

CENG501 – Deep Learning

Week 11

Fall 2024

Sinan Kalkan

Dept. of Computer Engineering, METU

Previously on ENGG501

Masked Autoencoders

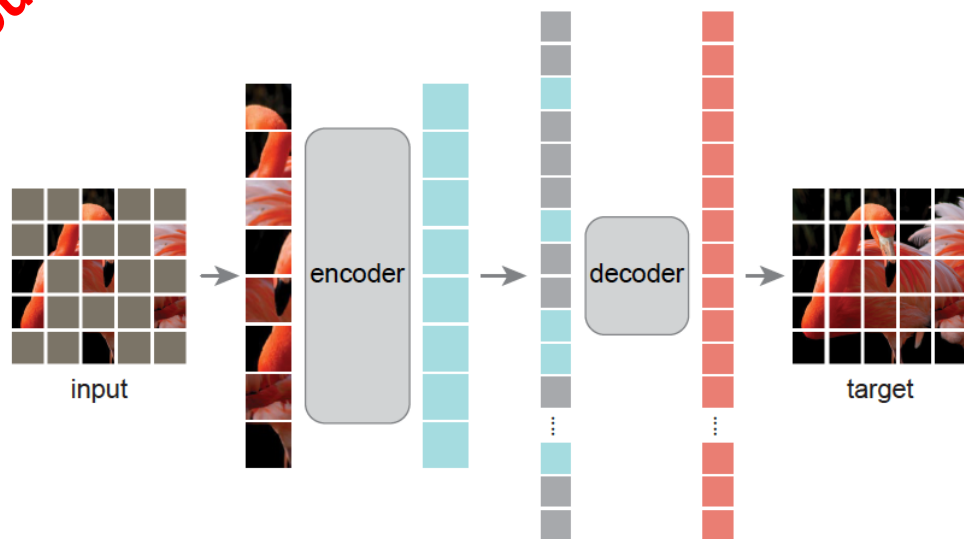


Figure 1. **Our MAE architecture.** During pre-training, a large random subset of image patches (e.g., 75%) is masked out. The encoder is applied to the small subset of *visible patches*. Mask tokens are introduced *after* the encoder, and the full set of encoded patches and mask tokens is processed by a small decoder that reconstructs the original image in pixels. After pre-training, the decoder is discarded and the encoder is applied to uncorrupted images (full sets of patches) for recognition tasks.

MSE loss between pixels for masked tokens only!

Masked Autoencoders Are Scalable Vision Learners

Kaiming He^{*,†} Xinlei Chen^{*} Saining Xie Yanghao Li Piotr Dollár Ross Girshick

^{*}equal technical contribution [†]project lead

Facebook AI Research (FAIR)

CVPR 2022

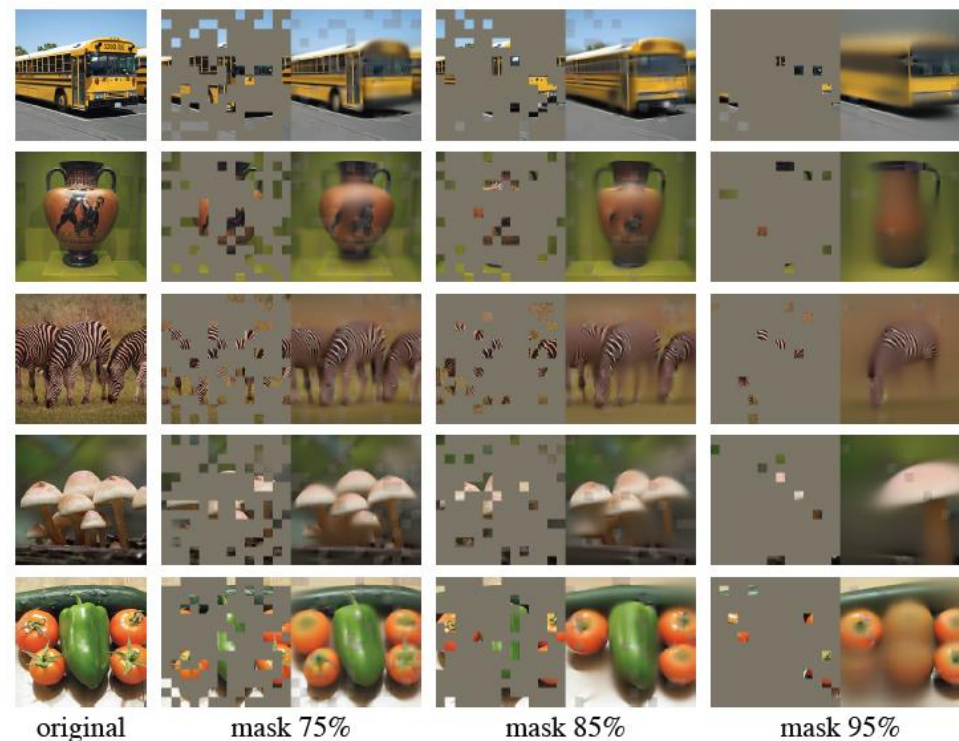


Figure 4. Reconstructions of ImageNet *validation* images using an MAE pre-trained with a masking ratio of 75% but applied on inputs with higher masking ratios. The predictions differ plausibly from the original images, showing that the method can generalize.

Previously on **SimMIM**
CVPR 2022

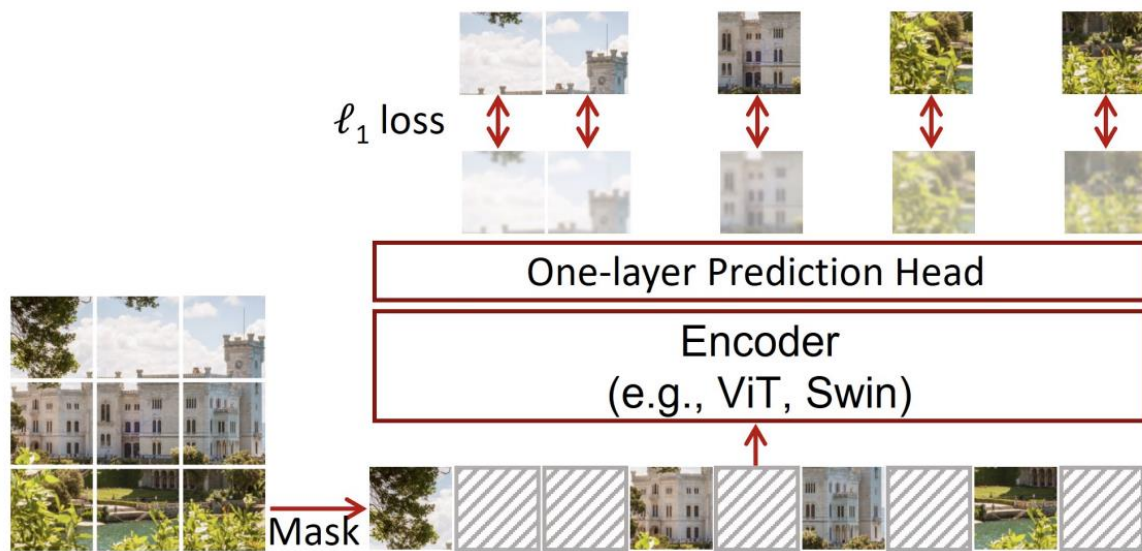


Figure 1. An illustration of our simple framework for masked language modeling, named *SimMIM*. It predicts raw pixel values of the randomly masked patches by a lightweight one-layer head, and performs learning using a simple ℓ_1 loss.

SimMIM: a Simple Framework for Masked Image Modeling

CVPR 2022

Zhenda Xie^{1*} Zheng Zhang^{2*} Yue Cao^{2*}

Yutong Lin³ Jianmin Bao² Zhuliang Yao¹ Qi Dai² Han Hu^{2*}

¹Tsinghua University ²Microsoft Research Asia ³Xi'an Jiaotong University

{t-zhxie, zhez, yuecao, t-yutonglin, jianmin.bao, t-zhuyao, qid, hanhu}@microsoft.com

Methods	Input Size	Fine-tuning	Linear eval	Pre-training
		Top-1 acc (%)	Top-1 acc (%)	costs
Sup. baseline [44]	224 ²	81.8	-	-
DINO [5]	224 ²	82.8	78.2	2.0×
MoCo v3 [9]	224 ²	83.2	76.7	1.8×
ViT [15]	384 ²	79.9	-	~4.0×
BEiT [1]	224 ²	83.2	56.7	1.5× [†]
Ours	224 ²	83.8	56.7	1.0×

Table 6. System-level comparison using ViT-B as the encoder. Training costs are counted in relative to our approach. [†] BEiT requires an additional stage to pre-train dVAE, which is not counted.

Önder Tuzcuoğlu^{1,3} Aybora Köksal^{1,3} Buğra Sofu⁴ Sinan Kalkan^{2,3} A. Aydın Alatan^{1,3}

¹ Dept. of Electrical and Electronics Eng. ² Dept. of Computer Eng.

³ Center for Image Analysis, Middle East Technical University, Ankara, Turkey

⁴ ROKETSAN Inc., Ankara, Turkey

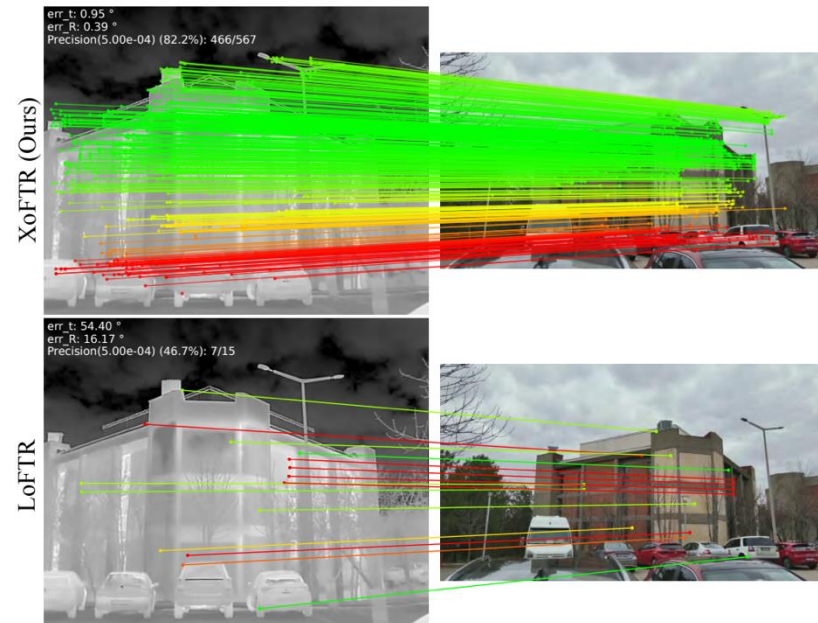


Figure 1. Our XoFTR provides significant improvements over LoFTR [69] on visible and thermal image pairs. Only the inlier matches after RANSAC are shown, and matches with epipolar error below 5×10^{-4} are drawn in green.

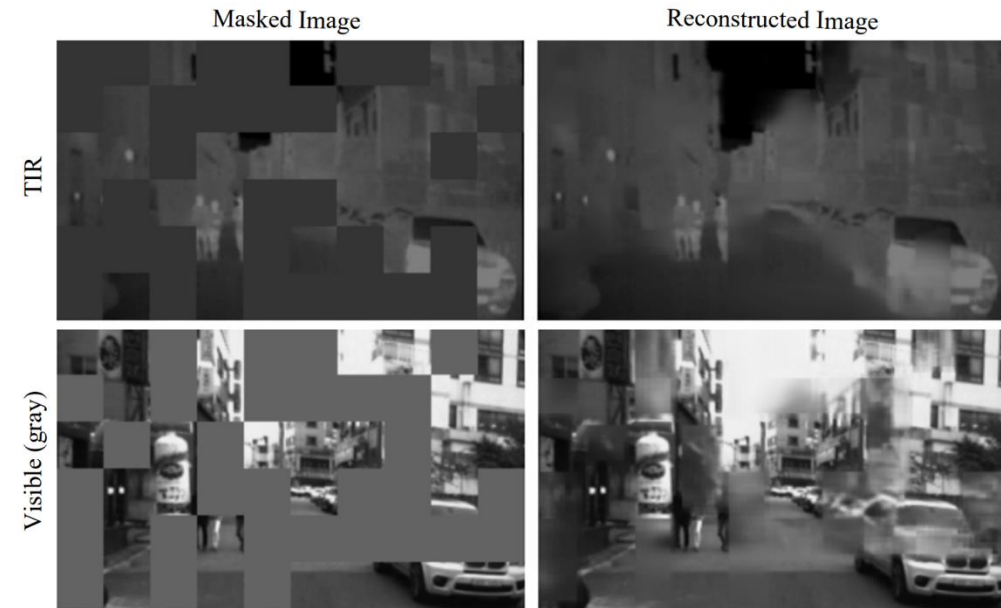


Figure 3. Visualization of reconstructed images using MIM pretext task. Input images are from [39].

Previously on CVENG501

DINO v1 & v2 (ICCV'21 & TMLR'24)

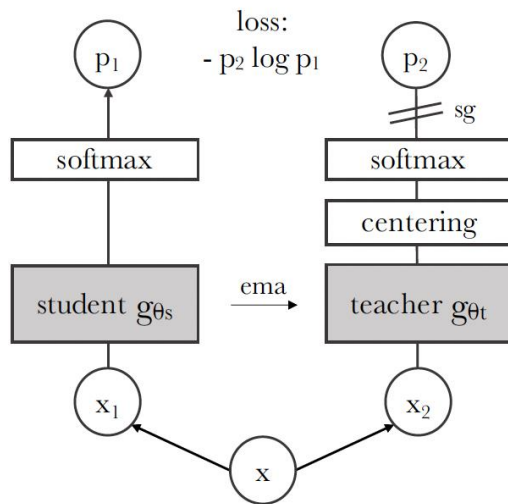


Figure 2: **Self-distillation with no labels.** We illustrate DINO in the case of one single pair of views (x_1, x_2) for simplicity. The model passes two different random transformations of an input image to the student and teacher networks. Both networks have the same architecture but different parameters. The output of the teacher network is centered with a mean computed over the batch. Each networks outputs a K dimensional feature that is normalized with a temperature softmax over the feature dimension. Their similarity is then measured with a cross-entropy loss. We apply a stop-gradient (sg) operator on the teacher to propagate gradients only through the student. The teacher parameters are updated with an exponential moving average (ema) of the student parameters.

Algorithm 1 DINO PyTorch pseudocode w/o multi-crop.

```
# gs, gt: student and teacher networks
# C: center (K)
# tps, tpt: student and teacher temperatures
# l, m: network and center momentum rates
gt.params = gs.params
for x in loader: # load a minibatch x with n samples
    x1, x2 = augment(x), augment(x) # random views

    s1, s2 = gs(x1), gs(x2) # student output n-by-K
    t1, t2 = gt(x1), gt(x2) # teacher output n-by-K

    loss = H(t1, s2)/2 + H(t2, s1)/2
    loss.backward() # back-propagate

    # student, teacher and center updates
    update(gs) # SGD
    gt.params = l*gt.params + (1-l)*gs.params
    C = m*C + (1-m)*cat([t1, t2]).mean(dim=0)

def H(t, s):
    t = t.detach() # stop gradient
    s = softmax(s / tps, dim=1)
    t = softmax((t - C) / tpt, dim=1) # center + sharpen
    return - (t * log(s)).sum(dim=1).mean()
```

Previously on ~~CVPR~~ **CVG501**
VideoMAE

Zhan Tong^{1,2*} Yibing Song² Jue Wang² Limin Wang^{1,3†}

¹State Key Laboratory for Novel Software Technology, Nanjing University

²Tencent AI Lab ³Shanghai AI Lab

tongzhan@smail.nju.edu.cn {yibingsong.cv, arphid}@gmail.com lmwang@nju.edu.cn

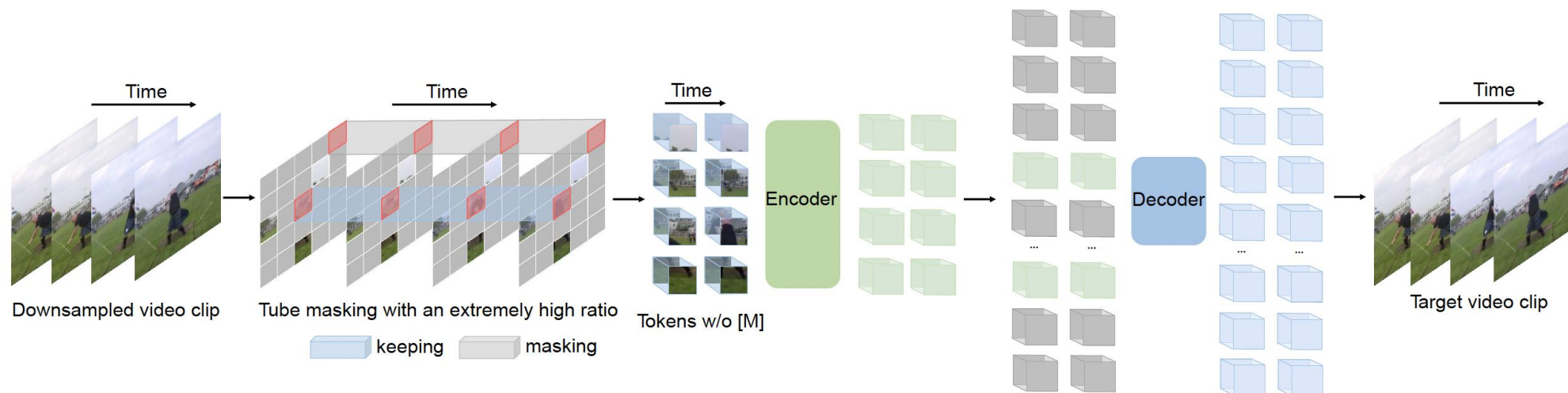


Figure 1: **VideoMAE** performs the task of masking random cubes and reconstructing the missing ones with an asymmetric encoder-decoder architecture. Due to high redundancy and temporal correlation in videos, we present the customized design of tube masking with an extremely high ratio (90% to 95%). This simple design enables us to create a more challenging and meaningful self-supervised task to make the learned representations capture more useful spatiotemporal structures.

Vision-Language Models: Overview

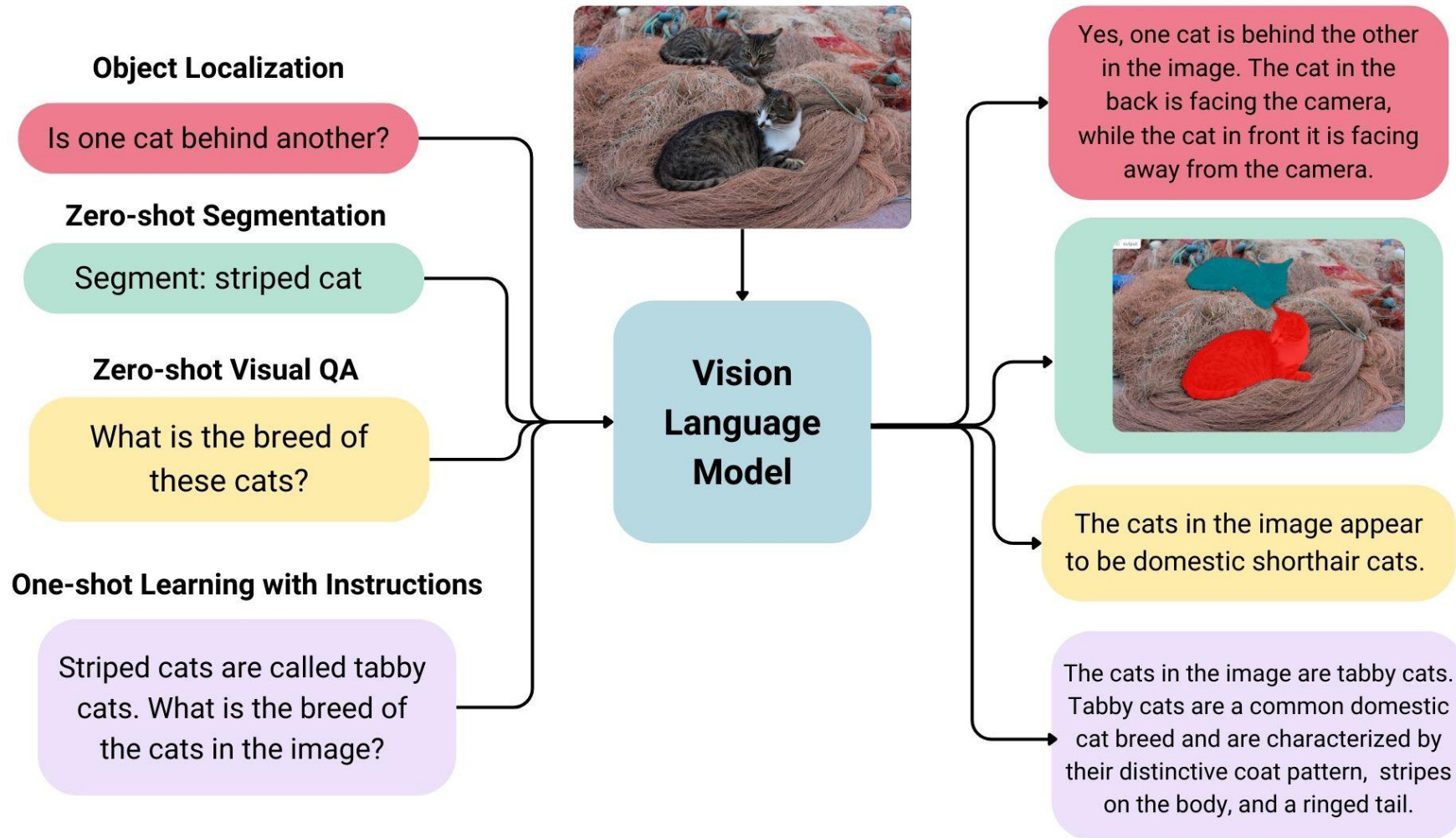


Fig: <https://huggingface.co/blog/vlms>

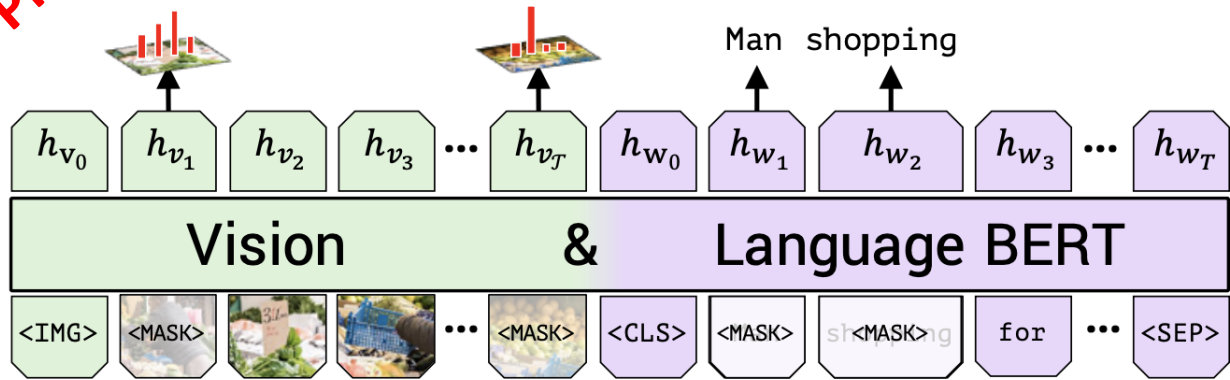
Previously on CENG501

Earlier Attempts:

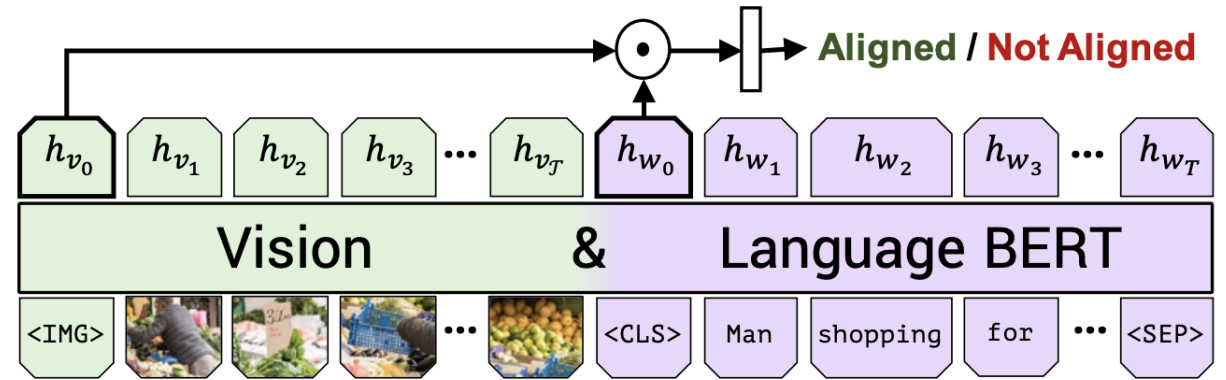
ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks



2019



(a) Masked multi-modal learning



(b) Multi-modal alignment prediction

Figure 3: We train ViLBERT on the Conceptual Captions [24] dataset under two training tasks to learn visual grounding. In masked multi-modal learning, the model must reconstruct image region categories or words for masked inputs given the observed inputs. In multi-modal alignment prediction, the model must predict whether or not the caption describes the image content.

