Deep Generative Models

Bùi Tiến Lên

2022

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Contents

Notation

symbol	meaning
$a, b, c, N \ldots$	scalar number
$\boldsymbol{w}, \boldsymbol{v}, \boldsymbol{x}, \boldsymbol{y} \ldots$	column vector
$\boldsymbol{X}, \boldsymbol{Y} \ldots$	matrix
$\mathbb{R}$	set of real numbers
$\mathbb{Z}$	set of integer numbers
$\mathbb{N}$	set of natural numbers
$\mathbb{R}^D$	set of vectors
$\mathcal{X}, \mathcal{Y}, \ldots$	set
$\mathcal{A}$	algorithm

operator	meaning
$\boldsymbol{w}^\mathsf{T}$	transpose
$\boldsymbol{X}\boldsymbol{Y}$	matrix multiplication
$\boldsymbol{X}^{-1}$	inverse
$\boldsymbol{X} \odot \boldsymbol{Y}$	an element-wise matrix-vector multiplication

Generative Models

A real image?

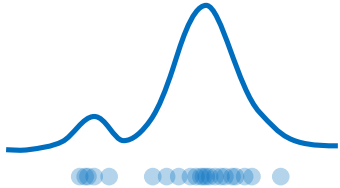
Supervised vs unsupervised learning

	Supervised Learning	Unsupervised Learning
Data	(x, y) x is data, y is label	x x is data, no labels!
Goal	Learn function to map $x \rightarrow y$	Learn some *hidden* or *underlying structure* of the data
Examples	Classification, regression, object detection, semantic segmentation, etc.	Clustering, feature or dimensionality reduction, etc.

Generative modeling

- **Goal**: Take as input training samples from some distribution and learn a model that represents that distribution
- How can we learn $P_{model}(x)$ similar to $P_{data}(x)$?

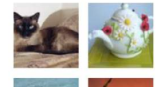
Density Estimation



Sample Generation

Input samples

Training data ~ $P_{data}(x)$



Generated samples

Generated ~ $P_{model}(x)$

Why generative models? Debiasing

- Capable of uncovering **underlying latent variables** in a dataset
- How can we use latent distributions to create fair and representative datasets?

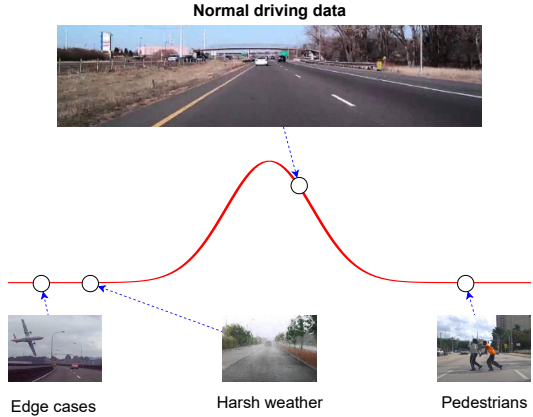
Homogeneous skin color, pose

VS

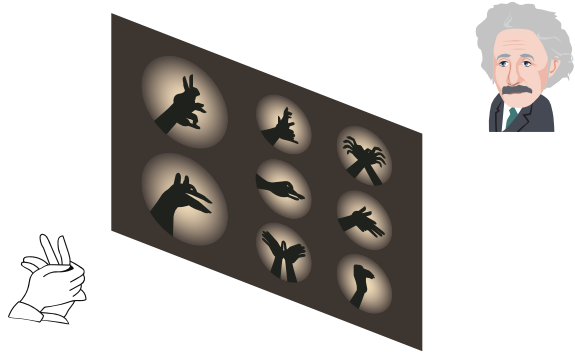
Diverse skin color, pose, illumination

Why generative models? Outlier detection

- **Problem**: How can we detect when we encounter something new or rare?
- **Strategy**: Leverage generative models, detect outliers in the distribution
  - Use outliers during training to improve even more!



