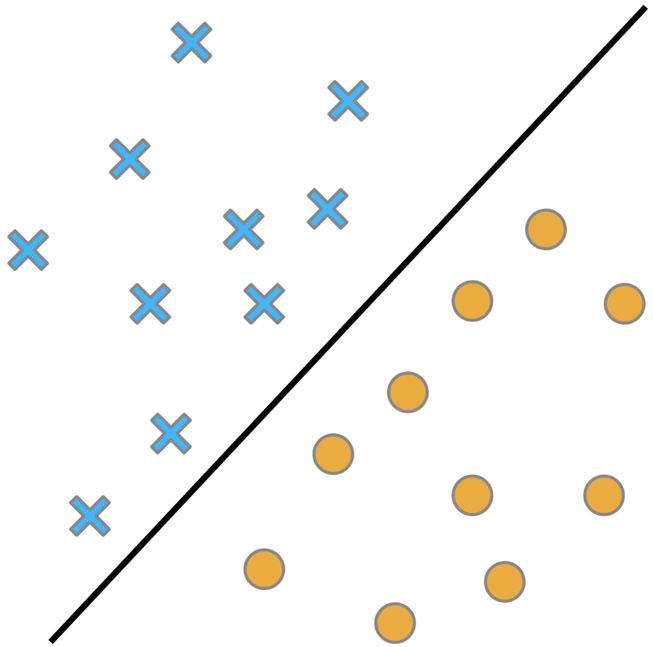
DS-GA 1003: Machine Learning

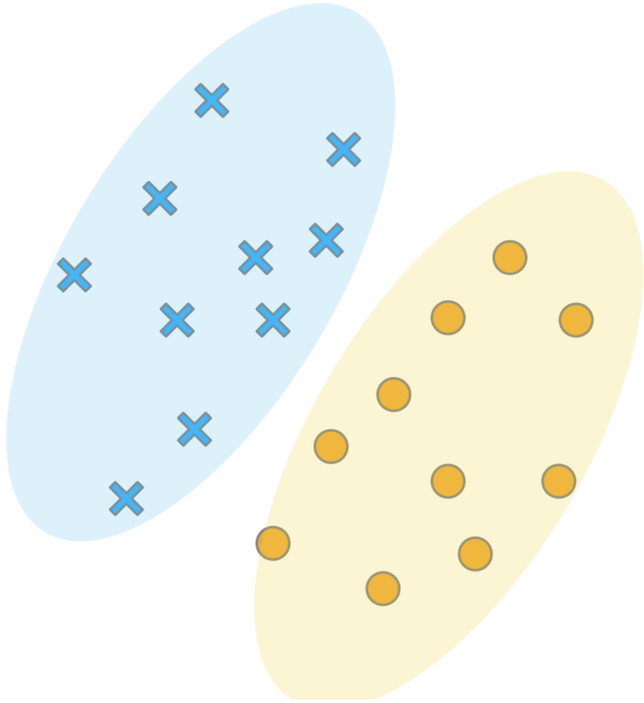
Lecture 11: Generative Models

Slides adapted from: Sergey Levine

Recall: Generative Models

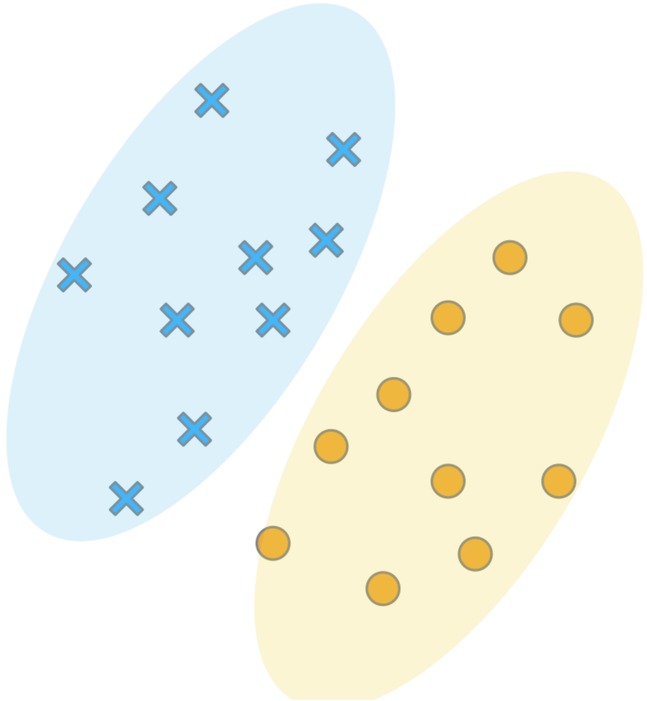


p(y | x)



p(x, y)

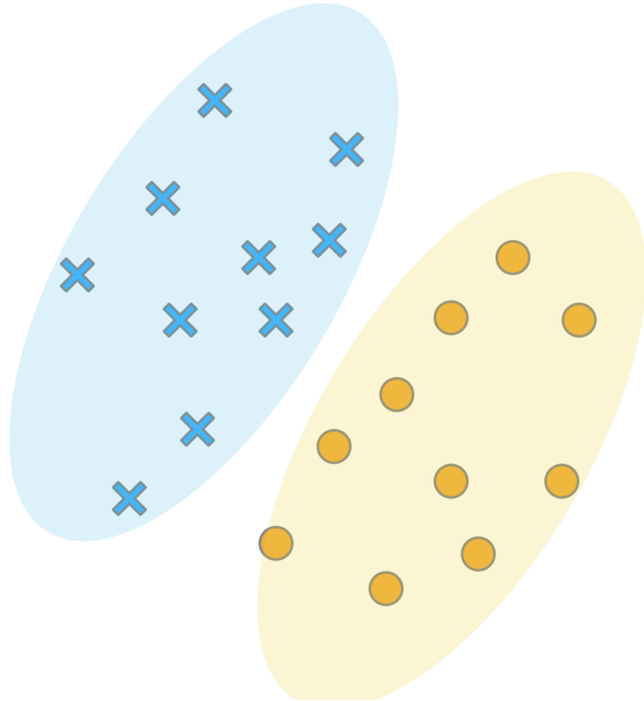
Recall: Generative Models



Today: p(x)

Recall: Generative Models

Why would we want to model p(x)?

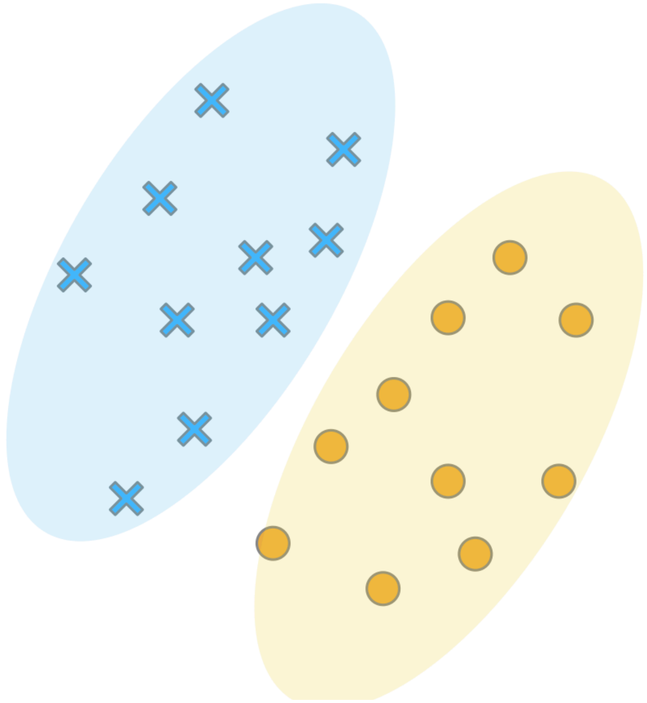


Today: p(x)

Recall: Generative Models

Why would we want to model p(x)?

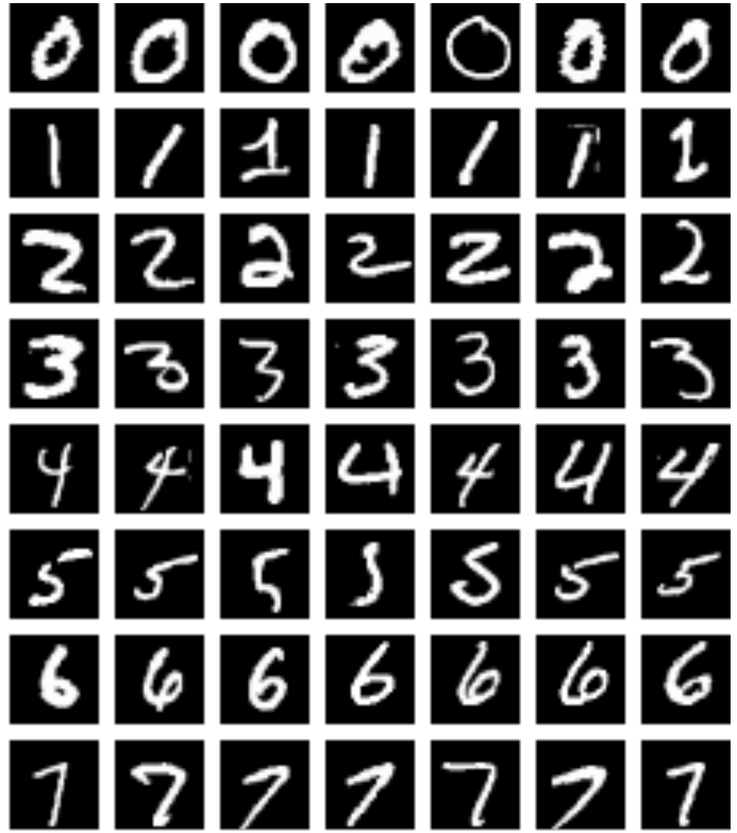
- Representation learning
- Could be useful for later fine-tuning/classification
- Density estimation, missing data imputation, etc.
- Compression
- Understanding latent structure in data
- Useful for generating new datapoints!



Today: p(x)

Recall: Generative Models

Why is modeling p(x) hard?

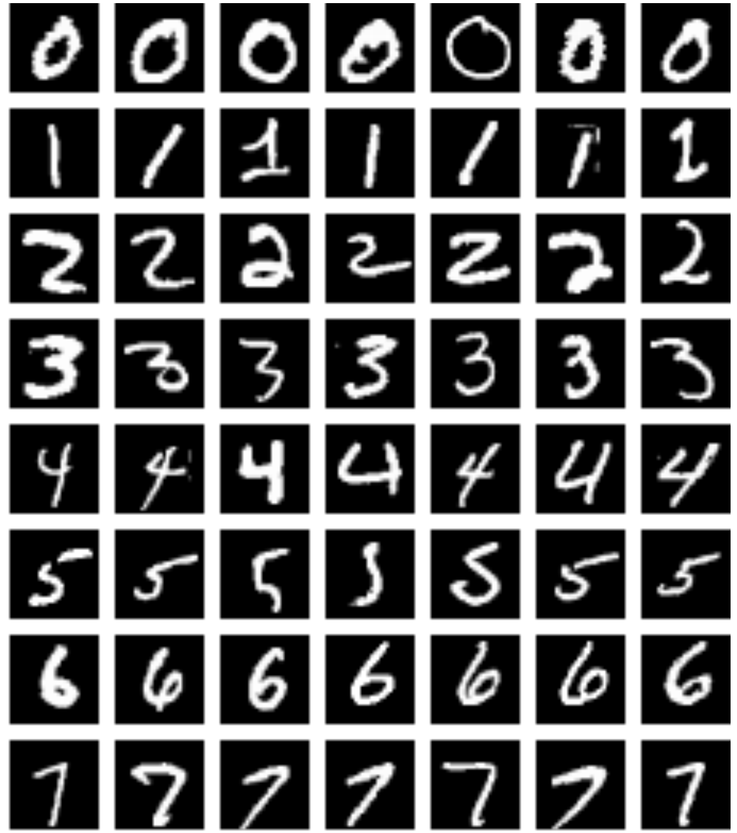


Today: p(x)

Recall: Generative Models

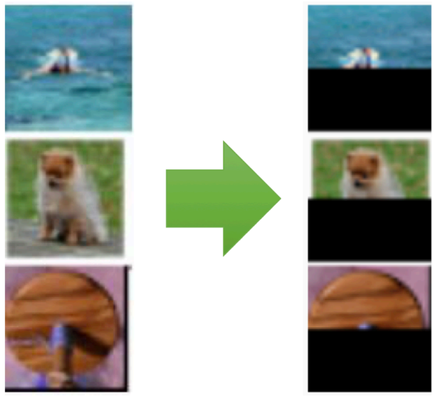
Why is modeling p(x) hard?

- High-dimensional output space: the vast majority of possible outputs are unacceptable!
- Difficult to evaluate; we don't just have a classification loss telling us which predictions are right/wrong anymore



Today: p(x)

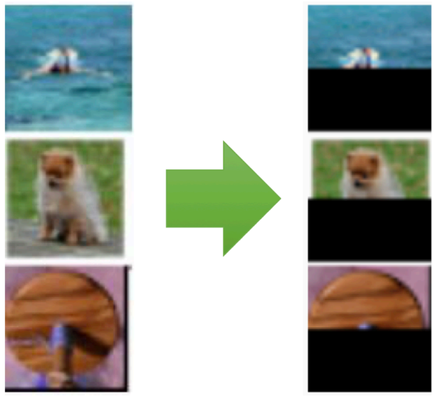
A motivating example: PixelCNN



Can we treat image modeling as a language modeling problem?

Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

A motivating example: PixelCNN

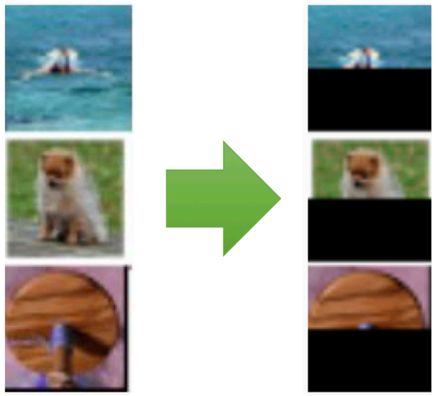


Can we treat image modeling as a language modeling problem?

We can mask out parts of an image and train a model to “fill in the blanks”

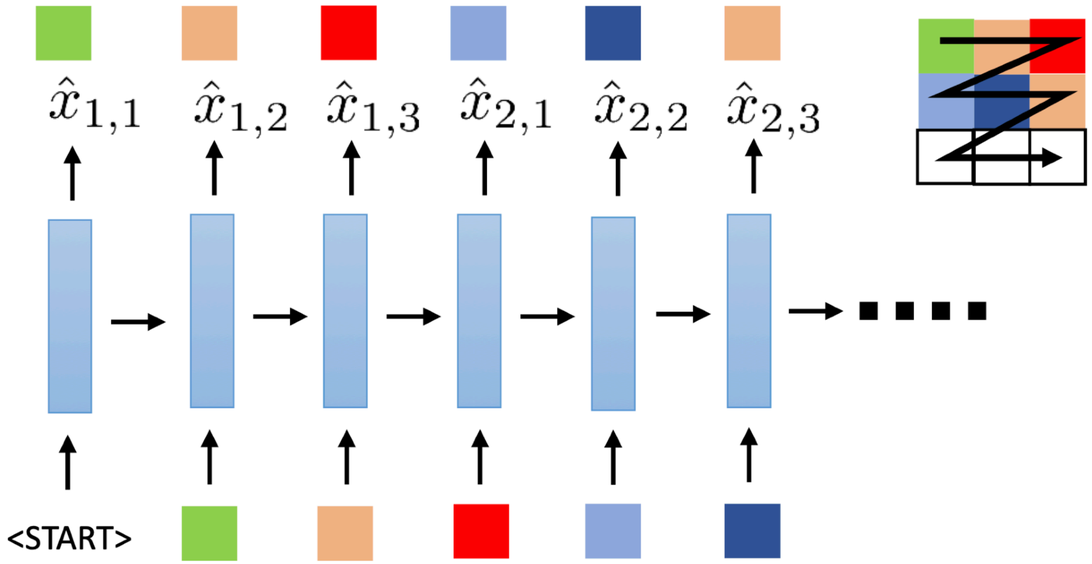
Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

A motivating example: PixelCNN



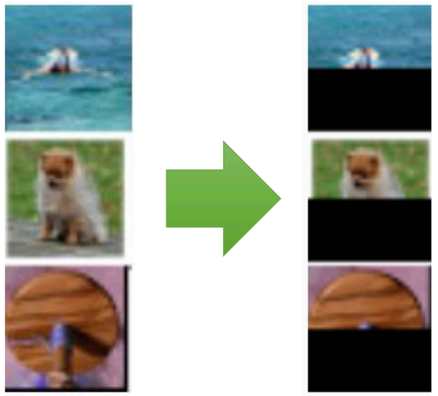
Can we treat image modeling as a language modeling problem?

We can mask out parts of an image and train a model to “fill in the blanks”



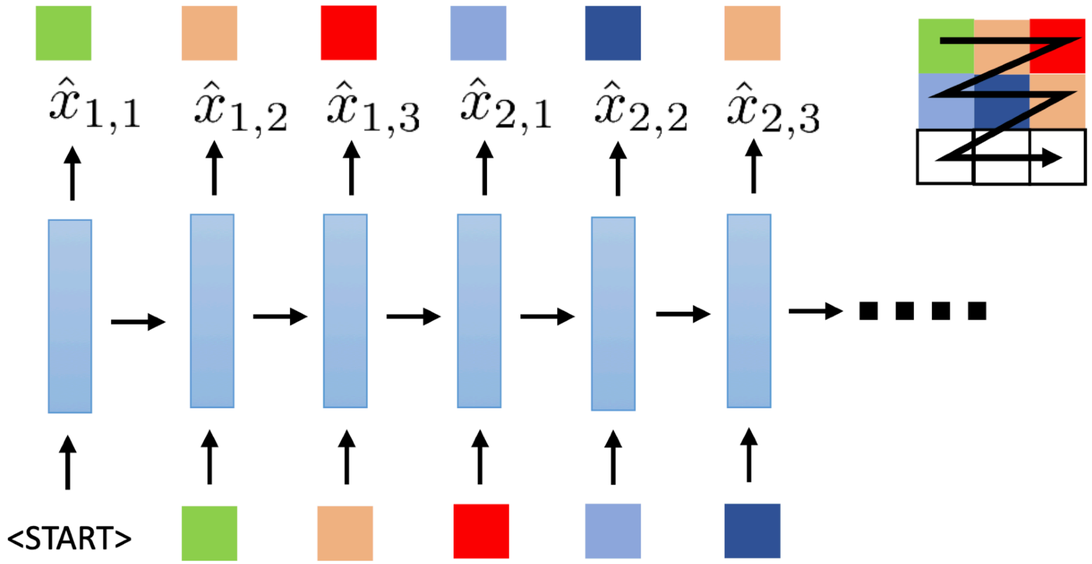
Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

A motivating example: PixelCNN



Can we treat image modeling as a language modeling problem?

