University of Paris-Saclay - UFR Sciences
Lab 2 report - Restricted Boltzmann Machine

Diyun LU - Marwan MASHRA - Yihan ZHONG

23 October 2022

1 Introduction

1.1 Globally normalized models

Energy-based models (aka Globally normalized models) have a long history in machine learning and were introduced in 1982. They define the distribution of sequences in different ways. They use an energy function to summarize each possible sequence into a scalar quantity called energy. Furthermore, it defines the unnormalized probability of each sequence in terms of energy, and then normalizes the probability distribution over all sequences. Therefore, they are also called global normalized models. The global normalized model is a strict generalization of the local normalized model.

1.2 What is a Restricted Boltzman Machine

Restricted Boltzman Machine is a globally normalized model that can learn a probability distribution over its set of inputs. An RBM is a two-layer neural network, the first layer is known as the visible layer and the second layer is known as the hidden layer. Each node in visible layer is connected to the node in hidden layer. An RBM is considered **restricted** because no two nodes in the same layer are connected.

Suppose we have a RBM with the following sample spaces:

- $x \in \mathbb{R}^d$
- $y \in \{0,1\}^k$

Where x is the observed variables and y is the binary latent variables. The connection between the hidden layer and the visible layer can be defined by the weighting function:

$$w_\theta(x,y) \ = \ y^T a + x^T c + x^T B y \tag{1}$$

Where $y\theta = a, B, c, a \in R^k, B \in R^{d\times k}$,$c \in R^d$.

The joint probability distribution over all the random variables is defined as follows:

$$p(x,y) \ = \ \frac{\exp(w_\theta(x,y))}{Z(\theta)} \tag{2}$$

Where $Z(\theta)$ is the partition function that is intractable since we have to sum over 2^k possible value of y. However, thanks to having no intralayer connection in the latent space of RBM, we can actually use a Monte-Carlo estimation to avoid having to compute the sum over all values of y, which makes computing (approximating) the gradient of the log likelihood tractable (see Equation 5). This makes the computation much faster, and thus, represents a big advantage for Restricted Boltzman Machine over traditional Boltzman Machine.

1.3 Training objective

Let $D = x^{(1)}...x^{(d)}$ be the training set. To train an RBM, we would like to minimize the average negative log-likelihood.

$$\min_\theta \frac{1}{|D|} \sum_{x\in D} -logp(x;\theta) \tag{3}$$

We would like to proceed by stochastic gradient descent on the log-likelihood, by which we obtain:

$$\begin{aligned}\nabla\mathcal{L}(\theta, x) &= \nabla - \log p(x;\theta) \\ &= -\nabla c(\theta; x) + \nabla c(\theta) \qquad (4)\end{aligned}$$

The first term is the clamped log-partition function of $p(y|x)$. This term is trivial to compute by transform the original sum over an exponential number of values as k sums under the supposition that the latent variables are independent between them. The second term is still intractable, but we can use the fact that coordinates of y are independent, and rewrite it as an expectation that could then be approximated through a Monte-Carlo estimation :

$$\nabla_\theta c(\theta) = \mathbb{E}_{x' \sim p(x';\theta)}[\nabla c(x';\theta)] \qquad (5)$$

1.4 Gibbs Sampling

As we saw before, computing the loss requires doing a Monte-Carlo estimation in which we'll need to sample $x \sim p(x;\theta)$. However, as we know, we are not able to sample directly from $p(x;\theta)$, so we'll have to use other methods like Gibbs sampling.

In order to use Gibbs sampling, we need to be able to sample from both conditional distributions, i.e. we need to be able to sample :

1. $x \sim p(X|Y;\theta)$
2. $y \sim p(Y|X;\theta)$

Then, to preform Gibbs sampling, we just sample iteratively from the Markov Chain :

- $y^{(1)} \sim q(Y)$
- $x^{(1)} \sim p(X|Y = y^{(1)};\theta)$
- $y^{(2)} \sim p(Y|x = x^{(1)};\theta)$
- $x^{(2)} \sim p(X|Y = y^{(2)};\theta)$
- $y^{(3)} \sim p(Y|X = x^{(2)};\theta)$
- ....

Note that $q(Y)$ is a proposal distribution. We'll see how to choose it later in the implementation.

