Introduction to Flow-based Generative Models

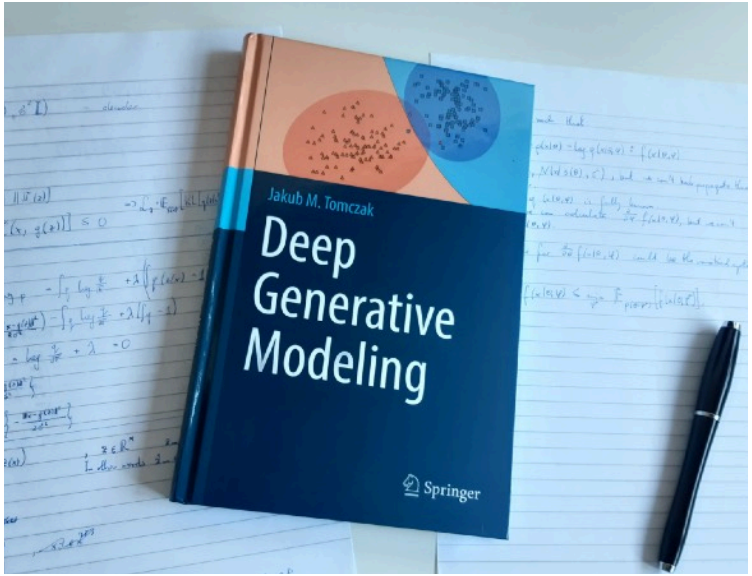
Jakub M. Tomczak

VRIJE UNIVERSITEIT AMSTERDAM

MORE ABOUT DEEP GENERATIVE MODELING

Blog: https://jmtomczak.github.io/blog.html

Book: https://link.springer.com/book/10.1007/978-3-030-93158-2



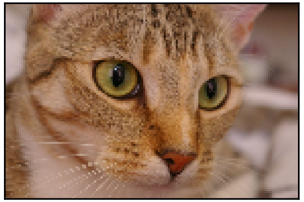
WHAT HAPPENS IF WE LEARN ONLY DECISION MAKING

The bulk of AI is focused **only** on the decision making part!

WHAT HAPPENS IF WE LEARN ONLY DECISION MAKING

The bulk of AI is focused **only** on the decision making part!

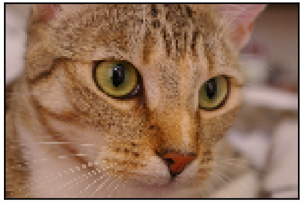
Example: Let’s say we have a model that is well trained.



$p(y = cat|\mathbf{x}) = 0.90$
$p(y = dog|\mathbf{x}) = 0.05$
$p(y = horse|\mathbf{x}) = 0.05$

The bulk of AI is focused **only** on the decision making part!

Example: Let’s say we have a model that is well trained.

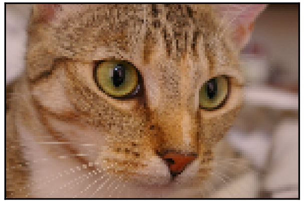


$p(y = cat|\mathbf{x}) = 0.90$
$p(y = dog|\mathbf{x}) = 0.05$
$p(y = horse|\mathbf{x}) = 0.05$

+

noise

=

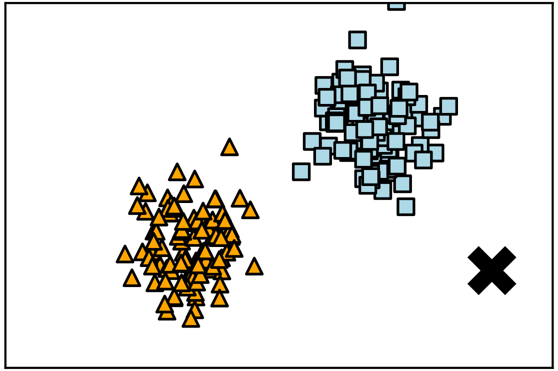


$p(y = cat|\mathbf{x}) = 0.05$
$p(y = dog|\mathbf{x}) = 0.05$
$p(y = horse|\mathbf{x}) = 0.90$

But after adding a little noise it could fail completely…

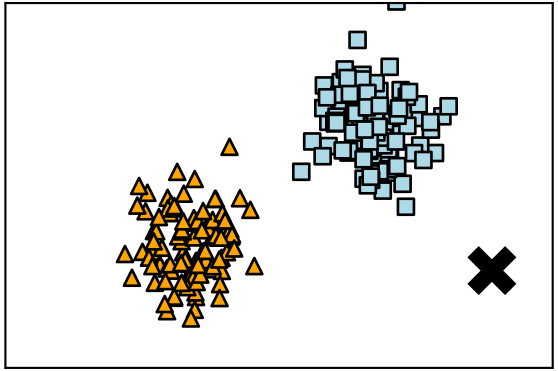
DEEP GENERATIVE MODELING: WHY DO WE NEED IT?

DEEP GENERATIVE MODELING: WHY DO WE NEED IT?

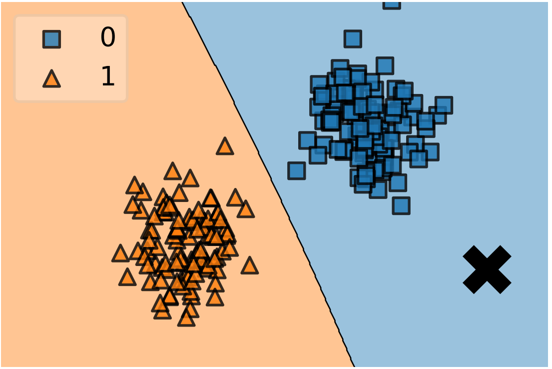


Data

DEEP GENERATIVE MODELING: WHY DO WE NEED IT?



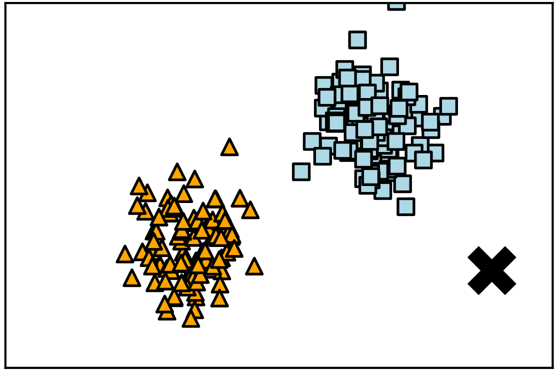
Data



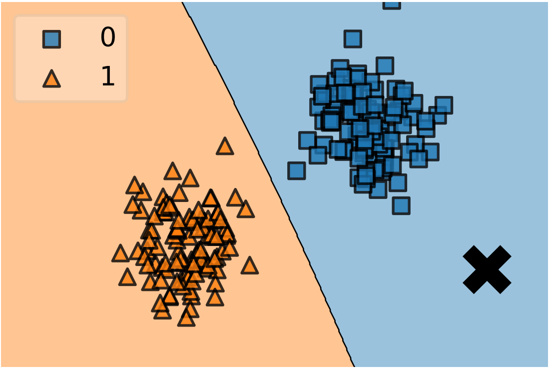
$p(y|\mathbf{x})$

$p(blue|\mathbf{x})$ is high
= certain decision!

DEEP GENERATIVE MODELING: WHY DO WE NEED IT?

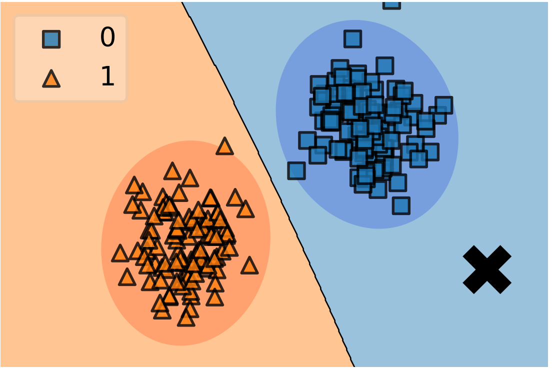


Data



$p(y|\mathbf{x})$

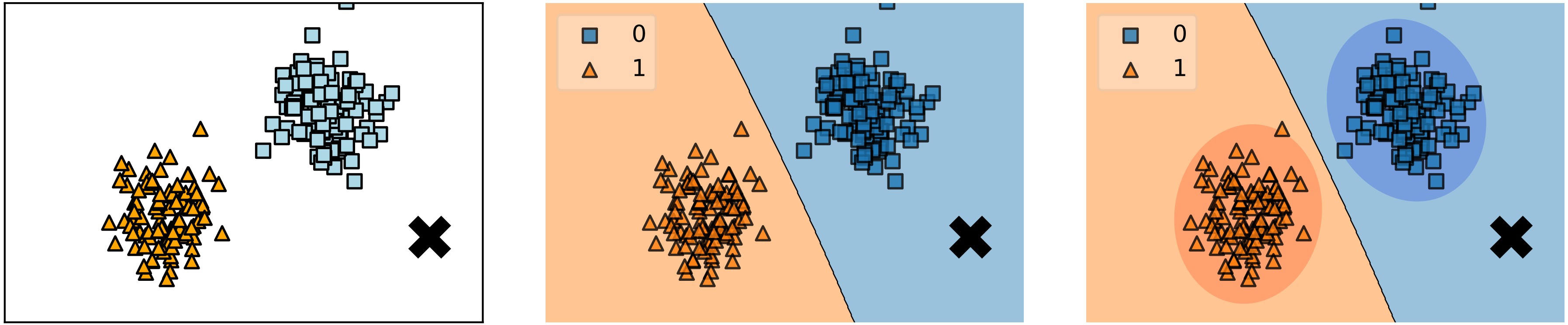
$p(blue|\mathbf{x})$ is high
= certain decision!



