Unsupervised Learning And Generative Models

Slides from : Sergey Levine UC Berkeley
& **Dhruv Batra**
& **CS231n**

Overview

- Unsupervised Learning

- Generative Models
  - PixelRNN and PixelCNN
  - Variational Autoencoders (VAE)
  - Generative Adversarial Networks (GAN)

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x → 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 → y

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

Cat

Classification

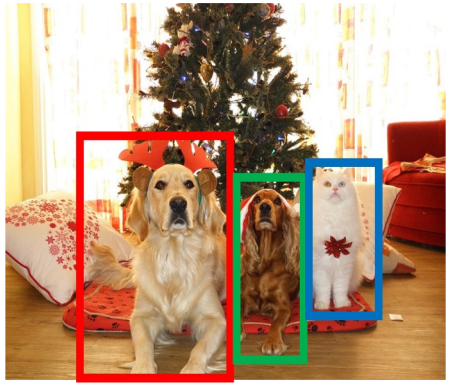
Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x → y

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



DOG, **DOG**, **CAT**

Object Detection

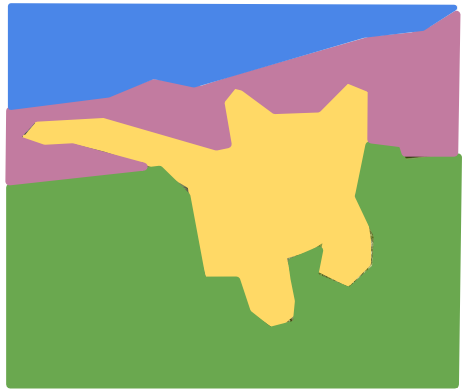
Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x → 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 → 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

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

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

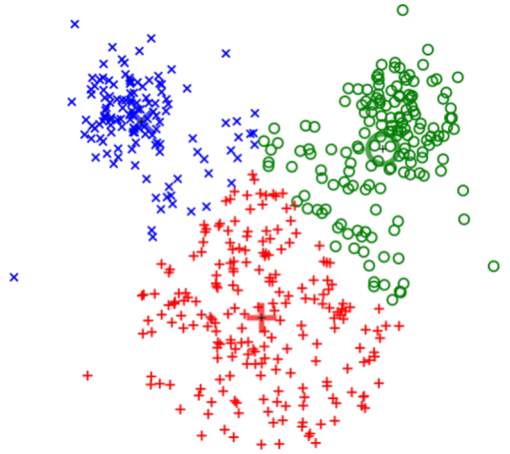
Supervised vs Unsupervised Learning

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

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



K-means clustering

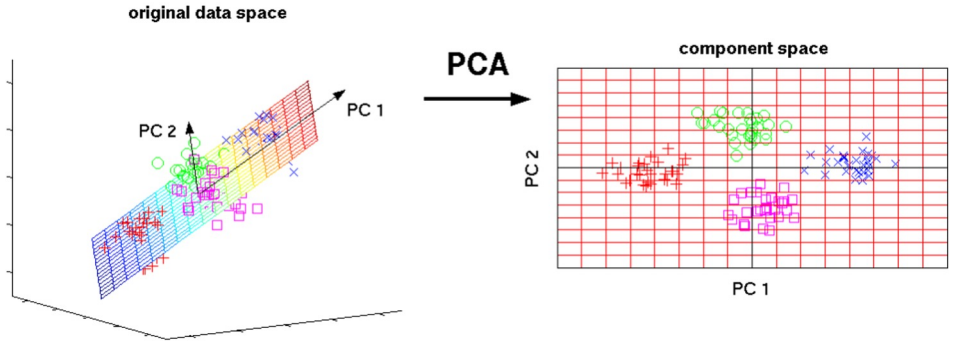
Supervised vs Unsupervised Learning

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

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



Principal Component Analysis
(Dimensionality reduction)

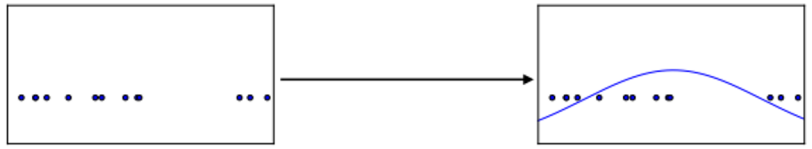
Supervised vs Unsupervised Learning

Unsupervised Learning

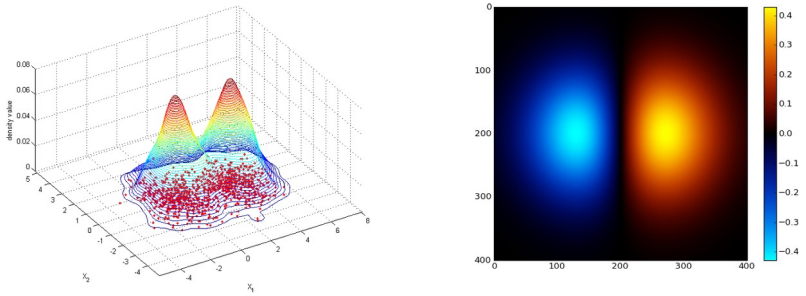
Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

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



1-d density estimation

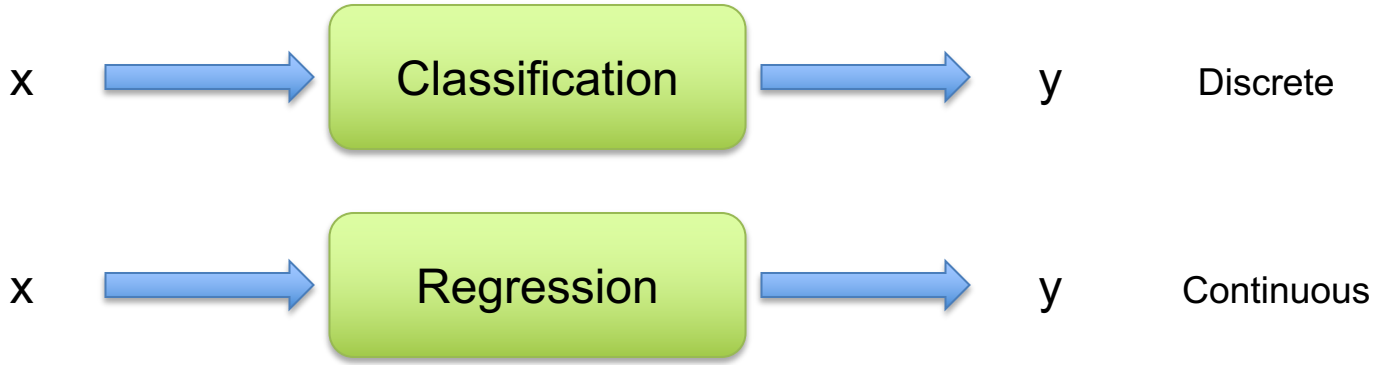


2-d density estimation

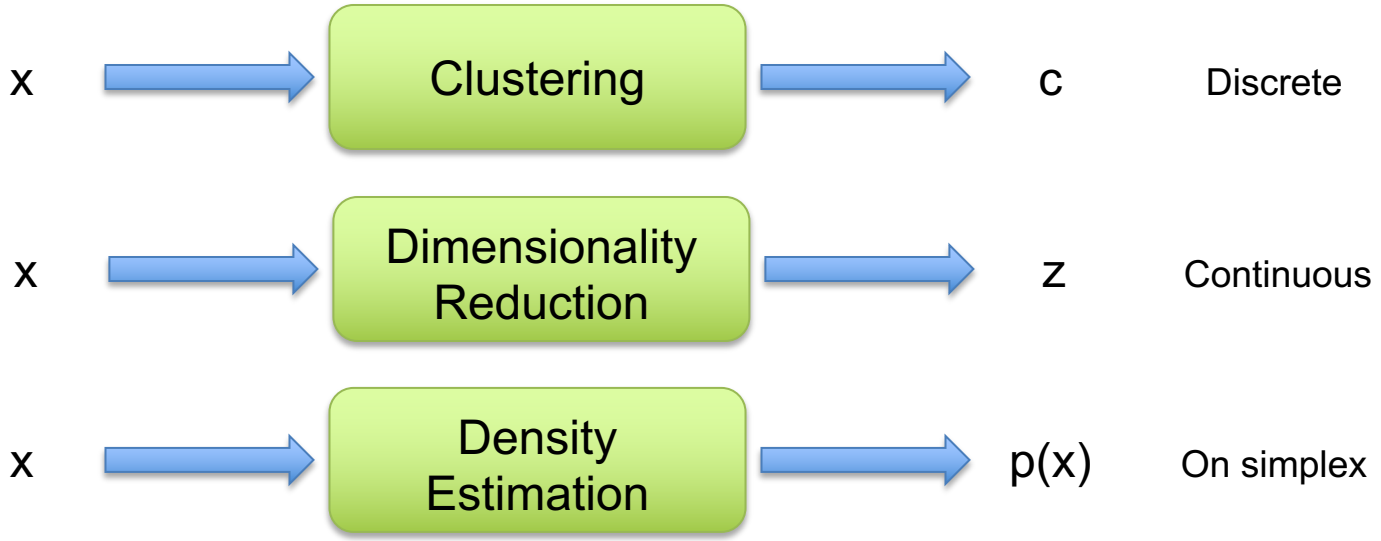
2-d density images left and right are CC0 public domain

