

# Deep Generative Models

## Lecture 4

Roman Isachenko

Ozon Masters

2021

## Likelihood-based models so far...

### Autoregressive models

$$p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{i=1}^m p(x_i|\mathbf{x}_{1:i-1}, \boldsymbol{\theta})$$

- ▶ tractable likelihood,
- ▶ no inferred latent factors.

### Latent variable models

$$p(\mathbf{x}|\boldsymbol{\theta}) = \int p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) d\mathbf{z}$$

- ▶ latent feature representation,
- ▶ intractable likelihood.

How to build model with latent variables and tractable likelihood?

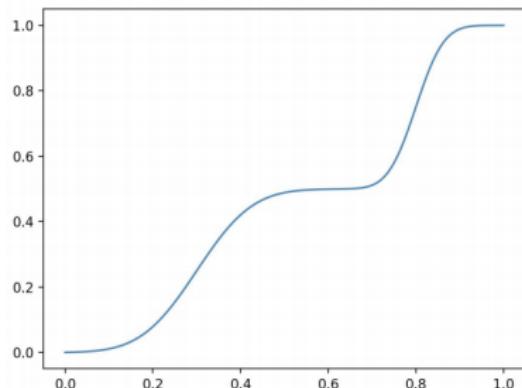
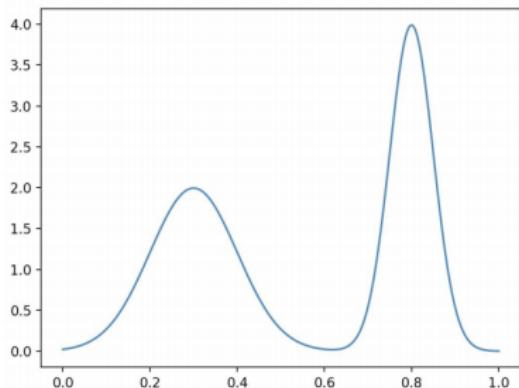
## Flows intuition

Let  $X$  be a random variable with density  $p_X(x)$ . Then

$$Z = F(X) = \int_{-\infty}^x p(t)dt \sim U[0, 1].$$

Hence

$$Z \sim U[0, 1]; \quad X = F^{-1}(Z) \quad X \sim p(x).$$



# Change of variables

## Theorem

Let

- ▶  $\mathbf{x}$  is a random variable,
- ▶  $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$  is a differentiable, invertible function,
- ▶  $\mathbf{z} = f(\mathbf{x})$ ,  $\mathbf{x} = f^{-1}(\mathbf{z}) = g(\mathbf{z})$ .

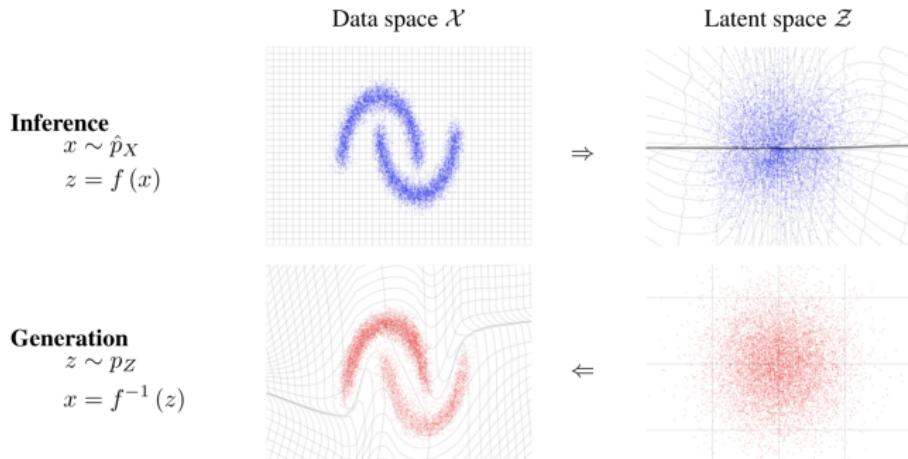
Then

$$p(\mathbf{x}) = p(\mathbf{z}) \left| \det \left( \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) \right| = p(f(\mathbf{x})) \left| \det \left( \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right) \right|.$$

