Boltzmann Machines

Yelysei Bondarenko

Contents

Boltzmann Machines and RBMs

Before BMs[19, 17, 12]

General RNNs

In early 1980-x general **recurrent neural networks** (RNNs), described by (general) *asyclic* graph began to arise.

- mostly binary units, $\{0, 1\}$ (or $\{-1, 1\}$);
- ✓ signal feedback $\Rightarrow$ **memory** compared to FF NNs (more specifically, *content-addressable memory* (CAM) $\Leftrightarrow$ weights themselves are used to store patterns);
- ✕ in general if activations are non-linear function they are hard to train (oscillations and chaotic behavior).

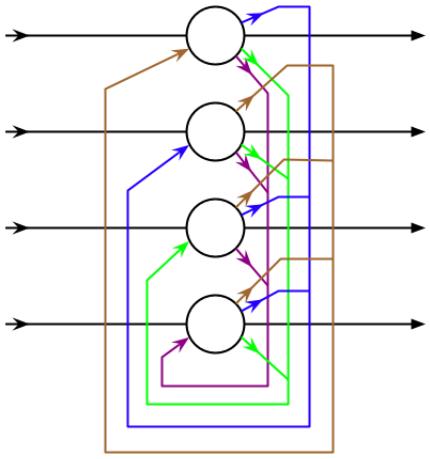


Figure 1: Example architecture of RNN

Hopfield Nets

Later in ≈1982 John Hopfield has shown that if RNN has symmetric weights ($W_{ij} = W_{ji}$), and has no self-loops ($W_{ii} = 0$), network's dynamics **is guaranteed** to converge to some stationary state. Fully-connected variant (complete graph) of such a network is called a *Hopfield Network*.

Moreover, he also shown that each state of the Hopfield Net is associated with a scalar value referred to as the *energy* of the network ($s_i \in \{-1, 1\}$ – state of i-th neuron, b_i – bias of i-th neuron ($-b_i$ is activation threshold for the unit)):

$$E = -\frac{1}{2}\sum_{i,j} W_{ij}s_i s_j - \sum_i b_i s_i = \left[\begin{array}{c} W_{ij} = W_{ji}, \\ W_{ii} = 0 \end{array}\right] = -\sum_{i<j} W_{ij}s_i s_j - \sum_i b_i s_i \tag{1}$$

and local minima of $E(\mathbf{s}; \mathbf{W}, \mathbf{b}) \Leftrightarrow$ stable configurations of network. It is the first example of so-called **energy-based model**. Learning algorithm for such a model alter its (global) energy function to achieve desired properties. For instance, if a Hopfield Net is trained as autoassociator, the goal of learning is to shape energy function in such a way, that its local minima correspond to training examples (= patterns to "remember").

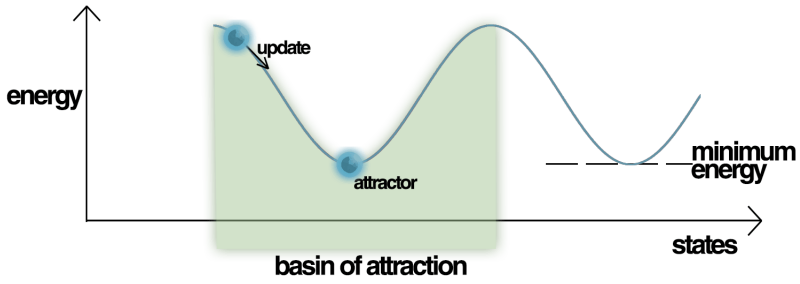


Figure 2: Energy Landscape of a Hopfield Network, highlighting the current state of the network (up the hill), an attractor state to which it will eventually converge, a minimum energy level and a basin of attraction shaded in green. Note how the update of the Hopfield Network is always going down in Energy.

Learning algorithms for Hopfield Nets

There are exist couple of learning algorithms for Hopfield Nets. First, thanks to simple quadratic shape of E, it is straightforward to calculate how i-th neuron influences the global energy:

$$\Delta E_i = E(\mathbf{s}|s_i = \text{off}) - E(\mathbf{s}|s_i = \text{on}) = E(\mathbf{s}|s_i = 0) - E(\mathbf{s}|s_i = 1) =$$

$$= -\sum_{k<j} W_{kj} s_k s_j \Big|_{s_i=0} - \sum_j b_j s_j \Big|_{s_i=0} + \sum_{k<j} W_{kj} s_k s_j \Big|_{s_i=1} + \sum_j b_j s_j \Big|_{s_i=1} =$$

$$= \underbrace{-\left(\sum_{k<j,k\neq i,j\neq i} W_{kj} s_k s_j + \sum_{j\neq i} b_j s_j\right) + \left(\sum_{k<j,k\neq i,j\neq i} W_{kj} s_k s_j + \sum_{j\neq i} b_j s_j\right)}_{=0} + \underbrace{\sum_{j<i} W_{ji} s_j}_{k=i} + \underbrace{\sum_{i<k} W_{ik} s_k}_{j=i} + b_i =$$

$$= [W_{ij} = W_{ji},\, W_{jj} = 0] = \sum_j W_{ij} s_i s_j + b_i$$

This lead to the learning algorithm called **Binary Threshold Decision Rule** (BTDR):

- Randomly initialize states (or to desired pattern if want CAM);
- While not converged or for fixed number of iterations:

  For each neuron:

Change its state if this will decrease global energy. More specifically:

$$s_i \leftarrow \begin{cases} +1, & \Delta E_i > 0, \\ s_i, & \Delta E_i = 0, \\ -1, & \Delta E_i < 0. \end{cases}$$

This learning algorithm is *local* and *incremental*, thus biologically plausible. Also neurons could have been updated simultaneously, but it is less likely that there is exists "global clock" in biological system, and such kind of updates can also cause oscillation or chaotic behavior.

One important drawback of such an algorithm, is that once we stuck in poor local minimum, we cannot escape it, it is also one of the reasons why Boltzmann Machines was more successful.

Another learning algorithm is based on famous **Hebb rule**: "Cells that fire together, wire together". In the simplest case it simply says:

$$W_{ij} \leftarrow x_i x_j, \tag{2}$$

where $\mathbf{x} = \{x_1 \dots x_N\}$ is binary input pattern. If x_i and x_j are the same, then W_{ij} is positive thus i-th and j-th state tend to become equal. Opposite happens when x_i and x_j are different. Now one can show that network's dynamics will be the same if

$$W_{ij} \leftarrow \frac{1}{N} x_i x_j \tag{3}$$

If we need to remember not 1 but P patterns, sum corresponding update for each pattern (for each pattern pretend there are no others):

$$W_{ij} \leftarrow \frac{1}{N} \sum_{p=1}^{P} x_i^{(p)} x_j^{(p)} \tag{4}$$

This way network will act as CAM, see Fig. 3. The network will converge to a "remembered" state if it is given only part of the state $\Rightarrow$ can be used to recover from a distorted input to the trained state that is most similar to the input.

For instance, train Hopfield net such that $(1, -1, 1, -1, 1)$ is a local minimum of E (phase 1 – memorization). Next, if the network is properly trained, it can recover $(1, -1, 1, -1, 1)$ from $(1, -1, \mathbf{-1}, -1, 1)$ input (phase 2 – recognition). Hopfield nets can be used for denoising of simple fonts.

Many generalizations of this rule exist, but fundamentally Hopfield nets had quite some drawbacks (even for binary data):

✕ patterns are stored in the network itself, thus can be intractable for large N;

✕ capacity, one can reliably store only $P << N$ patterns;

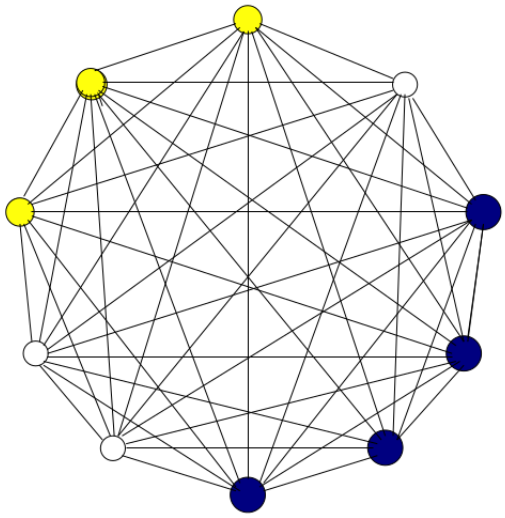


Figure 3: A Hopfield network as an autoassociator. One enters a pattern in blue nodes and let the network evolve. After a while one reads out the yellow nodes. The memory of the Hopfield network associates the yellow node pattern with the blue node pattern.

- ✕ spurious minima, poor local minima;
- ✕ no probabilistic interpretation.

But anyway Hopfield Nets and energy-based models played a big role in development of deep learning and now we proceed to the one of the most famous modification of Hopfield Net – Boltzmann Machine.

Boltzmann Machines[12, 3, 5, 27]

Main ideas

Boltzmann Machine (developed ≈1980-85 by G. Hinton) is a *stochastic*, *generative* counter-part of Hopfield Nets. From PGM point of view, it is an example of MRF (undirected graphical model). It is also an example of Ising model.

Two main ideas:

1. Instead of storing "memories" in stable configurations of Hopfield net, use them for "interpretation" of the input data. Thus, all units are divided into 2 groups: *visible* and *hidden* (see Fig. 4).
2. To escape local minima, assume units are *Gibbs distibuted* with the same energy function (1) and use more advanced learning algorithms (see below).

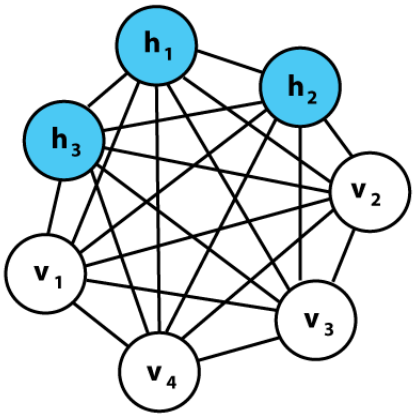


Figure 4: A graphical representation of an example Boltzmann machine. Each undirected edge represents dependency. In this example there are 3 hidden units and 4 visible units. This is not a restricted Boltzmann machine.

Historical development and informal explanation

Inspired from statistical physics, assume that units' states are Gibbs distributed (= Boltzmann distributed) random variables for global energy (1) $\Rightarrow$ energy of the state is proportional to the negative log-probability of that state and also the following identity holds:

$$\Delta E_i = E(\mathbf{s}|s_i = 0) - E(\mathbf{s}|s_i = 1) \ \propto \ -k_B T \log p\{s_i = 0\} + k_B T \log p\{s_i = 1\}, \tag{5}$$

By absorbing Boltzmann's constant k_B into introduced notion of (artificial) temperature T ($T \leftarrow k_B T$) ($\Leftrightarrow$ by considering *dimensionless* units where $k_B = 1$) , we can rearrange equation (5) and solve for $p\{s_i = 1\}$. We will obtain so-called *normal logistic equation*:

$$p\{s_i = 1\} = \frac{1}{1 + \exp\left(-\frac{\Delta E_i}{T}\right)} = \text{sigm}\left(-\frac{\Delta E_i}{T}\right) \tag{6}$$

We obtained a **Modified BTDR**:

- Randomly initialize states;
- In cycle for each unit if its state reversal will yield energy decrease, reverse its state with probability described by (6).

After running for long enough time at certain T, it turns out that probability of a global state of network will *depend only upon global state's energy*, and not on the initial state (in accordance with Boltzmann distribution).
$\Rightarrow$ Distribution of all possible configurations (global states) converges to same *stationary distribution* (this state in physics is called thermal equilibrium). Note that being in thermal equilibrium, system does not necessary is in the state of the lowest energy, which still might be oscillating, **but**

if we start running the network from a high temperature, and gradually decrease it over time until we reach a thermal equilibrium at low T, we may converge to a distribution where the energy level fluctuates around the global minimum (at least this can happen with higher probability than in Hopfield Nets).
This process is called **simulated annealing**. Yes, it seems like development of BMs gave fundament to one of the most famous and successful algorithms of global optimization.

Now if we want to train the network so that the chance it will converge to a global state is according to an external distribution (e.g. data) that we have over these states, we need to *set the weights* so that the global states with the highest probabilities will get the lowest energies. By altering parameters we can shape the distribution produced by BM using (6), it is the main idea of Boltzmann machine.

BM is interpreted in the following way. Visible units provide open interface to the world and represent data. Hidden units represent hidden patterns in the data. Usually it is trained by using maximum likelihood estimation using approximate (why see below) gradient ascent. It is also equivalent [7] to minimizing KL-divergence between external distribution (which is usually empirical data distribution) and model distribution produced by the network.

More formal derivations

Notations:
$\mathbf{v} \in \{0,1\}^D = \{0,1\}^V$ – vector of visible units,
$\mathbf{h} \in \{0,1\}^H$ – vector of hidden units,

Energy of a particular state $(\mathbf{v}, \mathbf{h})$ is:

$$E(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi}) = -\frac{1}{2}\mathbf{v}^T\mathbf{L}\mathbf{v} - \frac{1}{2}\mathbf{h}^T\mathbf{J}\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h} - \mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h}, \tag{7}$$

it is nothing else but (1) with old $\mathbf{W}$ split into $\mathbf{L}$ describing vis-vis weights, $\mathbf{J}$ describing hid-hid weights, and new $\mathbf{W}$ describing vis-hid connections, and the same for biases. The reason is to keep notations consistent with later parts where we will get rid of $\mathbf{L}$ and $\mathbf{J}$. Of course, $\mathbf{L}$ and $\mathbf{J}$ are symmetric with zeros on the main diagonal. $\boldsymbol{\psi} = \{\mathbf{L}, \mathbf{J}, \mathbf{W}, \mathbf{b}, \mathbf{c}\}$ are model parameters.

Probability of the configuration $(\mathbf{v}, \mathbf{h})$ is (according to the Boltzmann distribution):

$$p(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi}) = \frac{e^{-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}}{\sum_{\widehat{\mathbf{v}},\widehat{\mathbf{h}}} e^{-E(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})}} =: \frac{p^*(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi})}{Z(\boldsymbol{\psi})}, \tag{8}$$

where $p^*(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi})$ is unnormalized probability of the configuration and $Z(\boldsymbol{\psi})$ is a normalizer also called *partition function.*
$\Rightarrow$
Probability that model assigns to a visible vector $\mathbf{v}$ is

$$p(\mathbf{v}; \boldsymbol{\psi}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi}) =: \frac{e^{-\mathcal{F}(\mathbf{v};\boldsymbol{\psi})}}{\sum_{\widehat{\mathbf{v}}} e^{-\mathcal{F}(\widehat{\mathbf{v}};\boldsymbol{\psi})}}, \tag{9}$$

where

$$\mathcal{F}(\mathbf{v};\boldsymbol{\psi}) = -\log\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}\right) \tag{10}$$

is called *free energy* (*free* because all the hidden states are marginalized out; the name is inspired again from physics), useful quantity which will be used later.

Now lets calculate $p(v_i = 1|\mathbf{h}, \mathbf{v}_{-i})$: (omit $\boldsymbol{\psi}$ for brevity)

$$p(v_i = 1|\mathbf{h}, \mathbf{v}_{-i}) = \frac{p(v_i = 1, \mathbf{v}_{-i}, \mathbf{h})}{p(\mathbf{v}_{-i}, \mathbf{h})} = \frac{p(v_i = 1, \mathbf{v}_{-i}, \mathbf{h})}{p(v_i = 0, \mathbf{v}_{-i}, \mathbf{h}) + p(v_i = 1, \mathbf{v}_{-i}, \mathbf{h})}$$

$$= \frac{e^{-E(v_i=1,\mathbf{v}_{-i},\mathbf{h})}}{e^{-E(v_i=0,\mathbf{v}_{-i},\mathbf{h})} + e^{-E(v_i=1,\mathbf{v}_{-i},\mathbf{h})}} = \frac{1}{1 + e^{-[E(v_i=0,\ldots)-E(v_i=1,\ldots)]}} = \text{sigm}\left[E(v_i = 0, \ldots) - E(v_i = 1, \ldots)\right] =$$

$$= \left[E(\mathbf{v}, \mathbf{h}) = -\sum_{j<k} L_{jk} v_j v_k - \sum_{l<m} J_{lm} h_l h_m - \sum_{j,l} W_{jl} v_j h_l - \sum_j b_j v_j - \sum_l c_l h_l\right] =$$

$$= [\text{sums w/o } v_i \text{ (2nd, 5th) cancel out}] = \text{sigm}\left[\underbrace{-\ldots + \sum_{j<k, j\neq i, k\neq i} L_{jk} v_j v_k}_{=0} + \underbrace{\sum_{i<k} L_{ik} v_k}_{j=i} + \underbrace{\sum_{j<i} L_{ji} v_j}_{k=i} - \right.$$

$$\left. \underbrace{-\ldots + \sum_{j\neq i,l} W_{jl} v_j h_l}_{=0} + \underbrace{\sum_l W_{il} h_l}_{j=i} + b_i\right] = [L_{ij} = L_{ji}, L_{ii} = 0] = \text{sigm}\left[\sum_l W_{il} h_l + \sum_k L_{ik} v_k + b_i\right]$$

So:

$$\boxed{p(v_i = 1|\mathbf{h}, \mathbf{v}_{-i}) = \text{sigm}\left(\sum_k L_{ik} v_k + \sum_l W_{il} h_l + b_i\right)} \tag{11}$$

Symmetrically,

$$\boxed{p(h_j = 1|\mathbf{v}, \mathbf{h}_{-j}) = \text{sigm}\left(\sum_l J_{jl} h_l + \sum_i W_{ij} v_i + c_j\right)} \tag{12}$$

Maximum Likelihood learning

Suppose we have dataset $\mathcal{D} = \{\mathbf{x}_1, \dots \mathbf{x}_N\}, \mathbf{x}_n \in \{0,1\}^D$. The goal is to maximize $\sum_{n=1}^N \log p(\mathbf{x}_n; \boldsymbol{\psi})$ for parameters $\boldsymbol{\psi}$. For a single training example $\mathbf{v}$ and any parameter θ:

$$\frac{\partial}{\partial\theta}\log p(\mathbf{v};\boldsymbol{\psi}) = \frac{\partial}{\partial\theta}\left(\log\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})} - \log\sum_{\widehat{\mathbf{v}},\widehat{\mathbf{h}}} e^{-E(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})}\right) =$$

$$= -\frac{1}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}}\sum_{\mathbf{h}}\left[e^{-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}\cdot\frac{\partial}{\partial\theta}E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})\right] + \frac{1}{\sum_{\widehat{\mathbf{v}},\widehat{\mathbf{h}}} e^{-E(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})}}\sum_{\widehat{\mathbf{v}},\widehat{\mathbf{h}}}\left[e^{-E(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})}\cdot\frac{\partial}{\partial\theta}E(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})\right] =$$

$$= \left[\frac{e^{-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}} = \frac{\frac{1}{Z(\theta)}e^{-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}}{\frac{1}{Z(\theta)}\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}} = \frac{p(\mathbf{h},\mathbf{v};\boldsymbol{\psi})}{p(\mathbf{v};\boldsymbol{\psi})} = p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi})\right] =$$

$$= \underbrace{-\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi})\cdot\frac{\partial}{\partial\theta}E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}_{\mathbb{E}_{\mathbf{h}|\mathbf{v};\boldsymbol{\psi}}\left[\frac{\partial}{\partial\theta}E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})\right]} + \underbrace{\sum_{\widehat{\mathbf{v}},\widehat{\mathbf{h}}} p(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})\cdot\frac{\partial}{\partial\theta}E(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})}_{\mathbb{E}_{\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi}}\left[\frac{\partial}{\partial\theta}E(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})\right]}$$

This we obtain

$$\boxed{\frac{\partial}{\partial\theta}\log p(\mathbf{v};\boldsymbol{\psi}) = -\mathbb{E}_{\mathbf{h}|\mathbf{v};\boldsymbol{\psi}}\left[\frac{\partial E}{\partial\theta}(\mathbf{v},\mathbf{h};\boldsymbol{\psi})\right] + \mathbb{E}_{\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi}}\left[\frac{\partial E}{\partial\theta}(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})\right]} \tag{13}$$

And avegared over all data points:

$$\frac{1}{N}\sum_{n=1}^N\frac{\partial}{\partial\theta}\log p(\mathbf{x}_n;\boldsymbol{\psi}) = -\frac{1}{N}\sum_{n=1}^N\mathbb{E}_{\mathbf{h}|\mathbf{v};\boldsymbol{\psi}}\left[\frac{\partial E}{\partial\theta}(\mathbf{x}_n,\mathbf{h};\boldsymbol{\psi})\right] + \mathbb{E}_{\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi}}\left[\frac{\partial E}{\partial\theta}(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})\right] \tag{14}$$

or solely in terms of expectations:

$$\boxed{\mathbb{E}_{\mathbf{v}\sim P_{\text{data}}(\mathbf{v})}\left[\frac{\partial}{\partial\theta}\log p(\mathbf{v};\boldsymbol{\psi})\right] = -\mathbb{E}_{\mathbf{v},\mathbf{h}\sim P_{\text{data}}(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}\left[\frac{\partial E}{\partial\theta}(\mathbf{v},\mathbf{h};\boldsymbol{\psi})\right] + \mathbb{E}_{\mathbf{v},\mathbf{h}\sim P_{\text{model}}(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}\left[\frac{\partial E}{\partial\theta}(\mathbf{v},\mathbf{h};\boldsymbol{\psi})\right]} \tag{15}$$

where $P_{\text{data}}(\mathbf{v}) = \frac{1}{N}\sum_{n=1}^N \delta_{\mathbf{v},\mathbf{x}_n}$ is *empirical distribution*, $P_{\text{data}}(\mathbf{v},\mathbf{h};\boldsymbol{\psi}) = p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi})P_{\text{data}}(\mathbf{v})$ is *complete-data distribution* and P_{model} is a distribution modelled by BM.

Before we move on, there is one more equivalent form of gradient of data log-likelihood:

$$-\frac{\partial}{\partial\theta}\log p(\mathbf{v};\boldsymbol{\psi}) = \frac{\partial}{\partial\theta}\mathcal{F}(\mathbf{v}) - \sum_{\widehat{\mathbf{v}}} p(\widehat{\mathbf{v}})\frac{\partial}{\partial\theta}\mathcal{F}(\widehat{\mathbf{v}}) \tag{16}$$

Notice that the above gradient contains two terms, which are referred to as the **positive** and **negative** phase. The terms positive and negative do not refer to the sign of each term in the

equation, but rather reflect their effect on the probability density defined by the model. The first term increases the probability of training data (by reducing the corresponding free energy), while the second term decreases the probability of samples generated by the model.

Derivatives of the energy w.r.t. model parameters:

$$E(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi}) = -\frac{1}{2}\mathbf{v}^T\mathbf{L}\mathbf{v} - \frac{1}{2}\mathbf{h}^T\mathbf{J}\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h} - \mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h},$$

- $\frac{\partial E}{\partial \mathbf{L}} = -\mathbf{v}\mathbf{v}^T$ (remember that $\mathbf{L}$ is symmetric)
- Similarly $\frac{\partial E}{\partial \mathbf{J}} = -\mathbf{h}\mathbf{h}^T$ and $\frac{\partial E}{\partial \mathbf{W}} = -\mathbf{v}\mathbf{h}^T$
- Finally $\frac{\partial E}{\partial \mathbf{b}} = -\mathbf{v}$ and $\frac{\partial E}{\partial \mathbf{c}} = -\mathbf{h}$

So 1 update of the weights $\mathbf{W}$ using vanilla gradient ascent looks like:

$$\mathbf{g_W} \leftarrow \mathbb{E}_{\mathbf{v},\mathbf{h}\sim P_{\text{data}}(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}\left[\mathbf{v}\mathbf{h}^T\right] - \mathbb{E}_{\mathbf{v},\mathbf{h}\sim P_{\text{model}}(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}\left[\mathbf{v}\mathbf{h}^T\right] \tag{17}$$

$$\mathbf{W} \leftarrow \mathbf{W} + \alpha \cdot \mathbf{g_W} \tag{18}$$

and similarly for other parameters.

Problems
So far it seems like everything is ok, but there is 1 very serious problem, **both expectations in (13) $\Leftrightarrow$ (15) are simply intractable**. First sum has $O\left(2^H\right)$ terms while the second one has $O\left(2^{V+H}\right)$ terms. Moreover, even $p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi})$ or $p(\mathbf{v},\mathbf{h};\boldsymbol{\psi})$ are intractable because of partition function in denominators which itself has exponentially many in $V+H$ terms.

To overcome this computational burden, G. Hinton and T. Sejnowski in 1983 proposed algorithm that uses Gibbs sampling to approximate both expectations. The idea is that both expectations are approximated by states of 2 Markov chains (Monte-Carlo sampling), which are updated using Gibbs sampling $\forall$ training example during training.

The main problem with the last algorithm is **time** required to approach distribution, especially when estimating the model's expectations, since the Gibbs chain may need to explore a highly multimodal energy landscape + exponential in machine size and magnitude of weights time to collect equilibrium statistics. We will see that in RBM Gibbs sampling can be performed much more efficiently thus having much more efficient learning algorithm.

To summarize, Boltzmann Machines is a Monte Carlo version of Hopfield Net with discrimination between visible and hidden units and that uses annealed Gibbs sampling. Boltzmann Machines were one of the first neural networks capable of learning internal representations, and are able to represent and (given sufficient time) solve difficult combinatoric problems + they are again are considered a biologically plausiable because during learning only local information is used, i.e. the update for a particular weight connecting two units depend only on the statistics of those two units (this can be seen by considering equation (17) element-wisely).

