Introduction to Deep Generative Modeling

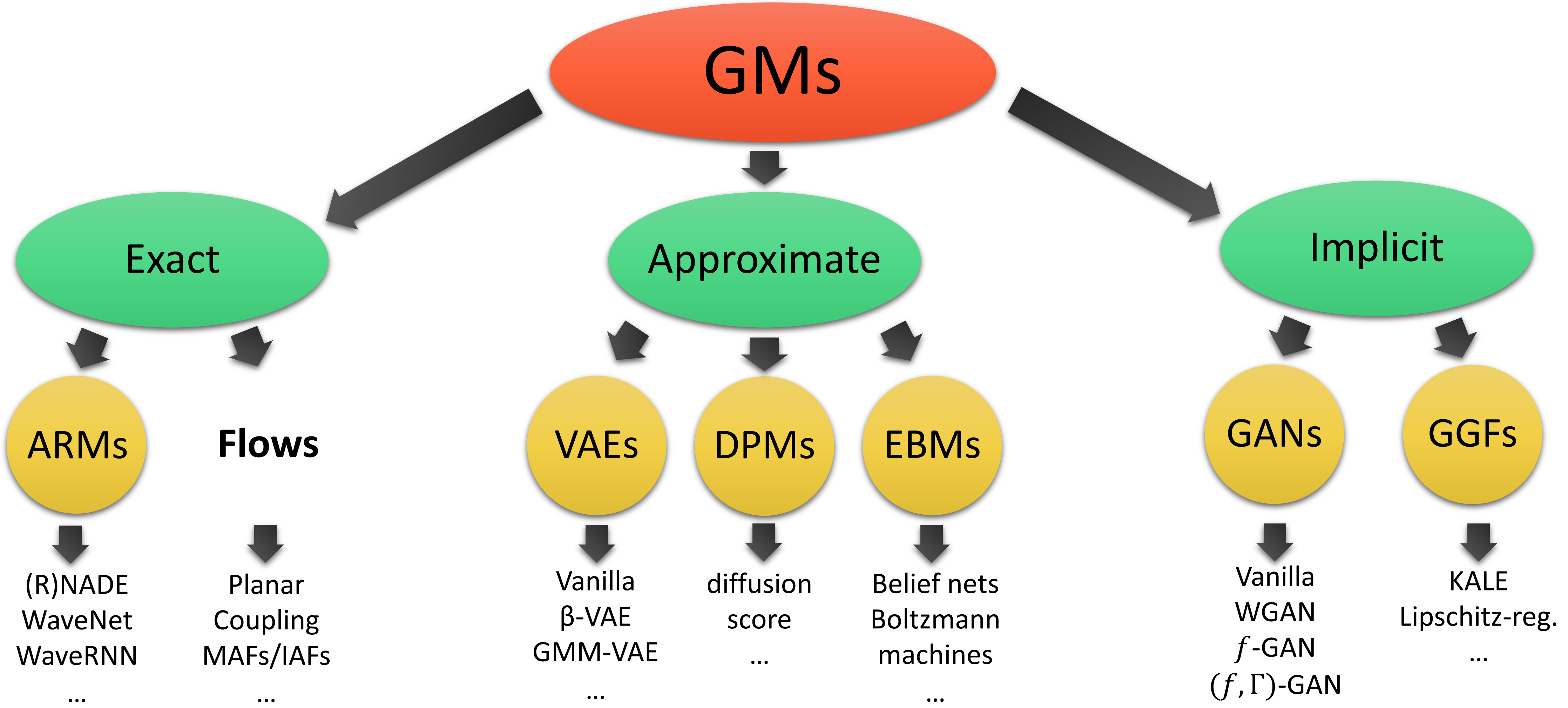
Lecture #8

HY-673 – Computer Science Dep., University of Crete

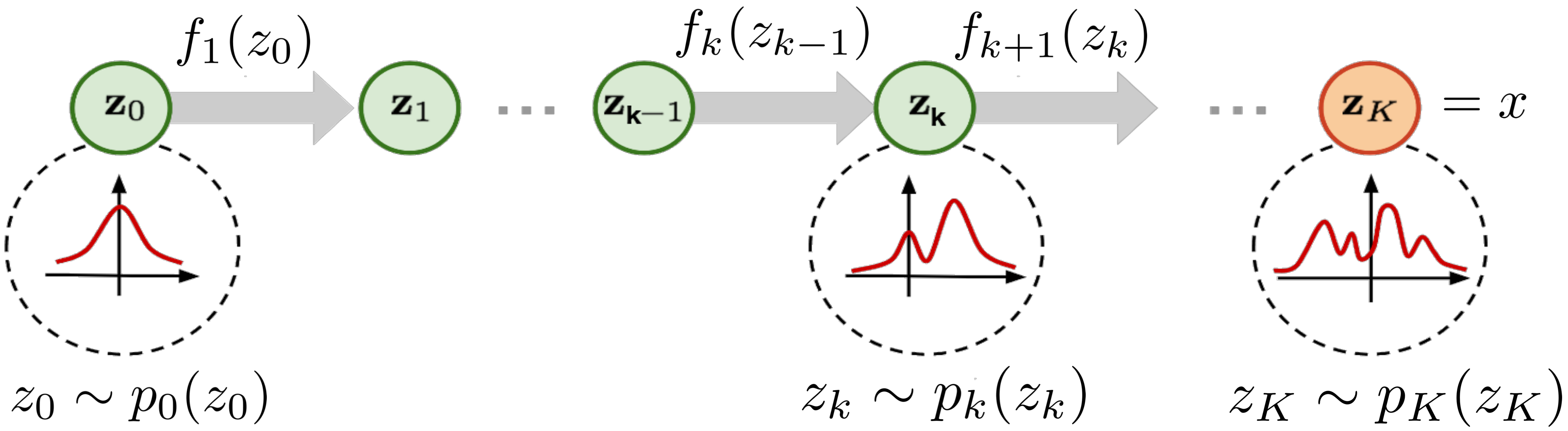
Professors: Yannis Pantazis, Yannis Stylianou

