Boltzmann Machine

A.L. Yuille. JHU. 2016.

Abstract

1. Introduction

The Boltzmann Machine (Hinton and Sejnowski) is a method for learning the weights of a probability distribution assuming that a subset of nodes (input nodes) are observed and the remainder are hidden.

Gibbs Distribution

The probability of the system of N neurons $\vec{S} = (S_1, ..., S_N)$, where each S_i takes value 0 or 1, is defined by a Gibbs distribution with energy $E(\vec{S}) = \frac{-1}{2} \sum_{ij} \omega_{ij} S_i S_j$ and distribution:

$$P(\vec{S}) = \frac{1}{Z} \exp\{-E(\vec{S})/T\}. \tag{1}$$

States/configurations $\vec{S}$ with low energy $E(\vec{S})$ will correspond to high probability $P(\vec{S})$. This depends on the parameter $T \geq 0$. When Gibbs distributions are used in Statistical Mechanics then T corresponds to the temperature of the system. In this course, T is a parameter that characterizes the degree of uncertainly of the distribution. For very small T, $T \approx 0$, the distribution will be strongly peaked at those configurations $\vec{S}$ that minimize the energy. But for large T, the distributions becomes less peaked (although the relative ordering of which states are more probable does not change as T varies. The value of T is not important for Boltzmann Machines, but it is for other applications (see the book chapter handout for previous lecture which describes how it used for annealing).

The term Z is a normalization constant defined so that $\sum_{\vec{S}} P(\vec{S}) = 1$. Hence $Z = \sum_{\vec{S}} \exp\{-E(\vec{X})/T\}$. It is important to realize that it is often *impossible to compute Z because it requires summing over an exponential number of configurations* (each neuron S_i takes two values, $\vec{S}$ contains N neurons, so $\vec{S}$ takes 2^N possible values). This means that *although we can compute $E(\vec{S})$ we cannot compute $P(\vec{S})$* . This makes doing computations involving these distributions difficult, e.g., computing $\sum_{\vec{S}} \vec{S} P(\vec{S})$, and is a reason why Gibbs sampling was invented. It enables us to obtain samples $\vec{S}^1, ..., \vec{S}^m$ from $P(\vec{S})$ and hence approximate $\sum_{\vec{S}} \vec{S} P(\vec{S})$ by $\frac{1}{m} \sum_{i=1}^{m} \vec{S}^i$. This will be used by Boltzmann Machines.

Note that Z is a function of the weights w_{ij} of the energy function $E(\vec{S})$. This has important mathematical implications, such as $\frac{\partial \log Z}{\partial w_{ij}} = \sum_{\vec{S}} S_i S_j P(\vec{S})$, which arise in the derivation of the Boltzmann Machine. But these are much more general than Boltzmann Machines and appear frequentlt in machine learning and statistical physics.

2. Gibbs Distribution for the Boltzmann Machine

We divide the nodes into two classes $\mathcal{V}_o$ and $\mathcal{V}_h$, which are the observed (input) and hidden nodes respectively. We use $\vec{S}_o$ and $\vec{S}_h$ to denote the states of the observed and the hidden nodes respectively. The components of $\vec{S}_o$ and $\vec{S}_h$ are $\{S_i : i \in \mathcal{V}_o\}$ and $\{S_i : i \in \mathcal{V}_h\}$ respectively. Hence $\vec{S} = (\vec{S}_o, \vec{S}_h)$.

Hence we can re-express the distribution over the states as:

$$P(\vec{S}_o, \vec{S}_h) = \frac{1}{Z} \exp\{-E(\vec{S})/T\}. \tag{2}$$

The marginal distribution over the observed nodes is

$$P(\vec{S}_o) = \sum_{\vec{S}_h} \frac{1}{Z} \exp\{-E(\vec{S})/T\}. \tag{3}$$

We assume that we can estimate a distribution $R(\vec{S}_0)$ of the observed nodes (see more in a later section). Then the goal of learning is to adjust the weights $\vec{\omega}$ of the model (i.e. the $\{\omega_{ij}\}$) so that the marginal distribution $P(\vec{S}_o)$ of the model is as similar as possible to the observed model $R(\vec{S}_0)$. This requires specifying a similarity criterion which is chosen to be the Kullback-Leibler divergence:

$$KL(\vec{w}) = \sum_{\vec{S}_o} R(\vec{S}_o) \log \frac{R(\vec{S}_o)}{P(\vec{S}_o)} \tag{4}$$

We will discuss in a later section how this relates to the standard maximum likelihood criterion for learning distributions (it is effectively the same).