2 Implementation

2.1 Training (Contrastive Divergence)

During the training, we'll need to compute the loss, which means as we saw before that we'll need to sample from the distribution using Gibbs sampling Markov Chain Monte Carlo. However, we'll rely here on a specific technique called **Contrastive Divergence**. The idea of it is to perform a Monte-Carlo estimation with a simple sample that will be sampled from a single step Markov Chain. Furthermore, the proposal distribution here $q(Y)$ will be $P(Y|X = x^{gold})$, where x^{gold} is the true data we have (the batch). Preforming Contrastive Divergence will look like the following :

1. $y \sim p(Y|X = x^{gold};\theta)$
2. $\hat{x} \sim p(X|Y = y;\theta)$

Then the loss will be computed as following:

$$\text{Loss} = c(x^{gold};\theta) - c(\hat{x};\theta) \qquad (6)$$

2.2 Generation

Once trained, we can generate new data by sampling using Gibbs sampling Markov Chain. However, there are different ways to do that :

2.2.1 Single Markov Chain

The general idea of generating data via our trained RBM, as discussed in the class, is to do a single k-step Markov Chain using Gibbs sampling, and keep some of the samples following two parameters: b and s. For Gibbs sampling, since the latent space in RBM is binary $y \in 0, 1^k$, the proposal distribution is a Bernoulli distribution $\mathbb{B}(0.5)$.

The Generation process will look like the following :

1. $y^{(1)} \sim \mathbb{B}(0.5)$, then keep sampling $x^{(i)}$ and $y^{(i+1)}$ iteratively, as seen in the Gibbs sampling section, until reaching $x^{(k)}$.
2. We obtain $x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(k)}$
3. choose b and s, note that the number of generated data points will be $\frac{k-b}{s}$
4. The generated data will be $x^{(b)}, x^{(b+s)}, x^{(b+2*s)}, x^{(b+3*s)}, ...$

However, and as you can see in the results section, this method has an issue. In most cases, the generated data doesn't explore the entire space, i.e. all $x^{(b)}, x^{(b+s)}, x^{(b+2*s)}, x^{(b+3*s)}, ...$ will be centered around a certain area and won't cover the entire space, and that's no matter the value of k,b and s. To solve this issue and to get data covers better our space, we need to preform multiple Markov chains.

2.2.2 Multiple Markov Chains

The idea here will be to preform multiple Markov chains in parallel. So, instead of sampling $y^{(1)} \sim \mathbb{B}(0.5)$ we will sample

$$y^{(1,1)} \sim \mathbb{B}(0.5), y^{(2,1)} \sim \mathbb{B}(0.5), y^{(3,1)} \sim \mathbb{B}(0.5), y^{(4,1)} \sim \mathbb{B}(0.5),\ ...., y^{(n,1)} \sim \mathbb{B}(0.5)$$

with n being the number of Markov Chains. And then instead of sampling $x^{(1)} \sim P(X|Y = y^{(1)})$, we sample

$$x^{(1,1)} \sim P(X|Y = y^{(1,1)}), x^{(2,1)} \sim P(X|Y = y^{(2,1)}),\ ...., x^{(n,1)} \sim P(X|Y = y^{(n,1)})$$

and we keep doing that until sampling at the end

$$x^{(1,k)} \sim P(X|Y = y^{(1,k)}), x^{(2,1)} \sim P(X|Y = y^{(2,k)}),\ ...., x^{(n,k)} \sim P(X|Y = y^{(n,k)})$$

And in the end, when we obtain $n \times k$ samples of x, we only keep the last sample of each Markov Chain, i.e. our generated data will be $x^{(1,k)}, x^{(2,k)}, x^{(3,k)}....x^{(n,k)}$. You'll see later in the results section that the data generated from different Markov Chains has a much better coverage of the entire space. In practice, this could be easily implemented by simply expanding the dimension of x and y, it's like sampling a batch of x and y at a time.

3 Results of our experiments

3.1 Single Markov Chain

We see here that the generated data (red dots) doesn't cover well the space of the original data (blue dots), and is only concentrated around a certain area of the ring. Those are examples taken from 4 different Markov Chains with different values of b and s and $k = 1000$.