Additional facts

• from formulae (11),(12) we can see that probability of one unit being on is given by a linear model (logistic regression) from the values of the other units.
• in the presence of hidden units, BM becomes a *universal approximator* in a sense that it can learn arbitraty point mass function over discrete variables (without hidden units it could have learned only linear relationships between variables) [7]
• it is also interesting that in this model we can compute $p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi})$ exactly and efficiently, unlike many other models with hidden variables like VAE etc., and still this model can learn fairly complex distributions (+ previous bullet)

Restricted Boltzmann Machines[12, 5, 27, 11, 6, 25, 1]

Intro and comparison with general BM

A *restricted Boltzmann machine* was invented by P. Smolensky in 1986 and is characterized by absense of hid-hid and vis-vis connections (see Fig. 5). In this case we have $\mathbf{L} = \mathbf{J} = \mathbf{0}$ and the

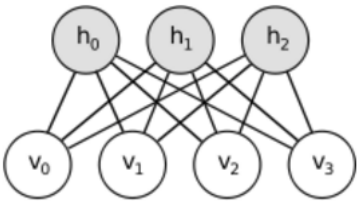


Figure 5: Diagram of a restricted Boltzmann machine with four visible units and three hidden units (no bias units).

energy function for RBM simplifies:

$$E(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi}) = -\mathbf{v}^T\mathbf{W}\mathbf{h} - \mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} \tag{19}$$

Because of the *bipartite* structure of the graph, in this MRF **all hidden units are conditionally independent given visible ones and vice versa** ($\boldsymbol{\psi}$ is omitted for brevity):

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}), \quad p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h}) \tag{20}$$

(Formally every path in this graphical model between different h_j and h_l is blocked by $\mathbf{v}$ and vice versa).

Taking this into account, formulae (11) and (12) simplify too:

$$p(v_i = 1|\mathbf{h}) = \text{sigm}\left(\sum_l W_{il} h_l + b_i\right), \tag{21}$$

$$p(h_j = 1|\mathbf{v}) = \text{sigm}\left(\sum_i W_{ij} v_i + c_j\right) \tag{22}$$

very important that each of these formulae can be computed **in parallel**:

$$p(\mathbf{v} = \mathbf{1}|\mathbf{h}) = \text{sigm}\left(\mathbf{W}\mathbf{h} + \mathbf{b}\right), \tag{23}$$

$$p(\mathbf{h} = \mathbf{1}|\mathbf{v}) = \text{sigm}\left(\mathbf{W}^T\mathbf{v} + \mathbf{c}\right) \tag{24}$$

this makes exact inference tractable (as opposed to general BM).

It is also easier to calculate marginalized distribution of the visible variables (see Fig. 6)

$$\begin{aligned}
p(\boldsymbol{v}) &= \frac{1}{Z}\sum_{\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}) = \frac{1}{Z}\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \\
&= \frac{1}{Z}\sum_{h_1}\sum_{h_2}\cdots\sum_{h_n} e^{\sum_{j=1}^{m} b_j v_j} \prod_{i=1}^{n} e^{h_i\left(c_i + \sum_{j=1}^{m} w_{ij} v_j\right)} \\
&= \frac{1}{Z} e^{\sum_{j=1}^{m} b_j v_j} \sum_{h_1} e^{h_1\left(c_1 + \sum_{j=1}^{m} w_{1j} v_j\right)} \sum_{h_2} e^{h_2\left(c_2 + \sum_{j=1}^{m} w_{2j} v_j\right)} \cdots \sum_{h_n} e^{h_n\left(c_n + \sum_{j=1}^{m} w_{nj} v_j\right)} \\
&= \frac{1}{Z} e^{\sum_{j=1}^{m} b_j v_j} \prod_{i=1}^{n} \sum_{h_i} e^{h_i\left(c_i + \sum_{j=1}^{m} w_{ij} v_j\right)} \\
&= \frac{1}{Z} \prod_{j=1}^{m} e^{b_j v_j} \prod_{i=1}^{n} \left(1 + e^{c_i + \sum_{j=1}^{m} w_{ij} v_j}\right) \qquad (22)
\end{aligned}$$

This equation shows why a (marginalized) RBM can be regarded as a *product of experts* model [15, 39], in which a number of "experts" for individual components of the observations are combined multiplicatively.

Figure 6: $p(\mathbf{v})$

Gradient of Log-Likelihood

Recall

$$\frac{\partial}{\partial\theta}\log p(\mathbf{v};\boldsymbol{\psi}) = -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi})\cdot\frac{\partial}{\partial\theta}E(\mathbf{v},\mathbf{h};\boldsymbol{\psi}) + \sum_{\widehat{\mathbf{v}},\widehat{\mathbf{h}}} p(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})\cdot\frac{\partial}{\partial\theta}E(\widehat{\mathbf{v}},\widehat{\mathbf{h}};\boldsymbol{\psi})$$

First term for $\theta = W_{ij}$ (omit $\boldsymbol{\psi}$):

$$[1] = -\sum_{\mathbf{h}} \prod_{k} p(h_k|\mathbf{v}) \cdot \frac{\partial}{\partial W_{ij}} E(\mathbf{v}, \mathbf{h}) = \left[\frac{\partial}{\partial W_{ij}} E(\mathbf{v}, \mathbf{h}) = -v_i h_j \right] = \sum_{\mathbf{h}} \prod_{k} p(h_k|\mathbf{v}) v_i h_j =$$
$$= \sum_{h_j} \sum_{\mathbf{h}_{-j}} p(h_j|\mathbf{v}) p(\mathbf{h}_{-j}|\mathbf{v}) h_j v_i = \sum_{h_j \in \{0,1\}} h_j p(h_j|\mathbf{v}) v_i \underbrace{\sum_{\mathbf{h}_{-j}} p(\mathbf{h}_{-j}|\mathbf{v})}_{=1} = p(h_j = 1|\mathbf{v}) \cdot v_i$$

The second term

$$[2] = \sum_{\widehat{\mathbf{v}}, \widehat{\mathbf{h}}} p(\widehat{\mathbf{v}}, \widehat{\mathbf{h}}) \cdot \frac{\partial}{\partial \theta} E(\widehat{\mathbf{v}}, \widehat{\mathbf{h}}) = \sum_{\widehat{\mathbf{v}}} p(\widehat{\mathbf{v}}) \left(\sum_{\widehat{\mathbf{h}}} p(\widehat{\mathbf{h}}|\widehat{\mathbf{v}}) \cdot \frac{\partial}{\partial \theta} E(\widehat{\mathbf{v}}, \widehat{\mathbf{h}}) \right)$$

and thus can be computed in the same manner.
Eventually we obtain the following expressions for log-likelihood gradients:

$$\frac{\partial}{\partial W_{ij}} \log p(\mathbf{v}; \boldsymbol{\psi}) = p(h_j = 1|\mathbf{v}) \cdot v_i - \sum_{\widehat{\mathbf{v}}} p(\widehat{\mathbf{v}}) \cdot p(h_j = 1|\widehat{\mathbf{v}}) \cdot \widehat{v}_i \tag{25}$$

$$\frac{\partial}{\partial b_i} \log p(\mathbf{v}; \boldsymbol{\psi}) = v_i - \sum_{\widehat{\mathbf{v}}} p(\widehat{\mathbf{v}}) \cdot \widehat{v}_i \tag{26}$$

$$\frac{\partial}{\partial c_j} \log p(\mathbf{v}; \boldsymbol{\psi}) = p(h_j = 1|\mathbf{v}) - \sum_{\widehat{\mathbf{v}}} p(\widehat{\mathbf{v}}) \cdot p(h_j = 1|\widehat{\mathbf{v}}) \tag{27}$$

+ use formulae (21), (22). Averaged over all training examples, these formulae can also be rewritten in terms of expectations. Note that $\sum_{\widehat{\mathbf{v}}} p(\widehat{\mathbf{v}})[\cdots] = \mathbb{E}_{\widehat{\mathbf{v}} \sim P_{\text{model}}}[\cdots]$.

Despite of the sound simplification and reducing of computational complexity because of the nice factorization, (25) - (27) are still intractable for regular RBMs because of the second terms which are exponential in $\min\{V, H\}$ (if $H < V$ we can factorize $p(\mathbf{v}, \mathbf{h}) = p(\mathbf{h})p(\mathbf{v}|\mathbf{h})$).

Thus, to avoid this complexity, we will approximate expectations by samples from the model distribution using MCMC-based algorithm *Contrastive Divergence* (described below).

Quick summary

	general BM	**RBM**
exact Maximum Likelihood learning	intractable	intractable*
inference	approximate	exact

* – learning still can be done efficiently using Contrastive Divergence.

Contrastive Divergence algorithm and modifications

All common training algorithms for RBMs approximate expectations in log-likelihood gradients (25) - (27) by only single sample from RBM and perform gradient ascent on these approximations.

Samples in RBM can be obtained by running a Markov chain to convergence, using Gibbs sampling as the transition operator. In general, Gibbs sampling of the joint of N random variables $\boldsymbol{\xi} = (\xi_1 \dots \xi_N)$ is done through a sequence of N sampling sub-steps of the form $\xi_i \sim p(\xi_i | \boldsymbol{\xi}_{-i})$. If done sequentially, one can show that these sub-steps define reversible Markov chain with desired invariant joint distribution.

For RBMs, ξ consists of the set of visible and hidden units. However, since they are conditionally independent, $p(v_i|\mathbf{h}, \mathbf{v}_{-i}) = p(v_i|\mathbf{h})$ and $p(h_j|\mathbf{v}, \mathbf{h}_{-j}) = p(h_j|\mathbf{v})$ one can perform **block** Gibbs sampling (which is impossible in BM). In this setting, visible units are sampled simultaneously given fixed values of the hidden units. Similarly, hidden units are sampled simultaneously given the visibles. A step in the Markov chain is thus taken as follows:

$$\mathbf{h}^{(n+1)} \sim \text{Ber}(\text{sigm}\left(\mathbf{W}^T\mathbf{v}^{(n)} + \mathbf{c}\right)), \tag{28}$$

$$\mathbf{v}^{(n+1)} \sim \text{Ber}(\text{sigm}\left(\mathbf{W}\mathbf{h}^{(n+1)} + \mathbf{b}\right)) \tag{29}$$

Graphically:

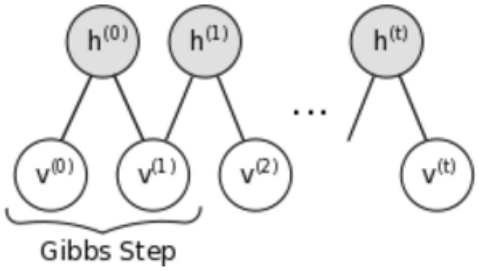


Figure 7: Gibbs sampling in RBM.

As $n \to \infty$, samples $\left(\mathbf{v}^{(n)}, \mathbf{h}^{(n)}\right)$ are guaranteed to be accurate samples of the model joint.

In theory, each parameter update in the learning process would require running one such chain to convergence to obtain unbiased estimates of gradients of log-likelihood, which requires many sampling steps (prohibitively expensive) and moreover it is typically unclear for how long chain has to be run. As such, several algorithms have been devised for RBMs, in order to efficiently sample from.

Contrastive Divergence (CD-k)
This algorithm was proposed in 2002 by G.Hinton and uses two tricks to speed up the sampling process:

- since we eventually want $p_{\text{model}}(\mathbf{v}) \approx p_{\text{data}}(\mathbf{v})$ (Maximum Likelihood $\Leftrightarrow$ Minimizing KL divergence between p_{model} and p_{data} (empirical distribution)), *we initialize the Markov chain with a training example*, so that the chain will be already close to having converged to its desired distribution.

- CD does not wait for the chain to converge. Samples are obtained after only k-steps of Gibbs sampling. In practice, $k = 1$ has been shown to work surprisingly well.

General formula for gradient approximation:

$$\frac{\partial}{\partial\theta}\log p(\mathbf{v}^{(0)};\boldsymbol{\psi}) \approx -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}^{(0)};\boldsymbol{\psi})\cdot\frac{\partial}{\partial\theta}E(\mathbf{v}^{(0)},\mathbf{h};\boldsymbol{\psi}) + \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}^{(k)};\boldsymbol{\psi})\cdot\frac{\partial}{\partial\theta}E(\mathbf{v}^{(k)},\mathbf{h};\boldsymbol{\psi}) \quad (30)$$

Concrete formulae for gradient approximations:

$$g_{W_{ij}} \leftarrow p(h_j = 1|\mathbf{v}^{(0)})\cdot v_i^{(0)} - p(h_j = 1|\mathbf{v}^{(k)})\cdot v_i^{(k)} \quad (31)$$

$$g_{b_i} \leftarrow v_i^{(0)} - v_i^{(k)} \quad (32)$$

$$g_{c_j} \leftarrow p(h_j = 1|\mathbf{v}^{(0)}) - p(h_j = 1|\mathbf{v}^{(k)}) \quad (33)$$

with $\mathbf{v}^{(0)} := \mathbf{v}$ – training example
Each gradient is a difference of the corresponding **positive** and **negative** gradients, see (16). It is thus described through sequence of *positive phases*, where we clamp visible units to a particular state vector sampled from the training set $\mathbf{v} \leftarrow \mathbf{x}_i \sim p_{\text{data}}$, and *negative phases*, where the network is run freely without fixing any states.

Note: originally formulae (31)-(33) should contain instead of $p(h_j = 1|\mathbf{v})$ the samples of hidden units themselves (apart from sampling involved to estimate model's expectations), but stated variants typically provide slightly less noisy and thus faster learning, and it is not so important at all when RBM is used to pretrain hidden layer of units for DBN/DBM [11].

Since $\mathbf{v}^{(k)}$ is not a sample from the stationary distribution the approximation the approximation (30) is biased. Obviously, as $k \to \infty$, bias vanishes. One can also show that CD does not maximize the likelihood of the data under the model, but minimize the difference of two KL-divergences:

$$D_{\text{KL}}(p_{\text{data}}(\mathbf{v}) \parallel p_{\text{model}}(\mathbf{v})) - D_{\text{KL}}(p^{(k)}(\mathbf{v}) \parallel p_{\text{model}}(\mathbf{v})), \quad (34)$$

where $p^{(k)}(\mathbf{v})$ is a distribution of visible variables after k steps of Markov chain. If chain already reached stationarity it holds that $p^{(k)} = p_{\text{model}} \Rightarrow D_{\text{KL}}(p^{(k)}(\mathbf{v}) \parallel p_{\text{model}}(\mathbf{v})) = 0$ and the approximation error of CD is vanishes.

CD does not follow the gradient of any function [11].

Persistent CD In [26] they rely on a single Markov chain, which has a persistent state (i.e., not restarting a chain for each observed example). For each parameter update, we extract new samples by simply running the chain for k-steps. The state of the chain is then preserved for subsequent updates. The general intuition is that if parameter updates are small enough compared to the

mixing rate of the chain, the Markov chain should be able to "catch up" to changes in the model.

Typically one uses as much markov chains as there are training examples in a minibatch. These persistent states of Markov chains can be used to generate samples after training. In [11] they mention PCD learns significantly better models than CD-k for various k and is the recommended method if the aim is to build the best density model of the data. We will use a modification of PCD for DBM training.

Also, recently the new algorithm Parallel tempering is proposed [5]. It introduces supplementary Gibbs chains that sample from more and more smoothed replicas of the original distribution. In the price of computational overhead it gives faster mixing Markov chain and thus less biased gradient approximation.

Extensions

More general RBMs/BMs

So far we considered only case of binary (Bernoulli) both visible and hiddent units. But RBMs (and BMs) can also be extended to model $\mathbb{R}$-valued inputs/outputs. This can be achieved by altering the energy functio (more examples below). More generally RBM can be defined as any MRF (undirected graphical model) with conditionally independent visible units given hidden and vice versa and with energy function of the following kind:

$$E(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi}) = \sum_{i,j} \phi_{ij}(v_i, h_j; \boldsymbol{\psi}) + \sum_i \omega_i(v_i; \boldsymbol{\psi}) + \sum_j \nu_j(h_j; \boldsymbol{\psi}) \tag{35}$$

with $\mathbb{R}$-valued functions $\phi_{ij}, \omega_i, \nu_j$ for which partition function is finite $Z(\boldsymbol{\psi}) < \infty$.

Conditional RBMs

some of the parameters in E are replaced by parameterized functions of some conditioning random variables.

Classification RBMs (cRBMs)

In regular RBM we model $p(\mathbf{x})$. In classification RBM there are couple of choices:

- model $p(\mathbf{x}, \mathbf{y})$ (*generative mode*)
- model $p(\mathbf{y}|\mathbf{x})$ (*discriminative mode*), equivalent to the above via Bayes theorem
- $\alpha \cdot p(\mathbf{x}, \mathbf{y}) + (1 - \alpha) \cdot p(\mathbf{y}|\mathbf{x}) \to \max$ (*hybrid mode*)

See more in [11].

Different types of RBM units

For a binary unit, the probability of turning on is given by the logistic sigmoid function of its total input, x.

$$p = \text{sigm}(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x + e^0} \tag{36}$$

The energy contributed by the unit is $-x$ if it is on and 0 if it is off. Equation (36) makes it clear that the probability of each of the two possible states is proportional to the negative exponential of its energy. This can be generalized to K alternative states.

Softmax units

$$p_j = \text{softmax}_j(\mathbf{x}) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \tag{37}$$

It is the appropriate way to deal with a quantity that has K alternative *mutually exclusive* values which are not ordered in any way. When viewed in this way, the learning rule for the binary units in a softmax is identical to the rule for standard binary units. **The only difference is in the way the probabilities of the states are computed and the samples are taken** + Fig. **??**.

Formally, for binary visible and softmax hidden:

$$\mathbf{v} \in \{0,1\}^D, \mathbf{h} \in \text{One-hot}(H) = \left\{ \mathbf{q} \in \{0,1\}^H \middle| \sum_k q_k = 1 \right\}, H = K$$

Energy function has the same functional form, as for binary-binary RBM:

$$E(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi}) = -\sum_{j,l} W_{jl} v_j h_l - \sum_j b_j v_j - \sum_l c_l h_l \tag{38}$$

Multinomial units

Afurther generalization of the softmax unit is to sample M times (with replacement) from the probability distribution instead of just sampling once. The K different states can then have integer values bigger than 1, but the values must add to M. This is called a *multinomial unit* and, again, the learning rule is unchanged. It is also equivalent [13] to M softmax units with shared weights.

Formally, for binary visible and multinomial hidden with M samples:

$$\mathbf{v} \in \{0,1\}^D, \widehat{\mathbf{h}} \in \left\{ \mathbf{q} \in \{0,1,\ldots,M\}^H \middle| \sum_k q_k = M \right\}, H = K$$

Energy function has the same functional form, as for binary-binary RBM:

$$E(\mathbf{v}, \widehat{\mathbf{h}}; \boldsymbol{\psi}) = -\sum_{j,l} W_{jl} v_j \widehat{h}_l - \sum_j b_j v_j - \sum_l c_l \widehat{h}_l, \tag{39}$$

where $\widehat{h}_l = \sum_{m=1}^{M} h_l^{(m)}$ – count for l-th discrete value of hidden units. Typically, this variant of RBM is used for topic modelling [13] (visible are Multinomial, hidden – binary). Often M is equal to K. In this case, it is also useful to scale hidden bias term by M, this will allow to behave sensible when deal with documents of the different lengths.

Gaussian visible units

To be able to model real-valued data $\mathbf{v} \in \mathbb{R}^D$, one solution is to replace the binary visible units by linear units with independent Gaussian noise. The energy function then becomes:

$$E(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi}) = \frac{1}{2} \sum_{i,j} \frac{(v_i - b_i)^2}{\sigma_i^2} - \sum_j c_j h_j - \sum_{i,j} W_{ij} \frac{v_i}{\sigma_i} h_j \tag{40}$$

where σ_i is the standard deviation of the Gaussian noise for visible unit i. It is possible to learn the variance of the noise for each visible unit but this is difficult using CD-k. In many applications, it is much easier to first normalise each component of the data to have zero mean and unit variance and then to use noise free reconstructions, with the variance in equation (40) set to 1.

From (38) one can derive formulae for activations [24, 15]:

$$p(h_j = 1|\mathbf{v}) = \text{sigm}\left(\sum_i W_{ij} \frac{v_i}{\sigma_i} + c_j\right), \tag{41}$$

$$v_i|\mathbf{h} \sim \mathcal{N}\left(\sigma_i \sum_j W_{ij} h_j + b_i;\ \sigma_i^2\right) = \sigma_i \sum_j W_{ij} h_j + b_i + \sigma_i \cdot \mathcal{N}(0; 1) \tag{42}$$

There also exist other types of units, such as gaussian for both visible and hidden units, binomial units, rectifier linear units etc., but they more rarely used and are out of the scope of these notes. See more in [11].

Free energies formulae

• Free energy for binary visible and hidden units (19) (Similarly to Fig. 6):

$$\mathcal{F}(\mathbf{v};\boldsymbol{\psi}) = -\log\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}\right) = -\log\sum_{\mathbf{h}}\exp\left(\sum_{i,j} W_{ij}v_i h_j + \sum_i b_i v_i + \sum_j c_j h_j\right) =$$

$$= -\log\left[\exp\left(\sum_i b_i v_i\right)\cdot\sum_{\mathbf{h}}\exp\left(\sum_{i,j} W_{ij}v_i h_j + \sum_j c_j h_j\right)\right] =$$

$$= -\sum_i b_i v_i - \log\sum_{\mathbf{h}}\underbrace{\exp\left(\sum_j h_j\left(\sum_i W_{ij}v_i + c_j\right)\right)}_{\prod_j \exp\left(h_j\left(\sum_i W_{ij}v_i + c_j\right)\right)} = \left[\sum_{\mathbf{h}}\prod_j f_j(h_j) = \prod_j\sum_{h_j} f_j(h_j)\right] =$$

$$= -\mathbf{b}\cdot\mathbf{v} - \sum_j \log\left(\underbrace{\sum_{h_j\in\{0,1\}}\exp\left[h_i\sum_i W_{ij}v_i + c_j\right]}_{1+\exp\left(\sum_i W_{ij}v_i + c_j\right)}\right) = -\mathbf{b}\cdot\mathbf{v} - \sum_j \text{softplus}\left(\sum_i W_{ij}v_i + c_j\right),$$

where $\text{softplus}(x) := \log(1+e^x)$.

$$\boxed{\mathcal{F}(\mathbf{v};\boldsymbol{\psi}) = -\mathbf{b}\cdot\mathbf{v} - \sum_j \text{softplus}\left(\sum_i W_{ij}v_i + c_j\right)} \tag{43}$$

• Free energy for Bernoulli-Softmax RBM (38):

$$\mathcal{F}(\mathbf{v};\boldsymbol{\psi}) = -\log\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}\right) = -\sum_i b_i v_i - \log\sum_{\mathbf{h}}\underbrace{\exp\left(\sum_j h_j\left(\sum_i W_{ij}v_i + c_j\right)\right)}_{\prod_j \exp\left(h_j\left(\sum_i W_{ij}v_i + c_j\right)\right)} =$$

$$= -\mathbf{b}\cdot\mathbf{v} - \log\sum_{j=1}^{K}\exp\left(\sum_i W_{ij}v_i + c_j\right),$$

$$\boxed{\mathcal{F}(\mathbf{v};\boldsymbol{\psi}) = -\mathbf{b}\cdot\mathbf{v} - \log\sum_{j=1}^{K}\exp\left(\sum_i W_{ij}v_i + c_j\right)} \tag{44}$$

• Free energy for Bernoulli-Multinomial RBM (39):

$$\mathcal{F}(\mathbf{v};\boldsymbol{\psi}) = -\log\left(\sum_{\widehat{\mathbf{h}}} e^{-E(\mathbf{v},\widehat{\mathbf{h}};\boldsymbol{\psi})}\right) = -\sum_i b_i v_i - \log \sum_{\widehat{\mathbf{h}}} \underbrace{\exp\left(\sum_j \widehat{h}_j \left(\sum_i W_{ij} v_i + c_j\right)\right)}_{\prod_j \exp\left(\widehat{h}_j \left(\sum_i W_{ij} v_i + c_j\right)\right)} =$$

$$= -\mathbf{b}\cdot\mathbf{v} - \log \sum_{\widehat{h}_1+\ldots+\widehat{h}_K=M, 0\le\widehat{h}_l\le M} \prod_j \exp\left(\widehat{h}_j \left(\sum_i W_{ij} v_i + c_j\right)\right),$$

For large M this equation seems to be intractable to compute. But we can approximate this by sampling $\widehat{\mathbf{h}}$ from Multinomial distribution with equiprobable states and applying appropriate scaling, note that

$$\left|\left\{\mathbf{q}\in\{0,1,\ldots,M\}^H \middle| \sum_k q_k = M\right\}\right| = \#_{M,K} = \binom{M+K-1}{K-1} = \frac{\Gamma(M+K)}{\Gamma(M+1)\Gamma(K)}$$

So

$$\widehat{\mathbf{h}} \sim \text{Multinomial}\left(\mathbf{p} = \left(\frac{1}{K}\ldots\frac{1}{K}\right); M\right),$$

$$\mathcal{F}(\mathbf{v};\boldsymbol{\psi}) \approx -\mathbf{b}\cdot\mathbf{v} - \log\left(\frac{\Gamma(M+K)}{\Gamma(M+1)\Gamma(K)}\right) - \sum_j \widehat{h}_j \left(\sum_i W_{ij} v_i + c_j\right)$$

$$\boxed{\mathcal{F}(\mathbf{v};\boldsymbol{\psi}) \approx -\mathbf{b}\cdot\mathbf{v} - \texttt{lgamma}(M+K) + \texttt{lgamma}(M+1) + \texttt{lgamma}(K) - \sum_j \widehat{h}_j \left(\sum_i W_{ij} v_i + c_j\right)} \tag{45}$$

• Free energy for Gaussian-Bernoulli RBM:
Derivation is straightforward, the formula is very similar to (43), but with accordingly changed bias term for visible units, and scaled visible units by their resp. std. deviation:

$$\boxed{\mathcal{F}(\mathbf{v};\boldsymbol{\psi}) = \frac{1}{2}\left\|\frac{\mathbf{v}-\mathbf{b}}{\boldsymbol{\sigma}}\right\|^2 - \sum_j \text{softplus}\left(\sum_i W_{ij}\frac{v_i}{\sigma_i} + c_j\right)} \tag{46}$$

or equivalently if $\widehat{\mathbf{v}} \leftarrow \mathbf{v}/\boldsymbol{\sigma}$ (element-wise division, and also in (46),(47)):

$$\boxed{\mathcal{F}(\mathbf{v};\boldsymbol{\psi}) = \frac{1}{2}\left\|\widehat{\mathbf{v}} - \frac{\mathbf{b}}{\boldsymbol{\sigma}}\right\|^2 - \sum_j \text{softplus}\left(\sum_i W_{ij}\widehat{v_i} + c_j\right)} \tag{47}$$

Practical aspects, implementation and experiments (RBM) [12, 27, 11, 29, 7]

If not states otherwise, Bernoulli-Bernoulli RBM is assumed.

