EURECOM
Sophia Antipolis

MALCOM

Machine Learning for Communication Systems

Lecture 4
Autoencoders

Slides: Marios Kountouris
Lecturer: Omid Esrafilian

Spring 2024

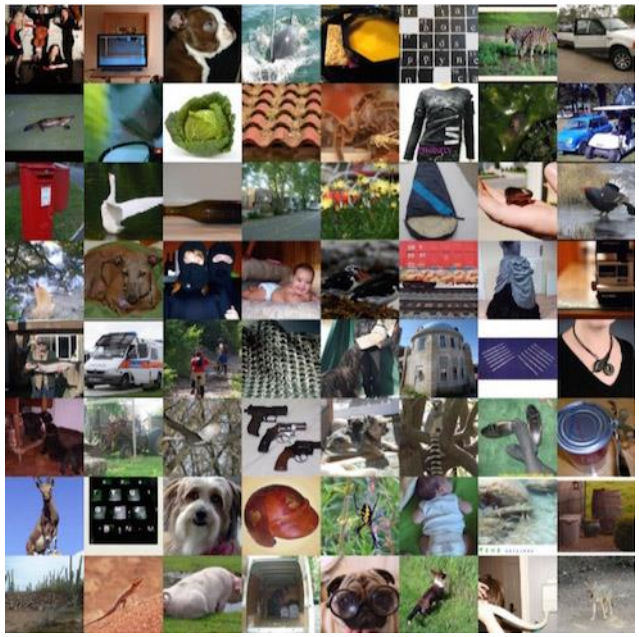
Generative Models

Given training data, how to generate new samples from the same distribution

1) Collect a large amount of data (e.g., millions of images, sentences, or sounds, etc.)

2) Train a model to generate data like it.

- Intuition: *"Whatever I cannot create, I do not understand"* (*R. Feynman*)
- Key trick: # NN parameters << amount of data we train them on ⇒ forcing to discover and efficiently internalize the essence of the data in order to generate it.

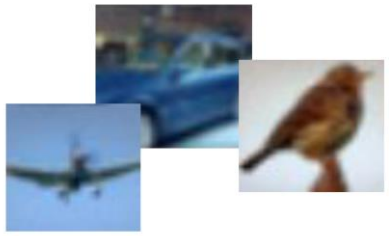


Real Images (ImageNet)

Generated Images

https://openai.com/blog/generative-models/

Taxonomy of Generative Models



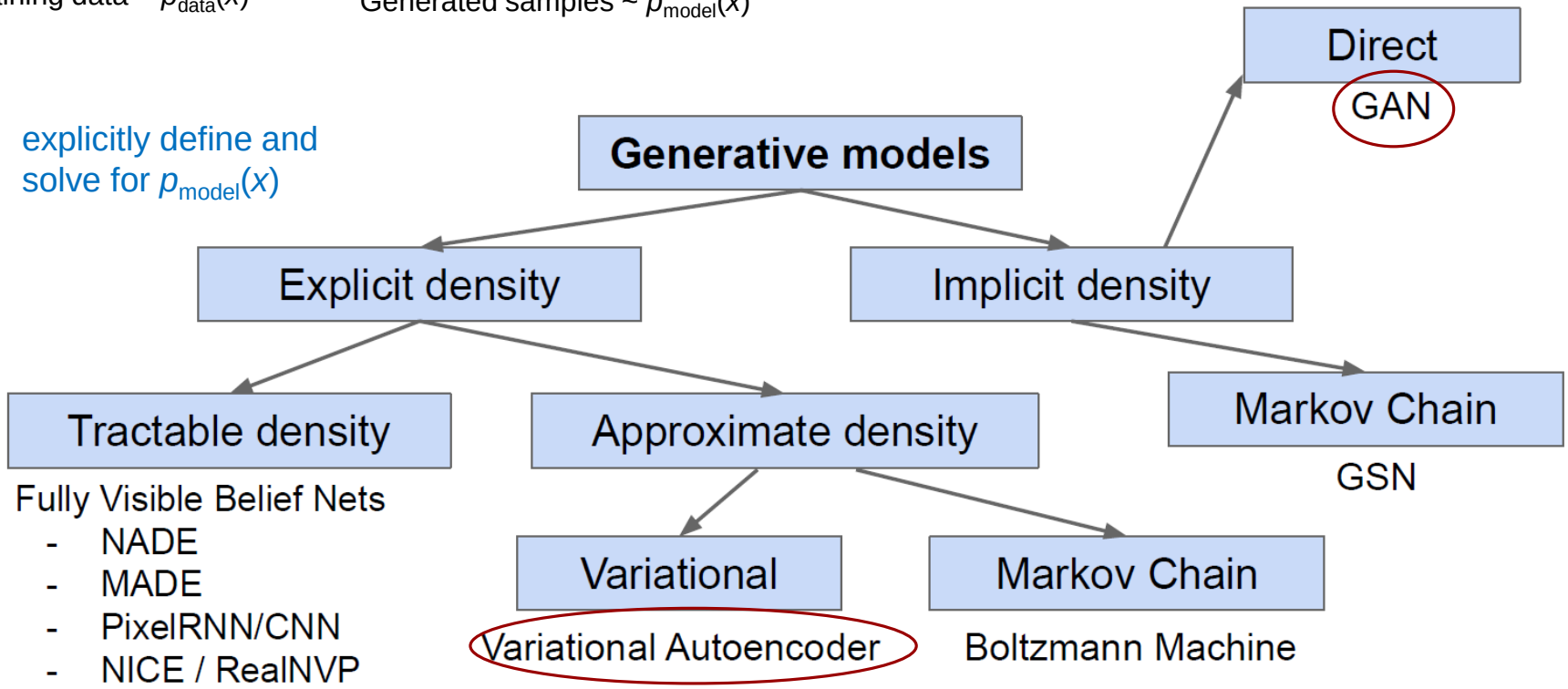
Training data ~ $p_{data}(x)$

vs

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

learn model that can draw samples from $p_{model}(x)$ without explicitly defining it

explicitly define and solve for $p_{model}(x)$



Goodfellow, NIPS 2016
arxiv.org/abs/1701.00160

EURECOM
Sophia Antipolis

Approaches to Generative Models

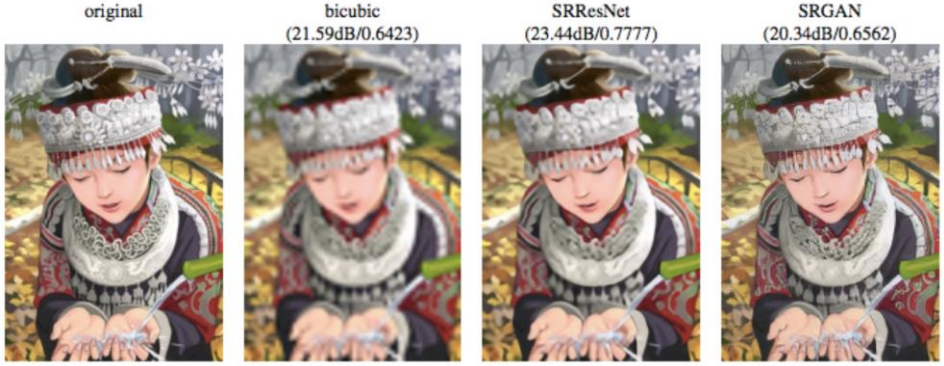
- **Variational Autoencoders (VAEs)**: probabilistic graphical models – maximize a lower bound on the log likelihood of the data.
  - Perform both learning and efficient Bayesian inference in sophisticated probabilistic graphical models with latent variables
  - generated samples tend to be slightly blurry.

- **Generative Adversarial Networks (GANs)**: training is a game between two separate networks: a generator network and a second discriminative network that tries to classify samples as either coming from the true distribution or the model distribution.
  - generate the sharpest images
  - more difficult to optimize due to unstable training dynamics (Nash equilibrium vs. objective function optimization).

- **Autoregressive models** (e.g., PixelRNN): train a network that models the conditional distribution of every individual pixel given previous pixels
  - very simple and stable training process (softmax loss)
  - give the best log likelihoods (i.e., plausibility of the generated data).
  - relatively inefficient during sampling and don't easily provide simple low-dimensional codes for images.

Applications

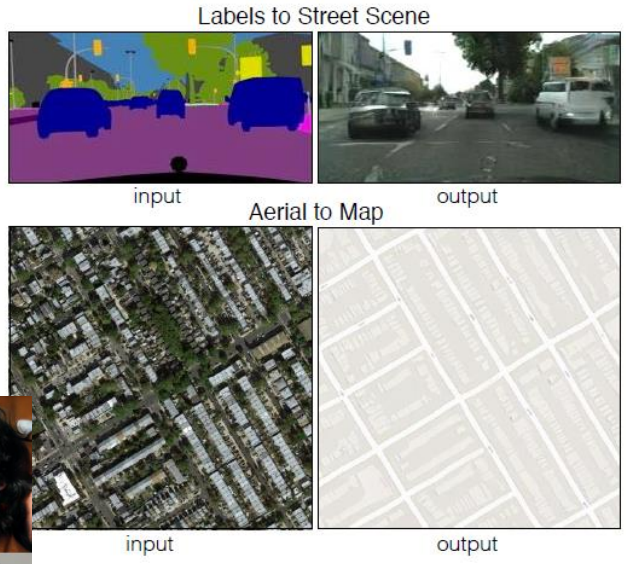
Single image Super-resolution



(Ledig et al. 2016)

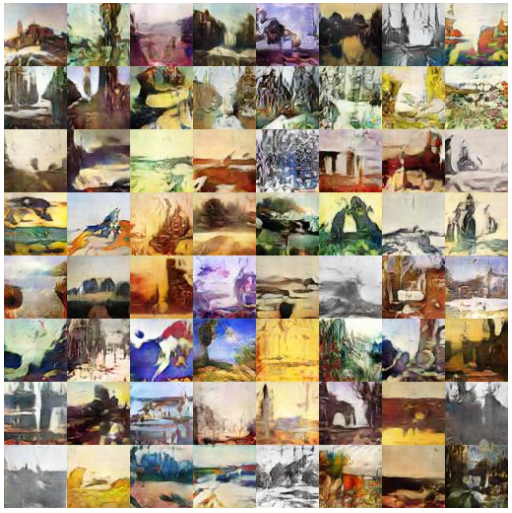
Image denoising, photo inpainting, photo blending, structured prediction, semantic-Image-to-Photo translation, exploration in RL, NN pretraining….

Image to image translation



(Isola et al. 2016)

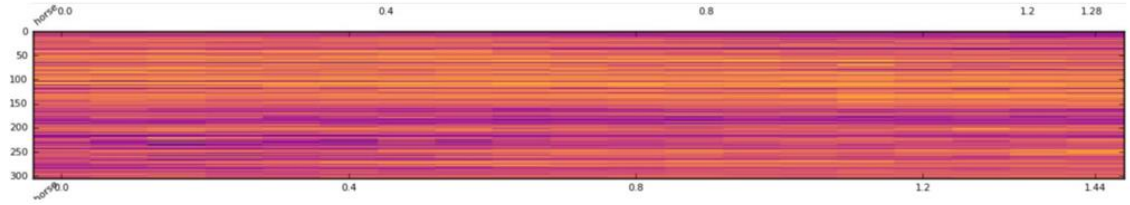
Creating Art (GANGogh)



Human Face Generation (StyleGAN)

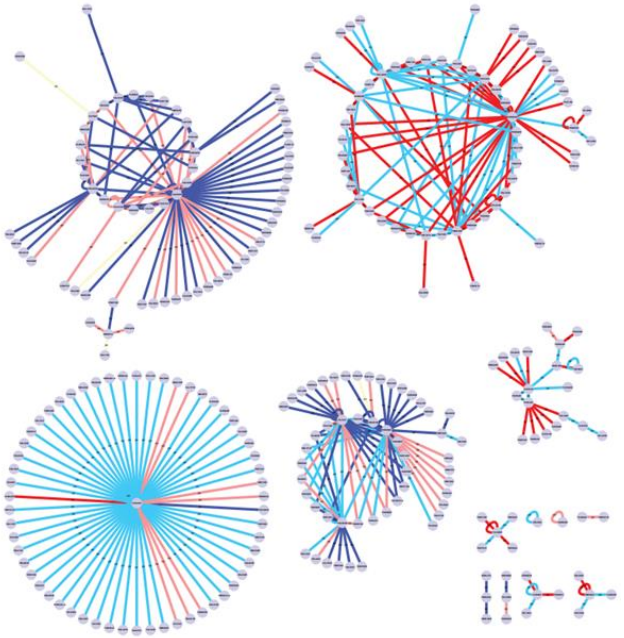
EURECOM
Sophia Antipolis

Generative Models: insights



MEG recording of brain activity

- What kind of structure can we find in complex observations?
- 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?



