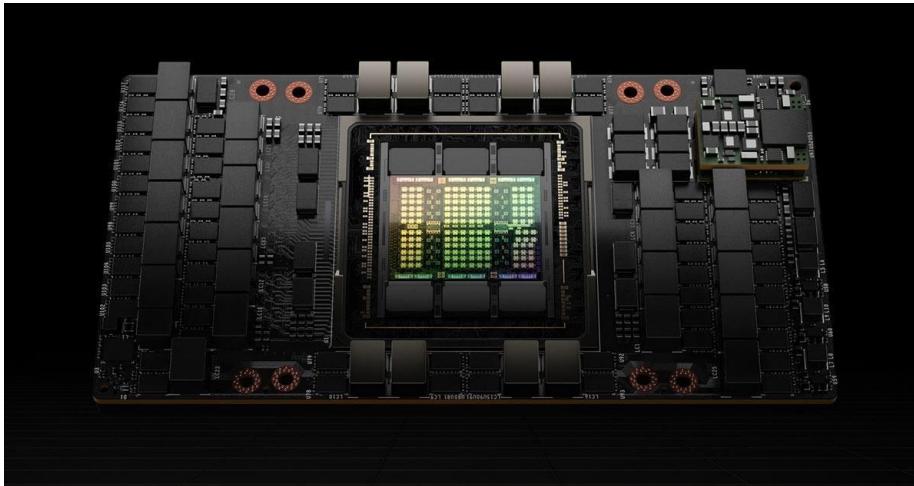


Previous Lecture: GPUs and How to Train On Them

A bit about GPU hardware



How to train on lots of GPUs



Lots of Computer Vision Tasks

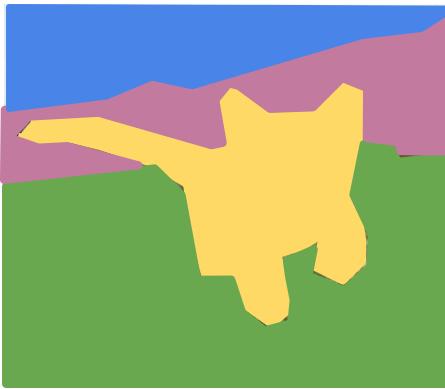
Classification



CAT

No spatial extent

Semantic Segmentation



GRASS, CAT, TREE,
SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Multiple Object

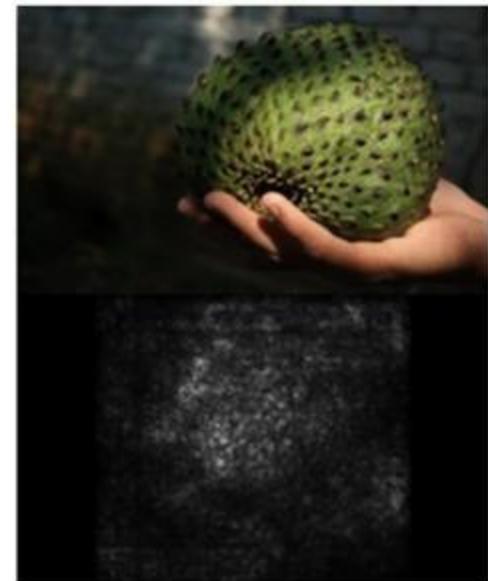
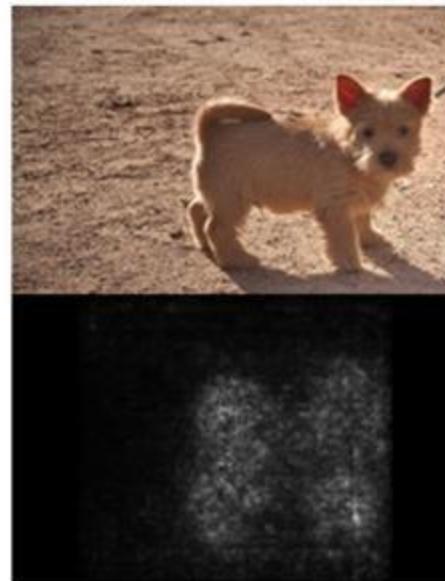
Instance Segmentation



DOG, DOG, CAT

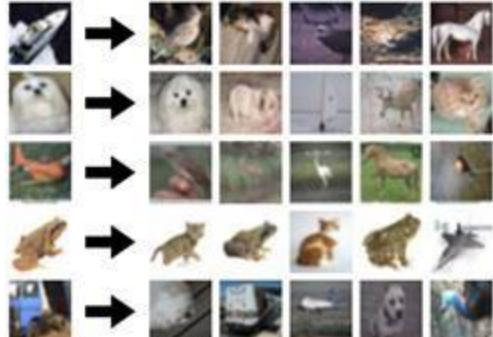
[This image is CC0 public domain](#)

Last Week: Visualizing and Understanding



Last Week: Visualizing and Understanding

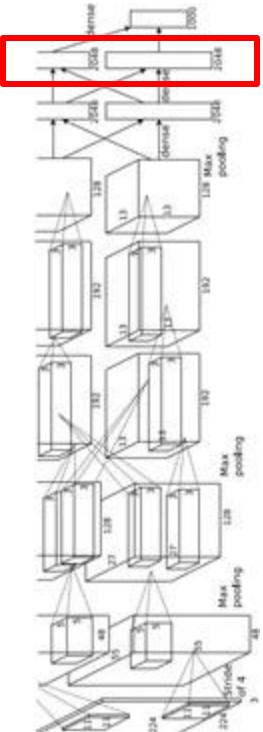
Recall: Nearest neighbors in pixel space



Test image L2 Nearest neighbors in feature space



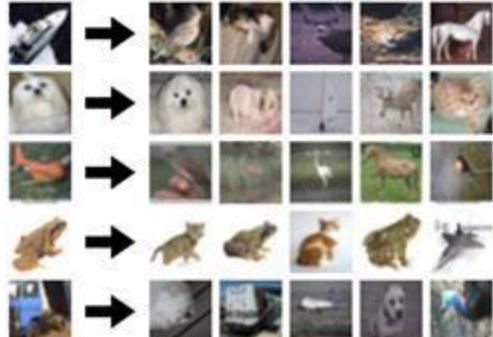
4096-dim vector



Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figures reproduced with permission.

Learned Representations

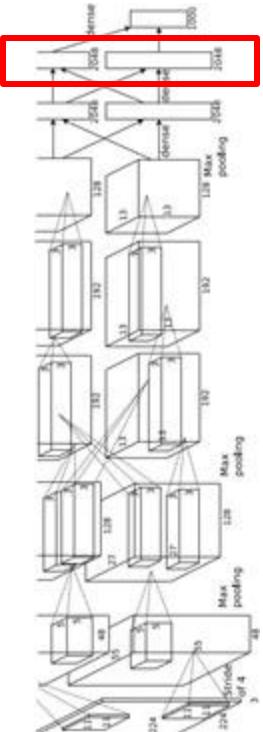
Recall: Nearest neighbors in pixel space



Test image L2 Nearest neighbors in feature space

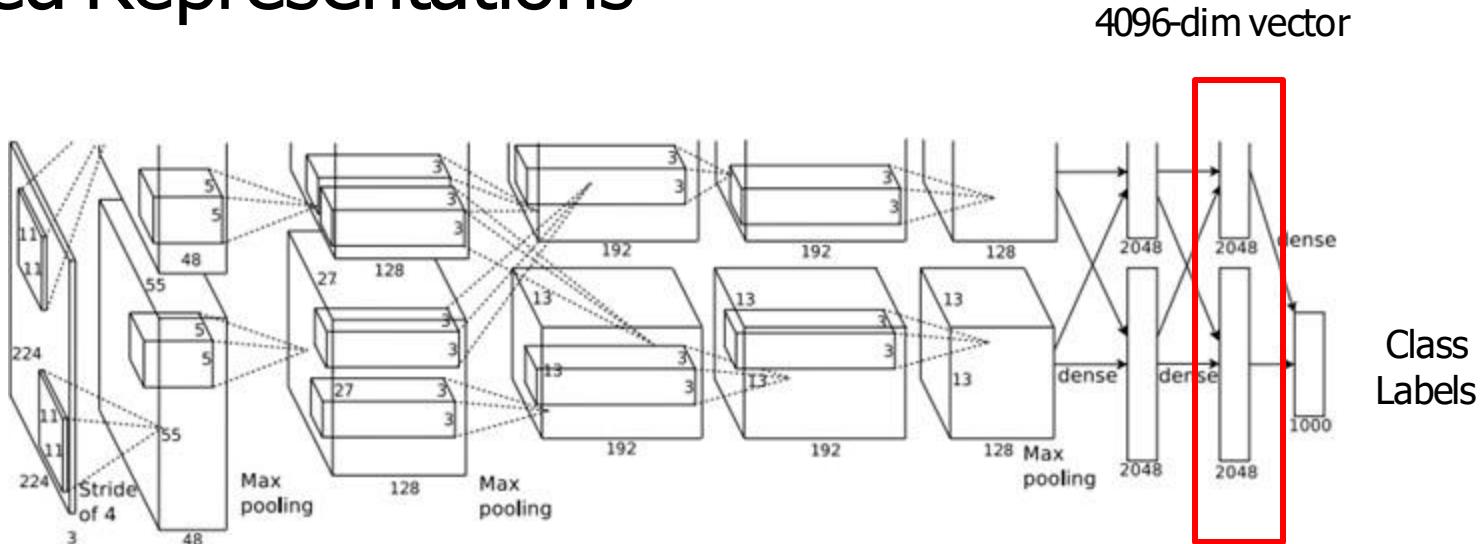


4096-dim vector



Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figures reproduced with permission.

Learned Representations

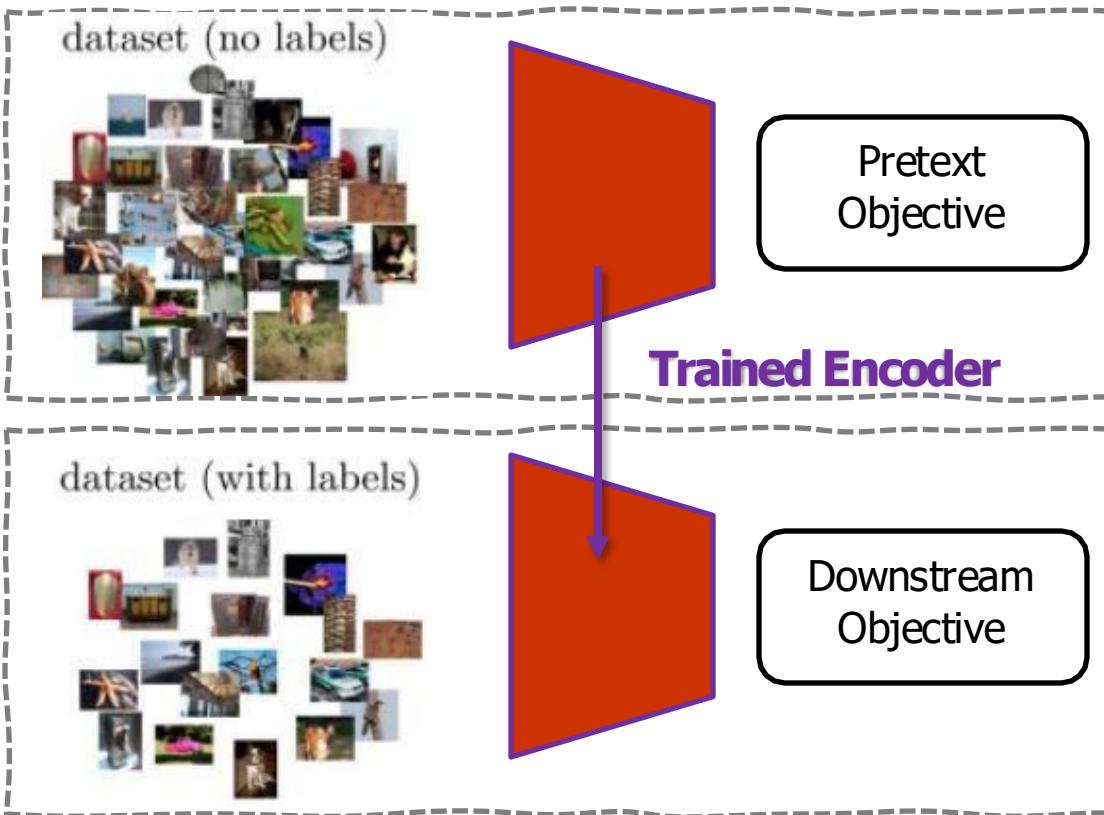


What is the problem with large-scale training?

- **We need a lot of labeled data**

Is there a way we can train neural networks without the need for huge manually labeled datasets?

Self-Supervised Learning



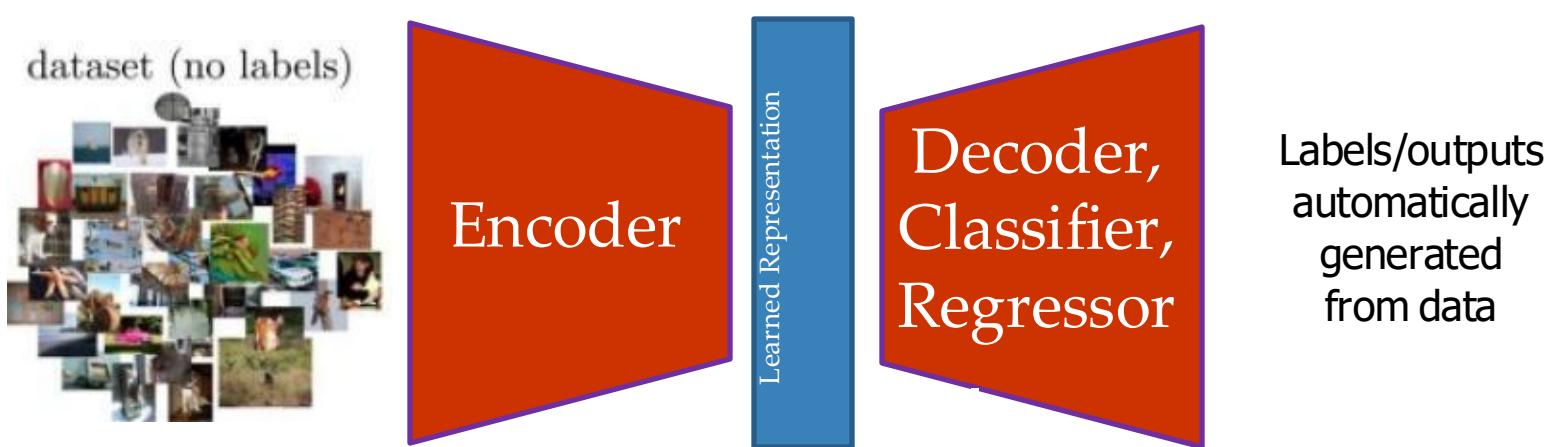
Pretext Task

- Define a task based on the data itself
- No manual annotation
- Could be considered an **unsupervised** task;
- but we learn with supervised learning objectives, e.g., classification or regression.

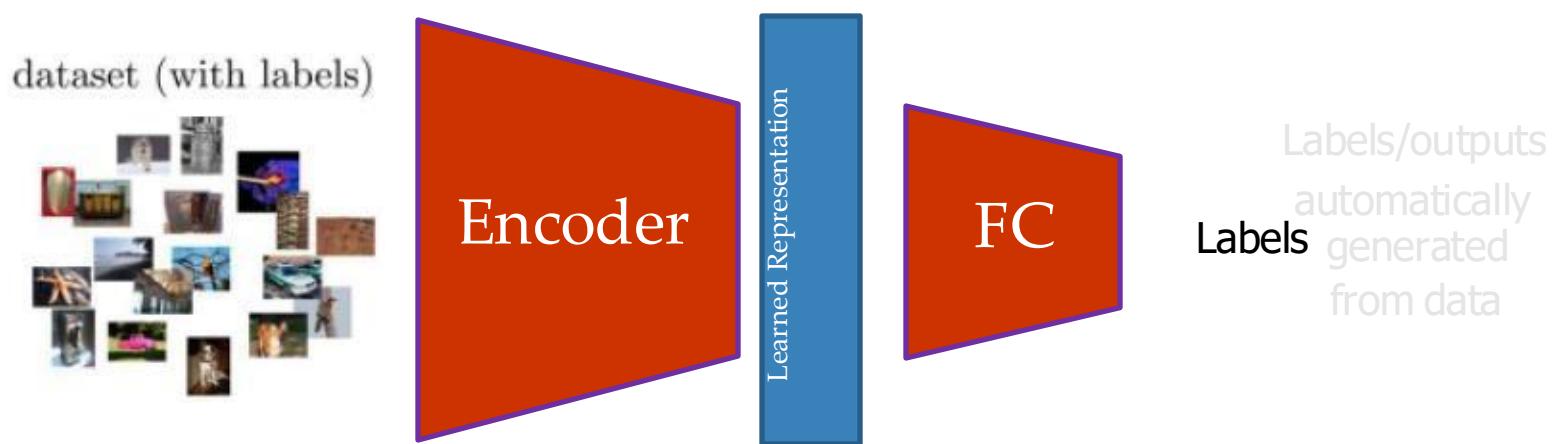
Downstream Task

- The application you care about
- You do not have large datasets
- The dataset is labeled

Self-Supervised Learning – Pretext Task



Self-Supervised Learning – Downstream Task



Self-supervised pretext tasks

Example: learn to predict image transformations / complete corrupted images

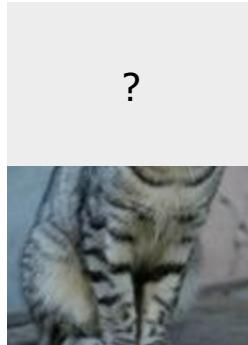
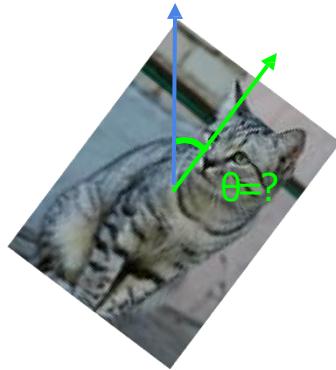


image completion



rotation prediction



"jigsaw puzzle"



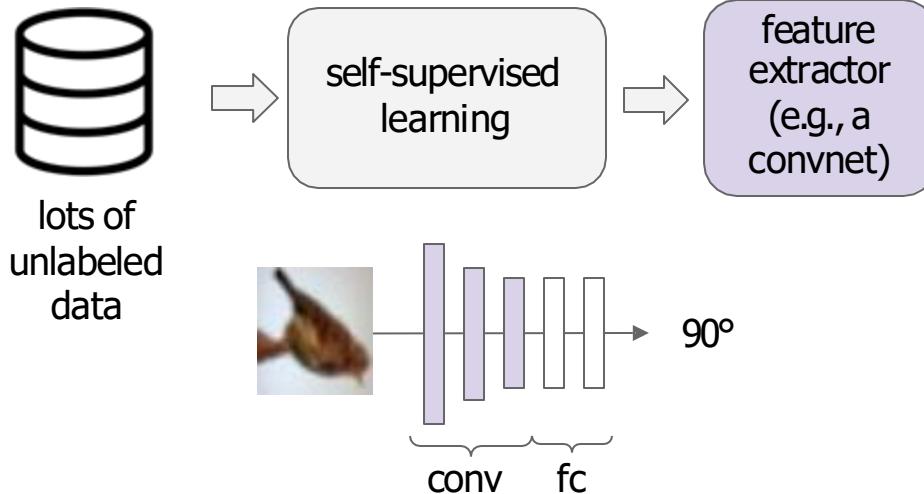
colorization

1. Solving the pretext tasks allow the model to learn good features.
2. We can automatically generate labels for the pretext tasks.

How to evaluate a self-supervised learning method?

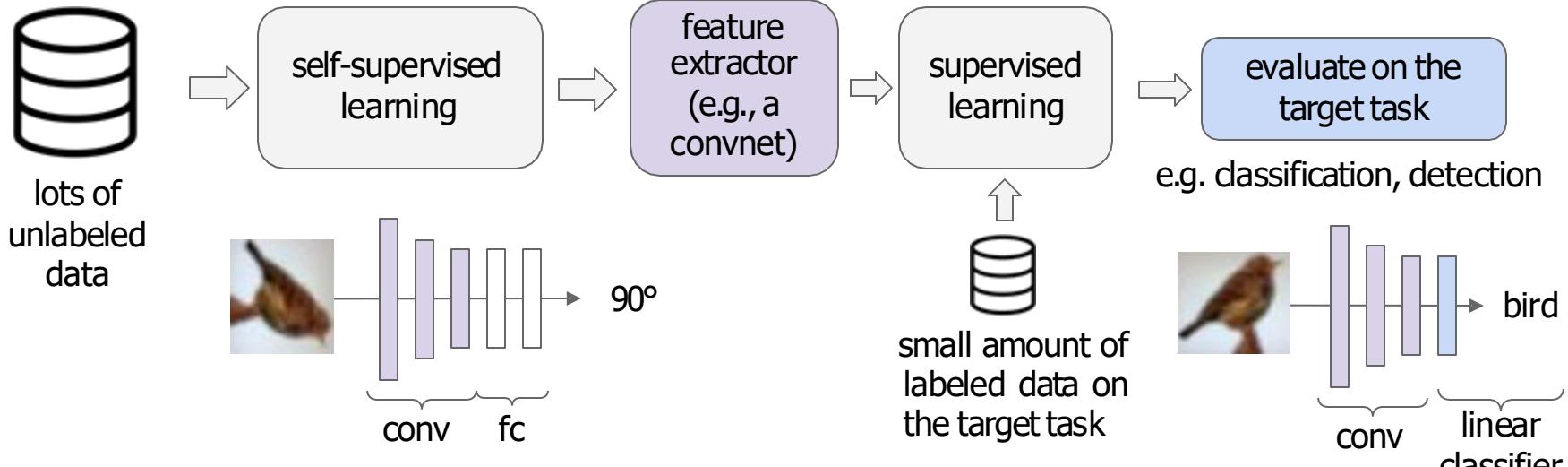
- **Pretext Task Performance**
 - Measure how well the model performs on the task it was trained on without labels.
- **Representation Quality**
 - Evaluate the quality of the learned representations
 - *Linear Evaluation Protocol*: Train a linear classifier on the learned representations;
 - *Clustering*: Measure clustering performance;
 - *t-SNE*: Visualize the representations to assess their separability.)
- **Robustness and Generalization**
 - Test how well the model generalizes to different datasets and is robust to variations.
- **Computational Efficiency**
 - Assess the efficiency of the method in terms of training time and resource requirements.
- **Transfer Learning and Downstream Task Performance**
 - Assess the utility of the learned representations by transferring them to a downstream supervised task.

How to evaluate a self-supervised learning method?



1. Learn good feature extractors from self-supervised pretext tasks, e.g., predicting image rotations

How to evaluate a self-supervised learning method?



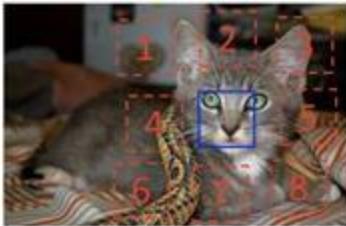
1. Learn good feature extractors from self-supervised pretext tasks, e.g., predicting image rotations

2. Attach a shallow network on the feature extractor; train the shallow network on the target task with small amount of labeled data

Broader picture

Today's lecture

computer vision



Doersch et al., 2015

robot / reinforcement learning



Dense Object Net (Florence and Manuelli et al., 2018)

language modeling

GPT-4 Technical Report

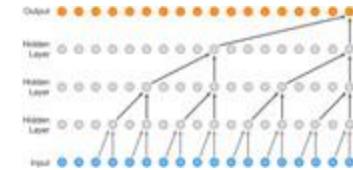
OpenAI*

Abstract

We report the development of GPT-4, a large-scale, multimodal model which can accept image and text inputs and produce text outputs. While less capable than humans in many real-world scenarios, GPT-4 exhibits human-level performance on various professional and academic benchmarks, including passing a simulated bar exam with a score around the top 10% of test takers. GPT-4 is a Transformer-based model pre-trained to predict the next token in a document. The post-training alignment process results in improved performance on measures of factuality and adherence to desired behavior. A core component of this project was developing infrastructure and optimization methods that behave predictably across a wide range of scales. This allowed us to accurately predict some aspects of GPT-4's performance based on models trained with no more than 1/1,000th the compute of GPT-4.

GPT-4 (OpenAI 2023)

speech synthesis



Wavenet (van den Oord et al., 2016)

■ ■ ■

Today's Agenda

Pretext tasks from image transformations

- Rotation, inpainting, rearrangement, coloring
- Reconstruction-based learning (MAE)

Contrastive representation learning

- Intuition and formulation
- Instance contrastive learning: SimCLR and MOCO
- Sequence contrastive learning: CPC
- Self-Distillation Without Labels, DINO

Today's Agenda

Pretext tasks from image transformations

- Rotation, inpainting, rearrangement, coloring
- Reconstruction-based learning (MAE)

Contrastive representation learning

- Intuition and formulation
- Instance contrastive learning: SimCLR and MOCO
- Sequence contrastive learning: CPC
- Self-Distillation Without Labels, DINO

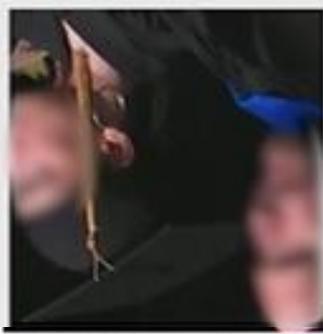
Pretext task: predict rotations



90° rotation



270° rotation



180° rotation



0° rotation

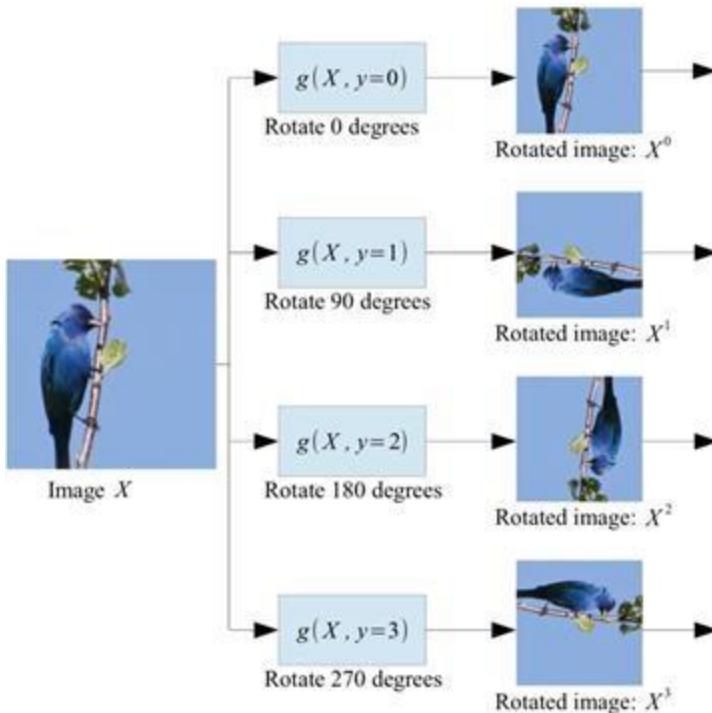


270° rotation

Hypothesis: a model could recognize the correct rotation of an object only if it has the “visual commonsense” of what the object should look like unperturbed.

(Image source: [Gidaris et al. 2018](#))

Pretext task: predict rotations

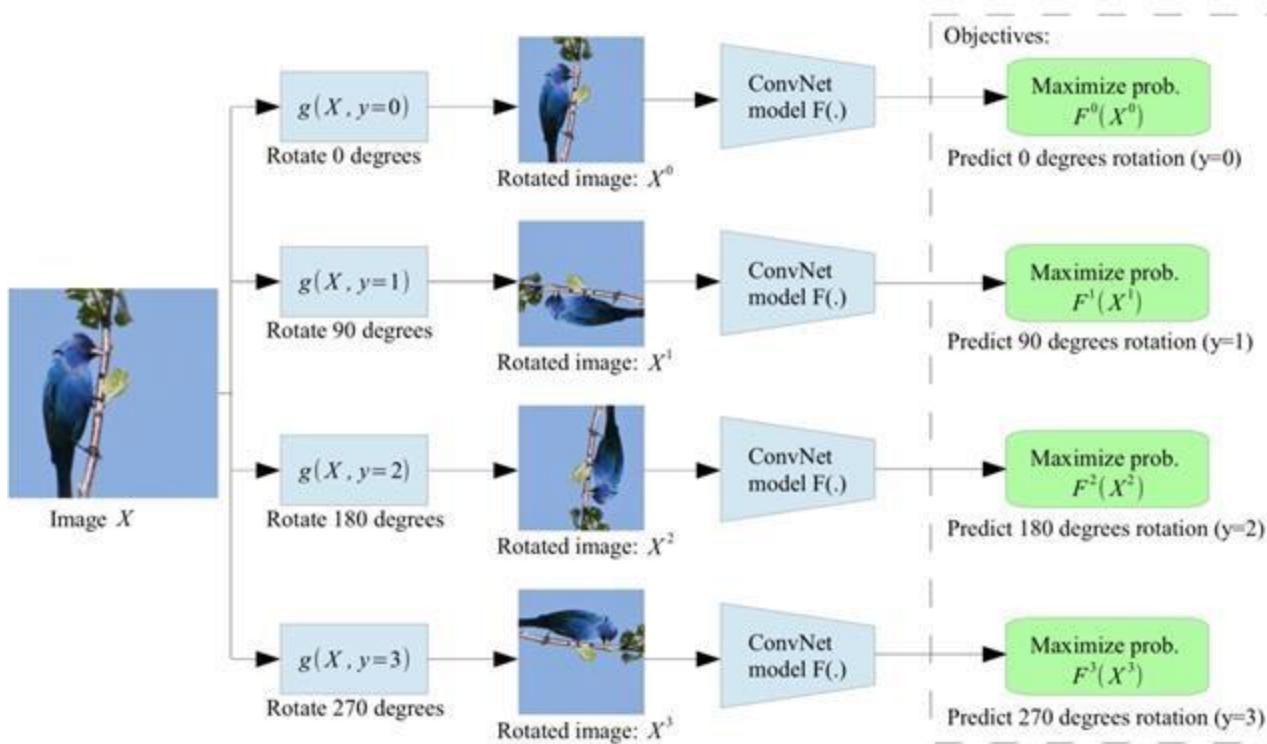


Self-supervised learning by rotating the entire input images.

The model learns to predict which rotation is applied (4-way classification)

(Image source: [Gidaris et al. 2018](#))

Pretext task: predict rotations

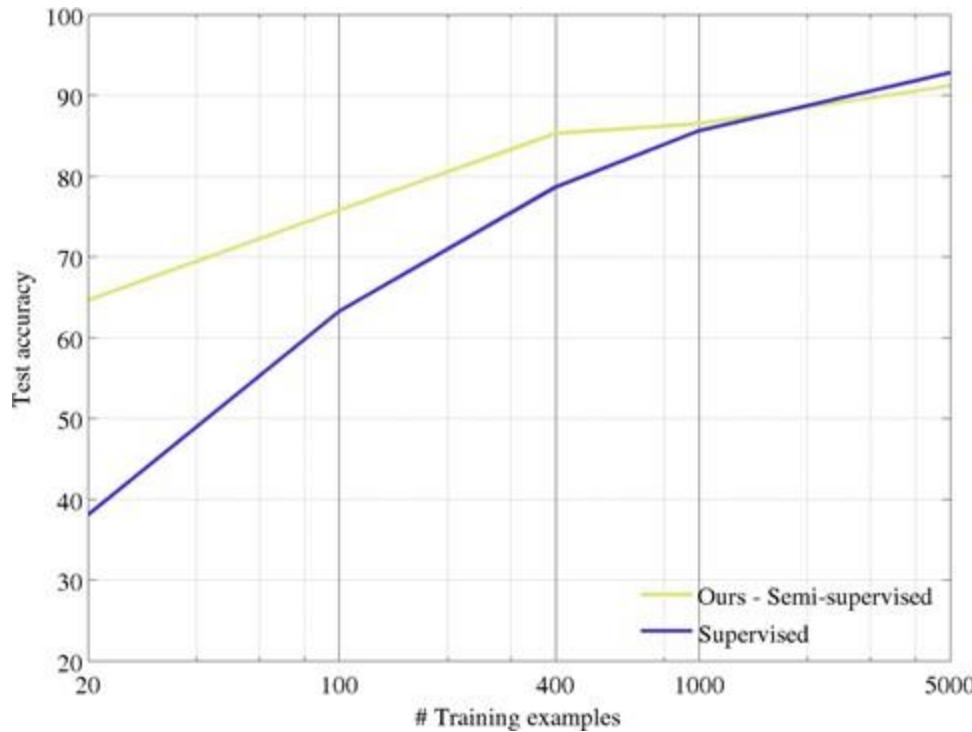


Self-supervised learning by rotating the entire input images.

The model learns to predict which rotation is applied (4-way classification)

(Image source: [Gidaris et al. 2018](#))

Evaluation on semi-supervised learning



Self-supervised learning on CIFAR10 (entire training set).

Freeze conv1 +conv2
Learn conv3 +linear layers with subset of labeled CIFAR10 data (classification).

(Image source: [Gidaris et al. 2018](#))

Transfer learned features to supervised learning

	Classification (%mAP)	Detection (%mAP)	Segmentation (%mIoU)
Trained layers	fc6-8	all	all
ImageNet labels	78.9	79.9	56.8
Random		53.3	43.4
Random rescaled Krähenbühl et al. (2015)	39.2	56.6	45.6
Egomotion (Agrawal et al., 2015)	31.0	54.2	43.9
Context Encoders (Pathak et al., 2016b)	34.6	56.5	44.5
Tracking (Wang & Gupta, 2015)	55.6	63.1	47.4
Context (Doersch et al., 2015)	55.1	65.3	51.1
Colorization (Zhang et al., 2016a)	61.5	65.6	46.9
BIGAN (Donahue et al., 2016)	52.3	60.1	46.9
Jigsaw Puzzles (Noroozi & Favaro, 2016)	-	67.6	53.2
NAT (Bojanowski & Joulin, 2017)	56.7	65.3	49.4
Split-Brain (Zhang et al., 2016b)	63.0	67.1	46.7
ColorProxy (Larsson et al., 2017)		65.9	
Counting (Noroozi et al., 2017)	-	67.7	51.4
(Ours) RotNet	70.87	72.97	54.4
			39.1

Self-supervised learning with rotation prediction

Pretrained with full ImageNet supervision

No pretraining

Self-supervised learning on ImageNet (entire training set) with AlexNet.

Finetune on labeled data from Pascal VOC 2007.

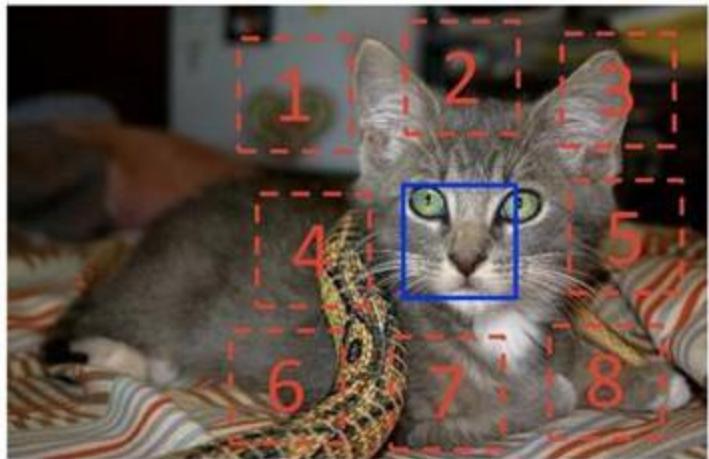
source: [Gidaris et al. 2018](#)

Visualize learned visual attentions



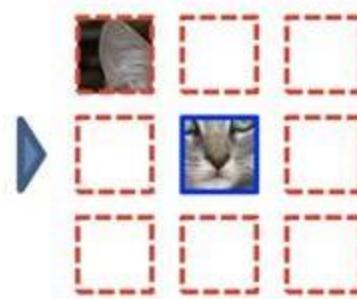
(Image source: [Gidaris et al. 2018](#))

Pretext task: predict relative patch locations



$$X = (\text{cat face}, \text{snake eye}); Y = 3$$

Example:



Question 1:

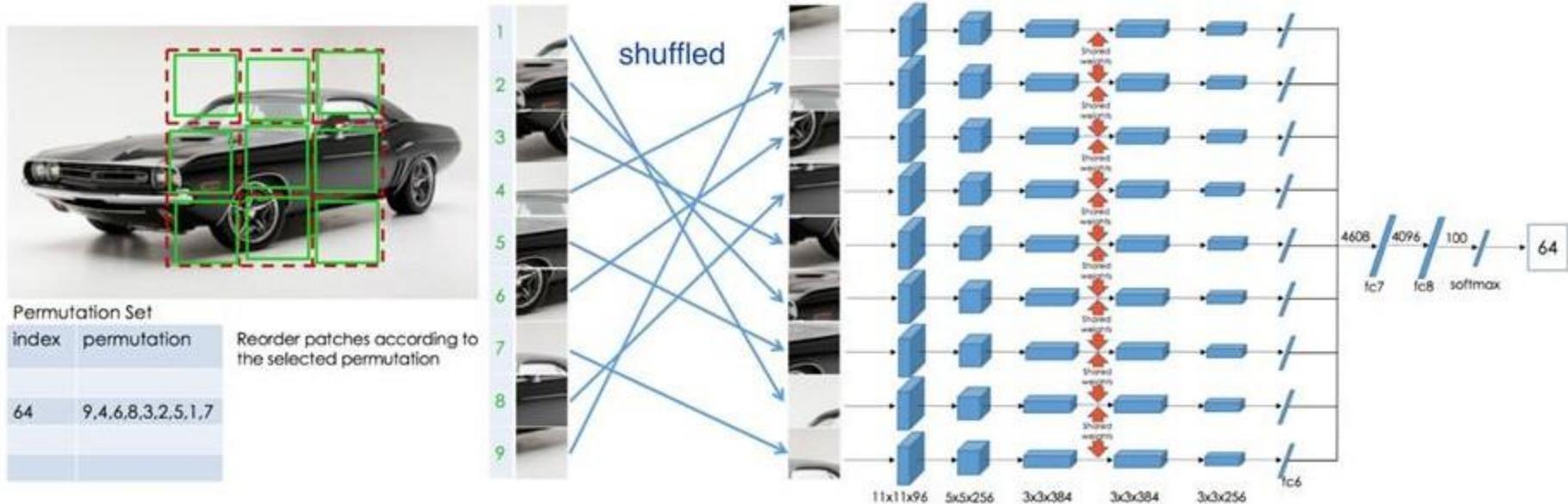


Question 2:



(Image source: [Doersch et al., 2015](#))

Pretext task: solving “jigsaw puzzles”



(Image source: [Noroozi & Favaro, 2016](#))

Transfer learned features to supervised learning

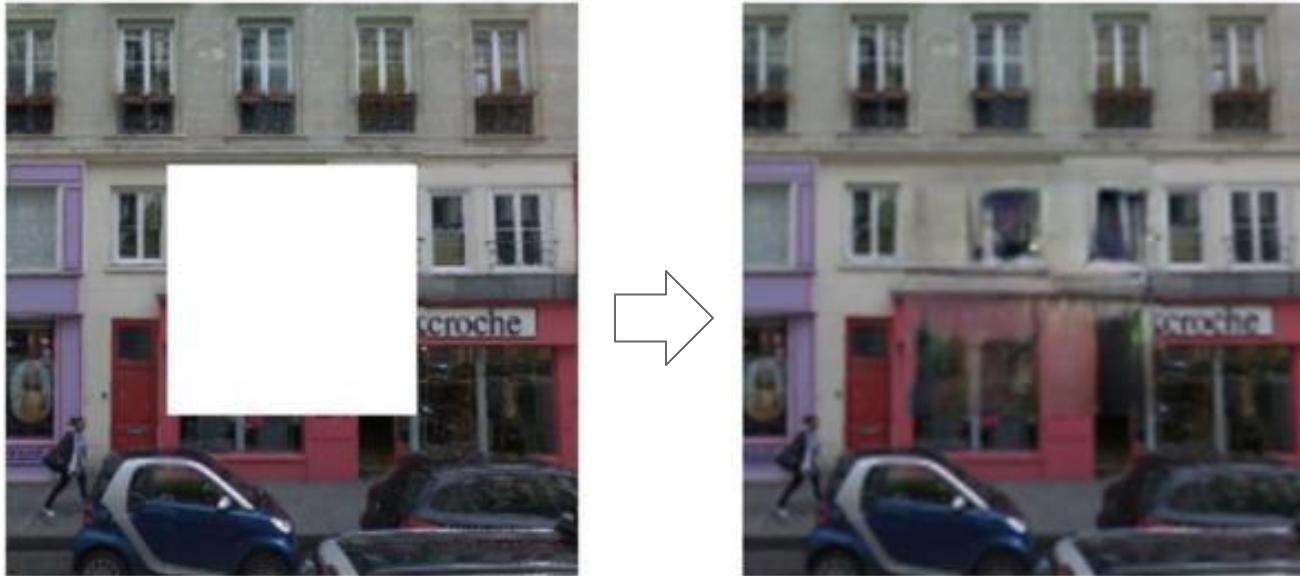
Table 1: Results on PASCAL VOC 2007 Detection and Classification. The results of the other methods are taken from Pathak *et al.* [30].

Method	Pretraining time	Supervision	Classification	Detection	Segmentation
Krizhevsky <i>et al.</i> [25]	3 days	1000 class labels	78.2%	56.8%	48.0%
Wang and Gupta [39]	1 week	motion	58.4%	44.0%	-
Doersch <i>et al.</i> [10]	4 weeks	context	55.3%	46.6%	-
Pathak <i>et al.</i> [30]	14 hours	context	56.5%	44.5%	29.7%
Ours	2.5 days	context	67.6%	53.2%	37.6%

“Ours” is feature learned from solving image Jigsaw puzzles (Noroozi & Favaro, 2016). Doersch et al. is the method with relative patch location

(source: [Noroozi & Favaro, 2016](#))

Pretext task: predict missing pixels (inpainting)

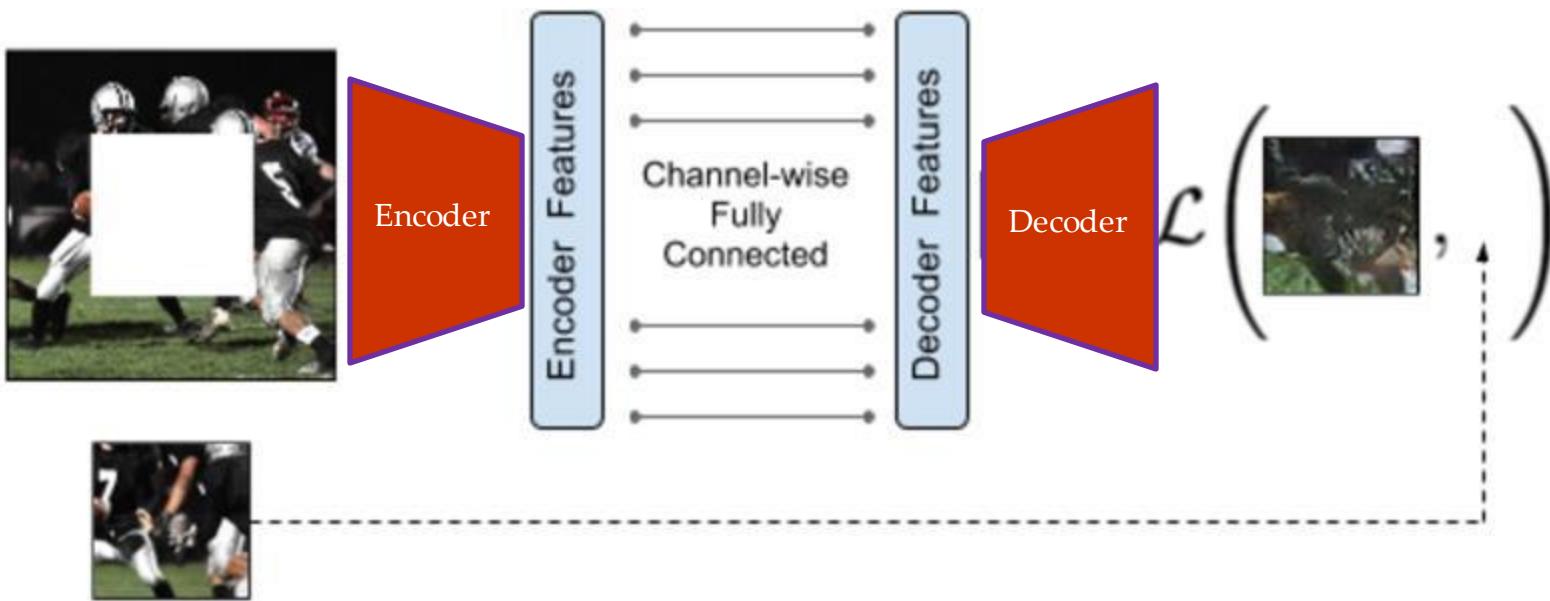


Context Encoders: Feature Learning by Inpainting (Pathak et al., 2016)

Source: [Pathak et al., 2016](#)

Learning to inpaint by reconstruction

Auto Encoder



Learning to reconstruct the missing pixels

Source: [Pathak et al., 2016](#)

Inpainting evaluation



Input (context)



reconstruction

Source: [Pathak et al., 2016](#)

Learning to inpaint by reconstruction

(We will talk about adversarial learning in the next lecture)

Loss =reconstruction +adversarial learning

$$L(x) = L_{recon}(x) + L_{adv}(x)$$

$$L_{recon}(x) = \|M * (x - F_\theta((1 - M) * x))\|_2^2$$

$$L_{adv} = \max_D \mathbb{E}[\log(D(x))] + \log(1 - D(F((1 - M) * x)))]$$

Adversarial loss between “real” images and inpainted images

Element wise multiplication

$M = \phi_1^0$ not masked
masked

Encoder

Inpainting evaluation



Input (context)



reconstruction



adversarial



recon +adv

Source: [Pathak et al., 2016](#)

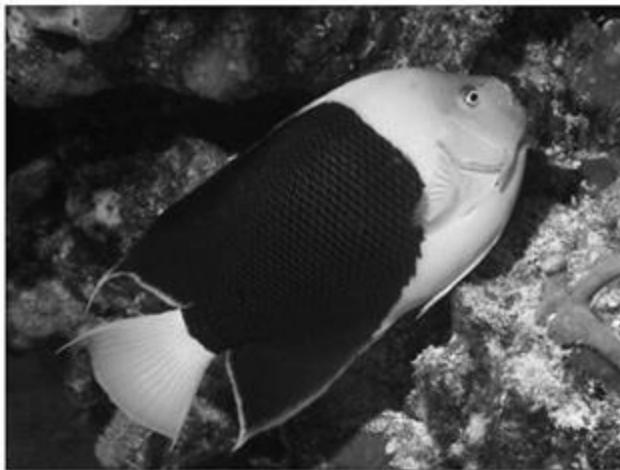
Transfer learned features to supervised learning

Pretraining Method	Supervision	Pretraining time	Classification	Detection	Segmentation
ImageNet [26]	1000 class labels	3 days	78.2%	56.8%	48.0%
Random Gaussian	initialization	< 1 minute	53.3%	43.4%	19.8%
Autoencoder	-	14 hours	53.8%	41.9%	25.2%
Agrawal <i>et al.</i> [1]	egomotion	10 hours	52.9%	41.8%	-
Wang <i>et al.</i> [39]	motion	1 week	58.7%	47.4%	-
Doersch <i>et al.</i> [7]	relative context	4 weeks	55.3%	46.6%	-
Ours	context	14 hours	56.5%	44.5%	30.0%

Self-supervised learning on ImageNet training set, transfer to classification (Pascal VOC 2007), detection (Pascal VOC 2007), and semantic segmentation (Pascal VOC 2012)

Source: [Pathak et al., 2016](#)

Pretext task: image coloring

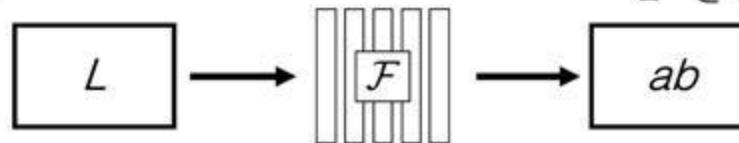


Grayscale image: L channel

$$\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$$

Color information: ab channels

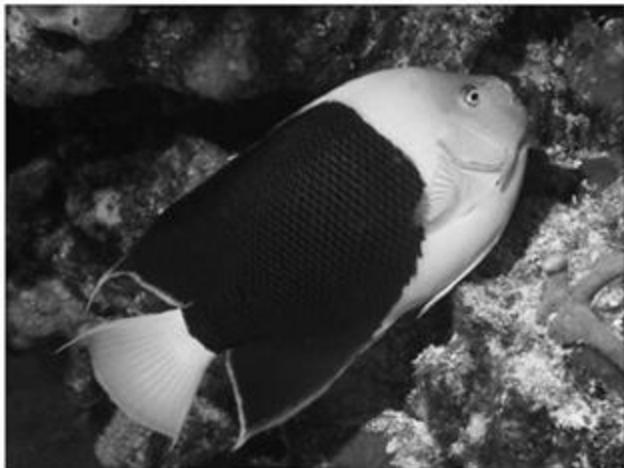
$$\hat{\mathbf{Y}} \in \mathbb{R}^{H \times W \times 2}$$



5

Source: Richard Zhang / Phillip Isola

Pretext task: image coloring

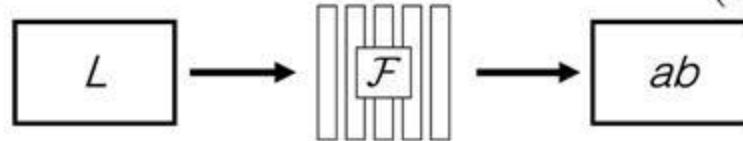


Grayscale image: L channel

$$\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$$

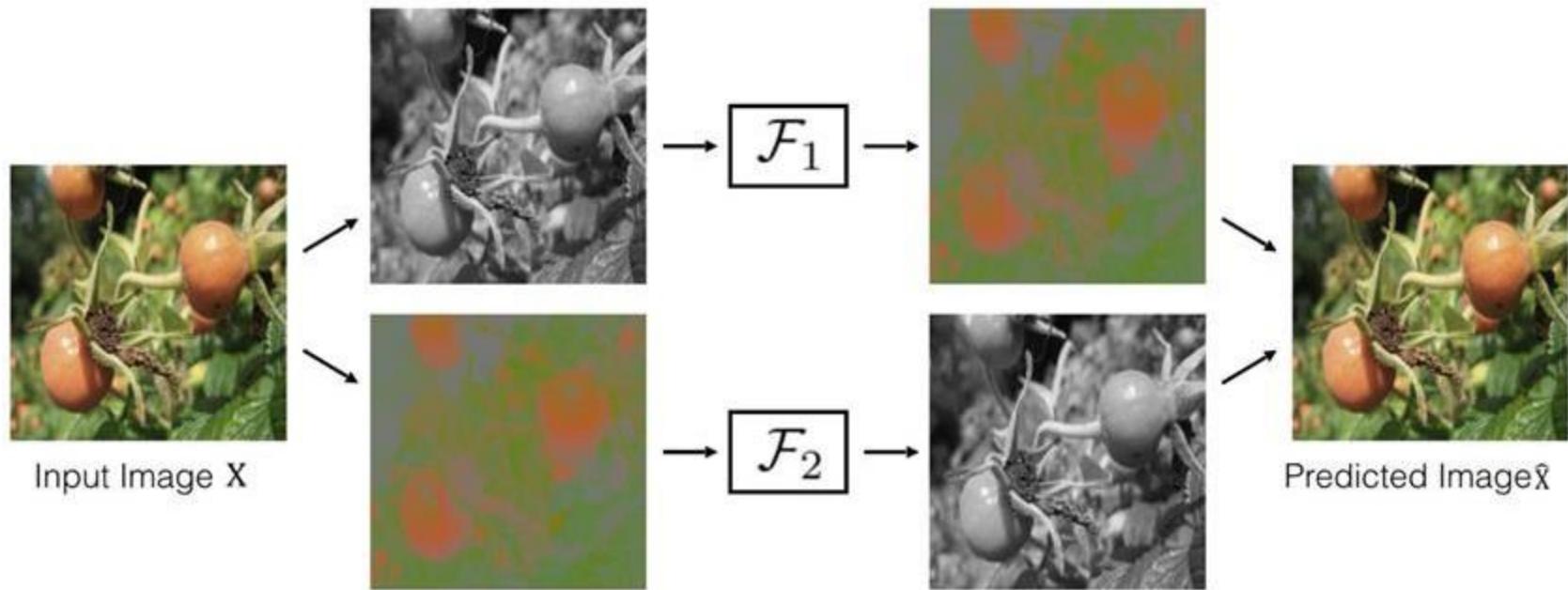
Concatenate (L, ab) channels

$$(\mathbf{X}, \hat{\mathbf{Y}})$$



Source: Richard Zhang / Phillip Isola

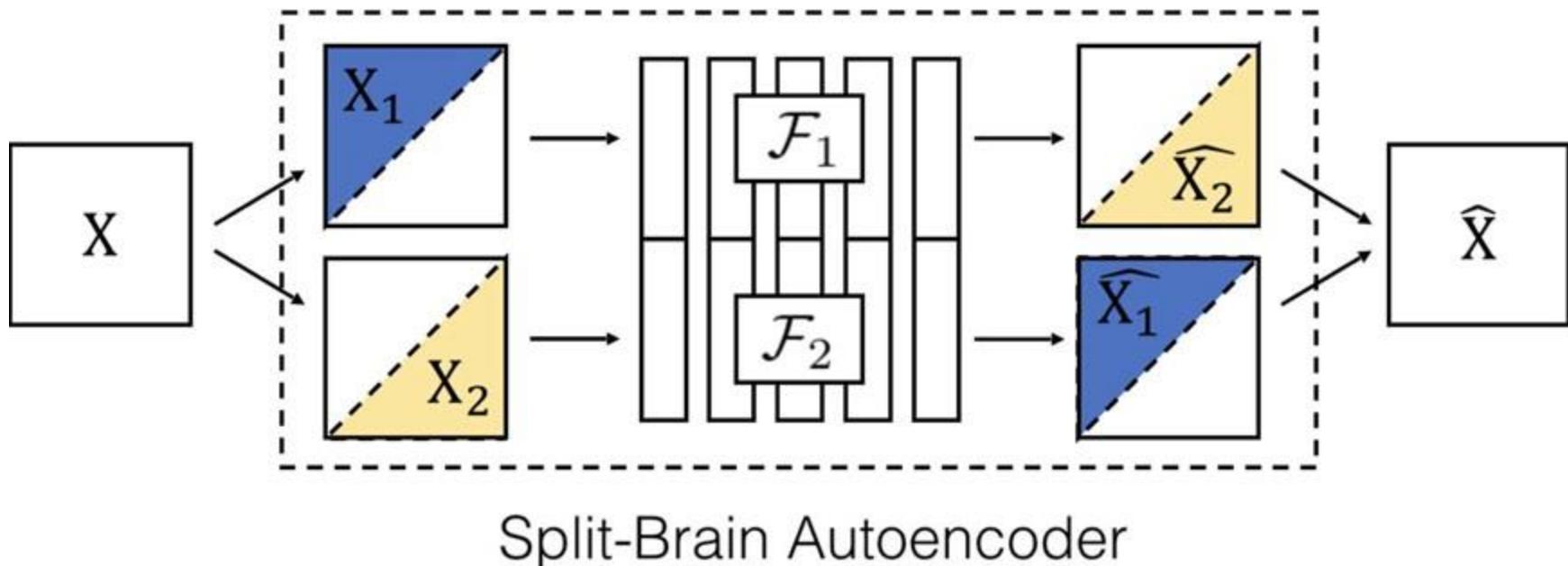
Learning features from colorization: Split-brain Autoencoder



Source: Richard Zhang / Phillip Isola

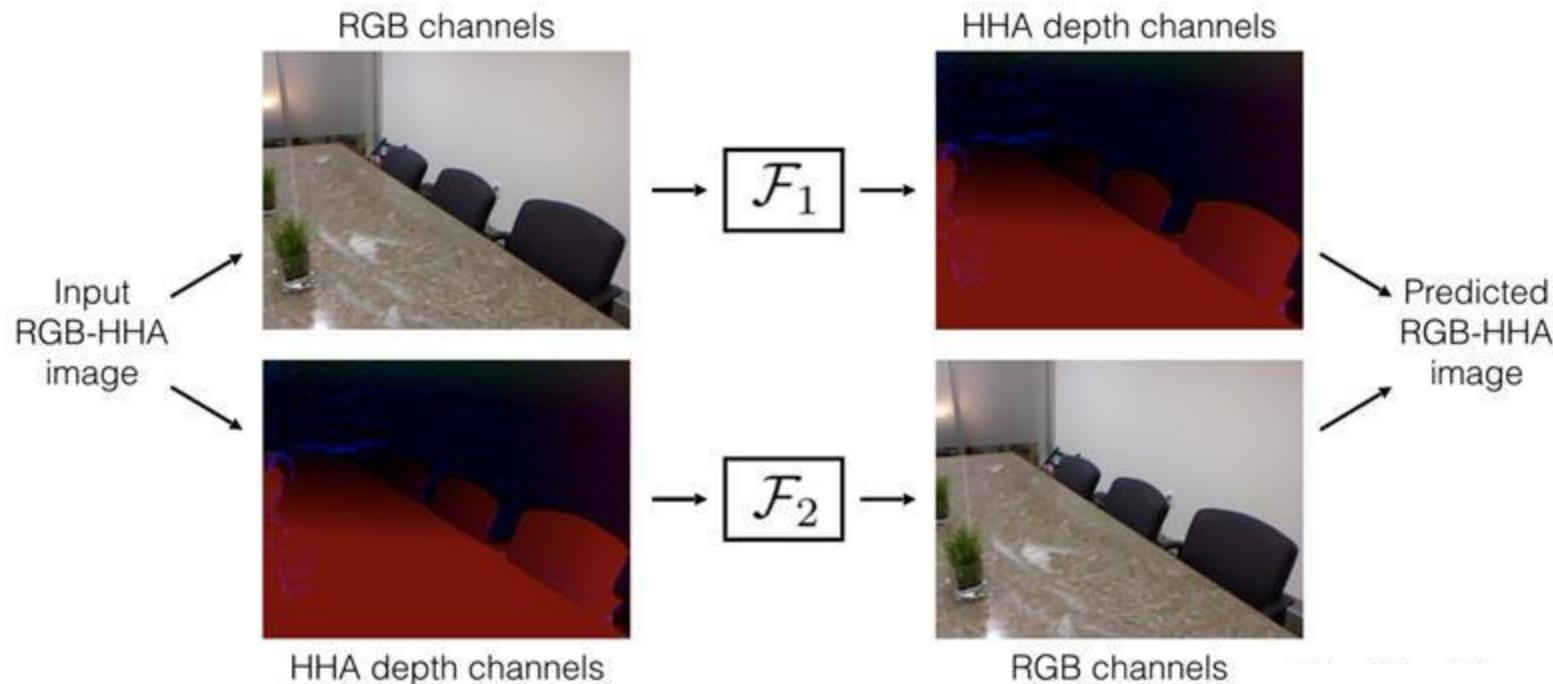
Learning features from colorization: Split-brain Autoencoder

Idea: cross-channel predictions



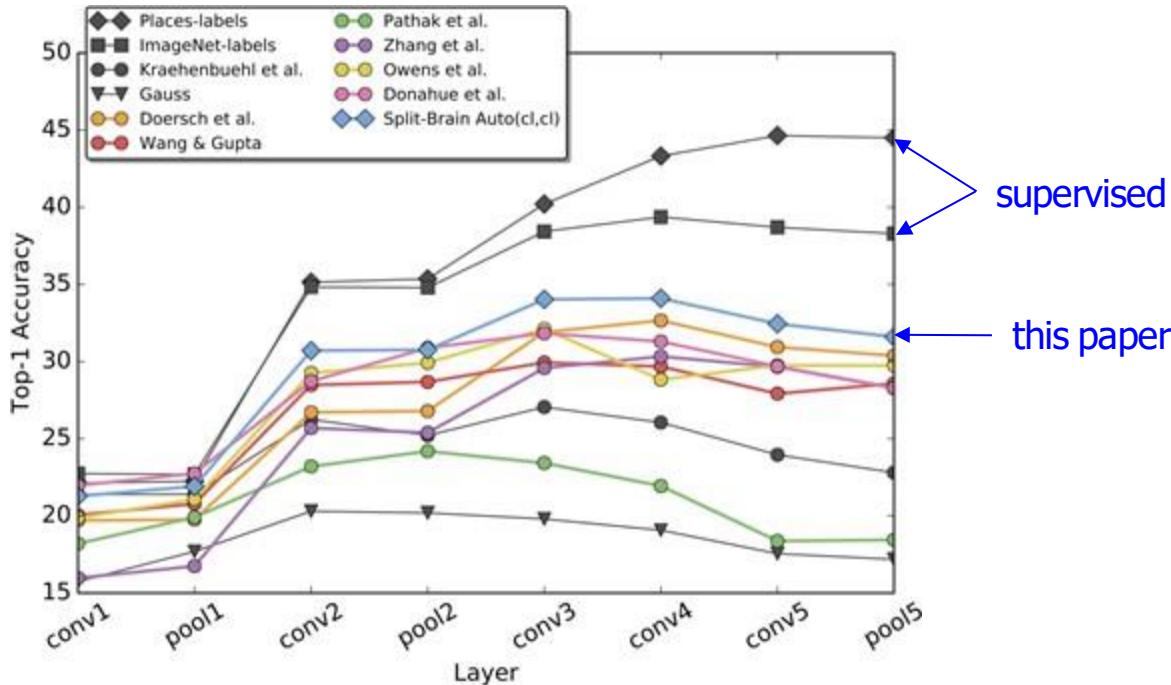
Source: Richard Zhang / Phillip Isola

Learning features from colorization: Split-brain Autoencoder



Source: Richard Zhang / Phillip Isola

Transfer learned features to supervised learning



Self-supervised learning on ImageNet (entire training set).

