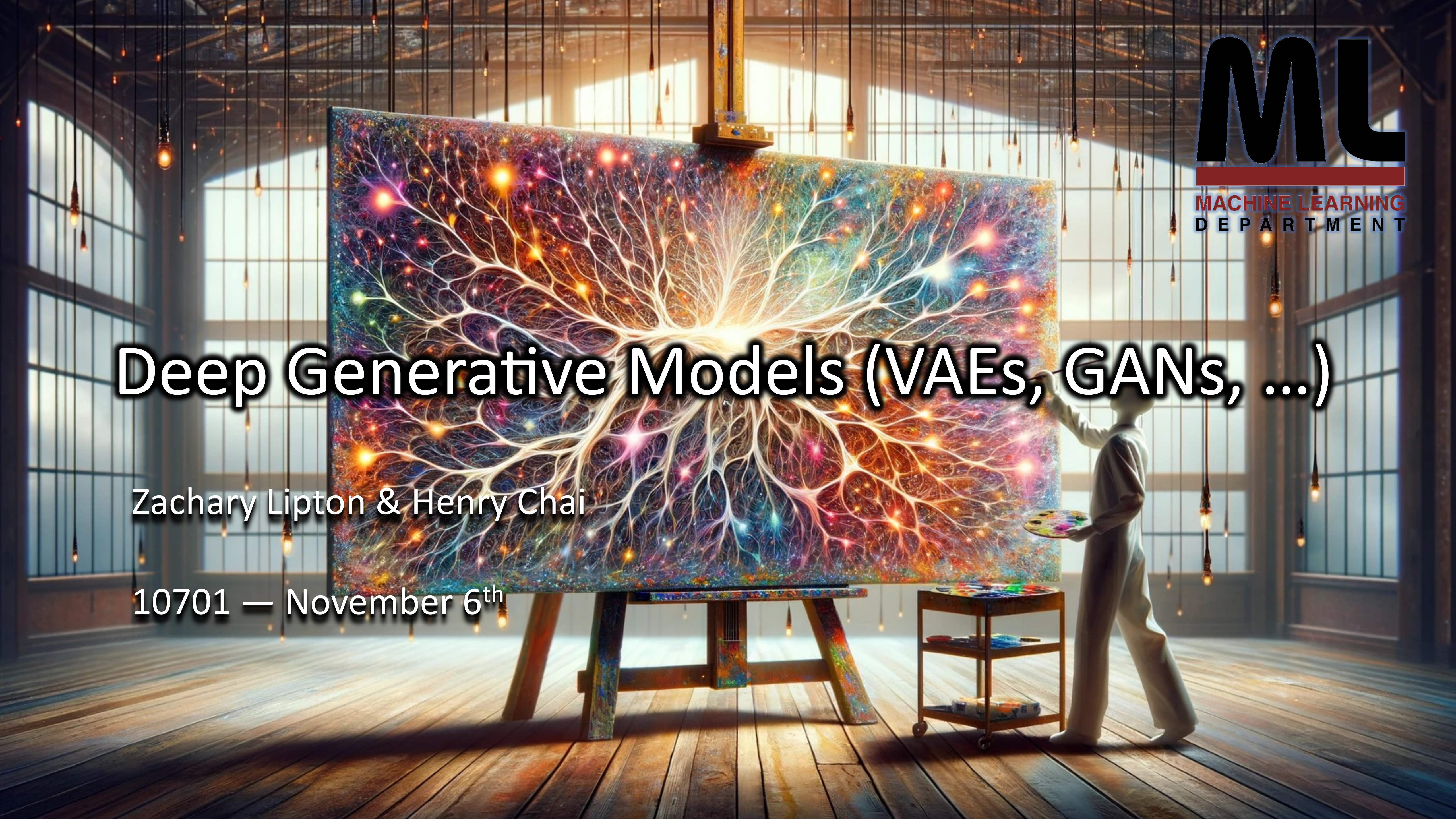


ML
MACHINE LEARNING
DEPARTMENT

Deep Generative Models (VAEs, GANs, …)

Zachary Lipton & Henry Chai

10701 — November 6th

Recap Unsupervised Learning

- Aims:
  - Exploratory data analysis
  - Leverage unlabeled data (e.g., for feature learning)
- Some unsupervised tasks:
  - (Hard) Clustering
  - Topic modeling (soft clustering)
  - Dimensionality reduction
  - Generative modeling
- Not cleanly distinguished from supervised learning by methodology
- **Weird nugget:** Whether what we are doing is “unsupervised” often hangs precariously on what we choose to call a “label”)

Generative AI section of the class

- Generative AI today is a weird category
- Pulls together two separate concerns
    1. Web-scale unsupervised (or loosely supervised) training.
    2. Generative models, that produce not just point predictions of class labels, regression targets, but representing distributions over complex media artifacts (documents, images)
        a) Unsupervised learning
        b) Distribution learning
        c) Latent variable modeling
        d) Dimensionality reduction

Foundation models

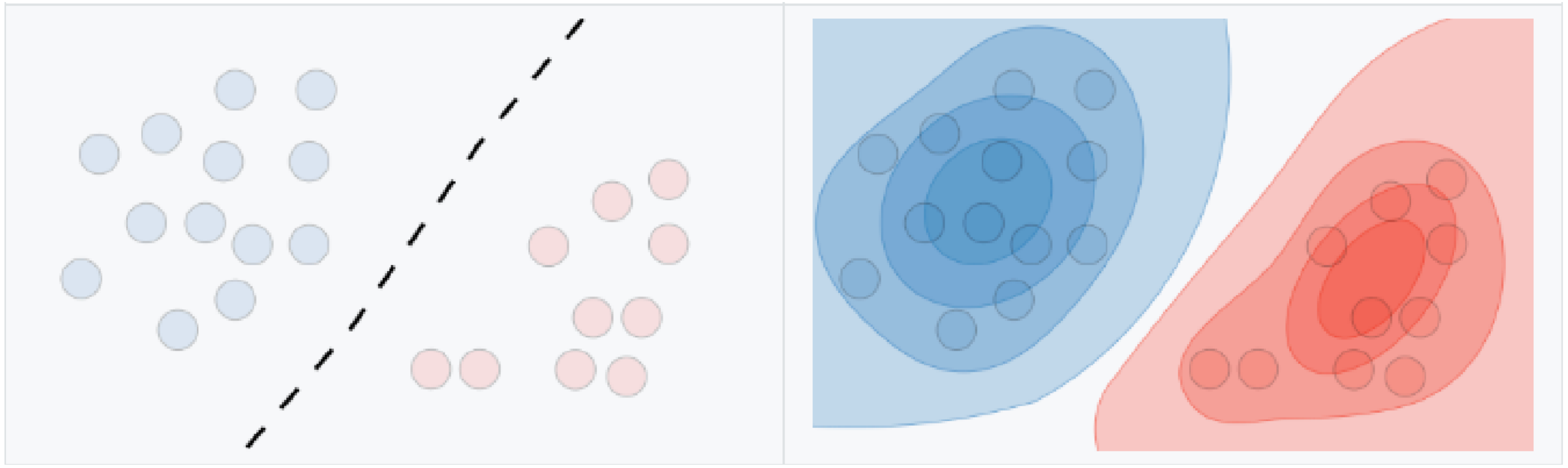
- Web-scale unsupervised learning
- Products → surprisingly general-purpose models used for many downstream tasks
- Single model, unsupervised learning on massive amountso f unstructured data used in many downstream tasks
- Examples:
  - Language models:
    - state of the art: GPT–4, PaLM2, Claude2
    - state of open source: LLaMA2, Zephyr, Falcon, MPT
  - Text→Image models: Midjourney, Stable Diffusion, DALL-E 3
  - Joint-embedding models: CLIP, Open-CLIP

Generative Modeling 101

- A generative model on X_1, … , X_n is a statistical model of the joint distribution of all modeled variables $P(X_1, \dots, X_n)$
- Classic generative models:
    - Gaussian Mixture Models
    - Probabilistic graphical models
        - Directed Acyclic Graphs (DAGs)
        - Hidden Markov Models (HMMs)
        - Markov Random Fields (MRFs)
- Classic intro to (un)supervised learning "given $\{(\boldsymbol{x}, y)\}$" vs "given $\{(\boldsymbol{x})\}$"
    - But with generative modeling, any x_j, can be your "*y*"
    - Different interpretation: we're not <u>given less</u>, but rather asked to <u>do more</u>

Generative vs Discriminative Modeling

- Discriminative models focus on a particular conditional p(y|x)
- Typically, we only care about getting a point estimate, most likely $\hat{y}$
- With generative models we typically want to “learn the distribution”



A very simple generative models

- Data consists of a single binary feature
- Draw some data from the real world

$$\mathcal{D} = \{x_1, ..., x_n\}$$

- Pose an appropriately expressive model of the data distribution

$$\mathcal{D} = \{x_1, ..., x_n\} \sim \text{Bernoulli}(\theta)$$

- Estimate the parameters of the model:

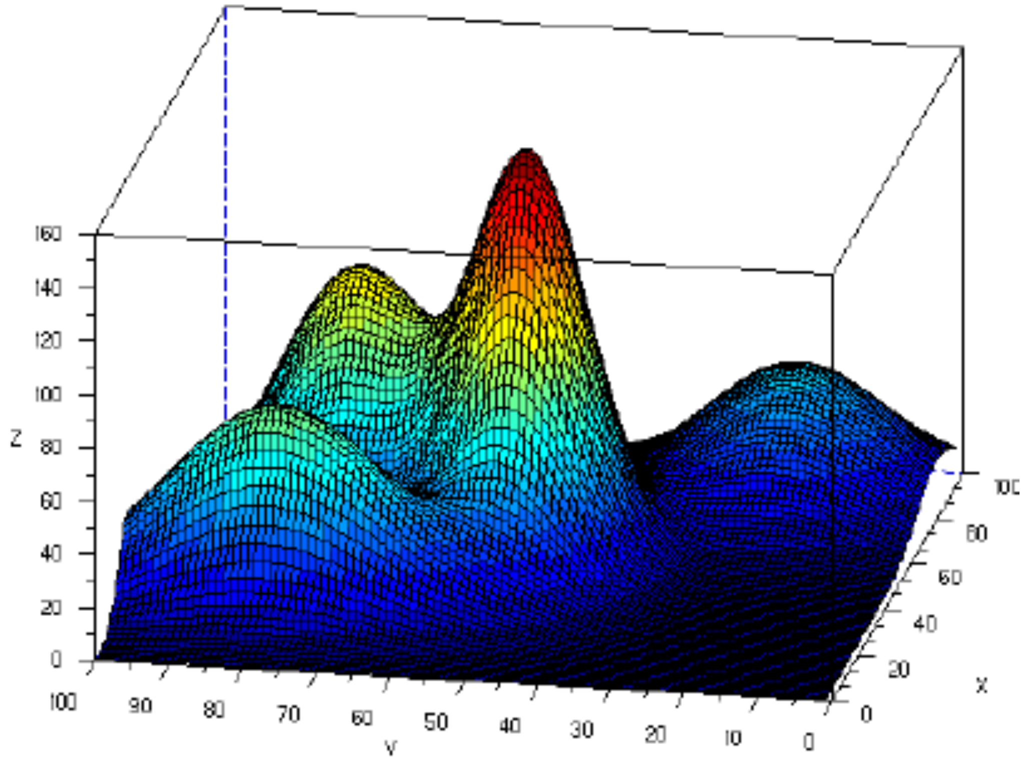
$$\hat{\theta}_{\text{MLE}}(\mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} x_i$$

- Now we can draw synthetic data from our generative model:

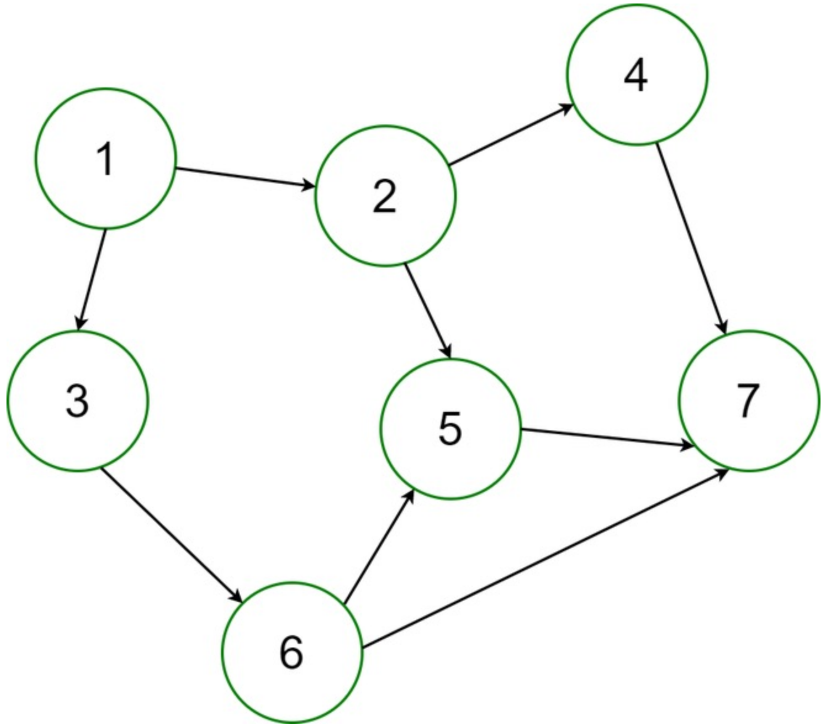
$$x'_i, ..., x'_m \sim \text{Bernoulli}(\hat{\theta})$$

Slightly more interesting: GMM

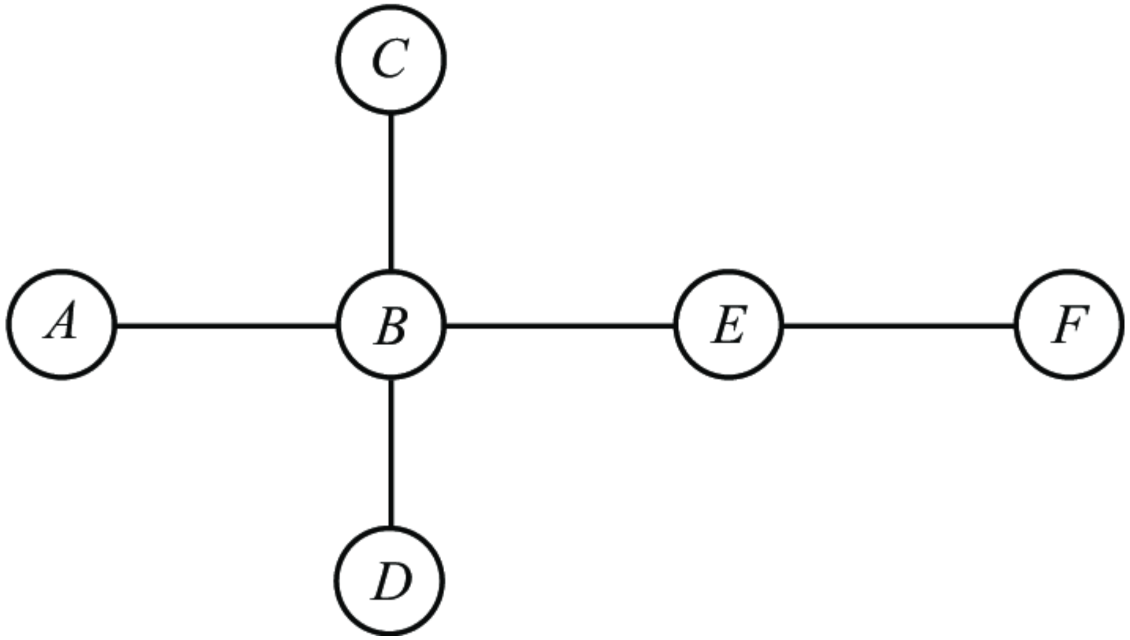
$$p(x) = \sum_{j=1}^{K} w_j \cdot \mathcal{N}(x|\mu_j, \Sigma_j)$$



Some Familiar Generative Models



Bayes nets



Markov Random Fields

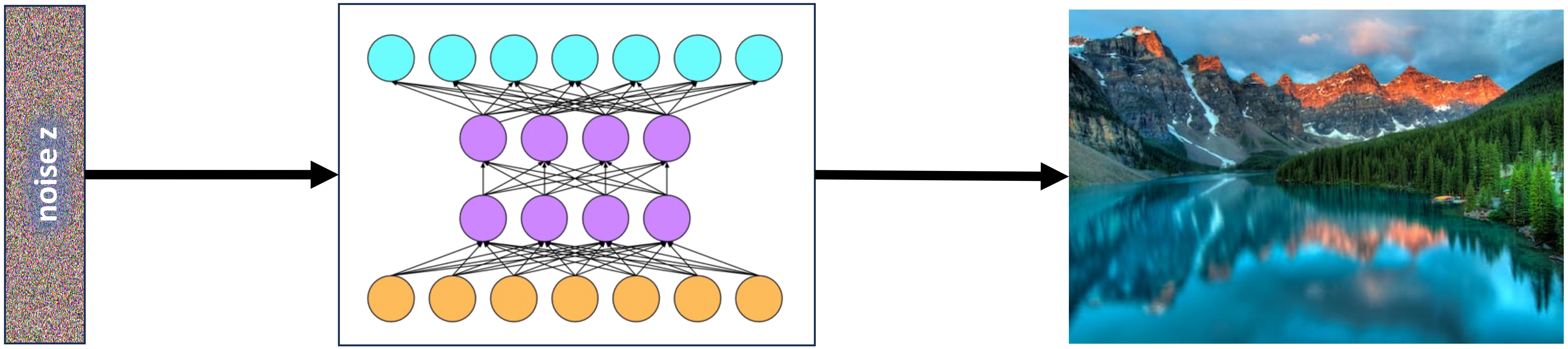
Things We ~~Can~~ Might do w Generative Models

- Density estimation: assessing the likelihood p(**x**)
- Sampling: draw data according to learned distribution x ~ p(**x**)
- Inference:
  - Compute arbitrary marginal likelihoods $p(x_1=a, x_3=b, x_7=c)$
  - Compute arbitrary conditional likelihoods $p(x_{10}=a|x_3=b, x_{17}=c)$

Can’t Always Have it All

- Some models make some operations easy but others hard
- For DAGs:
  - Sampling is easy
  - Exact likelihood calculation for a fully observed data point is easy
  - General marginalization / conditional inference is in general hard
  - Typically, people rely on approximate inference algorithms
- For MRFs
  - Expressive set of conditional independencies
  - Exact inference is #P-complete, generally intractable
  - Approximated with MCMC methods