Tasks

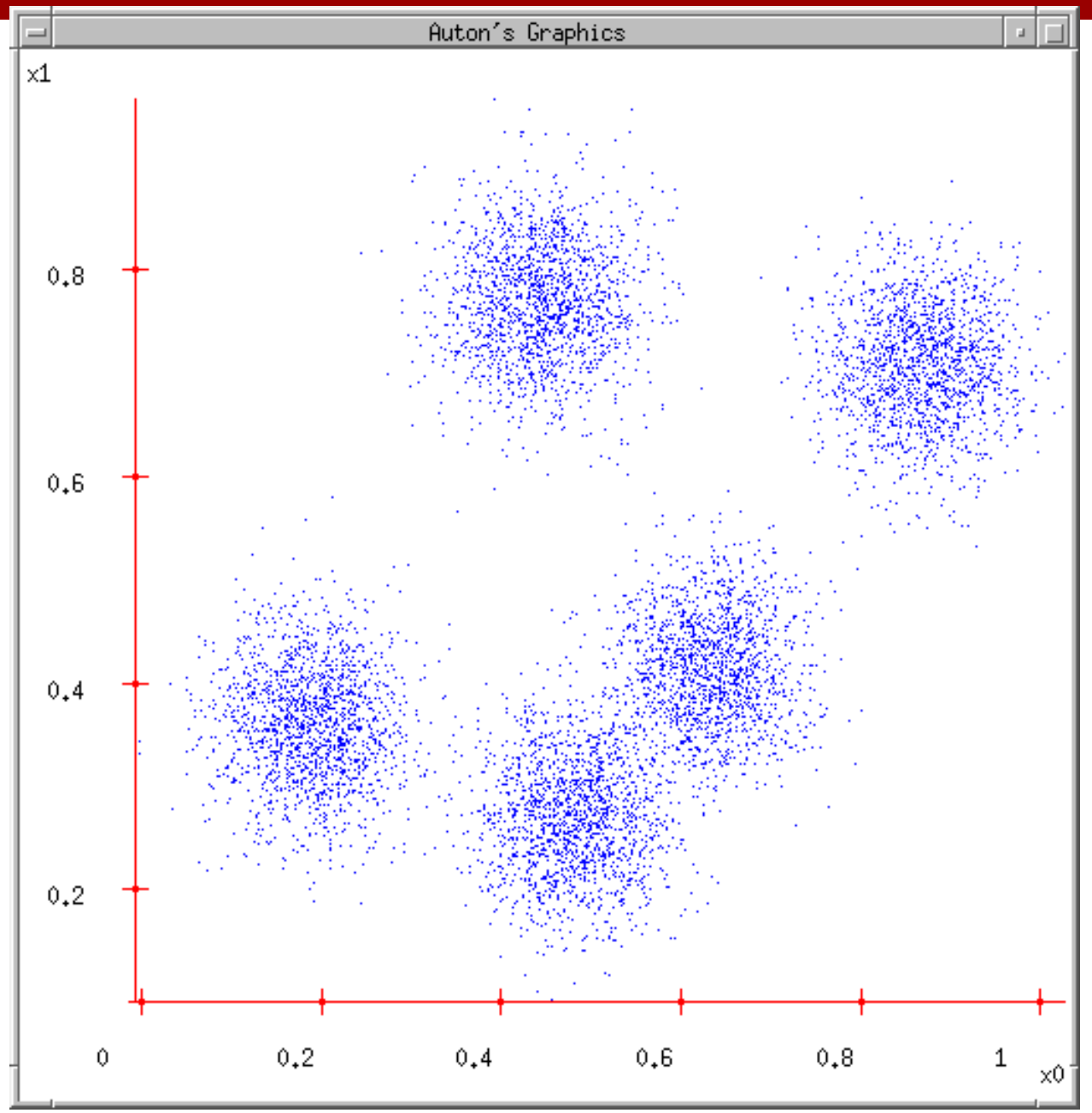
Supervised Learning



Unsupervised Learning

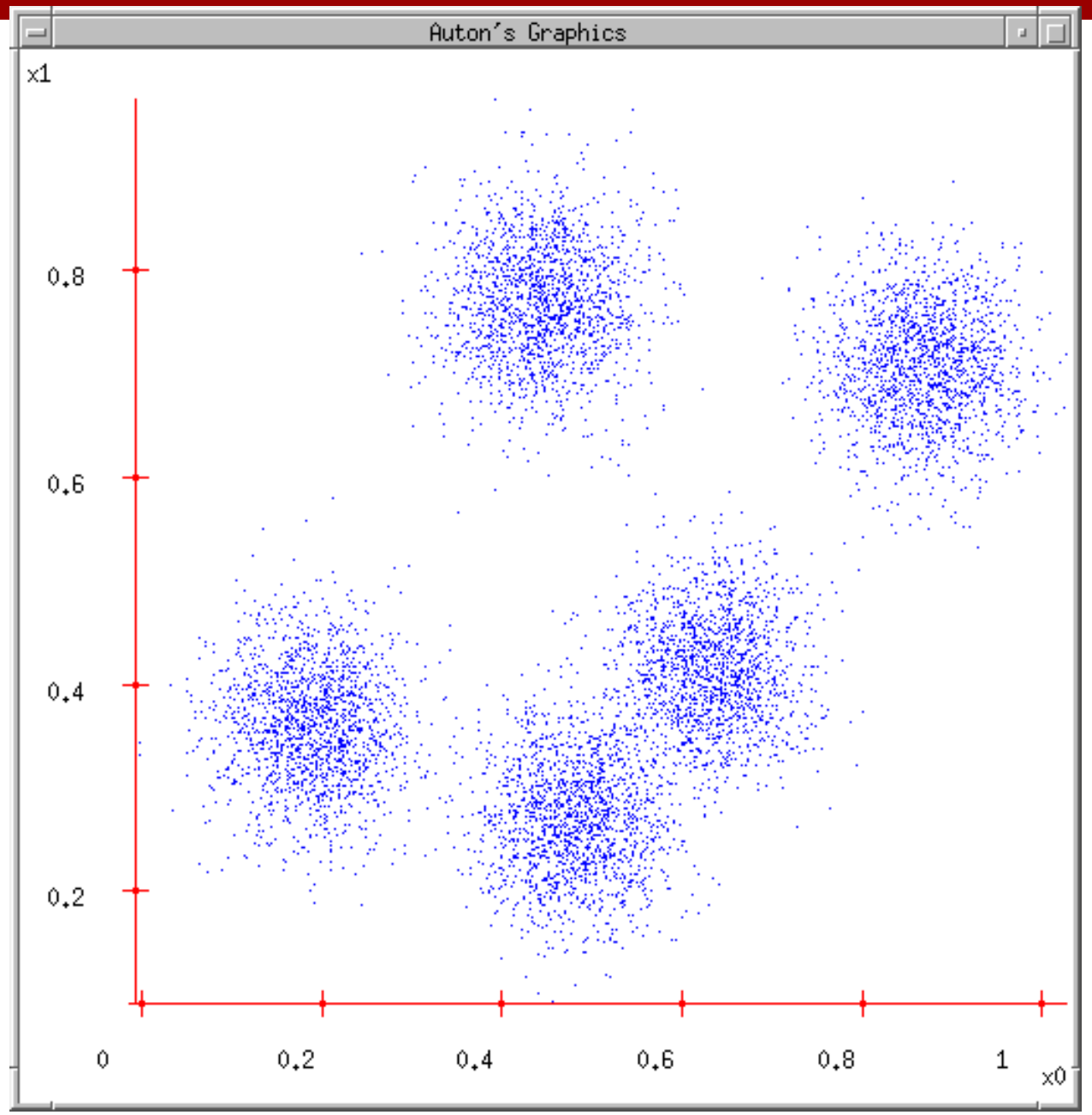


Some Data



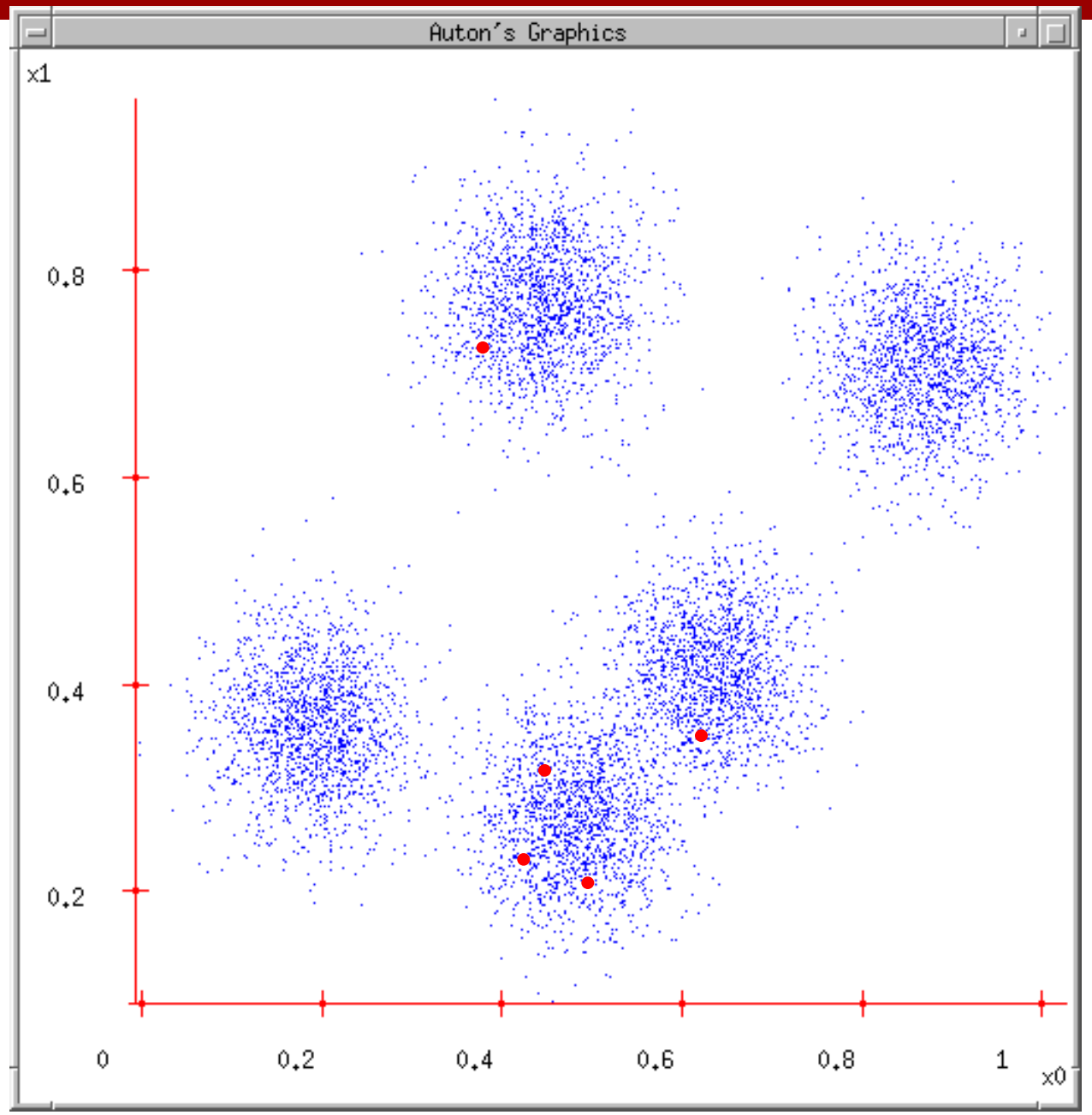
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*