## Note

- ▶  $\mathbf{x}$  and  $\mathbf{z}$  have the same dimensionality;
- ▶  $\left| \det \left( \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right) \right| = \left| \det \left( \frac{\partial g^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right| = \left| \det \left( \frac{\partial g(\mathbf{z})}{\partial \mathbf{z}} \right) \right|^{-1}$ ;
- ▶  $f(\mathbf{x}, \theta)$  could be parametric.

# Flows



- ▶ Latent representation is given by  $\mathbf{z} = f(\mathbf{x}, \theta)$ .
- ▶ Likelihood is given by  $\mathbf{z} = f(\mathbf{x}, \theta)$  and change of variables.
- ▶ Sampling of  $\mathbf{x}$  is performed by sampling from a base distribution  $p(\mathbf{z})$  and applying  $\mathbf{x} = f^{-1}(\mathbf{z}, \theta) = g(\mathbf{z}, \theta)$ .

# Fitting flows

## MLE problem

$$\boldsymbol{\theta}^* = \arg \max_{\boldsymbol{\theta}} p(\mathbf{X}|\boldsymbol{\theta}) = \arg \max_{\boldsymbol{\theta}} \prod_{i=1}^n p(\mathbf{x}_i|\boldsymbol{\theta}) = \arg \max_{\boldsymbol{\theta}} \sum_{i=1}^n \log p(\mathbf{x}_i|\boldsymbol{\theta}).$$

## Challenge

$p(\mathbf{x}|\boldsymbol{\theta})$  could be intractable.

## Fitting flow to solve MLE

$$p(\mathbf{x}|\boldsymbol{\theta}) = p(\mathbf{z}) \left| \det \left( \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) \right| = p(f(\mathbf{x}, \boldsymbol{\theta})) \left| \det \left( \frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}} \right) \right|$$

## Stacking flows

Let  $\mathbf{z} = f(\mathbf{x}) = f_2 \circ f_1(\mathbf{x})$  and  $f_1, f_2$  satisfy conditions of change of variable theorem (differentiable and invertible).

$$\begin{aligned} p(\mathbf{x}) &= p(\mathbf{z}) \left| \det \left( \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) \right| = p(f(\mathbf{x})) \left| \det \left( \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right) \right| = \\ &= p(f(\mathbf{x})) \left| \det \left( \frac{\partial f_2 \circ f_1(\mathbf{x})}{\partial \mathbf{x}} \right) \right| = p(f(\mathbf{x})) \left| \det \left( \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1} \cdot \frac{\partial \mathbf{f}_1}{\partial \mathbf{x}} \right) \right| = \\ &\quad = p(f(\mathbf{x})) \left| \det \left( \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1} \right) \right| \cdot \left| \det \left( \frac{\partial \mathbf{f}_1}{\partial \mathbf{x}} \right) \right| \end{aligned}$$

What we will get in the case  $\mathbf{z} = f(\mathbf{x}) = f_n \circ \cdots \circ f_1(\mathbf{x})$ ?

# Flows

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f(\mathbf{x}, \boldsymbol{\theta})) + \log \left| \det \left( \frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}} \right) \right|$$

## Definition

Normalizing flow is a *differentiable, invertible* mapping from data  $\mathbf{x}$  to the noise  $\mathbf{z}$ .

