Deep Learning

Deep generative networks

Alexandre Fournier Montgieux

M2 BDMA, CentraleSupélec, Université Paris Saclay

October 15, 2024

Plan

Supervised vs Unsupervised learning

Supervised learning

- **Data:** data points X and labels Y
- **Goal:** learn a *function* that maps $f : X \to Y$
- **Examples:** Classification, regression, object detection, semantic segmentation, image captioning, ...

Classification

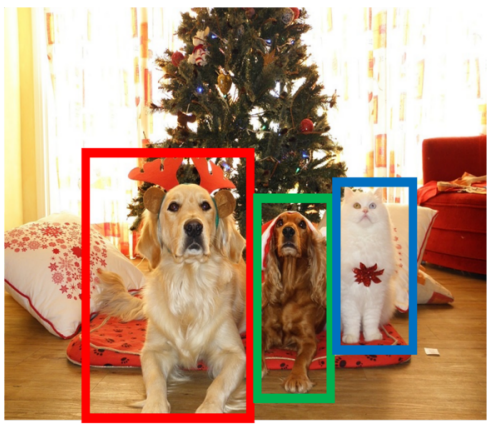
Cat

Supervised vs Unsupervised learning

Supervised learning

- **Data:** data points X and labels Y
- **Goal:** learn a *function* that maps $f : X \to Y$
- **Examples:** Classification, regression, object detection, semantic segmentation, image captioning, ...

Object Detection

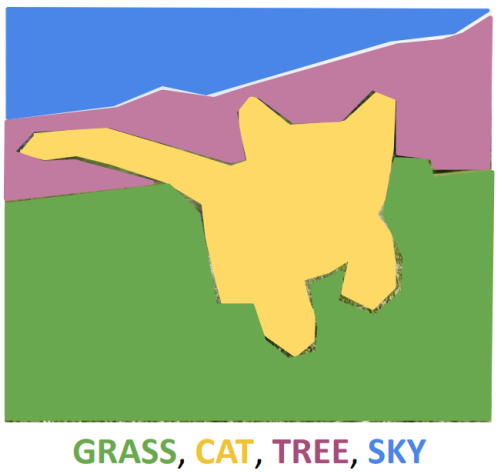


DOG, DOG, CAT

Supervised vs Unsupervised learning

Supervised learning

- **Data:** data points X and labels Y
- **Goal:** learn a *function* that maps $f : X \to Y$
- **Examples:** Classification, regression, object detection, semantic segmentation, image captioning, ...



Supervised vs Unsupervised learning

Supervised learning

- **Data:** data points X and labels Y
- **Goal:** learn a *function* that maps $f : X \to Y$
- **Examples:** Classification, regression, object detection, semantic segmentation, image captioning, ...

Image captioning

A cat sitting on a suitcase on the floor

Supervised vs Unsupervised learning

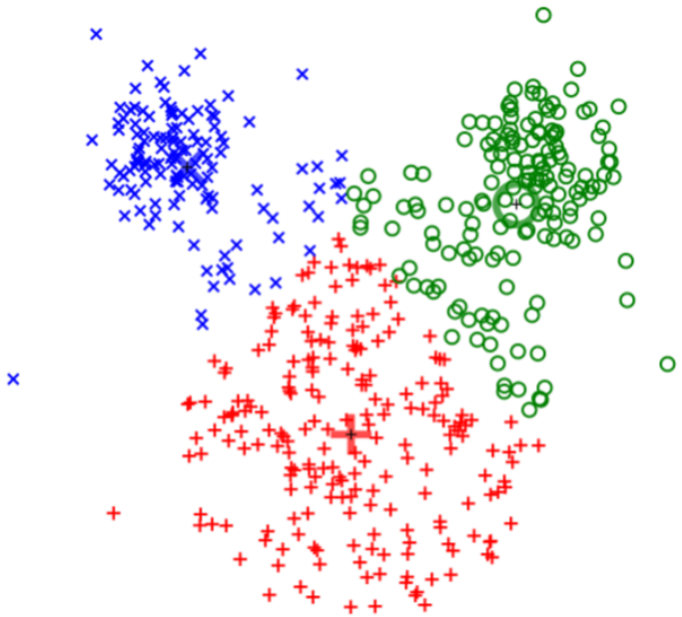
Supervised learning

- **Data:** data points X and labels Y
- **Goal:** learn a *function* that maps $f : X \to Y$
- **Examples:** Classification, regression, object detection, semantic segmentation, image captioning, ...

Unsupervised learning

- **Data:** X just data, no labels !
- **Goal:** learn underlying *structure* of the data
- **Examples:** Clustering, dimensionality reduction, feature learning, density estimation, ...

Supervised vs Unsupervised learning

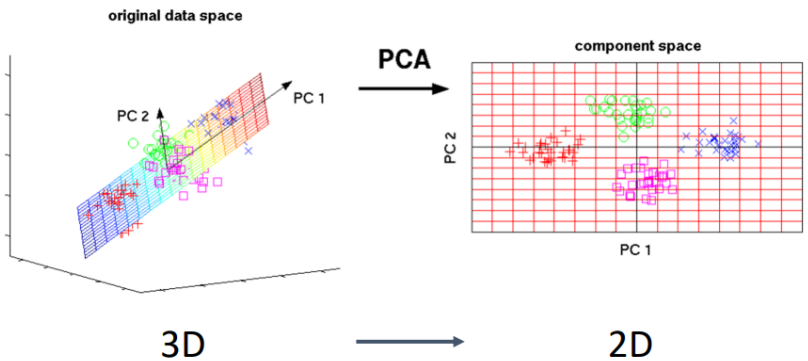


Unsupervised learning

- **Data:** X just data, no labels !
- **Goal:** learn underlying *structure* of the data
- **Examples:** Clustering, dimensionality reduction, feature learning, density estimation, ...

Supervised vs Unsupervised learning

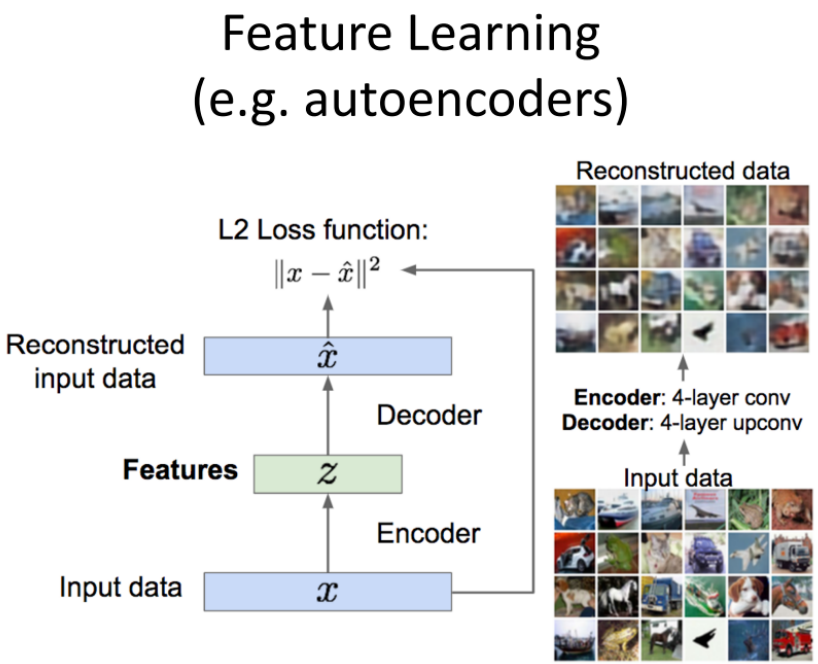
Dimensionality Reduction
(e.g. Principal Components Analysis)



Unsupervised learning

- **Data:** X just data, no labels !
- **Goal:** learn underlying *structure* of the data
- **Examples:** Clustering, dimensionality reduction, feature learning, density estimation, ...

Supervised vs Unsupervised learning

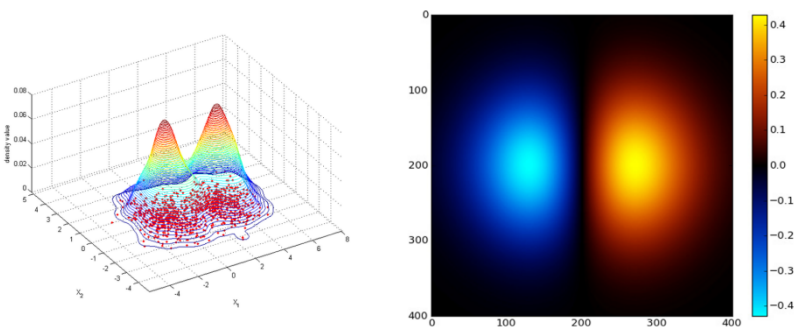


Unsupervised learning

- **Data:** X just data, no labels !
- **Goal:** learn underlying *structure* of the data
- **Examples:** Clustering, dimensionality reduction, feature learning, density estimation, ...

Supervised vs Unsupervised learning

Density Estimation



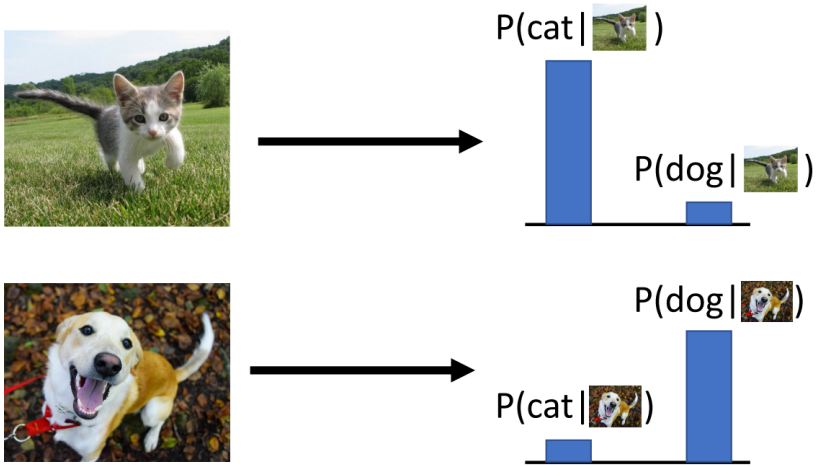
Unsupervised learning

- **Data:** X just data, no labels !
- **Goal:** learn underlying *structure* of the data
- **Examples:** Clustering, dimensionality reduction, feature learning, density estimation, ...

Discriminative vs Generative models

Discriminative model:

- learn a probability distribution $p(y|x)$
- given X predict Y

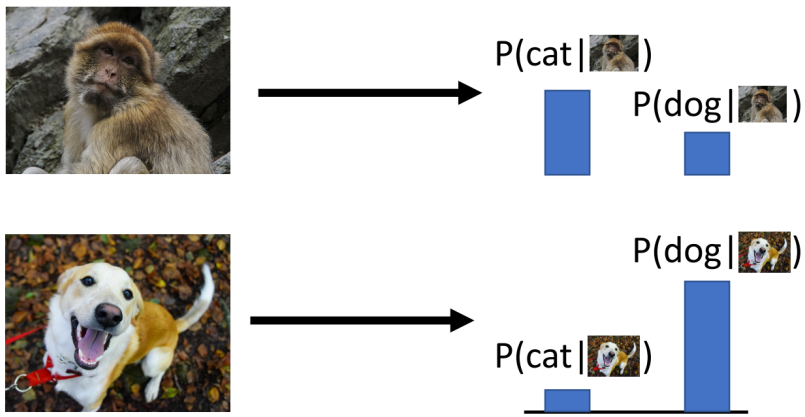


Possible labels for each input "compete" for probability mass. But no competition **between images**.

