USING NEURAL NETWORKS TO MONITOR FOR RARE FAILURES

Geoffrey E. Hinton and Brendan J. Frey
University of Toronto

SUMMARY

Neural networks are typically trained to perform a non-linear mapping from a set of measurements to a response. For monitoring purposes, it would be useful to learn a mapping from sensor values into the probability that the plant was malfunctioning. For rare or unanticipated failures, however, this is infeasible because there are few or no examples to use for training. This paper describes a new type of neural network that can be used to extract the underlying regularities from a normally functioning system and then notice when these regularities are violated.

INTRODUCTION

Artificial neural networks typically consist of many simple, neuron-like processing elements called "units" that interact using weighted connections. Each unit has a "state" or "activity level" that is determined by the input received from other units in the network. There are many possible variations within this general framework. One common, simplifying assumption is that the combined effects of the rest of the network on the j^{th} unit are mediated by a single scalar quantity, x_j. This quantity, which is called the "total input" of unit j, is usually taken to be a *linear* function of the activity levels of the units that provide input to j:

$$x_j = b_j + \sum_i y_i w_{ij} \tag{1}$$

where y_i is the state of the i^{th} unit, w_{ij} is the weight on the connection from the i^{th} to the j^{th} unit and b_j is the bias of the j^{th} unit. The state of a unit is typically defined to be a non-linear function of its total input. For units with continuous states one typical non-linear input-output function is the logistic function which has a nice simple derivative

$$y_j = \frac{1}{1+e^{-x_j}} \tag{2}$$

$$\frac{dy_j}{dx_j} = y_j(1-y_j) \tag{3}$$

Some artificial neural networks restrict the connectivity to be feedforward whereas others allow cycles in the connectivity so that the network state can follow complex trajectories over time or settle down to a stable state over many iterations. In this paper we focus on layered feedforward networks.

Neural network learning procedures can be divided into three broad classes: Supervised procedures (*i.e.* classification or regression) which require a teacher to specify the desired output vector, reinforcement procedures which only require a single scalar evaluation of the goodness of the output, and unsupervised or self-supervised procedures (probability density estimation) which construct internal models that capture regularities in their input vectors without receiving any additional information. Most of this paper focuses on self-supervised learning.

The most widely used neural network learning procedure is a called "backpropagation" [1]. It can be viewed as a generalization of logistic regression to feedforward networks which have layers of hidden units between the input and output units. Backpropagation is used to compute the derivatives, with respect to the weights, of an error measure, E, which is usually the squared difference between the actual and the desired output of the network.

For each training case, c, we first use a forward pass, starting at the input units, to compute the activity levels of all the units in the network. Then we use a backward pass, starting at the output units, to compute $\partial E/\partial y_j$ for all the hidden units. For a hidden unit, j, in layer ℓ the only way it can affect the error is via its effects on the units, k, in the next layer, $\ell+1$. So we have

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k}\frac{dy_k}{dx_k}\frac{dx_k}{dy_j} = \sum_k \frac{\partial E}{\partial y_k} y_k(1-y_k)w_{jk} \tag{4}$$

where the index c has been suppressed for clarity. So if $\partial E/\partial y_k$ is already known for all units in layer $\ell+1$, it is easy to compute the same quantity for units in layer ℓ. Notice that the computation performed during the backward pass is very similar in form to the computation performed during the forward pass (though it propagates error derivatives instead of activity levels, and it is entirely linear in the error derivatives).

Once $\partial E/\partial y_j$ is known for all of the units, it is straighforward to compute $\partial E/\partial w_{ij}$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j}\frac{dy_j}{dx_j}\frac{\partial x_j}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} y_j(1-y_j)y_i \tag{5}$$

In networks with hidden units, the error surface may contain many local minima, so it is possible that steepest descent in weight space will get stuck at poor local minima. In practice, this does not seem to be a serious problem. Backpropagation has been tried for a wide variety of tasks and *poor* local minima are rarely encountered. Starting with small random weights to break symmetry, we typically find that the network produces different solutions each time, but these solutions are typically fairly similar in terms of their performance on training and test sets. In practice, the most serious limitation of backpropagation is the speed of convergence rather than the presence of non-global minima. Convergence time can be improved by using the gradient information in more sophisticated ways, but for large netorks, much bigger improvements are likely to result from using a more modular approach.

SELF-SUPERVISED BACKPROPAGATION

Standard backpropagation is useful for classification or regression but it is not so obvious how it can be used for unsupervised learning in which there is no teacher to specify the desired output for the training cases. The trick is to use the input itself to do the supervision, using a multilayer "encoder" network in which the desired output vector is identical with the input vector. The network must learn to compute an approximation to the identity mapping for all the input vectors in its training set, and if the middle layer of the network contains fewer units than the input layer, the learning procedure must construct a compact, invertible code for each input vector.