$p(\mathbf{x}, y) = p(y|\mathbf{x})\ p(\mathbf{x})$

$p(blue|\mathbf{x})$ is high
and $p(\mathbf{x})$ is low
= uncertain decision!

DEEP GENERATIVE MODELING: WHY DO WE NEED IT?



Data

$p(y|\mathbf{x})$

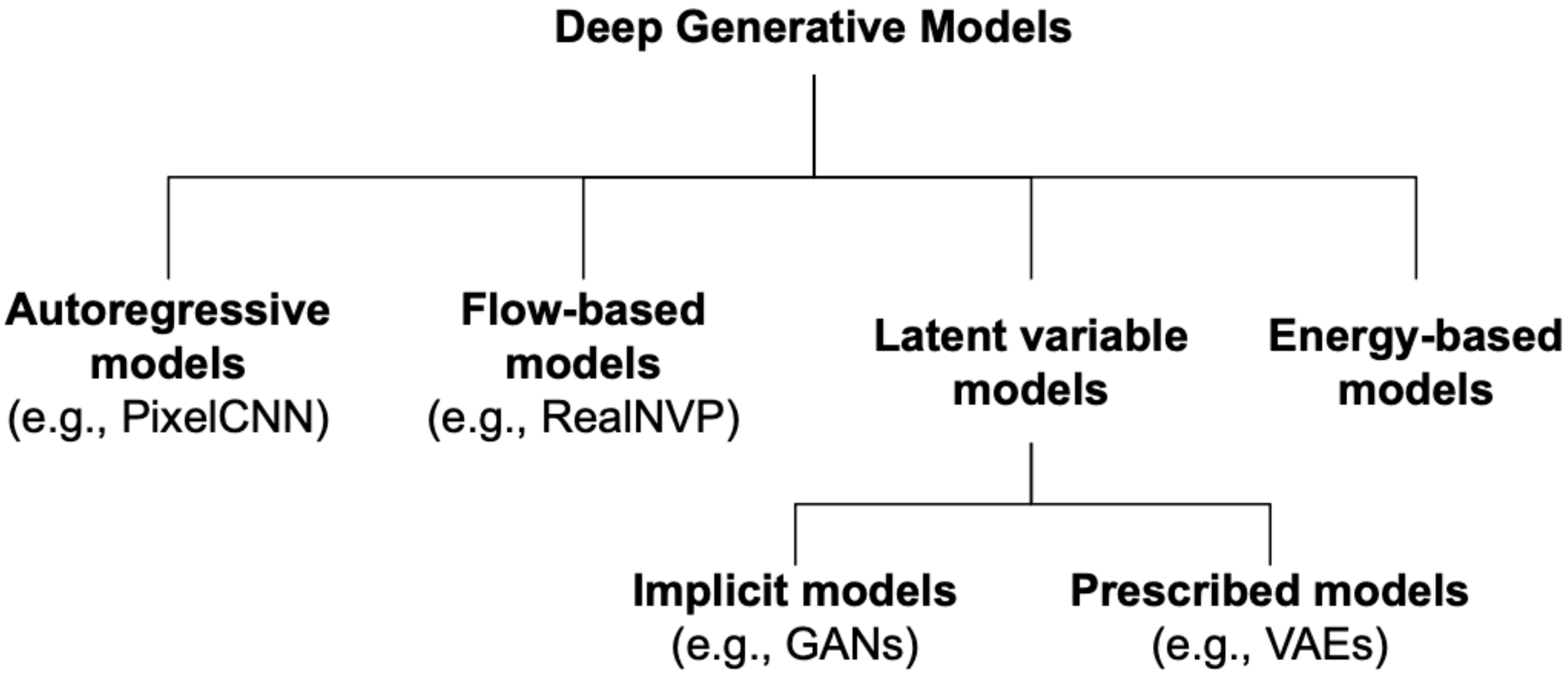
$p(\mathbf{x}, y) = p(y|\mathbf{x})\ p(\mathbf{x})$

$p(blue|\mathbf{x})$ is high
= certain decision!

$p(blue|\mathbf{x})$ is high
and $p(\mathbf{x})$ is low
= uncertain decision!

Thus, learning the conditional is only a part of the story!
How can we learn p(x)?

DEEP GENERATIVE MODELING: HOW WE CAN FORMULATE IT?



DEEP GENERATIVE MODELING: HOW WE CAN FORMULATE IT?

Generative models	Training	Likelihood	Sampling	Compression	Representation
Autoregressive models	stable	exact	slow	lossless	no
Flow-based models	stable	exact	fast/slow	lossless	yes
Implicit models	unstable	no	fast	no	no
Prescribed models	stable	approximate	fast	lossy	yes
Energy-based models	stable	unnormalized	slow	rather not	yes

DEEP GENERATIVE MODELING: WHERE CAN WE USE IT?

" i want to talk to you . "
"i want to be with you . "
"i do n't want to be with you . "
i do n't want to be with you .
she did n't want to be with him .

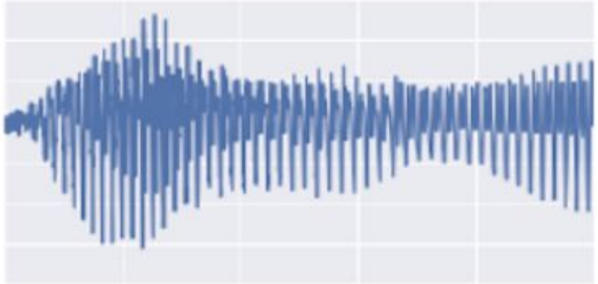
he was silent for a long moment .
he was silent for a moment .
it was quiet for a moment .
it was dark and cold .
there was a pause .
it was my turn .

Text

Text synthesis (e.g., chatbots)

DEEP GENERATIVE MODELING: WHERE CAN WE USE IT?

Images

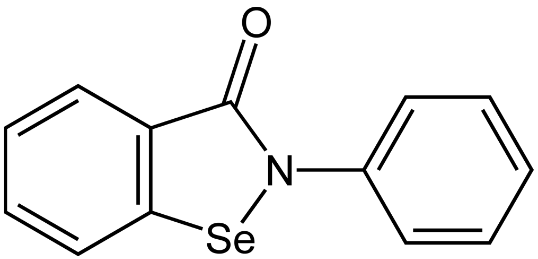


Audio

Speech synthesis (e.g., chatbots)

Deep fakes

DEEP GENERATIVE MODELING: WHERE CAN WE USE IT?



Graphs

Drug discovery

3D structure discovery

Molecular dynamics

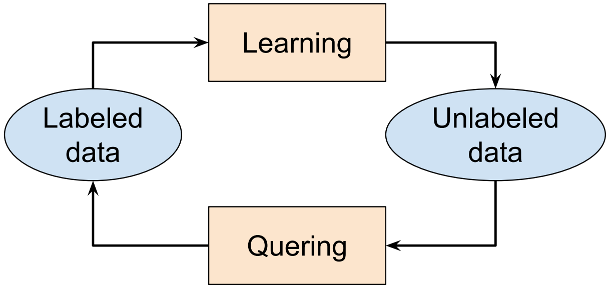
DEEP GENERATIVE MODELING: WHERE CAN WE USE IT?

“ i want to talk to you . ”
“i want to be with you . ”
“i do n’t want to be with you . ”
i do n’t want to be with you .
she did n’t want to be with him .

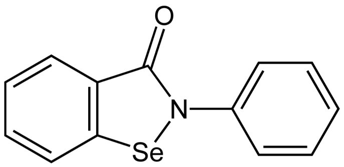
he was silent for a long moment .
he was silent for a moment .
it was quiet for a moment .
it was dark and cold .
there was a pause .
it was my turn .

Text

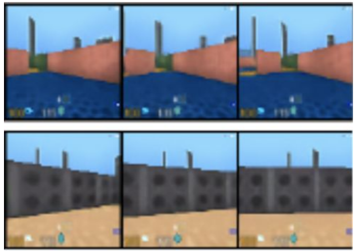
Images



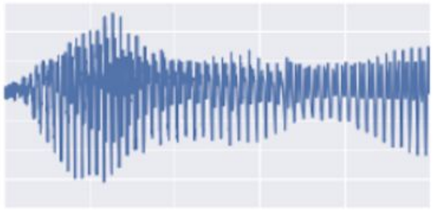
Active learning



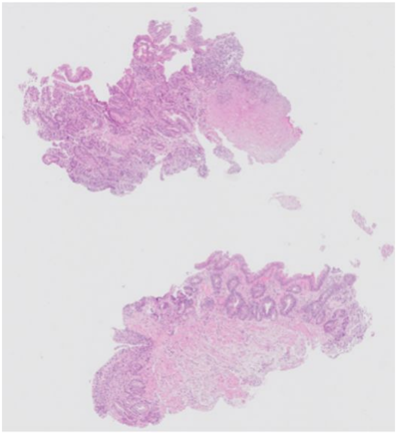
Graphs



Reinforcement learning



Audio



Medical imaging

BASIC RULES FOR DEEP GENERATIVE MODELING

Two rules of probability theory:

- **Sum rule**:

$$p(x) = \sum_{y} p(x, y)$$

- **Product rule**:

$$p(x, y) = p(x | y) p(y)$$

or

$$p(x, y) = p(y | x) p(x)$$

The objective (typically): **the log-likelihood function**

Given ***iid* data**: $\mathcal{D} = \{x_1, x_2, \ldots, x_N\}$

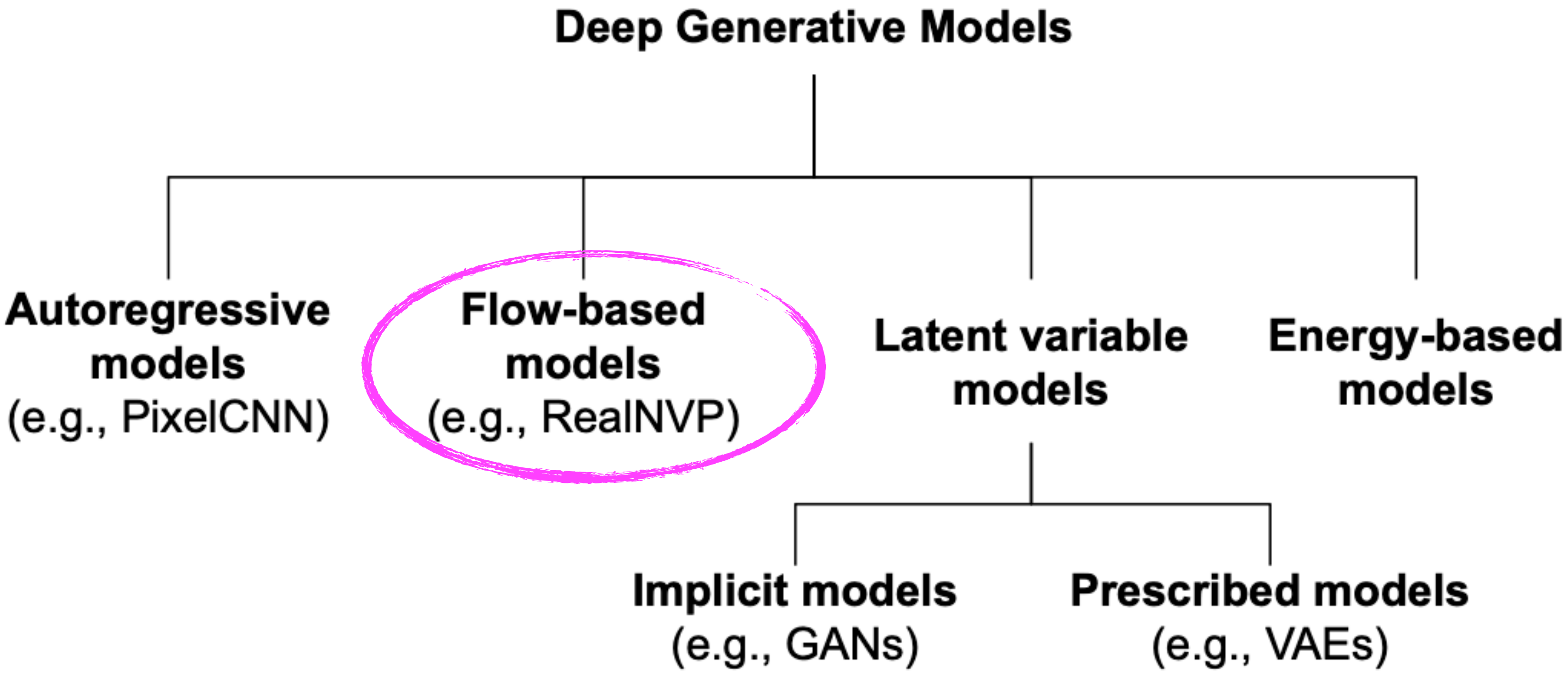
A **model** $p(x \mid \theta)$.

The log-likelihood function is:

$$\ln p(\mathcal{D} \mid \theta) = \ln \prod_{n=1}^{N} p(x_n \mid \theta)$$

$$= \sum_{n=1}^{N} \ln p(x_n \mid \theta)$$

DEEP GENERATIVE MODELING: HOW WE CAN FORMULATE IT?



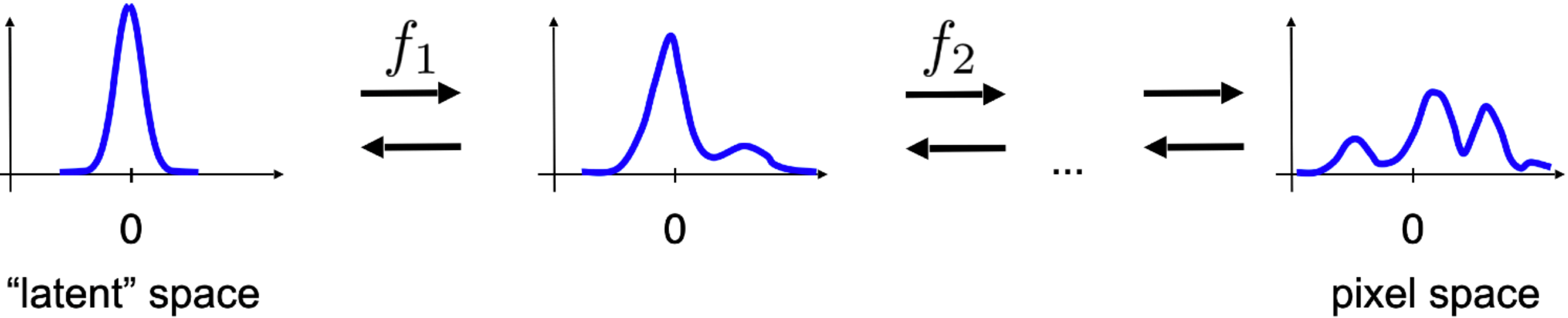
FLOWS (FLOW-BASED MODELS)

We change a random variable **x** to another random variable **z** using **invertible** transformations, $\mathbf{x}, \mathbf{z} \in \mathbb{R}^D$:

$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right) \prod_{i=1}^{K} \left| \mathbf{J}_{f_i}(z_{i-1}) \right|^{-1}$$

We change a random variable **x** to another random variable **z** using **invertible** transformations, $\mathbf{x}, \mathbf{z} \in \mathbb{R}^D$:

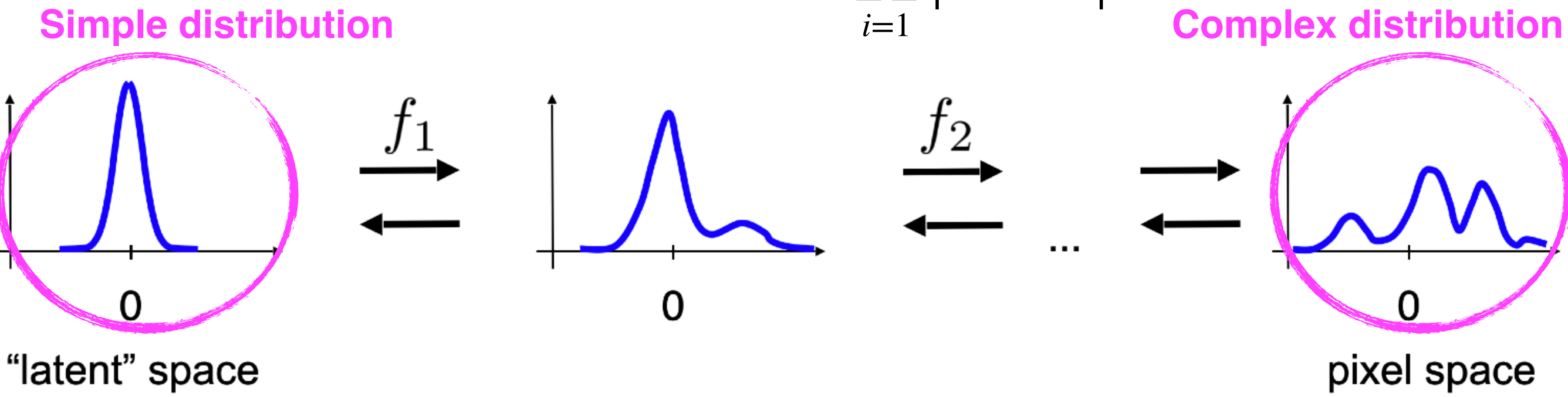
$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right) \prod_{i=1}^{K} \left| \mathbf{J}_{f_i}(z_{i-1}) \right|^{-1}$$



FLOWS (FLOW-BASED MODELS)

We change a random variable **x** to another random variable **z** using **invertible** transformations, $\mathbf{x}, \mathbf{z} \in \mathbb{R}^D$:

$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right) \prod_{i=1}^{K} \left| \mathbf{J}_{f_i}(z_{i-1}) \right|^{-1}$$

