8 Unsupervised learning

In the previous learning problems we had training examples with feature vectors $\mathbf{x}$ and labels $\mathbf{y}$. In this chapter we discuss unsupervised learning problems in which no labels are given. Training on unlabeled examples restricts the type of learning that can be done, but unsupervised learning has important applications and even can be an important part in aiding supervised learning. Unsupervised does not mean that the learning is not guided at all; the learning follows specific principles that are used to organize the system based on the characteristics provided by the data. We will discuss several examples in this chapter.

8.1 K-means clustering

The first example is **data clustering**. In this problem domain we are given unlabelled data described by a set of features and asked to put them into k categories. In the first example of such clustering we categorize the data by proximity to a mean value. That is, we assume a model that specifies a mean feature value of the data and classifies the data based on the proximity to the mean value. Of course, we do not know this mean value for each class. The idea of the following algorithm is that we start with a guess for this mean value and label the data accordingly. We then use the labeled data from this hypothesis to improve the model by calculating a new mean value, and repeat these steps until convergence is reached. Such an algorithm usually converges quickly to a stable solution. More formally, given a training set of data points $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ and a hypothesis of the number of clusters, k, the k-means clustering algorithm is shown in Table **??**. An example is shown in Figure 8.1.

1. Initialize the means $\mu_1, ...\mu_k$ randomly.
2. Repeat until convergence: $\{$

Model prediction:

For each data point i, classify data to class with closest mean

$c^{(i)} = arg\min_j ||x^{(i)} - \mu_j||$

Model refinement:

Calculate new means for each class

$\mu_j = \frac{\mathbb{1}(c^{(i)}=j)x^{(i)}}{\mathbb{1}(c^{(i)}=j)}$

$\}$ convergence

where the function $\mathbb{1}()(c^{(i)} = j)$ indicates if the i-th example point belongs to class j,

$$\mathbb{1}(c^{(i)} = j) = \begin{cases} 1 & \text{if } c^{(i)} = j, \\ 0 & \text{otherwise.} \end{cases} \tag{8.1}$$

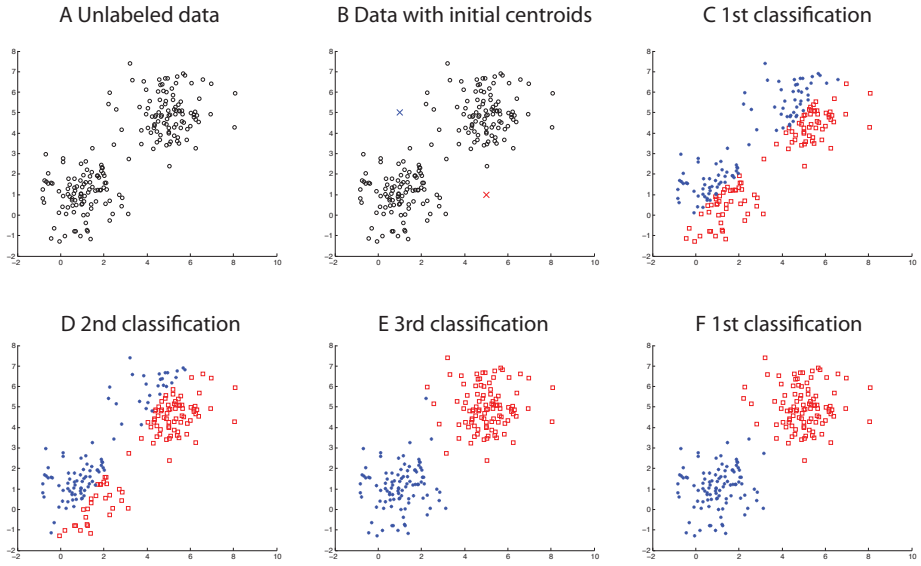

Fig. 8.1 Example of k-means clustering with two clusters.

8.2 Mixture of Gaussian and the EM algorithm

The K-means clustering is an example of unsupervised classification as we calculate class labels from examples without class labels. The crucial ingredient is however a "parameterized model" that states in this case that we expect class distributions where the density depend on the Euclidean distance to a class mean. We can make this a bit more systematic for any probabilistic generative model, though we will illustrate the idea with a simple Gaussian model that follows closely our previous supervised example of linear discriminant analysis. In the case of the linear discriminant analysis, we assumed that each class is Gaussian distributed, and we used examples with labels to determine the parameters of the generative models.

