Generative modelling and normalizing flows

Pierre Mordant

April 19th 2021

Unsupervised learning

Supervised learning

Given 2 random variables $X \in \mathbb{R}^n$ and $Y \in \mathbb{R}$, we want to construct a function $f : \mathbb{R}^n \to \mathbb{R}$ so that $f(X) = Y$

We have a usually large but finite set of samples $\{(X_i, Y_i)\} \in (\mathbb{R}^n \times \mathbb{R})^p$

For that purpose, we usually construct a loss function L over a definite set of functions $\mathcal{G}$ and we take $f = \arg\min_{g \in \mathcal{G}} L(g)$

Unsupervised learning

Supervised learning

Given 2 random variables $X \in \mathbb{R}^n$ and $Y \in \mathbb{R}$, we want to construct a function $f : \mathbb{R}^n \to \mathbb{R}$ so that $f(X) = Y$

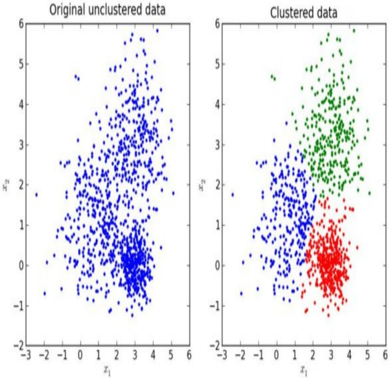
Unsupervised learning

Given a random variable $X \in \mathbb{R}^n$ of probability distribution $\mathcal{X}$, we want to construct a model that will be able to determine the inner patterns of the dataset.

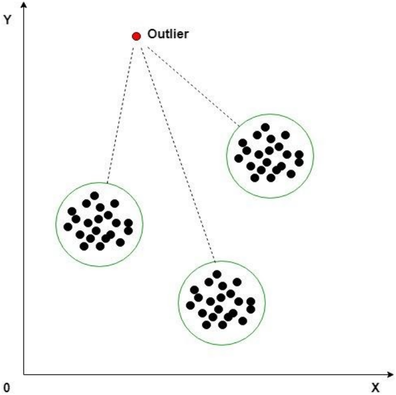
Main difference : Build a model based on $p(X|Y)$ VS build a model based on $p(X)$

Examples :

Clustering



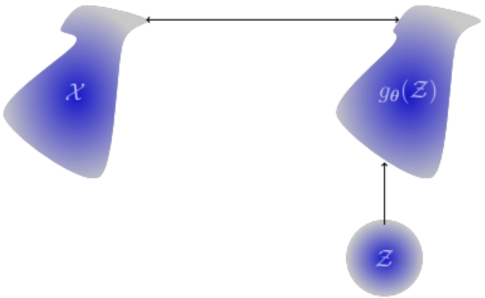
Anomaly detection



Generative modelling

Definition

Given a random variable $X \in \mathbb{R}^n$ of probability distribution $\mathcal{X}$, we want to learn a representation of that distribution.
For that purpose, we train a generator $g : \mathbb{R}^q \to \mathbb{R}^n$ so that $g(\mathcal{Z}) = \mathcal{X}$ where $\mathcal{Z}$ is a tractable probability distribution supported in $\mathbb{R}^q$



Applications

- Generate new samples : deepfake
- Estimate density function
- Estimate likelihood of new data points

Deep Generative modelling

Generative models trained with neural networks.

3 main types of deep generative models :

- Generative adversarial networks
- Variatonal auto-encoders
- Normalizing flows

Normalizing flows

Change of variables theorem

If $x = g_\theta(z)$ where g_θ is a diffeomorphism, we have

$$p_X(x) = p_Z(z)\left(\det J_{g_\theta}(z)\right)^{-1}$$

with J_{g_θ} the Jacobian matrix of g_θ

Normalizing flows

Constructing a diffeomorphism $T : \mathbb{R}^n \to \mathbb{R}^n$ such that $T(Z) = X$, using multiple transformations T_i such that $T_i(Z_i) = Z_{i+1}$ with $Z_0 = Z$ and $Z_p = X$

Maximum likelihood training

We train the model by minimizing the negative log-likelihood :

$$J_{ML}(x) = E_X\left(-log\left(p_Z(g_\theta^{-1}(x))(\det J_{g_\theta^{-1}}(x))\right)\right)$$

Residual flows

We consider a class of invertible transformation of the general form :

$$z' = z + f(z)$$

2 main ways to get an invertible transformation :

- Contractive residual flows
- Residual flows based on the matrix determinant lemma

Continuous normalizing flows

Idea : replacing a large number of finite steps by a continuous time approach.

Definition

Let z_t be the state of the flow at time t.
A continuous time flow is constructed using a function g_θ such that

$$\frac{\mathrm{d}z_t}{\mathrm{d}t} = g_\theta(z_t, t)$$

with $z_{t_0} = Z$ and $z_{t_1} = X$

We can then compute : $x = z + \int_{t_0}^{t_1} g_\theta(z_t, t)dt$
The log density can be computed with the trace of the Jacobian matrix

Continuous normalizing flows

Advantages :

- Memory efficiency : single call to an ODE solver
- Adaptive computation
- Using multiple hidden units in linear cost

Thanks for the attention!