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



A person hits a ball with a tennis racket

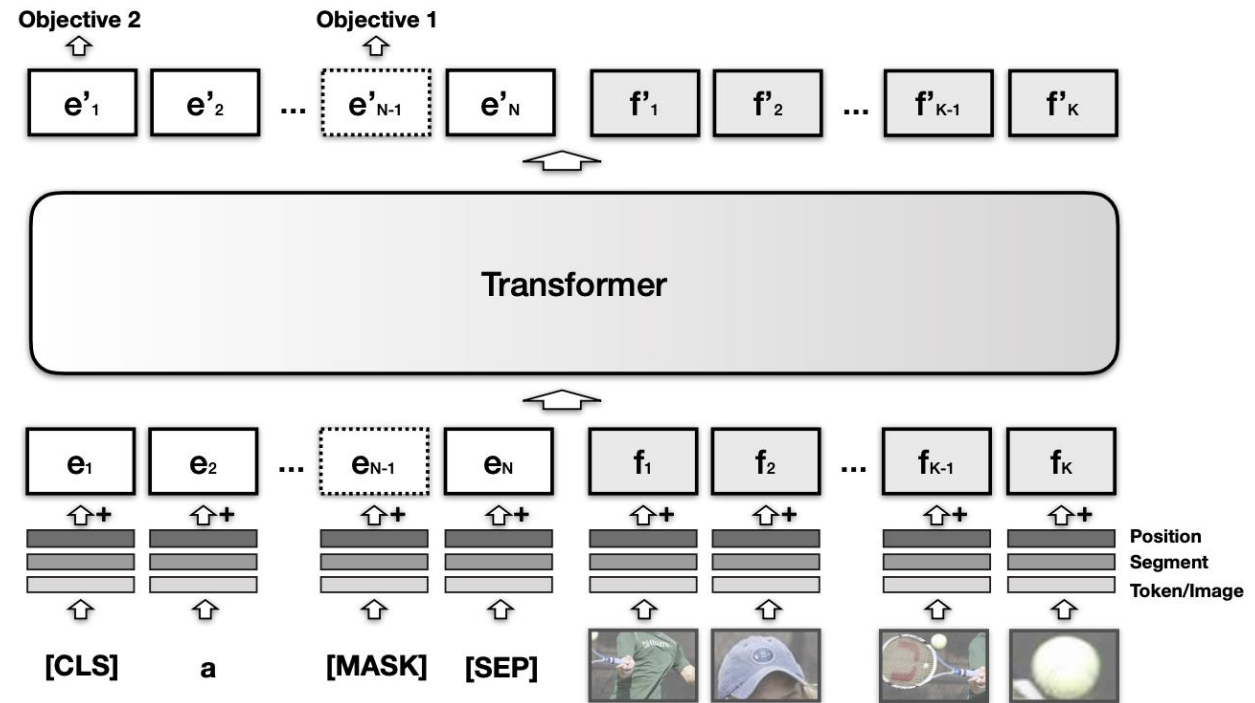
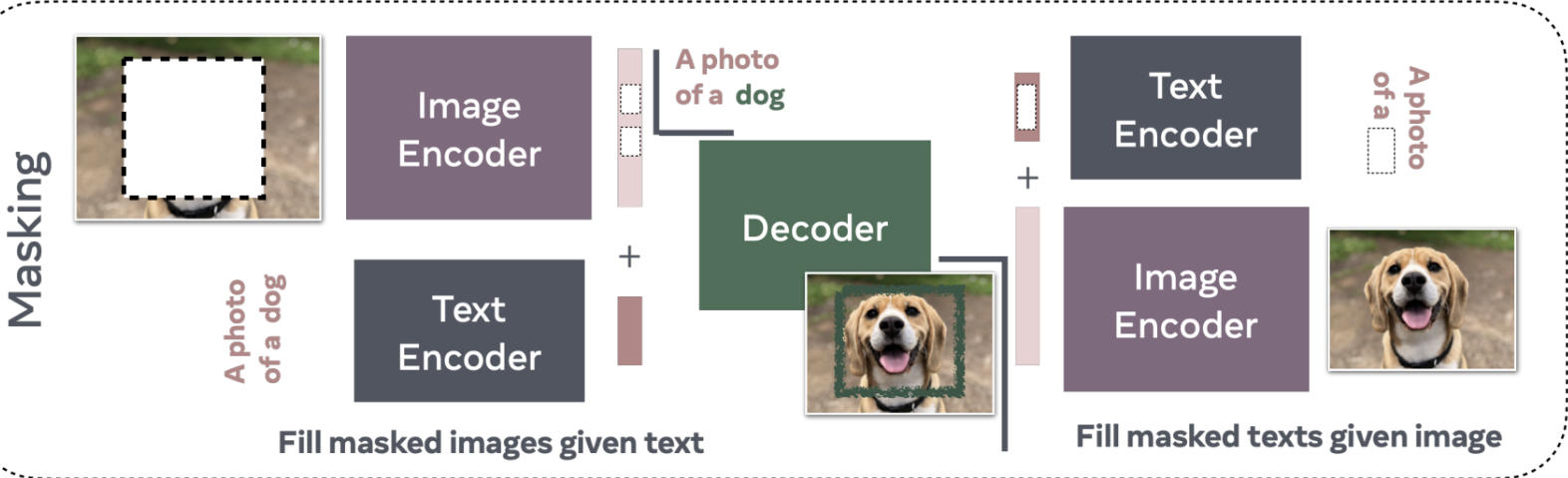
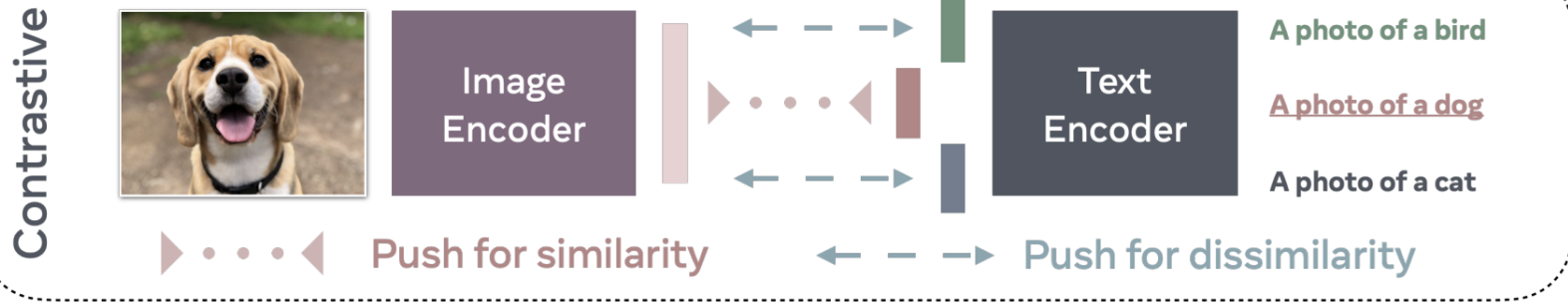


Figure 2: The architecture of VisualBERT. Image regions and language are combined with a Transformer to allow the self-attention to discover implicit alignments between language and vision. It is pre-trained with a masked language modeling (Objective 1), and sentence-image prediction task (Objective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

Previously Overview



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

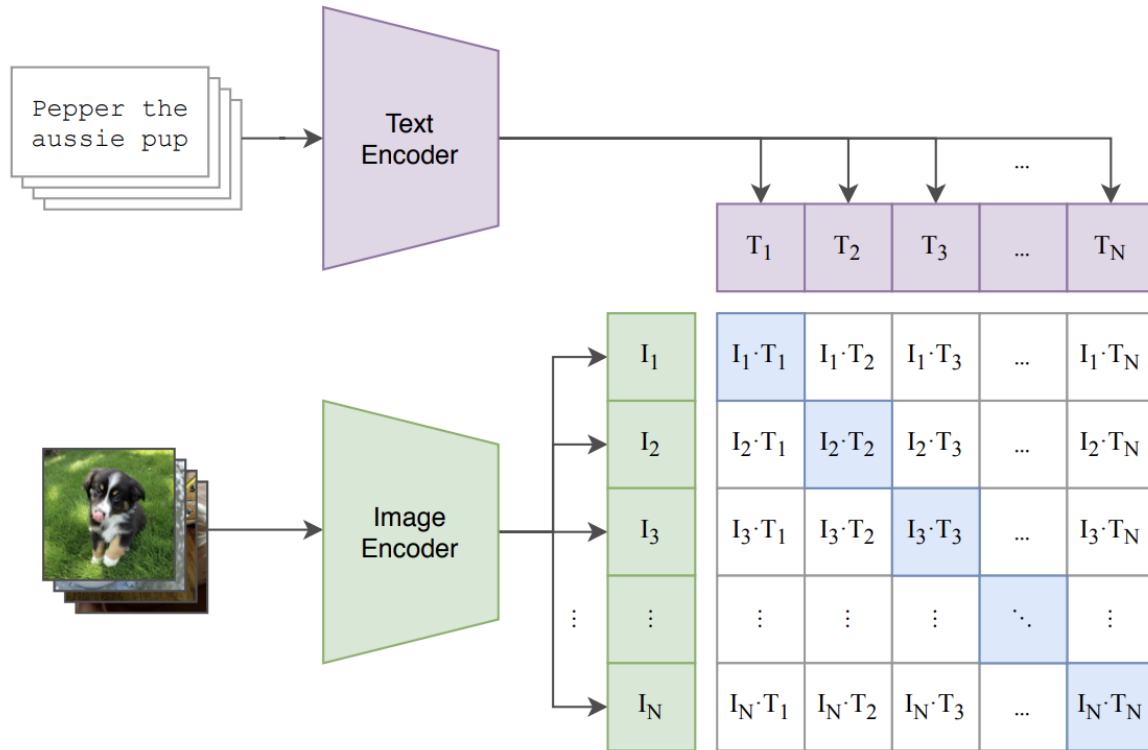
Previously on CS-ENG501

Learning Transferable Visual Models From Natural Language Supervision

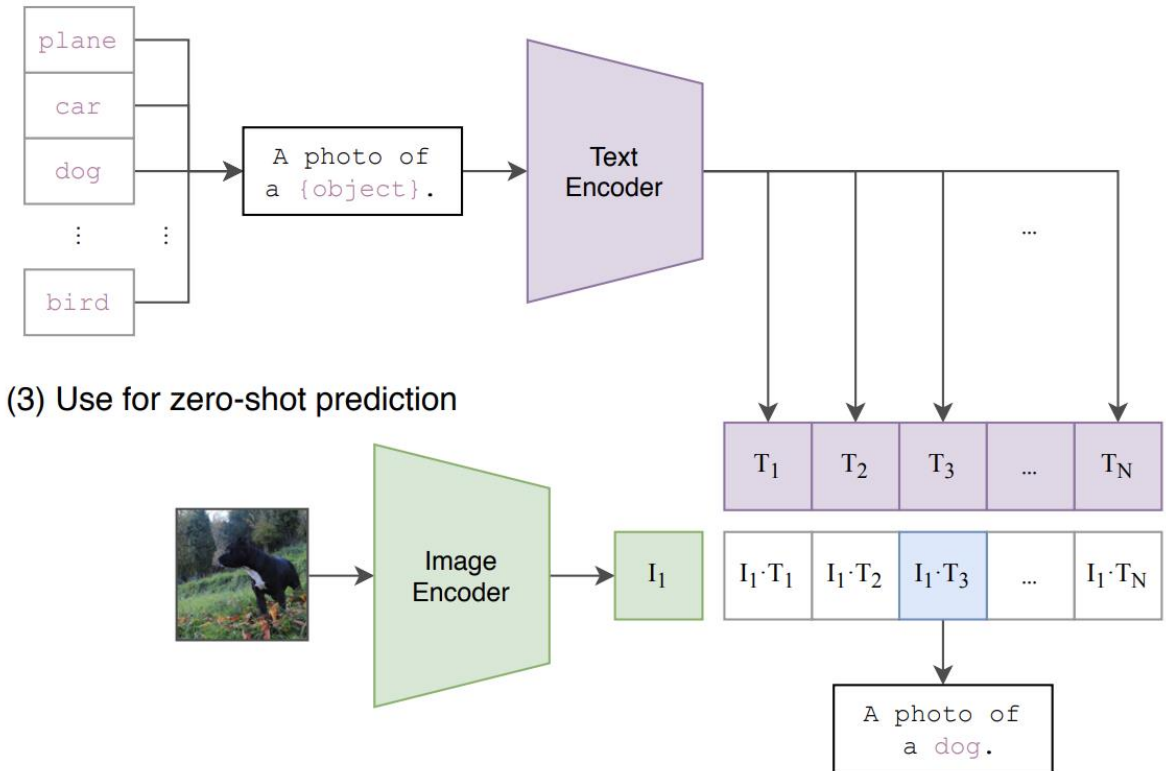
Alec Radford^{*1} Jong Wook Kim^{*1} Chris Hallacy¹ Aditya Ramesh¹ Gabriel Goh¹ Sandhini Agarwal¹
Girish Sastry¹ Amanda Askell¹ Pamela Mishkin¹ Jack Clark¹ Gretchen Krueger¹ Ilya Sutskever¹

2021

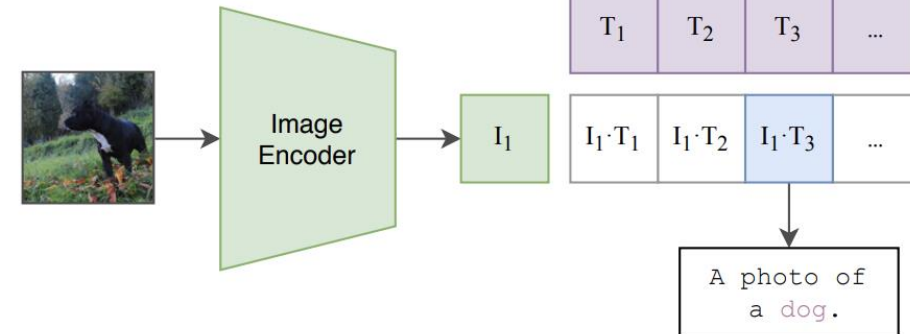
(1) Contrastive pre-training



(2) Create dataset classifier from label text



(3) Use for zero-shot prediction



CLIP Results

Previously on CENG 504

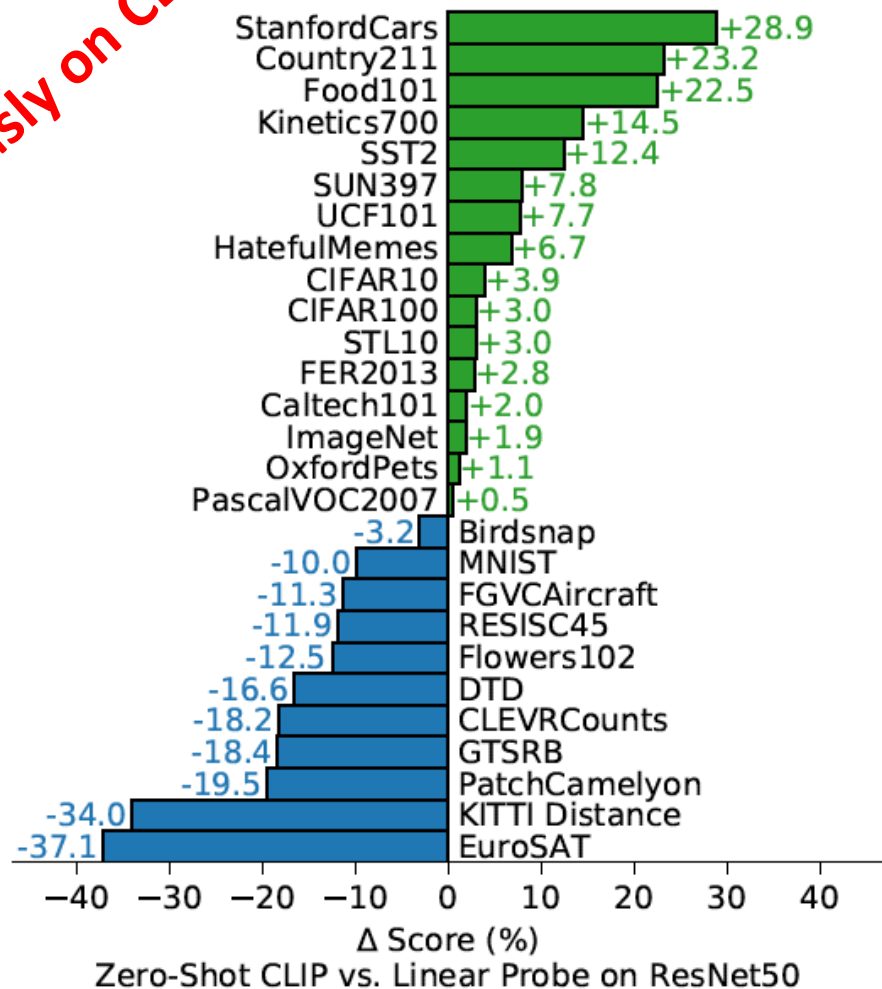


Figure 5. Zero-shot CLIP is competitive with a fully supervised baseline. Across a 27 dataset eval suite, a zero-shot CLIP classifier outperforms a fully supervised linear classifier fitted on ResNet-50 features on 16 datasets, including ImageNet.

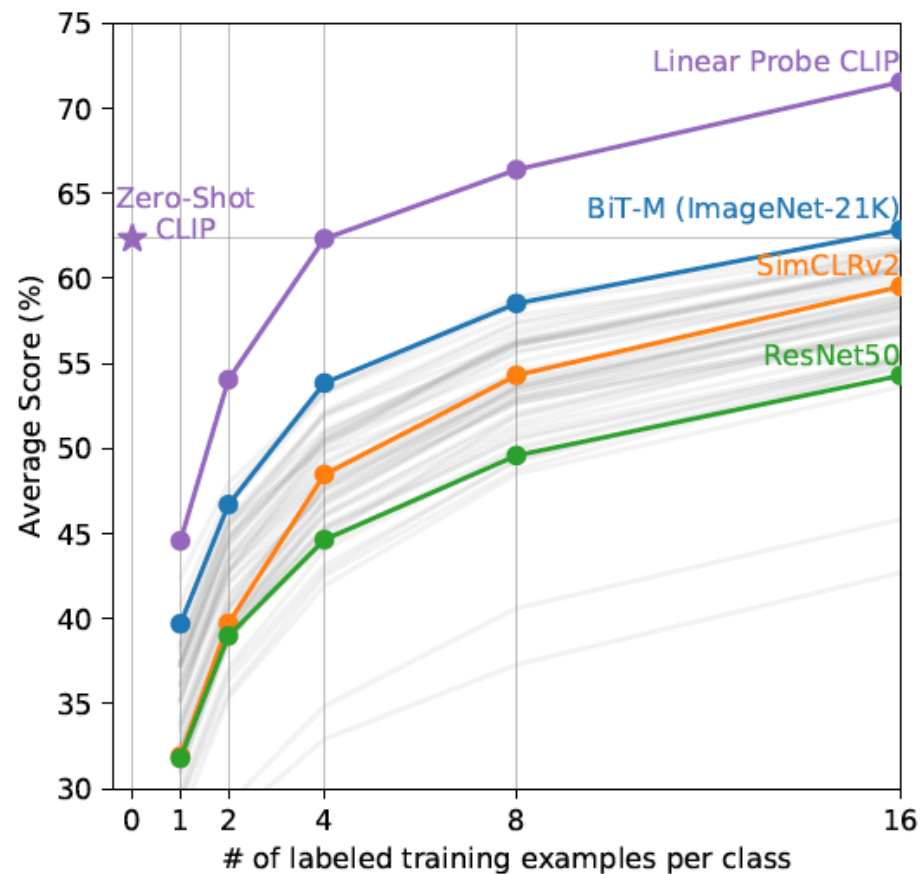


Figure 6. Zero-shot CLIP outperforms few-shot linear probes. Zero-shot CLIP matches the average performance of a 4-shot linear classifier trained on the same feature space and nearly matches the best results of a 16-shot linear classifier across publicly available models. For both BiT-M and SimCLRv2, the best performing model is highlighted. Light gray lines are other models in the eval suite. The 20 datasets with at least 16 examples per class were used in this analysis.

