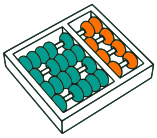

Flow Models

Lesson No. 03

Samuel Felipe Chenatti

1 Motivation

Most of the real life applications of Machine Learning involves dealing with continuous data. In these scenarios, since most of popular ML models only works with discrete inputs, one should discretize the data before using it. Flow models are Likelihood models designed to learn continuous data distributions while being efficient for sampling and evaluation/inference.

2 Flows

Given a true, unkwnown data distribution $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$, a flow $f_\theta(.)$ is a function that maps from the data $\mathbf{x}$ to noise $\mathbf{z}$ by applying a series of transformations to p_{data}.

So as we have done before, our task is to learn the likelihood of the underlying data distribution. The main difference is that instead of trying to directly approximate it with through a parametrized probability density p_θ, we learn the Flow f_θ that maps $\mathbf{x}$ to noise $\mathbf{z}$ from a base distribution $z \sim p(\mathbf{z})$.

A good and simple example that ilustrates the concept of a flow is the *Cumulative Density Distribution*. Given we have found a distribution p_θ that fits the data, we know by probability theory that the inverse of its CDF (equation 1) f_θ^{-1} is defined in $(0,1) \mapsto \mathbb{R}^D$, where $\mathbb{R}^D$ is $mathbf X$ domain. We then can sample a noise value $\mathbf{z} \sim U(0,1)$ and plug it on f_θ^{-1} to sample data in the feature space. For efficiently sampling data from the inverse CDF, f_θ **should be easily invertible**. The core idea here is to **learn the Flow** $f_\theta(.)$ instead of learning p_θ.

Note that now p can come from any family of distributions as long as it is easy to sample from. We dont need to know nothing about the real distribution to pick a *base distribution*. In a Reinforcement Learning applications, for example, we tend to use a spherical gaussian to model the policy function π_θ (which maps states to the probability of and agent executing some action) since it is simplier to be modeled by a Neural Network. But we now in reality that agents like robots does not necessarily presents a normal distribution over actions/actuation. We could instead learn a flow that maps the unkwnown distribution to a gaussian.

$$f_\theta(x) = \int_{-inf}^{x} p_\theta(t)dt \tag{1}$$

3 Change of variables

Before learning how to train a Flow we need to review how to the change of variables work in probability theory.

Let X be the distribution $U(0,1)$ and take $Y = f(X) = 2X + 1$. Y is another random variable, defined as an affine transformation of the underlying distribution X. So for every $x \in X$ we can get a $y \in Y$ by applying the transformation f to it. Since $0 \leq x \leq 1$, we have that $1 \leq y \leq 3$. We can see in figure 1 that in order to keep Y integrable to 1, since we expanded the domain by 2, we need to divide the probability density by 2 over the entire domain.

This area preservation can be represented mathematically by equation 2. For high dimensional cases we want to preserve the volume so we generalize $\frac{|dz|}{|dx|}$ to be the Jacobian seem in equation 3.

$$p(x)|dx| = p(z)|dz| \implies \frac{p(x)}{p(z)} = \frac{|dz|}{|dx|} \tag{2}$$

$$\frac{p(x)}{p(z)} = \left| \frac{f_\theta(x)}{\partial x} \right| \tag{3}$$

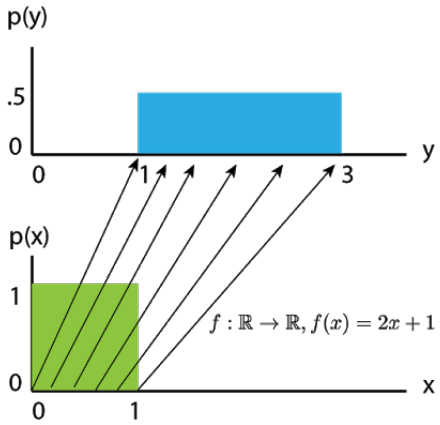

Figure 1. Geometrical representation of X and Y distributions. Adapted from [1]

4 Training Flows

By training flows we are dealing with data independently and identically distributed sampled from the underlying distribution, so we can still use the Maximum Likelihood framework. To do so, we can use the relations of equation 2 to achieve equation 4. Note that now it is a requirement that the **determinant of the Jacobian computation should be cheap** for allowing fast training.

$$\text{argmin}_\theta E_{\mathbf{x}}[-\log p_\theta(\mathbf{x})] = E_{\mathbf{x}}\left[-\log p(f_\theta(x)) - \log\det\left|\frac{f_\theta(x)}{\partial x}\right|\right] \tag{4}$$

Flows can also be composed by stacking transformations in order to increase the model expressiviness. For a composition of k flows, equation 2 becomes 5.

$$\log p_\theta = \log p(f_\theta(x)) + \sum_i^k \left|\frac{\partial f_\theta(x)}{\partial x}\right| \tag{5}$$

Untill now we have considered that Flows were applied to the feature vector $\mathbf{X}$ as whole, but we can instead compute the flows in a element-wise manner, considering each of the data dimensions as a single random variable. This technique allow us to build invertible flows, as we are going to see in the next section.

