Resticted Boltzmann Machines
A Short Tutorial

These notes are meant as a practical tutorial providing the minimum sufficient knowledge required to build a Restricted Boltzmann Machine (RBM).

RBMs are among the top methods in unsupervised machine learning, where the training data are inputs without labels and the task is, broadly speaking, to extract some meaningful information from this data.

An RBM is a parametrized generative model representing a probability distribution. The training data is assumed to be a sample drawn independently from an unknown **target distribution** q. The goal of training is to fit the parameters λ of the RBM's distribution p_λ such that it resembles the target distribution q as accurately as possible.

Formally, RBMs belong to the class of *undirected graphical models*, also known as *Markov Random Fields* (MRFs). Probabilistic graphical models describe probability distributions by mapping conditional (in)dependence properties between random variables on a graph structure. Visualization by graphs is useful for understanding and motivating probabilistic models. Moreover, it can be helpful for deriving complex computations by using algorithms that exploit the graph structure. Much can be said about the theoretical properties of graphical models and the implications for RBMs. However, with the goal of keeping this tutorial short and hands-on, we omit this discussion here and refer to the references provided at the end.

Practically, an RBM is a two-layer network with bidirectionally connected stochastic processing units, as shown in figure 1. The V units in the first layer, denoted by the vector $v = (v_1, ..., v_V)$, correspond to the components of an observation and are therefore called *"visible"*, while the H units in the second layer $h = (h_1, ..., h_H)$ represent latent variables, and are referred to as *"hidden"*. Hidden units model dependencies between the observation components and can be viewed as feature detectors. The term *"restricted"* refers to the connections between the units: Each visible unit is connected with each hidden unit, but there are no connections between units of the same kind. In the simplest case (which is the case considered in this tutorial), all units are binary, such that $v \in \mathcal{V} = \{0, 1\}^V$ and $h \in \mathcal{H} = \{0, 1\}^H$, where curved letters $\mathcal{V}$ and $\mathcal{H}$ are used to denote the space of the visible and hidden vectors, respectively. Further, we use $\mathcal{D}$ to denote the training set containing $|\mathcal{D}|$ instances of the visible vector (i.e., observations).

In analogy to spin models in statistical physics, the joint configuration (v, h) of

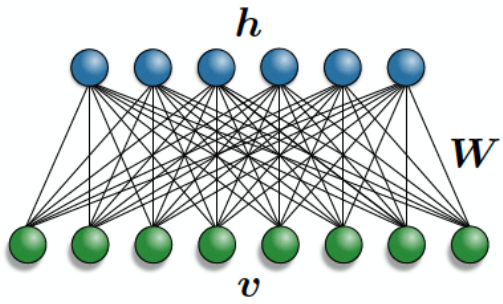

Figure 1: RBM as a bipartite graph. (Figure from Giac's Thesis)

visible and hidden units is characterized by an **energy**

$$\begin{aligned} E_\lambda(v,h) &= -b^T v - c^T h - h^T W v \\ &= -\sum_{i=1}^{V} b_i v_i - \sum_{j=1}^{H} c_j h_j - \sum_{ij} v_i W_{ij} h_j, \end{aligned} \tag{1}$$

where v_i, h_j are binary states of visible unit i and hidden unit j, and b_i, c_j are their respective **biases**. W_{ij} is the symmetric connection **weight** between the units. The complete **set of parameters** is denoted by $\lambda = \{b, c, W\}$. All parameter values are real numbers.

Based on this energy function, the RBM assigns a probability to each joint configuration (v, h), which by convention is high when the energy of the configuration is low:

$$p_\lambda(v,h) = \frac{1}{Z} e^{-E_\lambda(v,h)}, \tag{2}$$

known as the *Gibbs distribution*. The **partition function** Z is given by summing over all possible pairs of visible and hidden vectors:

$$Z = \sum_{v \in \mathcal{V}} \sum_{h \in \mathcal{H}} e^{-E_\lambda(v,h)}. \tag{3}$$

What we actually want to model is the **probability distribution of an input vector** v, $p_\lambda(v)$. It is obtained by marginalizing the joint probability distribution $p_\lambda(v,h)$ over all possible hidden vectors h:

$$p_\lambda(v) = \frac{1}{Z} \sum_{h \in \mathcal{H}} e^{-E_\lambda(v,h)}. \tag{4}$$

