Generative Models II: Explicit Density Models

AI602: Recent Advances in Deep Learning

Lecture 6

Slide made by

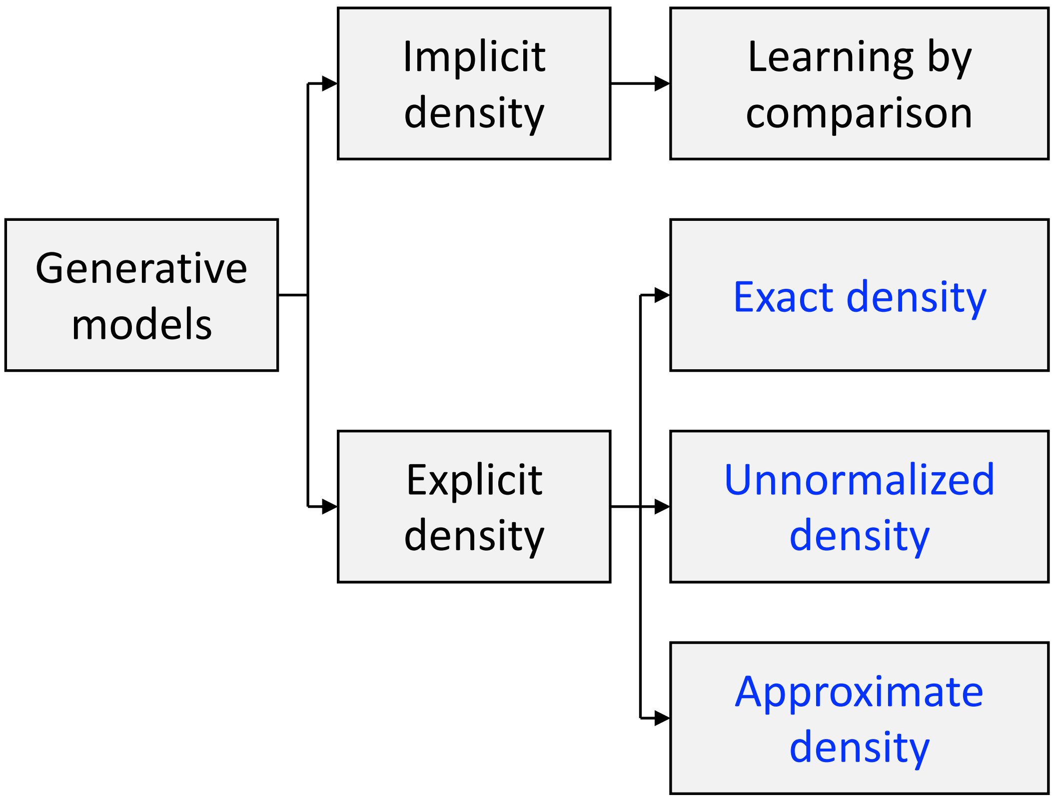
Sangwoo Mo, Chaewon Kim, Minkyu Kim, Younggyo Seo

KAIST EE

Table of Contents

Table of Contents

- From now on, we study generative models with explicit density estimation:



- Only care about generation, not $p(x)$
- Instead of maximizing density, compare real vs generated samples (i.e., reduce generation problem to classification)
- Example: GAN, GMMN

- Directly learn density $p(x)$
- Example: autoregressive, flow

- Learn unnormalized density $E(x) \propto p(x)$
- Example: EBM, score matching

- Learn approximation (e.g., lower bound) of density $\mathcal{L}(x) \leq p(x)$
- Example: VAE, diffusion model

Implicit vs Explicit Density Models

- From now on, we study generative models with explicit density estimation:

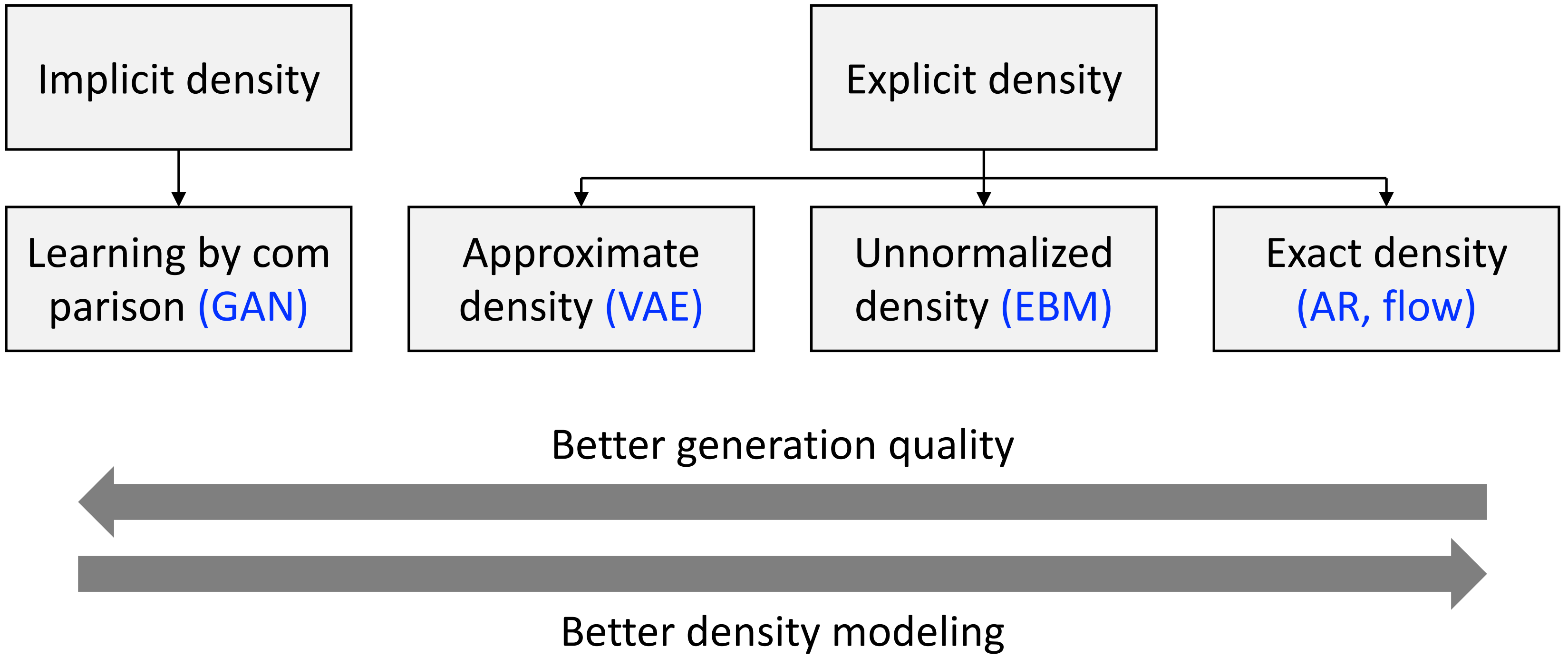
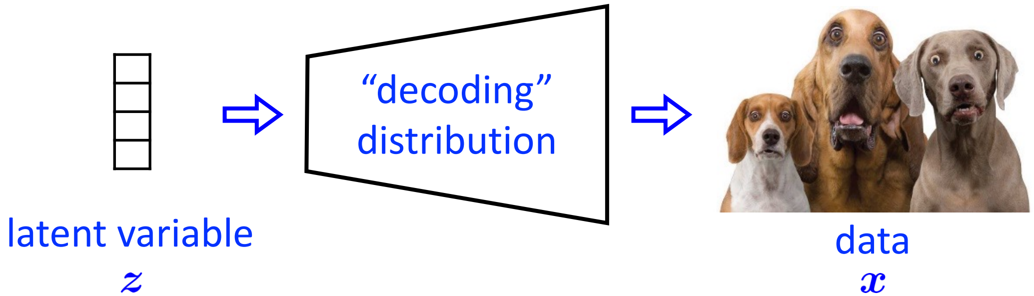


Table of Contents

- Consider the following generative model:



  - Fixed prior on random latent variable
    - e.g., standard Normal distribution

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbb{I})$$

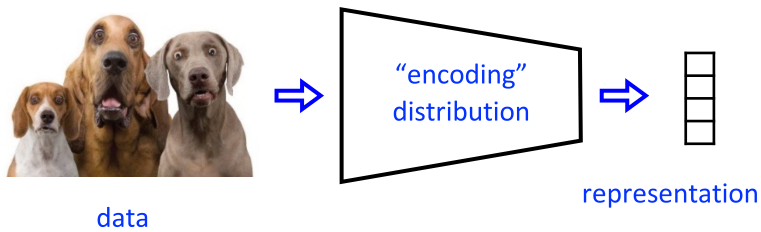
  - Parameterized likelihood (decoder) for generation:
    - e.g., Normal distribution parameterized by neural network

$$p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; f_{\mathtt{dec}}(\boldsymbol{z}), \mathbb{I})$$

  - Resulting generative distribution (to optimize):

$$\log p_\theta(\boldsymbol{x}) = \log \int_{\boldsymbol{z}} p_\theta(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) d\boldsymbol{z} = \log \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[p(\boldsymbol{x}|\boldsymbol{z})]$$

- Variational autoencoder (VAE) introduce an auxiliary distribution (encoder) [Kingma et al., 2013]



$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; f_{\mathtt{enc},\mu}(\boldsymbol{x}), f_{\mathtt{enc},\sigma}(\boldsymbol{x}))$$

- Each $\log p_\theta(\boldsymbol{x})$ term is replaced by its <u>lower bound</u>:

$$\begin{aligned}
\log p_\theta(\boldsymbol{x}) &\geq \log p_\theta(\boldsymbol{x}) - \min_\phi \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})) \\
&= \log p_\theta(\boldsymbol{x}) + \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{z}|\boldsymbol{x}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})] \\
&= \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}) + \log p_\theta(\boldsymbol{z}|\boldsymbol{x}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})] \\
&= \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))
\end{aligned}$$

- Bound becomes equality when $q_\phi(\boldsymbol{z}|\boldsymbol{x}) \approx p_\theta(\boldsymbol{z}|\boldsymbol{x})$

- The training objective becomes:

tractable between two Gaussian distributions

$$\max_\theta \sum_{n=1}^{N} \log p_\theta(\boldsymbol{x}^{(n)}) \geq \max_\theta \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))$$

$$\approx \max_\theta \max_\phi \sum_{n=1}^{N} \sum_{k=1}^{N} \log p_\theta(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)}) - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})||p(\boldsymbol{z}))$$

where latent variables are sampled by $\boldsymbol{z}^{(n,k)} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})$

- However, non-trivial to train with back propagation due to sampling procedure:

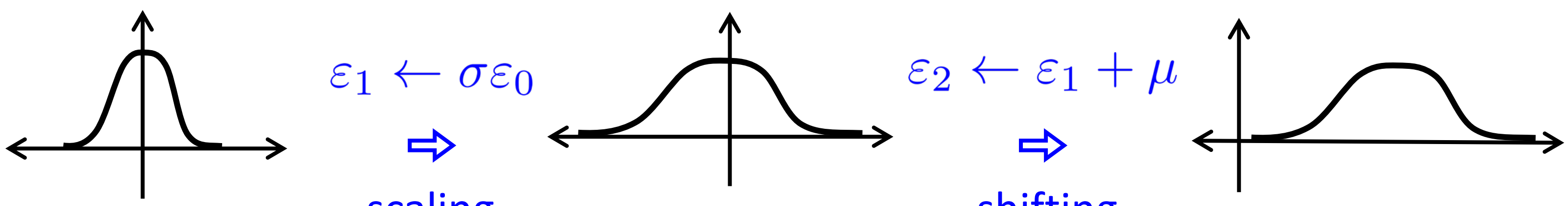
$$\nabla_\phi \mathcal{L} = \sum_{n=1}^{N} \sum_{k=1}^{N} - \nabla_\phi \log p_\theta(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)}) + \nabla_\phi \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})||p(\boldsymbol{z}))$$

$\Downarrow$

Since $\boldsymbol{z}^{(n,k)}$ is fixed after being sampled, $\nabla_\phi \log p(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)}) = 0$?

- Reparameterization trick is based on the change-of-variables formula:

$$\varepsilon_2 \sim \mathcal{N}(\varepsilon_2|\mu,\sigma) \Leftrightarrow \varepsilon_2 = \mu + \sigma\varepsilon_0, \qquad \varepsilon_0 \sim \mathcal{N}(\varepsilon_0|0,1)$$



$\varepsilon_1 \leftarrow \sigma\varepsilon_0$ ⇨ scaling

$\varepsilon_2 \leftarrow \varepsilon_1 + \mu$ ⇨ shifting

$\varepsilon_0 \sim \mathcal{N}(\varepsilon_0|0,1)$ $\qquad$ $\varepsilon_1 \sim \mathcal{N}(\varepsilon_1|0,\sigma)$ $\qquad$ $\varepsilon_2 \sim \mathcal{N}(\varepsilon_2|\mu,\sigma)$

- Latent variable $\boldsymbol{z}^{(n,k)}$ can be similarly parameterized by encoder network:

$$\boldsymbol{z}^{(n,k)} \sim \mathcal{N}(\boldsymbol{z}; f_{\mathtt{enc},\mu}(\boldsymbol{x}^{(n)}), f_{\mathtt{enc},\sigma}(\boldsymbol{x}^{(n)}))$$

$$\Updownarrow$$

$$\boldsymbol{z}^{(n,k)} = f_{\mathtt{enc},\mu}(\boldsymbol{x}^{(n)}) + f_{\mathtt{enc},\sigma}(\boldsymbol{x}^{(n)}) \odot \boldsymbol{\varepsilon}^{(n,k)}, \qquad \boldsymbol{\varepsilon}^{(n,k)} \sim \mathcal{N}(\boldsymbol{\varepsilon}|\mathbf{0},\mathbf{1})$$

- Total loss of variational autoencoder:

$$\nabla_\phi \mathcal{L} = \sum_{n=1}^{N} \sum_{k=1}^{N} \underbrace{-\nabla_\phi \log p_\theta(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)})}_{\nabla_\phi \mathcal{L}_1} + \underbrace{\nabla_\phi \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})||p(\boldsymbol{z}))}_{\nabla_\phi \mathcal{L}_2}$$

- Recall that $f_{\texttt{dec}}, f_{\texttt{enc},\mu}, f_{\texttt{enc},\sigma}$ are parameterized by ϕ

- Derivative of first part:

$$\nabla_\phi \mathcal{L}_1 = \nabla_\phi \log \mathcal{N}(\boldsymbol{x}^{(n)}; f_{\texttt{dec}}(\boldsymbol{z}^{(n,k)}), \mathbf{1})$$

$\Downarrow$ log-normal distribution

$$= \nabla_\phi \frac{1}{2} \|\boldsymbol{x}^{(n)} - f_{\texttt{dec}}(\boldsymbol{z}^{(n,k)})\|_2^2$$

$\Downarrow$ reparameterization trick

$$= \nabla_\phi \frac{1}{2} \|\boldsymbol{x}^{(n)} - f_{\texttt{dec}}(f_{\texttt{enc},\mu}(\boldsymbol{x}^{(n)}) + f_{\texttt{enc},\sigma}(\boldsymbol{x}^{(n)}) \odot \boldsymbol{\varepsilon}^{(n,k)})\|_2^2$$

- Total loss of variational autoencoder:

$$\nabla_\phi \mathcal{L} = \sum_{n=1}^{N}\sum_{k=1}^{N} \underbrace{-\nabla_\phi \log p_\theta(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)})}_{\nabla_\phi \mathcal{L}_1} + \underbrace{\nabla_\phi \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})||p(\boldsymbol{z}))}_{\nabla_\phi \mathcal{L}_2}$$

- Recall that $f_{\mathtt{dec}}, f_{\mathtt{enc},\mu}, f_{\mathtt{enc},\sigma}$ are parameterized by ϕ

- Derivative of second part:

$$\nabla_\phi \mathcal{L}_1 = \nabla_\phi \mathrm{KL}(\mathcal{N}(\boldsymbol{z}; f_{\mathtt{enc},\mu}(\boldsymbol{x}^{(n)}), f_{\mathtt{enc},\sigma}(\boldsymbol{x}^{(n)}))||\mathcal{N}(\boldsymbol{z};\mathbf{0},\mathbf{1}))$$

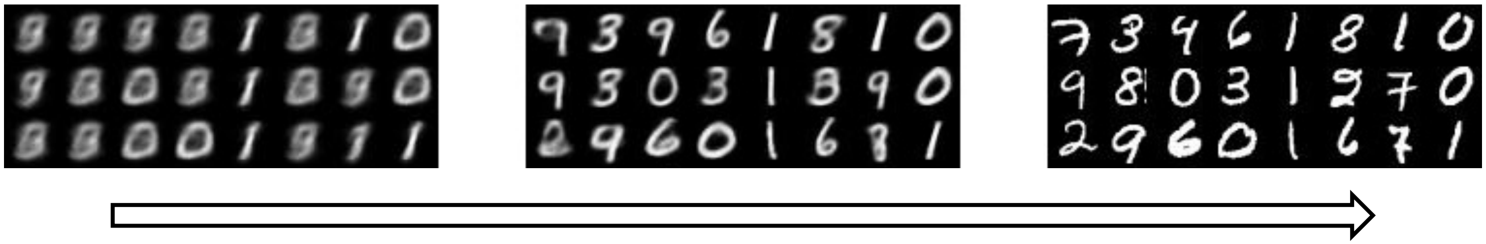
$\Downarrow$ element-wise factorization ($\boldsymbol{z} = [z_1, \cdots, z_K]$)

$$= \sum_{k=1}^{K} \nabla_\phi \mathrm{KL}(\mathcal{N}(z_k; f_{\mathtt{enc},\mu,k}(\boldsymbol{x}^{(n)}), f_{\mathtt{enc},\sigma,k}(\boldsymbol{x}^{(n)}))||\mathcal{N}(z_k;0,1))$$

$\Downarrow$ KL divergence between normal distributions

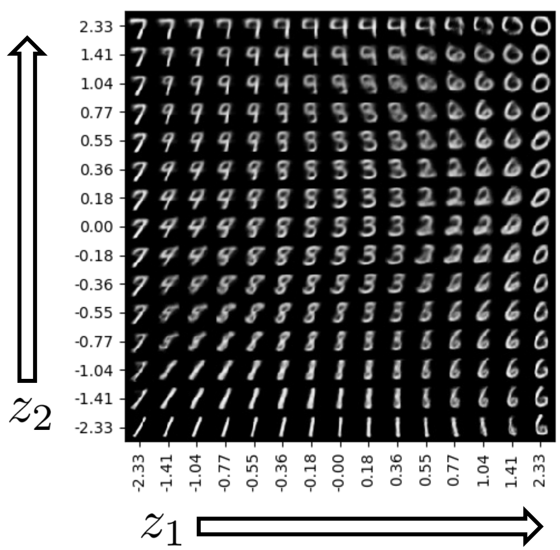
$$= \sum_{k=1}^{K} \nabla_\phi - \log f_{\mathtt{enc},\sigma,k}(\boldsymbol{x}^{(n)}) + \frac{1}{2} f_{\mathtt{enc},\sigma,k}(\boldsymbol{x}^{(n)})^2 + \frac{1}{2} f_{\mathtt{enc},\sigma,k}(\boldsymbol{x}^{(n)})^2$$

- Based on the proposed scheme, variational autoencoder successfully generates images:



Training on MNIST

- Interpolation of latent variables induce transitions in generated images:



- Although VAE has many advantages (e.g., fast sampling, full mode covering, latent embedding), there are issues that lead to **poor generation quality**

- Tighter objective bound
  - **Reduce approximation (model) error:** Importance-weighted AE (IWAE)
  - **Reduce amortization (sample-wise) error:** Semi-amortized VAE (SA-VAE)

- Posterior collapse (latents are ignored when paired with powerful decoder)
  - **Careful optimization:** various techniques for continuous latent-space VAEs
  - **Use discrete latent space:** Vector-quantized VAE (VQ-VAE, VQ-GAN)

- Improve model expressivity
  - **Use expressive prior distribution:** Gaussian mixtures, normalizing flow
  - **Use hierarchical architectures:** Hierarchical VAE, Diffusion Models

Improving VAEs

- Although VAE has many advantages (e.g., fast sampling, full mode covering, latent embedding), there are issues that lead to **poor generation quality**

- Tighter objective bound
  - **Reduce approximation (model) error:** Importance-weighted AE (IWAE)
  - **Reduce amortization (sample-wise) error:** Semi-amortized VAE (SA-VAE)

- Posterior collapse (latents are ignored when paired with powerful decoder)
  - **Careful optimization:** various techniques for continuous latent-space VAEs
  - **Use discrete latent space:** Vector-quantized VAE (VQ-VAE, VQ-GAN)

- Improve model expressivity
  - **Use expressive prior distribution:** Gaussian mixtures, normalizing flow
  - **Use hierarchical architectures:** Hierarchical VAE, Diffusion Models

- Observe that ELBO can also be proved by the Jensen's inequality:

$$\log p(\boldsymbol{x}) = \log \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] \geq \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right]$$

  - Based on convexity, interchange order of logarithm and summation

- Importance weighted AE (IWAE) relax the inequality [Burda et al., 2018]:

$$\log p(\boldsymbol{x}) = \log \mathbb{E}_{\boldsymbol{z}^{(1)},\cdots,\boldsymbol{z}^{(K)}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{1}{K}\sum_{k=1}^{K}\frac{p(\boldsymbol{x},\boldsymbol{z}^{(k)})}{q_\phi(\boldsymbol{z}^{(k)}|\boldsymbol{x})}\right]$$

$$\geq \mathbb{E}_{\boldsymbol{z}^{(1)},\cdots,\boldsymbol{z}^{(K)}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{1}{K}\sum_{k=1}^{K}\boxed{\frac{p(\boldsymbol{x},\boldsymbol{z}^{(k)})}{q_\phi(\boldsymbol{z}^{(k)}|\boldsymbol{x})}}\right]$$

also called importance weights

  - Becomes original ELBO when $K=1$ and becomes exact bound when $K=\infty$

$$\mathbb{E}_{\boldsymbol{z}^{(1)},\cdots,\boldsymbol{z}^{(K)}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{1}{K}\sum_{k=1}^{K}\frac{p(\boldsymbol{x},\boldsymbol{z}^{(k)})}{q_\phi(\boldsymbol{z}^{(k)}|\boldsymbol{x})}\right] \approx p(\boldsymbol{x})$$

Semi-amortized VAE (SA-VAE)

- Inference gap of VAE can be decomposed to approximation gap (model error) and amortization gap (single neural network amortizes all posteriors)

- Semi-amortized VAE: In addition to the global inference network, update the posterior of each local instance for a few steps [Kim et al., 2018]
  - Resembles MAML (see future lecture)

1. Sample $\mathbf{x} \sim p_{\mathcal{D}}(\mathbf{x})$
2. Set $\lambda_0 = \text{enc}(\mathbf{x}; \phi)$

   → shared to all samples
3. For $k = 0, \ldots, K-1$, set
   $\lambda_{k+1} = \lambda_k + \alpha \nabla_\lambda \text{ELBO}(\lambda_k, \theta, \mathbf{x})$

