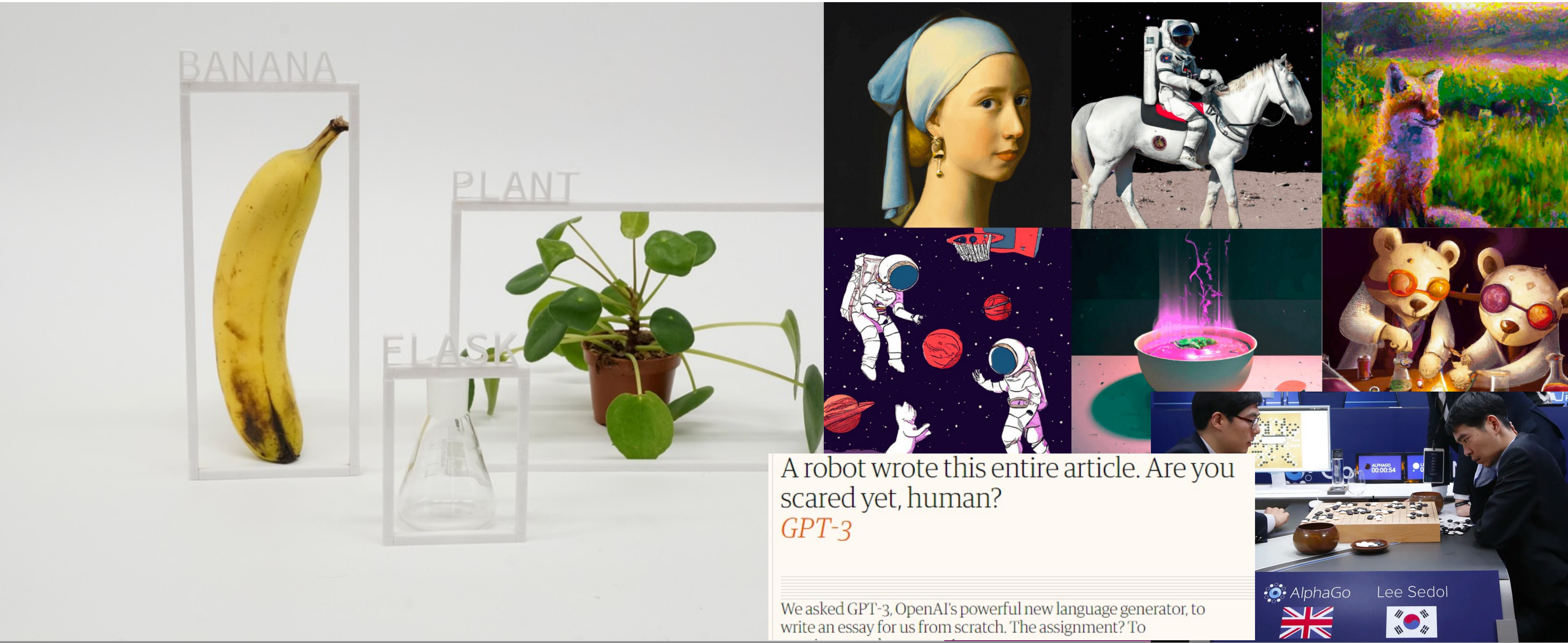


A robot wrote this entire article. Are you scared yet, human?

GPT-3

We asked GPT-3, OpenAI's powerful new language generator, to write an essay for us from scratch. The assignment? To

Lecture 9: Generative modelling and Deep Variational Inference

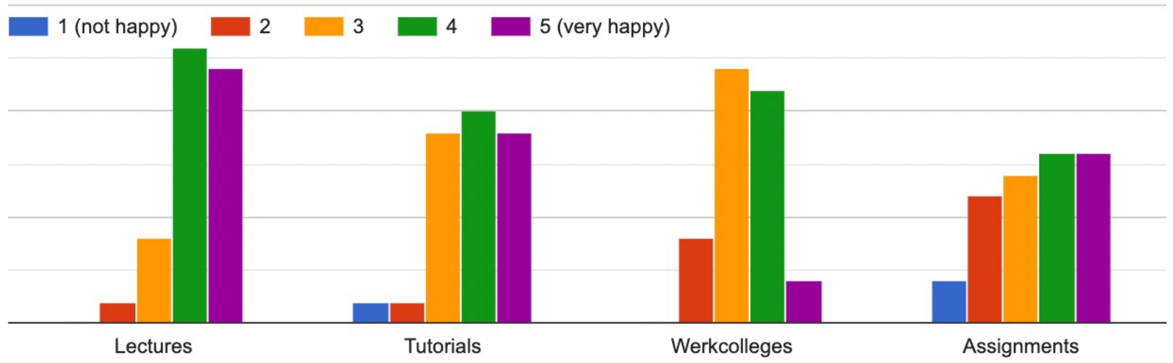
Deep Learning 1 @ UvA
Yuki M. Asano

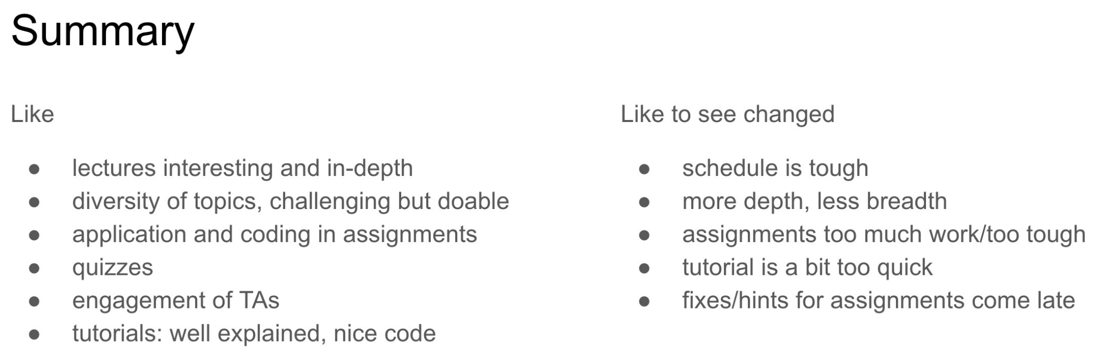
Organisation

- Guest Lecture on 6th December will be remote, we will have
  Prof. Andrea Vedaldi from the University of Oxford
  talking about Recent trends in unsupervised learning for 3D
  https://uva-live.zoom.us/j/6466222109
- Please be there in-person for lecture on the 13th December (teaching recording)

Organisation

- Thanks for filling out mid-way feedback





Summary

Like

- lectures interesting and in-depth
- diversity of topics, challenging but doable
- application and coding in assignments
- quizzes
- engagement of TAs
- tutorials: well explained, nice code

Like to see changed

- schedule is tough
- more depth, less breadth
- assignments too much work/too tough
- tutorial is a bit too quick
- fixes/hints for assignments come late

- Third assignment is being edited to be lighter load
- Schedule: we understand. Aim to accommodate this esp. in teaching after lectures
- More “depth”: Lecture 8, 9, 10: are indeed more depth than breadth.
- Many more individual points received & taken into account. Thanks again.

Lecture overview

Generative Modelling

Autoencoders

Latent variable models

Variational inference

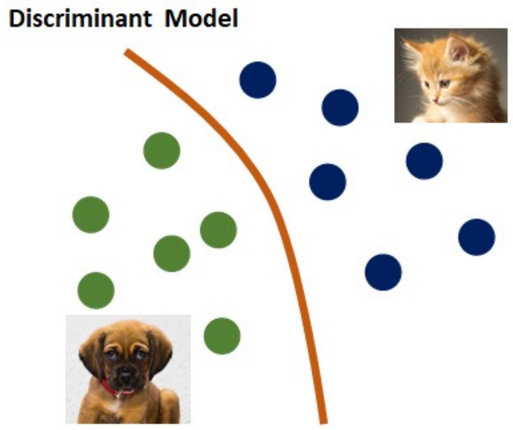
Variational autoencoders

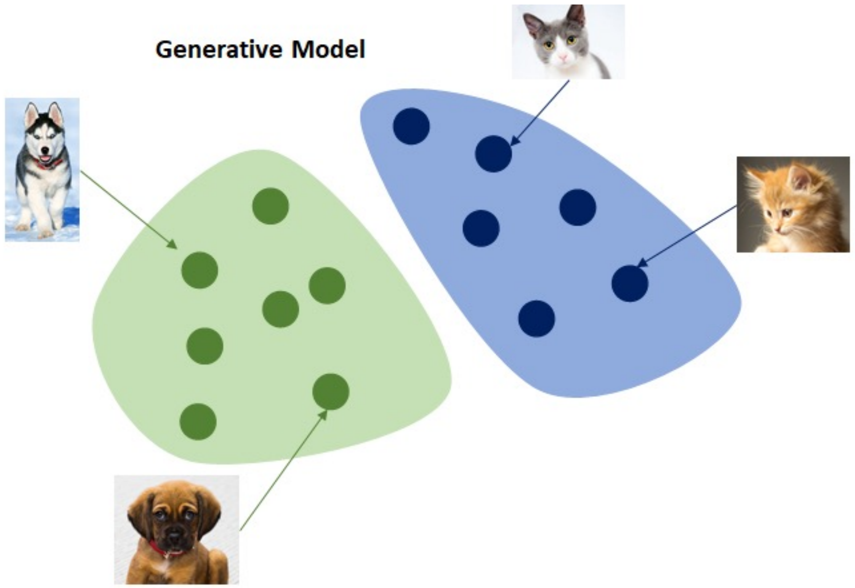
Inference suboptimality

VAE variants

What is generative modelling?

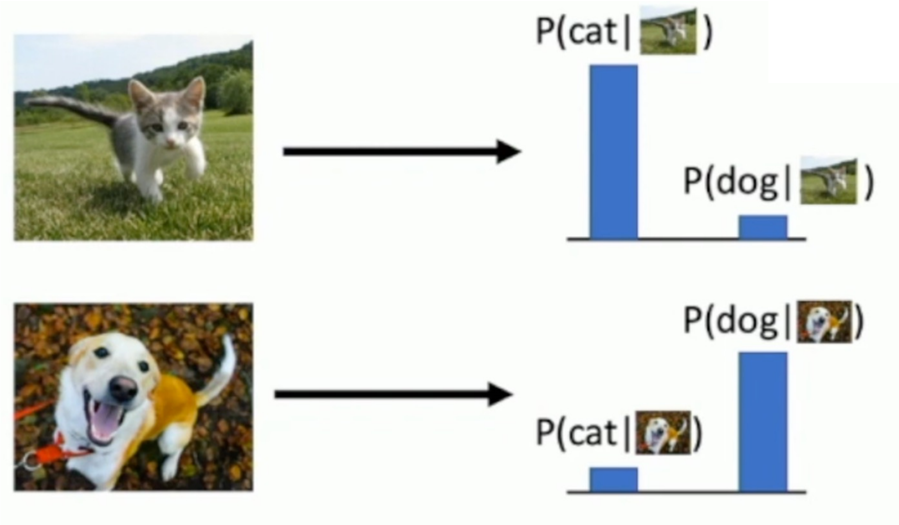
- Often in ML we talk about discriminative vs generative modelling
  - p(y|x) vs p(x)
  - p is a probability density function: high means x is *likely*.
  - p is normalized: $\int p(x)dx = 1$

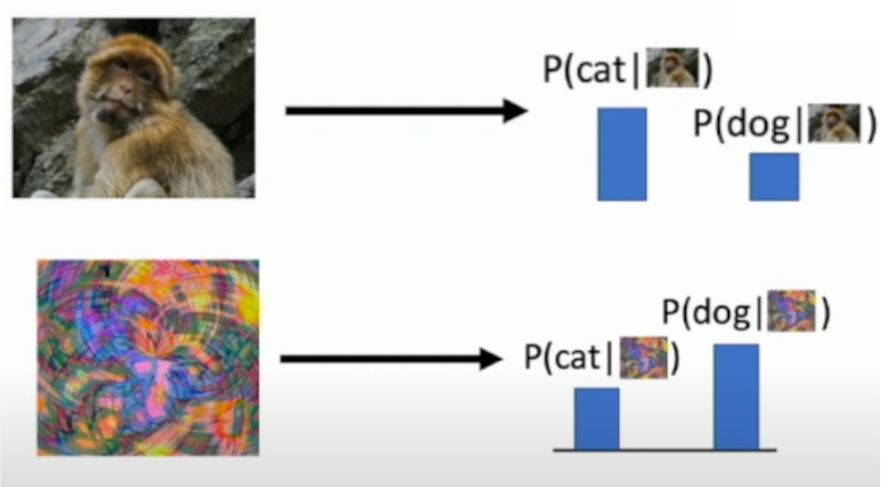




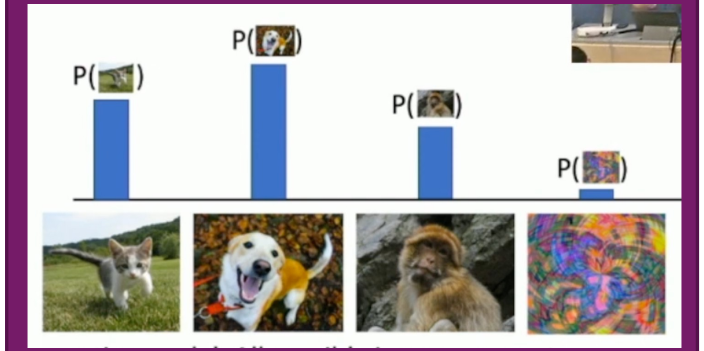
Why generative modelling?

Discriminative model: p is normalized for *outputs*, but not for inputs:



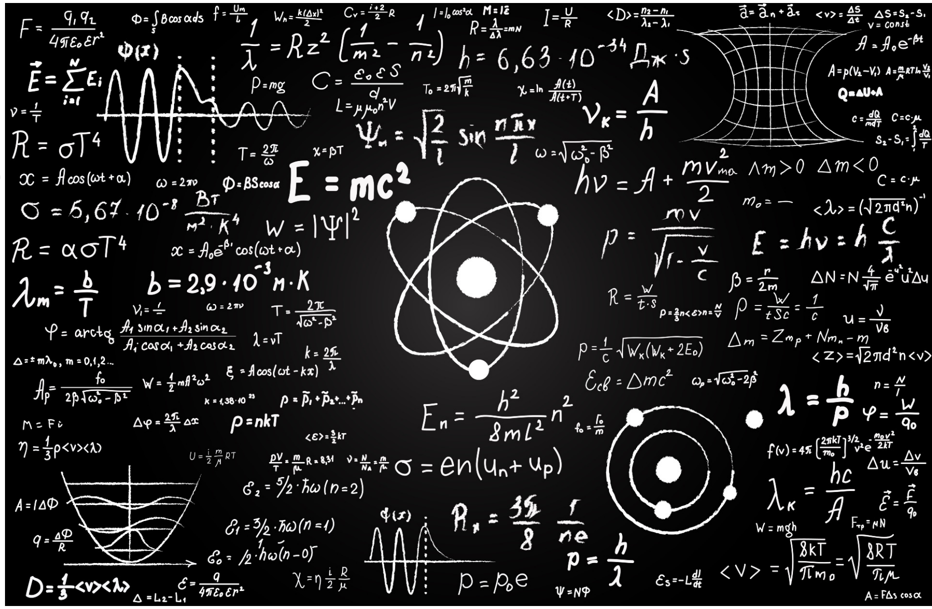


Generative model:
p is normalized for inputs



Why generative modelling? More reasons

- Modelling in other disciplines: aims to capture underlying processes
  - Physics (Newton's law --> planet motions)
  - Economics (assumptions + free parameters that are fitted --> forecasts)
  - Mathematics, biology, geology, …

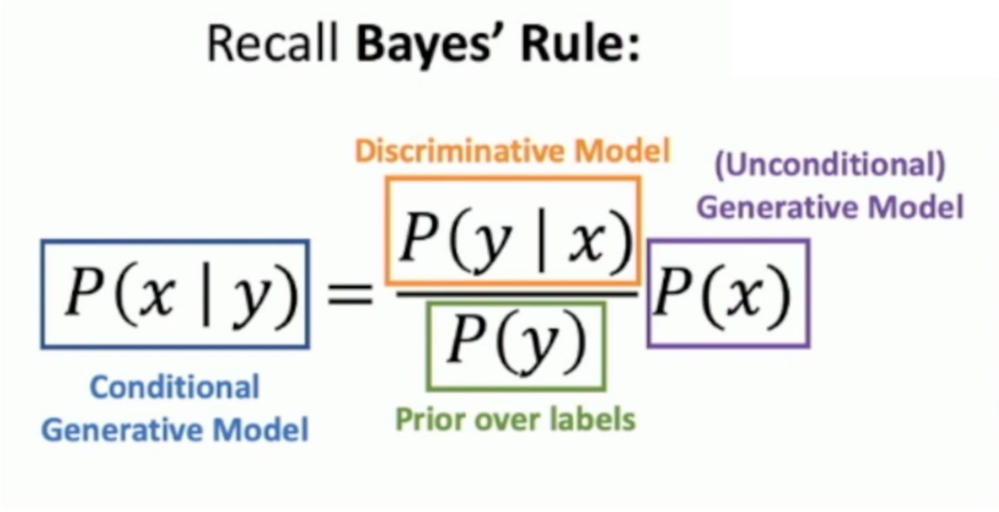


- Benefits
  - Interpretable: because models are built in a manner easy to understand
  - Testable: if it fits data and is simple it's likely to be good (c.f. Occam's razor)
  - Can help with discriminative models: e.g. more robust predictions
  - For vision/language: "might" be similar to brain (predictive coding)

