Introduction to Data Mining

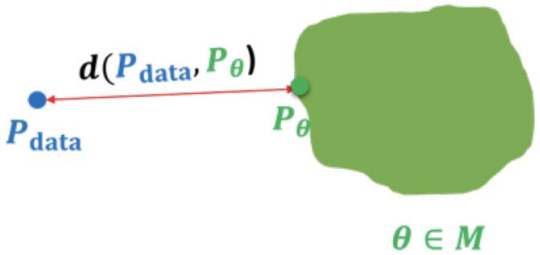
CS 145

Lecture 10:

Self-Supervised Learning: (Variational) AutoEncoder and Contrastive Learning

- We are given a training set of examples, e.g., images of dogs

$x_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



- We want to learn a probability distribution $p(x)$ over images x such that
  - **Generation:** If we sample $x_{new} \sim p(x)$, x_{new} should look like a dog (*sampling*)
  - **Density estimation:** $p(x)$ should be high if x looks like a dog, and low otherwise (*anomaly detection*)
  - **Unsupervised representation learning:** We should be able to learn what these images have in common, e.g., ears, tail, etc. (*features*)
- First question: how to represent $p_\theta(x)$. Second question: **how to learn it**.

$$\log p(\mathcal{D};\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)};\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} p(z^{(n)}, \mathbf{x}^{(n)};\boldsymbol{\theta}) \right)$$

- We introduce auxilliary distributions $q_n(z^{(n)})$ over each of the latent variables

$$\begin{aligned} \sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} p(z^{(n)}, \mathbf{x}^{(n)};\boldsymbol{\theta}) \right) &= \sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} q_n(z^{(n)}) \frac{p(z^{(n)}, \mathbf{x}^{(n)};\boldsymbol{\theta})}{q_n(z^{(n)})} \right) \\ &= \sum_{n=1}^{N} \log \left(\mathbb{E}_{q_n(z^{(n)})} \left[\frac{p(z^{(n)}, \mathbf{x}^{(n)};\boldsymbol{\theta})}{q_n(z^{(n)})} \right] \right) \end{aligned}$$

- We introduce auxilliary distributions $q_n(z^{(n)})$ over each of the latent variables

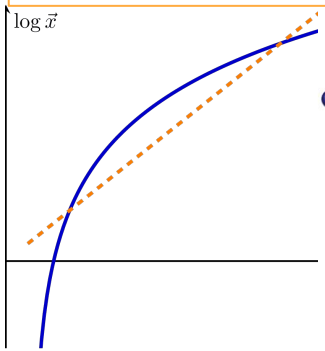
$$\sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta}) \right) = \sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} q_n(z^{(n)}) \frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right)$$

$$= \sum_{n=1}^{N} \log \left(\mathbb{E}_{q_n(z^{(n)})} \left[\frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right] \right)$$

$$\geq \sum_{n=1}^{N} \mathbb{E}_{q_n(z^{(n)})} \left[\log \frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right]$$

If g is **concave**, the inequality changes directions:

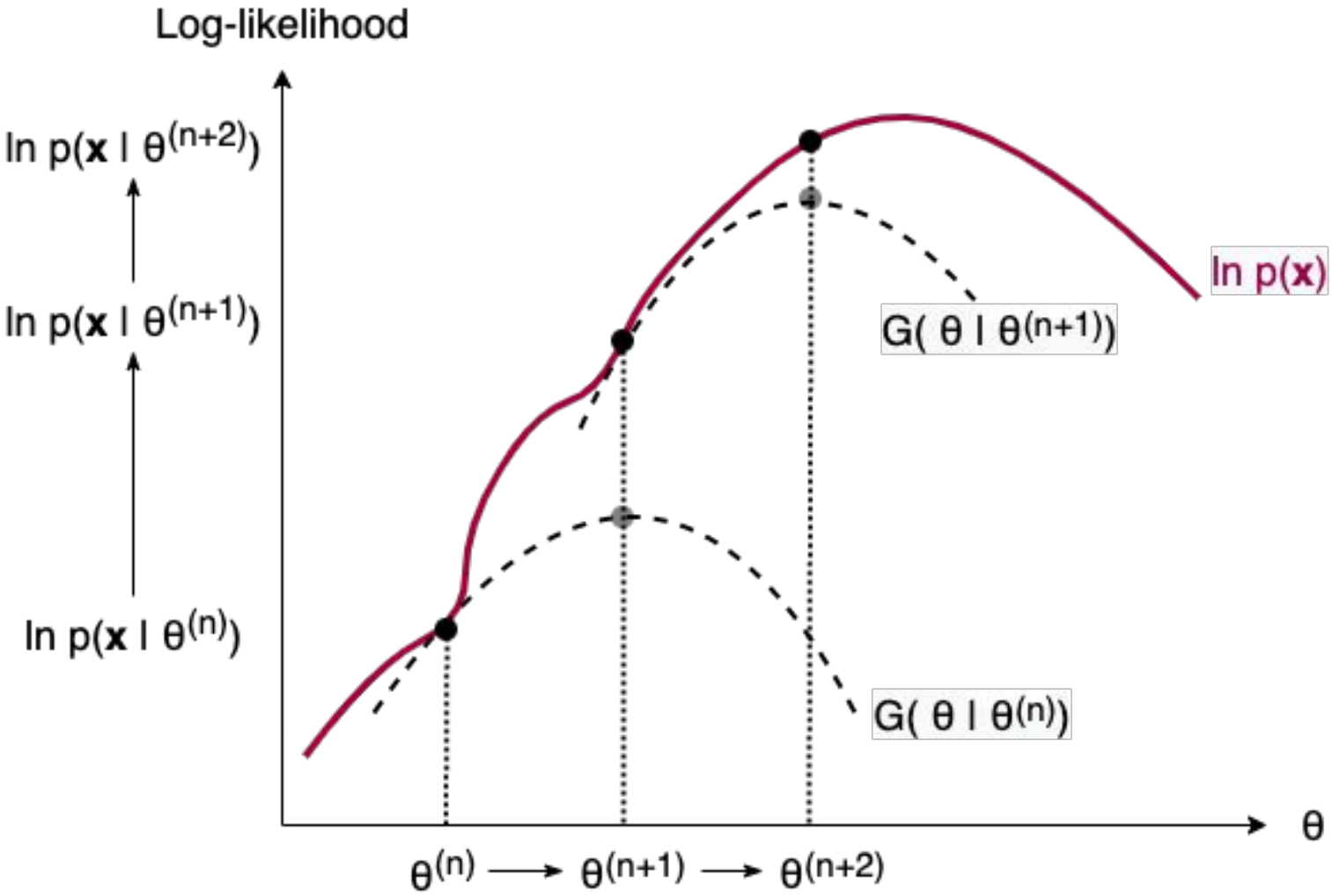
$$g(\mathbb{E}[X]) \geq \mathbb{E}[g(X)]$$

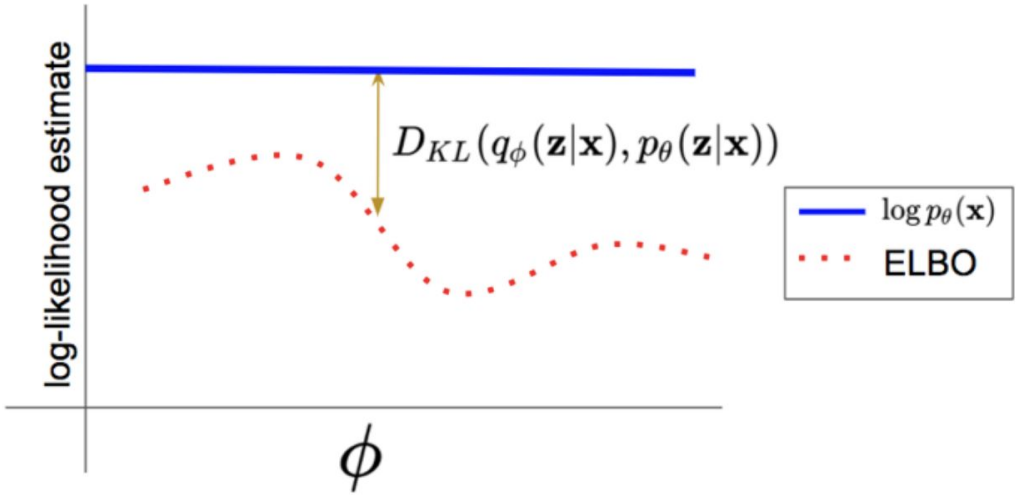


- In the last step, we use Jensen's Inequality. Since log is concave:

$$\log \left(\mathbb{E}_{q_n(z^{(n)})} \left[\frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right] \right) \geq \mathbb{E}_{q_n(z^{(n)})} \left[\log \frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right]$$

Called Evidence Lower Bound (**ELBO**)


Log-likelihood
ln p(x | θ(n+2))
ln p(x | θ(n+1))
ln p(x | θ(n))
ln p(x)
G(θ | θ(n+1))
G(θ | θ(n))
θ
θ(n) → θ(n+1) → θ(n+2)



$$\log p(\mathbf{x};\theta) \geq \sum_{\mathbf{z}} q(\mathbf{z};\phi)\log p(\mathbf{z},\mathbf{x};\theta) + H(q(\mathbf{z};\phi)) = \underbrace{\mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{ELBO}}$$

$$\log p(\mathbf{x};\theta) = \mathcal{L}(\mathbf{x};\theta,\phi) + D_{KL}(q(\mathbf{z};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))$$

The better $q(\mathbf{z};\phi)$ can approximate the posterior $p(\mathbf{z}|\mathbf{x};\theta)$, the smaller $D_{KL}(q(\mathbf{z};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))$ we can achieve, the closer ELBO will be to $\log p(\mathbf{x};\theta)$. Next: jointly optimize over θ and ϕ to maximize the ELBO over a dataset

- **Evidence lower bound** (ELBO) holds for any q

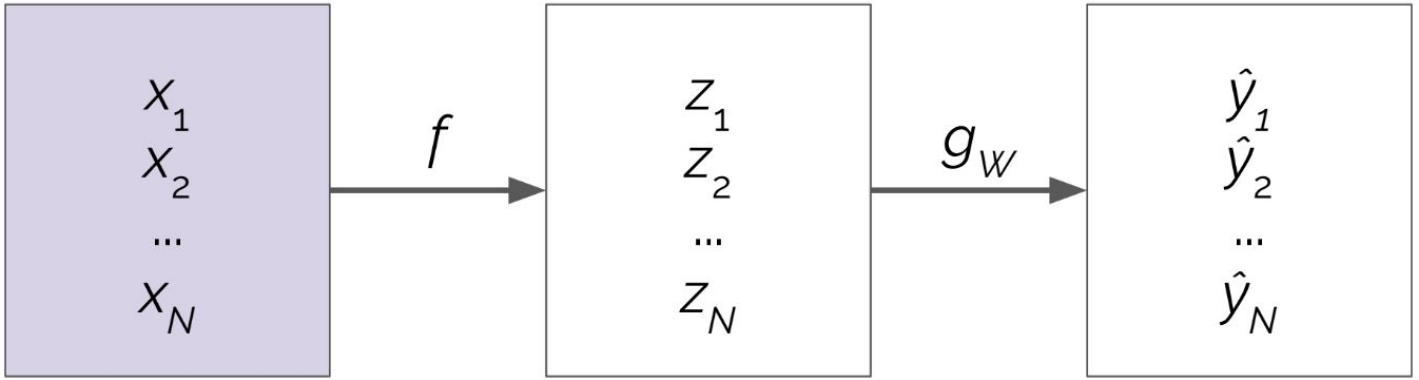
$$\log p(\mathbf{x};\theta) \geq \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z},\mathbf{x};\theta) + H(q)$$

- Equality holds if $q = p(\mathbf{z}|\mathbf{x};\theta)$ because $D_{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x};\theta)) = 0$

$$\log p(\mathbf{x};\theta) = \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z},\mathbf{x};\theta) + H(q)$$

- Confirms our intuition that we seek likely completions $\mathbf{z}$ given the observed values (evidence) $\mathbf{x}$.

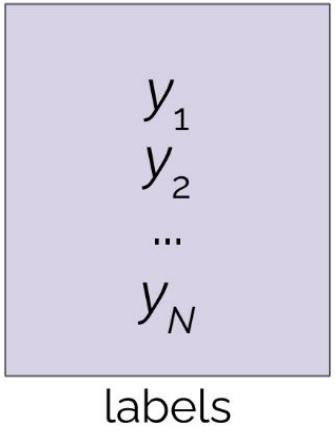
Recap | Supervised Learning



$z_i = f(x_i)$

$\hat{y}_i = g_W(z_i)$

$\hat{y}_i = g_W(f(x_i))$

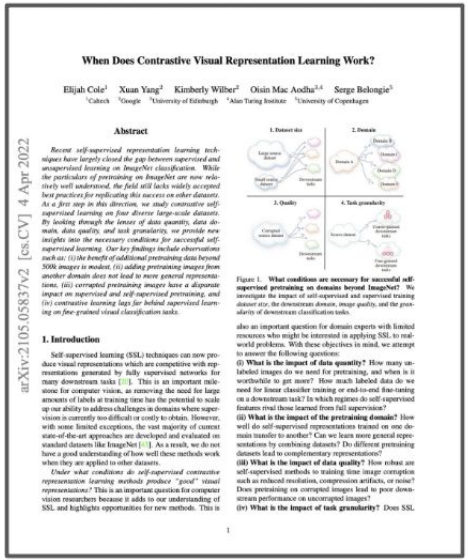


$\text{minimize}_W \sum_i L(\hat{y}_i, y_i)$

L: loss function

Recap | Supervised Learning

"feature engineering"



When Does Contrastive Visual Representation Learning Work?

Elijah Cole[1] Xuan Yang[2] Kimberly Wilber[2] Oisin Mac Aodha[3,4] Serge Belongie[5]
[1]Caltech [2]Google [3]University of Edinburgh [4]Alan Turing Institute [5]University of Copenhagen

arXiv:2105.05837v2 [cs.CV] 4 Apr 2022

Abstract

Recent self-supervised representation learning techniques have largely closed the gap between supervised and unsupervised learning on ImageNet classification. While the particulars of pretraining on ImageNet are now relatively well understood, the field still lacks widely accepted best practices for replicating this success on other datasets. As a first step in this direction, we study contrastive self-supervised learning on four diverse large-scale datasets. By looking through the lenses of data quantity, data domain, data quality, and task granularity, we provide new insights into the necessary conditions for successful self-supervised learning. Our key findings include observations such as: (i) the benefit of additional pretraining data beyond 500k images is modest, (ii) adding pretraining images from another domain does not lead to more general representations, (iii) corrupted pretraining images have a disparate impact on supervised and self-supervised pretraining, and (iv) contrastive learning lags far behind supervised learning on fine-grained visual classification tasks.

1. Introduction

Self-supervised learning (SSL) techniques can now produce visual representations which are competitive with representations generated by fully supervised networks for many downstream tasks [10]. This is an important milestone for computer vision, as removing the need for large amounts of labels at training time has the potential to scale up our ability to address challenges in domains where supervision is currently too difficult or costly to obtain. However, with some limited exceptions, the vast majority of current state-of-the-art approaches are developed and evaluated on standard datasets like ImageNet [43]. As a result, we do not have a good understanding of how well these methods work when they are applied to other datasets.

Under what conditions do self-supervised contrastive representation learning methods produce "good" visual representations? This is an important question for computer vision researchers because it adds to our understanding of SSL and highlights opportunities for new methods. This is also an important question for domain experts with limited resources who might be interested in applying SSL to real-world problems. With these objectives in mind, we attempt to answer the following questions:

(i) What is the impact of data quantity? How many unlabeled images do we need for pretraining, and when is it worthwhile to get more? How much labeled data do we need for linear classifier training or end-to-end fine-tuning on a downstream task? In which regimes do self-supervised features rival those learned from full supervision?

(ii) What is the impact of the pretraining domain? How well do self-supervised representations trained on one domain transfer to another? Can we learn more general representations by combining datasets? Do different pretraining datasets lead to complementary representations?

(iii) What is the impact of data quality? How robust are self-supervised methods to training time image corruption such as reduced resolution, compression artifacts, or noise? Does pretraining on corrupted images lead to poor downstream performance on uncorrupted images?

(iv) What is the impact of task granularity? Does SSL

Figure 1. **What conditions are necessary for successful self-supervised pretraining on domains beyond ImageNet?** We investigate the impact of self-supervised and supervised training dataset *size*, the downstream domain, image quality, and the granularity of downstream classification tasks.

1

raw document data

f

Word frequencies?

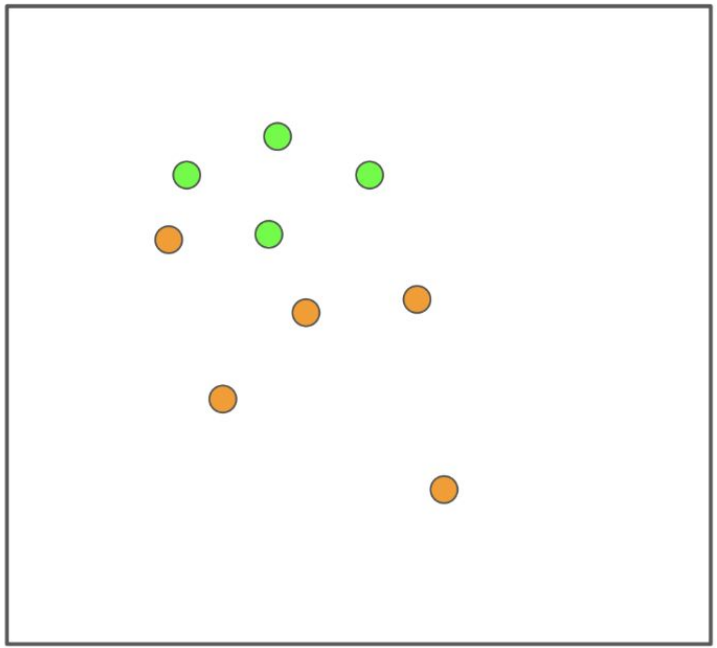
Word co-occurrences?

