# Normalizing Flows for Probabilistic Modeling and Inference

G. Papamakarios - E. Nalisnick - D. J. Rezende - S. Mohamed - B. Lakshminarayanan

Advanced Machine Learning Course

**March 2020**

## Motivation

Normalizing flows provide a general mechanism for defining expressive probability distributions, only requiring the specification of a (usually simple) base distribution and a series of bijective transformations

- First introduced in 2015.
- Let $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ be a dataset, assumed to be iid from the unknown $\boldsymbol{p_x}$.
- A normalizing flow approximates the distibution $\boldsymbol{p_x}$ by $\boldsymbol{p_\theta}$.
- $\boldsymbol{p_\theta} = \mathcal{N}(\mu, \Sigma)$ is a very simple normalizing flow (blackboard).
- Can be see as a concurrent method to GAN and VAE.

# Motivation

## Generation: Images and video

https://openai.com/blog/glow/

# Motivation

## Generation: Audio

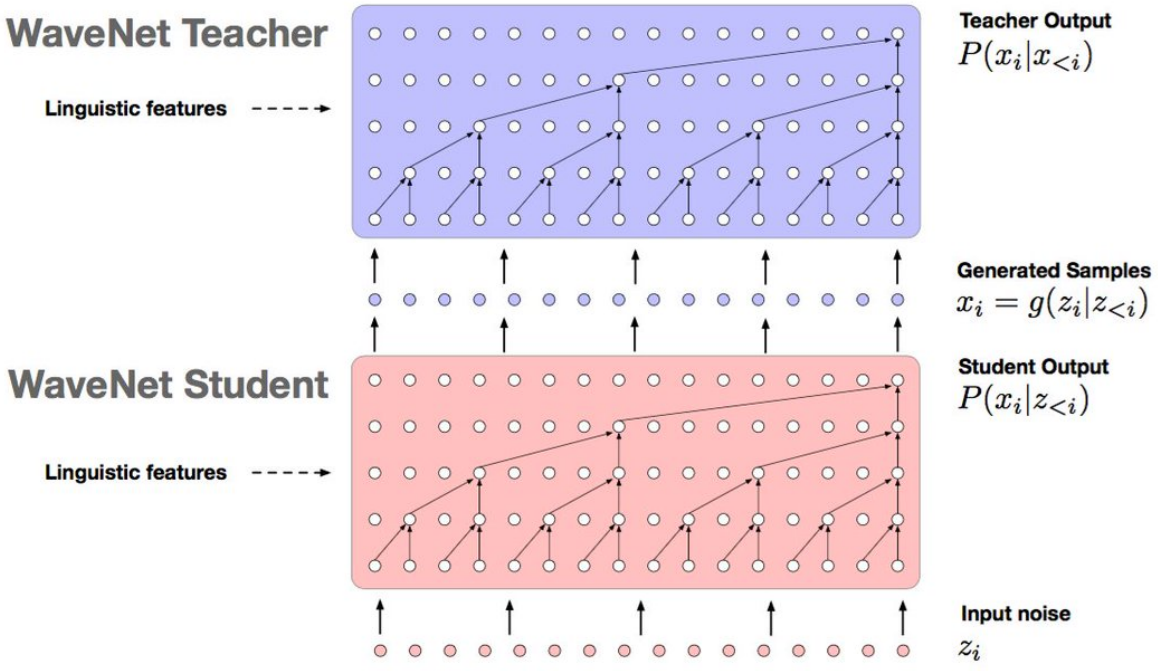



Figure 2: **Overview of Probability Density Distillation**. A pre-trained WaveNet teacher is used to score the samples $\boldsymbol{x}$ output by the student. The student is trained to minimise the KL-divergence between its distribution and that of the teacher by maximising the log-likelihood of its samples under the teacher and maximising its own entropy at the same time.