Discriminative vs Generative models

Discriminative model:

- learn a probability distribution $p(y|x)$
- given X predict Y



No way for the model to handle unreasonable inputs; it must give label distributions for all images.

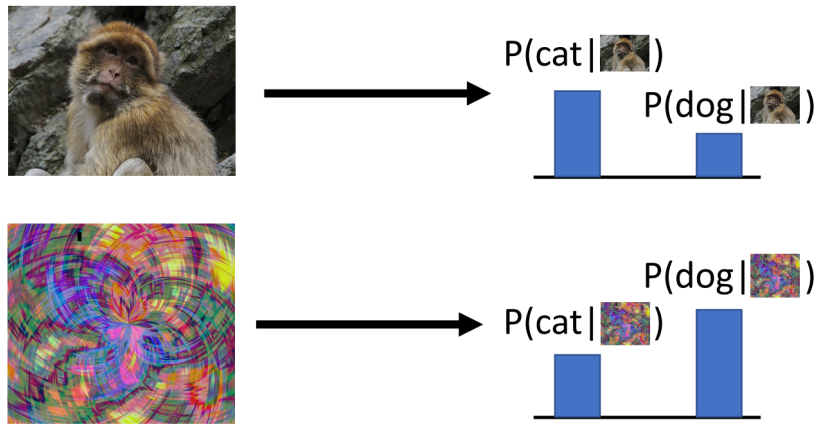
Discriminative vs Generative models

Discriminative model:

- learn a probability distribution $p(y|x)$
- given X predict Y

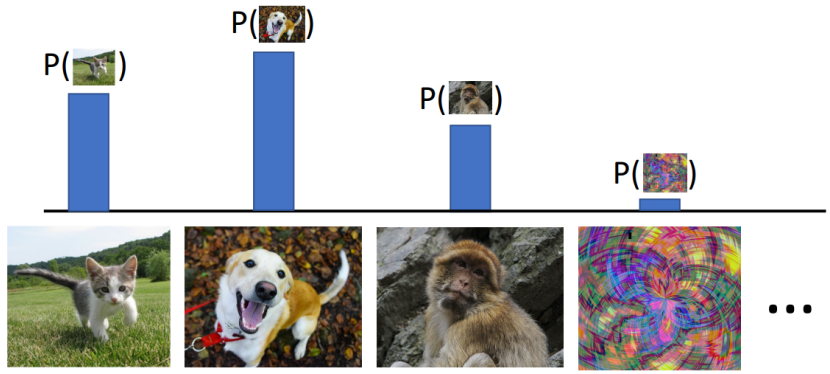
But...

- can't model $p(x)$
- can't generate X (image...)



No way for the model to handle unreasonable inputs; it must give label distributions for all images.

Discriminative vs Generative models



Generative model:

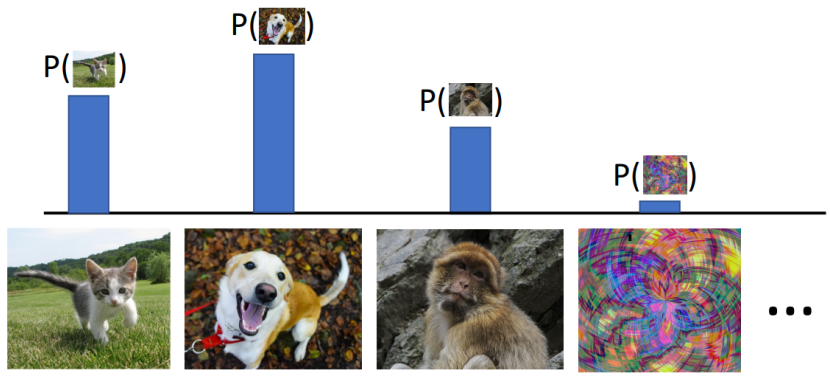
- learn a probability distribution $p(x)$
- can generate images !

- All possible images compete with each other for probability mass.
- Requires deep image understanding! Is a dog more likely to sit or stand? How about 3-legged dog vs 3-armed monkey?
- Model can "reject" unreasonable inputs by

Discriminative vs Generative models

Conditional generative model:

- learn a probability distribution $p(x|y)$
- examples: images from label/text, text-to-speech, music generation from score...



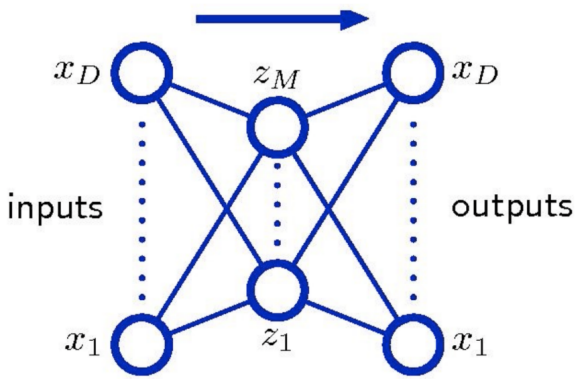
Each possible label induces a competition among all images.

Types of generative models

- Autoregressive models (see LLM class)
- Autoencoders VAE
- Generative adversarial networks (GAN)
- Diffusion models

Plan

Non-linear Dimension Reduction



- Neural networks can be used for nonlinear dimensionality reduction.
- This is achieved by having the same number of outputs as inputs. These models are called autoencoders.
- Consider a multilayer perceptron that has D inputs, D outputs, and M hidden units, with $M < D$.
- We can squeeze the information through some kind of bottleneck.
- If we use a linear network (linear activation) this is very similar to Principal Components Analysis.

Autoencoders and PCA

- Given an input $\boldsymbol{x}$, its corresponding reconstruction is given by:

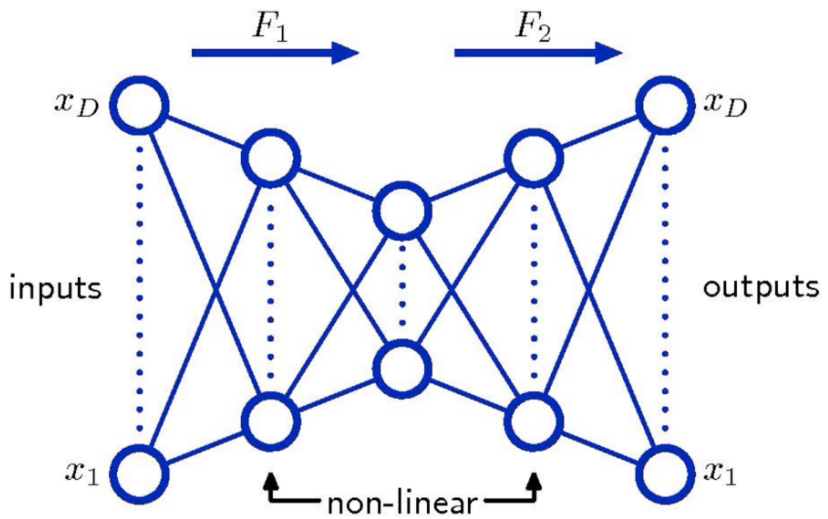
$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=1}^{M} w_{kj}^{(2)} \sigma \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i \right), \quad k = 1, \ldots, D$$

- We can determine the network parameters $\boldsymbol{w}$ by minimizing the reconstruction error:

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \| y\left(\boldsymbol{x}_n, \boldsymbol{w}\right) - \boldsymbol{x}_n \|^2$$

- If the hidden and output layers are linear, it will learn hidden units that are linear functions of the data and minimize squared error.
- M hidden units will span the same space as the first M principal components.

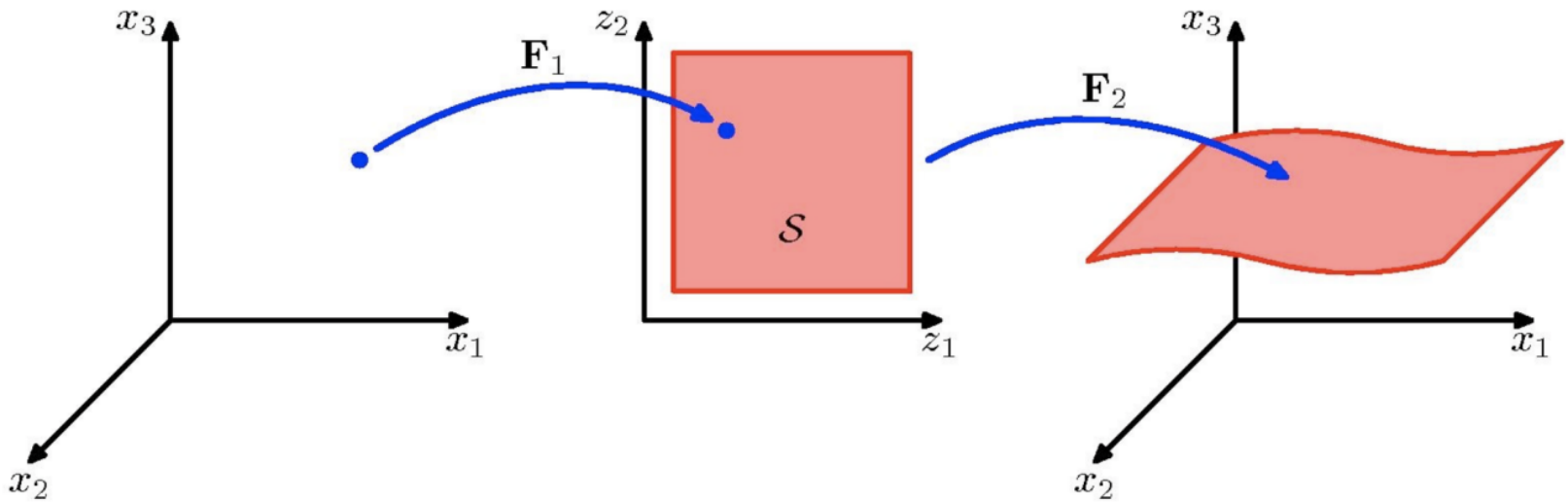
Deep Autoencoders



- We can put extra nonlinear hidden layers between the input and the bottleneck and between the bottleneck and the output.
- This gives nonlinear generalization of PCA, providing non-linear dimensionality reduction.
- The network can be trained by the minimization of the reconstruction error function.
- Much harder to train.