The principle idea behind the class of unsupervised learning algorithms call **Expectation Maximization (EM)** is that we can make some random choice of the parameters for the generative models and use these specific models to calculate an expected label (E-step) and then use these predicted labels to maximize likelihood of the parameters of the models (M-step).

More formally, let us assume we have k Gaussian classes, where each class is chosen randomly from a multinominal distribution,

$$p(z^{(i)} = j) \propto \text{multinomial}(\Phi_j) \tag{8.2}$$

$$p(x^{(i)}|z^{(i)} = j) \propto N(\mu_j, \Sigma_j) \tag{8.3}$$

This is called a **Gaussian Mixture Model**. The corresponding log-likelihood function is

$$l(\Phi, \mu, \sigma) = \sum_{i=1}^{m} \log \sum_{z^{(i)}=1}^{k} p(x^{(i)}|z^{(i)}; \mu, \Sigma) p(z^{(i)}; \Phi). \tag{8.4}$$

Since we consider here unsupervised learning in which we are given data without labels, the random variables $z^{(i)}$ are latent variables. This makes the problem hard. If we would be give the class membership, than the log-likelihood would be

$$l(\Phi, \mu, \sigma) = \sum_{i=1}^{m} \log p(x^{(i)}; z^{(i)}, \mu, \Sigma), \tag{8.5}$$

which we could use to calculate the maximum likelihood estimates of the parameter (see equations 7.32-7.34),

$$\phi_k = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}(z^{(i)} = j) \tag{8.6}$$

$$\mu_k = \frac{\sum_{i=1}^{m} \mathbb{1}(z^{(i)} = j)\mathbf{x}^{(i)}}{\sum_{i=1}^{m} \mathbb{1}(z^{(i)} = j)} \tag{8.7}$$

$$\Sigma_k = \frac{\sum_{i=1}^{m} \mathbb{1}(z^{(i)} = j)(x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} \mathbb{1}(y^{(i)} = k)}. \tag{8.8}$$

While we do not know the class labels, we can follow a similar strategy to the k-means clustering algorithm and just propose some labels and use them to estimate the parameters. We can then use the new estimate of the distributions to find better labels for the data, and repeat this procedure until a stable configuration is reached. In general, this strategy is called the **EM algorithm** for expectation-maximization. The algorithm is outlined in Fig.8.2. In this version we do not hard classify the data into one or another class, but we take a more soft classification approach that considers the probability estimate of a data point belonging to each class.

1. Initialize parameters ϕ, μ, Σ randomly.
2. Repeat until convergence: $\{$

E step:

For each data point i and class j (soft-)classify data as

$w_j^{(i)} = p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$

M step:

Update the parameters according to

$\phi_j = \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)}$

$\mu_j = \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}}$

$\Sigma_k = \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} \mathbb{1} w_j^{(i)}}$.

$\}$ convergence

Fig. 8.2 EM algorithm

An example is shown in Fig. 8.3. In this simple world, data are generated with equal likelihood from two Gaussian distributions, one with mean $\mu_1 = -1$ and standard deviation $\sigma_1 = 2$, the other with mean $\mu_2 = 4$ and standard deviation $\sigma_2 = 0.5$. These two distributions are illustrated in Fig. 8.3A with dashed lines. Let us assume that we

know that the world consists only of data from two Gaussian distributions with equal likelihood, but that we do not know the specific realizations (parameters) of these distributions. The pre-knowledge of two Gaussian distributions encodes a specific **hypothesis** which makes up this **heuristic model**. In this simple example, we have chosen the heuristics to match the actual data-generating system (world), that is, we have explicitly used some knowledge of the world.