- ▶ Normalizing - convert data distribution to *noise*.
- ▶ Flow - sequence of such mapping is also a flow

$$\mathbf{z} = f_K \circ \cdots \circ f_1(\mathbf{x}); \quad \mathbf{x} = f_1^{-1} \circ \cdots \circ f_K^{-1}(\mathbf{z}) = g_1 \circ \cdots \circ g_K(\mathbf{z})$$

$$\begin{aligned} p(\mathbf{x}) &= p(f_K \circ \cdots \circ f_1(\mathbf{x})) \left| \det \left( \frac{\partial f_K \circ \cdots \circ f_1(\mathbf{x})}{\partial \mathbf{x}} \right) \right| = \\ &= p(f_K \circ \cdots \circ f_1(\mathbf{x})) \prod_{k=1}^K \left| \det \left( \frac{\partial \mathbf{f}_k}{\partial \mathbf{f}_{k-1}} \right) \right|. \end{aligned}$$

# Flows

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(f(\mathbf{x}, \boldsymbol{\theta})) + \log \left| \det \left( \frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}} \right) \right|$$

What is the computational complexity of computing the determinant?

## What we want

- ▶ Efficient computation of Jacobian  $\frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}}$ ;
- ▶ Efficient sampling from the base distribution  $p(\mathbf{z})$ ;
- ▶ Easy to invert  $f(\mathbf{x}, \boldsymbol{\theta})$ .

# Planar Flows, 2015

$$g(\mathbf{z}, \theta) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T \mathbf{z} + b).$$

- ▶  $\theta = \{\mathbf{u}, \mathbf{w}, b\}$ ;
- ▶  $h$  is a smooth element-wise non-linearity.

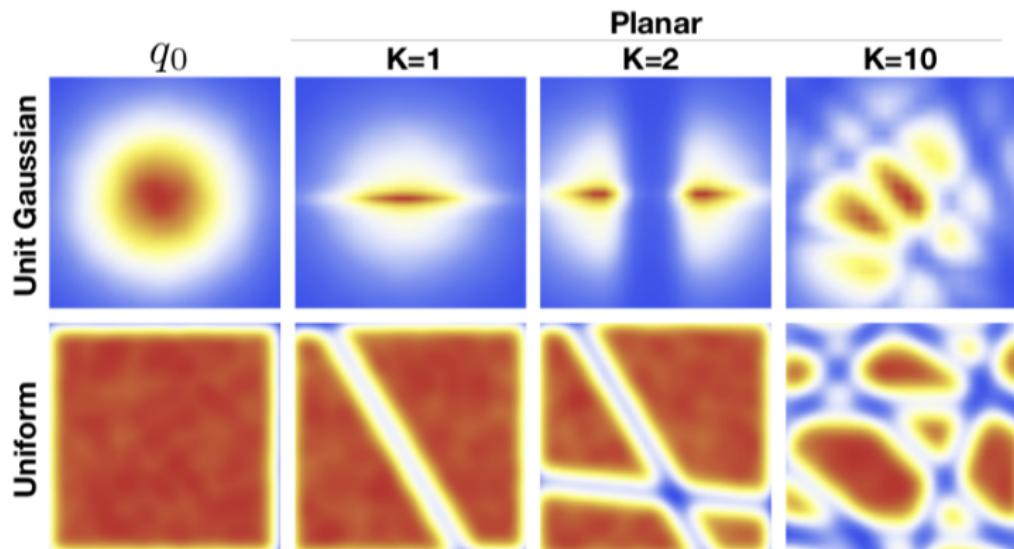
$$\begin{aligned}\left| \det \left( \frac{\partial g(\mathbf{z}, \theta)}{\partial \mathbf{z}} \right) \right| &= \left| \det \left( \mathbf{I} + h'(\mathbf{w}^T \mathbf{z} + b) \mathbf{w} \mathbf{u}^T \right) \right| \\ &= \left| 1 + h'(\mathbf{w}^T \mathbf{z} + b) \mathbf{w}^T \mathbf{u} \right|\end{aligned}$$

The transformation is invertible if (just one of example)

$$h = \tanh; \quad h'(\mathbf{w}^T \mathbf{z} + b) \mathbf{u}^T \mathbf{w} \geq -1.$$

