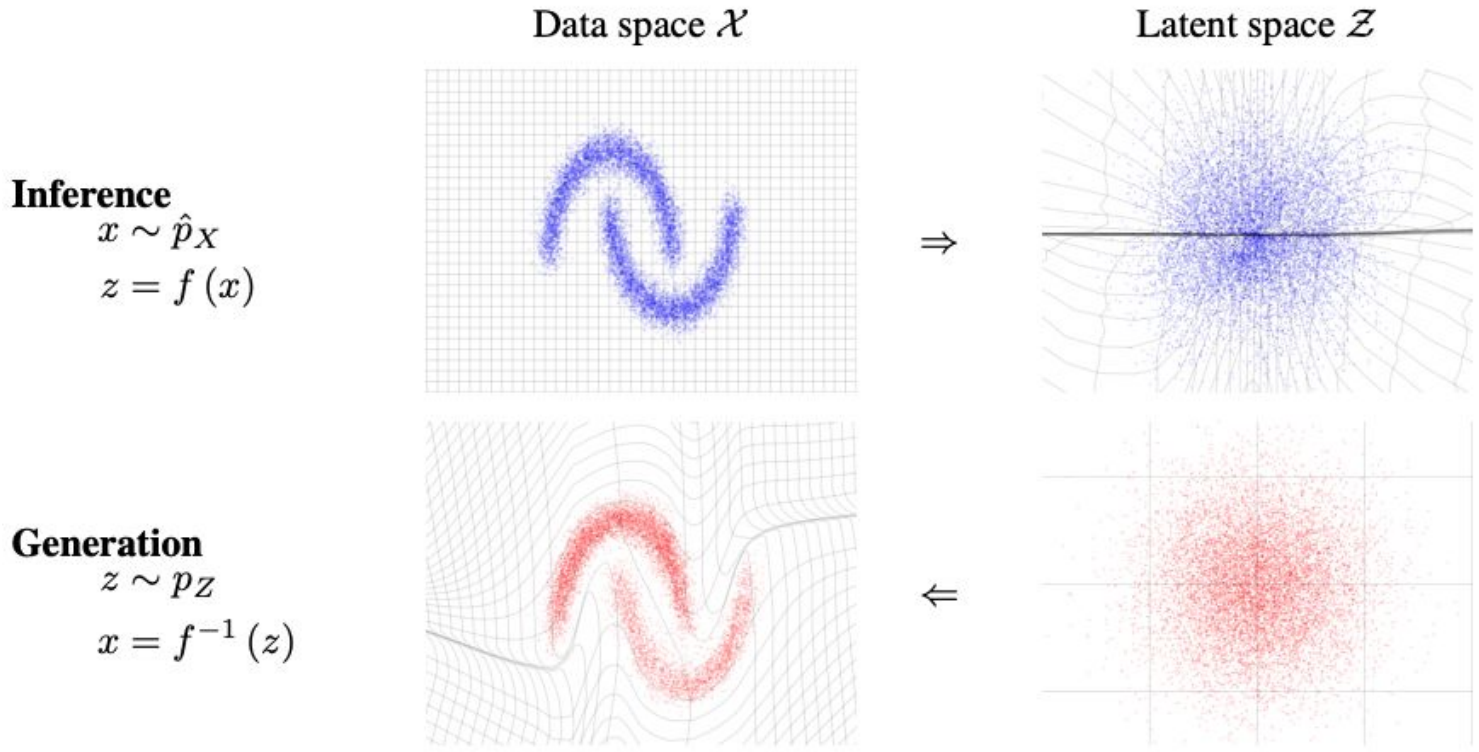
TOOPLOOX

Exploration of flow-based models

Michał Stypułkowski, Maciej Zięba,
Maciej Zamorski

Real NVP:
Real-valued Non-volume Preserving Transformations

Real NVP - main idea



Source: Dinh, Laurent, et. al. **"Density Estimation Using Real NVP "** ICLR 2017

Problem statement

We want to learn data distribution p_X having $x \sim p_X$.

Approach:

Find an **invertible** function f that maps x into $z \sim p_Z$,
where p_Z is simple distribution, e.g. Gaussian.

Problem statement

We want to learn data distribution p_X having $x \sim p_X$.

Approach:

Find an **invertible** function f that maps x into $z \sim p_Z$,
where p_Z is simple distribution, e.g. Gaussian.

What is our optimization goal here?

Optimization via change of variable formula

We can express p_X (**likelihood** of the data) as:

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

Optimization via change of variable formula

We can express p_X (**likelihood** of the data) as:

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

Jacobian of the function with respect to the data

Optimization via change of variable formula

We can express p_X (**likelihood** of the data) as:

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

Jacobian of the function with respect to the data

And in the logarithmic form (log-likelihood):

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

Constraints

We want:

- Easily invertible function f
- Determinant of the Jacobian which doesn't require complicated computing
- No second derivatives to calculate

Constraints

We want:

- Easily invertible function f
- Determinant of the Jacobian which doesn't require complicated computing
- No second derivatives to calculate

How do we find such a transformation?

Quick reminder from linear algebra

A matrix with elements above main diagonal equal to 0 is called **lower triangular matrix**. Its determinant is equal to product of the elements on the main diagonal, i.e.

$$det \begin{pmatrix} a_{11} & & & & \\ a_{21} & a_{22} & & 0 & \\ a_{31} & a_{32} & a_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ a_{n1} & \cdots & \cdots & \cdots & a_{nn} \end{pmatrix} = a_{11} \cdot a_{22} \cdot \ldots \cdot a_{nn}$$

Affine coupling layer

Given a data point $x \in \mathbb{R}^D$ and $d < D$ the **affine coupling transformation** is defined as:

$$y_{1:d} = x_{1:d}$$
$$y_{d+1:D} = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d})$$

where s and t are some functions from $\mathbb{R}^d \mapsto \mathbb{R}^{D-d}$.

Affine coupling layer

The Jacobian of this transformation is

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}^T} & \operatorname{diag}(\exp[s(x_{1:d})]) \end{bmatrix}$$

with the determinant

$$\exp\left[\sum_{j=1}^{D-d} s(x_{1:d})_j\right]$$

Affine coupling layer

The Jacobian of this transformation is

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}^T} & \operatorname{diag}(\exp[s(x_{1:d})]) \end{bmatrix}$$

Diagonal matrix

with the determinant

$$\exp\left[\sum_{j=1}^{D-d} s(x_{1:d})_j\right]$$

Affine coupling layer

Let's rewrite Jacobian term from the log-likelihood

$$\log\left(\left|\det\left(\frac{\partial f(x))}{\partial x^T}\right)\right|\right) = \log\left(\left|\exp\left[\sum_{j=1}^{D-d} s(x_{1:d})_j\right]\right|\right)$$

$$= \sum_{j=1}^{D-d} s(x_{1:d})_j$$

Inverse of the coupling layer

We can easily invert coupling layer

$$
\begin{cases} y_{1:d} = x_{1:d} \\ y_{d+1:D} = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d}) \end{cases}
$$

$$
\Leftrightarrow \begin{cases} x_{1:d} = y_{1:d} \\ x_{d+1:D} = \left(y_{d+1:D} - t(y_{1:d})\right) \odot \exp(-s(y_{1:d})) \end{cases}
$$

Note that we **don't** have to find inverse function to either s or t.

Inverse of the coupling layer

We can easily invert coupling layer

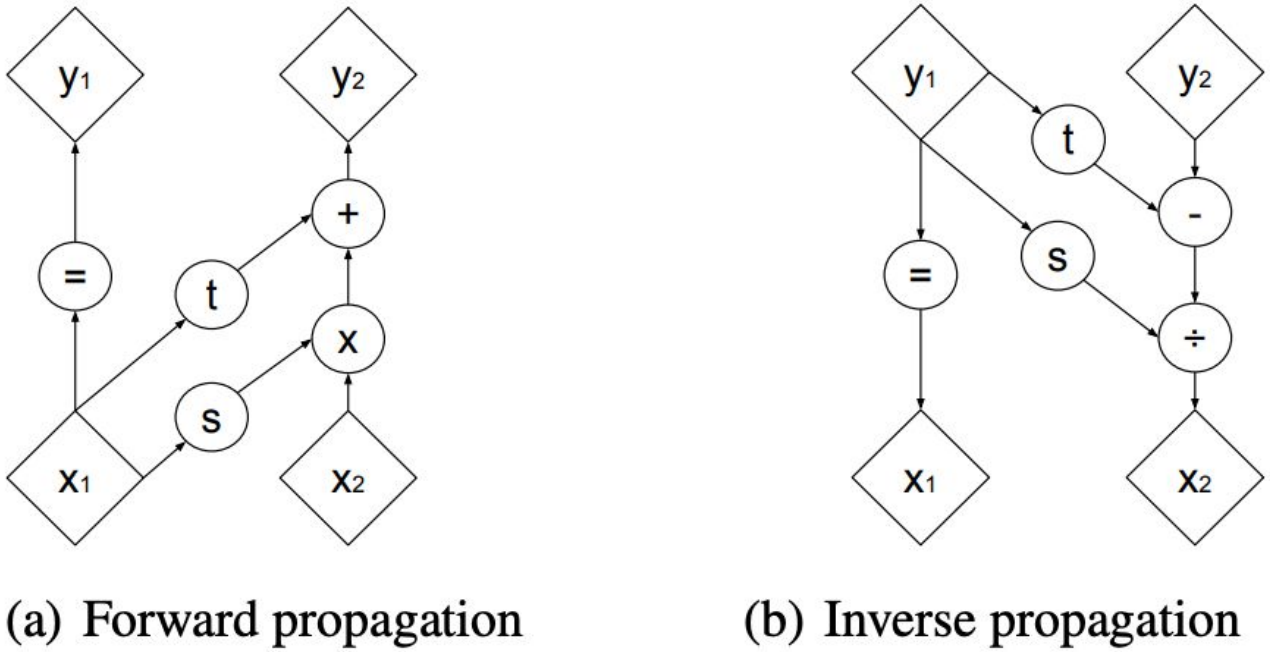
$$
\begin{cases}
y_{1:d} = x_{1:d} \\
y_{d+1:D} = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d})
\end{cases}
$$

$$
\Leftrightarrow
\begin{cases}
x_{1:d} = y_{1:d} \\
x_{d+1:D} = \left(y_{d+1:D} - t(y_{1:d})\right) \odot \exp(-s(y_{1:d}))
\end{cases}
$$

Note that we **don't** have to find inverse function to either s or t.

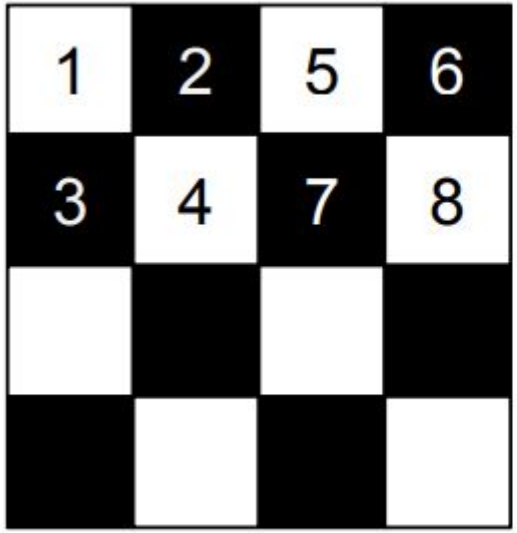
We can use Neural Networks!

