CENG501 – Deep Learning

Week 12

Spring 2026

Sinan Kalkan

Dept. of Computer Engineering, METU

Previously on CENG501

What is a Vision-Language Model?

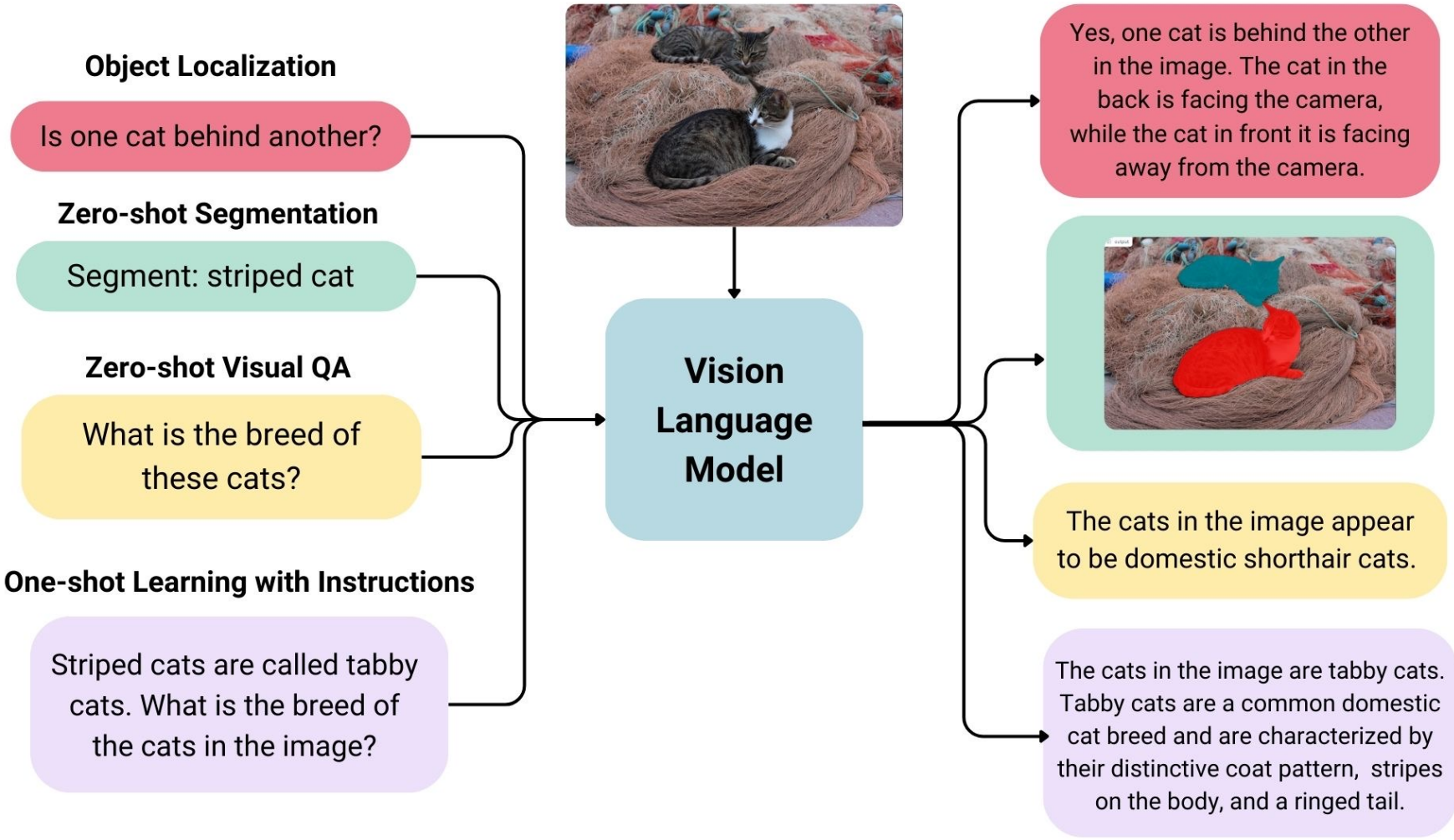


Fig: https://huggingface.co/blog/vlms

Previously on CENG501

Earlier Attempts:

VisualBERT: A Simple and Performant Baseline for Vision and Language

Liunian Harold Li†, Mark Yatskar*, Da Yin°, Cho-Jui Hsieh† & Kai-Wei Chang†
†University of California, Los Angeles
*Allen Institute for Artificial Intelligence
°Peking University

2019

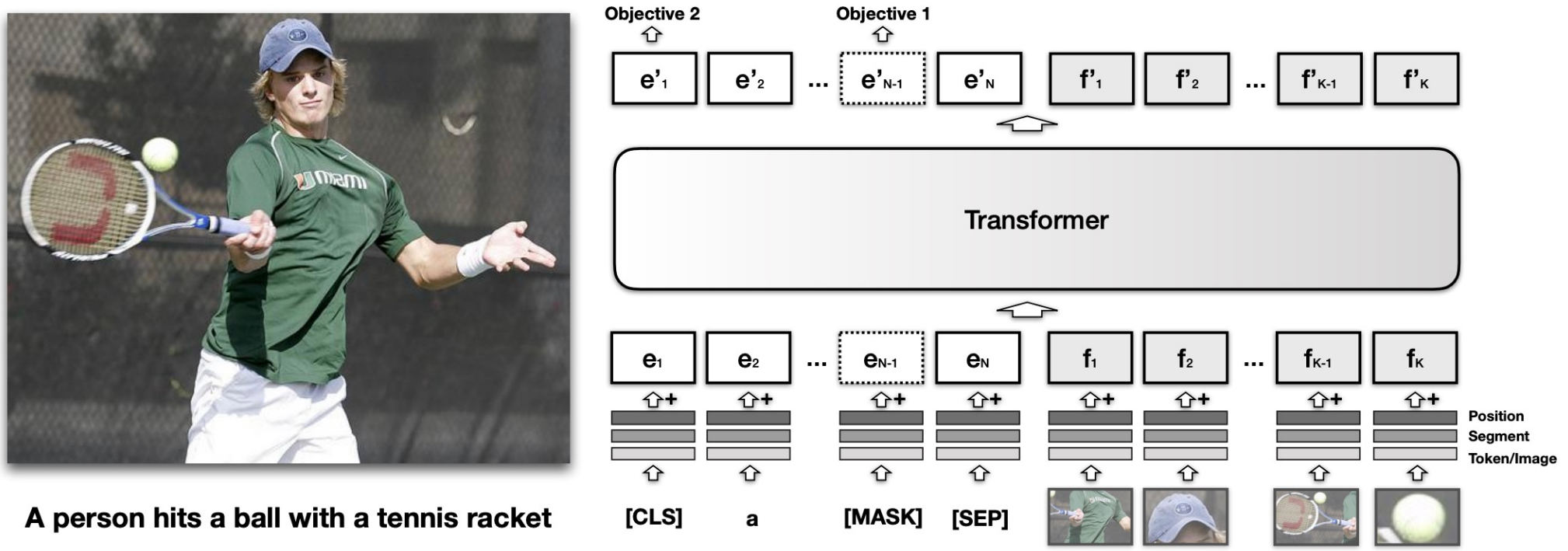
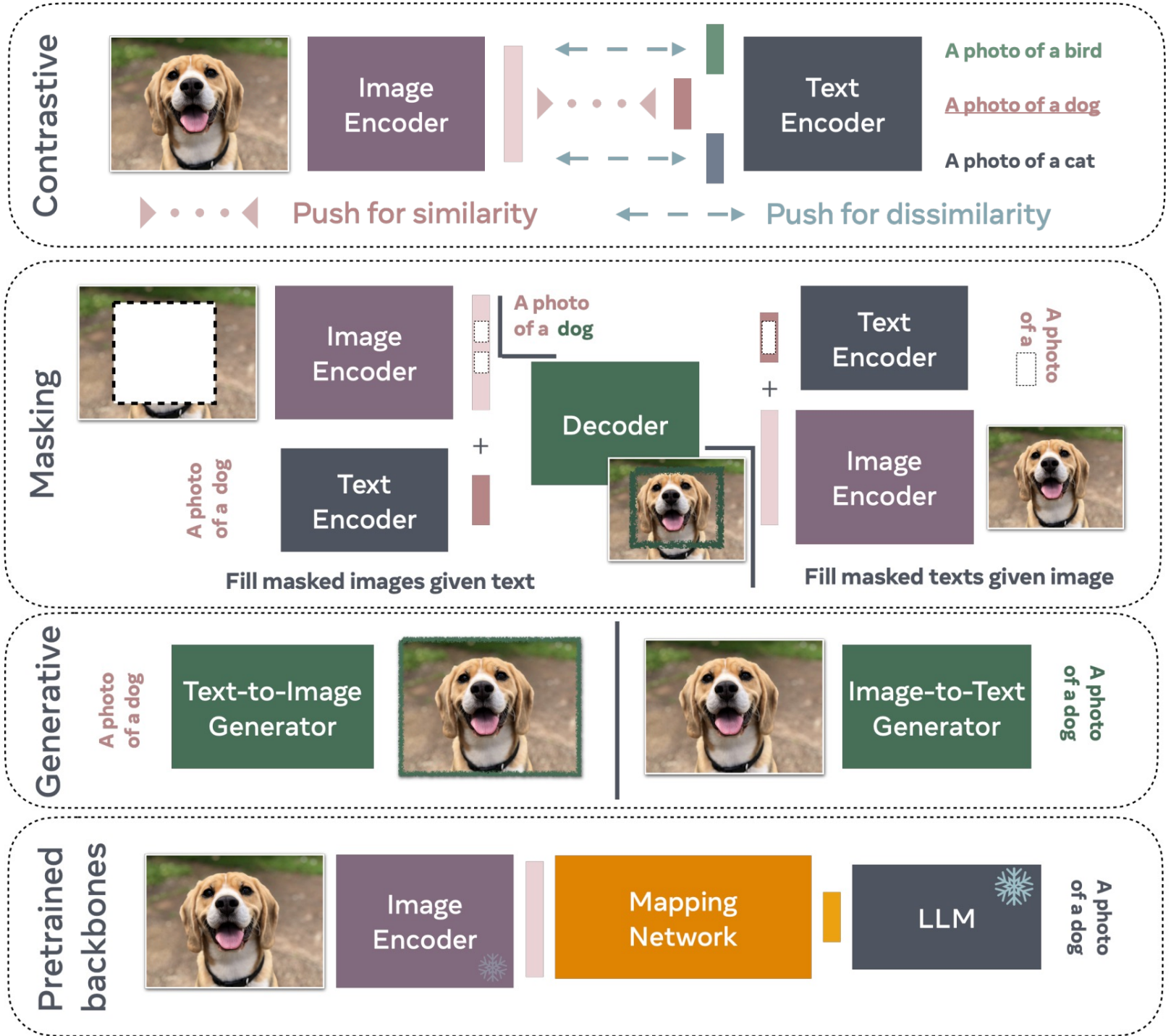


Figure 2: The architecture of VisualBERT. Image regions and language are combined with a Transformer to allow the self-attention to discover implicit alignments between language and vision. It is pre-trained with a masked language modeling (Objective 1), and sentence-image prediction task (Objective 2), on caption data and then fine-tuned for different tasks. See §3.3 for more details.

Previously on ENG501

"Modern" VLMs:

An Overview of the Approaches



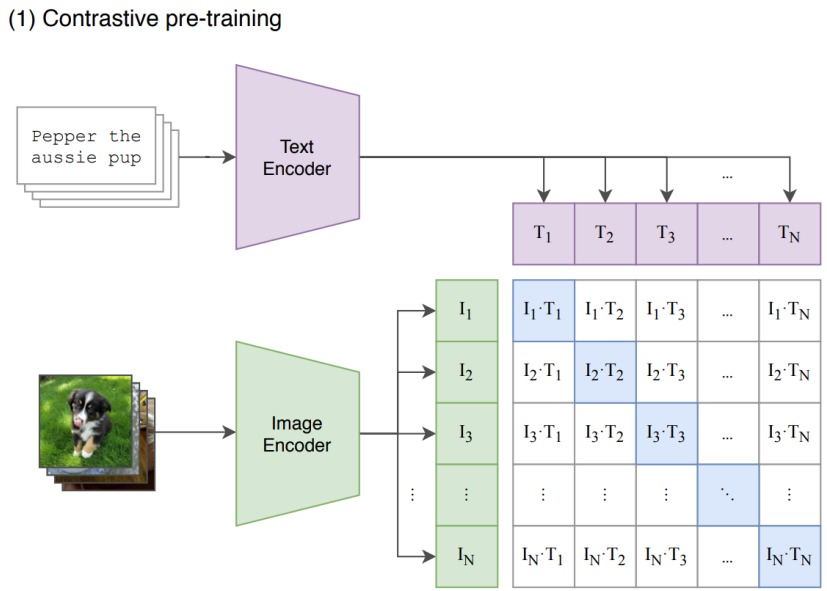
Bordes et al., "An Introduction to Vision-Language Modeling", 2024. https://arxiv.org/pdf/2405.17247

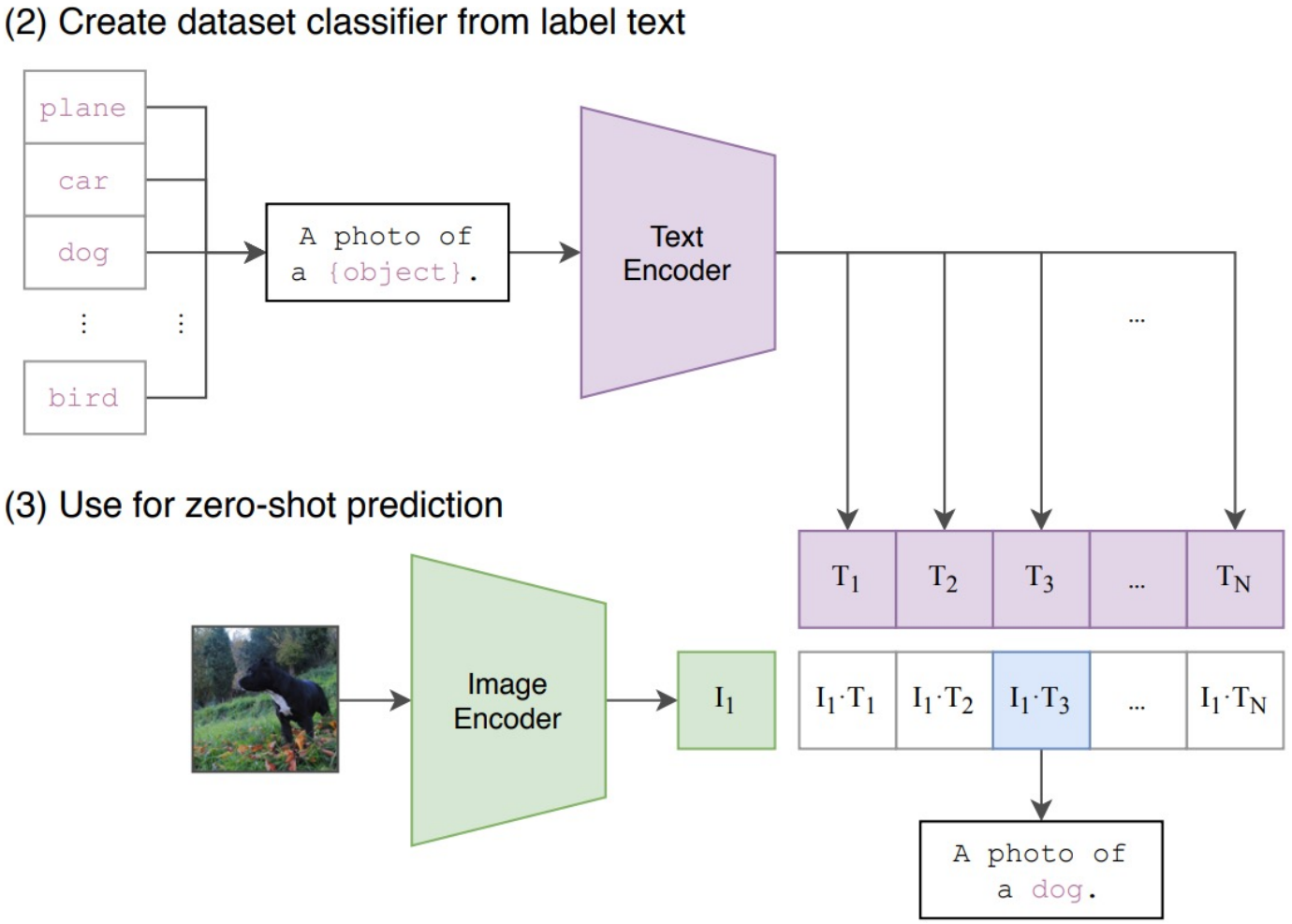
Previously on CENG501

Learning Transferable Visual Models From Natural Language Supervision

Alec Radford [*1] Jong Wook Kim [*1] Chris Hallacy [1] Aditya Ramesh [1] Gabriel Goh [1] Sandhini Agarwal [1]
Girish Sastry [1] Amanda Askell [1] Pamela Mishkin [1] Jack Clark [1] Gretchen Krueger [1] Ilya Sutskever [1]

2021





Previously on CENG501

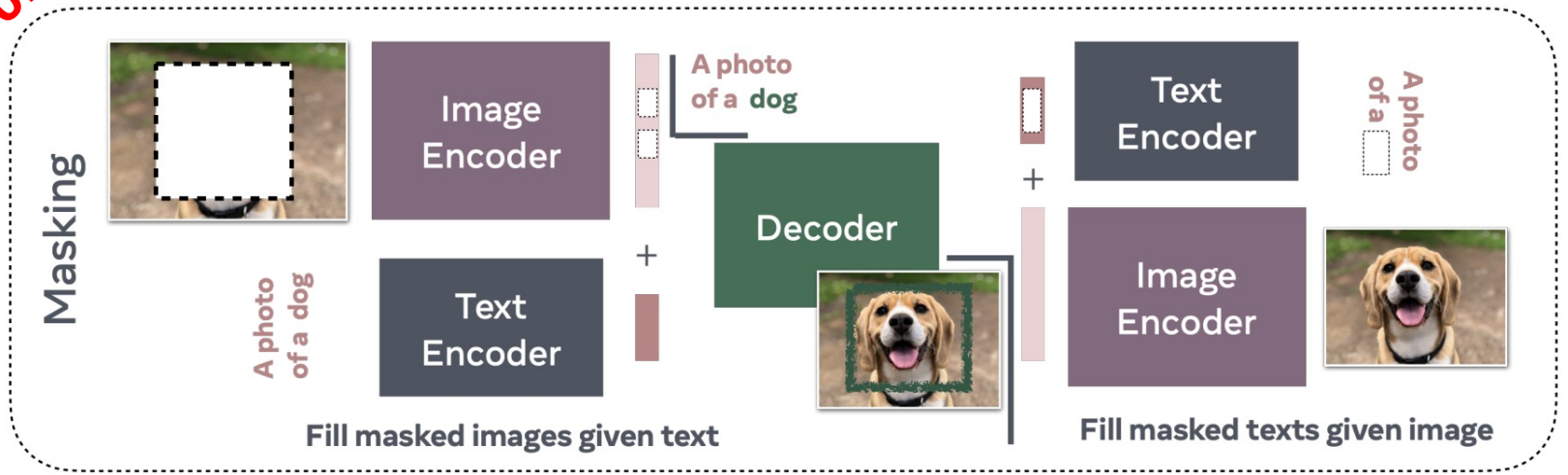


Fig: Bordes et al., “An Introduction to Vision-Language Modeling”, 2024.