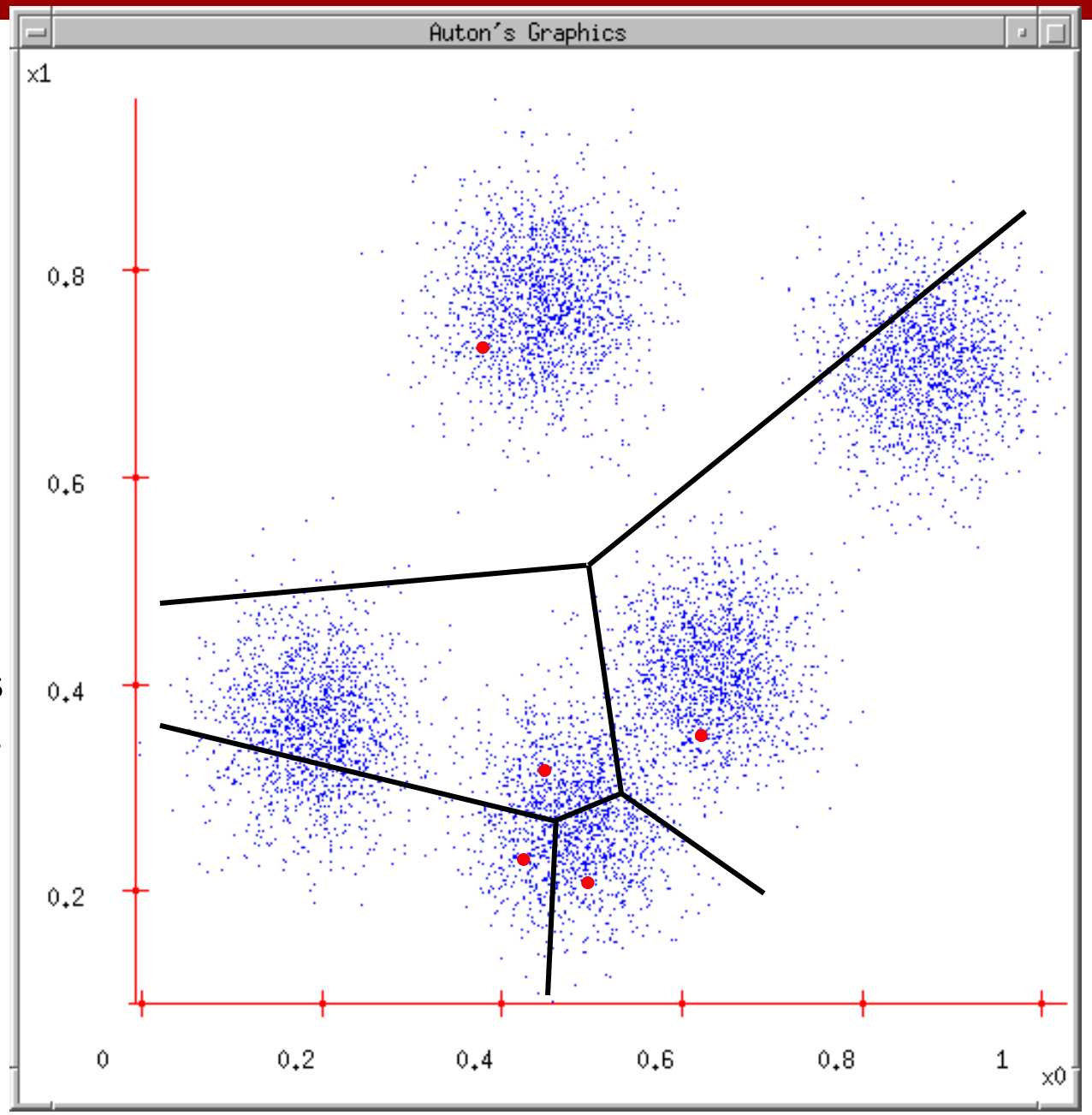
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations



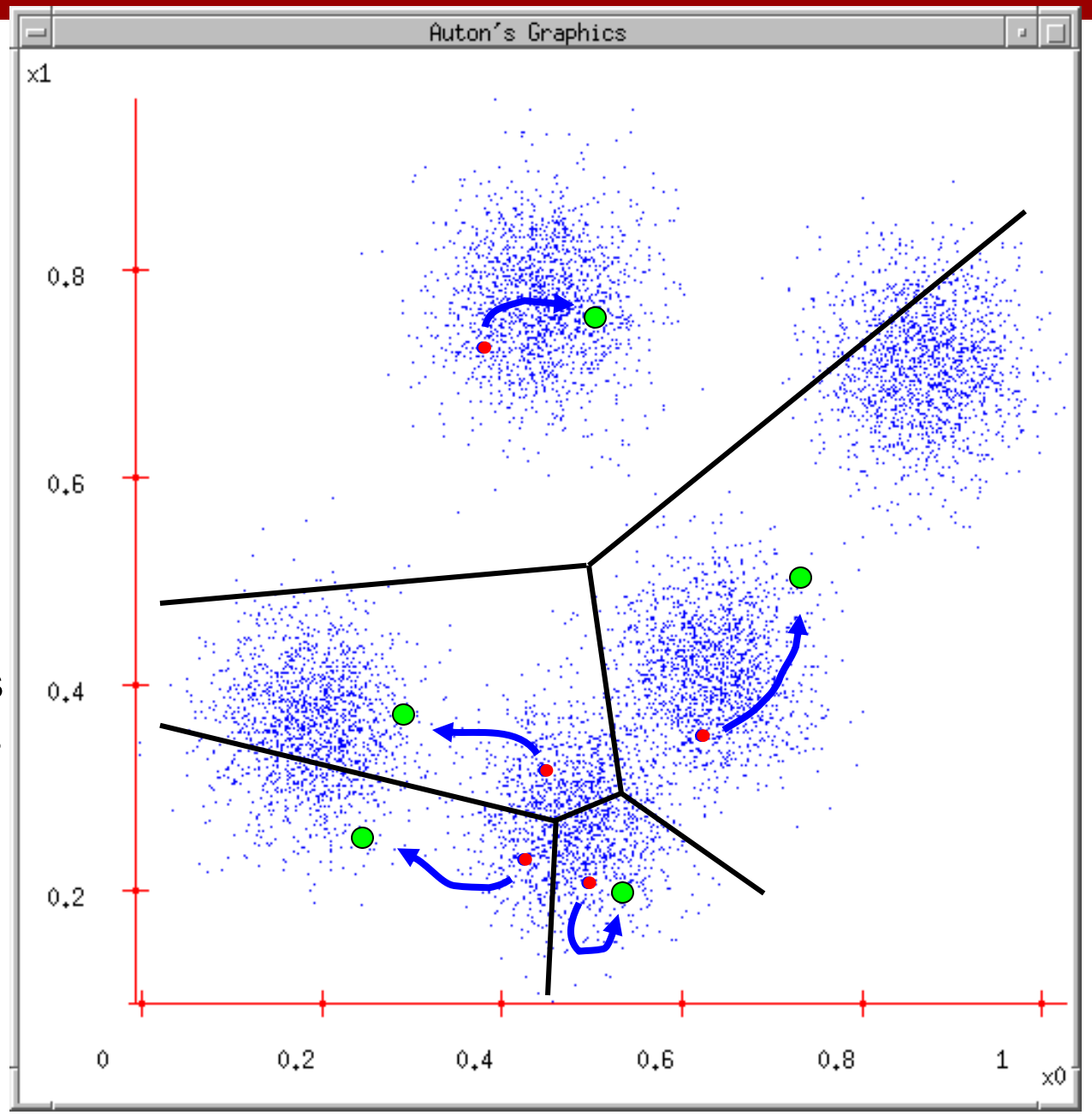
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