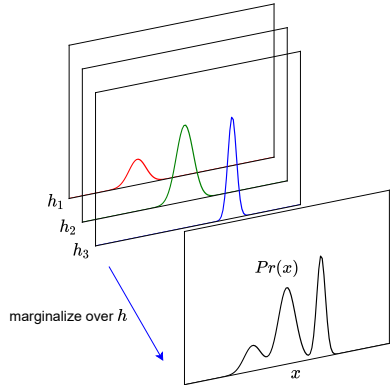
What is a latent variable?



latent variables observed variables

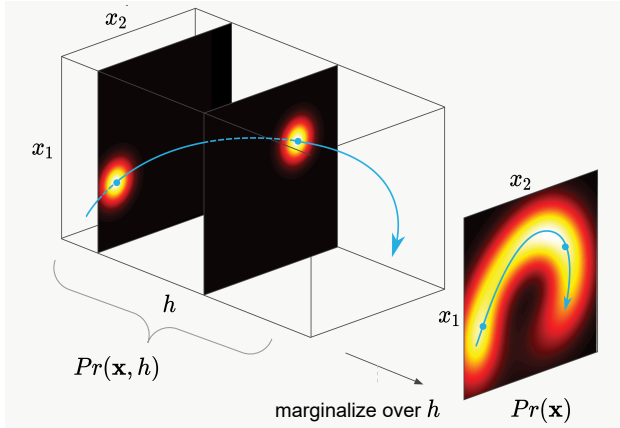
- Can we learn the true **explanatory factors**, e.g. latent variables, from only observed data?
- **Latent variable model**: learn a mapping from some latent variable z (or h) to a complicated distribution on x.

Discrete latent variable



$$Pr(x) = \sum_{i=1}^{k} Pr(x \mid h_i) Pr(h_i) \tag{1}$$

Continuous latent variable



$$Pr(x) = \int Pr(x \mid h) Pr(h) dh \tag{2}$$

Latent model

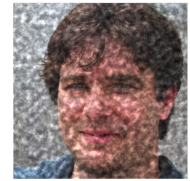
observed variables

latent variables

Encoder

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix}$$

Decoder/Generator



$\mathbf{x}$ $\mathbf{z}$ $\hat{\mathbf{x}}$

Autoencoders

AutoEncoder

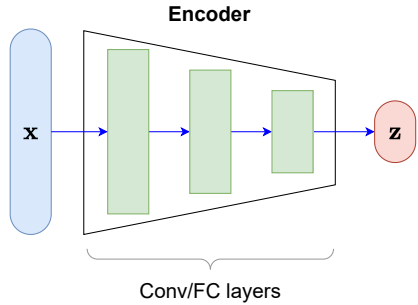
Concept 1

Autoencoder (**auto**matically **encoding** data) is an unsupervised approach for learning a **lower-dimensional feature representation** from unlabeled training data

Encoder

Concept 2

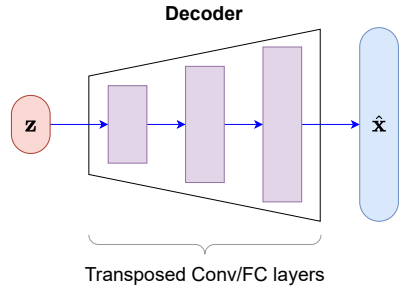
"**Encoder**" learns mapping from the data, $\boldsymbol{x}$, to a low-dimensional latent space, $\boldsymbol{z}$



Decoder

Concept 3

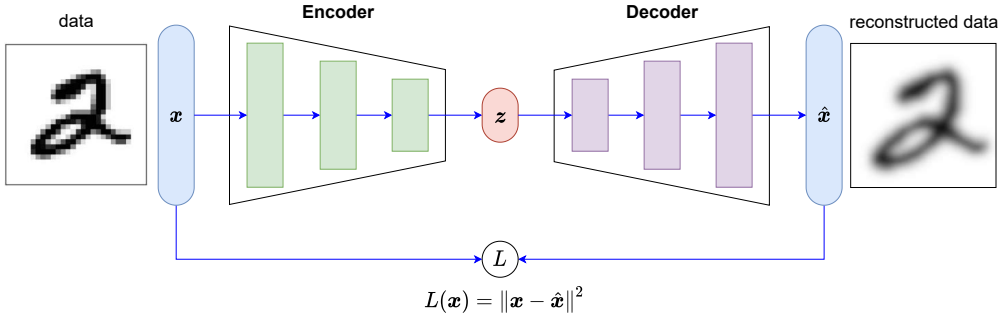
"**Decoder**" learns mapping back from latent, $\boldsymbol{z}$, to a reconstructed observation, $\hat{\boldsymbol{x}}$



Loss function

- How can we **learn** this latent space?
- Train the model to use these features to **reconstruct the original data**
- We define a loss function L (loss function doesn't use any labels!)

$$L(\boldsymbol{x}) = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 \tag{3}$$



Dimensionality of latent space

- Autoencoding is a form of compression!
- Smaller latent space will force a larger training bottleneck

