Chapter 5: Unsupervised Learning

Mathematical Foundations of Deep Neural Networks

Fall 2022

Department of Mathematical Sciences

Ernest K. Ryu

Seoul National University

Unsupervised learning

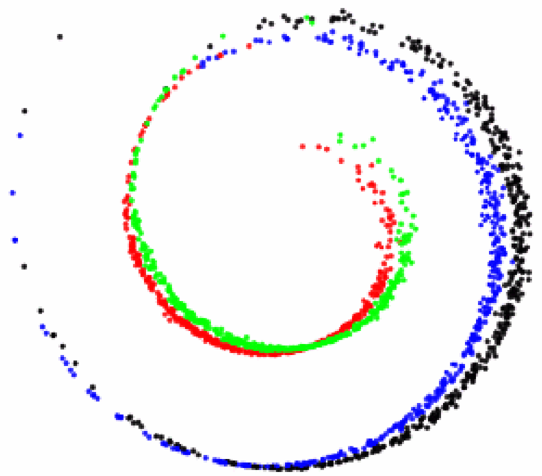
Unsupervised learning utilizes data $X_1, \dots, X_N$ to learn the “structure” of the data. No labels are utilized.

There are a wide range of unsupervised learning tasks. In this class, we discuss just a few.

Generally, unsupervised learning tasks tend to have more mathematical complexity.

Low-dimensional latent representation

Many high-dimensional data has some underlying low-dimensional structure.*



If you randomly generate the pixels of a color image $X \in \mathbb{R}^{3\times m\times n}$, it will likely make no sense. Only a very small subset of pixel values correspond to meaningful images.

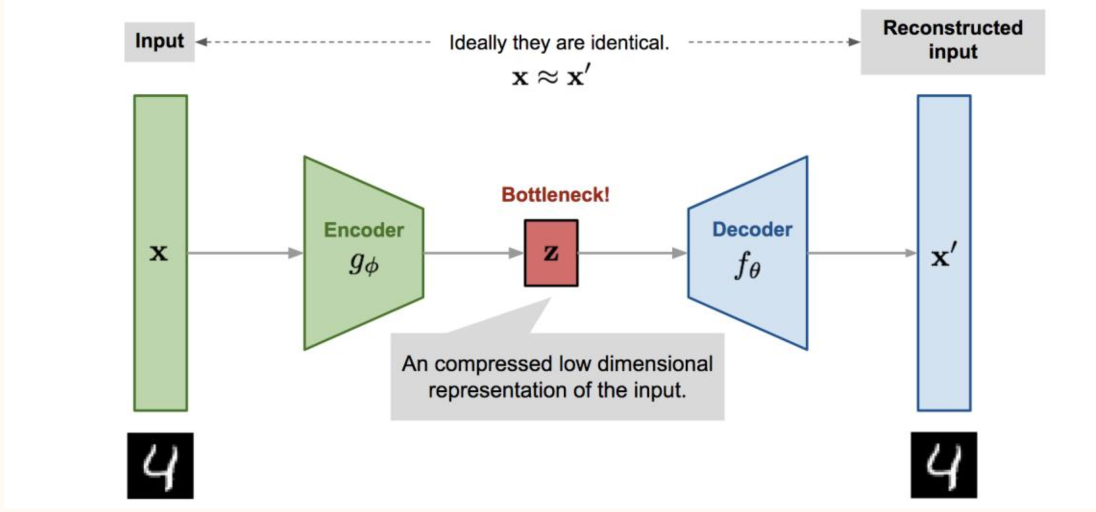
*One can model this assumption as data residing in a low dimensional manifold and utilize ideas from differential geometry. We won't pursue this direction.

Finding latent representations

In machine learning, especially in unsupervised learning, finding a “meaningful” low-dimensional latent representation is of interest.

A good lower-dimensional representation of the data implies you have a good understanding of the data.

Autoencoder


Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

An *autoencoder* (AE) has *encoder* $E_\theta : \mathbb{R}^n \to \mathbb{R}^r$ and *decoder* $D_\varphi : \mathbb{R}^r \to \mathbb{R}^n$ networks, where $r \ll n$. (If $r \geq n$, AE learns identity mapping, so pointless.) The two networks are trained through the loss

$$\mathcal{L}(\theta, \varphi) = \sum_{i=1}^{N} \left\| X_i - D_\varphi\big(E_\theta(X_i)\big) \right\|^2$$

The low-dimensional output $E_\theta(X)$ is the *latent vector*. The encoder performs *dimensionality reduction*.

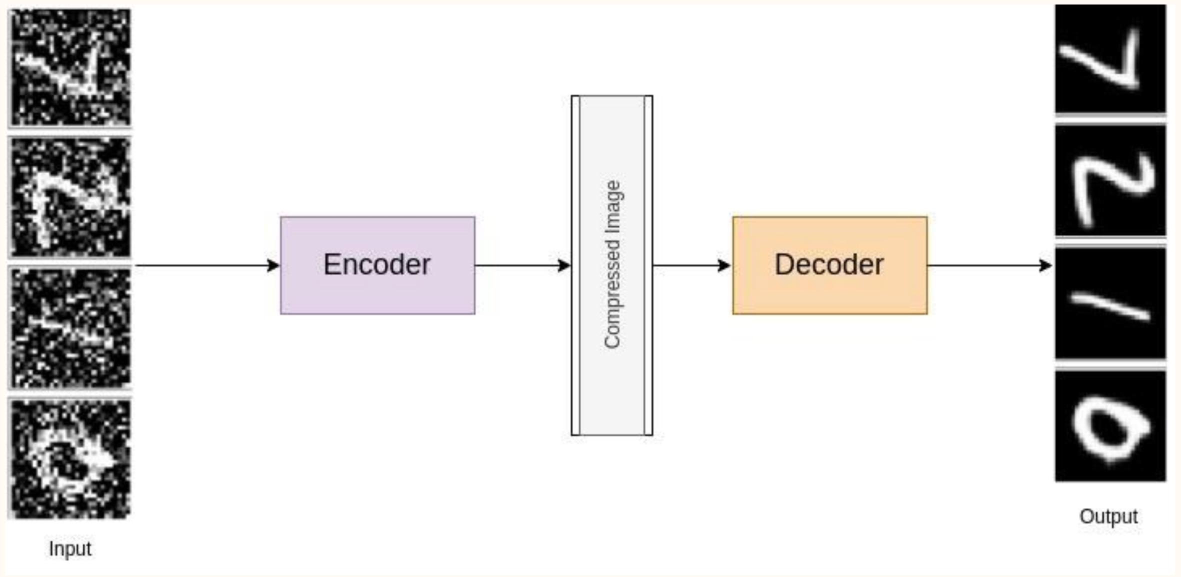
The autoencoder can be thought of as a deep non-linear generalization of the principle component analysis (PCA).

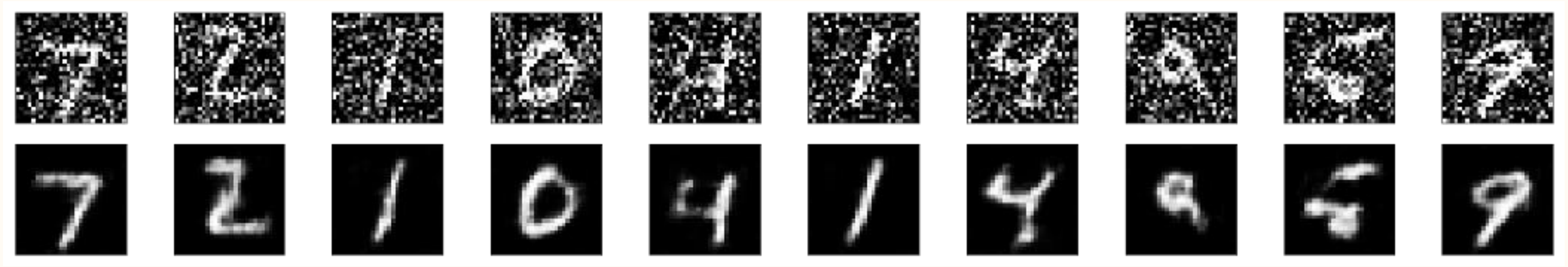
Autoencoder with MNIST

PyTorch demo

Applications of AE: Denoising

Autoencoders can be used to denoise or reconstruct corrupted images.





P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, *JMLR*, 2010.
G. Nishad, Reconstruct corrupted data using Denoising Autoencoder, *Medium*, 2020.

Applications of AE: Compression

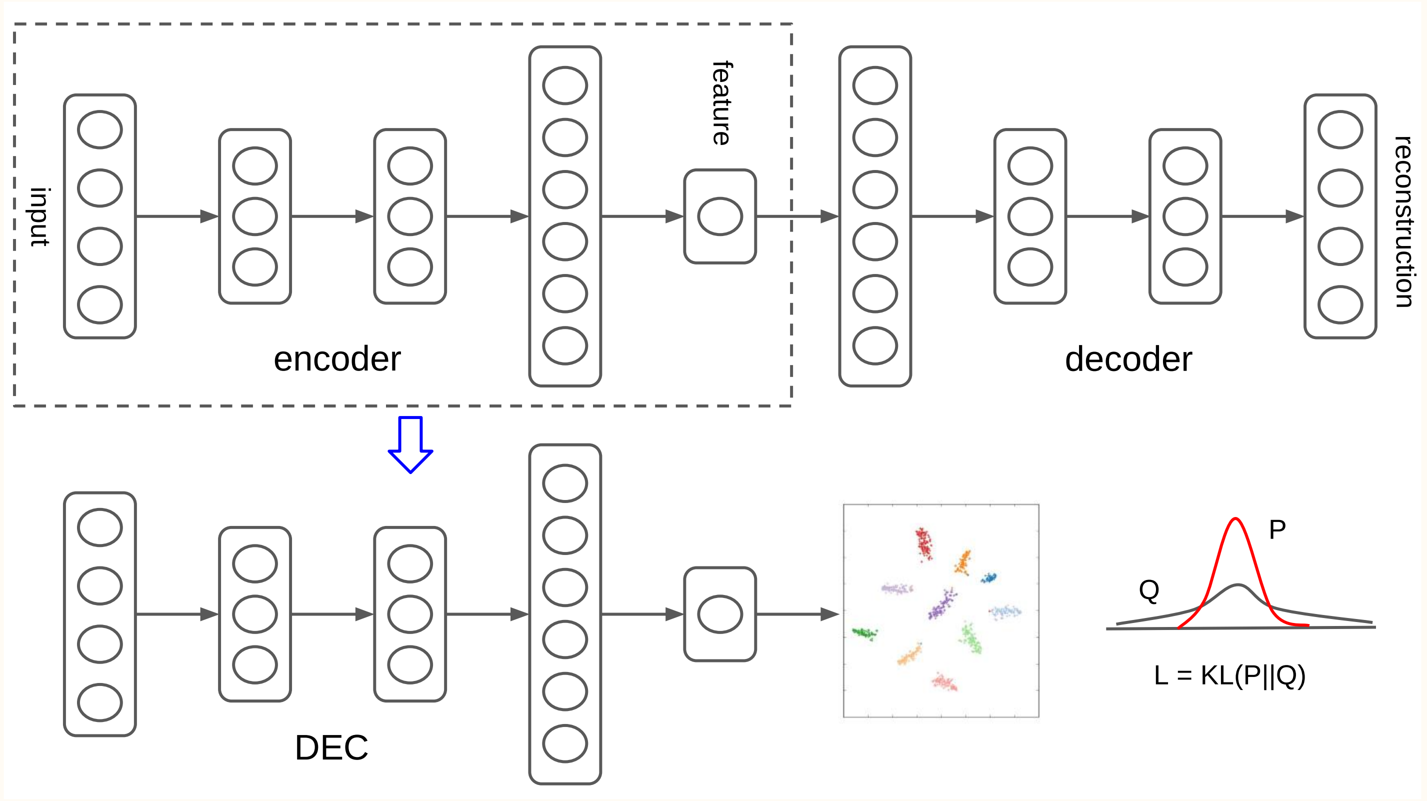
Once an AE has been trained, storing the latent variable representation, rather than the original image can be used as a compression mechanism.

More generally, latent variable representations can be used for video compression.

https://youtu.be/NqmMnjJ6GEg

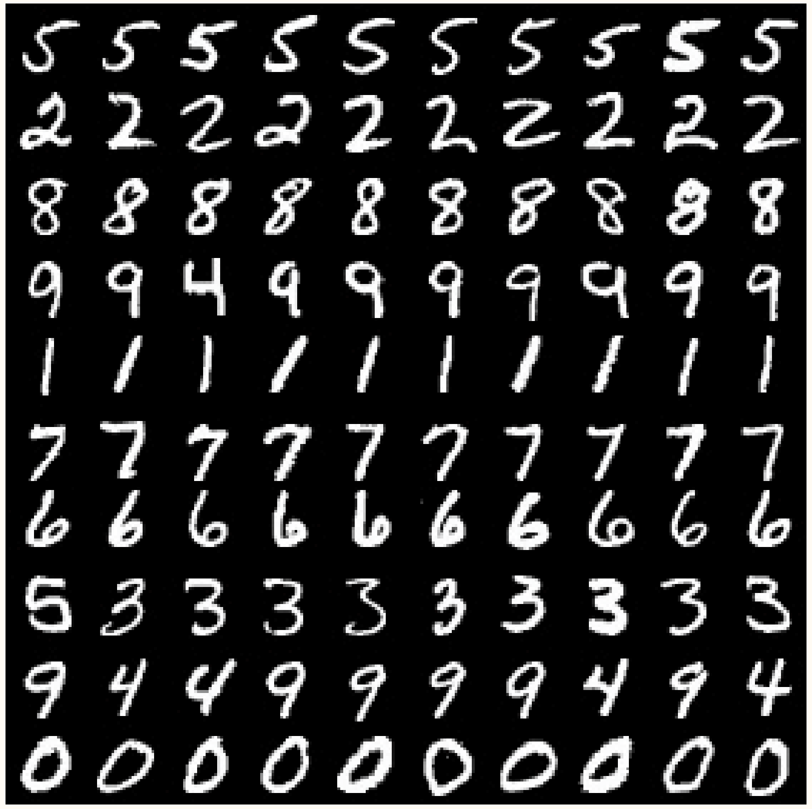
Applications of AE: Clustering

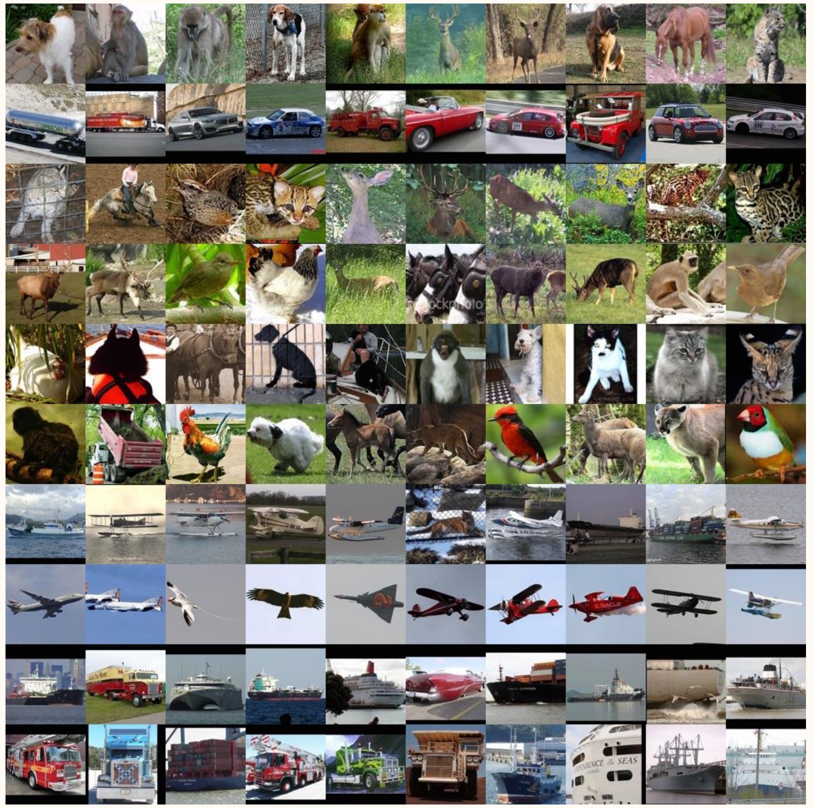
Train an AE and then perform clustering on the latent variables. For the clustering algorithm, one can use things like k-means, which groups together



Applications of AE: Clustering

Clustering is also referred to as unsupervised classification. Without labels, we want the group “similar” data.





Anomaly/outlier detection

Problem: detecting data that is significantly different from the data seen during training.

Insight: AE should not be able to faithfully reconstruct novel data.

Solution: Train an AE and define the *score function* to be the reconstruction loss:

$$s(X) = \left\| X - D_\varphi\big(E_\theta(X)\big) \right\|^2$$

If score is high, determine the datapoint to be an outliner. (Cf. hw7.)

S. Hawkins, H. He, G. Williams, and R. Baxter, Outlier detection using replicator neural networks, *DaWaK*, 2002.

Probabilistic generative models

A *probabilistic generative model* learns a distribution p_θ from $X_1, \dots, X_N \sim p_{\mathrm{true}}$ such that $p_\theta \approx p_{\mathrm{true}}$ and such that we can generate new samples $X \sim p_\theta$.

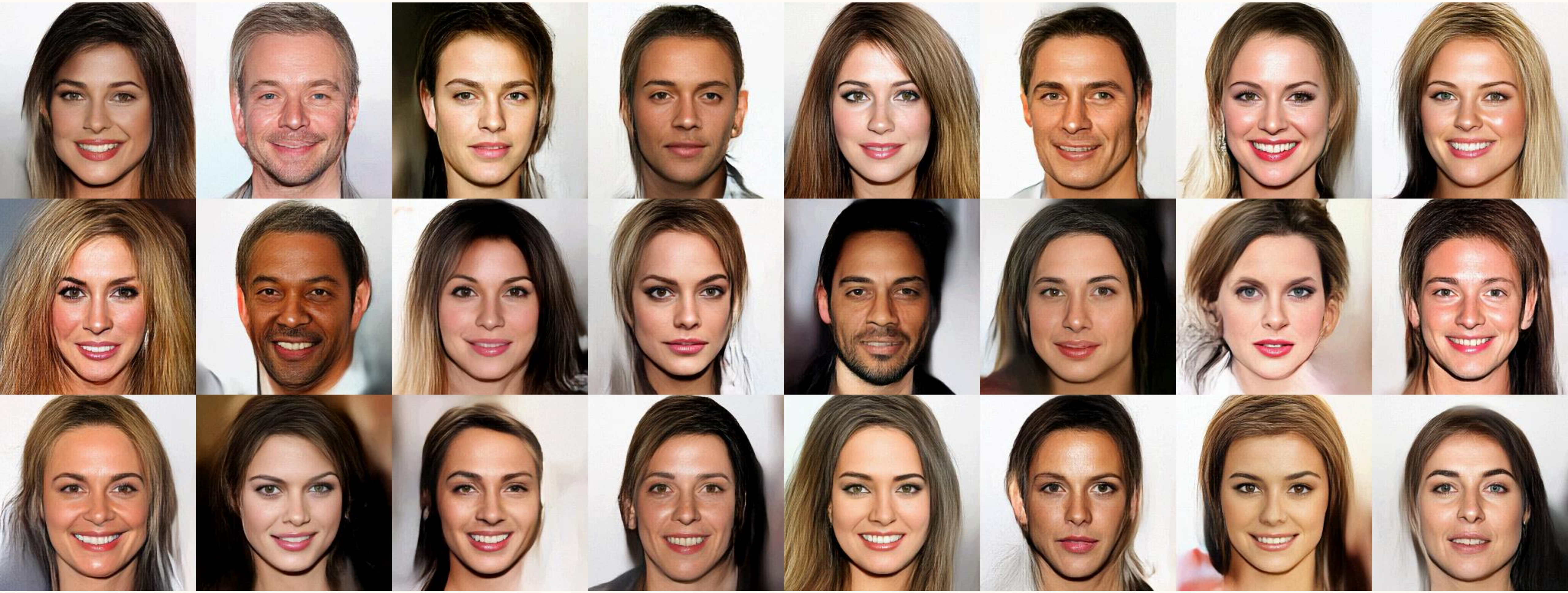
The ability to generate new synthetic data is interesting, but by itself not very useful.*

The structure of the data learned through the unsupervised learning is of higher value. However, we won't talk about the downstream applications in this course.

In this class, we will talk about flow models, VAEs, and GANs.

*Generating fake images to use in fake social media accounts is the only direct application that I can think of.

Flow model: Change of variable formula combined with deep neural networks



Flow models

Fit a probability density function $p_\theta(x)$ with continuous data $X_1, \dots, X_N \sim p_{\text{true}}(x)$.

- We want to fit the data $X_1, \dots, X_N$ (or really the underlying distribution p_{true}) well.
- We want to be able to sample from p_θ.
- (We want to get a good latent representation.)

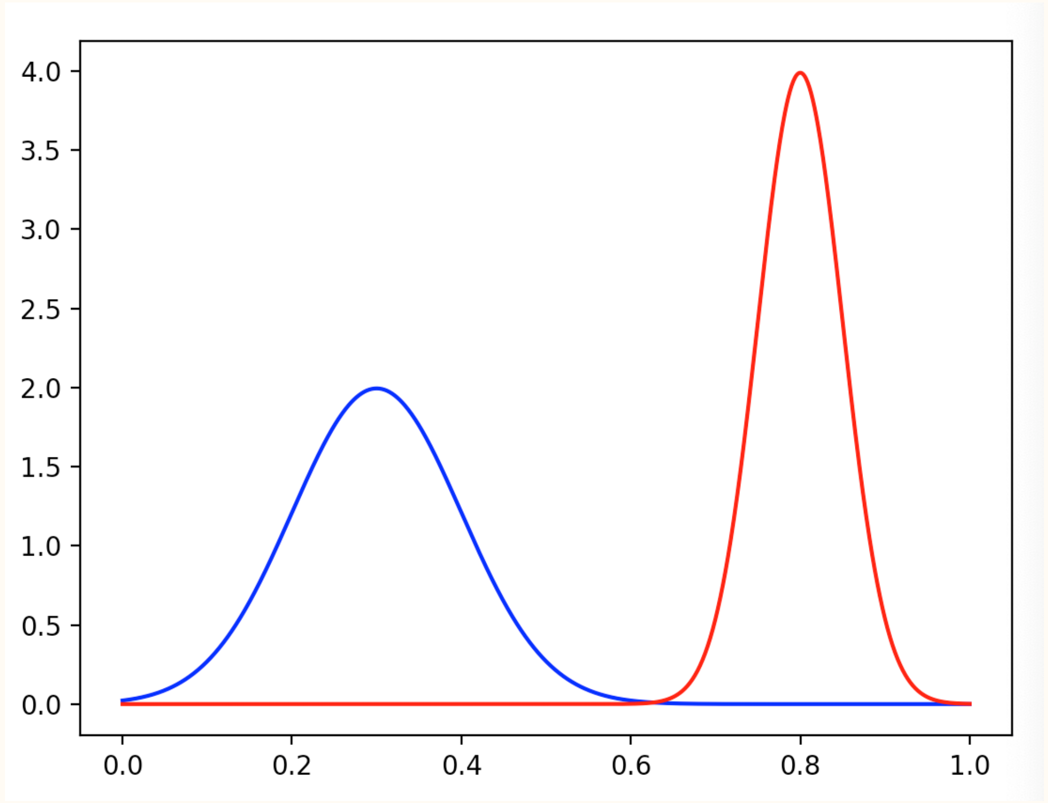
We first develop the mathematical discussion with 1D flows, and then generalize the discussion to high dimensions.

