Normalizing Flows

Shresth Grover — Aman Agrawal
190820 — 190102

January 28, 2022

1 Background

Normalizing flows is a a class of generative model which passes a simple known probability distribution through a series of transformations to produce a richer, potentially more multi-modal distribution—like a fluid flowing through a set of tubes. The target complex probability distribution then can be used to model anything from a mixtures of gaussians to a set of images of cars.

2 Transformation of variables

Suppose we are given a random variable X with pdf given as $f_X(x)$. If we apply a function g to it to produce a random variable $Y = g(X)$. This process is called transformation of random variables. Suppose we want to know the density function of Y. If $g(X)$ is a bijective function then g^{-1} exists. Then -
Event $x < X < x + \delta$
$g(X) <= Y <= g(x + \delta)$
$g(x) <= Y <= g(x) + \delta|(\frac{dg(x)}{dx})|$
Hence, we get that $f_X(x) = f_Y(y)|\frac{dg(x)}{dx}|$

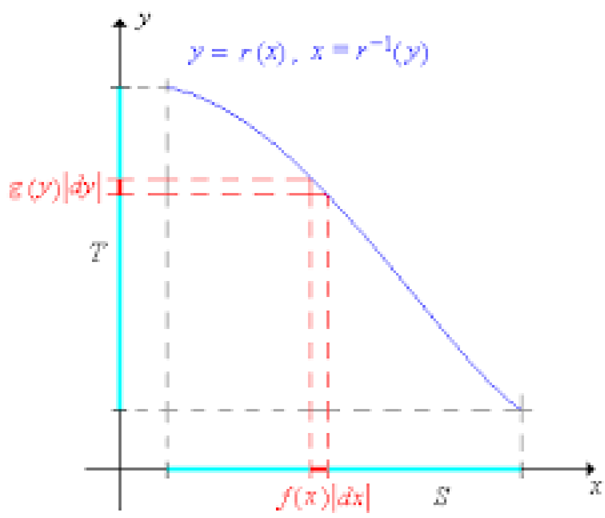
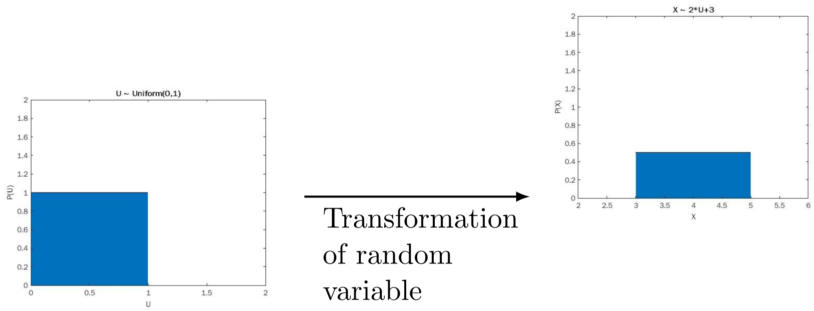


Figure 1: Caption

For example if we have a random variable U distributed $Uniform(0, 1)$ and let the transformation $g(X) = 2U + 3$ the pdf using the above formula is given by $f_X(x) = f_U(\frac{x-3}{2})\frac{1}{2}$



3 Normalizing Flow

Normalizing flows provide a general way of constructing flexible probability distributions over continuous random variables. Let X be a d dimensional random variable. Suppose we want to generate samples x from $p_X(x)$ when the distribution is given or some samples are provided instead of the distribution and we would like to generate more samples.

The purpose of normalizing flows is to express x as a transformation T over a real valued d dimensional random variable u which has been sampled from a known distribution.

$$x = T(u) \quad where \quad u \sim p_U(u)$$

Here $p_U(u)$ is the distribution which is easy to sample from also known as base distribution. The transformation T here has the parameters θ which provides $p_X(x; \theta)$ whereas base distribution $p_U(u; \phi)$ has the parameter ϕ. In case of normalizing flows the transformation T must be invertible and T and T^{-1} must be differentiable. Under these conditions the pdf of X is well defined and can be obtained as follows.

$$p_X(x) = p_U(u)|det(J_{T^{-1}}(u))|$$

A Neural Network is also a transformation which transforms the input x to output y. For training a Neural Network, we first define a loss function which is a function of the parameters in the neural net and it's differentiable and then optimise it by gradient based methods to find optimal parameters.

We would like to learn transformation T such that it is invertible, differentiable and determinant of Jacobian is computationally easy to compute.

3.1 Examples

- Let us look into the two-dimensional linear transformation case where $u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$, $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ such that $x = T(u)$ such that $x_1 = a * u_1 + b * u_2$ and $x_2 = c * u_1 + d * u_2$. The transformation T is invertible when $ad - bc \neq 0$ (How?).

