Generative Models

Introduction

- x_i random variable $\in \{0,1\}$
  - Discrete r.v.
- $X = (x_i, ..., x_p) \in \{0,1\}^p$
  - Multidimensional
- $f : \mathbb{R}^p \to \mathbb{R}$

- $\mathbb{E}(f(x)) = \sum_{x_i,...,x_p} f(x_i, ..., x_p)\ p(x_i, ..., x_p)$
  - Or on more compact notation:
- $\mathbb{E}(f(x)) = \sum_X f(x)\ p(x)$
  - Usually we don't even know the law $p(x)$
  - This is hard! The complexity is in the sum (???)
  - However, we could use some method to approximate this expected value

---

- $s : x^{(i)} \to \text{i.i.d} \to p(x)$
- $\frac{1}{n}\sum_{i=1}^{n} f(x^{(i)}) \to \text{p.s. (presque sûrement)} \to_{n\to\infty} \mathbb{E}(f(x))$
  - L.G.N. - loi *forte* des grands nombres
- We want to find out $x^{(i)}$

How to transform a uniform law in wathever distribution we want?

Gibbs sampler algorithm

- Hypothesis: We know the probability of each sample conditionally to the samples that come before and after it: $p(x_i | x_i, ..., x_{i-1}, x_{i+1}, ..., x_p)\ \forall i$
- Initialize: $x^{(0)} \sim \prod_{i=1}^{p} \text{where} q(x_i) = \text{Bernoulli}(\frac{1}{2})$
  - We sample a x_0 following a law of our choice

- For $p : 1 : L$
  - $x^{(l)} \sim p(x_i | x_i ... x_{i-1}, x_{i+1} ... x_p)$

Complexity: p * L

Theorem

$$\frac{1}{L}\sum_{p=0}^{L} f(x^p) \to \text{p.s. (presque sûrement)} \to_{n\to\infty} \mathbb{E}(f(x))$$

- As $n \to \infty$, we tend to the expected value $\mathbb{E}$ (esperance)

Restricted Boltzmann Machines (R.B.M.)

- In generative models, we want to learn the distribution of the real data

- We assume that real data follows a distribution
  - However, we'll never know this distribution analitically
- Latent variables
- The joint distribution between latent and observed variables is simple
- How to go from joint distribution to the law of the observations
  - Go from $p(x, y)$ to $p(x)$
- Discrete law multiplied by a conditional
- Wathever is the model we want to work with:
  - We have a simple joint distribution law
    - For ex a Markov Chain
  - We marginalize over the observed, and the remaining complicated law is an approximation of the original data

---

$$V = (v_1, ..., v_p), v_i \in \{0, 1\}$$

- *e.g.: Binary encoding over the movies a user has seen*

$$H = (H_1, ..., H_q), H_j \in \{0, 1\}$$

- *e.g.: Kind of represents a class ? For example, sci-fi fans*

Chose a simple $p(v, h)$

- Energy model:

$$p_\theta(v, h) = \frac{\exp(-\mathbf{E}(v, h))}{Z}$$

with

$$\mathbf{E}(v, h) = -\sum_{i=1}^{p} a_i v_i - \sum_{j=1}^{q} b_j, h_j - \sum_{i,j} w_{ij}, v_i, h_j$$

and

$$Z = \sum_{v_i, h_j} \exp(-\mathbf{E}(v, h))$$

- Model depends on parameter θ : $(a_i, b_j, w_{ij})_{i=1,...,p\ j=1,...,q}$

> It can be shown that:
>
> $$p(j|x) \text{and} p(x|j) \iff p(x, j)$$
>
> - If we have both conditional laws on a pair of variables, we have the joint law