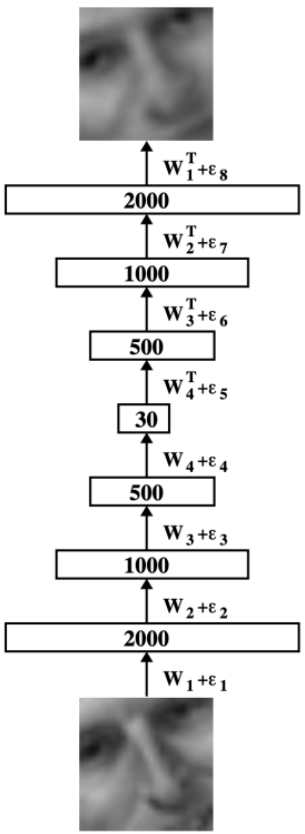
Geometrical Interpretation

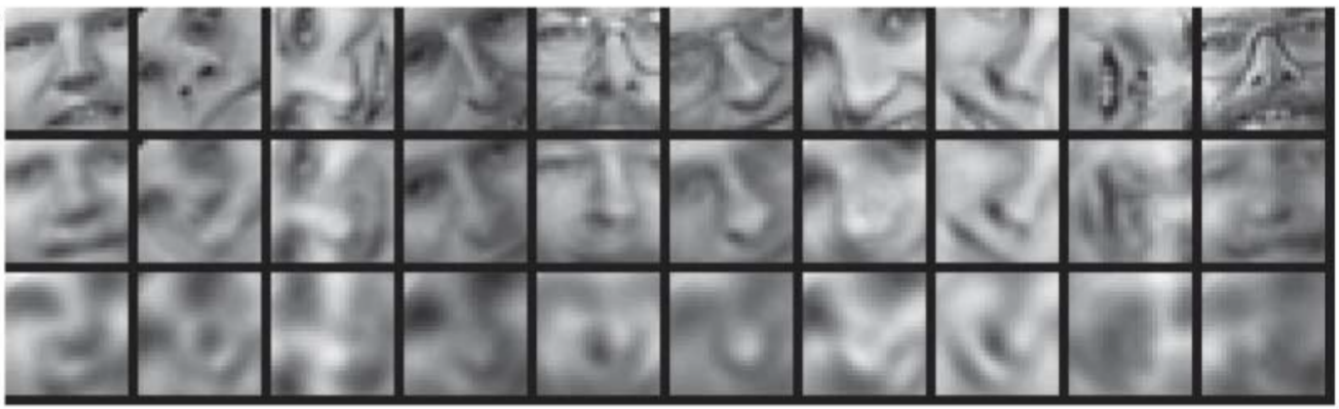
- Geometrical interpretation of the mappings performed by the network with 2 hidden layers for the case of $D = 3$ and $M = 2$ units in the middle layer.



- The mapping F_1 defines a nonlinear projection of points in the original D-space into the M-dimensional subspace.
- The mapping F_2 maps from an M-dimensional space into D-dimensional space.

Deep Autoencoders

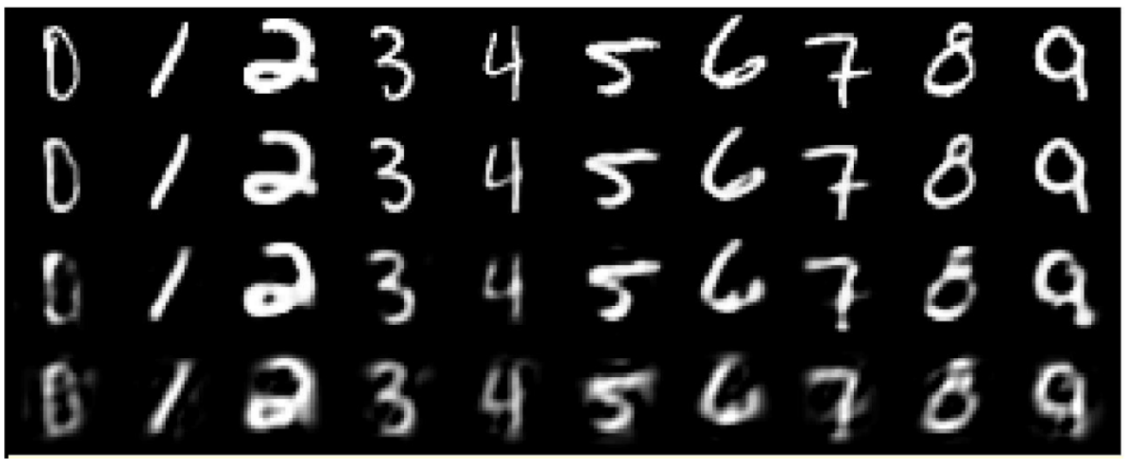




- We can consider very deep autoencoders.
- By row: Real data, Deep autoencoder with a bottleneck of 30 linear units, and 30-d PCA.

Deep Autoencoders

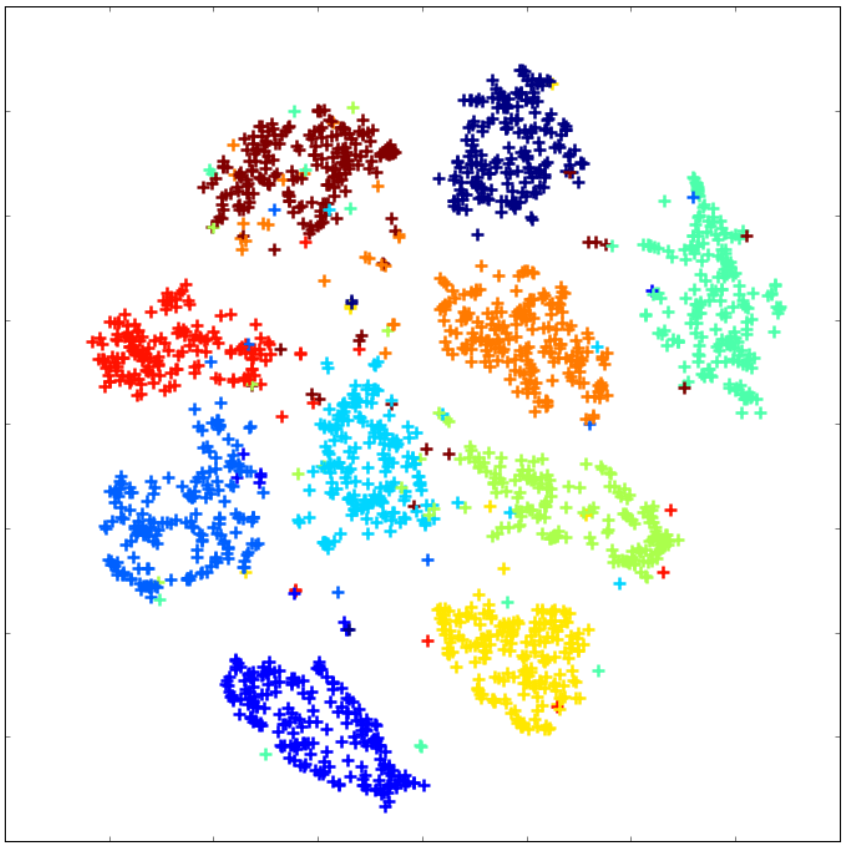
- Similar model for MNIST handwritten digits:



- Deep autoencoders produce much better reconstructions.

Class Structure of the Data

- Do the 2-D codes found by the deep autoencoder preserve the class structure of the data?

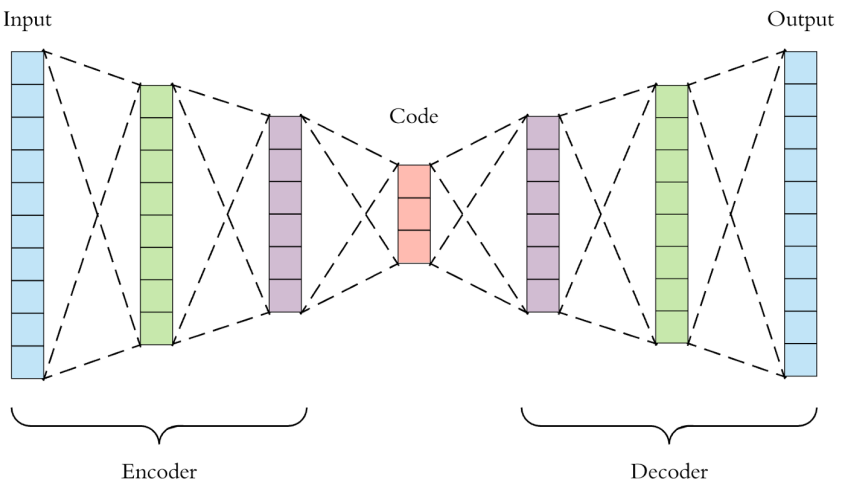


- This is how the codes would look like in the latent space.

Autoencoders: Summary

Autoencoders reconstruct their input via an encoder and a decoder.

- **Encoder**: $g(x) = z \in F, \quad x \in X$
- **Decoder**: $f(z) = \tilde{x} \in X$
- where X is the data space, and F is the feature (latent) space.
- z is the code, compressed representation of the input, x. It is important that this code is a bottleneck, i.e. that $\dim F \ll \dim X$
- Goal: $\tilde{x} = f(g(x)) \approx x$



Issues with (deterministic) Autoencoders

- **Issue 1:** Proximity in data space does not mean proximity in feature space
  - The codes learned by the model are deterministic, i.e.

$$g(x_1) = z_1 \Rightarrow f(z_1) = \tilde{x}_1$$
$$g(x_2) = z_2 \Rightarrow f(z_2) = \tilde{x}_2$$

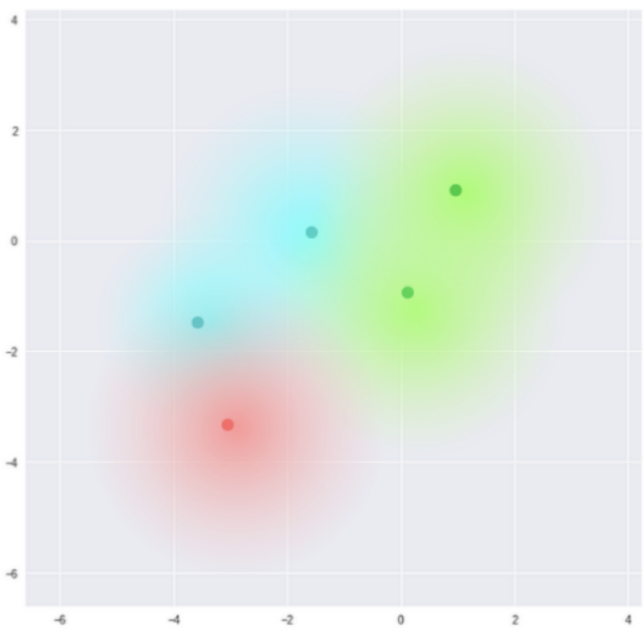
  - but proximity in feature space is not "directly" enforced for inputs in close proximity in data space, i.e.

$$x_1 \approx x_2 \neq z_1 \approx z_2$$

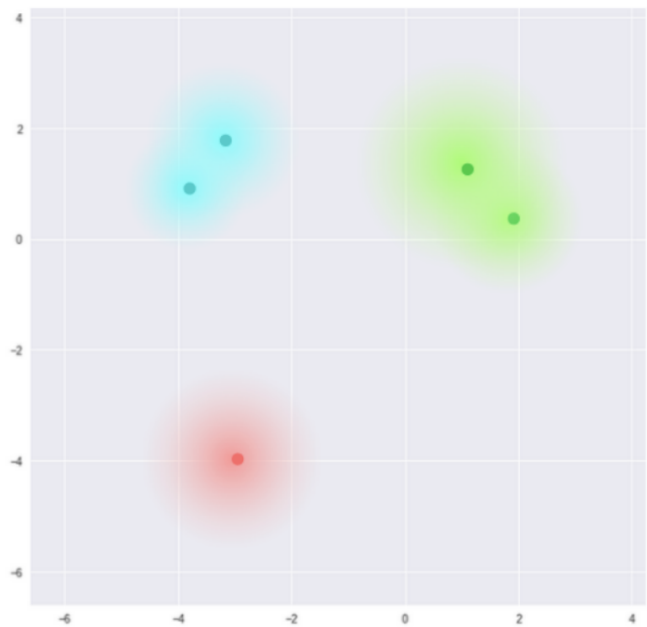
  - The latent space may not be continuous, or allow easy interpolation.

