Turbo Factor Analysis

Brendan J. Frey
Computer Science, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada.

A few years ago, Pearl's probability propagation algorithm in graphs with cycles was shown to produce excellent results for error-correcting "turbo-decoding". Ever since, we have wondered whether iterative probability propagation could be used successfully for machine learning. As a first step in this direction, we study iterative inference and learning in the simple factor analyzer network – a two-layer densely connected network that models bottom layer sensory inputs as a linear combination of top layer factors plus independent Gaussian sensor noise. The number of bottom-up/top-down iterations needed to exactly infer the factors given a network and an input scales with the number of factors in the top layer. In online learning, this iterative procedure must be reinitialized upon each pattern presentation and so learning becomes prohibitively slow in big networks, such as those used for face recognition and for large-scale models of the cortex. We show that probability propagation in a factor analyzer usually takes just a few iterations to achieve a low inference error, even in networks with 320 sensors and 80 factors. We derive an expression for the algorithm's fixed point and provide an eigenvalue condition for global convergence. We also show how iterative inference can be used to do online learning and give results on using this method to do online dimensionality reduction for the purpose of recognizing 560-pixel images of faces using a 40-dimensional subspace. This work suggests that iterative probability propagation in densely connected networks may lead to a broad class of useful algorithms for machine learning.

1 Encoding Data using Linearly Combined Components

A simple way to encode input patterns is to suppose that each input can be well-approximated by a linear combination of component vectors, where the amplitudes of the vectors are modulated to match the input. For a given training set, the most appropriate set of component vectors will depend on how we expect the modulation levels to behave and how we measure the distance between the input and its approximation. These effects can be captured by a generative probability model that specifies a distribution $p(z)$ over modulation levels $z = (z_1, \ldots, z_K)^T$ and a distribution $p(x|z)$ over sensors $x = (x_1, \ldots, x_N)^T$ given the modulation levels. The linear combination
Figure 1: A two layer generative network that describes principal component analysis, independent component analysis and factor analysis.

The constraint is given by

$$E[x|z] = \Lambda z,$$  \hspace{1cm} (1)

where $E[x|z] = \int_x x p(x|z)dx$ and each column of $\Lambda$ is a component vector. $\Lambda$ is an $N \times K$ matrix with elements $\lambda_{nk}$.

Principal component analysis (Jolliffe 1986; Linsker 1988; Oja 1989), independent component analysis (Comon, Jutten and Herault 1991; Bell and Sejnowski 1995; Amari, Cichocki and Yang 1996), and factor analysis (Rubin and Thayer 1982; Everitt 1984) can be viewed as maximum likelihood estimation in models of this type, where we assume that the appropriate modulation levels are independent and the overall distortion is given by the sum of the individual sensor distortions.

The two layer belief network that describes this process is shown in Fig. 1. The distribution over the top layer activities (modulation levels) is

$$p(z) = \prod_{k=1}^{K} p(z_k),$$  \hspace{1cm} (2)

and the distribution over the bottom layer activities (sensors) given the top layer activities is

$$p(x|z) = \prod_{n=1}^{N} p(x_n|z).$$  \hspace{1cm} (3)

If we choose probabilities of the form

$$p(z_k) = \mathcal{N}(z_k; 0, 1), \quad p(x_n|z) = \mathcal{N}(x_n; \sum_{k=1}^{K} \lambda_{nk} z_k, \psi),$$  \hspace{1cm} (4)

where the same variance $\psi$ is used for all sensors, then maximum likelihood estimation of $\Lambda$ and $\psi$ produces components that span the first $K$ principal
components of the training data (Roweis and Ghahramani 1999; Tipping and Bishop 1999).

If \( p(z_k) \) is a heavy-tailed distribution and no distortion is allowed (achieving this generally requires using a complete or over-complete basis, where \( K > N \)), maximum likelihood estimation of \( \Lambda \) is equivalent to independent component analysis (Pearlmutter and Parra 1997; MacKay 1997). For example, we may take

\[
p(z_k) \propto 1/(e^{z_k} + e^{-z_k}), \quad p(x_n \mid z) = \delta(x_n - \sum_{k=1}^{K} \lambda_{nk} z_k).
\]

The advantage of independent component analysis over principal component analysis and factor analysis is the use of heavy-tailed distributions over the modulation levels. These distributions effectively produce sparse representations of the data, since a heavy-tailed distribution places probability mass near zero and at large values (i.e., it has high kurtosis).

1.1 Factor Analysis. If both \( p(z_k) \) and \( p(x_n \mid z) \) are Gaussian and each sensor has its own noise variance,

\[
p(z_k) = \mathcal{N}(z_k; 0, 1), \quad p(x_n \mid z) = \mathcal{N}(x_n; \sum_{k=1}^{K} \lambda_{nk} z_k, \psi_n),
\]

then maximum likelihood estimation of \( \Lambda \) and \( \psi_n, n = 1, \ldots, N \) performs factor analysis. In factor analysis, the modulation levels are called “factors” and \( \Lambda \) is called the “factor loading matrix”.

In contrast to principal component analysis, factor analysis can model a different level of noise on each sensor. In contrast to independent component analysis, the noise model used in factor analysis permits a more compact latent representation, where \( K < N \).

The factor analysis model (factor analyzer) can be written

\[
p(z) = \mathcal{N}(z; 0, I), \quad p(x \mid z) = \mathcal{N}(x; \Lambda z, \Psi),
\]

where \( \Psi \) is a diagonal covariance matrix whose \( n \)th diagonal entry is \( \psi_n \). The marginal distribution over \( x \) is

\[
p(x) = \int_{z} \mathcal{N}(z; 0, I) \mathcal{N}(x; \Lambda z, \Psi) dz = \mathcal{N}(x; 0, \Lambda \Lambda^T + \Psi).
\]

Although the factor analyzer is not able to represent the rich, nonlinear, hierarchical processes that can be learned by nonlinear Gaussian belief networks (Hinton and Ghahramani 1997; Frey 1998; Frey and Hinton 1999; Attias 1999), the simplicity of the factor analyzer makes it a good starting point for studying iterative probability propagation.
2 Inferring the Factors in a Factor Analyzer

Since the factor analyzer is a Gaussian model, the distribution over the factors given the expected value of \( z \) and its covariance matrix, which are obtained by normalizing \( p(z)p(x|z) \) with respect to \( z \) using linear algebra:

\[
E[z|x] = (\Lambda^T \Psi^{-1} \Lambda + I)^{-1} \Lambda^T \Psi^{-1} x, \tag{9}
\]
\[
\text{COV}(z|x) = (\Lambda^T \Psi^{-1} \Lambda + I)^{-1}. \tag{10}
\]

Notice that the covariance matrix does not depend on the value of the input, although it does depend on an input being observed. (Recall that \( \text{COV}(z) = I \).)

