Machine Learning for Graphs and Sequential Data

Deep Generative Models

lecturer: Prof. Dr. Stephan Günnemann

Summer Term 20

Data Analytics and
Machine Learning

Roadmap

- Chapter: Deep Generative Models
  1. **Introduction**
  2. Normalizing Flows
  3. Variational Inference
  4. Generative Adversarial Networks

Generative Models

- Deterministic Generative Models
  - Image = Renderer(object=cube, color=red, size=, position=, …)
  - Image = Renderer(object=cylinder, color=blue, size=, position=, …)

- Statistical Generative Models

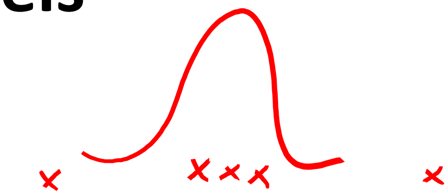
+

- Model family
- Loss function
- Optimization algorithm
- ….

$\overset{\text{learning}}{=}$ $p(\boldsymbol{x})$

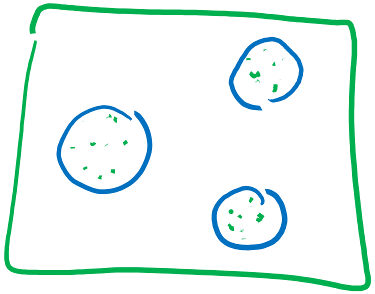
Data | Prior Knowledge | Probability Distribution

Desiderata for Statistical Generative Models



1. Efficient Sampling
   - Should be easy to sample a new instance $\boldsymbol{x}_{new} \sim p(\boldsymbol{x})$
   - Sampled/generated instances $\boldsymbol{x}_{new}$ should be similar to the training data

2. Efficient Likelihood Evaluation
   - Should be easy to evaluate $p(\boldsymbol{x})$ for any instance $\boldsymbol{x}$, e.g. $p($  $)$

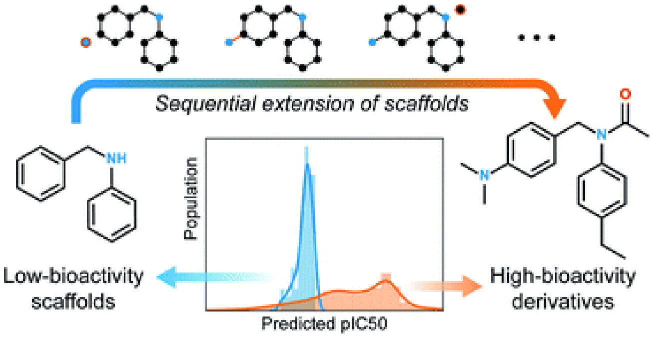


3. (Optionally) Extract Features
   - For any instance $\boldsymbol{x}$ extract latent features/representations
   - Capture/summarize the important aspects of the instance/image

Some Applications of Generative Models

- Image generation
  - https://www.thispersondoesnotexist.com/
  - https://www.youtube.com/watch?v=p5U4NgVGAwg
- 3D graphics & fluid dynamics
  - https://www.youtube.com/watch?v=i6JwXYypZ3Y
- Speech & music synthesis
  - https://deepmind.com/blog/article/wavenet-generative-model-raw-audio
- Drug discovery

[Achlioptas+, 2017]

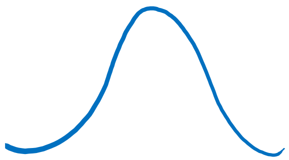


[Lim+, 2019]

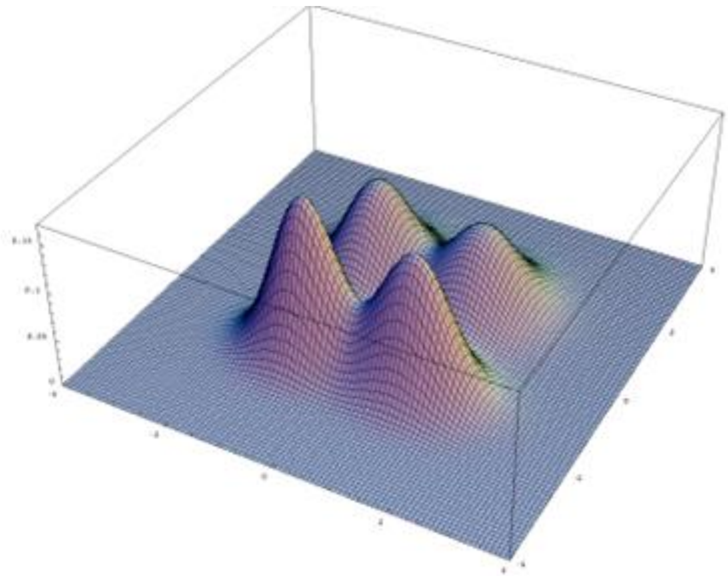
[Karras+, 2018]

Continuous Distributions over High-dimensional Data

- "Classic" probability distributions (e.g. multivariate normal) do not capture the complexity of real-world datasets

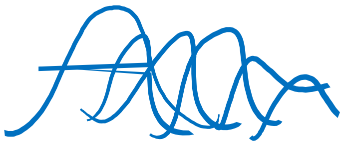


  - Real distributions are multi-modal, asymmetric

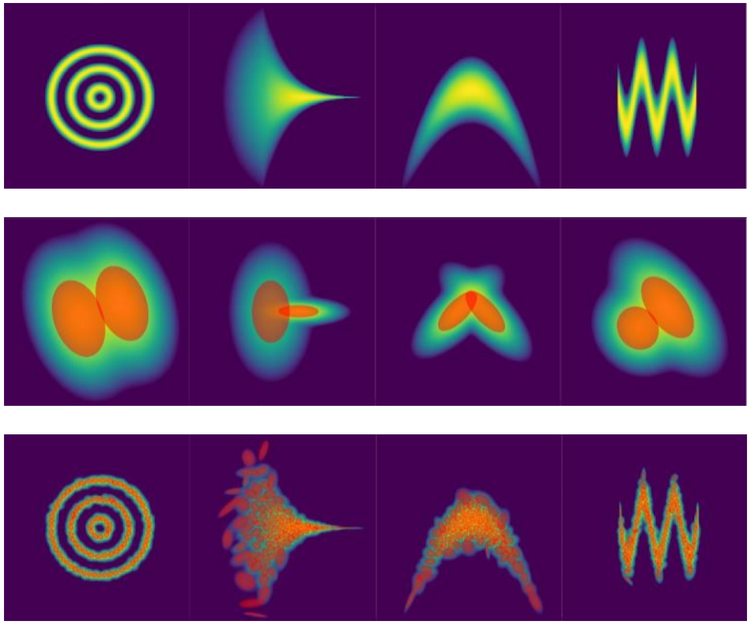


- Can we use mixture models to capture this behavior?