How to debug/track progress

RBMs are particularly tricky to train. Because of the partition function Z we cannot estimate the log-likelihood $\log p(\mathbf{v})$ directly during training. We therefore have no direct useful metric for choosing the optimal hyperparameters.

Inspection of Negative particles

Negative samples (samples used to estimate negative phase of the log-likelihood gradient) obtained during training can be visualized. As training progresses, we know that the model defined by the RBM becomes closer to the true underlying distribution. For example, see Fig. 8.

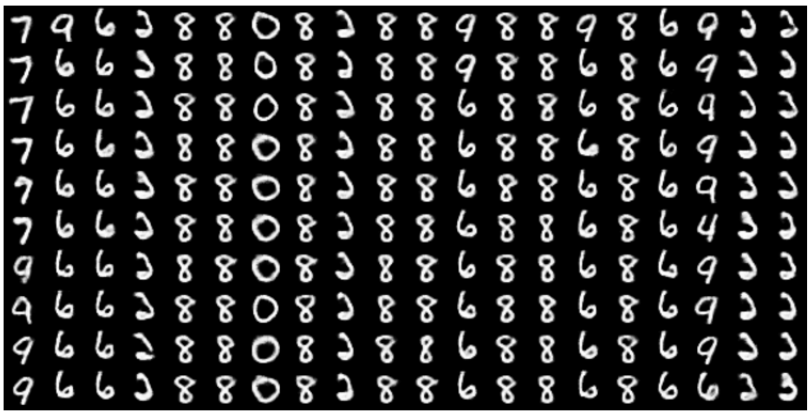


Figure 8: Samples generated by the RBM after PCD-15 training on MNIST. Each row represents a mini-batch of negative particles (samples from independent Gibbs chains). 1000 steps of Gibbs sampling were taken between each of those rows.

Visual Inspection of Filters

The filters learnt by the model can be visualized. This amounts to plotting the weights of each unit as a gray-scale image (after reshaping to a square matrix). Filters should pick out strong features in the data. While it is not clear for an arbitrary dataset, what these features should look like, training on MNIST usually results in filters which act as stroke detectors, while training on natural images lead to Gabor like filters if trained in conjunction with a **sparsity criteria** (more details in next subsections). For example, see Fig. 9. It is worth noting that filters may or may not be *sparse*. If filters are sparse, they will respond to very local features. Dense filters,

on the other hand, respond to stimuli across the entire filter. Although the two types of filter are qualitatively different, in [29] they observed cases in which both types are successful in learning the underlying density.

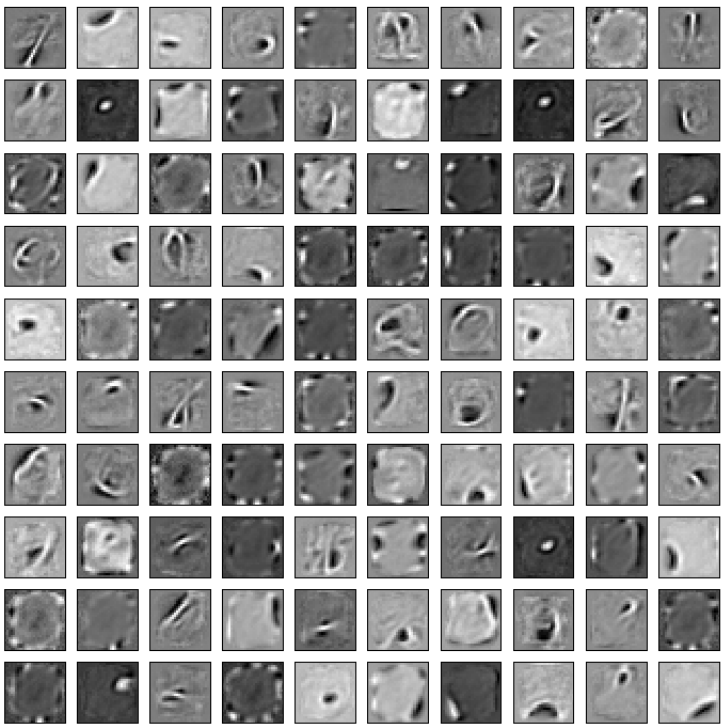


Figure 9: Filters learned by RBM after CD-1 training on MNIST.

Probabilities of hidden activations

It is useful to plot activations of hidden neurons to see how they are being used, how often they are likely to be on vs. off, are there any "dead" neurons (which are always on of always off) etc. This can be effectively done by plotting greyscale matrix of probabilities of (some) hidden units for each of the input example in a minibatch, for instance. For example, see Fig. 10.

Before any training, the probability plot should be mostly a flat gray, perhaps with a little visible noise. That is, most hidden probabilities should be around .5, with some as low as .4 or as high as .6. If the plot is all black (near 0) or all white (near 1), the weights or hidden biases were set incorrectly. The weights $\mathbf{W}$ should initially be random and centered at 0, and hidden biases should

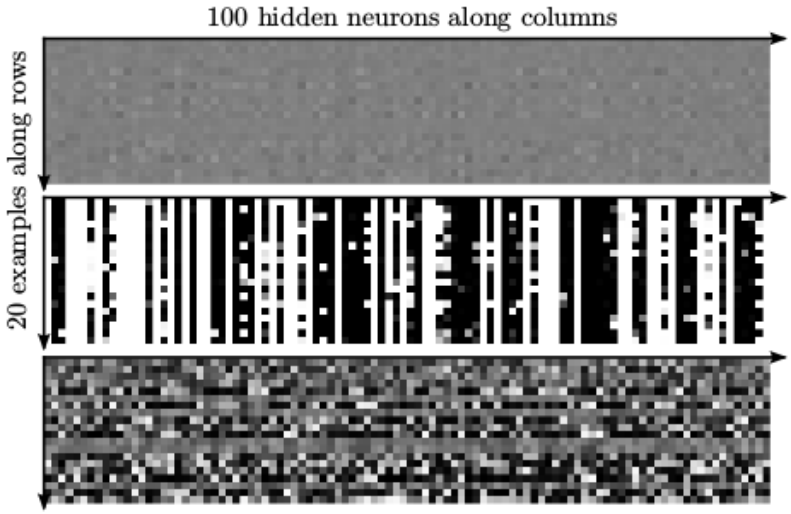


Figure 10: Hidden neuron activation probabilities for the first 100 neurons (of 1,000) and the first 20 example data points (of 50,000), where black represents p = 0 and white, p = 1. Each row shows different neurons activations for a given input example, and each column shows a given neurons activations across many examples. Top: correct dithered gray before training begins. Values are mostly in [.4, .6]. Middle: Values pegged to black or white after one mini-batch. Decrease initial **W** values or learning rate. Bottom: the learning has converged well after 45 epochs of training.

be zero or at least centered at zero. If the probability plot contains both pixels pegged to black and pixels pegged to white, then the **W** has been initialized with values too large. Intuitively, the problem with this case is that all hidden neurons have already determined what features they are looking for before seeing any of the data.
TL;DR: when learning is working properly, this display should look thoroughly random w/o any obvious vertical or horizontal lines.

Distribution/histograms of weights and weights updates

In addition to the hidden probability plots above, it is often useful to monitor distributions of the model parameters (weights and biases) themselves, and their corresponding parameters **updates** (for vanilla SGA (stoch. gradient ascent) they are gradient×learning rate, and slightly more complicated formulae when using momentum or others more advanced variants of SGA). Under normal, desired conditions in the middle of training, all histograms should look roughly Gaussian in shape, and the mean magnitudes of parameters updates should be smaller by a factor of 10^2 to 10^4 than mean magnitudes of the respective parameters. If the change in weights is too small (i.e. a separation of more than 10^4), then the learning rate can probably be increased. If the change in weights is too large, the learning may explode and the weights diverge to infinity.

Proxies to Likelihood

Other, more tractable functions can be used as a proxy to the likelihood. When training an RBM with CD/PCD, one can use **pseudo-likelihood** as the proxy. Pseudo-likelihood (PL) is much less expensive to compute, as it assumes that all bits are independent

$$\mathrm{PL}(\mathbf{v}) = \prod_{i=1}^{D} p(v_i|\mathbf{v}_{-i}), \text{ and} \tag{48}$$

$$\mathrm{PLL}(\mathbf{v}) := \log \mathrm{PL}(\mathbf{v}) = \sum_{i=1}^{D} \log p(v_i|\mathbf{v}_{-i}) \tag{49}$$

[Compare to $\log p(\mathbf{v}) = \log p(v_1) + \log p(v_2|v_1) + \ldots + \log p(v_D|\mathbf{v}_{1:D-1})$].

For general probabilistic graphical model, $\log p(v_i|\mathbf{v}_{-i}) = \log p(v_i|\mathbf{v}_{\mathcal{N}(i)})$, where $\mathcal{N}(i)$ – set of neighbors of variable i. One can show that estimation by maximizing the pseudolikelihood is asymptotically consistent [7].

Lets calculate $p(v_i = 1|\mathbf{v}_{-i})$ for RBM (omit $\boldsymbol{\psi}$ for brevity)

$$p(v_i = 1|\mathbf{v}_{-i}) = \frac{p(v_i = 1, \mathbf{v}_{-i})}{p(\mathbf{v}_{-i})} = \frac{p(v_i = 1, \mathbf{v}_{-i})}{p(v_i = 1, \mathbf{v}_{-i}) + p(v_i = 0, \mathbf{v}_{-i})} = \frac{\frac{1}{Z}e^{-\mathcal{F}(v_i=1,\mathbf{v}_{-i})}}{\frac{1}{Z}e^{-\mathcal{F}(v_i=1,\mathbf{v}_{-i})} + \frac{1}{Z}e^{-\mathcal{F}(v_i=0,\mathbf{v}_{-i})}} =$$
$$= \frac{e^{-\mathcal{F}(v_i=1,\mathbf{v}_{-i})}}{e^{-\mathcal{F}(v_i=1,\mathbf{v}_{-i})} + e^{-\mathcal{F}(v_i=0,\mathbf{v}_{-i})}} = \frac{1}{1 + e^{-[\mathcal{F}(v_i=0,\mathbf{v}_{-i}) - \mathcal{F}(v_i=1,\mathbf{v}_{-i})]}} = \mathrm{sigm}\left[\mathcal{F}(v_i = 0, \mathbf{v}_{-i}) - \mathcal{F}(v_i = 1, \mathbf{v}_{-i})\right]$$

So:

$$\boxed{p(v_i = 1|\mathbf{v}_{-i}) = \mathrm{sigm}\left[\mathcal{F}(v_i = 0, \mathbf{v}_{-i}) - \mathcal{F}(v_i = 1, \mathbf{v}_{-i}))\right]} \tag{50}$$

Note how this formula (and derivation) resembles formulae (11),(12). This is because free energy is designed as $p(\mathbf{v}) \propto e^{-\mathcal{F}(\mathbf{v})} =: p^*(\mathbf{v})$, similarly to $p(\mathbf{v}, \mathbf{h}) \propto e^{-E(\mathbf{v},\mathbf{h})} =: p^*(\mathbf{v}, \mathbf{h})$. We also reduced complexity to only $O(V)$ invocations of unnormalized probability $p^*(\mathbf{v})$ instead of $O(2^{V+H})$ invocations of $p^*(\mathbf{v}, \mathbf{h})$ to compute partition function for true log-likelihood. Using equations (49),(50),(43), we can now compute PLL, which is the sum of the log-probabilities of each bit/feature v_i, conditioned on the state of all other bits. For moderate D (e.g. for MNIST $D = 784$), this sum remains rather expensive. For this reason, the following stochastic approximation to PLL is often used [27, 18]:

$$\boxed{\widehat{\mathrm{PLL}}(\mathbf{v}) = D \cdot \log \mathrm{sigm}\left(\mathcal{F}(\widehat{\mathbf{v}}_\tau) - \mathcal{F}(\mathbf{v})\right), \quad \tau \sim U(\{1 \ldots D\})}, \tag{51}$$

where $\widehat{\mathbf{v}}_i$ is $\mathbf{v}$ with i-th bit flipped ($0 \to 1$, $1 \to 0$).
Note: $\mathbb{E}_\tau[\widehat{\mathrm{PLL}}(\mathbf{v})] = \mathrm{PLL}(\mathbf{v})$.
Note: Do not compute $\log \mathrm{sigm}(x)$ naively, it is numerically unstable operation. Either observe

that $\log \text{sigm}(x) = -\text{softplus}(-x)$ and use numerically stable implementation of softplus or use built-in functions (e.g. `tf.log_logistic` in TensorFlow ≥ 1.2).

This stochastic approximation uses only $2 = O(1)$ invocations of $p^*(\mathbf{v})$/free-energy **per training example**. Note that typically one compute average free-energy over all (or subset of) training set.

Note Some also monitor *reconstruction error* during training. Although it is convenient, it is poor measure of actual RBM performance, since this is not the function Contrastive Divergence optimizes. However, typically, nicely trained model has low reconstruction error.

Also there is Markov-chain based method to estimate partition function of RBM called *Annealed Importance Sampling* [24], which can be used to estimate log-probabilities of validation data directly. We will use this methods for DBMs.

Monitoring overfitting: Free energy gap

Unfortunately, for large RBMs, it is very difficult to compute this probability because it requires knowledge of the partition function. Nevertheless, it is possible to directly monitor the overfitting by comparing the free energies of training data and held out validation data. In this comparison, the partition function cancels out. As we saw, the free energy of a data vector can be computed in a time that is linear in the number of hidden units. As the model starts to overfit the average free energy of the validation data will rise relative to the average free energy of the training data and this *gap* represents the amount of overfitting. (Since $p(\mathbf{v}) \propto e^{-\mathcal{F}(\mathbf{v})}$ this means network will put noticeably higher probability to training data). Free energy can be computed for subset of training (and validation data), but the same subsets should be used for the whole training.

Choice of hyperparameters, various tricks to improve/speedup learning

Most of the tricks and recipes are taken from [11].

Number of Gibbs steps

In theory the more steps we use to estimate model expectations, the better approximation of gradients we should obtain, and better learning should be observed.

[✗] However, in my experiments on MNIST, more steps decrease performance for both RBM and DBM in terms of both pseudo-loglik and quality of filters and samples.
• In [11] they suggest to gradually increase k in CD-k during training. This is not currently implemented.

Updating hidden and visible states

The question can we or should we use probabilities instead of samples in update rules of contrastive divergence (31)-(33). It is very important to make hidden units driven by data stochastic – this will act as a strong regularizer (network won't be able to communicate real values to the hidden

units – information bottleneck). Further updates of hidden units and all updates of visible units should use probabilities instead of sampling for less noisy and faster learning.

[✗] In the experiments if data-driven hidden units were sampled and for the rest probabilities were used, this model had the lowest pseudo-likelihood and sharp filters, see Fig. 11. This is typically means overfitting

• If probabilities were used only for visible units, this model had the highest PLL, but still some of the filters were sharp. This model is best suited for supervised tasks (e.g. finetuning using backprop for classification)

• If both hidden and visible units are sampled when estimating model statistics, the learned filters are smooth and large number of them are sparse. This model has slightly worse PLL, thus we observe underfitting. When RBMs are trained to learn compound models, such as DBMs/DBNs, generative model is not the ultimate objective and it may be possible to save time by underfitting it.

[✓] Data-driven hidden states should absolutely be sampled.

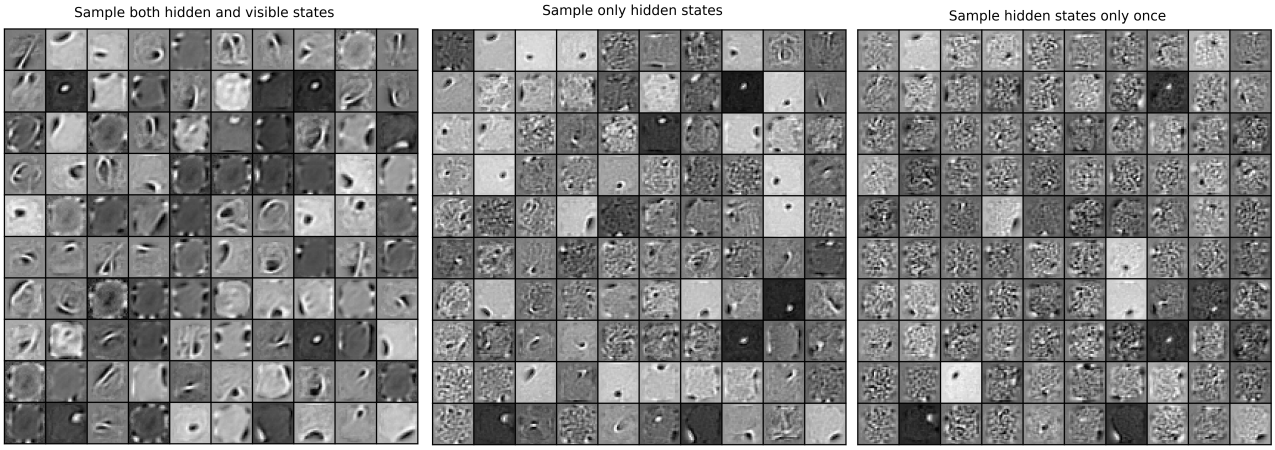


Figure 11: RBM filters trained on MNIST depending on whether to sample hidden and/or visible units.

Minibatch size

[✓] It is a serious mistake to make batchsizes too large. Batch size of 1 is ok, but more efficient 10-100 for more efficient computations using linalg libraries. For datasets that contain a small number of equiprobable classes, the ideal mini-batch size is often equal to the number of classes and each mini-batch should contain one example of each class.

Learning rate

[✓] Shouldn't be too high (weights could explode) and not too low (slow learning). Typically it should be from 10^{-4} to 10^{-2} of order of magnitude of weights (check histogram).

[✓] For Gaussian visible units the learning rate needs to be about one or two orders of magnitude smaller than when using binary visible units.
• For larger k in CD-k and in case of Persistent CD-k it is reasonable to slightly reduce learning rate.

Initial values of the weights and biases

[✓] weights are typically initialized to small zero-centered Gaussian random variables with std of about 0.01 (too large could speedup learning but also can cause "died" units (with either too large or too small activation probabilities), which slows learning).
• weights can also be initialized to zeros, but only if hidden states are sampled (this way they still will be different even if they initially had identical connectivities)
[✓] it is usually helpful to initialize the bias of visible unit i to $\log[p_i/(1-p_i)] = \text{sigm}^{-1}(p_i)$, where p_i is the proportion of training vectors in which unit i is on. This will ensure that in early stages of training visible units will have activation probabilities close to p_i. Models with such initialization had noticeably higher PLL on MNIST.
• $\log[t/(1-t)] = \text{sigm}^{-1}(t)$ for hidden units if sparsity target t is used, otherwise zero.
[✗] Do not initialize hidden biases to large negative values, like -4 to crudely encourage sparsity. Such models had lower PLL and worse filters.

Momentum method

[✓] Using momentum dramatically speedups learning. In [11] and [2] they suggest to start with 0.5 and after couple of epochs to instantly increase momentum to 0.9. I found that gradually increasing momentum from 0.5 to 0.9 is slightly better. Common constant schedules for momentum, like 0.9 are also fine.

Weight decay

[✓] It definitely helps, and there are multiple reasons to use weight decay (not only to prevent overfitting!):

- to improve generalization by preventing overfitting
- "unstick" hidden units that were either firmly "on" or "off"
- improve mixing rate of Gibbs chain which makes CD better approximate log-likelihood gradients etc.

Good values for weight decay coefficient are $10^{-5} \dots 10^{-2}$.

Different regularization techniques

• L1 weight-decay often leads to strongly localized receptive fields (many of the weights to become exactly zero whilst allowing a few of the weights to grow quite large, this can make it easier to interpret the weights).

• Maxnorm helps to avoid hidden units getting stuck with extremely small weights, but a sparsity target is probably a better way to avoid this problem. Maxnorm will play more important role in DBM training, though.
• Dropout: does not seem to help in any way.

Encouraging sparse hidden activities: sparsity targets

Sparse activities of the binary hidden units can be achieved by specifying a "sparsity target" which is the desired probability of being active, $t << 1$ (typically $t = 0.01$ to $t = 0.1$. An additional penalty term is then used to encourage the actual probability of being active q to be close to t. q is estimated by using an exponentially decaying average of the mean probability that a unit is active in each mini-batch:

$$q \leftarrow \omega q + (1 - \omega) q_{\text{current}}, \tag{52}$$

where q_{current} is the mean activation probability of the hidden unit on the current mini-batch, decay rate ω is typically 0.9 to 0.99. For logistic hidden units the natural penalty measure to use is the cross entropy between the desired and actual distributions:

$$\text{Sparsity penalty} = \lambda \times \left[-t \log q - (1 - t) \log(1 - q)\right], \tag{53}$$

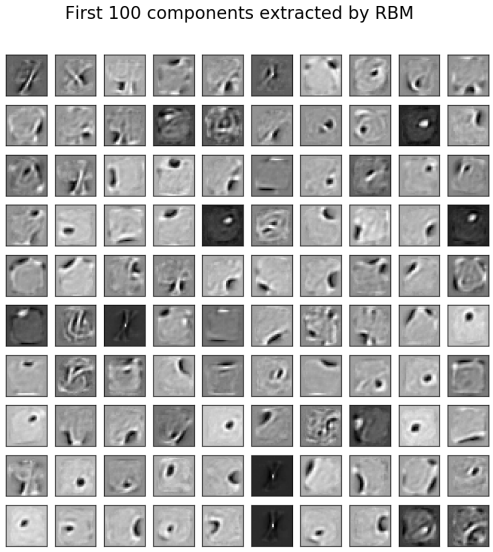
where λ is a hyperparameter called "sparsity cost". (53) has simple derivative of $\lambda(q - t)$ w.r.t. total input of a unit. It is imprtant to apply the same derivative to both weights and hidden biases. Histogram the mean activities of the hidden units and set the sparsity-cost so that the hidden units have mean probabilities in the vicinity of the target. If the probabilities are tightly clustered around the target value, reduce the sparsity-cost so that it interferes less with the main objective of the learning.

Number of hidden units

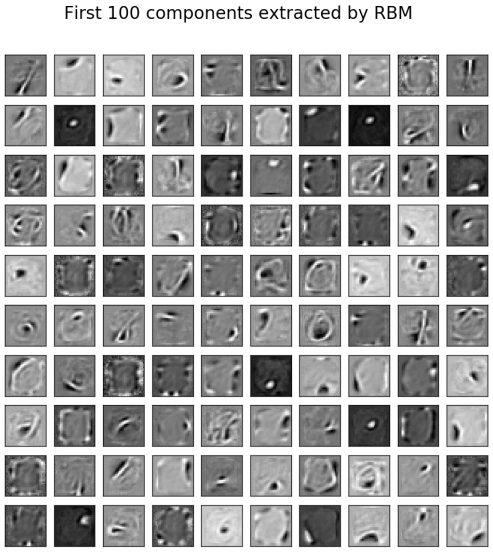
[✓] It is common and useful to use more hidden units than visible, especially in conjunction with a sparsity target. (In discriminative learning labels usually contain very few bits of information, so using more parameters than training cases will typically cause severe overfitting; when learning generative models of high-dimensional data, however, it is the number of bits that it takes to specify a data vector that determines how much constraint each training case imposes on the parameters of the model. This can be several orders of magnitude greater than number of bits required to specify a label.)

Experiments

Also check jupyter notebooks in github repo. All RBMs here are trained on MNIST.



(a) 256 hidden layers



(b) 1024 hidden layers

Figure 12: Different number of hidden layers

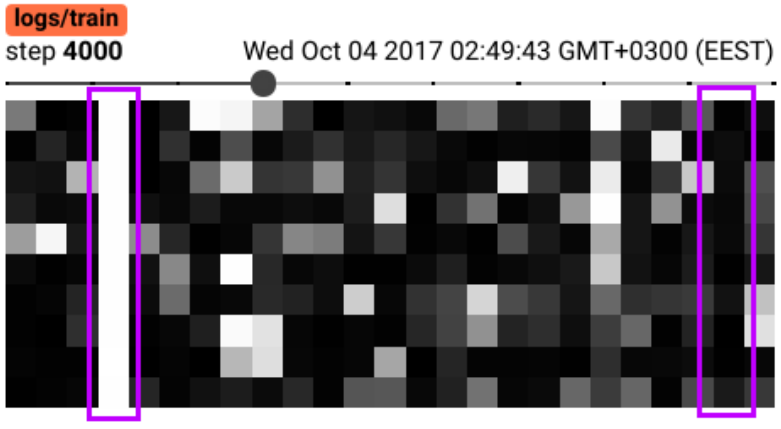


Figure 13: Dropout $p = 0.9$ kills some hidden units.