Section lengths?

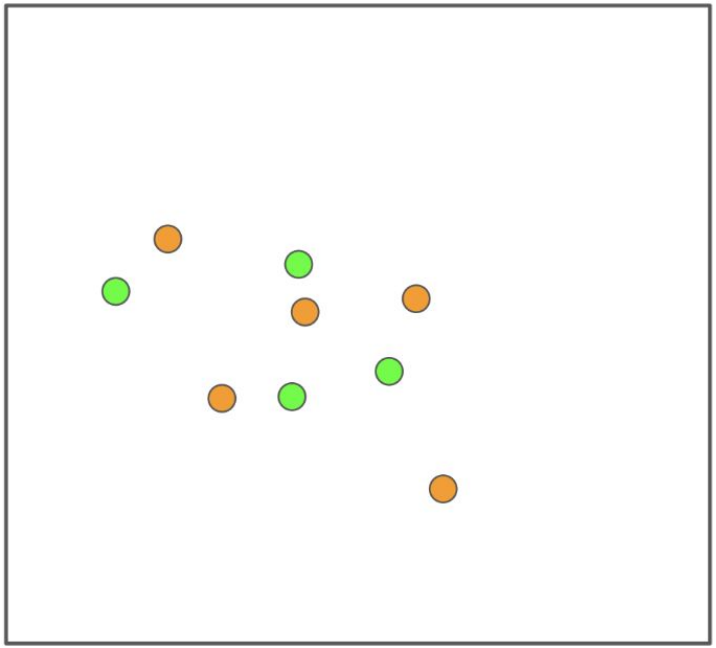
features

Recap | Supervised Learning

good* features



bad* features

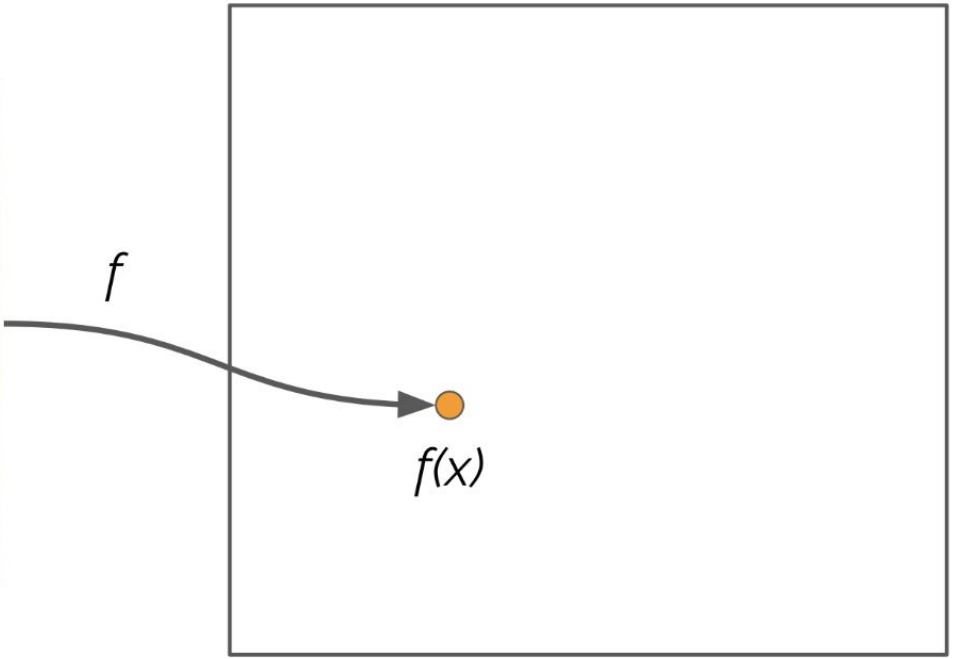


*For solving this binary classification task with a linear classifier.

Recap | Supervised Learning

It's hard to design good features, especially for complex objects.

x



Idea: Learn f from data.

Recap | Supervised Learning

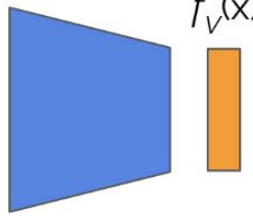
In **deep learning**, we use special models (*deep neural networks*) which allow us to do *supervised learning* with *raw data* as inputs, eliminating the need for hand-designed features.

Example: Image Classification

"Leopard tortoise"

Example: Image Classification

image
x

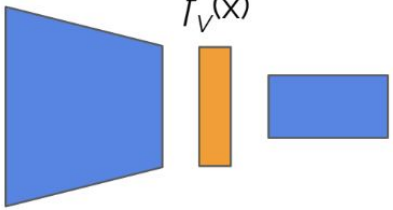


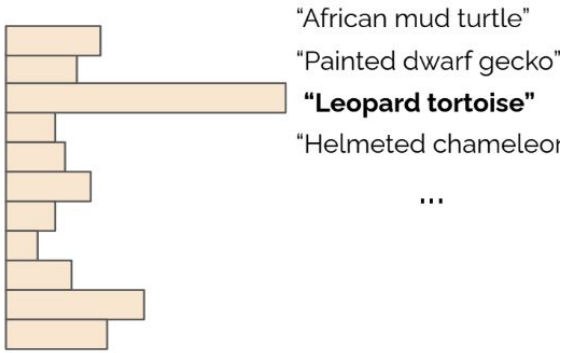
feature
extractor
f_V

representation
$f_V(x)$

Example: Image Classification

image
x

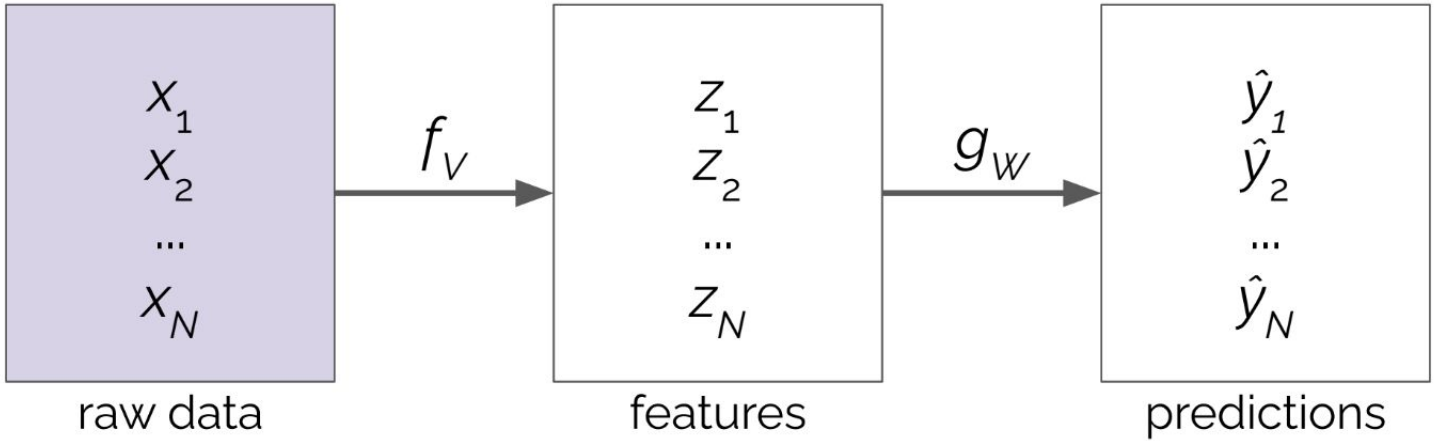




predicted distribution
$\mathbf{y}' = softmax(g_W(f_V(x)))$

$$V^*, W^* = \text{argmin}_{V,W} \, E[L(\hat{y}, y)]$$

Recap | Supervised Learning



$z_i = f_V(x_i)$

$\hat{y}_i = g_W(z_i)$

$\hat{y}_i = g_W(f_V(x_i))$

y_1
y_2
...
y_N

labels

$\text{minimize}_{V,W} \Sigma_i L(\hat{y}_i, y_i)$

L: loss function

Recap | Supervised Learning

Summary:

- Supervised learning from hand-designed features
  - Feature engineering
- Supervised learning from raw data
  - AKA deep learning, representation learning
- Important ideas
  - Feature extractor + task-specific head
  - Task difficulty depends on features
  - Learning features from data is preferable
  - Lots of free parameters => need lots of data

Recap

Story so far:

- We know how to learn a representation from labeled data.
- We know how to re-use and evaluate a learned representation.

- **What if you don't have enough labeled data to learn the representation?**

Self-Supervised (Unsupervised) Learning

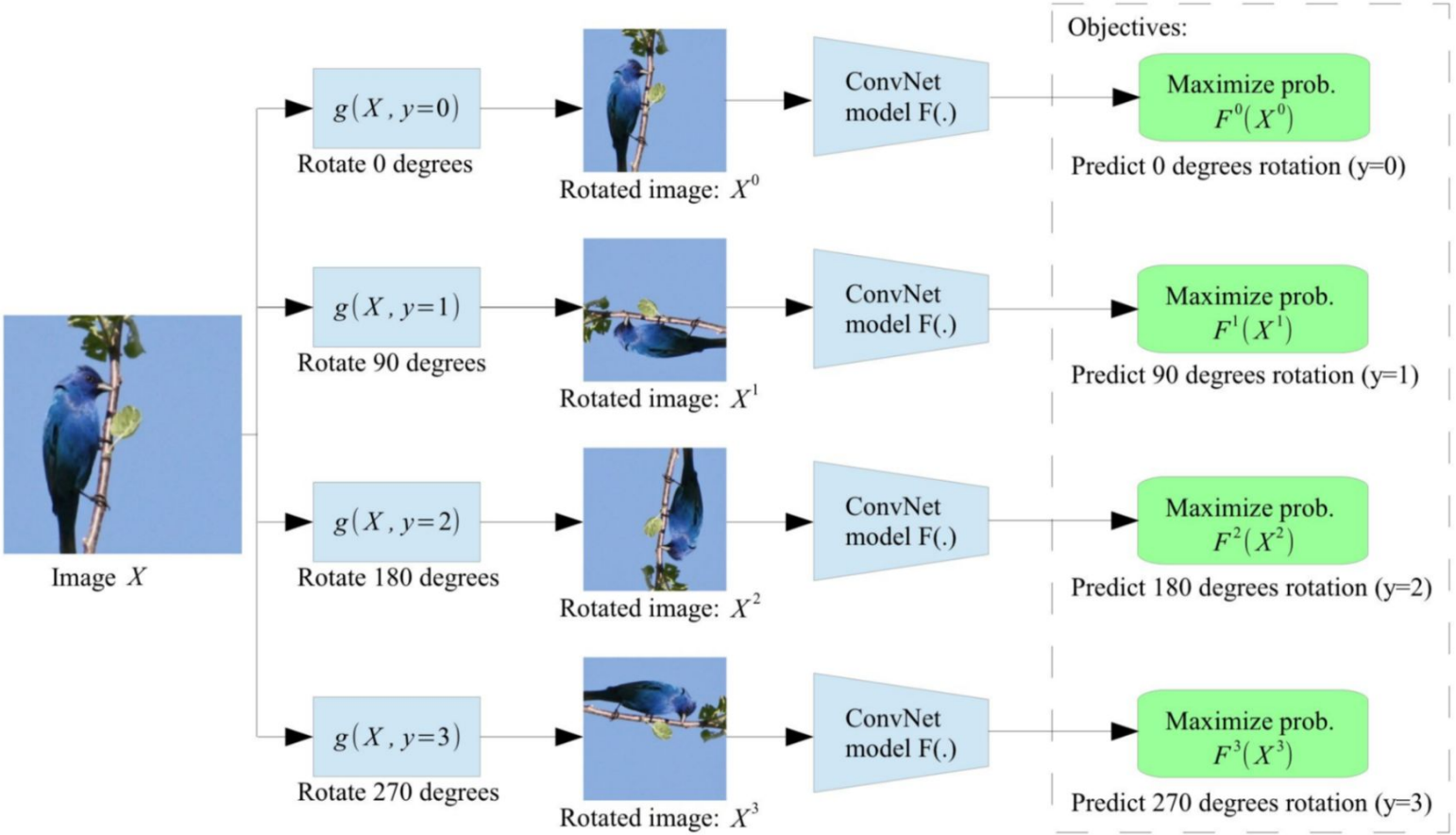
SSL Basics | Motivation

Self-supervised pretraining looks a lot like supervised pretraining.

The only difference is that the labels are automatically generated *pseudo-labels* instead of human-curated ground truth labels. This is called a **pretext task**.

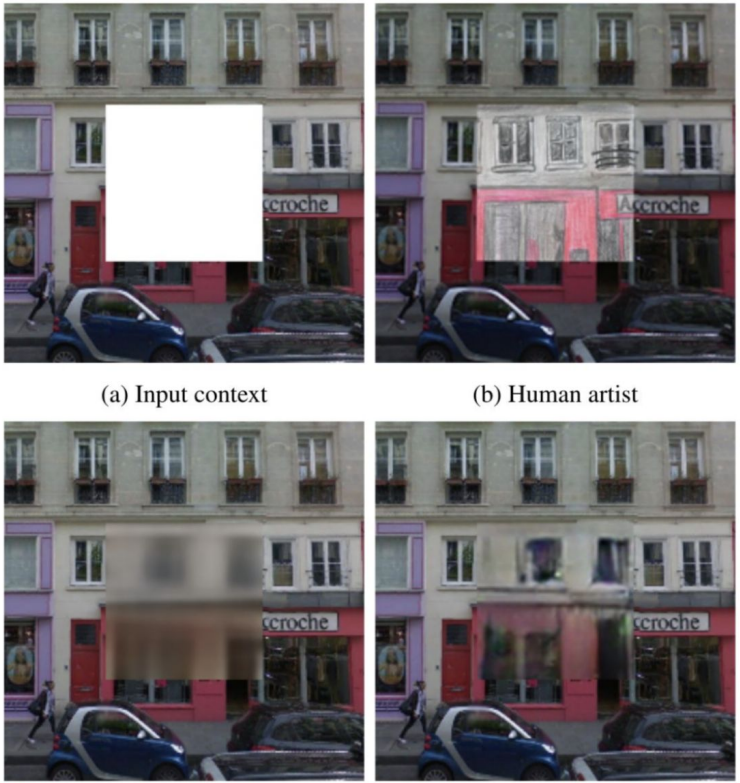
Then we do transfer learning.