Example density model: Gaussian mixture model

$$p_\theta(x) = \sum_{i=1}^{k} \pi_i \mathcal{N}\left(x; \mu_i, \sigma_i^2\right)$$

Parameters: means and variances of components, mixture weights

$$\theta = (\pi_1, \ldots, \pi_k, \mu_1, \ldots, \mu_k, \sigma_1, \ldots, \sigma_k)$$



Problems with GMM:

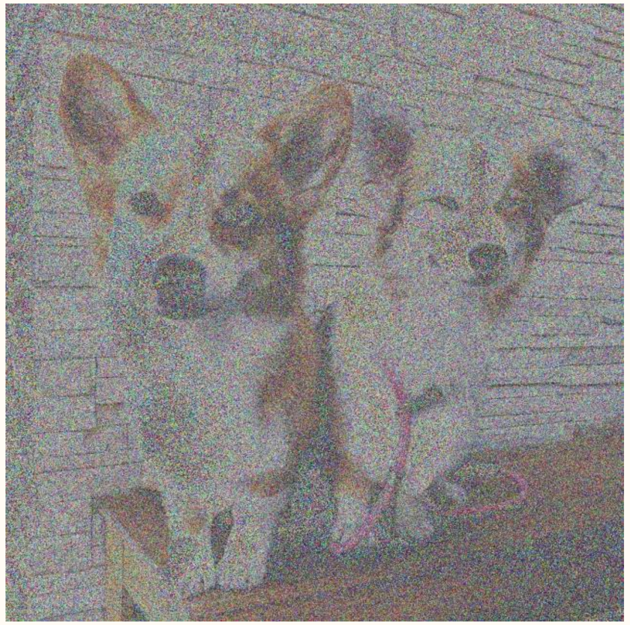
- Highly non-convex optimization problem. Can easily get stuck in local minima.
- It is does not have the representation power to express high-dimensional data.

Example density model: Gaussian mixture model

GMM doesn't work with high-dimensional data. The sampling process is:

1. Pick a cluster center
2. Add Gaussian noise

If this is done with natural images, a realistic image can be generated only if it is a cluster center, i.e., the clusters must already be realistic images.



So then how do we fit a general (complex) density model?

Math review: 1D continuous RV

A random variable X is continuous if there exists a probability density function $p_X(x) \geq 0$ such that

$$\mathbb{P}(a \leq X \leq b) = \int_a^b p_X(x)\,dx$$

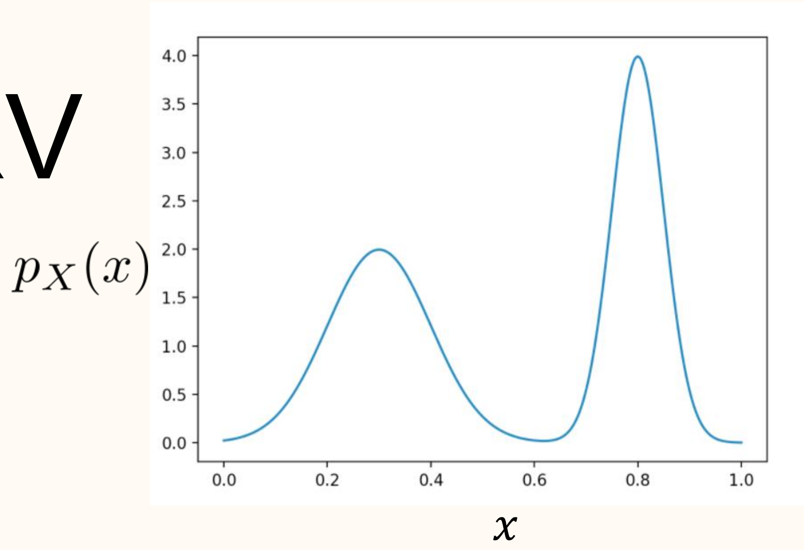
In this case, we write $X \sim p_X$.

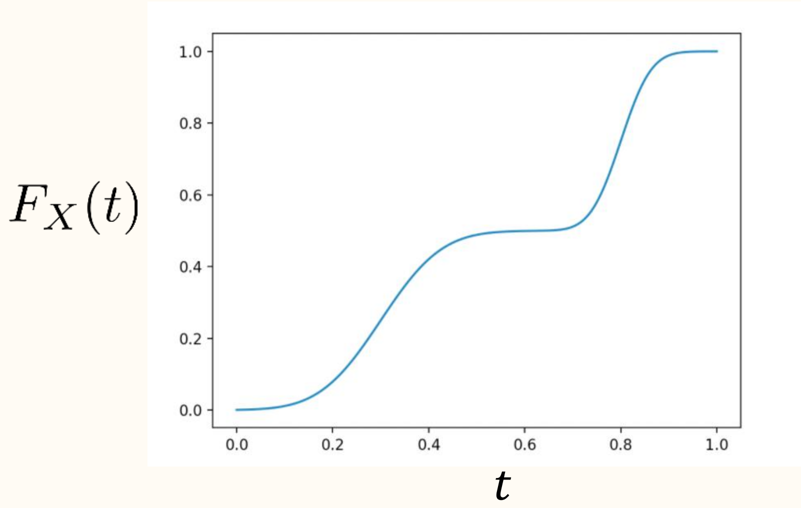
The cumulative distribution function (CDF) of X is defined as

$$F_X(t) = \mathbb{P}(X \leq t) = \int_{-\infty}^{t} p_X(x)\,dx$$

$F_X(t)$ is a nondecreasing function.

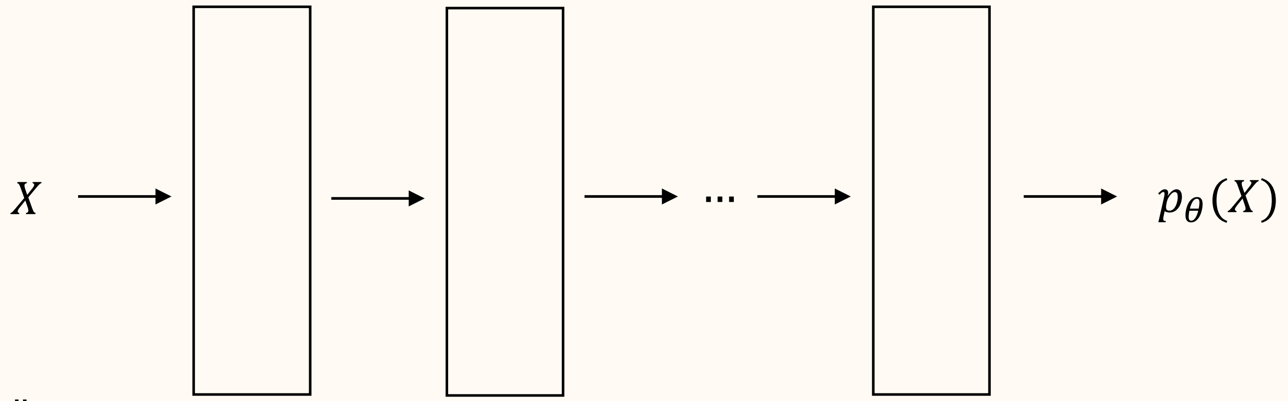
$F_X(t)$ is a continuous function if X is a continuous random variable.





Naïve approach: prameterize p_θ as DNN

Naïve approach for fitting a density model. Represent $p_\theta(x)$ with DNN.



There are some challenges:

1.How to ensure proper distribution?

$$\int_{-\infty}^{+\infty} p_\theta(x)dx = 1, \qquad p_\theta(x) \geq 0, \qquad x \in \mathbb{R}$$

2. How to sample?

Normalization of p_θ

For discrete random variables, one can use the soft-max function $\mu : \mathbb{R}^k \to \mathbb{R}^k$ defined as

$$\mu_i(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

to normalize probabilities.

For continuous random variables, we can ensure $p_\theta \geq 0$ with $p_\theta(x) = e^{f_\theta(x)}$, where f_θ is the output of the neural network. However, ensuring the normalization

$$\int_{-\infty}^{+\infty} p_\theta(x) dx = 1$$

is not a simple matter. (Any Bayesian statistician can tell you how difficult this is.)

What happens if we ignore normalization?

Do we really need this normalization thing? Yes, we do.

Without normalization, one can just assign arbitrarily large probabilities everywhere when we perform maximum likelihood estimation:

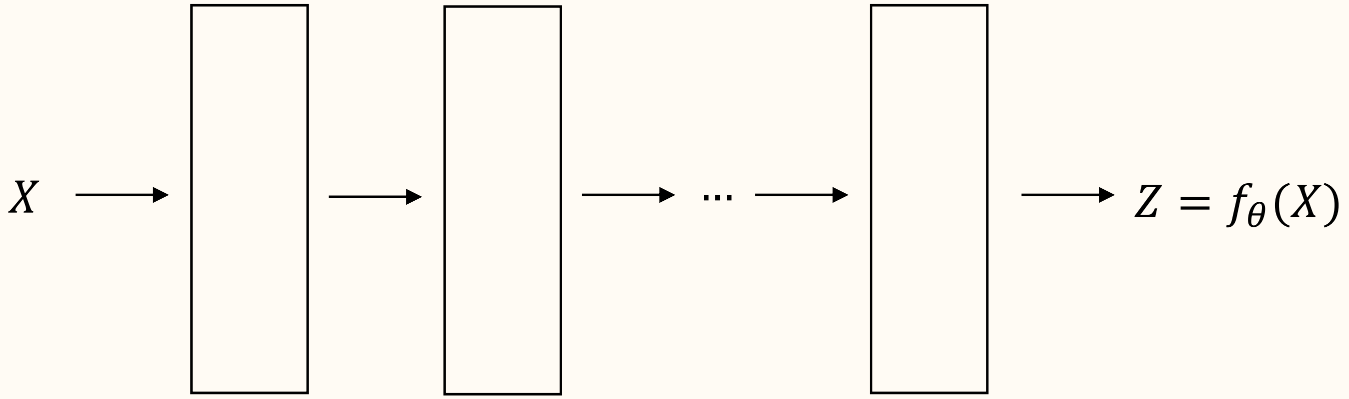
$$\underset{\theta \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_\theta \left(X_i \right)$$

The solution is to set $p_\theta(x) = M$ with $M \to \infty$.

We want model to place large probability on data $X_1, \ldots, X_N$ while placing small probability elsewhere. Normalization forces model to place small probability where data doesn't reside.

Key insight: Parameterize $Z = f_\theta(X)$ with DNN

Key insight of normalizing flow: DNN outputs random variable Z, rather than $p_\theta(X)$



In *normalizing flow*, find θ such that the flow f_θ *normalizes* the random variable $X \sim p_X$ into $Z \sim \mathcal{N}(0,1)^*$.

Important questions to resolve:

1. How to train? (How to evaluate $p_\theta(x)$? DNN outputs f_θ, not p_θ.)
2. How to sample X?

*Generally, we can consider $Z \sim p_Z$. The choice of p_Z, however, does not seem to make a significant difference.

1D change of variable formula

Assume f is invertible, f is differentiable, and f^{-1} is differentiable.

If $X \sim p_X$, then $Z = f(X)$ has pdf

$$p_Z(z) = p_X\big(f^{-1}(z)\big)\left|\frac{dx}{dz}\right|$$

If $Z \sim p_Z$, then $X = f^{-1}(Z)$ has pdf

$$p_X(x) = p_Z\big(f(x)\big)\left|\frac{df(x)}{dx}\right|$$

Since $Z = f(X)$, one might think $p_X(x) = p_Z(z) = p_Z\big(f(x)\big)$. ← This is wrong.

Invertibility of f is essential; it is not a minor technical issue.

Training flow models

Train model with MLE

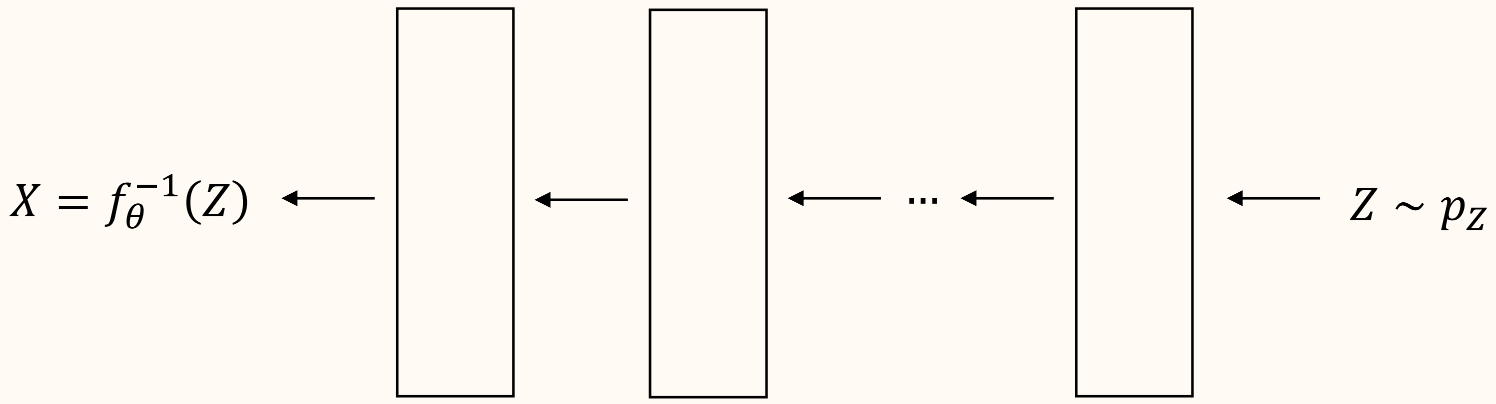
$$\underset{\theta \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_\theta(X_i) \quad = \quad \underset{\theta \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_Z(f_\theta(X_i)) + \log \left| \frac{\partial f_\theta}{\partial x}(X_i) \right|$$

where f_θ is invertible and differentiable, and $X = f_\theta^{-1}(Z)$ with $Z \sim p_Z$ so

$$p_X(x) = p_Z\big(f_\theta(x)\big) \left| \frac{\partial f_\theta}{\partial x}(x) \right|.$$

Can optimize with SGD, if we know how to perform backprop on $\left| \frac{\partial f_\theta}{\partial x}(X_i) \right|$. More on this later.

Sampling from flow models



Step 1: Sample $Z \sim p_Z$

Step 2: Compute $X = f_\theta^{-1}(Z)$

Requirements of flow f_θ

Theoretical requirement:

- $f_\theta(x)$ invertible and differentiable.

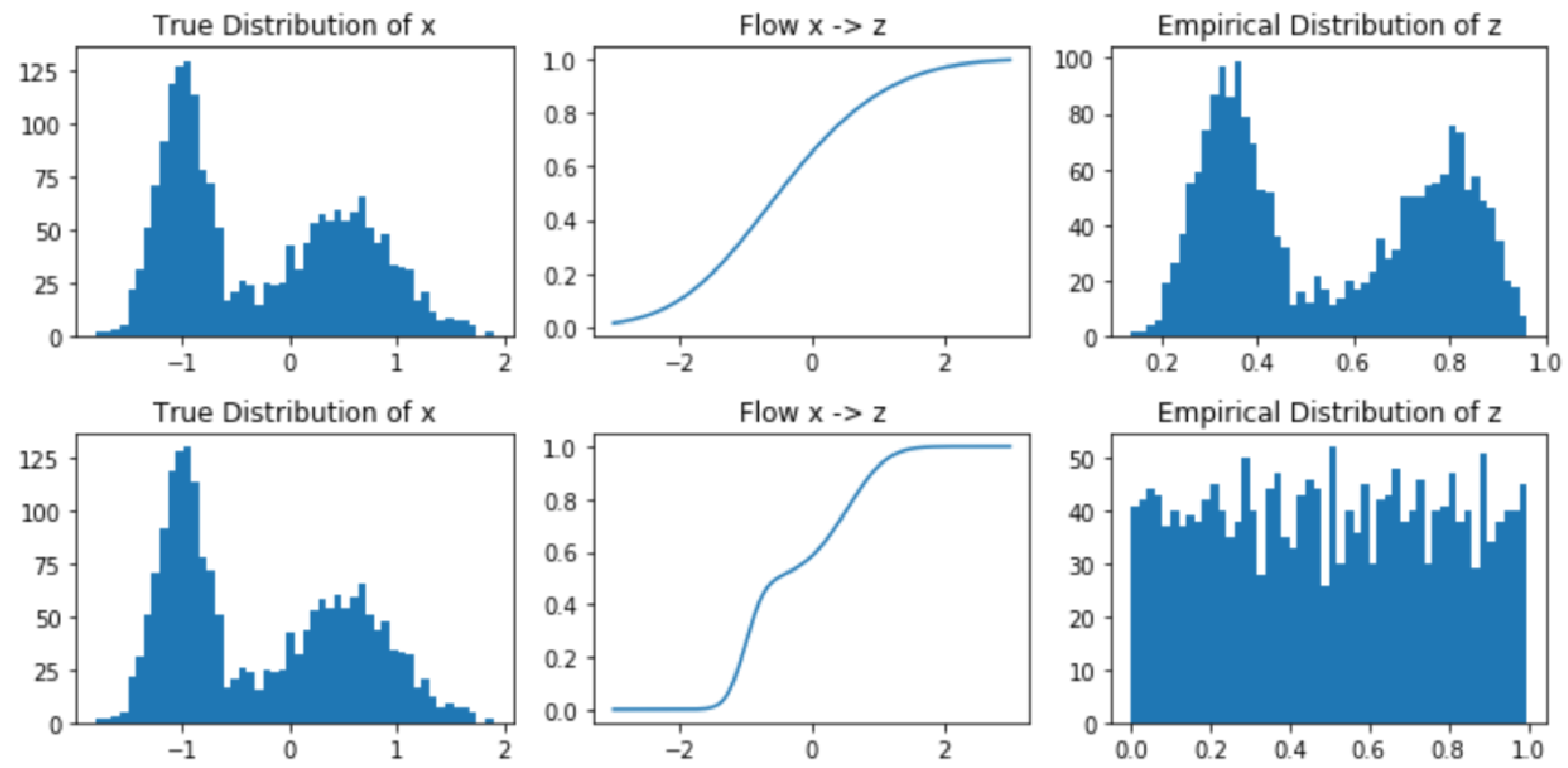
Computational requirements:

- $f_\theta(x)$ and $\nabla_\theta f_\theta(x)$ efficient to evaluate (for training)
- $\left|\frac{\partial f_\theta}{\partial x}(x)\right|$ and $\nabla_\theta \left|\frac{\partial f_\theta}{\partial x}(x)\right|$ efficient to evaluate (for training)
- f_θ^{-1} efficient to evaluate (for sampling)

Example: Flow to $Z \sim \text{Uniform}([0,1])$

Before training

After training

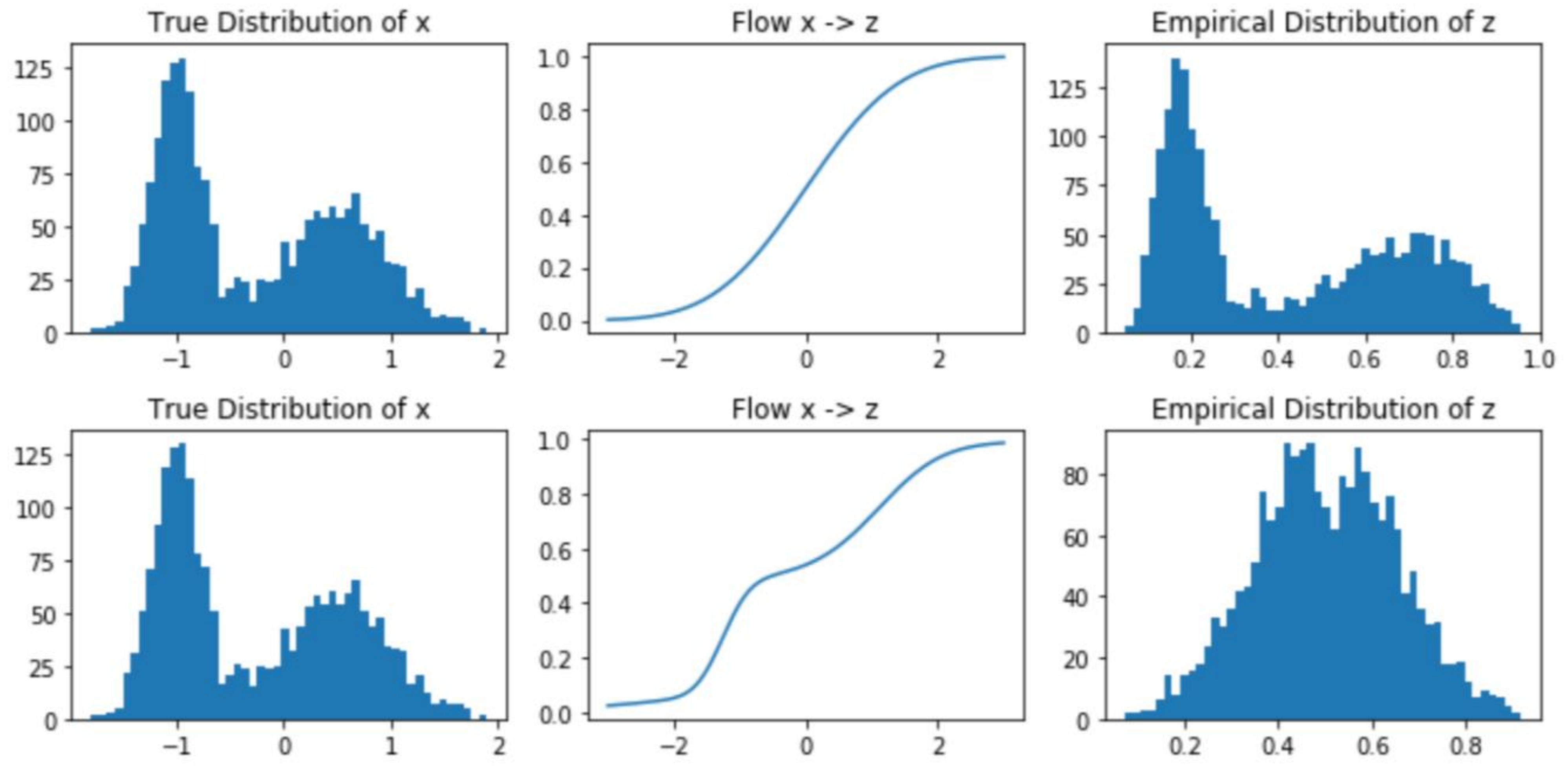


True distribution of x   Flow x → z   Empirical distribution of z

Example: Flow to $Z \sim \text{Beta}(5,5)$

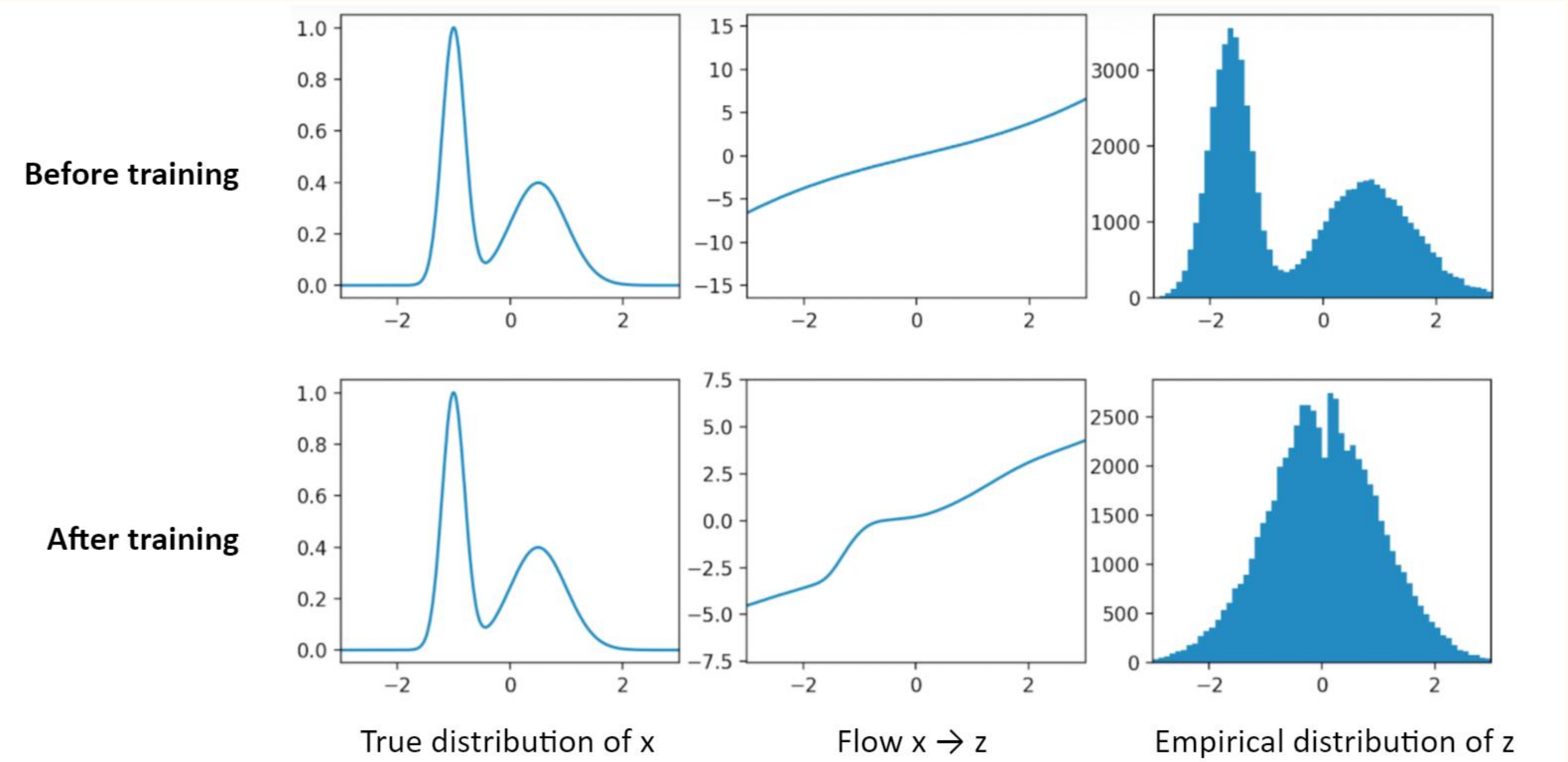
Before training

After training



True distribution of x Flow x → z Empirical distribution of z

Example: Flow to $Z \sim \mathcal{N}(0,1)$



1D flow demonstration

PyTorch demo

Universality of flows

Are flows universal, i.e., can $f_\theta^{-1}(Z) \sim p_X$ for any X provided that f_θ can represent any invertible function?