Masking Approaches

Previously on ENG501

FLAVA

GC: Global Contrastive Loss (same as CLIP)

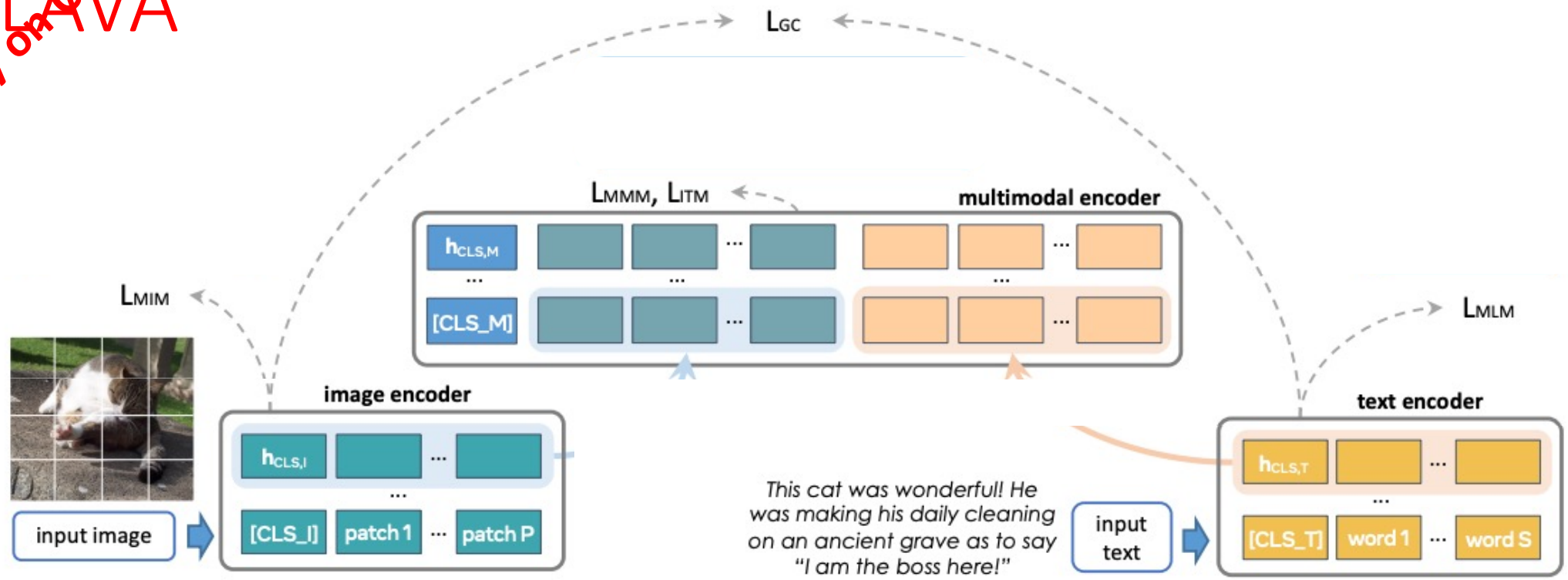
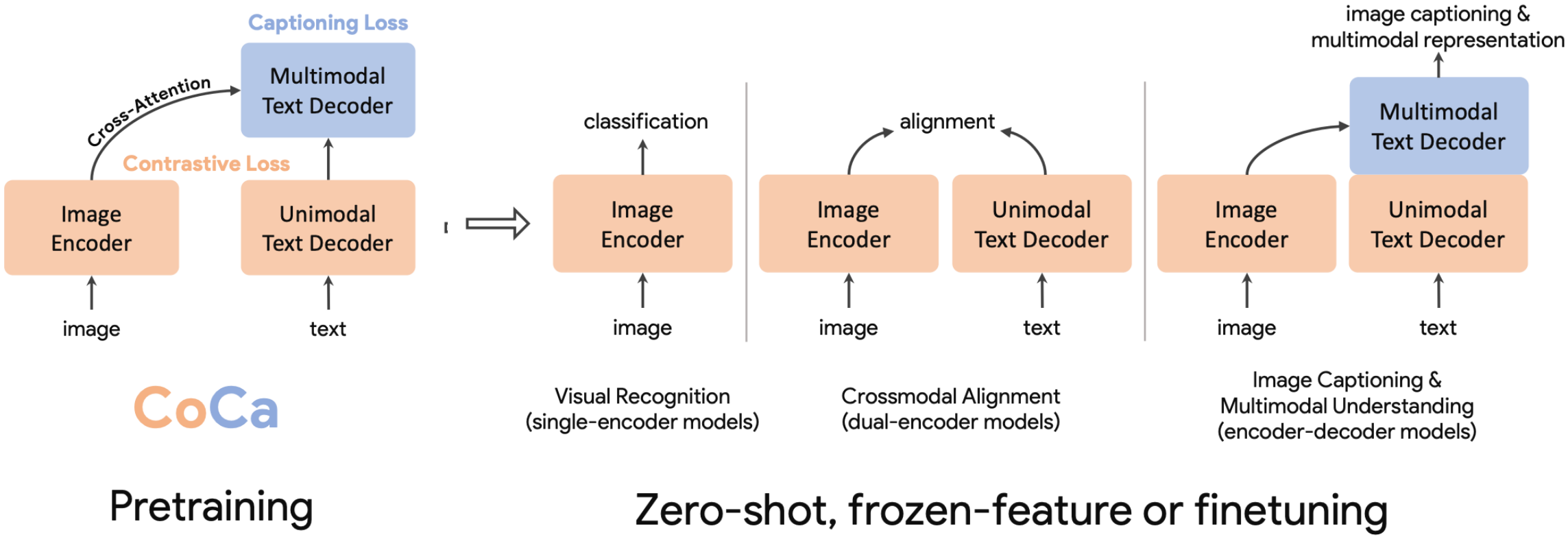


Figure 2. **An overview of our FLAVA model**, with an image encoder transformer to capture unimodal image representations, a text encoder transformer to process unimodal text information, and a multimodal encoder transformer that takes as input the encoded unimodal image and text and integrates their representations for multimodal reasoning. **During pretraining**, masked image modeling (MIM) and mask language modeling (MLM) losses are applied onto the image and text encoders over a single image or a text piece, respectively, while contrastive, masked multimodal modeling (MMM), and image-text matching (ITM) loss are used over paired image-text data. **For downstream tasks**, classification heads are applied on the outputs from the image, text, and multimodal encoders respectively for visual recognition, language understanding, and multimodal reasoning tasks.

CoCa

Contrastive Captioner (CoCa),
Yu et al., 2022.



Previously on CANG501

Chameleon

Chameleon: Mixed-Modal Early-Fusion Foundation Models, Meta, 2024.

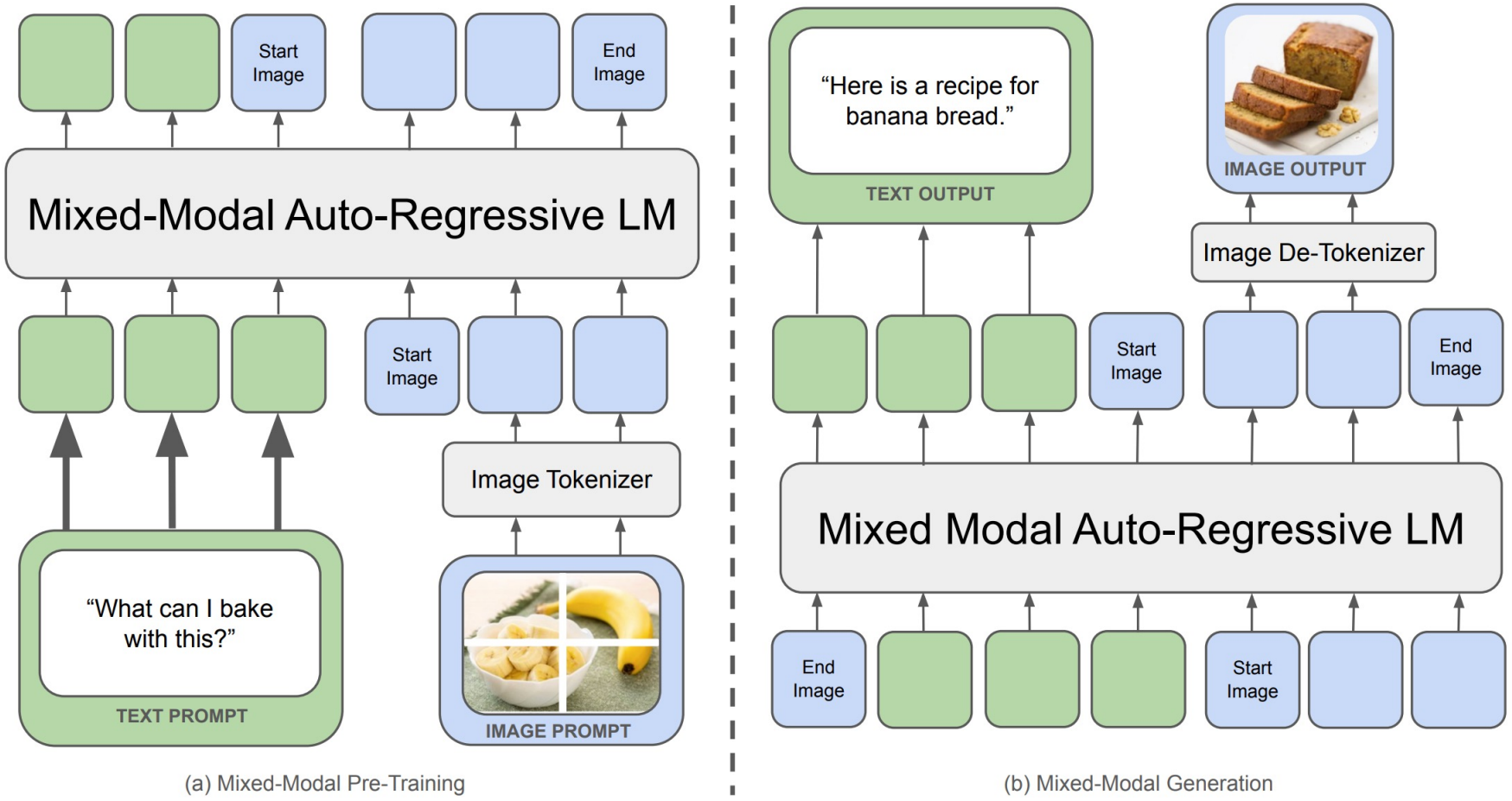


Figure 1 **Chameleon** represents all modalities — images, text, and code, as discrete tokens and uses a uniform transformer-based architecture that is trained from scratch in an end-to-end fashion on ∼10T tokens of interleaved mixed-modal data. As a result, **Chameleon** can both reason over, as well as generate, arbitrary mixed-modal documents. Text tokens are represented in green and image tokens are represented in blue.

Previously on CENG501

Fig: Bordes et al., “An Introduction to Vision-Language Modeling”, 2024.

Approaches Using Pre-trained Backbones

Previously on ECENG501

Frozen

Multimodal Few-Shot Learning with Frozen Language Models, 2021

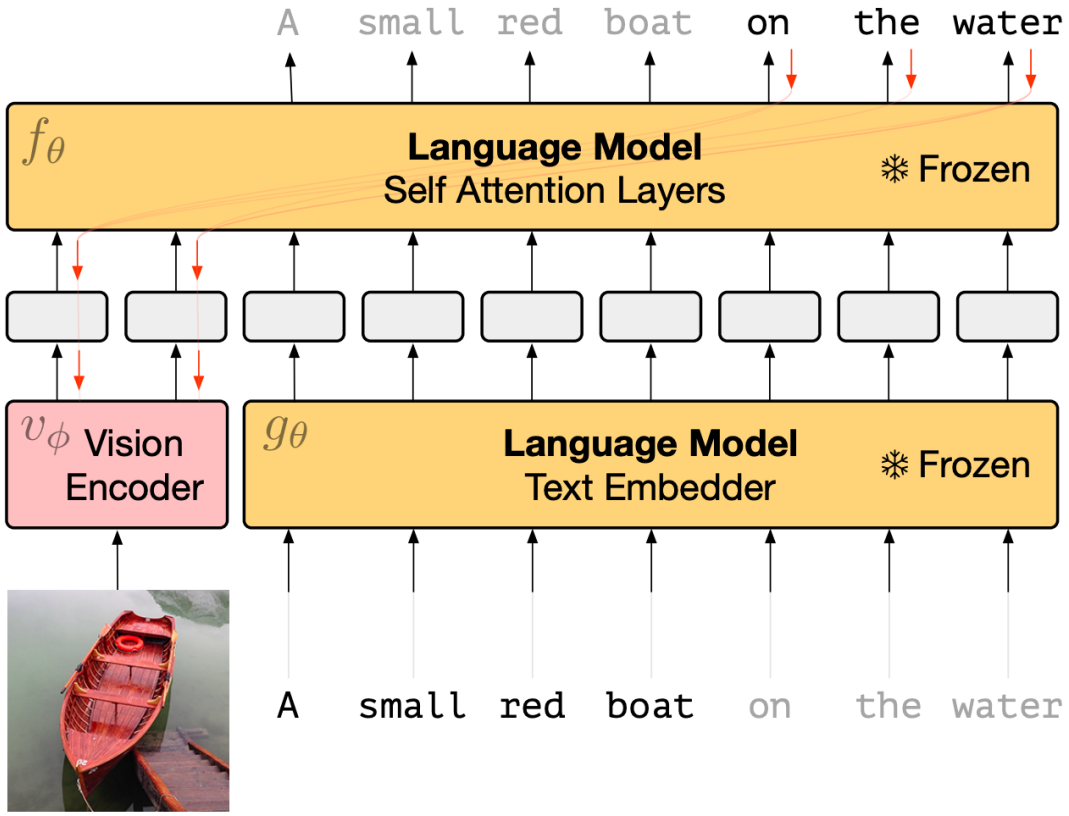


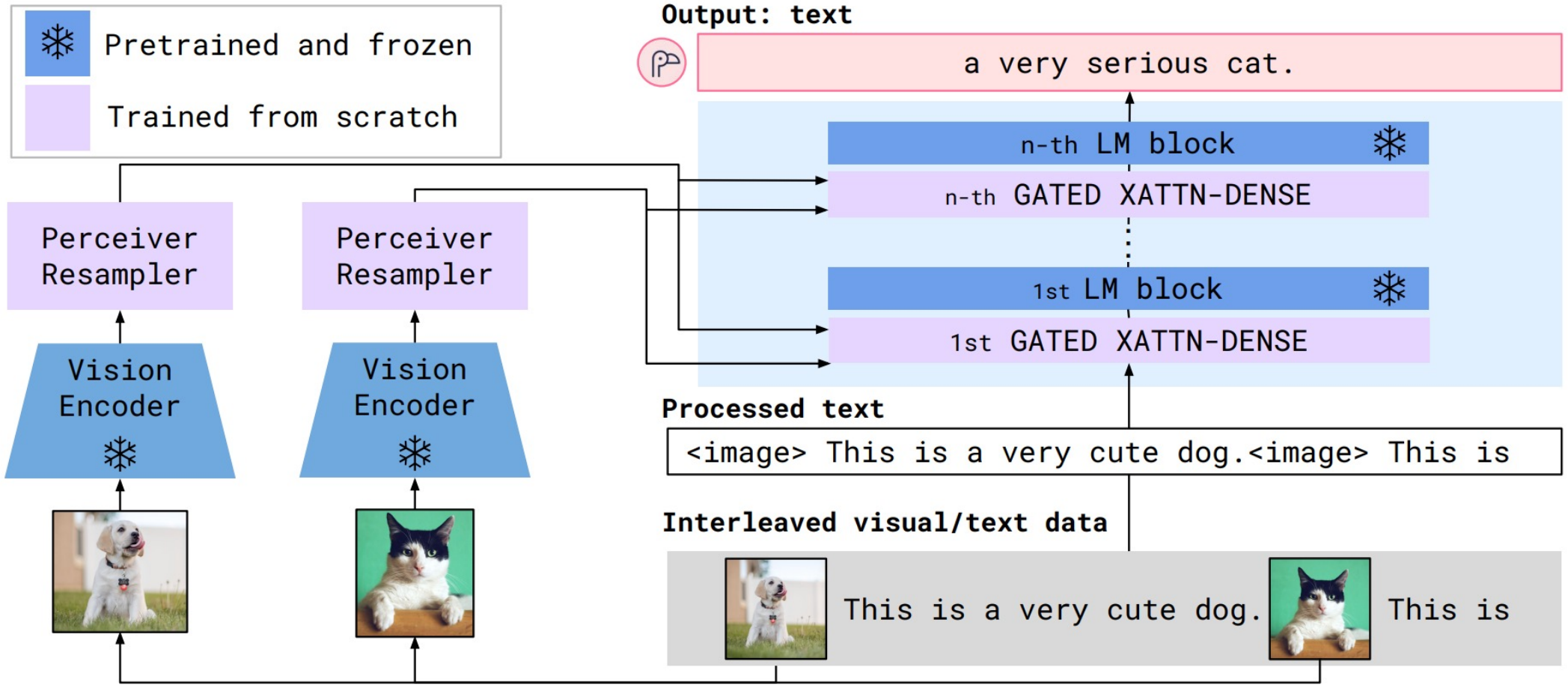
Figure 2: Gradients through a frozen language model's self attention layers are used to train the vision encoder.

Previously on CENG501

Flamingo

Flamingo: a Visual Language Model for Few-Shot Learning, Deepmind, 2022

Vision Encoder: Normalizer-Free ResNet (NFNet)
Perceiver Sampler: Fixed # of queries attend to variable length of visual tokens. (input images can have different resolutions)
LLM: Chinchilla



Previously on CENG501

BLIP

BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation, Salesforce, 2022.

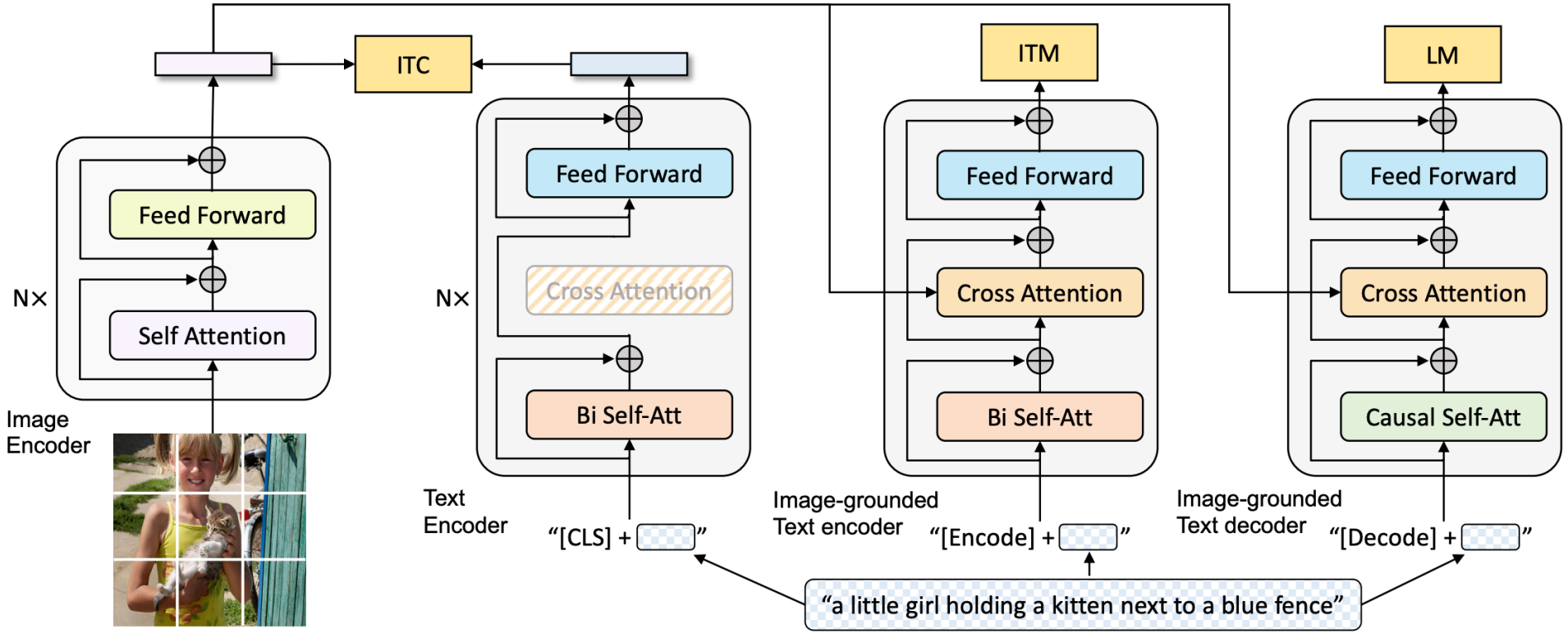


Figure 2. Pre-training model architecture and objectives of BLIP (same parameters have the same color). We propose multimodal mixture of encoder-decoder, a unified vision-language model which can operate in one of the three functionalities: (1) Unimodal encoder is trained with an image-text contrastive (ITC) loss to align the vision and language representations. (2) Image-grounded text encoder uses additional cross-attention layers to model vision-language interactions, and is trained with a image-text matching (ITM) loss to distinguish between positive and negative image-text pairs. (3) Image-grounded text decoder replaces the bi-directional self-attention layers with causal self-attention layers, and shares the same cross-attention layers and feed forward networks as the encoder. The decoder is trained with a language modeling (LM) loss to generate captions given images.

Previously on ENG501

BLIP-2

BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models, Salesforce, 2023.

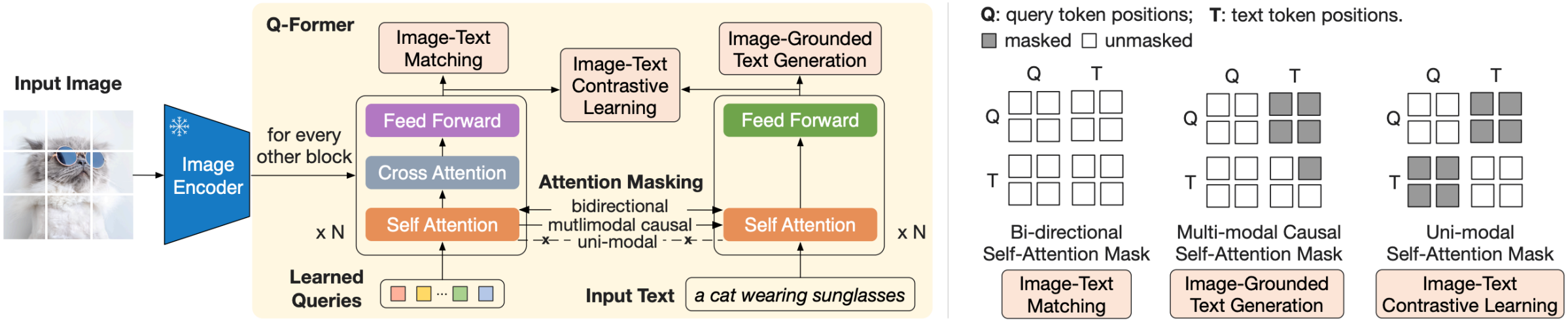


Figure 2. (**Left**) Model architecture of Q-Former and BLIP-2's first-stage vision-language representation learning objectives. We jointly optimize three objectives which enforce the queries (a set of learnable embeddings) to extract visual representation most relevant to the text. (**Right**) The self-attention masking strategy for each objective to control query-text interaction.

Previously on ENG501

BLIP-2

BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models, Salesforce, 2023.