SSL Basics | Pretext Task Examples



https://arxiv.org/abs/1803.07728

SSL Basics | Pretext Task Examples



(a) Input context (b) Human artist

(c) Context Encoder ($L2$ loss) (d) Context Encoder ($L2$ + Adversarial loss)

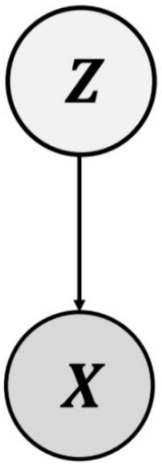
Inpainting (Pathak et al. 2016)

https://arxiv.org/abs/1604.07379

(Variational) AutoEncoder

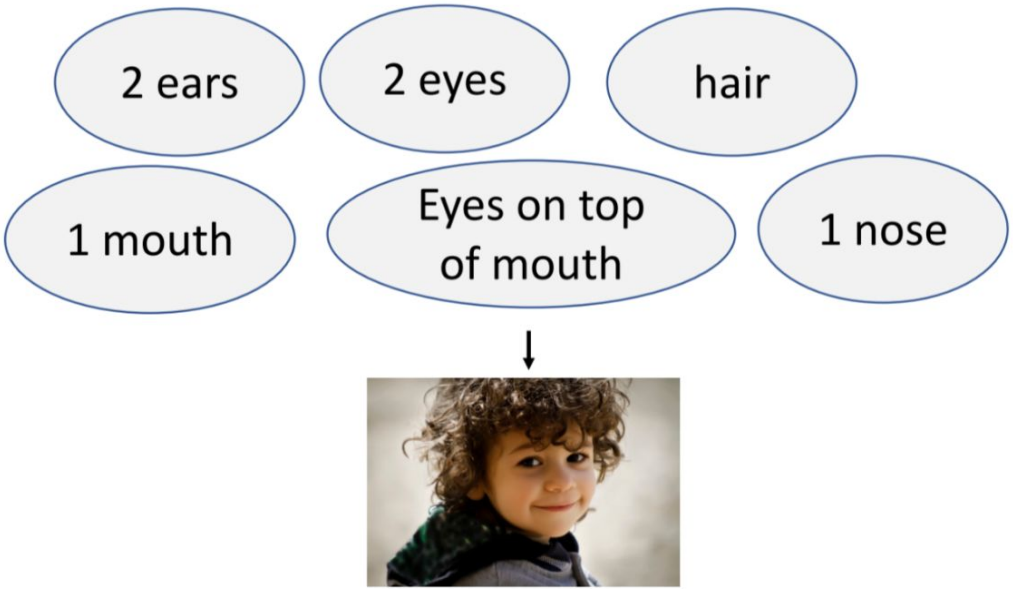
Variational Autoencoders

Belong in the family of latent variable probabilistic models that can infer the hidden structure in the underlying data



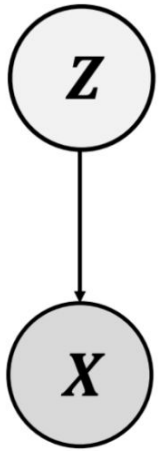
Latent Variable
(**unobserved**)

Observed Variable
(**data**)



Variational Autoencoders

Belong in the family of latent variable probabilistic models that can infer the hidden structure in the underlying data



Latent Variable
(**unobserved**)

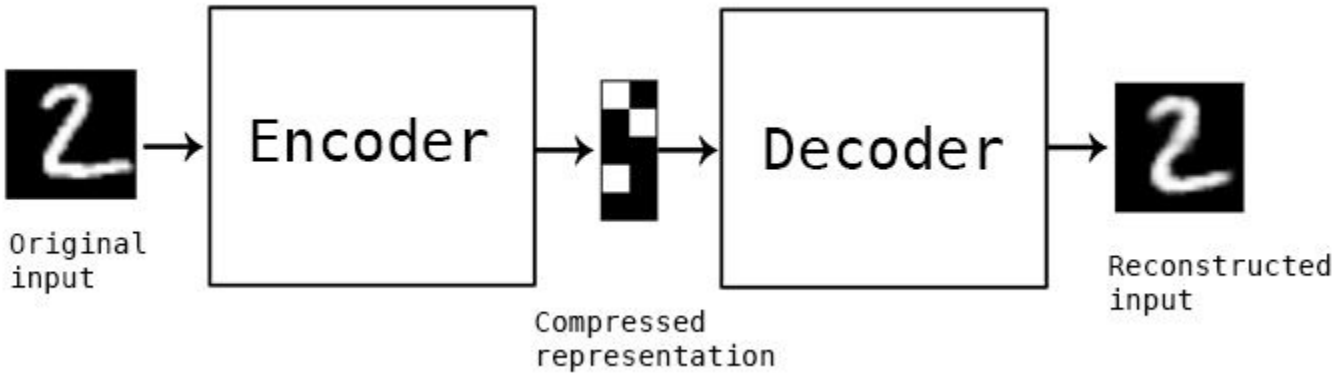
Observed Variable
(data)

The goal of variational autoencoders is twofold:

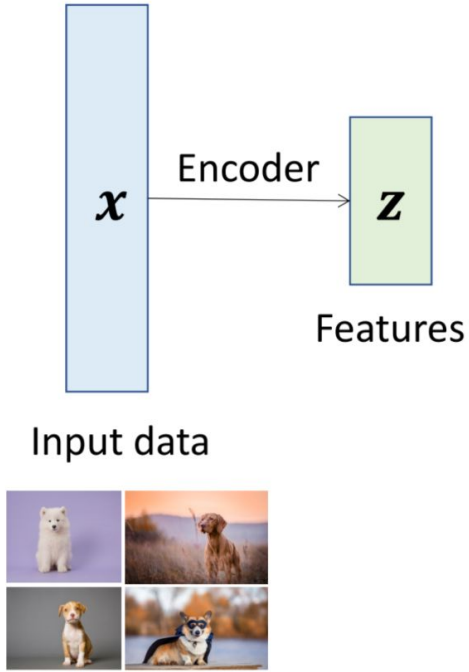
1. Model the data distribution $p(\boldsymbol{x})$ – as is the case with all generative models

2. Extract a latent representation $\boldsymbol{Z}$ that "describes" the data.

Variational Autoencoders

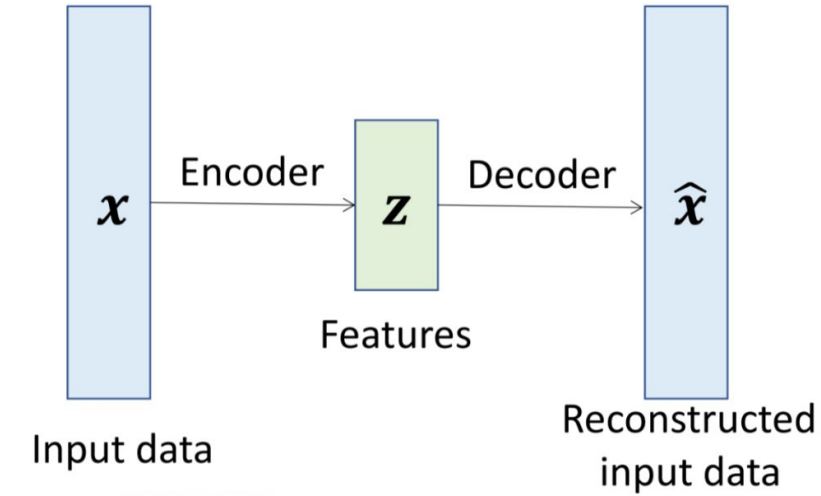


~~Variational~~ Autoencoders



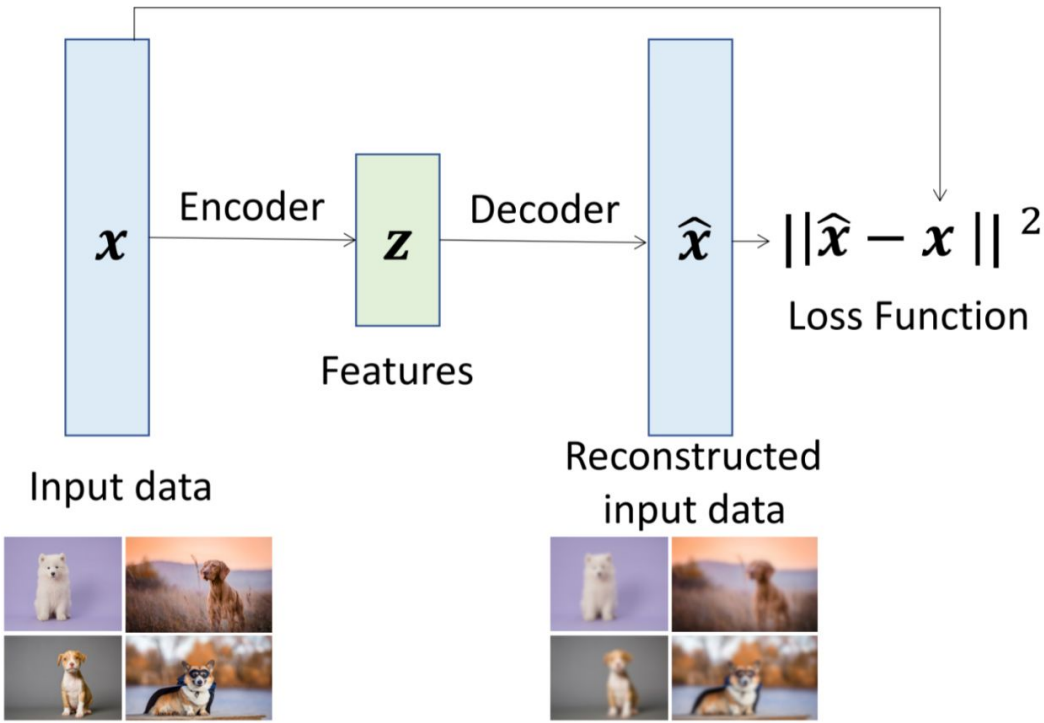
- The encoder extracts features which contain useful information from the data and can be used for downstream tasks
- We don't have labels, only data. So, we want to use unsupervised methods to learn these features.
- How?

~~Variational~~ Autoencoders



- Use the features z to reconstruct the input data x with a decoder

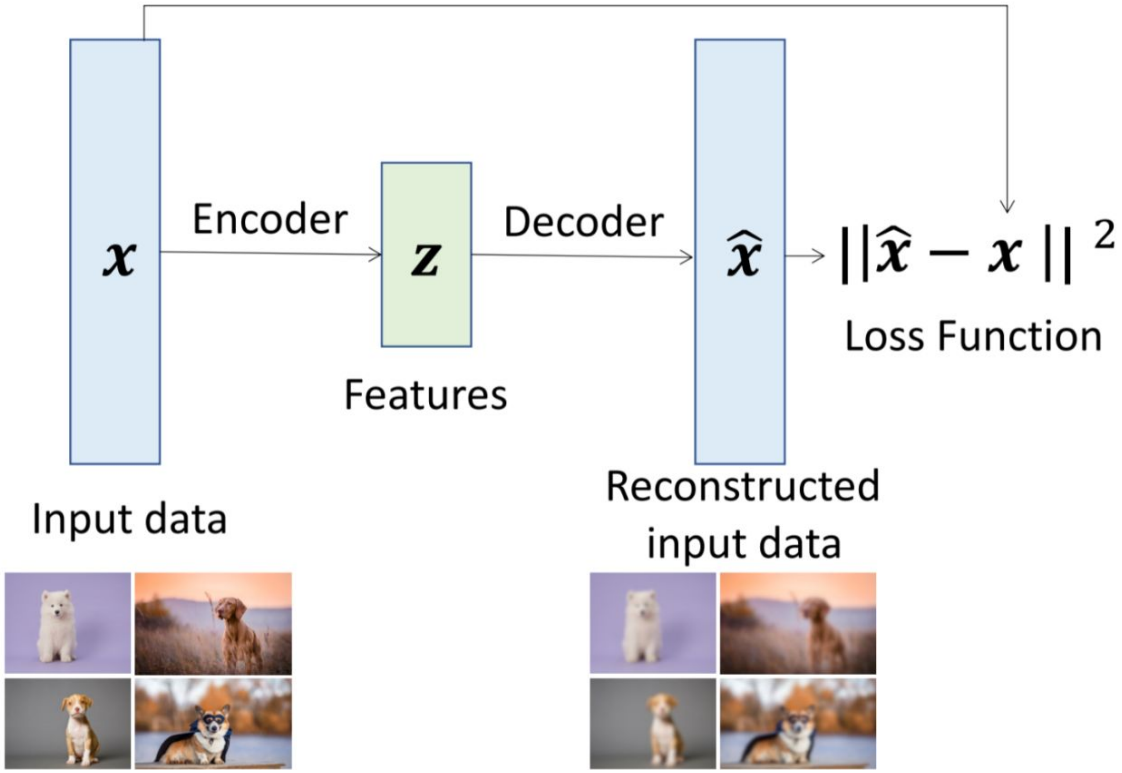
~~Variational~~ Autoencoders



During training:

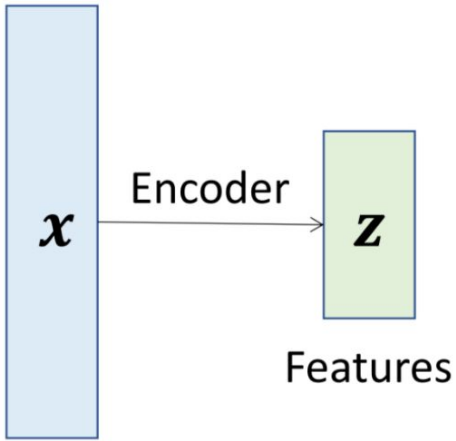
- We minimize an L2 distance between the input and the reconstructed data
- No labels used. Just data!

Autoencoders



The features z need to be lower dimensional than the data (→ dimensionality reduction, data compression)

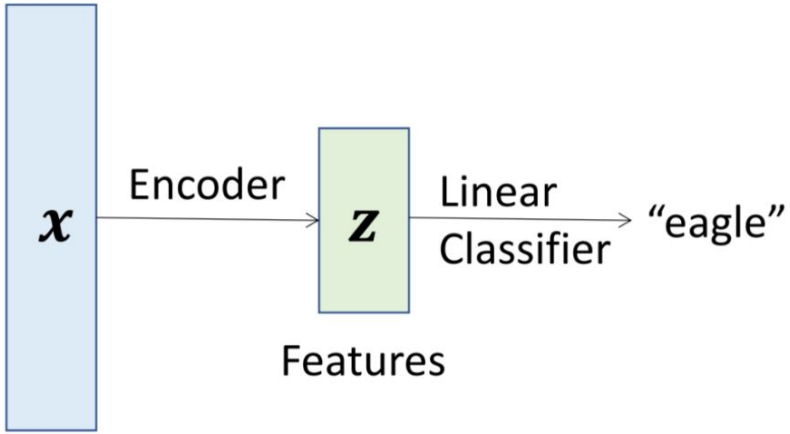
Autoencoders



After training:

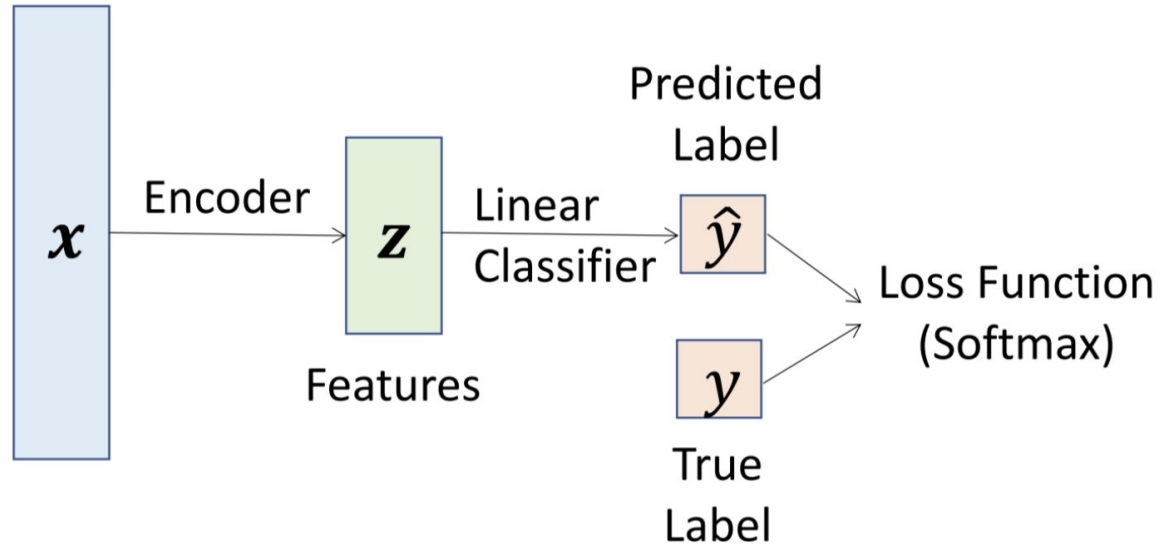
- Throw away the decoder
- Use encoder to encode input images to “useful” features
- These features can be used for downstream tasks because they encode useful bits about the input (since one can reconstruct the input from them)

Autoencoders: Downstream Tasks



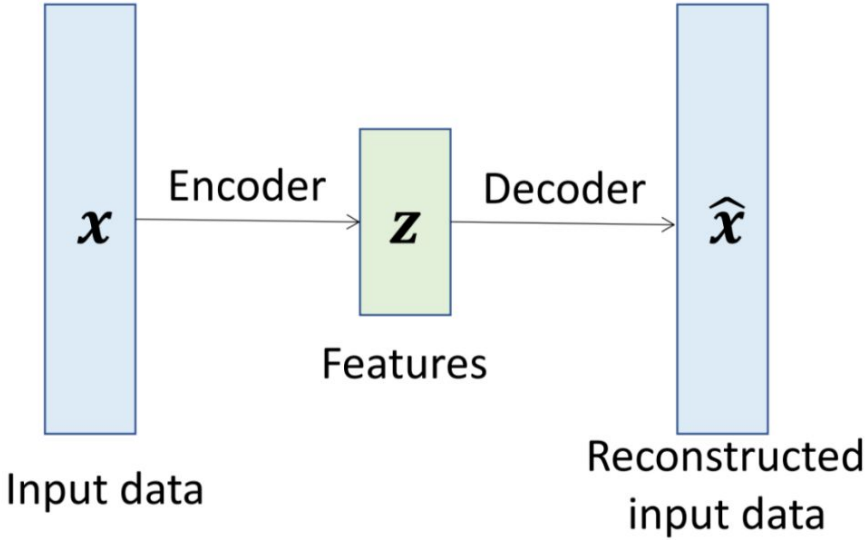
- A linear classifier inputs the features of the input image
- We train on a downstream supervised task with few annotated data points