2D latent space

5D latent space

Ground truth

Autoencoders for representation learning

- **Bottleneck hidden layer** forces network to learn a compressed latent representation
- **Reconstruction loss** forces the latent representation to capture (or encode) as much "information" about the data as possible
- **Not probabilistic**: No way to **sample** new data from learned model

Variational Autoencoders (VAEs)

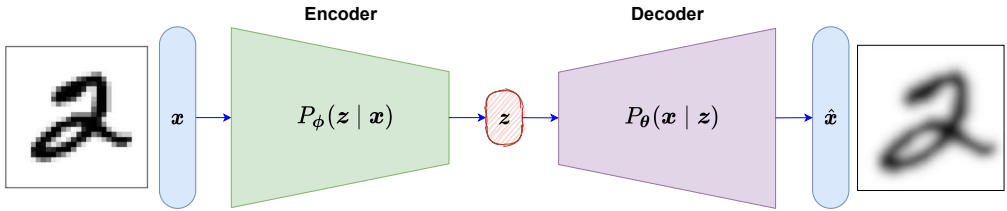
- VAEs
- Latent Space

Variational Autoencoders

Concept 4

Variational autoencoders are a probabilistic twist on autoencoders!

- Sample from $P_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ to compute latent sample
- Sample from $P_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ to generate data sample

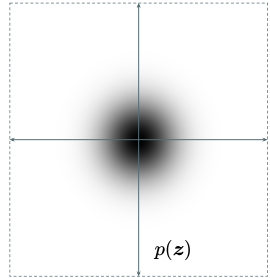


Priors on the latent distribution

- Common choice of prior $p(\boldsymbol{z}), \boldsymbol{z} = (z_1, ..., z_m)$:

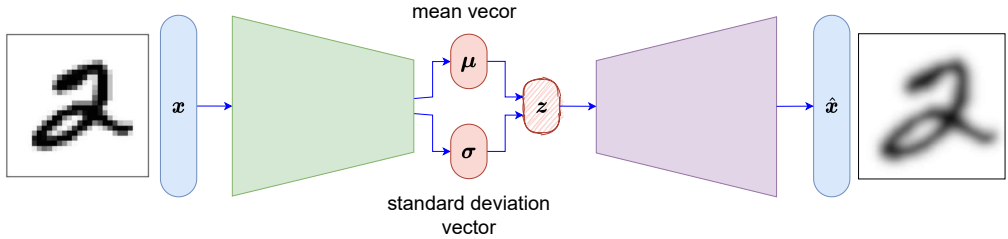
$$\forall z_i,\ p(z_i) = \mathcal{N}(\mu = 0, \sigma^2 = 1) \tag{4}$$

- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to "cheat" by clustering points in specific regions (ie. memorizing the data)



VAE Model

- Variational autoencoders with the prior $p(\boldsymbol{z})$
- The mean vector $\boldsymbol{\mu} = (\mu_1, ..., \mu_m)$
- The standard deviation vector $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_m)$

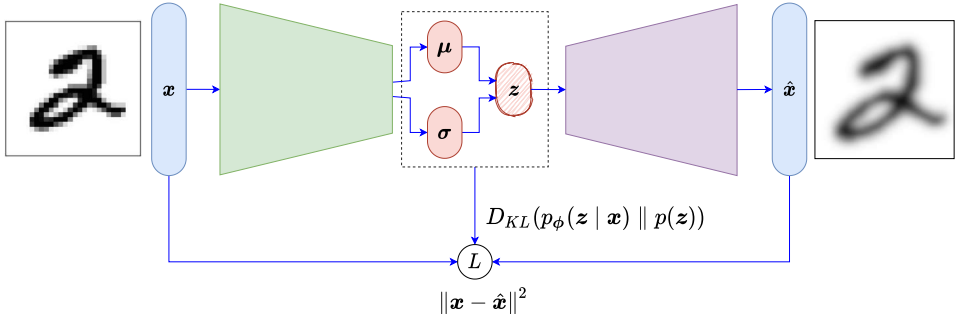


$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$$

Lost function

- We define a lost function L?

$$L_{\phi,\theta}(\boldsymbol{x}) = (\text{reconstruction lost}) + (\text{regularization term}) \tag{5}$$



Lost function (cont.)

Concept 5

Kullback–Leibler divergence, D_{KL} (also called **relative entropy**), is a measure of how one probability distribution is different from a second, reference probability distribution.