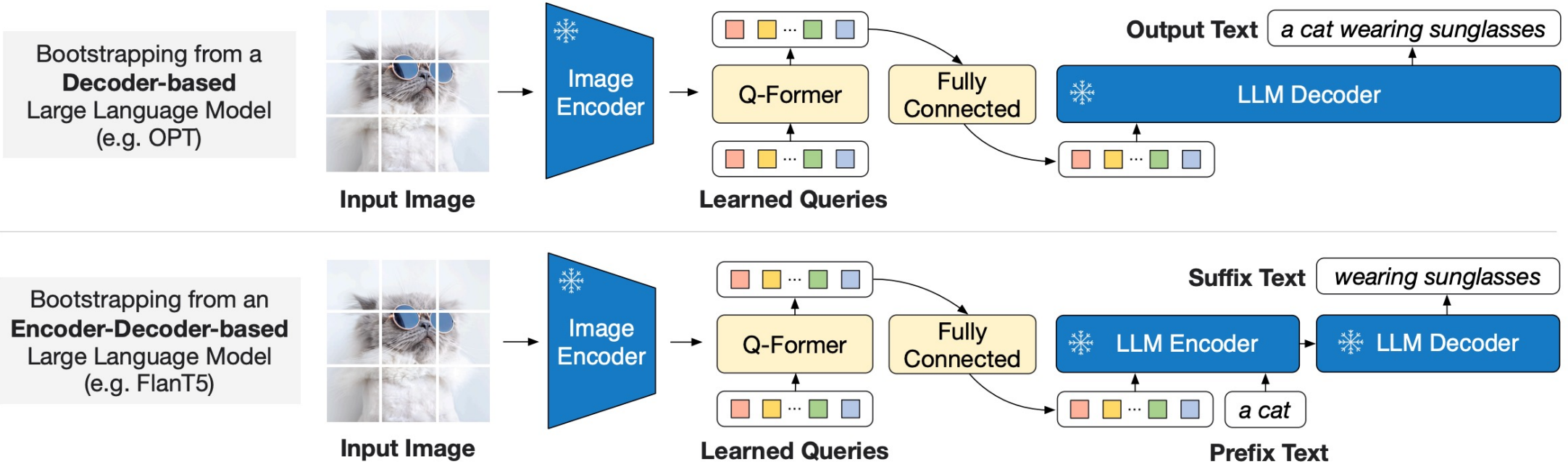


Figure 3. BLIP-2's second-stage vision-to-language generative pre-training, which bootstraps from frozen large language models (LLMs). (**Top**) Bootstrapping a decoder-based LLM (e.g. OPT). (**Bottom**) Bootstrapping an encoder-decoder-based LLM (e.g. FlanT5). The fully-connected layer adapts from the output dimension of the Q-Former to the input dimension of the chosen LLM.

Today

- (Deep) Generative Models
  - Autoregressive models
  - Variational AEs
  - Flow Models
  - Generative Adversarial Networks
  - Energy-based Models
  - Diffusion Models

CENG796 DEEP GENERATIVE MODELS

Course Code:	5710796
METU Credit (Theoretical-Laboratory hours/week):	3(3-0)
ECTS Credit:	8.0
Department:	Computer Engineering
Language of Instruction:	English
Level of Study:	Graduate
Course Coordinator:	Assoc.Prof.Dr. RAMAZAN GÖKBERK CİNBİŞ
Offered Semester:	Fall Semesters.

Course Objectives

At the end of the course, the students will be expected to:

- Comprehend a variety of deep generative models.
- Apply deep generative models to several problems.
- Know the open issues in learning deep generative models, and have a grasp of the current research directions.

Course Content

Deep generative modeling with Autoregressive models; Energy-based models; Adversarial models; Variational models.

Administrative Notes

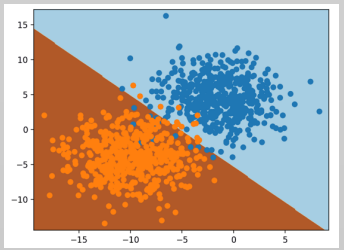
- Project next steps:
  - Milestones:
    1. Milestone (April 10, midnight):
       - Read & understand the paper
       - Download the datasets
       - Prepare the Readme file excluding the results & conclusion
    2. Milestone (May 4, midnight)
       - The results of the first experiment
    3. Milestone (June 1, midnight)
       - Final report (Readme file)
       - Repo with all code & trained models

Fırından Sıcak Sıcak

Overview & Problem Formulation

Supervised

Instance	Label
	happy
	unhappy
	unhappy
	happy
	unhappy

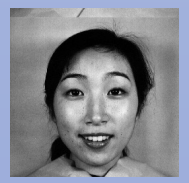


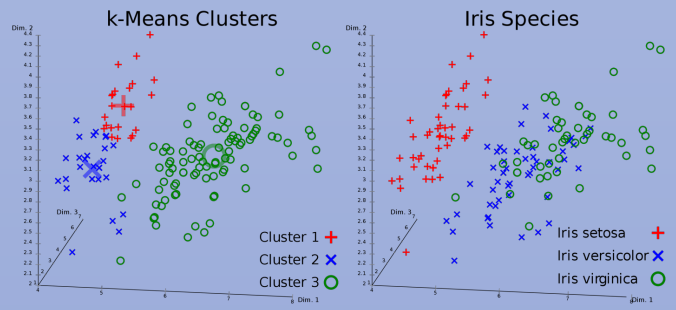
e.g. SVM

Extract Features

Learn a model

Unsupervised





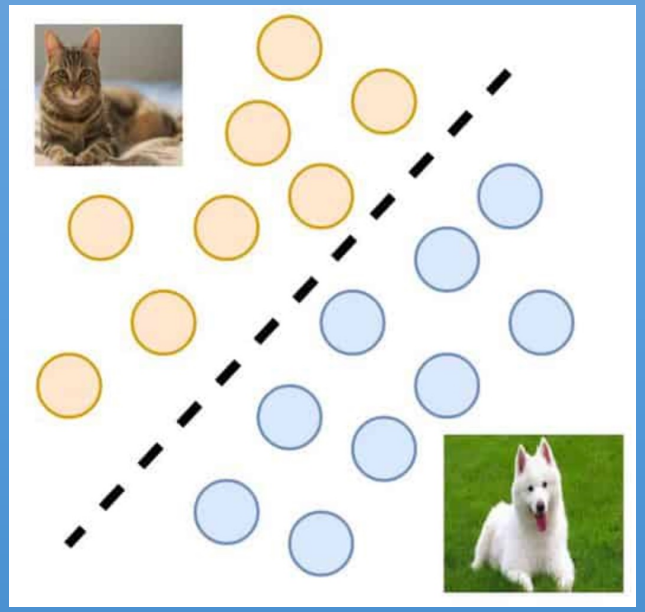
e.g. k-means clustering

Extract Features

Learn a model

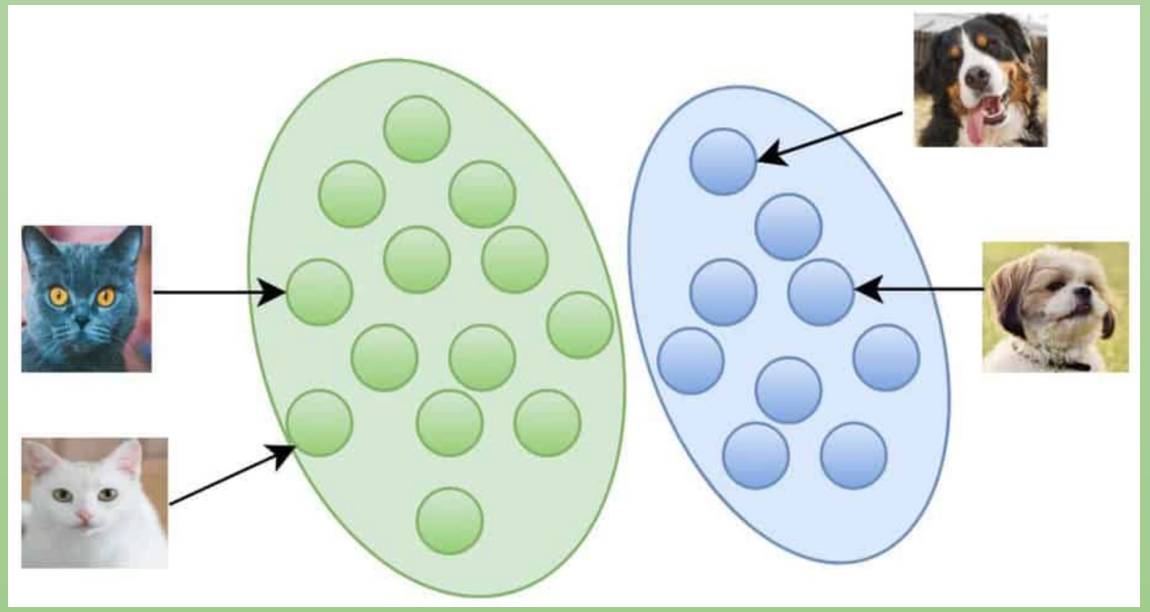
Plots: https://machinelearningmastery.com/plot-a-decision-surface-for-machine-learning/ & Wikipedia
Facial images from the JAFFE database: https://figshare.com/articles/journal_contribution/jaffe_desc_pdf/5245003

Discriminative



Find separating line
(in general: hyperplane)

Generative



Learn a model for
each class.

Figs: learnopencv.com

Unsupervised Learning via Density Estimation

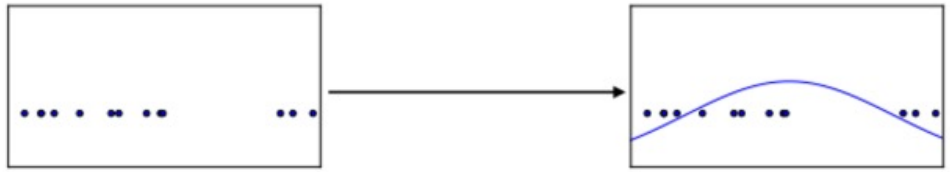
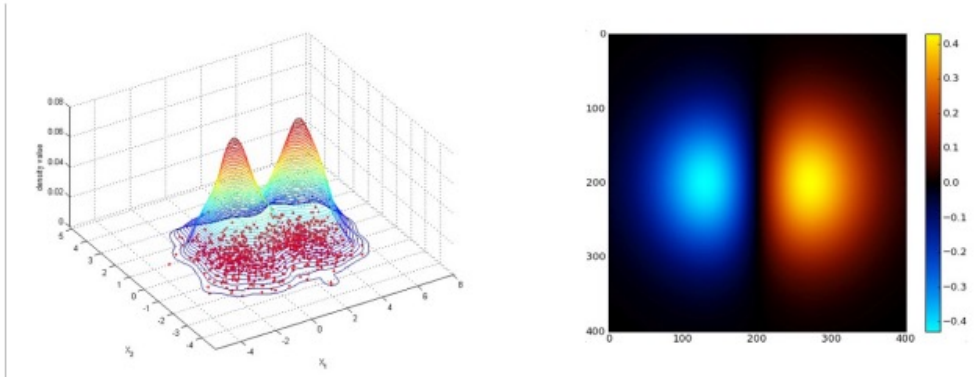


Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

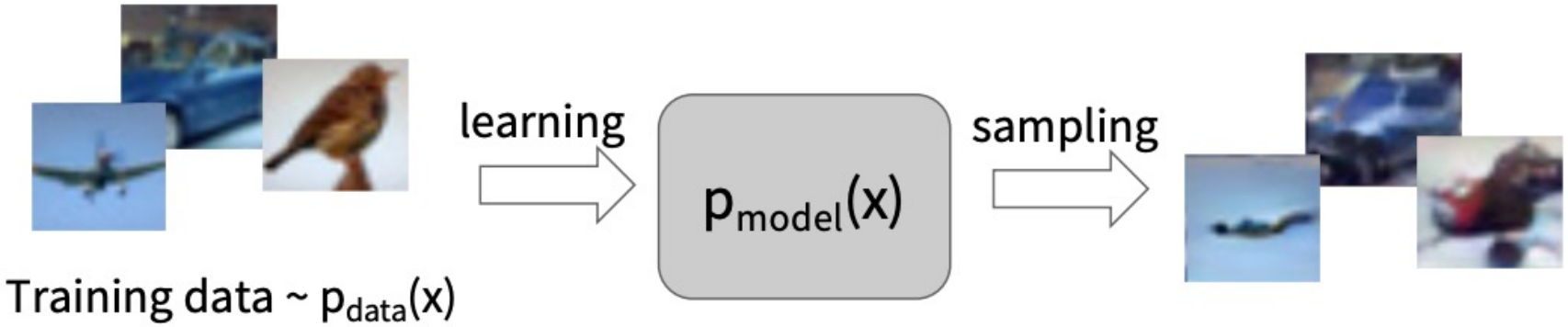
Modeling p(x)

2-d density images left and right are CC0 public domain

Figure: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Generative Modeling

- Learning the probability distribution of data



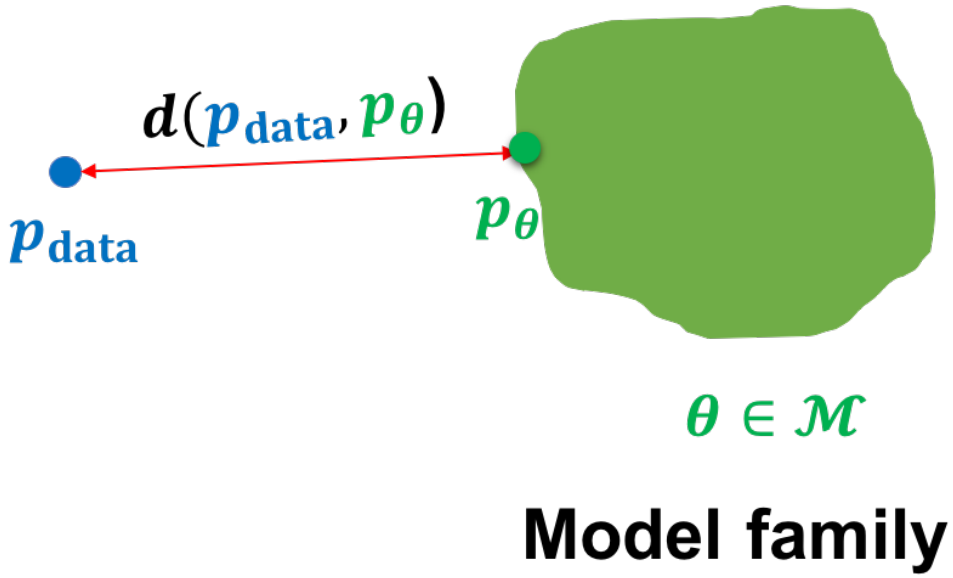
Objectives:

1. Learn $p_{model}(x)$ that approximates $p_{data}(x)$
2. Sampling new x from $p_{model}(x)$

Figure: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Generative Modeling

- Learning the probability distribution of data



$p_\theta \equiv p_{\text{model}}$

Figure: https://deepgenerativemodels.github.io/notes/introduction/

Why Generative Modeling?



- Realistic samples for artwork, super-resolution, colorization, etc.
- Learn useful features for downstream tasks such as classification.
- Getting insights from high-dimensional data (physics, medical imaging, etc.)
- Modeling physical world for simulation and planning (robotics and reinforcement learning applications)
- Many more ...

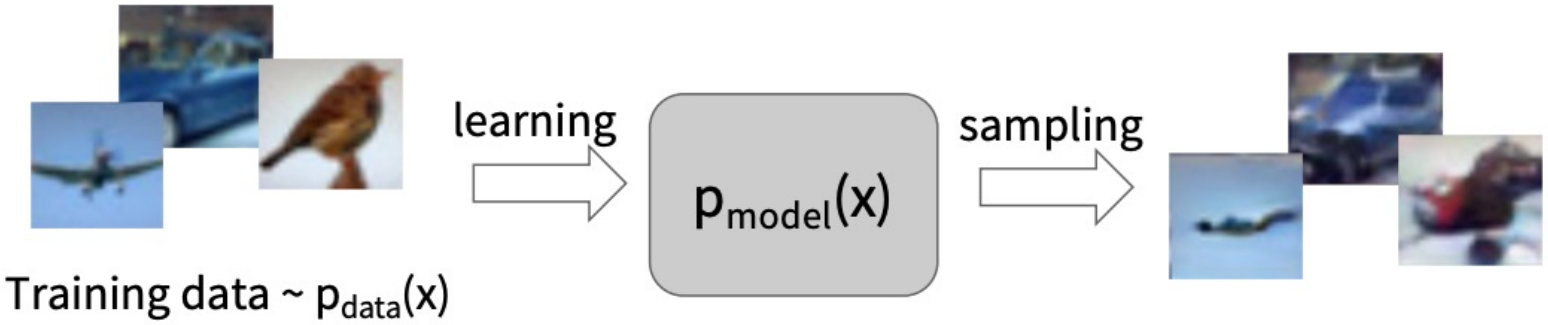
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Generative Modeling: State of the Art

- Image Generation -- Midjourney
  - https://clickup.com/blog/midjourney-prompt-examples/
- Video Generation -- OpenAI
  - https://openai.com/index/video-generation-models-as-world-simulators/
- Audio Generation -- Stable audio
  - https://stableaudio.com/
- “World” Generation -- Genie 2
  - https://deepmind.google/discover/blog/genie-2-a-large-scale-foundation-world-model/

Generative Modeling

- Learning the probability distribution of data



Formulate as density estimation problems:

- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ without explicitly defining it.

Figure: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Taxonomy of Generative Models

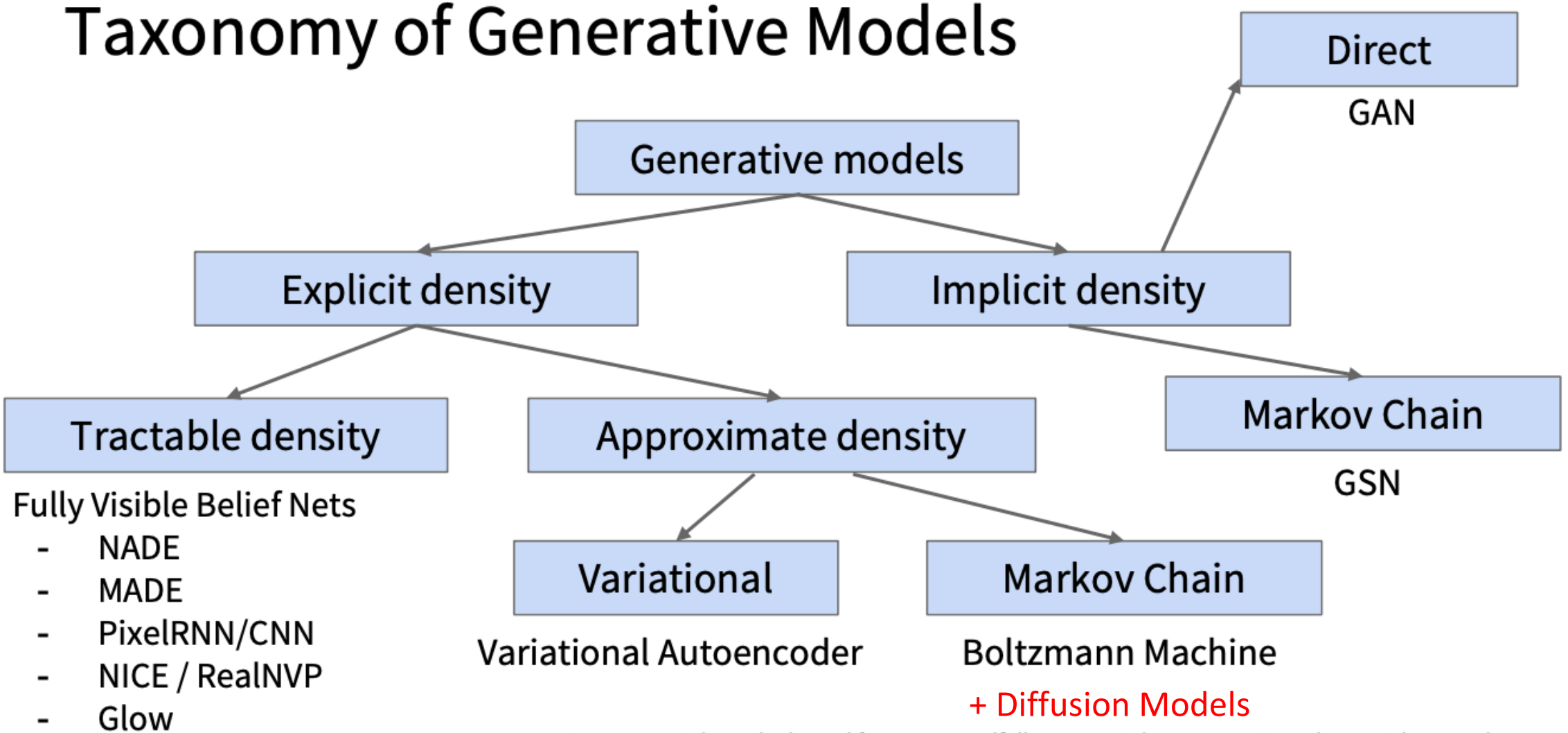


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

ADD THIS

- https://arxiv.org/abs/2512.07829

Autoregressive Models

Fully visible belief network (FVBN)

Explicit density model

$$p(x) = p(x_1, x_2, \ldots, x_n)$$

Likelihood of image x

Joint likelihood of each pixel in the image

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Fully visible belief network (FVBN)

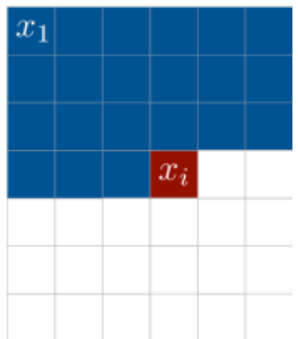
Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Likelihood of image x

Probability of i'th pixel value given all previous pixels



Then maximize likelihood of training data

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Fully visible belief network (FVBN)

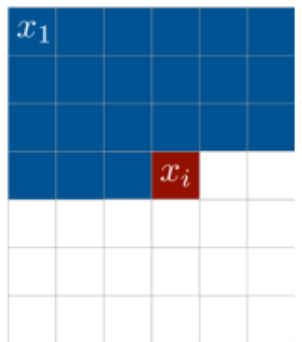
Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Likelihood of image x

Probability of i'th pixel value given all previous pixels

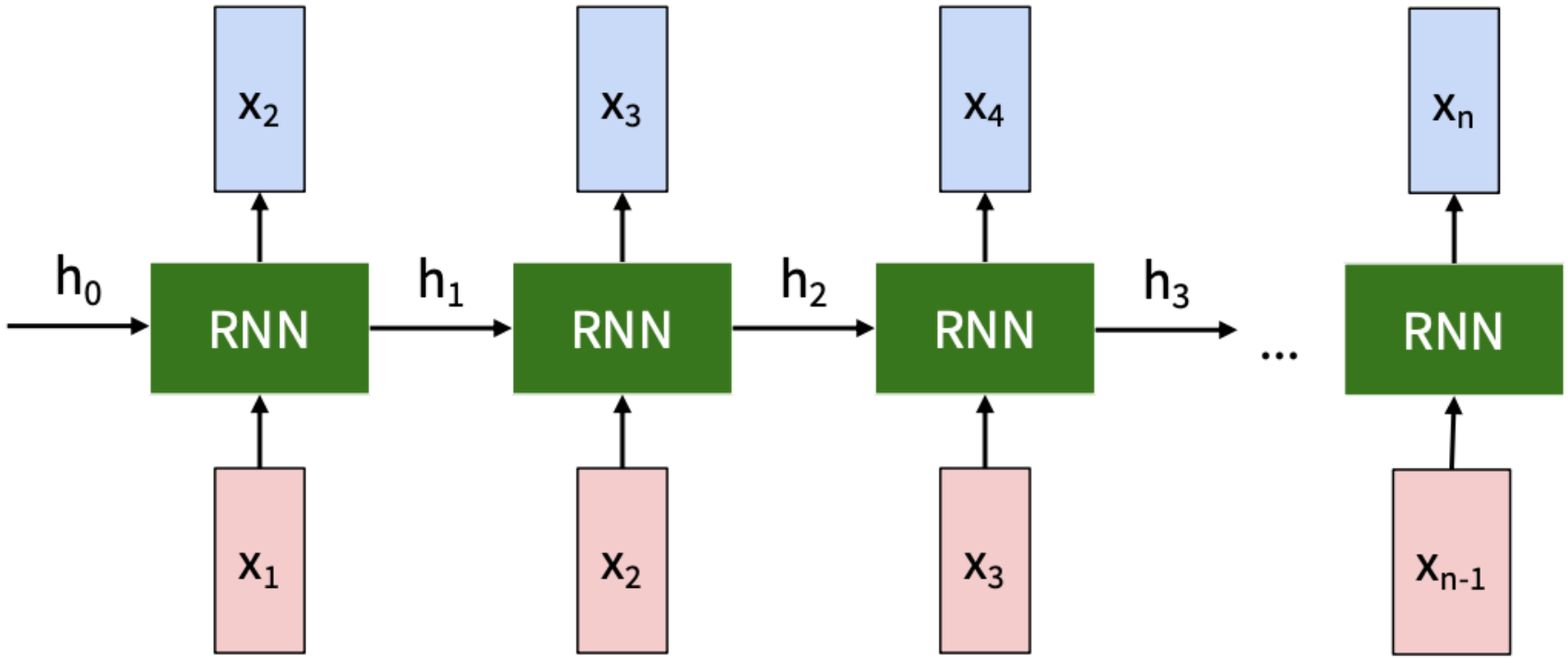


Then maximize likelihood of training data

Complex distribution over pixel values => Express using a neural network!