Autoencoders: Downstream Tasks



Finetune the encoder and the linear classifier

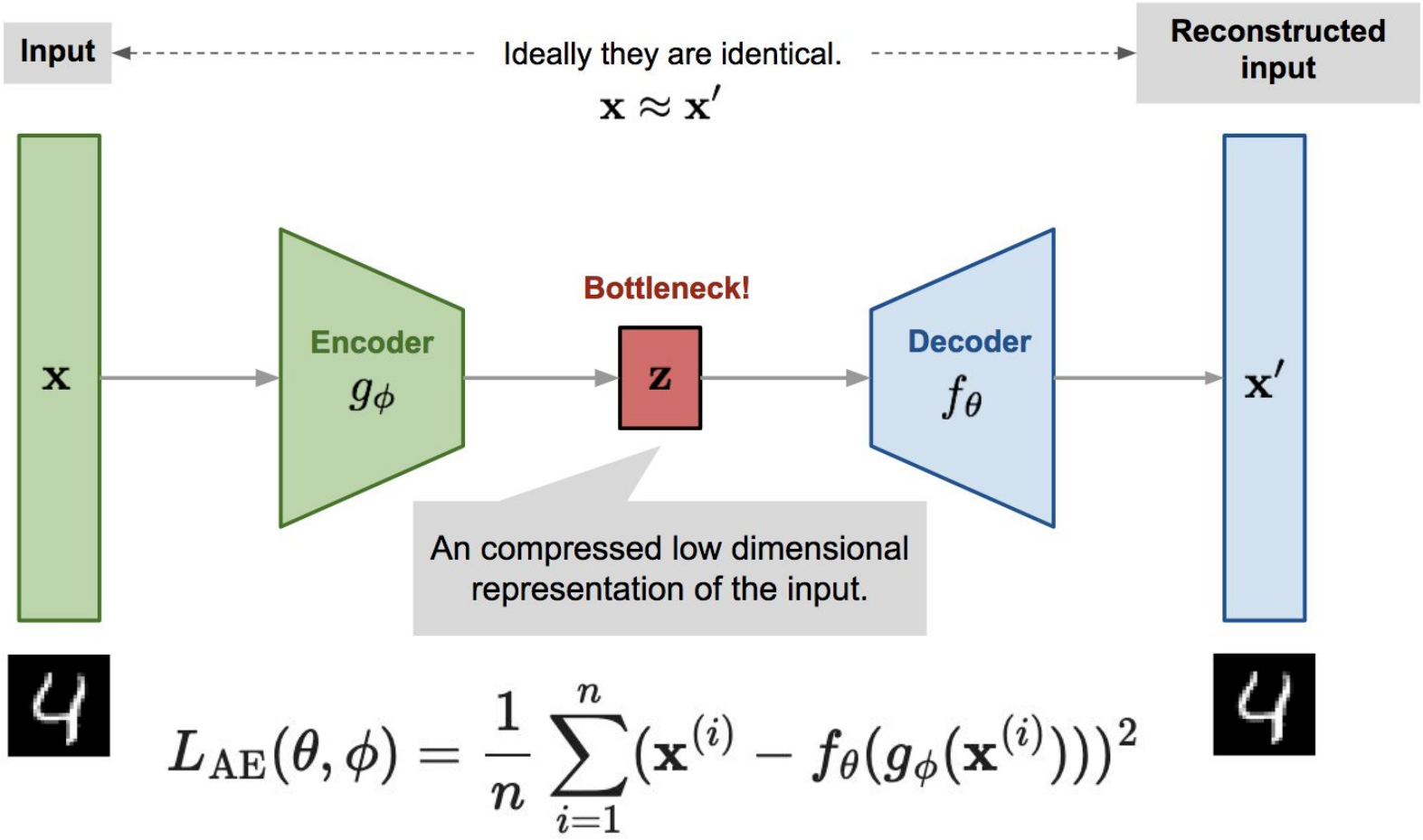
Autoencoders



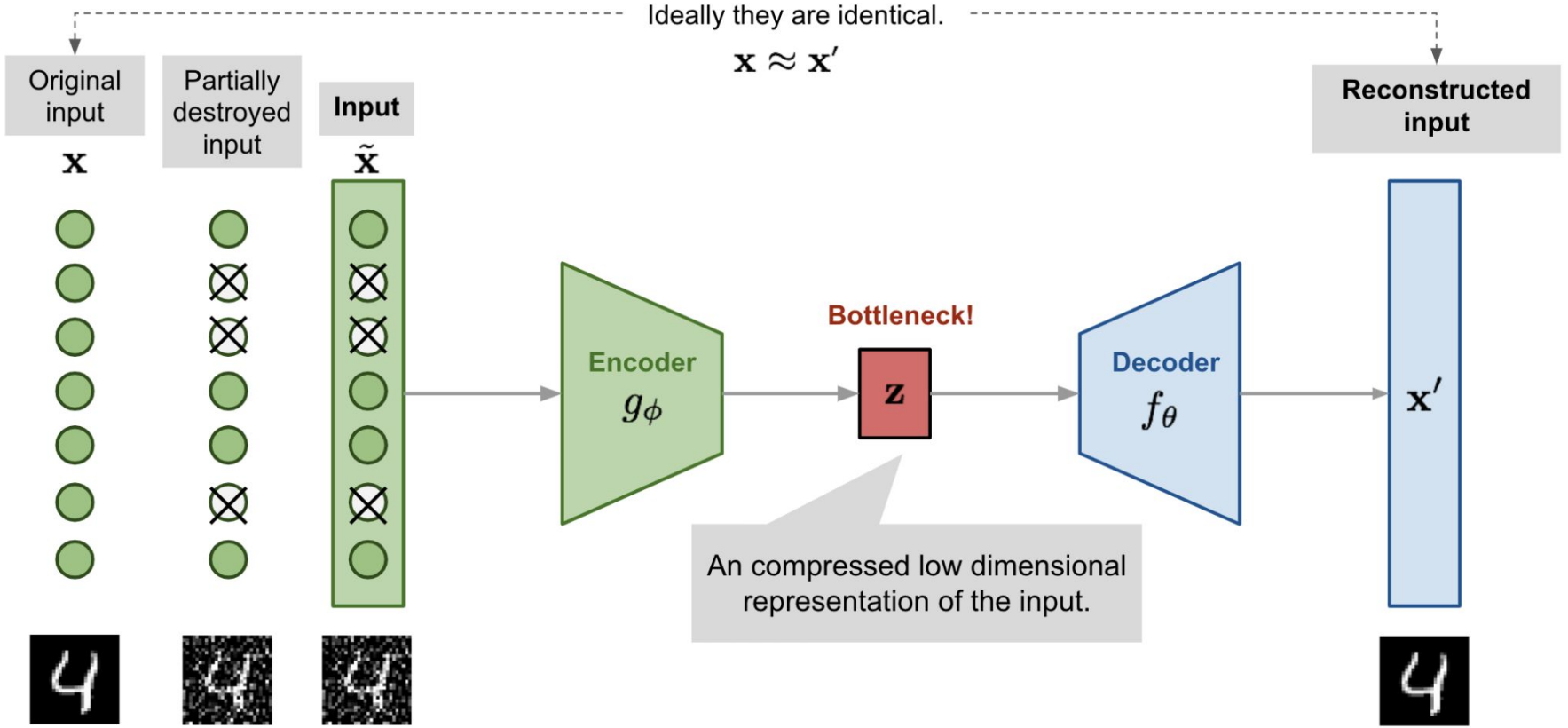
✓ Autoencoders learn latent features for data without any labels

✓ The features can be used to initialize a supervised task

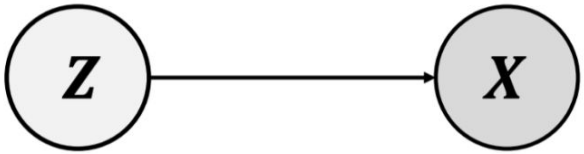
But they are not probabilistic and cannot be used to sample new data from. Remember we want to capture $p(x)$ and sample from it.


Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.
$$L_{\text{AE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2$$

Denoising Auto-Encoder



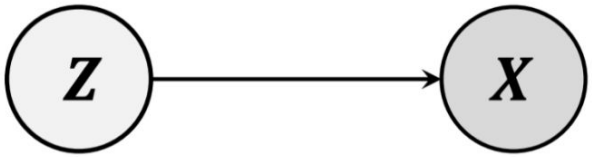
Variational Autoencoders



Assume we have training data $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{N}$ which are generated by an unobserved (latent) representation $\boldsymbol{z}$.

For example, $\boldsymbol{x}$ is a dog image, and $\boldsymbol{z}$ is a latent vector which captures all the information one might need to generate an image of a dog, e.g. it has 2 eyes, 2 ears, 4 legs, it's cute etc.

Variational Autoencoders



$p(z)$ $p(x|z)$

The joint distribution

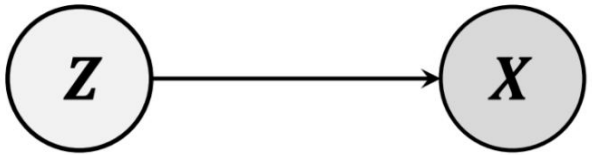
$$p(x, z) = p(x|z)p(z)$$

After training,

1. Sample from prior: $\hat{z} \sim p(z)$
2. Sample data from conditional: $\hat{x} \sim p(x|\hat{z})$

Intuitively, first generate the “high-level” semantic information about the data. Then generate the data.

Variational Autoencoders



$p(z)$ $p(x|z)$

Assume a simple prior $p(z)$, e.g. Gaussian

Represent $p(x|z)$ with a neural network

- The conditional distribution is parametrized, $p_\theta(x|z)$
- This neural network is the decoder

Starting from Gaussian, we can approximate very complicated data distribution (even real-world images / videos)

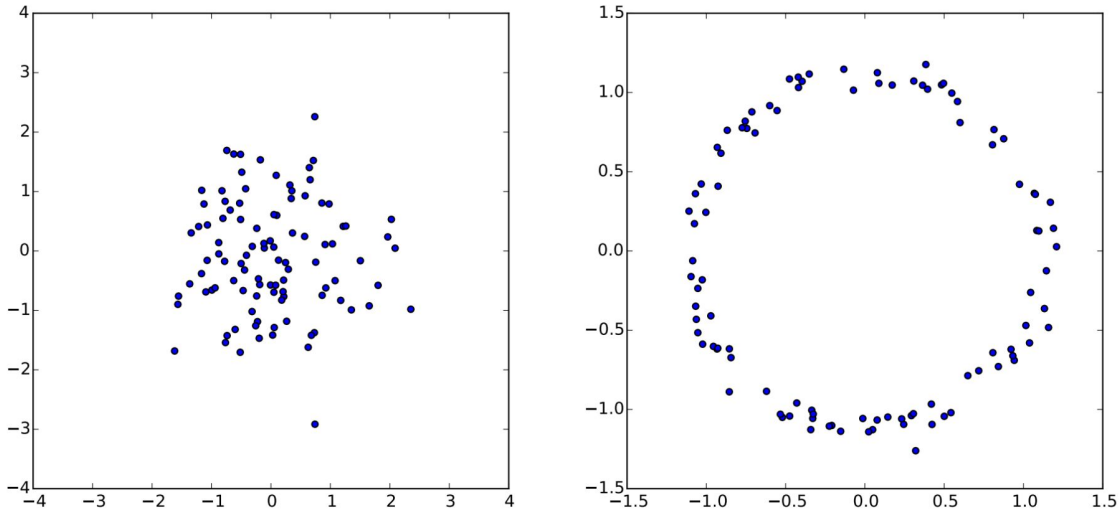
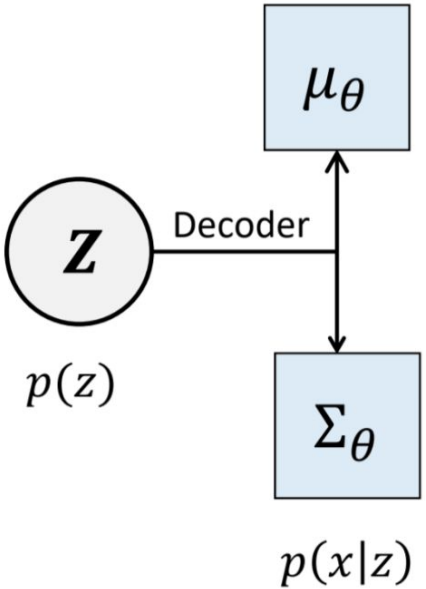


Figure 2: Given a random variable z with one distribution, we can create another random variable $X = g(z)$ with a completely different distribution. Left: samples from a gaussian distribution. Right: those same samples mapped through the function $g(z) = z/10 + z/||z||$ to form a ring. This is

Variational Autoencoders

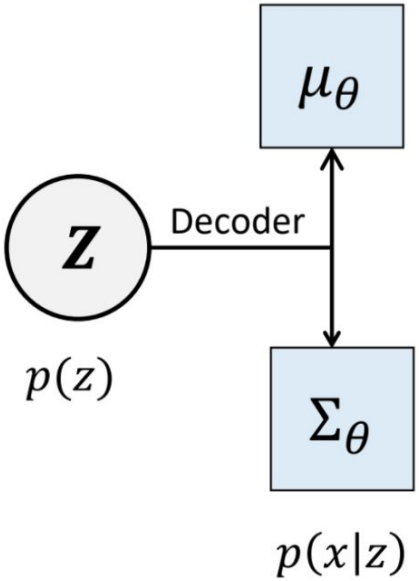


Unlike in original autoencoders, the decoder in variational autoencoders is probabilistic. It inputs z and outputs μ_θ and (diagonal) covariance Σ_θ

$$p(x|z) = N(\mu_\theta(z), \Sigma_\theta(z))$$

One can sample the decoder by sampling from a normal distribution with mean μ_θ and covariance Σ_θ

Variational Autoencoders

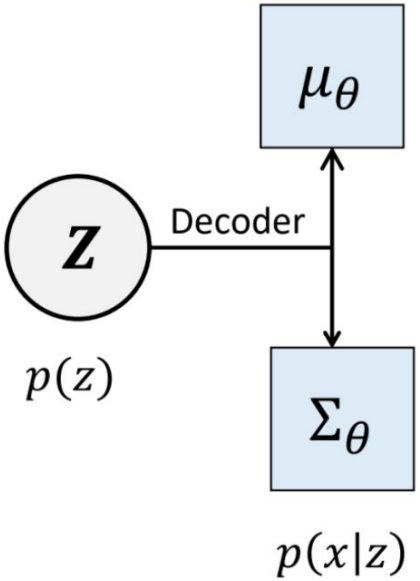


Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Note that we don't observe z, so to get the density we need to marginalize

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

Variational Autoencoders



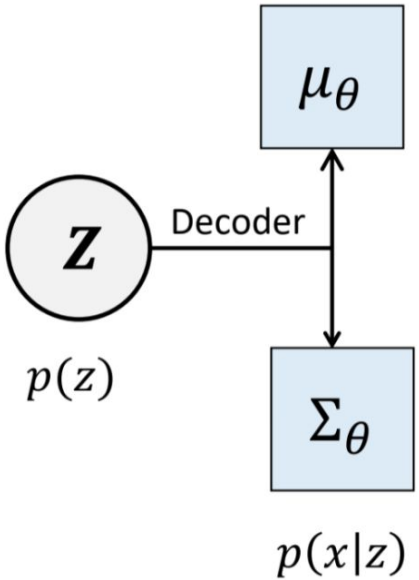
Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Note that we don't observe z, so to get the density we need to marginalize

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

Impossible to integrate over all z. Intractable!

Variational Autoencoders

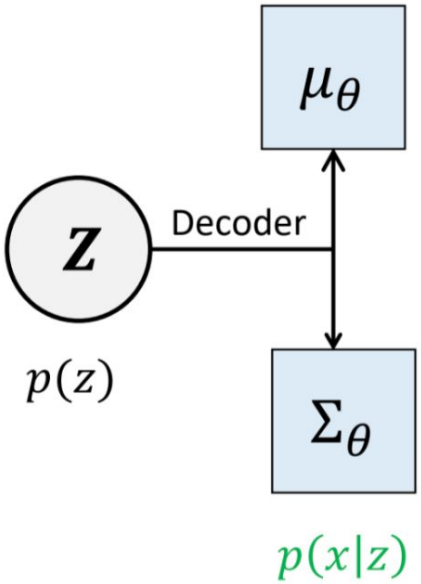


Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Bayes' rule says:

$$p(x) = \frac{p(x|z)p(z)}{p(z|x)}$$

Variational Autoencoders



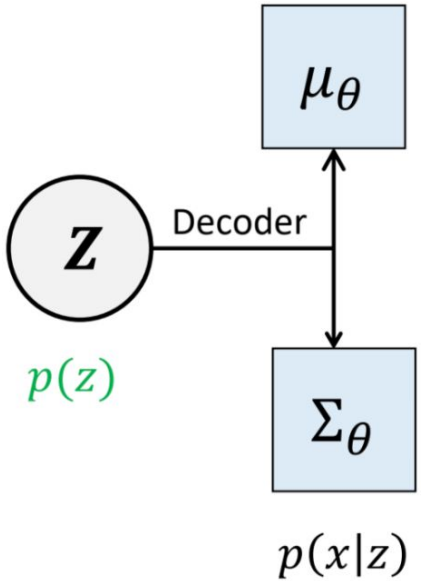
Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Bayes' rule says:

$$p(x) = \frac{p(x|z)p(z)}{p(z|x)}$$

Easy! My decoder network outputs $p(x|z)$ parametrized as a gaussian with μ_θ, Σ_θ

Variational Autoencoders



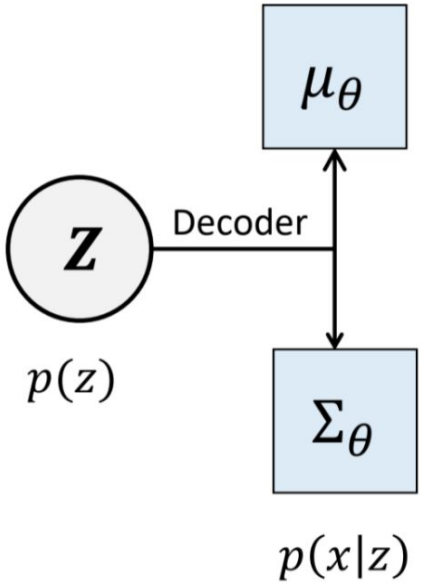
Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Bayes' rule says:

$$p(x) = \frac{p(x|z)p(z)}{p(z|x)}$$

Easy! I have assumed a Gaussian prior $N(0, I)$.

