Deep Probabilistic Generative Models - Restricted Boltzmann Machine

Romain MUSSARD, Salma OUARDI

22nd October 2022

1 Introduction

In this lab exercise we had to implement a Restricted Boltzmann Machine from which we sampled data using Gibbs sampling, a Markov Chain Monte Carlo (MCMC). Since all the mathematical computation were provided we choose to re-write each part we had to implement using only matrices. The final goal being to mathematically describe the model in the almost same way we implemented it. Then we will discuss about this implementation and the sampling process.

2 RBM Implementation

The model was already mathematically describe in the lab work. However, in order to implement RMB we need to update the mathematical description to replace all sum by matrix operations, not only over one data-point but over a batch of data.

Let's start by defining our variables : $n \in \mathbb{N}$ is the size of our latent variable, $b \in \mathbb{R}^2$ is the bias of the observed variable, $\sigma \in \mathbb{R}^2_{++}$ is the variance of the observed variable, $W \in \mathbb{R}^{2\times n}$ is our matrix of weights, $d \in \mathbb{R}^n$ is the bias of the latent variable, $x \in \mathbb{R}^2$ is our observed variable and $y \in \mathbb{R}^n$ is the latent variable. In the next part, x_i^n and y_i^n will always refer to the dimension i of the sample n.

Furthermore, we introduce $m \in \mathbb{N}_{++}$ as our batch size, $X \in \mathbb{R}^{2\times m}$ the batch of observed variable such that $X_i = x^{(i)} \in \mathbb{R}^2$ and $Y \in \mathbb{R}^{m\times 2}$ the batch of latent variable such that $Y_i = y^{(i)} \in \mathbb{R}^{m\times n}$.

We have to extend σ, b and d to matrices since we want to adapt the formula to a batch of data. Thus, we introduce $\Sigma \in \mathbb{R}^{m\times 2}$ the batch of σ^2 such that $\forall i \in \{0, ..., m\}$, $\Sigma_i = \sigma^2$. In the same way, we define $D \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{m\times 2}$ such that $\forall i \in \{0, ..., m\}$, we have $D_i = d$ and $B_i = b$.

2.1 Conditional Probability of X

We have :

$$\forall i \in \{1, 2\},\ p(x_i|y;\theta) = \mathcal{N}\left(x_i \middle| b_i + \sum_{j=1}^{n} W_{ij}y_j; \sigma_i^2\right)$$

We are particularly interested by the mean of this Gaussian, as this is one of the parameters we want our model to learn. Thus we would like to be able to compute the mean for all the data of our batch in one time and avoid for loop. Let's then introduce $M \in \mathbb{R}^{m\times 2}$ such that $M_{ki} = p(x_i^{(k)}|y;\theta)$. We then have :

$$M = YW^T + B$$

In the lab work the function `p_x_given_z(self, z)` will return the matrix M and Σ, but this last parameter is fixed so we can return Σ as it is.

Sampling : By using the fact that $p(x_i^k|y^k;\theta) = \mathcal{N}\left(x_i^k \middle| b_i + \sum_{j=1}^{n} W_{ij}y_j^k; \sigma_i^2\right)$ we can sample new variables in the observed space. In order to do so we just have to compute M and Σ then each new data point element can be computed as $\forall i \in \mathbb{R}$, $x_i^{\prime(k)} = M_{ki} \times \mathcal{N}(0, 1) + \sigma_i$. Off-course to have a better distribution we will use Gibbs sampling Markov Chain Monte Carlo to generate new data points.

2.2 Conditional Probability of Y

This time we have :

$$\forall j \in \{1, ..., n\},\ p(y_j|x;\theta) = \mathcal{B}\left(y_j \middle| sigmoid\left(d_j + \sum_{i=1}^{2} \frac{x_i}{\sigma_i^2} W_{ij}\right)\right)$$

We are particularly interested by computing $\mu_j = sigmoid\left(d_j + \sum_{i=1}^{2} \frac{x_i}{\sigma_i^2} W_{ij}\right)$ in an efficient way as this is one of the parameter we want our model to learn.

Let's then introduce $L \in \mathbb{R}^{m\times n}$ such that $M_{kj} = \mu_j^k$. We then have :

$$L = sigmoid\left(\left(\frac{X}{\Sigma}\right) W + D\right)$$

Here $\frac{x}{\Sigma}$ is an element wise division, which is possible only because both have the same shape and each element of Σ is strictly positive.

In the lab work the function `p_z_given_x(self, x)` will return the matrix L. In the same way the sigmoid is an element wise function, here it make sure that the Bernoulli parameter is included in $[0, 1]$.

Finally, by using the fact that $p(y_j^k|x^k;\theta) = \mathcal{B}\left(y_j^k \middle| \mu_j^k\right)$ we can sample new latent variable. It is useful for example to compute the conditional probability of x. Generally, we will start the generation process by sampling $y^0 \sim \mathcal{B}(0.5)$

2.3 Forward

In the same way we would like, as much as possible to re-write the log partition function $c(X;\theta)$ with as much matrix operation as possible. We know that :

$$c(x;\theta) = \sum_{i=1}^{2} \frac{(x_i - b_i)}{2\sigma_i^2} - \sum_{j=1}^{n} \log\left[1 + \exp\left(d_j + \sum_{i=1}^{2} \frac{x_i}{\sigma_i^2} W_{ij}\right)\right]$$

Let's define $F = \dfrac{(X - B)}{2\Sigma}$ were the division is an element wise division, only possible because each element of Σ is strictly positive. Let's define J_n as a vector of size n such that each element of this vector is equal to one. Thus we can deduce that :

$$c(X;\theta) = J_2 F^T - J_n L^T$$

Thus $c(X;\theta)$ will be a vector of which each element will be the log partition function of each data point of the batch X. Of course using PyTorch we can just sum over the element of the matrix and we don't need vectors of ones, but we used this for more mathematical consistency.

