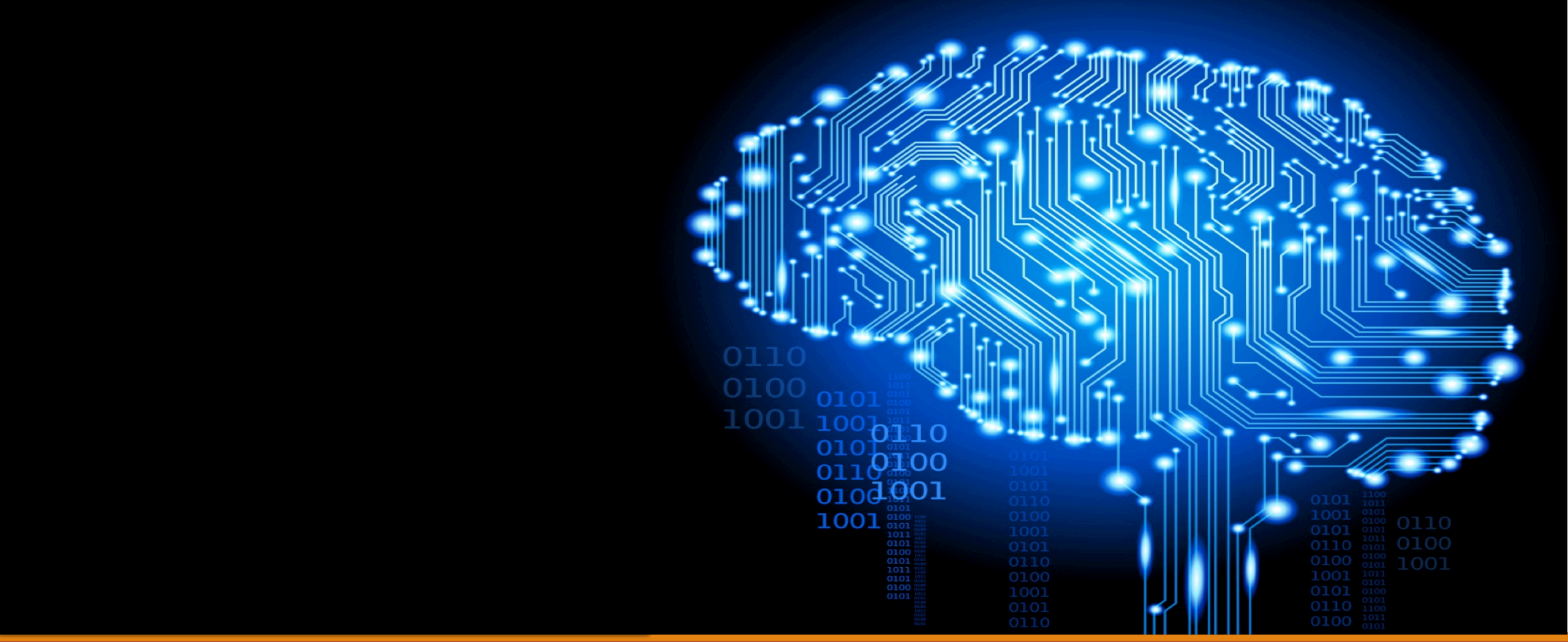


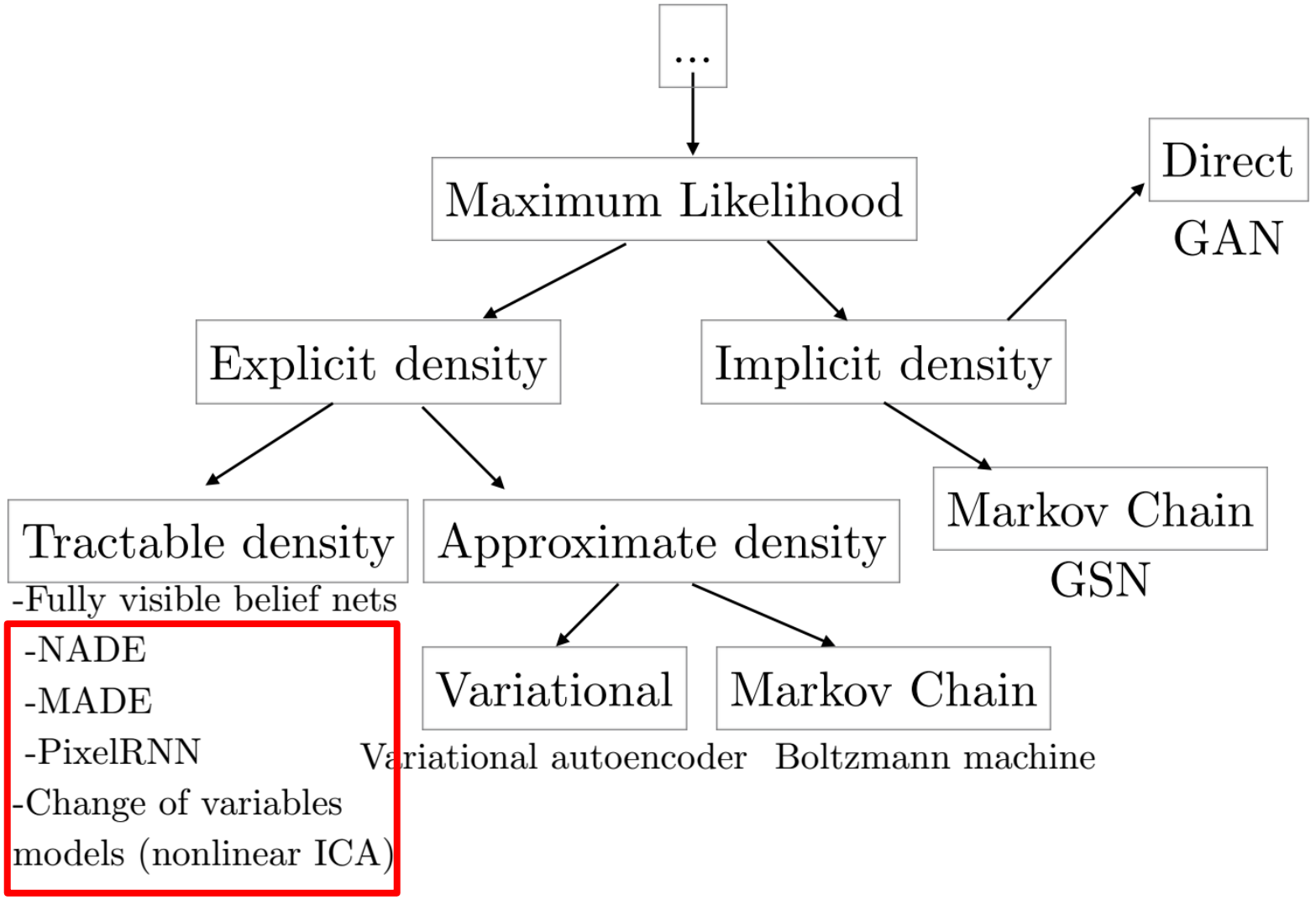
Autoregressive and Flow-based Generative Models

Efstratios Gavves

Lecture overview

- Early autoregressive models
- Modern autoregressive models
- Normalizing flows
- Flow-based models

A map of generative models



Beyond independent dimensions

- Often, in data there is either an order or we can make up an order
  - From a generation point of view, data dimensions depend on each other

Decomposing likelihood of sequential data

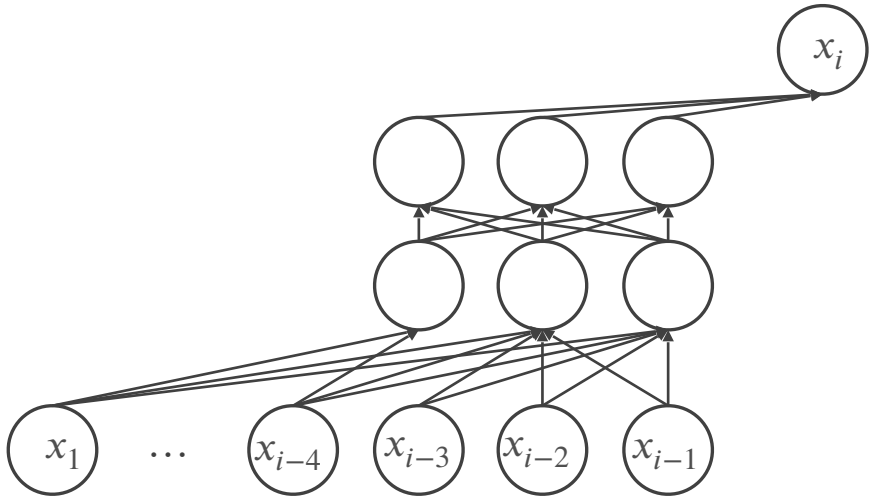
- If $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]$ is sequential, $p(\boldsymbol{x})$ decomposes with chain rule of probabilities

$$p(\boldsymbol{x}) = p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_1, x_2) \cdot \ldots \cdot p(x_d \mid x_1, \ldots, x_{d-1}) = \prod_{i=1}^{d} p(x_i \mid x_{<i})$$

- If $\boldsymbol{x}$ is *not* sequential, we can assume an artificial order
  - *e.g.*, the order with which pixels make (generate) an image
  - This can create artificial bias, however