Issues with (deterministic) Autoencoders

- **Issue 1:** Proximity in data space does not mean proximity in feature space
  - If the space has discontinuities (eg. gaps between clusters) and you sample/generate a variation from there, the decoder will simply generate an unrealistic output.



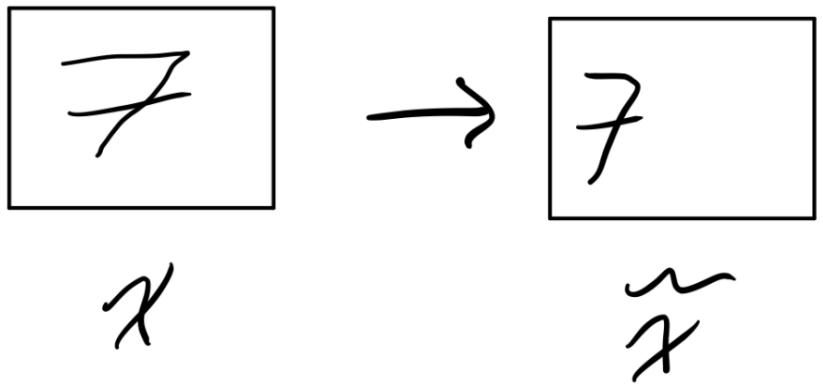
What we require



What we may inadvertently end up with

Issues with (deterministic) Autoencoders

- **Issue 2:** How to measure the goodness of a reconstruction?



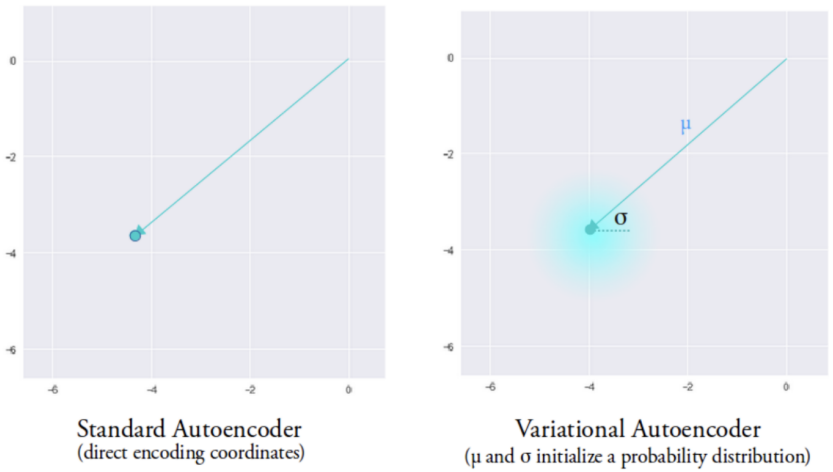
  - The reconstruction looks quite good. However, if we chose a simple distance metric between inputs and reconstructions, we would heavily penalize the left-shift in the reconstruction $\tilde{x}$.
  - Choosing an appropriate metric for evaluating model performance can be difficult, and that a miss-aligned objective can be disastrous.

Variational Autoencoders

- Variational autoencoders (VAEs) encode inputs with uncertainty.
- Unlike standard autoencoders, the encoder of a VAE outputs a probability distribution, $q_\phi(z \mid x)$.
- Instead of the encoder learning an encoding vector, it learns two vectors: vector of means, μ, and another vector of standard deviations, σ.

Variational Autoencoders

- The mean μ controls where encoding of input is centered while the standard deviation controls how much can the encoding vary.



- Encodings are generated at random from the "circle", the decoder learns that all nearby points refer to the same input.

VAE: Specifics

- Our model is generated by the joint distribution over the latent codes and the input data $p(x, z)$. Decomposing

$$p(x, z) = \text{ prior } \times \text{ likelihood } = p(z)p(x \mid z)$$

- The encoder is $p(z \mid x) = p(x, z)/p(x)$
- However, learning $p(x) = \int p(x \mid z)p(z)dz$ is intractable.
- We introduce an approximation with its own set of parameters, q_ϕ, and learn these parameters such that

$$q_\phi(z \mid x) \approx p(z \mid x)$$

VAE: Specifics

- VI idea: we want to maximize $\log p(x)$ which was lower bounded by the ELBO

$$\begin{aligned}\mathcal{L}(\theta, \phi; x) &= \text{ELBO} \\ &= \mathbb{E}_{z \sim q_\phi}\left[\log p_\theta(x \mid z)\right] - D_{KL}\left(q_\phi(z \mid x) \| p(z)\right)\end{aligned}$$

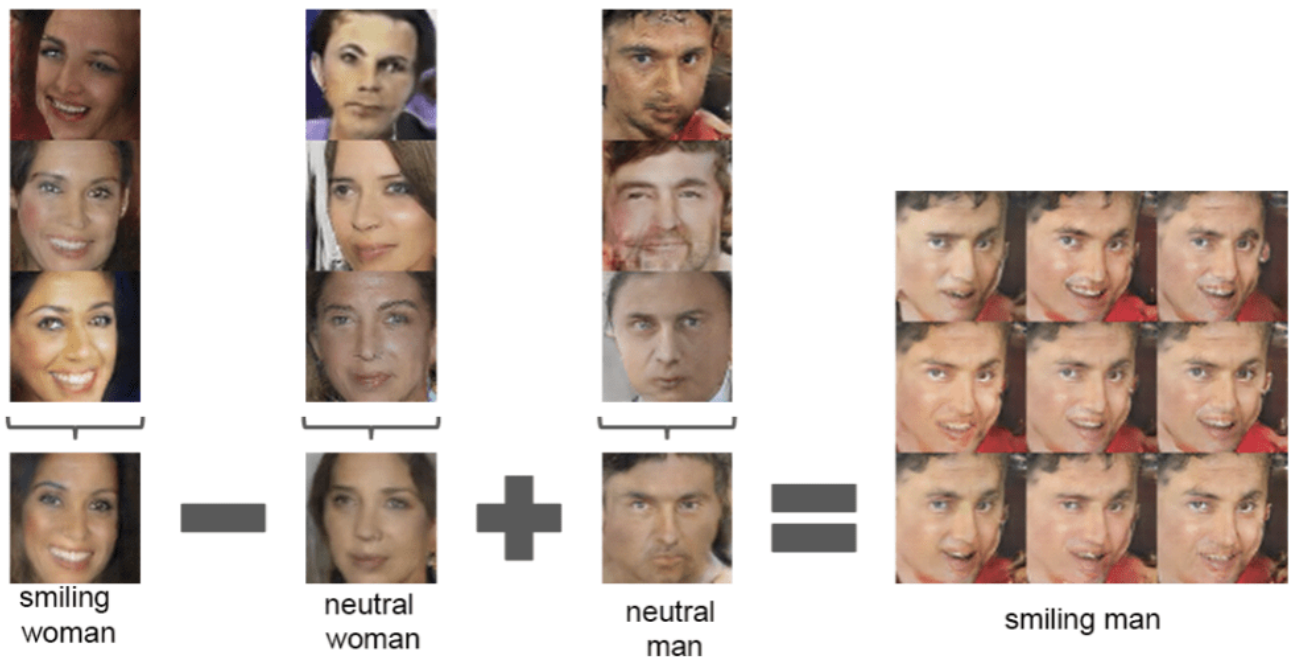
- First term is the expected log-likelihood and the second is the divergence of q_ϕ from the true prior.
- The encoder and decoder in a VAE become:
  - **Encoder:** $q_{\phi_i}\left(z \mid x_i\right) = \mathcal{N}\left(\mu_i, \sigma_i^2\right)$ where $\phi_i = (\mu_i, \log \sigma_i)$
  - **Decoder:** $f\left(z_i\right) = \theta_i$ typically a neural network

After VAE is trained

- Once a VAE is trained, we can sample new inputs

$$z \sim p(z) \quad \tilde{x} \sim p_\theta(x \mid z)$$

- We can also interpolate between inputs, using simple vector arithmetic.



Example: MNIST

- We choose the prior on z to be the standard Gaussian

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

- our likelihood function to be

$$p_\theta(x \mid z) = \text{Bernoulli}(\theta)$$

- and our approximate posterior is

$$q_{\phi_i}(z \mid x_i) = \mathcal{N}\left(\mu_i, \sigma_i^2 I\right)$$

- To get our reconstructed input, we simply evaluate

$$\tilde{x} \sim p_\theta(x \mid z)$$

- We will use neural networks as our encoder and decoder

The Reparametrization Trick

- Encoder generates a code by sampling from the distribution $q_\phi(z \mid x)$.
- This sampling process introduces a major problem: gradients are blocked from flowing into the encoder, and hence it will not train.
- To solve this problem, we use the **reparameterization trick**:
- Instead of sampling z directly from its distribution (e.g. $z_i \sim \mathcal{N}\left(\mu_i, \sigma_i^2\right)$) we express z_i as

$$z_i = \mu_i + \sigma_i \times \varepsilon_i \quad \text{where } \varepsilon_i \sim \mathcal{N}(0, I)$$

with this, gradients can now flow through the entire network.

Recap: Standard VAE

- VI idea: we want to maximize $\log p(x)$ which was lower bounded by the ELBO

$$\begin{aligned}\mathcal{L}(\theta, \phi; x) &= \text{ELBO} \\ &= \mathbb{E}_{z \sim q_\phi}\left[\log p_\theta(x \mid z)\right] - D_{KL}\left(q_\phi(z \mid x) \| p(z)\right)\end{aligned}$$

  which is the loss function we use when training VAEs.
- First term is the expected log-likelihood and the second is the divergence of q_ϕ from the true prior.
- The encoder and decoder in a VAE become:
  - **Encoder:** $q_{\phi_i}\left(z \mid x_i\right) = \mathcal{N}\left(\mu_i, \sigma_i^2\right)$ where $\phi_i = (\mu_i, \log \sigma_i)$
  - **Decoder:** $f(z) = \theta$ typically a neural network. The number of parameters for the encoder: $N \times \left(|\mu_i| + \left|\sigma_i^2\right|\right)$
- The number of parameters for the encoder: $N \times \left(|\mu_i| + \left|\sigma_i^2\right|\right)$

Amortized Inference

- Instead of doing VI from scratch every time we see a new datapoint, we learn a function that can look at the data for a person x_i, and then output an approximate posterior $q_\phi(z_i \mid x_i)$. We'll call this a "recognition model"
- Instead of a separate ϕ_i for each data example, we'll just have a single global ϕ that specifies the parameters of the recognition model.
- Because the relationship between data and posteriors is complex and hard to specify by hand, we'll do this with a neural network!

