11-695: AI Engineering
Normalizing Flows

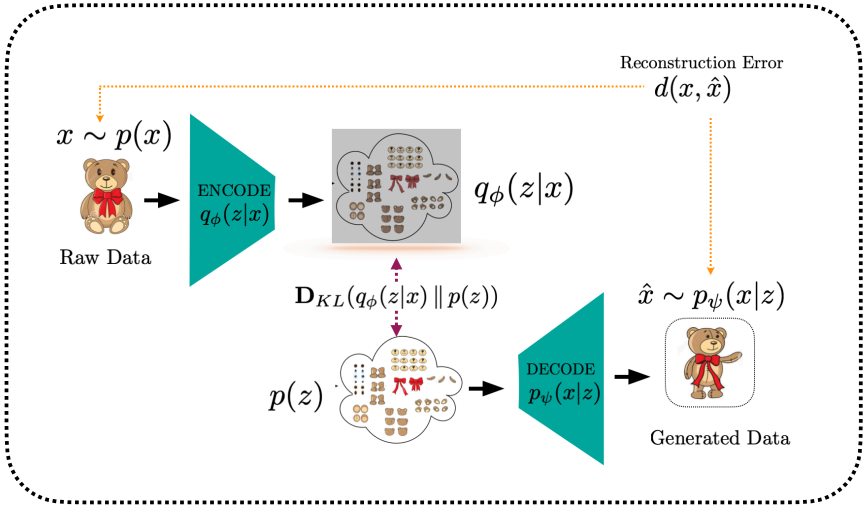
LTI/SCS

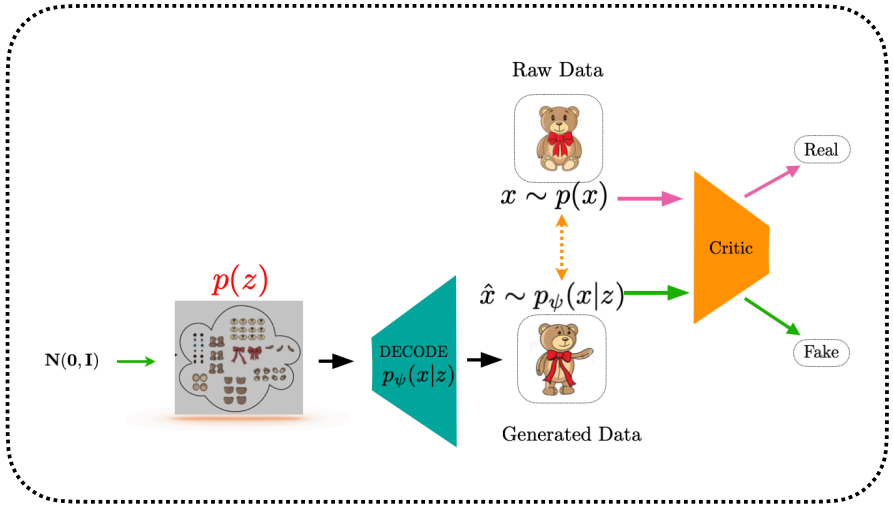
Spring 2020

Outline

About loglikelihood $\log p(x)$

Carnegie Mellon





- VAE tries to get lower-bound approx of $\log p(x)$

$$\max_{\phi,\psi} (\text{ELBO} = \log p(x) - \mathbf{D}_{KL}(q_\phi(z|x) \| p(z)))$$

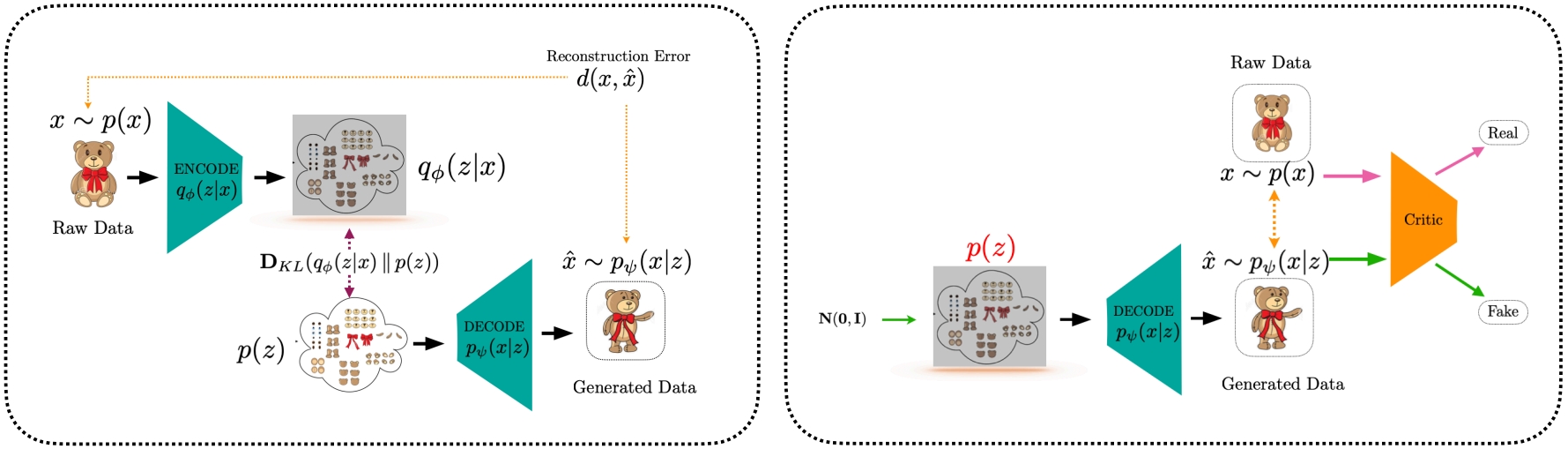
- GAN tries to fool D, so not having a method to calculate $\log p(x)$

$$\min_{\psi} [\mathbb{E}_{x \sim Data} \log D_\theta(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_\theta(G_\psi(z))]$$

$$= \min_{\psi} \mathbb{E}_{z \sim p(z)} \log(1 - D_\theta(G_\psi(z))]$$

About latent representation z from data

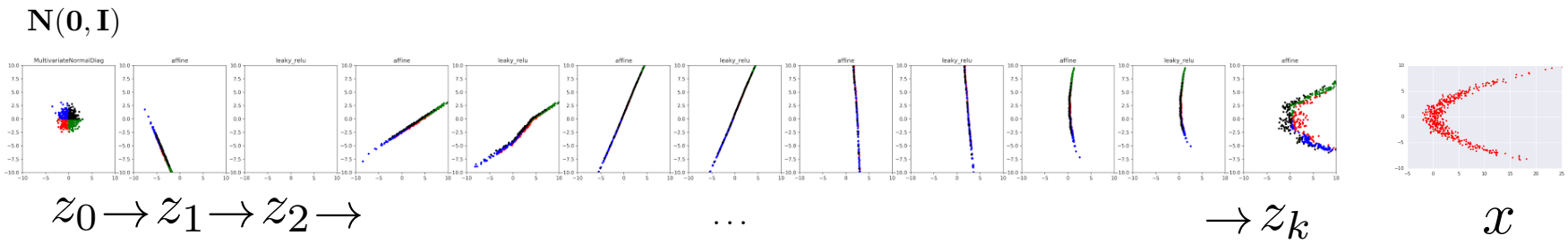
Carnegie Mellon



- The goal of representation learning, for unsupervised learning and any downstream task
  - VAE get a structured approx. latent z from Encoder
  - GAN does not have a learned one ($\rightarrow$ implicit, likelihood-free)
- Many time we want **exact** $p(x)$ and latent $p(z|x)$?