Previously on FLAVENG501

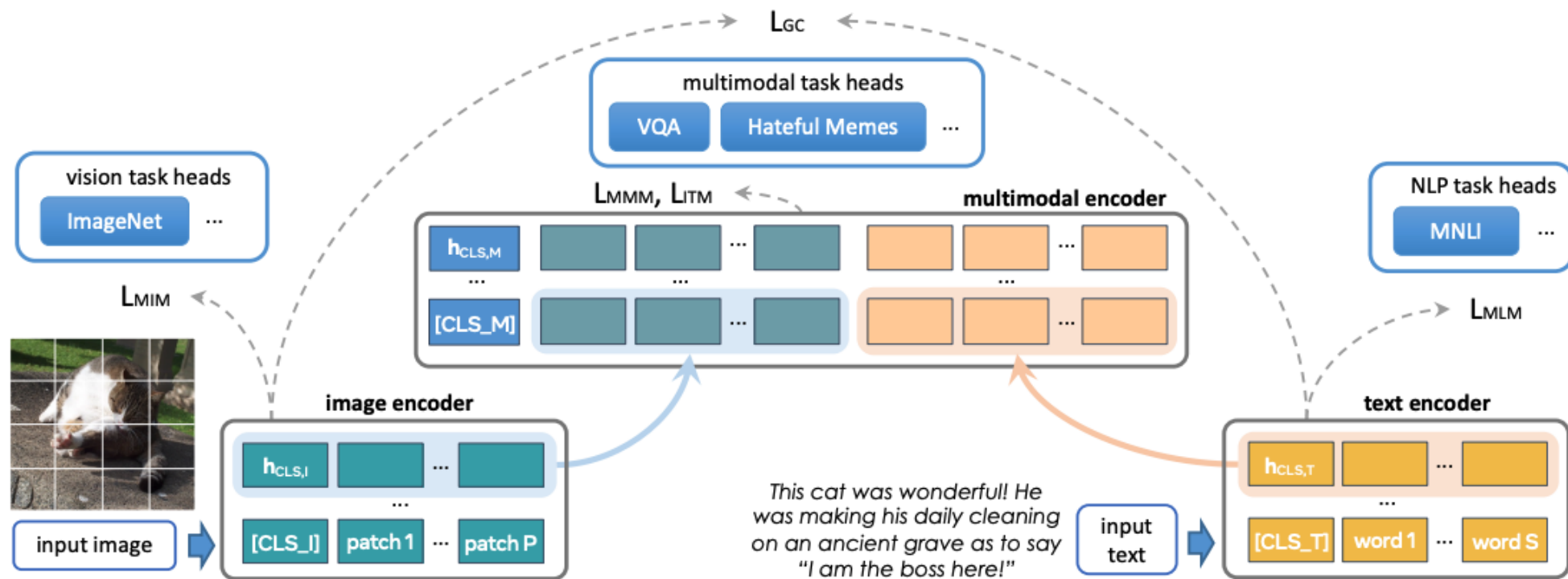
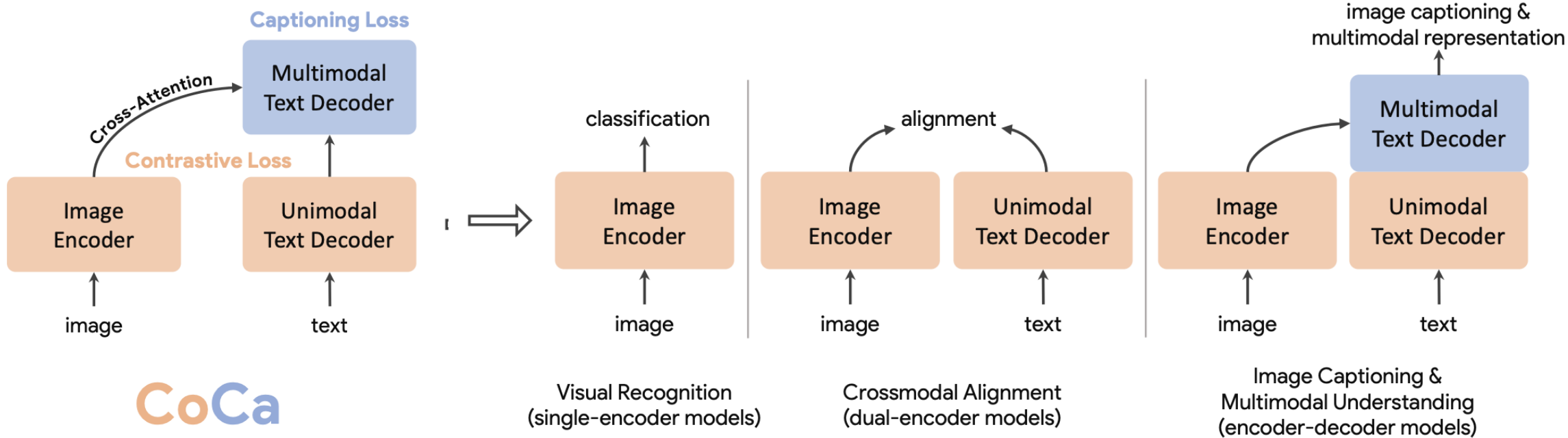


Figure 2. **An overview of our FLAVA model**, with an image encoder transformer to capture unimodal image representations, a text encoder transformer to process unimodal text information, and a multimodal encoder transformer that takes as input the encoded unimodal image and text and integrates their 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.

Previously on CoCa

CoCa

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



Previously on **Chameleon**

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

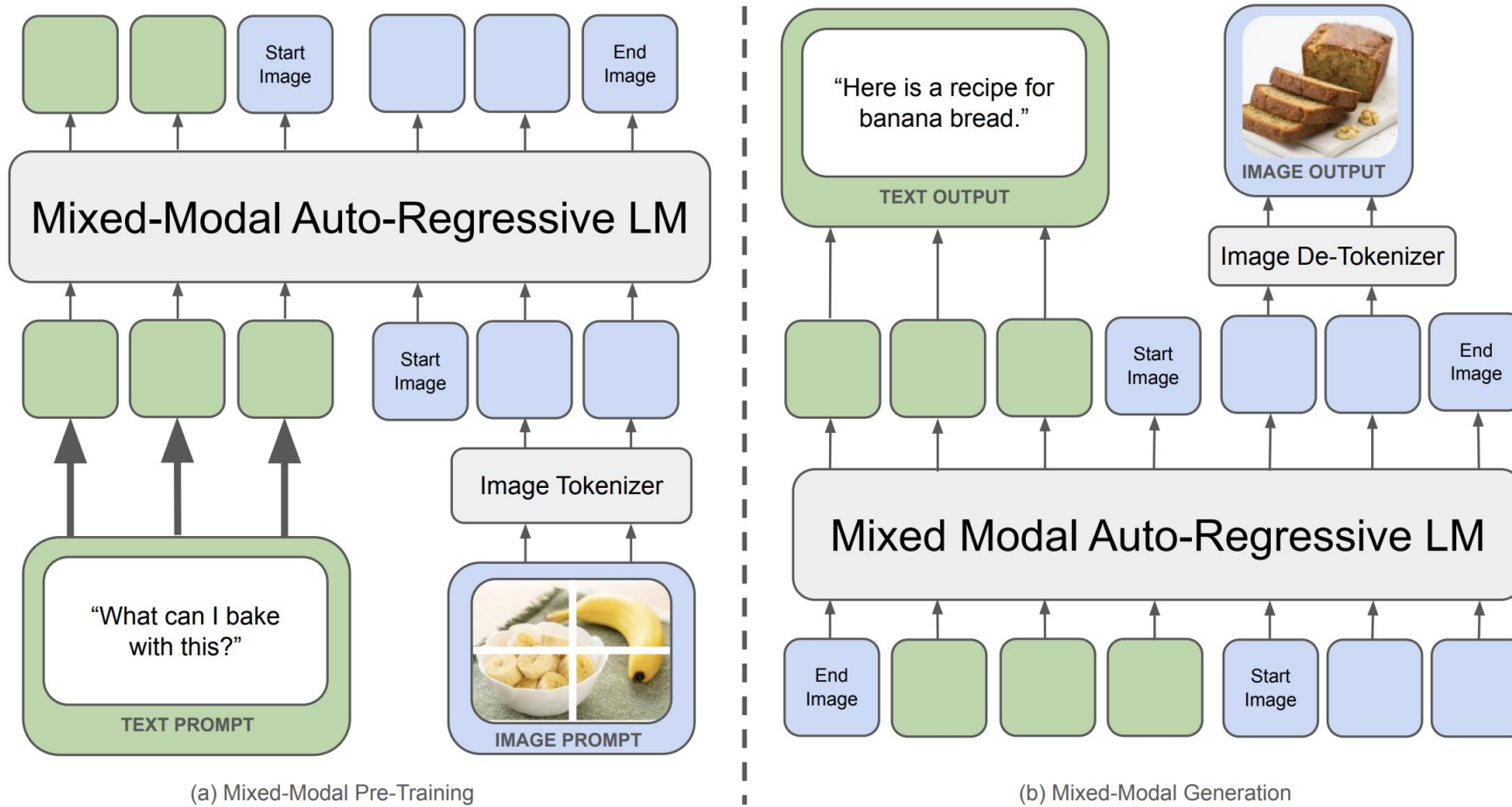


Figure 1 Chameleon represents all modalities — images, text, and code, as discrete tokens and uses a uniform transformer-based architecture that is trained from scratch in an end-to-end fashion on $\sim 10^7$ tokens of interleaved mixed-modal data. As a result, Chameleon can both reason over, as well as generate, arbitrary mixed-modal documents. Text tokens are represented in green and image tokens are represented in blue.

2 Pre-Training

Chameleon represents images, in addition to text, as a series of discrete tokens and takes advantage of the scaling properties of auto-regressive Transformers (Ramesh et al., 2021; Aghajanyan et al., 2022, 2023; Yu et al., 2023). We present any ordering of images and text during training ranging from text-only, to single text/image pairs to full interleaved text-image documents.



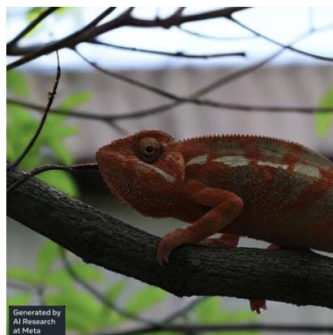
Prompt: `` While chameleons are masters of camouflage in the natural world, let's see how they might fare in a man-made setting.

1. Analyze the colors, patterns, and textures in this image and identify the most challenging spot for a chameleon to blend in.
2. Then generate an image of a chameleon with the optimal pattern to camouflage itself.
3. Then, rate and explain how successful each attempt is on a scale from 1-10.

Response: The image is of a stop sign in a forested area, with some sunlight filtering through the trees. The sign has a red octagonal shape with a white border and the word "STOP" written in white letters. In the background, there are green leaves and trees.

Challenging Spot for a Chameleon: The stop sign would be the most challenging spot for a chameleon to blend in due to its bright red color and bold white letters, which would stand out in the forested environment.

Optimal Pattern for Camouflage: To camouflage itself on the stop sign, a chameleon would need to have a pattern that would break up its outline and make it less conspicuous. A pattern with shades



of brown and red would help the chameleon blend in more effectively. Additionally, a mottled or dappled pattern would help to scatter light and further reduce the chameleon's visibility.

``

Rating and Explanation: A rating of 2/10 would be given for the chameleon's ability to blend in on the stop sign. While the chameleon's green and brown pattern helps to break up its outline, the bright red color of the stop sign will still make it stand out in the forested environment.

Previously on ENG501
Frozen

Multimodal Few-Shot Learning with Frozen Language Models, 2021

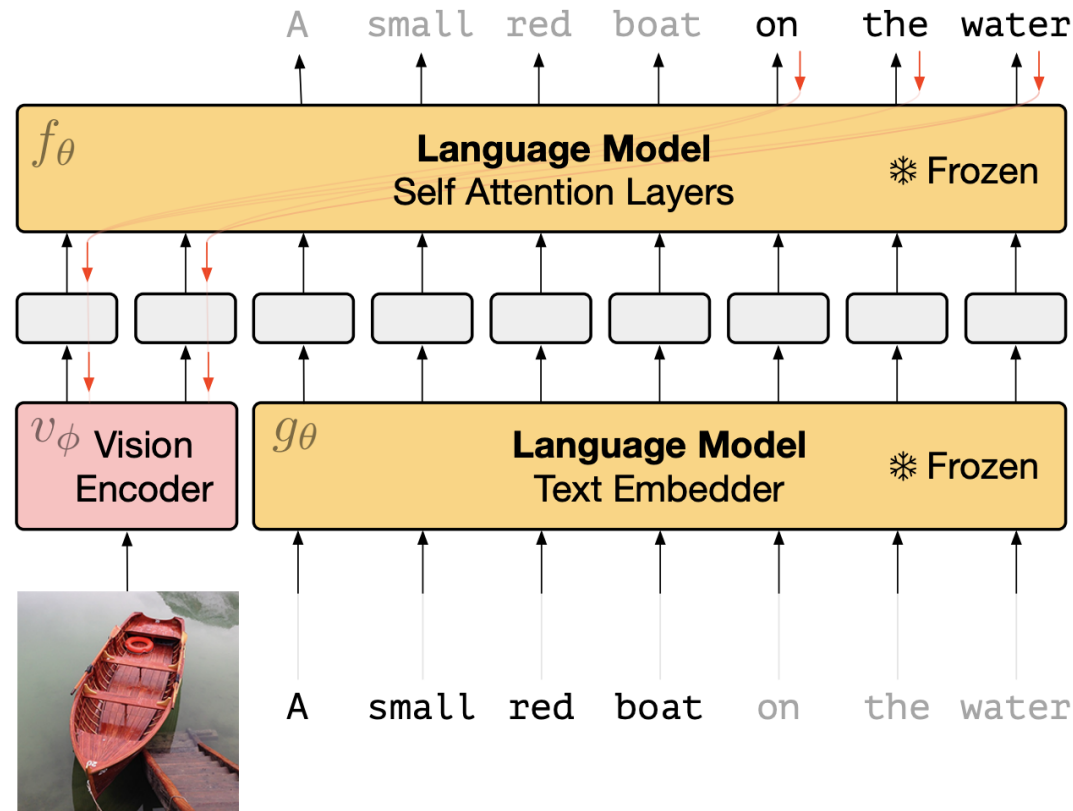


Figure 2: Gradients through a frozen language model's self attention layers are used to train the vision encoder.

Previously on FLAN501
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.

LLM: Chinchilla

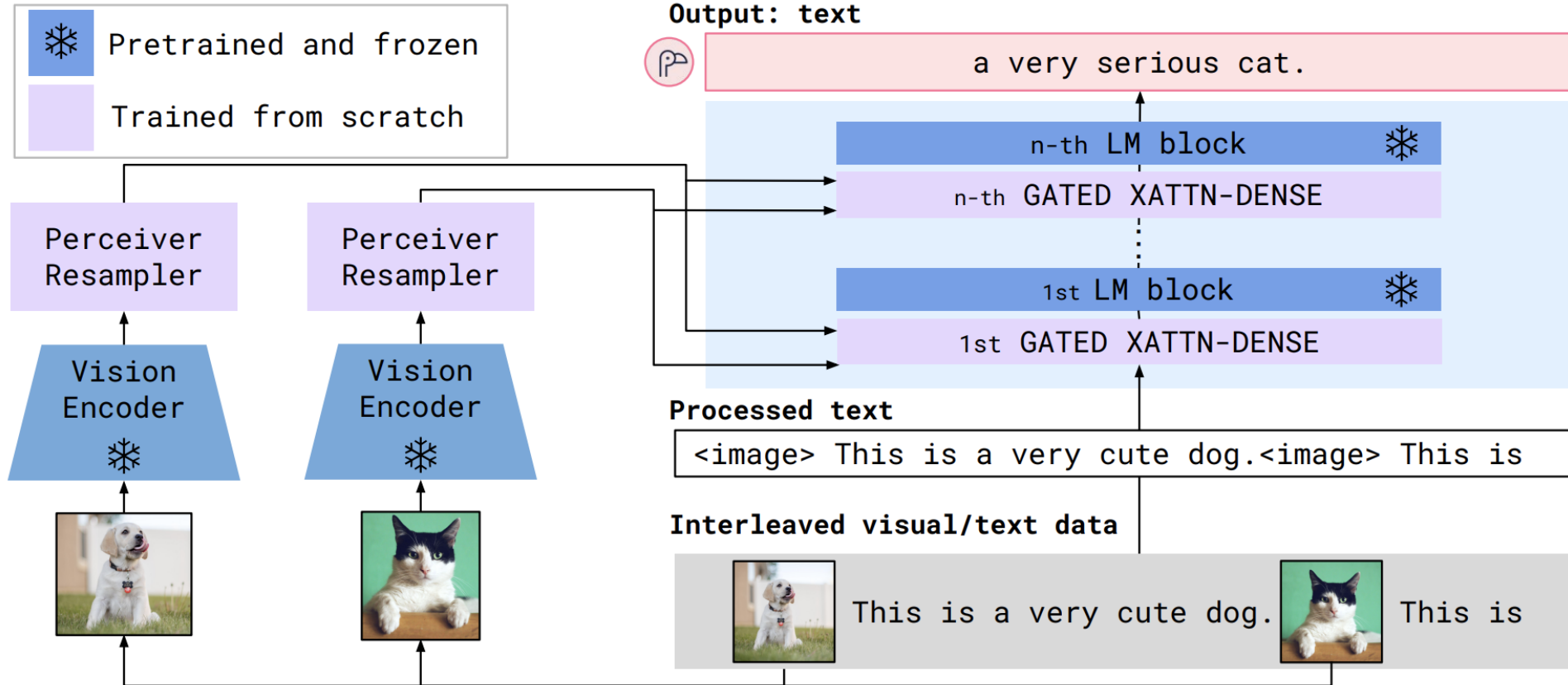


Figure 3: **Flamingo architecture overview.** Flamingo is a family of visual language models (VLMs) that take as input visual data interleaved with text and produce free-form text as output.

BLIP-2

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

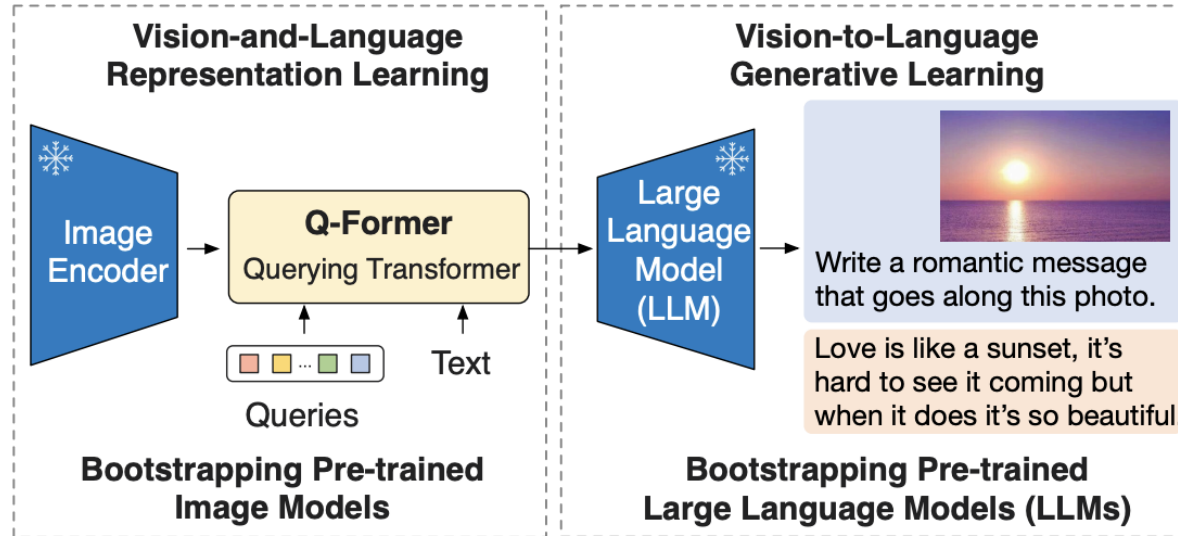


Figure 1. Overview of BLIP-2’s framework. We pre-train a lightweight Querying Transformer following a two-stage strategy to bridge the modality gap. The first stage bootstraps vision-language representation learning from a frozen image encoder. The second stage bootstraps vision-to-language generative learning from a frozen LLM, which enables zero-shot instructed image-to-text generation (see Figure 4 for more examples).

Previously on FLN-2
BLIP-2

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

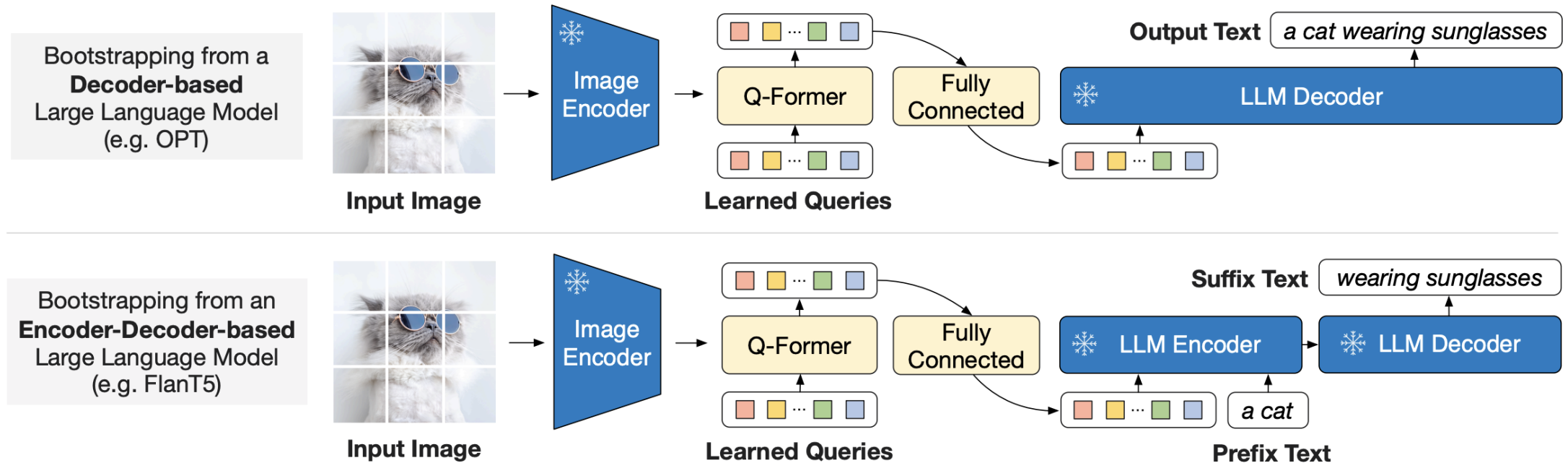


Figure 3. BLIP-2’s second-stage vision-to-language generative pre-training, which bootstraps from frozen large language models (LLMs). **(Top)** Bootstrapping a decoder-based LLM (e.g. OPT). **(Bottom)** Bootstrapping an encoder-decoder-based LLM (e.g. FlanT5). The fully-connected layer adapts from the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

Previously on CENG501

Segment Anything Model (SAM) 2023

Segment Anything

Alexander Kirillov^{1,2,4} Eric Mintun² Nikhila Ravi^{1,2} Hanzi Mao² Chloe Rolland³ Laura Gustafson³
Tete Xiao³ Spencer Whitehead Alexander C. Berg Wan-Yen Lo Piotr Dollár⁴ Ross Girshick⁴
¹project lead ²joint first author ³equal contribution ⁴directional lead
Meta AI Research, FAIR

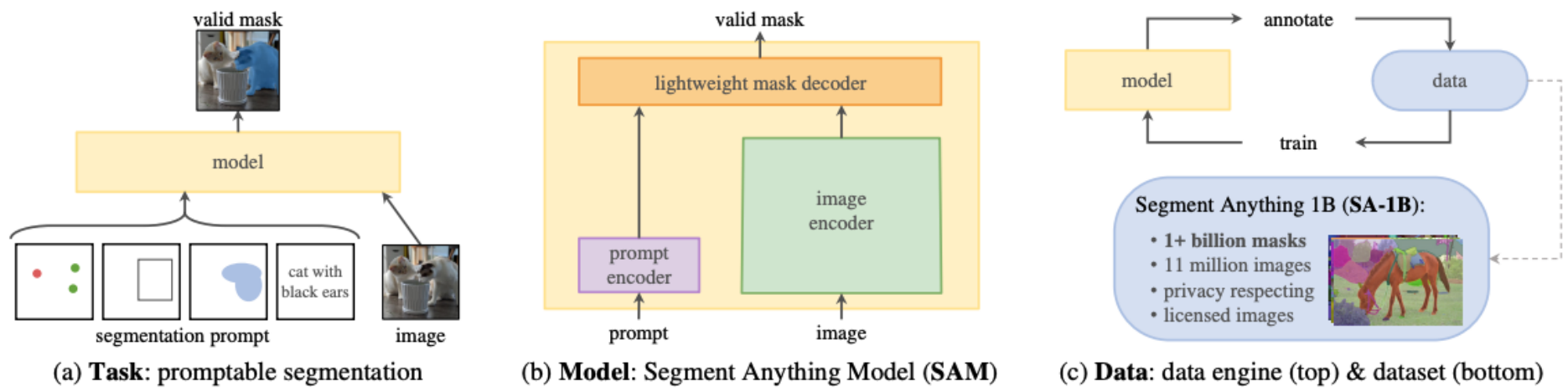


Figure 1: We aim to build a foundation model for segmentation by introducing three interconnected components: a promptable segmentation *task*, a segmentation *model* (SAM) that powers data annotation and enables zero-shot transfer to a range of tasks via prompt engineering, and a *data* engine for collecting SA-1B, our dataset of over 1 billion masks.

