Restricted Boltzmann Machines

Chaimaa Kadaoui chaimaa.kadaoui*
Othman SBAI othman.sbai
Xi Shen xi.shenxi

1 Abstract

For our project in Probabilistic Graphical Models course, we studied the use and the training of Restricted Boltzmann Machines as an instance of undirected graphical models. We mainly base our approach on the practical guide for training Restricted Boltzmann Machines (Hinton 2010 [2]). We start by an introduction to RBM and the interest that they have raised in the last years, then we define the theoretical aspects of main training methods of RBM. Finally we present our results by comparing our implementation of training RBM to an implementation from another deep learning library, on two popular image datasets MNIST and CIFAR-10. We show our convergence results and investigate the influence of different parameters of the training algorithm.

2 Introduction

2.1 Definition of an RBM

A Restricted Boltzmann Machine (RBM) is an undirected energy-based probabilistic graphical model which can be seen as a two layer neural network (visible and hidden units). RBMs can be used to model and learn important aspects of a probability distribution of a training data. They are called restricted, because we impose restrictions on the network topology, by allowing only connections between units of different layers. RBM are energy-based, since they define probability distribution through an energy function. Learning corresponds to shaping the energy so that desirable configurations have a low energy and thus maximize probability of training data under the model. Maximum likelihood learning is challenging for undirected graphical models because MLE parameters cannot be found analytically and the log likelihood gradient based optimization is not tractable. This optimization requires obtaining samples through Markov Chain Monte Carlo, which is computationally demanding.

2.2 Interest in RBM

A major interest have been dedicated to directed graphical models with one layer of observed variables and one or more layers of hidden variables such Mixture of Gaussians, probabilistic PCA, factor analysis, latent dirichlet analysis LDA... However, the undirected two layered analogue which was first introduced and called Harmonium by [6] and renamed as "Restricted Boltzmann Machines" by [3] have also seen advances through new training methods, which unlike directed graphical models, makes inference and learning very fast.

3 Applications of RBM

Restricted Boltzmann machines have successfully been applied to problems involving high dimensional data such as images, text [4], [8] as unsupervised feature extractors.

Recent developments have demonstrated the capacity of RBM to be powerful generative models, able to extract useful features from input data or construct deep artificial neural networks. In this setting

*mail extension @eleves.enpc.fr for all authors

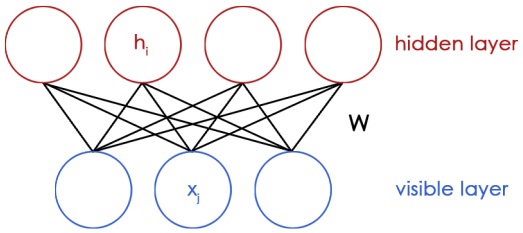

Figure 1: The network graph of an RBM

they can be seen as good preprocessing or initialization for some other models. They can also be used as self-contained classifiers as shown in [5].

- **RBM as a generative model**: learns and generates samples from data distribution
- **RBM as feature extractor**: unsupervised feature extraction from data, instead of handcrafting features
- **RBM for computer vision**: such as object recognition, image denoising and inpainting.
- **RBM for collaborative filtering**: given a set of N users and M movies, recommend movies to the users.

4 Theory of RBM

4.1 Definition of RBM

Restricted Boltzmann Machines are particular instance of undirected graphical models inspired by neural networks, where the units are organized in two layers.

- The visible layer $\mathbf{x}$ which contains all the visible nodes.
- The hidden layer $\mathbf{h}$ with all the latent variables.

It is *restricted* because no connections between nodes of the same layer are allowed. Each node of one layer is connected to all the other nodes of the other layer ($h_j \leftrightarrow x_k$). The matrix W characterizes the connections between the two layers: $W_{j,k}$ is the value for the connection between h_j and x_k. For simplification, we suppose that the variables $\mathbf{x}$ and $\mathbf{h}$ are binary.

4.2 Energy and probability

We introduce the two bias vectors c and b to define the energy of this graph:

$$E(x, h) = -h^\top W x - c^\top x - b^\top h$$

We can write the joint probability of $\mathbf{x}$ and $\mathbf{h}$ as:

$$p(x, h) = \frac{1}{Z} \exp(-E(x, h))$$

Z is the partition parameter which can be computed by calculating all the possibles values of $\mathbf{x}$ and $\mathbf{h}$. In practice, this parameter is intractable.

We see that to increase the probability, we want to decrease the energy. We have:

- if $c_k < 0$ and $x_k = 1$, the energy is higher; therefore we prefer having $x_k = 0$ (the probability of $x_k = 1$ decreases).
- if $c_k > 0$, similarly, we prefer $x_k = 1$ over $x_k = 0$.