Yes, 1D flows are universal due to the inverse CDF sampling technique.[*]

Higher dimensional flows are also universal as shown by Huang et al.[#] or earlier by the general theory of optimal transport.[%]

[*]Some basic conditions are being omitted.
[#]C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville, Neural Autoregressive Flows, *ICML*, 2018.
[%]https://en.wikipedia.org/wiki/Transportation_theory_(mathematics)

Math review: Sampling via inverse CDF

Inverse CDF sampling is a technique for sampling $X \sim p_X$.

If $F_X(t)$ is furthermore a strictly increasing function, then F_X is invertible, i.e., F_X^{-1} exists.

Generate a random number $U \sim \mathrm{Uniform}([0,1])$ and compute $F_X^{-1}(U)$. Then

$$F_X^{-1}(U) \sim p_X$$

since

$$\mathbb{P}(F_X^{-1}(U) \le t) = \mathbb{P}\big(U \le F_X(t)\big) = F_X(t)$$

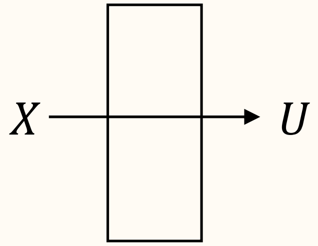
Technique can be generalized to when F_X is not invertible.

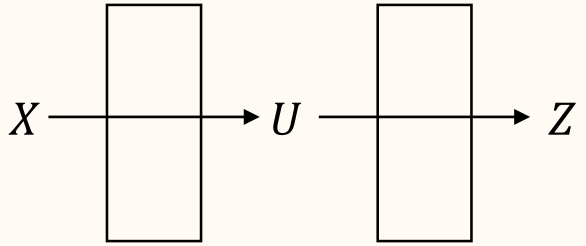
Universality of 1D flows

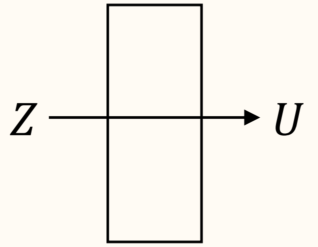
Composition of flows is a flow, and inverse of a flow is a flow

Universality of 1D flows:

- Use inverse CDF as flow to transform $X \sim p_X$ into $U \sim \mathrm{Uniform}([0,1])$ and $Z \sim \mathcal{N}(0,1)$ into $U \sim \mathrm{Uniform}([0,1])$.
- Compose flow $X \to U$ and inverse flow $U \to Z$







Jacobian notation

Let $f : \mathbb{R}^n \to \mathbb{R}^n$, such that

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix}$$

The Jacobian matrix is

$$\frac{\partial f}{\partial x}(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \cdots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \cdots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \frac{\partial f_n}{\partial x_2}(x) & \cdots & \frac{\partial f_n}{\partial x_n}(x) \end{bmatrix} = \begin{bmatrix} \left(\nabla f_1(x)\right)^\top \\ \left(\nabla f_2(x)\right)^\top \\ \vdots \\ \left(\nabla f_n(x)\right)^\top \end{bmatrix}$$

The Jacobian determinant is $\det\left(\frac{\partial f}{\partial x}\right)$. We use the notation

$$\left|\frac{\partial f}{\partial x}(x)\right| = \left|\det\left(\frac{\partial f}{\partial x}(x)\right)\right|$$

where the second $|\cdot|$ is the absolute value of the determinant. (This notation is not completely standard.)

Math review: Multivariate change of variables

Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be an invertible function such that both f and f^{-1} are differentiable. Let $U \subseteq \mathbb{R}^n$. Then

$$\int_{f(U)} h(v)\, dv = \int_{U} h\big(f(u)\big) \left| \frac{\partial f}{\partial u}(u) \right| du$$

for any $h : \mathbb{R}^n \to \mathbb{R}$. (Change of variable from $v = f(u)$ to $u = f^{-1}(v)$.)

The conditions for this change of variable formula can be further generalized.

Math review: Multivariate continuous RV

A multivariate random variable $X \in \mathbb{R}^n$ is continuous if there exists a probability density function $p_X(x)$ such that

$$\mathbb{P}(X \in A) = \int_A p_X(x)\, dx$$

where the integral is over the volume $A \subseteq \mathbb{R}^n$. In this case, we write $X \sim p_X$.

The joint cumulative distribution function (the copula) does not seem to be useful in the context of high-dimensional flow models.

Math review: Mult. change of variables for RV

Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be an invertible function such that both f and f^{-1} are differentiable. Let X be a continuous random variable with probability density function p_X and let $Y = f(X)$ have density p_Y. Then

$$p_X(x) = p_Y\big(f(x)\big)\left|\frac{\partial f}{\partial x}(x)\right|$$

Proof)

$$\mathbb{P}(f^{-1}(Y) \in A) = \mathbb{P}\big(Y \in f(A)\big) = \int_{f(A)} p_Y(y)\, dy = \int_A p_Y\big(f(x)\big)\left|\frac{\partial f}{\partial x}(x)\right| dx = \mathbb{P}(X \in A)$$

■

Invertibility of f is essential; it is not a minor technical issue.

Math review: Determinant formulae

Fact: Determinant definitions in undergraduate linear algebra textbooks require exponentially many operations to compute:

$$\det(A) = \sum_{\sigma \in S_n} \left(\operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i} \right)$$

Efficient computation of determinant for general matrices and performing backprop through the computation is difficult. Therefore, high-dimensional flow model are designed to compute determinants only on simple matrices.

Product formula: if A and B are square, then

$$\det(AB) = \det(A)\det(B)$$

Block lower triangular formula: if $A \in \mathbb{R}^{n \times n}$ and $C \in \mathbb{R}^{m \times m}$, then

$$\det\begin{pmatrix} A & 0 \\ B & C \end{pmatrix} = \det(A)\det(C)$$

Lower triangular formula: if $a_1, \ldots, a_n \in \mathbb{R}$ and $*$ represents arbitrary values, then

$$\det\begin{pmatrix} a_1 & 0 & \cdots & 0 \\ * & a_2 & & \vdots \\ * & * & \ddots & 0 \\ * & * & * & a_n \end{pmatrix} = \prod_{i=1}^{n} a_i$$

Upper triangular formula: same as for lower triangular matrices.

Training high-dim flow models

Train model with MLE

$$\underset{\theta\in\mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_\theta\left(X_i\right) \quad = \quad \underset{\theta\in\mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_Z\left(f_\theta(X_i)\right) + \log\left|\frac{\partial f_\theta}{\partial x}(X_i)\right|$$

where $f_\theta(z)$ is invertible and differentiable, and $X = f^{-1}(Z)$ with $Z \sim p_Z$ so

$$p_X(x) = p_Z\big(f_\theta(x)\big)\left|\frac{\partial f_\theta}{\partial x}(x)\right|.$$

(Exactly the same formula as with 1D flow.)

Can optimize with SGD, if we know how to perform backprop on $\left|\frac{\partial f_\theta}{\partial x}(X_i)\right|$.

Composing flows

Flows can be composed to increase expressiveness. (Deep NN more expressive.)

Consider composition of k flows

$$x \to f_1 \to f_2 \to \cdots \to f_k \to z$$

$$z = f_k \circ \cdots \circ f_1 (x)$$

$$x = f_1^{-1} \circ \cdots \circ f_k^{-1} (z)$$

Determinant computation splits nicely due to chain rule and product formula

$$\det\left(\frac{\partial z}{\partial x}\right) = \det\left(\frac{\partial f_k}{\partial f_{k-1}} \cdots \frac{\partial f_1}{\partial f_0}\right) = \det\left(\frac{\partial f_k}{\partial f_{k-1}}\right) \cdots \det\left(\frac{\partial f_1}{\partial f_0}\right)$$

$$\log p_\theta(x) = \log p_\theta(z) + \sum_{i=1}^{k} \log\left|\frac{\partial f_i}{\partial f_{i-1}}\right|$$

Basic example: Affine flows

An affine (linear) transformation

$$f_{A,b}(x) = A^{-1}(x - b)$$

is a flow if matrix A is invertible. Then

$$\frac{\partial f_{A,b}}{\partial x} = A^{-1}$$

and

$$\left|\frac{\partial f_{A,b}}{\partial x}\right| = |\det(A^{-1})| = \frac{1}{|\det(A)|}$$

Sampling: $X = AZ + b$, where $Z \sim \mathcal{N}(0, I)$.

Problem with affine flows:

- Computing $|\det(A)|$ is expensive and performing backprop over it is difficult. We want $\frac{\partial f_{A,b}}{\partial x}$ to be further structured so that determinant is easy to compute.
- One affine flow is insufficient to generate complex data. However, composing multiple affine flows yields an affine flow and therefore is pointless. We need to introduce nonlinearities.

Coupling flows

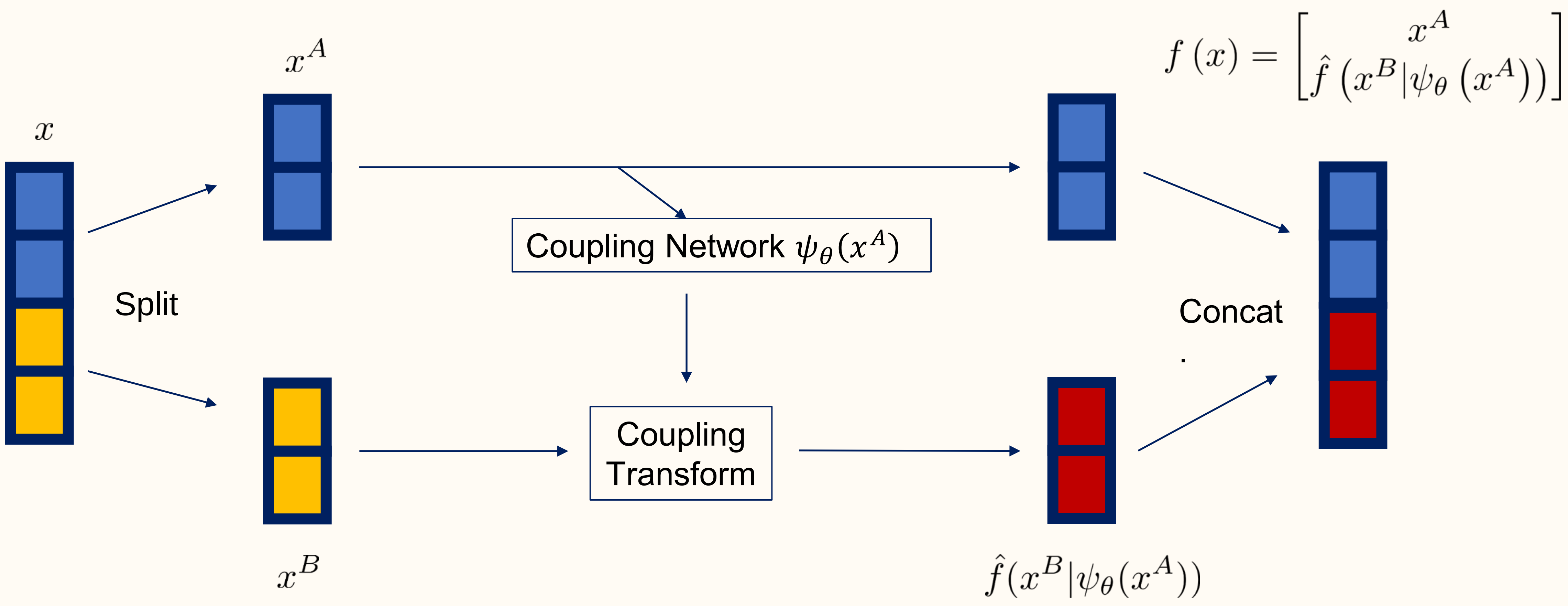
A coupling flow is a general and practical approach for constructing non-linear flows.

Partition input into two disjoint subsets $x = (x^A, x^B)$. Then

$$f(x) = \left(x^A, \hat{f}(x^B | \psi_\theta(x^A))\right)$$

where ψ_θ is a neural network and $\hat{f}(x^B | \psi_\theta(x^A))$ is another flow whose parameters depend on x^A.

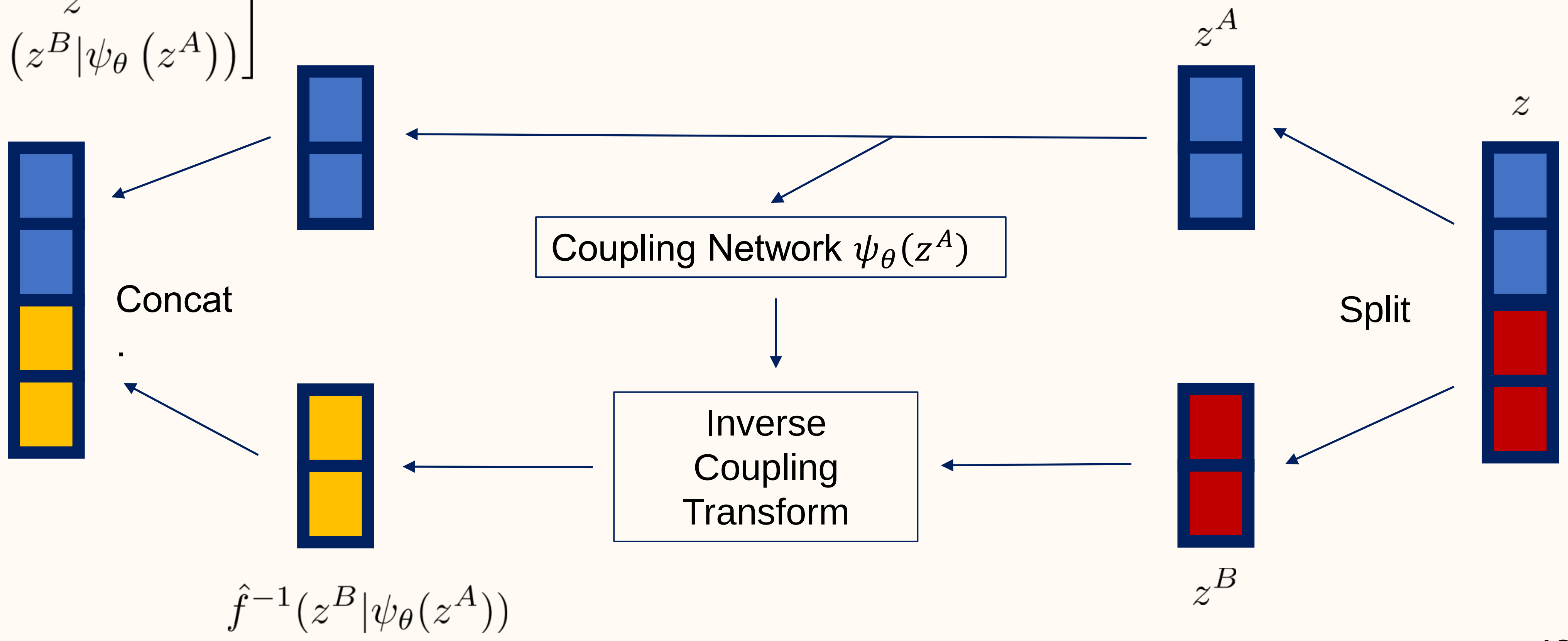
Coupling flow: forward evaluation



x
x^A
x^B
Split
Coupling Network $\psi_\theta(x^A)$
Coupling Transform
$\hat{f}(x^B|\psi_\theta(x^A))$
Concat
$f(x) = \begin{bmatrix} x^A \\ \hat{f}\left(x^B|\psi_\theta\left(x^A\right)\right) \end{bmatrix}$

Coupling flow: inverse evaluation

$$f^{-1}(z) = \begin{bmatrix} z^A \\ \hat{f}^{-1}\left(z^B | \psi_\theta\left(z^A\right)\right) \end{bmatrix}$$



z^A
z
Coupling Network $\psi_\theta(z^A)$
Concat
.
Split
Inverse Coupling Transform
$\hat{f}^{-1}(z^B|\psi_\theta(z^A))$
z^B

Jacobian of coupling flows

The Jacobian of a coupling flow has a nice block structure

$$\frac{\partial f_\theta}{\partial x}(x) = \begin{bmatrix} I & 0 \\ \frac{\partial \hat{f}}{\partial x^A}\left(x^B | \psi_\theta\left(x^A\right)\right) & \frac{\partial \hat{f}}{\partial x^B}\left(x^B | \psi_\theta(x^A)\right) \end{bmatrix}$$

which leads to the simplified determinant formula

$$\det\left(\frac{\partial f_\theta}{\partial x}(x)\right) = \det\left(\frac{\partial \hat{f}}{\partial x^B}\left(x^B | \psi_\theta(x^A)\right)\right)$$

Note $\frac{\partial \hat{f}}{\partial x^A}\left(x^B | \psi_\theta\left(x^A\right)\right)$, which will be very complicated, does not appear in the determinant.

Coupling transformation $\hat{f}(x|\psi)$

Additive transformations (NICE)*

$$\hat{f}(x|\psi) = x + t$$

where $\psi = t$.

Affine transformations (Real NVP)#

$$\hat{f}(x|\psi) = e^s \odot x + t$$

where $\psi = (s, t)$.

Other transformations studied throughout the literature.

*L. Dinh, D. Krueger, and Y. Bengio, NICE: Non-linear independent components estimation, *ICLR Workshop*, 2015.
#L. Dinh, J. Sohl-Dickstein, and S. Bengio, Density estimation using Real NVP, *ICLR*, 2017.

