P. Cisek, T. Drew & J.F. Kalaska (Eds.)
Progress in Brain Research, Vol. 165
ISSN 0079-6123

CHAPTER 34

To recognize shapes, first learn to generate images

Geoffrey E. Hinton*

Department of Computer Science, University of Toronto, 10 Kings College Road, Toronto, M5S 3G4 Canada

Abstract: The uniformity of the cortical architecture and the ability of functions to move to different areas of cortex following early damage strongly suggest that there is a single basic learning algorithm for extracting underlying structure from richly structured, high-dimensional sensory data. There have been many attempts to design such an algorithm, but until recently they all suffered from serious computational weaknesses. This chapter describes several of the proposed algorithms and shows how they can be combined to produce hybrid methods that work efficiently in networks with many layers and millions of adaptive connections.

Keywords: learning algorithms; multilayer neural networks; unsupervised learning; Boltzmann machines; wake–sleep algorithm; contrastive divergence; feature discovery; shape recognition; generative models

Five strategies for learning multilayer networks

Half a century ago, Oliver Selfridge (1958) proposed a pattern recognition system called "Pandemonium" consisting of multiple layers of feature detectors. His actual proposal contained a considerable amount of detail about what each of the layers would compute but the basic idea was that each feature detector would be activated by some familiar pattern of firing among the feature detectors in the layer below. This would allow the layers to extract more and more complicated features culminating in a top layer of detectors that would fire if and only if a familiar object was present in the visual input. Over the next 25 years, many attempts were made to find a learning algorithm that would be capable of discovering appropriate connection strengths (weights) for the feature detectors in every layer. Learning the weights of a single feature detector is quite easy if we are given both the inputs to the feature detector and its desired firing behavior, but learning is much more difficult if we are not told how the intermediate layers of feature detectors ought to behave. These intermediate feature detectors are called "hidden units" because their desired states cannot be observed. There are several strategies for learning the incoming weights of the hidden units.

The first strategy is denial. If we assume that there is only one layer of hidden units, it is often possible to set their incoming weights by hand using domain-specific knowledge. So the problem of learning hidden units does not exist. Within neuroscience, the equivalent of using hand-coded features is to assume that the receptive fields of feature detectors are specified innately — a view that is increasingly untenable (Merzenich et al., 1983; Karni et al., 1994; Sharma et al., 2000). Most of the work on perceptrons (Rosenblatt, 1962; Minsky and Papert, 1969) used hand-coded feature detectors, so learning only occurred for the weights from the feature detectors to the final decision units whose desired states were known. To

*Corresponding author. Tel.: +1 416-978-7564;
Fax: +1 416-978-1455; E-mail: hinton@cs.toronto.edu

DOI: 10.1016/S0079-6123(06)65034-6

be fair to Rosenblatt, he was well aware of the limitations of this approach — he just did not know how to learn multiple layers of features efficiently. The current version of denial is called "support vector machines" (Vapnik, 2000). These come with a fixed, non-adaptive recipe for converting a whole training image into a feature and a clever optimization technique for deciding which training cases should be turned into features and how these features should be weighted. Their main attraction is that the optimization method is guaranteed to find the global minimum. They are inadequate for tasks like 3-D object recognition that cannot be solved efficiently using a single layer of feature detectors (LeCun et al., 2004) but they work undeniably well for many of the simpler tasks that are used to evaluate machine learning algorithms (Decoste and Schoelkopf, 2002).

The second strategy is based on an analogy with evolution — randomly jitter the weights and save the changes that cause the performance of the whole network to improve. This is attractive because it is easy to understand, easy to implement in hardware (Jabri and Flower, 1992) and it works for almost any type of network. But it is hopelessly inefficient when the number of weights is large. Even if we only change one weight at a time, we still have to classify a large number of images to see if that single change helps or hurts. Changing many weights at the same time is no more efficient because the changes in other weights create noise that prevents each weight from detecting what effect it has on the overall performance. The evolutionary strategy can be significantly improved by applying the jitter to the activities of the feature detectors rather than to the weights (Mazzoni et al., 1991; Seung, 2003), but it is still an extremely inefficient way to discover gradients. Even blind evolution must have stumbled across a better strategy than this.

The third strategy is procrastination. Instead of learning feature detectors that are designed to be helpful in solving the classification problem, we can learn a layer of feature detectors that capture interesting regularities in the input images and put off the classification problem until later. This strategy can be applied recursively: we can learn a second layer of feature detectors that capture interesting regularities in the patterns of activation of the first layer detectors, and so on for as many layers as we like. The hope is that the features in the higher layers will be much more useful for classification than the raw inputs or the features in lower layers. As we shall see, this is not just wishful thinking. The main difficulties with the layer-by-layer strategy are that we need a quantitative definition of what it means for a regularity to be "interesting" and we need a way of ensuring that different feature detectors within a layer learn to detect different regularities even if they receive the same inputs.