Figure 14: PLL of RBMs trained using various approaches on 10k MNIST digits.

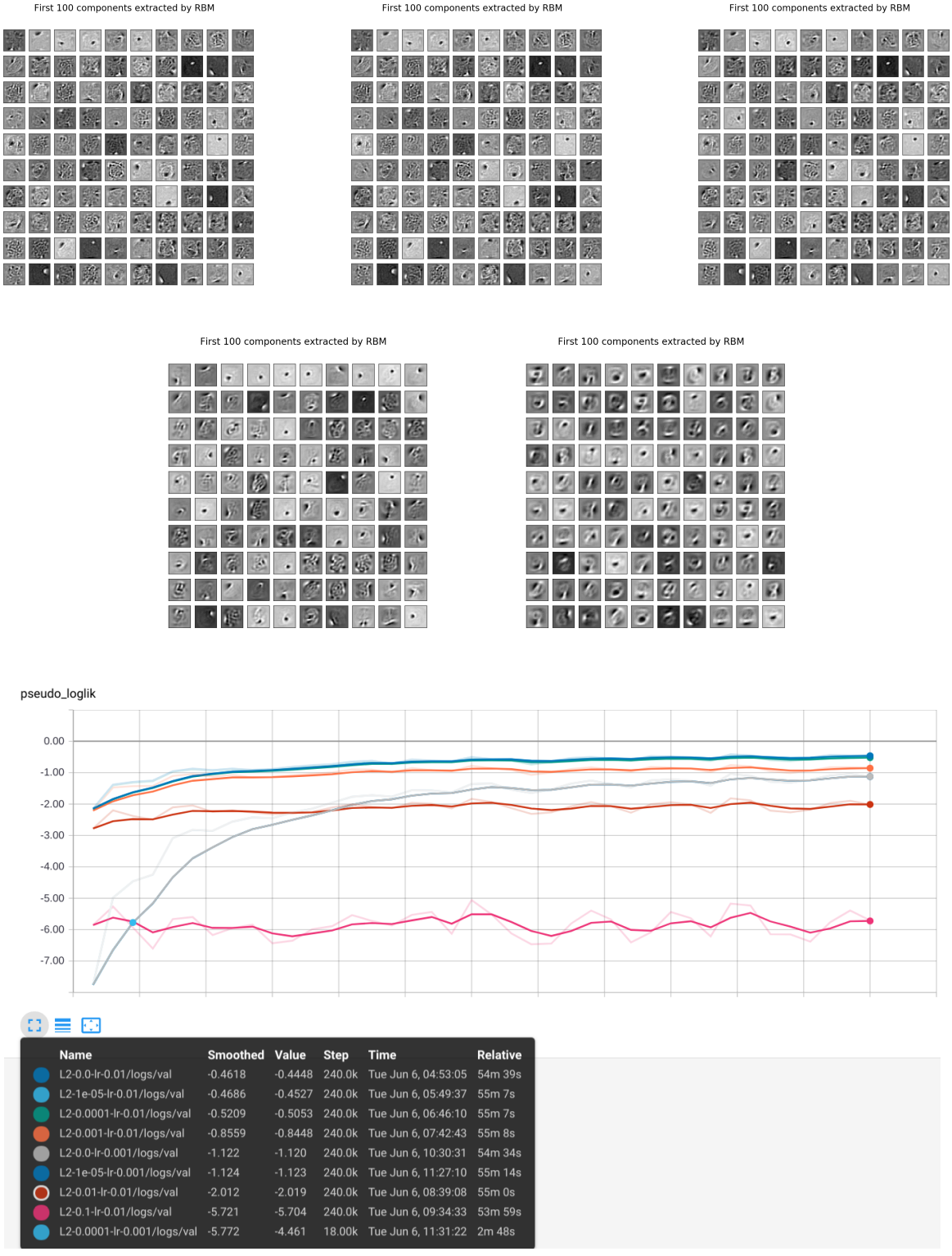


Figure 15: Different amount of regularization. *Left to right, top to bottom*, $\lambda = 10^{-5} \dots 10^{-1}$. *Bottom*: PLL of RBM as λ varies.

Figure 16: PLL (top) and Reconstruction error (bottom) of RBMs trained using various approaches on full MNIST dataset. It is an example when you shouldn't trust reconstruction error: models with lowest values of this quantity have very poor PLL.

Figure 17: PLL and reconstruction error as learning rate varies (full MNIST).

Figure 18: PLL and free energy gap for sparsity target $t \in \{0, 1\}$ and sparsity cost $\lambda \in \{0, 10^{-5:-2}\}$ are used (full MNIST).

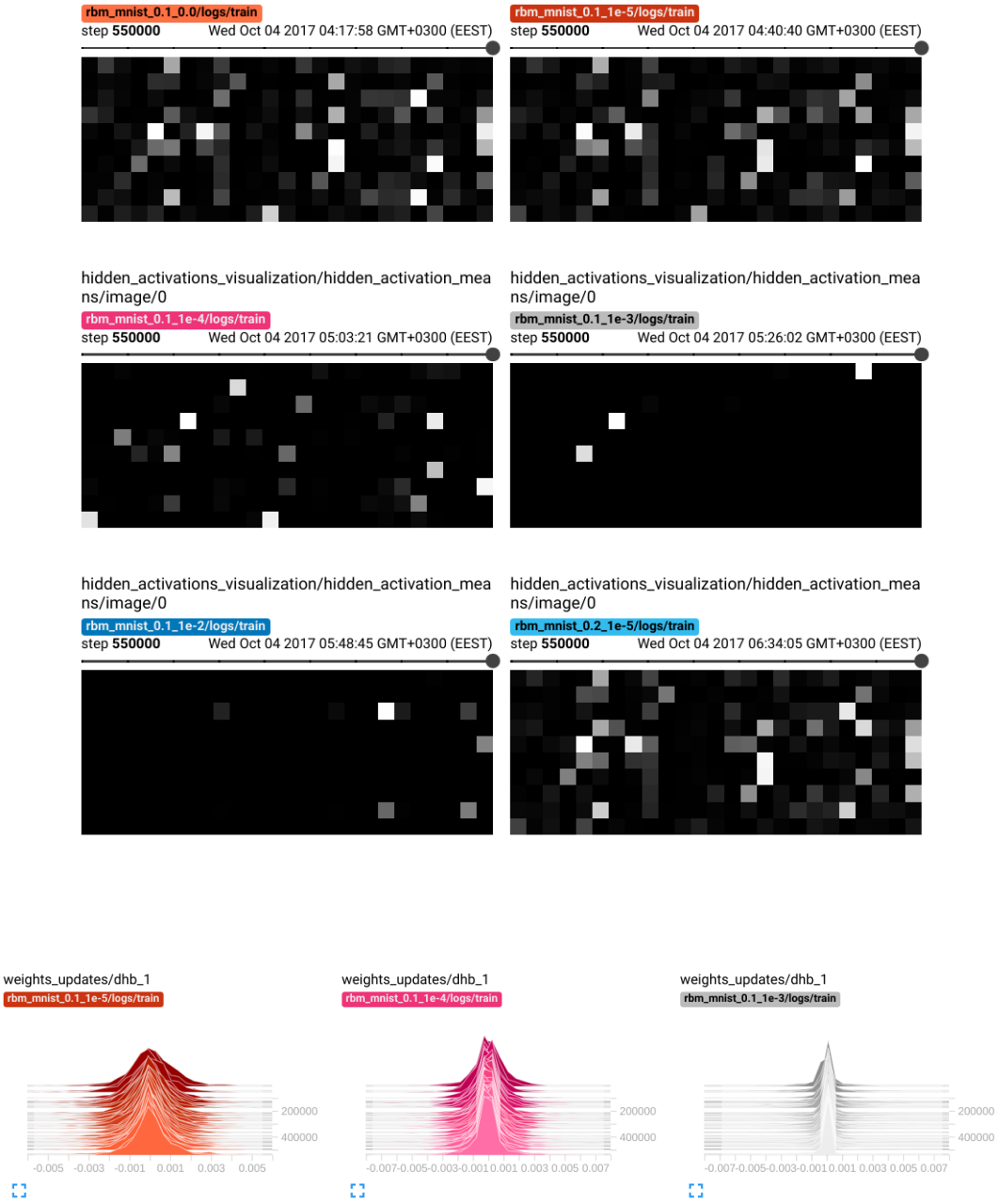
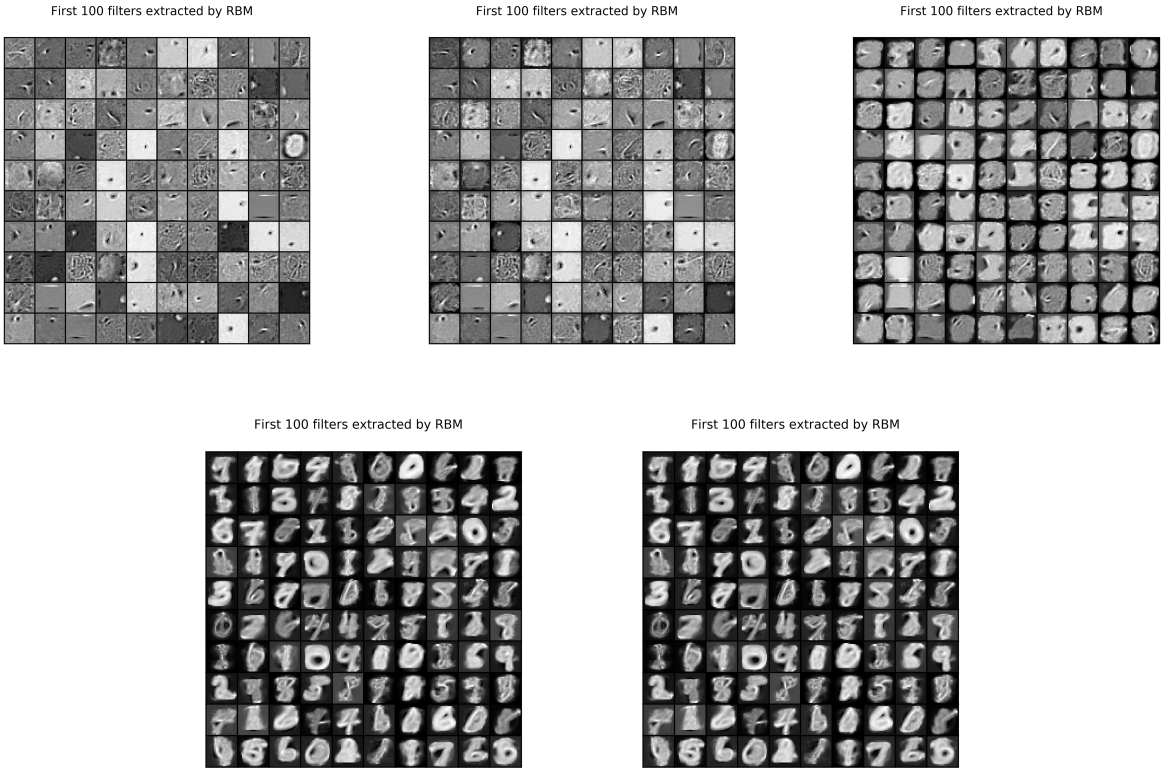


Figure 19: Hidden activations and hidden bias updates distributions from previous experiment. We clearly see that for larger λ hidden activities become more and more sparse and distribtion becomes more and more peaky (same for weight updates).



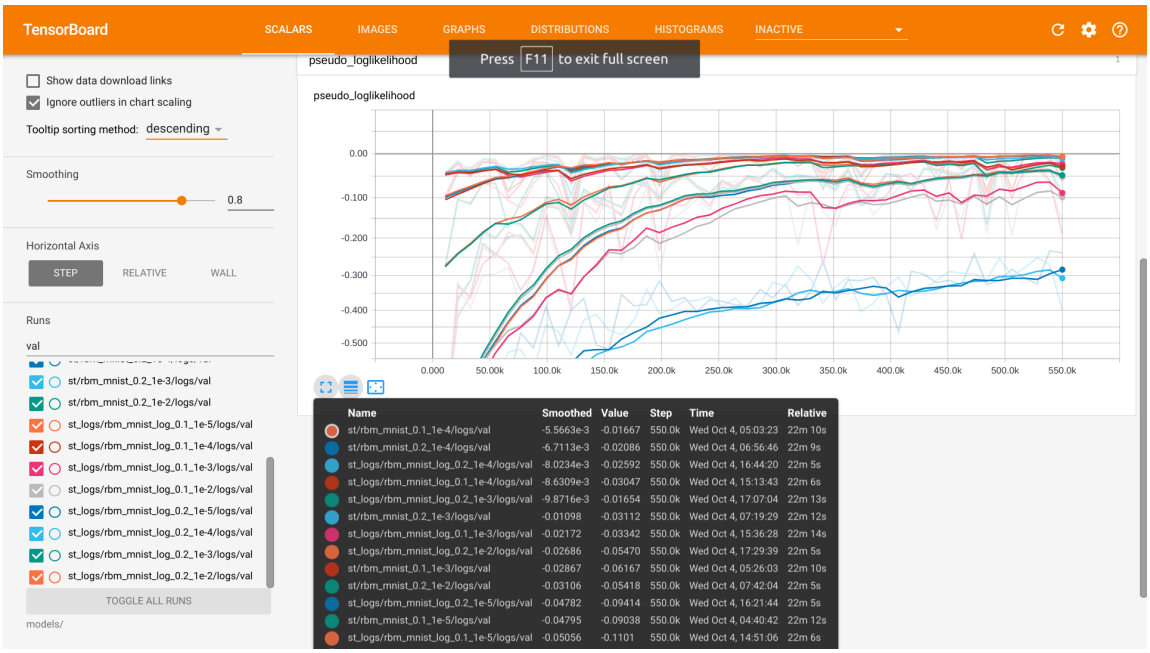


Figure 20: *Top*: filters learned in the previous experiment as λ increases (from top to bottom, from left to right). *Bottom*: PLL of best models from previous experiment and ones with hidden biases initialized as $\log\left(\frac{t}{1-t}\right)$ for sparsity target t.

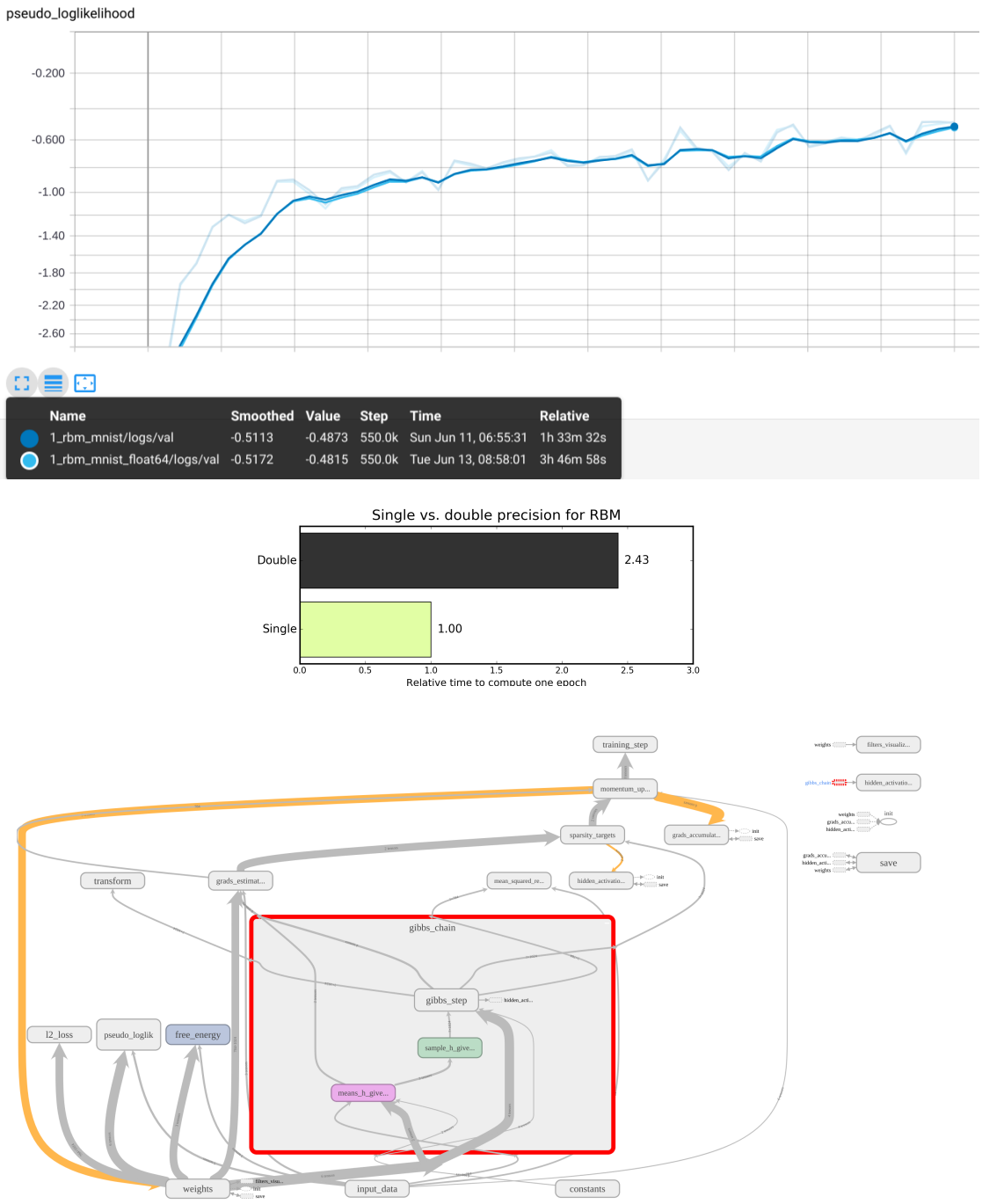


Figure 21: *Top*: When single precision is used, the difference in performance is negligible compared to the double precision, training however is almost 2.5 times faster (pseudo log-lik is slightly more noisy at the beginning of training though). *Bottom*: high-level computational graph for RBM.

Deep Boltzmann Machines [24, 20, 7]

More efficient learning algorithm for general binary-binary BMs

PCD-k

To recap:

$$\frac{1}{N}\sum_{n=1}^{N}\nabla_{\mathbf{W}}\log p(\mathbf{x}_n;\boldsymbol{\psi}) = \mathbb{E}_{\mathbf{v},\mathbf{h}\sim P_{\text{data}}(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}\left[\mathbf{v}\mathbf{h}^T\right] - \mathbb{E}_{\mathbf{v},\mathbf{h}\sim P_{\text{model}}(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}\left[\mathbf{v}\mathbf{h}^T\right] \tag{54}$$

and similar formulae for log-likelihood gradients w.r.t. $\mathbf{L}, \mathbf{J}, \mathbf{b}, \mathbf{c}$, see (17).

Now, instead of CD-k we use PCD-k (i.e. we keep one Markov chain without restarting its state between the updates), which falls into class of so-called *Stochastic approximation procedures* (SAP) to estimate **model's** expectations.

Let $\boldsymbol{\psi}^t$ and $\mathbf{x}^t$ be the current model parameters and the state. Then $\mathbf{x}^t$ and $\boldsymbol{\psi}^t$ are updated sequentially as follows:

- given $\mathbf{x}^t$, a new state $\mathbf{x}^{t+1}$ is sampled from a transition operator $T_{\boldsymbol{\psi}^t}(\mathbf{x}^{t+1} \leftarrow \mathbf{x}^t)$ that leaves $p(\cdot,\cdot;\boldsymbol{\psi}^t)$ invariant (which in our case is performing Gibbs sampling using equations (11) and (12) for k steps);
- a new parameter $\boldsymbol{\psi}^{t+1}$ is then obtained by replacing the intractable model's expectation by the point estimate $\mathbf{x}^t$ (see also subsubsection 2.2.2 for whether to sample or use probabilities/means instead of sampling, and for which type of states)

In practice, we typically maintain a set of P "persistent" sample "particles" $X^t = \{\mathbf{x}_1^t \dots \mathbf{x}_P^t\}$ and use average over those particles.

The intuition behind why this procedure works is the following: as the learning rate becomes sufficiently small compared with the mixing rate of the Markov chain, this "persistent" chain will always stay very close to the stationary distribution even if it is only run for a few MCMC updates per parameter update.

Provided $\|\boldsymbol{\psi}^t\|$ is bounded and Markov chain, governed by a transition kernel $T_{\boldsymbol{\psi}^t}$ is ergodic (which is typically true in practice), and sequence of learning rates α_t satisfies $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$, this stochastic approximation procedure procedure is almost surely convergent to an asymptotically stable point.
Note that in practice, α_t is not approached to zero, but rather to some small but positive constant ε (e.g. $10^{-6}, 10^{-5}$).

Variational learning

Another approach is used to approximate **data**-dependent expectations. We approximate true posterior over latent variables $p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi})$ (which is intractable in general BM, tractable in RBM, but

will be again intractable in DBM) by approximate posterior $q(\mathbf{h};\boldsymbol{\mu})$ and the variational parameters $\boldsymbol{\mu}$ are updated to follow the gradient of a *lower bound on the log-likelihood*:

In general:

$$\log p(\mathbf{v};\boldsymbol{\psi}) = \int q(\mathbf{h};\boldsymbol{\mu}) \log p(\mathbf{v};\boldsymbol{\psi}) \mathrm{d}\mathbf{h} = \int q(\mathbf{h};\boldsymbol{\mu}) \log \frac{p(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}{p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi})} \mathrm{d}\mathbf{h} =$$
$$= \int q(\mathbf{h};\boldsymbol{\mu}) \log \left(\frac{p(\mathbf{v},\mathbf{h};\boldsymbol{\psi})}{q(\mathbf{h};\boldsymbol{\mu})} \cdot \frac{q(\mathbf{h};\boldsymbol{\mu})}{p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi})} \right) \mathrm{d}\mathbf{h} =$$
$$= \int q(\mathbf{h};\boldsymbol{\mu}) \log p(\mathbf{v},\mathbf{h};\boldsymbol{\psi}) \mathrm{d}\mathbf{h} \underbrace{- \int q(\mathbf{h};\boldsymbol{\mu}) \log q(\mathbf{h};\boldsymbol{\mu}) \mathrm{d}\mathbf{h}}_{\mathcal{H}(q)} + \underbrace{\int q(\mathbf{h};\boldsymbol{\mu}) \log \frac{q(\mathbf{h};\boldsymbol{\mu})}{p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi})} \mathrm{d}\mathbf{h}}_{D_{\mathrm{KL}}(q(\mathbf{h};\boldsymbol{\mu}) \parallel p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi})) \geq 0} \geq$$
$$\geq \int q(\mathbf{h};\boldsymbol{\mu}) \log p(\mathbf{v},\mathbf{h};\boldsymbol{\psi}) \mathrm{d}\mathbf{h} + \mathcal{H}(q) =: \mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{\mu};\boldsymbol{\psi})$$

For Boltzmann Machine:

$$\bullet\ [1] = \sum_{\mathbf{h}} q(\mathbf{h};\boldsymbol{\mu}) \left[-E(\mathbf{v},\mathbf{h};\boldsymbol{\psi}) = -\log Z(\boldsymbol{\psi}) \right] - \underbrace{\log Z(\boldsymbol{\psi})}_{=\text{ const w.r.t.}\boldsymbol{\mu}} \cdot \underbrace{\sum_{\mathbf{h}} q(\mathbf{h};\boldsymbol{\mu})}_{=1} +$$
$$+ \sum_{\mathbf{h}} q(\mathbf{h};\boldsymbol{\mu}) \left[\sum_{j<k} L_{jk} v_j v_k + \sum_{l<m} J_{lm} h_l h_m + \sum_{j,l} W_{jl} v_j h_l + \sum_j b_j v_j + \sum_l c_l h_l \right] =$$
$$= \sum_{l<m} J_{lm} \mathbb{E}_{\mathbf{h}\sim q(\mathbf{h};\boldsymbol{\mu})}[h_l h_m] + \sum_{j,l} W_{jl} v_j \mathbb{E}_{\mathbf{h}\sim q(\mathbf{h};\boldsymbol{\mu})}[h_l] + \sum_l c_l \mathbb{E}_{\mathbf{h}\sim q(\mathbf{h};\boldsymbol{\mu})}[h_l] + \text{const}$$

For Boltzmann Machine and fully-factorizable $q(\mathbf{h};\boldsymbol{\mu}) = \prod_l q(h_l;\mu_l), q(h_l = 1;\mu_l) = \mu_l$ (*mean-field* approach):

$$\bullet\ [1] = \sum_{l<m} J_{lm} \mu_l \mu_m + \sum_{j,l} W_{jl} v_j \mu_l + \sum_l c_l \mu_l + \text{const}$$
$$\bullet\ [2] = \mathcal{H}(q) = -\sum_{\mathbf{h}} q(\mathbf{h};\boldsymbol{\mu}) \log q(\mathbf{h};\boldsymbol{\mu}) = -\sum_{\mathbf{h}} q(\mathbf{h};\boldsymbol{\mu}) \sum_j \log q(h_j;\mu_j) =$$
$$= -\sum_j \sum_{h_j \in \{0,1\}} q(h_j;\mu_j) \log q(h_j;\mu_j) \underbrace{\sum_{\mathbf{h}_{-j}} q(\mathbf{h}_{-j};\boldsymbol{\mu}_{-j})}_{=1} = -\sum_j \mu_j \log \mu_j + (1-\mu_j) \log(1-\mu_j)$$

$$\boxed{\mathcal{L}_{\text{ELBO}}(\boldsymbol{\mu};\boldsymbol{\psi}) = \sum_{l<m} J_{lm}\mu_l\mu_m + \sum_{j,l} W_{jl}v_j\mu_l + \sum_l c_l\mu_l - \sum_j \mu_j \log \mu_j + (1-\mu_j)\log(1-\mu_j) + \text{C}} \tag{55}$$

Let us maximize (55) for $\boldsymbol{\mu}$ for fixed $\boldsymbol{\psi}$:

$$0 \doteq \frac{\partial}{\partial \mu_i}\mathcal{L} = \underbrace{[l=i]\sum_{m>i} J_{im}\mu_m + [m=i]\sum_{l<i} J_{li}\mu_l}_{=[J_{ij}=J_{ji},J_{ii}=0]\sum_l J_{il}\mu_l} + \sum_j W_{ji}v_j + c_i - \log\mu_i - 1 + \log(1-\mu_i) + 1$$

$$\Leftrightarrow$$

$$\text{sigm}^{-1}(\mu_i) = \log\frac{\mu_i}{1-\mu_i} = \sum_{l<i} J_{li}\mu_l + \sum_j W_{ji}v_j + c_i$$

$$\boxed{\mu_i \leftarrow \text{sigm}\left(\sum_{l<i} J_{li}\mu_l + \sum_j W_{ji}v_j + c_i\right)} \tag{56}$$

Note that this is exactly the formula for (12) for computing $p(h_j = 1|\mathbf{v}, \mathbf{h}_{-i})$ in BM! So, updates of variational parameters can be computed using Gibbs sampler. This is not a coincidence, but the same holds if replace types of units, use RBM or even DBM (see below)!