In this paper, we explore methods that infer independent factors, so we focus on inferring the factor means and factor variances,

\[
\text{diag}(\text{COV}(z|x)) = \text{diag}((\Lambda^T \Psi^{-1} \Lambda + I)^{-1}). \tag{11}
\]

Whereas \( \text{COV}(z|x) \) contains \( K^2 \) elements derived from \( N \) sensors, \( E[z|x] \) contains just \( K \) elements derived from \( N \) sensors and \( \text{diag}(\text{COV}(z|x)) \) contains just \( K \) elements, so we might hope that the factor means and variances can be computed using order \( KN \) scalar operations in a small number of iterations.

Unfortunately, it turns out that for a given network and input, order \( K \) bottom-up/top-down iterations are needed to exactly infer the factor means and variances. A simple way to see this is by looking at the computation of \( \Lambda^T \Psi^{-1} \Lambda \) in (9) and (11), which generally requires order \( KN \) scalar operations to compute exactly. Each iteration in a network with \( KN \) connections performs order \( KN \) scalar operations, so the number of iterations is at least order \( K \). This number of iterations is prohibitively large in many practical applications and in large-scale models of the cortex.

Of course, multiplying \( x \) by \( (\Lambda^T \Psi^{-1} \Lambda + I)^{-1} \Lambda^T \Psi^{-1} \) takes just \( KN \) operations, so if a factor analyzer with fixed parameters is to be used for inference many times (say, more than \( K \) times, which is typical for batch processing), it is a good idea to precompute \( (\Lambda^T \Psi^{-1} \Lambda + I)^{-1} \Lambda^T \Psi^{-1} \) beforehand and reuse it for each input.

There are several situations where direct precomputation of \( (\Lambda^T \Psi^{-1} \Lambda + I)^{-1} \Lambda^T \Psi^{-1} \) is not so good an idea, including online learning, where the parameters change from case to case; inference in a biological neural network, which should use local computations in the network; and single-shot inference, where a network is given and the goal is to do inference for one pattern as fast as possible. In these cases, we need an algorithm that can infer the factor means and variances quickly for a given network and input.

Before describing the iterative probability propagation algorithm, which takes just a few bottom-up/top-down iterations to do inference, we review
a diagonal approximation to the inverse covariance matrix, recognition networks and “conjugate iterative inference”.

2.1 Diagonal Approximation to the Inverse Covariance Matrix. If we approximate the inverse covariance matrix $\Lambda^T\Psi^{-1}\Lambda + I$ with its diagonal, we need only compute $\text{diag}(\Lambda^T\Psi^{-1}\Lambda)$, which takes just $KN$ scalar operations. This diagonal approximation is equivalent to approximating $p(z|x)$ with a product-form distribution $q(z)$ that minimizes the Kullback-Leibler divergence, $\int q(z) \log q(z)/p(z|x)dz$. The approximate diagonal covariance matrix is

$$\text{COV}^\text{PF}(z|x) = (\text{diag}(\Lambda^T\Psi^{-1}\Lambda) + I)^{-1}. \tag{12}$$

The $k$th diagonal element of $\text{COV}^\text{PF}(z|x)$ is

$$\text{VAR}^\text{PF}(z_k|x) = 1/(1 + \sum_{m=1}^N \lambda_m^2 / \psi_m). \tag{13}$$

These variances can be computed in one bottom-up pass through the network in Fig. 1.

2.2 Recognition Networks. Helmholtz machines (Dayan et al. 1995; Hinton et al. 1995) use a “recognition network” to infer the factor means given the sensory input. In one bottom-up pass, the recognition network computes $\hat{z} = Ty$, where $Y$ is the $K \times N$ matrix of recognition weights. Exact inference of the factor means is given by the weights, $\bar{Y} = (\Lambda^T\Psi^{-1}\Lambda + I)^{-1}\Lambda^T\Psi^{-1}$.

Direct computation of $\bar{Y} = (\Lambda^T\Psi^{-1}\Lambda + I)^{-1}\Lambda^T\Psi^{-1}$ can be circumvented using the wake-sleep algorithm (Hinton et al. 1995), which simulates samples from the generative network in Fig. 1 and then adjusts the recognition weights using a local delta rule. Although inference using a recognition network takes just one iteration, the network must be learned, so recognition networks are not efficient at doing one-shot inference. Also, learning uses sample statistics, so a large number of iterations may be needed to average out noise (Neal and Dayan 1997). Nonetheless, the wake-sleep algorithm in a Helmholtz machine is very attractive as a biological model, since it uses simple, local computations in both a feedforward network and a feedback network.

2.3 Conjugate Iterative Inference. One approach to computing $\text{E}[z|x]$ is to repeatedly update an approximation, while ensuring that each successive update does not spoil the improvements achieved by previous updates. This is the idea used in the conjugate gradient optimization method (Fletcher 1987) and this approach can also be viewed as a form of variational inference (Jordan et al. 1998) that uses a product-form Gaussian approximation of $p(z|x)$. Inferring the factor means is equivalent to maximizing
\( p(z)p(x|z) \) with respect to \( z \), since the posterior is proportional to the joint distribution.

