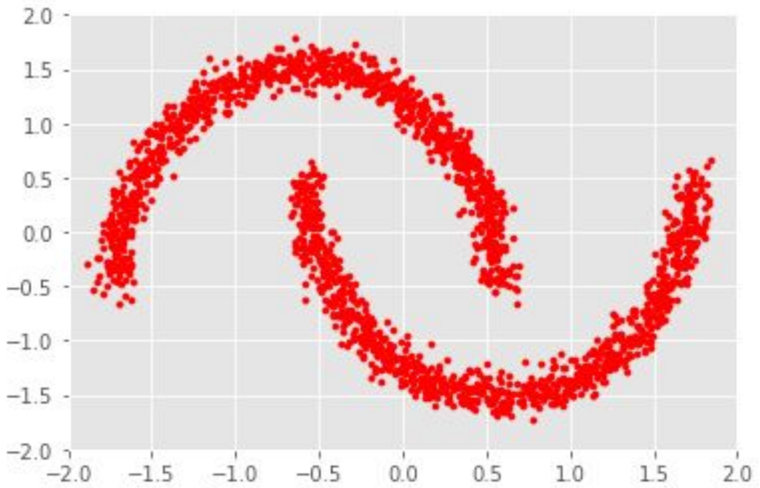
Normalizing Flows

Andrew H. Fagg
Symbiotic Computing Laboratory
University of Oklahoma

Goals

- Want to be able to represent an arbitrary probability distribution of examples in some domain
  - E.g., p(x): distribution of all possible images
  - Can only infer this distribution given a (large) set of examples
- Want to be able to:
  - Sample from this distribution
  - Construct realistic combinations of a set of examples

‘Half Moon’ Dataset



https://engineering.papercup.com/posts/normalizing-flows-part-1/

Transformations between Scalar Spaces

Approach:

- Assume a base distribution p(z) that is easy to represent and sample from:
  - E.g., p(z) ~ N(0,1)
- Construct a transformation for individual samples:
  - Generative direction: $x = f(z, \Phi)$
  - Must be invertible! Normalizing direction:

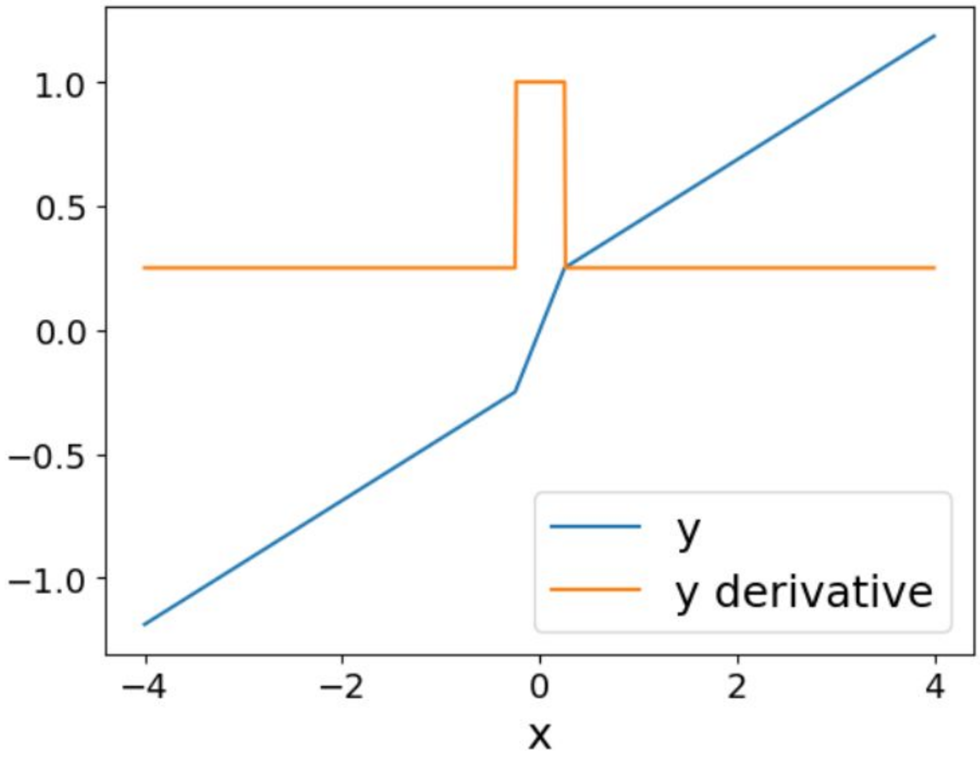
$$z = f^{-1}(x, \Phi)$$

Transformations between Scalar Spaces

Given: $x = f(z, \Phi)$

What is the relationship between p(x) and p(z)?