Gene-expression network

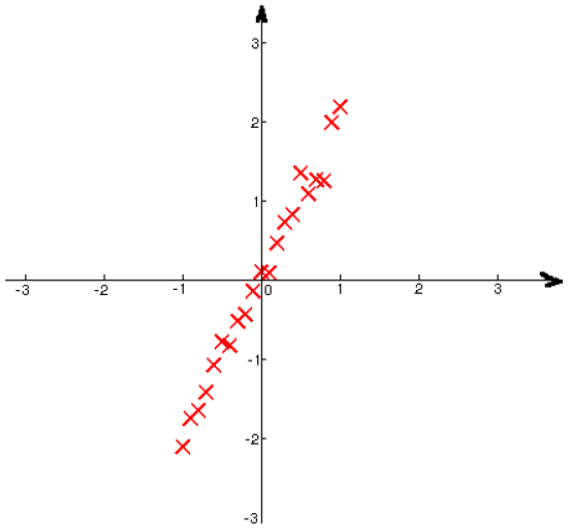
https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-12-327

Autoencoders

Autoencoders – Prelude

Data compression:

- I want to send some data from my device to the cloud (or edge)
- Data is a collection of data points, each has two dimensions.



Visualization of my data

- Value of the 2nd dimension is approximately twice as much as that of the 1st dimension.

Idea:

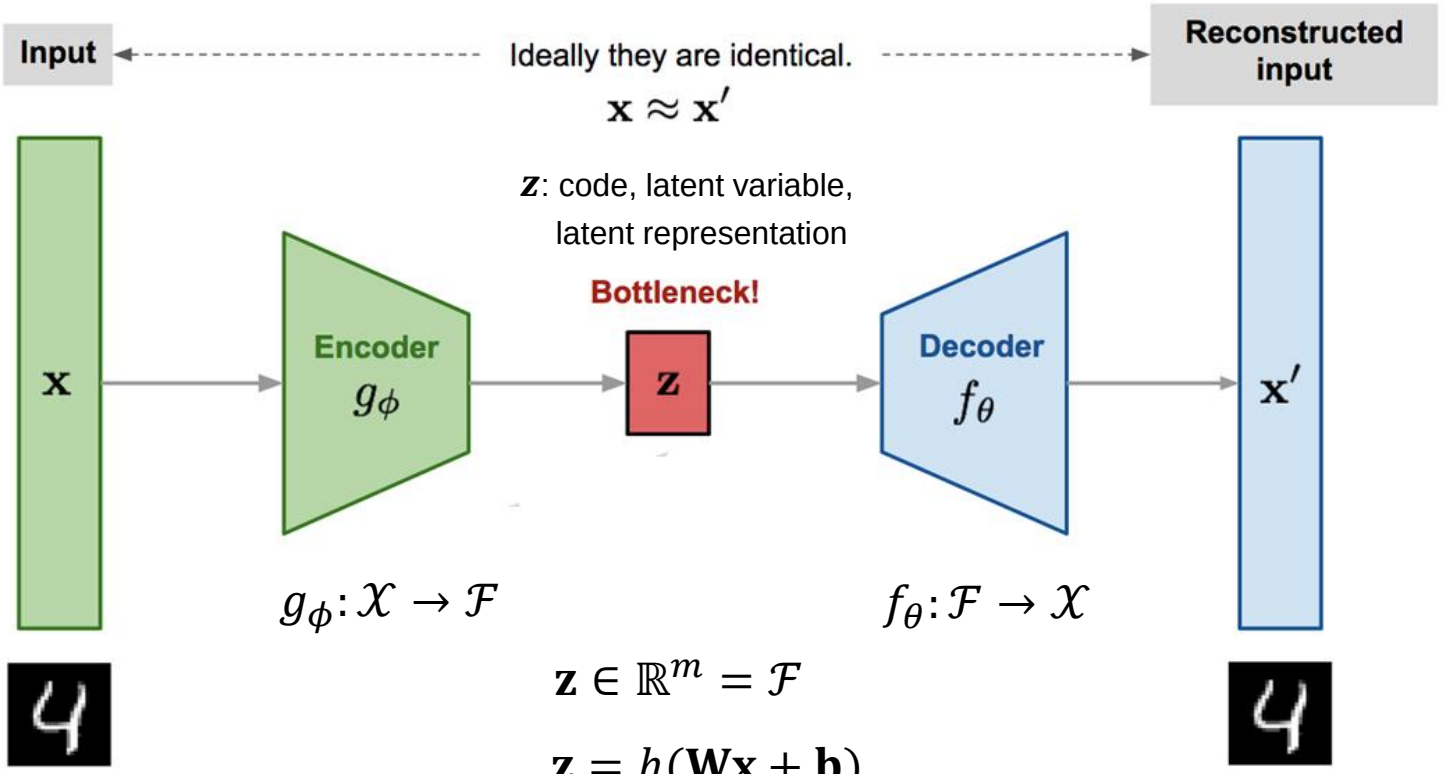
- send only the 1st dimension of every data point to the cloud.
- @Cloud: compute the value of the 2nd dimension by doubling the value of the 1st dimension.

Lossy compression + extra computation **but** 50% network traffic reduction

- High-dimensional data: set of data points $\mathbf{x} = \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ where each data point has many dimensions.
- **Goal:** find a general way to map them to another set of data points $\mathbf{z} = \{z^{(1)}, z^{(2)}, \dots, z^{(m)}\}$, where z's have lower dimensionality than x's & z's can faithfully **reconstruct** x's.

- Approach:
  - 1. Encoding (@device): map data $x^{(i)}$ to compressed data $z^{(i)}$
  - 2. Transmission: send $z^{(i)}$ to the cloud.
  - 3. Decoding (@cloud): map from compressed data $z^{(i)}$ back to $\tilde{x}^{(i)}$, which approximates the original data.

Autoencoders



$g_\phi : \mathcal{X} \to \mathcal{F}$

$f_\theta : \mathcal{F} \to \mathcal{X}$

$\mathbf{z} \in \mathbb{R}^m = \mathcal{F}$

$\mathbf{z} = h(\mathbf{W}\mathbf{x} + \mathbf{b})$

$\phi = \{\mathbf{W}, \mathbf{b}\}$

$\mathbf{x} \in \mathbb{R}^d = \mathcal{X}$

$\mathbf{x}' \in \mathbb{R}^d = \mathcal{X}$

$\mathbf{x}' = s(\mathbf{W}'\mathbf{z} + \mathbf{b}')$

$\theta = \{\mathbf{W}', \mathbf{b}'\}$

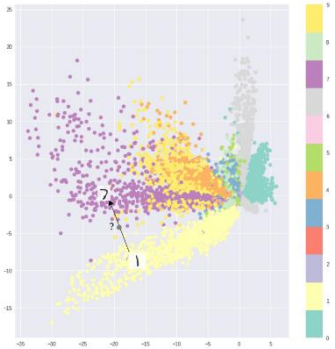
$$g_\phi, f_\theta = \arg\min_{g_\phi, f_\theta} \mathbb{E}_{p_\mathbf{x}}\left[\mathcal{L}(\mathbf{x}, (f_\theta \circ g_\phi)\mathbf{x}\right]$$

Reconstrcution loss function (usually set to ℓ_2 norm)

https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

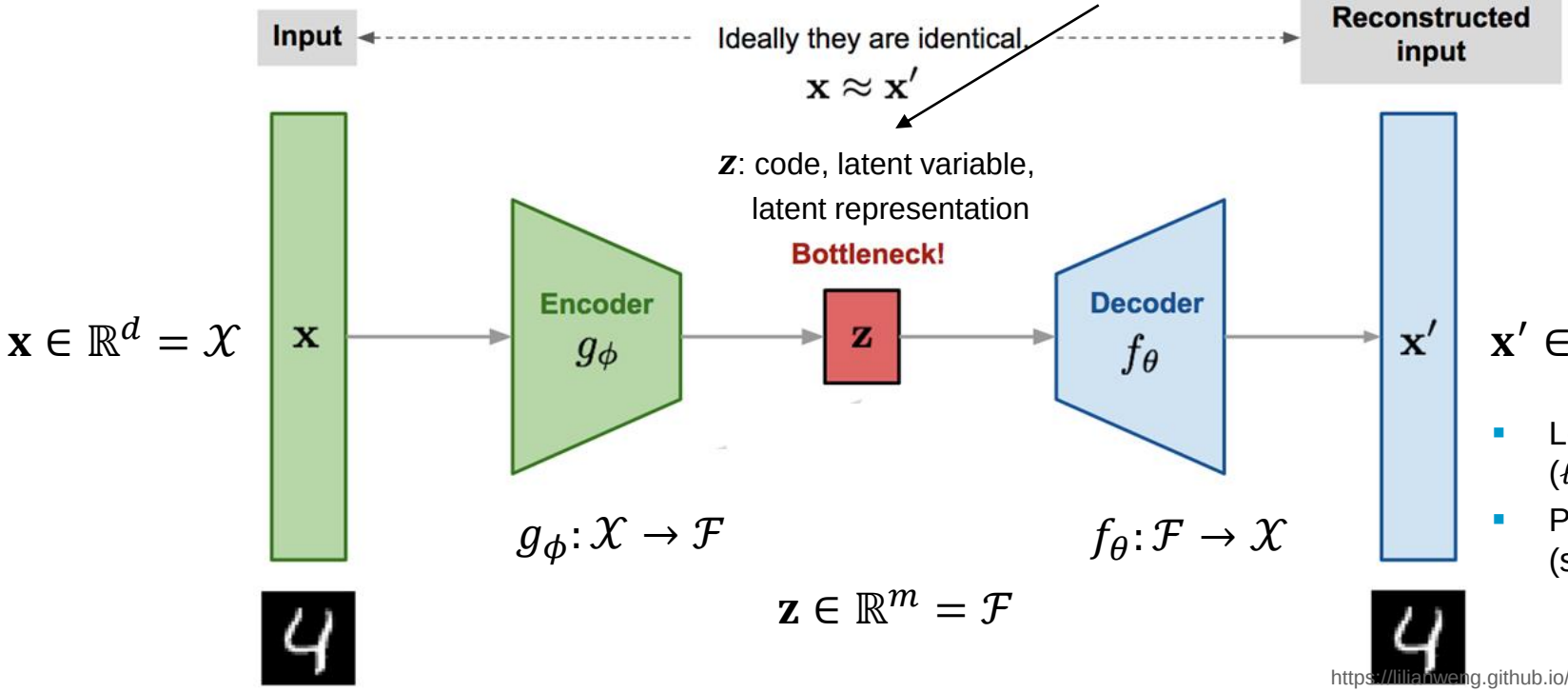
Autoencoders

Encode the input into a representation (bottleneck) and reconstruct it with the decoder



Latent space learnt on MNIST

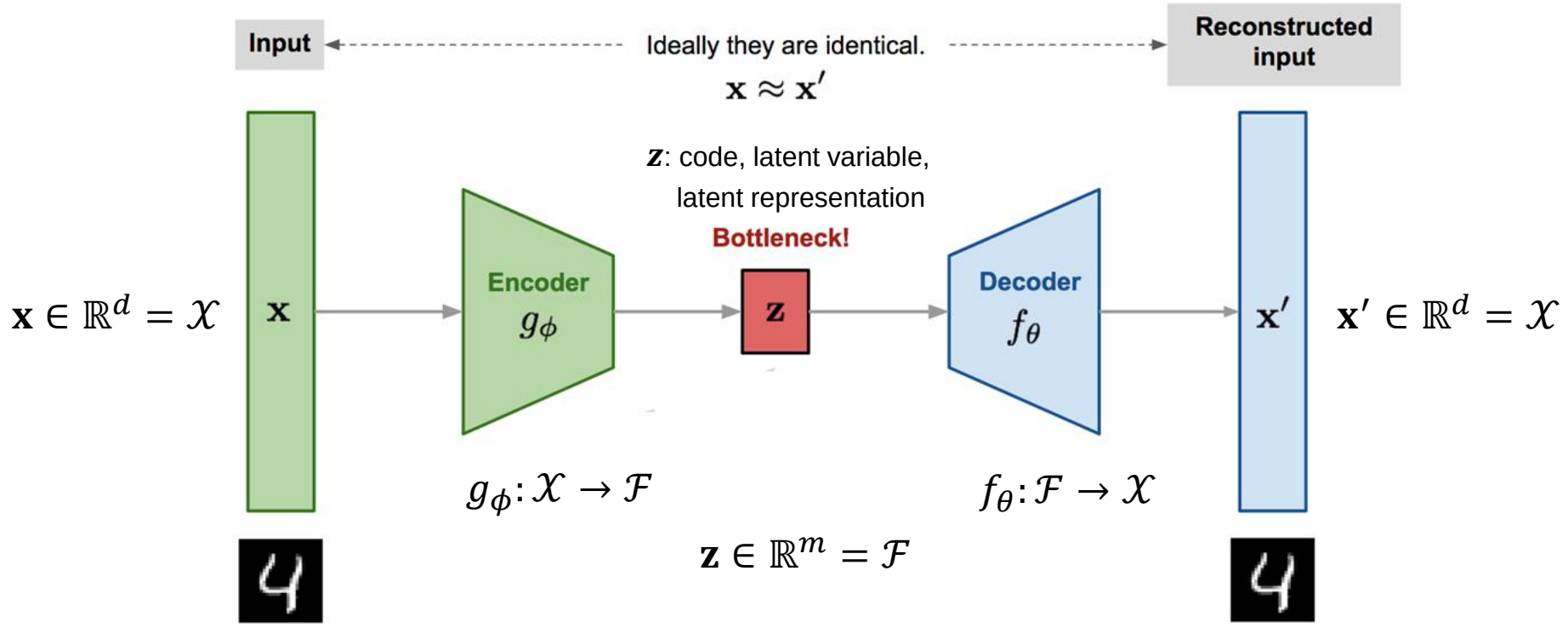
https://bit.ly/37ctFMS



- Learn features (ℓ_2 loss)
- Predict labels (softmax, etc.)

https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

Autoencoders



- Encoder and decoder are separated by a **penalty** which is either a *dimensionality constraint* or *regularization*
- **Undercomplete** ($m < d$): compression/dimensionality reduction
  - Capture only the most important features of $\mathbf{x}$
- **Overcomplete** ($m \geq d$): adds some form of redundancy to $\boldsymbol{z}$
  - Could learn the identity function (regularization can avoid this)

https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

Autoencoders

- A NN to learn efficient data representation (encoding) in unsupervised manner

- (1) Encode the input into a compressed and meaningful representation (latent space representation)
- (2) Decode it back such that the reconstructed input (lossy) is similar as possible to the original one.