Having exact likelihood and latent

Carnegie Mellon

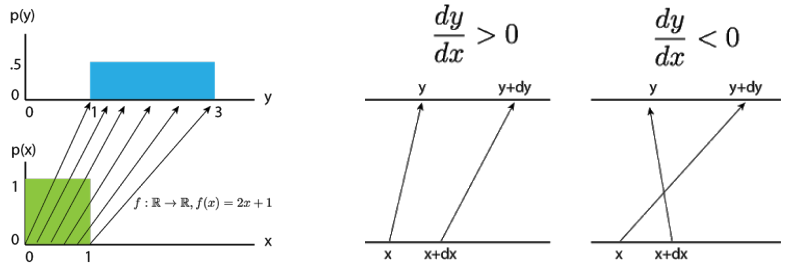


- Start from a simple distribution, *e.g.* $\mathbf{N}(\mathbf{0}, \mathbf{I})$ and transform it to data distribution[1]
- Usually, using a series of transformation (more later):
  - Previously, we approx. transformation using NNs, *e.g.* forward by E and backward by D
  - Now how do we have exact calculation?

[1] This figure is for illustration purpose only. In practice, the notations are reversed! Image credit: Eric Jang

Outline

Simple case of 2 variables x, y



- $X, Y \in \mathbb{R}^1$ and $Y = f(X) = 2X + 1 \Leftrightarrow X = f^{-1}(Y) = \frac{Y-1}{2}$
- Step dx corresponds to dy so that (think about volumes):

$$p_X(x)\,dx = p_Y(y)\,dy \qquad \text{(prob. preservation)} \quad (1)$$

$$p_X(x) = \left|\frac{dy}{dx}\right| p_Y(y) = \left|\frac{df(x)}{dx}\right| p_Y(f(x)) \qquad \text{(prob. non-negative)} \quad (2)$$

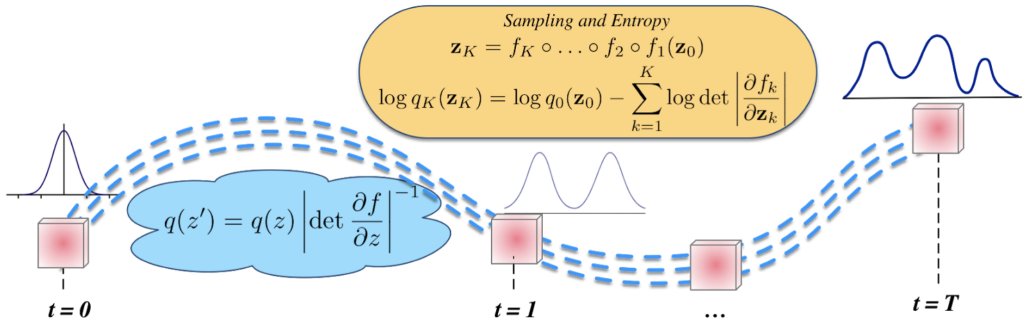
$$p_X(x) = \left|\det\left(\frac{df}{dx}\right)\right| p_Y(f(x)) \qquad \text{(multivariate case)} \quad (3)$$

$$\log p_X(x) = \log\left|\det\left(\frac{df}{dx}\right)\right| + \log p_Y(f(x)) \quad (4)$$

Image credit: Eric Jang

General case: series of transformations

Carnegie Mellon



Distribution flows through a sequence of invertible transforms

- $\log \left| \det \left(\frac{df}{dx} \right) \right| = \log |\det \mathbf{J}(f(x))|$ is the log determinant of Jacobian
- One transformation might not work, we usually need a series:

$$z_k = f_k \circ \ldots f_2 \circ f_1(z_0) \tag{5}$$

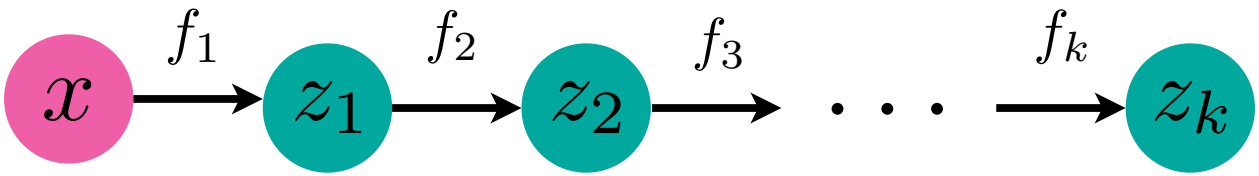
$$\log p_Z(z_0) = \log \left(\prod_{i1}^{k} \left| \det \left(\frac{dz_i}{dz_{i-1}} \right) \right| \right) + \log(p_Z\left(f(z_k)\right) \tag{6}$$

$$= \sum_{i=1}^{k} \log \left(\left| \det \left(\frac{dz_i}{dz_{i-1}} \right) \right| \right) + \log(p_Z\left(f(z_k)\right) \tag{7}$$

Image credit: Renzede & Mohamed 2015

General case: series of transformations

Carnegie Mellon

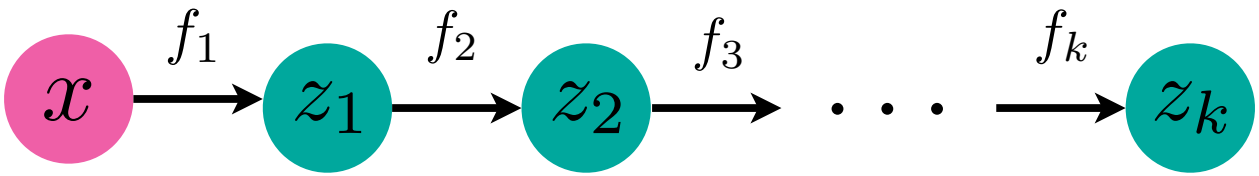


- Set $x = z_0$

$$\log p_X(x) = \sum_{i=1}^{k} \log \left(\left| \det \left(\frac{dz_i}{dz_{i-1}} \right) \right| \right) + \log(p_Z\left(f(z_k)\right) \tag{8}$$

- To model x, we need to maximize RHS of Equation (8)
- Facilitate computation by choosing f_i having triangular Jacobian
- We are free to choose p_Z, if $z_k \sim \mathbf{N}(\mathbf{0}, \mathbf{I})$ then it belongs to a family called: *normalizing* flows

Properties of flows



- Constraints:
  - each step is invertible and differentiable
  - x and z_i are of same dimensions D
- Inference z: do the forward transformation f_i
  - Challenge: To calculate Jacobian determinants efficiently!
- Generate x: do the backward transformation f_i^{-1}
  - Challenge: To build powerful invertible transformations!

Outline

Residual Flows

- General form:

$$\mathbf{z}_i = \mathbf{z}_{i-1} + f_i^{\phi}(\mathbf{z}_{i-1}) \tag{9}$$

- Useful matrix determinant lemma[2] with $\mathbf{A} \in \mathbb{R}^{D \times D}$ and $\mathbf{V}, \mathbf{W} \in \mathbb{R}^{M \times D}$ and $M < D$:

$$\det(\mathbf{A} + \mathbf{V}\mathbf{W}^T) = \det\left(\mathbf{I} + \mathbf{W}^T \mathbf{A}^{-1} \mathbf{V}\right) \det(\mathbf{A}) \tag{10}$$

[2] https://en.wikipedia.org/wiki/Matrix_determinant_lemma

Residual Flows: Planar[4]

- Formulation: single NN with 1 hidden unit

$$\mathbf{z}_i = \mathbf{z}_{i-1} + \mathbf{v}\sigma(\mathbf{w}^T\mathbf{z}_{i-1} + b) \tag{11}$$

- Jacobian determinant in $\mathcal{O}(D)$, using Eq. (10):

$$\det\left(\mathbf{J}(\mathbf{z}_i)\right) = \det\left(\mathbf{I} + \sigma'(\mathbf{w}^T\mathbf{z}_{i-1} + b)\mathbf{v}\mathbf{w}^T\right) \tag{12}$$

$$= 1 + \left[\sigma'(\mathbf{w}^T\mathbf{z}_{i-1} + b)\mathbf{w}^T\right]\mathbf{v} \tag{13}$$