Bayes rule: if we have generative models: we have it all

- We can get a conditional generative model for free once we have a generative model
- Note:
  - Previously we focussed on learning $p(y|x)$
  - What we will do today and Friday: $p_\theta(x)$
- If true distribution of data is $p^*(x)$

  we wish to learn $\theta \ s.t.: \ p_\theta(x) \approx p^*(x)$

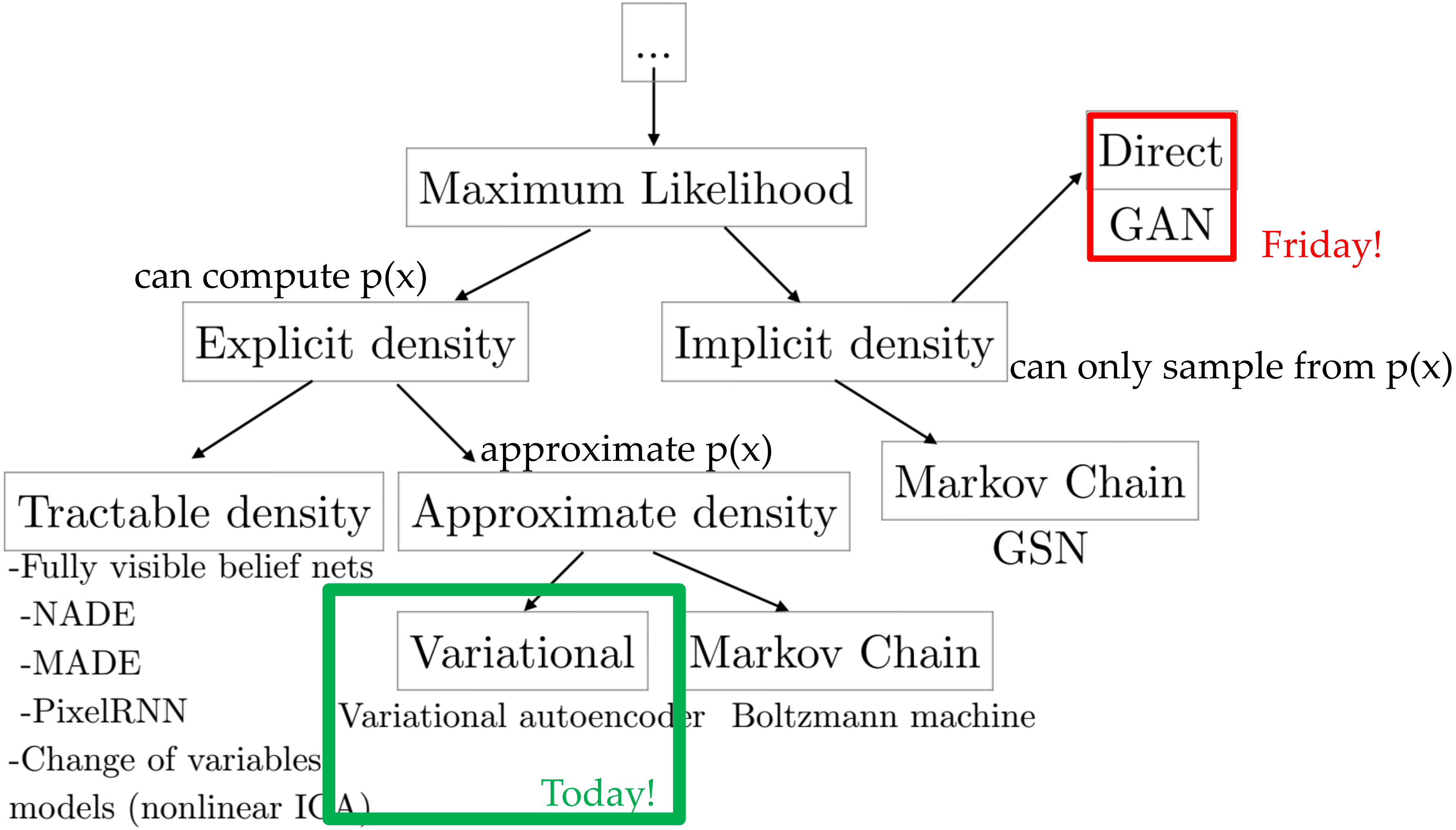


Recall **Bayes' Rule:**

$$\boxed{P(x \mid y)} = \frac{\boxed{P(y \mid x)}}{\boxed{P(y)}}\boxed{P(x)}$$

Discriminative Model
(Unconditional) Generative Model
Conditional Generative Model
Prior over labels

A map of generative models

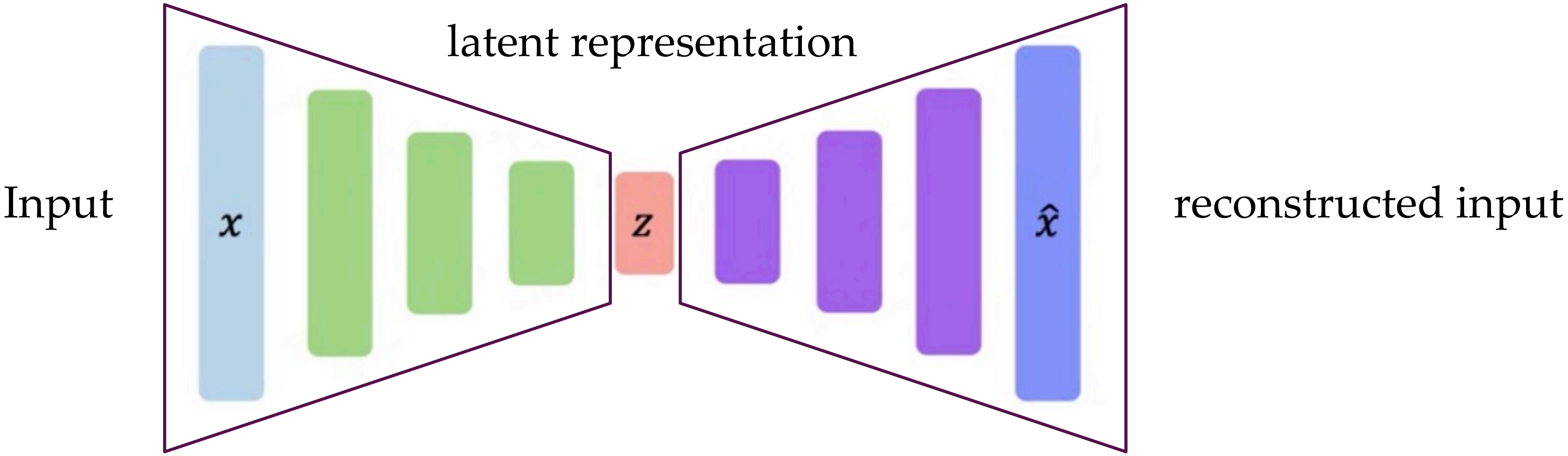


Variational Autoencoders

Autoencoders

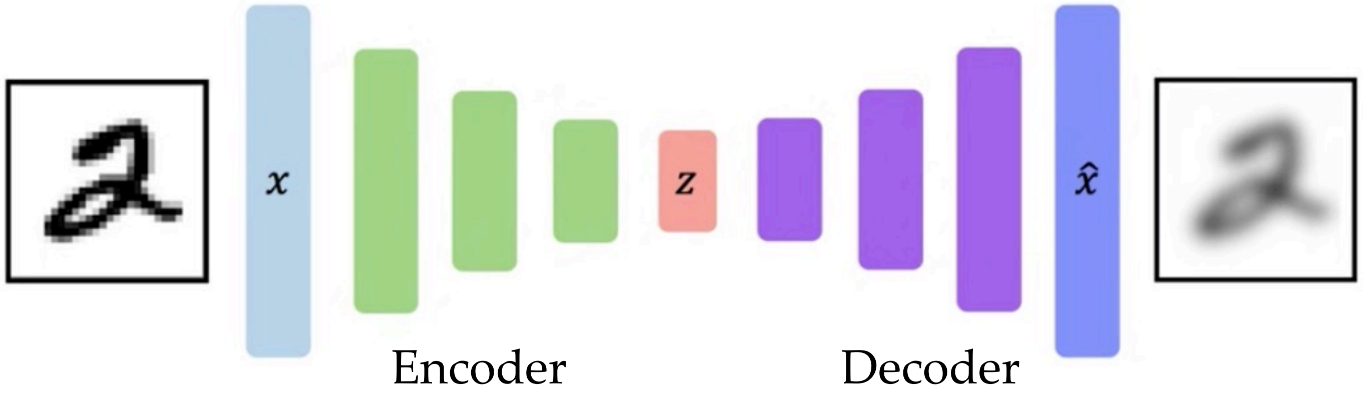
Feedforward network with a bottleneck layer of fewer dimensions than the input

- Output layer with the same dimensionality as the input
- An autoencoder is used to learn efficient codings of unlabelled data



Autoencoders

An autoencoder has an encoder and a decoder



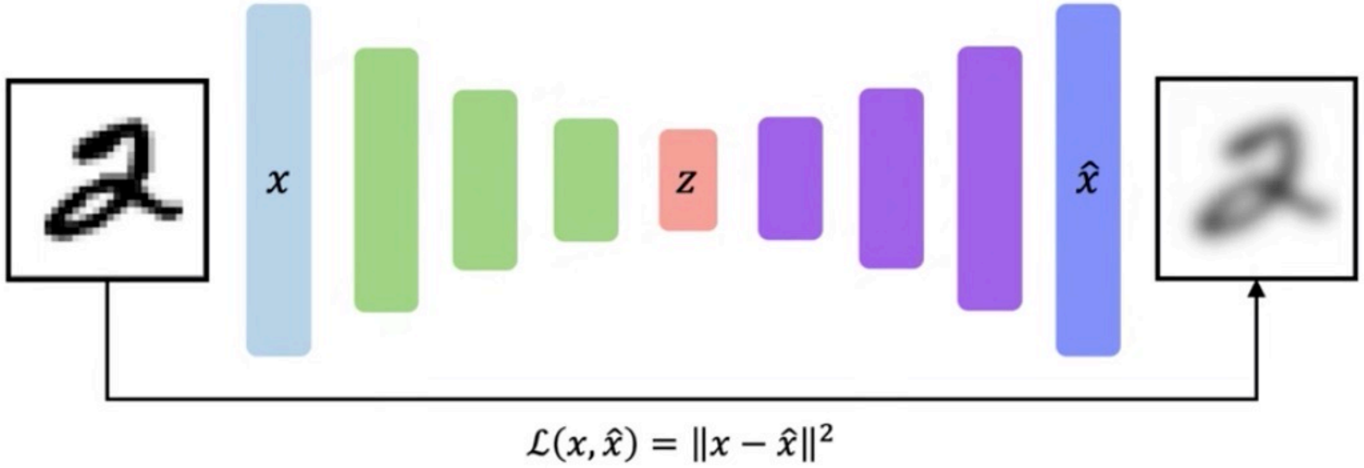
Encoder learns mapping from the data x to a low dimensional latent space

Decoder learns mapping back from latent space z to a reconstructed observation x

By itself, we have no assumptions about the architecture (in practice: CNNs with transposed convolutions)

Autoencoders

Train the model to use latent features to reconstruct the original data



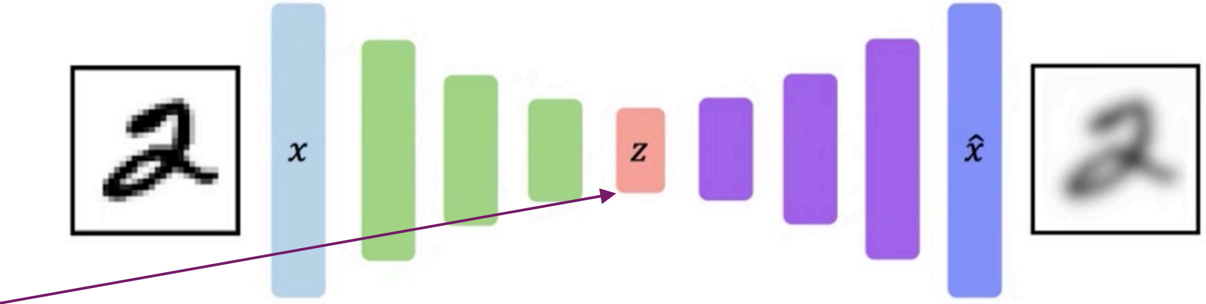
Minimizing the difference between the input and the output (e.g. L2 loss per pixel)

Loss function does not use any labels

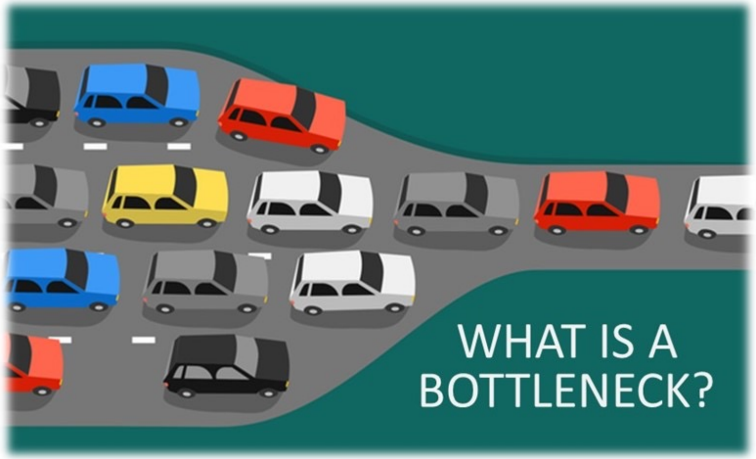
Autoencoders: why though?

WHY?

- we don't need to wait 8hours on Lisa just to compute a not-very-good identity function…
- the key is the "bottleneck"
- which is of lower dimension than the input
- thus it learns a compressed representation



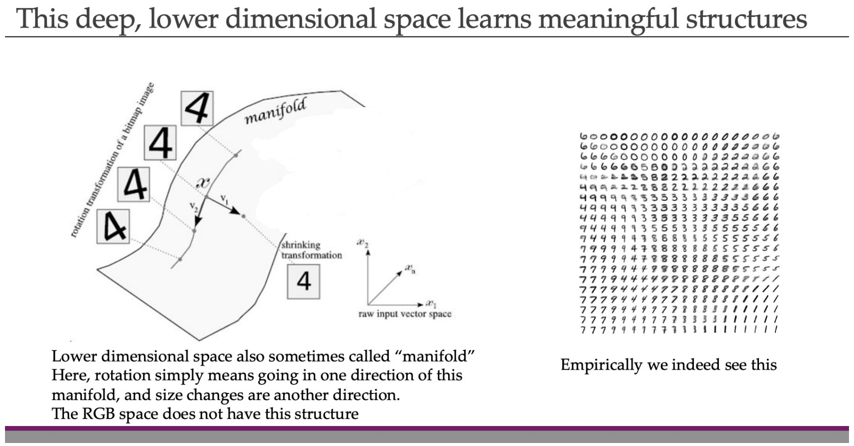
A not so useful bottleneck



Autoencoders for representation learning (ie use the encoder afterwards)

Learning lower-dimensional feature representations

- an unsupervised learning technique
- Remember --------------------->



This deep, lower dimensional space learns meaningful structures

Lower dimensional space also sometimes called "manifold"
Here, rotation simply means going in one direction of this manifold, and size changes are another direction.
The RGB space does not have this structure

Empirically we indeed see this

Bottleneck hidden layer

- encourages learning compressed representations (n.b.: compression is very related to learning)
- removes redundant information from the raw input

Reconstruction loss

- forces the latent representation to capture as much information about the data as possible

Autoencoders for representation learning

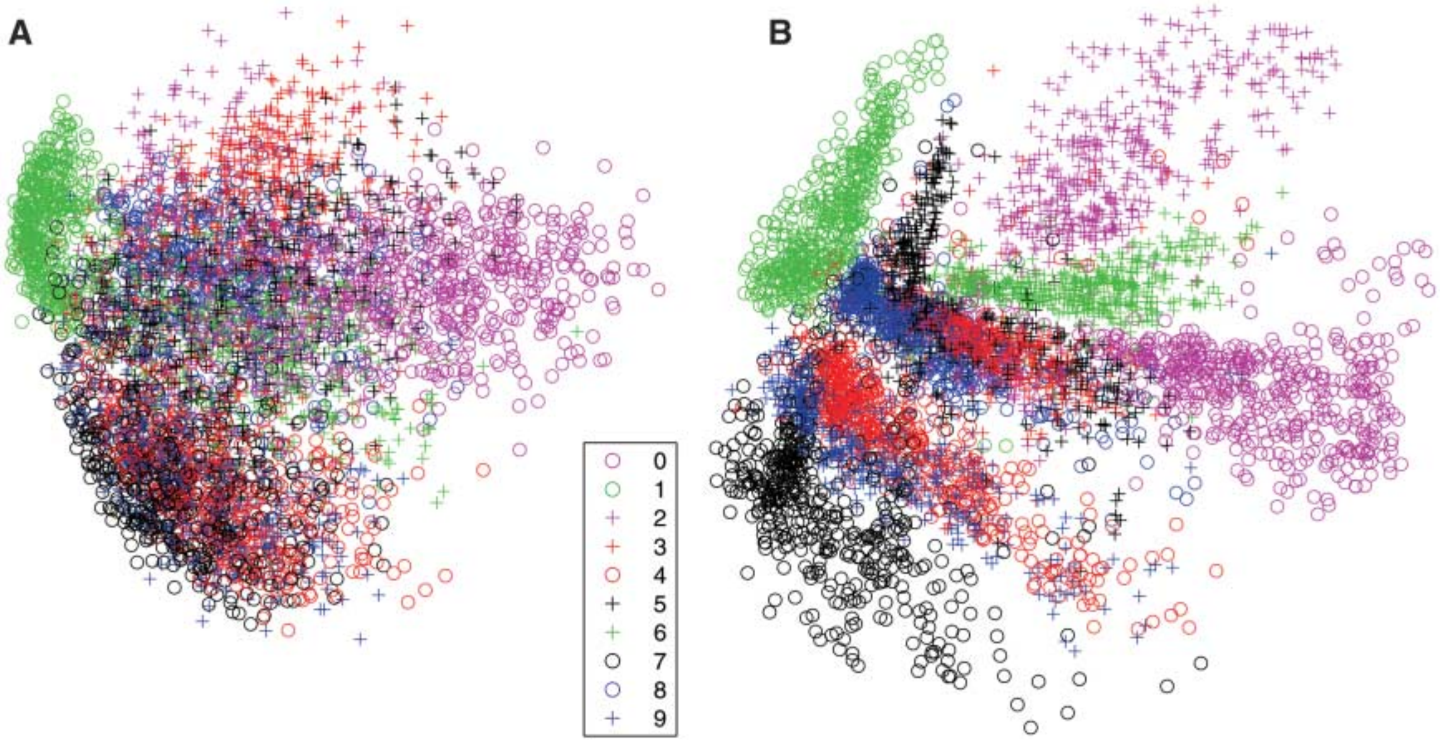


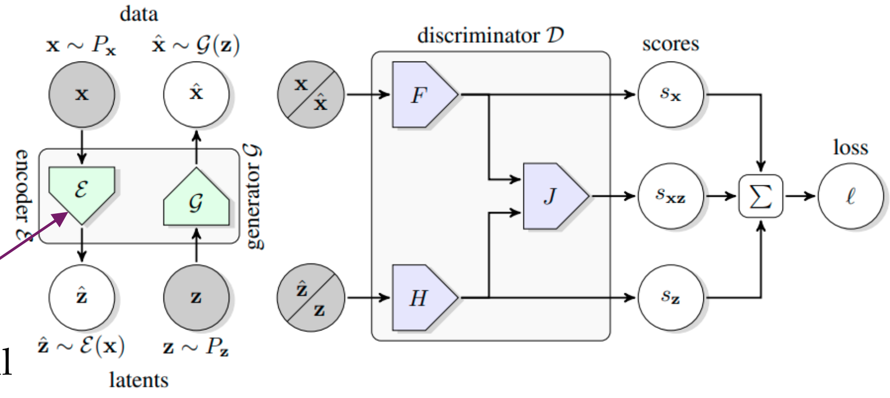
Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).