Previously on CENG501

Segment Anything Model (SAM) v2 2024

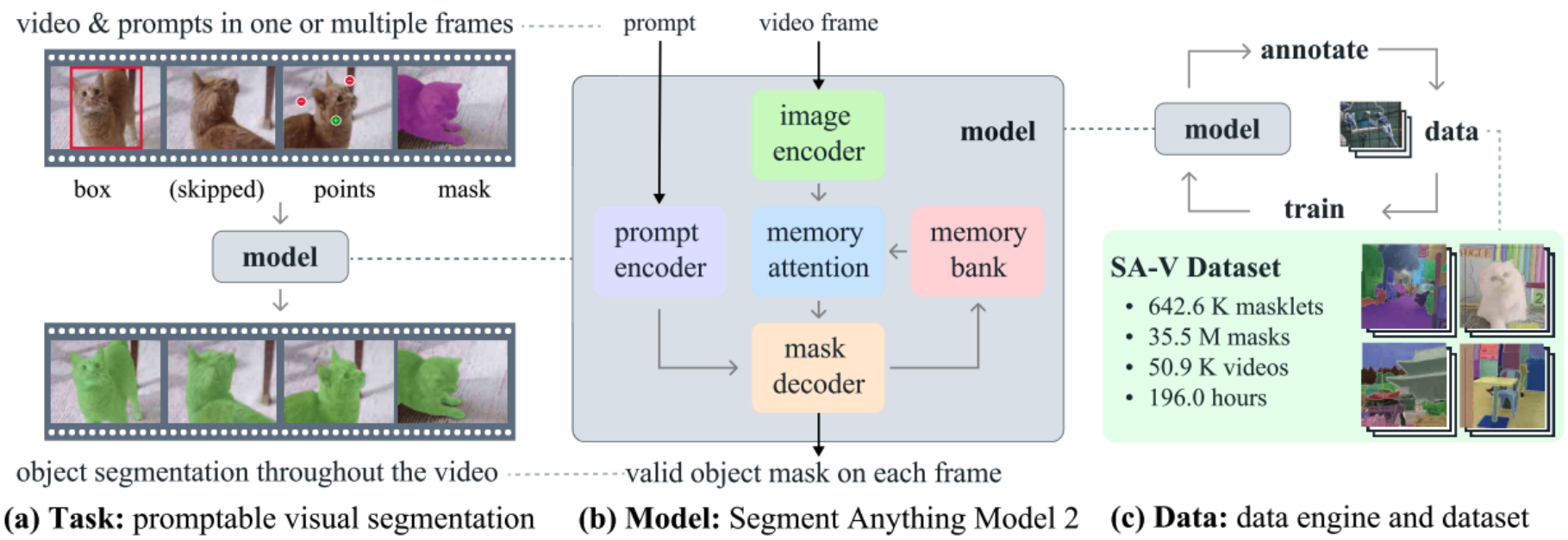


Figure 1 We introduce the Segment Anything Model 2 (SAM 2), towards solving the promptable visual segmentation task (a) with our foundation model (b), trained on our large-scale SA-V dataset collected through our data engine (c). SAM 2 is capable of interactively segmenting regions through prompts (clicks, boxes, or masks) on one or multiple video frames by utilizing a streaming memory that stores previous prompts and predictions.

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.

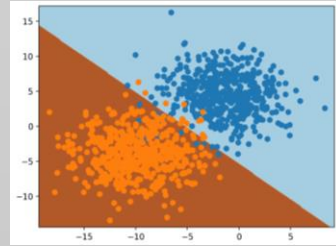
Administrative Notes

- New quiz this week
 - Deadline: Thursday midnight
- Time plan for the projects
 1. Milestone (November 24, midnight):
 - Github repo will be ready
 - Read & understand the paper
 - Download the datasets
 - Prepare the Readme file excluding the results & conclusion
 2. Milestone (December 8, midnight)
 - The results of the first experiment
 3. Milestone (January 5, midnight)
 - Final report (Readme file)
 - Repo with all code & trained models

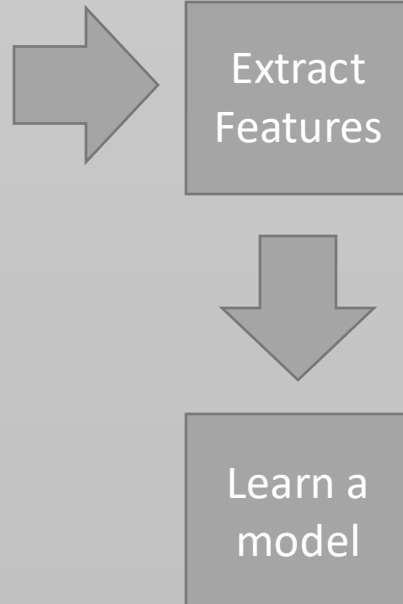
Overview & Problem Formulation

Supervised

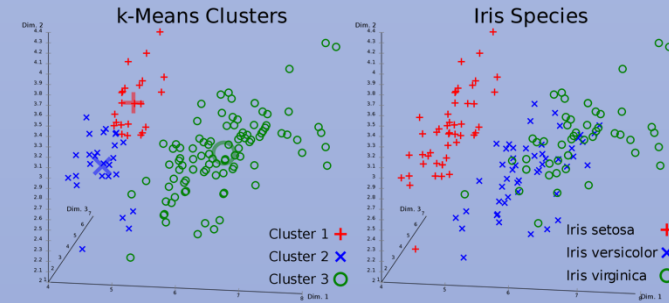
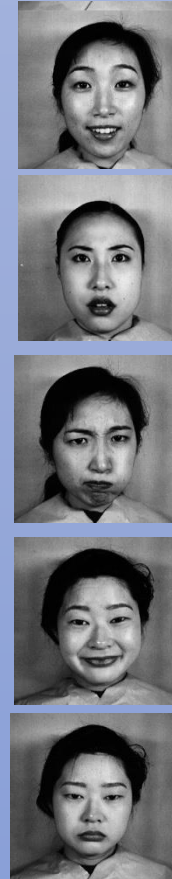
Instance	Label
	happy
	unhappy
	unhappy
	happy
	unhappy



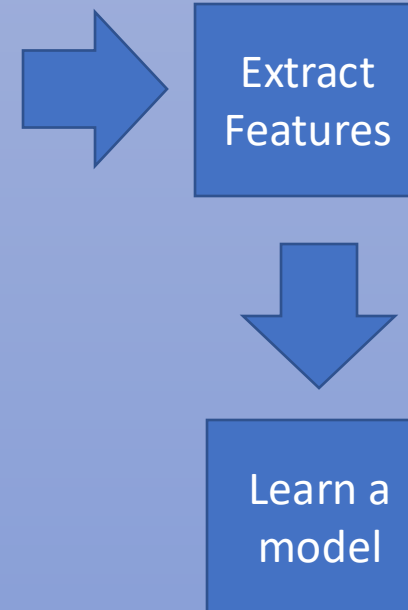
e.g. SVM



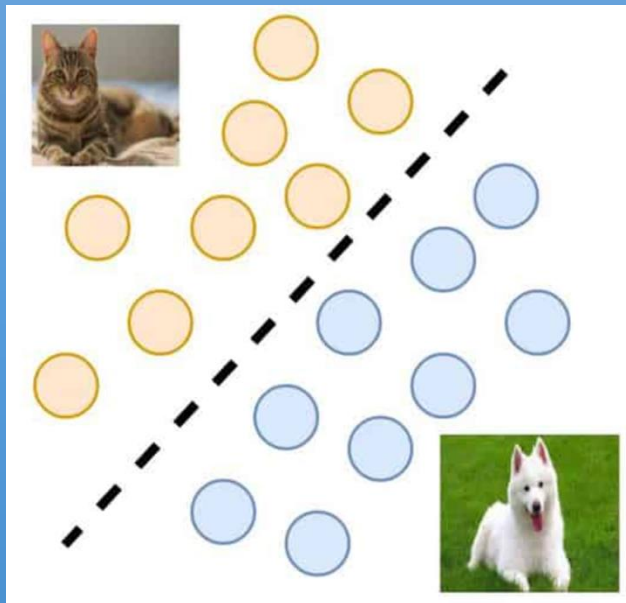
Unsupervised



e.g. k-means clustering

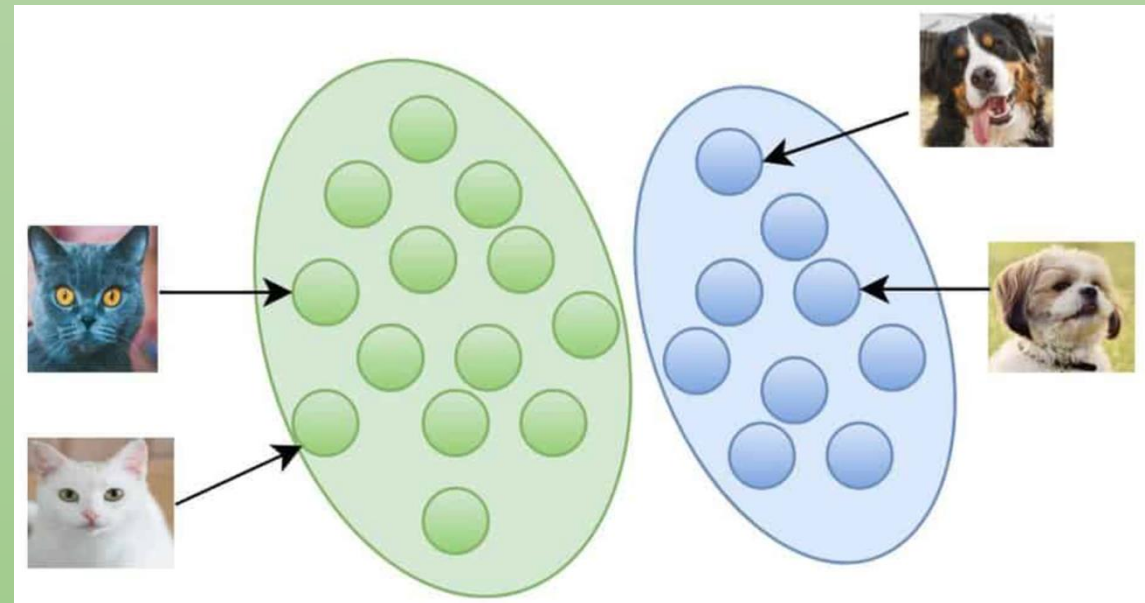


Discriminative



Find separating line
(in general: hyperplane)

Generative



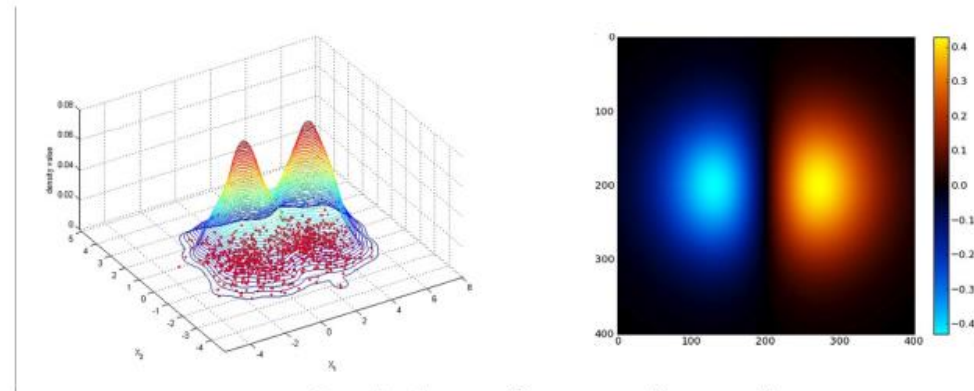
Learn a model for
each class.

Unsupervised Learning via Density Estimation



Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



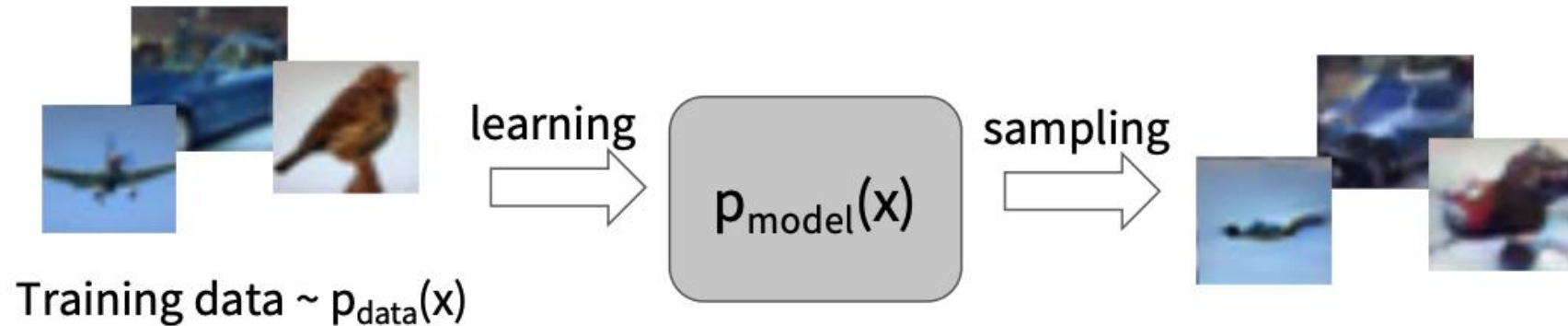
2-d density estimation

Modeling $p(x)$

2-d density images [left](#) and [right](#) are [CC0 public domain](#)

Generative Modeling

- Learning the probability distribution of data

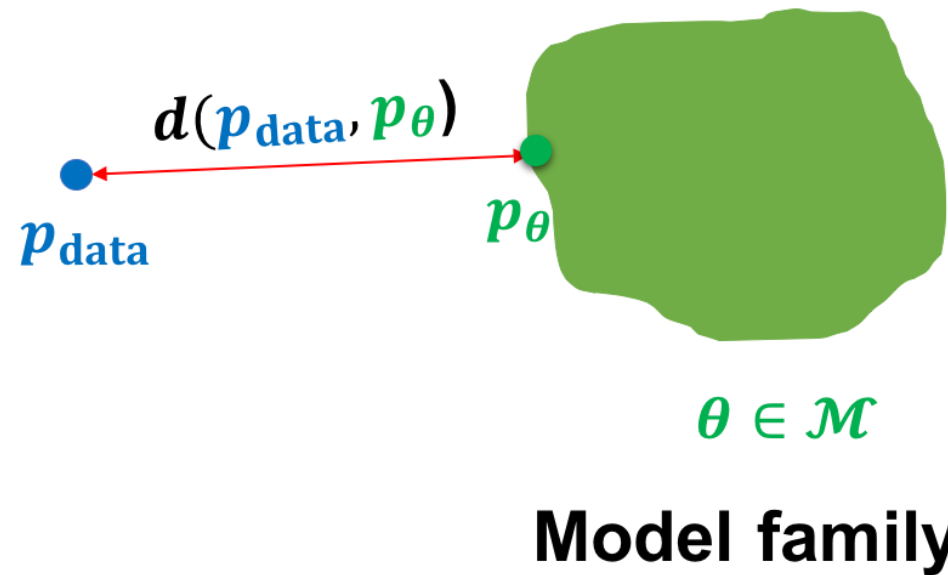


Objectives:

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

Generative Modeling

- Learning the probability distribution of data



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

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 ...

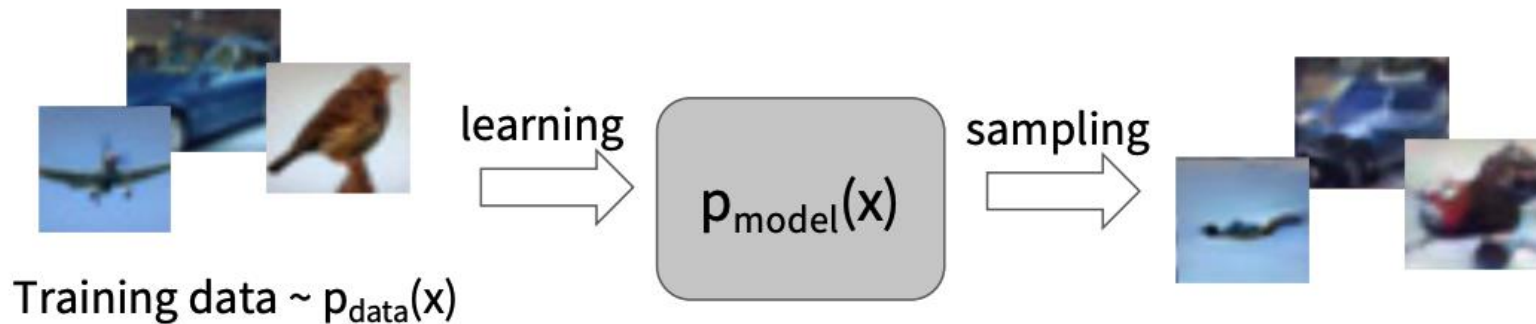
Figures from L-R are copyright: (1) [Alec Radford et al. 2016](#); (2) [Phillip Isola et al. 2017](#). Reproduced with authors permission (3) [BAIR Blog](#).

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 $p_{\text{model}}(x)$
- Implicit density estimation: learn model that can sample from $p_{\text{model}}(x)$ without explicitly defining it.

Taxonomy of Generative Models

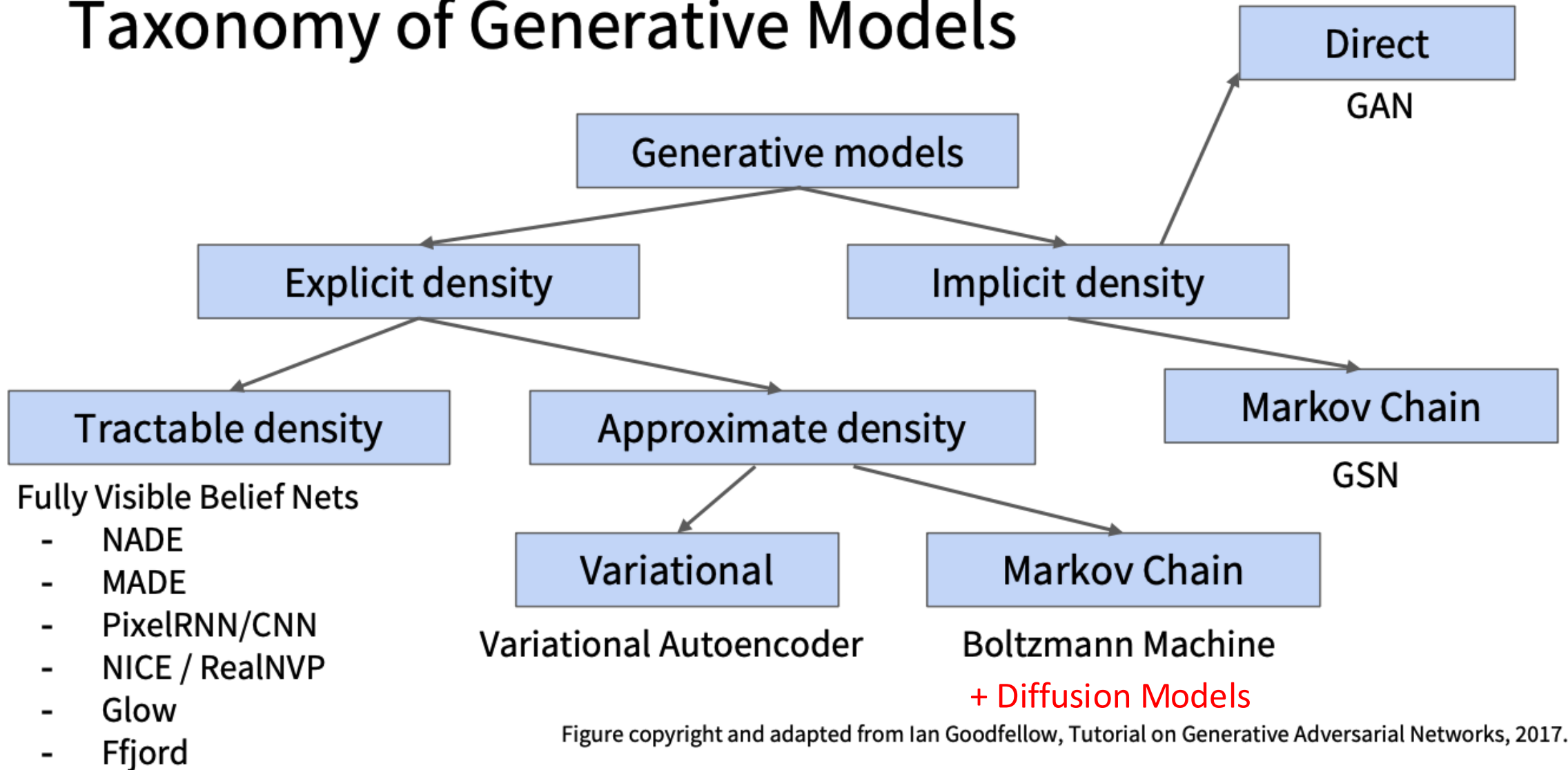


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Autoregressive Models

Fully visible belief network (FVBN)

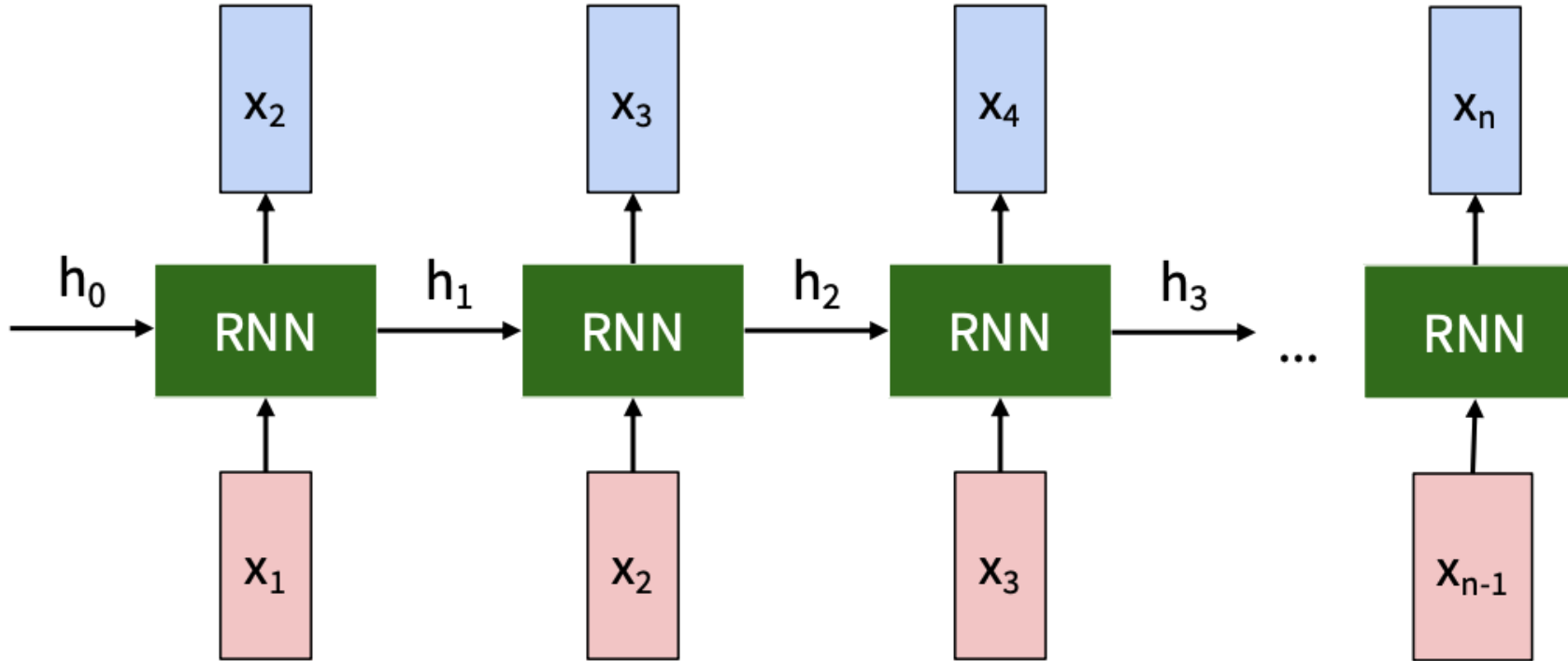
Explicit density model

$$p(\mathbf{x}) = p(x_1, x_2, \dots, x_n)$$

↑
Likelihood of
image \mathbf{x}

↑
Joint likelihood of each
pixel in the image

Recurrent Neural Network

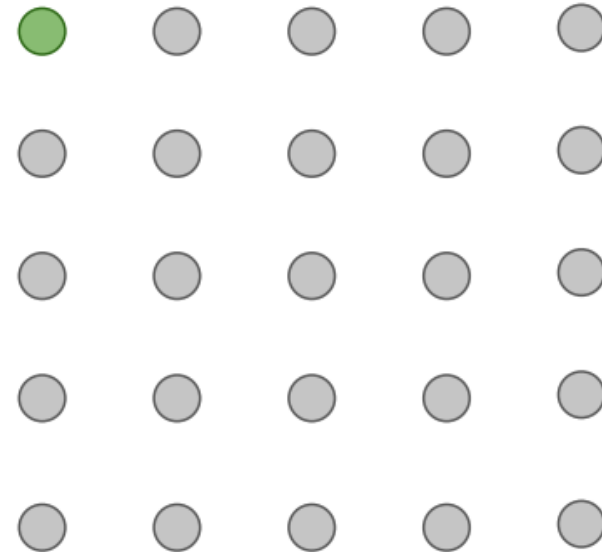


$$p(x_i | x_1, \dots, x_{i-1})$$

PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

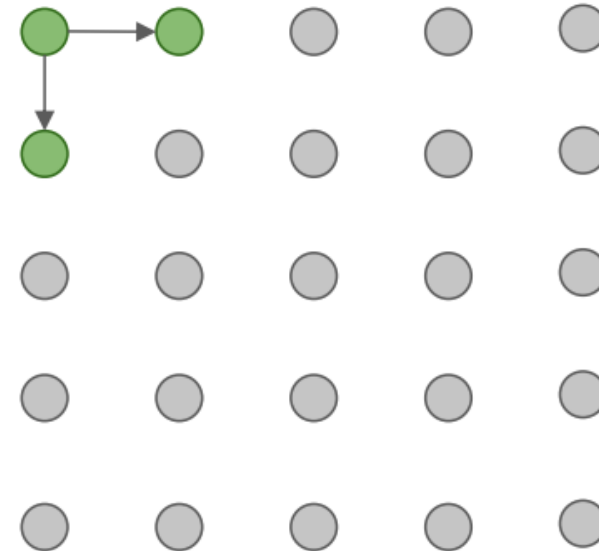
Dependency on previous pixels modeled using an RNN (LSTM)



PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

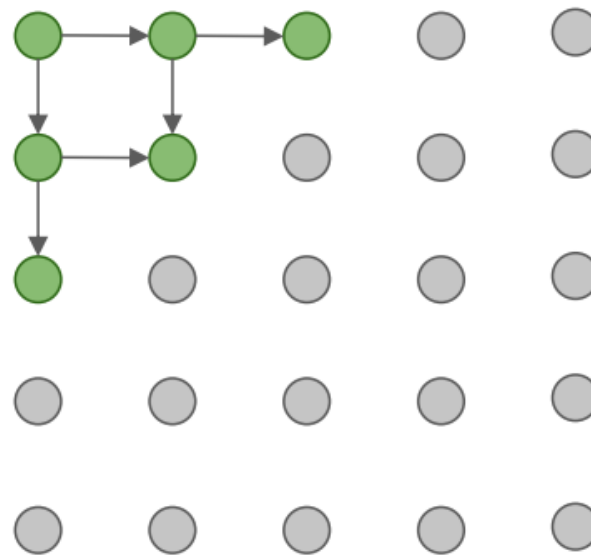
Dependency on previous pixels modeled using an RNN (LSTM)



PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

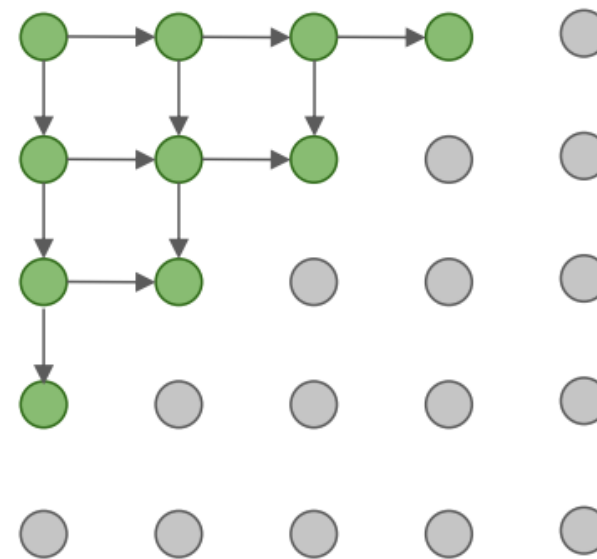


PixelRNN [van der Oord et al. 2016]

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!



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)

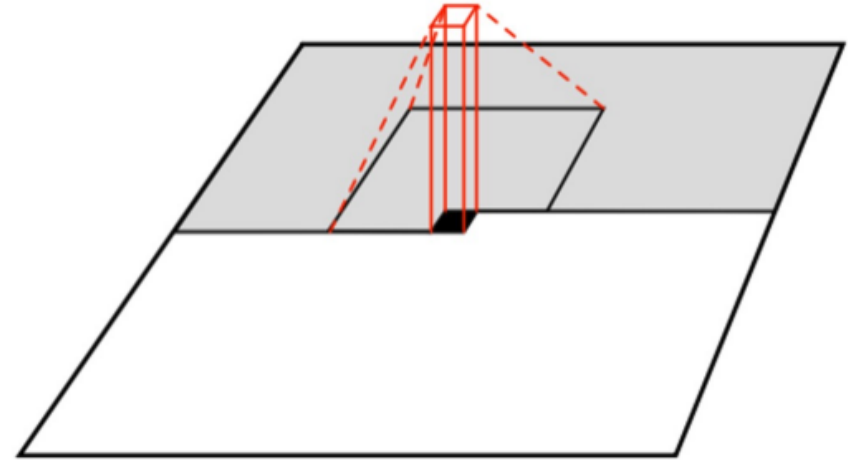


Figure copyright van der Oord et al., 2016. Reproduced with permission.

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

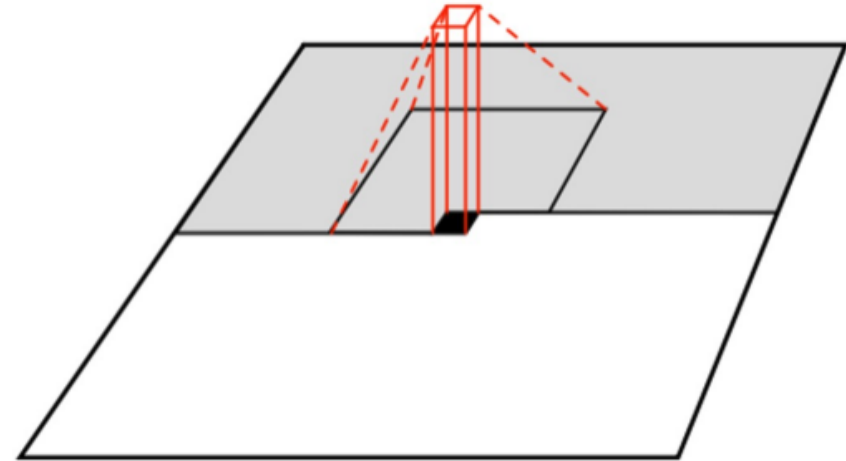
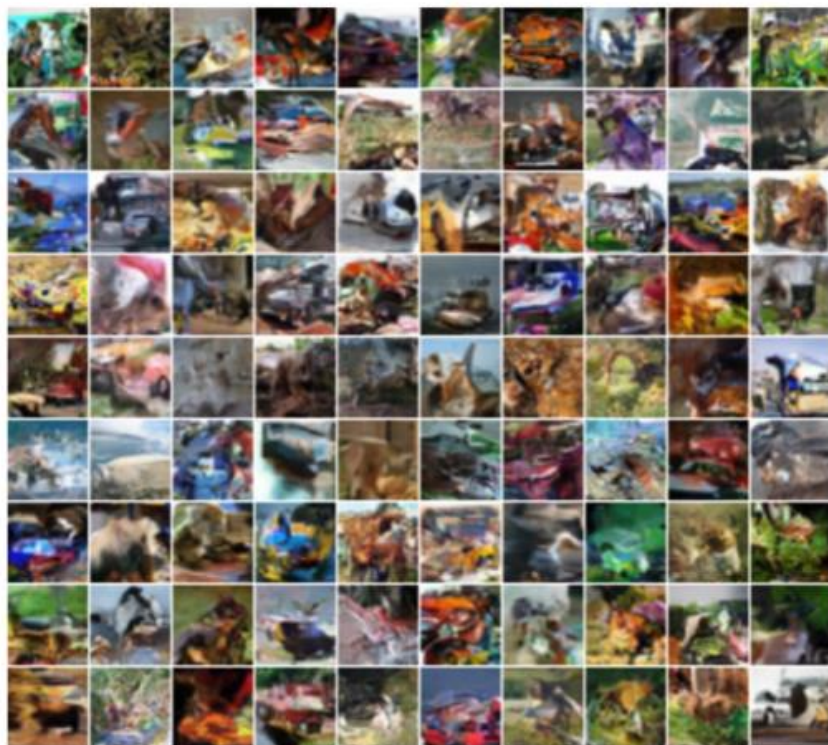
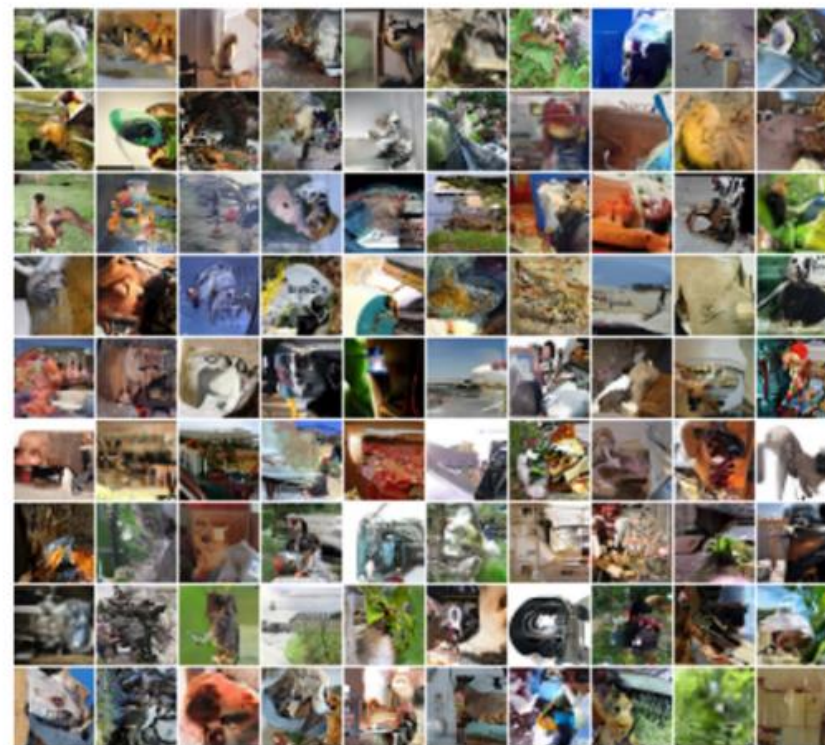


Figure copyright van der Oord et al., 2016. Reproduced with permission.

Generation Samples



32x32 CIFAR-10



32x32 ImageNet

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

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++)

Variational Autoencoders

Autoregressive Models vs Variational Autoencoders

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

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

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

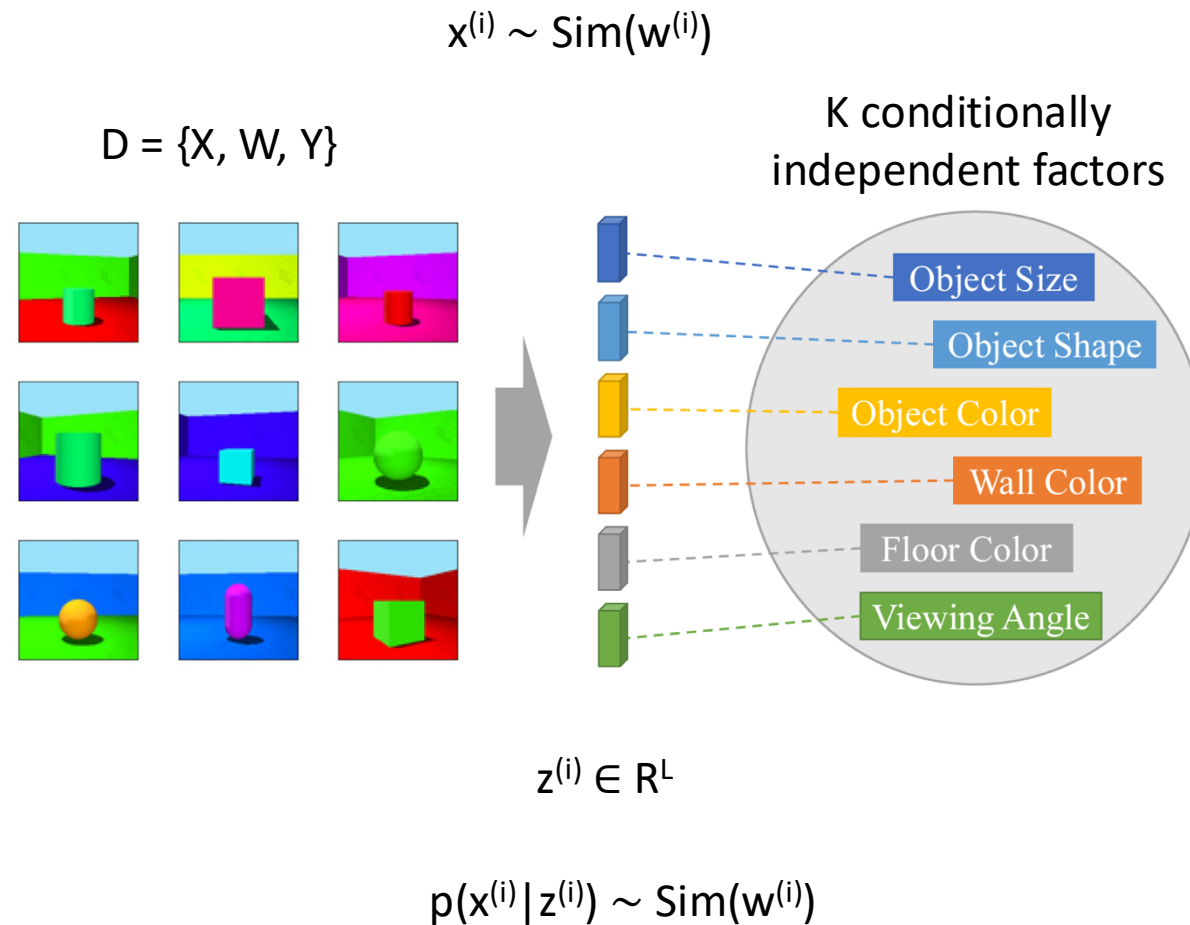
$$p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$$

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

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

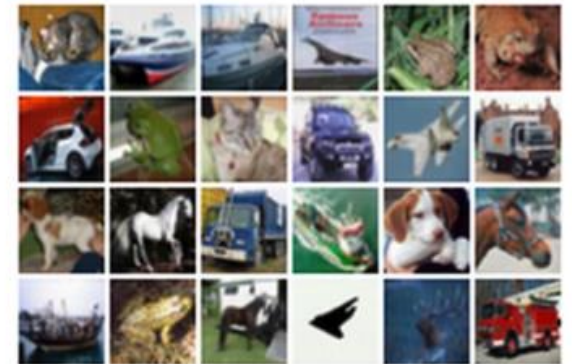
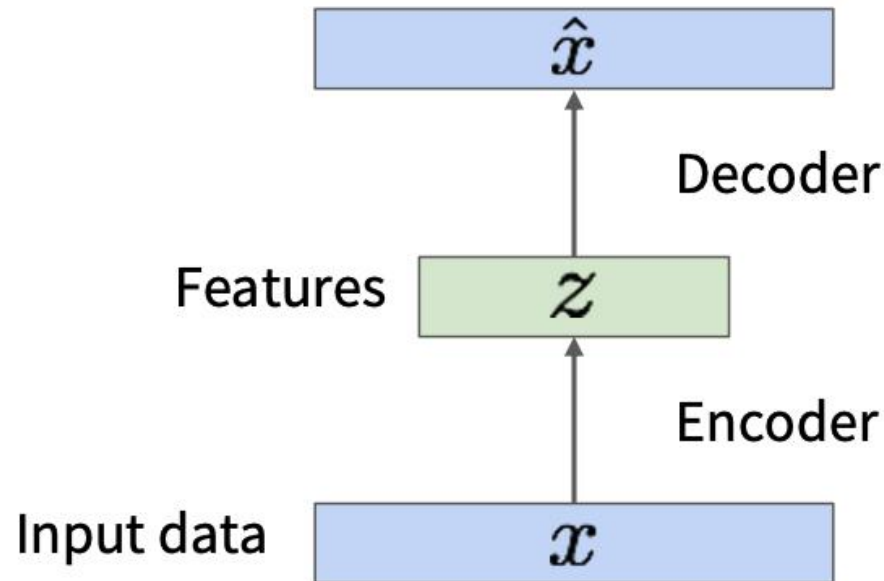
Why latent z ?

Disentangled Representation Learning



Recap: Autoencoders

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

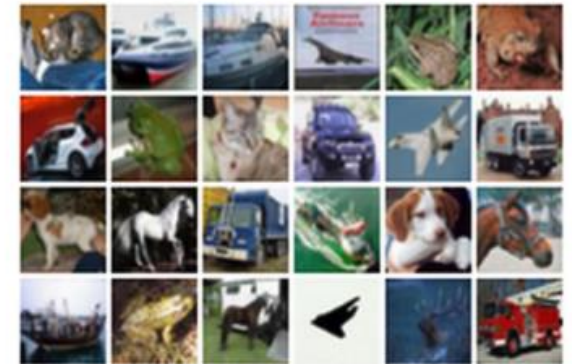
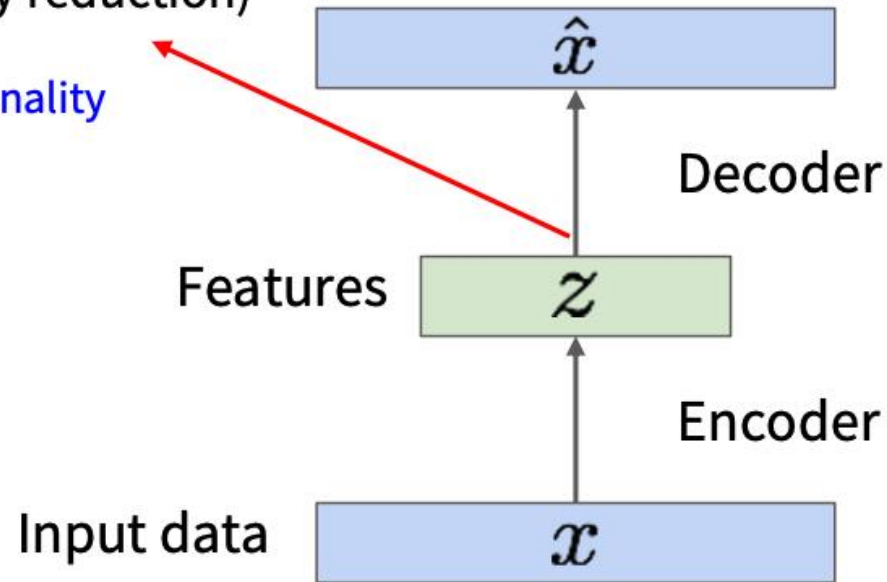


Recap: Autoencoders

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

z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?



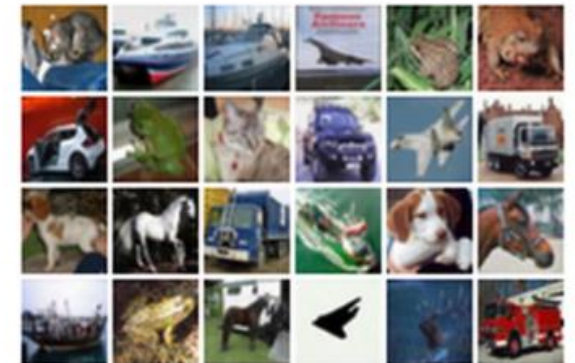
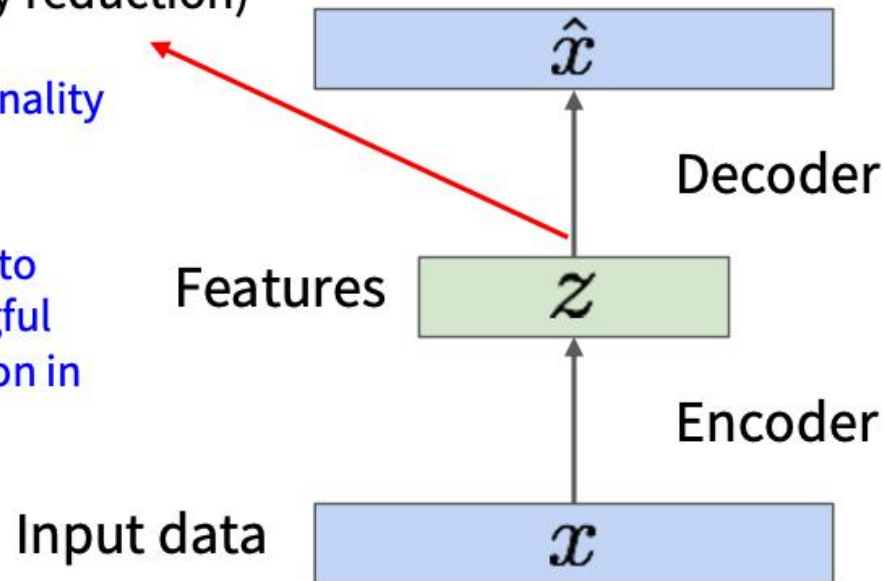
Recap: Autoencoders

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

z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?

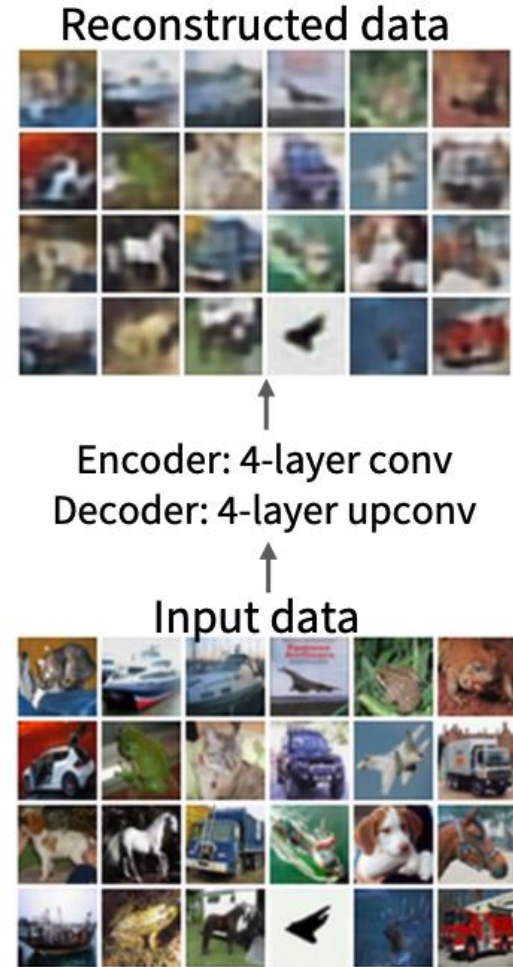
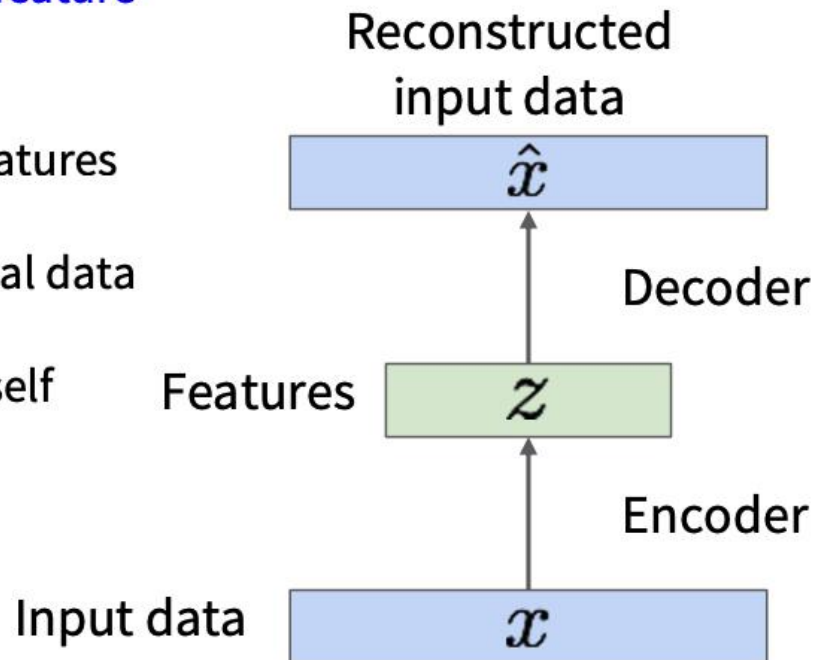
A: Want features to capture meaningful factors of variation in data



Recap: Autoencoders

How to learn this feature representation?