The use of self-supervised backpropagation to construct compact codes resembles the use of principal components analysis to perform dimensionality reduction, but it has the advantage that it allows the code to be a non-linear transformation of the input vector. Early research on self-supervised backpropagation focussed on minimizing the squared error between the input to the network and its reconstruction from the activities of the hidden units in the "code" layer. But if we take the coding analogy seriously, we must also consider the cost of transmitting the hidden activities. In the coding framework, the aim is to communicate the input vector to a receiver using as few bits as possible. We do this by sending the code plus the reconstruction error using that code. Minimizing the squared reconstruction error is equivalent to minimizing the number of bits required to communicate this error when it is coded using a zero-mean Gaussian. If we also minimize the number of bits required to communicate the activities of the code units, we get interesting algorithms for discovering non-linear latent variables. In the special case in which the hidden units are linear and their activities are coded using a Gaussian this coding approach reduces to maximum likelihood factor analysis.

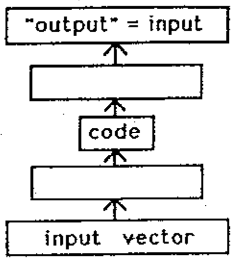

Figure 1: A network which attempts to reconstruct its input vector on its output units. If the units are all linear, this network performs a version of principal components analysis, with the weight vectors of the hidden units corresponding to the principal components. The squared reconstruction error of an input vector measures the squared distance of that vector from the hyperplane spanned by the principal components. This distance can be used to detect input vectors that are not typical of the training distribution, provided that distribution is well-characterized by the principal components.

The part of an encoder network which converts the code back into a reconstruction of the input can be viewed as a generative model because it converts the states of hidden variables into an expected observation. The part of an encoder network that converts the input into a code is a "recognition" model. An interesting aspect of self-supervised backpropagation is that each of these two models is involved in training the other. It is not immediately obvious how the recognition model relates to standard statistical methods of fitting generative models to data. It turns out that the recognition model can be viewed as a way of using a neural network to approximate the E step of the EM algorithm which is a standard statistical method of fitting models to data [2]. The use of a separate recognition model allows EM to be applied in situations where it is intractable to perform the E step exactly [3, 4, 5].

In introducing the idea of self-supervised backpropagation, we assumed that the input would be reconstructed on a different set of output units. For a biological system it is more sensible to fold over the network shown in figure 1 so that the input is reconstructed on the very same units. The generative model then corresponds to top-down weights from a hidden layer to the input and the recognition model corresponds to bottom-up weights from the input to the hidden layer. It is easy to generalize this architecture to a hierarchical system in which there are many levels of coding.

STOCHASTIC GENERATIVE NETWORKS

Instead of using deterministic neurons whose output is a real number as in equation 2, we can use stochastic binary neurons which use the same equation to determine the *probability* of outputting a 1. Multilayer networks of stochastic neurons can be used as top-down generative models that transform random uncorrelated noise in the top hidden layer into highly structured patterns in the bottom, visible layer as shown in figure 2.

Given a generative model of this kind parameterized by the top-down weights there is an obvious way to perform unsupervised learning. The generative weights are adjusted to maximize the probability that the visible vectors generated by the model would match the observed data. If we could fit a model of this kind to typical vectors of sensor values from a plant, we could then use

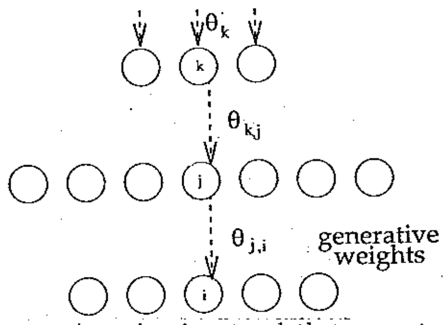

Figure 2: A stochastic network that *generates* data. If the weights can be set so that the generated data resembles observed data, this network is a good model of the observed data. It can then be used to estimate the likelihood that data of unknown origin came from the same source. In particular, a generative model of a normally functioning plant can be used to estimate the likelihood that the current sensor readings are normal.

the model to detect cases in which the sensor readings departed significantly from the usual distribution.