Amortized Inference

- We can simply have a network take in x_i, and output the mean and variance vector for a Gaussian:
- Then the approximate posterior is given by

$$q_\phi\left(z_i \mid x_i\right) = \mathcal{N}\left(z_i \mid \mu_\phi\left(x_i\right), \Sigma_\phi\left(x_i\right)\right)$$

VAE vs Amortized VAE Pipeline

- For a given input (or minibatch) x_i
  - **Standard VAE**
  - $z_i \sim q_{\phi_i}(z \mid x_i) = \mathcal{N}\left(\mu_i, \sigma_i^2 I\right)$
  - **Amortized VAE**
  - $z_i \sim q_{\phi}(z \mid x_i) = \mathcal{N}\left(\mu_{\phi}(x_i), \Sigma_{\phi}(x_i)\right)$
- Run the code through decoder and get likelihood: $p_\theta(x \mid z)$
  Compute the loss function:

$$L(x; \theta, \phi) = -E_{z_\phi \sim q_\phi}\left[\log p_\theta(x \mid z)\right] + KL\left(q_\phi(z \mid x) \| p(z)\right)$$

- Use gradient-based optimization to backpropogate $\nabla_\theta L, \nabla_\phi L$

Standard vs Amortized VAE

- This allows us to use the share parameters for all data points, and reduce the number of parameter for the encoder to that of the encoding NN.
- Standard VAE encoder is more expressive since no parameters are shared across different data points.

Example: MNIST

- We choose the prior on z to be the standard Gaussian

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

- our likelihood function to be

$$p_\theta(x \mid z) = \text{Bernoulli}(\theta)$$

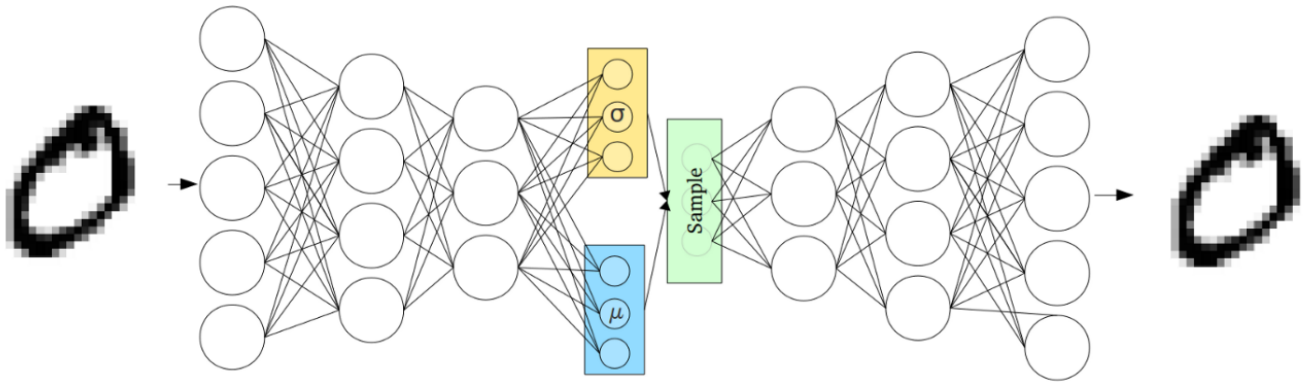
- and our approximate posterior is

$$q_\phi\left(z \mid x_i\right) = \mathcal{N}\left(\mu_\phi\left(x_i\right), \Sigma_\phi\left(x_i\right)\right)$$

- Finally, we use neural networks as our encoder and decoder
  - **Encoder:** $g_\phi\left(x_i\right) = \left[\mu\left(x_i\right), \log \Sigma\left(x_i\right)\right]$
  - $f_\theta\left(z_i\right) = \theta\left(z_i\right)$
  - where θ_i are parameters of a Bernoulli rv for each input pixel.
- To get our reconstructed input, we simply evaluate

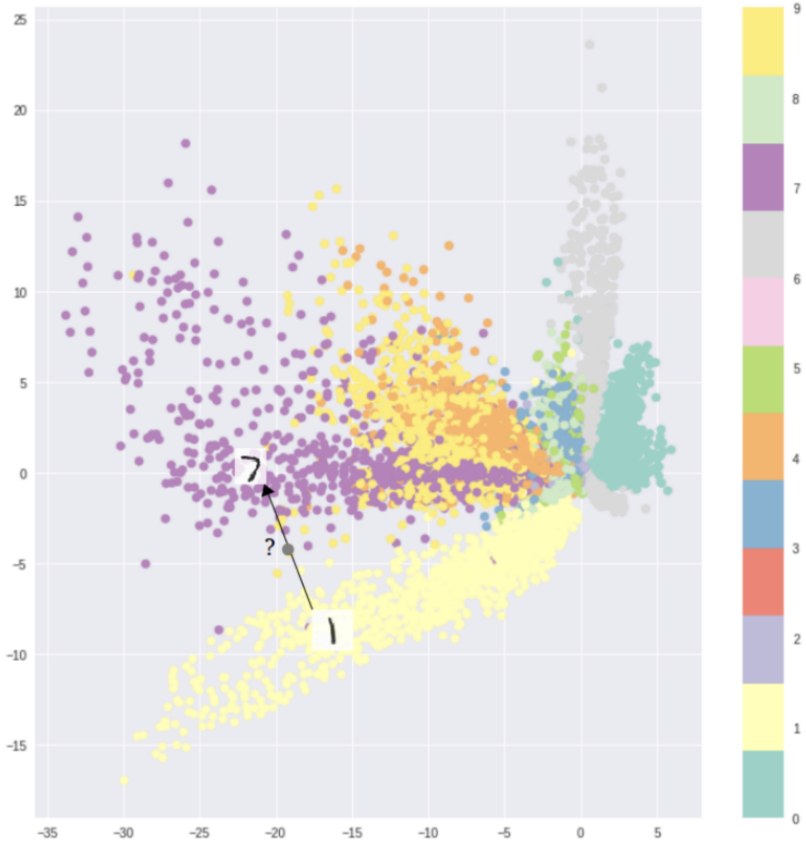
$$\tilde{x} \sim p_\theta(x \mid z)$$

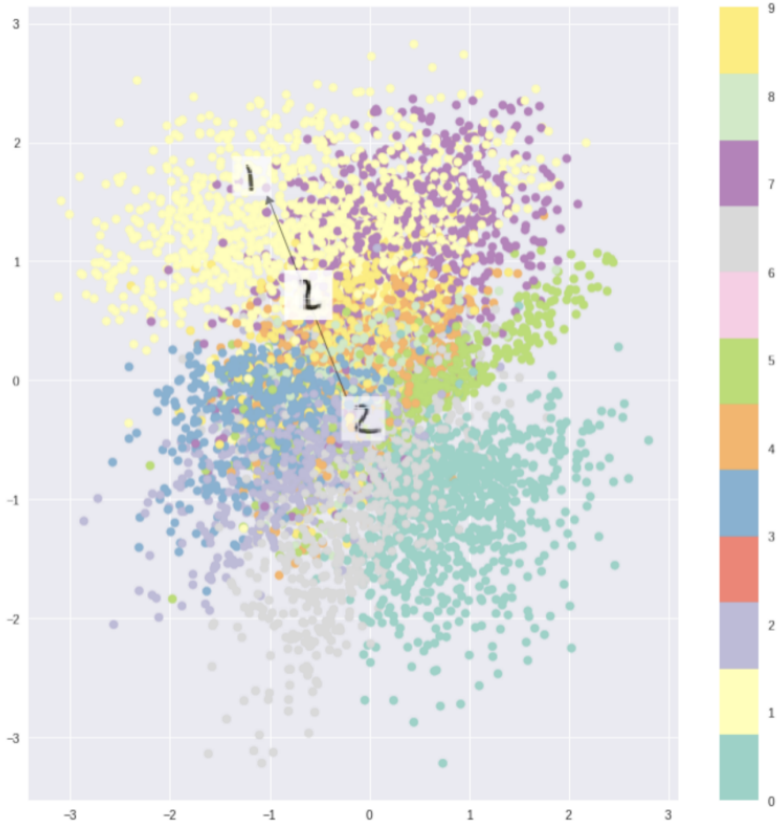
Example: MNIST



- We use neural networks for both the encoder and the decoder.
- We compute the loss function $-\mathcal{L}(\theta, \phi; x)$ and propagate its derivative with respect to θ and $\phi, \nabla_\theta L, \nabla_\phi L$, through the network during training.
- We need reparametrization trick as before!