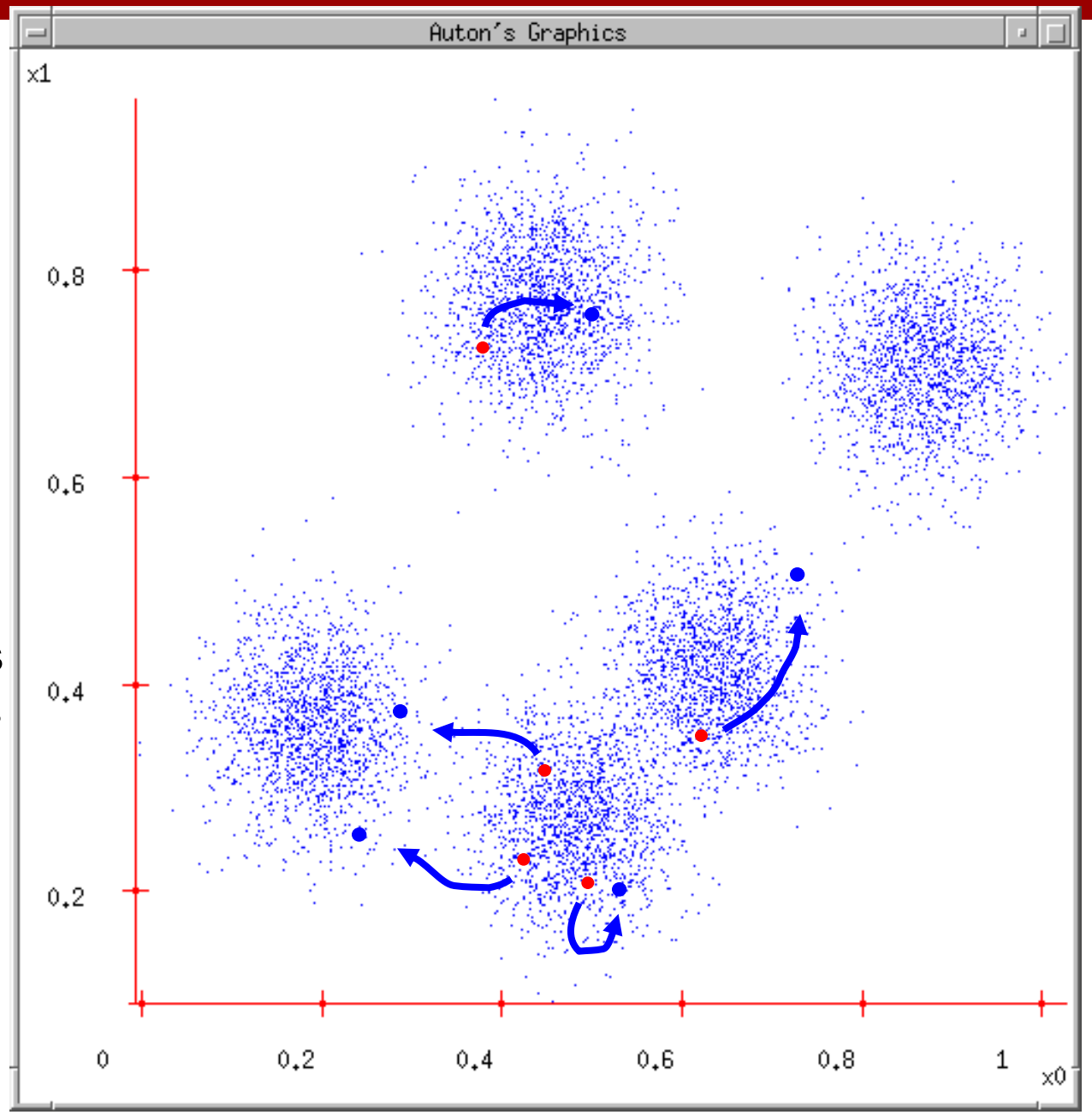
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!



K-means

- Randomly initialize *k* centers
  - $\square^{(0)} = \square_1^{(0)}, \ldots, \square_k^{(0)}$

- **Assign**:
  - Assign each point i∈{1,…n} to nearest center:
  - $$C(i) \longleftarrow \underset{j}{\operatorname{argmin}} ||\mathbf{x}_i - \boldsymbol{\mu}_j||^2$$

- **Recenter**:
  - $\square_j$ becomes centroid of its points

K-means

- Demo
  - http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html

What is K-means optimizing?

- Objective F(□,C): function of centers □ and point allocations C:
  - $$F(\boldsymbol{\mu}, C) = \sum_{i=1}^{N} ||\mathbf{x}_i - \boldsymbol{\mu}_{C(i)}||^2$$
  - 1-of-k encoding

$$F(\boldsymbol{\mu}, \boldsymbol{a}) = \sum_{i=1}^{N} \sum_{j=1}^{k} a_{ij} ||\mathbf{x}_i - \boldsymbol{\mu}_j||^2$$

- Optimal K-means:
  - $\min_{□} \min_{a} F(□, a)$

Coordinate descent algorithms

- Want: $\min_a \min_b F(a,b)$

- Coordinate descent:
  - fix a, minimize b
  - fix b, minimize a
  - repeat

- Converges!!!
  - if F is bounded
  - to a (often good) local optimum
    - as we saw in applet (play with it!)

- K-means is a coordinate descent algorithm!

K-means as Co-ordinate Descent

- Optimize objective function:

$$\min_{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_k} \min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_N} F(\boldsymbol{\mu}, \boldsymbol{a}) = \min_{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_k} \min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_N} \sum_{i=1}^{N}\sum_{j=1}^{k} a_{ij}||\mathbf{x}_i - \boldsymbol{\mu}_j||^2$$

- Fix □, optimize a (or C)

K-means as Co-ordinate Descent

- Optimize objective function:

$$\min_{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_k} \min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_N} F(\boldsymbol{\mu}, \boldsymbol{a}) = \min_{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_k} \min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_N} \sum_{i=1}^{N}\sum_{j=1}^{k} a_{ij}||\mathbf{x}_i - \boldsymbol{\mu}_j||^2$$

- Fix a (or C), optimize □

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x -> y

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

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

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

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map x -> y

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

Unsupervised Learning

Training data is cheap

Data: x

Just data, no labels!

Holy grail: Solve unsupervised learning => understand structure of visual world

Goal: Learn some underlying hidden *structure* of the data

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

Generative Models

Given training data, generate new samples from same distribution

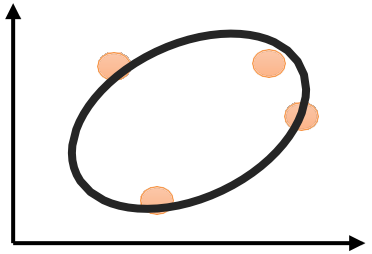
Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Probabilistic models

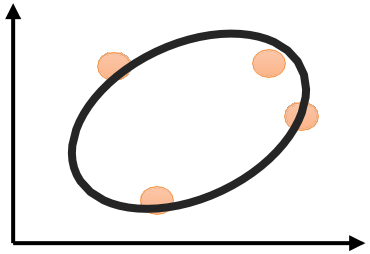
$p(x)$



Why would we want to do this?

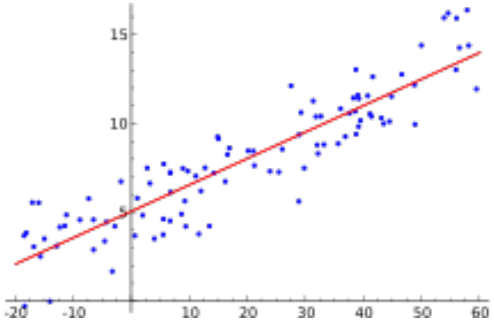
Probabilistic models

$p(x)$



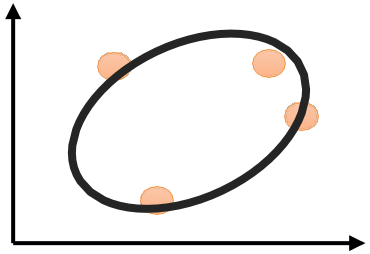
Why would we want to do this?

$p(y|x)$



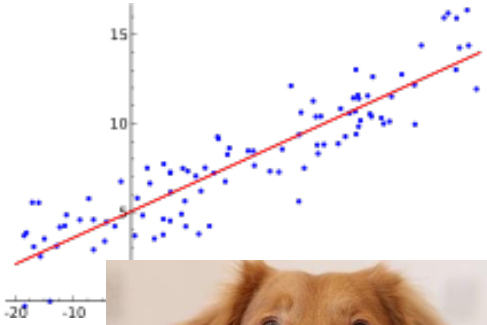
Probabilistic models

$p(x)$



Why would we want to do this?