Deep networks to model conditional likelihoods

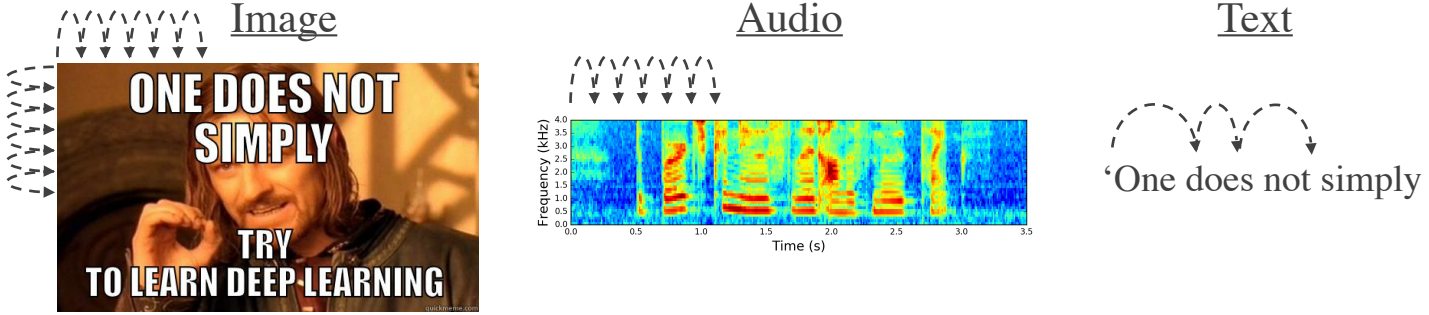
- Model the conditional likelihoods with deep neural networks
  - Logistic regression (Frey et al., 1996), Neural nets (Bengio and Bengio, 2000)
  - *E.g.*, learn a deep net to generate one pixel at a time given past pixels
- The learning objective is to maximize the log-likelihood $\log p(\boldsymbol{x})$
  - If each conditional is tractable, $\log p(\boldsymbol{x})$ is tractable
  - Model conditional probabilities directly and witn no partition functions Z



Tractability vs Flexibility

$$p(\boldsymbol{x}) = p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_1, x_2) \cdot \ldots \cdot p(x_d \mid x_1, \ldots, x_{d-1}) = \prod_{i=1}^{d} p(x_i \mid x_{<i})$$

- Autoregressivity means there is no integral in our computations per se
- The Bayes rule ensures the final quantity to be a probability for as long as the decompositions are probabilities
- With the caveat of sequential computation, so we literally must compute pixel by pixel
  - During the forward pass, the backward pass, inference, sampling



Neural Autoregressive Density Estimation

- Inspired by RBMs but with tractable density estimation
  - Each conditional modelled with sigmoidal neural net like in RBMs
- Parameter matrix W maps past inputs $\boldsymbol{v}_{<i}$ to hidden feature $\boldsymbol{h}_i$
- Parameter matrix V generates pixel v_i given the hidden feature $\boldsymbol{h}_i$

$$p\left(v_i \middle| \boldsymbol{v}_{<i}\right) = \sigma\left(b_i + \left(V^T\right)_{i,\cdot} \boldsymbol{h}_i\right)$$

$$\boldsymbol{h}_i = \sigma\left(\boldsymbol{c} + W_{\cdot,<i} \boldsymbol{v}_{<i}\right)$$

- Teacher forcing
  - During training use ground truth past inputs $\boldsymbol{v}_{<i}$
  - During testing use predicted past inputs $\hat{\boldsymbol{v}}_{<i}$

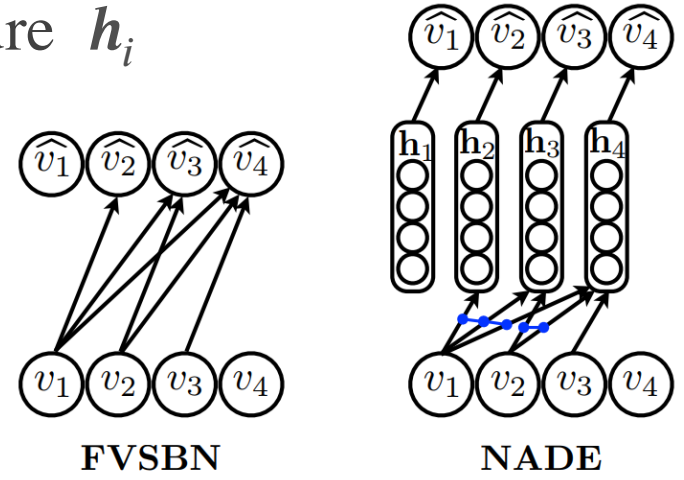


Figure 1: **(Left)** Illustration of a fully visible sigmoid belief network. **(Right)** Illustration of a neural autoregressive distribution estimator. $\hat{v}_i$ is used as a shorthand for $p(v_i = 1|\mathbf{v}_{<i})$. Arrows connected by a blue line correspond to connections with shared or tied parameters.

Larochelle and Murray, Neural Autoregressive Distribution Estimation

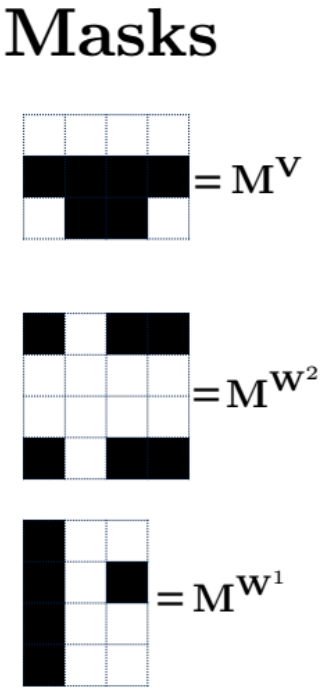
Masked Autoencoder for Distribution Estimation

- Make an autoregressive autoencoder by setting each output x_i depend only on previous outputs $\boldsymbol{x}_{<i}$
  - In autoencoders the output dimensions depend on 'future' dimensions also
- Implement this by introducing a masking matrix M to multiply weights

$$h(\boldsymbol{x}) = g\left(\boldsymbol{b} + \left(W \odot M^W\right) \cdot \boldsymbol{x}\right) \hat{x} = \sigma\left(\boldsymbol{c} + \left(V \odot M^V\right) \cdot h(\boldsymbol{x})\right)$$

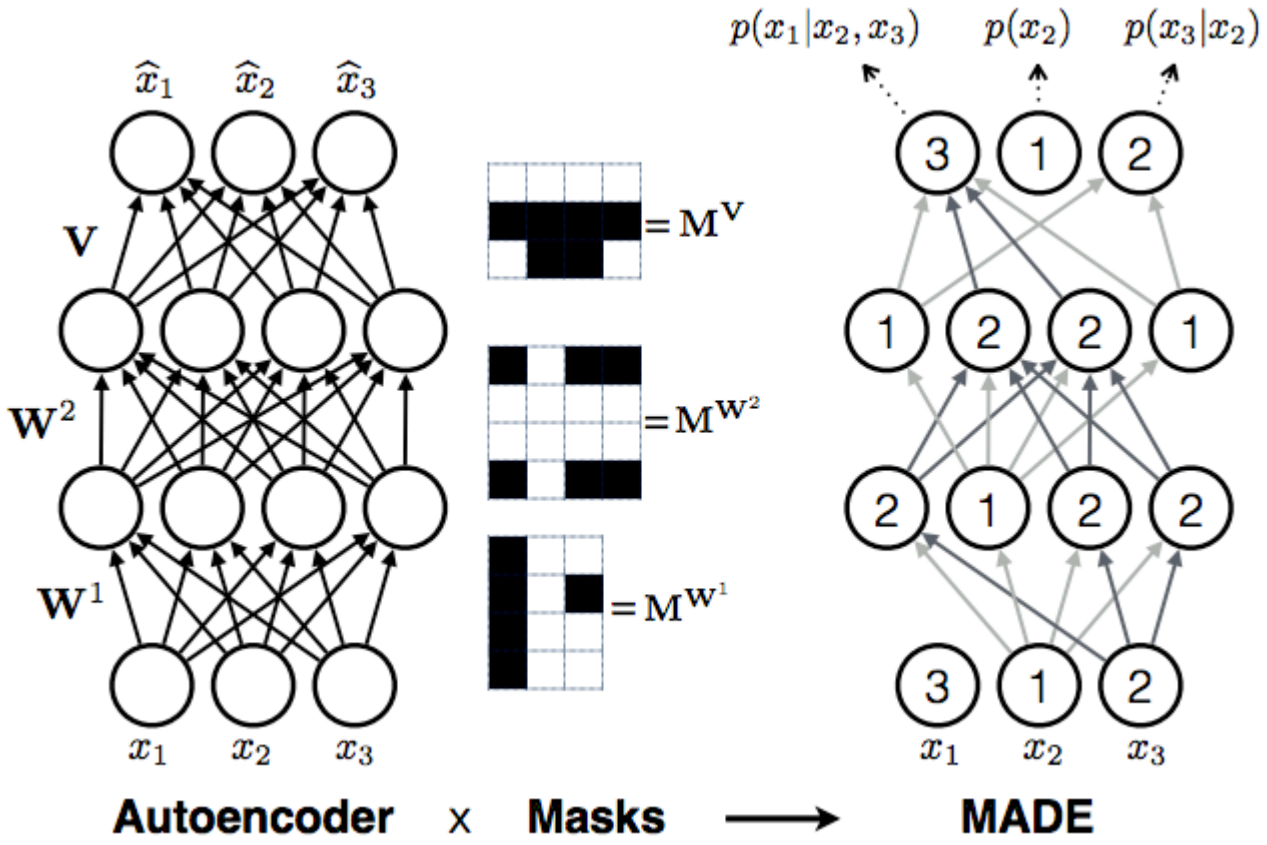
For the k-th neuron the mask column is $M_{k,d} = \begin{cases} 1 & m(k) \geq d \\ 0 & \text{otherwise} \end{cases}$

And $m(k)$ is a integer between 1 and $d-1$



Germain, Gregor, Murray, Larochelle, Masked Autoencoder for Distribution Estimation

MADE architecture



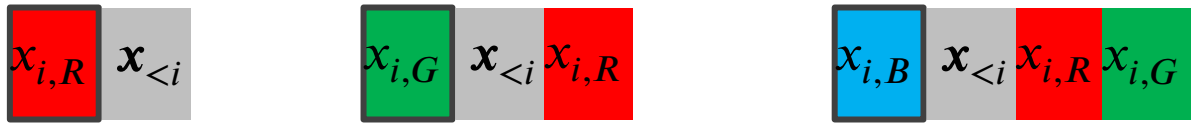
Modern autoregressive models

Figure 1. Image completions sampled from a PixelRNN.