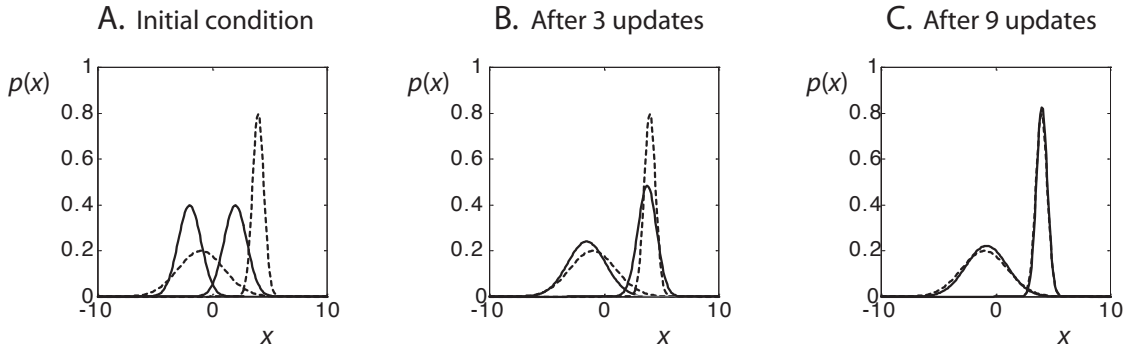

Fig. 8.3 Example of the expectation maximization (EM) algorithm for a world model with two Gaussian distributions. The Gaussian distributions of the world data (input data) are shown with dashed lines. (A) The generative model, shown with solid lines, is initialized with arbitrary parameters. In the EM algorithm, the unlabelled input data are labelled with a recognition model, which is, in this example, the inverse of the generative model. These labelled data are then used for parameter estimation of the generative model. The results of learning are shown in (B) after three iterations, and in (C) after nine iterations .

Learning the parameters of the two Gaussians would be easy if we had access to the information about which data point was produced by which Gaussian, that is, which cause produced the specific examples. Unfortunately, we can only observe the data without a teacher label that could supervise the learning. We choose therefore a self-supervised strategy, which repeats the following two steps until convergence:

E-step: We make assumptions of training labels from the current model (expectation step)

M-step: use this hypothesis to update the parameters of the model to maximize the probability of the observations (maximization step).

Since we do not know appropriate parameters yet, we just choose some arbitrary values as the starting point. In the example shown in Fig. 8.3A we used $\mu_1 = 2$, $\mu_2 = -2$, $\sigma_1 = \sigma_2 = 1$. These distributions are shown with solid lines. Comparing the generated data with the environmental data corresponds to hypothesis testing.

The results are not yet very satisfactory, but we can use the generative model to express our **expectation** of the data. Specifically, we can assign each data point to the class which produces the larger probability within the current world model. Thus, we are using our specific hypothesis here as a **recognition model**. In the example we can use Bayes' rule to invert the generative model into a recognition model as detailed in the simulation section below. If this inversion is not possible, then we can introduce

a separate recognition model, Q, to approximate the inverse of the generative model. Such a recognition model can be learned with similar methods and interleaved with the generative model.

Of course, the recognition with the recognition model early in learning is not expected to be exact, but estimation of new parameters from the recognized data in the M-step to maximize the expectation can be expected to be better than the model with the initial arbitrary values. The new model can then be compared to the data again and, when necessary, be used to generate new expectations from which the model is refined. This procedure is known as the **expectation maximization** (EM) algorithm. The distributions after three and nine such iterations, where we have chosen new data points in each iteration, are shown in Figs 8.3B and C.

8.3 The Boltzmann machine

8.3.1 General one-layer module

Our last model that uses unsupervised learning is again a general learning machine invented by Geoffrey Hinton and Terrance Sejnowski in the mid 1980 called **Boltzmann machine**. This machine is a general form of a recurrent neural network with **visible nodes** that receive input or provide output, and **hidden notes** that are not connected to the outside world directly. Such a stochastic dynamic network, a recurrent system with hidden nodes, together with the adjustable connections, provide the system with enough degrees of freedom to approximate any dynamical system. While this has been recognized for a long time, finding practical training rules for such systems have been a major challenge for which there was only recently major progress. These machines use unsupervised learning to learn hierarchical representations based on the statistics of the world. Such representations are key to more advanced applications of machine learning and to human abilities.

The basic building block is a one-layer network with one visible layer and one hidden layer. An example of such a network is shown in Fig. 8.4. The nodes represent

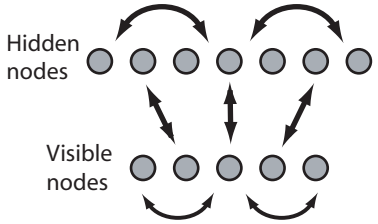

Fig. 8.4 A Boltzmann machine with one visible and one hidden layer.