# Motivation

## Inference: Importance and rejection sampling

- Example:
  - Two populations $X_1 \sim p_{x_1}$ and $X_2 \sim p_{x_2}$.
  - You have the result $y_1$ of applying the function $f(.)$ to $X_1$.
  - Use a flow to model the population distributions.
  - Approximate the average effect over the population $X_2$ by

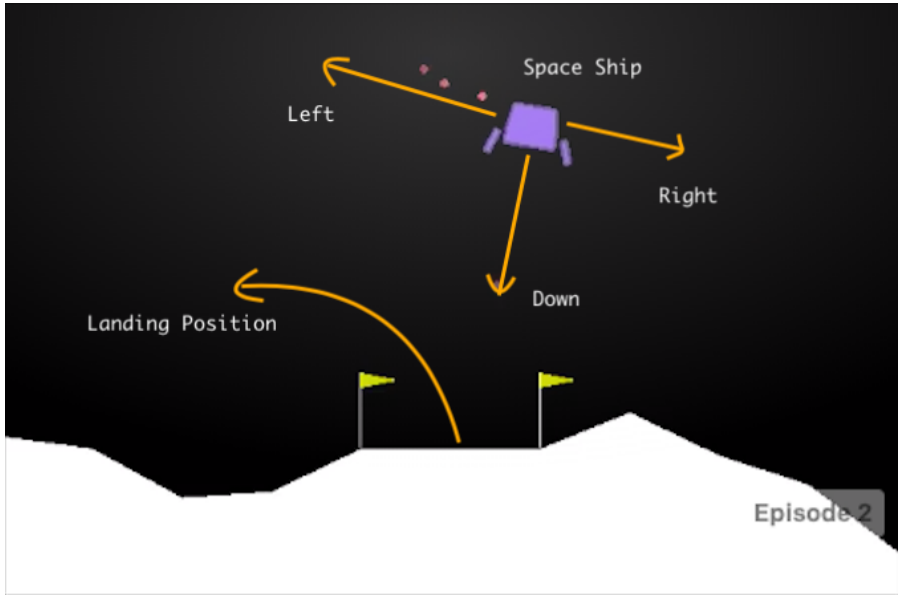
$$\frac{1}{N} \sum_{i=1}^{N} y_1(i) \frac{p_{x_2}}{p_{x_1}}$$

.

# Motivation

## Reinforcement Learning

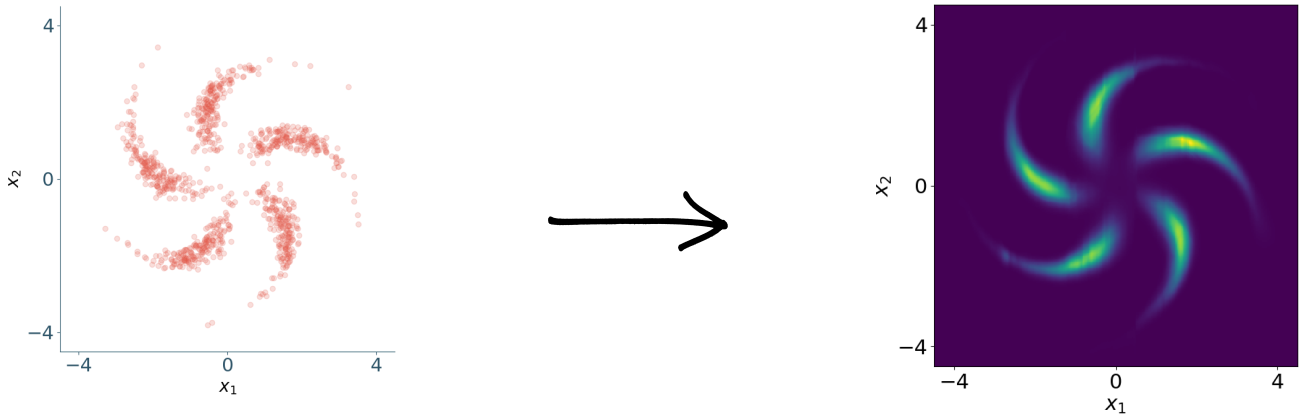
- Parameterization of continuous policies:
  - $p(a_t|s_t)$ modeled by a Normalizing Flow.
  - Sample actions and evaluate the corresponding density with $\hat{a}_t = T(z_t; s_t), z_t \sim \mathcal{N}(0, I)$ and $T^{-1}\exists$.
  - Optimize with a soft actor critic framework.




———

https://mc.ai/solving-lunar-lander-openaigym-reinforcement-learning/

# Density Estimation 👇




> Density estimation aims at estimating the pdf of underlying data from an iid dataset.