The fourth strategy is to use calculus. To apply this strategy we need the output of each hidden unit to be a smooth function of the inputs it receives from the layer below. We also need a cost function that measures how poorly the network is performing. This cost function must change smoothly with the weights, so the number of classification errors is not the right function. For classification tasks, we can interpret the outputs of the top-level units as class probabilities and an appropriate cost function is then the cross-entropy, which is the negative log probability that the network assigns to the correct class. Given appropriate hidden units and an appropriate cost function, the chain rule can be used to compute how the cross-entropy changes as each weight in the network is changed. This computation can be made very efficient by first computing, for each hidden unit, how the cross-entropy changes as the activity of that hidden unit is changed. This is known as backpropagation because the computation starts at the output layer and proceeds backwards through the network one layer at a time. Once we know how the activity of a hidden unit affects the cross-entropy on each training case we have a surrogate for the desired state of the hidden unit and it is easy to change the incoming weights to decrease the sum of the cross-entropies on all the training cases. Compared with random jittering of the weights or feature activations, backpropagation is more efficient by a factor of the number of weights or features in the network.

Backpropagation was discovered independently by several different researchers (Werbos, 1974; Bryson and Ho, 1975; LeCun, 1985; Parker, 1985;

Rumelhart et al., 1986) and it was the first effective way to learn neural networks that had one or more layers of adaptive hidden units. It works very well for tasks such as the recognition of handwritten digits (LeCun et al., 1998; Simard et al., 2003), but it has two serious computational problems that will be addressed in this chapter. First, it is necessary to choose initial random values for all the weights. If these values are small, it is very difficult to learn deep networks because the gradients decrease multiplicatively as we backpropagate through each hidden layer. If the initial values are large, we have randomly chosen a particular region of the weight-space and we may well become trapped in a poor local optimum within this region. Second, the amount of information that each training case provides about the mapping between images and classes is at most the log of the number of possible classes. This means that large networks require a large amount of labeled training data if they are to learn weights that generalize well to novel test cases.

Many neuroscientists treat backpropagation with deep suspicion because it is not at all obvious how to implement it in cortex. In particular, it is hard to see how a single neuron can communicate both its activity and the deriviative of the cost function with respect to its activity. It seems very unlikely, however, that hundreds of millions of years of evolution have failed to find an effective way of tuning lower level feature detectors so that they provide the outputs that higher level detectors need in order to make the right decision.

The fifth and last strategy in this survey was designed to allow higher level feature detectors to communicate their needs to lower level ones whilst also being easy to implement in layered networks of stochastic, binary neurons that have activation states of 1 or 0 and turn on with a probability that is a smooth non-linear function of the total input they receive:

$$p(s_j = 1) = \frac{1}{1 + \exp(-b_j - \sum_i s_i w_{ij})} \qquad (1)$$

where s_i and s_j are the binary activities of units i and j, w_{ij} the weight on the connection from i to j and b_j the bias of unit j. Imagine that the training data was generated top-down by a multilayer "graphics" model of the type shown in Fig. 1. The binary state of a hidden unit that was actually used to generate an image top-down could then be used as its desired state when learning the bottom-up "recognition" weights. At first sight, this idea of using top-down "generative" connections to provide desired states for the hidden units does not appear to help because we now have to learn a graphics model that can generate the training data. If, however, we already had some good recognition connections we could use a bottom-up pass from the real training data to activate the units in every layer and then we could learn the generative weights by trying to reconstruct the activities in each layer from the activities in the layer above. So we have a chicken-and-egg problem: given the generative weights we can learn the recognition weights and given the recognition weights we can learn the generative weights. It turns out that we can learn both sets of weights by starting with small random values and alternating between two phases of learning. In the "wake" phase, the recognition weights are used to drive the units bottom-up, and the binary states of units in adjacent layers can then be used to train the generative

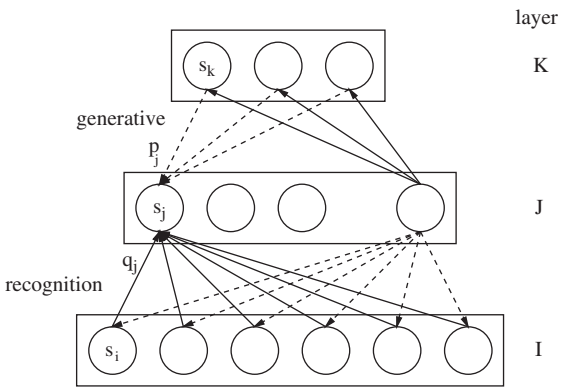

Fig. 1. This shows a three-layer neural network. Activities in the bottom layer represent the sensory input and activities in higher layers learn to represent the causes of the sensory input. The bottom-up "recognition" connections convert the sensory input into an internal representation. They can be trained by assuming that they are trying to invert a generative process (like computer graphics) that converts hidden causes into sensory data. The assumed generative process is represented in the top-down "generative" connections and it too is learned just by observing sensory data.

weights. In the "sleep" phase, the top-down generative connections are used to drive the network, so it produces fantasies from its generative model. The binary states of units in adjacent layers can then be used to learn the bottom-up recognition connections (Hinton et al., 1995). The learning rules are very simple. During the wake phase, a generative weight, g_{kj}, is changed by

$$\Delta g_{kj} = \varepsilon s_k (s_j - p_j) \tag{2}$$

where unit k is in the layer above unit j, ε a learning rate and p_j the probability that unit j would turn on if it were being driven by the current states of the units in the layer above using the current generative weights. During the sleep phase, a recognition weight, w_{ij}, is changed by

$$\Delta w_{ij} = \varepsilon s_i (s_j - q_j) \tag{3}$$

where q_j is the probability that unit j would turn on if it were being driven by the current states of the units in the layer below using the current recognition weights.

