Notes on Real-valued Non-volume Preserving Transformations

Nils Hallerfelt

October 2022

Implicit models

In an implicit generative model we have a predefined latent space p_z where sampling is done, and then a deterministic transformation is done from the latent space to the observed space:

$$(1)\ z \sim p_Z(z) \quad (2)\ x = g(z).$$

In order to train the model using the likelihood of the observed data and inference, the change of variable theorem can be utilized:

$$p_X(x) = p_Z(z)|det(\mathbf{J}g(z))|^{-1}$$

Where $\mathbf{J}g(z)$ is the Jacobian of the function g. Assuming that g is an invertible function, so that $z = g^{-1}(x) = f(x)$, we can rewrite:

$$p_X(x) = p_Z(f(x))|det(\mathbf{J}f(x))| \rightarrow \log p_X(x) = \log p_Z(f(x)) + \log|det(\mathbf{J}f(x))|$$

The log-likelihood is tractable assuming certain structures on the function f - in general the computation of determinants is $O(n^3)$, however functions with triangular Jacobian can be computed in linear time.

Designing f

By setting f up in the following way we get the desired properties:

$$\begin{aligned} z_{1:n} &= x_{1:n} \\ z_{n+1:N} &= x_{n+1:N} \odot \exp s(x_{1:n}) + t(x_{1:n}). \end{aligned}$$

Here we let one part of the input vector be "copied" to the output, while the second half gets updated with a translation and scaling. The function f is obviously easily invertible, and the Jacobian is computable:

$$\mathbf{J}f(x) = \begin{bmatrix} \mathbb{I}_n & \mathbf{0} \\ \frac{\partial z_{n+1:N}}{\partial x_{1:d}^T} & diag(\exp s(x_{1:n})) \end{bmatrix}$$

$$\rightarrow$$

$$|det(\mathbf{J}f(x))| = \prod_{i=1}^{n} \exp s(x_i)$$

$$\rightarrow$$

$$\log |det(\mathbf{J}f(x))| = \sum_{i=1}^{n} s(x_i)$$

and thus the determinant can be easily evaluated. We notice that the the evaluation of the determinant is not affected by the translation t and the scaling s, which can be arbitrary functions. In our implementation we let s and t be parameterized by neural networks so that they are trainable. Each function f here can thus be seen as a layer in out neural network, in turn consisting of two neural nets s and t.

Implementing using Masked Convolution

The partitioning function f can be implemented using a binary mask b. In this implementation a binary vector where the first n entries are ones and the rest zeros ore vice versa is used. This way f can be implemented in functional form in the following way:

$$z = b \odot x + (1 - b) \odot (x \odot \exp s(b \odot x) + t(b \odot x))$$

Layering

In the above sections we examined one layer in our model, where only part of the input vector was operated on. In order to use the complete input effectively, we can use multiple alternating layers (alternating in the sense that the first layer applies s_1, t_1 on the upper part of the input, and the second layer applies s_2, t_2 on the lower part, etc.). This way we make the model operate on the complete input. With some abuse of notation from the above, we let f_i denote the function from one layer, and f denote the complete model, i.e. the neural network.

$$f(x) = f_L(...f_i(...f_1(x)...)...)$$

The determinant of the Jacobian of the compound f remains tractable, relying on the chain rule together with the fact that the determinant of a product is the product of determinants, each of which is easy to compute. The inverse also remains easy to compute, since we can compute the inverse of each f_i separately:

$$(f_1 \circ ... \circ f_i \circ ... \circ f_L)^{-1} = f_1^{-1} \circ ... \circ f_i^{-1} \circ ... \circ f_L^{-1}$$

Flow models

Compared to other common generative models, such as the Restricted Boltzmann Machine (RBM) and Variational Autoencoder (VAE), the inference in Flow models work differently. From the above description of the

model, notice the absence of Bayes Theorem and the difficult marginalization that comes with it. In RBM the marginalization over latent variables leads to an intractable loss function (the negative log-likelihood of the data). The problem can be solved in multiple ways, however they all (as far as my knowledge extends) depend on approximation techniques based on MCMC or similar sampling techniques as well as Mean Field Assumptions. This hurts the results in the optimization as well as introduces difficulties in convergence. To work around the problem of intractable log-likelihood in the training of VAE:s a lower bound is constructed and maximized, often combined with the reparameterization trick. However this adds some noise to the model. If the reparameterization cannot be applied the VAE instead relies on Score Function Estimation and BASELINE (REINFORCE). The Score Function Estimation comes with other problems as it is an estimation technique - and is of course not exact.

The Flow model proposes a way to use exact inference in the training, making the training process more stable and more likely to converge. Flow models also allow for exact and efficient sampling from the resulting model since the prior can be chosen such that it is easy to sample from and the function g determined during the training is deterministic in nature. The hardship comes with function g, that must be designed so that the training is efficient. Also, as g should be bijective the latent space of Flow models are usually very high dimensional, making them hard to interpret and not as intuitive as they can be for VAE:s and RBM:s. The restrictions on g also means that it is harder to get a very expressive model.