The Jacobian is given by $J_T(u) = \begin{bmatrix} \frac{\partial x_1}{\partial u_1} & \frac{\partial x_1}{\partial u_2} \\ \frac{\partial x_2}{\partial u_1} & \frac{\partial x_2}{\partial u_2} \end{bmatrix} = \begin{bmatrix} \mathrm{a} & \mathrm{b} \\ \mathrm{c} & \mathrm{d} \end{bmatrix}$.

Hence, $p_X(x) = p_U(u)|det(J_T(u)|^{-1} = p_U(u)|ad - bc|^{-1}$

- Now, let's look at the two-dimensional non-linear transformation case where $u = \begin{pmatrix} \mathrm{u}_1 \\ \mathrm{u}_2 \end{pmatrix}$, $x = \begin{pmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \end{pmatrix}$ such that $x = T(u)$ such that $x_1 = u_1$ and $x_2 = u_2 * \sigma_2(u_1) + \mu_2(u_1)$ where σ_2 and μ_2 are non-linear differentiable functions.
The transformation is invertible and differentiable with respect to the parameters. The determinant of Jacobian $det(J_T(u)) = \sigma_2(u_1)$ is easy to compute.

3.2 Real Non-Volume Preserving Flow

In the previous example, we saw that the determinant of Jacobian was very easy to compute. In the example, x_1 was linearly depended on u_1 and x_2 was depended on u_2 linearly and past terms (x_1) non-linearly.

So, if we extend the similar idea to N dimensions where each x_i is depended progressively on the past terms, we will be able to compute the determinant of Jacobian efficiently. Such type of flow is known as Real Non-Volume Preserving Flow or in short R-NVP.

3.3 Implementationial details

While using a specific transformation function if we pass x_1 and x_2 in the same order it as shown in transformation when using neural network as a transformation it can form an unintended hierarchy in random variables x_1 and x_2 hence while implementing it its better to flip the order of random variables

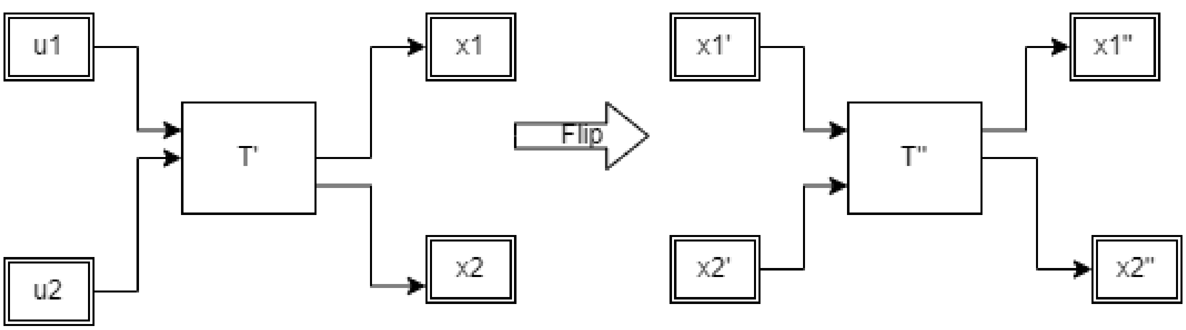


Figure 2: Flipping transformation

4 Training the Transformation

We obtain $p_X(x;\theta)$ from $p_U(u)$ which is known to us using normalizing flow with parameters θ. The parameter θ can consists of two types of parameters - parameter ϕ of transformation T such that $x = T(u;\phi)$ and parameter ψ for $p_U(u;\psi)$ if we want this distribution to be trainable.

Our goal is to match $p_X(x;\theta)$ with $p^*_X(x)$ which is our target distribution. We can use KL divergence, cross-entropy etc to match both these distributions. Let's use KL divergence to define our loss function.

$$
\begin{aligned}
L(\theta) &:= D_{KL}(p_X^*(x)||p_X(x;\theta)) \\
&= \mathbf{E}_{x\sim p_X^*(x)}[log(p_X^*(x)) - log(p_X(x;\theta))] \\
&= -\mathbf{E}_{x\sim p_X^*(x)}[log(p_U(u);\psi) + log(|det(J_{T^{-1}}(x;\phi))|)] + c
\end{aligned}
$$

where c is some constant.

To calculate this expectation value, we can use Monte Carlo Approximation. If $x_1, x_2, ..., x_N$ are samples from $p_X(x)$ then

$$\mathbf{E}_{x\sim p_X(x)}[f(x)] = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

In general, 20-30 samples are sufficient to estimate the expectation value.