PixelRNN

- Decompose the data likelihood of an $n \times n$ image $p(\boldsymbol{x}) = \prod_{i=1}^{n^2} p(x_i \mid \boldsymbol{x}_{<i})$
- Each pixel conditional corresponds to a triplet of colors
  → Further decompose per color (same as above)

$$p(x_i \mid \boldsymbol{x}_{<i}) = p(x_{i,R} \mid \boldsymbol{x}_{<i}) \cdot p(x_{i,G} \mid \boldsymbol{x}_{<i}, x_{i,R}) \cdot p(x_{i,B} \mid \boldsymbol{x}_{<i}, x_{i,R}, x_{i,G})$$

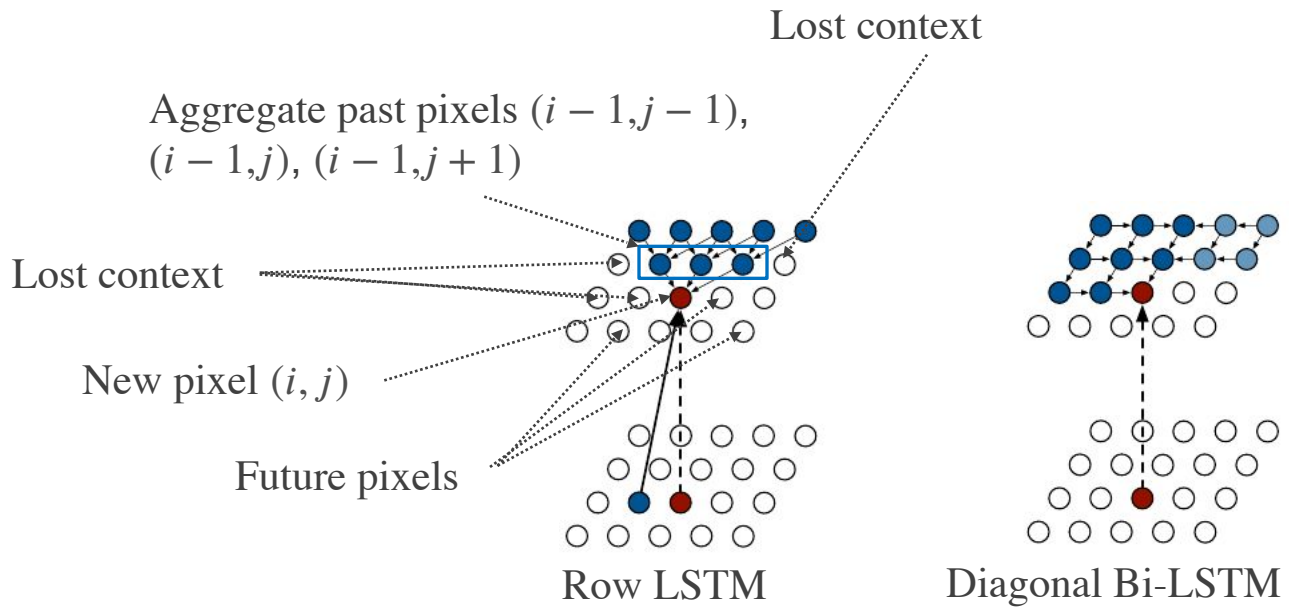


- Model the conditionals $p(x_{i,R} \mid x_{<i}), \ldots$ with 12-layer convolutional RNN
  - The MLP from NADE cannot easily scale and statistics are not shared
- Model the output as a categorical distribution
  - 256-way softmax

van den Oord, Kalchbrenner and Kavukcuoglu, Pixel Recurrent Neural Networks

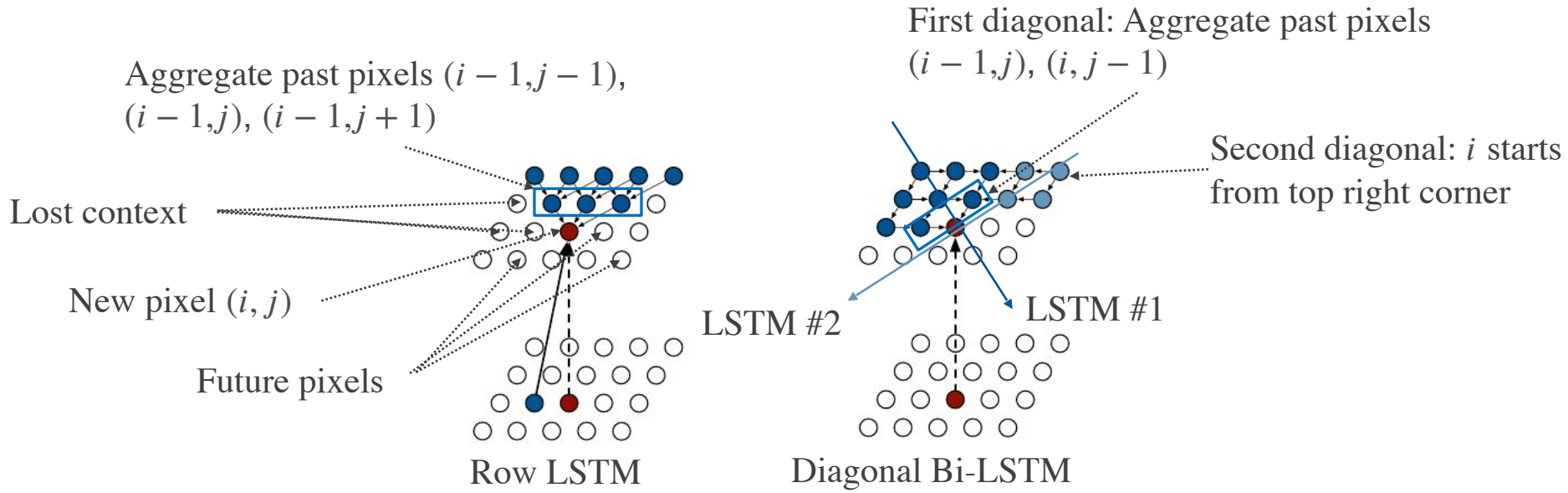
Row LSTM

- Row LSTM with 'causal' triangular receptive field
  - Per new pixel (row i) use 1-d conv (size 3) to aggregate pixels above ($i-1$)
  - The effective receptive field spans a triangle
  - Convolution only on 'past' pixels ($i-1$), not 'future pixels' → causal
  - Loses some context



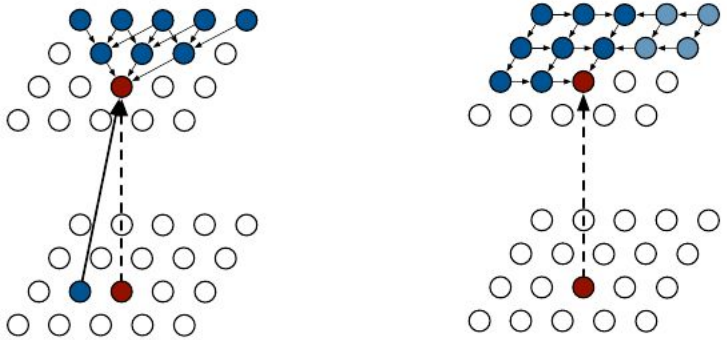
Diagonal BiLSTM

- Have two LSTMs moving on oppose diagonals
  - First diagonal: the convolution past is $(i-1, j), (i, j-1)$
- Combine the two LSTMs
  - recursively the entirety of past context is captured



Why not a regular LSTM?

- It would require sequential, pixel-wise computations
  - Less parallelization
  - Slower training
- With Row LSTM and Diagonal BiLSTM we process one row at a time
  - Parallelization possible

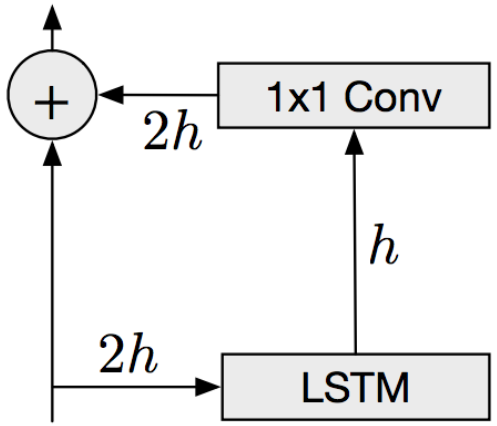


Row LSTM

Diagonal Bi-LSTM

Deep LSTMs with Residual connections

- Use 12 layers of LSTMs
- Add residual connections to speed up learning
- Although good modelling of $p(x) \rightarrow$ nice image generation
- Slow training because of LSTM, slow generation

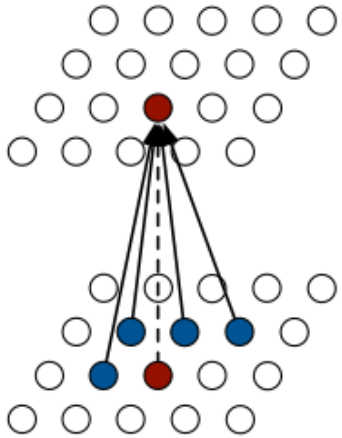


PixelRNN - Generations

Figure 1. Image completions sampled from a PixelRNN.

