CS60010: Deep Learning
Spring 2023

Sudeshna Sarkar

Generative Modeling
AutoEncoder
VAE
Part 1

22 Mar 2023

Generative Modelling

What is a generative model

- A model for the probability distribution of a data x

- Computational equivalent: a model that can be used to “generate” data with a distribution similar to the given data x

seed $\longrightarrow$  $\longrightarrow x$

Question: how do we generate the random seeds…

Generative vs Discriminative: an analogy

Task: Determine the language that someone is speaking

- Discriminative approach:

- is determine the linguistic differences without learning any language– a much easier task!

- Generative approach:

- is to learn each language and determine as to which language the speech belongs to

[Slide adapted from Srihari]

- From a large collection of face images, can a network learn to generate a new portrait
  - Generate samples from the distribution of “face” images

- From a large collection of landscapes, can a network learn to generate new landscape pictures
  - Generate samples from the distribution of “landscape” images
    - How do we even characterize this distribution?

Generative vs Discriminative

Data: $X = (x, y)$

- Discriminative Model:
  - $p(y|x)$ (conditional distribution)
  - E.g. classification, detection tasks etc.

Conditional Generative Model: $p \;\; x \;\; y)$

E.g. caption -> image, image translation (sketch -> photo)

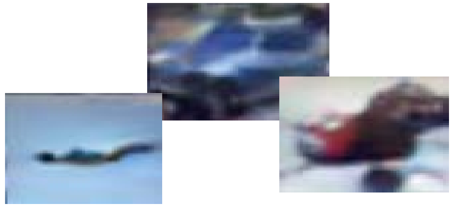
- Generative Model:
  - $p(x, y)$ (joint over data & labels) or $p(x)$ (joint over data)
- Then you can do classification via Bayes Rule:

$$p(y|x) = \frac{p(x, y)}{p(x)}$$

Generative Models

Given training data, generate new samples from same distribution

learning → $p_{model}(x)$ → sampling



Training data ~ $p_{data}(x)$ Generated samples ~ $p_{model}(x)$

1. Learn $p_{model}(x)$ similar to $p_{data}(x)$
2. Sample new x from $p_{model}(x)$

Formulate as density estimation problems

- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it

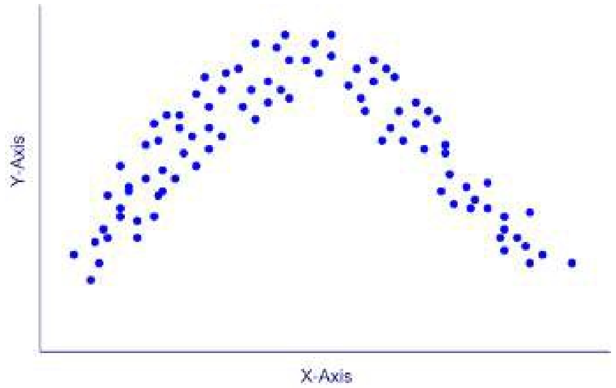
The distribution of data

Hypothesis: The data are distributed about a non-linear manifold in high dimensional space

- To generate data for this class, we must select a point on this manifold

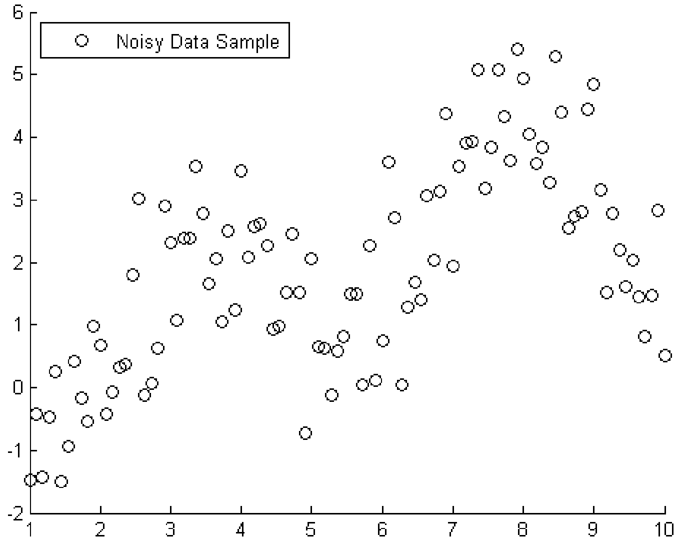
Problems:

- Characterizing the manifold
- Having a good strategy for selecting points from it



- Given some set of observed data $X = \{x\}$.
- Choose a model $P(x;\theta)$ for the distribution of x
  - θ are the parameters of the model

Estimate the θ such that $P(x;\theta)$ best “fits” the observations



Application of Generative Models

- Realistic samples for artwork, super-resolution, colorization, etc
- Learn useful features for downstream tasks such as classification
- Training generative models can also enable inference of latent representations that can be useful as general features
- Modeling physical world for simulation and planning (reinforcement learning applications, robotics)
- Image augmentation
- Natural language generation