Teaching Assistant: Michail Raptakis

Taxonomy of GMs



Recap: Normalizing Flow Models



1. Transform simple to complex distributions via sequence of invertible and differentiable transformations.
2. Directed latent variable models with marginal likelihood given by the change of variables formula.
3. Triangular Jacobian permits efficient evaluation of log-likelihoods.

Normalizing Flow Models Recap

- Functionality of Normalizing Flows:
  - Sampling via: $x = f_\theta(z), \quad z \sim p_Z(z).$
  - Density evaluation via: $p_\theta(x) = p_Z(f_\theta^{-1}(x))|\det J_{f_\theta^{-1}}(x)|,$
  - or via: $p_\theta(x) = p_Z(z)|\det J_{f_\theta}(u)|^{-1}$ where $z = f_\theta^{-1}(x).$
- Training with MLE requires:
  - Compute $f_\theta^{-1}(x)$.
  - Compute $f_\theta^{-1}(x)$'s Jacobian determinant with $O(d)$ cost.
  - Differentiate the above w.r.t. θ.
  - Compute base density $p_Z(z)$.

Caution: Being invertible and being able to explicitly calculate the inverse are **not synonymous**!

Training Normalizing Flow Models

- *Recall:* Planar flows didn't have easy-to-calculate inverse transformation.
  - Thus, MLE estimation is not suitable for planar flows.
- *Recall:* MLE is equivalent to Kullback-Leibler divergence minimization (at the infinite number of sample limit).

$$\text{argmin}_\theta D_{KL}(p_d(x)||p_\theta(x)).$$

- *Recall:* What if we minimize the reverse Kullback-Leibler divergence?

$$\text{argmin}_\theta D_{KL}(p_\theta(x)||p_d(x)).$$

Reverse KLD Minimization

$$\underline{\textit{Goal:}}\ \ \text{argmin}_\theta D_{KL}(p_\theta(x)||p_d(x)).$$

- Suitable when we can evaluate $p_d(x)$ (up to a multiplicative factor).
- *Requirements:* (i) sample from base distribution $p_Z(z)$, (ii) compute and (iii) differentiate through the transformation f_θ and its Jacobian determinant.
  - Planar flows satisfy all three requirements!
- *Applications:*
  1. Variational Inference (e.g, compine variational autoencoders with flows),
  2. Model Distilation (e.g., Parallel Wavenet).

Designing Invertible Transformations

- NICE or Nonlinear Independent Components Estimation (Dinh et al., 2014): Composes two kinds of invertible transformations: additive, coupling layers and rescaling layers.
- Real-NVP (Dinh et al., 2017): NICE extension to non-volume preserving transformations.
- Masked Autoregressive Flow (Papamakarios et al., 2017).
- Inverse Autoregressive Flow (Kingma et al., 2016).

} HY-673

- I-resnet (Behrmann et al., 2018).
- Glow (Kingma et al., 2018).
- MintNet (Song et al., 2019).
- And many more.

NICE - Additive Coupling Layers

- Partition the multi-dimensional variable z into two disjoint subsets, say $z_{1:j} := (z_1, ..., z_j)$ and $z_{j+1:d}$ for any $1 \leq j < d$:

1. Forward mapping $z \to x$:

   - $x_{1:j} = z_{1:j}$ (identity transformation).
   - $x_{j+1:d} = z_{j+1:d} + m_\theta(z_{1:j})$, where $m_\theta(\cdot)$ is typically a neural network with parameters θ, input units j, and output units $d - j$.
   - $x = f_\theta(z) = \begin{bmatrix} z_{1:j} \\ z_{j+1:d} + m_\theta(z_{1:j}) \end{bmatrix}.$

NICE - Additive Coupling Layers

- Partition the multi-dimensional variable z into two disjoint subsets, say $z_{1:j} := (z_1, ..., z_j)$ and $z_{j+1:d}$ for any $1 \leq j < d$:

2. Inverse mapping $x \to z$:

    - $z_{1:j} = x_{1:j}$ (identity transformation).

    - $z_{j+1:d} = x_{j+1:d} - m_\theta(x_{1:j})$, where $m_\theta(\cdot)$ is the same neural network.

    - $z = f_\theta^{-1}(x) = \begin{bmatrix} x_{1:j} \\ x_{j+1:d} - m_\theta(x_{1:j}) \end{bmatrix}$.