Example: Piecewise Linear Function



Transformations between Scalar Spaces

Given: $x = f(z, \Phi)$

What is the relationship between p(x) and p(z)?

$$p(x) = \left[\frac{\partial f(z)}{\partial z}\right]^{-1} p(z)$$

The inverse of the derivative counteracts the stretching that f() performs

Notebook …

Transformations between Vector Spaces

- Multiple dimensions (e.g., images)
- The Z and X spaces must have the same dimensionality (otherwise we cannot have an invertible function)
- Base distribution is still a standard Normal: $z \sim N(0, I)$

Transformations between Vector Spaces

x and q are now a vectors (assume both are dimensionality D):

$$x_i = f_i(z_0, z_1, ..., z_{D-1}, \Phi_i)$$

And:

$$x = f(z, \Phi) = \begin{bmatrix} f_0(z_0, z_1, \ldots, z_{D-1}, \Phi_0) \\ f_1(z_0, z_1, \ldots, z_{D-1}, \Phi_1) \\ \vdots \end{bmatrix}$$

Transformations between Vector Spaces

Given: $x = f(z, \Phi)$

What is the relationship between p(x) and p(z)?

Transformations between Vector Spaces

The Jacobian describes the local relationship between z and x:

$$J = \frac{\partial f(z, \Phi)}{\partial z} = \begin{bmatrix} \frac{\partial f_0()}{\partial z_0} & \frac{\partial f_0()}{\partial z_1} & \cdots \\ \frac{\partial f_1()}{\partial z_0} & \frac{\partial f_1()}{\partial z_1} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

Transformations between Vector Spaces

Given: $x = f(z, \Phi)$

What is the relationship between p(x) and p(z)?

$$p(x) = \left| \frac{\partial f(z, \Phi)}{\partial z} \right|^{-1} p(z)$$

The inverse of the determinant counteracts the stretching along all dimensions

Transformation Requirements

1. Expressive
2. Invertible
3. Inexpensive to compute inverse
4. Inexpensive to compute determinant of the Jacobian

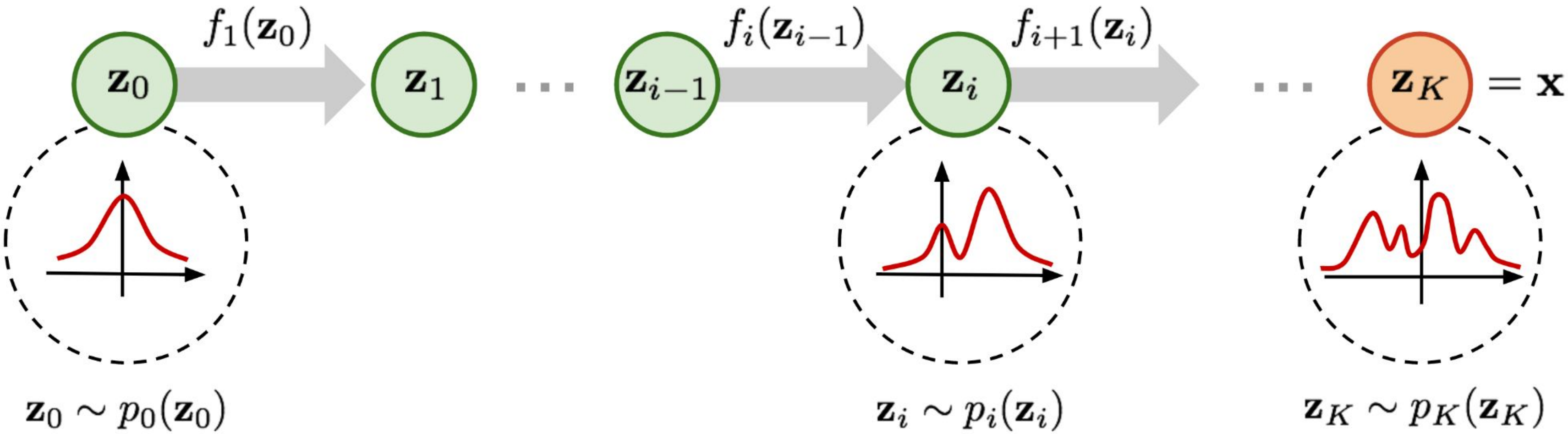
A general deep network is not guaranteed to satisfy these requirements.