PixelCNN

- Replace LSTMs with fully convolutional networks
  - 15 layers
  - No pooling layers to preserve spatial resolution
- Use masks to mask out future pixels in convolutions
  - Otherwise ‘access to future’ → no ‘autoregressiveness’
- Faster training as no recurrent steps required
  → Better parallelization
  - Pixel generation still sequential and thus slow



PixelCNN

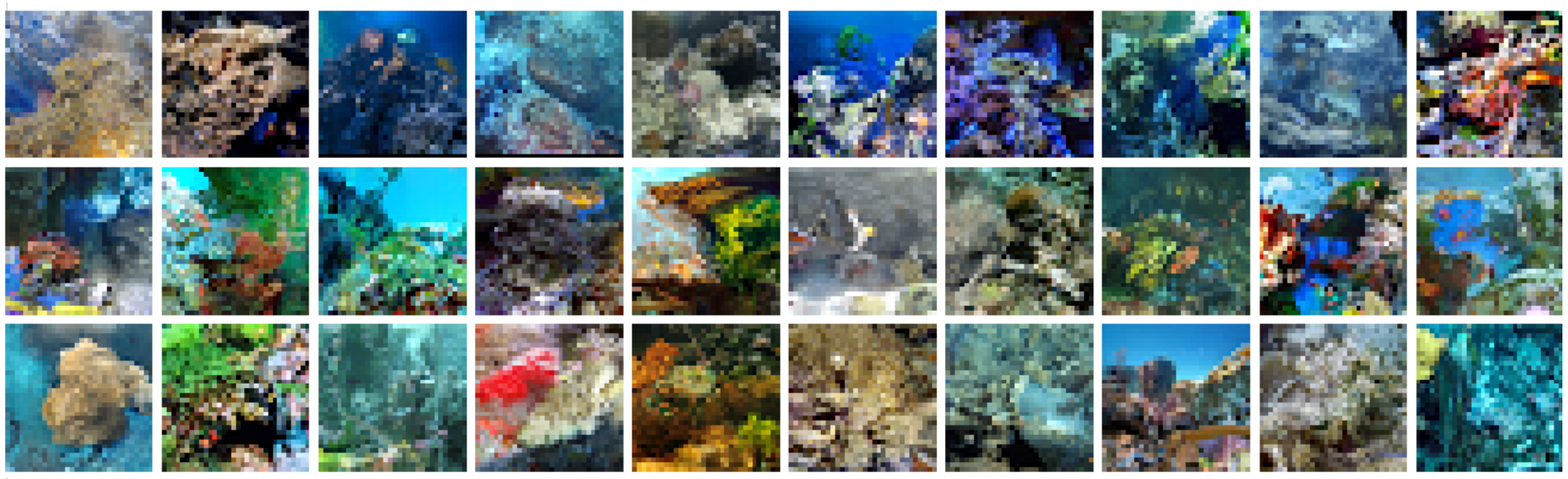
1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

Masking convolutions

van den Oord, Kalchbrenner and Kavukcuoglu, Pixel Recurrent Neural Networks

PixelCNN - Generations

Coral reef

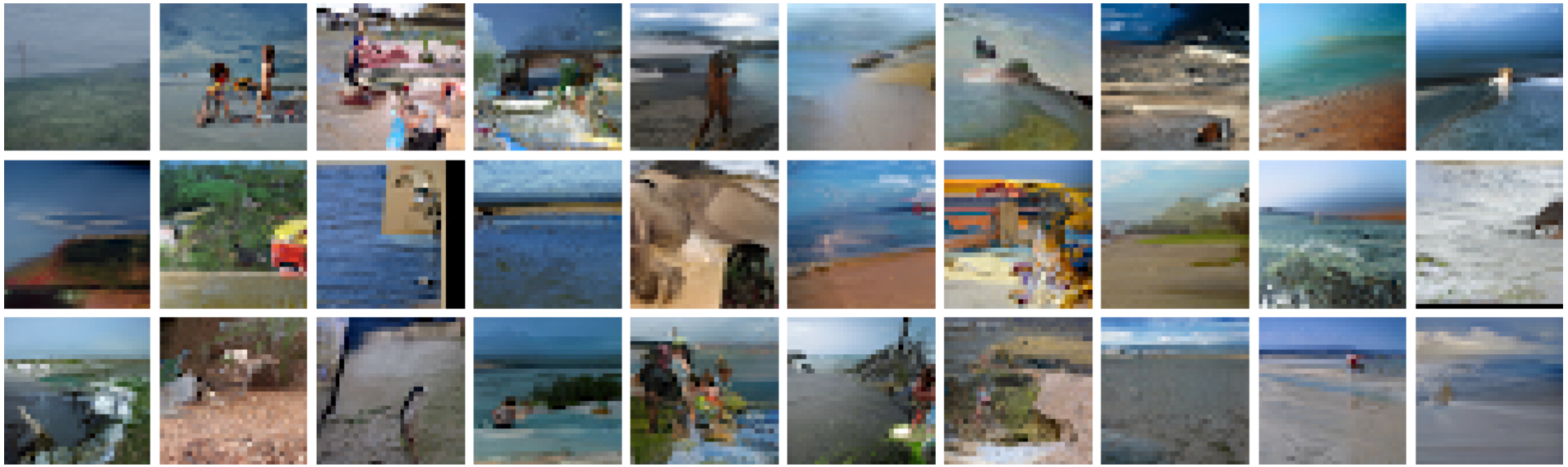


PixelCNN - Generation

Sorrel horse

PixelCNN - Generation

Sandbar

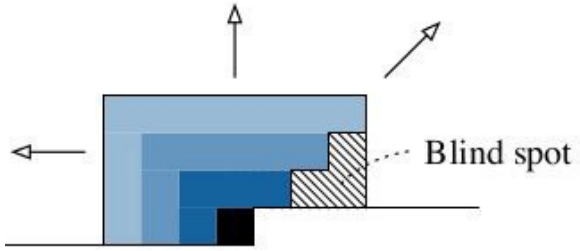


PixelCNN - Generation

Lhasa Apso

PixelCNN: Pros and Cons

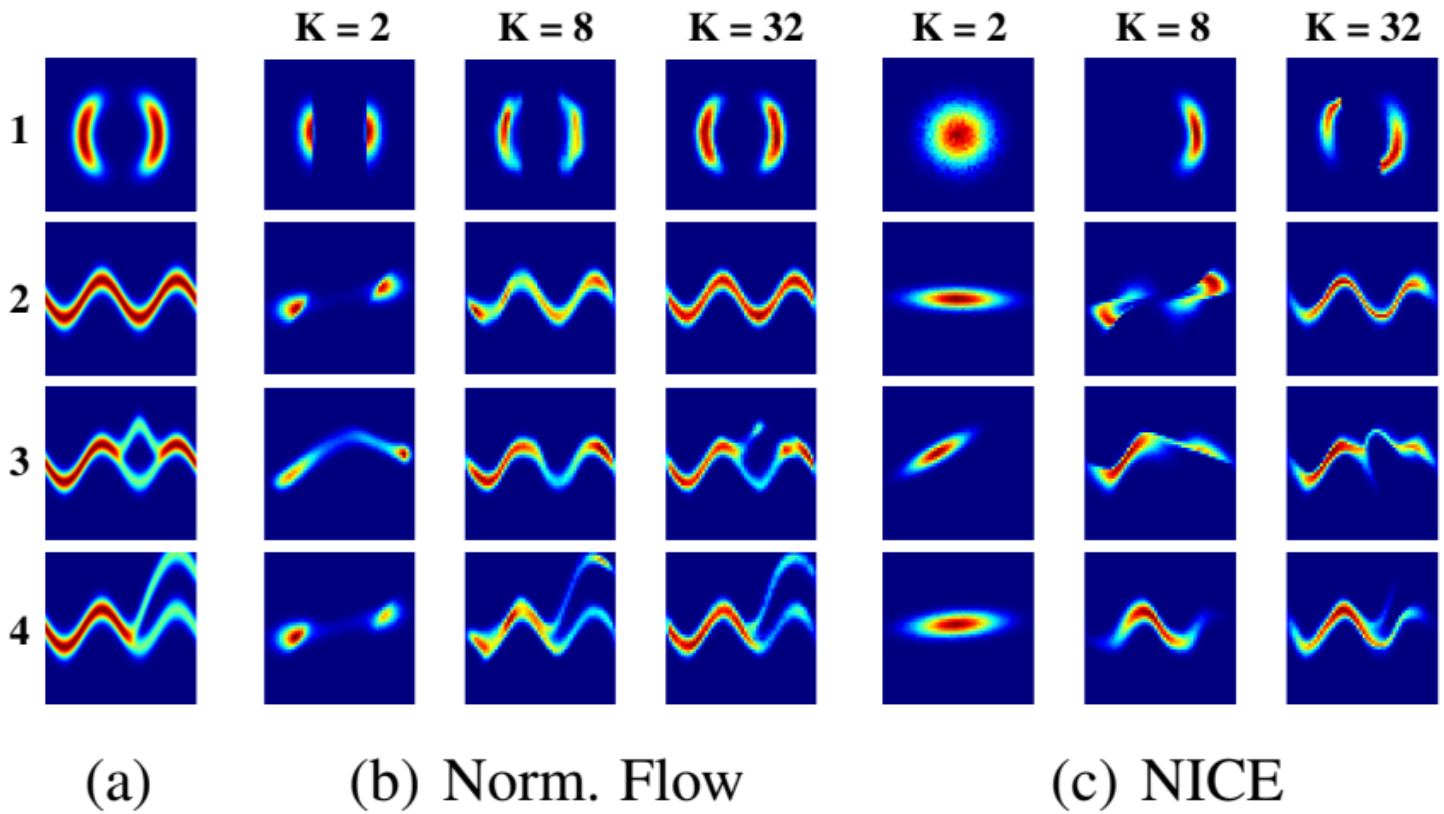
- Faster training
- Performance is worse than PixelRNN as context is discarded
- The cascaded convolutions create a ‘blind spot’
  - Use Gated PixelCNN to fix
- No latest space
- PixelCNN++ improves PixelCNN by (Salimans et al.)
  - Model output by discretized logistic mixture likelihood ← Softmax requires to many parameters and yields very sparse gradients
  - Condition on whole pixels, not colors
  - Architectural innovations