Using the negative of the exponent in \( p(z)p(x|z) \) as a cost function, the estimates \( z_k^{(s)}, k = 1, \ldots, K \) of the factor means at step \( s \) give a cost

\[
C^{(s)} = \frac{1}{2} \sum_{k=1}^{K} (z_k^{(s)})^2 + \frac{1}{2} \sum_{n=1}^{N} (x_n - \sum_{k=1}^{K} \lambda_{nk} z_k^{(s)})^2 / \psi_n. \tag{14}
\]

Each step of conjugate gradient optimization computes the current gradient of the cost function (one scalar for each factor), uses this gradient to compute a new search direction (one scalar for each factor) and finds the minimum of the cost along the new search direction. For a quadratic function in \( K \) variables, this procedure finds the minimum (exact values of the factor means) in \( K \) steps, or \( 2K \) top-down/bottom-up iterations.

For the current estimates of the factor means, the error at the input can be computed in a top-down pass:

\[
e_n^{(s)} = x_n - \sum_{k=1}^{K} \lambda_{nk} z_k^{(s)}. \tag{15}\]

The gradient for the \( k \)th top layer unit is computed in a subsequent bottom-up pass:

\[
\delta_k^{(s+1)} = \frac{\partial C^{(s)}}{\partial z_k^{(s)}} = z_k^{(s)} - \sum_{n=1}^{N} \lambda_{nk} e_n^{(s)} / \psi_n. \tag{16}\]

The search direction for the \( k \)th top layer unit used to compute the new estimates of the factor means is

\[
s_k^{(s+1)} = -\delta_k^{(s+1)} + \frac{\sum_{j=1}^{K} (\delta_j^{(s+1)})^2}{\sum_{j=1}^{K} (\delta_j^{(s)})^2} \delta_k^{(s)}. \tag{17}\]

This computation is not local in the graph in Fig. 1, but it could be implemented using a network connecting the top layer units. Also, both the current gradient and the gradient at the previous iteration need to be stored.

The new estimate for the \( k \)th factor is

\[
z_k^{(s+1)} = z_k^{(s)} - \frac{\sum_{j=1}^{K} (\delta_j^{(s+1)}) \delta_j^{(s+1)}}{\sum_{j=1}^{K} (\delta_j^{(s+1)})^2 + \sum_{j=1}^{K} (\delta_j^{(s+1)})^2 \sum_{n=1}^{N} \lambda_{nj}} \lambda_{nk} \delta_k^{(s+1)}/ \psi_n \tag{18}\]

Again, this computation requires communication between the top layer units.

Initially, we take \( \delta_k^{(0)} = 0, e_n^{(0)} = x_n, \delta_k^{(1)} = -\sum_{n=1}^{N} \lambda_{nk} e_n^{(0)}/ \psi_n \) and \( s_k^{(1)} = -\delta_k^{(0)} \) before applying (18) to compute \( z_k^{(1)} \).

The last pair of summations in the denominator of (18) require an additional top-down/bottom-up pass. So, each step of conjugate iterative inference takes two top-down/bottom-up iterations in the network. Approximate inference can be performed in fewer than \( 2K \) iterations simply by stopping the above procedure after a small number of iterations.
3 Iterative Probability Propagation – A Fast, Local Method

For networks that are trees, the probability propagation algorithm can be used to do exact inference. This algorithm was used extensively in the 20th century (Gallager 1963; Tanner 1981; Pearl 1988; Lauritzen and Spiegelhalter 1988), but dates back to the 19th century, e.g., in work by T. N. Thiele in 1880 (Steffen Lauritzen, personal communication). The probability propagation algorithm uses simple, local computations to produce numbers that are passed on the edges in a network and then combines these numbers locally to compute marginal distributions exactly.

Recent results on error-correcting coding show that in some cases probability propagation gives excellent performance even if the network contains so many cycles that exact inference is exponentially difficult (MacKay and Neal 1996; Frey and Kschischang 1996; Frey 1998; Frey and MacKay 1998a; Kschischang and Frey 1998; McEliece, MacKay and Cheng 1998). Probability propagation for error-correction using “turbo-codes” (Berrou and Glavieux 1996) and low-density parity-check codes (MacKay 1999) is now widely considered to be a major breakthrough in the information theory community.

When the network contains cycles, the local computations give rise to an iterative algorithm, which hopefully converges to a good answer. Despite excellent performance in error-correcting coding applications, little is known about the convergence properties of iterative probability propagation. Probability propagation in networks containing a single cycle has been successfully analyzed by Weiss (1999) and Smyth et al. (1997) and in the case of so-called “tail-biting trellises” by Forney et al. (1998) and Aji et al. (1998). However, results for networks containing many cycles are much less revealing (Wiberg 1996; Richardson 1998; Frey, Koetter and Vardy 1998; MacKay 1999).

We show that iterative probability propagation converges to the correct answer within a few iterations in random factor analysis networks and we derive an expression for the algorithm’s fixed point and provide an eigenvalue condition for global convergence. In Sec. 4, we show that this fast inference method can be used successfully in online learning and that “turbo factor analysis” can perform dimensionality reduction for face modeling and recognition.

3.1 Iterative Propagation of Means and Variances. The factor analyzer is a Gaussian model, so each probability message produced by probability propagation can be characterized by a mean and a variance. Each iteration of probability propagation in the factor analysis network shown in Fig. 1 consists of passing a mean and a variance along each edge in a bottom-

---

1In fact, Gallager (1963) invented the algorithm 3 decades earlier, but the computers available at the time were not powerful enough to manipulate the large amounts of data that are needed to show the algorithm works amazingly well.
up pass, followed by passing a mean and a variance along each edge in a top-down pass. At any instant, the bottom-up means and variances can be combined to form estimates of the means and variances of the factors given the input.

The prior distribution on each top layer unit is a standard normal distribution, so initially we set the variance and mean sent from the kth top layer unit to the nth sensor to $\nu_{kn}^{(0)} = 1$ and $\eta_{kn}^{(0)} = 0$.