Coupling transformation

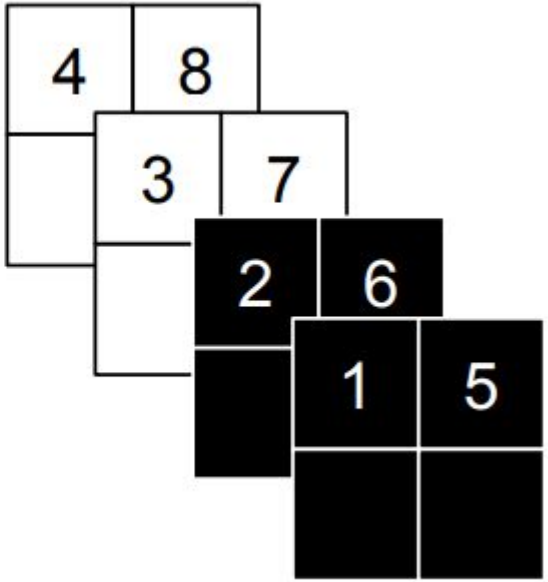


Source: Dinh, Laurent, et. al. **"Density Estimation Using Real NVP "** ICLR 2017

Mask types



Checkerboard



Channel-wise

Source: Dinh, Laurent, et. al. **"Density Estimation Using Real NVP "** ICLR 2017

Stacking coupling layers

We can stack many coupling layers

$$f = f_n \circ f_{n-1} \circ \ldots \circ f_1$$

Jacobian is still easy to compute due to the property

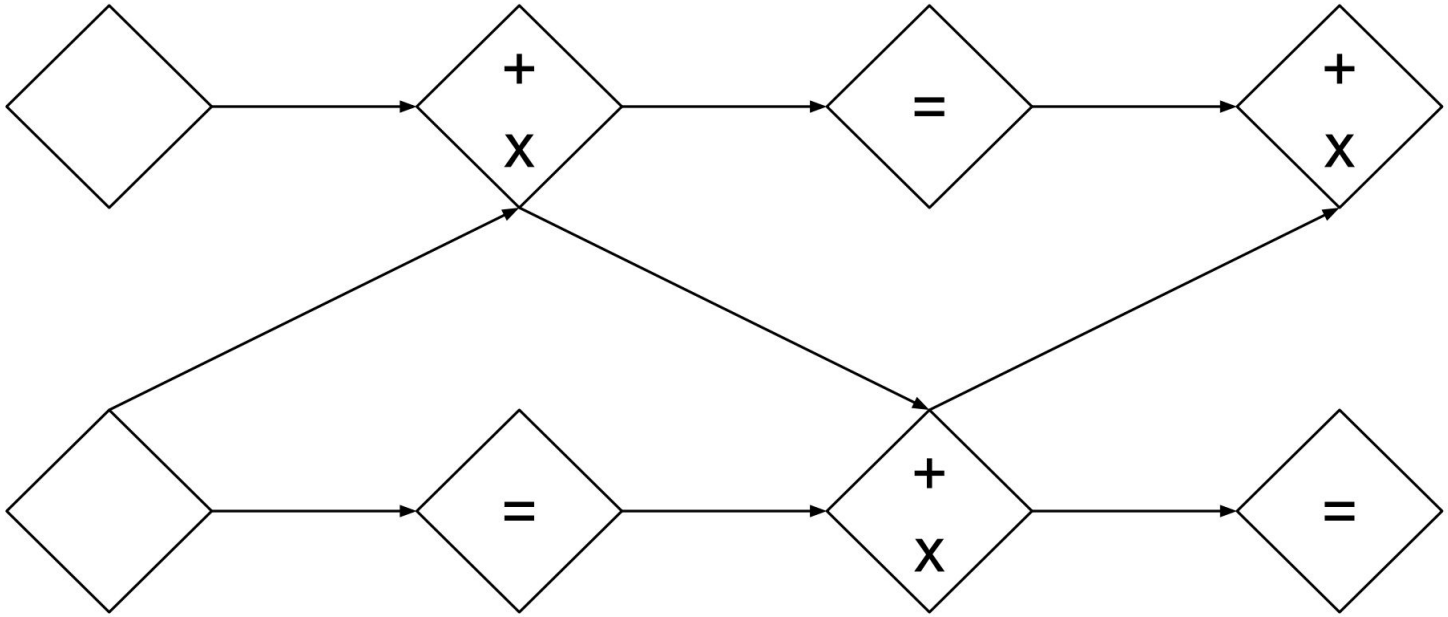
$$\frac{\partial (f_b \circ f_a)}{\partial x_a^T}(x_a) = \frac{\partial f_a}{\partial x_a^T}(x_a) \cdot \frac{\partial f_b}{\partial x_b^T}(x_b)$$

where $x_b = f_a(x_a)$.

And the inverse is given by $(f_b \circ f_a)^{-1} = f_a^{-1} \circ f_b^{-1}$.

Stacking coupling layers

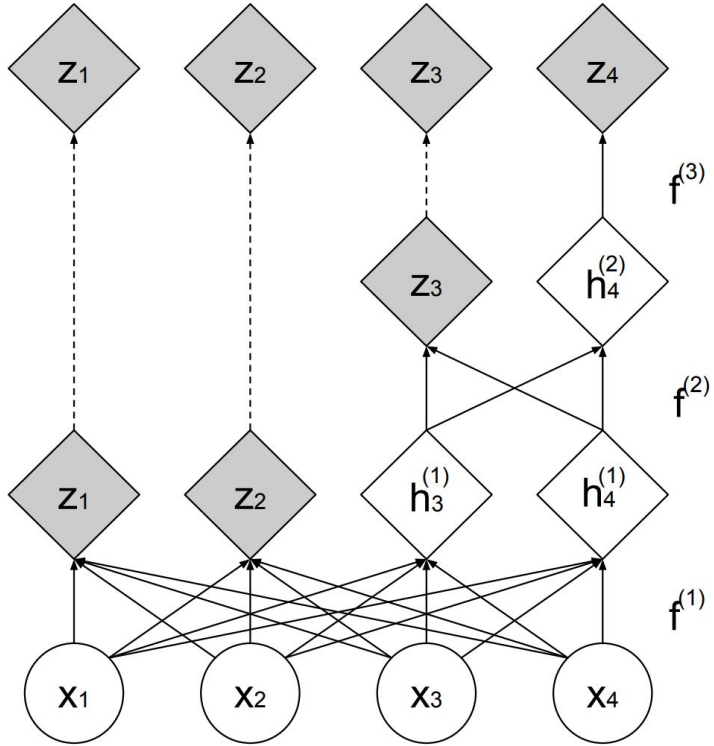
We take opposite mask after each coupling layer.



Discarding variables

To make model more efficient, we factor out half of the variables after each coupling layer.

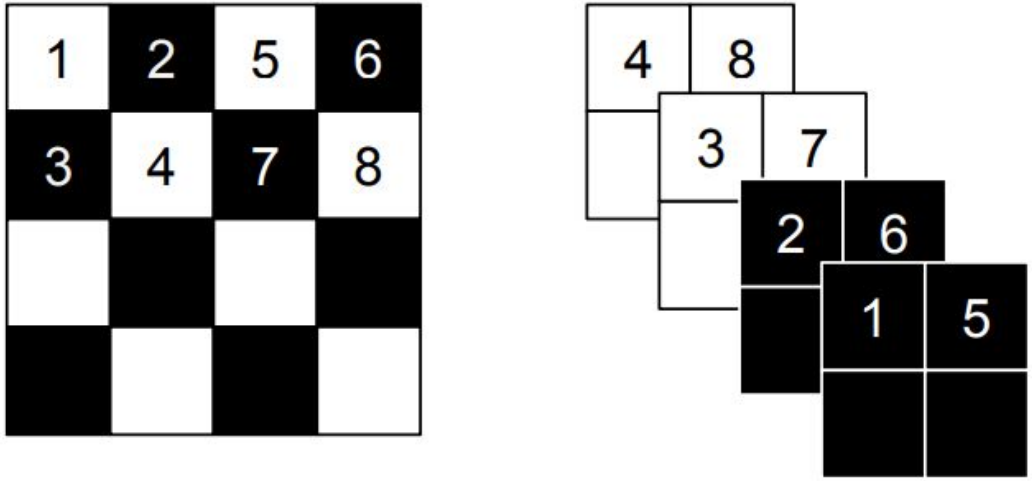
They are directly modeled as Gaussians.



Source: Dinh, Laurent, et. al. **"Density Estimation Using Real NVP "** ICLR 2017

Squeezing operation

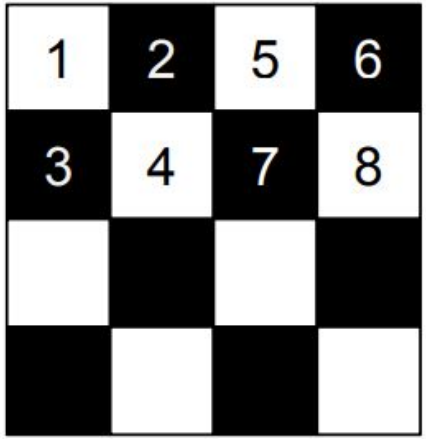
We split image into $2 \times 2 \times c$ squares
and reshape them into squares of shape $1 \times 1 \times 4c$.

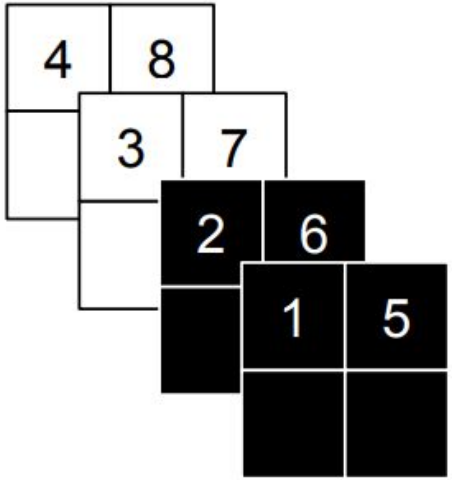


Source: Dinh, Laurent, et. al. **"Density Estimation Using Real NVP "** ICLR 2017

Squeezing operation

We split image into $2 \times 2 \times c$ squares
and reshape them into squares of shape $1 \times 1 \times 4c$.