Deep Latent Generative Model

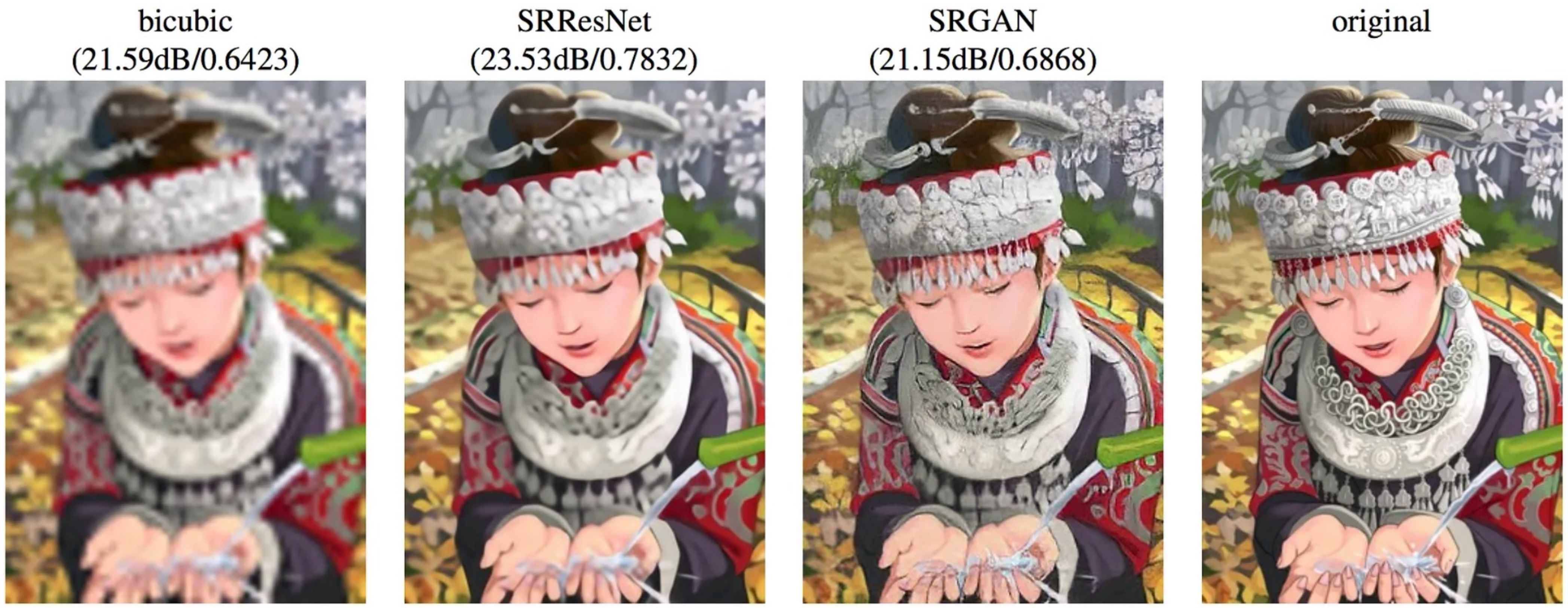


key idea: sample z from some "nice" distribution, learn a mapping from $z \sim p(z)$ to samples $G(z) \sim p_{data}(x)$

Dominant Methods

- **Variational Autoencoders**
- **Generative Adversarial Networks**
- Normalizing flows, score matching, deep diffusion

Super-resolution (Ledig et al. 2017)



Infilling

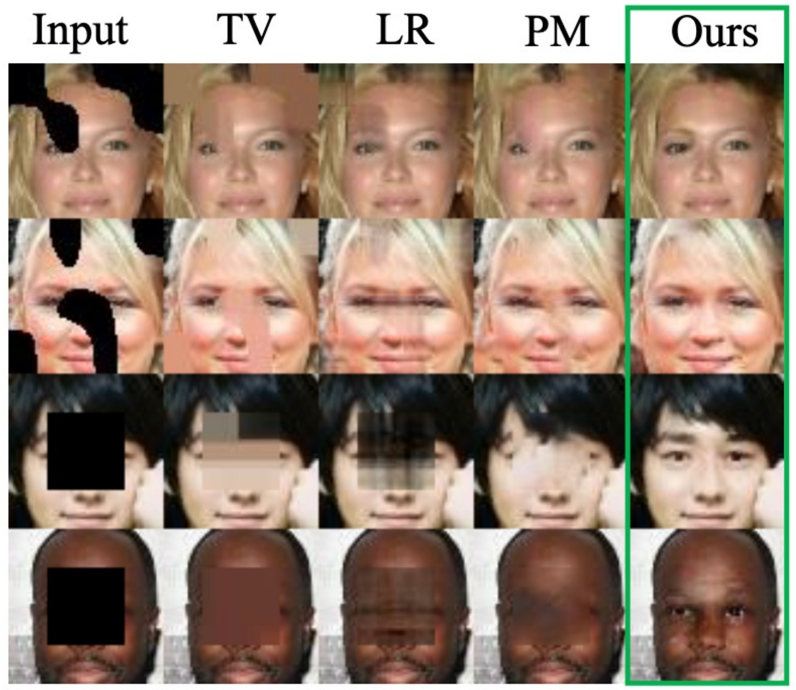


Figure 1. Semantic inpainting results by TV, LR, PM and our method. Holes are marked by black color.

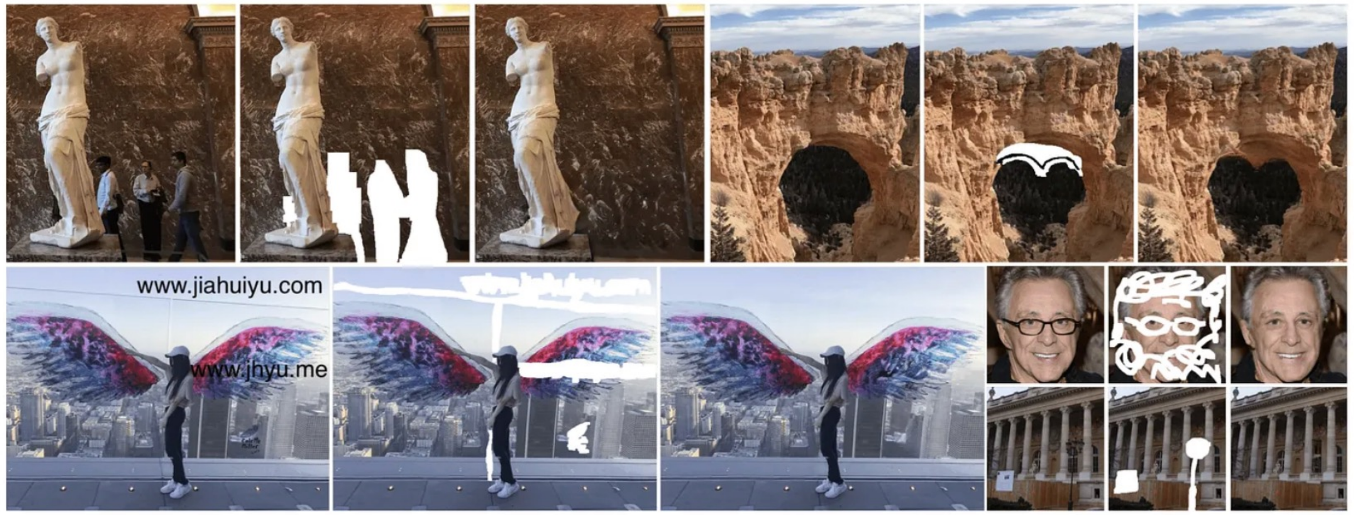


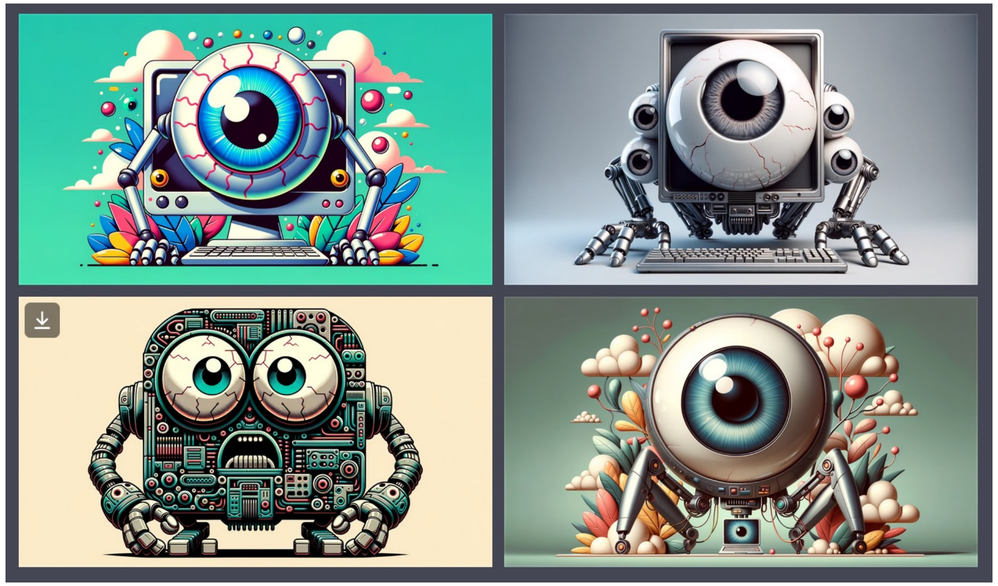
Figure 1. Examples of Image Inpainting Applications. Image by Jiahui Yu et al. from their paper, DeepFill v2 [13]

Yeh et al. 2016

Yu et al 2019

Instantaneous Illustrations to Spec

“computer droid with giant funny eyeballs”



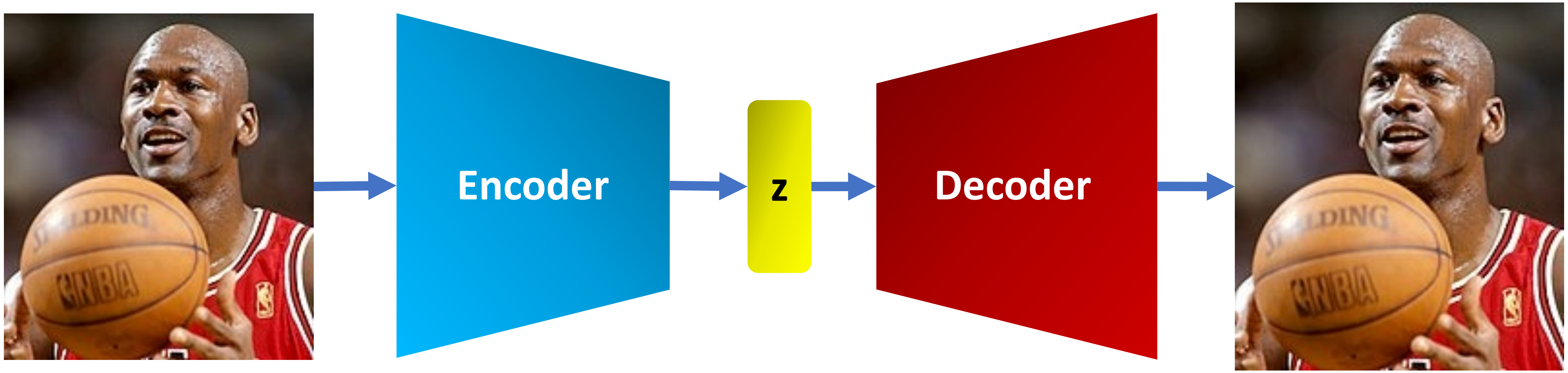
DALL-E3 (via ChatGPT)

A group of teddy bears in suit in a corporate office celebrating the birthday of their friend. There is a pizza cake on the desk.

[“Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, Saharia et al., 2022]()

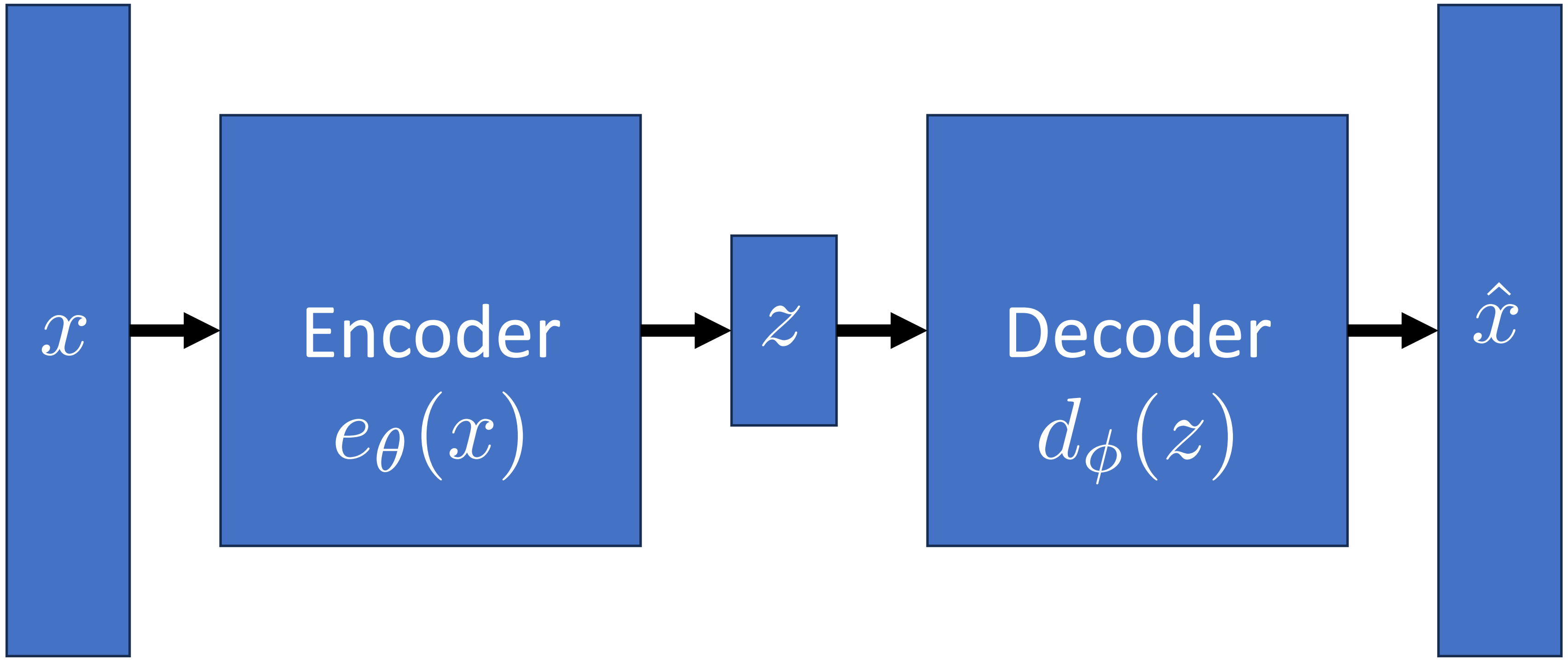
(Google Imagen)

Autoencoders



1. Map the input to a compact representation **z**
2. Use the representation **z** to **reconstruct** the input
3. Can learn encoder & decoder mappings **without any labels**
4. Learned **z** picks up the most important **factors of variation**

Autoencoders, abstractly



Neural Network Autoencoders

- Typically trained to reduce dimensionality
- Intuition:
  - Learn compact (non-linear) representation
  - Captures important factors of variation
- Objective: minimize reconstruction error (like PCA!)

$$\min_{\theta,\phi} ||x - \hat{x}||^2$$

$$= \min_{\theta,\phi} ||x - d_\phi(e_\theta(x))||^2$$

Autoencoders for Semi-supervised Learning

1. Fit an autoencoder using unlabeled data

$$\{x_i\}_{i=1}^{n} \to e_\theta$$

2. Wrap compose the encoder with a task-appropriate output layer (with randomly initialized parameters):

$$f_\phi(e_\theta)$$

3. Starting from this initialization, optimize on new task-specific data

$$\min_{\phi,\theta} \sum_{x_i, y_i \in \mathcal{D}} \mathcal{L}\left(y_i, f_\phi(e_\theta)(x_i)\right)$$

Variant: Denoising Autoencoders

- Encode input
- Randomly drop hidden nodes
- Reconstruct from corrupted code
- Denoising makes representation learning non-trivial
- Requires redundancy in latent z

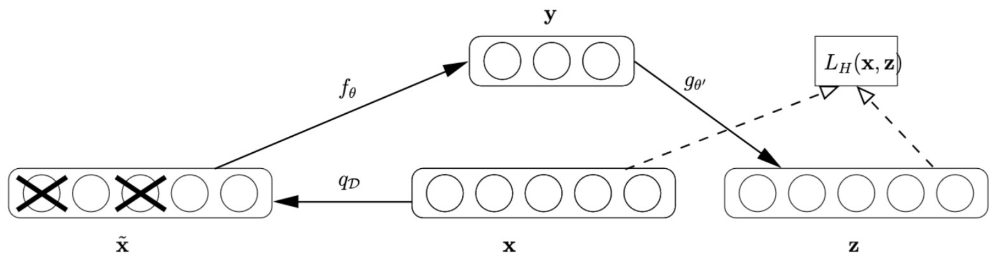


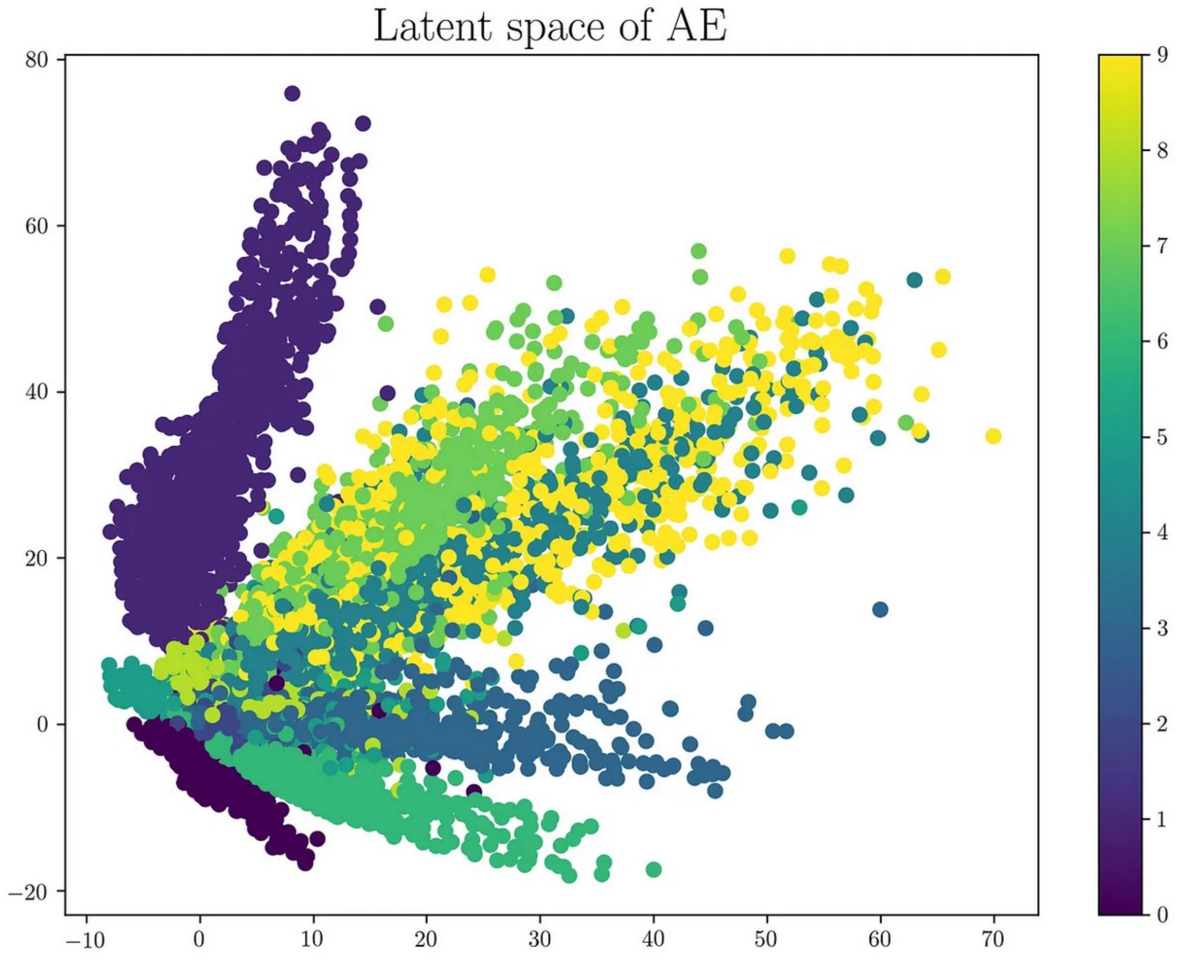
Figure 1. An example $\mathbf{x}$ is corrupted to $\tilde{\mathbf{x}}$. The autoencoder then maps it to $\mathbf{y}$ and attempts to reconstruct $\mathbf{x}$.

