CHAPTER 7

Deep Neural Networks

I think the brain is essentially a computer and consciousness is like a computer program. It will cease to run when the computer is turned off. Theoretically, it could be re-created on a neural network, but that would be very difficult, as it would require all one's memories.

—Stephen Hawking, *Time* magazine

Proposed in the 1940s as a simplified model of the elementary computing unit in the human cortex, *artificial neural networks* (ANNs) have since been an active research area. Among the many evolutions of ANN, *deep neural networks* (DNNs) (Hinton, Osindero, and Teh 2006) stand out as a promising extension of the shallow ANN structure. The best demonstration thus far of hierarchical learning based on DNN, along with other Bayesian inference and deduction reasoning techniques, has been the performance of the IBM supercomputer Watson in the legendary tournament on the game show *Jeopardy!*, in 2011.

This chapter starts with some basic introductory information about ANN then outlines the DNN structure and learning scheme.

Introducting ANNs

ANNs have been successfully used in many real-life applications, especially in supervised-learning modes. However, ANNs have been plagued by a number of notable challenges and shortcomings. Among the many challenges in supervised learning is the curse of dimensionality (Arnold et al. 2011), which occurs when the number of features and training points becomes significantly large. Big data thus makes ANN learning more difficult, owing to the overwhelming amount of data to process and the consequent memory and computational requirements. Another challenge in classification is the data nonlinearity that characterizes the feature overlap of different classes, making the task of separating the classes more difficult. Primarily for these reasons and the heuristic approach to select the appropriate network architecture, ANNs lagged through the 1990s and 2000s behind the widely adopted support vector machines (SVMs), which proved to be, in many respects, superior to ANNs.

Note SVM offers a principled approach to machine learning problems because of its mathematical foundations in statistical learning theory. SVM constructs solutions as a weighted sum of support vectors, which are only a subset of the training input. Like ANN, SVM minimizes a particular error cost function, based on the training data set, and relies on an empirical risk model. Additionally, SVM uses structural risk minimization and imposes an additional constraint on the optimization problem, forcing the optimization step to find a model that will eventually generalize better as it is situated at an equal and maximum distance between the classes.

With advancements in hardware and computational power, DNNs have been proposed as an extension of ANN shallow architectures. Some critics consider deep learning just another "buzzword for neural nets" (Collobert 2011). Although they borrow the concept of neurons from the biological brain, DNNs do not attempt to model it as cortical algorithms (CAs) or other biologically inspired machine learning approaches do. DNN concepts stem from the neocognitron model proposed by Fukushima (1980). Broadly defined as a consortium of machine learning algorithms that aims to learn in a hierarchical manner and that involves multiple levels of abstraction for knowledge representation, DNN architectures are intended to realize strong artificial intelligence (AI) models. These architectures accumulate knowledge as information propagates through higher levels in a manner such that the learning at the higher level is defined by and built on the statistical learning that happens at the lower-level layers.

With such a broad definition of deep learning in mind, we can construe the combinations of the backpropagation algorithm (available since 1974) with recurrent neural networks and convolution neural networks (introduced in the 1980s) as being the predecessors of deep architectures. However, it is only with the advent of Hinton, Osindero, and Teh's (2006) contribution to deep learning training that research on deep architectures has picked up momentum. The following sections give a brief overview of ANN, along with introducing in more detail *deep belief networks* (DBNs) and *restricted Boltzmann machines* (RBMs).

Early ANN Structures

One of the first ANN attempts dates back to the late 1940s, when the psychologist Donald Hebb (Hebb 1949) introduced what is known today as *Hebbian learning*, based on the plasticity feature of neurons: when neurons situated on either side of a synapse are stimulated synchronously and recurrently, the synapse's strength is increased in a manner proportional to the respective outputs of the firing neurons (Brown et al. 1990), such that

$$w_{ij}(t+1) = w_{ij}(t) + \eta_{ij} x_i(t) x_j(t),$$

where t represents the training epoch, w_{ij} is the weight of the connection between the ith and the jth neurons, x_i is the output of the ith neuron, and h_{ij} is a learning rate specific to the synapse concerned.

The *Hebbian rule* is an unsupervised-learning scheme that updates the weights of a network locally; that is, the training of each synapse depends on the weights of the neurons connected to it only. With its simple implementation the Hebbian rule is considered the first ANN learning rule, from which multiple variants have stemmed. The first implementations of this algorithm were in 1954, at the Massachusetts Institute of Technology, using computational machines (Farley and Clark, 1954).

The 1950s also saw the introduction of the *perceptron*, a two-layer neural network model for pattern recognition, using addition and subtraction operations (Rosenblatt 1958). The model consists of four components, as depicted in Figure 7-1. The retina, or input region, receives stimulus through sensory units. The connections are called localized because their origin points tend to cluster around a certain point or in a certain area. Although units in the projection area are identical to those in the association area, the projection area receives input through localized connections, whereas input to the association area emerges from the projection area through random connections; as if the input is generated from scattered areas. The A-units receive a set of transmitted impulses that may be excitatory or inhibitory. If the stimulus exceeds a certain threshold, the units respond by firing. The random connections between the association area and the response units are bidirectional. The feedforward connections transmit synapses from the association area to the responses, whereas the feedback connections transmit excitatory synapses to the source points in the association area from which the connection is generated. Inhibitory synapses complement the source points in the association areas that do not transmit to the response concerned.

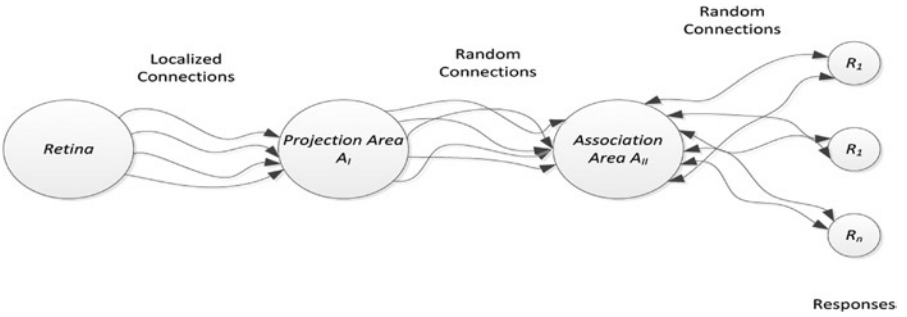

Figure 7-1. *A Rosenblatt perceptron structure*

Classical ANN

The basic structure of an ANN is the artificial neuron shown in Figure 7-2, which resembles the biological neuron in its shape and function (Haykin 1994).

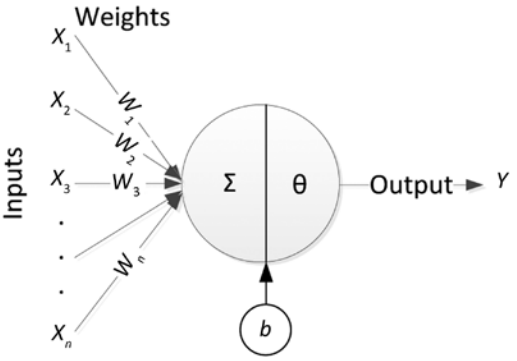

Figure 7-2. *An artificial neuron*

■ **Note** In the human body's nervous system, neurons generate, transmit, and receive electrical signals called *action potential*. A typical biological neuron has the following three basic components:

- *Cell body*: Can have a variety of sizes and shapes
- *Dendrites*: Numerous, treelike structures that extend from the cell body and that constitute the receptive portion of the neuron (i.e., the input site)
- *Axon*: A long, slender structure, with relatively few branches, that transmits electrical signals to connected areas

---

The inputs (X) are connected to the neuron through weighted connections emulating the dendrite's structure, whereas the summation, the bias (b), and the activation function (θ) play the role of the cell body, and the propagation of the output is analogous to the axon in a biological neuron.