Use concatenated features from F_1 and F_2

Labeled data is from the Places (Zhou 2016).

Source: [Zhang et al., 2017](#)

Pretext task: image coloring



Source: Richard Zhang / Phillip Isola

Pretext task: image coloring



Source: Richard Zhang / Phillip Isola

Pretext task: video coloring

Idea: model the temporal coherence of colors in videos

reference frame



$t=0$

how should I color these frames?



$t=1$



$t=2$



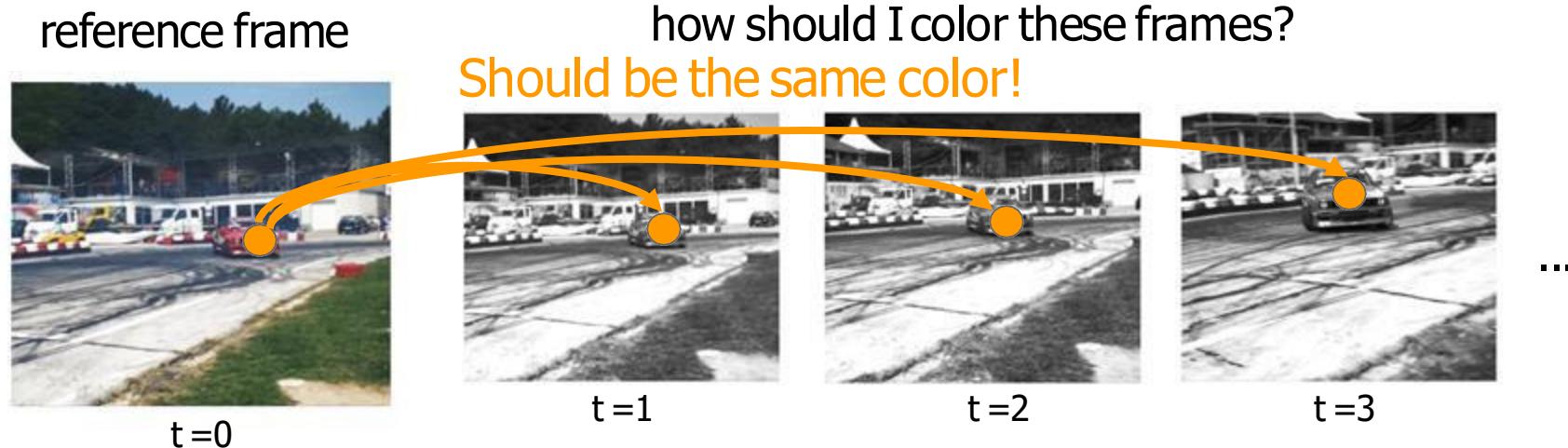
$t=3$

...

Source: [Vondrick et al., 2018](#)

Pretext task: video coloring

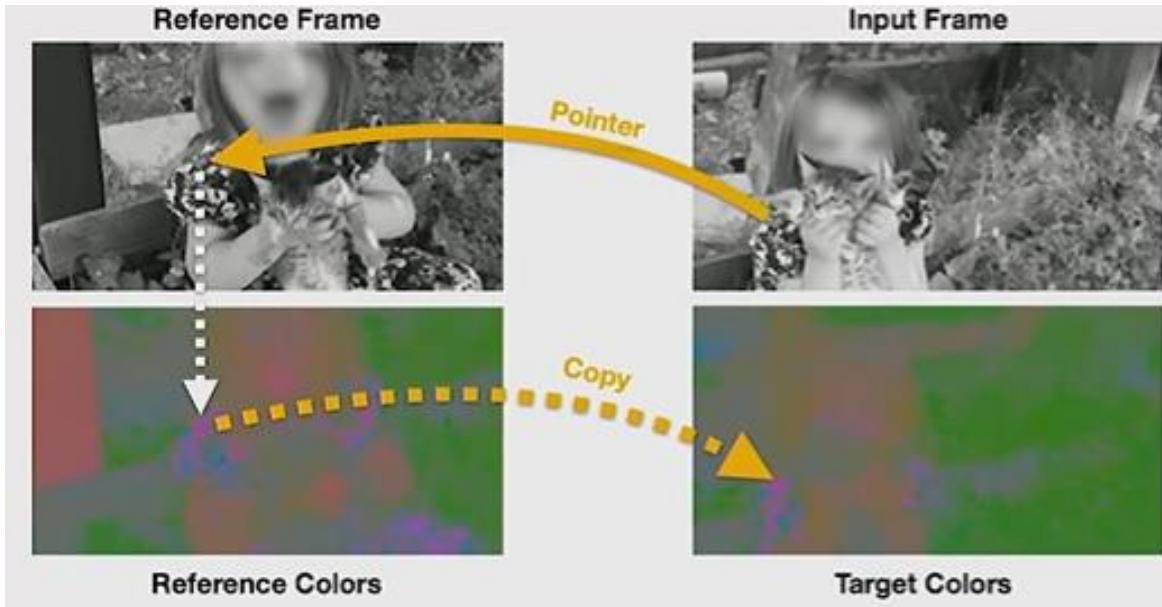
Idea: model the temporal coherence of colors in videos



Hypothesis: learning to color video frames should allow model to learn to track regions or objects without labels!

Source: [Vondrick et al., 2018](#)

Learning to color videos



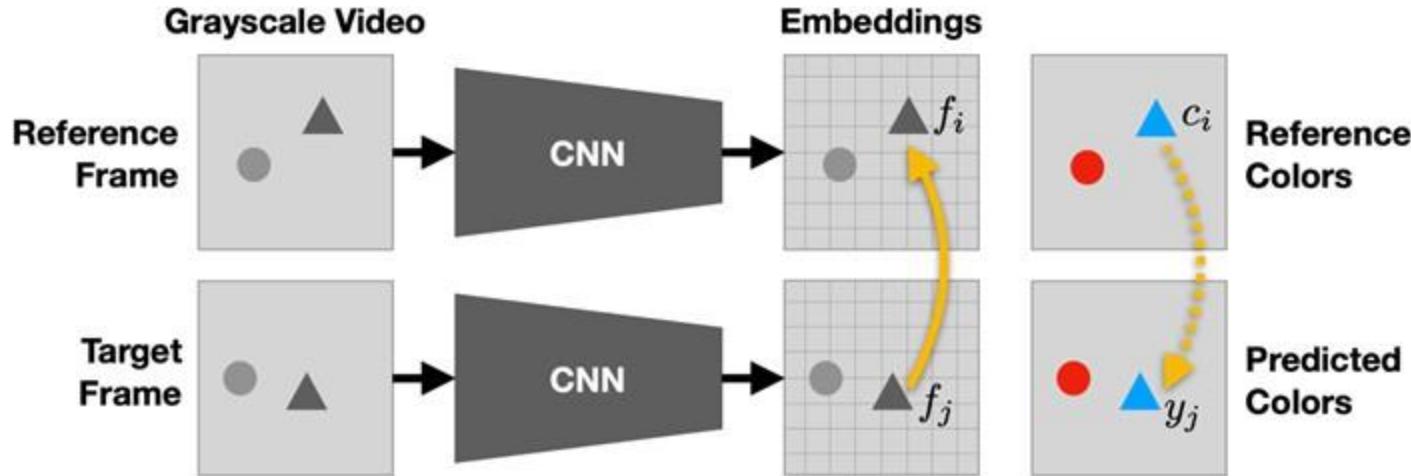
Learning objective:

Establish mappings between reference and target frames in a learned feature space.

Use the mapping as “pointers” to copy the correct color (LAB).

Source: [Vondrick et al., 2018](#)

Learning to color videos

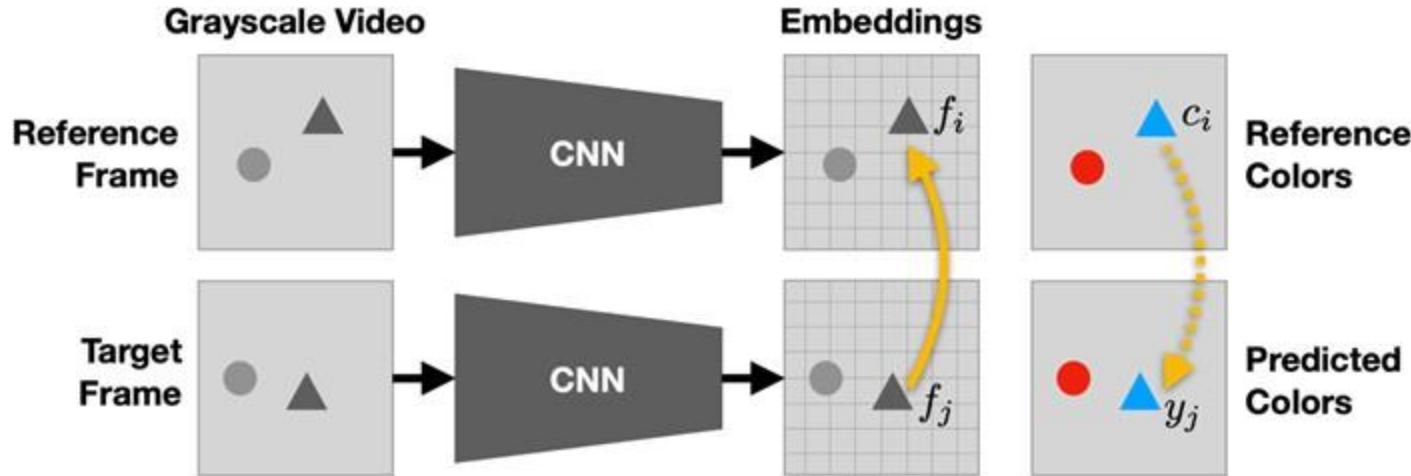


attention map on the reference frame

$$A_{ij} = \frac{\exp(f_i^T f_j)}{\sum_k \exp(f_k^T f_j)}$$

Source: [Vondrick et al., 2018](#)

Learning to color videos



attention map on the reference frame

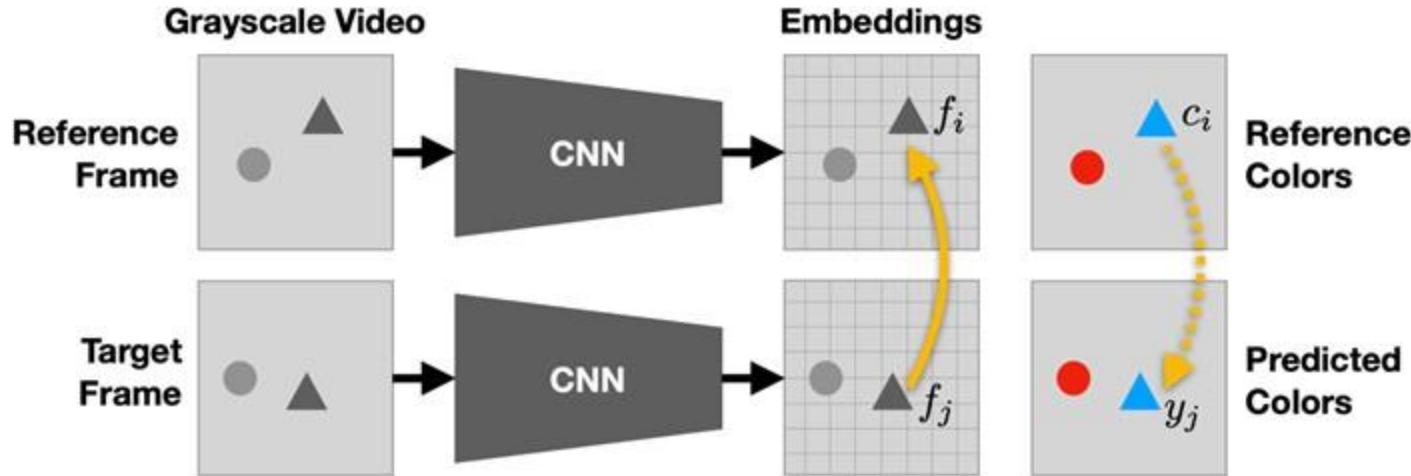
predicted color =weighted sum of the reference color

$$A_{ij} = \frac{\exp(f_i^T f_j)}{\sum_k \exp(f_k^T f_j)}$$

$$y_j = \sum_i A_{ij} c_i$$

Source: [Vondrick et al., 2018](#)

Learning to color videos



attention map on the reference frame

$$A_{ij} = \frac{\exp(f_i^T f_j)}{\sum_k \exp(f_k^T f_j)}$$

predicted color = weighted sum of the reference color

$$y_j = \sum_i A_{ij} c_i$$

loss between predicted color and ground truth color

$$\min_{\theta} \sum_j \mathcal{L}(y_j, c_j)$$

Source: [Vondrick et al., 2018](#)

Colorizing videos (qualitative)

reference frame



target frames (gray)



predicted color



Source: [Google AI blog post](#)

Colorizing videos (qualitative)

reference frame



target frames (gray)



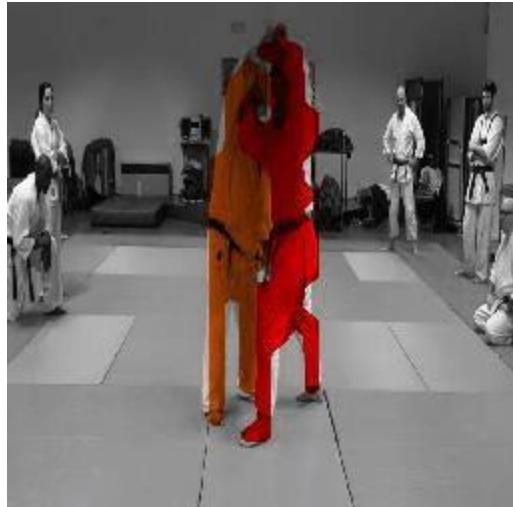
predicted color



Source: [Google AI blog post](#)

Tracking emerges from colorization

Propagate segmentation masks using learned attention



Source: [Google AI blog post](#)

Tracking emerges from colorization

Propagate pose keypoints using learned attention



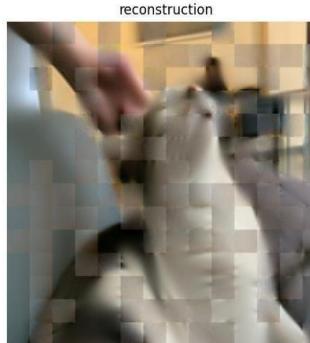
Source: [Google AI blog post](#)

Masked Auto Encoders (MAE)

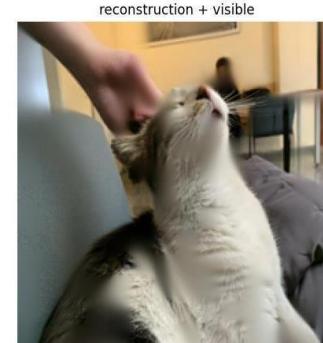
Reconstruction with a larger masked portion



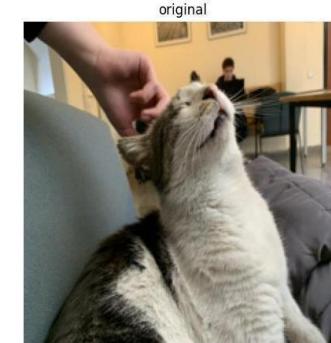
50% masking ratio



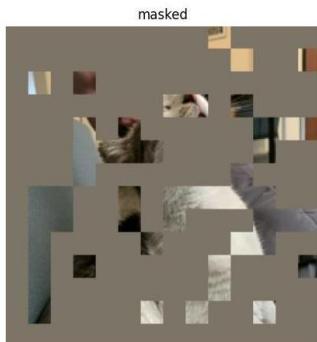
reconstruction



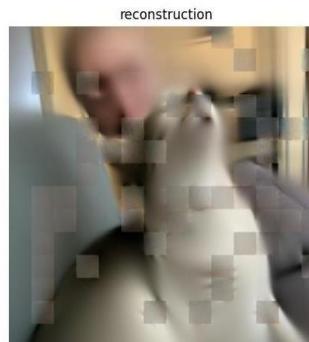
reconstruction + visible



original



75% masking ratio



reconstruction



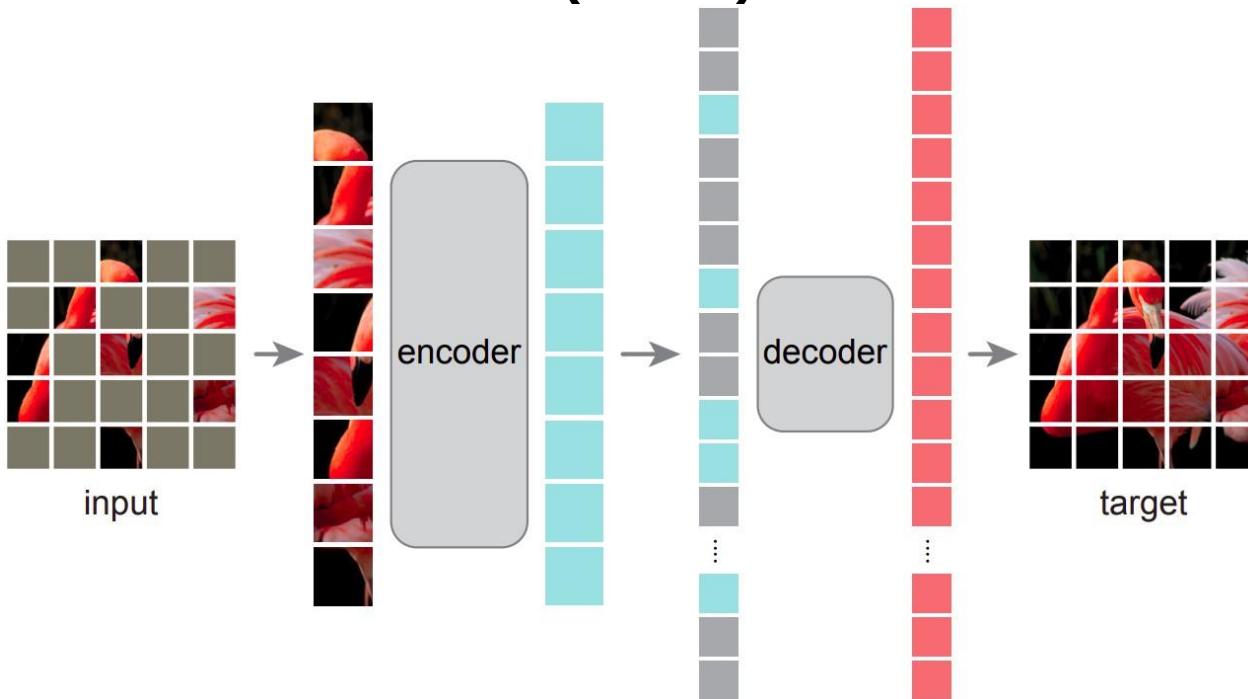
reconstruction + visible



original

He et al., 2021 Masked Autoencoders Are Scalable Vision Learners

Masked Auto Encoders (MAE)



He et al., 2021 Masked Autoencoders
Are Scalable Vision Learners

Masking Methos

- Similar to the original ViT, divide the input into non-overlapping patches.
- Uniformly sample a very large proportion (75%) of these patches and mask them
- Masking a high ratio makes predicting the task challenging and meaningful.
- Also, not using mask tokens and picking a high sampling ratio (masking most of the image, e.g., 75%, and sampling a small visible portion, e.g., 25%, to feed into the encoder) enables the encoder to be very large.

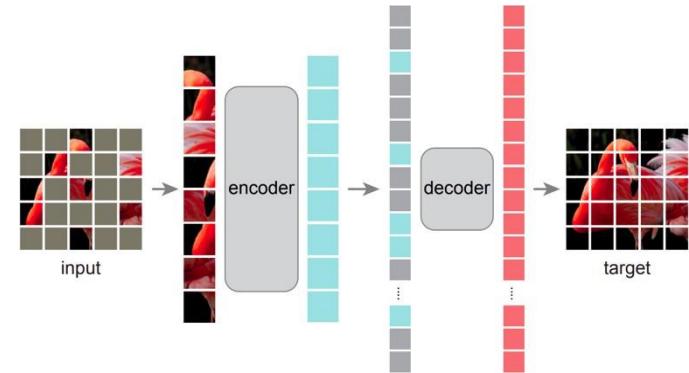


Figure 6. MAE architecture
from the paper

MAE Encoder

- The encoder only operates on unmasked patches (25%)
- Embeds the patches by linear projection and add positional embeddings
- Uses transformer blocks
- Since the input patches are a small part of the input, the encoder is chosen to be very large. (encoder has over 9 times computations per token vs decoder)

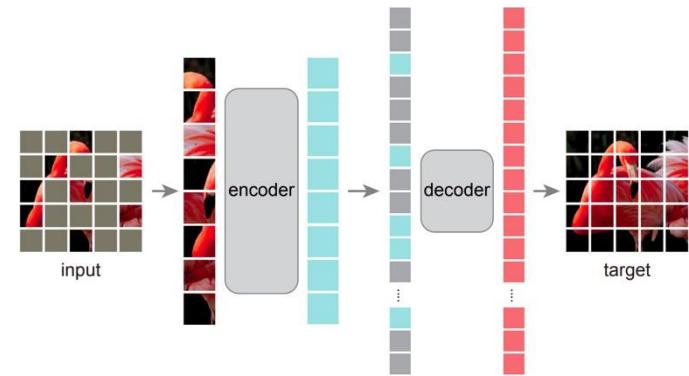


Figure 6. MAE architecture
from the paper

MAE Decoder

- Merges the encoder outputs with the shared mask tokens in previously masked places, adding positional encodings to them.
- Uses transformer blocks, followed by a linear projection for finalizing pixel reconstruction.
- Is solely responsible for reconstruction, meaning it is not used post-training. Hence, it is independent of the encoder design, making it flexible (unlike traditional AEs or UNet). This is an **asymmetrical** autoencoder design.

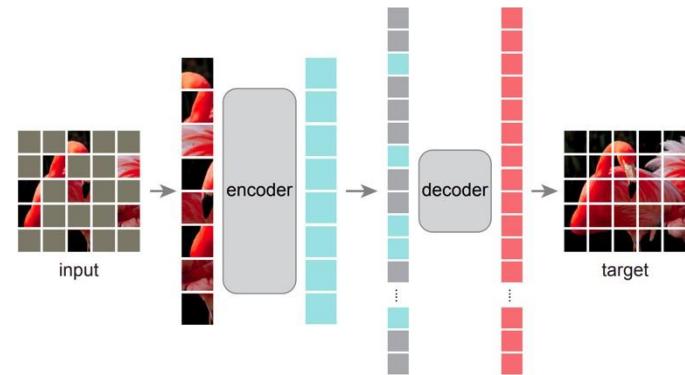


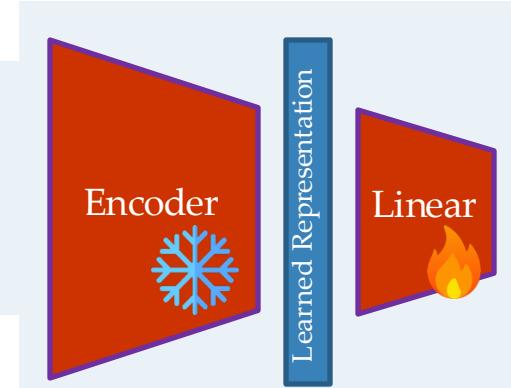
Figure 6. MAE architecture from the paper

Reconstruction

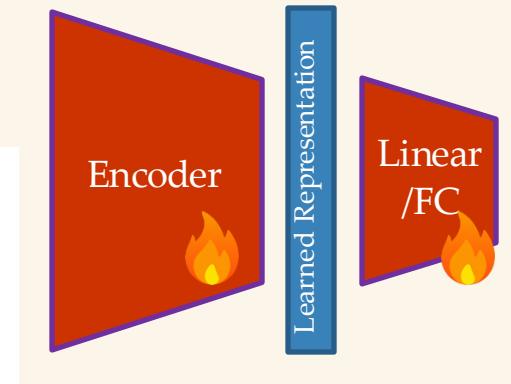
- The MSE (mean squared error loss) in the pixel space between the input image and the reconstructed image is adopted.
- Loss is only computed for masked patches

Linear Probing vs Full Fine-tuning

- In linear probing, the pre-trained model is fixed, and only one linear layer is added at the end, to predict the labels (or produce the output). This method is used to assess the quality of representations from a pre-trained feature extraction model.



- In fine-tuning, pre-trained model is further trained (not fixed), and one or more layers, possibly with non-linearities are added.
- linear probing: provides a measure of representation quality of a pre-training in restricted conditions
- fine-tuning: exploits models near-true potential to adopt for new tasks

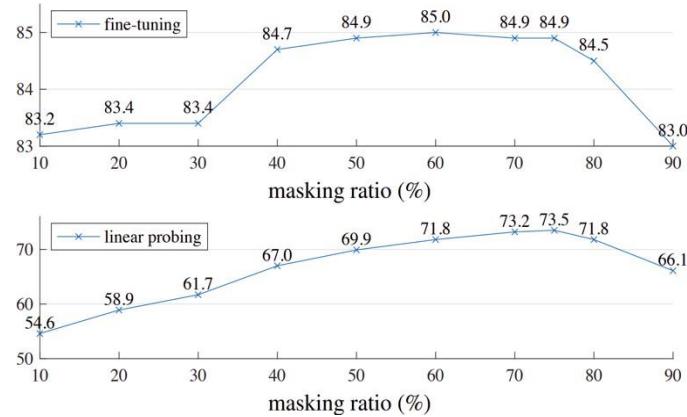


He et al., 2021 Masked Autoencoders Are Scalable Vision Learners

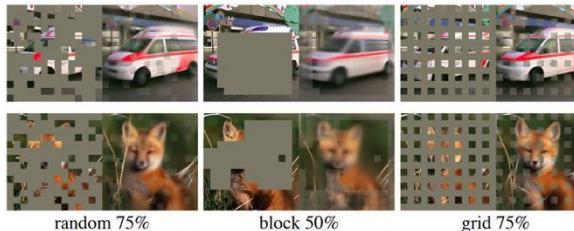
Ablation Studies

So many modeling/hyperparameter choices:

- Masking ratio
- Decoder depth
- Decoder width
- Mask token (used or not in encoder)
- Reconstruction target
- Data augmentation
- Mask sampling method
- Training schedule

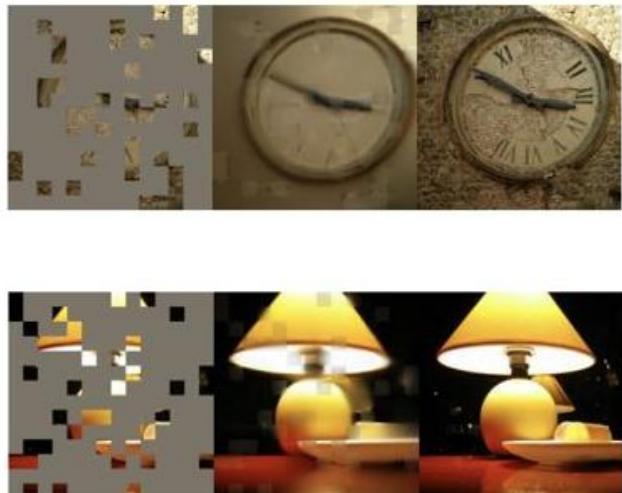


case	ratio	ft	lin
random	75	84.9	73.5
block	50	83.9	72.3
block	75	82.8	63.9
grid	75	84.0	66.0



He et al., 2021 Masked Autoencoders Are Scalable Vision Learners

Masked Autoencoder – Comparisons



method	pre-train data	ViT-B	ViT-L	ViT-H	ViT-H ₄₄₈
scratch, our impl.	-	82.3	82.6	83.1	-
DINO [5]	IN1K	82.8	-	-	-
MoCo v3 [9]	IN1K	83.2	84.1	-	-
BEiT [2]	IN1K+DALLE	83.2	85.2	-	-
MAE	IN1K	<u>83.6</u>	<u>85.9</u>	<u>86.9</u>	87.8

Summary: pretext tasks from image transformations

- Pretext tasks focus on “visual common sense”, e.g., predict rotations, inpainting, rearrangement, and colorization.
- The models are forced learn good features about natural images, e.g., semantic representation of an object category, in order to solve the pretext tasks.
- We often do not care about the performance of these pretext tasks, but rather how useful the learned features are for downstream tasks (classification, detection, segmentation).

Summary: pretext tasks from image transformations

- Pretext tasks focus on “visual common sense”, e.g., predict rotations, inpainting, rearrangement, and colorization.
- The models are forced learn good features about natural images, e.g., semantic representation of an object category, in order to solve the pretext tasks.
- We often do not care about the performance of these pretext tasks, but rather how useful the learned features are for downstream tasks (classification, detection, segmentation).
- Problems: 1) coming up with individual pretext tasks is tedious, and 2) the learned representations may not be general.

Pretext tasks from image transformations

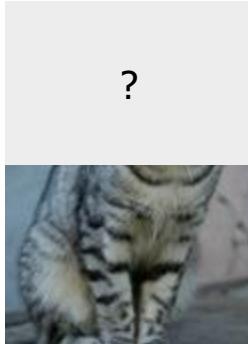
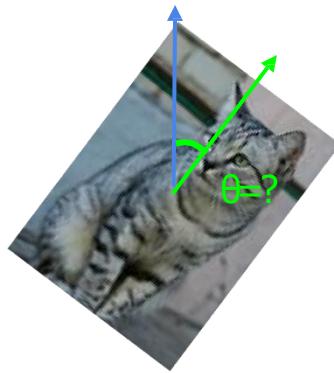
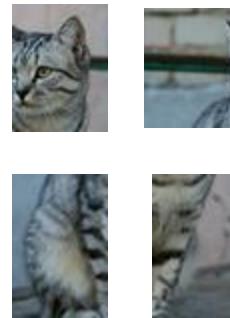


image completion



rotation prediction



"jigsaw puzzle"



colorization

Learned representations may be tied to a specific pretext task!

Can we come up with a more general pretext task?

Today's Agenda

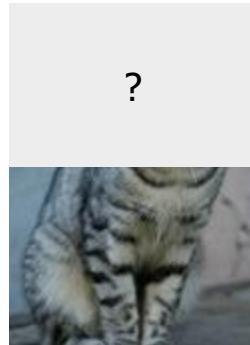
Pretext tasks from image transformations

- Rotation, inpainting, rearrangement, coloring
- Reconstruction-based learning (MAE)

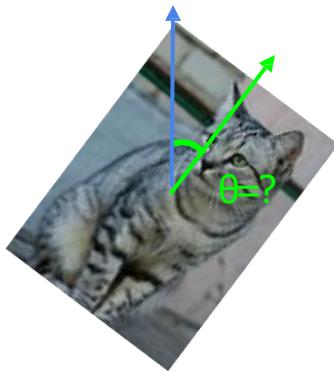
Contrastive representation learning

- Intuition and formulation
- Instance contrastive learning: SimCLR and MOCO
- Sequence contrastive learning: CPC
- Self-Distillation Without Labels, DINO

A more general pretext task?



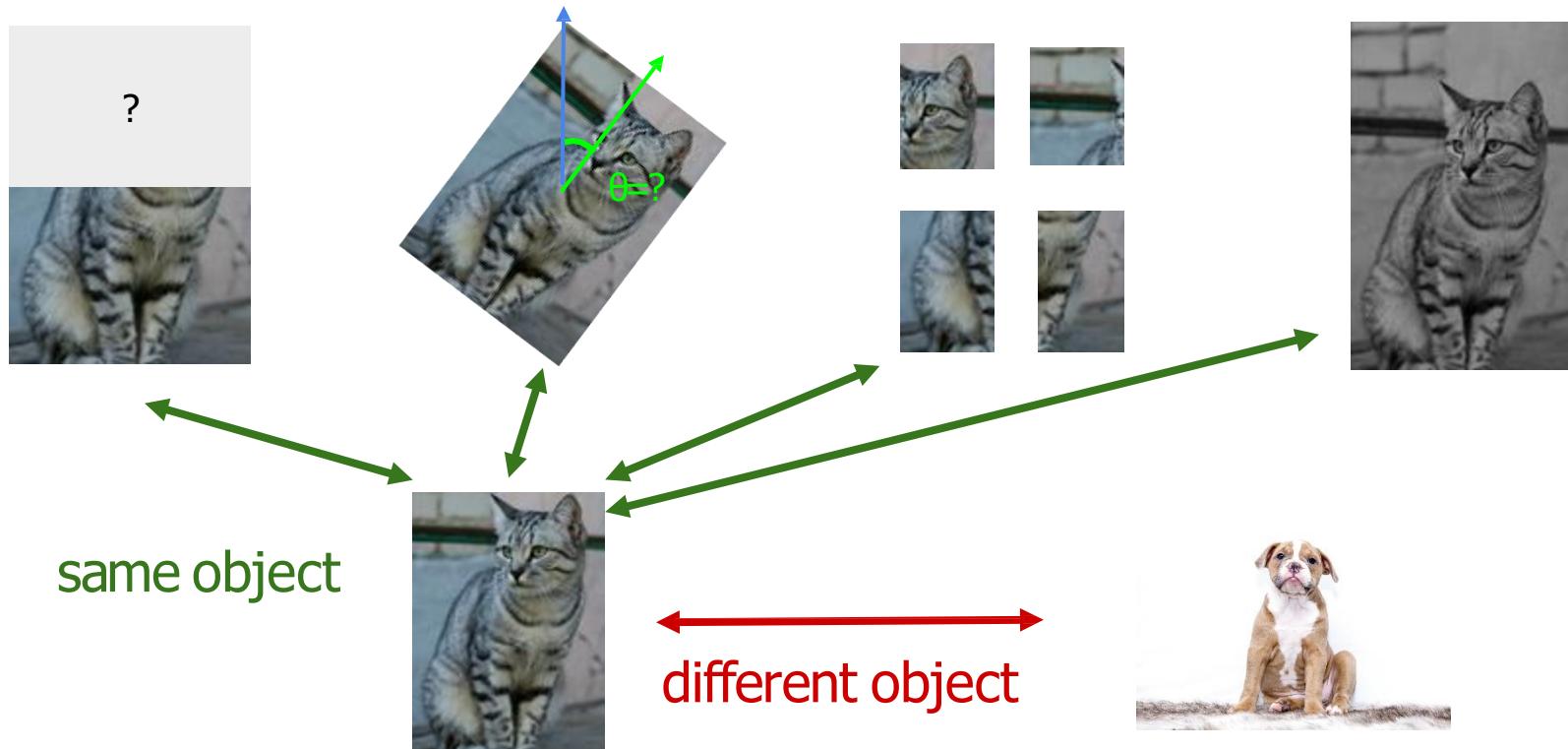
?



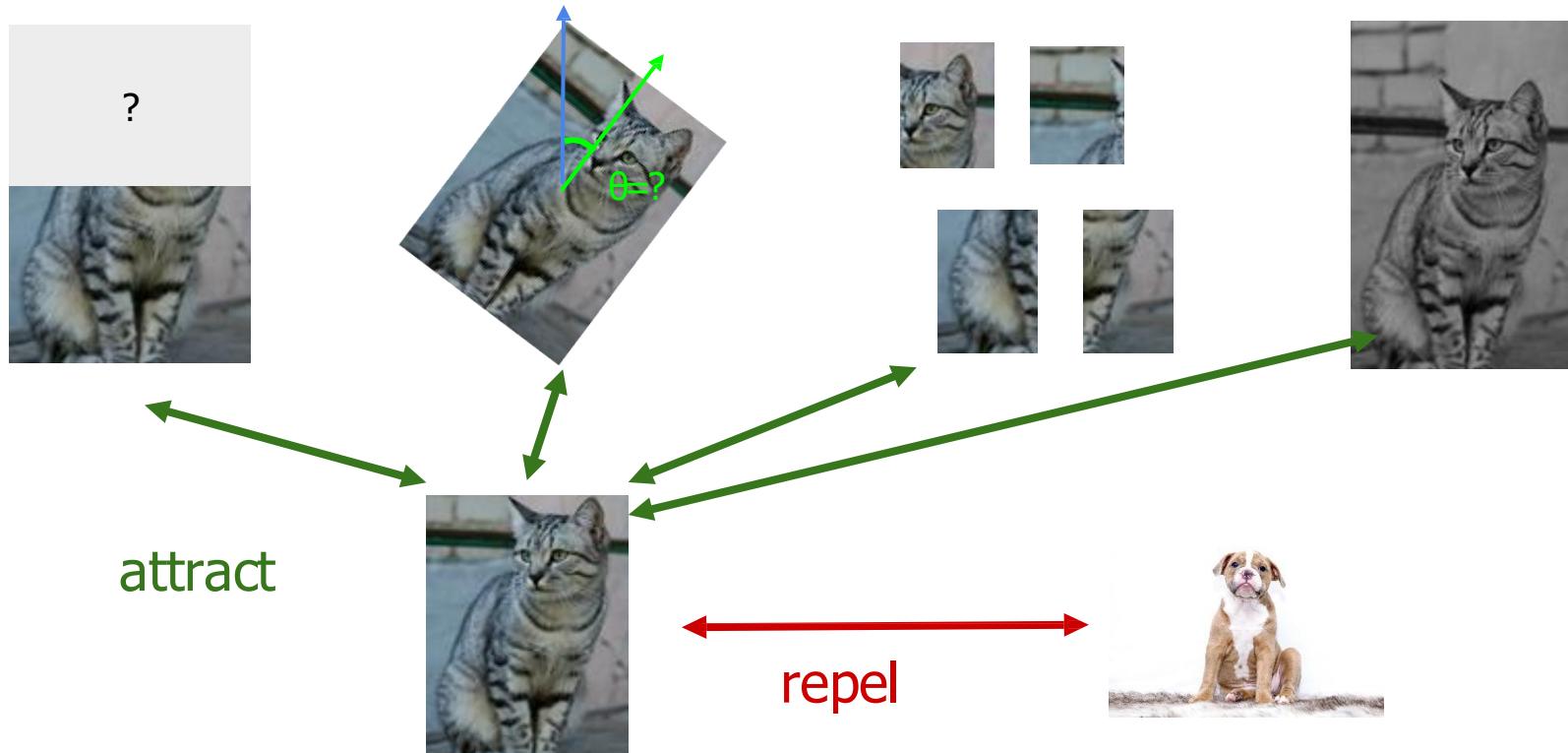
same object



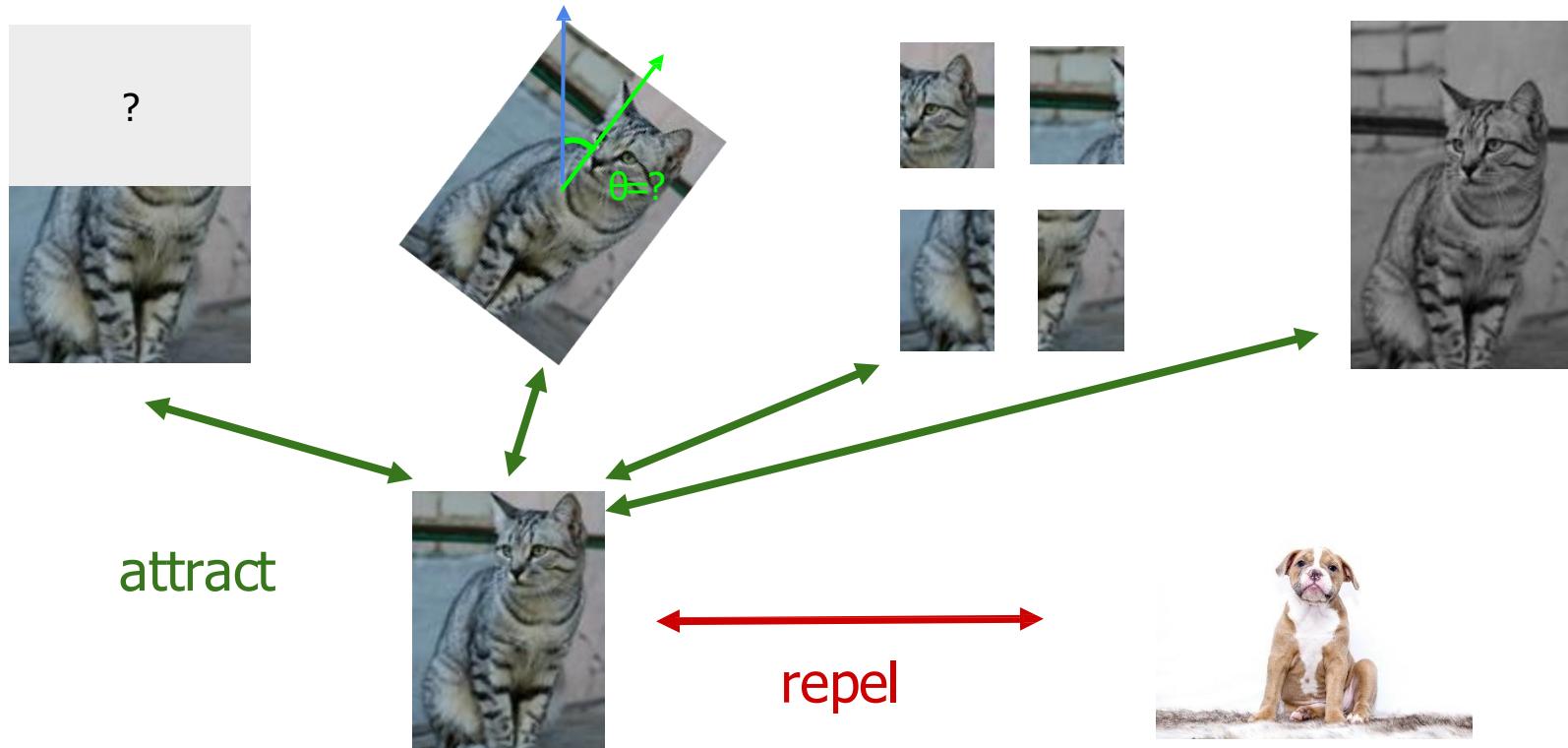
A more general pretext task?



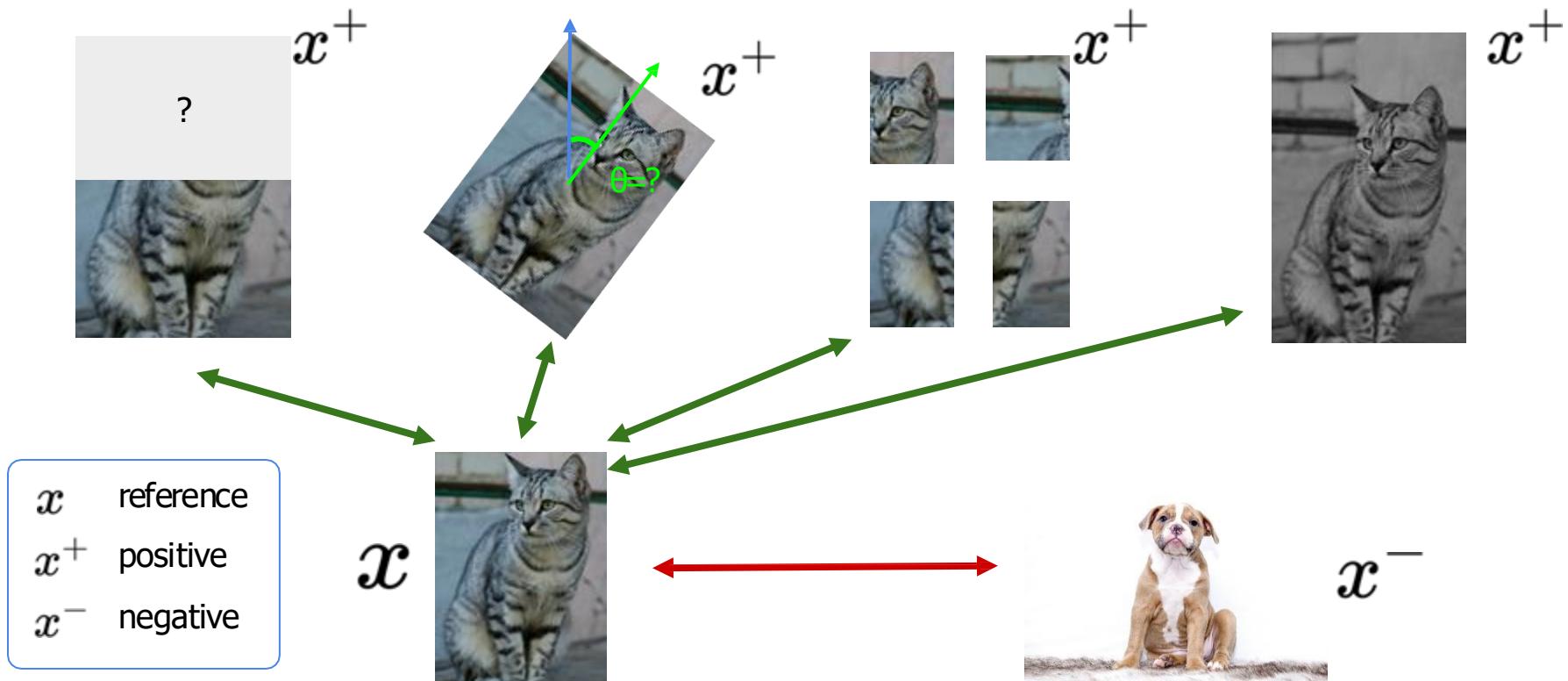
Contrastive Representation Learning



Contrastive Representation Learning



Contrastive Representation Learning



A formulation of contrastive learning

What we want:

$$\text{score}(f(x), f(x^+)) \gg \text{score}(f(x), f(x^-))$$

x : reference sample; x^+ positive sample; x^- negative sample

Given a chosen score function, we aim to learn an encoder function f that yields high score for positive pairs (x, x^+) and low scores for negative pairs (x, x^-) .

A formulation of contrastive learning

Loss function given 1 positive sample and $N - 1$ negative samples:

$$L = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+)))}{\exp(s(f(x), f(x^+))) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-)))} \right]$$

A formulation of contrastive learning

Loss function given 1 positive sample and $N - 1$ negative samples:

$$L = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+)))}{\exp(s(f(x), f(x^+))) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-)))} \right]$$

 x  x^+  x  x_1^-  x_2^-  x_3^- \dots

A formulation of contrastive learning

Loss function given 1 positive sample and $N - 1$ negative samples:

$$L = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+)))}{\exp(s(f(x), f(x^+))) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-)))} \right]$$

score for the positive pair
score for the N-1 negative pairs

This seems familiar ...

A formulation of contrastive learning

Loss function given 1 positive sample and $N - 1$ negative samples:

$$L = -\mathbb{E}_X \left[\log \frac{\overline{\exp(s(f(x), f(x^+))}}}{\overline{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-)))}}} \right]$$

score for the positive pair score for the $N-1$ negative pairs

This seems familiar ...

Cross entropy loss for a N -way softmax classifier!

I.e., learn to find the positive sample from the N samples

A formulation of contrastive learning

Loss function given 1 positive sample and $N - 1$ negative samples:

$$L = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+)))}{\exp(s(f(x), f(x^+))) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-)))} \right]$$

Commonly known as the InfoNCE loss ([van den Oord et al., 2018](#))

A lower bound on the mutual information between $f(x)$ and $f(x^+)$

$$MI[f(x), f(x^+)] - \log(N) \geq -L$$

The larger the negative sample size (N), the tighter the bound

Detailed derivation: [Poole et al., 2019](#)

SimCLR: A Simple Framework for Contrastive Learning

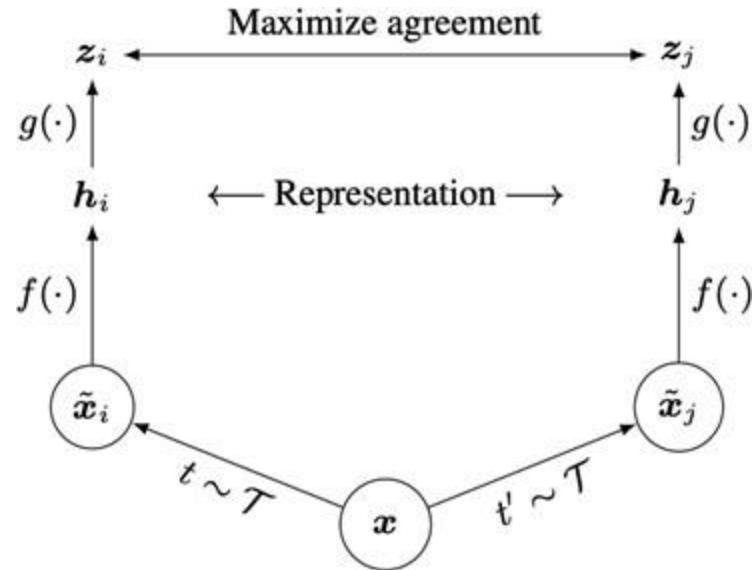
Cosine similarity as the score function:

$$s(u, v) = \frac{u^T v}{\|u\| \|v\|}$$

Use a projection network $g(\cdot)$ to project features to a space where contrastive learning is applied

Generate positive samples through data augmentation:

- random cropping, random color distortion, and random blur.



Source: [Chen et al., 2020](#)

SimCLR

Generate a positive pair
by sampling data
augmentation functions

Algorithm 1 SimCLR's main learning algorithm.

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

*We use a slightly different formulation in the assignment. You should follow the assignment instructions.

Source: [Chen et al., 2020](#)

SimCLR

Generate a positive pair
by sampling data
augmentation functions

Algorithm 1 SimCLR's main learning algorithm.

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

*We use a slightly different formulation in the assignment. You should follow the assignment instructions.

InfoNCE loss:
Use all non-positive
samples in the batch
as \mathbf{x}^-

Source: [Chen et al., 2020](#)

SimCLR: generating positive samples from data augmentation



(a) Original



(b) Crop and resize



(c) Crop, resize (and flip)



(d) Color distort. (drop)



(e) Color distort. (jitter)



(f) Rotate $\{90^\circ, 180^\circ, 270^\circ\}$



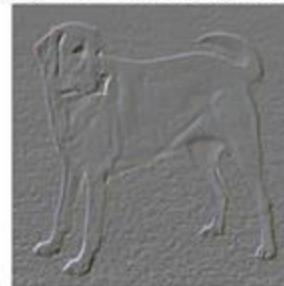
(g) Cutout



(h) Gaussian noise



(i) Gaussian blur



(j) Sobel filtering

Original Image: CC-BY Von.grzanka

Source: [Chen et al., 2020](#)

SimCLR

Generate a positive pair
by sampling data
augmentation functions

Iterate through and use
each of the $2N$ sample
as reference, compute
average loss

Algorithm 1 SimCLR's main learning algorithm.

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

*We use a slightly different formulation in the assignment. You should follow the assignment instructions.

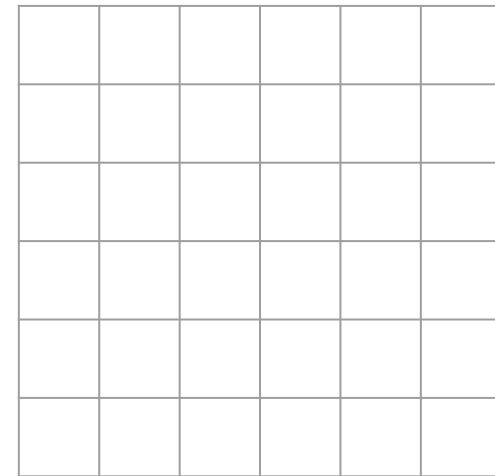
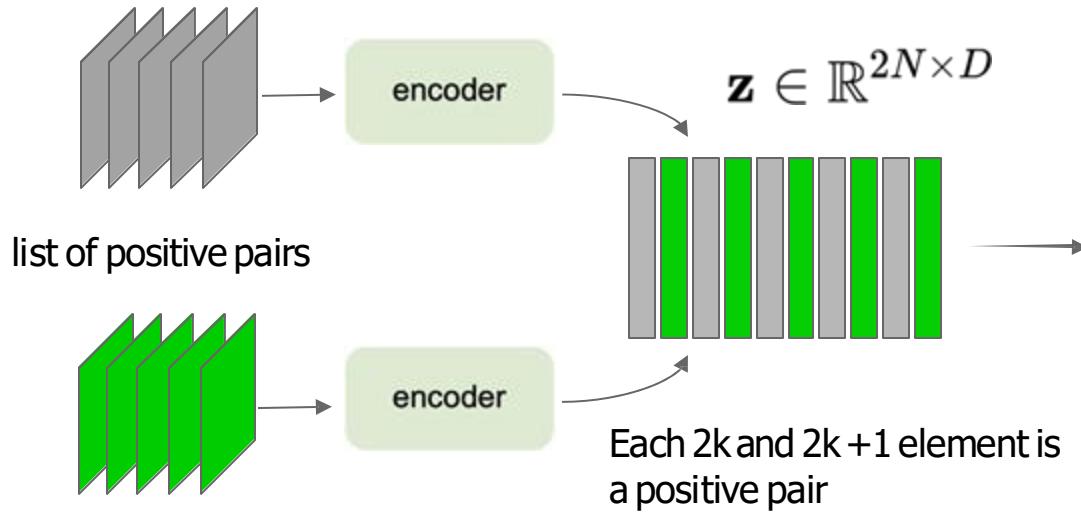
InfoNCE loss:
Use all non-positive
samples in the batch
as \mathbf{x}^-

Source: [Chen et al., 2020](#)

SimCLR: mini-batch training

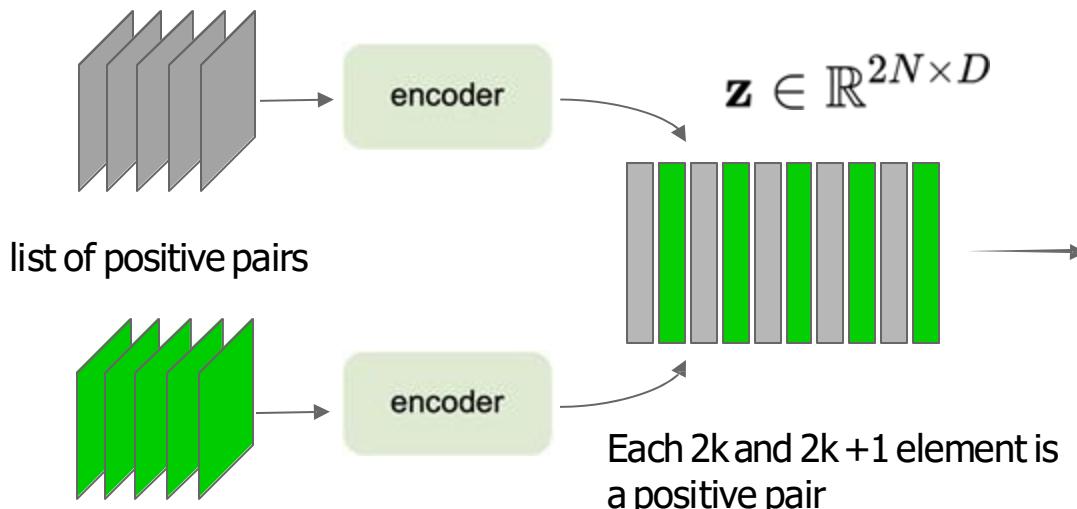
$$s_{i,j} = \frac{z_i^T z_j}{\|z_i\| \|z_j\|}$$

"Affinity matrix"



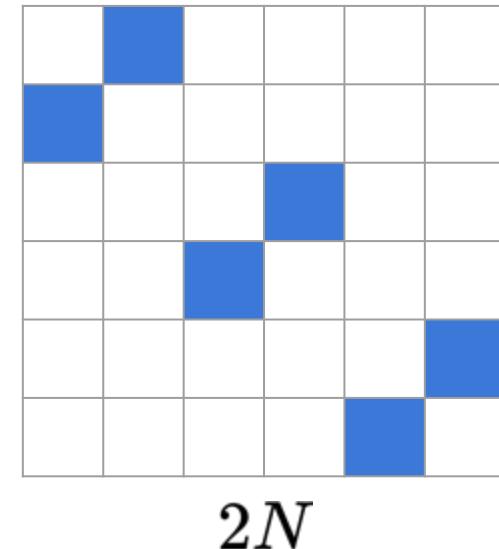
*We use a slightly different formulation in the assignment.
You should follow the assignment instructions.

SimCLR: mini-batch training



$$s_{i,j} = \frac{\mathbf{z}_i^T \mathbf{z}_j}{\|\mathbf{z}_i\| \|\mathbf{z}_j\|}$$

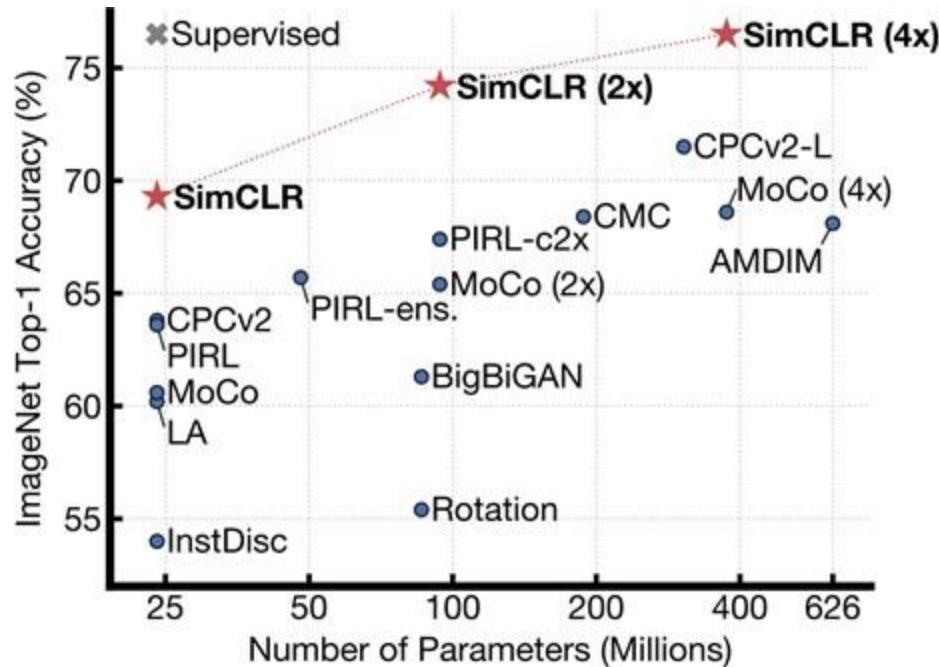
"Affinity matrix"



*We use a slightly different formulation in the assignment.
You should follow the assignment instructions.