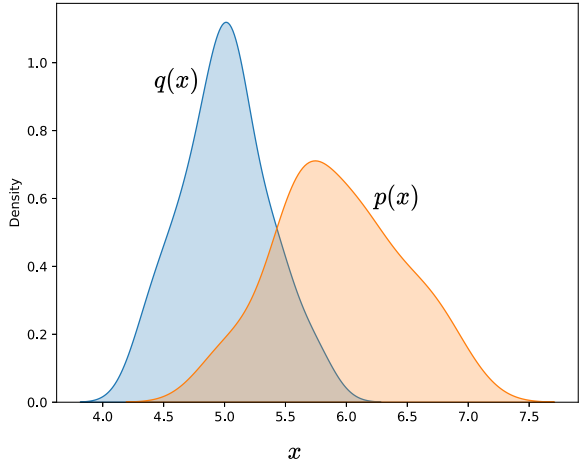
- For discrete probability distributions P and Q defined on the same *probability space* $\mathcal{X}$

$$D_{KL}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{6}$$

- For distributions P and Q of a continuous random variable

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx \tag{7}$$

Lost function (cont.)



Lost function (cont.)

- We have

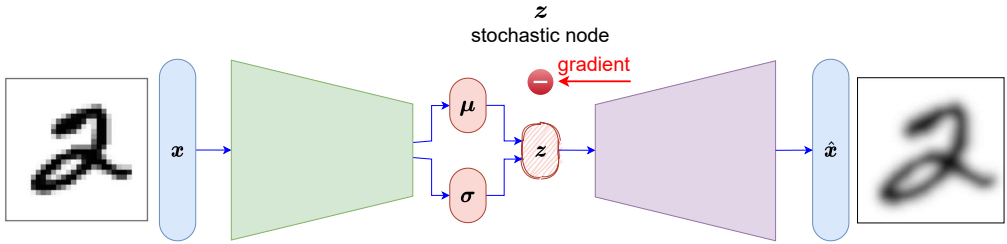
$$\text{reconstruction lost} = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 \tag{8}$$

and

$$\begin{aligned}\text{regularization term} &= D_{KL}(p_{\phi}(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z})) && (9)\\ &= D_{KL}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}) \parallel \mathcal{N}(\mathbf{0}, \mathbf{1})) && (10)\\ &= -\frac{1}{2}\sum_{i=1}^{m}(\sigma_i + \mu_i^2 - 1 - \log \sigma_i) && (11)\end{aligned}$$

VAE Optimization

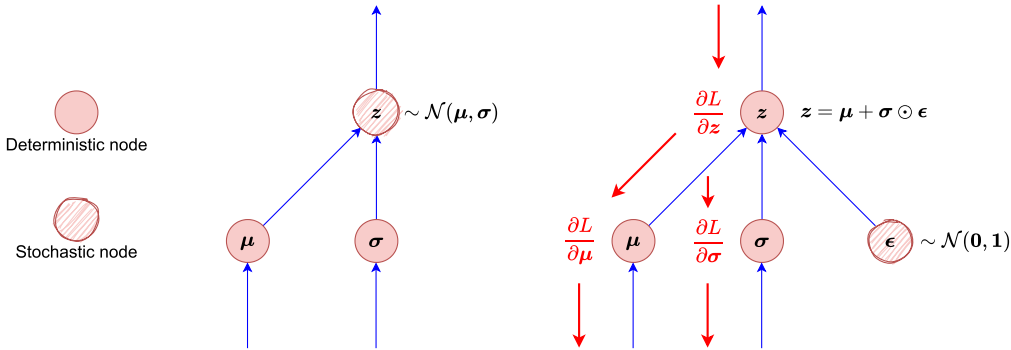
- **Problem**: The network involves a sampling step. We cannot backpropagate gradients through sampling layers!



The reparameterization trick

- **Solution**: we can move the stochastic part into a branch of the network which draws a sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ and then use the following formula

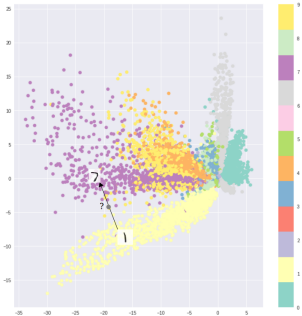
$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{12}$$