NICE (Non-linear Independent Components Estimation)

NICE uses additive coupling layers:

Split variables in half: $x_{1:n/2}$, $x_{n/2:n}$

$$z_{1:n/2} = x_{1:n/2}$$

$$z_{n/2:n} = x_{n/2:n} + t_\theta(x_{1:n/2})$$

Easily invertible:

$$x_{1:n/2} = z_{1:n/2}$$

$$x_{n/2:n} = z_{n/2:n} - t_\theta(x_{1:n/2})$$

Jacobian determinant is easy to compute:

$$\det \frac{\partial f_\theta}{\partial x}(x) = \det \begin{bmatrix} I & 0 \\ \frac{\partial \hat{f}}{\partial x^A}\left(x^B | \psi_\theta\left(x^A\right)\right) & \frac{\partial \hat{f}}{\partial x^B}\left(x^B | \psi_\theta(x^A)\right) \end{bmatrix} = \det \begin{bmatrix} I & 0 \\ \frac{\partial \hat{f}}{\partial x^A}\left(x^B | \psi_\theta\left(x^A\right)\right) & I \end{bmatrix} = 1$$

Real NVP (Real-valued Non-Volume Preserving)

Real NVP uses affine coupling layers:

$$z_{1:n/2} = x_{1:n/2}$$

$$z_{n/2:n} = e^{s_\theta(x_{1:n/2})} \odot x_{n/2:n} + t_\theta(x_{1:n/2})$$

Easily invertible:

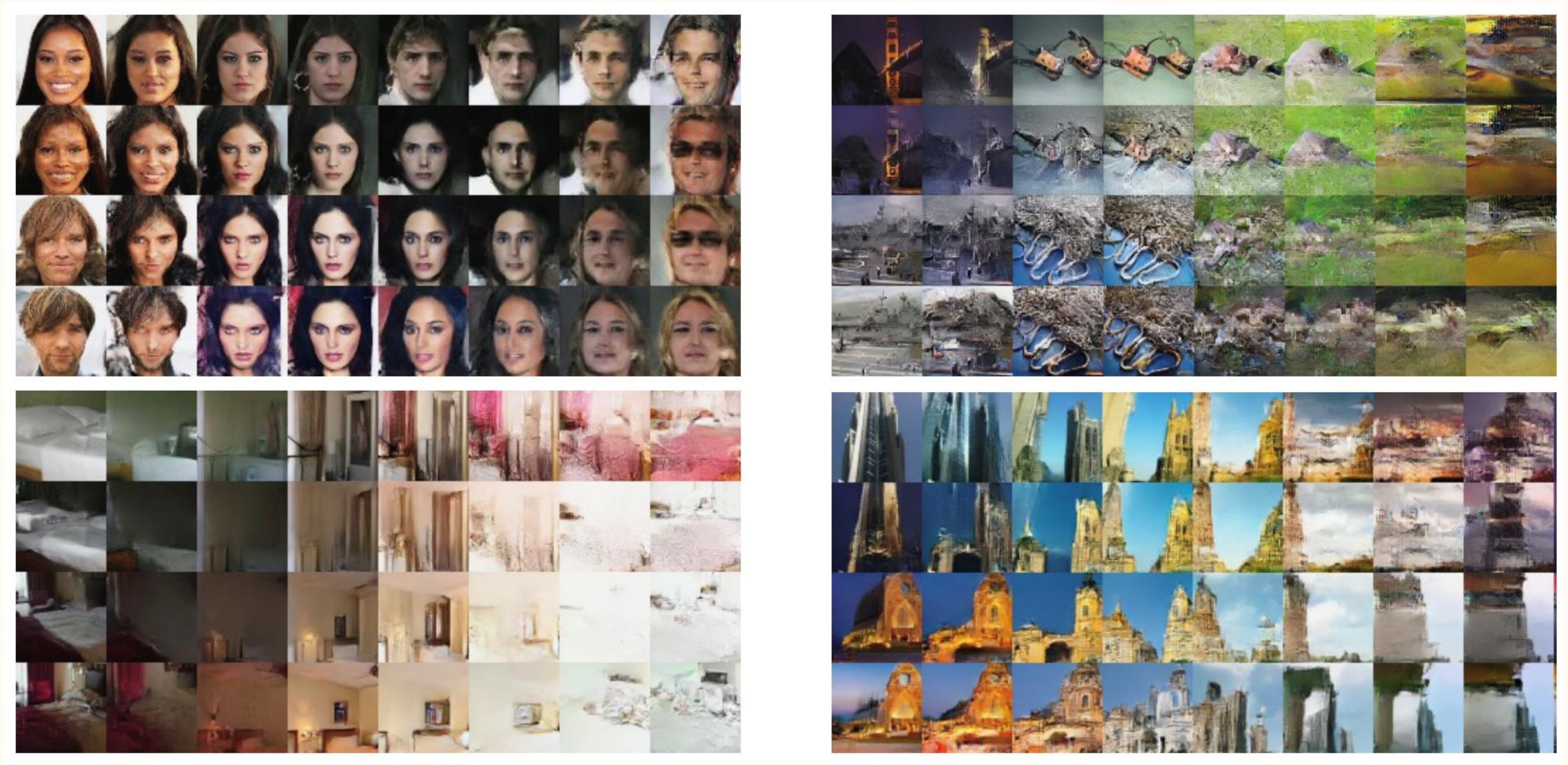
$$x_{1:n/2} = z_{1:n/2}$$

$$x_{n/2:n} = (z_{n/2:n} - t_\theta(x_{1:n/2})) \odot e^{-s_\theta(x_{1:n/2})}$$

Jacobian determinant is easy to compute:

$$\begin{aligned}\det \frac{\partial f_\theta}{\partial x}(x) &= \det \begin{bmatrix} I & 0 \\ \frac{\partial \hat{f}}{\partial x^A}\left(x^B | \psi_\theta\left(x^A\right)\right) & \frac{\partial \hat{f}}{\partial x^B}\left(x^B | \psi_\theta(x^A)\right) \end{bmatrix} \\ &= \det \begin{bmatrix} I & 0 \\ \frac{\partial \hat{f}}{\partial x^A}\left(x^B | \psi_\theta\left(x^A\right)\right) & \operatorname{diag}(e^{s_\theta(x_{1:n/2})}) \end{bmatrix} = \exp(\mathbf{1}^\intercal_{n/2} s_\theta(x_{1:n/2}))\end{aligned}$$

Real NVP - Results



How to partition variables?

Note that the additive and affine coupling layers of NICE and Real NVP are nonlinear mappings from $x_{1:n}$ to $z_{1:n}$, since $s_\theta(x_{1:n/2})$ and $t_\theta(x_{1:n/2})$ are nonlinear.

Flow models compose multiple nonlinear flows. But if $x_{1:n/2}$ is always unchanged, then the full composition will leave it unchanged. Therefore, we change the partitioning for every coupling layer.

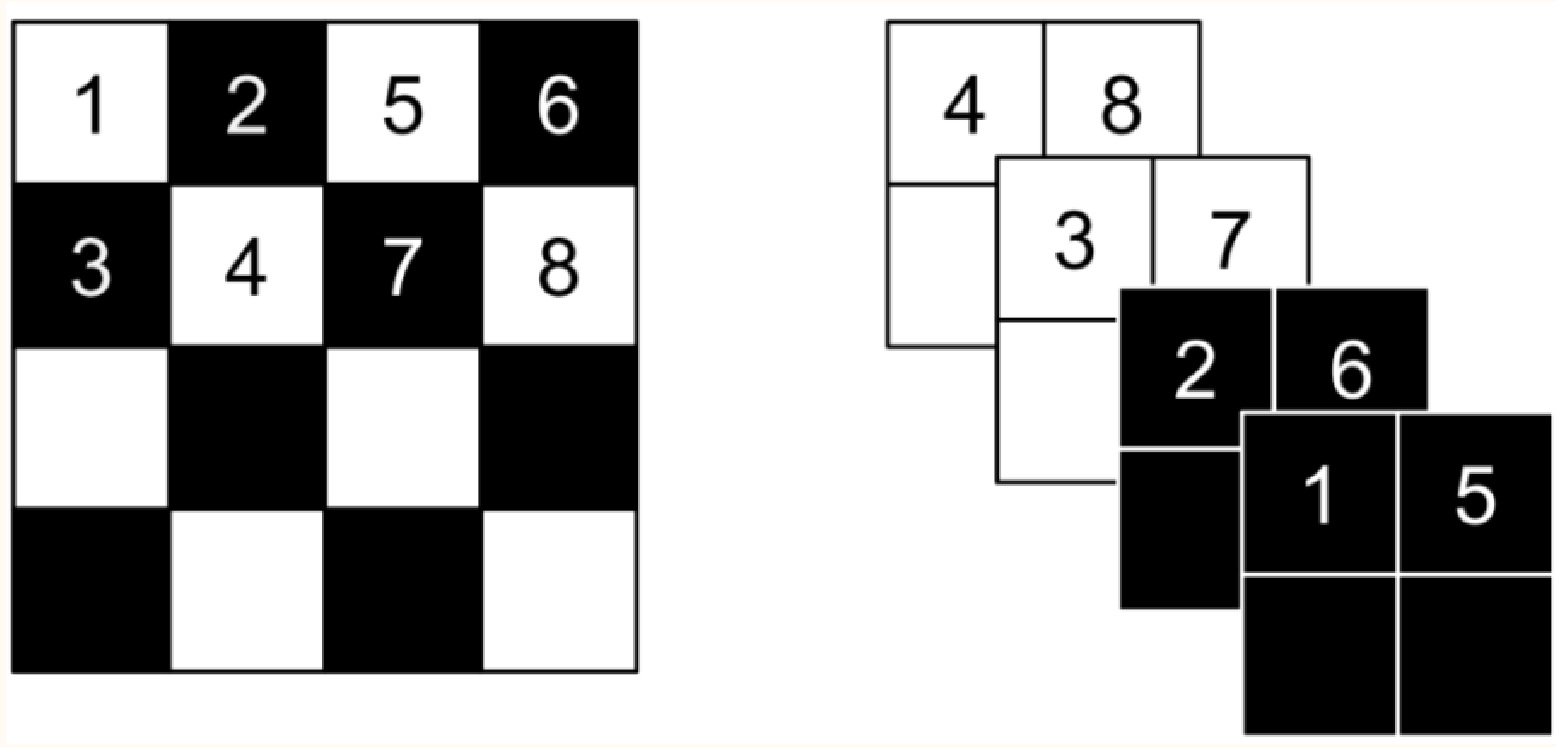
NICE architecture

PyTorch demo

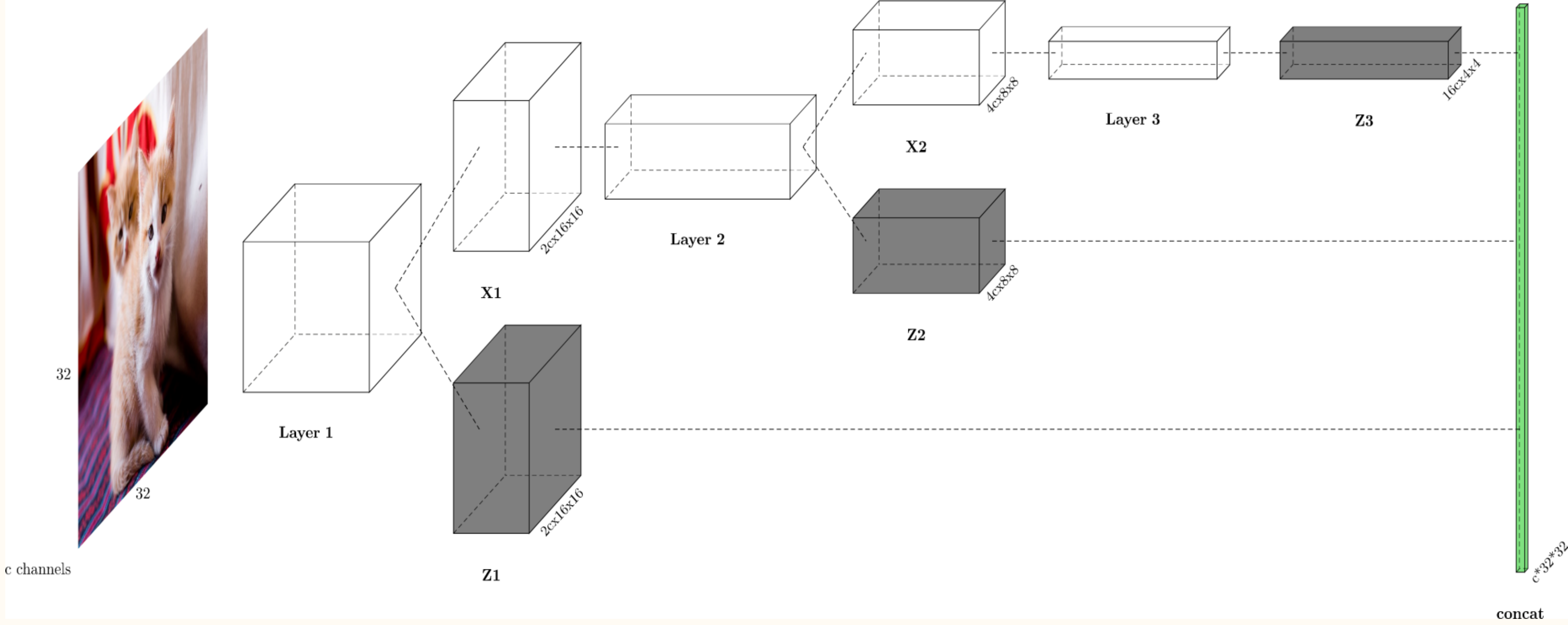
Real NVP variable partitioning

Two partition strategies:

1. Partition with checkerboard pattern.
2. Reshape tensor and then partition channelwise.



Real NVP Architecture



Input X: $c \times 32 \times 32$ image with $c = 3$

Layer 1: Input X: $c \times 32 \times 32$

- Checkerboard $\times$ 3, channel reshape into $4c \times 16 \times 16$, channel $\times$ 3
- Output: Split result to get X_1: $2c \times 16 \times 16$ and Z_1: $2c \times 16 \times 16$ (fine-grained latents)

Layer 2: Input X_1: $2c \times 16 \times 16$ from layer 1

- Checkerboard $\times$ 3, channel reshape into $8c \times 8 \times 8$, channel $\times$ 3
- Split result to get X_2: $4c \times 8 \times 8$ and Z_2: $4c \times 8 \times 8$ (coarser latents)

Layer 3: Input X_2: $4c \times 8 \times 8$ from layer 2

- Checkerboard $\times$ 3, channel reshape into $16c \times 4 \times 4$, channel $\times$ 3
- Get Z_3: $16c \times 4 \times 4$ (latents for highest-level details)

Output $Z = (Z_1, Z_2, Z_3) \in \mathbb{R}^{c \cdot 32^2}$

Batch normalization

To train deep flows, BN is helpful. However, the large model size forces the use of small batch sizes, and BN is not robust with small batch sizes. RealNVP uses a modified form of BN

$$x \mapsto \frac{x - \tilde{\mu}}{\sqrt{\tilde{\sigma}^2 + \varepsilon}}$$

(No β and γ parameters.) This layer has the log Jacobian determinant

$$-\frac{1}{2} \sum_i \log(\tilde{\sigma}_i^2 + \varepsilon)$$

The mean and variance parameters are updated with

$$\begin{aligned} \tilde{\mu}_{k+1} &= \rho \tilde{\mu}_k + (1 - \rho) \hat{\mu}_k \\ \tilde{\sigma}_{k+1}^2 &= \rho \tilde{\sigma}_k^2 + (1 - \rho) \hat{\sigma}_k^2 \end{aligned}$$

where ρ is the momentum. During gradient computation, only backprop through the current batch statistics $\hat{\mu}_k$ and $\hat{\sigma}_k^2$.

s_θ and t_θ networks

The s_θ and t_θ do not need to be invertible. The original RealNVP paper does not describe its construction.

We let (s_θ, t_θ) be a deep (20-layer) convolutional neural network using residual connections and standard batch normalization.

Real NVP architecture

PyTorch demo

Glow paper

The authors of the Glow paper also released a blog post.

https://openai.com/blog/glow/

FFJORD

Instead of a discrete composition of flows, what if we have a continuous-time flow?

$$\begin{aligned} z_0 &= x \\ z_t &= z_0 + \int_0^t h(t, z_t)\, dt \\ f(x) &= z_1 \end{aligned}$$

Inverse:

$$\begin{aligned} z_1 &= z \\ z_t &= z_1 - \int_t^1 h(t, z_t)\, dt \\ f^{-1}(z) &= z_0 \end{aligned}$$

R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, Neural ordinary differential equations, *NeurIPS*, 2018.
W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, FFJORD: Free-form continuous dynamics for scalable reversible generative models, *ICLR*, 2019.

Math review: Conditional probabilities

Let A and B be probabilistic events. Assume A has nonzero probability.

Conditional probability satisfies

$$\mathbb{P}(B|A)\mathbb{P}(A) = \mathbb{P}(A \cap B)$$

Bayes’ theorem is an application of conditional probability:

$$\mathbb{P}(B|A) = \frac{\mathbb{P}(A|B)\mathbb{P}(B)}{\mathbb{P}(A)}$$

Math review: Conditional densities

Let $X \in \mathbb{R}^m$ and $Z \in \mathbb{R}^n$ be continuous random variables with joint density $p(x, z)$.

The marginal densities are defined by

$$p_X(x) = \int_{\mathbb{R}^n} p(x, z)\, dz\,, \qquad p_Z(z) = \int_{\mathbb{R}^m} p(x, z)\, dx$$

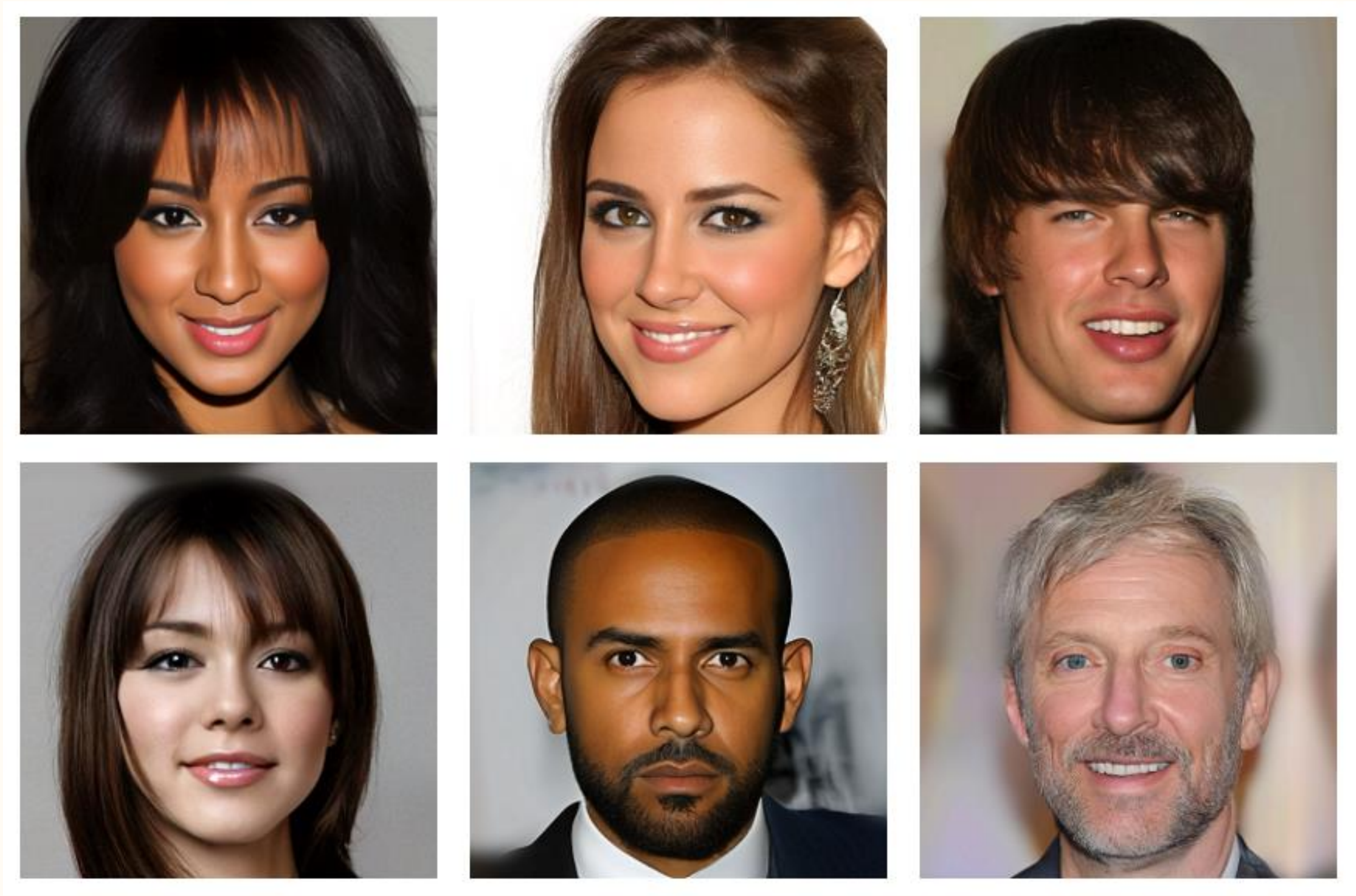
The conditional density function $p(z|x)$ has the following properties

$$\mathbb{P}(Z \in S | X = x) = \int_S p(z|x)\, dz$$

$$p(z|x)p_X(x) = p(x, z), \qquad p(z|x) = \frac{p(x|z)p_Z(Z)}{p_X(X)}$$

Variational autoencoders (VAE)

These are synthetic (fake) images.



Variational autoencoders (VAE)

Key idea of VAE:

- Latent variable model with conditional probability distribution represented by $p_\theta(x|z)$.
- Efficiently estimate $p_\theta(x) = \mathbb{E}_{Z\sim p_Z}[p_\theta(x|Z)]$ by importance sampling with $Z \sim q_\phi(z|x)$.

We can interpret $q_\phi(z|x)$ as an *encoder* and $p_\theta(x|z)$ as a *decoder*.

VAEs differ from autoencoders as follows:

- Derivations (latent variable model vs. dimensionality reduction)
- VAE regularizes/controls latent distribution, while AE does not.

Latent variable model

Assumption on data $X_1, \ldots, X_N$: Assumes there is an underlying *latent variable* Z representing the "essential structure" of the data and an *observable variable* X which generation is conditioned on Z. Implicitly assumes the conditional randomness of $X \sim p_{X|Z}$ is significantly smaller than the overall randomness $X \sim p_X$.

Example: X is a cat picture. Z encodes information about the body position, fur color, and facial expression of a cat. Latent variable Z encodes the overall content of the image, but X does contain details not specified in Z.

Specification VAE's model: VAEs implements a *latent variable model* with a NN that generates X given Z. More precisely, NN is a deterministic function that outputs the conditional distribution $p_\theta(x|Z)$, and X is randomly generated according to this distribution. This structure may effectively learn the latent structure from data if the assumption on data is accurate.

Latent variable model

$$\begin{tikzcd} Z \arrow[d, "p_\theta(x \mid z)"] \\ X \end{tikzcd}$$

Sampling process:

$$X \sim p_\theta(x \mid Z), \qquad Z \sim p_Z(z)$$

Usually p_Z is a Gaussian (fixed) and $p_\theta(x|z)$ is a NN parameterized by θ.

Evaluating density (likelihood):

$$p_\theta(X_i) = \int_z p_Z(z) p_\theta(X_i \mid z)\, dz = \mathbb{E}_{Z \sim p_Z}\left[p_\theta(X_i \mid Z)\right]$$

Training via MLE:

$$\underset{\theta \in \Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_\theta\left(X_i\right) \quad = \quad \underset{\theta \in \Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log \mathbb{E}_{Z \sim p_Z}\left[p_\theta(X_i \mid Z)\right]$$

Latent variable model

When p_Z is a discrete:

$$p_\theta(x) = \mathbb{E}_{Z\sim p_Z}[p_\theta(x \mid Z)] = \sum_z p_Z(z) p_\theta(x \mid Z)$$

When p_Z is a continuous:

$$p_\theta(x) = \mathbb{E}_{Z\sim p_Z}[p_\theta(x \mid Z)] = \int_z p_Z(z) p_\theta\left(x \mid z\right) dz$$

To clarify, specification of $p_Z(z)$ and $p_\theta(x|z)$ fully determines $p_\theta(x)$ (as above) and

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z) p_Z(z)}{p_\theta(x)}$$

Latent variable model: Training

Training

$$\underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_\theta(X_i) \;=\; \underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log \mathbb{E}_{Z\sim p_Z}\left[p_\theta(X_i \mid Z)\right]$$

requires evaluation $\mathbb{E}_Z$.

Scenario 1: If Z is discrete and takes a few of values, then compute $\sum_z$ exactly.

Scenario 2: If Z takes many values or if it is a continuous, then $\sum_z$ or $\mathbb{E}_Z$ is impractical to compute. In this case, approximate expectation with Monte Carlo and importance sampling.

Example latent variable model: Mixture of Gaussians

Mixture of 3 Gaussians in $\mathbb{R}^2$, uniform prior over components. (We can make the mixture weights a trainable parameter.)

