CMPTG 5 - Statistical Physics of Neural Networks

Sidharth Kannan

April 2025

1 Lecture 3 - Boltzmann Machines

1.1 Recap and Loose Threads

Last week, we derived an expression for the Boltzmann distribution, which is the probability distribution over microstates for a system in thermal equilibrium with a reservoir. We saw that while it is in principle possible to compute probabilities of occupation with this distribution, it is in practice impossible, due to the intractability of the partition function.

To mitigate this issue, we proposed the mean field approximation, in which we assume that the energy levels are independent, so that their marginal probabilities are tractable. We then assume some "average interaction", adjust the energies, and repeat. This gave rise to an iterative algorithm which produced a similar distribution to the Boltzmann distribution.

We then removed the physics from our discussion, and instead considered a general system of coupled binary units, and saw that if we assume a particular form for the energy, then this system's distribution of states will converge to the Boltzmann distribution. In particular, we used this to define a kind of invertible logic gate. This is where we will pick up.

In particular, we will start by highlighting two aspects of our invertible circuit that were glossed over last time: temperature, and Gibbs sampling. Once this is done, we will move on to how we can use these circuits as generative models.

Let's start with the p-bit equations:

$$I_i = \sum_j J_{ij} m_j + h_i \tag{1}$$

$$m_i = \operatorname{sgn}\left(\tanh \beta I_i + \operatorname{rand}_U[-1, 1]\right) \tag{2}$$

The first thing to note is the effect of this parameter β. β is defined as $\frac{1}{T}$, and so intuitively, we would expect that low β correspond to high temperatures, where all states become equally likely, whereas high β corresponds to low temperature, where the network state becomes frozen. This is correct, and we can see that by looking at the behavior of the hyperbolic tangent.

The next thing to note is the issue of update order, and parallelism. We are iterating through our network, computing the equations above, and updating our p-bit state. We can ask the following questions:

1. Can I update spins in any order? *Yes.*
2. Can I update spins in parallel? *Carefully.*

To see why we have to be careful with parallel spin updates, let's consider the simple, two spin ferromagnetically coupled Ising. If we perform parallel updates when the model starts in state [-1,1], it will stay in that state, which is incorrect.

This is a killer. If we can't update in parallel, we are relegated to slow for-loop type programming forever, which kills the efficiency of our network. It turns out that we can do better with *blocked* Gibbs sampling.

Next, we will discuss simulated annealing, and parallel tempering.

1.2 Boltzmann Machines and Log-Likelihood

Now that we have a decent understanding of Ising machines, it's time to finally tackle the learning problem. Given a dataset, how can we determine the correct $\mathbf{J}$ matrix and $\mathbf{h}$ matrix, such that our Ising machine will sample from the distribution we desire.

Given some set of parameters, $\boldsymbol{\theta}$, which in our case are the weights and biases, we wish to maximize the *likelihood* of our training samples. That is, we wish to maximize $\mathcal{P}(\{\mathbf{x}\}|\boldsymbol{\theta})$ for some training set $\{\mathbf{x}\}$. It is often more convenient to maximize the quantity known as the *log-likelihood*, $\log \mathcal{P}(\{\mathbf{x}\}|\boldsymbol{\theta})$.

In the case of our Boltzmann machine, this actually takes a fairly simple form. We know that our machine samples from the Boltzmann distribution, and so for any training state $\mathbf{m}$, the log-likelihood becomes

$$\log \mathcal{P}(\mathbf{x}|\boldsymbol{\theta}) = \log\left(\frac{1}{Z}e^{-E_{\boldsymbol{\theta}}(\beta \mathbf{s}))}\right) \tag{3}$$

$$= -\log Z - E_{\boldsymbol{\theta}}(\beta \mathbf{s}) \tag{4}$$

$$= -\log Z + \beta \sum_i \sum_{j<i} J_{ij} m_i m_j + \beta \sum_i h_i m_i \tag{5}$$

When considering the entire training set, we simply have to take expectation of the states against the dataset, like so. Here, I also drop the β, since it can be absorbed into either the weights or the biases.

$$\mathcal{L} \equiv \langle \log \mathcal{P}(\mathbf{x}|\boldsymbol{\theta})\rangle_{data} = -\log Z + \sum_i \sum_{j<i} J_{ij}\langle m_i m_j\rangle_{data} + \sum_i h_i \langle m_i\rangle_{data} d \tag{6}$$

So, in order to maximize the log-likelihood of our data, we need to perform gradient *ascent* on this function.