- Extension version: Sylvester flows[3] with many hidden units

[3] https://arxiv.org/pdf/1803.05649.pdf

[4] https://arxiv.org/pdf/1505.05770.pdf

Residual Flows: Radial[5]

Carnegie Mellon

- Formulation with 2 scalars $\alpha > 0, \beta$:

$$\mathbf{z}_i = \mathbf{z}_{i-1} + \frac{\beta}{\alpha + r(\mathbf{z}_{i-1})}(\mathbf{z}_{i-1} - \mathbf{z}_0), \quad r(\mathbf{z}_i) = \|\mathbf{z}_i - \mathbf{z}_0\|_2 \tag{14}$$

- "Contraction" gradually with center $\mathbf{z}_0$
- Jacobian:

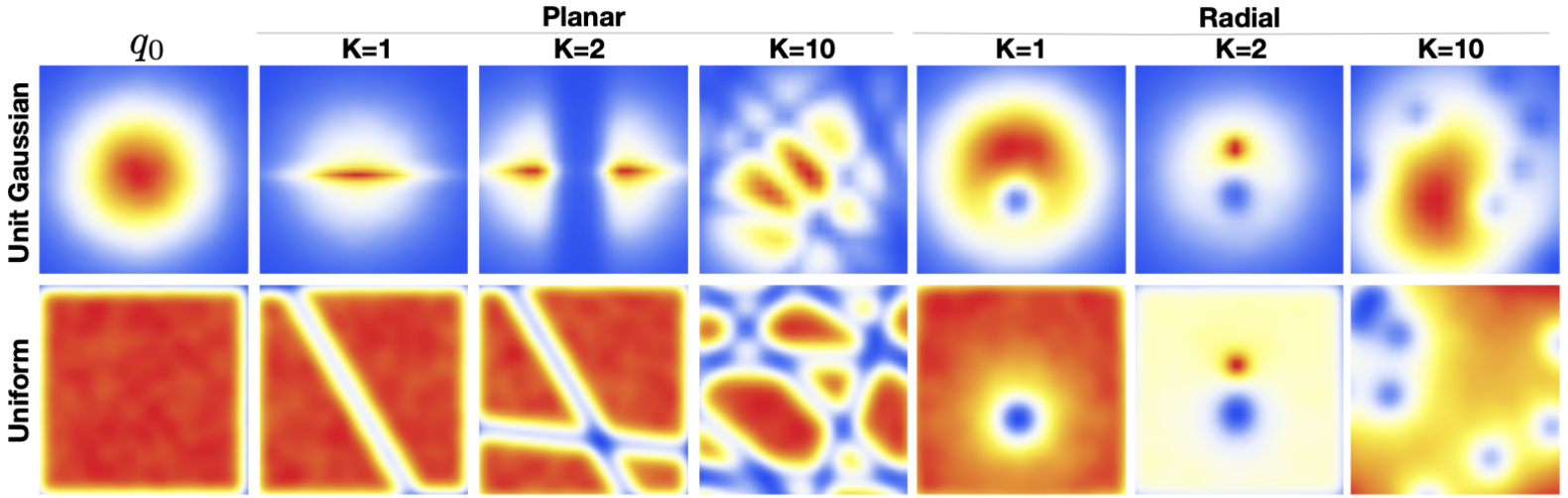
$$\mathbf{J}(\mathbf{z}_i) = \left(1 + \frac{\alpha\beta}{\alpha + r(\mathbf{z}_{i-1})}\right)\mathbf{I} - \frac{\beta}{r(\mathbf{z}_{i-1})(\alpha + r(\mathbf{z}_{i-1}))^2}\|\mathbf{z}_{i-1} - \mathbf{z}_0\|^2 \tag{15}$$

- Jacobian determinant in $\mathcal{O}(D)$, using Eq. (10):

$$\det\left(\mathbf{J}(\mathbf{z}_i)\right) = \left(1 + \frac{\alpha\beta}{(\alpha + r(\mathbf{z}_{i-1}))^2}\right)\left(1 + \frac{\beta}{\alpha + r(\mathbf{z}_{i-1})}\right)^{D-1} \tag{16}$$

[5] https://ri.conicet.gov.ar/bitstream/handle/11336/8930/CONICET_Digital_Nro.12124.pdf

Residual Flows: Results

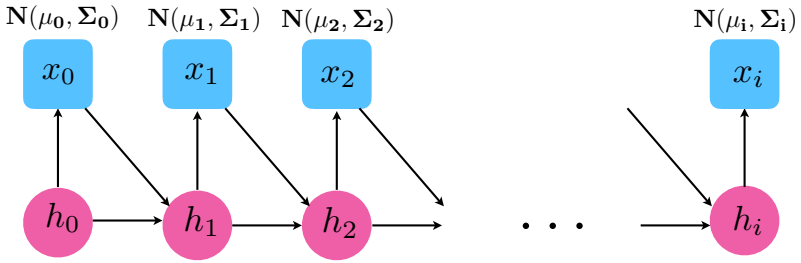


- No analytical form for inverse so can hardly be used for generative models
- But can be used for variational approximation

Image credit: Renzede & Mohamed 2015

Outline

Autoregressive Model



- General form:

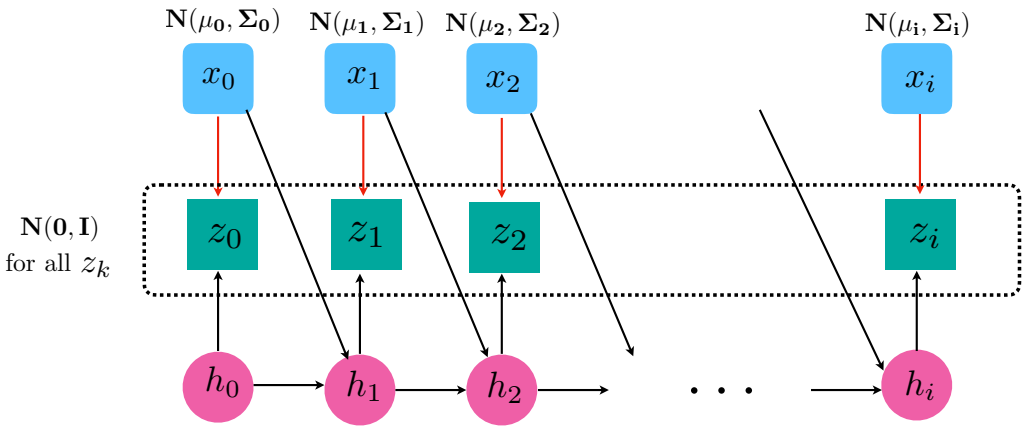
$$p(x_i) = p(x_1) \prod_{k}^{i=2} p(x_k | \mathbf{x}_{<k}) \tag{17}$$

- Can be represented with a transformer $\mathcal{T}$ and conditioners $\mathcal{C}_i$:[6]

$$x_i = \mathcal{T}(x_{i-1}, \mathbf{h}_i), \quad \mathbf{h}_i = \mathcal{C}_i(\mathbf{x}_{<i}) \tag{18}$$