Mixture models



- In theory, a mixture with enough components can represent any density
  - How many is "enough"?



[Vergari+, 2019]

- Even for simple 2D densities we need hundreds of mixture components!
  - The situation gets (exponentially) worse as we increase the dimensionality

Discrete Distributions over High-dimensional Data

- What about discrete distributions?

- Suppose x_1, x_2, x_3 are binary variables
  - $p(x_1, x_2, x_3)$ can be specified with $2^3 - 1 = 7$ parameters
  - $p(0{,}0{,}0),\ p(0{,}0{,}1), \dots\ p(1, 1, 0),\ \cancel{p(1, 1, 1)}$

- For an image with N black or white pixels need to specify $2^N - 1$ values
  - The number of parameters grows exponentially with dimension

Challenges of High-dimensional Data

- "Classic" distributions
  - Do not capture the complexity of the data
- (Finite) mixture models
  - Require ridiculous amounts of parameters to specify even simple densities
  - Do not work in higher dimensions
- For discrete distributions – combinatorial explosion

- In this section, you will learn how to design flexible and efficient generative models for high-dimensional data using deep learning techniques

References

- Lim et al. 2019, Scaffold-based molecular design with a graph generative model
- Achlioptas et al. 2017, https://arxiv.org/abs/1707.02392
- Karras et al. 2019, https://github.com/NVlabs/stylegan
- Vegari et al. 2019, https://web.cs.ucla.edu/~guyvdb/slides/TPMTutorialUAI19.pdf

Roadmap

- Chapter: Deep Generative Models
  1. Introduction
  2. **Normalizing Flows**
     - **Change of Variable Formula**
     - Forward and Reverse Parametrization
     - Jacobian Determinant Computation
  3. Variational Inference
  4. Generative Adversarial Networks

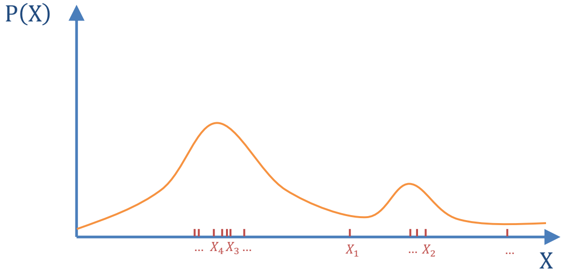
Motivation

- We assume that the data $\boldsymbol{x}$ follows a probability distribution $\mathrm{p}(\boldsymbol{x})$ i.e.

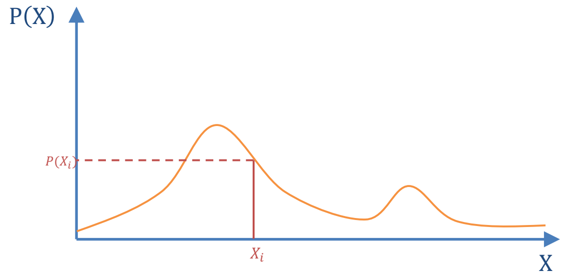
$$\mathrm{p}(\boldsymbol{x}) \text{ where } \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix}$$

- We can do two interesting things with a distribution:

Data sampling: generate data sample $\boldsymbol{x}_i$ following the distribution $\mathrm{p}(\boldsymbol{x})$



Density evaluation: given any $\boldsymbol{x}_i$, compute the probability density at this point $\mathrm{p}(\boldsymbol{x}_i)$



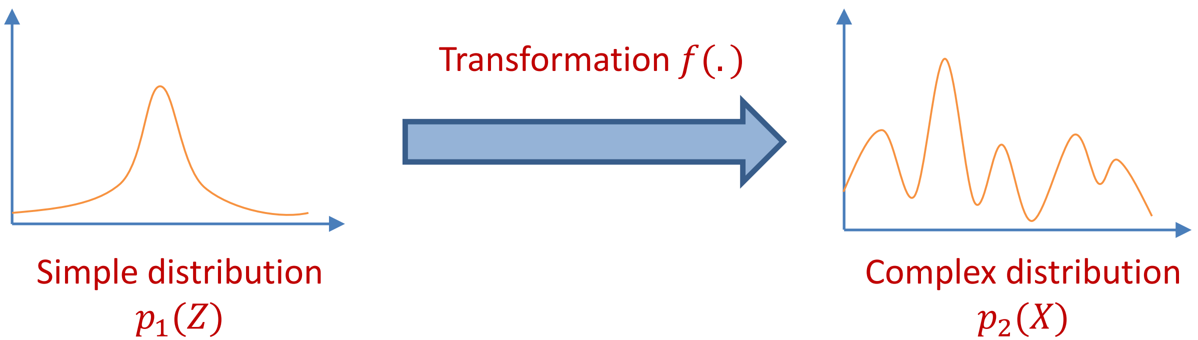
Motivation

- Normalizing Flows (NF) can model flexible distributions for *data sampling* and *density evaluation*.

- Normalizing Flows intuition:

Model a complex distribution

by applying a transformation on a simple distribution

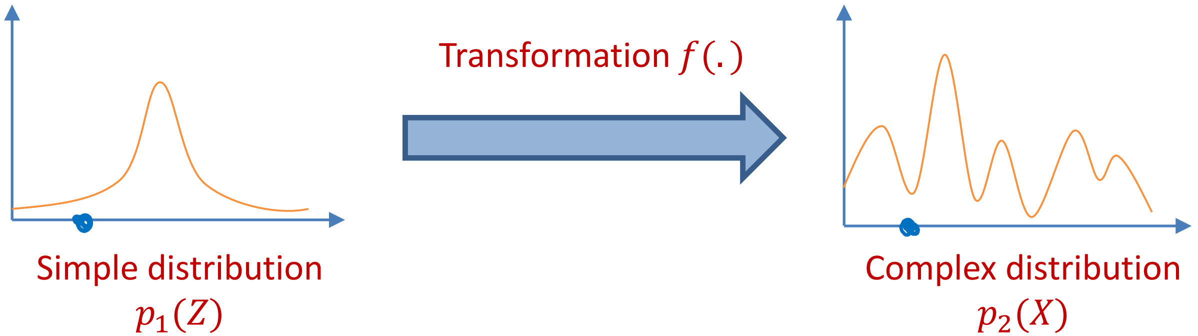


Idea

- Normalizing Flows are based on the change of variable formula

- It substitutes a variable $\boldsymbol{z}$ to another variable $\boldsymbol{x}$ by using a transformation function f i.e. $f(\boldsymbol{z}) = \boldsymbol{x}$