We apply the same arguments for b and W and deduce similar interpretations of theses parameters.

5 Inference: Conditional inference

Because $p(x)$ is intractable, the only possible type of inference is computing conditional inference. We will prove that computing the conditional probabilities $p(x|h)$ and $p(h|x)$ is easy.

Because of the construction of the graph, the variables $h_1, \dots, h_H$ are independant conditionally on $\mathbf{x}$:

$$p(h|x) = \prod_j p(h_j|x)$$

We can also prove that given $\mathbf{x}$, h_j follows a Bernoulli:

$$p(h_j = 1|x) = sigm(b_j + W_{j.}x)$$

$W_{j.}$ is the j-th row of W and $sigm$ defines the sigmoid function:

$$sigm(x) = \frac{1}{1 + e^{-x}}$$

We can prove that either by derivation of $p(h|x)$ or by applying the *Local Markov Property*: $p(z_i|z_j,\ j \neq i) = p(z_i|\mathcal{N}(z_i))$.

5.1 Free energy

To have a better understanding of the parametrs W , c , b, let's introduce the concept of free energy. Using a marginalization and some derivations, we can write the probability $p(x)$ as :

$$p(x) = \sum_h p(x,h) = \exp\left(c^\top x + \sum_{j=1}^{H} \log\left(1 + \exp(b_j + W_{j.}x)\right)\right)/Z$$

The function $u \mapsto \log(1 + \exp(u))$ is called `softplus` and is a soft approximation of $u \mapsto \max(u, 0)$.

We can see that $p(x) = \exp(-F(x))/Z$ where F is the free energy. We want to increase the probability of our data x and thus decreases its energy. Let's look at each term of this energy:

- As seen in the previous subsection (4.2), the sign of c_k impacts wether x_k would be a 0 or 1. Thus, having $c^\top x$ in the probability means that we want $\mathbf{x}$ and the bias c to be aligned. This bias characterizes the probability.
- Similarly, $\mathbf{x}$ and $W_{j.}$ should align for each j. And we can see b_j as a control parameter: $W_{j.}x$ has to be big enough to have $b_j + W_{j.}x > 0$. Therefors, we can considers the hidden variables as features and b_j the bias for each feature (it defines the importance of each feature). The goal of the RBM is to represent meaningful features.

6 Contrastive divergence method for training RBM efficiently

To train the RBM with our data $\mathbf{x}^{(t)}$, we would like to minimize the average loss:

$$\frac{1}{T}\sum_t l(f(\mathbf{x}^{(t)})) = \frac{1}{T}\sum_t -\log p(\mathbf{x}^{(t)})$$

We want to apply a stochastic gradient descent. We derive each term of the sum with respect to our model parameter θ as follow (where $\mathbb{E}_h$ and $\mathbb{E}_{x,h}$ are expectations of h and (x,h) respectively):

$$\frac{\partial - \log p(\mathbf{x}^{(t)})}{\partial \theta} = \mathbb{E}_h\left[\frac{\partial E(\mathbf{x}^{(t)}, h)}{\partial \theta}|\mathbf{x}^{(t)}\right] - \mathbb{E}_{x,h}\left[\frac{\partial E(\mathbf{x}, h)}{\partial \theta}\right]$$

The first term is called the *positive phase* and the second term the negative phase. Because of the difficulty to compute the seconde term, we will use an algorithm called *Contrastive Divergence*. The idea is:

1. We estimate the expectation $\mathbb{E}_{x,h}$ by sampling a single point $\tilde{\mathbf{x}}$.
2. To do so, we use Gibbs sampling in chain (we apply it k times).
3. We initialize our Gibbs sampling with $\mathbf{x^{(t)}}$.

Therefore we can rewrite each term as:

$$\mathbb{E}_h \left[\frac{\partial E(\mathbf{x}^{(t)}, h)}{\partial \theta} | \mathbf{x}^{(t)} \right] \simeq \frac{\partial E(\mathbf{x}^{(t)}, h(\mathbf{x}^{(t)}))}{\partial \theta}$$

$$\mathbb{E}_{x,h} \left[\frac{\partial E(\mathbf{x}, h)}{\partial \theta} \right] \simeq \frac{\partial E(\tilde{\mathbf{x}}, h(\tilde{\mathbf{x}}))}{\partial \theta}$$

The goal is to minimize the energy of our training data and to increase the energy of our samplings at each iteration until we have samplings similar to our training data.