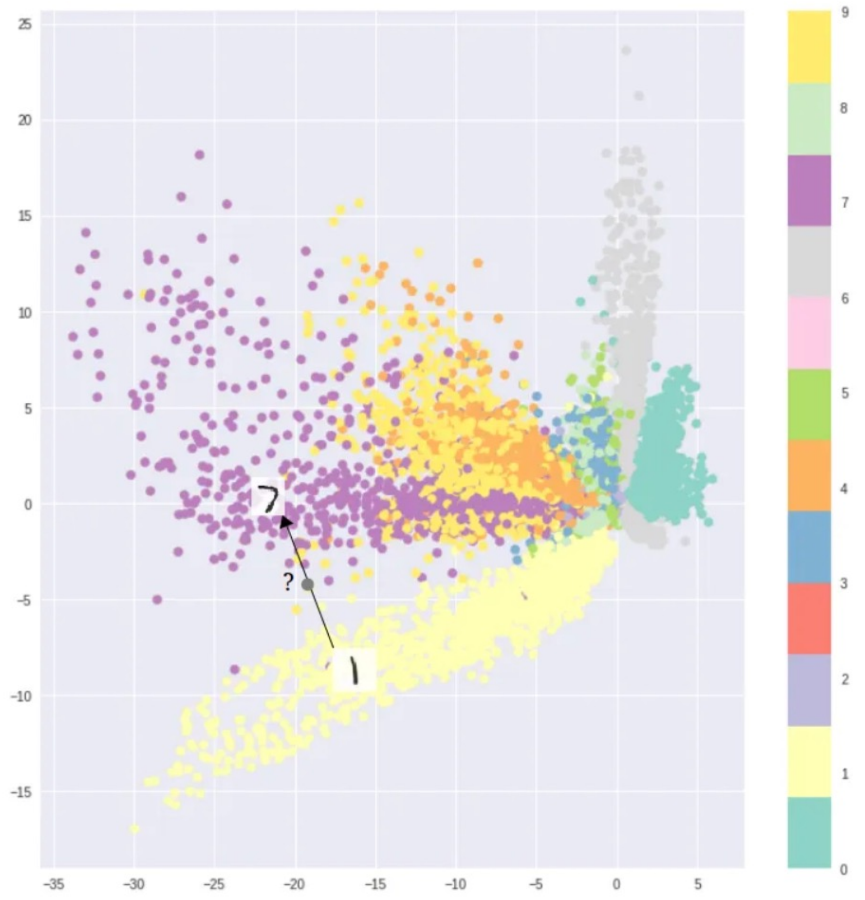
"Extracting and Composing Robust Features with Denoising Autoencoders" Vincent et al., ICML 2008

Variant: Sparse Autoencoders

- Have large number of hidden nodes but coerce sparsity in latent code
- Methods:
  1. k-sparse autoencoder: clamp all but k-highest hidden values to 0
  2. Relaxed sparse autoencoder: train with regularization loss to flush high fraction of hidden nodes to 0

Why Not Sample from a Vanilla Autoencoder?

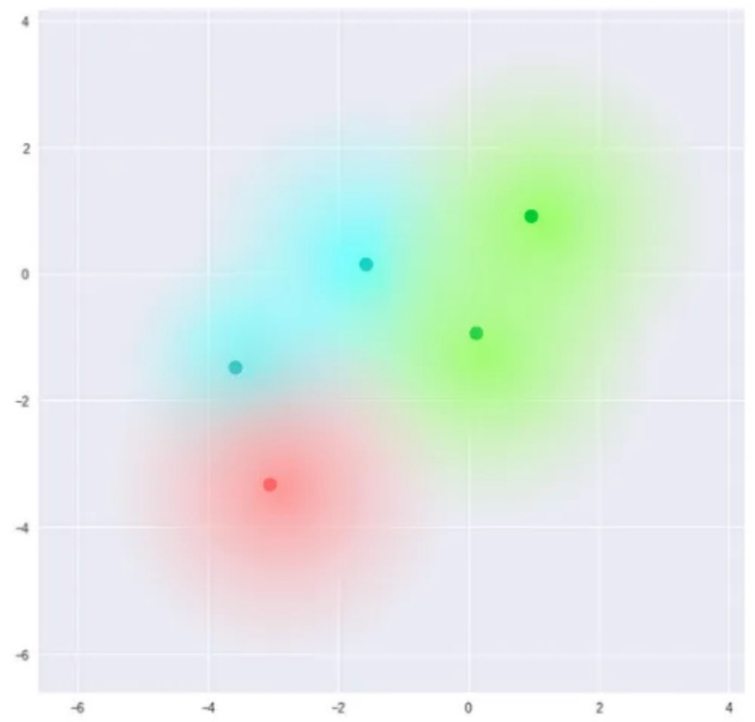




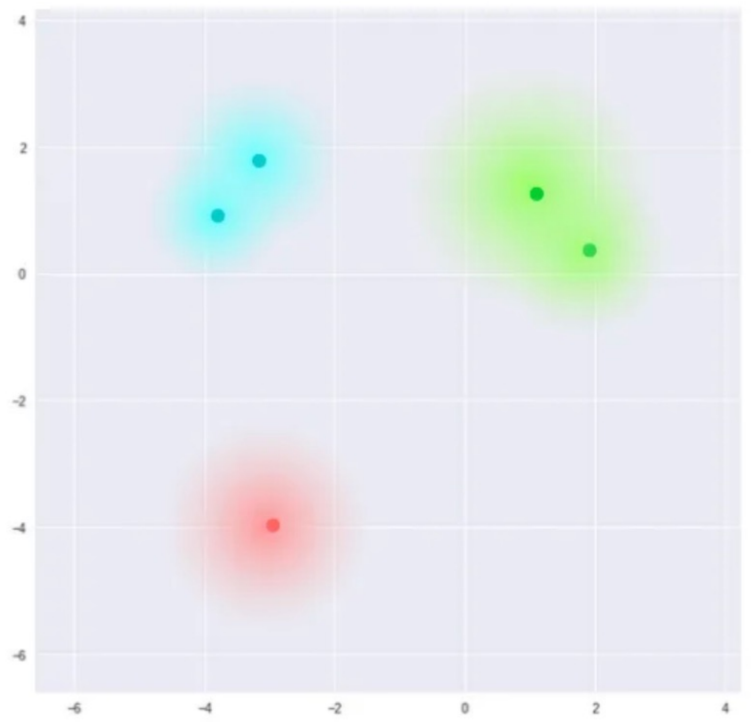
Blog post by Aqeel Anwar 2021

Blog post by Irhum Shafkat 2018

Autoencoder Latent Space



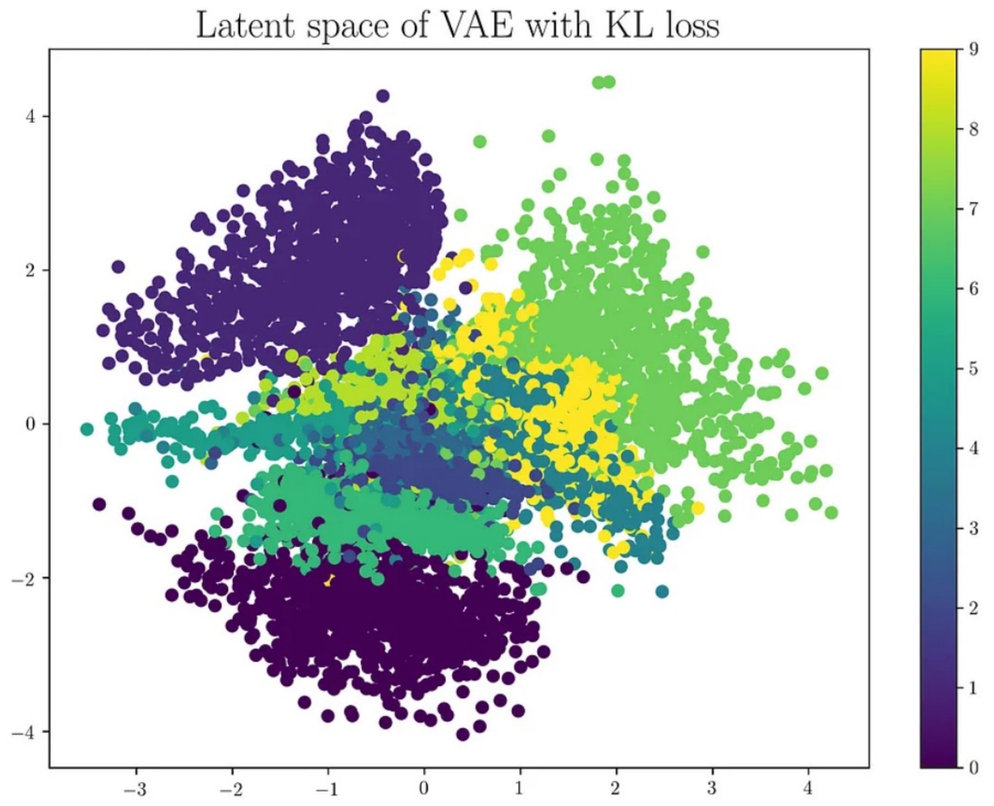
What we require

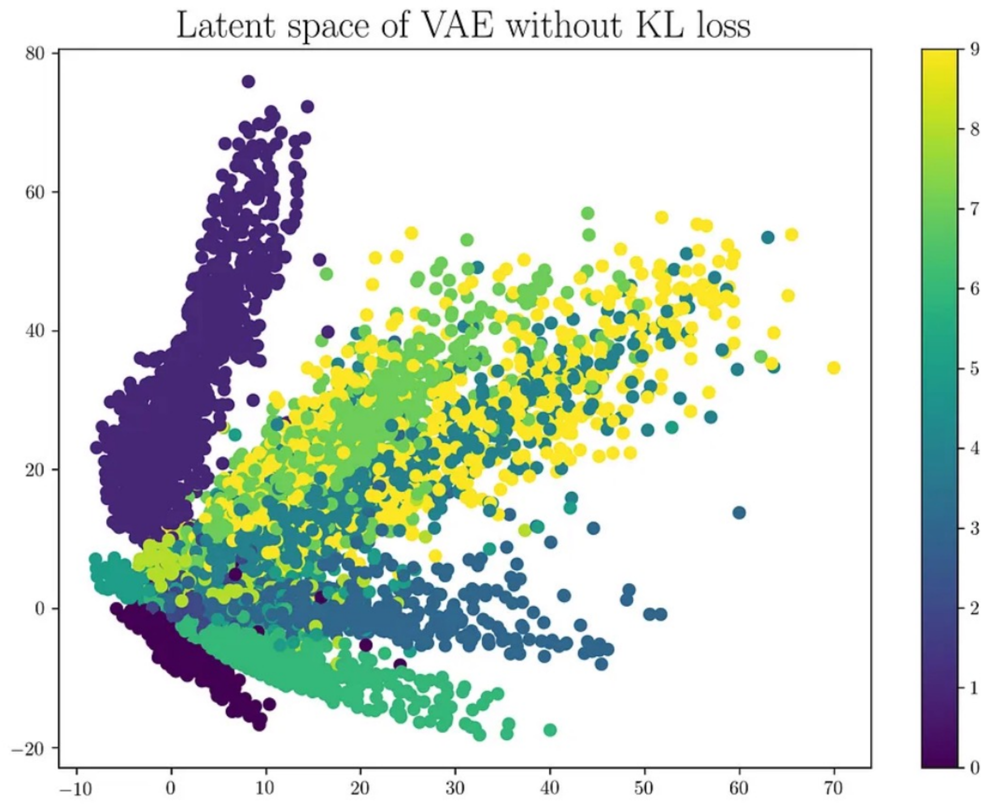


What we may inadvertently end up with

Blog post by Irhum Shafkat 2018

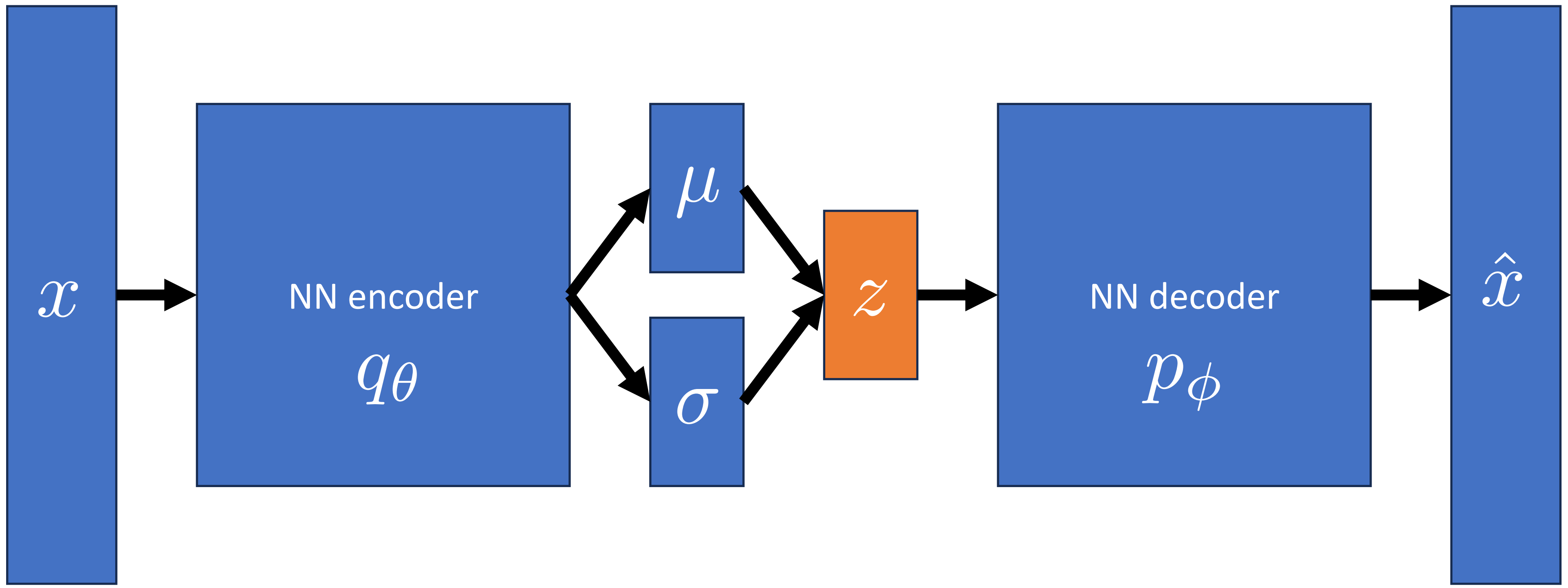
VAE vs Autoencoder Latent Space Visualized