We can mask out parts of an image and train a model to “fill in the blanks”

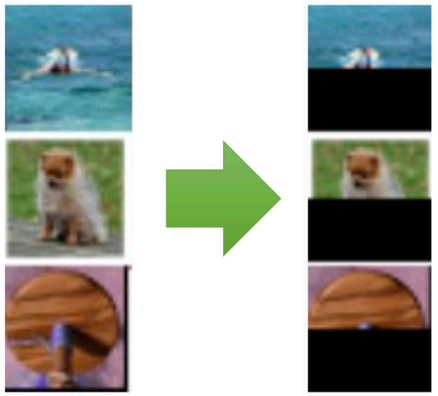


$$p(x) = p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_{1:2}) \cdot p(x_4 \mid x_{1:3}) \cdot p(x_5 \mid x_{1:4}) \cdots$$

(Given some arbitrary ordering of pixels $x_1, x_2, \ldots$)

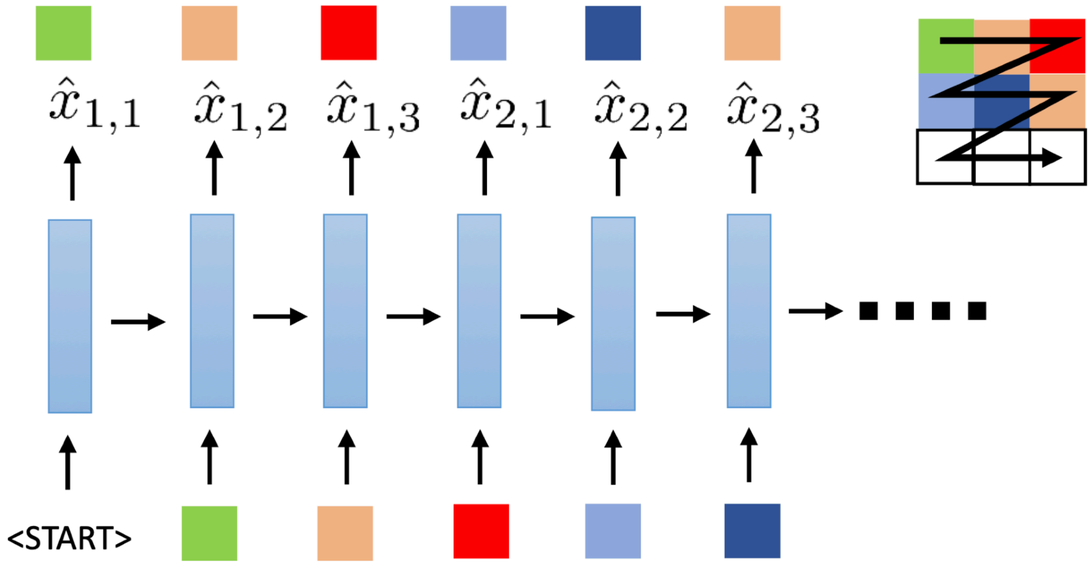
Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

A motivating example: PixelCNN



Can we treat image modeling as a language modeling problem?

We can mask out parts of an image and train a model to “fill in the blanks”



$$p(x) = p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_{1:2}) \cdot p(x_4 \mid x_{1:3}) \cdot p(x_5 \mid x_{1:4}) \cdots$$

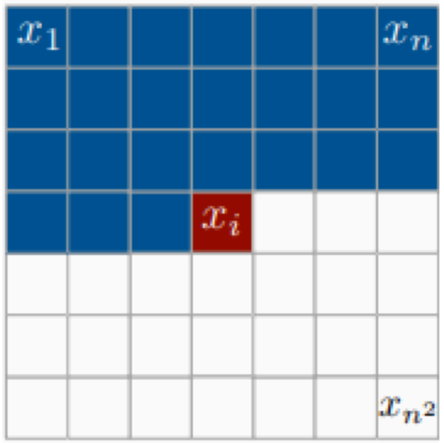
(Given some arbitrary ordering of pixels $x_1, x_2, \ldots$)

Need to figure out:

- How do we order the pixels?
- What model should we use?

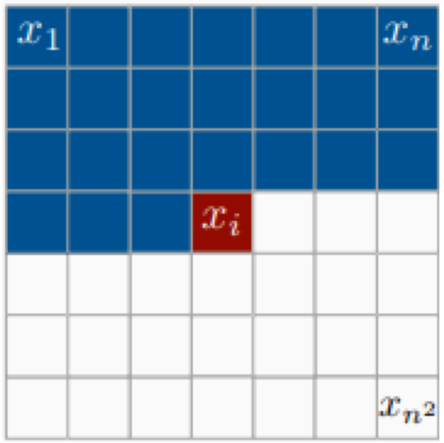
Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

PixelRNN



Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

PixelRNN

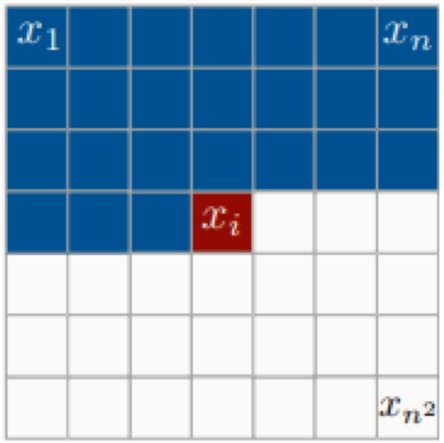


Pixels generated one at a time, left-to-right, top-to-bottom:

$$p(x) = \prod_{i=1}^{n^2} p(x_i \mid x_1, \ldots, x_{i-1})$$

Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

PixelRNN



Pixels generated one at a time, left-to-right, top-to-bottom:

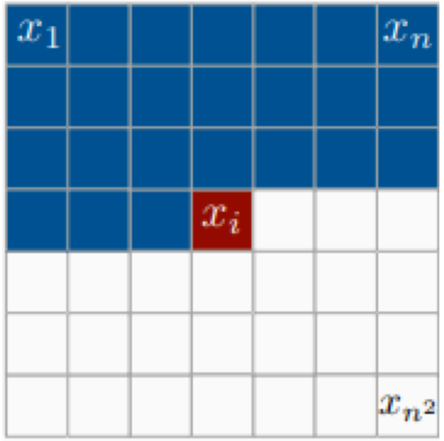
$$p(x) = \prod_{i=1}^{n^2} p(x_i \mid x_1, \ldots, x_{i-1})$$

Generate one color channel at a time:

$$p(x_{i,R} \mid x_{<i}) \cdot p(x_{i,G} \mid x_{<i}, x_{i,R}) \cdot p(x_{i,B} \mid x_{<i}, x_{i,R}, x_{i,G})$$

Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

PixelRNN



Pixels generated one at a time, left-to-right, top-to-bottom:

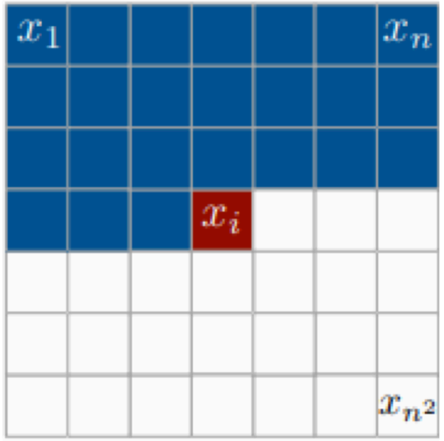
$$p(x) = \prod_{i=1}^{n^2} p(x_i \mid x_1, \ldots, x_{i-1})$$

Generate one color channel at a time:

$$p(x_{i,R} \mid x_{<i}) \cdot p(x_{i,G} \mid x_{<i}, x_{i,R}) \cdot p(x_{i,B} \mid x_{<i}, x_{i,R}, x_{i,G})$$

Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

PixelRNN



Pixels generated one at a time, left-to-right, top-to-bottom:

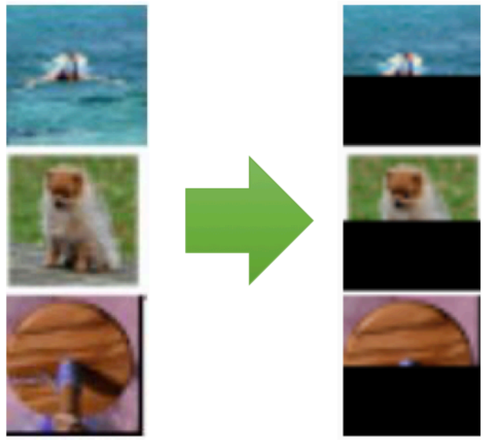
$$p(x) = \prod_{i=1}^{n^2} p(x_i \mid x_1, \ldots, x_{i-1})$$

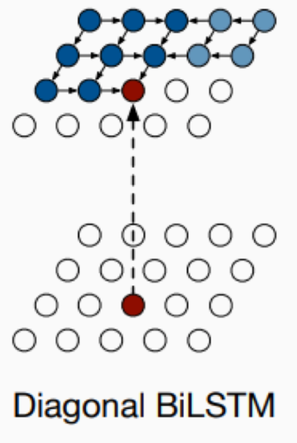
Generate one color channel at a time:

$$p(x_{i,R} \mid x_{<i}) \cdot p(x_{i,G} \mid x_{<i}, x_{i,R}) \cdot p(x_{i,B} \mid x_{<i}, x_{i,R}, x_{i,G})$$

Practical considerations:

- Autoregressive generation is slow
- Row-by-row LSTMs might struggle to capture spatial context
- Many practical improvements and better architectures are possible

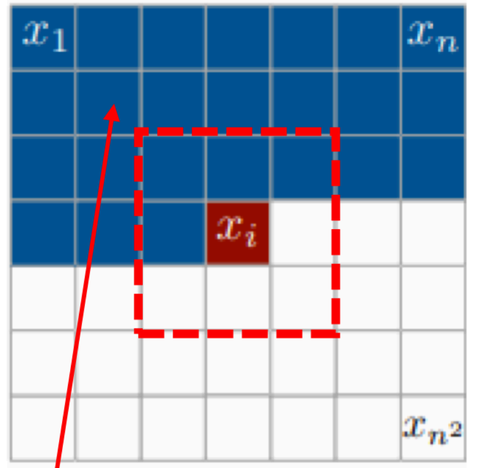




Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

PixelCNN

Idea: make this much faster by not building a full RNN over all pixels, but just using a convolution to determine the value of a pixel based on its neighborhood

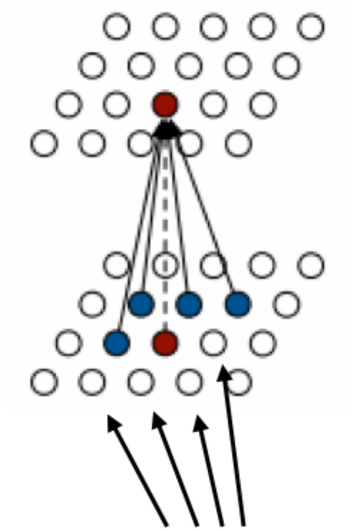


This pixel still influences x_i – why?

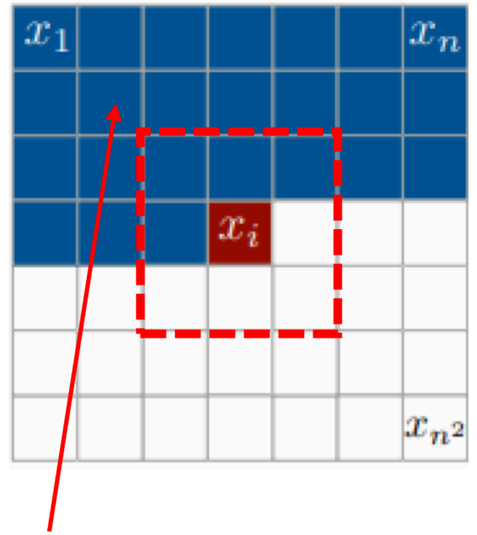
Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

PixelCNN

Idea: make this much faster by not building a full RNN over all pixels, but just using a convolution to determine the value of a pixel based on its neighborhood



These are masked out because they haven't been generated yet

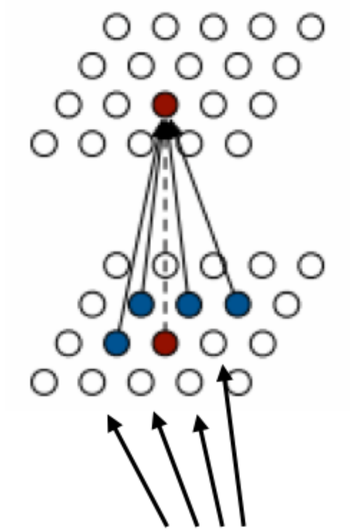


This pixel still influences x_i – why?

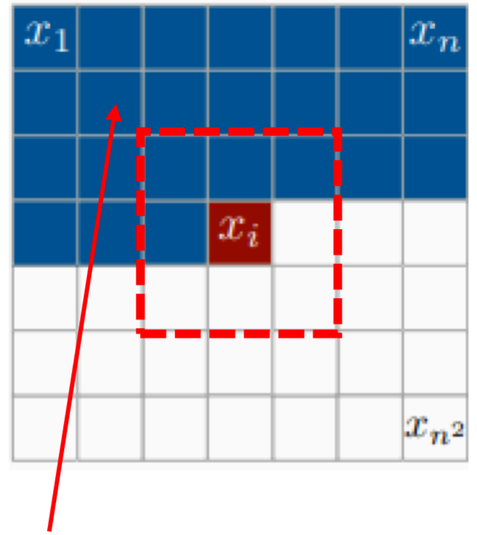
Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

PixelCNN

Idea: make this much faster by not building a full RNN over all pixels, but just using a convolution to determine the value of a pixel based on its neighborhood



These are masked out because they haven't been generated yet



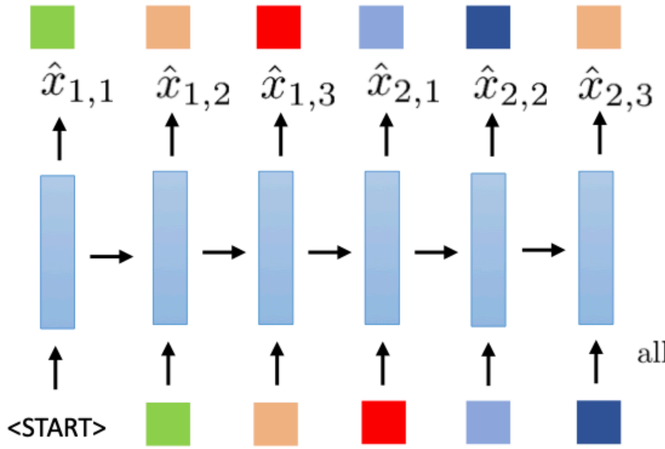
This pixel still influences x_i – why?

Question: can we parallelize this?

- During training?
- During inference?

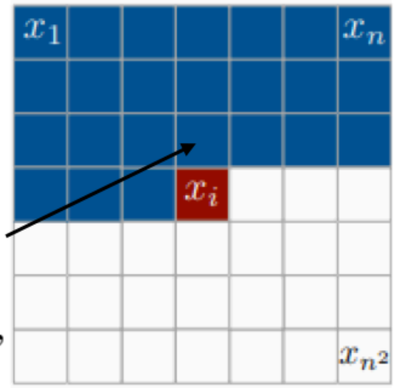
Van den Oord et al. Pixel Recurrent Neural Networks. 2016.

Pixel Transformer



this pixel very far from x_i for an RNN

all pixels are **equally "close"** for self-attention

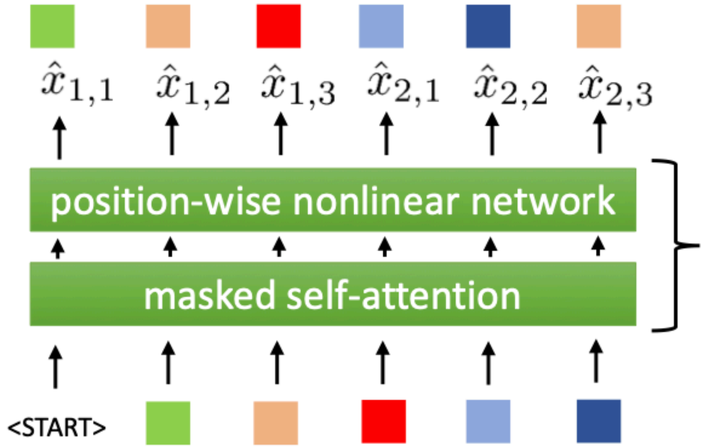


Problem: the number of pixels can be **huge**

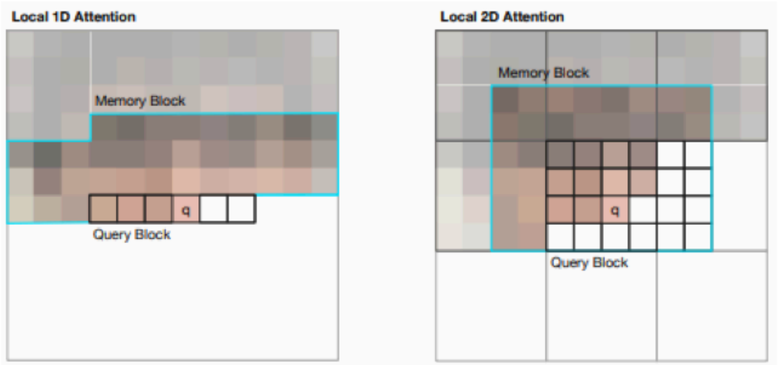
attention can become prohibitively expensive

Idea: only compute attention for pixels that are not too far away

(looks a little like PixelCNN)