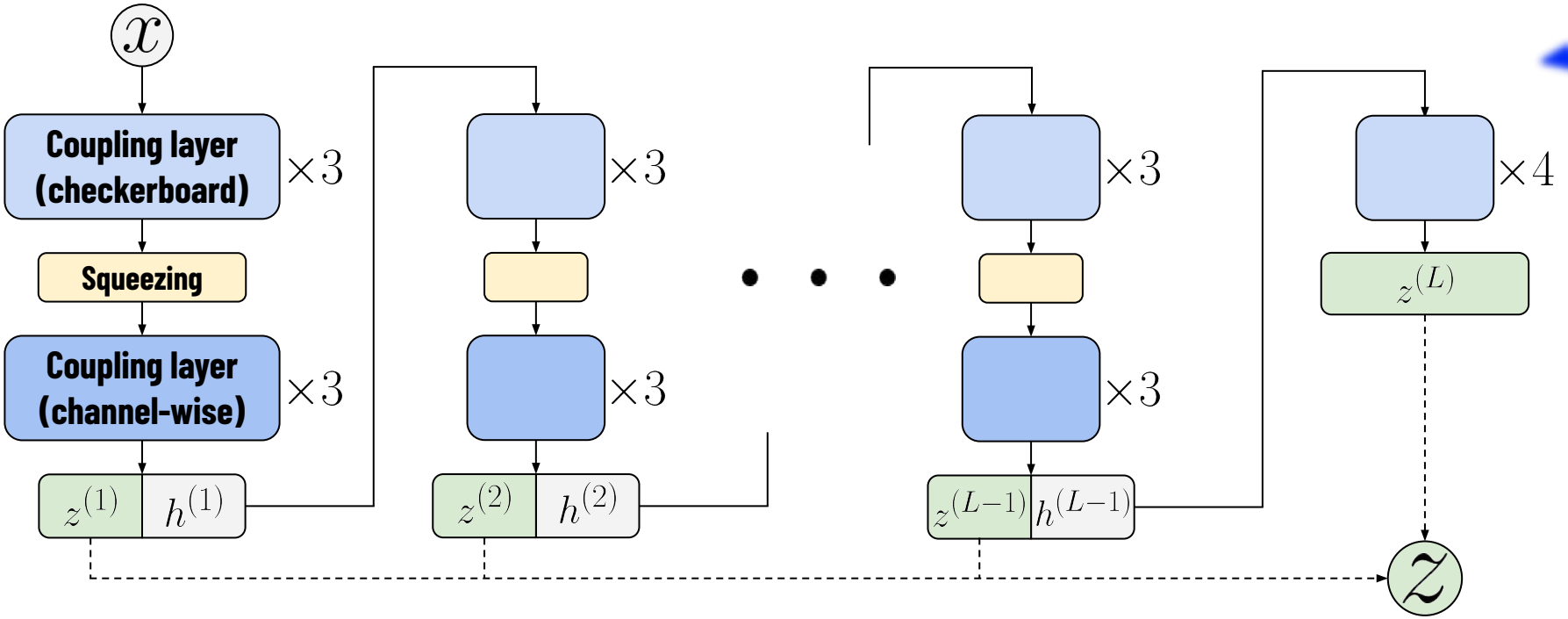
$$(s \times s \times c)$$

$$\downarrow$$

$$\left(\frac{s}{2} \times \frac{s}{2} \times 4c\right)$$

Source: Dinh, Laurent, et. al. **"Density Estimation Using Real NVP "** ICLR 2017

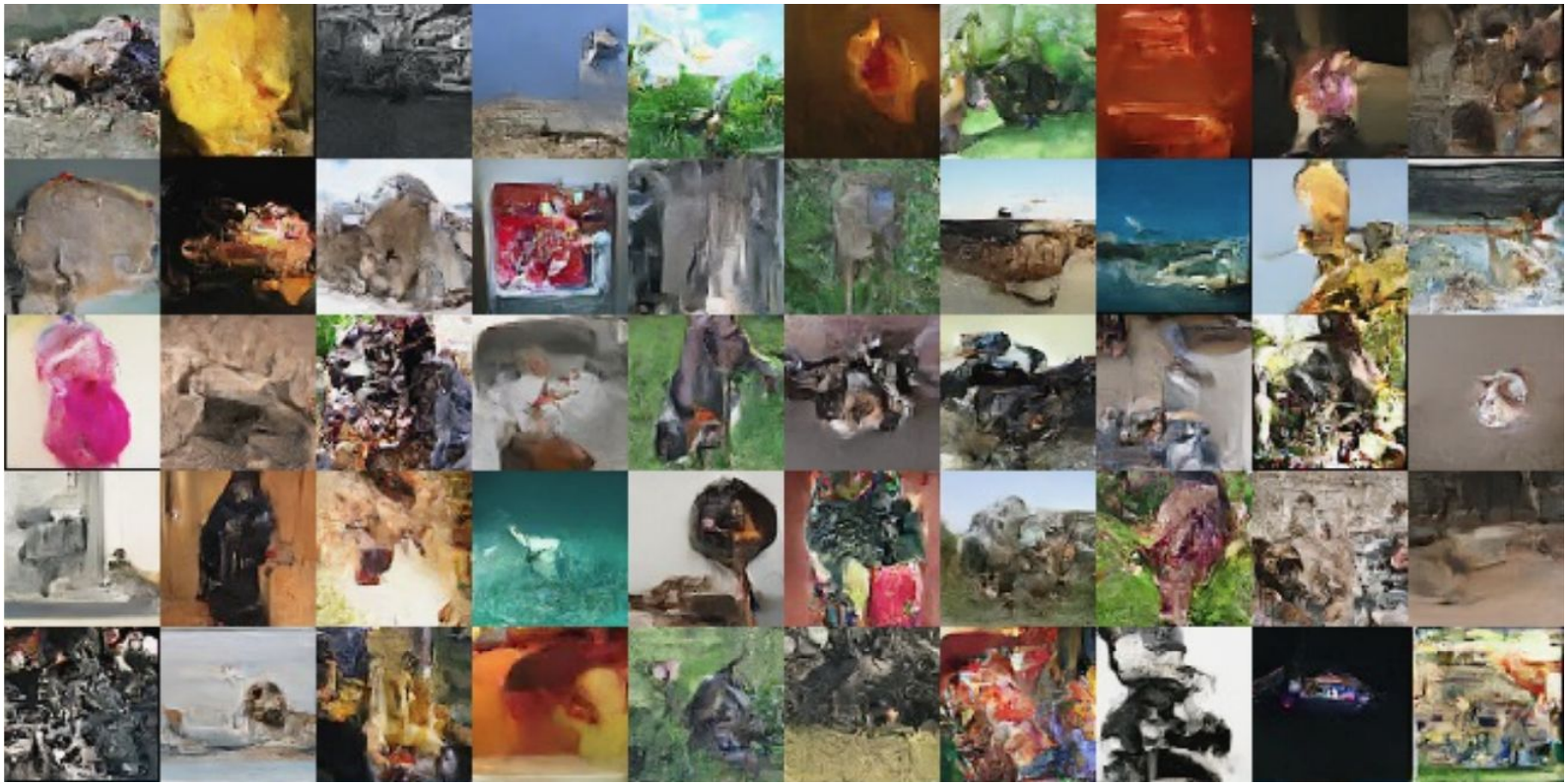
Architecture



Results

Source: Dinh, Laurent, et. al. **"Density Estimation Using Real NVP "** ICLR 2017

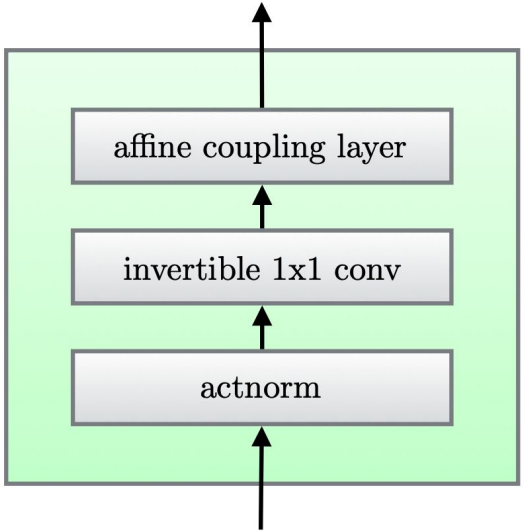
Results



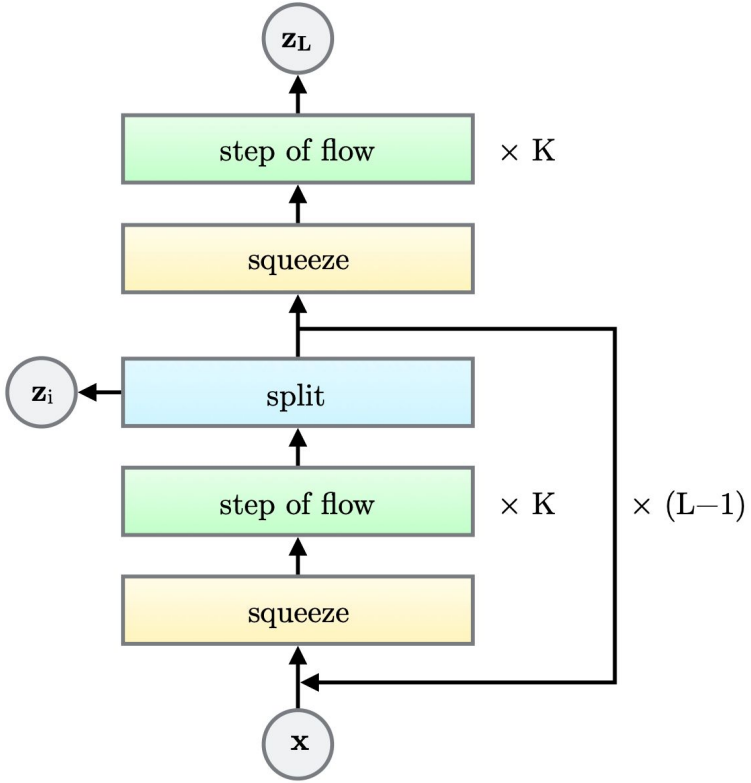
Source: Dinh, Laurent, et. al. **"Density Estimation Using Real NVP "** ICLR 2017

Glow:
Generative Flow
with Invertible 1×1 Convolutions

Architecture



One step of flow



Multi-scale architecture

Source: Kingma, Diederik P., Dhariwal, Prafulla. **"Glow: Generative Flow with Invertible 1×1 Convolutions "** 2018

Activation normalization (actnorm)

Modified version of **batch normalization.**

$$\forall_{i,j} : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$$

Inverse:

$$\forall_{i,j} : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$$

Log-determinant:

$$\log \left(\left| \prod_{i=1}^{h} \prod_{j=1}^{w} \mathbf{s} \right| \right) = \sum_{i=1}^{h} \sum_{j=1}^{w} \log |\mathbf{s}| = h \cdot w \cdot \log |\mathbf{s}|$$