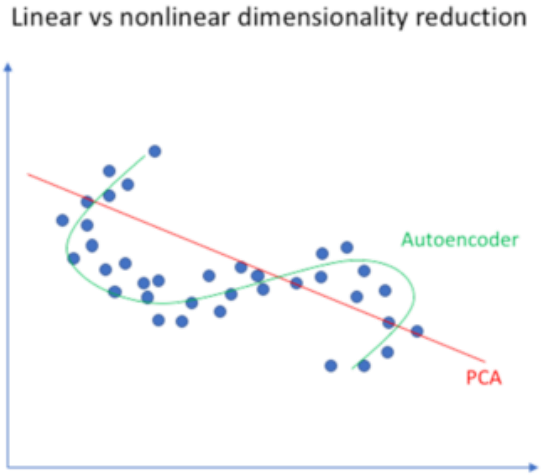
- Network trained to ignore the signal “noise” - the target output is the input

Types

- **Regularized** autoencoders: denoising, sparse, contractive
- **Variational** autoencoders (VAEs)

- **Linear** vs. **Non-Linear**
  (e.g., depending on the activation function)
- A non-linear AE can basically perform nonlinear PCA
- Restricted Boltzmann Machines (RBMs): non-linear AE

- **Boolean** AE: $\mathcal{X} = \mathcal{F} = \{0,1\}$ (e.g., $GF(2) = \mathbb{F}_2$),
  g_ϕ, f_θ unrestricted Boolean functions
  $\mathcal{L}$ Hamming distance



Generating new data with Autoencoders?

- Autoencoders can reconstruct data and can learn features to initialize a supervised model.
- **Q:** Can we generate new data from an autoencoder? **A**: Not really, why?

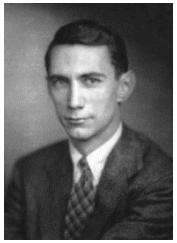
- Dimensionality reduction with no reconstruction loss comes with a price: lack of interpretable and exploitable structures in the latent space (lack of regularity).
- Dimensionality reduction reduces the number of dimensions of the data **while** keeping the major part of the data structure information in the reduced representations.
- Thus, the dimension of the latent space and the “depth” of autoencoders (that define degree and quality of compression) have to be carefully controlled and adjusted.

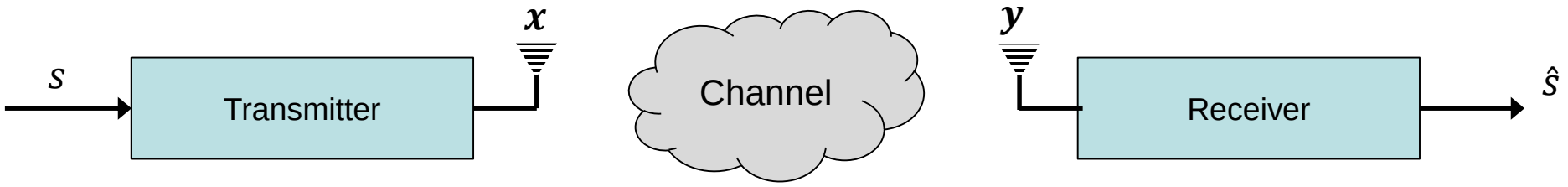
- At first sight: if the latent space is regular enough (well “organized” by the encoder during the training process), we could take a point randomly from that latent space and decode it to get a new content (i.e., decoder acts like the generator of a GAN)
- BUT: regularity of the latent space for AEs depends on the distribution of the data in the initial space, the dimension of the latent space and the architecture of the encoder.
- The high degree of freedom of the AE that makes possible to encode and decode with no information loss (despite the low dimensionality of the latent space) leads to a severe overfitting (some points of the latent space will give meaningless content once decoded).

- AE is solely trained to encode and decode with as minimum loss (no matter how the latent space is organized).
- If not careful, the network takes advantage of any overfitting possibilities to achieve its task as well as it can… unless we explicitly regularize it!

The Communication Problem

"The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.
C. E. Shannon. A mathematical theory of communication. Bell System Technical Journal, 27:379-423, 625-56, 1948.





Goal: minimize $\mathbb{P}(\hat{s} \neq s)$

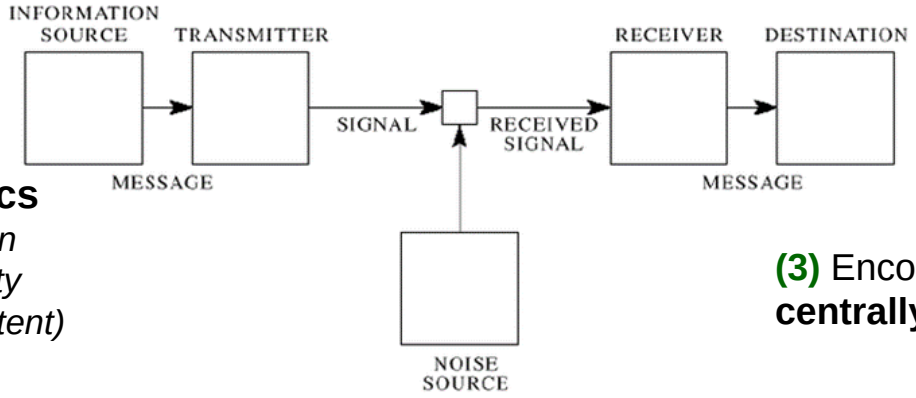
- $s \in \mathcal{M} = \{1, \dots, M\}, k = \log_2 M$
- $\boldsymbol{x} \in \mathbb{C}^n$ with constraint: energy $\|\boldsymbol{x}\|_2^2 \leq n$, amplitude $|x_i| \leq 1, \forall i$, average power $\mathbb{E}[|x_i|^2] \leq 1, \forall i$.
- $\boldsymbol{y} \in \mathbb{C}^n \sim p(\boldsymbol{y}|\boldsymbol{x})$
- $\hat{s} \in \mathcal{M}$
- Rate: $R = \frac{k}{n}$ bits/channel use

The Communication Problem

(1) The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. (2) Frequently the messages have *meaning*; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. (3) The significant aspect is that the actual message is one *selected from a set* of possible messages.

C. Shannon. A mathematical theory of communication. Bell System Technical Journal, 27:379-423, 625-56, 1948.

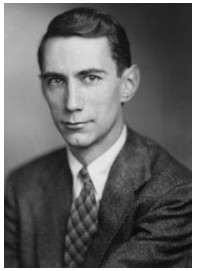
(1) Reliable transfer of information

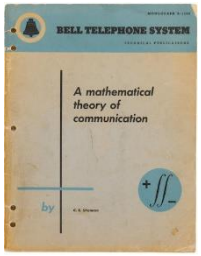


Shannon's model

(2) No semantics
dichotomy between information quantity and meaning (content)

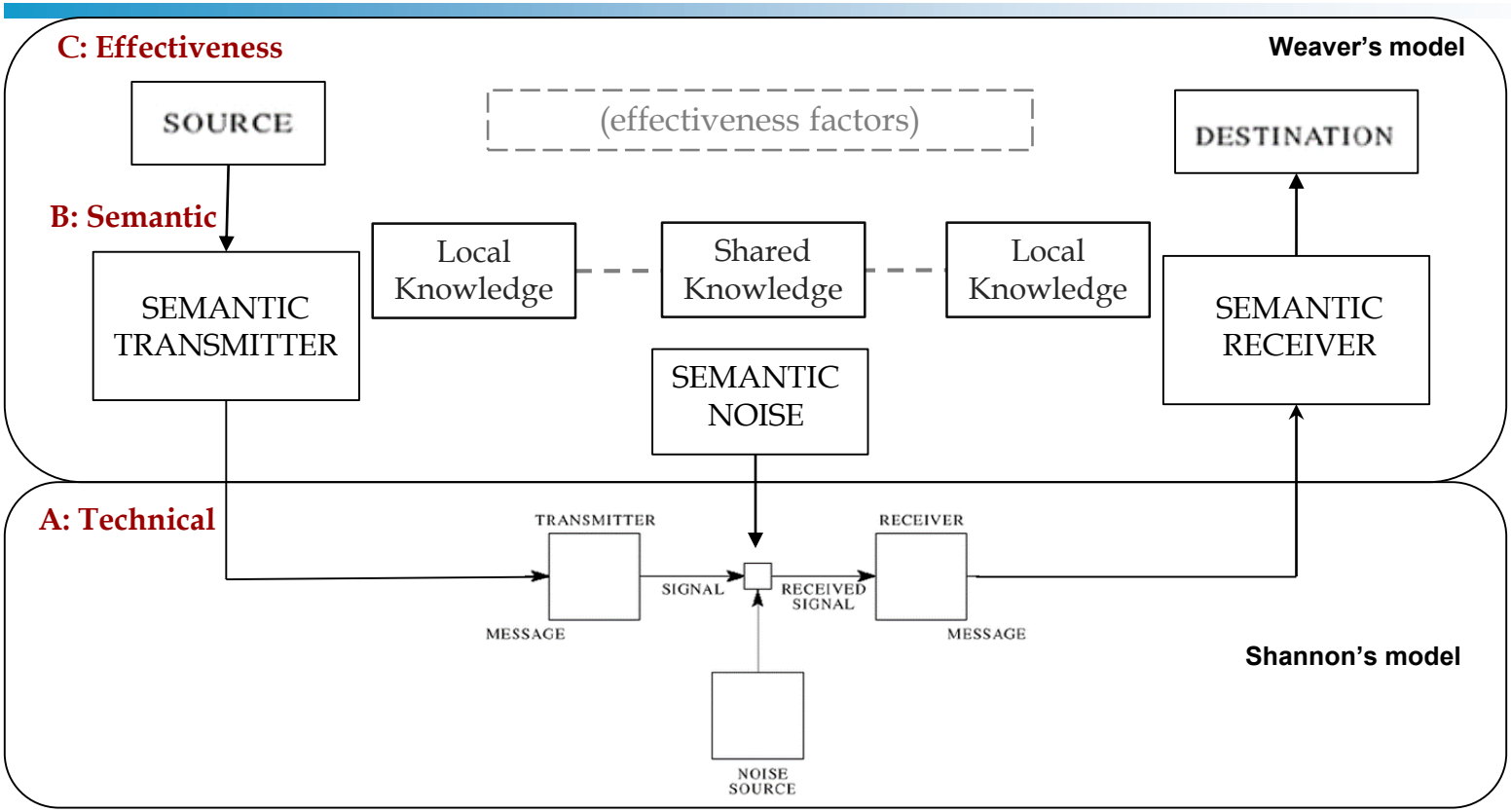
(3) Encoder and decoder **centrally/jointly** designed

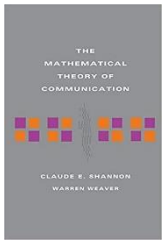


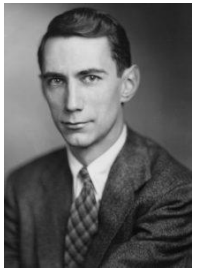


- **Lore**: theory of information quantity, but not of information content
- Focus on "*noise*" (and *"equivocation"*) rather than "*signal*"
- Suitable and instrumental for classical human-generated data communication

The General Problem







LEVEL A. How accurately can the symbols of communication be transmitted? (The technical problem.)

LEVEL B. How precisely do the transmitted symbols convey the desired meaning? (The semantic problem.)