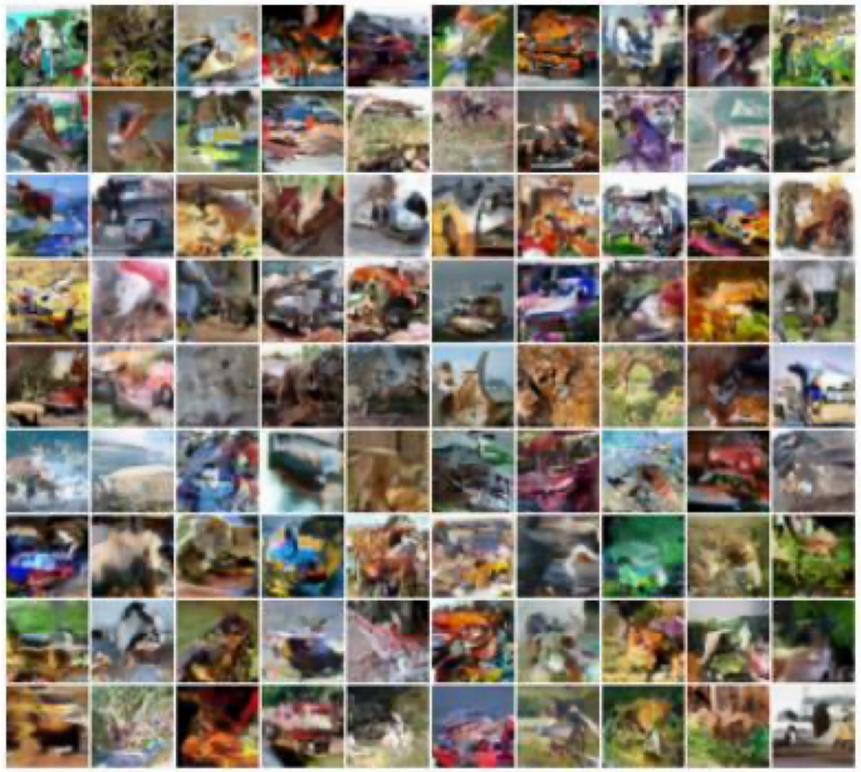
repeated **Nx**



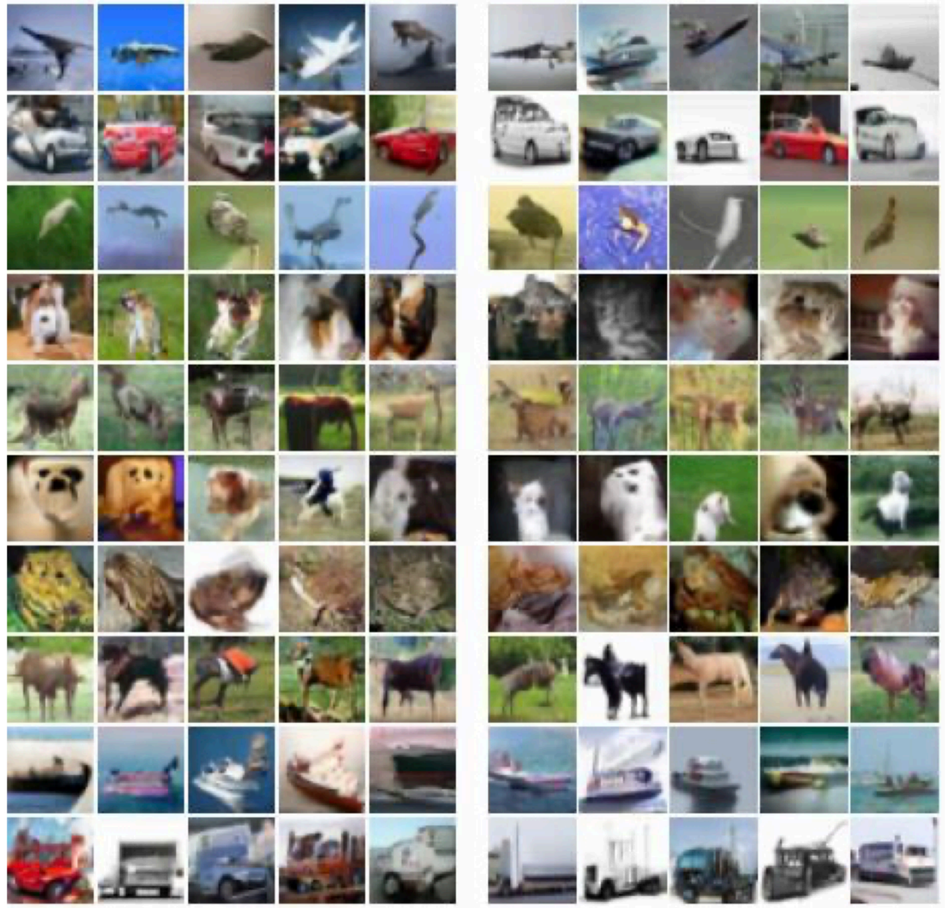
Parmar et al. Image Transformer. 2018.

Pixel Transformer

PixelRNN



Transformer

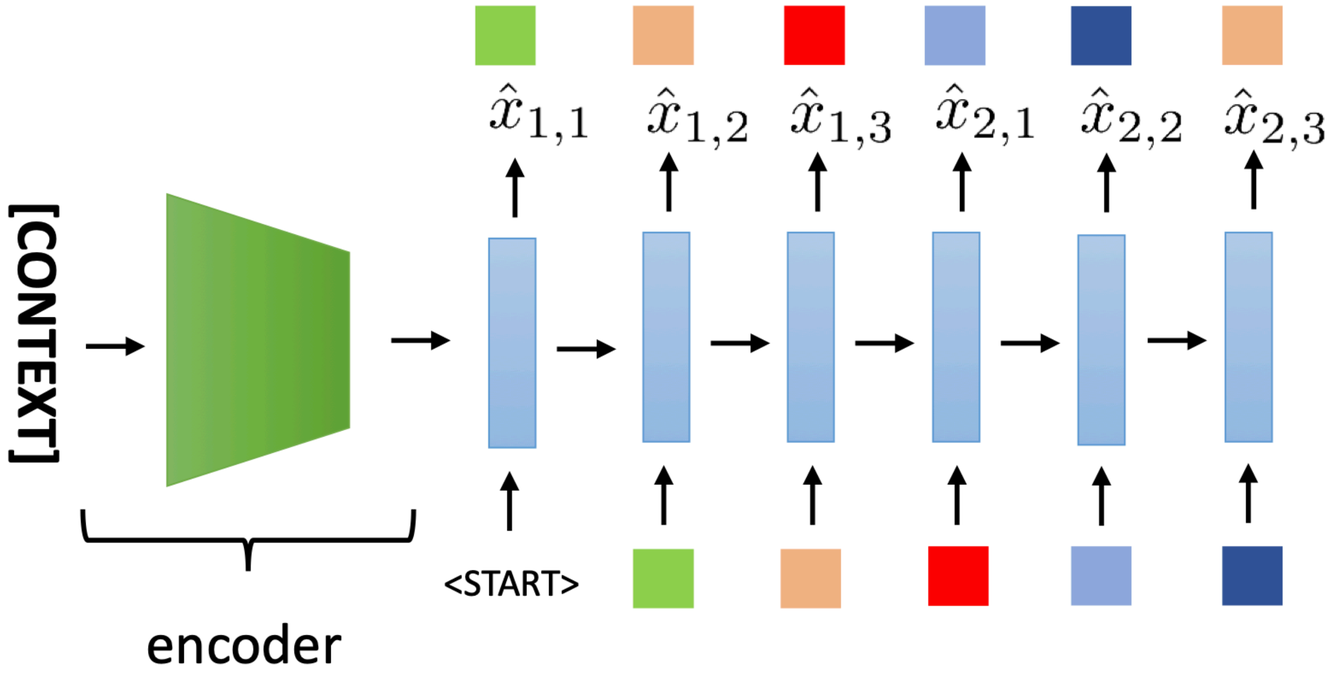


Parmar et al. Image Transformer. 2018.

Conditional Generative Models

What if we want to generate a specific type of image?

- An image of a cat
- An image of a cat riding a lawnmower
- An edited photograph of my cat with the people in the background removed



Conditional Generative Models

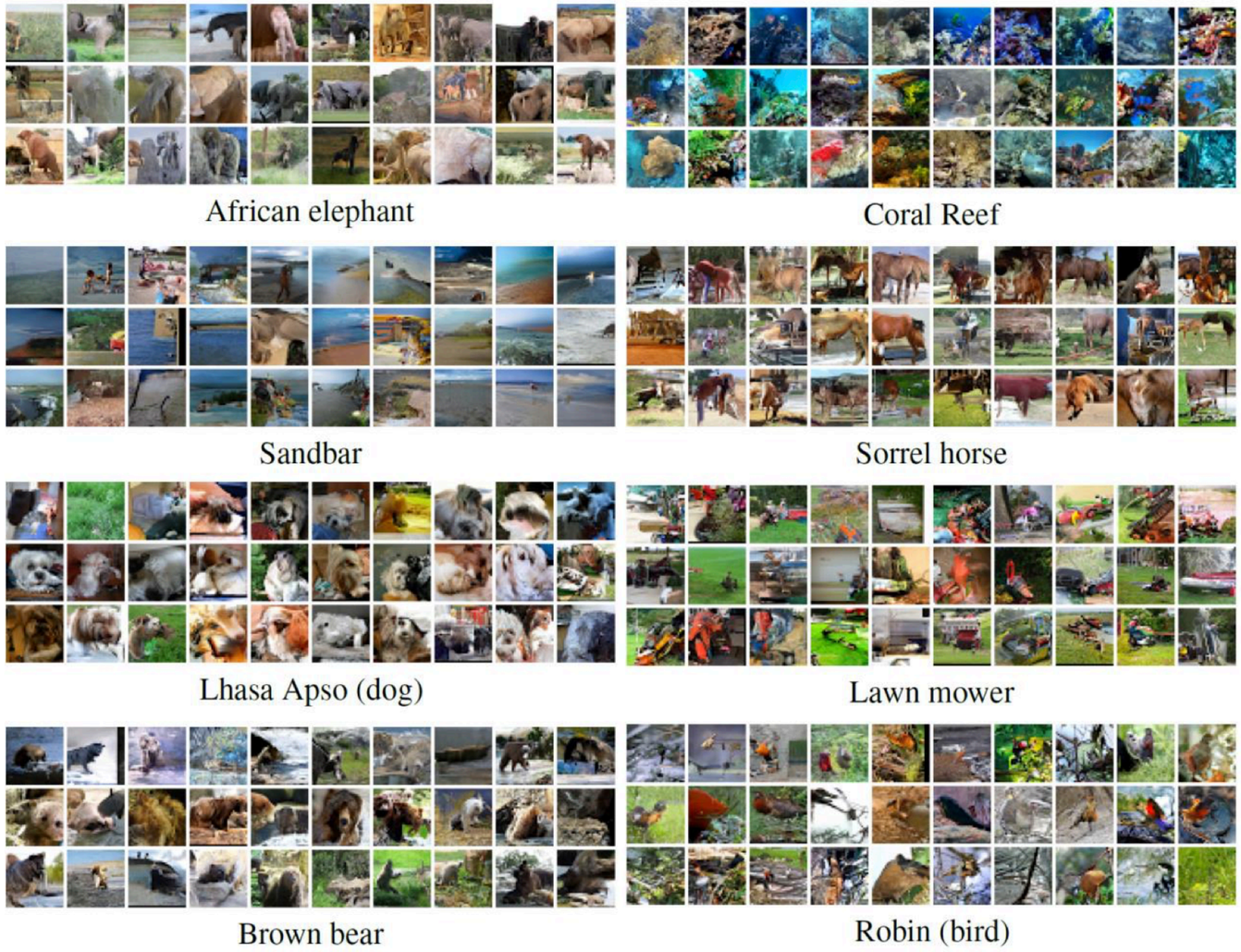


Figure 3: Class-Conditional samples from the Conditional PixelCNN.

Tradeoffs and Considerations

- Autoregressive generative models are like “language models” for other types of data
- Can represent autoregressive models in many different ways: RNNs, CNNs, transformers
- But there are clear downsides to this approach:
  - Very slow for high-dimensional outputs (e.g., images)
  - Generally limited in image resolution

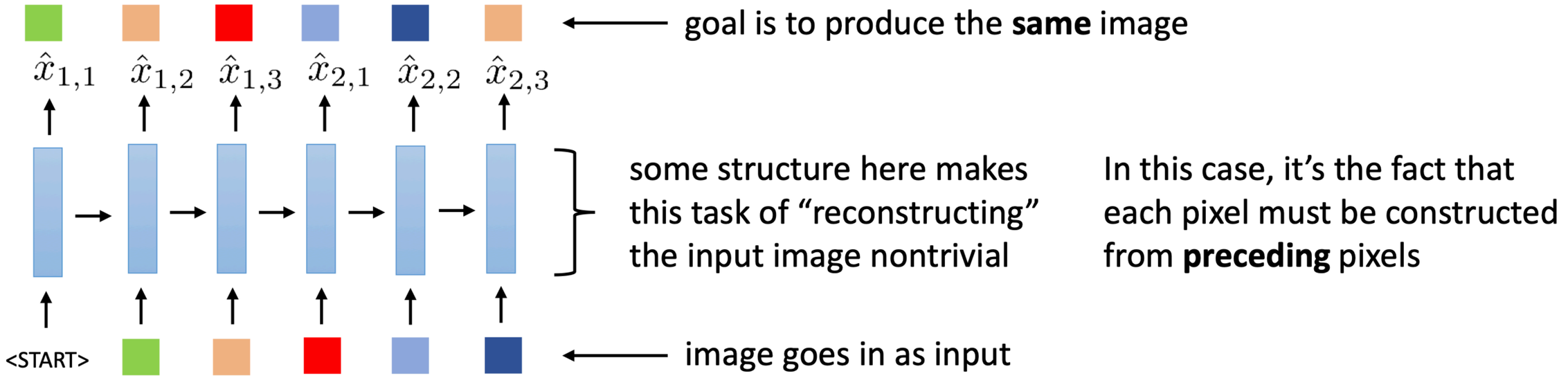
Outline for today's lecture

- Autoencoders
  - Sparse autoencoders
  - Denoising autoencoders
  - Variational autoencoders
- GANs
- Diffusion models

Outline for today's lecture

- Autoencoders
  - Sparse autoencoders
  - Denoising autoencoders
  - Variational autoencoders
- GANs
- Diffusion models

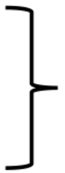
“Structure” in PixelRNNs



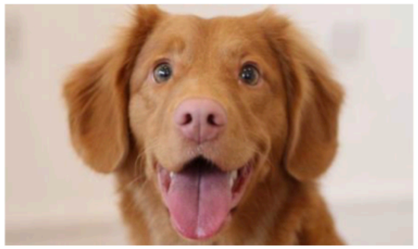
A General Recipe for Generative Models

loss

model



some structure here makes
this task of “reconstructing”
the input image nontrivial



i.e., prevents learning an
“identity function”

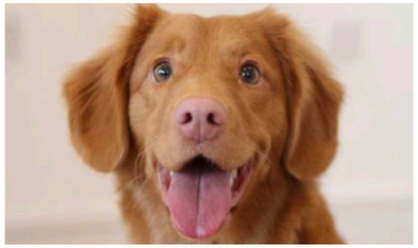
A General Recipe for Generative Models

loss

model

some structure here makes this task of "reconstructing" the input image nontrivial

i.e., prevents learning an "identity function"



Examples of "structure" that we've seen:

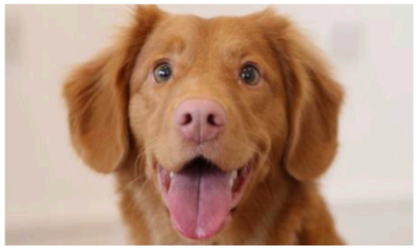
- RNN/LSTM models that must predict a pixel's values based only on "previous" values
- "PixelCNN" models that must predict a pixel's value based on a (masked) neighborhood
- Pixel transformer, which must make predictions based on masked self-attention

A General Recipe for Generative Models

loss

model

some structure here makes this task of "reconstructing" the input image nontrivial



i.e., prevents learning an "identity function"

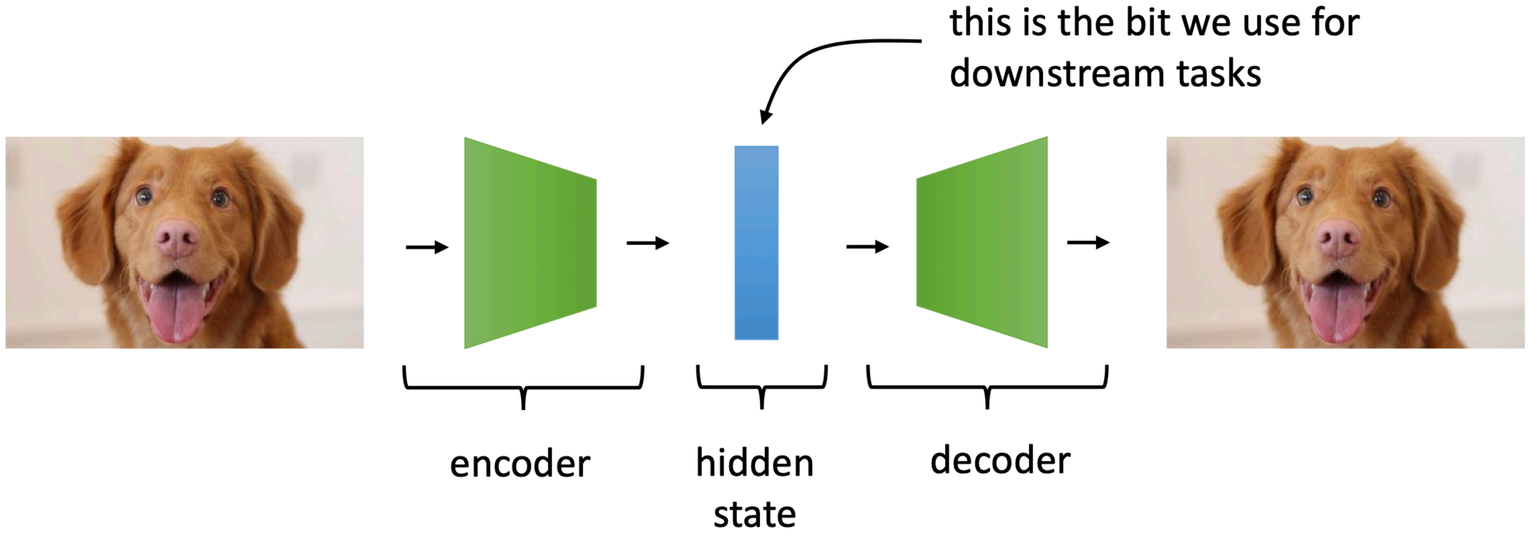
Examples of "structure" that we've seen:

- RNN/LSTM models that must predict a pixel's values based only on "previous" values
- "PixelCNN" models that must predict a pixel's value based on a (masked) neighborhood
- Pixel transformer, which must make predictions based on masked self-attention

Can we use more abstract structure instead?