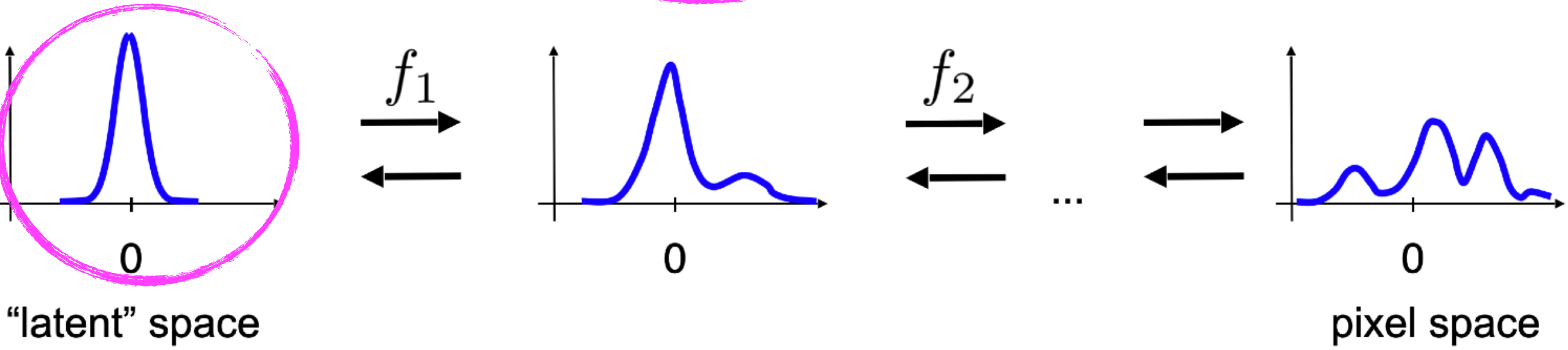


FLOWS (FLOW-BASED MODELS)

We change a random variable **x** to another random variable **z** using **invertible** transformations, $\mathbf{x}, \mathbf{z} \in \mathbb{R}^D$:

Known, e.g., Gaussian

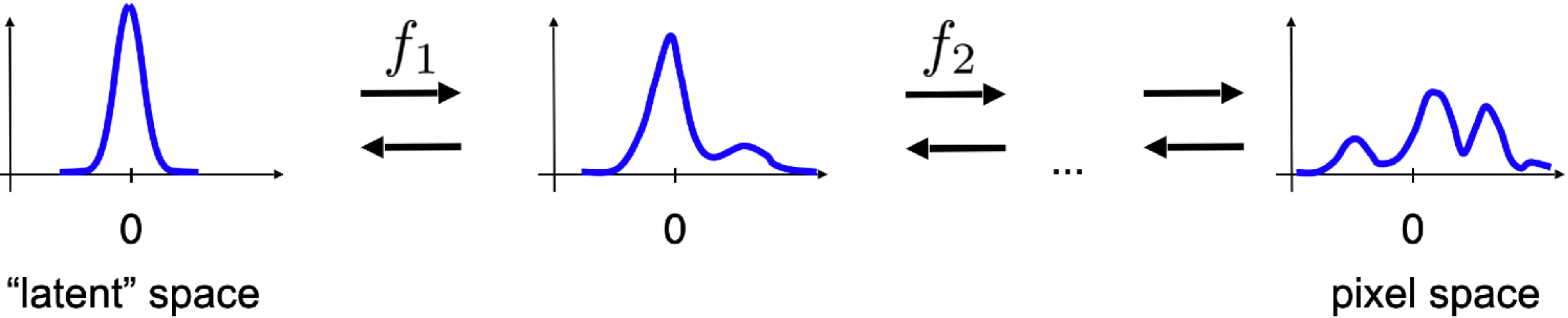
$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right) \prod_{i=1}^{K} \left| \mathbf{J}_{f_i}(z_{i-1}) \right|^{-1}$$



We change a random variable **x** to another random variable **z** using **invertible** transformations, $\mathbf{x}, \mathbf{z} \in \mathbb{R}^D$:

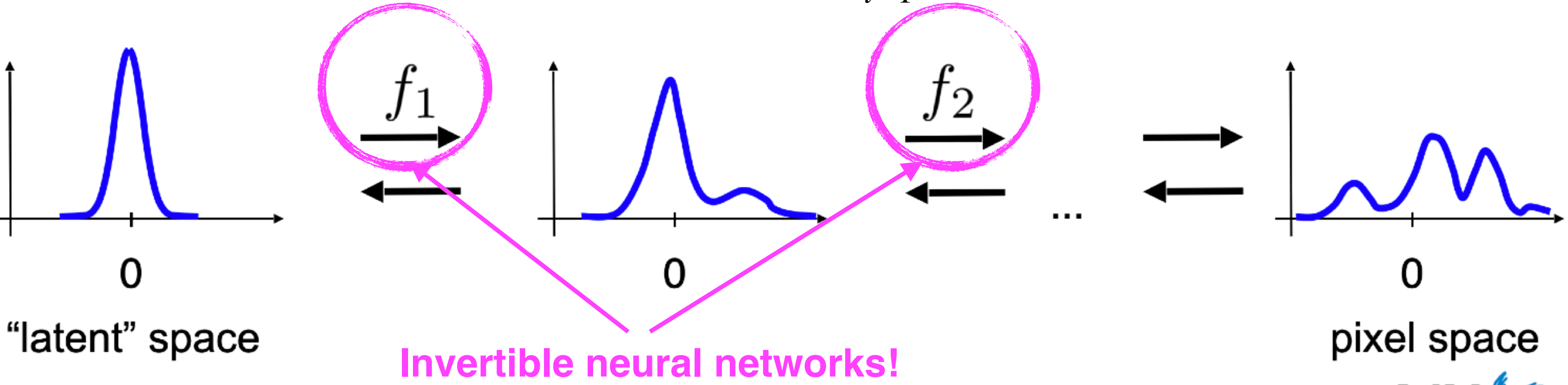
Jacobian must be tractable

$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right) \prod_{i=1}^{K} \left| \mathbf{J}_{f_i}(z_{i-1}) \right|^{-1}$$



We change a random variable **x** to another random variable **z** using **invertible** transformations, $\mathbf{x}, \mathbf{z} \in \mathbb{R}^D$:

$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right) \prod_{i=1}^{K} \left| \mathbf{J}_{f_i}(z_{i-1}) \right|^{-1}$$

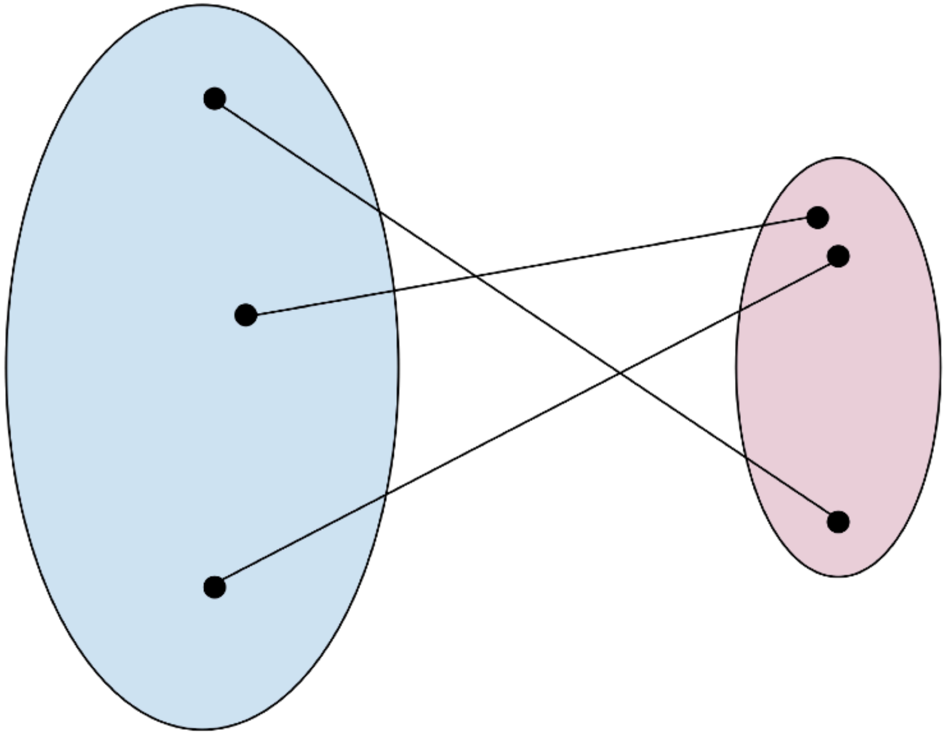


We change a random variable **x** to another random variable **z** using **invertible** transformations, $\mathbf{x}, \mathbf{z} \in \mathbb{R}^D$:

$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right) \prod_{i=1}^{K} \left| \mathbf{J}_{f_i}(z_{i-1}) \right|^{-1}$$

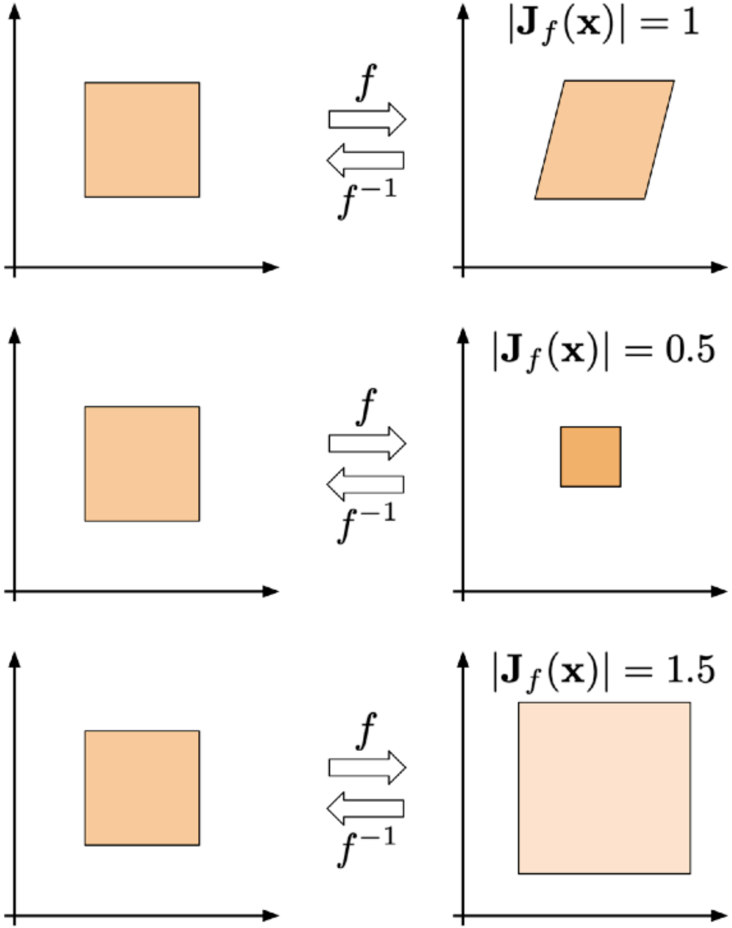
Training objective:

$$\ln p(\mathbf{x}) = \ln \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right) - \sum_{i=1}^{K} \ln \left| \mathbf{J}_{f_i}(z_{i-1}) \right|$$



Remember! Every neural network must be a bijection!

