Unsupervised Discovery of Non-Linear Structure using Contrastive Backpropagation

G. E. Hinton, S. Osindero, M. Welling and Y. W. Teh

Department of Computer Science

University of Toronto

Toronto, Canada M5S 3G4

December 28, 2005

Abstract

We describe a way of modelling high-dimensional data-vectors by using an unsupervised, non-linear, multilayer neural network in which the activity of each neuron-like unit makes an additive contribution to a global energy score that indicates how surprised the network is by the data-vector. The connection weights which determine how the activity of each unit depends on

the activities in earlier layers are learned by minimizing the energy assigned to data-vectors that are actually observed and maximizing the energy assigned to "confabulations" that are generated by perturbing an observed data-vector in a direction that decreases its energy under the current model.

The backpropagation algorithm (Rumelhart et al., 1986) trains the units in the intermediate layers of a feedforward neural net to represent features of the input vector that are useful for predicting the desired output. This is achieved by propagating information about the discrepancy between the actual output and the desired output backwards through the net to compute how to change the connection weights in a direction that reduces the discrepancy. In this paper we show how to use backpropagation to learn features and constraints when each input vector is not accompanied by a supervision signal that specifies the desired output.

When no desired output is specified, it is not immediately obvious what the goal of learning should be. We assume here that the aim is to characterize the observed data in terms of many different features and constraints that can be interpreted as hidden factors. These hidden factors could be used for subsequent decision making or they could be used to detect highly improbable data-vectors by using the global energy. We define the probability that the network assigns to a data-vector, $\mathbf{x}$, by comparing its global energy, $E(\mathbf{x})$, with the energies of all possible data-vectors, $\mathbf{v}$:

$$p(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{\sum_{\mathbf{v}} e^{-E(\mathbf{v})}} \tag{1}$$

Intuitively, a good unsupervised learning procedure should find hidden factors that assign high probability to patterns that typically occur. This can be achieved by lowering the energies of observed data-vectors and raising the energies of "negative" data-vectors — patterns that ought to be observed if the hidden factors constituted a good model of the data. By using the current model to generate a set of negative data-vectors we can convert an unsupervised learning task into the supervised task of assigning low energies to the observed data-vectors and high energies to the negative data-vectors. But notice that the set of negative data-vectors depends on the current model and it will change as the model learns.

The quality of the set of features and constraints discovered by the neural network can be quantified by the summed log probability that gets assigned to the observed data-vectors. The contribution of a single data-vector to this sum is:

$$\log p(\mathbf{x}) = -E(\mathbf{x}) - \log \sum_{\mathbf{v}} e^{-E(\mathbf{v})} \tag{2}$$

The features and constraints can be improved by repeatedly adjusting the weights on the connections so as to maximize the log probability of the observed data. To perform gradient ascent in the log likelihood we would need to compute exact derivatives of the log probabilities:

$$\Delta w_{ij} \propto \frac{\partial \log p(\mathbf{x})}{\partial w_{ij}} \;=\; -\frac{\partial E(\mathbf{x})}{\partial w_{ij}} \;+\; \sum_{\mathbf{v}} p(\mathbf{v}) \frac{\partial E(\mathbf{v})}{\partial w_{ij}} \tag{3}$$

where w_{ij} is the weight on the connection from unit i in one layer to unit j in the next layer.

The first term is easy to compute. We assume that each unit, j, sums the weighted activities coming from units, i, in the layer below to get its total input, $z_j = \sum_i y_i w_{ij}$, where an activity y_i in the layer below is equal to x_i if it is the input layer. A smooth non-linear function of z_j is then used to compute the unit's activity, y_j. The energy contributed by the unit can be any smooth function of its activity. In this paper we use two layers of non-linear hidden units and the energy is determined by the activities of units in the second hidden layer:

$$E(x) = \sum_j \lambda_j log(1 + y_j^2) \tag{4}$$

where y_j is the activity of unit j in the second hidden layer and λ_j is a scale parameter that is also learned by contrastive backpropagation. This "heavy-tailed" energy contribution is good for modeling constraints that are usually satisfied fairly precisely and occasionally violated by a lot. In images of natural scenes, for example, a local, oriented edge-filter will have an output of almost exactly zero almost everywhere. On the few occasions when its output departs from zero, however, it may be quite large, so the distribution of the violations is very non-Gaussian. By using the energy contributions in Eq. 4 we encourage the network to model the data distribution by finding constraints of this type (Hinton and Teh, 2001).