PCA

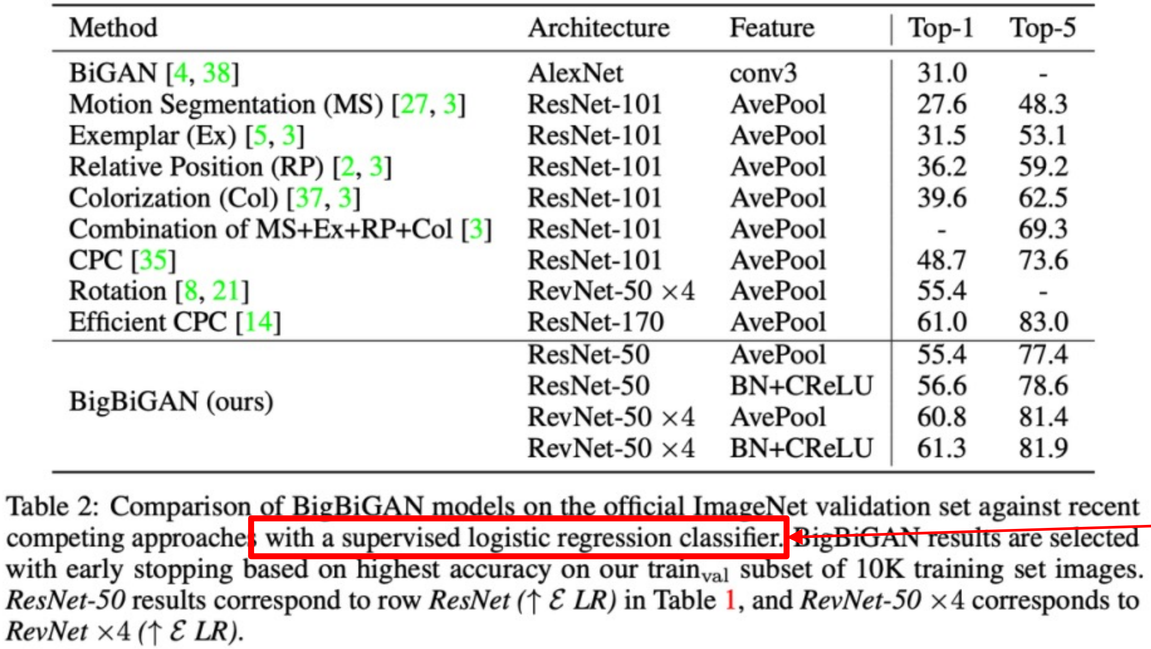
Autoencoder

2006 Science paper by Hinton and Salakhutdinov

“Autoencoders“ for representation learning (13years later): BigBiGAN


data
$\mathbf{x} \sim P_{\mathbf{x}}$
$\hat{\mathbf{x}} \sim \mathcal{G}(\mathbf{z})$
$\mathbf{x}$
$\hat{\mathbf{x}}$
encoder $\mathcal{E}$
$\mathcal{E}$
$\mathcal{G}$
generator $\mathcal{G}$
$\hat{\mathbf{z}}$
$\mathbf{z}$
$\hat{\mathbf{z}} \sim \mathcal{E}(\mathbf{x})$
$\mathbf{z} \sim P_{\mathbf{z}}$
latents
discriminator $\mathcal{D}$
$\mathbf{x}$ / $\hat{\mathbf{x}}$
$\hat{\mathbf{z}}$ / $\mathbf{z}$
F
J
H
scores
$s_{\mathbf{x}}$
$s_{\mathbf{xz}}$
$s_{\mathbf{z}}$
$\sum$
loss
ℓ

additional



Method	Architecture	Feature	Top-1	Top-5
BiGAN [4, 38]	AlexNet	conv3	31.0	-
Motion Segmentation (MS) [27, 3]	ResNet-101	AvePool	27.6	48.3
Exemplar (Ex) [5, 3]	ResNet-101	AvePool	31.5	53.1
Relative Position (RP) [2, 3]	ResNet-101	AvePool	36.2	59.2
Colorization (Col) [37, 3]	ResNet-101	AvePool	39.6	62.5
Combination of MS+Ex+RP+Col [3]	ResNet-101	AvePool	-	69.3
CPC [35]	ResNet-101	AvePool	48.7	73.6
Rotation [8, 21]	RevNet-50 ×4	AvePool	55.4	-
Efficient CPC [14]	ResNet-170	AvePool	61.0	83.0
BigBiGAN (ours)	ResNet-50	AvePool	55.4	77.4
	ResNet-50	BN+CReLU	56.6	78.6
	RevNet-50 ×4	AvePool	60.8	81.4
	RevNet-50 ×4	BN+CReLU	61.3	81.9

Table 2: Comparison of BigBiGAN models on the official ImageNet validation set against recent competing approaches with a supervised logistic regression classifier. BigBiGAN results are selected with early stopping based on highest accuracy on our train$_{\text{val}}$ subset of 10K training set images. *ResNet-50* results correspond to row *ResNet* (↑ $\mathcal{E}$ *LR*) in Table 1, and *RevNet-50* ×4 corresponds to *RevNet* ×4 (↑ $\mathcal{E}$ *LR*).

The linear probing/evaluation protocol mentioned in lecture 6: freeze network, train linear layer

--> Encoder is general purpose

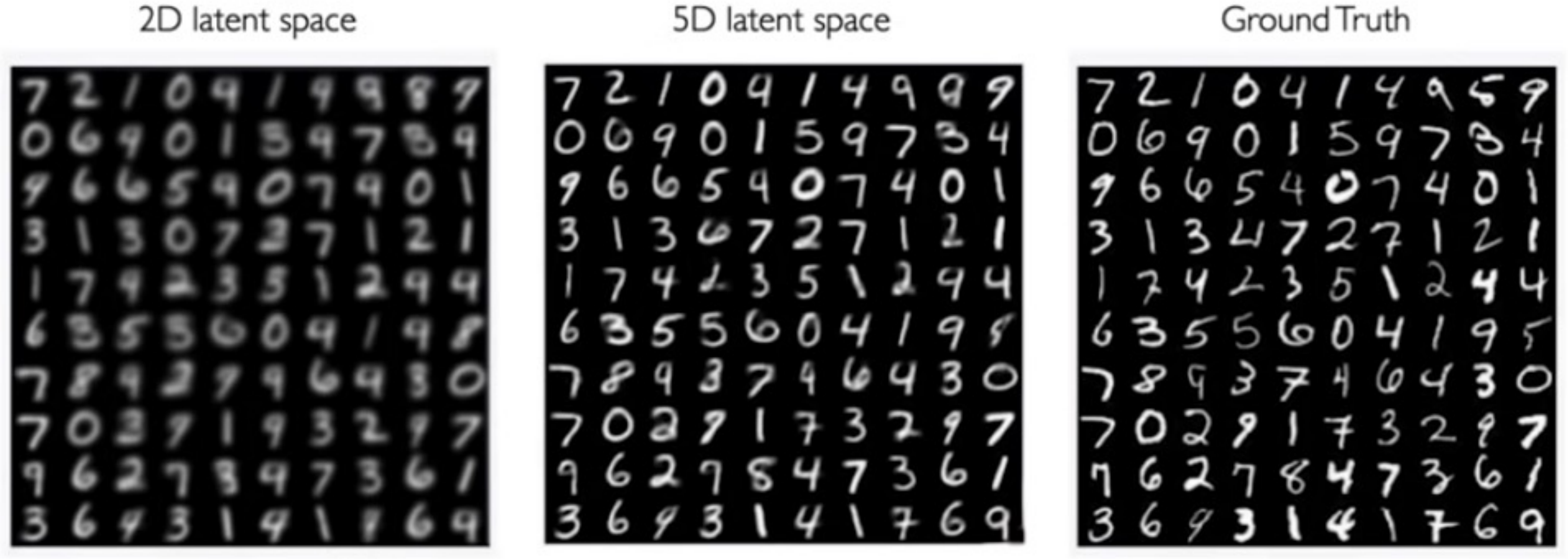
Generated images:

Large Scale Adversarial Representation Learning. Donahue et al. NeurIPS 2019

Dimensionality of latent space

Lower latent dimension ⟶ poorer reconstruction

Too high latent dimension ⟶ keeps irrelevant/noisy information



2D latent space

5D latent space

Ground Truth

Quiz:
PCA is a motivating technique behind the first initial AutoEncoders. What is true?

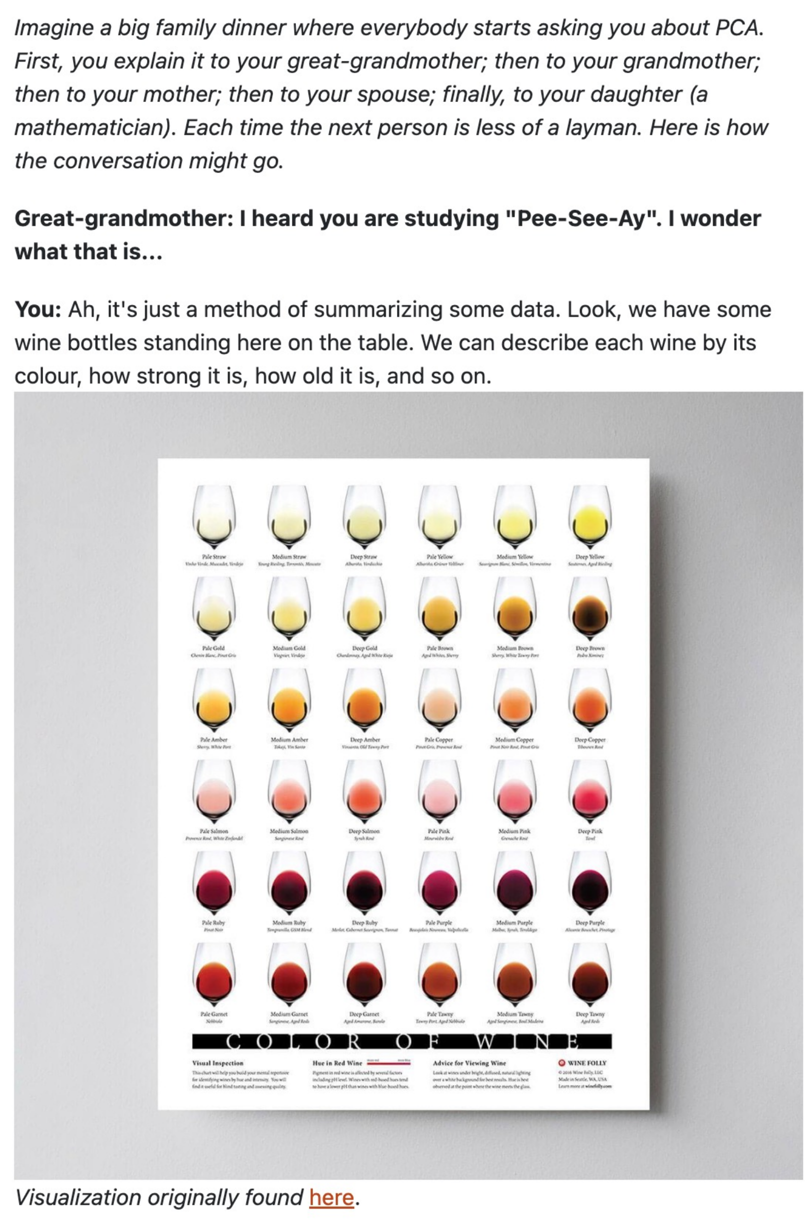
1)PCA can have local optima and should be run multiple times

2)PCA generally returns more interpretable features

3)PCA generally yields the same results as linear regression

4)PCA generally struggles with outliers

PCA refresher 🍷

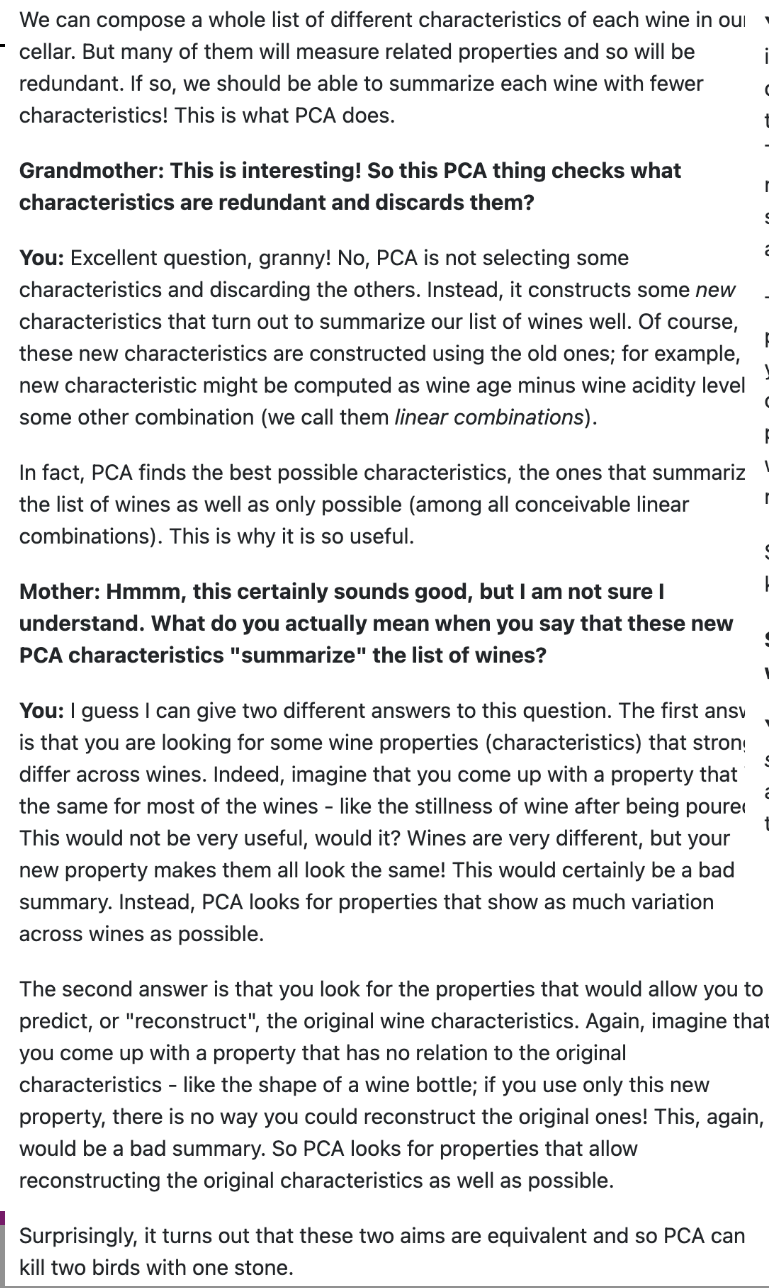


Imagine a big family dinner where everybody starts asking you about PCA. First, you explain it to your great-grandmother; then to your grandmother; then to your mother; then to your spouse; finally, to your daughter (a mathematician). Each time the next person is less of a layman. Here is how the conversation might go.

Great-grandmother: I heard you are studying "Pee-See-Ay". I wonder what that is...