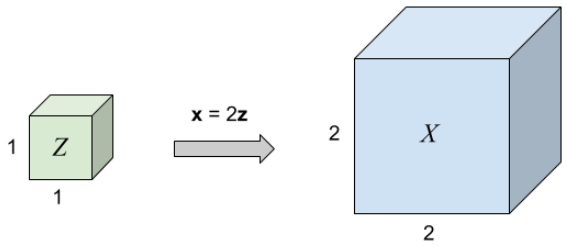
# Change of Variables Theorem (1)

> Given a random variable $\mathcal{Z}$ and a bijective function $f$, how does the density of $\mathcal{X} = f(\mathcal{Z})$ behave in terms of $p(\mathbf{z})$ and $f$?

# Change of Variables Theorem (1)

> Given a random variable $\mathcal{Z}$ and a bijective function $f$, how does the density of $\mathcal{X} = f(\mathcal{Z})$ behave in terms of $p(\mathbf{z})$ and $f$?

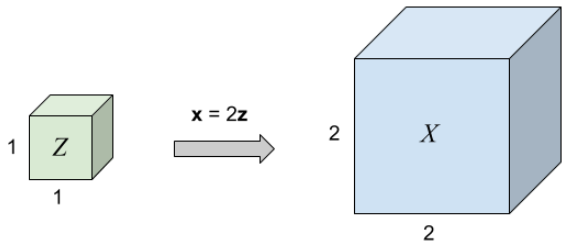
Assume $p(z)$ is a uniformely distributed unit cube in $\mathbb{R}^3$, and $\mathbf{x} = f(\mathbf{z}) = 2\mathbf{z}$.



# Change of Variables Theorem (1)

> Given a random variable $\mathcal{Z}$ and a bijective function $f$, how does the density of $\mathcal{X} = f(\mathcal{Z})$ behave in terms of $p(\mathbf{z})$ and $f$?

Assume $p(z)$ is a uniformely distributed unit cube in $\mathbb{R}^3$, and $\mathbf{x} = f(\mathbf{z}) = 2\mathbf{z}$.



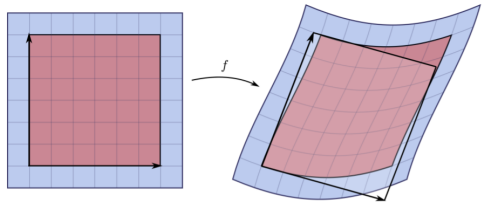

The total probability mass must be conserved, therefore

$p(\mathbf{x} = f(\mathbf{z})) = p(\mathbf{z})\frac{V_\mathbf{z}}{V_\mathbf{x}} = p(\mathbf{z})\frac{1}{8}$, where $\frac{1}{8} = \left|\det \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}\right|^{-1}$ is the determinant of the linear transformation $f$.

# Change of Variables Theorem (2)

## What if the transformation is non linear?

- The Jacobian $J_f(\mathbf{z})$ of $\mathbf{x} = f(\mathbf{z})$ represents the infinitesimal linear transformation in the neighbourhood of $\mathbf{z}$.



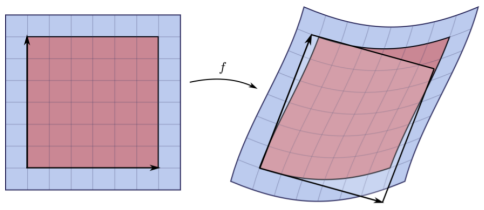

———

https://www.lesswrong.com/posts/KKwv9kcQz29vqPLAD/a-primer-on-matrix-calculus-part-2-jacobians-and-other-fun

# Change of Variables Theorem (2)

## What if the transformation is non linear?

- The Jacobian $J_f(\mathbf{z})$ of $\mathbf{x} = f(\mathbf{z})$ represents the infinitesimal linear transformation in the neighbourhood of $\mathbf{z}$.




- If the function is a bijective map then the mass must be conserved locally.