We define for a given x, the vector $h(x)$ as:

$$h(x) = \begin{bmatrix} p(h_1 = 1|x) \\ \vdots \\ p(h_H = 1|x) \end{bmatrix}$$

$$h(x) = sigm(b + Wx)$$

Let's look at the derivative of the energy E with respect to W, we can prove that:

$$\nabla_{\mathbf{W}} E(x, h) = -h(x)x^\top$$

Therefore, we can update W (α is the step of the gradient descent):

$$W = W + \alpha \left(h(\mathbf{x^{(t)}}) \mathbf{x^{(t)}}^\top - h(\tilde{\mathbf{x}}) \tilde{\mathbf{x}}^\top \right)$$

We do the same to update b and c:

$$b = b + \alpha \left(h(\mathbf{x^{(t)}}) - h(\tilde{\mathbf{x}}) \right) \qquad , \qquad c = c + \alpha \left(\mathbf{x^{(t)}} - \tilde{\mathbf{x}} \right)$$

1. For each $\mathbf{x^{(t)}}$
   (a) Generate sample $\tilde{\mathbf{x}}$ using k steps Gibbs sampling starting at $\mathbf{x^{(t)}}$
   (b) Update parameters:
   - $W = W + \alpha \left(h(\mathbf{x^{(t)}}) \mathbf{x^{(t)}}^\top - h(\tilde{\mathbf{x}}) \tilde{\mathbf{x}}^\top \right)$
   - $b = b + \alpha \left(h(\mathbf{x^{(t)}}) - h(\tilde{\mathbf{x}}) \right)$
   - $c = c + \alpha \left(\mathbf{x^{(t)}} - \tilde{\mathbf{x}} \right)$
2. Go back to step 1 until stopping criterion

The more steps we use for Gibbs sampling, the lower is the bias of our estimate. However it takes more time to compute. In practice, taking $k = 1$ works well to initialize the parameters of the algorithm.

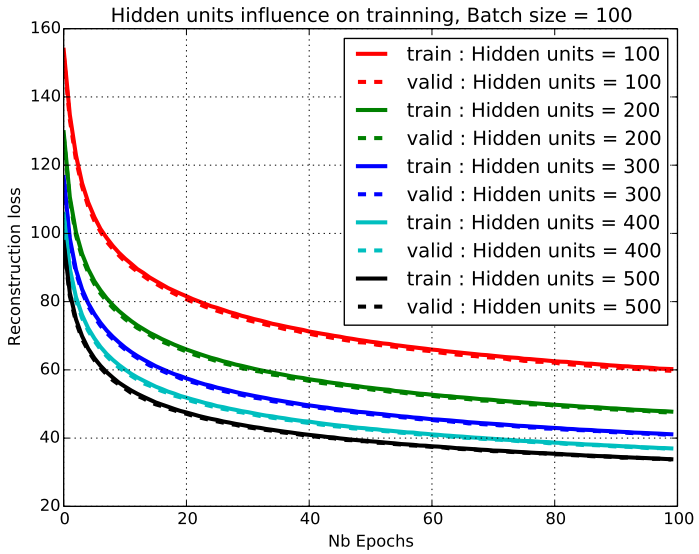

Figure 2: Influence of the number of hidden units

7 Implementation and results

We implemented the RBM training algorithm in python and then we compared our implementation performance with an existing deep learning library using tensorflow (YADLT), we applied it to the MNIST dataset (images of handwritten digits) and CIFAR-10 (color images of different classes). In the case of image data, each pixel is considered as a node; the intensity $p \in [0, 1]$ of a pixel is considered as a probability.

7.1 Implementation of RBM on python

7.2 Implementation of RBM on tensorflow using a GPU

We use the implementation in tensorflow provided by Gabriele Angeletti as a library gathering many deep learning models and algorithms such as convolutional networks, RNN, RBM, Deep Belief Networks...

7.3 Convergence plot

7.4 Influence of different algorithm parameters

7.4.1 Influence of batch size

7.4.2 Influence of number of hidden units

As we can see in the figure **??**, the number of hidden units has a direct impact on the performance of the algorithm. By increasing the number of hidden units, we can decrease the average stochastic reconstruction. In the figure, we can see that clearly 100 hidden units isn't enough; however for bigger values the curves become closer.

Since we deal with high dimensional data (images of abouts 1000 pixels each), we don't risk overfitting with a number of hidden units between 100 and 500 (the valiation loss is still below the training loss in each case). Hence, our main constraint for the choice of this parameter is the time complexity. We found that 200 or 300 was a good compromise between computing time and performances.

8 Conclusion and future work

References

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

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

[3] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[4] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[5] Hugo Larochelle, Michael Mandel, Razvan Pascanu, and Yoshua Bengio. Learning algorithms for the classification restricted boltzmann machine. *Journal of Machine Learning Research*, 13(Mar):643–669, 2012.

[6] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document, 1986.

[7] Tijmen 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.

[8] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.