Autoregressive models: pros and cons

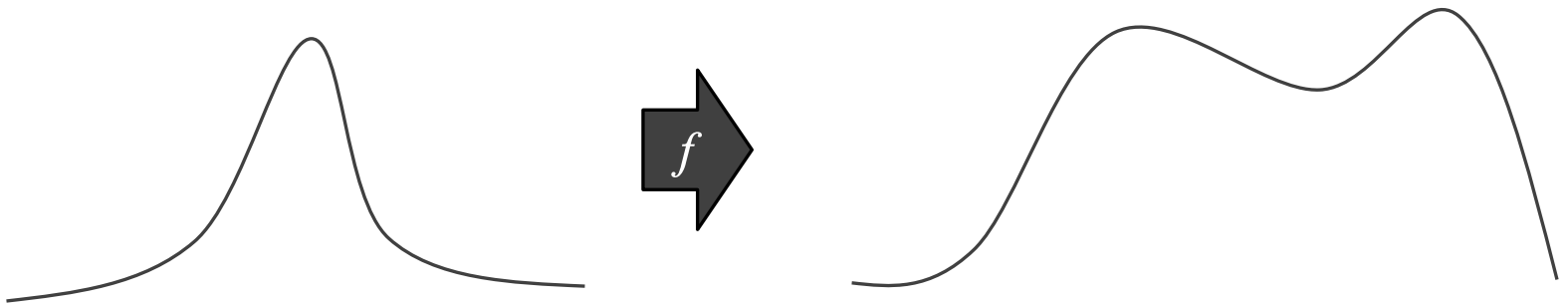
- Top density estimation
- They take into account complex co-dependencies
  - Potentially, better generations and more accurate likelihoods
- Autoregressive models are not necessarily latent variable models
  - They neither have necessarily an encoder nor learn representations
- Slow in learning, inference and generation
  - Computations are sequential (one at a time) → limited parallelism
  - E.g., to generate the next word we must generate past words first
- They may introduce artificial bias when assumed order is imposed

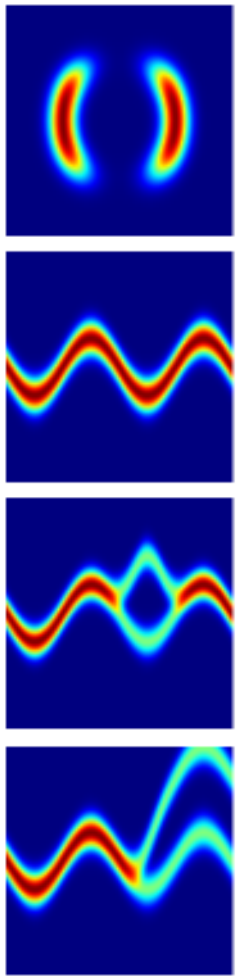
Normalizing flows



Intuition

- Often our posterior approximation is not enough
  - Imagine data generated by two modes
    → approximating with a standard Gaussian would be problematic
- If we applying a transformation to a simple density input
  - *e.g.*, a Gaussian
  - We can morph it into a more complicated density function
- Doing it many times → model any complex density





Change of variables

- For 1-d variables we know that

$$\int f\big(g(x)\big)g'(x)dx = \int f(u)du, \text{where } u = g(x)$$

  - This is called change of variables (or integration by substitution)
  - The density is $du = g'(x)dx$

- For multivariate cases

$$\int f\big(g(\boldsymbol{x})\big)\left|\det\frac{dg}{d\boldsymbol{x}}\right|d\boldsymbol{x} = \int f(\boldsymbol{u})d\boldsymbol{u}, \text{where } \boldsymbol{u} = g(\boldsymbol{x})$$

And $\boldsymbol{u}$ and $\boldsymbol{x}$ have the same dimensionality

Normalizing flows

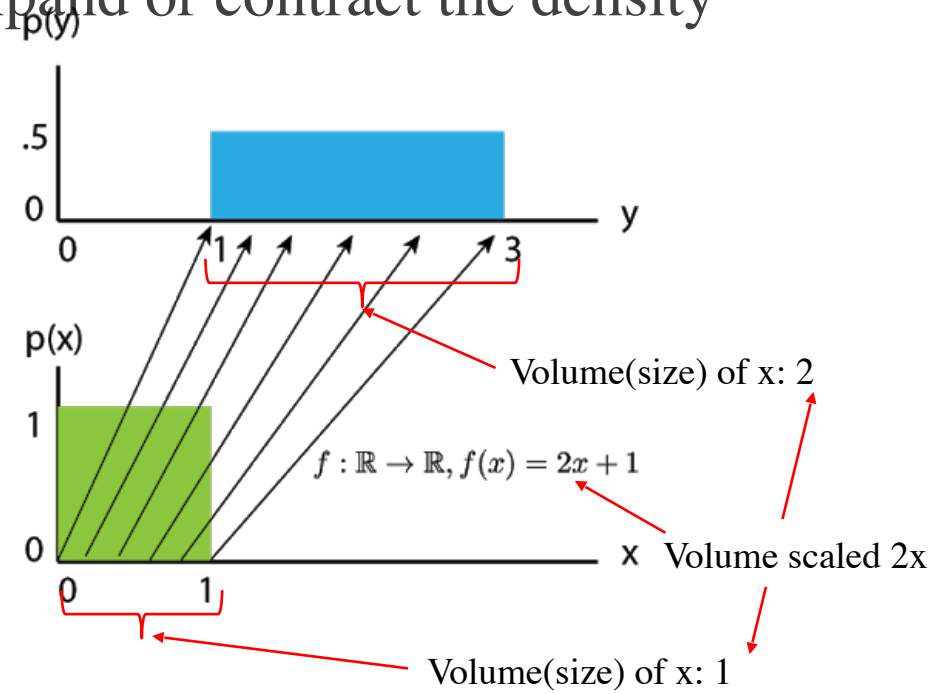
- Our model is an encoder $f: \boldsymbol{x} \rightarrow \boldsymbol{z}$
  - It maps the input $\boldsymbol{x}$ with density $p(\boldsymbol{x})$ to the latent $\boldsymbol{z}$ with density $p(\boldsymbol{z})$
- The inverse model is the decoder $f^{-1}: \boldsymbol{z} \rightarrow \boldsymbol{x}$
  - It maps the latent $\boldsymbol{z}$ back to the input $\boldsymbol{x}$
- For our forward model we have

$$p(\boldsymbol{z}) \left| \det \frac{df}{d\boldsymbol{x}} \right| = p(\boldsymbol{x}) \Leftrightarrow \log p(\boldsymbol{z}) + \log \left| \det \frac{df}{d\boldsymbol{x}} \right| = \log p(\boldsymbol{x})$$

Rezende and Mohamed, Variational Inference with Normalizing Flows

Normalizing flows geometrically

- The determinant shows how the volume of the input and output probability spaces changes
  - The volumes (sizes) must change so that $\int p(\boldsymbol{z}) = 1$ and $\int p(\boldsymbol{x}) = 1$
  - Normalizing flows expand or contract the density



Stacking normalizing flows

- The change of variables can be applied recursively

$$\boldsymbol{x} = \boldsymbol{z}_K = f_K \circ \ldots \circ f_2 \circ f_1(\boldsymbol{z}_0)$$

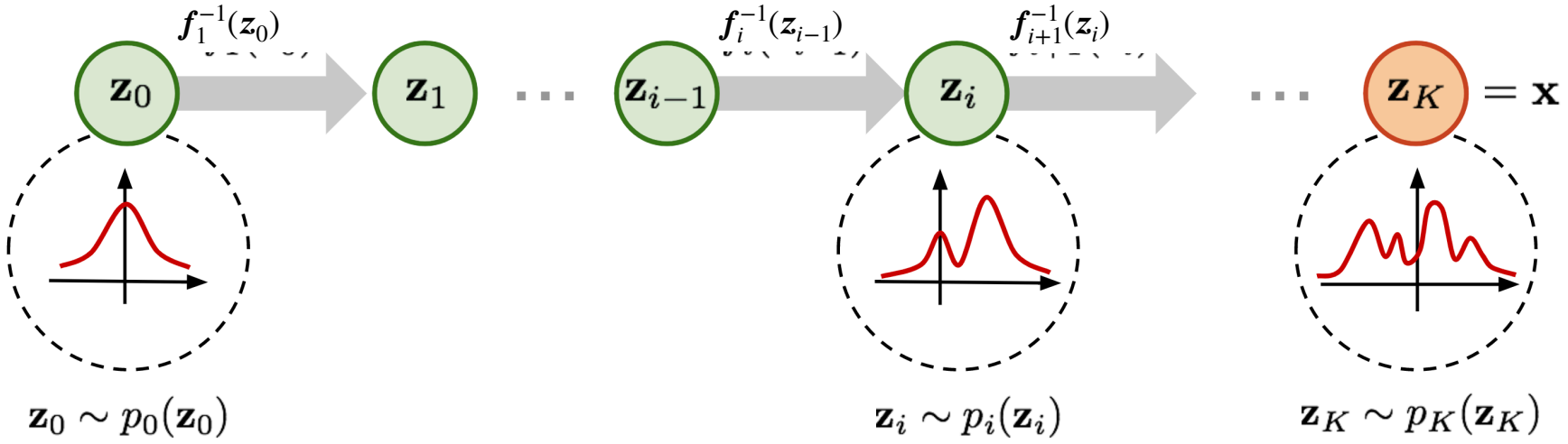
$(e.g., \; \boldsymbol{z}_0 = f_1(\boldsymbol{z}_1) \Leftrightarrow \boldsymbol{z}_1 = f_1^{-1}(\boldsymbol{z}_0))$

- The log density of our data is

$$\log p(\boldsymbol{x}) = \log p_K(\boldsymbol{z}_K) = \log p_0(\boldsymbol{z}_0) - \sum_{k=1}^{K} \log\det \left| \frac{\partial f_k}{\partial \boldsymbol{z}_k} \right|$$

- Optimize with maximum likelihood

Stacking normalizing flows



What transformations?

$$\log p_K(\boldsymbol{z}_K) = \log p_0(\boldsymbol{z}_0) - \sum_{k=1}^{K} \log\det \left| \frac{\partial f_k}{\partial \boldsymbol{z}_k} \right|$$

- We want smooth, differentiable transformations f_k
  - For which it with easy to compute inverse f_k^{-1}
  - and determinant of the Jacobian $\det \frac{df_k}{d\boldsymbol{z}_k}$
- Example transformations
  - Planar flows
  - Radial flows
  - Coupling layers

Planar flow

- The transformation is

$$f(\boldsymbol{z}) = \boldsymbol{z} + \boldsymbol{u}h(\boldsymbol{w}^T\boldsymbol{z} + b)$$

