1 Introduction

Restricted Boltzmann Machines (RBMs) are neural networks that can reproduce a probability distribution, given a set of training samples. They have the architectural peculiarity of being composed of one visible (input) layer and one hidden layer. As opposed to general Boltzmann Machines, they are restricted in the sense that the units, or neurons, are not connected to other units in the same layer, only ones in the other layer. The values themselves of the units are binary (i.e, 0 or 1).

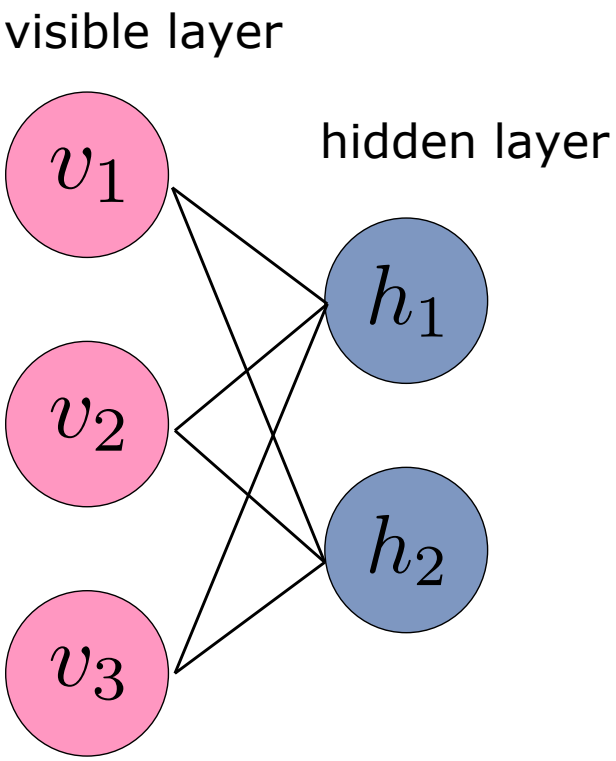


Figure 1: Example RBM with visible layer $\mathbf{v}$ consisting of $D = 3$ units and hidden layer consisting of $n_h = 2$ units.

2 Theory

Consider an RBM with a visible layer $\mathbf{v}$ of length D and hidden layer $\mathbf{h}$ of length n_h, where each component of $\mathbf{v}$ and $\mathbf{h}$ can be either 0 or 1 and let the parameters of this network be $\lambda = \{\mathbf{a}, \mathbf{b}, \mathbf{w}\}$. The probability of a configuration consisting of visible layer $\mathbf{v}$ and hidden

layer $\mathbf{h}$, with parameters λ, is given by the joint Boltzmann distribution:

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

where the partition function is:

$$\mathcal{Z}_\lambda = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{2}$$

and the "energy" is:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{D} a_i v_i - \sum_{j=1}^{n_h} b_i h_i - \sum_{i,j} v_i w_{ij} h_j \tag{3}$$

The marginalized probability of a visible vector can be obtained by summing out the hidden vector dependence:

$$P(\mathbf{v}) = \frac{1}{\mathcal{Z}_\lambda} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{4}$$

and likewise for the marginal probability of a hidden vector:

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

The conditional probability of a visible vector given a hidden vector is:

$$P_\lambda(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{D} P_\lambda(v_i|\mathbf{h}) \tag{6}$$

where the factorization of probabilities is due to the fact the input unit activations v_i are mutually independent, since there are no connections between units in the same layer. Similarly, the probability of a hidden vector given an input is:

$$P_\lambda(\mathbf{h}|\mathbf{v}) = \prod_{j=1}^{n_h} P_\lambda(h_j|\mathbf{v}) \tag{7}$$

The activation probabilities of each unit are given by:

$$P_\lambda(h_j|\mathbf{v}) = \begin{cases} \sigma(z_j) & h_j = 1 \\ 1 - \sigma(z_j) & h_j = 0 \end{cases} \tag{8}$$

where $\sigma(z)$ is the logistic sigmoid function:

$$\sigma(z_j) = \frac{1}{1 + e^{-z_j}} \tag{9}$$

and z_j is the weighted sum of inputs:

$$z_j = b_j + \sum_{i=1}^{D} v_i w_{ij} \tag{10}$$

The activation probabilities of input units are obtained similarly, replacing the weighted sum of inputs with:

$$z_i = a_i + \sum_{j=1}^{n_h} h_j w_{ij} \tag{11}$$

Using these probabilities, a Markov chain can be produced that generates hidden vectors from visible ones, and from the generated hidden vector, a new visible one:

$$\mathbf{v} \to \mathbf{h}' \to \mathbf{v}' \to \mathbf{h}'' \to \mathbf{v}'' \to \mathbf{h}''' \to \dots$$

At the end of the random walk, a model for the probability distribution of input vectors is obtained.