You: Ah, it's just a method of summarizing some data. Look, we have some wine bottles standing here on the table. We can describe each wine by its colour, how strong it is, how old it is, and so on.

Visualization originally found here.



We can compose a whole list of different characteristics of each wine in ou cellar. But many of them will measure related properties and so will be redundant. If so, we should be able to summarize each wine with fewer characteristics! This is what PCA does.

Grandmother: This is interesting! So this PCA thing checks what characteristics are redundant and discards them?

You: Excellent question, granny! No, PCA is not selecting some characteristics and discarding the others. Instead, it constructs some *new* characteristics that turn out to summarize our list of wines well. Of course, these new characteristics are constructed using the old ones; for example, new characteristic might be computed as wine age minus wine acidity level some other combination (we call them *linear combinations*).

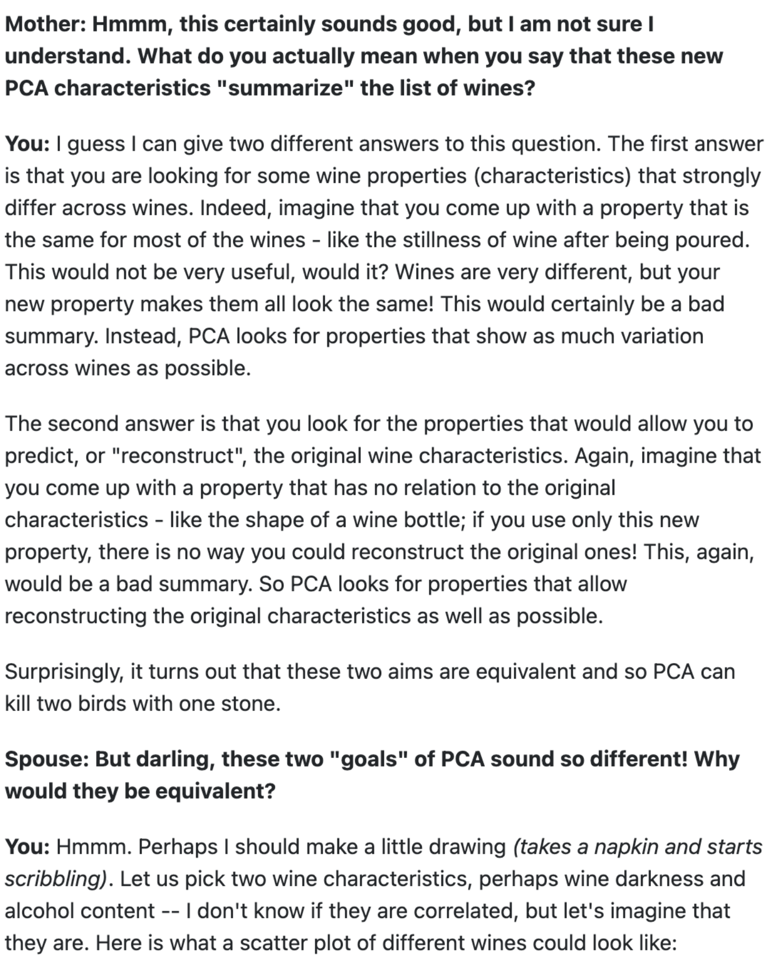
In fact, PCA finds the best possible characteristics, the ones that summariz the list of wines as well as only possible (among all conceivable linear combinations). This is why it is so useful.

Mother: Hmmm, this certainly sounds good, but I am not sure I understand. What do you actually mean when you say that these new PCA characteristics "summarize" the list of wines?

You: I guess I can give two different answers to this question. The first answ is that you are looking for some wine properties (characteristics) that stron differ across wines. Indeed, imagine that you come up with a property that the same for most of the wines - like the stillness of wine after being poure This would not be very useful, would it? Wines are very different, but your new property makes them all look the same! This would certainly be a bad summary. Instead, PCA looks for properties that show as much variation across wines as possible.

The second answer is that you look for the properties that would allow you to predict, or "reconstruct", the original wine characteristics. Again, imagine that you come up with a property that has no relation to the original characteristics - like the shape of a wine bottle; if you use only this new property, there is no way you could reconstruct the original ones! This, again, would be a bad summary. So PCA looks for properties that allow reconstructing the original characteristics as well as possible.

Surprisingly, it turns out that these two aims are equivalent and so PCA can kill two birds with one stone.



Mother: Hmmm, this certainly sounds good, but I am not sure I understand. What do you actually mean when you say that these new PCA characteristics "summarize" the list of wines?

You: I guess I can give two different answers to this question. The first answer is that you are looking for some wine properties (characteristics) that strongly differ across wines. Indeed, imagine that you come up with a property that is the same for most of the wines - like the stillness of wine after being poured. This would not be very useful, would it? Wines are very different, but your new property makes them all look the same! This would certainly be a bad summary. Instead, PCA looks for properties that show as much variation across wines as possible.

The second answer is that you look for the properties that would allow you to predict, or "reconstruct", the original wine characteristics. Again, imagine that you come up with a property that has no relation to the original characteristics - like the shape of a wine bottle; if you use only this new property, there is no way you could reconstruct the original ones! This, again, would be a bad summary. So PCA looks for properties that allow reconstructing the original characteristics as well as possible.

Surprisingly, it turns out that these two aims are equivalent and so PCA can kill two birds with one stone.

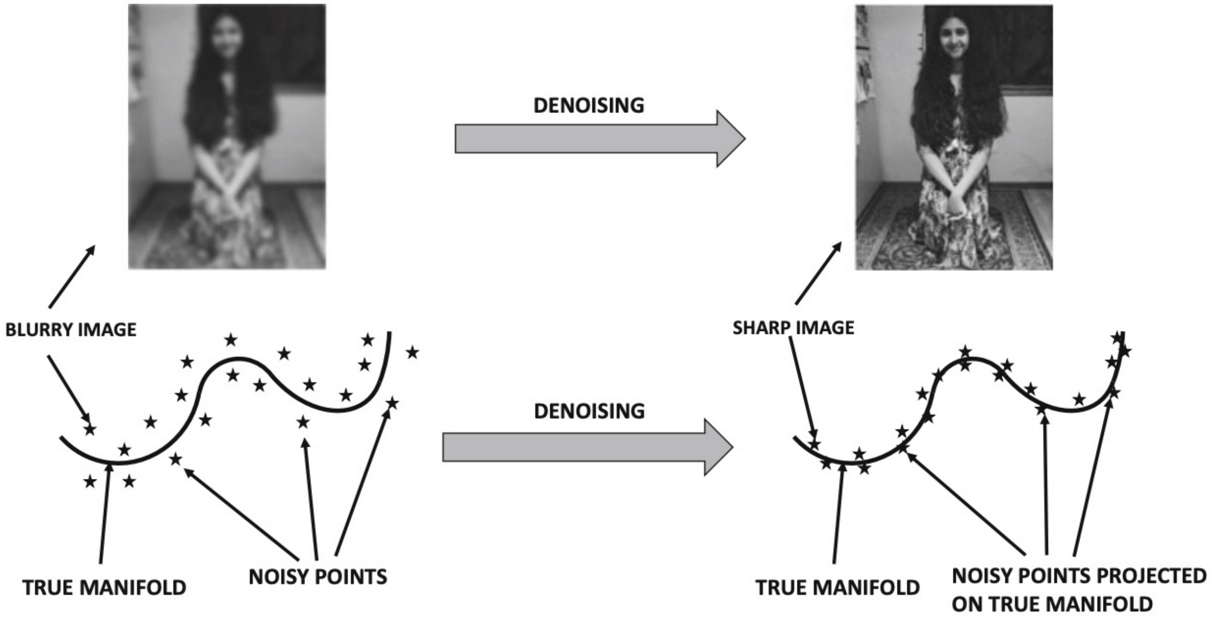
Spouse: But darling, these two "goals" of PCA sound so different! Why would they be equivalent?

You: Hmmm. Perhaps I should make a little drawing *(takes a napkin and starts scribbling)*. Let us pick two wine characteristics, perhaps wine darkness and alcohol content -- I don't know if they are correlated, but let's imagine that they are. Here is what a scatter plot of different wines could look like:

Read it in full here: https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues

Denoising autoencoders

- Add some noise to the input: $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \varepsilon$
- Then train autoencoder to reconstruct original input $\boldsymbol{x}$.
- "Denoising"; ~ augmentation-invariance
- Requires features that capture useful structure in the input distribution.
- Resulting latent representations: stabler against corruption of the input.
- Can be used for, e.g., old photographs

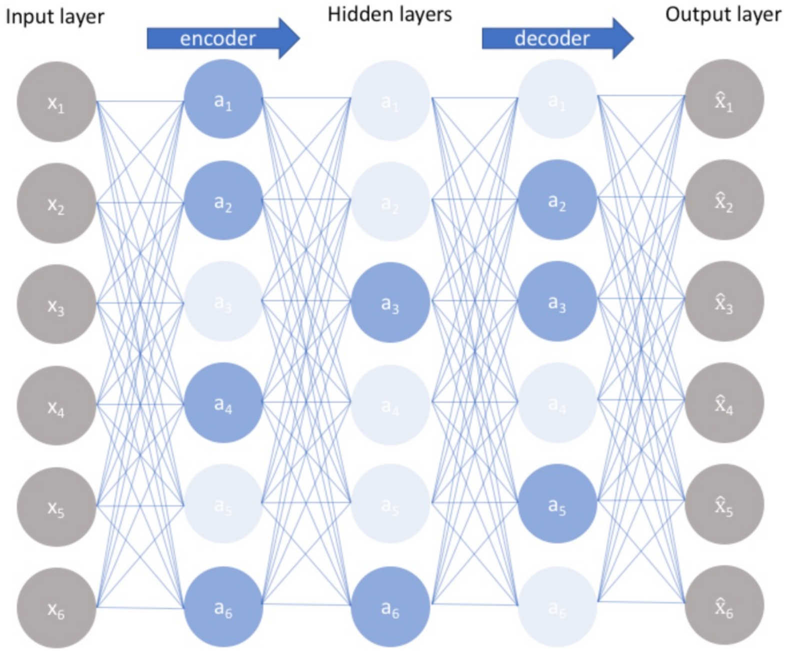


Sparse autoencoders

- Include more hidden units than input ("overcomplete")
- Impose sparsity constraints, e.g., l_1, on activations in the hidden layer

$$\mathcal{L}(x, x') + \Omega(h)$$

- Or use only top-k units
- An alternative way to introducing bottleneck



Autoencoder applications

Autoencoders mostly used for dimensionality reduction

- In fact, linear autoencoders with W_in.T == W_out learn the same subspace as PCA

Autoencoders are a generic machine learning principle: *use what you have*

No need for prior knowledge about invariances/augmentations

Some applications in denoising and (a bit of) representation learning

But: **Autoencoders are not probabilistic and not generative models**

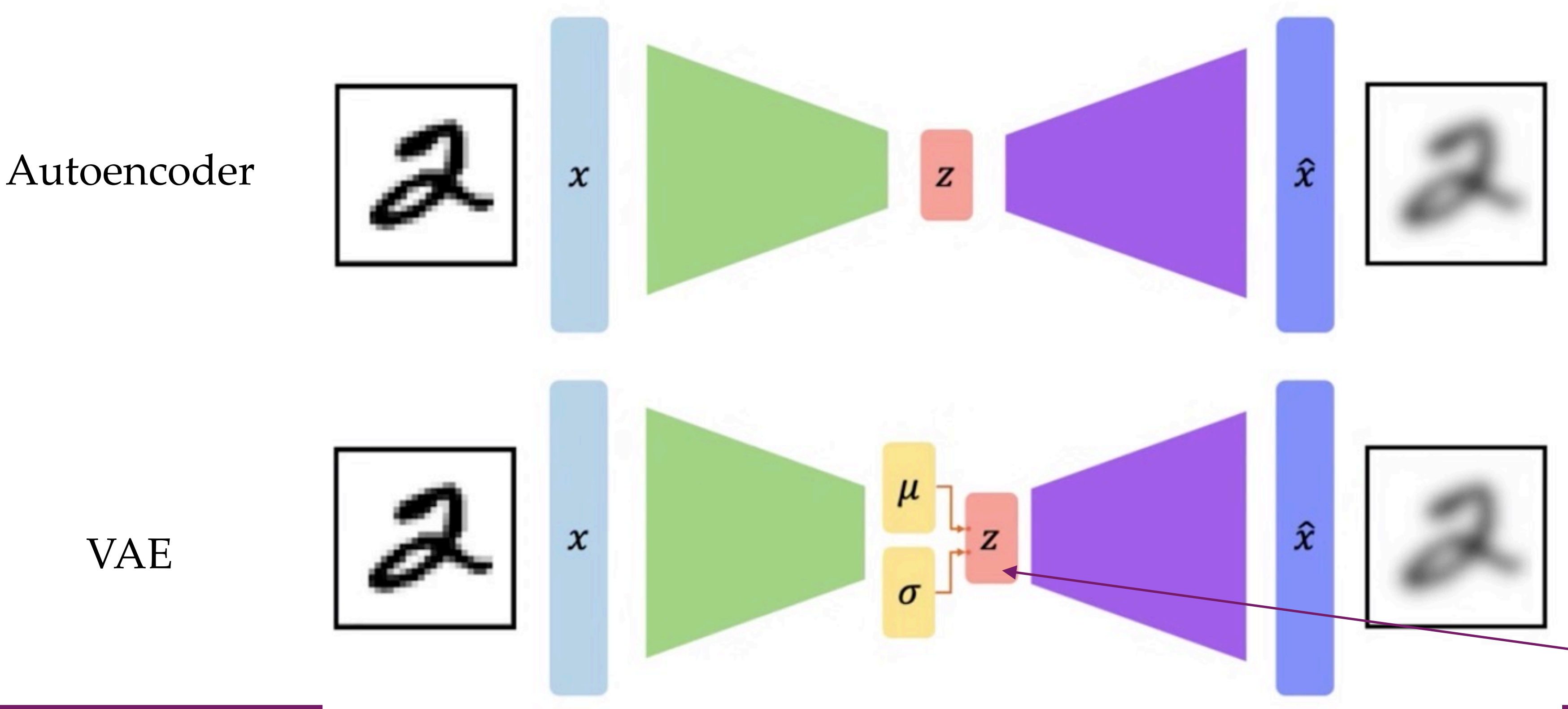
- They can reconstruct the input
- They cannot generate new data points
- They cannot tell you how probable one image is

What we will arrive at in this lecture

The Variational Autoencoder (VAE) will solve those issues.

Learn latent features z from data; sample x from model p(z) to yield p(x|z)

p(z) often simply assumed to be Gaussian.



z is a latent variable

Latent variable models

A *latent variable model* defines a distribution over observations x by using a latent (unobserved) variable z

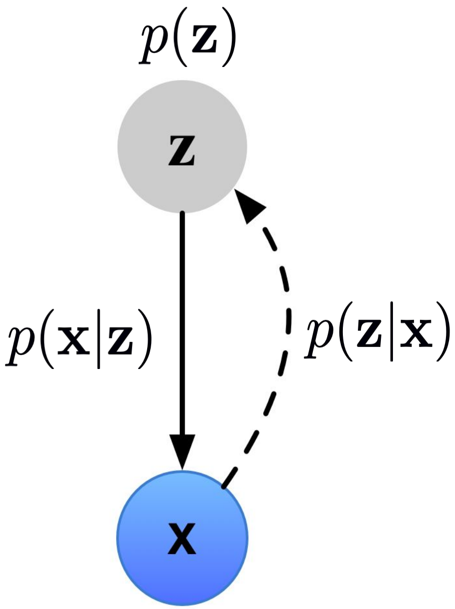
We specify

- The prior distribution $p(z)$ for the latent variable
- The likelihood $p(x|z)$ that connects the latent variable to the observation

The prior and the likelihood define the joint distribution

$$p(x, z) = p(x|z)p(z)$$

We will be interested in computing the marginal likelihood $p(x)$ and the posterior distribution $p(z|x)$.



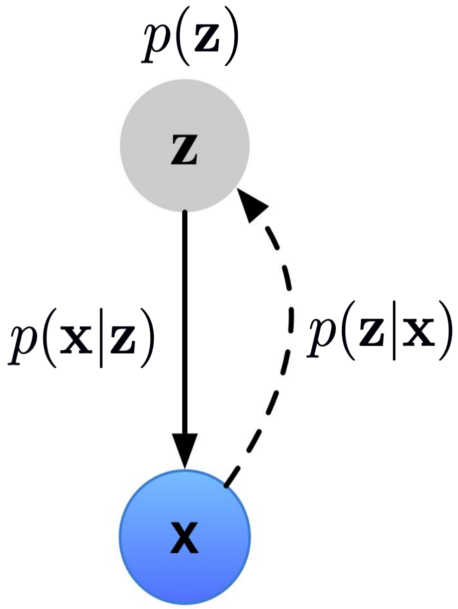
Likelihood vs probability. A critical difference between probability and likelihood is in the interpretation of what is fixed and what can vary. In the case of a conditional probability, P(D|H), the hypothesis is fixed and the data are free to vary. Likelihood, however, is the opposite. The likelihood of a hypothesis, L(H), is conditioned on the data, as if they are fixed while the hypothesis can vary. The distinction is subtle, so it is worth repeating: For conditional probability, the hypothesis is treated as a given, and the data are free to vary. For likelihood, the data are treated as a given, and the hypothesis varies. (from Introduction to the Concept of Likelihood and Its Applications)

Latent variable models

Think of latent variable value as explaining the observation.