Mathematically, a neuron is equivalent to the function:

$$Y = \theta\left(\sum_{i=1}^{n} W_i X_i + b\right),$$

which can be conveniently modeled, using a matrix form,

$$Y = \theta(W.X + b),$$

where $W = \begin{bmatrix} W_1 & W_2 & \cdots & W_n \end{bmatrix}$, and $X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}$.

The activation function shapes the output or state of the neuron. Multiple activation functions can be used, the most common of which are as follows:

- *Hard limiter*: $\theta(a) = \begin{cases} 0 \ if \ a < 0 \\ 1 \ if \ a > 0 \end{cases}$
- *Saturating linear function*: $\theta(a) = \begin{cases} 0 & if \ a < 0 \\ a & if \ 0 \le a \le 1 \\ 1 & if \ a > 1 \end{cases}$
- *Log-sigmoid function*: $\theta(a) = \dfrac{1}{1+e^{-a}}$
- *Hyperbolic tangent sigmoid function*: $\theta(a) = \dfrac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$

The bias shifts the activation function to the right or the left, as necessary for learning, and can in some cases be omitted.

A *neural network* is simply an association of cascaded layers of neurons, each with its own weight matrix, bias vector, and output vector. A layer of neurons is a "column" of neurons that operate in parallel, as shown in Figure 7-3. Each element of this column is a single neuron, with the output of the layer being the vector output, which is formed by the individual outputs of neurons. If an input vector is constituted of N inputs and a layer of M neurons, W_{ij} represents the weight of the connection of the jth input to the ith neuron of the layer; Y_i and b_i are, respectively, the output of and the bias associated with the jth neuron.

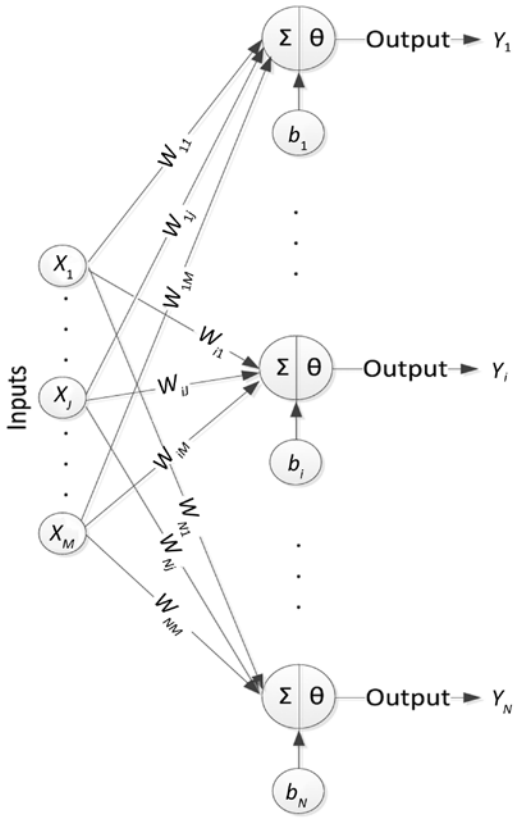

Figure 7-3. *A layer of neurons*

A layer of neurons can be conveniently represented, using matrix notation, as follows:

$$W = \begin{bmatrix} W_{11} & \dots & W_{1M} \\ \vdots & \vdots & \vdots \\ W_{N1} & \dots & W_{NM} \end{bmatrix}.$$

The row index in each element of this matrix represents the destination neuron of the corresponding connection, whereas the column index refers to the input source of the connection.

Designating by Y the output of the layer, you can write

$$Y = \begin{bmatrix} Y_1 \\ \vdots \\ Y_i \\ \vdots \\ Y_N \end{bmatrix} = \begin{bmatrix} \theta\left(\sum_{j=1}^{M} W_{1j} X_j + b_1\right) \\ \vdots \\ \theta\left(\sum_{j=1}^{M} W_{ij} X_j + b_i\right) \\ \vdots \\ \theta\left(\sum_{j=1}^{M} W_{Nj} X_j + b_N\right) \end{bmatrix} = \theta\left(W.X + b\right),$$

where $= \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix}$.

To aid in identifying the layer corresponding to a particular matrix, superscript indexes are used. Thus, W_{ij}^k represents the weight of the connection between the *j*th neuron in layer *k*–1and the *i*th neuron in layer *k*, and Y_i^k is the output of the *i*th neuron of the *k*th layer. The network output is the output of the last layer (also called the output layer), and the other layers are called hidden layers. A network with two hidden layers is illustrated in Figure 7-4. For generalization purposes, you designate by N_k the number of hidden neurons in the *k*th layer.

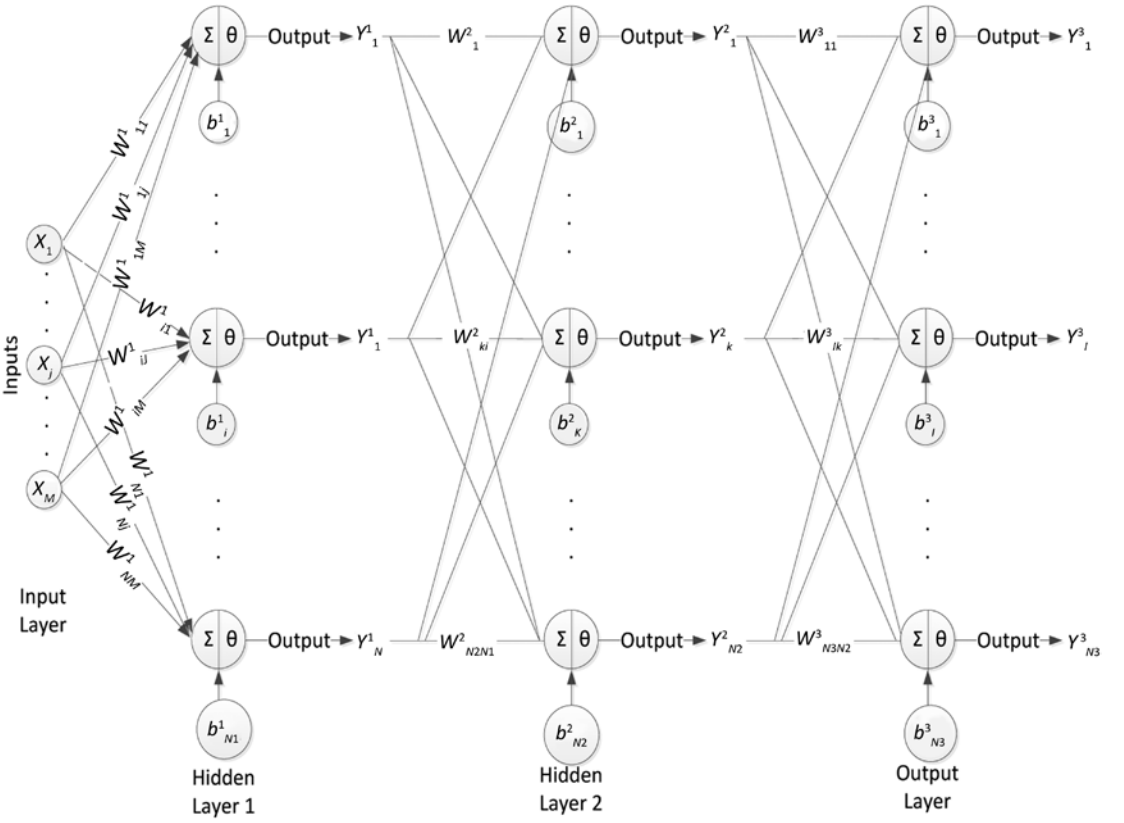

Figure 7-4. *A three-layer ANN*

The function achieved by this network is

$$Y^3 = \begin{bmatrix} Y_1^3 \\ \vdots \\ Y_i^3 \\ \vdots \\ Y_{N3}^3 \end{bmatrix} = \theta\left(W^3 Y^2 + b^3\right) = \theta\left(W^3\theta\left(W^2 Y^1 + b^2\right) + b^3\right) = \theta\left(W^3\theta\left(W^2\left(\theta\left(W^1 X + b^1\right)\right) + b^2\right) + b^3\right).$$