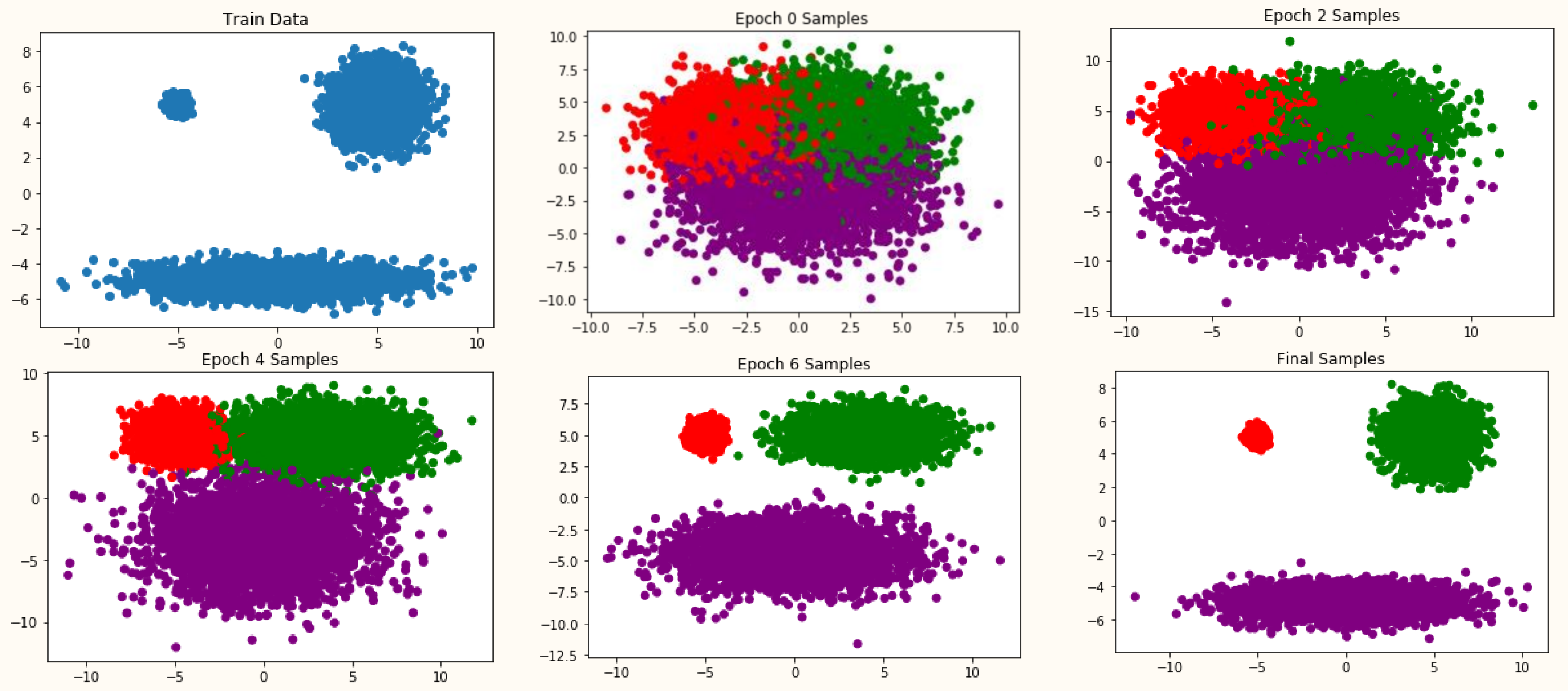
$$p_Z(Z = A) = p_Z(Z = B) = p_Z(Z = C) = \frac{1}{3}$$

$$p_\theta(x \mid Z = k) = \frac{1}{2\pi \left|\Sigma_k\right|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\left(x - \mu_k\right)^\top \Sigma_k^{-1}\left(x - \mu_k\right)\right)$$

Training objective:

$$\begin{aligned}\underset{\mu,\Sigma}{\text{maximize}} \sum_{i=1}^{N} \log p_\theta(X_i) = \underset{\mu,\Sigma}{\text{maximize}} \sum_{i=1}^{N} \log \Bigg[&\frac{1}{3}\frac{1}{2\pi \left|\Sigma_A\right|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\left(X_i - \mu_A\right)^\top \Sigma_A^{-1}\left(X_i - \mu_A\right)\right) \\ &+ \frac{1}{3}\frac{1}{2\pi \left|\Sigma_B\right|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\left(X_i - \mu_B\right)^\top \Sigma_B^{-1}\left(X_i - \mu_B\right)\right) \\ &+ \frac{1}{3}\frac{1}{2\pi \left|\Sigma_C\right|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\left(X_i - \mu_C\right)^\top \Sigma_C^{-1}\left(X_i - \mu_C\right)\right) \Bigg]\end{aligned}$$

Example: 2D mixture of Gaussians



VAE outline

Train latent variable model with MLE

$$\underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_\theta(X_i) \quad = \quad \underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log \mathbb{E}_{Z\sim p_Z}\left[p_\theta(X_i \mid Z)\right]$$

Outline of variational autoencoder (VAE):

1. Approximate intractable objective with a single Z sample

$$\sum_{i=1}^{N} \log \mathbb{E}_{Z\sim p_Z}\left[p_\theta(X_i \mid Z)\right] \approx \sum_{i=1}^{N} \log p_\theta(X_i \mid Z_i), \qquad Z_i \sim p_Z$$

2. Improve accuracy of approximation by sampling Z_i with importance sampling

$$\sum_{i=1}^{N} \log \mathbb{E}_{Z\sim p_Z}\left[p_\theta(X_i \mid Z)\right] \approx \sum_{i=1}^{N} \log \frac{p_\theta(X_i \mid Z_i)p_Z(Z_i)}{q_i(Z_i)}, \qquad Z_i \sim q_i$$

3. Optimize approximate objective with SGD.

IWAE outline

Importance weighted autoencoders (IWAE) approximates intractable with K samples of Z:

$$\sum_{i=1}^{N} \log \mathbb{E}_{Z \sim p_Z} \left[p_\theta(X_i \mid Z) \right] \approx \sum_{i=1}^{N} \log \frac{1}{K} \sum_{k=1}^{K} \frac{p_\theta(X_i \mid Z_{i,k}) p_Z(Z_{i,k})}{q_i(Z_{i,k})}, \qquad Z_{i,1}, \ldots, Z_{i,K} \sim q_i$$

More on this in hw 9.

Why does VAE need IS?

Sampling $Z_i \sim p_Z$ results in a high-variance estimator:

$$\mathbb{E}_{Z \sim p_Z}\left[p_\theta(X_i \mid Z)\right] \approx p_\theta(X_i \mid Z_i),$$

In the Gaussian mixture example, only $1/3$ of the Z samples meaningfully contribute to the estimate. More specifically, if X_i is near μ_A but is far from μ_B and μ_C, then $p_\theta(X_i|Z = A) \gg 0$ but $p_\theta(X_i|Z = B) \approx 0$ and $p_\theta(X_i|Z = C) \approx 0$.

The issue worsens as the observable and latent variable dimension increases.

Naïvely using IS for each X_i

To improve estimation of $\mathbb{E}_{Z\sim p_Z}\left[p_\theta(X_i \mid Z)\right]$, consider importance sampling (IS) with sampling distribution $Z_i \sim q_i(z)$:

$$\mathbb{E}_{Z\sim p_Z}\left[p_\theta(X_i \mid Z)\right] \approx p_\theta(X_i \mid Z_i)\frac{p_Z(Z_i)}{q_i(Z_i)}$$

Optimal IS sampling distribution

$$q_i^\star(z) = \frac{p_\theta(X_i \mid z)p_Z(z)}{p_\theta(X_i)} = p_\theta(z \mid X_i)$$

To clarify, optimal sampling distribution depends on X_i. To clarify, $p_\theta(X_i)$ is the unkown normalizing factor so $p_\theta(z|X_i)$ is also unkown. We call $q_i^\star(z) = p_\theta(z|X_i)$ the true *posterior* distribution and we will soon consider the approximation $q_\phi(z|x) \approx p_\theta(z|x)$, which we call the *approximate posterior*.

Naïvely using IS for each X_i

For each X_i, consider

$$
\begin{aligned}
&\underset{q_i}{\text{minimize}} \quad D_{\mathrm{KL}}\left(q_i(\cdot) \| p_\theta(\cdot \mid X_i)\right) \\
=\ &\underset{q_i}{\text{minimize}} \quad \mathbb{E}_{Z \sim q_i} \log\left(\frac{q_i(Z)}{p_\theta(Z \mid X_i)}\right) \\
=\ &\underset{q_i}{\text{minimize}} \quad \mathbb{E}_{Z \sim q_i} \log\left(\frac{q_i(Z)}{p_\theta(X_i \mid Z) p_Z(Z) / p_\theta(X_i)}\right) \\
=\ &\underset{q_i}{\text{minimize}} \quad \mathbb{E}_{Z \sim q_i}\left[\log q_i(Z) - \log p_Z(Z) - \log p_\theta(X_i \mid Z)\right] + \log p_\theta(X_i)
\end{aligned}
$$

Note, $q_i(z)$, $p_Z(z)$, and $p_\theta(x|z)$ are tractable/known while $p_\theta(X_i)$ and $p_\theta(z|X_i)$ are intractable/unknown. Since $\log p_\theta(X_i)$ does not depend on q_i, all quantities needed in the optimization problems are tractable. However, solving this minimization problem to obtain each q_i for each data point X_i is computationally too expensive.

Non-amortized inference

Individual inference (not amortized): For each $X_1, \dots, X_N$, find corresponding optimal $q_1, \dots, q_N$ by solving

$$\underset{q_i}{\text{minimize}} \quad D_{\mathrm{KL}}\left(q_i(\cdot) \| p_\theta\left(\cdot \mid X_i\right)\right)$$

This is expensive as it requires solving N separate optimization problems.

We need variational approach and amortized inference.

Variational approach and amortized inference

General principle of variational approach: We can't directly use the q we want. So, instead, we propose a parameterized distribution q_ϕ that we can work with easily (in this case, sample from easily), and find a parameter setting that makes it as good as possible.

Parametrization of VAE:

$$q_\phi(z \mid X_i) \approx q_i^\star(z) = p_\theta(z \mid X_i) \qquad \text{for all } i = 1, \ldots, N$$

Amortized inference: Train a neural network $q_\phi(\cdot\,|x)$ such that $q_\phi(\cdot\,|X_i)$ approximates the optimal $q_i(\cdot)$.

$$\underset{\phi\in\Phi}{\text{minimize}} \quad \sum_{i=1}^{N} D_{\text{KL}}\left(q_\phi(\cdot \mid X_i) \| p_\theta(\cdot \mid X_i)\right)$$

Approximation $q_\phi(z|X_i) \approx p_\theta(z|X_i)$ is often less precise than that of individual inference $q_i(z) \approx p_\theta(z|X_i)$, but amortized inference is often significantly faster.

Encoder q_ϕ optimization

In analogy with autoencoders, we call q_ϕ the *encoder*.

Optimization problem for encoder

$$\begin{aligned}
&\underset{\phi\in\Phi}{\text{minimize}} \quad \sum_{i=1}^{N} D_{\mathrm{KL}}\left(q_\phi(\cdot \mid X_i) \| p_\theta\left(\cdot \mid X_i\right)\right) \\
&= \quad \underset{\phi\in\Phi}{\text{maximize}} \quad \sum_{i=1}^{N} \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log\left(\frac{p_\theta(X_i \mid Z)p_Z(Z)}{q_\phi(Z \mid X_i)}\right)\right] + \text{constant independent of } \phi \\
&= \quad \underset{\phi\in\Phi}{\text{maximize}} \quad \sum_{i=1}^{N} \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log p_\theta(X_i \mid Z)\right] - D_{\mathrm{KL}}\left(q_\phi(\cdot \mid X_i) \| p_Z(\cdot)\right)
\end{aligned}$$

Decoder p_θ optimization

In analogy with autoencoders, we call p_θ the *decoder*. Perform approximate MLE with

$$\begin{aligned}
\underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_\theta(X_i) \quad &= \quad \underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log \mathbb{E}_{Z\sim p_Z}\left[p_\theta(X_i \mid Z)\right] \\
&\overset{(a)}{\approx} \quad \underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log\left(\frac{p_\theta(X_i \mid Z_i)p_Z(Z_i)}{q_\phi(Z_i \mid X_i)}\right), \qquad Z_i \sim q_\phi(z \mid X_i) \\
&\overset{(b)}{\approx} \quad \underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log\left(\frac{p_\theta(X_i \mid Z)p_Z(Z)}{q_\phi(Z \mid X_i)}\right)\right] \\
&= \quad \underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log p_\theta(X_i \mid Z)\right] - D_{\mathrm{KL}}\left(q_\phi(\cdot \mid X_i) \| p_Z(\cdot)\right)
\end{aligned}$$

The $\overset{(a)}{\approx}$ step replaces expectation inside the log with an estimate with Z_i. The $\overset{(b)}{\approx}$ step replaces the random variable with the expectation. These steps take $\mathbb{E}_Z$ outside of the log. More on this later.

VAE optimization

The optimization objectives for the encoder and decoder are the same.

Simultaneously train p_θ and q_ϕ by solving

$$\underset{\theta\in\Theta,\,\phi\in\Phi}{\text{maximize}} \quad \sum_{i=1}^{N} \underbrace{\mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log p_\theta(X_i \mid Z)\right] - D_{\mathrm{KL}}\left(q_\phi(\cdot \mid X_i)\|p_Z(\cdot)\right)}_{\overset{\mathrm{def}}{=}\mathrm{VLB}_{\theta,\phi}(X_i)}$$

We refer to the optimization objective as the *variational lower bound* (VLB) or *evidence lower bound* (ELBO) for reasons that will be explained soon.

VAE standard instance

A standard VAE setup:

$$\begin{aligned} p_Z &= \mathcal{N}(0, I) \\ q_\phi(z \mid x) &= \mathcal{N}(\mu_\phi(x), \Sigma_\phi(x)) \ \text{ with diagonal } \Sigma_\phi \\ p_\theta(x \mid z) &= \mathcal{N}\left(f_\theta(z), \sigma^2 I\right) \end{aligned}$$

Remember from hw6 that

$$D_{\mathrm{KL}}\left(\mathcal{N}(\mu_\phi(X), \Sigma_\phi(X)) \| \mathcal{N}(0, I)\right)$$
$$= \frac{1}{2}\left(\operatorname{tr}\left(\Sigma_\phi(X)\right) + \|\mu_\phi(X)\|^2 - d - \log\det(\Sigma_\phi(X))\right)$$

$\mu_\phi(x)$, $\Sigma_\phi^2(x)$, and $f_\theta(z)$ are deterministic NN. The training objective

$$\underset{\theta\in\Theta,\,\phi\in\Phi}{\text{maximize}} \quad \sum_{i=1}^{N} \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log p_\theta(X_i \mid Z)\right] - D_{\mathrm{KL}}\left(q_\phi(\cdot \mid X_i) \| p_Z(\cdot)\right)$$

becomes

$$\underset{\theta\in\Theta,\,\phi\in\Phi}{\text{minimize}} \quad \sum_{i=1}^{N} \frac{1}{\sigma^2} \mathbb{E}_{Z\sim\mathcal{N}(\mu_\phi(X_i),\Sigma_\phi(X_i))} \|X_i - f_\theta(Z)\|^2 + \operatorname{tr}\left(\Sigma_\phi(X_i)\right) + \|\mu_\phi(X_i)\|^2 - \log\det(\Sigma_\phi(X_i))$$

With reparameterization trick

The standard instance of VAE

$$\underset{\theta\in\Theta,\,\phi\in\Phi}{\text{minimize}} \quad \sum_{i=1}^{N}\frac{1}{\sigma^2}\mathbb{E}_{Z\sim\mathcal{N}(\mu_\phi(X_i),\Sigma_\phi(X_i))}\|X_i-f_\theta(Z)\|^2+\mathrm{tr}\left(\Sigma_\phi(X_i)\right)+\|\mu_\phi(X_i)\|^2-\log\det(\Sigma_\phi(X_i))$$

can be equivalently written with the reparameterization trick

$$\underset{\theta\in\Theta,\,\phi\in\Phi}{\text{minimize}} \quad \sum_{i=1}^{N}\frac{1}{\sigma^2}\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}\left\|X_i-f_\theta\left(\mu_\phi(X_i)+\Sigma_\phi^{1/2}(X_i)\varepsilon\right)\right\|^2+\mathrm{tr}\left(\Sigma_\phi(X_i)\right)+\|\mu_\phi(X_i)\|^2-\log\det(\Sigma_\phi(X_i))$$

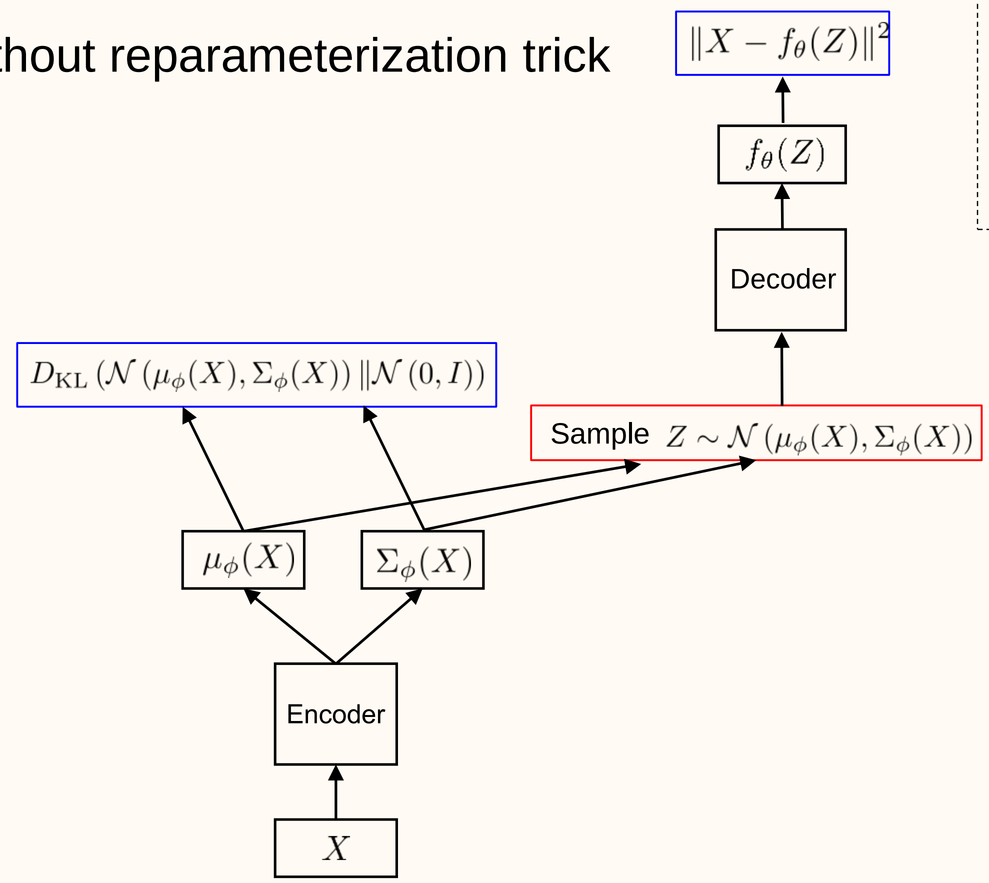
where $\Sigma_\phi^{1/2}$ is diagonal with $\sqrt{\cdot}$ of the diagonal elements of Σ_ϕ. (Remember, Σ_ϕ is diagonal.)

To clarify $Z\overset{\mathcal{D}}{=}\mu_\phi(X_i)+\Sigma_\phi^{1/2}(X_i)\varepsilon$, where $\overset{\mathcal{D}}{=}$ denotes equality in distribution.

We now have an objective amenable to stochastic optimization.

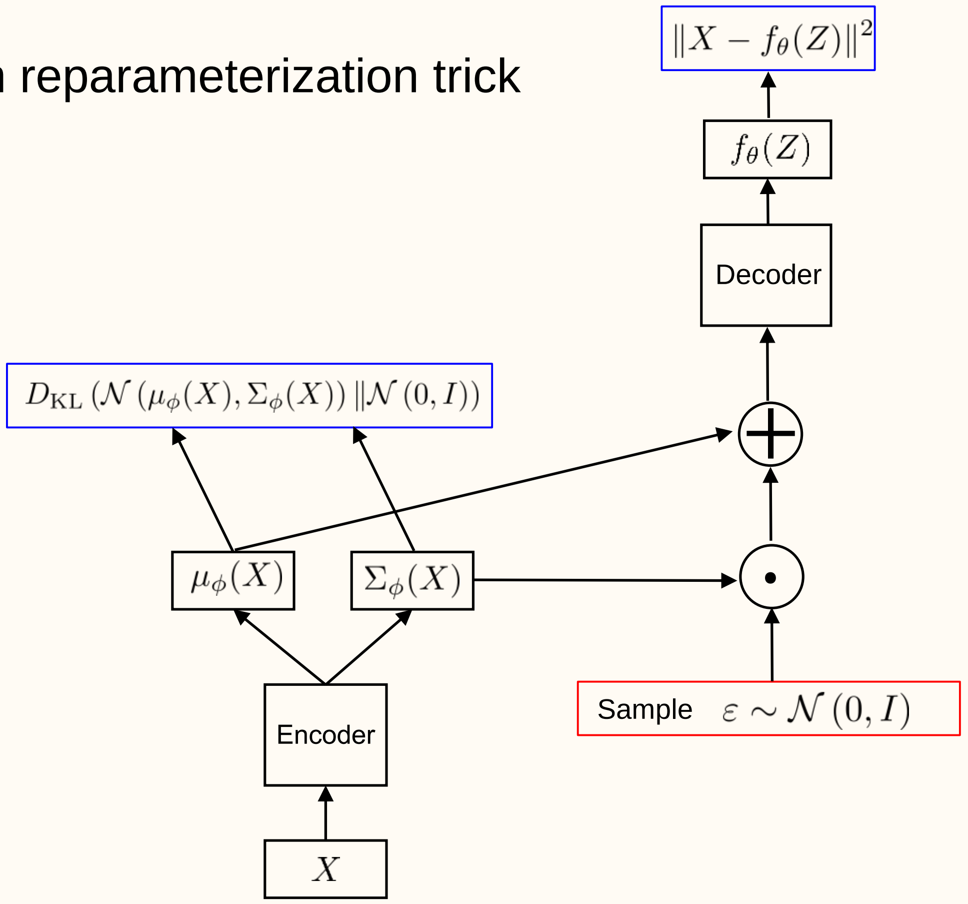
VAE standard instance architecture: Training

Without reparameterization trick



$\|X - f_\theta(Z)\|^2$
$f_\theta(Z)$
Decoder
$D_{\mathrm{KL}}\left(\mathcal{N}\left(\mu_\phi(X), \Sigma_\phi(X)\right) \| \mathcal{N}(0, I)\right)$
Sample $Z \sim \mathcal{N}\left(\mu_\phi(X), \Sigma_\phi(X)\right)$
$\mu_\phi(X)$
$\Sigma_\phi(X)$
Encoder
X

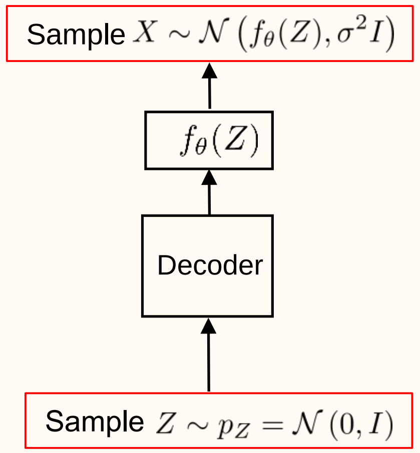
With reparameterization trick



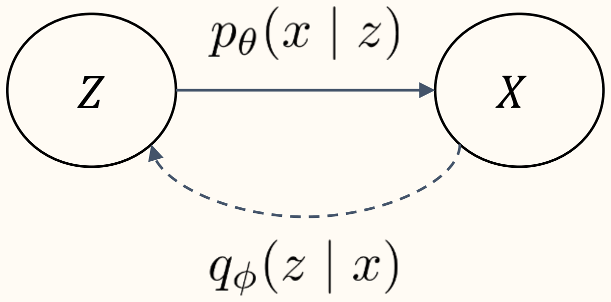
$\|X - f_\theta(Z)\|^2$
$f_\theta(Z)$
Decoder
$D_{\mathrm{KL}}\left(\mathcal{N}\left(\mu_\phi(X), \Sigma_\phi(X)\right) \| \mathcal{N}(0, I)\right)$
$\bigoplus$
$\odot$
$\mu_\phi(X)$
$\Sigma_\phi(X)$
Sample $\varepsilon \sim \mathcal{N}(0, I)$
Encoder
X

VAE standard instance architecture: Sampling

During sampling, only the decoder network is used.