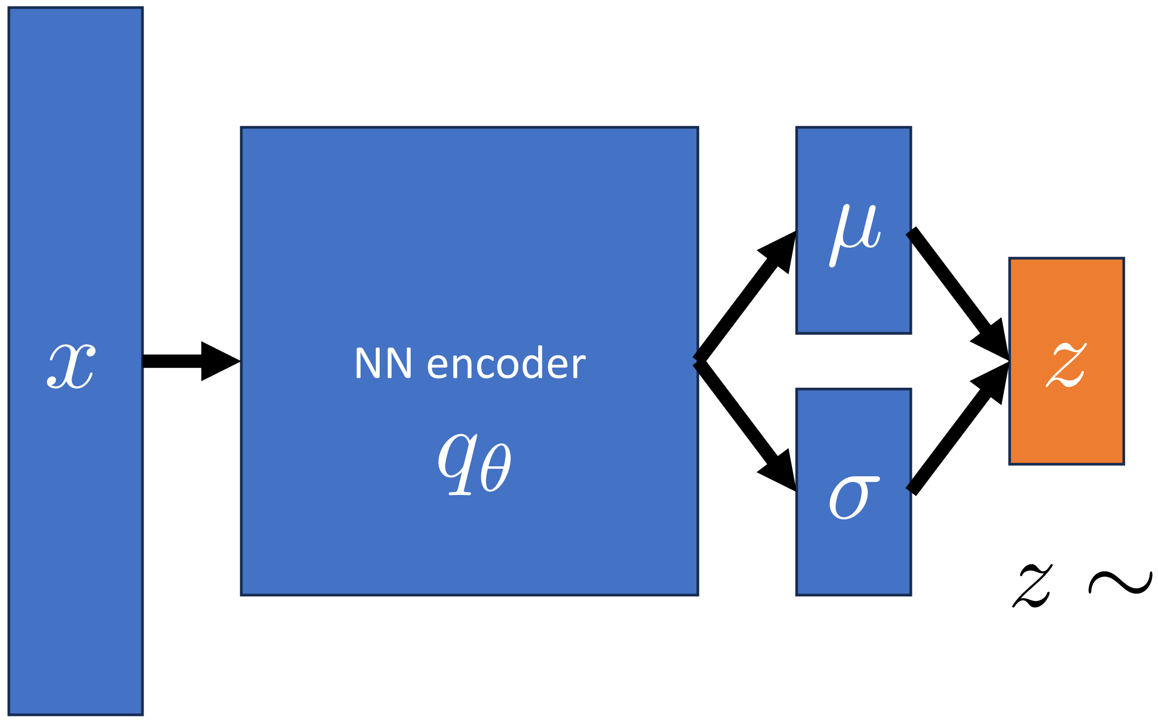
Blog post by Aqeel Anwar 2021

Variational Autoencoder



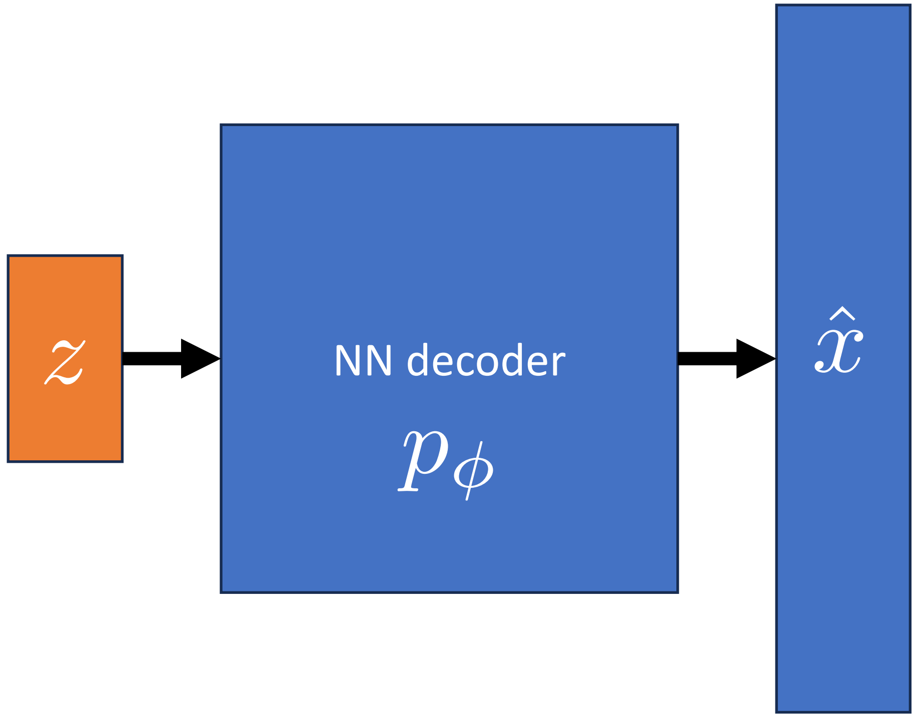
Auto-Encoding Variational Bayes (Kingma & Welling 2013)
Stochastic Backpropagation and Approximate Inference in Deep Generative Models (Rezende et al. 2014)

Inferring the latent Code



$$z \sim \mathcal{N}(\mu_\theta(x), \mathrm{diag}(\sigma^2_\theta(x)))$$

Generating from VAE



$$z \sim \mathcal{N}(0, I) \qquad x = p_\phi(x|z)$$

VAE (Neural Network Perspective)

- Encoder: $q_\theta(z|x)$ $z \sim \mathcal{N}(\mu_\theta(x), \mathrm{diag}(\sigma_\theta^2(x)))$
- Decoder: $p_\phi(x|z)$
- Objective:
  - minimize reconstruction error + regularization term

$$\mathcal{L}(\theta, \phi) = \sum_{i=1}^{n} -\mathbb{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] + \mathbb{KL}(q_\theta(z|x_i)||p(z))$$

Backprop through VAE

- Recall, the forward pass through a VAE is stochastic
- How can we backpropagate through a stochastic operation?
- Solve with “reparameterization trick” treat noise as an input
- Original form:

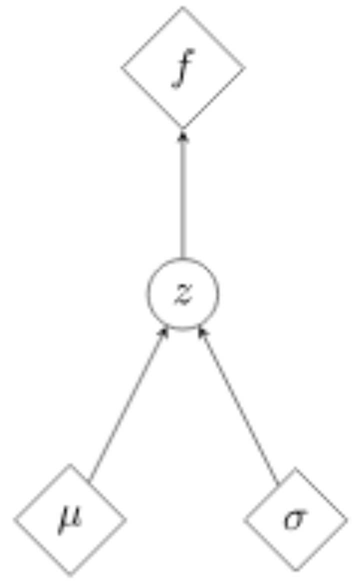
$$z \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$$

- Reparameterized form:

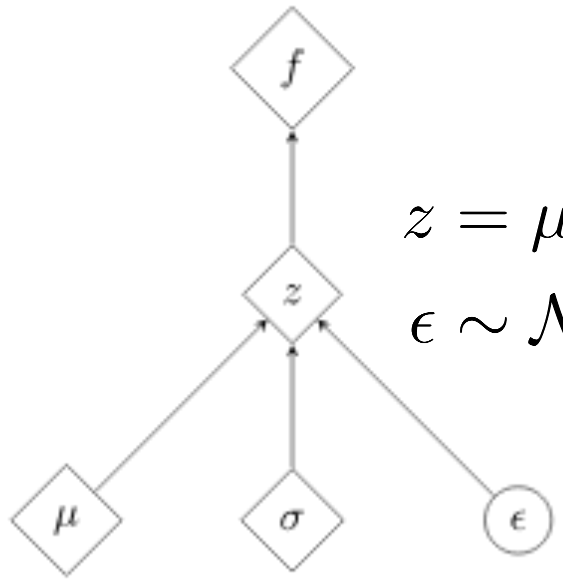
$$z = \mu + \sigma \odot \epsilon$$

$$\epsilon \sim \mathcal{N}(0, I)$$

The Reparameterization Trick Visualized



A: Without Reparameterization



$$z = \mu + \sigma \odot \epsilon$$
$$\epsilon \sim \mathcal{N}(0, I)$$

B: With Reparameterization

Probabilistic Perspective

- Joint probability: $p(x, z) = p(z)p(x|z)$

- Generative process:
  - Draw $z_i \sim p(z)$ Assume $p(z)$ multivariate Gaussian
  - Draw $x \sim p(x|z_i)$

- Idea: learn parameters to maximize the *evidence*, i.e., likelihood of our data

$$p_\phi(x) = \int p(z)p_\phi(x|z)dz$$

- Problem: intractable, cannot evaluate for all values *z*

Deriving the variational lower bound

$$\log p_\phi(x_i) = \mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i)]$$

no dependence on z

Deriving the variational lower bound

$$\log p_\phi(x_i) = \mathbb{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i)]$$

no dependence on z

$$= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\right]$$

Bayes

Deriving the variational lower bound

$$
\begin{aligned}
\log p_\phi(x_i) &= \mathbb{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i)] \\
&= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\right] \\
&= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\frac{q_\theta(z|x_i)}{q_\theta(z|x_i)}\right]
\end{aligned}
$$

no dependence on z

Bayes

multiply by constant

Deriving the variational lower bound

$$\begin{aligned}
\log p_\phi(x_i) &= \mathbb{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i)] \\
&= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\right] \\
&= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\frac{q_\theta(z|x_i)}{q_\theta(z|x_i)}\right]
\end{aligned}$$

no dependence on z

Bayes

multiply by constant

Deriving the variational lower bound

$$\log p_\phi(x_i) = \mathbb{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i)]$$

no dependence on z

$$= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\right]$$

Bayes

$$= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\frac{q_\theta(z|x_i)}{q_\theta(z|x_i)}\right]$$

multiply by constant

$$= \mathbb{E}_z\left[\log p_\phi(x_i|z)\right] - \mathbb{E}_z\left[\log \frac{q_\theta(z|x_i)}{p_\phi(z)}\right] + \mathbb{E}_z\left[\log \frac{q_\theta(z|x_i)}{p_\phi(z|x_i)}\right]$$

logarithms

Deriving the variational lower bound

$$\log p_\phi(x_i) = \mathbb{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i)]$$

no dependence on z

$$= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\right]$$

Bayes

$$= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\frac{q_\theta(z|x_i)}{q_\theta(z|x_i)}\right]$$

multiply by constant

$$= \mathbb{E}_z\left[\log p_\phi(x_i|z)\right] - \mathbb{E}_z\left[\log \frac{q_\theta(z|x_i)}{p_\phi(z)}\right] + \mathbb{E}_z\left[\log \frac{q_\theta(z|x_i)}{p_\phi(z|x_i)}\right]$$

logarithms

$$= \mathbb{E}_z\left[\log p_\phi(x_i|z)\right] - \mathbb{KL}(q_\theta(z|x_i)||p_\phi(z)) + \mathbb{KL}(q_\theta(z|x_i)||p_\phi(z|x_i))$$

Deriving the variational lower bound

$$\begin{aligned}
\log p_\phi(x_i) &= \mathbb{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i)] \\
&= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\right] \\
&= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\frac{q_\theta(z|x_i)}{q_\theta(z|x_i)}\right] \\
&= \mathbb{E}_z\left[\log p_\phi(x_i|z)\right] - \mathbb{E}_z\left[\log \frac{q_\theta(z|x_i)}{p_\phi(z)}\right] + \mathbb{E}_z\left[\log \frac{q_\theta(z|x_i)}{p_\phi(z|x_i)}\right] \\
&= \mathbb{E}_z\left[\log p_\phi(x_i|z)\right] - \mathbb{KL}(q_\theta(z|x_i)||p_\phi(z)) + \mathbb{KL}(q_\theta(z|x_i)||p_\phi(z|x_i))
\end{aligned}$$

no dependence on z

Bayes

multiply by constant

logarithms

reconstruction loss

regularizer on latent encodings

> 0

Deriving the variational lower bound

$$\begin{aligned}
\log p_\phi(x_i) &= \mathbb{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i)] \\
&= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\right] \\
&= \mathbb{E}_z\left[\log \frac{p_\phi(x_i|z)p_\phi(z)}{p_\phi(z|x_i)}\frac{q_\theta(z|x_i)}{q_\theta(z|x_i)}\right] \\
&= \mathbb{E}_z\left[\log p_\phi(x_i|z)\right] - \mathbb{E}_z\left[\log \frac{q_\theta(z|x_i)}{p_\phi(z)}\right] + \mathbb{E}_z\left[\log \frac{q_\theta(z|x_i)}{p_\phi(z|x_i)}\right] \\
&= \underbrace{\mathbb{E}_z\left[\log p_\phi(x_i|z)\right] - \mathbb{KL}(q_\theta(z|x_i)||p_\phi(z))}_{\text{ELBO, easy to optimize, lower bound on data likelihood}} + \mathbb{KL}(q_\theta(z|x_i)||p_\phi(z|x_i))
\end{aligned}$$

no dependence on z

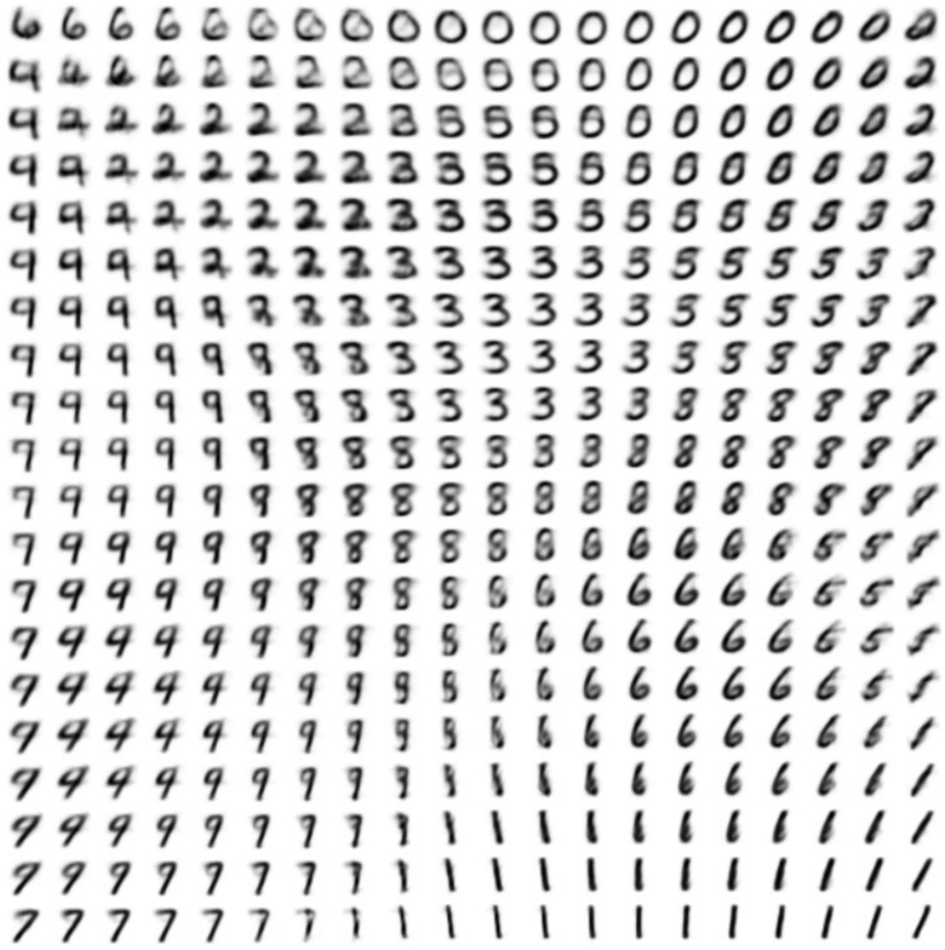
Bayes

multiply by constant

logarithms

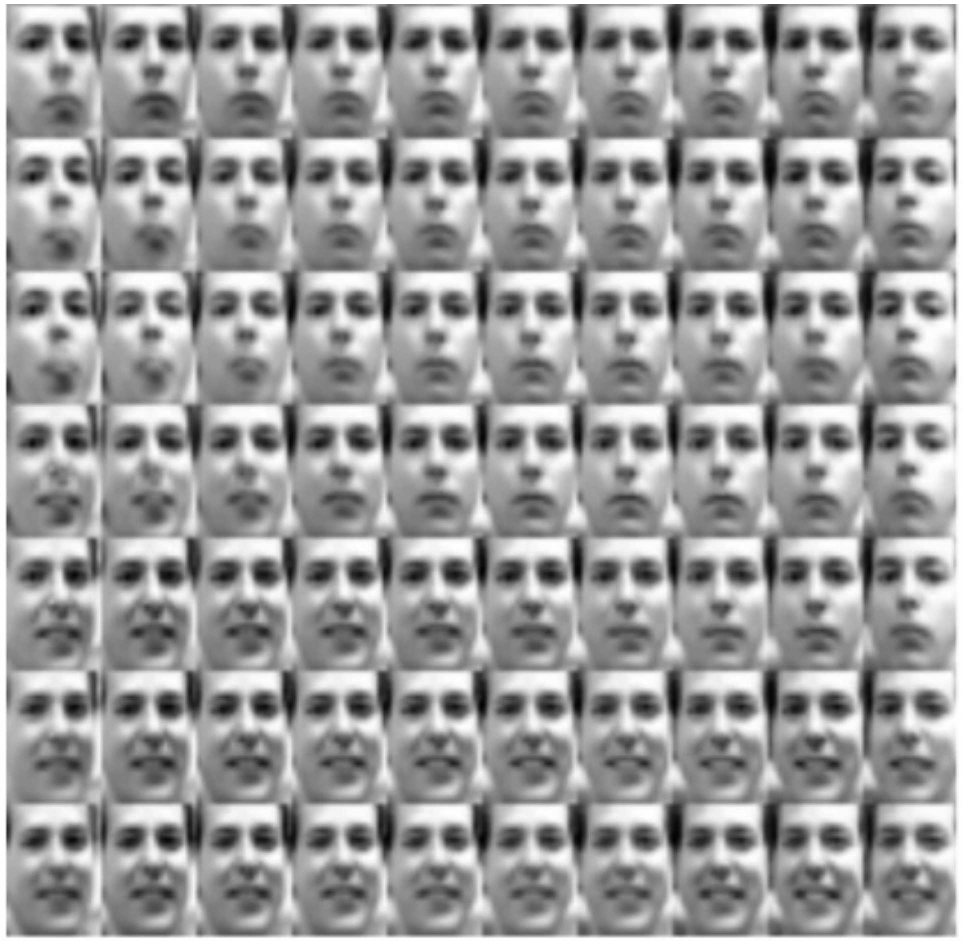
> 0

Learned 2D Manifold on MNIST



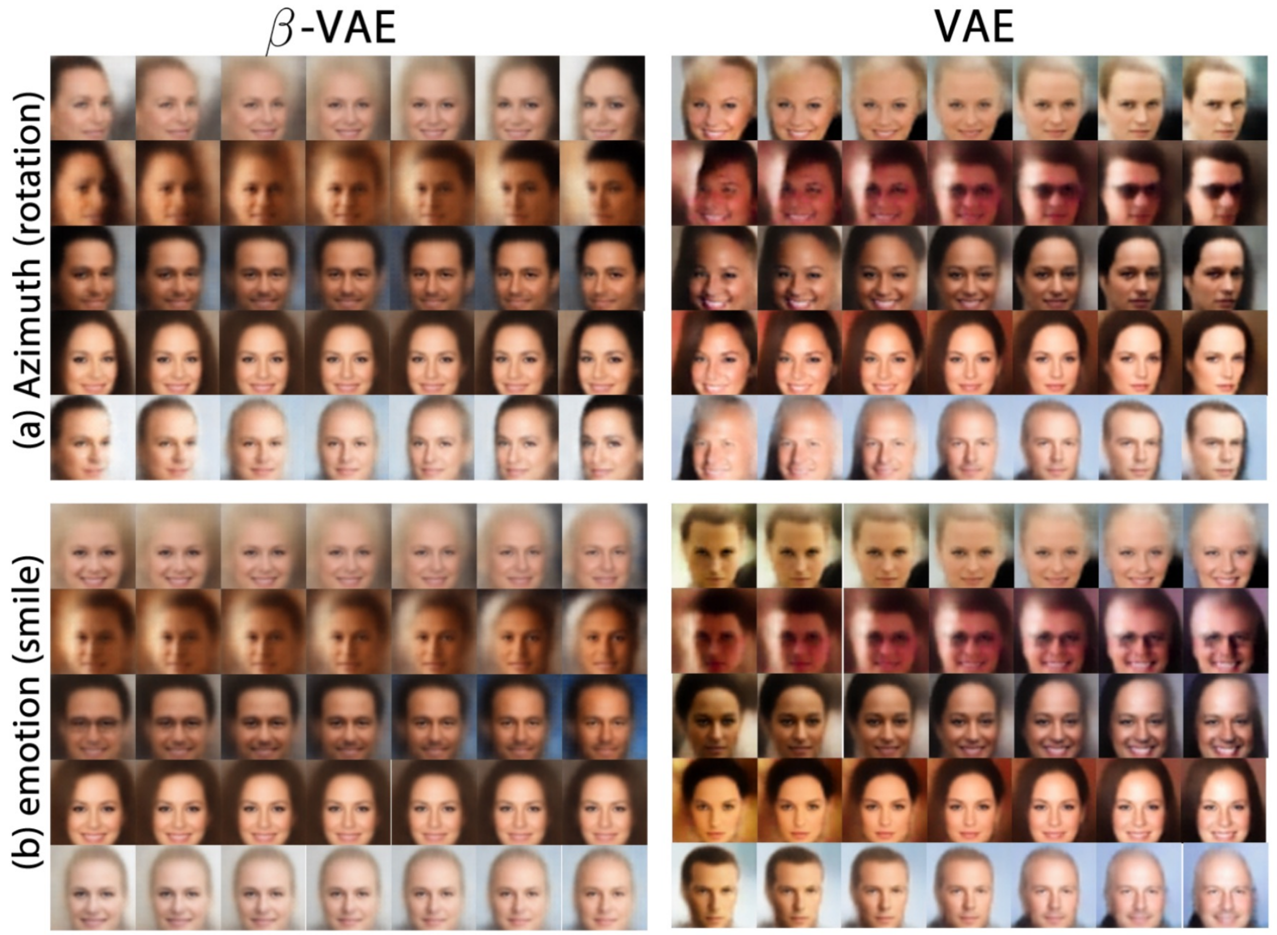
Auto-Encoding Variational Bayes (Kingma & Welling 2013)