random variable similar to the Bayesian networks discussed before. We will specifically consider binary nodes that mimic neuronal states which are either firing or not. The connections between the have weights w_{ij} which specify how much they influence the on-state of connected nodes. Such systems can be described by an energy function. The energy between two nodes that are symmetrically connected with strength w_{ij} is

$$H^{nm} = -\frac{1}{2}\sum_{ij} w_{ij} s_i^n s_j^m. \tag{8.9}$$

The state variables, s, have superscripts n or m which can have values (v) or (h) to indicate visible and hidden nodes. We consider again the probabilistic update rule,

$$p(s_i^n = +1) = \frac{1}{1 + \exp(-\beta \sum_j w_{ij} s_j^n)}, \tag{8.10}$$

with inverse temperature, β, which is called the Glauber dynamics in physics and describes the competitive interaction between minimizing the energy and the randomizing thermal force. The probability distribution for such a stochastic system is called the Boltzmann–Gibbs distribution. Following this distribution, the distribution of visible states, in thermal equilibrium, is given by

$$p(\mathbf{s}^v; \mathbf{w}) = \frac{1}{Z}\sum_{m \in h} \exp(-\beta H^{vm}), \tag{8.11}$$

where we summed over all hidden states. In other words, this function describes the distribution of visible states of a Boltzmann machine with specific parameters, $\mathbf{w}$, representing the weights of the recurrent network. The normalization term, $Z = \sum_{n,m} \exp(-\beta H^{nm})$, is called the **partition function**, which provides the correct normalization so that the sum of the probabilities of all states sums to one. These stochastic networks with symmetrical connections have been termed **Boltzmann machines** by **Ackley**, **Hinton** and **Sejnowski**.

Let us consider the case where we have chosen enough hidden nodes so that the system can, given the right weight values, implement a generative model of a given world. Thus, by choosing the right weight values, we want this dynamical system to approximate the probability function, $p(\mathbf{s}^v)$, of the sensory states (states of visible nodes) caused by the environment. To derive a learning rule, we need to define an objective function. In this case, we want to minimize the difference between two density functions. A common measure for the difference between two probabilistic distributions is the Kulbach–Leibler divergence (see Appendix 3.7),

$$\mathrm{KL}(p(\mathbf{s}^v), p(\mathbf{s}^v; \mathbf{w})) = \sum_{\mathbf{s}}^{v} p(\mathbf{s}^v) \log \frac{p(\mathbf{s}^v)}{p(\mathbf{s}^v; \mathbf{w})} \tag{8.12}$$

$$= \sum_{\mathbf{s}}^{v} p(\mathbf{s}^v) \log p(\mathbf{s}^v) - \sum_{\mathbf{s}}^{v} p(\mathbf{s}^v) \log p(\mathbf{s}^v; \mathbf{w}). \tag{8.13}$$

To minimize this divergence with a gradient method, we need to calculate the derivative of this 'distance measure' with respect to the weights. The first term in the difference in eqn 8.13 is the entropy (see Appendix **??**) of sensory states, which does not depend on the weights of the Boltzmann machine. Minimizing the Kulbach–Leibler divergence is therefore equivalent to maximizing the average log-likelihood function,

$$l(\mathbf{w}) = \sum_{\mathbf{s}}^{v} p(\mathbf{s}^v) \log p(\mathbf{s}^v; \mathbf{w}) = \langle \log p(\mathbf{s}^v; \mathbf{w}) \rangle. \tag{8.14}$$

In other words, we treat the probability distribution produced by the Boltzmann machine as a function of the parameters, $w_i j$, and choose the parameters which maximize

the likelihood of the training data (the actual world states). Therefore, the averages of the model are evaluated over actual visible states generated by the environment. The log-likelihood of the model increases the better the model approximates the world. A standard method of maximizing this function is gradient ascent, for which we need to calculate the derivative of $l(\mathbf{w})$ with respect to the weights. We omit the detailed derivation here, but we note that the resulting learning rule can be written in the form

$$\Delta w_{ij} = \eta \frac{\partial l}{\partial w_{ij}} = \eta \frac{\beta}{2} \left(\langle s_i s_j \rangle_{\text{clamped}} - \langle s_i s_j \rangle_{\text{free}} \right). \tag{8.15}$$

The meaning of the terms on the right-hand side is as follows. The term labelled 'clamped' is the thermal average of the correlation between two nodes when the states of the visible nodes are fixed. The termed labelled 'free' is the thermal average when the recurrent system is running freely. The Boltzmann machine can thus be trained, in principle, to represent any arbitrary density functions, given that the network has a sufficient number of hidden nodes.