Why generative models? Many right answers

Caption -> Image

https://openreview.net/pdf?id=Hyvw0L9el

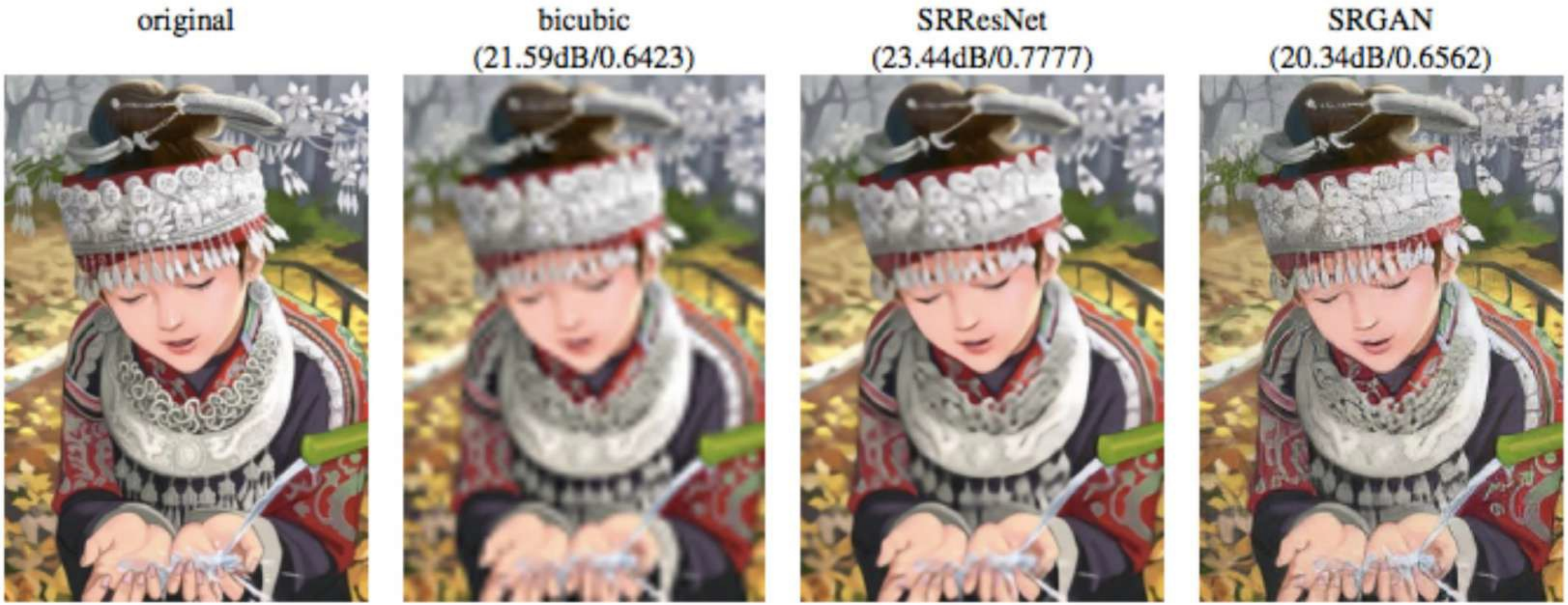
A man in an orange jacket with
sunglasses and a hat skis down a hill

Outline -> Image

https://arxiv.org/abs/1611.07004

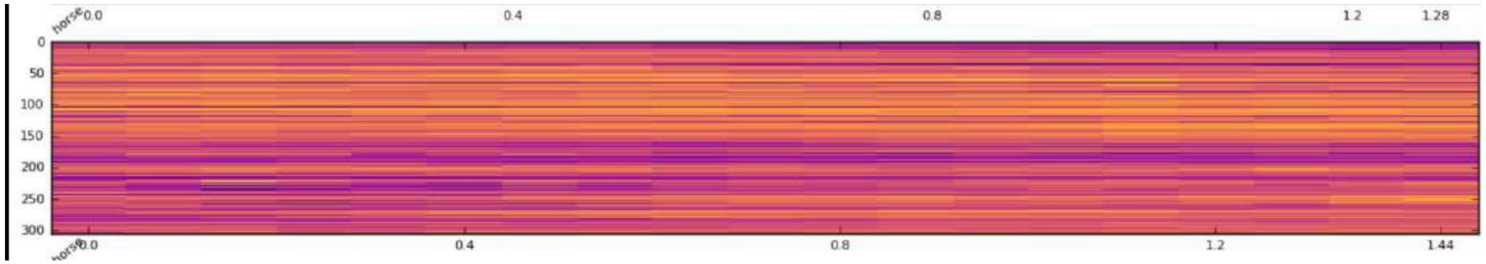
Why generative models? Intrinsic to task