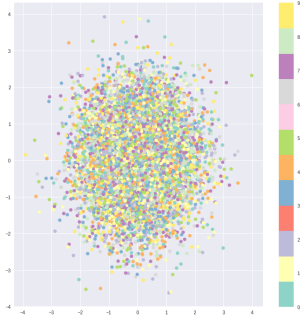
Latent space

- Latent space of dataset MNIST

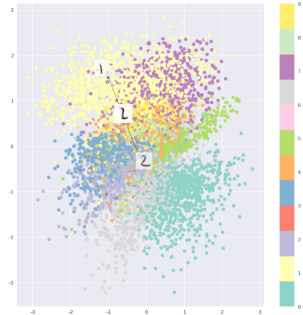
Only reconstruction lost



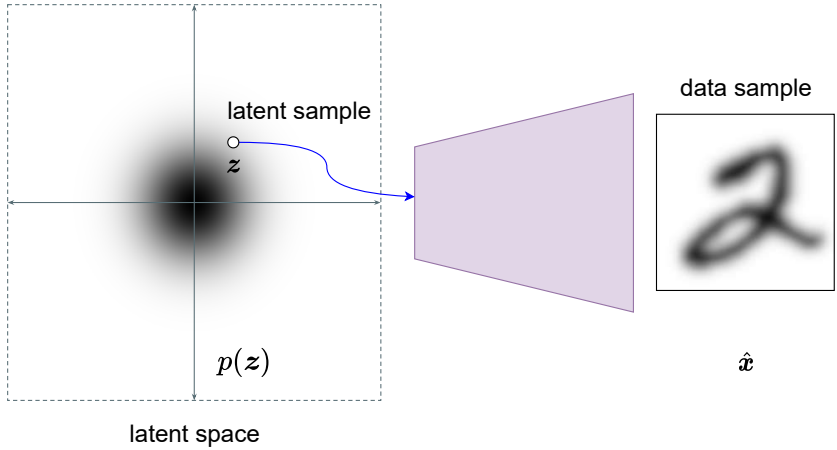
Only KL divergence



Combination



Sampling from latent space

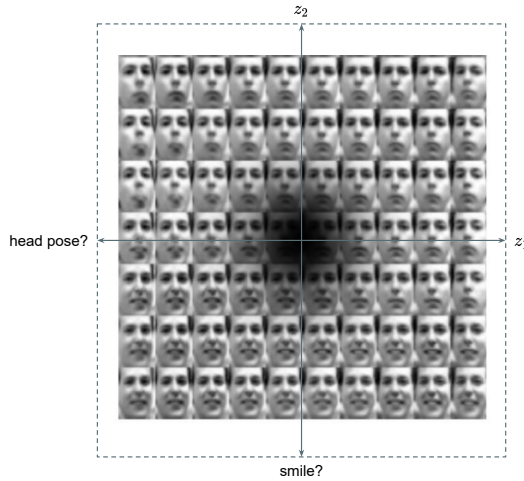


Latent perturbation

- Slowly **increase** or **decrease** a single latent variable and **keep** all other variables fixed
- Different dimensions of $\boldsymbol{z}$ encodes **different interpretable latent features**
  - Ideally, we want latent variables that are uncorrelated with each other
  - Enforce diagonal prior on the latent variables to encourage independence

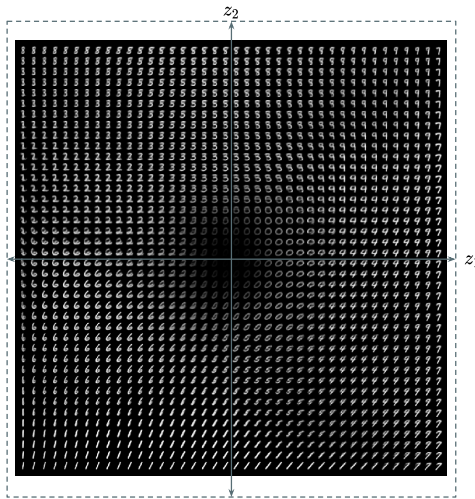
Latent perturbation (cont.)

- Olivetti faces dataset



Latent perturbation (cont.)

- MNIST dataset



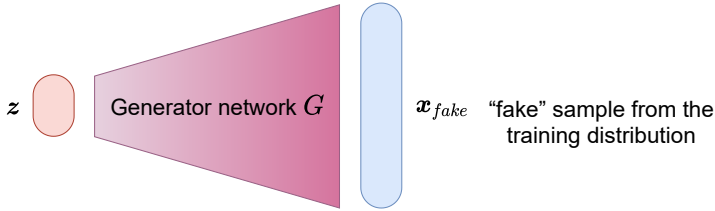
VAE summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end
4. Interpret hidden latent variables using perturbation
5. Generating new examples

Generative Adversarial Networks (GANs)

What if we just want to sample?