MNIST: Autoencoder vs VAE

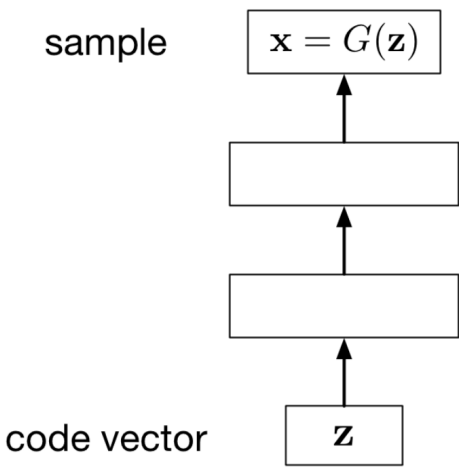




Plan

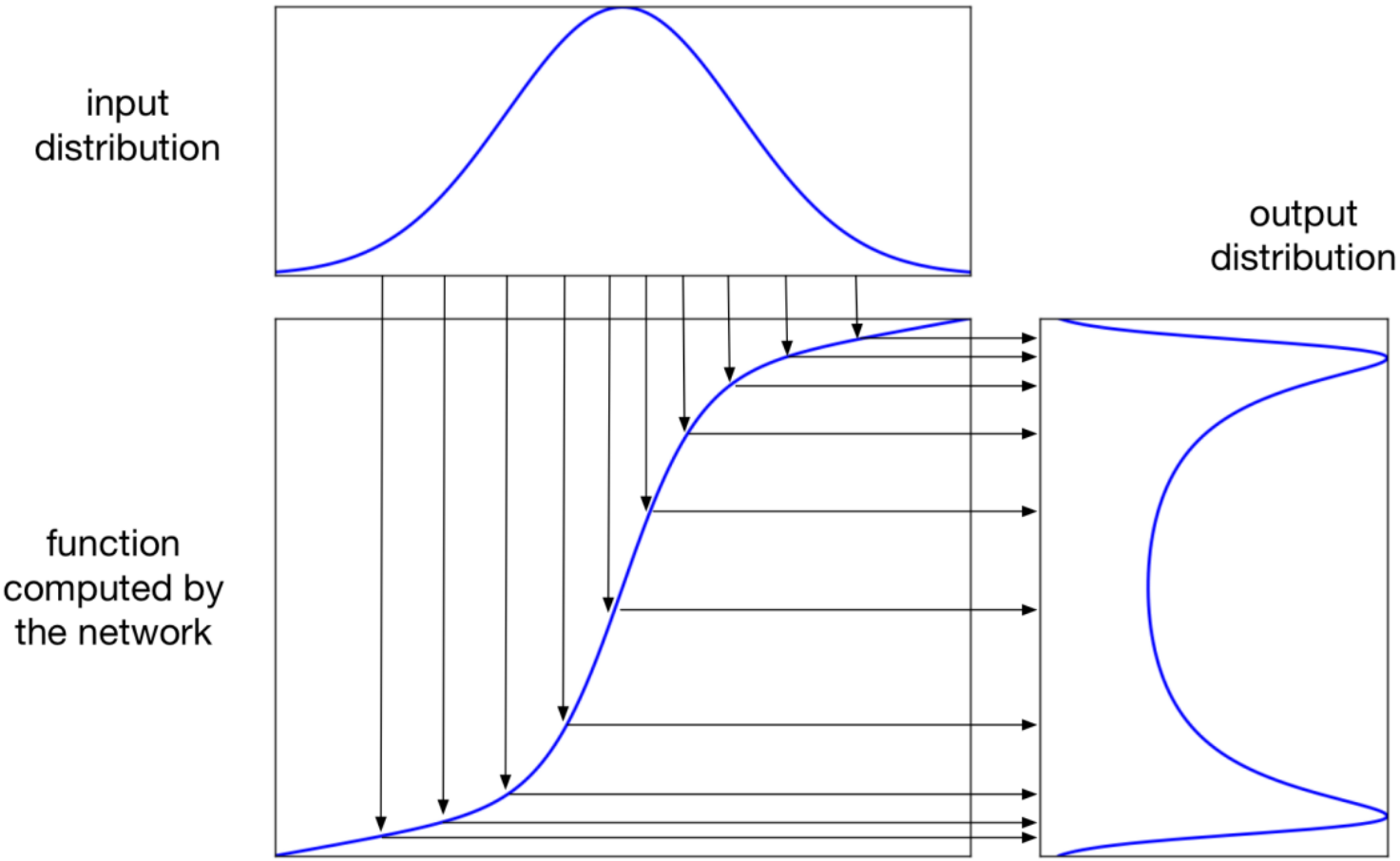
Implicit Generative Models

- **Implicit generative models** implicitly define a probability distribution
- Start by sampling the **code vector** z from a fixed, simple distribution (e.g. spherical Gaussian)
- The **generator network** computes a differentiable function G mapping z to an x in data space

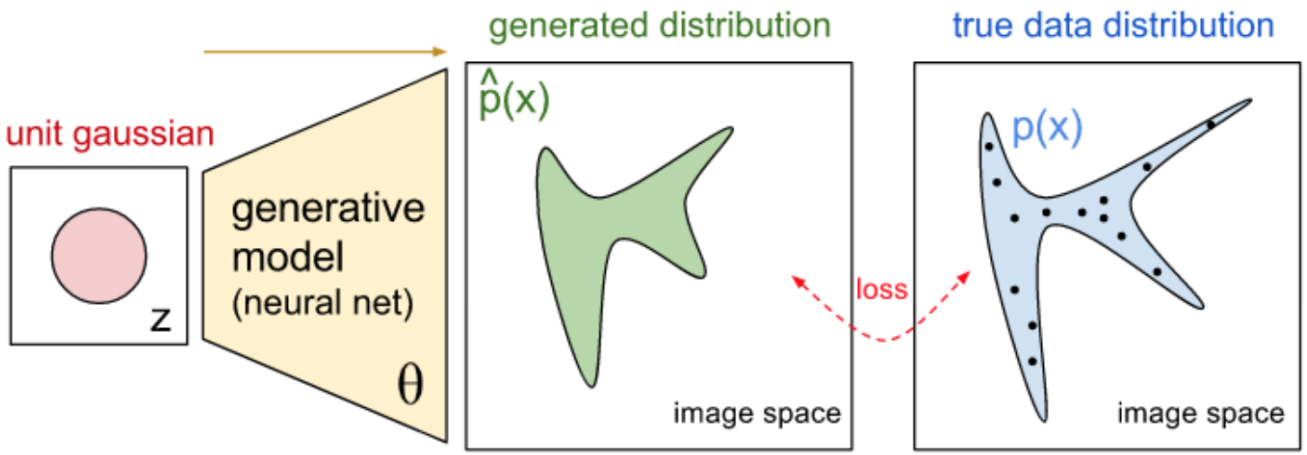


Implicit Generative Models

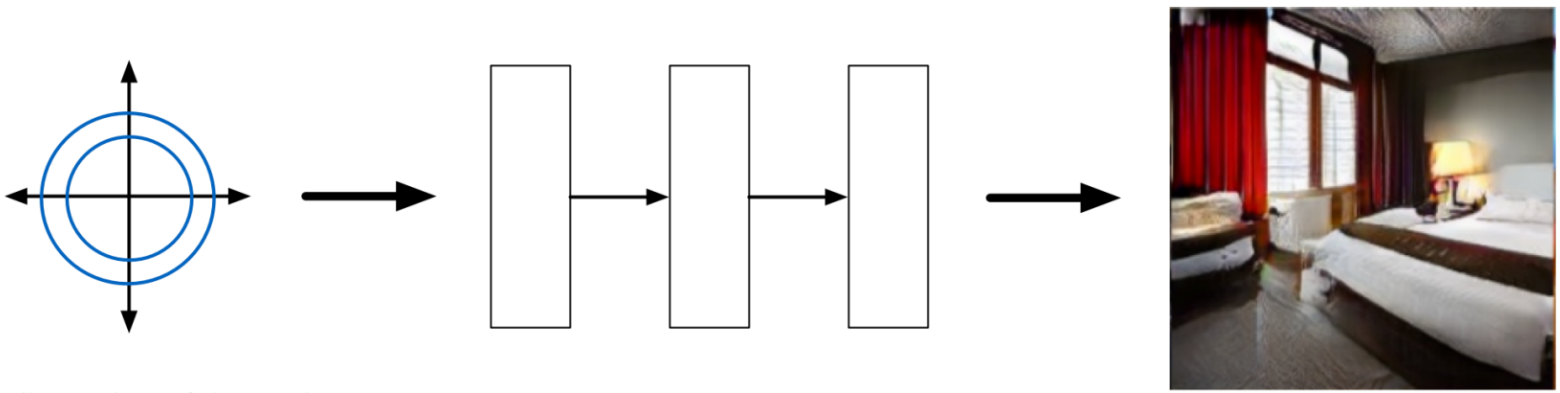
A 1-dimensional example



Implicit Generative Models



Implicit Generative Models



Each dimension of the code vector is sampled independently from a simple distribution, e.g. Gaussian or uniform.

This is fed to a (deterministic) generator network.

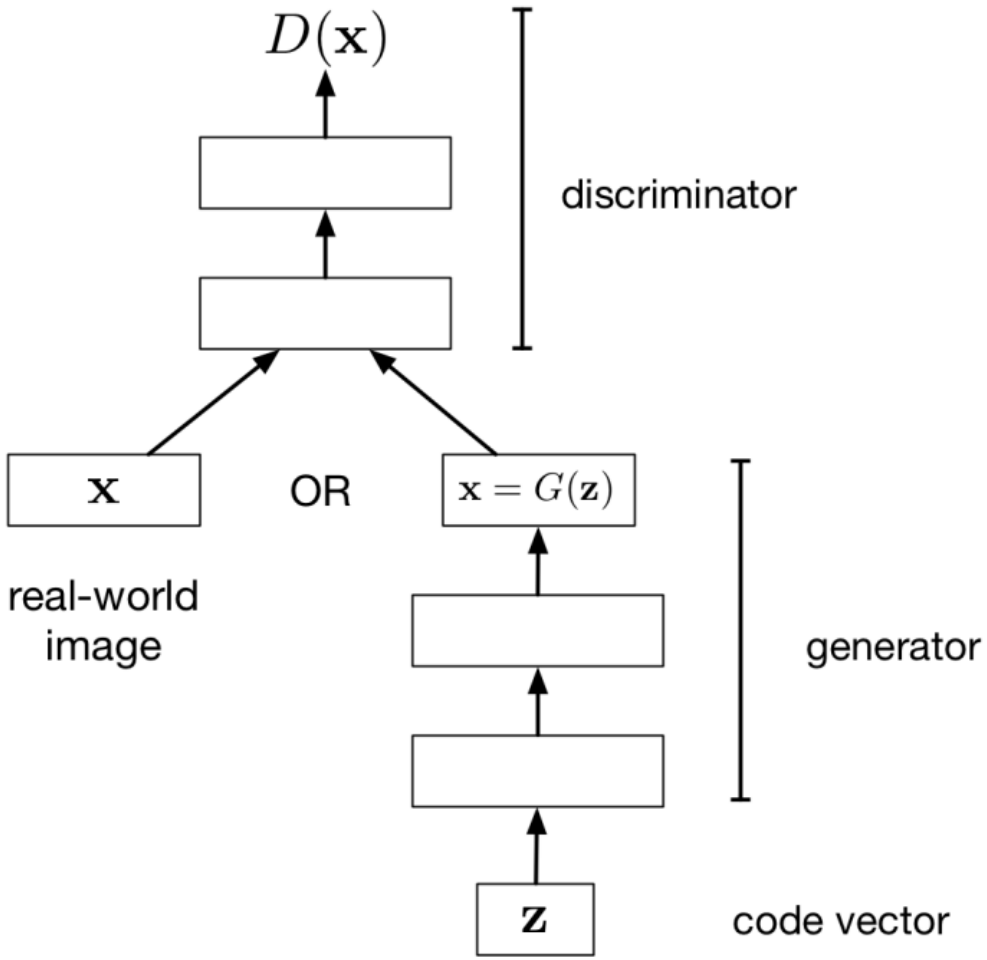
The network outputs an image.

This sort of architecture sounded preposterous to many of us, but amazingly, it works.