Normalizing Flows

Approach:

- Construct a menu of simple transformation types
    - Individually, not very expressive
    - But: satisfy the other requirements
- Stack a sequence of these transformations together to achieve our needed expressive power

Stacking Simple Functions



Lilian Weng: https://lilianweng.github.io/posts/2018-10-13-flow-models/

Large Space of Options for Transformations

- Linear
- Elementwise non-linear
- Coupling
- Autoregressive

Permutation

$$z_{i+1} = Pz_i$$

where ***P*** is a permutation matrix (all zeros, except exactly one '1' in each column and row)

- Typically fixed & randomly generated
- Easy to invert
- Determinant is 1

Linear Flows

$$z_{i+1} = W^T z_i + \beta$$

- *W* and *β* are trainable parameters
- In the general form, inverting W or computing its determinant is O(n^3)
- For special forms of *W* (LU decomposed), inversion is O(n^2) & determinant computation is O(n)

Linear Flows

Can implement limited transformations on the pdf:

- Translation
- Scaling along individual dimensions
- Skewing
- Rotation

Will never change the number of modes in the original distribution

Elementwise Flows

For every element *j* in the z_i vector:

$$z_{i+1,j} = f_{i,j}(z_{i,j}, \Phi_i)$$

Where:

- $f_{i,j}(\ldots)$ is an invertible non-linear function
  - E.g., piecewise linear
  -

Elementwise Flows

$$z_{i+1,j} = f_i(z_{i,j}, \Phi_i)$$

- Inverse: compute inverse for each element - O(n)
- Determinant: product of the absolute derivatives - O(n)

Coupling Flows

- Split z_i into two pieces: $z_i^{(1)}$ and $z_i^{(2)}$
- First component is used to compute parameters $\Phi\left(z_i^{(1)}\right)$
- Second component is transformed:

$$z_{i+1}^{(2)} = g\left(z_i^{(2)}, \Phi\left(z_i^{(1)}\right)\right)$$

$$z_{i+1} = \begin{bmatrix} z_i^{(1)} \\ \\ z_{i+1}^{(2)} \end{bmatrix}$$

Coupling Flows

- Computation of the inverse and determinant is as complex as computing these for *g*()

$$z_{i+1}^{(2)} = g\left(z_i^{(2)}, \Phi\left(z_i^{(1)}\right)\right)$$

- Typically preceded by a permutation transform
  - This allows sorting of the individual elements into the parameter / value sets

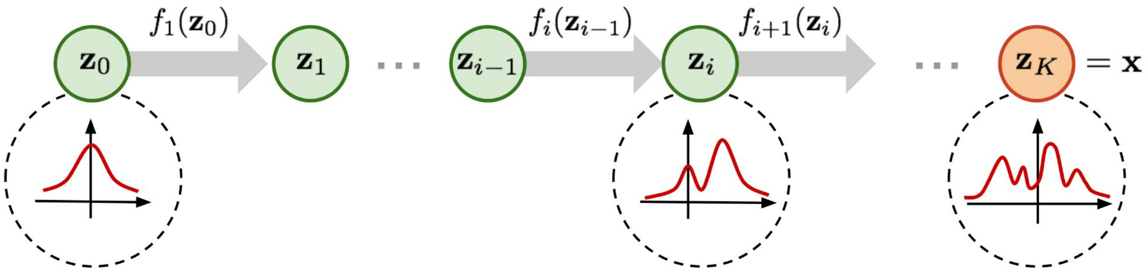
Inverse of a Flow

Computing the inverse of all steps:

- Sequentially evaluate individual inverses from right to left:

$$z_{K-1} = f_K^{-1}(x)$$

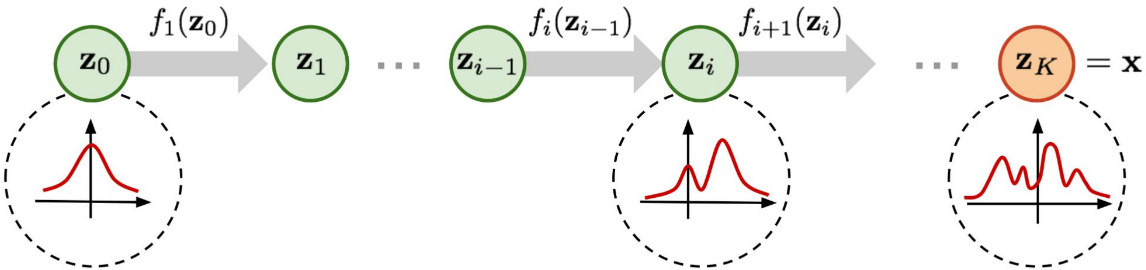
$$z_i = f_{i+1}^{-1}(z_{i+1})$$



Determinant of a Jacobian of a Flow

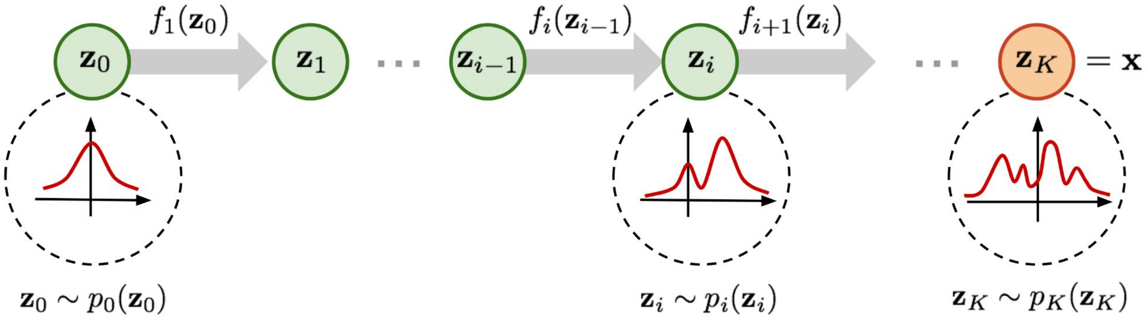
Product of the individual absolute determinants:

$$\left|\frac{\partial f(z)}{\partial z}\right| = \left|\frac{\partial f_K(z_{K-1})}{\partial z_{K-1}}\right| \times \left|\frac{\partial f_{K-1}(z_{K-2})}{\partial z_{K-2}}\right| \times \ldots \times \left|\frac{\partial f_1(z_0)}{\partial z_0}\right|$$



Training a Normalizing Flow

- Each (or most) f's have trainable parameters
- Given a set of samples, ***X***, we want to choose the set of parameters so we maximize the likelihood of these data



Training a Normalizing Flow

$$L = p(\mathbf{X}|\Phi)$$

Training a Normalizing Flow

$$
\begin{aligned}
L &= p(\mathbf{X}|\Phi) \\
&= p(x_0, x_1, ...p_{N-1}|\Phi)
\end{aligned}
$$

Training a Normalizing Flow

$$
\begin{aligned}
L &= p(\mathbf{X}|\Phi) \\
&= p(x_0, x_1, ...p_{N-1}|\Phi) \\
&= \prod_{i=0}^{N-1} p(x_i|\Phi)
\end{aligned}
$$

Training a Normalizing Flow

$$
\begin{aligned}
L &= p(\mathbf{X}|\Phi) \\
&= p(x_0, x_1, ...p_{N-1}|\Phi) \\
&= \prod_{i=0}^{N-1} p(x_i|\Phi) \\
&= \prod_{i=0}^{N-1} p(z_i|\Phi) \times \left| \frac{\partial f(z_i, \Phi)}{\partial z_i} \right|^{-1}
\end{aligned}
$$

Training a Normalizing Flow

$$L = \prod_{i=0}^{N-1} p(z_i|\Phi) \times \left| \frac{\partial f(z_i, \Phi)}{\partial z_i} \right|^{-1}$$