- **Idea**: don't explicitly model density, and instead just sample to generate new instances.
- **Problem**: want to sample from complex distribution – can't do this directly!
- **Solution**: sample from something simple (noise), learn a transformation to the training distribution.



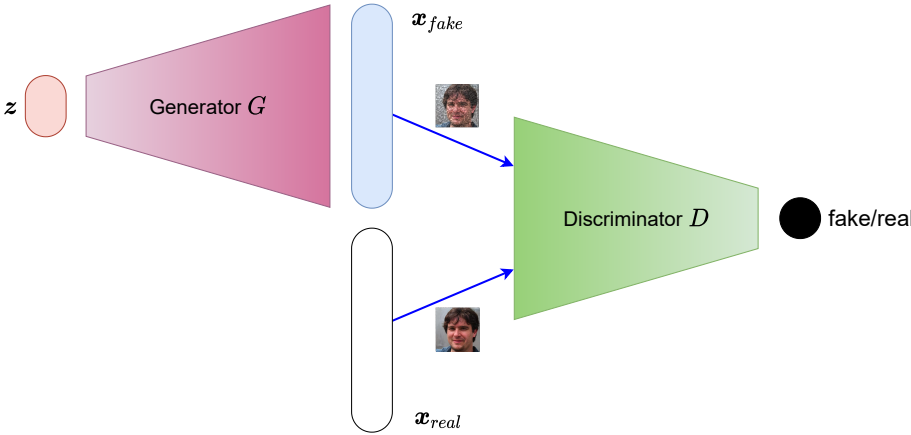
Generative Adversarial Networks (GANs)

Concept 6

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

- The **generator** D turns noise into an imitation of the data to try to trick the discriminator.
- The **discriminator** G tries to identify real data from fakes created by the generator.

Generative Adversarial Networks (GANs) (cont.)



GAN objective

- Expected conditional log likelihood for real and generated data

$$V(G,D) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p(\text{data})} \log D_{\boldsymbol{\theta}_d}(\boldsymbol{x})}_{\text{real data}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})} \log(1 - D_{\boldsymbol{\theta}_d}(G_{\boldsymbol{\theta}_g}(\boldsymbol{z})))}_{\text{generated data}} \tag{13}$$

- The discriminator wants to correctly distinguish real and fake samples

$$D^* = \arg\max_D V(G,D) \tag{14}$$

- The generator wants to fool the discriminator

$$G^* = \arg\min_G V(G,D) \tag{15}$$

Training GANs

- **Discriminator** tries to identify real data from fakes created by the generator.
- **Generator** tries to create imitations of data to trick the discriminator.

Train GAN jointly via **minimax** game

$$\arg\min_{\boldsymbol{\theta}_g}\max_{\boldsymbol{\theta}_d}\left[\mathbb{E}_{\boldsymbol{x}\sim p(\text{data})}\log D_{\boldsymbol{\theta}_d}(\boldsymbol{x})+\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\log(1-D_{\boldsymbol{\theta}_d}(G_{\boldsymbol{\theta}_g}(\boldsymbol{z})))\right] \tag{16}$$

- **Discriminator** wants to maximize objective s.t. $D(\boldsymbol{x})$ close to $\mathbf{1}$, $D(G(\boldsymbol{z}))$ close to $\mathbf{0}$.
- **Generator** wants to minimize objective s.t. $D(G(\boldsymbol{z}))$ close to $\mathbf{1}$

Algorithm

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:

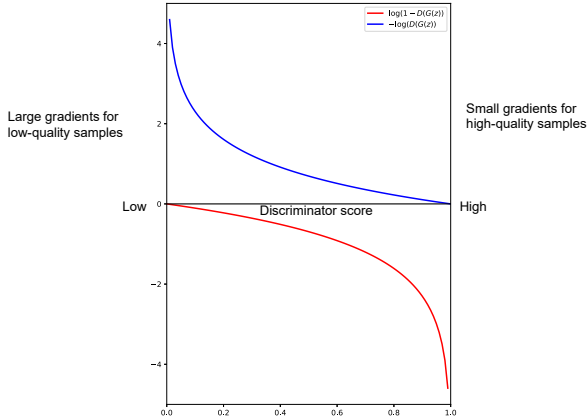
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

Training in practice

- The minimax objective leads to **vanishing gradients** as the discriminator saturates.



Training in practice (cont.)