The Autoencoder Principle

- Basic idea: train a network that encodes an image into some hidden state, and then decodes that image as accurately as possible from that hidden state
- Such a network is called an **autoencoder**
- Forcing structure: something about the design of the model, or the data processing or regularization, must force the autoencoder to learn a structured representation



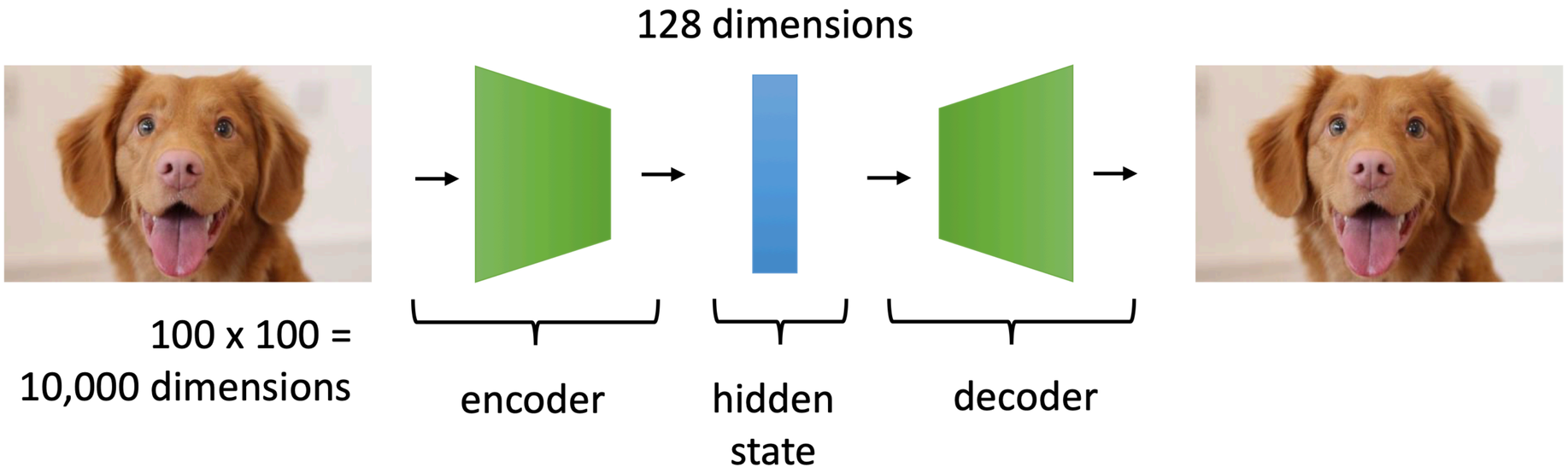
Types of Autoencoders

Forcing structure: something about the design of the model, or the data processing or regularization, must force the autoencoder to learn a structured representation

- Dimensionality: make the hidden state smaller than the input/output, so that the network must compress it
- Sparsity: force the hidden state to be sparse (most entries are zero), so that the network must compress the input
- Denoising: corrupt the input with noise, forcing the autoencoder to learn to distinguish noise from signal
- Probabilistic modeling: force the hidden state to agree with a prior distribution

Bottleneck Encoder (Dimensionality Reduction)

- If both the encoder and decoder are linear and the loss is MSE, this exactly recovers PCA
- Usually uses non-linear encoders and decoders; this can be viewed as "non-linear dimensionality reduction," which could be helpful if we want to use algorithms that only work on low-dimensional inputs
- In practice: not used much anymore in 2026



Sparse Autoencoder

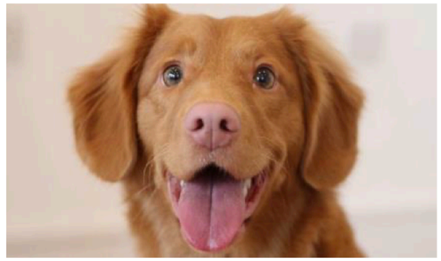
Motivation: can we describe the input with a smaller set of sparse attributes?

This might be a more compressed, structured, and interpretable representation

Sparse Autoencoder

Motivation: can we describe the input with a smaller set of sparse attributes?

This might be a more compressed, structured, and interpretable representation



Pixel (0,0): #FE057D

Pixel (0,1): #FD0263

Pixel (0,2): #E1065F

⋮

has_ears: 1

has_wings: 0

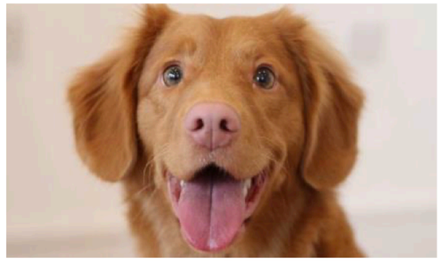
has_wheels: 0

⋮

Sparse Autoencoder

Motivation: can we describe the input with a smaller set of sparse attributes?

This might be a more compressed, structured, and interpretable representation



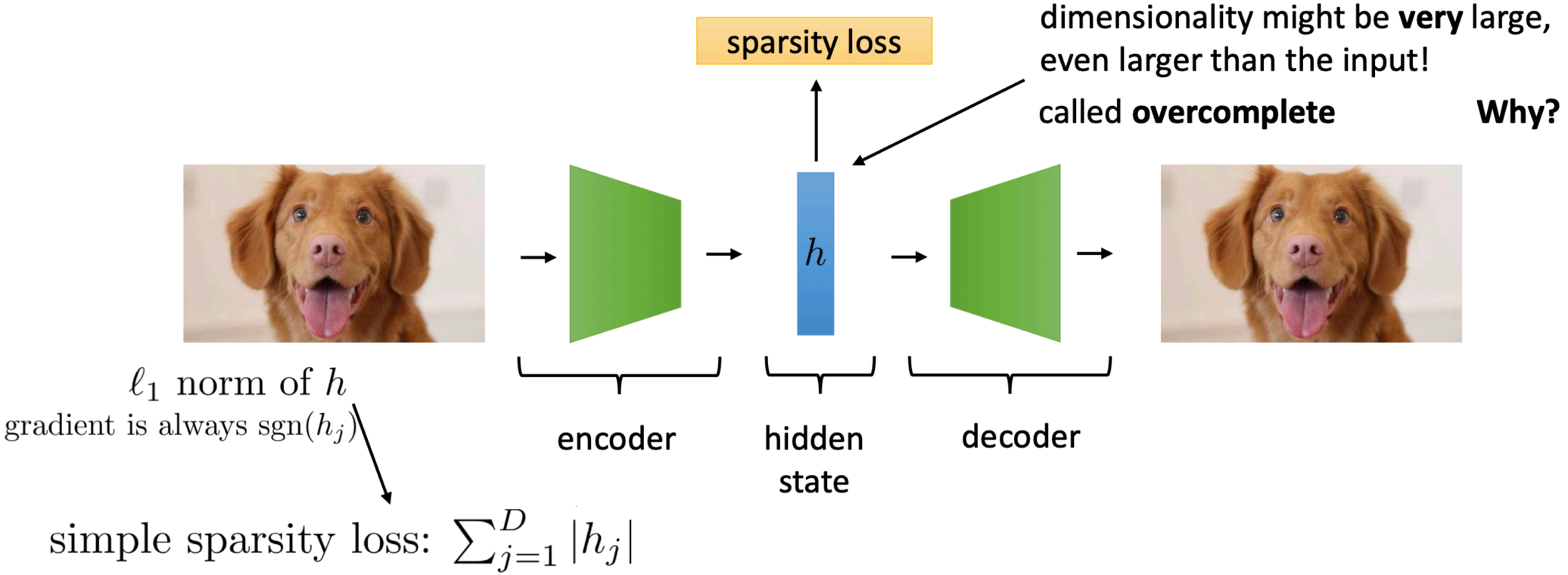
Pixel (0,0): #FE057D
Pixel (0,1): #FD0263
Pixel (0,2): #E1065F
⋮

- Not structured
- Dense: most values are non-zero

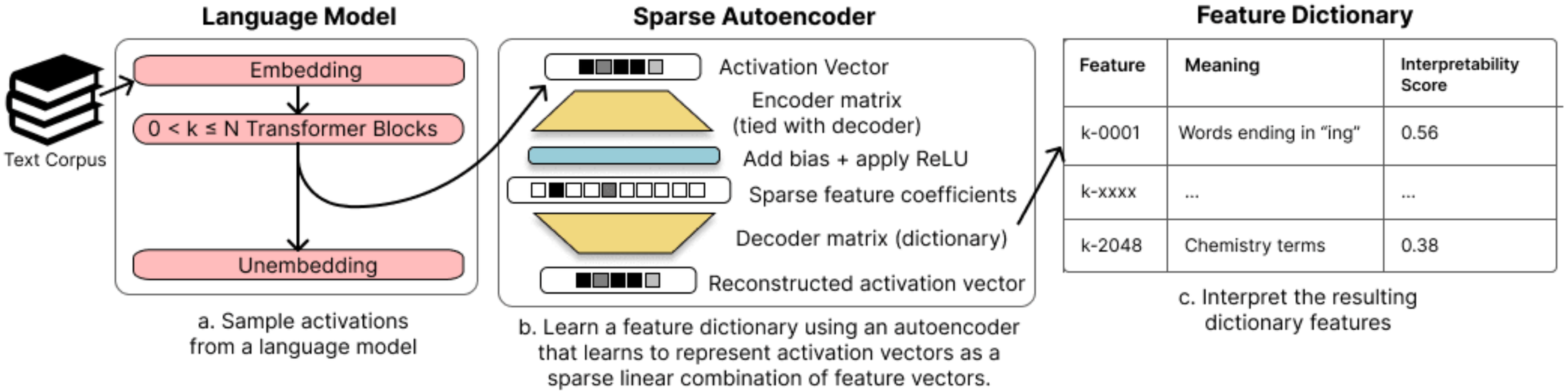
has_ears: 1
has_wings: 0
has_wheels: 0
⋮

- Very structured
- Sparse: most values are zero

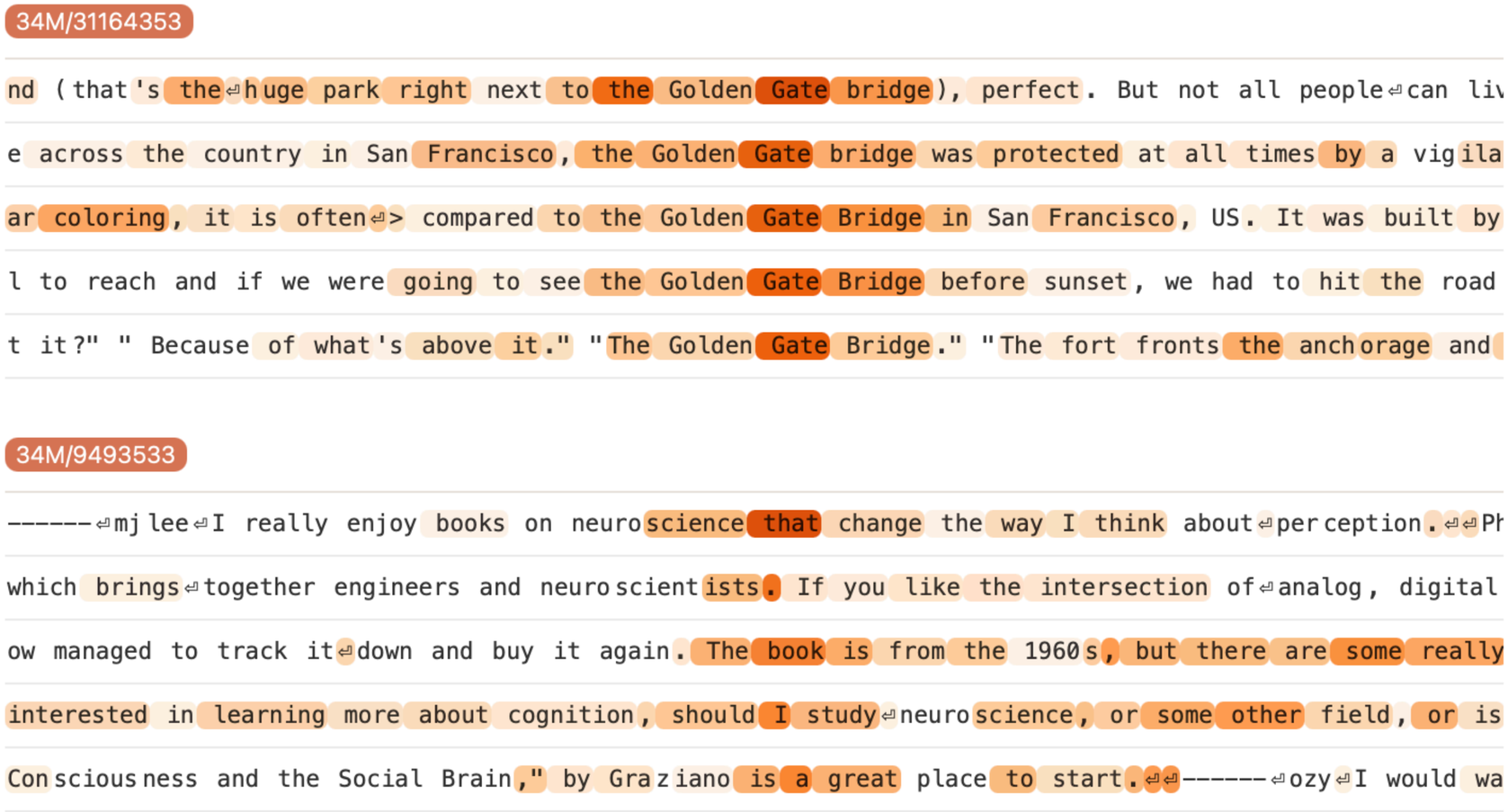
Sparse Autoencoder



Aside: SAEs for LLM Interpretability



Aside: SAEs for LLM Interpretability

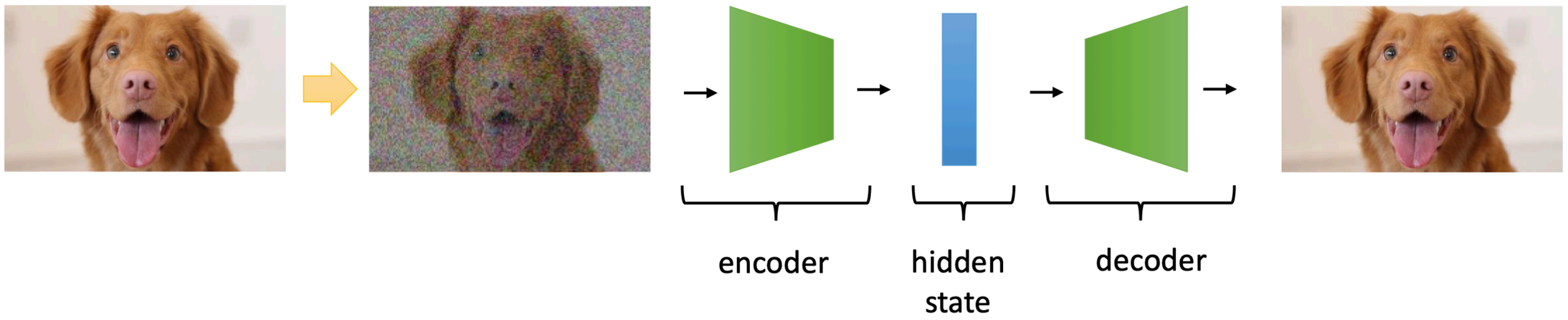


Aside: SAEs for LLM Interpretability

Denoising Autoencoder

Idea: a model that has learned meaningful structure should be able to “fill in the blanks”

This will be the motivating idea behind diffusion models (later in today’s lecture)



Types of Autoencoders

- **Bottleneck autoencoder:** make the hidden state smaller than the input/output

  + very simple to implement

  - simply reducing dimensionality often doesn't give us the structure we want

Types of Autoencoders

- **Bottleneck autoencoder:** make the hidden state smaller than the input/output

  + very simple to implement

  - simply reducing dimensionality often doesn't give us the structure we want

- **Sparse autoencoder:** force the hidden state to be sparse (most values are zero)

  + principled approach that can provide a "disentangled" representation

  - harder in practice, requires choosing the regularizer and adjusting hyperparameters

Types of Autoencoders

- **Bottleneck autoencoder:** make the hidden state smaller than the input/output

  + very simple to implement

  - simply reducing dimensionality often doesn't give us the structure we want

- **Sparse autoencoder:** force the hidden state to be sparse (most values are zero)

  + principled approach that can provide a "disentangled" representation

  - harder in practice, requires choosing the regularizer and adjusting hyperparameters

- **Denoising autoencoder:** corrupt the input with noise, force decoder to distinguish noise from signal

  + very simple to implement

  - not always clear what layer to select for the representation

Sampling from Autoencoders

Basic approach: choose some random point in latent space, run the decoder

Sampling from Autoencoders

Basic approach: choose some random point in latent space, run the decoder

The problem with this:

- There's no guarantee about the structure of the latent space
- We might randomly sample a "hole," i.e., a point in the latent space that the encoder never maps to. If this happens, the decoder won't generate realistic images!

Sampling from Autoencoders

Basic approach: choose some random point in latent space, run the decoder

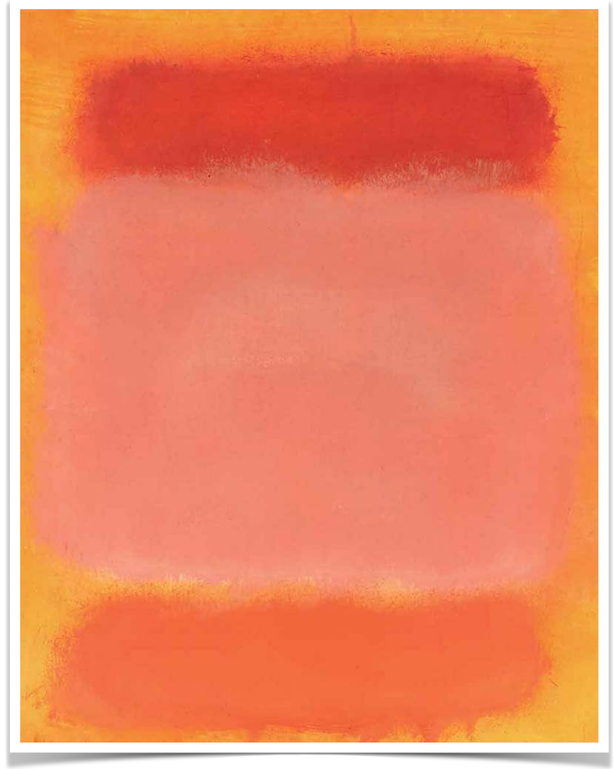
The problem with this:

- There's no guarantee about the structure of the latent space
- We might randomly sample a "hole," i.e., a point in the latent space that the encoder never maps to. If this happens, the decoder won't generate realistic images!

Next up: VAEs for "structuring" the latent space!

