ProbAI

Malavika Vasist

September 2022

1 Introduction

Normalizing Flows are invertible mappings transforming a simple probability distribution to a complex one. The change in the probability of a random variable z_0 due to such a transformation g is given by the change of variables equation below as

$$\log p(z_1) = \log p(z_0) - \log \left| \det \frac{\partial g}{\partial z_0} \right| \tag{1}$$

where $z_1 = g(z_0)$, and the determinant accounts for the change in volume between the two distributions. To increase the expressiveness of these transformations, they can be stacked up one on top of another as $z_K = g_{K-1} \circ g_{K-2} \circ \ldots \circ g_0(z_0)$, with the probability distribution given by

$$\log p(z_K) = \log p(z_0) - \sum_{k=0}^{K-1} \log \left| \det \frac{\partial g_k}{\partial z_k} \right| \tag{2}$$

Once the transformation is known, we can compute any expectation $\mathbb{E}_{p(z_K)}[h(z_K)]$ as

$$\mathbb{E}_{p(z_K)}[h(z_K)] = \mathbb{E}_{p(z_0)}[h(g_{K-1} \circ g_{K-2} \circ \ldots \circ g_0(z_0))] \tag{3}$$

The term Normalizing Flows comes from a complex distribution flowing towards a simpler 'normal' distribution, hence the direction $z_0 = f(z_1)$, where $f = g^{-1}$ is called the forward transformation, whereas g is called the reverse transformation.

In this report, I implement 3 types of NFs namely, Planar, Real NVP and Continuous NFs. There are 3 target distribution samples x $\sim p(x)$ implemented are two moons , boomerang and two blobs.

2 Planar

The transformation for the Planar flow is given by Rezende and Mohamed [2015] as,

$$z = x + uh(w^T x + b) \tag{4}$$

where, the flow parameteres $\lambda = \{w \epsilon \mathbb{R}^D, u \epsilon \mathbb{R}^D, b \epsilon \mathbb{R}\}$, h is the activation function and h' its gradient. The logarithm of the Jacobian is computed as,

$$log \left| \frac{\delta z}{\delta x^T} \right| = \left| \det \frac{\partial f}{\partial x} \right| = \left| 1 + u^T \phi(x) \right| \quad \text{where} \quad \phi(x) = h'(w^T x + b) w \tag{5}$$

and the change of variables formula can be written as,

$$\log p(x) = \log \pi(z) + \log \left| \det \frac{\partial f}{\partial x} \right| \tag{6}$$

where $\pi(z)$ is a standard gaussian distribution of D dimensions. These transformations are referred to as planar flows because the transformation occurs perpendicular to the hyperplane $w^T x + b$. Multiple planar transformations can be stacked up one on top of another to improve expressiveness.

To train the Planar flow, the (implied) target distribution $x \sim p(x)$ is sampled from and run through the forward transformation f defined by the neural network in order to obtain the latent parameters $z = f(x)$ and the jacobian $log \left| \frac{\delta z}{\delta x^T} \right|$. This is used to evaluate the density - log $p(x)$ using equation 6, which is treated as the loss, and minimized over the network parameters. Since Planar

flows donot have a closed form inverse, and can only be approximated, I use a root finding algorithm called Newton-Raphon method to sample $X_{\text{est}} \sim f^{-1}(Z) = g(Z)$ to plot. The architecture used for the planar forward transformation is a single neuron network.

Since there is no target distribution, I also implemented Planar flows in variational inference. Variational auto-encoders are generative models that estimate the target distribution $p(x)$ from the observation samples $x \in \mathbb{R}^{d_x}$, by considering a lower dimensional latent space $z \in \mathbb{R}^{d_z}$ (where $d_z < d_x$), with a joint density $p(x, z) = p(x|z)p(z)$, which is too complex to integrate in closed form. This problem is solved by re-structuring the setup as an optimization problem where the KL divergence D_{KL} between the real latent space posterior $p(z|x)$ and its estimate $q_\phi(z|x)$ is minimized. This is equivalent to optimizing the objective ELBO which is equal to,

$$\mathcal{L}_{\theta,\phi} = \quad ELBO = \mathbb{E}[\log_\theta p(x|z)] - \beta D_{KL}[q_\phi(z|x)||p_\theta(z)] \tag{7}$$