  - $\boldsymbol{u}, \boldsymbol{w}, b$ are free parameters
  - h is an element-wise non-linearity (element-wise so that it is easy to invert)
  - The log-determinant of the Jacobian is

$$\psi(\boldsymbol{z}) = h'\left(\boldsymbol{w}^T\boldsymbol{z} + b\right)\boldsymbol{w}$$

$$\det\left|\frac{\partial f}{\partial \boldsymbol{z}}\right| = |1 + u^T\psi(\boldsymbol{z})|$$

Radial flow

- The transformation is

$$f(z) = z + \beta h(\alpha, r)(z - z_0)$$

Where $h(\alpha, r) = 1/(\alpha + r)$

- The log-determinant of the Jacobian is

$$\det \left| \frac{\partial f}{\partial z} \right| = \left[1 + \beta h(\alpha, r)\right]^{d-1} \left[1 + \beta h(\alpha, r) + h'(\alpha, r) r\right]$$

VAE with Normalizing Flows

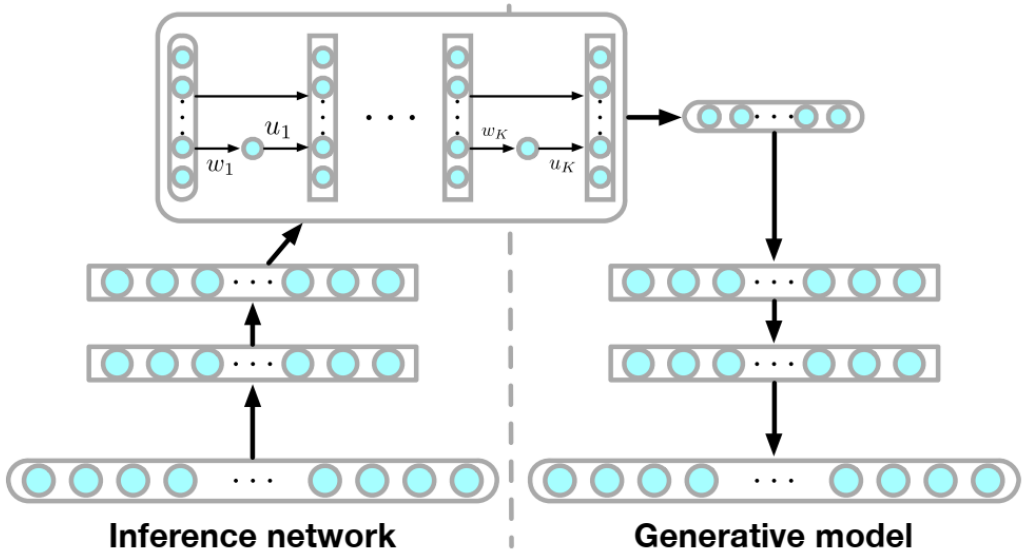


Figure 2. Inference and generative models. Left: Inference network maps the observations to the parameters of the flow; Right: generative model which receives the posterior samples from the inference network during training time. Round containers represent layers of stochastic variables whereas square containers represent deterministic layers.

Learning better posteriors with variational inference

- Again: the evidence lower bound is

$$\text{ELBO}_{\theta,\varphi}(\boldsymbol{x}) = \log p(\boldsymbol{x}) - \text{KL}(q_{\varphi}(\boldsymbol{z}\,|\,\boldsymbol{x})\,||\,p(\boldsymbol{z}\,|\,\boldsymbol{x}))$$

- Replace the simple approximate posterior by normalizing flows

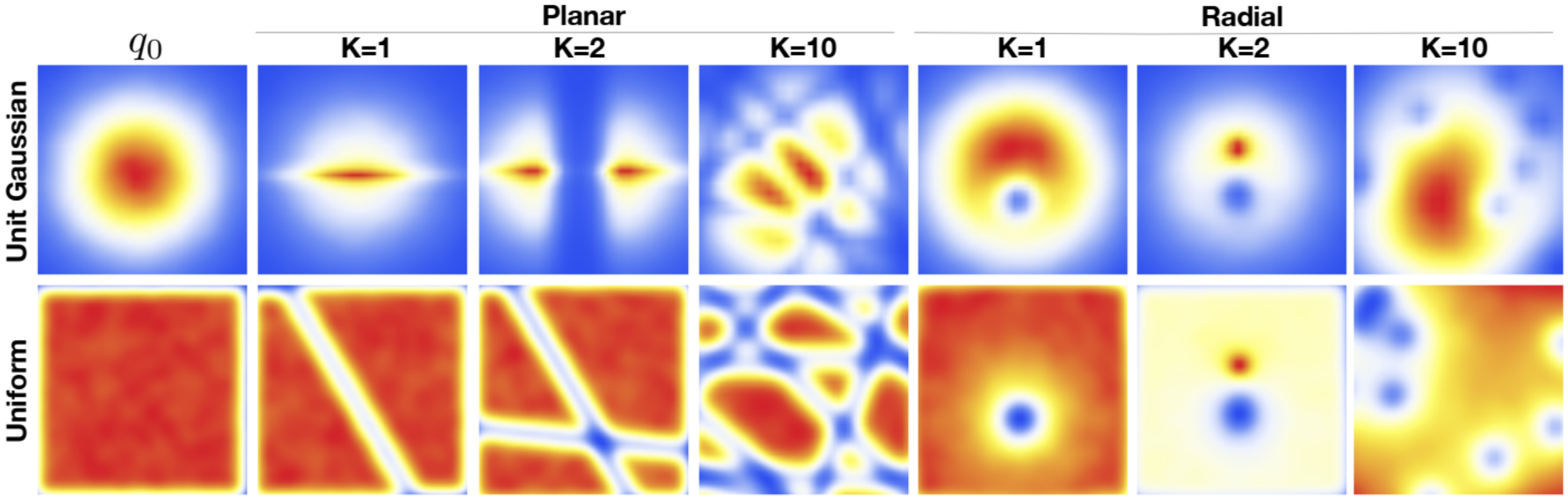
$$\mathbb{E}_{q_0(\boldsymbol{z}_0|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}\,|\,\boldsymbol{z}_K)\right] - \text{KL}(q_0(\boldsymbol{z}_0\,|\,\boldsymbol{x})\parallel p(\boldsymbol{z})) + \mathbb{E}_{q_0(\boldsymbol{z}_0\,|\,\boldsymbol{x})}\left[\sum_{k=1}^{K}\log\left|\det\frac{df_k}{d\boldsymbol{z}_k}\right|^{-1}\right]$$

Algorithm 1 Variational Inf. with Normalizing Flows

Parameters: ϕ variational, $\boldsymbol{\theta}$ generative
while not converged **do**
  $\mathbf{x} \leftarrow$ {Get mini-batch}
  $\mathbf{z}_0 \sim q_0(\bullet|\mathbf{x})$
  $\mathbf{z}_K \leftarrow f_K \circ f_{K-1} \circ \ldots \circ f_1(\mathbf{z}_0)$
  $\mathcal{F}(\mathbf{x}) \approx \mathcal{F}(\mathbf{x}, \mathbf{z}_K)$
  $\Delta\boldsymbol{\theta} \propto -\nabla_{\theta}\mathcal{F}(\mathbf{x})$
  $\Delta\boldsymbol{\phi} \propto -\nabla_{\phi}\mathcal{F}(\mathbf{x})$
end while

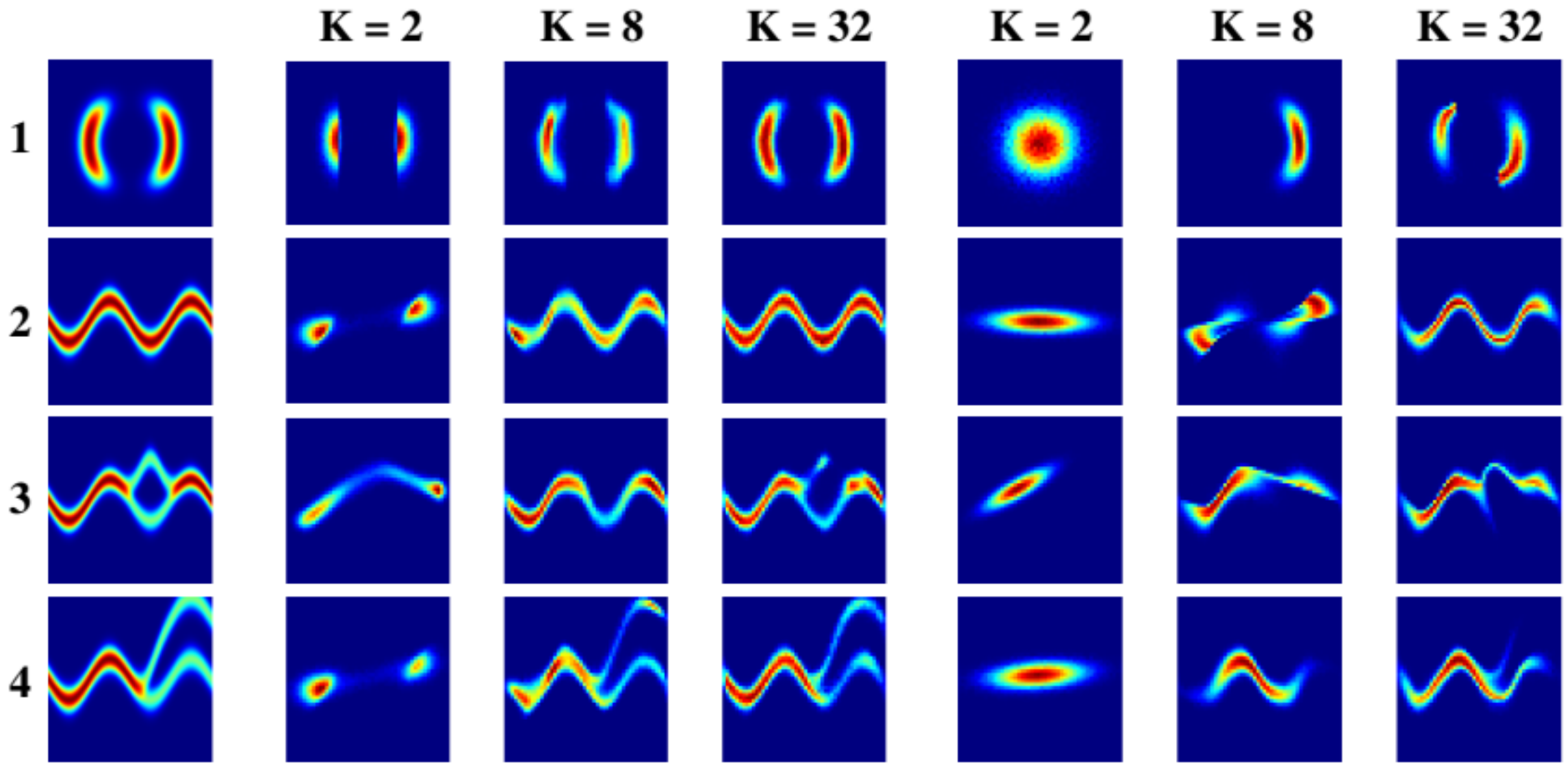
Rezende and Mohamed, Variational Inference with Normalizing Flows