To generate an observation from the model, we sample as follows:

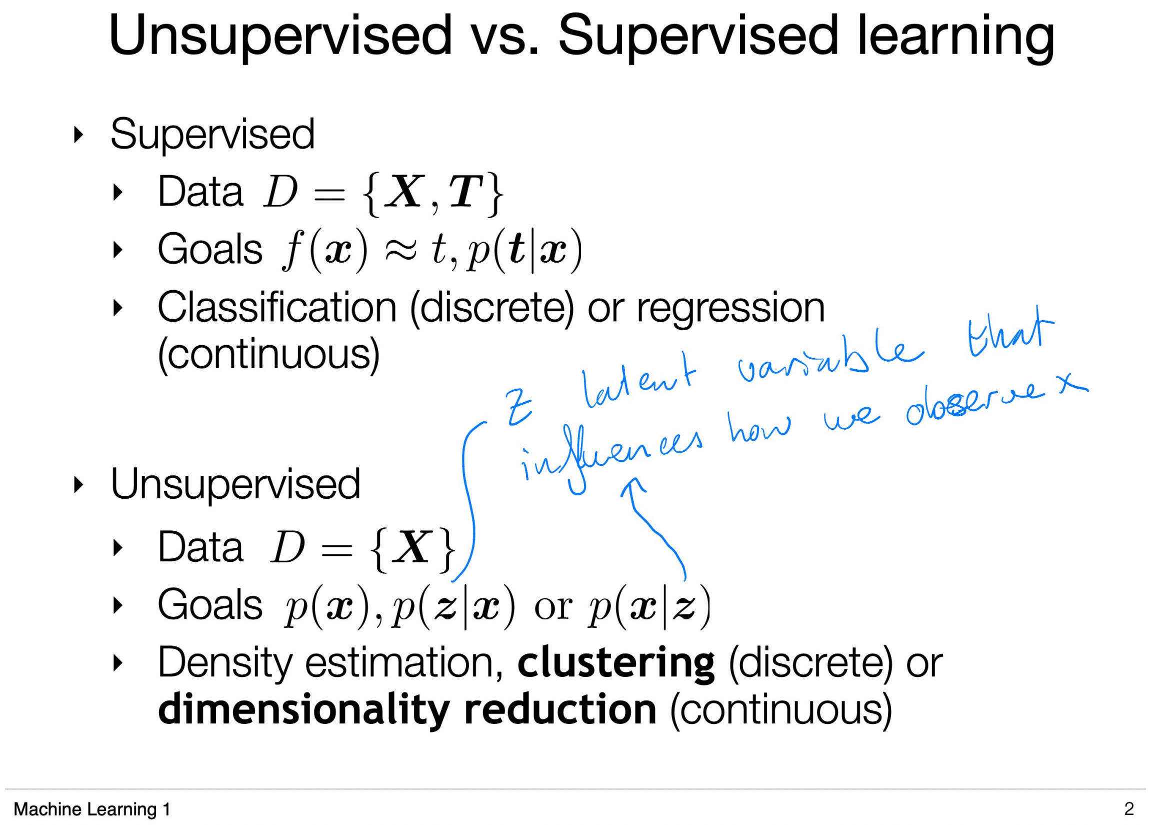
$$z \sim p(z)$$
$$x \sim p(x|z)$$



Next step: “Inference”

- The process going from observations x to the latent variable z
- We want to know the factors that generate the data
- (note: sometimes in ML, we also use inference to mean “test-time” (like “inference speed”), but not here)

Reminder: notes from ML1

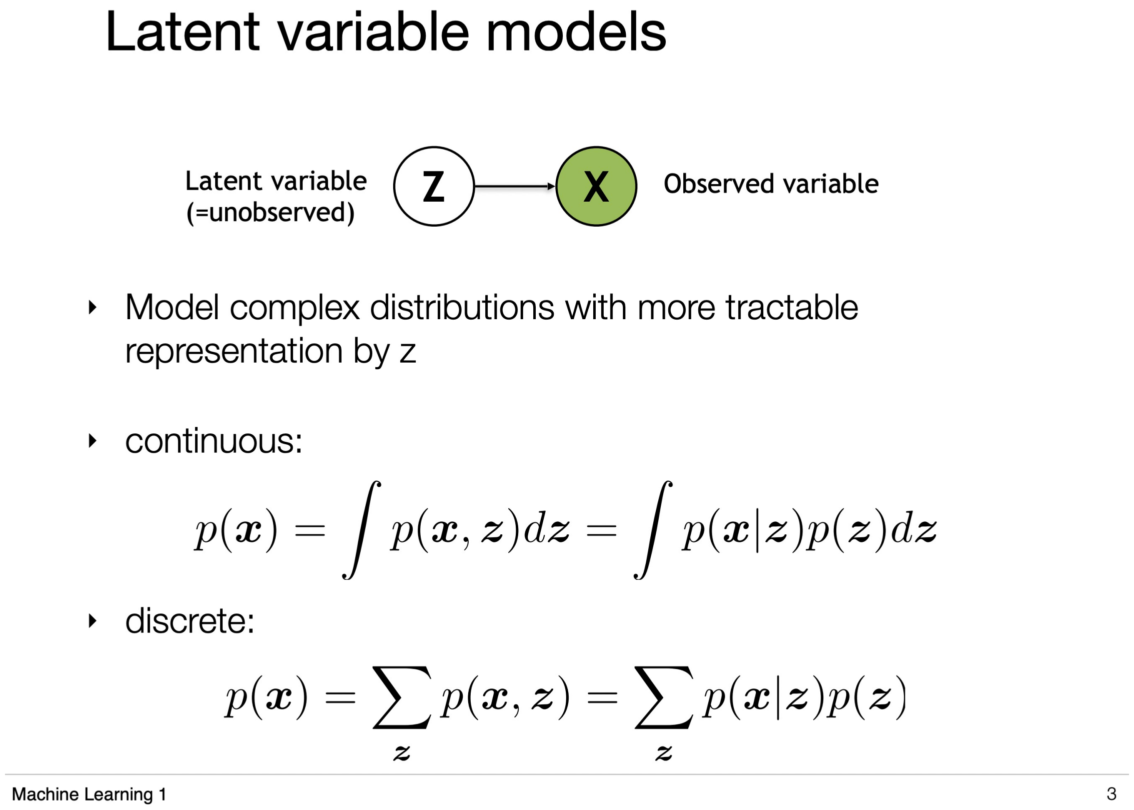


Unsupervised vs. Supervised learning

- Supervised
  - Data $D = \{\boldsymbol{X}, \boldsymbol{T}\}$
  - Goals $f(\boldsymbol{x}) \approx t, p(\boldsymbol{t}|\boldsymbol{x})$
  - Classification (discrete) or regression (continuous)

z latent variable that influences how we observe x

- Unsupervised
  - Data $D = \{\boldsymbol{X}\}$
  - Goals $p(\boldsymbol{x}), p(\boldsymbol{z}|\boldsymbol{x})$ or $p(\boldsymbol{x}|\boldsymbol{z})$
  - Density estimation, **clustering** (discrete) or **dimensionality reduction** (continuous)



Latent variable models

Latent variable (=unobserved)

Z → X

Observed variable

- Model complex distributions with more tractable representation by z
- continuous:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \int p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) d\boldsymbol{z}$$

- discrete:

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z})$$

Inference

Inference: computing the posterior distribution for given x:

$$p(z|x) = \frac{p(x,z)}{p(x)} = \frac{p(x,z)}{\int p(x,z')dz'}$$

This requires solving the important sub-problem of computing the **marginal likelihood** of the observation:

$$p(x) = \int p(x,z)dz$$

which is usually intractable (we cannot sum up all points in a D dim. Space)

Inference

Inference as *inverse process* of generation.

Generate pairs (x, z) from the model in two ways:

Since we have

$$\boxed{\begin{array}{l} \mathbf{z} \sim p(\mathbf{z}) \\ \mathbf{x} \sim p(\mathbf{x}|\mathbf{z}) \end{array}} \longrightarrow \boxed{\begin{array}{l} \mathbf{x} \sim p(\mathbf{x}) \\ \mathbf{z} \sim p(\mathbf{z}|\mathbf{x}) \end{array}}$$

The joint distribution of these pairs is **exactly the same**, no matter how they were generated. (Bayes' rule)

$$p(x|z)p(z) = p(x,z) = p(z|x)p(x)$$

Why shall we do inference?

Explaining the observations

- inferring the posterior distribution for a datapoint allow us to determine which latent configurations could have plausibly generated it
- This will allow us to generate new data points

Learning the generative model

- training latent variable models requires performing inference

Now, how do we do it?

Inference via maximum likelihood

Maximum Likelihood: dominant estimation principle for probabilistic models

Given a set of data points $\{x_i\}$, we define a model $p_\theta(x)$ to represent the data.

Find θ that maximize the probability of data under the model:

$$\theta^{ML} = \arg\max_{\theta} \sum_{i=1}^{N} \log p_\theta(x_i)$$

For latent variable models: no closed-form solution

Reminder: Why the sum of logarithms?

Goal: $p(x) = f(x, \theta)$, given dataset $\{x^i\}$, i = 1,…,N

$\theta^* = \text{argmax}_\theta \prod_i p(x_i)$ assumption: every datapoint is independent

$= \text{argmax}_\theta \Sigma_i \log(p(x_i))$

$= \text{argmax}_\theta \Sigma_i \log(f(x_i, \theta))$, with f being our neural network

The gradient of max likelihood

Let's compute the gradient of the log-likelihood for a single datapoint:

$$\begin{aligned}\nabla_\theta \log p_\theta(\mathbf{x}) &= \frac{\nabla_\theta p_\theta(\mathbf{x})}{p_\theta(\mathbf{x})} = \frac{\int \nabla_\theta p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}}{p_\theta(\mathbf{x})} \\ &= \frac{\int p_\theta(\mathbf{x}, \mathbf{z}) \nabla_\theta \log p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}}{p_\theta(\mathbf{x})} \\ &= \int p_\theta(\mathbf{z}|\mathbf{x}) \nabla_\theta \log p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}\end{aligned}$$

Maths:

$$\boxed{\nabla_\theta \log p_\theta(\mathbf{x}) = \frac{\nabla_\theta p_\theta(\mathbf{x})}{p_\theta(\mathbf{x})}}$$

We need to compute the posterior distribution to compute the gradient!

But: Exact inference is hard

$$\boxed{p(z|x) = \frac{p(x,z)}{\int p(x,z)dz}}$$

Inference for continuous latent variables involves computing high-dimensional integrals:

$$p(x) = \int p(x,z)dz$$

Inference for discrete latent variables involves summing over exponentially many latent configurations

$$\log \prod_{\boldsymbol{x} \in D} p(\boldsymbol{x}) = \sum_{\boldsymbol{x}} \log p(\boldsymbol{x}) = \sum_{\boldsymbol{x}} \log \sum_{\boldsymbol{z}} p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})$$

- E.g., for a 3-dimensional binary $\boldsymbol{z}$ iterate over [0,0,0],[0,0,1],[0,1,1],…
- For 20 dimensions $2^{20} \approx 1M$ latents and generations. Per image $\boldsymbol{x}$!

Let's take a breath. Where are we?

- Autoencoders: nice idea but does not give us probabilities
- Idea 1: let's model it with latent variables: z -> x
  - If we simply fix p(z) in some manner, p(x) can be computed as p(x|z)p(z)
  - Nice
- So how do we do inference (going from observations to model parameters)?
  - We do this by learning p(z|x): learn which z was used for a given x
  - However the "normalize across data" integral is intractable to compute

Variational Inference

We will use the calculus of variations to approximate those intractable integrals.

This will turn the inference problem into an optimization problem

Approximate inference

Markov Chain Monte Carlo: generate samples from the exact posterior using a Markov Chain

- Very general; exact in the limit of infinite time / computation
- Computationally expensive; convergence is hard to diagnose

Variational inference: approximate the posterior with a tractable distribution, e.g., fully factorized or autoregressive

- Fairly efficient, as inference is reduced to optimization w.r.t. the distribution parameters
- Cannot trade computation for accuracy easily

Revisit: Kullback–Leibler divergence

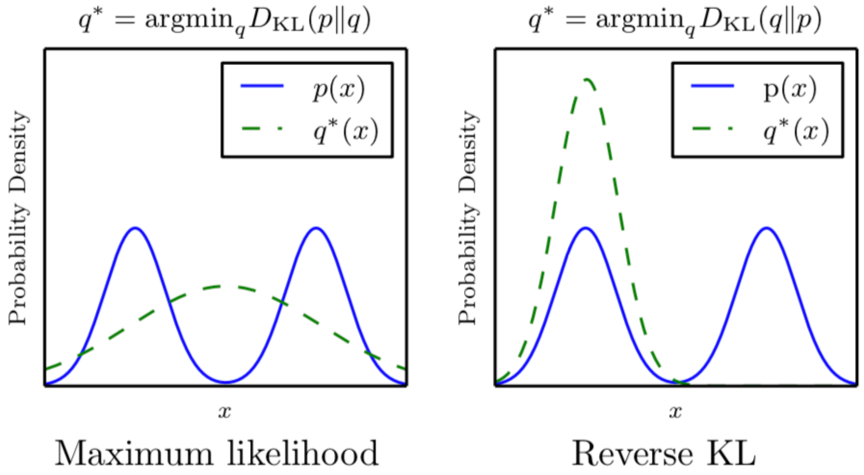
Kullback–Leibler divergence provides a way of quantifying the difference between two probability distributions.

KL divergence between q and p is defined as

$$KL(q||p) = E_q(\log \frac{q}{p})$$

KL divergence is

- non-negative $KL(q||p) \geq 0$;
- $KL(q||p) = 0$ if only if $q = p$;
- Not symmetric, in general $KL(q||p) \neq KL(p||q)$



Tool 1: Jensen’s inequality

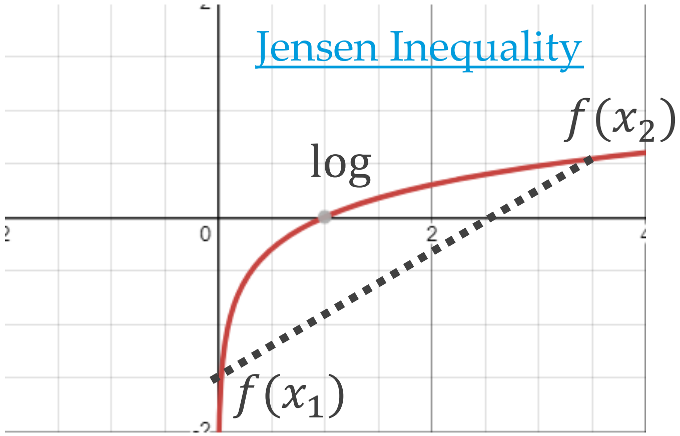
A concave function f (like a logarithm) on a sum will always be larger than the sum of f on individual summands

- Basically, a line connecting two points of a function will be always below the function

$$f(tx_1 + (1-t)x_2) \geq tf(x_1) + (1-t)f(x_2)$$

With probabilities and random variables this translates to

$$f(\mathbb{E}[\boldsymbol{x}]) \geq \mathbb{E}[f(\boldsymbol{x})]$$

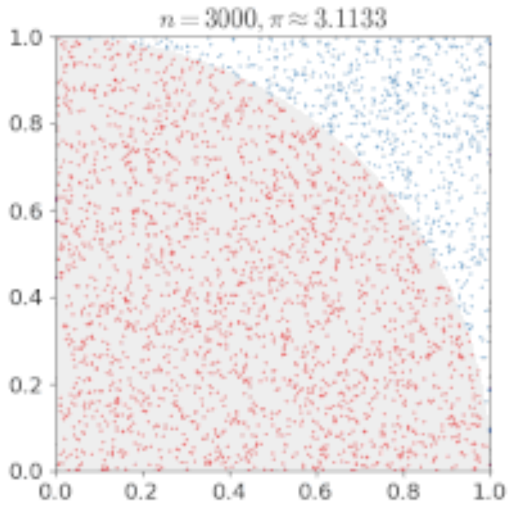


Tool 2: Monte Carlo methods

The methods rely on repeated random sampling to obtain numerical results.

We want to estimate the value of π

- 1. Draw a square, then inscribe a quadrant within it
- 2. Uniformly scatter a given number of points over the square
- 3. Count the number of points inside the quadrant, i.e. having a distance from the origin of less than 1
- 4. The ratio of the inside-count and the total-sample-count is an estimate of the ratio of the two areas, $\pi/4$. Multiply the result by 4 to estimate π.



Wikipedia

Tool 2: Monte Carlo methods

Consider the one dimensional integral

$$I = E_{x \sim p}[f(x)] = \int p(x) f(x) dx$$

We can approximate the integral by

$$E_p[f(x)] \approx I_N = \frac{1}{T} \sum_{t=1}^{T} f(x^t)$$

$x^1, \dots, x^T$ are samples drawn from p.

The MC estimate is an unbiased estimator.

In the limit of a large number of points N, I_N tends to the exact value I (Law of large numbers).

Variational inference

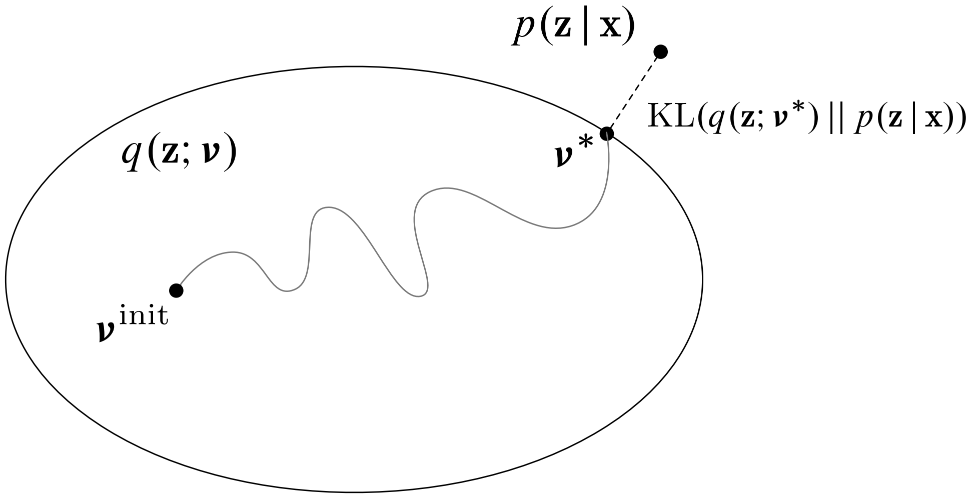
Variational inference turns the task of finding the posterior distribution into an optimization problem.