The bottom-up pass begins by computing a noise level and an error signal at each sensor using the top-down variances and means from the previous iteration:

$$s_n^{(i)} = \psi_n + \sum_{k=1}^{K} \lambda_{nk}^2 \nu_{kn}^{(i-1)}$$  
$$e_n^{(i)} = x_n - \sum_{k=1}^{K} \lambda_{nk} \eta_{kn}^{(i-1)}.$$  

These are used to compute bottom-up variances and means as follows:

$$\phi_{nk}^{(i)} = s_n^{(i)} / \lambda_{nk}^2 - \nu_{kn}^{(i-1)},$$  
$$\mu_{nk}^{(i)} = e_n^{(i)} / \lambda_{nk} + \eta_{kn}^{(i-1)}.$$  

If $\lambda_{nk}$ is allowed to be 0 for some $n$ and $k$, $\lambda_{nk}^2 \phi_{nk}^{(i)}$ and $\lambda_{nk} \mu_{nk}^{(i)}$ should be propagated instead of $\phi_{nk}^{(i)}$ and $\mu_{nk}^{(i)}$. We will assume $\lambda_{nk} \neq 0$ in this paper.

The bottom-up variances and means are then combined to form the current estimates of the factor variances and means:

$$\nu_k^{(i)} = 1 / (1 + \sum_{n=1}^{N} 1/\phi_{nk}^{(i)}),$$  
$$\mu_k^{(i)} = \nu_k^{(i)} \sum_{n=1}^{N} \mu_{nk}^{(i)} / \phi_{nk}^{(i)}.$$  

The top-down pass proceeds by computing top-down variances and means as follows:

$$\nu_{kn}^{(i)} = 1 / (1/\nu_k^{(i)} - 1/\phi_{nk}^{(i)}),$$  
$$\eta_{kn}^{(i)} = \nu_{kn}^{(i)} (\mu_k^{(i)} / \nu_k^{(i)} - \mu_{nk}^{(i)} / \phi_{nk}^{(i)}).$$  

Notice that the variance updates are independent of the mean updates, whereas the mean updates depend on the variance updates.

### 3.2 Performance of Iterative Probability Propagation.

We created a total of 200,000 factor analysis networks with 20 different sizes ranging from $K = 5, N = 10$ to $K = 80, N = 320$ and for each size of network we measured the inference error as a function of the number of iterations of probability propagation.

Each of the 10,000 networks of a given size was produced by drawing the $\lambda_{nk}$s from independent standard normal distributions and then drawing each sensor variance $\psi_n$ from an exponential distribution with mean equal
to $\sum_{k=1}^{K} \lambda_{ik}^2$. This procedure ensured that the scales of the components varied significantly and that the sensor noise levels varied about a mean level that was equal to the variation due to the components. (A similar procedure was used in (Neal and Dayan 1997).)

For each random network, a pattern was simulated from the network and probability propagation was applied using the simulated pattern as input. We measured the error between the estimate $\hat{z}^{(o)}$ and the correct value $E[z|x]$ by computing the difference between their coding costs under the exact posterior distribution and then normalizing by $K$ to get an average number of nats per top layer unit:

$$error^{(o)} = -\log p(\hat{z}^{(o)}|x)/K - (\log p(E[z|x]|x))/K$$

$$= (\hat{z}^{(o)} - E[z|x])^T\text{COV}(z|x)^{-1}(\hat{z}^{(o)} - E[z|x])/2K.$$

This is equivalent to using a normalized Mahalanobis distance between $\hat{z}^{(o)}$ and $E[z|x]$.

Fig. 2 shows the inference error on a logarithmic scale versus the number of iterations (maximum of 20) of probability propagation in the 20 different network sizes. Each plot shows the median error (the bold curve), the range within which 98% of the errors lie (the two curves adjacent to the bold curve) and the error below which 99.9% of the errors lie (the fourth, topmost curve).

In all cases, the median error is reduced below .01 nats within 6 iterations.

The number of iterations needed to achieve a given inference error increases sublinearly with $K$. Comparing the median error for $K = 5$, $N = 320$ to the median error for $K = 80$, $N = 320$, we see that although $K$ increases by a factor of 16, the number of iterations needed to reduce the error to 0.01 nats increases by a factor of just 2.

The rate of convergence of the error appears to increase for larger $N$, as indicated by a general trend for the error curves to steepen when $N$ is increased.

For network sizes $K < N/4$, 99% of the networks converge to 1 nat within 5 iterations. For network sizes $K \geq N/4$, 99% of the networks converge to 1 nat within 10 iterations. For network sizes $K \geq N/8$, 0.1% of the networks actually diverge.

### 3.3 Convergent Variances and Divergent Means.

To better understand the divergent cases, we have plotted the means and variances as a function of the number of iterations in networks that were found to diverge. We call a network “divergent” if the inference error after 20 iterations is greater than the error after 10 iterations. 54 of the 10,000 networks with $K = 5$, $N = 10$ diverged, 69 of the 10,000 networks with $K = 10$, $N = 20$ diverged, and 70 of the 10,000 networks with $K = 20$, $N = 40$ diverged.
Figure 2: Performance of iterative probability propagation. Median inference error (bold curve) on a logarithmic scale as a function of the number of iterations for different sizes of network parameterized by $K$ and $N$. The two curves adjacent to the bold curve show the range within which 98% of the errors lie. 99.9% of the errors were below the fourth, topmost curve.

Fig. 3 shows the error (log scale), bottom-up variances (log scale) and top-down means for up to 20 iterations in 10 divergent networks of each of the above sizes. In all cases, the variances converge within a few iterations whereas the means oscillate and diverge. In fact, in every one of the 200,000 networks we explored, the variances converged and did so within a few iterations. This observation suggests that in general, the dynamics of iterative probability propagation in factor analysis networks are determined by the dynamics of the mean updates. Of course, the mean updates depend on the values of the converged variances.