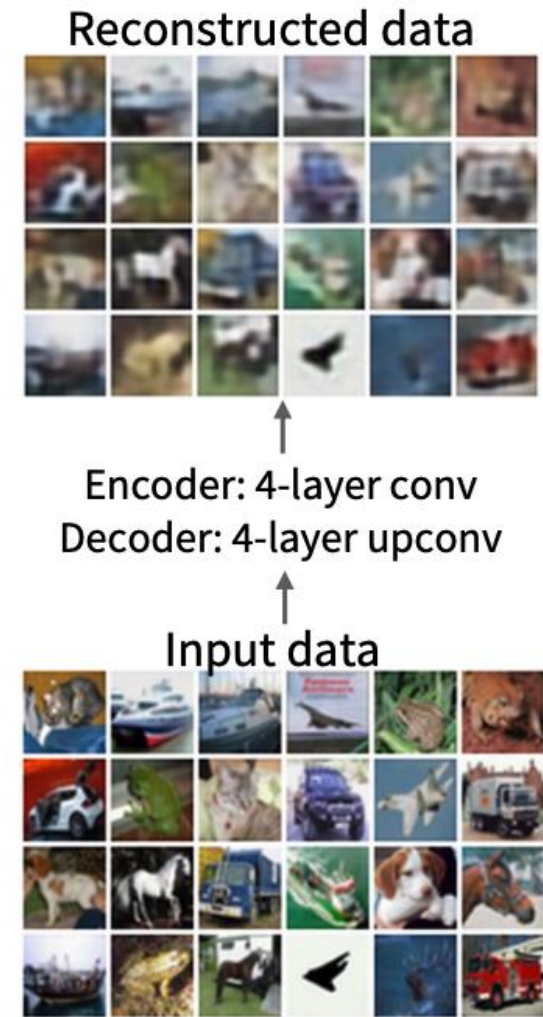
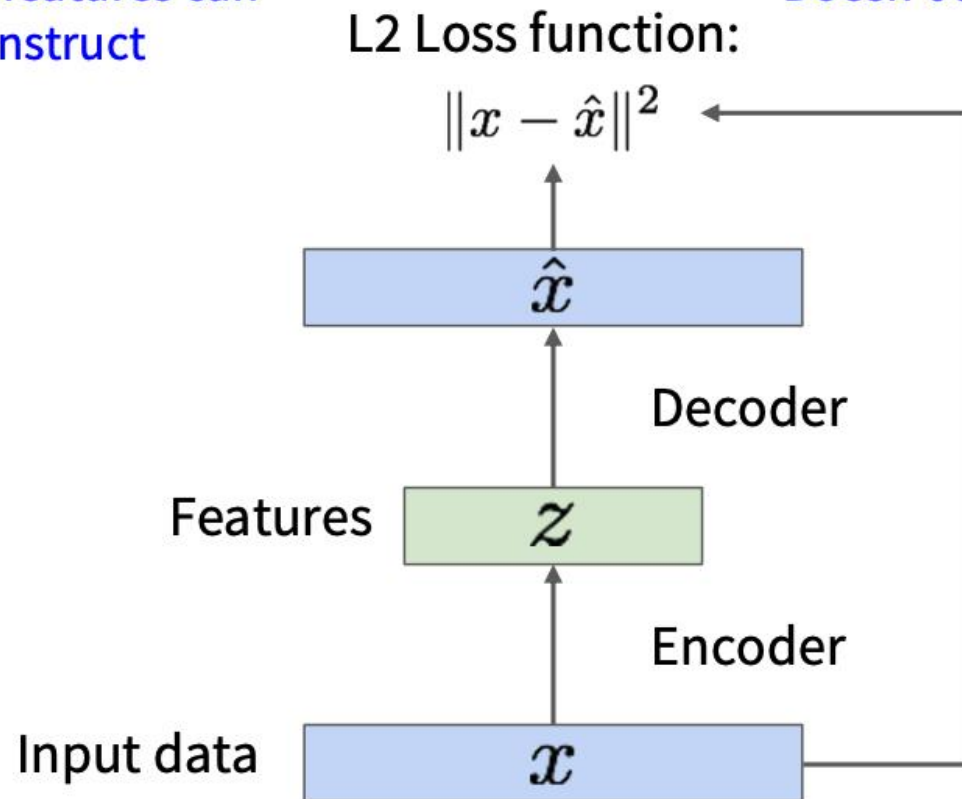
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding input itself



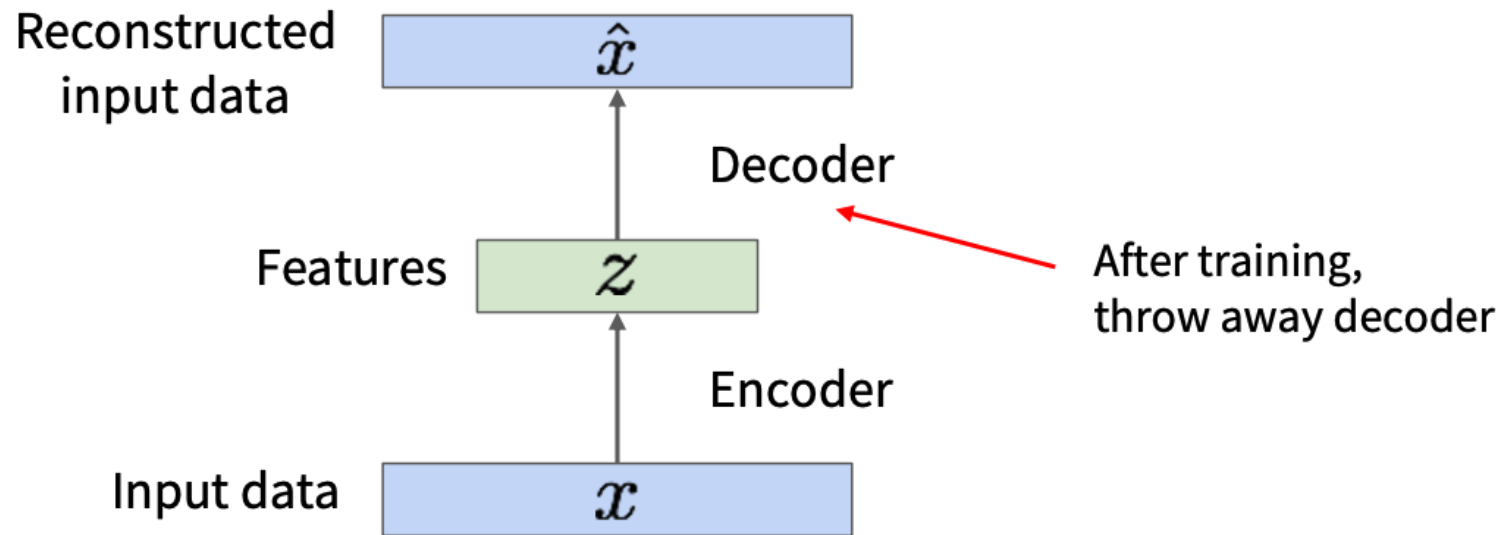
Recap: Autoencoders

Train such that features can be used to reconstruct original data

Doesn't use labels!



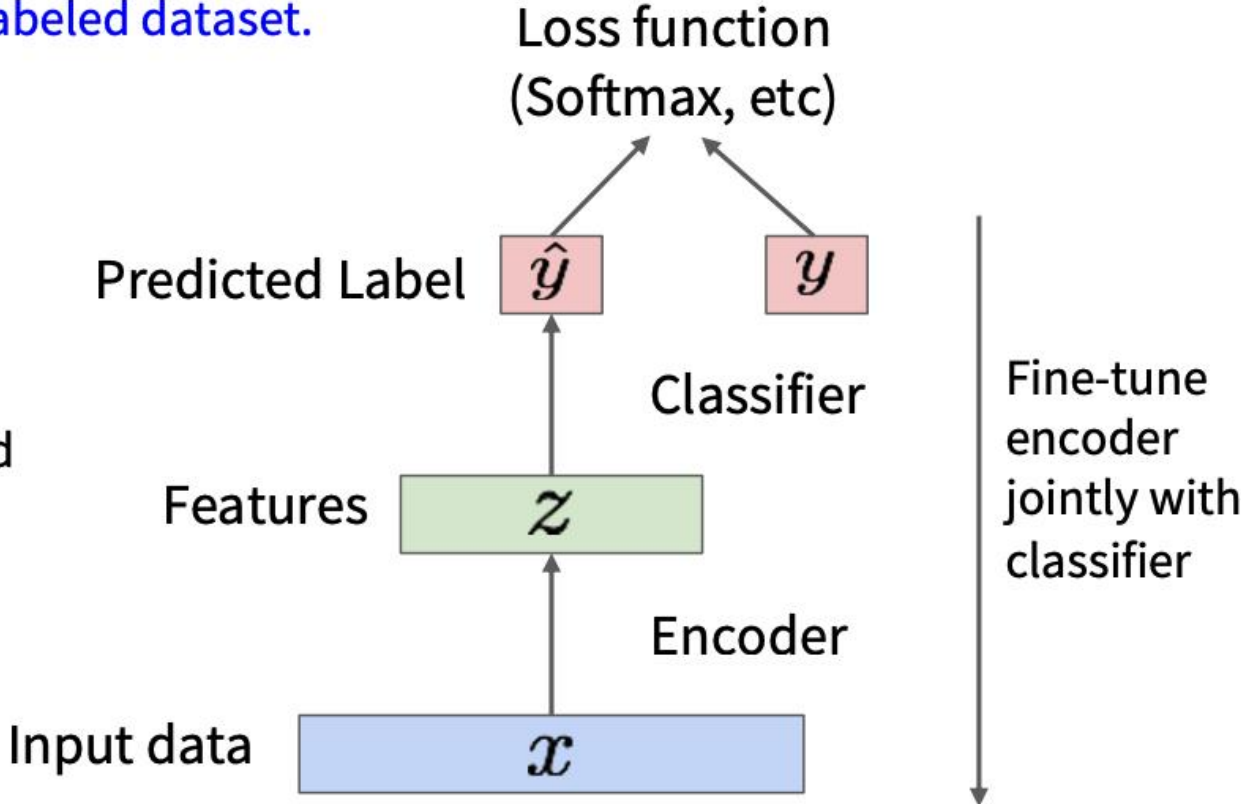
Recap: Autoencoders



Recap: Autoencoders

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

Encoder can be used to initialize a supervised model

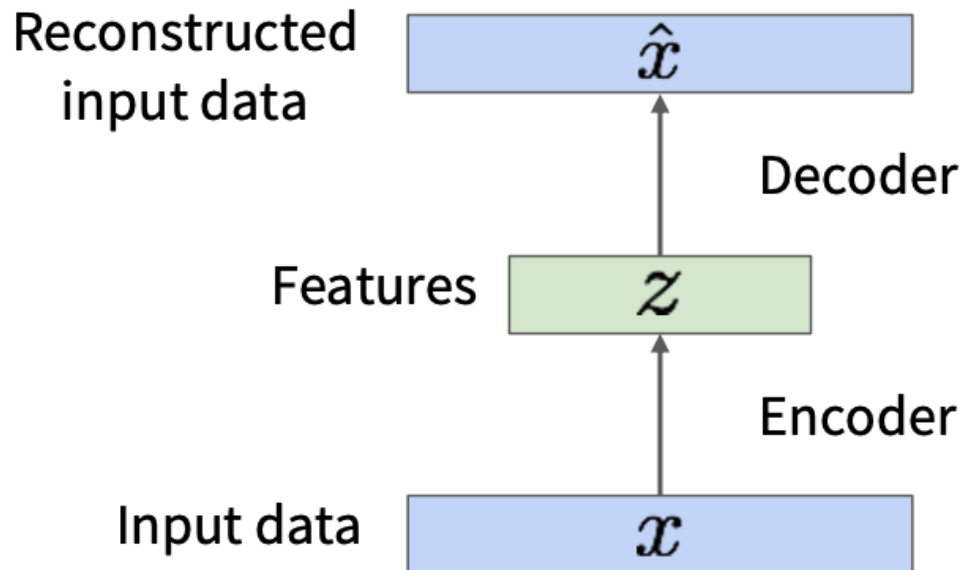


bird plane
dog deer truck

Train for final task (sometimes with small data)



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?

Variational Autoencoders (VAEs)

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

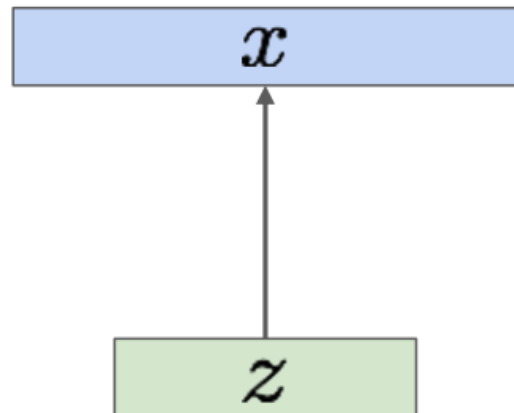
Assume training data $\{x^{(i)}\}_{i=1}^N$ is generated from the distribution of unobserved (latent) representation z

Sample from true conditional

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

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.

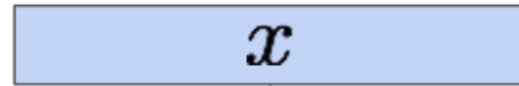
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders (VAEs)

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

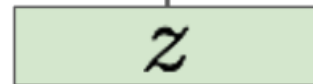
Sample from
true conditional

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



Sample from
true prior

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



x

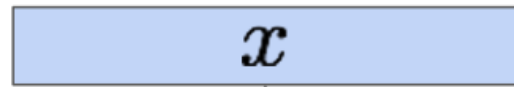
z

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

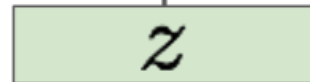
Variational Autoencoders (VAEs)

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

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



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



How should we represent this model?

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

Variational Autoencoders (VAEs)

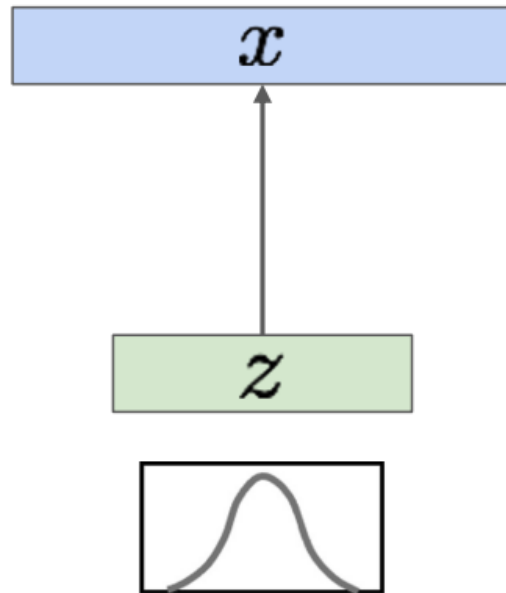
We want to estimate the true parameters θ^* of this generative model given training data x .

Sample from true conditional

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

Sample from true prior

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



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

Variational Autoencoders (VAEs)



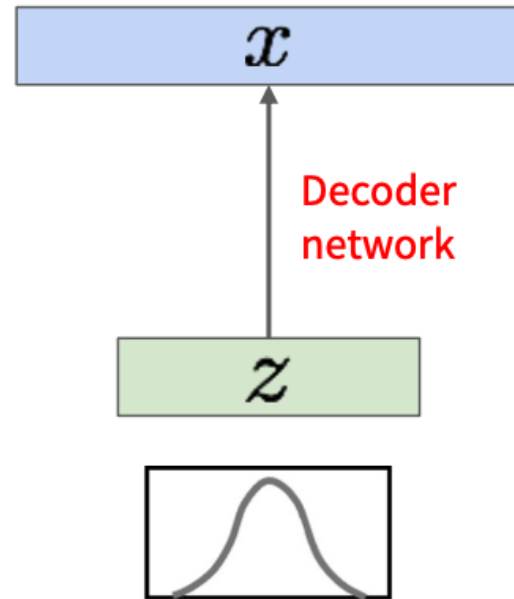
We want to estimate the true parameters θ^* of this generative model given training data x .

Sample from true conditional

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

Sample from true prior

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



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

Variational Autoencoders (VAEs)

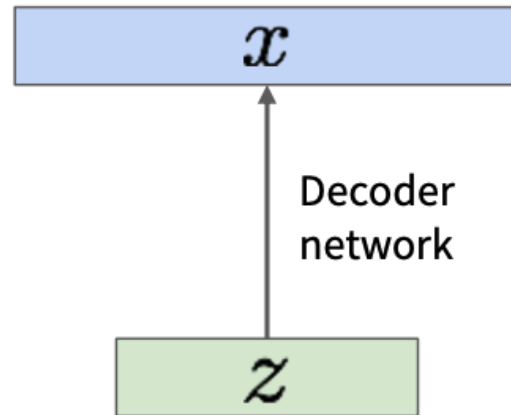
We want to estimate the true parameters θ^* of this generative model given training data x .

Sample from true conditional

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

Sample from true prior

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



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

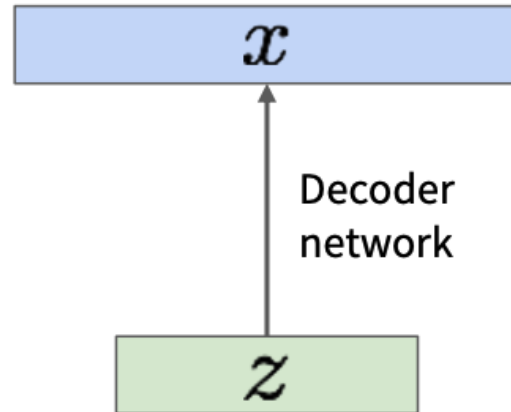
Variational Autoencoders (VAEs)

Sample from
true conditional

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

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

Variational Autoencoders (VAEs): Intractability

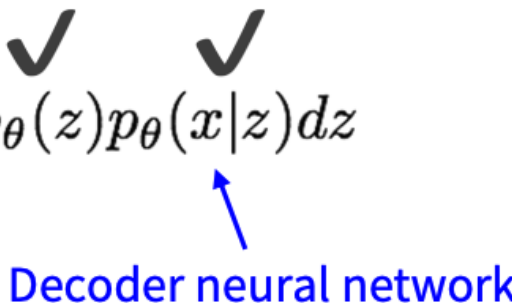
Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$

↑
Simple Gaussian prior

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

Variational Autoencoders (VAEs): Intractability

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



Decoder neural network

The diagram shows the equation $p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$. There are two checkmarks above the terms $p_{\theta}(z)$ and $p_{\theta}(x|z)$. A blue arrow points from the text "Decoder neural network" below to the $p_{\theta}(x|z)$ term in the equation.

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

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

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

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

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

Variational Autoencoders (VAEs)

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

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

Variational Autoencoders (VAEs)

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

Variational Autoencoders (VAEs)

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

Variational Autoencoders (VAEs)

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

Variational Autoencoders (VAEs)

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

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

Variational Autoencoders (VAEs)

We want to maximize the data likelihood

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

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 .

Variational Autoencoders (VAEs)

We want to maximize the data likelihood

$$\begin{aligned}\log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[\log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\ &= \mathbf{E}_z \left[\log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\ &= \underbrace{\mathbf{E}_z \left[\log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))}_{\geq 0}\end{aligned}$$

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

Variational Autoencoders (VAEs)

$$\begin{aligned}
 \log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[\log p_{\theta}(x^{(i)}) \right] && (p_{\theta}(x^{(i)})) \text{ Does not depend on } z \\
 &= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] && (\text{Bayes' Rule}) \\
 &= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] && (\text{Multiply by constant}) \\
 &= \mathbf{E}_z \left[\log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] && (\text{Logarithms}) \\
 &= \underbrace{\mathbf{E}_z \left[\log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))}_{\geq 0}
 \end{aligned}$$

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)

Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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

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

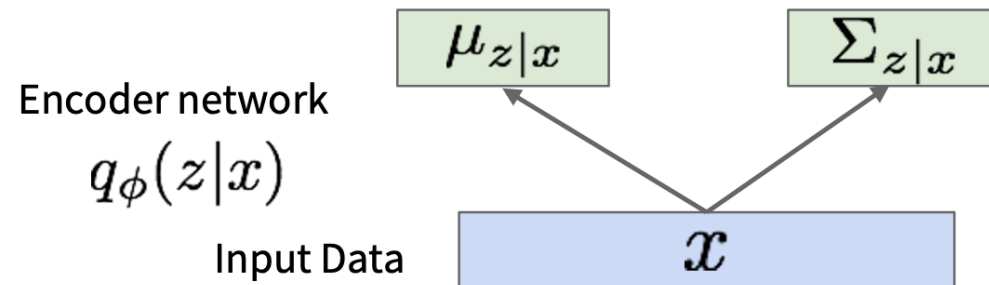
Input Data

\mathcal{X}

Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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



Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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

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

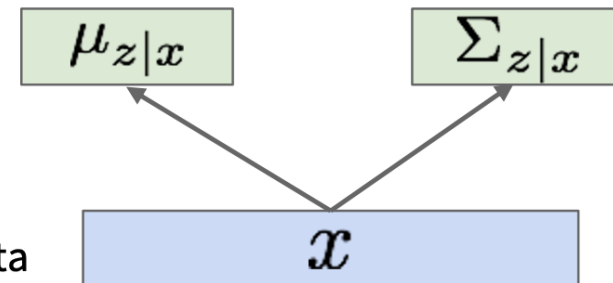
Have analytical solution

Make approximate posterior distribution close to prior

Encoder network

$$q_\phi(z|x)$$

Input Data



Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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

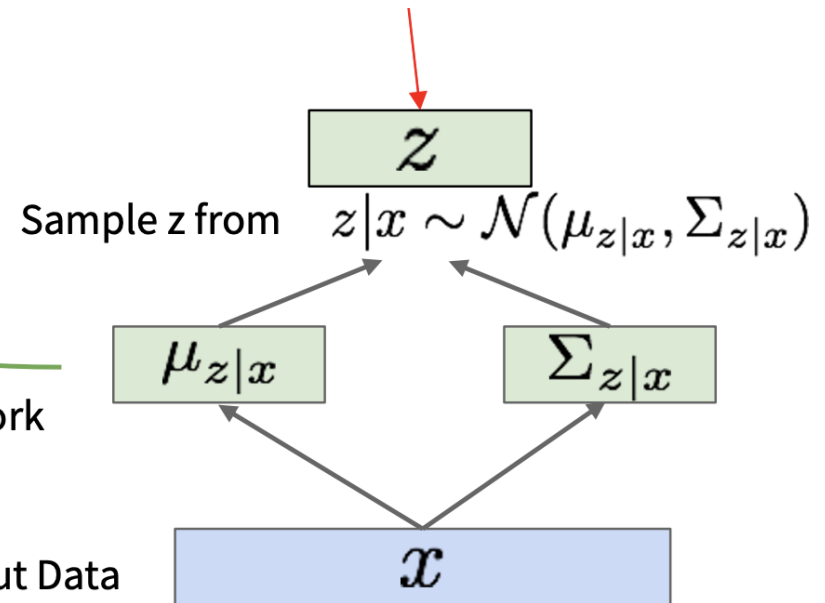
Make approximate posterior distribution close to prior

Encoder network

$$q_\phi(z|x)$$

Input Data

Not part of the computation graph!