Sample $X \sim \mathcal{N}\left(f_\theta(Z), \sigma^2 I\right)$
$f_\theta(Z)$
Decoder
Sample $Z \sim p_Z = \mathcal{N}(0, I)$

Discussions



Review of terminology

- Likelihood $p_\theta(x)$ (exact evaluation intractable)
- Prior $p_Z(z)$
- Conditional distribution $p_\theta(x|z)$
- True posterior $p_\theta(z|x)$ (exact evaluation intractable)
- Approximate posterior $q_\phi(z|x)$

Conditional distribution $p_\theta(x|z)$ and prior $p_Z(z)$ determines the posterior $p_\theta(z|x)$.

There is no easy way to evaluate $p_\theta(x)$, but we can sample $X \sim p_\theta(x)$ easily: $Z \sim p_Z(z)$ then $X \sim p_\theta(x|Z)$.

NN in VAE do not directly generate random output. NN outputs parameters for random sampling.

Training VAE with RT

To obtain stochastic gradients of the VAE objective

$$\underset{\theta\in\Theta,\,\phi\in\Phi}{\text{minimize}} \quad \sum_{i=1}^{N} \underbrace{\frac{1}{\sigma^2}\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}\left\|X_i - f_\theta\left(\mu_\phi(X_i)+\Sigma_\phi^{1/2}(X_i)\varepsilon\right)\right\|^2 + \operatorname{tr}\left(\Sigma_\phi(X_i)\right) + \|\mu_\phi(X_i)\|^2 - \log\det(\Sigma_\phi(X_i))}_{\overset{\text{def}}{=}-\mathrm{VLB}_{\theta,\phi}(X_i)}$$

select a data X_i, sample $\varepsilon_i \sim \mathcal{N}(0,I)$, evaluate

$$-\mathrm{VLB}_{\theta,\phi}(X_i,\varepsilon_i) \overset{\text{def}}{=} \frac{1}{\sigma^2}\left\|X_i - f_\theta\left(\mu_\phi(X_i)+\Sigma_\phi^{1/2}(X_i)\varepsilon_i\right)\right\|^2 + \operatorname{tr}\left(\Sigma_\phi(X_i)\right) + \|\mu_\phi(X_i)\|^2 - \log\det(\Sigma_\phi(X_i))$$

and backprop on $\mathrm{VLB}_{\theta,\phi}(X_i,\varepsilon_i)$.

Usually, batch of X_i is selected.
One can sample multiple $Z_{i,1},\dots,Z_{i,K}$ (equivalently $\varepsilon_{i,1},\dots,\varepsilon_{i,K}$) for each X_i.

Training VAE with log-derivative trick

Computing stochastic gradients without the reparameterization trick.

$$\underset{\theta\in\Theta,\,\phi\in\Phi}{\text{maximize}} \quad \sum_{i=1}^{N} \underbrace{\mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log\left(\frac{p_\theta(X_i \mid Z)p_Z(Z)}{q_\phi(Z \mid X_i)}\right)\right]}_{\overset{\text{def}}{=}\text{VLB}_{\theta,\phi}(X_i)}$$

To obtain unbiased estimates of ∇_θ, compute

$$\frac{1}{K}\sum_{k=1}^{K}\log p_\theta(X_i \mid Z_{i,k}), \qquad Z_{i,1},\ldots,Z_{i,K} \sim q_\phi(z \mid X_i)$$

and backprop with respect to θ.

Training VAE with log-derivative trick

We differentiate the VLB objectives (cf. hw 8 problem 8)

$$\begin{aligned}\nabla_\phi \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log\left(\frac{p_\theta(X_i \mid Z)p_Z(Z)}{q_\phi(Z \mid X_i)}\right)\right] &= \nabla_\phi \int \log\left(\frac{p_\theta(X_i \mid z)p_Z(z)}{q_\phi(z \mid X_i)}\right) q_\phi(z \mid X_i)\, dz \\ &= \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[(\nabla_\phi \log q_\phi(Z \mid X_i))\log\left(\frac{p_\theta(X_i \mid Z)p_Z(Z)}{q_\phi(Z \mid X_i)}\right)\right]\end{aligned}$$

To obtain unbiased estimates of ∇_ϕ, compute

$$\frac{1}{K}\sum_{k=1}^{K}(\nabla_\phi \log q_\phi(Z_{i,k} \mid X_i))\log\left(\frac{p_\theta(X_i \mid Z_{i,k})p_Z(Z_{i,k})}{q_\phi(Z_{i,k} \mid X_i)}\right), \qquad Z_{i,1},\dots,Z_{i,K} \sim q_\phi(z \mid X_i)$$

Why variational “autoencoder”?

VAE loss (VLB) contains a reconstruction loss resembling that of an autoencoder.

$$\begin{aligned}
\mathrm{VLB}_{\theta,\phi}(X_i) &= \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log p_\theta(X_i \mid Z)\right] - D_{\mathrm{KL}}\left(q_\phi(\cdot \mid X_i)\|p_Z(\cdot)\right) \\
&= -\frac{1}{2\sigma^2}\mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\|X_i - f_\theta(Z)\|^2\right] - D_{\mathrm{KL}}\left(q_\phi(\cdot \mid X_i)\|p_Z(\cdot)\right) \\
&= \underbrace{-\frac{1}{2\sigma^2}\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}\left\|X_i - f_\theta\left(\mu_\phi(X_i) + \Sigma_\phi^{1/2}(X_i)\varepsilon\right)\right\|^2}_{\text{Reconstruction loss}} - \underbrace{D_{\mathrm{KL}}\left(q_\phi(\cdot \mid X_i)\|p_Z(\cdot)\right)}_{\text{Regularization}}
\end{aligned}$$

VLB also contains a regularization term on the output of the encoder, which is not present in standard autoencoder losses.

The choice of σ determines the relative weight between the reconstruction loss and the regularization.

How tight is the VLB?

How accurate is the approximation?

$$\begin{aligned}
\underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_\theta(X_i) \quad &= \quad \underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log \mathbb{E}_{Z\sim q_\phi(z|X_i)} \left[\frac{p_\theta(X_i \mid Z) p_Z(Z)}{q_\phi(Z \mid X_i)} \right] \\
&\overset{?}{\approx} \quad \underset{\theta\in\Theta,\phi\in\Phi}{\text{maximize}} \quad \sum_{i=1}^{N} \mathbb{E}_{Z\sim q_\phi(z|X_i)} \left[\log \left(\frac{p_\theta(X_i \mid Z) p_Z(Z)}{q_\phi(Z \mid X_i)} \right) \right] \\
&= \quad \underset{\theta\in\Theta,\phi\in\Phi}{\text{maximize}} \quad \sum_{i=1}^{N} \text{VLB}_{\theta,\phi}(X_i)
\end{aligned}$$

This turns out that $\log p_\theta(X_i) \geq \text{VLB}_{\theta,\phi}(X_i)$. So we are maximizing a lower bound of the log likelihood. How large is the gap?

Log-likelihood $\geq$ VLB: Derivation 1

Derivation via Jensen:

$$\begin{aligned}
\log p_\theta(X_i) &= \log \mathbb{E}_{Z\sim p_Z}\left[p_\theta(X_i \mid Z)\right] \\
&= \log\left(\mathbb{E}_{Z\sim q_\phi(Z|X_i)}\left[p_\theta(X_i \mid Z)\frac{p_Z(Z)}{q_\phi(Z \mid X_i)}\right]\right) \\
&\geq \mathbb{E}_{Z\sim q_\phi(Z|X_i)}\left[\log\left(p_\theta(X_i \mid Z)\frac{p_Z(Z)}{q_\phi(Z \mid X_i)}\right)\right] \\
&\overset{\text{def}}{=} \mathrm{VLB}_{\theta,\phi}(X_i)
\end{aligned}$$

Does not explicitly characterize gap.

Log-likelihood ≥ VLB: Derivation 2

Derivation via KL divergence:

$$\begin{aligned} D_{\mathrm{KL}}\left[q_\phi(\cdot \mid X_i) \| p_\theta(\cdot \mid X_i)\right] &= \mathbb{E}_{Z \sim q_\theta(z \mid X_i)}\left[\log q_\theta(Z \mid X_i) - \log p_\theta(Z \mid X_i)\right] \\ &= \underbrace{\mathbb{E}_{Z \sim q_\theta(z \mid X_i)}\left[\log q_\theta(Z \mid X_i) - \log p_Z(Z) - \log p_\theta(X_i \mid Z)\right]}_{= -\mathrm{VLB}_{\theta,\phi}(X_i)} + \log p_\theta(X_i) \end{aligned}$$

and

$$\begin{aligned} \log p_\theta(X_i) &= \mathrm{VLB}_{\theta,\phi}(X_i) + D_{\mathrm{KL}}\left[q_\phi(\cdot \mid X_i) \| p_\theta(\cdot \mid X_i)\right] \\ &\geq \mathrm{VLB}_{\theta,\phi}(X_i) \end{aligned}$$

This derivation explicitly characterizes the gap as $D_{\mathrm{KL}}\left[q_\phi(\cdot \mid X_i) \| p_\theta(\cdot \mid X_i)\right]$.

VLB is tight if encoder infinitely powerful

If the encoder q_ϕ is powerful enough such that there is a $\phi^\star$ achieving

$$q_{\phi^\star}(\cdot \mid X_i) = p_\theta(\cdot \mid X_i)$$

or equivalently

$$D_{\mathrm{KL}}\left[q_{\phi^\star}(\cdot \mid X_i) \| p_\theta(\cdot \mid X_i)\right] = 0$$

Then

$$\underset{\theta\in\Theta}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_\theta(X_i) \quad = \quad \underset{\theta\in\Theta,\phi\in\Phi}{\text{maximize}} \quad \sum_{i=1}^{N} \mathrm{VLB}_{\theta,\phi}(X_i)$$

This follows from

$$\log p_\theta(X_i) = \mathrm{VLB}_{\theta,\phi}(X_i) + \underbrace{D_{\mathrm{KL}}\left[q_\phi(\cdot \mid X_i) \| p_\theta(\cdot \mid X_i)\right]}_{\geq 0}$$

and hw 8 problem 2.

VQ-VAE

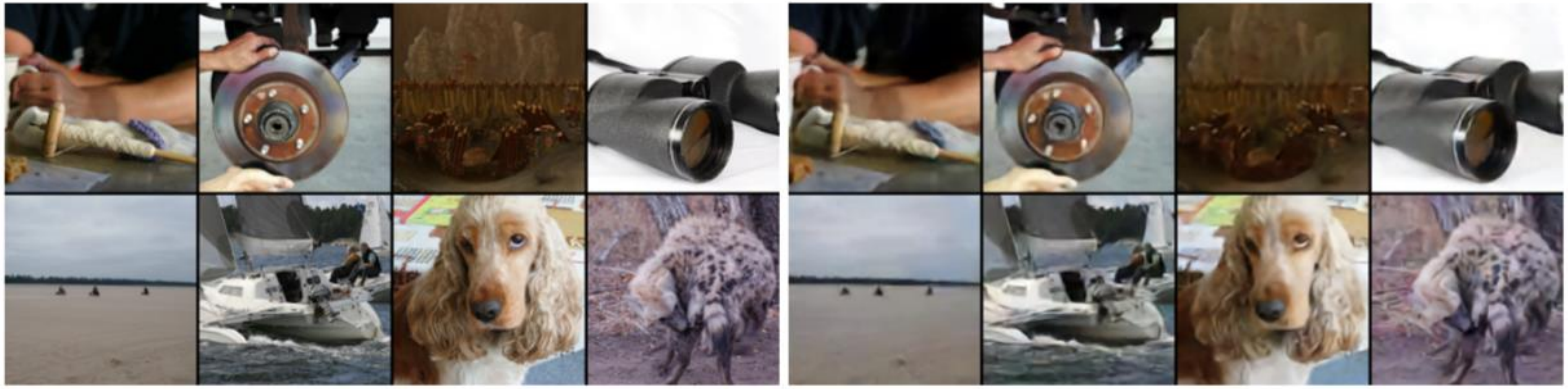


Figure 2: Left: ImageNet 128x128x3 images, right: reconstructions from a VQ-VAE with a 32x32x1 latent space, with K=512.

VQ-VAE

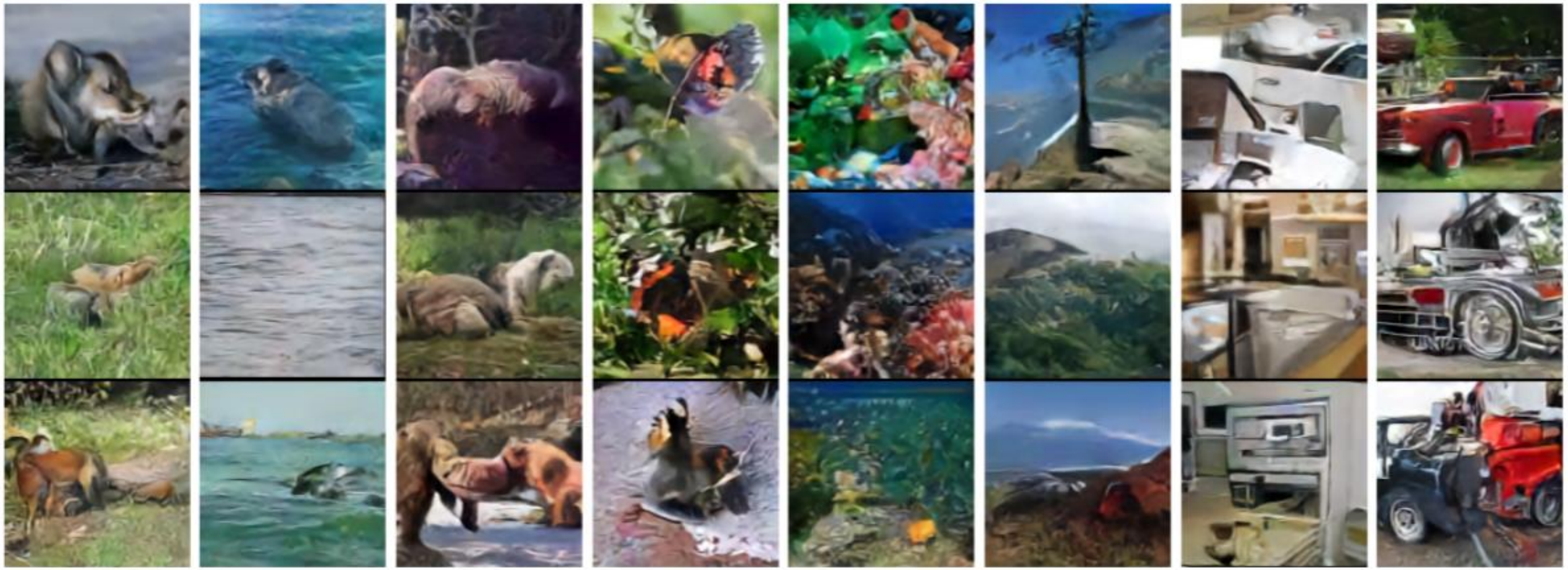
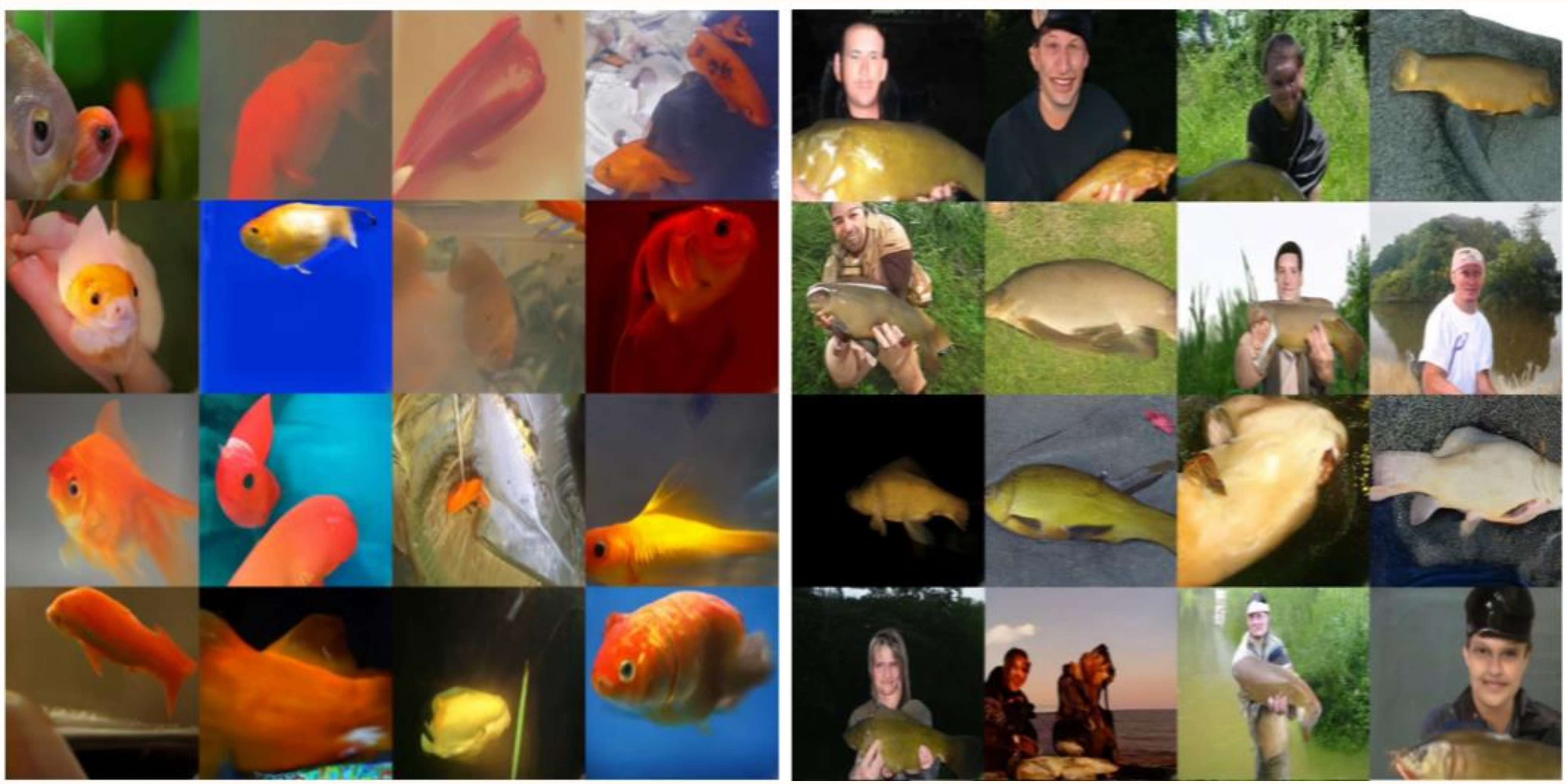


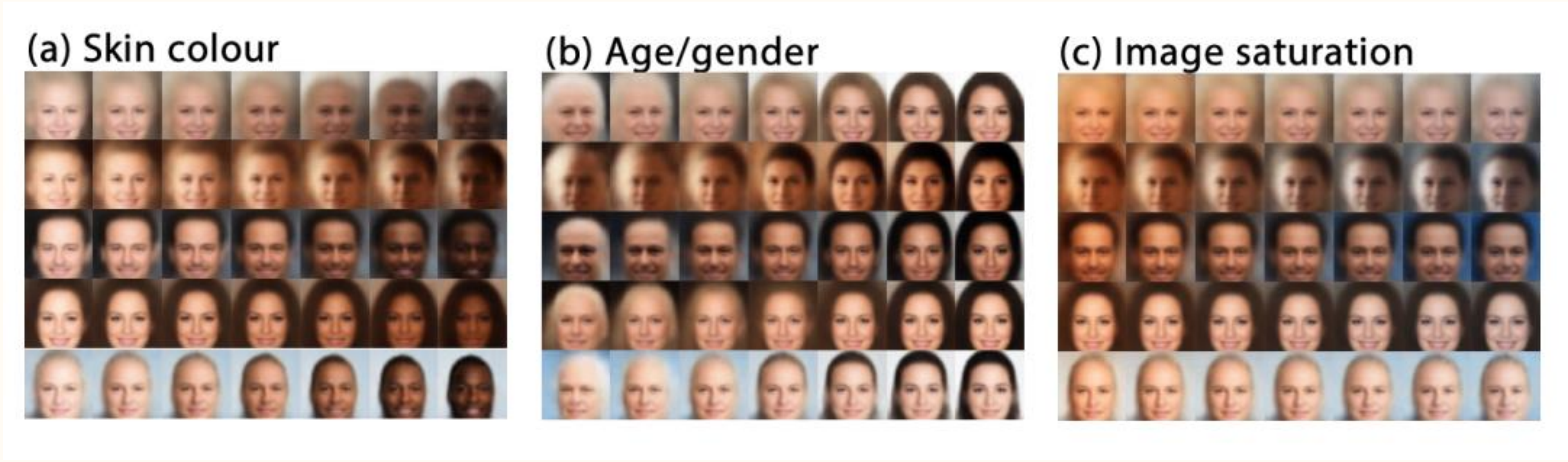
Figure 3: Samples (128x128) from a VQ-VAE with a PixelCNN prior trained on ImageNet images.

VQ-VAE-2

VQ-VAE-2



β-VAE



Uses the loss

$$\ell_{\theta,\phi}(X_i) = \mathbb{E}_{Z \sim q_\phi(z \mid X_i)}\left[\log p_\theta(X_i \mid Z)\right] - \beta D_{\mathrm{KL}}\left(q_\phi(\cdot \mid X_i) \| p_Z(\cdot)\right)$$

when $\beta = 1$, $\ell_{\theta,\phi}(X_i) = \mathrm{VLB}_{\theta,\phi}(X_i)$, i.e., β-VAE coincides with VAE when $\beta = 1$.

With $\beta > 1$, authors observed better feature disentanglement.

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, β-VAE: Learning basic visual concepts with a constrained variational framework, *ICLR*, 2017.

Minimax optimization

In a *minimax optimization problem* we minimize with respect to one variable and maximize with respect to another:

$$\underset{\theta\in\Theta}{\text{minimize}} \quad \underset{\phi\in\Phi}{\text{maximize}} \quad \mathcal{L}(\theta,\phi)$$

We say $(\theta^\star,\phi^\star)$ is a *solution*[*] to the minimax problem if $\theta^\star \in \Theta$, $\phi^\star \in \Phi$, and

$$\mathcal{L}(\theta^\star,\phi) \le \mathcal{L}(\theta^\star,\phi^\star) \le \mathcal{L}(\theta,\phi^\star), \qquad \forall\,\theta\in\Theta,\ \phi\in\Phi.$$

In other words, unilaterally deviating from $\theta^\star \in \Theta$ increases the value of $\mathcal{L}(\theta,\phi)$ while unilaterally deviating from $\phi^\star \in \Phi$ decreases the value of $\mathcal{L}(\theta,\phi)$. In yet other words, the solution is defined as a Nash equilibrium in a 2-player zero-sum game.

[*]There are other broader definitions of a “solution” in minimax optimization problems. Our definition is, in a sense, the strictest definition.

Minimax optimization

So far, we trained NN by solving minimization problems.

However, GANs are trained by solving minimax problems. Since the advent of GANs, minimax training has become more widely used in all areas of deep learning.

