Density Estimation using RealNVP

Generative Models:

1. GANs
2. Likelihood based Generative Models
    a. Variational Autoencoders
    b. Autoregressive Models
    c. Normalising Flow Models

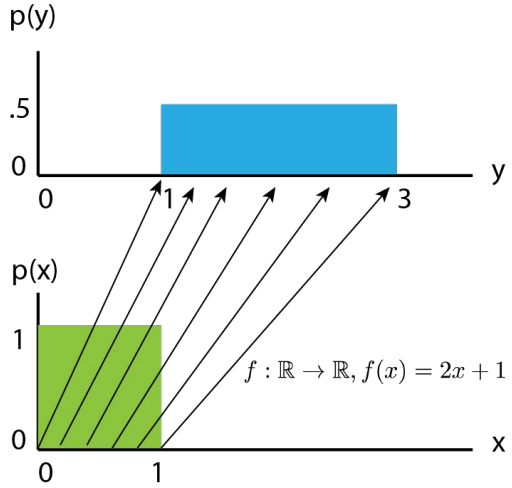
Some Revision :

1. Change of Variables:

   p(x)=p(y)(dy/dx)

   Takeaway ?

   a. Change in spread
   b. Jacobian for higher dims



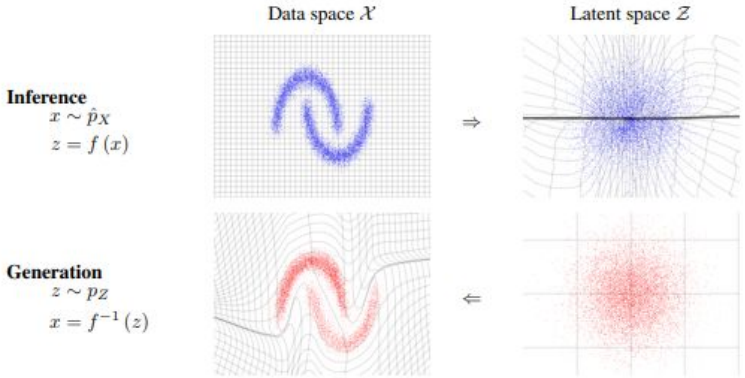
2. Likelihood Estimation:

$$L(\mu, \sigma^2; x_1, \ldots, x_n) = \prod_{j=1}^{n} f_X(x_j; \mu, \sigma^2)$$

If we know the data and the mean and std then we can estimate how likely that data belongs to that distribution.

Normalising Flow Models:

1. Map the input data into a prior distribution
2. Learn an invertible function which can transform input distribution into the prior and its inverse does the opposite effectively.
3. The idea is very simple, they propose to use the change in variables in distribution functions and maximize the log likelihood using this change of variables formula.



Idea and Implementation:

Goal: Learn $P_X(x)$ using maximum likelihood estimation. But how do we compute likelihood of the data without knowing anything about the distribution??

Here comes the role of the Latent Distribution.

Given an observed data variable $x \in X$, a simple prior probability distribution p_Z on a latent variable $z \in Z$, and a bijection $f : X(\text{inp}) \rightarrow Z(\text{prior})$ (with $g = f^{-1}$), the change of variable formula defines a model distribution on X by

$$p_X(x) = p_Z(f(x)) \left| \det \left(\frac{\partial f(x)}{\partial x^T} \right) \right| \tag{2}$$

$$\log(p_X(x)) = \log\left(p_Z(f(x))\right) + \log\left(\left|\det\left(\frac{\partial f(x)}{\partial x^T}\right)\right|\right), \tag{3}$$

where $\frac{\partial f(x)}{\partial x^T}$ is the Jacobian of f at x.

How is the function modeled through NN?

Let input has d dims then a series of transformations are applied to the input such that in each transformation some dimensions are preserved but some are transformed. Let input be D dims and output of first transformation is given by:

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

$$y_{d:D} = ß * x_{d:D} + Ɛ$$

The proposed idea to use the transformation parameters ß,Ɛ using the neural network which have input as the unchanged elements of the input vector. This is only what one transformation does, for each transformation they use different masking pattern. By doing this they reduced the Jacobian to a triangular matrix whose determinant can be easily found by multiplying the diagonal elements.

How does the Jacobian work for series of operations?

The Jacobian determinant of the resulting function remains tractable, relying on the fact that

$$\frac{\partial(f_b \circ f_a)}{\partial x_a^T}(x_a) = \frac{\partial f_a}{\partial x_a^T}(x_a) \cdot \frac{\partial f_b}{\partial x_b^T}(x_b = f_a(x_a)) \tag{10}$$

$$\det(A \cdot B) = \det(A)\det(B). \tag{11}$$

Similarly, its inverse can be computed easily as

$$(f_b \circ f_a)^{-1} = f_a^{-1} \circ f_b^{-1}. \tag{12}$$

Observation:
The idea to keep this function tractable is done by using the scaling parameter as exponential term so that when the determinant is taken the power gets added up for each transformation and finally when log is applied the jacobian can be estimated easily by taking the sum of the powers of the scaling term.