Generative Adversarial Networks

- The advantage of implicit generative models: if you have some criterion for evaluating the quality of samples, then you can compute its gradient with respect to the network parameters, and update the network's parameters to make the sample a little better
- The idea behind **Generative Adversarial Networks (GANs)**: train two different networks:
  - The **generator network** tries to produce realistic-looking samples
  - The **discriminator network** tries to figure out whether an image came from the training set or the generator network
- The generator network tries to fool the discriminator network

Generative Adversarial Networks



Generative Adversarial Networks

- Let D denote the discriminator's predicted probability of being data
- Discriminator's cost function: cross-entropy loss for task of classifying real vs. fake images

$$\mathcal{J}_D = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[-\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}}[-\log(1 - D(G(\mathbf{z})))]$$

- One possible cost function for the generator: the opposite of the discriminator's

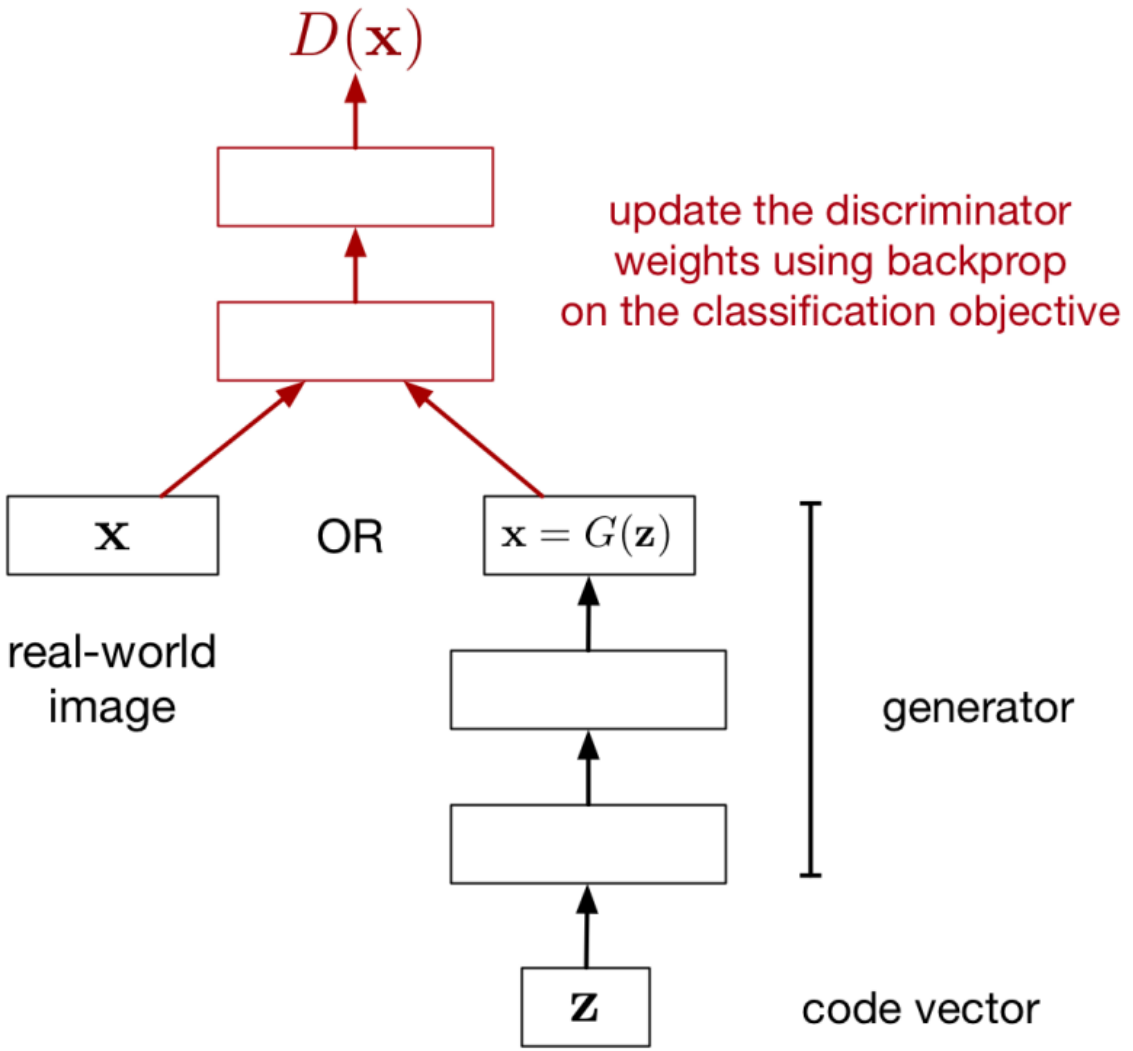
$$\begin{aligned}\mathcal{J}_G &= -\mathcal{J}_D \\ &= \text{ const } + \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]\end{aligned}$$

- This is called the **minimax formulation**, since the generator and discriminator are playing a **zero-sum game** against each other:

$$\max_G \min_D \mathcal{J}_D$$

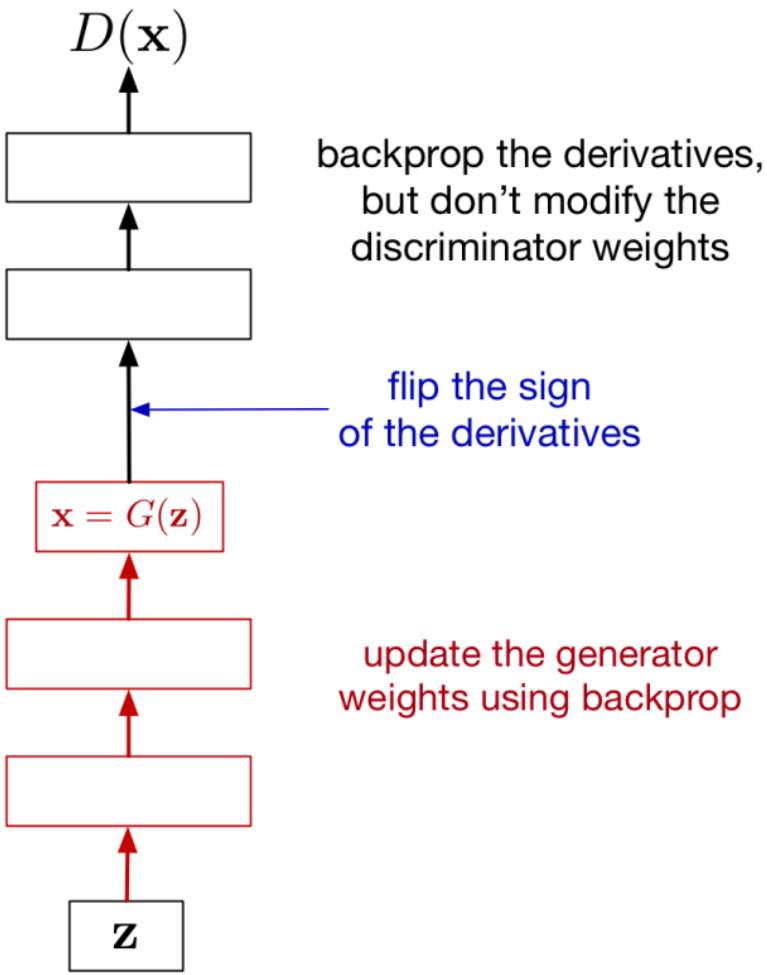
Generative Adversarial Networks

Updating the discriminator:



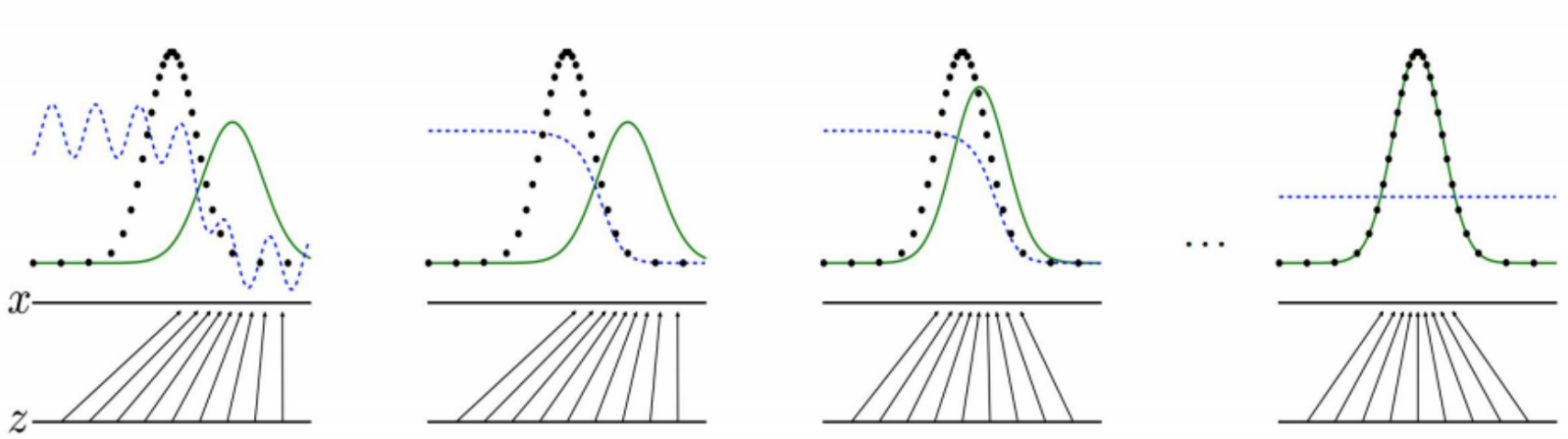
Generative Adversarial Networks

Updating the generator:



Generative Adversarial Networks

Alternating training of the generator and discriminator:



A Better Cost Function

- We introduced the minimax cost function for the generator:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

- One problem with this is **saturation**.
- If the generated sample is really bad, the discriminator's prediction is close to 0, and the generator's cost is flat.

A Better Cost Function

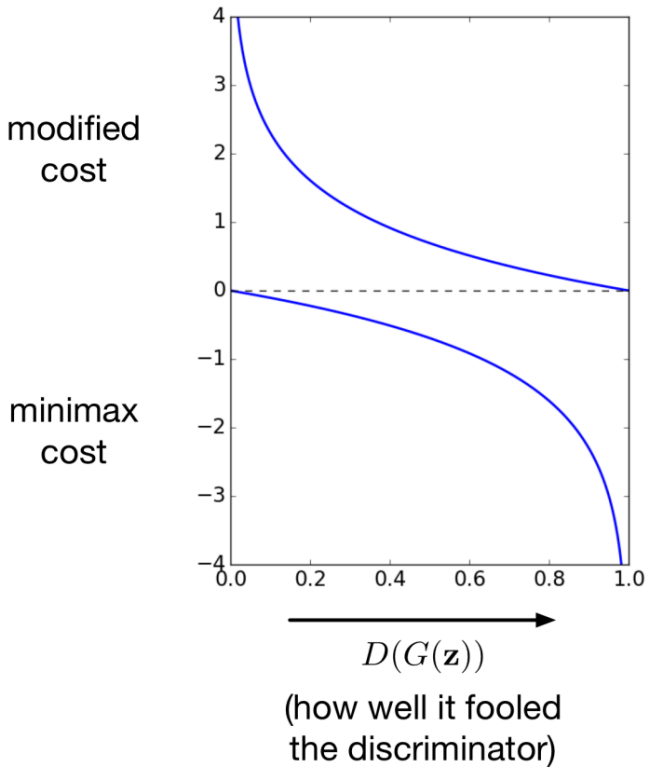
- Original minimax cost:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

- Modified generator cost:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[-\log D(G(\mathbf{z}))]$$

- This fixes the saturation problem.