The Boltzmann Machine adjusts the weights by the iterative update rule:

$$w_{ij} \mapsto w_{ij} + \Delta w_{ij} \tag{5}$$

$$\Delta w_{ij} = -\delta \frac{\partial KL(\vec{w})}{\omega_{ij}} \tag{6}$$

$$\Delta w_{ij} = -\frac{\delta}{T}\{< S_i S_j >_{clamped} - < S_i S_j >\} \tag{7}$$

Here δ is a small positive constant. The derivation of the update rule is given in a later section. The way to compute the update rule is described in the next section.

$< S_i S_j >_{clamped}$ and $< S_i S_j >$ are the expectation (e.g., correlation) between the state variables S_i, S_j when the data is generated by the *clamped* distribution $R(\vec{S}_o)P(\vec{S}_h|\vec{S}_o)$ and by the distribution $P(\vec{S}_o, \vec{S}_h)$ respectively. I.e. $< S_i S_j >= \sum_{\vec{S}} S_i S_j P(\vec{S})$. The conditional distribution $P(\vec{S}_h|\vec{S}_o)$ is the distribution over the hidden states conditioned on the observed states. So it is given by $P(\vec{S}_h|\vec{S}_o) = P(\vec{S}_h, \vec{S}_o)/P(\vec{S}_o)$.

The learning rule, equation (7), has two components. The first term $< S_i S_j >_{clamped}$ is Hebbian and the second term $< S_i S_j >$ is anti-Hebbian (because of the sign). This is a balance between the activity of the model when it is driven by input data (i.e. clamped) and when it is driven by itself A wild speculation is that the Hebbian learning is done when you are awake, hence exposed to external stimuli, while the anti-Hebbian learning is done when you are asleep with your eyes shut but, by sampling from $P(\vec{S}_o|\vec{S}_h)$ you are creating images, or dreaming.

The algorithm will convergence when the model accurately fits the data, i.e.. when $< S_i S_j >_{clamped} =< S_i S_j >$ and the right hand side of the update rule, equation (7), is zero.

What is the observed distribution $R(\vec{S}_o)$? We do not know $R(\vec{S}_o)$ exactly and so we approximate it by the *training data* $\{\vec{S}_o^{\mu}; \mu = 1, ..., N\}$. This is equivalent to assuming that

$$R(\vec{S}) = \frac{1}{N} \sum_{\mu=1}^{N} \delta(\vec{S}_o - \vec{S}_o^{\mu}) \tag{8}$$

.

(We return to this later when we show the relationship to maximum likelihood learning).

2.1. Estimating the $< S_i S_j >$

The main difficulty of the learning rule for the Boltzmann Machine is how to compute $< S_i S_j >_{clampaed}$ and $< S_i S_j >$.

To do this, it is natural to use Gibbs sampling. Recall from earlier lectures that the stochastic update rule for neurons is performing Gibbs sampling – i.e. selecting a neuron i at random, and then sampling S_i from the conditional distribution $P(S_i|\vec{S}_{/}i)$.

By performing Gibbs sampling multiple times on the distribution $P(\vec{S}_o, \vec{S}_h)$ we obtain M samples $\underline{\vec{S}}^1, ..., \underline{\vec{S}}^M$. Then we can approximate $< S_i S_j >$ by:

$$< S_i S_j > \approx \frac{1}{M} \sum_{a=1}^{M} \underline{S}_i^a \underline{S}_j^a \tag{9}$$

Similarly we can obtain samples from $R(\vec{S}_o)P(\vec{S}_h|\vec{S}_o)$ (the clamped case) by first generating samples $\underline{\vec{S}_o}^1, ..., \underline{\vec{S}_o}^M$ from $R(\vec{S}_0)$ and then converting them to samples

$$\underline{\vec{S}}^1, ..., \underline{\vec{S}}^M \tag{10}$$

where $\underline{\vec{S}} = (\underline{\vec{S}_o}^i, \underline{\vec{S}_h}^i)$, and $\underline{\vec{S}_h}^i$ is a random sample from $P(\vec{S}_h|\vec{S}_o)$, again performed by Gibbs sampling.