Note, that this variational approach cannot be used to approximate model-expectations because of minus sign in formulae (54),(17). This would cause variational learning to change the parameters so as to *maximize* $D_{\text{KL}}(q(\mathbf{h};\boldsymbol{\mu}) \parallel p(\mathbf{h}|\mathbf{v};\boldsymbol{\psi}))$.

The naive mean-field approach was chosen because:

- ✓ its convergence is usually fast;
- ✓ it is unimodal.

Note that in general, we don't have to provide a parametric form of the approximating distribution beyond enforcing the independence assumptions. The variational approximation procedure is generally able to recover the functional form of the approximate distribution [7].

Deep Boltzmann Machine

Again assume unless specifically mentioned that DBM contains all binary units.

High-level overview

• DBM is a deep generative model, that consists of a layer of visible units and a series of layers of hidden units. In comparison to another deep generative model, DBN (which is hybrid, and

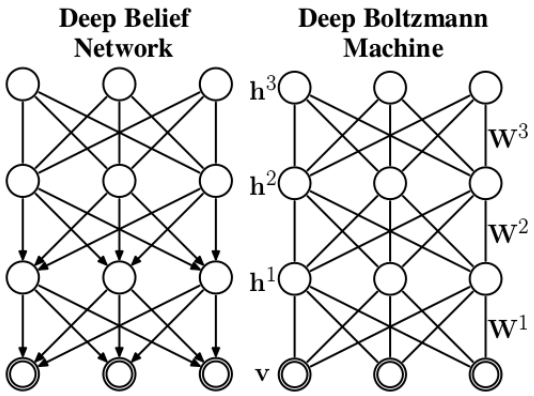


Figure 22: A three-layer Deep Belief Network and a three-layer Deep Boltzmann Machine.

has directed layers and one undirected), DBM is entirely undirected model, see Fig. 22. DBN is trained using greedy, layer by layer training of corresponding RBMs (one bottom-top pass). At the same time, All parameters in DBM are learned **jointly**, which greatly facilitates learning better generative models. Even though both models have a potential of learning series of internal representations that become increasingly complex, DBM's approximate bottom-top and top-bottom inference better propagate uncertainty $\Rightarrow$ deal more robustly with ambiguous inputs, than DBN.
• Formally, suppose number of (hidden) layers $L = 3$.

$$\mathbf{v} \in \mathbb{R}^D, \mathbf{h} = \{\mathbf{h^{(1)}}, \mathbf{h^{(2)}}, \mathbf{h^{(3)}}\}, \mathbf{h^{(s)}} \in \mathbb{R}^{H_s}, s \in \{1, 2, 3\};$$

Energy function:

$$E(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi}) = -\mathbf{v}^T \mathbf{W^{(1)}} \mathbf{h^{(1)}} - \mathbf{h^{(1)}}^T \mathbf{W^{(2)}} \mathbf{h^{(2)}} - \mathbf{h^{(2)}}^T \mathbf{W^{(3)}} \mathbf{h^{(3)}} - \mathbf{b} \cdot \mathbf{v} - \mathbf{c^{(1)}} \cdot \mathbf{h^{(1)}} - \mathbf{c^{(2)}} \cdot \mathbf{h^{(2)}} - \mathbf{c^{(3)}} \cdot \mathbf{h^{(3)}}, \tag{57}$$

where $\boldsymbol{\psi} = \{\mathbf{W^{(1)}}, \mathbf{W^{(2)}}, \mathbf{W^{(3)}}, \mathbf{b}, \mathbf{c^{(1)}}, \mathbf{c^{(2)}}, \mathbf{c^{(3)}}\}$. Probability that the model assign to a configuration $(\mathbf{v}, \mathbf{h})$:

$$p(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi}) \ \propto \ \exp(-E(\mathbf{v}, \mathbf{h}; \boldsymbol{\psi})) \tag{58}$$

• Now observe that conncections between units in the DBM are restricted in such a way, that unit from a layer depends only on the units from the *neighboring* layers, and does not depend from other units in the same layer or in the layers beyond. This is a multi-layer generalization of RBM, and allows to compute probabilities of units on given the others efficiently. For instance,

$$p(h_j^{(1)} = 1 | \mathbf{v}, \mathbf{h^{(2)}}) = \text{sigm}\left(\sum_i W_{ij}^{(1)} v_i + \sum_i W_{jl}^{(2)} h_l^{(2)} + c_j^{(1)}\right) \tag{59}$$

Observe how this formula resembles formulae (21),(22). This is also easily generalizes to other layers and other types of layers:

To compute probability of unit being on given all the others, **add** linear combinations of states of units from neighboring layers + bias and apply **activation function of a unit** (e.g. sigmoid for binary, softmax for softmax/multinomial, affine for gaussian etc.).

Note, however, that the distribution over **all** hidden layers generally does not factorize because of interactions between layers. For instance, for $L = 2$, $p(\mathbf{h}^{(1)}, mbh^{(2)}|\mathbf{v}; \boldsymbol{\psi})$ does not factorize due to interaction weights $\mathbf{W}^{(2)}$ between $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$ which render those variables mutually dependent.
• Formulae for log-likelihood gradients are derived the same way as for RBM and have similar form. For instance:

$$\frac{1}{N}\sum_{n=1}^{N}\nabla_{\mathbf{W}^{(2)}}\log p(\mathbf{x}_n;\boldsymbol{\psi}) = \mathbb{E}_{\mathbf{h}^{(1)},\mathbf{h}^{(2)}\sim P_{\text{data}}(\mathbf{h}^{(1)},\mathbf{h}^{(2)};\boldsymbol{\psi})}\left[\mathbf{h}^{(1)}\mathbf{h}^{(2)^T}\right] - \mathbb{E}_{\mathbf{h}^{(1)},\mathbf{h}^{(2)}\sim P_{\text{model}}(\mathbf{h}^{(1)},\mathbf{h}^{(2)};\boldsymbol{\psi})}\left[\mathbf{h}^{(1)}\mathbf{h}^{(2)^T}\right] \tag{60}$$

• Finally, we apply new learning algortihms for BMs described in the previous subsection with fully-factorizable mean-field approach:

$$q(\mathbf{h};\boldsymbol{\mu}) = \prod_j\prod_l\prod_m q(h_j^{(1)};\mu_j^{(1)})\cdot q(h_l^{(2)};\mu_l^{(2)})\cdot q(h_m^{(3)};\mu_m^{(3)}) \tag{61}$$

Thanks to lack of intra-layer interaction makes it possible to use fixed point equations (just like for general BM algorithm) to actually optimize the variational lower bound and find the true optimal mean field expectations.
• Further we will use DBM with Gaussian visible units, Multinomial top-most layer hidden unist, and Bernoulli hidden units for intermediate layers. In this setting, again the learning algortihms remains the same, the difference is only in the way probabilities are computed, and samples are made.
• One unfortunate property of DBMs is that sampling from them is relatively difficult. DBNs only need to use MCMC sampling in their top pair of layers. The other layers are used only at the end of the sampling process, in one efficient ancestral sampling pass. To generate a sample from a DBM, it is necessary to use MCMC across all layers, with every layer of the model participating in every Markov chain transition.

Gibbs sampling in DBMs

• Similar to RBM, Gibbs sampling using equations (59) can be made in parallel thus allowing to perform block Gibbs sampling for each layer of units. In addition to that, as illustrated in Fig. 23, the DBM layers can be organized into a bipartite graph, with odd layers on one side and even layers on the other. This immediately implies that when we condition on the variables in the even layer, the variables in the odd layers become conditionally independent. In conjuction with block Gibbs sampling for each layer, this allow to perform a Gibbs sampling in the whole DBM in **only 2 iterations**, instead of $L + 1$, as one might naively think at first.
• Good news that in TF no additional work need to be done beyond implementing block Gibbs sampling for each layer. Each independent branch in the computational graph should be executed in parallel. • Note that Contrastive Divergence algorithm is slow for DBMs because they do not

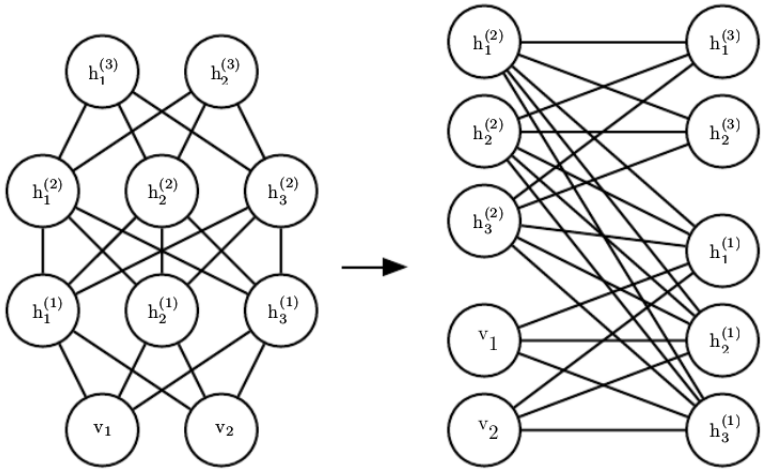


Figure 23: A deep Boltzmann machine, re-arranged to reveal its bipartite graph structure.

allow efficient sampling of the hidden states given the visible units – instead, CD would require burning in a Markov chain every time a new negative phase sample is needed.

Greedy layerwise pretraining of DBMs

DBM can be trained using the aforementioned learning algorithm from random initialization (typically the results are quite bad even on MNIST, see [9, 7]), but it works much better if weights are initialized sensibly. Greedy layerwise pretraining = learning procedure that consists of learning a stack of RBM's one layer at a time. After the stack is learned, the whole stack can be viewed as a single probabilistic model, called Deep Belief Net. Thus, pre-training for DBN is straightforward. In case of DBM though, a layer in the middle of the stack of RBMs is trained with only bottom-up input, but after the stack is combined to form DBM, the layer will have both bottom-up and top-down input. To account for this so-called *evidence double counting problem* [20, 7], Fig. 24, two modifications are required:

- bottom RBM should be trained using two "copies" of each visible unit and the weights tied to be equal between these two copies ($\cong$ simply double the total input to hidden layer during upward pass); similarly, top RBM should be trained with two copies of topmost layer. Training of all intermediate RBMs if there are any, should not be modified.
- the weights of all intermediate RBMs though, should be divided by 2 before inserting into DBM

Joint training of DBMs

Classic DBMs require greedy unsupervised pretraining, and to perform classification well, require a separate MLP-based classifier on top of the hidden features they extract. It is hard to track performance during training because we cannot evaluate properties of the full DBM while training

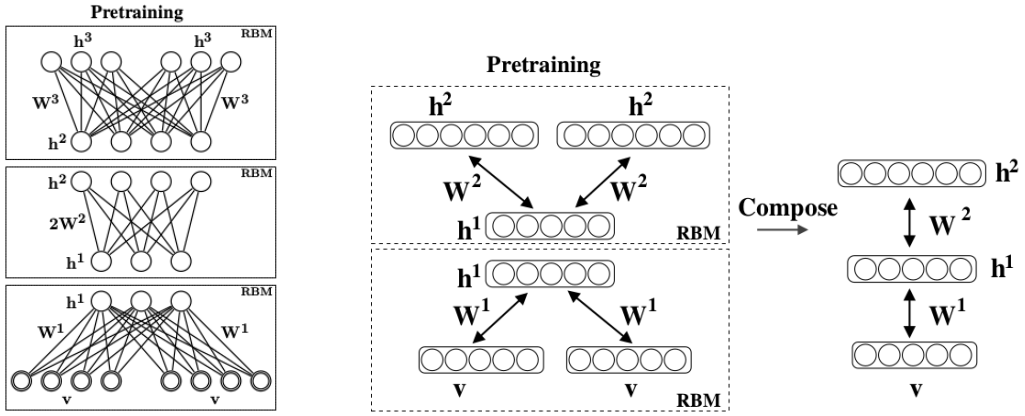


Figure 24: Pre-training consists of learning a stack of modified RBMs, that are then composed to create a DBM.

the first RBM. Software implementations of DBMs need to have many different components for CD training of individual RBMs, PCD training of the full DBM, and training based on back-propagation through the MLP. Finally, the MLP on top of the Boltzmann machine loses many of the advantages of the Boltzmann machine probabilistic model, such as being able to perform inference when some input values are missing.

There are two main ways to resolve the joint training problem of the deep Boltzmann machine: **multi-prediction DBMs**[8], which is currently beyond the scope of this project, and the **centering trick** [16], which reparametrizes the model in order to make the Hessian of the cost function better-conditioned at the beginning of the learning process. More specifically, if we consider energy function of generalized Boltzmann Machine (BM, RBM, DBM can all be represented by appropriate choice of $\mathbf{x}$ – states, $\mathbf{U}$ – weights, $\mathbf{a}$ – biases):

$$E(\mathbf{x}; \boldsymbol{\psi}) = -\mathbf{x}^T \mathbf{U} \mathbf{x} - \mathbf{a} \cdot \mathbf{x} \tag{62}$$

Then the idea of centering trick is simply to reparameterize this energy function as

$$E(\mathbf{x}; \boldsymbol{\psi}) = -(\mathbf{x} - \boldsymbol{\beta})^T \mathbf{U} (\mathbf{x} - \boldsymbol{\beta}) - \mathbf{a} \cdot (\mathbf{x} - \boldsymbol{\beta}) \tag{63}$$

Where new hyperparameter vector $\boldsymbol{\beta}$ is chosen to be $\mathbf{x} - \boldsymbol{\beta} \approx \mathbf{0}$ at the beginning of training. This does not change the set of probability distributions that the model can represent, but it does change the learning dynamics so much, that it is actually possible to train DBM from random initialization w/o pre-training and achieve sensible results. However, in [8] they say when DBM is trained using centering trick, they have never shown to have good classification performance, if this was the primary goal.

Annealed importance sampling [23, 20, 14, 28]

Let $p_A(\mathbf{x}) = \frac{p_A^*(\mathbf{x})}{\mathcal{Z}_A}$ be simple proposal distribution from which we can sample easily, and $p_B(\mathbf{x}) = \frac{p_B^*(\mathbf{x})}{\mathcal{Z}_B}$ our complex target distribution. We also have to make sure $p_B \ll p_A$, which is easy in our

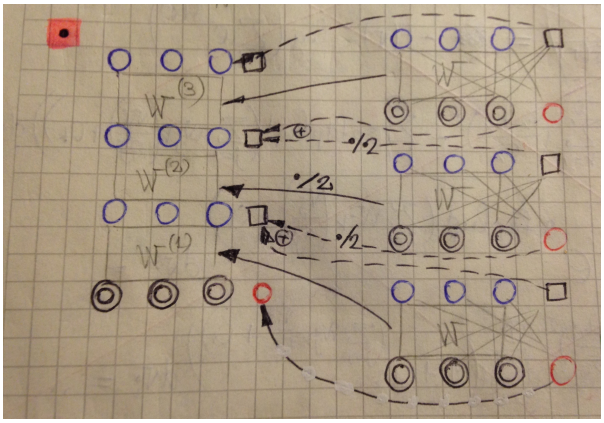


Figure 25: A more detailed scheme how to initialize 3-layer DBM from learned stack or RBMs, including biases. Black circles – visible units, blue – hidden units, red circel – visible bias, black square – hidden bias. Biases can be summed or averaged.

case, since we can always choose p_A to be uniform pmf, which dominates every other probability mass function on discrete units (of finite cardinality).

(Classical) Importance Sampling
The ratio of partition functions can be estimated as follows

$$\frac{\mathcal{Z}_B}{\mathcal{Z}_A} = \frac{p_B^*(\mathbf{x})}{p_A^*(\mathbf{x})} = \sum_{\mathbf{x}} \frac{p_B^*(\mathbf{x})}{p_A^*(\mathbf{x})} p_A(\mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim p_A}\left[\frac{p_B^*(\mathbf{x})}{p_A^*(\mathbf{x})}\right] \approx \frac{1}{N}\sum_{i=1}^{N} \frac{p_B^*(\mathbf{x}_i)}{p_A^*(\mathbf{x}_i)} \tag{64}$$

The problem is when p_A and p_B are very different, as in our case, this estimator is very poor: its variance is very large, possibly infinite.

Annealed Importance Sampling
To handle this issue, we define sequence of probability mass functions $(p_m)_{m=0:M}$ such that $p_0 = p_A$ and $p_M = p_B$, and for which we know unnormalized probabilities p_m^*, which typically are mixtures of target and proposal:

$$p_m(\mathbf{x}) = p_B(\mathbf{x})^{\beta_m} \cdot p_A(\mathbf{x})^{1-\beta_m}, \beta_m = \frac{m}{M} \tag{65}$$

Also, in order not to sample from p_s we also need sequence of transition operators $(T_i(\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i))_{i=1:M-1}$ each that leaves the corresponding p_i invariant. The importance weight can then be computed as

$$\omega_{\text{AIS}} \leftarrow \prod_{m=1}^{M} \frac{p_m^*(\mathbf{x}_m)}{p_{m-1}^*(\mathbf{x}_m)} \tag{66}$$

where $\mathbf{x}_1 \sim p_0 = p_A$; $\mathbf{x}_2 \sim T_1(\mathbf{x}_2 \leftarrow \mathbf{x}_1)$; $\ldots$ $\mathbf{x}_M \sim T_{M-1}(\mathbf{x}_M \leftarrow \mathbf{x}_{M-1})$. The ratio of partition functions can then be estimated as average over many AIS runs:

$$\frac{\mathcal{Z}_B}{\mathcal{Z}_A} = \frac{\mathcal{Z}_M}{\mathcal{Z}_0} \approx \frac{1}{L}\sum_{l=1}^{L} \omega_{\text{AIS}}^{(l)} \tag{67}$$

Notice also that we don't need to compute partition functions of any of the intermediate distributions.
Note: to avoid numerical problems and overflow errors (partition functions are very large numbers even for very moderate sized BM), all computations are performed in log-domain, as usual.

Annealed Importance Sampling for 2-layer Bernoulli BM

It turns out that we can reduce state space of AIS to only hidden units in first layer $\mathbf{x} = \{\mathbf{h}^{(1)}\}$ by explicitly summing out visible and top-most layer hidden units:

$$\begin{aligned}
\log p^*\left(\mathbf{h}^{(1)}\right) &= \log \sum_{\mathbf{v},\mathbf{h^{(2)}}} p^*\left(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}\right) = \\
&= \log \sum_{\mathbf{v},\mathbf{h^{(2)}}} \exp\left(\mathbf{v}^T\mathbf{W}^{(1)}\mathbf{h}^{(1)} + \mathbf{h}^{(1)^T}\mathbf{W}^{(2)}\mathbf{h}^{(2)} + \mathbf{b}\cdot\mathbf{v} + \mathbf{c}^{(1)}\cdot\mathbf{h}^{(1)} + \mathbf{c}^{(2)}\cdot\mathbf{h}^{(2)}\right) \\
&= \mathbf{c}^{(1)}\cdot\mathbf{h}^{(1)} + \log\left[\sum_{\mathbf{v}} \exp\left(\mathbf{v}^T\mathbf{W}^{(1)}\mathbf{h}^{(1)} + \mathbf{b}\cdot\mathbf{v}\right) \sum_{\mathbf{h}^{(2)}} \exp\left(\mathbf{h}^{(1)^T}\mathbf{W}^{(2)}\mathbf{h}^{(2)} + \mathbf{c}^{(2)}\cdot\mathbf{h}^{(2)}\right)\right] \\
&= \mathbf{c}^{(1)}\cdot\mathbf{h}^{(1)} + \sum_i^V \text{softplus}\left(\sum_j^{H_1} W_{ij}^{(1)} h_j^{(1)} + b_i\right) + \sum_k^{H_2} \text{softplus}\left(\sum_j^{H_1} W_{jk}^{(2)} h_k^{(2)} + c_k^{(2)}\right)
\end{aligned}$$

$$\boxed{\log p^*\left(\mathbf{h}^{(1)}\right) = \mathbf{c}^{(1)}\cdot\mathbf{h}^{(1)} + \sum_i^V \text{softplus}\left(\sum_j^{H_1} W_{ij}^{(1)} h_j^{(1)} + b_i\right) + \sum_k^{H_2} \text{softplus}\left(\sum_j^{H_1} W_{jk}^{(2)} h_k^{(2)} + c_k^{(2)}\right)} \tag{68}$$

From this we can easily derive equation for $\log p_m^*$ by simply scaling all weights by β_m:

$$\begin{aligned}
\log p_m^*\left(\mathbf{h}^{(1)}\right) = \beta_m\mathbf{c}^{(1)}\cdot\mathbf{h}^{(1)} &+ \sum_i^V \text{softplus}\left(\beta_m \cdot \left(\sum_j^{H_1} W_{ij}^{(1)} h_j^{(1)} + b_i\right)\right) + \\
&+ \sum_k^{H_2} \text{softplus}\left(\beta_m \cdot \left(\sum_j^{H_1} W_{jk}^{(2)} h_k^{(2)} + c_k^{(2)}\right)\right)
\end{aligned} \tag{69}$$

When $\beta_m = 1$ we obtain target distribution, when $\beta_m = 0$ we obtain uniform distribution:

$$\log p_0^*\left(\mathbf{h}^{(1)}\right) \equiv 0 + \sum_i^V \text{softplus}(0) + \sum_k^{H_2} \text{softplus}(0) = (V + H_2)\log 2 \tag{70}$$

thus $\log \mathcal{Z}_0 = (V + H_1 + H_2)\log 2$.

Thus we gradually increase "inverse temperature" β from 0 to 1 and can estimate partition function using procedure described above. Staring from randomly initialized $\mathbf{h}^{(1)}$, we apply sequence of

transition operators T_i which are simply alternating Gibbs sampler with weights scaled by β_i.

We can do the same for different types of units and larger number of layers. In the latter case we can again analytically sum out visible and top-most hidden units.

Variational lower-bound

Having estimate of partition function $\widehat{\mathcal{Z}}$, we can estimate variational lower-bound on test vector $\mathbf{v}^*$ as follows

$$\begin{aligned}\log p(\mathbf{v}^*;\boldsymbol{\psi}) \geq & -\sum_{\mathbf{h}} q(\mathbf{h};\boldsymbol{\mu})E(\mathbf{v}^*,\mathbf{h};\boldsymbol{\psi}) + \mathcal{H}(\boldsymbol{\mu}) - \log \mathcal{Z}(\boldsymbol{\psi}) \\ &= \mathbf{v}^{*^T}\mathbf{W}^{(1)}\boldsymbol{\mu}^{(1)}_{\mathbf{v}^*} + \boldsymbol{\mu}^{(1)^T}_{\mathbf{v}^*}\mathbf{W}^{(2)}\boldsymbol{\mu}^{(2)}_{\mathbf{v}^*} + \mathbf{b}\cdot\mathbf{v}^* + \mathbf{c}^{(1)}\cdot\boldsymbol{\mu}^{(1)}_{\mathbf{v}^*} + \mathbf{c}^{(2)}\cdot\boldsymbol{\mu}^{(2)}_{\mathbf{v}^*} + \mathcal{H}(\boldsymbol{\mu}_{\mathbf{v}^*}) - \log \mathcal{Z}(\boldsymbol{\psi}) \\ &\approx \mathbf{v}^{*^T}\mathbf{W}^{(1)}\boldsymbol{\mu}^{(1)}_{\mathbf{v}^*} + \boldsymbol{\mu}^{(1)^T}_{\mathbf{v}^*}\mathbf{W}^{(2)}\boldsymbol{\mu}^{(2)}_{\mathbf{v}^*} + \mathbf{b}\cdot\mathbf{v}^* + \mathbf{c}^{(1)}\cdot\boldsymbol{\mu}^{(1)}_{\mathbf{v}^*} + \mathbf{c}^{(2)}\cdot\boldsymbol{\mu}^{(2)}_{\mathbf{v}^*} + \mathcal{H}(\boldsymbol{\mu}_{\mathbf{v}^*}) - \log \widehat{\mathcal{Z}}\end{aligned} \tag{71}$$

where $\boldsymbol{\mu}_{\mathbf{v}^*}$ are variational parameters obtained by running fixed-point equations using Gibbs sampler unitl convergence with visible units clamped to $\mathbf{v}^*$.