The rest of this chapter shows that the performance of both backpropagation (strategy four) and the "wake–sleep" algorithm (strategy five) can be greatly improved by using a "pretraining" phase in which unsupervised layer-by-layer learning is used to make the hidden units in each layer represent regularities in the patterns of activity of units in the layer below (strategy three). With this type of pretraining, it is finally possible to learn deep, multilayer networks efficiently and to demonstrate that they are better at classification than shallow methods.

Learning feature detectors with no supervision

Classification of isolated, normalized shapes like those shown in Fig. 2 has been one of the paradigm tasks for demonstrating the pattern recognition abilities of artificial neural networks. The connection weights in a multilayer network are typically initialized by using small random values, which are then iteratively adjusted by backpropagating the difference between the desired output of the network on each training case and the output that it actually produces on that training case. To prevent the network from modeling accidental regularities that arise from the random selection of training examples, it is common to stop the training early or to impose a penalty on large

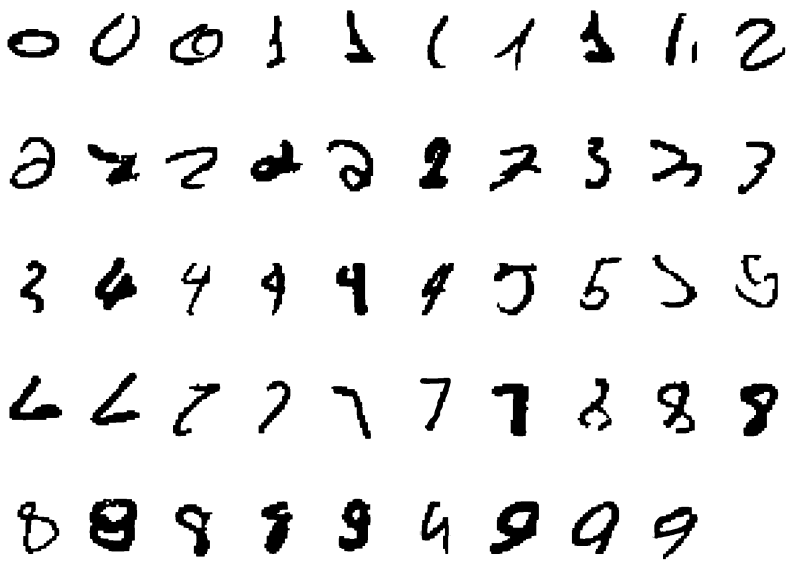

Fig. 2. Some examples of real handwritten digits from the MNIST test set that are hard to recognize. A neural network described at the end of this chapter gets all of these examples right, even though it has never seen them before. However, it is not confident about its classification for any of these examples. The true classes are arranged in standard scan order.

connection weights. This improves the final performance of the network on a test set, but it is not nearly as effective as using a more intelligent strategy for initializing the weights.

A discriminative training procedure like backpropagation ignores the structure in the input and only tries to model the way in which the output depends on the input. This is a bad idea if the input contains a lot of structure that can be modeled by latent variables and the output is a class label that is more simply related to these latent variables than it is to the raw input. Consider, for example a set of images of a dog. Latent variables such as the position, size, shape and color of the dog are a good way of explaining the complicated, higher order correlations between the individual pixel intensities, and some of these latent variables are very good predictors of the class label. In cases like this, it makes sense to start by using unsupervised learning to discover latent variables (i.e. features) that model the structure in the ensemble of training images. Once a good set of features has been found using unsupervised learning, discriminative learning can then be used to model the dependence of the class label on the features and to fine-tune the features so that they work better for discrimination. The features are then determined mainly by the input images, which contain a lot of information, and only slightly by the labels which typically contain much less information.

Learning one layer of feature detectors

Images composed of binary pixels can be modeled by using a "Restricted Boltzmann machine" (RBM) that uses a layer of binary feature detectors to model the higher order correlations between pixels. If there are no direct interactions between the feature detectors and no direct interactions between the pixels, there is a simple and efficient way to learn a good set of feature detectors from a set of training images (Hinton, 2002). We start with zero weights on the symmetric connections between each pixel i and each feature detector j. Then we repeatedly update each weight, w_{ij}, using the difference between two measured, pairwise correlations

$$\Delta w_{ij} = \varepsilon(\langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle_{\text{recon}}) \tag{4}$$

where ε is a learning rate, $\langle s_i s_j \rangle_{\text{data}}$ the frequency with which pixel i and feature detector j are on together when the feature detectors are being driven by images from the training set and $\langle s_i s_j \rangle_{\text{recon}}$ the corresponding frequency when the feature detectors are being driven by reconstructed images. A similar learning rule can be used for the biases.