Example: Super resolution

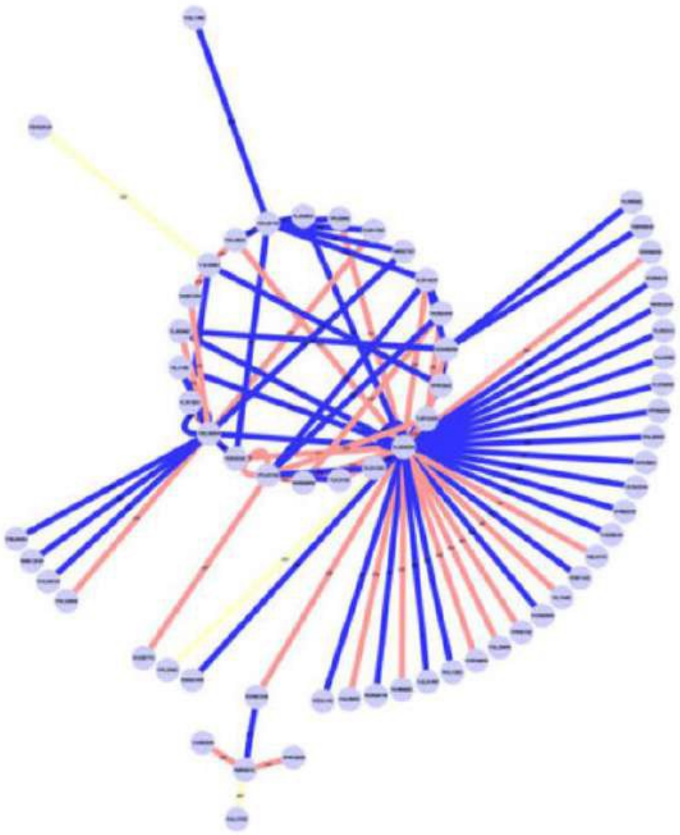


https://arxiv.org/abs/1609.04802

Why generative models? Insight



- What kind of structure can we find in complex observations (MEG recording of brain activity above, gene-expression network to the left)?
- Is there a low dimensional manifold underlying these complex observations?
- What can we learn about the brain, cellular function, etc. if we know more about these manifolds?



https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-12-327

Taxonomy of Generative Models

- **Generative models**
  - Explicit density — Model can compute p(x)
    - Tractable density
      - Can compute p(x)
        - Autoregressive
        - NADE / MADE
        - NICE / RealNVP
        - Glow
        - Ffjord
    - Approximate density — Can compute approximation to p(x)
      - Variational
        - Variational Autoencoder
      - Markov Chain
        - Boltzmann Machine
  - Implicit density — Model does not explicitly compute p(x), but can sample from p(x)
    - Markov Chain
      - GSN
    - Direct
      - Generative Adversarial Networks (GANs)

Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Autoregressive generative models

Main principle for training:

1. Divide up x into dimensions $x_1, \ldots, x_n$
2. Discretize each x_i into k values
3. Model $p(x)$ via the chain rule

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$

4. Use your favorite sequence model to model $p(x)$

Using autoregressive generative models:

Sampling: ancestral sampling in sequence (x_1, then x_2, etc.)

Completion: feed in actual values for known x_i values

Representations: same idea as ELMo or BERT

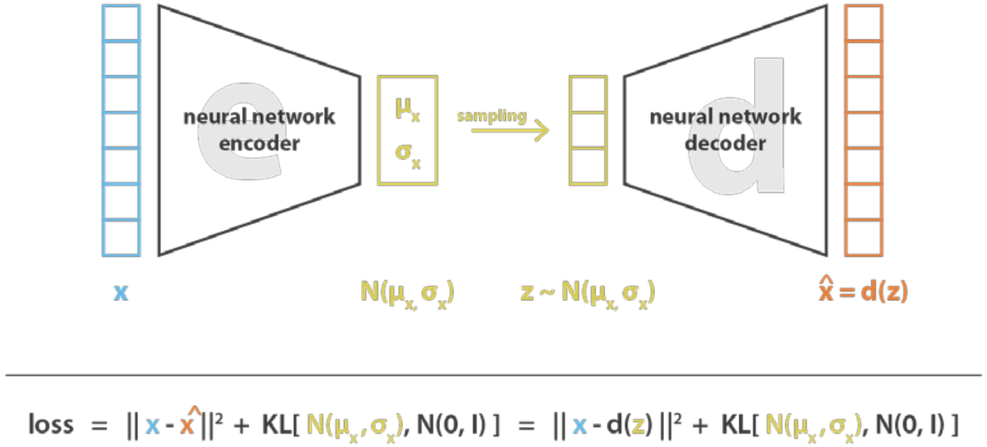
Deep Generative Models

We will introduce two representative deep generative models:

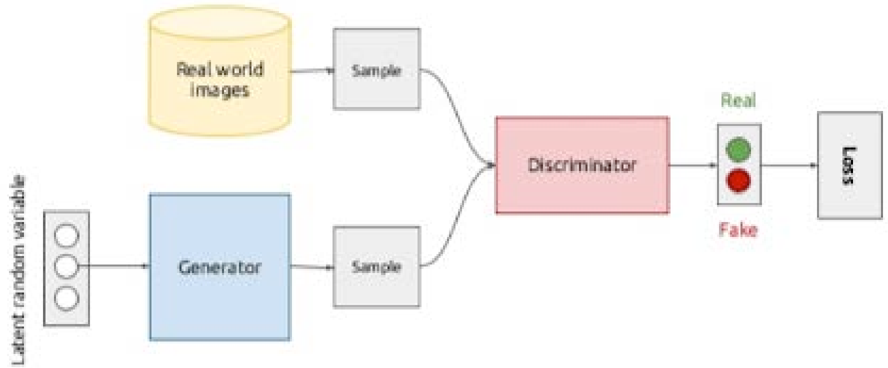
1. **Variational Autoncoder (VAE)** is a variational algorithm that infers the statistical characteristics of latent variables using an autoencoder neural network architecture.
2. **Generative Adversarial Networks (GAN)** train two neural networks (a generative neural network and a discriminative neural network) contesting with each other in a zero-sum game framework.

Latent Variable Models

Autoencoders and VAE



Generative Adversarial Networks



Autoencoder

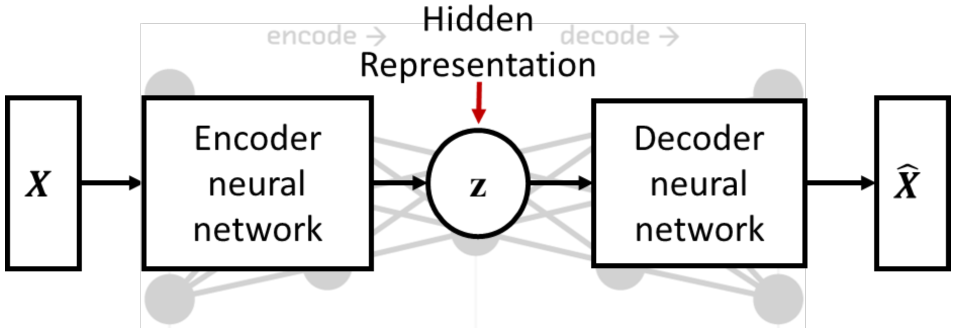
A neural network that reconstructs its own input.

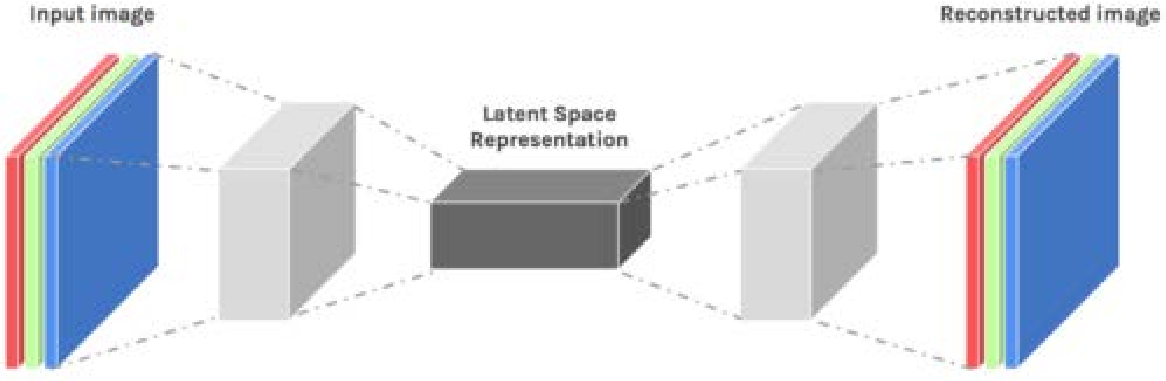
reproduces the input from a learned encoding.

Basic idea:

1. Train a network that **encodes the input** into some **hidden state**

2. decodes that input **as accurately as possible** from that **hidden state**



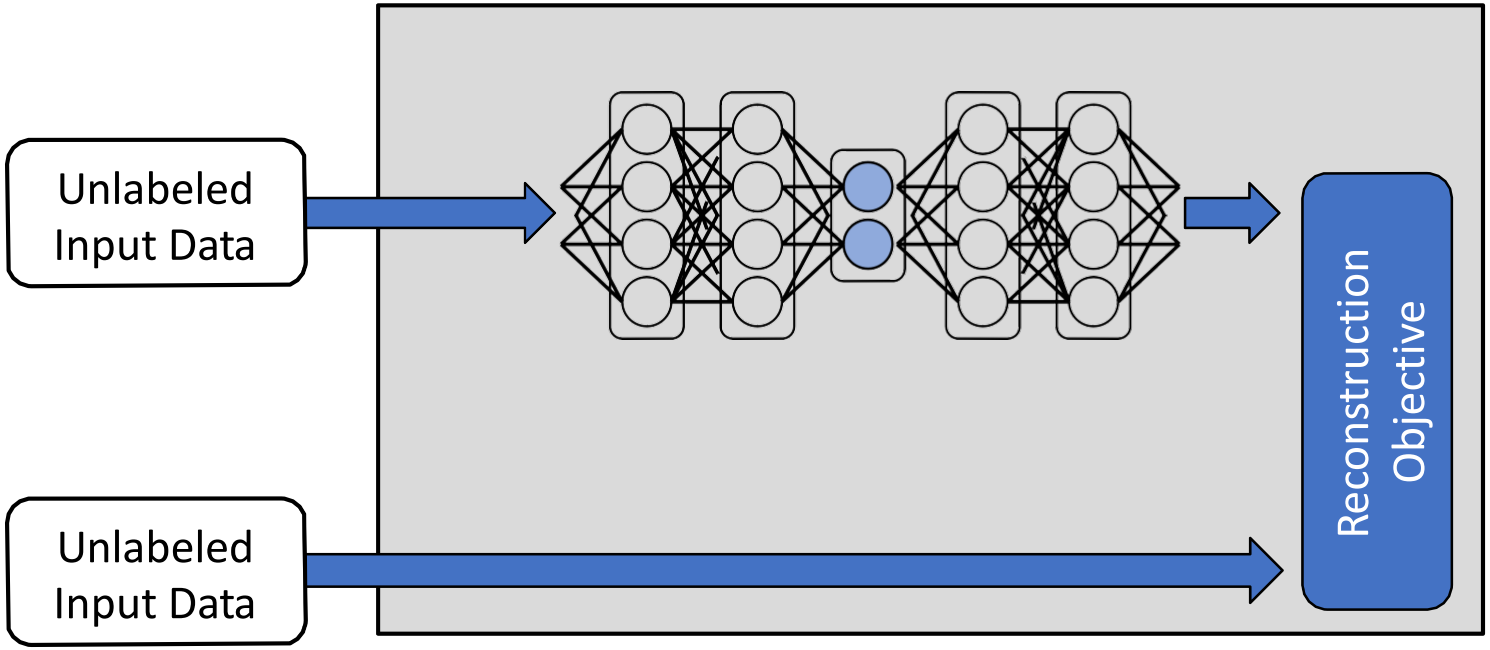


encoder decoder

Hidden state
this is what we use
for downstream tasks

The autoencoder captures the underlying manifold of the data

Autoencoders



Trivial solution unless:
-Constrain number of units in Layer 2 (learn compressed representation), or
- Constrain Layer 2 to be **sparse**.

$$\min_{\theta} \underbrace{\|h_\theta(x) - x\|^2}_{\text{Reconstruction error term}} + \lambda \underbrace{\sum_i |a_i|}_{L_1 \text{ sparsity term}}$$

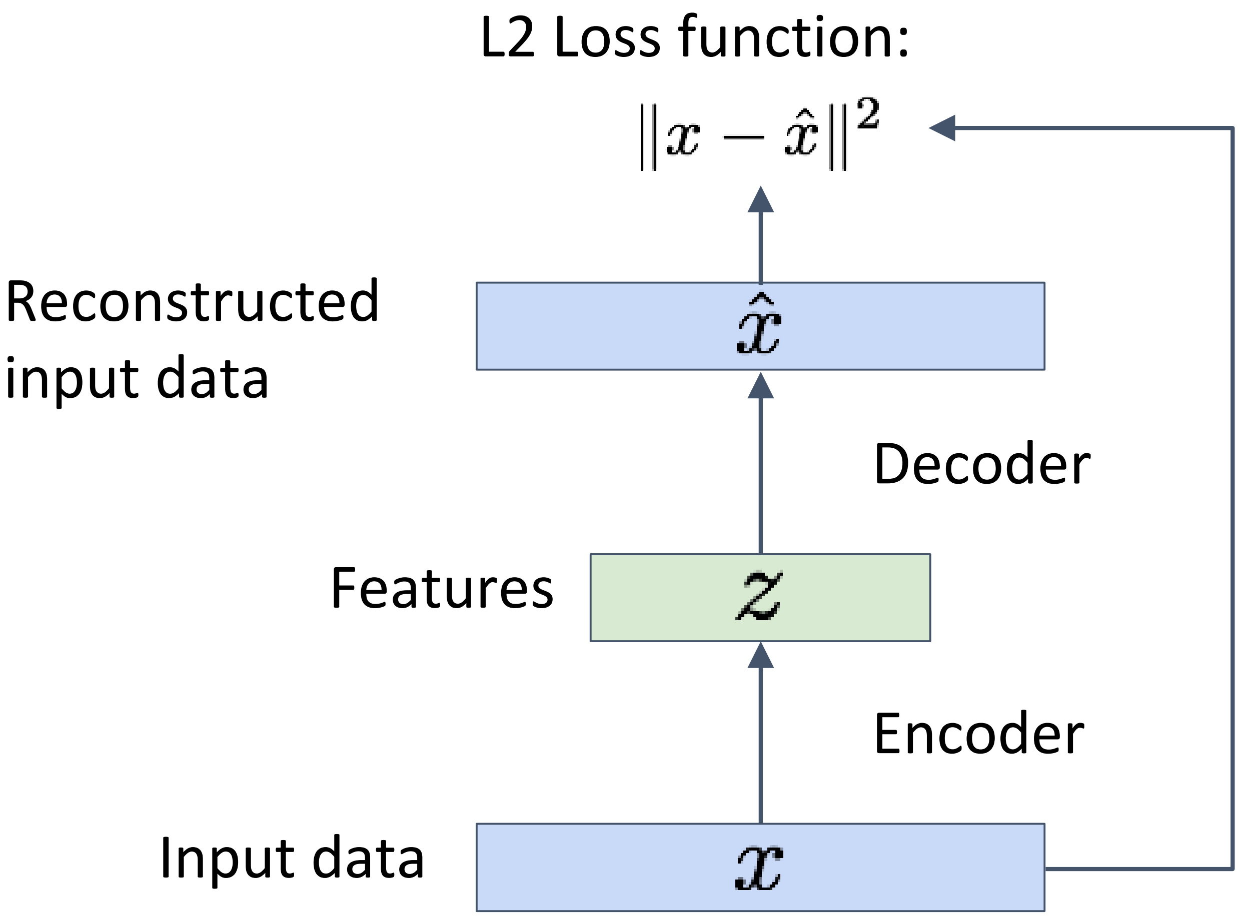
[Hinton & Salakhutdinov, Science 2006]