■ **Note** For the sake of simplicity, the same activation function θ has been adopted in all layers. However, multiple activation functions can be used in different layers in a network. Also, the number of neurons per layer may not be constant throughout the network.

The optimal number of layers and neurons for best performance is a question yet to be answered decisively, because this number is application dependent. A layer of hidden neurons divides the input space into regions whose boundaries are defined by the hyperplanes associated with each neuron.

The smaller the number of hidden neurons, the fewer the subregions created and the more the network tends to cluster points and map them to the same output. The output of each neuron is a non linear transformation of a hyperplane. In the case of classification, this separating curve formed by weighted inputs coming from the previous layer contributes, with other neurons in the same layer, in defining the final classification boundary. With a large number of neurons, the risk of overfitting increases, and the generalized performance decreases, because of overtraining. The network must be trained with enough data points to ensure that the partitions obtained at each hidden layer correctly separate the data.

ANN Training and the Backpropagation Algorithm

To enable an ANN to recognize patterns belonging to different classes, training on an existing dataset seeks to obtain iteratively the set of weights and biases that achieves the highest performance of the network (Jain, Mao, and Mohiuddin 1996).

In a network with M inputs, N output neurons, and L hidden layers, and given a set of labeled data—that is, a set of P pairs (X, T), where X is an M-dimensional vector, and T is an N-dimensional vector—the learning problem is reduced to finding the optimal weights, such that a cost function is optimized. The output of the network should match the target T_i and minimize the mean squared error,

$$E = \frac{1}{2}\sum_{i=1}^{P}\left\|Y_i - T_i^2\right\|,$$

where Y_i is the output obtained by propagating input X_i through the network.

ANNs can also use entropy as a cost function. Training requires at least a few epochs to update weights according to a weight update rule. It is to be noted that the backpropagation algorithm is widely adopted. It consists of the following steps:

1. *Initialization*: This step initializes the weights of the network in a random, weak manner; that is, it assigns random values close to 0 to the connections' weights.
2. *Feedforward*: The input X_i is fed into the network and propagated to the output layer. The resultant error is computed.
3. *Feedback*: The weights and biases are updated with:

$$W_{ij}^k(t+1) = W_{ij}^k(t) - \alpha\frac{\partial E}{\partial W_{ij}^k}$$

$$b_i^k(t+1) = b_i^k(t) - \alpha\frac{\partial E}{\partial b_i^k},$$

where a is a positive, tunable learning rate. The choice of a affects whether the backpropagation algorithm converges and how fast it converges. A large learning rate may cause the algorithm to oscillate, whereas a small learning rate may lead to a very slow convergence.

Because the update of the weights necessitates computing the gradient of the error (the cost function), it is essential for it to be differentiable. Failure to satisfy this condition prevents from using the backpropagation algorithm.

The computation of the gradient in the backpropagation algorithm can be simplified, using the chain rule, which calls for the following steps:

1. For each output unit $i = 1, 2, \ldots, N$ (in output layer L of Figure 7-4), the backpropagated error is computed, using

$$\delta_i^L = \frac{d\left(Y_i^L(t)\right)}{dt} \cdot \left(T_i - Y_i^L(t)\right),$$

where, T_i is the desired output; and, for the sigmoidal function,

$$\frac{d\left(Y_i^L(t)\right)}{dt} = Y_i^L(t)\left(1 - Y_i^L(t)\right),$$

resulting in the following expression:

$$\delta_i^L = Y_i^L(t)\left(1 - Y_i^L(t)\right)\left(T_i - Y_i^L(t)\right).$$

2. For each hidden unit $h = 1, 2, \ldots, N_k$ (in a hidden layer k with N_k hidden units), and moving from layer L–1, backward to the first layer, the backpropagated error can be computed as shown:

$$\delta_h^k = Y_h^k(t)\left(1 - Y_k^k(t)\right)\sum_{q=1}^{N_k} W_{qh}^{k+1}\delta_q^{k+1}.$$

3. The weights and biases are updated according to the following gradient descent:

$$W_{ij}^k(t+1) = W_{ij}^k(t) - \alpha\delta_i^k Y_j^{k\text{-}1}$$

$$b_i^k(t+1) = b_i^k(t) - \alpha\delta_i^k.$$

The network error is eventually reduced via this gradient-descent approach. For instance, considering a one-dimensional training point that belongs to class 1 (+1) and that is wrongly classified as class 2 (–1), the hyperplane should be moved away from class 1. Because, the hyperplane will be shifted to the left (decrease in W_{ij}^k) if $\delta_i^k Y_j^{k-1} > 0$, and it will be shifted to the right (increase in W_{ij}^k) if $\delta_i^k Y_j^{k-1} < 0$.

DBN Overview

DBNs, a deep architecture widely seen in the literature since the introduction of a fast, greedy training algorithm (Hinton, Osindero, and Teh 2006), are a network of stochastic neurons grouped in layers, with no intralayer neuron connections. The first two layers of the network contain neurons with undirected connections, which form an associative memory, analogous to biological neurons, whereas the remaining hidden layers form a directed acyclic graph, as displayed in Figure 7-5.

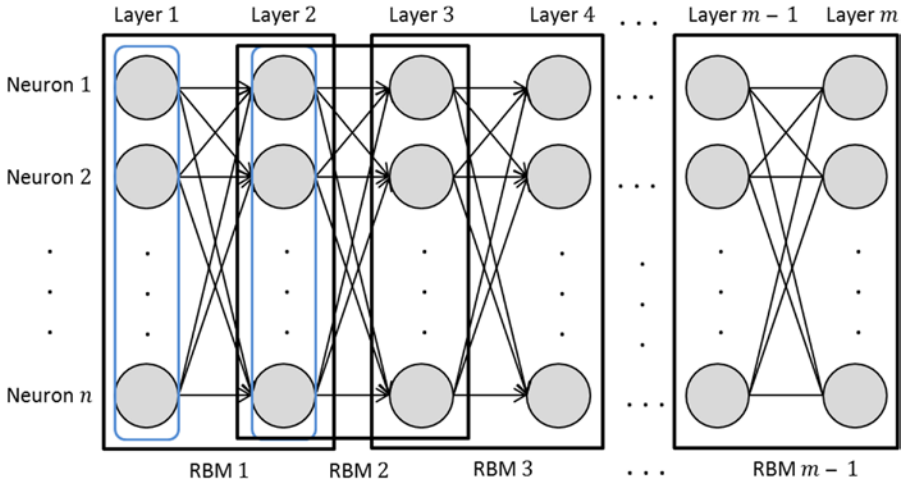

Figure 7-5. *DBN architecture*

Although a DBN can be viewed as an ANN with more hidden layers, training a DBN, using backpropagation, does not produce a good machine learning model, because the explaining-away phenomenon makes inference more difficult in deep models. When training a network, the simplifying assumption that layers are independent. *Explaining away* (also called *Berkson's paradox* or *selection bias*), makes this assumption invalid; the hidden nodes become anticorrelated. For example, if an output node can be activated by two equally rare and independent events with an even smaller chance of occurring simultaneously (because the probability of two independent events' occurring simultaneously is the product of both probabilities), then the occurrence of one event negates ("explains away") the occurrence of the other, such that a negative correlation is obtained between the two events. As a result of the difficulty of training deep architectures, DBNs lost popularity until Hinton and Salakhutdinov (2006) proposed a greedy training algorithm to train them efficiently. This algorithm broke down DBNs into sequentially stacked RBMs, which is a two-layer network constrained to contain only interlayer neuron connections, that is, connections between neurons that do not belong to the same layer.