Frey Face Manifold



Auto-Encoding Variational Bayes (Kingma & Welling 2013)

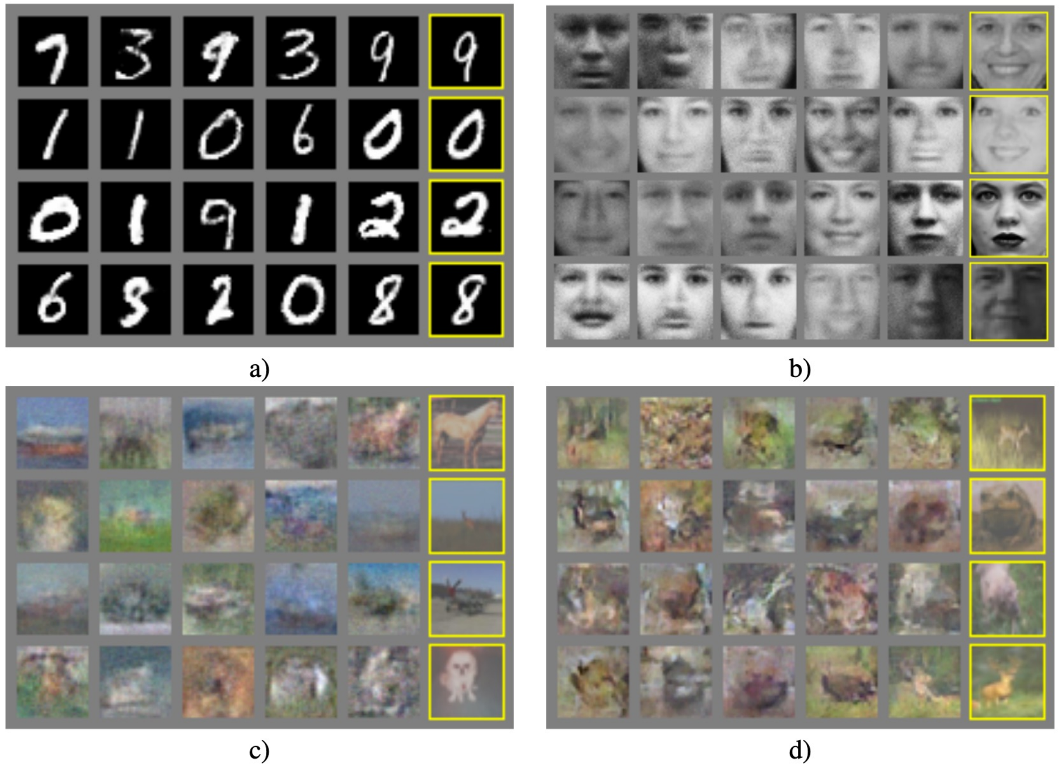
β-VAE



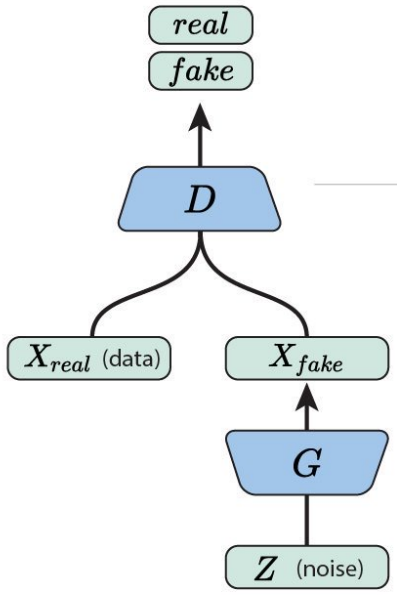
$$\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}, \beta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta \, D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

“Learning Basic Visual Concepts with a Constrained Variational Framework” Higgins et al. ICLR 2017

2014 Generative Adversarial Networks



Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.



The **discriminator** tries to distinguish genuine data from forgeries created by the generator.

The **generator** turns random noise into immitations of the data, in an attempt to fool the discriminator.

Figure credit: Chris Olah

Generative Adversarial Networks (Goodfellow et al. NeuRIPS 2014)

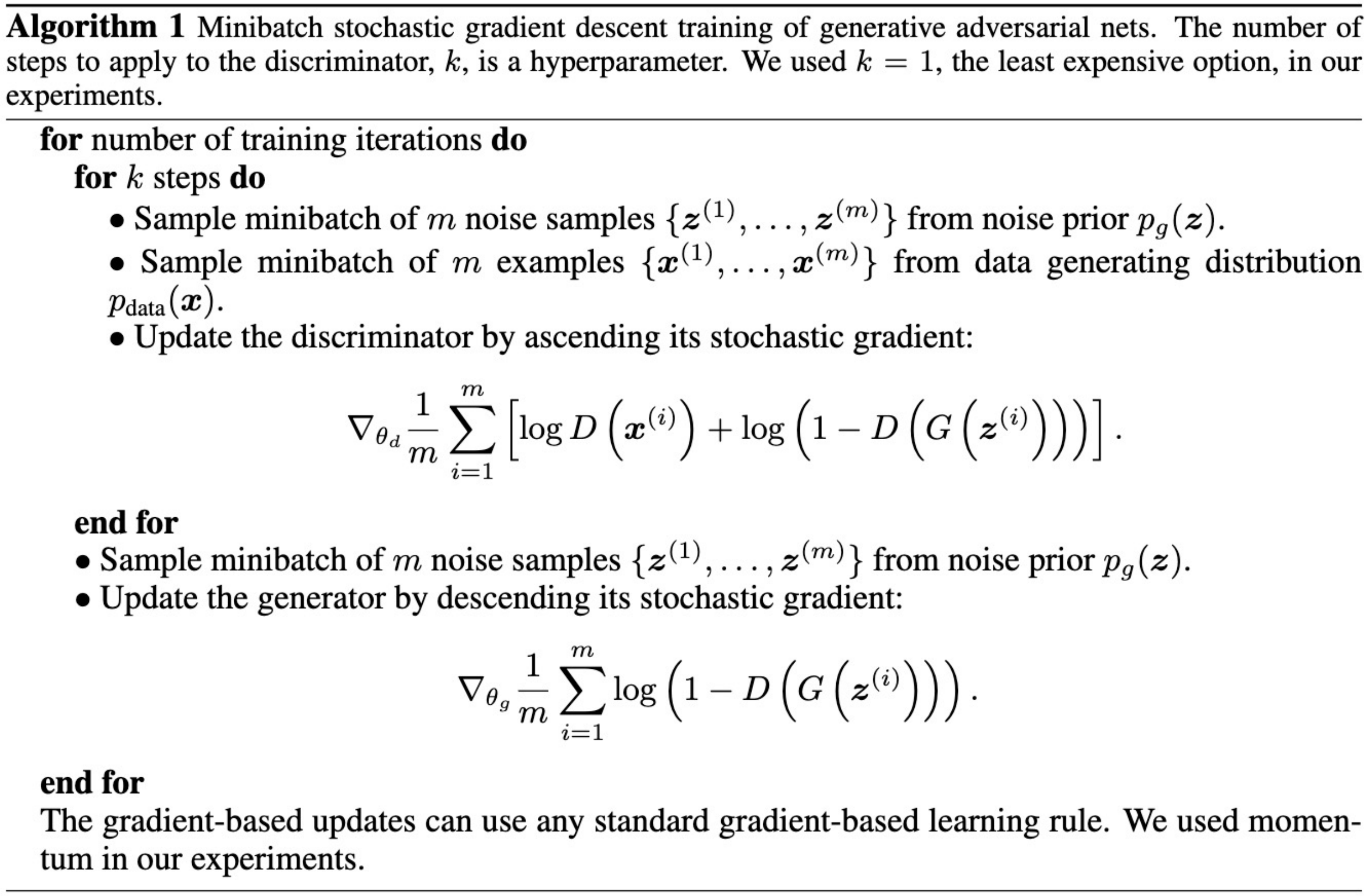
GANs: original training setup

- Dispense with representing likelihood, settle for implicit generation
- Two-player game:
  - Generator (w parameters θ_g)
  - Discriminator (w parameters θ_d)
- Minmax objective

$$\min_{\theta_g} \max_{\theta_d} \quad \mathbb{E}_{x \sim \text{data}} \log D_{\theta_g}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

- Discriminator works to classify real data as real, fake data as fake
- Generator works to make discriminator misclassify fake data as real

Pseudocode



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
        • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
        • Update the discriminator by ascending its stochastic gradient:
            $$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$
    end for
    • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Update the generator by descending its stochastic gradient:
        $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$
end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Iterative training, modified objective

1. Steps of gradient ascent to improve discriminator on

$$\max_{\theta_d} \quad \mathbb{E}_{x\sim\text{data}} \log D_{\theta_g}(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

2. Steps of gradient ascent to improve generator on different objective

$$\max_{\theta_g} \quad \mathbb{E}_{z\sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Optimizing GANs

- Ideally, at equilibrium:
  1. Generator exactly matches the real data distribution
  2. Discriminator cannot distinguish real from fake $D_{\theta_d}(x) = .5, \forall x$

- In reality:
  - Training is difficult and unstable
  - Balance between discriminator and generator hard to maintain
  - Susceptible to mode collapse: G(x) lacks diversity, D(x) super accurate

Mode Collapse

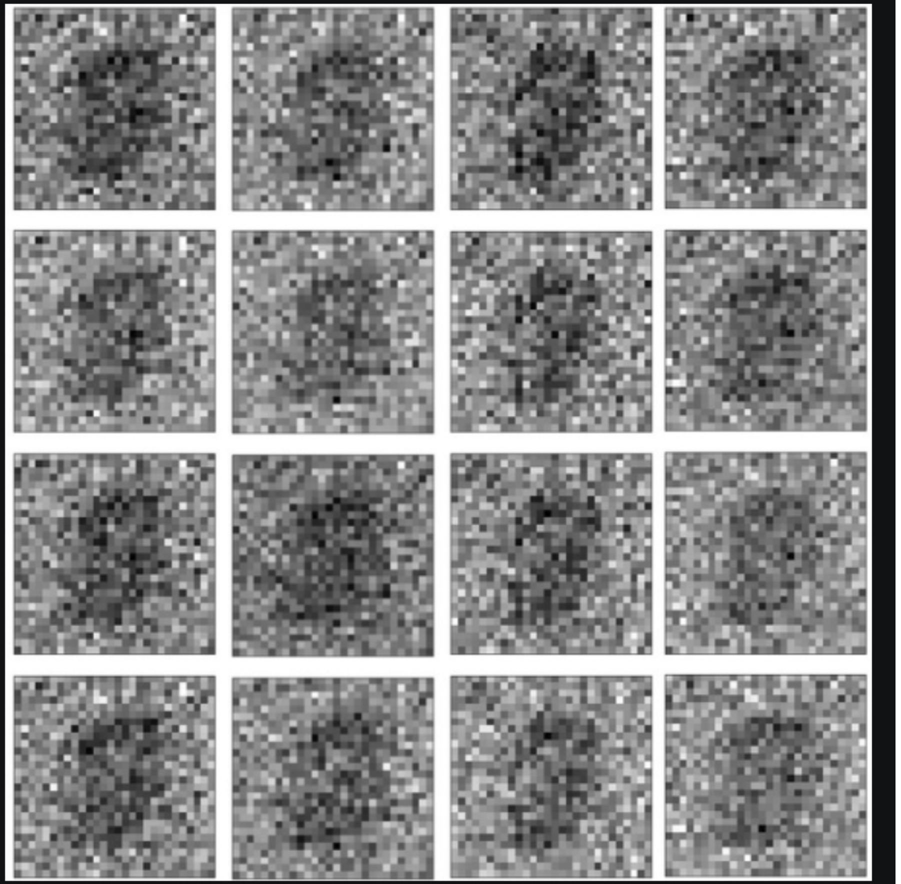


Fig. 4 – These are the output from one of the unstable training. This is on the same training code as above and slightly tweaked hyperparameters, but even after 300 epochs, you can see how bad our images are – an example of convergence failure | Source: Author

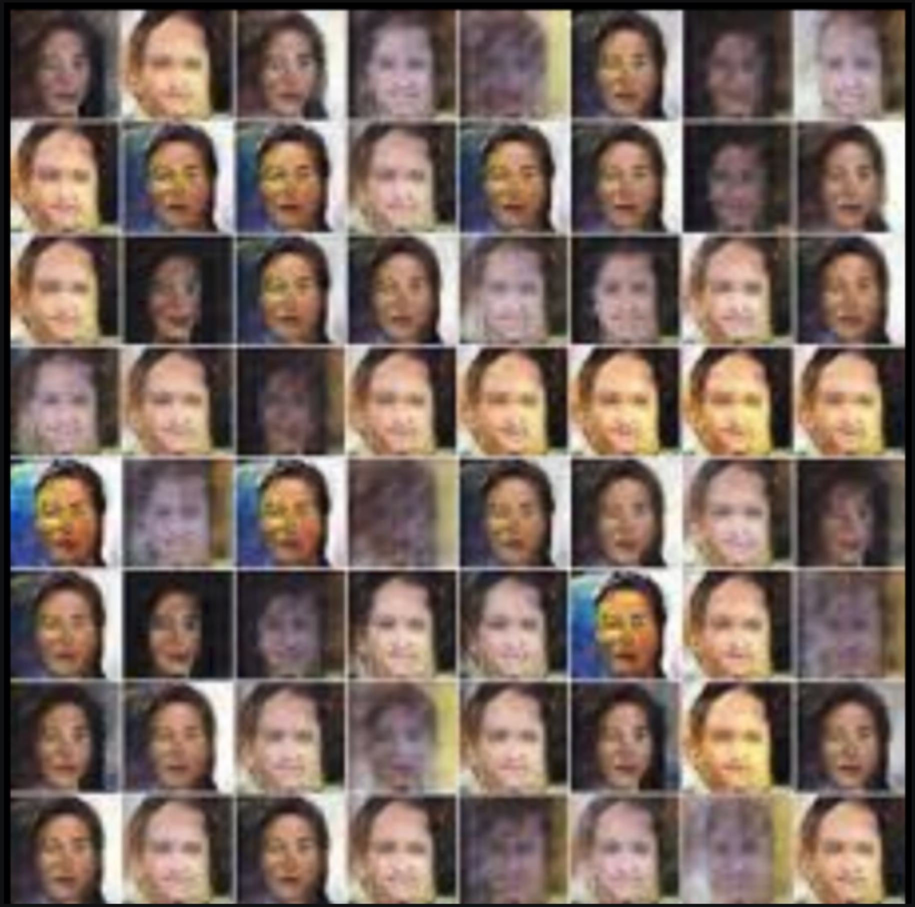
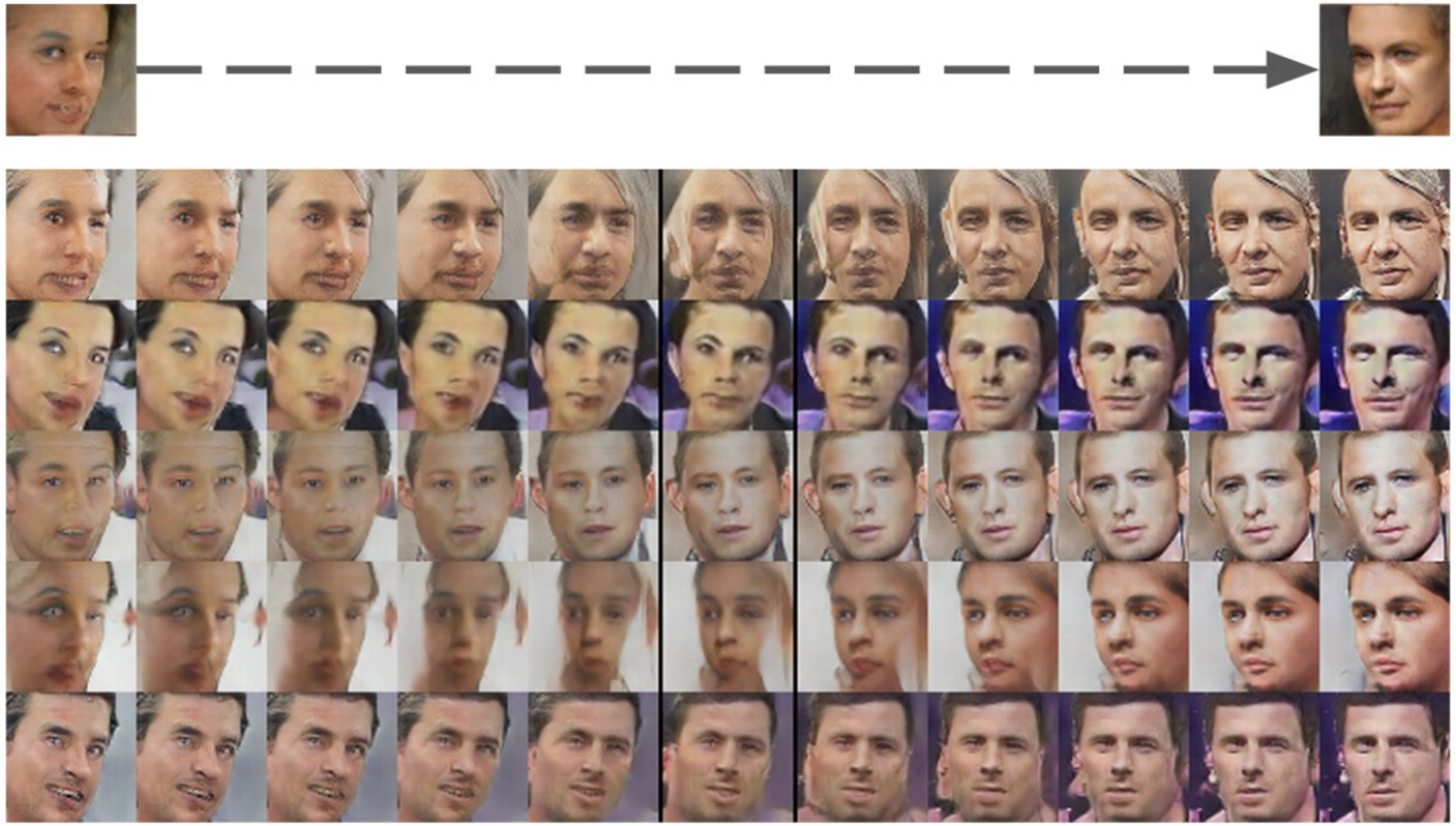


