Boltzmann Machine

Alan Yuille

Feb 5 2024

Boltzmann Machine: The Gibbs Distribution

- The probability distribution for 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:

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

- State configurations $\vec{S}$ with low energy $E(\vec{S})$ will correspond to high probabilities $P(\vec{S})$. Z is specified by the normalization condition $\sum_{\vec{S}} P(\vec{S}) = 1$, by $Z = \sum_{\vec{S}} \exp\{-E(\vec{S})/T\}$. The ω_{ij} are the weights of the distribution (like weights in a neural network) and are symmetric $\omega_{ij} = \omega_{ji}$ $\forall i, j$ with $\omega_{ii} = 0$, $\forall i$.
- The "temperature" T controls the "sharpness" of the distribution. For very small T, the distribution is strongly peaked about $\vec{S}^* = \arg\min_{\vec{S}} E(\vec{S})$. As T increases, the distribution becomes less peaked as T becomes large ($T \mapsto \infty$) all states become equally likely. Intuitively, T is similar to the variance.

The Boltzmann Machine is already in the form of an exponentioal distribution.

The parameters are the $\{\omega_{ij}\}$. The statistics are $\{S_i S_j\}$. This is an exponential distribution because some of the states s_i areobserved and others are hidden.

We can learn the parameters of the Boltzmann Machine by the general proceedure for learning exponential models described earlier in the course (using the EM algorithm to deal with the hidden variables). The Boltzmann Machine contains many closed loops so we cannot use algorithms like dynamic programming for inference. Instead we use Gibbs sampling for estimating quantities like the expected statistics.

But this was not known in 1983 when Hinton and Sejnowski invented the Boltzmann Machine.

Boltzmann Machine: Inference

- The inference task is to compute, or estimate, the most probably state(s) $\vec{S}^* = \arg\max_{\vec{S}} P(\vec{S}) = \arg\min_{\vec{S}} E(\vec{S})$. But this is impossible because $\vec{S}$ takes 2^N possible states and so we cannot simply evaluate the probability of every state and find the maximum, and similarly we cannot compute Z. (But there are a few special cases where computing $\vec{S}^*$ is possible).

- We have discussed two types of algorithm that can get approximate estimates of $\vec{S}^*$: (I) Gibbs Sampling. (II) Mean Field Theory.

- In this lecture we will be using Gibbs sampling. Recall that this: (i) initializes the states $\vec{S}$ randomly, (ii) selects a node i at random, (iii) samples s_i from the conditional distribution $P(s_i|\vec{S}/i) = \frac{\exp s_i\{\sum_j w_{ij} s_j\}}{1+\exp\{\sum_j w_{ij} s_j\}}$, and (iv) repeat (ii) and (iii).

- It can be shown that Gibbs sampling converges to samples $\vec{S}$ from $P(\vec{S})$. This implies that the final states will have high probabilities. So if we have a set $\{\vec{S}^n : n = 1, ..., N\}$ from $P(\vec{S})$ then they are likely to have high probabilities $\{P(\vec{S}^n) : n = 1, ..., N\}$ and be close to $\vec{S}^*$. Importantly, for this lecture, we can approximate the expected statistics of $P(\vec{S})$ by $< s_j s_j >= \sum_{\vec{S}} s_i s_j P(\vec{S}) \approx \sum_{n=1}^{N} s_i^n s_j^n$.

Boltzmann Machine: Learning

- Divide the nodes into two classes $\mathcal{V}_o$ and $\mathcal{V}_h$, which are the observed (input) and hidden nodes respectively. $\vec{S}_o$ and $\vec{S}_h$ 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. $\vec{S} = (\vec{S}_o, \vec{S}_h)$.
- We 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 estimate a distribution $R(\vec{S}_0)$ of the observed nodes (from the observed data $\{\vec{S}_o^n : n = 1, ..., N\}$ where N are the number of training examples). *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:
$$KL(\vec{w}) = \sum R(\vec{S}_o) \log \frac{R(\vec{S}_o)}{P(\vec{S}_o)} \tag{4}$$

Boltzmann Machine: The Learning Rule

- 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 later slides (so is how to compute the update rule).

- $< S_i S_j >_{clamped}$ 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 $P(\vec{S}_h|\vec{S}_o) = P(\vec{S}_h, \vec{S}_o)/P(\vec{S}_o)$.

Boltzmann Machine: Understanding the Learning Rule

- 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

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

.

Estimating the $< S_i S_j >$

- The Boltzmann Machine requires computing $< S_i S_j >_{clampaed}$ and $< S_i S_j >$. This is done by Gibbs sampling (earlier lectures). .
- 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.
- Gibbs sampling is not a very effective algorithm. So Boltzmann machines are hard to use in practice (with extra ingredients).

Derivation of the BM update rule (I)

- 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:

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

- This can be re-expressed as:

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

Derivation of the BM update rule (II)

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

Boltzmann Machine is Maximum Likelihood Learning

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

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

- 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 $P(\vec{S}_o)$. This shows that Maximum Likelihood is a good strategy to learn a distribution *even if we do not know the correct form of the distribution*. We are simply finding the best fit model.