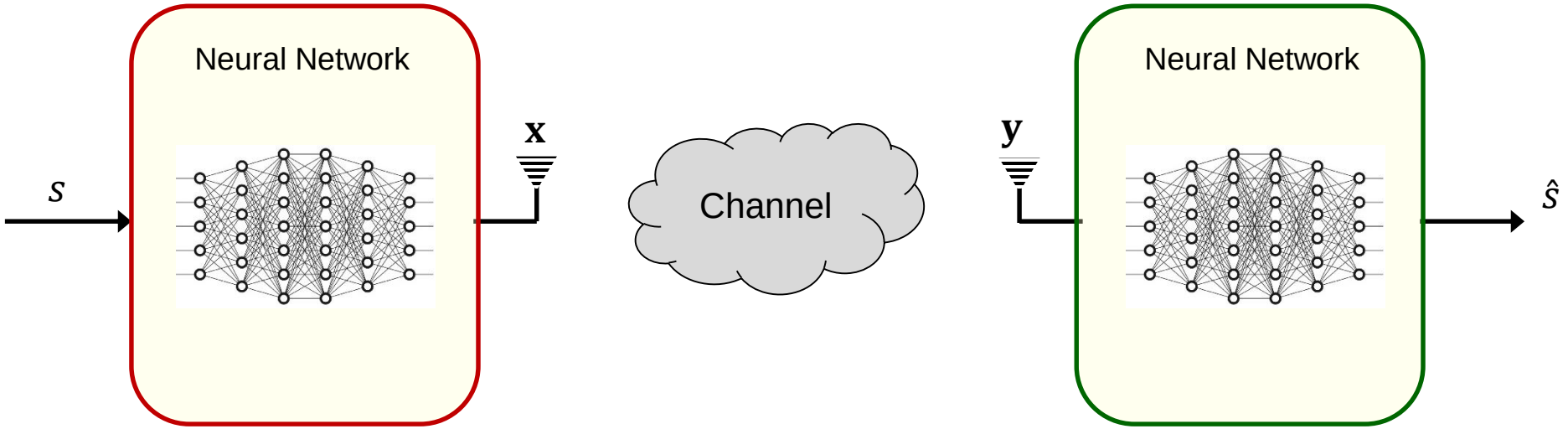
LEVEL C. How effectively does the received meaning affect conduct in the desired way? (The effectiveness problem.)

Syntactic level

Semantic level

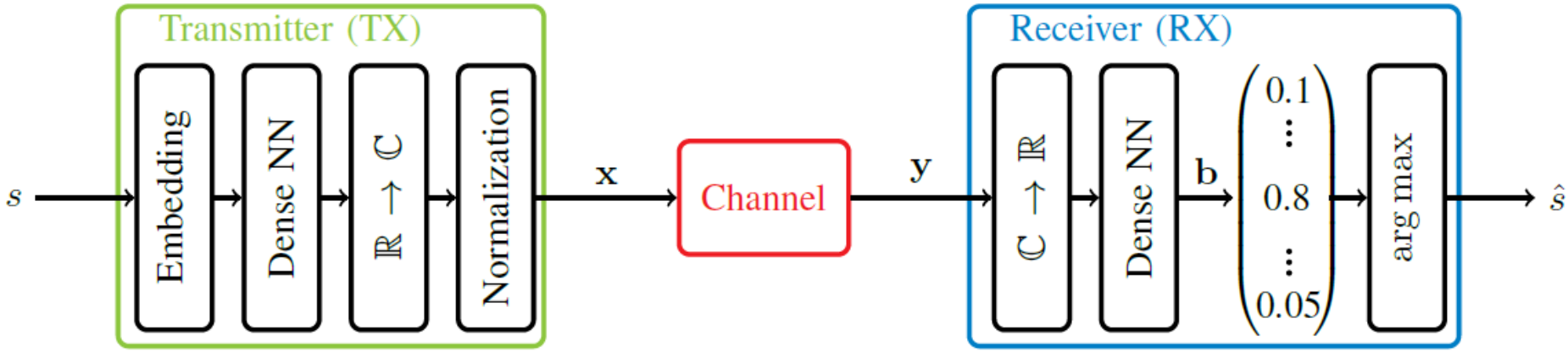
Pragmatic level

Communication seen as an Autoencoder



- Learn a robust message representation
- Trainable from end-to-end to minimize $\mathbb{P}(\hat{s} \neq s)$
- Universal concept that applies to any channel $p(\boldsymbol{y}|\boldsymbol{x})$

Communication system as Autoencoder

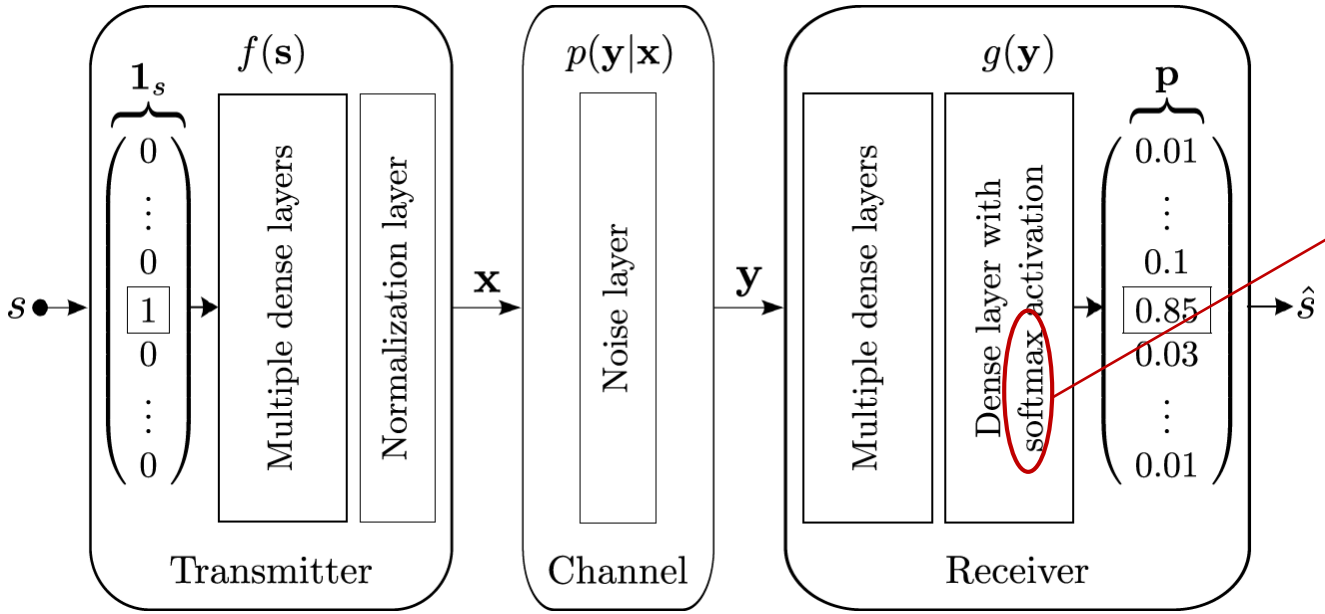


- **Goal** (usually): find a low-dimensional representation of input at some intermediate layer that allows reconstruction at the output with minimal error.
- Autoencoder learns non-linearly to compress and reconstruct the input.

- Communication systems: learn representations $\boldsymbol{x}$ of the messages s that are robust wrt channel impairments mapping $\boldsymbol{x}$ to $\boldsymbol{y}$ (i.e., noise, fading, distortion, etc.), so that the transmitted message can be recovered with small probability of error.

- "**Channel Autoencoder**": adds redundancy to input data, learning an intermediate representation robust to channel perturbations

Communication system as Autoencoder



$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

E.g., logit scores

$$\boldsymbol{y} = \begin{pmatrix} 2.0 \\ 1.0 \\ 0.1 \end{pmatrix} \mapsto \boldsymbol{p} = \begin{pmatrix} 0.7 \\ 0.2 \\ 0.1 \end{pmatrix}$$

- Feedforward NN with multiple dense layers
- Input: one-hot vector

- additive noise layer with a fixed variance $\beta = \frac{1}{2R}\left(\frac{E_b}{N_0}\right)^{-1}$

- Feedforward NN
- *Softmax* activation: output $\boldsymbol{p} \in (0,1)^M$ - probability vector over all possible messages.
- Decoded message $\hat{s}$: index of the element of $\boldsymbol{p}$ with the highest probability.

Embedding

- **Embedding** maps integers to vectors, i.e., essentially, a lookup table that returns columns $s \in \mathbb{M}$ of matrix $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M]$.

- Simply a more efficient implementation of a dense layer with **one-hot** encoded inputs:

$$\mathbf{W} \begin{bmatrix} \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} = \mathbf{w}_s$$

- *One-hot vector* $\mathbf{1}_s \in \mathbb{R}^M$: an *M*-dimensional vector, the *s*-th element of which is equal to one and zero otherwise.

- $\mathbf{W}$ is trainable like the weight matrix of a dense layer

Dealing with Complex Values?

- In Comm/SP, we deal with complex numbers – Deep Learning mostly works with real numbers.
- All math operations in the complex domain can be represented with a purely real-valued NN of twice the size - each complex number represented by two real values.
- Example: NN with a scalar complex input and output connected through a single complex weight, i.e., $y = wx$, where $y, w, x \in \mathbb{C}$, can be represented as a real-valued NN $\mathbf{y} = \mathbf{W}\mathbf{x}$, where the vectors $\mathbf{y}, \mathbf{x} \in \mathbb{R}^{\mathbf{2}}$ contain the real and imaginary parts of y and x in each dimension and $\mathbf{W} \in \mathbb{R}^{\mathbf{2}\times\mathbf{2}}$ is a weight matrix.
- Transformations:

$$\mathbb{R}2\mathbb{C}\colon\ \mathbb{R}^n \mapsto \mathbb{C}^{n/2} \qquad \mathbb{C}2\mathbb{R}\colon \mathbb{C}^{n/2} \mapsto \mathbb{R}^n$$

$$\mathbb{R}2\mathbb{C}(\boldsymbol{x}) = \begin{bmatrix} x_1 \\ \vdots \\ x_{\frac{n}{2}-1} \\ \vdots \\ x_{n-1} \end{bmatrix} + j \begin{bmatrix} x_2 \\ \vdots \\ x_{\frac{n}{2}} \\ \vdots \\ x_n \end{bmatrix} \qquad \mathbb{C}2\mathbb{R}(\boldsymbol{x}) = \begin{bmatrix} \Re(\boldsymbol{x}) \\ \Im(\boldsymbol{x}) \end{bmatrix}$$

- Extensions to complex NNs exist **but** gains unclear, gradients not defined (traditional loss and activation functions are generally not holomorphic[1] – Wirtinger calculus to the rescue)

[1]holomorphic is a complex-valued function of one or more complex variables that is, at every point of its domain, complex differentiable in a neighborhood of the point.

Normalization Layer

- Normalization is necessary to ensure that constraints on $\boldsymbol{x}$ are met
- Can be seen as a neural network layer without any trainable parameters, i.e., a differentiable operation

- Instantaneous normalization: $\frac{x}{|x|}$

- Constraint on symbol amplitude: $\min(\max(x_i, x_{min}), x_{max})$

- Average power normalization:

$$\frac{x(s)}{\sqrt{\frac{1}{\frac{M}{2}}\sum_{s=1}^{M}\|x(s)\|^2}} \approx \frac{x(s)}{\sqrt{\frac{1}{\frac{N}{2}}\sum_{i=1}^{N}\|x_i\|^2}}$$

- Even (pseudo) quantization of $\boldsymbol{x}$ can be done

Channel Layer

- We require a differentiable generative model for $p(\boldsymbol{y}|\boldsymbol{x})$, i.e.,

$$\nabla_{\boldsymbol{x}} y_i, \forall i \text{ must be known}$$

- No trainable parameters , stochastic transformation of the input

- Autoencoder penalty layer: e.g., regularization by adding noise

→ Encoder is forced to learn robust message representations

- Examples
  - Additive white Gaussian noise channel: $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{n}$

$$\nabla_{\boldsymbol{x}} y_i = \mathbf{1}$$

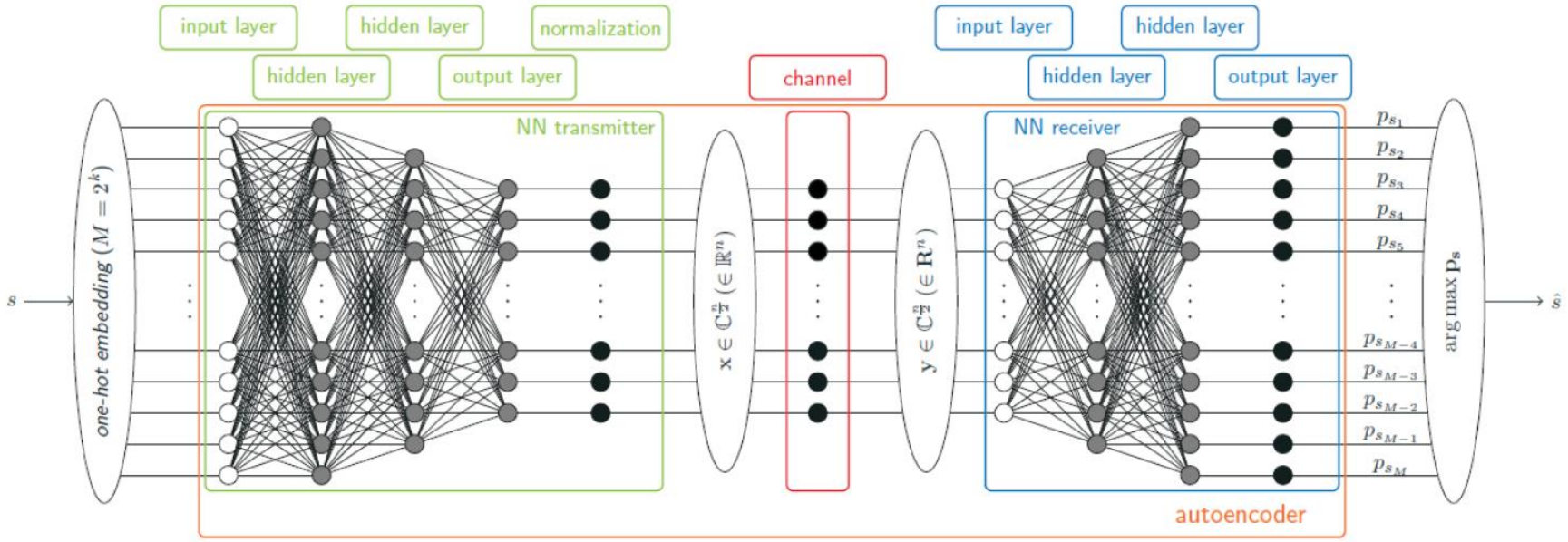
  - Memoryless fading channel: $\boldsymbol{y} = \mathrm{h}\boldsymbol{x} + \boldsymbol{n}$

$$\nabla_{\boldsymbol{x}} y_i = \mathrm{h}\mathbf{1}$$

  - Multi tap fading channel: $y_i = \sum_{l=1}^{L} h_l x_{i-l+1} + n_i$

$$\nabla_{\boldsymbol{x}} y_i = [\cdots \quad 0 \quad h_L \quad \cdots \; h_1 \; \cdots]$$

