Chapter 18

Boltzmann Machines

In the last chapter, we introduced principal components analysis (PCA), an approach that allowed us to *learn* useful low-dimensional representations. However, the key limitation of PCA is that the mapping function—which generates the low-dimensional representations—is linear. Indeed, one can view PCA as the optimal linear model for learning low-dimensional representations, since it corresponds to the (orthonormal) linear map that minimizes the reconstruction error.

In this chapter, we will introduce a different approach for representation learning, termed *Boltzmann machines*, which work on binary datasets. Boltzmann machines can allow us to learn *non-linear* low-dimensional representations of binary data. As we will see, however, this can come at a significant computational cost. Boltzmann machines will also illustrate connections between representation learning and latent variable models, and Boltzmann machines will serve as our entryway to the topic of neural networks.

18.1 A Network of Binary Units

The basic idea behind a Boltzmann machine is that we build a network of interconnected binary "neurons" or "units" (Figure 18.1). Each unit i is connected to all other units in the network based on a weighted connection, and we denote the connection strength between unit i and j as $w_{i,j}$. The total input z_i to a unit i is then given by

$$z_i = b_i + \sum_{j \in \mathcal{S}} w_{i,j} s_j. \tag{18.1}$$

Here, s_j denotes the binary state of the unit j and $\mathcal{S}$ denotes the full set of units in the network.

We can then compute the probability that unit i has a binary state of one based on the logistic function:

$$P(s_i = 1) = \frac{1}{1 + e^{-z_i}}. \tag{18.2}$$

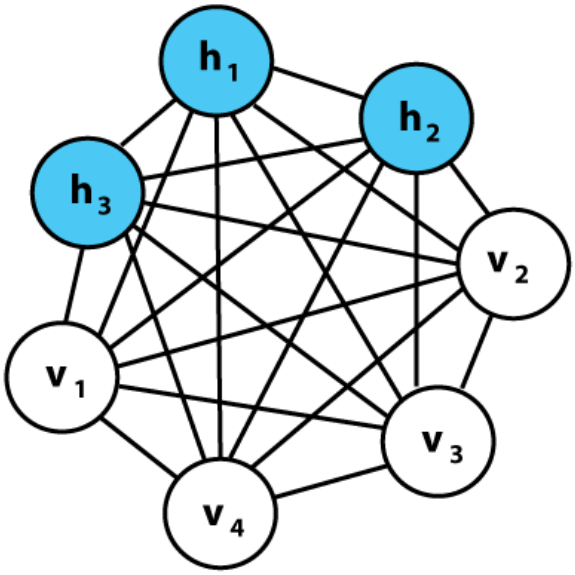

Figure 18.1: Illustration of a Boltzmann machine.

Thus, we have that the state of every unit depends on the state of all the other units in the network (in a probabilistic manner), and the strength of the connection between two units mediates their interdependence.

We denote a binary assignment of all the units in the network with a vector $\mathbf{s} \in \{0,1\}^{|\mathcal{S}|}$, and we call this a *configuration* of the network. For example, $s_j = \mathbf{s}[j]$ would correspond to the state of unit i in the configuration specified by the vector $\mathbf{s}$.

18.1.1 Sampling to an equilibrium

A key aspect of Boltzmann machines is the fact that all the units are interdependent. This means that the model is never (or at least very rarely) characterized by a particular configuration. Instead, the network is characterized by a distribution of configurations, which depend on the interconnections between the different units.

We can determine the distribution that characterizes a particular network by continually sampling the binary states of individual units. The basic strategy is as follows:

1. Start at $t = 0$ with a random configuration $\mathbf{s}^{(t)}$.
2. Sample a random unit $i \in \mathcal{S}$.
3. Sample a new state $s_j^{(t+1)}$ of unit i based on Equation 18.2 and set

$\mathbf{s}^{(t+1)}[j] = s_j^{(t+1)}$.

4. Iterates steps 2 and 3 until the distribution stabilizes.

If we run this sampling process long enough, we will eventually reach what is known as the equilibrium distribution.