---

$$p(h|v) = \frac{p(h, v)}{p(v)}$$

$$= \frac{p(v,h)}{\sum^h p(v,h)}$$

$$= \frac{\exp(-\mathbf{E}(v,h))}{\sum^h \exp(-\mathbf{E}(v,h))}$$

> exponentielle des sommes = produit des exponnentielles

> Sum of products is the product of sums
>
> $$\sum_{h_1,\ldots,h_q} g(h_1)\ldots g(h_q) = \sum_{h_1} g(h_1)\ldots \sum_{h_q} g(h_q)$$

$$= \frac{(\prod_i \exp(a_i v_i))\ (\prod_j \exp(b_j k_j) + \sum_i w_{ij} v_i h_j)}{(\prod_i \exp(a_i v_i))\ (\prod_j \exp(b_j k_j) + \sum_i w_{ij} v_i h_j)}$$

$$\ldots$$

$$= \prod_{j=1}^{q} g(h_j)$$

As seen above, the law $p(v,h)$ can be written as a product: conditionally to v, w and h are independent

With

$$g(h_j = 1) = \frac{1}{1 + \exp(-(b_j + \sum_{i=1}^{p} w_{ij} v_i))} = \mathrm{sigm}(b_j + \sum_{i=1}^{p} w_{ij} v_i)$$

$$g(h_j = 0) = 1 - g(h_j = 1)$$

Conclusion

$$p(h|v) = \prod_{j=1}^{q} p(h_j|v)$$

with

$$p(h_j = 1|v) = \mathrm{sigm}(b_j + \sum_{i=1}^{p} w_{ij} v_i)$$

And the other way around?

$$p_\theta(v|h) = \prod_{i=1}^{p} p_\theta(v_i|h)$$

with

$$p_\theta(v_i = 1|h) = \mathrm{sigm}(b_j + \sum_{i=1}^{p} w_{ij} v_i)$$

Visualizing

- The films aren't independent among them - if they were, there would be no point in what we are doing.
- However, when I know h, the v are independent
- Stochastic NN with only one layer
  - Neurons are actually variables
- From the NN generality theorem:
  - In theory, we can estimate whichever probability distribution law

$$KLD(???)$$

> **When using Generative models**
>
> - Generative power of the model
>   - Can the model architecture, theoretically, model the data?
> - Estimate the model's parameters, using the data
> - How to use it in practice
>   - How to choose the latent variables
>   - How to generate samples from the model

We have found the two conditional laws, so we can obtain the joint law

---

Applications

1. Data Generation

- Sample from the data distribution, but how to do it if we don't have $p_\theta(v)$?
  - (We have used the data to estimate the parameters θ)

$$v \sim p_\theta(v)?$$

Since we have both conditional laws:

$$p(v|h) \text{ and } p(h|v) \Rightarrow \text{Gibbs}$$

2. Dimensionality Reduction

$$q << p$$

Use h associated to v for a supervised estimation algorithm

Training the model (estimate θ)

- x is now the user
  - Each user's taste is assumed to be independent $x^{(k)}$ i.i.d.
  - x is the realization of v

$$x^{(1)}, ..., x^{(n)}, \text{with } x^{(k)} \in \{0,1\}^p$$

The log-likelihood will be the sum of the likelihoods on each of the x

> Likelihood is the law of what we observe

> We want to find theta as the argmax of the (log-)likelihood If on VAE, the marginal of the joint law - but we can't integrate it. We then use the ELBO, a lower bound on the likelihood.

For one x, we maximize w.r.t. θ:

$$\log p_\theta(x) = \log \sum_h p_\theta(x,h)$$

$$= \log \sum_h p_\theta(x,h)$$

> From Energy model:
>
> $$p_\theta(v,h) = \frac{\exp(-\mathbf{E}(v,h))}{Z}$$

$$= \log \sum_h \frac{\exp(-\mathbf{E}_\theta(x,h))}{Z}$$

> Z depends on θ, so we'll keep it

$$= \log \sum_h \exp(-\mathbf{E}_\theta(x,h)) - \log(Z)$$

$$= \log \sum_h \exp(-\mathbf{E}_\theta(x,h)) - \log\left(\sum_{v,h} \exp(-\mathbf{E}_\theta(x,h))\right)$$