Normalizing flows are invertible. How do you make a neural net invertible?:

So just by scaling and adding a translation term to the input for each transformation we get the transformed points of the new distribution. But the converse is also feasible which is the beauty of the idea that it learns a bijective function

- Forward Transformation: $y_{1:d} = x_{1:d}$

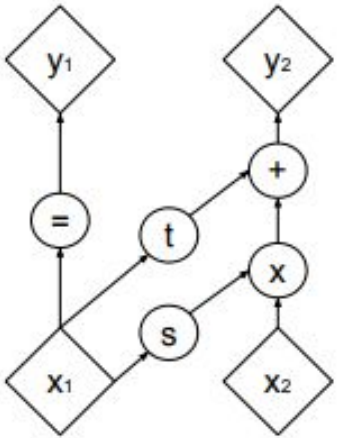
  $y_{d:D} = ß * x_{d:D} + Ɛ$

- Reverse Transformation: $x_{1:d} = y_{1:d}$

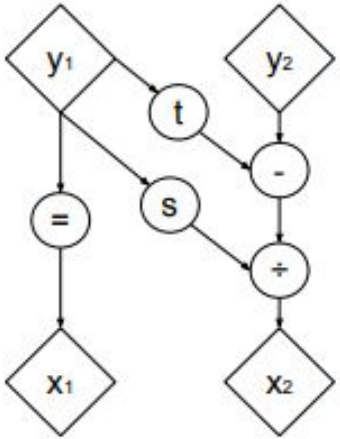
  $x_{d:D} = ß * (y_{d:D} - Ɛ)$

If we see carefully that the input to the NN during the forward transformation is same as that during the reverse transformation which leads to the generation of the same parameters while performing the reverse transformation.

Coupling Layer:



(a) Forward propagation



(b) Inverse propagation

Objective Function:

$$\log\left(p_X(x)\right) = \log\left(p_Z\left(f(x)\right)\right) + \log\left(\left|\det\left(\frac{\partial f(x)}{\partial x^T}\right)\right|\right),$$

- After a series of transformations are done we generate the output as f(x).
- We already calculated the jacobian the above is what we want to maximize.
- The negative of the above equation is the equation which we want to minimize to maximize the log likelihood of the input data.

Multi Scale Architecture for high dimensions:

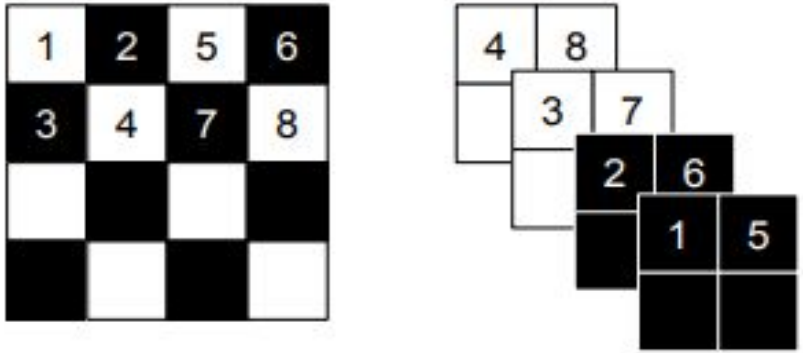
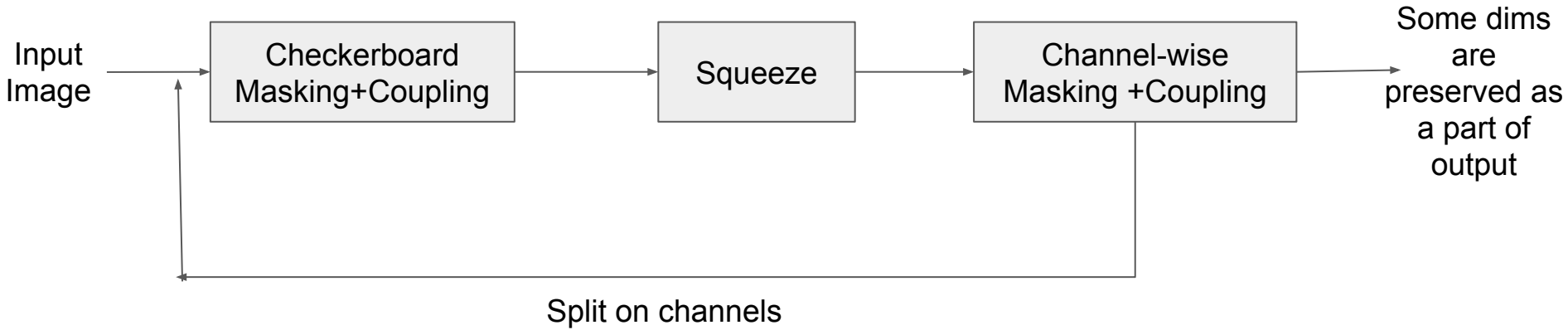


Figure 3: Masking schemes for affine coupling layers. On the left, a spatial checkerboard pattern mask. On the right, a channel-wise masking. The squeezing operation reduces the $4 \times 4 \times 1$ tensor (on the left) into a $2 \times 2 \times 4$ tensor (on the right). Before the squeezing operation, a checkerboard pattern is used for coupling layers while a channel-wise masking pattern is used afterward.

Multi Scale Architecture for high dimensions:



Recursive approach is used by authors to reduce time and space complexity

Summary:

1. New Class of Generative models.
2. The very basic idea of change of variable is used to express likelihood of the input data.
3. Coupling layers act as a bijection between the input data and the latent space distribution.
4. Train using maximum likelihood
5. This approach can effectively transform distribution during both:
    a. Sampling (Input $\longrightarrow$ latent space)
    b. Inference (latent space $\longrightarrow$ input)