=classification label for each row

Training linear classifier on SimCLR features



Train feature encoder on ImageNet (entire training set) using SimCLR.

Freeze feature encoder, train a linear classifier on top with labeled data.

Source: [Chen et al., 2020](#)

Semi-supervised learning on SimCLR features

Method	Architecture	Label fraction		
		1%	10%	Top 5
Supervised baseline	ResNet-50	48.4	80.4	
<i>Methods using other label-propagation:</i>				
Pseudo-label	ResNet-50	51.6	82.4	
VAT+Entropy Min.	ResNet-50	47.0	83.4	
UDA (w. RandAug)	ResNet-50	-	88.5	
FixMatch (w. RandAug)	ResNet-50	-	89.1	
S4L (Rot+VAT+En. M.)	ResNet-50 (4×)	-	91.2	
<i>Methods using representation learning only:</i>				
InstDisc	ResNet-50	39.2	77.4	
BigBiGAN	RevNet-50 (4×)	55.2	78.8	
PIRL	ResNet-50	57.2	83.8	
CPC v2	ResNet-161(*)	77.9	91.2	
SimCLR (ours)	ResNet-50	75.5	87.8	
SimCLR (ours)	ResNet-50 (2×)	83.0	91.2	
SimCLR (ours)	ResNet-50 (4×)	85.8	92.6	

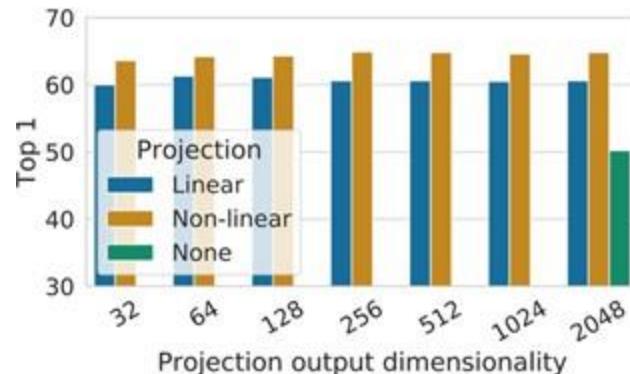
Table 7. ImageNet accuracy of models trained with few labels.

Train feature encoder on ImageNet (entire training set) using SimCLR.

Finetune the encoder with 1% / 10% of labeled data on ImageNet.

Source: [Chen et al., 2020](#)

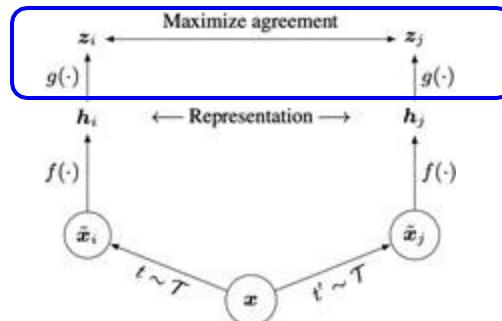
SimCLR design choices: projection head



Linear / non-linear projection heads improve representation learning.

A possible explanation:

- contrastive learning objective may discard useful information for downstream tasks
- representation space z is trained to be invariant to data transformation.
- by leveraging the projection head $g(\cdot)$, more information can be preserved in the h representation space



Source: [Chen et al., 2020](#)

SimCLR design choices: large batch size

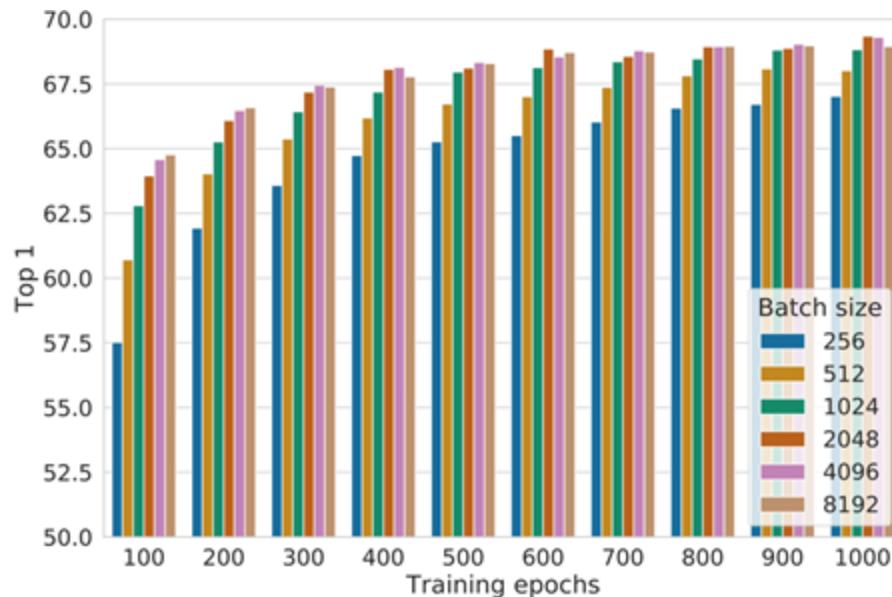


Figure 9. Linear evaluation models (ResNet-50) trained with different batch size and epochs. Each bar is a single run from scratch.¹⁰

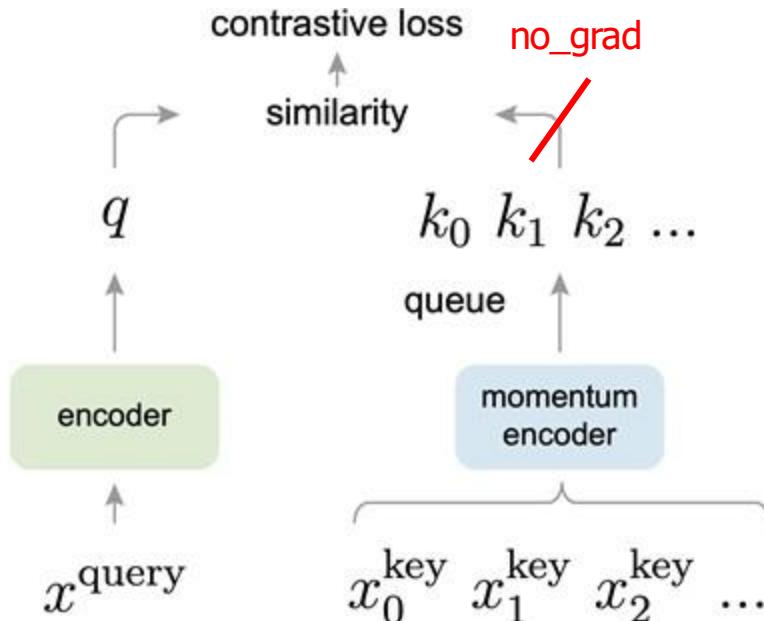
Large training batch size is crucial for SimCLR!

Large batch size causes large memory footprint during backpropagation:
requires distributed training on TPUs
(ImageNet experiments)

$$MI[f(x), f(x^+)] - \log(N) \geq -L$$

Source: [Chen et al., 2020](#)

Momentum Contrastive Learning (MoCo)

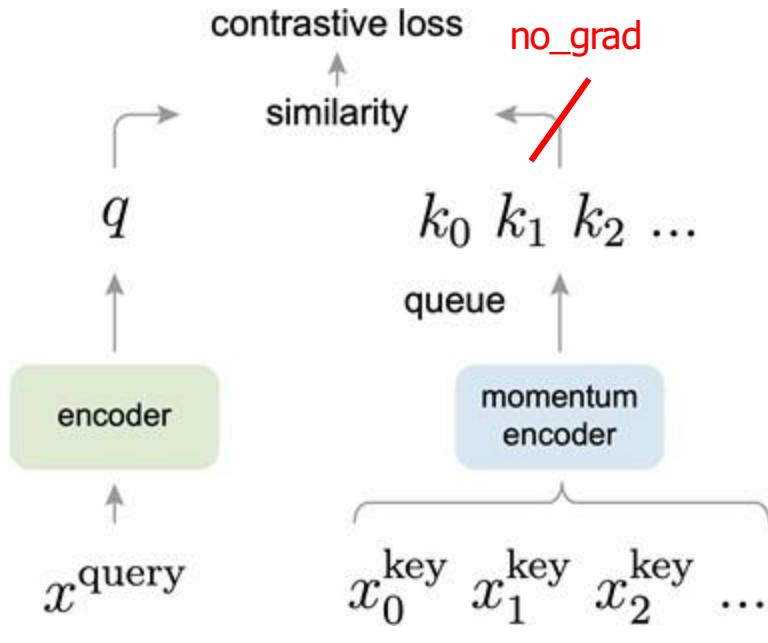


Key differences to SimCLR:

- Keep a running **queue** of keys (negative samples).
- Compute gradients and update the encoder **only through the queries**.
- Decouple min-batch size with the number of keys: can support **a large number of negative samples**.

Source: [He et al., 2020](#)

Momentum Contrastive Learning (MoCo)



Key differences to SimCLR:

- Keep a running **queue** of keys (negative samples).
- Compute gradients and update the encoder **only through the queries**.
- Decouple min-batch size with the number of keys: can support **a large number of negative samples**.
- The key encoder is **slowly progressing** through the momentum update rules:
$$\theta_k \leftarrow m\theta_k + (1 - m)\theta_q$$

Source: [He et al., 2020](#)

MoCo

Generate a positive pair
by sampling data
augmentation functions

No gradient through
the key

Update the FIFO negative
sample queue

Algorithm 1 Pseudocode of MoCo in a PyTorch-like style.

```
# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (CxK)
# m: momentum
# t: temperature

f_k.params = f_q.params # initialize
for x in loader: # load a minibatch x with N samples
    x_q = aug(x) # a randomly augmented version
    x_k = aug(x) # another randomly augmented version

    q = f_q.forward(x_q) # queries: NxC
    k = f_k.forward(x_k) # keys: NxC
    k = k.detach() # no gradient to keys

    # positive logits: Nx1
    l_pos = bmm(q.view(N,1,C), k.view(N,C,1))

    # negative logits: NxK
    l_neg = mm(q.view(N,C), queue.view(C,K))

    # logits: Nx(1+K)
    logits = cat([l_pos, l_neg], dim=1)

    # contrastive loss, Eqn.(1)
    labels = zeros(N) # positives are the 0-th
    loss = CrossEntropyLoss(logits/t, labels)

    # SGD update: query network
    loss.backward()
    update(f_q.params)

    # momentum update: key network
    f_k.params = m*f_k.params+(1-m)*f_q.params

    # update dictionary
    enqueue(queue, k) # enqueue the current minibatch
    dequeue(queue) # dequeue the earliest minibatch
```

Use the running
queue of keys as the
negative samples

InfoNCE loss

Update f_k through
momentum

bmm: batch matrix multiplication; mm: matrix multiplication; cat: concatenation.

Source: [He et al., 2020](#)

“MoCo V2”

Improved Baselines with Momentum Contrastive Learning

Xinlei Chen Haoqi Fan Ross Girshick Kaiming He
Facebook AI Research (FAIR)

A hybrid of ideas from SimCLR and MoCo:

- From SimCLR: non-linear projection head and strong data augmentation.
- From MoCo: momentum-updated queues that allow training on a large number of negative samples (no TPU required!).

Source: [Chen et al., 2020](#)

MoCo vs. SimCLR vs. MoCo V2

case	unsup. pre-train				ImageNet acc.	VOC detection		
	MLP	aug+	cos	epochs		AP ₅₀	AP	AP ₇₅
supervised					76.5	81.3	53.5	58.8
MoCo v1				200	60.6	81.5	55.9	62.6
(a)	✓			200	66.2	82.0	56.4	62.6
(b)		✓		200	63.4	82.2	56.8	63.2
(c)	✓	✓		200	67.3	82.5	57.2	63.9
(d)	✓	✓	✓	200	67.5	82.4	57.0	63.6
(e)	✓	✓	✓	800	71.1	82.5	57.4	64.0

Table 1. **Ablation of MoCo baselines**, evaluated by ResNet-50 for (i) ImageNet linear classification, and (ii) fine-tuning VOC object detection (mean of 5 trials). “MLP”: with an MLP head; “aug+”: with extra blur augmentation; “cos”: cosine learning rate schedule.

Key takeaways:

- Non-linear projection head and strong data augmentation are crucial for contrastive learning.

Source: [Chen et al., 2020](#)

MoCo vs. SimCLR vs. MoCo V2

case	MLP	aug+	cos	epochs	batch	ImageNet acc.
MoCo v1 [6]				200	256	60.6
SimCLR [2]	✓	✓	✓	200	256	61.9
SimCLR [2]	✓	✓	✓	200	8192	66.6
MoCo v2	✓	✓	✓	200	256	67.5

results of longer unsupervised training follow:

SimCLR [2]	✓	✓	✓	1000	4096	69.3
MoCo v2	✓	✓	✓	800	256	71.1

Table 2. **MoCo vs. SimCLR**: ImageNet linear classifier accuracy (**ResNet-50, 1-crop 224×224**), trained on features from unsupervised pre-training. “aug+” in SimCLR includes blur and stronger color distortion. SimCLR ablations are from Fig. 9 in [2] (we thank the authors for providing the numerical results).

Key takeaways:

- Non-linear projection head and strong data augmentation are crucial for contrastive learning.
- Decoupling mini-batch size with negative sample size allows MoCo-V2 to outperform SimCLR with smaller batch size (256 vs. 8192).

Source: [Chen et al., 2020](#)

MoCo vs. SimCLR vs. MoCo V2

mechanism	batch	memory / GPU	time / 200-ep.
MoCo	256	5.0G	53 hrs
end-to-end	256	7.4G	65 hrs
end-to-end	4096	93.0G [†]	n/a

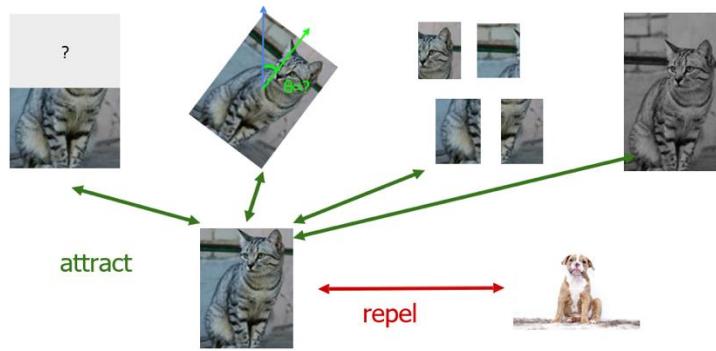
Table 3. **Memory and time cost** in 8 V100 16G GPUs, implemented in PyTorch. [†]: based on our estimation.

Key takeaways:

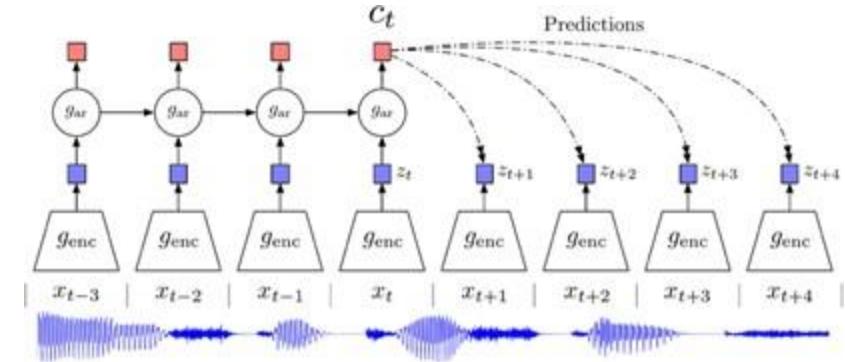
- Non-linear projection head and strong data augmentation are crucial for contrastive learning.
- Decoupling mini-batch size with negative sample size allows MoCo-V2 to outperform SimCLR with smaller batch size (256 vs. 8192).
- ... all with much smaller memory footprint! ("end-to-end" means SimCLR here)

Source: [Chen et al., 2020](#)

Instance vs. Sequence Contrastive Learning



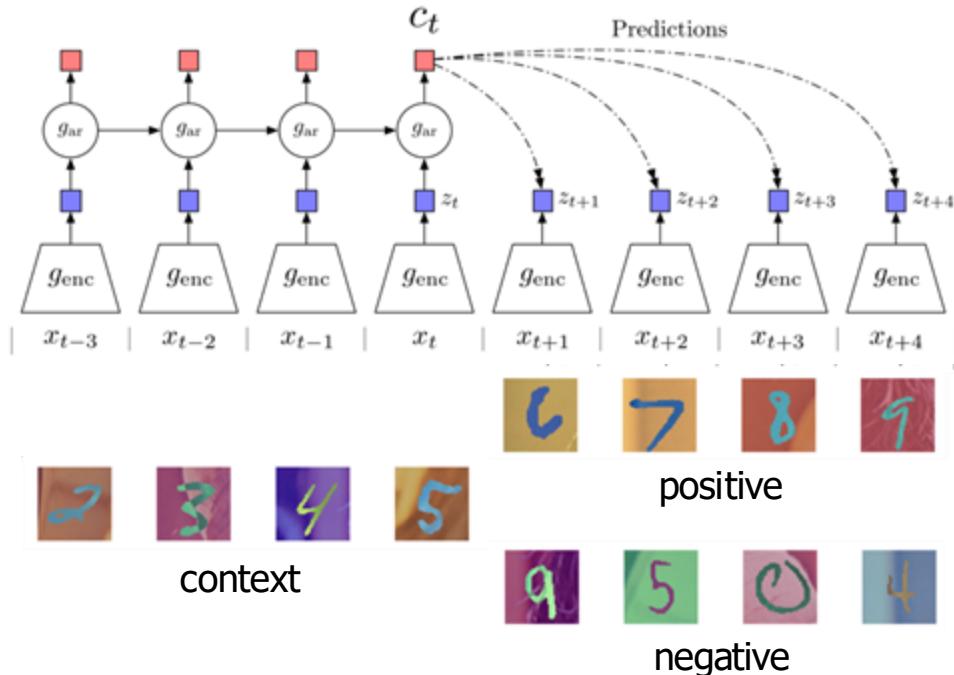
Instance-level contrastive learning:
contrastive learning based on
positive & negative instances.
Examples: SimCLR, MoCo



Source: [van den Oord et al., 2018](#)

Sequence-level contrastive learning:
contrastive learning based on
sequential / temporal orders.
Example: Contrastive Predictive Coding (CPC)

Contrastive Predictive Coding (CPC)



Contrastive: contrast between “right” and “wrong” **sequences** using contrastive learning.

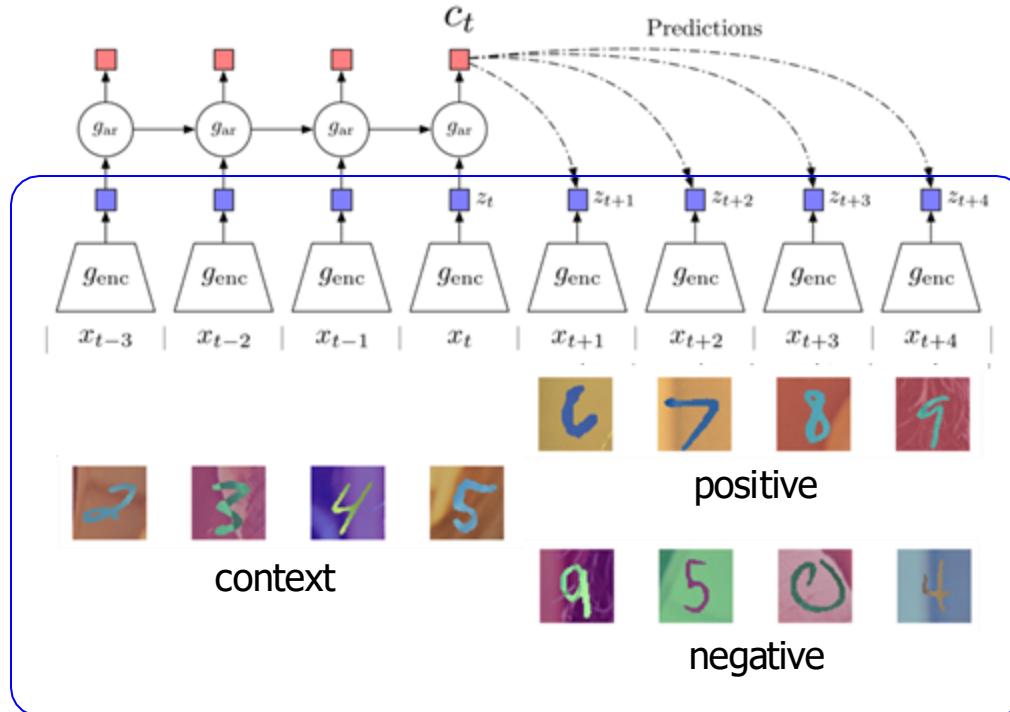
Predictive: the model has to predict **future patterns** given the current context.

Coding: the model learns **useful feature vectors**, or “code”, for downstream tasks, similar to other self-supervised methods.

Figure [source](#)

Source: [van den Oord et al., 2018](#),

Contrastive Predictive Coding (CPC)

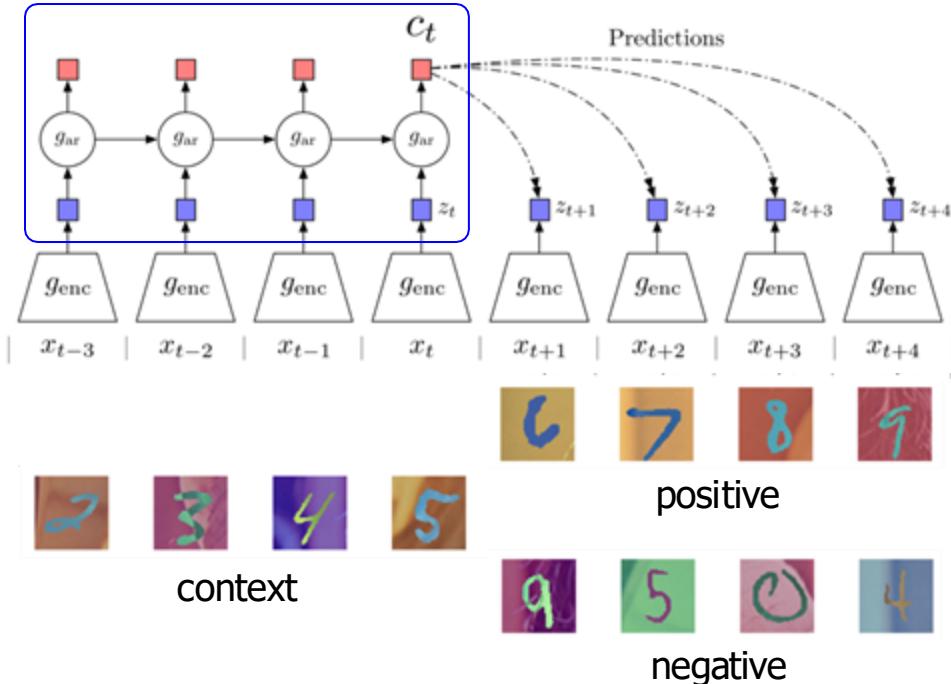


1. Encode all samples in a sequence into vectors $z_t = g_{\text{enc}}(x_t)$

Figure [source](#)

Source: [van den Oord et al., 2018](#),

Contrastive Predictive Coding (CPC)

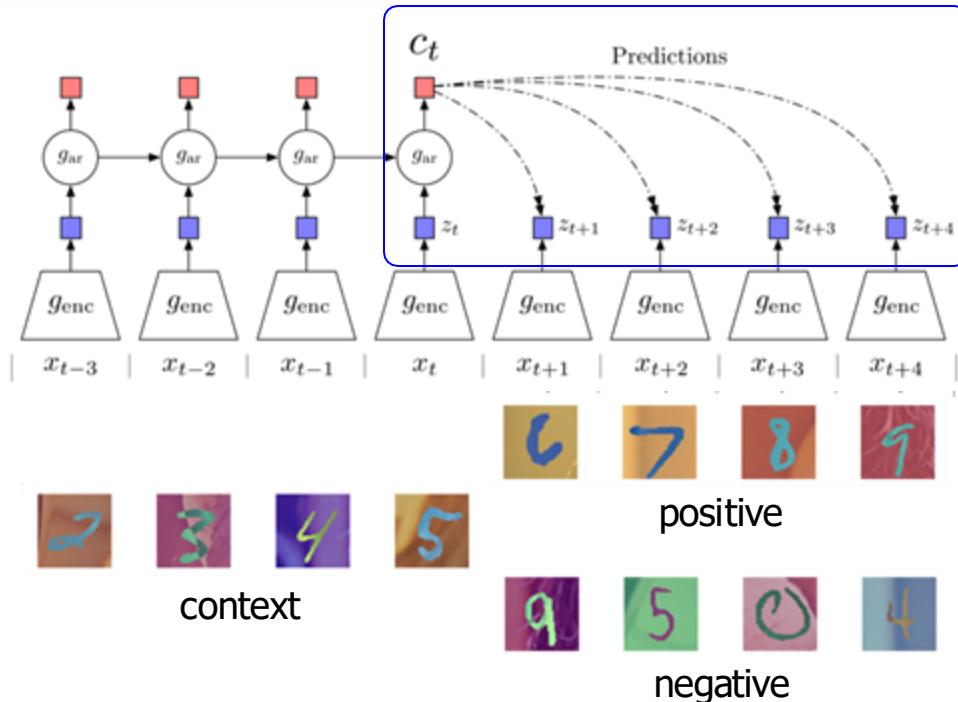


1. Encode all samples in a sequence into vectors $z_t = g_{\text{enc}}(x_t)$
2. Summarize context (e.g., half of a sequence) into a context code c_t using an auto-regressive model (g_{ar}). The original paper uses GRU-RNN here.

Figure [source](#)

Source: [van den Oord et al., 2018](#),

Contrastive Predictive Coding (CPC)



1. Encode all samples in a sequence into vectors $z_t = g_{\text{enc}}(x_t)$
2. Summarize context (e.g., half of a sequence) into a context code c_t using an auto-regressive model (g_{ar})
3. Compute InfoNCE loss between the context c_t and future code z_{t+k} using the following **time-dependent score function**:

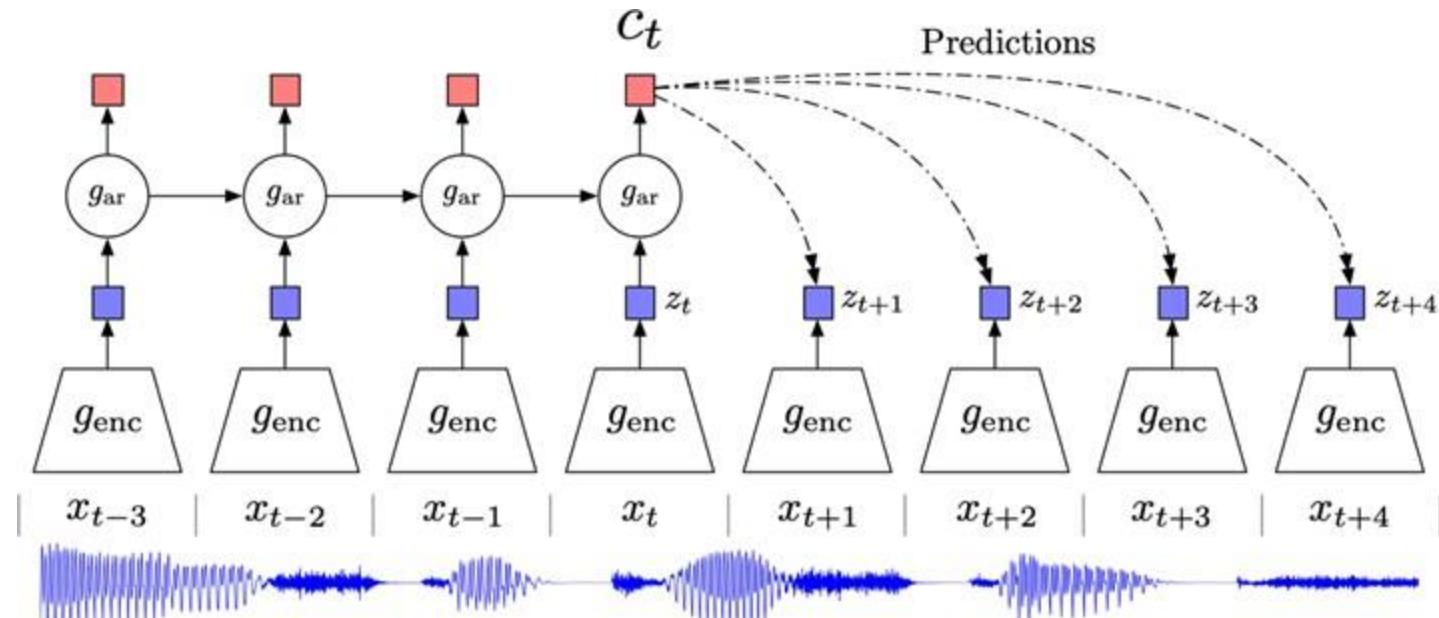
$$s_k(z_{t+k}, c_t) = z_{t+k}^T W_k c_t$$

where W_k is a trainable matrix.

Figure [source](#)

Source: [van den Oord et al., 2018](#),

CPC example: modeling audio sequences



Source: [van den Oord et al., 2018](#),

CPC example: modeling audio sequences

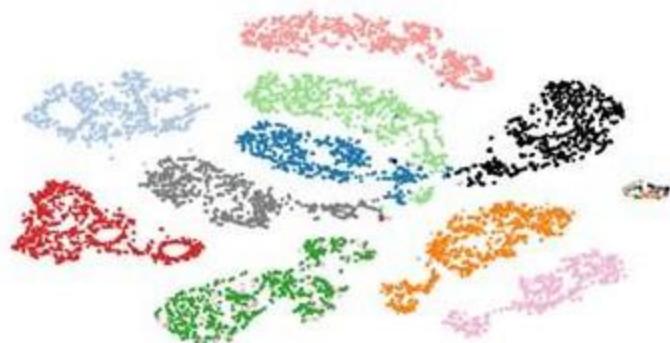


Figure 2: t-SNE visualization of audio (speech) representations for a subset of 10 speakers (out of 251). Every color represents a different speaker.

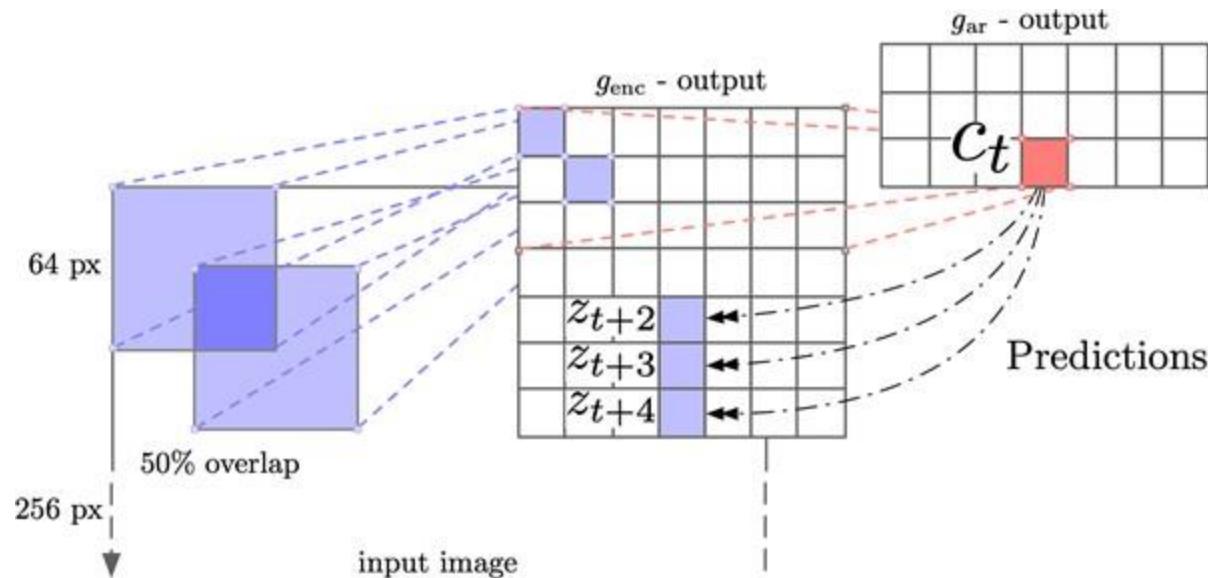
Method	ACC
Phone classification	
Random initialization	27.6
MFCC features	39.7
CPC	64.6
Supervised	74.6
Speaker classification	
Random initialization	1.87
MFCC features	17.6
CPC	97.4
Supervised	98.5

Linear classification on trained representations (LibriSpeech dataset)

Source: [van den Oord et al., 2018](#),

CPC example: modeling visual context

Idea: split image into patches, model rows of patches from top to bottom as a sequence. I.e., use top rows as context to predict bottom rows.



Source: [van den Oord et al., 2018](#),

Other examples: MoCo v3

An Empirical Study of Training Self-Supervised Vision Transformers

"This paper does not describe a novel method."

Xinlei Chen* Saining Xie* Kaiming He
Facebook AI Research (FAIR)

Code: <https://github.com/facebookresearch/moco-v3>

Abstract

This paper does not describe a novel method. Instead, it studies a straightforward, incremental, yet must-know baseline given the recent progress in computer vision: self-supervised learning for Vision Transformers (ViT). While the training recipes for standard convolutional networks have been highly mature and robust, the recipes for ViT are yet to be built, especially in the self-supervised scenarios where training becomes more challenging. In this work, we go back to basics and investigate the effects of several fundamental components for training self-supervised ViT. We observe that instability is a major issue that degrades accuracy, and it can be hidden by apparently good results. We reveal that these results are indeed partial failure, and they can be improved when training is made more stable. We benchmark ViT results in MoCo v3 and several other self-supervised frameworks, with ablations in various aspects. We discuss the currently positive evidence as well as challenges and open questions. We hope that this work will provide useful data points and experience for future research.

	framework	model	params	acc. (%)
<i>linear probing:</i>				
iGPT [9]	iGPT-L	1362M	69.0	
iGPT [9]	iGPT-XL	6801M	72.0	
MoCo v3	ViT-B	86M	76.7	
MoCo v3	ViT-L	304M	77.6	
MoCo v3	ViT-H	632M	78.1	
MoCo v3	ViT-BN-H	632M	79.1	
MoCo v3	ViT-BN-L/7	304M	81.0	
<i>end-to-end fine-tuning:</i>				
masked patch pred. [16]	ViT-B	86M	79.9 [†]	
MoCo v3	ViT-B	86M	83.2	
MoCo v3	ViT-L	304M	84.1	

Table 1. **State-of-the-art Self-supervised Transformers** in ImageNet classification, evaluated by linear probing (top panel) or end-to-end fine-tuning (bottom panel). Both iGPT [9] and masked patch prediction [16] belong to the masked auto-encoding paradigm. MoCo v3 is a contrastive learning method that compares two (224×224) crops. ViT-B, -L, -H are the Vision Transformers proposed in [16]. ViT-BN is modified with BatchNorm, and “/7” denotes a patch size of 7×7 . [†]: pre-trained in JFT-300M.

DINO: Self-Distillation with No Labels

Emerging Properties in Self-Supervised Vision Transformers

Mathilde Caron^{1,2} Hugo Touvron^{1,3} Ishan Misra¹ Hervé Jegou¹
Julien Mairal² Piotr Bojanowski¹ Armand Joulin¹

¹ Facebook AI Research

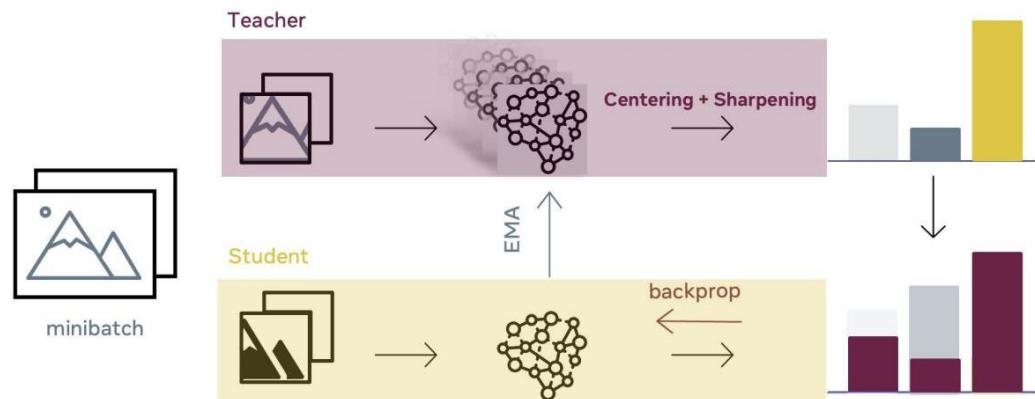
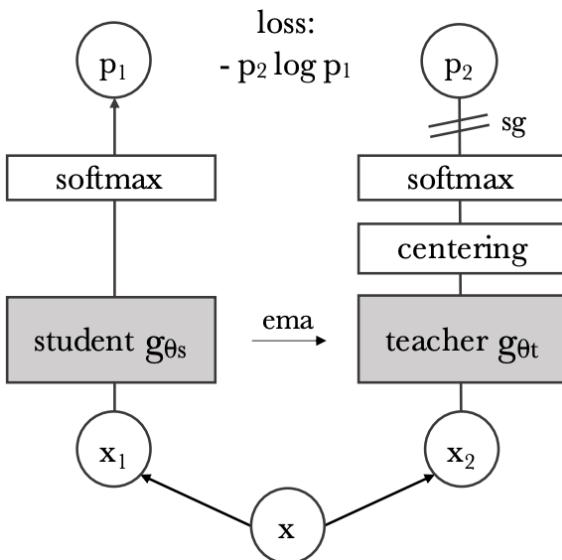
² Inria*

³ Sorbonne University



Figure 1: **Self-attention from a Vision Transformer with 8×8 patches trained with no supervision.** We look at the self-attention of the [CLS] token on the heads of the last layer. This token is not attached to any label nor supervision. These maps show that the model automatically learns class-specific features leading to unsupervised object segmentations.

DINO



$$\theta_{\text{teacher}} \leftarrow \tau \cdot \theta_{\text{teacher}} + (1 - \tau) \cdot \theta_{\text{student}}$$