Approximate exact posterior $p_\theta(z|x)$ with **variational posterior** $q_\phi(z|x)$.

ϕ: **variational parameters** which we optimize to fit variational posterior to the exact posterior.

We usually use Kullback-Leibler divergence.

(don't worry about the difficult sounding things)


$p(\mathbf{z} \mid \mathbf{x})$
$\mathrm{KL}(q(\mathbf{z}; \boldsymbol{\nu}^*) \,||\, p(\mathbf{z} \mid \mathbf{x}))$
$q(\mathbf{z}; \boldsymbol{\nu})$
$\boldsymbol{\nu}^*$
$\boldsymbol{\nu}^{\text{init}}$

Variational inference

We can use any distribution $q_\phi(z)$ for as long as

- We can sample from it
- We can compute and its gradient w.r.t ϕ

We will use the easiest form possible

Training with variational inference

What do we use as the training objective if the marginal log-likelihood is intractable?

$$\max \ \log \, p_\theta(x)$$

The variational posterior induces a **variational lower bound** on the marginal log-likelihood.

We train a model with VI by maximizing **variational lower bound**

- w.r.t. both the model parameters θ and the variational parameters ϕ.

Important: How to arrive at the variational lower bound

For any density $q_\phi(z)$ as long as $q_\phi(z) > 0$, we have

$$\log p_\theta(x) = \log \int p_\theta(x,z)dz \quad = \log \int q_\phi(z) \frac{p_\theta(x,z)}{q_\phi(z)} dz$$

$$\geq \int q_\phi(z) \log \frac{p_\theta(x,z)}{q_\phi(z)} dz$$

Jensen's inequality

$$\boxed{\log E_{q_\phi(z)}[f(z)] \geq E_{q_\phi(z)}[\log f(z)]}$$

$$= E_{q_\phi(z)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z)}\right]$$

variational lower bound

Variational lower bound

Therefore, for any such $q_\phi(z)$

$$\mathcal{L}_{\theta,\phi}(x) = E_{q_\phi(z)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z)}\right] \leq \log p_\theta(x)$$

We now can maximize $\mathcal{L}_{\theta,\phi}(x)$ instead of $\log p_\theta(x)$.

Variational lower bound

We design a variational posterior $q_\phi(z|x)$ as $q_\phi(z)$ (remember q only came from “multiplying by 1”):

$$\mathcal{L}_{\theta,\phi}(x) = E_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right]$$

which is called **Evidence Lower Bound (ELBO),** the simplest and widely used variational lower bound.

Higher ELBO → smaller difference to true $p_\theta(z|x)$ → better latent representation

Higher ELBO→ gap to log-likelihood tightens → better density model

Derive ELBO in a different way

Start with KL of q and p

$$\mathrm{KL}\big(q_{\phi}(z|x) \parallel p_{\theta}(\boldsymbol{z}|\boldsymbol{x})\big) = \int q_{\phi}(z|x)\,(\log q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) - \log p_{\theta}(z|x))\,d\boldsymbol{z}$$

$$= \int q_{\phi}(z|x)\,(\log q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) - \log \frac{p_{\theta}(x|z)p(z)}{p(x)})\,d\boldsymbol{z}$$

$$= \int q_{\varphi}(z|x)\,(\log q_{\phi}\ \ (\boldsymbol{z}|\boldsymbol{x}) - \log\ p_{\theta}(x|z) - \log p(z) + \log p(x))\,d\boldsymbol{z}$$

$$= -\mathbb{E}_{q_{\phi}\ \ (z|x)}[\log p_{\theta}(x|z)] + KL\big(q_{\phi}(\boldsymbol{z})||p(z)\big) +\ \log p(\boldsymbol{x}) \geq 0$$

$$\boxed{\log p(\boldsymbol{x}) \geq \mathbb{E}_{q_{\phi}(z|x)}\,[\log p_{\theta}(x|z)] - KL\left(q_{\phi}\ \ (z|x)||p(z)\right)}$$

Entropy regularization

We can also expand the ELBO as

$$\mathbb{E}_{q_\phi(\boldsymbol{z})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z})}\right] = \mathbb{E}_{q(\boldsymbol{z})}[\log p(\boldsymbol{x}, \boldsymbol{z})] - \mathbb{E}_{q(\boldsymbol{z})}[\log q_\phi(\boldsymbol{z})]$$

$$= \mathbb{E}_{q_\phi(\boldsymbol{z})}[\log p(\boldsymbol{x}, \boldsymbol{z})] + H(q_\phi(\boldsymbol{z}))$$

where H(·) is the entropy

Maximising ELBO: increases likelihood of data and latents, and maintains enough entropy (‘uncertainty’) in the distribution of latents

- Avoiding latents to collapse to point estimates ($\boldsymbol{z}$ as single values)

Variational gap

Rewriting the ELBO:

$$\mathcal{L}_{\theta,\phi}(x) = E_{q_\phi(z)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] = E_{q_\phi(z)}\left[\log \frac{p_\theta(z|x)p_\theta(x)}{q_\phi(z|x)}\right]$$

$$= E_{q_\phi(z|x)}[\log p_\theta(x)] + E_{q_\phi(z|x)}\left[\log \frac{p_\theta(z|x)}{q_\phi(z|x)}\right]$$

$$= \log p_\theta(x) - KL(q_\phi(z|x)||p_\theta(z|x))$$

Variational gap

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

This means maximizing $\mathcal{L}_{\theta,\phi}(x)$ w.r.t. ϕ is equivalent to minimizing

$$KL(q_\phi(z|x)||p_\theta(z|x))$$

$\log p_\theta(x)$ does not depend on ϕ

which is known as the variational gap, because it is the difference between $\log p_\theta(x)$ and the variational bound $\mathcal{L}_{\theta,\phi}(x)$

Fitting the variational posterior

We can fit the variational posterior to the exact posterior by maximizing the ELBO w.r.t. ϕ, which minimizes the

$$KL(q_\phi(z|x)||p_\theta(z|x))$$

This is remarkable because we cannot compute $KL(q_\phi(z|x)||p_\theta(z|x))$ or even $p_\theta(z|x)$.

$$\boxed{\mathcal{L}_{\theta,\phi}(x) = \log p_\theta(x) - KL(q_\phi(z|x)||p_\theta(z|x))}$$

ELBO is the difference between two intractable quantities, which is tractable!!!

Fitting the variational posterior

$$\boxed{\mathcal{L}_{\theta,\phi}(x) = \log p_\theta(x) - KL(q_\phi(z|x)||p_\theta(z|x))}$$

If the variational distribution family is expressive enough, the ELBO is maximized w.r.t. ϕ, then the variational posterior is equal to the exact posterior (and the variational gap is zero).

By updating the variational parameters ϕ to increase ELBO,

- makes the variational distribution closer the true posterior
- Improves the variational approximation without affecting the model

Training the model

When we update the **model** parameters θ to increase the ELBO

$$\boxed{\mathcal{L}_{\theta,\phi}(x) = \log p_\theta(x) - KL(q_\phi(z|x)||p_\theta(z|x))}$$

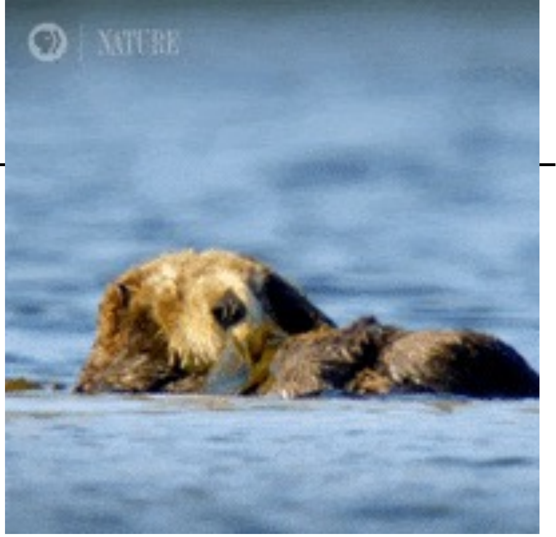
$\log p_\theta(x)$ will increase and/or

$KL(q_\phi(z|x)||p_\theta(z|x))$ will decrease

--> We should use the most expressive variational posterior we can.

- how to choose the variational posterior?

Where are we?



- Autoencoders: nice idea but does not give us probabilities
- Idea 1: let's model it with latent variables: z -> x
  - If we simply fix p(z) in some manner, p(x) can be computed as p(x|z)p(z)
  - Nice
- So how do we do inference (going from observations to model parameters)?
  - We do this by learning p(z|x): learn which z was used for a given x
  - However the "normalize across data" integral is intractable to compute
- By using another probability distribution and Jensen's inequality, we can arrive at a lower bound on the quantity we want to optimize
  - This ELBO is given by $\mathcal{L}_{\theta,\phi}(x) = E_{q_\phi(z)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] = E_{q_\phi(z)}[\log p_\theta(x,z)] - KL(q_\phi(z|x)||p(z))$
  - We have also quantified the gap of ELBO to p(x) to be $KL(q_\phi(z|x)||p(z|x))$
  - Now: how to choose and optimize q?

Choosing the form of the variational posterior

The default choice is a fully factorized distribution

$$q_\phi(z|x) = \prod_i q_\phi(z_i|x)$$

In classic VI, this is known as the **mean field** approximation.

Several options for more expressive posteriors:

- Mixture distributions
- Gaussian with a non-diagonal covariance matrix
- Autoregressive
- Flow-based

Trade off between training speed and approximation quality

Amortized variational inference

The posterior distribution $p(z|x)$ is **different** for each observation x

- each x has its own latent distribution over z

In classic variational inference, we learn a *different* set of variational parameters ϕ for each datapoint using iterative optimization.

- ϕ represents μ and σ for Gaussian distributions.

We adopt the *amortized inference* technique: much more efficient

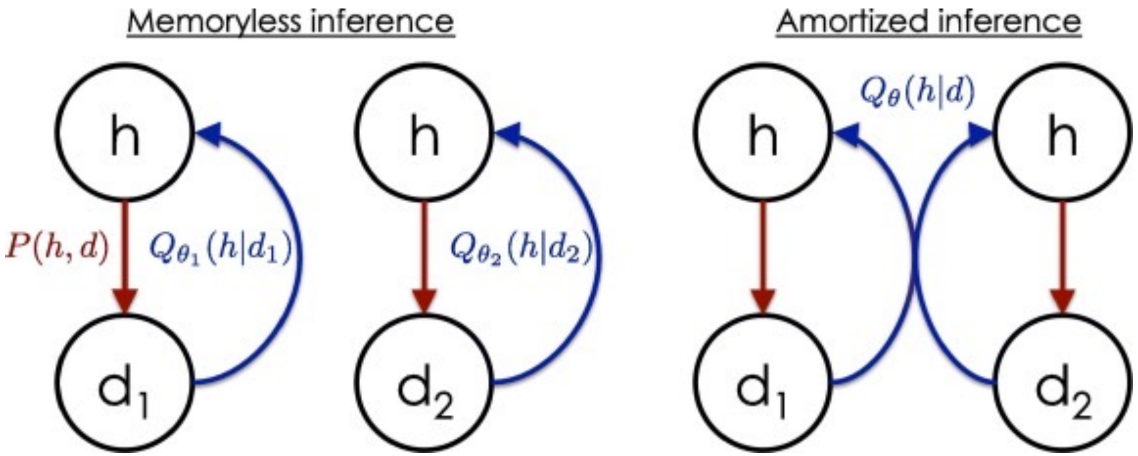
Amortized variational inference

Parameterized function: from observation space to parameters of the approximate posterior

- instead of optimizing a set of free parameters

For this: a neural network: x as input, and outputs the μ and σ for z

- the neural network is shared by different observations!



Amortized variational inference

Then optimize the parameters of amortized neural networks instead of the individual parameters of each observation.

- e.g., ϕ represents the network weights now!
- trained jointly with the model by maximizing the ELBO.

Amortized inference can cause an amortization gap (talked about later)

- a reason for suboptimality

Benefits of using amortization

First, the number of variational parameters is constant w.r.t. to the dataset size!

- it is the size of the inference network
- we only need to specify the parameters of the network

Second, for new observation, all we need to do is to pass it through the network,

- we have an approximate posterior distribution over its associated latent variables!
- At the constant cost of a forward pass through a network (no optimization)

Variational vs. exact inference

What did we gain by using variational inference?

- Inference in an efficient and principled way.
- Freedom in model design
- Inference is fast compared to MCMC methods.

What did we lose?

- VI can make the model effectively less expressive
- thus lead to suboptimal performance.

Maximizing the ELBO

Our objective is to maximize the ELBO

$$\mathcal{L}_{\theta,\phi}(x) = E_{q_\phi(z)}\left[\log\frac{p_\theta(x,z)}{q_\phi(z|x)}\right]$$

w.r.t. the model parameter θ and variational parameter ϕ.

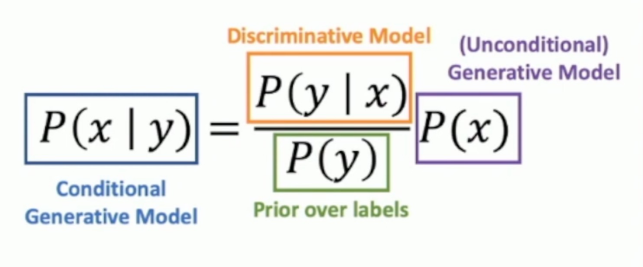
It is non-convex optimization

We need to compute the gradients.

In modern variational inference, we estimate gradients using Monte Carlo sampling.

Take 3 min and discuss with your neighbor the following points
(think about what is the main idea, what difficult thing to understand, etc.)

1) Generative models, Bayes' rule



$$\boxed{P(x \mid y)} = \frac{\boxed{P(y \mid x)}}{\boxed{P(y)}}\boxed{P(x)}$$

2) Jensen's inequality

3) Inference, variational inference and amortization

Gradient w.r.t. the model parameters

Gradient with regards to θ:

$$\nabla_\theta \mathcal{L}_{\theta,\phi}(x) = \nabla_\theta E_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right]$$

$$= E_{q_\phi(z|x)}[\nabla_\theta p_\theta(x,z)]$$

$$= \frac{1}{K}\sum_{k=1}^{K} \nabla_\theta p_\theta(x, z^k) \quad \text{with} \quad z^k \sim q_\phi(z|x)$$

We simply generate one or more samples from the variational posterior and average the resulting gradients of the log-joint

Gradient w.r.t. variational parameters

Estimating the gradient w.r.t. ϕ is nontrivial

- using samples from the variational posterior which depends on ϕ

$$\nabla_\phi \mathcal{L}_{\theta,\phi}(x) = \nabla_\phi E_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] = ???$$

- which involves in gradients of an expectation

$$\nabla_\phi E_{q_\phi(z|x)}[f(z)]$$

- One last trick to make this all work

Gradients of expectations

$$\boxed{\nabla_\phi E_{q_\phi(z|x)}[f(z)]}$$

REINFORCE / likelihood-ratio estimator

- Very general
- Applicable to both discrete and continuous latent variables
- Can handle non-differentiable function $f(z)$

Reparameterization

- Less general
- Applicable only to continuous latent variables
- Requires $f(z)$ to be differentiable
- Tends to have relatively low variance

Reparameterization trick

We reparametrize a sample from $q_\phi(z|x)$ by expressing it as a function of a sample ϵ from some fixed distribution $p(\epsilon)$:

$$z = g(\epsilon, \phi)$$

$g(\epsilon, \phi)$ needs to be differentiable w.r.t. ϕ

$$\begin{aligned} \nabla_\phi E_{q_\phi(z)}[f(z)] &= \nabla_\phi E_{p(\epsilon)}[f(g(\epsilon, \phi))] \\ &= E_{p(\epsilon)}\left[\nabla_\phi f(g(\epsilon, \phi))\right] && \boxed{p(\epsilon) \text{ does not depend on } \phi} \\ &= E_{p(\epsilon)}\left[\nabla_z f(z) \nabla_\phi g(\epsilon, \phi)\right] && \boxed{\text{Chain rule}} \end{aligned}$$

Reparameterization trick

The reparameterization trick essentially moves the dependence on the distribution parameters inside the expectation.