Examples:

- Adversarial training to make NN robust against adversarial attacks.
- Domain adversarial networks to train NN to make fair decisions (e.g. not base its decision on a persons race or gender).

Minimax vs. maximin

When a solution (as we defined it) does not exist, then min-max is not the same as max-min:

$$\underset{\theta\in\Theta}{\text{minimize}} \quad \underset{\phi\in\Phi}{\text{maximize}} \quad \mathcal{L}(\theta,\phi) \neq \underset{\phi\in\Phi}{\text{maximize}} \quad \underset{\theta\in\Theta}{\text{minimize}} \quad \mathcal{L}(\theta,\phi)$$

This is a technical distinction that we will not explore in this class.

Minimax optimization algorithm

First, consider deterministic gradient setup. Let α and β be the stepsizes (learning rates) for the descent and ascent steps respectively.

Simultaneous gradient ascent-descent:

$$\phi^{k+1} = \phi^k + \beta \nabla_\phi \mathcal{L}(\theta^k, \phi^k)$$
$$\theta^{k+1} = \theta^k - \alpha \nabla_\theta \mathcal{L}(\theta^k, \phi^k)$$

Alternating gradient ascent-descent:

$$\phi^{k+1} = \phi^k + \beta \nabla_\phi \mathcal{L}(\theta^k, \phi^k)$$
$$\theta^{k+1} = \theta^k - \alpha \nabla_\theta \mathcal{L}(\theta^k, \phi^{k+1})$$

Minimax optimization algorithm

Gradient multi-ascent-single-descent:

$$\begin{aligned}
\phi_0^{k+1} &= \phi_{n_{\text{dis}}}^k \\
\phi_{i+1}^{k+1} &= \phi_i^{k+1} + \beta \nabla_\phi \mathcal{L}(\theta^k, \phi_i^{k+1}), \qquad \text{for } i = 0, \ldots, n_{\text{dis}} - 1 \\
\theta^{k+1} &= \theta^k - \alpha \nabla_\theta \mathcal{L}(\theta^k, \phi_{n_{\text{dis}}}^{k+1})
\end{aligned}$$

(n_{dis} stands for number of discriminator updates.) When $n_{\text{dis}} = 1$, this algorithm reduces to alternating ascent-descent.

Stochastic minimax optimization

In deep learning, however, we have access to stochastic gradients.

Stochastic gradient simultaneous ascent-descent

$$\begin{aligned} \phi^{k+1} &= \phi^k + \beta g_\phi^k, \qquad & \mathbb{E}[g_\phi^k] &= \nabla_\phi \mathcal{L}(\theta^k, \phi^k) \\ \theta^{k+1} &= \theta^k - \alpha g_\theta^k, \qquad & \mathbb{E}[g_\theta^k] &= \nabla_\theta \mathcal{L}(\theta^k, \phi^k) \end{aligned}$$

Stochastic gradient alternating ascent-descent

$$\begin{aligned} \phi^{k+1} &= \phi^k + \beta g_\phi^k, \qquad & \mathbb{E}[g_\phi^k] &= \nabla_\phi \mathcal{L}(\theta^k, \phi^k) \\ \theta^{k+1} &= \theta^k - \alpha g_\theta^k, \qquad & \mathbb{E}[g_\theta^k] &= \nabla_\theta \mathcal{L}(\theta^k, \phi^{k+1}) \end{aligned}$$

Stochastic gradient multi-ascent-single-descent

$$\begin{aligned} \phi_0^{k+1} &= \phi_{n_{\text{dis}}}^k \\ \phi_{i+1}^{k+1} &= \phi_i^{k+1} + \beta \nabla_\phi g_\phi^{k,i}, \qquad & \mathbb{E}[g_\phi^{k,i}] &= \nabla_\phi \mathcal{L}(\theta^k, \phi_i^{k+1}), \qquad \text{for } i = 0, \ldots, n_{\text{dis}} - 1 \\ \theta^{k+1} &= \theta^k - \alpha g_\theta^k, \qquad & \mathbb{E}[g_\theta^k] &= \nabla_\theta \mathcal{L}(\theta^k, \phi_{n_{\text{dis}}}^{k+1}) \end{aligned}$$

Minimax optimization in PyTorch

To perform minimax optimization in PyTorch, we maintain two separate optimizers, one for the ascent, one for the descent. The `OPTIMIZER` can be anything like `SGD` or `Adam`.

```
G = Generator(...).to(device)
D = Discriminator(...).to(device)

D_optimizer = optim.OPTIMIZER(D.parameters(), lr = beta)
G_optimizer = optim.OPTIMIZER(G.parameters(), lr = alpha)
```

Simultaneous ascent-descent:

```
Evaluate D_loss
D_loss.backward()
Evaluate G_loss
G_loss.backward()

D_optimizer.step()
G_optimizer.step()
```

Minimax optimization in PyTorch

Alternating ascent-descent

```
Evaluate D_loss
D_loss.backward()
D_optimizer.step()

Evaluate G_loss
G_loss.backward()
G_optimizer.step()
```

Minimax optimization in PyTorch

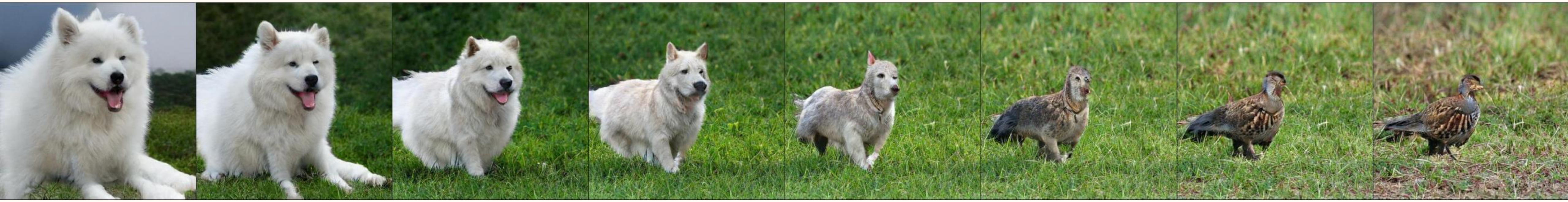
Multi-ascent-single-descent

```
for _ in range(ndis) :
  Evaluate D_loss
  D_loss.backward()
  D_optimizer.step()

Evaluate G_loss
G_loss.backward()
G_optimizer.step()
```

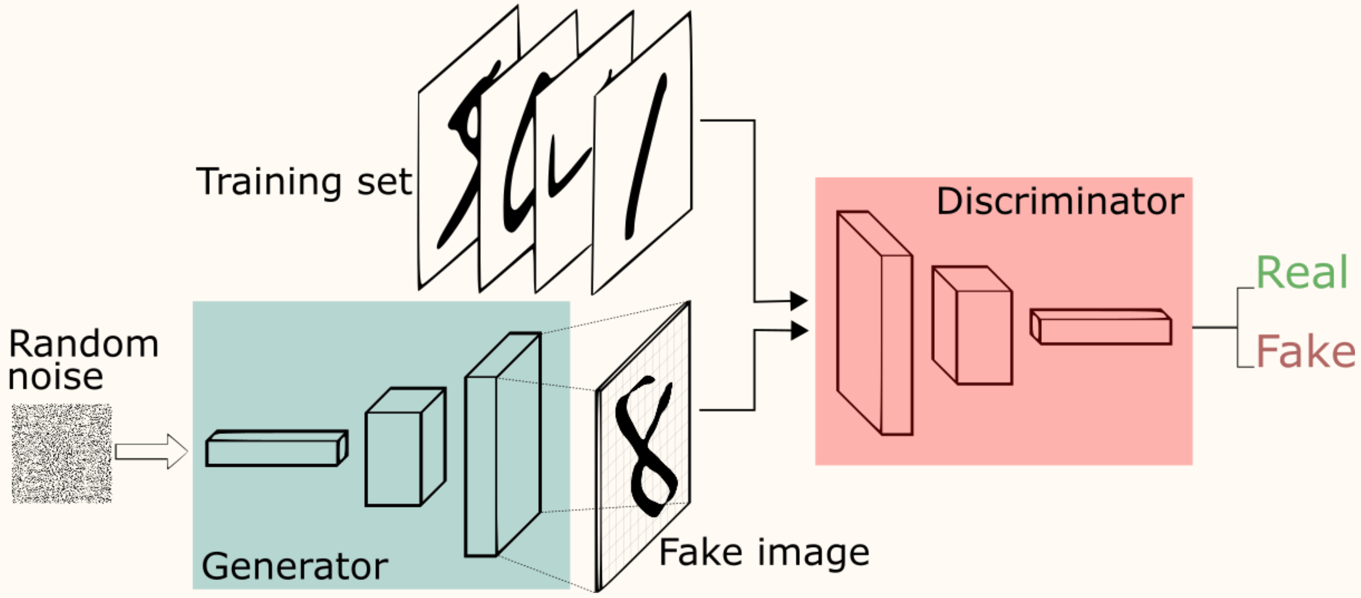
Generative adversarial networks (GAN)

These are synthetic (fake) images.



GAN

In *generative adversarial networks* (GAN) a generator network and a discriminator network compete adversarially.



Given data $X_1, \ldots, X_N \sim p_{\text{true}}$. GAN aims to learn $p_\theta \approx p_{\text{true}}$.

Generator aims to generate fake data similar to training data.

Discriminator aims to distinguish the training data from fake data.

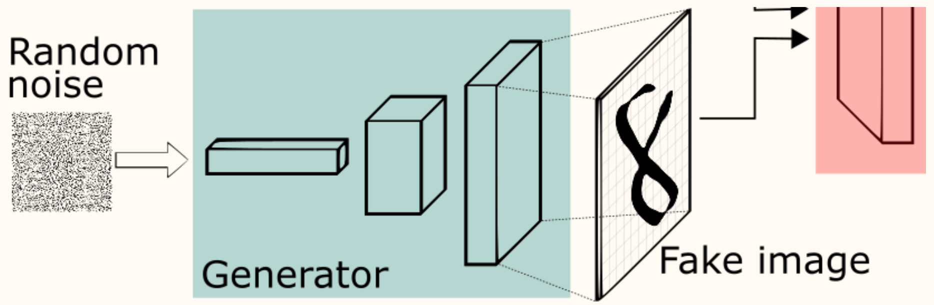
Analogy: Criminal creating fake money vs. police distinguishing fake money from real.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative adversarial networks, *NeurIPS*, 2014.

Generator network

The *generator* $G_\theta : \mathbb{R}^k \to \mathbb{R}^n$ is a neural network parameterized by $\theta \in \Theta$. The generator takes a random latent vector $Z \sim p_Z$ as input and outputs generated (fake) data $\tilde{X} = G_\theta(Z)$. The latent distribution is usually $p_Z = \mathcal{N}(0, I)$.

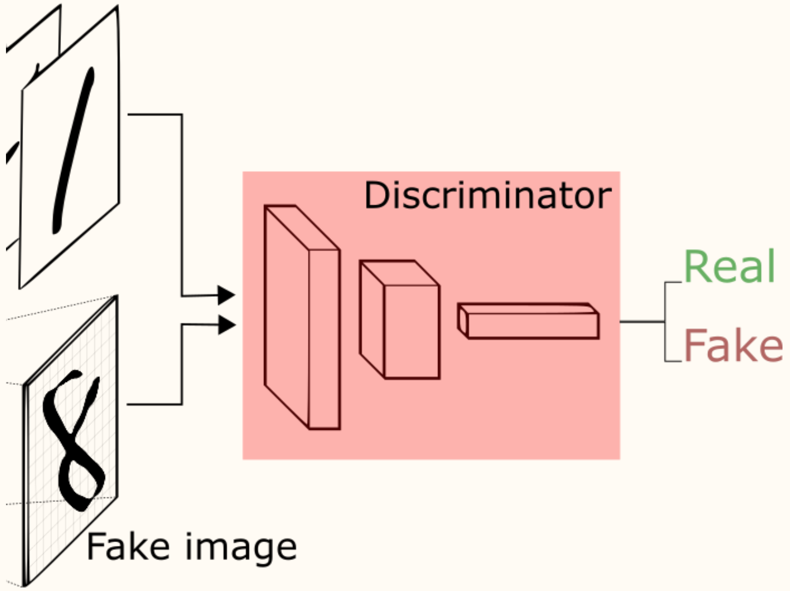
Write p_θ for the probability distribution of $\tilde{X} = G_\theta(Z)$. Although we can't evaluate the density $p_\theta(x)$, neither exactly nor approximately, we can sample from $\tilde{X} \sim p_\theta$.



Discriminator network

The *discriminator* $D_\phi : \mathbb{R}^n \to (0,1)$ is a neural network parameterized by $\phi \in \Phi$. The discriminator takes an image X as input and outputs whether X is a real or fake.[#]

- $D_\phi(X) \approx 1$: discriminator confidently predicts X is real.
- $D_\phi(X) \approx 0$: discriminator confidently predicts X is fake.
- $D_\phi(X) \approx 0.5$: discriminator is unsure whether X is real or fake.



[#]Real: X comes from a data set, i.e., $X \sim p_{\text{true}}$. Fake: generated by G_θ, i.e., $X \sim p_\theta$.

Discriminator loss

Cost of incorrectly classifying real as fake (type I error):

$$\mathbb{E}_{X\sim p_{\mathrm{true}}}\left[-\log D_\phi(X)\right]$$

Cost of incorrectly classifying fake as real (type II error):

$$\mathbb{E}_{\tilde{X}\sim p_\theta}\left[-\log(1-D_\phi(\tilde{X}))\right] = \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[-\log(1-D_\phi(G_\theta(Z)))\right]$$

Discriminator solves

$$\underset{\phi\in\Phi}{\text{maximize}} \quad \mathbb{E}_{X\sim p_{\mathrm{true}}}\left[\log D_\phi(X)\right] + \mathbb{E}_{\tilde{X}\sim p_\theta}\left[\log(1-D_\phi(\tilde{X}))\right]$$

which is equivalent to

$$\underset{\phi\in\Phi}{\text{maximize}} \quad \mathbb{E}_{X\sim p_{\mathrm{true}}}\left[\log D_\phi(X)\right] + \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[\log(1-D_\phi(G_\theta(Z)))\right]$$

Discriminator loss

We can view

$$\mathbb{E}_{\tilde{X}\sim p_\theta}\left[\log(1-D_\phi(\tilde{X}))\right] = \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[\log(1-D_\phi(G_\theta(Z)))\right]$$

as an instance of the reparameterization technique.

The loss

$$\mathbb{E}_{X\sim p_{\text{true}}}\left[\log D_\phi(X)\right] + \mathbb{E}_{\tilde{X}\sim p_\theta}\left[\log(1-D_\phi(\tilde{X}))\right]$$

puts equal weight on type I and type II errors. Alternatively, one can use the loss

$$\mathbb{E}_{X\sim p_{\text{true}}}\left[\log D_\phi(X)\right] + \lambda\mathbb{E}_{\tilde{X}\sim p_\theta}\left[\log(1-D_\phi(\tilde{X}))\right]$$

where $\lambda > 0$ represents the relative significance of a type II error over a type I error.

Generator loss

Since the goal of the generator is to deceive the discriminator, the generator minimizes the same loss.

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \mathbb{E}_{X \sim p_{\text{true}}} [\log D_\phi(X)] + \mathbb{E}_{Z \sim \mathcal{N}(0,I)} [\log(1 - D_\phi(G_\theta(Z)))]$$

(The generator and discriminator operate under a zero-sum game.)

Note, only the second term depend on θ, while the both terms depend on ϕ.

Empirical risk minimization

In practice, we have finite samples $X_1, \dots, X_N$, so we instead use the loss

$$\frac{1}{N}\sum_{i=1}^{N} \log D_\phi(X_i) + \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[\log(1 - D_\phi(G_\theta(Z)))\right]$$

Since $\tilde{X} = G_\theta(Z)$ is generated with $Z \sim p_Z$, we have unlimited $\tilde{X}$ samples. So we replace $\mathbb{E}_X \approx \frac{1}{N}\sum$ while leaving $\mathbb{E}_Z$ as is.

Minimax training (zero-sum game)

Train generator and discriminator simultaneously by solving

$$\underset{\theta\in\Theta}{\text{minimize}} \quad \underset{\phi\in\Phi}{\text{maximize}} \quad \mathcal{L}(\theta,\phi)$$

where

$$\mathcal{L}(\theta,\phi) = \frac{1}{N}\sum_{i=1}^{N}\log D_\phi(X_i) + \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[\log(1-D_\phi(G_\theta(Z)))\right]$$

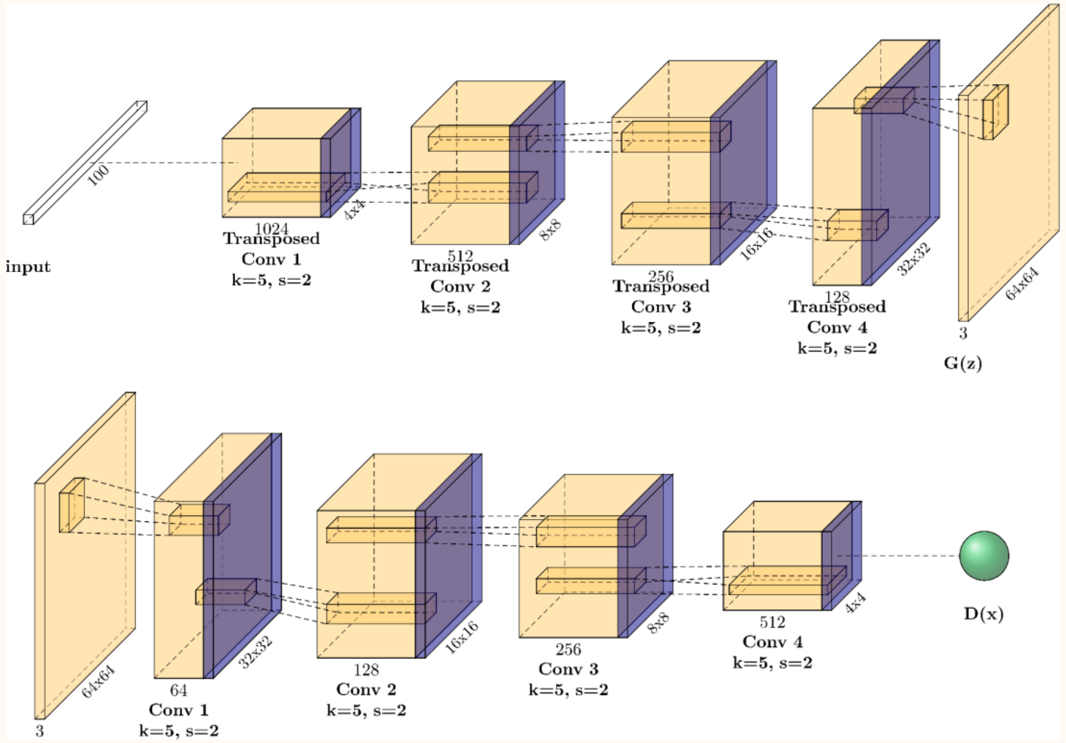
It remains to specify the architectures for G_θ and D_ϕ.

GAN demo

PyTorch demo

DCGAN

The original GAN was also deep and convolutional. However, Radford et al.'s Deep Convolutional Generative Adversarial Networks (DCGAN) paper proposed the following architectures, which crucially utilize batchnorm.



Use batchnorm in both the generator and the discriminator after transposed conv and conv layers.

Math review: f-divergence

The f-divergence of p from q , where f is a convex function such that $f(1) = 0$, is

$$D_f(p\|q) = \int f\left(\frac{p(x)}{q(x)}\right) q(x)\, dx,$$

This includes the KL divergence:

- If $f(u) = u \log u$, then $D_f(p\|q) = D_{\mathrm{KL}}(p\|q)$.
- If $f(u) = -\log u$, then $D_f(p\|q) = D_{\mathrm{KL}}(q\|p)$.

Math review: JS-divergence

Jensen–Shannon-divergence (JS-divergence) is

$$D_{\mathrm{JS}}(p,q) = \frac{1}{2} D_{\mathrm{KL}}\left(p \| \tfrac{1}{2}(p+q)\right) + \frac{1}{2} D_{\mathrm{KL}}\left(q \| \tfrac{1}{2}(p+q)\right)$$

With, $f(u) = \begin{cases} \frac{1}{2}(u \log u - (u+1)\log \frac{u+1}{2}) & \text{for } u \geq 0 \\ \infty & \text{otherwise} \end{cases}$ we have $D_f = D_{\mathrm{JS}}$.

With, $f(u) = \begin{cases} u \log u - (u+1)\log(u+1) + \log 4 & \text{for } u \geq 0 \\ \infty & \text{otherwise} \end{cases}$ we have $D_f = 2D_{\mathrm{JS}}$.

GAN ≈ JSD minimization

Let us understand the minimax problem

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \underset{\phi \in \Phi}{\text{maximize}} \quad \mathcal{L}(\theta, \phi)$$

via the minimization problem

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \mathcal{J}(\theta)$$

where

$$\mathcal{J}(\theta) = \sup_{\phi \in \Phi} \mathcal{L}(\theta, \phi)$$

For simplicity, assume the discriminator is infinitely powerful, i.e., $D_\phi(x)$ can represent any arbitrary function.

GAN ≈ JSD minimization

Note

$$\begin{aligned}\mathcal{L}(\theta,\phi) &= \mathbb{E}_{X\sim p_{\text{true}}}\left[\log D_\phi(X)\right] + \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[\log(1-D_\phi(G_\theta(Z)))\right]\\ &= \mathbb{E}_{X\sim p_{\text{true}}}\left[\log D_\phi(X)\right] + \mathbb{E}_{\tilde{X}\sim p_\theta}\left[\log(1-D_\phi(\tilde{X}))\right]\\ &= \int p_{\text{true}}(x)\log D_\phi(x) + p_\theta(x)\log(1-D_\phi(x))\,dx\end{aligned}$$

Since

$$\frac{d}{dy}(a\log y + b\log(1-y)) = 0 \quad \Rightarrow \quad y^\star = \frac{a}{a+b}$$

The integral is maximized by

$$D_{\phi^\star}(x) = \frac{p_{\text{true}}(x)}{p_{\text{true}}(x)+p_\theta(x)}$$

GAN ≈ JSD minimization

If we plug in the optimal discriminator,

$$D_{\phi^\star}(x) = \frac{p_{\text{true}}(x)}{p_{\text{true}}(x) + p_\theta(x)}$$

we get

$$\begin{aligned}\mathcal{L}(\theta, \phi^\star) &= \mathbb{E}_{X \sim p_{\text{true}}}\left[\log \frac{p_{\text{true}}(X)}{p_{\text{true}}(X) + p_\theta(X)}\right] + \mathbb{E}_{\tilde{X} \sim p_\theta}\left[\log \frac{p_\theta(\tilde{X})}{p_{\text{true}}(\tilde{X}) + p_\theta(\tilde{X})}\right] \\ &= 2D_{\text{JS}}(p_{\text{true}}, p_\theta) - \log(4)\end{aligned}$$

Therefore,

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \underset{\phi \in \Phi}{\text{maximize}} \quad \mathcal{L}(\theta, \phi) \approx \underset{\theta \in \Theta}{\text{minimize}} \quad D_{\text{JS}}(p_{\text{true}}, p_\theta)$$

f-GAN

With GANs, we started from a minimax formulation and later reinterpreted it as minimizing the JS-divergence.

Let us instead the start from an f-divergence minimization

$$\underset{\theta \in \Theta}{\text{minimize}} \quad D_f(p_{\text{true}} \| p_\theta)$$

and then variationally approximate D_f to obtain a minimax formulation.