Variational Autoencoders



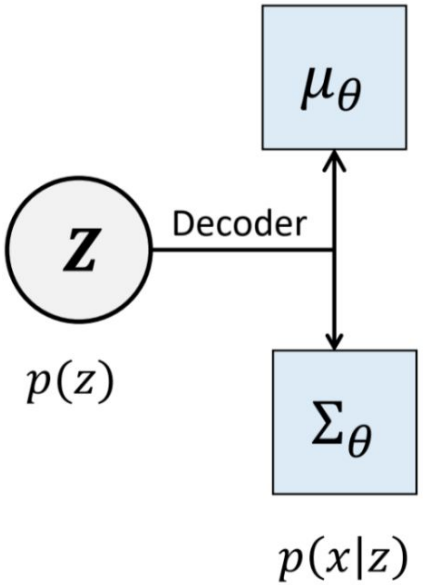
Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Bayes' rule says:

$$p(x) = \frac{p(x|z)p(z)}{p(z|x)}$$

Not easy! I have no way of computing this.

Variational Autoencoders

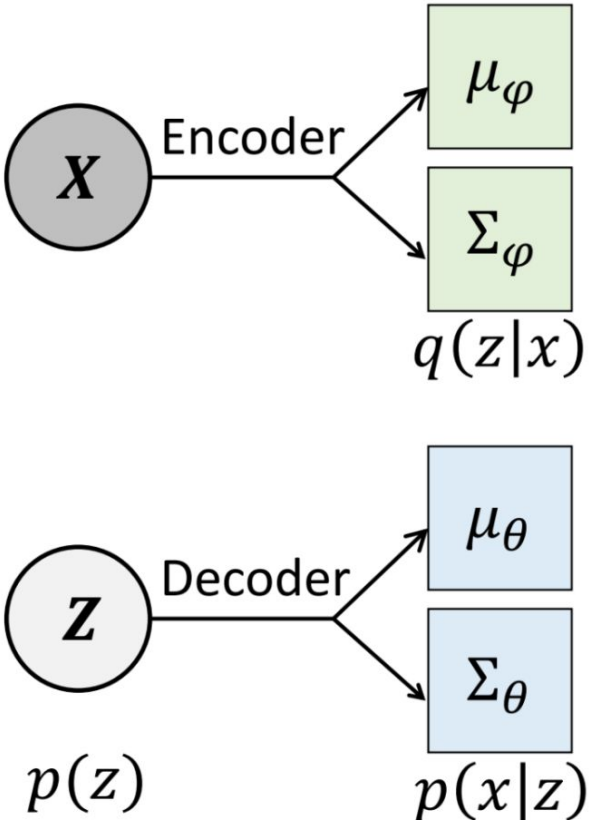


$$p(x) = \frac{p(x|z)p(z)}{p(z|x)}$$

We introduce a new parametric distribution $q(z|x)$which will approximate the unknown $\mathrm{p}(z|x)$

$$q(z|x) \approx \mathrm{p}(z|x)$$

Variational Autoencoders



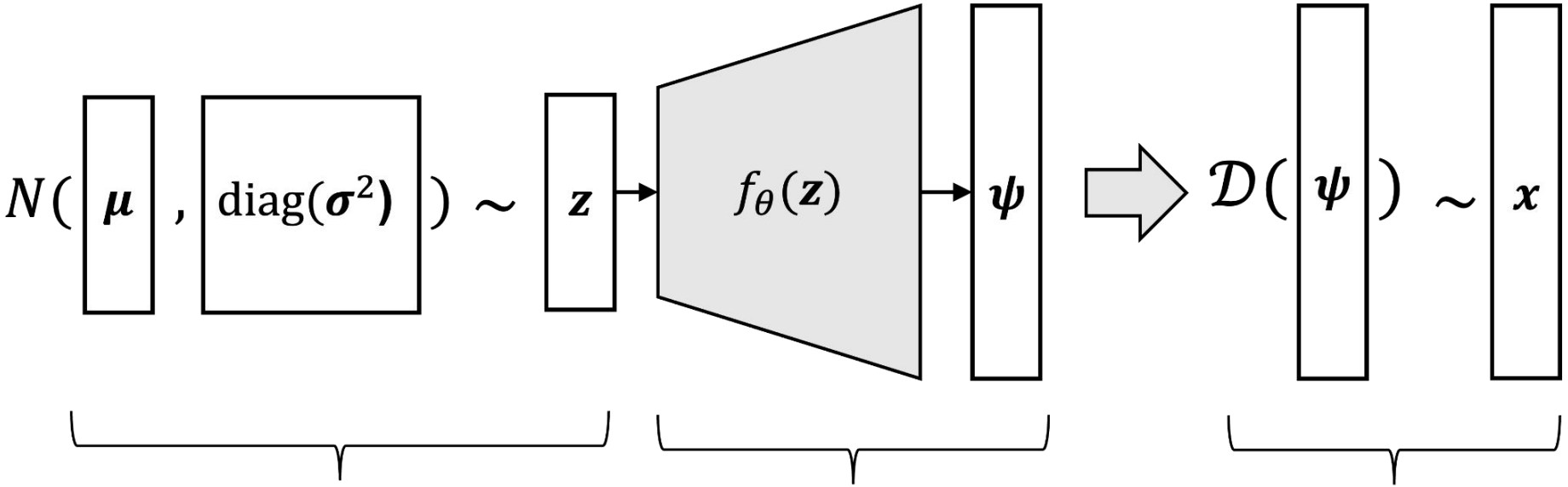
$$p(x) = \frac{p(x|z)\,p(z)}{p(z|x)} \approx \frac{p(x|z)p(z)}{q(z|x)}$$

$q(z|x)$ is parametrized by an encoder network with parameters φ which maps the input data to mean μ_φ and a **diagonal** covariance Σ_φ such that

$$q_\varphi(z|x) = \mathrm{N}(\mu_\varphi, \Sigma_\varphi)$$

Why diagonal Σ_φ?

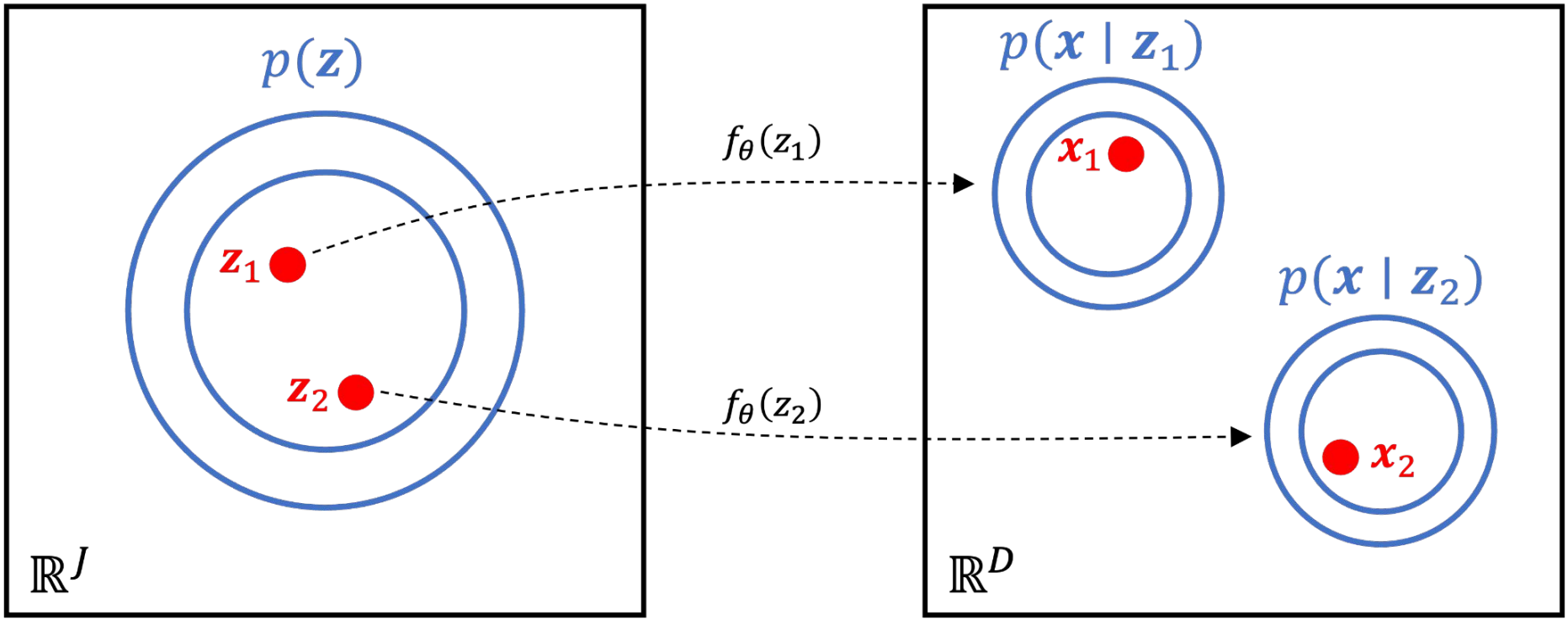
It makes computations easy.



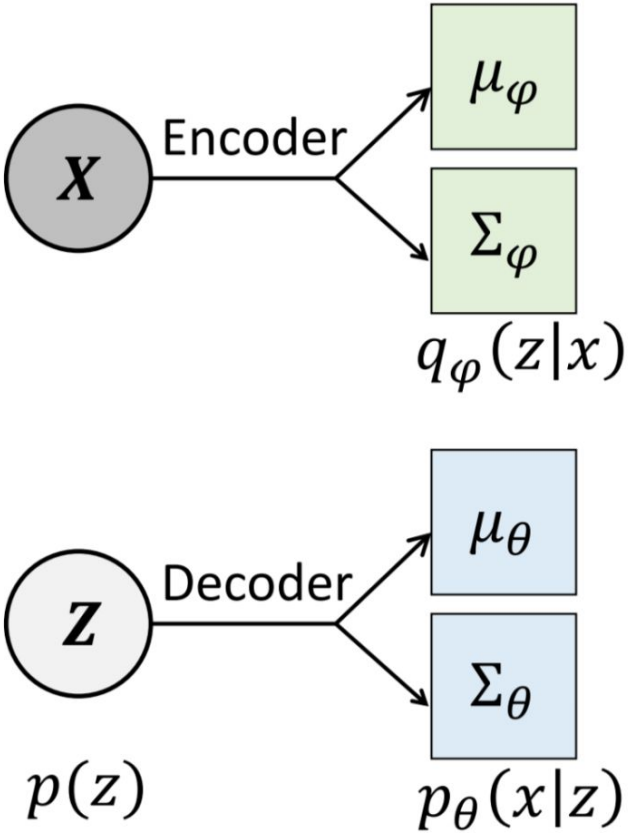
Sample $\boldsymbol{z}$ from a normal distribution parameterized by $\boldsymbol{\mu}$ and a diagonal covariance matrix with the values in $\boldsymbol{\sigma}^2$ along the diagonal

Decode $\boldsymbol{z}$ into the parameters, $\boldsymbol{\psi}$, used to sample $\boldsymbol{x}$

Sample $\boldsymbol{x}$


$p(z)$
z_1
z_2
$\mathbb{R}^J$
$f_\theta(z_1)$
$f_\theta(z_2)$
$p(x \mid z_1)$
x_1
$p(x \mid z_2)$
x_2
$\mathbb{R}^D$

Variational Autoencoders

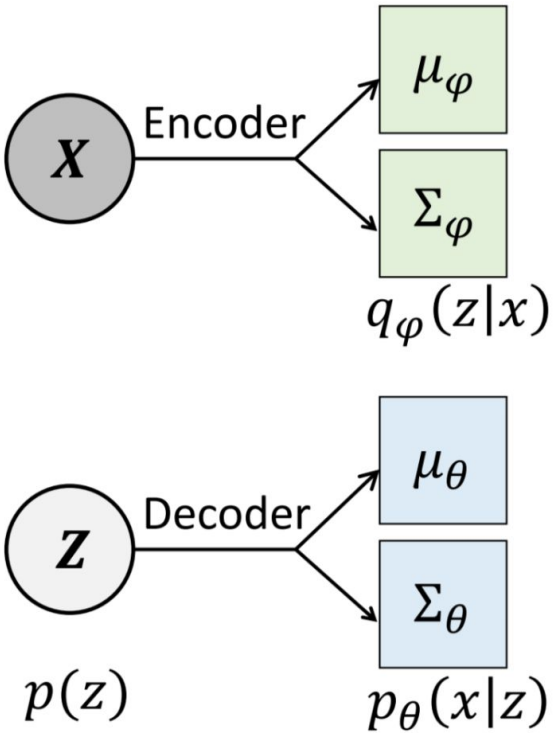


Optimization in VAEs involves jointly optimizing

1. The generative model parameters θ (decoder) to reduce the reconstruction error between input and output

2. The encoder parameters φ to model the posterior $q_\varphi(z|x)$

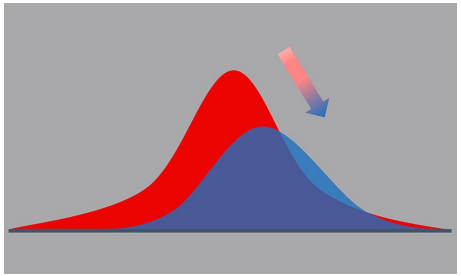
$$max_{\theta,\varphi}\, \mathcal{L}_{\theta,\varphi}(x) = \mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)] - d_{KL}(q_\varphi(z|x)|p(z))$$



Reconstruction Loss: This term measures given encoded latent code (z), how well we can reconstruct data (x)

Encoder Prior Loss: This term measures the distance between encoded latent distribution q(z|x) to prior gaussian p(z).

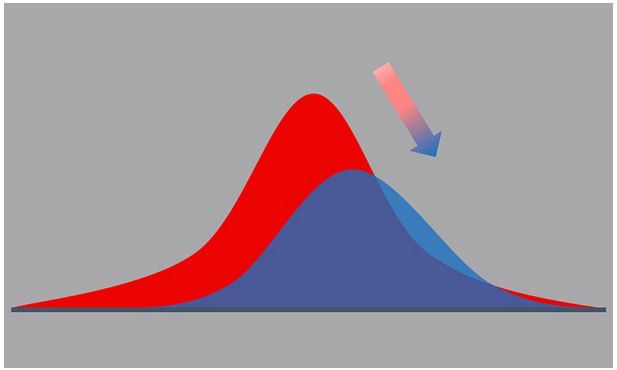
After this term becomes small



How to derive it?

Note that we want to learn q(z|x) to approximate p(z|x).

Let's start with the KL Divergence between them.