NICE - Additive Coupling Layers

- Partition the multi-dimensional variable z into two disjoint subsets, say $z_{1:j} := (z_1, ..., z_j)$ and $z_{j+1:d}$ for any $1 \leq j < d$:

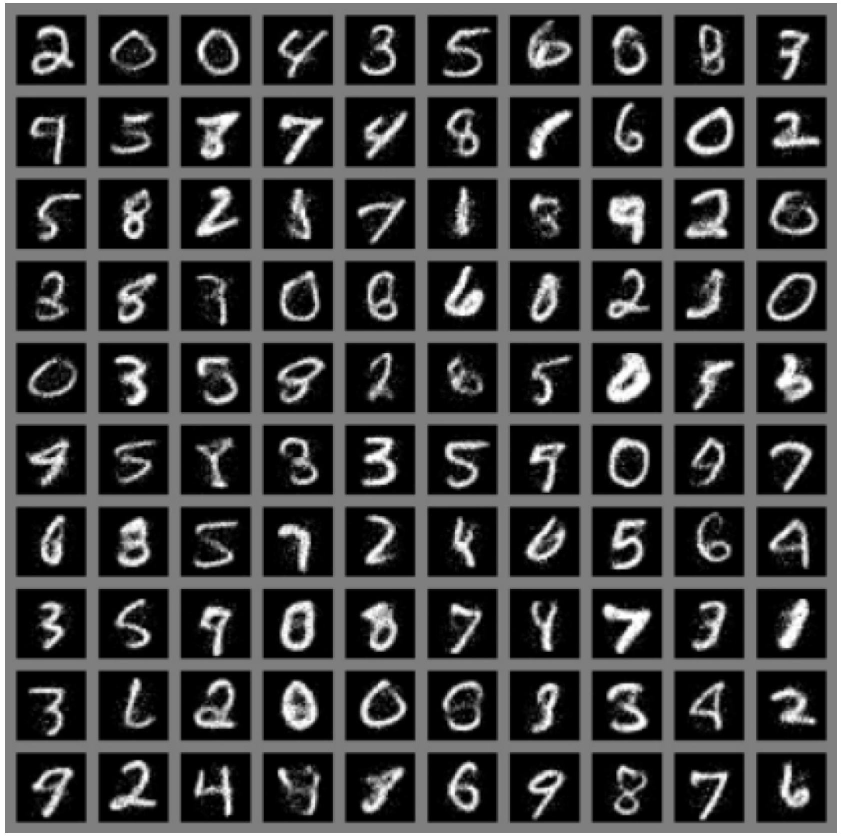
3. Jacobian of forward mapping:

   - $J = \frac{\partial f_\theta}{\partial z} = \begin{pmatrix} \frac{\partial x_{1:j}}{\partial z_{1:j}} & \frac{\partial x_{1:j}}{\partial z_{j+1:d}} \\ \frac{\partial x_{j+1:d}}{\partial z_{1:j}} & \frac{\partial x_{j+1:d}}{\partial z_{j+1:d}} \end{pmatrix} = \begin{pmatrix} I_j & 0 \\ \frac{\partial m_\theta}{\partial z_{1:j}} & I_{d-j} \end{pmatrix}.$

   - $\Rightarrow \det(J) = 1.$

   - **Volume preserving transformation** (since the determinant is 1).

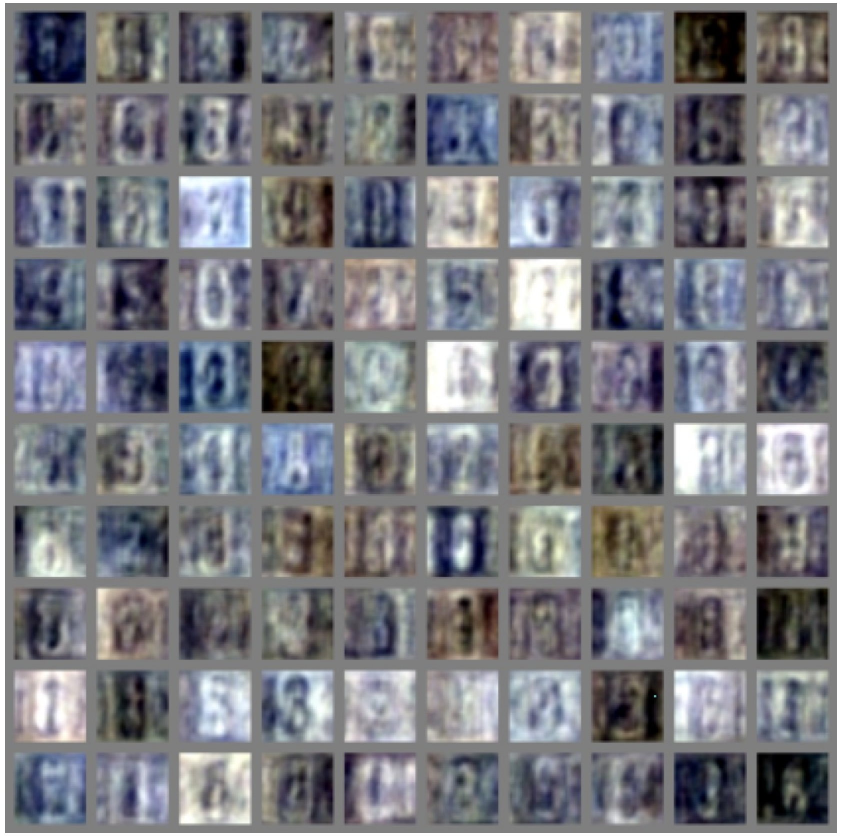
Samples Generated via NICE



(a) Model trained on MNIST

(b) Model trained on TFD

Samples Generated via NICE



(a) Model trained on SVHN

(b) Model trained on CIFAR-10

Real-NVP:
Non-Volume Preserving (NICE Extension)

