Statistical Mechanics for Neural Networks and Artificial Intelligence

Chapter 7: Introduction to Energy-Based Neural Networks: The Hopfield Network and the (Restricted) Boltzmann Machine

Alianna J. Maren
Northwestern University School of Professional Studies
Master of Science in Data Studies

Draft: 2019-05-15

7.1 Introduction to Energy-Based Neural Networks

One of the most important aspects of advanced machine learning / deep learning studies is the shift into a physics-based approach; specifically into *statistical mechanics*. Statistical mechanics, combined with Bayesian probability theory and also with neural network methods, contribute to the central themes of machine learning, as illustrated in the following Fig. 7.1.

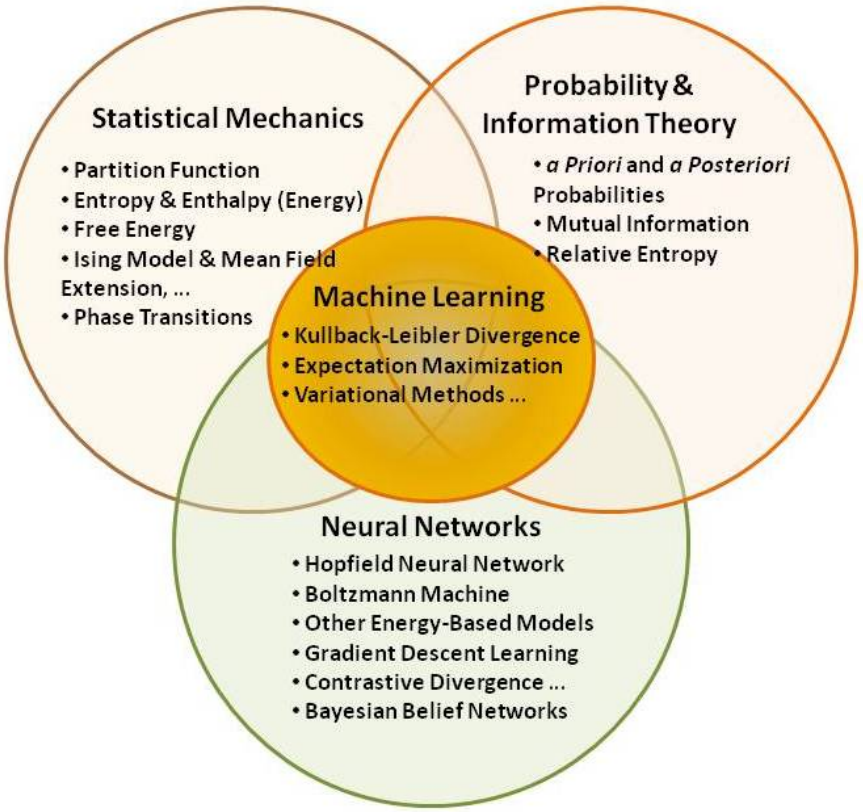

Figure 7.1: Machine learning algorithms are at the confluence of statistical mechanics, probability and information theory, and neural networks.

One of the most interesting, distinctive, and even arcane aspects about advanced neural networks and machine learning algorithms is that they use two very different forms of probability-thinking. These two different methods,

coming from statstical mechanics and Bayesian probabilities (respectively) are hugely different ways of thinking about the likelihood of whether or not something will happen.

Statistical mechanics, a realm of theoretical physics, is used in neural networks largely as an allegory; as a model created in one field that has been (very usefully) applied to another. It's almost like using physics as story-telling. The notion that these methods could be successfully used is so extreme that its almost shocking that these methods could find a new home in neural networks and deep learning.

The notions of statistical mechanics are central to the learning methods for restricted Boltzmann machines (RBMs). A restricted Boltzmann machine learns using a very different underlying approach than that used by stochastic gradient descent implementations (e.g., backpropagation). This means that RBMs can have multi-layered architectures and learn to distinguish between more complex patterns, overcoming the limitations of simple Multilayer Perceptrons (MLPs), as we previously discussed.

Statistical mechanics deals with the probabilities of occurrence of small units that can be distinguished from each other only by their energy states. In contrast, Bayesian probabilities provide a remarkably different way of thinking about the probabilities with which things can happen. Together, these two probability-oriented methods provide the foundations for advanced machine learning methods.

Now that we've identified the importance of both statistical mechanics and Bayesian methods, we will restrict our attention (for this chapter and the immediately-following ones) to statistical mechanics and its foundational relationship with neural networks. We'll pick up on the full confluence of statistical mechanics and Bayesian methods later, when we address more advanced topics.

The first time that the role of statistical mechanics became well-known in neural networks was when John Hopfield presented his work in 1982 [1]. His work drew on some similar lines of thinking developed by Little and colleagues in 1974 [2].

This chapter presents some of the key concepts in statistical mechanics; sufficient to understand the subject of some classic papers: Hopfield's original work (introducing what became known as the Hopfield network), and a few key works on the Boltzmann machine, developed by Geoffrey Hinton and colleagues.

7.2 Introduction to the Hopfield Neural Network and the Boltzmann Machine

The Hopfield neural network and its immediates successor, the Boltzmann machine (in both original and restricted forms) are instances of energy-based neural networks. They each achieve their desired connection weight values by minimizing an *energy equation*.