Invertible 1×1 convolution

Convolution with weight matrix $\mathbf{W} : [c \times c]$:

$$\forall_{i,j} : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$$

Inverse:

$$\forall_{i,j} : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$$

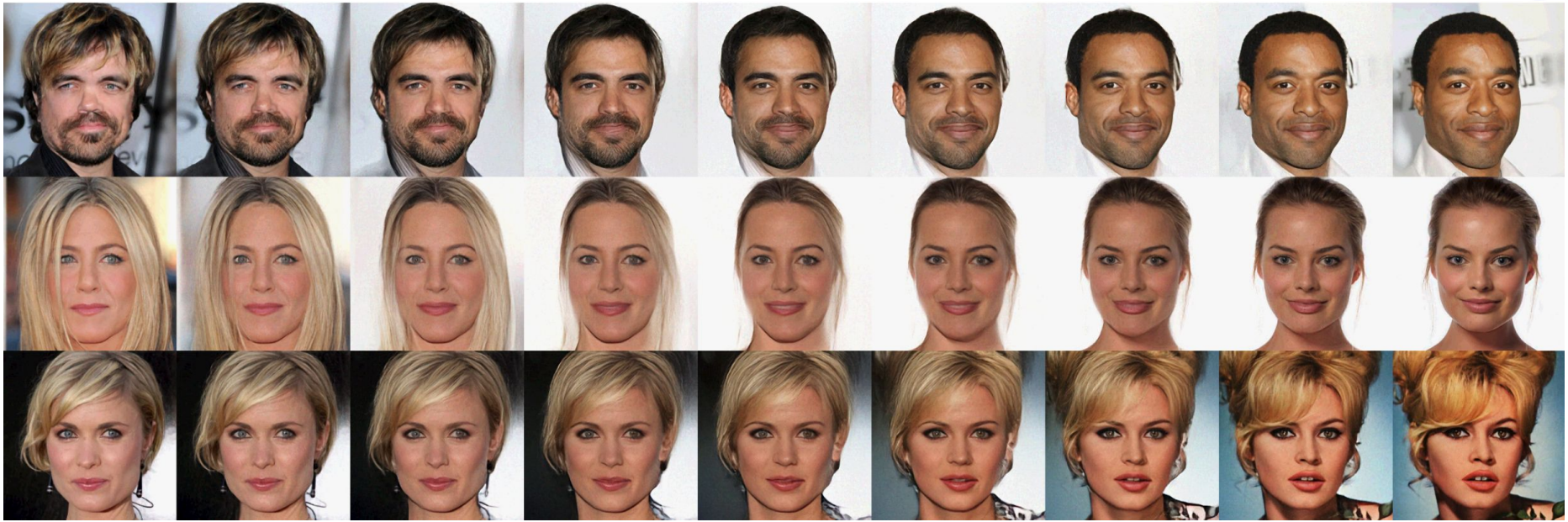
Log-determinant:

$$h \cdot w \cdot \log \left| \det(\mathbf{W}) \right|$$

Results - sampling

Source: Kingma, Diederik P., Dhariwal, Prafulla. **"Glow: Generative Flow with Invertible 1×1 Convolutions "** 2018

Results - interpolation



Source: Kingma, Diederik P., Dhariwal, Prafulla. **"Glow: Generative Flow with Invertible 1×1 Convolutions "** 2018

Results - attributes manipulation

(a) Smiling

(b) Pale Skin

(c) Blond Hair

(d) Narrow Eyes

Source: Kingma, Diederik P., Dhariwal, Prafulla. **"Glow: Generative Flow with Invertible 1×1 Convolutions "** 2018

CIF:
Conditional Invertible Flow for Point Cloud Generation

Introduction

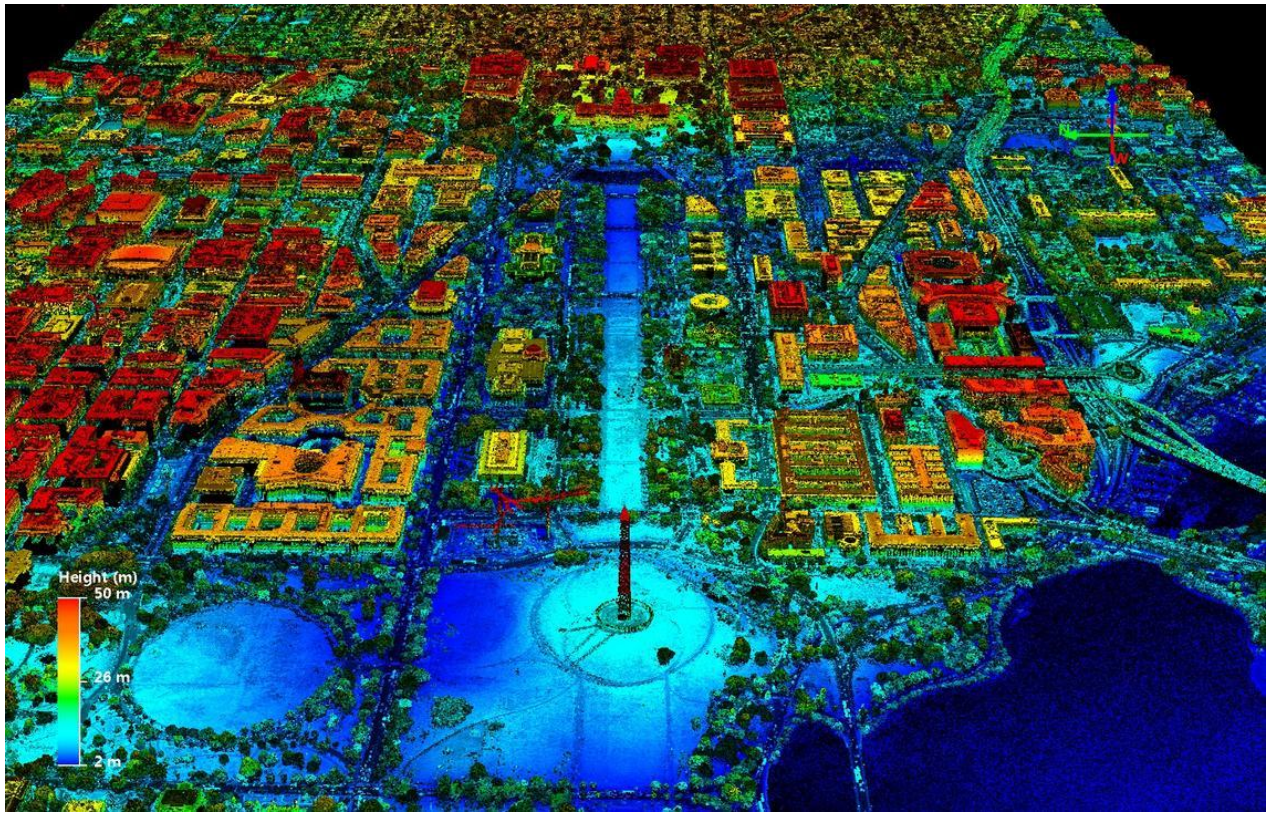
So far we've discussed flow-based models for **image** generation.

Introduction

So far we've discussed flow-based models for **image** generation.

Let's now adapt them to **point clouds!**

Point clouds - what are they?

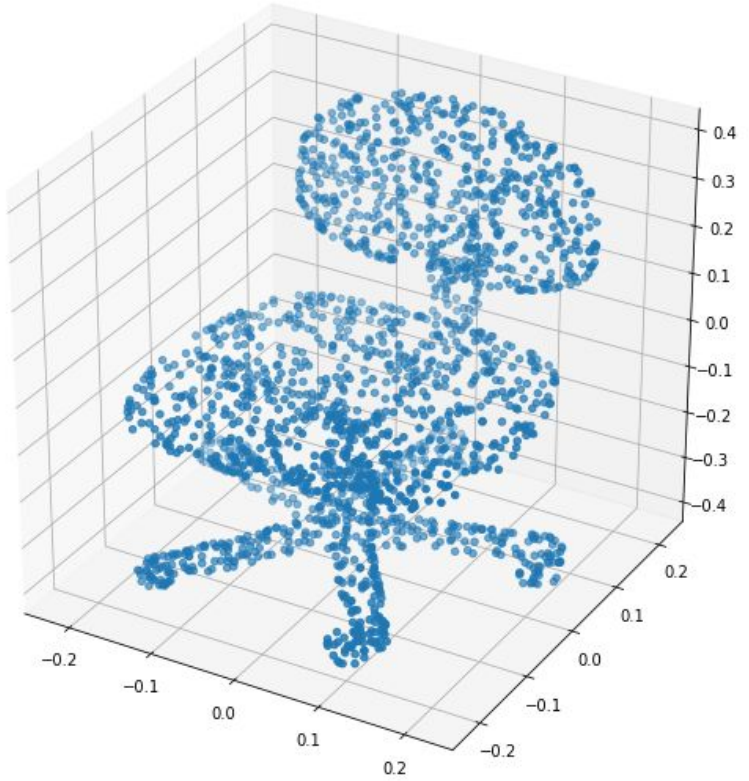


Source: Stoker, Jason. **Lidar point cloud Washington, DC**, 2017, *usgs.gov*

Point clouds - what are they?

Each point $\mathbf{x}$ in a cloud
is a 3D vector, i.e. $\mathbf{x} = (x_1, x_2, x_3)$.

We will denote m-th point from the
n-th cloud as $\mathbf{x}_{n,m}$.