By carrying out the sum over h, one obtains a related expression

$$p_\lambda(v) = \frac{1}{Z} e^{-\mathcal{E}_\lambda(v)}, \tag{5}$$

where $\mathcal{E}_\lambda(v)$ is the **effective energy**, defined as

$$\begin{aligned}\mathcal{E}_\lambda(v) &= -b^T v - \sum_{j=1}^{H} \ln\left[1 + \exp\left(c_j + \sum_i W_{ij} v_i\right)\right] \\ &= -b^T v - \sum_{j=1}^{H} \text{softplus}\left(c_j + \sum_i W_{ij} v_i\right).\end{aligned}$$

Note that in some ML references this quantity is called the "free energy", but it is not the same as the free energy in physics, which is defined as $F = -\ln Z_\lambda$.

The joint probability distribution is related to the conditional distributions via the chain rule as $p_\lambda(v,h) = p_\lambda(v|h)p_\lambda(h) = p_\lambda(h|v)p_\lambda(v)$. Because there are no direct connections between units of the same layer in an RBM, the **conditional distributions** $p_\lambda(h|v)$ and $p_\lambda(v|h)$ factorize over each unit and are easy to compute. With some straightforward algebra, one can show that

$$p_\lambda(h|v) = \prod_{j=1}^{H} p_\lambda(h_j|v), \tag{6}$$

$$p_\lambda(v|h) = \prod_{i=1}^{V} p_\lambda(v_i|h), \tag{7}$$

and

$$p_\lambda(h_j = 1|v) = \mathcal{S}\left(c_j + \sum_i v_i W_{ij}\right), \tag{8}$$

$$p_\lambda(v_i = 1|h) = \mathcal{S}\left(b_i + \sum_j h_j W_{ij}\right), \tag{9}$$

with $\mathcal{S}$ denoting the standard logistic sigmoid function:

$$\mathcal{S}(x) = \frac{1}{1 + e^{-x}}. \tag{10}$$

Training the RBM means adjusting parameters λ based on the given data set $\mathcal{D}$, such that $p_\lambda(v)$ is a good approximation of the true distribution $q(v)$. In the supervised setting, we would define a cost function that measures the discrepancy between the network prediction and the desired output (which is part of the training set), and perform a minimization of the cost function. Similarly, in the unsupervised setting we introduce a **cost function** C_λ that measures how the probability distribution p_λ is different from q, known as the *Kullback-Leibler (KL) divergence* or the *relative entropy*:

$$C_\lambda = D_{KL}(q\|p_\lambda) \equiv \sum_v q(v) \ln \frac{q(v)}{p_\lambda(v)} = -H(q) - \langle \ln p_\lambda(v) \rangle_q. \tag{11}$$

The first term on the right-hand side is simply the *Shannon entropy* of q,

$$H(q) = -\sum_v q(v) \ln q(v), \tag{12}$$

and the second term is an expectation value of the quantity $\ln p_\lambda(v)$ called the *log-likelihood*:

$$\langle \ln p_\lambda(v) \rangle_q = \sum_v q(v) \ln p_\lambda(v). \tag{13}$$

The goal of the training procedure is then to find a set of parameters λ that minimizes the cost function C_λ. The minimization is performed by *gradient descent*, and the **update rule** for the parameters has the familiar form

$$\lambda \leftarrow \lambda - \eta \nabla_\lambda C_\lambda, \tag{14}$$

where η is the learning rate.

Note that $H(q)$ does not depend on λ, such that only the log-likelihood term is relevant for the optimization:

$$\nabla_\lambda C_\lambda = -\nabla_\lambda \langle \ln p_\lambda(v) \rangle_q. \tag{15}$$

However, the log-likelihood term involves the unknown target distribution q. In order to proceed with the minimization, we approximate q by the **empirical distribution** of the training data $q_\mathcal{D}$:

$$q(v) \simeq q_\mathcal{D}(v) = \frac{1}{|\mathcal{D}|} \sum_{\tilde{v} \in \mathcal{D}} \delta(v - \tilde{v}). \tag{16}$$