Given a training image, we set the binary state, s_j, of each feature detector to be 1 with probability

$$p(s_j = 1) = \frac{1}{1 + \exp(-b_j - \sum_{i \in \text{pixels}} s_i w_{ij})} \tag{5}$$

where b_j is the bias of j and s_i the binary state of pixel i. Once binary states have been chosen for the hidden units we produce a "reconstruction" of the training image by setting the state of each pixel to be 1 with probability

$$p(s_i = 1) = \frac{1}{1 + \exp(-b_i - \sum_{j \in \text{features}} s_j w_{ij})} \tag{6}$$

On 28×28 pixel images of handwritten digits like those shown in Fig. 2, good features can be found by using 100 passes through a training set of 50,000 images, with the weights being updated after every 100 images using the pixel-feature correlations measured on those 100 images and their reconstructions. Figure 3 shows a randomly selected subset of the features that are learned. We will use the letters A–F to refer to the rows of this figure and the numbers 1–10 to refer to the columns. Some features have an on-center off-surround structure (e.g. B3) or the reverse (A5). These features are a good way to model the simple fact that if a pixel is on, nearby pixels tend to be on. Some features detect parts of strokes (A9), and they typically inhibit the region of the image that is further from the center than the stroke fragment. Some features, which look more like fingerprints (D2), encode the phase and amplitude of high-frequency Fourier components of a large part of the whole image. These features tend to turn on about half the time and can be eliminated by forcing features to only turn on rarely (Ranzato et al., 2007). The three features with unnaturally sharp black lines (A4, D9, E4) capture the fact that if a pixel is on, pixels that are more than 20 rows above or below it cannot be on because of the way the data was normalized.

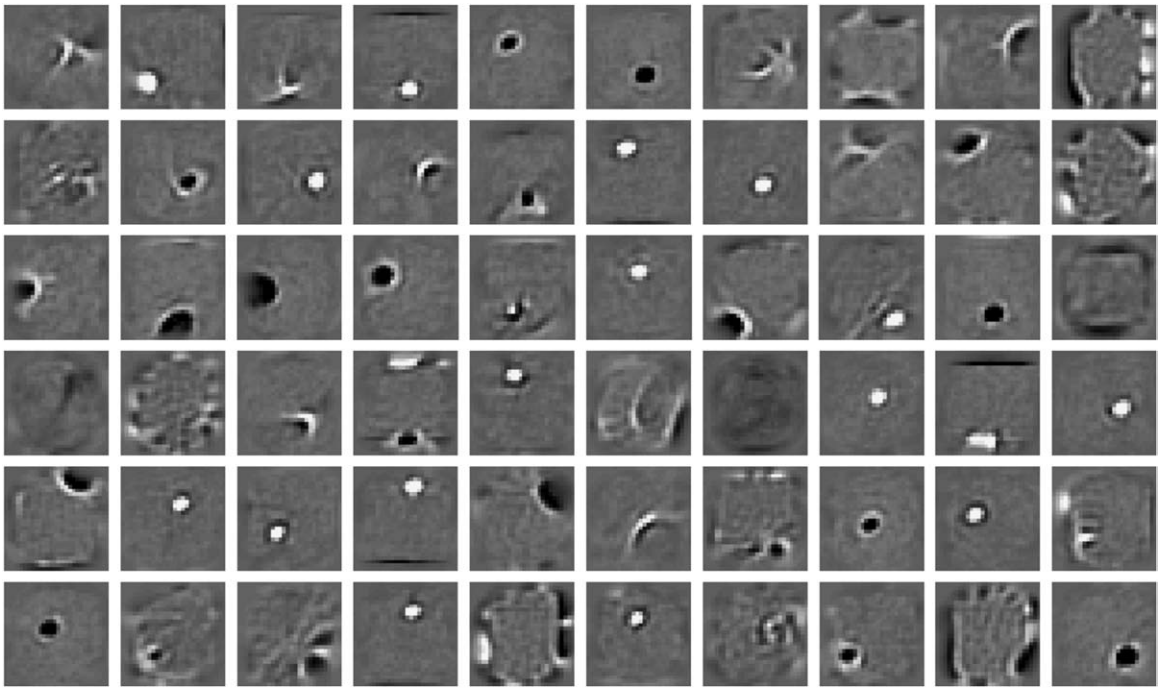

Fig. 3. The receptive fields of some feature detectors. Each gray square shows the incoming weights to one feature detector from all the pixels. Pure white means a positive weight of at least 3 and pure black means a negative weight of at least −3. Most of the feature detectors learn highly localized receptive fields.

The learned weights and biases of the features implicitly define a probability distribution over all possible binary images. Sampling from this distribution is difficult, but it can be done by using "alternating Gibbs sampling." This starts with a random image and then alternates between updating all of the features in parallel using Eq. (5) and updating all of the pixels in parallel using Eq. (6). After Gibbs sampling for sufficiently long, the network reaches "thermal equilibrium." The states of pixels and features detectors still change, but the probability of finding the system in any particular binary configuration does not.

A greedy learning algorithm for multiple hidden layers

A single layer of binary features is not the best way to model the structure in a set of images. After learning the first layer of feature detectors, a second layer can be learned in just the same way by treating the existing feature detectors, when they are being driven by training images, as if they were data. To reduce noise in the learning signal, the binary states of feature detectors (or pixels) in the "data" layer are replaced by their real-valued probabilities of activation when learning the next layer of feature detectors, but the new feature detectors have binary states to limit the amount of information they can convey. This greedy, layer-by-layer learning can be repeated as many times as desired. To justify this layer-by-layer approach, it would be good to show that adding an extra layer of feature detectors always increases the probability that the overall generative model would generate the training data. This is almost true: provided the number of feature detectors does not decrease and their weights are initialized correctly, adding an extra layer is guaranteed to raise a lower bound on the log probability of the training data (Hinton et al., 2006). So after learning several layers there is good reason to believe that the feature detectors will have captured many of the statistical

regularities in the set of training images and we can now test the hypothesis that these feature detectors will be useful for classification.