[6] http://www.jmlr.org/papers/volume17/16-272/16-272.pdf

Autoregressive Flows: Invertibility



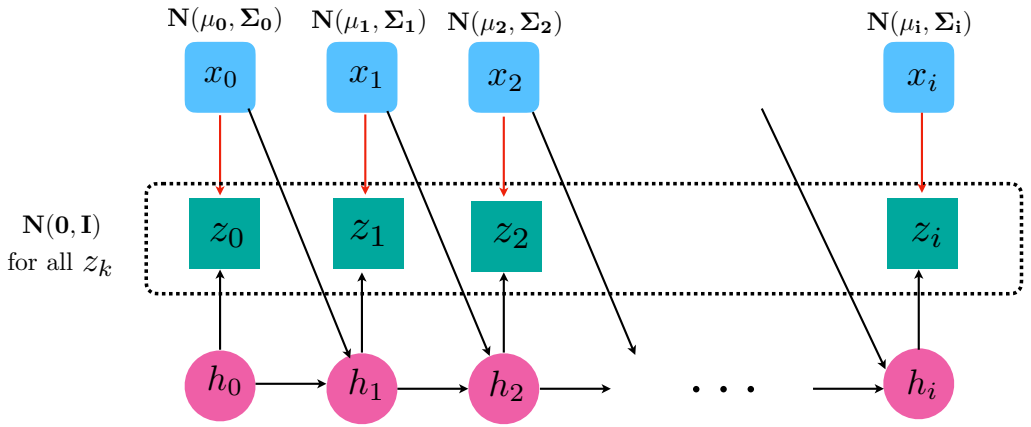
- Now the output is z instead of x (inference process):

$$z_i = \mathcal{T}(x_{i-1}, \mathbf{h}_i), \quad \mathbf{h}_i = \mathcal{C}_i(\mathbf{x}_{<i}) \tag{19}$$

- If $\mathcal{T}$ is invertible, then the sampling process happens as:

$$x_i = \mathcal{T}^{-1}(z_i, \mathbf{h}_i), \quad \mathbf{h}_i = \mathcal{C}_i(\mathbf{x}_{<i}) \tag{20}$$

Autoregressive Flows: Jacobian



- Jacobian is triangular because the components $w.r.t$ $\mathbf{z}_{>i}$ are zeros:

$$\mathbf{J}(\mathbf{z}) = \begin{bmatrix} \frac{\partial \mathcal{T}}{\partial z_1}(z_1, \mathbf{h}_1) & & \mathbf{0} \\ & \ddots & \\ \mathbf{nonzero}(\mathbf{z}) & & \frac{\partial \mathcal{T}}{\partial z_D}(z_D, \mathbf{h}_D) \end{bmatrix} \tag{21}$$

- Log Jacobian determinant:

$$\log \left| \det \mathbf{J}(\mathbf{z}) \right| = \log \left| \prod_{j=1}^{D} \frac{\partial \mathcal{T}}{\partial z_i}(z_i, \mathbf{h}_i) \right| = \sum_{j=1}^{D} \left| \frac{\partial \mathcal{T}}{\partial z_i}(z_i, \mathbf{h}_i) \right| \tag{22}$$

Autoregressive Flows: Conditioners $\mathcal{C}_i$

Carnegie Mellon

- Can be a RNN such as LSTM, GRU but not popular[7] [8]
- Masking (as in Transformer architecture): is easy to evalulate, popular, avoid recurrent computation [9] [10] [11] [12] [13] [14]
- Coupling layers: divide $\mathbf{h} = [\mathbf{h}_{1:d}; \mathbf{h}_{d+1:D}] = [\text{const}, \text{NN}(\mathbf{z}_{\leq d})]$
  - Also splits $\mathbf{z} = [\mathbf{z}_{1:d}; \mathbf{z}_{d+1:D}] = [\mathbf{x}_{1:d}; \mathbf{x}_{d+1:D} \odot s(\mathbf{x}_{1:d}) + t(\mathbf{x}_{1:d})]$
  - Requires permutation of dimensions in $\mathbf{z}$ but is efficient [15] [16] [17] [18] [19]

[7] https://arxiv.org/pdf/1606.04934.pdf

[8] https://arxiv.org/pdf/1801.09819.pdf

[9] https://arxiv.org/pdf/1606.04934.pdf

[10] https://arxiv.org/pdf/1705.07057.pdf

[11] https://arxiv.org/pdf/1804.00779.pdf

[12] http://auai.org/uai2019/proceedings/papers/511.pdf

[13] https://arxiv.org/pdf/1907.07945.pdf

[14] https://arxiv.org/pdf/1902.04208.pdf

[15] https://arxiv.org/pdf/1605.08803.pdf

[16] https://arxiv.org/pdf/1807.03039.pdf

[17] https://arxiv.org/pdf/1811.00002.pdf

[18] https://arxiv.org/pdf/1811.02155.pdf

[19] https://arxiv.org/pdf/1902.00275.pdf

Image credit: Jaini *et al.* 2019

Autoregressive Flows: Some Models[20]

Carnegie Mellon

Table 1. Various auto-regressive and flow-based methods expressed under a unified framework. All the conditioners can take inputs $\mathbf{x}$ instead of $\mathbf{z}$. The symbol 🔗 is used for weight sharing, 👻 for use of masks for efficient implementation, 🏛 for universality of the method and, Δ if the method learns a triangular transformation explicitly (E) or implicitly (I). ? implies that universality of these methods has neither been proved or disproved although it can now be analyzed with ease using our framework. $S_j(z_j; \boldsymbol{\theta}_j)$ is defined in eq. (6) and $\mathfrak{P}_{2r+1}(z_j; \mathbf{a}_j)$ is defined in eq. (8).

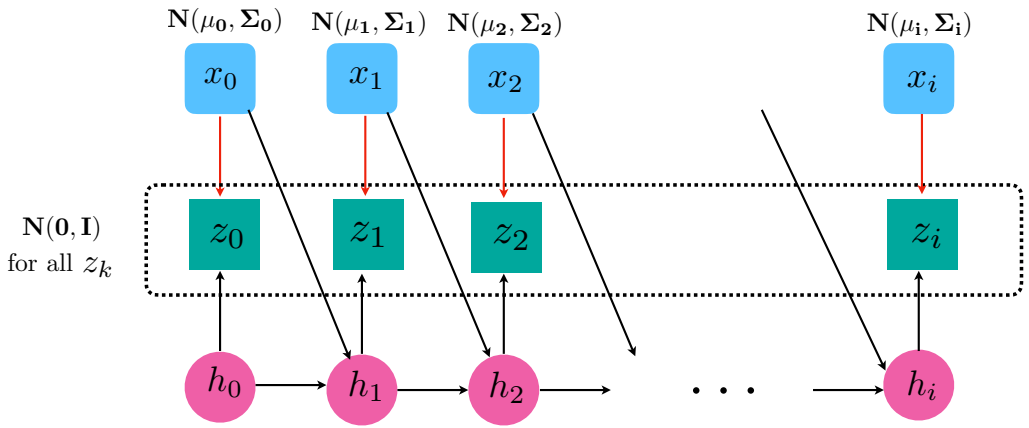
Model	conditioner C_j output	$T_j\big(z_j\,; C_j(z_1, \ldots, z_{j-1})\big)$	🔗	👻	🏛	Δ
Mixture (e.g. McLachlan & Peel, 2004)	$\boldsymbol{\theta}_j$	$S_j(z_j; \boldsymbol{\theta}_j)$	✗	✗	✓	I
(Bengio & Bengio, 1999)	$\boldsymbol{\theta}_j(z_{<j})$	$S_j(z_j; \boldsymbol{\theta}_j)$	✗	✗	?	I
MADE (Germain et al., 2015)	$\boldsymbol{\theta}_j(z_{<j})$	$S_j(z_j; \boldsymbol{\theta}_j)$	✓	✓	?	I
NICE (Dinh et al., 2015)	$\mu_j(z_{<l})$	$z_j + \mu_j \cdot \mathbf{1}_{j \notin [l]}$	✗	✗	?	E
NADE (Uria et al., 2016)	$\boldsymbol{\theta}_j(z_{<j})$	$S_j(z_j; \boldsymbol{\theta}_j)$	✓	✗	?	I
IAF (Kingma et al., 2016)	$\sigma_j(z_{<j}),\ \mu_j(z_{<j})$	$\sigma_j z_j + (1 - \sigma_j)\mu_j$	✓	✓	?	E
MAF (Papamakarios et al., 2017)	$\alpha_j(z_{<j}),\ \mu_j(z_{<j})$	$z_j \exp(\alpha_j) + \mu_j$	✓	✓	?	E
Real-NVP (Dinh et al., 2017)	$\alpha_j(z_{<l}), \mu_j(z_{<l})$	$\exp(\alpha_j \cdot \mathbf{1}_{j \notin [l]}) \cdot z_j + \mu_j \cdot \mathbf{1}_{j \notin [l]}$	✗	✗	?	E
NAF (Huang et al., 2018)	$\mathbf{w}_j(z_{<j})$	$\text{DNN}(z_j\,; \mathbf{w}_j)$	✓	✓	✓	E
SOS	$\mathbf{a}_j(z_{<j})$	$\mathfrak{P}_{2r+1}(z_j; \mathbf{a}_j)$	✓	✓	✓	E

- Note: this is however not an exhaustive list.