18.1.2 Energy of a configuration

An important property of the equilibrium distribution of a Boltzmann machine is that the probability of a configuration only depends on the *energy* of that configuration. The energy $E(\mathbf{s})$ of a configuration in a Boltzmann machine is given by

$$E(\mathbf{s}) = -\sum_{i \in \mathcal{S}} \mathbf{s}[i] b_i - \sum_{j \in \mathcal{S}: j > i} w_{i,j} \mathbf{s}[i] \mathbf{s}[j], \tag{18.3}$$

and we can compute the probability of a configuration at the equilibrium distribution as

$$P(\mathbf{s}) = \frac{e^{-E(\mathbf{s})}}{\sum_{\mathbf{s}' \in \{0,1\}^{|\mathcal{S}|}} e^{-E(\mathbf{s}')}} \tag{18.4}$$

The energy essentially tells us how likely a particular distribution is, and it can be derived as a consequence of the individual unit equations (i.e., Equations 18.1 and 18.2). Low energy states are more likely, while high energy states are less likely. In terms of individual units, the basic idea is that we are more likely to set $\mathbf{s}[i] = 1$ if unit i has a large bias term and if this unit has a strong connection with other units that are active.

18.2 Learning in a Boltzmann Machine

Learning in a Boltzmann machine involves optimizing the weight values $w_{i,j}$ in order to achieve a desired equilibrium distribution. In particular, our goal is to model the distribution $P(\mathbf{x})$ of some binary-valued feature vector $\mathbf{x} \in \{0,1\}^m$. However, in order to understand how this learning proceeds, we first must introduce the notions of visible and hidden units.

Visible and hidden units

We can divide the units in a Boltzmann machine into two mutually exclusive categories: visible units and hidden units, denoted by the sets $\mathcal{V}$ and $\mathcal{H}$, respectively (where $\mathcal{V} \cup \mathcal{H} = \mathcal{S}$ and $\mathcal{V} \cap \mathcal{H} = \emptyset$). The visible units correspond to the dimensions of our input features. In other words, we have that $|\mathcal{V}| = m$ and there is a one-to-one mapping between the visible units $i \in \mathcal{V}$ and the feature indices $i \in [m]$. The hidden units, on the other hand, are uncoupled from the input data. These hidden units correspond to latent variables that can explain or represent the data.

We can sample conditioned on visible data by *clamping* the visible units to particular set of observed values and using the sampling process in Section

18.1.1 to only sample the hidden units. In this way, we can get a distribution of the hidden units conditioned on the visible units being set to some particular observed values.

Thus, the challenge for learning in a Boltzmann machine is to optimize the weights so that the equilibrium distribution matches a given data distribution on the visible units, while allowing the hidden units to be unconstrained. We can formalize this by computing the probability over an input feature based the probabilities learned for the visible units

$$P_\Theta(\mathbf{x}) = \sum_{\mathbf{h}' \in \{0,1\}^{|\mathcal{H}|}} P(\mathbf{v} = \mathbf{x}, \mathbf{h} = \mathbf{h}'), \tag{18.5}$$

where we marginalize over the hidden units. Here, we use $\mathbf{v}$ to denote the portion of the configuration vector $\mathbf{s}$ that represents the visible units and $\mathbf{h}$ to denote the portion that represents the hidden units (i.e., $\mathbf{s} = [\mathbf{v}, \mathbf{h}]$). As usual, we subscript by Θ to indicate that this is a learned distribution, in this case based on the parameters $\Theta = \{b_i, w_{i,j}, \forall i = 1, ..., |\mathcal{S}|, j < i\}$.

We can then attempt to maximize the log-likelihood of a given dataset, i.e.,

$$\log \mathcal{L}(\mathcal{D}, \Theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log(P_\Theta(\mathbf{x})). \tag{18.6}$$

By learning in this way, the hidden units can be used as latent variables or low-dimensional representations that explain the data.

Gradient descent in a Boltzmann machine

In order to optimize the Boltzmann machine to maximize the data likelihood, we can use gradient descent. The gradient update can be derived by differentiating the log-likelihood based on Equations 18.4 and 18.5 and using the fact that $\frac{\partial w_{i,j} \mathbf{s}[i]\mathbf{s}[j]}{\partial w_{i,j}} = \mathbf{s}[i]\mathbf{s}][\mathbf{j}]$. Overall, the gradient update for a connection weight in the Boltzmann machine is given by

$$\frac{\partial \log \mathcal{L}(\mathcal{D}, \Theta)}{\partial w_{i,j}} = \mathbb{E}_{\text{data}} \left[\mathbf{s}[i]\mathbf{s}[j]\right] - \mathbb{E}_{\text{model}} \left[\mathbf{s}[i]\mathbf{s}[j]\right]. \tag{18.7}$$

Here, the first term

$$\mathbb{E}_{\text{data}} \left[\mathbf{s}[i]\mathbf{s}[j]\right] \tag{18.8}$$

is the expected value of $\mathbf{s}[i]\mathbf{s}[j]$ when we sample from the network according to the data distribution (i.e., with the visible units clamped to a uniformly sampled training example $\mathbf{x} \in \mathcal{D}$). The second term

$$\mathbb{E}_{\text{model}} \left[\mathbf{s}[i]\mathbf{s}[j]\right] \tag{18.9}$$

is what we get from the unconstrained equilibrium sampling process.

