Deep Generative Models

Roman Isachenko

Moscow Institute of Physics and Technology

2019

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 $\mathbf{z}$.

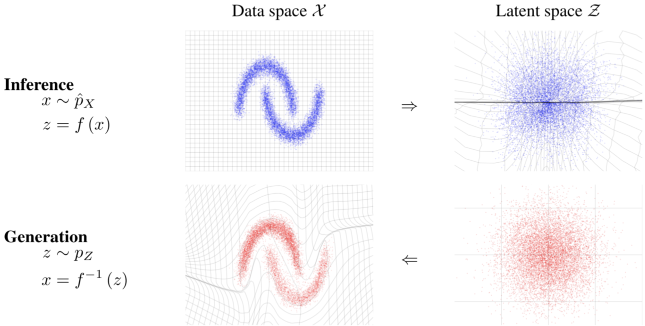
- Normalizing - convert data distribution to *noise*.
- Flow - sequence of such mapping is also a flow

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

$$p(\mathbf{x}) = p(f_K \circ \dots \circ f_1(\mathbf{x})) \left| \det \left(\frac{\partial f_K \circ \dots \circ f_1(\mathbf{x})}{\partial \mathbf{x}} \right) \right| =$$

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

Flows



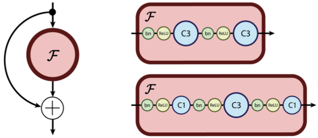
- Likelihood is given by $\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta})$ and change of variables.
- Sampling of $\mathbf{x}$ is performed by sampling from a base distribution $p(\mathbf{z})$ and applying $\mathbf{x} = f^{-1}(\mathbf{z}, \boldsymbol{\theta}) = g(\mathbf{z}, \boldsymbol{\theta})$.
-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

---

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

RevNets, 2017

- 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

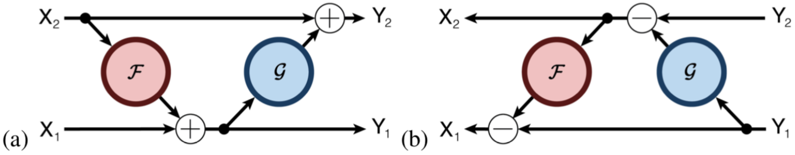
RevNets, 2017

NICE

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

RevNet

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



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

RevNets, 2017

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

- If the network contains non-reversible blocks (poolings, strides), activations for this 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

i-RevNet, 2018

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.

---

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

i-RevNet, 2018

Hypothesis

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

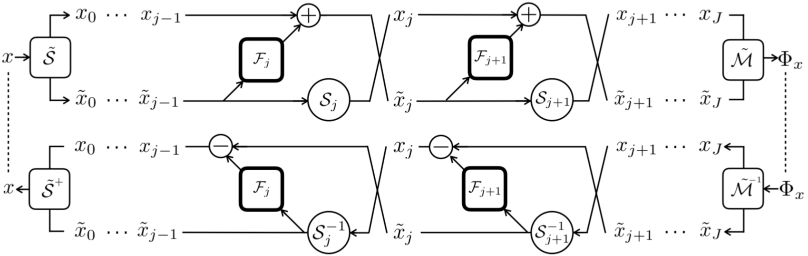
Idea

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



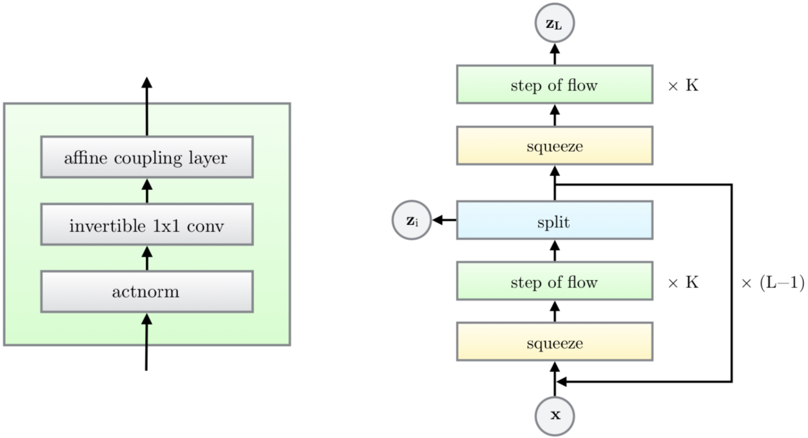
Architecture	Injective	Bijective	Top-1 error	Parameters
ResNet	-	-	24.7	26M
RevNet	-	-	25.2	28M
i-RevNet (a)	yes	-	24.7	181M
i-RevNet (b)	yes	yes	26.7	29M