As shown in Figure 7-6, connections between neurons in layer 1 are not allowed, and the same goes for layer 2; connections have to link a neuron from layer 1 to a neuron in layer 2 only. In a DBN the first two layers are allowed to have bidirectional connections, whereas the remaining layers have just directed connections. Therefore, interest in deep architectures was renewed, as training them became feasible and fast, involving training RBM units independently before adjusting the weights, using an up–down algorithm to avoid underfitting (Hinton, Osindero, and Teh 2006).

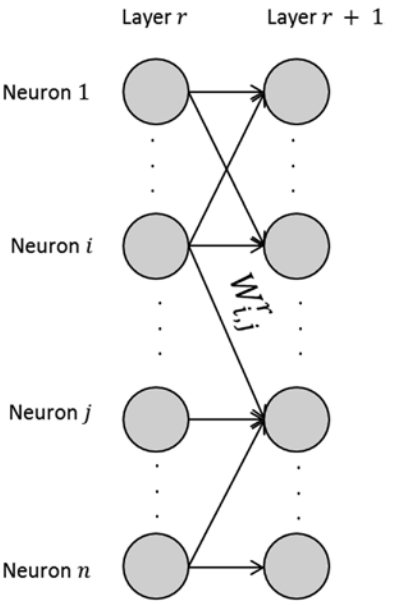

Figure 7-6. *Weight labeling*

Following is list of the DBN nomenclature adopted here:

DNN Nomenclature

$W^r_{i,j}$: Weight of the edge connecting neuron *i* in layer *r* to neuron *j* in layer; *r* is suppressed when there are only two layers in the network

$\boldsymbol{W}^r_i$: Weight vector of all connections leaving neuron *i* in layer *r*

$\boldsymbol{W}_r$: Weight vector connecting layer *r* to layer $r+1$

μ: Learning rate

k: Number of Gibbs sampling steps performed in contrastive divergence

n: Total number of hidden layer neurons

m: Total number of input layer neurons

$Q(.|.)$: Conditional probability distribution

h^r: Binary configuration of layer *r*

$p(h^r)$: Prior probability of h^r under the current weight values

v^0: Input layer datapoint $v^{(t)}_j$: binary configuration of neuron *j* in the input layer at sampling step *t*

H_i: Binary configuration variable of neuron *i* in the hidden layer at sampling step *t*

$h^{(t)}_i$: Binary configuration value of neuron *i* in the hidden layer at sampling step *t*

b_j: Bias term for neuron *j* in the input layer

c_i: Bias term for neuron *i* in the hidden layer

Restricted Boltzmann Machines

Boltzmann machines (BMs) are two-layer neural network architectures composed of neurons connected in an interlayer and intralayer fashion. *Restricted Boltzmann machines* (RBMs), first introduced under the name *Harmonium*, by Smolensky (1986), are constrained to form a bipartite graph. A bipartite graph is a two-layer graph, in which the nodes of the two layers form two disjoint sets of neurons This is achieved by restricting intralayer connections, such that connections between nodes in the same layer are not permitted. This restriction is what distinguishes BMs from RBMs and makes RBMs simpler to train. An RBM with undirected connections between neurons of the different layers forms an *autoassociative memory*, analogous to neurons in the human brain. Autoassociative memory is characterized by feedback connections that allow the exchange of information between neurons in both directions (Hawkins 2007).

RBMs can be trained in a supervised and unsupervised fashion. The weight vector is updated, using Hinton's *contrastive divergence* (CD) algorithm (Hinton 2002). CD is an algorithm that approximates the log-likelihood gradient and that requires fewer sampling steps than the *Markov chain Monte Carlo* (MCMC) algorithm (Hinton 2002). CD performs k steps of Gibbs sampling and gradient descent to find the weight vector that maximizes the objective function (Hinton 2010), which is the product of probabilities. As k increases, the performance of the learned model improves, however at the cost of a longer training time. A typical value for this parameter is $k = 1$ (Hinton 2010). The workflow of the training algorithm is shown in Table 7-1.

Table 7-1. *RBM Training Algorithm Workflow, Using CD (Fischer and Igel, 2012)*

1. Initialize the weights to 0.
2. For each sample from the training batch:
   a. Apply the sample to the network input.
   b. For 0 to k-1 sampling steps,
      i. for each hidden layer neuron from 1 to n, sample $h_i^{(t)} \sim p\left(h_i | \boldsymbol{v}^{(t)}\right)$;
      ii. for each input layer neuron from 1 to m, sample $v_j^{(t)} \sim p\left(v_j | \boldsymbol{h}^{(t)}\right)$.
   c. For each input and hidden layer neuron, compute
      i. $\Delta w_{ij} = \Delta w_{ij} + p\left(H_i = 1 | v^{(0)}\right) v_j^{(0)} - p\left(H_i = 1 | v^{(k)}\right) v_j^{(k)}$
      ii. $\Delta b_j = \Delta b_j + v_j^{(0)} - v_j^{(k)}$
      iii. $\Delta c_i = \Delta c_i + p\left(H_i = 1 | v^{(0)}\right) - p(H_i = 1 \,|\, v^{(k)})$

Based on the Gibbs distribution, the energy function or loss function used to describe the joint probability distribution is denoted in Equation 7-1, where w_{ij}, b_j, and c_i are real-valued weights, and h_i and v_j can take values in the set (Aleksandrovsky et al. 1996):

$$E(v,h) = -\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij} h_i v_j - \sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} c_i h_i. \tag{7-1}$$

The joint probability distribution is thus computed using Equation 7-2:

$$p(v,h) = \frac{1}{\sum_{v}\sum_{h} e^{-E(v,h)}} e^{-E(v,h)}. \tag{7-2}$$

DNN Training Algorithms

Backpropagation is one of the most popular algorithms used to train ANNs (Werbos 1974). Equation 7-3 displays a simple formulation of the weight update rule, used in backpropagation:

$$w_1^r(\text{new}) = w_1^r(\text{old}) - \mu \frac{\partial J}{\partial w_1^r} \tag{7-3}$$

However, as the depth of the network increases, backpropagation's performance degradation increases as well, making it unsuitable for training general deep architectures. This is due to the vanishing gradient problem (Horchreiter 1991; Horchreiter et al. 2001; Hinton 2007; Bengio 2009), a training issue in which the error propagated back in the network shrinks as it moves from layer to layer, becoming negligible in deep architectures and making it almost impossible for the weights in the early layers to be updated. Therefore, it would be too slow to train and obtain meaningful results from a DNN.