Generative Adversarial Networks

- Since GANs were introduced in 2014, there have been hundreds of papers introducing various architectures and training methods.
- Most modern architectures are based on the Deep Convolutional GAN (DC-GAN), where the generator and discriminator are both conv nets.
- GAN Zoo: `https://github.com/hindupuravinash/the-gan-zoo`

GAN Samples

Celebrities:

GAN Samples

Bedrooms:

GAN Samples

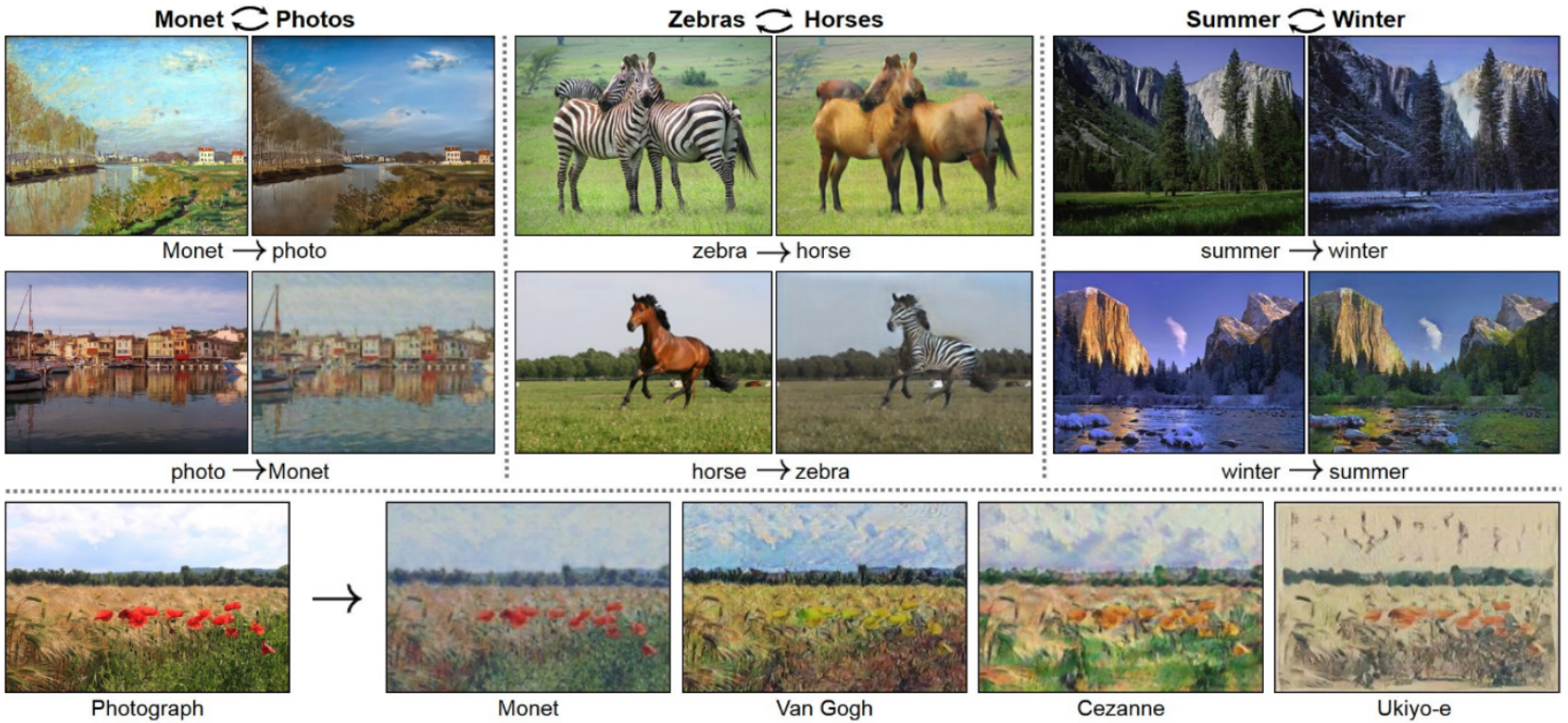
Objects:

GAN Samples

- GANs revolutionized generative modeling by producing crisp, high-resolution images.
- The catch: we don't know how well they're modeling the distribution.
  - Can't measure the log-likelihood they assign to held-out data.
  - Could they be memorizing training examples? (E.g., maybe they sometimes produce photos of real celebrities?)
  - We have no way to tell if they are dropping important modes from the distribution.

Variant: CycleGAN

Style transfer problem: change the style of an image while preserving the content.

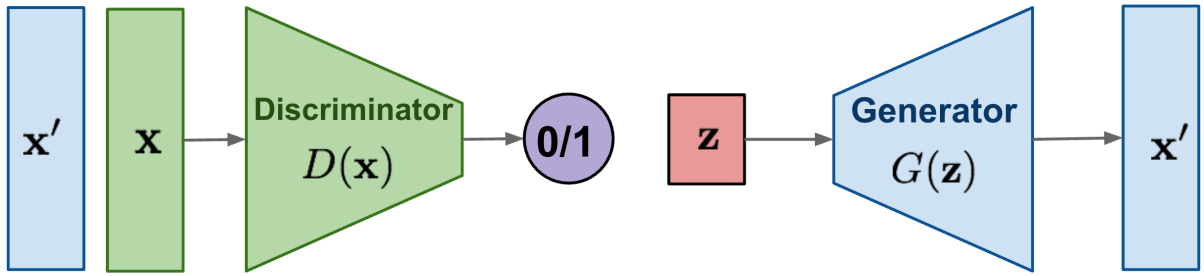


Data: Two unrelated collections of images, one for each style

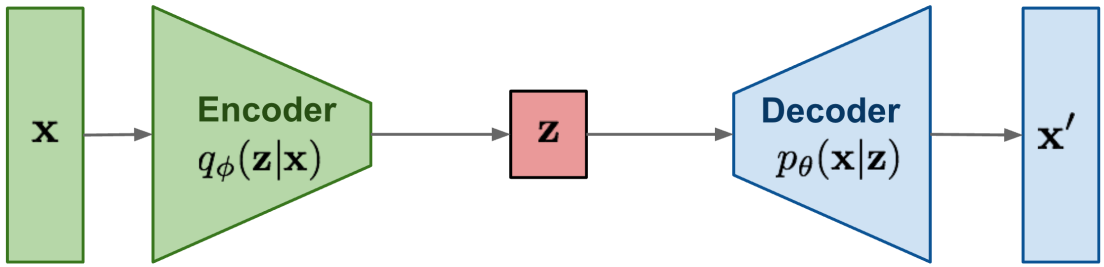
Plan

Generative models

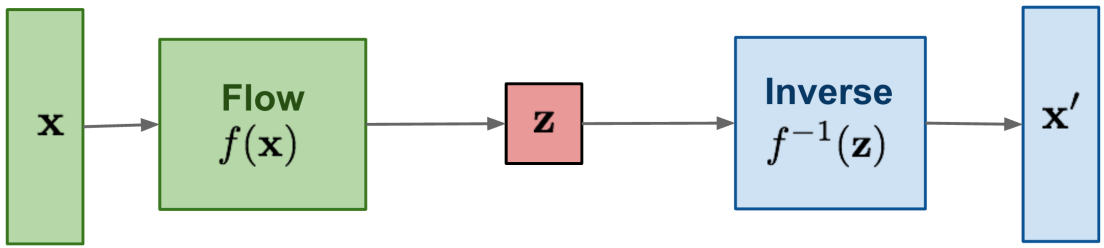
GAN: Adversarial training



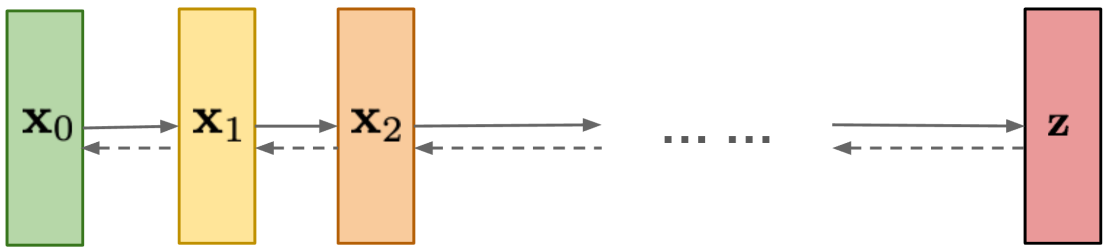
VAE: maximize variational lower bound



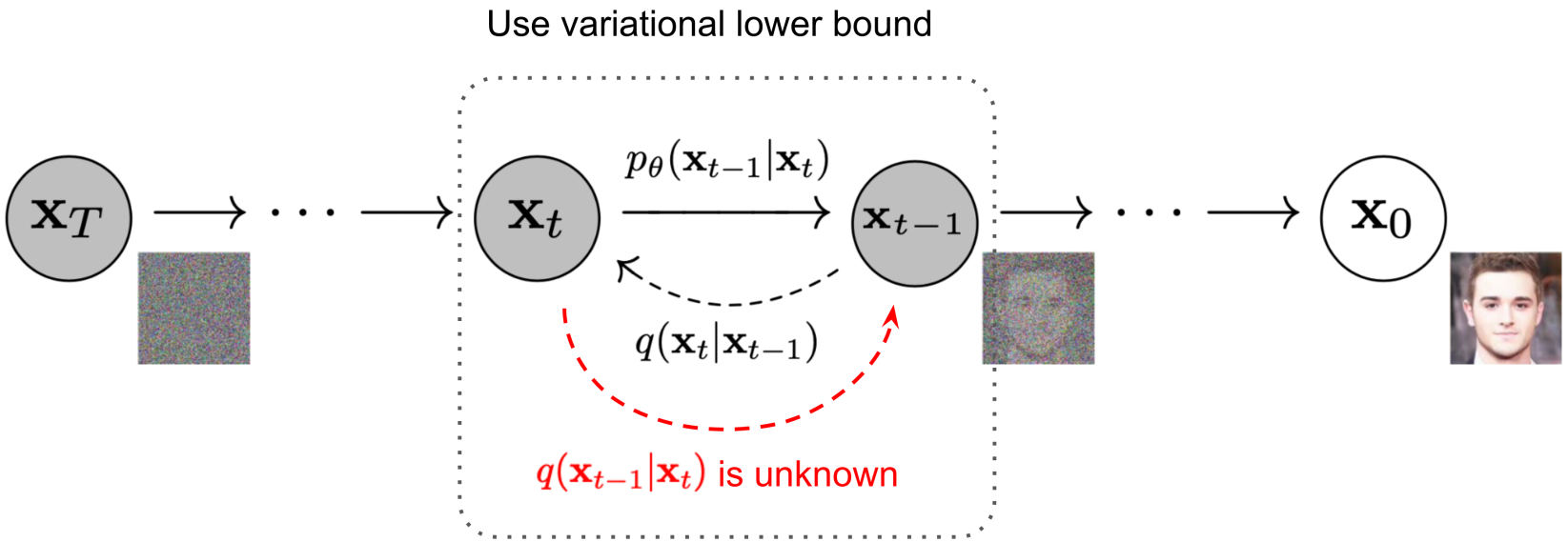
Flow-based models: Invertible transform of distributions



Diffusion models: Gradually add Gaussian noise and then reverse



Diffusion process



$$q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)=\mathcal{N}\left(\mathbf{x}_t ; \sqrt{1-\beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}\right) \quad q\left(\mathbf{x}_{1: T} \mid \mathbf{x}_0\right)=\prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-}\right.$$

This principle lead to a lot of new high quality generative models. (latest: Stable diffusion).

Next course:

- **Autoregressive models**
  - Recurrent neural networks: RNN, LSTM...
  - Transformers