   → specific to each sample x

- Semi-amortized VAE can further reduce ELBO, applied on top of any VAEs

Model	Oracle Gen	Learned Gen
VAE	≤ 21.77	≤ 27.06
SVI	≤ 22.33	≤ 25.82
SA-VAE	≤ 20.13	≤ 25.21
True NLL (Est)	19.63	–

* SVI: Instance-specific posterior only, without amortization

Improving VAEs

- Although VAE has many advantages (e.g., fast sampling, full mode covering, latent embedding), there are issues that lead to **poor generation quality**

- Tighter objective bound
  - **Reduce approximation (model) error:** Importance-weighted AE (IWAE)
  - **Reduce amortization (sample-wise) error:** Semi-amortized VAE (SA-VAE)

- Posterior collapse (latents are ignored when paired with powerful decoder)
  - **Careful optimization:** various techniques for continuous latent-space VAEs
  - **Use discrete latent space:** Vector-quantized VAE (VQ-VAE, VQ-GAN)

- Improve model expressivity
  - **Use expressive prior distribution:** Gaussian mixtures, normalizing flow
  - **Use hierarchical architectures:** Hierarchical VAE, Diffusion Models

- Posterior collapse [Bowman et al., 2016]:
  - When paired with powerful decoder, VAEs often ignore the posterior $q_\phi(z|x)$ and generates generic samples (i.e., reconstruction loss does not decrease well)

- To mitigate posterior collapse, prior works attempt

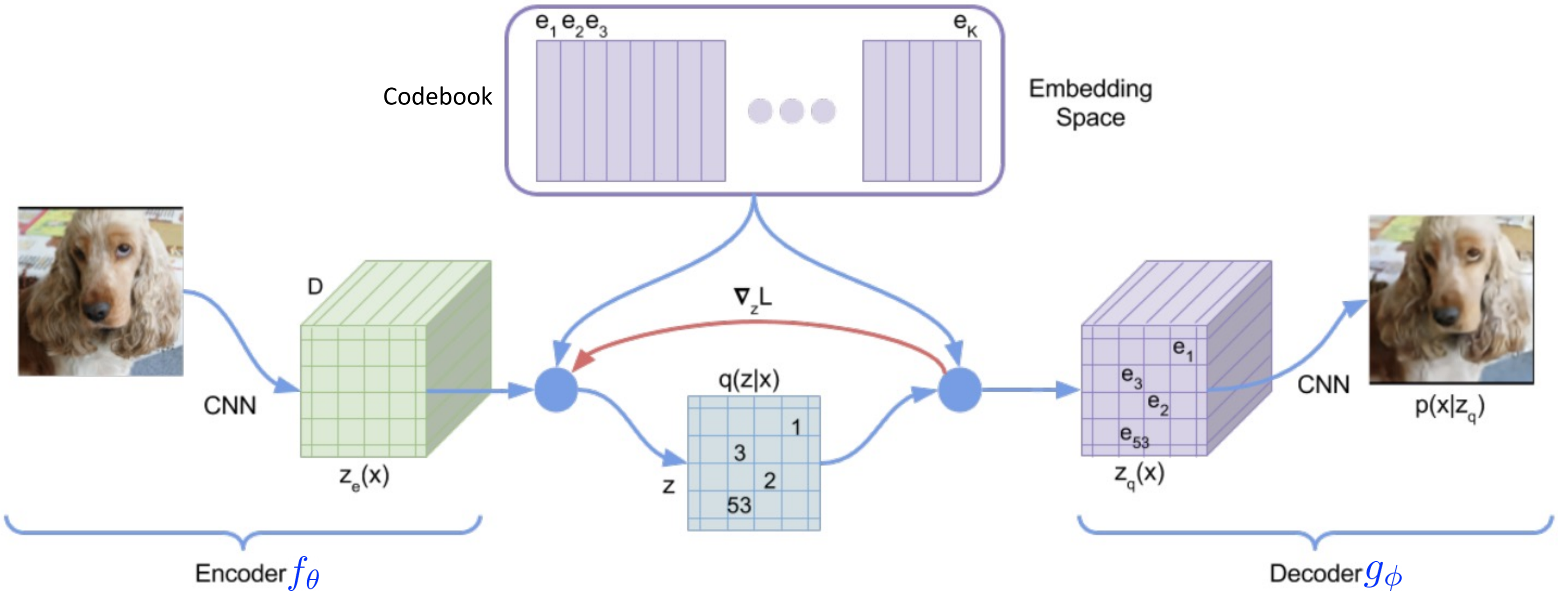
  1. Weaken the KL regularization term [Bowman et al., 2016, Razavi et al., 2019a]
     - Recall: KL regularization term minimizes $\mathrm{KL}(p_\phi(z|x), p(z))$
     - Anneal the weight during training, or constraint $\geq \delta$

  2. Match aggregated posterior instead of individuals [Tolstikhin et al., 2018]
     - Instead of matching $p_\phi(z|x) \approx p(z)$ for all x, match the aggregated posterior $\mathbb{E}_{x\sim p(x)}\, p_\phi(z|x) \approx p(z)$ (each $p_\phi(z|x)$ is now a deterministic, single point)
     - Need implicit distribution matching techniques (e.g., GAN)

  3. Improve optimization procedure [He et al., 2019]
     - Strengthen the encoder: update encoder until converge, and decoder once

Vector-quantized VAE (VQ-VAE)

- VQ-VAE [Oord et al., 2017]
  - Each data is embedded into combination of 'discrete' latent vectors: $\{e_1, \cdots, e_K\}$
  - **i.e.)** each encoder output is quantized to the nearest vector among K codebook vectors



- Restriction of latent space achieves high generation quality including:
  - Images, videos, audios, etc.

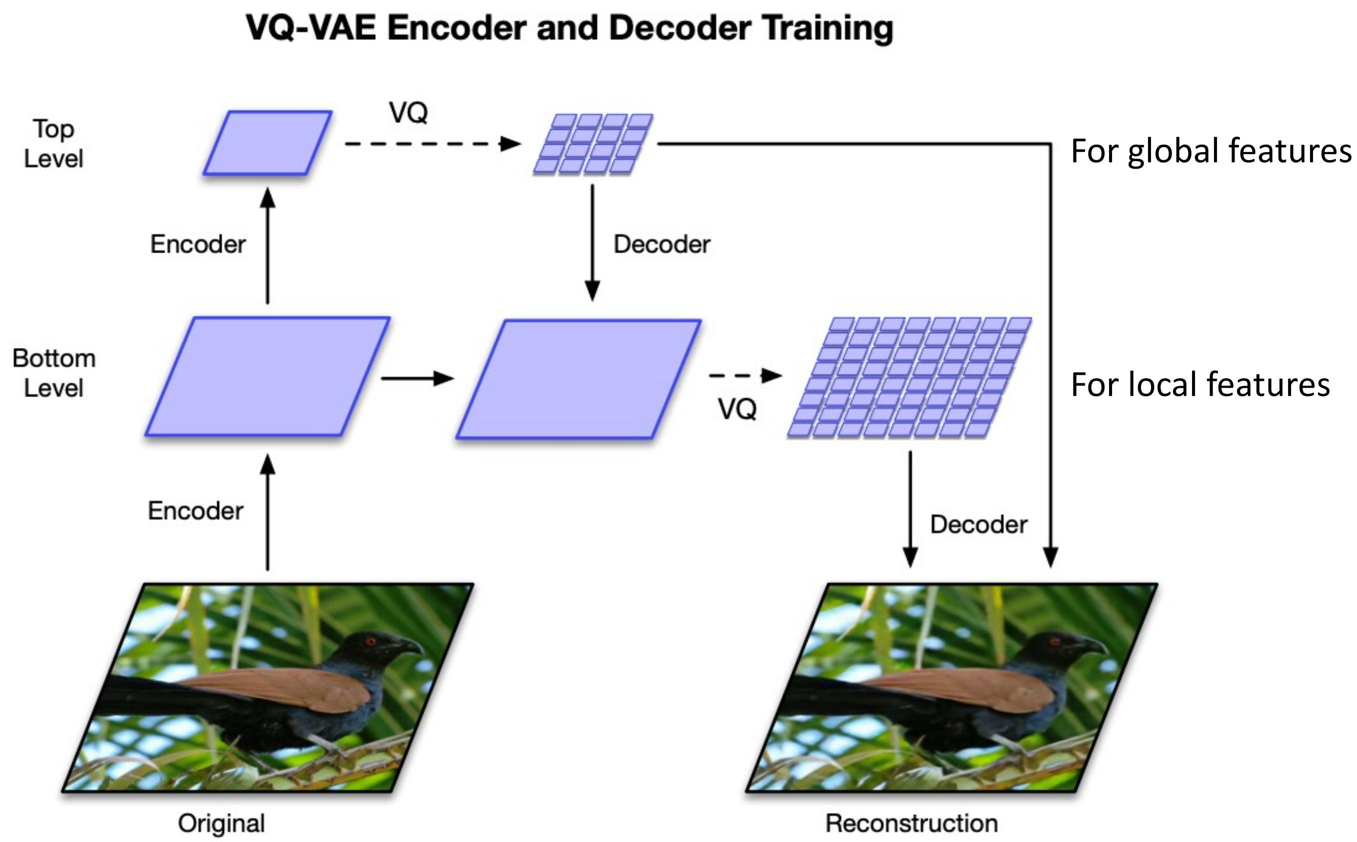
- VQ-VAE [Oord et al., 2017]
  - The objective of VQ-VAE composed of three terms:
    - Reconstruction loss (1)
    - VQ loss (2):
      - Optimization of codebook vectors
    - Commitment loss (3):
      - Regularization to get encoder outputs and codebook close

$$\mathcal{L} = \underbrace{||g_\phi(e) - x||_2^2}_{(1)} + \underbrace{||\mathbf{sg}(f_\theta(x)) - e||_2^2}_{(2)} + \underbrace{\beta||f_\theta(x) - \mathbf{sg}(e)||_2^2}_{(3)}$$

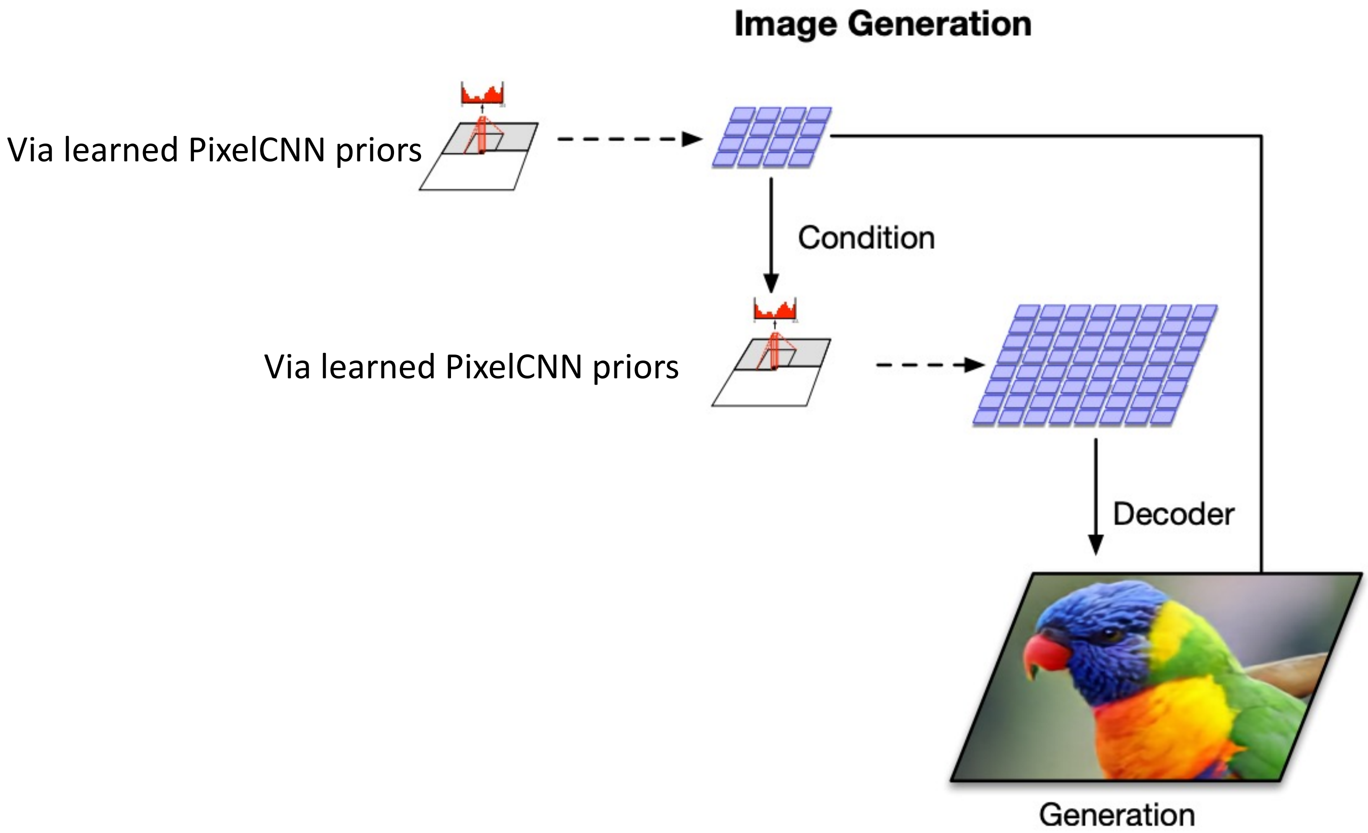
- VQ-VAE like methods (i.e. discrete prior) recently shows remarkable success on:
  - DALL-E (text-image generative model) – image is encoded via VQ-VAE
  - Many audio self-supervised learning method

Vector-quantized VAE + Hierarchical Architecture (VQ-VAE-2)

- VQ-VAE-2 [Razavi et al., 2019b]
  - Different from VQ-VAE, vector quantization occurs twice (top, bottom level)
  - For both consideration of local/global features for high-fidelity image



- VQ-VAE-2 [Razavi et al., 2019b]
    - After VQ-VAE-2 training, train two pixelCNN priors for new image generation
    - They autoregressively fill out each quantized latent vector space



- Generated images are comparable to state-of-the-art GAN model (e.g. BigGAN)

- Although VAE has many advantages (e.g., fast sampling, full mode covering, latent embedding), there are issues that lead to **poor generation quality**

- Tighter objective bound
  - **Reduce approximation (model) error:** Importance-weighted AE (IWAE)
  - **Reduce amortization (sample-wise) error:** Semi-amortized VAE (SA-VAE)

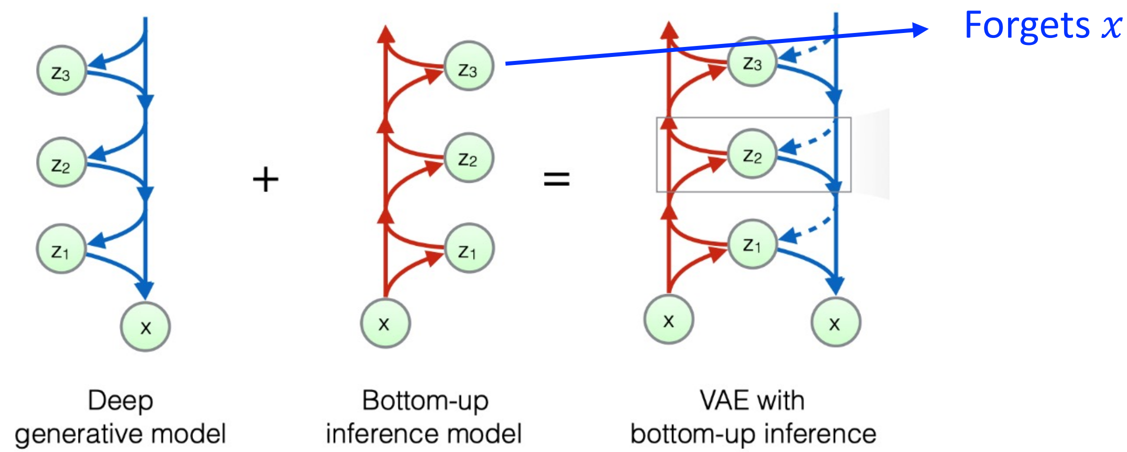
- Posterior collapse (latents are ignored when paired with powerful decoder)
  - **Careful optimization:** various techniques for continuous latent-space VAEs
  - **Use discrete latent space:** Vector-quantized VAE (VQ-VAE)

- Improve model expressivity
  - **Use expressive prior distribution:** Gaussian mixtures, normalizing flow
  - **Use hierarchical architectures:** Hierarchical VAE, Diffusion Models

- NVAE [Vahdat et al., 2020]
  - Hierarchical VAEs use the factorized latent space $p_\theta(z) = \prod_l p_\theta(z_l | z_{<l})$
  - Here, the ELBO objective is given by

$$\mathcal{L}_{\text{VAE}}(\boldsymbol{x}) := \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] - \text{KL}(q(\boldsymbol{z}_1|\boldsymbol{x})||p(\boldsymbol{z}_1)) - \sum_{l=2}^{L} \mathbb{E}_{q(\boldsymbol{z}_{<l}|\boldsymbol{x})}\left[\text{KL}(q(\boldsymbol{z}_l|\boldsymbol{x}, \boldsymbol{z}_{<l})||p(\boldsymbol{z}_l|\boldsymbol{z}_{<l}))\right],$$

  - However, **prior attempts** on hierarchical VAE were not so successful due to:
    1. Long-range correlation: upper latents often **forget** the data information

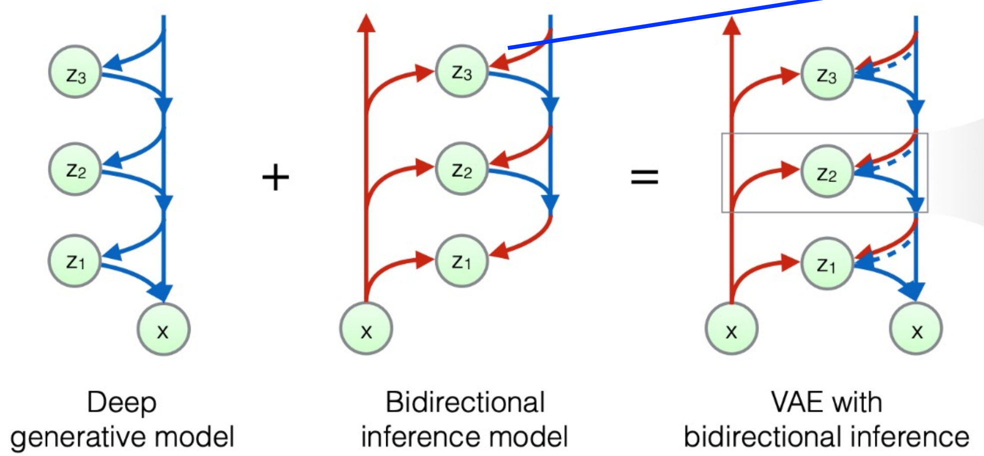


    2. Unstable (unbounded) KL term: even more severe for hierarchical VAEs since they **jointly learn** the prior distribution $p_\theta(z)$

Both $q_\phi(z|x)$ and $p_\theta(z)$ are moving during training

- NVAE [Vahdat et al., 2020]
  - **Idea 1.** Bidirectional encoder (originally from [Kingma et al., 2016])
    - Enforce upper latents (e.g., z_3) to predict the lower latents (e.g., z_1) → Improve the long-range correlation issue

Better remembers x (should reconstruct z_1)



- **Training: posterior** $q_\phi(z|x)$ is inferred by both encoder and decoder (aggregate them) and **prior** $p_\theta(z)$ is jointly inferred by decoder
  - Recall that the KL term is a function of $q_\phi(z|x)$ and $p_\theta(z)$

- **Inference:** Sample **prior** $p_\theta(z)$ from decoder and generate sample x

- NVAE [Vahdat et al., 2020]
  - **Idea 2.** Taming the unstable KL term

1. Residual normal distribution
   - For each factorized **prior** distribution

$$p(z_l^i|\boldsymbol{z}_{<l}) := \mathcal{N}(\mu_i(\boldsymbol{z}_{<l}), \sigma_i(\boldsymbol{z}_{<l})),$$

   define **approximate posterior** as (instead of directly predict μ_i, σ_i)

$$q(z_l^i|\boldsymbol{z}_{<l}, \boldsymbol{x}) := \mathcal{N}(\mu_i(\boldsymbol{z}_{<l}) + \Delta\mu_i(\boldsymbol{z}_{<l}, \boldsymbol{x}), \sigma_i(\boldsymbol{z}_{<l}) \cdot \Delta\sigma_i(\boldsymbol{z}_{<l}, \boldsymbol{x})),$$

   - Then, the **KL term** of ELBO is given by

$$\mathrm{KL}(q(z^i|\boldsymbol{x})||p(z^i)) = \frac{1}{2}\left(\frac{\Delta\mu_i^2}{\sigma_i^2} + \Delta\sigma_i^2 - \log\Delta\sigma_i^2 - 1\right)$$

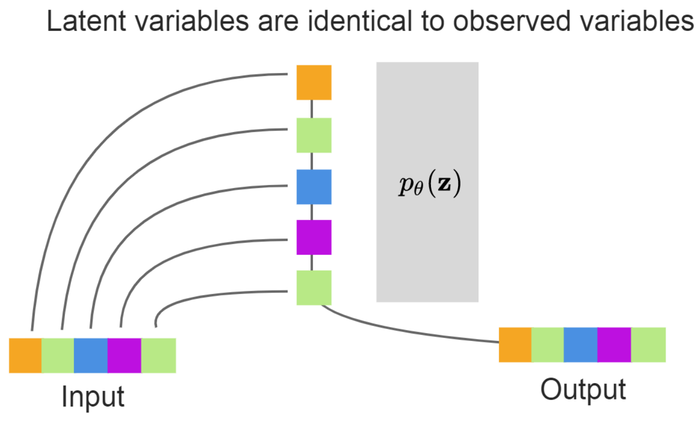
2. Spectral regularization
   - Enforce *Lipschitz smoothness* of encoder to bound KL divergence
   - Regularize the *largest singular value* of convolutional layers (estimated by power iteration [Yoshida & Miyato, 2017])

- NVAE [Vahdat et al., 2020]
  - **Results:**
    - Generate high-resolution (256x256) images