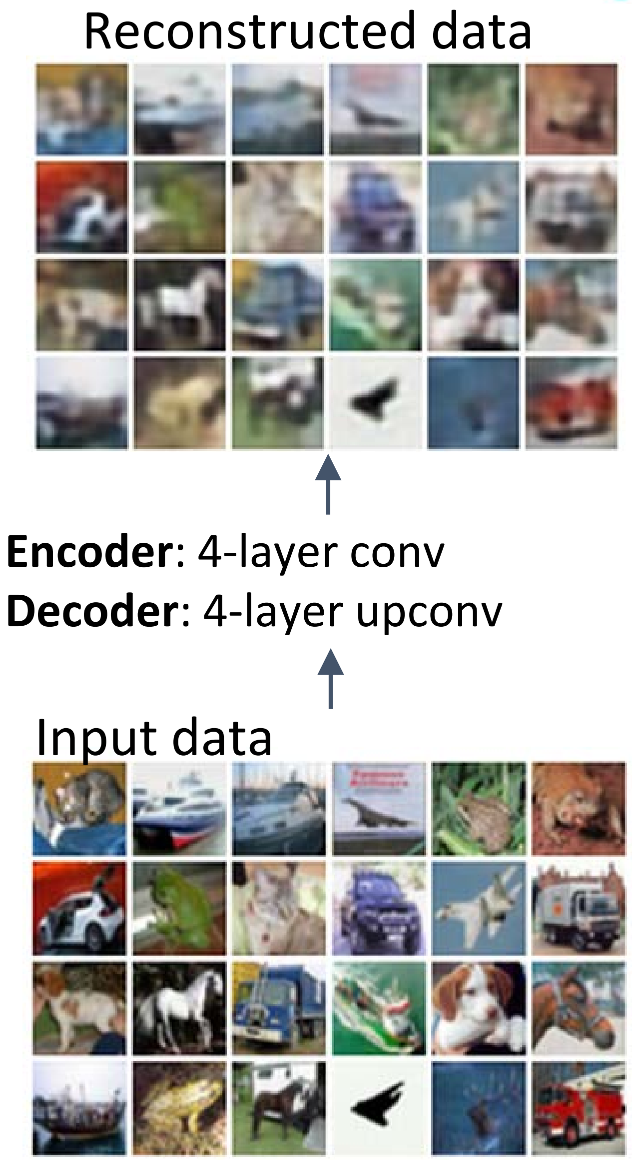
Forcing structure in Autoencoder

Forcing structure: something about the design of the model, or in the data processing or regularization, must force the autoencoder to learn a **structured** representation