Point clouds - difficulties

- Permutation invariance

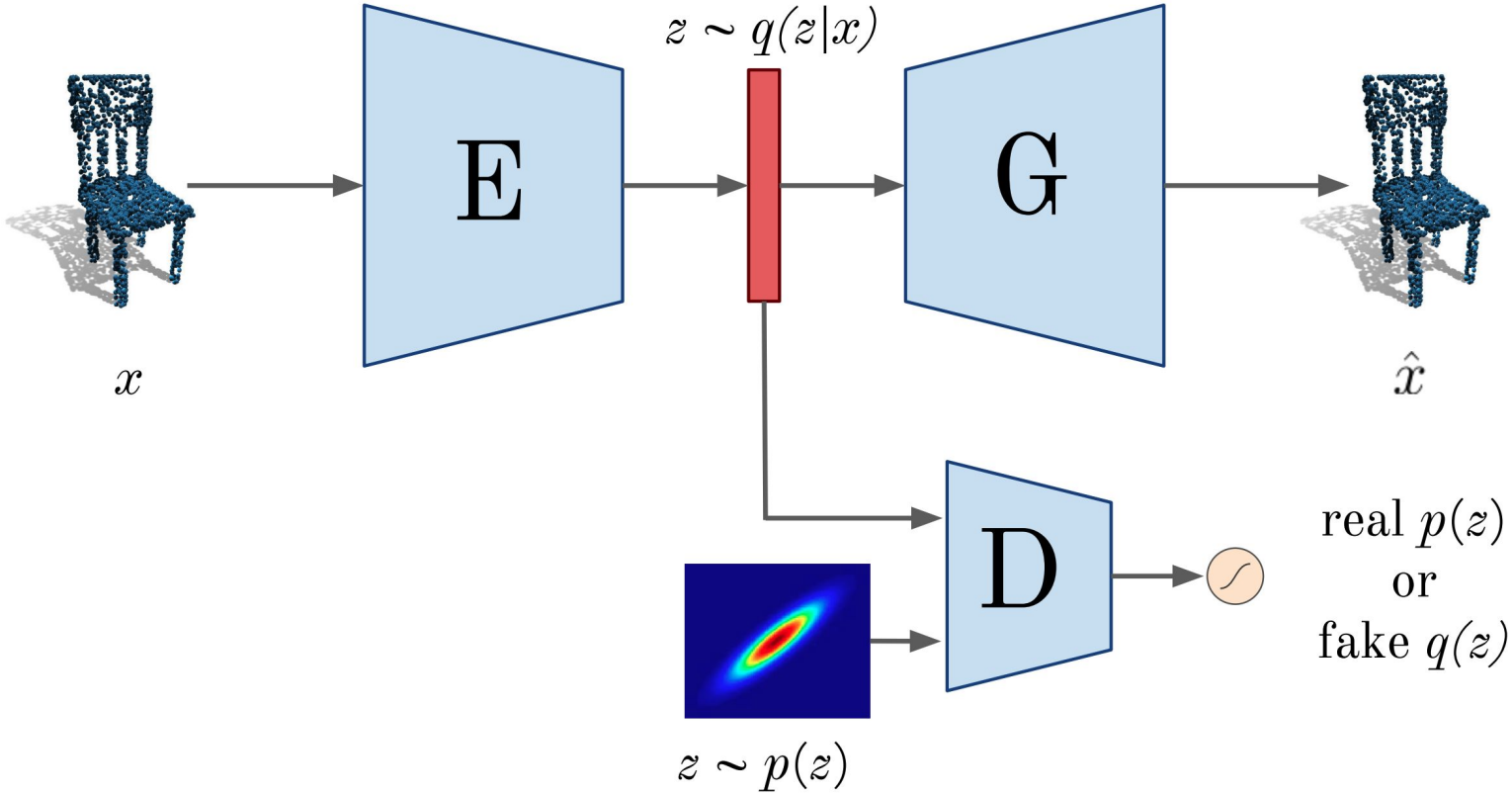
Point clouds - difficulties

- Permutation invariance
- Arbitrary number of points

Point clouds - difficulties

- Permutation invariance
- Arbitrary number of points
- Transformation immutability

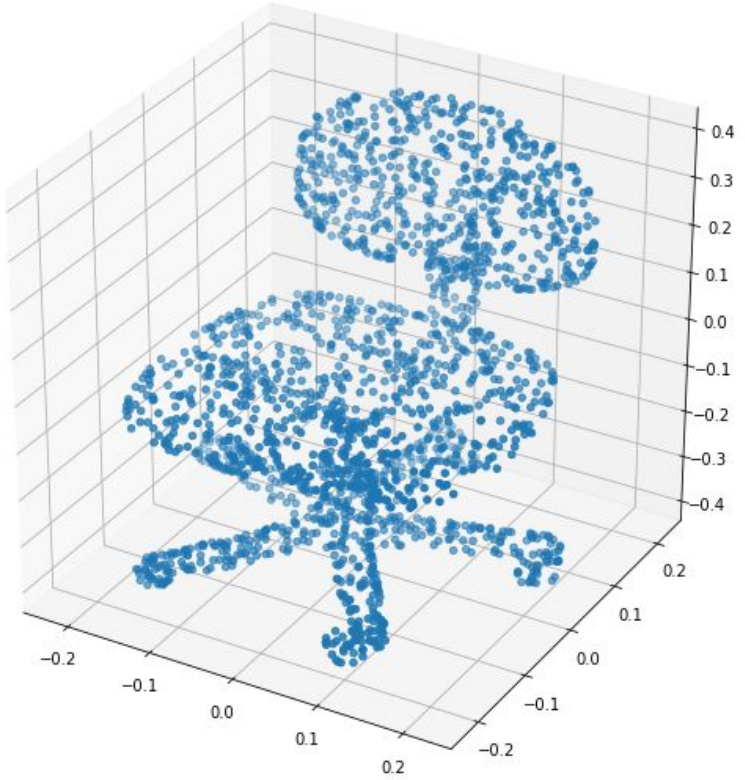
Point clouds - related work



Source: Zamorski, M., Zięba, M., et al. **"Adversarial Autoencoders for Compact Representations of 3D Point Clouds"** 2019

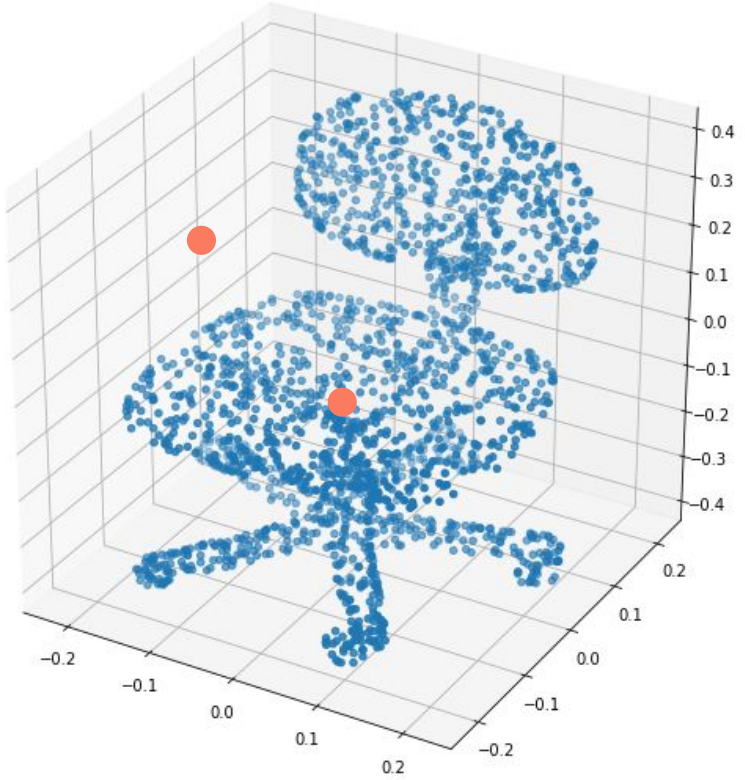
Point-level generation

Think about points as samples from some **distribution over the surface.** Our goal is to find this distribution in order to gain ability to generate **one point at the time.**



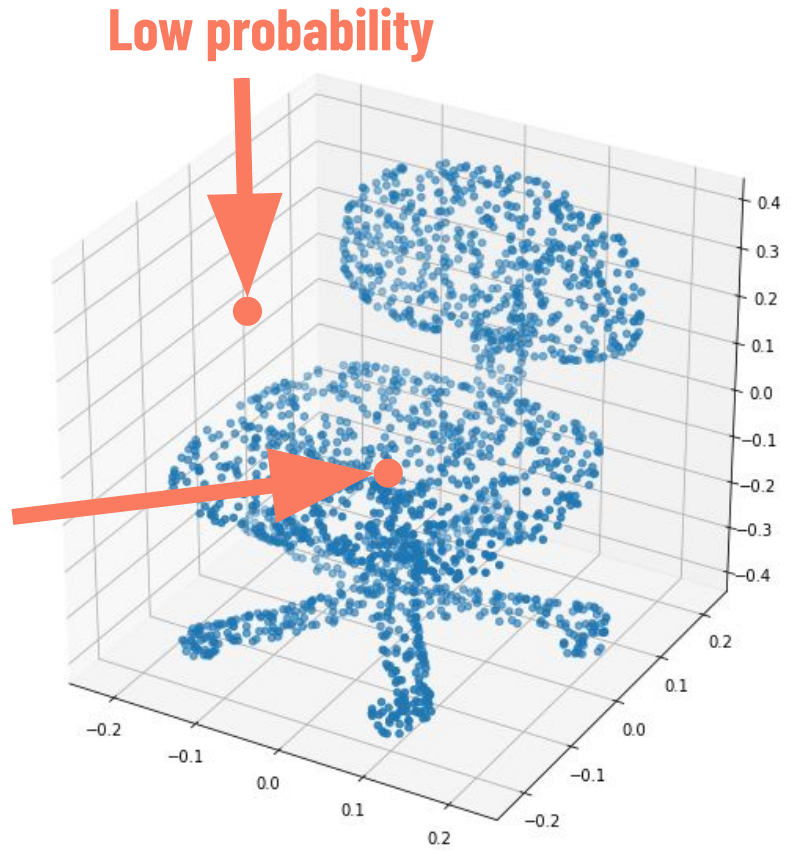
Point-level generation

Think about points as samples from some **distribution over the surface.** Our goal is to find this distribution in order to gain ability to generate **one point at the time.**



Point-level generation

Think about points as samples from some **distribution over the surface.** Our goal is to find this distribution in order to gain ability to generate **one point at the time.**



Point-level generation

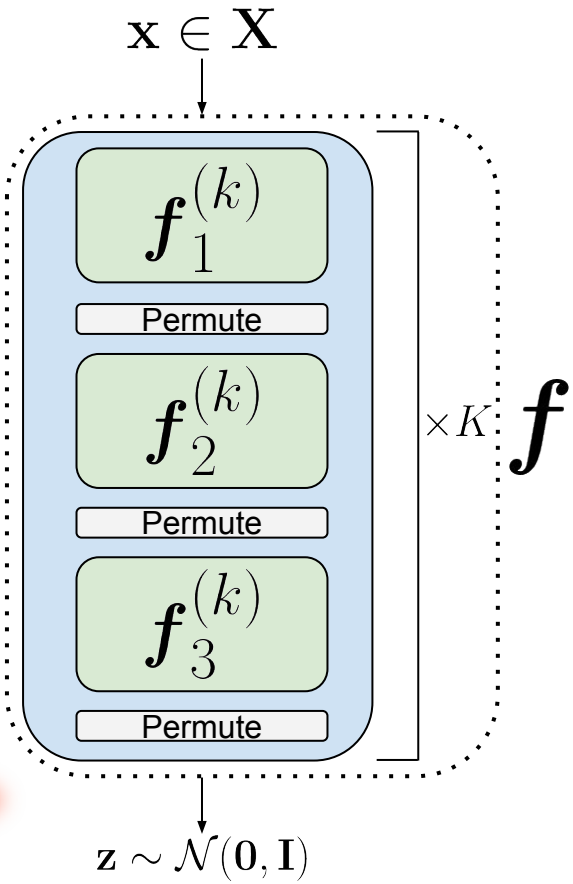
We define **simple flow** $\mathbf{f}_i^{(k)}$ as an affine coupling layer from Real NVP

$$y_{1:d} = x_{1:d}$$
$$y_{d+1:D} = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d})$$

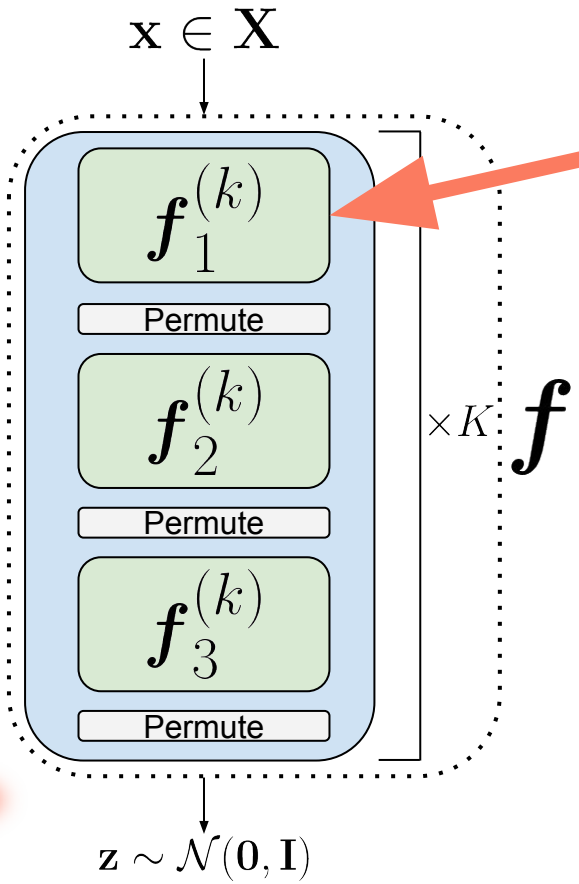
where $D = 3$ and $d = 2$.