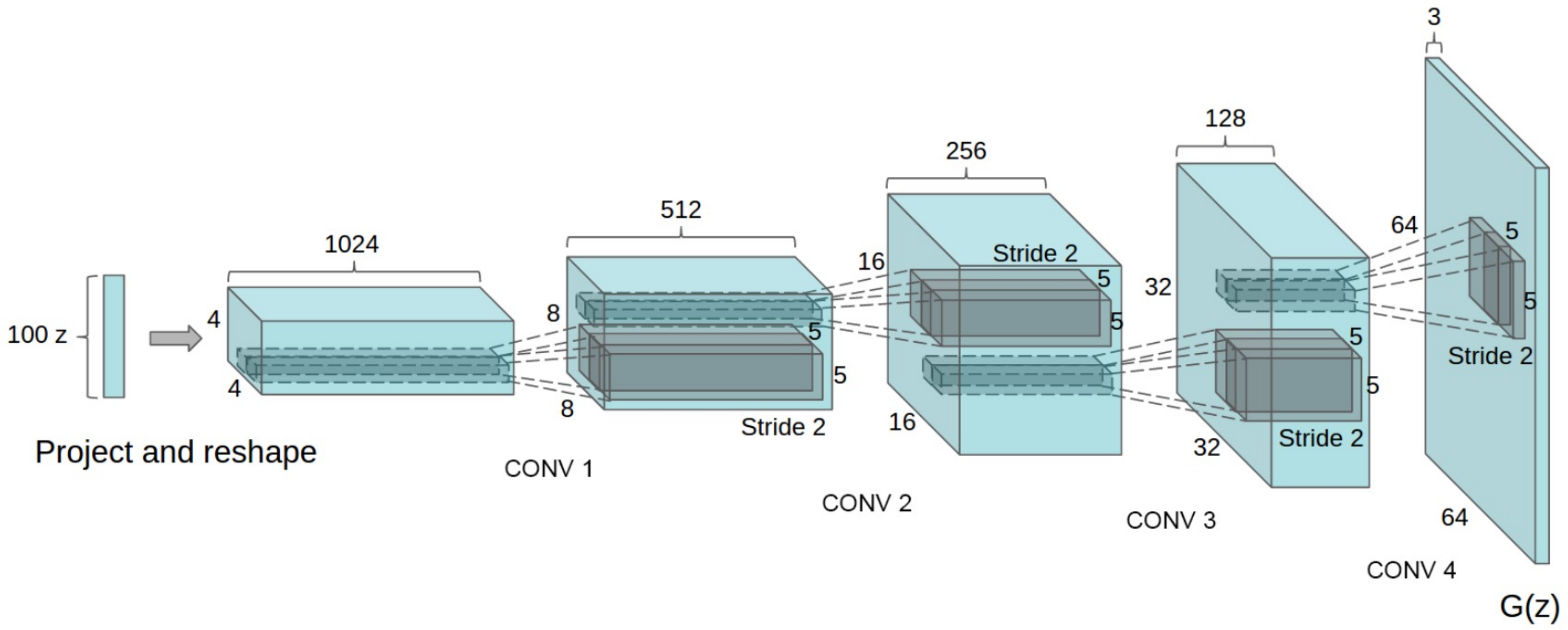
Fig. 5 – This is another example, you can see the same kind of images generated indicating Mode Collapse | Source

Deep Convolutional GANs



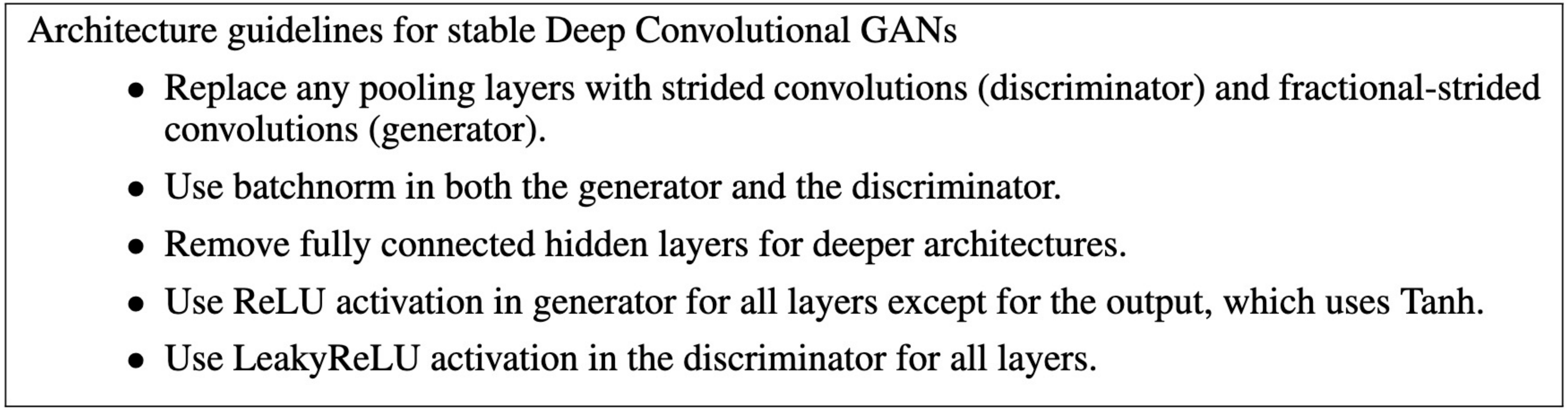
Radford et al. (ICLR 2015)

UpConvolutional Architecture



Radford et al. (ICLR 2015)

Tips & Tricks (DC-GANs)

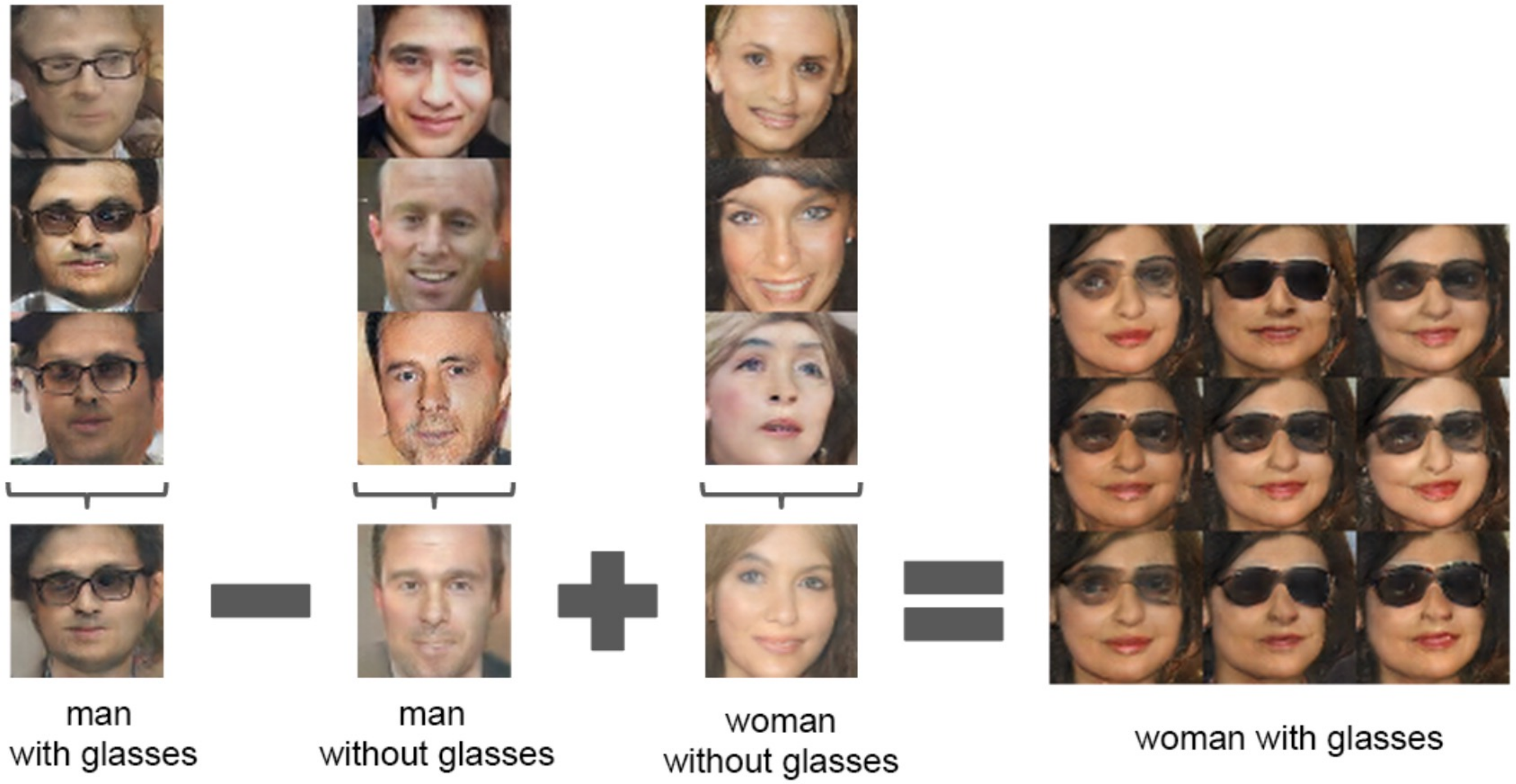


Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

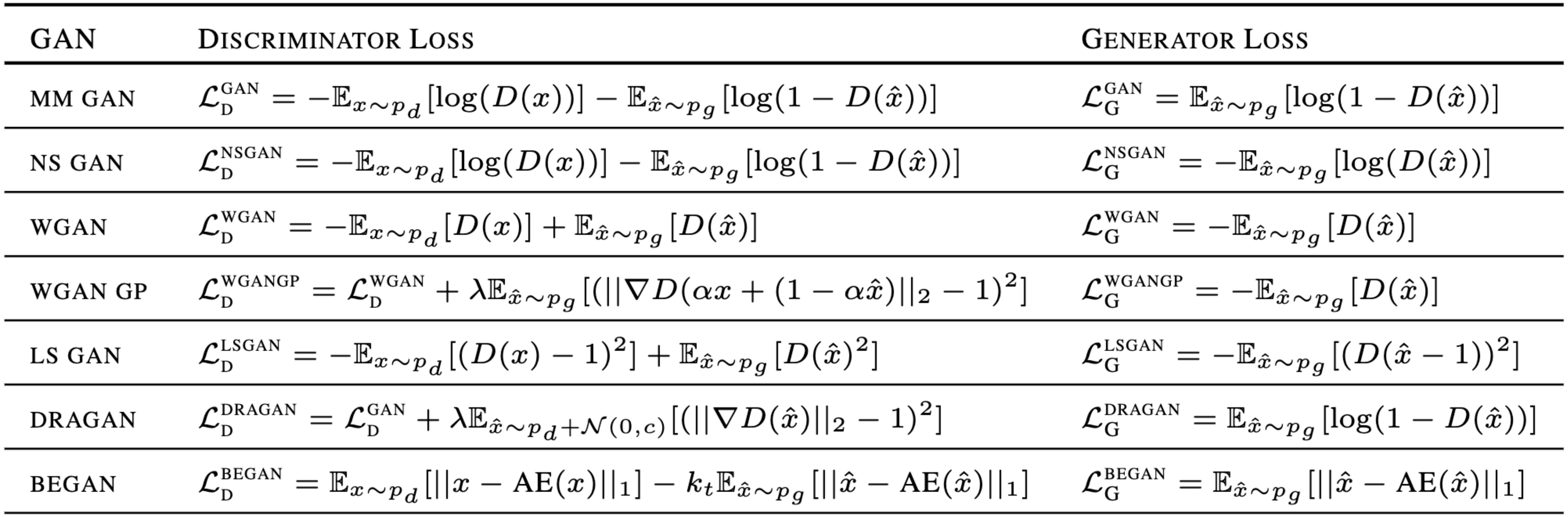
Radford et al. (ICLR 2015)

Latent Vector Arithmetic



Radford et al. (ICLR 2015)

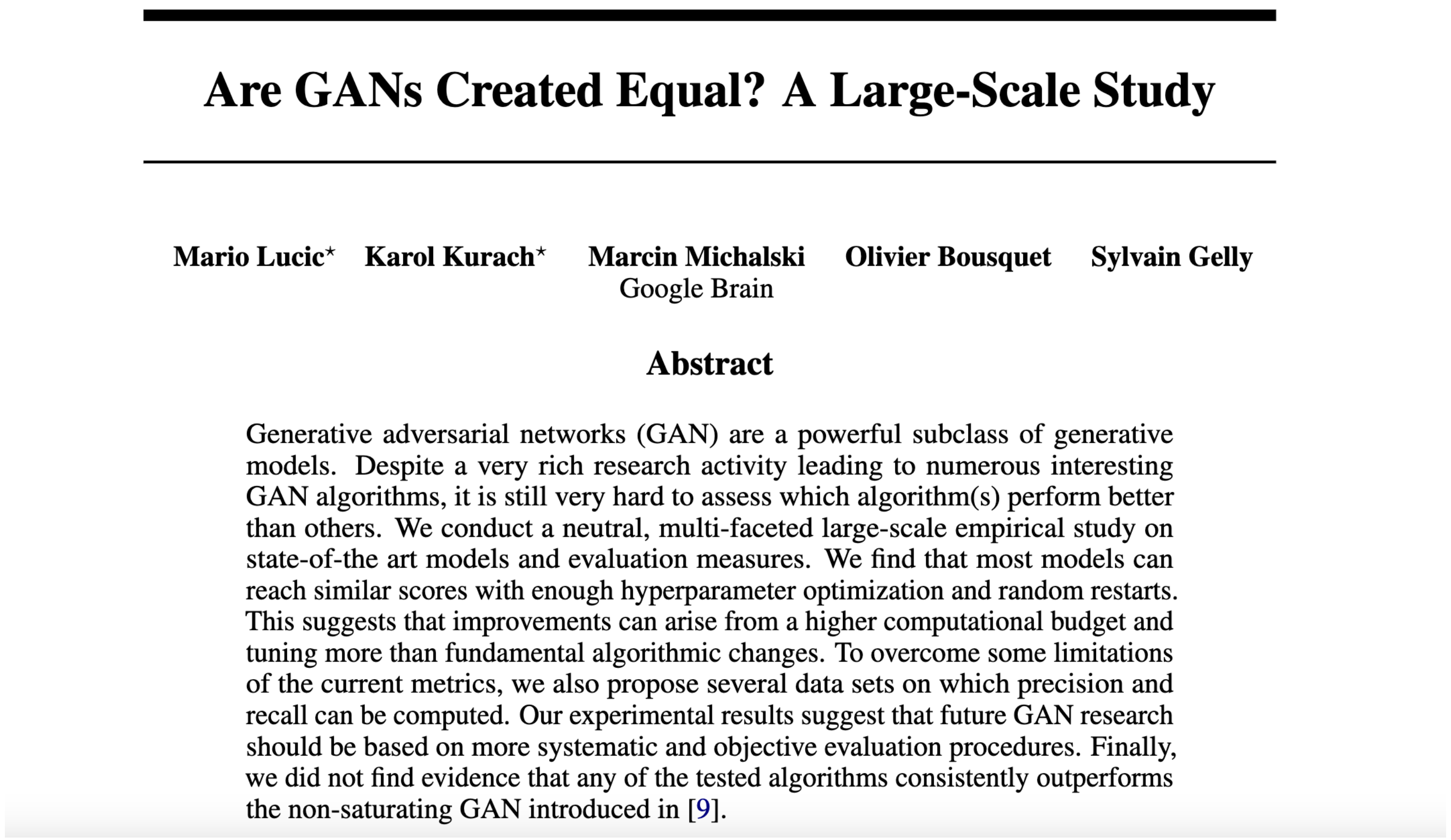
Many Variants of GAN Objectives



GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\text{D}}^{\text{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\text{G}}^{\text{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\text{D}}^{\text{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\text{G}}^{\text{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\text{D}}^{\text{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\text{G}}^{\text{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\text{D}}^{\text{WGANGP}} = \mathcal{L}_{\text{D}}^{\text{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\vert\vert\nabla D(\alpha x + (1 - \alpha\hat{x})\vert\vert_2 - 1)^2]$	$\mathcal{L}_{\text{G}}^{\text{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\text{D}}^{\text{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x) - 1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\text{G}}^{\text{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x} - 1))^2]$
DRAGAN	$\mathcal{L}_{\text{D}}^{\text{DRAGAN}} = \mathcal{L}_{\text{D}}^{\text{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d + \mathcal{N}(0,c)}[(\vert\vert\nabla D(\hat{x})\vert\vert_2 - 1)^2]$	$\mathcal{L}_{\text{G}}^{\text{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\text{D}}^{\text{BEGAN}} = \mathbb{E}_{x\sim p_d}[\vert\vert x - \text{AE}(x)\vert\vert_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\vert\vert\hat{x} - \text{AE}(\hat{x})\vert\vert_1]$	$\mathcal{L}_{\text{G}}^{\text{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\vert\vert\hat{x} - \text{AE}(\hat{x})\vert\vert_1]$