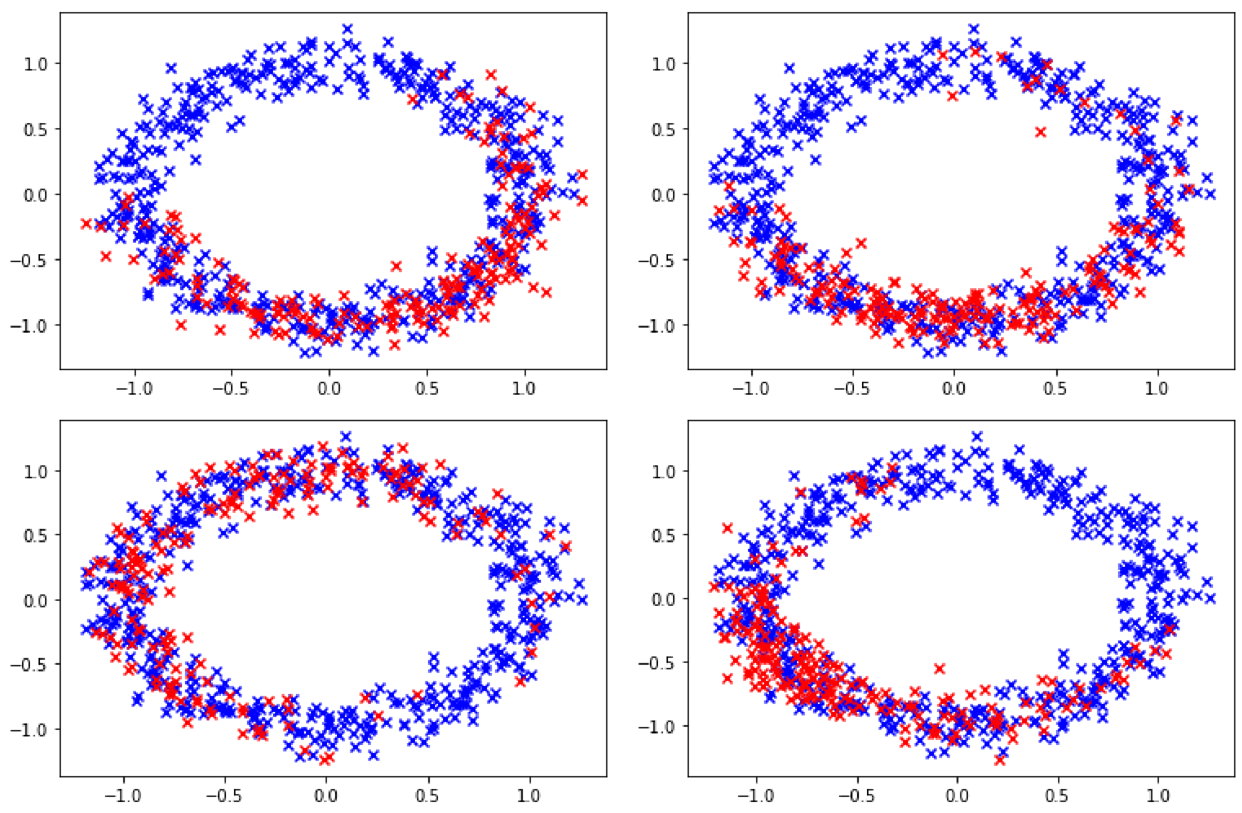

Figure 1: Data generated from a single Markov Chain

3.2 Multiple Markov Chains

As we see here, data generated from multiple Markov Chains results in a better coverage of the entire space. In this example, we have 500 Markov Chains that produced 500 data point (red dots), and they cover almost all areas of the ring.

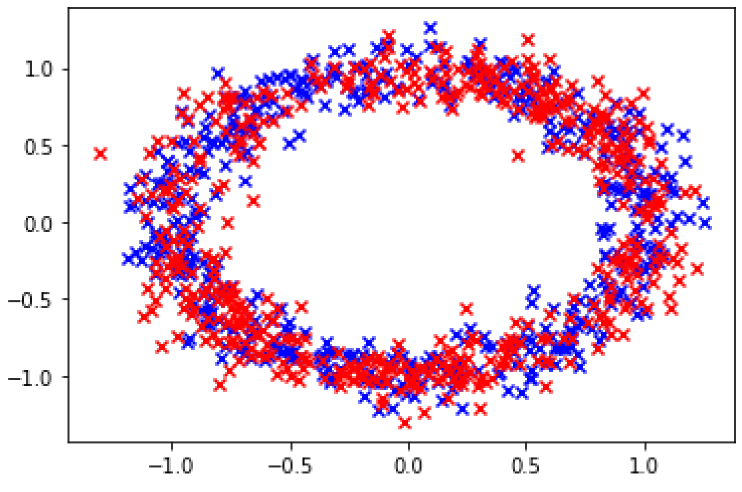

Figure 2: Data generated from a multiple Markov Chains