involving $q_\phi(z|x)$ that encodes observation x into a latent representation z, and $p(x|z)$ that decodes a latent configuration to observation vector x. This structure defines the Variational autor-encoder. ELBO collectively reduces the reconstruction error (first term) and the posterior estimation approximation error (second term). Normalizing flows are used to estimate the complex posterior $q_\phi(z|x)$. The setup involves an **encoder network** that maps the target observation samples x to the parameters of the simple initial density $q_0 = \mathcal{N}(\mu, \sigma)$ and the parameters of the flows. This initial density undergoes k **planar transformations** to give a more complex distribution q_k whose samples are then mapped onto the target observation vector x, using a **decoder network**. The complete objective function which is optimized is given by,

$$\mathcal{L}_{\theta,\phi} = \mathbb{E}_{q_0}[\log p_\theta(x|z_k)] + \mathbb{E}_{q_0}[\ln q_0(z_0) - \log p(z_k)] - \mathbb{E}_{q_0} \sum_{k=0}^{K-1} \log \left| \det \frac{\partial g_k}{\partial z_k} \right| \tag{8}$$

2.1 Results

Th results for Planar could not be obtained in either implementations. In the first implementation, since the inverse doesnt exist, the newton raphson method was used to compute the numerical inverse estimate. However the output of the function was constant for every sample of the input. In the second implementation, the output is not satisfactory, since it does not resemble the input distribution.

3 Real NVP

The transformation used in Real-valued Non-Volume Preserving (Real NVP) flows by Dinh et al. [2016], is an "affine coupling layer", where one half of the variables in the sample is scaled and shifted according to the scale and shift values computed, using a neural network by the other half in the forward direction f. It is given by the equations,

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

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

where D is the dimensions of the samples from the target distribution $x \sim p(x)$, $d < D$, s and t are scale and shift parameters respectively. Following this, the reverse is given by,

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

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

The logarithm of the Jacobian, which is a right triangular matrix, is computed as

$$log \left| \frac{\delta z}{\delta x^T} \right| = \log \left| \det \frac{\partial f}{\partial x} \right| = \Sigma_j s_j(x_{1:d}) \tag{9}$$

and the change of variables formula can be written as in equation 6. Multiple NVP transformations can be stacked up one on top of another to improve expressiveness.

Similar to the Planar flow, to train the NVP flow, the (implied) target distribution $x \sim p(x)$ is sampled from, and run through the forward transformation f defined by the neural network and

the coupling, to obtain the latent parameters $z = f(x)$ and the jacobian $log \left|\frac{\delta z}{\delta x^T}\right|$. This is used to evaluate the density - log $p(x)$ which is treated as the loss, and minimized over the network parameters. To sample from the NVP flow, the inverse transform $x = f^{-1}(z) = g(z)$ is computed.

For the boomerang dataset, the neural network defining the forward transformation is defined by 4 fully connected linear layers with hidden dimensions 256, 128 and 32 with an output of 2 units. Sigmoid non-linearities and 15 bijection layers were chosen. For training, an Adam optimizer was used with a learning rate of 0.5e-4 for batches of size 4. The training was carried out for 2000 epochs taking around 2 cpu hours with a 4GB RAM.

Similarly, for datasets two moons and two blobs, 3 fully connected layers with 32 hidden units and an output of 2 units was used. ReLU non linearities and 9 bijection layers were chosen. For training, an Adam optimizer was used with a learning rate of 0.5e-4 for batches of size 32. The training was carried out for 2000 epochs upon emperical evaluation taking around 1.5 cpu hours with a 4GB RAM for each dataset.

3.1 Results

The results for the three target samples are shown below.