    - SOTA test negative log-likelihood (NLL) on non-autoregressive models

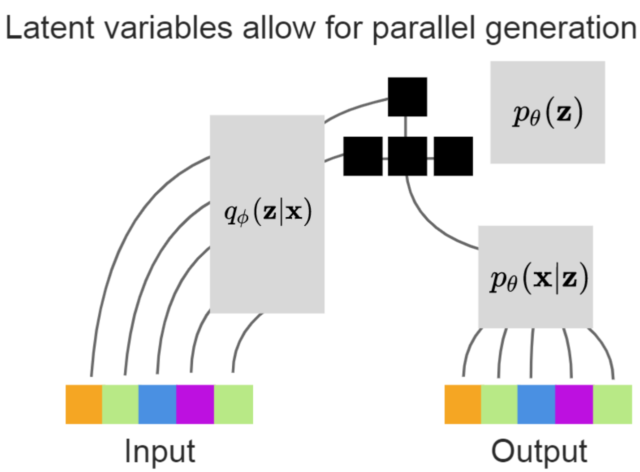
Method	MNIST 28×28	CIFAR-10 32×32	ImageNet 32×32	CelebA 64×64	CelebA HQ 256×256	FFHQ 256×256
NVAE w/o flow	**78.01**	2.93	-	2.04	-	0.71
NVAE w/ flow	78.19	**2.91**	3.92	**2.03**	**0.70**	**0.69**
VAE Models with an Unconditional Decoder						
BIVA [36]	78.41	3.08	3.96	2.48	-	-
IAF-VAE [4]	79.10	3.11	-	-	-	-
DVAE++ [20]	78.49	3.38	-	-	-	-
Conv Draw [42]	-	3.58	4.40	-	-	-
Flow Models without any Autoregressive Components in the Generative Model						
VFlow [59]	-	2.98	-	-	-	-
ANF [60]	-	3.05	3.92	-	0.72	-
Flow++ [61]	-	3.08	**3.86**	-	-	-
Residual flow [50]	-	3.28	4.01	-	0.99	-
GLOW [62]	-	3.35	4.09	-	1.03	-
Real NVP [63]	-	3.49	4.28	3.02	-	-

- VD-VAE [Child, 2021]
  - Autoregressive models have outperformed VAEs (will be covered later)
  - **Main idea:** However, very deep VAEs generalize autoregressive models



Observation 1: Hierarchical VAEs with N layers (N = dimension of data D) generalizes autoregressive models

- e.g.) learns deterministic identity function

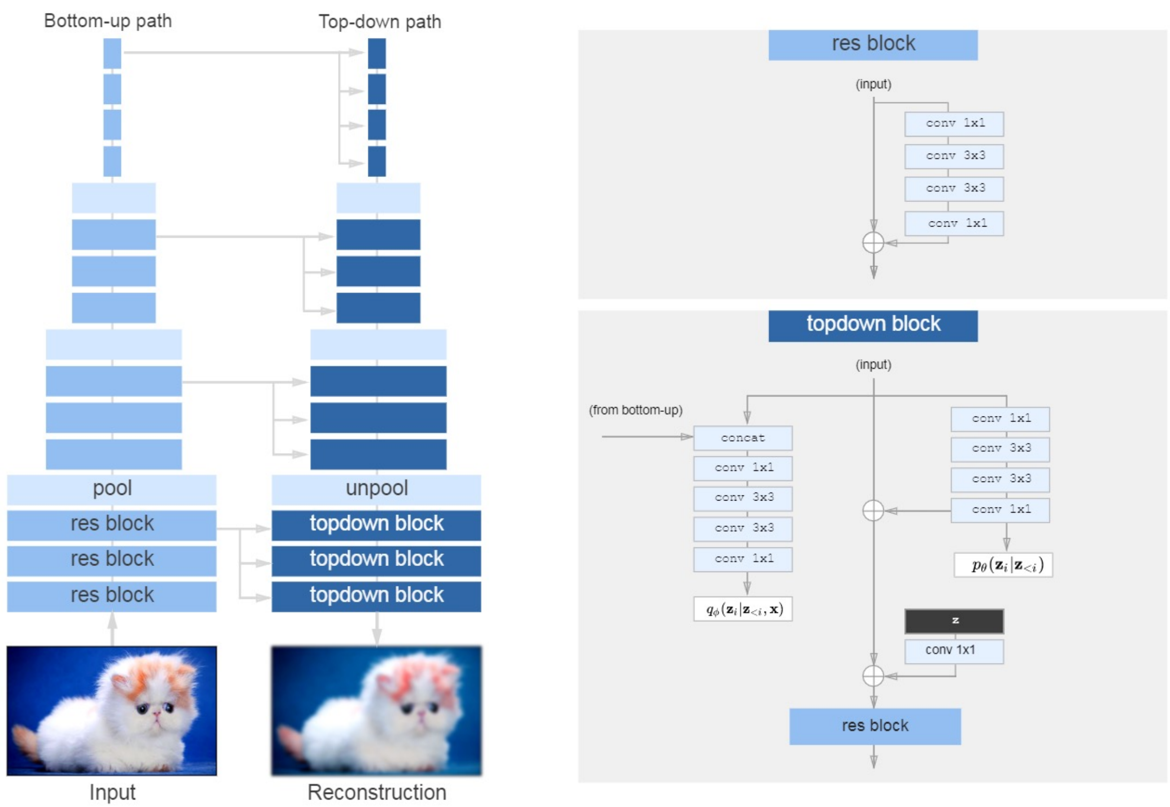


Observation 2: VAEs with fewer layers (N < D) can still model data by learning efficient hierarchies of latent variables

- e.g.) learns conditional independence

Very Deep VAE (VD-VAE)

- VD-VAE [Child, 2021]
  - Empirically, deep VAEs often suffer from unstable training
    - Recap: NVAE requires complex techniques to stabilize KL
  - **Q:** How to make VAE deeper?
    - **Idea 1:** Top-down architecture with bottleneck residual blocks



- VD-VAE [Child, 2021]
  - Empirically, deep VAEs often suffer from unstable training
    - Recap: NVAE requires complex techniques to stabilize KL
  - **Q:** How to make VAE deeper?
    - **Idea 2:** Additional simple techniques
      - Transposed CNNs **=> Nearest-neighbor upsampling**
      - **Scale down weight initialization** of final layer in residual block
      - **Gradient skipping**: skip updates when gradient norm is above threshold

Very Deep VAE (VD-VAE)

- VD-VAE [Child, 2021]
  - **Results:** Very deep VAEs (>50 layers) can outperform autoregressive models with fe wer parameters while maintaining fast sampling

	Model type	Params	Depth	Sampling	NLL
CIFAR-10					
PixelCNN++ (Salimans et al., 2017)	AR	53M*		D	2.92
PixelSNAIL (Chen et al., 2017)	AR			D	2.85
Sparse Transformer (Child et al., 2019)	AR	59M		D	**2.80**
VLAE (Chen et al., 2016)	VAE			D	≤ 2.95
IAF-VAE (Kingma et al., 2016)	VAE		12	1	≤ 3.11
Flow++ (Ho et al., 2019)	Flow	31M		1	≤ 3.08
BIVA (Maaløe et al., 2019)	VAE	103M	15	1	≤ 3.08
NVAE (Vahdat & Kautz, 2020)	VAE	131M	30	1	≤ 2.91
Very Deep VAE **(ours)**	VAE	39M	45	1	$\leq$ **2.87**
ImageNet-32					
Gated PixelCNN	AR	177M*	10	D	3.83
Image Transformer (Parmar et al., 2018)	AR			D	**3.77**
BIVA	VAE	103M*	15	1	≤ 3.96
NVAE	VAE	268M	28	1	≤ 3.92
Flow++	Flow	169M		1	≤ 3.86
Very Deep VAE **(ours)**	VAE	119M	78	1	$\leq$ **3.80**
ImageNet-64					
Gated PixelCNN	AR	177M*		D	3.57
SPN (Menick & Kalchbrenner, 2018)	AR	150M		D	3.52
Sparse Transformer	AR	152M		D	**3.44**
Glow (Kingma & Dhariwal, 2018)	Flow			1	3.81
Flow++	Flow	73M		1	≤ 3.69
Very Deep VAE **(ours)**	VAE	125M	75	1	$\leq$ **3.52**
FFHQ-256 (5 bit)					
NVAE	VAE		36	1	≤ 0.68
Very Deep VAE **(ours)**	VAE	115M	62	1	$\leq$ **0.61**
FFHQ-1024 (8 bit)					
Very Deep VAE **(ours)**	VAE	115M	72	1	$\leq$ **2.42**

Table of Contents

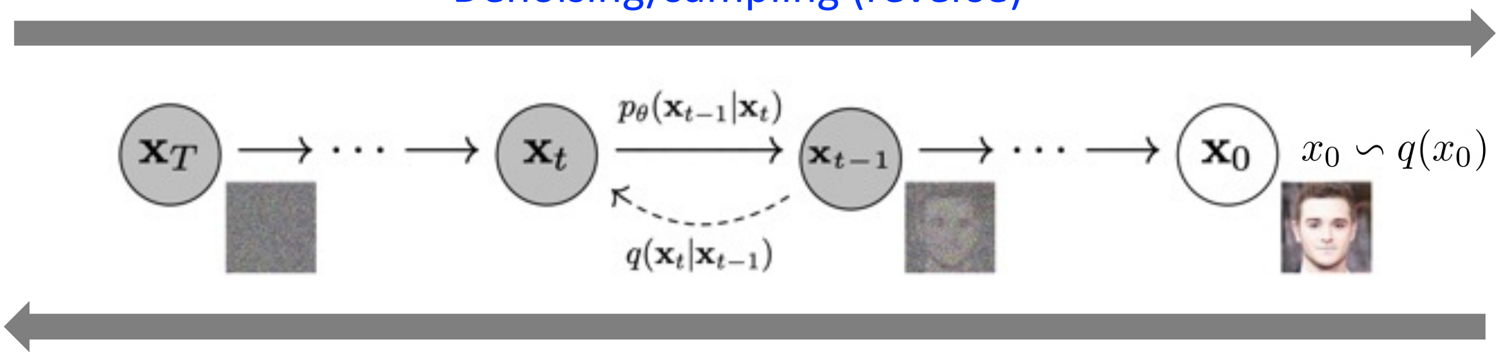
- Diffusion probabilistic models [Sohl-Dickstein et al., 2015]
  - Diffusion (forward) process: Markov chain that **gradually add noise** (of same dimension of data) to data until original the signal is destroyed

$$q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

  - Sampling (backward) process: Markov chain with learned Gaussian **denoising transition**, starting from standard Gaussian noise $p(x_T) = \mathcal{N}(x_T; 0, I)$

$$p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

Denoising/sampling (reverse)



Diffusion process (forward)

- Diffusion probabilistic models [Sohl-Dickstein et al., 2015]
  - Here, the forward distribution $q(x_{t-1}|x_t, x_0)$ can be expressed as a closed form
  - **Variational Lower Bound (VLB)** objective is given by the sum of local KL divergences (between Gaussians)

$$E_q[D_{\mathrm{KL}}(q(x_T|x_0)||p(x_T)) + \underbrace{\sum_{t>1} D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))}^{L_{VLB}} - \log p_\theta(x_0|x_1)]$$

  - Specifically, how to calculate the posterior $q(x_{t-1}|x_t, x_0)$?
    - Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=0}^{t} \alpha_s$, then marginal can be derived:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t)\mathbf{I})$$
$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$$

    - Using Bayes theorem,

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t\mathbf{I})$$

$$\tilde{\beta}_t := \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t$$

$$\tilde{\mu}_t(x_t, x_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}x_t$$

Denoising Diffusion Probabilistic Models (DDPM)

- Diffusion probabilistic models [Sohl-Dickstein et al., 2015]
  - For modelling p_θ, we should model μ_θ and Σ_θ

$$p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1}; \underline{\mu_\theta(x_t, t)}, \underline{\Sigma_\theta(x_t, t)})$$

  - DDPM [Ho et al., 2020] proposes a simple objective & model:
    - For Σ_θ, DDPM fix the variance $\Sigma_\theta(x_t, t) = \sigma_t^2 \boldsymbol{I}$, where $\sigma_t^2 = \beta_t$ or $\tilde{\beta}_t$
    - For μ_θ, DDPM predicts the noise ϵ and use the following derivation:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$$

      - Where ϵ_θ is trained by optimizing following objective:

$$L_{\text{simple}} = E_{t,x_0,\epsilon}\left[||\epsilon - \epsilon_\theta(x_t, t)||^2\right]$$

    - Then, the training/sampling scheme resembles *denoising score matching* (will be discussed later in this lecture)
    - Intuitively, the reverse process adds the (learned) noise ϵ_θ for each step (resembles stochastic Langevin dynamics)

Denoising Diffusion Probabilistic Models (DDPM)

- Diffusion probabilistic models [Sohl-Dickstein et al., 2015]
  - DDPM achieved the SOTA FID score (3.17) on CIFAR-10 generation

  - DDPM also generates high-resolution (256x256) images

- Improved Denoising Diffusion Probabilistic Models [Nichol and Dhariwal, 2021]
  - This paper improves upon DDPM by introducing additional techniques:

1. Learned variance instead of fixed variance

$$\Sigma_\theta(x_t, t) = \exp(v \log \beta_t + (1 - \underline{v}) \log \tilde{\beta}_t)$$

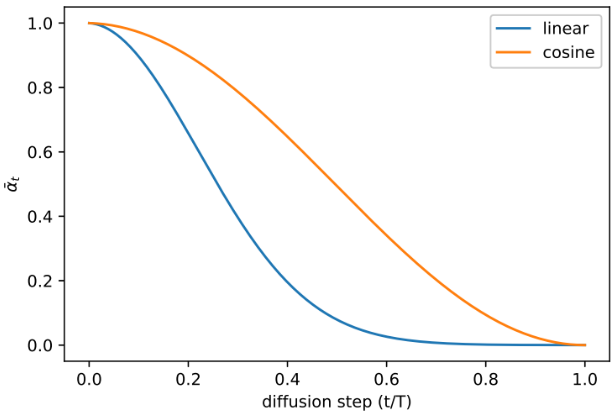
Learnable parameters

2. Hybrid objective of VLB and Simple objectives

$$L_{\text{hybrid}} = L_{\text{simple}} + \lambda \underline{L_{\text{vlb}}}$$

Σ_θ can be learned through this loss

3. Different diffusion (cosine) schedule
   - Instead of linear schedule in DDPM



- Improved Denoising Diffusion Probabilistic Models [Nichol and Dhariwal, 2021]
  - **Results:** Simple techniques can improve performance of DDPM

Table 3. Comparison of DDPMs to other likelihood-based models on CIFAR-10 and Unconditional ImageNet 64×64. NLL is reported in bits/dim. On ImageNet 64×64, our model is competitive with the best convolutional models, but is worse than fully transformer-based architectures.

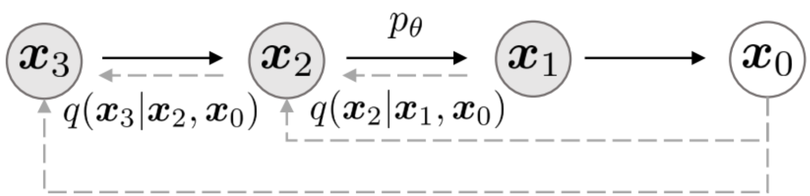
Model	ImageNet	CIFAR
Glow (Kingma & Dhariwal, 2018)	3.81	3.35
Flow++ (Ho et al., 2019)	3.69	3.08
PixelCNN (van den Oord et al., 2016c)	3.57	3.14
SPN (Menick & Kalchbrenner, 2018)	3.52	-
NVAE (Vahdat & Kautz, 2020)	-	2.91
Very Deep VAE (Child, 2020)	3.52	2.87
PixelSNAIL (Chen et al., 2018)	3.52	2.85
Image Transformer (Parmar et al., 2018)	3.48	2.90
Sparse Transformer (Child et al., 2019)	3.44	**2.80**
Routing Transformer (Roy et al., 2020)	**3.43**	-
DDPM (Ho et al., 2020)	3.77	3.70
DDPM (cont flow) (Song et al., 2020b)	-	2.99
Improved DDPM (ours)	**3.53**	**2.94**

- Denoising Diffusion Implicit Models (DDIM) [Song et al., 2021]
  - Generalizes DDPM with much faster sampling process

  - **Main idea:** Introduce non-Markovian forward process
    - Since DDPM objective only depends on marginal $q(x_t|x_0)$, **any arbitrary infere nce distribution that has same marginal** can be used

    - Specifically, DDIM proposes a following inference distribution:

$$q_\sigma(\boldsymbol{x}_t|\underline{\boldsymbol{x}_{t-1}}, \underline{\boldsymbol{x}_0}) = \frac{q_\sigma(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) q_\sigma(\boldsymbol{x}_t|\boldsymbol{x}_0)}{q_\sigma(\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}$$

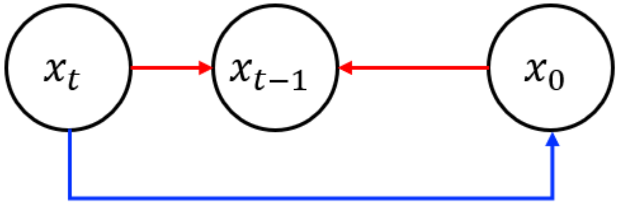


    - Where q_σ is set to have same marginal with DDPM

$$q_\sigma(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) = \mathcal{N}\left(\sqrt{\alpha_{t-1}}\boldsymbol{x}_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \frac{\boldsymbol{x}_t - \sqrt{\alpha_t}\boldsymbol{x}_0}{\sqrt{1-\alpha_t}}, \sigma_t^2 \boldsymbol{I}\right)$$

- Denoising Diffusion Implicit Models (DDIM) [Song et al., 2021]

  - Generative process $p_\theta^{(t)}(x_{t-1}|x_t)$ is defined by leveraging $q_\sigma(x_{t-1}|x_t, x_0)$:
    1. From x_t, predict "denoised" observation x_0
    2. Obtain sample x_{t-1} from $q_\sigma(x_{t-1}|x_t, x_0)$ using predicted x_0 and x_t



  - How to predict x_0 from x_t?
    - **Marginal**: $q(x_t|x_0) \coloneqq \int q(x_{1:t}|x_0)dx_{1:(t-1)} = N(x_t; \sqrt{\alpha_t}x_0, (1 - \alpha_t)\boldsymbol{I})$
    - From this, we can obtain $x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon_t$
    - By introducing a model $\epsilon_\theta^{(t)}(x_t)$ that predicts ϵ_t, prediction of x_0 is given as:

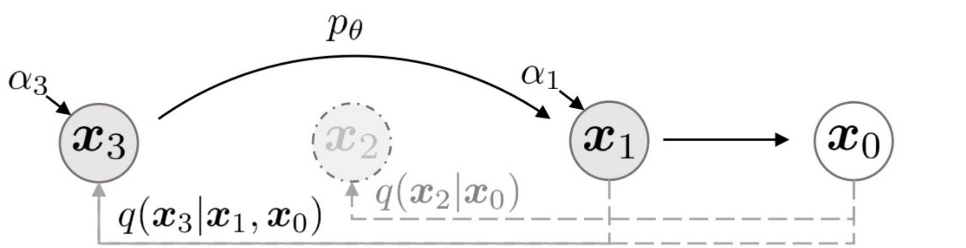
$$f_\theta^{(t)}(x_t) \coloneqq (x_t - \sqrt{1-\alpha_t}\,\epsilon_\theta^{(t)}(x_t))/\sqrt{\alpha_t}$$

- Denoising Diffusion Implicit Models (DDIM) [Song et al., 2021]
  - **Resulting generative process:**

$$p_\theta^{(t)}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \begin{cases} \mathcal{N}(f_\theta^{(1)}(\boldsymbol{x}_1), \sigma_1^2 \boldsymbol{I}) & \text{if } t = 1 \\ q_\sigma(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, f_\theta^{(t)}(\boldsymbol{x}_t)) & \text{otherwise,} \end{cases}$$

$$\boldsymbol{x}_{t-1} = \sqrt{\alpha_{t-1}} \underbrace{\left(\frac{\boldsymbol{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}(\boldsymbol{x}_t)}{\sqrt{\alpha_t}} \right)}_{\text{" predicted } \boldsymbol{x}_0\text{"}} + \underbrace{\sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \epsilon_\theta^{(t)}(\boldsymbol{x}_t)}_{\text{"direction pointing to } \boldsymbol{x}_t\text{"}} + \underbrace{\sigma_t \epsilon_t}_{\text{random noise}}$$

    - Which becomes a DDPM when $\sigma_t = \sqrt{(1-\alpha_{t-1})/(1-\alpha_t)}\sqrt{1-\alpha_t/\alpha_{t-1}}$
    - Which becomes a deterministic generative process from x_t to x_0 when $\sigma_t = 0$
      - i.e., Denoising Diffusion Implicit Model (DDIM)
  - **Accelerated generation process**
    - Objective does not depend on specific forward process if $q_\sigma(x_t|x_0)$ is fixed
    - Hence, we can consider "shorter" forward processes
      - Forward process over a subset $\{x_{\tau_1}, \ldots, x_s\}$ that matches the marginals
      - $q(x_{\tau_i}|x_0) = N(\sqrt{\alpha_{\tau_i}}x_0, (1-\alpha_{\tau_i})\boldsymbol{I})$



- Denoising Diffusion Implicit Models (DDIM) [Song et al., 2021]
  - **Experiments:** Investigation on hyperparameters
    - Number of sampling steps **S**
    - Degree of stochasticity $\boldsymbol{\sigma}$
      - Interpolation between DDPM ($\eta = 1$) and DDIM ($\eta = 0$)

$$\sigma_{\tau_i}(\eta) = \eta\sqrt{(1-\alpha_{\tau_{i-1}})/(1-\alpha_{\tau_i})}\sqrt{1-\alpha_{\tau_i}/\alpha_{\tau_{i-1}}}$$

Table 1: CIFAR10 and CelebA image generation measured in FID. $\eta = 1.0$ and $\hat{\sigma}$ are cases of DDPM (although Ho et al. (2020) only considered $T = 1000$ steps, and $S < T$ can be seen as simulating DDPMs trained with S steps), and $\eta = 0.0$ indicates DDIM.

		CIFAR10 (32 × 32)					CelebA (64 × 64)				
	S	10	20	50	100	1000	10	20	50	100	1000
η	0.0	**13.36**	**6.84**	**4.67**	**4.16**	4.04	**17.33**	**13.73**	**9.17**	**6.53**	3.51
	0.2	14.04	7.11	4.77	4.25	4.09	17.66	14.11	9.51	6.79	3.64
	0.5	16.66	8.35	5.25	4.46	4.29	19.86	16.06	11.01	8.09	4.28
	1.0	41.07	18.36	8.01	5.78	4.73	33.12	26.03	18.48	13.93	5.98

Deterministic generation process (DDIM) can generate good samples with 10x~100x smaller sampling steps (=fast)

- Diffusion Models Beat GANs on Image Synthesis [Dhariwal and Nichol, 2021]
  - **Main idea**: Class information could improve the image fidelity of diffusion model.
    - Class-conditional GANs already make heavy use of class information [Brock et al., 2019].
  - Propose classifier guidance to give class information to the diffusion model.
    - Trade off between the fidelity and diversity of generated image.
    - As strong class guidance is given, the fidelity of images improves but the diversity decreases.
  - Train a classifier $p_\phi(y|x_t, t)$ on noisy images x_t, and use gradients $\nabla_{x_t} \log p_\phi(y|x_t, t)$ to guide the diffusion sampling process.

