# 18 Deep Belief Networks

**Goal**

Belief network, sigmoid belief network, deep belief network, maximum likelihood, Gibbs sampling.

**Alert 18.1: Convention**

Gray boxes are not required hence can be omitted for unenthusiastic readers.
This note is likely to be updated again soon.

**Definition 18.2: DAG**

A directed acyclic graph (DAG) is a directed graph that contains no directed cycles. Note that the underlying undirected graph could still contain cycles.

**Definition 18.3: Notions in DAGs**

A useful property of a DAG is that we can always define a partial ordering on its vertices so that $u \leq v$ iff there is a directed path from $u$ to $v$. For each vertex $v$ in the directed graph $\mathcal{G} = (V, E)$, we use $\mathrm{pa}(v) := \{u : \overrightarrow{uv} \in E\}$, $\mathrm{ch}(v) := \{u : \overrightarrow{vu} \in E\}$, $\mathrm{an}(v) := \{u : \exists w_1, \dots, w_k \in V, \overrightarrow{uw_1}, \overrightarrow{w_1w_2}, \dots, \overrightarrow{w_kv} \in E\}$, $\mathrm{de}(v) := \{u : \exists w_1, \dots, w_k \in V, \overrightarrow{vw_1}, \overrightarrow{w_1w_2}, \dots, \overrightarrow{w_ku} \in E\}$, and $\mathrm{nd}(v) := V \setminus (\{v\} \cup \mathrm{de}(v))$ to denote the parents, children, ancestors, descendants, and non-descendants of $v$, respectively. Similarly we define such sets for a set of nodes by taking unions.

**Definition 18.4: Belief/Bayes networks (BNs) (Pearl 1986)**

For each vertex $v$ of a DAG $\mathcal{G} = (V = \{1, \dots, m\}, E)$, we associate a random variable $S_v$ with it. Interchangeably we refer to the vertex either by $v$ or $S_v$. Together $\mathbf{S} = (S_1, \dots, S_m)$ defines a Belief network w.r.t. $\mathcal{G}$ iff the joint density $\mathsf{p}$ factorizes as follows:

$$\mathsf{p}(s_1, \dots, s_m) = \prod_{v \in V} \mathsf{p}(s_v \mid \mathrm{pa}(s_v)). \tag{18.1}$$

Pearl, Judea (1986). "Fusion, propagation, and structuring in belief networks". *Artificial Intelligence*, vol. 29, no. 3, pp. 241–288.

**Theorem 18.5: Factorization of BNs**

*Fix a DAG $\mathcal{G}$. For any set of normalized probability densities $\{\mathsf{p}_v(s_v \mid \mathrm{pa}(s_v))\}_{v \in V}$,*

$$\mathsf{p}(s_1, \dots, s_m) = \prod_{v \in V} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v)) \tag{18.2}$$

*defines a BN over $\mathcal{G}$, whose conditionals are precisely given by $\{\mathsf{p}_v\}$.*

*Proof.* W.l.o.g. we may assume the nodes are so arranged that $u \in \mathrm{de}(v) \implies u \geq v$. This is possible since the graph is a DAG. We claim that for all $j$,

$$\mathsf{p}(s_1, \dots, s_j) = \prod_{1 \leq v \leq j} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v)).$$

The idea is to perform marginalization bottom-up. Indeed,

$$\begin{aligned}
\mathsf{p}(s_1,\dots,s_j) &:= \int \mathsf{p}(s_1,\dots,s_m)\,\mathrm{d}s_{j+1}\cdots\mathrm{d}s_m \\
&= \prod_{1\le v\le j} \mathsf{p}_v(\underbrace{s_v \mid \mathrm{pa}(s_v)}_{\text{no } s_{j+1},\dots,s_m}) \int \prod_{j+1\le v\le m} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v))\,\mathrm{d}s_{j+1}\cdots\mathrm{d}s_m \\
&= \prod_{1\le v\le j} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v)) \int \prod_{j+1\le v\le m-1} \mathsf{p}_v(\underbrace{s_v \mid \mathrm{pa}(s_v)}_{\text{no } s_m})\,\mathrm{d}s_{j+1}\cdots\mathrm{d}s_{m-1}\Big(\int \mathsf{p}_m(s_m \mid \mathrm{pa}(s_m))\,\mathrm{d}s_m\Big) \\
&= \prod_{1\le v\le j} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v)) \int \prod_{j+1\le v\le m-1} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v))\,\mathrm{d}s_{j+1}\cdots\mathrm{d}s_{m-1} \\
&= \cdots = \prod_{1\le v\le j} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v)).
\end{aligned}$$