- In practice, Goodfellow et al (2014) advocate the heuristic training objective:
- **Discriminator**

$$\arg\max_{\boldsymbol{\theta}_d} \left[\mathbb{E}_{\boldsymbol{x}\sim p(\text{data})} \log D_{\boldsymbol{\theta}_d}(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})} \log(1 - D_{\boldsymbol{\theta}_d}(G_{\boldsymbol{\theta}_g}(\boldsymbol{z})))\right] \tag{17}$$

- **Generator**

$$\arg\max_{\boldsymbol{\theta}_g} \left[\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})} \log(D_{\boldsymbol{\theta}_d}(G_{\boldsymbol{\theta}_g}(\boldsymbol{z})))\right] \tag{18}$$

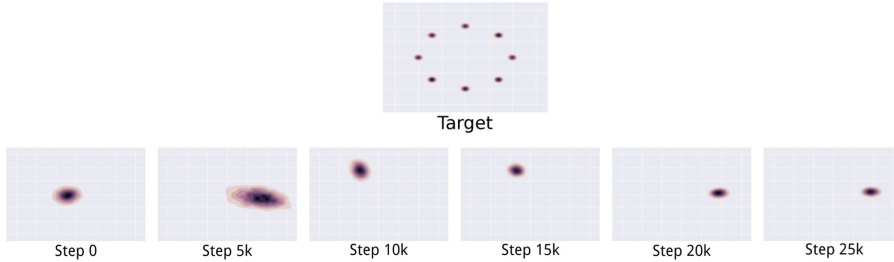
Other problems with GAN training

Non-convergence

- Parameters can oscillate or diverge, generator loss does not correlate with sample quality
- Behavior very sensitive to hyperparameter selection

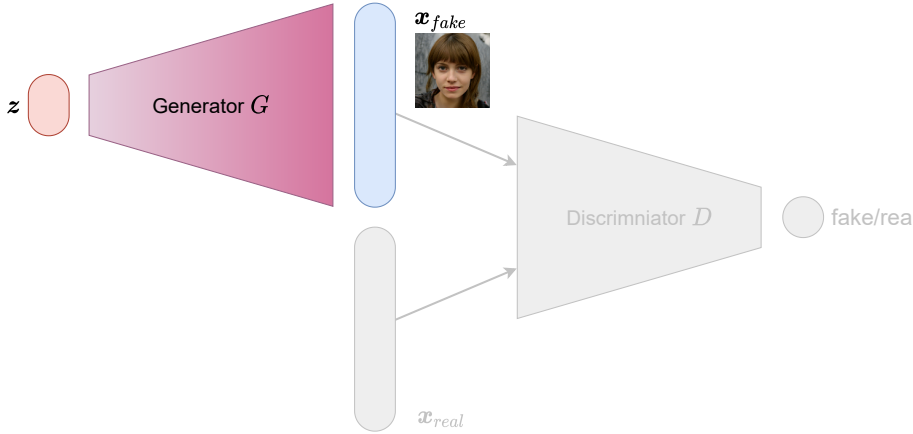
Mode collapse

- Generator ends up modeling only a small subset of the training data



Generating new data with GANs

- After training, use generator network to create new data that's never been seen before.

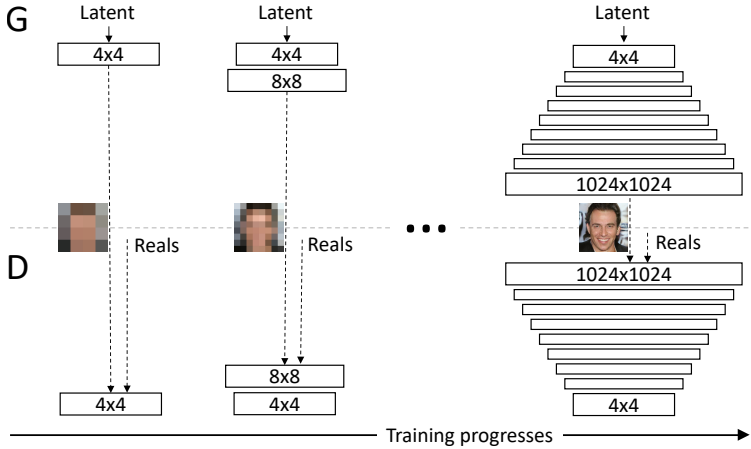


GAN Evaluation

GAN Recent Advances

Progressive GANs (NVIDIA)

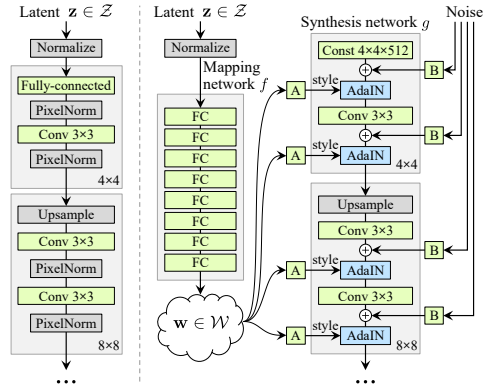
- **Key idea**: train lower-resolution models, gradually add layers corresponding to higher-resolution outputs (Karras et al., ICLR 2018.)