At first glance, the two networks do not seem to be structurally the same. However, they have a great deal in common, as we'll see shortly.

The following Fig. 7.2 illustrates both the Hopfield and the Boltzmann machine neural networks, so that we can easily compare the structures. The Hopfield neural network is shown on the left-hand-side, and the Boltzmann machine (in two different configurations, but still the same network) is shown in the center and right-hand-side graphs.

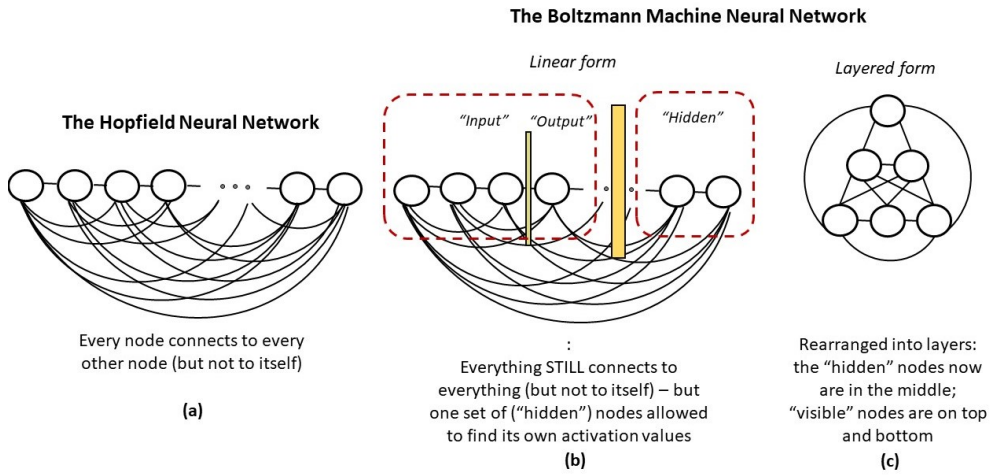

Figure 7.2: Illustration of the Hopfield and Boltzmann machine neural network archtiectures; the Boltzmann machine is essentially a Hopfileld neural network with certain connections removed.

To understand the restricted Boltzmann machine (RBM), which is central to current deep learning theory, it helps to first understand the simple (non-restricted) Boltzmann machine. And to understand the simple Boltzmann machine, it helps us to first understand the Hopfield neural network. Thus, we'll briefly examine the Hopfield neural network.

The Hopfield neural network, as it came to be known, was immediately interesting to the newly-forming neural networks community. Hopfield networks can be used for different tasks; one of the most interesting was as an optimization method. However, its first and most fundamental application was as an *autoencoder*. An autoencoder is a device that can remember previously-stored patterns, if triggered to their recall by presentation of a partial or noisy version of an original pattern.

Despite the Hopfield network's interesting ability to reconstruct stored patterns, it had a severe memory limitation. It could only learn a number of patterns that was about 15% of the total number of nodes in the system. So if, for example, a Hopfield network was created with 20 nodes (or neurons, or units), then it could learn and retrieve only three distinct patterns. This memory restriction caused many people to lose interest in this network.

Geoffrey Hinton, who (like John Hopfield) was also a physicist, came up with a novel insight into how the structure of the Hopfield neural network could be rearranged. This led to creation of the Boltzmann machine, and then the restricted Boltzmann machine (RBM), which has been the cornerstone of of deep learning.

So, in order to understand the restricted Boltzmann machine, we're going to start at the beginning - with the equations and structure of the Hopfield neural network. Once we understand that, it's a straightforward, natural, and intuitive step to understand Boltzmann machines - both in their original and restricted forms. This then paves the way for us to understand and use the wide range of methods involving energy-based systems in neural networks and machine learning.

7.3 The Hopfield Neural Network - Energy Equation and Structure

The Hopfield neural network, most often simply called the *Hopfield network*, is a beautiful instance of how ***form and function*** perfectly reflect each other. The *function* of this network is expressed in its energy equation. This energy equation is perfectly mirrored in its *form*, or the structure of this network.

We'll begin by looking at the energy equation, as presented by John Hopfield in his classic 1982 paper, shown in the following Fig. 7.3.

Notice that there is just one energy equation here, given as *Eqn. [7]* in the Hopfield 1982 paper. The second equation, *Eqn. [8]*, is an energy update equation; it describes how the energy is changed as the weight update rule is applied. Thus, our focus is on that first *Eqn. [7]*, as illustrated in Fig. 7.3.

Energy Equation Used by Hopfield (1982)

Studies of the collective behaviors of the model

The model has stable limit points. Consider the special case $T_{ij} = T_{ji}$, and define

$$E = -\frac{1}{2}\sum_{i \neq j}\sum T_{ij}V_iV_j \ . \qquad [7]$$

ΔE due to ΔV_i is given by

$$\Delta E = -\Delta V_i \sum_{j \neq i'} T_{ij}V_j \ . \qquad [8]$$

Thus, the algorithm for altering V_i causes E to be a monotonically decreasing function. State changes will continue until a least (local) E is reached. This case is isomorphic with an Ising model. T_{ij} provides the role of the exchange coupling, and there is also an external local field at each site. When T_{ij} is symmetric but has a random character (the spin glass) there are known to be many (locally) stable states (29).