The effect of number of transformations/flows



Rezende and Mohamed, Variational Inference with Normalizing Flows

Some results



Rezende and Mohamed, Variational Inference with Normalizing Flows

Flow-based models

Figure 4: Random samples from the model, with temperature 0.7

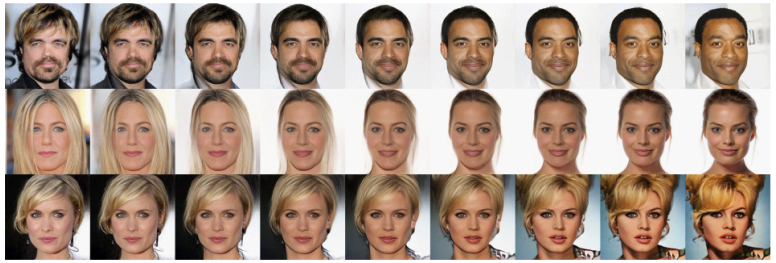
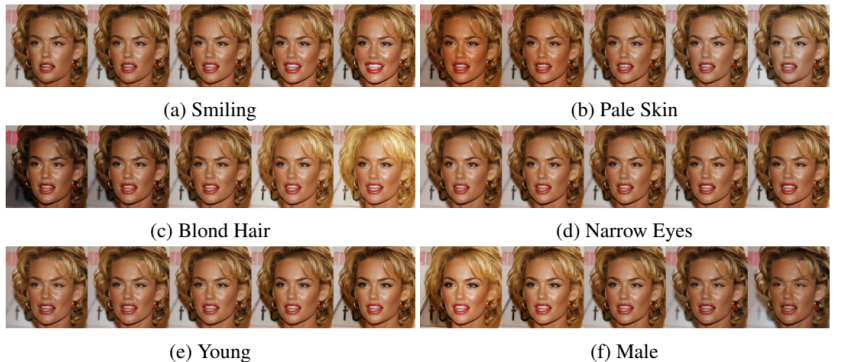
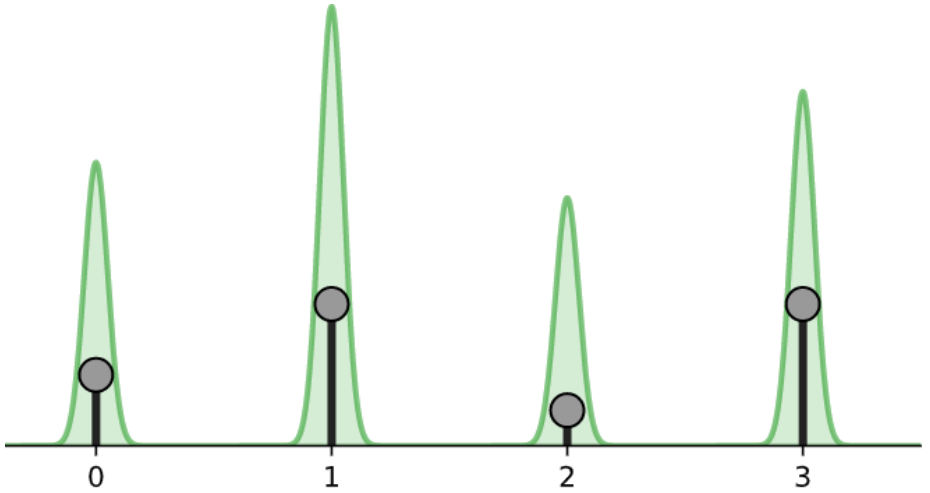


Figure 5: Linear interpolation in latent space between real images



Normalizing flows on images

- Normalizing flows are continuous transformations
- Images contain discrete values
  → The model will assign δ-peak probabilities on integer (pixel) values only
  - These probabilities will be nonsensical, there is no smoothness



UVADLC tutorial

(Variational) dequantization

- Add (continuous) noise $u \sim q(u \mid x)$ to input variables $v = x + u$
- The data log-likelihood then is

$$\log p(x) = \log \int p(x+u)\, du = \log \mathbb{E}_{u \sim q(u \mid x)} \left[\frac{p(x+u)}{q(u \mid x)} \right] \geq \mathbb{E}_{u \sim q(u \mid x)} \log \left[\frac{p(x+u)}{q(u \mid x)} \right]$$

- If $q(u \mid x)$ is the uniform distribution the standard dequantization
  - Probability between two consecutive values is fixed
    → resemble boxy boundaries between values
- Better learn $q(u \mid x)$ in a variational manner
  → Variational dequantization

Coupling layers

- Given input z the output of the transformation is

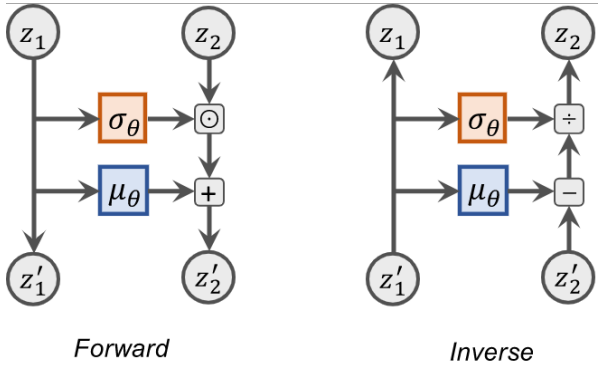
$$z' = \begin{bmatrix} z'_{1:j} \\ z'_{j+1:d} \end{bmatrix} = \begin{bmatrix} z_{1:j} \\ \mu_\theta\left(z_{1:j}\right) + \sigma_\theta\left(z_{1:j}\right) \odot z_{j+1:d} \end{bmatrix}$$

  - $\mu_\theta,\ \sigma_\theta$ are neural networks with shared parameters

- Easy inverse: $z = \begin{bmatrix} z_{1:j} \\ \frac{\left(z'_{j+1:d} - \mu_\theta\left(z_{1:j}\right)\right)}{\sigma_\theta\left(z_{1:j}\right)} \end{bmatrix}$

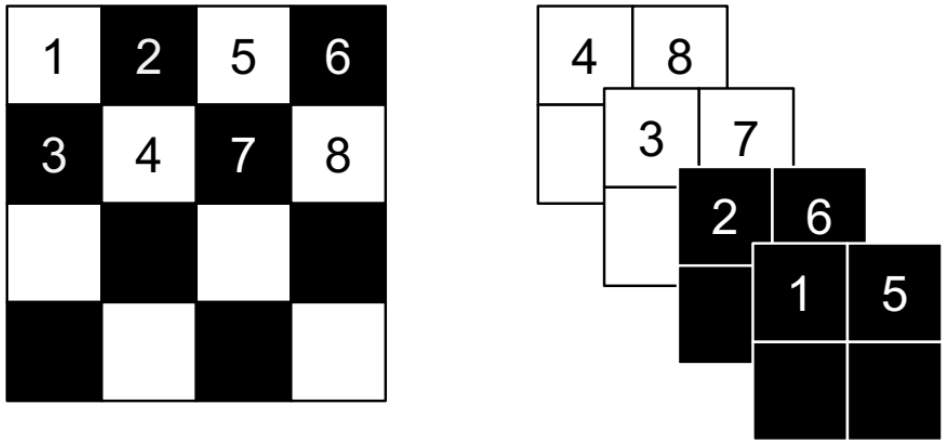
- Easy triangular Jacobian $\frac{\partial z'}{\partial z} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial z'_{j+1:d}}{\partial z_{1:j}} & \text{diag}\left(\sigma_\theta\left(z_{1:j}\right)\right) \end{bmatrix}$

  - The log determinant is $\sum_j \log \sigma_\theta\left(z_j\right)$



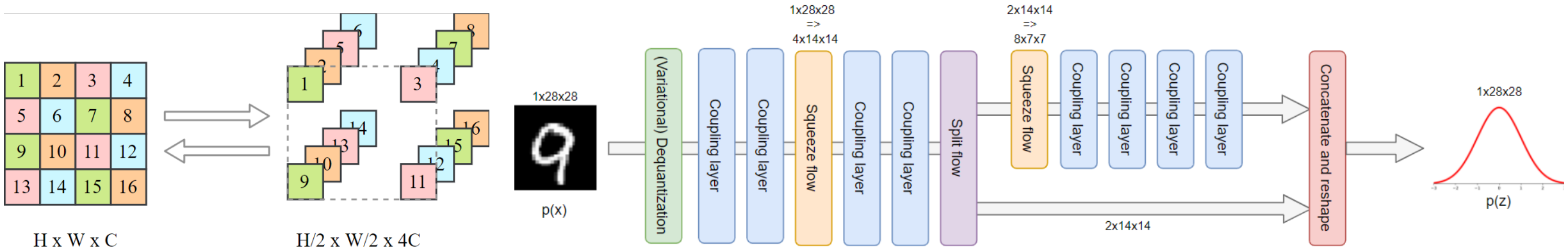
Splitting dimensions in images

- Use masking
  - Checkers pattern
  - Splitting across channels
- Alternate dimensions between consecutive layers
  → not always the same $1:d$ dimensions remain untouched