Caron et al. 2021 Emerging Properties in
Self-Supervised Vision Transformers

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()
```

DINO v2

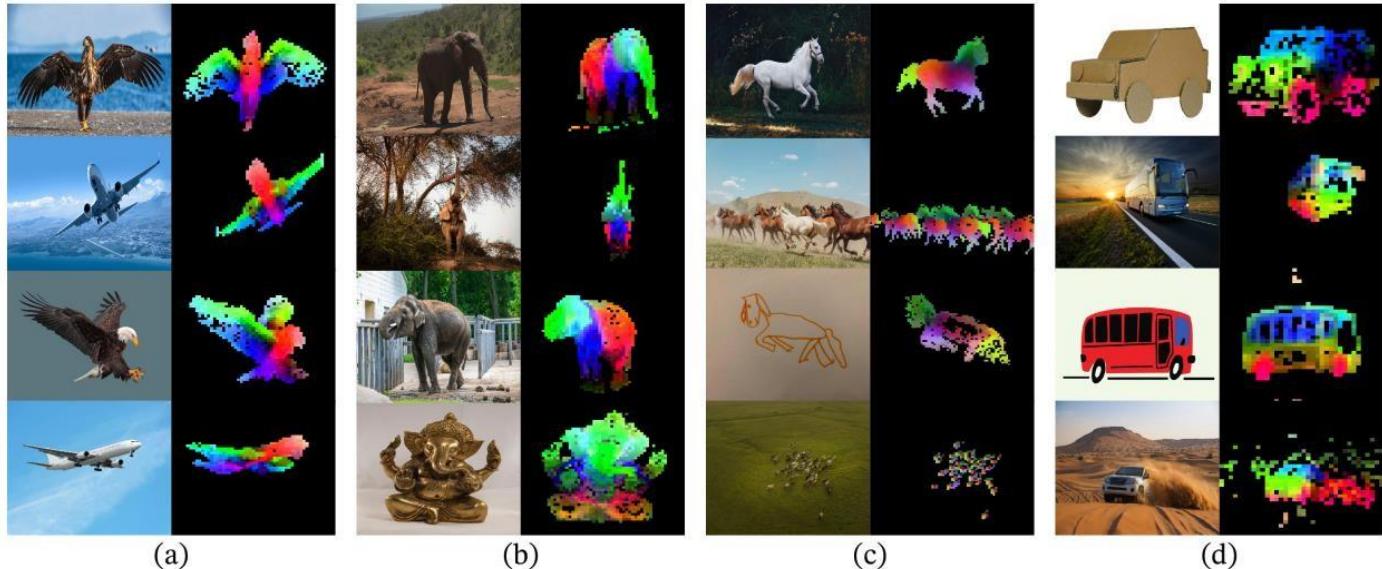
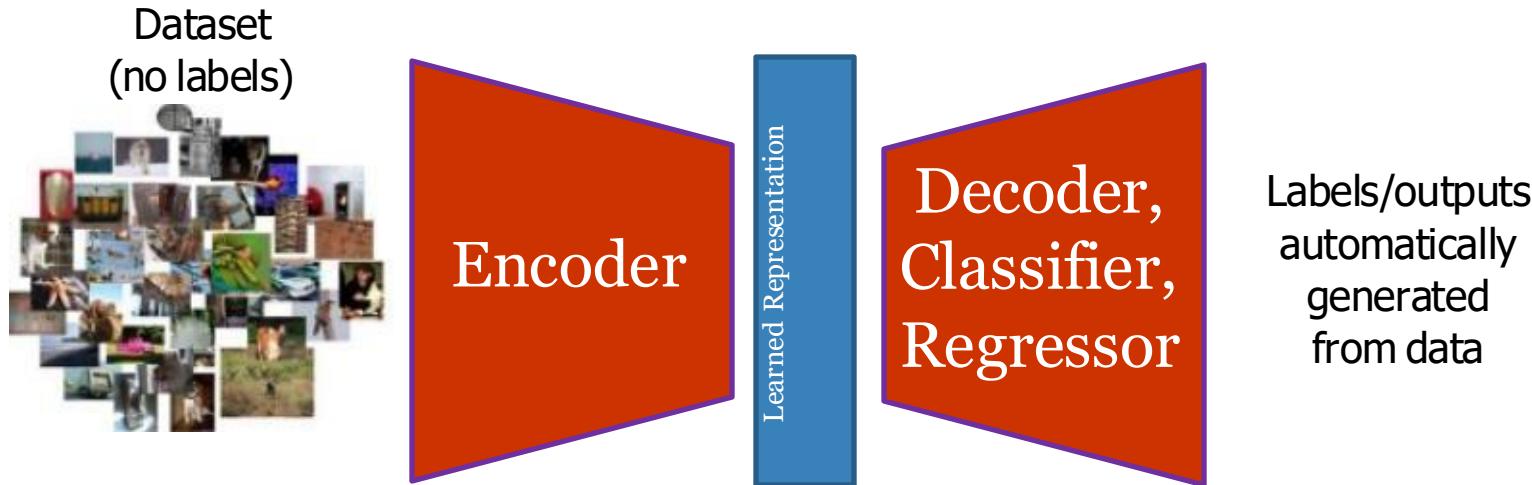


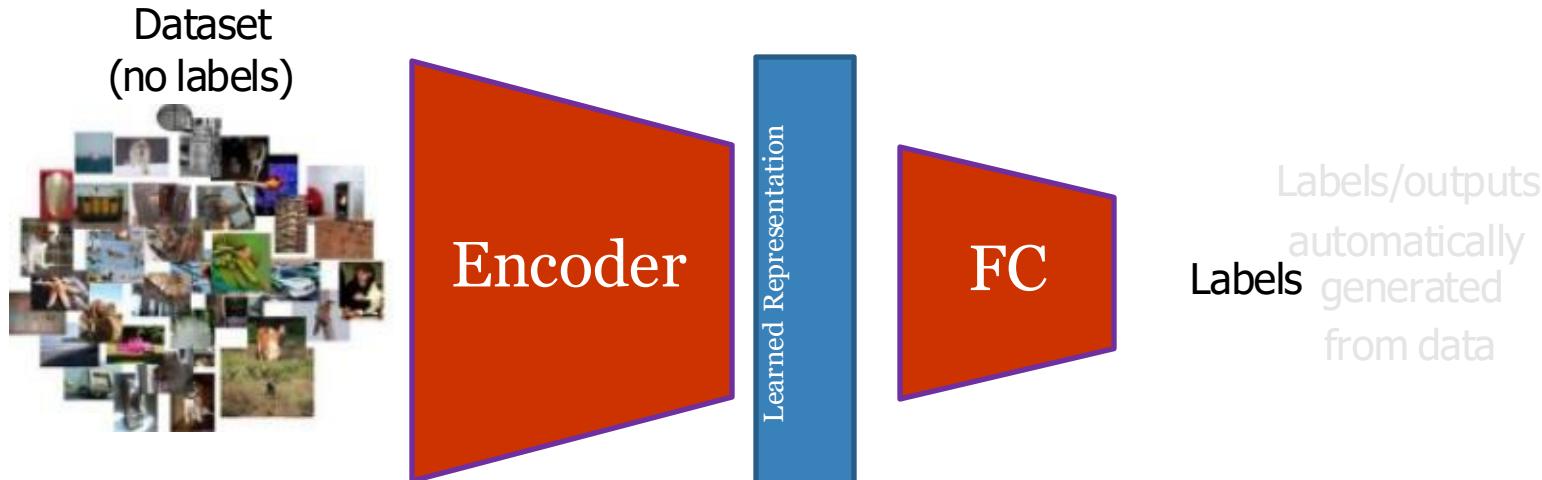
Figure 1: Visualization of the first PCA components. We compute a PCA between the patches of the images from the same column (a, b, c and d) and show their first 3 components. Each component is matched to a different color channel. Same parts are matched between related images despite changes of pose, style or even objects. Background is removed by thresholding the first PCA component.

Lecture 13: Generative Models (part 1)

Last Time: Self-Supervised Learning



Last Time: Self-Supervised Learning



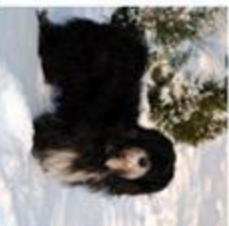
Last Time: Self-Supervised Learning

Pretext tasks from image transformations

- Rotation, inpainting, rearrangement, coloring
- Reconstruction-based learning (MAE)



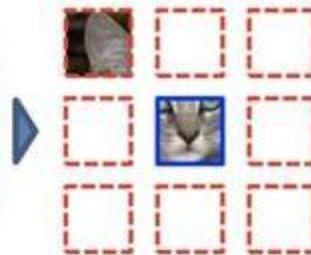
90° rotation



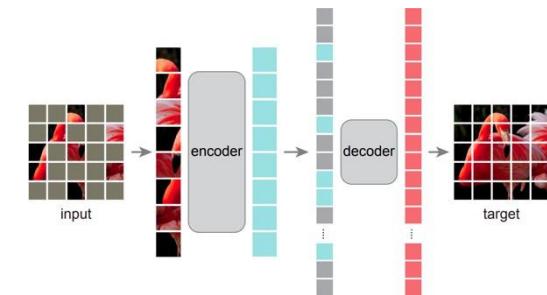
270° rotation

Rotation

Example:



Rearrangement



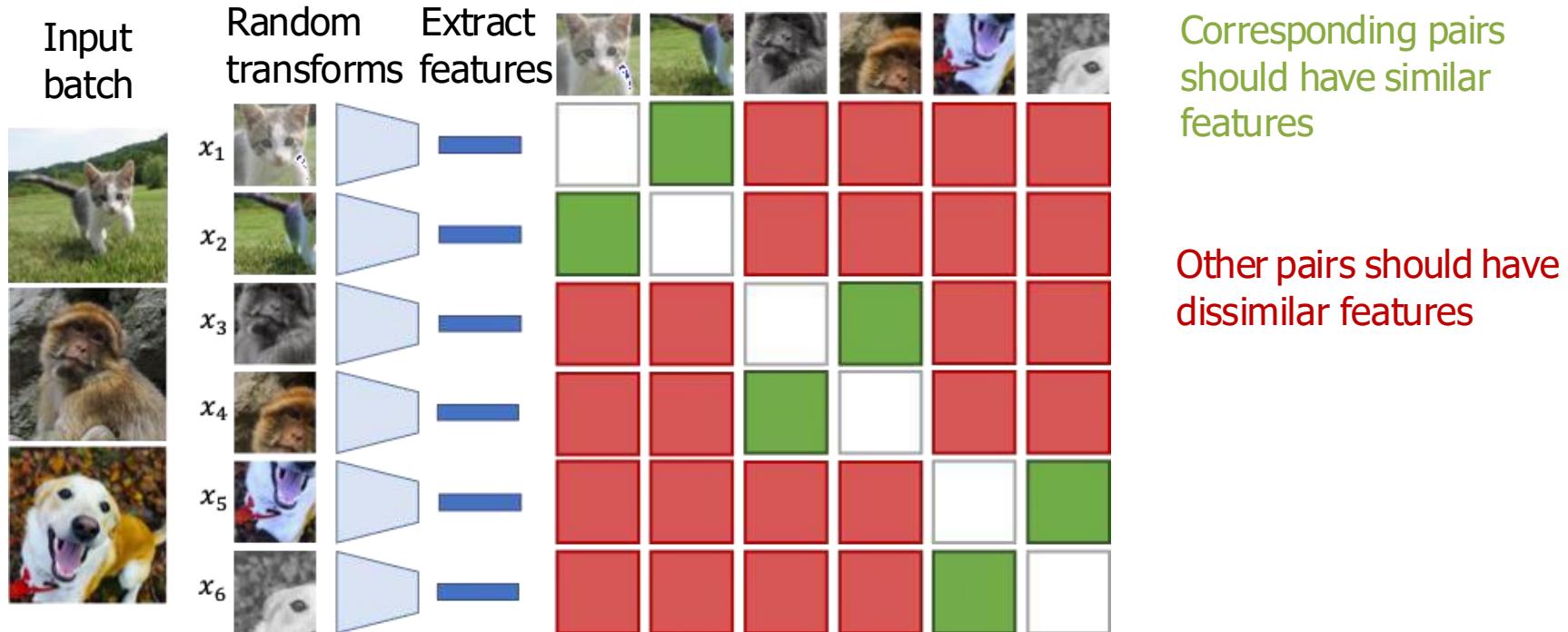
Reconstruction

Last Time: Self-Supervised Learning

Contrastive representation learning

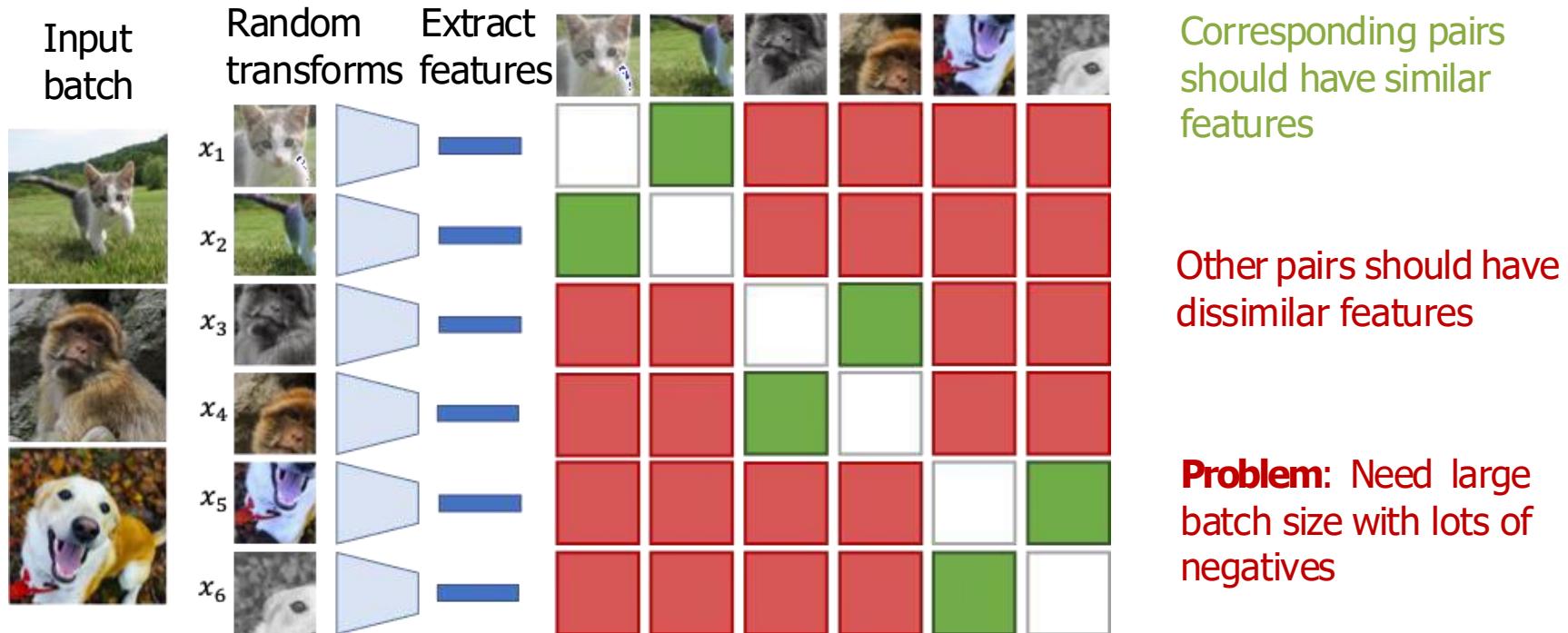
- Intuition and formulation
- Instance contrastive learning: SimCLR and MOCO
- Sequence contrastive learning: CPC
- Self-Distillation Without Labels, DINO

Last Time: Contrastive Learning (SimCLR)



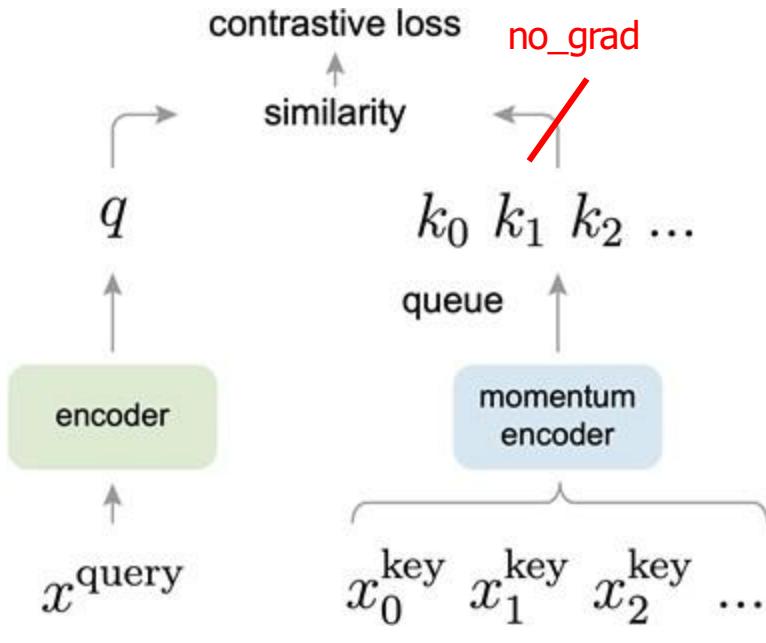
Chen et al, "A simple framework for contrastive learning of visual representations", ICML 2020

Last Time: Contrastive Learning (SimCLR)



Chen et al, "A simple framework for contrastive learning of visual representations", ICML 2020

Contrastive Learning: MoCo

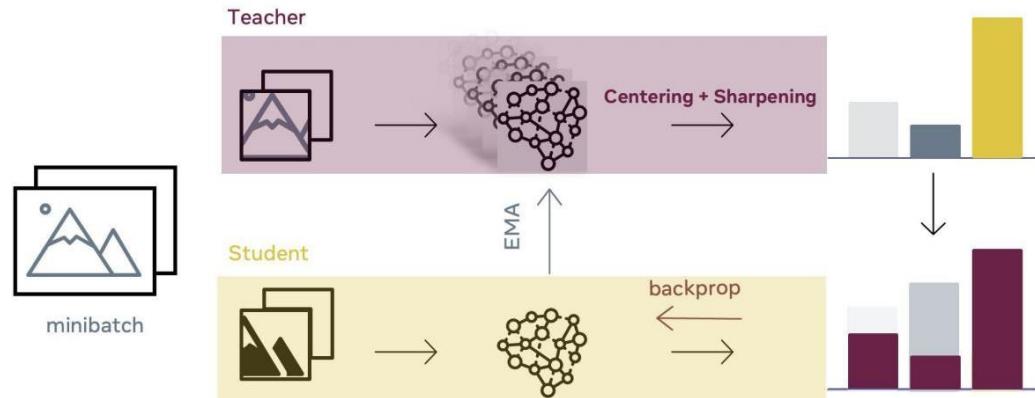
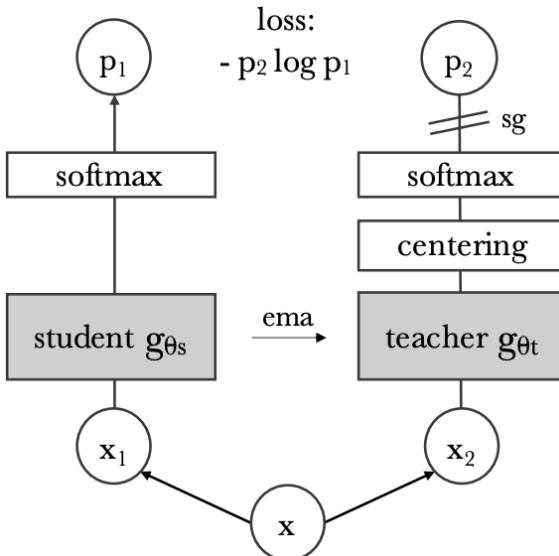


Key differences to SimCLR:

- Keep a running **queue** of keys (negative samples).
- Compute gradients and update the encoder **only through the queries**.
- Decouple min-batch size with the number of keys: can support **a large number of negative samples**.
- The key encoder is **slowly progressing** through the momentum update rules:
$$\theta_k \leftarrow m\theta_k + (1 - m)\theta_q$$

Self-Supervised Learning: DINO

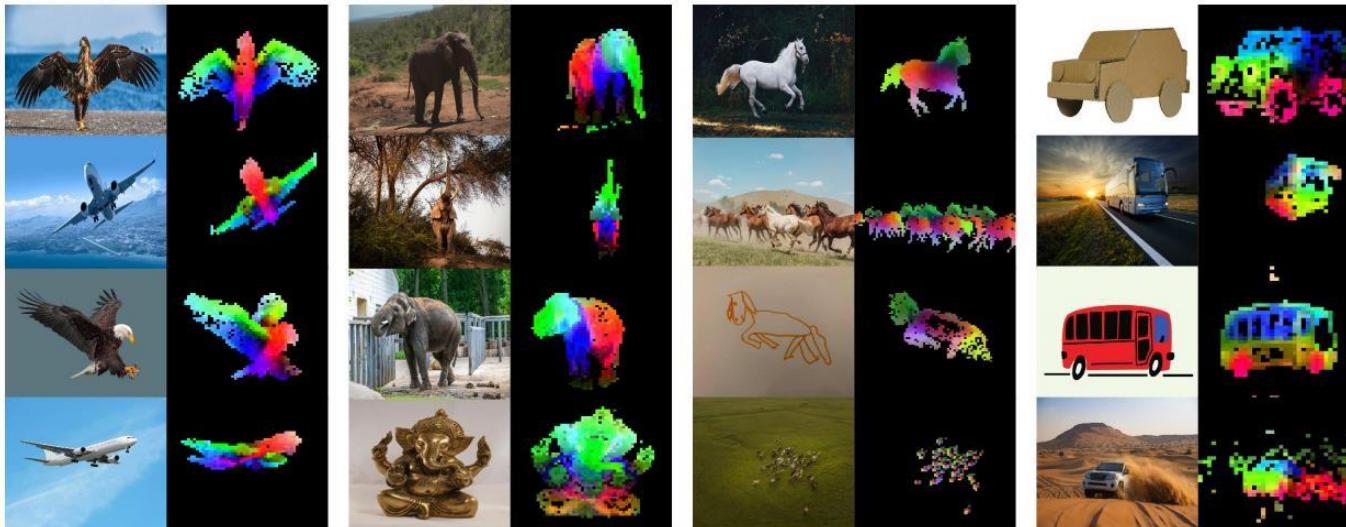
Similar in spirit to MoCo, but matches features using KL divergence instead of dot product, and uses Vision Transformers instead of ResNets



$$\theta_{\text{teacher}} \leftarrow \tau \cdot \theta_{\text{teacher}} + (1 - \tau) \cdot \theta_{\text{student}}$$

Self-Supervised Learning: DINOv2

Scales up training data from 1M ImageNet images to 142M images
Very strong image features, commonly used in practice



PCA feature
visualization

Today:
Generative Models (part 1)

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a function to map $x \rightarrow y$

Examples: Classification, regression,
object detection, semantic
segmentation, image captioning, etc.

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a function to map $x \rightarrow y$

Examples: Classification, regression,
object detection, semantic
segmentation, image captioning, etc.



→ Cat

Classification

[This image is CC0 public domain](#)

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a function to map $x \rightarrow y$

Examples: Classification, regression,
object detection, semantic
segmentation, image captioning, etc.



A cat sitting on a suitcase on the floor

Image captioning

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a function to map $x \rightarrow y$

Examples: Classification, regression,
object detection, semantic
segmentation, image captioning, etc.



DOG, DOG, CAT

Object Detection

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a function to map $x \rightarrow y$

Examples: Classification, regression,
object detection, semantic
segmentation, image captioning, etc.



GRASS, CAT,
TREE, SKY

Semantic Segmentation

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a function to map $x \rightarrow y$

Examples: Classification, regression,
object detection, semantic
segmentation, image captioning, etc.

Unsupervised Learning

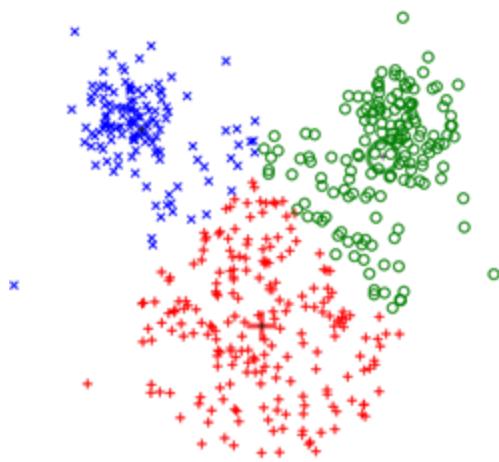
Data: x

Just data, no labels!

Goal: Learn hidden structure in data

Examples: Clustering, dimensionality
reduction, density estimation, etc.

Supervised vs Unsupervised Learning



K-means clustering

Unsupervised Learning

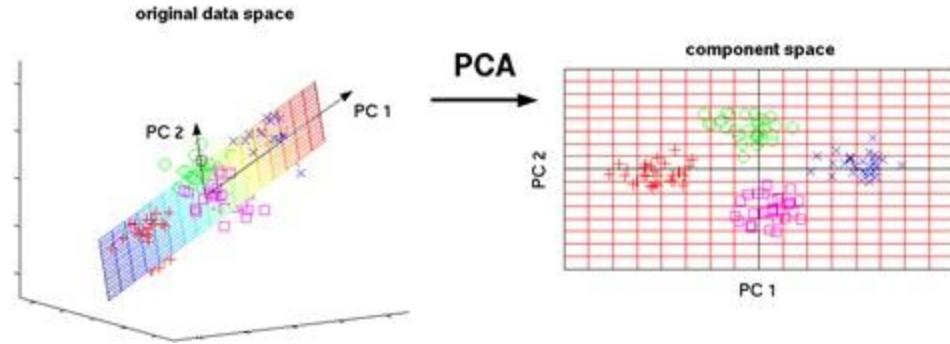
Data: x

Just data, no labels!

Goal: Learn hidden structure in data

Examples: Clustering, dimensionality reduction, density estimation, etc.

Supervised vs Unsupervised Learning



3-d → 2-d

Principal Component Analysis
(Dimensionality reduction)

Unsupervised Learning

Data: x

Just data, no labels!

Goal: Learn hidden structure in data

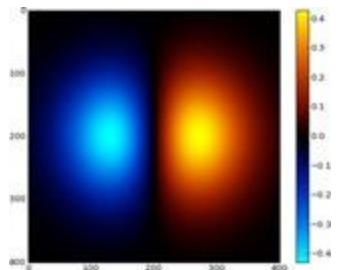
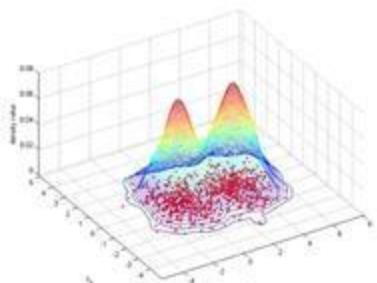
Examples: Clustering, dimensionality reduction, density estimation, etc.

Supervised vs Unsupervised Learning



Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

Modeling $p(x)$

Unsupervised Learning

Data: x

Just data, no labels!

Goal: Learn hidden structure in data

Examples: Clustering, dimensionality reduction, density estimation, etc.

Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$

Data: x



Generative Model:

Learn a probability distribution $p(x)$

Conditional Generative Model:

Learn $p(x|y)$

Label: y

Cat

Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$

Generative Model:

Learn a probability distribution $p(x)$

Conditional Generative Model:

Learn $p(x|y)$

Data: x



Label: y

Cat

Probability Recap:

Density Function

$p(x)$ assigns a positive number to each possible x ; higher numbers mean x is more likely.

Density functions are **normalized**:

$$\underset{X}{\mathbb{E}} p(x) dx = 1$$

Different values of x compete for density

Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$

Data: x



$P(\text{cat} | \text{ })$

$P(\text{dog} | \text{ })$



Label: y

Cat

$$\underset{X}{\approx} p(x)dx = 1$$

Different values of x
compete for density

Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$



$$P(\text{cat} | \text{img})$$



$$P(\text{dog} | \text{img})$$



$$P(\text{dog} | \text{img})$$

$$P(\text{cat} | \text{img})$$

Possible **labels** for each image compete for probability.
No competition between **images**

Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$



$$P(\text{cat}|\text{monkey})$$



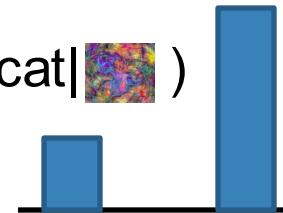
$$P(\text{dog}|\text{monkey})$$



$$P(\text{dog}|\text{colorful image})$$



$$P(\text{cat}|\text{colorful image})$$



No way to handle unreasonable inputs; must give a label distribution for all possible inputs

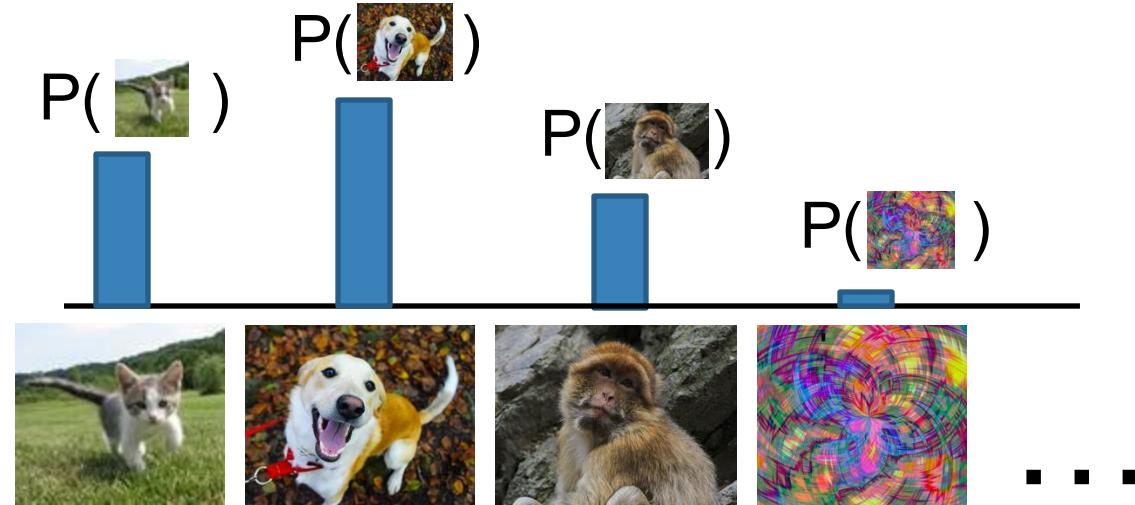
Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$

Generative Model:

Learn a probability distribution $p(x)$



All possible **images** compete for probability mass

Conditional Generative Model: Learn $p(x|y)$

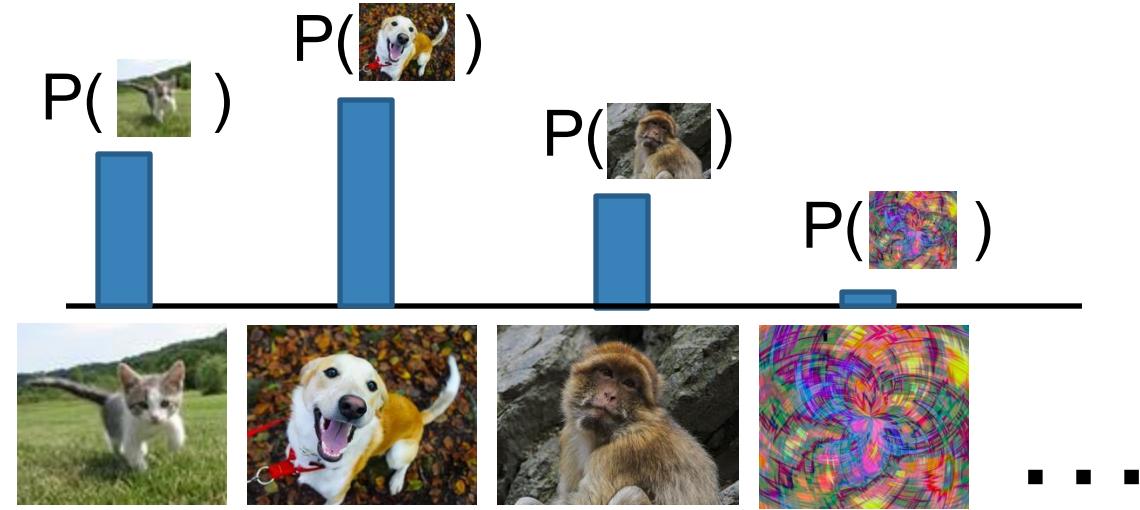
Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$

Generative Model:

Learn a probability distribution $p(x)$



Conditional Generative Model: Learn $p(x|y)$

All possible **images** compete for probability mass

Requires deep understanding: Is a dog more likely to sit or stand? Is a 3-legged dog more likely than a 3-armed monkey?

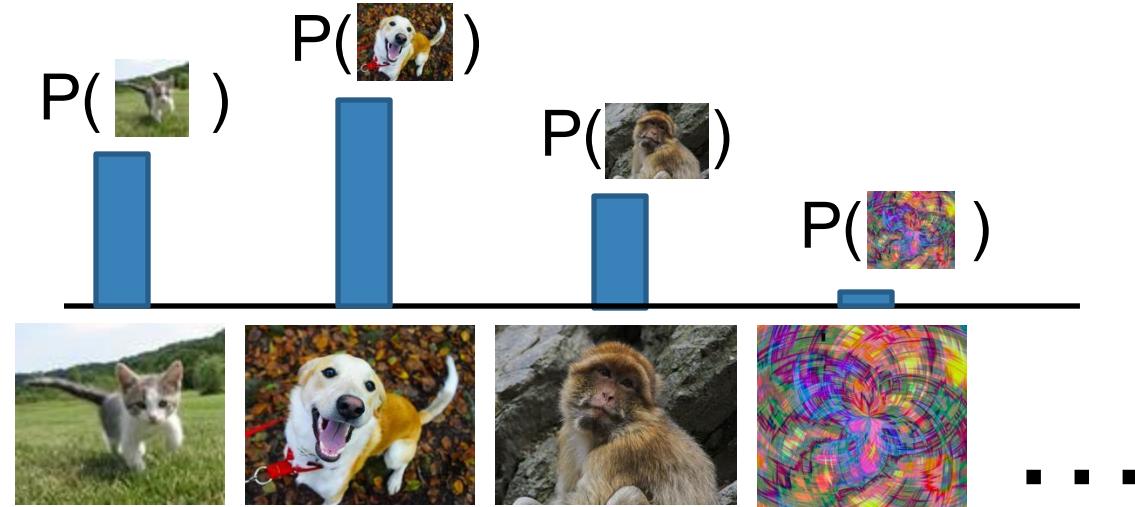
Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$

Generative Model:

Learn a probability distribution $p(x)$

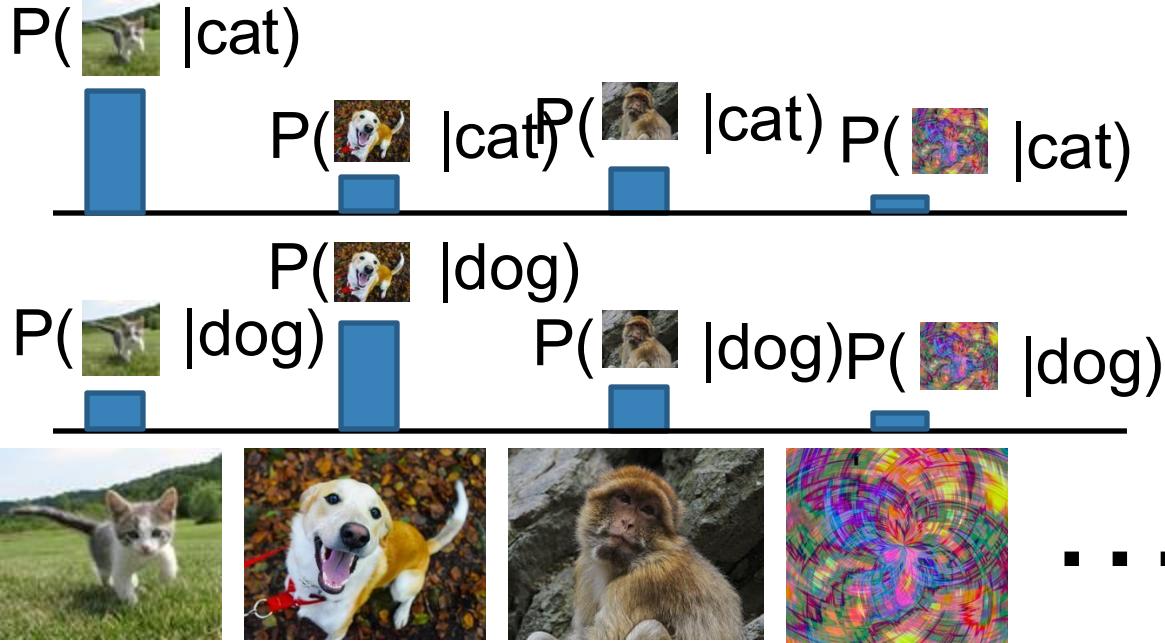


Conditional Generative Model: Learn $p(x|y)$

All possible **images** compete for probability mass

Model can “reject” unreasonable inputs by giving them small probability mass

Generative vs Discriminative Models



Conditional Generative Model: Learn $p(x|y)$

Each possible **label** induces a competition across all possible **images**

Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$

Recall **Bayes' Rule**:

Generative Model:

Learn a probability distribution $p(x)$

$$P(x | y) = \frac{P(y | x)}{P(y)} P(x)$$

Conditional Generative Model:

Learn $p(x|y)$

Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$

Generative Model:

Learn a probability distribution $p(x)$

Conditional Generative Model:

Learn $p(x|y)$

Recall **Bayes' Rule**:

$$P(x | y) = \frac{P(y | x)}{P(y)} P(x)$$

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

We can build a conditional generative model from other components ... but not common in practice

Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$



Assign labels to data
Feature learning (with labels)

Generative Model:

Learn a probability distribution $p(x)$

Conditional Generative Model: Learn $p(x|y)$

Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$



Assign labels to data
Feature learning (with labels)

Generative Model:

Learn a probability distribution $p(x)$



Detect outliers
Feature learning (without labels)
Sample to generate new data

Conditional Generative Model: Learn $p(x|y)$

Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$



Assign labels to data
Feature learning (with labels)

Generative Model:

Learn a probability distribution $p(x)$



Detect outliers
Feature learning (without labels)
Sample to generate new data

Conditional Generative Model: Learn $p(x|y)$



Assign labels while rejecting outliers
Sample to generate data from labels

Generative vs Discriminative Models

Discriminative Model:

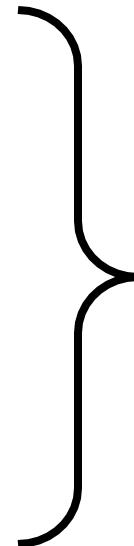
Learn a probability distribution $p(y|x)$

Generative Model:

Learn a probability distribution $p(x)$

Conditional Generative Model:

Learn $p(x|y)$



"Generative models" means either of these; conditional generative models are most common in practice

Why Generative Models?

Modeling ambiguity: If there are many possible outputs x for an input y , we want to model $P(x | y)$

Language Modeling: Produce output text x from input text y

*Write me a short
rhyming poem about
generative models*



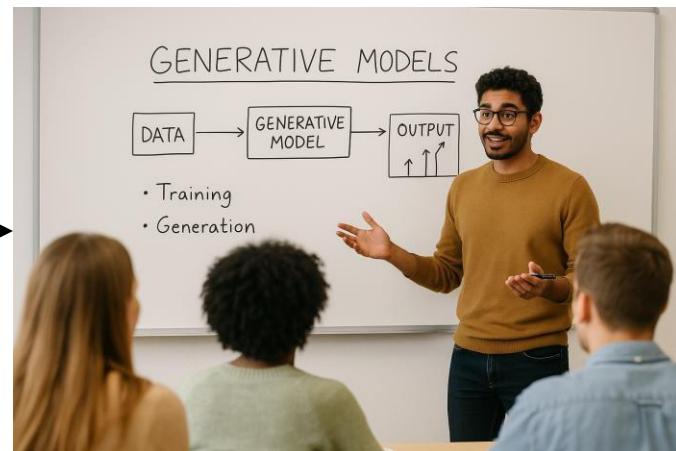
*They sample from a learned P ,
A distribution—structured, free.
Each token comes conditionally,
On all the ones that used to be.*

Why Generative Models?

Modeling ambiguity: If there are many possible outputs x for an input y , we want to model $P(x | y)$

Text to Image: Produce output image x from input text y

*Make me an image showing
a person teaching a class on
generative models in front of
a whiteboard*



Why Generative Models?

Modeling ambiguity: If there are many possible outputs x for an input y , we want to model $P(x | y)$

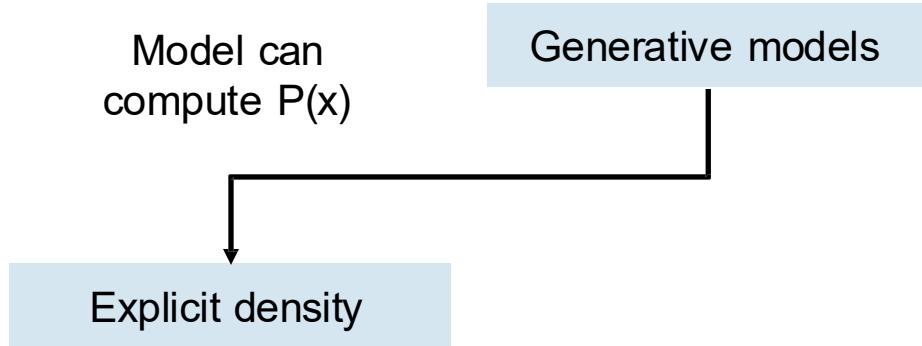
Image to Video: What happens next?



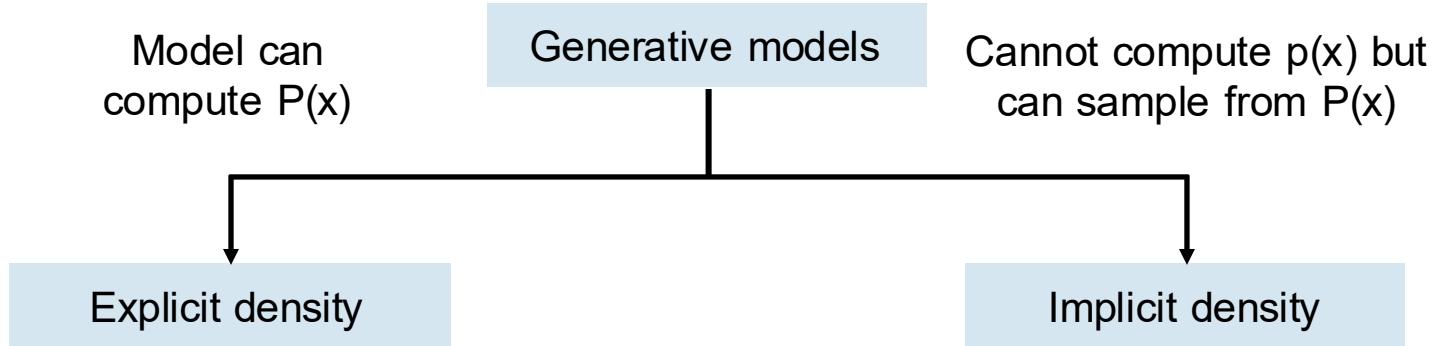
Taxonomy of Generative Models

Generative models

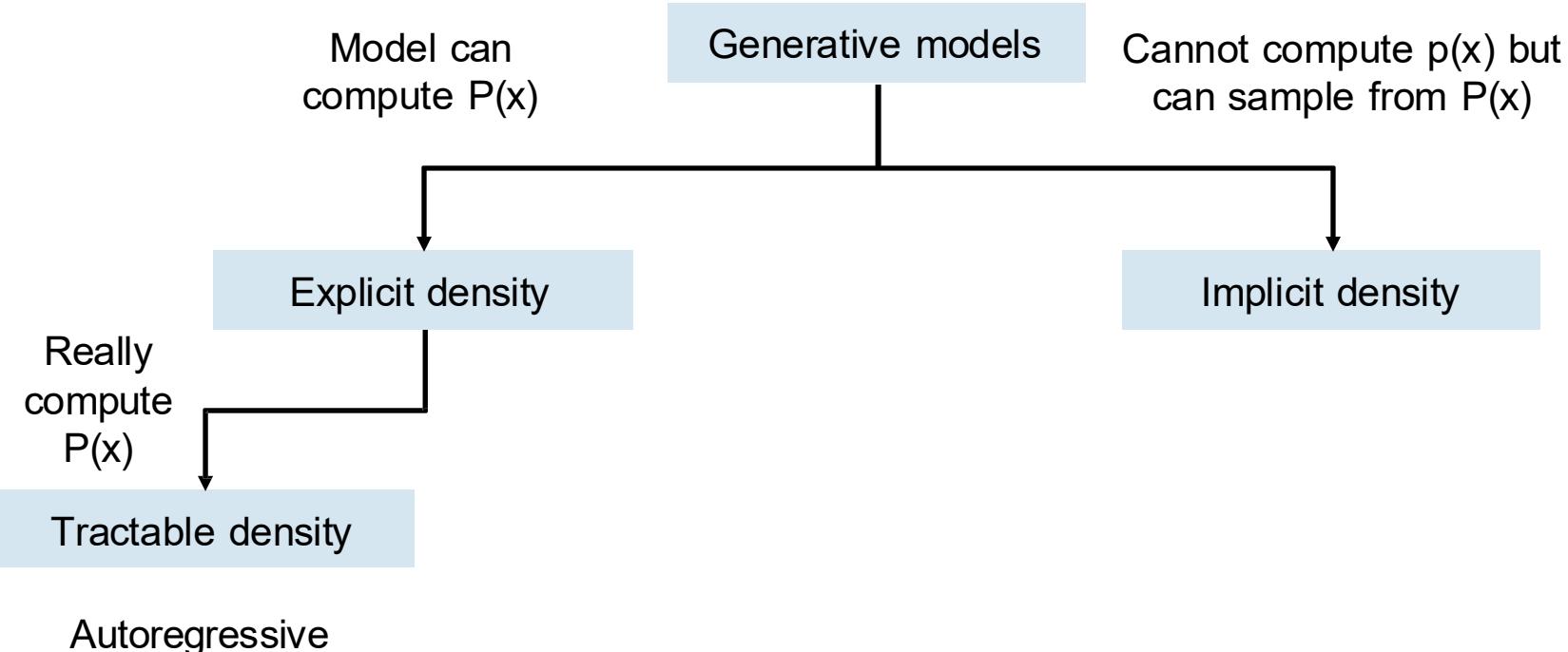
Taxonomy of Generative Models



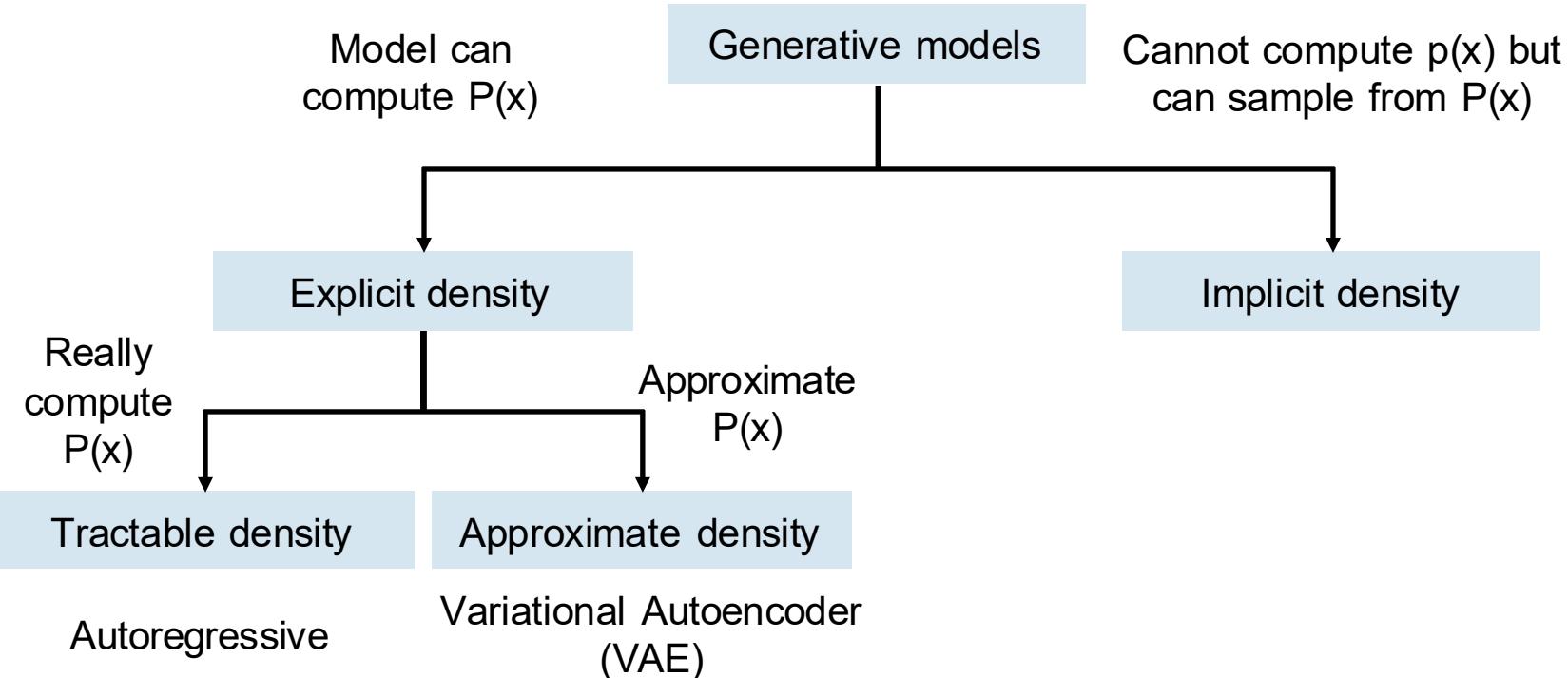
Taxonomy of Generative Models



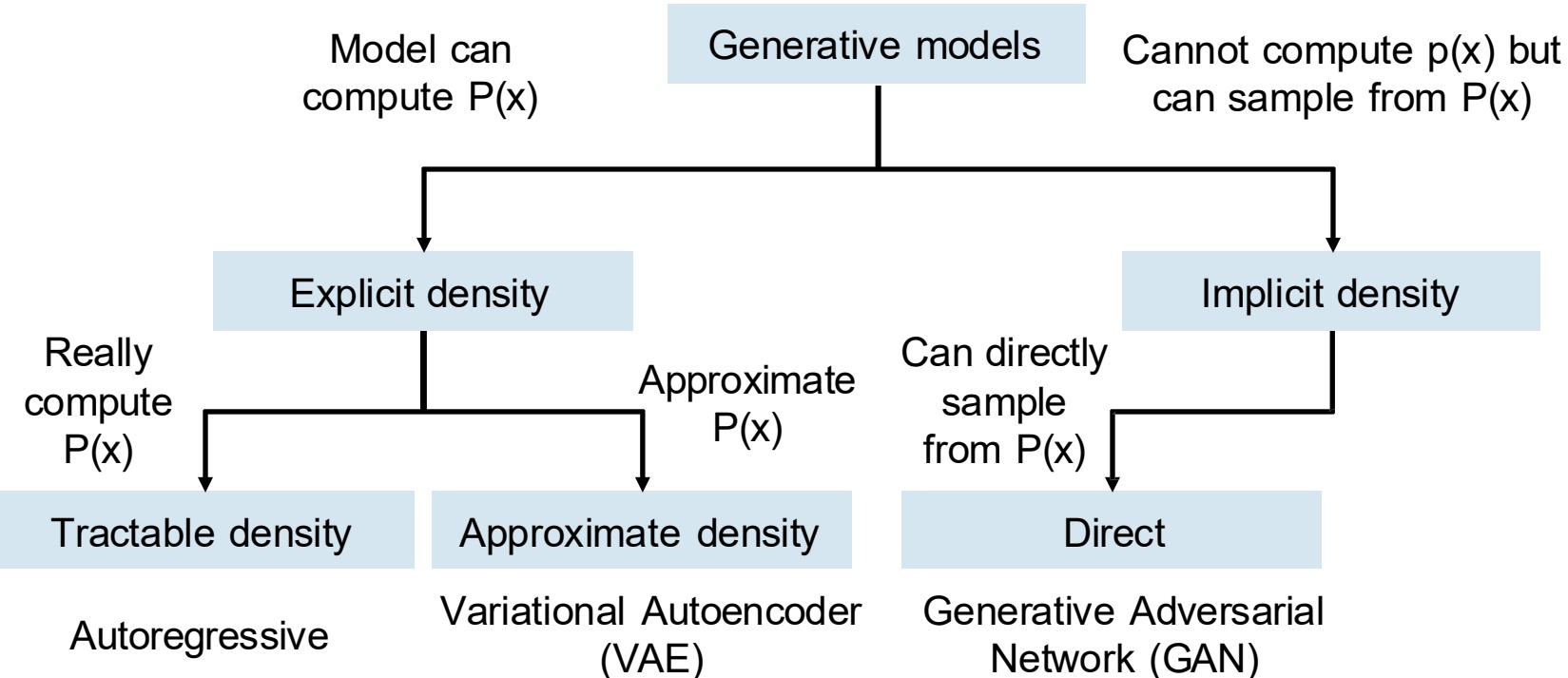
Taxonomy of Generative Models



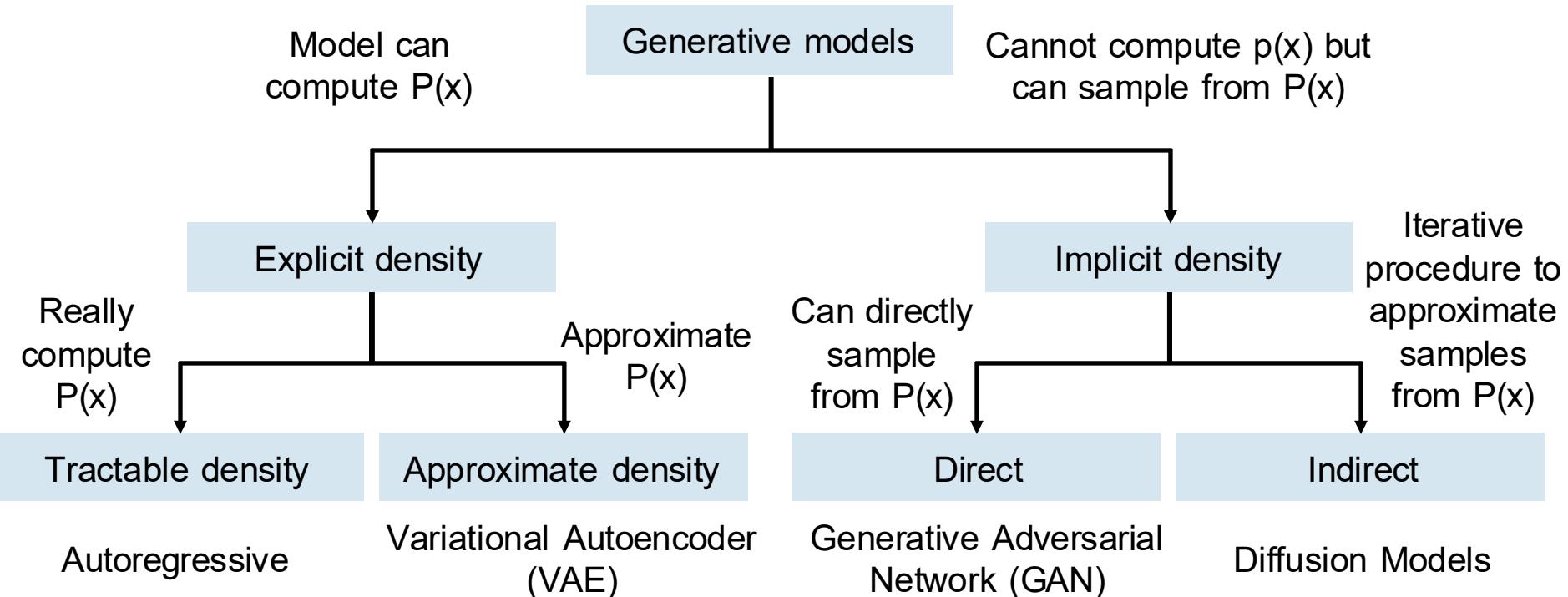
Taxonomy of Generative Models



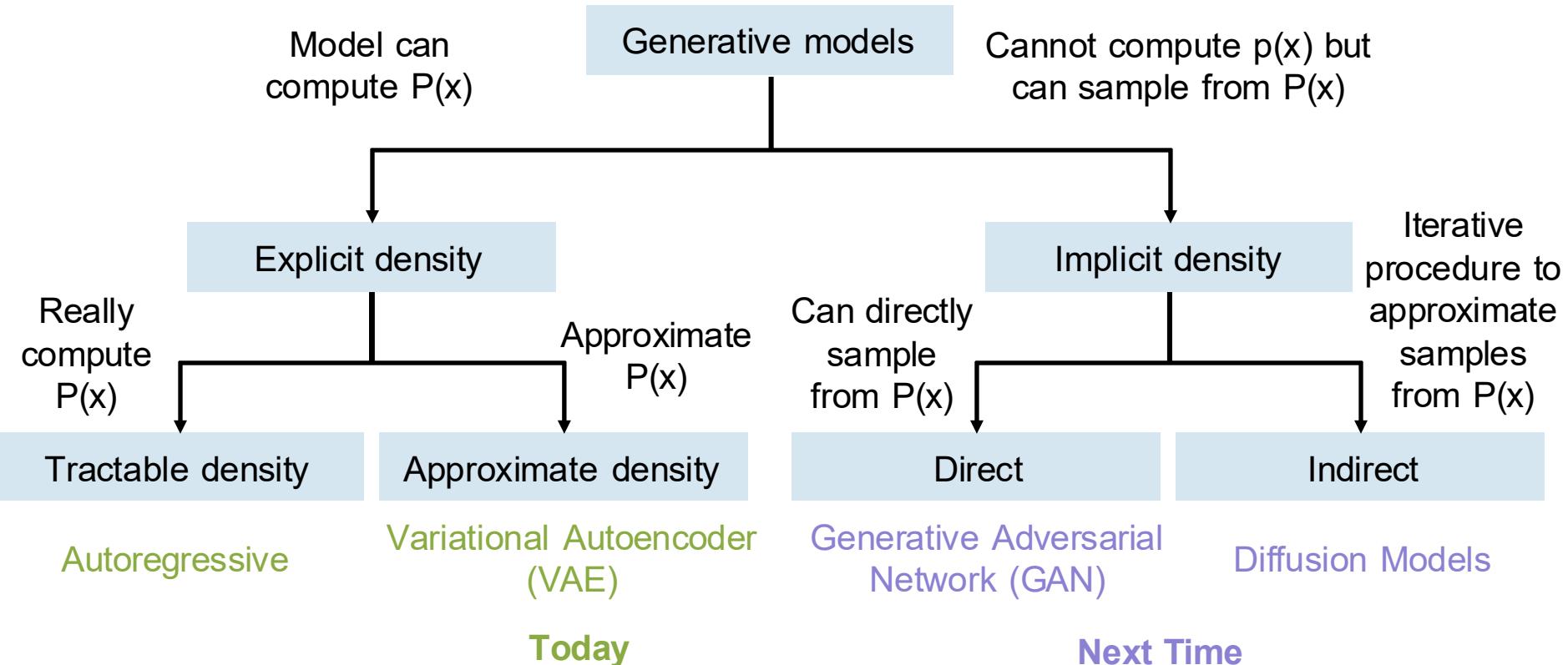
Taxonomy of Generative Models



Taxonomy of Generative Models



Taxonomy of Generative Models



Autoregressive Models

Maximum Likelihood Estimation

Goal: Write down an explicit function for $p(x) = f(x, W)$

Maximum Likelihood Estimation

Goal: Write down an explicit function for $p(x) = f(x, W)$

Given dataset $x^{(1)}, x^{(2)}, \dots x^{(N)}$, train the model by solving:

$$W^* = \arg \max_W \prod_i p(x^{(i)})$$

Maximize probability of training data
(Maximum likelihood estimation)

Maximum Likelihood Estimation

Goal: Write down an explicit function for $p(x) = f(x, W)$

Given dataset $x^{(1)}, x^{(2)}, \dots x^{(N)}$, train the model by solving:

$$W^* = \arg \max_W \prod_i p(x^{(i)})$$

Maximize probability of training data
(Maximum likelihood estimation)

$$= \arg \max_W \sum_i \log p(x^{(i)})$$

Log trick: Swap product and sum

Maximum Likelihood Estimation

Goal: Write down an explicit function for $p(x) = f(x, W)$

Given dataset $x^{(1)}, x^{(2)}, \dots x^{(N)}$, train the model by solving:

$$W^* = \arg \max_W \prod_i p(x^{(i)})$$

Maximize probability of training data
(Maximum likelihood estimation)

$$= \arg \max_W \sigma_i \log p(x^{(i)})$$

Log trick: Swap product and sum

$$= \arg \max_W \sigma_i \log f(x^{(i)}, W)$$

This is our loss function.
maximize it with gradient descent

Autoregressive Models

Goal: Write down an explicit function for $p(x) = f(x, W)$

Assume x is a sequence: $x = (x_1, x_2, \dots, x_T)$

Autoregressive Models

Goal: Write down an explicit function for $p(x) = f(x, W)$

Assume x is a sequence: $x = (x_1, x_2, \dots, x_T)$

Use the chain rule of probability:

$$\begin{aligned} p(x) &= p(x_1, x_2, x_3, \dots, x_T) \\ &= p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \dots \\ &= \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1}) \end{aligned}$$

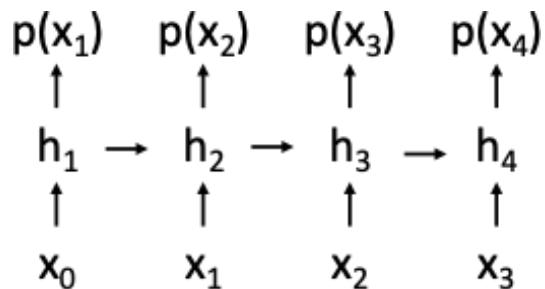
Autoregressive Models

Goal: Write down an explicit function for $p(x) = f(x, W)$

Assume x is a sequence:

$$x = (x_1, x_2, \dots, x_T)$$

We have already seen this!



Use the chain rule of probability:

$$\begin{aligned} p(x) &= p(x_1, x_2, x_3, \dots, x_T) \\ &= p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \dots \\ &= \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1}) \end{aligned}$$

Language modeling with RNN

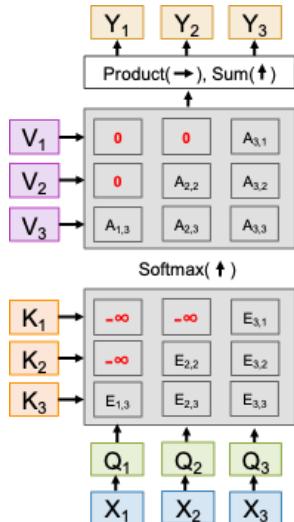
LLMs are Autoregressive Models

Goal: Write down an explicit function for $p(x) = f(x, W)$

Assume x is a sequence:

$$x = (x_1, x_2, \dots, x_T)$$

Language
modeling
with masked
Transformer



Use the chain rule of probability:

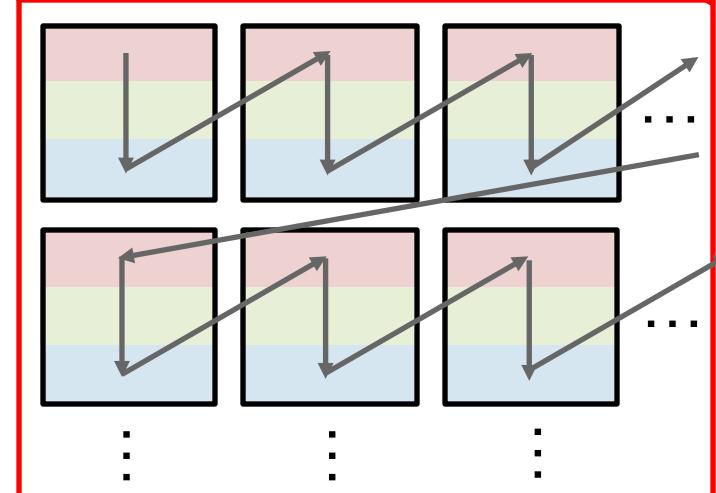
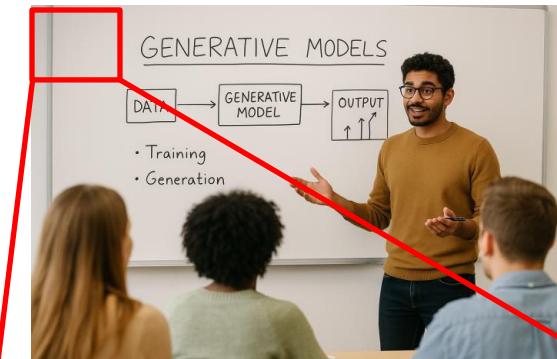
$$\begin{aligned} p(x) &= p(x_1, x_2, x_3, \dots, x_T) \\ &= p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \dots \\ &= \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1}) \end{aligned}$$

Autoregressive Models of Images

Treat an image as a sequence of 8-bit subpixel values (scanline order)

Predict each subpixel as a classification among 256 values [0...255]

Model with an RNN or Transformer



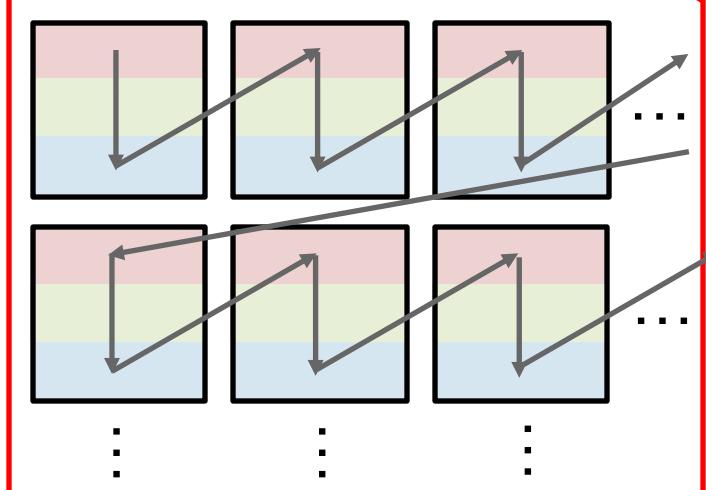
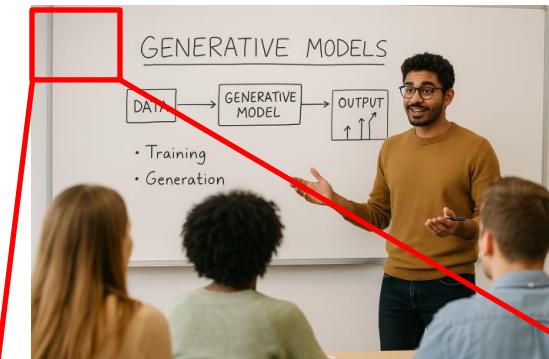
Autoregressive Models of Images

Treat an image as a sequence of 8-bit subpixel values (scanline order)

Predict each subpixel as a classification among 256 values [0...255]

Model with an RNN or Transformer

Problem: Too expensive. 1024x1024 image is a sequence of 3M subpixels



Autoregressive Models of Images

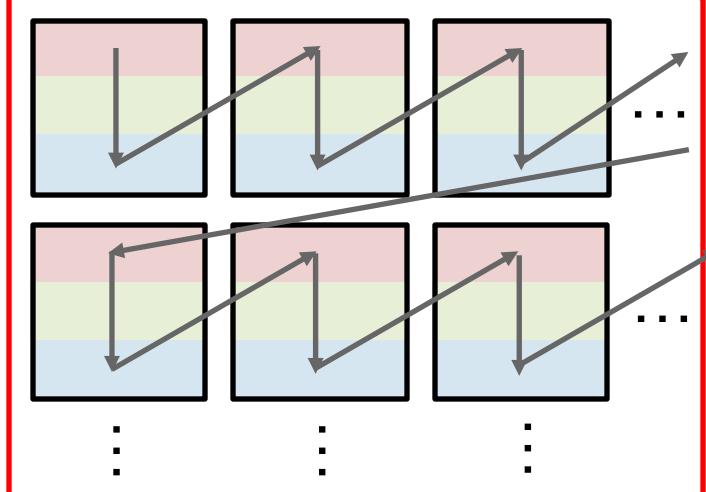
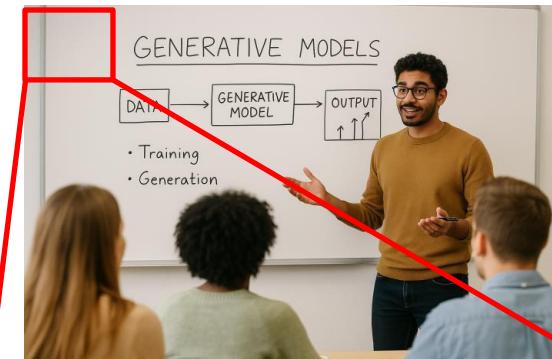
Treat an image as a sequence of 8-bit subpixel values (scanline order)

Predict each subpixel as a classification among 256 values [0...255]

Model with an RNN or Transformer

Problem: Too expensive. 1024x1024 image is a sequence of 3M subpixels

Solution (jumping ahead): Model as sequence of tiles, not sequence of subpixels



Variational Autoencoders (VAEs)

Variational Autoencoders

PixelRNN / PixelCNN explicitly parameterizes density function with a neural network, so we can train to maximize likelihood of training data:

$$p_W(x) = \prod_{t=1}^T p_W(x_t | x_1, \dots, x_{t-1})$$

Variational Autoencoders (VAE) define an **intractable density** that we cannot explicitly compute or optimize

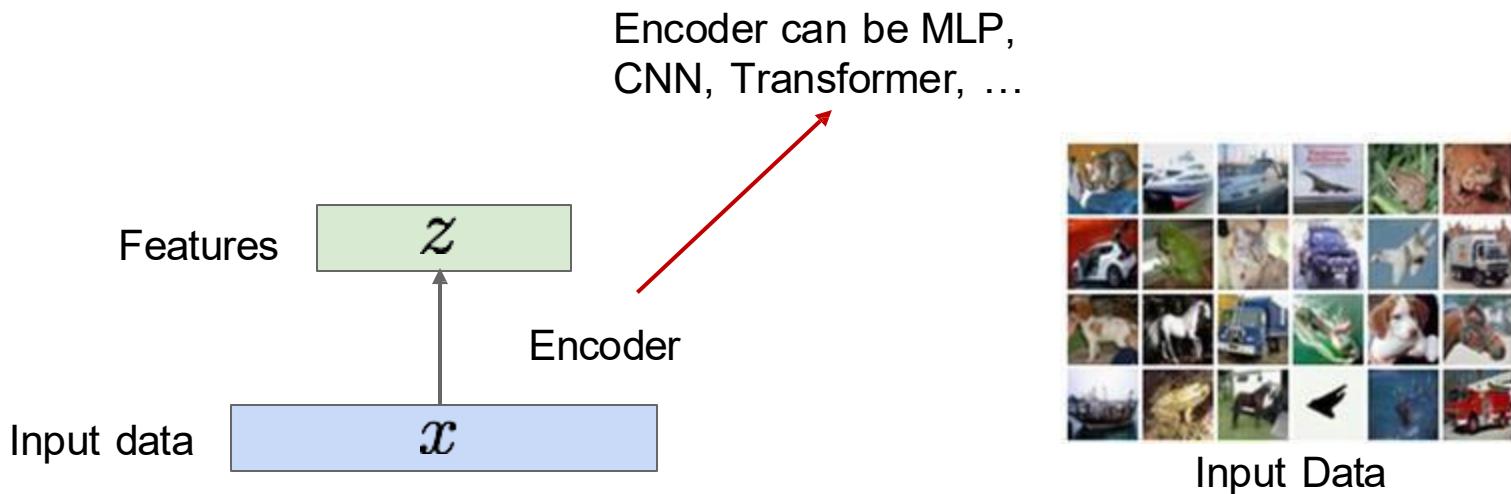
But we will be able to directly optimize a **lower bound** on the density

Variational Autoencoders (VAEs)

(Non-Variational) Autoencoders

Idea: Unsupervised method for learning to extract features z from inputs x , without labels

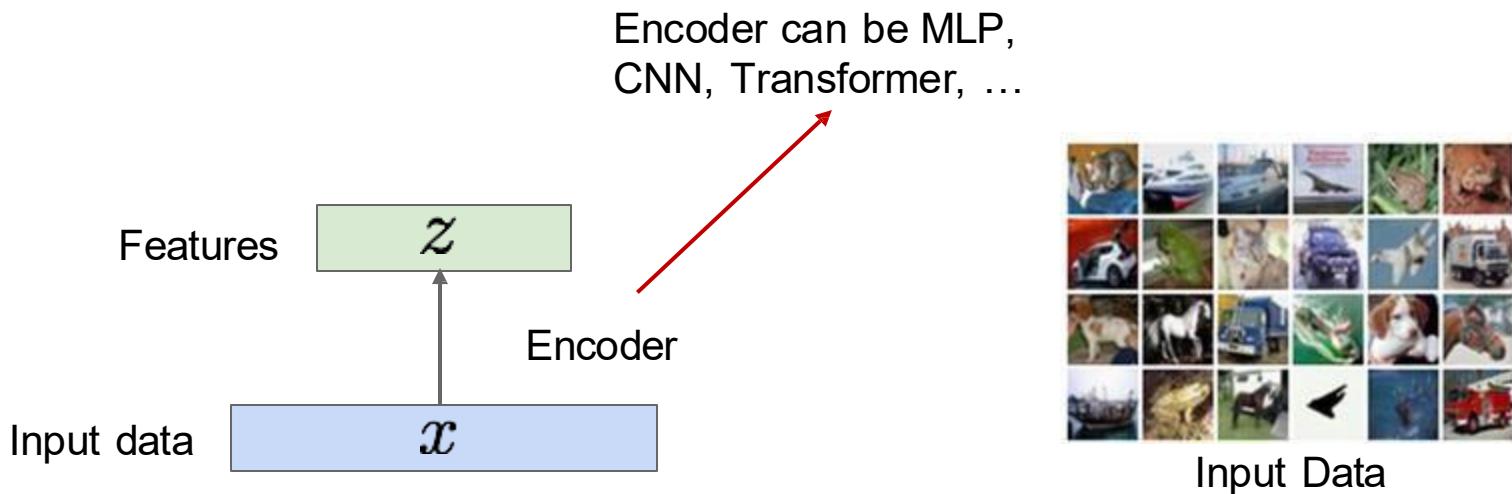
Features should extract useful information
(object identity, appearance, scene type, etc)
that can be used for downstream tasks



(Non-Variational) Autoencoders

Problem: How can we learn without labels?

Features should extract useful information
(object identity, appearance, scene type, etc)
that can be used for downstream tasks



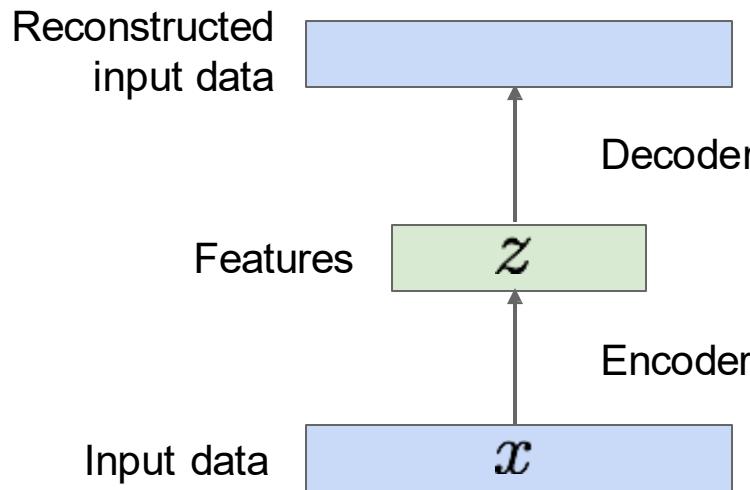
(Non-Variational) Autoencoders

“Autoencoding” =
Encoding yourself

Problem: How can we learn without labels?

Solution: Reconstruct the input data with a decoder.

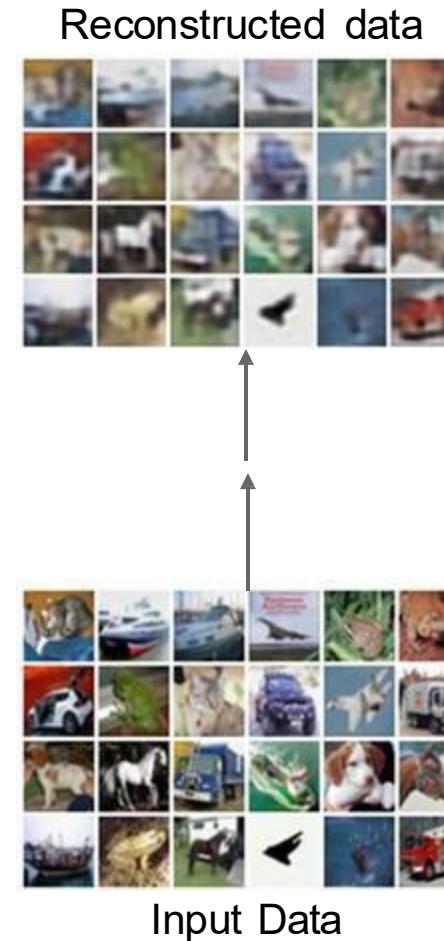
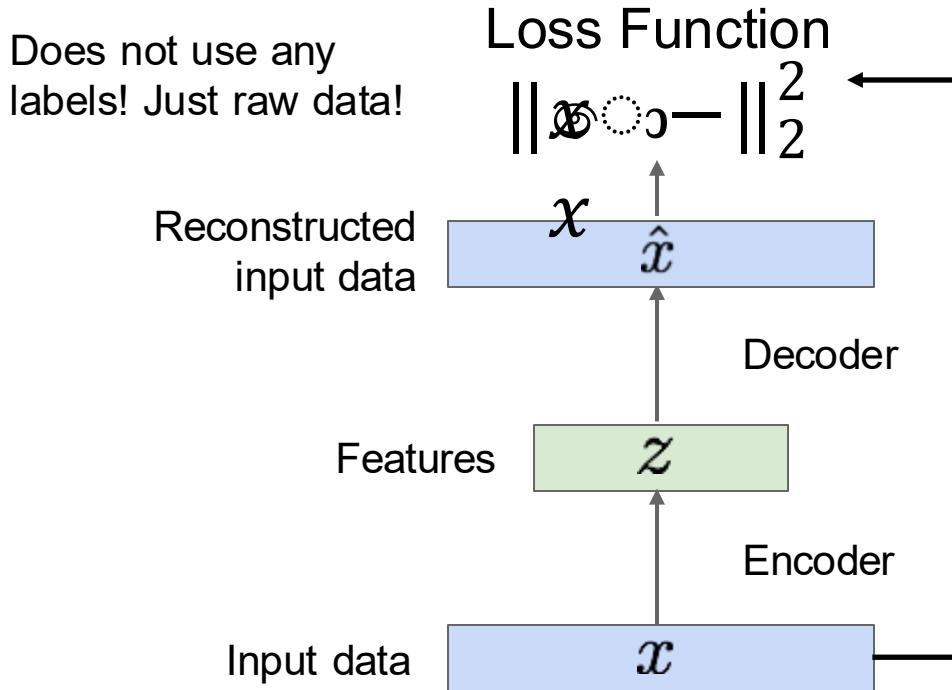
Decoder can be MLP,
CNN, Transformer, ...



Input Data

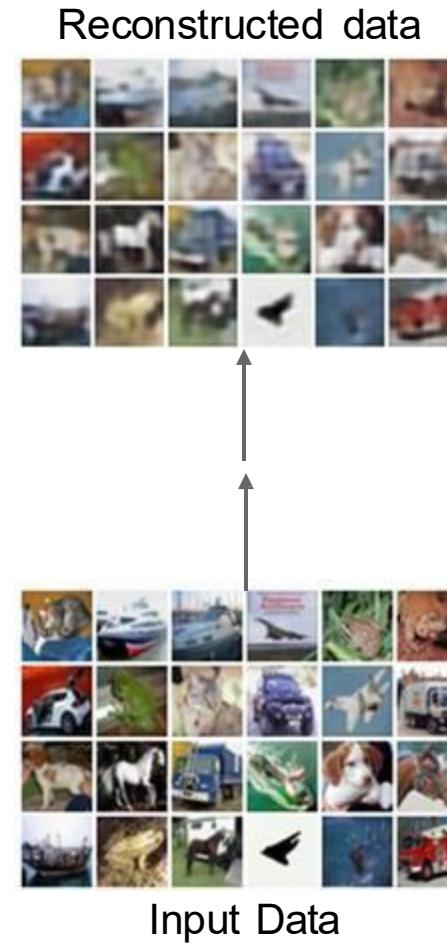
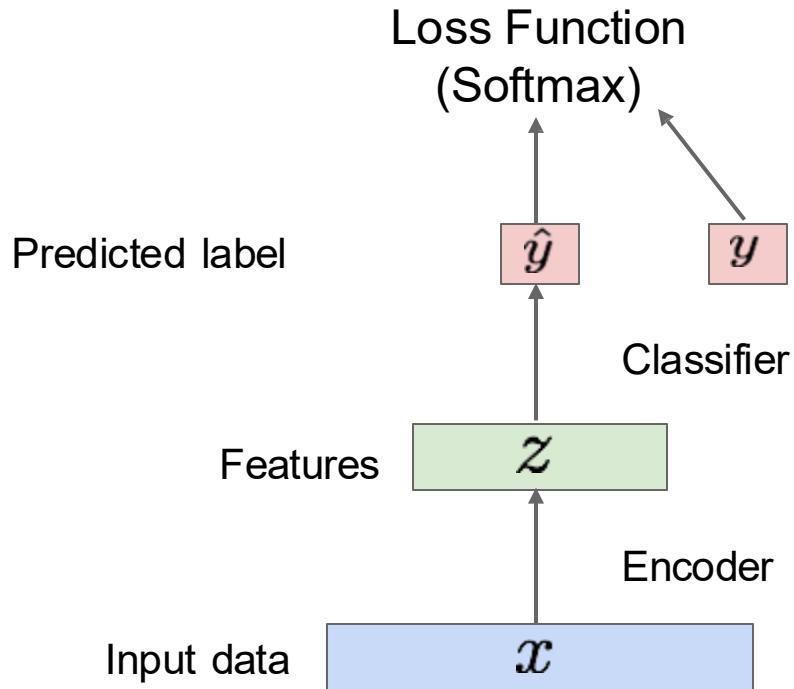
(Non-Variational) Autoencoders

Loss: L2 distance between input and reconstructed data.



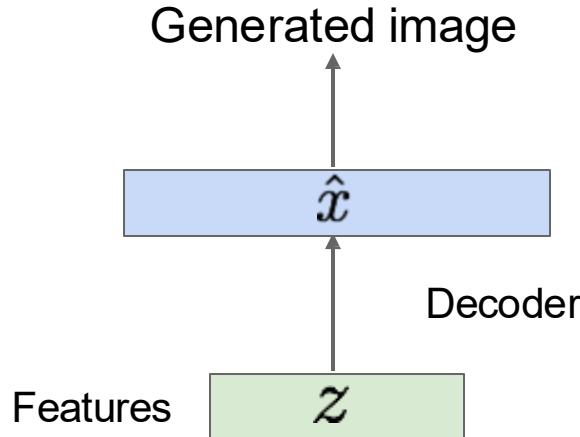
(Non-Variational) Autoencoders

After training, can use encoder for downstream tasks



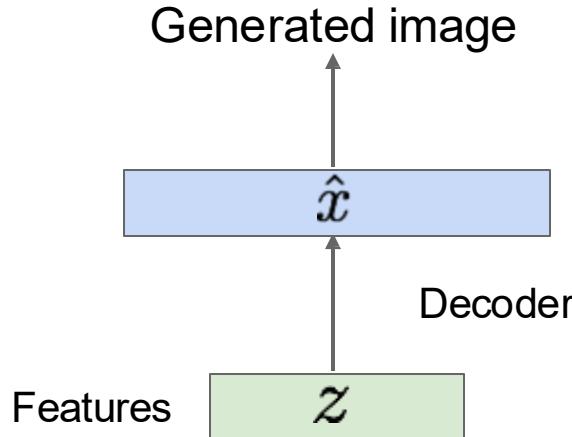
(Non-Variational) Autoencoders

If we could generate new z , could use the decoder to generate images



(Non-Variational) Autoencoders

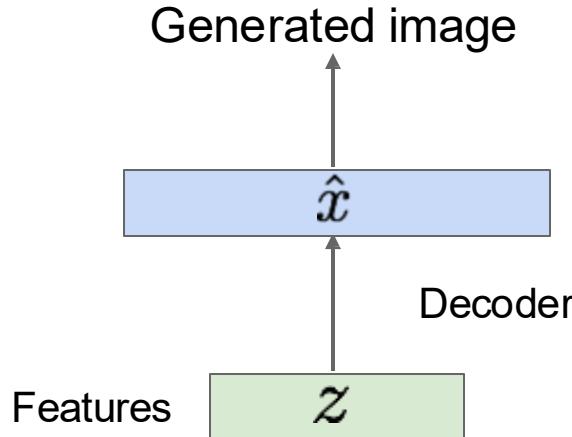
If we could generate new z , could use the decoder to generate images



Problem: Generating new z is not any easier than generating new x

(Non-Variational) Autoencoders

If we could generate new z , could use the decoder to generate images



Problem: Generating new z is not any easier than generating new x

Solution: What if we force all z to come from a known distribution?

Variational Autoencoders (VAEs)

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

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

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

Intuition: x is an image, z is latent factors used to generate x : attributes, orientation, etc.

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

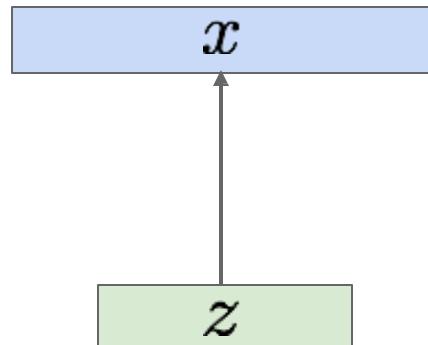
After training, sample new data like this:

Sample from
conditional

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

Sample z
from prior

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



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

Intuition: x is an image, z is latent factors used to generate x : attributes, orientation, etc.

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

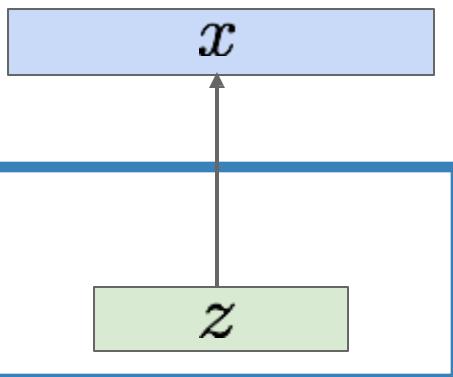
After training, sample new data like this:

Sample from
conditional

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

Sample z
from prior

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



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

Intuition: x is an image, z is latent factors used to generate x : attributes, orientation, etc.

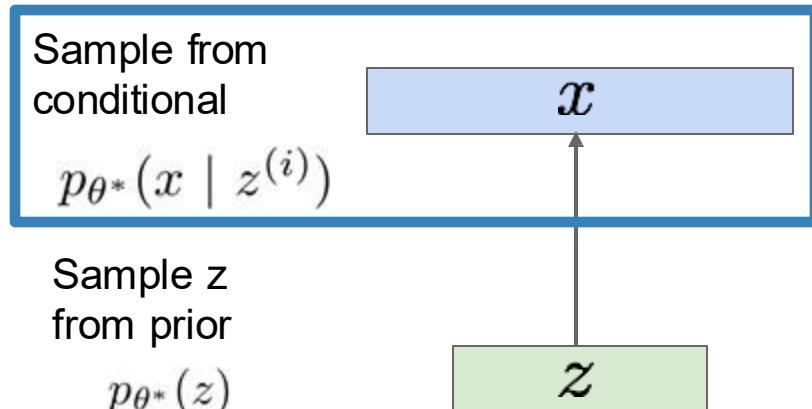
Assume simple prior $p(z)$, e.g. Gaussian

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



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

How can we train this?

Basic idea: **maximum likelihood**

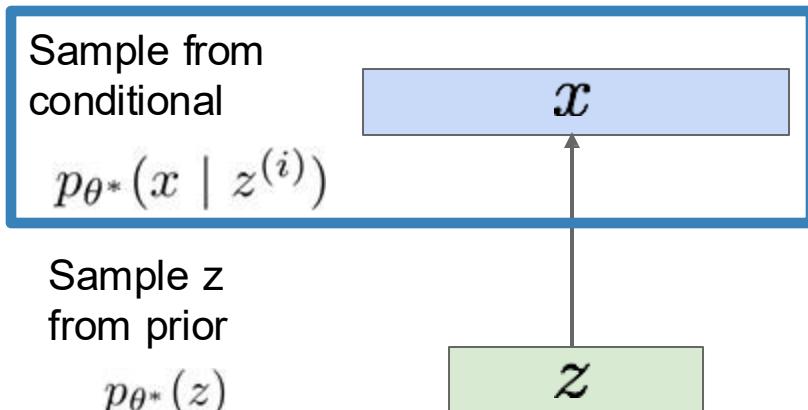
If we had a dataset of (x, z) then train a *conditional generative model* $p(x \mid z)$

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



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

How can we train this?

Basic idea: **maximum likelihood**

We don't observe z , so **marginalize**:

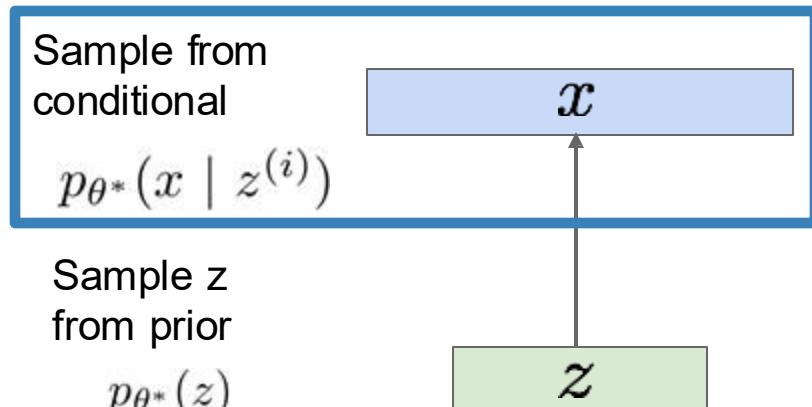
$$p_{\theta}(x) = \int p_{\theta}(x, z) dz = \int p_{\theta}(x \mid z) p_{\theta}(z) dz$$

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



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

How can we train this?

Basic idea: **maximum likelihood**

We don't observe z , so **marginalize**:

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

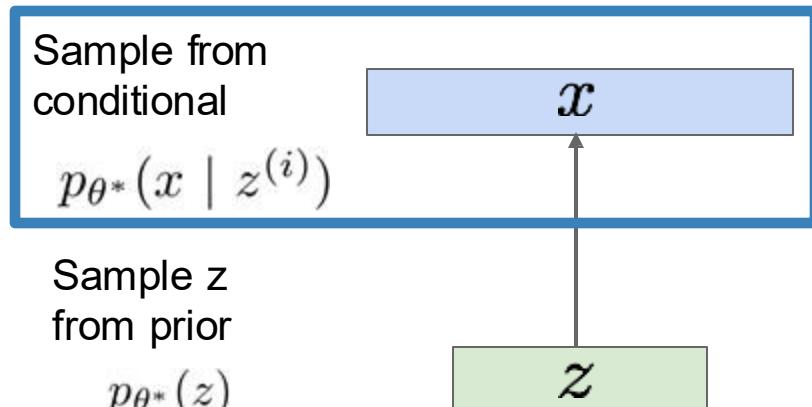
Ok, we can compute this with the decoder

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



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

How can we train this?

Basic idea: **maximum likelihood**

We don't observe z , so **marginalize**:

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

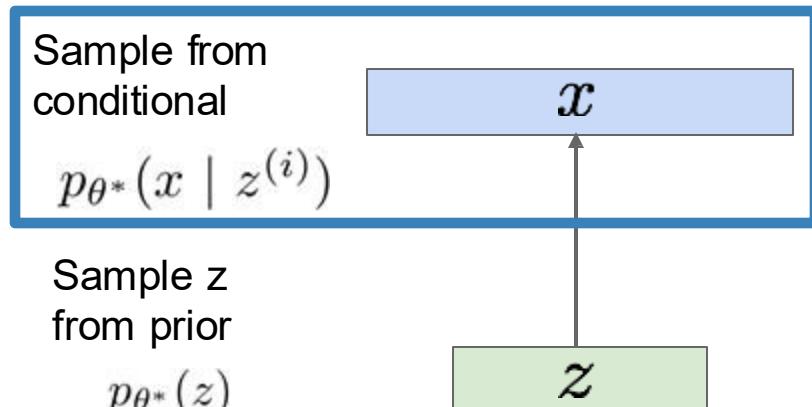
Ok, we assumed Gaussian prior for z

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



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

How can we train this?

Basic idea: **maximum likelihood**

We don't observe z , so **marginalize**:

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

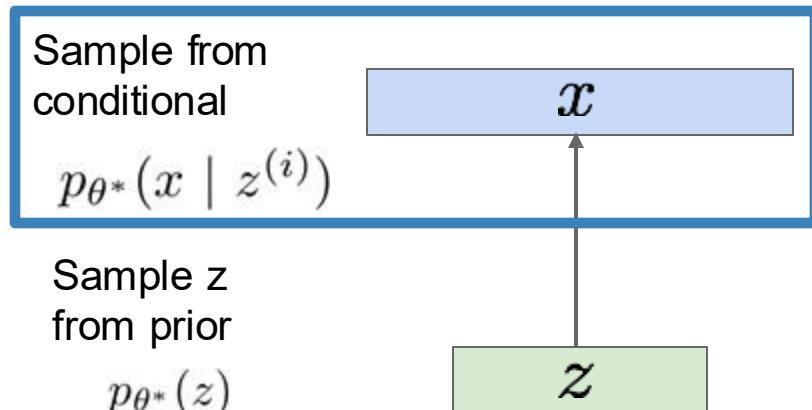
Problem, we can't integrate over all z

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



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

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

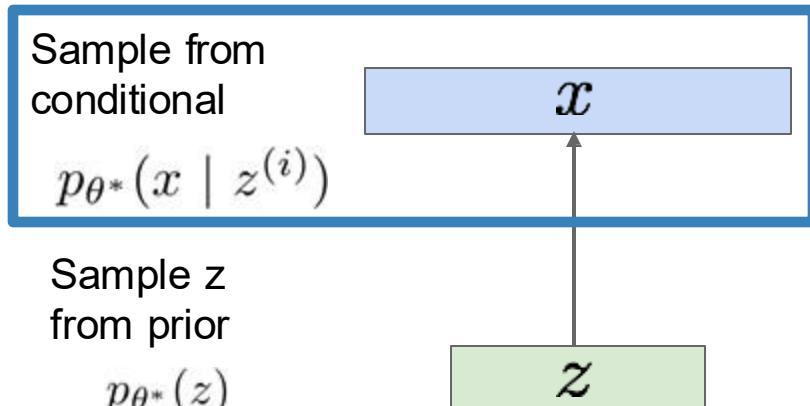
$$p_{\theta}(x) = \frac{p_{\theta}(x \mid z)p_{\theta}(z)}{p_{\theta}(z \mid x)}$$

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



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

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

$$p_{\theta}(x) = \frac{p_{\theta}(x \mid z)p_{\theta}(z)}{p_{\theta}(z \mid x)}$$

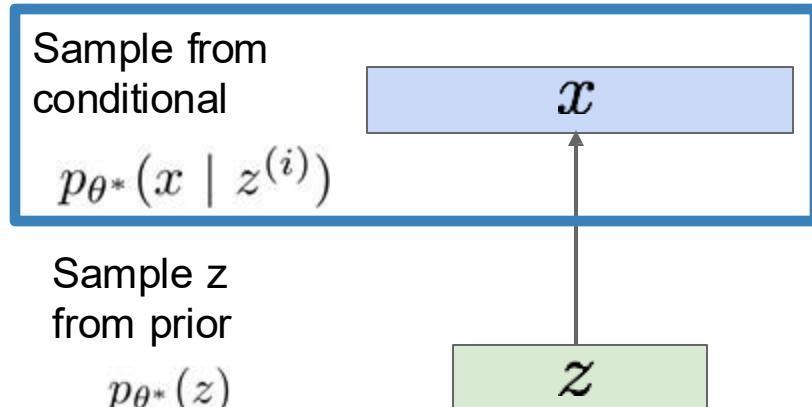
Ok, we can compute this with the decoder

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



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

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

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

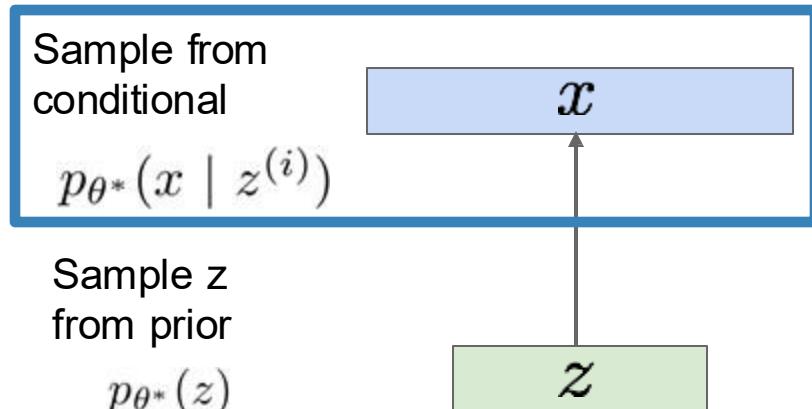
Ok, we assumed Gaussian prior for z

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



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

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

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

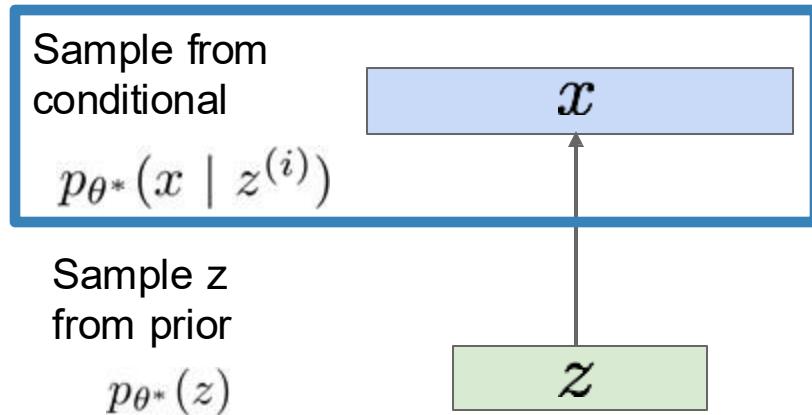
Problem: no way to compute this

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



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

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

$$p_{\theta}(x) = \frac{p_{\theta}(x \mid z)p_{\theta}(z)}{p_{\theta}(z \mid x)}$$

Problem: no way to compute this

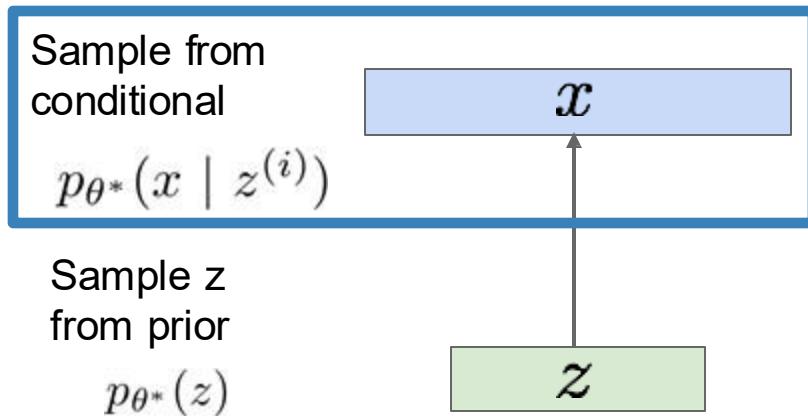
Solution: Train another network that learns $q_{\phi}(z \mid x) \approx p_{\theta}(z \mid x)$

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



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

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

$$p_{\theta}(x) = \frac{p_{\theta}(x \mid z)p_{\theta}(z)}{p_{\theta}(z \mid x)} \approx \frac{p_{\theta}(x \mid z)p_{\theta}(z)}{q_{\phi}(z \mid x)}$$

Solution: Train another network that learns $q_{\phi}(z \mid x) \approx p_{\theta}(z \mid x)$

Variational Autoencoders

Decoder Network:

Input latent code z ,

Output distribution over data x

Encoder Network:

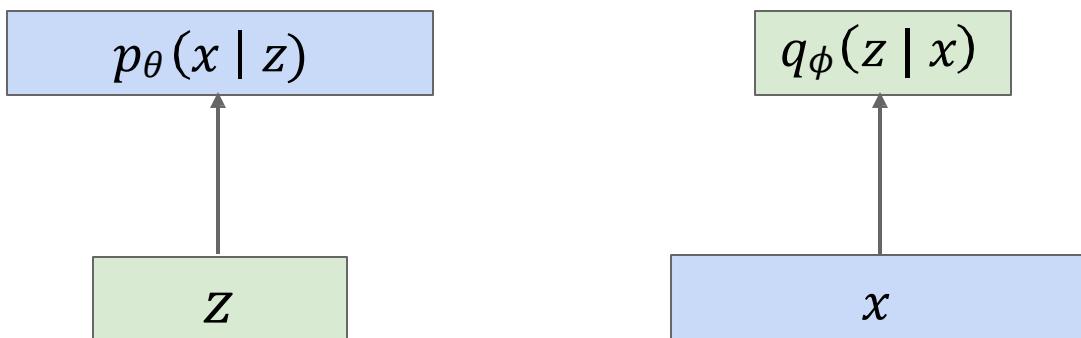
Input data x ,

Output distribution
over latent codes z

If we can ensure that
 $q_\phi(z | x) \approx p_\theta(z | x)$,

then we can approximate

$$p_\theta(x) \approx \frac{p_\theta(x | z)p(z)}{q_\phi(z | x)}$$



Idea: Jointly train both
encoder and decoder

Variational Autoencoders

Decoder Network:

Input latent code z ,

Output distribution over data x

Encoder Network:

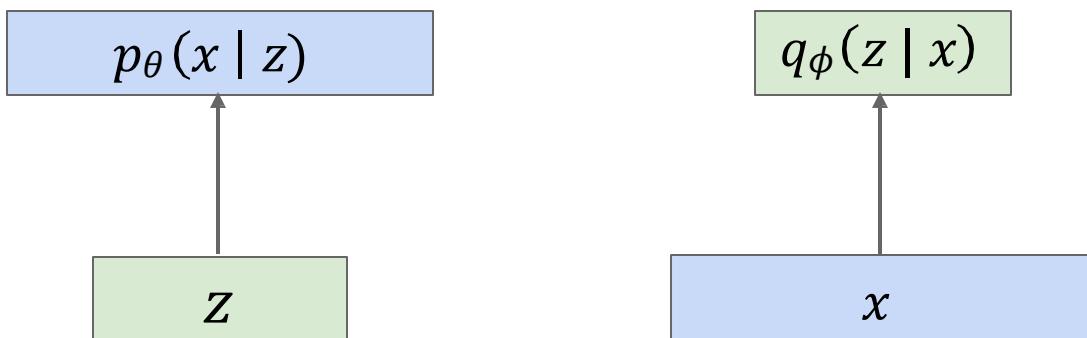
Input data x ,

Output distribution
over latent codes z

If we can ensure that
 $q_\phi(z | x) \approx p_\theta(z | x)$,

then we can approximate

$$p_\theta(x) \approx \frac{p_\theta(x | z)p(z)}{q_\phi(z | x)}$$



Idea: Jointly train both
encoder and decoder

Aside: How to output probability
distributions from neural networks?

Network outputs mean (and std) of
a (diagonal) distribution

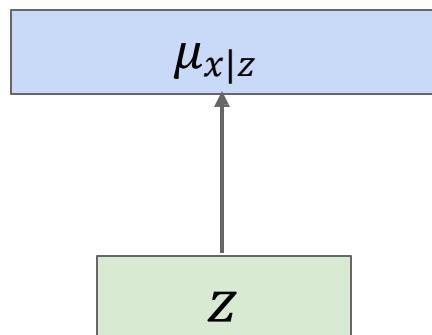
Variational Autoencoders

Decoder Network:

Input latent code z ,