Unfortunately, to compute the derivatives of the log probability of a visible vector, **d**, with respect to the generative weights, θ, it is necessary to consider all possible ways in which **d** could be generated. For each possible binary representation α in the hidden units the derivative needs to be weighted by the posterior probability of α given **d** and θ:

$$P(\alpha|\mathbf{d},\theta) = \frac{P(\alpha|\theta)P(\mathbf{d}|\alpha,\theta)}{\sum_\beta P(\beta|\theta)P(\mathbf{d}|\beta,\theta)} \qquad (6)$$

It is intractable to compute every $P(\alpha|\mathbf{d},\theta)$, so instead of minimizing $-\log P(\mathbf{d}|\theta)$, we minimize an easily computed upper bound on this quantity that depends on some additional parameters, ϕ:

$$F(\mathbf{d}|\theta,\phi) = -\sum_\alpha Q(\alpha|\mathbf{d},\phi)\log P(\alpha,\mathbf{d}|\theta) + \sum_\alpha Q(\alpha|\mathbf{d},\phi)\log Q(\alpha|\mathbf{d},\phi) \qquad (7)$$

If we view $-\log P(\alpha,\mathbf{d}|\theta)$ as an energy, $F(\mathbf{d}|\theta,\phi)$ is a Helmholtz free energy and is equal to $-\log P(\mathbf{d}|\theta)$ when the distribution $Q(.|\mathbf{d},\phi)$ is the same as the posterior distribution $P(.|\mathbf{d},\theta)$. Otherwise, $F(\mathbf{d}|\theta,\phi)$ exceeds $-\log P(\mathbf{d}|\theta)$ by the asymmetric divergence:

$$\sum_\alpha Q(\alpha|\mathbf{d},\phi)\log\frac{Q(\alpha|\mathbf{d},\phi)}{P(\alpha|\mathbf{d},\theta)} \qquad (8)$$

We restrict $Q(.|\mathbf{d},\phi)$ to be a product distribution within each layer that is conditional on the binary states in the layer below. We can then compute the distribution efficiently using a bottom-up recognition network as shown in figure 3. The recognition weights, ϕ, take the binary activities in one layer and stochastically produce binary activities in the layer above using probabilities given by a logistic function. So for a given visible vector, the recognition weights may produce many different representations in the hidden layers, but we can get an unbiased sample from the distribution $Q(.|\mathbf{d},\phi)$ in a single bottom-up pass through the recognition net.

The highly restricted form of $Q(.|\mathbf{d},\phi)$ means that even if we use the optimal recognition weights, the gap between $F(\mathbf{d}|\theta,\phi)$ and $-\log P(\mathbf{d}|\theta)$ is large for some generative models. However, when $F(\mathbf{d}|\theta,\phi)$ is minimized with respect to the generative weights, these models can be avoided.

$F(\mathbf{d}|\theta,\phi)$ can be viewed as the expected number of bits required to communicate a visible vector to a receiver. First we use the recognition model to get a sample from the distribution $Q(.|\mathbf{d},\phi)$. Then, starting at the top layer, we communicate the activities in each layer using the top-down expectations generated from the already communicated activities in the layer above. Using an argument described in [5], it can be shown that the effective number of bits required for communicating the state of each binary unit is:

$$s_k\log\frac{q_k}{p_k} + (1-s_k)\log\frac{1-q_k}{1-p_k} \qquad (9)$$

where s_k is the binary state of unit k, p_k is the top-down probability that $s_k = 1$ and q_k is the bottom-up probability that $s_k = 1$.

There is a very simple online algorithm that minimizes $F(\mathbf{d}|\theta,\phi)$ with respect to the generative weights. We simply use the recognition network to generate a sample from the distribution $Q(.|\mathbf{d},\phi)$ and then we adjust each top-down weight using the delta rule:

$$\Delta\theta_{kj} = \epsilon s_k(s_j - p_j) \qquad (10)$$

where θ_{kj} connects unit k to unit j and ϵ is a learning rate.

It is much more difficult to exactly follow the gradient of $F(\mathbf{d}|\theta,\phi)$ with respect to the recognition weights, but

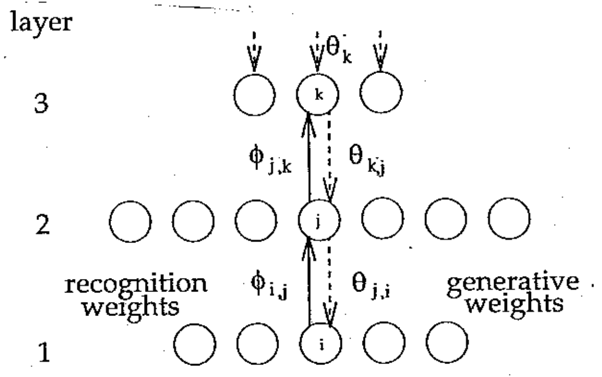

Figure 3: A simple three layer Helmholtz machine modeling the activity of 5 binary inputs (layer 1) using a two stage hierarchical model. Generative weights (θ) are shown as dashed lines, including the generative biases, the only such input to the units in the top layer. Recognition weights (ϕ) are shown with solid lines.