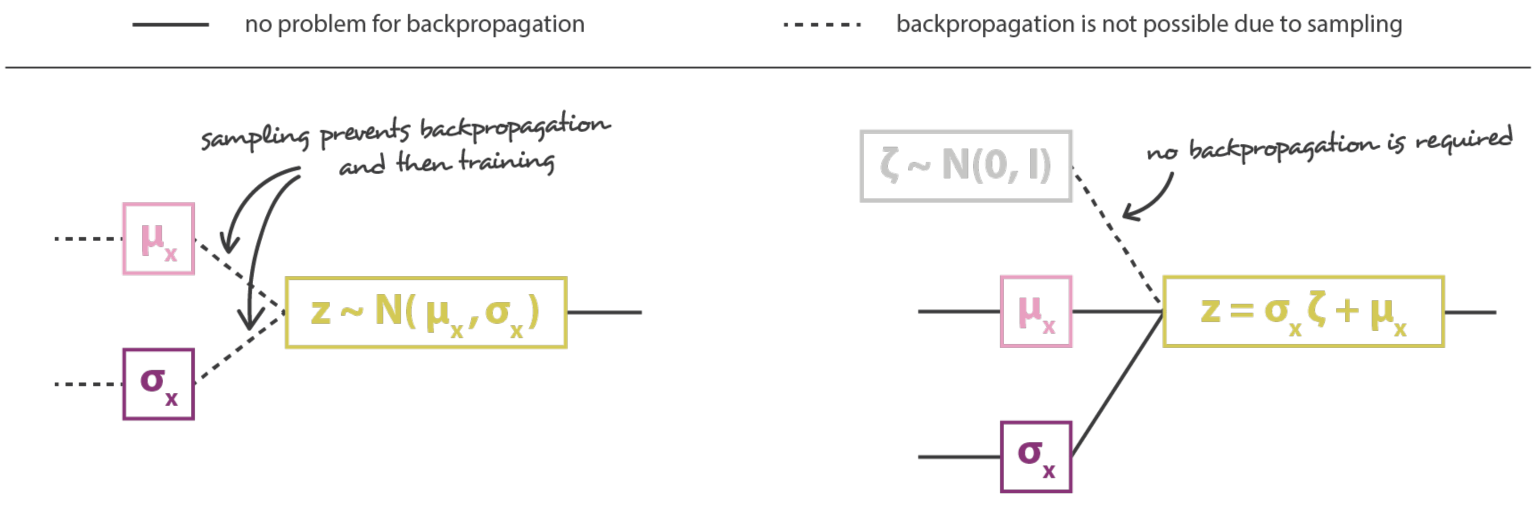
- This can be seen as propagating gradients through z
- To get the correct gradients, the function $g(\epsilon, \phi)$ mapping ϵ to z has to be differentiable w.r.t. the distribution parameters ϕ.

Reparameterizing a Gaussian variable: $z \sim \mathrm{N}(\mu, \sigma)$

$$z = \mu + \epsilon \odot \sigma\,, \quad \text{with } \epsilon \sim N(0, I)$$

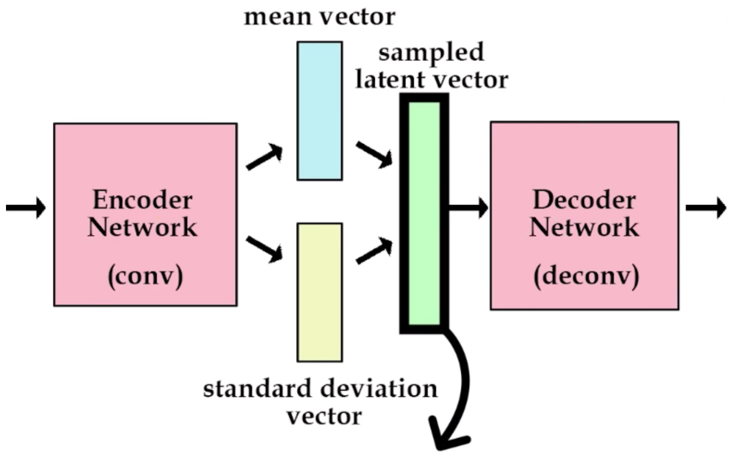
Note that this mapping from ϵ to z is differentiable w.r.t. both parameters of the distribution, as required.

Reparameterization trick visualised


no problem for backpropagation
backpropagation is not possible due to sampling
sampling prevents backpropagation and then training
μ_x
σ_x
$z \sim N(\mu_x, \sigma_x)$
$\zeta \sim N(0, I)$
no backpropagation is required
μ_x
σ_x
$z = \sigma_x \zeta + \mu_x$

sampling without reparametrisation trick

sampling with reparametrisation trick



$$z = \mu + \sigma \odot \varepsilon$$

where $\varepsilon \sim \text{Normal}(0,1)$

Where are we?

- Autoencoders: nice idea but does not give us probabilities
- Idea 1: let's model it with latent variables: z -> x
  - If we simply fix p(z) in some manner, p(x) can be computed as p(x|z)p(z)
  - Nice
- So how do we do inference (going from observations to model parameters)?
  - We do this by learning p(z|x): learn which z was used for a given x
  - However the "normalize across data" integral is intractable to compute
- By using another probability distribution and Jensen's inequality, we can arrive at a lower bound on the quantity we want to optimize
  - This ELBO is given by $\mathcal{L}_{\theta,\phi}(x) = E_{q_\phi(z)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] = E_{q_\phi(z)}[\log p_\theta(x,z)] - KL(q_\phi(z|x)||p(z))$
  - We have also quantified the gap of ELBO to p(x) to be $KL(q_\phi(z|x)||p(z|x))$
  - Now how to choose and optimize q?
- We can choose any simple q, but optimization requires difficult gradient of an expectation involving the variational parameters that we wish to take gradients from
  - Broader context: sampling is a non-differentiable computation. But we can approximate it with one sample that carries the same mean and variation
  - Key idea: reparametrize z: have generic, stochastic ε and learn the scales and shifts (for which gradients can flow)

Variational autoencoders

A generative model with continuous latent variables:

The likelihood and the variational posterior are neural networks.

Both the prior and the variational posterior are usually fully factorized Gaussians.

VAEs are trained using amortized variational inference.

Take the advantage of the reparameterization trick.

Variational autoencoders

Rewrite the ELBO as follows:

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

The first term measures how well the model predicts / reconstructs an observation from a sample from the variational posterior.

- Known as the negative reconstruction error.

The second term acts as a regularizer, pushing the variational posterior towards the prior.

- It measures the amount of information about the observation in the latents.
- Often computed analytically.

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

$$\log P(x|\hat{x}) \sim \log e^{-|x-\hat{x}|^2} \sim (x - \hat{x})^2$$

Quiz: what problems might happen with the reconstructions?

1) They might be risk averse and not have sharp edges
2) Sampling at every location will yield a noisy image
3) Very high or low colors (close to 0 or 255) will be rare
4) Neighboring pixels will not have any dependencies

Variational autoencoders

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

$q_\phi(z|x)$ is implemented as the encoder network, also known as the recognition network

$p_\theta(x|z)$ is implemented as the decoder network

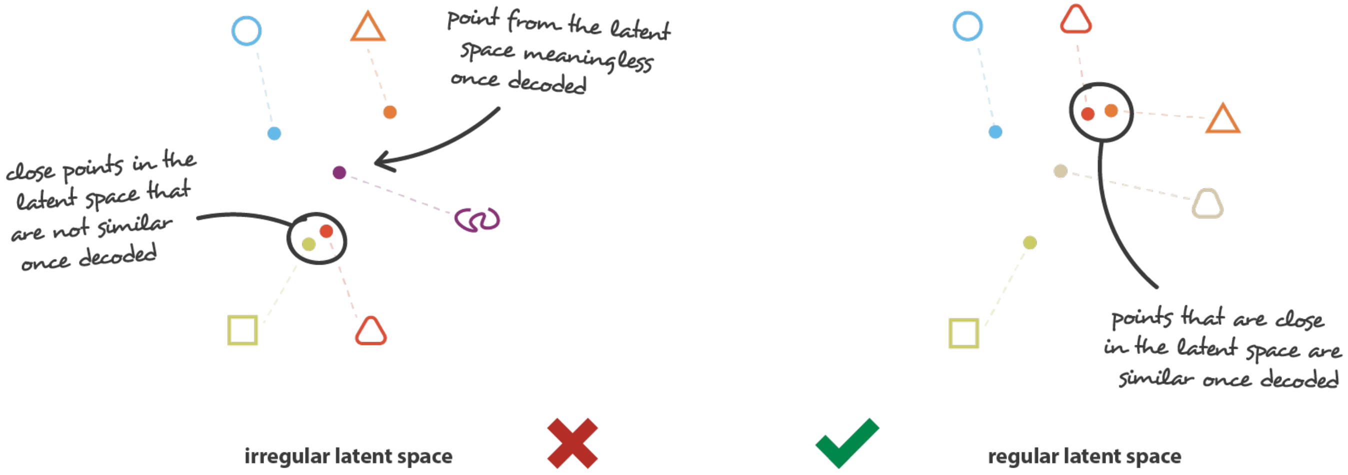
Both encoder and decoder networks are amortized networks

Intuition of the KL regularizer:

Make generative process have two main properties:

Continuity: two close points in the latent space should not give two completely different contents once decoded (thanks noise!)

Completeness: for a chosen distribution, a point sampled from the latent space should give “meaningful” content once decoded.


point from the latent space meaningless once decoded
close points in the latent space that are not similar once decoded
irregular latent space
points that are close in the latent space are similar once decoded
regular latent space

Inference suboptimality

The quality of approximate inference is determined by two factors

- **Variational posterior**: The capacity of the variational distribution to match the true posterior
- **Amortized inference**: The ability of the amortized network to produce good variational parameters for each datapoint.

Observation:

- divergence from the true posterior is often due to imperfect amortized inference network
- rather than the limited complexity of the approximating distribution.

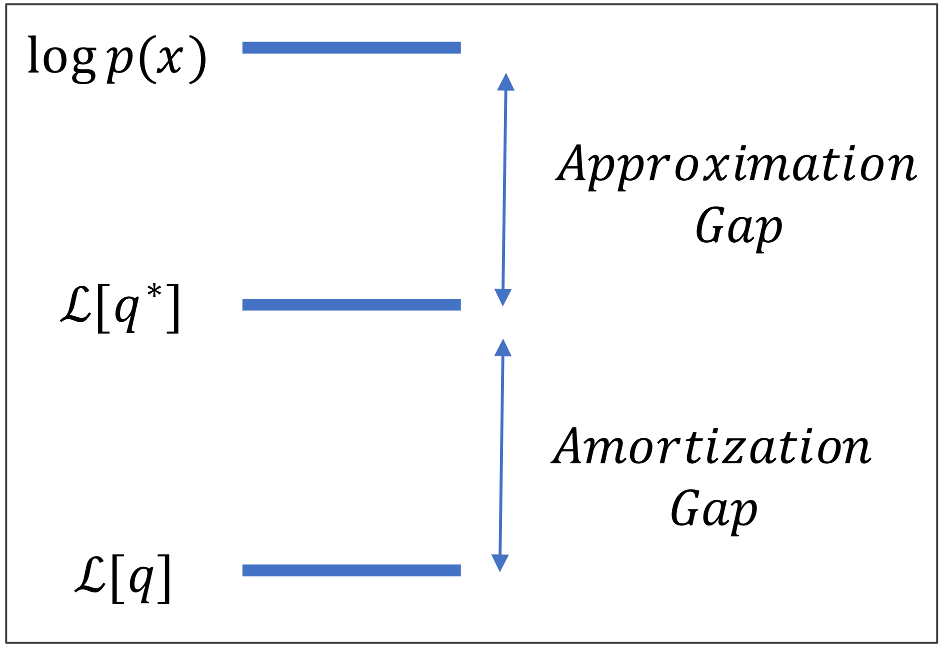
Inference Gaps

Approximation Gap

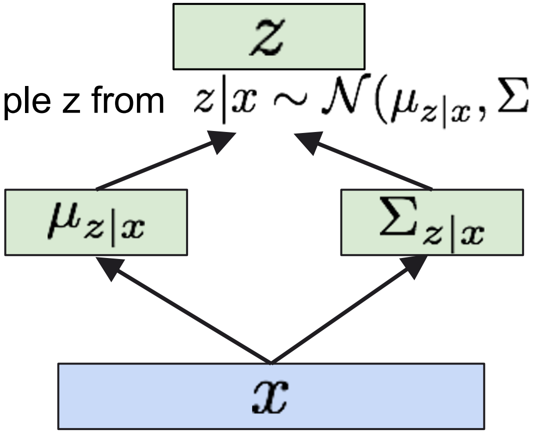
- Inability of the variational distribution to approximate the true posterior $KL(q_\phi(z|x)||p_\theta(z|x))$
- An expressive variational distribution → small approximation gap

Amortization Gap

- Limited capacity of the amortization/recognition network to generalize inference over all datapoints
- Stronger networks → small amortization gap