Because of backpropagation's shortcomings, many attempts were made to develop a fast training algorithm for deep networks. *Schmidhuber's algorithm* (Schmidhuber 1992) trained a multilevel hierarchy of recurrent neural networks by using unsupervised pretraining on each layer and then fine-tuning the resulting weights via backpropagation.

Interest in DNNs was renewed in 2006, when Hinton and Salakhutdinov (2006) proposed a greedy, layer-by-layer training algorithm for DBNs that attempts to learn simpler models sequentially and then fine-tune the results for the overall model. Using *complementary priors* to eliminate the explaining-away effect, the algorithm consists of two main steps:

1. A greedy layer-wise training to learn the weights by
   a. Tying the weights of the unlearned layers.
   b. Applying CD to learn the weights of the current layer.
2. An up-down algorithm for fine-tuning the weights

Instead of learning the weights of millions of connections across many hidden layers at once, this training scheme finds the optimal solution for a single layer at a time, which makes it a greedy algorithm. This is accomplished by tying all the weights of the following layers and learning only the weights of the current layer. Tying weights also serves to eliminate the explaining-away phenomenon, which results in poorly trained deep networks when adopting other training algorithms. As illustrated in Figure 7-7, the weights W_0 between layers 1 and 2 are learned. The weights between all the following layers are tied to W_0. Once CD learning has converged, the weights W_1, between layers 2 and 3, are learned by tying the weights of all the following layers to W_1 and fixing the weights between layers 1 and 2 that were learned in the previous stage to W_0. Similarly, when the CD converges to the optimal values for W_1, the weights of the third RBM block are untied from the second RBM block, and CD is used to learn the final set of weights W_2.

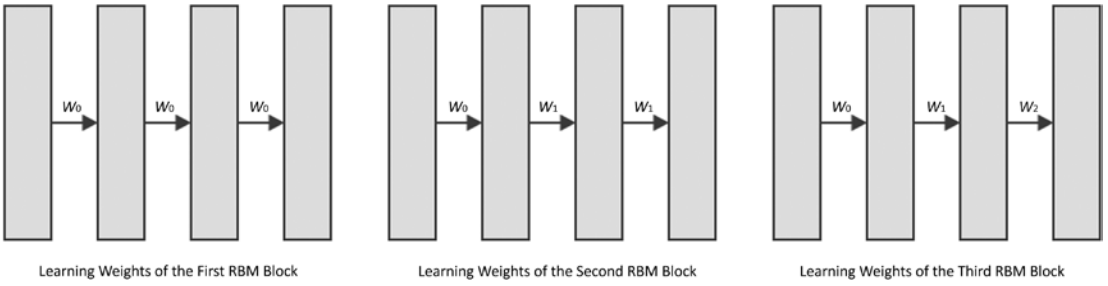

Figure 7-7. *Sequential training*

This process of tying, learning, and untying weights is repeated until all layers have been processed. DBNs with tied weights resemble RBMs. Therefore, as mentioned earlier, each RBM is learned, using CD learning. However, this algorithm can only be applied if the first two layers form an undirected graph, and the remaining hidden layers form a directed, acyclic graph.

The energy of the directed model is computing, using Equation 7-4, which is bounded by Equation 7-5. Tying the weights produces equality in Equation 7-5 and renders $Q(.|v^0)$ and $p(v^0|h^0)$ constant. The derivative of Equation 7-5 is simplified and equal to Equation 7-6. Therefore, tying the weights leads to a simpler objective function to maximize. Applying this rule recursively allows the training of a DBN (Hinton, Osindero, and Teh 2006).

$$E\left(v^0,h^0\right)=-\left(\log p\left(h^0\right)+\log p\left(v^0|h^0\right)\right) \tag{7-4}$$

$$\log p\left(v^0\right)$$

$$\geq \sum_{all\,h^0} Q\left(h^0|v^0\right)\left(\log p\left(h^0\right)+\log p\left(v^0|h^0\right)\right)$$

$$-\sum_{all\,h^0} Q\left(h^0|v^0\right)\log Q\left(h^0|v^0\right) \tag{7-5}$$

$$\frac{\partial\left(\log p\left(v^0\right)\right)}{\partial w_{ij}}=\sum_{allh^0} Q\left(h^0|v^0\right)\log p\left(h^0\right) \tag{7-6}$$

Once the weights have been learned for each layer, a variant of the wake–sleep algorithm with the CD weight update rule is used to fine-tune the learned parameters. The up–down algorithm is used to backfit the obtained solution to avoid underfitting—an important concern when training in an unsupervised and greedy fashion. The up–down algorithm subjects lower-level layers, whose weights were learned early in the training, to the influence of the higher-level layers, whose weights were learned toward the end of training. In the bottom-up pass the generative weights on directed connections are adjusted by computing the positive phase probabilities, sampling the states, using the CD weight update rule, and running Gibbs sampling for a limited number of iterations. The top-down pass will stochastically activate each of the lower layers, using the top-down connections. This is done by computing the negative phase probabilities, sampling the states, and computing the predictions of the network. Appropriate adjustments to the generative and inference parameters as well as the top-layer weights are performed in a contrastive form of the wake–sleep algorithm, because it addresses issues in the sleep phase of the algorithm. The workflow for this algorithm is shown in Table 7-2.

Table 7-2. *Up–Down Algorithm Workflow (Hinton and Salakhutdinov 2006)*

1. In the bottom-up pass:
   a. Compute positive phase probabilities.
   b. Sample states.
   c. Compute CD statistics, using the positive phase probabilities.
   d. Perform Gibbs sampling for a predefined number of iterations, based on the associative memory part of the network.
   e. Compute negative phase contrastive divergence statistics, using information from step 1d.
2. In the top-down pass:
   a. Calculate negative phase probabilities.
   b. Sample states.
   c. Compute predictions.
3. Update generative parameters.
4. Update associative memory part of the network.
5. Update inference parameters.

Despite its limitations when applied to DNNs, interest in the backpropagation algorithm was renewed, because of the surge in graphics processing unit (GPU) computational power. Ciresan et al. (2010) investigated the performance of the backpropagation algorithm on deep networks. It was observed that, even with the vanishing gradient problem, given enough epochs, backpropagation can achieve results comparable to those of other, more complex training algorithms.

It is to be noted that supervised learning with deep architectures has been reported as performing well on many classification tasks. However, when the network is pretrained in an unsupervised fashion, it almost always performs better than the scenarios where pretraining is omitted without the pretraining phase (Erhan et al. 2010). Several theories have been proposed to explain this phenomenon, such as that the pretraining phase acts as a regularizer (Bengio 2009; Erhan et al. 2009) and an aid (Bengio et al. 2007) for the supervised optimization problem.

DNN-Related Research

The use of DBN in various machine learning applications has flourished since the introduction of Hinton's fast, greedy training algorithm. Furthermore, many attempts have been made to speed up DBN and address its weaknesses. The following sections offer a brief survey of the most recent and relevant applications of DBN, a presentation on research aimed at speeding up training as well as a discussion of several DBN variants and DNN architectures.

DNN Applications

DNN has been applied to many machine learning applications, including feature extraction, feature reduction, and classification problems, to name a few.

Feature extraction involves transforming raw input data to feature vectors that represent the input; raw data can be audio, image, or text. For example, DBN has been applied to *discrete Fourier transform* (DFT) representation of music audio (Hamel and Eck 2010) and found to outperform Mel frequency cepstral coefficients (MFCCs), a widely used method of music audio feature extraction.