FLOWS (FLOW-BASED MODELS): INVERTIBLE LAYERS



How to formulate invertible layers then?

How to formulate invertible layers then?

1) **Coupling layers**

$$\begin{aligned} \mathbf{y}_a &= \mathbf{x}_a \\ \mathbf{y}_b &= \exp\left(s\left(\mathbf{x}_a\right)\right) \odot \mathbf{x}_b + t\left(\mathbf{x}_a\right) \end{aligned} \quad \Longleftrightarrow \quad \begin{aligned} \mathbf{x}_b &= \left(\mathbf{y}_b - t(\mathbf{y}_a)\right) \odot \exp\left(-s(\mathbf{y}_a)\right) \\ \mathbf{x}_a &= \mathbf{y}_a \end{aligned}$$

How to formulate invertible layers then?

1) **Coupling layers**

$$\mathbf{y}_a = \mathbf{x}_a$$
$$\mathbf{y}_b = \exp\left(s\left(\mathbf{x}_a\right)\right) \odot \mathbf{x}_b + t\left(\mathbf{x}_a\right)$$

$$\longleftrightarrow$$

$$\mathbf{x}_b = \left(\mathbf{y}_b - t(\mathbf{y}_a)\right) \odot \exp\left(-s(\mathbf{y}_a)\right)$$
$$\mathbf{x}_a = \mathbf{y}_a$$

Jacobian is tractable!

$$\det(\mathbf{J}) = \prod_{j=1}^{D-d} \exp\left(s\left(\mathbf{x}_a\right)\right)_j = \exp\left(\sum_{j=1}^{D-d} s\left(\mathbf{x}_a\right)_j\right)$$

How to formulate invertible layers then?

1) **Coupling layers**

$$\begin{aligned} \mathbf{y}_a &= \mathbf{x}_a \\ \mathbf{y}_b &= \exp\left(s\left(\mathbf{x}_a\right)\right) \odot \mathbf{x}_b + t\left(\mathbf{x}_a\right) \end{aligned} \quad \longleftrightarrow \quad \begin{aligned} \mathbf{x}_b &= \left(\mathbf{y}_b - t(\mathbf{y}_a)\right) \odot \exp\left(-s(\mathbf{y}_a)\right) \\ \mathbf{x}_a &= \mathbf{y}_a \end{aligned}$$

Jacobian is tractable!

$$\det(\mathbf{J}) = \prod_{j=1}^{D-d} \exp\left(s\left(\mathbf{x}_a\right)\right)_j = \exp\left(\sum_{j=1}^{D-d} s\left(\mathbf{x}_a\right)_j\right)$$

2) **Permutation layers**

How to formulate invertible layers then?

1) **Coupling layers**

$$\mathbf{y}_a = \mathbf{x}_a$$
$$\mathbf{y}_b = \exp\left(s\left(\mathbf{x}_a\right)\right) \odot \mathbf{x}_b + t\left(\mathbf{x}_a\right)$$

$$\longleftrightarrow$$

$$\mathbf{x}_b = \left(\mathbf{y}_b - t(\mathbf{y}_a)\right) \odot \exp\left(-s(\mathbf{y}_a)\right)$$
$$\mathbf{x}_a = \mathbf{y}_a$$

Jacobian is tractable!

$$\det(\mathbf{J}) = \prod_{j=1}^{D-d} \exp\left(s\left(\mathbf{x}_a\right)\right)_j = \exp\left(\sum_{j=1}^{D-d} s\left(\mathbf{x}_a\right)_j\right)$$

2) **Permutation layers**

$$\det(\mathbf{J}) = 1$$

FLOWS (FLOW-BASED MODELS)

 Kingma, D.P., and Prafulla D. "Glow: Generative flow with invertible 1x1 convolutions." *NeurIPS, 2018*

FLOW-BASED MODELS (SELECTED)

- RealNVP:

Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using real nvp. arXiv preprint arXiv:1605.08803.

- GLOW:

Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. NeurIPS.

- Sylvester Flows:

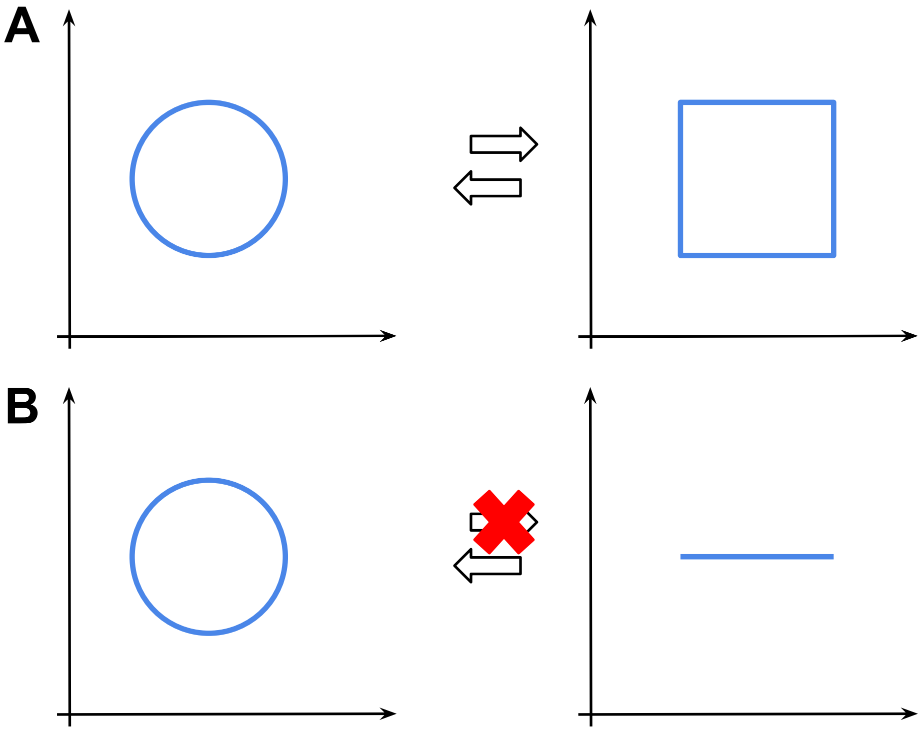
Hoogeboom, E., Garcia Satorras, V., Tomczak, J., & Welling, M. (2020). The convolution exponential and generalized Sylvester flows. NeurIPS

- Residual Flows & invertible DenseNet Flows

Chen, R. T., Behrmann, J., Duvenaud, D. K., & Jacobsen, J. H. (2019). Residual flows for invertible generative modeling. NeurIPS

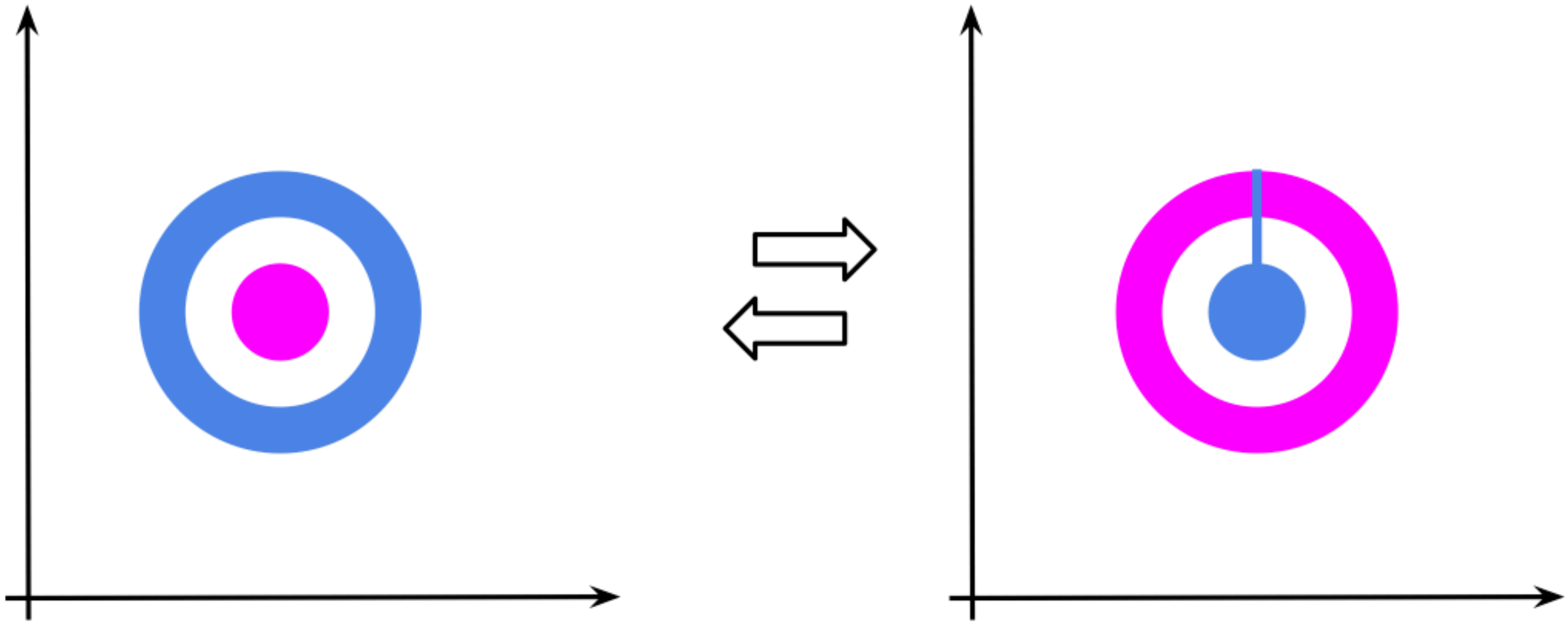
Perugachi-Diaz, Y., Tomczak, J., & Bhulai, S. (2021). Invertible densenets with concatenated lipswish. NeurIPS