- Diffusion Models Beat GANs on Image Synthesis [Dhariwal and Nichol, 2021]

  - **Classifier guided diffusion sampling** (as DDPM)
    - Substitute the denoising process to conditional likelihood $p_{\theta,\phi}(x_t|x_{t+1},y)$

      $$p_{\theta,\phi}(x_t|x_{t+1},y) = Zp_\theta(x_t|x_{t+1})p_\phi(y|x_t)$$

      where Z is a normalizing constant.
    - By approximating $\log p_\phi(y|x_t)$ using a Talyor expansion,

      $$\begin{aligned}\log(p_\theta(x_t|x_{t+1})p_\phi(y|x_t)) &\approx -\frac{1}{2}(x_t-\mu-\Sigma g)^T\Sigma^{-1}(x_t-\mu-\Sigma g)+C_3 \\ &= \log p(z)+C_4, z\sim\mathcal{N}(\mu+\Sigma g,\Sigma)\end{aligned}$$

      Here, $g = \nabla_{x_t}\log p_\phi(y|x_t)|_{x_t=\mu}$ which is a gradient from the classifer.

    - Conditional transition could be approximated by a Gaussian similar to unconditional transition operator with shifted mean.

- Diffusion Models Beat GANs on Image Synthesis [Dhariwal and Nichol, 2021]
  - **Classifier guided DDIM sampling**
    - Score function is derived from the noise prediction model $\epsilon_\theta(x_t)$:

$$\nabla_{x_t} \log p_\theta(x_t) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t)$$

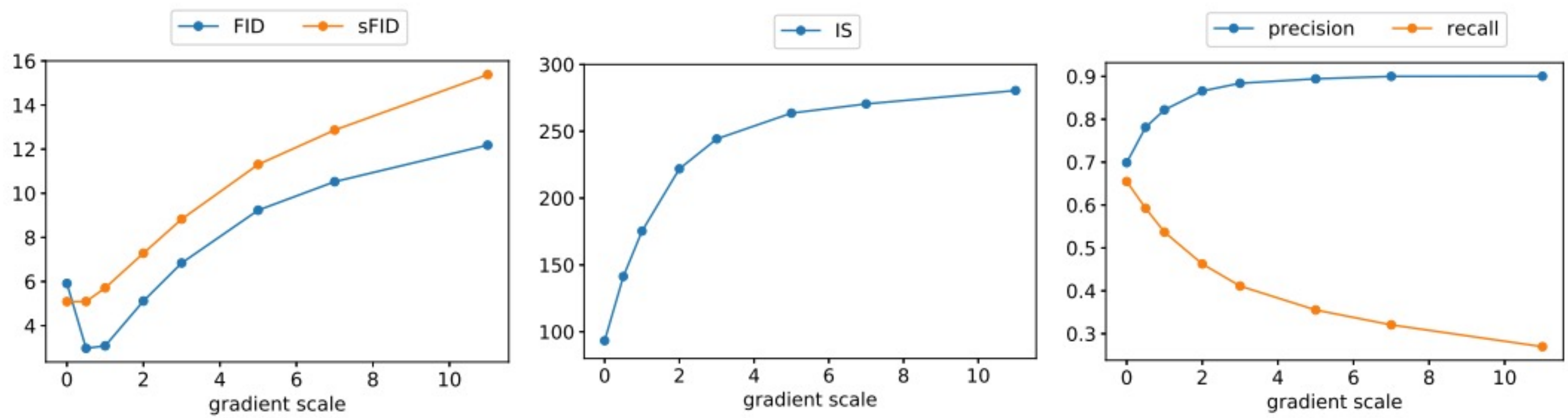
    - Score function for conditional generation is given by,

$$\begin{aligned}\nabla_{x_t} \log(p_\theta(x_t)p_\phi(y|x_t)) &= \nabla_{x_t} \log p_\theta(x_t) + \nabla_{x_t} \log p_\phi(y|x_t) \\ &= -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t) + \nabla_{x_t} \log p_\phi(y|x_t)\end{aligned}$$

    - Hence, we could use the modified noise prediction for the same procedure.

$$\hat{\epsilon}(x_t) := \epsilon_\theta(x_t) - \sqrt{1-\bar{\alpha}_t}\,\nabla_{x_t} \log p_\phi(y|x_t)$$

- Diffusion Models Beat GANs on Image Synthesis [Dhariwal and Nichol, 2021]
  - Trade off between fidelity and diversity.
    - As scaling classifier gradients, metrics indicate quality (IS and precision) improv es and metrics imply diversity (FID and recall) become worse.



- Diffusion Models Beat GANs on Image Synthesis [Dhariwal and Nichol, 2021]
  - Difusion model guided with classifier outperforms the state of the art generative models.

Model	FID	sFID	Prec	Rec
LSUN Bedrooms 256×256				
DCTransformer† [42]	6.40	6.66	0.44	**0.56**
DDPM [25]	4.89	9.07	0.60	0.45
IDDPM [43]	4.24	8.21	0.62	0.46
StyleGAN [27]	2.35	6.62	0.59	0.48
ADM (dropout)	**1.90**	**5.59**	**0.66**	0.51
LSUN Horses 256×256				
StyleGAN2 [28]	3.84	6.46	0.63	0.48
ADM	2.95	**5.94**	0.69	**0.55**
ADM (dropout)	**2.57**	6.81	**0.71**	**0.55**
LSUN Cats 256×256				
DDPM [25]	17.1	12.4	0.53	0.48
StyleGAN2 [28]	7.25	**6.33**	0.58	0.43
ADM (dropout)	**5.57**	6.69	**0.63**	**0.52**
ImageNet 64×64				
BigGAN-deep* [5]	4.06	3.96	**0.79**	0.48
IDDPM [43]	2.92	**3.79**	0.74	0.62
ADM	2.61	**3.77**	0.73	0.63
ADM (dropout)	**2.07**	4.29	0.74	**0.63**

Model	FID	sFID	Prec	Rec
ImageNet 128×128				
BigGAN-deep [5]	6.02	7.18	**0.86**	0.35
LOGAN† [68]	3.36			
ADM	5.91	**5.09**	0.70	**0.65**
ADM-G (25 steps)	5.98	7.04	0.78	0.51
ADM-G	**2.97**	**5.09**	0.78	0.59
ImageNet 256×256				
DCTransformer† [42]	36.51	8.24	0.36	**0.67**
VQ-VAE-2†‡ [51]	31.11	17.38	0.36	0.57
IDDPM‡ [43]	12.26	5.42	0.70	0.62
SR3†‡ [53]	11.30			
BigGAN-deep [5]	6.95	7.36	**0.87**	0.28
ADM	10.94	6.02	0.69	0.63
ADM-G (25 steps)	5.44	5.32	0.81	0.49
ADM-G	**4.59**	**5.25**	0.82	0.52
ImageNet 512×512				
BigGAN-deep [5]	8.43	8.13	**0.88**	0.29
ADM	23.24	10.19	0.73	**0.60**
ADM-G (25 steps)	8.41	9.67	0.83	0.47
ADM-G	**7.72**	**6.57**	0.87	0.42

- Classifier-Free Diffusion Guidance [Ho and Salimans, 2021]
  - Introduce classifier guidance without training any additional classifier.
  - Train a single neural network which **parameterize both conditional and unconditional diffusion model**.
    - Conditional model $p_\theta(\boldsymbol{z}|\boldsymbol{c})$ with score $\epsilon_\theta(\boldsymbol{z}_\lambda, \boldsymbol{c})$
    - Unconditional diffusion model $p_\theta(\boldsymbol{z})$ with score $\epsilon_\theta(\boldsymbol{z}_\lambda) = \epsilon_\theta(\boldsymbol{z}_\lambda, \boldsymbol{c} = \boldsymbol{0})$
  - Perform sampling using the linear combination of conditional and unconditional score estimates.

$$\tilde{\boldsymbol{\epsilon}}_\theta(\mathbf{z}_\lambda, \mathbf{c}) = (1+w)\boldsymbol{\epsilon}_\theta(\mathbf{z}_\lambda, \mathbf{c}) - w\boldsymbol{\epsilon}_\theta(\mathbf{z}_\lambda)$$

    - Inspired by the gradient of an implicit classifier $p^i(\boldsymbol{c}|\boldsymbol{z}_\lambda) \propto p(\boldsymbol{z}_\lambda|\boldsymbol{c})/p(\boldsymbol{z}_\lambda)$.
    - If exact score $\boldsymbol{\epsilon}^*(\boldsymbol{z}_\lambda, \boldsymbol{c}), \boldsymbol{\epsilon}^*(\boldsymbol{z}_\lambda)$ exists, then the gradient of the implicit classifier is given by $\nabla_{\mathbf{z}_\lambda} \log p^i(\mathbf{c}|\mathbf{z}_\lambda) = -\frac{1}{\sigma_\lambda}[\boldsymbol{\epsilon}^*(\mathbf{z}_\lambda, \mathbf{c}) - \boldsymbol{\epsilon}^*(\mathbf{z}_\lambda)]$
    - Its guidance would be linear interpolation between two score functions.

- Classifier-Free Diffusion Guidance [Ho and Salimans, 2021]
  - Suggesting higher guidance generates less diverse and high fidelity images.

Method	FID (↓)	IS (↑)
ADM [3]	2.07	-
CDM [6]	**1.48**	67.95
Ours, no guidance	1.80	53.71
Ours, with guidance		
$w = 0.1$	1.55	66.11
$w = 0.2$	2.04	78.91
$w = 0.3$	3.03	92.8
$w = 0.4$	4.30	106.2
$w = 0.5$	5.74	119.3
$w = 0.6$	7.19	131.1
$w = 0.7$	8.62	141.8
$w = 0.8$	10.08	151.6
$w = 0.9$	11.41	161
$w = 1.0$	12.6	170.1
$w = 2.0$	21.03	225.5
$w = 3.0$	24.83	250.4
$w = 4.0$	26.22	**260.2**

Figure 1: ImageNet 64x64 results

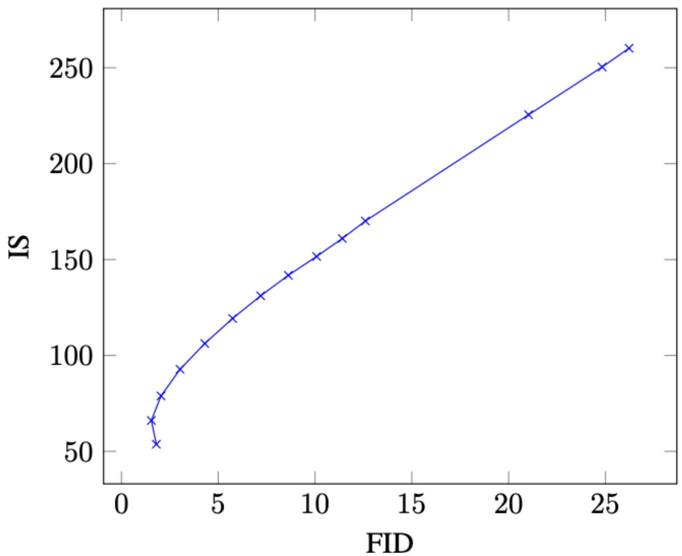


Figure 2: ImageNet 64x64 FID vs. IS

- GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models [Nichol et al., 2022]
  - Text conditional image synthesis with classifier-free guidance to diffusion model.

  - **How to input text as the condition?**
    - Encode the text into a sequence of K tokens and feed theses tokens into a Transformer model.
    - Final token embedding is used in place of a class embedding.
    - The last layer of token embeddings is separately projected to the dimensionality of each attention layer, and then concatenated to the attention context.

- GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided D iffusion Models [Nichol et al., 2022]
    - Text conditional image synthesis with classifier-free guidance to diffusion model.

"a hedgehog using a calculator"

"a corgi wearing a red bowtie and a purple party hat"

"robots meditating in a vipassana retreat"

"a fall landscape with a small cottage next to a lake"

"a surrealist dream-like oil painting by salvador dalí of a cat playing checkers"

"a professional photo of a sunset behind the grand canyon"

"a high-quality oil painting of a psychedelic hamster dragon"

"an illustration of albert einstein wearing a superhero costume"

Table of Contents

- EBM [LeCun et al., 2006, Du & Mordatch, 2019]
  - Instead of directly modeling the density $p(x)$, learn the unnormalized density (i.e., energy) $E(x)$ such that

$$p_\theta(x) = \frac{\exp(-E_\theta(x))}{Z_\theta}, \quad Z_\theta = \int_{x \in \mathcal{X}} \exp(-E_\theta(x))$$

  - Here, we don't care about the **exact density** (which needs to compute the partition function Z_θ), but only interested in the **relative order** of densities

  - **Training:** The gradient of negative log-likelihood (NLL) is decomposed to:

$$\begin{aligned}\mathbb{E}_{x \sim p_{\text{data}}(x)}[-\nabla_\theta \log p_\theta(x)] &= \mathbb{E}_{x \sim p_{\text{data}}(x)}[\nabla_\theta E_\theta(x)] + \nabla_\theta \log Z_\theta \\ &= \underbrace{\mathbb{E}_{x \sim p_{\text{data}}(x)}[\nabla_\theta E_\theta(x)]}_{\text{data gradient}} - \underbrace{\mathbb{E}_{x' \sim p_\theta(x)}[\nabla_\theta E_\theta(x')]}_{\text{model gradient}}\end{aligned}$$

  - Note that this **contrastive** objective resembles (Wasserstein) GAN, but EBM uses an implicit MCMC generating procedure and no gradient through sampling
    - One can modify the discriminator of GAN to be an EBM [Zhao et al., 2017]

- EBM [LeCun et al., 2006, Du & Mordatch, 2019]
  - Instead of directly modeling the density $p(x)$, learn the unnormalized density (i.e., energy) $E(x)$ such that

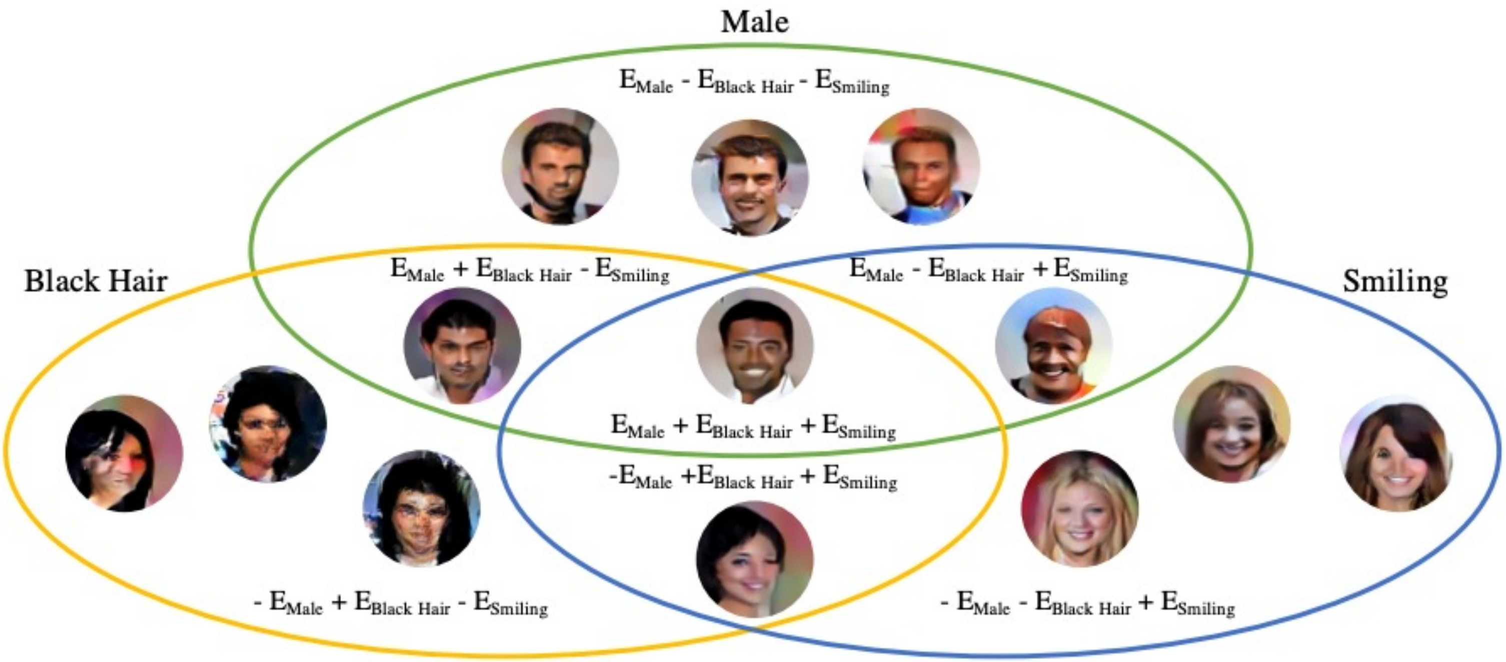
$$p_\theta(x) = \frac{\exp(-E_\theta(x))}{Z_\theta}, \quad Z_\theta = \int_{x \in \mathcal{X}} \exp(-E_\theta(x))$$

  - **Sampling:** Run Markov chain Monte Carlo (MCMC) to draw a sample from $p_\theta(x)$
    - For high-dimensional data (e.g., image generation), stochastic gradient Langevin dynamics (SGLD) [Welling & Teh, 2011] is popularly used:
      - Given an initial sample x^0, iteratively update x^{k+1} $(k = 0, \ldots, K-1)$

$$x^{k+1} \leftarrow x^k + \frac{\alpha}{2} \underbrace{\nabla_x \log p_\theta(x^k)}_{-\nabla_x E_\theta(x)} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \alpha)$$

      - Due to the Gaussian noise, it does not collapse to the MAP solution but converges to $p_\theta(x)$ as $\alpha \to 0$ and $K \to \infty$

- **Advantages of EBMs**
  1. Compositionality: One can add or subtract multiple energy functions (e.g., male, black hair, smiling) to sample the composite distribution



  2. No generator network: Unlike GAN/VAEs, EBMs do not need a specialized generator architecture (one can reuse the standard classifier architectures)
  3. Adaptive computation time: Since the sampling is given by iterative SGLD, the user can choose from the fast coarse samples to slow fine samples

- EBM [LeCun et al., 2006, Du & Mordatch, 2019]
  - The gradient of partition function can be reformulated as follow:

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \log Z_{\boldsymbol{\theta}} &= \nabla_{\boldsymbol{\theta}} \log \int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \\
&\overset{(i)}{=} \left(\int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \right)^{-1} \nabla_{\boldsymbol{\theta}} \int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \\
&= \left(\int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \right)^{-1} \int \nabla_{\boldsymbol{\theta}} \exp(-E_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \\
&\overset{(ii)}{=} \left(\int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \right)^{-1} \int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))(-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \\
&= \int \left(\int \exp(-E_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \right)^{-1} \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))(-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \\
&\overset{(iii)}{=} \int \frac{\exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))}{Z_{\boldsymbol{\theta}}} (-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \\
&\overset{(iv)}{=} \int p_{\boldsymbol{\theta}}(\mathbf{x})(-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \\
&= \mathbb{E}_{\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x})} \left[-\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x}) \right],
\end{aligned}
$$

- JEM [Grathwohl et al., 2020]
  - Use standard classifier architectures for *joint distribution* EBMs
  - Recall that the classifier $p_\theta(y|x)$ is expressed by the logits $f_\theta(x)$

$$p_\theta(y|x) = \frac{\exp(f_\theta(x)[y])}{\sum_{y'} \exp(f_\theta(x)[y'])}$$

  - Here, one can re-interpret the logits to define an energy-based model

$$p_\theta(x,y) = \frac{\exp(f_\theta(x)[y])}{Z_\theta}, \quad p_\theta(x) = \frac{\sum_y \exp(f_\theta(x)[y])}{Z_\theta}$$

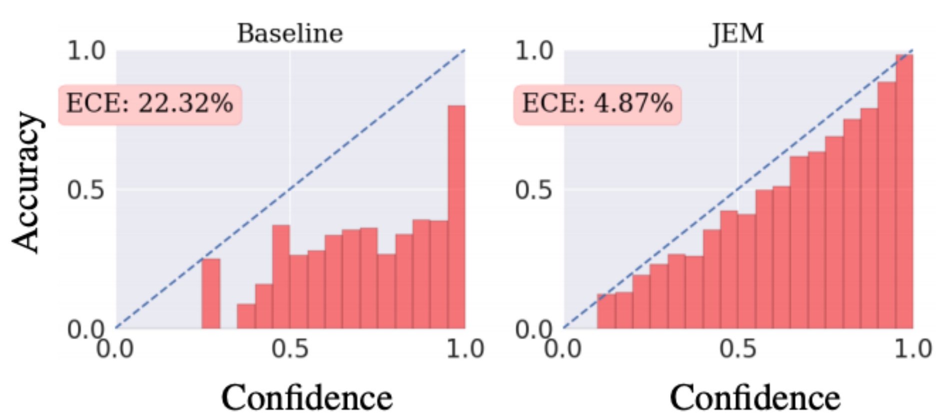
  - Note that shifting the logits does not affect $p_\theta(y|x)$ but $p_\theta(x)$; hence, EBM gives an extra degree of freedom
  - The objective of JEM is a **sum** of density and conditional models, where the density model is trained by contrastive objective of EBM

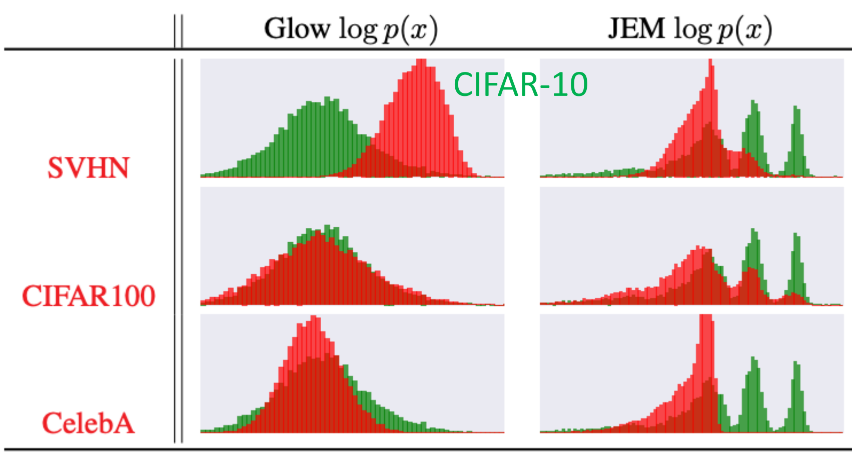
$$\log p_\theta(x,y) = \log p_\theta(x) + \log p_\theta(y|x)$$

- JEM [Grathwohl et al., 2020]
  - JEM achieves a competitive performance as both classifier and generative model

Class	Model	Accuracy% ↑	IS↑	FID↓
Hybrid	Residual Flow	70.3	3.6	46.4
	Glow	67.6	3.92	48.9
	IGEBM	49.1	8.3	**37.9**
	JEM $p(\mathbf{x}\|y)$ factored	30.1	6.36	61.8
	JEM (Ours)	**92.9**	**8.76**	38.4
Disc.	Wide-Resnet	95.8	N/A	N/A
Gen.	SNGAN	N/A	8.59	25.5
	NCSN	N/A	8.91	25.32