Progressive GANs: results

StyleGAN

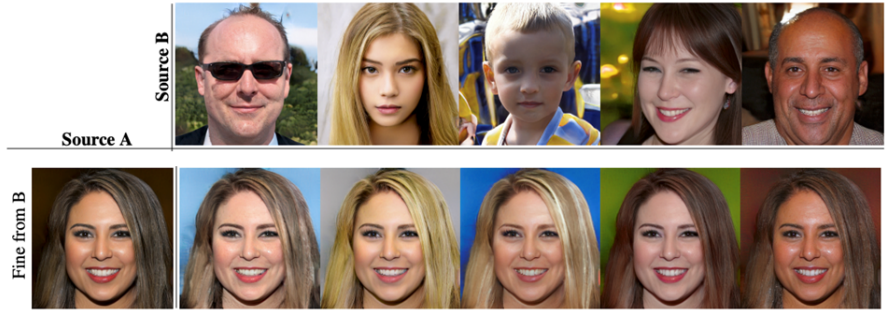
- Built on top of Progressive GAN
- Start with learned constant (instead of noise vector)
- Use a mapping network to produce a style code $\boldsymbol{w}$ using learned affine transformations A
- Use adaptive instance normalization (`AdaIN`): scale and bias each feature map using learned style values
- Add noise after each convolution and before nonlinearity (enables stochastic detail)



StyleGAN: results

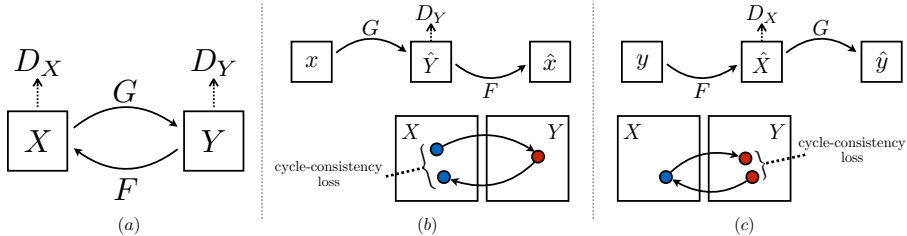
Mixing styles

- Two sets of images were generated from their respective latent codes (sources A and B); the rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A.

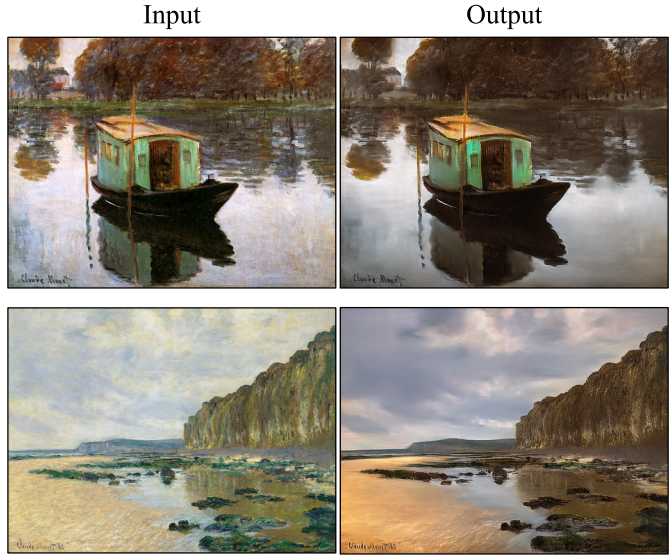


CycleGAN

- CycleGAN learns transformations across domains with unpaired data.



CycleGAN: results



References

Goodfellow, I., Bengio, Y., and Courville, A. (2016).
Deep learning.
MIT press.

Lê, B. and Tô, V. (2014).
Cở sở trí tuệ nhân tạo.
Nhà xuất bản Khoa học và Kỹ thuật.

Russell, S. and Norvig, P. (2021).
Artificial intelligence: a modern approach.
Pearson Education Limited.