After performing a forward pass through the network to compute the activities of all the units, we do a backward pass as described in Rumelhart et al. (1986). The backward pass uses the chain rule to compute $\partial E(\mathbf{x})/\partial w_{ij}$ for every connection weight, and by backpropagating all the way to the inputs we can also compute $\partial E(\mathbf{x})/\partial x_i$ for each component, x_i, of the input vector.

Unfortunately, the second term in Eq. 3 is much harder to deal with. It involves a weighted average of the derivatives from all conceivable data-vectors so it cannot be computed efficiently except in special cases. We usually expect, however, that this average will be dominated by a very small fraction of the conceivable data-vectors, so it seems reasonable to approximate this term by averaging $\partial E(\mathbf{x})/\partial w_{ij}$ over a relatively small number of samples from the distribution $p(\cdot)$. One way to generate samples from this distribution is to run a Markov chain that simulates a physical process with thermal noise. If we think of the dataspace as forming a horizontal plane and we represent the energy of each data-vector as height, the neural network defines a potential energy surface whose height and gradient are easy to compute. We imagine a particle on this surface that tends to move downhill but is also jittered by additional Gaussian noise. After enough steps, the particle will have lost all information about where it started and if we use small enough steps, its probability of being at any particular point in the dataspace will be given by the Boltzmann distribution in Eq 1. This is a painfully slow way of generating samples and even if the equilibrium distribution is reached, the variance created by sampling may mask the true learning signal.

Rather surprisingly, it is unnecessary to allow the simulated physical process to reach the equilibrium distribution. If we start the process at an observed data-vector and just run it for a few steps, we can generate a "confabulation" that works very well for adjusting the weights (Hinton, 2002). Intuitively, if the Markov chain starts to diverge from the data in a systematic way, we already have evidence that the model is imperfect and that it can be improved (in this local region of the dataspace) by reducing the energy of the initial data-vector and raising the energy of the confabulation. It is theoretically possible that this learning procedure will cause the model to assign very low energies to unvisited regions of the dataspace that are far from any data-vector. But the fact that the learning works well on a variety of tasks suggests that this theoretical problem is insufficient grounds for rejecting the learning procedure, just as the existence of local minima was insufficient grounds for rejecting backpropagation.

The "contrastive backpropagation" learning procedure cycles through the observed data-vectors adjusting each weight by:

$$\Delta w_{ij} = \eta \left(-\frac{\partial E(\mathbf{x})}{\partial w_{ij}} \; + \; \frac{\partial E(\hat{\mathbf{x}})}{\partial w_{ij}} \right) \tag{5}$$

where η is a learning rate and $\hat{\mathbf{x}}$ is a confabulation produced by starting at $\mathbf{x}$ and noisily following the gradient of the energy surface for a few steps (see note 1).

To illustrate the learning procedure, we applied it to the task of discovering the non-linear kinematic constraints in a simulated three dimensional "arm" that has five rigid links and five ball-joints. The first ball-joint attaches the arm to the origin, and each data-vector consists of the 15 cartesian coordinates of the remaining link end-points. This apparently 15-dimensional data really has only 10 degrees of freedom because of the 5 one-dimensional constraints imposed by the 5 rigid links. These constraints are of the form:

$$(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2 + (z_i - z_{i+1})^2 - l_{i,i+1}^2 = 0 \tag{6}$$

where i and $i+1$ index neighboring joints and $l_{i,i+1}$ is the length of the link between them. Because the constraints are highly non-linear, linear dimensionality-reduction methods like principal components analysis or factor analysis are of little help.

We used a neural net with 15 input units and two hidden layers. Each of the 15 units in the first hidden layer computes a weighted average of the inputs and then squares it. Each of the 5 units in the top layer computes a weighted average of the squares provided by the first hidden layer and adds a learned bias. It is clear that with the right weights and biases, each top-layer unit could implement one of the constraints represented by Eq. 6 by producing an output of exactly zero if and only if the constraint is satisfied. The question is whether the network can discover the appropriate weights and biases just by observing the data.