  - Also, JEM (generative classifier) improves uncertainty and robustness
    - (a) calibration, (b) out-of-distribution detection, (c) adversarial robustness





- Score matching [Hyvärinen, 2005]
  - Score = gradient of the log-likelihood $s(x) \coloneqq \nabla_x \log p(x)$
  - Score matching = Match the *scores* of data and model distribution
    - However, we **don't know** the scores of data distribution
    - Instead, one can use the **equivalent form** (proof by integration of parts)

$$\frac{1}{2}\mathbb{E}_{x\sim p_{\texttt{data}}(x)}[\|s_\theta(x) - s_{\texttt{data}}(x)\|_2^2] = \mathbb{E}_{x\sim p_{\texttt{data}}(x)}\left[\mathrm{tr}(\nabla_x s_\theta(x)) + \frac{1}{2}\|s_\theta(x)\|_2^2\right] + \mathrm{const.}$$

🤔 😀

- Recent works mostly consider **denoising score matching** [Vincent, 2011]
  - Match the score of **perturbed distribution** $q_\sigma(\tilde{x}) \coloneqq \int q_\sigma(\tilde{x}|x)\, p_{\mathrm{data}}(x)$ where $q_\sigma(\tilde{x}|x) = \mathcal{N}(x, \sigma)$
  - Then, the score matching objective is **equivalent** to

$$\frac{1}{2}\mathbb{E}_{\tilde{x}\sim q_\sigma(\tilde{x}|x)p_{\texttt{data}}(x)}[\|s_\theta(\tilde{x}) - \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x)\|_2^2]$$

  - It is tractable since the gradient $\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) = \nabla_{\tilde{x}} \log \mathcal{N}(\tilde{x}|x, \sigma) = \nabla_{\tilde{x}} \log \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{1}{2}(\frac{\tilde{x}-x}{\sigma})^2)$ can be **analytically computed**
  - The objective can learn the scores of data distribution if $\sigma \approx 0$

- Score matching [Hyvärinen, 2005]
  - The score matching objective can be reformulated as follow:

$$\frac{1}{2}\mathbb{E}_{x\sim p_{\texttt{data}}(x)}[\|s_\theta(x)-s_{\texttt{data}}(x)\|_2^2]=\mathbb{E}_{x\sim p_{\texttt{data}}(x)}\left[\mathrm{tr}(\nabla_x s_\theta(x))+\frac{1}{2}\|s_\theta(x)\|_2^2\right]+\text{const.}$$

  - It is sufficient to show that

$$\begin{aligned}\mathbb{E}_{p_{\texttt{data}}(x)}[-s_{\texttt{data}}(x)s_\theta(x)]&=\sum_i\int -p_{\texttt{data}}(x)\frac{\partial\log p_{\texttt{data}}(x)}{dx_i}s_{\theta,i}(x)dx\\&=\sum_i\int -\frac{\partial p_{\texttt{data}}(x)}{dx_i}s_{\theta,i}(x)dx\\&=\sum_i\int p_{\texttt{data}}(x)\frac{\partial s_{\theta,i}(x)}{dx_i}dx+\text{const.}\end{aligned}$$

  - The last equality comes from the integration of parts

$$\int p'(x)f(x)dx=p(x)f(x)\Big|_{-\infty}^{\infty}-\int p(x)f'(x)dx$$

and assumption $p_{\text{data}}(x)s_\theta(x)\to 0$ for both side of infinity

- NCSN [Song et al., 2019]
  - Previous works mostly define the score as a gradient of the **energy function** $s_\theta(x) \coloneqq -\nabla_x E_\theta(x)$
  - This work: **Directly model** the score $x \in \mathbb{R}^d \mapsto s_\theta(x) \in \mathbb{R}^d$ as an output

  - **Noise-conditional Score Network**
    - Denoising score matching is stable for large σ but unbiased for small σ
    - **Idea:** Learn multiple noise levels (with a single neural network) and anneal the noise level during sampling $\sigma_1 > \cdots > \sigma_L$

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.
Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
    $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ $\quad \triangleright$ α_i is the step size.
    for $t \leftarrow 1$ to T **do**
        Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
        $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$
    end for
    $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
end for
return $\tilde{\mathbf{x}}_T$

  - One can extend score matching to **continuous version** (stochastic differential equations, SDEs) [Song et al., 2021]
    - NCSN and DDPM can be viewed as different discretization of some SDEs
    - This view provides a better approach for **generation** and **likelihood estimation**

See Appendix for details

- NCSN [Song et al., 2019]
  - The continuous version of NCSN [Song et al., 2021] is SOTA for both **likelihood estimation** and **sample generation** on CIFAR-10

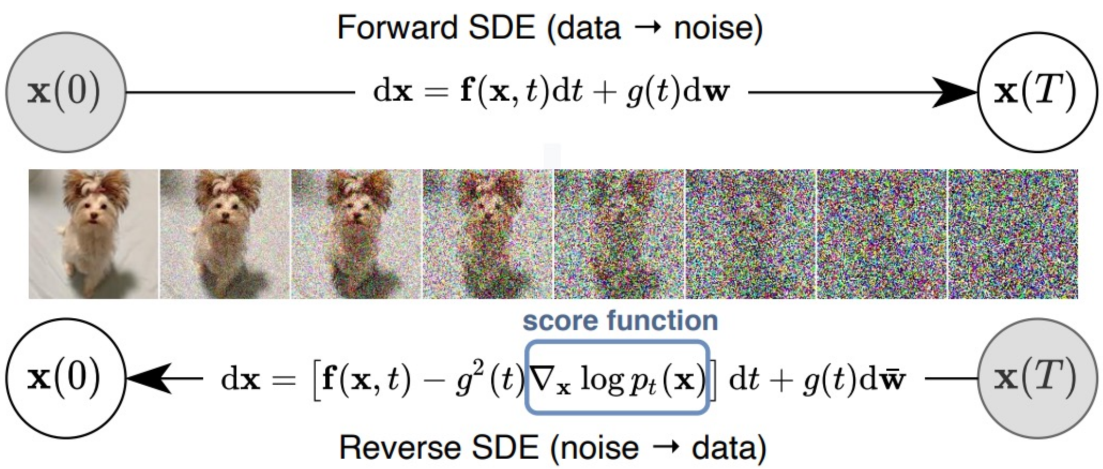
Table 2: NLLs and FIDs (ODE) on CIFAR-10.

Model	NLL Test ↓	FID ↓
RealNVP (Dinh et al., 2016)	3.49	-
iResNet (Behrmann et al., 2019)	3.45	-
Glow (Kingma & Dhariwal, 2018)	3.35	-
MintNet (Song et al., 2019b)	3.32	-
Residual Flow (Chen et al., 2019)	3.28	46.37
FFJORD (Grathwohl et al., 2018)	3.40	-
Flow++ (Ho et al., 2019)	3.29	-
DDPM (L) (Ho et al., 2020)	$\leqslant 3.70^*$	13.51
DDPM (L_{simple}) (Ho et al., 2020)	$\leqslant 3.75^*$	3.17
DDPM	3.28	3.37
DDPM cont. (VP)	3.21	3.69
DDPM cont. (sub-VP)	3.05	3.56
DDPM++ cont. (VP)	3.16	3.93
DDPM++ cont. (sub-VP)	3.02	3.16
DDPM++ cont. (deep, VP)	3.13	3.08
DDPM++ cont. (deep, sub-VP)	**2.99**	**2.92**

Table 3: CIFAR-10 sample quality.

Model	FID↓	IS↑
Conditional		
BigGAN (Brock et al., 2018)	14.73	9.22
StyleGAN2-ADA (Karras et al., 2020a)	**2.42**	**10.14**
Unconditional		
StyleGAN2-ADA (Karras et al., 2020a)	2.92	9.83
NCSN (Song & Ermon, 2019)	25.32	8.87 ± .12
NCSNv2 (Song & Ermon, 2020)	10.87	8.40 ± .07
DDPM (Ho et al., 2020)	3.17	9.46 ± .11
DDPM++	2.78	9.64
DDPM++ cont. (VP)	2.55	9.58
DDPM++ cont. (sub-VP)	2.61	9.56
DDPM++ cont. (deep, VP)	2.41	9.68
DDPM++ cont. (deep, sub-VP)	2.41	9.57
NCSN++	2.45	9.73
NCSN++ cont. (VE)	2.38	9.83
NCSN++ cont. (deep, VE)	**2.20**	**9.89**

- Score matching through SDE [Song et al., 2021]



  - Like DDPM, we consider some forward diffusion process (SDE):

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}},$$

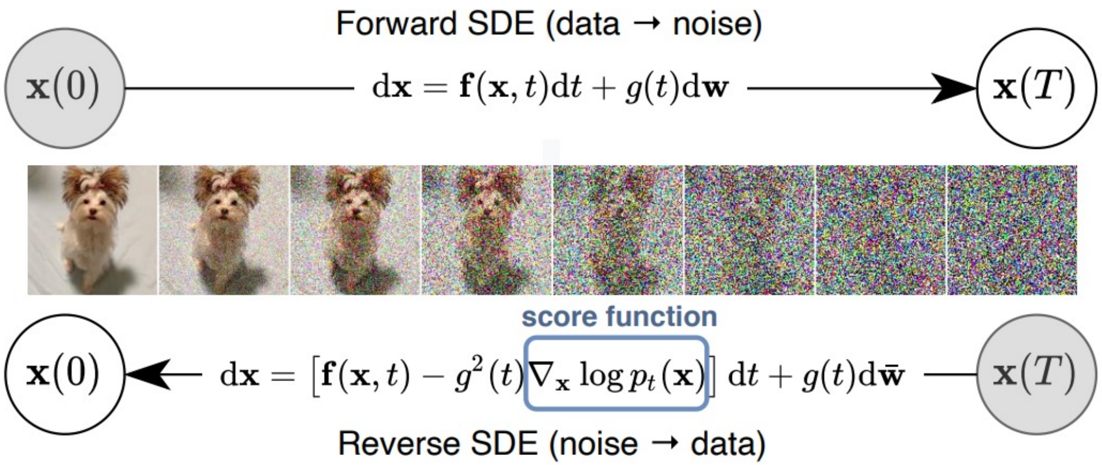
  - Then, the reverse diffusion process also follows some SDE:

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}},$$

  - One can learn the score function by score matching

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_t \Big\{ \lambda(t) \mathbb{E}_{\mathbf{x}(0)} \mathbb{E}_{\mathbf{x}(t) \mid \mathbf{x}(0)} \Big[\big\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) - \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0)) \big\|_2^2 \Big] \Big\}.$$

- Score matching through SDE [Song et al., 2021]



- Like DDPM, we consider some forward diffusion process (SDE):

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}},$$

- Here, NCSN and DDPM can be viewed as different discretizations some stochastic differential equations (SDEs)

  - NCSN: $\mathrm{d}\mathbf{x} = \sqrt{\frac{\mathrm{d}\left[\sigma^2(t)\right]}{\mathrm{d}t}}\mathrm{d}\mathbf{w}$ $\rightarrow$ $\mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2}\mathbf{z}_i$
  - DDPM: $\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\,\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}\mathbf{w}$ $\rightarrow$ $\mathbf{x}_i = \sqrt{1-\beta_i}\mathbf{x}_{i-1} + \sqrt{\beta_i}\mathbf{z}_i$

- Score matching through SDE [Song et al., 2021]
  - The reverse diffusion process can be solved by **3 ways**:

1. Run a general-purpose SDE solver (a.k.a. predictor)
2. Utilize the score-based model $s_\theta(x, t) \approx \nabla_x \log p_t(x)$ (a.k.a. corrector)

→ Combining predictor and corrector gives the **SOTA generation** performance

Algorithm 2 PC sampling (VE SDE)

$\mathbf{x}_N \sim \mathcal{N}(\mathbf{0}, \sigma_{\max}^2 \mathbf{I})$
for $i = N-1$ **to** 0 **do**
  $\mathbf{x}_i' \leftarrow \mathbf{x}_{i+1} + (\sigma_{i+1}^2 - \sigma_i^2)\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_{i+1}, \sigma_{i+1})$
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_i \leftarrow \mathbf{x}_i' + \sqrt{\sigma_{i+1}^2 - \sigma_i^2}\,\mathbf{z}$
  for $j = 1$ **to** M **do**
    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \epsilon_i \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i, \sigma_i) + \sqrt{2\epsilon_i}\,\mathbf{z}$
return $\mathbf{x}_0$

Continuous ver. of NCSN

Algorithm 3 PC sampling (VP SDE)

$\mathbf{x}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $i = N-1$ **to** 0 **do**
  $\mathbf{x}_i' \leftarrow (2 - \sqrt{1 - \beta_{i+1}})\mathbf{x}_{i+1} + \beta_{i+1}\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_{i+1}, i+1)$
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_i \leftarrow \mathbf{x}_i' + \sqrt{\beta_{i+1}}\,\mathbf{z}$
  for $j = 1$ **to** M **do**
    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \epsilon_i \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i, i) + \sqrt{2\epsilon_i}\,\mathbf{z}$
return $\mathbf{x}_0$

Predictor

Corrector

Continuous ver. of DDPM

- Score matching through SDE [Song et al., 2021]
  - The reverse diffusion process can be solved by **3 ways**:

1. Run a general-purpose SDE solver (a.k.a. predictor)
2. Utilize the score-based model $s_\theta(x,t) \approx \nabla_x \log p_t(x)$ (a.k.a. corrector)
3. Convert to deterministic ODE
   - Every SDE (Ito process) has a corresponding deterministic ODE

$$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x},t) - \frac{1}{2}g(t)^2\nabla_{\mathbf{x}}\log p_t(\mathbf{x})\right]\mathrm{d}t,$$

   whose trajectories include the same evolution of densities

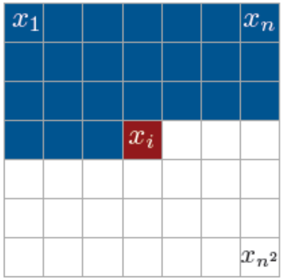
   - Deterministic ODE defines an invertible model (a.k.a. normalizing flow) [Chen et al., 2018]
   - Using this formulation, one can
     a) Compute exact likelihood
     b) Manipulate latents with encoder (model is invertible)

Table of Contents

Table of Contents

- Autoregressive generation (e.g., pixel-by-pixel for images) :

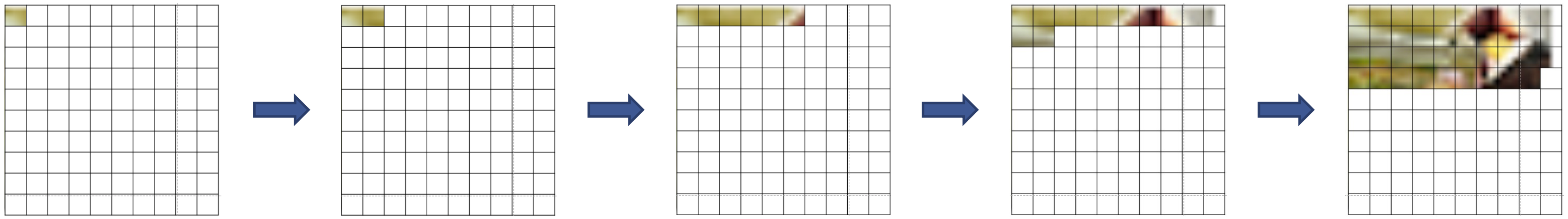
$$p(\boldsymbol{x}) = \prod_{k=1}^{K^2} p(x_k | x_1, \cdots, x_{k-1})$$
$$= \prod_{k=1}^{K^2} p(x_k | \boldsymbol{x}_{<k})$$



- For example, each RBG pixel is generated autoregressively:

$$p(x_k | \boldsymbol{x}_{<k}) = p(x_{k,R}, x_{k,B}, x_{k,G} | \boldsymbol{x}_{<k})$$
$$= p(x_{k,R} | \boldsymbol{x}_{<k}) p(x_{k,B} | \boldsymbol{x}_{<k}, x_{k,R}) p(x_{k,G} | \boldsymbol{x}_{<k}, x_{k,R}, x_{k,B})$$

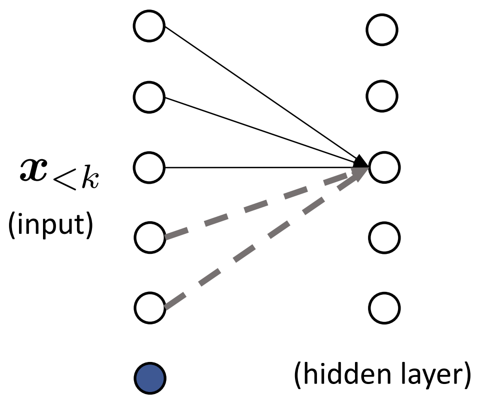
- Each pixel is treated as discrete variables, sampled from softmax distributions:



Pixel Convolutional/Recurrent Neural Network (PixelCNN/PixelRNN)

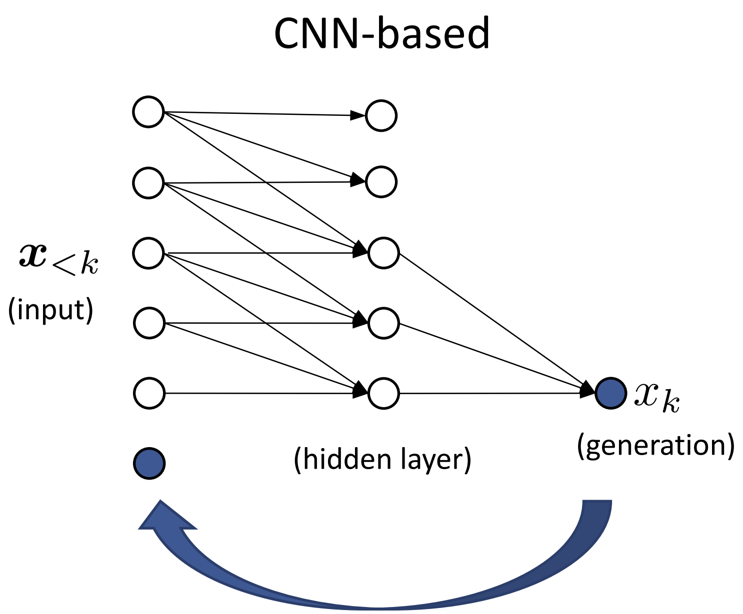
- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Simply treating $\boldsymbol{x}_{<k}$ as one-dimensional (instead of two-dimensional) vector:

CNN-based



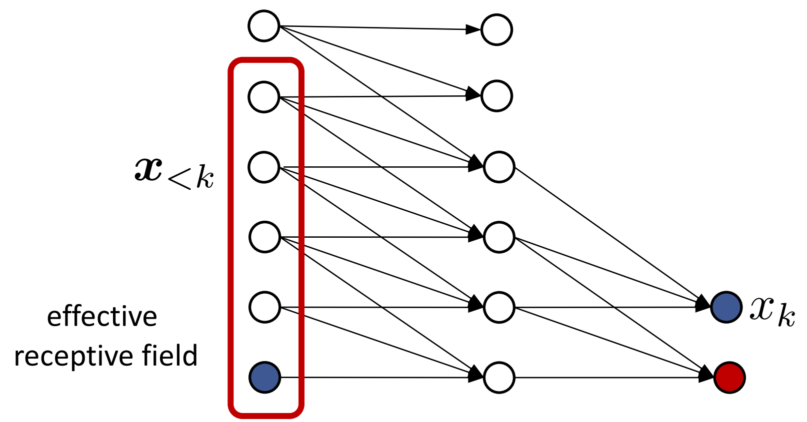
masked convolution

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Simply treating $\boldsymbol{x}_{<k}$ as one-dimensional (instead of two-dimensional) vector:



- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Simply treating $\boldsymbol{x}_{<k}$ as one-dimensional (instead of two-dimensional) vector:

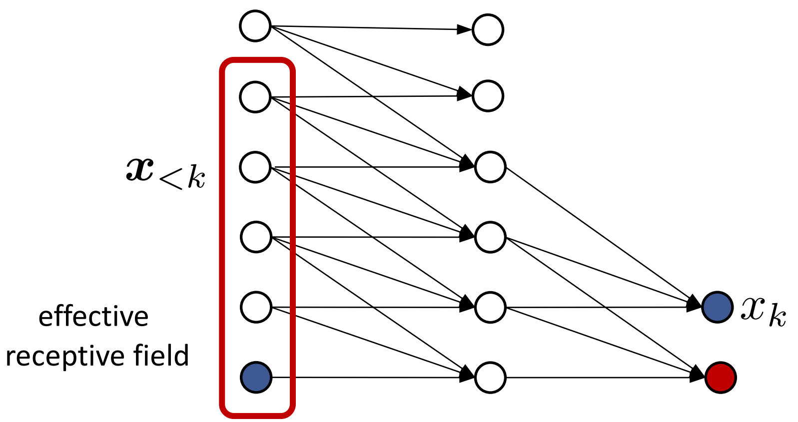
CNN-based



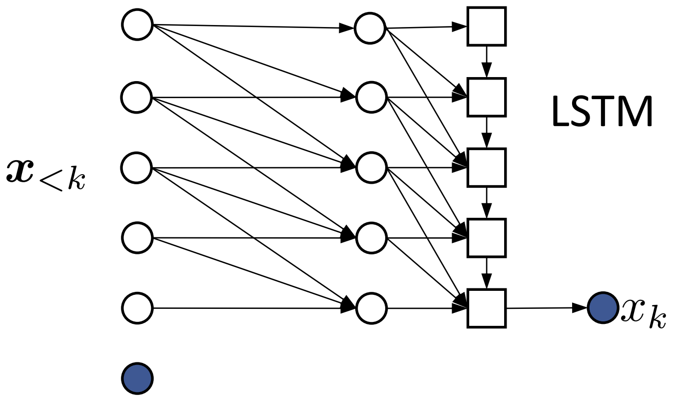
Pixel Convolutional/Recurrent Neural Network (PixelCNN/PixelRNN)

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Simply treating $\boldsymbol{x}_{<k}$ as one-dimensional (instead of two-dimensional) vector:

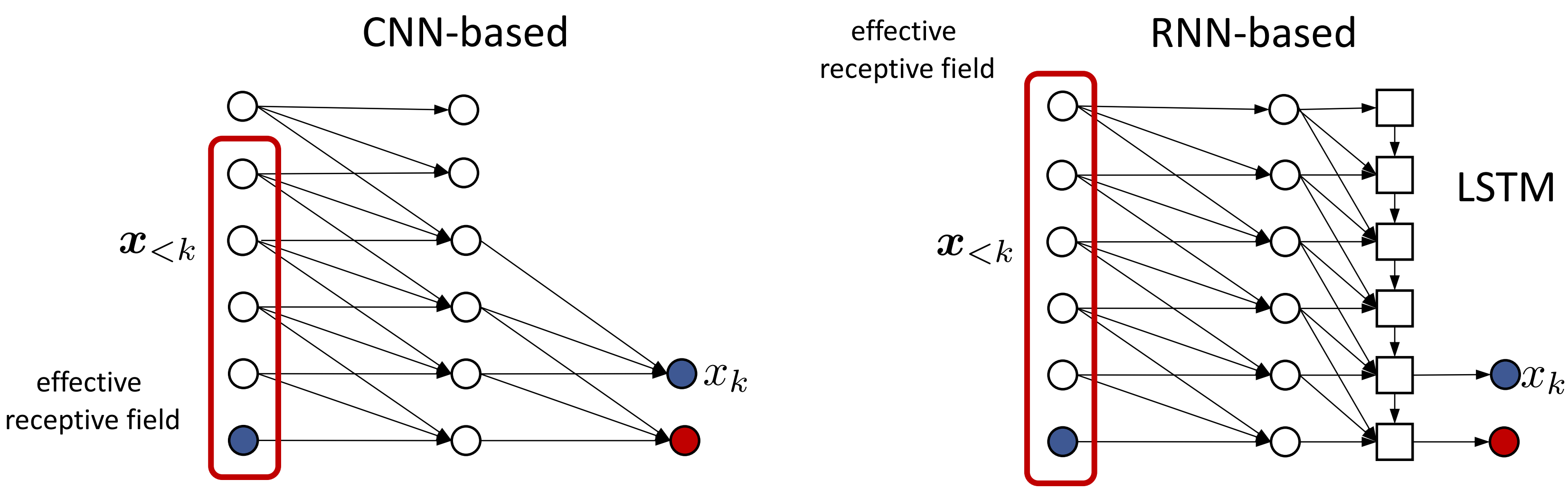
CNN-based



RNN-based



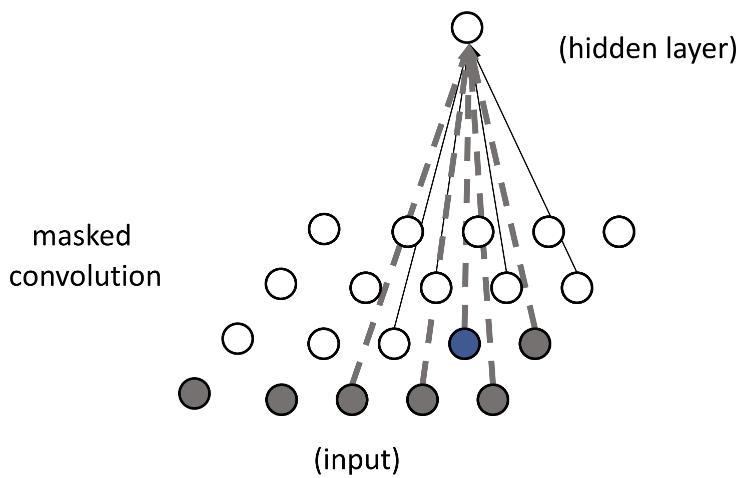
- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Simply treating $\boldsymbol{x}_{<k}$ as one-dimensional (instead of two-dimensional) vector:



  - **Inference** requires iterative forward procedure (slow)
  - **Training** requires single forward pass for CNN, but multiple pass for RNN (slow)
  - Effective receptive field (context of pixel generation) is unbounded for RNN, but bounded for CNN (constrained)

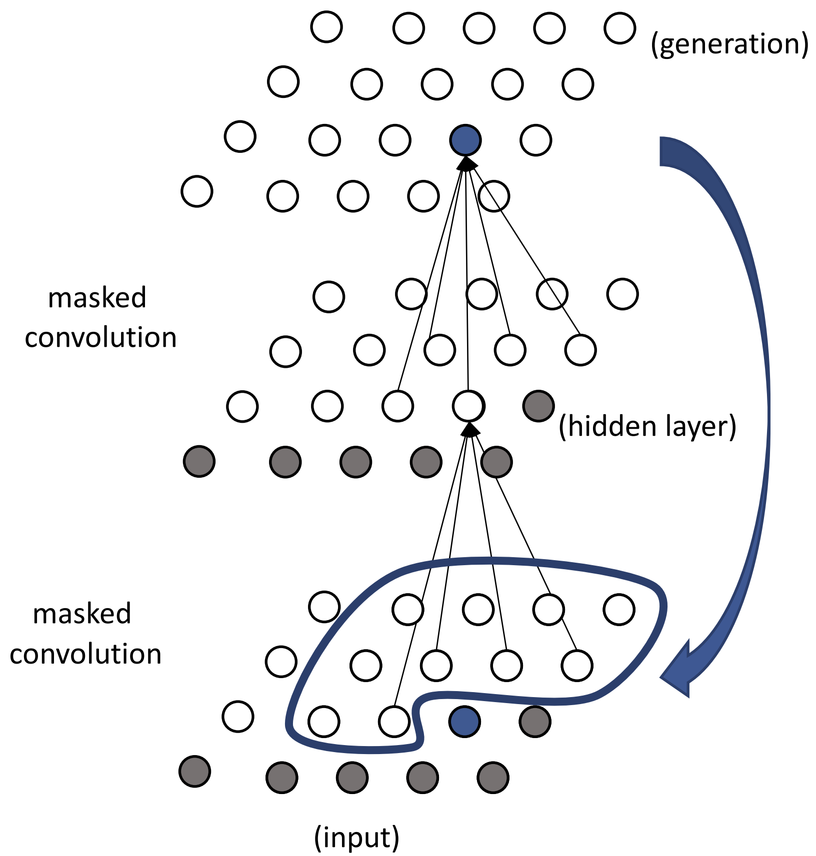
Next, extending to two-dimensional data

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)



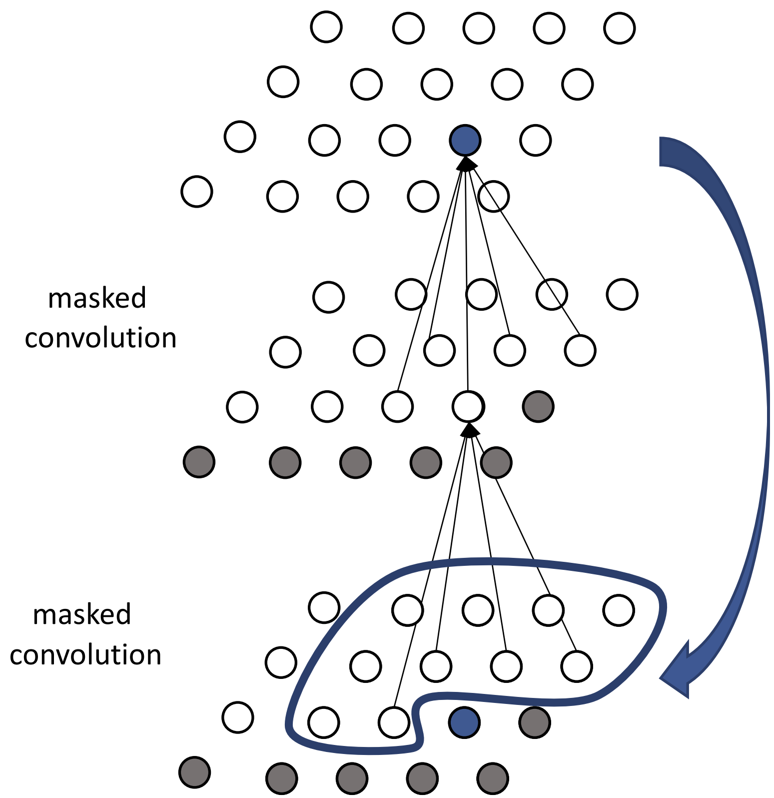
Pixel CNN

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)

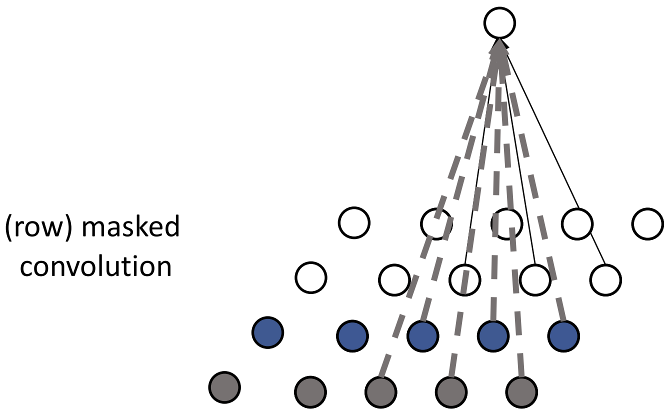


Pixel CNN

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
    - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
    - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel)

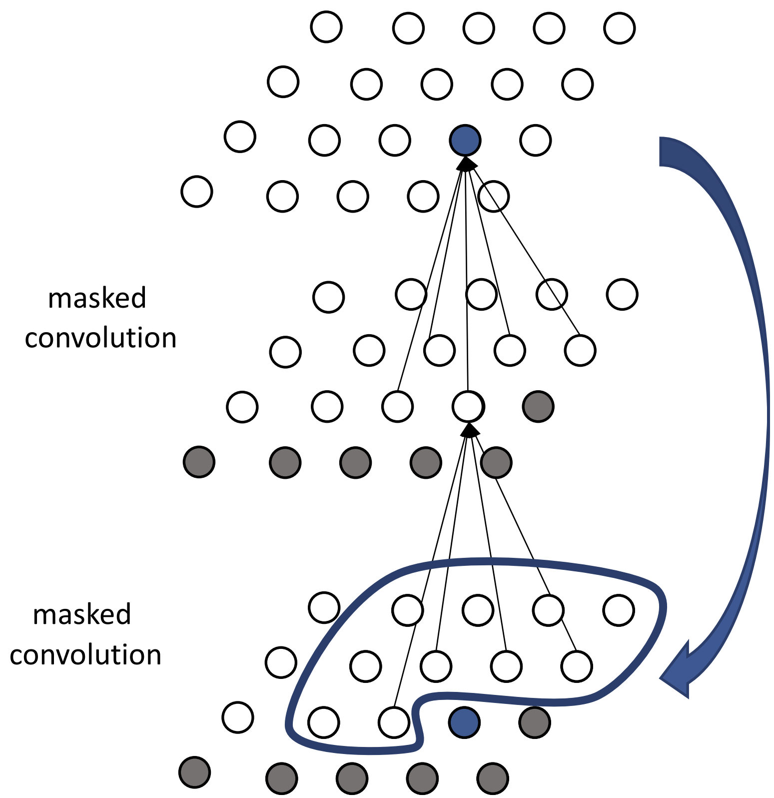


Pixel CNN

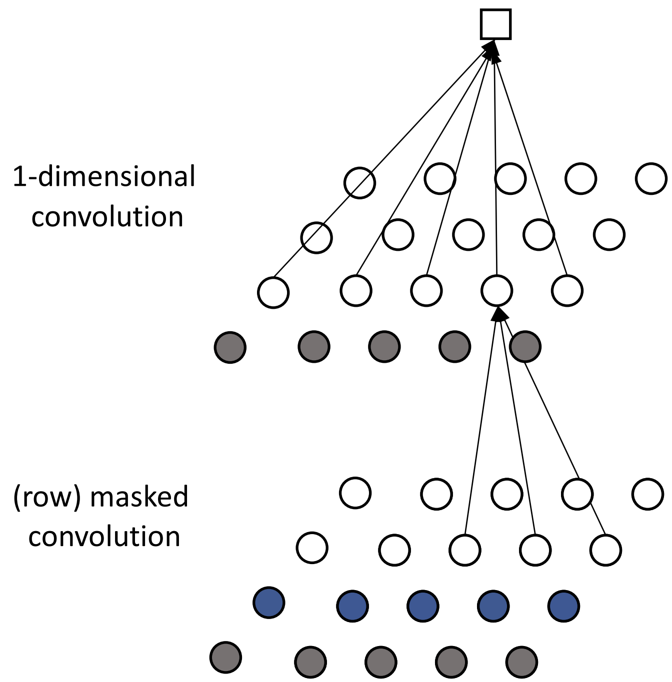


Row LSTM

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
    - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
    - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel)

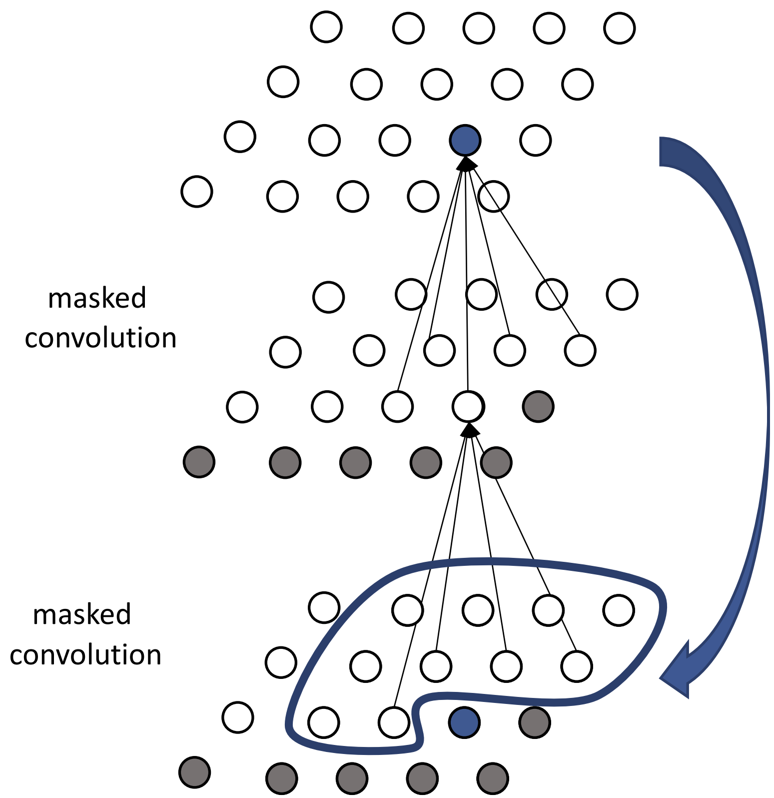


Pixel CNN

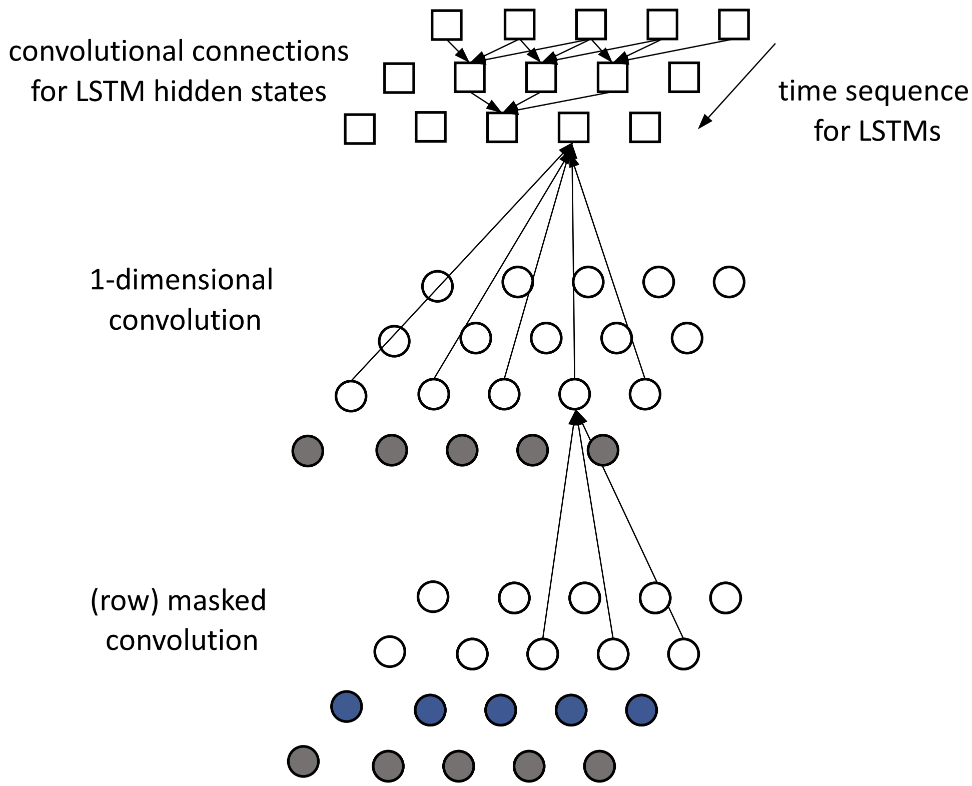


Row LSTM

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel)



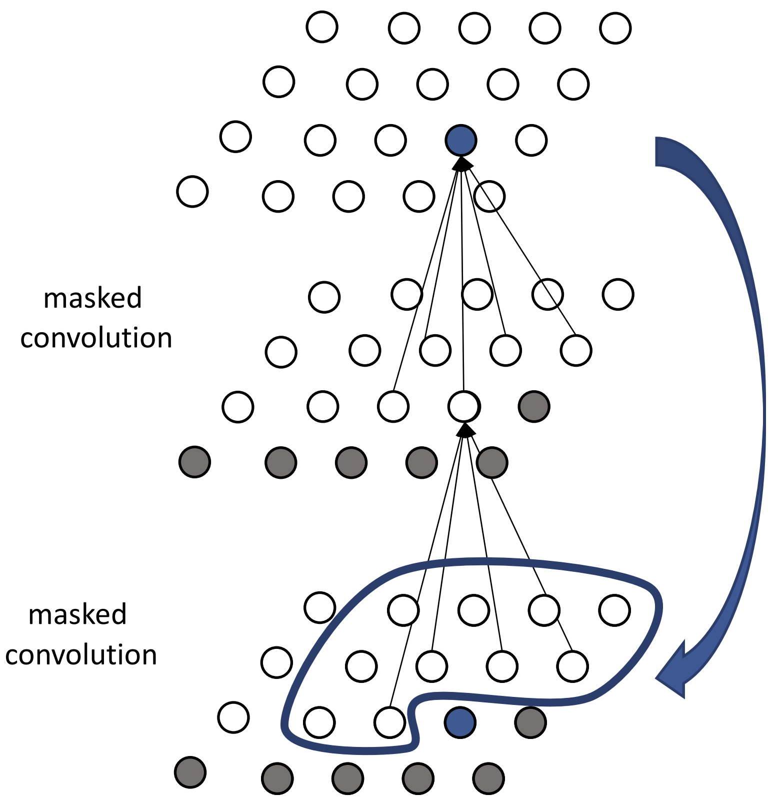
Pixel CNN



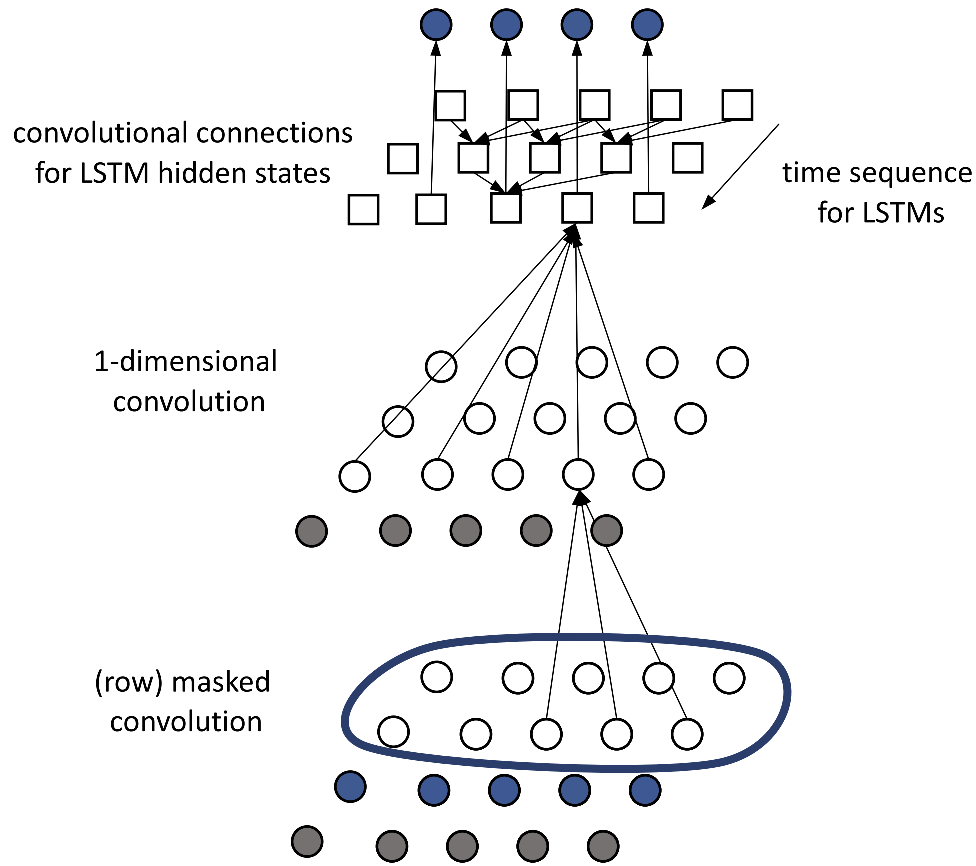
Row LSTM

Pixel Convolutional/Recurrent Neural Network (PixelCNN/PixelRNN)

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image <u>row-by-row</u> (not pixel-by-pixel)



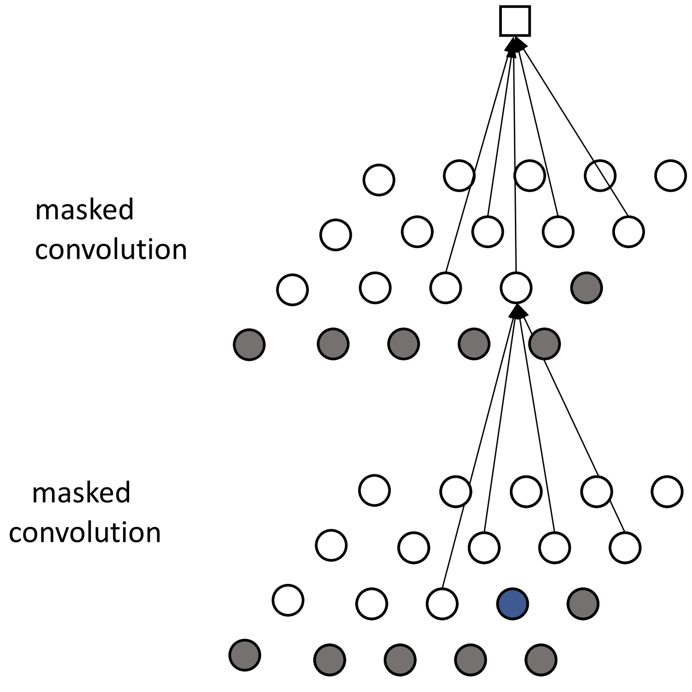
Pixel CNN



Row LSTM

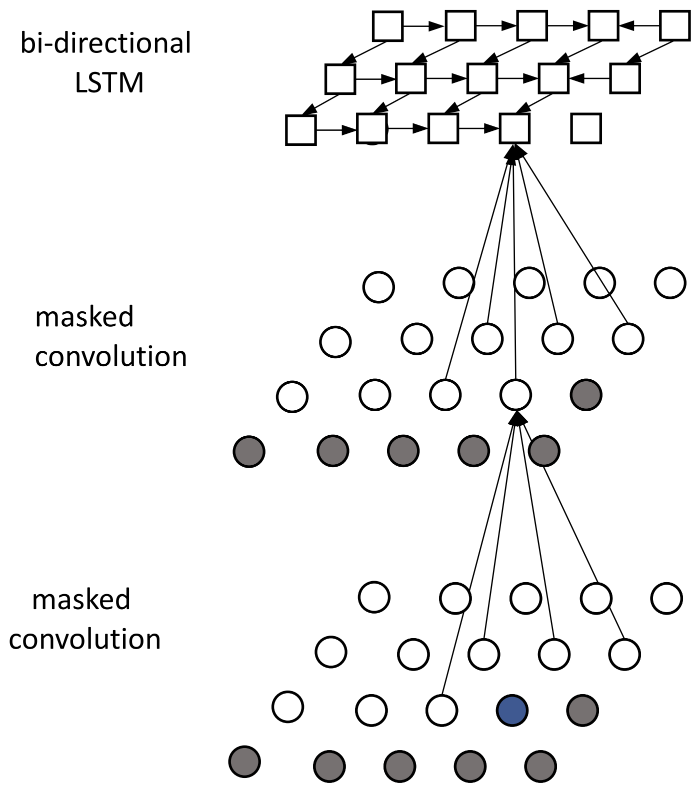
Next, introducing column-wise dependencies using LSTMs