Variational approach: Evaluating D_f directly is difficult, so we pose it as a maximization problem and parameterize the maximizing function as a “discriminator” neural network.

f-GAN

For simplicity, however, we only consider the order

$$\underset{\theta\in\Theta}{\text{minimize}} \quad D_f(p_{\text{true}} \| p_\theta)$$

However, one can also consider

$$\underset{\theta\in\Theta}{\text{minimize}} \quad D_f(p_\theta \| p_{\text{true}})$$

to obtain similar results.

(During our coverage of f-GANs, we will have notational conflict between D_f, the f-divergence, and D_ϕ, the discriminator network. Hopefully there won't be any confusion.)

Convex conjugate

Let $f : \mathbb{R} \to \mathbb{R} \cup \{\infty\}$. Define the *convex conjugate* of f as

$$f^*(t) = \sup_{u \in \mathbb{R}} \{tu - f(u)\}$$

where $f^* : \mathbb{R} \to \mathbb{R} \cup \{\infty\}$. This is also referred to as the Legendre transform.

If f is a nice# convex function, then f^* is convex and $f^{**} = f$, i.e., the conjugate of the conjugate is the original function.% So

$$f(u) = \sup_{t \in \mathbb{R}} \{tu - f^*(t)\}$$

#Closed and proper.

%So conjugacy is an involution in the space of convex functions.

Convex conjugate: Examples

The following are some examples. Computation of f^* uses basic calculus.

$$f_{\mathrm{KL}}(u) = \begin{cases} u \log u & \text{for } u \geq 0 \\ \infty & \text{otherwise} \end{cases} \qquad f^*_{\mathrm{KL}}(t) = \exp(t-1)$$

$$f_{\mathrm{LK}}(u) = \begin{cases} -\log u & \text{for } u > 0 \\ \infty & \text{otherwise} \end{cases} \qquad f^*_{\mathrm{LK}}(t) = \begin{cases} -1 - \log(-t) & \text{for } t < 0 \\ \infty & \text{otherwise} \end{cases}$$

$$f_{\mathrm{SH}}(u) = (\sqrt{u} - 1)^2 \qquad f^*_{\mathrm{SH}}(t) = \begin{cases} \frac{1}{1/t - 1} & \text{for } t < 1 \\ \infty & \text{otherwise} \end{cases}$$

$$f_{\mathrm{JS}}(u) = \begin{cases} u \log u - (u+1)\log(u+1) + \log 4 & \text{for } u \geq 0 \\ \infty & \text{otherwise} \end{cases} \qquad f^*_{\mathrm{JS}}(t) = \begin{cases} -\log(1 - \exp(t)) - \log 4 & \text{for } t < 0 \\ \infty & \text{otherwise} \end{cases}$$

(Keeping track of the ∞ output is necessary.)

KL=KL, LK=reverse-KL, SH=squared Hellinger distance, JS=JS

Convex conjugate: Examples

We get the following f-divergences:

$$
\begin{aligned}
D_{f_{\mathrm{KL}}}(p\|q) &= D_{\mathrm{KL}}(p\|q) \\
D_{f_{\mathrm{LK}}}(p\|q) &= D_{\mathrm{KL}}(q\|p) \\
D_{f_{\mathrm{SH}}}(p\|q) &= D_{\mathrm{SH}}(q,p) \\
D_{f_{\mathrm{JS}}}(p\|q) &= 2D_{\mathrm{JS}}(q,p)
\end{aligned}
$$

We don't use the following property, but it's interesting so we mention it. If f and f^* are differentiable, then $(f')^{-1} = (f^*)'$:

$$
\begin{aligned}
\tfrac{d}{du} f_{\mathrm{KL}}(u) &= 1 + \log u & \tfrac{d}{dt} f^*_{\mathrm{KL}}(t) &= \exp(t-1) \\
\tfrac{d}{du} f_{\mathrm{LK}}(u) &= -\tfrac{1}{u} & \tfrac{d}{dt} f^*_{\mathrm{LK}}(t) &= -\tfrac{1}{t} \\
\tfrac{d}{du} f_{\mathrm{SH}}(u) &= 1 - \tfrac{1}{\sqrt{u}} & \tfrac{d}{dt} f^*_{\mathrm{SH}}(t) &= \tfrac{1}{(1-t)^2} \\
\tfrac{d}{du} f_{\mathrm{JS}}(u) &= \log \tfrac{u}{1+u} & \tfrac{d}{dt} f^*_{\mathrm{JS}}(t) &= \tfrac{1}{e^{-t}-1}
\end{aligned}
$$

Variational formulation of f-divergence

Variational formulation of f-divergence:

$$\begin{aligned}
D_f(p\|q) &= \int q(x) f\left(\frac{p(x)}{q(x)}\right)\, dx \\
&= \int q(x) \sup_t \left\{ t\frac{p(x)}{q(x)} - f^*(t) \right\}\, dx = \int q(x) T^\star(x) \frac{p(x)}{q(x)} - q(x) f^*(T^\star(x))\, dx \\
&= \sup_{T\in\mathcal{T}} \left(\int p(x)T(x)\, dx - \int q(x) f^*(T(x))\, dx \right) = \sup_{T\in\mathcal{T}} \left(\mathbb{E}_{X\sim p}[T(X)] - \mathbb{E}_{\tilde{X}\sim q}[f^*(T(\tilde{X}))] \right) \\
&\geq \sup_{\phi\in\Phi} \left(\mathbb{E}_{X\sim p}[D_\phi(X)] - \mathbb{E}_{\tilde{X}\sim q}[f^*(D_\phi(\tilde{X}))] \right)
\end{aligned}$$

where $\mathcal{T}$ is the set of all# functions. In particular, $\mathcal{T}$ contains $T^\star(x) = \underset{t}{\mathrm{argmax}}\{t\frac{p(x)}{q(x)} - f^*(t)\}$.

D_ϕ is a neural network parameterized by ϕ.

#All measurable functions.

f-GAN minimax formulation

Minimax formulation of f-GANs.

$$\begin{aligned} &\underset{\theta\in\Theta}{\text{minimize}} \quad D_f(p_{\text{true}}\|p_\theta) \\ \approx \quad &\underset{\theta\in\Theta}{\text{minimize}} \quad \underset{\phi\in\Phi}{\text{maximize}} \quad \mathbb{E}_{X\sim p_{\text{true}}}[D_\phi(X)] - \mathbb{E}_{Z\sim\mathcal{N}(0,I)}[f^*(D_\phi(G_\theta(Z)))] \end{aligned}$$

f-GAN with KL-divergence

Instantiate f-GAN with KL-divergence: $f^*(t) = e^{t-1}$.

$$\begin{aligned}
&\underset{\theta\in\Theta}{\text{minimize}} \quad D_{\mathrm{KL}}(p_{\mathrm{true}} \| p_\theta) \\
\approx\ &\underset{\theta\in\Theta}{\text{minimize}} \quad \underset{\phi\in\Phi}{\text{maximize}} \quad \mathbb{E}_{X\sim p_{\mathrm{true}}}[D_\phi(X)] - \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[e^{D_\phi(G_\theta(Z))-1}\right] \\
\overset{(*)}{=}\ &\underset{\theta\in\Theta}{\text{minimize}} \quad \underset{\phi\in\Phi}{\text{maximize}} \quad 1 + \mathbb{E}_{X\sim p_{\mathrm{true}}}[D_\phi(X)] - \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[e^{D_\phi(G_\theta(Z))}\right] \\
=\ &\underset{\theta\in\Theta}{\text{minimize}} \quad \underset{\phi\in\Phi}{\text{maximize}} \quad \mathbb{E}_{X\sim p_{\mathrm{true}}}[D_\phi(X)] - \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[e^{D_\phi(G_\theta(Z))}\right]
\end{aligned}$$

Step (*) uses the substitution $D_\phi \mapsto D_\phi + 1$, which is valid if the final layer of D_ϕ has a trainable bias term. ($D_\phi : \mathbb{R}^n \to \mathbb{R}$.)

f-GAN with squared Hellinger

Instantiate f-GAN with squared Hellinger distance#: $f^*(t) = \begin{cases} \frac{1}{1/t-1} & \text{if } t < 1 \\ \infty & \text{otherwise} \end{cases}$

$$\begin{array}{llll} & \underset{\theta\in\Theta}{\text{minimize}} & D_{\text{SH}}(p_{\text{true}}, p_\theta) & \\ \approx & \underset{\theta\in\Theta}{\text{minimize}} & \underset{\phi\in\Phi}{\text{maximize}} & \mathbb{E}_{X\sim p_{\text{true}}}[D_\phi(X)] - \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[\frac{1}{1/(D_\phi(G_\theta(Z)))-1}\right] \\ & & \text{subject to} & D_\phi(G_\theta(z)) < 1 \text{ for all } z\in\mathbb{R}^k \end{array}$$

When the constraint is violated, the $f^*(t) = \infty$ case makes the maximization objective $-\infty$. However, directly enforcing the neural networks to satisfy $D_\phi\big(G_\theta(z)\big) < 1$ is awkward.

#The Hellinger distance is a symmetric distance between two probability distributions. Here, we simply use it as yet another distance measure.

Solution: Output activation ρ

When $D_\phi : \mathbb{R}^n \to \mathbb{R}$ and $\{t \mid f^*(t) < \infty\} \neq \mathbb{R}$, then $f^*\left(D_\phi(\tilde{X})\right) = \infty$ is possible. To prevent this, substitute $T(x) \mapsto \rho\left(\tilde{T}(x)\right)$, where $\rho : \mathbb{R} \to \{t \mid f^*(t) < \infty\}$ is a one-to-one function:

$$\begin{aligned}
D_f(p\|q) &= \sup_{T \in \mathcal{T}} \left\{ \mathbb{E}_{X\sim p}[T(X)] - \mathbb{E}_{\tilde{X}\sim q}[f^*(T(\tilde{X}))] \right\} \\
&\overset{(*)}{=} \sup_{\substack{T \in \mathcal{T} \\ f^*(T(x))<\infty}} \left\{ \mathbb{E}_{X\sim p}[T(X)] - \mathbb{E}_{\tilde{X}\sim q}[f^*(T(\tilde{X}))] \right\} \\
&\overset{(**)}{=} \sup_{\tilde{T} \in \mathcal{T}} \left\{ \mathbb{E}_{X\sim p}[\rho(\tilde{T}(X))] - \mathbb{E}_{\tilde{X}\sim q}[f^*(\rho(\tilde{T}(\tilde{X})))] \right\} \\
&\geq \sup_{\phi \in \Phi} \left\{ \mathbb{E}_{X\sim p}[\rho(D_\phi(X))] - \mathbb{E}_{\tilde{X}\sim q}[f^*(\rho(D_\phi(\tilde{X})))] \right\}
\end{aligned}$$

(*) We can restrict the search over T since if $f^*(T(x)) = \infty$, then the objective becomes $-\infty$.[#]

(**) With $T = \rho \circ \tilde{T}$, have $[T \in \mathcal{T}$ and $f^*\left(T(x)\right) < \infty] \Leftrightarrow [\tilde{T} \in \mathcal{T}]$ since ρ is one-to-one.

[#]The precise justification of this step requires more analytical details since the distribution q may not have full support.

f-GAN with output activation

Formulate f-GAN with output activation function ρ:

$$\begin{aligned} &\underset{\theta\in\Theta}{\text{minimize}} \quad D_f(p_{\text{true}}\|p_\theta) \\ \approx \quad &\underset{\theta\in\Theta}{\text{minimize}} \quad \underset{\phi\in\Phi}{\text{maximize}} \quad \mathbb{E}_{X\sim p_{\text{true}}}\left[\rho(D_\phi(X))\right] - \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[f^*(\rho(D_\phi(G_\theta(Z))))\right] \end{aligned}$$

f-GAN with squared Hellinger

Instantiate f-GAN with squared Hellinger distance using $\rho(r) = 1 - e^{-r}$ and

$$f^*(t) = \begin{cases} \dfrac{1}{1/t - 1} & \text{if } t < 1 \\ \infty & \text{otherwise} \end{cases}$$

Note that $f^*\big(\rho(r)\big) = -1 + e^r$.

$$\begin{aligned} & \underset{\theta \in \Theta}{\text{minimize}} \quad D_{\text{SH}}(p_{\text{true}}, p_\theta) \\ \approx\ & \underset{\theta \in \Theta}{\text{minimize}} \quad \underset{\phi \in \Phi}{\text{maximize}} \quad 2 - \mathbb{E}_{X \sim p_{\text{true}}}\left[e^{-D_\phi(X)}\right] - \mathbb{E}_{Z \sim \mathcal{N}(0,I)}\left[e^{D_\phi(G_\theta(Z))}\right] \\ =\ & \underset{\theta \in \Theta}{\text{minimize}} \quad \underset{\phi \in \Phi}{\text{maximize}} \quad -\mathbb{E}_{X \sim p_{\text{true}}}\left[e^{-D_\phi(X)}\right] - \mathbb{E}_{Z \sim \mathcal{N}(0,I)}\left[e^{D_\phi(G_\theta(Z))}\right] \end{aligned}$$

f-GAN with reverse KL

Instantiate f-GAN with reverse KL using $\rho(r) = -e^r$ and

$$f^*(t) = \begin{cases} -1 - \log(-t) & \text{if } t < 0 \\ \infty & \text{otherwise} \end{cases}$$

Note that $f^*\big(\rho(r)\big) = -1 - r$.

$$\begin{aligned} &\underset{\theta\in\Theta}{\text{minimize}} \quad D_{\mathrm{KL}}(p_\theta \| p_{\mathrm{true}}) \\ \approx\ &\underset{\theta\in\Theta}{\text{minimize}} \quad \underset{\phi\in\Phi}{\text{maximize}} \quad 1 - \mathbb{E}_{X\sim p_{\mathrm{true}}}\left[e^{D_\phi(X)}\right] + \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[D_\phi(G_\theta(Z))\right] \\ =\ &\underset{\theta\in\Theta}{\text{minimize}} \quad \underset{\phi\in\Phi}{\text{maximize}} \quad -\mathbb{E}_{X\sim p_{\mathrm{true}}}\left[e^{D_\phi(X)}\right] + \mathbb{E}_{Z\sim\mathcal{N}(0,I)}\left[D_\phi(G_\theta(Z))\right] \end{aligned}$$

Recovering standard GAN

We recover standard GAN with

$$\rho(r) = \log(\sigma(r)), \quad \sigma(r) = \frac{1}{1+e^{-r}}, \quad f^*(t) = \begin{cases} -\log(1-\exp(t)) - \log 4 & \text{for } t < 0 \\ \infty & \text{otherwise} \end{cases}$$

Note that σ is the familiar sigmoid and

$$f^*(\rho(r)) = -\log(1-\sigma(r)) - \log 4$$

$$\underset{\theta\in\Theta}{\text{minimize}} \quad D_{\text{JS}}(p_{\text{true}}, p_\theta)$$

$$\approx \underset{\theta\in\Theta}{\text{minimize}} \quad \underset{\phi\in\Phi}{\text{maximize}} \quad \mathbb{E}_{X\sim p_{\text{true}}}[\log\sigma(D_\phi(X))] + \mathbb{E}_{Z\sim\mathcal{N}(0,I)}[\log(1-\sigma(D_\phi(G_\theta(X))))]$$

where $D_\phi : \mathbb{R}^n \to \mathbb{R}$.

(Standard GAN has $D_\phi : \mathbb{R}^n \to (0,1)$. Here, $(\sigma \circ D_\phi) : \mathbb{R}^n \to (0,1)$ serves the same purpose.)

WGAN

The *Wasserstein GAN* (WGAN) minimizes the Wasserstein distance:

$$\underset{\theta \in \Theta}{\text{minimize}} \quad W(p_{\text{true}}, p_\theta)$$

The $W(p, q)$ is a distance (metric) on probability distributions defined as

$$W(p, q) = \inf_f \mathbb{E}_{(X,Y) \sim f(x,y)} \|X - Y\|$$

where the infimum is taken over joint probability distributions f with marginals p and q, i.e.,

$$p(x) = \int f(x, y)\, dy, \qquad q(y) = \int f(x, y)\, dx$$

(The mathematics of $W(p, q)$ exceeds the scope of this class, but I still want to give you a high-level exposure to WGANs.)

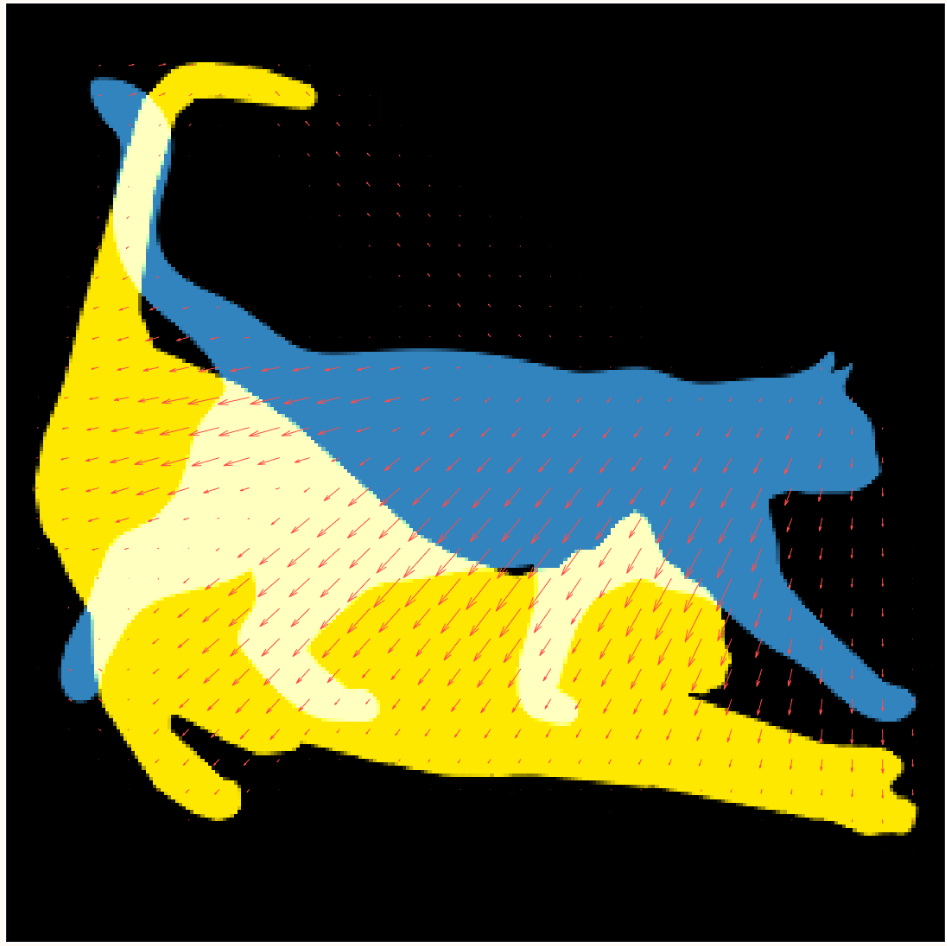
M. Arjovsky, S. Chintala, and L. Bottou, Wasserstein GAN, *ICML*, 2017.

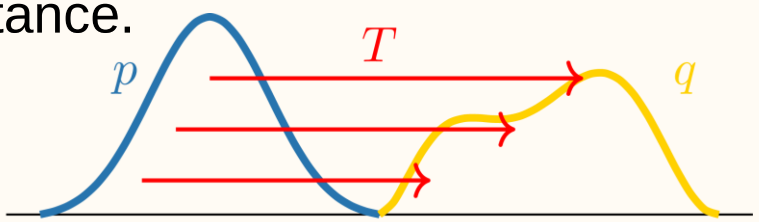
$W(p,q)$ by optimal transport

Another equivalent formulation of the Wasserstein distance is by the theory of optimal transport. Given distributions p and q (initial and target)

$$W(p,q) = \inf_T \int \|x - T(x)\| p(x)\, dx$$

where T is a transport plan that transports p to q.[%] Figuratively speaking, we are transporting grains of sand from one pile to another, and we wan to minimize the aggregate transport distance.





Transport plan T from p to q.

[%]In measure theoretic language, $q = f_\# p$.

Image from W. Li, E. K. Ryu, S. Osher, W. Yin, and W. Gangbo, A parallel method for earth mover's distance, *J. Sci. Comput.*, 2018.

Minimax via KR duality

Kantorovich–Rubinstein duality[#] establishes:

$$W(p_{\mathrm{true}}, p_\theta) = \sup_{\|T\|_L \le 1} \mathbb{E}_{X \sim p_{\mathrm{true}}}[T(X)] - \mathbb{E}_{\tilde{X} \sim p_\theta}[T(\tilde{X})]$$

Minimax formulation of WGAN:

$$\begin{array}{lll} \underset{\theta \in \Theta}{\text{minimize}} & W(p_{\mathrm{true}}, p_\theta) & \\ \approx \underset{\theta \in \Theta}{\text{minimize}} & \underset{\phi \in \Phi}{\text{maximize}} & \mathbb{E}_{X \sim p_{\mathrm{true}}}[D_\phi(X)] - \mathbb{E}_{\tilde{X} \sim p_\theta}[D_\phi(\tilde{X})] \\ & \text{subject to} & D_\phi \text{ is 1-Lipschitz} \end{array}$$

[#]L.V. Kantorovich and G. Rubinstein, On a space of completely additive functions, *Vestnik Leningradskogo Universiteta*, 1958.
The Kantorovich–Rubinstein dual as the convex (Lagrange) dual of a “flux” formulation of the optimal transport.

Spectral normalization

How do we enforce the constraint that D_ϕ is 1-Lipschitz? Consider an MLP:

$$\begin{aligned} y_L &= A_L y_{L-1} + b_L \\ y_{L-1} &= \sigma(A_{L-1} y_{L-2} + b_{L-1}) \\ &\vdots \\ y_2 &= \sigma(A_2 y_1 + b_2) \\ y_1 &= \sigma(A_1 x + b_1), \end{aligned}$$

where σ is a 1-Lipschitz continuous activation function, such as ReLU and tanh. If

$$\|A_i\|_{\text{op}} = \sigma_{\max}(A_i) \le 1$$

for $i = 1, \dots, L$, where $\sigma_{\max}$ denotes the largest singular value, then each layer is 1-Lipschitz continuous and the entire mapping $x \mapsto y_L$ is 1-Lipschitz. (A sufficient, but not a necessary, condition.)

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, Spectral normalization for generative adversarial networks, *ICLR*, 2018.

Spectral normalization

Replace Lipschitz constraint with a singular-value constraint

$$\begin{array}{lll} \underset{\theta\in\Theta}{\text{minimize}} & \underset{\phi\in\Phi}{\text{maximize}} & \frac{1}{N}\sum_{i=1}^{N} D_\phi(X_i) - \mathbb{E}_{Z\sim\mathcal{N}(0,I)}[D_\phi(G_\theta(Z))] \\ & \text{subject to} & \sigma_{\max}(A_i) \le 1, \qquad i = 1,\dots,L \end{array}$$

Constraint is handled with a projected gradient method. (Note that $A_1, \dots, A_L$ are part of the discriminator parameters ϕ.)

(Specifically, one performs an (approximate) projection after the ascent step in the stochastic gradient ascent-descent methods. The approximate projection involves computing the largest singular with the power iteration.)

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, Spectral normalization for generative adversarial networks, *ICLR*, 2018.

Conclusion

We discussed the following unsupervised learning techniques:

- Autoencoders
- Flow models
- Variational autoencoders
- GANs

Unsupervised learning techniques, particularly generative models, tend to utilize more math in their formulations. This chapter provided a brief and gentle introduction to the mathematical foundations of these formulations.