Therefore, we can compute the local change of density as

$$p(\mathbf{x}) = p(\mathbf{z}) \left|\det J_f(\mathbf{z})\right|^{-1} .$$

———

https://www.lesswrong.com/posts/KKwv9kcQz29vqPLAD/a-primer-on-matrix-calculus-part-2-jacobians-and-other-fun

# Change of Variables Theorem (3)

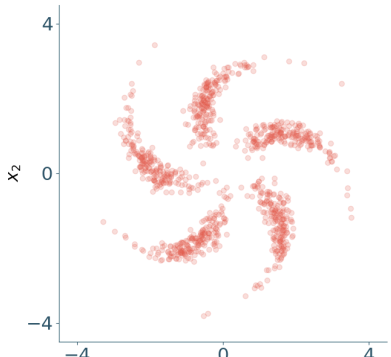
The combination of the right bijective map and any base distribution allows to represent any continuous random variable.

# Change of Variables Theorem (3)

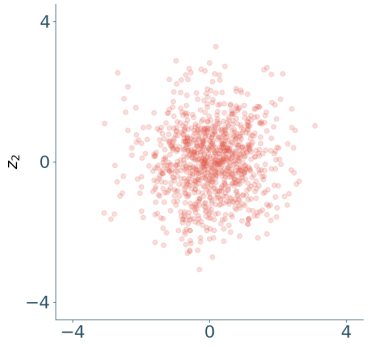
The combination of the right bijective map and any base distribution allows to represent any continuous random variable.

$$p(\mathbf{x};\theta) = p(\mathbf{z} = \mathbf{g}(\mathbf{x};\theta)) \left|\det J_g(\mathbf{x};\theta)\right|, \quad \mathbf{g}(.;\theta) \text{ a neural network.}$$

- The bijective function takes in samples and maps them to latent variables.
- This process is refered as normalization if the latent variables distribution is normal.

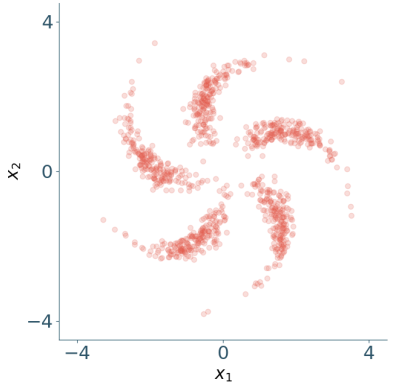



Density Estimation

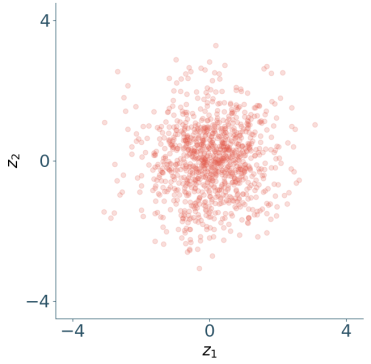


# Change of Variables Theorem (3)

The combination of the right bijective map and any base distribution allows to represent any continuous random variable.




Sampling




Once learned, the function can be inverted in order to generate samples.

# Bijectivity with Neural Nets? 🤔

- $\begin{bmatrix} z_1 & \dots & z_d \end{bmatrix} = g(\begin{bmatrix} x_1 & \dots & x_d \end{bmatrix})$, $g$ can be a NN.
- $g$ is autoregressive if it can be decomposed as: $z_i = g_i(\begin{bmatrix} x_1 & \dots & x_i \end{bmatrix})$
- If the $g_i$ are invertible with respect to $x_i \forall i$, $g$ is bijective.

# Bijectivity with Neural Nets? 🤔

- $\begin{bmatrix} z_1 & \dots & z_d \end{bmatrix} = g(\begin{bmatrix} x_1 & \dots & x_d \end{bmatrix})$, $g$ can be a NN.
- $g$ is autoregressive if it can be decomposed as: $z_i = g_i(\begin{bmatrix} x_1 & \dots & x_i \end{bmatrix})$
- If the $g_i$ are invertible with respect to $x_i \forall i$, $g$ is bijective.