# Planar Flows, 2015

$$\mathbf{z}_K = g_1 \circ \cdots \circ g_K(\mathbf{z}); \quad g_k = g(\mathbf{z}_k, \theta_k).$$



## Jacobian structure

- ▶ What is the determinant of a diagonal matrix?

$$\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta}) = (f_1(x_1, \boldsymbol{\theta}), \dots, f_m(x_m, \boldsymbol{\theta})).$$

$$\log \left| \det \left( \frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}} \right) \right| = \log \left| \prod_{i=1}^m f'_i(x_i, \boldsymbol{\theta}) \right| = \sum_{i=1}^m \log |f'_i(x_i, \boldsymbol{\theta})|.$$

- ▶ What is the determinant of a triangular matrix?

Let  $z_i$  depends only on  $\mathbf{x}_{1:i}$  (or without loss of generality  $x_i$  depends on  $\mathbf{z}_{1:i}$ ).

What is the inverse of such transformations?

## Coupling layer

$$\begin{cases} \mathbf{z}_{1:d} = \mathbf{x}_{1:d} \\ \mathbf{z}_{d:m} = \tau(\mathbf{x}_{d:m}, c(\mathbf{x}_{1:d})) \end{cases} \quad \begin{cases} \mathbf{x}_{1:d} = \mathbf{z}_{1:d} \\ \mathbf{x}_{d:m} = \tau^{-1}(\mathbf{z}_{d:m}, c(\mathbf{z}_{1:d})) \end{cases}$$

- ▶  $c : \mathbb{R}^d \rightarrow \mathbb{R}^k$  – coupling function;
- ▶  $\tau : \mathbb{R}^{m-d} \times c(\mathbb{R}^d) \rightarrow \mathbb{R}^{m-d}$  – coupling law.
- ▶

$$\det \left( \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) = \det \begin{pmatrix} \mathbf{I}_d & \mathbf{0}_{d \times m-d} \\ \frac{\partial \mathbf{z}_{d:m}}{\partial \mathbf{x}_{1:d}} & \frac{\partial \mathbf{z}_{d:m}}{\partial \mathbf{x}_{d:m}} \end{pmatrix} = \det \left( \frac{\partial \mathbf{z}_{d:m}}{\partial \mathbf{x}_{d:m}} \right)$$

## Coupling layer

$$\begin{cases} \mathbf{z}_{1:d} = \mathbf{x}_{1:d}; \\ \mathbf{z}_{d:m} = \tau(\mathbf{x}_{d:m}, c(\mathbf{x}_{1:d})); \end{cases} \Rightarrow \begin{cases} \mathbf{x}_{1:d} = \mathbf{z}_{1:d}; \\ \mathbf{x}_{d:m} = \tau^{-1}(\mathbf{z}_{d:m}, c(\mathbf{z}_{1:d})). \end{cases}$$

Coupling function  $c(\cdot)$ 

Any complex function (without restrictions). For example, neural network.

Coupling law  $\tau(\cdot, \cdot)$ 

- ▶  $\tau(x, c) = x + c$  – *additive*;
- ▶  $\tau(x, c) = x \odot c, c \neq 0$  – *multiplicative*;
- ▶  $\tau(x, c) = x \odot c_1 + c_2, c_1 \neq 0$  – *affine*.

To obtain more flexible class of distributions, stack more coupling layers (with different ordering of components!).

$$\det \left( \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) = \det \begin{pmatrix} \mathbf{I}_d & \mathbf{0}_{d \times m-d} \\ \frac{\partial \mathbf{z}_{d:m}}{\partial \mathbf{x}_{1:d}} & \frac{\partial \mathbf{z}_{d:m}}{\partial \mathbf{x}_{d:m}} \end{pmatrix} = \det \left( \frac{\partial \mathbf{z}_{d:m}}{\partial \mathbf{x}_{d:m}} \right)$$

What is the Jacobian for the additive coupling law

$$\tau(x + c) = x + c?$$

In this case the transformation is *volume preserving*.