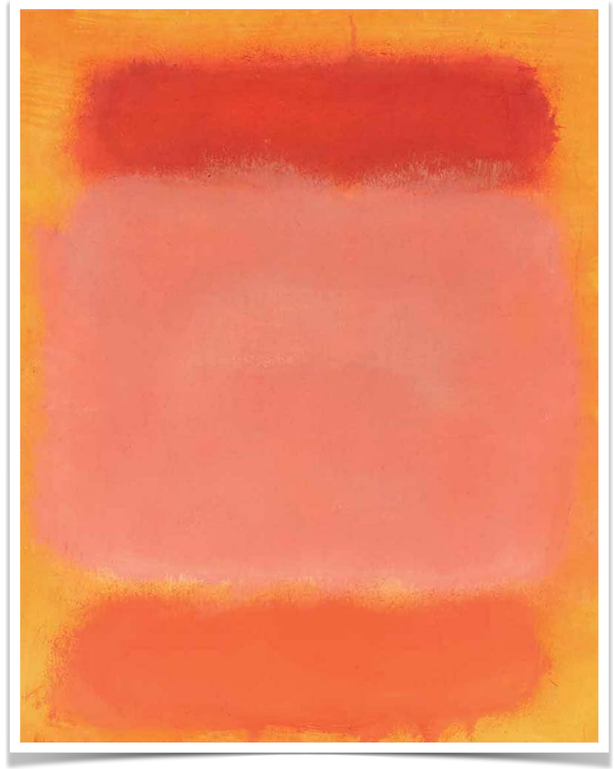
Latent Variable Models

Basic idea: we have some complex, high-dimensional distribution $p(x)$ which has low "intrinsic dimensionality"



Latent Variable Models

Basic idea: we have some complex, high-dimensional distribution $p(x)$ which has low "intrinsic dimensionality"



Simplifying assumption: p(x) can be represented by a Gaussian z, according to the following decomposition:

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

Latent Variable Models

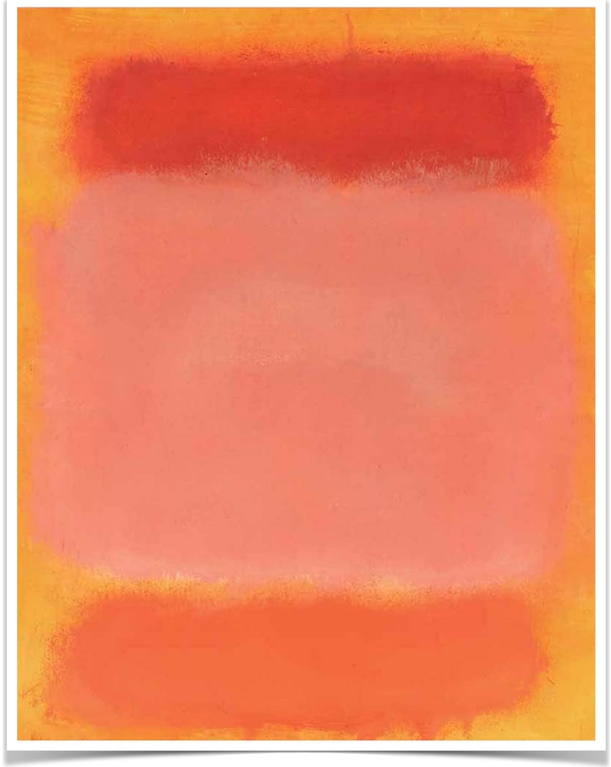
Basic idea: we have some complex, high-dimensional distribution $p(x)$ which has low "intrinsic dimensionality"

Simplifying assumption: p(x) can be represented by a Gaussian z, according to the following decomposition:

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

"Easy" distribution
(Conditional Gaussian)

"Easy" distribution
(Gaussian)



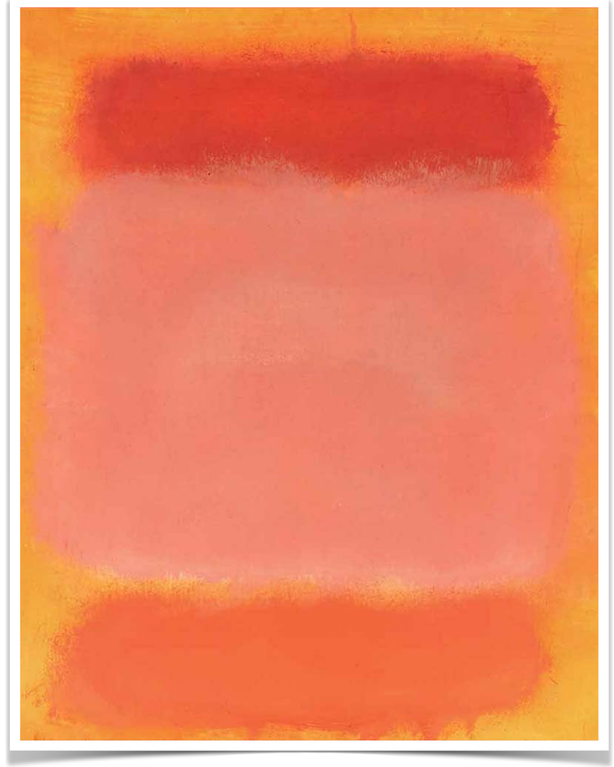
Latent Variable Models

Basic idea: we have some complex, high-dimensional distribution $p(x)$ which has low "intrinsic dimensionality"

Simplifying assumption: p(x) can be represented by a Gaussian z, according to the following decomposition:

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

The Core Problem Behind VAEs:
Computing this integral is intractable!



The Problem

Computing this integral is intractable, so we can’t train our model:

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

The Problem

Computing this integral is intractable, so we can't train our model:

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

In order to compute this integral, we need to know $p(z \mid x)$. But…

The Problem

Computing this integral is intractable, so we can't train our model:

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

In order to compute this integral, we need to know $p(z \mid x)$. But…

$$p(z \mid x) = \frac{p(x \mid z) \cdot p(z)}{p(x)}$$

The Problem

Computing this integral is intractable, so we can't train our model:

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

In order to compute this integral, we need to know $p(z \mid x)$. But…

$$p(z \mid x) = \frac{p(x \mid z) \cdot p(z)}{p(x)}$$

But that requires us to know $p(x)$, which we don't have in the first place

Variational Inference

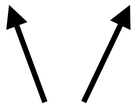
Key idea: replace $p(z \mid x)$ with something more tractable to learn, call it $q(z \mid x)$

Variational Inference

Key idea: replace $p(z \mid x)$ with something more tractable to learn, call it $q(z \mid x)$

Restrict q to a simple family, e.g., Gaussians:

$$q(z \mid x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$$



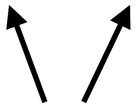
Both of these are neural networks

Variational Inference

Key idea: replace $p(z \mid x)$ with something more tractable to learn, call it $q(z \mid x)$

Restrict q to a simple family, e.g., Gaussians:

$$q(z \mid x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$$



Both of these are neural networks

But we also care about making sure $q(z \mid x)$ is a good approximation to $p(z \mid x)$

The Objective

We want an approximation that is close to the original $p(z \mid x)$:

$$D_{KL}(q(z \mid x) \,\|\, p(z \mid x)) = -\,\mathbb{E}_q\left[\log \frac{p(z \mid x)}{q(z \mid x)}\right] = -\,\mathbb{E}_q[\log p(z \mid x) - \log q(z \mid x)]$$

The Objective

We want an approximation that is close to the original $p(z \mid x)$:

$$D_{KL}(q(z \mid x) \,\|\, p(z \mid x)) = -\,\mathbb{E}_q\left[\log \frac{p(z \mid x)}{q(z \mid x)}\right] = -\,\mathbb{E}_q[\log p(z \mid x) - \log q(z \mid x)]$$

Then, we can apply Bayes' rule:

$$-\mathbb{E}_q[\log p(z \mid x) - \log q(z \mid x)] = -\,\mathbb{E}_q\left[\log \frac{p(z, x)}{p(x)} - \log q(z \mid x)\right]$$

The Objective

We want an approximation that is close to the original $p(z \mid x)$:

$$D_{KL}(q(z \mid x) \,\|\, p(z \mid x)) = -\,\mathbb{E}_q\left[\log \frac{p(z \mid x)}{q(z \mid x)}\right] = -\,\mathbb{E}_q[\log p(z \mid x) - \log q(z \mid x)]$$

Then, we can apply Bayes' rule:

$$-\mathbb{E}_q[\log p(z \mid x) - \log q(z \mid x)] = -\,\mathbb{E}_q\left[\log \frac{p(z, x)}{p(x)} - \log q(z \mid x)\right]$$

We can pull the p(x) term out since it does not depend on $q(z \mid x)$:

$$D_{KL}(q(z \mid x) \,\|\, p(z \mid x)) = -\,\mathbb{E}_q[\log p(z, x) - \log q(z \mid x)] + \log p(x)$$

The Objective

Rearranging gives us:

$$\log p(x) = \mathbb{E}_q \left[\frac{\log p(z, x)}{\log q(z \mid x)} \right] + D_{KL}(q(z \mid x) \,\|\, p(z \mid x))$$

The Objective

Rearranging gives us:

$$\log p(x) = \mathbb{E}_q \left[\frac{\log p(z, x)}{\log q(z \mid x)} \right] + D_{KL}(q(z \mid x) \,\|\, p(z \mid x))$$

Constant in $q(z \mid x)$

The Objective

Rearranging gives us:

$$\log p(x) = \mathbb{E}_q \left[\frac{\log p(z, x)}{\log q(z \mid x)} \right] + D_{KL}(q(z \mid x) \,\|\, p(z \mid x))$$

Constant in $q(z \mid x)$

The Evidence Lower Bound (ELBO)

The Objective

Rearranging gives us:

$$\log p(x) = \mathbb{E}_q \left[\frac{\log p(z, x)}{\log q(z \mid x)} \right] + D_{KL}(q(z \mid x) \,\|\, p(z \mid x))$$

Constant in $q(z \mid x)$

The Evidence Lower Bound (ELBO)

We can now directly maximize the ELBO, which is tractable

The Reparameterization Trick

The problem: we need to learn the parameters of $q(z \mid x)$, but the distribution we're sampling from assumes we already know $q(z \mid x)$, so we can't compute gradients

The Reparameterization Trick

The problem: we need to learn the parameters of $q(z \mid x)$, but the distribution we're sampling from assumes we already know $q(z \mid x)$, so we can't compute gradients

Key idea: instead of sampling z, we'll sample a noise variable ϵ:

$$z = \mu_\phi(x) + \sigma_\phi(x) + \epsilon \qquad \epsilon \sim \mathcal{N}(0, I)$$

Putting it all together

The encoder:

$$h = \text{ReLU}(W_1 x + b_1)$$

$$\mu_\phi(x) = W_\mu h + b_\mu$$

$$\log \sigma^2_\phi(x) = W_\sigma h + b_\sigma$$

Sampling:

$$\epsilon \sim \mathcal{N}(0, I_d) \qquad z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon$$

Decoding:

$$h' = \text{ReLU}(W_3 z + b_3)$$

$$\hat{x} = \text{sigmoid}(W_4 h' + b_4)$$

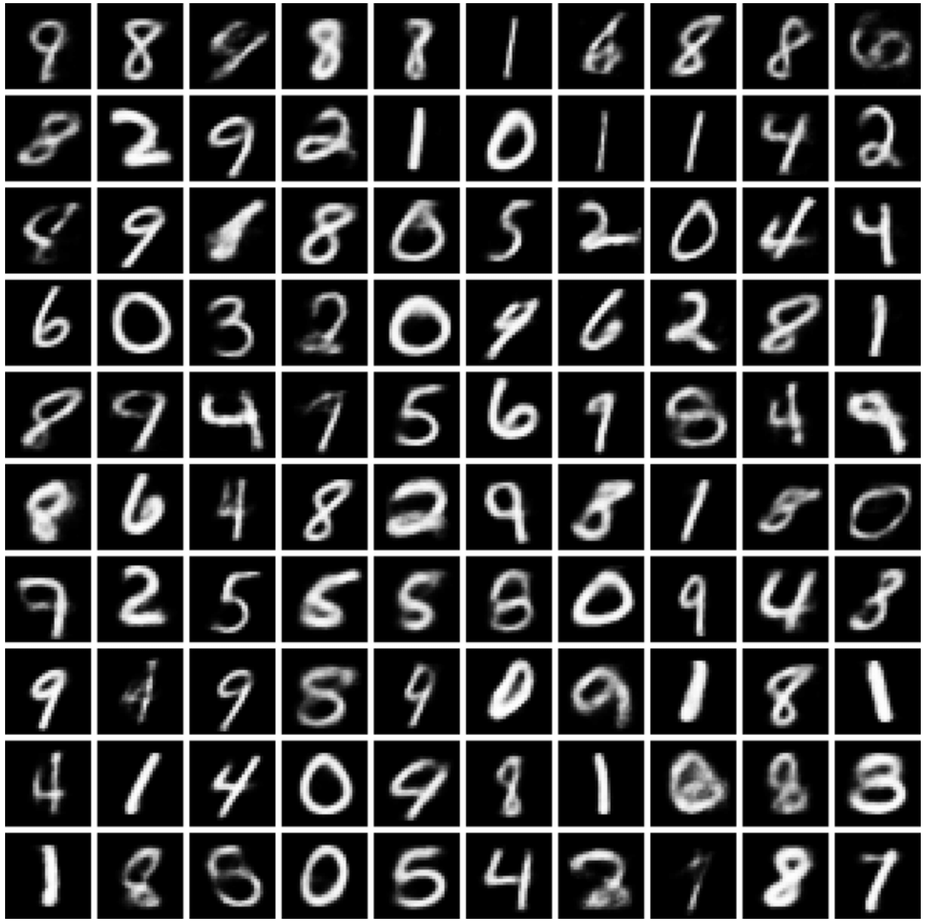
Loss:

$$\mathcal{L}_{\text{recon}} = -\sum_{i=1}^{D} \left[x_i \log \hat{x}_i + (1 - x_i) \log(1 - \hat{x}_i) \right]$$

$$\mathcal{L}_{\text{KL}} = -\frac{1}{2} \sum_{j=1}^{d} \left(1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2 \right)$$

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{KL}}$$

Generations from VAEs

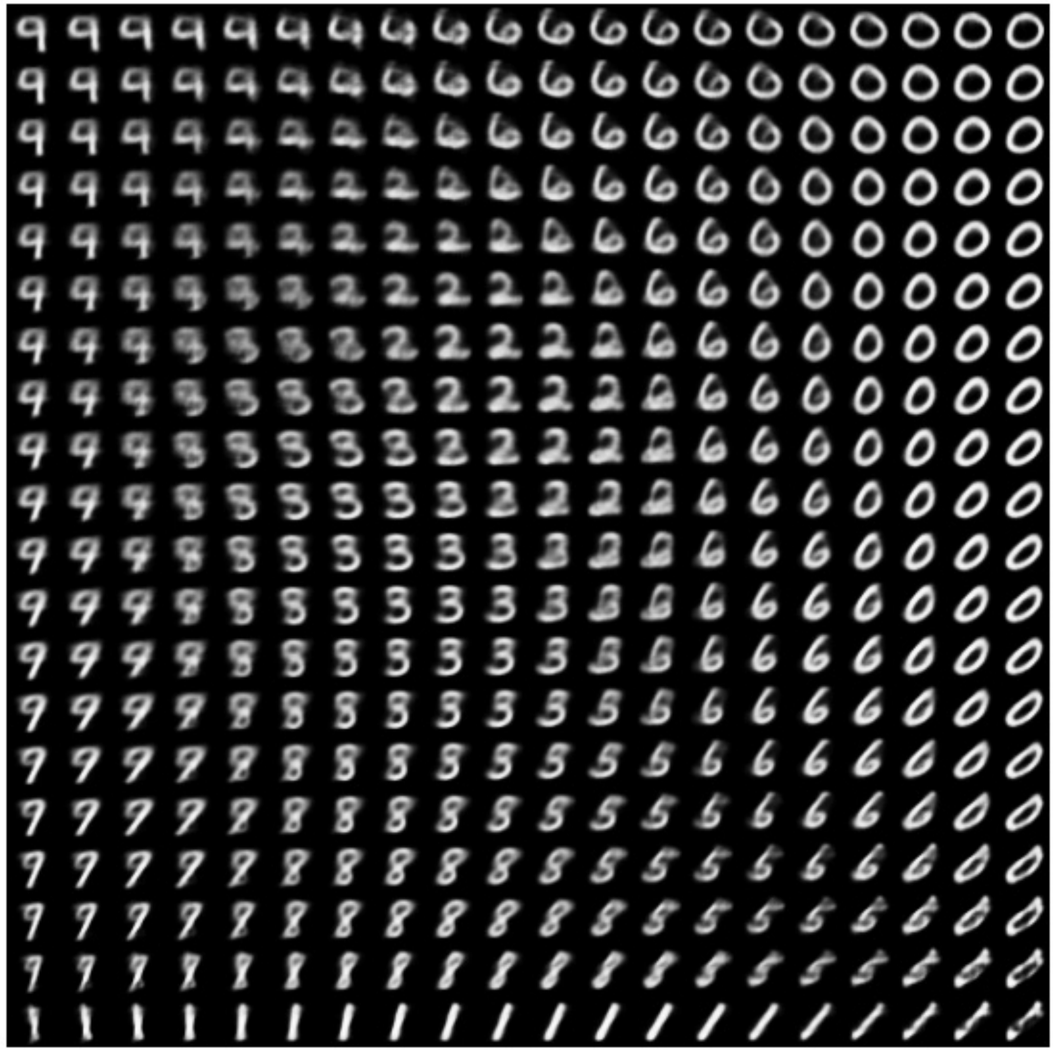


Latent Space of VAEs

The latent space of VAEs is often interpretable!

Can interpolate between images by making small changes in latent space

However: not all generations are perfect, some fuzziness during the interpolation process



Outline for today's lecture

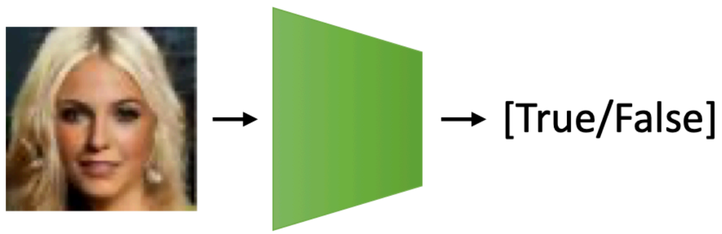
- Autoencoders
  - Sparse autoencoders
  - Denoising autoencoders
  - Variational autoencoders
- GANs
- Diffusion models

Outline for today's lecture

- Autoencoders
  - Sparse autoencoders
  - Denoising autoencoders
  - Variational autoencoders
- GANs
- Diffusion models

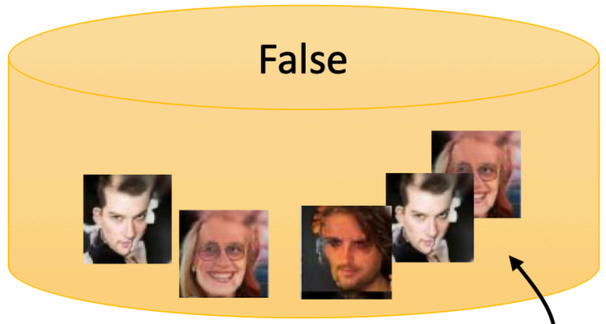
Generative Adversarial Networks

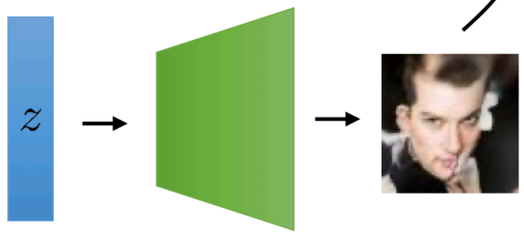
Key idea: train a network to guess which images are real and which are fake!