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

Glow, 2018

https://arxiv.org/pdf/1807.03039.pdf

Glow, 2018



https://arxiv.org/pdf/1807.03039.pdf

Glow, 2018

Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i,j: \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j: \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \mathtt{sum}(\log \lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W}: [c \times c]$. See Section 3.2.	$\forall i,j: \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i,j: \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert\det(\mathbf{W})\rvert$ or $h \cdot w \cdot \mathtt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \mathtt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \mathtt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \mathtt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \mathtt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\mathtt{sum}(\log(\lvert\mathbf{s}\rvert))$

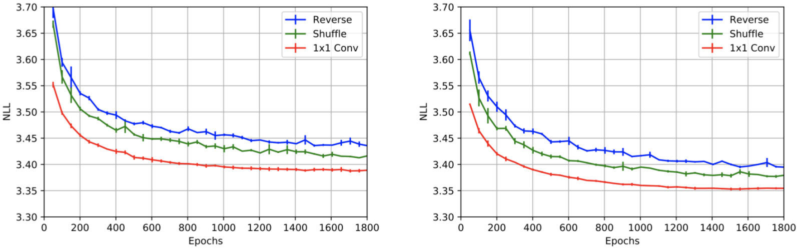
https://arxiv.org/pdf/1807.03039.pdf

Glow, 2018

Invertible 1x1 conv

Cost to compute $\det(\mathbf{W})$ is $O(c^3)$.
LU-decomposition reduces the cost to $O(c)$:

$$\mathbf{W} = \mathbf{PL}(\mathbf{U} + \mathrm{diag}(\mathbf{s})).$$

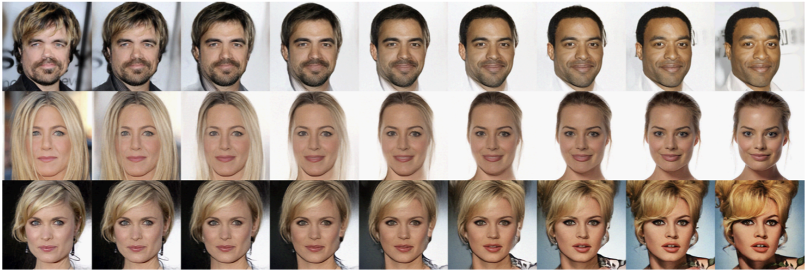


(a) Additive coupling. (b) Affine coupling.

https://arxiv.org/pdf/1807.03039.pdf

Glow, 2018

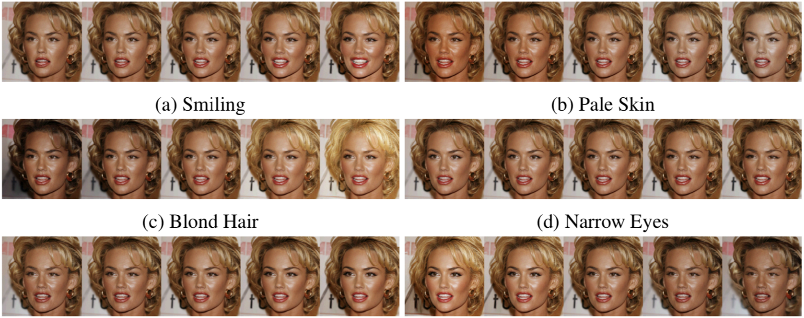
Face interpolation



https://arxiv.org/pdf/1807.03039.pdf

Glow, 2018

Face attributes manipulation

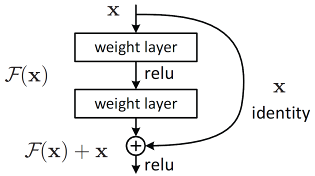


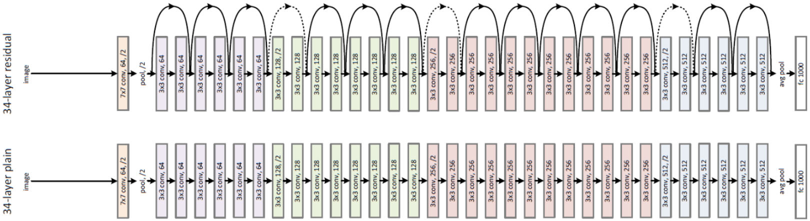
(a) Smiling
(b) Pale Skin
(c) Blond Hair
(d) Narrow Eyes
(e) Young
(f) Male

https://arxiv.org/pdf/1807.03039.pdf

Neural ODE, 2018

How did it become possible to train neural networks with hundreds of layers? Skip connections eliminates exploding/vanishing gradients.