## The determinant of the Jacobian can be efficiently computed.

The Jacobian of an autoregressive tranformation has the following form:

$$J_g(\mathbf{x}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \frac{\partial g_1}{\partial x_3} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \frac{\partial g_2}{\partial x_3} \\ \frac{\partial g_3}{\partial x_1} & \frac{\partial g_3}{\partial x_2} & \frac{\partial g_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & 0 & 0 \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & 0 \\ \frac{\partial g_3}{\partial x_1} & \frac{\partial g_3}{\partial x_2} & \frac{\partial g_3}{\partial x_3} \end{bmatrix}.$$

# Bijectivity with Neural Nets? 🤔

- $\begin{bmatrix} z_1 & \dots & z_d \end{bmatrix} = g(\begin{bmatrix} x_1 & \dots & x_d \end{bmatrix})$, $g$ can be a NN.
- $g$ is autoregressive if it can be decomposed as: $z_i = g_i(\begin{bmatrix} x_1 & \dots & x_i \end{bmatrix})$
- If the $g_i$ are invertible with respect to $x_i \forall i$, $g$ is bijective.

## The determinant of the Jacobian can be efficiently computed.

The Jacobian of an autoregressive tranformation has the following form:

$$J_g(\mathbf{x}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \frac{\partial g_1}{\partial x_3} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \frac{\partial g_2}{\partial x_3} \\ \frac{\partial g_3}{\partial x_1} & \frac{\partial g_3}{\partial x_2} & \frac{\partial g_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & 0 & 0 \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & 0 \\ \frac{\partial g_3}{\partial x_1} & \frac{\partial g_3}{\partial x_2} & \frac{\partial g_3}{\partial x_3} \end{bmatrix}.$$

## Chain Rule

An autoregressive density estimator learns the chain rule's factors:
$p(\mathbf{x}) = p(x_1)\Pi_{i=2}^{d} p(x_i|x_1, ..., x_{i-1}).$

# Affine Autoregressive Networks

> Idea: Autoregressive Networks combined with linear transformations.

- $z_1 = \sigma_1 \times x_1 + \mu_1$
- $z_i = \sigma_i(x_1, ..., x_{i-1}) \times x_i + \mu_i(x_1, ..., x_{i-1})$

# Affine Autoregressive Networks

> Idea: Autoregressive Networks combined with linear transformations.

- $z_1 = \sigma_1 \times x_1 + \mu_1$
- $z_i = \sigma_i(x_1, ..., x_{i-1}) \times x_i + \mu_i(x_1, ..., x_{i-1})$

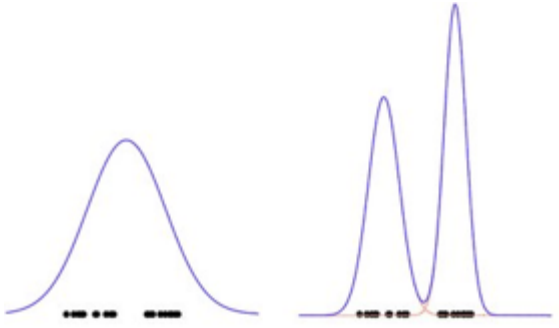
## Invertible?

- $x_1 = g_1^{-1}(\begin{bmatrix} z_1 & ... & z_d \end{bmatrix}) = g_1^{-1}(z_1) = \frac{(z_1 - \mu_1)}{\sigma_1}$
- $x_i = \frac{z_i - \mu_i(\begin{bmatrix} x_1 & ... & x_{i-1} \end{bmatrix})}{\sigma_i(\begin{bmatrix} x_1 & ... & x_{i-1} \end{bmatrix})}$

# Affine Autoregressive Networks

> Idea: Autoregressive Networks combined with linear transformations.

But linear transformations are not very expressive:

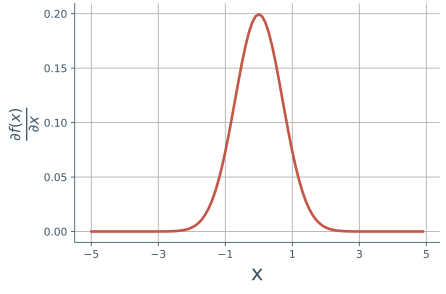


# Non affine Transformations

> How can we enforce the monotonicity of a function modeled by a neural network

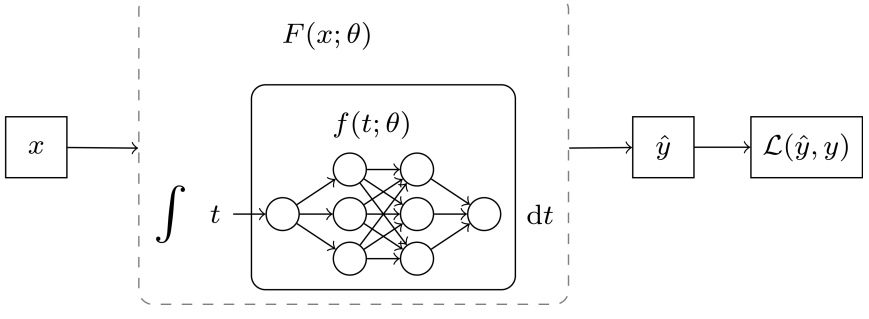
## UMNN: Unconstrained Monotonic Neural Network

Possible solution: To model and integrate the derivative.




The only constraint is on the output value which must be of constant sign (e.g. positive).

This can be achieved by applying an exponential on the output neuron.

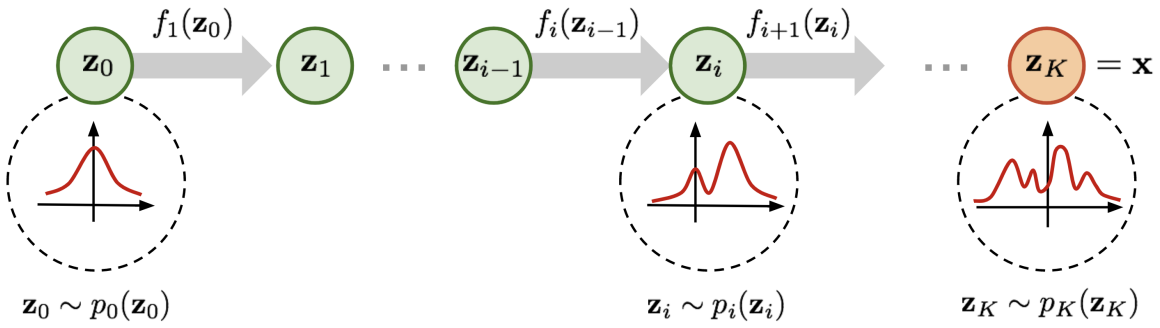


# Finite Composition

- Stack multiple bijective transformations: $\mathbf{g} = \mathbf{g}_K \circ ... \circ \mathbf{g}_1$
- $\mathbf{z}_0 = \mathbf{x}$ and $\mathbf{z}_K = \mathbf{z}$
- $\mathbf{z}_k = \mathbf{g}_k(\mathbf{z}_{k-1})$ for $k = 1 : K$
- $\log |J_g(\mathbf{x})| = \log |\prod_{k=1}^{K} J_{g_k}(\mathbf{z}_{k-1})| = \sum_{k=1}^{K} \log |J_{g_k}(\mathbf{z}_{k-1})|$

# Finite Composition

- Stack multiple bijective transformations: $\mathbf{g} = \mathbf{g}_K \circ ... \circ \mathbf{g}_1$
- $\mathbf{z}_0 = \mathbf{x}$ and $\mathbf{z}_K = \mathbf{z}$
- $\mathbf{z}_k = \mathbf{g}_k(\mathbf{z}_{k-1})$ for $k = 1 : K$
- $\log|J_g(\mathbf{x})| = \log|\prod_{k=1}^{K} J_{g_k}(\mathbf{z}_{k-1})| = \sum_{k=1}^{K} \log|J_{g_k}(\mathbf{z}_{k-1})|$
- $\mathbf{g}^{-1} = \mathbf{f} = \mathbf{f}_K \circ ... \circ \mathbf{f}_1$



# Autoregressive flows

- $z_i = \tau(x_i; \mathbf{h}_i)$ where $\mathbf{h}_i = c_i(\mathbf{x}_{<i})$
- The determinant of the Jacobian is equal to the products of $\frac{\partial \tau(x_i; \mathbf{h}_i)}{\partial x_i}$.
- (See previous slides for more details)