Once features are extracted from raw data, the high-dimensional data representation may have to be reduced to alleviate the memory and computational requirements of classification tasks as well as enable

better visualization of the data and decrease the memory needed to store the data for future use. Hinton and Salakhutdinov (Hinton and Salakhutdinov 2006; Salakhutdinov and Hinton 2007) used a stack of RBMs to pretrain the network and then employed autoencoder networks to learn the low-dimensional features.

Extracting expressive and low-dimensional features, using DBN, was shown to be possible for fast retrieval of documents and images, as tested on some ever-growing databases. Ranzato and Szummer (2008) were able to produce compact representations of documents to speed up search engines, while outperforming shallow machine learning algorithms. Applied to image retrieval from large databases, DBN produced results comparable to state-of-the art algorithms, including latent Dirichlet allocation and probabilistic latent semantic analysis (Hörster and Lienhart 2008).

Transferring learned models from one domain to another has always been an issue for machine learning algorithms. However, DNN was able to extract domain-independent features (Bengio and Delalleau 2011), making transfer learning possible in many applications (Collobert and Weston 2008; Glorot, Bordes, and Bengio 2011; Bengio 2012; Ciresan, Meier, and Schmidhuber 2012;Mesnil et al. 2012). DNNs have also been used for curriculum learning, in which data are learned is a specific order (Bengio et al. 2009).

DBN has been applied to many classification tasks in fields such as vision, speech, medical ailments, and *natural language processing* (NLP). Object recognition from images has been widely addressed, and DBN's performance exceeded state-of-the-art algorithms (Desjardins and Bengio 2008; Uetz and Behnke 2009; Ciresan et al. 2010; Ciresan, Meier, and Schmidhuber 2012). For instance, Ciresan et al. (2010) achieved an error rate of 0.35 percent on the Mixed National Institute of Standards and Technology (MNIST) database. Nair and Hinton (2009) outperformed shallow architectures, including SVM, on three-dimensional object recognition, achieving a 6.5 percent error rate, on the New York University Object Recognition Benchmark (NORB) dataset, compared with SVM's 11.6 percent. Considering speech recognition tasks, deep architectures have improved acoustic modeling (Mohamed et al. 2011; Hinton et al. 2012), speech-to-text transcription (Seide, Li, and Yu 2011), and large-vocabulary speech recognition (Dahl et al. 2012; Jaitly et al. 2012; Sainath et al. 2011). On phone recognition tasks, DBN achieved an error rate of 23 percent on the TIMIT database—better than reported errors, ranging from 24.4 percent to 36 percent, using other machine learning algorithms (Mohamed, Yu, and Deng 2010).

DBN produced classification results comparable to other machine learning algorithms in seizure prediction, using electroencephalography (EEG) signals, but reached those results in significantly faster times—between 1.7 and 103.7 times faster (Wulsin et al. 2011). McAfee (2008) adopted DBN for document classification and showed promise for succeeding on such databases.

Generating synthetic images—specifically facial expressions—from a high-level description of human emotion is another area in which DBN has been successfully applied, producing a variety of realistic facial expressions (Susskind et al. 2008).

NLP, in general, has also been investigated with deep architectures to improve on state-of-the-art results. Such applications include machine transliteration (Deselaers et al. 2009), sentiment analysis (Zhou, Chen, and Wang 2010; Glorot, Bordes, and Bengio 2011), and language modeling (Collobert and Weston 2008; Weston et al. 2012)—including part-of-speech tagging, similar-word recognition, and chunking. The complexity of these problems requires a machine learning algorithm with more depth (Bengio and Delalleau 2011) to produce meaningful results. For example, machine transliteration poses a challenge to machine learning algorithms, because the words do not have a unified mapping, which leads to a many-to-many mapping that does not exist in dictionaries. Additionally, the large number of source-to-target language-pair character symbols and different sound structures leading to missing sounds are just a few properties of transliteration that make it difficult for machines to do well.

Parallel Implementations to Speed Up DNN Training

Sequentially training a DBN layer by layer becomes more time-consuming as the layer and network sizes increase. Stacking the layers to form networks, called *deep-stacking networks*, and training the network on CPU clusters, as opposed to one supercomputer (Deng, Hutchinson, and Yu 2012), exploit the inherent parallelism in the greedy training algorithm to achieve significant training-time savings.

However, this method does not speed up the training time per layer. This can be achieved by parallelizing the training algorithm for the individual RBM layers, using GPUs (Cai et al. 2012).

However, use of the large and sparse data commonly employed to train RBMs creates challenges for parallelizing this algorithm. Modifying the computations for matrix-based operations and optimizing the matrix–matrix multiplication code for sparse matrices make GPU implementation much faster than CPU implementation.

As opposed to speeding up training via software, attempts have been made to speed up training via hardware, using field-programmable gate arrays (FPGAs). Ly and Chow (2010) mapped RBMs to FPGAs and achieved significant speedup of the optimized software code. This work was extended to investigate the scalability of the approach by Lo (2010).

Deep Networks Similar to DBN

One variation of DBN, called *modular DBN* (M-DBN), trains different parts of the network separately, while adjusting the learning rate as training progresses (Pape et al. 2011), as opposed to using one training set for the whole network. This allows M-DBN to avoid forgetting features learned early in training, a weakness of DBN that can hinder its performance in online learning applications in which the data distribution changes dynamically over time.

Sparse DBN learns sparse features—unlike Hinton's DBN, which learns nonsparse data representations—by adding a penalty in the objective function for deviations from the expected activation of hidden units in the RBM formulation (Lee, Ekanadham, and Ng 2007).

Convolutional DBN integrates translation invariance into the image representations by sharing weights between locations in an image, allowing inference to be done when the image is scaled up by using convolution (Lee et al. 2009). Therefore, convolutional DBN scales better to real-world-sized images without suffering from computational intractability as a result of the high dimensionality of these images.

DBNs are not the only deep architectures available. *Sum product network* (SPN) is a deep architecture represented as a graph with directed and weighted edges. SPN is *acyclic* (contains no loops), with variables on the leaves of the graph, and its internal nodes consist of sum and product operations (Poon and Domingo 2011). SPN trains, using backpropagation and expectation maximization (EM) algorithms. These simple operations result in a network that is more accurate, faster to train, and more tractable than DBN.

Deep Boltzmann machines (DBMs) are similar to but have a more general deep architecture than DBNs. They are composed of BMs stacked on top of each others (Salakhutdinov and Hinton 2009). Although more complex and slower to train than DBNs, owing to the symmetrical connections between all neurons in the BM network, the two-way edges let DBMs propagate input uncertainty better than DBNs, making their generative models more robust. The more complex architecture requires an efficient training algorithm to make training feasible. The DBN greedy training algorithm was modified to achieve a more efficient training algorithm for DBM by using an approximate inference algorithm. However, this rendered DBM training approximately three times slower than DBN training (Salakhutdinov and Larochelle 2010).

References

Aleksandrovsky, Boris, James Whitson, Gretchen Andes, Gary Lynch, and Richard Granger. "Novel Speech Processing Mechanism Derived from Auditory Neocortical Circuit Analysis." In *Proceedings of the Fourth International Conference on Spoken Language*, edited by H. Timothy Bunnell and William Idsardi, 558–561. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 1996.

Arnold, Ludovic, Sébastien Rebecchi, Sylvain Chevallier, and Hélène Paugam-Moisy. "An Introduction to Deep Learning." In Proceedings of the 19th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, Bruges, Belgium, April 27–29, 2011, edited by Michel Verleysen, 477–488. Leuven, Belgium: Ciaco, 2011.

Bengio, Yoshua. “Learning Deep Architectures for AI.” In *Foundations and Trends in Machine Learning* 2, no. 1 (2009): 1–127.