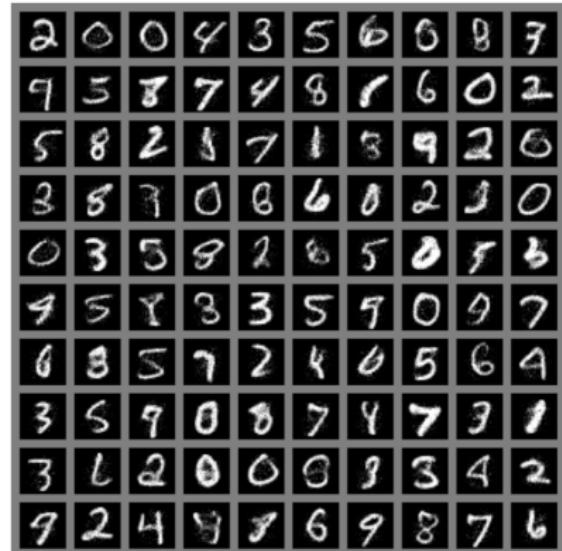
The last layer is rescaling:

$$z_i = s_i x_i; \quad x_i = z_i / s_i.$$

What is the Jacobian of the last layer?

---

<https://arxiv.org/pdf/1410.8516.pdf>



(a) Model trained on MNIST



(b) Model trained on TFD

## Affine coupling law

$$\begin{cases} \mathbf{z}_{1:d} = \mathbf{x}_{1:d}; \\ \mathbf{z}_{d:m} = \mathbf{x}_{d:m} \odot \exp(c_1(\mathbf{x}_{1:d}, \theta)) + c_2(\mathbf{x}_{1:d}, \theta). \end{cases}$$

$$\begin{cases} \mathbf{x}_{1:d} = \mathbf{z}_{1:d}; \\ \mathbf{x}_{d:m} = (\mathbf{z}_{d:m} - c_2(\mathbf{x}_{1:d}, \theta)) \odot \exp(-c_1(\mathbf{x}_{1:d}, \theta)). \end{cases}$$

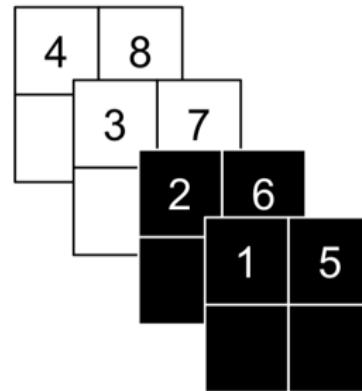
## Jacobian

$$\det\left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right) = \det\begin{pmatrix} \mathbf{I}_d & \mathbf{0}_{d \times m-d} \\ \frac{\partial \mathbf{z}_{d:m}}{\partial \mathbf{x}_{1:d}} & \frac{\partial \mathbf{z}_{d:m}}{\partial \mathbf{x}_{d:m}} \end{pmatrix} = \prod_{i=1}^{m-d} \exp(c_1(\mathbf{x}_{1:d}, \theta)_i).$$

## Non-Volume Preserving.

---

<https://arxiv.org/pdf/1605.08803.pdf>

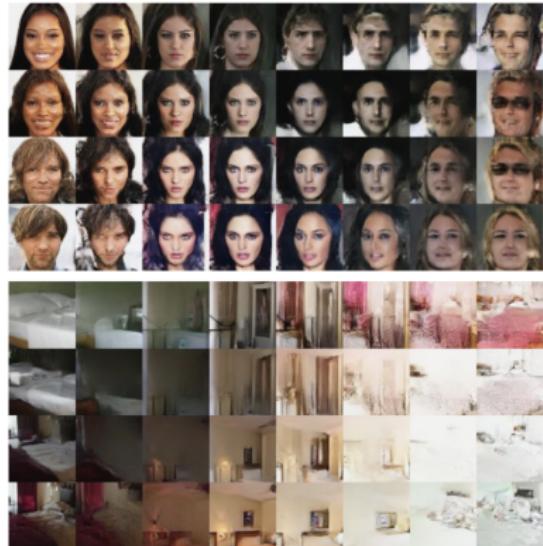


Masked convolutions are used to define ordering.