- It is particularly useful to simplify computations when working with distributions (or integrals)



Change of Variables: Example

$$\int_0^1 p_1(z)\,dz = 1 = \int_1^3 p_2(x)\,dx$$

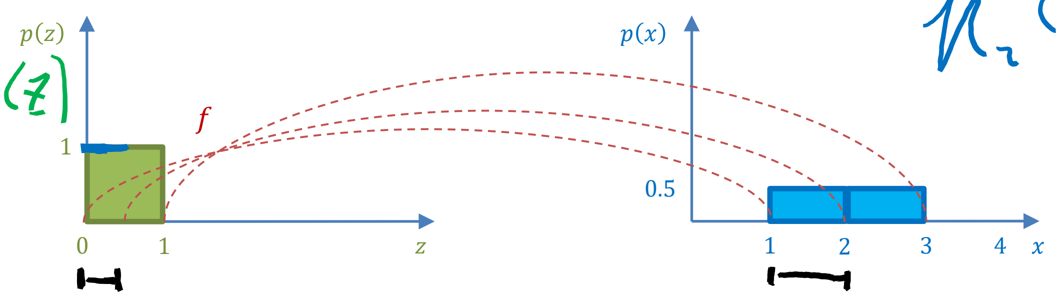
Introductory example: $D = 1, p_1(z) = Unif([0,1]), f(z) = 2z + 1 = x$

$f'(z) = 2$

- The probability in the input space should be the same as in the output space i.e. $p_1(z)\partial z = p_2(x)\partial x$

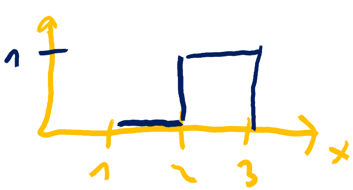
$$\frac{\partial f(z)}{\partial z} = f'(z)$$

$p_2(x) = Unif([1,3])$



$p_2(x) = p_1(z) \cdot \left|\frac{1}{2}\right|$

- Abusing the notations leads to $p_2(x) = p_1(z)\left|\frac{\partial z}{\partial x}\right|$ $= p_1(z) \cdot \left|\frac{\partial z}{\partial f(z)}\right|$
  - The term $\left|\frac{\partial z}{\partial x}\right|$ renormalize the probability distribution in the output space
  - The normalization term is the same if the transformation is increasing or decreasing

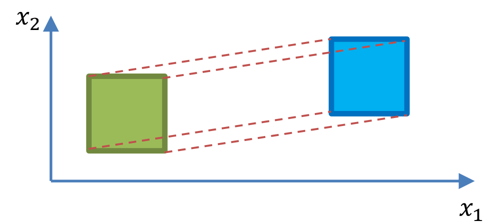


$$\int_0^{0.5} p_1(z)\,dz = \int_1^2 p_2(x)\,dx$$

Change of Variables: Example

Introductory example: $D = 2, p_1(\boldsymbol{z}) = Unif([0,1]^2), f(\boldsymbol{z}) = \boldsymbol{z} + shift = \boldsymbol{x}$

- Applying a constant shift does not change the area after the transformation i.e. $p_2(\boldsymbol{x}) = p_1(\boldsymbol{z})$



Introductory example: $D = 2, p_1(\boldsymbol{z}) = Unif([0,1]^2), f(\boldsymbol{z}) = M\boldsymbol{z} = \boldsymbol{x}, M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

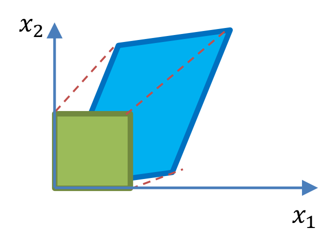
- The linear transformation M changes the area from 1 to $ad - bc = \det(M)$.
- The probability distribution $p_2(\boldsymbol{x})$ has to be normalized

$f^{-1}(x) = z = M^{-1} \cdot x$

i.e. $p_2(\boldsymbol{x}) = p_1(\boldsymbol{z}) \frac{1}{det(M)}$

$= p_1(z) \cdot det(M^{-1})$

$= p_1(z) \cdot \left| det \frac{\partial f^{-1}(x)}{\partial x} \right|$



$\frac{\partial f^{-1}(x)}{\partial x} = M^{-1}$

Change of Variable Formula

Change of variable formula (General case): if $D \in \mathbb{N}$, $\mathrm{p}_1(\boldsymbol{z})$ a D -dimensional distribution, $f(\boldsymbol{z}) = \boldsymbol{x}$ an *invertible* and *differentiable* transformation, then distribution $p_2(\boldsymbol{x})$ is

$$p_2(\boldsymbol{x}) = p_1(\underbrace{f^{-1}(\boldsymbol{x})}_{z}) \cdot \left|\det\left(\frac{\partial f^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|$$

- The determinant term accounts for the **distortion rate** of the transformation (see introductory examples). If it is equal to 1, $p_2(\boldsymbol{x})$ and $p_1\ (\boldsymbol{z})$ have the same « volume » at this point i.e. $p_2(\boldsymbol{x}) = p_1(f^{-1}(\boldsymbol{x}))$.
  - It considers that the transformation is locally linear (see last example)

- The term $\frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}}$ is called **Jacobian** of g; here: a $D \times D$ matrix
  - We have $\frac{\partial f^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \left(\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\right)^{-1}$.
- The transformation f should be ***valid*** *(invertible* and *differentiable).*

Conditions

(Sufficient) Conditions for a valid transformation f:

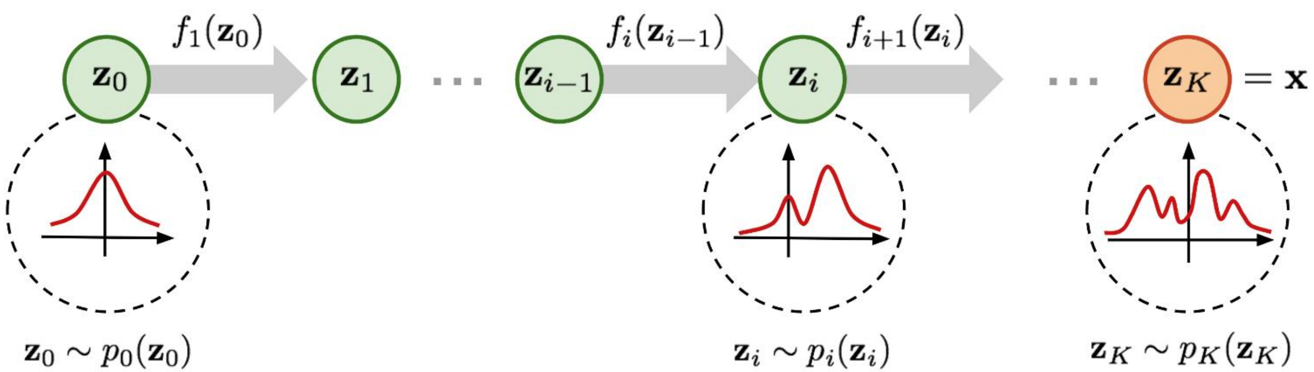
- *Invertibility:*
  - The input and output space of the mapping should have the **same dimension** $\boldsymbol{D}$.
  - If $\boldsymbol{D} = \mathbf{1}$, the mapping f should be **strictly monotonic** (increasing or decreasing).
  - If the transformation f is **linear**, its determinant should be nonzero i.e. $\mathbf{det}(\boldsymbol{f}) \neq \mathbf{0}$

- *Differentiability:*
  - *The mapping* f should be "smooth" i.e. the Jacobian $\frac{\boldsymbol{\partial f^{-1}(x)}}{\boldsymbol{\partial x}}$ **should exist**.
  - Note: Differentiability is a sufficient condition; in theory, the mapping f does not have to be differentiable everywhere, we can even have discontinuous; in practice we usually use only differentiable transformation

Stacking

Stacking transformations f_i:

- We can apply the change of variable formula multiple time:
  - The first variable z_0 is transformed by $z_1 = f_1(z_0)$
  - The variable z_{i-1} is transformed by $z_i = f_i(z_{i-1})$
  - The last variable z_{K-1} is transformed by $\mathbf{x} = z_K = f_K(z_{K-1})$



- The change of variable formula becomes:

[Lilian Weng blog]

$$p_K(\boldsymbol{x}) = p_0(\boldsymbol{z_0}) \prod_{i=1}^{K} \left| \det\left(\frac{\partial f_i^{-1}(\boldsymbol{z_i})}{\partial \boldsymbol{z_i}} \right) \right|$$

Change of Variable Formula: Log Version

Change of variable formula (log version): if $D \in \mathbb{N}$, $\mathrm{p}_1(\boldsymbol{z})$ a D-dim. distribution, $f(\boldsymbol{z}) = \boldsymbol{x}$ an *invertible* and *differentiable* transformation, then $p_2(\boldsymbol{x})$ is

$$\log(p_2(\boldsymbol{x})) = \log(p_1(f^{-1}(\boldsymbol{x}))) + \log\left|\det\left(\frac{\partial f^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|$$

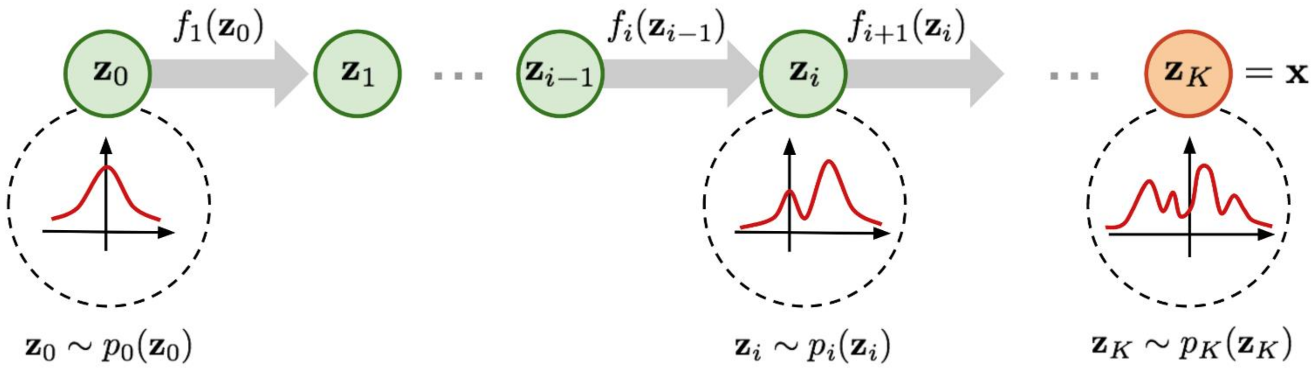
- Optimization is easier when working with sums. Consequently, we generally consider log probabilities (e.g. maximize log likelihood)

- The log version when stacking transformations is:

$$\log(p_2(\boldsymbol{x})) = \log(p_1(f^{-1}(\boldsymbol{x}))) + \sum_{i=1}^{K} \log\left|\det\left(\frac{\partial f_i^{-1}(\boldsymbol{z_i})}{\partial \boldsymbol{z_i}}\right)\right|$$

The name: "Normalizing flow"

- A NF transforms a simple distribution (e.g. uniform, Gaussian) in a complex distribution. For some Normalizing Flows the universality theorem has been proven.
- NFs stack valid transformations to model a complex mapping between the input and output space. The input variable **flows** through the transformations.
- The change of variable formula allows to compute the distribution of the output space based on the distribution in the input space. The determinant terms **normalize** the distribution in the output space.



Questions – NF1

1. Is $\mathrm{f}(z) = 1 - \mathrm{z}$ a valid transformation ?

2. Is $\mathrm{f}(z) = 2 - 3\,z$ a valid transformation ?

3. Is $\mathrm{f}(z) = \begin{cases} -z, z \in [0,1[\\ 1 - z, z \in [1,2] \end{cases}$ a valid transformation ?

Roadmap

- Chapter: Deep Generative Models
  1. Introduction
  2. **Normalizing Flows**
     - Change of Variable Formula
     - **Forward and Reverse Parametrization**
     - Jacobian Determinant Computation
  3. Variational Inference
  4. Generative Adversarial Networks