How do we sample from $R(\vec{S}_o)$? Recall that we only know samples $\{\vec{S}_o^\mu; \mu = 1, ..., N\}$ (the training data). Hence sampling from $R(\vec{S}_o)$ reduces to selecting one of the training examples at random.

The fact that the Boltzmann Machines uses Gibbs sampling is a big limitation. If the model is complicated – i.e. there are many hidden nodes and weights ω_{ij} – then Gibbs sampling can take a long time to converge. This means the calculating the learning rule, equation (7), becomes impractical. We can approximate the expectations $< S_i S_j >$ by equations (9,10), but these approximations are sometimes very bad.

This means that Boltzmann Machines are of limited effectiveness. In a later section we discuss Restricted Boltzmann Machines (RBMs) for which we can perform efficient sampling and hence estimate the $< S_i S_j >$ and $< S_i S_j >_{clamped}$ effectively. RBMs are used as components to build one type of deep neural networks.

Note that in some accounts of Boltzmann Machines say that the BMs have to run to reach *thermal equilibrium*. This is equivalent to saying that Gibbs sampling yields samples from $(\vec{S}_o, \vec{S}_h)$ (and from $P(\vec{S}_h|\vec{S}_o)$).

3. Derivation of the BM update rule

To justify the learning rule, equation (7), we need to take the derivative of the cost function $\partial KL(\vec{\omega})/\partial\omega_{ij}$.

$$\frac{\partial KL(\vec{w})}{\partial \omega_i j} = -\sum_{\vec{S}_o} \frac{R(\vec{S}_o)}{P(\vec{S}_o)} \frac{\partial P(\vec{S}_o)}{\partial \omega_{ij}} \tag{11}$$

Expressing $P(\vec{S}_o) = \frac{1}{Z} \sum_{\vec{S}_h} \exp\{-E(\vec{S})/T\}$, we can express $\frac{\partial P(\vec{S}_o)}{\partial \omega_{ij}}$ in two terms:

$$\frac{1}{Z} \frac{\partial}{\partial \omega_{ij}} \sum_{\vec{S}_h} \exp\{-E(\vec{S})/T\} - \frac{1}{Z} \sum_{\vec{S}_h} \exp\{-E(\vec{S})/T)\} \frac{\partial \log Z}{\partial \omega_{ij}} \tag{12}$$

which can be re-expressed as:

$$\frac{-1}{T} \sum_{\vec{S}_h} S_i S_j P(\vec{S}) + \{\sum_{\vec{S}_h} P(\vec{S}) \frac{1}{T} \sum_{\vec{S}} S_i S_j P(\vec{S})\} \tag{13}$$

Hence we can compute:

$$\frac{\partial P(\vec{S}_o)}{\partial \omega_{ij}} = \frac{-1}{T} \sum_{\vec{S}_h} S_i S_j P(\vec{S}) + P(\vec{S}_o) \frac{1}{T} \sum_{\vec{S}} S_i S_j P(\vec{S}) \tag{14}$$

Substituting equation (14) into equation (11) yields

$$\frac{\partial KL(\vec{w})}{\partial \omega_i j} = \frac{1}{T} \sum_{\vec{S}_h, \vec{S}_o} S_i S_j \frac{P(\vec{S})}{P(S_o)} R(\vec{S}_o) - \frac{1}{T} \{ \sum_{\vec{S}_o} R(\vec{S}_o) \} \sum_{\vec{S}} S_i S_j P(\vec{S}) \tag{15}$$

Which can be simplified to give:

$$\frac{\partial KL(\vec{w})}{\partial \omega_i j} = \frac{1}{T} \sum_{\vec{S}} S_i S_j P(\vec{S}_h | \vec{S}_o) R(\vec{S}_o) - \frac{1}{T} \sum_{\vec{S}} S_i S_j P(\vec{S}) \tag{16}$$

Note this derivation requires $\partial \log Z / \partial w_{ij} = \sum_{\vec{S}} S_i S_j P(\vec{S})$.

4. How does the Boltmann Machine relate to Maximum Likelihood Learning?

They are equivalent.