Figure 7.3: Extract from J. Hopfield (1982, April). Neural networks and physical systems with emergent collective computational abilities, *Proc. Natl. Acad. Sci. U.S.A.*, *79*, pp. 2554-2558.

The first of the two equations shown in Fig. 7.3 defines the overall energy of the system, and the second shows what happens to the overall energy when we flip any given node from *1* to *0*, or vice versa. We will focus on the first equation here, and defer the second equation to a later chapter.

Before we interpret the first equation (*Eqn. [7]*) in Hopfield's paper, we're going to briefly note where he says "This case is isomorphic with an Ising model ..." This is a reference to statistical mechanics, which we'll address starting with the next chapter. An *Ising model* is a classic model in statistical mechanics, and is very relevant to our work in energy-based neural networks. We could say that the entirety of energy-based neural networks is built on a foundation that uses the Ising model as a starting point. We'll follow this

line of thought in the next few chapters. For now, we focus our attention on the first equation, *Eqn. [7]* in Hopfield's 1982 paper, as shown in Fig. 7.3.

For readability, the equation in Fig. 7.3 is reproduced here as Eqn. 7.1.

$$E = -\frac{1}{2}\sum_{i \neq j}\sum T_{i,j} V_i V_j. \tag{7.1}$$

Our first step in understanding this equation is to interpret the terms V_i, V_j, and $T_{i,j}$.We refer to Hopfield's original 1982 paper, where he states:

"The processing devices will be called neurons. Each neuron i has two states like those of McCulloch and Pitts (*Author's note:* the reference citation is updated here for the reader's benefit, see [3]): $V_i = 0$ ("not firing") and $V_i = 1$ ("firing at maximum rate"). When neuron i has a connection made to it from j, the strength of connection is defined as T_{ij}. (Nonconnected neurons have $T_{ij} \equiv 0$.)"

[*Author's Note: Minor notational difference - this book consistently puts a comma between indices of elements that have more than one index, as is done here with $T_{i,j}$. This is a common mathematical style, although neither Hopfield nor Hinton use the comma separations.*]

Let's examine both the energy equation for the Hopfield neural network, and the illustration of its structure. For ease in visualizing the Hopfield neural network, the depiction of it from from Fig. 7.2 is presented here at larger scale, as Fig. 7.4.

From Fig. 7.4, we see that every node is connected to each other node, but not to itself.

In the Hopfield neural network, we have connections between each of the different nodes. The notion of having some nodes be "visible" and other nodes "hidden" is not present in the Hopfield neural network; the notion of having "hidden" nodes was a innovation introduced a few years later by Geoffrey Hinton, and was essential to the notion of the Boltzmann machine. Essentially, all nodes in a Hopfield network are "visible."

The energy equation for the Hopfield neural network addresses all possible combinations of nodes in the network; this is why we see the double sumamtion sign in Eqn. 7.1. This equation has the values for two different nodes in it; we see both V_i and V_j. We know that there are no connections from any given node back to itself, because that would be a connection between V_i and V_i; for example node *1* connecting back to *1*. The equation explicitly states that we don't have these connections, because we see $i \neq j$ as the subscript under

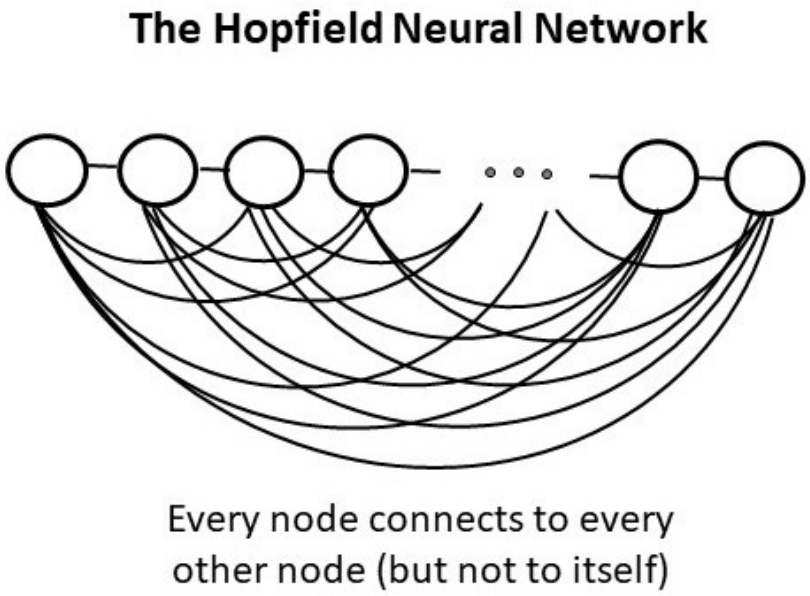

Figure 7.4: The Hopfield neural network archtiecture; expanded from Fig. 7.2.

the two summation signs. As we refer back to Fig. 7.4, we see this is the case; each node connects to each other node, but there are no "loops" connecting a node back to itself.

There's just two more things that we can glean about the structure and operation of the Hopfield neural network from its energy equation. First, we see that there is a multiplying factor of $-1/2$ in front of the summations. Also, from Hopfield's original work, from which we saw an excerpt in Fig. 7.3, we read "Consider the special case $T_{i,j} = T_{j,i}$ ..." This means that the connection strength between any two nodes is the same, whether we read it from the first node to the second or vice versa. For example, the connection strength going from node *2* to node *3* is the same as the connection strength going from node *3* back to node *2*.