Using backpropagation for discriminative fine-tuning

After greedily learning layers of 500, 500 and 2000 feature detectors without using any information about the class labels, gentle backpropagation was used to fine-tune the weights for discrimination. This produced much better classification performance on test data than using backpropagation without the initial, unsupervised phase of learning. The MNIST dataset used for these experiments has been used as a benchmark for many years and many different researchers have tried using many different learning methods, including variations of backpropagation in nets with different numbers of hidden layers and different numbers of hidden units per layer.

There are several different versions of the MNIST learning task. In the most difficult version, the learning algorithm is not given any prior knowledge of the geometry of images and it is forbidden to increase the size of the training set by using small affine or elastic distortions of the training images. Consequently, if the same random permutation is applied to the pixels of every training and test image, the performance of the learning algorithm will be unaffected. For this reason, this is called the "permutation invariant" version of the task. So far as the learning algorithm is concerned, each 28×28 pixel image is just a vector of 784 numbers that has to be given one of 10 labels. The best published backpropagation error rate for this version of the task is 1.6% (Simard et al., 2003). Support vector machines can achieve 1.4% (Decoste and Schoelkopf, 2002). Table 1 shows that the error rate of backpropagation can be reduced to about 1.12% if it is only used for fine-tuning features that are originally discovered by layer-by-layer pretraining.

Details of the discriminative fine-tuning procedure

Using three different splits of the 60,000 image training set into 50,000 training examples and 10,000 validation examples, the greedy learning algorithm was used to initialize the weights and biases and gentle backpropagation was then used to fine-tune the weights. After each sweep through the training set (which is called an "epoch"), the classification error rate was measured on the validation set. Training was continued until two conditions were satisfied. The first condition involved the average cross-entropy error on the validation set. This is the quantity that is being minimized by the learning algorithm so it always falls on the training data. On the validation data, however, it starts rising as soon as overfitting occurs. There is a strong tendency for the number of classification errors to continue to fall after the cross-entropy has bottomed-out on the validation data, so the

Table 1. Neta, Netb and Netc were greedily pretrained on different, unlabeled, subsets of the training data that were obtained by removing disjoint validation sets of 10,000 images

Pretrained Network	Backpropagation training Set size	Train Epochs	Train cost Per 100	Train Errors	Valid. cost Per 100	Valid. errors In 10^4	Test cost Per 100	Test errors In 10^4
Neta	50,000	33	0.12	1	6.49	129	6.22	122
Netb	50,000	56	0.04	0	7.81	118	6.21	116
Netc	50,000	63	0.03	0	8.12	118	6.73	124
Combined							5.75	110
Neta	60,000	33 + 16	<0.12	1			5.81	113
Netb	60,000	56 + 28	<0.04	0			5.90	106
Netc	60,000	63 + 31	<0.03	0			5.93	118
Combined							5.40	106
Not pretrained	60,000	119	<0.063	0			18.43	227

Note: After pretraining, they were trained on those same subsets using backpropagation. Then the training was continued on the full training set until the cross-entropy error reached the criterion explained in the text.

first condition is that the learning must have already gone past the minimum of the cross-entropy on the validation set. It is easy to detect when this condition is satisfied because the cross-entropy changes very smoothly during the learning. The second condition involved the number of errors on the validation set. This quantity fluctuates unpredictably, so the criterion was that the minimum value observed so far should have occurred at least 10 epochs ago. Once both conditions were satisfied, the weights and biases were restored to the values they had when the number of validation set errors was at its minimum, and performance on the 10,000 test cases was measured. As shown in Table 1, this gave test error rates of 1.22, 1.16 and 1.24% on the three different splits. The fourth line of the table shows that these error rates can be reduced to 1.10% by multiplying together the three probabilities that the three nets predict for each digit class and picking the class with the maximum product.

Once the performance on the validation set has been used to find a good set of weights, the cross-entropy error on the *training* set is recorded. Performance on the test data can then be further improved by adding the validation set to the training set and continuing the training until the cross-entropy error on the expanded training set has fallen to the value it had on the original training set for the weights selected by the validation procedure. As shown in Table 1 this eliminates about 8% of the errors. Combining the predictions of all three models produces less improvement than before because each model has now seen all of the training data. The final line of Table 1 shows that backpropagation in this relatively large network gives much worse results if no pretraining is used. For this last experiment, the stopping criterion was set to be the average of the stopping criteria from the previous experiments.

To avoid making large changes to the weights found by the pretraining, the backpropagation stage of learning used a very small learning rate which made it very slow, so a new trick was introduced which sped up the learning by about a factor of three. Most of the computational effort is expended computing the almost non-existent gradients for "easy" training cases that the network can already classify confidently and correctly. It is tempting to make a note of these easy cases and then just ignore them, checking every few epochs to see if the cross-entropy error on any of the ignored cases has become significant. This can be done without changing the expected value of the overall gradient by using a method called importance sampling. Instead of being completely ignored, easy cases are selected with a probability of 0.1, but when they are selected, the computed gradients are multiplied by 10. Using more extreme values like 0.01 and 100 is dangerous because a case that used to be easy might have developed a large gradient while it was being ignored, and multiplying this gradient by 100 could give the network a shock. When using importance sampling, an "epoch" was redefined to be the time it takes to sample as many training cases as the total number in the training set. So an epoch typically involves several sweeps through the whole set of training examples, but it is the same amount of computation as one sweep without importance sampling.