5 Affine transformations and RealNVP

In ReanNVP[2] we use a simple element-wise affine transformation as our Flow (which is also the case for other Deep Flow Models, like NICE[3]). Taking an image as an example, the process consist of splitting its pixels into two groups $\mathbf{x}_{1:d/2}$ $\mathbf{x}_{d/2+1:d}$ following a checkerboard ordering for the pixels and the channels.

The foward transformation is then given by equation 6, and its inverse (taking $\mathbf{z}$ into $\mathbf{x}$) is given by 7.

In both equations, s_θ and t_θ are parametrizable functions that outputs scale and translation constants, so they don't need to be invertible and can easily be modeled to be any complex function, such as Neural Networks.

$$f(\mathbf{x}) = \begin{cases} \mathbf{z}_{1:d/2} = \mathbf{x}_{1:d/2} \\ \mathbf{z}_{d/2:d} = \mathbf{x}_{d/2:d} \odot s_\theta(\mathbf{x_{1:d/2}}) + t_\theta(\mathbf{x}_{1:d/2}) \end{cases} \tag{6}$$

$$f^{-1}(\mathbf{z}) = \begin{cases} \mathbf{x}_{1:d/2} = \mathbf{z}_{1:d/2} \\ \mathbf{x}_{d/2:d} = (\mathbf{z}_{d/2:d} - + t_\theta(\mathbf{z}_{1:d/2})) \odot \exp(s_\theta(\mathbf{z}_{1:d/2})) \end{cases} \tag{7}$$

The Jacobian of the Flow in equation 6 is also tractable, as we can see in equation 8. It is easy to demonstrate this result by taking every $x \in \mathbf{X}$ and $z \in \mathbf{Z}$ and computing its partials for a 4x4 sized feature matrix.

Since the upper-right elements are all 0 and the bottom-right elements form a diagonal matrix, the determinant of the Jacobian is simply the product of the bottom-right matrix, resulting in $\exp\left[\sum_j s_\theta(x_{1:d})j\right]$.

$$\frac{\partial \mathbf{Z}}{\partial \mathbf{X}} = \begin{bmatrix} \mathbb{I} & \mathbf{0} \\ \frac{\partial \mathbf{Z}_{d/2:d}}{\partial \mathbf{X}_{d/2:d}} & \text{diag}(s_\theta(\mathbf{X}_{1:d/2})) \end{bmatrix} \tag{8}$$

6 Conclusion

The study of Flows as an alternative for unsupersived learning is in its first steps (in fact, most of the great results appeared in the last year) and has showed promising results. There are many ways to explore its potential, such as increasing its the expressiviness by using non-linear transformations for achieving better performance. This is the case for the Flow++[4] model, which uses CDFs and inverse CDFs for Mixture of Gaussian in the transformations. The recent advances in Flows also explores more complex Neural Networks architectures such as Gatted CNNS, Invertible 1x1 Convolutions [5] and attention blocks[4].

References

[1] E. Jang, "Normalizing flows tutorial, part 1: Distributions and determinants," *https://blog.evjang.com/*, 2018. [Online]. Available: https://blog.evjang.com/2018/01/nf1.html.

[2] L. Dinh, J. Sohl-Dickstein, and S. Bengio, *Density estimation using real nvp*, 2016. arXiv: 1605.08803 [cs.LG].

[3] L. Dinh, D. Krueger, and Y. Bengio, "NICE: non-linear independent components estimation," in *3rd International Conference on Learning Representations, ICLR 2015,San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[4] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, *Flow++: Improving flow-based generative models with variational dequantization and architecture design*, 2019. arXiv: 1902.00275 [cs.LG].

[5] D. P. Kingma and P. Dhariwal, *Glow: Generative flow with invertible 1x1 convolutions*, 2018. arXiv: 1807.03039 [stat.ML].