The way that the equation is set up, we count each direction of connections. For example, we separately count the interactions between node *2* to node *3* ($T_{2,3}V_2V_3$) and between node *3* to node *2* ($T_{3,2}V_3V_2$). Because we've essentially counted the same thing twice, we need to divide by two; this is why we have the normalizing factor of $1/2$ in front of the double summation.

The negative sign is introduced so that we can have positive values for

the connection parameter $T_{i,j}$. Remember, our algorithms are going to seek a minimum in the energy equation. This is similar to how, when we did stochastic gradient descent using the backpropagation algorithm, we were seeking a minimum value. Our values for $T_{i,j}$ are not constrained to be positive, but putting the negative sign in front of the double summation energy term allows them to be positive more often than not, and the system will still (likely) come to a minimum state as we apply our energy-minimization algorithm.

Summing up what we've learned so far, we note that the Hopfield neural network, which is the predecessor neural network for the Boltzmann machine, has the following properties:

1. The network is constructed from a set of nodes, all of which are "visible,"
2. Each node connects to each other node, but not back to itself,
3. The nodes in this system are *binary*; meaning that they can each be in one of two states; "on" or "off;" for both the Hopfield and the Boltzmann machine networks, these values are $(1, 0)$,
4. The connections between nodes in this system are *symmetric*, so that $T_{i,j} = T_{j,i}$, and
5. The stable state of this network is governed by an *energy equation*, and the training algorithm adapts the connection parameter $T_{i,j}$ between each pair of nodes (V_i and V_j) in order to achieve a total minimum value.

We're not going to address the training algorithm right now (the energy-minimizing algorithm), as the purpose of this section was just to make a connection between the formalism of the energy equation and the structure of the Hopfield neural network. We've seen that, for the Hopfield neural network, ***form equals function***, in that the set of fully-connected nodes (without self-connection) is illustrated in both Fig. 7.4 and Eqn. 7.1.

The important thing that we've done here has been to lay a foundation, because our next step is to similarly understand the correspondence between the energy equation for the Boltzmann machine and the structure and nature of the Boltzmann machine neural network. That is the goal of the next section.

7.4 The Boltzmann Machine - An Energy-Based Neural Network

Eqn. 7.2, and the thinking behind it, is an evolution and a step forward from the idea encapsulated in the Hopfield neural network [1], which we just discussed in the previous section. Each of the first two terms involves a scalar multiplying a node activation; v_i or h_j. The third term is the only one in Eqn. 7.2 that has the energies of two different nodes involved; both v_i *and* h_j are involved.

We begin by taking a look, in Fig. 7.5, at an equation used by Geoffrey Hinton and colleagues to describe deep learning methods. The particular source for this figure is from Hinton et al. in a 2012 paper for acoustic modeling [4].

Energy Equation Used by Hinton et al. (2012)

B. An efficient learning procedure for RBMs

A joint configuration, $(\mathbf{v}, \mathbf{h})$ of the visible and hidden units of an RBM has an energy given by:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in \text{visible}} a_i v_i - \sum_{j \in \text{hidden}} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \quad (6)$$

where v_i, h_j are the binary states of visible unit i and hidden unit j, a_i, b_j are their biases and w_{ij} is the weight between them. The network assigns a probability to every possible pair of a visible and a hidden vector via this energy function as in Eqn. (5) ● ● ●

Figure 7.5: Extract from Hinton et al. (2012, November), Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal Processing Magazine, 29*, pp 82-97.

For readability, the equation shown in Figure 7.5 is reproduced as Eqn. 7.2.

$$E(v, h) = -\sum_{i \epsilon visible} a_i v_i - \sum_{j \epsilon hidden} b_j h_j - \sum_{i,j} v_i h_j w_{i,j} \quad (7.2)$$

Eqn. 7.2 describes the *energy* of a simple neural network as the linear combination of three negative terms. These three terms tell us a lot about the nature and structure of the corresponding neural network, which is called the *restricted* Boltzmann machine.

The Boltzmann machine is derived from the Hopfield neural network, and as we read the text in Fig. 7.5, see that Hinton et al. identify the terms in the Boltzmann machine, stating that " v_i, h_j, are the binary states of visible unit v_i and hidden unit h_j ...". In the Boltzmann machine, we have some units that are "visible," and others that are "hidden." By way of comparison, in the Hopfield neural network, all units are "visible." Thus, when we write an equation about the Boltzmann machine, we make a distinction between visible and hidden units, with the terms v_i and h_j. When we write about the Hopfield neural network, we only need to write about the visible nodes, and so we just have the term v_i. And actually, Hopfield uses the notation V_i; it's the same thing.

Before we examine the restricted Boltzmann machine (RBM), we'll first take a look at the general Boltzmann machine - and the notion of clamping, which Hinton and colleagues have used in relation to the training process for a Boltzmann machine (whether general or restricted).