3 Training

Typically, RBMs are trained using the method of contrastive divergence (CD), which performs Gibbs sampling, and inside a gradient descent (GD) procedure, which updates the

network weights.

Unbiased samples are obtained via alternating Gibbs sampling using the unit activation probabilities derived in the previous section.

The cost function to be used is:

$$C = -\sum_{\mathbf{v}} P(\mathbf{v}) \ln P_\lambda(\mathbf{v}) \tag{12}$$

which is known as the Kullback-Leibler divergence, without the constant term. (Alternatively, it is the difference between two Kullback-Leibler divergence functions)

For network parameters $\lambda = \{\mathbf{a}, \mathbf{b}, \mathbf{w}\}$, the gradients for GD are:

$$\delta_a = \frac{\partial C}{\partial a_i} = -\langle v_i \rangle + \langle v_i' \rangle \tag{13}$$

$$\delta_b = \frac{\partial C}{\partial b_j} = -\langle h_j \rangle + \langle h_j' \rangle \tag{14}$$

$$\delta_w = \frac{\partial C}{\partial w_{ij}} = -\langle v_i h_j \rangle + \langle v_i' h_j' \rangle \tag{15}$$

The averages that show up in the gradients are computed by taking the average of the unit over sample in the batch, or mini-batch:

$$\langle v_i \rangle = \frac{1}{N_{\text{mini}}} \sum_{n=1}^{N_{\text{mini}}} v_i^{(n)} \tag{16}$$

$$\langle h_j \rangle = \frac{1}{N_{\text{mini}}} \sum_{n=1}^{N_{\text{mini}}} h_j^{(n)} \tag{17}$$

$$\langle v_i h_j \rangle = \frac{1}{N_{\text{mini}}} \sum_{n=1}^{N_{\text{mini}}} v_i^{(n)} h_j^{(n)} \tag{18}$$

To track the training progress, the cost function needs to be computed for the various

iterations of network parameters. Computing it though, will not be tractable due to the exponentially large size of the configuration space of hidden-visible vector pairs:

$$C = -\sum_{\mathbf{v}} P_\lambda(\mathbf{v}) \ln P_\lambda(\mathbf{v}) \tag{19}$$

$$\approx -\frac{1}{N}\sum_{n=1}^{N} \ln P_\lambda(\mathbf{v}^{(n)}) \tag{20}$$

$$= -\frac{1}{N}\sum_{n=1}^{N} [\sum_{\mathbf{h}} \ln P_\lambda(\mathbf{v}^{(n)}, \mathbf{h})] \tag{21}$$

$$C = -\frac{1}{N}\sum_{n=1}^{N} F(\mathbf{v}^{(n)}) + \ln \mathcal{Z}_\lambda \tag{22}$$

where the "free energy" $F(\mathbf{v}^{(n)})$ is defined such that:

$$e^{-F(\mathbf{v})} = \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{23}$$

which is exponentially hard to compute.

There are three alternative approaches to keep track of the training progress:

Approach 1: Track squared error between data and reconstructions

$$C_1 = \frac{1}{N}\sum_{n=1}^{N} \|\boldsymbol{v}^{(n)} - \boldsymbol{v}^{(n)\prime}\|^2 \tag{24}$$

Approach 2: Track categorical cross-entropy of visible layer's binary units.

$$C_2 = -\frac{1}{N}\sum_{n=1}^{N} \left\{ \sum_{i=1}^{D} v_i \log \sigma(\tilde{z}_i) + (1 - v_i) \log \left[1 - \sigma(\tilde{z}_i)\right] \right\} \tag{25}$$

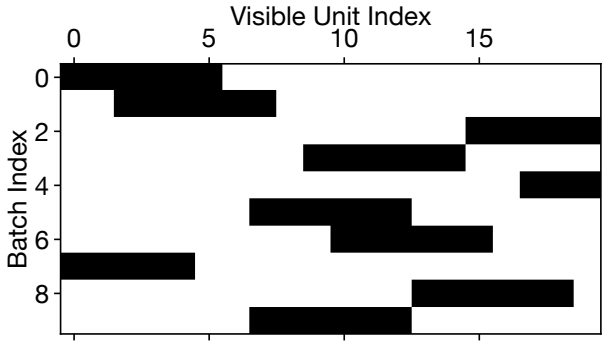
where $\tilde{z}_i = a_i + \sum_{j=1}^{n_h} w_{ij} h_j$.