$p(y|x)$



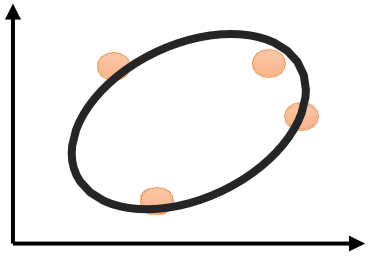
$f_\theta(x) = y$

[object label]

Generative models

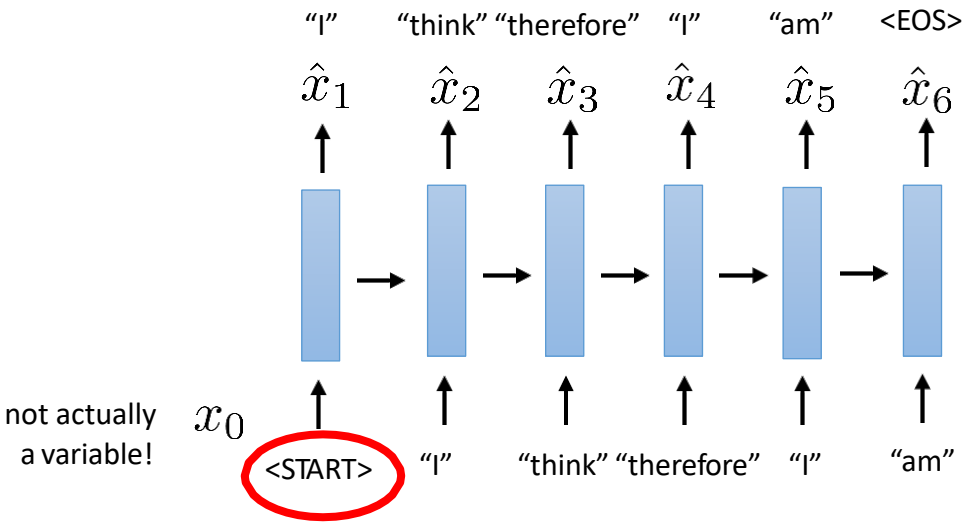
$p(x)$

$\mathcal{N}(\mu, \sigma)$



Just different ways to solve the same problem!

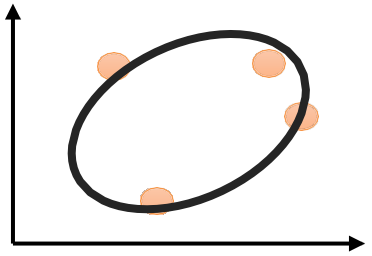
$$p(x) = p(x_1|x_0)p(x_2|x_{0:1})p(x_3|x_{0:2})p(x_4|x_{0:3})p(x_5|x_{0:4})p(x_6|x_{0:5})$$



Generative models

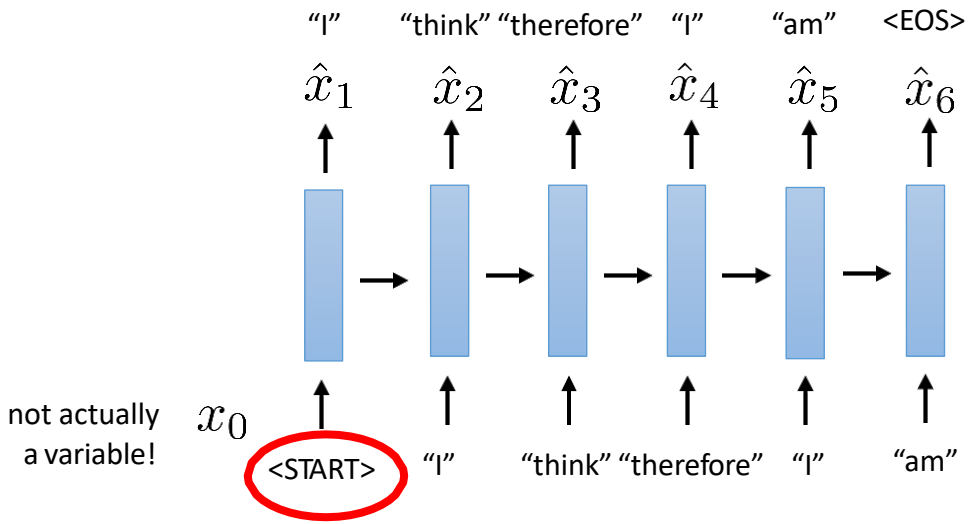
$p(x)$

$\mathcal{N}(\mu, \sigma)$



Just different ways to solve the same problem!

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$



Why would we want to do this?

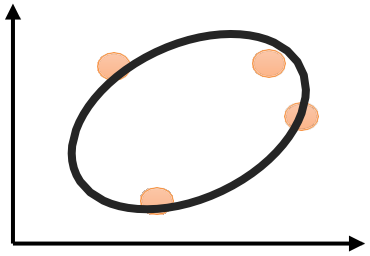
Same reasons as language modeling!

- Unsupervised pretraining on lots of data
- Representation learning
- Pretraining for later finetuning
- Actually generating things!

Generative models

$p(x)$

$\mathcal{N}(\mu, \sigma)$

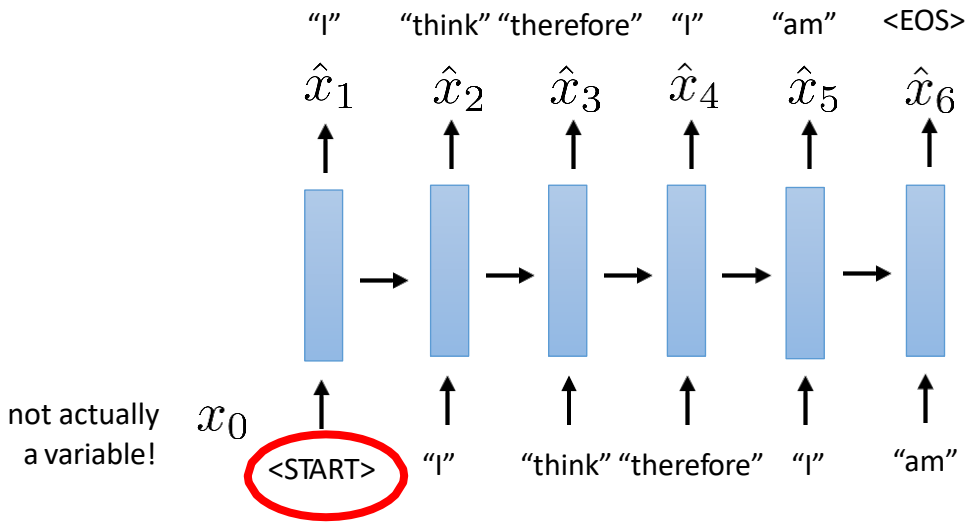


Just different ways to solve the same problem!

Today: can we go from "language models" to "everything models"?

This is called **unsupervised learning**

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$

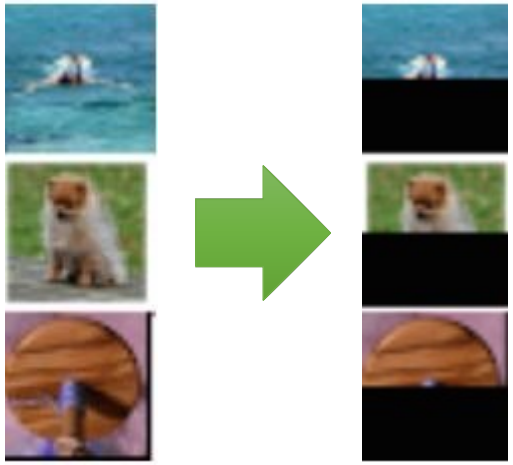


Why would we want to do this?

Same reasons as language modeling!

- Unsupervised pretraining on lots of data
- Representation learning
- Pretraining for later finetuning
- Actually generating things!

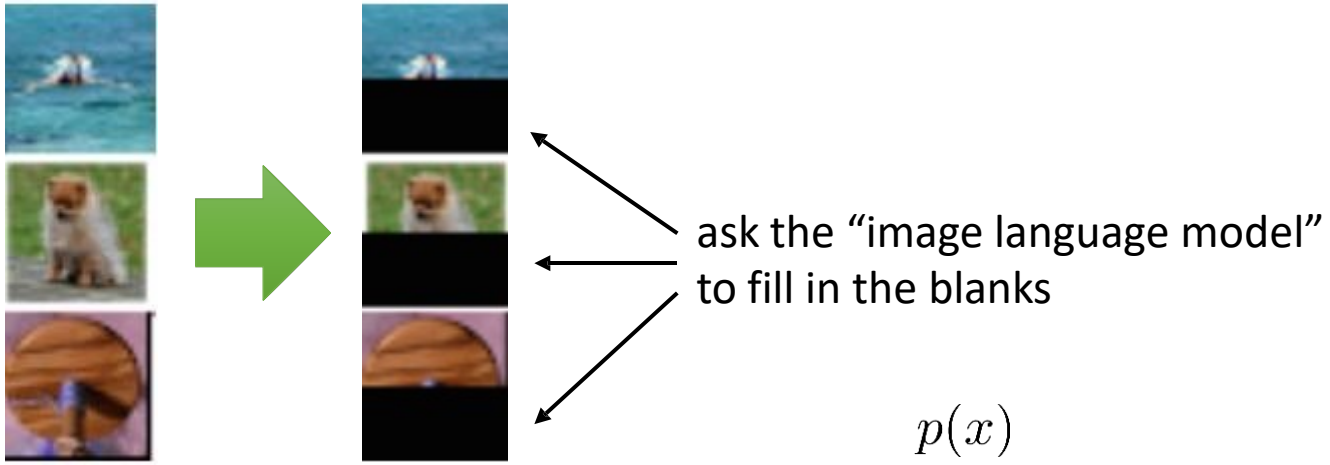
Can we “language model” images?