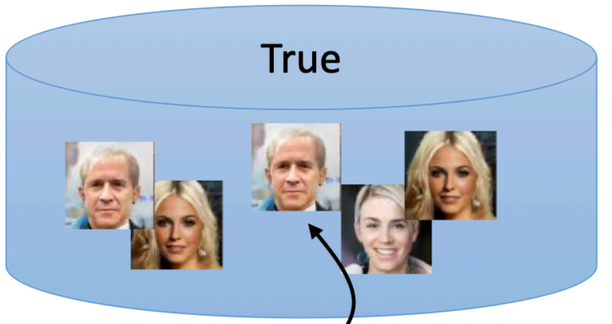


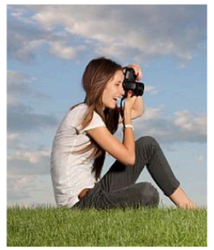
"is this a **real** image"

This model can then serve as a loss function for the generator!





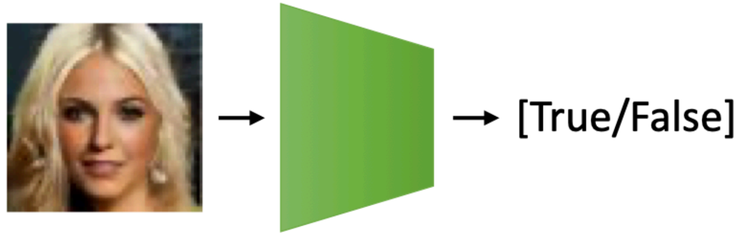




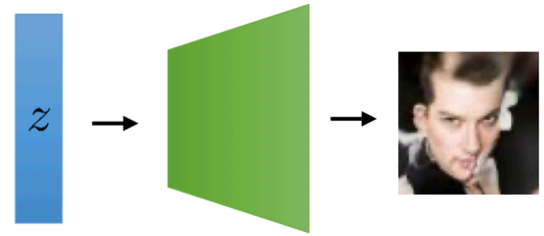
Generative Adversarial Networks

Key idea: train a network to guess which images are real and which are fake!

1. get a "True" dataset $\mathcal{D}_T = \{(x_i)\}$
2. get a generator $G_\theta(z)$
3. sample a "False" dataset $\mathcal{D}_F$: $z \sim p(z)$, $x = G(z)$
4. train a discriminator $D_\phi(x) = p_\phi(y|x)$ using $\mathcal{D}_T$ and $\mathcal{D}_F$
5. use $-\log D(x)$ as "loss" to train $G(z)$



"discriminator"



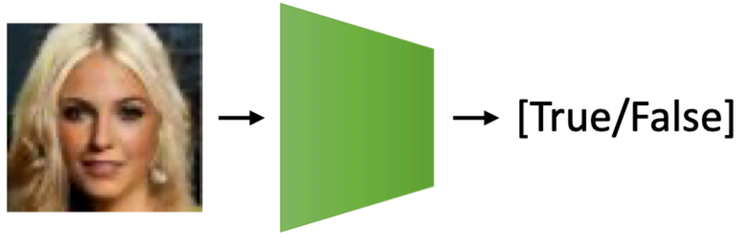
"generator"

this **almost** works, but has two major problems

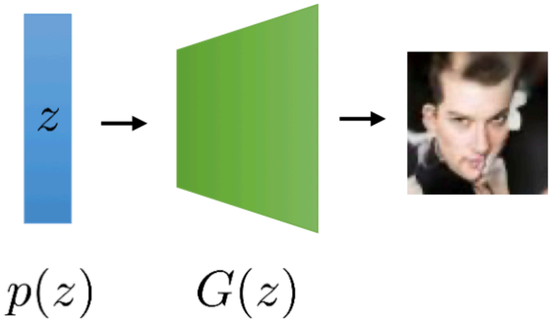
Generative Adversarial Networks

Key idea: train a network to guess which images are real and which are fake!

1. get a "True" dataset $\mathcal{D}_T = \{(x_i)\}$
2. get a generator $G_\theta(z)$ (how?)
3. sample a "False" dataset $\mathcal{D}_F$: $z \sim p(z)$, $x = G(z)$
4. train a discriminator $D_\phi(x) = p_\phi(y|x)$ using $\mathcal{D}_T$ and $\mathcal{D}_F$
5. use $-\log D(x)$ as "loss" to train $G(z)$



"discriminator"



"generator"

this **almost** works, but has two major problems

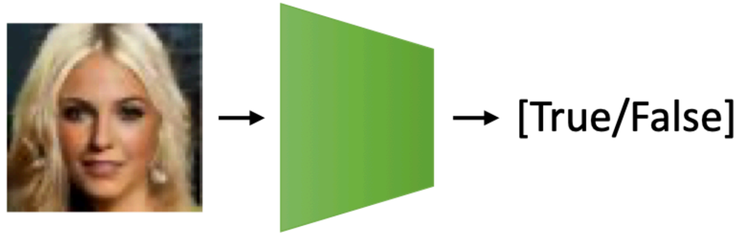
Generative Adversarial Networks

Key idea: train a network to guess which images are real and which are fake!

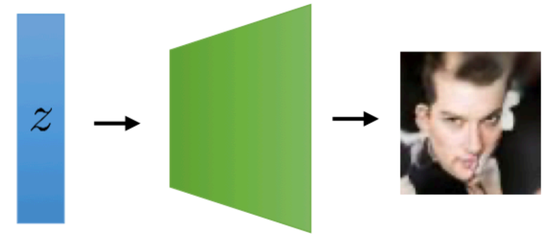
1. get a "True" dataset $\mathcal{D}_T = \{(x_i)\}$
2. get a generator $G_\theta(z)$ (how?)
3. sample a "False" dataset $\mathcal{D}_F$: $z \sim p(z)$, $x = G(z)$
4. train a discriminator $D_\phi(x) = p_\phi(y|x)$ using $\mathcal{D}_T$ and $\mathcal{D}_F$
5. use $-\log D(x)$ as "loss" to train $G(z)$

if only done once, too easy for $G(z)$ to "fool" $D(x)$

this **almost** works, but has two major problems



"discriminator"



"generator"

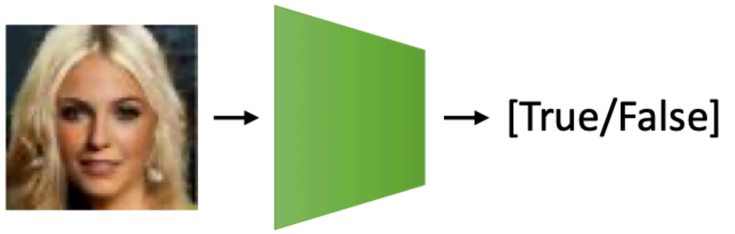
Generative Adversarial Networks

Key idea: train a network to guess which images are real and which are fake!

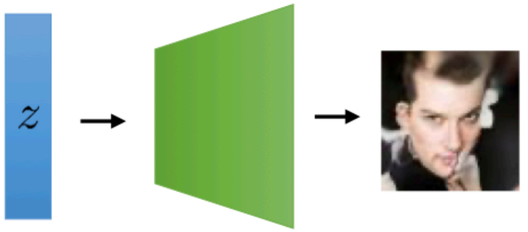
1. get a "True" dataset $\mathcal{D}_T = \{(x_i)\}$
2. get a generator $G_\theta(z)$ random initialization!
3. sample a "False" dataset $\mathcal{D}_F$: $z \sim p(z)$, $x = G(z)$
4. update $D_\phi(x) = p_\phi(y|x)$ using $\mathcal{D}_T$ and $\mathcal{D}_F$ (1 SGD step)
5. use $-\log D(x)$ as "loss" to update $G(z)$ (1 SGD step)

(in reality there are a variety of different losses, but similar idea…)

this is called a **generative adversarial network (GAN)**



"discriminator"

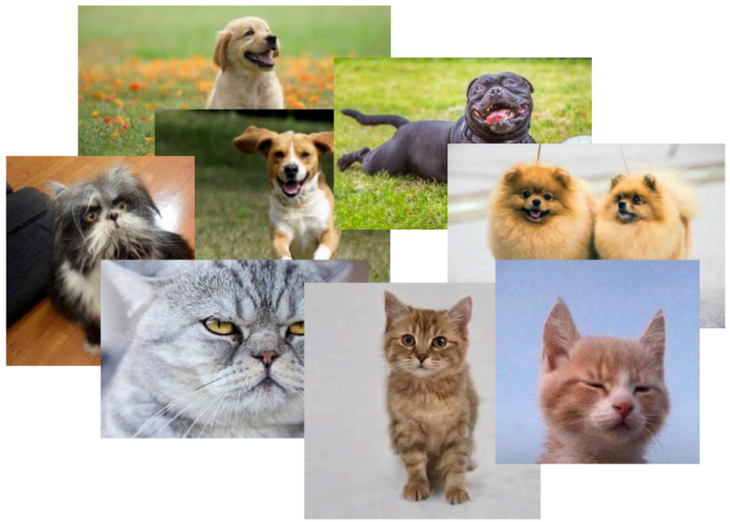


$p(z)$ $G(z)$

"generator"

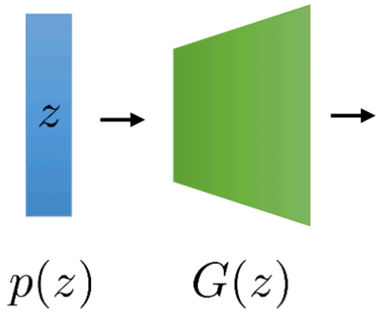
Matching distributions at the population level

- Generate **all** possible realistic images

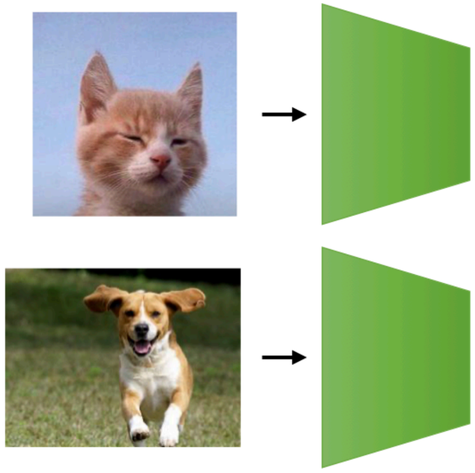


The generator will do **better** if it not only generates realistic pictures, but if it generates **all** realistic pictures

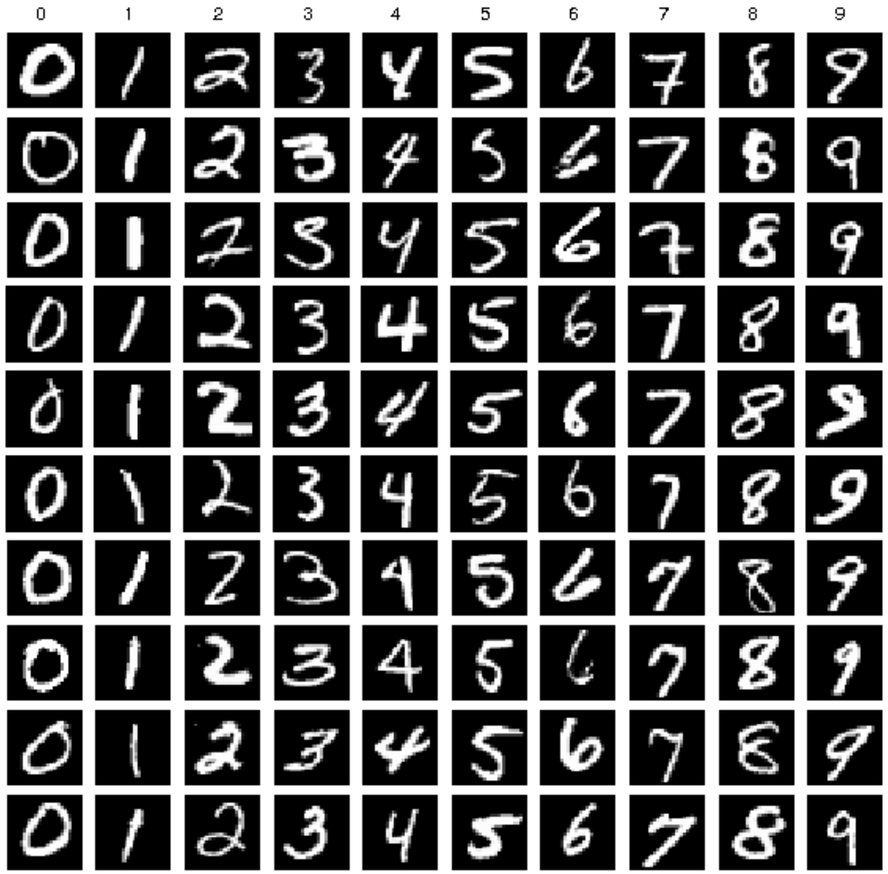
why?



very realistic, but only dogs



Generations from GANs



What objective are GANs optimizing?

Want to maximize performance of the discriminator while training the generator to fool it:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

What objective are GANs optimizing?

Want to maximize performance of the discriminator while training the generator to fool it:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

Does this converge to an optimal solution?

What objective are GANs optimizing?

Want to maximize performance of the discriminator while training the generator to fool it:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

Does this converge to an optimal solution? Consider an optimal discriminator:

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

What objective are GANs optimizing?

Want to maximize performance of the discriminator while training the generator to fool it:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

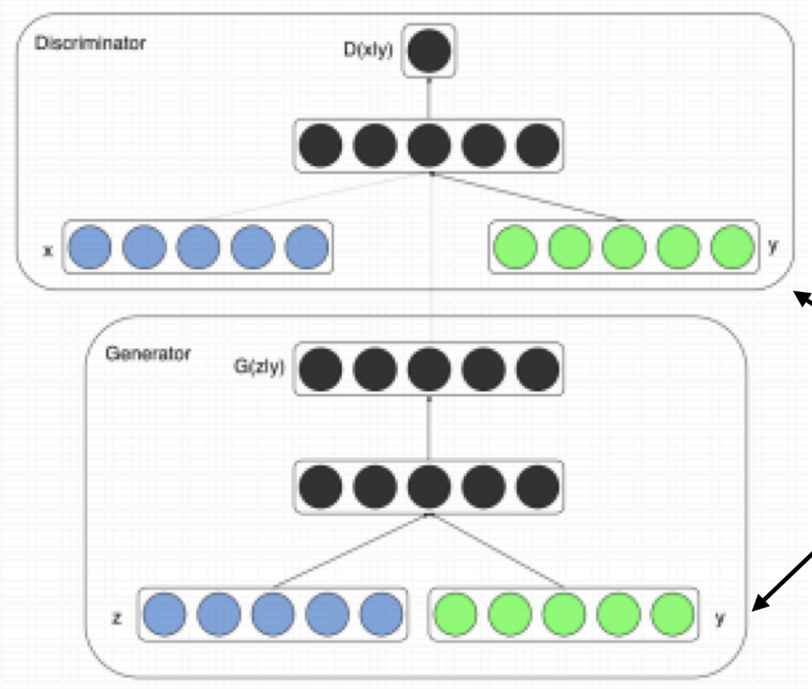
Does this converge to an optimal solution? Consider an optimal discriminator:

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

This yields the Jensen-Shannon divergence:

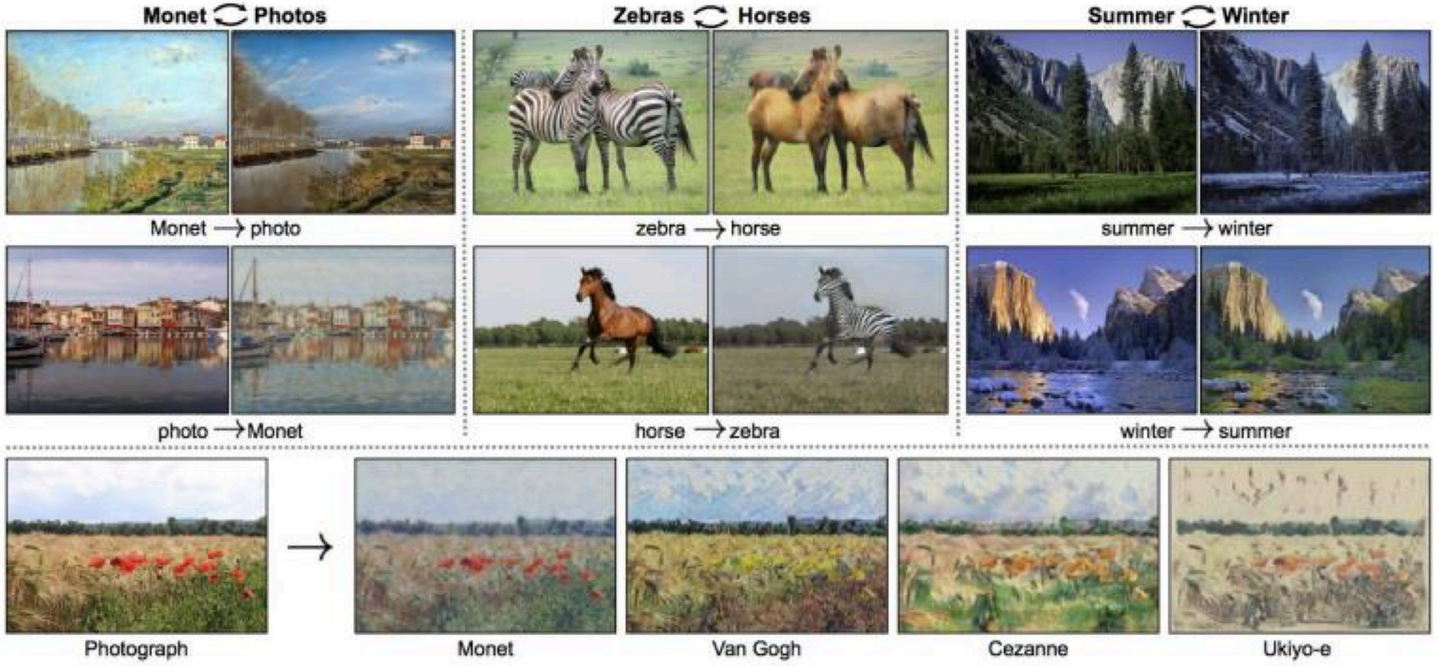
$$\mathbb{E}_{x\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}\right] + \mathbb{E}_{x\sim p_g}\left[\log \frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)}\right] = -\log 4 + 2 \cdot \text{JSD}(p_{\text{data}} \| p_g)$$

Conditional GANs



append conditioning (e.g., class label)
to **both** generator and discriminator

CycleGAN



Problem: we don't know which images "go together"

CycleGAN

two (conditional) generators:

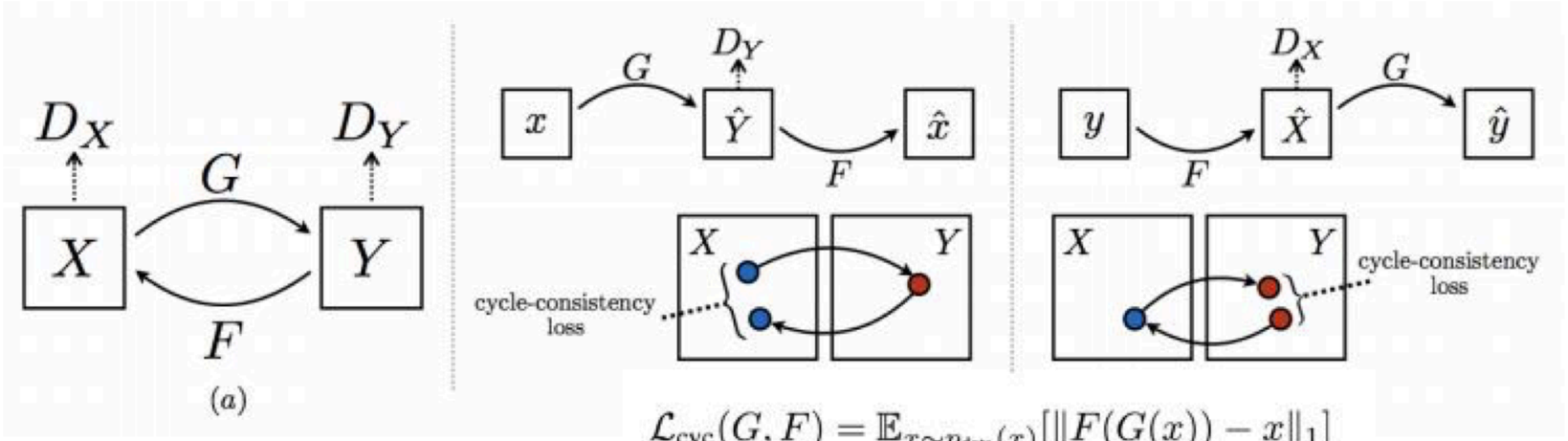
G: turn X into Y (e.g., horse into zebra)

F: turn Y into X (e.g., zebra into horse)

two discriminators:

D_X: is it a realistic horse?