1. Forward mapping $z \to x$:

- $x_{1:j} = z_{1:j}$ (identity transformation).
- $x_{j+1:d} = z_{j+1:d} \odot \exp(\alpha_\theta(z_{1:j})) + m_\theta(z_{1:j})$.
- Both $m_\theta(\cdot)$ and $\alpha_\theta(\cdot)$ are neural networks with parameters θ, input units j, and output units $d - j$ ($\odot$ denotes elementwise product).

Real-NVP: Non-Volume Preserving (NICE Extension)

2. Inverse mapping $x \to z$:

- $z_{1:j} = x_{1:j}$ (identity transformation).
- $z_{j+1:d} = \big(x_{j+1:d} - m_\theta(x_{1:j})\big) \odot \exp(-\alpha_\theta(x_{1:j}))$.
- Both $m_\theta(\cdot)$ and $\alpha_\theta(\cdot)$ are the same neural networks.

Real-NVP: Non-Volume Preserving (NICE Extension)

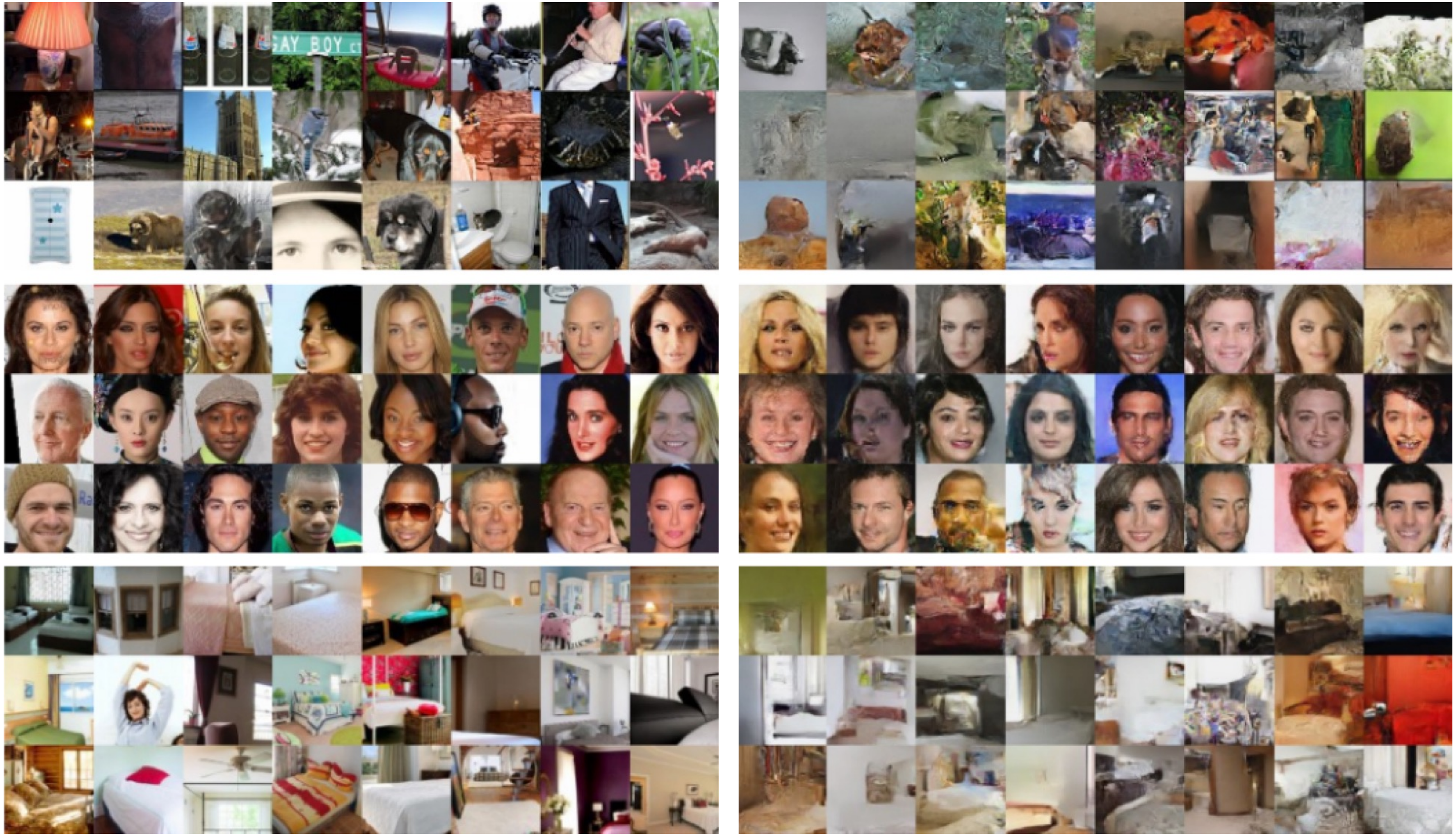
3. Jacobian of forward mapping:

- $J = \frac{\partial f_\theta}{\partial z} = \begin{pmatrix} I_j & 0 \\ \frac{\partial x_{j+1:d}}{\partial z_{1:j}} & \text{diag}(\exp(\alpha_\theta(z_{1:j}))) \end{pmatrix}.$

- $\det(J) = \prod_{i=j+1}^{d} \exp(\alpha_\theta(z_{1:j})) = \exp\left(\sum_{i=j+1}^{d} \alpha_\theta(z_{1:j})\right).$

- **Non-volume preserving transformation** (in general, since the determinant can be less or greater than 1).

Samples Generated via Real-NVP



Latent Space Interpolations via Real-NVP

Using with four validation samples $z^{(1)}, \ldots, z^{(4)}$, define the interpolated sample z as:

$$z = \cos(\phi)\big(z^{(1)} cos(\phi') + z^{(2)} \sin(\phi')\big) + \sin(\phi)\big(z^{(3)} \cos(\phi') + z^{(4)} \sin(\phi')\big),$$

with interpolation parameters ϕ and ϕ'.

Summary: Coupling Layers

- Coupling layers allow both density evaluation and sampling to be fast.
  - One of the most popular flow-based implementations.
- The efficiency of coupling layers comes at the cost of reduced expressive power.
  - Solution: Composing multiple coupling layers with different z elements being transformed each time.
  - Thus, all dimensions have the change to be transformed and be correlated to each other.
- Apart from NICE and RealNVP, Glow, WaveGlow, FloWaveNet and Flow++ are models based on coupling layers.

Continuous AR Models as Flow Models

- Consider a Gaussian autoregressive model:

$$p_d(x) = \prod_{j=1}^{d} p(x_j | x_{<j}),$$

such that $p(x_j|x_{<j}) = \mathcal{N}\big(\mu_j(x_1, \ldots, x_{j-1}), \exp(\alpha_j(x_1, \ldots, x_{j-1}))^2\big).$

- Here, $\mu_j(\cdot)$ and $\alpha_j(\cdot)$ are neural networks for $j > 1$ and constants for $j = 1$.

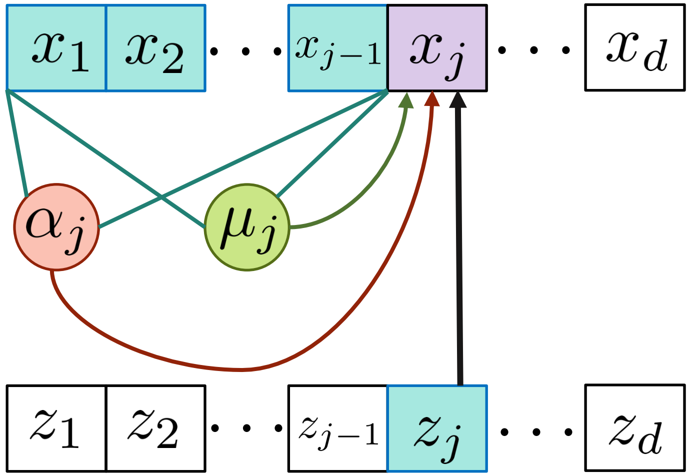
Continuous AR Models as Flow Models

$$p(x) = \prod_{j=1}^{d} p(x_j | x_{<j}).$$

- Sampler for this model:
  1. Sample $z_j \sim \mathcal{N}(0, 1)$ for $j = 1, \ldots, d$.
  2. Let $x_1 = \exp(\alpha_1) z_1 + \mu_1$. Compute $\mu_2(x_1), \alpha_2(x_1)$.
  3. Let $x_2 = \exp(\alpha_2) z_2 + \mu_2$. Compute $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$.
  4. Let $x_3 = \exp(\alpha_3) z_3 + \mu_3$. $\ldots.$

- **Flow Interpretation:** Transforms samples from the standard Gaussian $(z_1, z_2, \ldots, z_d)$ to those generated from the model $(x_1, x_2, \ldots, x_d)$ via invertible transformations (parametrized by $\mu_j(\cdot), \alpha_j(\cdot)$).

Masked Autoregressive Flow (MAF)



Transformed Distribution

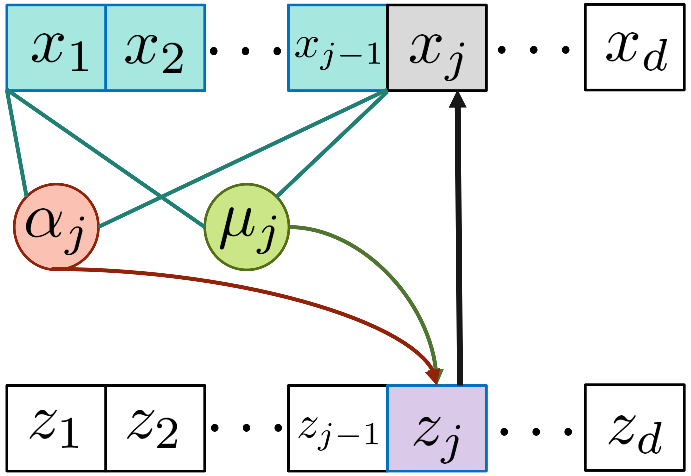
$$x_j = \exp(\alpha_j) z_j + \mu_j,$$
$$\forall j \in \{1, \ldots, d\}.$$

Base Distribution

- Sampling is sequential and slow (i.e., autoregressive): $O(d)$ time.