For this example, the units in the first hidden layer do not contribute to the energy and the units in the second hidden layer each contribute an energy of $\lambda_j \log(1 + y_j^2)$ This "heavy-tailed" energy function penalizes top-level units with non-zero outputs, but changing the output has little effect on the penalty if the output is already large.

Figure 1 shows the weights and top-level biases that were learned by contrastive backpropagation. For each pair of neighboring joints, there are three units in the first hidden layer that have learned to compute differences between the coordinates of the two joints. These differences are always computed in three orthogonal directions. Each unit in the second hidden layer has learned a linear combination of the 5 constraints, but it uses weights of exactly the same size for the three squared differences in each constraint so that it can exactly cancel the fixed sum of these three squared differences by using its bias.

The same network can also learn the 5 constraints when a random 10% of the input variables are missing from each data-vector. The missing input variables are treated as additional parameters which are initialized at random values and are learned using a version of Eq. 5 in which w_{ij} is replaced by x_i. The random inputs mean that each instance of a constraint is only satisfied with a probability of $.9^6 = .53$ at the start of learning. However, the heavy-tailed energy function means that strongly violated constraints only contribute a very small gradient, so the learning is driven by the accurately satisfied constraints.

We have also applied a similar neural network on the more challenging task of learning features that allow us to compactly describe the statistical structure within small patches of digitised images of natural scenes. For this task, we used the same layered architecture, activation functions, and energy functions as described previously, but this time in a net with 256 units in the input layer and 400 units in each of the two hidden layers. We also arranged the units within each hidden layer on a 20×20 square grid, and topographically restricted the connectivity so that each unit in the first hidden layer could only send connections to the unit at the same grid position in the second hidden layer and to this unit's 24 nearest neighbours. (see note 2).

Figure 2 illustrates some of the features learned in such a model. The first layer units have self-organised to form a representation of the image patches in terms of a set of oriented, band-pass features. These features bear a striking resemblance to the receptive fields of simple cells found in the primary visual cortex of most mammals and are also similar to the features learned in other models that seek to capture statistical structure in natural images (Olshausen and Field, 1996; Bell and Sejnowski, 1997). The second layer units display similar response preferences for orientation and spatial frequency, but appear to be somewhat insensitive to the spatial phase present in the input. As a result of the restricted connectivity between the two hidden layers, the features form a topographic map with local continuity in spatial location, orientation and spatial frequency.

The contrastive backpropagation learning procedure is quite flexible. It puts no constraints other than smoothness on the activation functions or the functions for converting activations into energy contributions. For example, the procedure can easily be modified to use recurrent neural networks that receive time-varying inputs such as video sequences. The energy of a whole sequence is simply defined to be some function of the time history of the activations of the hidden units. Backpropagation through time (Werbos, 1990) can then be used to obtain the derivatives of the energy with respect to the connection weights and also the energy gradients required for generating a whole confabulated sequence.

Notes

1. We used a simplified version of the Hybrid Monte Carlo procedure in which the particle is given a random initial momentum and its deterministic trajectory along the energy surface is then simulated for a number of time steps (20 for the example

in figure 1 and 10 for figure 2. If this simulation has no numerical errors the increase, ΔE, in the combined potential and kinetic energy will be zero. If ΔE is positive, the particle is returned to its initial position with a probability of $1 - exp(-\Delta E)$. The step size is slowly adapted so that only about 10% of the trajectories get rejected. Numerical errors up to second order are eliminated by using a "leapfrog" method (Neal, 1996) which uses the potential energy gradient at time t to compute the velocity increment between time $t - \frac{1}{2}$ and $t + \frac{1}{2}$ and uses the velocity at time $t + \frac{1}{2}$ to compute the position increment between time t and $t + 1$.

2. The original data for this model were vectors representing the pixel intensities in 20×20 patches extracted from photographs of natural scenes (van Hateren and van der Schaaf, 1998). These vectors then underwent standard, and biologically motivated pre-processing (van Hateren and van der Schaaf, 1998; Olshausen and Field, 1996) which involved subtracting the mean value from each pixel and then taking the variance normalised projection onto the leading 256 eigenvectors of the pixel covariance matrix. Topographic maps can also be learned by using a similar architecture and energy function, but replacing contrastive backpropagation with a stochastic sampling procedure (Osindero et al., 2006). It is harder, however, to extend the stochastic sampling approach to work with more hidden layers, whereas this extension is trivial with contrastive backpropagation. Other methods of learning topographic maps from natural image patches (Hyvarinen and Hoyer, 2001) are also hard to extend to more hidden layers.