Variational Autoencoders (VAEs)

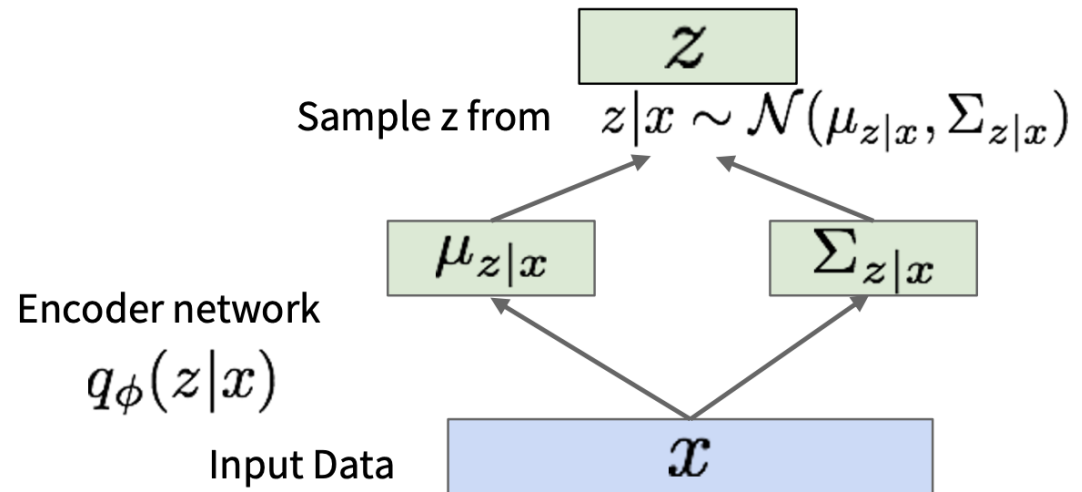
Putting it all together: maximizing the likelihood lower bound

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

Reparameterization trick to make sampling differentiable:

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

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



Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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

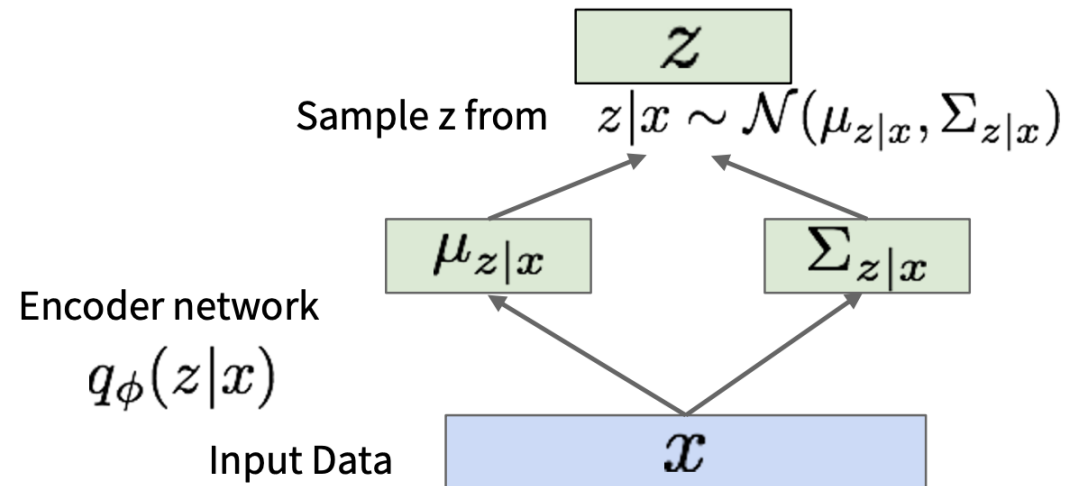
Reparameterization trick to make sampling differentiable:

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

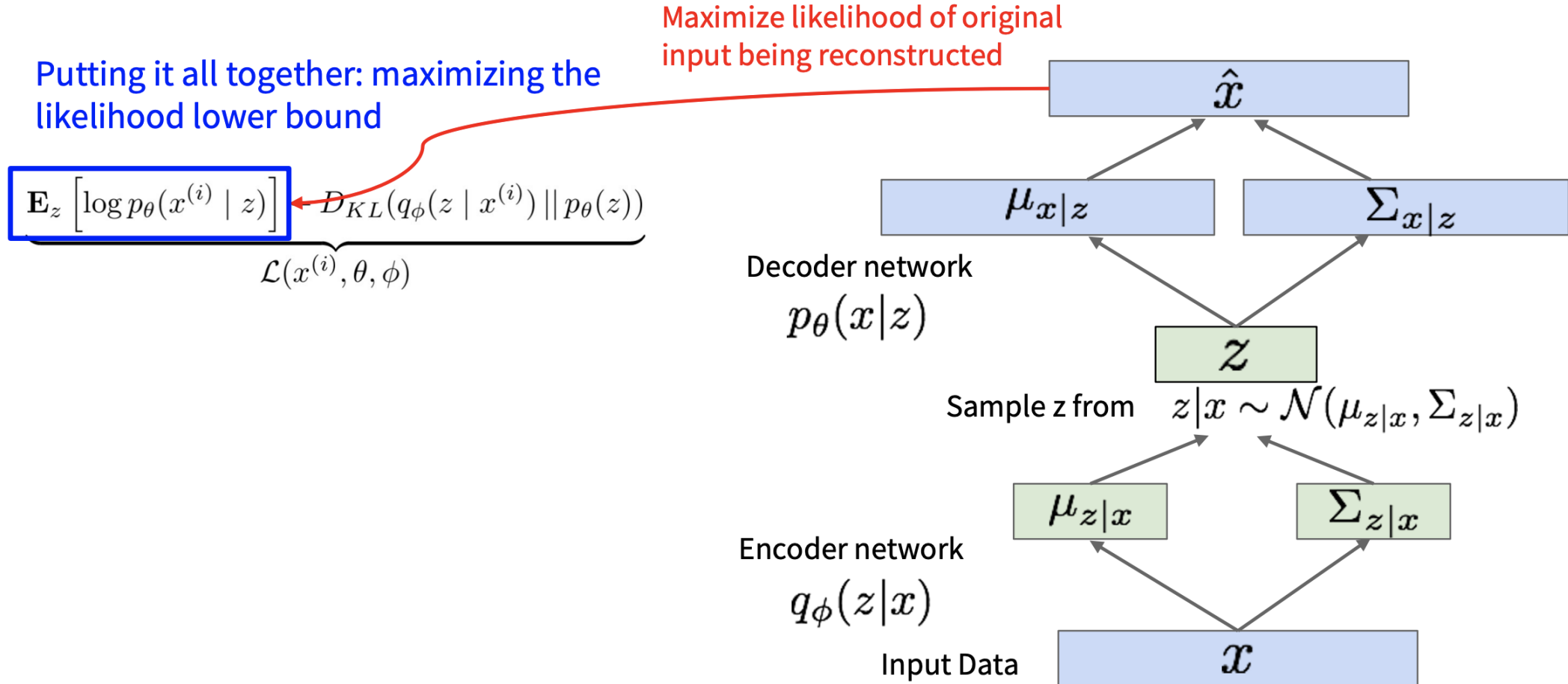
$$z = \mu_{z|x} + \epsilon \sigma_{z|x}$$

Input to the graph

Part of computation graph



Variational Autoencoders (VAEs)

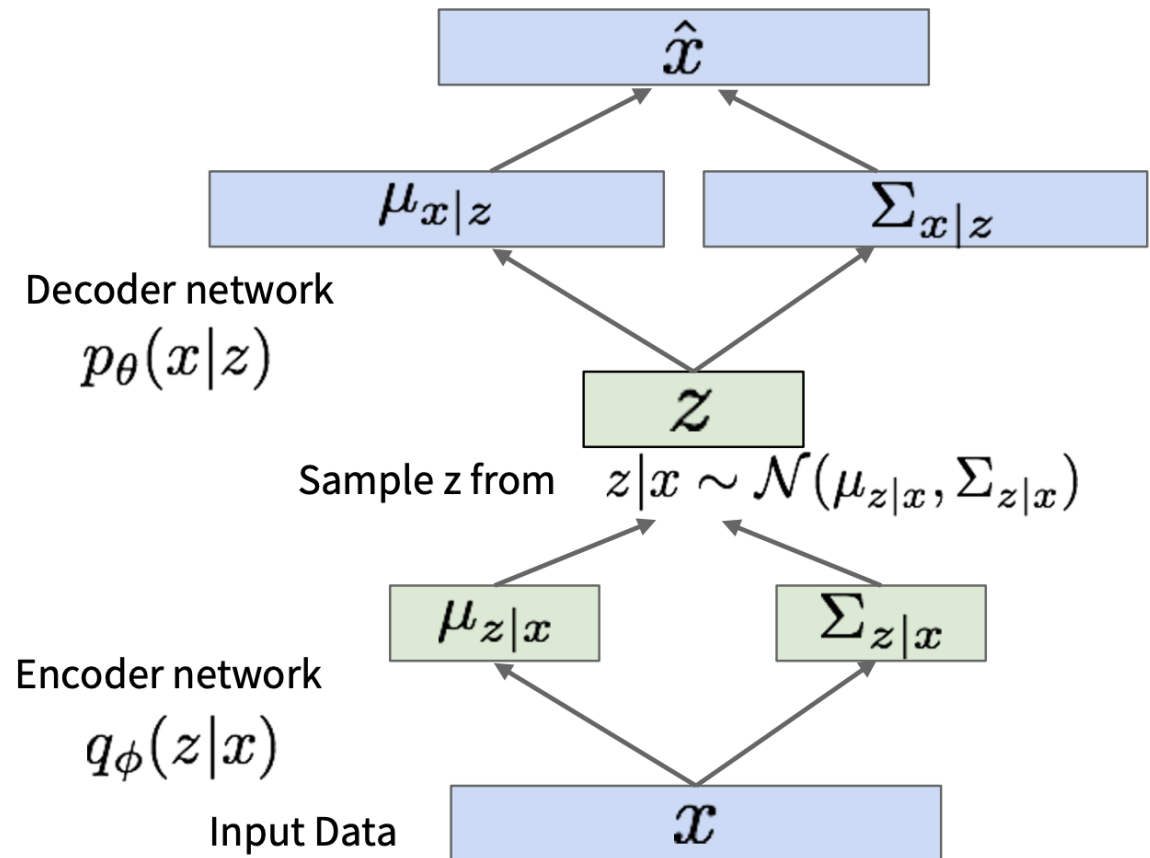


Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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

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

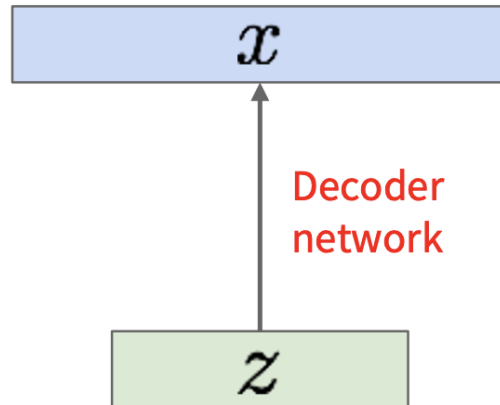


Variational Autoencoders (VAEs)

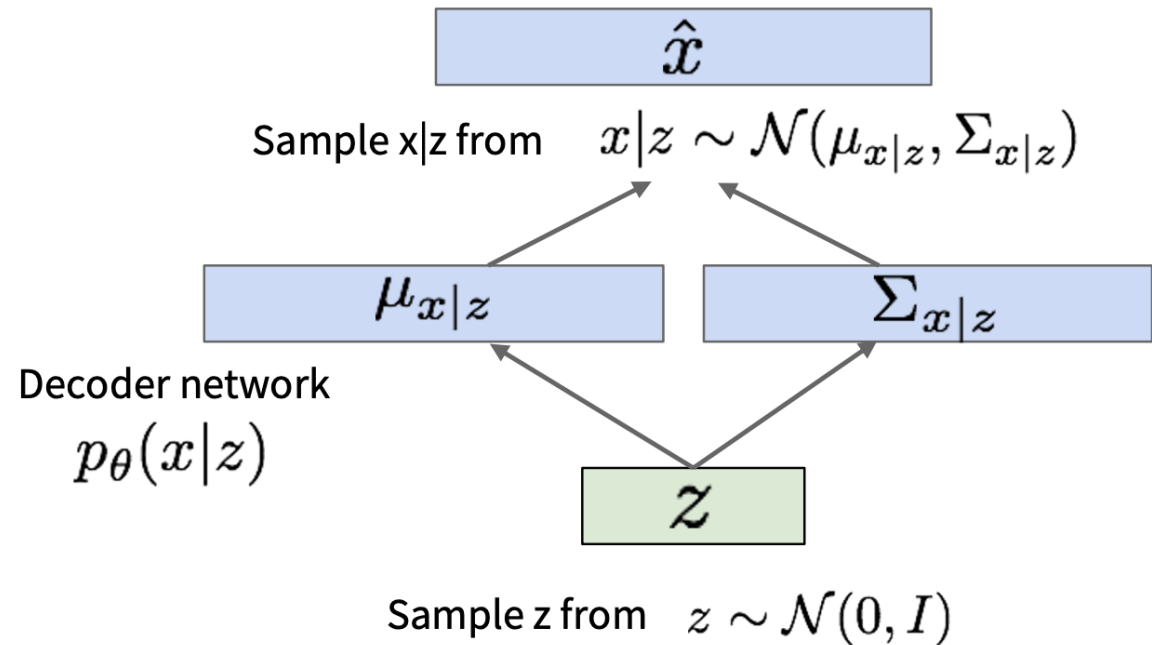
Our assumption about data generation process

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

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



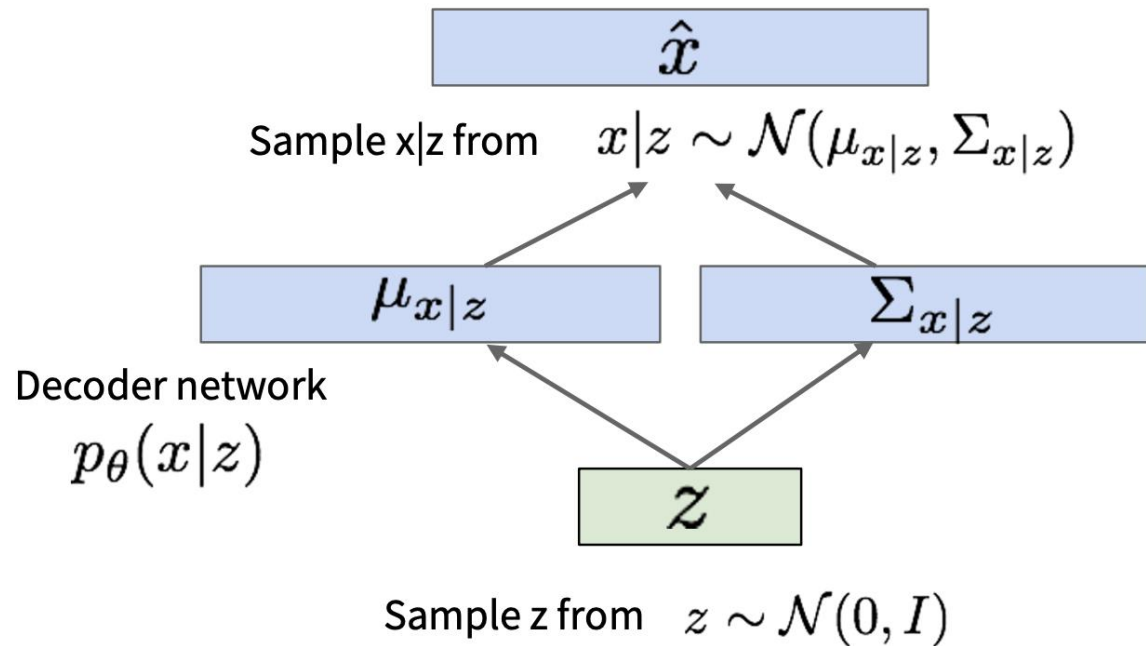
Now given a trained VAE:
use decoder network & sample z from prior!



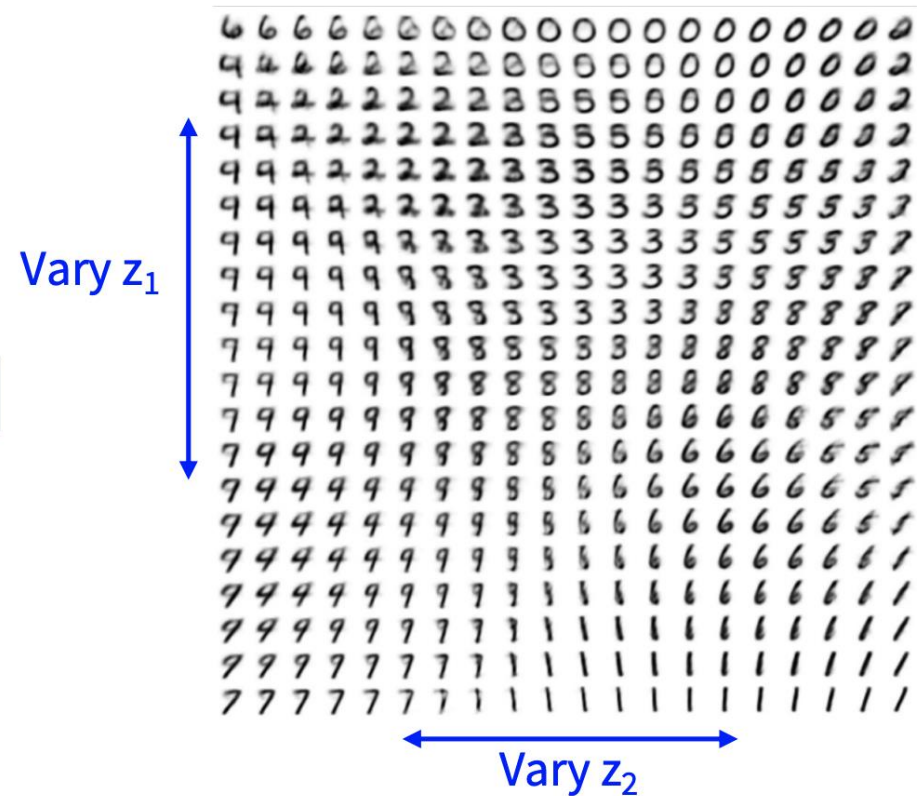
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders (VAEs)

Use decoder network. Now sample z from prior!



Data manifold for 2-d z



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

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 $q_\phi(z|x)$!

Degree of smile

Vary z_1



Head pose

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

Variational Autoencoders (VAEs)



32x32 CIFAR-10



Labeled Faces in the Wild

Figures copyright (L) Dirk Kingma et al. 2016; (R) Anders Larsen et al. 2017. Reproduced with permission.

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

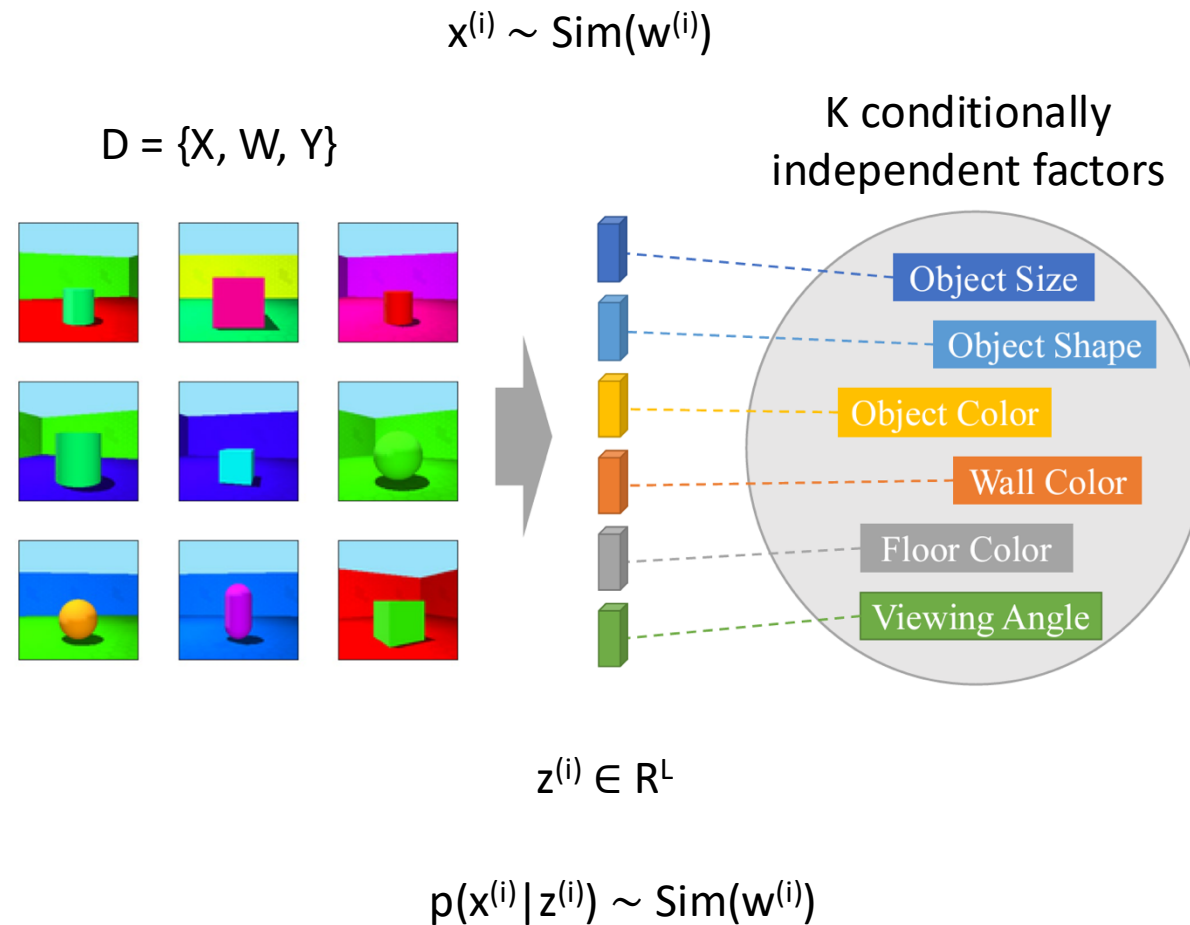
Cons:

- 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)