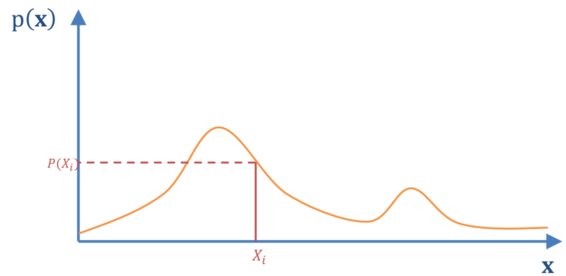
$f(z) = x$

Forward and Reverse Pparametrization

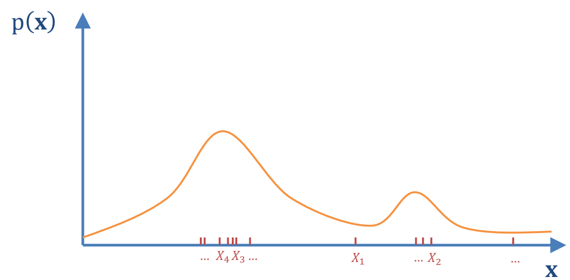
- The change of variable formula does a mapping between two distributions

$$p_2(\boldsymbol{x}) = p_1(\underbrace{f^{-1}(\boldsymbol{x})}_{z}) \cdot \left|\det\left(\frac{\partial f^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right| \longrightarrow \left(\frac{\partial f(z)}{\partial z}\right)^{-1}$$

- How to use the change of variable formula to evaluate $p_2(\boldsymbol{x})$ at any point $\boldsymbol{x}$?
  - **Reverse parametrization**



- How to use the change of variable formula to sample points from $p_2(\boldsymbol{x})$?
  - **Forward parametrization**



Forward and Reverse Parametrization

- There exist many different flows (see [3] for further details):
  - Planar/Radial flows
  - RealNVP
  - IAF
  - MAF
  - Spline
  - …

  Autoregressive Flows

- These flows generally differ in the parametrization of the transformation f.
  - Some parametrizations are efficient for sampling.
  - Some parametrizations are efficient for density evaluation.

- Efficient sampling and density evaluation (at the same time) might not be required in all applications

$g_\varphi(x) = w^T x + b$

Reverse Parametrization

- We parametrize the inverse transformation $g = f^{-1}$ that we know analytically.
  - $g_\varphi(\boldsymbol{x}) = \boldsymbol{z}$ is computable and parameter φ can be learned

$x \rightsquigarrow z$ ✓

- We know that the inverse $f = g^{-1}$ exists, but we might not know it analytically.
  - $g_\varphi^{-1}(\boldsymbol{z}) = \boldsymbol{x}$ might not be (easily) computable

$z \rightsquigarrow x$ HARD

- The change of variable formula with forward parametrization is

$$p_2(\boldsymbol{x}) = p_1(g_\varphi(\boldsymbol{x})) \cdot \left|\det\left(\frac{\partial g_\varphi(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|$$

  - The formula only uses the known parametrized function g_φ.
  - Given any point $\boldsymbol{x}^{(j)}$, we can compute $p_2(\boldsymbol{x}^{(j)})$.

Reverse Parametrization

Stacking:

- We can also stack transformations with reverse parametrization i.e.

$$g_\varphi = g_{\varphi_K} \circ \ldots \circ g_{\varphi_1}$$

- To **compute the density**
  1. For any $\boldsymbol{x}^{(j)}$, we can set $\boldsymbol{x}^{(j)} = \boldsymbol{z_K}$
  2. Compute the transformations $\boldsymbol{z}_{\boldsymbol{i-1}}^{(j)} = g_{\varphi_i}(\boldsymbol{z}_{\boldsymbol{i}}^{(j)})$ and $\left|\det\left(\frac{\partial g_{\varphi_i}\left(\boldsymbol{z}_{\boldsymbol{i}}^{(j)}\right)}{\partial \boldsymbol{z}_{\boldsymbol{i}}^{(j)}}\right)\right|$
  3. Given $\boldsymbol{z}_{\boldsymbol{0}}^{(j)}$, we can compute $p_0(\boldsymbol{z}_{\boldsymbol{0}}^{(j)})$ (e.g. Gaussian or Uniform) and thus $p_K(\boldsymbol{x}^{(j)})$

- Important recall: While $p_0(\boldsymbol{z})$ has a simple shape, $p_K(\boldsymbol{x})$ can capture very complex structure
  - This is all realized by g_φ

Normalizing Flows in Machine Learning

$g_\varphi(x) = w^T x + b$

PARAMETERS

- Why do we care about computing the density $p_K(x^{(j)})$?
- We can use it for learning!
  - We aim to learn g_φ , i.e. the transformation is not given but learned
  - ➢ We can call the distribution $p_\varphi(x)$ to make the dependency on φ clear

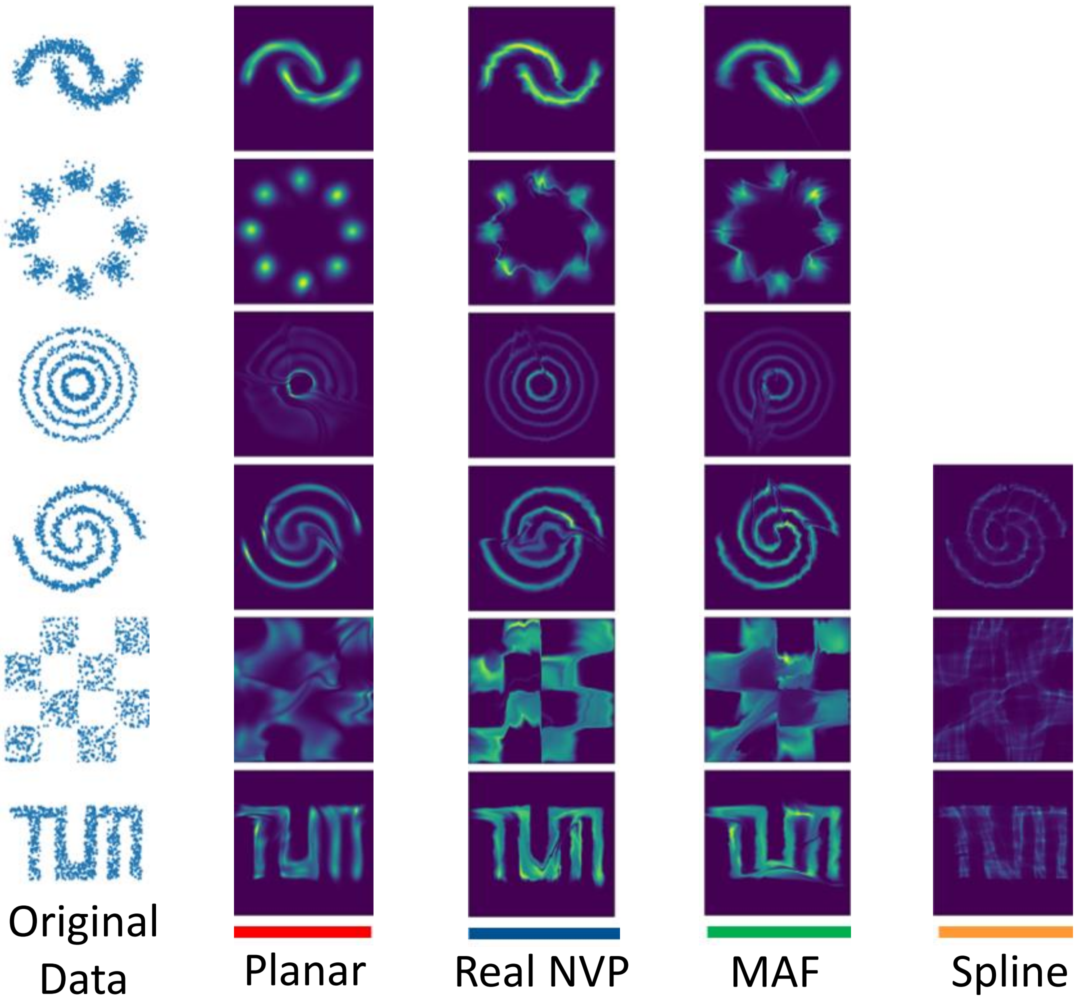
- Learning: Given a dataset $\mathcal{D} = \{\boldsymbol{x}^{(j)}\}_{j=1}^{N}$ (usually consisting of i.i.d. samples), find the parameters φ that best explain the data
  - Typically done by maximizing the marginal log-likelihood