- Forward mapping from $z \to x$:
  1. Let $x_1 = \exp(\alpha_1) z_1 + \mu_1$. Compute $\mu_2(x_1), \alpha_2(x_1)$.
  2. Let $x_2 = \exp(\alpha_2) z_2 + \mu_2$. Compute $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$.
  3. Let $x_3 = \exp(\alpha_3) z_3 + \mu_3$. $\cdots$

Masked Autoregressive Flow (MAF)



Transformed Distribution

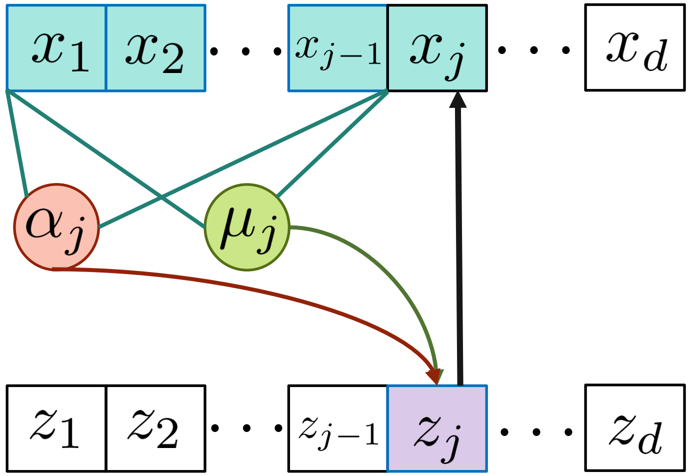
$$z_j = (x_i - \mu_i)\exp(-\alpha_i),$$
$$\forall j \in \{1, \ldots, d\}.$$

Base Distribution

- Inverse mapping from $x \rightarrow z$:

  1. Compute **all** μ_j, α_j (can be done in parallel).
  2. Let $z_1 = (x_1 - \mu_1)\text{exp}(-\alpha_1)$, (scale and shift)
  3. Let $z_2 = (x_2 - \mu_2)\text{exp}(-\alpha_2)$, $\quad \cdots$

Masked Autoregressive Flow (MAF)



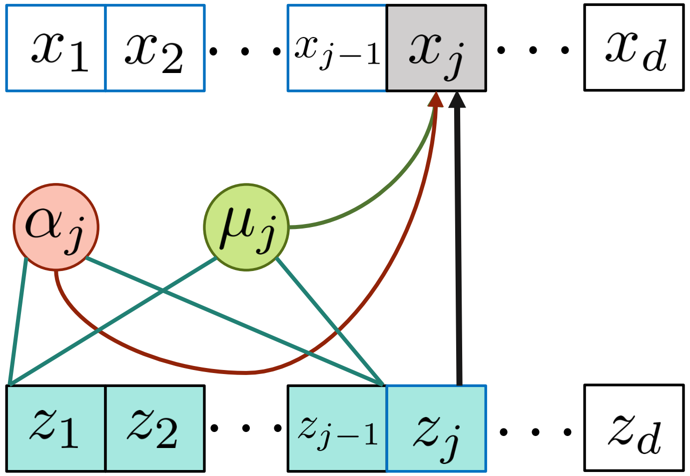
Transformed Distribution

$$z_j = (x_i - \mu_i)\exp(-\alpha_i),$$
$$\forall j \in \{1, \ldots, d\}.$$

Base Distribution

- Jacobian is lower diagonal, hence efficient determinant computation.
- Likelihood evaluation is easy and parallelizable (as in MADE).
- MAF transformations with different variable orderings can be stacked.

Masked Autoregressive Flow (MAF)