2.4 Loss function

In our training algorithm we have a batch of data X, and through the RBM we reconstructed the data. So, to examine the reconstructed data X' we need to calculate its loss function which is basically a way we measure error, or the difference between our model reconstructions, X', and the input batch X. This loss function can be computed by simply calculating the mean of $(X - X')$

- X is our input batch data.
- X' is generated with our RBM machine. In our case the X' is obtained by sampling Z from X and then sampling X' from Z.

3 Implementation discussion.

3.1 Why is the RBM and interesting GNM?

RBMs have generally two roles:

1. Extract important features.
2. Reconstruct the input data.

These models make decisions about which features are important and how they should be combined to form patterns. In other words, it could be a feature extractor neural network used to discover hidden structures in the data.The weight and the bias help The RBM decide which input features are the most important when detecting patterns.

RBMs can automatically find patterns in our data by reconstructing the input. The reconstruction goes through 2 phases:

- Forward pass: takes input and translates them to a set of numbers that encode the inputs.
- Backward pass: takes the set of numbers and translate them back to form the reconstructed inputs.

RBMs also have a very efficient structure of a neural network, because one input layer can be used for many hidden layers for training.

And one more interesting aspect of an RBM is that the input data does not need to be labeled, which is helpful for real world data sets like photos,videos, etc. All of which tend to be unlabeled.

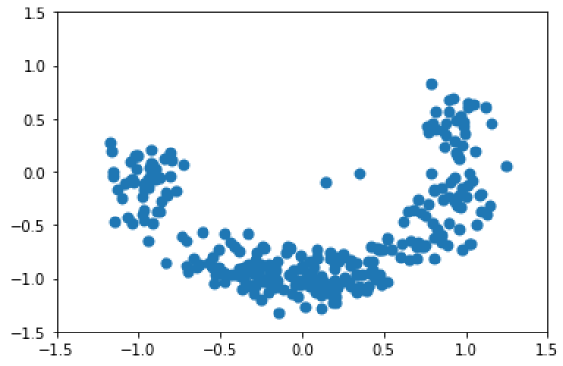
3.2 Gibbs sampling Markov Chain Monte Carlo.

To sample from the distribution, we can rely on Gibbs sampling Markov Chain Monte Carlo as it is easy to sample from both conditional distributions. In practice, we will rely on the contrastive divergence objective, i.e. we take a single sample using a single step of a Markov chain initialized with the datapoint x.

1. We use a single sample per data point in the Monte Carlo approximation of the partition function contribution to the loss.
2. We start one Markov Chain per data point in the batch.
3. We execute a few steps and then we take the last sample from each chain.

- **Observations**

Case 1: *Sampling with a single Markov Chain.(Figure 1a)*
Case 2: *Sampling with 500 Markov Chains at each time step.(Figure 1b)*


(a) case 1

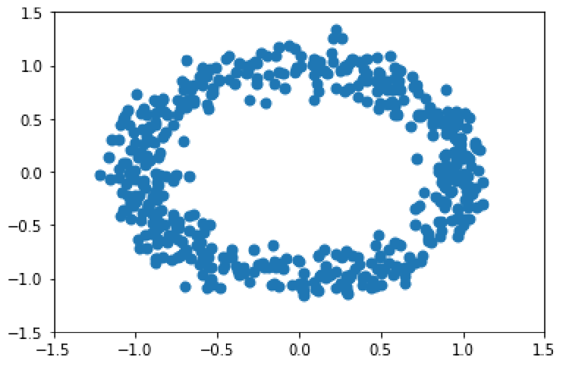

(b) Case 2

Figure 1: Two cases of Gibbs MCMC sampling.

- We can see (Figure 1a) that using only one Markov chain to generate many data points is not a good idea as it will not be a good representation of our data set, and that's because it will only sample from a given part of the original data set, which is obviously not representative enough.
- Otherwise, using multiple Markov Chains gave an accurate representation of the target distribution. (Figure 1b)

3.3 Advice to Generate Many Samples from the Model

- **Burn-in** is the practice of throwing away some iterations at the beginning of an **MCMC** run. We discard the first n samples before we start collecting points.

  The idea is that a *"bad"* starting point may over-sample regions that are actually very low probability under the equilibrium distribution before it settles into the equilibrium distribution. ***So we do not use the first* b *samples* $x^{(i)}$ *with* $(0 \leqslant i \leqslant b)$*.***
- If you happen to need several samples from you data, then only use 1-in-S samples.

4 Conclusion

In this lab work, we were able to implement the RBM and then generate new data by using Gibbs sampling, a Markov Chain Monte Carlo (MCMC). The results were good enough for us to draw conclusions and write observations regarding the training and the sampling process. At the end we came to the conclusion that using a single Markov Chain in the sampling will not be enough to represent well our data, the more the merrier in this case. Also, we were able to experiment a lot with the Matplotlib animation function ***(animate mc)*** that made it possible for us to better understand our experiments.