[20] https://arxiv.org/pdf/1905.02325.pdf

Image credit: Jaini *et al.* 2019

Autoregressive Affine Transformation



- Transformation $\mathcal{T}$ becomes:

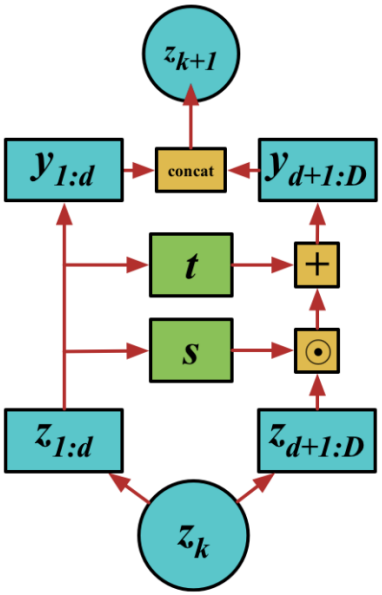
$$z_i = \mathcal{T}(x_i, \mathbf{h}_i) = \exp(\alpha_i) + \beta_i, \quad \mathbf{h}_i = \{\exp(\alpha_i), \beta_i\} \tag{23}$$

- Exponential operation ensures invertibility
- Log Jacobian determinant:

$$\log|\det \mathbf{J}(\mathbf{z})| = \sum_{i=1}^{D} \log|\exp(\alpha_i)| = \sum_{i=1}^{D} \alpha_i \tag{24}$$

Real Non-volume Preserving (NVP)[21]

Carnegie Mellon



- Coupling layer: $\mathbf{z} = [\mathbf{z}_{1:d}, \mathbf{z}_{d_1:D}]$
- First part is only an identity transformation:

$$\mathbf{x}_{1:d} = \mathbf{z}_{1:d} \tag{25}$$

$$\mathbf{x}_{d+1:D} = \mathbf{z}_{d+1:D} \odot \exp(s(\mathbf{z}_{1:d})) + t(\mathbf{z}_{1:d}) \tag{26}$$

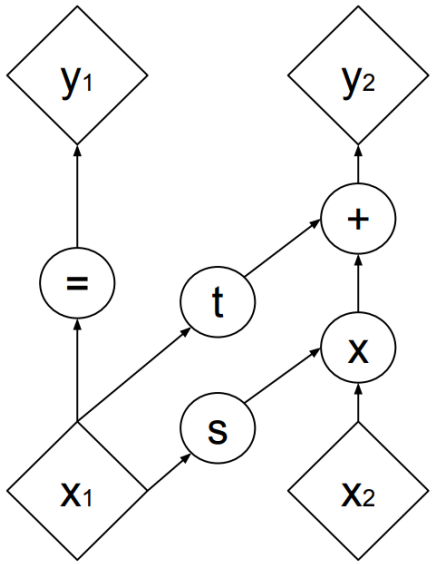
- $s, t : \mathbb{R}^d \mapsto \mathbb{R}^{D-d}$ are for scale and transform
- Recall: Jacobian is triangular:

$$\mathbf{J}(\mathbf{z}) = \begin{bmatrix} \mathbf{I}_d & \mathbf{0} \\ \frac{\partial x_{d+1:D}}{z_{1:d}} & \frac{\partial x_{d+1:D}}{z_{d+1:D}} \end{bmatrix} \tag{27}$$

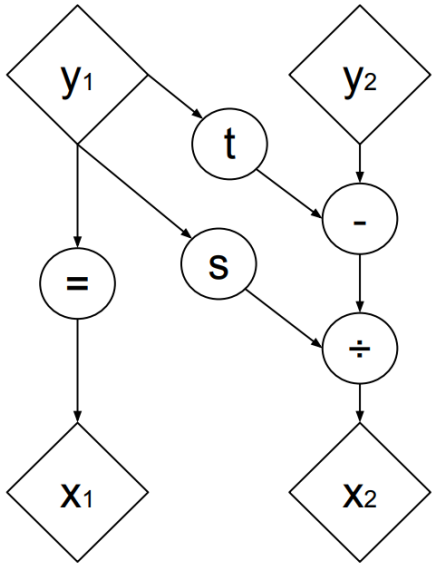
[21] https://arxiv.org/pdf/1605.08803.pdf

Image credit: Renzede & Mohamed 2015

Real-NVP: Invertibility



(a) Forward propagation



(b) Inverse propagation

- Easily attainable via:

$$\mathbf{z}_{1:d} = \mathbf{x}_{1:d} \tag{28}$$

$$\begin{aligned}\mathbf{z}_{d+1:D} &= (\mathbf{x}_{d+1:D} - t(\mathbf{z}_{1:d})) \odot \exp(-s(\mathbf{z}_{1:d})) \\ &= (\mathbf{x}_{d+1:D} - t(\mathbf{x}_{1:d})) \odot \exp(-s(\mathbf{x}_{1:d}))\end{aligned} \tag{29}$$