$$\max_\varphi \log p_\varphi(\mathcal{D}) = \max_\varphi \frac{1}{N} \sum_{\boldsymbol{x}^{(j)} \in \mathcal{D}} \log p_\varphi(\boldsymbol{x}^{(j)})\textbf{?}$$

$D = \{x_i \in \mathbb{R}^2\}$

Learning with Normalizing Flows

- **Reverse parametrization (density estimation):**



*TUM Lab Course:
- Lukas Rinder
- Markus Kittel
- Murat Can

Forward Parametrization

$z \rightsquigarrow x$ ✓

- We parametrize the transformation f that we know analytically.
  - $f_\theta(\boldsymbol{z}) = \boldsymbol{x}$ is computable and parameter θ can be learned

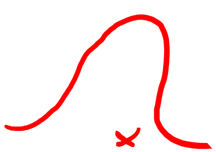
- We know that the inverse f^{-1} exists, but we might not know it analytically.
  - $f_\theta^{-1}(\boldsymbol{x}) = \boldsymbol{z}$ might not be (easily) computable

$x \rightsquigarrow z$ HARD

- The change of variable formula with forward parametrization is

$$p_2(\overset{f(z)}{\boldsymbol{x}}) = p_1(\boldsymbol{z}) \cdot \left| \det\left(\frac{\partial f_\theta(\boldsymbol{z})}{\partial \boldsymbol{z}} \right) \right|^{-1}$$

  - The formula only uses the known parametrized function f_θ.
  - Given a sample $\boldsymbol{z}^{(j)}$, we can compute a sample $\boldsymbol{x}^{(j)} \sim p_2(\boldsymbol{x})$ and $p_2(\boldsymbol{x}^{(j)})$.



Forward Parametrization

- We can also stack transformations with forward parametrization i.e.

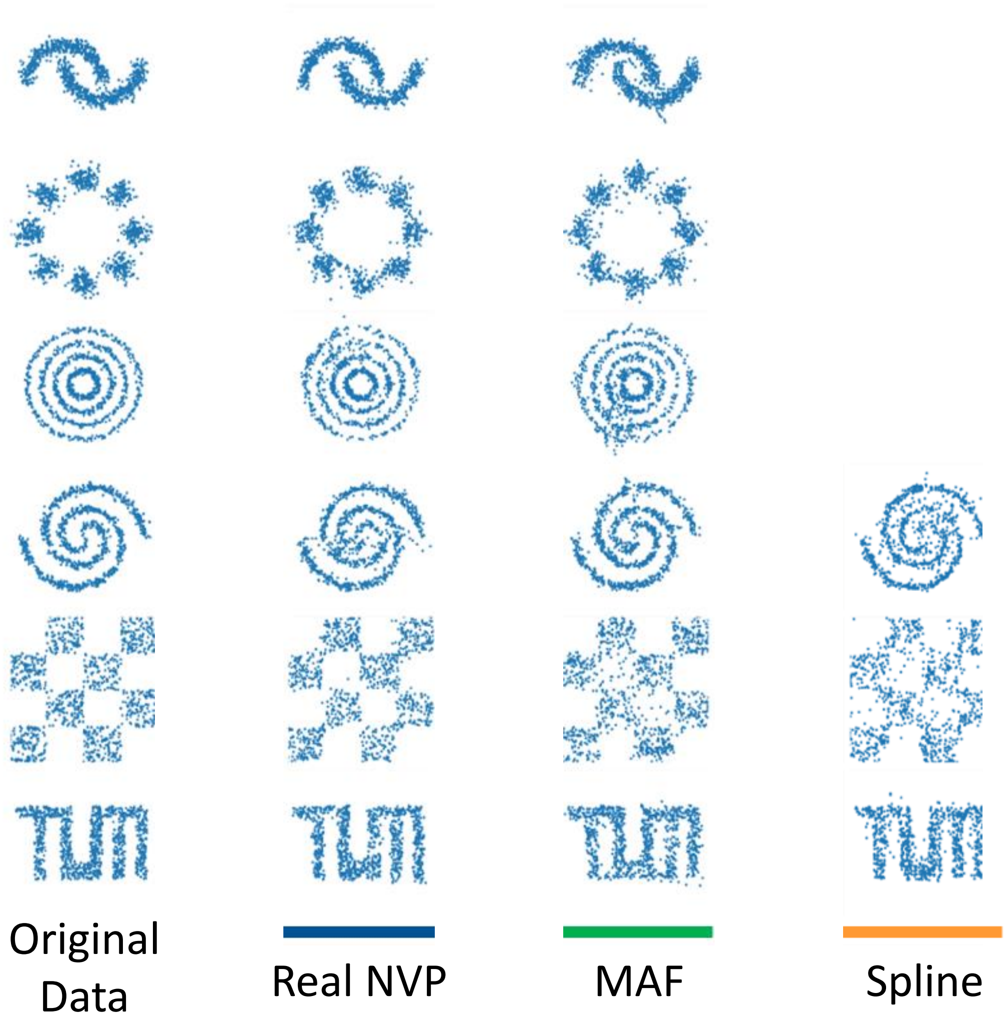
$$f_\theta = f_{\theta_K} \circ \ldots \circ f_{\theta_1}$$