> Here we use v as notation for the sum, we can do it since is a mute variable. We cannot use x to sum, because x is the data "qui tombe du ciel". x is a "réalisation" of v

In order to make a **gradient ascent**:

$$\frac{\partial \log(p_\theta(x))}{\partial w_{ij}} = \frac{\sum_h x_i\ h_j\ \exp(-\mathbf{E}_\theta(x,h))}{\sum_h \exp(-\mathbf{E}_\theta(x,h))} - \frac{\sum_{v,h} v_i\ h_j\ \exp(-\mathbf{E}_\theta(v,h))}{\sum_{v,h} \exp(-\mathbf{E}_\theta(v,h))}$$

$$= \sum_h x_i\ h_j\ p_\theta(h \mid v = x) - \sum_{v,h} v_i\ h_j\ p_\theta(v,h)$$

> **Marginalization**
>
> $$\sum_{x,z} p(x,y,z) = p(y)$$
>
> So:
>
> $$\sum_h ....Details$$

$$= \sum_{h_j} x_i \; h_j \; p_\theta(h_j \mid v = x) - \sum_{v,h_j} h_j \; v_i \; p_\theta(v, h_j)$$

$$= x_i \; p_\theta(h_j = 1|x) - \sum_v v_i \; p(h_j = 1|v) \; p(v)$$

With $f_{i,j}(v) = v_i \; p(h_j = 1|v)$:

$$= x_i \; p_\theta(h_j = 1|x) - \sum_v f_{i,j}(v) \; p(v)$$

$$= x_i \; p_\theta(h_j = 1|x) - \mathbb{E}\left(f_{i,j}(v) \; p(v)\right)$$

$p(h = 1|x)$ is what we know how to obtain

> To obtain the subtracting term, Gibbs will work. However, doing a Gibbs sampling for wach data point (our batch) will take too long - we want to initialize Gibbs in order to do only one iteration.

Gibbs

$$v^{(0)} = x$$

$$h^{(0)} \sim p_\theta(h|v^{(0)}) = \prod_j p_\theta(h_j|v^{(0)})$$

$$v^{(1)} \sim p_\theta(v|h^{(0)}) = \prod_i p_\theta(v_i|h^{(0)})$$

$$\Rightarrow \frac{\partial \log p_\theta(x)}{\partial w_{ij}} \approx x_i \; p_\theta(h_j = 1|x) - f_{i,j}(v^{(1)})$$

$$\frac{\partial \log p_\theta(x)}{\partial a_i} \approx x_i \; p_\theta(h_j = 1|x) - v_i^{(1)} p_\theta(h_j = 1|v^{(1)})$$

$$=$$

....

> $v_i^{(1)}$ is the reconstruction of the data x_i

- How to know if the model has learned?
    - Gradient is small, and the reconstruction is close to the initial data

Algorithm

For one data x

- Current θ
- $v^{(0)} = x$
- $h^{(0)} \sim p_\theta(h|x)$
- $v^{(1)} \sim p_\theta(v|h)$

> Parameters: One matrix of weights and two vectors of biases

- $\forall i, j$:
  - $\text{grad\_}w_{ij} = x_i p_\theta(h_j = 1|x) - v_i^{(1)} p_\theta(h_j = 1|v^{(1)})$
  - $\text{grad\_}a_i = x_i - v_i^{(1)}$
  - $\text{grad\_}b_j = p_\theta(h_j = 1|x) - p_\theta(h_j = 1|v^{(1)})$
  - $w_{ij} \leftarrow w_{ij} + \epsilon.\text{grad\_}w_{ij}$
  - $a_i \leftarrow a_i + \epsilon.\text{grad\_}a_i$
  - $b_j \leftarrow b_j + \epsilon.\text{grad\_}b_j$

Write pseudocode

- For each character, 39 samples
- q is hyperparameter

```
def init_RBM(p,q)

def in_out (X)
# Inputs X of size n x p
# Outputs n x q

def out_in (H)
# Inputs H of size n x q
# Outputs n x p

def train(X, size_batch, eps, epoch)
```