Output distribution over data x

$$p_\theta(x | z) = N(\mu_{x|z}, \sigma^2)$$

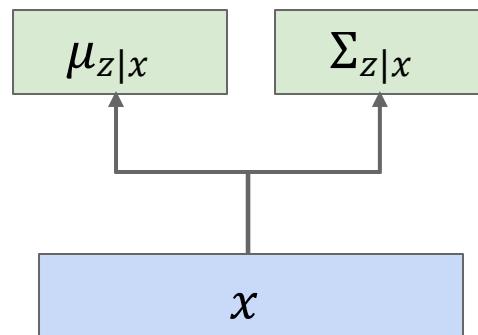


Encoder Network:

Input data x ,

Output distribution over latent codes z

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



If we can ensure that
 $q_\phi(z | x) \approx p_\theta(z | x)$,

then we can approximate

$$p_\theta(x) \approx \frac{p_\theta(x | z)p(z)}{q_\phi(z | x)}$$

Idea: Jointly train both encoder and decoder

Aside: How to output probability distributions from neural networks?

Network outputs mean (and std) of a (diagonal) distribution

Variational Autoencoders

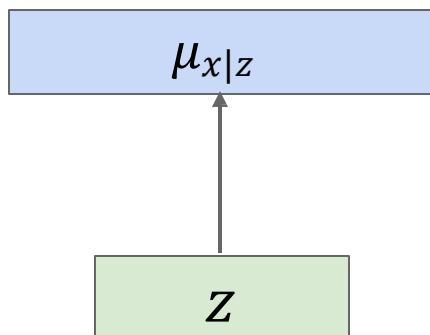
Decoder Network:

Input latent code z ,

Output distribution over data x

$$p_\theta(x | z) = N(\mu_{x|z}, \sigma^2)$$

$$\log p_\theta(x | z) = -\frac{1}{2\sigma^2} \|x - \mu\|_2^2 + C_2$$

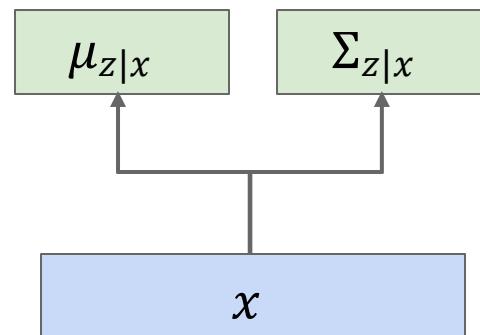


Encoder Network:

Input data x ,

Output distribution over latent codes z

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



If we can ensure that $q_\phi(z | x) \approx p_\theta(z | x)$,

then we can approximate

$$p_\theta(x) \approx \frac{p_\theta(x | z)p(z)}{q_\phi(z | x)}$$

Idea: Jointly train both encoder and decoder

Maximizing $\log p_\theta(x | z)$ is equivalent to minimizing L2 distance between x and network output!

Variational Autoencoders

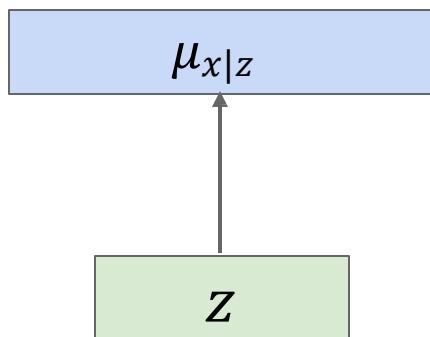
Decoder Network:

Input latent code z ,

Output distribution over data x

$$p_\theta(x | z) = N(\mu_{x|z}, \sigma^2)$$

$$\log p_\theta(x | z) = -\frac{1}{2\sigma^2} \|x - \mu\|_2^2 + C_2$$

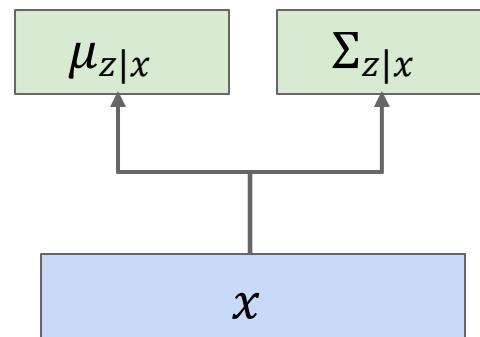


Encoder Network:

Input data x ,

Output distribution over latent codes z

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



If we can ensure that $q_\phi(z | x) \approx p_\theta(z | x)$,

then we can approximate

$$p_\theta(x) \approx \frac{p_\theta(x | z)p(z)}{q_\phi(z | x)}$$

Idea: Jointly train both encoder and decoder

Q: What's our training objective?

Variational Autoencoders (ELBO)

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

Bayes' Rule

Variational Autoencoders (ELBO)

$$\log p_\theta(x) = \log \frac{p_\theta(x | z)p(z)}{p_\theta(z | x)} = \log \frac{p_\theta(x|z)p(z)}{p_\theta(z|x)q_\phi(z|x)}$$

Multiply top and bottom by $q_\phi(z|x)$

Variational Autoencoders (ELBO)

$$\log p_\theta(x) = \log \frac{p_\theta(x | z)p(z)}{p_\theta(z | x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

$$= \log p_\theta(x|z) - \log \frac{q_\phi(z|x)}{p(z)} + \log \frac{q_\phi(z|x)}{p_\theta(z|x)}$$

Logarithms + rearranging

Variational Autoencoders (ELBO)

$$\log p_\theta(x) = \log \frac{p_\theta(x | z)p(z)}{p_\theta(z | x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

$$= \log p_\theta(x|z) - \log \frac{q_\phi(z|x)}{p(z)} + \log \frac{q_\phi(z|x)}{p_\theta(z|x)}$$

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

We can wrap in an expectation since it doesn't depend on z

Variational Autoencoders (ELBO)

$$\log p_\theta(x) = \log \frac{p_\theta(x | z)p(z)}{p_\theta(z | x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

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

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

We can wrap in an expectation since it doesn't depend on z

Variational Autoencoders (ELBO)

$$\log p_\theta(x) = \log \frac{p_\theta(x | z)p(z)}{p_\theta(z | x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

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

$$= E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z)) + D_{KL} (q_\phi(z|x), p_\theta(z|x))$$

Variational Autoencoders (ELBO)

$$\log p_\theta(x) = \log \frac{p_\theta(x | z)p(z)}{p_\theta(z | x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

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

$$= E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z)) + D_{KL} (q_\phi(z|x), p_\theta(z|x))$$

Data reconstruction: $x \Rightarrow$ encoder \Rightarrow decoder should reconstruct x

Can compute in closed form for Gaussians.

Variational Autoencoders (ELBO)

$$\log p_\theta(x) = \log \frac{p_\theta(x | z)p(z)}{p_\theta(z | x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

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

$$= E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z)) + D_{KL} (q_\phi(z|x), p_\theta(z|x))$$

Prior: Encoder output should match prior over z .

Can compute in closed form for Gaussians.

Variational Autoencoders (ELBO)

$$\log p_\theta(x) = \log \frac{p_\theta(x|z)p(z)}{p_\theta(z|x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

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

$$= E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z)) + D_{KL} (q_\phi(z|x), p_\theta(z|x))$$

Posterior Approximation: Encoder output $q_\phi(z|x)$ should match $p_\theta(z|x)$

We cannot compute this for Gaussians...

Variational Autoencoders (ELBO)

$$\log p_\theta(x) = \log \frac{p_\theta(x|z)p(z)}{p_\theta(z|x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

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

$$= E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z)) + D_{KL} (q_\phi(z|x), p_\theta(z|x))$$

Posterior Approximation: Decoder output $q_\phi(z|x)$ should match $p_\theta(z|x)$

KL is ≥ 0 , so we can drop it to get a lower bound on likelihood

Variational Autoencoders (ELBO)

$$\log p_\theta(x) = \log \frac{p_\theta(x|z)p(z)}{p_\theta(z|x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

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

$$= E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z)) + D_{KL} (q_\phi(z|x), p_\theta(z|x))$$

$$\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z))$$

This is our VAE training objective

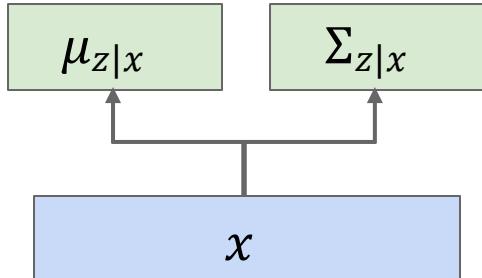
Variational Autoencoders

Jointly train **encoder** q and **decoder** p to maximize the **variational lower bound** on the data likelihood
Also called **Evidence Lower Bound (ELBo)**

$$\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$

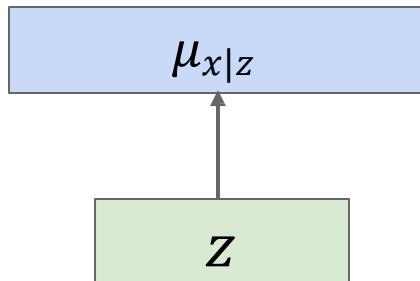
Encoder Network

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



Decoder Network

$$p_\theta(x | z) = N(\mu_{x|z}, \sigma^2)$$



Variational Autoencoders: Training

Train by maximizing the
variational lower bound

$$E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z))$$

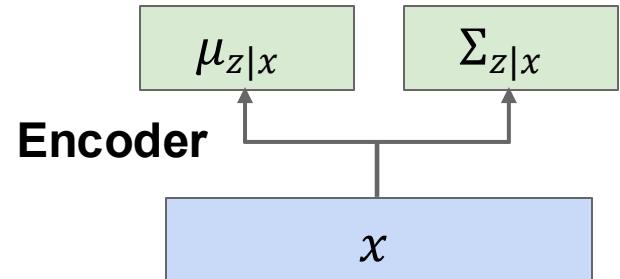
Variational Autoencoders: Training

Train by maximizing the
variational lower bound

$$E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z))$$

1. Run input data through **encoder** to get distribution over z

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



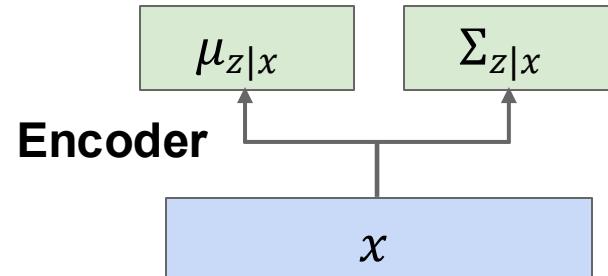
Variational Autoencoders: Training

Train by maximizing the
variational lower bound

$$E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z))$$

1. Run input data through **encoder** to get distribution over z
2. Prior loss: Encoder output should be unit Gaussian (zero mean, unit variance)

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

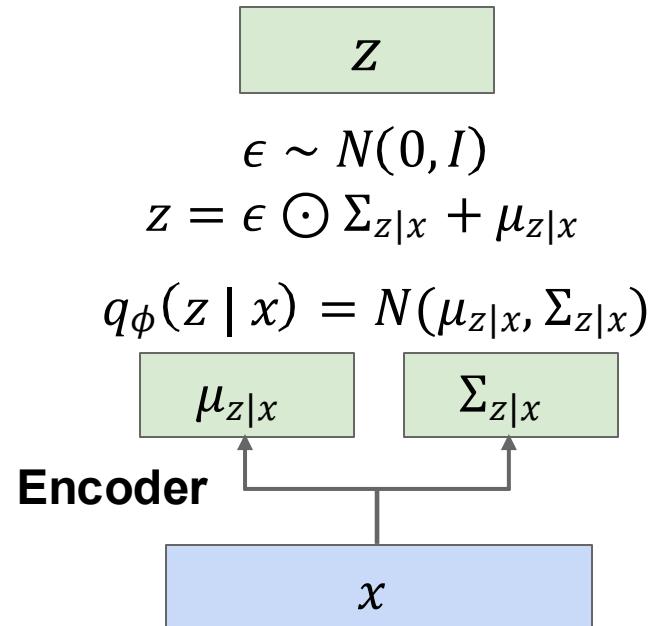


Variational Autoencoders: Training

Train by maximizing the
variational lower bound

$$E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z))$$

1. Run input data through **encoder** to get distribution over z
2. Prior loss: Encoder output should be unit Gaussian (zero mean, unit variance)
3. Sample z from encoder output $q_\phi(z | x)$
(Reparameterization trick)

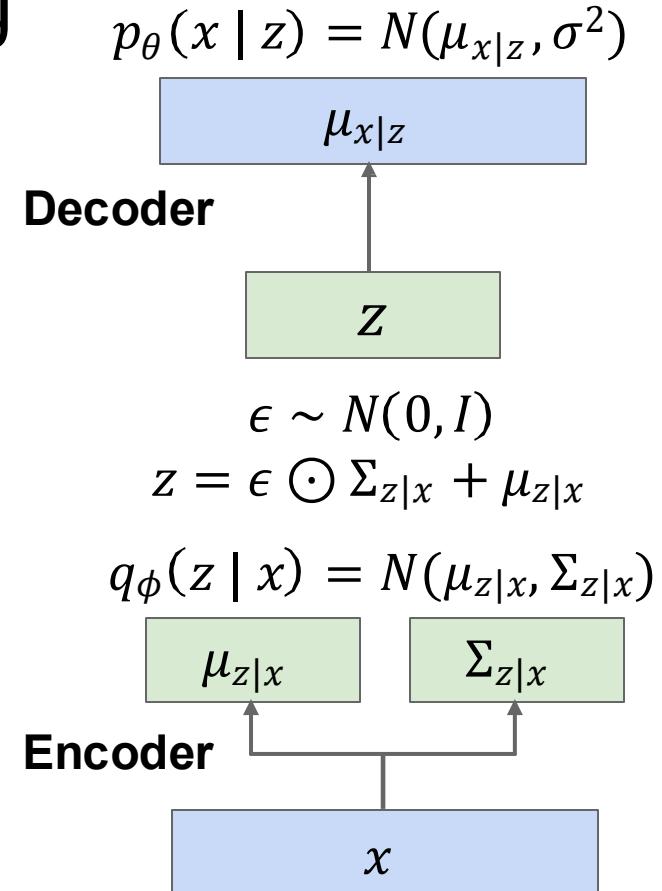


Variational Autoencoders: Training

Train by maximizing the
variational lower bound

$$E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL} (q_\phi(z|x), p(z))$$

1. Run input data through **encoder** to get distribution over z
2. Prior loss: Encoder output should be unit Gaussian (zero mean, unit variance)
3. Sample z from encoder output $q_\phi(z|x)$ (Reparameterization trick)
4. Run z through **decoder** to get predicted data mean

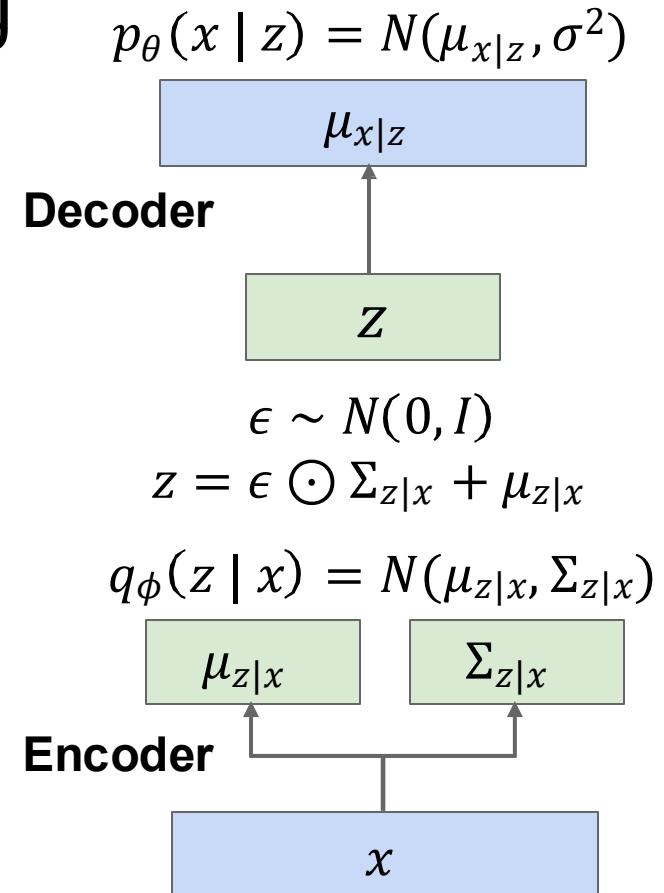


Variational Autoencoders: Training

Train by maximizing the
variational lower bound

$$E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$

1. Run input data through **encoder** to get distribution over z
2. Prior loss: Encoder output should be unit Gaussian (zero mean, unit variance)
3. Sample z from encoder output $q_\phi(z|x)$ (Reparameterization trick)
4. Run z through **decoder** to get predicted data mean
5. Reconstruction loss: predicted mean should match x in L2



Variational Autoencoders: Training

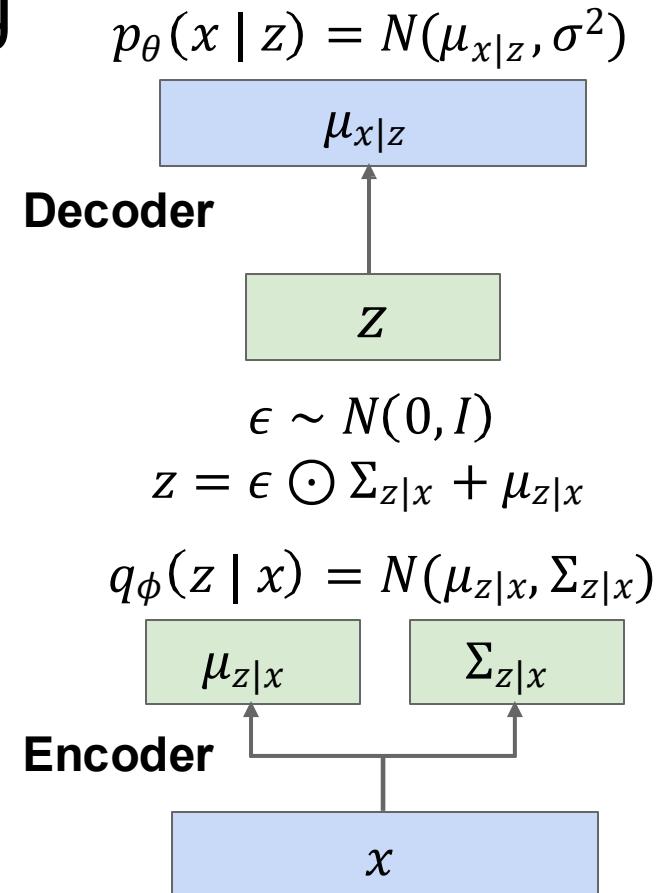
Train by maximizing the
variational lower bound

$$E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$

The loss terms fight against each other!

Reconstruction loss wants $\Sigma_{z|x} = 0$ and $\mu_{z|x}$ to be unique for each x , so decoder can deterministically reconstruct x

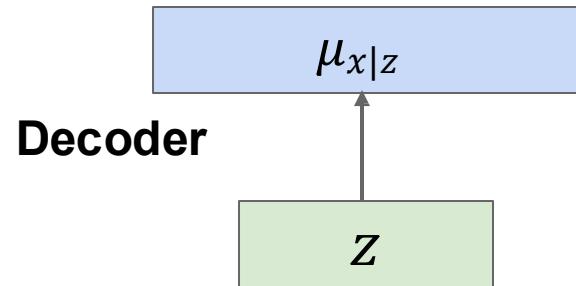
Prior loss wants $\Sigma_{z|x} = \mathbf{I}$ and $\mu_{z|x} = 0$ so encoder output is always a unit Gaussian



Variational Autoencoders: Sampling

1. Sample z from the prior
2. Run through decoder to get an image

$$p_{\theta}(x | z) = N(\mu_{x|z}, \sigma^2)$$



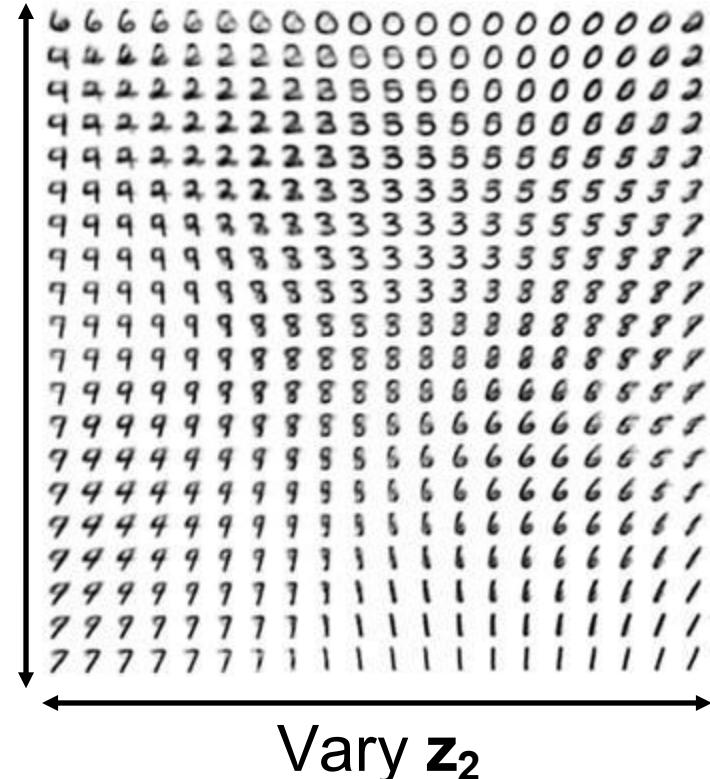
$$z \sim N(0, I)$$

Variational Autoencoders: Disentangling

The diagonal prior on $p(z)$ causes dimensions of z to be independent

“Disentangling factors of variation”

Vary z_1



Recap: Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a function to map $x \rightarrow y$

Examples: Classification, regression,
object detection, semantic
segmentation, image captioning, etc.

Unsupervised Learning

Data: x

Just data, no labels!

Goal: Learn hidden structure in data

Examples: Clustering, dimensionality
reduction, density estimation, etc.

Recap: Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$

Generative Model:

Learn a probability distribution $p(x)$

Conditional Generative Model:

Learn $p(x|y)$

Data: x



Label: y

Cat

Density Function

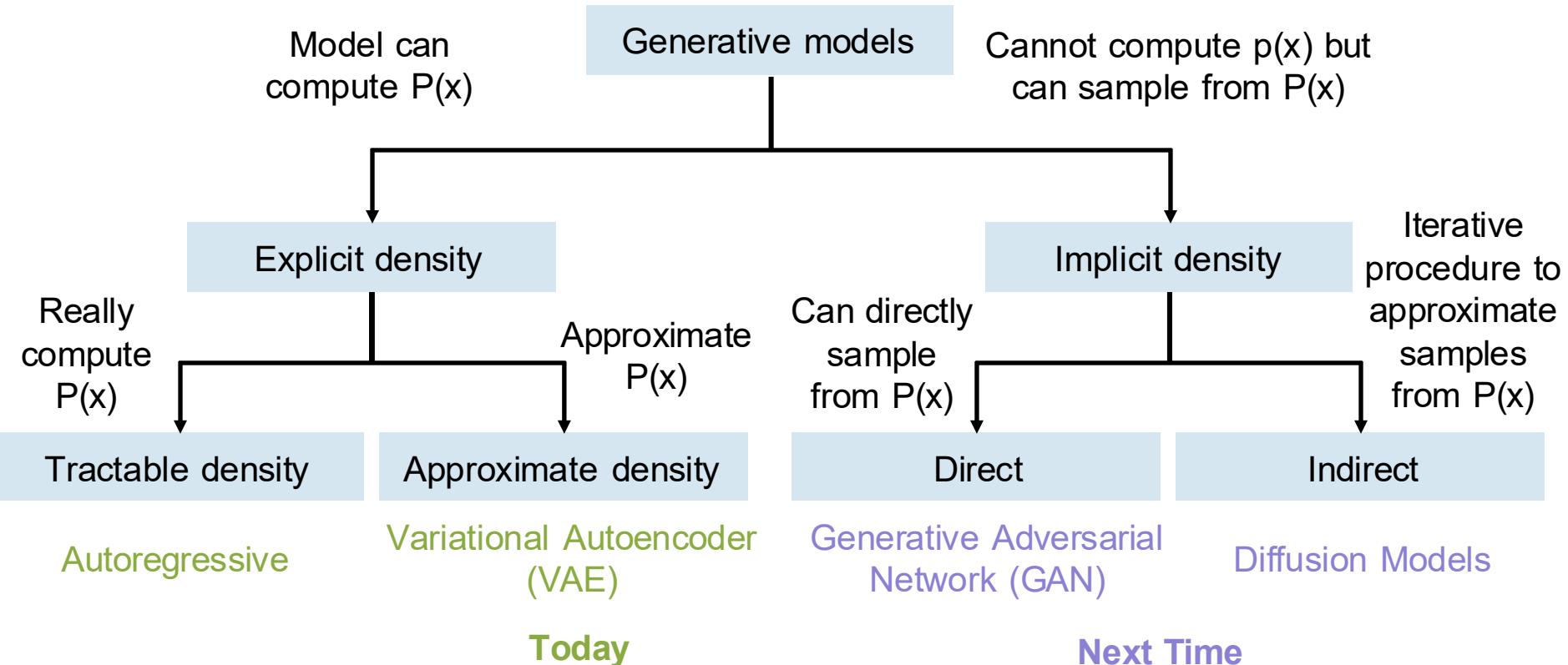
$p(x)$ assigns a positive number to each possible x ; higher numbers mean x is more likely.

Density functions are **normalized**:

$$\underset{X}{\mathbb{E}} p(x) dx = 1$$

Different values of x compete for density

Recap: Generative Models



Next Time:
Generative Models (part 2)
Generative Adversarial Networks
Diffusion Models

Lecture 14: Generative Models (part 2)

Last Time: Generative vs Discriminative Models

Discriminative Model:

Learn a probability distribution $p(y|x)$

Generative Model:

Learn a probability distribution $p(x)$

Conditional Generative Model: Learn $p(x|y)$

Data: x



Label: y

Cat

Density Function

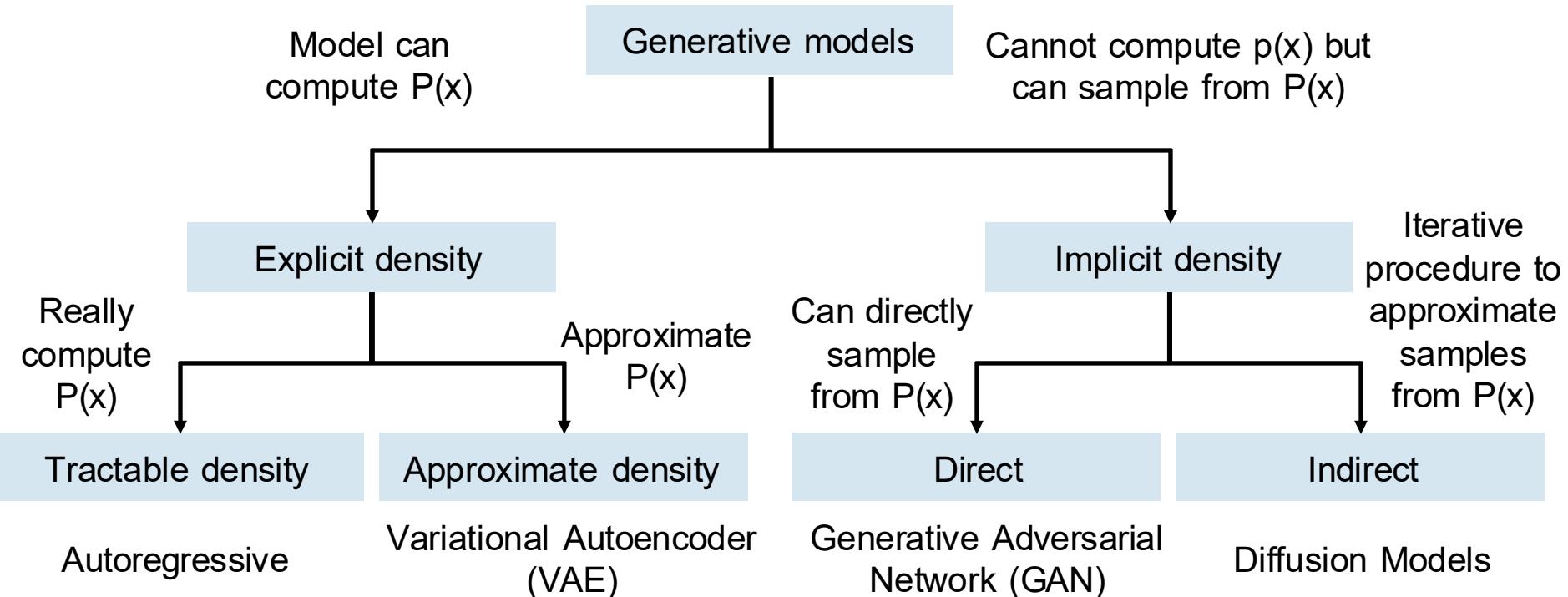
$p(x)$ assigns a positive number to each possible x ; higher numbers mean x is more likely.

Density functions are **normalized**:

$$\underset{X}{\oint} p(x)dx = 1$$

Different values of x **compete** for density

Last Time: Generative Models

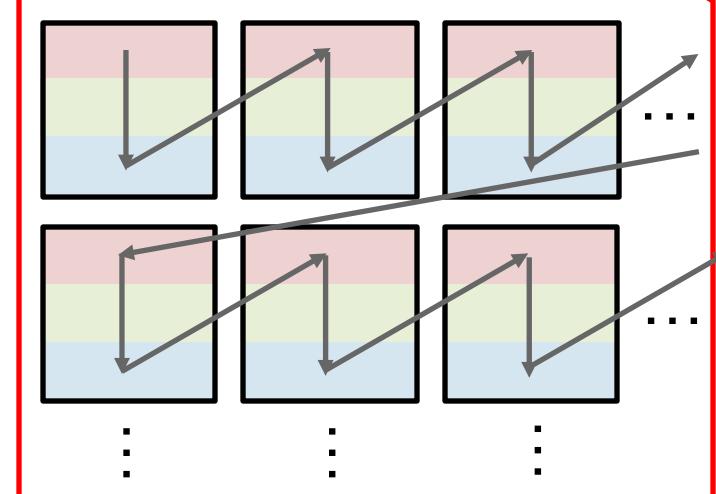
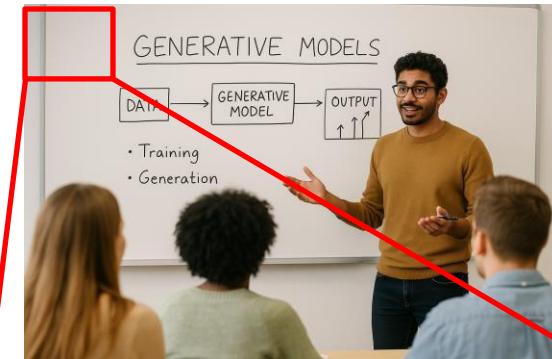


Last Time: Autoregressive Models

Treat data as a sequence
(e.g. image as sequence of pixels)

$$\begin{aligned} p(x) &= p(x_1, x_2, \dots, x_N) \\ &= p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \dots \\ &= \zeta_{t=1}^T p(x_t | x_1, \dots, x_{t-1}) \end{aligned}$$

Model with an RNN or Transformer



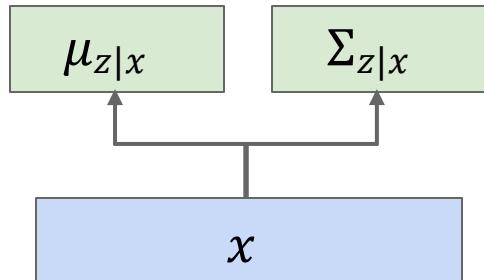
Last Time: Variational Autoencoders

Jointly train **encoder** q and **decoder** p to maximize
the **variational lower bound** on the data likelihood
Also called **Evidence Lower Bound (ELBo)**

$$\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$

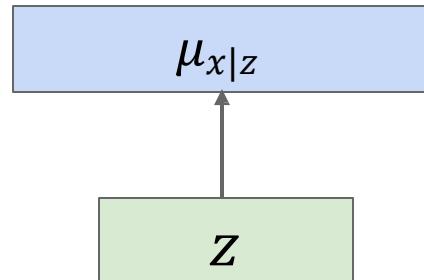
Encoder Network

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

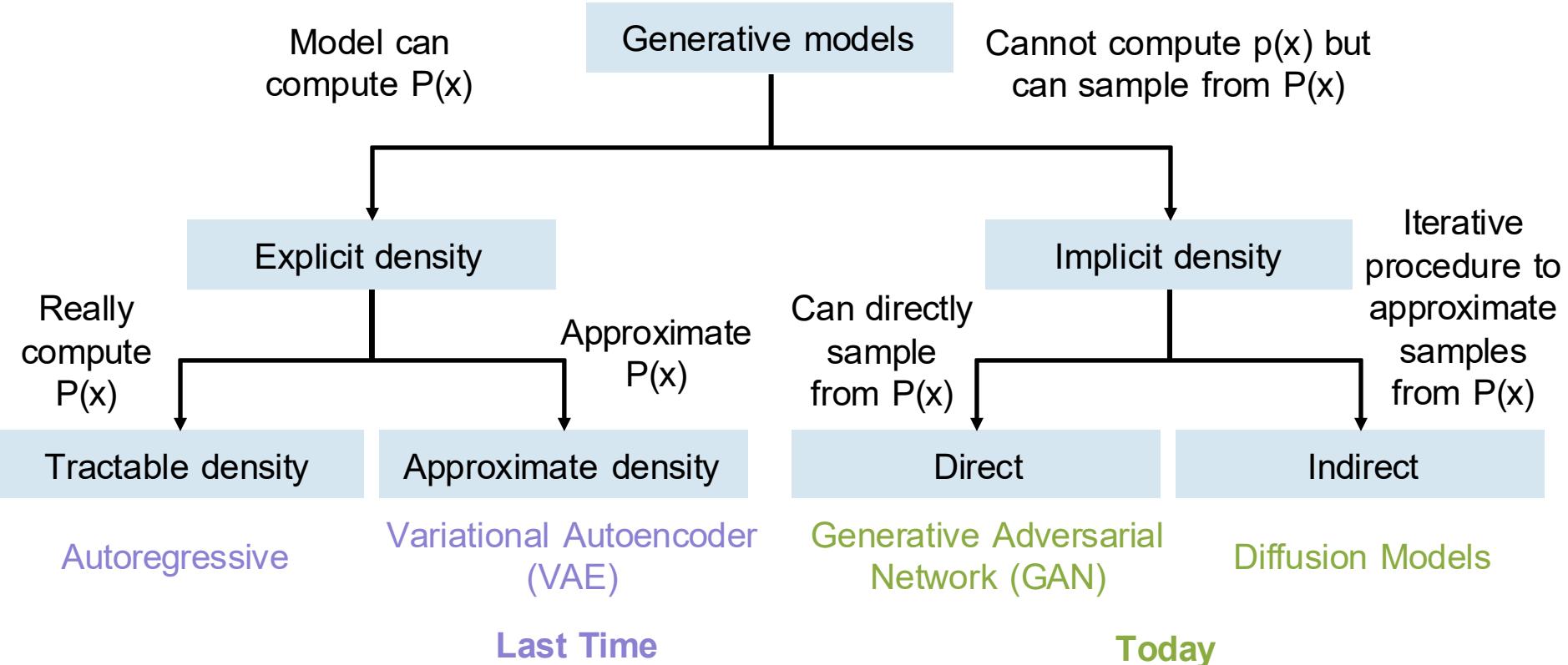


Decoder Network

$$p_\theta(x | z) = N(\mu_{x|z}, \sigma^2)$$



Today: More Generative Models



Generative Adversarial Networks (GANs)

Generative Models So Far

Autoregressive Models directly maximize likelihood of training data:

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

Variational Autoencoders introduce a latent z , and maximize a lower bound:

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

Generative Models So Far

Autoregressive Models directly maximize likelihood of training data:

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

Variational Autoencoders introduce a latent z , and maximize a lower bound:

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

Generative Adversarial Networks give up on modeling $p(x)$, but allow us to draw samples from $p(x)$

Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{\text{data}}(x)$. Want to sample from p_{data}

Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{\text{data}}(x)$. Want to sample from p_{data}

Idea: Introduce a latent variable z with simple prior $p(z)$ (e.g. unit Gaussian)

Sample $z \sim p(z)$ and pass to a **Generator Network** $x = G(z)$

Then x is a sample from the **Generator distribution** p_G . Want $p_G = p_{\text{data}}$!

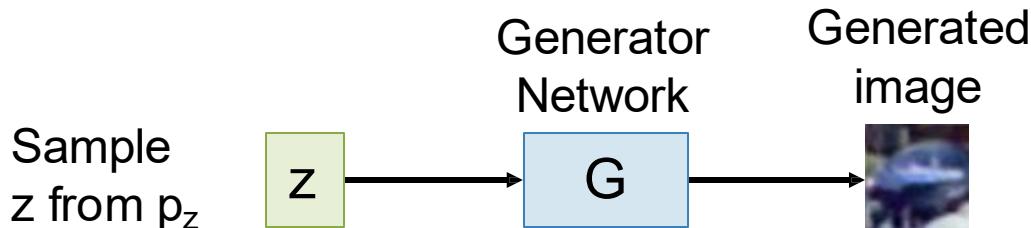
Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{\text{data}}(x)$. Want to sample from p_{data}

Idea: Introduce a latent variable z with simple prior $p(z)$ (e.g. unit Gaussian)

Sample $z \sim p(z)$ and pass to a **Generator Network** $x = G(z)$

Then x is a sample from the **Generator distribution** p_G . Want $p_G = p_{\text{data}}$!



Train **Generator Network** G to convert
 z into fake data x sampled from p_G

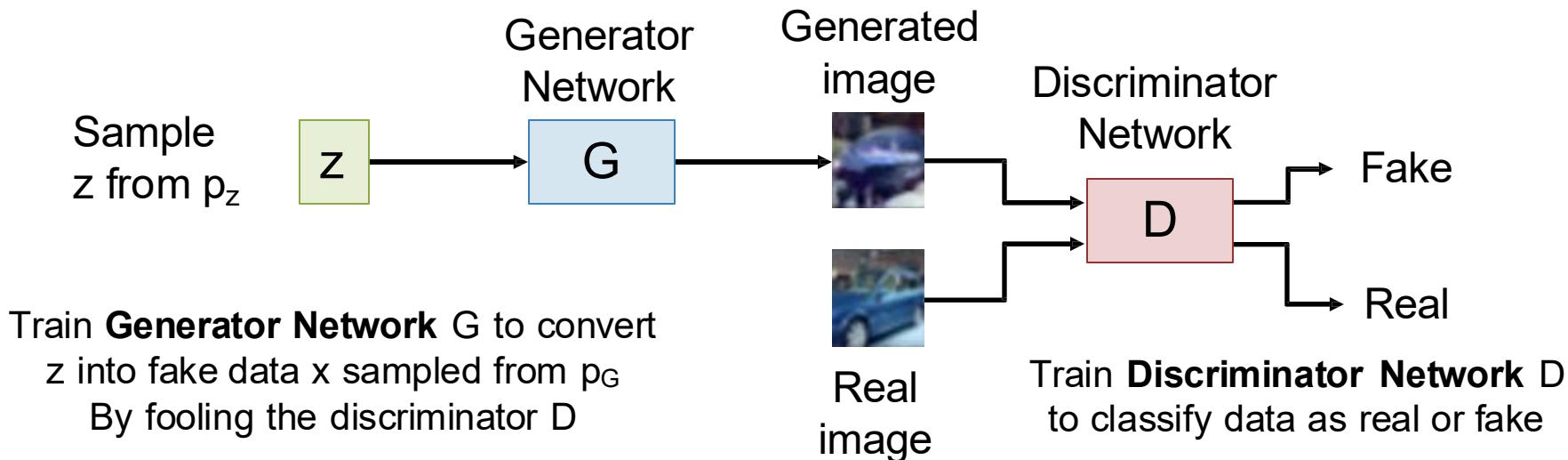
Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{\text{data}}(x)$. Want to sample from p_{data}

Idea: Introduce a latent variable z with simple prior $p(z)$ (e.g. unit Gaussian)

Sample $z \sim p(z)$ and pass to a **Generator Network** $x = G(z)$

Then x is a sample from the **Generator distribution** p_G . Want $p_G = p_{\text{data}}$!



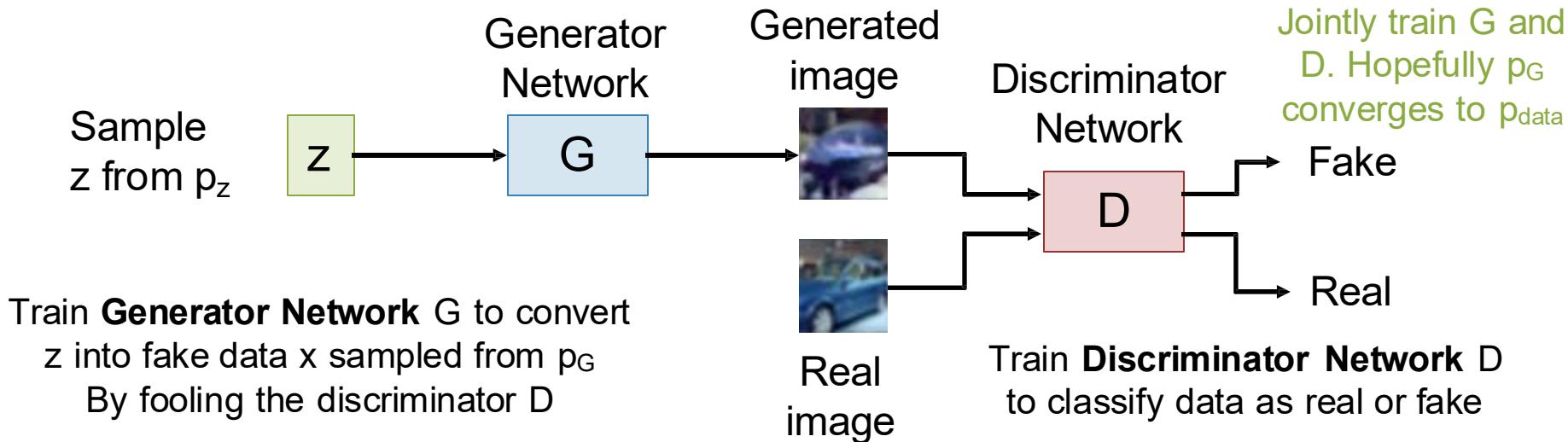
Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{\text{data}}(x)$. Want to sample from p_{data}

Idea: Introduce a latent variable z with simple prior $p(z)$ (e.g. unit Gaussian)

Sample $z \sim p(z)$ and pass to a **Generator Network** $x = G(z)$

Then x is a sample from the **Generator distribution** p_G . Want $p_G = p_{\text{data}}$!



Generative Adversarial Networks: Training Objective

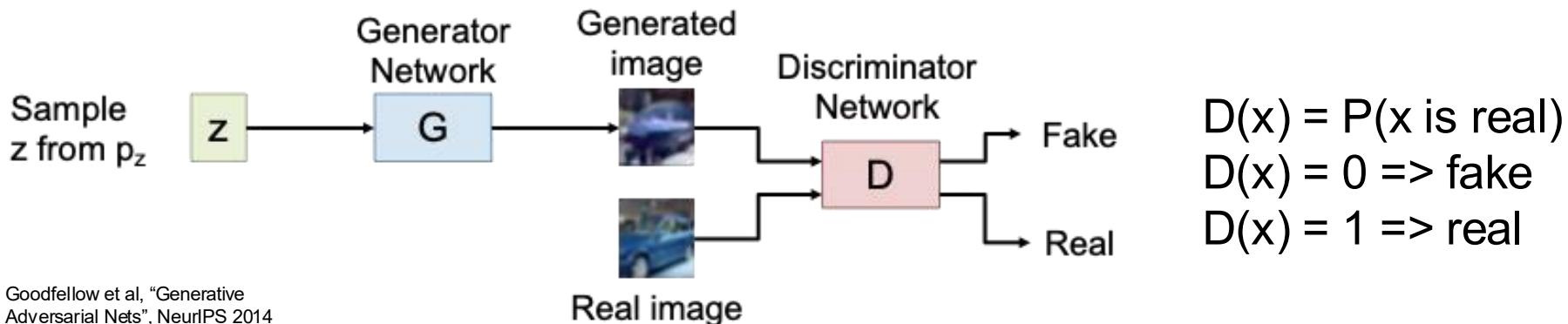
Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log (1 - D(G(z))) \right] \right)$$

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$



Generative Adversarial Networks: Training Objective

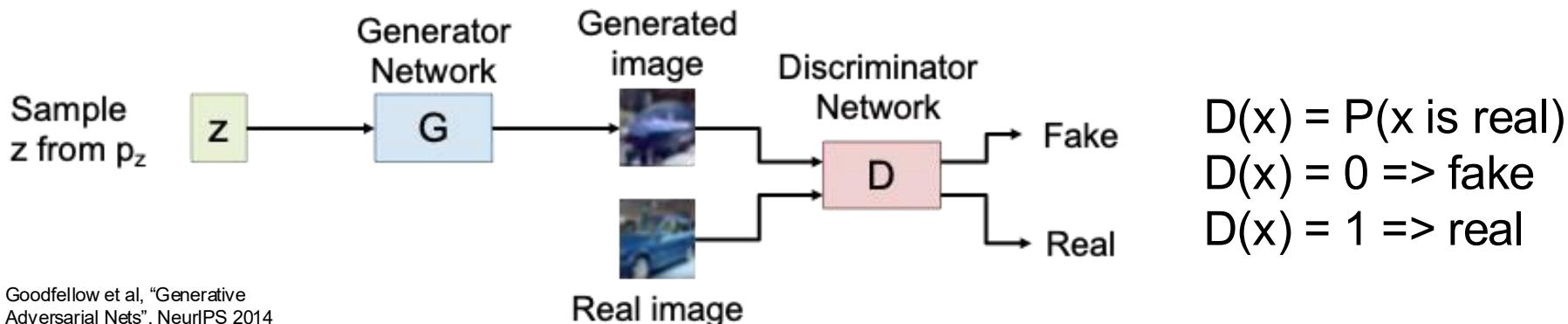
Jointly train generator G and discriminator D with a **minimax game**

Imagine fixing G

Discriminator wants $D(x) = 1$ for real data

Discriminator wants $D(x) = 0$ for fake data

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$



Generative Adversarial Networks: Training Objective

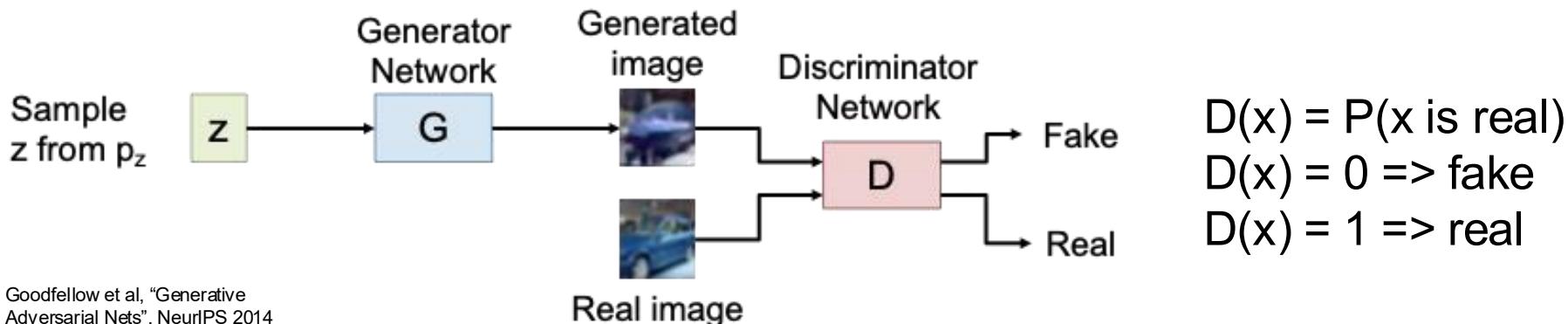
Jointly train generator G and discriminator D with a **minimax game**

Imagine fixing D

This term does not depend on G

Generator wants $D(x) = 1$ for fake data

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log \left(1 - D(G(z)) \right) \right] \right)$$



Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train **G** and **D** using alternating gradient updates

$$\min_{\textcolor{blue}{G}} \max_{\textcolor{red}{D}} \left(E_{x \sim p_{data}} [\log \textcolor{red}{D}(x)] + E_{z \sim p(z)} [\log (1 - \textcolor{red}{D}(\textcolor{blue}{G}(z)))] \right)$$

$$= \min_{\textcolor{blue}{G}} \max_{\textcolor{red}{D}} V(\textcolor{blue}{G}, \textcolor{red}{D})$$

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train **G** and **D** using alternating gradient updates

$$\min_{\textcolor{blue}{G}} \max_{\textcolor{red}{D}} \left(E_{x \sim p_{data}} [\log \textcolor{red}{D}(x)] + E_{z \sim p(z)} [\log (1 - \textcolor{red}{D}(\textcolor{blue}{G}(z)))] \right)$$

$$= \min_{\textcolor{blue}{G}} \max_{\textcolor{red}{D}} V(\textcolor{blue}{G}, \textcolor{red}{D})$$

While True:

$$D = D + \alpha_{\textcolor{red}{D}} \frac{dV}{dD}$$

$$G = G - \alpha_{\textcolor{blue}{G}} \frac{dV}{dG}$$

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train **G** and **D** using alternating gradient updates

$$\min_{\mathbf{G}} \max_{\mathbf{D}} \left(E_{x \sim p_{data}} [\log \mathbf{D}(x)] + E_{z \sim p(z)} [\log (1 - \mathbf{D}(\mathbf{G}(z)))] \right)$$

$$= \min_{\mathbf{G}} \max_{\mathbf{D}} V(\mathbf{G}, \mathbf{D})$$

We are not minimizing any overall loss! No training curves to look at!

While True:

$$\mathbf{D} = \mathbf{D} + \alpha_{\mathbf{D}} \frac{dV}{d\mathbf{D}}$$

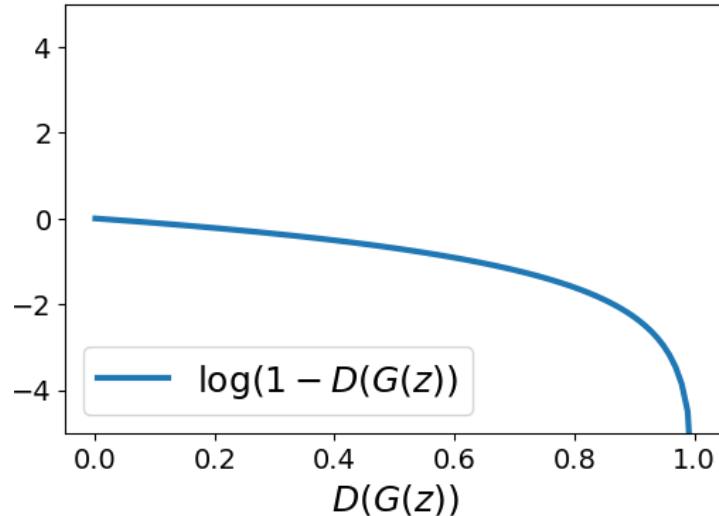
$$\mathbf{G} = \mathbf{G} - \alpha_{\mathbf{G}} \frac{dV}{d\mathbf{G}}$$

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so $D(G(z))$ close to 0



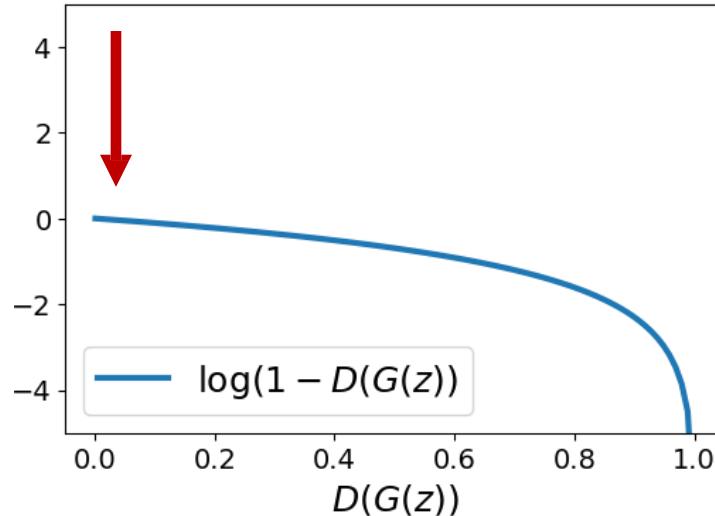
Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so $D(G(z))$ close to 0

Problem: Gradients for G are close to 0



Generative Adversarial Networks: Training Objective

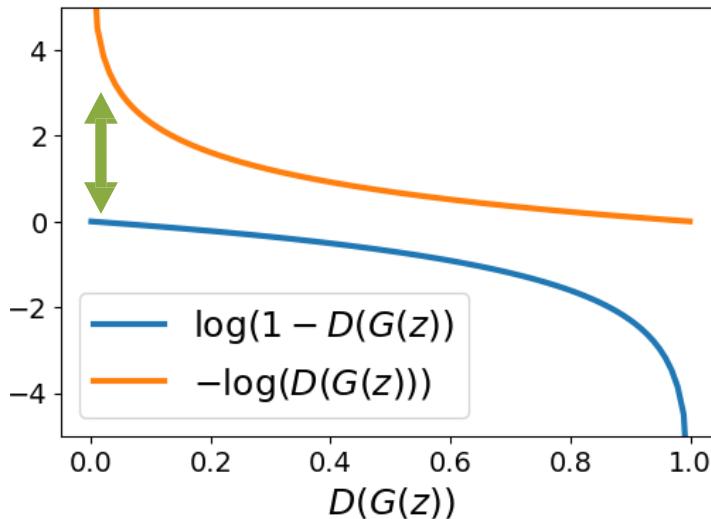
Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so $D(G(z))$ close to 0

Problem: Gradients for G are close to 0

Solution: Generator wants $D(G(z)) = 1$. Train generator to minimize $-\log(D(G(z)))$ and discriminator to maximize $\log(1-D(G(z)))$ so generator gets strong gradients at start



Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$

Why is this a good objective?

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$

Why is this a good objective?

Inner objective is maximized by

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

(for any p_G)

(Proof omitted)

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$

Why is this a good objective?

Inner objective is maximized by

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

(for any p_G)

Outer objective is
then minimized by
 $p_G(x) = p_{data}(x)$

(Proof omitted)

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$

Why is this a good objective?

Inner objective is maximized by

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

(for any p_G)

(Proof omitted)

Outer objective is
then minimized by
 $p_G(x) = p_{data}(x)$

Caveats:

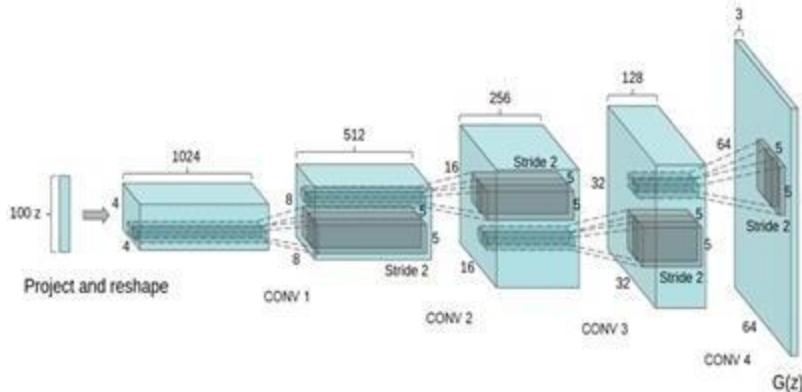
1. Neural nets with fixed capacity may not be able to represent optimal D and G
2. This tells us nothing about convergence to the solution with finite data

GAN Architectures: DC-GAN

Generator G and discriminator D
are both neural networks

Usually CNNs ... GANs fell out of
favor before ViT became popular

DC-GAN was the first GAN
architecture that worked on non-
toy data



Radford et al, ICLR 2016

GAN Architectures: DC-GAN

GPT-1 Paper (2018)

Improving Language Understanding
by Generative Pre-Training

Alec Radford
OpenAI

Karthik Narasimhan
OpenAI

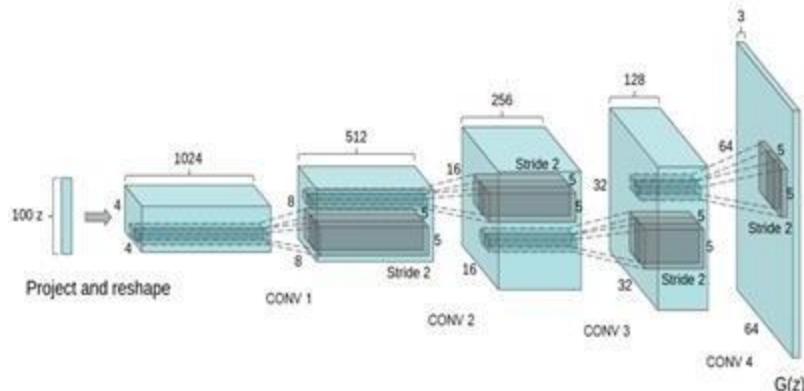
Tim Salimans
OpenAI

Ilya Sutskever
OpenAI

GPT-2 Paper (2019)

Language Models are Unsupervised Multitask Learners

Alec Radford ^{*†} Jeffrey Wu ^{*†} Rewon Child [†] David Luan [†] Dario Amodei ^{**†} Ilya Sutskever ^{**†}



Radford et al, ICLR 2016

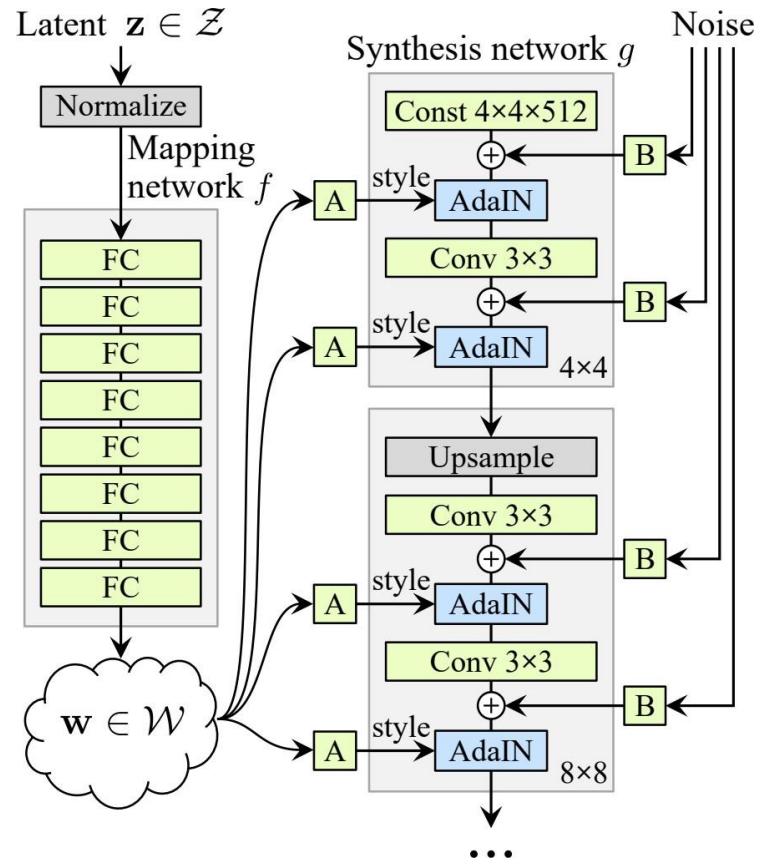
GAN Architectures: StyleGAN

Generator G and discriminator D are both neural networks

StyleGAN uses a more complex architecture that injects noise via **adaptive normalization**.

At each layer predict a scale w and shift b the same shape as x:

$$AdaIN(x, w, b)_i = w_i \frac{x_i - \mu(x)}{\sigma(x)} + b_i$$



Karras et al, "A Style-Based Generator Architecture for Generative Adversarial Networks", CVPR 2019

GANs: Latent Space Interpolation

Latent space is **smooth**.

Given latent vectors z_0 and z_1 , we can **interpolate** between them:

$$\begin{aligned} z_t &= tz_0 + (1 - t)z_1 \\ x_t &= G(z_t) \end{aligned}$$

The resulting image x_t smoothly interpolate between samples!

GANs: Latent Space Interpolation

Latent space is **smooth**.

Given latent vectors z_0 and z_1 , we can **interpolate** between them:

$$z_t = tz_0 + (1 - t)z_1$$
$$x_t = G(z_t)$$

The resulting image x_t smoothly interpolate between samples!



StyleGAN3 (Ours)

Generative Adversarial Networks: Summary

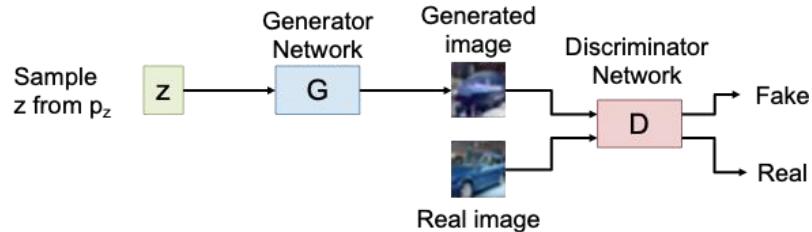
Jointly train Generator and Discriminator with a minimax game

Pros:

- Simple formulation
- Very good image quality

Cons:

- No loss curve to look at
- Unstable training
- Hard to scale to big models + data



These were the go-to generative models from ~2016 – 2021

Diffusion Models

Sohl-Dickstein et al, "Deep Unsupervised Learning using nonequilibrium thermodynamics", ICML 2015
Song and Ermon, "Generative modeling by estimating gradients of the data distribution", NeurIPS 2019
Ho et al, "Denoising Diffusion Probabalistic Models", NeurIPS 2020
Song et al, "Score-Based Generative Modeling through Stochastic Differential Equations", ICLR 2021
Song et al, "Denoising Diffusion Implicit Models", ICLR 2021

Diffusion Models

Warning: Terminology and notation in this area is a mess!

There are many different mathematical formalisms; tons of variance in terminology and notation between papers.

We'll just cover the basics of a modern “clean” implementation (Rectified Flow)

Diffusion Models: Intuition

Pick a **noise distribution** $z \sim p_{noise}$
(Usually unit Gaussian)

Diffusion Models: Intuition

Pick a **noise distribution** $z \sim p_{noise}$
(Usually unit Gaussian)

Consider data x corrupted under varying
noise levels t to give noisy data x_t



Diffusion Models: Intuition

Pick a **noise distribution** $z \sim p_{noise}$
(Usually unit Gaussian)

Consider data x corrupted under varying
noise levels t to give noisy data x_t

Train a neural network to **remove a little bit of noise**: $f_\theta(x_t, t)$



$t = 0$
No noise

$t = 1$
Full noise

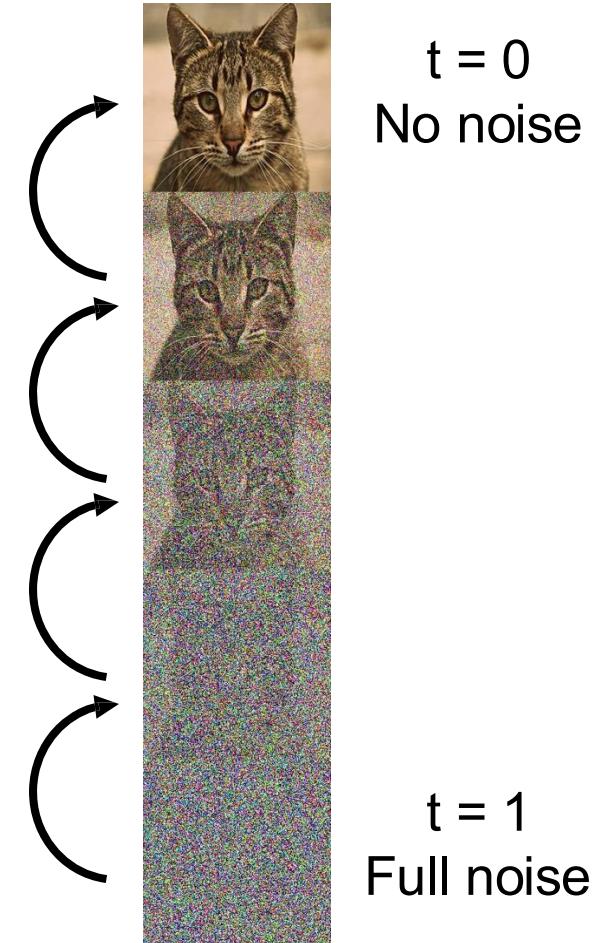
Diffusion Models: Intuition

Pick a **noise distribution** $z \sim p_{noise}$
(Usually unit Gaussian)

Consider data x corrupted under varying
noise levels t to give noisy data x_t

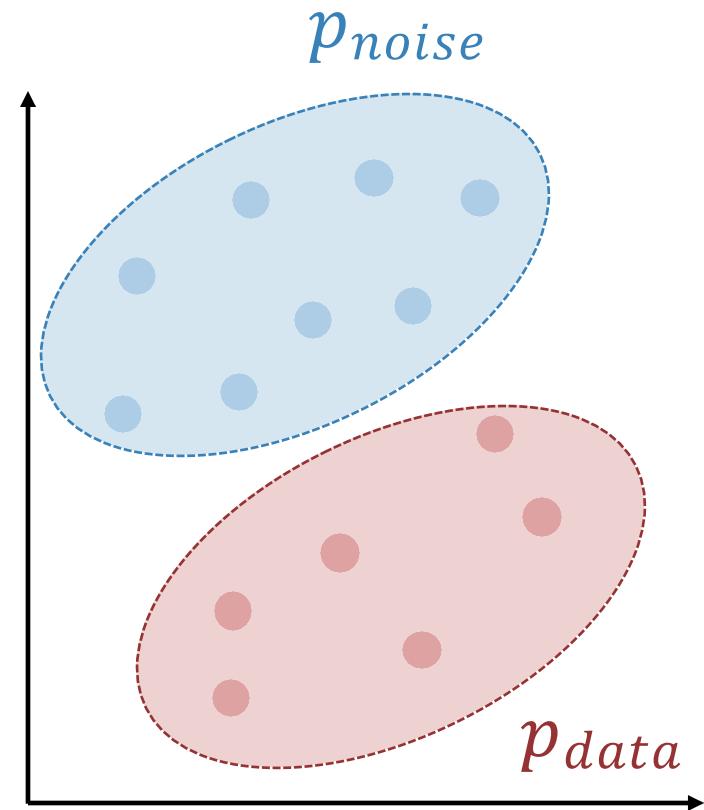
Train a neural network to **remove a little bit of noise**: $f_\theta(x_t, t)$

At inference time, sample $x_1 \sim p_{noise}$ and apply f_θ many times in sequence to generate a noiseless sample x_0



Diffusion Models: Rectified Flow

Suppose we have a simple p_{noise}
(e.g. Gaussian) and samples from p_{data}

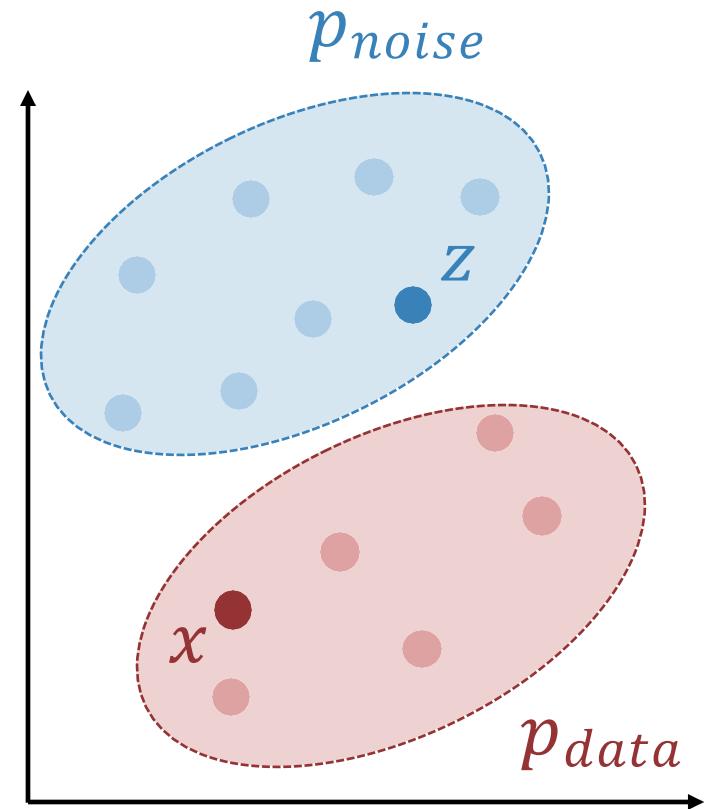


Diffusion Models: Rectified Flow

Suppose we have a simple p_{noise}
(e.g. Gaussian) and samples from p_{data}

On each training iteration, sample:

$$z \sim p_{noise} \quad x \sim p_{data} \quad t \sim Uniform[0, 1]$$



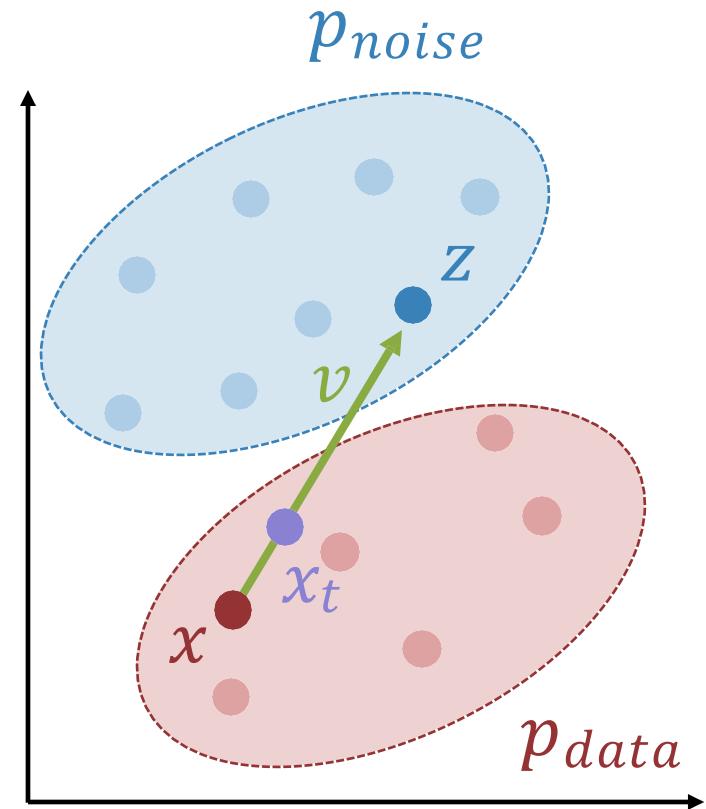
Diffusion Models: Rectified Flow

Suppose we have a simple p_{noise}
(e.g. Gaussian) and samples from p_{data}

On each training iteration, sample:

$$z \sim p_{noise} \quad x \sim p_{data} \quad t \sim Uniform[0, 1]$$

Set $x_t = (1 - t)x + tz$, $v = z - x$



Diffusion Models: Rectified Flow

Suppose we have a simple p_{noise}
(e.g. Gaussian) and samples from p_{data}

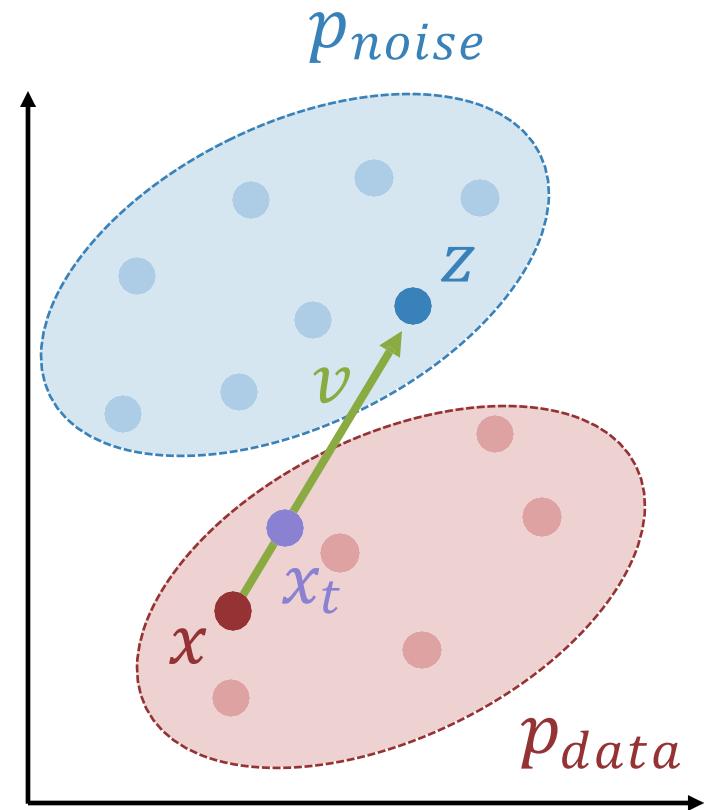
On each training iteration, sample:

$$z \sim p_{noise} \quad x \sim p_{data} \quad t \sim Uniform[0, 1]$$

Set $x_t = (1 - t)x + tz$, $v = z - x$

Train a neural network to predict v :

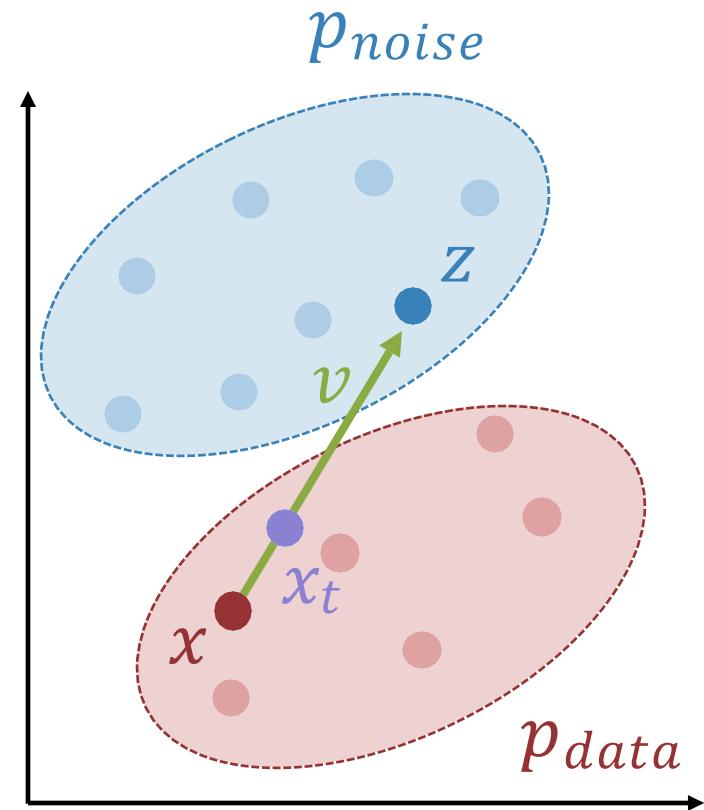
$$L = \|f_\theta(x_t, t) - v\|_2^2$$



Diffusion Models: Rectified Flow

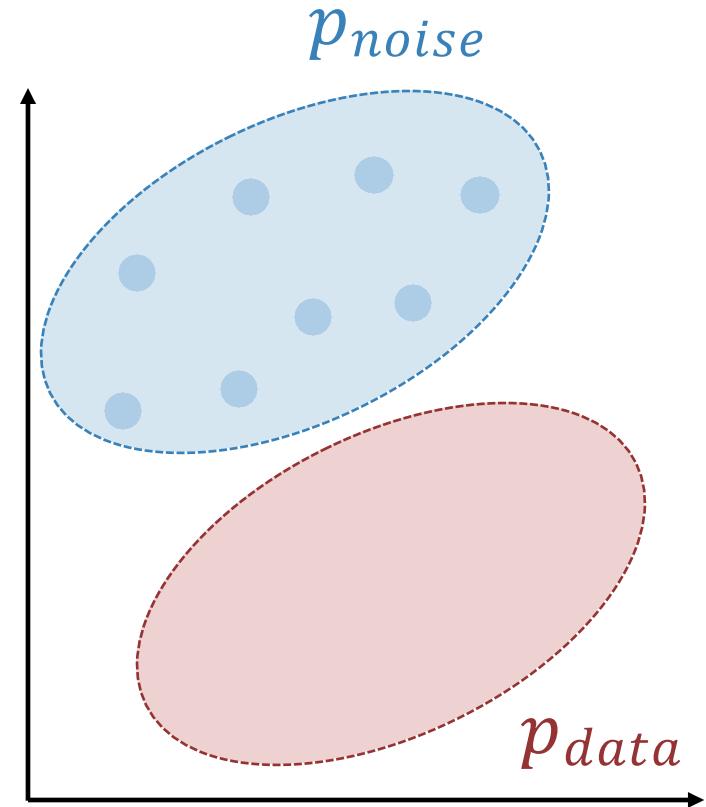
Core training loop is just
a few lines of code!