- The forward parametrization enables **sampling** from the distribution $p_K(\boldsymbol{x})$.
  1. Sample $\boldsymbol{z}_{\boldsymbol{0}}^{(j)} \sim p_0(\boldsymbol{z}_{\boldsymbol{0}})$ (e.g. Gaussian or Uniform)
  2. Compute the transformations $\boldsymbol{z}_{\boldsymbol{i}}^{(j)} = f_{\theta_i}(\boldsymbol{z}_{\boldsymbol{i-1}}^{(j)})$ and $\left|\det\left(\frac{\partial f_{\theta_i}\left(\boldsymbol{z}_{\boldsymbol{i-1}}^{(j)}\right)}{\partial \boldsymbol{z}_{\boldsymbol{i-1}}^{(j)}}\right)\right|^{-1}$
  3. For the particular sample $\boldsymbol{x}^{(j)} = \boldsymbol{z}_{\boldsymbol{K}}^{(j)}$, we can compute $p_K(\boldsymbol{x}^{(j)})$

- Forward pointer: This is exactly what we need in Variational Inference
  - Sample $\boldsymbol{x}$ from a distribution q and compute the probability $q(\boldsymbol{x})$ for this sample

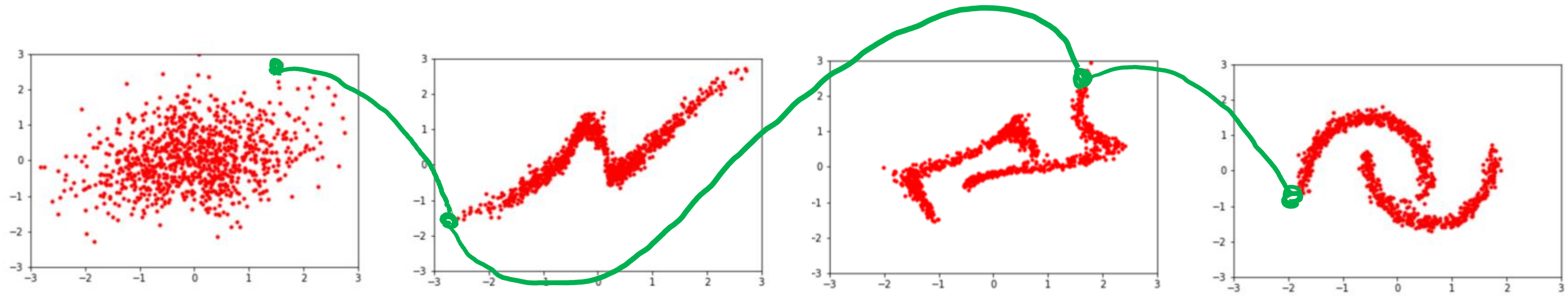
Forward Parametrization

Forward parametrization (Sampling):



*TUM Lab Course:
- Lukas Rinder
- Markus Kittel
- Murat Can

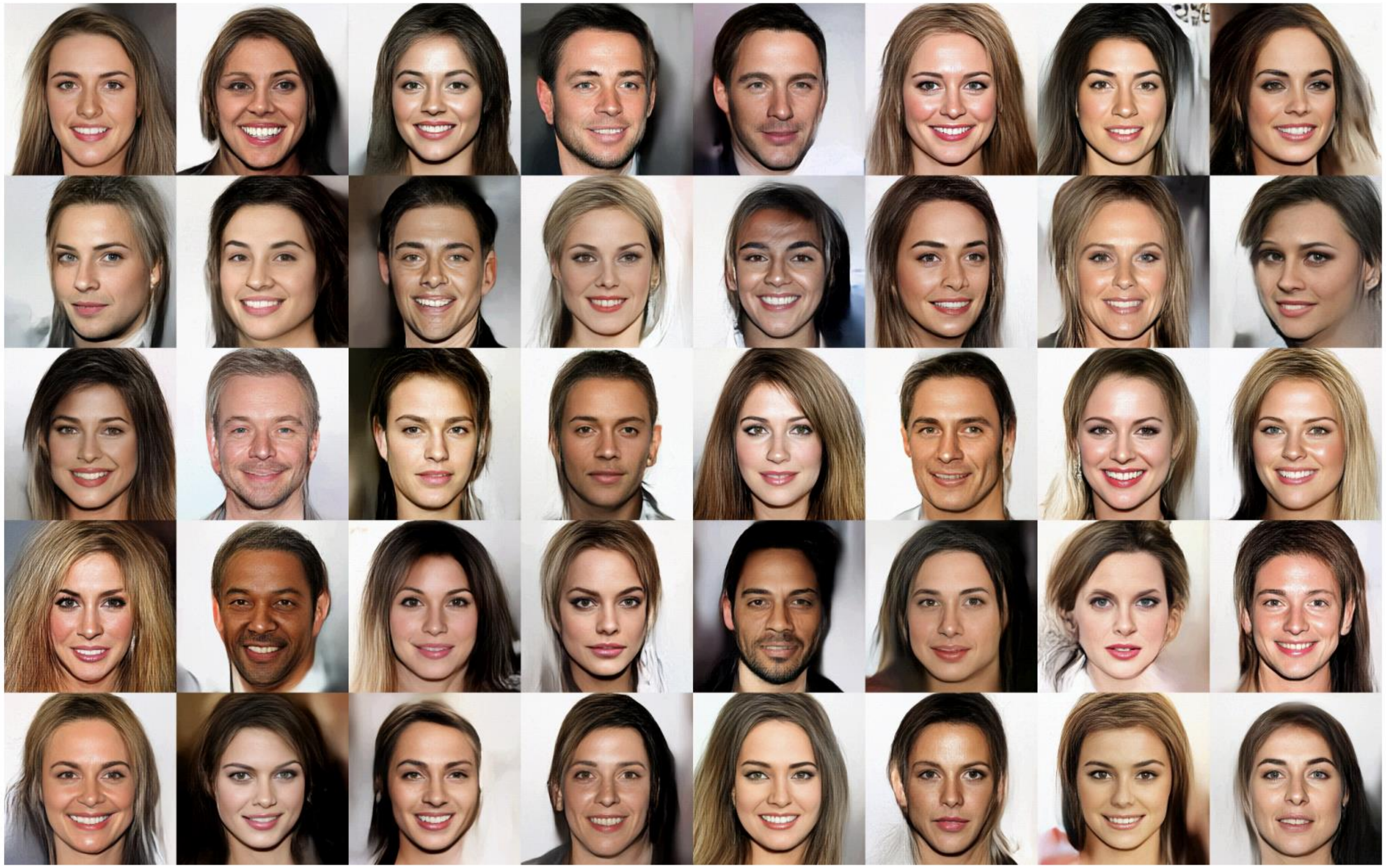
Transformation via Stacking



Visualization of a 4 layer RealNVP flow

[Eric Jang blog]