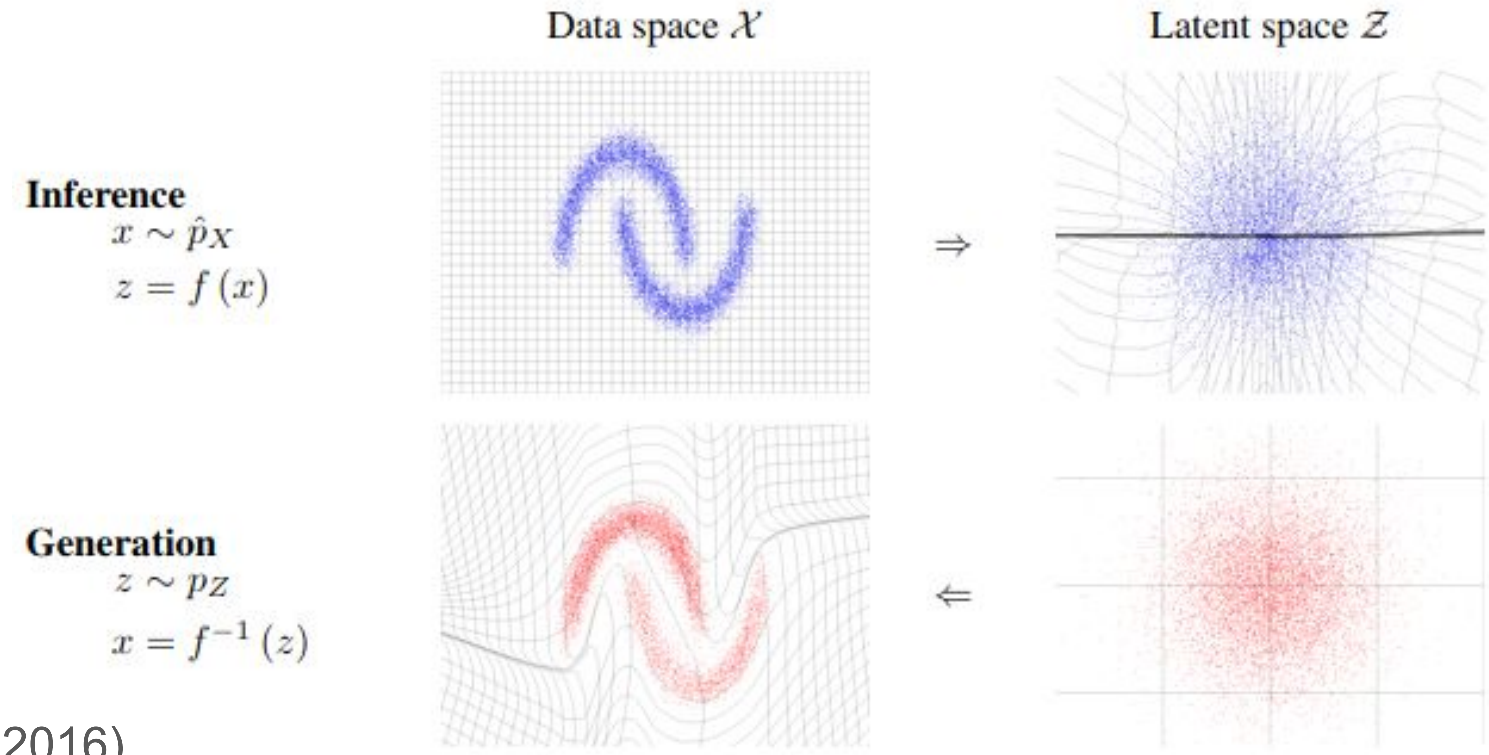
$$log\ L = \sum_{i=0}^{N-1} -log \left| \frac{\partial f(z_i, \Phi)}{\partial z_i} \right| + log\ p(z_i|\Phi)$$

Training a Normalizing Flow

$$log\ L = \sum_{i=0}^{N-1} -log\left|\frac{\partial f(z_i, \Phi)}{\partial z_i}\right| + log\ p(z_i|\Phi)$$

Follow the gradient: $\dfrac{\partial\, log\ L}{\partial \Phi}$

Half Moon: Learned Transformation



Dinh et al. (2016)

Interpolation

- Interpolation between two x's:
  - Transform each into the latent space
  - Compute the weighted average of the two
  - Transform the new z back into x space
- Because the base distribution is compact, we can have high expectation that the resulting image is a reasonable one

Face Interpolation Example

Fadal et al. (2021)

Summary

- Normalizing flows are all about transforming data between two spaces
  - One is easy to sample from & measure likelihoods in
  - The other can have a likelihood function that has a complex shape
- Transformation functions must be invertible
- Relative to GANs: can be more stable and easier to train