7.4.1 "Clipping" and "Clamping" in the Hopfield and the Boltzmann Machine Neural Networks

One of the terms that David Ackley, Geoffrey Hinton, and Terrence Sejnowski introduced, in their 1985 paper presenting a learning algorithm for the early Boltzmann machine [5], was the notion of *clamping*. This term has been used by colleagues and devotees of Hinton ever since, and someone new to the field often has to figure out what this means via inference and interpretation.

Let's begin by looking at both the Hopfield network and the (general) Boltzmann machine, as shown in Fig. 7.6. In a *general* Boltzmann machine, just as with the Hopfield network, there are connections between all the nodes.

Very succinctly, the notion of "clamping" is something that makes sense with a Boltzmann machine neural network, but not with a Hopfield network. The idea is that during training, for a Boltzmann machine, only the *visible* nodes will have a pattern superimposed on them. This is called "clamping" a pattern (onto the visible nodes). The training progresses by adjusting connection weight strengths and also the *on/off* values of the hidden nodes, for all the different patterns that are *clamped* onto the hidden nodes. This is done many, many times, for all the different pattern instances.

As a historical note, Hopfield [1] used the term "clipping." However, a

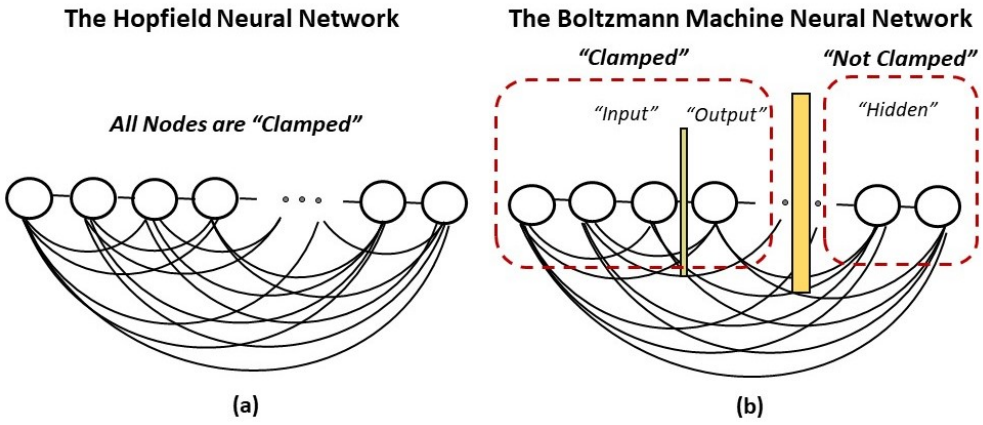

Figure 7.6: The Boltzmann machine.

Hopfield neural network is not trained in the same way as is a Boltzmann machine, and the notion of "clipping" is not quite the same as "clamping." In Hopfield's usage of the term, "clipping" means actually replacing the values for $T_{i,j}$ with ± 1, in order to examine what would happen with highly nonlinear connection weights.

We'll pick up on the actual training methods for the restricted Boltzmann machine in a later chapter, after we've looked at some of the statistical mechanics underlying both the Hopfield and Boltzmann machine networks.

7.4.2 Comparing Interaction Energy Terms in the Hopfield and Boltzmann Machine Neural Networks

To understand how the Boltzmann machine evolved from the Hopfield network, let's compare the third term in Eqn. 7.2 with the Hopfield energy equation. This third term from Eqn. 7.2 is

$$E(v,h)_{term3} = -\sum_{i,j} v_i h_j w_{i,j}.$$

In this equation, v_i refers to the *energy* of the visible node i, and h_j refers to the *energy* of the hidden node j. The third element of this term is $w_{i,j}$, which refers to the connection weight between node i and node j.

For comparison, the Hopfield energy equation is

$$E = -\frac{1}{2}\sum_{i \neq j}\sum T_{i,j} V_i V_j.$$

We might be tempted to say that these are the same equation, with only some small differences in the notation. And really, these *are* two different ways of expressing the same equation - with one ***very important difference*** in the two forms. First, though, we identify the similarities.

The equations both have negative signs in front of them. As we previously discussed, this lets us use positive values for (most of) the connection weights, expressed in the Hinton equation as $w_{i,j}$ and in the Hopfield equation as $T_{i,j}$.

There is a factor of 1/2 in the Hopfield equation; this can be easily subsumed into the connection weights themselves. (If nothing else were a factor, we might say that the values for $T_{i,j}$ would be about one-half the values for $w_{i,j}$.)

The summations are essentially the same; the double subscript indicates that there are two summations going on, whether or not we show the capital *sigma* summation sign twice or only once.

The real difference is a bit more subtle. Note that in the Hopfield equation, the two nodes involved in each double summation step are represented as $V_i V_j$. These are not only the same *kind* of node; they're drawn from the same *pool* of nodes. This is why Hopfield had to be careful to specify $i \neq j$, so that the energy equation did not include a connection from a node back to itself.

In contrast, the nodes in the Hinton et al. equation are of two distinct groups; one represented as v_i and the other as h_j. These are the same *kind* of node, but they are separated into *two distinct pools*, as shown in the following Fig. 7.7.

The set of node connections for a Boltzmann machine is isomorphic with that of a Hopfield network, as the notion of "visible" and "hidden" nodes has to do with creating the input training patterns and with the actual learning algorithm. That is, the structure (*form*) of the Boltzmann machine and Hopfield neural networks are essentially the same; they differ in that for a Boltzmann machine, the values for hidden nodes are not specified in the training and testing process; rather, these values are learned over time. For the Hopfield neural network, the values for all nodes are specified within the training data patterns.