This also verifies that $\mathsf{p}$, as defined in (18.2), is indeed a density. Moreover, for all $j$,

$$\mathsf{p}(s_j \mid s_1,\dots,s_{j-1}) := \frac{\mathsf{p}(s_1,\dots,s_j)}{\mathsf{p}(s_1,\dots,s_{j-1})} = \mathsf{p}_j(s_j \mid \mathrm{pa}(s_j)).$$

Therefore,

$$\begin{aligned}
\mathsf{p}(s_j,\mathrm{pa}(s_j)) &= \int \mathsf{p}(s_1,\dots,s_j) \prod_{v\in\{1,\dots,j-1\}\setminus\mathrm{pa}(s_j)} \mathrm{d}s_v \\
&= \mathsf{p}_j(s_j \mid \mathrm{pa}(s_j)) \int \mathsf{p}(s_1,\dots,s_{j-1}) \prod_{v\in\{1,\dots,j-1\}\setminus\mathrm{pa}(s_j)} \mathrm{d}s_v \\
&= \mathsf{p}_j(s_j \mid \mathrm{pa}(s_j)) \cdot \mathsf{p}(\mathrm{pa}(s_j)),
\end{aligned}$$

implying the coincidence of the conditionals

$$\mathsf{p}(s_j \mid s_1,\dots,s_{j-1}) = \mathsf{p}_j(s_j \mid \mathrm{pa}(s_j)) = \mathsf{p}(s_j \mid \mathrm{pa}(s_j)),$$

hence completing our proof. □

The main significance of this theorem is that by specifying local conditional probability densities $\mathsf{p}_v(s_v \mid \mathrm{pa}(s_v))$ for each node $v$, we automatically obtain a *bona fide* joint probability density $\mathsf{p}$ over the DAG.

**Example 18.6: Necessity of the DAG condition**

Theorem 18.5 can fail if the underlying graph is not a DAG. Consider the simple graph with two nodes $X_1, X_2$ taking values in $\{0,1\}$. Define the conditionals as follows:

$$\mathsf{p}(X_1=1 \mid X_2=1)=0.5, \mathsf{p}(X_1=1 \mid X_2=0)=1, \mathsf{p}(X_2=1 \mid X_1=1)=1, \mathsf{p}(X_2=1 \mid X_1=0)=0.5.$$

Then if the graph is cyclic and the factorization (**??**) holds, then we have

$$\begin{aligned}
\mathsf{p}(X_1=1,X_2=1) &= \mathsf{p}(X_1=1 \mid X_2=1)\cdot\mathsf{p}(X_2=1 \mid X_1=1) = 0.5 \\
\mathsf{p}(X_1=1,X_2=0) &= \mathsf{p}(X_1=1 \mid X_2=0)\cdot\mathsf{p}(X_2=0 \mid X_1=1) = 0 \\
\mathsf{p}(X_1=0,X_2=1) &= \mathsf{p}(X_1=0 \mid X_2=1)\cdot\mathsf{p}(X_2=1 \mid X_1=0) = 0.25 \\
\mathsf{p}(X_1=0,X_2=0) &= \mathsf{p}(X_1=0 \mid X_2=0)\cdot\mathsf{p}(X_2=0 \mid X_1=0) = 0,
\end{aligned}$$

not a joint distribution. Note that even if we re-normalize the joint distribution, we won't be able to recover the conditionals: $\mathsf{p}(X_1=1 \mid X_2=1) = \frac{0.5}{0.75} \neq 0.5$.

**Remark 18.7: Economic parameterization**

For a binary valued random vector $\mathbf{X} \in \{\pm 1\}^d$, in general we need $2^d - 1$ positive numbers to specify its joint density. However, if $\mathbf{X}$ is a BN over a DAG whose maximum in-degree is $k$, then we need only (at most) $d(2^{k+1} - 1)$ positive numbers to specify the joint density (by specifying each conditional in (18.1)).

For $k = 0$ we obtain the independent setting while for $k = 1$ we include the so-called naive Bayes model.

**Definition 18.8: Sigmoid belief network (SBN) (Neal 1992)**

Instead of parameterizing each conditional densities $\mathsf{p}(s_v|\mathrm{pa}(s_v))$ directly, we now consider more efficient ways, which is also necessary if $S_v$ is continuous valued.

