Real NVP normalizing flow

Romain MUSSARD, Salma OUARDI

4th November 2022

1 Normalizing Flows Models

The main goal of normalizing flows is to map simple distributions (easy to sample and evaluate densities) to complex ones (learned via data) in an invertible manner.

Many of the models we studied show very good results for learning complex data distributions and having easy inference methods, However, most of them (e.g VAEs) lack the exact evaluation and inference of the probability distribution, which can result in blurry results in the case of VAEs.

The normalizing flows solve this issue by using reversible functions. Furthermore, they are are much easier to converge compared to other models. The training process is also easier, as it doesn't require careful hyper-parameter tuning.

2 Real NVP

The basic idea behind Real NVP is to constructs a transformation/flow between latent variables and observed variables such that the transformation satisfies the following two conditions:

- It is invertible and its Jacobian matrix's determinant is easy to compute.
- It is flexible such that it can transform random variables of simple distributions into random variables of complex distributions.

The transformation used in the real NVP method is a composition of multiple affine coupling transformations. In each affine coupling transformation, a subset of the random variables is kept the same and an affine transformation, parameterized using the fixed random variables, is applied to the remaining subset of the random variables. It is straightforward to show that the affine coupling transformation satisfies the above two conditions.

This allow us to define a tractable loss function while, with the previous models we studied like RBM, the loss was not directly tractable and required us to use a Lower Bound.

However, this model also have its disadvantages for example the fact that a part of the data is fixed in each layer or the necessity to use an invertible function to map the data.

3 Layer's Implementation

Let's first define our data-points $x \in \mathbb{R}^D$ such that $x \sim P_X$ with P_X is the data distribution we want to learn.

In a similar way we have a latent space $z \in \mathbb{R}^D$ such that $z \sim P_Z$, a known Gaussian distribution for example. We can notice that the latent space is of same dimension than our data.

The generative story that we want to follow is :

- $z \sim P_Z(z)$
- $x = g(z)$

Here g is a deterministic transformation. By using the change of variable formula we then have :

$$P_X(x) = P_Z(z) \left| det \left(\frac{\partial g(z)}{z^T} \right) \right|^{-1}$$

Thus this is the same as considering the log-likelihood :

$$\log \left(P_X(x) \right) = P_Z \left(g^{-1}(x) \right) + \log \left(\left| det \left(\frac{\partial g^{-1}(x)}{x^T} \right) \right| \right)$$

Where $\dfrac{\partial g^{-1}(x)}{x^T}$ is the Jacobian of g^{-1} at x

3.1 Jacobian's Constraints

One of the constrains we need to take into account for fast computation is that we want a Jacobian with an easy determinant's computation. Thus we can make sure to have such a property by constraining our Jacobian matrix to be a Lower Triangular Matrix. Indeed, the determinant of such matrices corresponds to the product of the elements of the matrix diagonal :

$$det \begin{bmatrix} \ell_{1,1} & & & & \\ \ell_{2,1} & \ell_{2,2} & & 0 & \\ \ell_{3,1} & \ell_{3,2} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{n,1} & \ell_{n,2} & \dots & \ell_{n,n-1} & \ell_{n,n} \end{bmatrix} = \prod_{i=0}^{n} \ell_{i,i}$$

3.2 Affine Coupling Layer

Thus a good away to assure this property is to fix one part of the data points and latent variables. In order to do so for $x \in \mathbb{R}^D$ we define $d \in \mathbb{N}$ such that $d < D$ and :

- $z_{1:d} = x_{1:d}$

- $z_{d:D} = x_{d:D} \cdot \exp\left(s(x_{1:d})\right) + t(x_{1:d})$

Here the dot $(\cdot)$ represent an elementwise multiplication. More precisely, s and t will be two neural network such that $s(x) = t(x) = f_1\left(\tanh\left(f_2(x)\right)\right)$ with f_1 and f_2 two linear function.

Thus the best way to implement the Affine Coupling Layer is by using a mask $m \in \mathbb{R}^D$ such that $m_{1:d} = 0$ and $m_{d:D} = 1$. This way we can rewrite this layer as :

$$z = x \cdot m + (1 - m) \cdot (x \cdot \exp(s(x \cdot m)) + t(x \cdot m))$$

These computations will be done in the forward function of our Real NVP and allow us to project from the latent space to the observed space. Thus this is the function we will use to generate new data-points by applying the inverse function after sampling z from a Gaussian law.

3.3 Inverse Coupling Layer

In the inverse layer the goal is to compute a datapoint x from a latent variable z. In order to do that we have to find the inverse function of $x_{d:D} \cdot \exp\left(s(x_{1:d})\right) + t(x_{1:d})$. As the functions s and t only depend of $x_{1:d}$ and that $z_{1:d} = x_{1:d}$, we can easily invert this function as we would do for a linear function. Indeed, $y = Ax + b \implies x = yA^{-1} - b$. Thus, we have :

- $x_{1:d} = z_{1:d}$
- $x_{d:D} = (z_{d:D} - t(z_{1:d})) \cdot \exp(-s(z_{1:d}))$

Again, by using the mask m we previously defined we obtain :

$$x = z \cdot m + (m - 1)\left((z - t(z \cdot m)) \cdot \exp(-s(z \cdot m))\right)$$