In contrast, a restricted Boltzmann machine (RBM) has a structure that

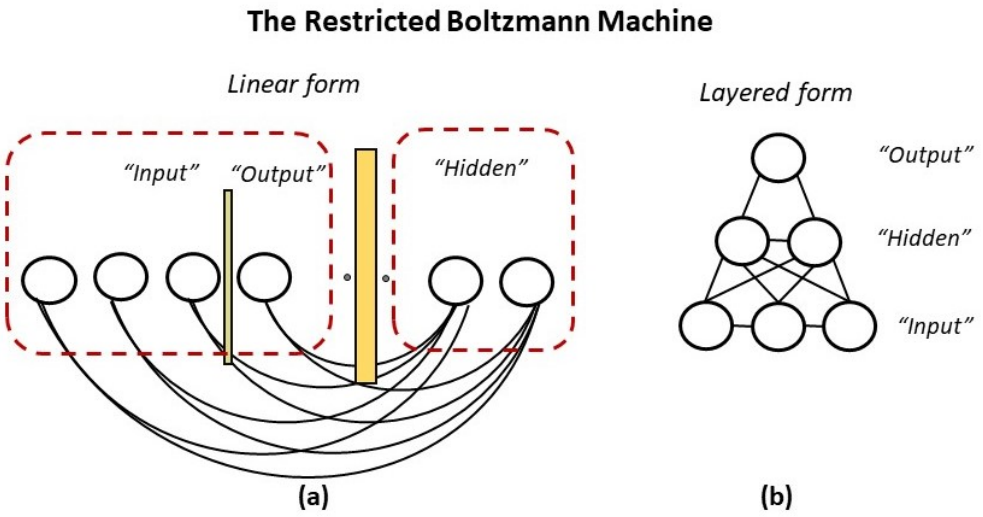

Figure 7.7: The restricted Boltzmann machine (RBM); (a) linear form, analagous to the Hopfield network depiction earlier, and (b) layered form, analagous to a Multilayered Perceptron.

is isomorphic with a Multilayer Perceptron (MLP). This is worth further exploration, which we'll reserve until a later section.

For now, we'll summarize by saying that the restricted Boltzmann machine has much in common with the energy formulation for the Hopfield neural network, but in practical application will function very similarly to a MLP. However, it's the energy-formulation for the RBM that allows it to be the foundational method for deep learning. This energy-based foundation is what has allowed certain challenges assocaited with simple gradient descent methods to be overcome, making it possible to build highly-layered network structures.

7.4.3 The Boltzmann Machine Energy Equation: The Two Linear Terms

All of our attention, up until now, has been given to the nonlinear term in the Boltzmann machine energy equation - the term that included *both* v_i and h_j. We focused on using this term as a basis for comparing the Boltzmann machine to its predecessor; the Hopfield neural network.

Now, we turn our attention to the remaining two terms in the Boltzmann

machine energy equation, both of which are linear in terms of a given node's activation. These two terms correspond, respectively, to the v_i and h_j nodes. For convenience, this portion of the energy equation is reproduced below as

$$E(v,h)_{terms1,2} = -\sum_{i \epsilon visible} a_i v_i - \sum_{j \epsilon hidden} b_j h_j.$$

Our first observation is that the processes involving single nodes are separated according to node type; this is really just a bit of a formality, because the two remaining terms are similar. They each identify that a single parameter (a_i or b_j) multiplies the "energy" of a single node (a *visible* or *hidden* node, respectively).

Pragmatically, there are some implications here.

One important quick note, which we'll develop further in the next section: although we're seeing different parameters multiplying the activations of the visible and hidden nodes, respectively, these parameters are not the same as the bias terms in a MLP. The bias terms act to "slide" the overall inputs into a hidden or output node into the range that gives the greatest slope in the output; that is, the biases learn to put the input values into a range near zero.

The linear terms in the Boltzmann energy equation have an entirely different role. They help drive the network towards a stronger negative value. In other words, they help to minimize the energy of teh network.

7.4.4 A Little Thought-Experiment

Let's pretend, for a moment, that the network has been trained, and so all the values for a_i and b_j are now fully determined. Let's also pretend that the a_i and b_j are largely positive. This doesn't have to be the case, but we'll play with this notion to get a sense of intution.

Suppose that a partial pattern is presented ("clamped") onto the network, so that a relatively large fraction of the visible nodes are defined. The network will respond by using its trained values of $w_{i,j}$ to create activations across the hidden nodes. These hidden nodes, now activated, will use their respective connections to the remaining visible nodes to induce activations in those nodes. At the end, all the nodes are activated (or not) in a pattern that has been "learned" in response to repeated presentation of the full set of visible node activations during training.

During the training process, the network will learn valeus for a_i and b_j that reduce the overall energy value, regardless of which pattern is presented ("clamped on") to the visible nodes. (Or, when in actual operation, partially clamped - so that the remaining visible nodes can take on values resulting from the network's processes using the values of the clamped visible nodes.)

The network can also learn which hidden nodes should become active.