End-to-End Training

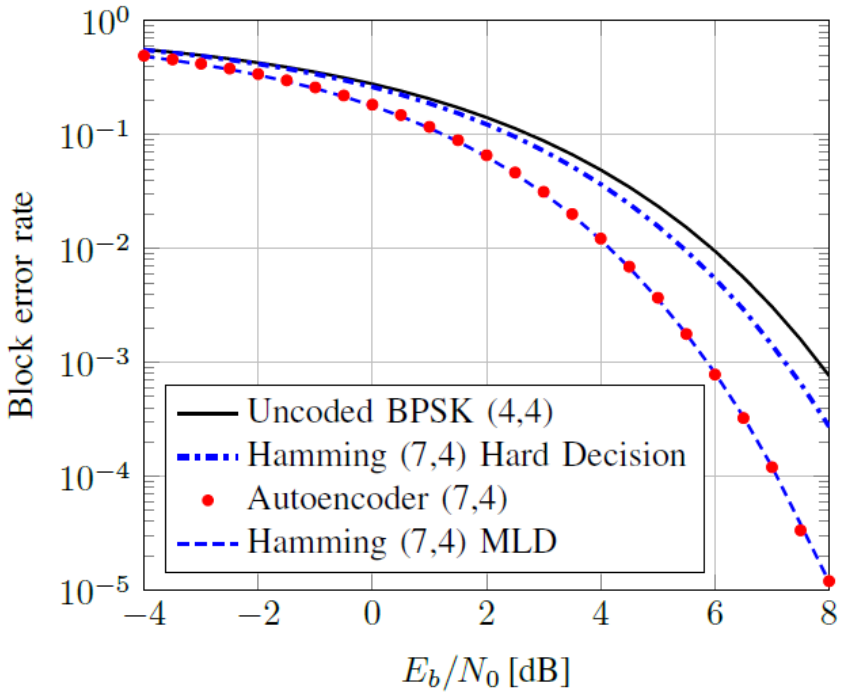


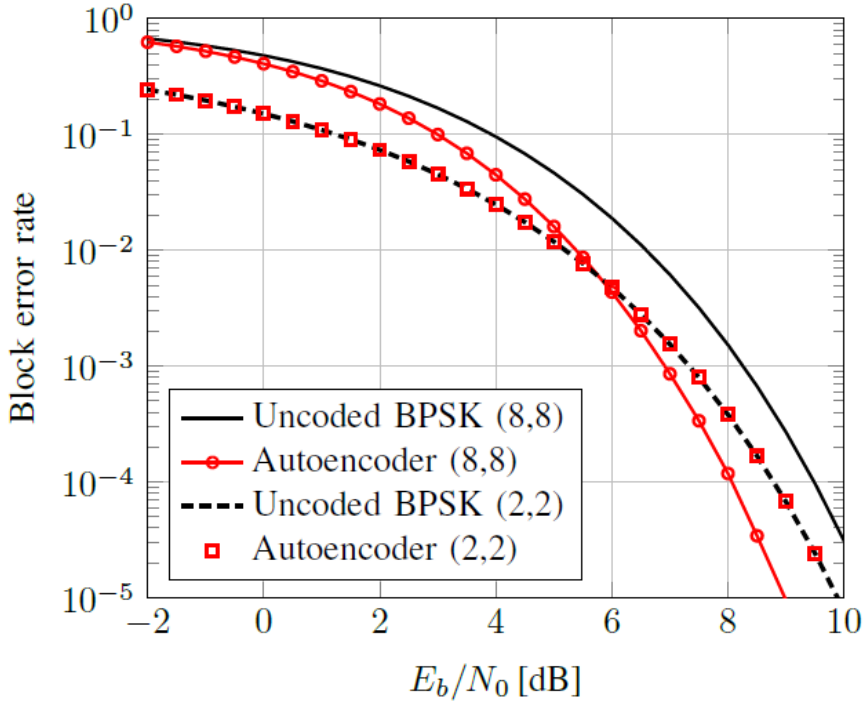
Training process:

- Classification task: (categorical) cross entropy loss
- Channel model $p(\boldsymbol{y}|\boldsymbol{x})$ is
  - stochastic : infinite amount of training data
  - differentiable : gradient can be computed through the channel

↦ SGD can optimize transmitter and receiver jointly!

Performance

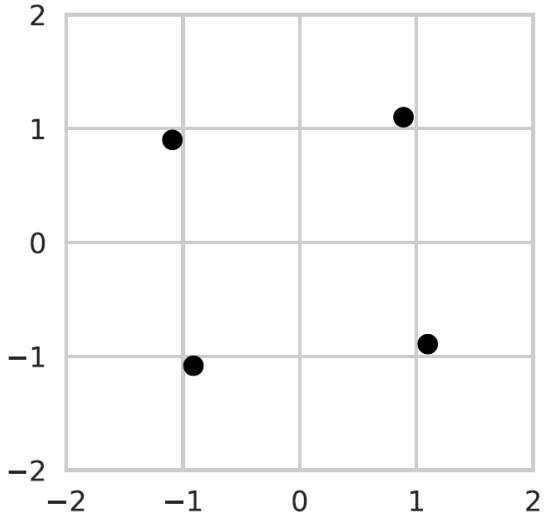




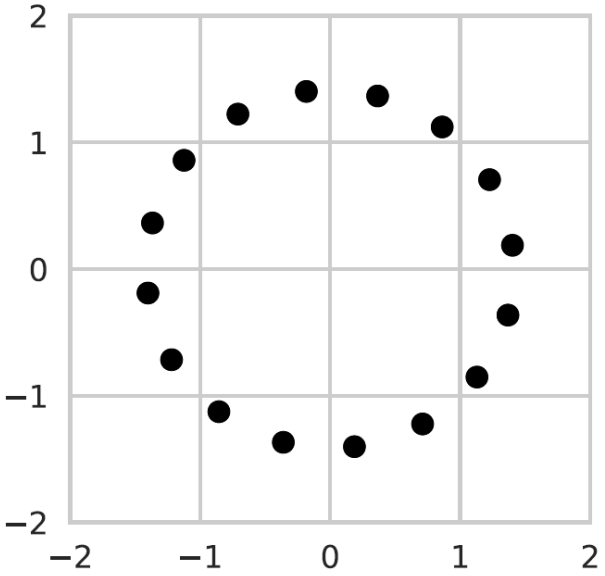
Rate: $R = k/n$ [bit/channel use], $k = \log_2 M$
(k, n): sends one out of $M = 2^k$ messages (i.e., k bits) through n channel uses.

Learned Constellations

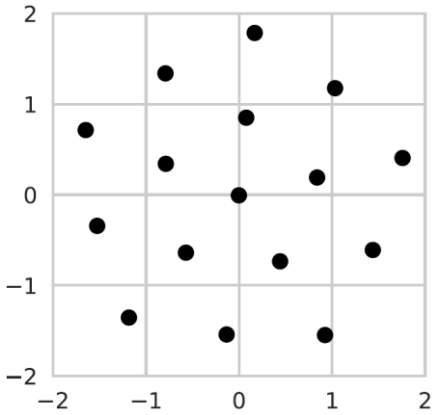
AE (2,2)



AE (2,4)



AE (2,4) with average power constraint



How to interpret Autoencoder Training

$$\min_{\theta_T,\theta_R} \mathbb{E}_{s,\boldsymbol{y}}\left[-\log[g_{\theta_R}(\boldsymbol{y})]_s\right] \geq \min_{\theta_T} \mathbb{E}_{s,\boldsymbol{y}}[-\log p(s|\boldsymbol{y})] = \min_{\theta_T} H(s|\boldsymbol{y})$$

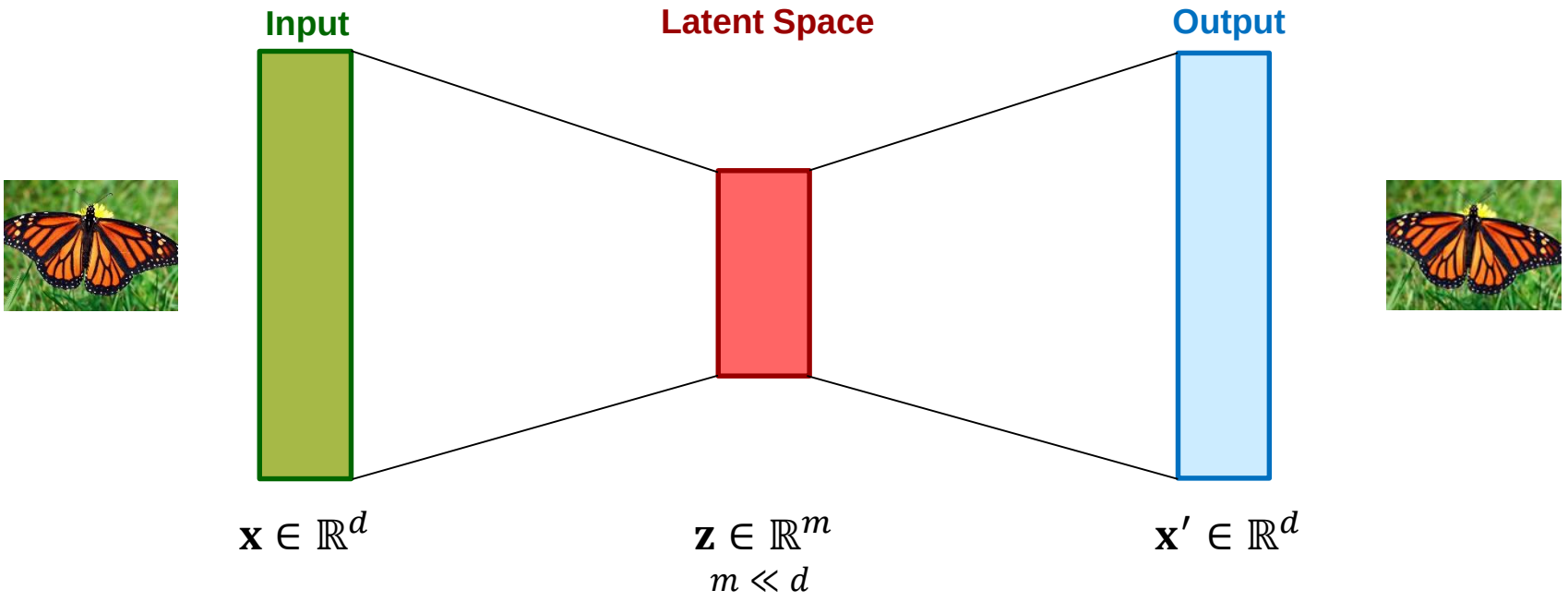
$s \sim \text{Uniform}(\mathcal{M})$
$\boldsymbol{y} \sim p(\boldsymbol{y}|f_{\theta_T}(s))$

ML estimator

- Expectation approximated through Monte Carlo Sampling
- Minimization through gradient descent
- Learns effectively to maximize $I(s;\boldsymbol{y}) = H(s) - H(s|\boldsymbol{y})$
- Transmitter learns a higher order constellation
  - a.k.a. geometric shaping
  - Can be implemented as a simple look up table
- Receiver learns the maximum likelihood detector for a sufficiently rich $g_{\theta_R}(\boldsymbol{y})$