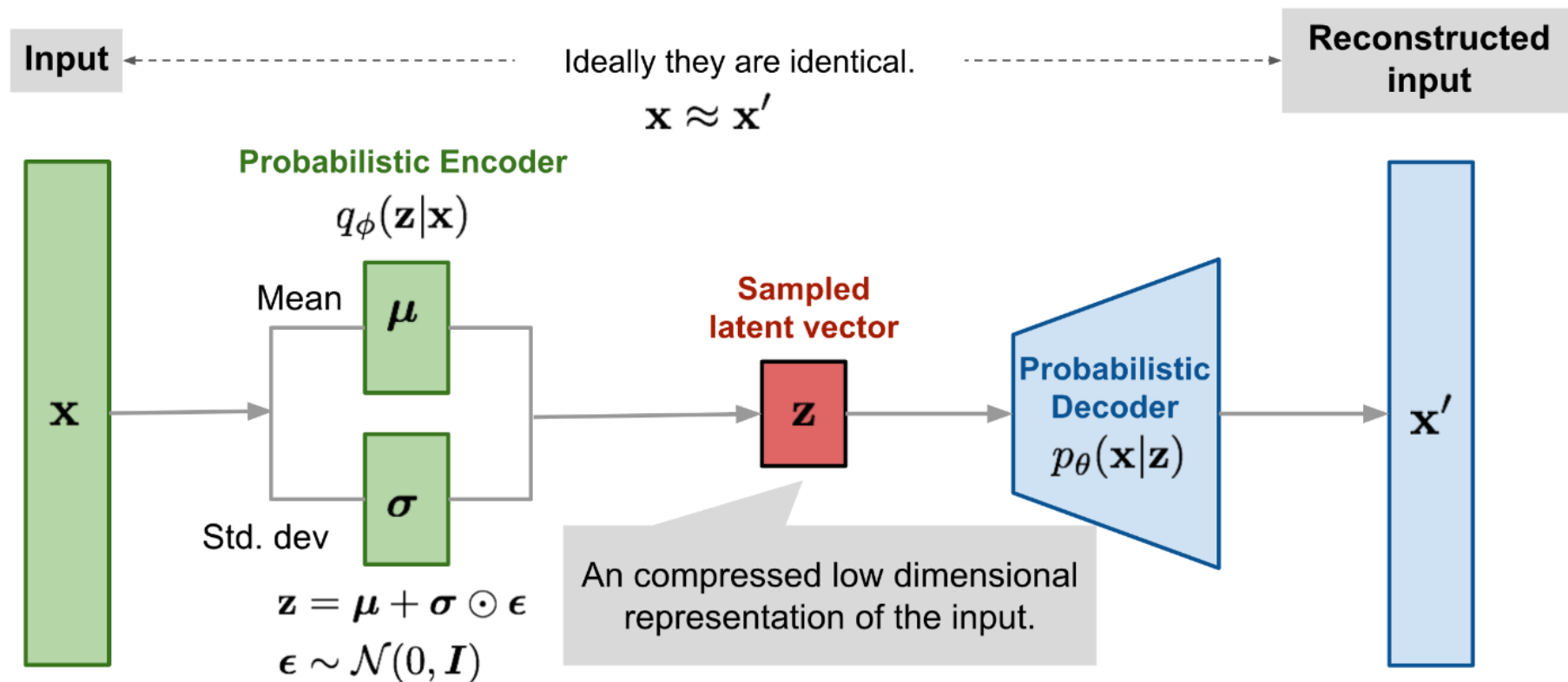
Active areas of research:

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

Disentangled Representation Learning



Variational Auto-Encoders and its Variations



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

β -VAE (Higgins et al., 2016)

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

- $\beta > 1$ implies stronger disentanglement*
- Limitations:
 - Increased reconstruction loss
 - Increased complexity

Our Work (Mogultay, Kalkan, Vural, 2024)

- Learnable VAE

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

- Dimensionwise-learnable VAE

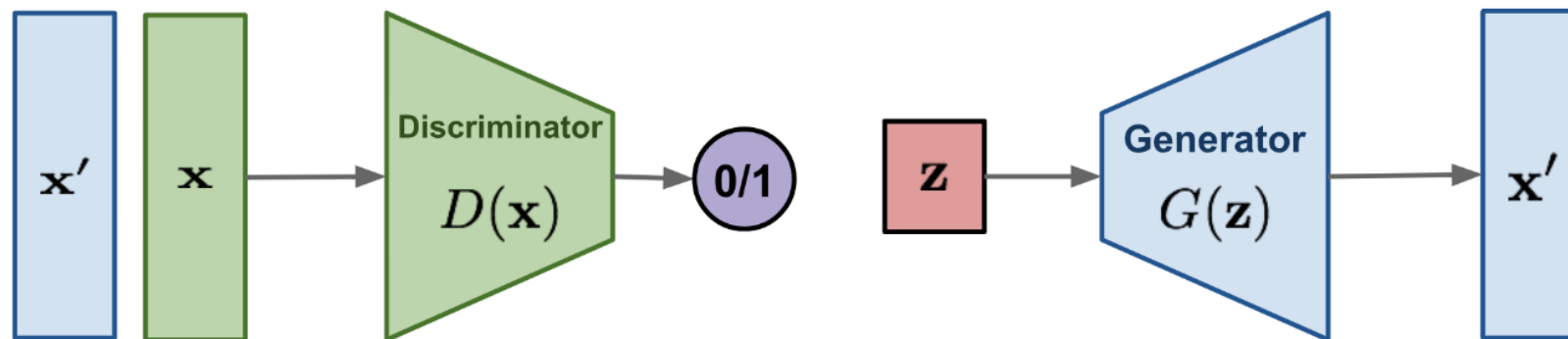
$$\begin{aligned} L_{dL-VAE}(\theta, \phi; \mathbf{x}, \mathbf{z}) = & \frac{1}{1 + \ln(1 + \sigma_0^2)} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) \right] \\ & - \sum_{i=1}^{L+1} \frac{1}{1 + \ln(1 + \sigma_i^2)} D_{KL} \left(q_\phi(\mathbf{z}_{i-1}|\mathbf{x})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

Our Work (Mogultay, Kalkan, Vural, 2024)

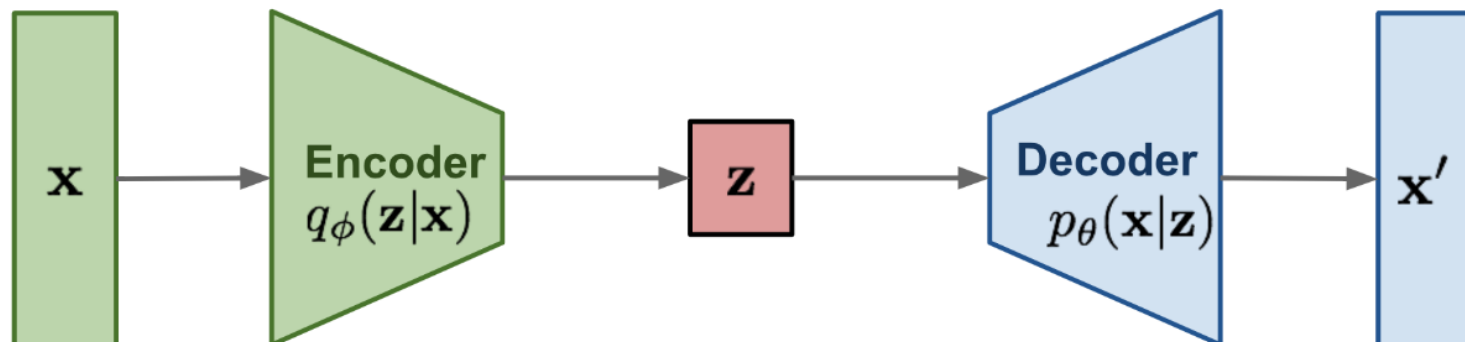
VAE	11.86	0.58	0.93	0.77	0.56	<u>0.63</u>	0.30	0.32	<u>0.92</u>	0.28
β -VAE ($\beta = 4$)	29.11	0.58	0.92	<u>0.75</u>	0.55	0.51	0.30	0.28	0.94	0.26
ControlVAE	24.35	0.60	0.97	0.76	0.59	0.58	0.30	<u>0.30</u>	0.94	0.30
DynamicVAE	33.75	0.56	0.89	0.58	0.50	<u>0.51</u>	0.33	0.29	0.85	<u>0.32</u>
σ -VAE	<u>12.30</u>	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	<u>0.59</u>	<u>0.96</u>	0.77	<u>0.57</u>	0.65	<u>0.31</u>	0.32	<u>0.92</u>	<u>0.32</u>
dL-VAE	19.99	0.60	0.97	0.77	0.59	0.57	<u>0.31</u>	<u>0.30</u>	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

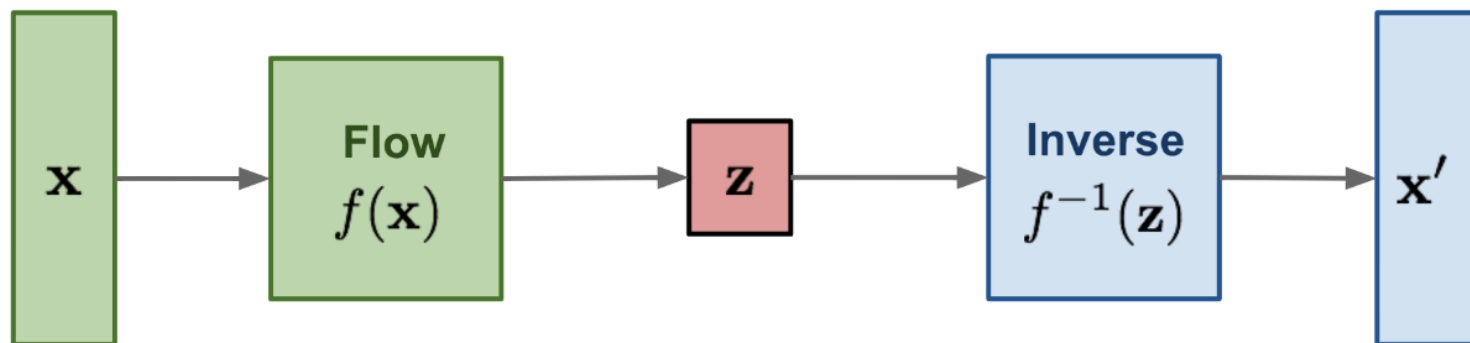


Figure: <https://lilianweng.github.io/posts/2018-10-13-flow-models/>

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, $p(x)$?*

$$\int p(x)dx = \int \pi(z)dz = 1 ; \text{Definition of probability distribution.}$$
$$p(x) = \pi(z) \left| \frac{dz}{dx} \right| = \pi(f^{-1}(x)) \left| \frac{df^{-1}}{dx} \right| = \pi(f^{-1}(x)) |(f^{-1})'(x)|$$

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

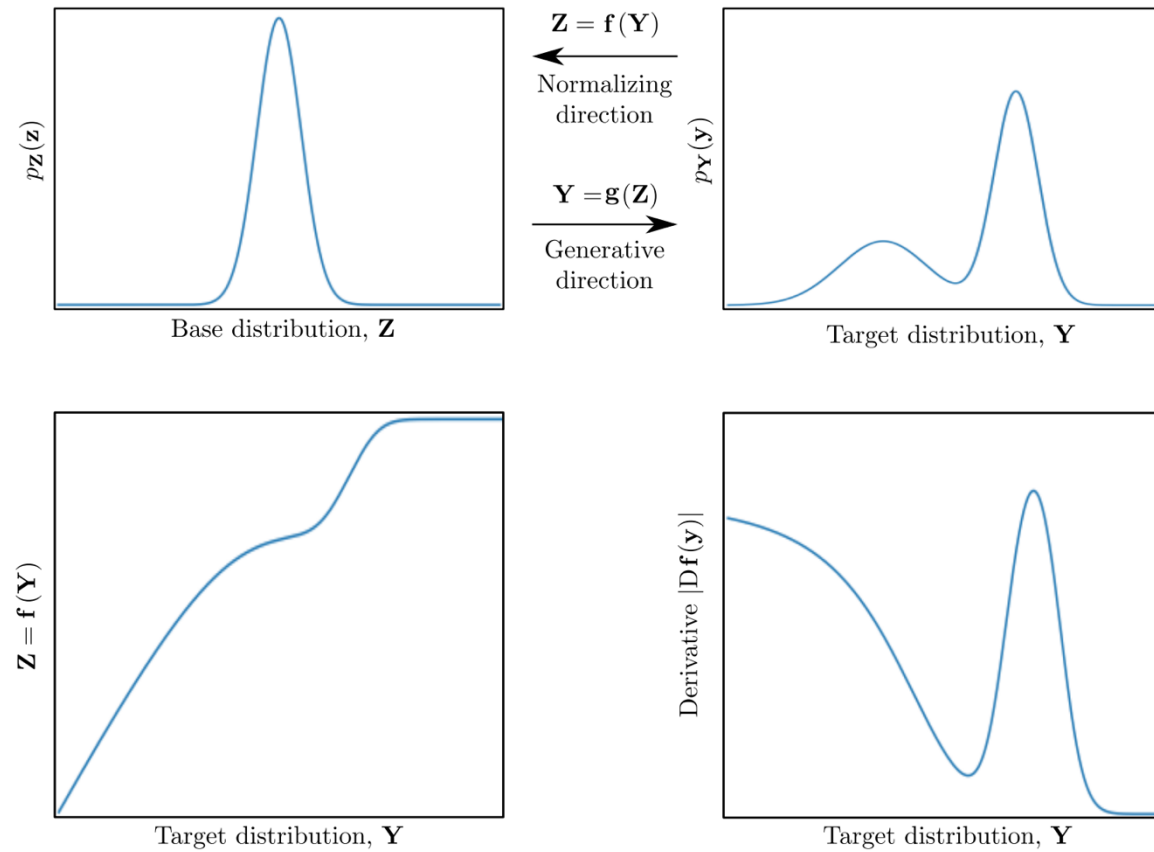


Fig. 1. Change of variables (Equation (1)). Top-left: the density of the source p_Z . Top-right: the density function of the target distribution $p_Y(y)$. There exists a bijective function g , such that $p_Y = g_* p_Z$, with inverse f . Bottom-left: the inverse function f . Bottom-right: the absolute Jacobian (derivative) of f .

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

Normalizing Flow

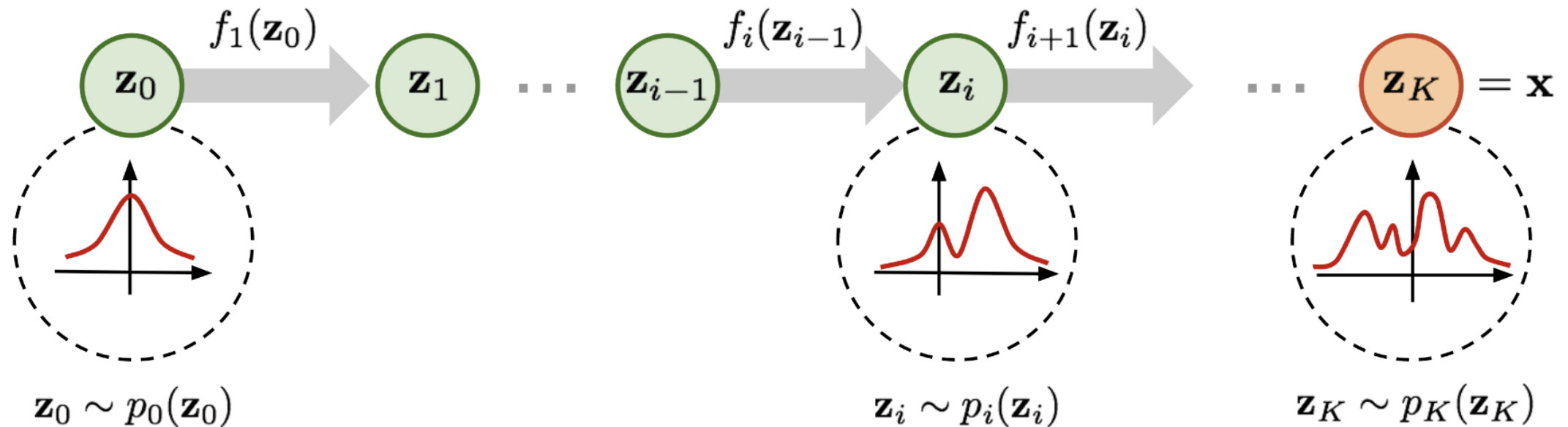


Fig. 2. Illustration of a normalizing flow model, transforming a simple distribution $p_0(\mathbf{z}_0)$ to a complex one $p_K(\mathbf{z}_K)$ step by step.

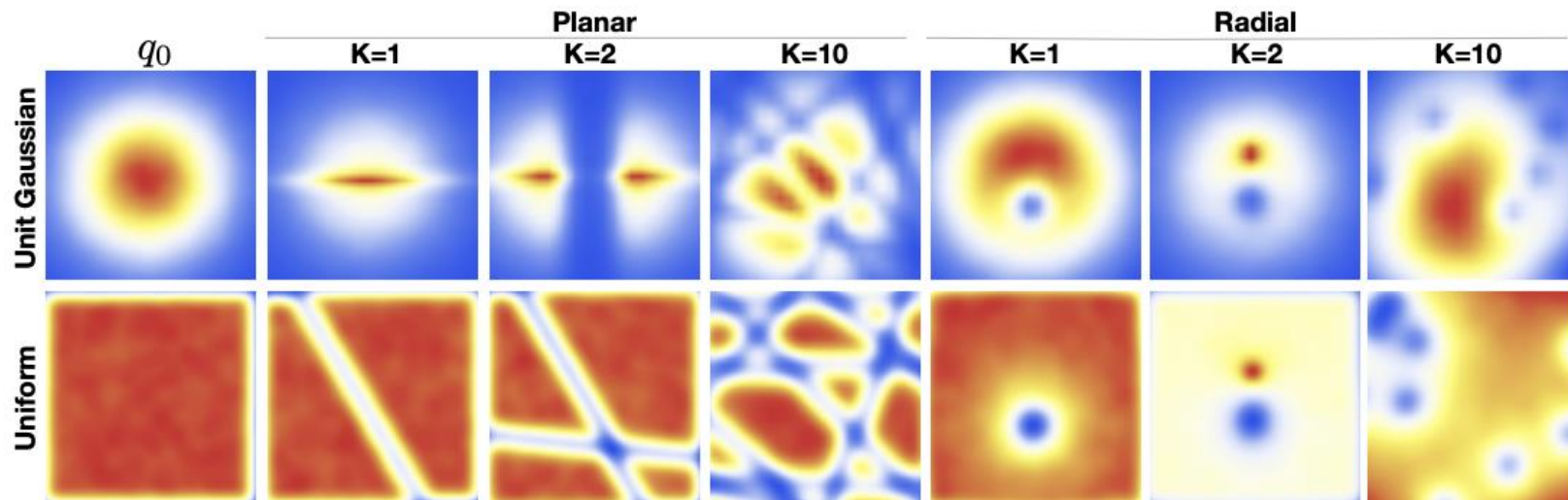


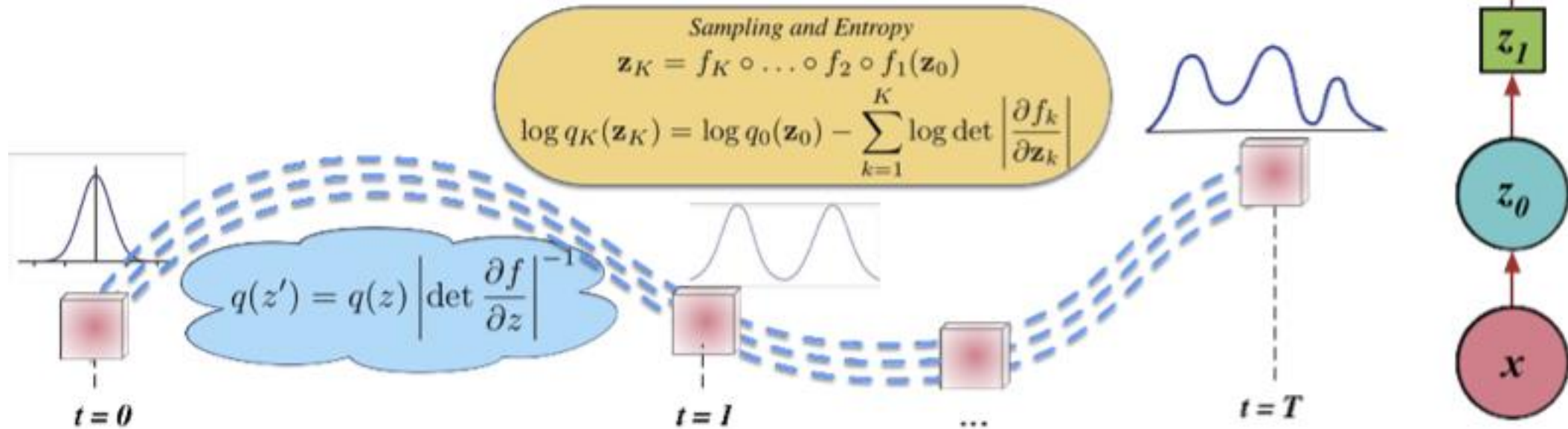
Figure 1. Effect of normalizing flow on two distributions.

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

Normalising Flows

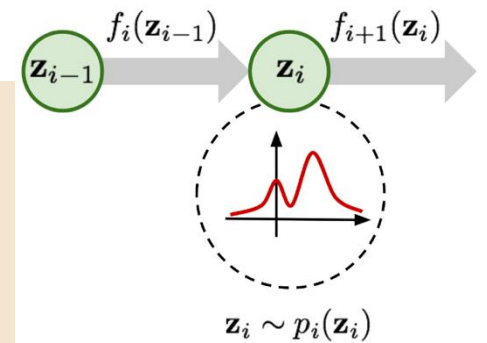
Exploit the rule for change of variables:

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



Distribution flows through a sequence of invertible transforms

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



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

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

$$\begin{aligned} p_i(\mathbf{z}_i) &= p_{i-1}(f_i^{-1}(\mathbf{z}_i)) \left| \det \frac{df_i^{-1}}{d\mathbf{z}_i} \right| \\ &= p_{i-1}(\mathbf{z}_{i-1}) \left| \det \left(\frac{df_i}{d\mathbf{z}_{i-1}} \right)^{-1} \right| \\ &= p_{i-1}(\mathbf{z}_{i-1}) \left| \det \frac{df_i}{d\mathbf{z}_{i-1}} \right|^{-1} \end{aligned}$$

; According to the inverse func theorem.

; According to a property of Jacobians of invertible func.

$$\log p_i(\mathbf{z}_i) = \log p_{i-1}(\mathbf{z}_{i-1}) - \log \left| \det \frac{df_i}{d\mathbf{z}_{i-1}} \right|$$

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 .

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

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.

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

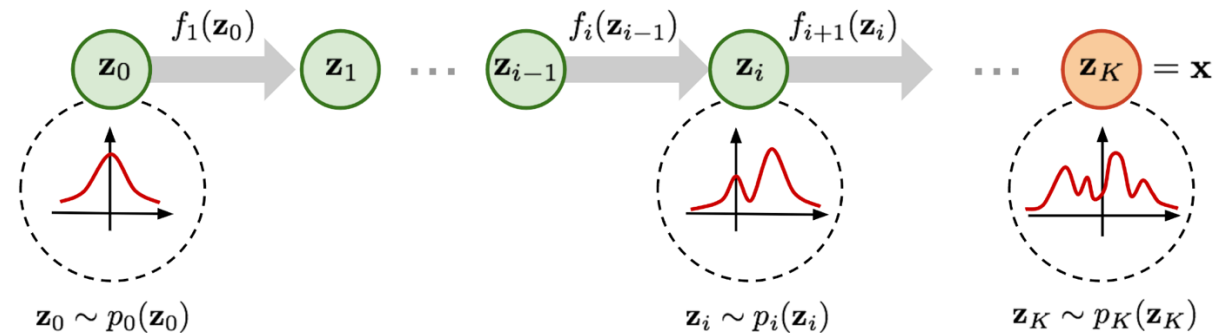
$$\arg \min_{\theta} D_{KL}[p^*(x) || p_\theta(x)] \quad \longrightarrow \quad \arg \max_{\theta} \sum_{i=0}^N \log(p_\theta(x_i))$$

Training

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

Pseudo-code

1. $\mathbf{x} \leftarrow$ Sample a batch
2. $\mathbf{z}_0 \sim p_{\theta}(\mathbf{z}_0 | \mathbf{x})$
3. $\mathbf{z}_K \leftarrow f_K \circ f_{K-1} \circ \dots \circ f_1(\mathbf{z}_0)$
4. $\Delta\theta \propto -\nabla_{\theta} d(\mathbf{x}, \mathbf{z}_K)$



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

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.

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

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.

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

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