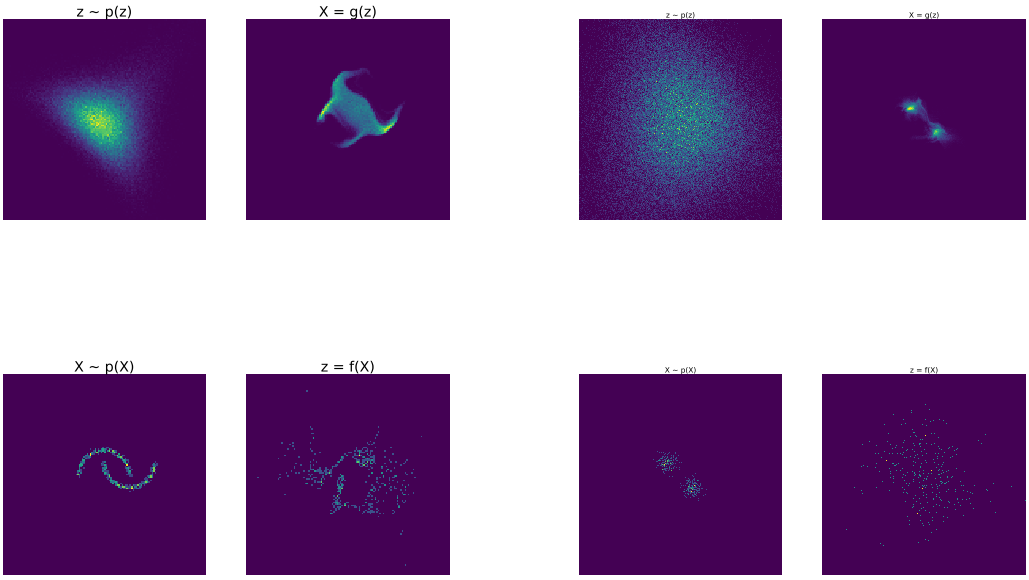

Figure 1: Two The results of the Real NVP Flow for datasets two moons and two blobs.

The tentacles seen in the two moons and boomerang reconstruction appear to be mirror images of the moon and boomerang legs that are transformed ambiguously. This might suggest that the coupling layers are not deep enough. With a higher number of bijections, one might be able to retrieve a better reconstruction.

3.2 Differences between Planar and Real NVP

The coupling of the affine layer in RealNVP in itself doesn't improve expressiveness but stacking them incredibly improves expressiveness. The coupling and stacking allows for a complete inter-mixing of dimensions capturing all aspects of the input vector resulting in complex density distributions.

RealNVP is invertible in closed form unlike Planar flows. This allows us to easily sample from it and move back and forth the forward and backward spaces efficiently. On the other hand in Planar flows one has to rely on numerical methods like iterative bisection and netwon raphson method to sample from the flow.

4 Continuous Normalizing Flows

Continuous Normalizing Flows (CNF) are a time counterpart of normalizing flows given by Chen et al., where the transformations of the flow are replaced by Ordinary Differential Equations (ODEs).

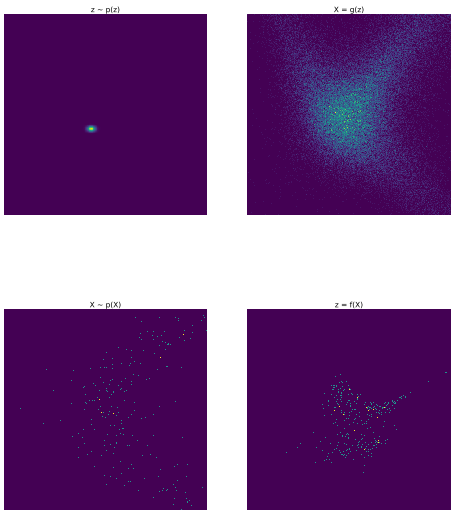

Figure 2: The results of the Real NVP Flow for dataset boomerang.

The ODE flow is characterized by the evolution of a simple probability distribution of a variable $z(t)$ (given as $p(z(t_0))$), into a complex distribution (given as $p(z(t_1))$), as time progresses. This is given by the equation of instantaneous change of variables,

$$\log p(z(t_1)) = \log p(z(t_0)) - \int_{t_0}^{t_1} \mathrm{tr}\left(\frac{dg}{dz(\tau)}\right) d\tau \tag{10}$$

for the ODE system $z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(t, z(\tau))d\tau$ where $g(z(t), t) = \frac{dz}{dt}$ is the continuous time transformation function. Here the determinant is replaced by the trace operator tr given as,

$$\frac{\partial \log p(z(t))}{\partial t} = -\mathrm{tr}\left(\frac{dg}{dz(t)}\right),$$

The trace function considers the instantaneous change of the distribution volume in time. Unlike the Jacobian determinants previously used for coupling flows, this is not a deterministic function.

To train the ODE flow, one treats samples from the target distribution $x \sim p(x)$ as $z(t_1)$. Backward solutions $\log p(z(t_0))$ and $\frac{\partial \log p(z(t_0))}{\partial t}$ are obtained using a neural network (defining the forward transformation f) and an ODE integral solver to integrate all time steps in between two states. Its density - $\log p(z(t_1))$ is evaluated using the equation above, and minimized over the network parameters.