Following Neal (1992), we restrict our discussion to binary $\mathbf{S} \in \{\pm 1\}^m$ and define

$$\forall j = 1, \dots, m, \quad \mathsf{p}_W(S_j = s_j | \mathbf{S}_{<j} = \mathbf{s}_{<j}) = \mathtt{sgm}\left( s_j (W_{j,m+1} + \sum_{k=1}^{j-1} s_k W_{jk}) \right) = \mathtt{sgm}(s_j W_{j:} \mathbf{s}),$$

where $W \in \mathbb{R}^{m \times (m+1)}$ is strictly lower-triangular (i.e. $W_{jk} = 0$ if $m \geq k \geq j$). Note that we have added a bias term in the last column of $W$ and appended the constant 1 to get $\mathbf{s} = (\mathbf{s}; 1)$. Note the similarity with Boltzmann machines (17.5).

The ordering of the nodes $S_j$ here are chosen rather arbitrarily. A different ordering may lead to consequences in later analysis, although we shall assume in the following a fixed ordering is given without much further discussion.

Applying Theorem 18.5 we obtain a joint density $\mathsf{p}_W(\mathbf{s}) := \mathsf{p}_W(\mathbf{x}, \mathbf{z})$, parameterized by the (strictly lower-triangular) weight matrix $W$. We will see how to learn $W$ based on a sample $\mathbf{X}_1, \dots, \mathbf{X}_n \sim \mathsf{p}_W(\mathbf{x})$.

Neal, Radford M. (1992). "Connectionist learning of belief networks". *Artificial Intelligence*, vol. 56, no. 1, pp. 71–113.

**Example 18.9: SBN with $m = 1$ and $m = 2$**

Let $m = 1$ in SBN. We have

$$\mathsf{p}_b(S = s) = \mathtt{sgm}(sb),$$

which is exactly the Bernoulli distribution whose mean parameter $p$ is parameterized as a sigmoid transformation of $b$. Compare with Example 17.9.

For $m = 2$, we have instead

$$\mathsf{p}_{w,b,c}(S_1 = s_1, S_2 = s_2) = \mathtt{sgm}(s_1 c) \cdot \mathtt{sgm}(s_2(ws_1 + b)).$$

These examples confirm that without introducing latent variables, SBNs may not approximate all distributions over the cube well.

**Definition 18.10: Noisy-OR belief network (Neal 1992)**

The sigmoid function is chosen in Definition 18.8 mostly to mimic Boltzmann machines, but it is clear that any univariate CDF works equally well. Indeed, we can even define:

$$\mathsf{p}(S_j = 1 | \mathbf{S}_{<j} = \mathbf{s}_{<j}) = 1 - \exp\left(-W \tfrac{1+\mathbf{s}}{2}\right),$$

where as before $W \in \mathbb{R}^{m \times (m+1)}$ is strictly lower-triangular. The awkward term $\frac{\mathbf{s}+1}{2}$ is to reduce to the $\{0, 1\}$-valued case, the setting where noisy-or is traditionally defined in. Note that we need the constraint $W(1 + \mathbf{s}) \geq 0$ (for instance $W \geq \mathbf{0}$ suffices) to ensure the resulting density is nonnegative.

See (Arora et al. 2017) for some recent results on learning noisy-or networks.

Arora, Sanjeev, Rong Ge, Tengyu Ma, and Andrej Risteski (2017). "Provable Learning of Noisy-OR Networks". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1057–1066.
Neal, Radford M. (1992). "Connectionist learning of belief networks". *Artificial Intelligence*, vol. 56, no. 1, pp. 71–113.

**Remark 18.11: Universality**

As shown in (Neal 1992), Boltzmann distributions, SBNs and Noisy-OR BNs in general represent *different* strict subsets of all discrete distributions over the cube $\{\pm 1\}^m$. If we introduce (a large number of) $t$ latent variables, then the marginal distribution on $\mathbf{X}$ can approximate any discrete distribution arbitrarily well, for all 3 networks. More refined universality results can be found in (Sutskever and Hinton 2008).

Neal, Radford M. (1992). "Connectionist learning of belief networks". *Artificial Intelligence*, vol. 56, no. 1, pp. 71–113.
Sutskever, Ilya and Geoffrey E. Hinton (2008). "Deep, Narrow Sigmoid Belief Networks Are Universal Approximators". *Neural Computation*, vol. 20, no. 11, pp. 2629–2636.