$$p(x)$$

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

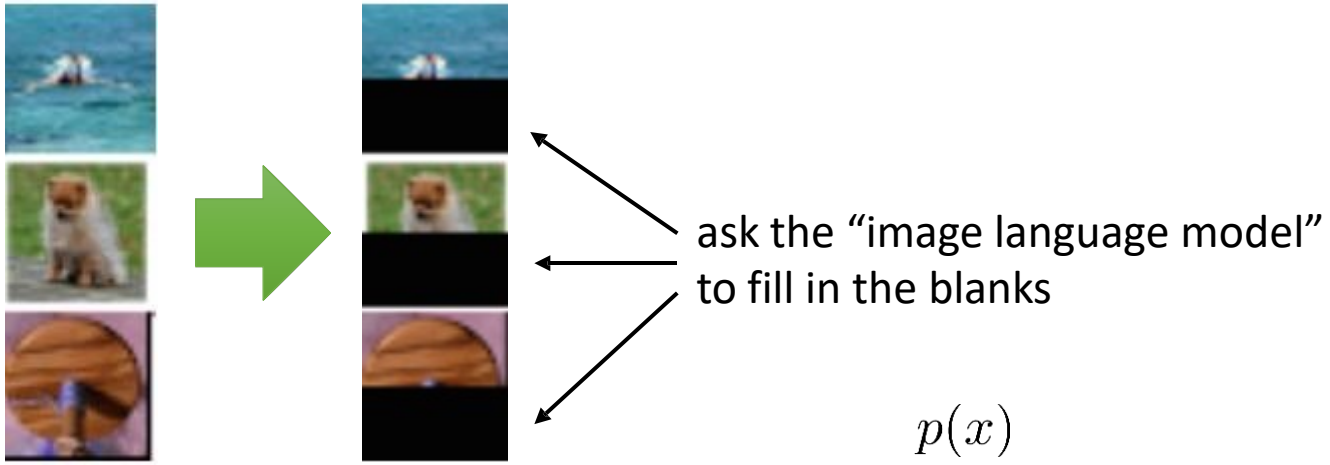
Can we “language model” images?



$$p(x)$$

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

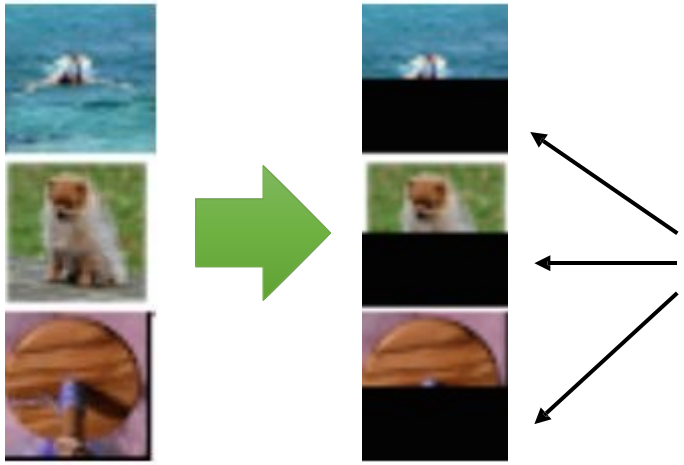
Can we “language model” images?



before	**now**
time step = word	time step = pixel

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

Can we “language model” images?



ask the “image language model”
to fill in the blanks

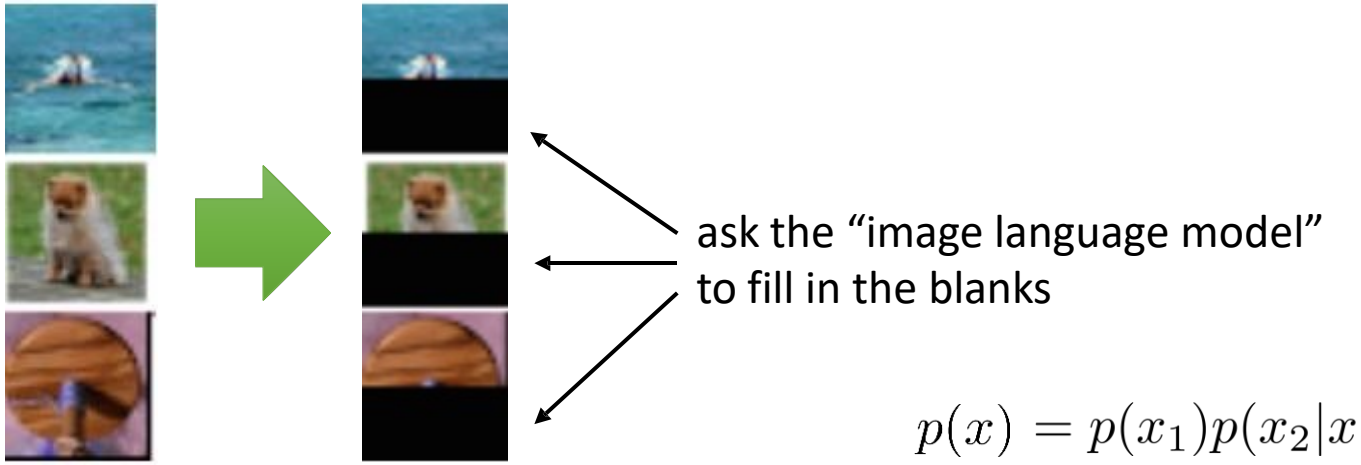
before	**now**
time step = word	time step = pixel

Factorize the p(x)

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$

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

Can we “language model” images?



ask the “image language model” to fill in the blanks

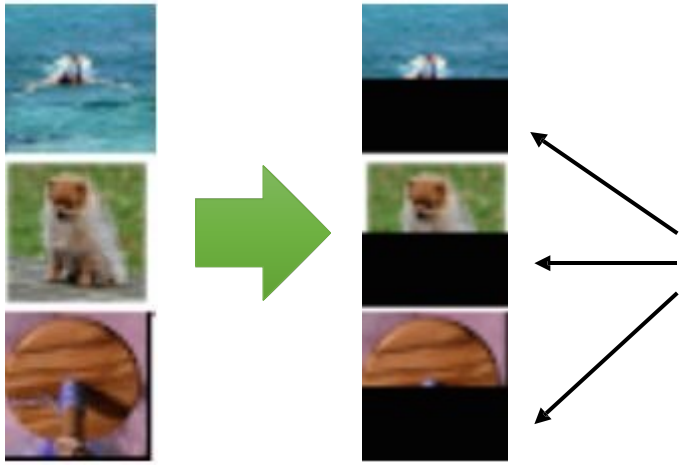
before	**now**
time step = word	time step = pixel

some (arbitrary) ordering on pixels

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$

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

Can we “language model” images?

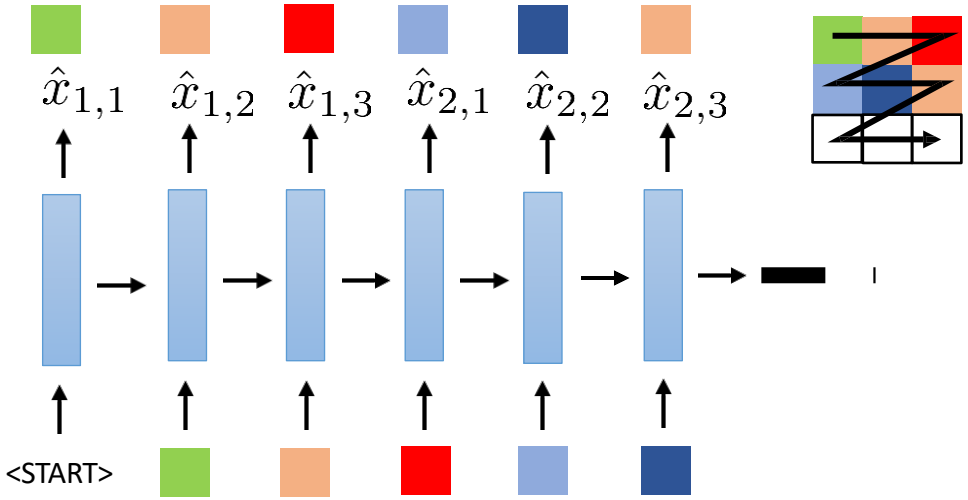


ask the “image language model” to fill in the blanks

before	**now**
time step = word	time step = pixel

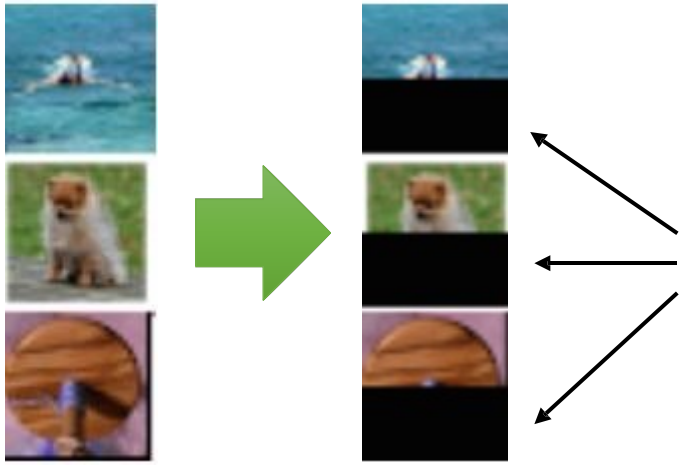
some (arbitrary) ordering on pixels

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$



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

Can we “language model” images?