3.4 Fixed Points and a Condition for Global Convergence  It turns out that when the variance updates converge, the dynamics of iterative probability propagation in factor analysis networks become linear. This allows us to derive the fixed point of probability propagation in closed form and write an eigenvalue condition for global convergence.
To analyze the system of mean updates, we define the following length $KN$ vectors of means and the input,

\[
\begin{align*}
\tilde{\eta}^{(i)} &= (\eta_{11}^{(i)}, \eta_{21}^{(i)}, \ldots, \eta_{K1}^{(i)}, \eta_{12}^{(i)}, \ldots, \eta_{K2}^{(i)}, \ldots, \eta_{KN}^{(i)})^T, \\
\tilde{\mu}^{(i)} &= (\mu_{11}^{(i)}, \mu_{12}^{(i)}, \ldots, \mu_{1K}^{(i)}, \mu_{21}^{(i)}, \ldots, \mu_{2K}^{(i)}, \ldots, \mu_{NK}^{(i)})^T, \\
\tilde{x} &= (x_1, x_1, \ldots, x_1, x_2, \ldots, x_2, \ldots, x_N, \ldots, x_N)^T, 
\end{align*}
\]

(28)

where each $x_n$ is repeated $K$ times in the last vector. The network parameters are represented using $KN \times KN$ diagonal matrices,

\[
\tilde{A} = \begin{pmatrix}
\lambda_{11} & \cdots & \lambda_{1K} \\
& \ddots & \vdots \\
& & \lambda_{K1}
\end{pmatrix}, \quad \Psi = \begin{pmatrix}
\psi_1 I & \psi_2 I & \cdots & \psi_N I
\end{pmatrix},
\]

(29)
where $\mathbf{I}$ is the $K \times K$ identity matrix. The converged bottom-up variances are represented using a diagonal matrix,

$$
\Phi = \begin{pmatrix}
\phi_{11} & \cdots & \phi_{1K} \\
\vdots & \ddots & \vdots \\
\phi_{N1} & \cdots & \phi_{NK}
\end{pmatrix}.
$$

(30)

The summation operations in the propagation formulas are represented by a $KN \times KN$ matrix $\hat{\Sigma}_z$ that sums over means sent down from the top layer and a $KN \times KN$ matrix $\hat{\Sigma}_x$ that sums over means sent up from the sensory input:

$$
\hat{\Sigma}_z = \begin{pmatrix}
1 & 1 & \cdots & 1 \\
1 & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots \\
1 & \cdots & \cdots & 1
\end{pmatrix},
\hat{\Sigma}_x = \begin{pmatrix}
1 & 1 & \cdots & 1 \\
1 & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots \\
1 & \cdots & \cdots & 1
\end{pmatrix}.
$$

(31)

These are $N \times N$ matrices of $K \times K$ blocks, where $\mathbf{I}$ is the $K \times K$ block of ones and $\mathbf{I}$ is the $K \times K$ identity matrix.

Using the above representations, the bottom-up pass is given by

$$
\hat{\mu}^{(0)} = \hat{\Lambda}^{-1}\hat{x} - \hat{\Lambda}^{-1}(\hat{\Sigma}_z - \mathbf{I})\hat{\Lambda}\hat{\eta}^{(0)-1},
$$

(32)

and the top-down pass is given by

$$
\eta^{(0)} = (\mathbf{I} + \text{diag}(\hat{\Sigma}_x\hat{\Phi}^{-1}\hat{\Sigma}_x))^{-1}(\hat{\Sigma}_x - \mathbf{I})\hat{\Phi}^{-1}\hat{\mu}^{(0)}.
$$

(33)

Substituting (33) into (32), we get the linear update for $\hat{\mu}$:

$$
\hat{\mu}^{(0)} = \hat{\Lambda}^{-1}\hat{x} - \hat{\Lambda}^{-1}(\hat{\Sigma}_z - \mathbf{I})\hat{\Lambda}(\mathbf{I} + \text{diag}(\hat{\Sigma}_x\hat{\Phi}^{-1}\hat{\Sigma}_x) - \hat{\Phi}^{-1})^{-1}(\hat{\Sigma}_x - \mathbf{I})\hat{\Phi}^{-1}\hat{\mu}^{(0)-1}.
$$

(34)

The variance updates cause the variances to be strictly greater than 0, so all of the inverses in this formula exist. Notice that the matrix, $\mathbf{I} + \text{diag}(\hat{\Sigma}_x\hat{\Phi}^{-1}\hat{\Sigma}_x) - \hat{\Phi}^{-1}$ is in fact a diagonal matrix.

A set of means $\hat{\mu}^*$ is a fixed point if $\hat{\mu}^{(i-1)} = \hat{\mu}^*$ implies $\hat{\mu}^{(i)} = \hat{\mu}^*$, or

$$
(\hat{\Lambda}\hat{\Phi} + (\hat{\Sigma}_z - \mathbf{I})\hat{\Lambda}(\mathbf{I} + \text{diag}(\hat{\Sigma}_x\hat{\Phi}^{-1}\hat{\Sigma}_x) - \hat{\Phi}^{-1})^{-1}(\hat{\Sigma}_x - \mathbf{I}))\hat{\Phi}^{-1}\hat{\mu}^* = \hat{x}.
$$

(35)

If a unique solution to this equation exists, there is a fixed point,

$$
\hat{\mu}^* = \hat{\Phi}(\hat{\Lambda}\hat{\Phi} + (\hat{\Sigma}_z - \mathbf{I})\hat{\Lambda}(\mathbf{I} + \text{diag}(\hat{\Sigma}_x\hat{\Phi}^{-1}\hat{\Sigma}_x) - \hat{\Phi}^{-1})^{-1}(\hat{\Sigma}_x - \mathbf{I}))^{-1}\hat{x}.
$$

(36)
A fixed point exists if the determinant of the \( NK \times NK \) matrix in large braces in (36) is nonzero. Computing this determinant directly is costly, but it can be simplified. The determinant can be written using an \( N \times N \) matrix of \( K \times K \) blocks,

\[
\det = \begin{vmatrix}
D_1 & A_1 D_1 & \ldots & A_1 D_1 \\
A_2 D_2 & D_2 & \ldots & A_2 D_2 \\
\vdots & \vdots & \ddots & \vdots \\
A_N D_N & A_N D_N & \ldots & D_N \\
\end{vmatrix},
\]

(37)

where

\[
D_n = \begin{pmatrix}
\phi_{n1} \lambda_{n1} \\
\phi_{n2} \lambda_{n2} \\
\vdots \\
\phi_{nK} \lambda_{nK}
\end{pmatrix},
\]