After the results in Table 1 were obtained using the rather complicated version of backpropagation described above, Ruslan Salakhutdinov discovered that similar results can be obtained using a standard method called "conjugate gradient" which takes the gradients delivered by backpropagation and uses them in a more intelligent way than simply changing each weight in proportion to its gradient (Hinton and Salakhutdinov, 2006). The MNIST data together with the Matlab code required for pretraining and fine-tuning the network are available at http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html.

Using extra unlabeled data

Since the greedy pretraining algorithm does not require any labeled data, it should be a very effective way to make use of unlabeled examples to improve performance on a small labeled dataset. Learning with only a few labeled examples is much more characteristic of human learning. We see many instances of many different types of object, but we are very rarely told the name of an object.

Preliminary experiments confirm that pretraining on unlabeled data helps a lot, but for a proper comparison it will be necessary to use networks of the appropriate size. When the number of labeled examples is small, it is unfair to compare the performance of a large network that makes use of unlabeled examples with a network of the same size that does not make use of the unlabeled examples.

Using geometric prior knowledge

The greedy pretraining improves the error rate of backpropagation by about the same amount as methods that make use of prior knowledge about the geometry of images, such as weight-sharing (LeCun et al., 1998) or enlarging the training set by using small affine or elastic distortions of the training images. But pretraining can also be combined with these other methods. If translations of up to two pixels are used to create 12 extra versions of each training image, the error rate of the best support vector machine falls from 1.4% to 0.56% (Decoste and Schoelkopf, 2002). The average error rate of the pretrained neural net falls from 1.12% to 0.65%. The translated data is presumably less helpful to the multilayer neural net because the pretraining can already capture some of the geometrical structure even without the translations. The best published result for a single method is currently 0.4%, which was obtained using backpropagation in a multilayer neural net that uses *both* weight-sharing and sophisticated, elastic distortions (Simard et al., 2003). The idea of using unsupervised pretraining to improve the performance of backpropagation has recently been applied to networks that use weight-sharing and it consistently reduces the error rate by about 0.1% even when the error rate is already very low (Ranzato et al., 2007).

Using contrastive wake–sleep for generative fine-tuning

Figure 4 shows a multilayer generative model in which the top two layers interact via undirected connections and form an associative memory. At the start of learning, all configurations of this

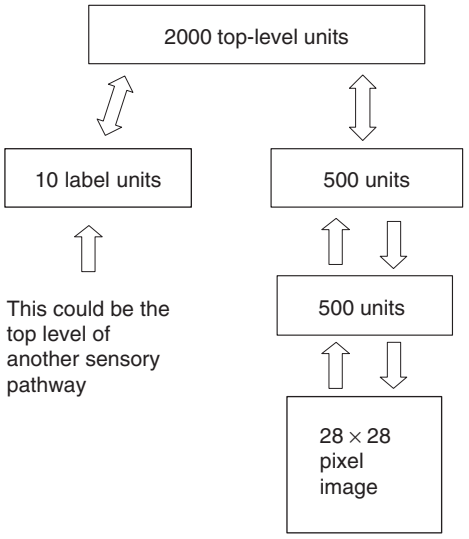

Fig. 4. A multilayer neural network that learns to model the joint distribution of digit images and digit labels. The top two layers have symmetric connections and form an associative memory. The layers below have directed, top-down, generative connections that can be used to map a state of the associative memory to an image. There are also directed, bottom-up, recognition connections that are used to infer a factorial representation in one layer from the binary activities in the layer below.

top-level associative memory have roughly equal energy. Learning sculpts the energy landscape and after learning, the associative memory will settle into low-energy states that represent images of digits. Valleys in the high-dimensional energy-landscape represent digit classes. Directions along the valley floor represent the allowable variations of a digit and directions up the side of a valley represent implausible variations that make the image surprising to the network. Turning on one of the 10 label units lowers one whole valley and raises the other valleys. The number of valleys and the dimensionality of each valley floor are determined by the set of training examples.

The states of the associative memory are just binary activity vectors that look nothing like the images they represent, but it is easy to see what the associative memory has in mind. First, the 500 hidden units that form part of the associative memory are used to stochastically activate some of the units in the layer below via the top-down, generative connections. Then these activated units

are used to provide top-down input to the pixels. Figure 5 shows some fantasies produced by the trained network when the top-level associative memory is allowed to wander stochastically between low-energy states, but with one of the label units clamped so that it tends to stay in the same valley. The fact that it can generate a wide variety of slightly implausible versions of each type of digit makes it very good at recognizing poorly written digits. A demonstration that shows the network generating and recognizing digit images is available at http://www.cs.toronto.edu/hinton/digits.html. In this chapter, each training case consists of an image and an explicit class label, but the same learning algorithm can be used if the "labels" are replaced by a multilayer pathway whose inputs are spectrograms from multiple different speakers saying isolated digits (Kaganov et al., 2007). The network then learns to generate pairs that consist of an image and a spectrogram of the same digit class.

The network was trained in two stages — pretraining and fine-tuning. The layer-by-layer pretraining was the same as in the previous section, except that when training the top layer of 2000 feature detectors, each "data" vector had 510 components. The first 500 were the activation probabilities of the 500 feature detectors in the penultimate layer and the last 10 were the label values. The value of the correct label was set to 1 and the remainder were set to 0. So the top layer of feature detectors learns to model the joint distribution of the 500 penultimate features and the 10 labels.