Furthermore, it's in this layer that we will compute the log determinant of the Jacobian. First, we can start by defining the Jacobian as :

$$\frac{\partial g^{-1}(x)}{x^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \dfrac{\partial z_{d:D}}{\partial x_{1:d}^T} & Diag\left(exp\left(-s(x_{1:d})\right)\right) \end{bmatrix}$$

Indeed, we have :

- $i \leq d,\ j \leq d : \dfrac{\partial z_j}{x_i} = \dfrac{\partial x_j}{x_i} = \mathbb{1}_{i=j}$, since we have $x_{1:d} = z_{1:d}$.
- $i > d,\ j \leq d : \dfrac{\partial z_j}{x_i} = \dfrac{\partial x_j}{x_i} = \mathbb{1}_{i=j} = 0$, again it's because we have $x_{1:d} = z_{1:d}$.
- $i > d,\ j > d : \dfrac{\partial z_j}{x_i} = \dfrac{\partial}{\partial x_j}\left(x_j - [t(x_{1:d}]_{i-d}) \cdot \exp(-[s(x_{1:d})]_{i-d}\right) = \mathbb{1}_{i=j} \cdot \exp(-[s(x_{1:d})]_{i-d})$

Thus, we have a lower triangular Jacobian and its log determinant is then equal to the product of the element on its diagonal:

$$\log\left(det\left(\frac{\partial g^{-1}(x)}{x^T}\right)\right) = \prod_{i=1}^{D} \exp\left(-s(z \cdot m)_i\right) = \exp\left(\sum_{i=1}^{D} -s(z \cdot m)_i\right)$$

Finally, this layer allow us to project from the observed space to the latent space. We will initialize the first latent variable z with our data x during the training. Given the previous computation, we will be able to easily retrieve the determinant of our Jacobian. Only this layer will be used to learn the parameters of our functions s and t, as we want to learn a good inverse matching function from x to the latent space.

4 Model Implementation

The coupling layers can be stacked (i.e one model can have multiple coupling layers) which is very useful to avoid that half of the variables stay unchanged. Indeed, in our implementation we alternate between Upper and Lower Triangular matrix for the Jacobian by inversing the mask. Furthermore, this allow us to ensure that our model reach every input, since each layer won't fix the same part of the data, and has enough capacity to learn the necessary invertible transform by using alternating patterns of spatial checkerboarding and channel wise masking with multiple coupling layers.

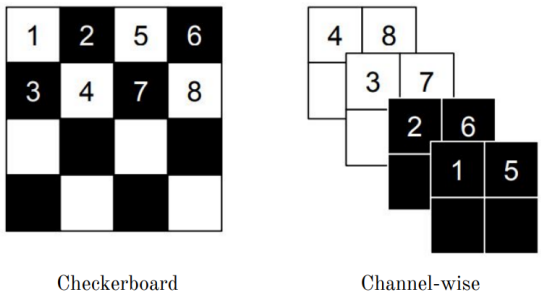


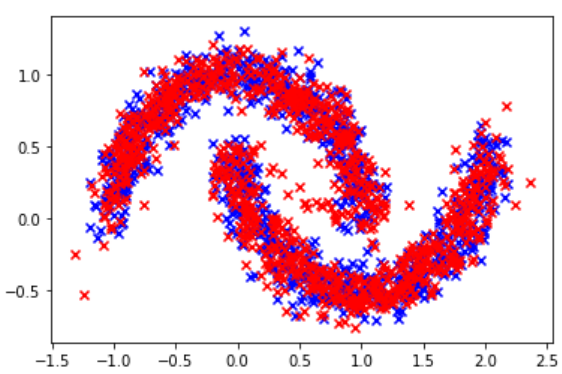
Figure 1: Mask types

Then we will need to implement two functions, responsible for calculating the forward and the inverse flows:

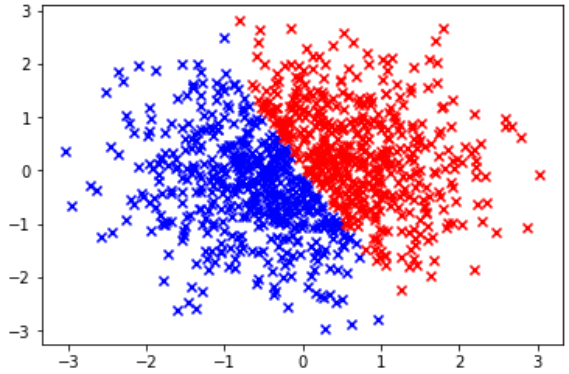
- ***forward***: is the composition of coupling layers on an input z and it returns x.
- ***inv***: is the composition of inversions of coupling layers on an encoded x and it returns z and $log\ det\ J$.

4.1 Training

Samples from the moon dataset, shown in the left of the following (Figure 2), are used to train a model with the real NVP method. After the training, these samples can be transformed into samples of standard normal distribution shown on the right.



(a) Samples from Moon dataset



(b) Latent space corresponding to each half moon

Figure 2: Training results

- (a) Blue is the real data distribution and Red is the generated one using our Real NVP model.
- (b) Each color is associated to one of the two moon and illustrates how they are represented in the latent space.