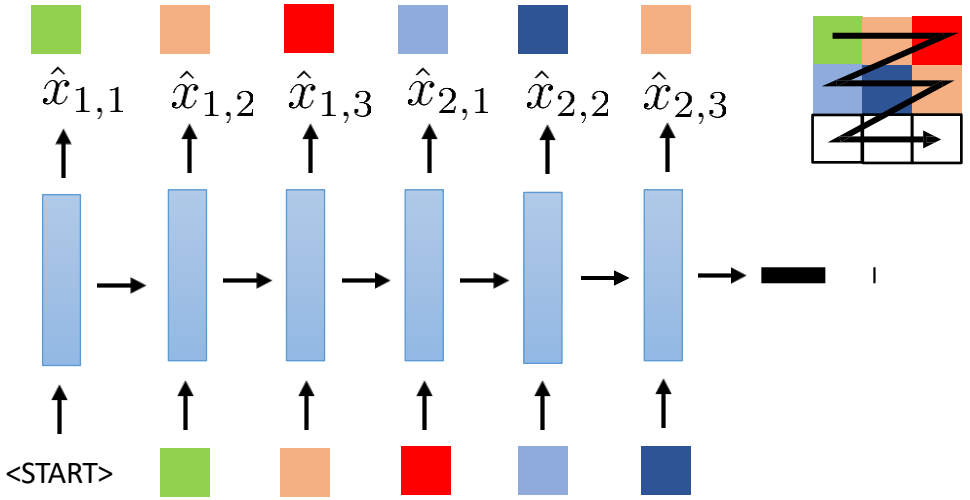


ask the “image language model” to fill in the blanks

before	**now**
time step = word	time step = pixel

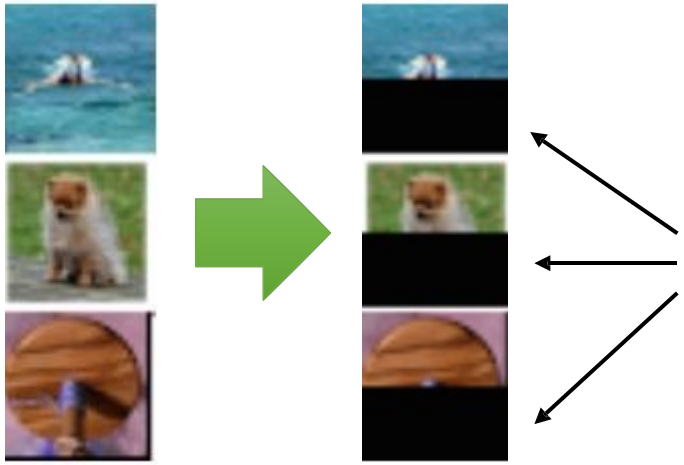
some (arbitrary) ordering on pixels

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$

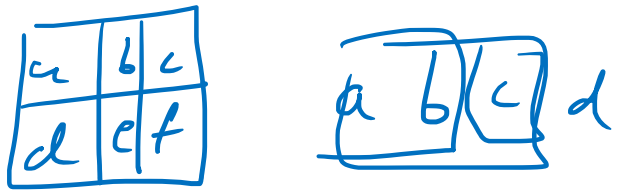


Van den Oord et al. **Pixel Recurrent Neural Networks**. 2016.

Can we “language model” images?



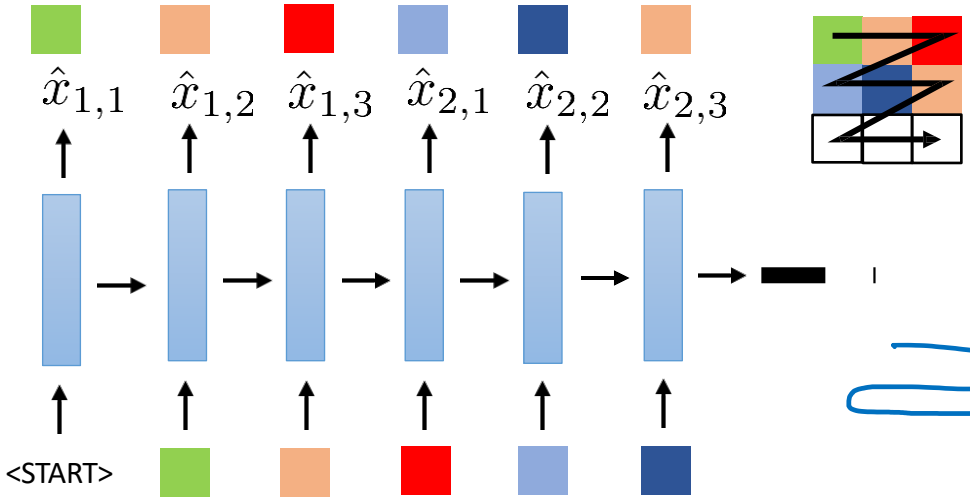
ask the “image language model” to fill in the blanks



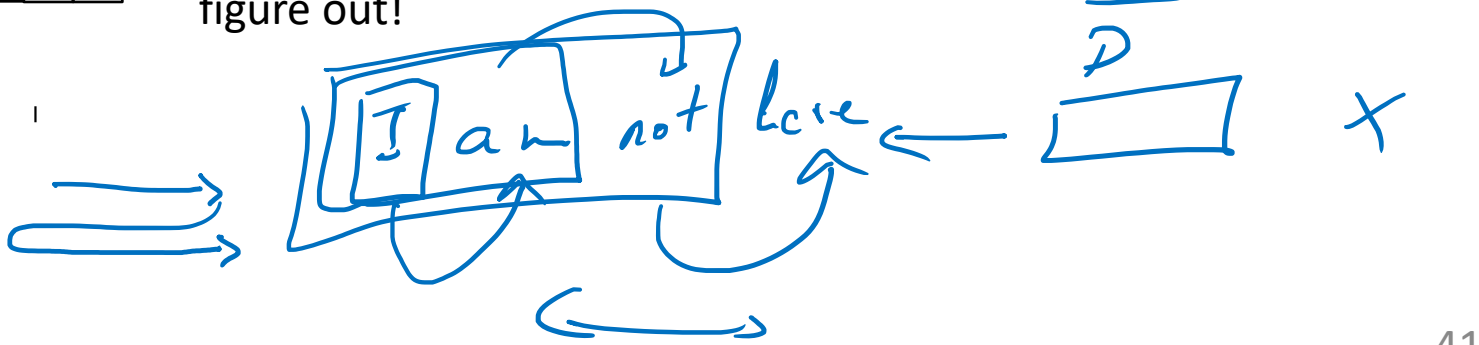
before	**now**
time step = word	time step = pixel

some (arbitrary) ordering on pixels

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$

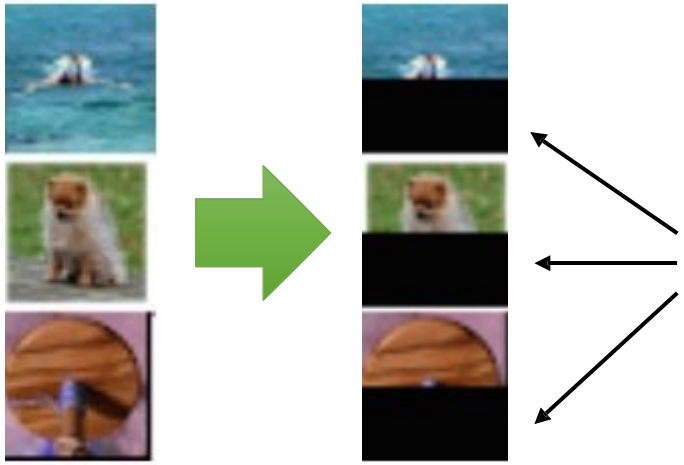


This is basically the main idea, but there are some details we need to figure out!



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

Can we “language model” images?

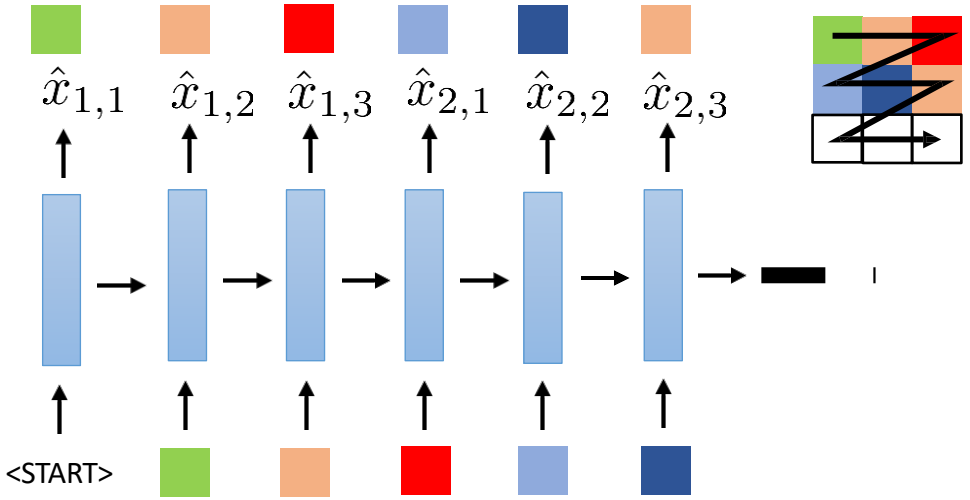


ask the “image language model” to fill in the blanks

before	**now**
time step = word	time step = pixel

some (arbitrary) ordering on pixels

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$



This is basically the main idea, but there are some details we need to figure out!