4.1 Architecture

The continuous time transformation function is parameterized by a neural network with 3 fully connected linear layers with 32 hidden layer units and Tanh non-linearities. A width parameter of value 64 is also defined to restructure and train the flow pararmeters. The output has 2 units.

For training, an Adam optimizer was chosen with a learning rate of 3e-3 for batches of size 32. Two time steps were considered for the flow. The training was carried out for 100 epochs upon emperical evaluation taking around 1.5 gpu hours to complete for each dataset.

4.2 Results

The results for the three target samples are shown below.

4.3 Advantages and disadvantages of CNF over coupling flows

CNFs are more expressive than coupling flows. CNF takes 100 epochs to train as apposed to RealNVP which takes 2000 epochs. This is seen in how there is no issue of mirroring structures like in RealNVP.

CNFs are smoother than RealNVP. Since the flow transformation is over a continuous time domain, one can theoretically solve for infinite time steps between two states. On the other hand,

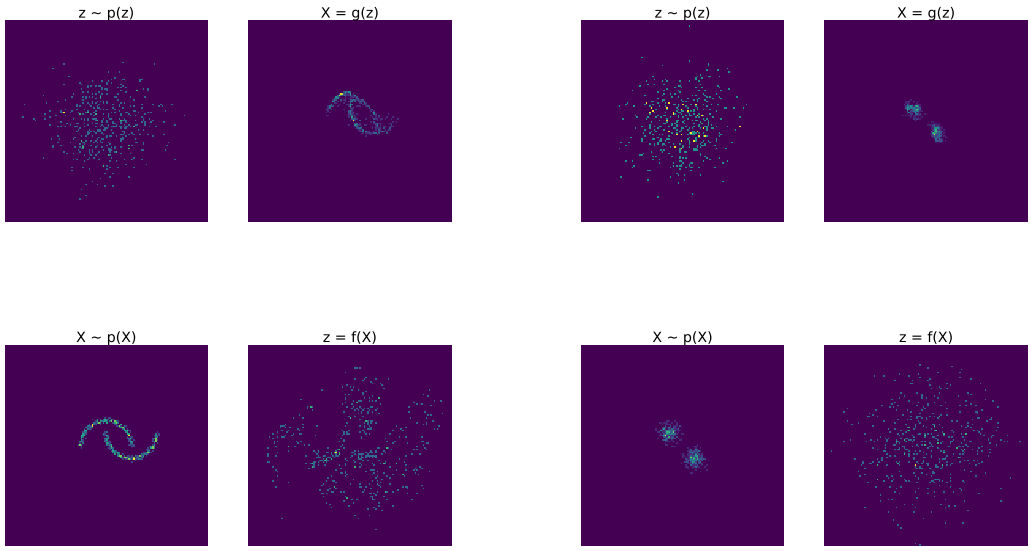

Figure 3: The results of the CNF Flow for datasets two moons and two blobs.

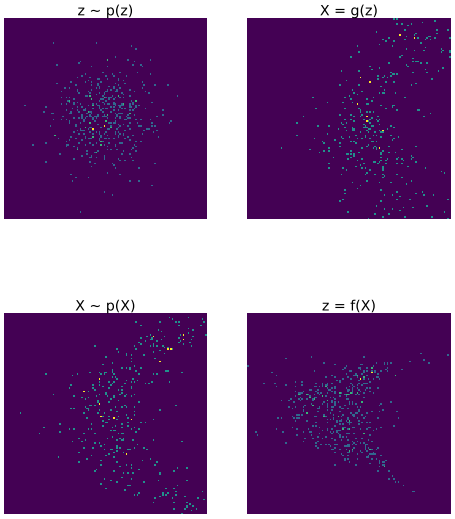

Figure 4: The results of the CNF Flow for dataset boomerang.

RealNVP takes discrete steps. Also, CNF has access to every single time step in the transformation, which is not the case in RealNVP.

On the flip side, because CNF uses an ODE integral solver to integrate the ODE system for each pass through the flow, every epoch takes much longer compared to RealNVP.

References

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Ricky TQ Chen, Y Rubanova, J Bettencourt, and D Duvenaud. Neural ordinary differential equations. arxiv 2018. *arXiv preprint arXiv:1806.07366*.