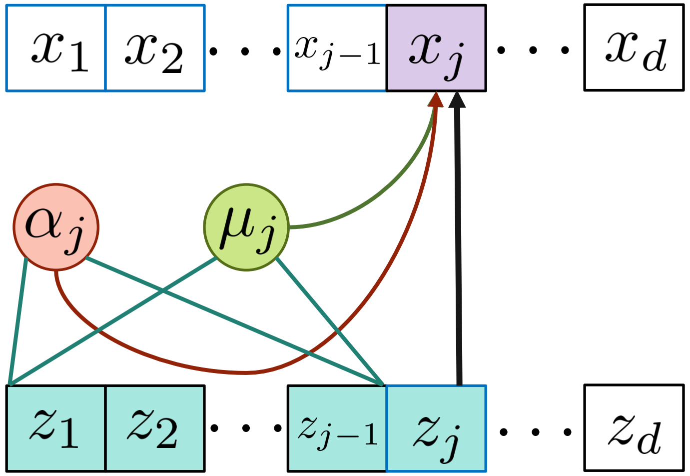
Transformed Distribution

$$x_j = \exp(\alpha_j) z_j + \mu_j,$$
$$\forall j \in \{1, \ldots, d\}.$$

Base Distribution

- Forward mapping from $z \to x$ (parallel):
  1. Sample $z_j \sim \mathcal{N}(0, 1)$, for $j = 1, \ldots, d$.
  2. Compute **all** μ_j, α_j (can be done in parallel).
  3. Let $x_1 = \exp(\alpha_1) z_1 + \mu_1$. Compute $\mu_2(z_1), \alpha_2(z_1)$.
  4. Let $x_2 = \exp(\alpha_2) z_2 + \mu_2$. $\ldots$

Inverse Autoregressive Flow (IAF)



Transformed Distribution

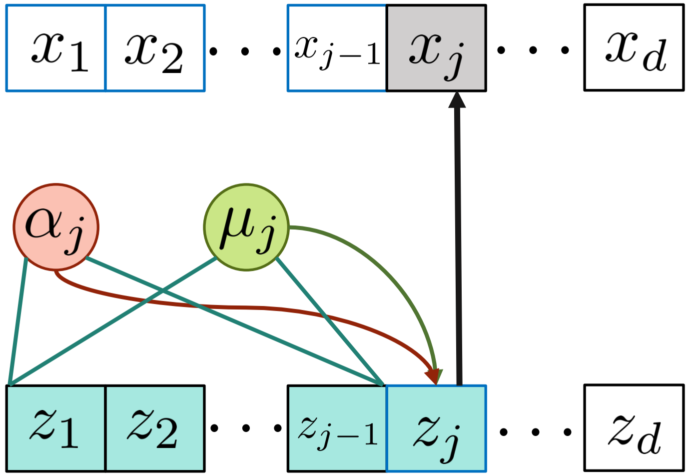
$$z_j = (x_j - \mu_j)\exp(-\alpha_j),$$
$$\forall j \in \{1, \ldots, d\}.$$

Base Distribution

- Inverse mapping from $x \to z$ (sequential):

  1. Let $z_1 = (x_1 - \mu_1)\exp(-\alpha_1)$. Compute $\mu_2(z_1), \alpha_2(z_1)$.
  2. Let $z_2 = (x_2 - \mu_2)\exp(-\alpha_2)$. Compute $\mu_3(z_1, z_2), \alpha_3(z_1, z_2)$.
  3. $\ldots$

Inverse Autoregressive Flow (IAF)



Transformed Distribution

$$x_j = \exp(\alpha_j) z_j + \mu_j,$$
$$\forall j \in \{1, \ldots, d\}.$$

Base Distribution

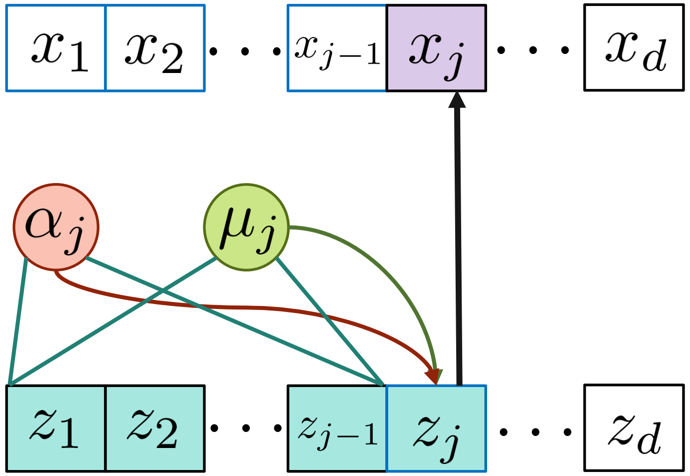
- Inverse mapping from $x \to z$ (sequential):

1. Let $z_1 = (x_1 - \mu_1)\exp(-\alpha_1)$. Compute $\mu_2(z_1), \alpha_2(z_1)$.

2. Let $z_2 = (x_2 - \mu_2)\exp(-\alpha_2)$. Compute $\mu_3(z_1, z_2), \alpha_3(z_1, z_2)$.

3. $\ldots$

Inverse Autoregressive Flow (IAF)



Transformed Distribution

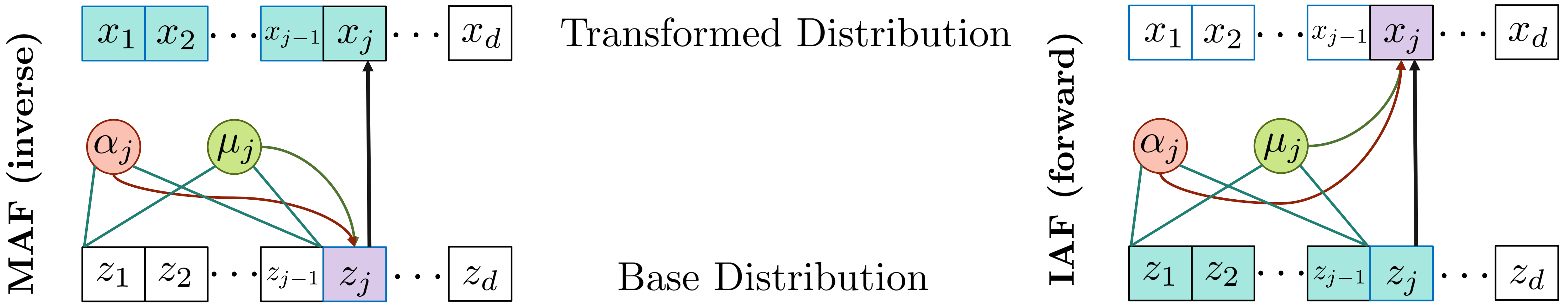
$$z_j = (x_j - \mu_j)\exp(-\alpha_j),$$
$$\forall j \in \{1, \ldots, d\}.$$