If, in our imaginary scenario, all the b_j values are positive, the it would reduce the overall energy of the network to have all of the hidden nodes h_j to be "on," for each pattern presented by a set of v_i.

However, if all of the h_j are "on," that means that all of the possible learned features are active for each pattern presented by a given set of visible nodes.

Obviously, this isn't what we want. The whole point of training the network and having the hidden nodes to learn features (specific combinations of active visible nodes) is that certain features distinguish certain specific kinds of visible node patterns. Thus, the last thing that we want is to have all the hidden nodes active; we want just the *right* ones active at a given time.

Carrying our *gedanken-expereiment* (German for "thought-experiment") just a little further, suppose that we have a relatively *sparse* pattern presented in the v_i; that is, relatively few v_i nodes activated.

Getting a hidden node activated depends on having active v_i nodes, because the network is trained with the energy term $v_i h_j w_{i,j}$. If the visible node v_i activation is zero, then the energy term connecting that visible node to the hidden node is also zero, and there is no build-up for the connection weight $w_{i,j}$.

In short, a sparse visible pattern will also induce a sparse activation in the hidden nodes. This means that the total energy is likely to be *higher*, not *lower*, because we're playing with the imaginary scenario where the multiplying parameters a_i or b_j are all positive, and there is a negative sign in front of each of them in the energy equation.

This means that the sparse patterns will not be the dominant force in driving the system to the lowest energy state possible. Instead, the more activation-rich patterns will hold stronger sway. In the more activation-rich patterns, the linear term involving the v_i will be at a stronger negative value (minimizing the network's energy). Also, the term involving the h_j will also have a stronger negative value, because more active visible nodes will mean more pattern complexity, which will mean more features to which the hidden nodes can respond, so there will be more active hidden nodes.

Further, when presented with an activation-rich visible pattern, the non-linear term ($v_i h_j w_{i,j}$) will also reach a strong negative value, since there will be many combinations of visible and hidden nodes where both of the nodes v_i and h_j are active at the same time. With positive values for $w_{i,j}$ for those pairwise node combinations, this will also yield a strong negative contribution to the energy.

We are not necesarily confining the parameters a_i, b_j, and $w_{i,j}$ to exclusively positive values. As a result of the training, they can take on negative values as well. We've worked with positive values just to conduct some mental envisionings of different scenarios, and to test how we understand how these parameters influence the overall energy.

The actual training for the (restricted) Boltzmann machine is done using the *contrastive divergence* algorithm, which is an iterative two-part training process. We'll study this algorithm in a later chapter, after we've looked at the statistical mechanics underpinnings of the energy formulation.

7.5 Relationship Between the Boltzmann Machine and the Multilayer Perceptron

For those who have become accustomed to thinking about neural networks using the structure of a Multilayer Perceptron (MLP) as their reference point, the notion of an energy-based network and the concepts behind it may at first seem not only unusual, but also a bit jarring.

The notion of separating a network, formally, into only *two* distinct groups of nodes (*visible* and *hidden*) comes via the evolution of energy-based neural networks. In contrast, when we think about a Multilayer Perceptron, we necessarily have to think about at least three different sets of nodes; *input*, *hidden*, and *output*.

In its most essential form, a Boltzmann machine is an *autoencoder*. Correspondingly, in its most essential form, an MLP is a *classifier*.

We can make a Boltzmann machine function as a classifier, of course, and conversely, we can also make a MLP function as an autoencoder. However, the essential nature of these two different networks means that we think of them in different ways. They are not really analogues.

This shows up most strongly when we think about how we can use the Boltzmann machine. We can force it to reconstruct or complete a pattern

that has been presented to it during training. (Or, if it is a new pattern, it will construct something that is the closest approximation from its training data, even if that requires merging the patterns inherent in several different training instances.)

As far as the Boltzmann machine is concerned, it makes no difference if the partically-completed pattern that we "clamp" onto the network puts values into the output layer instead of the input layer, or into a mixture of both. In contrast, the only place where we enter a pattern (completed or not) into an MLP is in the input layer.

The reason for this difference is that in the Boltzmann machine, the notion of "input" and "output" layers are an artifact; they're something of our imagining. The only real distinction that the Boltzmann machine makes is between *visible* and *hidden*, and the *visible* nodes are separated into "input" and "output" layers just in terms of how we're drawing the network; not in any real functional sense.

This has a profound impact on how the network performs, and is inherent in how deep learning has its roots.

In concluding this chapter, we'll take a deeper look at the differences between the Boltzmann machine and a typical MLP.

7.5.1 The *Microstructure* of Energy-Based Neural Networks

In a classic MLP, the activation of a hidden or output node is found by applying a transfer function to the summed inputs to that respective node. The transfer function is typically one that has is smoothly differentiable. (We're ignoriing, for now, non-smoothly differentiable transfer functions such as ReLUs.) As we recall from a previous chapter, the transfer function serves multiple purpposes. It scales the output of a given node from what could potentially range from minus to posibive infinty, to a range between *0* and *1*. (Or between *-1* and *1*, depending on the function of choice.) The important point for us to keep in mind, as we do our comparision between MLP and a Boltzmann machine architectures, is that the output of both hidden and output nodes of the MLP can take on continuous values between specific ranges.

In contrast, the values of both the visible and hidden nodes in a Boltzmann machine are either *0* and *1*.