The expectation value in eq. (15) can then be evaluated as follows:

$$\begin{aligned} \langle \ln p_\lambda(v) \rangle_q \simeq \langle \ln p_\lambda(v) \rangle_\mathcal{D} &= \frac{1}{|\mathcal{D}|} \sum_v \sum_{\tilde{v} \in \mathcal{D}} \delta(v - \tilde{v}) \ln p_\lambda(v) \\ &= \frac{1}{|\mathcal{D}|} \sum_{v \in \mathcal{D}} \ln p_\lambda(v) \\ &= -\frac{1}{|\mathcal{D}|} \sum_{v \in \mathcal{D}} [\mathcal{E}_\lambda(v) + \ln Z_\lambda] \\ &= -\frac{1}{|\mathcal{D}|} \sum_{v \in \mathcal{D}} \mathcal{E}_\lambda(v) - \ln Z_\lambda. \end{aligned} \tag{17}$$

Thus, the gradient of the cost function is

$$\begin{aligned} \nabla_\lambda C_\lambda \simeq -\nabla_\lambda \langle \ln p_\lambda(v) \rangle_\mathcal{D} &= \frac{1}{|\mathcal{D}|} \sum_{v \in \mathcal{D}} \nabla_\lambda \mathcal{E}_\lambda(v) + \nabla_\lambda \ln Z_\lambda \\ &= \sum_v q_\mathcal{D}(v) \nabla_\lambda \mathcal{E}_\lambda(v) - \sum_v p_\lambda(v) \nabla_\lambda \mathcal{E}_\lambda(v) \\ &= \langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_\mathcal{D} - \langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{p_\lambda}. \end{aligned} \tag{18}$$

We evaluate $\nabla_\lambda \mathcal{E}_\lambda$ for each parameter in λ by computing the gradient element-wise:

$$\frac{\partial \mathcal{E}_\lambda(v)}{\partial W_{ij}} = -v_i \, \mathcal{S}\left(c_j + \sum_k v_k W_{kj}\right) = -v_i \, p_\lambda(h_j = 1|v) \tag{19}$$

$$\frac{\partial \mathcal{E}_\lambda(v)}{\partial c_j} = -\mathcal{S}\left(c_j + \sum_i v_i W_{ij}\right) = -p_\lambda(h_j = 1|v) \tag{20}$$

$$\frac{\partial \mathcal{E}_\lambda(v)}{\partial b_i} = -v_i \tag{21}$$

The expectation value over the empirical probability distribution, $\langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{\mathcal{D}}$, can easily be computed based on the training set. The term $\langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{p_\lambda}$, however, poses some serious problems. This expectation value is over the marginalized probability distribution $p_\lambda(v) = e^{-\mathcal{E}_\lambda(v)}/Z$ and involves the partition function Z which requires evaluation of the sum over all $h \in \mathcal{H}$ **and** $v \in \mathcal{V}$, and is thus in general intractable. There are several possible approximate methods to handle this term, but not all of them are practicable. The straighforward approach is to approximate the expectation $\langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{p_\lambda}$ by an *estimator* sampled from the model distribution p_λ using *Gibbs sampling*. In Gibbs sampling, each variable is sampled from its conditional distribution given the current states of the other variables. Starting from a randomly initialized visible state $v = v^{(0)}$, we alternate between updating h and v according to $p_\lambda(h|v)$ and $p_\lambda(v|h)$. In RBMs, this procedure is particularly efficient because the visible units are conditionally independent given the hidden units (and vice versa), such that the calculation can be carried out for all units in parallel, as illustrated in figure 2. This is referred to as *Block Gibbs Sampling*. We repeat this procedure on M different initial states and use the average to approximate the expectation value:

$$\langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{p_\lambda} = \sum_v p_\lambda(v) \nabla_\lambda \mathcal{E}_\lambda(v) \simeq \frac{1}{M} \sum_{v^{(t)} \in \mathcal{M}} \nabla_\lambda \mathcal{E}_\lambda(v^{(t)}), \tag{22}$$

where the index t refers to the number of Gibbs steps performed and $\mathcal{M}$ denotes the set of M visible state vectors $v^{(t)}$.

However, to ensure that the Markov chain converges to stationarity, the sampling process has to be run for a long time, i.e., the number of Gibbs steps t has to be large. Since this process has to be repeated for each parameter update in the learning procedure, this technique is computationally not feasible. Therefore, all methods that are employed in practice introduce additional approximations.