$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x$$

$$d_{KL}(q_\varphi(z|x)\big|\, p_\theta(z|x)\big) = \mathbb{E}_{z\sim q_\varphi}\big[\log q_\varphi(z|x) - \log\, p_\theta(z|x)\big]$$

Definition of KL divergence

$$d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(z|x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z) + \log\, p_\theta(x)]$$

From Bayes' rule

$$d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(z|x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z) + \log\, p_\theta(x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z)] + \log\, p_\theta(x)$$

Since $p_\theta(x)$ does not depend on z we can move it out of the expectation

$$d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(z|x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z) + \log\, p_\theta(x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z)] + \log\, p_\theta(x)$$

Ah! This is exactly the objective we want to maximize! Maximum likelihood estimation (MLE)

$$d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(z|x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z) + \log\, p_\theta(x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z)] + \log\, p_\theta(x)$$

$\Rightarrow$

$$\log\, p_\theta(x) - d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)] - d_{KL}(q_\varphi(z|x)|p(z))$$

The definition of KL divergence

$$d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(z|x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z) + \log\, p_\theta(x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z)] + \log\, p_\theta(x)$$

$$\Rightarrow$$

$$\underbrace{\log\, p_\theta(x)} - d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)] - d_{KL}(q_\varphi(z|x)|p(z))$$

The quantity we want to maximize!

$$d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(z|x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z) + \log\, p_\theta(x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z)] + \log\, p_\theta(x)$$

$\Rightarrow$

$$\log\, p_\theta(x) - \underbrace{d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x))} = \mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)] - d_{KL}(q_\varphi(z|x)|p(z))$$

≥ 0. We can't compute this but this term will become small if the encoder is high capacity. Why?

$$d_{KL}(q_\varphi(z|x)|\, p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(z|x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z) + \log\, p_\theta(x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z)] + \log\, p_\theta(x)$$

$\Rightarrow$

$$\log\, p_\theta(x) - d_{KL}(q_\varphi(z|x)|\, p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)] - d_{KL}(q_\varphi(z|x)|p(z))$$

This is a lower bound on our desired objective: $\log p_\theta(x)$
The VAE objective!

$$d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(z|x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z) + \log\, p_\theta(x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z)] + \log\, p_\theta(x)$$

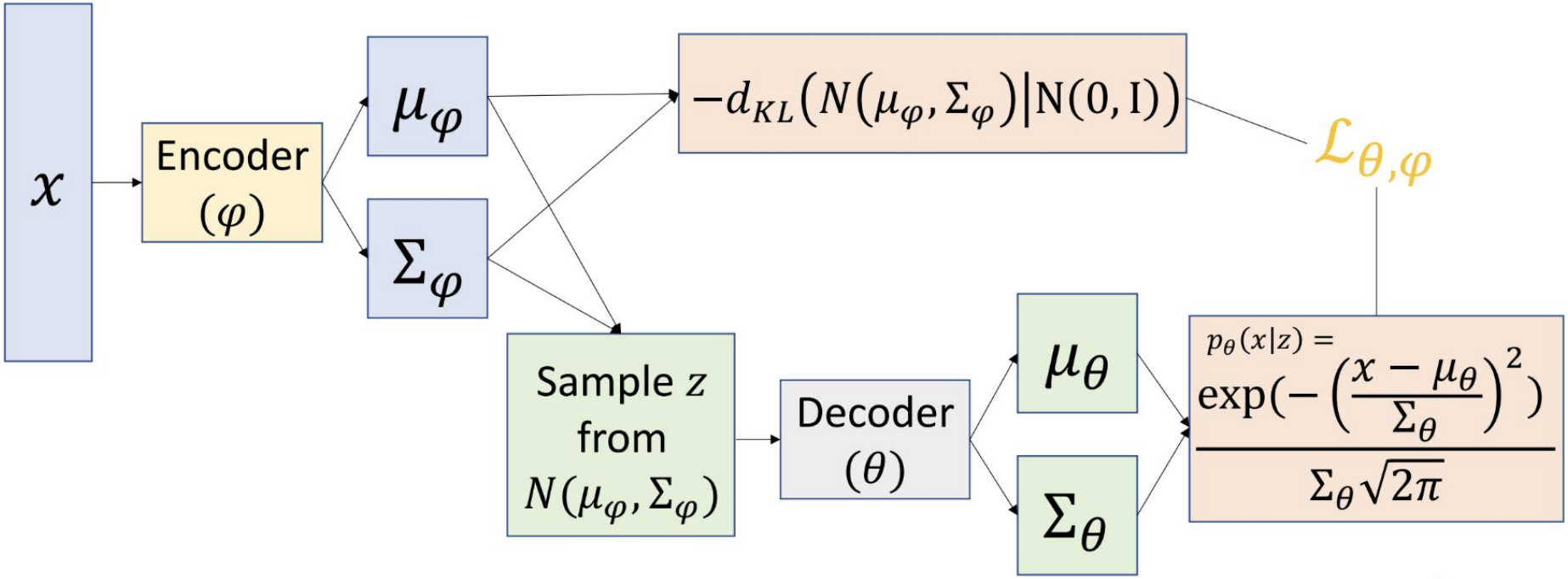
$\Rightarrow$

$$\log\, p_\theta(x) - d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \underbrace{\mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)] - d_{KL}(q_\varphi(z|x)|p(z))}$$

It's called the Evidence Lower Bound (ELBO)

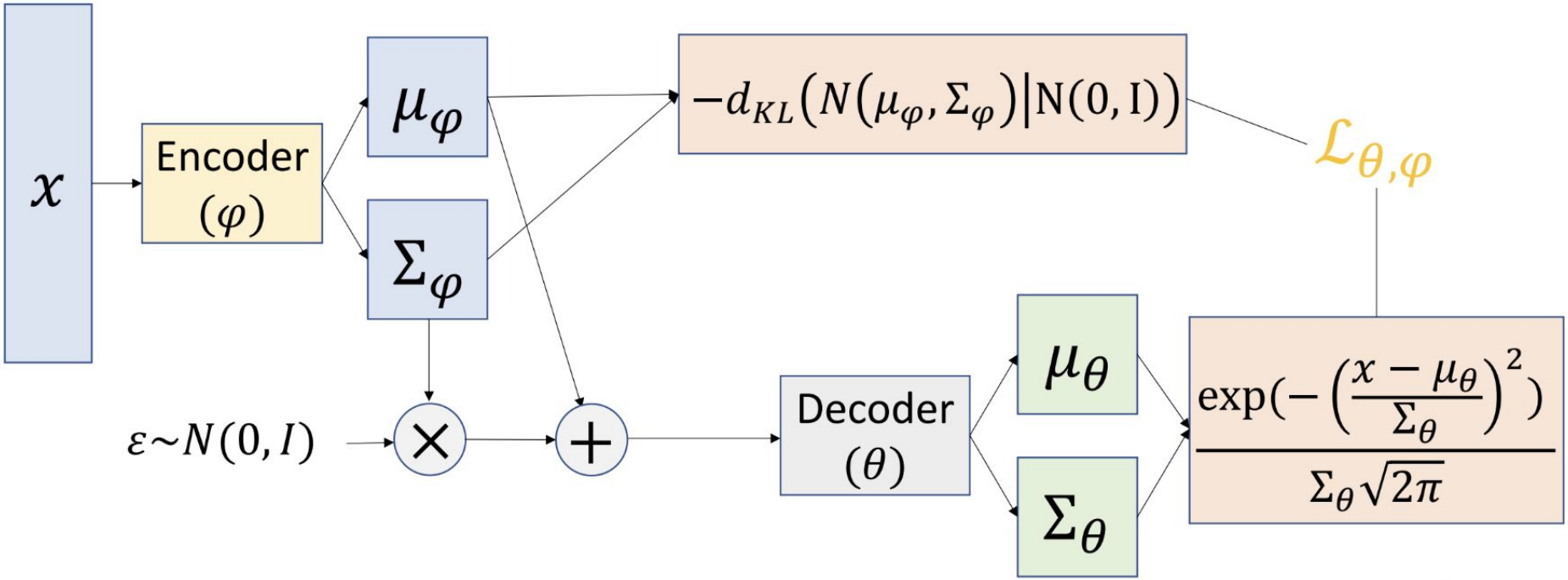
$$\mathcal{L}_{\theta,\varphi}(x) = \mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)] \; - \; d_{KL}(q_\varphi(z|x)|p(z))$$

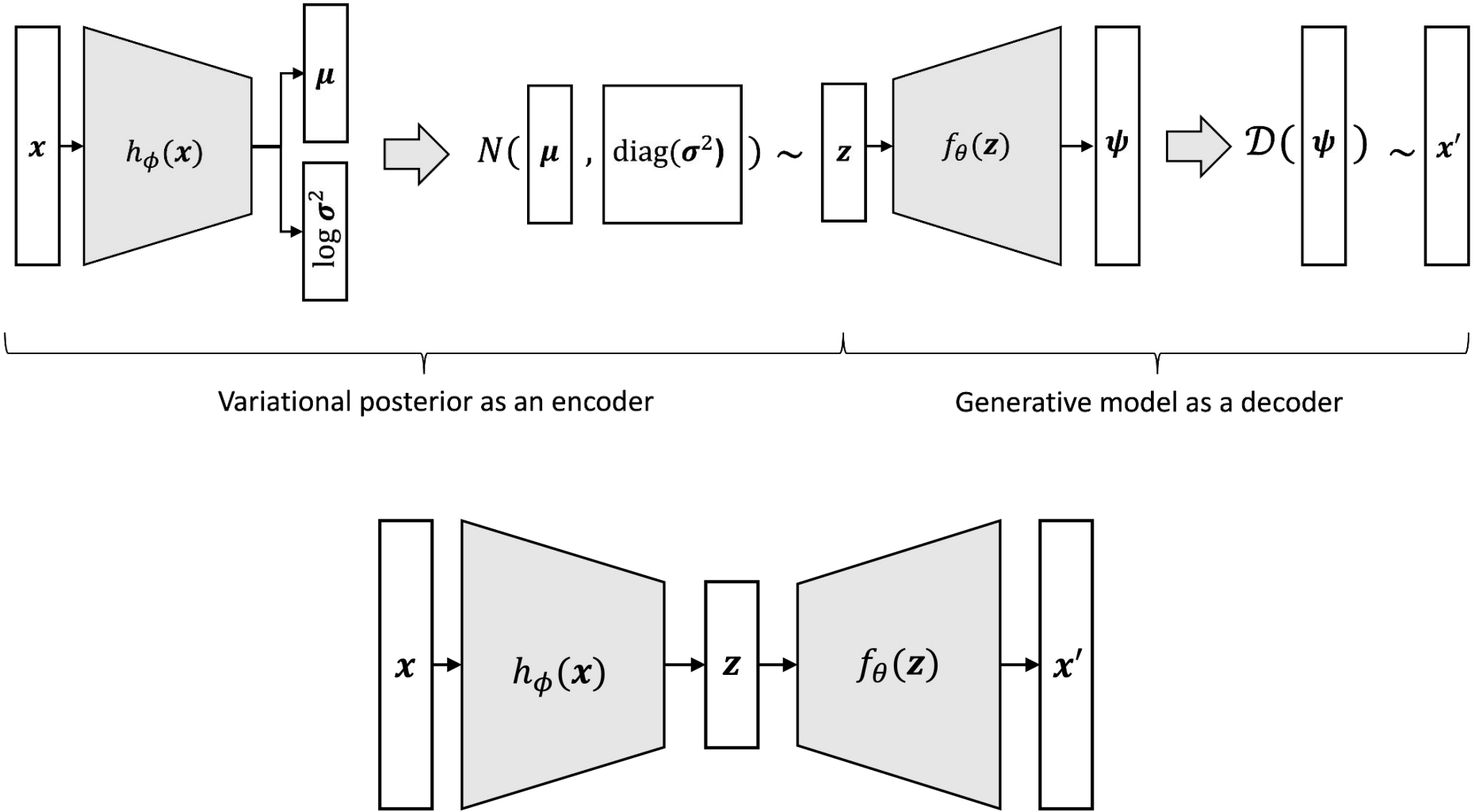
Training VAEs



Reparameterization trick

Training VAEs

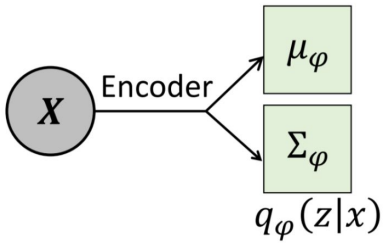


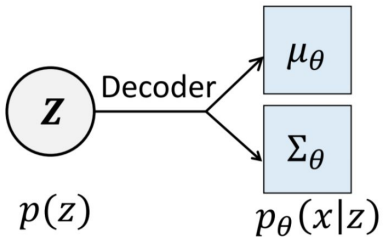

x
$h_\phi(x)$
μ
$\log \sigma^2$
$N(\ \mu\ ,\ \mathrm{diag}(\sigma^2)\ ) \sim z$
$f_\theta(z)$
ψ
$\mathcal{D}(\ \psi\ ) \sim x'$
Variational posterior as an encoder
Generative model as a decoder
x
$h_\phi(x)$
z
$f_\theta(z)$
x'

Optimizing the VAE objective

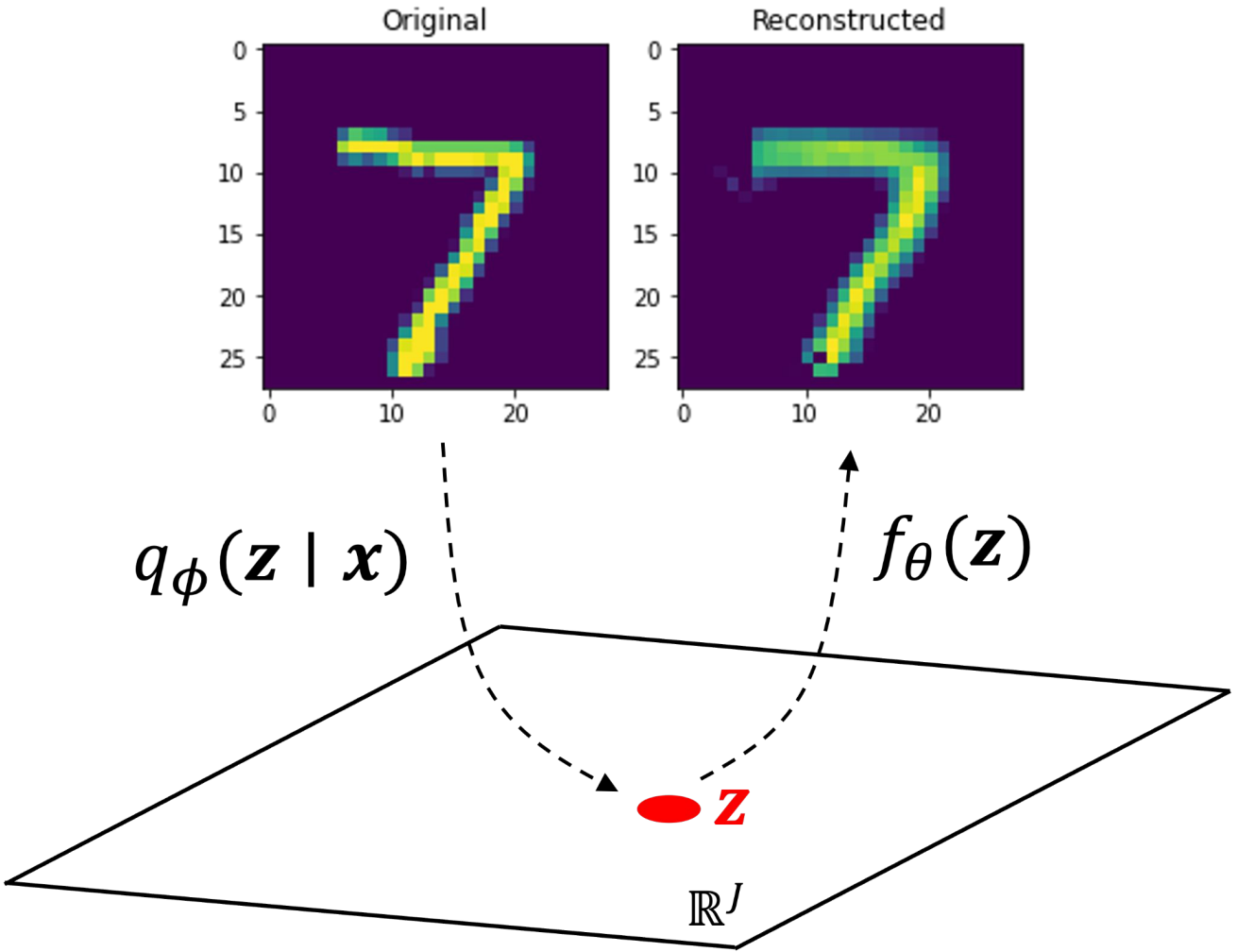
This term only depend on □

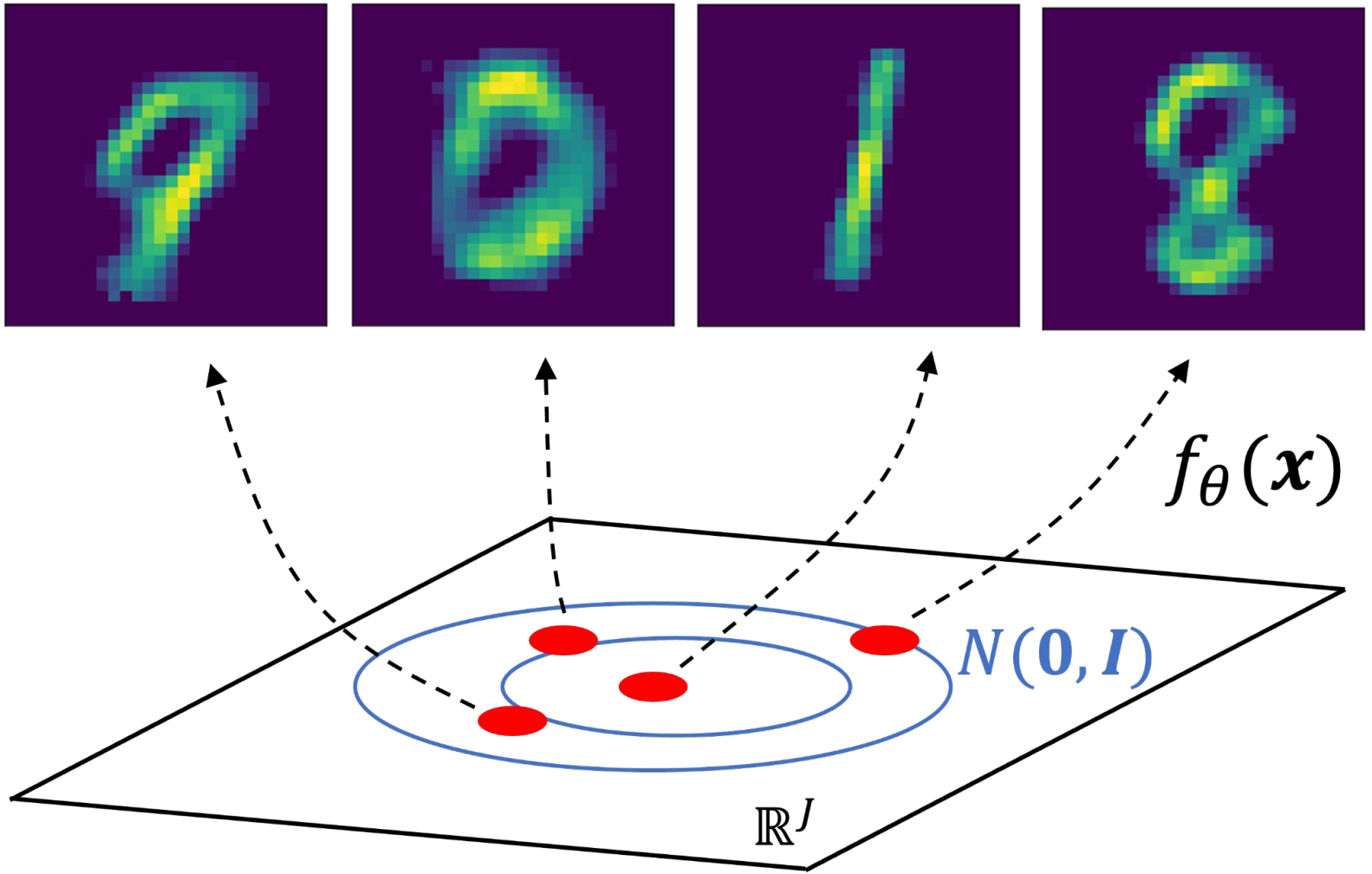
$$max_{\theta,\varphi}\, \mathcal{L}_{\theta,\varphi}(x) = \underbrace{\mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)]}_{\text{This term depends on both } \theta \text{ and } \varphi} - d_{KL}(q_\varphi(z|x)|p(z))$$