Multi-scale architecture

- Invertibility → number of dimensions before and after f is the same
  - High computational complexity for large images
- Apply new transformations to half the input only
  - For the other half use the prior (previous) trasnfromations
- Use squeeze to turn spatial to channel dimensions
  - And split for halving the input



UVADLC tutorial

GLOW, FLOW, FLOW++

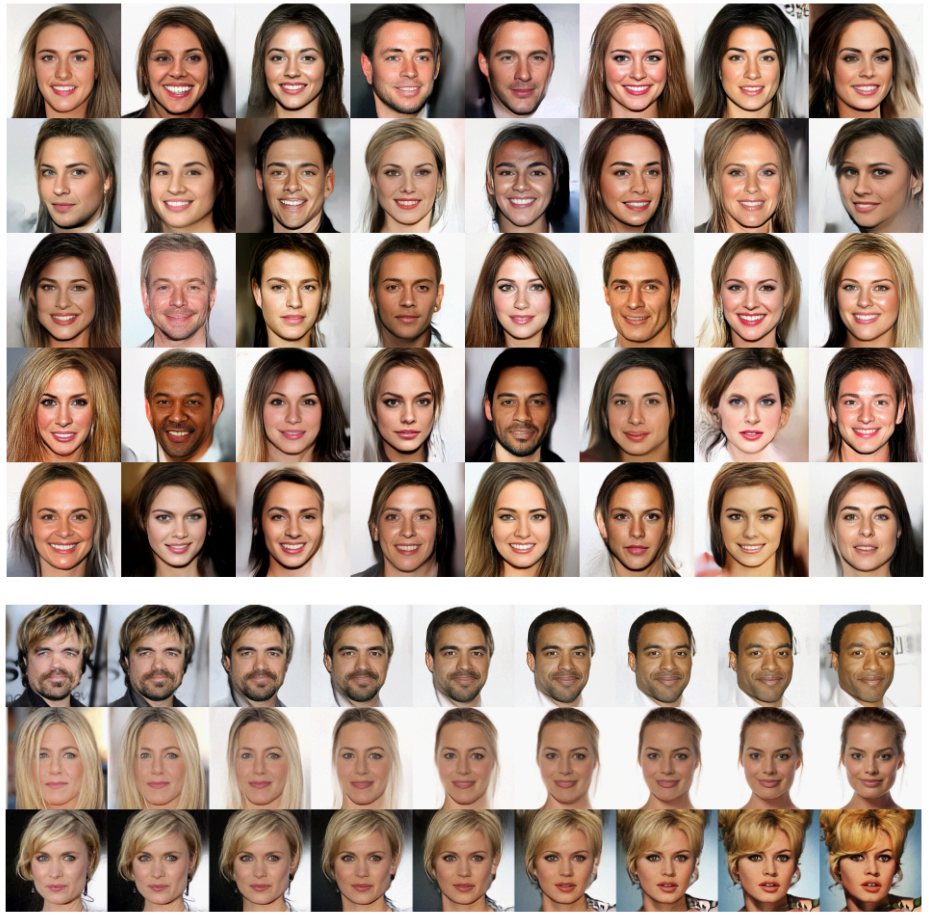


Figure 5: Linear interpolation in latent space between real images

Kingma, Dhariwal, Glow: Generative Flow with Invertible 1x1 Convolutions

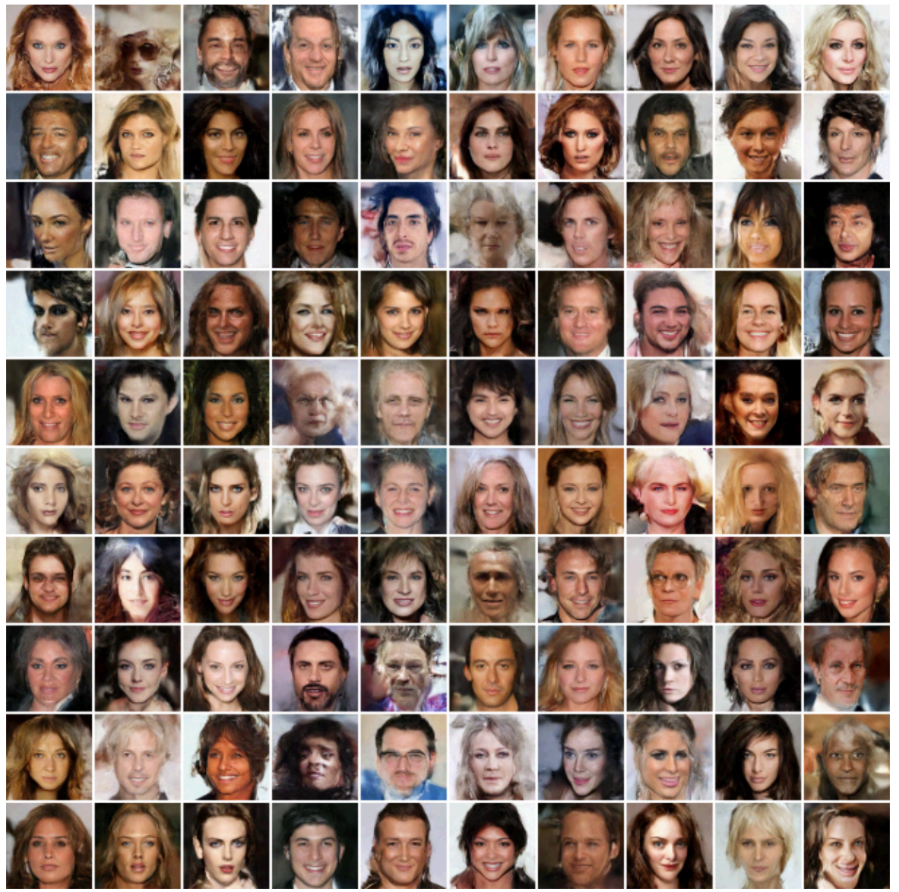


Figure 4. Samples from Flow++ trained on 5-bit 64x64 CelebA, without low-temperature sampling.

Kingma, Dhariwal, Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design

Categorical normalizing flows

- Normalizing flows with variational inference to learn representations of categorical data on continuous space
  - Learnable, smooth, support for higher dimensions
- Learning must ensure no loss of information

  → the volumes that represent categorical data must not-overlap
  - Otherwise, to which category does the representation correspond to?

$$p(\boldsymbol{x}) \geq \mathbb{E}_{\boldsymbol{z}\sim q(\,\cdot\,|\,\boldsymbol{x})}\left[\frac{\prod_i p\left(x_i \,\middle|\, \boldsymbol{z}_i\right)}{q(\boldsymbol{z}\,|\,\boldsymbol{x})} p(\boldsymbol{z})\right]$$

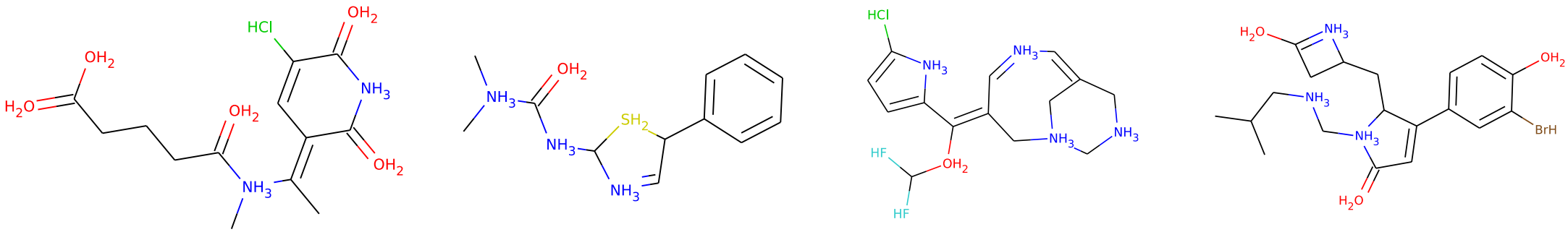
  - Factorized posterior $\prod_i p\left(x_i \,\middle|\, \boldsymbol{z}_i\right)$ to encourage learning non-overlapping $\boldsymbol{z}_i$

Lippe and Gavves, Categorical Normalizing Flows via Continuous Transformations, in submission to ICLR 2021

Graph generation with categorical normalizing flows

Results on the Zinc250k dataset (224k examples)

Method	Validity	Uniqueness	Novelty	Reconstruction	Parallel	General
JT-VAE	100%	100%	100%	71%	✗	✗
GraphAF	68%	99.10%	100%	100%	✗	✓
R-VAE	34.9%	100%	–	54.7%	✓	✓
GraphNVP	42.60%	94.80%	100%	100%	✓	✓
GraphCNF	83.41% (±2.88)	99.99% (±0.01)	100% (±0.00)	100% (±0.00)	✓	✓
+ Sub-graphs	96.35% (±2.21)	99.98% (±0.01)	99.98% (±0.02)	100% (±0.00)	✓	✓



Normalizing flows: pros and cons

- Starting from a simple density like a unit Gaussian we can obtain any complex density that match our data without even knowing its analytic form
- Tractable density estimation
- Efficient parallel sampling and learning
- Often very many transformations required → Very large networks needed
- Constrained to invertible transformations with tractable determinant
- Tied encoder and decoder weights
- Transformations cannot easily introduce bottlenecks

UVADLC tutorial

A summary of properties

	Training	Likelihood	Sampling	Compression
Autoregressive models (e.g., PixelCNN)	Stable	Yes	Slow	No
Flow-based models (e.g., RealNVP)	Stable	Yes	Fast/Slow	No
Implicit models (e.g., GANs)	Unstable	No	Fast	No
Prescribed models (e.g., VAEs)	Stable	Approximate	Fast	Yes

17

J. Tomczak's lecture from April, 2019

Summary

- Early autoregressive models
- Modern autoregressive models
- Normalizing flows
- Flow-based models