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Recurrent Neural Network



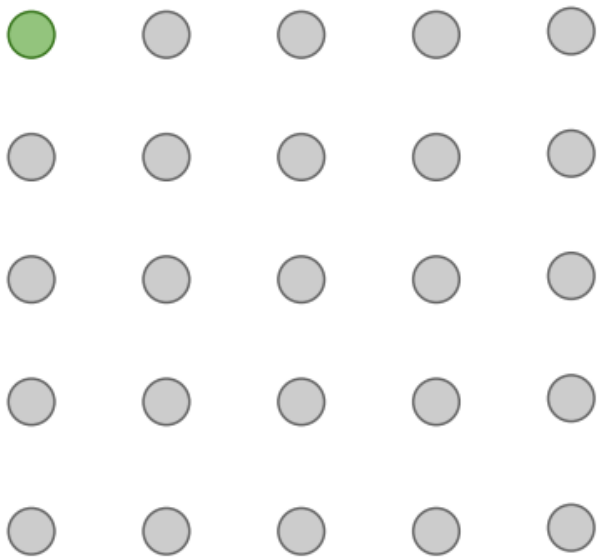
$$p(x_i|x_1, \ldots, x_{i-1})$$

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

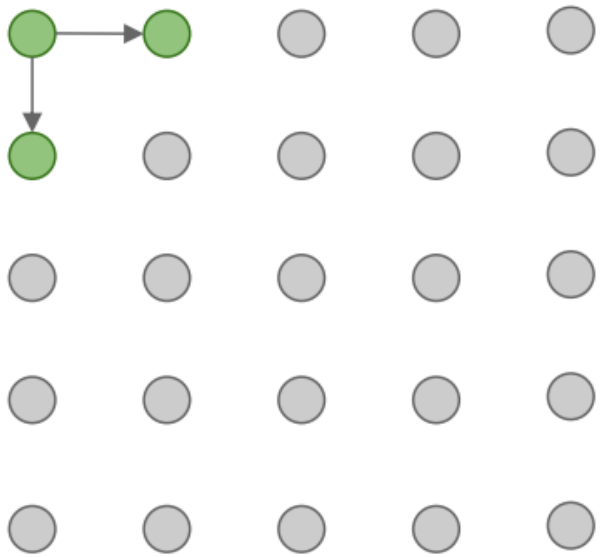


Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

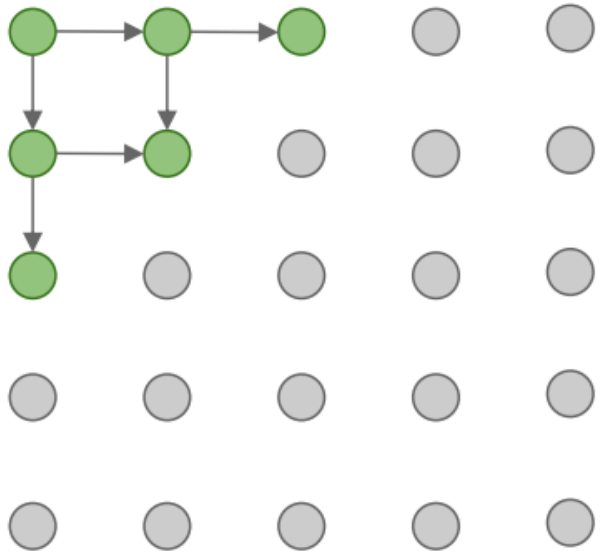


Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

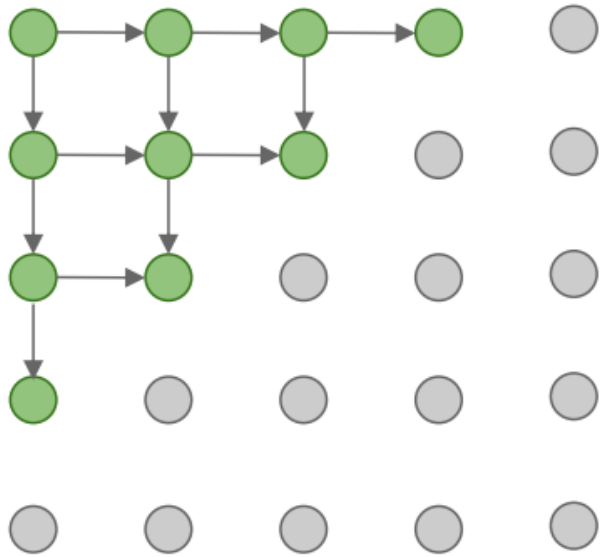


Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)



Drawback: sequential generation is slow in both training and inference!

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

PixelCNN [van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region (masked convolution)

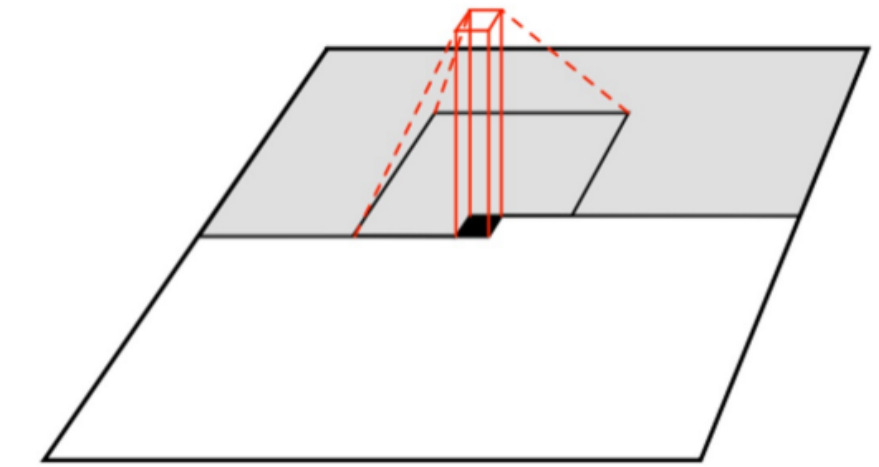


Figure copyright van der Oord et al., 2016. Reproduced with permission.

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

PixelCNN [van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region (masked convolution)

Training is faster than PixelRNN
(can parallelize convolutions since context region values known from training images)

Generation is still slow:
For a 32x32 image, we need to do forward passes of the network 1024 times for a single image

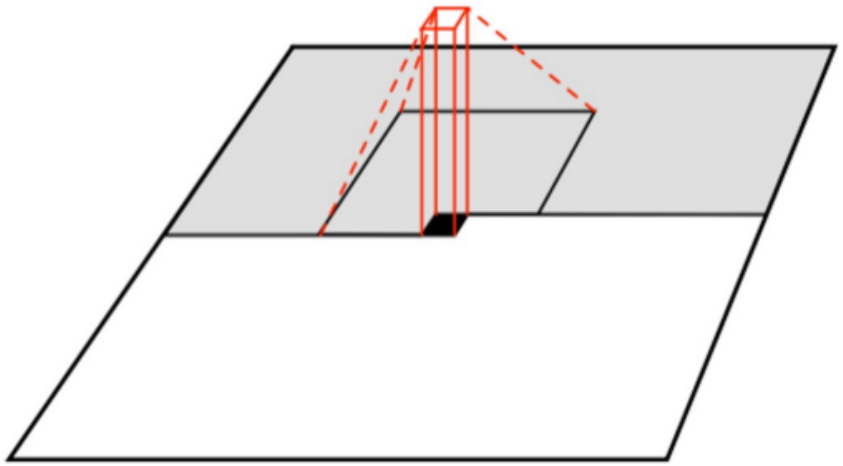
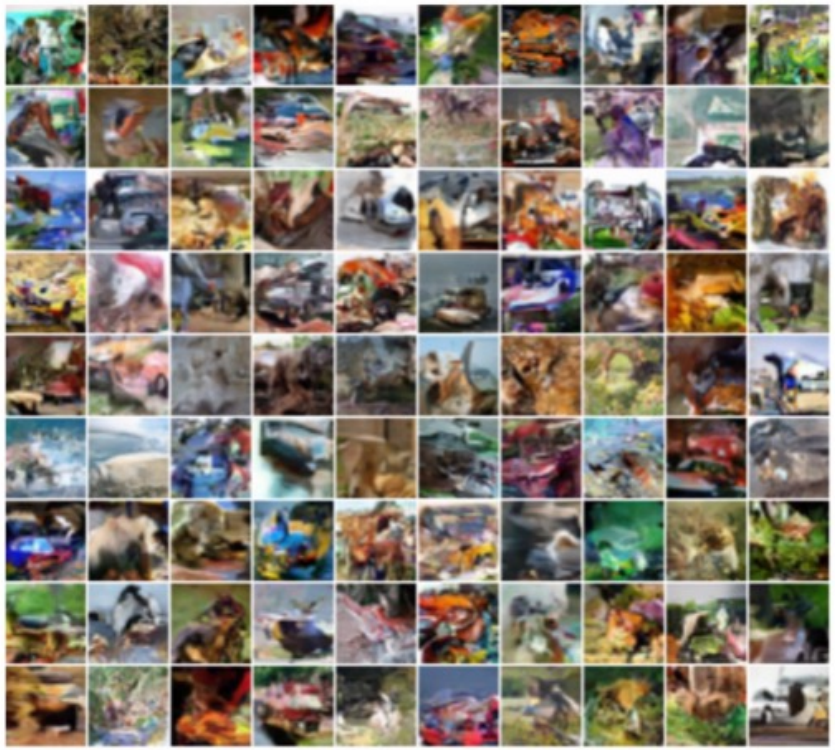
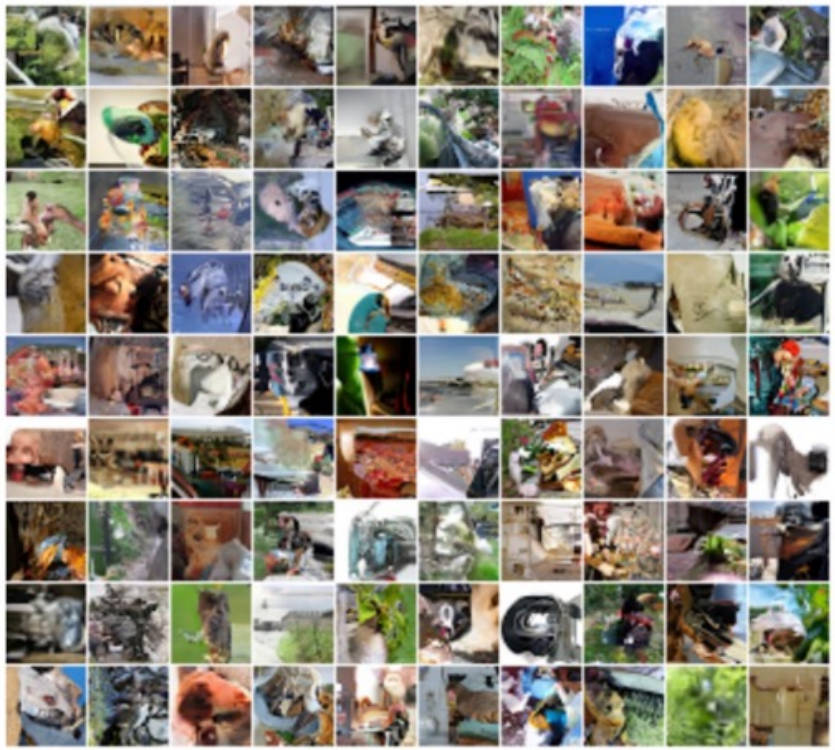


Figure copyright van der Oord et al., 2016. Reproduced with permission.

Generation Samples



32x32 CIFAR-10



32x32 ImageNet

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

PixelRNN and PixelCNN

Pros:

- Can explicitly compute likelihood p(x)
- Easy to optimize
- Good samples

Con:

- Sequential generation => slow

Improving PixelCNN performance

- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc…

See

- Van der Oord et al. NIPS 2016
- Salimans et al. 2017 (PixelCNN++)

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders

Autoregressive Models vs Variational Autoencoders

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

Variational Autoencoders (VAEs) define intractable density function with latent z:

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

No dependencies among pixels, can generate all pixels at the same time!

Cannot optimize directly, derive and optimize lower bound on likelihood instead

Why latent z?

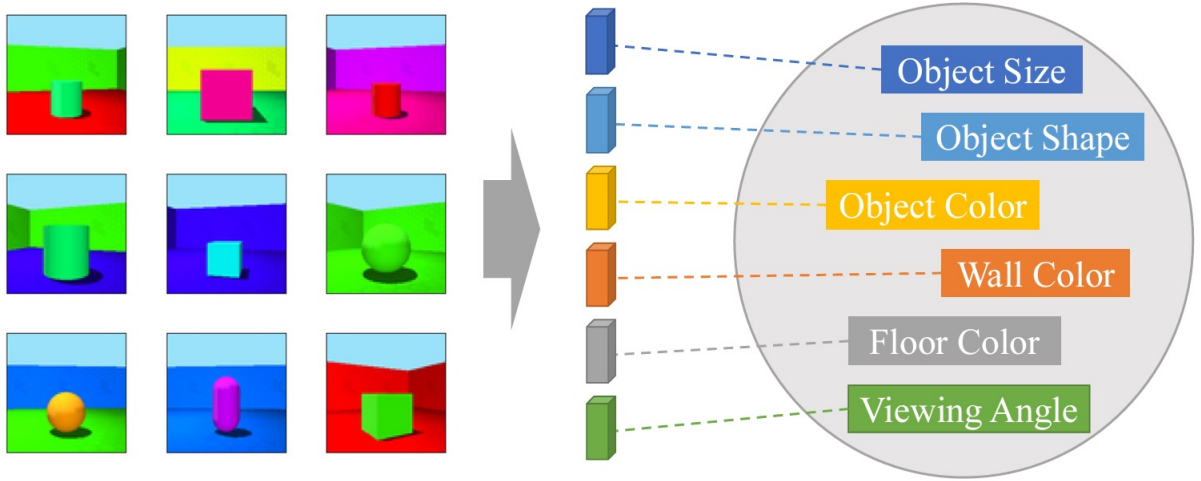
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Disentangled Representation Learning

$x^{(i)} \sim \text{Sim}(w^{(i)})$

$D = \{X, W, Y\}$

K conditionally independent factors



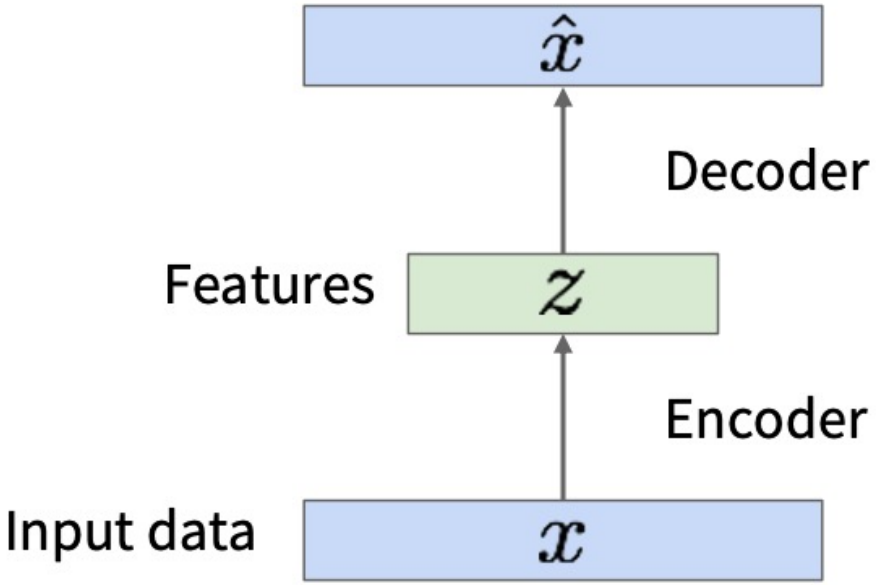
$z^{(i)} \in R^L$

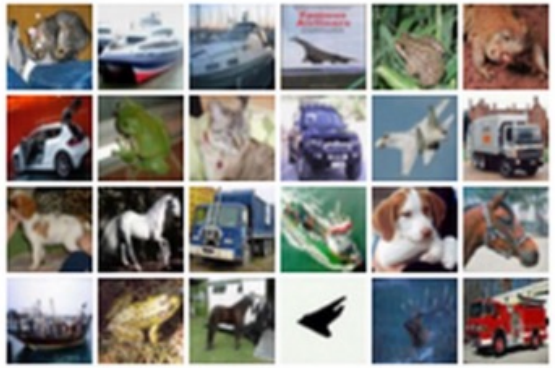
$p(x^{(i)}|z^{(i)}) \sim \text{Sim}(w^{(i)})$

Figure: X. Wang, H. Chen, S. Tang, Z. Wu, and W. Zhu. Disentangled representation learning. arXiv preprint arXiv:2211.11695, 2022.

Recap: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data





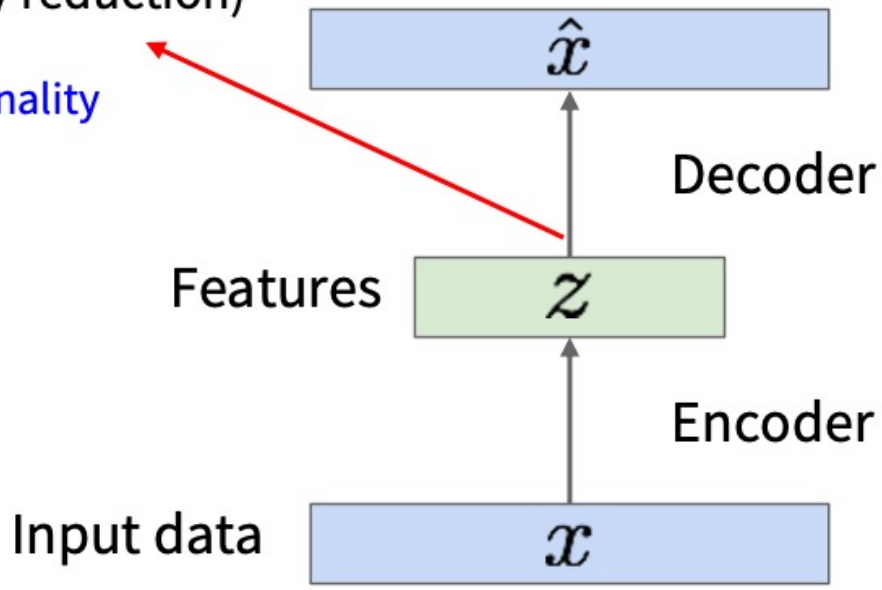
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

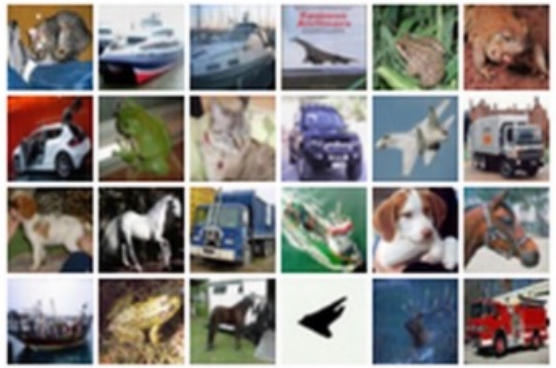
Recap: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?





Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

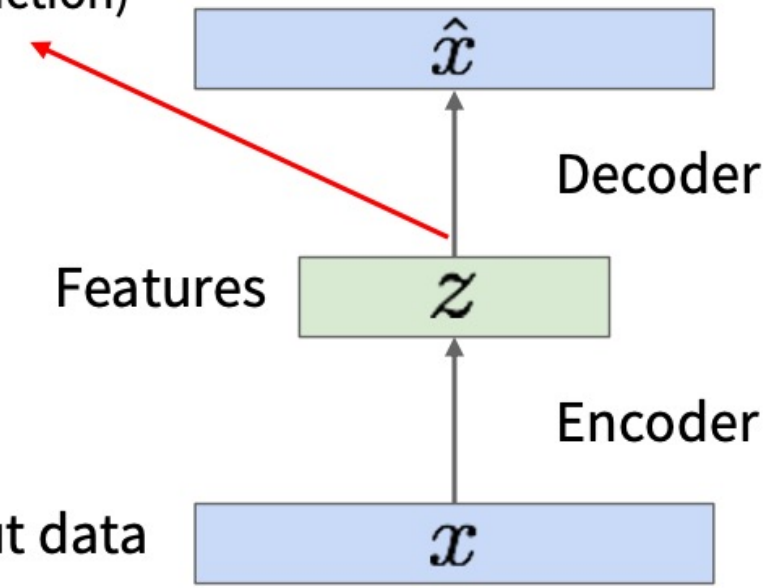
Recap: Autoencoders

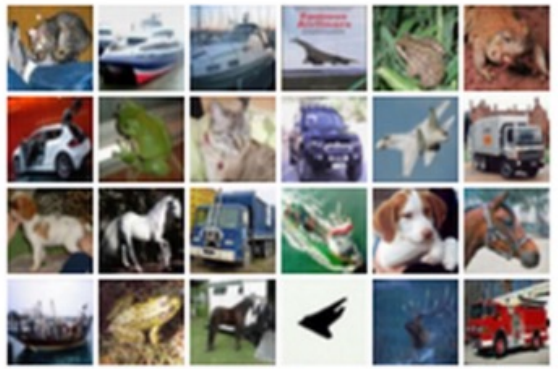
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?

A: Want features to capture meaningful factors of variation in data



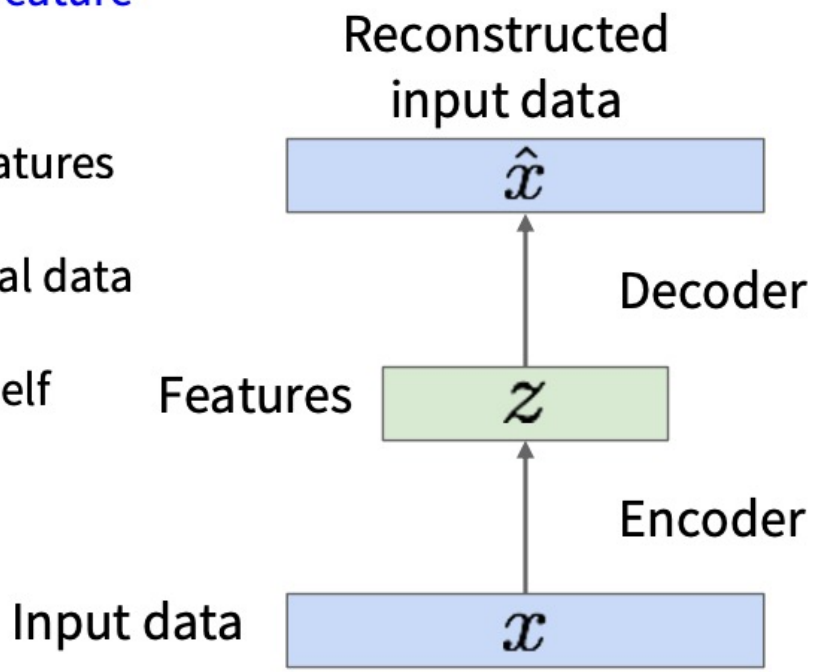


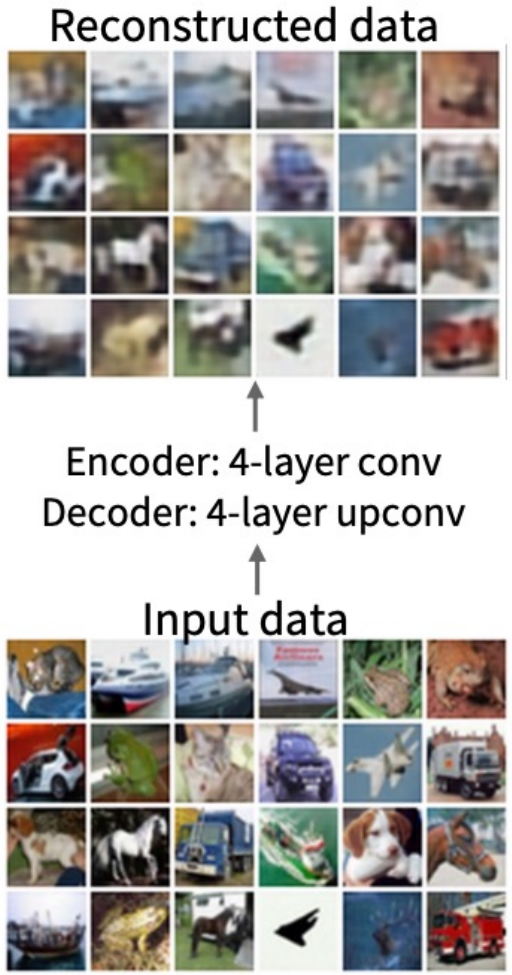
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Recap: Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data
"Autoencoding" - encoding input itself



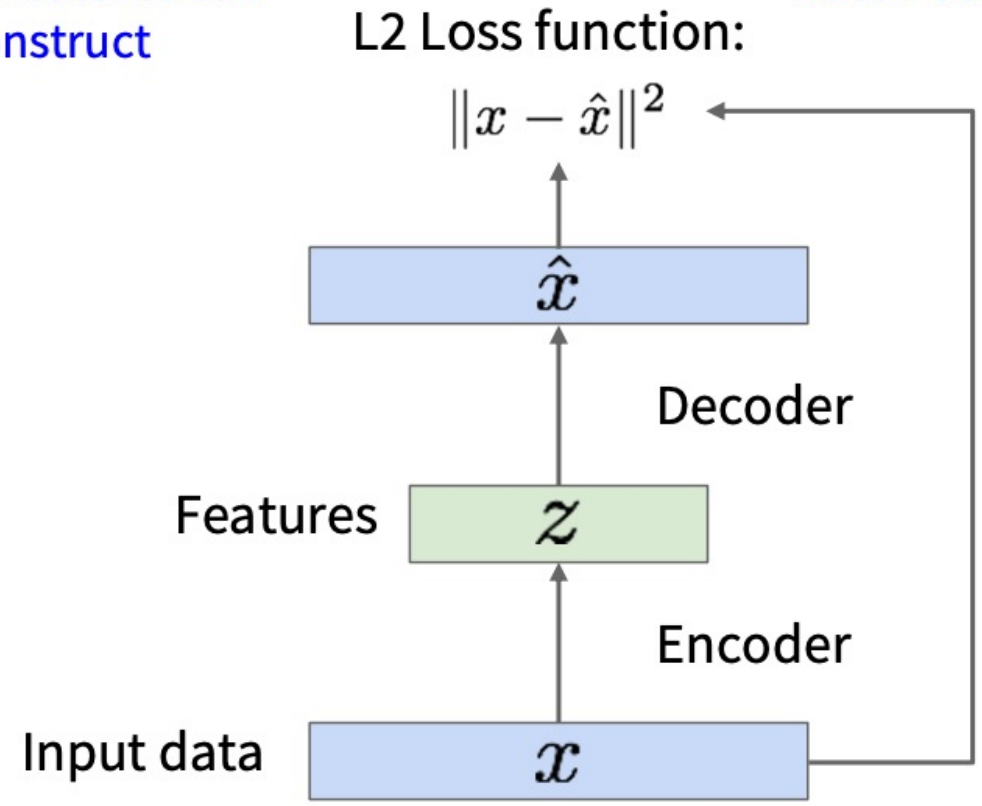


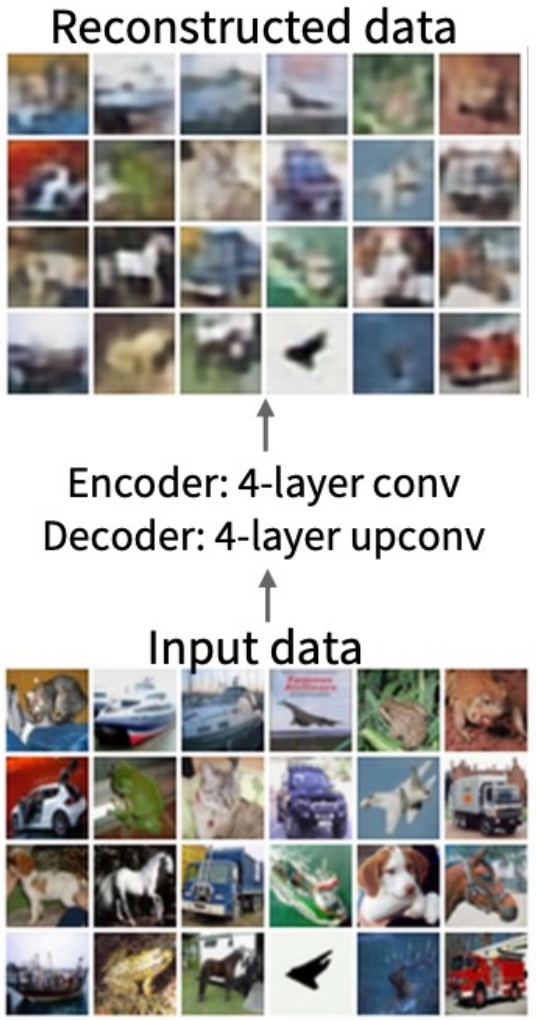
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Recap: Autoencoders

Train such that features can be used to reconstruct original data

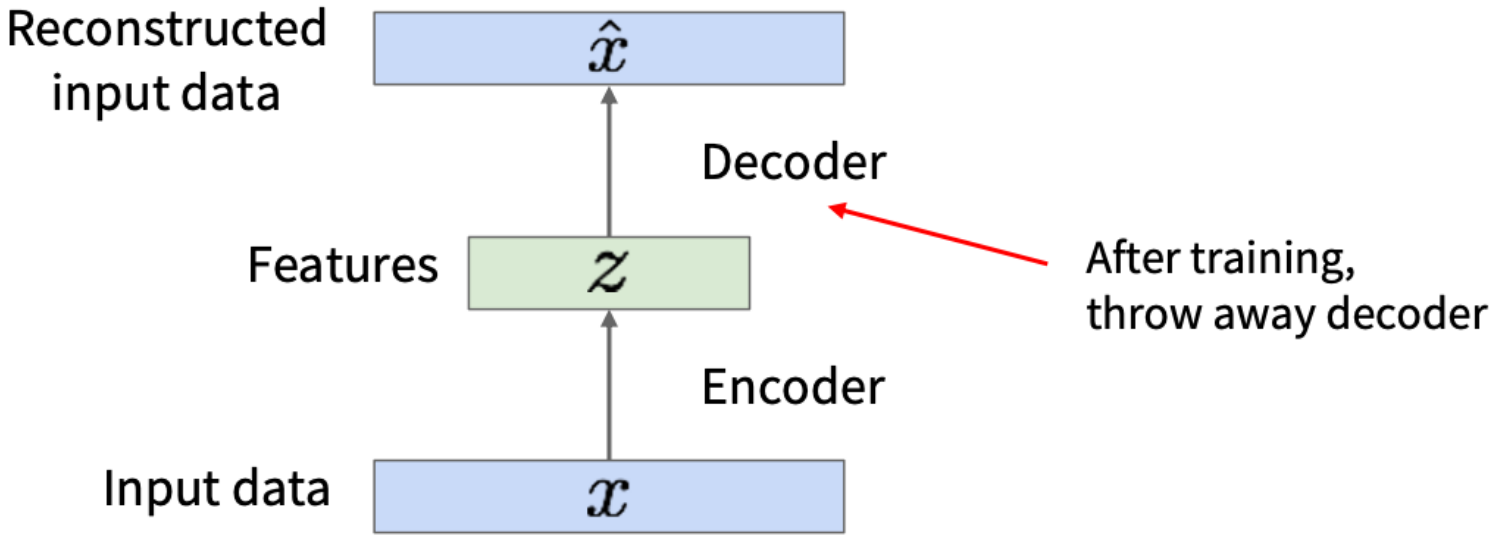
Doesn't use labels!





Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

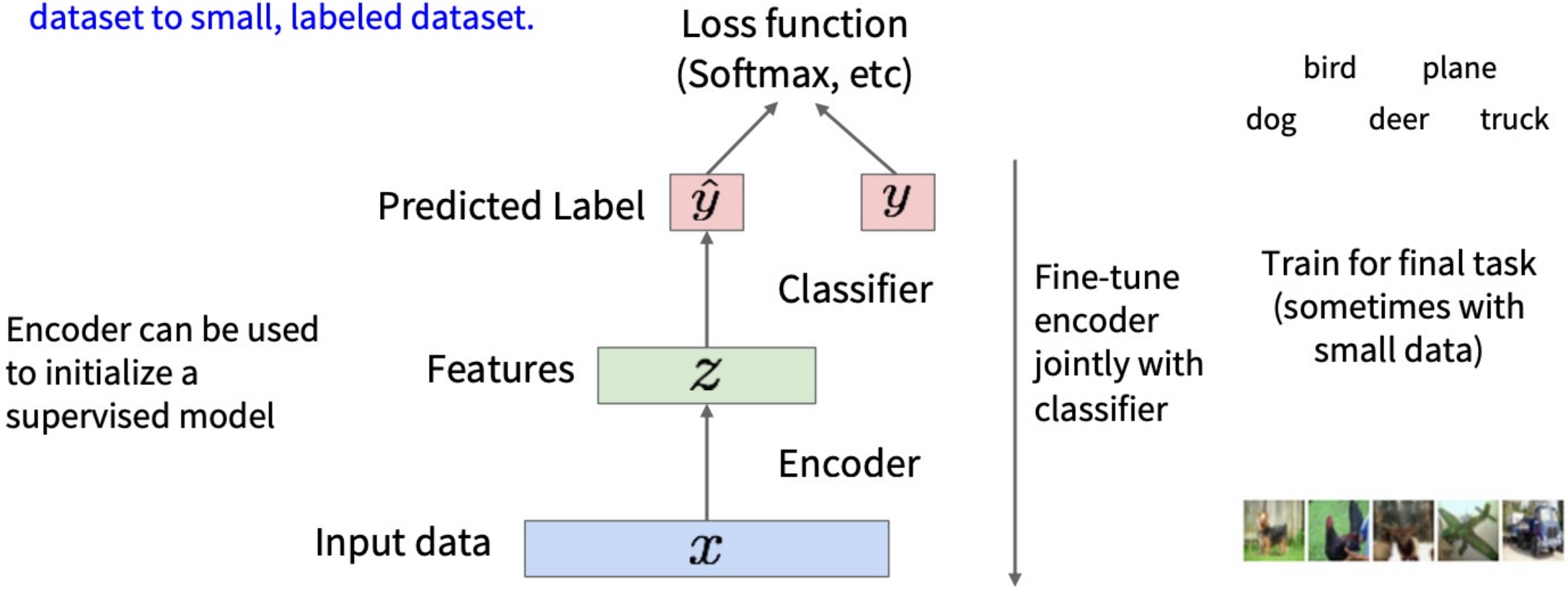
Recap: Autoencoders



Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

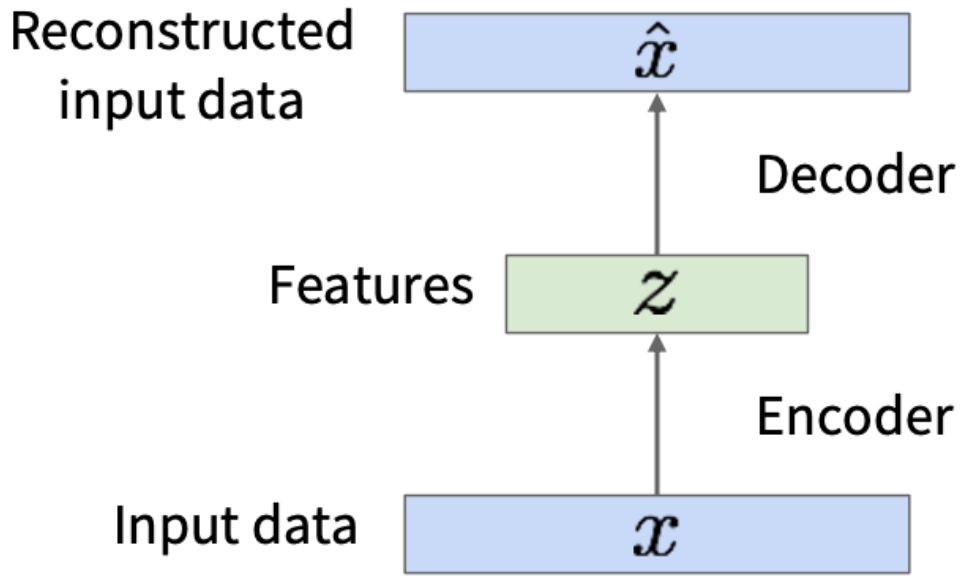
Recap: Autoencoders

Transfer from large, unlabeled dataset to small, labeled dataset.



Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Recap: Autoencoders



Autoencoders can reconstruct data, and can learn features to initialize a supervised model

Features capture factors of variation in training data.

But we can't generate new images from an autoencoder because we don't know the space of z.

How do we make autoencoder a generative model?

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

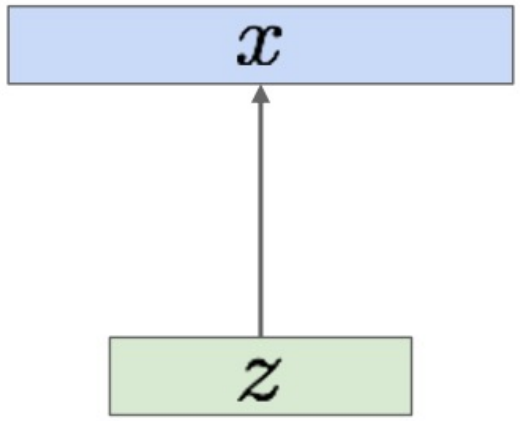
Variational Autoencoders (VAEs)

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

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

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

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



Intuition (remember from autoencoders!): x is an image, z is latent factors used to generate x: attributes, orientation, etc.

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

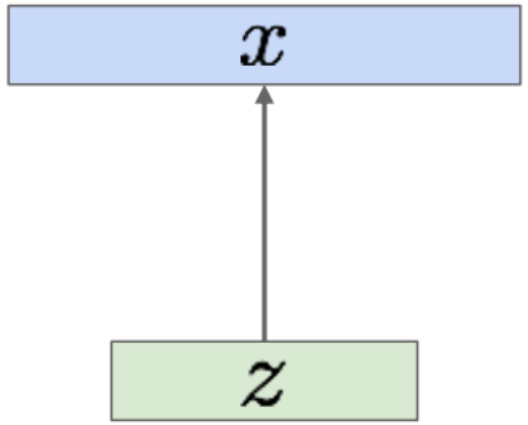
We want to estimate the true parameters θ^* of this generative model given training data x.

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$z^{(i)} \sim p_{\theta^*}(z)$



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

We want to estimate the true parameters θ^* of this generative model given training data x.

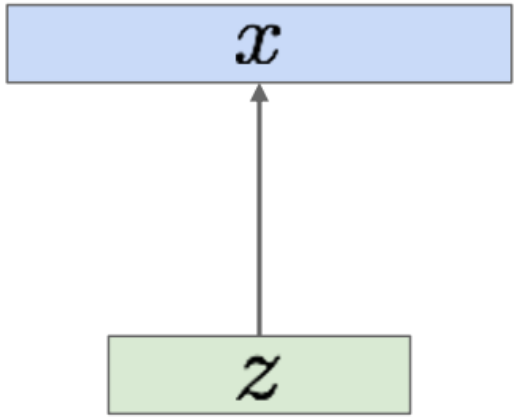
How should we represent this model?

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$z^{(i)} \sim p_{\theta^*}(z)$



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

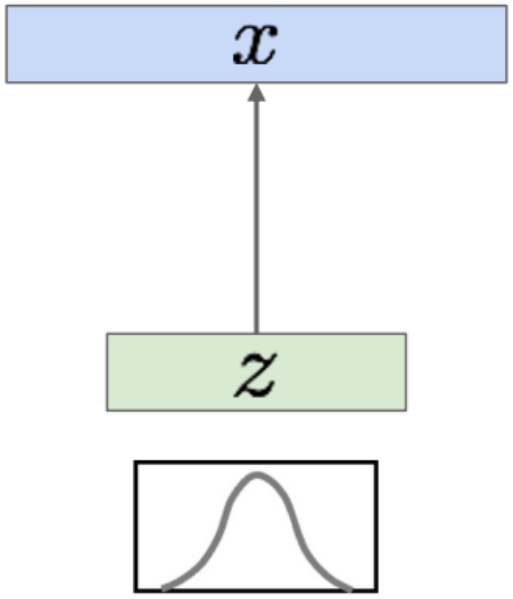
Variational Autoencoders (VAEs)

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$z^{(i)} \sim p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model given training data x.