Before we get to that, I would like to point out a few things:

1. It may not be obvious how to evaluate the expectations in that formula, much less the partition function, but bear with me for a moment. It turns out we can make a bit more analytical progress before we turn our attention to that.

2. This general procedure of maximizing the log likelihood is how most generative models work. We have taken two specific inductive biases here. First, we have say that the functional form of $\mathcal{P}$ will be that of the Boltzmann distribution, and second, we have chose a specific form for the energy function in that distribution. If you change the form of the energy function, you will get a different class of *energy based model*. If you change the form of $\mathcal{P}$, perhaps not even

insisting on an analytic form for it, then you will get different kinds of generative models, such as variational autoencoders, normalizing flow models, etc.

Okay, back to Boltzmann machines. We want to evaluate the following derivatives: $\frac{\partial \mathcal{L}}{\partial J_{ij}}, \frac{\partial \mathcal{L}}{\partial h_i}$. Let's start with the first one.

$$\frac{\partial \mathcal{L}}{\partial J_{ij}} = -\frac{\partial}{\partial J_{ij}} \log Z + \frac{\partial}{\partial J_{ij}} \left(\sum_i \sum_{j<i} J_{ij} \langle m_i m_j \rangle_{data} + \sum_i h_i \langle m_i \rangle_{data} \right) \tag{7}$$

Let's start with the first term again.

$$-\frac{\partial \log Z}{\partial J_{ij}} = -\frac{1}{Z} \frac{\partial Z}{\partial J_{ij}} = -\frac{1}{Z} \frac{\partial}{\partial J_{ij}} \sum_{\text{states}} \exp \left(-\sum_i \sum_{j<i} J_{ij} m_i m_j - \sum_i h_i m_i \right) \tag{8}$$

$$= \frac{1}{Z} \sum_{\text{states}} m_i m_j \exp \left(-\sum_i \sum_{j<i} J_{ij} m_i m_j - \sum_i h_i m_i \right) \equiv -\langle m_i m_j \rangle_{model} \tag{9}$$

Now let's take a look at the second term in the derivative.

$$= \sum_{\text{training samples}} \frac{\partial}{\partial J_{ij}} \left(\sum_i \sum_{j<i} J_{ij} m_i m_j + \sum_i h_i m_i \right) \tag{10}$$

$$= \langle m_i m_j \rangle_{data} \tag{11}$$

Putting it together, we get that

$$\frac{\partial \mathcal{L}}{\partial J_{ij}} = \frac{1}{N} \sum_{\text{training samples}} \langle m_i m_j \rangle_{data} - \langle m_i m_j \rangle_{model} \tag{12}$$

That is, the derivative log likelihood of the training dataset with respect to the connection weight between neurons i and j is the difference between their average correlation in the training data minus their average correlation in the current model. Similar arguments show that

$$\frac{\partial \mathcal{L}}{\partial h_i} = \frac{1}{N} \sum_{\text{training samples}} \langle m_i \rangle_{data} - \langle m_i \rangle_{model} \tag{13}$$

Now comes the question of how to evaluate these expectations. In the case of the data, it is simple. We can just apply the definition of the two point correlator, and see that

$$\langle m_i m_j \rangle = \mathbf{u}\mathbf{u}^T \tag{14}$$

where $\mathbf{u}$ is the matrix of training samples.

However, in the case of the model, things are harder because we see that the partition function makes an appearance. Thus, we have to sum over all 2^N states of our Boltzmann machine. This is obviously not scalable. In practice, we can use MCMC algorithms to approximate the average correlation.

Now, this doesn't entirely solve the problem. As we saw in the last lecture, even with Gibbs sampling, each neuron must be updated in series. If you did question 3 on the homework from last week, you saw that things are slightly better, in that we can update disconnected spins in parallel. So, can we structure the Boltzmann machine in a way that allows us to more efficiently sample?