```
for x in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    v = model(xt, t)  
    loss = (z - x - v).square().sum()
```



Rectified Flow: Sampling

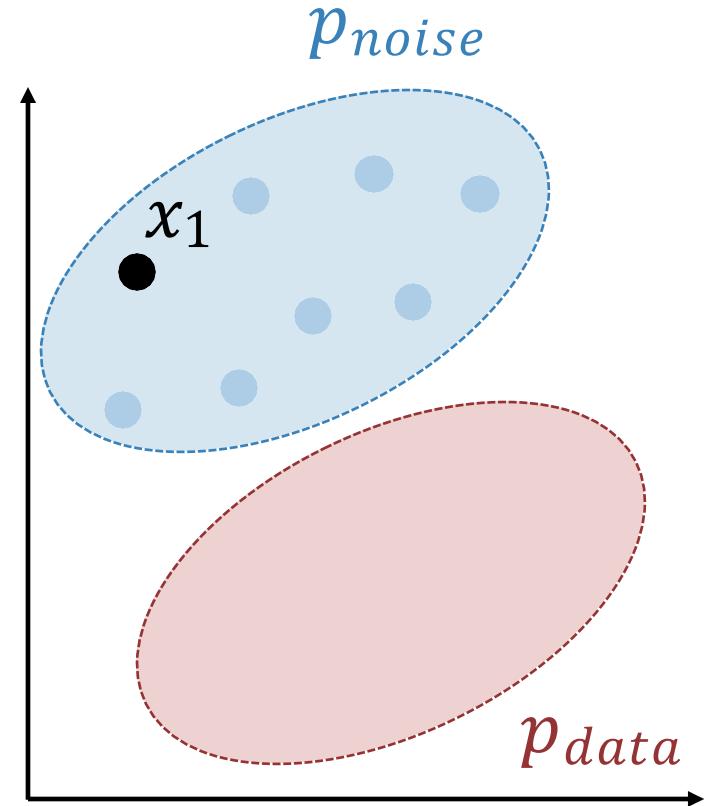
Choose number of steps T (often T=50)



Rectified Flow: Sampling

Choose number of steps T (often $T=50$)

Sample $x \sim p_{noise}$



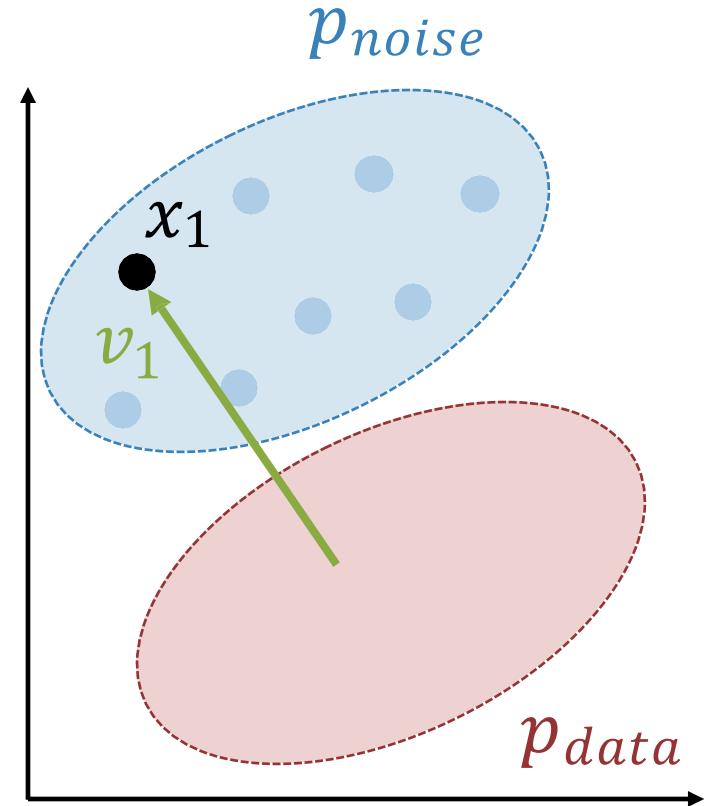
Rectified Flow: Sampling

Choose number of steps T (often $T=50$)

Sample $x \sim p_{noise}$

For t in $[1, 1 - \frac{1}{T}, 1 - \frac{2}{T}, \dots, 0]$:

Evaluate $v_t = f_\theta(x_t, t)$



Rectified Flow: Sampling

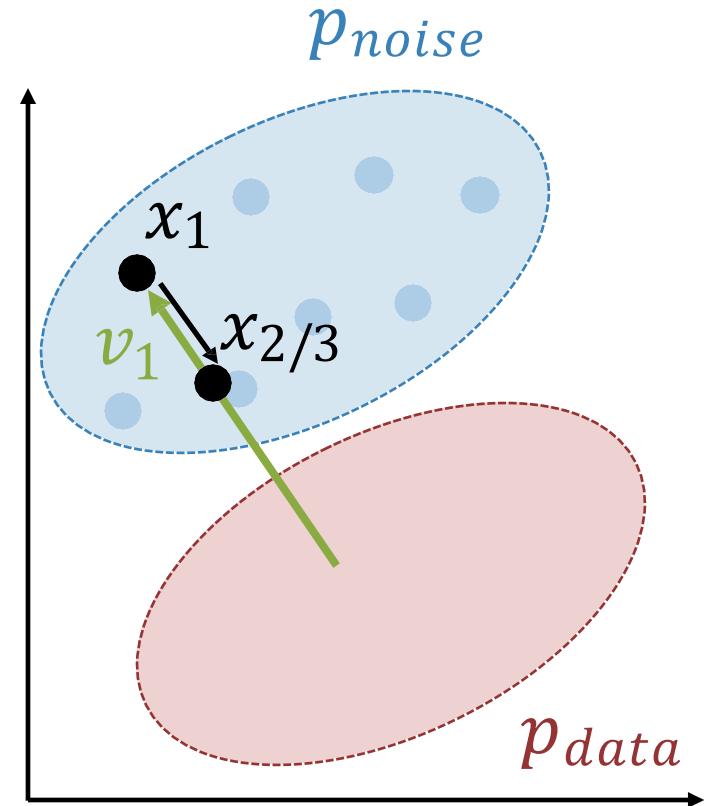
Choose number of steps T (often $T=50$)

Sample $x \sim p_{noise}$

For t in $[1, 1 - \frac{1}{T}, 1 - \frac{2}{T}, \dots, 0]$:

Evaluate $v_t = f_\theta(x_t, t)$

Step $x = x - v_t/T$



Rectified Flow: Sampling

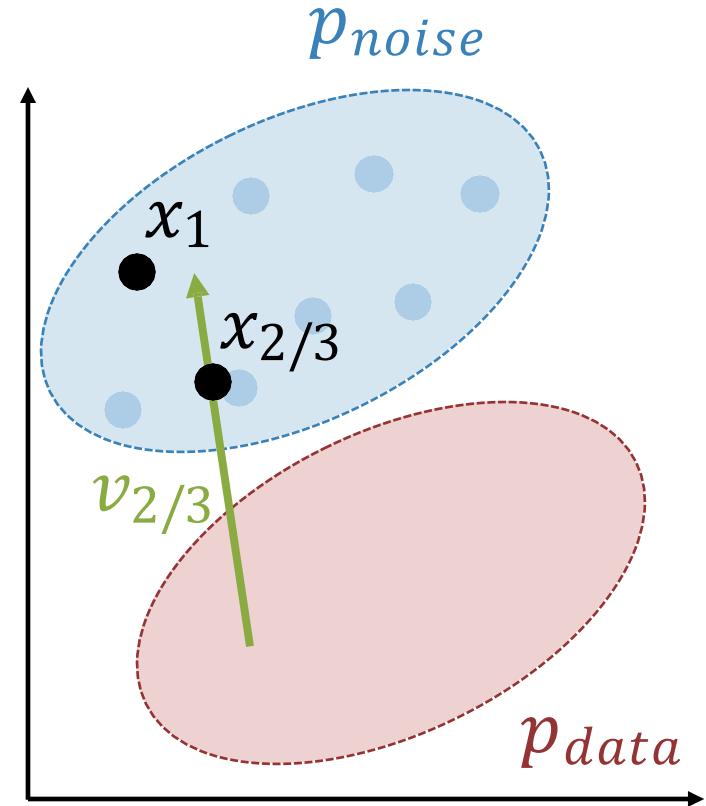
Choose number of steps T (often $T=50$)

Sample $x \sim p_{noise}$

For t in $[1, 1 - \frac{1}{T}, 1 - \frac{2}{T}, \dots, 0]$:

Evaluate $v_t = f_\theta(x_t, t)$

Step $x = x - v_t/T$



Rectified Flow: Sampling

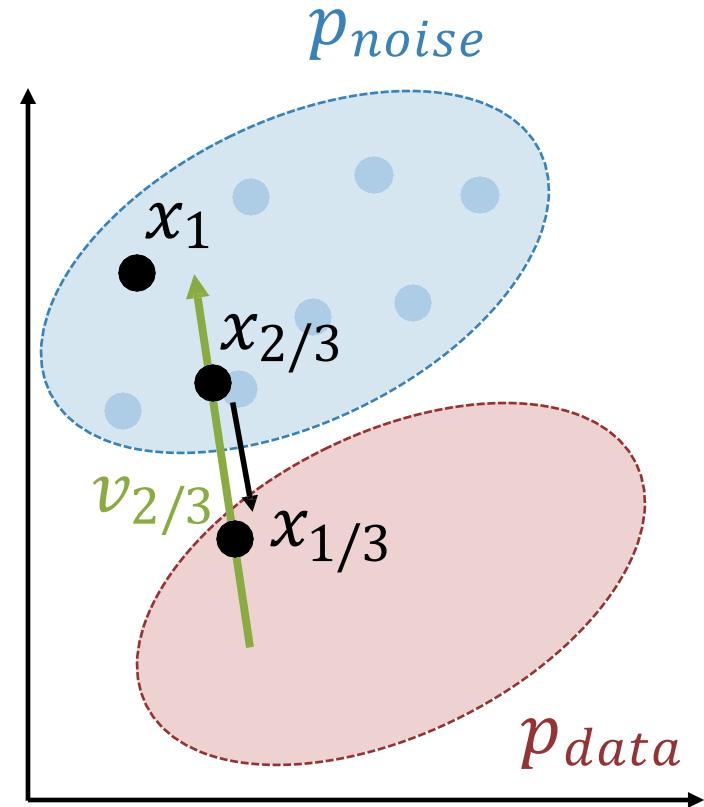
Choose number of steps T (often $T=50$)

Sample $x \sim p_{noise}$

For t in $[1, 1 - \frac{1}{T}, 1 - \frac{2}{T}, \dots, 0]$:

Evaluate $v_t = f_\theta(x_t, t)$

Step $x = x - v_t/T$



Rectified Flow: Sampling

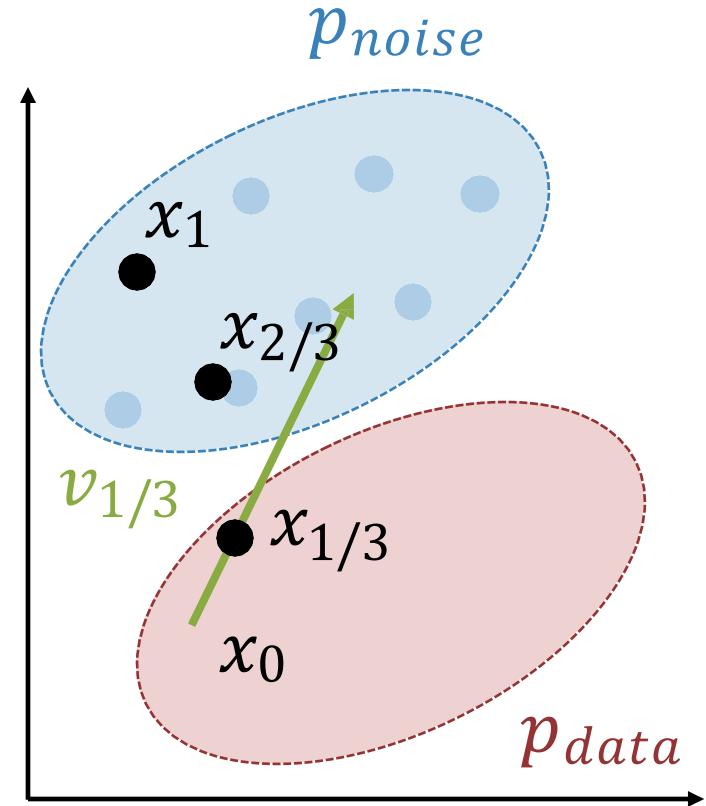
Choose number of steps T (often $T=50$)

Sample $x \sim p_{noise}$

For t in $[1, 1 - \frac{1}{T}, 1 - \frac{2}{T}, \dots, 0]$:

Evaluate $v_t = f_\theta(x_t, t)$

Step $x = x - v_t/T$



Rectified Flow: Sampling

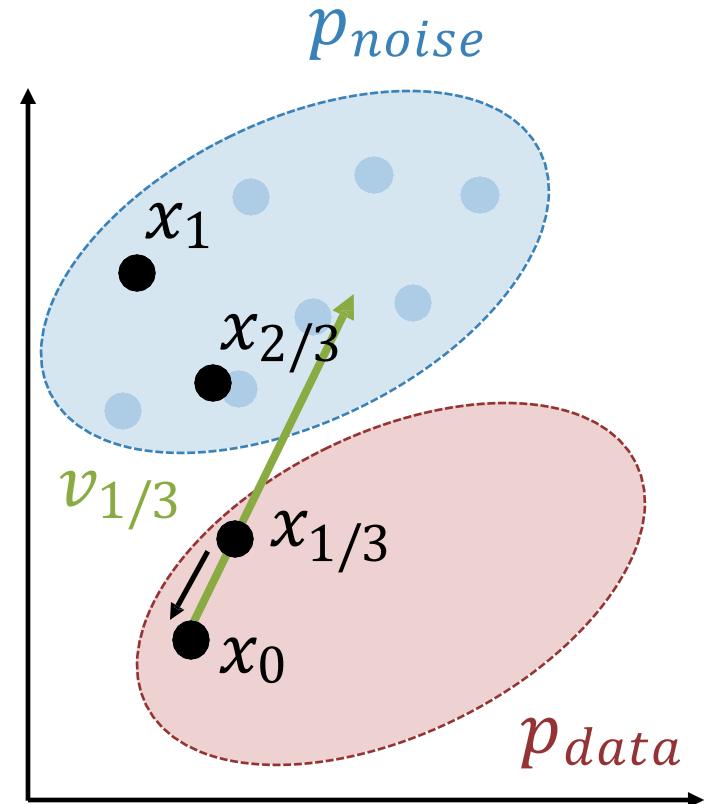
Choose number of steps T (often $T=50$)

Sample $x \sim p_{noise}$

For t in $[1, 1 - \frac{1}{T}, 1 - \frac{2}{T}, \dots, 0]$:

Evaluate $v_t = f_\theta(x_t, t)$

Step $x = x - v_t/T$



Rectified Flow: Sampling

Choose number of steps T (often $T=50$)

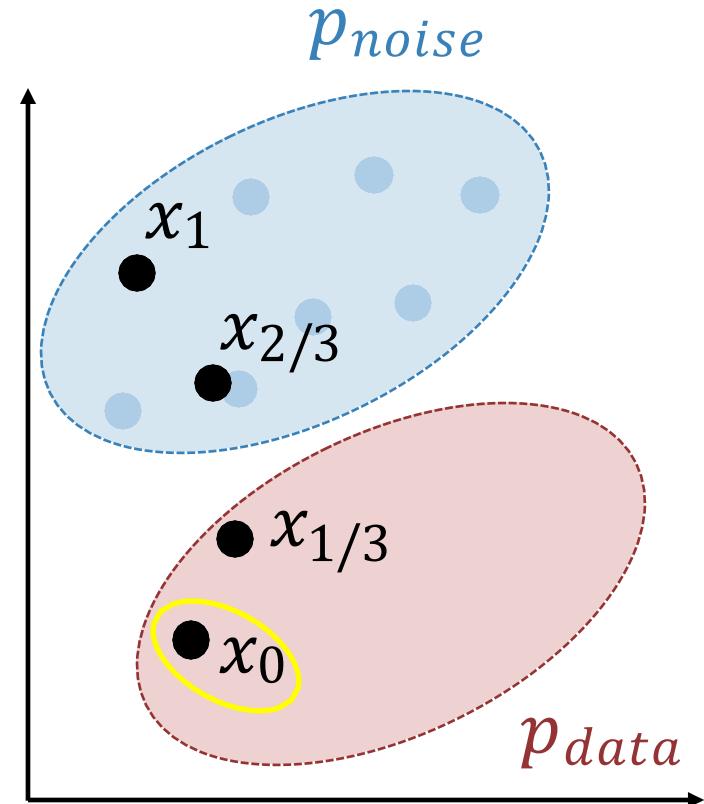
Sample $x \sim p_{noise}$

For t in $[1, 1 - \frac{1}{T}, 1 - \frac{2}{T}, \dots, 0]$:

Evaluate $v_t = f_\theta(x_t, t)$

Step $x = x - v_t/T$

Return x



Rectified Flow: Sampling

Choose number of steps T (often $T=50$)

Sample $x \sim p_{noise}$

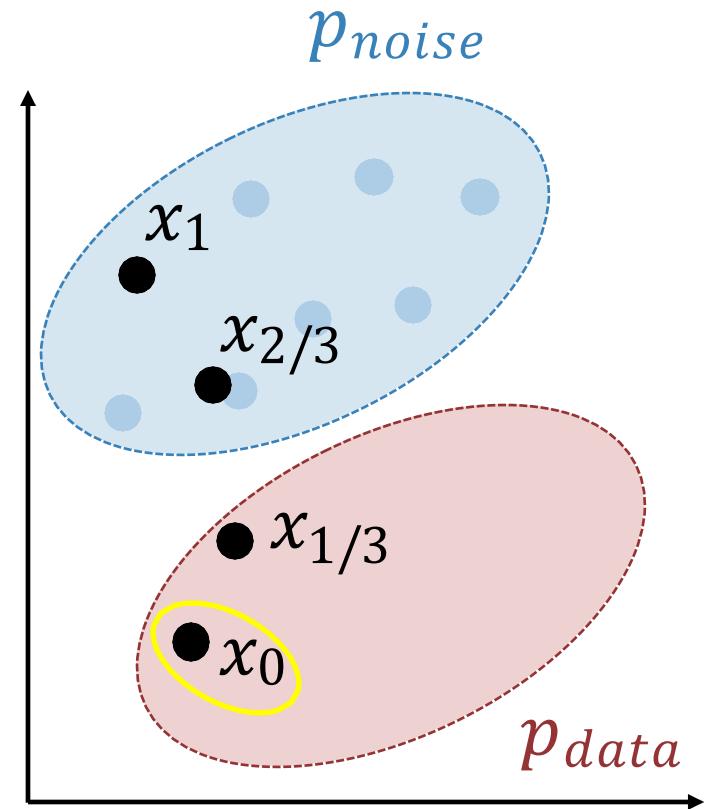
For t in $[1, 1 - \frac{1}{T}, 1 - \frac{2}{T}, \dots, 0]$:

Evaluate $v_t = f_\theta(x_t, t)$

Step $x = x - v_t/T$

Return x