there is a simple approximate method called the "wake-sleep" algorithm [6]. In the "sleep" phase of the learning algorithm, we generate a stochastic sample from the generative model and then we apply the delta rule to increase the log probability that the recognition weights would produce the correct activities in the layer above:

$$\Delta\phi_{ij} = \epsilon s_i(s_j - q_j) \tag{11}$$

AN EASILY VISUALIZED EXAMPLE

To demonstrate that the simple wake-sleep algorithm can build good models of complicated high-dimensional distributions, we applied it to images of handwritten digits that were taken from the zip codes on real pieces of US mail. For each digit class we trained a separate network that had 64 input units for the 8 x 8 binary image and three layers of hidden units containing 16, 16, and 4 hidden units [6]. After training, the network could be run in top-down mode (the "sleep" phase) so it was possible to get a good idea of the model it had extracted simply by looking at the kinds of fantasy it generated. The fantasies generated by the model for a particular digit class were almost always recognizable as instances of that class (see figure 4), and each model captured the variations in the style of its digit quite well.

Once we have a model for the images of a particular digit class, we can use it to discriminate instances of that class. When an image of a different digit class is presented to the model it should find it much harder to code. To test this idea we took some test images that had not been used for training and presented each image to all ten networks. For each network we computed the average description length of the image and we picked the network that produced the shortest description length. This amounts to picking the network that finds the image to be least abnormal. This gave an error rate of 4.8% for these very difficult images which is considerably better than we could achieve using nearest neighbors or standard backpropagation networks. Details are given in [6].

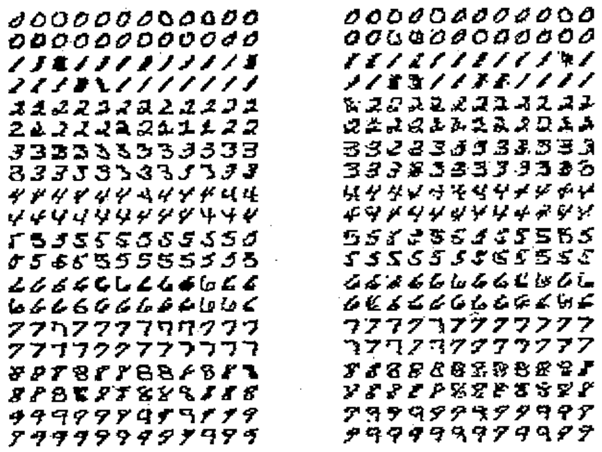

Figure 4: Handwritten digits were normalized and binarized to produce 8 x 8 images. 24 images of each digit are shown on the left. A separate network was trained on 700 examples of each digit class and after training the fantasies produced by these networks are shown on the right. The training of each net required 500 "wake" passes through the 700 examples to train the generative weights, interleaved with the same number of fantasies to train the recognition weights.

CONCLUSION

The fact that we could discriminate digits so well using this approach suggests that models of this kind should also be good for detecting abnormal patterns of sensor readings from complex plants. This in an application area to which neural networks have not yet been widely applied but for which they hold great promise.

Acknowledgements

This research was funded by the Institute for Robotics and Intelligent Systems, the Information Technology Research Center, and NSERC. Hinton is the Noranda fellow of the Canadian Institute for Advanced Research.

References

1. Rumelhart, D. E., Hinton, G. E., and Williams, R. J. "Learning representations by back-propagating errors." *Nature*, **323**, 1986, pp. 533–536.
2. Baum, L. E. and Eagon, J. A. "An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology." *Bulletin of the American Mathematical Society*, **73**, 1967, pp. 360-363.
3. Dayan, P., Hinton, G. E., Neal, R., and Zemel, R. S. "Helmholtz Machines." *Neural Computation*, bf 7, 1995, pp. 1022-1037.
4. Zemel, R. S. and Hinton, G. E. "Learning Population Codes by Minimizing Description Length." *Neural Computation*, **7**, 1995, pp. 549-564.
5. Hinton, G. E. and Zemel, R. S. "Autoencoders, Minimum Description Length, and Helmholtz Free Energy." *Advances in Neural Information Processing Systems 6.* J. D. Cowan, G. Tesauro and J. Alspector (Eds.), Morgan Kaufmann: San Mateo, CA. 1994
6. Hinton, G. E., Dayan, P., Frey, B. J. and Neal, R. "The wake-sleep algorithm for self-organizing neural networks." *Science*, **268**, 1995, pp. 1158-1161.