POTENTIAL ISSUES WITH FLOWS



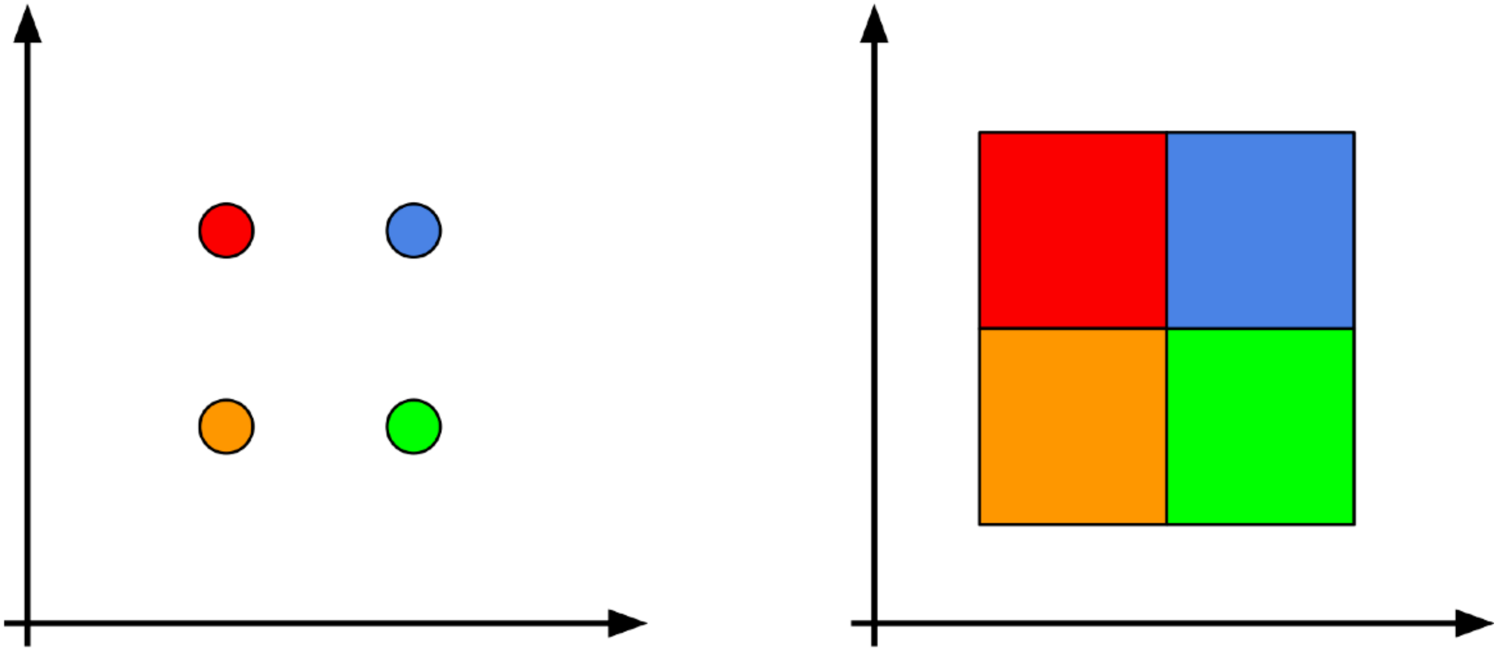
If we cut the circle at some point, we cannot invert it back. Why?
We don't know where the "start" and the "end" should be joined.

POTENTIAL ISSUES WITH FLOWS



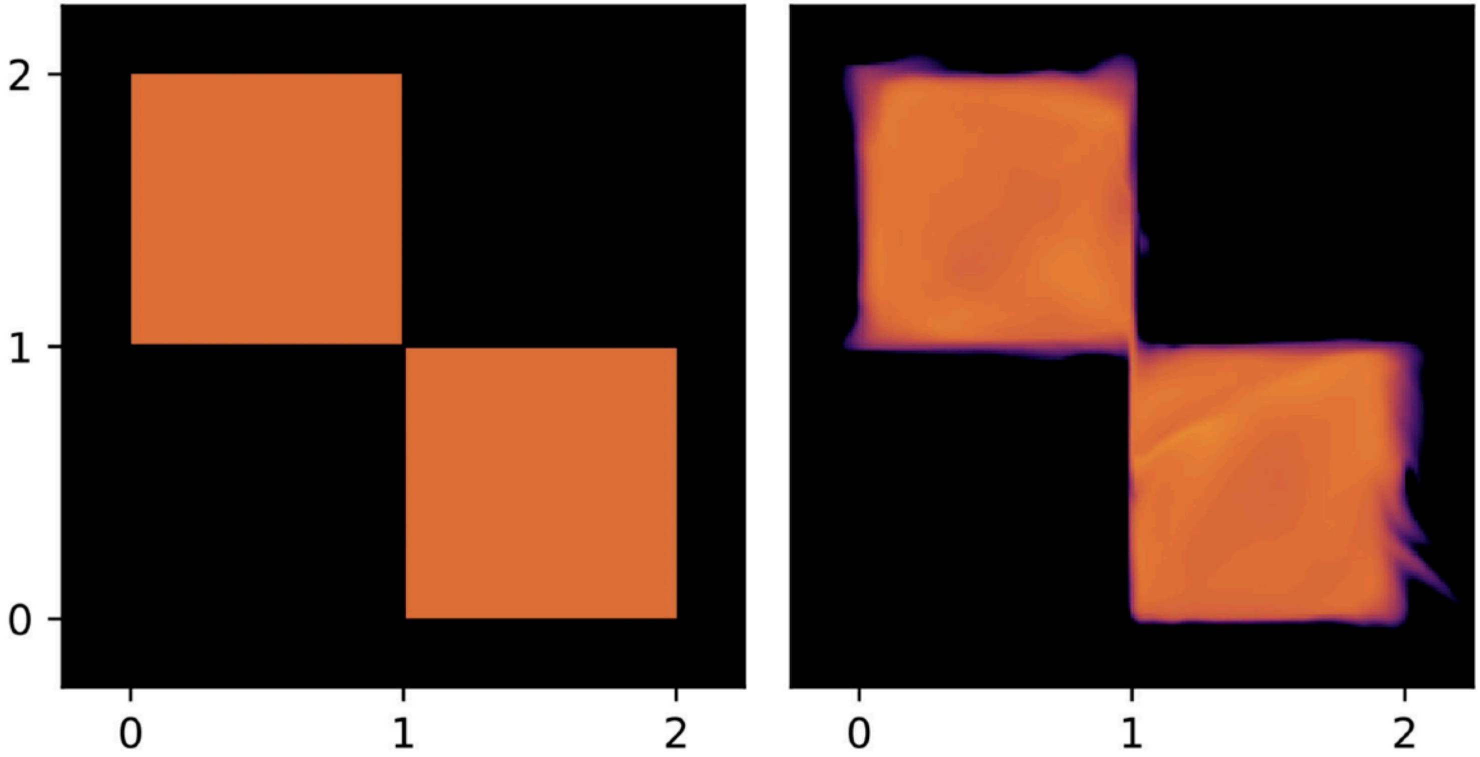
Replacing the positions of the two circles is impossible, unless we leave a “trace”.

POTENTIAL ISSUES WITH FLOWS (*DEQUANTIZATION*)



Many data (e.g., images) take discrete values. To use flows, we need to apply *dequantization*.

POTENTIAL ISSUES WITH FLOWS (*DEQUANTIZATION*)



The problem is that after training a flow-based model, we may still assign positive probability to regions outside the domain.

DISCRETE-VALUED INVERTIBLE TRANSFORMATIONS

We change a random variable **x** to another random variable **z** using **invertible** transformations, $\mathbf{x}, \mathbf{z} \in \mathbb{Z}^D$:

$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right)$$

We change a random variable **x** to another random variable **z** using **invertible** transformations, $\mathbf{x}, \mathbf{z} \in \mathbb{Z}^D$:

$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right)$$

We don't have the Jacobian here! **Why**?