```
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, t)
    sample = sample - v / num_steps
```



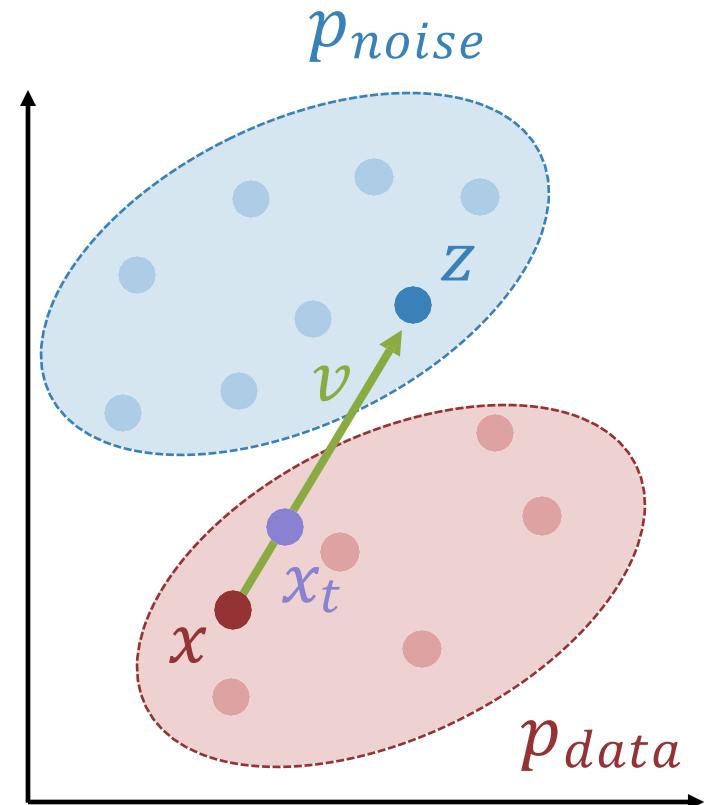
Rectified Flow: Summary

Training

```
for x in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    v = model(xt, t)  
    loss = (z - x - v).square().sum()
```

Sampling

```
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    v = model(sample, t)  
    sample = sample - v / num_steps
```



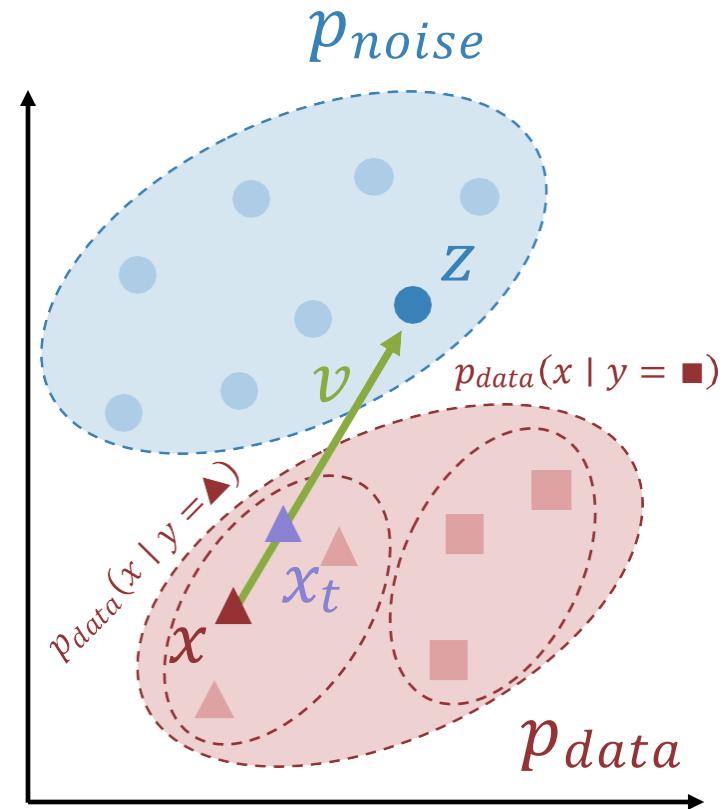
Conditional Rectified Flow

Training

```
for x in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    v = model(xt, t)  
    loss = (z - x - v).square().sum()
```

Sampling

```
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    v = model(sample, t)  
    sample = sample - v / num_steps
```



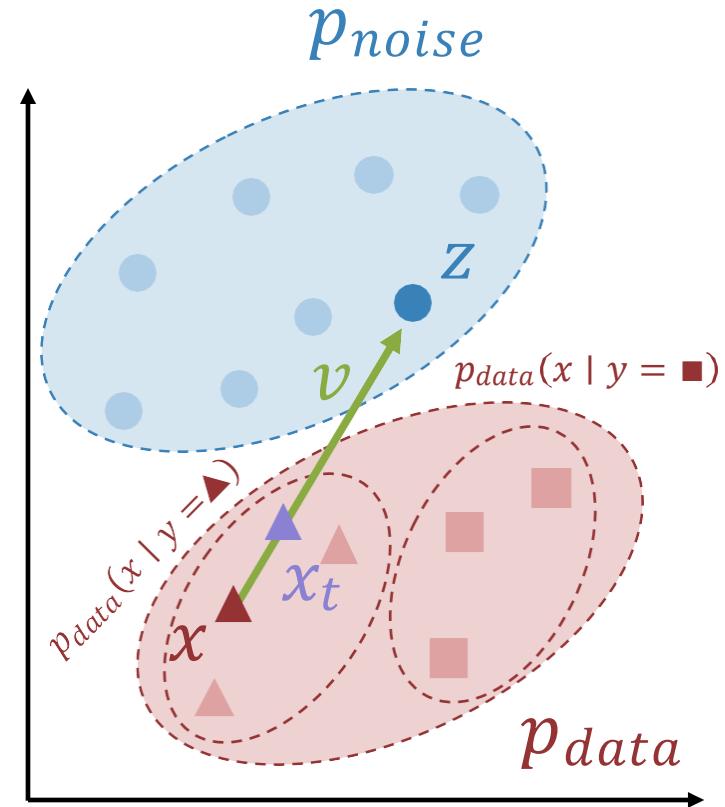
Conditional Rectified Flow

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Sampling

```
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    v = model(sample, t)  
    sample = sample - v / num_steps
```



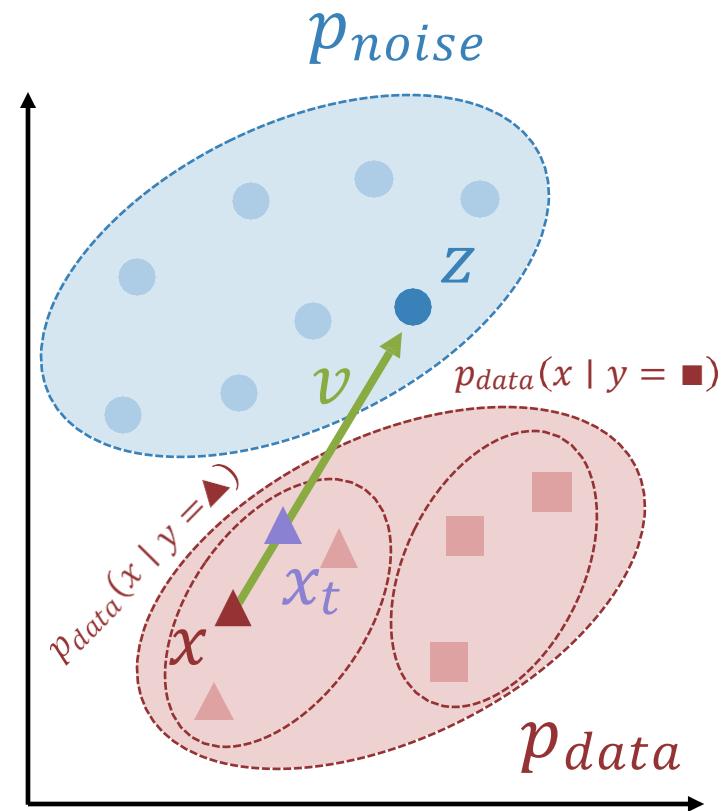
Conditional Rectified Flow

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Sampling

```
y = user_input()  
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    v = model(sample, y, t)  
    sample = sample - v / num_steps
```



Conditional Rectified Flow

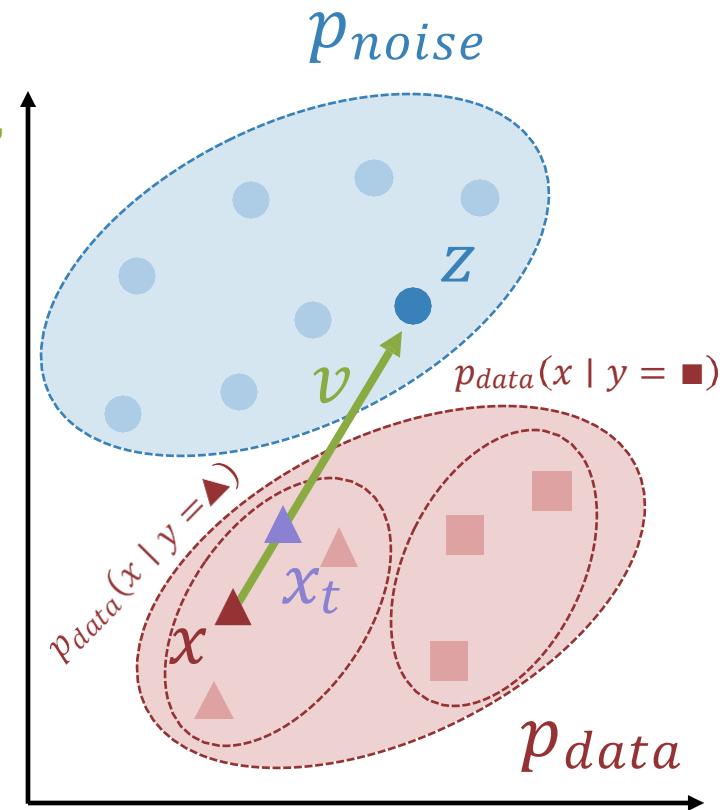
Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y ?

Sampling

```
y = user_input()  
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    v = model(sample, y, t)  
    sample = sample - v / num_steps
```



Classifier-Free Guidance (CFG)

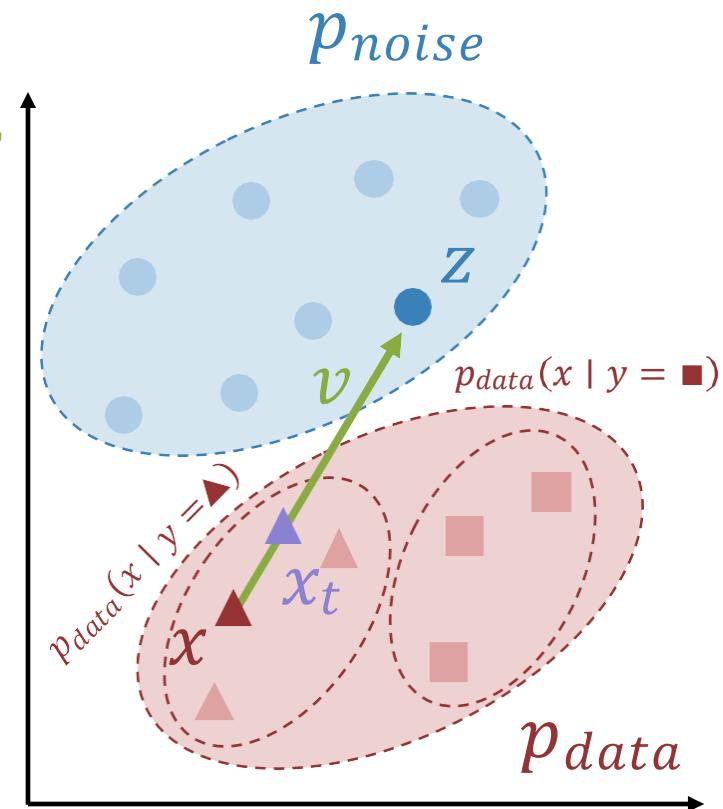
Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y ?

Randomly drop y during training.

Now the same model is conditional and unconditional!



Classifier-Free Guidance (CFG)

Training

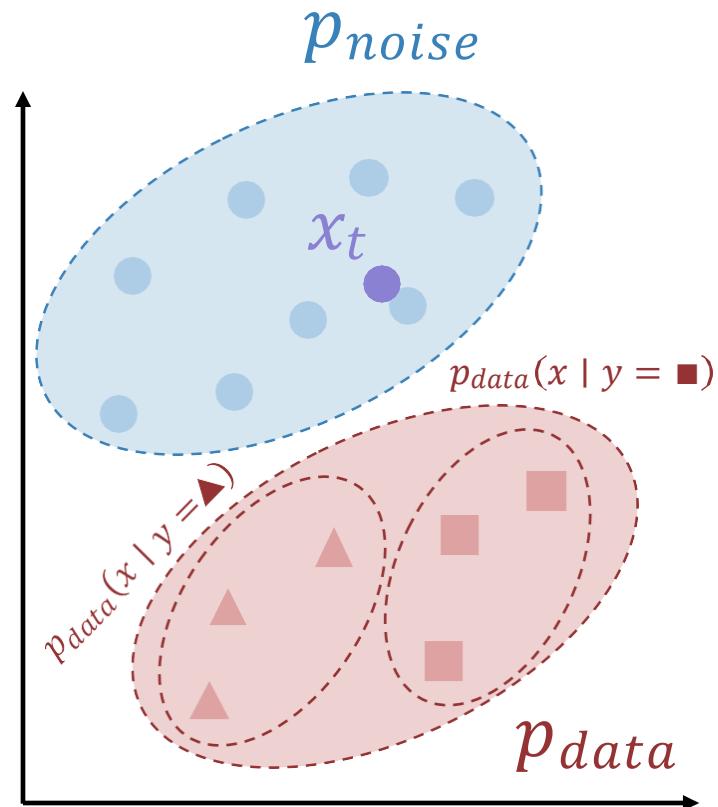
```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y ?

Randomly drop y during training.

Now the same model is conditional and unconditional!

Consider a noisy x_t :



Classifier-Free Guidance (CFG)

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

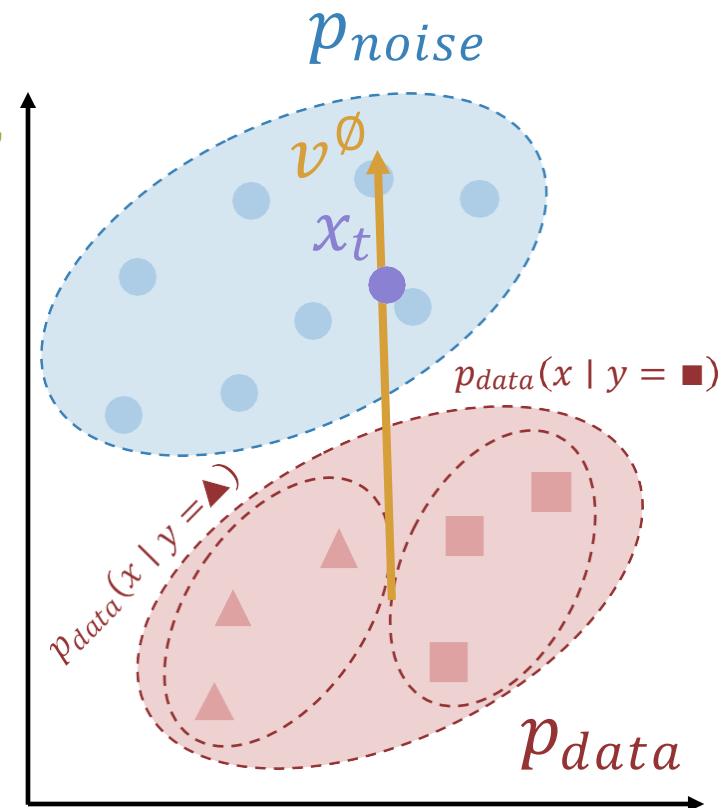
Can we control how much we “emphasize” the conditioning y ?

Randomly drop y during training.

Now the same model is conditional and unconditional!

Consider a noisy x_t :

$v^\emptyset = f_\theta(x_t, y_\emptyset, t)$ points toward $p(x)$



Classifier-Free Guidance (CFG)

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y ?

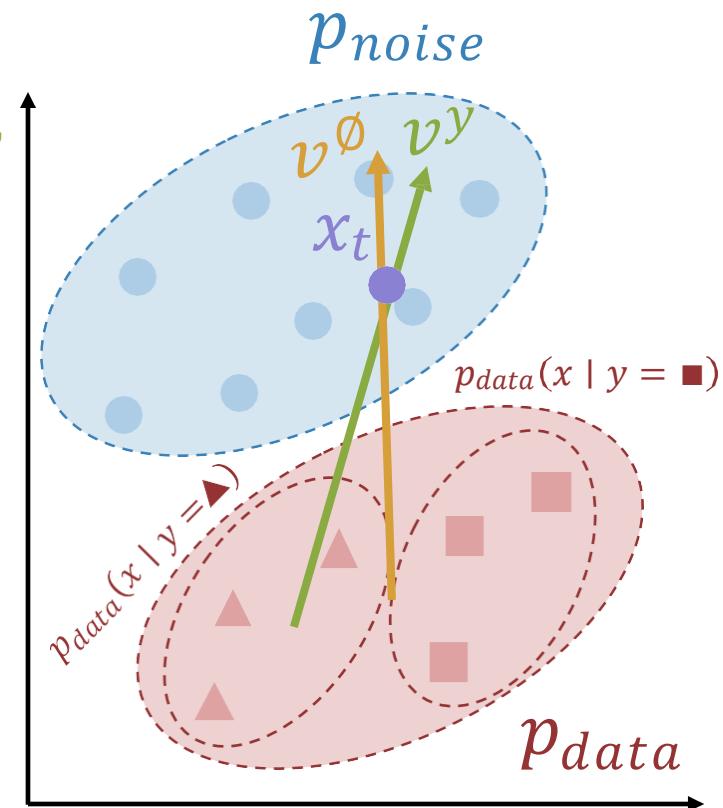
Randomly drop y during training.

Now the same model is conditional and unconditional!

Consider a noisy x_t :

$v^\emptyset = f_\theta(x_t, y_\emptyset, t)$ points toward $p(x)$

$v^y = f_\theta(x_t, y, t)$ points toward $p(x | y)$



Classifier-Free Guidance (CFG)

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y ?

Randomly drop y during training.

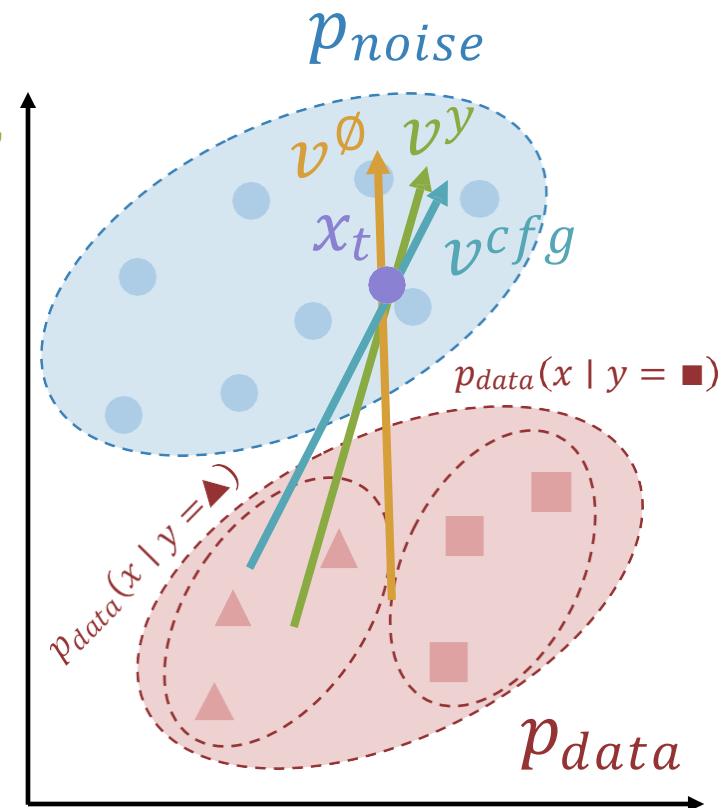
Now the same model is conditional and unconditional!

Consider a noisy x_t :

$v^\emptyset = f_\theta(x_t, y_\emptyset, t)$ points toward $p(x)$

$v^y = f_\theta(x_t, y, t)$ points toward $p(x | y)$

$v^{cfg} = (1 + w)v^y - wv^\emptyset$ points more toward $p(x | y)$



Classifier-Free Guidance (CFG)

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y ?

Randomly drop y during training.

Now the same model is conditional and unconditional!

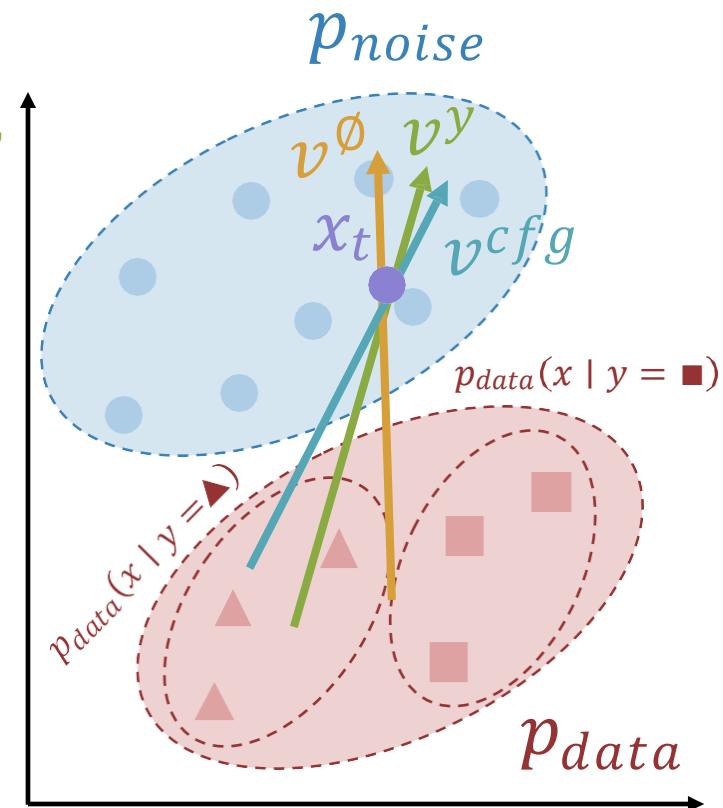
Consider a noisy x_t :

$v^\emptyset = f_\theta(x_t, y_\emptyset, t)$ points toward $p(x)$

$v^y = f_\theta(x_t, y, t)$ points toward $p(x | y)$

$v^{cfg} = (1 + w)v^y - wv^\emptyset$ points more toward $p(x | y)$

During sampling, step according to v^{cfg}



Classifier-Free Guidance (CFG)

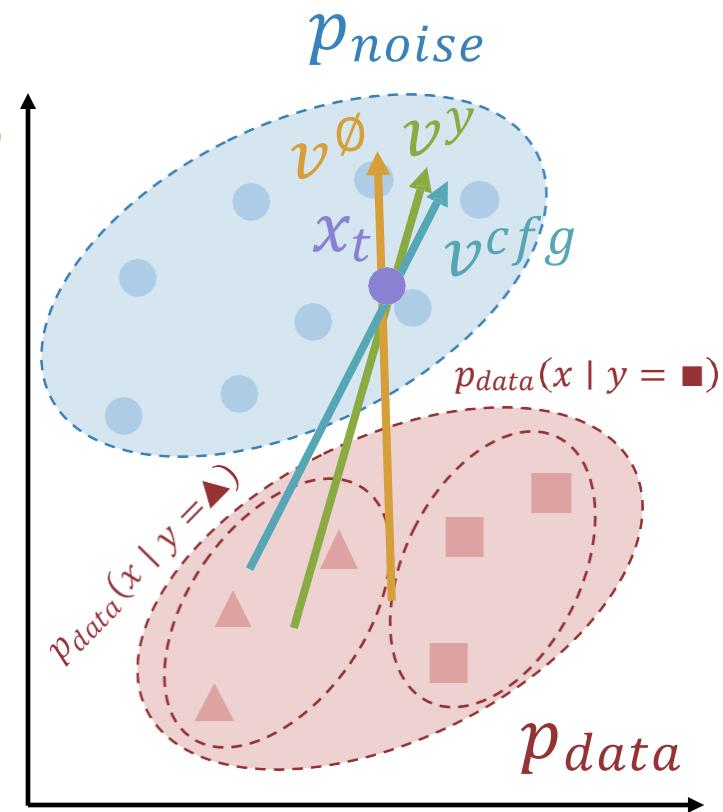
Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y ?

Sampling

```
y = user_input()  
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    vy = model(sample, y, t)  
    v0 = model(sample, y_null, t)  
    v = (1 + w) * vy - w * v0  
    sample = sample - v / num_steps
```



Classifier-Free Guidance (CFG)

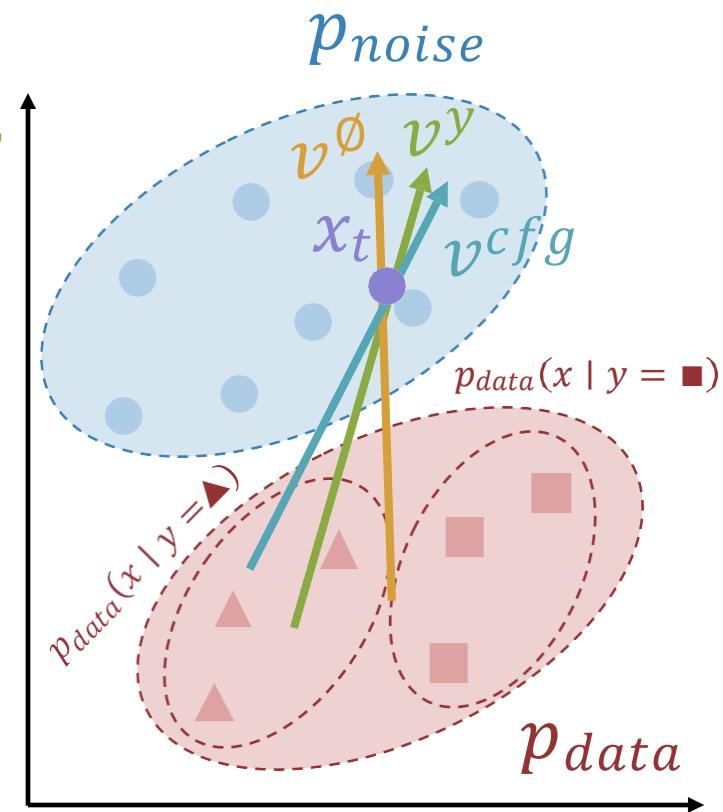
Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y ?

Sampling

```
y = user_input()  
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    vy = model(sample, y, t)  
    v0 = model(sample, y_null, t)  
    v = (1 + w) * vy - w * v0  
    sample = sample - v / num_steps
```



Classifier-Free Guidance (CFG)

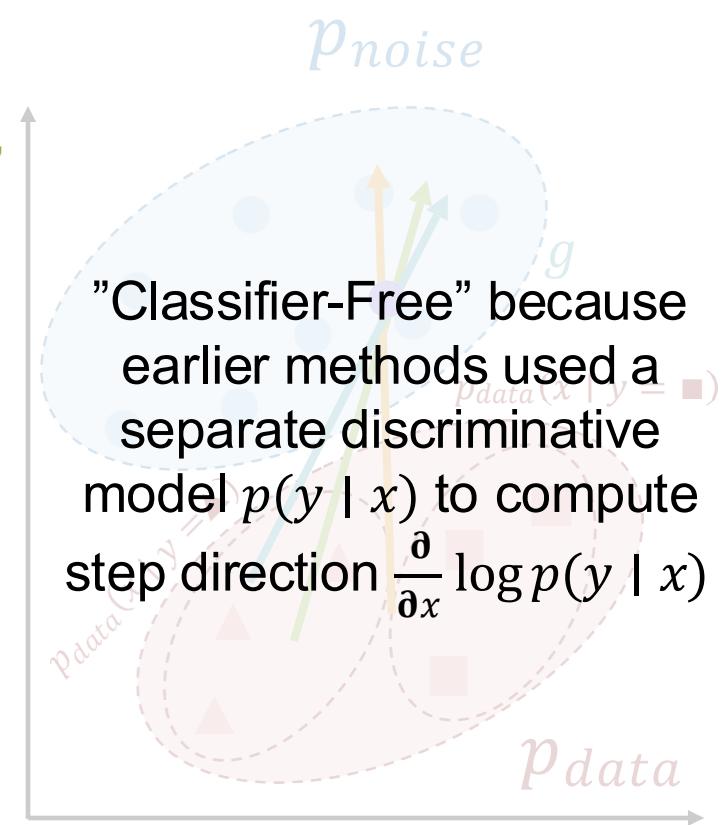
Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y ?

Sampling

```
y = user_input()  
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    vy = model(sample, y, t)  
    v0 = model(sample, y_null, t)  
    v = (1 + w) * vy - w * v0  
    sample = sample - v / num_steps
```



Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Classifier-Free Guidance (CFG)

Training

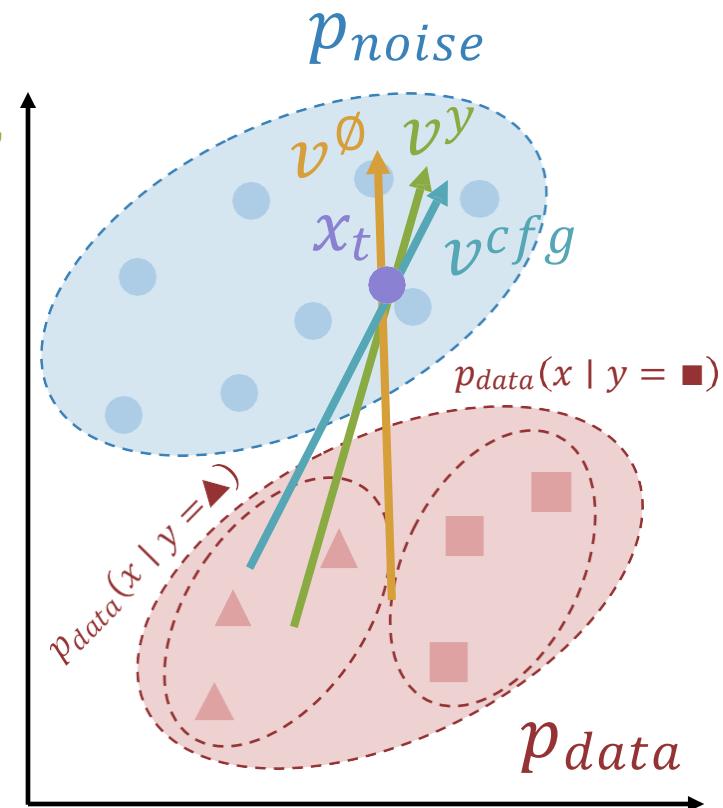
```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y ?

Sampling

```
y = user_input()  
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    vy = model(sample, y, t)  
    v0 = model(sample, y_null, t)  
    v = (1 + w) * vy - w * v0  
    sample = sample - v / num_steps
```

Used everywhere in practice! Very important for high-quality outputs



Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Classifier-Free Guidance (CFG)

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

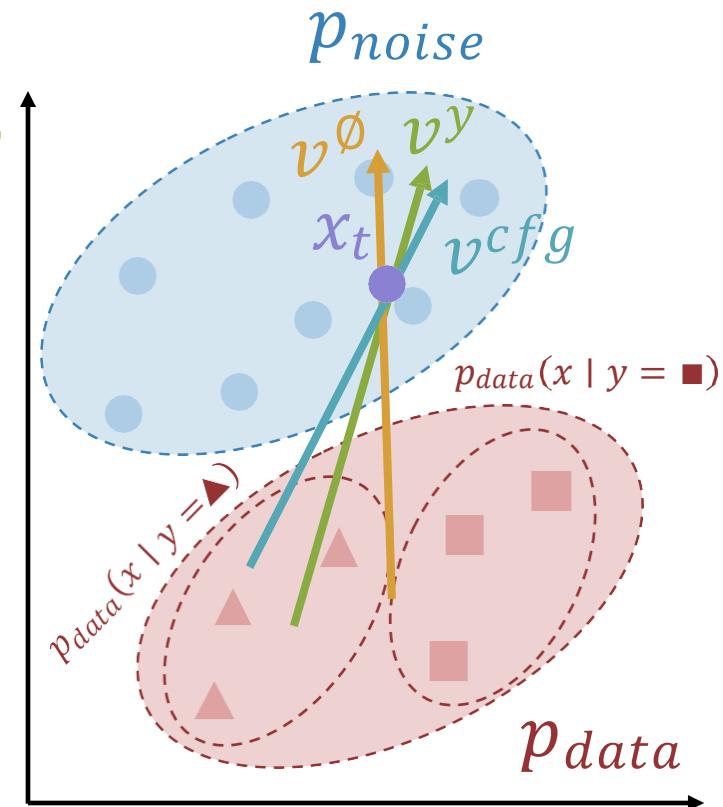
Can we control how much we “emphasize” the conditioning y ?

Sampling

```
y = user_input()  
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    vy = model(sample, y, t)  
    vθ = model(sample, y_null, t)  
    v = (1 + w) * vy - w * vθ  
    sample = sample - v / num_steps
```

Used everywhere in practice! Very important for high-quality outputs

Doubles the cost of sampling...



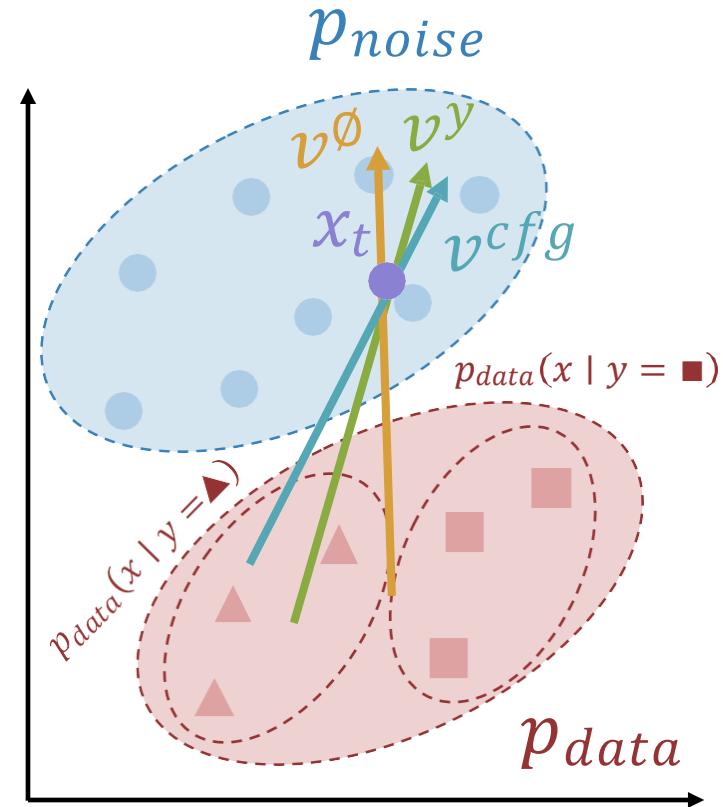
Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Optimal Prediction

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?



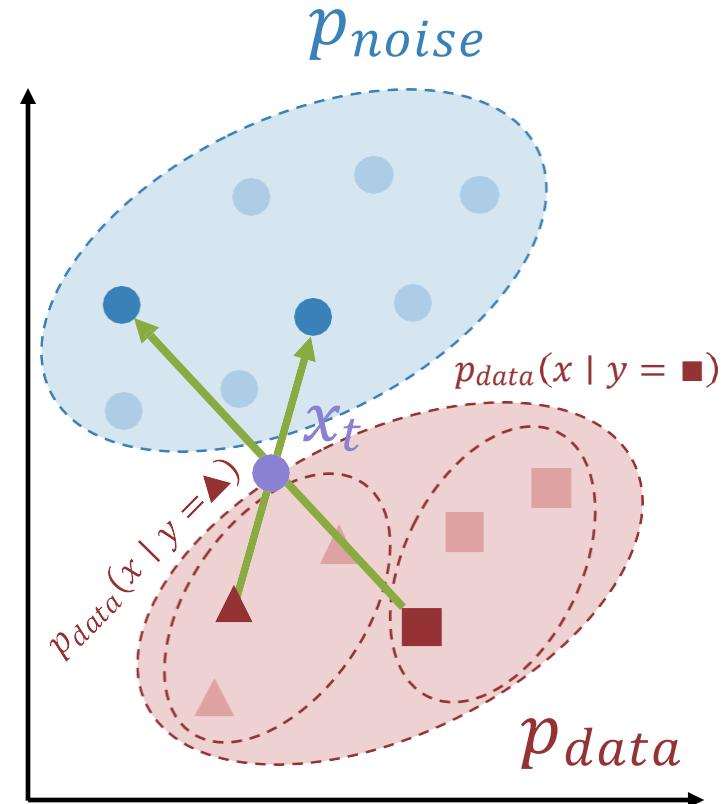
Optimal Prediction

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

There may be many pairs (x, z) that give the same x_t ; network must average over them



Dhariwal and Nichol, "Diffusion Models beat GANs on Image Synthesis", arXiv 2021
Ho and Salimans, "Classifier-Free Diffusion Guidance", arXiv 2022

Optimal Prediction

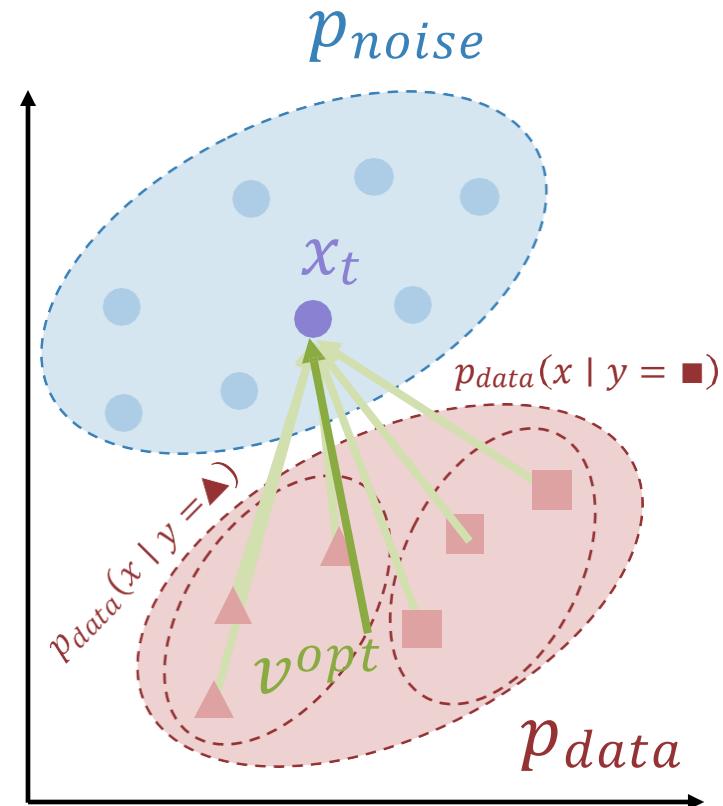
Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

There may be many pairs (x, z) that give the same x_t ; network must average over them

Full noise ($t=1$) is easy: optimal v is mean of p_{data}



Dhariwal and Nichol, "Diffusion Models beat GANs on Image Synthesis", arXiv 2021
Ho and Salimans, "Classifier-Free Diffusion Guidance", arXiv 2022

Optimal Prediction

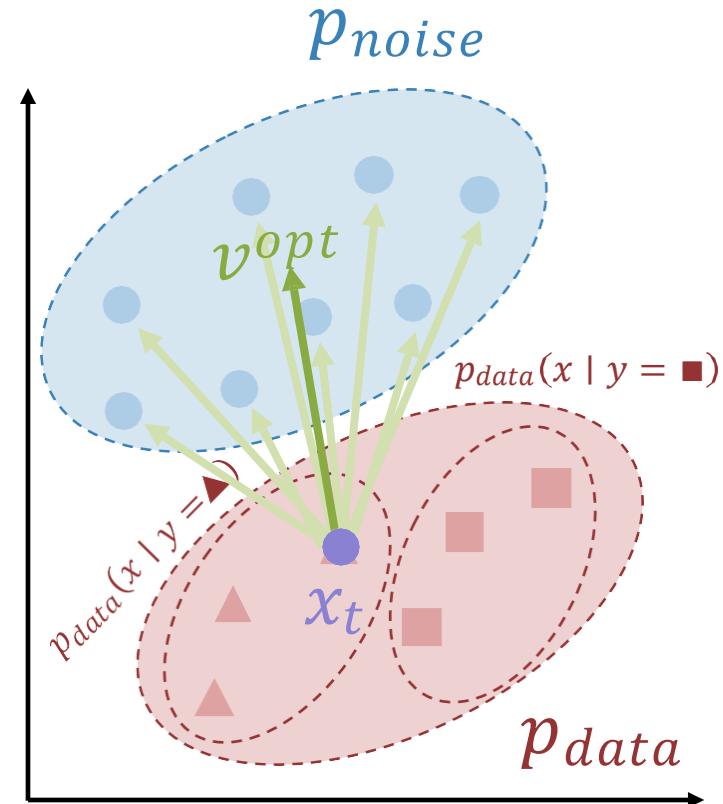
Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

There may be many pairs (x, z) that give the same x_t ; network must average over them

Full noise ($t=1$) is easy: optimal v is mean of p_{data}
No noise ($t=0$) is easy: optimal v is mean of p_{noise}



Optimal Prediction

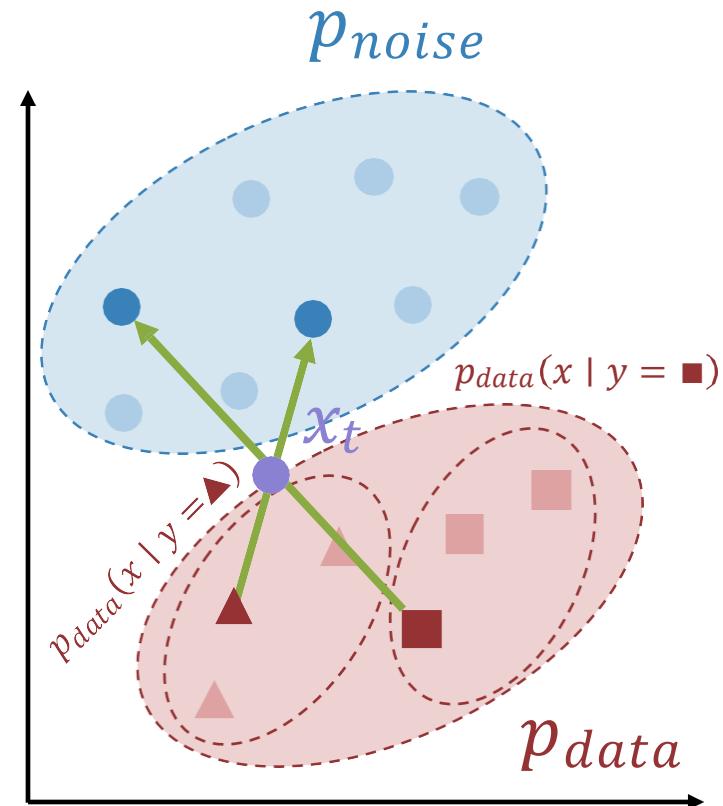
Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

There may be many pairs (x, z) that give the same x_t ; network must average over them

Full noise ($t=1$) is easy: optimal v is mean of p_{data}
No noise ($t=0$) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous



Optimal Prediction

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

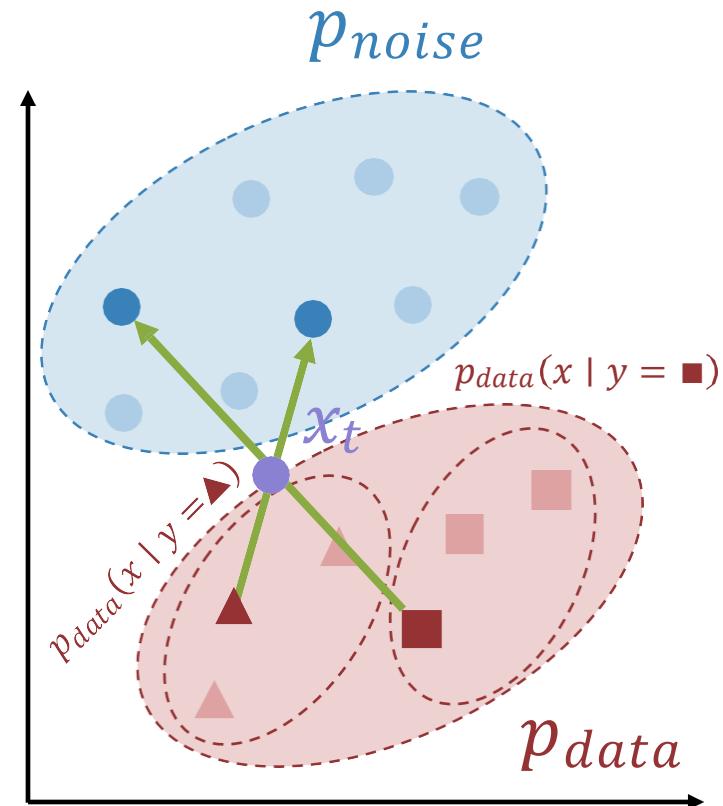
There may be many pairs (x, z) that give the same x_t ; network must average over them

Full noise ($t=1$) is easy: optimal v is mean of p_{data}

No noise ($t=0$) is easy: optimal v is mean of p_{noise}

Middle noise is hardest, most ambiguous

But we give equal weight to all noise levels!



Dhariwal and Nichol, "Diffusion Models beat GANs on Image Synthesis", arXiv 2021
Ho and Salimans, "Classifier-Free Diffusion Guidance", arXiv 2022

Optimal Prediction

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

There may be many pairs (x, z) that give the same x_t ; network must average over them

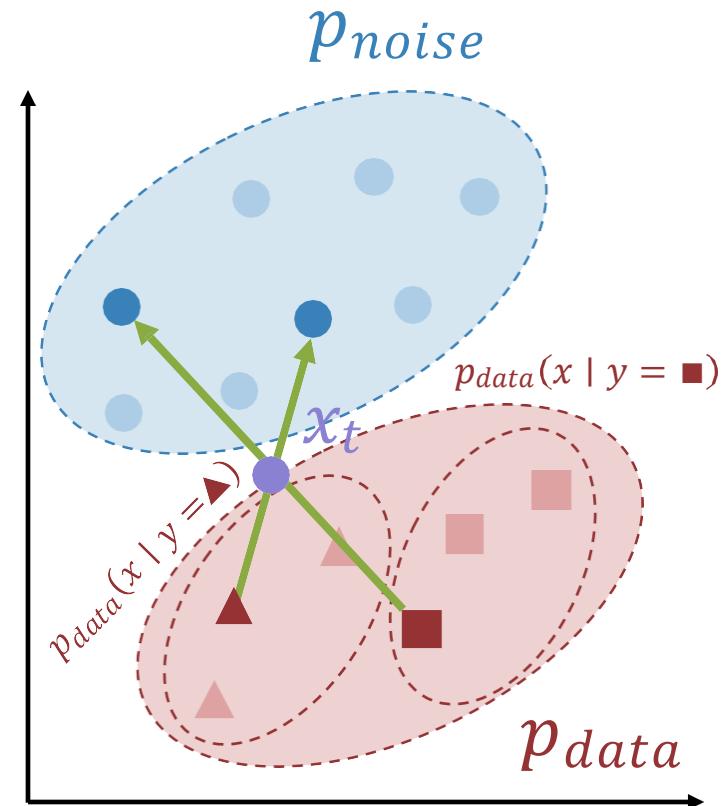
Full noise ($t=1$) is easy: optimal v is mean of p_{data}

No noise ($t=0$) is easy: optimal v is mean of p_{noise}

Middle noise is hardest, most ambiguous

But we give equal weight to all noise levels!

Solution: Use a non-uniform noise schedule



Dhariwal and Nichol, "Diffusion Models beat GANs on Image Synthesis", arXiv 2021
Ho and Salimans, "Classifier-Free Diffusion Guidance", arXiv 2022

Noise Schedules

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

There may be many pairs (x, z) that give the same x_t ; network must average over them

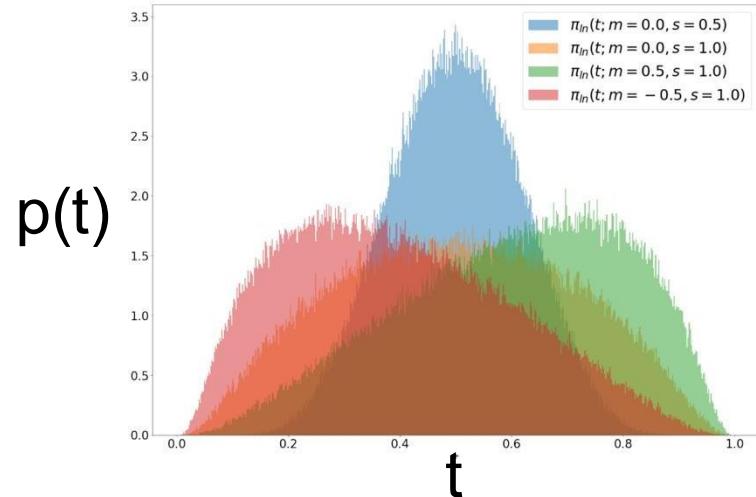
Full noise ($t=1$) is easy: optimal v is mean of p_{data}

No noise ($t=0$) is easy: optimal v is mean of p_{noise}

Middle noise is hardest, most ambiguous

But we give equal weight to all noise levels!

Solution: Use a non-uniform noise schedule



Put more emphasis on middle noise

Noise Schedules

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = torch.randn(()).sigmoid()  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

There may be many pairs (x, z) that give the same x_t ; network must average over them

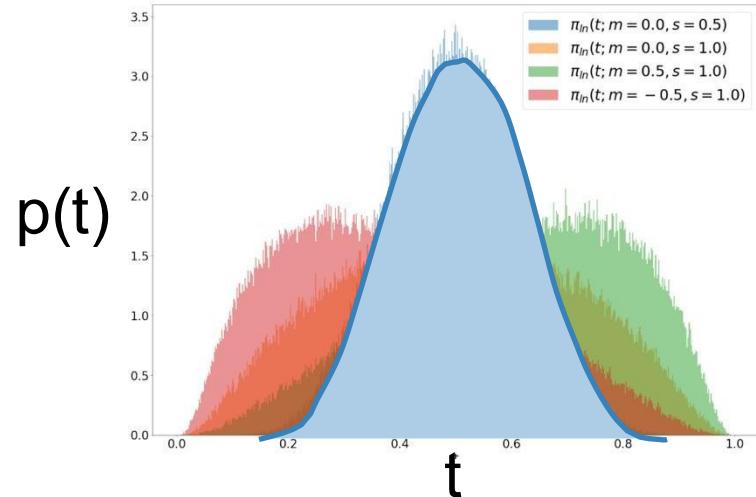
Full noise ($t=1$) is easy: optimal v is mean of p_{data}

No noise ($t=0$) is easy: optimal v is mean of p_{noise}

Middle noise is hardest, most ambiguous

But we give equal weight to all noise levels!

Solution: Use a non-uniform noise schedule



Put more emphasis on middle noise

Common choice: **logit-normal sampling**

Noise Schedules

Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = torch.randn(()).sigmoid()  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

There may be many pairs (x, z) that give the same x_t ; network must average over them

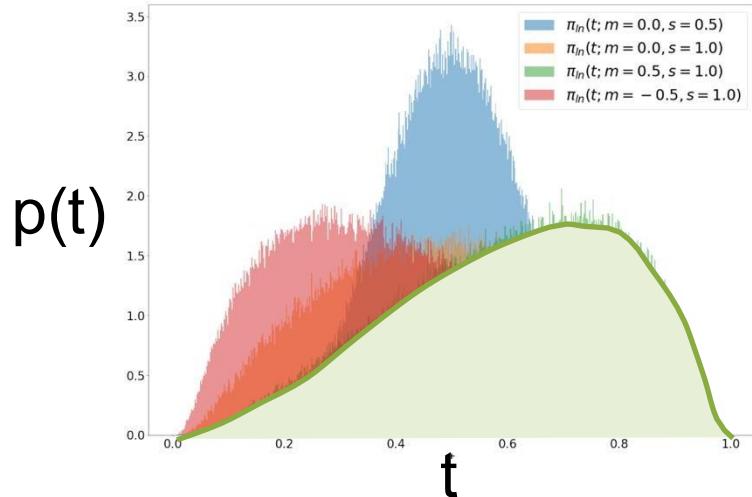
Full noise ($t=1$) is easy: optimal v is mean of p_{data}

No noise ($t=0$) is easy: optimal v is mean of p_{noise}

Middle noise is hardest, most ambiguous

But we give equal weight to all noise levels!

Solution: Use a non-uniform noise schedule



Put more emphasis on middle noise

Common choice: **logit-normal sampling**

For high-res data, often **shift to higher noise** to account for pixel correlations

Diffusion: Rectified Flow

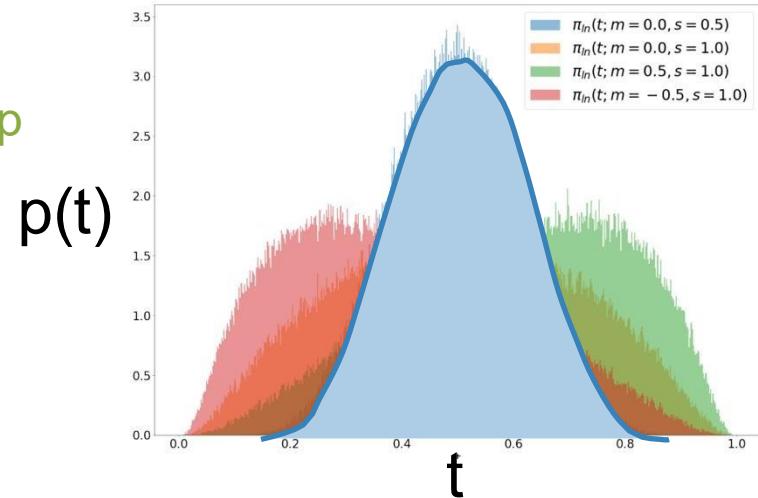
Training

```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = torch.randn(()).sigmoid()  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Simple and scalable setup
for many generative
modeling problems!

Sampling

```
y = user_input()  
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    vy = model(sample, y, t)  
    v0 = model(sample, y_null, t)  
    v = (1 + w) * vy - w * v0  
    sample = sample - v / num_steps
```



Put more emphasis on middle noise

Common choice: **logit-normal sampling**

For high-res data, often **shift to higher noise** to account for pixel correlations

Diffusion: Rectified Flow

Training

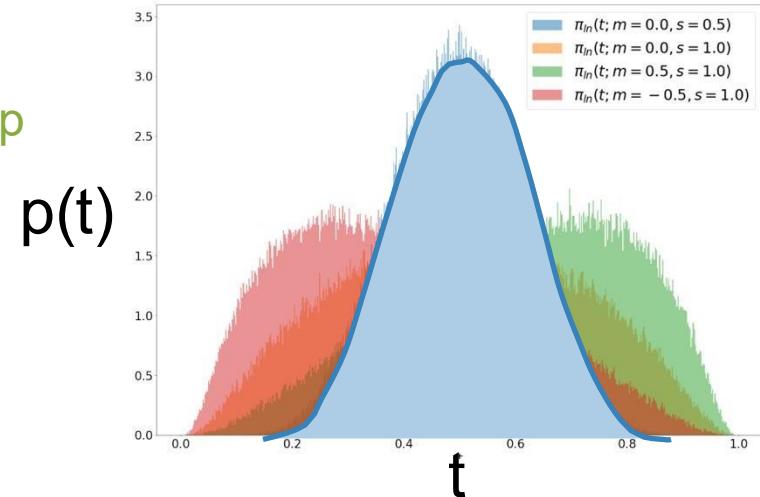
```
for (x, y) in dataset:  
    z = torch.randn_like(x)  
    t = torch.randn(()).sigmoid()  
    xt = (1 - t) * x + t * z  
    if random.random() < 0.5: y = y_null  
    v = model(xt, y, t)  
    loss = (z - x - v).square().sum()
```

Simple and scalable setup
for many generative
modeling problems!