figure credit: "Are GANs Created Equal? A Large-Scale Study" Lucic et al. 2018

Some Confusion About What Really Matters



Are GANs Created Equal? A Large-Scale Study

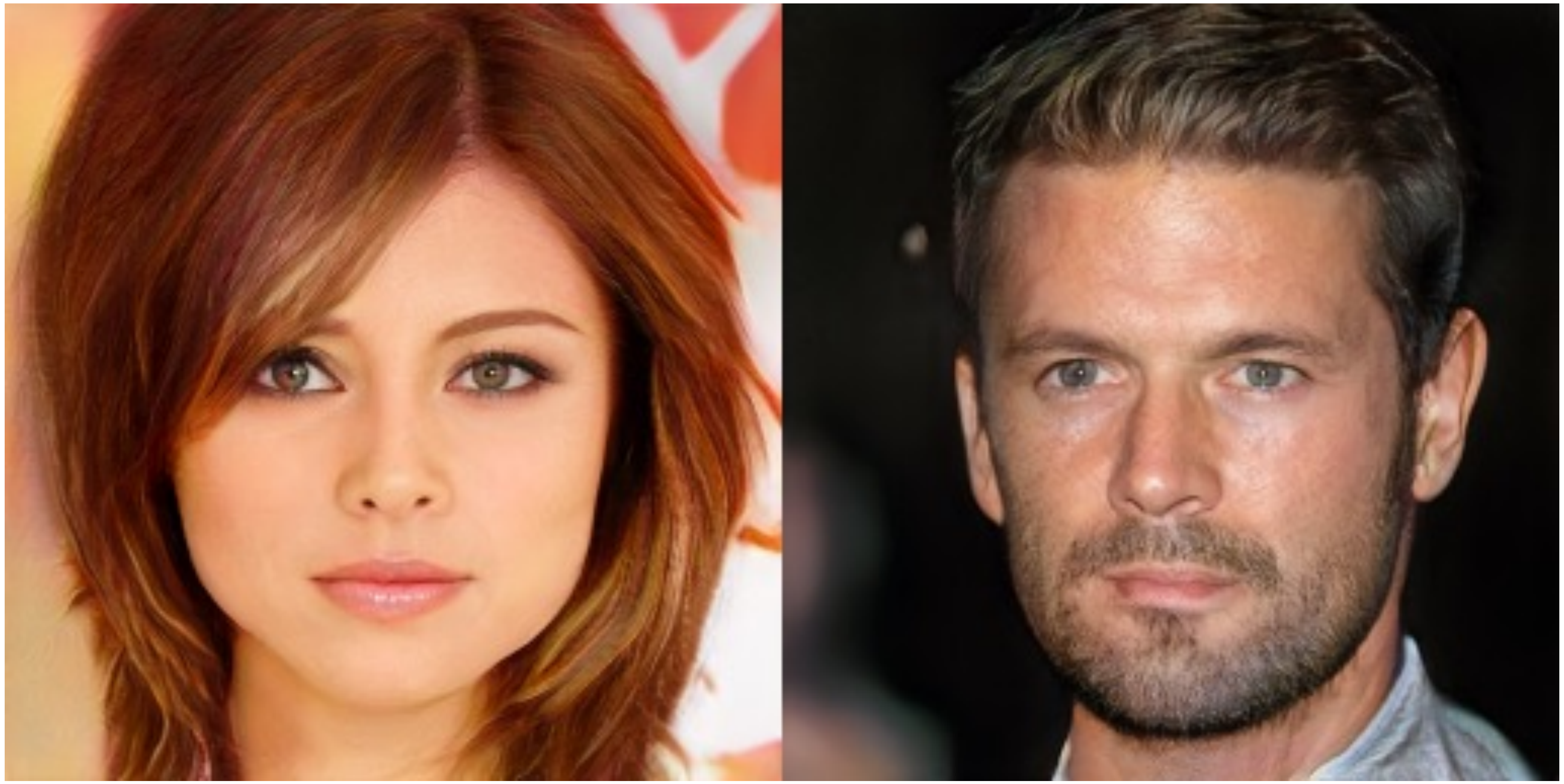
Mario Lucic* **Karol Kurach*** **Marcin Michalski** **Olivier Bousquet** **Sylvain Gelly**
Google Brain

Abstract

Generative adversarial networks (GAN) are a powerful subclass of generative models. Despite a very rich research activity leading to numerous interesting GAN algorithms, it is still very hard to assess which algorithm(s) perform better than others. We conduct a neutral, multi-faceted large-scale empirical study on state-of-the art models and evaluation measures. We find that most models can reach similar scores with enough hyperparameter optimization and random restarts. This suggests that improvements can arise from a higher computational budget and tuning more than fundamental algorithmic changes. To overcome some limitations of the current metrics, we also propose several data sets on which precision and recall can be computed. Our experimental results suggest that future GAN research should be based on more systematic and objective evaluation procedures. Finally, we did not find evidence that any of the tested algorithms consistently outperforms the non-saturating GAN introduced in [9].

“Are GANs Created Equal? A Large-Scale Study” Lucic et al. 2018

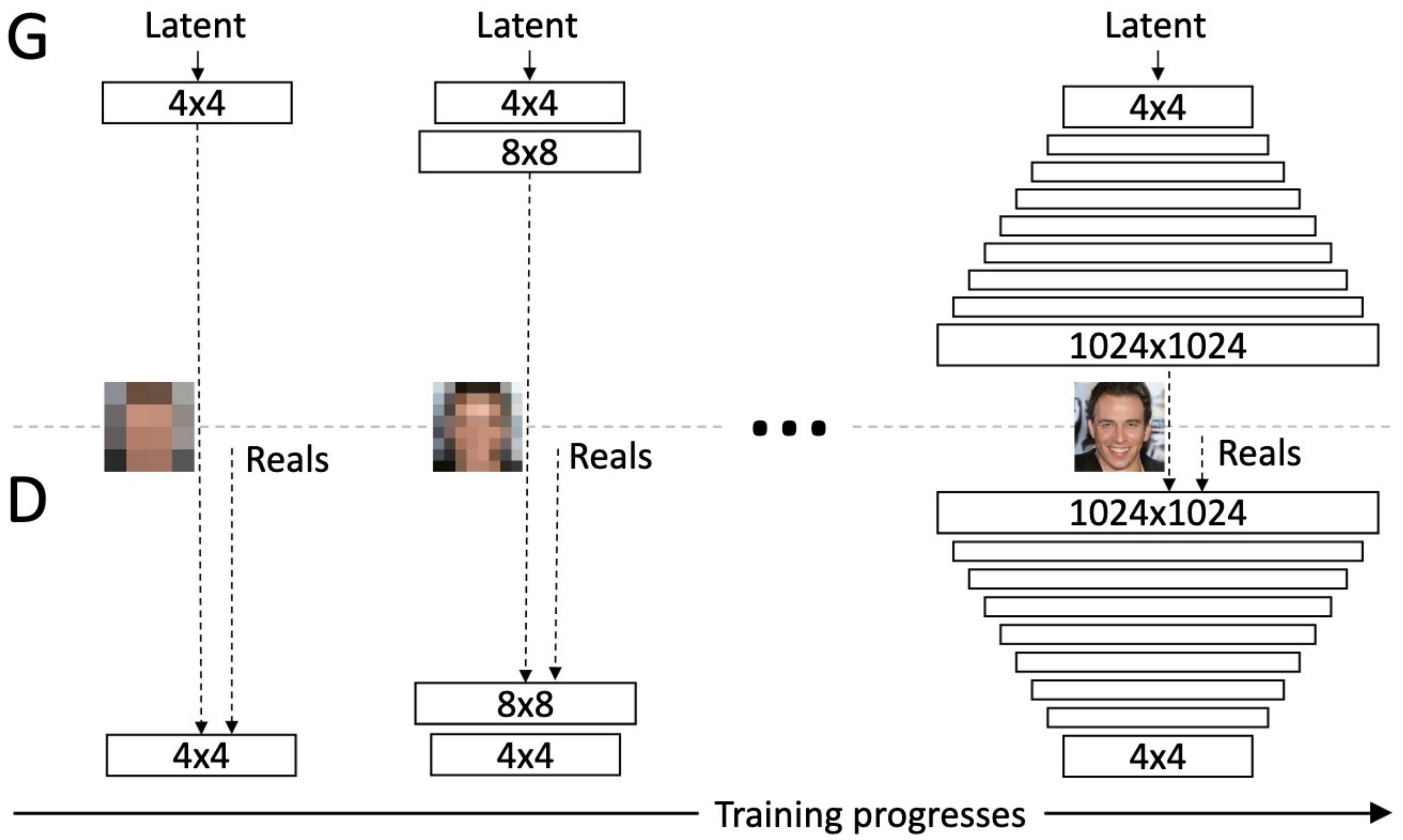
Progressive Growing of GANS



https://www.youtube.com/watch?v=XOxxPcy5Gr4

Progressive Growing (Karras et al., ICLR 2018)

Progressive Growing



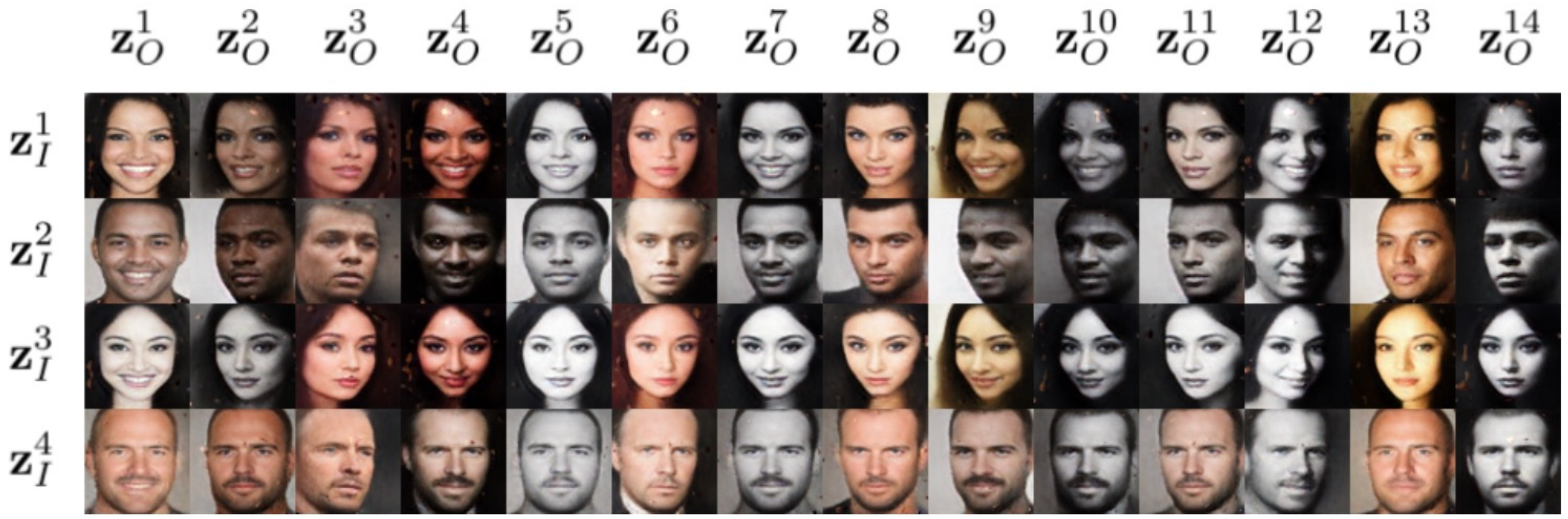
Progressive Growing (Karras et al., ICLR 2018)

Progressive Growing Details

- Start with 4x4 special resolution for both G and D
- As training advances, incrementally add layers, increasing resolution
- All layers remain trainable throughout the process
- Findings
  - More stable synthesis in high resolutions
  - Speeds up training considerably

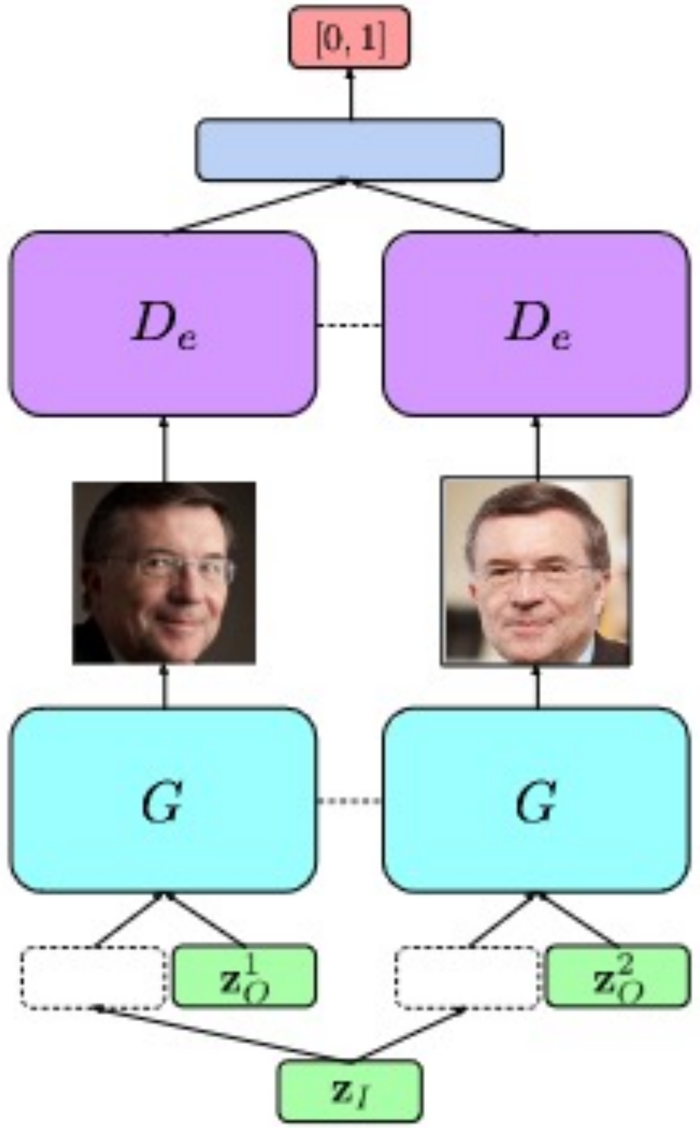
Progressive Growing (Karras et al., ICLR 2018)

Semantically Decomposing Latents (SD–GAN)

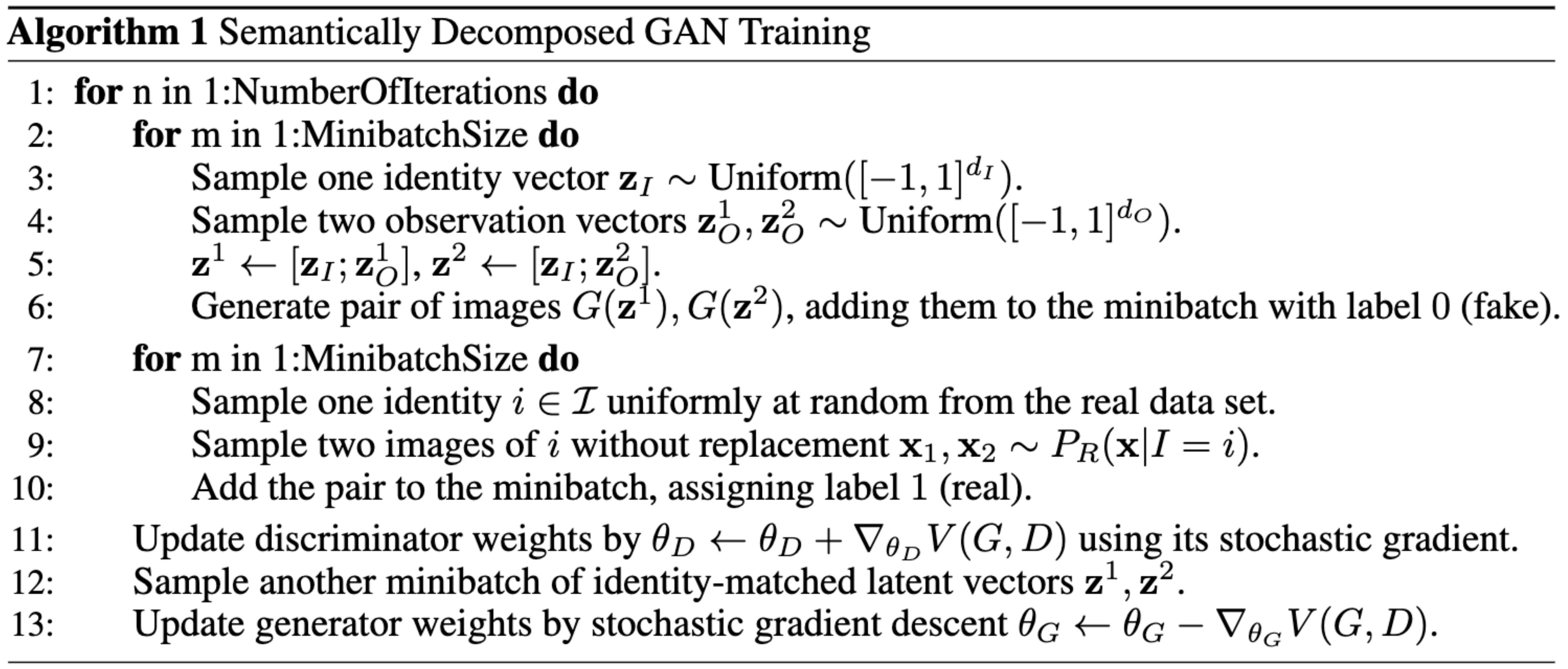


Semantically Decomposing the Latent Spaces of Generative Adversarial Networks
(Donahue, Lipton et al. ICLR 2017)