Bengio, Yoshua. “Deep Learning of Representations for Unsupervised and Transfer Learning.” In *ICML 2011: Proceedings of the International Conference on Machine Learning Unsupervised and Transfer Learning Workshop,* edited by Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, 17–36. 2012. http://jmlr.csail.mit.edu/proceedings/papers/v27/bengio12a/bengio12a.pdf.

Bengio, Yoshua, and Olivier Delalleau. “On the Expressive Power of Deep Architectures.” In *Algorithmic Learning Theory*, edited by Jyrki Kivinen, Csaba Szepesvári, Esko Ukkonen, and Thomas Zeugmann, 18–36. Berlin: Springer, 2011.

Bengio, Yoshua, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. “Greedy Layer-Wise Training of Deep Networks.” In *NIPS ’06: Proceedings of Advances in Neural Information Processing Systems 19*, edited by Bernhard Schlkopf, John Platt, and Thomas Hofmann, 153–160. Cambridge, MA: Massachusetts Institute of Technology Press, 2007.

Bengio, Yoshua, Jérôme Louradour, Ronan Collobert, and Jason Weston. “Curriculum Learning.” In *ICML ’09: Proceedings of the 26th Annual International Conference on Machine Learning*, edited by Léon Bottou and Michael Littman, 41–48. New York: ACM, 2009.

Brown, Thomas H., Edward W. Kairiss, and Claude L. Keenan. “Hebbian Synapses: Biophysical Mechanisms ad Algorithms.” *Annual Review of Neuroscience* 13, no. 1 (1990): 475–511.

Cai, Xianggao, Zhanpeng Xu, Guoming Lai, Chengwei Wu, and Xiaola Lin. “GPU-Accelerated Restricted Boltzmann Machine for Collaborative Filtering.” In *Algorithms and Architectures for Parallel Processing: Proceedings of the 12th International ICA3PP Conference, Fukuoka, Japan, September 2012,* edited by Yang Xiang, Ivan Stojmenović, Bernady O. Apduhan, Guojun Wang, Koji Nakano, and Albert Zomaya, 303–316. Berlin: Springer, 2012.

Ciresan, Dan Claudiu, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. “Deep, Big, Simple Neural Nets for Handwritten Digit Recognition.” *Neural Computation* 22, no. 12 (2010): 3207–3220.

Ciresan, Dan Claudiu, Ueli Meier, and Jürgen Schmidhuber. “Transfer Learning for Latin and Chinese Characters with Deep Neural Networks.” In *Proceedings of the 2012 International Joint Conference on Neural Networks*, 1–6. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 2012.

Collobert, Robert. “Deep Learning for Efficient Discriminative Parsing.” Recorded April 2011. AISTATS video, 21:16. Posted May 6, 2011. http://videolectures.net/aistats2011_collobert_deep/.

Collobert, Ronan, and Jason Weston. “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning.” In *ICML ’08: Proceedings of the 25th International Conference on Machine Learning*, edited by Andrew McCallum and Sam Roweis, 160–167. New York: ACM, 2008.

Dahl, George E., Dong Yu, Li Deng, and Alex Acero. “Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition.” *IEEE Transactions on Audio, Speech, and Language Processing* 20, no. 1 (2012): 30–42.

Deng, Li, Brian Hutchinson, and Dong Yu. “Parallel Training for Deep Stacking Networks.” In *Interspeech 2012: Proceedings of the 13th Annual Conference of the International Speech Communication Association*. 2012. www.isca-speech.org/archive/interspeech_2012.

Deselaers, Thomas, Saša Hasan, Oliver Bender, and Hermann Ney. “A Deep Learning Approach to Machine Transliteration.” In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, e233–241. Stroudsburg, PA: Association for Computational Linguistics, 2009.

Desjardins, Guillaume, and Yoshua Bengio. “Empirical Evaluation of Convolutional RBMs for Vision.” Technical report, Université de Montréal, 2008.

Erhan, Dumitru, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. “Why Does Unsupervised Pre-Training Help Deep Learning?” *Journal of Machine Learning Research* 11 (2010): 625–660.

Erhan, Dumitru, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. “The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training.” In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, edited by David van Dyk and Max Welling, 153–160. 2009. http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS09_ErhanMBBV.pdf.

Farley, B. G., and W. Clark. “Simulation of Self-Organizing Systems by Digital Computer.” *IEEE Transactions of the IRE Professional Group on Information Theory* 4, no. 4 (1954): 76–84.

Fischer, Asja, and Christian Igel. “An Introduction to Restricted Boltzmann Machines.” In Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: Proceedings of the 17th Iberoamerican Congress, CIARP 2012, Buenos Aires, Argentina, September 3–6, 2012, edited by Luis Alvarez, Marta E. Mejail, Luis E. Gomez, and Julio E. Jacobo, 14–36. Berlin: Springer, 2012.

Fukushima, Kunihiko. “Neocognition: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.” *Biological Cybernetics* 36 (1980): 193–202.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. “Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach.” In *ICML ’11: Proceedings of the 28th International Conference on Machine Learning*, 513–520. 2011. www.icml-2011.org/papers/342_icmlpaper.pdf.

Hamel, Philippe, and Douglas Eck. “Learning Features from Music Audio with Deep Belief Networks.” In ISMIR 2010: Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010), August 9–13, 2010, Utrecht, the Netherlands, edited by J. Stephen Downie and Rembo C. Veltkamp, 339–344. International Society for Music Information Retrieval, 2010. http://ismir2010.ismir.net/proceedings/ISMIR2010_complete_proceedings.pdf.

Hawkins, Jeff, and Sandra Blakeslee. *On Intelligence*. New York: Macmillan, 2007.

Haykin, Simon. *Neural Networks*. Upper Saddle River, NJ: Prentice Hall, 1994.

Hebb, Donald. *The Organization of Behavior*. New York: Wiley, 1949.

Hinton, Geoffrey E. “Training Products of Experts by Minimizing Contrastive Divergence.” *Neural Computation* 14, no. 8 (2002): 1771–1800.

Hinton, Geoffrey E. “To Recognize Shapes, First Learn to Generate Images.” *Progress in Brain Research* 165 (2007): 535–547.

Hinton, Geoffrey E.. “A Practical Guide to Training Restricted Boltzmann Machines.” *Momentum* 9, no. 1 (2010).

Hinton, Geoffrey E., Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, et al. “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups.” *IEEE Signal Processing Magazine* 29, no. 6 (2012): 82–97.

Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. “A Fast Learning Algorithm for Deep Belief Nets.” *Neural Computation* 18, no. 7 (2006): 1527–1554.

Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks.” *Science* 313, no. 5786 (2006): 504–507.

Hochreiter, Sepp. “Untersuchungen zu dynamischen neuronalen Netzen.” Master's thesis, Technical University of Munich, 1991.

Hochreiter, Sepp, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. “Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies.” In *A Field Guide to Dynamical Recurrent Neural Networks*, edited by John F. Kolen and Stefan C. Kremer, 237–244. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 2001.

Hörster, Eva, and Rainer Lienhart. “Deep Networks for Image Retrieval on Large-Scale Databases.” In *Proceedings of the 16th ACM International Conference on Multimedia*, 643–646. New York: ACM, 2008.

Jain, Anil K., Jianchang Mao, and K. M. Mohiuddin. “Artificial Neural Networks: A Tutorial.” *Computer* 29, no. 3 (1996): 31–44.