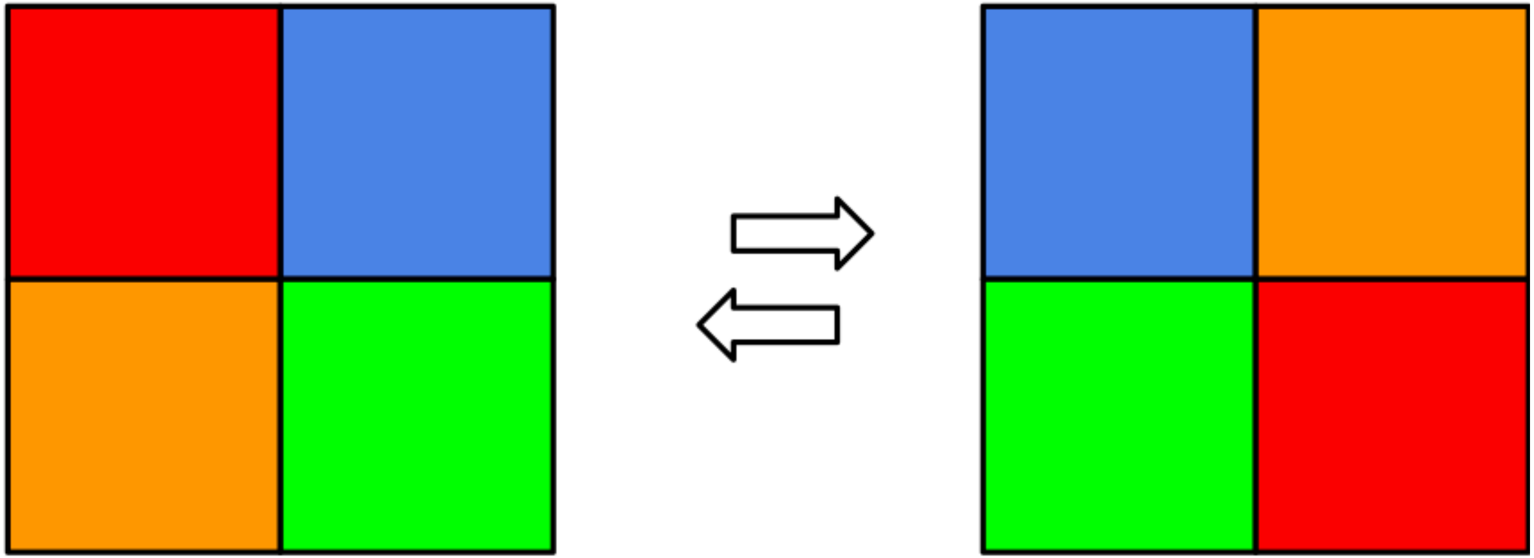
We change a random variable **x** to another random variable **z** using **invertible** transformations, $\mathbf{x}, \mathbf{z} \in \mathbb{Z}^D$:

$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right)$$

We don't have the Jacobian here! Why?

Because it's discrete, so we can only **"shuffle" probabilities**.

DISCRETE-VALUED INVERTIBLE TRANSFORMATIONS



Because it’s discrete, so we can only **“shuffle” probabilities**.

DISCRETE-VALUED INVERTIBLE TRANSFORMATIONS

We change a random variable **x** to another random variable **z** using **invertible** transformations, $\mathbf{x}, \mathbf{z} \in \mathbb{Z}^D$:

$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right)$$

We don't have the Jacobian here! Why?

Because it's discrete, so we can only "shuffle" probabilities.

Is it still useful then?

(v.d. Berg et al., 2020) showed that if we consider $\mathbf{x}, \mathbf{z} \in \mathscr{X} \subset \mathbb{Z}^D$ and $|\mathscr{X}| = M$, then we can only permute probability mass tensors.

$$p_{\boldsymbol{x}}(x_1, x_2) : \quad \begin{array}{c|c} x_1 \backslash x_2 & \begin{array}{cc} 0 & \quad 1 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{pmatrix} 0.1 & 0.3 \\ 0.2 & 0.4 \end{pmatrix} \end{array}$$

DISCRETE-VALUED INVERTIBLE TRANSFORMATIONS

(v.d. Berg et al., 2020) showed that if we consider $\mathbf{x}, \mathbf{z} \in \mathscr{X} \subset \mathbb{Z}^D$ and $|\mathscr{X}| = M$, then we can only permute probability mass tensors.

$$p_{\boldsymbol{x}}(x_1, x_2): \quad \begin{array}{c|cc} x_1 \backslash x_2 & 0 & 1 \\ \begin{matrix} 0 \\ 1 \end{matrix} & \multicolumn{2}{c}{\begin{pmatrix} 0.1 & 0.3 \\ 0.2 & 0.4 \end{pmatrix}} \end{array}$$

$$\begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix}$$

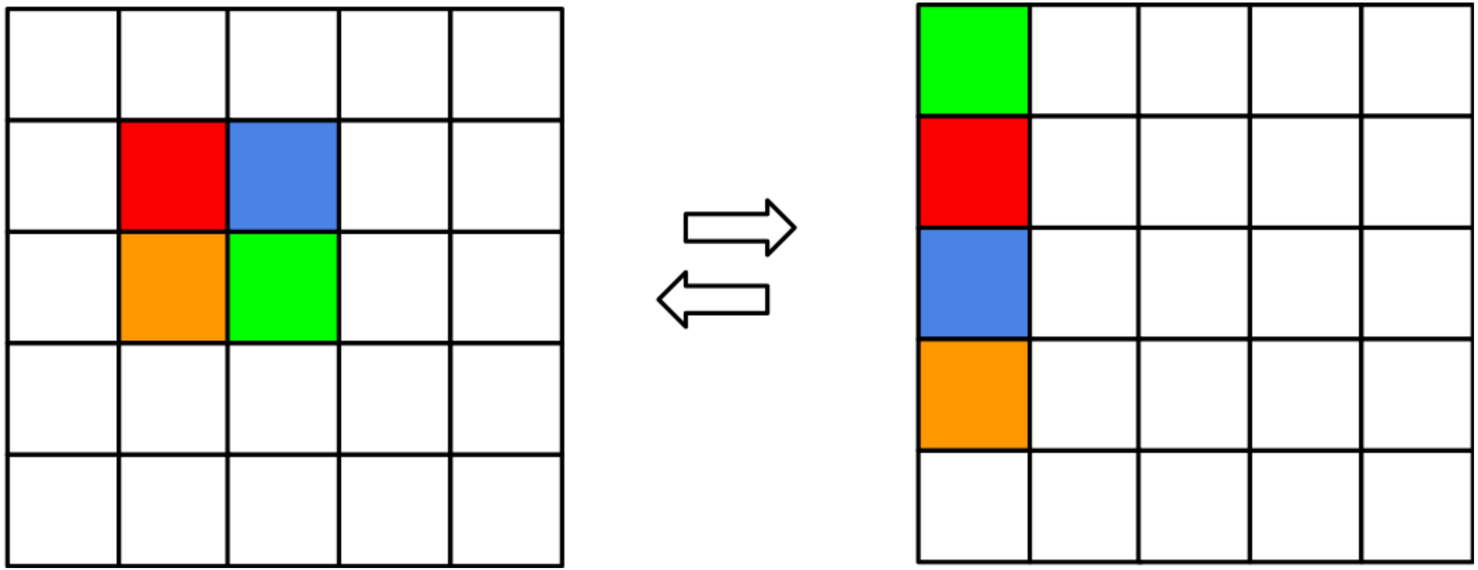
DISCRETE-VALUED INVERTIBLE TRANSFORMATIONS

(v.d. Berg et al., 2020) showed that if we consider $\mathbf{x}, \mathbf{z} \in \mathscr{X} \subset \mathbb{Z}^D$ and $|\mathscr{X}| = M$, then we can only permute probability mass tensors.

$$p_{\boldsymbol{x}}(x_1, x_2): \quad \begin{array}{ccc} x_1 \backslash x_2 & 0 & 1 \\ 0 & 0.1 & 0.3 \\ 1 & 0.2 & 0.4 \end{array}$$

If we consider an extended $\mathscr{X}$, we can learn a factorized distributions!

$$p_{\boldsymbol{x}}(x_1, x_2): \begin{pmatrix} 0.1 & 0.3 & 0 & 0 \\ 0.2 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow p_{\boldsymbol{y}}(y_1, y_2): \begin{pmatrix} 0.1 & 0 & 0 & 0 \\ 0.2 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0 \\ 0.4 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$



If we consider an extended $\mathscr{X}$, we can learn a factorized distributions!

DISCRETE-VALUED INVERTIBLE TRANSFORMATIONS

Ok, what does it mean?

It means that flow-based models are rather useless for finite domains.

BUT, they could learn any distribution for extended or infinite domains!

For details, see Lemma 1 in (v.d. Berg et al., 2020).

DISCRETE-VALUED INVERTIBLE TRANSFORMATIONS

How to formulate invertible transformations for integer-valued data?