This result is encouraging as it gives as an exact algorithm to train general recurrent networks to approximate arbitrary density functions. The learning rule looks interesting since the clamped phase could be associated with a sensory driven agent during an awake state, whereas the freely running state could be associated with a sleep phase. Unfortunately, it turns out that this learning rule is too demanding in practice. The reason for this is that the averages, indicated by the angular brackets in eqn 8.15, have to be evaluated at thermal equilibrium. Thus, after applying each sensory state, the system has to run for a long time to minimize the initial transient response of the system. The same has to be done for the freely running phase. Even when the system reaches equilibrium, it has to be sampled for a long time to allow sufficient accuracy of the averages so that the difference of the two terms is meaningful. Further, the applicability of the gradient method can be questioned since such methods are even problematic in recurrent systems without hidden states since small changes of system parameters (weights) can trigger large changes in the dynamics of the dynamical systems. These problems prevented, until recently, more practical progress in this area. Recently, Hinton and colleagues developed more practical, and biologically more plausible, systems which are described next.

8.3.2 The restricted Boltzmann machine and contrastive Hebbian learning

Training of the Boltzmann machine with the above rule is challenging because the states of the nodes are always changing. Even with the visible states clamped, the states of the hidden nodes are continuously changing for two reasons. First, the update rule is probabilistic, which means that even with constant activity of the visible nodes, hidden nodes receive variable input. Second, the recurrent connections between hidden nodes can change the states of the hidden nodes rapidly and generate rich dynamics in the system. We certainly want to keep the probabilistic update rule since we need to generate different responses of the system in response to sensory data. However, we can simplify the system by eliminating recurrent connections within each layer, although connections between the layers are still bidirectional. While the simplification of omitting collateral connections is potentially severe, much of the abilities of general

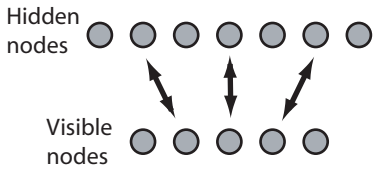

Fig. 8.5 Restricted Boltzmann machine in which recurrences within each later are removed.

recurrent networks with hidden nodes can be recovered through the use of many layers which bring back indirect recurrencies. A **restricted Boltzmann machine** (RBM) is shown in Fig. 8.5.

When applying the learning rule of eqn 8.15 to one layer of an RBM, we can expect faster convergence of the rule due to the restricted dynamics in the hidden layer. We can also write the learning rule in a slightly different form by using the following procedure. A sensory input state is applied to the input layer, which triggers some probabilistic recognition in the hidden layer. The states of the visible and hidden nodes can then be used to update the expectation value of the correlation between these nodes, $\langle s_i^v s_j^h \rangle^0$, at the initial time step. The pattern in the hidden layer can then be used to approximately reconstruct the pattern of visible nodes. This **alternating Gibbs sampling** is illustrated in Fig. 8.6 for a connection between one visible node and one hidden node, although this learning can be done in parallel for all connections. The learning rule can then be written in form,

$$\Delta w_{ij} \propto \langle s_i^v s_j^h \rangle^0 - \langle s_i^v s_j^h \rangle^\infty. \tag{8.16}$$

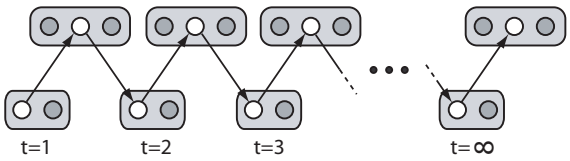

Fig. 8.6 Alternating Gibbs sampling.

Alternating Gibbs sampling becomes equivalent to the Boltzmann machine learning rule (eqn 8.15) when repeating this procedure for an infinite number of time steps, at which point it produces pure fantasies. However, this procedure still requires averaging over long sequences of simulated network activities, and sufficient evaluations of thermal averages can still take a long time. Also, the learning rule of eqn 8.16 does not seem to correspond to biological learning. While developmental learning also takes some time, it does not seems reasonable that the brain produces and evaluates long sequences of responses to individual sensory stimulations. Instead, it seems more reasonable to allow some finite number of alternations between hidden responses and the reconstruction of sensory states. While this does not formally correspond to the mathematically derived gradient leaning rule, it is an important step in solving the learning problem for practical problems, which is a form of **contrastive divergence** introduced by Geoffrey Hinton. It is heuristically clear that such a restricted training procedure can work. In each step we create only a rough approximation of ideal

average fantasies, but the system learns the environment from many examples, so that it continuously improves its expectations. While it might be reasonable to use initially longer sequences, as infants might do, Hinton and colleagues showed that learning with only a few reconstructions is able to self-organize the system. The self-organization, which is based on input from the environment, is able to form internal representations that can be used to generate reasonable sensory expectations and which can also be used to recognize learned and novel sensory patterns.