---

<https://arxiv.org/pdf/1605.08803.pdf>

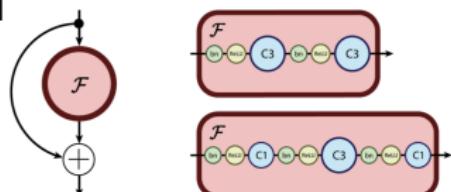
# RealNVP, 2016



---

<https://arxiv.org/pdf/1605.08803.pdf>

- ▶ Modern neural networks are trained via backpropagation.
- ▶ Residual networks are state of the art in image classification.
- ▶ Backpropagation requires storing the network activations.



## Problem

Storing the activations imposes an increasing memory burden. GPUs have limited memory capacity, leading to constraints often exceeded by state-of-the-art architectures (with thousand layers).

---

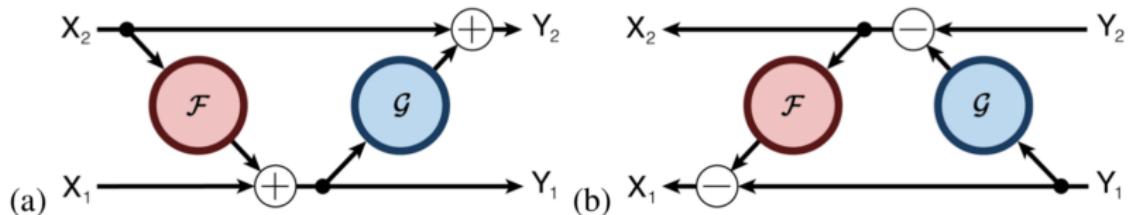
<https://arxiv.org/pdf/1707.04585.pdf>

## NICE

$$\begin{cases} \mathbf{z}_1 = \mathbf{x}_1; \\ \mathbf{z}_2 = \mathbf{x}_2 + \mathcal{F}(\mathbf{x}_1, \theta); \end{cases} \Leftrightarrow \begin{cases} \mathbf{x}_1 = \mathbf{z}_1; \\ \mathbf{x}_2 = \mathbf{z}_2 - \mathcal{F}(\mathbf{z}_1, \theta). \end{cases}$$

## RevNet

$$\begin{cases} \mathbf{y}_1 = \mathbf{x}_1 + \mathcal{F}(\mathbf{x}_2, \theta); \\ \mathbf{y}_2 = \mathbf{x}_2 + \mathcal{G}(\mathbf{y}_1, \theta); \end{cases} \Leftrightarrow \begin{cases} \mathbf{x}_2 = \mathbf{y}_2 - \mathcal{F}(\mathbf{y}_1, \theta); \\ \mathbf{x}_1 = \mathbf{y}_1 - \mathcal{G}(\mathbf{x}_2, \theta). \end{cases}$$



Architecture	CIFAR-10 [15]		CIFAR-100 [15]	
	ResNet	RevNet	ResNet	RevNet
32 (38)	<b>7.14%</b>	7.24%	29.95%	<b>28.96%</b>
110	<b>5.74%</b>	5.76%	26.44%	<b>25.40%</b>
164	5.24%	<b>5.17%</b>	<b>23.37%</b>	23.69%

- ▶ If the network contains non-reversible blocks (poolings, strides), activations for these blocks should be stored.
- ▶ To avoid storing activations in the modern frameworks, the backward pass should be manually redefined.

---

<https://arxiv.org/pdf/1707.04585.pdf>

## Hypothesis

The success of deep convolutional networks is based on progressively discarding uninformative variability about the input with respect to the problem at hand.