One can also estimate true log-probability using AIS by clamping visible units to test example (estimating log-probability for one test example is computationally equivalent to estimating a partition function).

Additional facts

• In [7] they say that obtaining sota results with DBM requires an additional partial mean field in negative phase, more details in [8].
• The inference can further be accelerated using separate *recognition model*, see [21] for details.
• DBMs were developed after DBNs. Compared to DBNs, the posterior distribution $p(\mathbf{h}|\mathbf{v})$ is simpler for DBMs. Somewhat counterintuitively, the simplicity of this posterior distribution allows richer approximations of the posterior [7]
• The use of proper mean field allows the approximate inference procedure for DBMs to capture the influence of top-down feedback interactions. This makes DBMs interesting from the point of view of neuroscience, because the human brain is known to use many top-down feedback connections[7].
• In [10] they observe that energy function $E(\mathbf{v},\mathbf{h};\boldsymbol{\psi})$ inevitably induces some prior $p(\mathbf{h};\boldsymbol{\psi})$ that is not motivated by the structure of any kind of data. The role of deeper layers in DBM is simply to provide a better prior on the first layer hidden units.

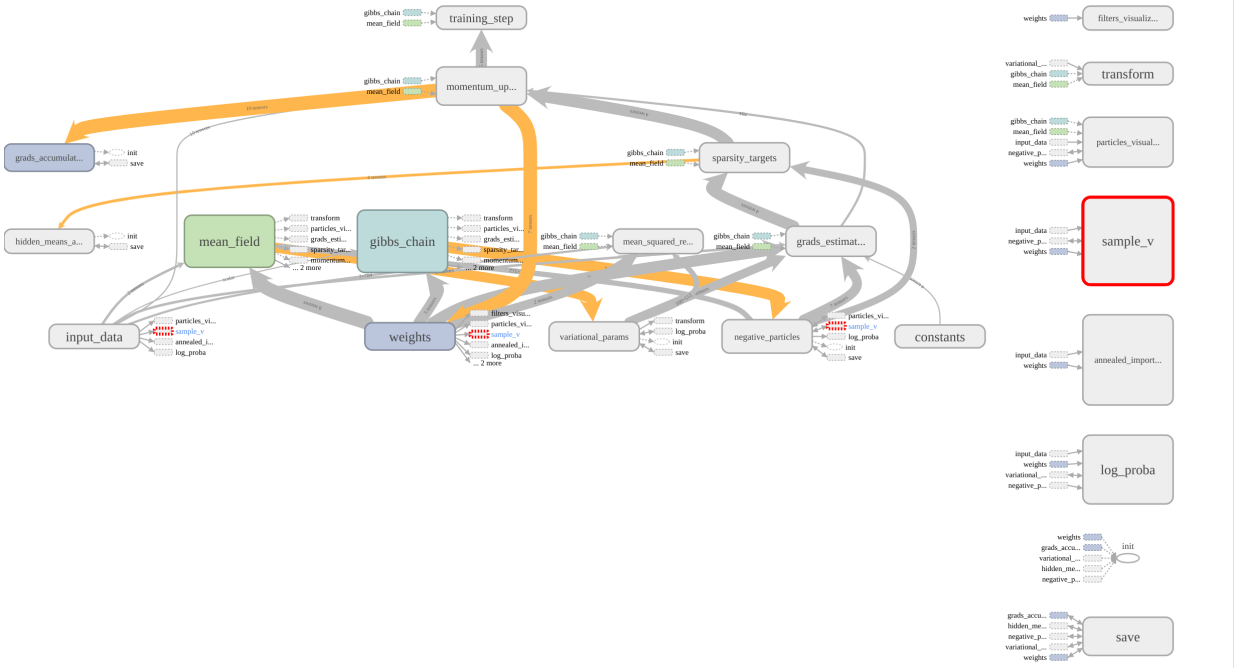


Figure 26: High-level computational graph for DBM model.

DBM experiments

Experiments on MNIST

Before sparsity targets, AIS

Architecture and hyperparameters initially were similar to those of [2], however, I observe that learning rate should be smaller (eventually I used exactly 1 order of magnitude smaller starting learning rate).

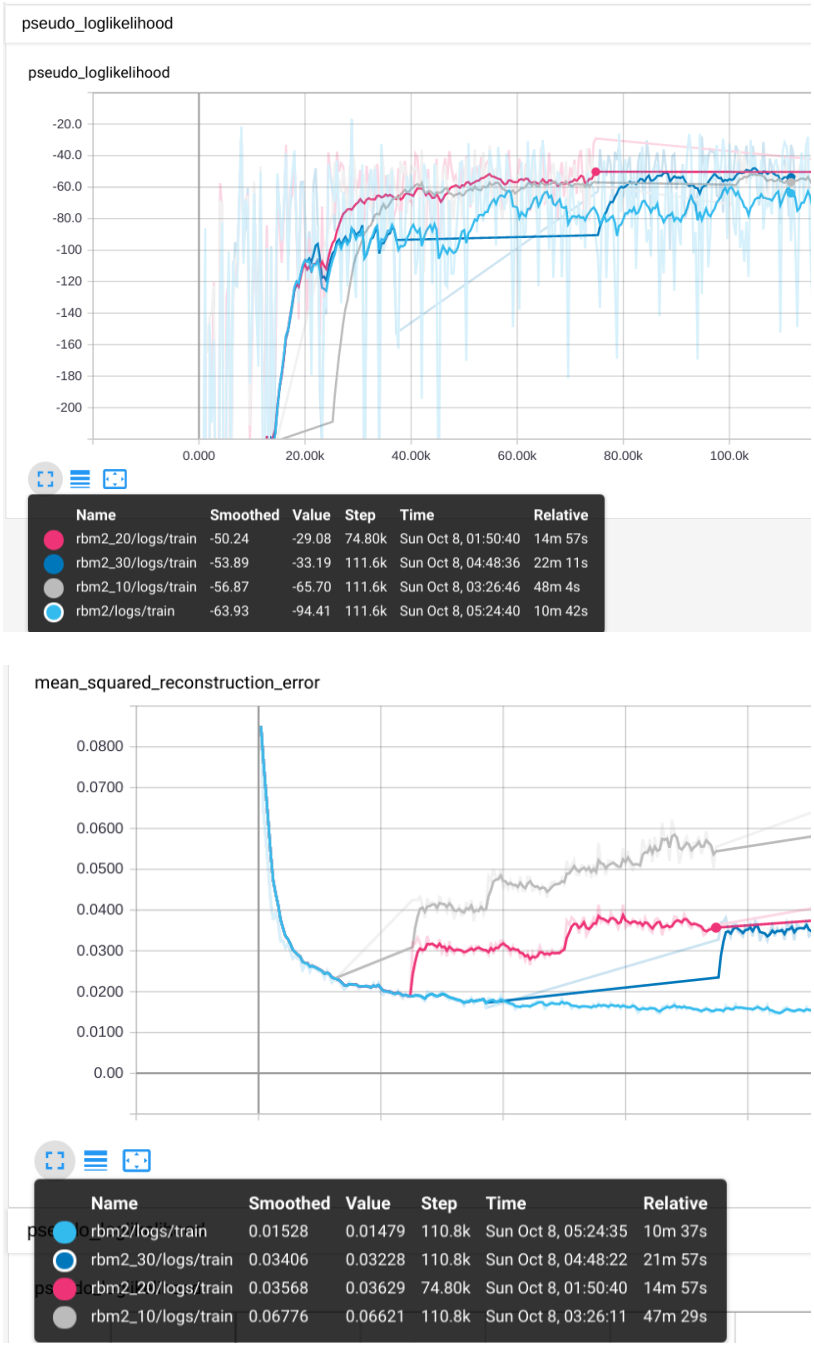


Figure 27: PLL and reconstruction error for RBM #2 trained on extracted features $\mathbf{q}_i = p(\mathbf{h}|\mathbf{v} = \mathbf{x}_i; \boldsymbol{\psi})$ using RBM #1 trained on MNIST. `rbm_n` means every n epochs number of Gibbs sampling increased by 1, and learning rate $\alpha_t \leftarrow \frac{\alpha_0}{n}$. This approach [2, 22] has quite noticeably higher PLL compared to CD-k with constant k (both 1 or higher).

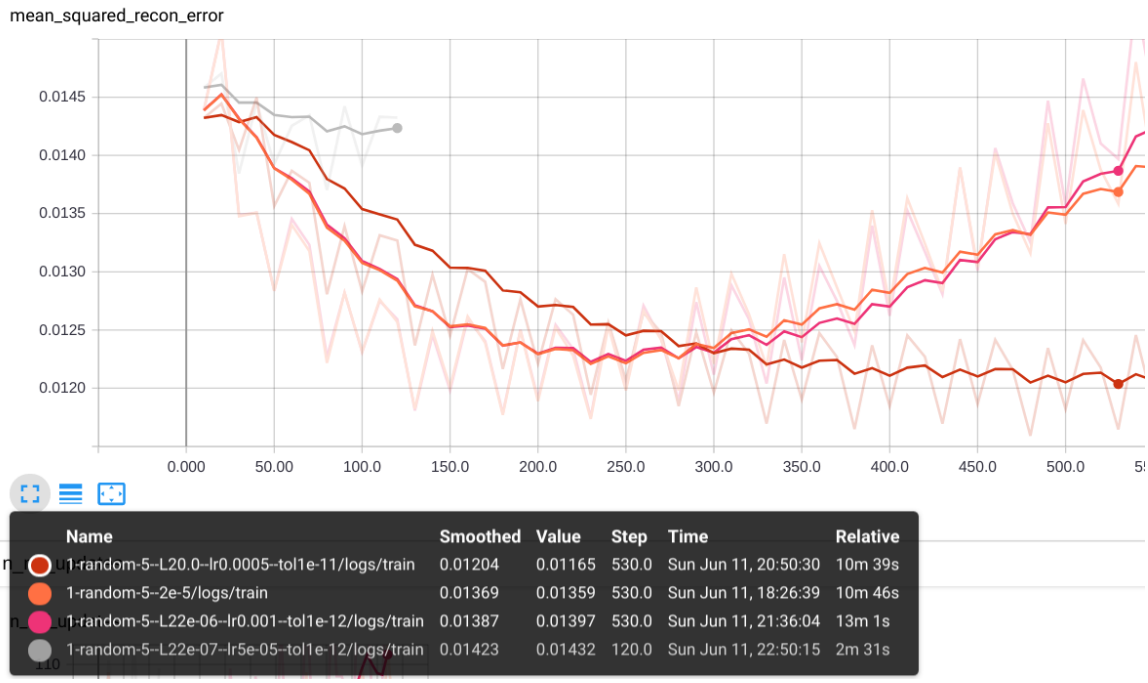
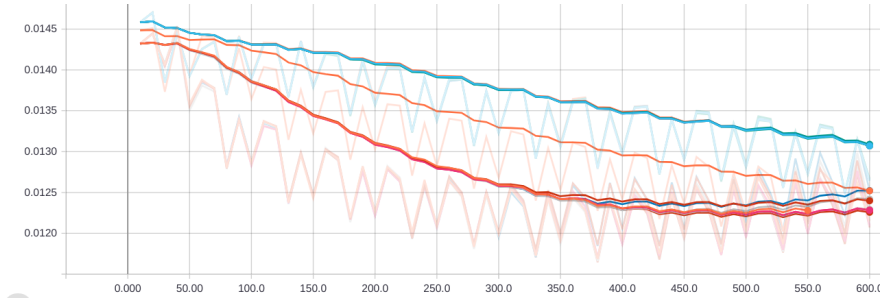
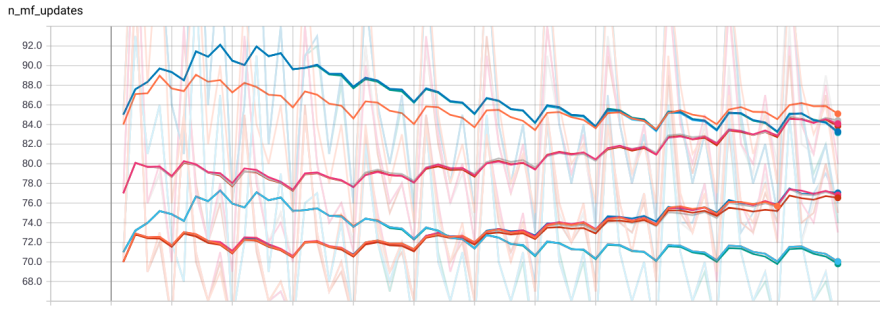


Figure 28: This illustrates how the use of L2 weight decay harms joint training of DBM in terms of MSRE (no matter how small λ is). The intuition here is when weights are not trained well enough for them to be useful for modeling the data, the weight decay term will drive them to become very small, and they will never have an opportunity to recover. In [10] and [8] the authors had the same issue and suggested to use *maxnorm* regularization instead. Trained on random subset of 5k images.



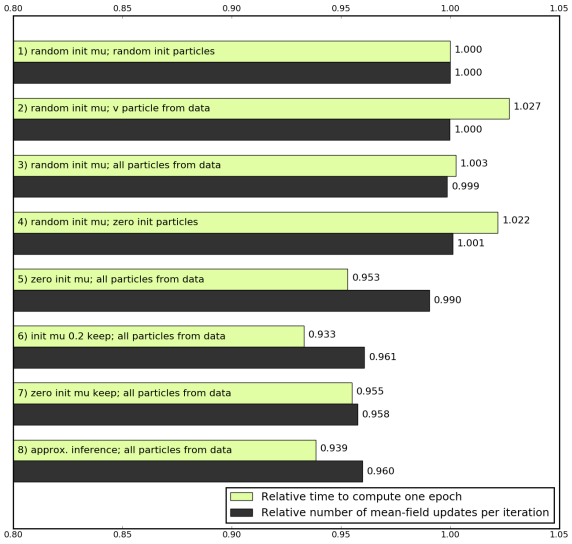
Name	Smoothed	Value	Step	Time	Relative
1-random-5--L22e-08--lr0.0005--tol1e-11/logs/train	0.01226	0.01207	600.0	Mon Jun 12, 00:12:30	12m 50s
1-random-5--2e-6--0.0005/logs/train	0.01228	0.01207	550.0	Sun Jun 11, 18:40:30	10m 37s
1-random-5--L22e-06--lr0.0005--tol1e-11/logs/train	0.01228	0.01209	600.0	Sun Jun 11, 21:04:23	12m 9s
1-random-5--L22e-06--lr0.0005--tol1e-10/logs/train	0.01228	0.01209	600.0	Sun Jun 11, 19:32:46	11m 17s
1-random-5--L22e-07--lr0.0005--tol1e-11/logs/train	0.01229	0.01215	600.0	Sun Jun 11, 21:17:34	12m 56s
1-random-5--L20.0--lr0.0005--tol1e-10/logs/train	0.01240	0.01223	600.0	Sun Jun 11, 19:21:15	11m 42s
1-random-5--L22e-07--lr0.0005--tol1e-10/logs/train	0.01242	0.01233	600.0	Sun Jun 11, 19:44:10	11m 11s
1-random-5--L22e-05--lr0.0002--tol1e-12/logs/train	0.01252	0.01215	600.0	Sun Jun 11, 23:04:06	13m 5s
1-random-5--L22e-08--lr0.0005--tol1e-10/logs/train	0.01252	0.01252	600.0	Sun Jun 11, 19:55:38	11m 14s
1-random-5--L22e-06--lr0.0001--tol1e-10/logs/train	0.01307	0.01266	600.0	Sun Jun 11, 20:17:37	10m 45s
1-random-5--L20.0--lr0.0001--tol1e-10/logs/train	0.01307	0.01264	600.0	Sun Jun 11, 20:06:38	10m 46s
1-random-5--L22e-08--lr0.0001--tol1e-10/logs/train	0.01307	0.01264	600.0	Sun Jun 11, 20:39:36	10m 46s
1-random-5--L22e-06--lr0.0001--tol1e-12/logs/train	0.01307	0.01265	600.0	Sun Jun 11, 22:21:05	13m 32s
1-random-5--L22e-07--lr0.0001--tol1e-10/logs/train	0.01308	0.01267	600.0	Sun Jun 11, 20:28:36	10m 46s
1-random-5--L22e-07--lr0.0001--tol1e-12/logs/train	0.01308	0.01264	600.0	Sun Jun 11, 22:34:08	12m 47s
1-random-5--L22e-06--lr5e-05--tol1e-12/logs/train	0.01309	0.01266	600.0	Sun Jun 11, 22:47:28	13m 4s

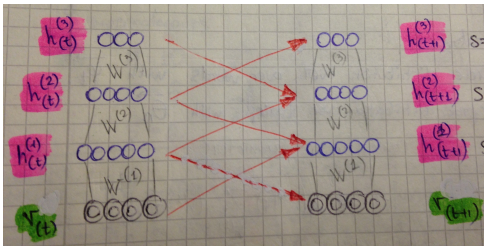


Name	Smoothed	Value	Step	Time	Relative
1-random-5--L22e-06--lr0.0001--tol1e-10/logs/train	69.80	63.00	600.0	Sun Jun 11, 20:17:37	10m 45s
1-random-5--L22e-07--lr0.0001--tol1e-10/logs/train	69.94	62.00	600.0	Sun Jun 11, 20:28:36	10m 46s
1-random-5--L22e-08--lr0.0001--tol1e-10/logs/train	70.04	63.00	600.0	Sun Jun 11, 20:39:36	10m 46s
1-random-5--L20.0--lr0.0001--tol1e-10/logs/train	70.04	63.00	600.0	Sun Jun 11, 20:06:38	10m 46s
1-random-5--2e-6--0.0005/logs/train	75.69	72.00	550.0	Sun Jun 11, 18:40:30	10m 37s
1-random-5--L20.0--lr0.0005--tol1e-10/logs/train	76.54	75.00	600.0	Sun Jun 11, 19:21:15	11m 42s
1-random-5--L22e-07--lr0.0005--tol1e-10/logs/train	76.80	74.00	600.0	Sun Jun 11, 19:44:10	11m 11s
1-random-5--L22e-06--lr0.0005--tol1e-10/logs/train	76.82	73.00	600.0	Sun Jun 11, 19:32:46	11m 17s
1-random-5--L22e-08--lr0.0005--tol1e-10/logs/train	77.04	76.00	600.0	Sun Jun 11, 19:55:38	11m 14s
1-random-5--L22e-07--lr0.0001--tol1e-12/logs/train	83.15	74.00	600.0	Sun Jun 11, 22:34:08	12m 47s
1-random-5--L22e-06--lr5e-05--tol1e-12/logs/train	83.22	74.00	600.0	Sun Jun 11, 22:47:28	13m 4s
1-random-5--L22e-06--lr0.0001--tol1e-12/logs/train	83.28	75.00	600.0	Sun Jun 11, 22:21:05	13m 32s
1-random-5--L22e-08--lr0.0005--tol1e-11/logs/train	83.85	78.00	600.0	Mon Jun 12, 00:12:30	12m 50s
1-random-5--L22e-06--lr0.0005--tol1e-11/logs/train	84.09	80.00	600.0	Sun Jun 11, 21:04:23	12m 9s

Figure 29: The effect of various choices of L2 weight decay coefficient, learning rate and desired mean field tolerance in terms of reconstruction error and number of mean-filed updates to achieve desired tolerance on random subset of 5k images.

Figure 30: *Top*: the effect of various choices of L2 weight decay coefficient, learning rate, maxnorm constraint and number of Gibbs steps on reconstruction error on random subset of 10k images from MNIST; 20 mean-filed updates, 100 particles, momentum $0.5 \rightarrow 0.9$. *Bottom*: the same, but on binarized version of MNIST, as suggested in [9].





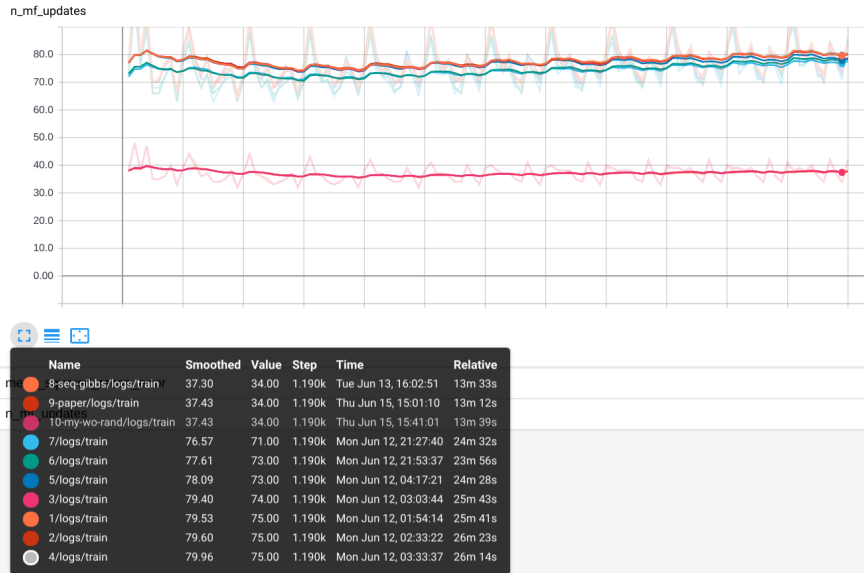


Figure 31: *Top left*: Various approaches on particles and variational parameters initialization to speedup mean-field convergence I've tried. Numbers of mean-filed updates are meant to achieve 10^{-11} tolerance. **In sum**: it is always better to initialize <u>all</u> particles (including hidden ones) from the training data (hidden particles on the first level, for instance, should be initialized from the extracted fratures $\mathbf{q}_i = p(\mathbf{h}|\mathbf{v} = \mathbf{x}_i; \boldsymbol{\psi})$ from the first pre-trained RBM. Not resetting variation parameters between different gibbs updates is also slightly faster than bottom-up approximate inference described in [20, 21]. It is thus a combination of these two best approaches (approx. inference + not resetting) is implemented in `DBM` class. Notwithstanding, the results (in terms of reconstruction error) were pretty robust to the exact number of mean-field updates: similar when used 10 or 25 mean-field updates. *Top right*: I also tried "parallel" version of Gibbs sampling. One mean-field update was faster ($\approx 1.1 - 1.2$) but the total number of them was nearly twice as many. *Bottom*: dynamics of mean field updates for various approaches. See also excel table. Experiments were tried on random subset 10k images.

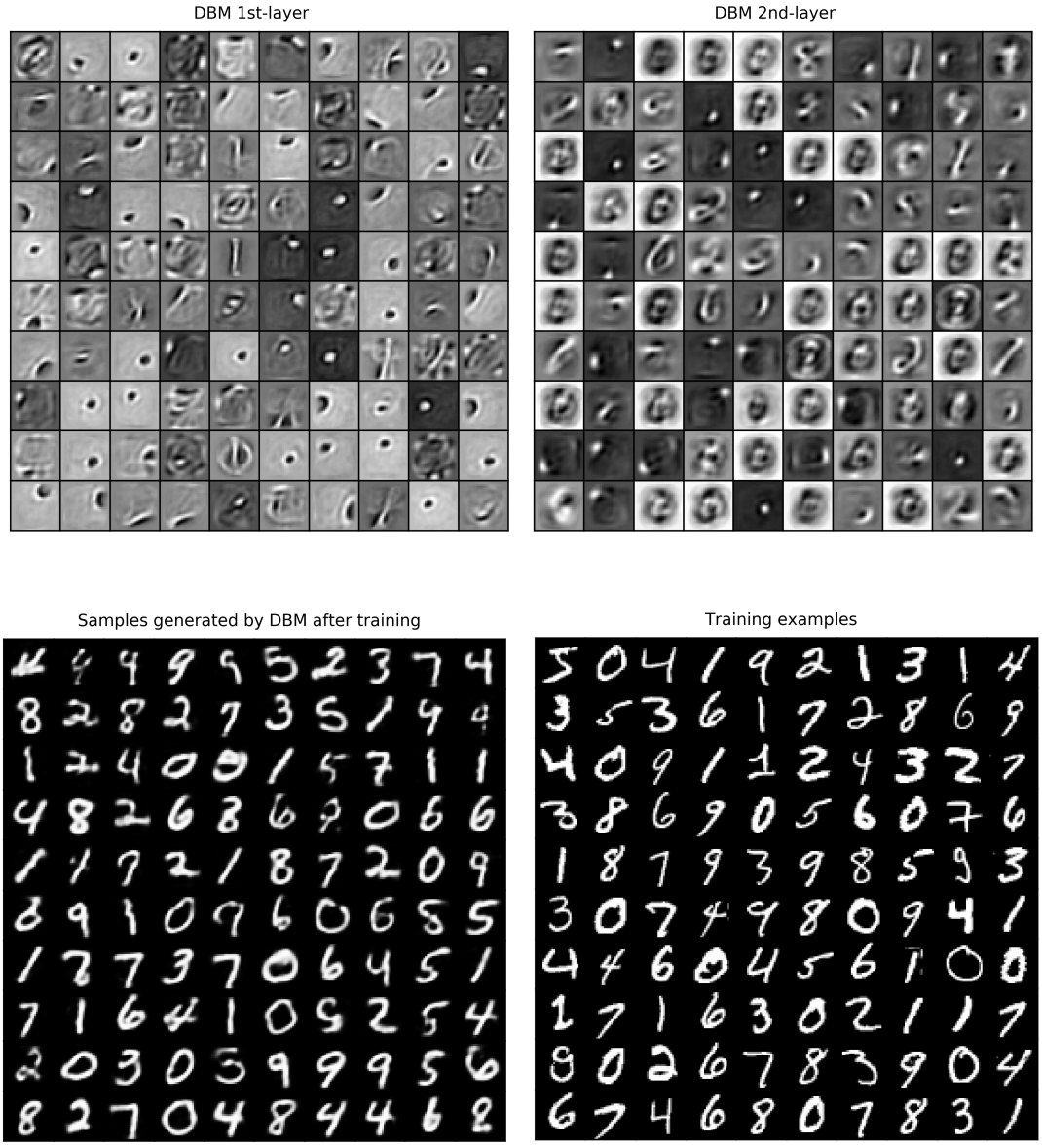


Figure 32: Best model trained on full MNIST in this phase in terms of the generated samples quality. 2nd layer filters are visualized using method of *weighted linear combinations* [4]. ***Hyperparams***: 100 particles, 25 mean-filed updates, 1 Gibbs step per iter, mean-field tolerance 10^{-7}, momentum $0.5 \rightarrow 0.9$, learning rate $5 \cdot 10^{-4} \rightarrow 10^{-5}$ by dividing by $(1.000015)^{600}$ each epoch, 200 epochs, batch-size 100, L2 $= 10^{-7}$, maxnorm $= 4$; sample all visible and hidden states.