The Key Idea



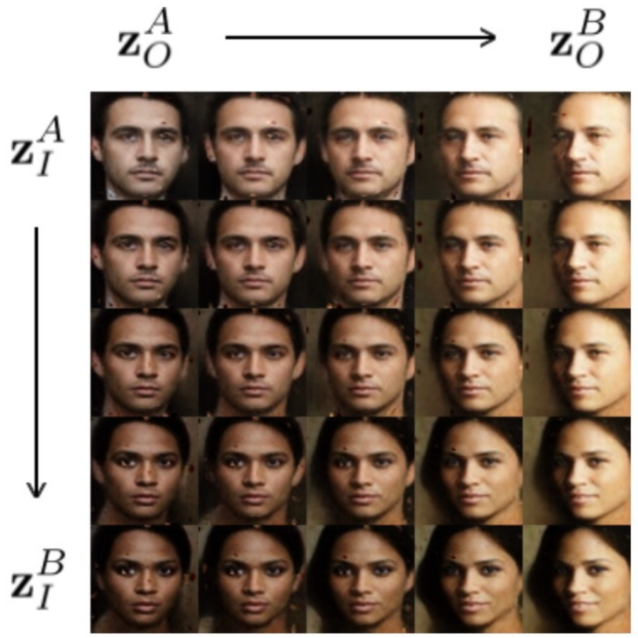
SD-GAN Pseudo-Code



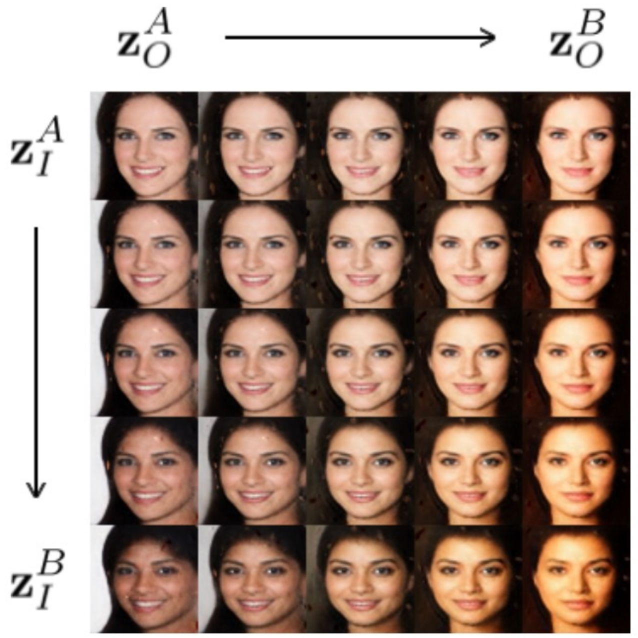
Algorithm 1 Semantically Decomposed GAN Training

```
 1: for n in 1:NumberOfIterations do
 2:     for m in 1:MinibatchSize do
 3:         Sample one identity vector $\mathbf{z}_I \sim \text{Uniform}([-1,1]^{d_I})$.
 4:         Sample two observation vectors $\mathbf{z}_O^1, \mathbf{z}_O^2 \sim \text{Uniform}([-1,1]^{d_O})$.
 5:         $\mathbf{z}^1 \leftarrow [\mathbf{z}_I; \mathbf{z}_O^1]$, $\mathbf{z}^2 \leftarrow [\mathbf{z}_I; \mathbf{z}_O^2]$.
 6:         Generate pair of images $G(\mathbf{z}^1), G(\mathbf{z}^2)$, adding them to the minibatch with label $0$ (fake).
 7:     for m in 1:MinibatchSize do
 8:         Sample one identity $i \in \mathcal{I}$ uniformly at random from the real data set.
 9:         Sample two images of $i$ without replacement $\mathbf{x}_1, \mathbf{x}_2 \sim P_R(\mathbf{x} | I = i)$.
10:         Add the pair to the minibatch, assigning label $1$ (real).
11:     Update discriminator weights by $\theta_D \leftarrow \theta_D + \nabla_{\theta_D} V(G, D)$ using its stochastic gradient.
12:     Sample another minibatch of identity-matched latent vectors $\mathbf{z}^1, \mathbf{z}^2$.
13:     Update generator weights by stochastic gradient descent $\theta_G \leftarrow \theta_G - \nabla_{\theta_G} V(G, D)$.
```

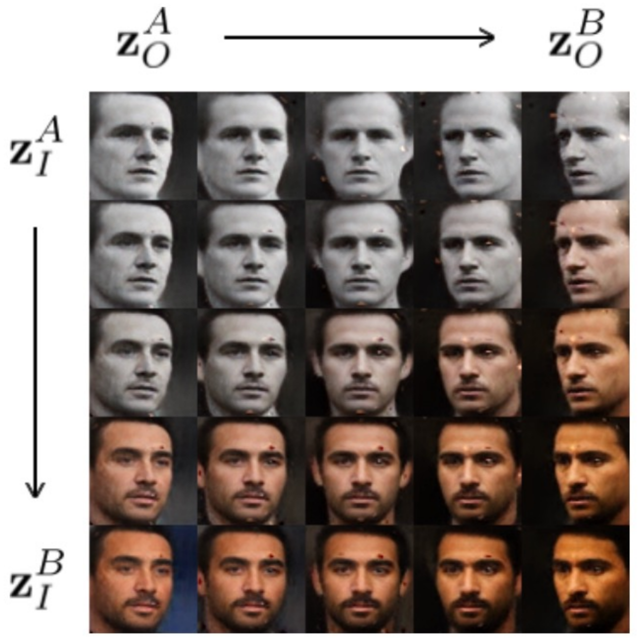
Examples



$\mathbf{z}_O^A$
$\mathbf{z}_O^B$
$\mathbf{z}_I^A$
$\mathbf{z}_I^B$



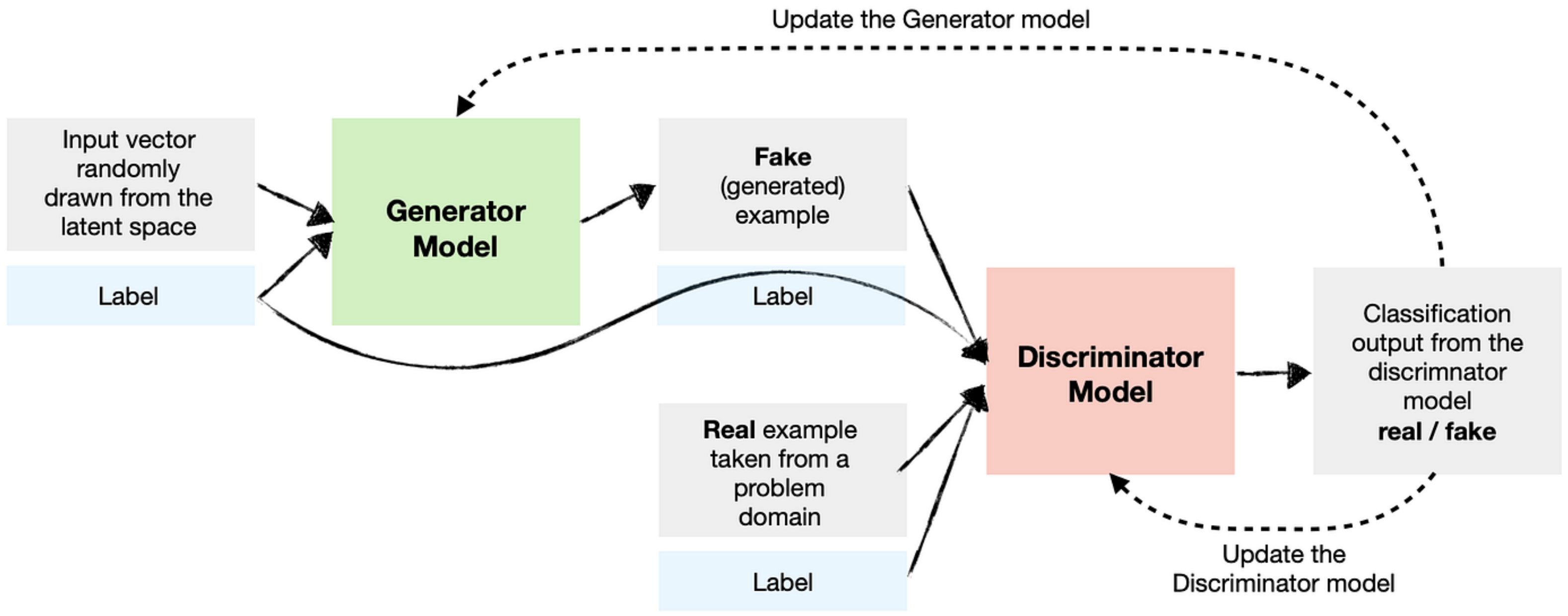
$\mathbf{z}_O^A$
$\mathbf{z}_O^B$
$\mathbf{z}_I^A$
$\mathbf{z}_I^B$



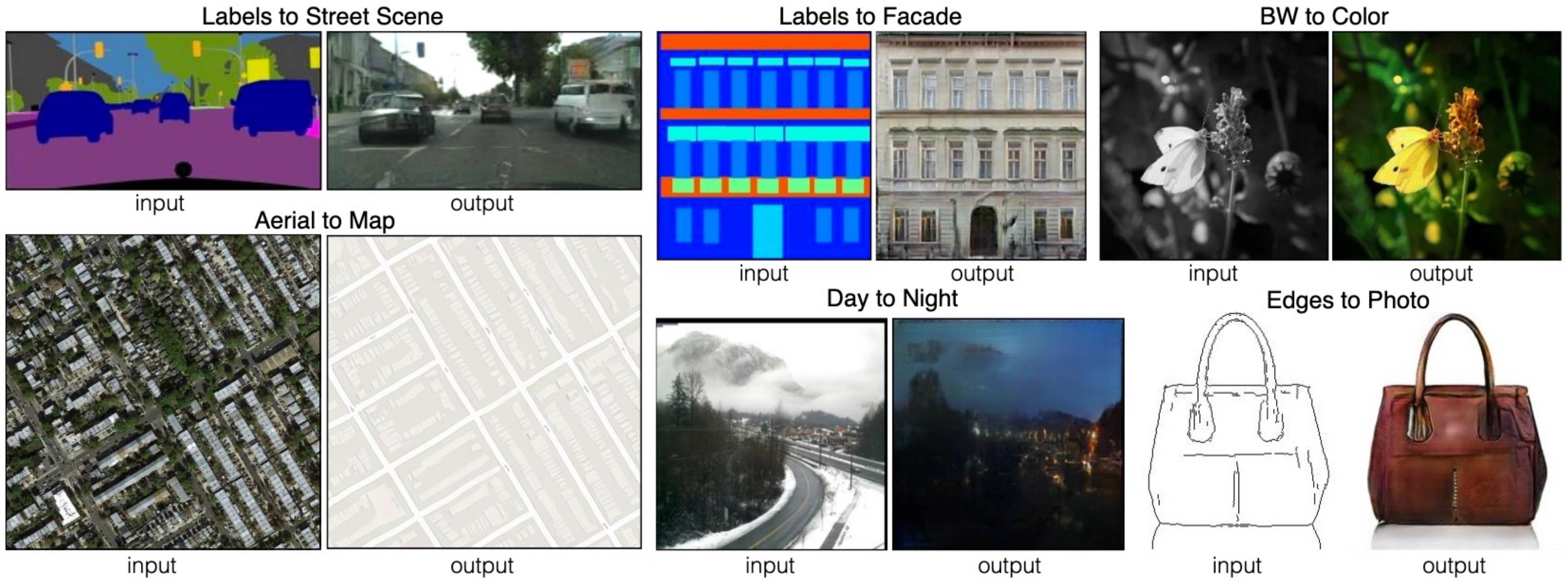
$\mathbf{z}_O^A$
$\mathbf{z}_O^B$
$\mathbf{z}_I^A$
$\mathbf{z}_I^B$

Conditional Generation

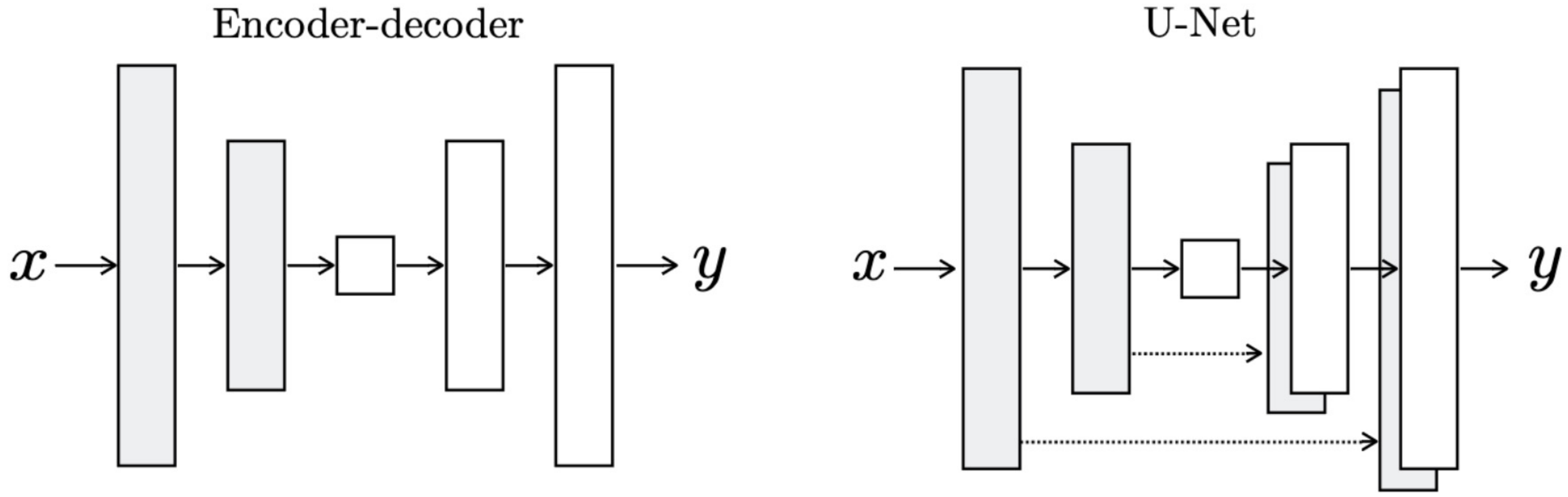
- Input Generator gets context *x, noise z*, produces output *y* = G(*x, z*)
- Discriminator tries to distinguish fake pairs $\{(G(x_i, z), \hat{y}_i)\}$ from real input, target pairs $\{(x_i, y_i)\}$


Update the Generator model
Input vector randomly drawn from the latent space
Label
Generator Model
Fake (generated) example
Label
Discriminator Model
Classification output from the discrimnator model real / fake
Real example taken from a problem domain
Label
Update the Discriminator model

Pix2Pix


Labels to Street Scene
input
output
Aerial to Map
input
output
Labels to Facade
input
output
Day to Night
input
output
BW to Color
input
output
Edges to Photo
input
output

The U-Net Architecture

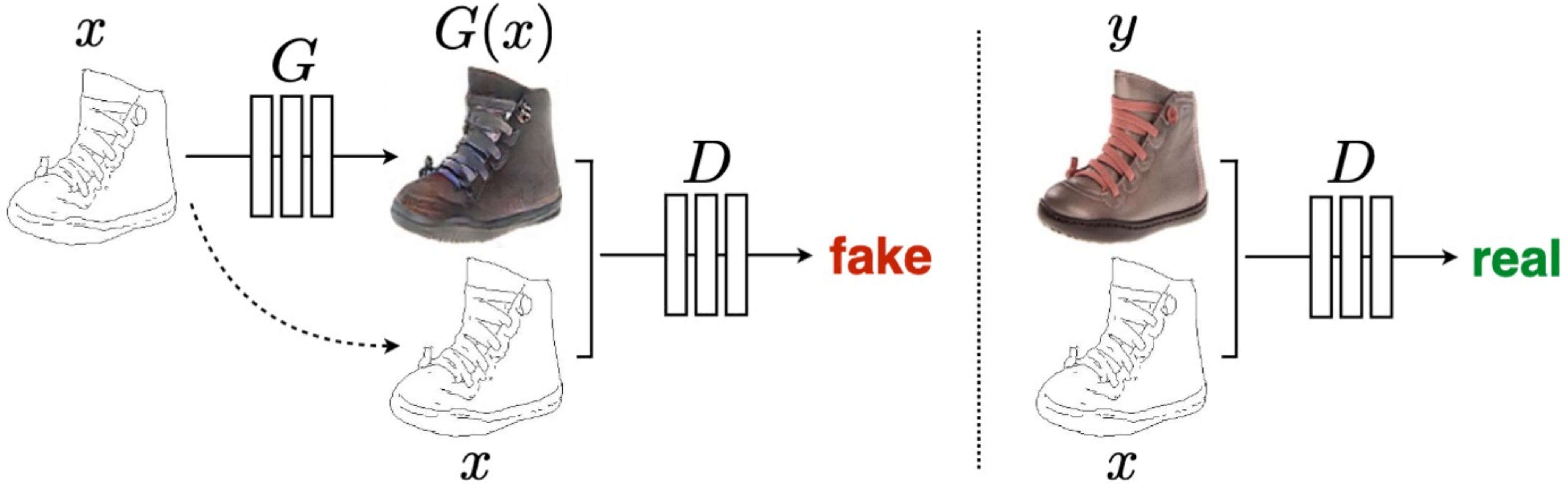


Pix2Pix Learning Setup Adversarial Learning

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G).$$

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))]$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1].$$

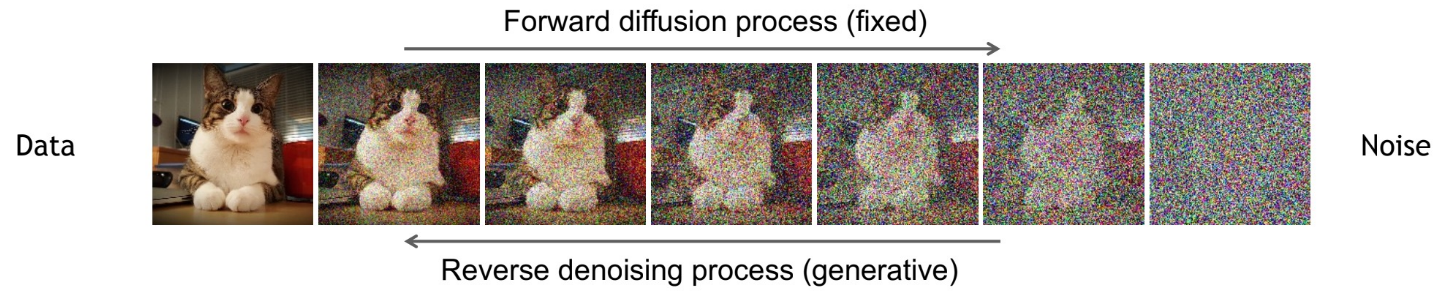


Measuring Sample Quality

- **Inception Score:** feed generated images through Inception v3 model, measure entropy of outputs (lower better), diversity of labels. But what if training data for generative model looks nothing like ImageNet?
- **Frechet Inception Distance (FID):** standard score, measures (mean & variance) of nodes in deepest representation layer of Inception v3
- **Discriminator-based metrics:** train classifier to distinguish real from fake, how easily can it do it (on frozen generator)
- **Human-in-the-loop Evaluation:** human assessments of quality are standard in many papers on image and music generation. Useful for assessing aesthetic properties but may not capture whether the model "learned the distribution"

Current State of the Art: Diffusion Models

- Diffusion models consist of two processes
- Forward process where noise is iteratively added to an input
- Reverse (denoising) process that produces data given noise


Forward diffusion process (fixed)
Data
Noise
Reverse denoising process (generative)

- Training idea: run forward process, learn to predict noise
- Start with random noise, iteratively apply de-noising model

slide credit: Kreis, Gao, Vahdat tutorial