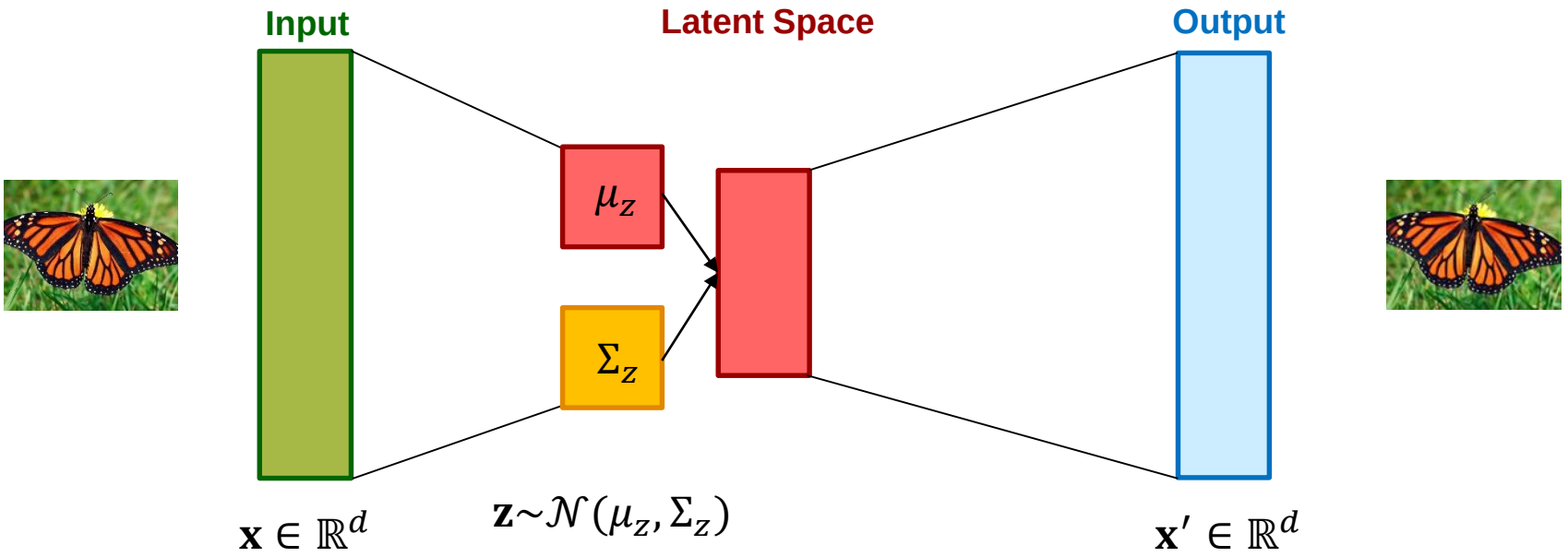
Variational Autoencoders

Autoencoders



- Constrained in some way (e.g., small latent vector representation)
- Training:
  - Given data $\mathbf{x}$, feedforward to $\mathbf{x}'$ output
  - Compute losses, e.g. $\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2$
  - Backpropagate loss gradient to update weights
- Not a generative model (in general)
- Can generate new images by giving different latent vectors to trained network

Variational Autoencoders



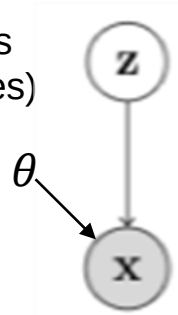
- Variational: use probabilistic latent encoding
- Latent space is a distribution (Gaussian)
- Generative process: latent space should be regular enough
- Regularity can be introduced by explicit regularization during the training process.
- VAE: an AE whose training is regularized to avoid overfitting and ensure that the latent space has good properties that enable generative process.

VAE – Probabilistic Model Perspective

- Consider a directed, latent variable model

- Joint pdf of the model: $p_\theta(x,z) = p(x|z)p(z)$ $(\theta \in \Theta)$
- Decomposes into likelihood (generator) $p(x|z)$ and prior $p(z)$

Graphical model
z: latent (hidden) variables
x: data (observed variables)



- Generative process:
  - Draw latent variables $z_i \sim p(z)$ [sample from true prior]
  - Draw data point $x_i \sim p(x|z)$ [sample from true conditional]

- Consider a family of distributions $\mathcal{P}_z$ where $p(z) \in \mathcal{P}_z$
- Consider a family of conditional distributions $\mathcal{P}_{x|z}$ where $p(x|z) \in \mathcal{P}_{x|z}$

- Hypothesis class of generative models is the set of all possible combinations
- $\mathcal{P}_{x,z} = \{p(x,z) \,|p(z) \in \mathcal{P}_z, p(x|z) \in \mathcal{P}_{x|z}\}$

- Given a dataset $\mathcal{D} = \{x^{(1)}, \ldots, x^{(N)}\}$, we are interest in the following learning and inference tasks

- Selecting $p \in \mathcal{P}_{x,z}$ that best "fits" $\mathcal{D}$
- Given a sample x and a model $p \in \mathcal{P}_{x,z}$, what is the **posterior** distribution over the latent variables z ?
  - Inferring good values of z, given observed data

VAE – Probabilistic Model Perspective

- Posterior distribution of the latent variables
- Bayes' Theorem $p(z|x) = \frac{p_\theta(x|z)p(z)}{p(x)} = \frac{p_\theta(x|z)p(z)}{\int_z p_\theta(x,z)} = \frac{p_\theta(x|z)p(z)}{\int p_\theta(x|z)p(z)dz}$
- Need to calculate **evidence**: $p(x) = \int_z p_\theta(x|z)p(z)\mathrm{d}z$

Integral over all configurations of latent variables

(usually) intractable!!! in higher dimensions

Could use Markov Chain Monte Carlo (MCMC) to approximate, but it's expensive

- Variational inference to the rescue !!!
- Approximate intractable true posterior $p(z|x)$ with a manageable distribution,

  i.e., with the "best" distribution $q_\lambda(z|x) \in \mathcal{Q}$ (family of variational distributions)

- Which choice of $\lambda \in \Lambda$ gives the "best" $q_\lambda(z|x)$?
- Kullback-Leibler (KL) divergence measures the closeness of two distributions, i.e., information lost when using q_λ to approximate p
- Choose λ to minimize $D_{KL}\left(q_\lambda(z|x)\|p(z|x)\right) = D_{KL}(q_\lambda\|p) = \mathbb{E}_{q_\lambda}\left[\log\frac{q_\lambda(z|x)}{p(z|x)}\right]$

Kullback-Leibler Divergence

- Measures how different the two distributions are
  - P = 'true' distribution
  - Q = alternative distribution that is used to encode data
- KL divergence is the expected extra # of bits per point that must be transmitted using Q instead of P if the data comes from P

$$
\begin{aligned}
D_{KL}\,(P||Q) &= \sum_i P(x_i) \log \frac{P(x_i)}{Q(x_i)} \\
&= -\sum_i P(x_i) \log Q(x_i) + \sum_i P(x_i) \log P(x_i) \\
&= H(P,Q) - H(P) \\
&= \text{Cross-entropy} - \text{entropy}
\end{aligned}
$$

- Measures how well a probability distribution Q approximates a distribution P (the "truth")
- The average information gained from moving from Q to P

Properties

- Divergence 0 if and only if P and Q are equal: $D_{KL}(P||Q) = 0$ iff $P = Q$
- Non-symmetric: $D_{KL}(P||Q) \neq D_{KL}(Q||P)$
- Non-negative: $D_{KL}(P||Q) \geq 0$
- Not a distance metric: not commutative, doesn't satisfy triangle inequality

$$D_{KL}(P||Q) \leq D_{KL}(P||R) + D_{KL}(R||Q)$$

Variational Inference - ELBO

- We start from the log probability of the observations (marginal probability of x)

$$\log p(x) = \log \int_z p(x,z)dz$$

Equality only holds if the support of q includes that of p

$$= \log \int_z \frac{q_\lambda(z|x)}{q_\lambda(z|x)} p(x,z)dz = \log\left(\mathbb{E}_{z\sim q_\lambda}\left[\frac{p(x,z)}{q_\lambda(z|x)}\right]\right)$$

Tightness of the lower bound depends on the specific choice of q

$$\geq \mathbb{E}_{z\sim q_\lambda}\left[\log\left(\frac{p(x,z)}{q_\lambda(z|x)}\right)\right]$$

Jensen's inequality

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$$

$$= \mathbb{E}_{z\sim q_\lambda}[\log p(x,z)] - \mathbb{E}_{z\sim q_\lambda}[\log q_\lambda(z|x)] = \mathbb{E}_{z\sim q_\lambda}[\log p(x,z)] + H(z|x)$$

Evidence Lower Bound (ELBO) (*variational lower bound or variational free energy*)

- We choose a family of variational distributions such that the expectations are computable.
- Minimizing $D_{KL}\ (q_\lambda \| p)$ w.r.t. λ is equivalent to maximizing ELBO(λ)
- Maximize ELBO to find the parameters that give as tight a bound as possible on the marginal probability of x

$$D_{KL}\ (q_\lambda \| p) = -\text{ELBO}(\lambda) + \log p(x)$$

Variational Lower Bound – ELBO (alt)

- We start from the log probability of the observations (marginal probability of x)

$$\log p(x) = \log \int_z p(z)p(x|z)dz$$

$$= \log \int_z p(z)p(x|z)dz = \log \int_z q_\phi(z) \frac{p(z)}{q_\phi(z)} p(x|z)dz$$

$$\geq \int_z q_\phi(z) \log \left[\frac{p(z)}{q_\phi(z)} p(x|z) \right] dz$$

$$= \underbrace{\mathbb{E}_{z\sim q_\phi} \left[\log \left(\frac{p(z)}{q_\phi(z)} \right) \right]}_{-D_{KL}\left(q_\phi \| p\right)} + \mathbb{E}_{z\sim q_\phi} [\log p(x|z)]$$

(contraction part)

likelihood of generated data

- So we try to maximize the variational lower bound

$$\log p(x) \geq \mathbb{E}_{z\sim q_\phi} [\log p(x|z)] - D_{KL}\left(q_\phi \| p\right) = L(\phi, q)$$

$N(0,1)$

- We'd like to choose q to make the bound as tight as possible.
- The gap is given by $\log p(x) - L(\phi, q) = D_{KL}\left(q_\phi \| p(z|x)\right)$
- Therefore, we'd like q to be as close as possible to the posterior distribution $p(z|x)$

Kullback-Leibler Divergence

$$D_{KL}\left(q_\lambda(z|x)\|p(z|x)\right) = \mathbb{E}_{z\sim q_\lambda}\left[\log\frac{q_\lambda(z|x)}{p(z|x)}\right]$$

$$= \mathbb{E}_{z\sim q_\lambda}[\log q_\lambda(z|x) - \log p(z|x)]$$

$$= \mathbb{E}_{z\sim q_\lambda}[\log q_\lambda(z|x)] - \mathbb{E}_{z\sim q_\lambda}[\log p(z|x)]$$

$$= \mathbb{E}_{z\sim q_\lambda}[\log q_\lambda(z|x)] - \mathbb{E}_{z\sim q_\lambda}[\log p(x,z)] + \log p(x)$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{-\mathcal{L}}$$

- Still contains $p(x)$ term – cannot compute directly
- But $p(x)$ does not depend on λ, q

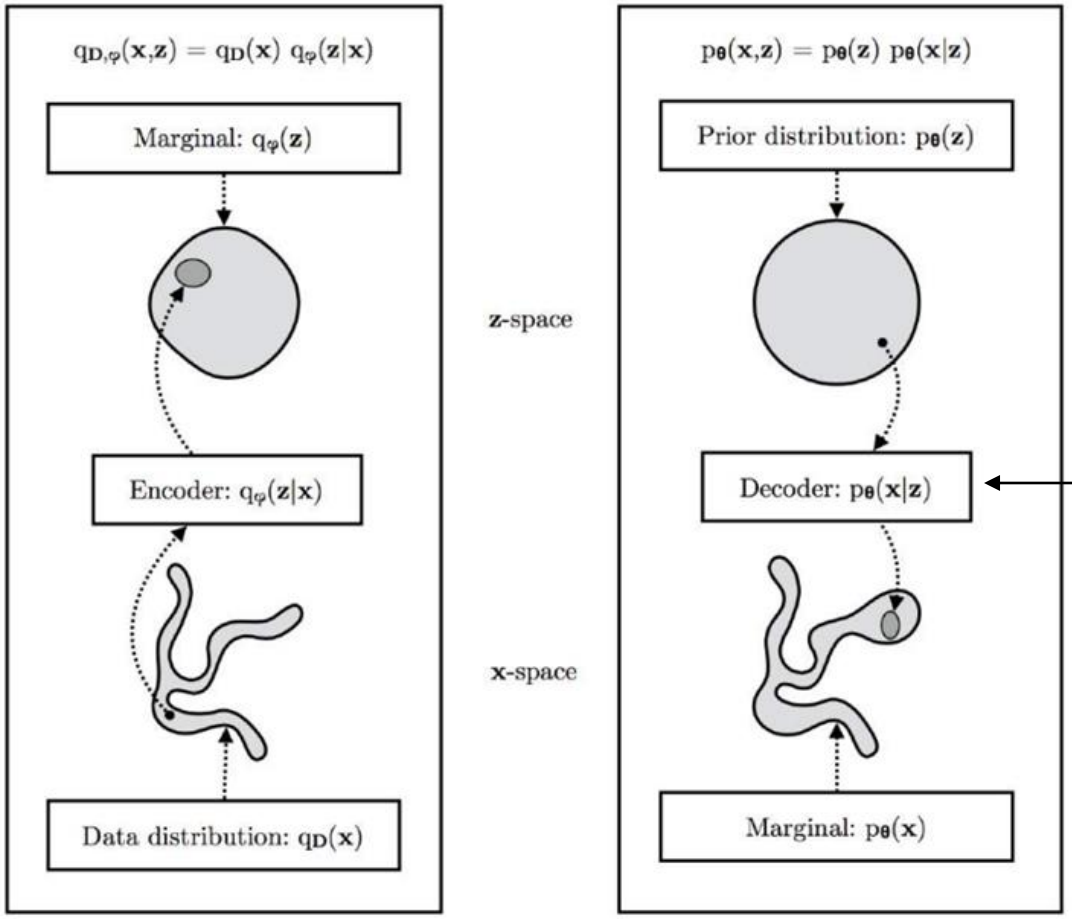
Evidence Lower Bound (ELBO)

$$\log p(x) = D_{KL}\left(q_\lambda(z|x)\|p(z|x)\right) + L[q_\lambda(z|x)]$$

lower bound

$$\text{ELBO}(\lambda) \triangleq L[q_\lambda(z|x)] = \mathbb{E}_{z\sim q_\lambda}[\log p(x,z)] - \mathbb{E}_{z\sim q_\lambda}[\log q_\lambda(z|x)] \quad \text{(ELBO)}$$