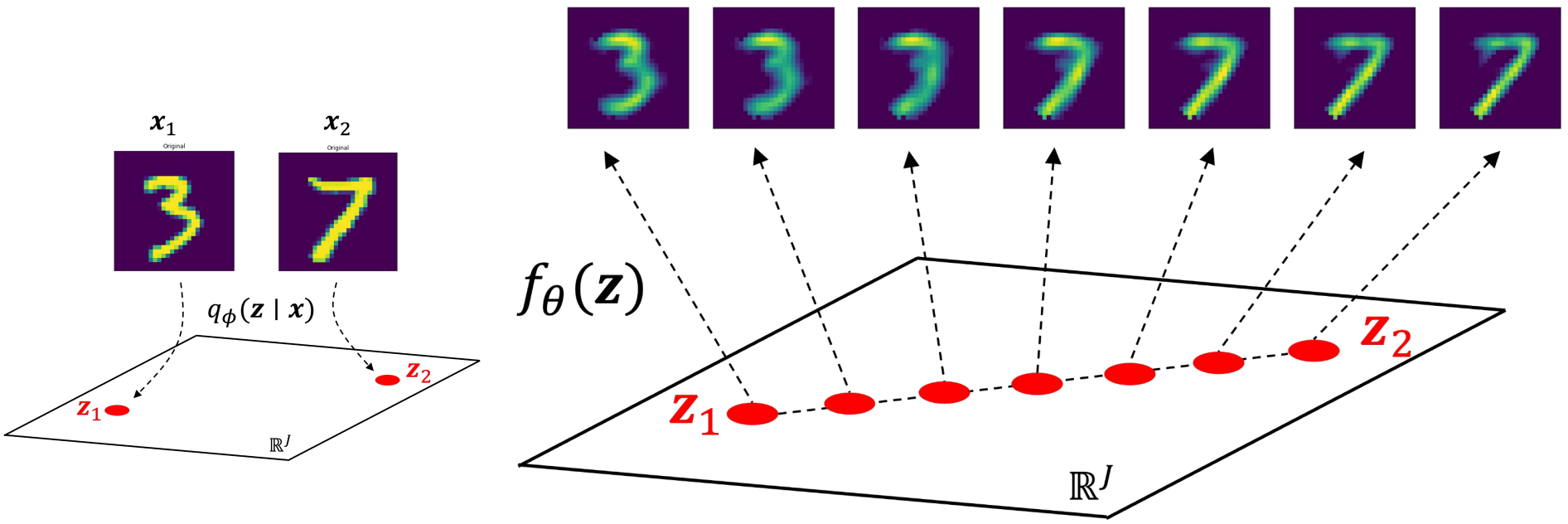




We will introduce how we get this in next lecture.

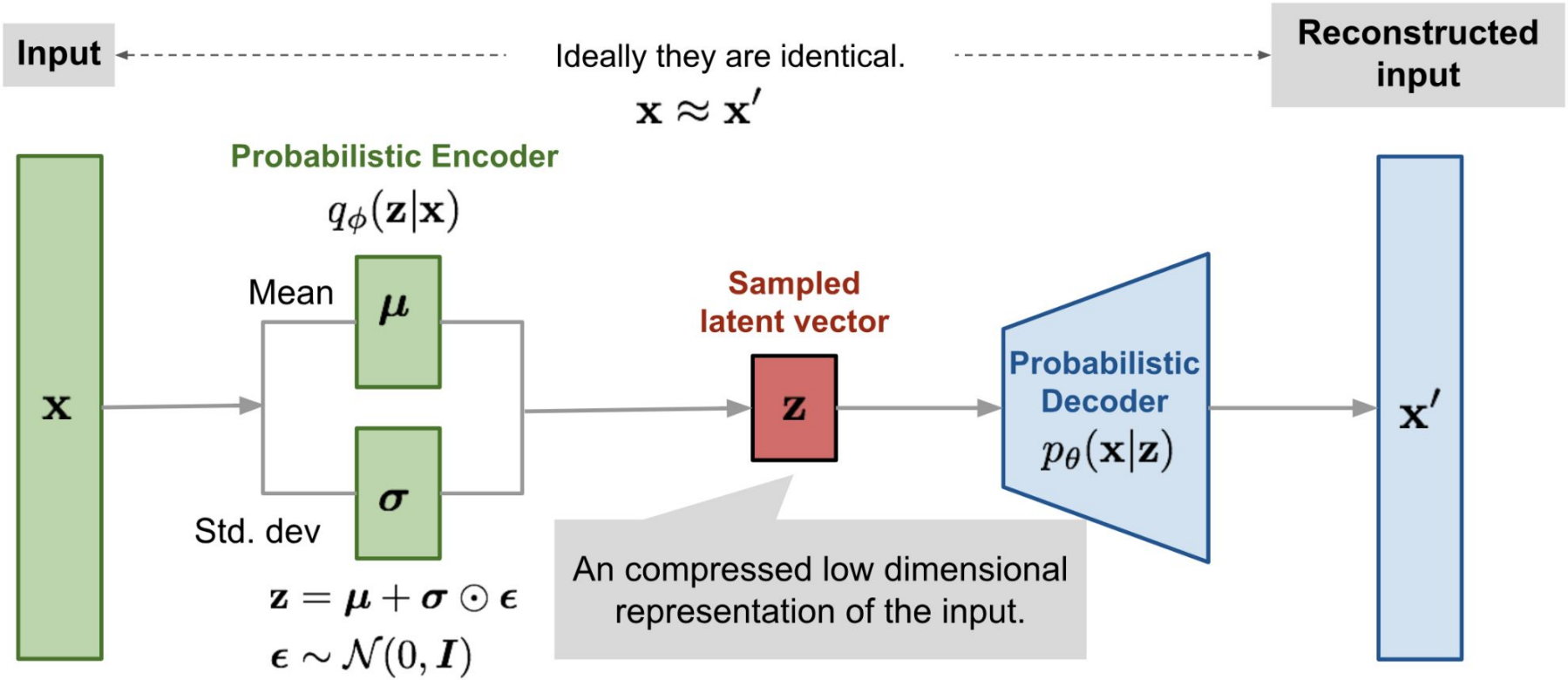

Original
Reconstructed
0
5
10
15
20
25
0
10
20
$q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$
$f_\theta(\boldsymbol{z})$
$\boldsymbol{z}$
$\mathbb{R}^J$


$f_\theta(\boldsymbol{x})$
$N(\mathbf{0}, \boldsymbol{I})$
$\mathbb{R}^J$


x_1
Original
x_2
Original
$q_\phi(z \mid x)$
z_1
z_2
$\mathbb{R}^J$
$f_\theta(z)$
z_1
z_2
$\mathbb{R}^J$

VAE Generations

The diagonal covariance constraint on the posterior forces the model to disentangle factors of variation in z.


Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
Mean
$\boldsymbol{\mu}$
$\boldsymbol{\sigma}$
Std. dev
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
$\mathbf{x}$
Sampled latent vector
$\mathbf{z}$
An compressed low dimensional representation of the input.
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$

We want 3 things from generative models

- *Density estimation*: Given a datapoint x, what is the probability assigned by the model, i.e. $p_\theta(x)$
- *Sampling*: How can we generate novel data from the model distribution, i.e. $x_{new} \sim p_\theta(x)$
- *Unsupervised representation learning*: How can we learn meaningful feature representations for a datapoint x?

We want 3 things from generative models

- *Density estimation*: Given a datapoint x, what is the probability assigned by the model, i.e. $p_\theta(x)$

Roughly! You can estimate the joint $p_\theta(x, z)$ or a lower bound on $p_\theta(x)$.

Optimizing the lower bound is perhaps why VAE images are often blurrier than the ones generated with autoregressive models.

We want 3 things from generative models

- *Density estimation*: Given a datapoint x, what is the probability assigned by the model, i.e. $p_\theta(x)$

- *Sampling*: How can we generate novel data from the model distribution, i.e. $x_{new} \sim p_\theta(x)$

We can sample new data fast with one forward pass through the decoder!

We want 3 things from generative models

- *Density estimation*: Given a datapoint x, what is the probability assigned by the model, i.e. $p_\theta(x)$
- *Sampling*: How can we generate novel data from the model distribution, i.e. $x_{new} \sim p_\theta(x)$
- *Unsupervised representation learning*: How can we learn meaningful feature representations for a datapoint x?

Yassss! The pro of VAEs is that you learn a latent code z which is produced by the encoder.

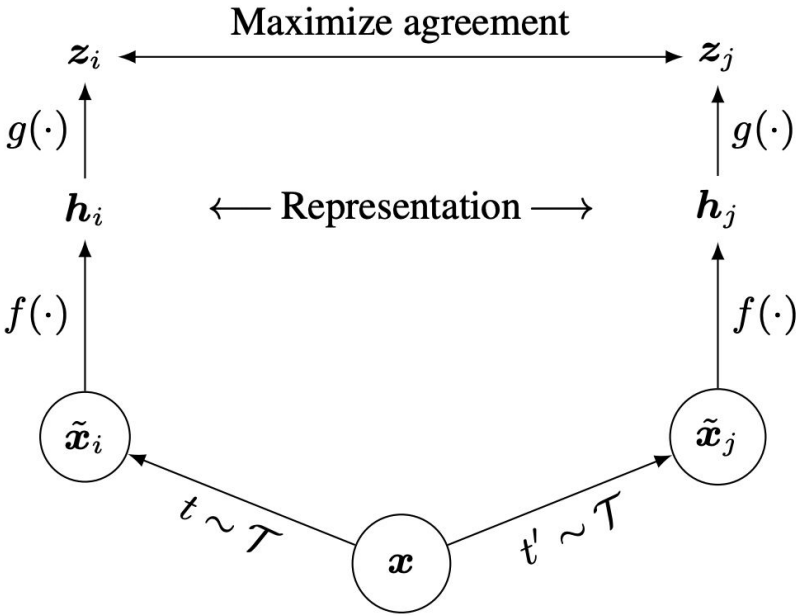
Contrastive Learning

Modern SSL | The Contrastive Paradigm

The Contrastive Paradigm:

Train a classifier where each instance in the training set is its own class.

AKA *instance discrimination*

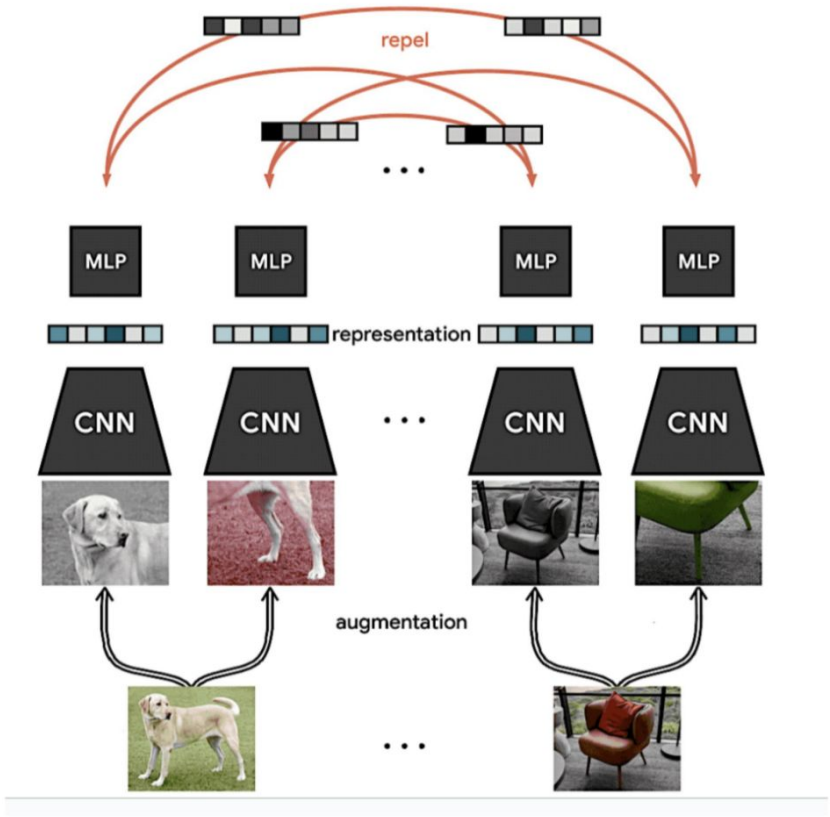


SimCLR: A Simple Framework for Contrastive Learning of Visual Representations

https://arxiv.org/pdf/2002.05709

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)}$$

Modern SSL | The Contrastive Paradigm

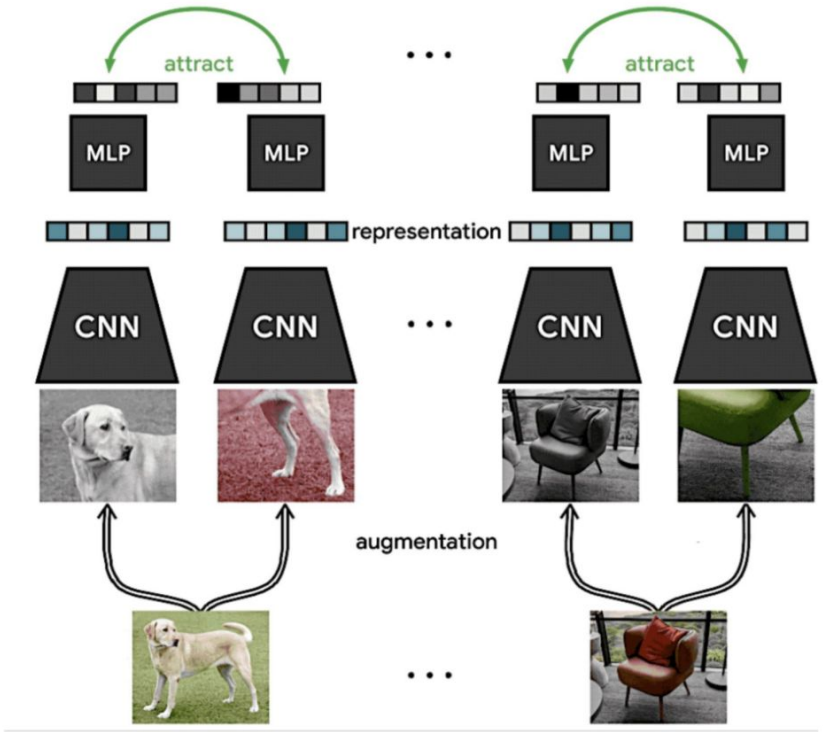


Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of f, g, $\mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**

end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Modern SSL | The Contrastive Paradigm

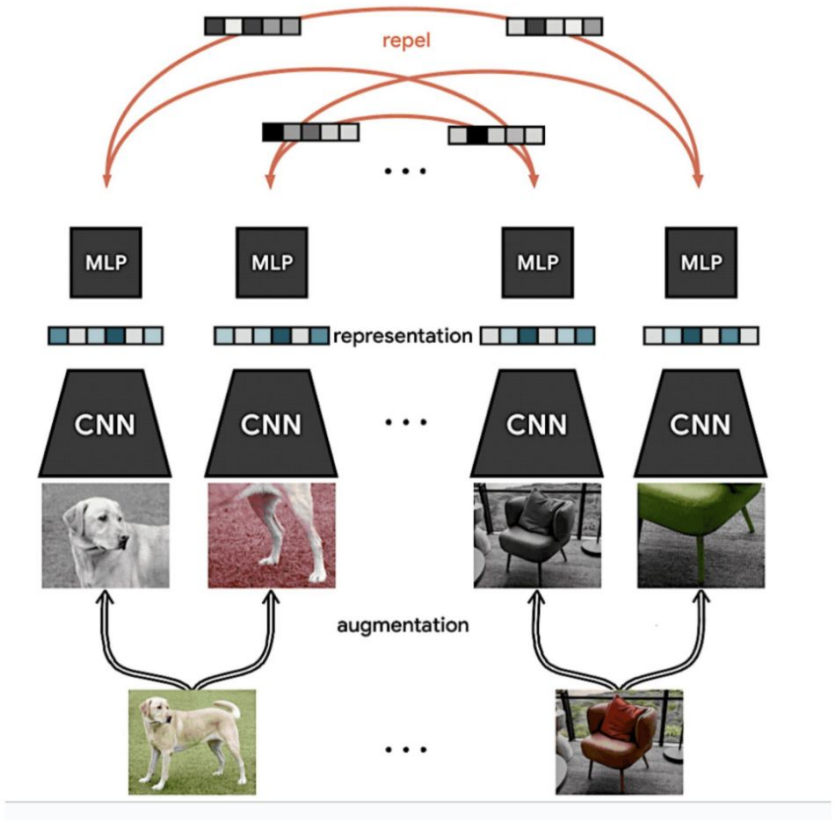


Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for

end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Modern SSL | The Contrastive Paradigm