- ▶ It is difficult of recovering images from their hidden representations.
- ▶ Information bottleneck principle: an optimal representation must reduce the MI between an input and its representation to reduce uninformative variability + maximize the MI between the output and its representation to preserve each class from collapsing onto other classes.

## Hypothesis

The success of deep convolutional networks is based on progressively discarding uninformative variability about the input with respect to the problem at hand.

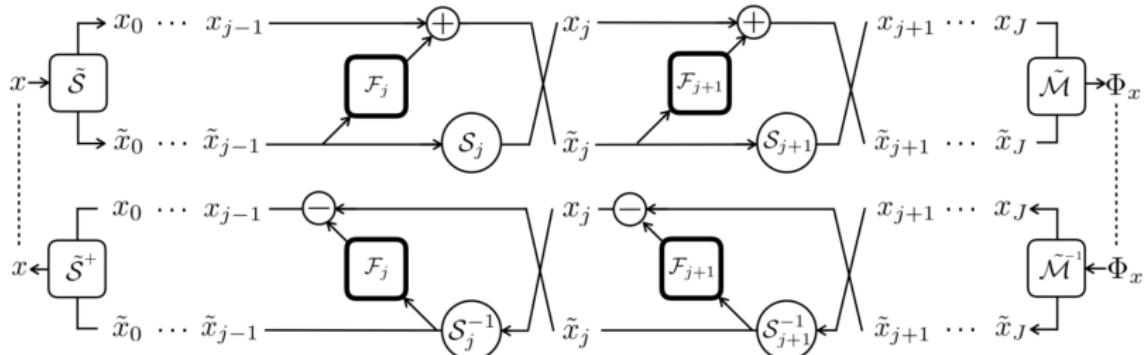
## Idea

Build a cascade of homeomorphic layers (i-RevNet), a network that can be fully inverted up to the final projection onto the classes, i.e. no information is discarded.

---

<https://arxiv.org/pdf/1802.07088.pdf>

# i-RevNet, 2018



Architecture	Injective	Bijective	Top-1 error	Parameters
ResNet	-	-	24.7	26M
RevNet	-	-	25.2	28M
<i>i</i> -RevNet (a)	yes	-	24.7	181M
<i>i</i> -RevNet (b)	yes	yes	26.7	29M

# References

- ▶ **NICE:** Non-linear Independent Components Estimation  
<https://arxiv.org/abs/1410.8516>  
**Summary:** Uses flows to model complex high-dimensional densities. Introduce the ways to compute determinant of Jacobian in a simple way. Triangular Jacobian, coupling layers, factorized distribution.
- ▶ **Variational Inference with Normalizing Flows**  
<https://arxiv.org/abs/1505.05770>  
**Summary:** Propose to use normalizing flows in variational inference. Discuss finite and infinitesimal flows. Uses invertible flows: planar, radial. Comparison with NICE.
- ▶ **RealNVP:** Density estimation using Real NVP  
<https://arxiv.org/pdf/1605.08803.pdf>  
**Summary:** Authors of NICE. The same idea and architecture, more practical. Lots of experiments and images. Coupling layers with checkerboard and channel-wise permutations.
- ▶ **RevNet:** The Reversible Residual Network: Backpropagation Without Storing Activations  
<https://arxiv.org/abs/1707.04585>  
**Summary:** RevNet allows not to store network activations. Each layer's activations can be computed from the next layer's activations. RevNets are composed of a series of reversible blocks. Could enable training larger and more powerful networks with limited computational resources.
- ▶ **i-RevNet:** Deep Invertible Networks  
<https://arxiv.org/abs/1802.07088>  
**Summary:** Invertible reversible networks. Remove noninvertible blocks (max-pooling, strides) from RevNets. Loss of information is not a necessary condition to learn representations that generalize well on hard problems, such as ImageNet.