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel)
  - Diagonal BiLSTM use bi-directional LSTMs, to generate image pixel-by-pixel



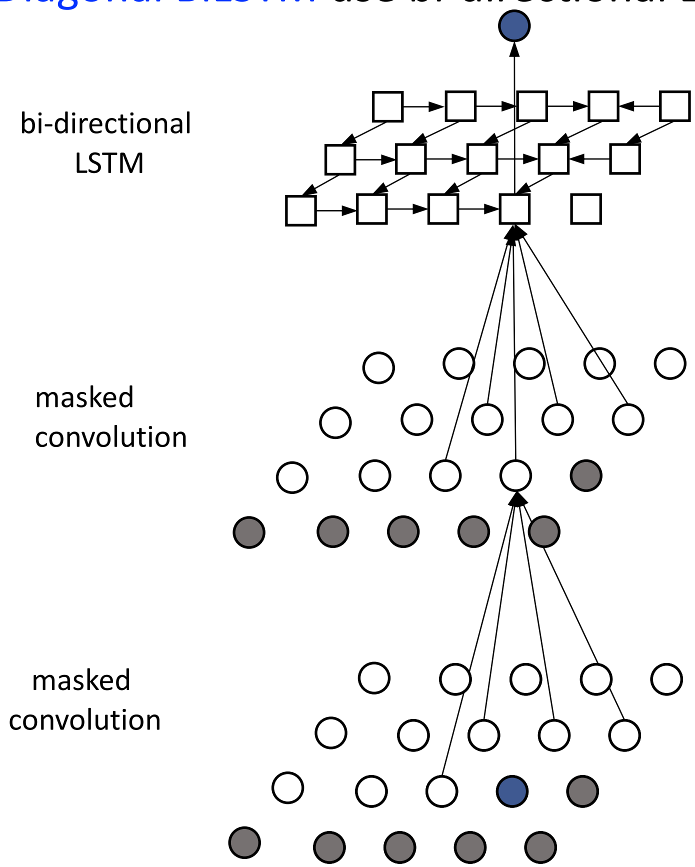
Diagonal BiLSTM

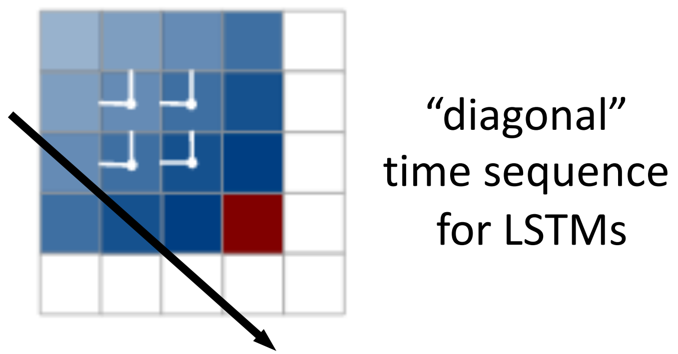
- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel)
  - Diagonal BiLSTM use bi-directional LSTMs, to generate image pixel-by-pixel



Diagonal BiLSTM

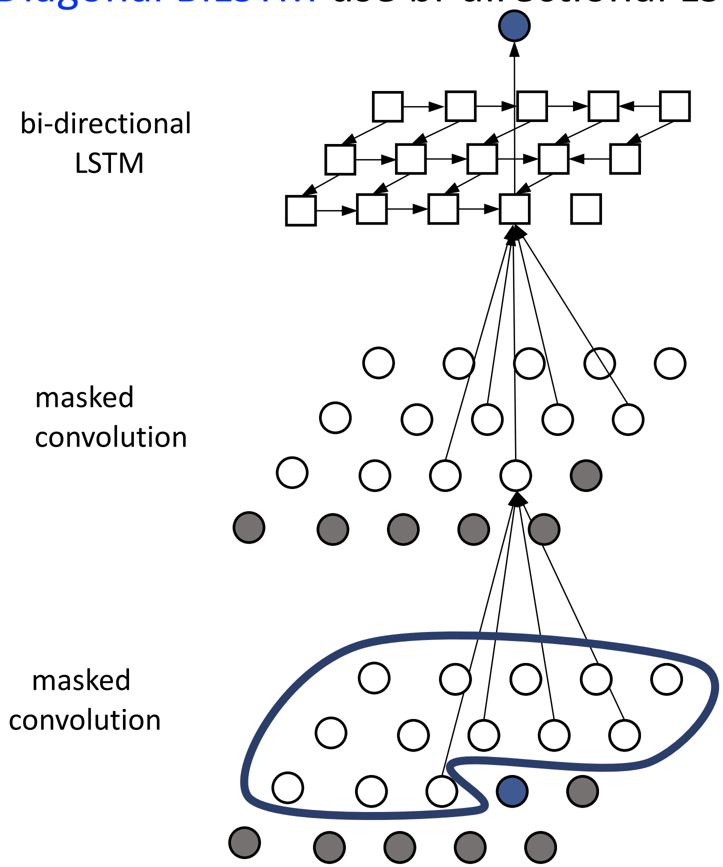
- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel)
  - Diagonal BiLSTM use bi-directional LSTMs, to generate image pixel-by-pixel

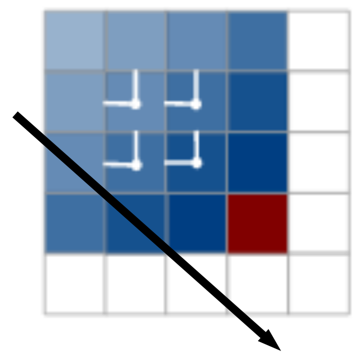




Diagonal BiLSTM

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$ [Oord et al., 2016]
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel)
  - Diagonal BiLSTM use bi-directional LSTMs, to generate image pixel-by-pixel





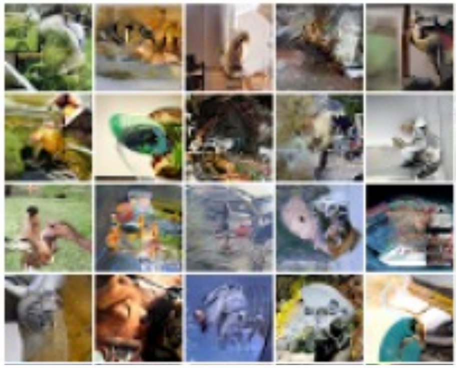
"diagonal" time sequence for LSTMs

- Receptive field now covers every pixels generated previously

Diagonal BiLSTM

- Image generation results from CIFAR-10 and ImageNet:

CIFAR-10



ImageNet

- Evaluation of negative log-likelihood (NLL) on MNIST and CIFAR-10 dataset:

Only explicit models (not GAN) can compute NLL

Model	NLL Test
PixelCNN:	81.30
Row LSTM:	80.54
Diagonal BiLSTM (1 layer, $h = 32$):	**80.75**
Diagonal BiLSTM (7 layers, $h = 16$):	**79.20**

MNIST

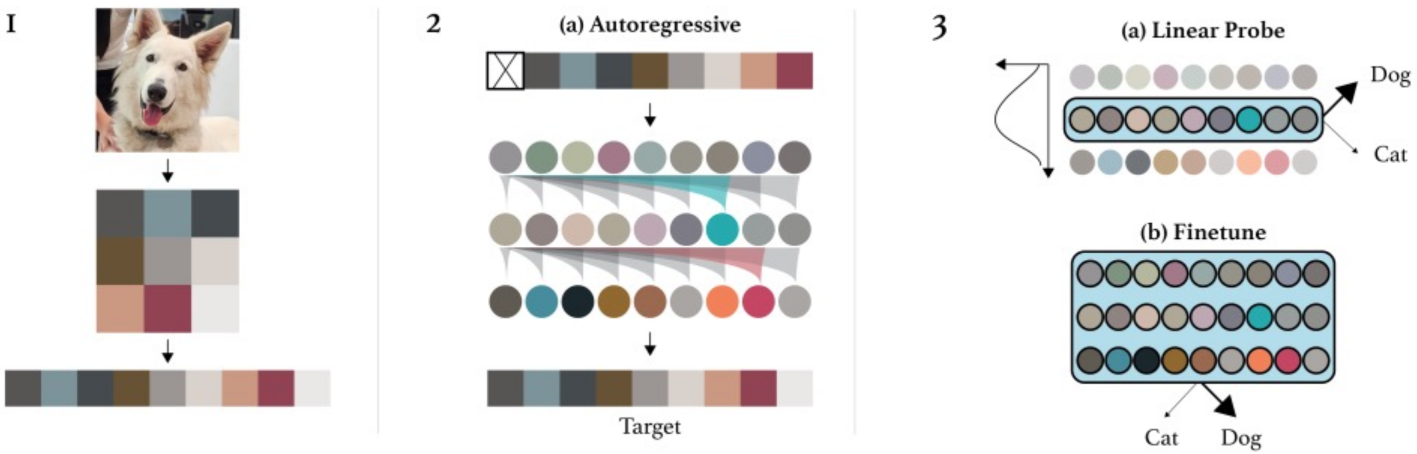
Model	NLL Test (Train)
PixelCNN:	3.14 (3.08)
Row LSTM:	3.07 (3.00)
Diagonal BiLSTM:	**3.00** (2.93)

CIFAR-10

- PixelCNN is easiest to train and Diagonal BiLSTM performs best

- **Generative Pretraining from Pixels** [Chen et al., 2020]
  - Apply GPT [Brown et al., 2020] to image domain by flattening image to 1D.
  - Train autoregressive transformer which predicts the pixels without knowledge of 2D input structure.

$$L_{AR} = \mathop{\mathbb{E}}_{x \sim X}[-\log p(x)] \quad \text{where} \quad p(x) = \prod_{i=1}^{n} p(x_{\pi_i} | x_{\pi_1}, ..., x_{\pi_{i-1}}, \theta)$$



- **Generative Pretraining from Pixels** [Chen et al., 2020]
  - ImageGPT not only learns image representations,
    - It outperforms supervised representation with ImageNet in transfer learning.

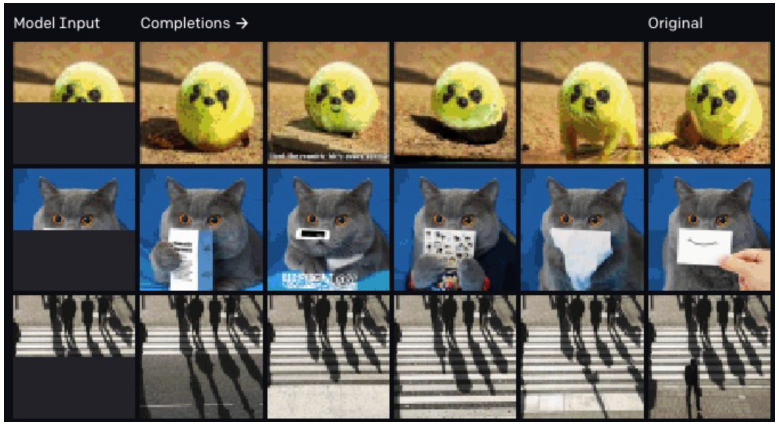
Model	Acc	Unsup Transfer	Sup Transfer
CIFAR-10			
ResNet-152	94		√
SimCLR	95.3	√	
iGPT-L	96.3	√	
CIFAR-100			
ResNet-152	78.0		√
SimCLR	80.2	√	
iGPT-L	82.8	√	
STL-10			
AMDIM-L	94.2	√	
iGPT-L	95.5	√	

Linear probing

Model	Acc	Unsup Transfer	Sup Transfer
CIFAR-10			
AutoAugment	98.5		
SimCLR	98.6	√	
GPipe	99.0		√
iGPT-L	99.0	√	
CIFAR-100			
iGPT-L	88.5	√	
SimCLR	89.0	√	
AutoAugment	89.3		
EfficientNet	91.7		√

Full finetuning

  - but also shows inpainting ability.



- **Scaling Autoregressive Video Models** [Weissenborn et al., 2020]
    - Apply GPT to video domain by flattening video to 1D.
    - However, using all pixels from a video is computationally infeasible
        - e.g.) 32x32 video of length 16 has 16 * 32 * 32 * 3 = 49,152 pixels
            - Much longer than the input length of GPT3 (=2048), ImageGPT (=3072)

    - **Main idea:** Reduce the complexity of autoregressive video generation by
        - 1) Designing an efficient self-attention layer for videos
        - 2) Operating on sub-sampled videos instead of pixels

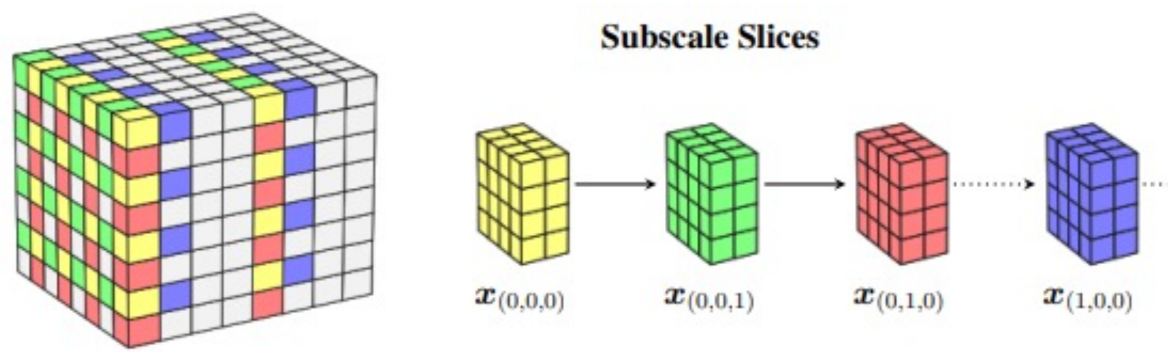
- **Scaling Autoregressive Video Models** [Weissenborn et al., 2020]
  - Apply GPT to video domain by flattening video to 1D.
  - However, using all pixels from a video is computationally infeasible

  - **Idea 1:** Video Transformer with multiple stacked block-local self-attention
    - Reduces the computation cost of self-attention over videos, by
      1. Decompose a video of (T, H, W) into $n_p = t \cdot h \cdot w$ blocks of (t, h, w)
      2. Separately apply self attentions over n_p blocks
         - Attention complexity $(T \cdot H \cdot W)^2 \Rightarrow n_p \cdot (t \cdot h \cdot w)^2$
      3. Concatenate the outputs and process through a fully connected layer

    - For the connectivity between all pixels, use different block sizes at every layer

- **Scaling Autoregressive Video Models** [Weissenborn et al., 2020]
  - Apply GPT to video domain by flattening video to 1D.
  - However, using all pixels from a video is computationally infeasible

  - **Idea 2:** Divide the video into non-overlapping 3D blocks
    - Further reduces the complexity by decomposing the video itself
    - Introduce a subscale factor $\boldsymbol{s} = (s_t, s_h, s_w)$ that divides a video into $s = (s_t \cdot s_h \cdot s_w)$ sub-sampled videos (slices)
      - Then, each slice is processed through the block-local self-attention layers



    - And sequentially generate $x_{(0,0,0)}, x_{(0,0,1)}, \ldots$
      - e.g.) If we use $\boldsymbol{s} = (4, 2, 2)$, each slice consists of 4 * 16 * 16 * 3 = 3072 pixels
        - Attention complexity: 49152^2 => 3072^2 (256 times lower)

- **VideoGPT** [Yan et al., 2021]
  - Other approach for autoregressive video generation
  - Learns downsampled discrete representations over space-time

  - **Main idea of VideoGPT**
    1. Train a VQ-VAE with 3D CNNs on the video data to learn discrete latent representations downsampled over space-time
    2. Train autoregressive transformer (Image-GPT architecture) in the latent space for learning a prior
    3. Decode the predicted discrete latents using the VQ-VAE decoder

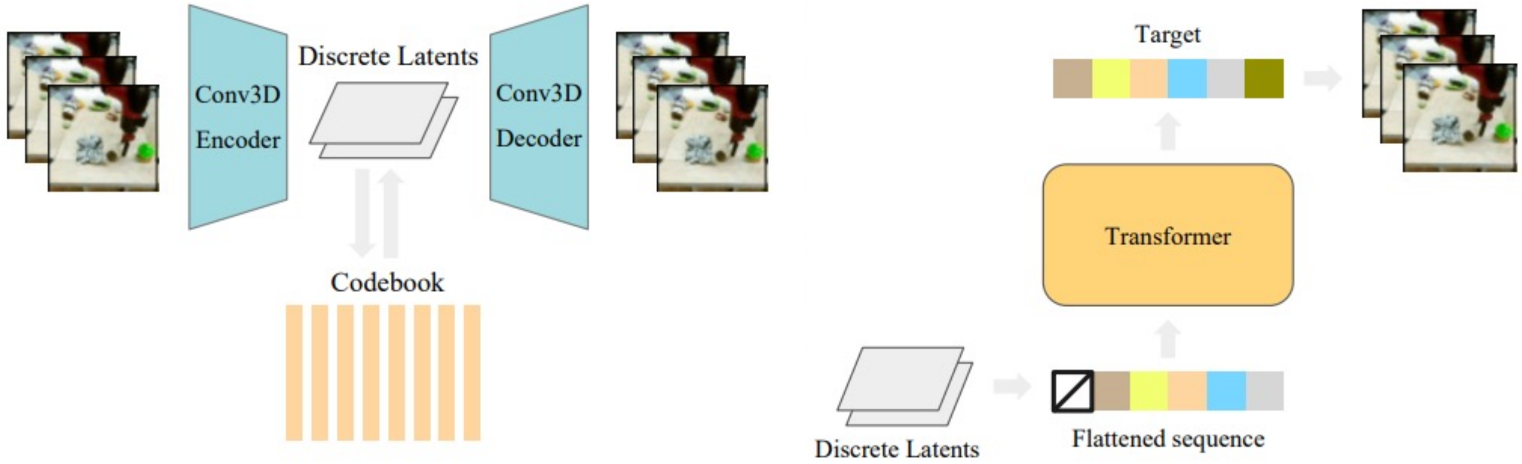


Table of Contents

1. **Introduction**
    - Implicit vs explicit density models

2. **Variational Autoencoders (VAE)**
    - Variational autoencoders
    - Tighter bounds for variational inference
    - Techniques to mitigate posterior collapse
    - Large-scale generation via hierarchical structures
    - Diffusion probabilistic models

3. **Energy-based Models (EBM)**
    - Energy-based models
    - Score matching generative models

4. **Autoregressive and Flow-based Models**
    - Autoregressive models
    - Flow-based models

Flow-based Models

- Modifying data distribution by flow (sequence) of invertible transformations:

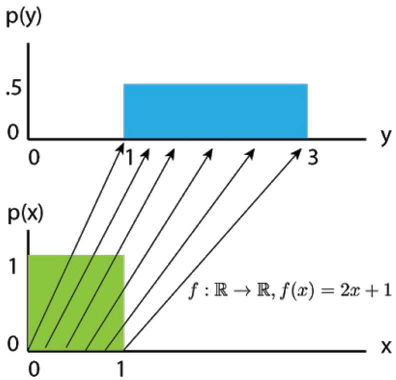
$$\boldsymbol{x} = \boldsymbol{z}_0 \ \rightarrow \ \boldsymbol{z}_T = f_T \circ f_{T-1} \circ \cdots f_1(\boldsymbol{z}_0) \qquad \boldsymbol{z}_t \in \mathbb{R}^K$$

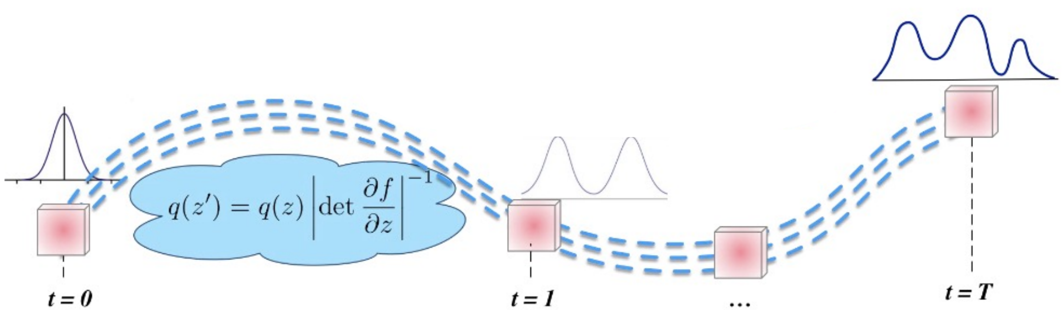
  - Final variable follows some specified prior $p_T(\boldsymbol{z}_T)$

- Data distribution is explicitly modeled by change-of-variables formula:

$$\log p(\boldsymbol{x}) = \log p(\boldsymbol{z}_0) = \log p_T(\boldsymbol{z}_T) + \sum_{t=1}^{T} \log \left| \det \left(\frac{\partial f_t(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} \right) \right|$$

  - Log-likelihood $\log p(\boldsymbol{x})$ can be maximized directly





* source: Jang, https://blog.evjang.com/2018/01/nf1.html,
Mohamed et al., https://www.shakirm.com/slides/DeepGenModelsTutorial.pdf

- Modifying data distribution by flow (sequence) of invertible transformations:

$$\boldsymbol{x} = \boldsymbol{z}_0 \;\; \boldsymbol{\rightarrow} \;\; \boldsymbol{z}_T = f_T \circ f_{T-1} \circ \cdots f_1(\boldsymbol{z}_0) \qquad\qquad \boldsymbol{z}_t \in \mathbb{R}^K$$

  - Final variable follows some specified prior $p_T(\boldsymbol{z}_T)$

- Data distribution is explicitly modeled by change-of-variables formula:

$$\log p(\boldsymbol{x}) = \log p(\boldsymbol{z}_0) = \log p_T(\boldsymbol{z}_T) + \sum_{t=1}^{T} \log \left| \det \left(\frac{\partial f_t(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} \right) \right|$$

  - Log-likelihood $\log p(\boldsymbol{x})$ can be maximized directly
  - Naïvely computing $\log |\det (\partial f_t(\boldsymbol{z}_{t-1}) / \partial \boldsymbol{z}_{t-1})|$ requires $\mathcal{O}(K^3)$ complexity, which is not scalable for large-scale neural networks

How to design flexible yet tractable form of invertible transformations?