The basic Bolzmann machine with a visible and hidden layer can easily be combined into hierarchical networks by using the activities of hidden nodes in one layer as inputs to the next layer. Hinton and colleagues have demonstrated the power of restricted Boltzmann machines for a number of examples. For example, they applied layered RBMs as auto-encoders where restricted alternating Gibbs sampling was used as pre-training to find appropriate initial internal representations that could be fine-tuned with backpropagation techniques to yield results surpassing support vector machines. However, for our discussions of brain functions it is not even necessary to yield perfect solutions in a machine learning sense, and machines can indeed outperform humans in some classification tasks solved by machine learning methods. For us, it is more important to understand how the brain works.

Simulation 1: Hinton

To illustrate the function of an anticipating brain model, we briefly outline a demonstration by the Hinton group. The online demonstration can be run in a browser from `http://www.cs.toronto.edu/~hinton/adi`, and a stand alone version of this demonstration is available at this book's resource page. MATLAB source code for restricted Boltzmann machines are available at Hinton's home page. An image of the demonstration program is shown in Fig. 8.7. The model consists of a combination of restricted Boltzmann machines and a Helmholz machine. The input layer is called the **model retina** in the figure, and the system also contains a **recognition-readout-and-stimulation** layer. The model retina is used to apply images of handwritten characters to the system. The recognition-readout-and-stimulation layer is a brain imaging and stimulation device from and to the uppermost RBM layer. This device is trained by providing labels as inputs to the RBM for the purpose of 'reading the mind' of the system and to give it high-level instructions. This device learns to recognize patterns in the uppermost layer and map them to their meaning, as supplied during supervised learning of this device. This is somewhat analogous to **brain–computer interfaces** developed with different brain-imaging devices such as EEG, fMRI, or implanted electrodes. The advantage of the simulated device is that it can read the activity of every neuron in the upper RBM layer. The device can also be used with the learned connections in the opposite direction to stimulate the upper RBM layer with typical patterns for certain image categories.

The model for this demonstration was trained on images of handwritten numbers from a large database. Some example images can be seen on the left-hand side. All layers of this model were first treated as RBMs with symmetrical weights. Specifically, these were trained by applying images of handwritten characters to the model retina and using three steps of alternating Gibbs sampling for training the different layers. The evolving representations in each layer are thus purely unsupervised. After this

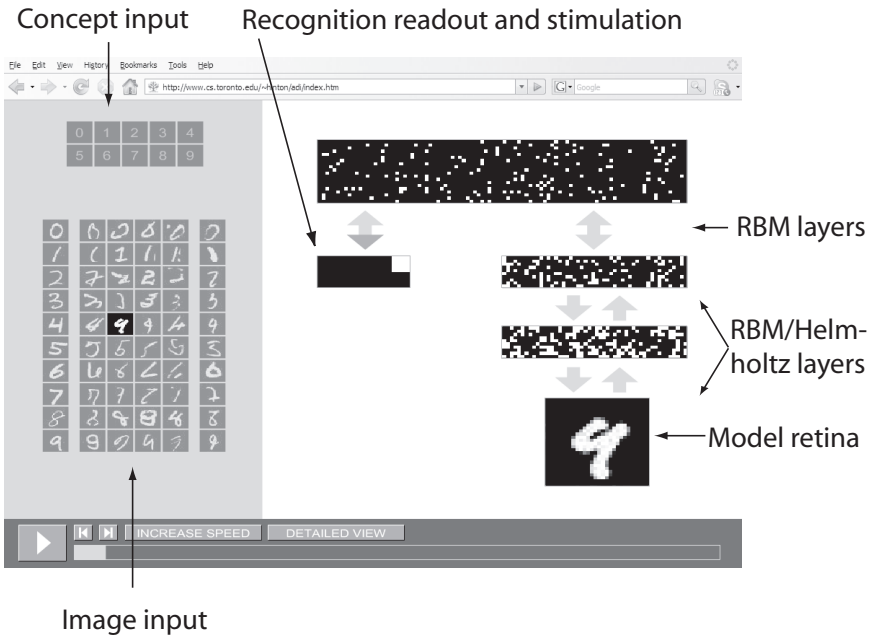

Fig. 8.7 Simulation of restricted Boltzmann machine by Geoffrey Hinton and colleagues, available at `www.cs.toronto.edu/~hinton/adi`.

basic training, the model was allowed, for fine-tuning purposes, to develop different weight values for the recognition and generative model as in Helmholtz machines with a wake–sleep training algorithm as mentioned above.