Jaitly, Navdeep, Patrick Nguyen, Andrew W. Senior, and Vincent Vanhoucke. “Application of Pretrained Deep Neural Networks to Large Vocabulary Speech Recognition.” In *Interspeech 2012: Proceedings of the 13th Annual Conference of the International Speech Communication Association.* 2012. www.isca-speech.org/archive/interspeech_2012/.

Lee, Honglak, Chaitanya Ekanadham, and Andrew Y. Ng. “Sparse Deep Belief Net Model for Visual Area V2.” *Proceedings of NIPS 2007: Advances in Neural Information Processing Systems*, edited by J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis. 2008. http://papers.nips.cc/paper/3313-sparse-deep-belief-net-model-for-visual-area-v2.pdf.

Lee, Honglak, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. “Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations.” In *ICML ’09: Proceedings of the 26th Annual International Conference on Machine Learning*, edited by Léon Bottou and Michael Littman, 609–616. New York: ACM, 2009.

Lo, Charles. “A FPGA Implementation of Large Restricted Boltzmann Machines.” In *Proceedings of the 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2–4, 2010, Charlotte, NC, 201–208. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 2010.

Ly, Daniel L., and Paul Chow. “High-Performance Reconfigurable Hardware Architecture for Restricted Boltzmann Machines.” *IEEE Transactions on Neural Networks* 21, no. 1 (2010): 1780–1792.

McAfee, Lawrence. “Document Classification Using Deep Belief Nets,” 2008.

Mesnil, Grégoire, Yann Dauphin, Xavier Glorot, Salah Rifai, Yoshua Bengio, Ian J. Goodfellow, Erick Lavoie, et al. “Unsupervised and Transfer Learning Challenge: A Deep Learning Approach.” In *ICML 2011: Proceedings of the International Conference on Machine Learning Unsupervised and Transfer Learning Workshop*, edited by Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, 97–110. 2012. http://jmlr.csail.mit.edu/proceedings/papers/v27/mesnil12a/mesnil12a.pdf.

Mohamed, Abdel-rahman, Tara N. Sainath, George Dahl, Bhuvana Ramabhadran, Geoffrey E. Hinton, and Michael A. Picheny. “Deep Belief Networks Using Discriminative Features for Phone Recognition.” In *Proceedings of the 2011 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 5060–5063. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 2011.

Mohamed, Abdel-rahman, Dong Yu, and Li Deng. “Investigation of Full-Sequence Training of Deep Belief Networks for Speech Recognition.” In *Interspeech 2010: Proceedings of 11th Annual Conference of the International Speech Communication Association,* edited by Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura, 2846–2849. 2010. www.isca-speech.org/archive/interspeech_2010/i10_2846.html.

Nair, Vinod, and Geoffrey E. Hinton. "3D Object Recognition with Deep Belief Nets." In *NIPS '09: Proceedings of Advances in Neural Information Processing Systems 22*, edited Yoshua Bengio, Dale Schuurmans, John Lafferty, Chris Williams, and Aron Culotta, 1339–1347. 2009. http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2009_0807.pdf.

Pape, Leo, Faustino Gomez, Mark Ring, and Jürgen Schmidhuber. "Modular Deep Belief Networks That Do Not Forget." In *Proceedings of the 2011 International Joint Conference on Neural Networks*, 1191–1198. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 2011.

Poon, Hoifung, and Pedro Domingos. "Sum-Product Networks: A New Deep Architecture." In *Proceedings of the 2011 IEEE International Conference on Computer Vision Workshops,* 689–690. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 2011.

Ranzato, Marc'Aurelio, and Martin Szummer. "Semi-Supervised Learning of Compact Document Representations with Deep Networks." In *ICML '08: Proceedings of the 25th International Conference on Machine Learning*, edited by Andrew McCallum and Sam Roweis, 792–799. New York: ACM, 2008.

Rosenblatt, Frank. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." *Psychological Review* 65, no. 6 (1958): 386–408.

Sainath, Tara N., Brian Kingsbury, Bhuvana Ramabhadran, Petr Fousek, Petr Novak, and Abdel-rahman Mohamed. "Making Deep Belief Networks Effective for Large Vocabulary Continuous Speech Recognition." In *Proceedings of the 2011 IEEE Workshop on Automatic Speech Recognition and Understanding,* edited by Thomas Hain and Kai Yu, 30–35. Piscataway, NJ: Institute of Electrical and Electronics Engineers, 2011.

Salakhutdinov, Ruslan, and Geoffrey Hinton. "Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure." In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, edited by Marina Meila and Xiaotong Shen, 412–419. 2007. http://jmlr.csail.mit.edu/proceedings/papers/v2/salakhutdinov07a/salakhutdinov07a.pdf.

Salakhutdinov, Ruslan, and Geoffrey Hinton. "Deep Boltzmann Machines." In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, edited by David van Dyk and Max Welling, 448–455. 2009. www.jmlr.org/proceedings/papers/v5/salakhutdinov09a/salakhutdinov09a.pdf.

Salakhutdinov, Ruslan, and Hugo Larochelle. "Efficient Learning of Deep Boltzmann Machines." In *Proceedings of the 13th Annual International Conference on Artificial Intelligence and Statistics*, edited by Yee Whye Teh and Mike Titterington, 693–700. 2010. www.dmi.usherb.ca/~larocheh/publications/aistats_2010_dbm_recnet.pdf.

Schmidhuber, Jurgen. "Learning Complex, Extended Sequences Using the Principle of History Compression." *Neural Computation* 4 (1992): 234–242.

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." In *Interspeech 2011: Proceedings of 11th Annual Conference of the International Speech Communication Association,* edited by Piero Cosi, Renato De Mori, Giuseppe Di Fabbrizio, and Roberto Pieraccini, 437–440. 2011. www.isca-speech.org/archive/interspeech_2011.

Smolensky, Paul. "Information Processing in Dynamical Systems: Foundations of Harmony Theory." In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1, edited by David E. Rumelhart, James L. McClelland, and the PDP Research Group, 194–281. Cambridge, MA: Massachusetts Institute of Technology Press, 1986.

Susskind, Joshua M., Geoffrey E. Hinton, Javier R. Movellan, and Adam K. Anderson. "Generating Facial Expressions with Deep Belief Nets." In *Affective Computing: Focus on EmotionExpression, Synthesis and Recognition*, edited by Jimmy Or, 421–440. Vienna: I-Tech, 2008.

Uetz, Rafael, and Sven Behnke. “Locally-Connected Hierarchical Neural Networks for GPU-Accelerated Object Recongition.” In *Proceedings of the NIPS 2009 Workshop on Large-Scale Machine Learning Parallelism and Massive Datasets*. 2009.

Werbos, Paul. “Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.” PhD thesis, Harvard University, 1974.

Weston, Jason, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. “Deep Learning via Semi-Supervised Embedding.” In *Neural Networks: Tricks of the Trade, Second Edition,* edited by Grégoire Montavon, Geneviève Orr, and Klaus-Robert Müller, 639–655. Berlin: Springer, 2012.

Wulsin, D. F., J. R. Gupta, R. Mani, J. A. Blanco, and B. Litt. “Modeling Electroencephalography Waveforms with Semi-Supervised Deep Belief Nets: Fast Classification and Anomaly Measurement.” *Journal of Neural Engineering* 8, no. 3 (2011): 036015.

Zhou, Shusen, Qingcai Chen, and Xiaolong Wang. “Active Deep Networks for Semi-Supervised Sentiment Classification.” In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, edited by Chu-Ren Huang and Dan Jurafsky, 1515–1523. Stroudsburg, PA: Association for Computational Linguistics, 2010.