**Example 18.12: SBN with $d = 1$ and $t = 1$**

Let $m = 2$ in SBN. We partition $\mathbf{S} = (X, Z)$, hence

$$\mathsf{p}_{w,b,c}(X = x, Z = z) = \mathtt{sgm}(xc) \cdot \mathtt{sgm}(z(wx + b)).$$

Marginalizing over $z = \{\pm 1\}$:

$$\mathsf{p}_{w,b,c}(X = x) = \mathtt{sgm}(xc)[\mathtt{sgm}(wx + b) + \mathtt{sgm}(-wx - b)] = \mathtt{sgm}(xc).$$

On the other hand, if we swap the position of $Z$ and $X$:

$$\mathsf{p}_{w,b,c}(Z = z, X = x) = \mathtt{sgm}(zc) \cdot \mathtt{sgm}(x(wz + b)).$$

Marginalizing over $z = \{\pm 1\}$:

$$\mathsf{p}_{w,b,c}(X = x) = [\mathtt{sgm}(c)\mathtt{sgm}(x(w + b)) + \mathtt{sgm}(-c)\mathtt{sgm}(x(-w + b))],$$

which is a mixture.

**Algorithm 18.13: SBN–Maximum Likelihood**

Given a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n \in \{\pm 1\}^d$, we apply ML to estimate $W$:

$$\min_{W \in \mathbb{R}^{m \times (m+1)}} \mathsf{KL}(\hat{\chi}(\mathbf{x}) \| \mathsf{p}_W(\mathbf{x})),$$

where the SBN $\mathsf{p}_W$ is defined in Definition 18.8 and we remind again that $W$ is always constrained to be strictly lower-triangular (modulus the last bias column) in this section (although we shall not signal this anymore).

To apply (stochastic) gradient descent, we compute the gradient:

$$\begin{aligned}\frac{\partial}{\partial W_{j:}} &= -\mathsf{E}_{\hat{\mathsf{p}}_W} \frac{\partial \log \mathsf{p}_W(\mathbf{s})}{\partial W_{j:}}, \quad \text{where} \quad \hat{\mathsf{p}}_W(\mathbf{s}) := \mathsf{p}_W(\mathbf{z}, \mathbf{x}) := \hat{\chi}(\mathrm{d}\mathbf{x}) \cdot \mathsf{p}_W(\mathbf{z}|\mathbf{x}) \\ &= -\mathsf{E}_{\hat{\mathsf{p}}_W} \frac{\partial \log \prod_{j=1}^m \mathsf{p}_W(s_j|\mathbf{s}_{<j})}{\partial W_{j:}} = -\mathsf{E}_{\hat{\mathsf{p}}_W} \frac{\partial \log \mathsf{p}_W(s_j|\mathbf{s}_{<j})}{\partial W_{j:}} = -\mathsf{E}_{\hat{\mathsf{p}}_W} \frac{\mathtt{sgm}'(s_j W_{j:}\mathbf{s})}{\mathtt{sgm}(s_j W_{j:}\mathbf{s})} s_j \tilde{\mathbf{s}}_{<j},\end{aligned}$$

where we use the tilde notation $\tilde{\mathbf{s}}_{<j}$ to denote the full size vector where coordinates $k \in [j, m]$ are zeroed out (since $W$ is strictly lower-triangular). Using the fact that $\mathtt{sgm}'(t) = \mathtt{sgm}(t)\mathtt{sgm}(-t)$ we obtain:

$$\forall j = 1, \ldots, d, \ \forall k \notin [j, m], \quad \frac{\partial}{\partial W_{jk}} = -\mathsf{E}_{\hat{\mathsf{p}}_W} \mathtt{sgm}(-s_j W_{j:}\mathbf{s}) s_j s_k.$$

Note that the gradient only involves 1 expectation, while there were 2 in BMs (cf. Remark 17.14). This is

perhaps expected, since in BM we have that intractable log-partition function to normalize the density while in BN the joint density is automatically normalized since each conditional is so.

**Algorithm 18.14: Conditional Gibbs sampling for SBN**

We now give the conditional Gibbs sampling algorithm for $\hat{p}_W(\mathbf{x},\mathbf{z}) = \hat{\chi}(\mathrm{d}\mathbf{x}) \cdot \mathsf{p}_W(\mathbf{z}|\mathbf{x})$. For that we derive the conditional density (Pearl 1987):

$$\begin{aligned}\mathsf{p}(S_j = t|\mathbf{S}_{\setminus j} = \mathbf{s}_{\setminus j}) &\propto \mathsf{p}(\mathbf{S}_{\setminus j} = \mathbf{s}_{\setminus j}, S_j = t)\\ &\propto \mathsf{p}(S_j = t|\mathbf{S}_{<j} = \mathbf{s}_{<j}) \cdot \prod_{k>j} \mathsf{p}(S_k = s_k|\mathbf{S}_{<k,\setminus j} = \mathbf{s}_{<k,\setminus j}, S_j = t)\\ &= \mathtt{sgm}(tW_{j:}\mathbf{s}) \prod_{k>j} \mathtt{sgm}[s_k(W_{k:}\mathbf{s} + W_{kj}(t - s_j))]\end{aligned}$$

We omit the pseudo-code but point out that the matrix vector product $W\mathbf{s}$ should be dynamically updated to save computation.