Encoder



Sample z from $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$

z

$\mu_{z|x}$

$\Sigma_{z|x}$

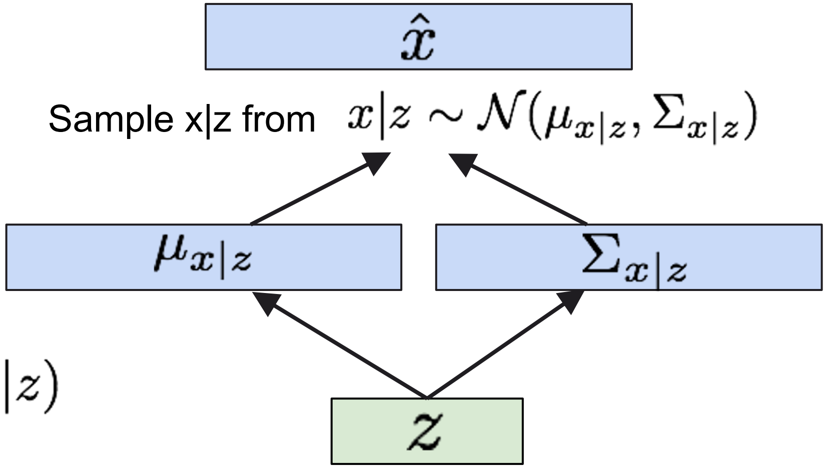
$q_\phi(z|x)$

x

Encoder network

Input Data

Decoder



$\hat{x}$

Sample x|z from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

$\mu_{x|z}$

$\Sigma_{x|z}$

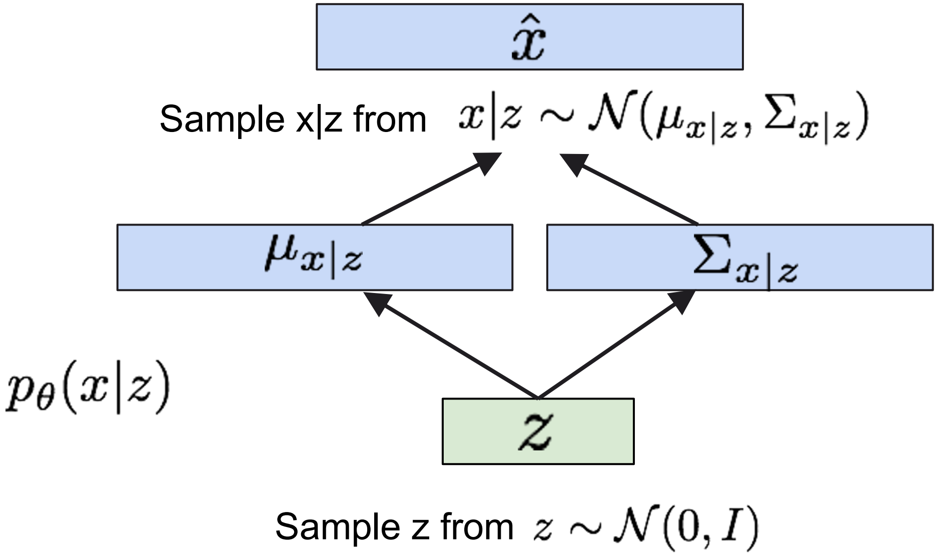
$p_\theta(x|z)$

z

Decoder network

Generating Data

Use decoder, and sample from the prior



$\hat{x}$

Sample x|z from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

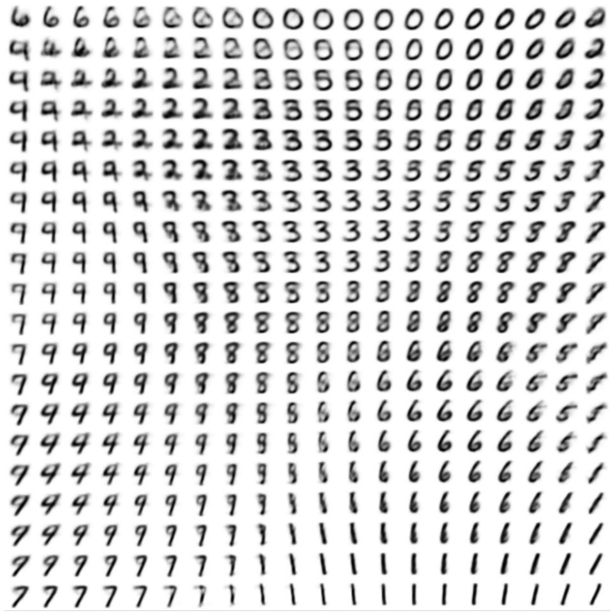
$\mu_{x|z}$

$\Sigma_{x|z}$

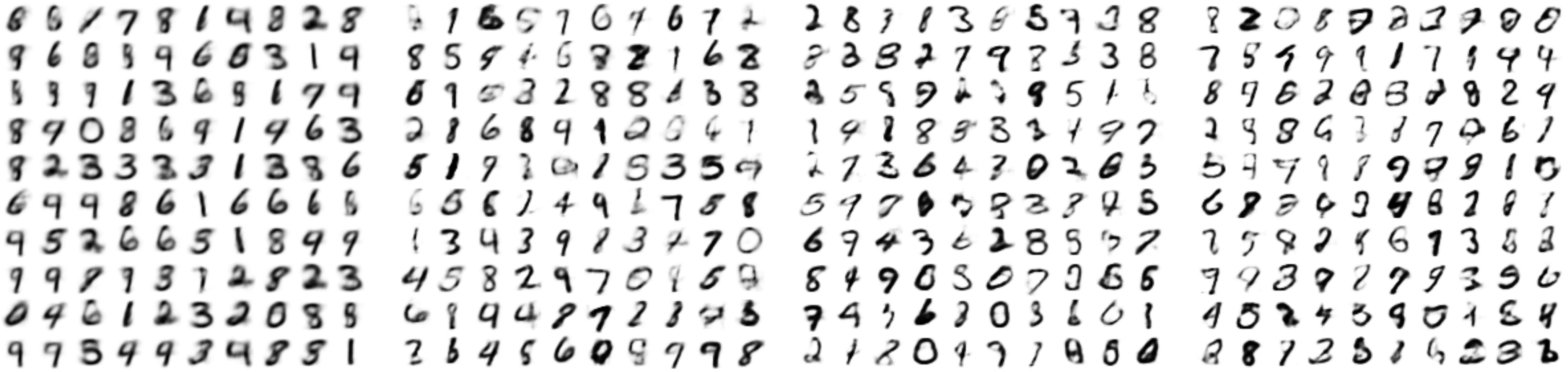
$p_\theta(x|z)$

z

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



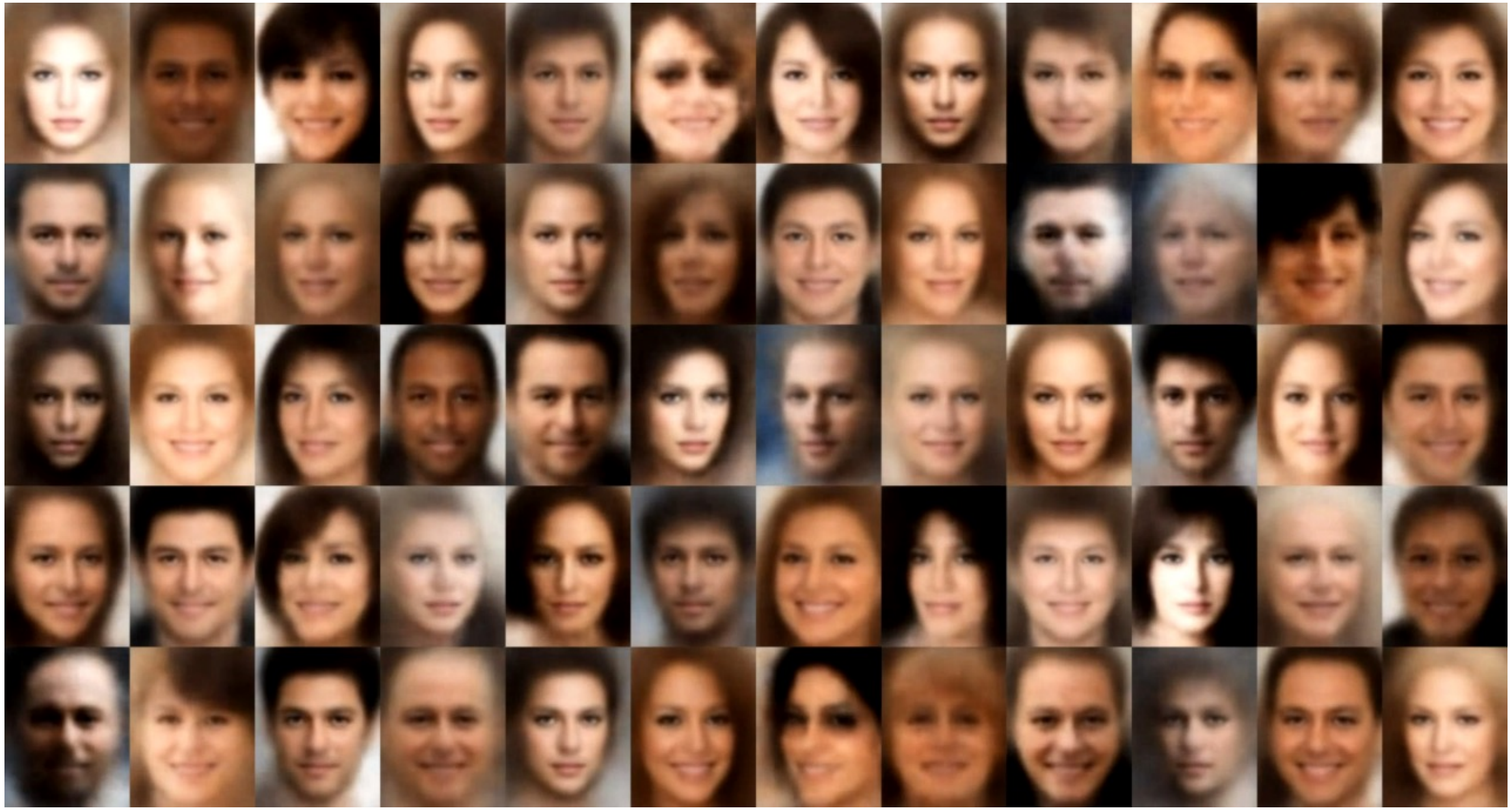
Dimensionality of latent space



(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Face generation

Why blurry? (Actually noisy)

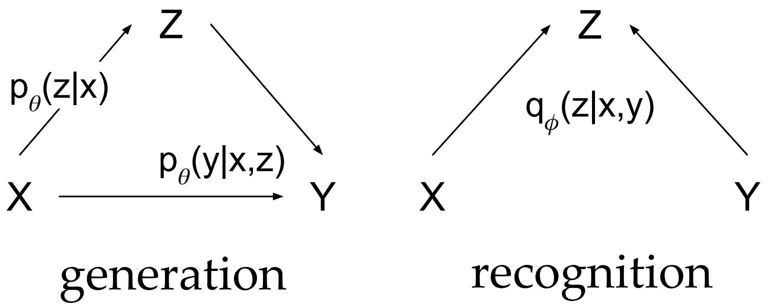


VAE variants

Conditional VAEs

Model the distribution of output space as a generative model conditioned on the input observation

- The input observations modulate the prior on Gaussian latent variables that generate the outputs.



Used for structured output predictions, e.g., object segmentation

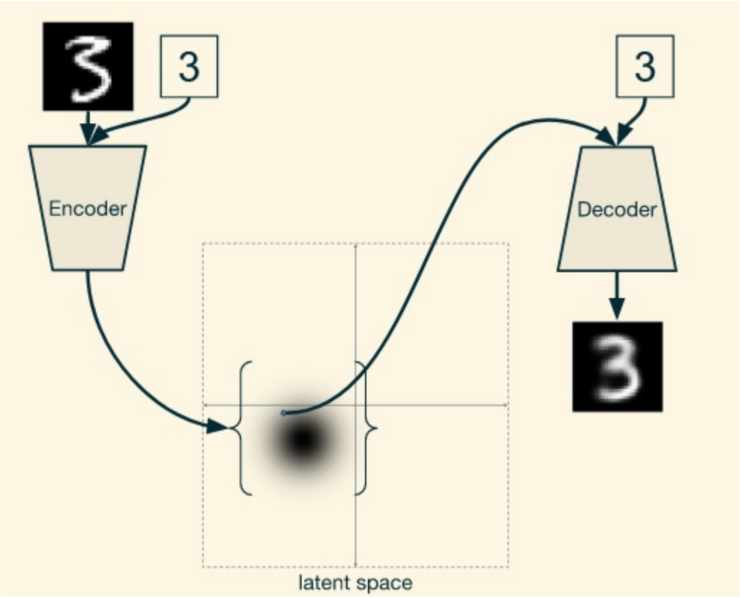
Conditional VAEs

The ELBO for conditional VAE

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

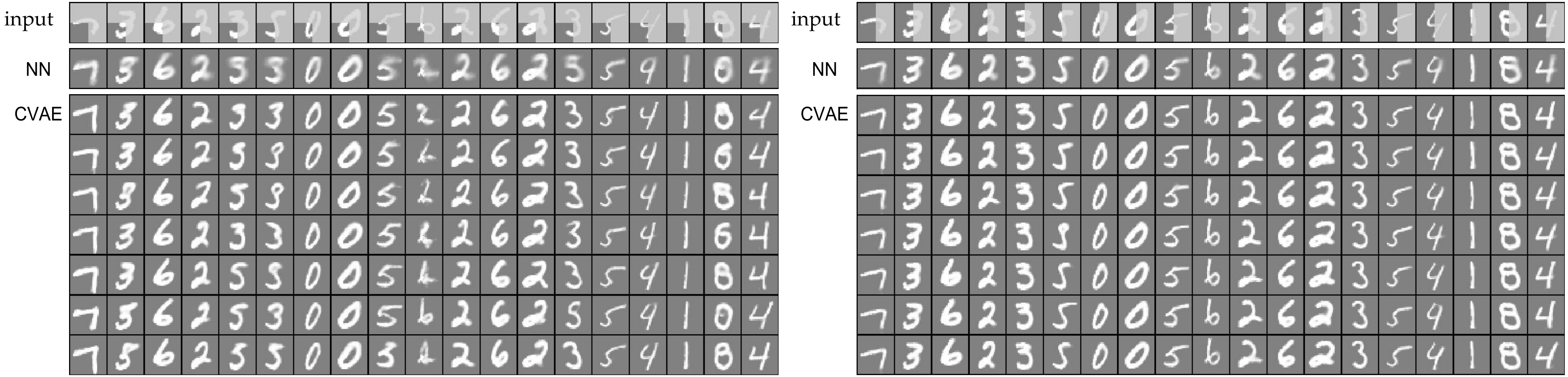
It is trained to maximize the conditional log-likelihood

The conditional variational autoencoder has an extra input to both the encoder and the decoder.



Conditional VAEs

Generated samples with (left) 1 quadrant and (right) 2 quadrants for an input



Beta-VAE

"More" Disentangled representations

- Each variable in the inferred latent representation *z* is only sensitive to one single generative factor
- relatively invariant to other factors

Good interpretability and easy generalization to a variety of tasks.

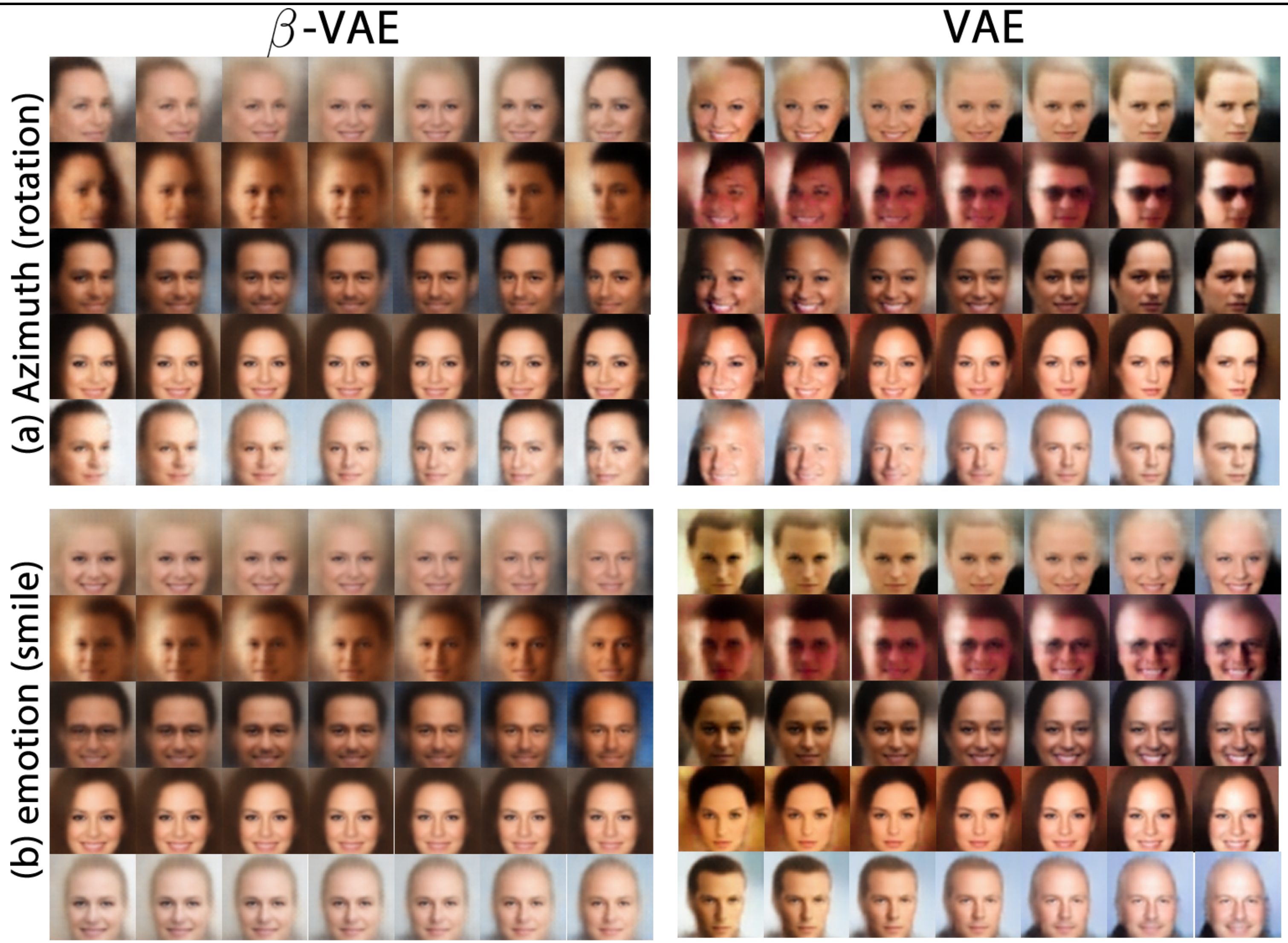
- e.g., model trained on photos of human faces might capture hair colour, hair length, etc.

The loss function of beta-VAE

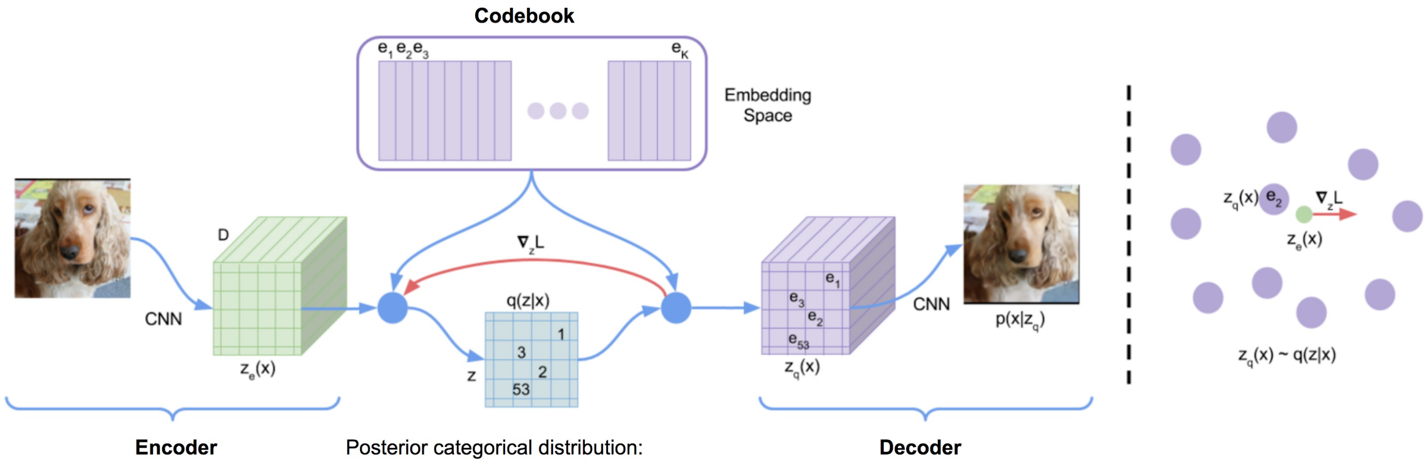
$$\mathcal{L}_{\theta,\phi}(x) = E_{q_\phi(z)}[\log p_\theta(x|z)] - \boldsymbol{\beta} KL(q_\phi(z|x)||p_\theta(z))$$

Beta-VAE

More steerable/interpretable

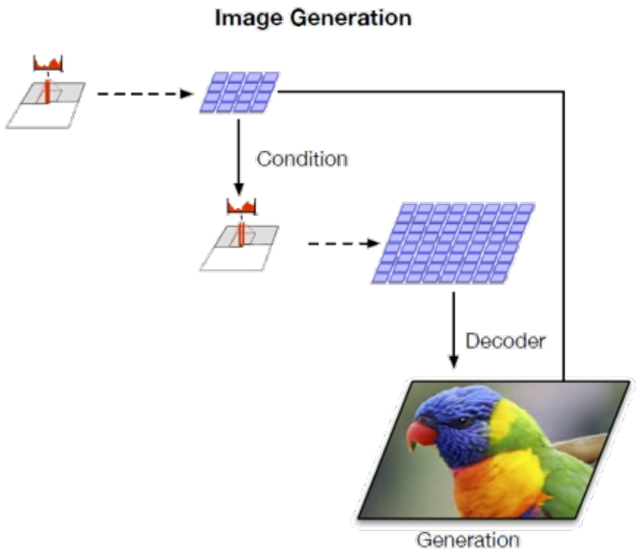


VQ-VAE and VQ-VAE2



Posterior categorical distribution:

$$q(\mathbf{z} = \mathbf{e}_k|\mathbf{x}) = \begin{cases} 1 & \text{if } k = \arg\min_i \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_i\|_2 \\ 0 & \text{otherwise.} \end{cases}$$



Neural Discrete Representation Learning. Van den Oord et al. NeurIPS 2017