---

https://arxiv.org/pdf/1806.07366.pdf

Neural ODE, 2018

Consider ODE

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), \boldsymbol{\theta}); \quad \mathbf{z}(t_0) = \mathbf{z}_0.$$

Euler update step

$$\mathbf{z}(t + \Delta t) = \mathbf{z}(t) + \Delta t f(\mathbf{z}(t), \boldsymbol{\theta}).$$

Residual block

$$\mathbf{z}_{t+1} = \mathbf{z}_t + f(\mathbf{z}_t, \boldsymbol{\theta}).$$

It is exactly Euler update step for solving ODE with $\Delta t = 1$!
Euler update step is unstable and trivial.

---

https://arxiv.org/pdf/1806.07366.pdf

Neural ODE, 2018

Residual block

$$\mathbf{z}_{t+1} = \mathbf{z}_t + f(\mathbf{z}_t, \boldsymbol{\theta}).$$

What happens as we add more layers and take smaller steps?
In the limit, we parameterize the continuous dynamics of hidden units using an ODE specified by a neural network:

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t, \boldsymbol{\theta}); \quad \mathbf{z}(t_0) = \mathbf{x}; \quad \mathbf{z}(t_1) = \mathbf{y}.$$

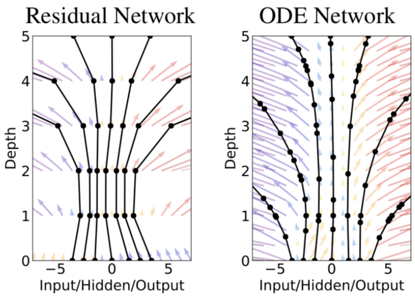
Loss function

$$\begin{aligned} L(\mathbf{y}) = L(\mathbf{z}(t_1)) &= L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \boldsymbol{\theta}) dt\right) \\ &= L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \boldsymbol{\theta})\right) \end{aligned}$$

https://arxiv.org/pdf/1806.07366.pdf

Neural ODE, 2018
Benefits

- memory efficient;
- adaptive computation;
- parameter efficient;
- scalable and invertible normalizing flows.



https://arxiv.org/pdf/1806.07366.pdf

Neural ODE, 2018

Loss function

$$L(\mathbf{y}) = L(\mathbf{z}(t_1)) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \boldsymbol{\theta})\right)$$

How to train such model? How to fit $\boldsymbol{\theta}$? How to compute efficiently $\frac{\partial L}{\partial \boldsymbol{\theta}}$? – Pontryagin theorem!

Adjoint function

$$\mathbf{a}(t) = \frac{\partial L(\mathbf{z}(t))}{\partial \mathbf{z}(t)}$$

Theorem

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \boldsymbol{\theta})}{\partial \mathbf{z}(t)}$$

---

https://arxiv.org/pdf/1806.07366.pdf

Neural ODE, 2018

Theorem

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \boldsymbol{\theta})}{\partial \mathbf{z}(t)}; \quad \mathbf{a}(t) = \frac{\partial L(\mathbf{z}(t))}{\partial \mathbf{z}(t)}$$

To obtain $\mathbf{a}(t)$ along the trajectory we could solve this ODE backward in time, starting from the initial value $\mathbf{a}(t_1) = \frac{\partial L(\mathbf{z}(t_1))}{\partial \mathbf{z}(t_1)}$.

Theorem

$$\frac{dL}{d\boldsymbol{\theta}} = -\int_{t_0}^{t_1} \mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} dt.$$

All these gradients could be computed at once.

---

https://arxiv.org/pdf/1806.07366.pdf

Continuous NF, 2018

Discrete NF

$$\mathbf{z}_{t+1} = f(\mathbf{z}_t, \boldsymbol{\theta}); \quad \log p(\mathbf{z}_{t+1}) = \log p(\mathbf{z}_t) - \log \left| \det \frac{\partial f(\mathbf{z}_t, \boldsymbol{\theta})}{\partial \mathbf{z}_t} \right|.$$

Function f should be bijective!

Theorem

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\mathrm{trace}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right).$$

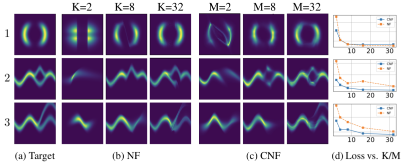
Function f is not necessary bijective! (uniformly Lipschitz continuous in $\mathbf{z}$ and continuous in t).