Pearl, Judea (1987). "Evidential reasoning using stochastic simulation of causal models". *Artificial Intelligence*, vol. 32, no. 2, pp. 245–257.

**Alert 18.15: Importance of ordering: training vs. testing**

If we choose to put the observed variables $\mathbf{X}$ before the latent variables $\mathbf{Z}$ so that $\mathbf{S} = (\mathbf{X}, \mathbf{Z})$. Then, drawing a sample $\mathbf{Z}$ necessitates the above (iterative) Gibbs sampling procedure. Thus, training is expensive. However, after training drawing a new (exact) sample for $\mathbf{X}$ only takes one-shot. In contrast, if we arrange $\mathbf{S} = (\mathbf{Z}, \mathbf{X})$. Then, drawing a sample $\mathbf{Z}$ during training no longer requires the Gibbs sampling algorithm hence training is fast. However, the price to pay is that at test time when we want to draw a new sample $\mathbf{X}$, we would have to run the iterative Gibbs sampling algorithm.

**Exercise 18.16: Failure of EM**

I know you must wondering if we can apply EM to SBN. The answer is no and the details are left as exercise.

**Definition 18.17: Deep belief network (DBN) (Bengio and Bengio 1999)**

We now extend SBN to handle any type of data and go deep. We achieve this goal through 3 key steps:

- First, we realize that SBN amounts to specifying $d$ univariate, parameterized densities. Indeed, let

  $$\mathsf{p}(s_j|s_{<j}) = \mathsf{p}_j(s_j; \theta_j(\mathbf{s}_{<j})),$$

  where $\mathsf{p}_j$ is a prescribed univariate density on $s_j$ with parameter $\theta_j$. For instance, $\mathsf{p}_j$ may be the exponential family with natural parameter $\theta_j$ (e.g. Bernoulli where $\theta_j = p_j$ or Gaussian where $\theta_j = (\mu_j, \sigma_j^2)$).

- Second, the parameter $\theta_j$ can be computed as a function from the previous observations $\mathbf{s}_{<j}$. For instance, SBN specifies $\mathsf{p}_j$ to be Bernoulli whose mean parameter $p_j$ is computed as the output of a two-layer neural network with sigmoid activation function:

  $$\begin{aligned}\mathbf{h} &= W\mathbf{s}\\ \mathbf{p} &= \mathtt{sgm}(\mathbf{h}).\end{aligned}$$

- It is then clear that we could use a deep neural net to compute the parameter $\theta_j$:

  $$\mathbf{h}_0 = \mathbf{s} \tag{18.3}$$

$$\begin{aligned} \forall \ell = 1, \ldots, L-1, \quad \mathbf{h}_\ell &= \sigma(W_\ell \mathbf{h}_{\ell-1}) \\ \boldsymbol{\theta} &= \sigma(W_L \mathbf{h}_{L-1}). \end{aligned}$$

Note that we need to make sure $\theta_j$ depends only on the first $j-1$ inputs $\mathbf{s}_{<j}$, which can be achieved by wiring the network appropriately. For instance, if $W_1$ is strictly lower-triangular while $W_{\geq 2}$ is lower-triangular (Bengio and Bengio 1999), then we obviously satisfy this constraint.

Bengio, Yoshua and Samy Bengio (1999). "Modeling High-Dimensional Discrete Data with Multi-Layer Neural Networks". In: *NeurIPS*.

**Alert 18.18: Weight sharing**

We compare the network structure (18.3) with the following straightforward alternative:

$$\begin{aligned} \mathbf{h}_0^{(j)} &= \mathbf{s}_{<j} \\ \forall \ell = 1, \ldots, L-1, \quad \mathbf{h}_\ell^{(j)} &= \sigma(W_\ell^{(j)} \mathbf{h}_{\ell-1}^{(j)}) \\ \theta_j &= \sigma(W_L^{(j)} \mathbf{h}_{L-1}^{(j)}), \end{aligned} \tag{18.4}$$

where we use separate networks for each parameter $\theta_j$. The weights $W_\ell^{(j)}$ above can be arbitrary. We observe that the parameterization in (18.3) is much more efficient, since the weights used to compute $\theta_j$ are shared with all subsequent computations for $\theta_{\geq j}$. Needless to say, the parameterization in (18.4) are more flexible.