(38)

and

\[
A_n = \begin{pmatrix}
0 & \phi_{n1}^{-1} & \ldots & \phi_{nK}^{-1} \\
\phi_{n1}^{-1} & 0 & \ldots & \phi_{nK}^{-1} \\
\vdots & \vdots & \ddots & \vdots \\
\phi_{n1}^{-1} & \phi_{n2}^{-1} & \ldots & 0
\end{pmatrix}.
\]

(39)

Assuming \( \lambda_{nk} \neq 0 \) and since the variance updates leave the variances strictly greater than 0, \( D_n^{-1} \) exists and \( A_n^{-1} \) exists. For the sake of determining whether (37) is nonzero, we can post-multiply each row \( n \) in (37) by \( D_n^{-1} A_n^{-1} \), obtaining

\[
\det \propto \begin{vmatrix}
A_1^{-1} & I & \ldots & I \\
I & A_2^{-1} & \ldots & I \\
\vdots & \vdots & \ddots & \vdots \\
I & I & \ldots & A_N^{-1}
\end{vmatrix} = |I + \sum_{n=1}^{N} (I - A_n^{-1})^{-1}|.
\]

(40)

Using this expression, the existence of a fixed point can be determined by inverting \( N K \times K \) matrices and computing the determinant of a \( K \times K \) matrix.

Reinterpreting the dynamics in (34) as dynamics for \( \tilde{\lambda} \tilde{\mu}^{(0)} \), the stability of a fixed point is determined by the largest eigenvalue of the update matrix,

\[
(\Sigma_x - I) \tilde{\lambda} \left( I + \text{diag}(\Sigma_x \Phi^{-1} \Sigma_x) - \Phi^{-1} \right)^{-1} (\Sigma_x - I) \Phi^{-1} \tilde{\lambda}^{-1}
\]

\[
= \begin{pmatrix}
0 & A_1 D_2^{-1} & \ldots & A_1 D_N^{-1} \\
A_2 D_1^{-1} & 0 & \ldots & A_2 D_N^{-1} \\
\vdots & \vdots & \ddots & \vdots \\
A_N D_1^{-1} & A_N D_2^{-1} & \ldots & 0
\end{pmatrix}.
\]

(41)

If the modulus of the largest eigenvalue is less than 1, the fixed point is stable. Since the system is linear, if a stable fixed point exists, the system will be globally convergent to this point.
Figure 4: The error (logarithmic scale) versus number of iterations (logarithmic scale, maximum of 1000) of probability propagation in the networks used in Fig. 3, with the means initialized to the fixed point solutions. Numerical errors due to machine precision cause divergence from the fixed points, whose errors are shown by horizontal lines. The modulus of the largest eigenvalue is given beneath each plot. Networks are of size (a) $K = 5, N = 10$, (b) $K = 10, N = 20$ and (c) $K = 20, N = 40$.

3.5 Examples of Unstable Fixed Points. Of the 200,000 networks we explored, about 99.9% of the networks converged. Since a stable fixed point implies a globally convergent network, it follows that divergent networks do not have stable fixed points. For each of the 30 divergent networks in Fig. 3, we used 1000 iterations of probability propagation to compute the steady state variances. Then, we computed the modulus of the largest eigenvalue of the system and we computed the fixed point. After initializing the bottom-up means to the fixed point values, we performed 1000 iterations of probability propagation to see if numerical errors due to machine precision would cause divergence from the fixed point.

Fig. 4 shows the error versus number of iterations (on logarithmic scales) for each network, along with the modulus of the largest eigenvalue. As expected, the modulus of the largest eigenvalue is greater than 1 in each case. It is interesting that in some cases, the network diverges from the fixed point and reaches a dynamic attractor that has a lower average error than the fixed point. The basins of attraction of some of the dynamic attractors are local, since in at least some cases the dynamic attractor was not found when the top-down means were initialized to 0 (see the corresponding plots in Fig. 3).
4 Turbo Factor Analysis

To perform maximum likelihood factor analysis in an online fashion, each parameter should be modified to slightly increase the log-probability of the current sensory input, \( \log p(x) \). However, whereas the derivatives of \( \log p(x, z) \) are easy to compute, the derivatives of \( \log p(x) \) are not easy to compute, since \( \log p(x) \) includes terms like \(- \log |\Lambda \Lambda^T + \Psi|\).

So that the derivatives of \( \log p(x, z) \) can be used, the unobserved variables \( z \) are “filled in” using probabilistic inference. If \( q(z) \) is the distribution given by an inference method, then \( \log p(x) \) can be bounded from below using Jensen’s inequality (see (Neal and Hinton 1998)):

\[
F = \int q(z) \log \frac{p(x, z)}{q(z)} dz \leq \log p(x).
\]

(42)

Once inference has determined \( q(z) \) for the current input, the derivatives of \( F \) are given by averaging the derivatives of \( \log p(x, z) \) with respect to \( q(z) \).

\( F \) can be rewritten to show that it consists of the true log-probability of the input minus the Kullback-Leibler pseudodistance between the inference distribution and the true posterior,

\[
F = \log p(x) - \int q(z) \log \frac{q(z)}{p(z|x)} dz.
\]

(43)

To perform online maximum likelihood factor analysis, \( q(z) = p(z|x) \) should be used so that the bound is tight and \( F = \log p(x) \). When inexact inference is used (i.e., \( q(z) \neq p(z|x) \)), following the gradient of \( F \) will compromise between increasing the likelihood of the current input and making the generative model better suited to the approximate inference distribution.

In this paper, we explore methods that infer independent Gaussian factors. If the estimated mean and variance of the \( k \)th factor are \( \hat{z}_k \) and \( v_k \), then, leaving out terms that don’t depend on the parameters,

\[
F = \frac{1}{2} \sum_{n=1}^{N} \log \psi_n + \left[ (x_n - \sum_{j=1}^{K} \lambda_{nj} \hat{z}_j)^2 + \sum_{j=1}^{K} \lambda_{nj}^2 v_j \right] / \psi_n.
\]

(44)

The parameter \( \lambda_{nk} \) can be adjusted by taking a gradient step:

\[
\lambda_{nk} \leftarrow \lambda_{nk} + \eta [\hat{z}_k (x_n - \sum_{j=1}^{K} \lambda_{nj} \hat{z}_j) - v_k \lambda_{nk}] / \psi_n,
\]

(45)

where \( \eta \) is a learning rate.