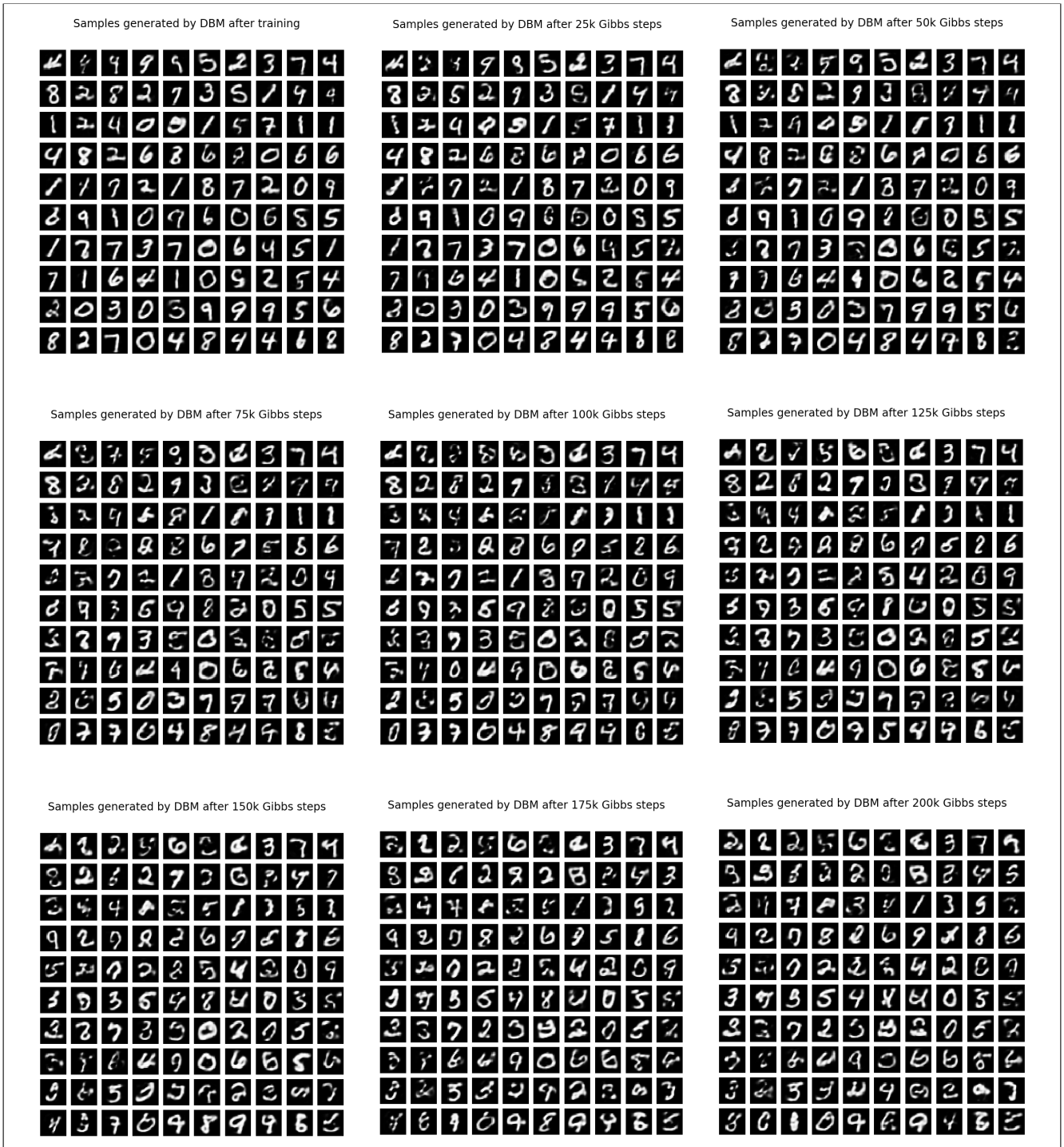


Figure 33: This illustrates how correlated generated samples are between one another because the approach used to generate them is MCMC-based one

.

After sparsity targets, AIS are implemented

After AIS is implemented, now it is possible to quantitatively evaluate DBM performance. To do this, I cross-validated 216 models with various hyperparameters on small held-out validation set $\rightarrow$ chose best values of hyperparameters based on ELBO estimation and quality of samples $\rightarrow$ repeated for 54 models with larger validation set $\rightarrow 4 \rightarrow 1$.
Interesting observation: AIS is (much) more accurate for better models and which are trained longer. Results:

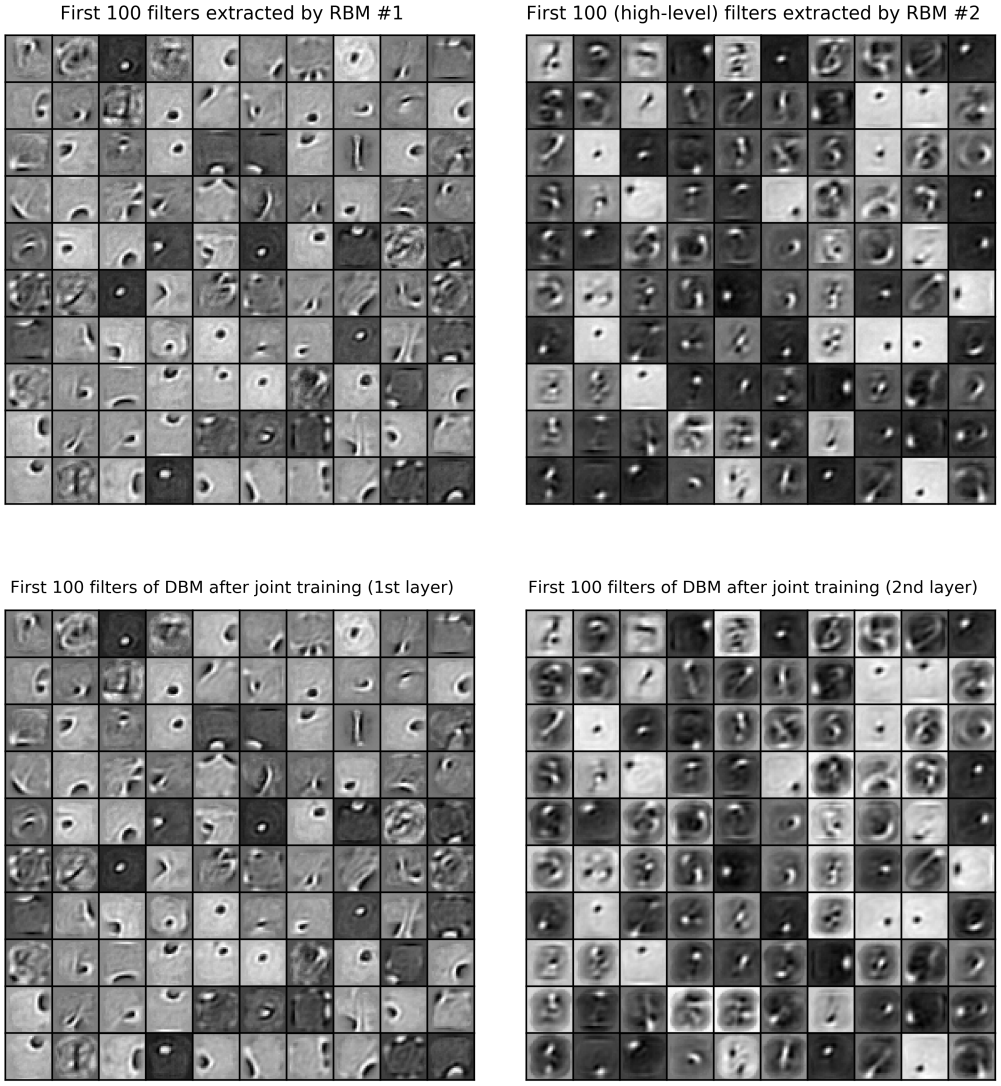
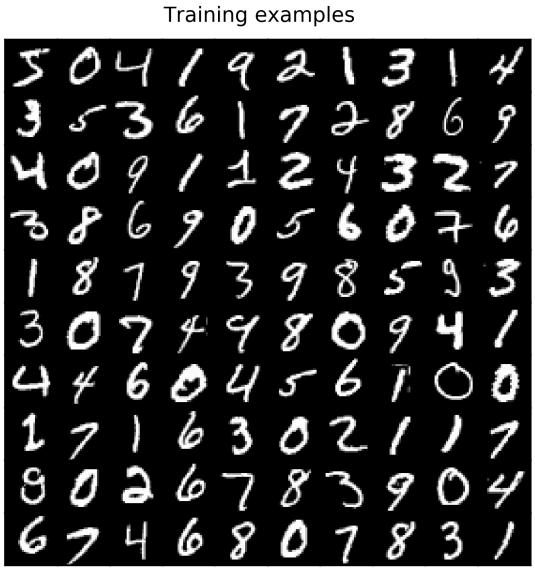
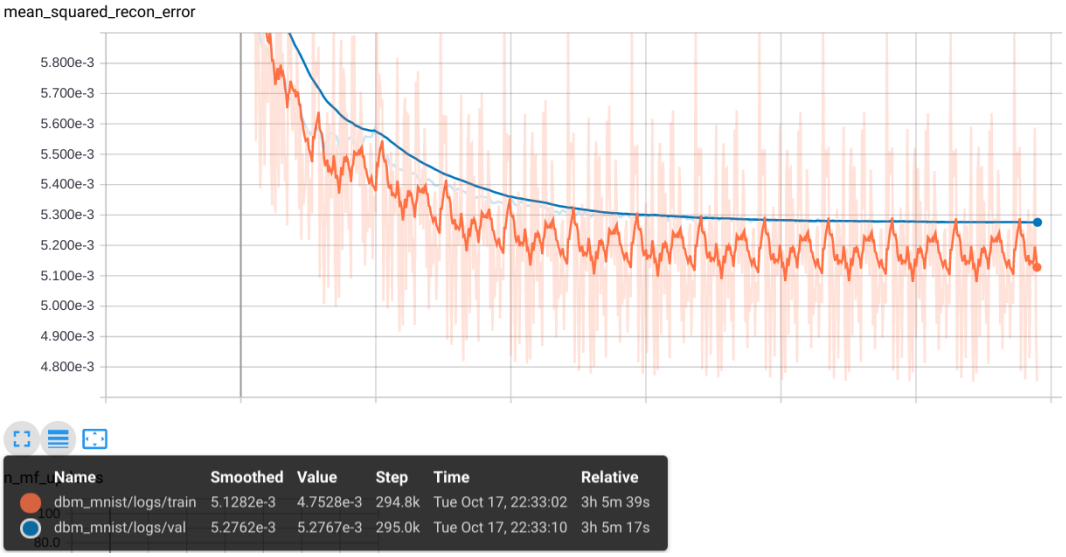


Figure 34: Weight filters

.





Name	Smoothed	Value	Step	Time	Relative
dbm_mnist/logs/train	5.1282e-3	4.7528e-3	294.8k	Tue Oct 17, 22:33:02	3h 5m 39s
dbm_mnist/logs/val	5.2762e-3	5.2767e-3	295.0k	Tue Oct 17, 22:33:10	3h 5m 17s

Figure 35: *Top*: random subset of training data and generated samples by DBM. *Bottom*: MSRE of DBM

.

Experiments on CIFAR-10

Before sparsity targets, AIS; naive training of Gaussian RBM

Architecture: 3072-5000-1000 Gaussian-Bernoulli-Multinomial DBM.

Preprocessing: Zeroing 1000 least significant variance directions as in [15].

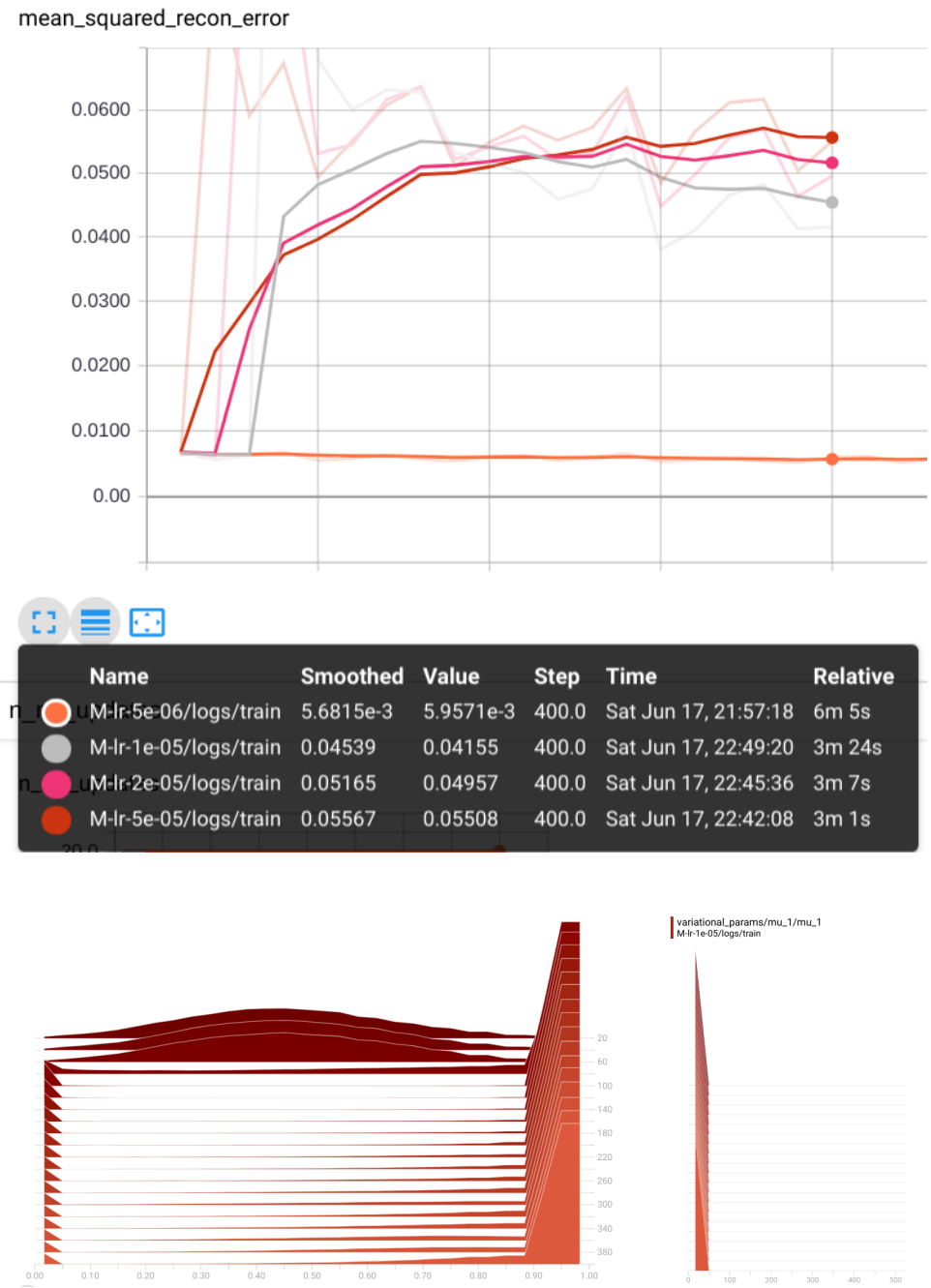


Figure 36: After pre-training during the joint training severe instabilities occured unless an extremely small learning rate was used. After inspection of distribution of variational parameters (in first and second hidden layers), the reason became clear: the total input to first layer hidden units was too large $\Rightarrow$ "overshoot" of sigmoid units. For training 5k random images were used.

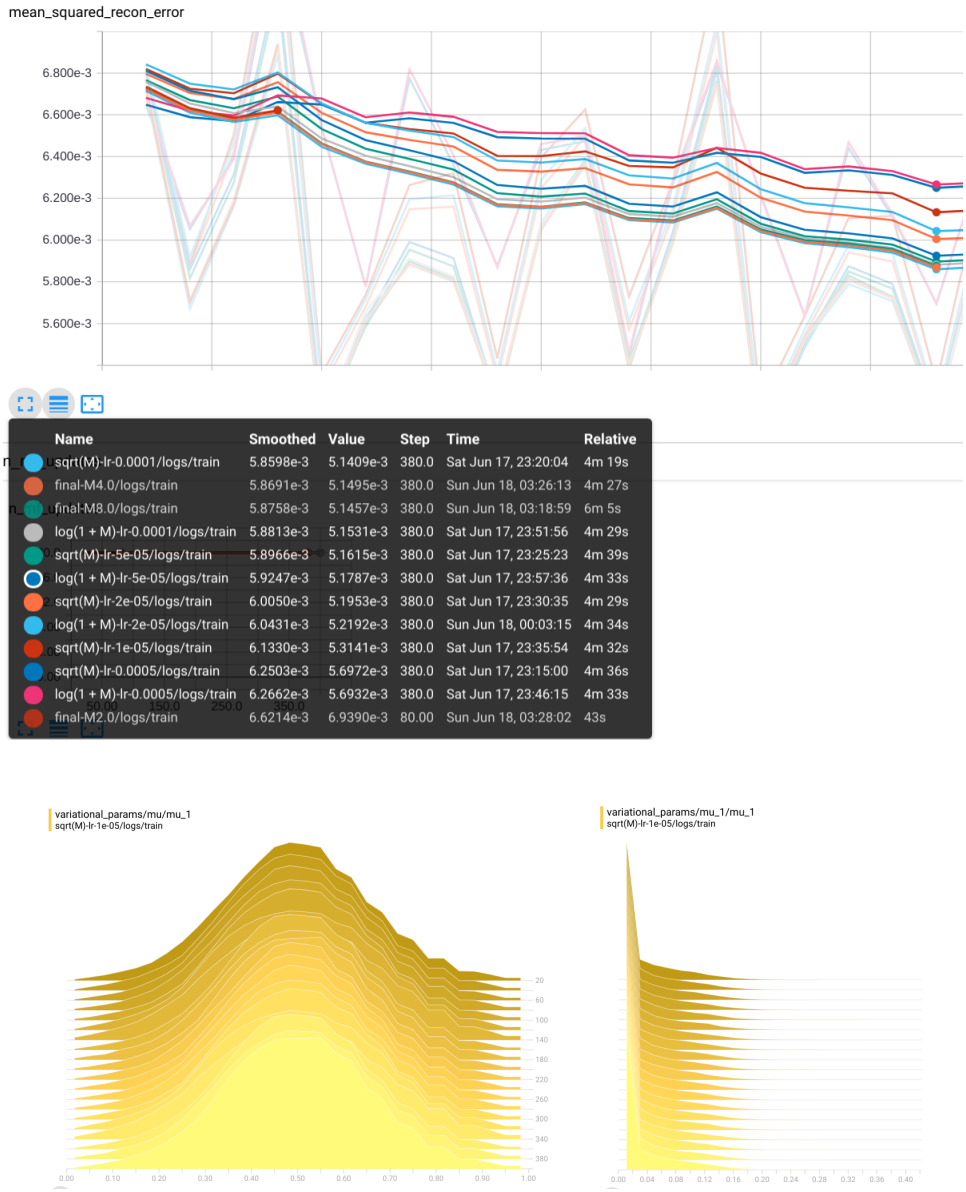


Figure 37: *Top*: reconstruction error when the total input from Multinomial layer were multiplied by $\frac{1}{\sqrt{M}}$ and $\frac{\log(1+M)}{M}$ from previous experiment. *Bottom*: distribution of variational parameters in first and second hidden layers after $\frac{1}{\sqrt{M}}$ scaling were applied.

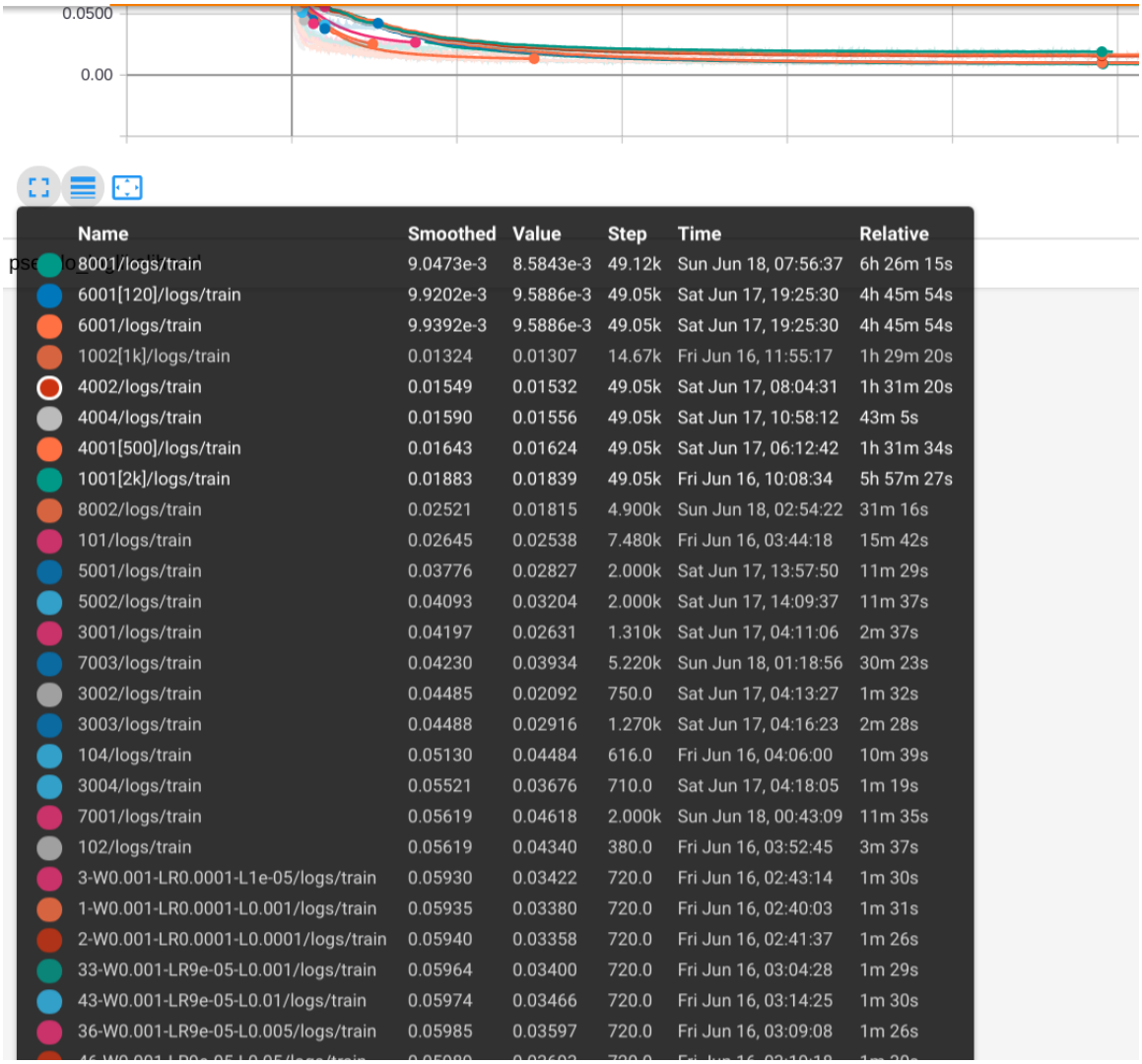


Figure 38: Reconstruction error of Multinomial RBM trained on extracted features $\mathbf{q}_i = p(\mathbf{h}|\mathbf{v} = \mathbf{x}_i; \boldsymbol{\psi})$ of Gaussian RBM on full dataset.