1. **Coupling layers**

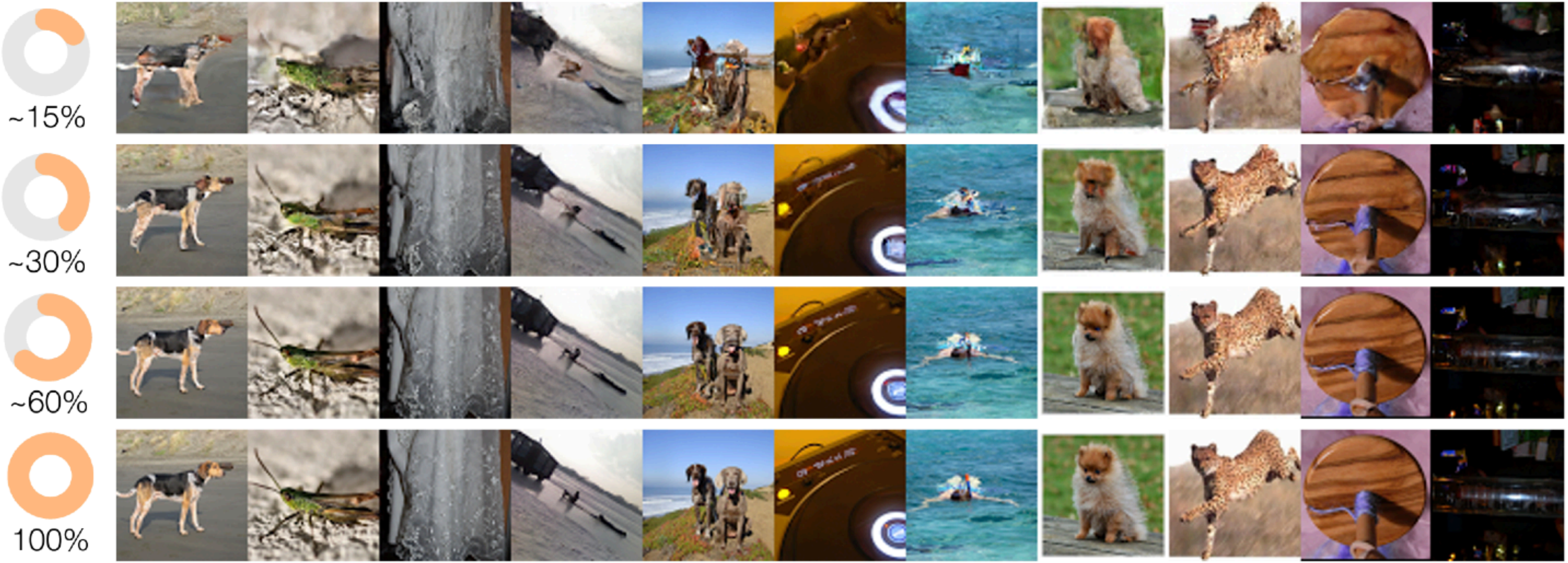
$$\mathbf{y}_a = \mathbf{x}_a$$
$$\mathbf{y}_b = \mathbf{x}_b + \lfloor t\left(\mathbf{x}_a\right)\rceil$$

where $\lfloor \cdot \rceil$ is the rounding operator and we use the straight-through estimator (STE) during training.

2. **Permutation layers**

 Hoogeboom, E., Peters, J., vd. Berg, R., & Welling, M. (2019). Integer discrete flows and lossless compression. *NeuriPS*

INTEGER DISCRETE FLOWS



Progressive display of the data stream for images.

Hoogeboom, E., Peters, J., vd. Berg, R., & Welling, M. (2019). Integer discrete flows and lossless compression. *NeuriPS*

GENERAL INVERTIBLE TRANSFORMATIONS

Proposition 3.1 (**[23]**) *Let us take* $\mathbf{x}, \mathbf{y} \in \mathcal{X}$. *If binary transformations* $\circ$ *and* $\triangleright$ *have inverses* $\bullet$ *and* $\blacktriangleleft$, *respectively, and* $g_2, \ldots, g_D$ *and* $f_1, \ldots, f_D$ *are arbitrary functions, where* $g_i : \mathcal{X}_{1:i-1} \to \mathcal{X}_i$, $f_i : \mathcal{X}_{1:i-1} \times \mathcal{X}_{i+1:n} \to \mathcal{X}_i$, *then the following transformation from* $\mathbf{x}$ *to* $\mathbf{y}$:

$$
\begin{aligned}
y_1 &= x_1 \circ f_1(\emptyset, \mathbf{x}_{2:D}) \\
y_2 &= (g_2(y_1) \triangleright x_2) \circ f_2(y_1, \mathbf{x}_{3:D}) \\
&\ldots \\
y_d &= (g_d(\mathbf{y}_{1:d-1}) \triangleright x_d) \circ f_d(\mathbf{y}_{1:d-1}, \mathbf{x}_{d+1:D}) \\
&\ldots \\
y_D &= (g_D(\mathbf{y}_{1:D-1}) \triangleright x_D) \circ f_D(\mathbf{y}_{1:D-1}, \emptyset)
\end{aligned}
$$

is invertible.

 Tomczak, J. M. (2021). General Invertible Transformations for Flow-based Generative Modeling. ICML Workshop INNF+

GENERAL INVERTIBLE TRANSFORMATIONS

Example:

$$\mathbf{y}_a = \mathbf{x}_a + \lfloor t\left(\mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d\right) \rceil$$

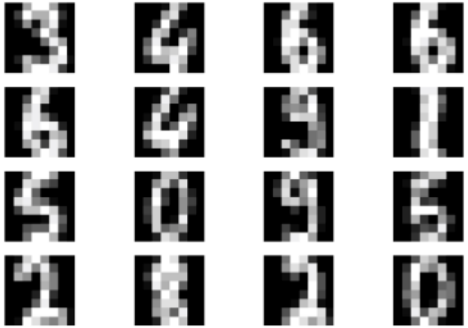
$$\mathbf{y}_b = \mathbf{x}_b + \lfloor t\left(\mathbf{y}_a, \mathbf{x}_c, \mathbf{x}_d\right) \rceil$$

$$\mathbf{y}_c = \mathbf{x}_c + \lfloor t\left(\mathbf{y}_a, \mathbf{y}_b, \mathbf{x}_d\right) \rceil$$

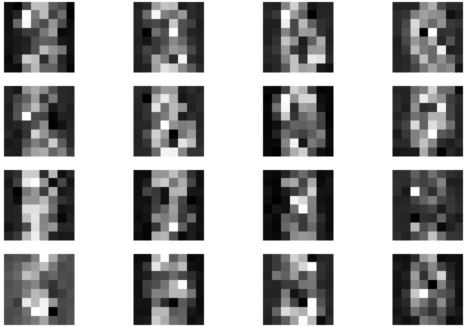
$$\mathbf{y}_d = \mathbf{x}_d + \lfloor t\left(\mathbf{y}_a, \mathbf{y}_b, \mathbf{y}_c\right) \rceil$$

 Tomczak, J. M. (2021). General Invertible Transformations for Flow-based Generative Modeling. ICML Workshop INNF+

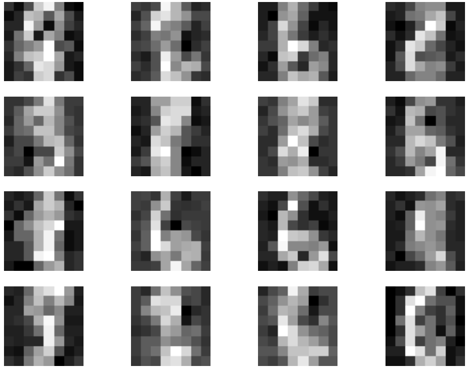
GENERAL INVERTIBLE TRANSFORMATIONS FOR IDF (EXAMPLE)



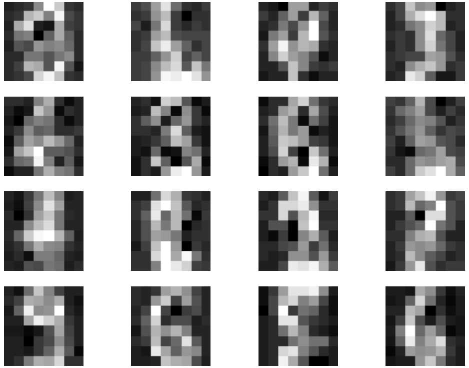
Real images



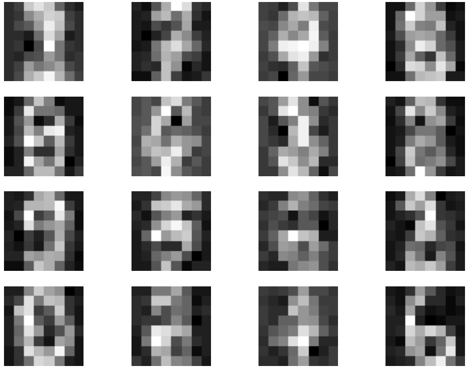
RealNVP



IDF



IDF-GIT(4)



IDF-GIT(8)

CONCLUSION

Flow-based models are powerful and theoretically-grounded.

Flow-based models **may suffer from serious issues**.

CONCLUSION

Flow-based models are powerful and theoretically-grounded.

Flow-based models **may suffer from serious issues**.

Flow-based models for **discrete variables with finite domains may not learn any distribution**.

Flow-based models for integer-valued discrete variables seem to be much better option!

CONCLUSION

Flow-based models are powerful and theoretically-grounded.

Flow-based models **may suffer from serious issues**.

Flow-based models for **discrete variables with finite domains may not learn any distribution**.

Flow-based models for integer-valued discrete variables seem to be much better option!

We are getting better transformations!

:

Jakub M. Tomczak
Computational Intelligence group
Vrije Universiteit Amsterdam

Webpage: https://jmtomczak.github.io/

Github: https://github.com/jmtomczak

Twitter: https://twitter.com/jmtomczak