At the end of the layer-by-layer pretraining, the weight between any two units in adjacent layers is the same in both directions and we can view the result of the pretraining as a set of three different RBMs whose only interaction is that the data for the higher RBMs is provided by the feature activations of the lower RBMs. It is possible, however, to take a very different view of exactly the same system (Hinton et al., 2006). We can view it as a single generative model that generates data by first letting the top-level RBM settle to thermal equilibrium, which may take a very long time, and then performing a single top-down pass to convert the 500 binary feature activations in the penultimate layer into an image. When it is viewed as a single generative model, the weights between the top two

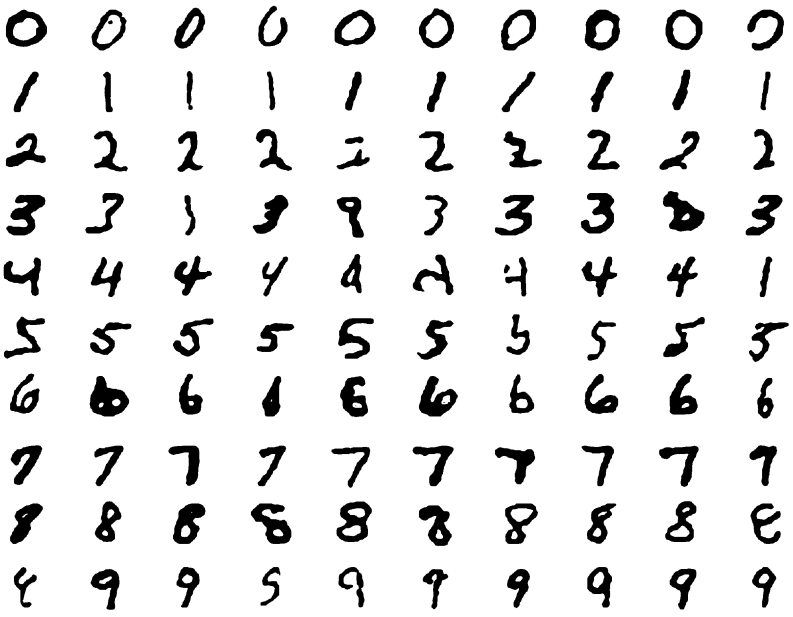

Fig. 5. Each row shows 10 samples from the generative model with a particular label clamped on. The top-level associative memory is run for 1000 iterations of alternating Gibbs sampling between samples.

layers need to be symmetric, but the weights between lower layers do not. In the top-down, generative direction, these weights form part of the overall generative model, but in the bottom-up, recognition direction they are not part of the model. They are merely an efficient way of inferring what hidden states probably caused the observed image. If the whole system is viewed as a single generative model, we can ask whether it is possible to fine-tune the weights produced by the pretraining to make the overall generative model more likely to generate the set of image-label pairs in the training data. The answer is that the generative model can be significantly improved by using a contrastive form of the wake–sleep algorithm. In the lower layers, this makes the recognition weights differ from the generative weights. In addition to improving the overall generative model, the generative fine-tuning makes the model much better at assigning labels to test images using a method, which will be described later.

In the standard wake–sleep algorithm, the network generates fantasies by starting with a pattern of activation of the top-level units that is chosen stochastically using only the generative bias of each top-level unit to influence its probability of being on. This way of initiating fantasies cannot be used if the top two layers of the generative model form an associative memory because it will not produce samples from the generative model. The obvious alternative is to use prolonged Gibbs sampling in the top two layers to sample from the energy landscape defined by the associative memory, but this is much too slow. A very effective alternative is to use the bottom-up recognition connections to convert a image-label pair from the training set into a state of the associative memory and then to perform brief alternating Gibbs sampling which allows the associative memory to produce a "confabulation" that it prefers to its initial representation of the training pair. The top-level associative memory is then trained as an RBM by using Eq. (4) to lower the energy of the initial representation of the training pair and raise the energy of the confabulation. The confabulation in the associative memory is also used to drive the system top-down, and the states of all the hidden units that are produced by this generative, top-down pass are used as targets to train the bottom-up recognition connections. The "wake" phase is just the same as in the standard wake–sleep algorithm: After the initial bottom-up pass, the top-down, generative connections in the bottom two layers are trained, using Eq. (2), to reconstruct the activities in the layer below from the activities in the layer above. The details are given in Hinton et al. (2006).

Fine-tuning with the contrastive wake–sleep algorithm is about an order of magnitude slower than fine-tuning with backpropagation, partly because it has a more ambitious goal. The network shown in Fig. 4 takes a week to train on a 3 GHz machine. The examples shown in Fig. 2 were all classified correctly by this network which gets a test error rate of 1.25%. This is slightly worse than pretrained networks with the same architecture that are fine-tuned with backpropagation, but it is better than the 1.4% achieved by the best support vector machine on the permutation-invariant version of the MNIST task. It is rare for a generative model to outperform a good discriminative model *at discrimination*.

There are several different ways of using the generative model for discrimination. If time was not an issue, it would be possible to use sampling methods to measure the relative probabilities of generating each of the ten image-label pairs that are obtained by pairing the test image with each of the 10 possible labels. A fast and accurate approximation can be obtained by first performing a bottom-up pass in which the activation probabilities of the first layer of hidden units are used to compute activation probabilities for the penultimate hidden layer. Using probabilities rather than stochastic binary states suppresses the noise due to sampling. Then the vector of activation probabilities of the feature detectors in the penultimate layer is paired with each of the 10 labels in turn and the "free energy" of the associative memory is computed. Each of the top-level units contributes additively to this free energy, so it is easy to calculate exactly (Hinton et al., 2006). The label that gives the lowest free-energy is the network's guess.