Intuitively, the gradient update depends on the difference between the distribution of configurations in the unconstrained model, compared to the distribution where the visible units are clamped to the training data. As expected, this

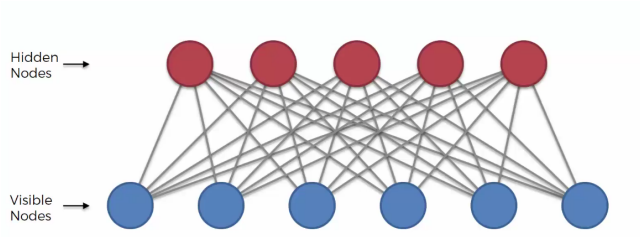

Figure 18.2: Illustration of a restricted Boltzmann machine.

would mean that the model would stop learning once these two distributions are identical.

It is also important to note that the gradient update for a weight $w_{i,j}$ only depends on the units i and j. This means that the weight updates are local in some sense. However, computing the expected values in the gradient update can involve many sampling iterations and is typically infeasible in networks with more than hundreds of units. The gradient update for a bias term b_i is analogous but with $\mathbf{s}[i]$ replacing $\mathbf{s}[i]\mathbf{s}[j]$.

18.3 Restricted Boltzmann Machines

The primary drawback of standard Boltzmann machines is their computational expense. In particular, it can take thousands of sampling iterations to reach the equilibrium distribution, which can make the gradient descent updates very expensive. In order to alleviate this computational expense, an important class of Boltzman machines, known as restricted Boltzmann machines (RBMs) have been proposed.

Unlike standard Boltzmann machines, RBMs restrict the connections between the units so that the only connections are between visible and hidden units, creating a bipartite graph of connections. In other words, RBMs do not allow connections between two visible units or between two hidden units. This idea is illustrated in Figure 18.2.

18.3.1 Learning in an RBM

The structure of the RBM makes learning far more straightforward. In particular, due to the connectivity structure, in an RBM all the hidden units are conditionally independent given the visible units. Similarly, all the visible units are conditionally independent given the hidden units. This means that sampling in an RBM is far more efficient than sampling in a standard Boltzmann machine. In particular, we can sample by the equilibrium distribution by iterating two steps: first, we sample all the hidden units based on the current values of the visible units; next, we sample all the visible units based on the current values of the hidden units. In each step of this two-step process, we can sample all the

hidden and visible units in parallel, which is far more efficient than a standard Boltzmann machine, which has to sequentially sample all the units.

Constrastive divergence

Based on the simplified nature of RBMs, a popular and efficient learning algorithm has been devised. Each step in this algorithm proceeds as follows:

1. Sample a datapoint $\mathbf{x} \in \mathcal{D}$ from the dataset.
2. Sample a configuration $\mathbf{s}_1$ of the network with the visible units clamped to the values given by the data vector $\mathbf{x}$.
3. Clamp the hidden units to the values from $\mathbf{s}_1$, re-sample the visible units with the hidden units clamped to get a value $\mathbf{v}'$, and then re-sample the hidden units with the visible units clamped to $\mathbf{v}'$. Call the final sampled configuration $\mathbf{s}_2$. It contains $\mathbf{v}'$ and the hidden units sampled conditioned on $\mathbf{v}'$.
4. Update each weight based on the update equation

$$w_{i,j}^{(t+1)} = w_{i,j}^{(t)} + \alpha(\mathbf{s}_1[i]\mathbf{s}_1[j] - \mathbf{s}_2[i]\mathbf{s}_2[j]), \tag{18.10}$$

   where α is a learning rate.

This algorithm is based on the same underlying principle as the standard Boltzmann machine learning algorithm, but it is far more efficient.

18.3.2 Stacking RBMs

RBMs are far more efficient than standard Boltzmann machines. However, this increased efficiency comes at significant cost, as RBMs can only learn linear representation functions. Indeed, each hidden unit in the RBM is essentially an independent logistic regression model. One way to overcome this limitation is by stacking multiple RBMs.

The basic idea in the *deep Boltzmann machines* is that we iteratively train a stack of RBMs. First, we train a single RBM on the input data. Next, we train a second RBM, where we keep the first RBM fixed and treat the output of the first RBM as our observations. We can then iterate this process to make a deep network of several RBM layers. In principle, such a model can represent complex non-linear functions. However, it is still a relatively challenging model to train, and—as we will see—there are more efficient and tractable ways to train multi-layer deep neural networks.