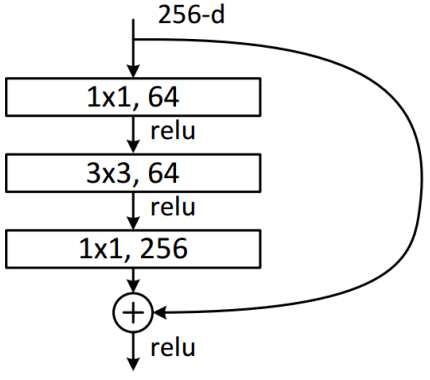
# Linear flows

- $z = Wx$ where $\det(W) \neq 0$
- The Jacobian is trivially given by $\det(W)$.
- It generalizes the idea of permutation.
- Example to parameterize $W$:
  - $W = PLU$
  - $W = \Sigma$

# Residual flows

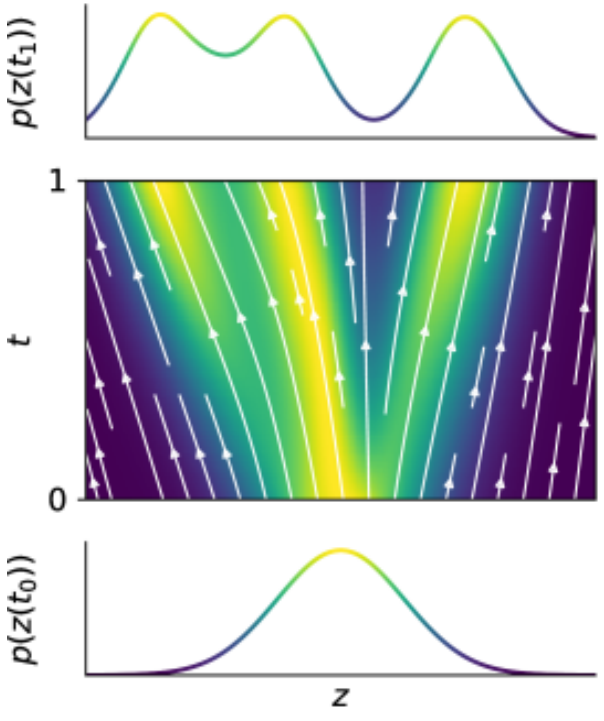
$$\mathbf{z} = F_\theta(\mathbf{x}) = \mathbf{x} + g_\theta(\mathbf{x})$$

- Invertible if $g_\phi : \mathbb{R}^d \to \mathbb{R}^d$ is contractive:
  - $\forall \mathbf{x}, \mathbf{y} \in \mathrm{Dom}_{g_\phi}, \|g_\phi(\mathbf{x}) - g_\phi(\mathbf{y})\| < \|\mathbf{x} - \mathbf{y}\|$
  - The sequence $\mathbf{x}_t = \mathbf{z} - g_\phi(\mathbf{x}_{t-1})$ converges to $F_\phi^{-1}(\mathbf{z})$.
- If the function $g_\theta$ is L-Lipschitz with $L < 1$ we are fine.
- We can combine these functions.
- Intractable Jacobian, we can rely on Hutchinson trace estimator for the determinant.



# Continuous-time transformations

- A neural network parameterizes an ordinary differential equation.
- $\frac{d\mathbf{x}_t}{dt} = g_\phi(t, \mathbf{x}_t)$
- $\mathbf{z} = \mathbf{x}_{t_1} = \int_{t_0}^{t_1} g_\phi(t, \mathbf{x}_t)dt + \mathbf{x}_{t_0}$.
- Solved with numerical integrator.
- Backward can be made memory efficient by solving another ODE.
- Jacobian determinant is estimated as for res-flows.



# Challenges

- **Inductive Bias**
- Computation Efficiency

# Challenges

- **Inductive Bias**
- Computation Efficiency
- **Inductive Bias**

# Challenges

- **Inductive Bias**
- Computation Efficiency
- **Inductive Bias**
- **Inductive Bias**

# Challenges

- **Inductive Bias**
- Computation Efficiency
- **Inductive Bias**
- **Inductive Bias**
- The problem of density estimation in high dimension is intractable without strong assumption, so what do we need?

# Challenges

- **Inductive Bias**
- Computation Efficiency
- **Inductive Bias**
- **Inductive Bias**
- The problem of density estimation in high dimension is intractable without strong assumption, so what do we need?