1.3 The Restricted Boltzmann Machine

Let's say that instead of letting arbitrary nodes be connected, we split our graph into two layers: the visible layer and hidden layer. See Fig. 1.

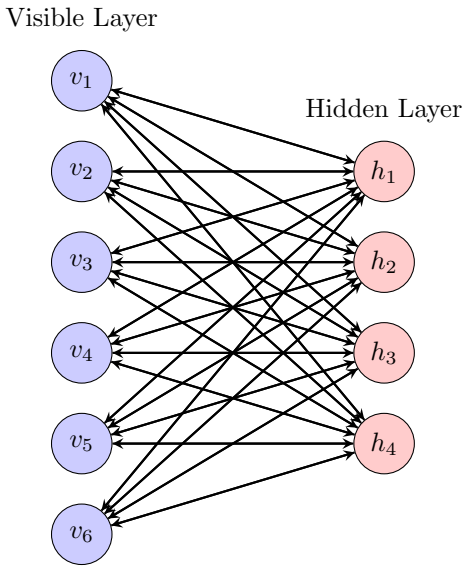

Figure 1: A schematic of a Restricted Boltzmann Machine.

To make this a bit more concrete, say that we are trying to train an RBM to generate photos of hand-written digits, like those in MNIST. Here, our visible units would correspond to the pixels in the image, and the hidden units correspond to latent features.

Then, the rules for the derivatives simplify to

$$\frac{\partial \mathcal{L}(\mathbf{v})}{\partial J_{ij}} = \langle h_i v_j \rangle_{data} - \langle h_i v_j \rangle_{model} \tag{15}$$

That is, for each training sample, $\mathbf{v}$, we can now just consider the correlations between the hidden and visible units. We can also now update all of the hidden nodes in parallel, and all of the visible nodes in parallel, since we have enforced that they are disconnected.

When training an RBM, the algorithm most commonly used is known as *contrastive divergence*. The gradient of the log-likelihood for an RBM includes two terms: the expectation of the data, and the expectation of the model

The expectation over the model requires computing expectations over all possible configurations of visible and hidden units, which is computationally infeasible for even medium size models. Contrastive divergence approximates the gradient by:

1. Starting a Markov chain from the data.
2. Running a few steps (often just one, known as CD-1) of Gibbs sampling.

3. Using the difference between data-driven and model-driven statistics to update the parameters.

In more detail,

Step 1: Positive Phase (Data-driven) Given a data point $v^{(0)}$:

1. Clamp the visible units to $v^{(0)}$. Compute the hidden unit activation probabilities:

$$p\left(h_j = 1 \mid v^{(0)}\right) = \sigma\left(c_j + \sum_i W_{ij} v_i^{(0)}\right),$$

   where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the logistic function.

2. Sample the hidden activations $h^{(0)}$ from these probabilities.

Step 2: Negative Phase (Reconstruction)

1. Reconstruct the visible units using the sampled hidden states:

$$p\left(v_i = 1 \mid h^{(0)}\right) = \sigma\left(b_i + \sum_j W_{ij} h_j^{(0)}\right).$$

   Sample a new visible vector $v^{(1)}$ from these probabilities.

2. Recompute the hidden activations using the reconstructed visible vector:

$$p\left(h_j = 1 \mid v^{(1)}\right) = \sigma\left(c_j + \sum_i W_{ij} v_i^{(1)}\right).$$

   Sample these to obtain $h^{(1)}$.

Step 3: Parameter Update The weight update is based on the difference between the data-driven and model-driven (reconstruction) statistics:

$$\Delta W \propto \langle v^{(0)} h^{(0)} \rangle - \langle v^{(1)} h^{(1)} \rangle.$$

Similarly, the biases are updated as:

$$\Delta b \propto v^{(0)} - v^{(1)} \quad \text{and} \quad \Delta c \propto h^{(0)} - h^{(1)}.$$

Now there are a few things to note here:

1. The contrastive divergence algorithm does not wait for the Markov chain to equilibrate.

2. It does not directly maximize the likelihood; instead it minimizes another objective called the KL-divergence between the model distribution and data distribution.