How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

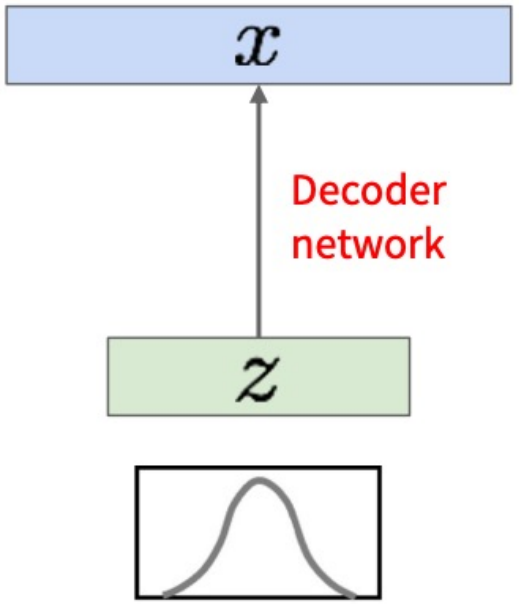
Variational Autoencoders (VAEs)

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$z^{(i)} \sim p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model given training data x.

How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

Conditional p(x|z) is complex (generates image) => represent with neural network

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

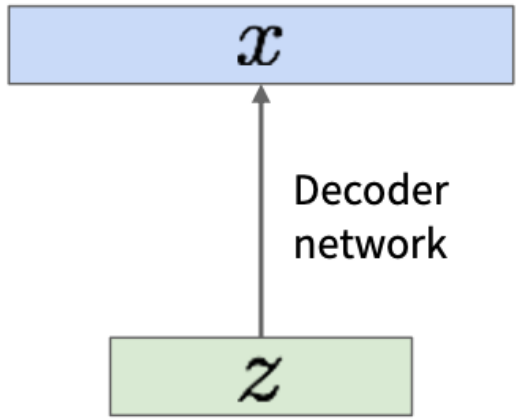
Variational Autoencoders (VAEs)

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$z^{(i)} \sim p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model given training data x.

How to train the model?

Learn model parameters to maximize likelihood of training data

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

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

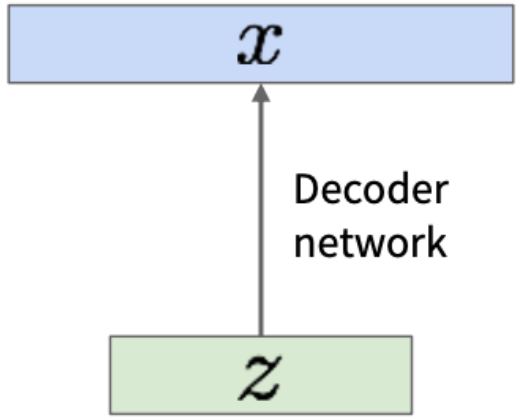
Variational Autoencoders (VAEs)

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$z^{(i)} \sim p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model given training data x.

How to train the model?

Learn model parameters to maximize likelihood of training data

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

Q: What is the problem with this?

Intractable!

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs): Intractability

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

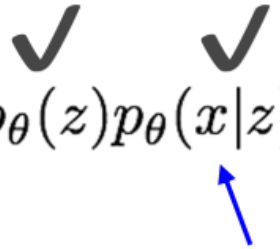
✓ Simple Gaussian prior

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs): Intractability

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



Decoder neural network

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs): Intractability

😢 ✓ ✓

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

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

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs): Intractability

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

😢 ✓ ✓

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

$$\log p(x) \approx \log \frac{1}{k} \sum_{i=1}^{k} p(x|z^{(i)}), \text{ where } z^{(i)} \sim p(z)$$

Monte Carlo estimation is too high variance

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs): Intractability

😢 ✓ ✓

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

Posterior density: $p_\theta(z|x) = p_\theta(x|z) p_\theta(z) / p_\theta(x)$

Intractable data likelihood

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

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

Posterior density also intractable: $p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$

Solution: In addition to modeling $p_\theta(x|z)$, learn $q_\phi(z|x)$ that approximates the true posterior $p_\theta(z|x)$.

Will see that the approximate posterior allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

Variational inference is to approximate the unknown posterior distribution from only the observed data x

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

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

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

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

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

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

$$p_\theta(z|x^{(i)}) = \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(x^{(i)})}$$

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

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

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

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

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)} \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)} \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}
\end{aligned}
$$

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}
$$

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

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

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

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

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

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

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

We want to maximize the data likelihood

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

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

$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always >= 0.

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

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

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

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

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

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

We want to maximize the data likelihood

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

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

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

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

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

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

Decoder: reconstruct the input data

Encoder: make approximate posterior distribution close to prior

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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

Let's look at computing the KL divergence between the estimated posterior and the prior given some data

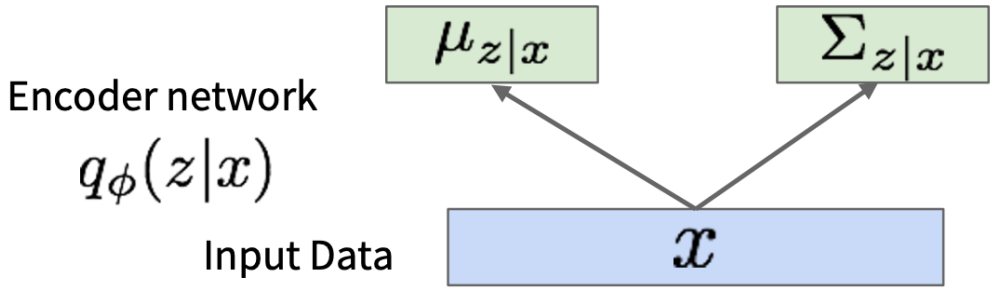
Input Data x

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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



Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

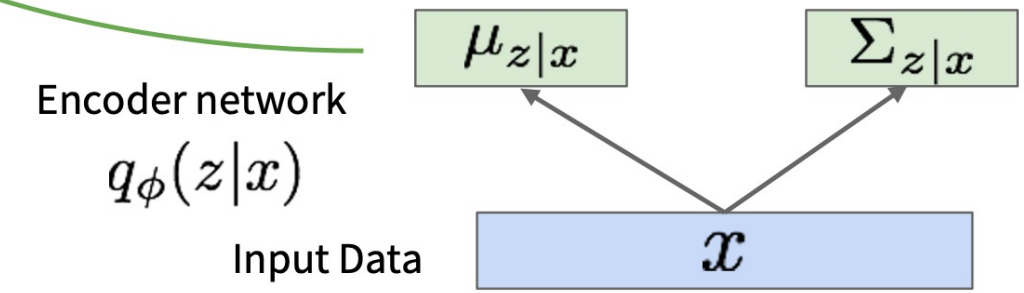
Putting it all together: maximizing the likelihood lower bound

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

$$D_{KL}(\mathcal{N}(\mu_{z|x}, \Sigma_{z|x})||\mathcal{N}(0, I))$$

Have analytical solution

Make approximate posterior distribution close to prior



Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

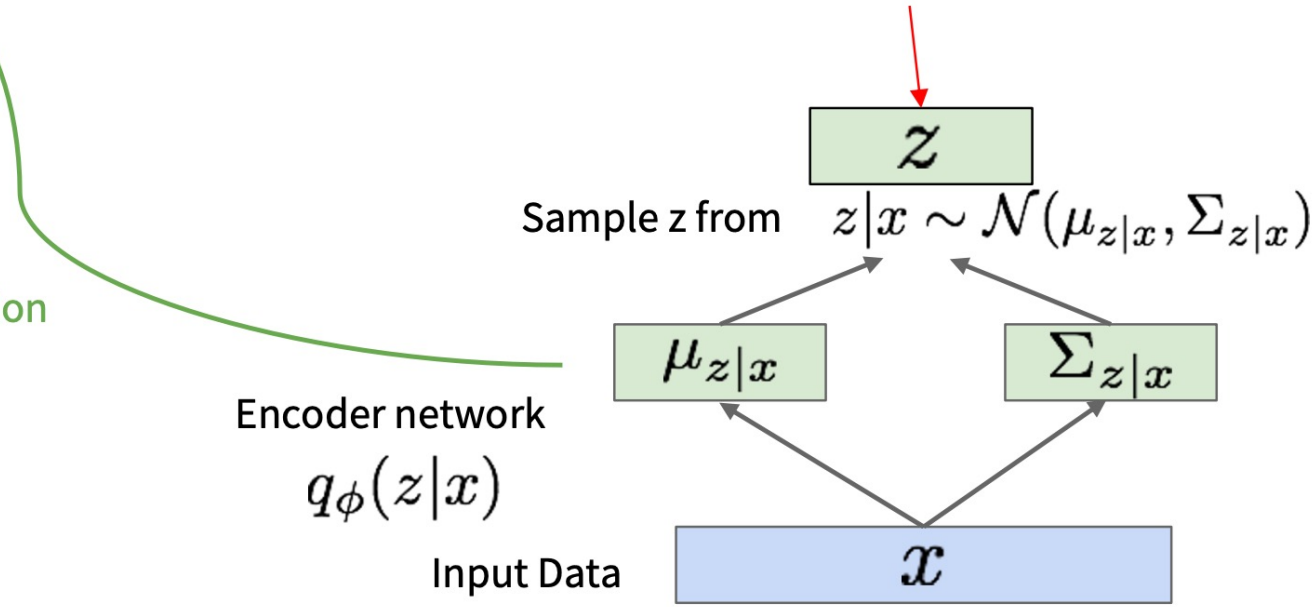
Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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

Make approximate posterior distribution close to prior

Not part of the computation graph!



Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

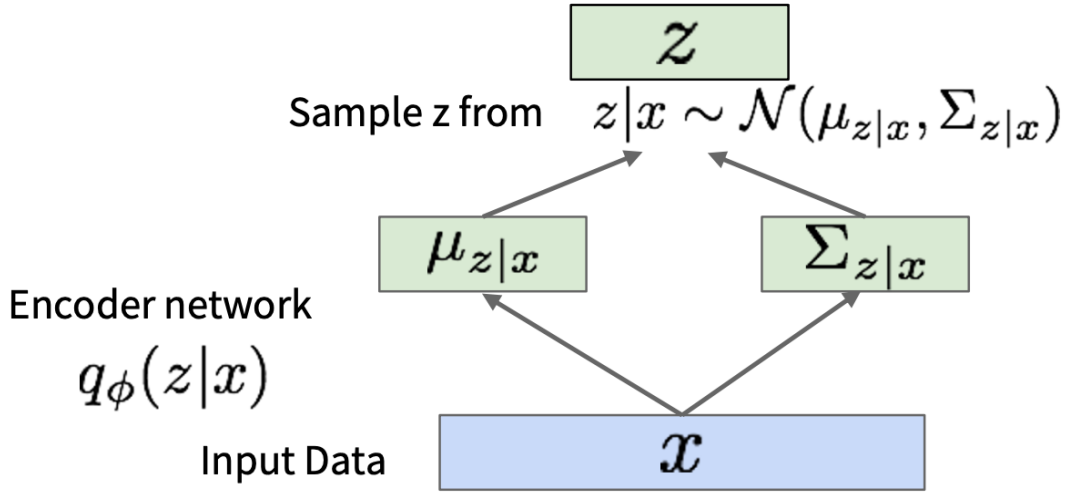
Putting it all together: maximizing the likelihood lower bound

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

Reparameterization trick to make sampling differentiable:

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

$$z = \mu_{z|x} + \epsilon\sigma_{z|x}$$



Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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

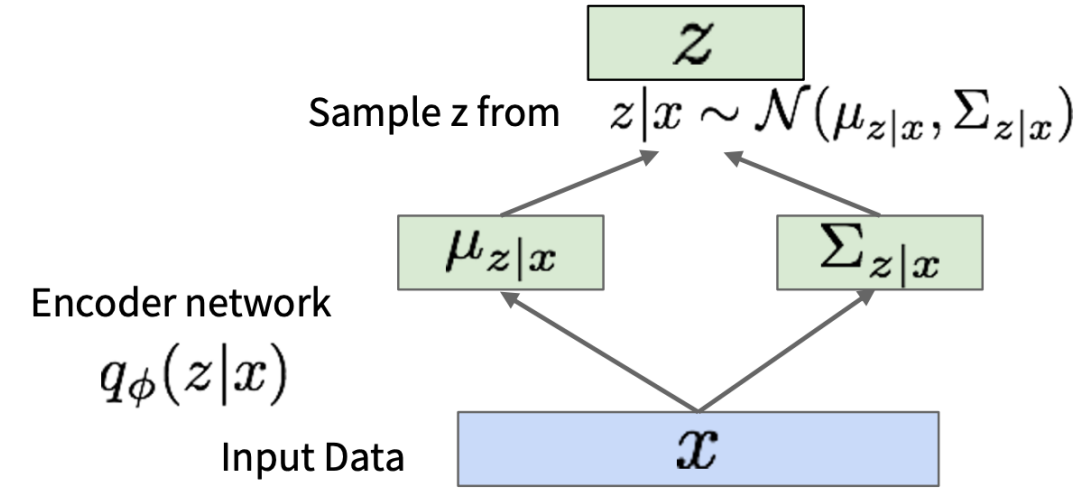
Reparameterization trick to make sampling differentiable:

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

$z = \mu_{z|x} + \epsilon \sigma_{z|x}$

Input to the graph

Part of computation graph



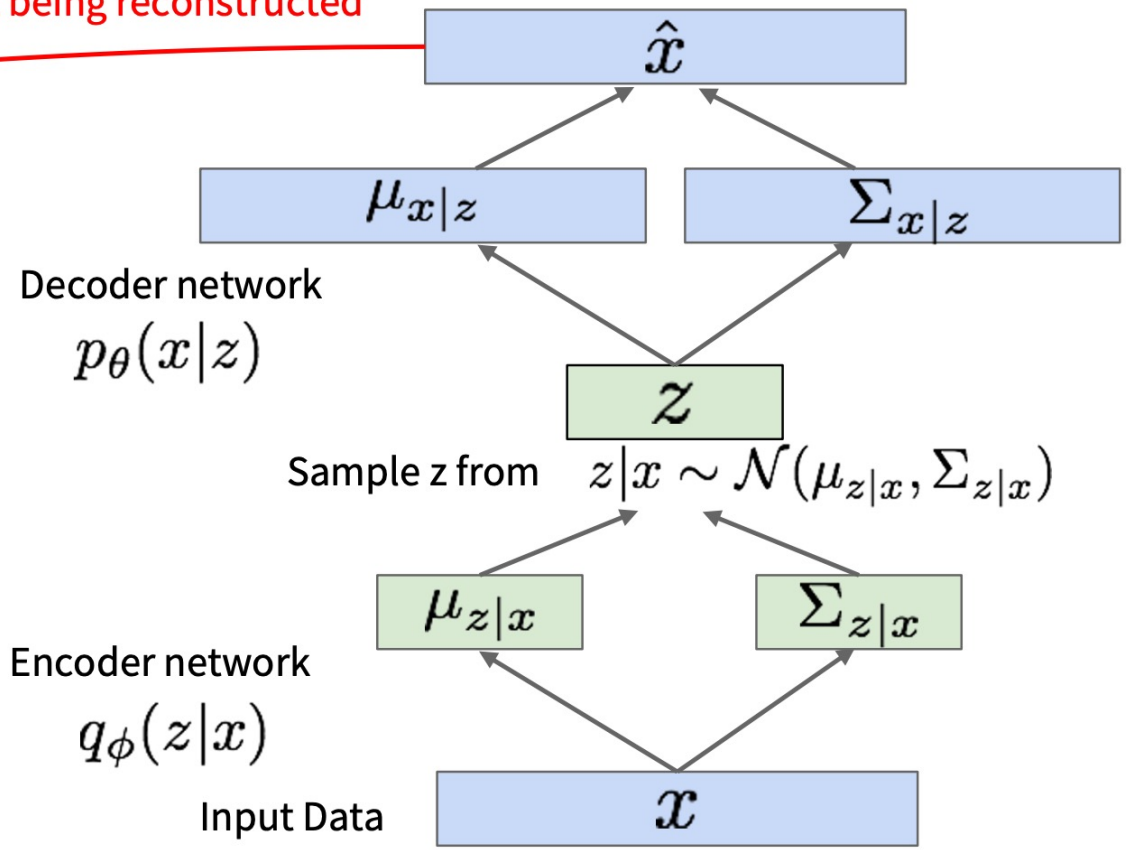
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

Maximize likelihood of original input being reconstructed

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



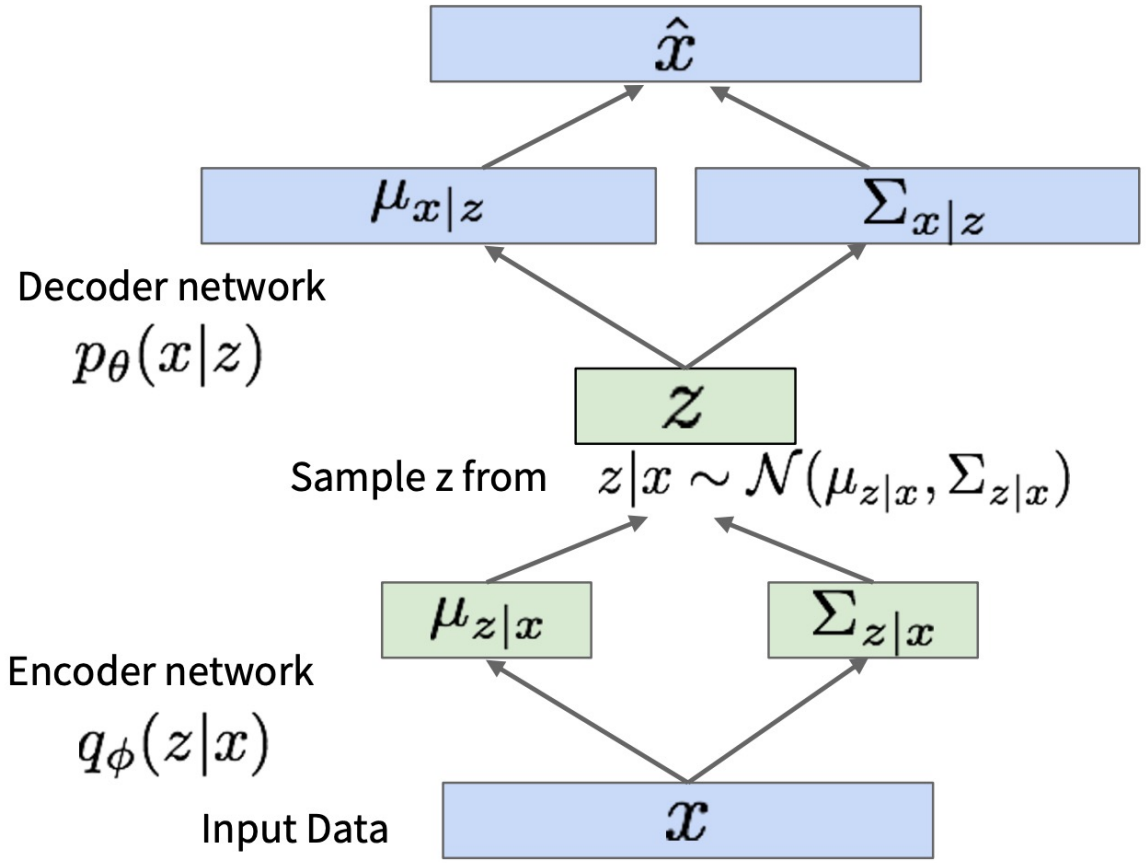
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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

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



Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

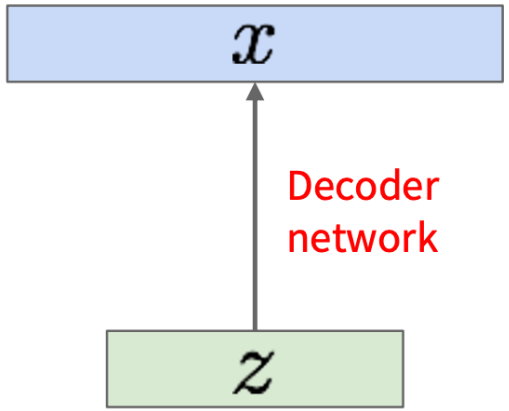
Our assumption about data generation process

Sample from true conditional

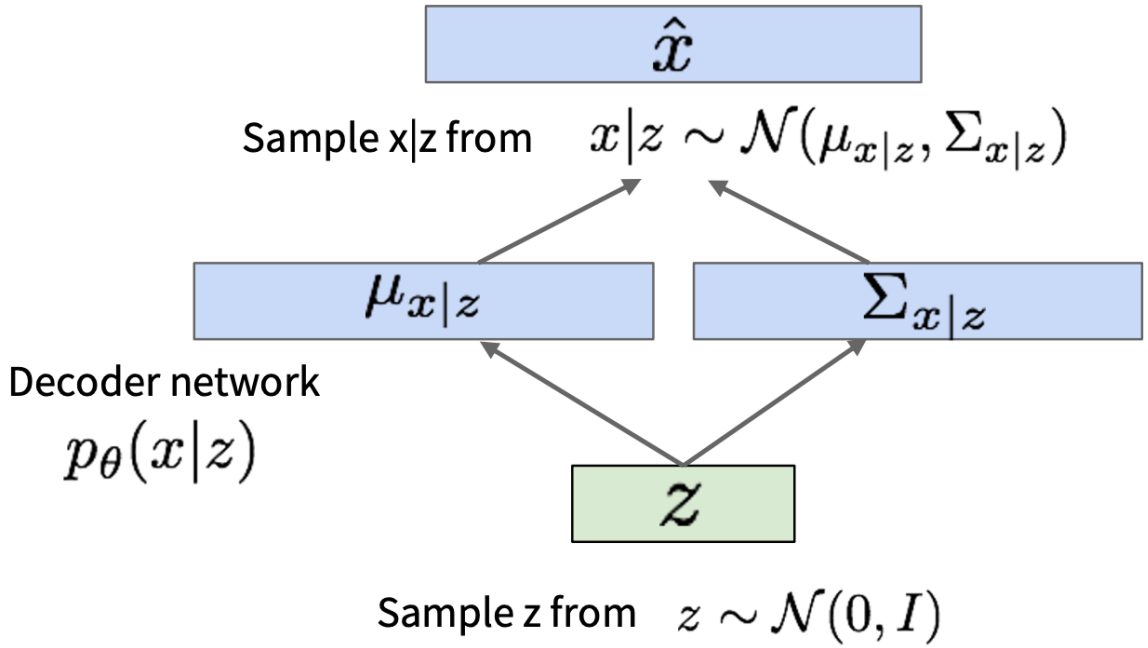
$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$z^{(i)} \sim p_{\theta^*}(z)$



Now given a trained VAE:
use decoder network & sample z from prior!