Image credit: Dinh *et al.* 2017

Real-NVP: Results

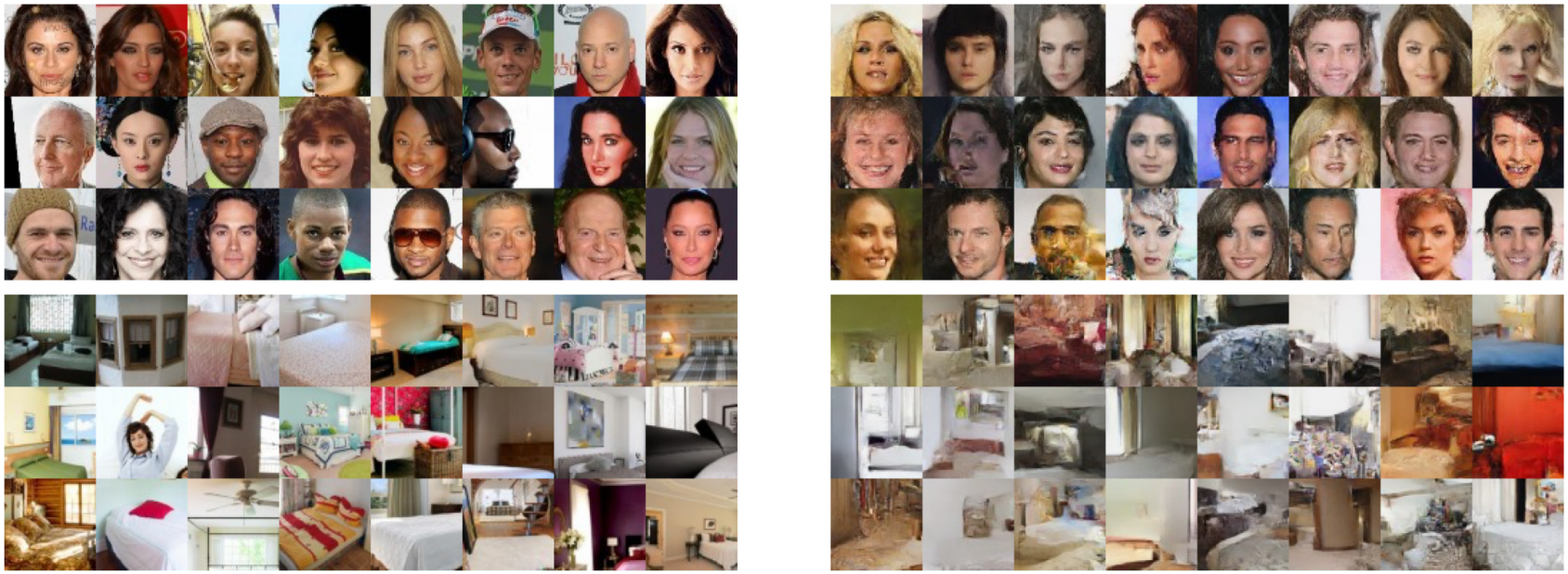
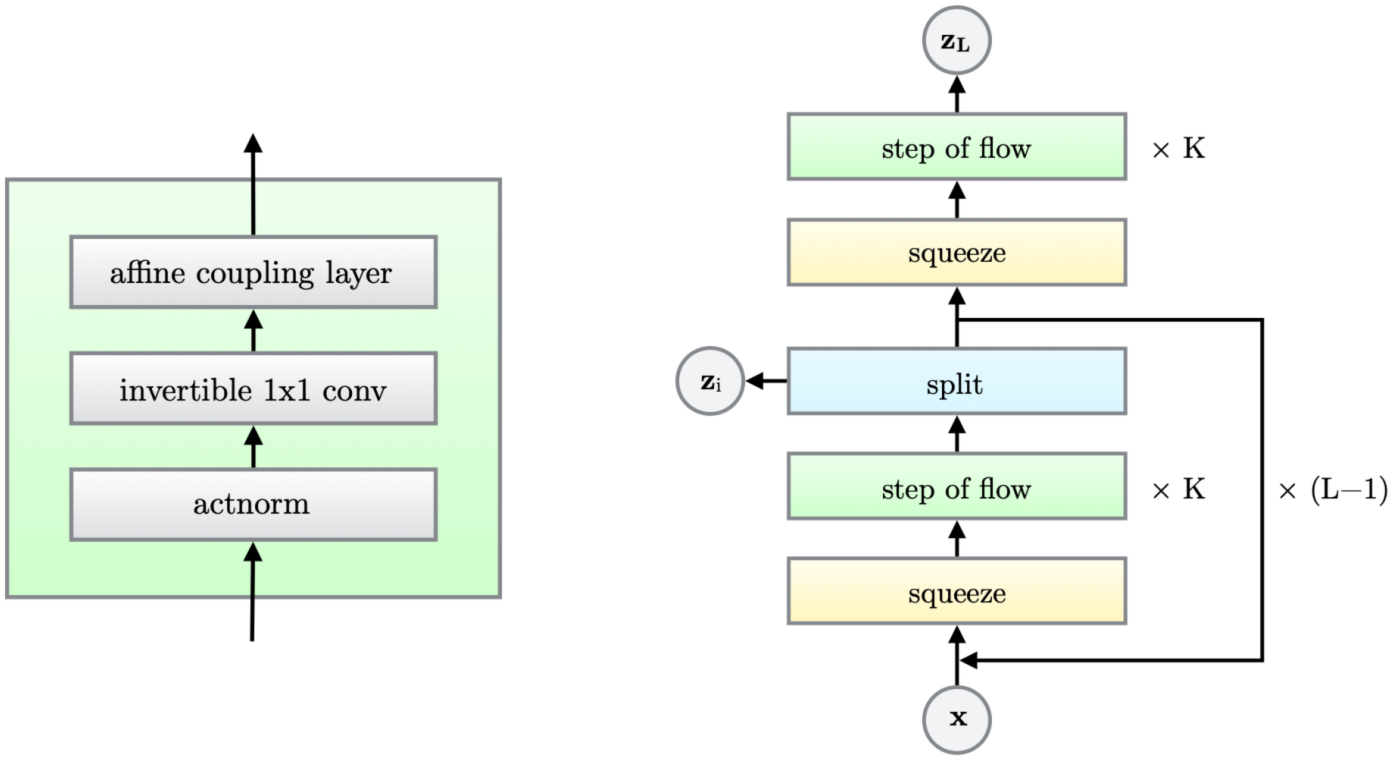


Figure 5: On the left column, examples from the dataset. On the right column, samples from the model trained on the dataset. The datasets shown in this figure are in order: CIFAR-10, Imagenet (32×32), Imagenet (64×64), CelebA, LSUN (bedroom).

Image credit: Dinh *et al.* 2017

Glow[22]: Architecture

Carnegie Mellon



(a) One step of our flow.

(b) Multi-scale architecture (Dinh et al., 2016).

- Extends from Real-NVP with additive coupling layers
- Add 1x1 convolution and arcnorm operations

[22] https://arxiv.org/pdf/1807.03039.pdf

Image credit: Kingma and Dhariwal 2018

Glow: Results

Image credit: Kingma and Dhariwal 2018

Glow: Interpolation

Carnegie Mellon

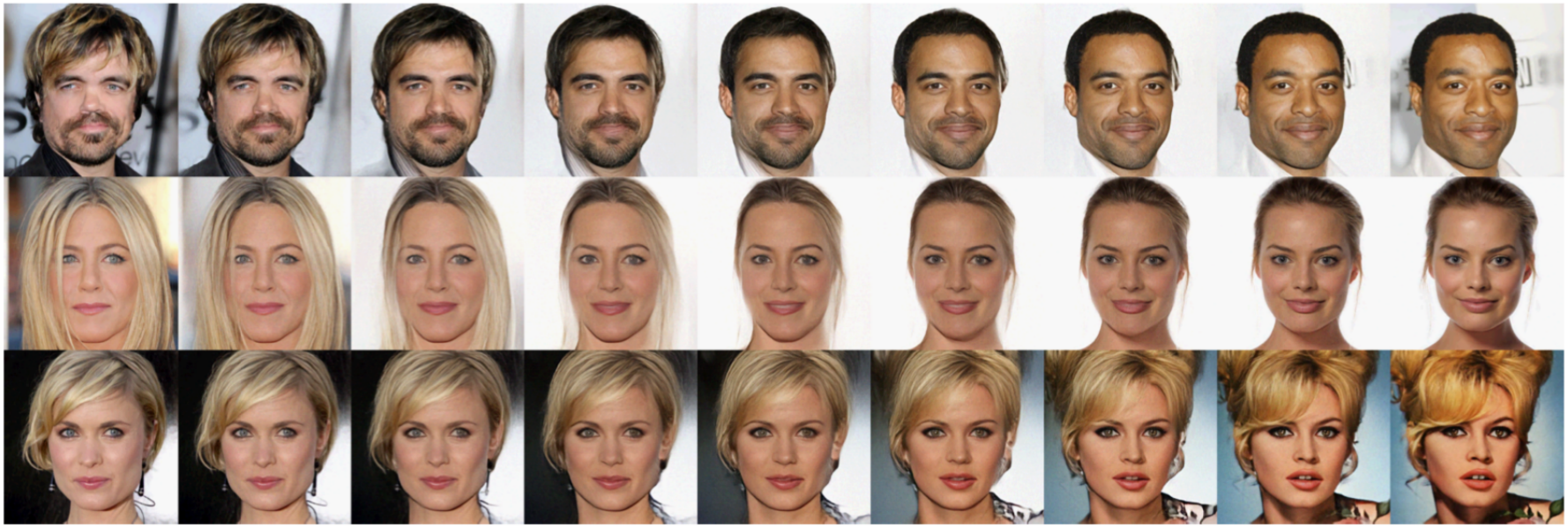


Figure 5: Linear interpolation in latent space between real images

Image credit: Kingma and Dhariwal 2018

Non-Affine Transformations[31]