Note that in each single flow we transform only **one dimension.**

Point-level generation

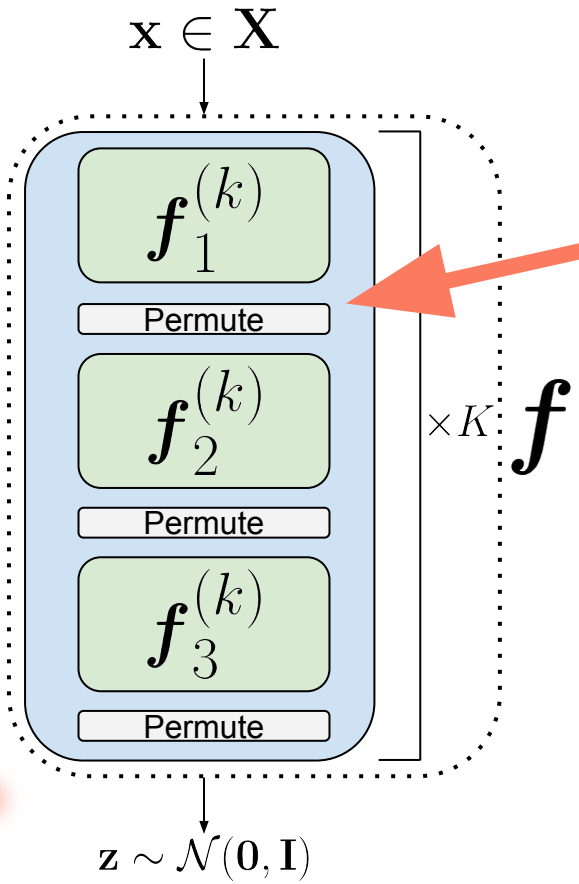


Point-level generation



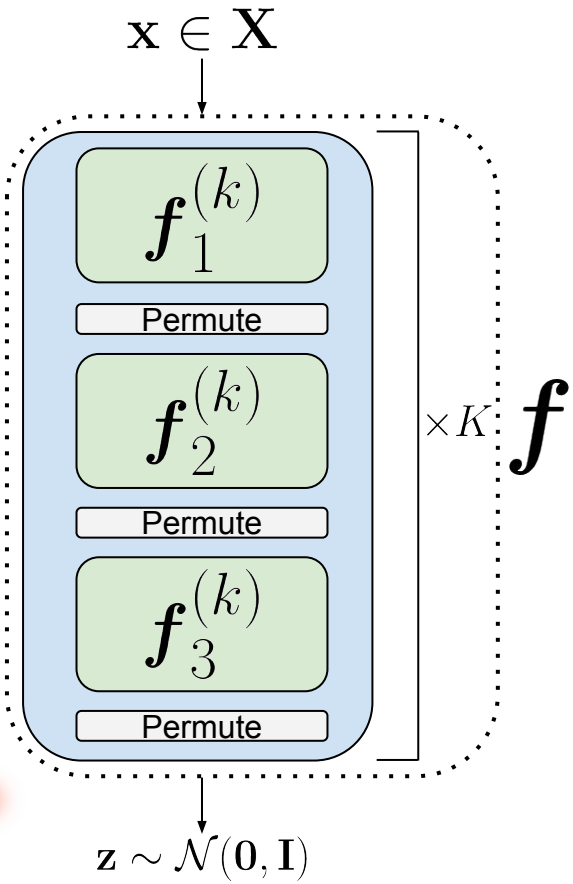
Affine coupling layer from Real NVP (simple flow)

Point-level generation



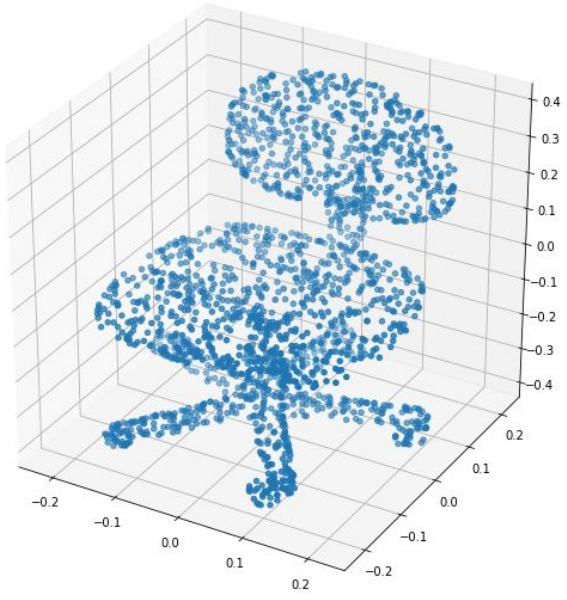
Permutation to affect every dimension

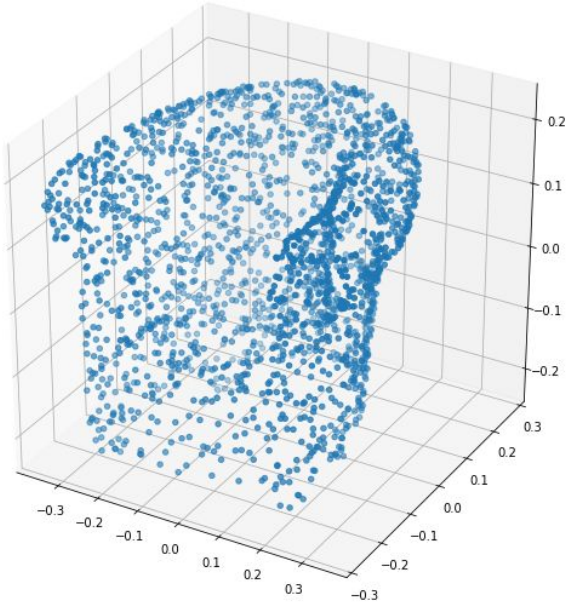
Point-level generation

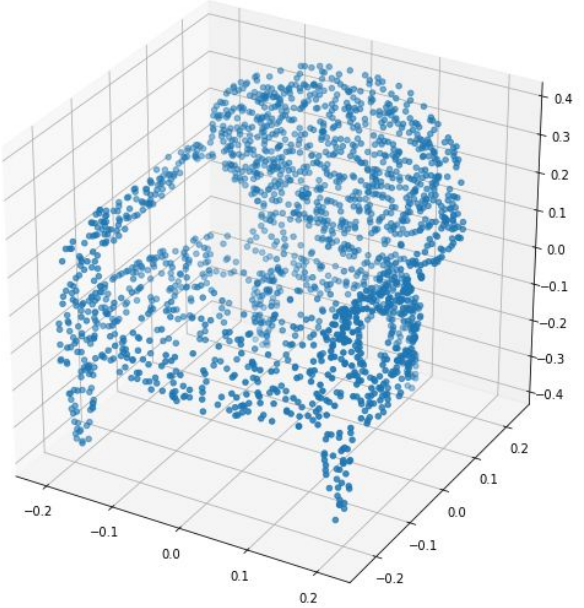


But sampling from just one point cloud seems kind of ... useless.