Acknowledgments

We would like to thank David MacKay, Radford Neal, Sam Roweis, Zoubin Ghahramani, Chris Williams, Carl Rasmussen, Brian Sallans, Javier Movellan and Tim Marks for helpful discussions and two anonymous referees for improving the manuscript. This research was supported by the Gatsby Charitable foundation, NSERC, CFI, and CIAR.

References

Bell, A. J. and Sejnowski, T. J. (1997). The "independent components" of natural scenes are edge filters. *Vision Research*, 37(23):3327–3338.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

Hinton, G. E. and Teh, Y. W. (2001). Discovering multiple constraints that are frequently approximately satisfied. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-2001)*. Morgan Kaufmann, San Francisco.

Hyvarinen, A. and Hoyer, P. O. (2001). A two-layer sparse coding model learns simple and complex cell receptive fields and topography from natural images. *Vision Research*, 41(18):2413–2423.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks, Lecture Notes in Statis-*

tics No. 118. Springer.

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

Osindero, S., Welling, M., and Hinton, G. E. (2006). Modelling the statistics of natural images with topographic product of student-t models. *Neural Computation*, 18(in press).

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

van Hateren, J. H. and van der Schaaf, A. (1998). Independent component filters of natural images compared with simple cells in primary visual cortex. *Proc R Soc Lond B Biol Sci*, 265(1394):359–66.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

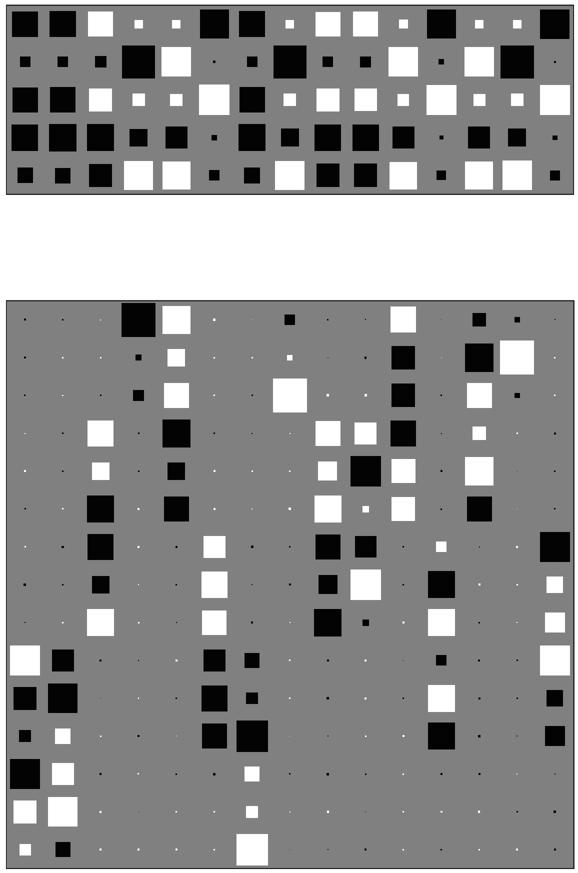

Figure 1: The areas of the small black and white rectangles represent the magnitudes of the negative and positive weights learned by the network. Each column in the lower block represents the weights on the connections to a unit in the first hidden layer from the joint coordinates $x_1, y_1, z_1, x_2, y_2, z_2 \ldots x_5, y_5, z_5$. For example, the first, second and seventh columns show the weights of three hidden units that compute the squared distances between the last two joints in three orthogonal directions. Each row in the higher block represents the weights on connections from units in the first hidden layer to a unit in the second hidden layer. For example, the first, second and seventh units in the first hidden layer have equal negative weights to the unit represented by the third row in the higher block. The weights started with very small random values and were learned by 3300 passes through a training set of 800 random arm configurations in which every link was of length 1. The weights were updated after every 100 training cases. To eliminate unnecessary weights, a decay towards zero of 0.0002 was added to the weight change, Δw_{ij} specified by Eq. 5 before multipling by the learning rate for that connection, η_{ij}, which started at 0.0001. η_{ij} increased by 1% if Δw_{ij} agreed in sign with its previous value and decreased by 5% if it disagreed. To further speed learning without causing divergent oscillations, each weight update included 0.9 times the previous weight update.