ELBO as KL Divergence

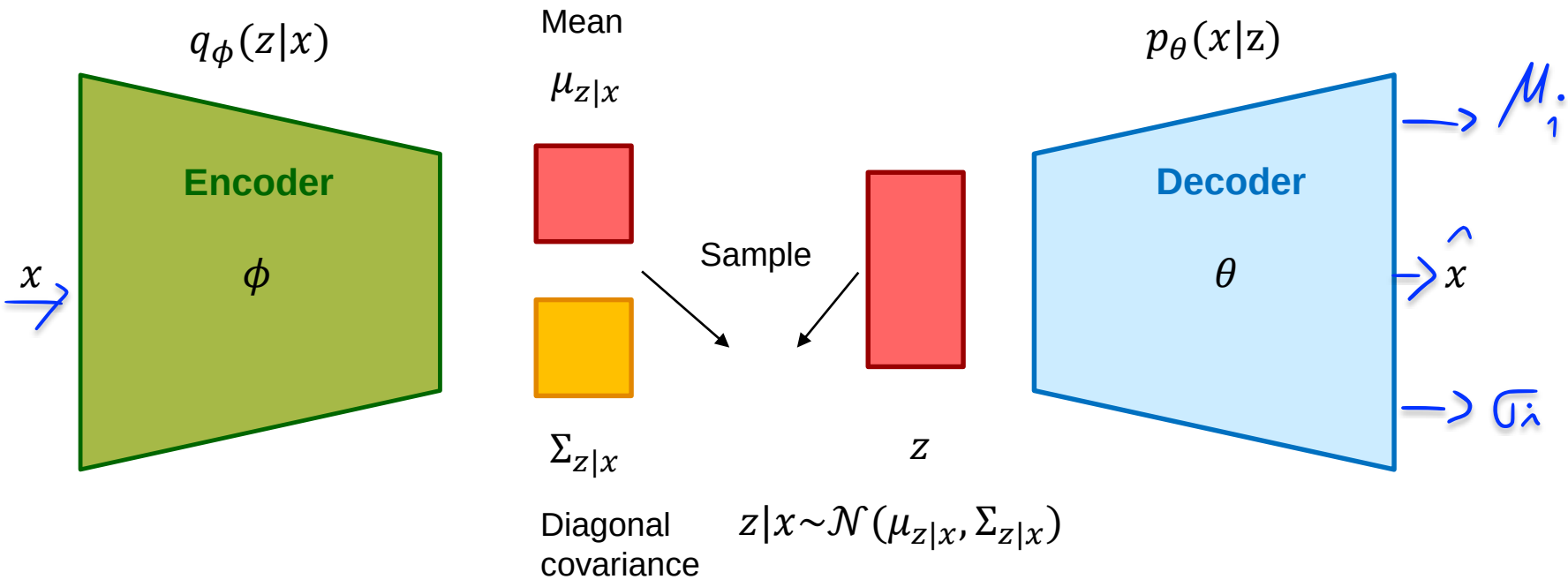


neural net

ML objective = - $D_{KL}(\ q_D(x)\ ||\ p_\theta(x)\ )$
ELBO objective = - $D_{KL}(\ q_D(x,z)\ ||\ p_\theta(x,z)\ )$

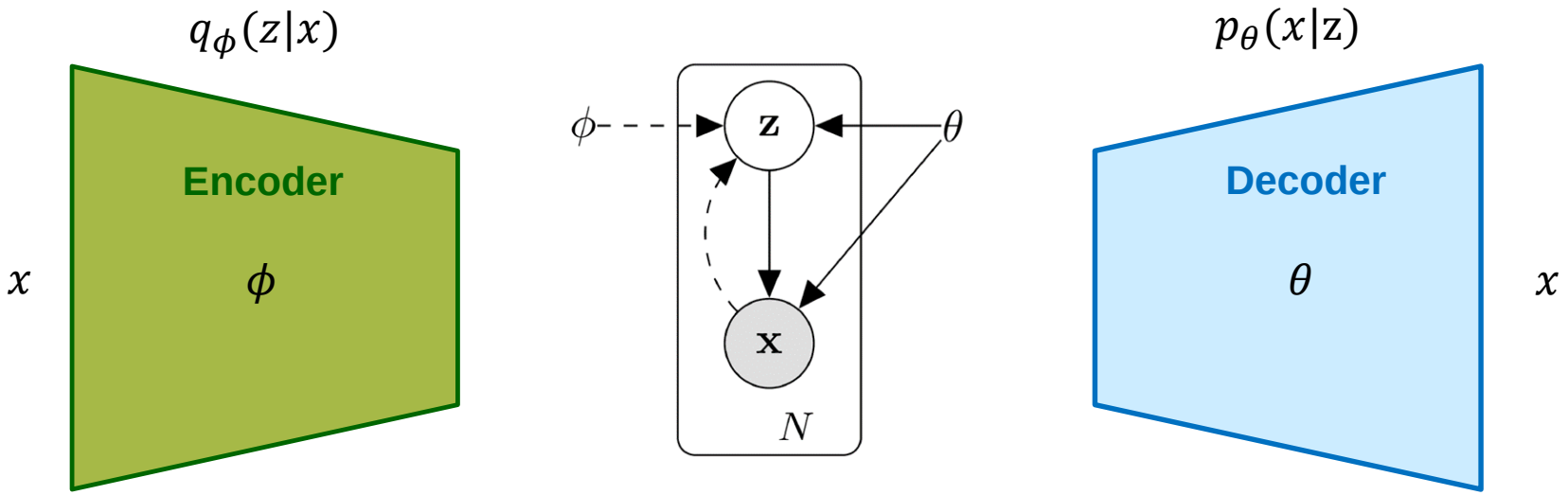
VAE – Deep Learning Perspective

- Latent space is a distribution (Gaussian)
- Sample from the mean and standard deviation to compute latent sample
- Training: loss makes sure the latent space is close to a Gaussian and the reconstructed output is close to the input
- Test: sample from the latent space



VAE – Deep Learning Perspective

- Implement $q_\phi(z|x)$ as a neural network (probabilistic encoder)
- Implement $p_\theta(x|z)$ as a neural network (probabilistic decoder)
- Sample $z \sim q_\lambda(z|x)$ in the middle
- VAE performs *probabilistic, non-linear dimensionality reduction*
- Uses a generative model with a latent variable distributed according to some prior distribution $p(z)$
- The observed variable is distributed according to a conditional distribution $p_\theta(x|z)$
- Training tries to maximize the data likelihood (nonlinear version of Factor Analysis)



Maximize ELBO

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

VAE – ELBO

- An unbiased gradient estimator of the ELBO w.r.t. the generative model parameters is straightforwardly obtained:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\nabla_{\boldsymbol{\theta}}(\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\right] \\ &\simeq \nabla_{\boldsymbol{\theta}}(\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\end{aligned}$$

- A gradient estimator of the ELBO w.r.t. the variational parameters φ is more difficult to obtain:

$$\begin{aligned}\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) &= \nabla_{\boldsymbol{\phi}}\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right] \\ &\neq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\nabla_{\boldsymbol{\phi}}(\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\right]\end{aligned}$$

Maximize ELBO

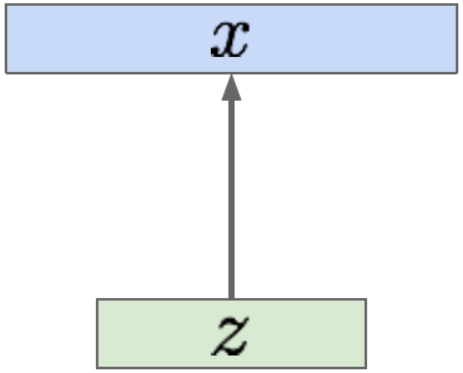
$$\mathcal{L}(\theta,\phi,x) = \mathbb{E}_{q_{\phi}}[\log p_{\theta}(x,z)] - \log q_{\phi}(z|x)]$$

Variational Autoencoders (VAEs)

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation **z**

We want to estimate true parameters θ^* of generative model.

Sample from true conditional $p_{\theta^*}(x \mid z^{(i)})$



Sample from true prior $p_{\theta^*}(z)$

Model representation

- Choose prior $p(z)$ to be simple, e.g. Gaussian.
- Conditional $p(x|z)$ is complex (generates input data) => represent with neural network

Training

- Learn model parameters to maximize likelihood of training data

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Intractable to compute $p(x|z)$ for every z!

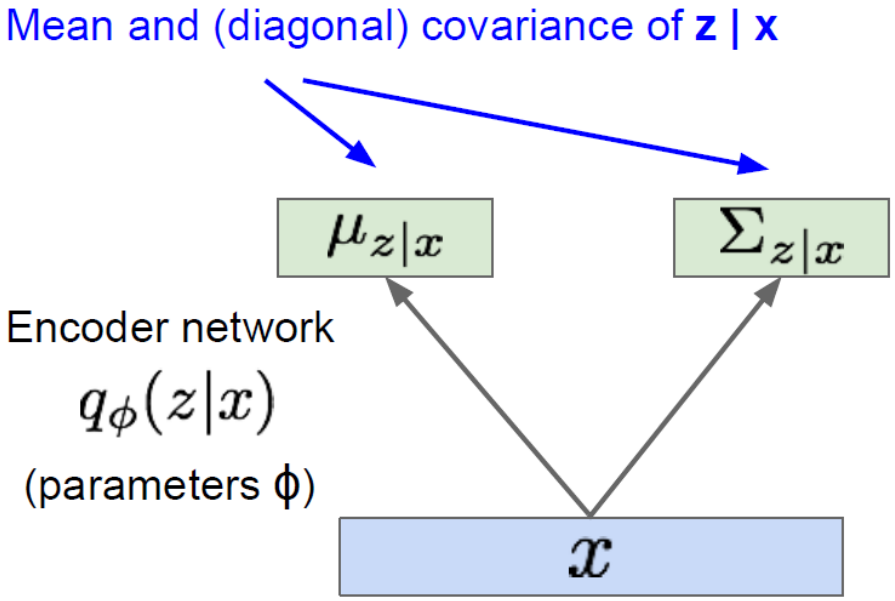
Simple Gaussian prior

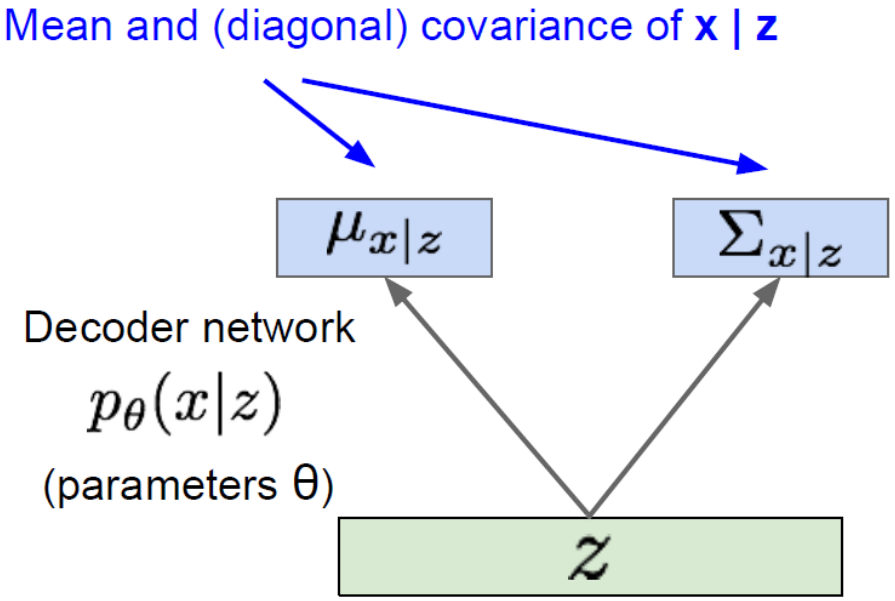
Decoder Neural Network

- Solution: In addition to decoder network modeling $p_\theta(x|z)$, define additional encoder network $q_\phi(z|x)$ that approximates $p_\theta(z|x)$
- This allows us to derive a lower bound on the data likelihood that is tractable