The standard **learning algorithm** employed for RBM training is called *Contrastive Divergence* (CD-k) [**?**]. Essentially, the idea of this algorithm is to perform only k steps of Gibbs sampling starting from a current training vector $v = v^{(0)}$ (note the difference: in standard Gibbs sampling you start from a randomly initialized vector!). Even though it is only crudely approximating the true expectation, the

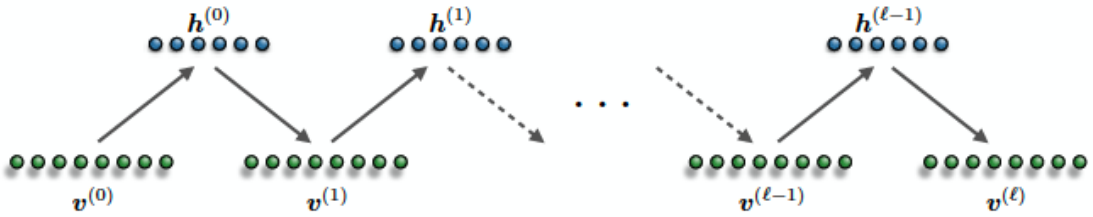

Figure 2: In RBMs, the visible and hidden units are conditionally independent of each other, such that Gibbs sampling can alternate between parallel updates of the hidden and visible units. At $t \to \infty$, the samples $(v^{(t)}, h^{(t)})$ are guaranteed to be accurate samples of the model distribution $p_\lambda(v, h)$ (Figure from Giac's Thesis).

learning works surprisingly well. In general, larger k yields a less biased estimate; however, $k = 1$ is often sufficient to extract meaningful features in practice.

To sum up, the main steps of the CD-k learning procedure are:

0. Select M vectors from the training data (referred to as *mini batch*) and perform the next two steps for each of the M vectors in parallel.
1. Initialize by setting the visibles to the training vector v.
2. Perform the following Gibbs sampling procedure k times:
   - Compute the states of the hiddens (in parallel) by setting each unit to 1 with a probability given by eq. (8).
   - Produce a *"reconstruction"* of v (denoted as $v^{(i)}$, where i is the iteration step number), by setting each visible to 1 with a probability given by eq. (7).
3. Use the M reconstructions $v^{(k)}$ to compute the estimator for $\langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{p_\lambda}$ and update the model parameters λ.
4. Go back to step 0 and repeat the procedure until stopping criteria fulfilled.

A remark on computing the state of a binary unit

Given a visible vector v and the set of parameters λ, how do we compute the state of a hidden unit h_j? Since the unit is binary, its state is sampled according to the *Bernoulli distribution* with the probability for h_j having the value 1 given by eq. (8). In practice, we sample a number u from a *uniform distribution* over the interval $[0, 1]$, $u \sim U[0, 1]$. If $u < p_\lambda(h_j = 1|v)$, set $h_j = 1$, otherwise $h_j = 0$.

A slight modification of the CD algorithm leads to the *Persistent Contrastive Divergence* (PCD) algorithm [**?**]. In PCD, instead of initializing the chain to $v^{(0)} = v \in \mathcal{D}$ each time, we use the negative sample from the previous iteration. That is,

PCD just keeps the Markov chain evolving, with parameter updates done after each k steps. The number of persistent chains used for sampling is a hyperparameter. In the standard case, there is one Markov chain per training example in a batch.

Suggested Reading and References

For composing this tutorial, we consulted a number of excellent references that elucidate the topic of RBMs from different perspectives:

- Hugo Larochelle's course on Neural Networks [**?**] – including lecture slides, youtube videos and very useful references; part 5 is about RBMs. Very beginner-friendly and intuitive, derivations of several central mathematical expressions are done explicitly.
- "An Introduction to RBMs" [**?**] by Fischer and Igel, with an emphasis on the mathematical background, in particular graphical models. Explains Markov Chains and Markov Chain Monte Carlo, Markov Random Fields, RBMs, Contrastive Divergence and related algorithms, and provides an extensive list of references.
- "A Practical guide to Training RBMs" [**?**] by Hinton & Co. – as the title promises, a practice-oriented text, with a very short intro to RBMs and a detailed guide on the actual training process; less theory, more empirically approved "recipies".
- "Training Products of Experts by Minimizing Contrastive Divergence" [**?**] – the original article by Hinton introducing the CD algorithm, presented in a clear and easily accessible form.
- Finally, "A high-bias, low-variance introduction to Machine Learning for physicists" [**?**] by Mehta et al. – a recently written review of ML methods from a physicist's perspective; covers many topics in ML, includes code examples. Most importantly, it points out some insightful connections to methods and models known to physicists from the fields of statistical and many-body physics.