Example: Generating Images



[Kingma, Dhariwal; Glow: Generative Flow with Invertible 1×1 Convolutions]

Questions – NF2

1. For which $\boldsymbol{x}$, is it possible to compute $p(\boldsymbol{x})$ with the forward parametrization ?

2. Propose a reverse parametrization of $\exp(-x^n)$. Is it possible for any n ?

3. Propose a reverse parametrization of a sigmoid .

Roadmap

- Chapter: Deep Generative Models
  1. Introduction
  2. **Normalizing Flows**
     – Change of Variable Formula
     – Forward and Reverse Parametrization
     – **Jacobian Determinant Computation**
  3. Variational Inference
  4. Generative Adversarial Networks

Jacobian Determinant Computation

- The change of variable formula involves the Jacobian determinant

$$p_2(\boldsymbol{x}) = p_1(f^{-1}(\boldsymbol{x})) \cdot \left|\det\left(\frac{\partial f^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|$$

- Jacobian computation can be hard/slow:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix}; \qquad g(\boldsymbol{x}) = \begin{bmatrix} g_1(\boldsymbol{x}) \\ \vdots \\ g_D(\boldsymbol{x}) \end{bmatrix}; \qquad J_g = \begin{bmatrix} \frac{\partial g_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial g_1(\boldsymbol{x})}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_D(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial g_D(\boldsymbol{x})}{\partial x_D} \end{bmatrix}$$

- How to compute effectively the Jacobian determinant ?
  - **Diagonal Jacobian**
  - **Triangular Jacobian**
  - **Full Jacobian**

Determinant properties

- Determinant of inverse: $$\det(A^{-1}) = \frac{1}{\det(A)}$$

- Determinant and eigenvalues: $$\det(A) = \prod_{i=1}^{D} \lambda_i$$

$$eigenvalues(A) = \{\lambda_i; \mathrm{i} = 1..\mathrm{D}\}$$

- Determinant and block matrices: $$\det(A) = \det(B)\det(C)$$

$$A = \begin{bmatrix} B & 0 \\ 0 & C \end{bmatrix}$$

Diagonal Jacobian

- The function is applied element wise i.e.

$$g(\boldsymbol{x}) = \begin{bmatrix} g_1(x_1) \\ \vdots \\ g_D(x_D) \end{bmatrix}$$

- The Jacobian is a diagonal matrix i.e.

$$J_g = \begin{bmatrix} \frac{\partial g_1(x_1)}{\partial x_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\partial g_D(x_D)}{\partial x_D} \end{bmatrix}$$

- The determinant is the product of the diagonal elements ($O(D)$ complexity) i.e.

$$\det(J_g) = \prod_{i=1}^{D} \frac{\partial g_i(x_i)}{\partial x_i}$$

Triangular Jacobian

$* \; g_1(x_1)$

$\triangle \; g_2(x_1, x_2)$

$\vdots$

$\circ \; g_D(x_1, \dots, x_D)$

$$g(\vec{x}) = \begin{bmatrix} * \\ \triangle \\ \vdots \\ \circ \end{bmatrix}$$

- The function is applied as

$$g(\boldsymbol{x}) = \begin{bmatrix} g_1(x_1) \\ \vdots \\ g_D(x_1, \dots, x_D) \end{bmatrix}$$

- The Jacobian is a triangular matrix i.e.

$$J_g = \begin{bmatrix} \frac{\partial g_1(x_1)}{\partial x_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \frac{\partial g_D(x_1, \dots, x_D)}{\partial x_1} & \cdots & \frac{\partial g_D(x_1, \dots, x_D)}{\partial x_D} \end{bmatrix}$$

- The determinant is the product of the diagonal elements ($O(D)$ complexity) i.e.

$$\det(J_g) = \prod_{i=1}^{D} \frac{\partial g_i(\boldsymbol{x})}{\partial x_i}$$

- Examples:
  - Autoregressive flows

Full Jacobian

- The function is applied in the most general form

$$g(\boldsymbol{x}) = \begin{bmatrix} g_1(\boldsymbol{x}) \\ \vdots \\ g_D(\boldsymbol{x}) \end{bmatrix}$$

- The Jacobian is

$$J_g = \begin{bmatrix} \frac{\partial g_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial g_1(\boldsymbol{x})}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_D(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial g_D(\boldsymbol{x})}{\partial x_D} \end{bmatrix}$$

- The determinant can be computed with LU decomposition in $O(D^3)$ complexity
  - $J_g = LU$ where L lower triangular matrix and U upper triangular matrix.
  - $\det(J_g) = \det(L)\det(U)$ where $\det(L)$ and $\det(U)$ are diagonal products.
- Alternative for full Jacobian:
  - Continuous-time flows

Questions – NF3

1. Let's assume you get the following Jacobian:

$$\begin{bmatrix} \frac{\partial g_1(x_1)}{\partial x_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \frac{\partial g_D(x_1, \ldots, x_D)}{\partial x_1} & \cdots & 0 \end{bmatrix}$$

   How expensive is it to compute the determinant?
   Can you comment on this in the context of NFs?

2. What is the complexity to compute the Jacobian determinant of an arbitrary valid transformation ?

3. What happens for the det(J) when D is high ?

References

– [1] Eric Jang blog : https://blog.evjang.com/2018/01/nf1.html

– [2] Lilian Weng blog : https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

– [3] Normalizing Flows for Probabilistic Modeling and Inference: https://arxiv.org/abs/1912.02762

External Sources

Web tutorial:

- Adam Kosiorek: http://akosiorek.github.io/ml/2018/04/03/norm_flows.html
- Eric Jang blog : https://blog.evjang.com/2018/01/nf1.html
- CS236 - Fall 2019 (Stanford) : https://deepgenerativemodels.github.io/notes/flow/

Survey papers:

- Normalizing Flows: An Introduction and Review of Current Methods: https://arxiv.org/pdf/1908.09257.pdf
- Normalizing Flows for Probabilistic Modeling and Inference: https://arxiv.org/abs/1912.02762

Video:

- What are normalizing flows ?: https://www.youtube.com/watch?v=i7LjDvsLWCg