Sampling

```
y = user_input()  
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    vy = model(sample, y, t)  
    v0 = model(sample, y_null, t)  
    v = (1 + w) * vy - w * v0  
    sample = sample - v / num_steps
```

Problem: Does not
work naively on high-
resolution data



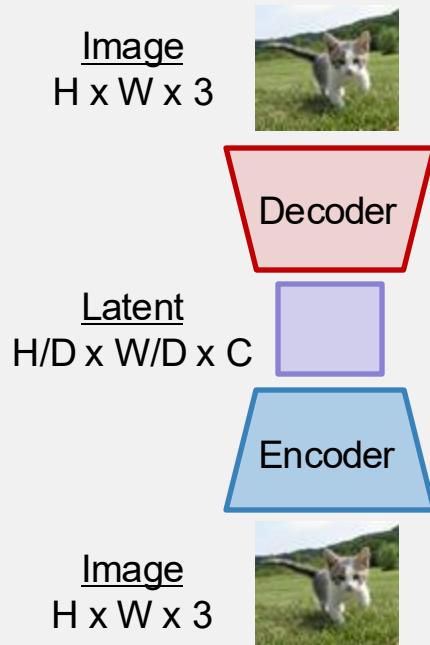
Put more emphasis on middle noise

Common choice: **logit-normal sampling**

For high-res data, often **shift to higher
noise** to account for pixel correlations

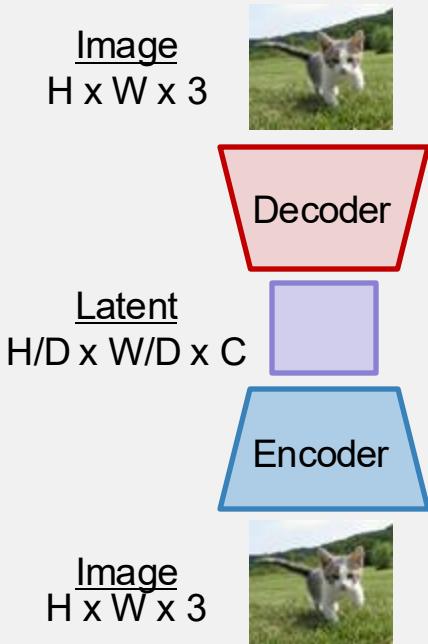
Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to
convert images to **latents**



Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to
convert images to **latents**



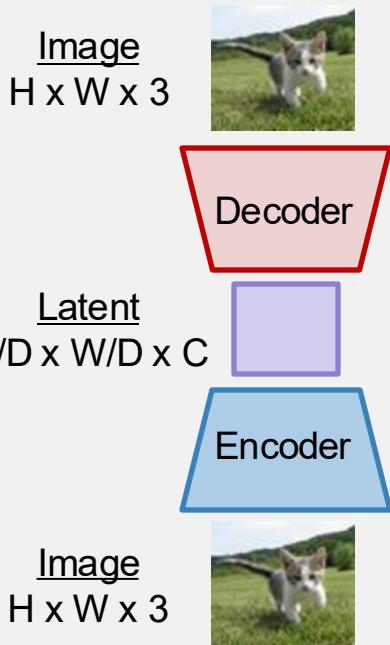
Common setting: D=8, C=16

Image: 256 x 256 x 3
=> **Latent:** 32 x 32 x 16

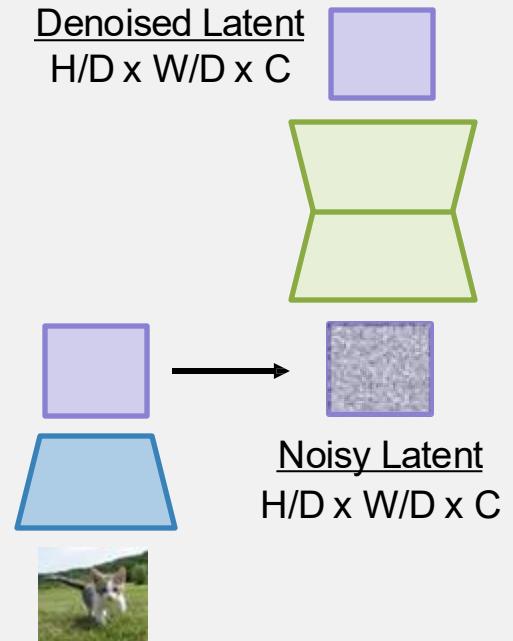
Encoder / Decoder are CNNs with attention

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**

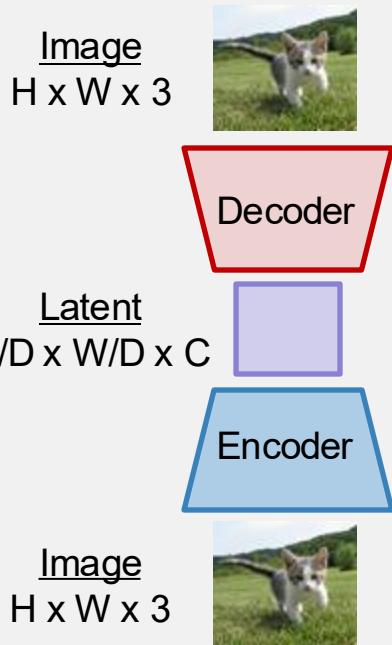


Train **diffusion model** to remove noise from **latents**
(**Encoder** is frozen)

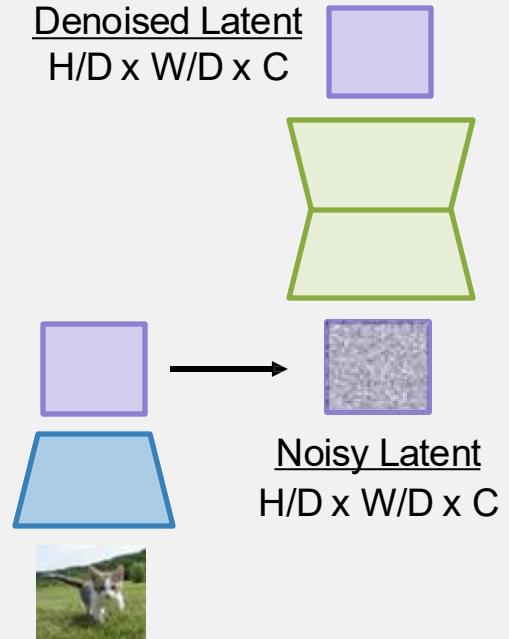


Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**



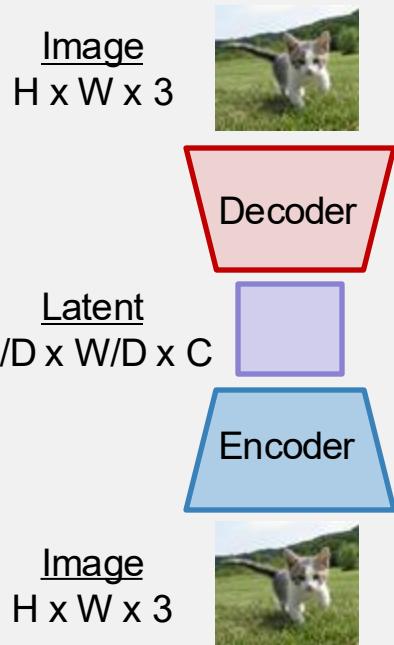
Train **diffusion model** to remove noise from **latents**
(**Encoder** is frozen)



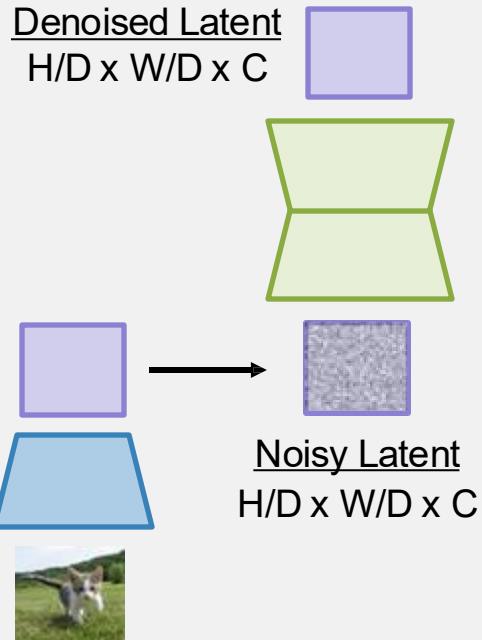
After training:

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**



Train **diffusion model** to remove noise from **latents**
(**Encoder** is frozen)

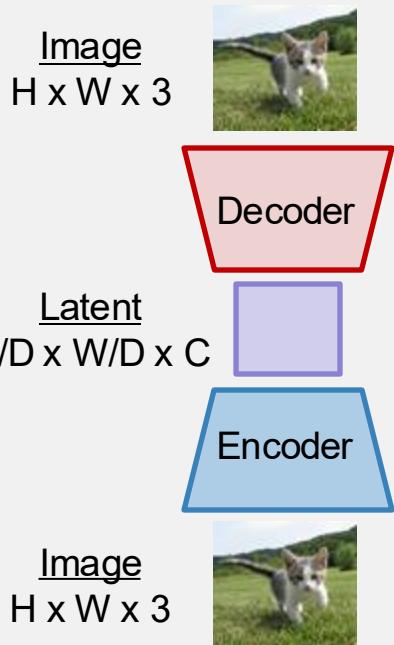


After training:
Sample random **latent**

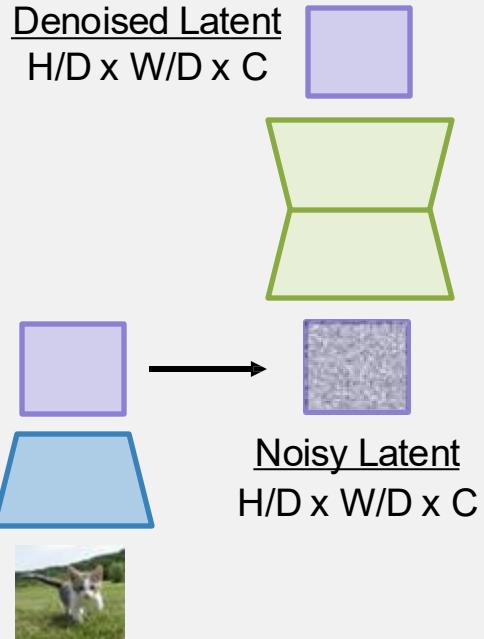


Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**



Train **diffusion model** to remove noise from **latents**
(**Encoder** is frozen)



After training:

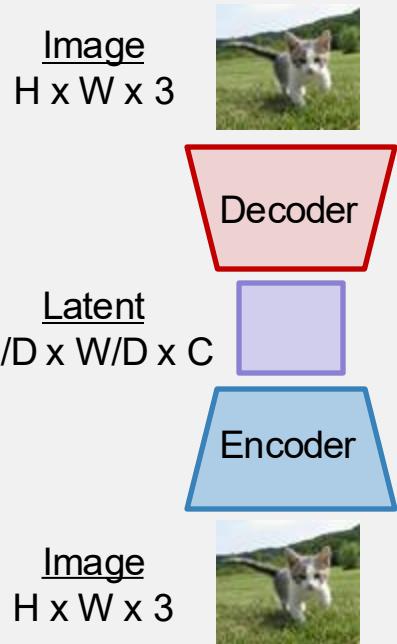
Sample random **latent**

Iteratively apply **diffusion model** to remove noise

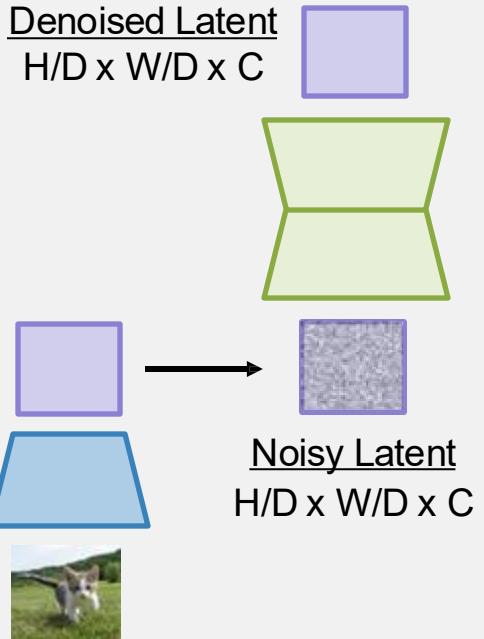


Latent Diffusion Models (LDMs)

Train **encoder + decoder** to convert images to **latents**



Train **diffusion model** to remove noise from **latents**
(**Encoder** is frozen)



After training:

Sample random **latent**

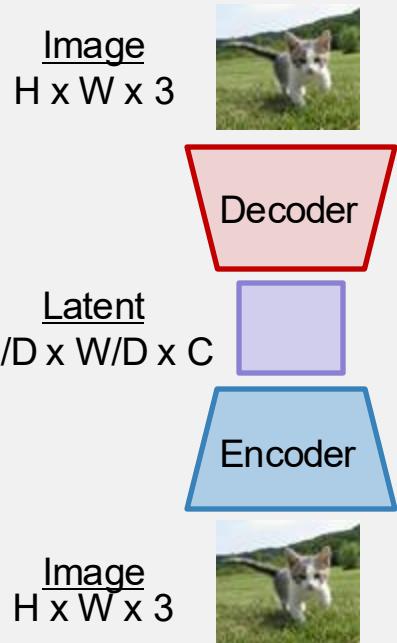
Iteratively apply **diffusion model** to remove noise

run **decoder** to get **image**

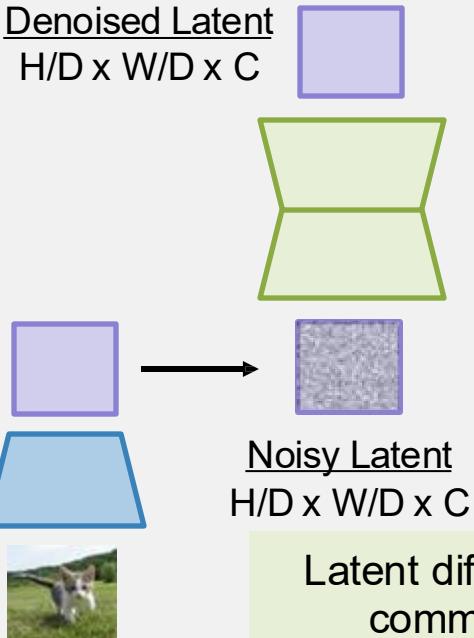


Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**



Train **diffusion model** to remove noise from **latents**
(**Encoder** is frozen)

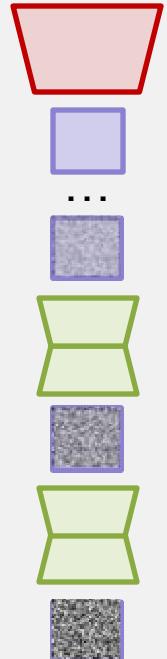


After training:

Sample random **latent**

Iteratively apply **diffusion model** to remove noise

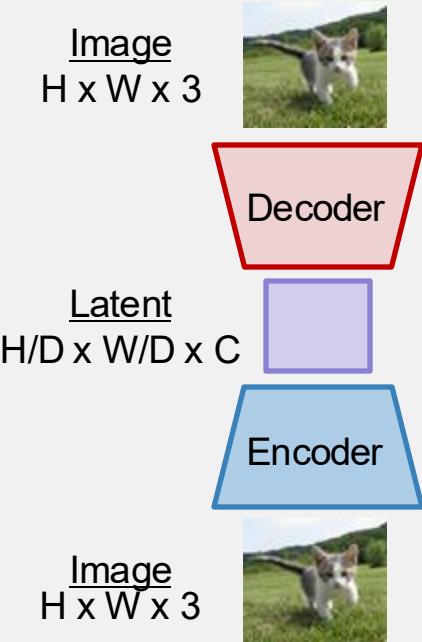
run **decoder** to get **image**



Latent diffusion is the most common form today

Latent Diffusion Models (LDMs)

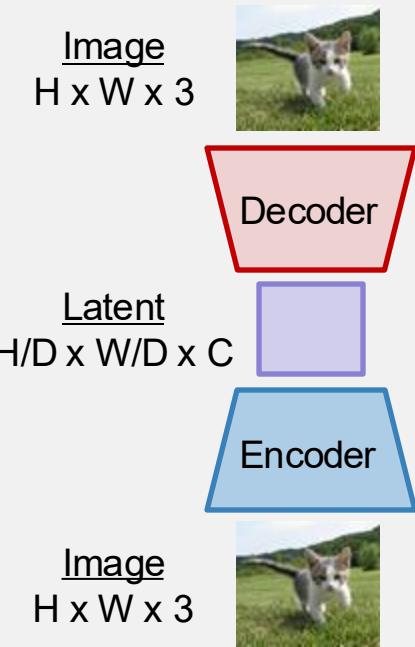
Train **encoder** + **decoder** to
convert images to **latents**



How do we train the
encoder+decoder?

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**

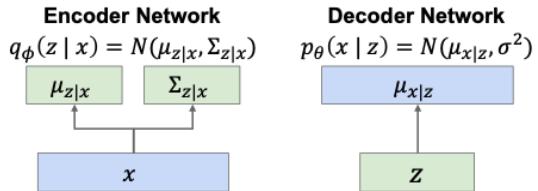


How do we train the encoder+decoder?

Solution: It's a VAE!
Typically with very small KL prior weight

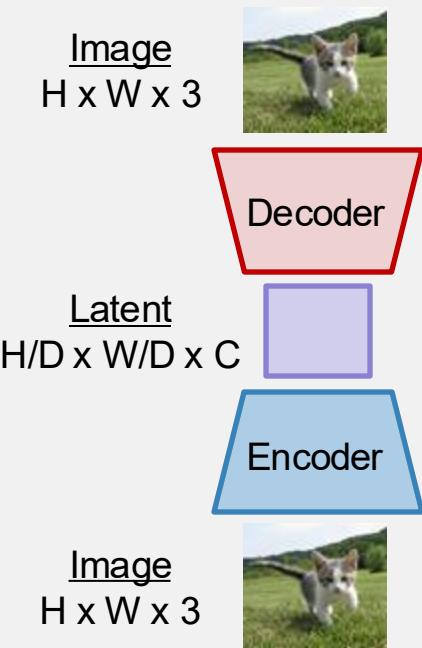
Recall: VAE

$$\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$



Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**



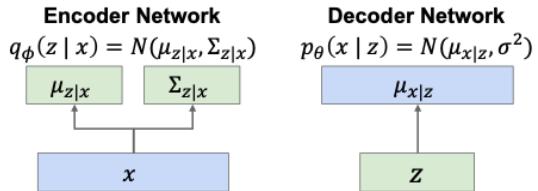
How do we train the encoder+decoder?

Solution: It's a VAE!
Typically with very small KL prior weight

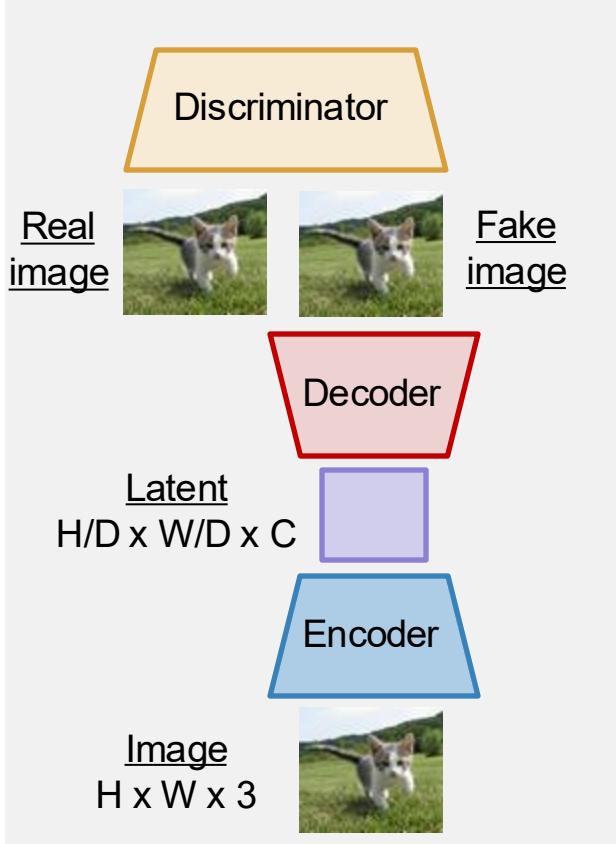
Problem: Decoder outputs often blurry

Recall: VAE

$$\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$



Latent Diffusion Models (LDMs)



How do we train the encoder+decoder?

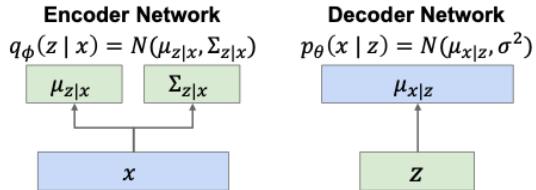
Solution: It's a VAE!
Typically with very small KL prior weight

Problem: Decoder outputs often blurry

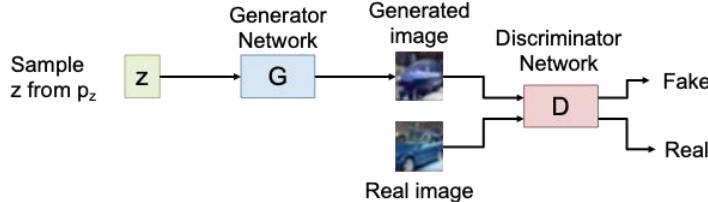
Solution: Add a discriminator!

Recall: VAE

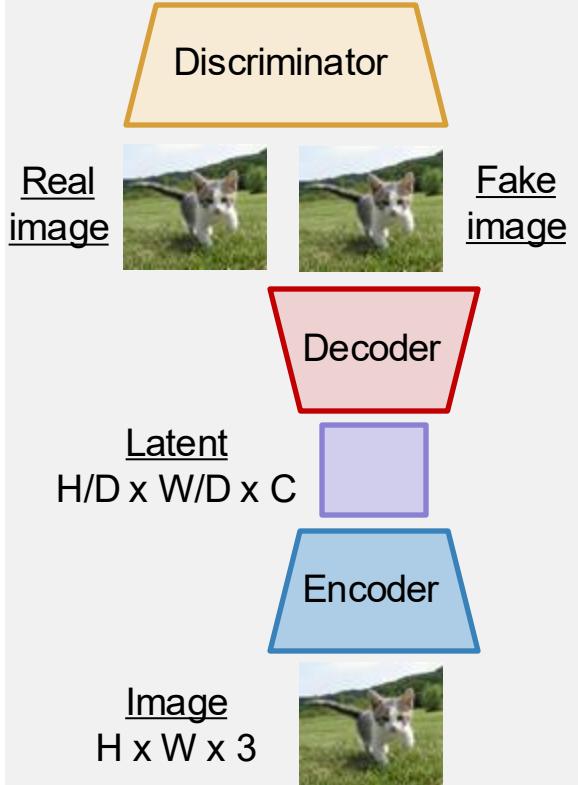
$$\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$



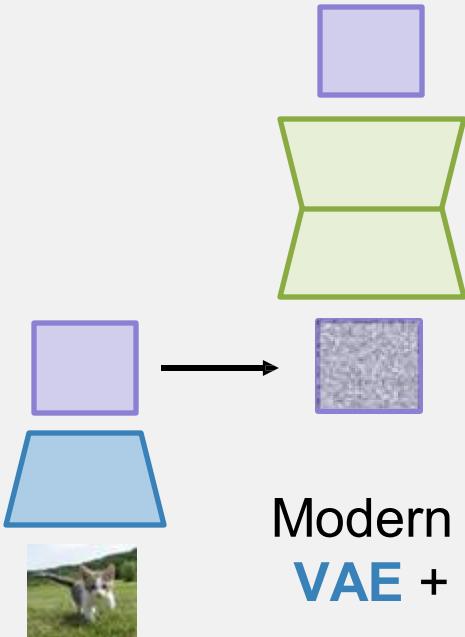
Recall: GAN



Latent Diffusion Models (LDMs)



Train **diffusion model** to remove noise from **latents**
(**Encoder** is frozen)



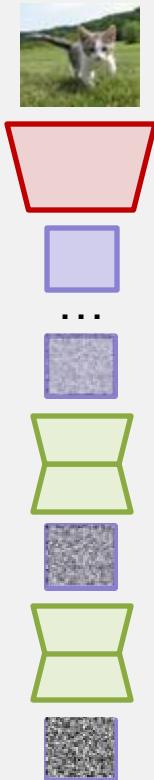
Modern LDM pipelines use
VAE + **GAN** + **diffusion!**

After training:

Sample random
latent

Iteratively apply
diffusion model
to remove noise

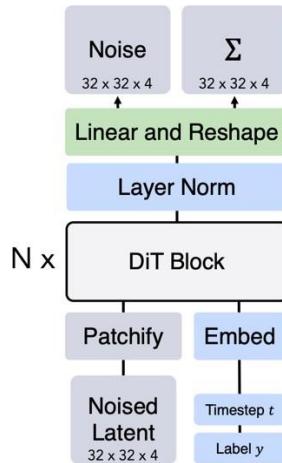
run **decoder** to
get **image**



Diffusion Transformer (DiT)

Diffusion uses standard Transformer blocks!

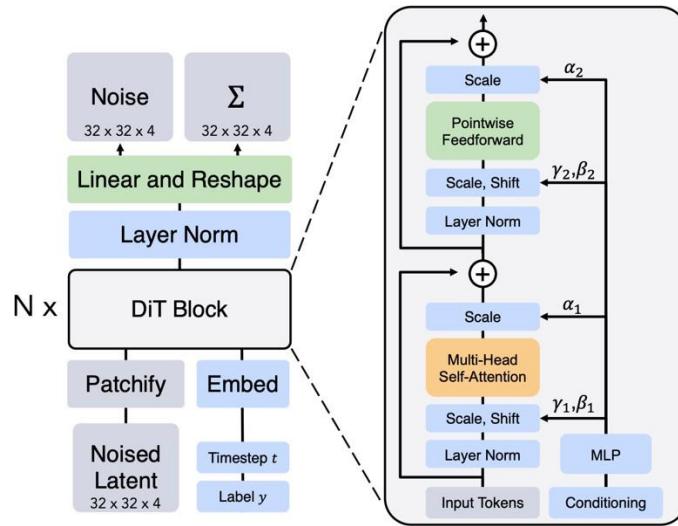
Main question: How to inject conditioning (timestep t , text, ...)



Diffusion Transformer (DiT)

Diffusion uses standard Transformer blocks!

Main question: How to inject conditioning (timestep t, text, ...)

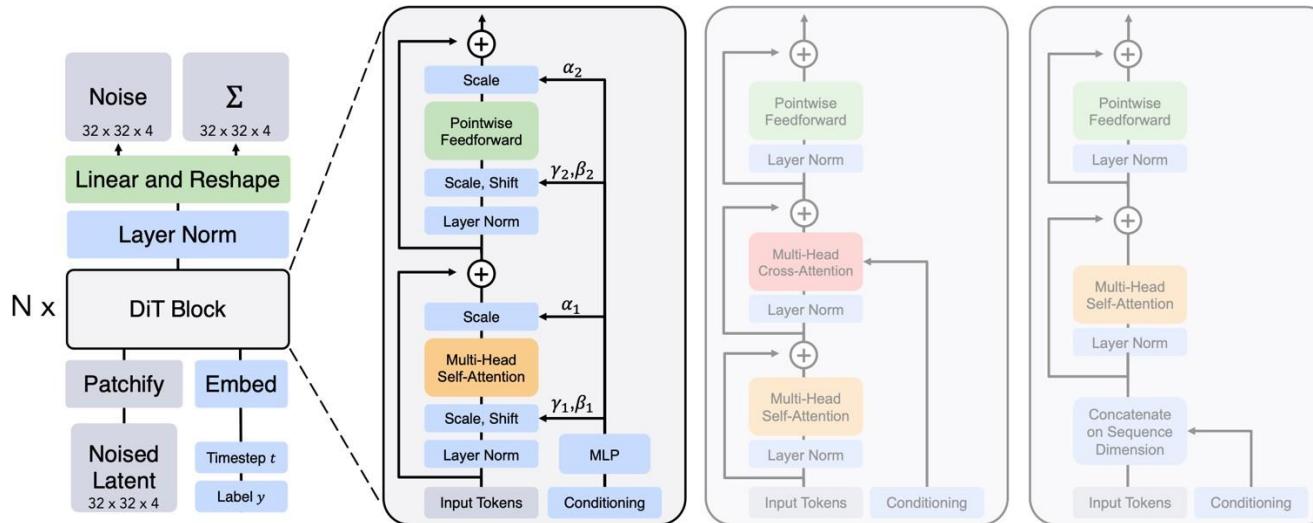


Predict scale/shift:
Most common for
diffusion timestep t

Diffusion Transformer (DiT)

Diffusion uses standard Transformer blocks!

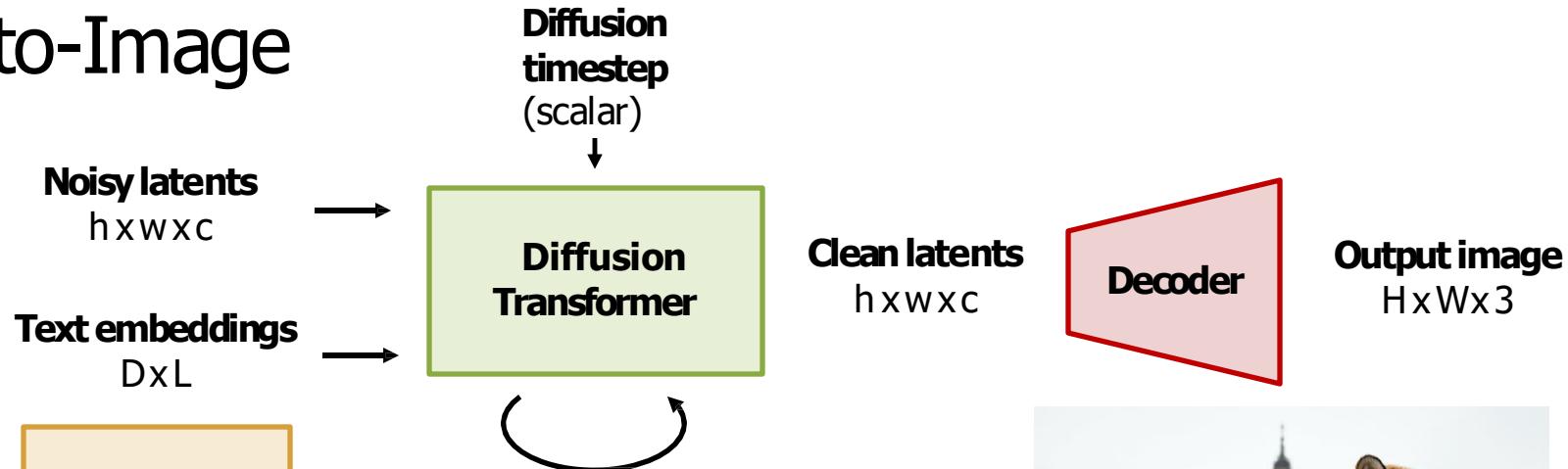
Main question: How to inject conditioning (timestep t, text, ...)



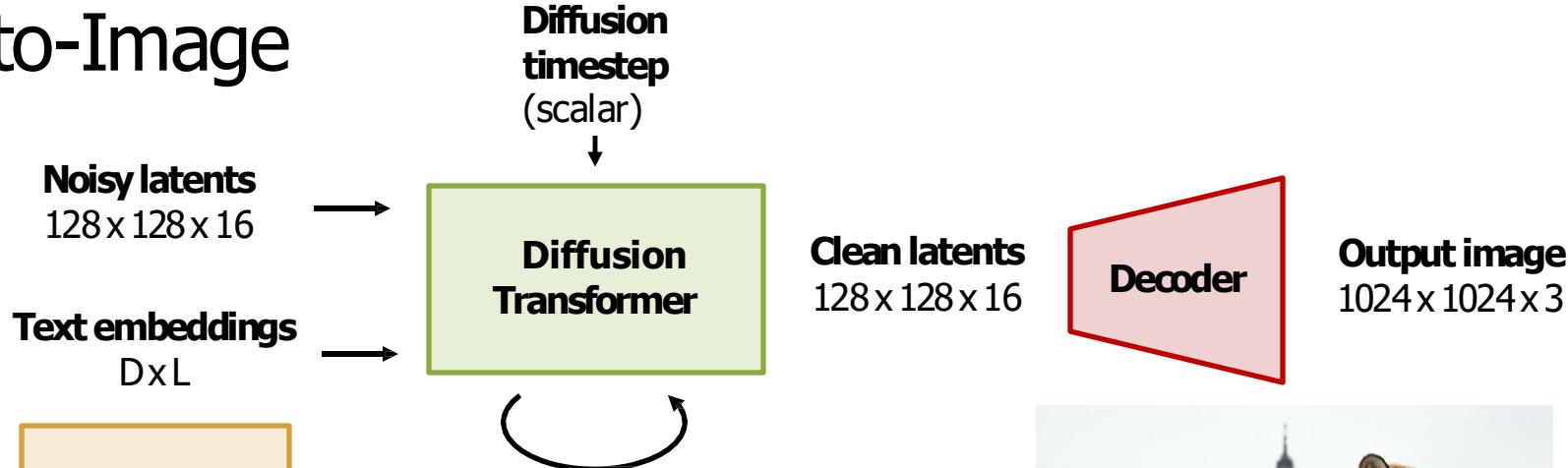
Predict scale/shift:
Most common for diffusion timestep t

Cross-Attention / Joint Attention:
Common for text, image, etc conditioning

Text-to-Image



Text-to-Image



Example: FLUX.1 [dev]

Text Encoder: T5 +CLIP

Encoder/Decoder: 8x8 downsampling

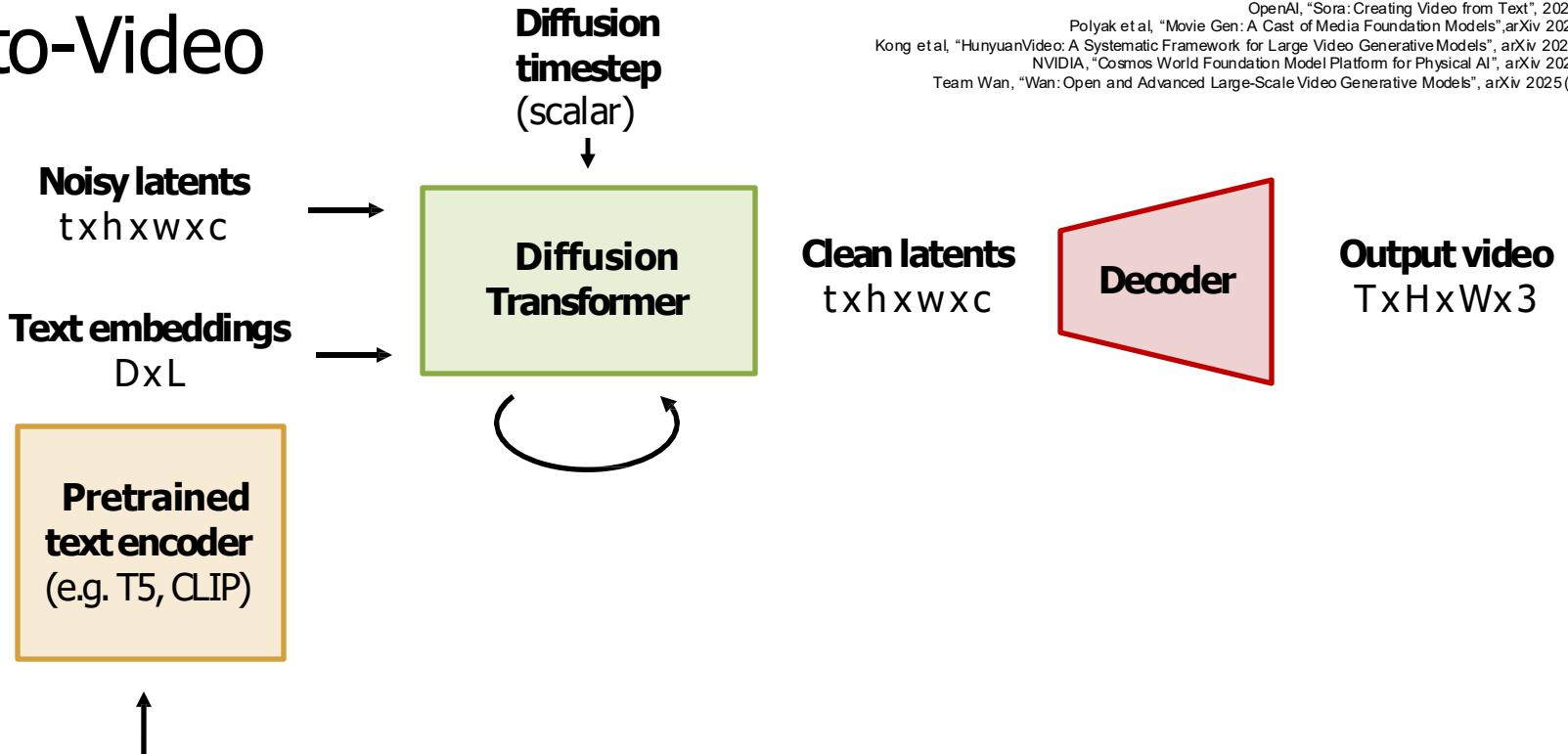
Diffusion model: 12B parameter model

2×2 patchify $\Rightarrow 64 \times 64 = 1024$ image tokens



<https://github.com/black-forest-labs/flux>

Text-to-Video



Gupta et al, "Photorealistic Video Generation with Diffusion Models", arXiv 2023 (Dec)

OpenAI, "Sora: Creating Video from Text", 2024 (Feb)

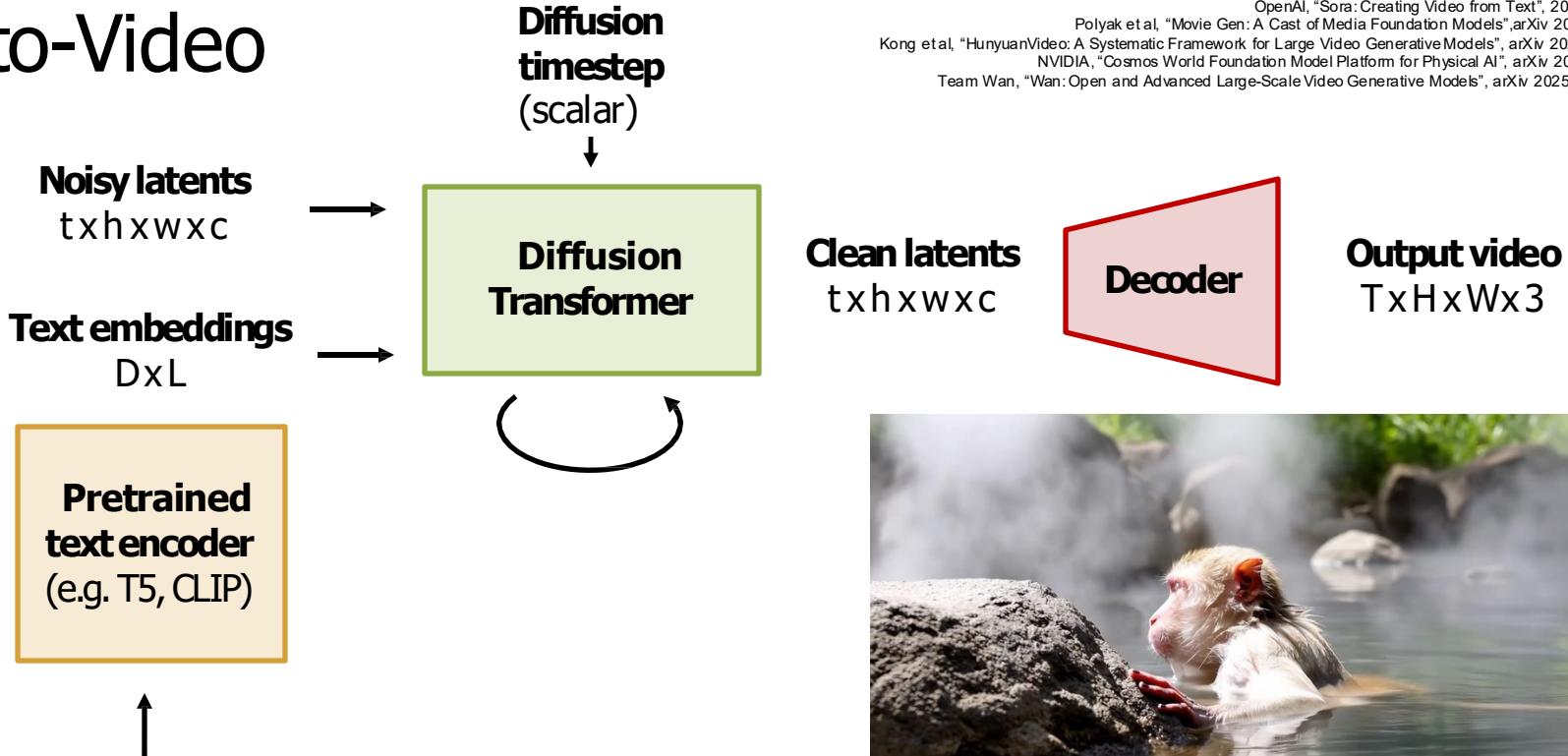
Polyak et al, "Movie Gen: A Cast of Media Foundation Models", arXiv 2024 (Oct)

Kong et al, "HunyuanVideo: A Systematic Framework for Large Video Generative Models", arXiv 2024 (Dec)

NVIDIA, "Cosmos World Foundation Model Platform for Physical AI", arXiv 2025 (Jan)

Team Wan, "Wan: Open and Advanced Large-Scale Video Generative Models", arXiv 2025 (March)

Text-to-Video

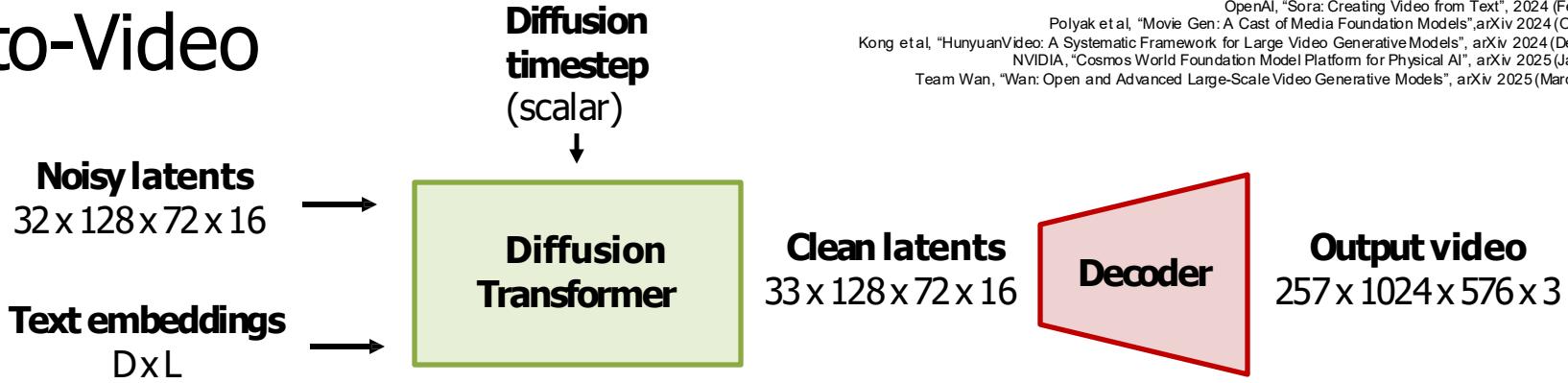


Gupta et al, "Photorealistic Video Generation with Diffusion Models", arXiv 2023 (Dec)
OpenAI, "Sora: Creating Video from Text", 2024 (Feb)
Polyak et al, "Movie Gen: A Cast of Media Foundation Models", arXiv 2024 (Oct)
Kong et al, "HunyuanVideo: A Systematic Framework for Large Video Generative Models", arXiv 2024 (Dec)
NVIDIA, "Cosmos World Foundation Model Platform for Physical AI", arXiv 2025 (Jan)
Team Wan, "Wan: Open and Advanced Large-Scale Video Generative Models", arXiv 2025 (March)



Video from Meta Movie Gen
(<https://ai.meta.com/research/movie-gen/>)

Text-to-Video



Example: Meta MovieGen

Text Encoder: UL2, ByT5, MetaCLIP

Encoder/Decoder: 8x8x8 downsample

Diffusion model: 30B param DiT

1x2x2 patchify => 76K tokens

Text Prompt

A red-faced monkey with white fur is bathing in a natural hot spring. The monkey is playing in the water with a miniature sail ship in front of it, made of wood with a white sail and a small rudder. The hot spring is surrounded by lush greenery, with rocks and trees.

Gupta et al, "Photorealistic Video Generation with Diffusion Models", arXiv 2023 (Dec)

OpenAI, "Sora: Creating Video from Text", 2024 (Feb)

Polyak et al, "Movie Gen: A Cast of Media Foundation Models", arXiv 2024 (Oct)

Kong et al, "HunyuanVideo: A Systematic Framework for Large Video Generative Models", arXiv 2024 (Dec)

NVIDIA, "Cosmos World Foundation Model Platform for Physical AI", arXiv 2025 (Jan)

Team Wan, "Wan: Open and Advanced Large-Scale Video Generative Models", arXiv 2025 (March)

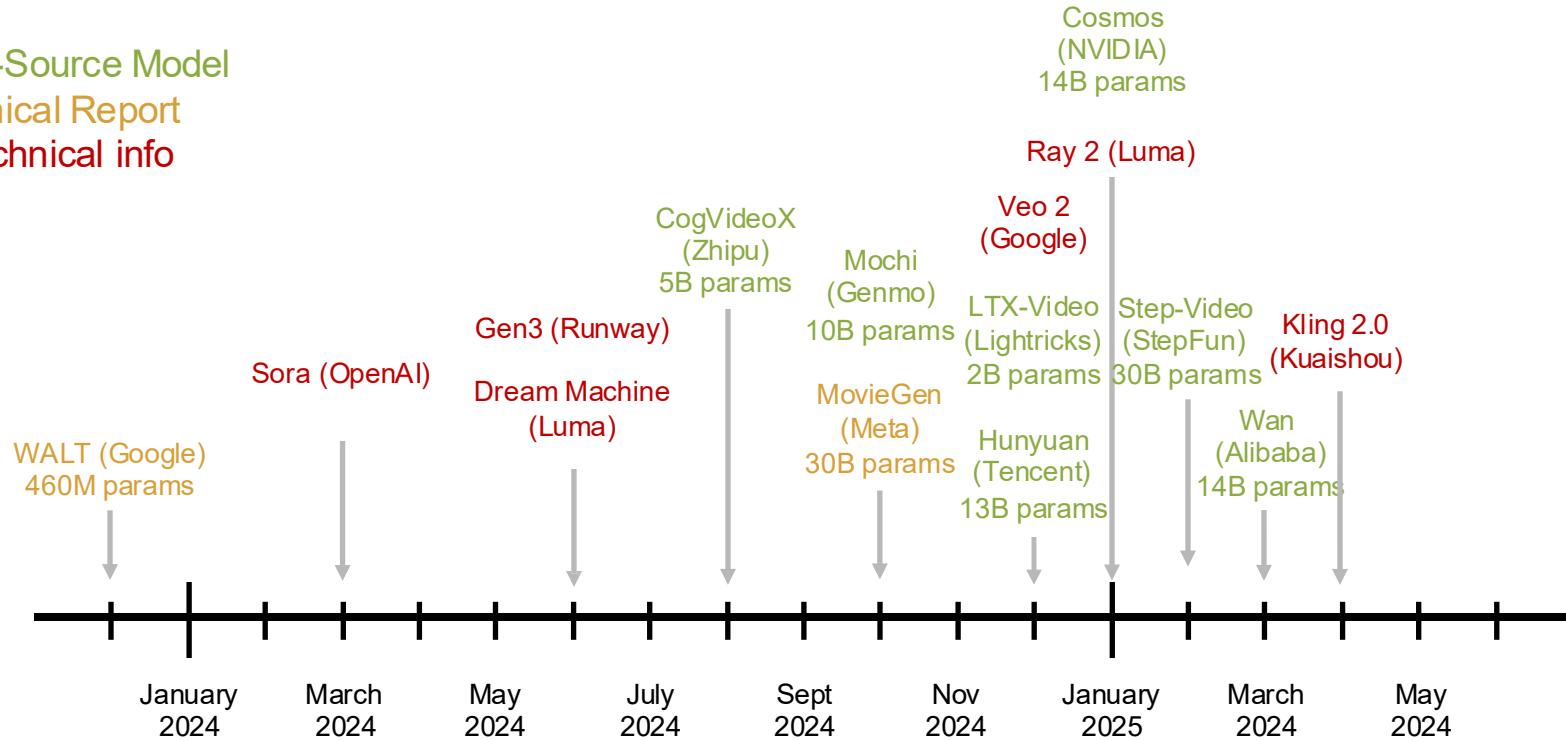


The Era of Video Diffusion Models

Open-Source Model

Technical Report

No technical info



Gupta et al., "Photorealistic Video Generation with Diffusion Models", arXiv 2023 (Oed)

Qian et al., "Cross-Task Cross-Model Video Generation", arXiv 2024 (Feb)

Polyak et al., "MovieGen: A Toolkit of Models for Generating and Editing Videos", arXiv 2024 (Mar)

Kong et al., "HunyuanVideo: A Systematic Framework for Large-Video Generative Models", arXiv 2024 (Jun)

NVIDIA, "Cosmo: A World's First Foundation Model Platform for Physical AI", arXiv 2025 (Jan)

Team Wan, "Wan: Open and Adaptive Large-Scale Video Generative Model", arXiv 2025 (Mar)

Diffusion Distillation

During sampling we need to run the diffusion model many times (~30 – 50 for rectified flow)

This is really slow!

After training:

Sample random **latent**



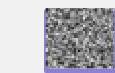
...



Iteratively apply **diffusion model** to remove noise



run **decoder** to get **image**



Diffusion Distillation

During sampling we need to run the diffusion model many times (~30 – 50 for rectified flow)

This is really slow!

Solution: **distillation** algorithms reduce the number of steps (sometimes all the way to 1), can also bake in CFG

After training:

Sample random
latent



Iteratively apply
diffusion model
to remove noise



...



run **decoder** to
get **image**



Salimans and Ho, "Progressive Distillation for Fast Sampling of Diffusion Models", ICLR 2022
Song et al, "Consistency Models", ICML 2023
Sauer et al, "Adversarial Diffusion Distillation", ECCV 2024
Sauer et al, "Fast High-Resolution Image Synthesis with Latent Adversarial Diffusion Distillation", arXiv 2024
Lu and Song, "Simplifying, Stabilizing and Scaling Consistency Models", ICLR 2025
Salimans et al, "Multistep Distillation of Diffusion Models via Moment Matching", NeurIPS 2025

Generalized Diffusion

Rectified Flow

Sample $x \sim p_{data}$, $z \sim p_{noise}$

Sample $t \sim p_t$

Set $x_t = (1 - t)x + tz$

Set $v_{gt} = z - x$

Compute $v_{pred} = f_\theta(x_t, t)$

Compute loss $\|v_{gt} - v_{pred}\|_2^2$

Generalized Diffusion

Rectified Flow

Sample $x \sim p_{data}$, $z \sim p_{noise}$

Sample $t \sim p_t$

Set $x_t = (1 - t)x + tz$

Set $v_{gt} = z - x$

Compute $v_{pred} = f_\theta(x_t, t)$

Compute loss $\|v_{gt} - v_{pred}\|_2^2$



Generalized Diffusion

Sample $x \sim p_{data}$, $z \sim p_{noise}$

Sample $\textcolor{blue}{t} \sim p_t$

Set $x_t = \textcolor{green}{a}(\textcolor{blue}{t})x + \textcolor{blue}{b}(\textcolor{blue}{t})z$

Set $y_{gt} = \textcolor{red}{c}(\textcolor{blue}{t})x + \textcolor{brown}{d}(\textcolor{blue}{t})z$

Compute $y_{pred} = f_\theta(x_t, \textcolor{blue}{t})$

Compute loss $\|y_{gt} - y_{pred}\|_2^2$

Generalized Diffusion

Rectified Flow

Sample $x \sim p_{data}$, $z \sim p_{noise}$

Sample $t \sim p_t$

Set $x_t = (1 - t)x + tz$

Set $v_{gt} = z - x$

Compute $v_{pred} = f_\theta(x_t, t)$

Compute loss $\|v_{gt} - v_{pred}\|_2^2$

$$a(t) = 1 - t$$

$$b(t) = t$$

$$c(t) = -1$$

$$d(t) = 1$$



Generalized Diffusion

Sample $x \sim p_{data}$, $z \sim p_{noise}$

Sample $t \sim p_t$

Set $x_t = a(t)x + b(t)z$

Set $y_{gt} = c(t)x + d(t)z$

Compute $y_{pred} = f_\theta(x_t, t)$

Compute loss $\|y_{gt} - y_{pred}\|_2^2$

Generalized Diffusion

Variance Preserving (VP)

$$a(t) = \sqrt{\sigma(t)}$$

$$b(t) = \sqrt{1 - \sigma(t)}$$

If x and z are independent and variance=1, then x_t also has variance=1

Generalized Diffusion

Sample $x \sim p_{data}$, $z \sim p_{noise}$

Sample $t \sim p_t$

Set $x_t = a(t)x + b(t)z$

Set $y_{gt} = c(t)x + d(t)z$

Compute $y_{pred} = f_\theta(x_t, t)$

Compute loss $\|y_{gt} - y_{pred}\|_2^2$

Generalized Diffusion

Variance Exploding (VE)

$$\mathbf{a}(\mathbf{t}) = 1$$

$$\mathbf{b}(\mathbf{t}) = \sigma(t)$$

$\sigma(1)$ Needs to be big enough
to drown out all signal in x

Generalized Diffusion

Sample $x \sim p_{data}$, $z \sim p_{noise}$

Sample $\mathbf{t} \sim p_t$

Set $x_t = \mathbf{a}(\mathbf{t})x + \mathbf{b}(\mathbf{t})z$

Set $y_{gt} = \mathbf{c}(\mathbf{t})x + \mathbf{d}(\mathbf{t})z$

Compute $y_{pred} = f_\theta(x_t, \mathbf{t})$

Compute loss $\|y_{gt} - y_{pred}\|_2^2$

Generalized Diffusion

x-prediction

$$y_{gt} = x \quad [\mathbf{c}(\mathbf{t}) = 1; \mathbf{d}(\mathbf{t}) = 0]$$

φ prediction

$$y_{gt} = z \quad [\mathbf{c}(\mathbf{t}) = 0; \mathbf{d}(\mathbf{t}) = 1]$$

v-prediction

$$y_{gt} = \mathbf{b}(\mathbf{t})z - \mathbf{a}(\mathbf{t})x \quad [\mathbf{c}(\mathbf{t}) = \mathbf{b}(\mathbf{t}); \mathbf{d}(\mathbf{t}) = -\mathbf{a}(\mathbf{t})]$$

Generalized Diffusion

Sample $x \sim p_{data}, z \sim p_{noise}$

Sample $\mathbf{t} \sim p_t$

Set $x_t = \mathbf{a}(\mathbf{t})x + \mathbf{b}(\mathbf{t})z$

Set $y_{gt} = \mathbf{c}(\mathbf{t})x + \mathbf{d}(\mathbf{t})z$

Compute $y_{pred} = f_\theta(x_t, \mathbf{t})$

Compute loss $\|y_{gt} - y_{pred}\|_2^2$

Generalized Diffusion

How do we choose these functions?

Usually through some **mathematical formalism**

Generalized Diffusion

Sample $x \sim p_{data}$, $z \sim p_{noise}$

Sample $\mathbf{t} \sim p_t$

Set $x_t = \mathbf{a}(\mathbf{t})x + \mathbf{b}(\mathbf{t})z$

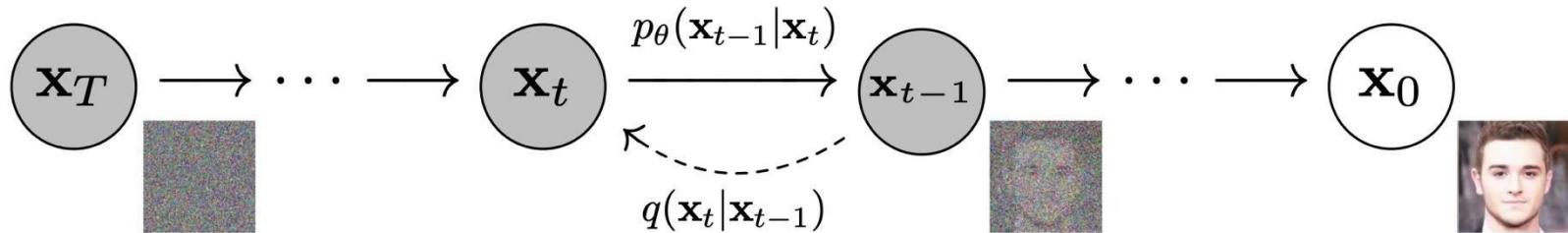
Set $y_{gt} = \mathbf{c}(\mathbf{t})x + \mathbf{d}(\mathbf{t})z$

Compute $y_{pred} = f_\theta(x_t, \mathbf{t})$

Compute loss $\|y_{gt} - y_{pred}\|_2^2$

Diffusion is a Latent Variable Model

We know the forward process: Add Gaussian noise



Learn a network to approximate the backward process

Optimize variational lower bound (same as VAE)

Diffusion Learns the Score Function

For any distribution $p(x)$ over $x \in \mathbb{R}^N$ the **score function**

$$s: \mathbb{R}^N \rightarrow \mathbb{R}^N \quad s(x) = \frac{\partial}{\partial x} \log p(x)$$

Is a vector field pointing toward areas of high probability density

Diffusion learns a neural network to approximate the score function of p_{data}

Diffusion Solves Stochastic Differential Equations

We can describe a continuous noising process as an SDE

$$dx = f(x, t)dt + g(t)d\omega$$

Gives a relationship between infinitesimal changes in data x , time t , and noise ω .

Diffusion learns a neural network to approximately solve this SDE

Perspectives on Diffusion

1. *Diffusion models are autoencoders*
2. *Diffusion models are deep latent variable models*
3. *Diffusion models predict the score function*
4. *Diffusion models solve reverse SDEs*
5. *Diffusion models are flow-based models*
6. *Diffusion models are recurrent neural networks*
7. *Diffusion models are autoregressive models*
8. *Diffusion models estimate expectations*

Great blog post by Sander Dieleman:

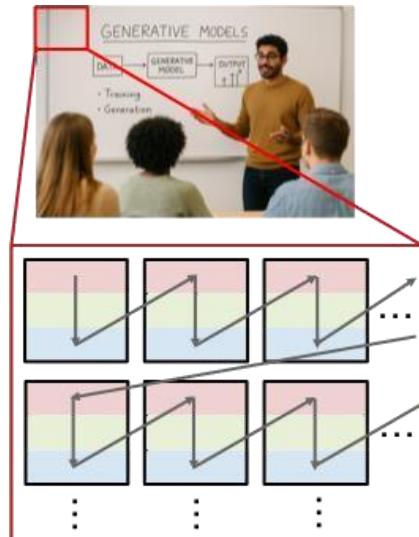
<https://sander.ai/2023/07/20/perspectives.html>

(All his blog posts are great)

Autoregressive Models Strike Back

Recall autoregressive models

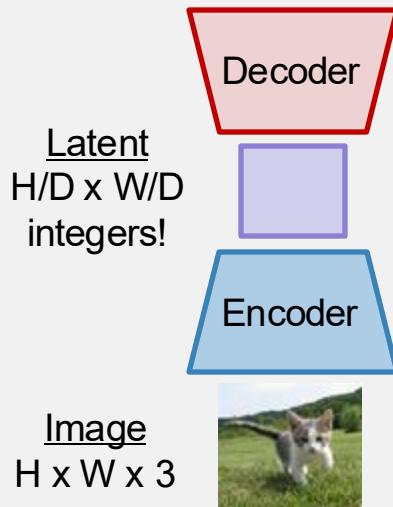
Too slow on raw pixels



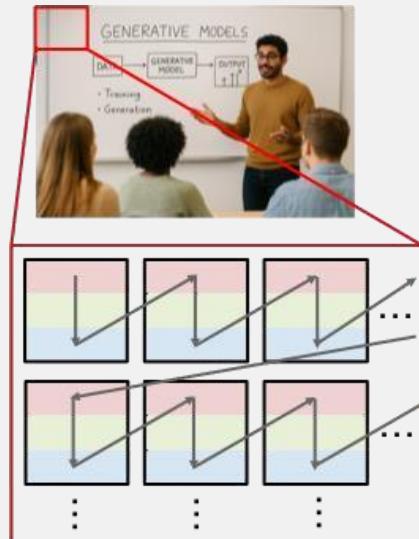
They work great on
(discrete) latents!

Autoregressive Models Strike Back

Train **encoder** + **decoder**
to convert images to
discrete latents



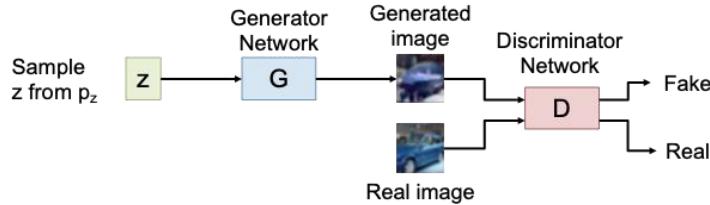
Train **autoregressive model**
to model sequences of
discrete latents



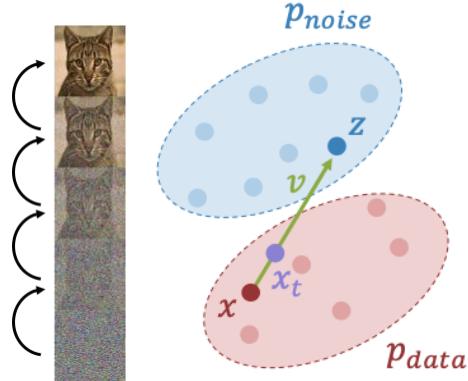
Sample discrete latents from
the **autoregressive model**,
pass to **decoder** to get an
image

Summary

Generative Adversarial Networks



Diffusion Models



Latent Diffusion Models