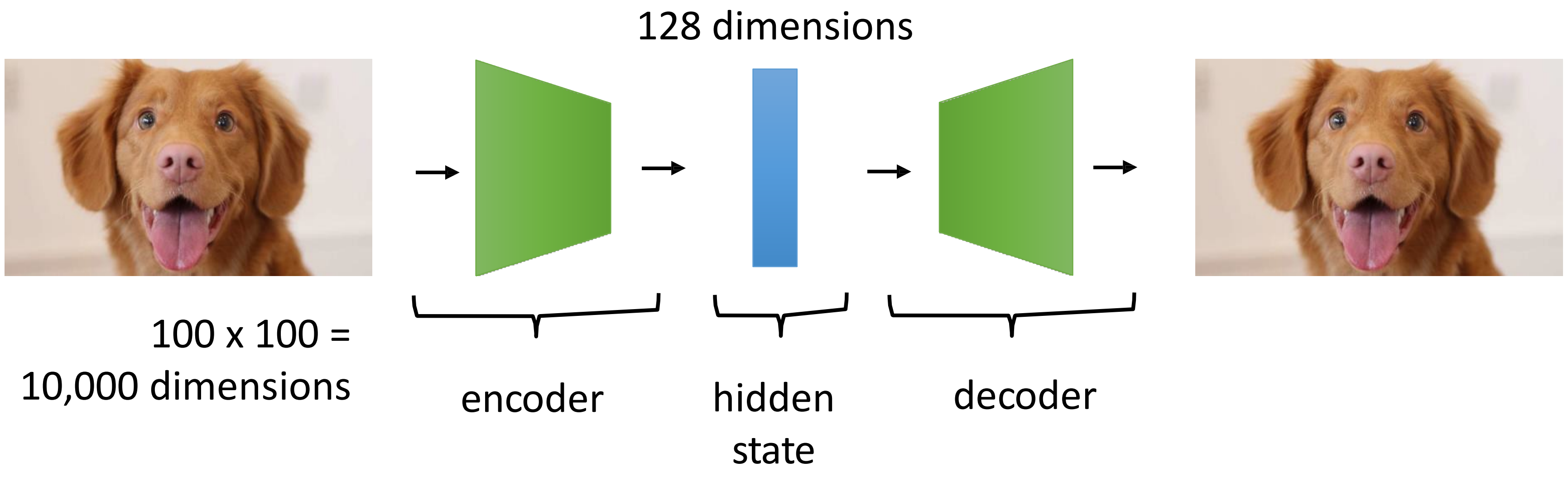
- **Dimensionality**: make the hidden state smaller than the input/output, so that the network must compress it
- **Sparsity**: force the hidden state to be sparse (most entries are zero), so that the network must compress the input
- **Denoising**: corrupt the input with noise, forcing the autoencoder to learn to distinguish noise from signal
- **Probabilistic modeling**: force the hidden state to agree with a prior distribution

Autoencoders





Bottleneck autoencoder



- This has some interesting properties:
  - If both encoder and decoder are linear, this exactly recovers PCA
  - Can be viewed as “non-linear dimensionality reduction” – could be useful simply because dimensionality is lower and we can use various algorithms that are only tractable in low-dimensional spaces (e.g., discretization)

Sparse autoencoder

Idea: can we describe the input with a small set of "attributes"?

This might be a more **compressed** and **structured** representation

Pixel (0,0): #FE057D

Pixel (0,1): #FD0263

Pixel (0,2): #E1065F

"dense representation": most values non-zero
Not structured

Idea: "sparse" representations are going to be more structured!

has_ears: 1

has_wings: 0

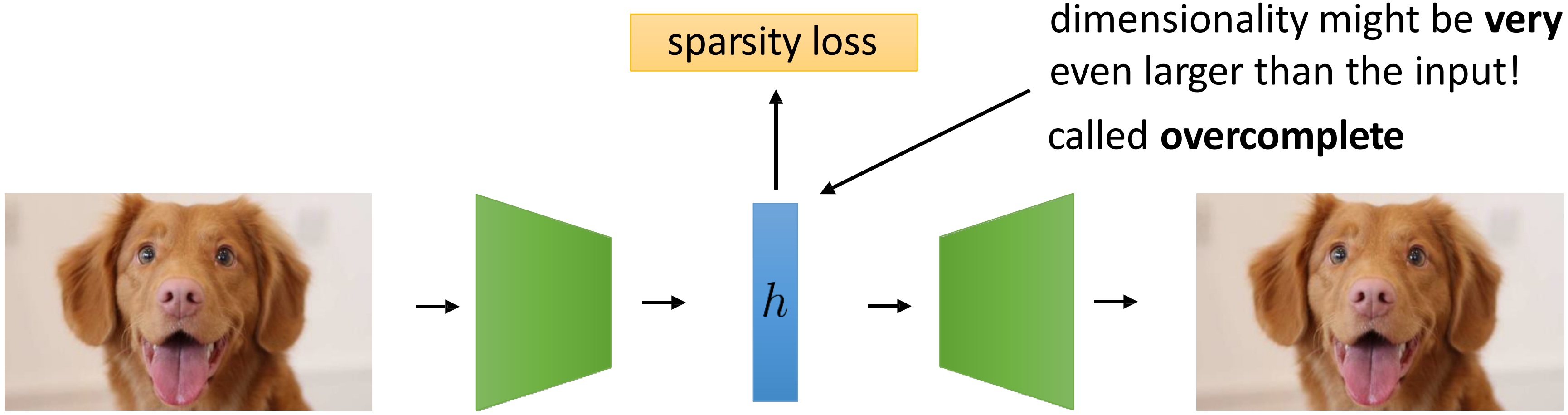
has_wheels: 0

very structured!