---

https://arxiv.org/pdf/1806.07366.pdf

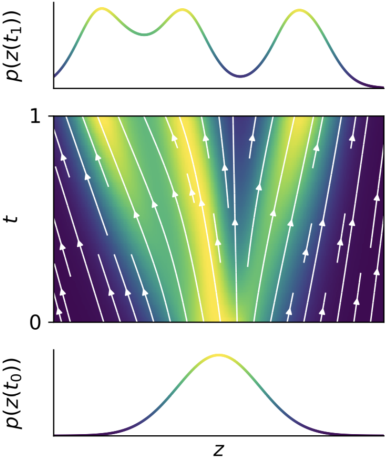
Continuous NF, 2018

$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \operatorname{trace}\left(\frac{\partial f}{\partial \mathbf{z}}\right) dt.$$



https://arxiv.org/pdf/1806.07366.pdf

FFJORD, 2018



---

https://arxiv.org/pdf/1810.01367.pdf

FFJORD, 2018

Hutchinson's trace estimator

$$\text{trace}(A) = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\boldsymbol{\epsilon}^T A \boldsymbol{\epsilon}\right]; \quad \mathbb{E}[\boldsymbol{\epsilon}] = 0; \quad \text{Cov}(\boldsymbol{\epsilon}) = I.$$

$$\begin{aligned}
\log p(\mathbf{z}(t_1)) &= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \text{trace}\left(\frac{\partial f}{\partial \mathbf{z}}\right) dt \\
&= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}} \boldsymbol{\epsilon}\right] dt \\
&= \log p(\mathbf{z}(t_0)) - \mathbb{E}_{p(\boldsymbol{\epsilon})} \int_{t_0}^{t_1} \left[\boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}} \boldsymbol{\epsilon}\right] dt.
\end{aligned}$$

This reduces the cost from quadratic to linear.

---

https://arxiv.org/pdf/1810.01367.pdf

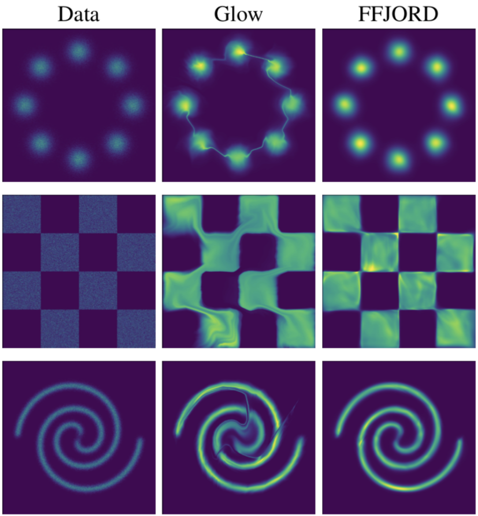
FFJORD, 2018

	Method	Train on data	One-pass Sampling	Exact log-likelihood	Free-form Jacobian
	Variational Autoencoders	✓	✓	✗	✓
	Generative Adversarial Nets	✓	✓	✗	✓
	Likelihood-based Autoregressive	✓	✗	✓	✗
Change of Variables	Normalizing Flows	✗	✓	✓	✗
	Reverse-NF, MAF, TAN	✓	✗	✓	✗
	NICE, Real NVP, Glow, Planar CNF	✓	✓	✓	✗
	FFJORD	✓	✓	✓	✓

---

https://arxiv.org/pdf/1810.01367.pdf

FFJORD, 2018



https://arxiv.org/pdf/1810.01367.pdf

References

- **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 complicated problems, such as ImageNet.
- **Glow:** Better Reversible Generative Models
  https://arxiv.org/abs/1807.03039
  Summary: Extension of RealNVP. Suggests 1x1 reversible convolutions instead of reversing channel ordering. 1x1 conv is square matrix which could be easily be inversed. Compares 1x1 conv with reversing and fixed shuffling.
- Neural Ordinary Differential Equations
  https://arxiv.org/abs/1806.07366
  Summary: New interpretation of resnets as special case of ode. Discrete sequence of layers are replaced with continuous dynamic. ODESolver is used for backpropagation. Pontryagin theorem gives the analog of the chain rule. Continuous version of normalizing flow is constructed.
- **FFJORD:** Free-form Continuous Dynamics for Scalable Reversible Generative Models
  https://arxiv.org/abs/1810.01367
  Summary: Continuous version of NF is investigated. Jacobian computation cost is reduced to O(D) by using Hutchinson's trace estimator.