Base Distribution

- Fast to sample from but slow to evaluate likelihoods of data points (which is needed during training).

- *But* it is fast to evaluate likelihoods of a generated point (given cached $(z_1, z_2, \ldots, z_d)$).

IAF is the Inverse of MAF



$$z_j = (x_j - \mu_j)\exp(-\alpha_j), \ \forall j \in \{1, \ldots, d\}.$$

$$x_j = \exp(\alpha_j) z_j + \mu_j, \ \forall j \in \{1, \ldots, d\}.$$

- Interchanging z and x in the inverse transformation of MAF gives the forward transformation of IAF.
- Similarly, forward transformation of MAF is inverse transformation of IAF.

IAF vs MAF

- Computational tradeoffs:
  1. MAF: Fast likelihood evaluation but slow sampling.
  2. IAF: Fast sampling but slow likelihood evaluation.
- MAF more suited for training based on MLE and for density estimation.
- IAF more suited for real-time generation (done in parallel).

Can we get the best of both worlds?

- *Parallel WaveNet* (to be presented by Vassilis Tsiaras).

Parallel WaveNet

- Two part training with a teacher and student model.

- Teacher is parameterized by MAF. Teacher can be efficiently trained via MLE.

- Once teacher is trained, initialize a student model parameterized by IAF. Student model cannot efficiently evaluate density for external datapoints but allows for efficient sampling.

- **Key observation**: IAF can also efficiently evaluate densities of its own generations (via caching the noise variates $z_1, z_2, \ldots, z_n$).

Parallel WaveNet

- Probability density distillation: Student distribution is trained to minimize the Kullback–Leibler (KL) divergence between student (s) and teacher (t):
$$D_{\mathrm{KL}}(s, t) = E_{x \sim s}\left[\log s(x) - \log t(x)\right].$$
- Evaluating and optimizing Monte Carlo estimates of this objective requires:
  1. Samples x from student model (IAF).
  2. Density of x assigned by student model.
  3. Density of x assigned by teacher model (MAF).
- All operations above can be implemented efficiently.

Parallel WaveNet

- Training:
  1. Train teacher model (MAF) via MLE.
  2. Train student model (IAF) to minimize KL divergence with teacher.
- Test-time: Use student model for testing.
- Improves sampling efficiency over original Wavenet (vanilla autoregressive model) by 1000 times!

References

1. Probabilistic Machine Learning: Advanced Topics (*Chapter 22*)
   Kevin P Murphy, The MIT Press (2023)

2. Normalizing Flows for Probabilistic Modeling and Inference, Papamakarios et al., JMLR, 2021. (A coherent and accessible summary to normalizing flows - Highly recommended read!)

3. `https://github.com/janosh/awesome-normalizing-flows` (list of papers and source code).

4. `https://lilianweng.github.io/posts/2018-10-13-flow-models/`

5. `https://tech.skit.ai/normalizing-flows-part-2/`

References

6. NICE: Non-linear Independent Components Estimation by Dinh, Krueger et al.
7. MADE: Masked Autoencoder for Distribution Estimation by Germain et al.
8. RealNVP: Density estimation using Real NVP by Dinh, Sohl-Dickstein et al.
9. IAF: Improving Variational Inference with Inverse Autoregressive Flow by Kingma, Salimans et al.
10. MAF: Masked Autoregressive Flow for Density Estimation by Papamakarios, Pavlakou et al.
11. Glow: Generative Flow with Invertible 1x1 Convolutions by Kingma & Dhariwal

References

12. Neural Autoregressive Flows by Huang, Krueger et al.
13. Sylvester Normalizing Flow for Variational Inference by Berg, Hasenclever et al.
14. Neural Spline Flows by Durkan, Bekasov et al.
15. FloWaveNet : A Generative Flow for Raw Audio by Kim, Lee et al.
16. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models by Grathwohl, Chen et al.
17. MintNet: Building Invertible Neural Networks with Masked Convolutions by Song, Meng et al.
18. Multivariate Probabilistic Time Series Forecasting via Conditioned Normalizing Flows by Rasul, Sheikh et al.
19. iUNets: Fully invertible U-Nets with Learnable Upand Downsampling by Etmann, Ke et al.

Introduction to Deep Generative Modeling

Lecture #8

HY-673 – Computer Science Dep., University of Crete

Professors: Yannis Pantazis, Yannis Stylianou

Teaching Assistant: Michail Raptakis