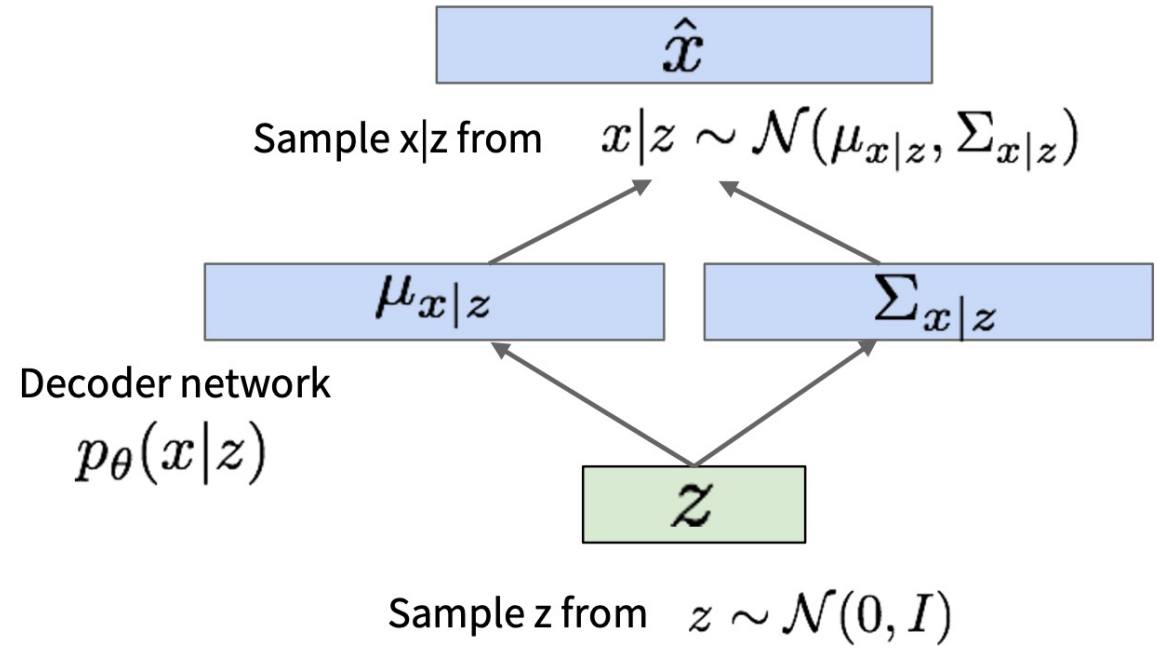


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

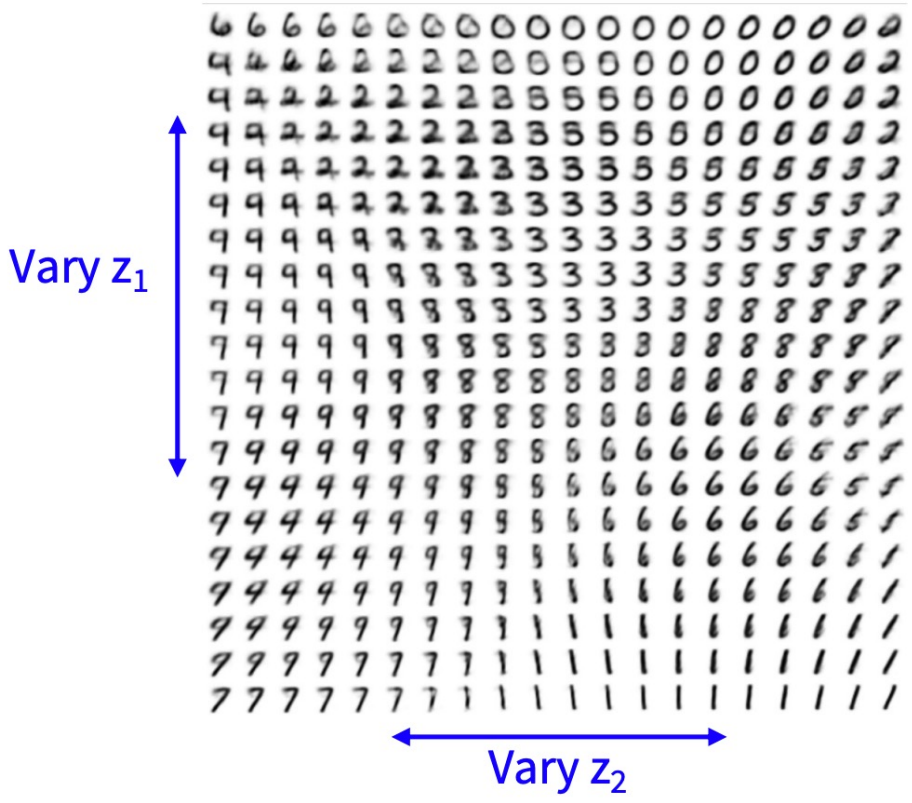
Variational Autoencoders (VAEs)

Use decoder network. Now sample z from prior!



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Data manifold for 2-d z



Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

Diagonal prior on z
=> independent
latent variables

Different dimensions
of z encode
interpretable factors
of variation

Also good feature representation that can be computed using $q_\phi(z|x)$!

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

32x32 CIFAR-10

Labeled Faces in the Wild

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Variational Autoencoders (VAEs)

Probabilistic spin to traditional autoencoders => allows generating data
Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models
- Interpretable latent space.
- Allows inference of q(z|x), can be useful feature representation for other tasks

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs), Categorical Distributions.
- Learning disentangled representations.

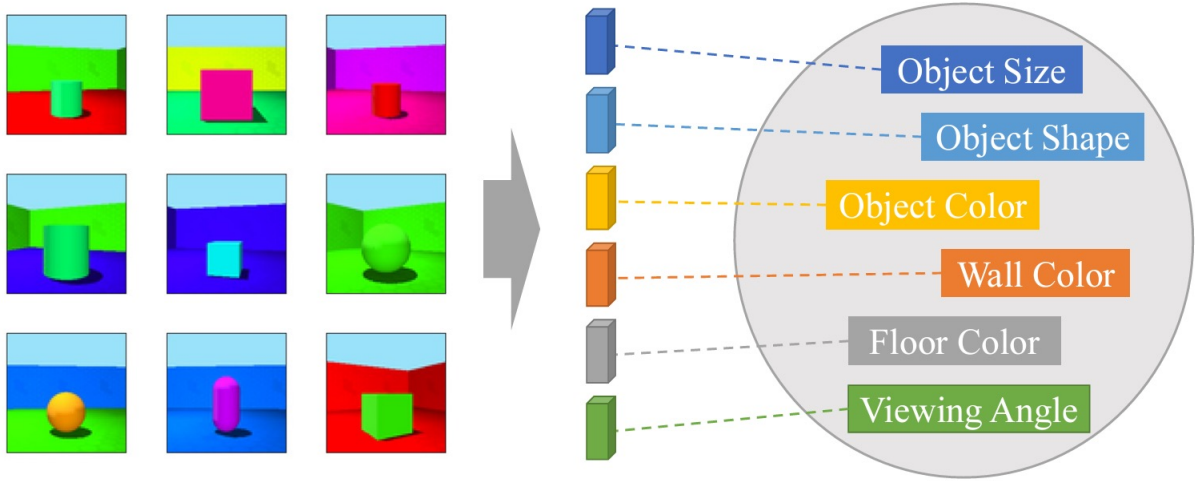
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Disentangled Representation Learning

$x^{(i)} \sim \text{Sim}(w^{(i)})$

$D = \{X, W, Y\}$

K conditionally independent factors

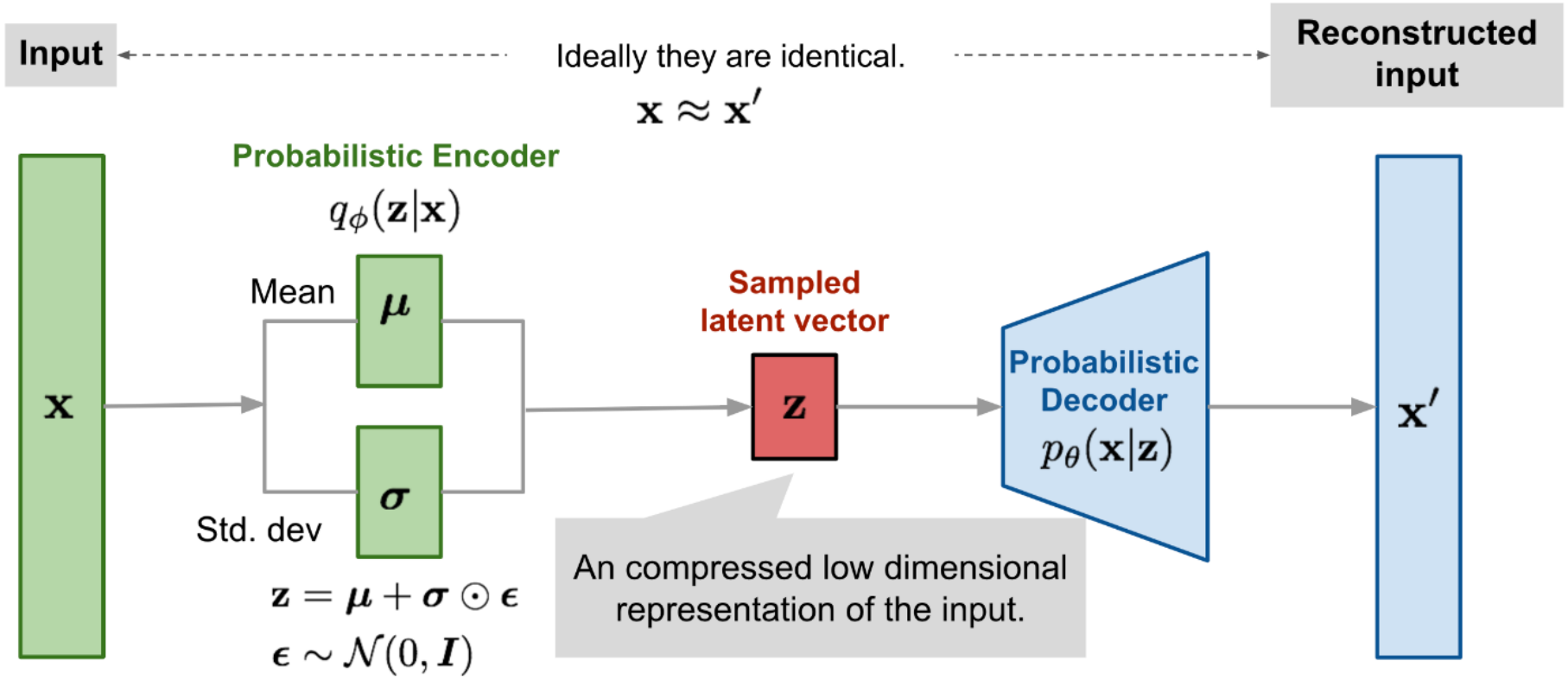


$z^{(i)} \in R^L$

$p(x^{(i)}|z^{(i)}) \sim \text{Sim}(w^{(i)})$

Figure: X. Wang, H. Chen, S. Tang, Z. Wu, and W. Zhu. Disentangled representation learning. arXiv preprint arXiv:2211.11695, 2022.

Variational Auto-Encoders and its Variations



$$L(\theta, \phi; \mathbf{x}, \mathbf{z}, \beta) = E_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x}|\mathbf{z})\Big] - D_{KL}\Big(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\Big)$$

β-VAE (Higgins et al., 2016)

$$L(\theta, \phi; \mathbf{x}, \mathbf{z}, \beta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]] - \beta D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

- β > 1 implies stronger disentanglement*

- Limitations:
  - Increased reconstruction loss
  - Increased complexity

Our Work (Mogultay, Kalkan, Vural, 2024)

- Learnable VAE

$$L_{L-VAE}(\theta, \phi; \mathbf{x}, \mathbf{z}) = \frac{1}{\sigma_0^2}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x}|\mathbf{z})\Big] - \frac{1}{\sigma_1^2}D_{KL}\Big(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\Big) + \sum_{i=0,1}\sigma_i^2$$

- Dimensionwise-learnable VAE

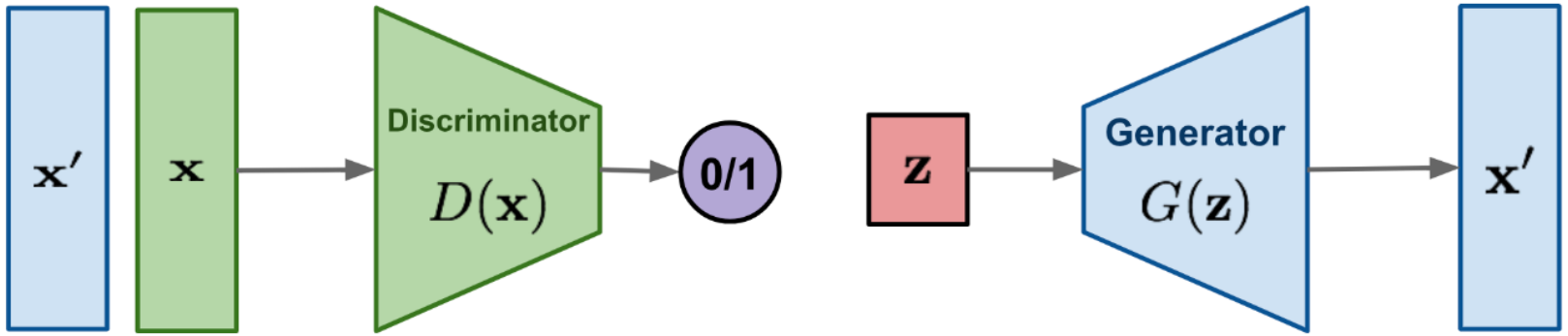
$$\begin{aligned} L_{dL-VAE}(\theta, \phi; \mathbf{x}, \mathbf{z}) = & \frac{1}{1+\ln(1+\sigma_0^2)}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x}|\mathbf{z})\Big] \\ & - \sum_{i=1}^{L+1}\frac{1}{1+\ln(1+\sigma_i^2)}D_{KL}\Big(q_\phi(\mathbf{z}_{i-1}|\mathbf{x}) \parallel p(\mathbf{z})\Big) \\ & + \sum_{i=0}^{L+1}\sigma_i^2. \end{aligned}$$

Our Work (Mogultay, Kalkan, Vural, 2024)

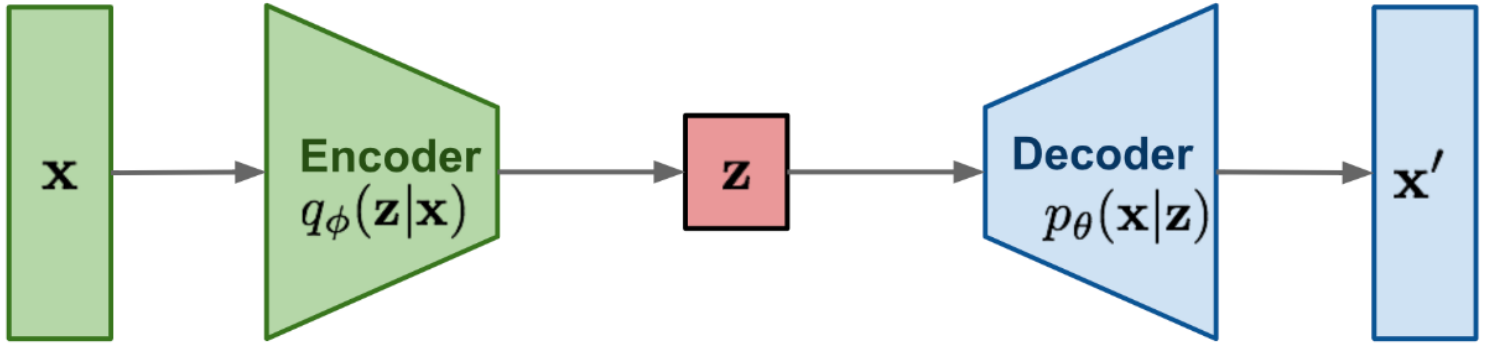
VAE	**11.86**	0.58	0.93	**0.77**	0.56	0.63	0.30	**0.32**	0.92	0.28
β-VAE ($\beta = 4$)	29.11	0.58	0.92	0.75	0.55	0.51	0.30	0.28	**0.94**	0.26
ControlVAE	24.35	**0.60**	**0.97**	0.76	**0.59**	0.58	0.30	0.30	**0.94**	0.30
DynamicVAE	33.75	0.56	0.89	0.58	0.50	0.51	**0.33**	0.29	0.85	0.32
σ-VAE	12.30	0.29	0.77	0.55	0.47	0.43	0.07	0.09	0.90	0.03
L-VAE ($\hat{\beta} = 0.89$)	**11.77**	0.59	0.96	**0.77**	0.57	**0.65**	0.31	**0.32**	0.92	0.32
dL-VAE	19.99	**0.60**	**0.97**	**0.77**	**0.59**	0.57	0.31	0.30	**0.94**	**0.33**

Flow-based Models

GAN: minimax the classification error loss.



VAE: maximize ELBO.



Flow-based generative models: minimize the negative log-likelihood

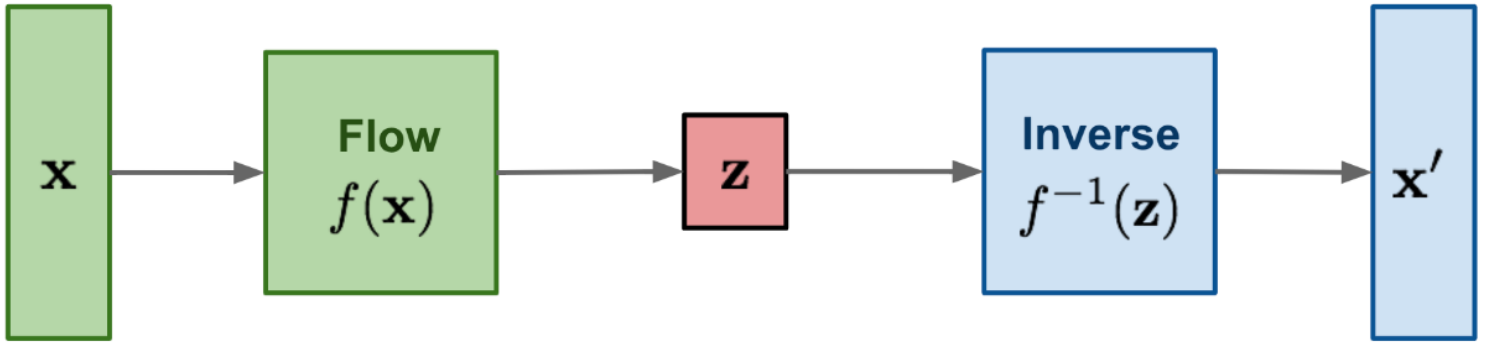


Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

Background

Given a random variable z and its known probability density function $z \sim \pi(z)$, we would like to construct a new random variable using a 1-1 mapping function $x = f(z)$. The function f is invertible, so $z = f^{-1}(x)$. Now the question is *how to infer the unknown probability density function of the new variable*, $p(x)$?

$$\int p(x)dx = \int \pi(z)dz = 1 \text{ ; Definition of probability distribution.}$$

$$p(x) = \pi(z)\left|\frac{dz}{dx}\right| = \pi(f^{-1}(x))\left|\frac{df^{-1}}{dx}\right| = \pi(f^{-1}(x))|(f^{-1})'(x)|$$

Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

Background

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

The multivariable version has a similar format:

$$\mathbf{z} \sim \pi(\mathbf{z}), \mathbf{x} = f(\mathbf{z}), \mathbf{z} = f^{-1}(\mathbf{x})$$

$$p(\mathbf{x}) = \pi(\mathbf{z}) \left| \det \frac{d\mathbf{z}}{d\mathbf{x}} \right| = \pi(f^{-1}(\mathbf{x})) \left| \det \frac{df^{-1}}{d\mathbf{x}} \right|$$

where $\det \frac{\partial f}{\partial \mathbf{z}}$ is the Jacobian determinant of the function f. The full proof of the multivariate

Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

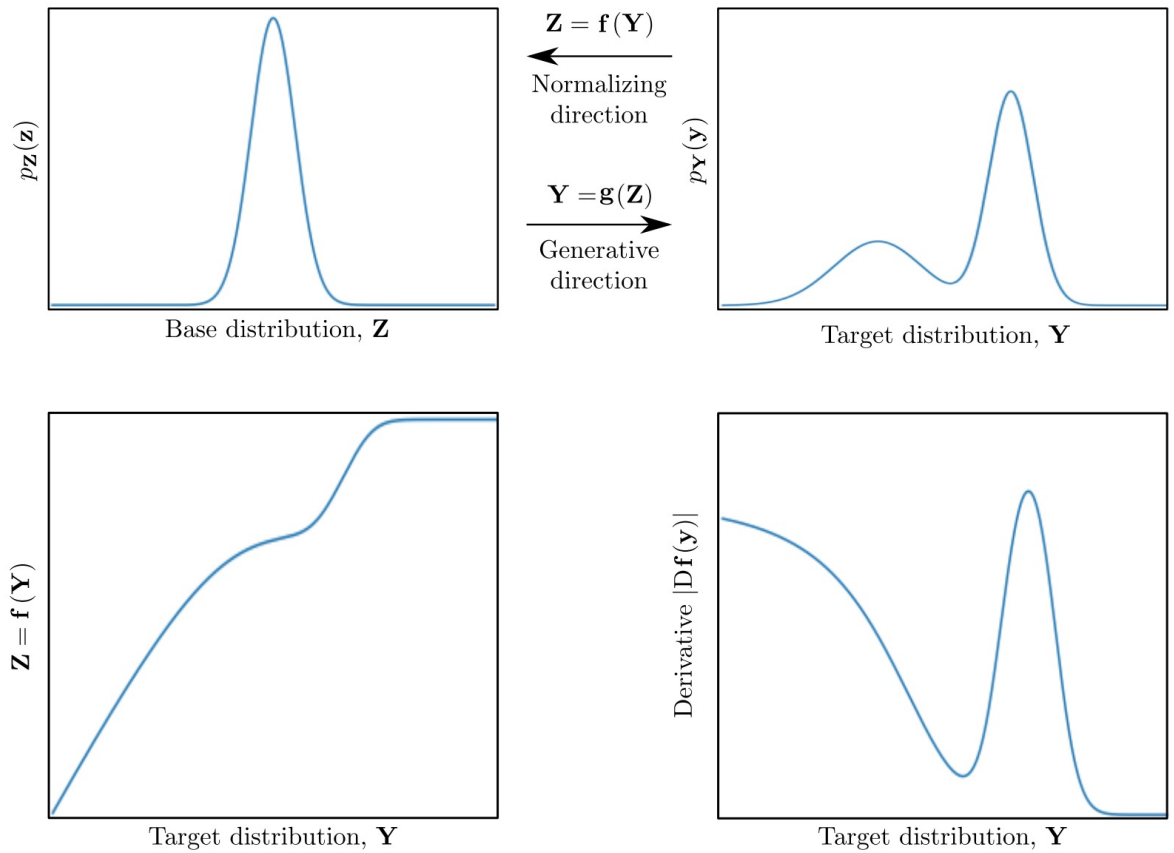


Fig. 1. Change of variables (Equation (1)). Top-left: the density of the source $p_{\mathbf{Z}}$. Top-right: the density function of the target distribution $p_{\mathbf{Y}}(\mathbf{y})$. There exists a bijective function $\mathbf{g}$, such that $p_{\mathbf{Y}} = \mathbf{g}_* p_{\mathbf{Z}}$, with inverse $\mathbf{f}$. Bottom-left: the inverse function $\mathbf{f}$. Bottom-right: the absolute Jacobian (derivative) of $\mathbf{f}$.