Fitting a generative model constrains the weights of the network far more strongly than fitting a discriminative model, but if the ultimate objective is discrimination, it also wastes a lot of the

discriminative capacity. This waste shows up in the fact that after fine-tuning the generative model, its discriminative performance on the training data is about the same as its discriminative performance on the test data — there is almost no overfitting. This suggests one final experiment. After first using contrastive wake–sleep for fine-tuning, further fine-tuning can be performed using a weighted average of the gradients computed by backpropagation and by contrastive wake–sleep. Using a validation set, the coefficient controlling the contribution of the backpropagation gradient to the weighted average was gradually increased to find the coefficient value at which the error rate on the validation set was minimized. Using this value of the coefficient, the test error rate was 0.97%, which is the current record for the permutation-invariant MNIST task. It is also possible to combine the gradient from backpropagation with the gradient computed by the pretraining (Bengio et al., 2007). This is much less computational effort than using contrastive wake–sleep, but does not perform as well.

Acknowledgments

I thank Yoshua Bengio, Yann LeCun, Peter Dayan, David MacKay, Sam Roweis, Terry Sejnowski, Max Welling and my past and present graduate students for their numerous contributions to these ideas. The research was supported by NSERC, CFI and OIT. GEH is a fellow of the Canadian Institute for Advanced Research and holds a Canada Research Chair in Machine Learning.

References

Bengio, Y., Lamblin, P., Popovici, D. and Larochelle, H. (2007) Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems, 19. MIT Press, Cambridge, MA, pp. 153–160.

Bryson, A. and Ho, Y. (1975) Applied optimal control. Wiley, New York.

Decoste, D. and Schoelkopf, B. (2002) Training invariant support vector machines. Machine Learn., 46: 161–190.

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

Hinton, G.E., Dayan, P., Frey, B.J. and Neal, R. (1995) The wake-sleep algorithm for self-organizing neural networks. Science, 268: 1158–1161.

Hinton, G.E., Osindero, S. and Teh, Y.W. (2006) A fast learning algorithm for deep belief nets. Neural Comput., 18: 1527–1554.

Hinton, G.E. and Salakhutdinov, R.R. (2006) Reducing the dimensionality of data with neural networks. Science, 313: 504–507.

Jabri, M. and Flower, B. (1992) Weight perturbation: an optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks. IEEE Trans. Neural Netw., 3(1): 154–157.

Kaganov, A., Osindero, S. and Hinton, G.E. (2007) Learning the relationship between spoken digits and digit images. In: Technical Report, Department of Computer Science, University of Toronto.

Karni, A., Tanne, D., Rubenstein, B., Askenasy, J. and Sagi, D. (1994) Dependence on REM sleep of overnight improvement of a perceptual skill. Science, 265(5172): 679.

LeCun, Y. (1985) Une procédure d'apprentissage pour réseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks). In: Proceedings of Cognitiva 85, Paris, France, pp. 599–604.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) Gradient based learning applied to document recognition. Proc. IEEE, 86(11): 2278–2324.

LeCun, Y., Huang, F.-J. and Bottou, L. (2004) Learning methods for generic object recognition with invariance to pose and lighting. In: Proceedings of CVPR'04. IEEE Press, New York.

Mazzoni, P., Andersen, R. and Jordan, M. (1991) A more biologically plausible learning rule for neural networks. Proc. Natl. Acad. Sci., 88(10): 4433–4437.

Merzenich, M., Kaas, J., Wall, J., Nelson, R., Sur, M. and Felleman, D. (1983) Topographic reorganization of somatosensory cortical areas 3b and 1 in adult monkeys following restricted deafferentation. Neuroscience, 8(1): 33–55.

Minsky, M. and Papert, S. (1969) Perceptrons: an introduction to computational geometry. MIT Press, Cambridge, MA.

Parker, D. (1985) Learning logic. In: Technical Report TR-47. Center for Computational Research in Economics and Management Science. Massachusetts Institute of Technology, Cambridge, MA.

Ranzato, M., Poultney, C., Chopra, S. and LeCun, Y. (2007) Efficient learning of sparse representations with an energy-based model. In: Advances in Neural Information Processing Systems 17. MIT Press, Cambridge, MA.

Rosenblatt, F. (1962) Principles of Neurodynamics. Spartan Books, New York.

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

Selfridge, O.G. (1958) Pandemonium: a paradigm for learning. In: Mechanisation of thought processes: Proceedings of a symposium held at the National Physical Laboratory. HMSO, London.

Seung, H. (2003) Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. Neuron, 40(6): 1063–1073.

Sharma, J., Angelucci, A. and Sur, M. (2000) Induction of visual orientation modules in auditory cortex. Nature, 404(6780): 841–847.

Simard, P.Y., Steinkraus, D. and Platt, J. (2003) Best practice for convolutional neural networks applied to visual document analysis. In: International Conference on Document Analysis and Recogntion (ICDAR). IEEE Computer Society, Los Alamitos, pp. 958–962.

Vapnik, V.N. (2000) The Nature of Statistical Learning Theory. Springer, New York.

Werbos, P. (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University.