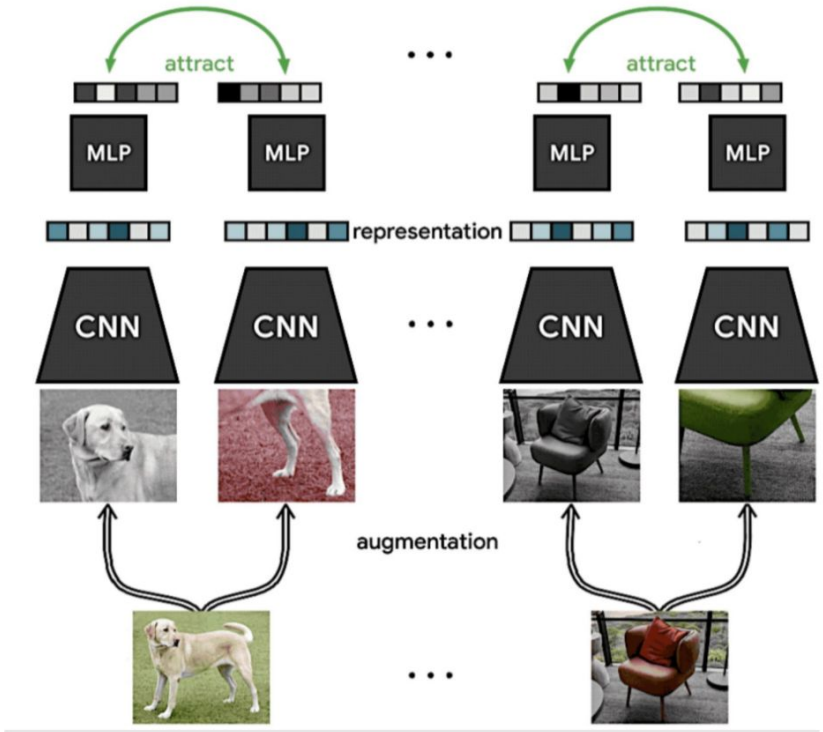


Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for

end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Modern SSL | The Contrastive Paradigm

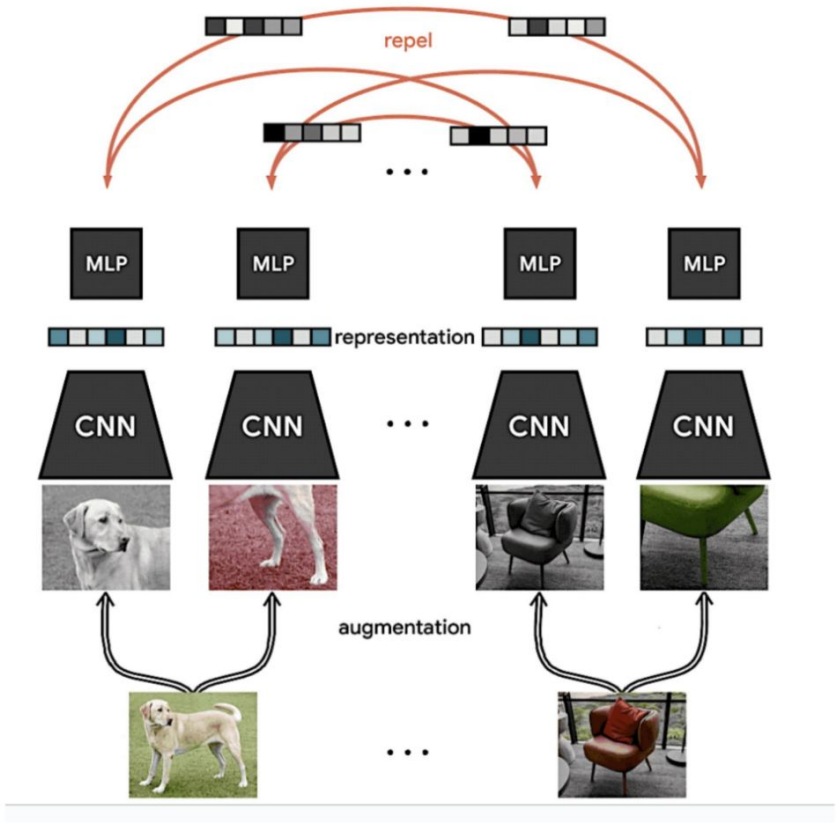


Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$

end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

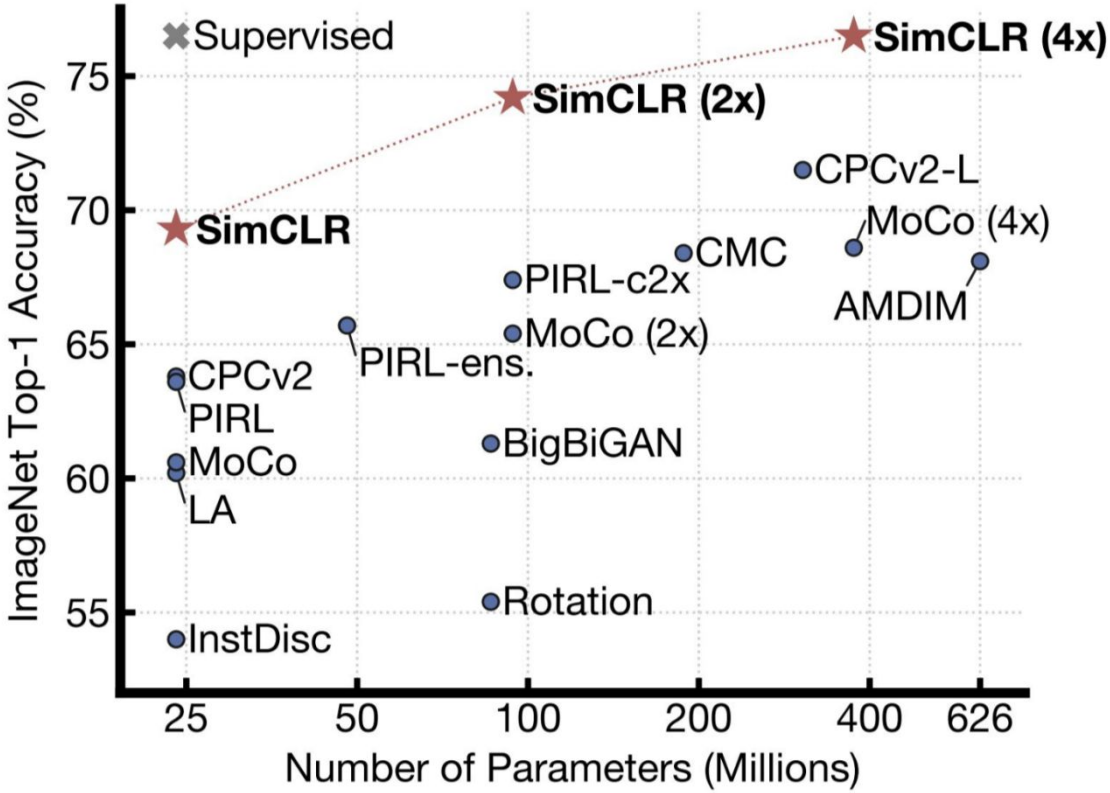
Modern SSL | The Contrastive Paradigm



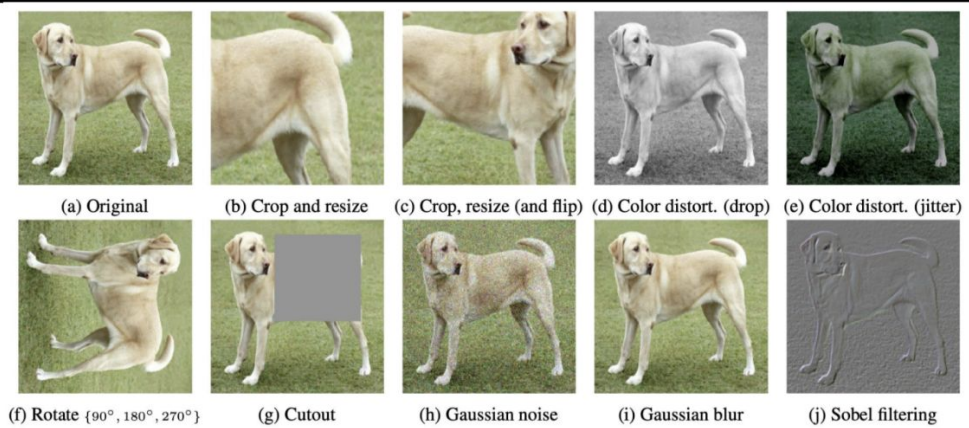
Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

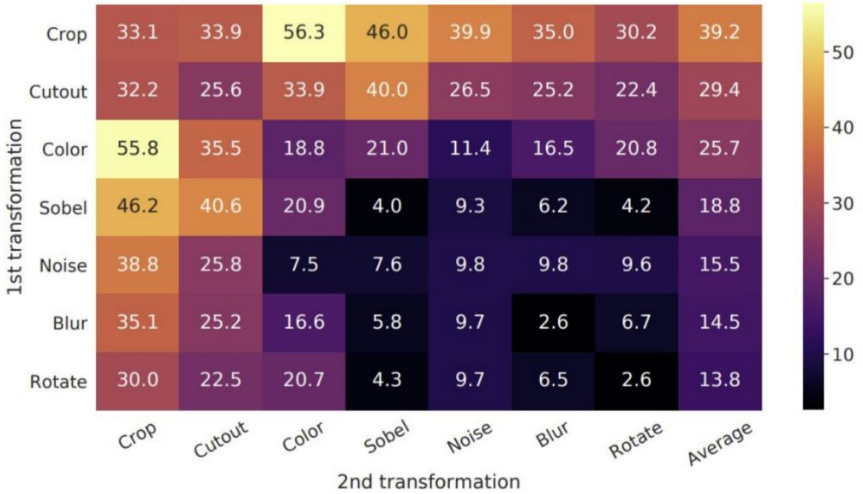
Modern SSL | The Contrastive Paradigm



Modern SSL | The Contrastive Paradigm

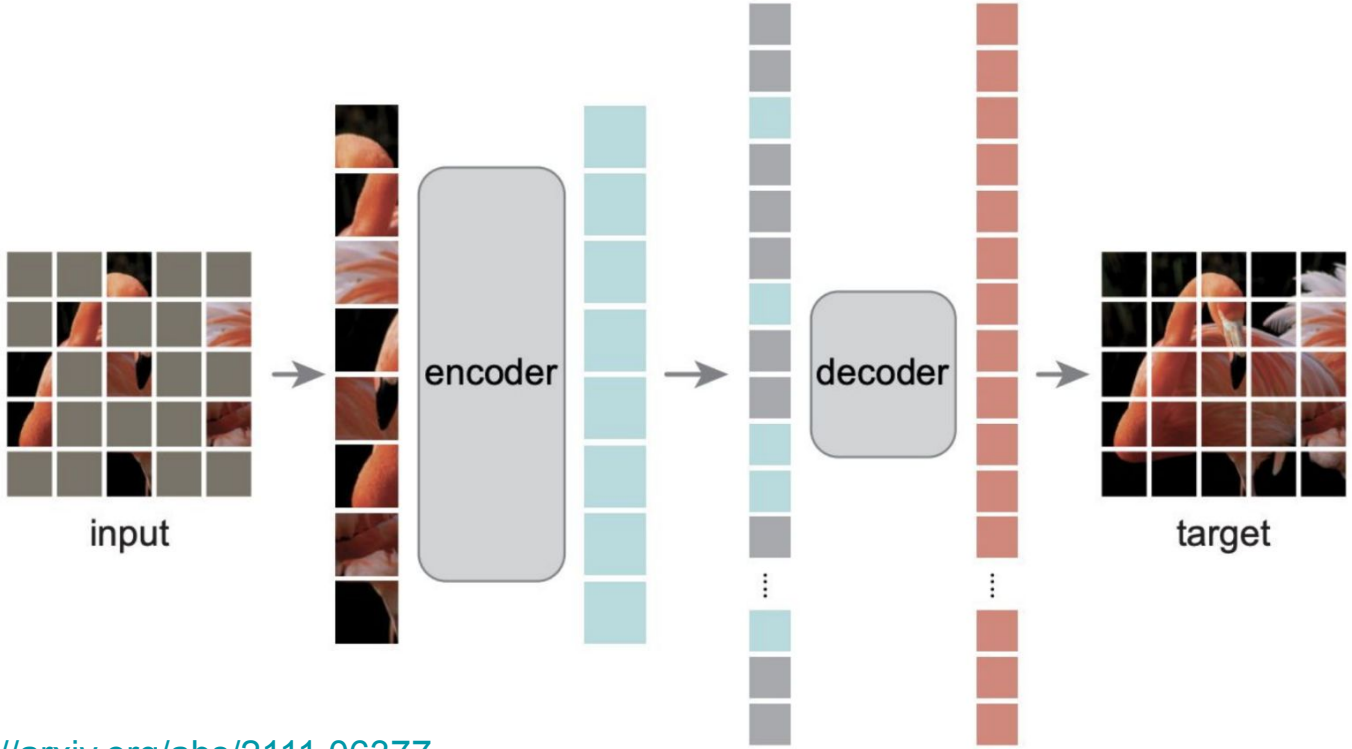


Composite Augmentation is the key!



Modern SSL | The Masking Paradigm | MAE

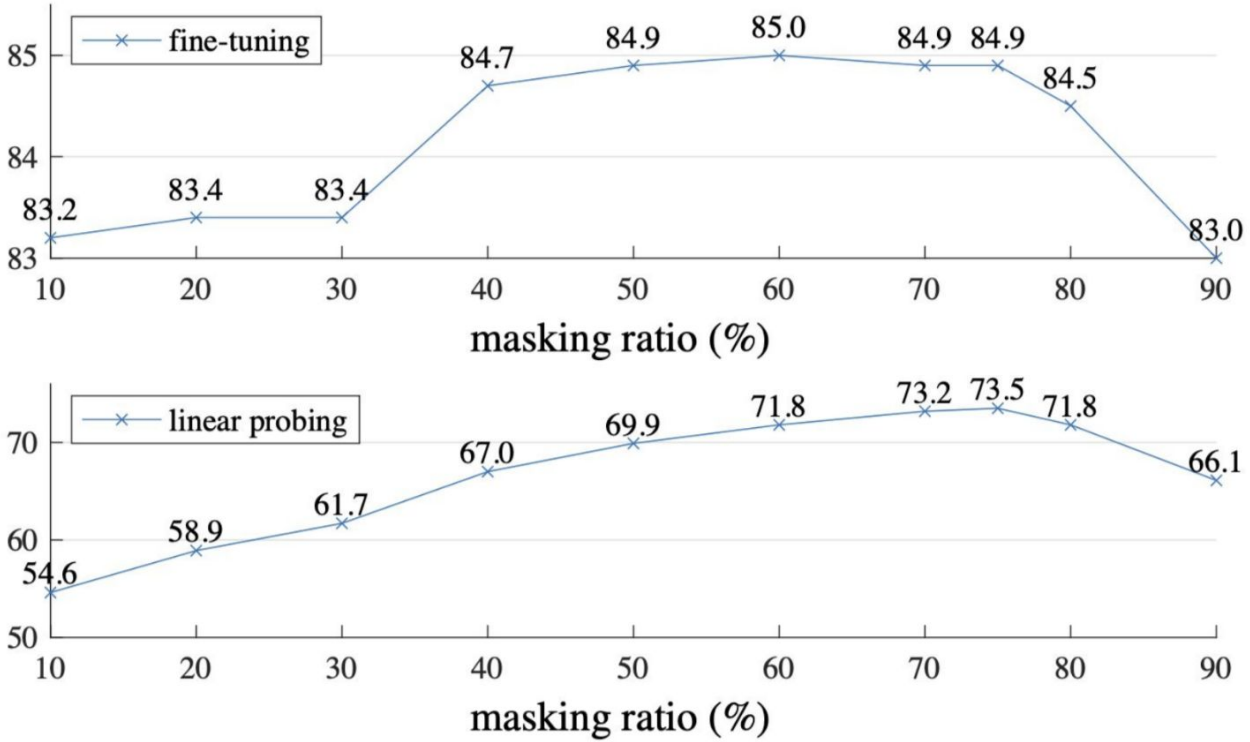
Masked Autoencoders Are Scalable Vision Learners
He et al. CVPR 2022



https://arxiv.org/abs/2111.06377

Modern SSL | The Masking Paradigm | MAE

Modern SSL | The Masking Paradigm | MAE



Intuition: Masking a large portion of the image to "create a challenging self supervisory task that requires holistic understanding beyond low-level image statistics."