can come together in the form of competitive learning algorithms that are neurally plausible, reliable, and probabilistically fully interpretable.

Acknowledgment. This work was funded by the Gatsby Charitable Foundation.

References

1. McLachlan, G., Peel, D.: Finite Mixture Models. Wiley, Chichester (2000)
2. Yuille, A.L., Geiger, D.: Winner-take-all networks. In: Arbib, M.A. (ed.) The handbook of brain theory and neural networks, pp. 1228–1231. MIT Press, Cambridge (2003)
3. Spratling, M.W., Johnson, M.H.: Preintegration lateral inhibition enhances unsupervised learning. Neural Computation 14, 2157–2179 (2002)
4. Lücke, J., von der Malsburg, C.: Rapid processing and unsupervised learning in a model of the cortical macrocolumn. Neural Computation 16, 501–533 (2004)
5. Spratling, M.W.: Learning image components for object recognition. Journal of Machine Learning Research 7, 793–815 (2006)
6. O'Reilly, R.C.: Generalization in interactive networks: The benefits of inhibitory competition and Hebbian learning. Neural Computation 13, 1199–1241 (2001)

7. Hochreiter, S., Schmidhuber, J.: Feature extraction through LOCOCODE. Neural Computation 11, 679–714 (1999)
8. Ueda, N., Nakano, R.: Deterministic annealing EM algorithm. Neural Networks 11(2), 271–282 (1998)
9. Saund, E.: A multiple cause mixture model for unsupervised learning. Neural Computation 7, 51–71 (1995)
10. Dayan, P., Zemel, R.S.: Competition and multiple cause models. Neural Computation 7, 565–579 (1995)