\( \psi_n \) can be solved for exactly in (44). So, following (Neal and Dayan 1997) we set \( v_k \) equal to a linear combination of its old value and its exact solution:

\[
\psi_n \leftarrow (1 - \eta) \psi_n + \eta [(x_n - \sum_{j=1}^{K} \lambda_{nj} \hat{z}_j)^2 + \sum_{j=1}^{K} \lambda_{nj}^2 v_j].
\]

(46)

Although a different learning rate can be used for updating the weights and the variances, we use equal learning rates.
Turbo factor analysis consists of performing some number of iterations of probability propagation for the current input (e.g., 4 iterations) and then modifying the parameters as described above before processing the next input.

4.1 Method Used to Simulate Data and Train Networks  
To extensively explore the performance of turbo factor analysis, we produced 95 training sets of 200 cases each, with input sizes ranging from 20 sensors to 320 sensors. For each of 19 sizes of factor analyzer, we randomly selected 5 sets of parameters and generated a training set. The factor analyzer sizes were $K \in \{5, 10, 20, 40, 80\}$, $N \in \{20, 40, 80, 160, 320\}$, $N > K$. The elements of $\Lambda$ were independently drawn from a standard normal distribution and each sensor variance was set equal to $1/5$ of its noise-free variance, i.e., $\psi_n = (\sum_k \lambda_n^2)/5$.

For each factor analyzer and simulated data set, we estimated the optimal log-probability of the data using 100 iterations of exact EM to try to find the best model. Since each training set was quite small, it is possible that the factor analyzer from which the data came was actually in the basin of attraction of a local optimum of the log-probability of the data.

For all learning methods, the size of the model to be trained was set equal to the size of the model that was used to generate the data. The parameters were initialized randomly using the sensor variances estimated from the first 30 training cases.

It is difficult to be fair when comparing online learning algorithms, since the speed of learning is sensitive to the learning rate and different methods may have different suitable ranges of learning rates. If the learning rate is not allowed to change during learning, there is usually a trade-off between speed of convergence and the final log-probability of the training set. Allowing the learning rate to change makes fair comparisons even more difficult, since different methods may have different ideal strategies for adapting the learning rate.

One way to try to circumvent these ambiguities is to search for achievable learning curves, regardless of whether or not a simple expression for modifying the learning rate exists. So, for a given method and a randomly drawn set of initial parameters, we performed one separate epoch of learning using each of the learning rates, $1, 0.5, 0.25, \ldots, 0.5^{30}$ and then picked the model and learning rate that gave the greatest increase in log-probability of the data. Each successive learning rate was determined by performing an epoch of learning with the old learning rate and then performing an epoch of learning with the old learning rate scaled by 0.75. The model and learning rate that gave the greatest increase in log-probability of the data was kept.
Figure 5: The achievable errors after the same number of epochs of learning for turbo factor analysis using 4 iterations versus turbo factor analysis using 1 iteration. In each plot, the horizontal axis gives the log-probability error on a logarithmic scale for turbo factor analysis using 1 iteration and the vertical axis gives the error after the same number of epochs for turbo factor analysis using 4 iterations.

### 4.2 Discussion of the Results for Simulated Data

In analyzing these results, we are mainly interested in comparing the achievable curves for different methods and how the differences scale with \( K \) and \( N \). For two methods with the same \( K \) and \( N \) trained on the same data, we plot the log-probability error (optimal log-probability minus log-probability under the learned model) of one method against the log-probability error of the other method. This approach removes the number of epochs from the comparison and illuminates what error is achievable with one method for the error achieved by another method using the same number of epochs.

Fig. 5 shows the achievable errors after the same number of epochs of learning for turbo factor analysis using 4 iterations versus turbo factor analysis using 1 iteration. Aside from a small number of local optima, using 4 iterations of probability propagation produces networks with lower errors.
Figure 6: The achievable errors after the same number of epochs of learning for turbo factor analysis using 4 iterations versus online factor analysis using the diagonal approximation to the inverse covariance matrix. In each plot, the horizontal axis gives the log-probability error on a logarithmic scale for the diagonal approximation and the vertical axis gives the error after the same number of epochs for turbo factor analysis using 4 iterations.

than those learned using 1 iteration of probability propagation. The difference is most significant for networks with large $K$, where in Sec. 3.2 we found that the convergence of the inference error was slower.

Fig. 6 shows the achievable errors after the same number of epochs of learning for turbo factor analysis using 4 iterations versus online factor analysis using the diagonal approximation described in Sec. 2.1. The diagonal approximation appears to be adequate for smaller networks, but is not as effective for larger networks.

Fig. 7 shows the achievable errors after the same number of epochs of learning for turbo factor analysis using 4 iterations versus wake-sleep learning using 4 iterations. Generally, turbo factor analysis achieves much smaller errors than wake-sleep learning, although for small $K$ wake-sleep performs
Figure 7: The achievable errors after the same number of epochs of learning for turbo factor analysis using 4 iterations versus wake-sleep learning using 4 iterations. In each plot, the horizontal axis gives the log-probability error on a logarithmic scale for wake-sleep learning and the vertical axis gives the error after the same number of epochs for turbo factor analysis.

better very close to the optimum log-probability. This may be due to the fact that the noise introduced by wake-sleep sampling is smaller for lower $K$. The most significant difference between the methods occurs for large $K$, where aside from local optima turbo factor analysis achieves nearly optimal log-probabilities while the log-probabilities for wake-sleep learning are still close to the values they had at the start of learning.

Fig. 8 shows the achievable errors after the same number of epochs of learning for turbo factor analysis using 4 iterations versus online learning using 4 iterations of conjugate inference. The two methods perform similarly in most cases, aside from the small number of local optima found by turbo factor analysis.
Figure 8: The achievable errors after the same number of epochs of learning for turbo factor analysis using 4 iterations versus online learning using 4 iterations of conjugate inference. In each plot, the horizontal axis gives the log-probability error on a logarithmic scale for the conjugate iterative inference and the vertical axis gives the error after the same number of epochs for turbo factor analysis.