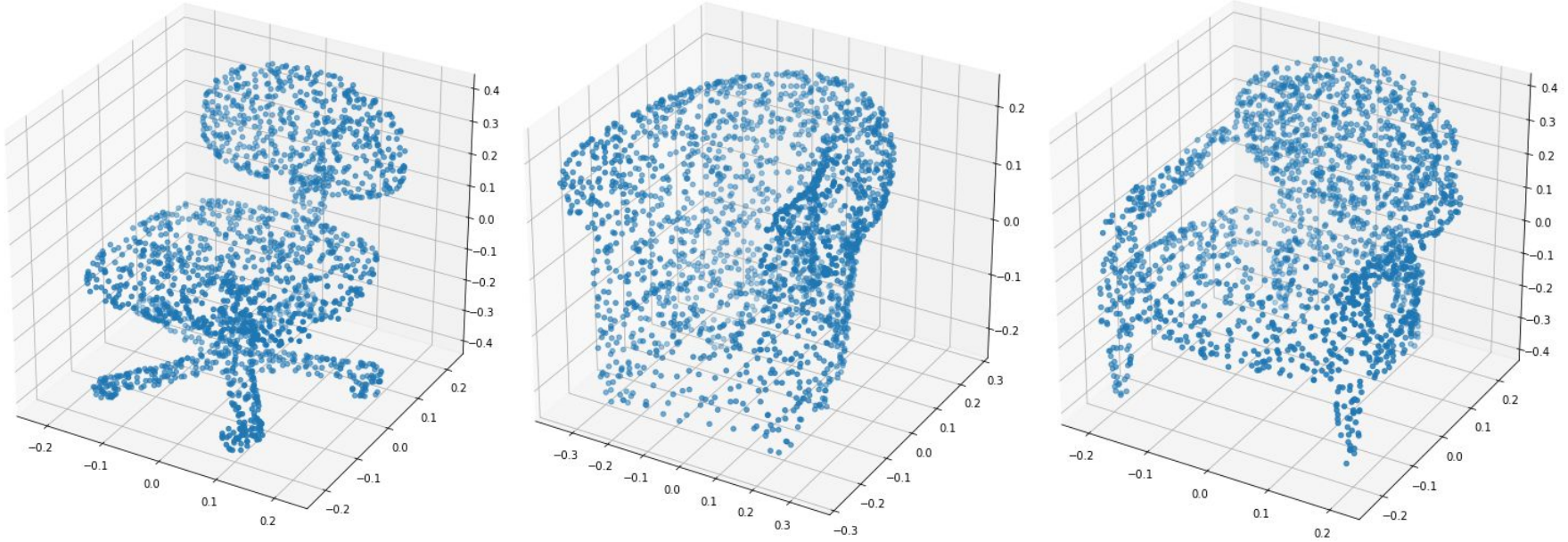
Conditional flow





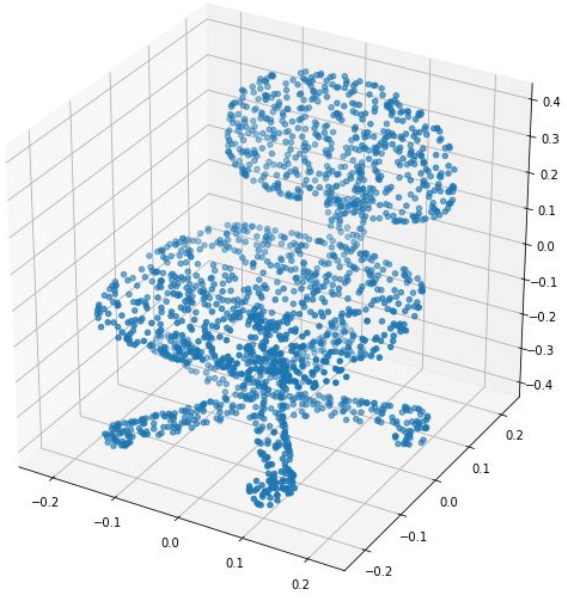


Conditional flow

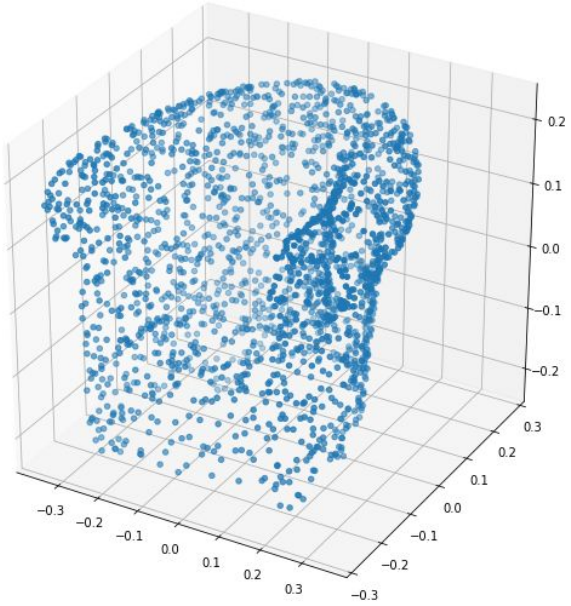


Let’s add some embeddings!

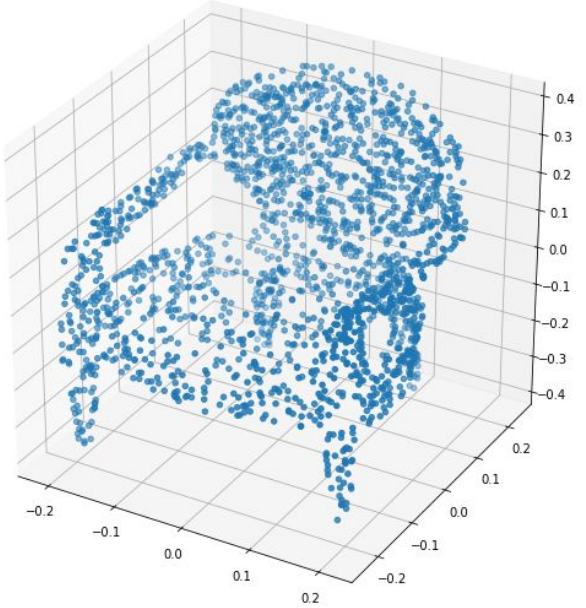
Conditional flow



$\mathbf{e}_1$



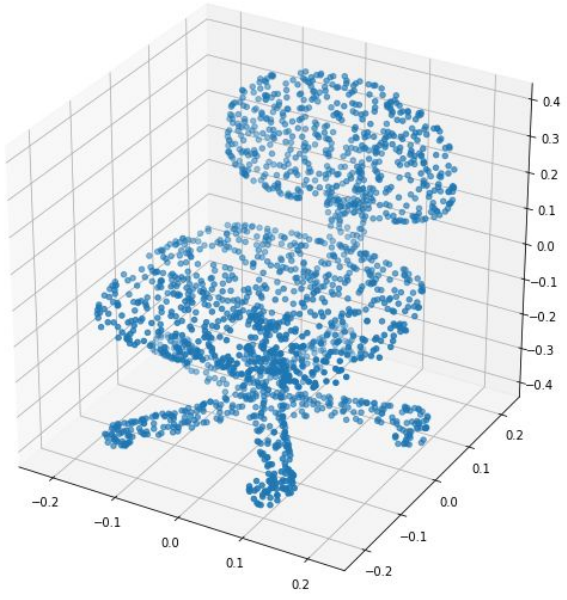
$\mathbf{e}_2$



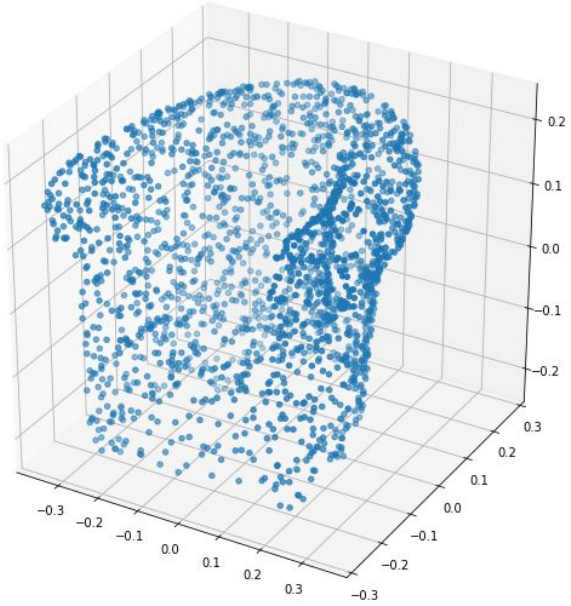
$\mathbf{e}_3$

Let's add some embeddings!

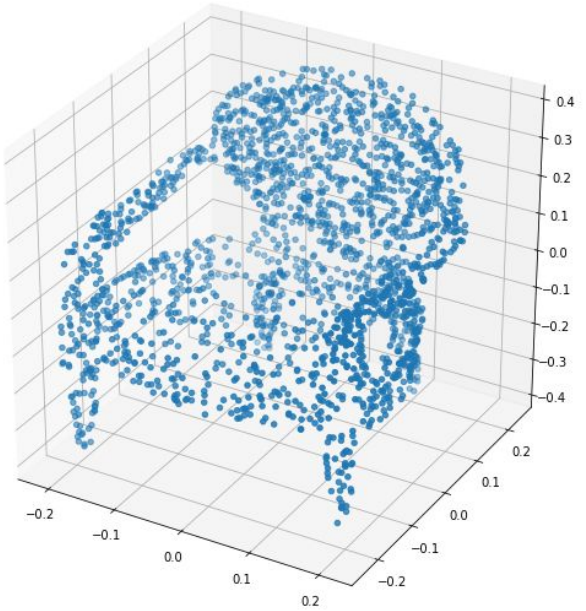
Conditional flow



$\mathbf{e}_1$



$\mathbf{e}_2$



$\mathbf{e}_3$

They will be a conditioning factor in our distribution!

Embeddings

We want **embeddings** to be:

- Normally distributed
- Reasonably placed in the space

Embeddings

We want **embeddings** to be:

- Normally distributed
- Reasonably placed in the space

How to find them?

Embeddings

Chamfer Distance (CD)

$$CD(\mathcal{S}_1, \mathcal{S}_2) = \sum_{\mathbf{x} \in \mathcal{S}_1} \min_{\mathbf{y} \in \mathcal{S}_2} \|\mathbf{x} - \mathbf{y}\|_2^2 + \sum_{\mathbf{y} \in \mathcal{S}_2} \min_{\mathbf{x} \in \mathcal{S}_1} \|\mathbf{x} - \mathbf{y}\|_2^2$$

Multidimensional Scaling (MDS)

We can think of MDS as an algorithm **maintaining similarities** between inputs (distances) in **different number of dimensions.**

Embeddings

Chamfer Distance (CD)

$$CD(\mathcal{S}_1, \mathcal{S}_2) = \sum_{\mathbf{x} \in \mathcal{S}_1} \min_{\mathbf{y} \in \mathcal{S}_2} \|\mathbf{x} - \mathbf{y}\|_2^2 + \sum_{\mathbf{y} \in \mathcal{S}_2} \min_{\mathbf{x} \in \mathcal{S}_1} \|\mathbf{x} - \mathbf{y}\|_2^2$$

Multidimensional Scaling (MDS)

We can think of MDS as an algorithm **maintaining similarities** between inputs (distances) in **different number of dimensions.**

Simple example:
Given the distances between cities, we can estimate their location on the map.

Embeddings

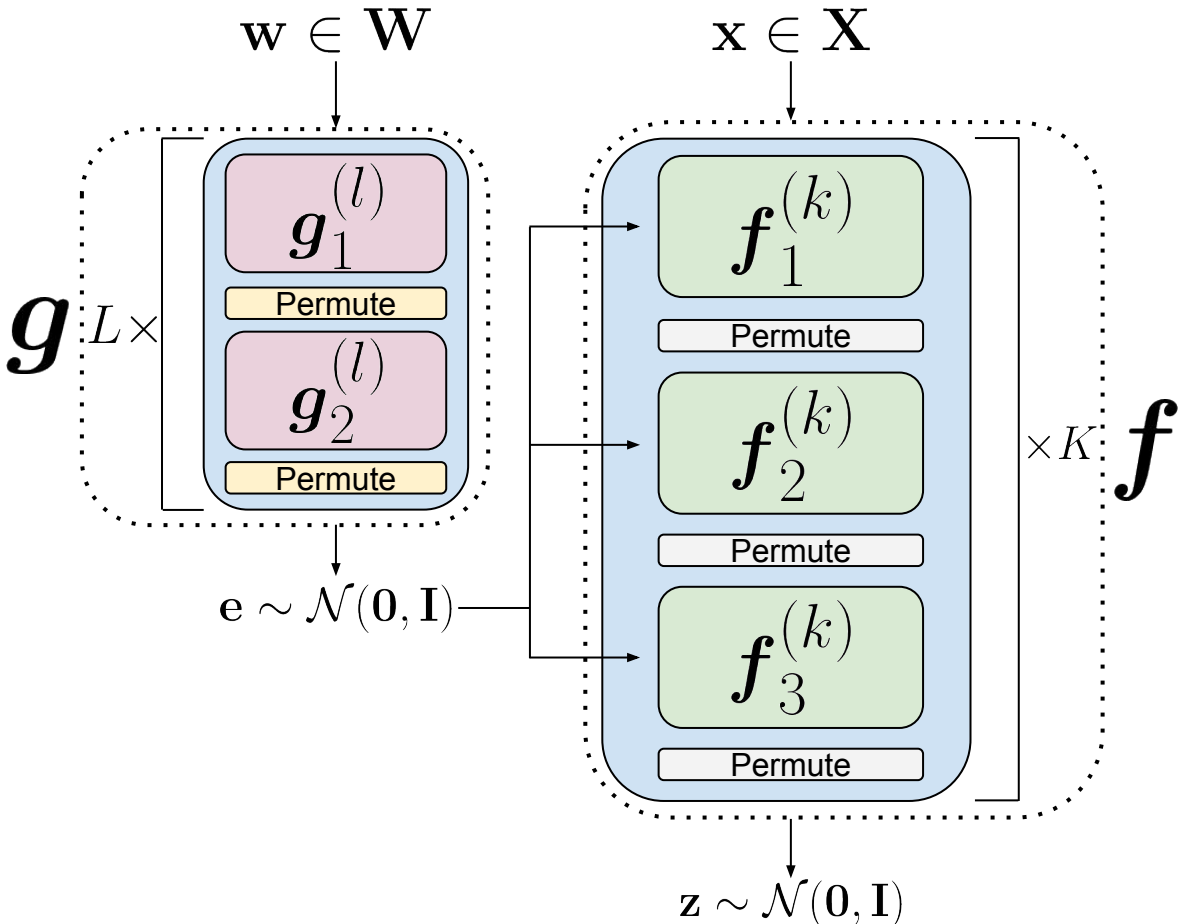
We create $\mathbf{e}$ as follows:

1. Initialize $\mathbf{w}$ using CD and MDS.

1. Add another flow $\mathbf{g}$ mapping from W to E.

1. Train $\mathbf{g}$ simultaneously with $\mathbf{f}$.

Conditional flow



Conditional flow

Given the data point $\mathbf{x}$ and corresponding vector $\mathbf{w}$, our goal is to find the joint distribution $p_{X,W}$.

$$p_{X,W}(\mathbf{x},\mathbf{w}) = p_{Z,E}(\mathbf{f}(\mathbf{x},\mathbf{e}),\mathbf{g}(\mathbf{w})) \cdot \left| \det \begin{pmatrix} \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{e})}{\partial \mathbf{x}^T} & \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{e})}{\partial \mathbf{w}^T} \\ 0 & \frac{\partial \mathbf{g}(\mathbf{w})}{\partial \mathbf{w}^T} \end{pmatrix} \right|$$

$$= p_Z(\mathbf{f}(\mathbf{x},\mathbf{e})) \cdot p_E(\mathbf{g}(\mathbf{w})) \cdot \left| \det \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{e})}{\partial \mathbf{x}^T} \right| \cdot \left| \det \frac{\partial \mathbf{g}(\mathbf{w})}{\partial \mathbf{w}^T} \right|$$

where $\mathbf{z} = \mathbf{f}(\mathbf{x},\mathbf{e})$ and $\mathbf{e} = \mathbf{g}(\mathbf{w})$.

Conditional flow

The **dataset** is composed of set of pairs $(\mathbf{x}_{n,m}, \mathbf{w}_n)$.

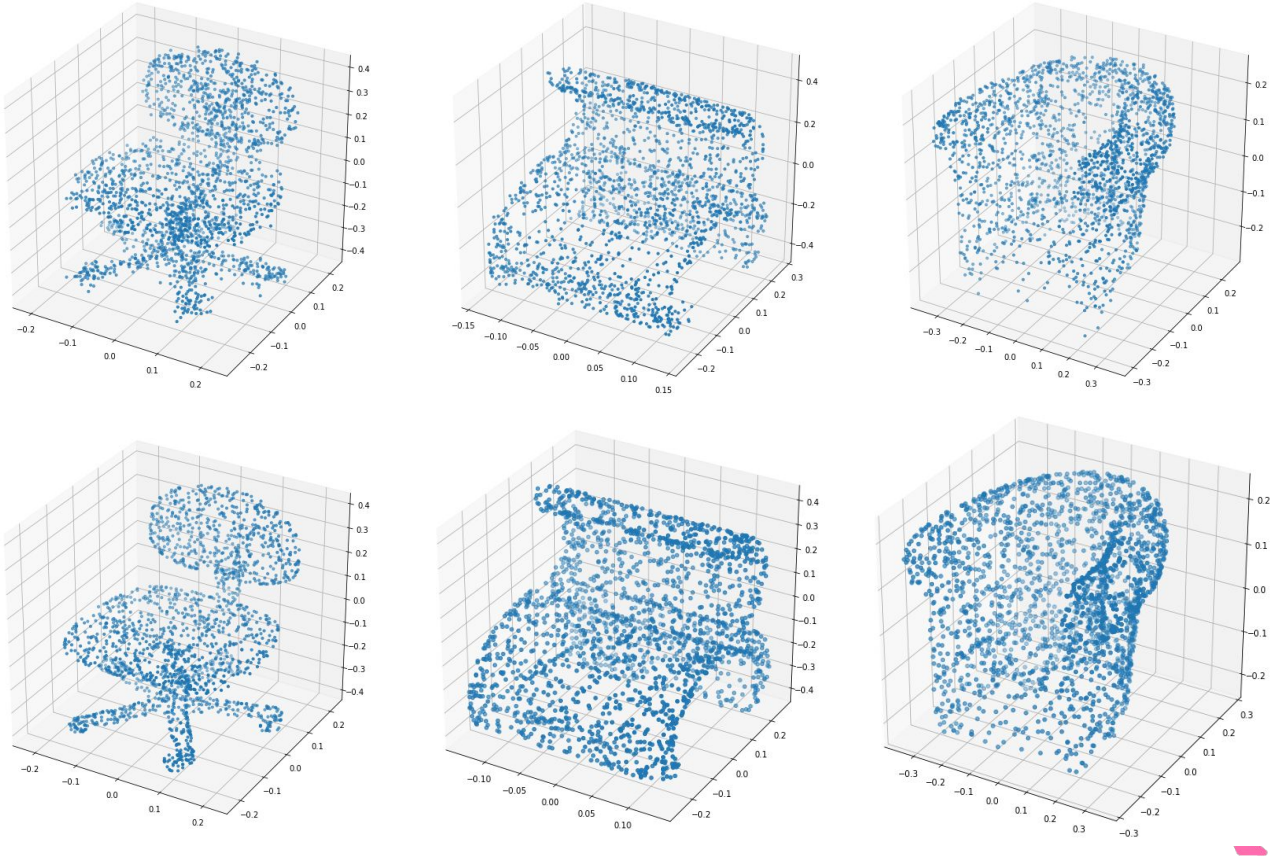
The **training objective** is of the form:

$$\sum_n \sum_m \log\left(p_{X,W}(\mathbf{x}_{n,m}, \mathbf{w}_n)\right) = \sum_n \sum_m \Bigg[\log\left(p_Z(\mathbf{f}(\mathbf{x}_{n,m}, \mathbf{e}_n))\right) + \log\left(p_E(\mathbf{g}(\mathbf{w}_n))\right) + \log\left|\det \frac{\partial \mathbf{f}(\mathbf{x}_{n,m}, \mathbf{e}_n)}{\partial \mathbf{x}^T}\right| + \log\left|\det \frac{\partial \mathbf{g}(\mathbf{w}_n)}{\partial \mathbf{w}^T}\right| \Bigg].$$

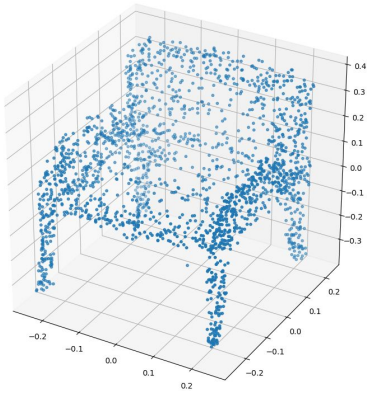
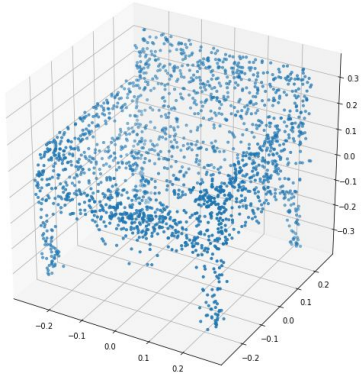
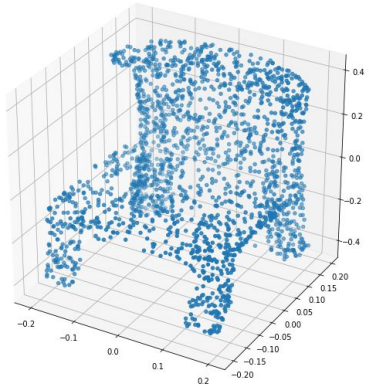
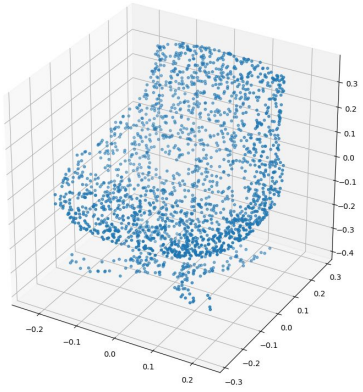
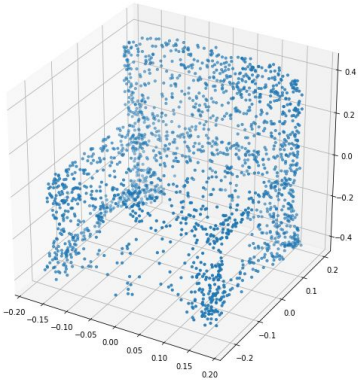
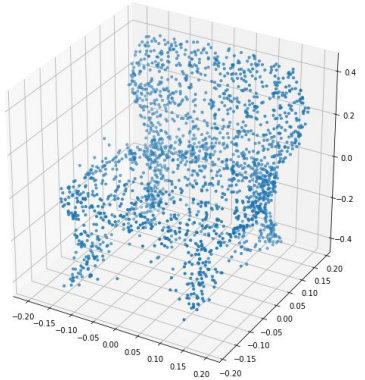
Experiments

- **Reconstruction**
  - Training set
  - Test set
- **Sampling**
- **Interpolation**

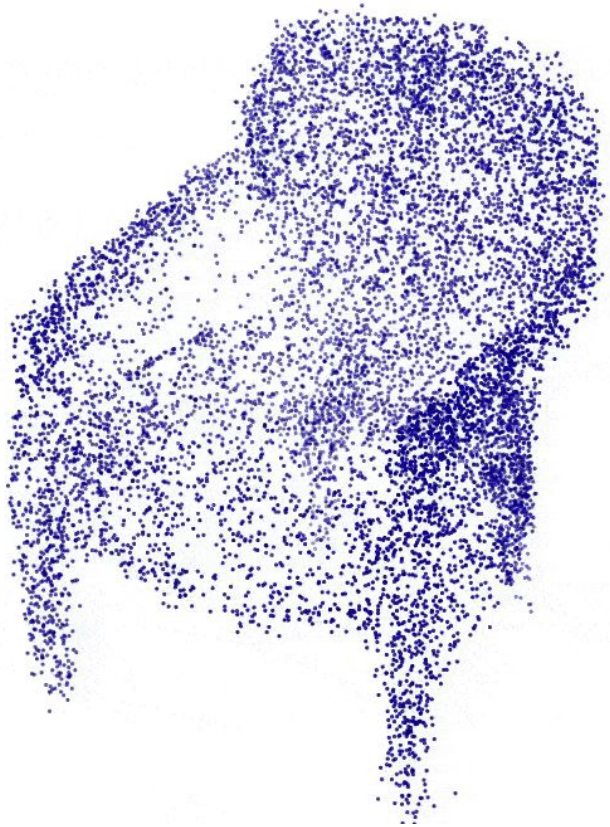
Results



Results

Results



TOOPLOOX

www.tooploox.com