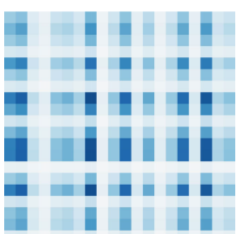
Design Schemes for Normalizing Flows

- To reduce complexity of log-det-Jacobian, prior works consider
  - Carefully designed architectures (low rank, coupling, autoregressive)
  - Stochastic estimator of free-form Jacobian

1. Det Identities

Planar NF
Sylvester NF
...

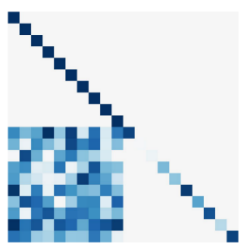
Jacobian



(Low rank)

2. Coupling Blocks

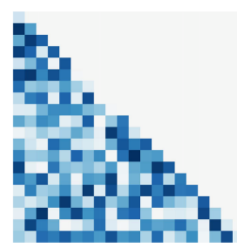
NICE
Real NVP
Glow
...



(Lower triangular + structured)

3. Autoregressive

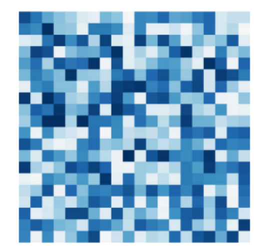
Inverse AF
Neural AF
Masked AF
...



(Lower triangular)

4. Unbiased Estimation

FFJORD
Residual Flows



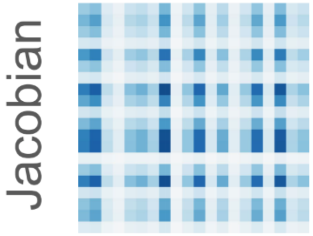
(Arbitrary)

* source: Chen, https://www.cs.toronto.edu/~duvenaud/talks/residual_flows_slides.pdf

- To reduce complexity of log-det-Jacobian, prior works consider
  - Carefully designed architectures (low rank, coupling, autoregressive)
  - Stochastic estimator of free-form Jacobian

1. Det Identities

Planar NF
Sylvester NF
...



(Low rank)

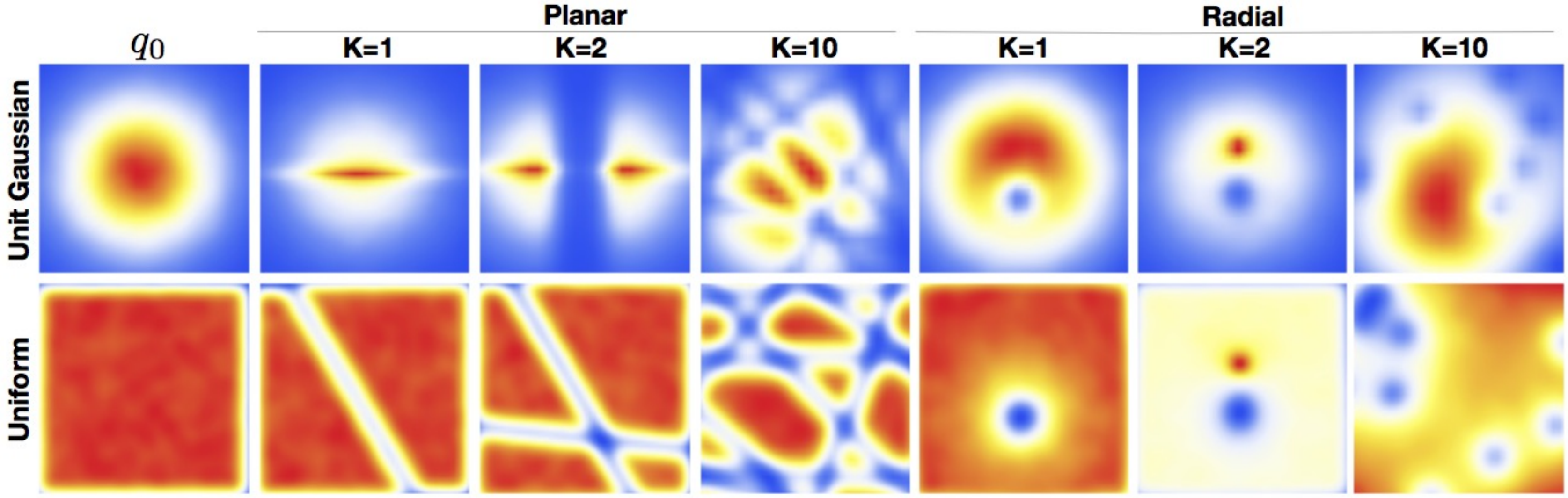
* source: Chen, https://www.cs.toronto.edu/~duvenaud/talks/residual_flows_slides.pdf

Normalizing Flow (NF)

- Basic layers with linear log-det-Jacobian complexity [Rezende et al., 2015]

- Planar flow: $f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\intercal \mathbf{z} + b)$
  - Determinant of Jacobian is $\left|\det \frac{\partial f}{\partial \mathbf{z}}\right| = |1 + \mathbf{u}^\intercal h'(\mathbf{w}^\intercal \mathbf{z} + b)\mathbf{w}|$

- Radial flow: $f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r)(\mathbf{z} - \mathbf{z}_0)$ $(r = |\mathbf{z} - \mathbf{z}_0|,\ h(\alpha, r) = 1/(\alpha + r))$
  - Determinant of Jacobian is $[1 + \beta h(\alpha, r)]^{d-1}[1 + \beta h(\alpha, r) + h'(\alpha, r)r)]$



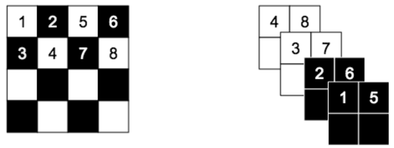
Design Schemes for Normalizing Flows

- To reduce complexity of log-det-Jacobian, prior works consider
  - Carefully designed architectures (low rank, coupling, autoregressive)
  - Stochastic estimator of free-form Jacobian

* source: Chen, https://www.cs.toronto.edu/~duvenaud/talks/residual_flows_slides.pdf

- Coupling layer $\boldsymbol{z}_t = f_t(\boldsymbol{z}_{t-1})$ for flow with tractable inference [Dinh et al., 2017]:
  1. Partition the variable into two parts:

$$\boldsymbol{z}_{t-1} \to [\boldsymbol{z}_{t-1,1:d}, \boldsymbol{z}_{t-1,d+1:K}]$$

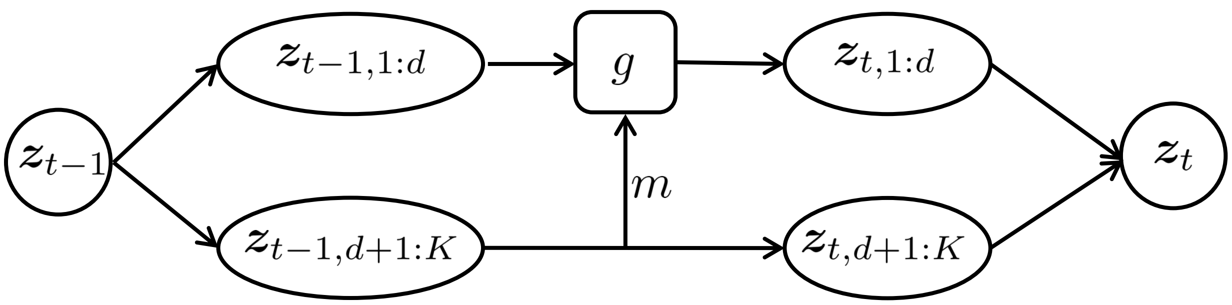


spatial-partition   channel-partition

  2. Coupling law defines a simple invertible transformation of the first partition given the second partition (g and m are described later)

$$\boldsymbol{z}_{t,d+1:K} = g(\boldsymbol{z}_{t-1,d+1:K}; m(\boldsymbol{z}_{t-1,1:d}))$$

  3. Second partition is left invariant ($\boldsymbol{z}_{t,1:d} = \boldsymbol{z}_{t-1,1:d}$)



- Affine coupling layer was shown to be effective in practice:

$$
\begin{aligned}
\boldsymbol{z}_{t,d+1:K} &= g(\boldsymbol{z}_{t-1,d+1:K}; m(\boldsymbol{z}_{t-1,1:d})) \\
&= \boldsymbol{z}_{t-1,d+1:K} \odot \exp(m_1(\boldsymbol{z}_{t-1,1:d})) + m_2(\boldsymbol{z}_{t-1,1:d})
\end{aligned}
$$

element-wise product

neural networks

- Jacobian of each transformation becomes a lower triangular matrix:

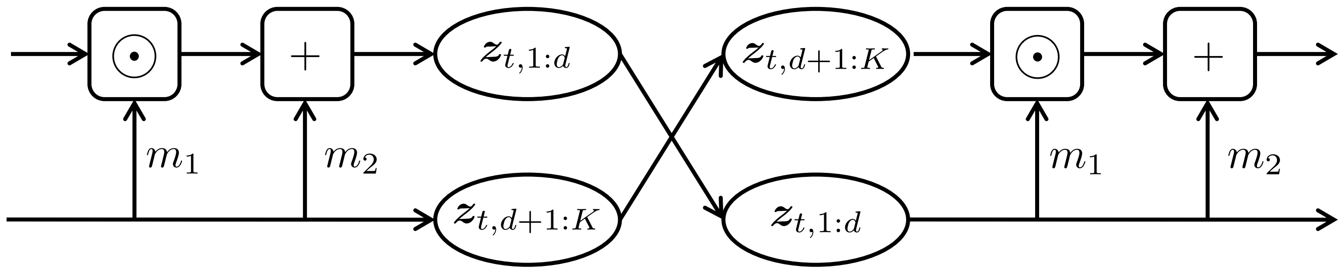
$$
\frac{\partial f_{t-1}(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} = \begin{bmatrix} \mathbb{I} & \mathbf{0} \\ \frac{\partial g_{t-1}(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} & \mathrm{diag}(\exp(m_1(\boldsymbol{z}_{t-1,1:d}))) \end{bmatrix} \rightarrow \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ & a_{22} & 0 & \vdots \\ & & \ddots & 0 \\ & & & a_{KK} \end{bmatrix}
$$

- Inference for such transformations can be done in tractable time
  - Determinant of lower triangular matrix is a product of diagonals

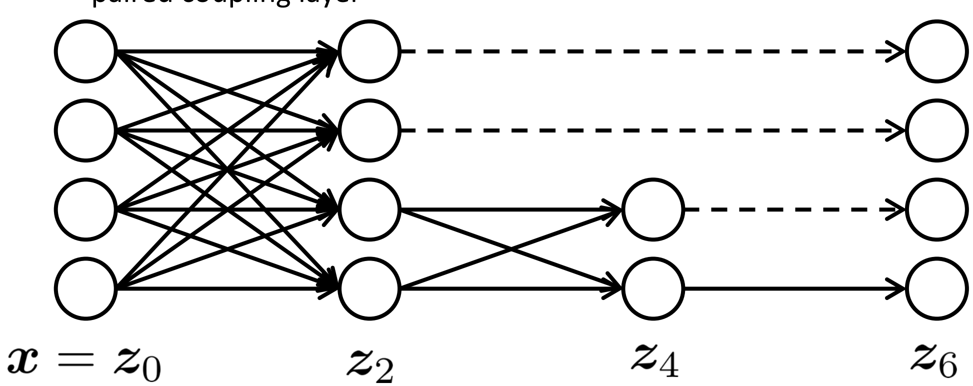
$$
\log p(\boldsymbol{x}) = \log p(\boldsymbol{z}_0) = \log p_T(\boldsymbol{z}_T) + \sum_{t=1}^{T} \log \left| \det \left(\frac{\partial f_t(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} \right) \right|
$$

Real-valued Non-volume Preserving Flow (Real NVP)

- For each coupling layer, there exists asymmetry since the first partition $z_{t-1,1:d}$ is left invariant
  - Two coupling layers are paired alternatively to overcome this issue



- Multi-scale architectures are used
  - Half variables follow Gaussian distribution at each scale

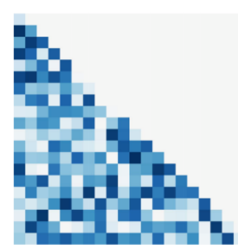


$$p(z_3) = \mathcal{N}(z_3; \mathbf{0}, \mathbf{1})$$

- To reduce complexity of log-det-Jacobian, prior works consider
  - Carefully designed architectures (low rank, coupling, autoregressive)
  - Stochastic estimator of free-form Jacobian

3. Autoregressive

Inverse AF
Neural AF
Masked AF
...

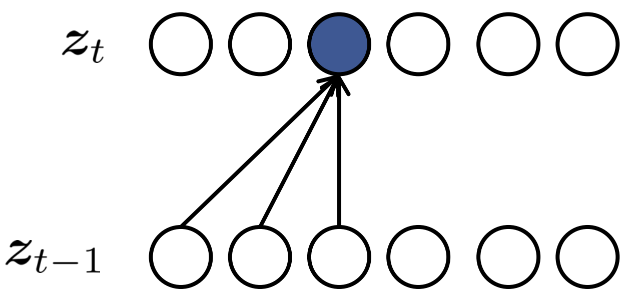


(Lower triangular)

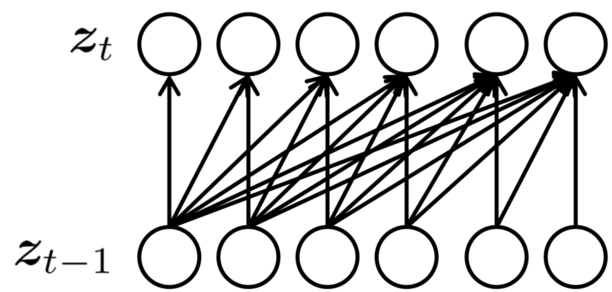
* source: Chen, https://www.cs.toronto.edu/~duvenaud/talks/residual_flows_slides.pdf

- Inverse autoregressive flow (IAF) modifies each dimension of variable in autoregressive manner [Kingma et al., 2016]:
  - Forward pass $z_0 \to z_T$ is fast, but backward pass $z_T \to z_0$ is slow
    - Used for VAE posterior: Only forward pass is required for approx. posterior

$$\boldsymbol{z}_{t,d} = \mu_{t,d}(\boldsymbol{z}_{t-1,1:d-1}) + \sigma_{t,d}(\boldsymbol{z}_{t-1,1:d-1})\boldsymbol{z}_{t-1,d}$$



case of $d = 3$



updates done in parallel

- Inference for corresponding normalizing flow is efficient:

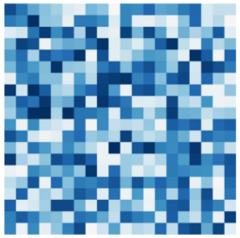
$$\log q(\boldsymbol{z}|\boldsymbol{x}) = \log q_0(\boldsymbol{z}_0|\boldsymbol{x}) + \sum_{t=1}^{T} \log \left| \det \left(\frac{\partial f_t(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} \right) \right|$$

$$\begin{bmatrix} \sigma_{t,1} & 0 & \cdots & 0 \\ & \sigma_{t,2} & 0 & \vdots \\ & & \ddots & 0 \\ & & & \sigma_{t,K} \end{bmatrix}$$

- To reduce complexity of log-det-Jacobian, prior works consider
  - Carefully designed architectures (low rank, coupling, autoregressive)
  - Stochastic estimator of free-form Jacobian

4. Unbiased Estimation

FFJORD
Residual Flows



(Arbitrary)

* source: Chen, https://www.cs.toronto.edu/~duvenaud/talks/residual_flows_slides.pdf

- Discrete normalizing flows need a carefully designed (less expressive) layers to achieve affordable (not cubic) complexity
  → Continuous normalizing flow affords an arbitrary network architecture

- Consider a continuous transformation $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$ (instead of $\mathbf{z}_1 = f(\mathbf{z}_0)$), then the sampling can be done by an **ordinary differential equation (ODE)**:

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta) dt$$

  - Here, the change in log-probability also follows an ODE:

$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial \mathrm{z}(t)}\right) dt$$

  - **Remark:** We only need a **trace** (not a **determinant**) to compute likelihood

  - The network $f(z(t), t, \theta)$ is learned by gradient descent (backpropagation follows another ODE) [Chen et al., 20018; Grathwohl et al., 2019]

References (VAE)

[Kingma et al., 2013] Auto-Encoding Variational Bayes, ICLR 2013
link: https://arxiv.org/abs/1802.06455

[Burda et al., 2016] Importance Weighted Autoencoders, ICLR 2016
link: https://arxiv.org/abs/1509.00519

[Kim et al., 2018] Semi-Amortized Variational Autoencoders, ICML 2018
link: https://arxiv.org/abs/1802.02550

[Bowman et al., 2016] Generating Sentences from a Continuous Space, CONLL 2016
link: https://arxiv.org/abs/1511.06349

[Razavi et al., 2019a] Preventing Posterior Collapse with delta-VAEs, ICLR 2019
link: https://arxiv.org/abs/1901.03416

[Tolstikhin et al., 2018] Wasserstein Auto-Encoders, ICLR 2018
link: https://arxiv.org/abs/1711.01558

[He et al., 2019] Lagging Inference Networks and Posterior Collapse in Variational Autoencoders, ICLR 2019
link: https://arxiv.org/abs/1901.05534

[Oord et al., 2017] Neural Discrete Representation Learning, NeurIPS 2017
link: https://arxiv.org/abs/1711.00937

[Razavi et al., 2017b] Generating Diverse High-Fidelity Images with VQ-VAE-2, NeurIPS 2019
link: https://arxiv.org/abs/1906.00446

[Vahdat et al., 2020] NVAE: A Deep Hierarchical Variational Autoencoder, NeurIPS 2020
link: https://arxiv.org/abs/2007.03898

[Child et al., 2021] Very Deep VAEs Generalize Autoregressive Models and Can Outperform Them on Images, ICLR 2021
link: https://arxiv.org/abs/2011.10650

References (VAE)

[Sohl-Dickstein et al., 2015] Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015
link: https://arxiv.org/abs/1503.03585

[Ho et al., 2020] Denoising Diffusion Probabilistic Models, NeurIPS 2020
link: https://arxiv.org/abs/2006.11239

[Nichol and Dhariwal, 2021] Improved Denoising Diffusion Probabilistic Models, ICML 2021
link: https://arxiv.org/abs/2102.09672

[Song et al., 2021] Denoising Diffusion Implicit Models, ICLR 2021
link: https://arxiv.org/abs/2010.02502

[Dhariwal and Nichol, 2021] Diffusion Models Beat GANs on Image Synthesis, NeurIPS 2021
link: https://arxiv.org/abs/2105.05233

[Ho et al., 2021] Classifier-Free Diffusion Guidance, NeurIPS Workshop on Deep Generative Models and Downstream Applications 2021
link: https://openreview.net/pdf?id=qw8AKxfYbI

[Nichol et al., 2021] GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models, arXiv 20 21
link: https://arxiv.org/abs/2112.10741

References (EBM, score matching)

[LeCun et al., 2006] A Tutorial on Energy-Based Learning, Technical report 2006
link: http://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf

[Du & Mordatch, 2019] Implicit Generation and Generalization in Energy-Based Models, NeurIPS 2019
link: https://arxiv.org/abs/1903.08689

[Welling & Teh, 2011] Bayesian Learning via Stochastic Gradient Langevin Dynamics, ICML 2011
link: https://dl.acm.org/doi/10.5555/3104482.3104568

[Zhao et al., 2017] Energy-based Generative Adversarial Network, ICLR 2017
link: https://arxiv.org/abs/1609.03126

[Grathwohl et al., 2020] Your Classifier is Secretly an Energy Based Model and You Should Treat it Like One, ICLR 2020
link: https://arxiv.org/abs/1912.03263

[Song & Kingma, 2021] How to Train Your Energy-Based Models, arXiv 2021
link: https://arxiv.org/abs/2101.03288

[Hyvärinen, 2005] Estimation of Non-Normalized Statistical Models by Score Matching, JMLR 2005
link: https://jmlr.org/papers/v6/hyvarinen05a.html

[Vincent, 2011] A Connection Between Score Matching and Denoising Autoencoders, Neural Computation 2011
link: https://ieeexplore.ieee.org/document/6795935

[Song et al., 2019] Generative Modeling by Estimating Gradients of the Data Distribution, NeurIPS 2019
link: https://arxiv.org/abs/1907.05600

[Song et al., 2021] Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021
link: https://arxiv.org/abs/2011.13456

References (AR, flow)

[Oord et al., 2016] Pixel Recurrent Neural Networks, ICML 2016
link: https://arxiv.org/abs/1601.06759

[Chen et al., 2020] Generative Pretraining from Pixels, ICML 2020
link: https://cdn.openai.com/papers/Generative_Pretraining_from_Pixels_V1_ICML.pdf

[Weissenborn et al., 2020] Scaling Autoregressive Video Models, ICLR 2020
link: https://arxiv.org/abs/1906.02634

[Yan et al., 2021] VideoGPT: Video Generation using VQ-VAE and Transformers, arXiv 2021
link: https://arxiv.org/abs/2104.10157

[Oord et al., 2017] WaveNet: A Generative Model for Raw Audio, arXiv 2017
link: https://arxiv.org/abs/1703.01961

[Rezende et al., 2015] Variational Inference with Normalizing Flows, ICML 2015
link: https://arxiv.org/abs/1705.08665

[Dinh et al., 2017] Density Estimation using Real NVP, ICLR 2017
link: https://arxiv.org/abs/1605.08803

[Kingma et al., 2018] Generative Flow with Invertible 1x1 Convolutions, NeurIPS 2018
link: https://arxiv.org/abs/1807.03039

[Kingma et al., 2016] Improving Inference with Inverse Autoregressive Flows, NeurIPS 2016
link: https://arxiv.org/abs/1710.10628

[Chen et al., 2018] Neural Ordinary Differential Equations, NeurIPS 2018
link: https://arxiv.org/abs/1806.07366

[Grathwohl et al., 2019] FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models, ICLR 2019
link: https://arxiv.org/abs/1810.01367

[Chen et al., 2019] Residual Flows for Invertible Generative Modeling, NeurIPS 2019
link: https://arxiv.org/abs/1906.02735