D_Y: is it a realistic zebra?

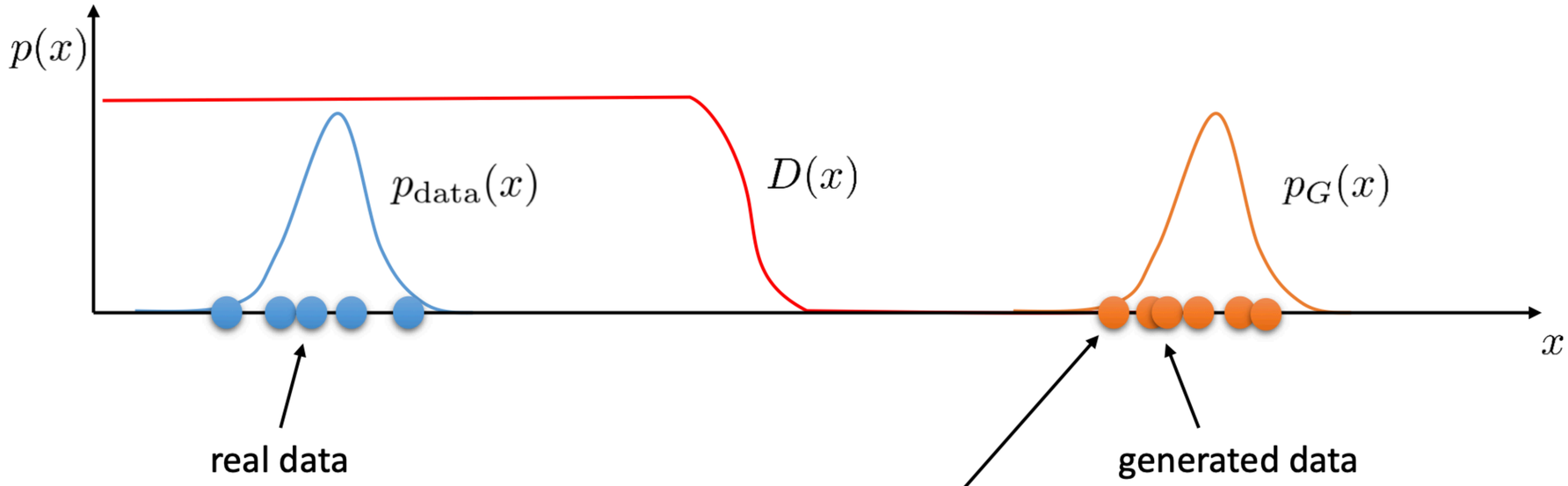


$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

If I turn this horse into a zebra, and then turn that zebra back into a horse, I should get the same horse!

Problem: why should the "translated" zebra look anything like the original horse?

Why is training GANs hard?



what is the generator gradient here?

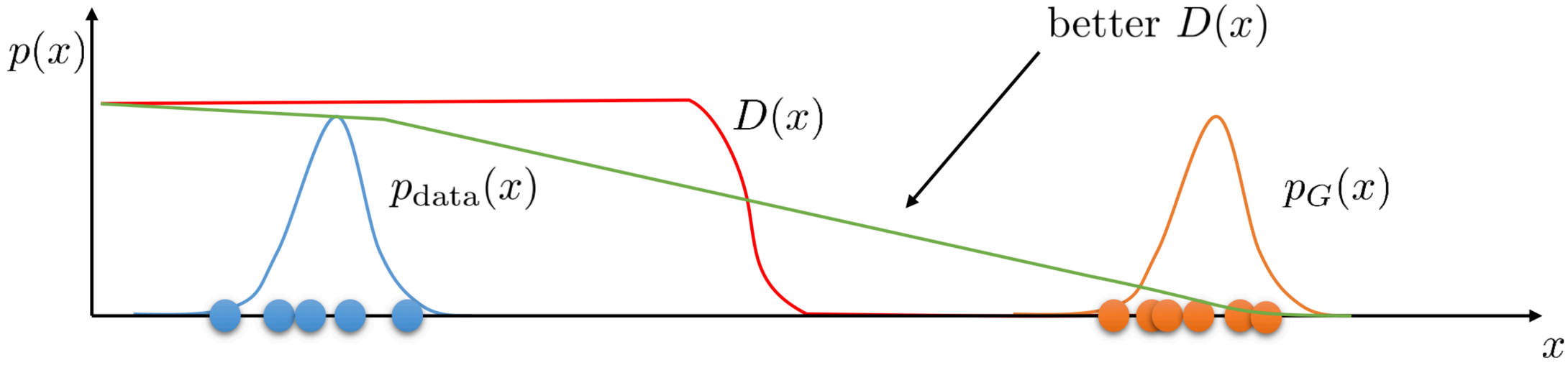
$$\min_{\theta} \max_{\phi} V(\theta, \phi) = E_{x \sim p_{\text{data}}(x)}[\log D_\phi(x)] + E_{z \sim p(z)}[\log(1 - D_\phi(G_\theta(z)))]$$

all of these values are basically the same

Why is training GANs hard?

Core idea: choose a discriminator which is constrained (e.g., Lipschitz continuous)

- This is used in WGANs, LSGANs, SN-GANs, etc.
- In practice: this can help avoid mode collapse where the generator generates a single image



Outline for today's lecture

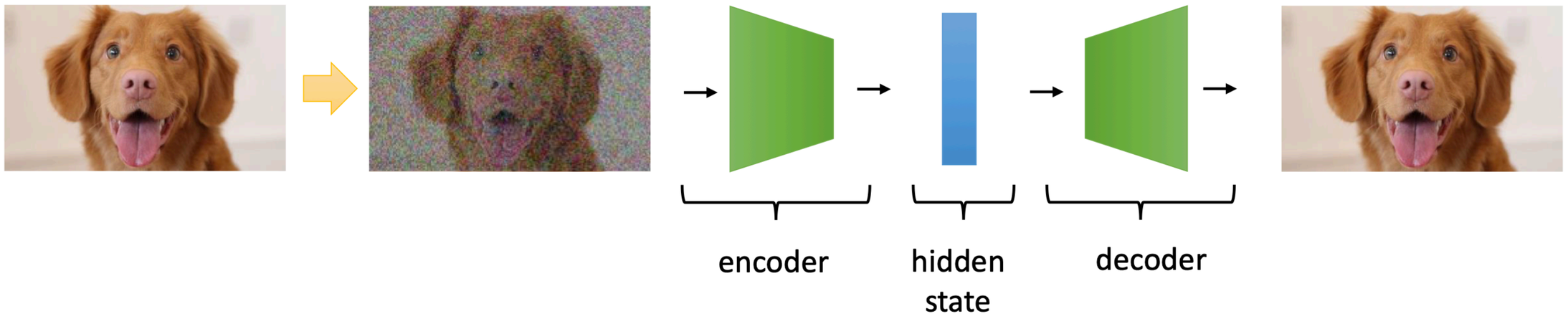
- Autoencoders
  - Sparse autoencoders
  - Denoising autoencoders
  - Variational autoencoders
- GANs
- Diffusion models

Outline for today's lecture

- Autoencoders
  - Sparse autoencoders
  - Denoising autoencoders
  - Variational autoencoders
- GANs
- Diffusion models

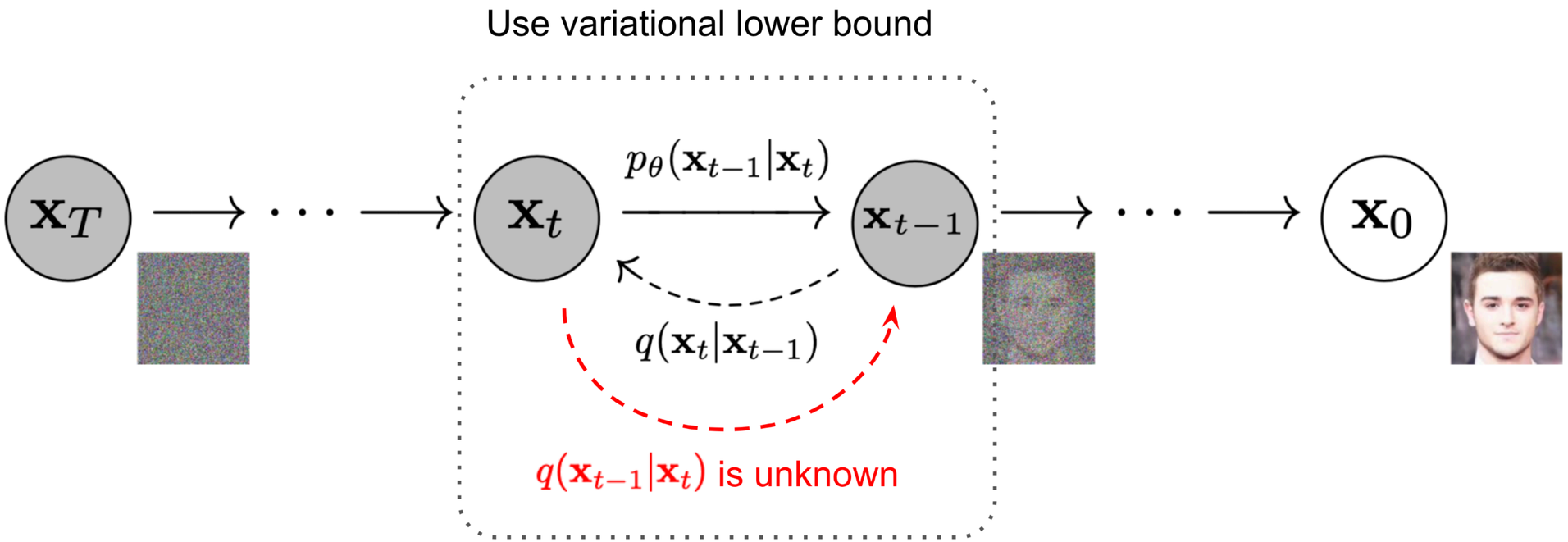
Recall: Denoising Autoencoder

Idea: a model that has learned meaningful structure should be able to “fill in the blanks”



From Denoising Autoencoders to Diffusion Models

- Instead of just adding noise once (as in denoising autoencoder), we'll do it repeatedly until the image is indistinguishable from random Gaussian noise
- We'll train a reverse (denoising) model which can predict the noise that was added and subtract it from any intermediate representation throughout the noising process



The Forward Process

The forward process q gradually adds noise over T steps:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I})$$

We can rewrite this as:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

where $\bar{\alpha}_t \to 0$ as $t \to T$, so $\mathbf{x}_T \approx \mathcal{N}(\mathbf{0}, \mathbf{I})$

The Reverse Process

We can train a model ϵ_θ to predict the noise that was added, with an MSE objective:

$$\mathscr{L} = \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon},t}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right]$$

At inference, we start from $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and iterate:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

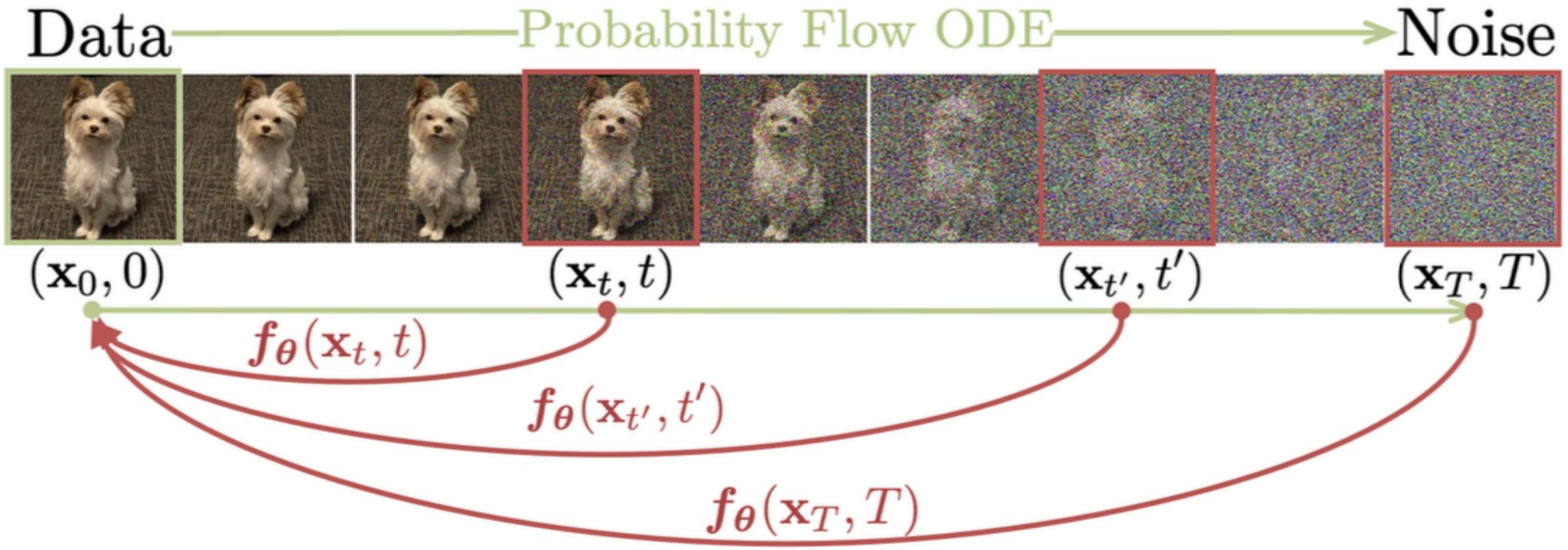
This will allow us to generate novel images by sampling from the latent space!

Why do diffusion models work?

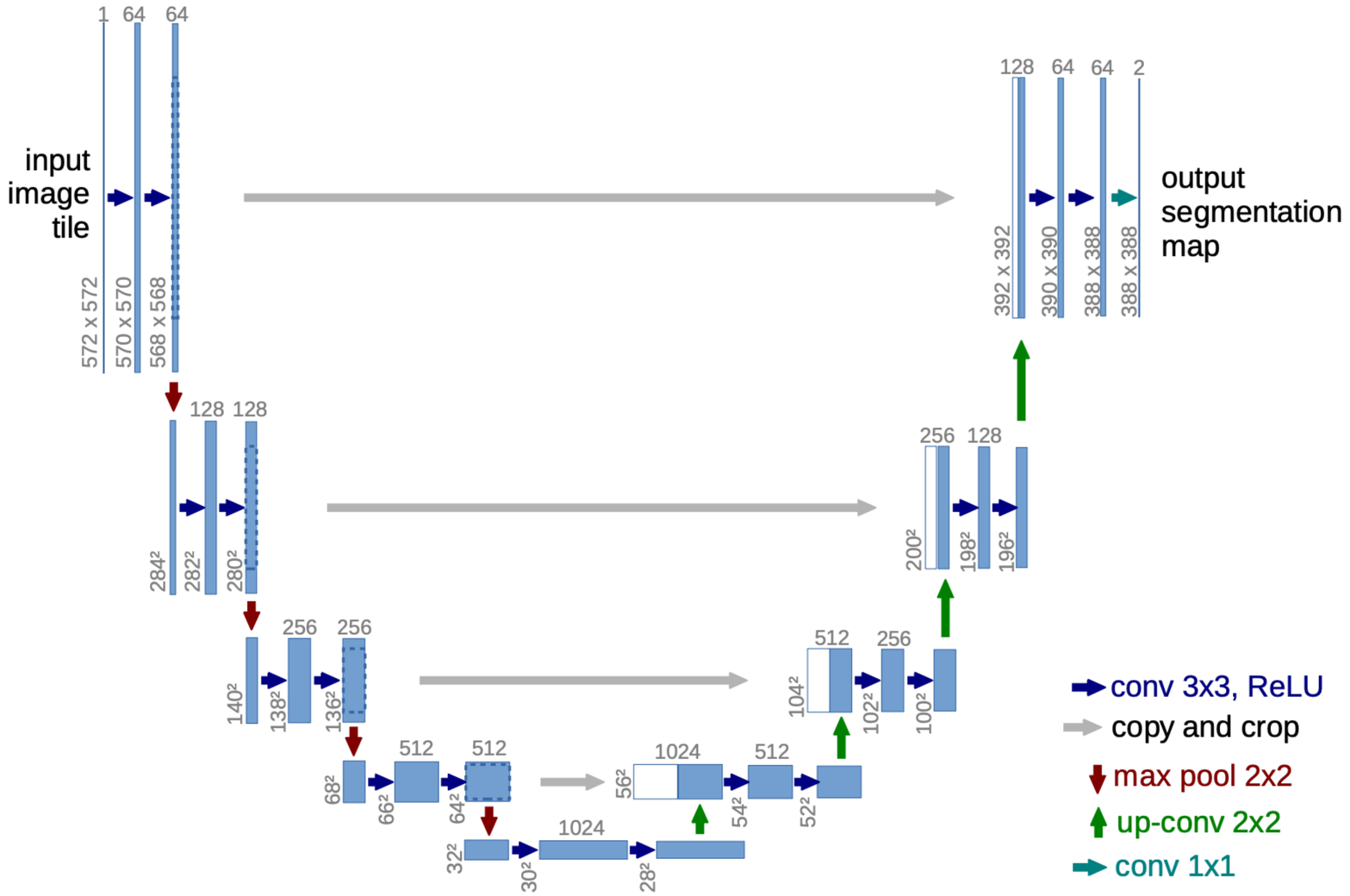
- Valid prior by construction: we design the forward process to be indistinguishable from random Gaussian noise, so there are no “holes” in the prior
- Easy local denoising: in contrast to a denoising autoencoder, each step of noising is relatively minor, making the individual prediction problems easier
- Shared parameters: the reverse model is shared across all timesteps, so it can learn from different levels of noise and be trained more efficiently