Approach 3: Compare current free-energy to free-energy of validation data. The hope is that these should be approximately equal.

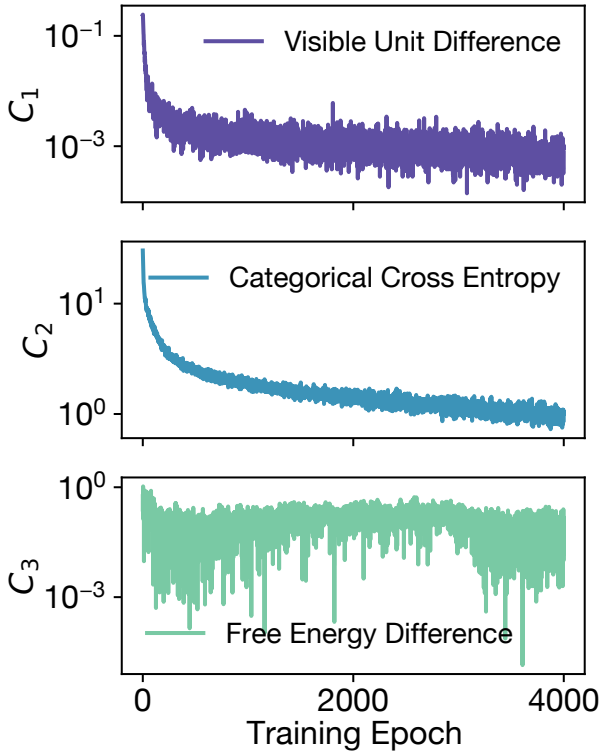
$$C_3 = \left| \frac{1}{N_{\text{val}}} \sum_{n=1}^{N_{\text{val}}} F(\boldsymbol{v}_{\text{val}}^{(n)}) - \frac{1}{N} \sum_{n=1}^{N} F(\boldsymbol{v}^{(n)}) \right| \tag{26}$$

4 Results

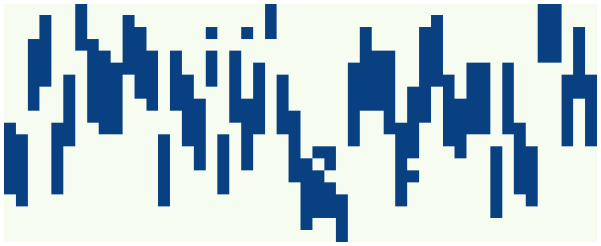
Example visible vectors for training



Comparison of cost function alternative approaches



Sampled visible vectors



5 Ising model

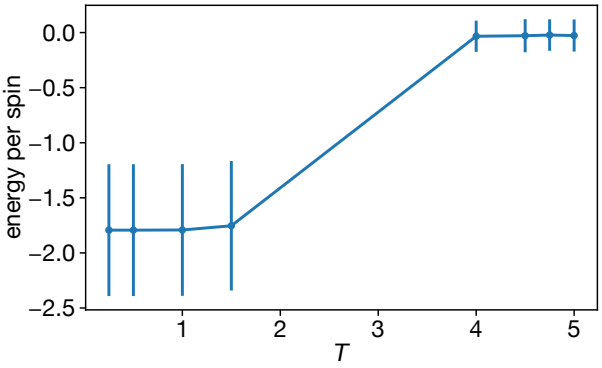


Figure 2: Energy per spin on a 10×10 lattice.

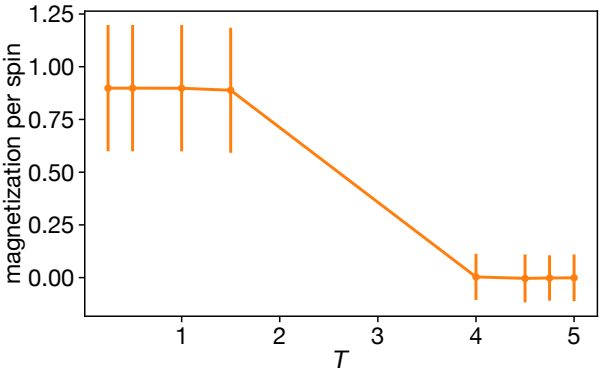


Figure 3: Magnetization per spin on a 10×10 lattice.

6 References

- Adrian's Machine Learning Lecture notes

- https://en.wikipedia.org/wiki/Restricted_Boltzmann_machine

- http://www.cs.toronto.edu/~hinton/absps/guideTR.pdf

- https://www.nature.com/articles/s41567-019-0545-1