Boltzmann Machine learns by Expectation-Maximization

- The Boltzmann Machine (BM) learning is a special case of the Expectation-Maximization (EM) algorithm. This algorithm can be applied to any learning problem where some variables are unobservable.
- For the BM, the distribution is $P(\vec{S}_o, \vec{S}_h; \omega)$ with observed data $\{\vec{S}_o^n : n = 1, ..., N\}$. We do not know the $\{\vec{S}_h^n : n = 1, ..., N\}$, so the $\vec{S}_h$ are *hidden, missing, or latent* variables.
- In theory we can compute the marginal distribution $P(\vec{S}_o; \omega) = \sum_{\vec{S}_h} P(\vec{S}_o, \vec{S}_h; \omega)$. Then we can learn the weights $\{\omega_{ij}\}$ by Maximum Likelihood: minimizing

$$-\sum_{n=1}^{N} \log P(\vec{S}_o; \omega), \quad \text{w.r.t. } \omega.$$

- The problem is that we cannot compute $P(\vec{S}_o; \omega)$ explicitly. This is where we need EM.

BM and EM: part 1

- We define a new (unknown) distributions $Q^n(\vec{S}_h) = \prod_{i=1}^m q_i^n(S_h^i)$ $n = 1, ..N$, where the $\{S_h^i : i = 1, .., m\}$ are the components of the hidden variables $\vec{S}_h$.
- We define a free energy:

$$\mathcal{F}(Q, \omega) = -\sum_{n=1}^{N} \log P(\vec{S}_o^n; \omega) + \sum_{n=1}^{N} \sum_{\vec{S}_h^n} Q^n(\vec{S}_h^n) \log \frac{Q^n(\vec{S}_h^n)}{P(\vec{S}_h^n | \vec{S}_o^n; \omega)}.$$

- This has two important properties. Firstly, we can minimize $\mathcal{F}(Q, \omega)$ with respect to each $Q^n(.)$ to obtain $Q^n(\vec{S}_h^n) = P(\vec{S}_h^n | \vec{S}_o^n; \omega)$. Substituting this value of $Q^n(.)$ back into $\mathcal{F}(Q, \omega)$ yields $-\sum_{n=1}^N \log P(\vec{S}_o^n; \omega)$.
- Therefore minimizing $\mathcal{F}(Q, \omega)$ with respect to Q and ω is equivalent to performing ML on $P(\vec{S}_o; \omega)$.
- This follows from the facts that $\sum_{\vec{S}} Q(\vec{S}) \log \frac{Q(\vec{S})}{P(\vec{S})} \geq 0$ and $= 0$ only when $Q(\vec{S}) = P(\vec{S})$.

BM and EM: part 2

- The second property is that we can minimize $\mathcal{F}(Q, \omega)$ by alternatively minimizing with respect to Q and to ω. This is the EM algorithm.
- Minimizing w.r.t. $Q(.)$ gives $Q^n(\vec{S}_h^n) = P(\vec{S}_h^n | \vec{S}_o^n; \omega)$.
- Minimizing w.r.t. ω gives:

$$\omega_{ij} = \arg\min \sum_{n=1}^{N} Q^n(\vec{S}_h^n) \log P(\vec{S}; \omega) = \arg\min -\{\sum_{n=1}^{N} Q^n(\vec{S}_h^n) E(\vec{S}) - \log Z(\omega)\}.$$

- This exploits $P(\vec{S}_h | \vec{S}_o; \omega) P(\vec{S}_o; \omega) = P(\vec{S}_h, \vec{S}_o; \omega)$.
- For the BM, these minimizations reduce to the BM learning rule (after some algebra). Gibbs sampling is needed to perform each step. Note: there is no guarantee that the EM algorithm will converge to the global optimum (i.e. to the real ML estimate).

The Restricted Boltzmann Machine

- RBMs are a special case of Boltmann Machines where there are no weights connecting the hidden nodes to each other with energy:

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

- The conditional distributions $P(\vec{S}_h|\vec{S}_o)$ and $P(\vec{S}_o|\vec{S}_h)$ can both be factorized:

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

- For $i \in \mathcal{V}_o$, $P(S_i|\vec{S}_h) = \frac{1}{Z_i} \exp\{-(1/T) S_i (\sum_{j \in \mathcal{V}_h} \omega_{ij} S_j)\}$. Z_i is 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 similarly for $P(S_j|\vec{S}_o)$ for $j \in \mathcal{V}_h$.

- These factorization 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 $P(S_i|\vec{S}_h)$). This makes learning fast and practical. Estimating $< S_i S_j >_{clamped}$ requires sampling from $P(\vec{S}_h|\vec{S}_o)$, which is very fast. Estimating $< S_i S_j >$, requires sampling 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 must be done multiple times until convergence (but it is much faster than Gibbs sampling).

- RBMs are too restricted to anything useful. But Hinton (2006) suggested stacking them on top of each other to create a Deep Network.