Generations from diffusion models

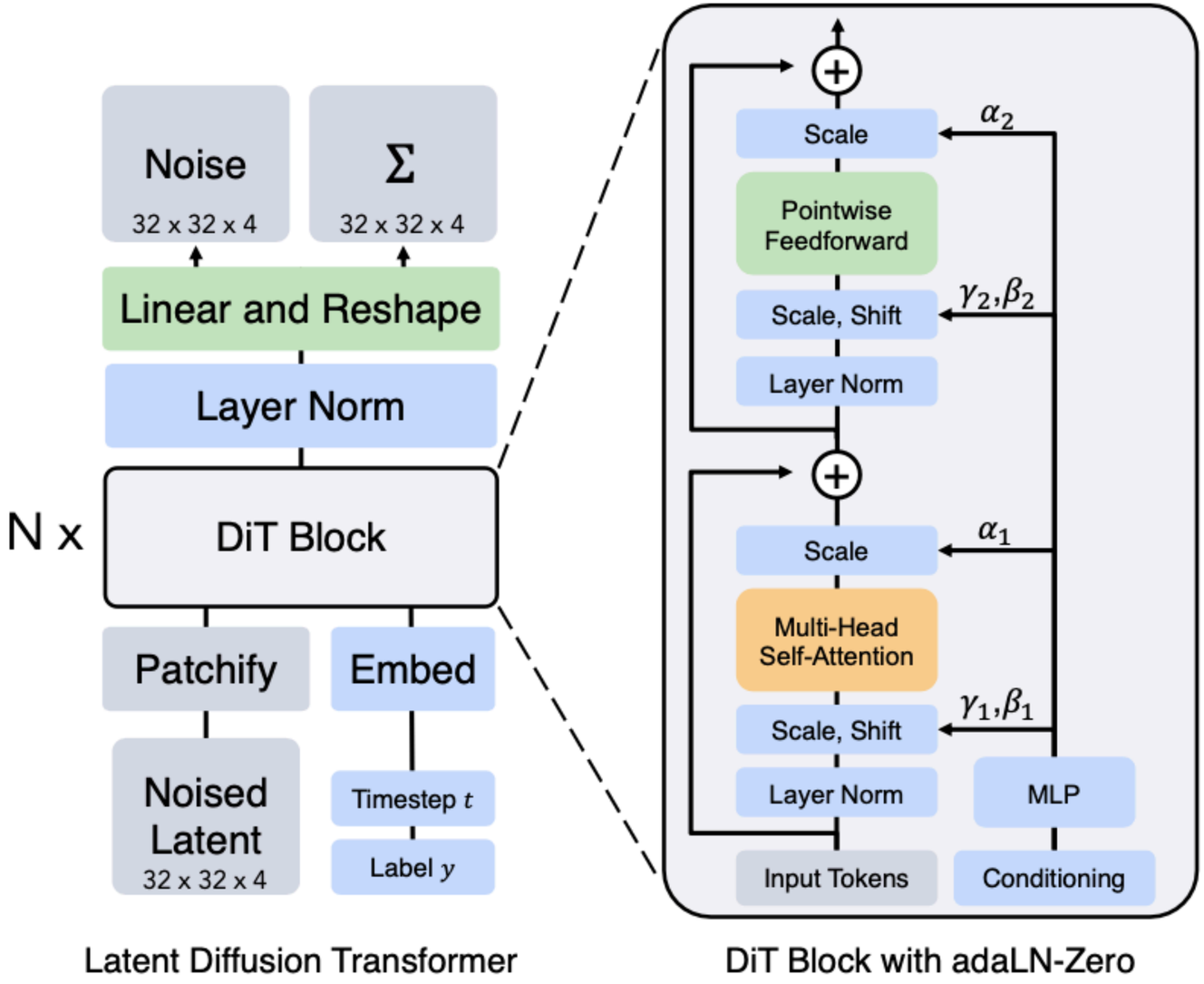
Consistency Models



Diffusion Model Architectures: U-Net



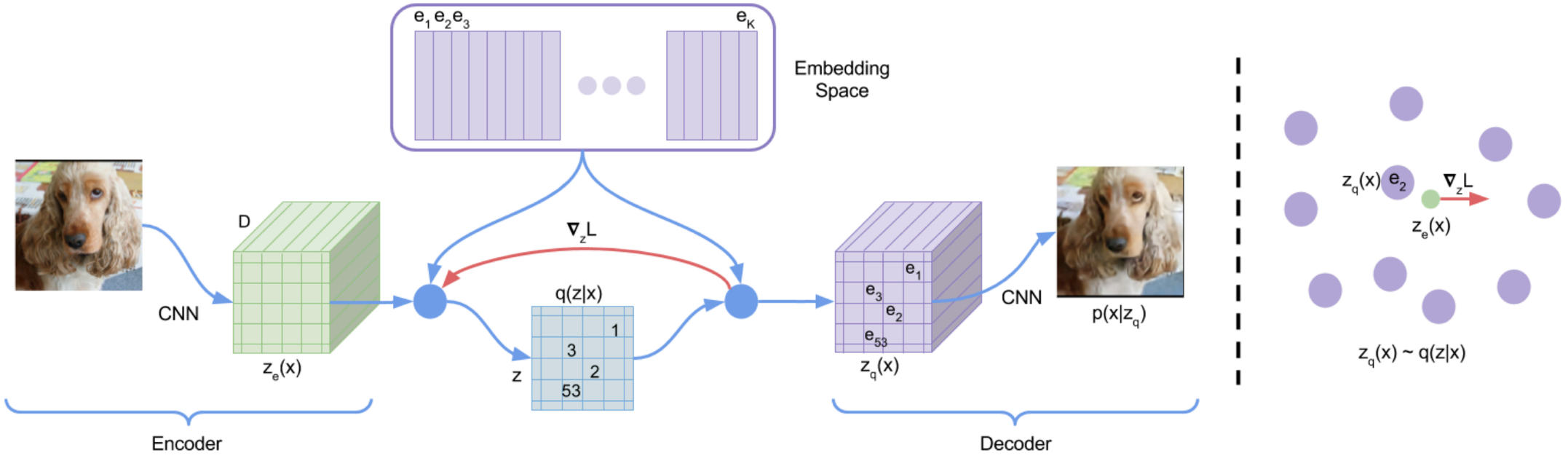
Diffusion Model Architectures: Diffusion Transformer



Latent Diffusion Transformer

DiT Block with adaLN-Zero

Tokenizing Images: VQ-VAE



Recap: Generative Models

Approach	Objective	Pros	Cons
Bottleneck Autoencoder	Reconstruction Loss	Easiest to implement Compression	Not good for generation “Holes” in latent space
VAE	ELBO: Reconstruction Loss + KL Divergence	Principled approach Smooth interpolation	Blurry outputs
GAN	Minimax: Generator vs. Discriminator	Sharp samples Fast generation	Unstable training Mode collapse
Diffusion	MSE on Noise Prediction	Best quality samples No mode collapse	Slow sampling Expensive to train

Bonus: Contrastive Learning

Motivation: Representation Learning

- Recall: one reason for generative modeling is to learn good representations that can then be used for a variety of downstream applications (e.g., training a classifier)

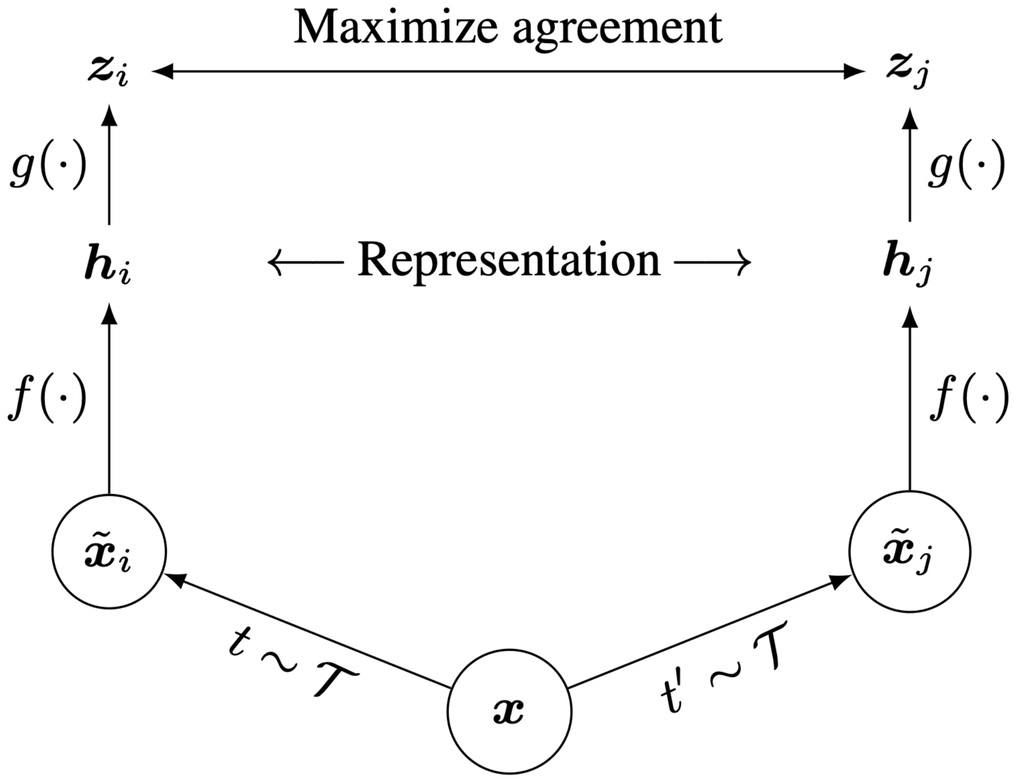
- Contrastive learning is another strategy for representation learning

- Key idea: learn what datapoints is similar, and which ones aren't

SimCLR

- Given an input, sample two augmentations of that input from a fixed set of perturbations
- Repeat T times, and then optimize the bipartite matching between 2T images:

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



SimCLR

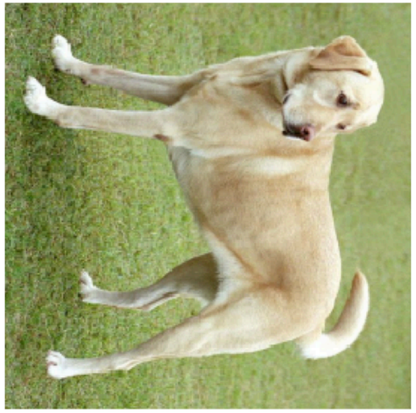
(a) Original

(b) Crop and resize

(c) Crop, resize (and flip)

(d) Color distort. (drop)

(e) Color distort. (jitter)



(f) Rotate $\{90°, 180°, 270°\}$

(g) Cutout

(h) Gaussian noise

(i) Gaussian blur

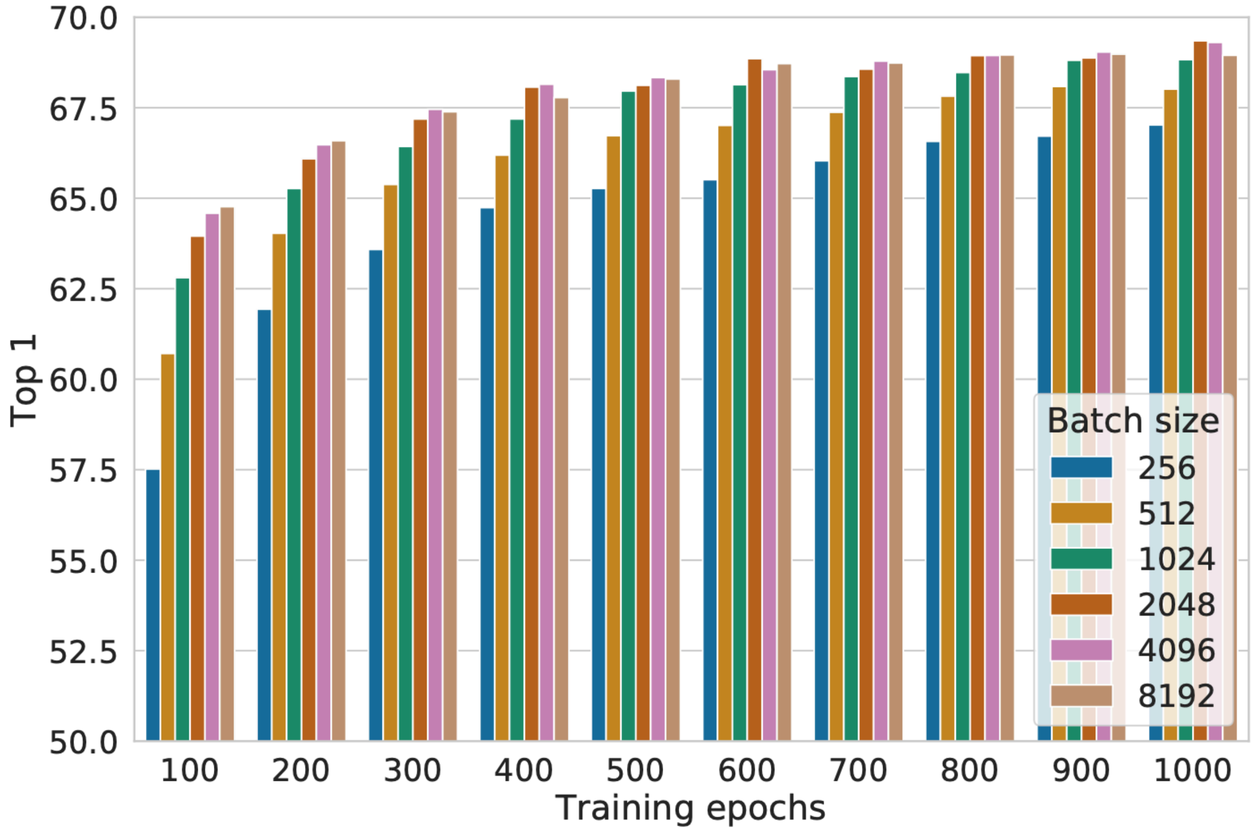
(j) Sobel filtering

Key Finding: Increase Batch Size!

- Can evaluate representations by training a linear classifier on top of them (e.g., for image classification)

- General trend: the usefulness of the learned representations depends on the difficulty of the contrastive learning objective

- Higher batch size makes the contrastive objective harder



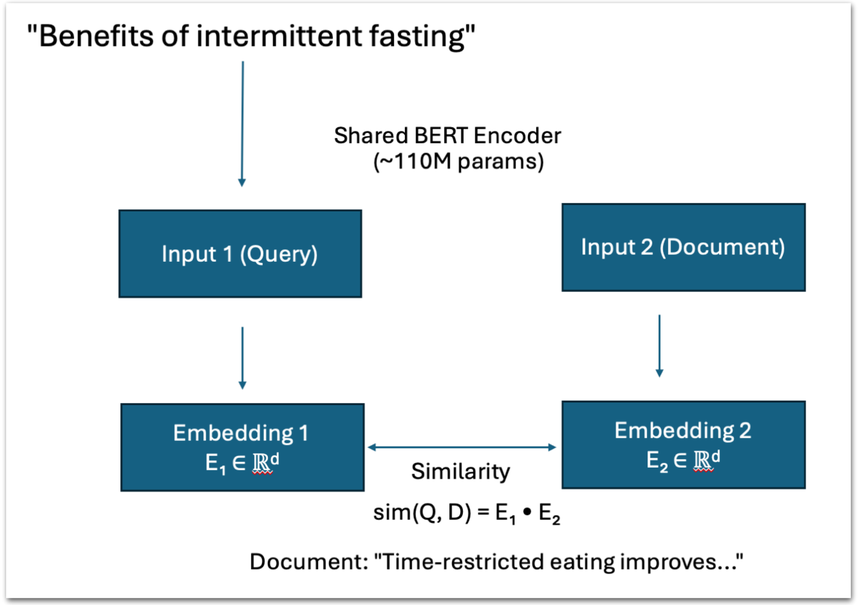
Dense Passage Retrieval

- The task: given a question and a corpus of documents, find the document that contains the answer to the question (commonly called RAG)

- The approach: embed both the question and the documents in shared vector space, and choose the documents which are closest in vector space to the question

Dense Passage Retrieval

- The task: given a question and a corpus of documents, find the document that contains the answer to the question (commonly called RAG)
- The approach: embed both the question and the documents in shared vector space, and choose the documents which are closest in vector space to the question
- **The problem:** need an objective to train the embedding model



Dense Passage Retrieval

How the contrastive objective works:

- Positive examples: we are given these in the dataset (could we learn them?)
- Negative examples: randomly generate these

$$
\begin{aligned}
& L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) \qquad (2) \\
= \quad & -\log \frac{e^{\mathrm{sim}(q_i, p_i^+)}}{e^{\mathrm{sim}(q_i, p_i^+)} + \sum_{j=1}^{n} e^{\mathrm{sim}(q_i, p_{i,j}^-)}}.
\end{aligned}
$$

Dense Passage Retrieval

How the contrastive objective works:

- Positive examples: we are given these in the dataset (could we learn them?)
- Negative examples: randomly generate these

$$L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) \tag{2}$$

But it turns out we can do better:

$$= -\log \frac{e^{\text{sim}(q_i, p_i^+)}}{e^{\text{sim}(q_i, p_i^+)} + \sum_{j=1}^{n} e^{\text{sim}(q_i, p_{i,j}^-)}}.$$

- Instead of randomly sampling the negative documents,
  use a strong rule-based baseline (BM25) to generate “hard negatives”

Dense Passage Retrieval

How the contrastive objective works:

- Positive examples: we are given these in the dataset (could we learn them?)
- Negative examples: randomly generate these

$$L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) \tag{2}$$

$$= -\log \frac{e^{\text{sim}(q_i, p_i^+)}}{e^{\text{sim}(q_i, p_i^+)} + \sum_{j=1}^{n} e^{\text{sim}(q_i, p_{i,j}^-)}}.$$

But it turns out we can do better:

- Instead of randomly sampling the negative documents,
  use a strong rule-based baseline (BM25) to generate “hard negatives”
- Can do multiple iterations of training and use the model itself to generate negatives

Recap: Contrastive Learning

- Can be a useful alternative approach to learning flexible representations
- How to evaluate models:
  - Use the learned embeddings as input to a classification task
  - Run simple linear classifiers over the representations to “interpret” them
- How to generate negatives:
  - Want to make the classification task as difficult as possible
  - Can do this in many ways: increase batch size, mining for “hard negatives”, etc.