The simulations provided by Hinton demonstrate the ability of the system after training. The system can be tested in two ways, either by supplying a handwritten image and asking for recognition, or by asking the system to produce images of a certain letter. These two modes can be initiated by selecting either an image or by selecting a letter category on the left-hand side. In the example shown in Fig. 8.7, we selected an example of an image of the number 4. When running the simulation, this image triggers response patterns in the layers. These patterns change in every time step, due to the probabilistic nature of the updating rule. The recognition read-out of the uppermost layer does, therefore, also fluctuate. In the shown example, the response of the system is 4, but the letter 9 is also frequently reported. This makes sense, as this image does also look somewhat like the letter 9. A histogram of responses can be constructed when counting the responses over time, which, when properly normalized, corresponds to an estimate of the probability over high-level concepts generated by this sensory state. Thus, this mode tests the recognition ability of the model.

The stimulation device connected to the upper RBM layer allows us to instruct the system to 'visualize' specific letters, which corresponds to testing the generative ability of the model. For example, if we ask the system to visualize a letter 4 by evoking corresponding patterns in the upper layer, the system responds with varying images on the model retina. There is not a single right answer, and the answers of the system change with time. In this way, the system produces examples of possible

images of letter 4, proportional to some likelihood that these images are encountered in the sensory world on which the system was trained. The probabilistic nature of the system much better resembles human abilities to produce a variety of responses, in contrast to the neural networks that have been popular in the 1980s, so called multilayer perceptrons, which were only able to produce single answers for each input.

Simulation 2: Simplified one layer model

A simplified version of the RBM trained on some letters are included in folder `RBM example` on the web resource page. The overage reconstruction error and some examples of reconstructions after training are shown in Fig.8.8.

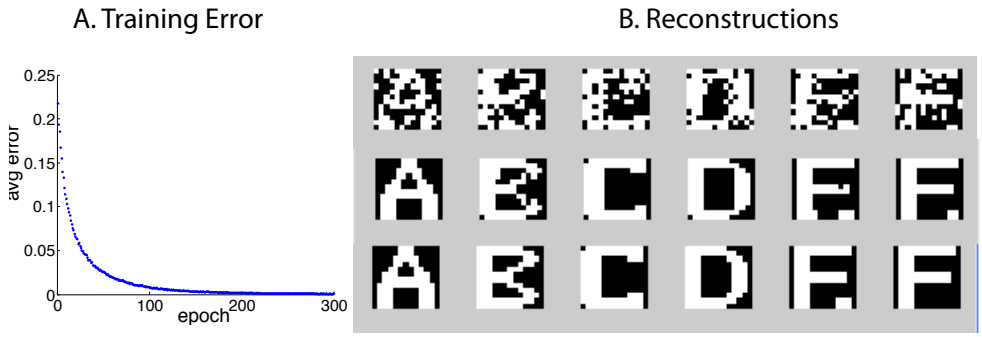

Fig. 8.8 (A) Reconstruction error during training of alphabet letters the letters, and (B) reconstructions after learning.

8.3.3 Other important unsupervised learning algorithms

We have mainly discussed the unsupervised learning of generative models in this chapter, but there are a number of unsupervised learning algorithms that are frequently used in data mining applications. In particular, these include dimensionality reduction algorithms like **PCA, ICA, local embedding methods, self-organizing maps**, etc. We have also discussed **deep auto-encoders** in the last chapter and have seen that they too can be thought of a compression (= dimensionality reduction).

We won't have time to discuss these methods in detail, but **sklearn** includes implementations of a large variety of these methods and also includes good explanations. Note that some of these models are a bit heuristic, although they might be quite good for some applications. In our discussion I tried to emphasize the underlying principles, that the principle solution of all these problems is to model the density functions of the data and their causal (conditional) relations.