Because we know that in a Multilayer Perceptron (MLP) architecture, we have bias terms multiplying the activation of the hidden and output nodes, we can make a correspondence. In a MLP, the a_i and b_j parameters function as bias scalars, or multiplying factors.

In a Boltzmann machine, both "input" and "output" nodes are *visible*; that is, they belong to the pool of v_i nodes. If we wanted to, we could make the bias values for the input nodes to be set equal to one; this would make the Boltzmann machine look a bit more like an MLP. However, the formalism expressed in Eqn. 7.2 makes it clear that we have the flexiblity to do otherwise.

7.5.2 The Impact of the Training Method on the Node Microstructures

The key differentiating factor between Boltzmann machine-style neural networks and the class of neural networks that are trained using a stochastic gradient descent method (e.g., a Multilayer Percptron, using a backpropagation implementation for gradient descent training) is that the individual nodes or neurons in a Boltzmann machine network are binary; they each have values of *0* or *1*. In contrast, the nodes in an MLP must necessarily take on a continuum of values, with outputs that are bounded by *0* or *1* (or, depending on the transfer function used, between *-1* or *1*).

This is hugely related to how the training algorithms work in the two different cases. With a stochastic gradient descent, a key feature is that we need to identify the gradient of the node's activation, as a function of summed and weighted inputs. With a Boltzmann machine, we don't need a gradient; in fact, that would make our training algorithm more cumbersome.

We may come across a figure depicting the probabilistic activation of a node in a Boltzmann machine network. *(Author's Note: This figure will be inserted at a later date.)*

This distinction is, of course, a broad and sweeping generalization. There are exceptions to this, as there are to every rule. However, this distinction broadly separates these two fundamentally different neural network classes.

7.5.3 The *Mesostructure* of Energy-Based Neural Networks

The structure of the network described by Eqn. 7.2 may look identical to that of the Multilayer Perceptron (MLP), discussed in previous chapters. There is, however, a key difference. In a traditional MLP, the notion of various "layers" in the networks is an essential part of how we think about the network; both its training and its performance.

By contrast, in the networks described by Hinton and by others doing energy-based neural networks and deep learning, the emphasis is not so much on the structuring into layers, and more on whether a given node is "visible" or "hidden." By way of comparison to an MLP architecture, we would say that the input and output nodes of a network are ''visible," and of course the internal, hidden layer nodes are the ones that are "hidden." This lets us separately describe the energies for the visible (input and output) nodes versus the energies of the hidden ones.

When we train a Boltzmann machine, we superimpose a set of activations on both the input and output layers. (This is what Hinton means by "clamping.") The training algorithm causes the weights to take on appropriate values, and the hidden nodes to also learn whether to be "on" or "off" in response to a specific input / output training pattern. However, we can regard the combined set of input/output training data sets as being - essentially - all one pattern.

In contrast, in an MLP, we think of the input/output training data as two sets of patterns; an input training data set and its associated output pattern.

Of course, once these two networks are trained, they functionally operate in a very similar manner. Also, their performance is about the same.

7.6 Summary of Essential Features: The Hopfield and (Restricted) Boltzmann Machine Neural Networks

In this chapter, we've introduced the Boltzmann machine, in both its general and restricted forms. We've also shown how the Boltzmann machine is both an logical and intuitive move beyond its predecessor; the Hopfield neural network. The restricted Boltzmann machine (RBM) is architecturally very

similar to a Multilayer Perceptron (MLP), and can function in a very similar manner, with aprpoximately the same results.

As each of these networks is trained, both the RBM and the MLP use their respective hidden layers to represent patterns in the input data. However, to be more precise, in the RBM, the hidden nodes learn features that represent patterns across the combination of "input" and "output" nodes, which together comprise the *visible* nodes in the network. The notion of dividing the network into layers (input, hidden, and output) is an artifact. In essence, for the RBM, there are only the two categores of *visible* and *hidden* nodes.

In contrast, in an MLP, the notion of "layers" is essential to not just the architecture, but to the learning in the network.The training process is driven between the difference ("summed squared error") between the actual values and the desired values in the output nodes. This difference is used to guide weight adjustments sequentially; first to the hidden-to-output connection weights, and then to the input-to-hidden connection weights.

In order to train an MLP, we need to know in advance the desired output values; we *must* have a set of training data with pre-defined desired outputs corresponding to specific input patterns.

In contrast, we have a bit more ambiguity in defining how a Boltzmann machine is trained, because we don't necessarily *have* to have those pre-defined output classes in as rigidly-defined a manner. Thus, the training algorithm for the Boltzmann machine can concentrate more on extracting features in the imposed traiining patterns that are inherent to those patterns, leaving the ultimate "classification" to be more of a discovery, and less of a superimposition. This has had a profound impact in making deep learning possible.

Bibliography

[1] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 79, p. 2554 2558, April 1982.

[2] W. Little, "The existence of persistent states in the brain," *Math Biosci.*, vol. 19, pp. 101–120, February 1974.

[3] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp. 115–133, 1943.

[4] G. Hinton, L. Deng, D. Yu, G. Dah, A. R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Proc. Mag.*, vol. 29, pp. 82–97, November 2012.

[5] D. Ackley, G. Hinton, and T. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive Science*, vol. 9, pp. 147–169, 1985.