Carnegie Mellon

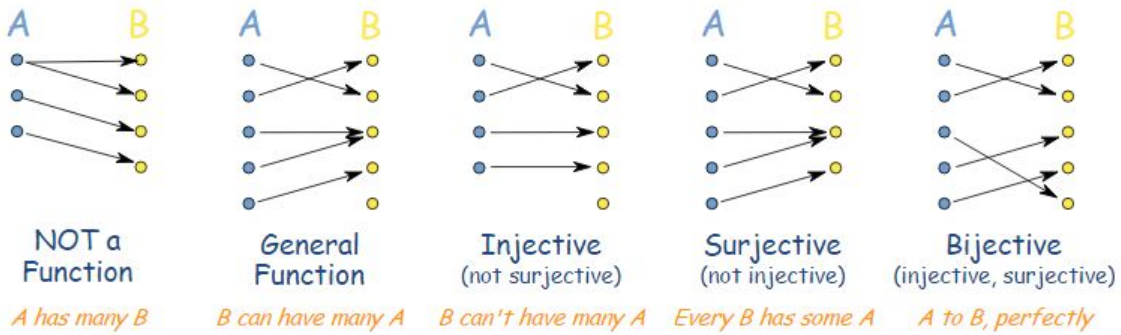
- Neural Transformations: conic combination + composition
  - Neural autoregressive flow (NAF)[23]
  - Block-NAF (B-NAF)[24]
  - Flow++[25]
- Integration-based Transformations
  - Unconstrained monotonic neural networks.[26]
  - Sum-of-squares polynomial flow[27]
- Neural spline flows: have analytic inverse
  - Neural importance sampling.[28]
  - Cubic-spline flows[29]
  - Neural spline flows[30]

[23] https://arxiv.org/pdf/1804.00779.pdf
[24] https://arxiv.org/pdf/1904.04676.pdf
[25] https://arxiv.org/pdf/1902.00275.pdf
[26] https://arxiv.org/pdf/1908.05164.pdf
[27] https://arxiv.org/pdf/1905.02325.pdf
[28] https://arxiv.org/pdf/1808.03856.pdf
[29] https://arxiv.org/pdf/1906.02145.pdf
[30] https://arxiv.org/pdf/1906.04032.pdf
[31] https://arxiv.org/pdf/1912.02762.pdf

Image credit: Kingma and Dhariwal 2018

Outline

Discrete Flows: Variables



- Recall: we need invertible mapping, so we choose *bijective*
- A bijective map f, variable X, prior Z:

$$p_X(x) = p_Z(z), \quad z = f(x) \tag{30}$$

- No Jacobian term involved! An advantage, as well as a restriction.
- Intuitively, $p_X(x)$ does not change, but rather permute $p_Z(z)$, and so we need to model $p_Z(.)$ carefully.

Image credit: http://www.aplustopper.com/injective-surjective-bijective/

Integer Discrete Flows (IDF)[32]

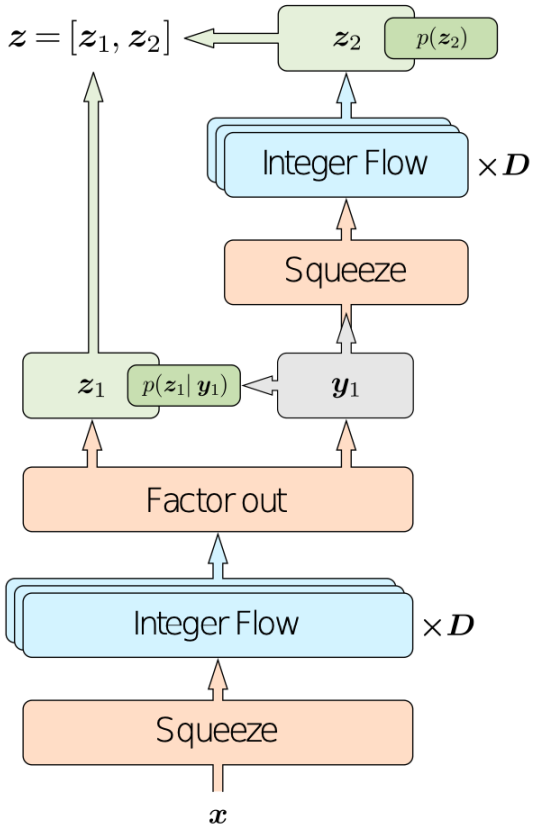


Figure 4: Example of a 2-level flow architecture. The squeeze layer reduces the spatial dimensions by two, and increases the number of channels by four. A single integer flow layer consists of a channel permutation and an integer discrete coupling layer. Each level consists of D flow layers.

- Follows autoregressive model with coupling layers ($\mathcal{T} : \mathbb{Z} \to \mathbb{Z}$):

$$\mathcal{T}(z_i, \mathbf{h}_i) = z_i + \text{round}(\mathbf{h}_i) \tag{31}$$

$$h_i = \mathcal{C}_i(\mathbf{z}_{<i}) \tag{32}$$

[32] https://arxiv.org/pdf/1905.07376.pdf

Image credit: Emiel Hoogeboom *et al.* 2019

IDF: Results

Carnegie Mellon

Table 1: Compression performance of IDFs on CIFAR10, ImageNet32 and ImageNet64 in bits per dimension, and compression rate (shown in parentheses). The Bit-Swap results are retrieved from [23]. The column marked IDF† denotes an IDF trained on ImageNet32 and evaluated on the other datasets.

Dataset	IDF	IDF†	Bit-Swap	FLIF [35]	PNG	JPEG2000
CIFAR10	**3.34 (2.40×)**	3.60 (2.22×)	3.82 (2.09×)	4.37 (1.83×)	5.89 (1.36×)	5.20 (1.54×)
ImageNet32	**4.18 (1.91×)**	**4.18 (1.91×)**	4.50 (1.78×)	5.09 (1.57×)	6.42 (1.25×)	6.48 (1.23×)
ImageNet64	**3.90 (2.05×)**	3.94 (2.03 ×)	–	4.55 (1.76×)	5.74 (1.39×)	5.10 (1.56×)

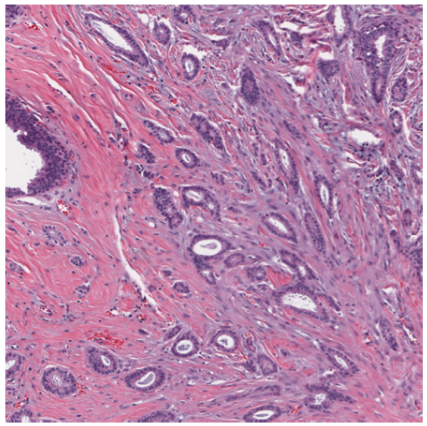


Figure 6: Left: An example from the ER + BCa histology dataset. Right: 625 IDF samples of size 80×80px.

Figure 7: 49 samples from the ImageNet 64×64 IDF.

- Good results in compression and generation.