4.3 Online Face Recognition. The factor analyzer can be used for probabilistic dimensionality reduction if we want to train a flexible model (e.g., a multilayer perceptron) on high-dimensional data. Fig. 9a shows examples from a set of 30,000 $20 \times 28$ greyscale face images of 18 different people. In contrast to other data sets used to test face recognition methods – notably, the FERET data set (Phillips et al. 1999) – the faces in this data set include wide variation in expression and pose.

To make classification more difficult, we normalized the images for each person so that each pixel has the same mean and variance. Examples of the normalized images are shown in Fig. 9b.

We used turbo factor analysis and wake-sleep learning to reduce the dimensionality of the data online from 560 dimensions to 40 dimensions. For turbo factor analysis, we rather arbitrarily chose a learning rate of 0.0001,
but for wake-sleep learning we tried learning rates ranging from 0.1 down to 0.0001.

At the same time that the factor analyzer was being trained online, a multilayer perceptron with one hidden layer of 160 tanh units and one output layer of 18 softmax units was trained using gradient descent to predict face identity from the factor means. The learning rate for the multilayer perceptron was set to the highest value that was generally found to work (0.05) and this value was used regardless of the method used for dimensionality reduction.

Upon each image presentation, a prediction of the identity of the person in the image was made and the 0/1 loss measured before the factor analyzer parameters and the multilayer perceptron parameters were modified. Note that the classification error was not backpropagated into the factor analyzer network.

Fig. 10 shows the online error curves obtained by filtering the 0/1 losses with a uniform window 3000 samples long. The performance of turbo factor analysis (solid line) is compared with 4 versions of wake-sleep learning
Figure 10: Online face recognition error curves for turbo factor analysis (solid), wake-sleep learning with different learning rates (dashed) and two forms of online nearest neighbor classification (dot-dashed). The dotted line at the top of the plot shows the error rate obtained by guessing.

that had different learning rates (dashed lines) and the error rate obtained by guessing (dotted line). The figure also shows the error curves for two forms of online nearest neighbor classification, where only the most recent $W$ cases are used to make a prediction. The form of nearest neighbor classification that performs the worst has $W$ set so that the storage requirements are the same as for the factor analysis / multilayer perceptron method. The better form of nearest neighbor classification has $W$ set so that the number of computations is the same as for the turbo factor analysis / multilayer perceptron method.

The final error rate for the turbo factor analysis method is lower than the final error rates for all the models that used wake-sleep learning to train the factor analyzer. In comparison with the nearest neighbor classifiers, the performance of turbo factor analysis is similar to what is often expected with parametric methods: it performs better than the nearest neighbor method with constrained memory.

5 Discussion

Our investigation into the behavior and performance of iterative probability propagation in a densely connected adaptive network was inspired by the
record-breaking performance of this technique applied to error-correcting decoding. We chose to explore the factor analyzer network, because exact inference can be used to measure the error of the iterative probability propagation method and because the propagation equations can be analyzed. We showed that iterative probability propagation in the densely connected factor analyzer network usually takes just a few iterations to achieve a low inference error, even in networks with 320 sensors and 80 factors. We also derived an expression for the algorithm’s fixed point and provided an eigenvalue condition for global convergence. Although we do not recommend iterative probability propagation as a replacement for exact inference in factor analysis networks, results on simulated data and real data show that the method performs remarkably well for both inference and learning, considering that the network is densely connected.

It was pointed out in (Frey and MacKay 1998b) that iterative probability propagation in networks for error-correcting codes may work well because the posterior probability mass is concentrated in a single mode, or “microstate”. The results in this paper partly support this hypothesis, but also show that even when the posterior is unimodal, iterative probability propagation sometimes – although very infrequently – diverges. Divergent cases in error-correcting decoding have been found for small networks (McEliece, Rodemich and Cheng 1996), but there have not been any reports (even anecdotal) of divergence in the large networks of practical interest. Our results also show that in factor analyzer networks, divergence occurs less often for larger networks. One direction for further research would be to study the eigenvalue condition for global convergence in the infinite limit of $K$ and $N$.

Although this work was motivated by research on iterative probability propagation in networks for error-correcting decoding, two important differences should be clarified. First, while the factor analyzer contains only real-valued variables, the networks studied so far for error-correcting decoding contain only discrete variables. Second, some of the techniques used to analyze iterative probability propagation in networks for error-correcting decoding (Gallager 1963; MacKay 1999) rely on the minimal cycle length growing with $K$ and $N$. Obviously, the minimal cycle length in the factor analyzer network is 4, independent of $K$ and $N$.

Another possible direction for further analysis of probability propagation in the factor analyzer network is to account for “double counting” of probability messages (Pearl 1988). Double counting occurs when a message circulating in the network passes through a vertex for the second time. Before double counting occurs, the network appears to be a tree and inference is exact, up to the subgraph that forms the tree. In the factor analyzer network, double counting occurs after the first iteration. We might hope that “equal double counting” occurs (Weiss 1999), so that the errors due to double counting balance out. This approach is related to balancing the “computation tree” (Frey, Koetter and Vardy 1998), which is the graph obtained
by unrolling the messages passed in the original network through time.  

Currently, we are interested in using iterative probability propagation
for inference and learning in hierarchical, nonlinear models for vision, where
exact inference is exponentially difficult. Although analyzing iterative prob-
ability propagation in these models is not so straightforward, we believe the
potential benefit is much greater.

Acknowledgments

This research was supported by grants from CITO, NSERC and the Beck-
man Foundation.

References

decoding on graphs with a single cycle. In Proceedings of IEEE International
Symposium on Information Theory.

signal separation. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., edi-
MA.


Bell, A. J. and Sejnowski, T. J. 1995. An information maximization approach to

Berrou, C. and Glavieux, A. 1996. Near optimum error correcting coding and de-


Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. 1995. The Helmholtz ma-

New York NY.

NY.

Forney, Jr., G. D., Kschischang, F. R., and Marcus, B. 1998. Iterative decoding of