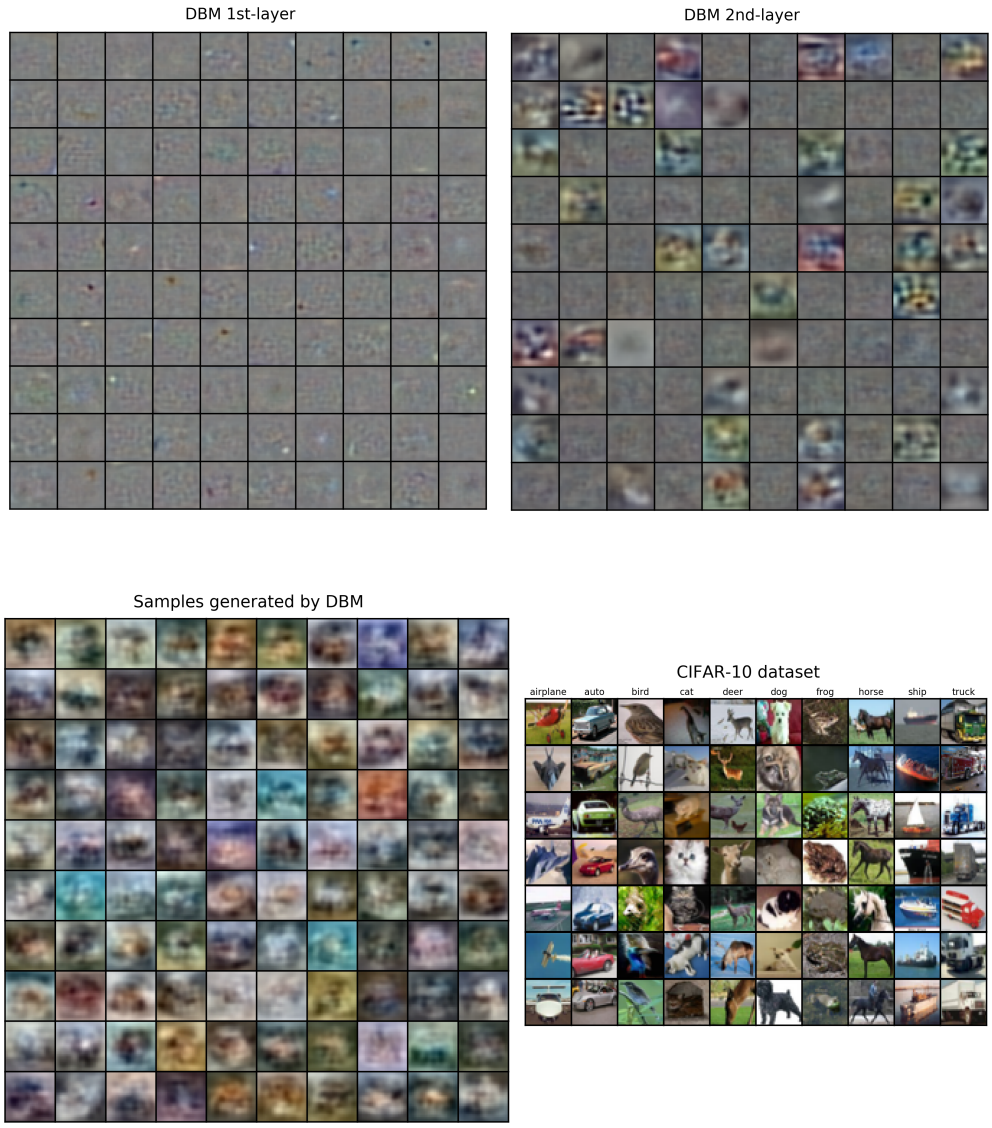


Figure 39: Best model trained on full CIFAR-10 dataset in this phase. 2nd layer filters are visualized using method of *weighted linear combinations* [4]. ***G-RBM Hyperparams***: L2 0.001, batch size 100, 99 epochs, lr 0.0005, momentum $0.5 \rightarrow 0.9$, 1 Gibbs step, w_std 0.0008, biases to 0, $\boldsymbol{\sigma}$ from data; sample both visible and hidden states. ***M-RBM Hyperparams***: L2 0.05, batch size 100, 118 epochs, learning rate 0.0001, momentum $0.5 \rightarrow 0.9$, 1 Gibbs step, number of samples $M = K = 1000$, w_std 0.01, biases to zero; sample only hidden states (for visible use probabilities w/o sampling); no scaling of total input. ***DBM Hyperparams***: 100 particles, initialize from data for all layers, 1 Gibbs step, 25 mean-field updates (10^{-13} tolerance), momentum $0.5 \rightarrow 0.9$, learning rate $9 \cdot 10^{-5} \rightarrow 10^{-5}$ by dividing by $(1.000015)^{600}$ each epoch, 200 epochs, batch size 50, L2 = 0, max norm = 2; sample all visible and hidden states.

"naive" training of Gaussian RBM

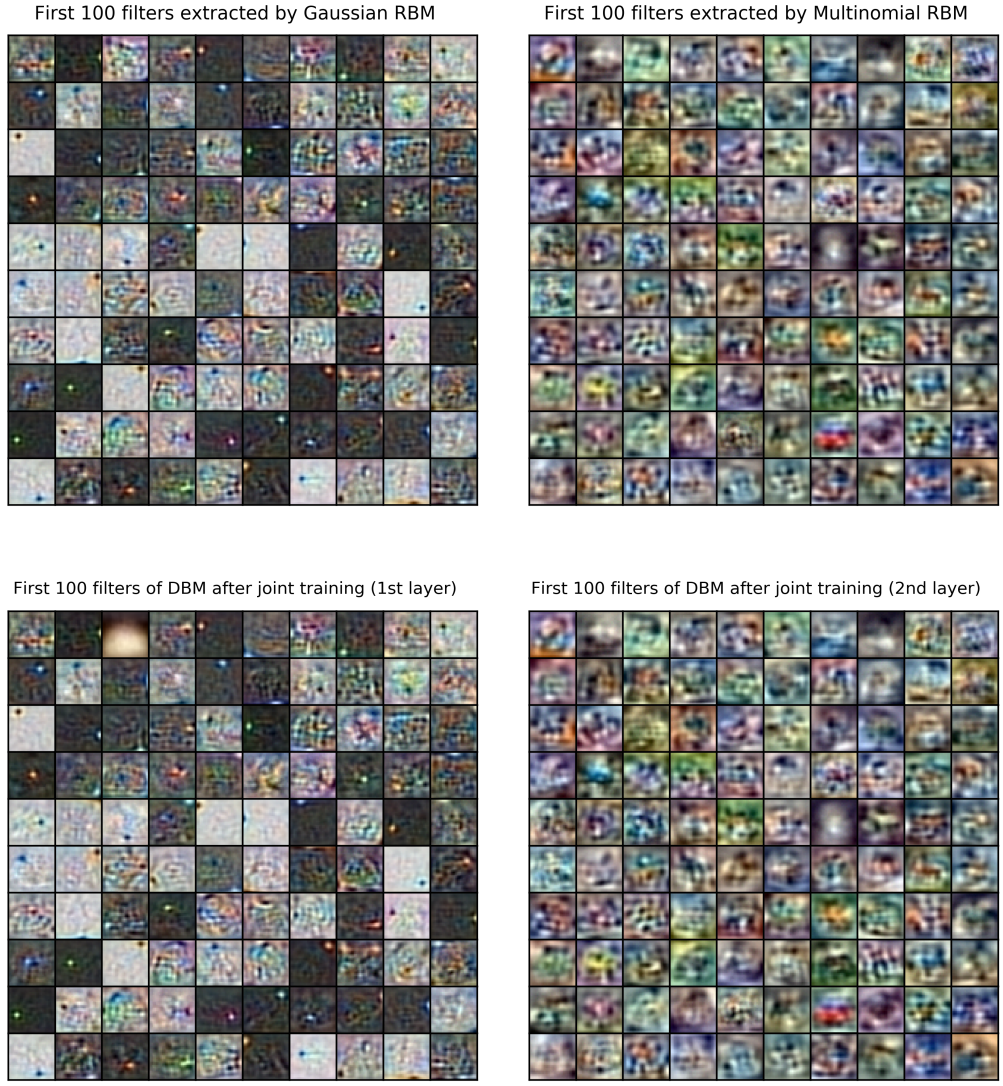


Figure 40: Weight filters

.

"naive" training of Gaussian RBM

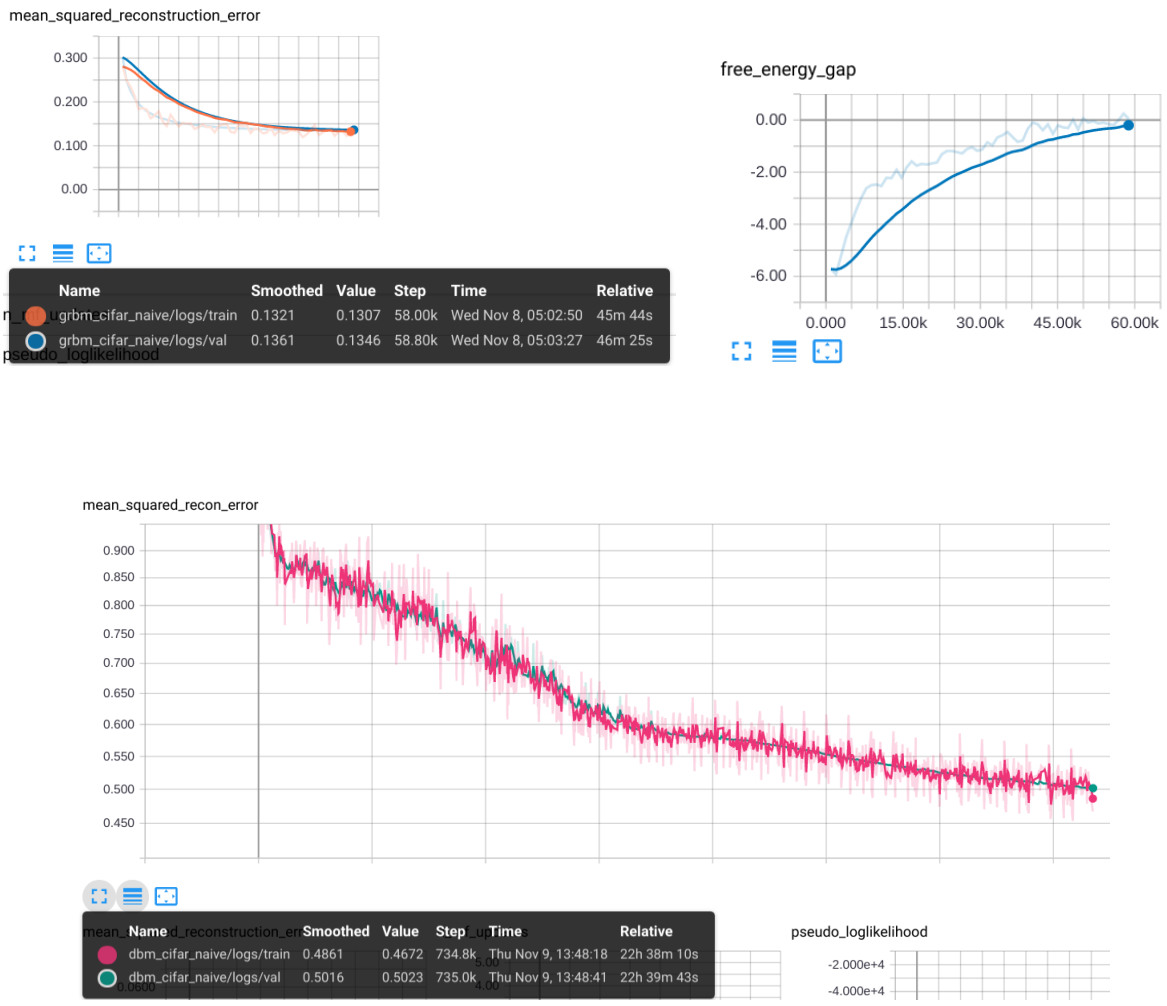


Figure 41: *Top*: reconstruction error and free nergy gap of Gaussian RBM. *Bottom*: reconstruction error of DBM after joint training.

.

advanced training of Gaussian RBM

In this experiment DBM was pretrained and trained on augmented (10 times) CIFAR-10 dataset with shifts by 1 pixel in all directions and mirroring. Also, Gaussian RBM was initialized from 26 small RBMs trained on patches of images, as in [15], see Fig. 42.
Observation: in [15] when they initialize large weight matrix, they leave zeros in places where there were no connections in small RBMs during pre-training. I found that using instead random values with small standard deviation (around 10^{-5}) yield slightly better performance with filters of large RBM smoothed in the neighborhood of corresponding filter from small RBM.

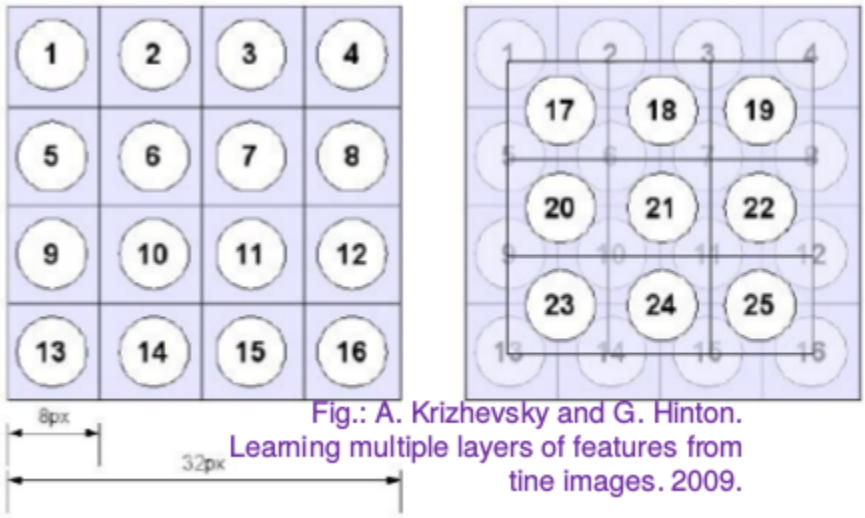


Figure 42: Pre-training 26 RBMs on patches of images. 26-th RBM was trained on subsampled versions of images.

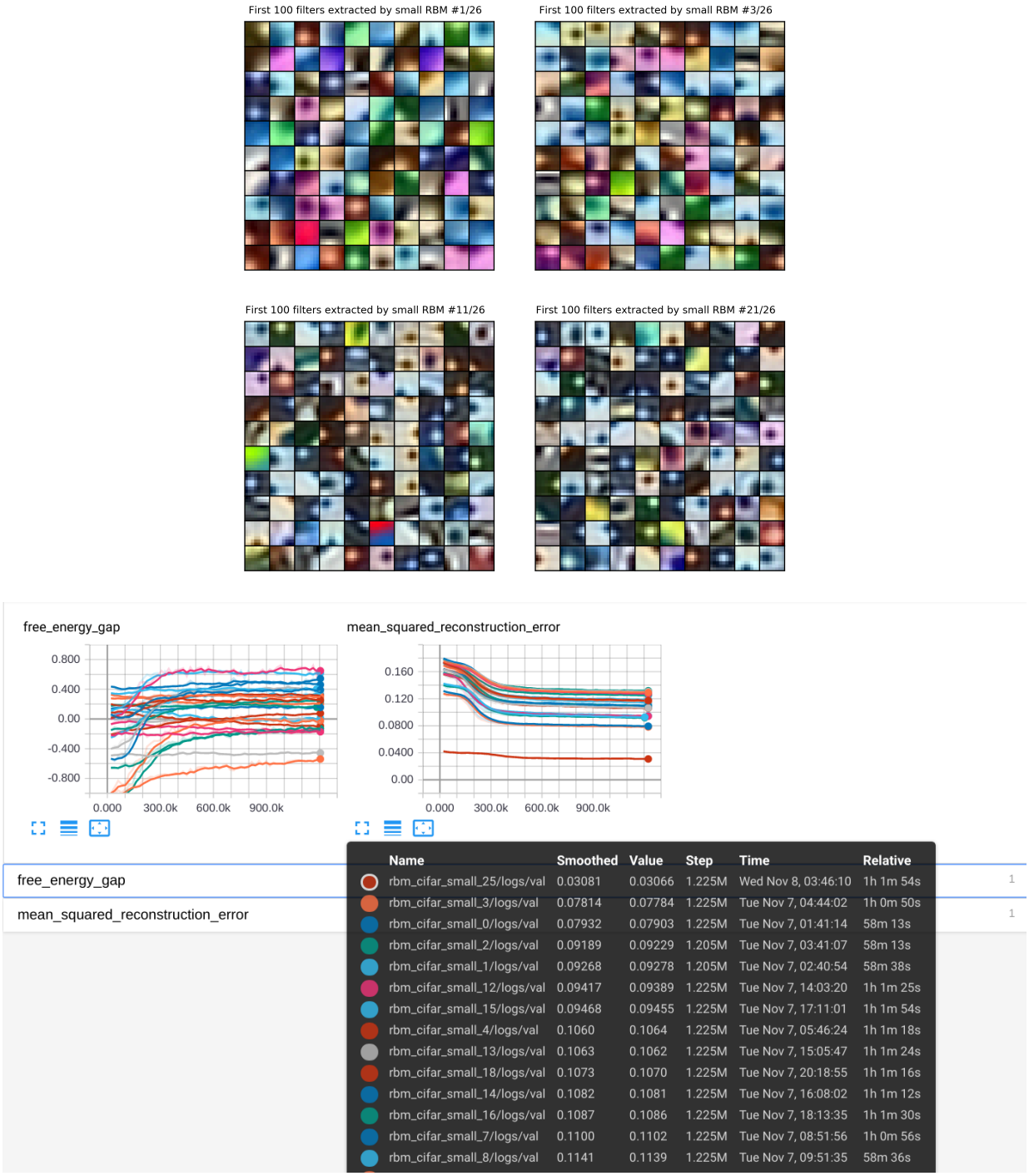


Figure 43: *Top*: Weight filters. *Bottom*: reconstruction error and free energy gap of small RBMs. Notice different groups of RBMs based on their reconstruction errors. The lowest (≈0.03) one has RBM which was trained on subsampled images. Next two (≈0.078) were trained on top left and right corners of the images, which are the smoothest among all patches. Next 4 (≈0.092) RBMs were trained on patches which are direct neighbors to top left and right, and after that come all the others.

.

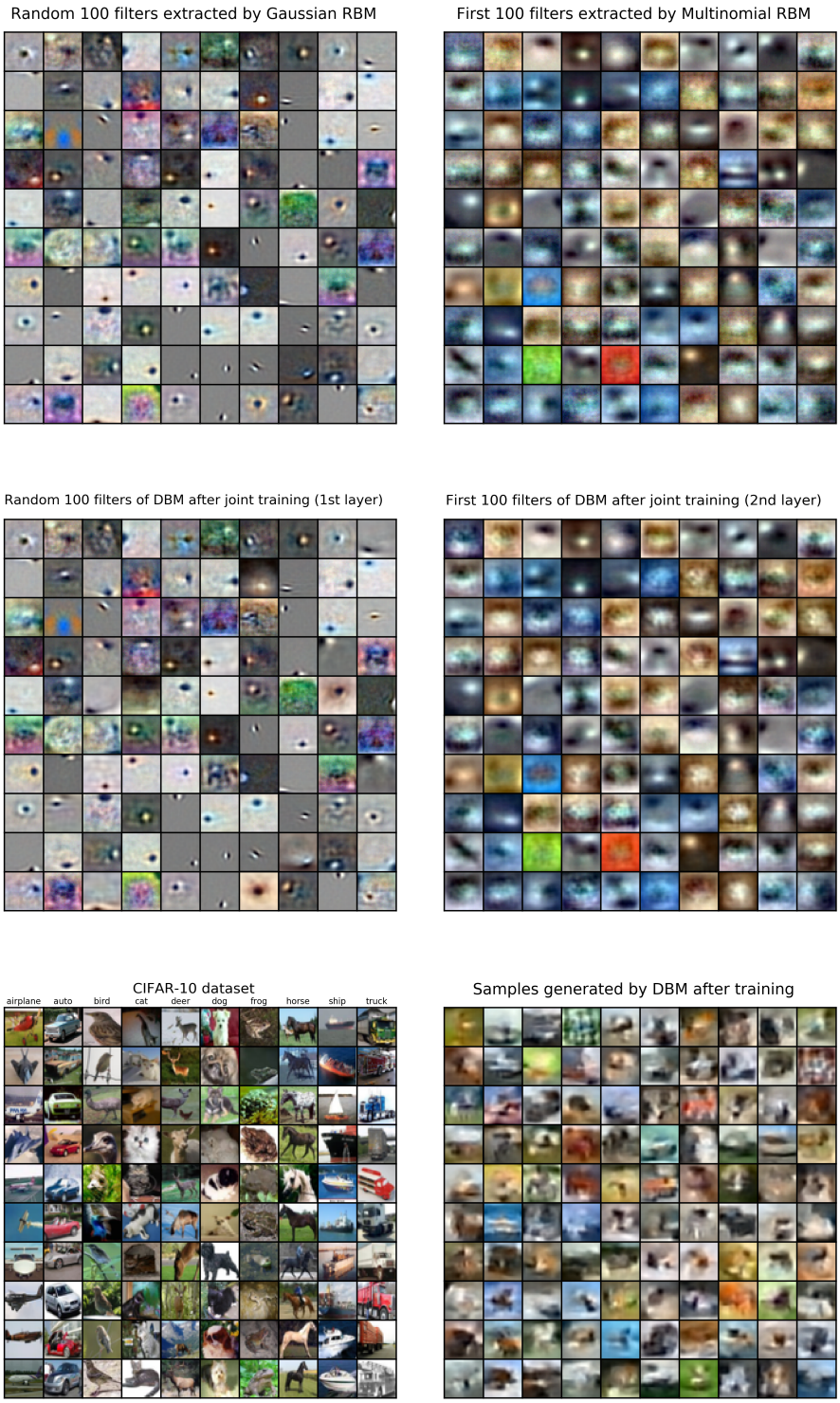


Figure 44: *Top*: Weight filters. *Bottom*: random subset of training data and generated samples by DBM.

.

Figure 45: *Top*: reconstruction error and free energy gap of Gaussian RBM *Bottom*: reconstruction error of DBM.

.

Some conclusions

- ✓ RBM, DBM: small models (1-2 layers) yet powerful;
- ✓ large enough to overfit (millions of learnable parameters) but they don't, thank to in-built regularizer / information bottleneck – random sampling;
- ✓ RBM, DBM and other energy-based models are quite principled from theoretical perspective: structure, distribution of visible and hidden units and basically everything is determined by and can be derived from energy function;
- ✓ for inference no need to input any additional information, as opposed to e.g. VAE (where for inference we need input latent code $\mathbf{z}$, and if we are not careful enough we may infer $\mathbf{x}$ somewhere between the modes which will result in unlikely image if the model was trained on images, unless something like cVAE has been used); this is not happen with RBM/DBM, we simply run Gibbs sampler using trained weights, and on decently trained model, most of the time particles output valid e.g. digits (MNIST);
- ✗ samples are highly correlated between one another, therefore many Gibbs steps need to be performed to obtain image from different mode ("slow-mixing problem");
- ✗ hard to tune, many hyperparameters, and they are not transferable between models, units' distriutions or tasks
- ✗ really hard to tune for complex tasks, many tricks;
- ✗ in papers lots of details omitted: exact hyperparameters in most cases, biases are omitted for "simplicity" in most papers etc.;
- ✗ hardly any working and complete examples of DBM codes;
- most of the features can be extracted in unsupervised manner, and then only limited amount of labeled information can be used to slightly adjust the layers of features already discovered by the DBM;
- RBMs and DBMs, especially classes for stochastic units could have been implemented purely in `edward`.

References

[1] Gibbs sampling. https://en.wikipedia.org/wiki/Gibbs_sampling. Accessed: 2017-09-17.

[2] Learning deep boltzmann machines, matlab code. http://www.cs.toronto.edu/~rsalakhu/code.html. Accessed: 2017-09-30.

[3] E. Aarts and J. Korst. Simulated annealing and boltzmann machines. 1988.

[4] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341:3, 2009.

[5] A. Fischer and C. Igel. An introduction to restricted boltzmann machines. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 14–36, 2012.

[6] A. Fischer and C. Igel. Training restricted boltzmann machines: An introduction. *Pattern Recognition*, 47(1):25–39, 2014.

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[8] I. Goodfellow, M. Mirza, A. Courville, and Y. Bengio. Multi-prediction deep boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 548–556, 2013.

[9] I. J. Goodfellow, A. Courville, and Y. Bengio. Joint training of partially-directed deep boltzmann machines. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, pages 1–10, 2012.

[10] I. J. Goodfellow, A. Courville, and Y. Bengio. Joint training deep boltzmann machines for classification. *arXiv preprint arXiv:1301.3568*, 2013.

[11] G. Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[12] G. Hinton. Coursera: Neural networks for machine learning, 2012.

[13] G. E. Hinton and R. R. Salakhutdinov. Replicated softmax: an undirected topic model. In *Advances in neural information processing systems*, pages 1607–1614, 2009.

[14] G. E. Hinton and R. R. Salakhutdinov. A better way to pretrain deep boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 2447–2455, 2012.

[15] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[16] G. Montavon and K.-R. Müller. Deep boltzmann machines and the centering trick. In *Neural Networks: Tricks of the Trade*, pages 621–637. Springer, 2012.

[17] U. of Leeds. Hopfield networks.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] R. Rojas. The hopfield model. In *Neural Networks*, pages 335–369. Springer, 1996.

[20] R. Salakhutdinov and G. Hinton. Deep boltzmann machines. In *Artificial Intelligence and Statistics*, pages 448–455, 2009.

[21] R. Salakhutdinov and H. Larochelle. Efficient learning of deep boltzmann machines. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 693–700, 2010.

[22] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[23] R. Salakhutdinov and I. Murray. On the quantitative analysis of Deep Belief Networks. In A. McCallum and S. Roweis, editors, *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pages 872–879. Omnipress, 2008.

[24] R. Salakhutdinov, J. B. Tenenbaum, and A. Torralba. Learning with hierarchical-deep models. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1958–1971, 2013.

[25] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE, 1986.

[26] T. Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM, 2008.

[27] D. L. Tutorial. Lisa lab. *University of Montreal*, 2014.

[28] V. Upadhya and P. Sastry. Empirical analysis of sampling based estimators for evaluating rbms. In *International Conference on Neural Information Processing*, pages 545–553. Springer, 2015.

[29] J. Yosinski and H. Lipson. Visually debugging restricted boltzmann machine training with a 3d example.