Image credit: Emiel Hoogeboom *et al.* 2019

Bipartie Discrete Flows[33]

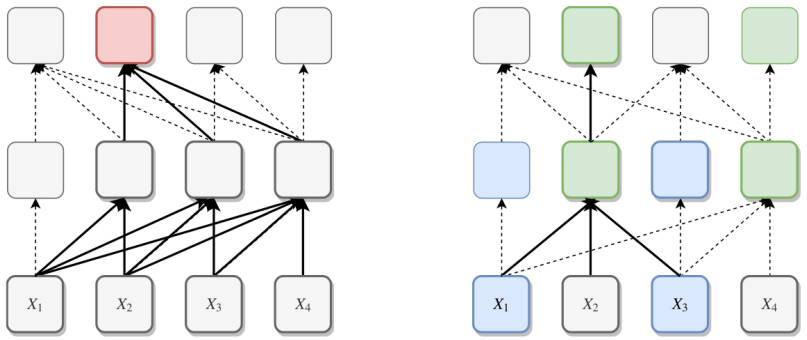


Figure 1: Flow transformation when computing log-likelihoods. **(a)** Discrete autoregressive flows stack multiple levels of autoregressivity. The receptive field of output unit 2 (red) includes left and right contexts. **(b)** Discrete bipartite flows apply a binary mask (blue and green) which determines the subset of variables to transform. With 2 flows, the receptive field of output unit 2 is $\mathbf{x}_{1:3}$.

- Similarly to IDF, this follows autoregressive model with coupling layers ($\mathcal{T} : \mathcal{X} \to \mathcal{X} := \{0, \ldots, K-1\}$), notation from Eq. (23):

$$\mathcal{T}(z_i, \mathbf{h}_i) = z_i + (\alpha_i z_i + \beta_i) \mod K \tag{33}$$

$$\mathbf{h}_i = \{\alpha_i, \beta_i\} = \mathcal{C}_i(\mathbf{z}_{<i}) \tag{34}$$

- Bipatie: just renamed from the coupling operations from Real-NVP (See Equations (25) and (26)).

[33] https://arxiv.org/pdf/1905.10347.pdf

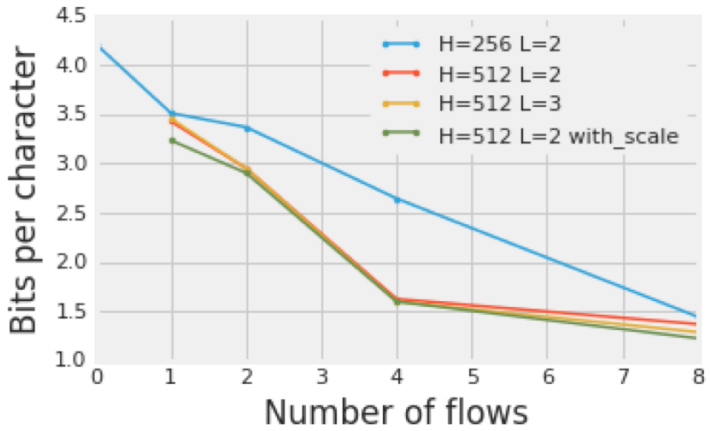
Image credit: Dustin Tran *et al.* 2019

Bipartie Discrete Flows: Results

Carnegie Mellon

	Test NLL (bpc)	Generation
3-layer LSTM (Merity et al., 2018)	**1.18**[3]	3.8 min
Ziegler and Rush (2019) (AF/SCF)	1.46	-
Ziegler and Rush (2019) (IAF/SCF)	1.63	-
Bipartite flow	1.38	**0.17 sec**

Table 3: Character-level language modeling results on Penn Tree Bank.



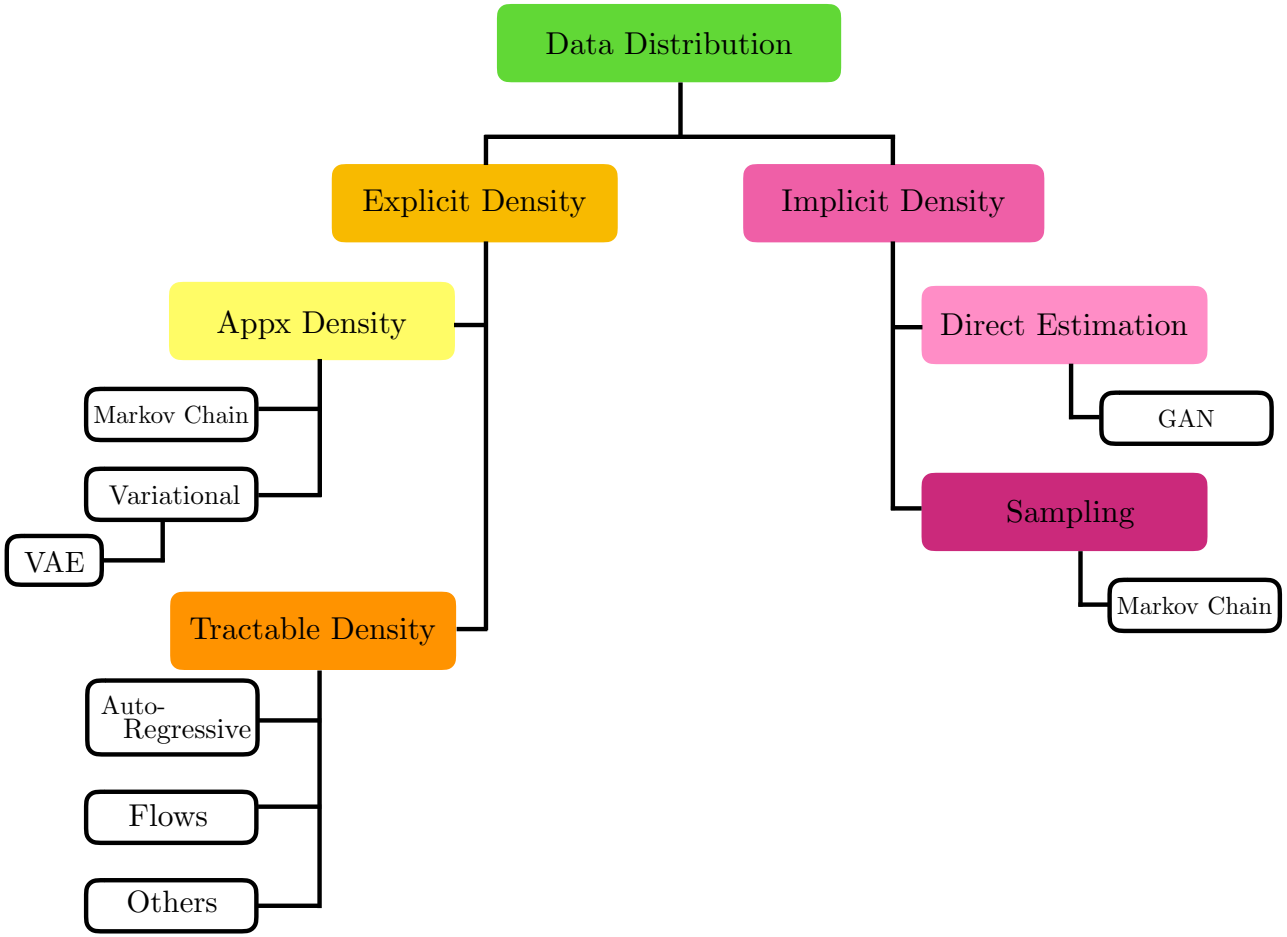
	bpc	Gen.
LSTM (Coojimans+2016)	1.43	19.8s
64-layer Transformer (Al-Rfou+2018)	**1.13**	35.5s
Bipartite flow (4 flows, w/ σ)	1.60	**0.15s**
Bipartite flow (8 flows, w/o σ)	1.29	**0.16s**
Bipartite flow (8 flows, w/ σ)	1.23	**0.16s**

Figure 3: Character-level language modeling results on text8. The test bits per character decreases as the number of flows increases. More hidden units H and layers L in the Transformer per flow, and applying a scale transformation instead of only location, also improves performance.

Image credit: Dustin Tran *et al.* 2019

Outline

Unsupervised Learning Landscape



- Note: this is not an exhaustive diagram

From Goodfellow 2016