Figure: “Normalizing Flows: An Introduction and Review of Current Methods”, 2021.

Normalizing Flow

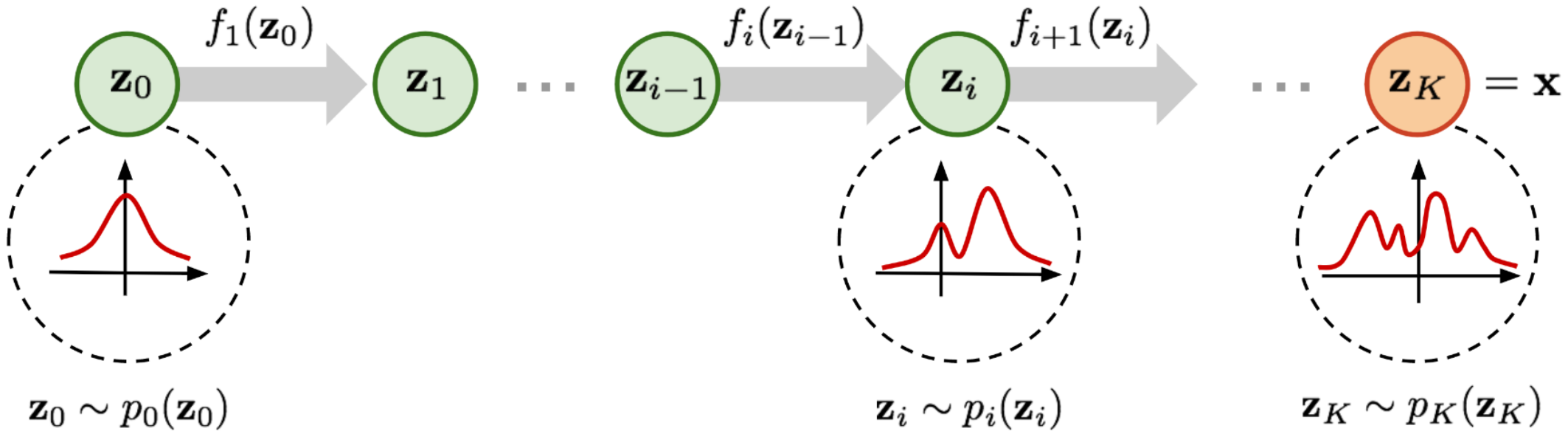


Fig. 2. Illustration of a normalizing flow model, transforming a simple distribution $p\_0(\mathbf{z}\_0)$ to a complex one $p\_K(\mathbf{z}\_K)$ step by step.

Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

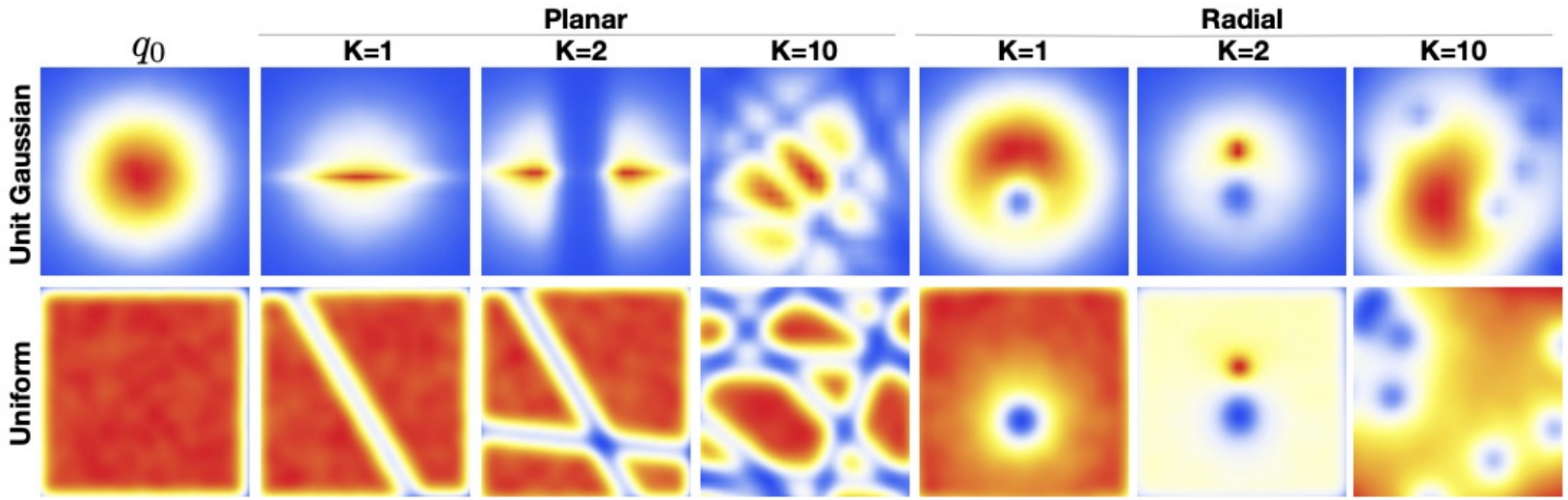


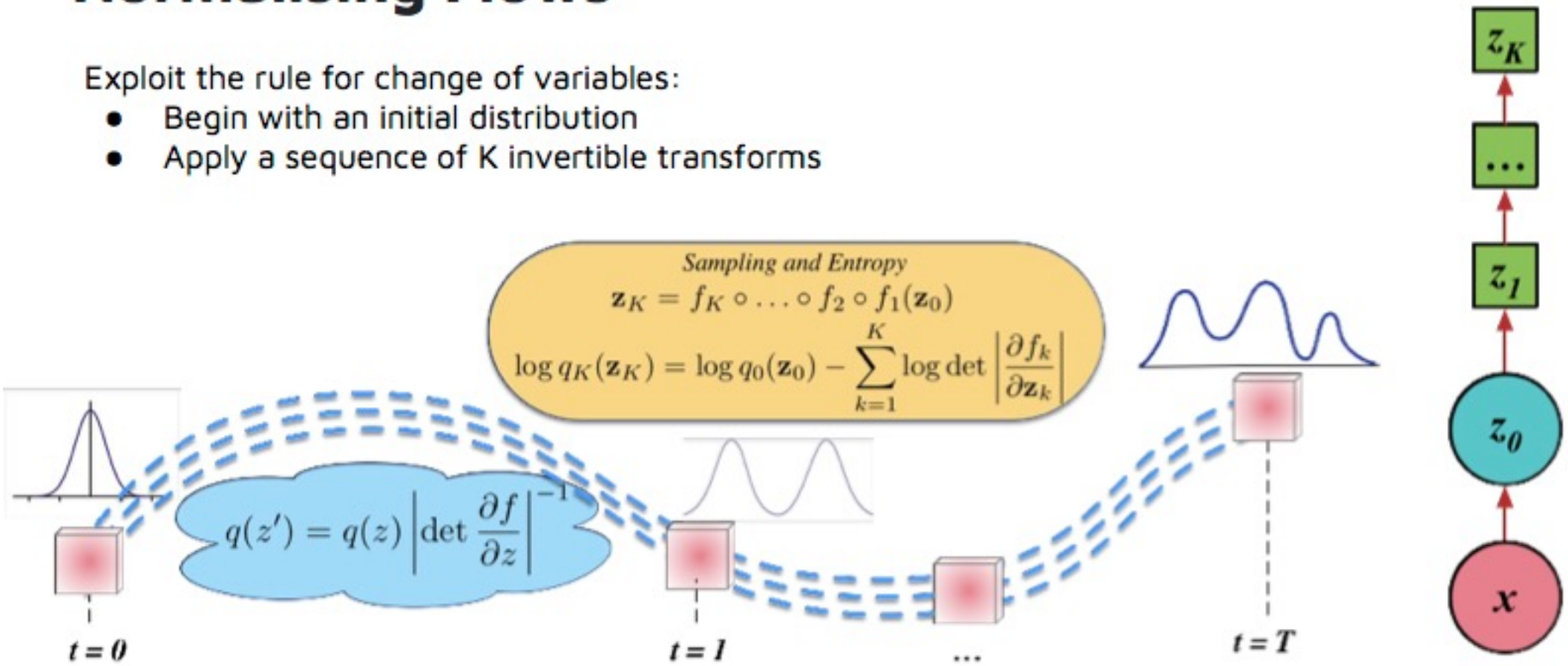
Figure 1. Effect of normalizing flow on two distributions.

Figure: ”Variational Inference with Normalizing Flows”, 2016.

Normalising Flows

Exploit the rule for change of variables:

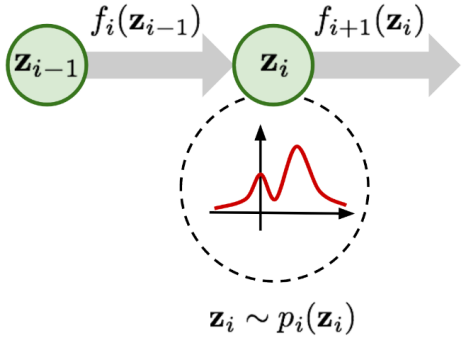
- Begin with an initial distribution
- Apply a sequence of K invertible transforms



Distribution flows through a sequence of invertible transforms

Rezende and Mohamed, 2015

$$\mathbf{z}_{i-1} \sim p_{i-1}(\mathbf{z}_{i-1})$$
$$\mathbf{z}_i = f_i(\mathbf{z}_{i-1}), \text{ thus } \mathbf{z}_{i-1} = f_i^{-1}(\mathbf{z}_i)$$
$$p_i(\mathbf{z}_i) = p_{i-1}(f_i^{-1}(\mathbf{z}_i)) \left| \det \frac{df_i^{-1}}{d\mathbf{z}_i} \right|$$



$$p(x) = \pi(z) \left| \frac{dz}{dx} \right| = \pi(f^{-1}(x)) \left| \frac{df^{-1}}{dx} \right|$$

Then let's convert the equation to be a function of $\mathbf{z}_i$ so that we can do inference with the base distribution.

$$p_i(\mathbf{z}_i) = p_{i-1}(f_i^{-1}(\mathbf{z}_i)) \left| \det \frac{df_i^{-1}}{d\mathbf{z}_i} \right|$$
$$= p_{i-1}(\mathbf{z}_{i-1}) \left| \det \left(\frac{df_i}{d\mathbf{z}_{i-1}} \right)^{-1} \right| \quad \text{; According to the inverse func theorem.}$$
$$= p_{i-1}(\mathbf{z}_{i-1}) \left| \det \frac{df_i}{d\mathbf{z}_{i-1}} \right|^{-1} \quad \text{; According to a property of Jacobians of invertible func.}$$
$$\log p_i(\mathbf{z}_i) = \log p_{i-1}(\mathbf{z}_{i-1}) - \log \left| \det \frac{df_i}{d\mathbf{z}_{i-1}} \right|$$

Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

Given such a chain of probability density functions, we know the relationship between each pair of consecutive variables. We can expand the equation of the output $\mathbf{x}$ step by step until tracing back to the initial distribution $\mathbf{z}_0$.

$$
\begin{aligned}
\mathbf{x} = \mathbf{z}_K &= f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{z}_0) \\
\log p(\mathbf{x}) = \log \pi_K(\mathbf{z}_K) &= \log \pi_{K-1}(\mathbf{z}_{K-1}) - \log \left| \det \frac{df_K}{d\mathbf{z}_{K-1}} \right| \\
&= \log \pi_{K-2}(\mathbf{z}_{K-2}) - \log \left| \det \frac{df_{K-1}}{d\mathbf{z}_{K-2}} \right| - \log \left| \det \frac{df_K}{d\mathbf{z}_{K-1}} \right| \\
&= \dots \\
&= \log \pi_0(\mathbf{z}_0) - \sum_{i=1}^{K} \log \left| \det \frac{df_i}{d\mathbf{z}_{i-1}} \right|
\end{aligned}
$$

Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

The path traversed by the random variables $\mathbf{z}_i = f_i(\mathbf{z}_{i-1})$ is the **flow** and the full chain formed by the successive distributions π_i is called a **normalizing flow**. Required by the computation in the equation, a transformation function f_i should satisfy two properties:

1. It is easily invertible.
2. Its Jacobian determinant is easy to compute.

Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

Training

Minimize the divergence between estimated distribution and real distribution:

$$D_{KL}[p^*(x)||p_\theta(x)] = -\mathbb{E}_{p^*(x)}[\log(p_\theta(x))] + \mathbb{E}_{p^*(x)}[\log(p^*(x))]$$

$$-\hat{\mathbb{E}}_{p^*(x)}[\log(p_\theta(x))] = -\frac{1}{N}\sum_{i=0}^{N}\log(p_\theta(x_i))$$

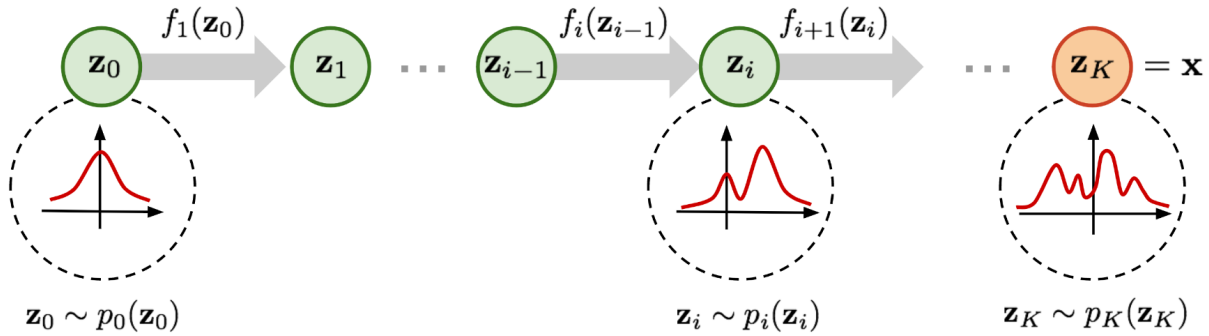
Constant for the dataset & does not depend on θ

$$\arg\min_\theta D_{KL}[p^*(x)||p_\theta(x)] \quad \Rightarrow \quad \arg\max_\theta \sum_{i=0}^{N}\log(p_\theta(x_i))$$

Training

$$\arg\max_{\theta} \sum_{i=0}^{N} \log(p_\theta(x_i))$$

Pseudo-code



1. $\mathbf{x} \leftarrow$ Sample a batch
2. $\mathbf{z}_0 \sim p_\theta(\mathbf{z}_0 \mid \mathbf{x})$
3. loss: $\log p_\theta(\mathbf{x}) = \log p_z(\mathbf{z}_0) + \log \left| \det \left(\frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$

Standard Normalizing Flow Training (MLE)

1. $\mathbf{x} \leftarrow$ **Sample a batch of real data.**

(Optional: If $\mathbf{x}$ *is discrete image data, apply dequantization here by adding uniform noise).*

2. $\mathbf{z}_0 \leftarrow f_\theta^{-1}(\mathbf{x})$

Run the data *backwards* through the flow's inverse transformations to find the exact corresponding latent variables.

3. $\log|\det J| \leftarrow \sum_{i=1}^{K} \log\left|\det \frac{\partial f_i^{-1}}{\partial \mathbf{z}_i}\right|$

While passing the data backwards, accumulate the log-determinant of the Jacobian for every layer. This measures how much the flow is "stretching" or "compressing" the probability space.

4. $\mathcal{L}(\theta) \leftarrow -\frac{1}{N}\sum_{\text{batch}} (\log p(\mathbf{z}_0) + \log|\det J|)$

Calculate the Negative Log-Likelihood (NLL) loss. We want to *maximize* the likelihood, so we *minimize* the negative log-likelihood.

- $\log p(\mathbf{z}_0)$ evaluates how likely that latent point is under your simple base distribution.
- $\log|\det J|$ adjusts that probability to account for the volume change caused by the flow.

5. $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$

Compute the gradients of the loss with respect to the model's parameters and update the weights using an optimizer (like Adam or SGD) with learning rate α.

RealNVP (Real-valued Non-Volume Preserving; Dinh et al., 2017)

The **RealNVP** (Real-valued Non-Volume Preserving; Dinh et al., 2017) model implements a normalizing flow by stacking a sequence of invertible bijective transformation functions. In each bijection $f : \mathbf{x} \mapsto \mathbf{y}$, known as *affine coupling layer*, the input dimensions are split into two parts:

- The first d dimensions stay same;
- The second part, $d+1$ to D dimensions, undergo an affine transformation ("scale-and-shift") and both the scale and shift parameters are functions of the first d dimensions.

$$\begin{aligned}\mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d})\end{aligned}$$

where $s(.)$ and $t(.)$ are *scale* and *translation* functions and both map $\mathbb{R}^d \mapsto \mathbb{R}^{D-d}$. The $\odot$ operation is the element-wise product.

Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

RealNVP (Real-valued Non-Volume Preserving; Dinh et al., 2017)

Now let's check whether this transformation satisfy two basic properties for a flow transformation.

Condition 1: "It is easily invertible."

Yes and it is fairly straightforward.

$$\begin{cases} \mathbf{y}_{1:d} & = \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} & = \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d}) \end{cases} \Leftrightarrow \begin{cases} \mathbf{x}_{1:d} & = \mathbf{y}_{1:d} \\ \mathbf{x}_{d+1:D} & = (\mathbf{y}_{d+1:D} - t(\mathbf{y}_{1:d})) \odot \exp(-s(\mathbf{y}_{1:d})) \end{cases}$$

Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

RealNVP (Real-valued Non-Volume Preserving; Dinh et al., 2017)

Condition 2: "Its Jacobian determinant is easy to compute."

Yes. It is not hard to get the Jacobian matrix and determinant of this transformation. The Jacobian is a lower triangular matrix.

$$\mathbf{J} = \begin{bmatrix} \mathbb{I}_d & \mathbf{0}_{d \times (D-d)} \\ \frac{\partial \mathbf{y}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \text{diag}(\exp(s(\mathbf{x}_{1:d}))) \end{bmatrix}$$

Hence the determinant is simply the product of terms on the diagonal.

$$\det(\mathbf{J}) = \prod_{j=1}^{D-d} \exp(s(\mathbf{x}_{1:d}))_j = \exp(\sum_{j=1}^{D-d} s(\mathbf{x}_{1:d})_j)$$

Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

Normalizing Flows

- Pros:
  - Successful results in estimating high-dimensional densities
  - Stable training compared to GANs
  - Easier to converge compared to GANs & VAEs

- Cons:
  - Latent space is not lower-dimensional than the input => may not be useful in some applications (e.g., image compression)
  - Fails in estimating the likelihood of out-of-distribution samples
  - Invertibility may not be guaranteed in practice due to numerical imprecision
  - Lower quality generation

Next Week

- (Deep) Generative Models
  - Autoregressive models
  - Variational AEs
  - Flow Models
  - Generative Adversarial Networks
  - Energy-based Models
  - Diffusion Models