"sparse": most values are zero

there are many possible "attributes," and most images don't have most of the attributes

Sparse autoencoder



dimensionality might be **very** large, even larger than the input!

called **overcomplete**

encoder

hidden state

decoder

ℓ_1 norm of h

gradient is always $\mathrm{sgn}(h_j)$

simple sparsity loss: $\sum_{j=1}^{D} |h_j|$

Sparse Autoencoder

- Regularize outputs of hidden layer to enforce sparsity:

$$\tilde{J}(x) = J(x, g(f(x))) + \alpha\, \Omega(h)$$

J : loss function, f : encoder, g : decoder, h=f(x), Ω penalizes non-sparsity of h

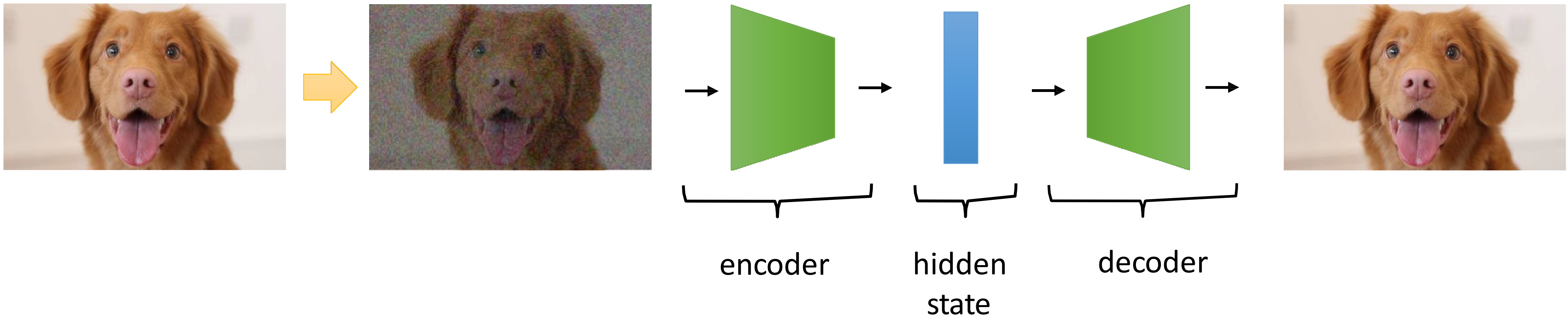
- E.g., can use $\Omega(h) = \sum_i |h_i|$ and ReLU activation to force many zero outputs in hidden layer
- Can also measure average activation of h_i across mini-batch and compare it to user-specified **target sparsity** value ρ (e.g., 0.1) via square error or **Kullback-Leibler divergence**:

$$p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q}$$

q is average activation of h_i over mini-batch

Denoising autoencoder

Idea: a good model that has learned meaningful structure should "fill in the blanks"

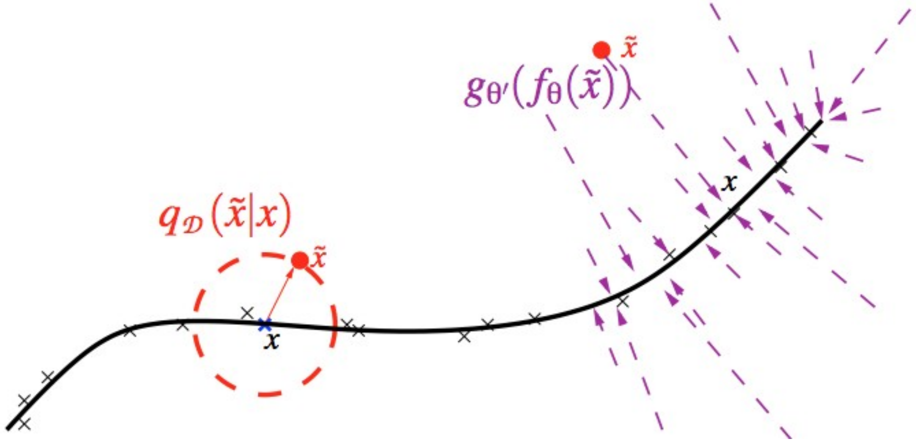


Can train an autoencoder to learn to **denoise** input by giving input **corrupted** instance $\tilde{\boldsymbol{x}}$ and targeting **uncorrupted** instance $\boldsymbol{x}$

There are **many variants** on this basic idea, and this is one of the most widely used simple autoencoder designs

Denoising Autoencoder

- How does it work?
- Even though, e.g., MNIST data are in a 784-dimensional space, they lie on a low-dimensional **manifold** that captures their most important featur
- Corruption process moves instance ***x*** off of manifol
- Encoder f_θ and decoder $g_{\theta\prime}$ are trained to project
back onto manifold



The types of autoencoders: Forcing Structure

1. Dimensionality: make the **hidden state** smaller than the **input/output**, so that the network must **compress** it

+ **very simple to implement**

- **simply reducing dimensionality often does not provide the structure we want**

2. Sparsity: force the **hidden state** to be sparse (most entries are 0), so that the network must **compress** input

+ **principled approach that can provide a "disentangled" representation**

- **harder in practice, requires choosing the regularizer and adjusting hyperparameters**

3. Denoising: corrupt the **input** with **noise**, forcing the autoencoder to learn to distinguish **noise from signal**

+ **very simple to implement**

- **not clear which layer to choose for the bottleneck, adhoc choicers (e.g., how much noise to add)**

4. Probabilistic modeling: force the **hidden state** to agree with a **prior distribution**

Autoencoders

Not probabilistic: No way to sample new data from learned model