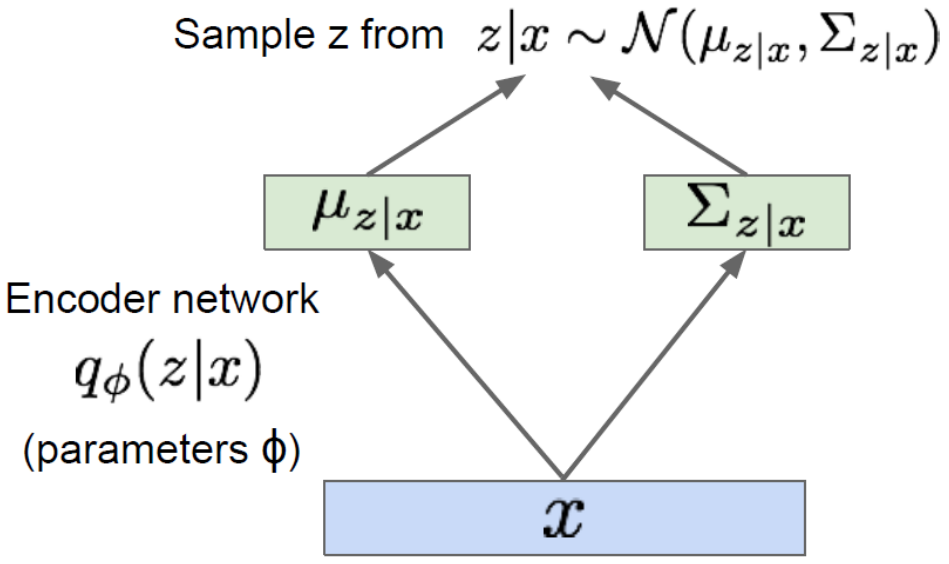
Variational Autoencoders

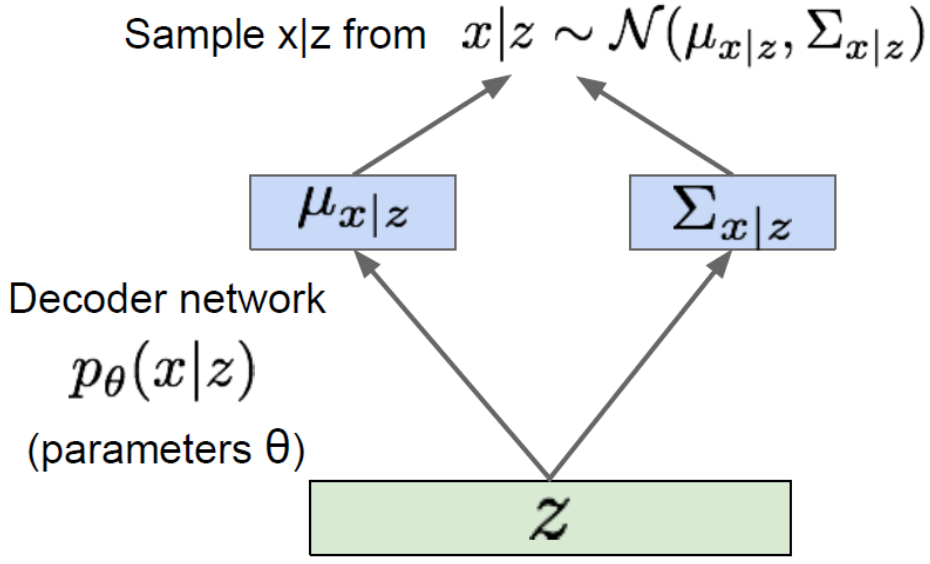
- Since we are modeling probabilistic generation of data, encoder and decoder networks are probabilistic





Variational Autoencoders





- Encoder and decoder networks also called “recognition”/“inference” and “generation” networks

Variational Autoencoders

- Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

Taking expectation wrt. z (using encoder network) will come in handy later

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z \sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z) p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Bayes' Rule})$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z) p_\theta(z)}{p_\theta(z \mid x^{(i)})} \frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Logarithms})$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

The expectation wrt. z (using encoder network) let us write nice KL terms

Variational Autoencoders

- Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling.

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_\theta(z|x)$ intractable, can't compute this KL term

But KL divergence always ≥ 0.

Variational Autoencoders

- Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Bayes' Rule})$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Logarithms})$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

$\mathcal{L}(x^{(i)}, \theta, \phi)$ — ≥ 0

We want to maximize the data likelihood

Tractable lower bound which we can take gradient of and optimize!

($p_\theta(x|z)$ differentiable, KL term differentiable)

Variational Autoencoders

- Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Bayes' Rule})$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Logarithms})$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

We want to maximize the data likelihood

Make approximate posterior distribution close to prior

Reconstruct input data

ELBO (Evidence Lower Bound) or negative free energy → $\mathcal{L}(x^{(i)}, \theta, \phi)$

≥ 0

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

Variational lower bound

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: Maximize lower bound

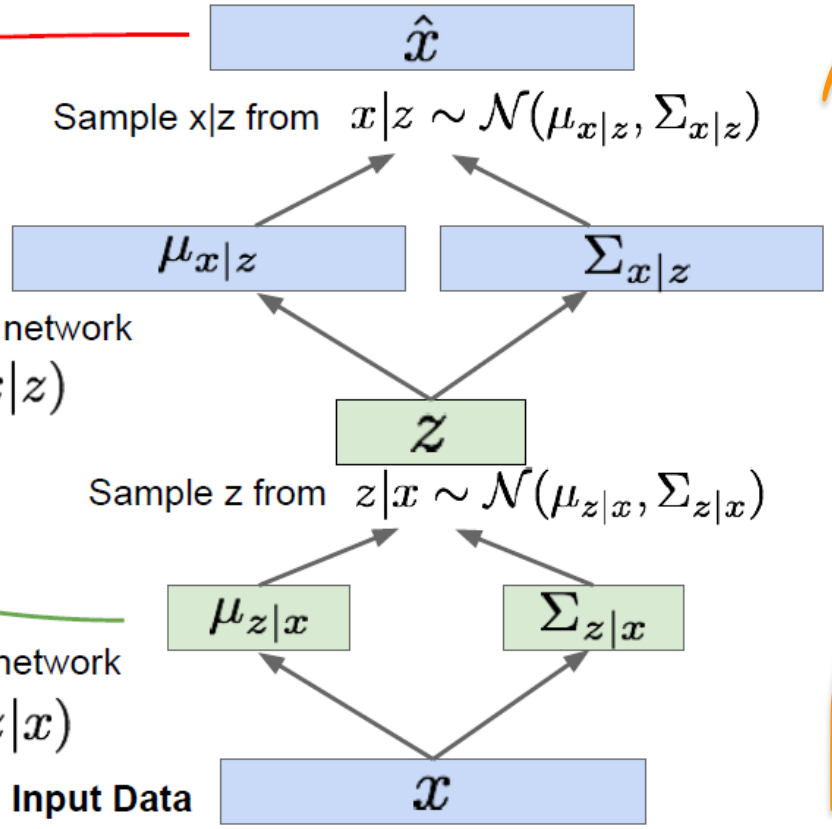
Variational Autoencoders

- Let's look at computing the bound (forward pass) for a given minibatch of input data

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Maximize likelihood of original input being reconstructed



Make approximate posterior distribution close to prior

For every minibatch of input data: compute this forward pass, and then backpropagation

VAEs & Regularity

- Regularity of latent space (for generative process) can be expressed through two main properties:
  1. Continuity: two close points in the latent space should not give two completely different contents once decoded
  2. Completeness: for a chosen distribution, a point sampled from the latent space should give “meaningful” content once decoded

- The only fact that VAEs encode inputs as distributions instead of simple points is not sufficient to ensure continuity and completeness.

- Need for a well defined regularization term: model can learn, in order to minimize its reconstruction error, to “ignore” the fact that distributions are returned and behave almost like classic autoencoders (leading to overfitting).

- So, in order to avoid these effects we have to regularize both the covariance matrix and the mean of the distributions returned by the encoder.

- In practice, this regularization is done by enforcing distributions to be close to a standard normal distribution (centered and reduced).

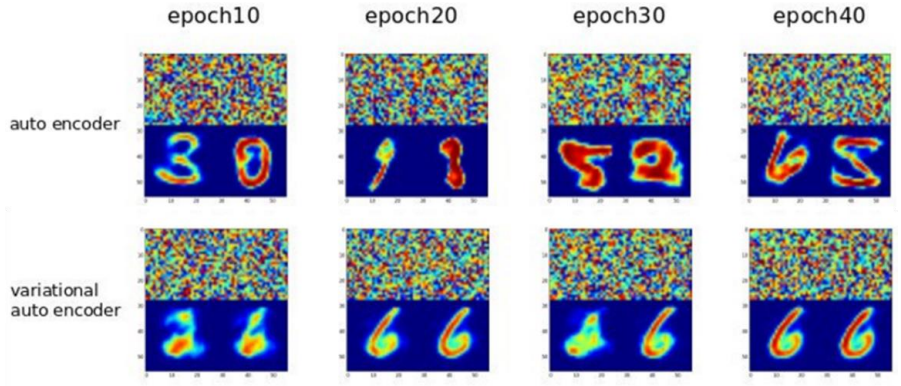
- This way, we require the covariance matrices to be close to the identity, preventing punctual distributions, and the mean to be close to 0, preventing encoded distributions to be too far apart from each others.

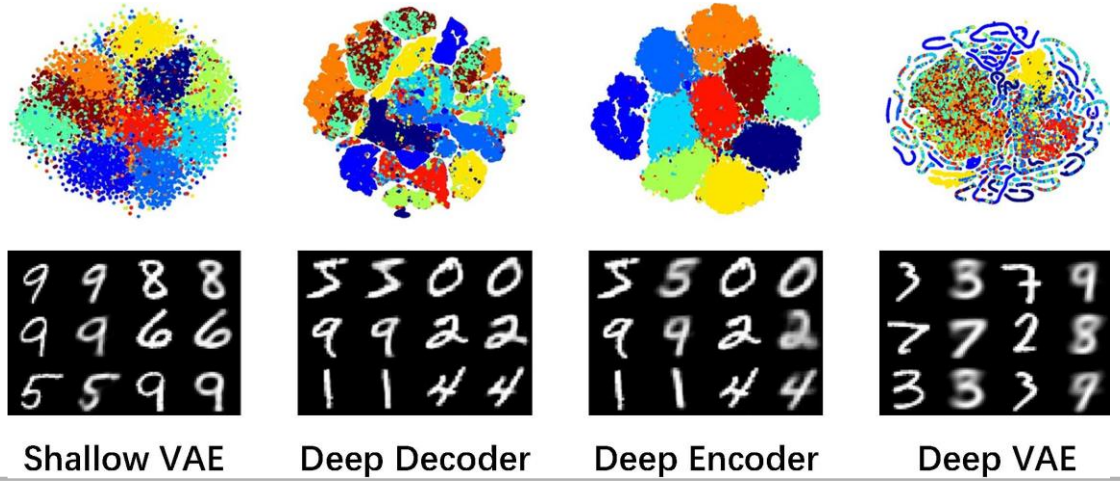
Variational Autoencoders vs. Autoencoders

- VAE is (roughly) an AE regularized to avoid overfitting + latent space has good properties (+interpretable)
- **Architecture**: VAE ≈ AE (encoder and decoder, trained to minimize reconstruction error).
- Difference in **Regularization** of the latent space: instead of encoding an input as a single point, we encode it as a distribution over the latent space.

Model Training

1. Input $\mathbf{x}$ encoded as distribution over the latent space $p(\mathbf{z}|\mathbf{x})$
2. Point from latent space is sampled from that distribution $\mathbf{z} \sim p(\mathbf{z}|\mathbf{x})$
3. Sampled point is decoded and reconstruction error is computed
4. Reconstruction error is backpropagated through the NN





Blurry image problem

- The samples from the VAE look blurry
- Three plausible explanations for this
  - Maximizing the likelihood
  - Restrictions on the family of distributions
  - The lower bound approximation

32x32 image sam

Generated from a DRAW model
(vanilla VAE samples would look even worse and more blurry)

Generated from a VAE trained with IAF

In a Nutshell

- Probabilistic twist to classic autoencoders → allows generating data
- Defines an intractable density → derive and optimize a (variational) lower bound

- Pros:
  - Principled approach to generative models
  - Allows inference of $q(z|x)$, can be useful feature representation for other tasks

- Cons:
  - Not asymptotically consistent unless *q* is perfect
  - Samples blurrier and lower quality compared to state-of-the-art (GANs)

- Active areas of research:
  - More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs)
  - Incorporating structure in latent variables, e.g., Categorical Distributions