The Kullback-Leibler criterion, equation (4), can be expressed as

$$KL(\vec{\omega}) = \sum_{\vec{S}} R(\vec{S}_o) \log R(\vec{S}_o) - \sum_{\vec{S}} R(\vec{S}_o) \log P(\vec{S}_h | \vec{S}_o) \tag{17}$$

Only the second term depends on $\vec{\omega}$ so we can ignore the first (since we want to minimize $KL(\vec{\omega})$ with respect to $\vec{\omega}$).

Using the expression for $R(\vec{S}_o)$ in terms of the training data, equation (8), we can express the second term as:

$$-\frac{1}{N} \sum_{\vec{S}_o} \frac{1}{N} \sum_{a=1}^{N} \delta(\vec{S}_o - \vec{S}_o^a) \log P(\vec{S}_o) \tag{18}$$

$$-\frac{1}{N} \frac{1}{N} \sum_{a=1}^{N} \log P(\vec{S}_o^a) \tag{19}$$

This is precisely, the Maximum Likelihood criterion for estimating the parameters of the distribution $P(\vec{S}_o)$.

More generally, estimating the parameters of a distribution by Maximum Likelihood can always be expressed in terms of minimizing a Kullback-Leibler divergence. Why do we care? Because it gives a richer way to think of Maximum Likelihood (ML). The standard justification for ML (from Bayes Decision Theory, or in classic Statistics) is to say that it is an asymptotically consistent estimator of the model parameters $\vec{\omega}$ provided that the data has really been generated by the probability model. This is ok, but it gives no justification for using ML if are not using the right distribution (i.e. if the data is generated by something else). But the Kullback-Leibler formulation says that you are learning the best distribution that your model allows ("best in the sense of K-L divergence). So this justifies using ML even if the distribution is only an approximation to the true model that has generated the data (which you probably never know).

5. Restricted Boltzmann Machines

RBMs are a special case of Boltmann Machines where there are no weights connecting the hidden nodes to each other. In this case the energy can be expressed as:

$$E(\vec{S}) = \sum_{i \in \mathcal{V}_o,\, j \in \mathcal{V}_h} \omega_{ij} S_i S_j. \tag{20}$$

This means that the conditional distributions $P(\vec{S}_h|\vec{S}_o)$ and $P(\vec{S}_o|\vec{S}_h)$ can both be factorized:

$$P(\vec{S}_o|\vec{S}_h) = \prod_{i \in \mathcal{V}_o} P(S_i|\vec{S}_h), \;\; P(\vec{S}_h|\vec{S}_o) = \prod_{j \in \mathcal{V}_h} P(S_j|\vec{S}_o) \tag{21}$$

where, for $i \in \mathcal{V}_o$, $P(S_i|\vec{S}_h) = \frac{1}{Z_i} \exp\{-(1/T) S_i (\sum_{j \in \mathcal{V}_h} \omega_{ij} S_j)\}$ with Z_i being the normalization constant $Z_i = \sum_{S_i \in \{0,1\}} \exp\{-(1/T) S_i (\sum_{j \in \mathcal{V}_h} \omega_{ij} S_j)\}$ – and with similar expressions for $P(S_j|\vec{S}_o)$ for $j \in \mathcal{V}_h$.

These factorized forms means that we can sample from $P(\vec{S}_o|\vec{S}_h)$ and $P(\vec{S}_h|\vec{S}_o)$ very rapidly (e.g., by sampling from simple distributions like $P(S_i|\vec{S}_h)$). This makes learning fast and practical. To estimate the terms $< S_i S_j >_{clamped}$ we only need to sample from $P(\vec{S}_h|\vec{S}_o)$, which is very fast. To estimate the terms $< S_i S_j >$, we can sample from $P(\vec{S}_o, \vec{S}_h)$ by alternatively sampling from $P(\vec{S}_o|\vec{S}_h)$ and $P(\vec{S}_h|\vec{S}_o)$. This is longer, because we have to do this multiple times until the sampling has converged (but it is much faster than doing Gibbs sampling for full Boltzmann Machines when there are weights connecting the hidden nodes).

This means that is practical to learn RBMs. But this is limited because RBMs are too simple to perform useful tasks. The solution (Hinton 2006) is to learn a hierarchy of RBMs, or a *deep network*. The strategy is simple. You train one RBM first using the training data. Then you train a second RBM which uses the output of the first RBM as input. Then you train a third RBM and so on. The idea is that these higher level RBMs compensate for the fact that there are no weights between output nodes of the RBMs.