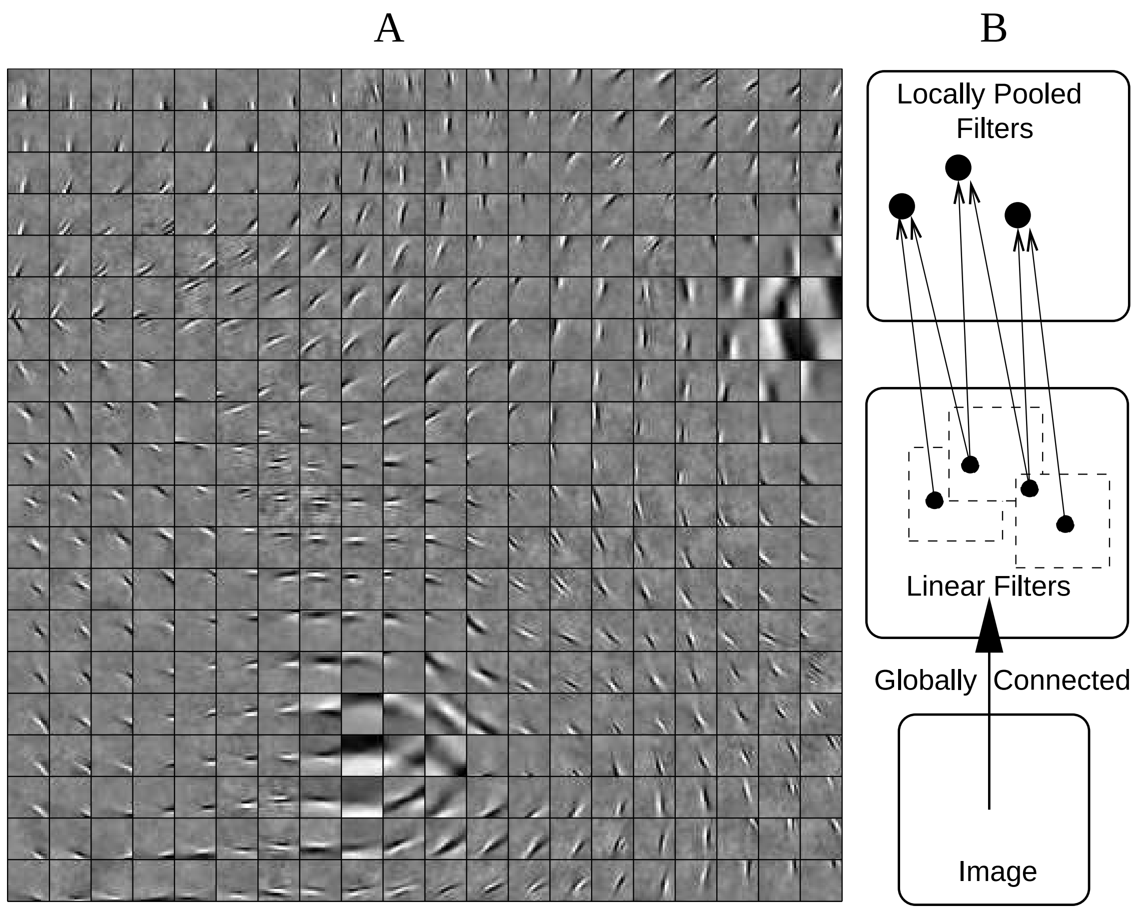

Figure 2: Each small square depicts the basis function associated with the unit at the corresponding grid position within the first layer of spatially ordered units. The image within each small square indicates the contribution to the "represented" image that each unit would have, were it's activity level set to 1. The basis functions are obtained by taking the pseudo-inverse of the bottom up weight matrix, and we show these rather than the weights themselves since they provide greater clarity within a small figure.