- How to order the pixels?
- What kind of model to use?

Many options, RNN, Bi-LSTM, et..

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

Autoregressive generative models

Main principle for training:

1. Divide up x into dimensions $x_1, \ldots, x_n$
2. Discretize each x_i into k values
3. Model $p(x)$ via the chain rule

   $$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$

4. Use your favorite sequence model to model $p(x)$

each of these is just a softmax

Using autoregressive generative models:

Sampling: ancestral sampling in sequence (x_1, then x_2, etc.)

Completion: feed in actual values for known x_i values

Representations: same idea as ELMo or BERT

Autoregressive generative models

Main principle for training:

1. Divide up x into dimensions $x_1, \ldots, x_n$
2. Discretize each x_i into k values
3. Model $p(x)$ via the chain rule

   $$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$

4. Use your favorite sequence model to model $p(x)$

each of these is just a softmax

Using autoregressive generative models:

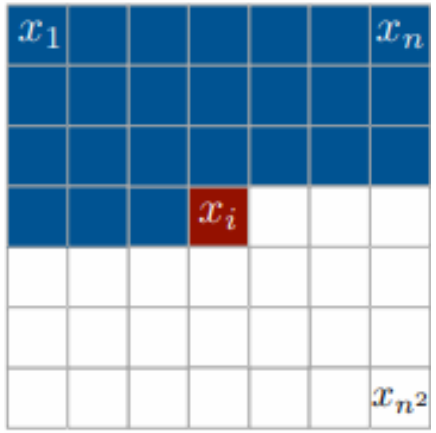
Sampling: ancestral sampling in sequence (x_1, then x_2, etc.)

Completion: feed in actual values for known x_i values

Representations: same idea as ELMo or BERT

We can use beam serach

PixelRNN



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

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

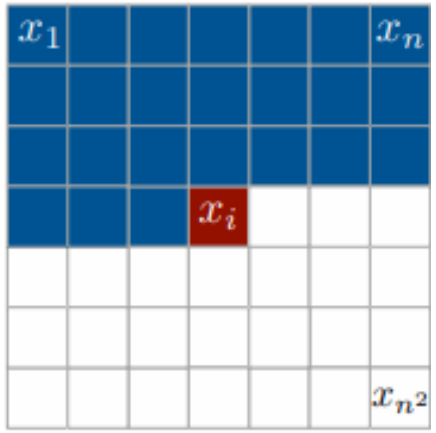
Generate one color channel at a time:

$$p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$

256-way softmax

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

PixelRNN



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

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

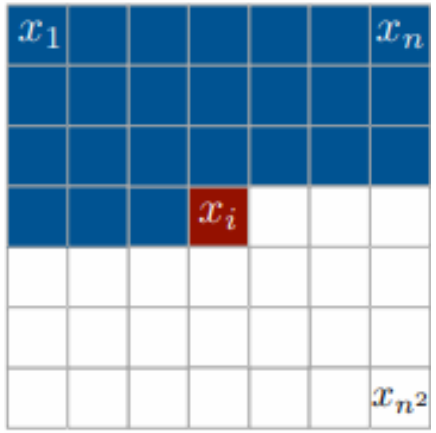
Generate one color channel at a time:

$$p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$

256-way softmax

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

PixelRNN



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

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

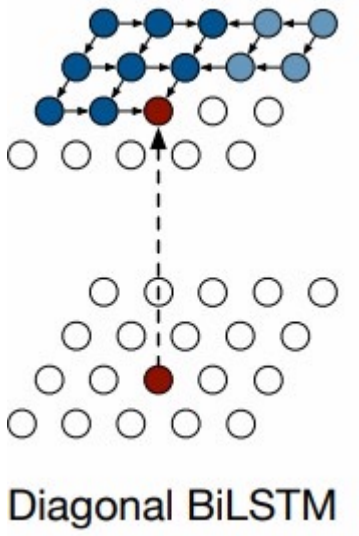
Generate one color channel at a time:

$$p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$

256-way softmax

Some practical considerations:

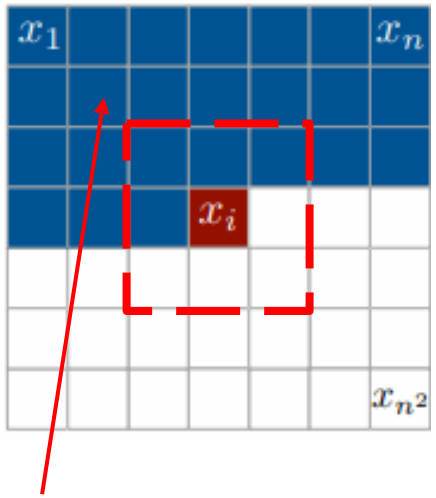
- It's very slow
- Row-by-row LSTMs might struggle to capture spatial context (pixels right above are "far away")
- Many practical improvements and better architectures are possible!



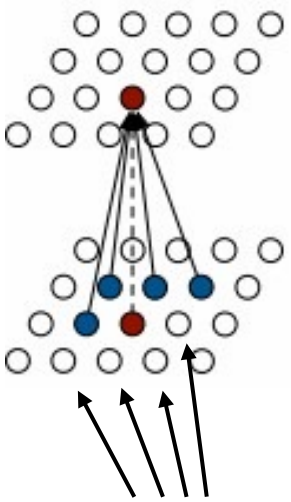
Van den Oord et al. **Pixel Recurrent Neural Networks**. 2016.

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?



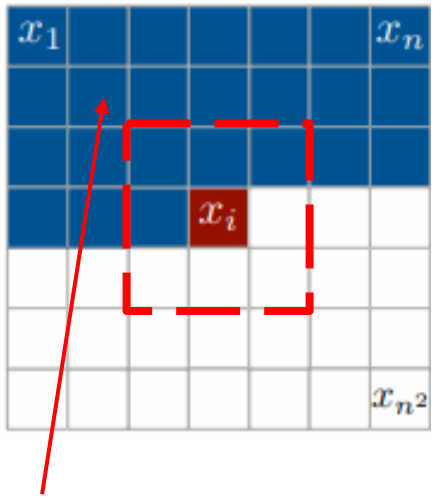
these are **masked out** because they haven't been generated yet

Question: can we parallelize this?

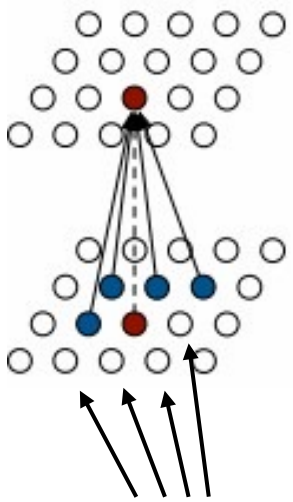
Van den Oord et al. **Pixel Recurrent Neural Networks**. 2016.

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?



these are **masked out** because they haven't been generated yet

Question: can we parallelize this?

During **training**? Yes

During **generation**? No

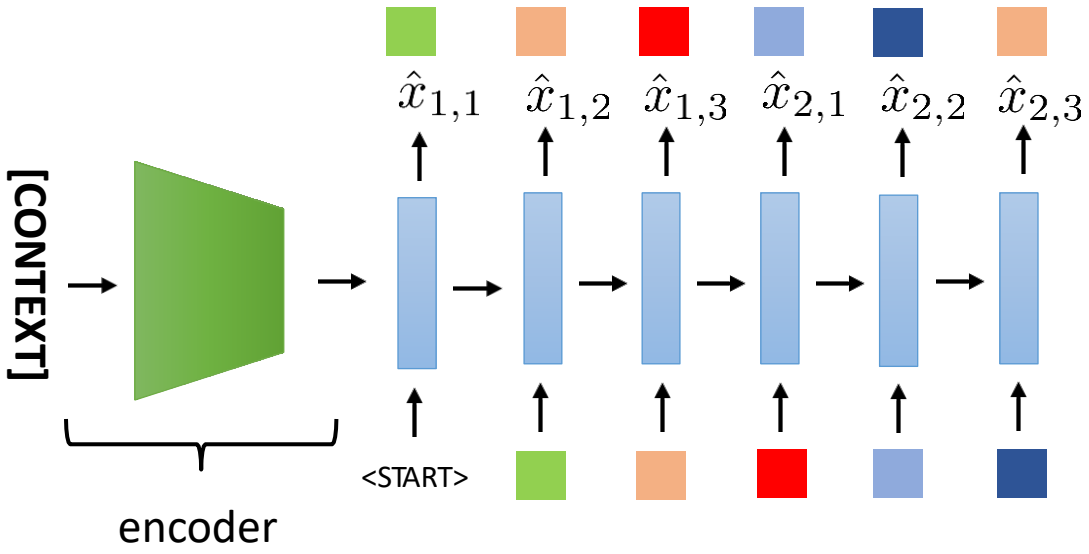
Van den Oord et al. **Pixel Recurrent Neural Networks**. 2016.

Conditional autoregressive models

What if we want to generate something **conditioned** on another piece of information?

Examples:

- Generate images of specific types of objects (e.g., categories)
- Generate distributions over actions for imitation learning conditioned on the observation
- Many other examples!



Just like conditional language models!

Encoder can be **extremely simple** (e.g., generate images of a class)

Encoder can be **extremely complex** (e.g., multimodal policy in IL)

Conditional autoregressive models

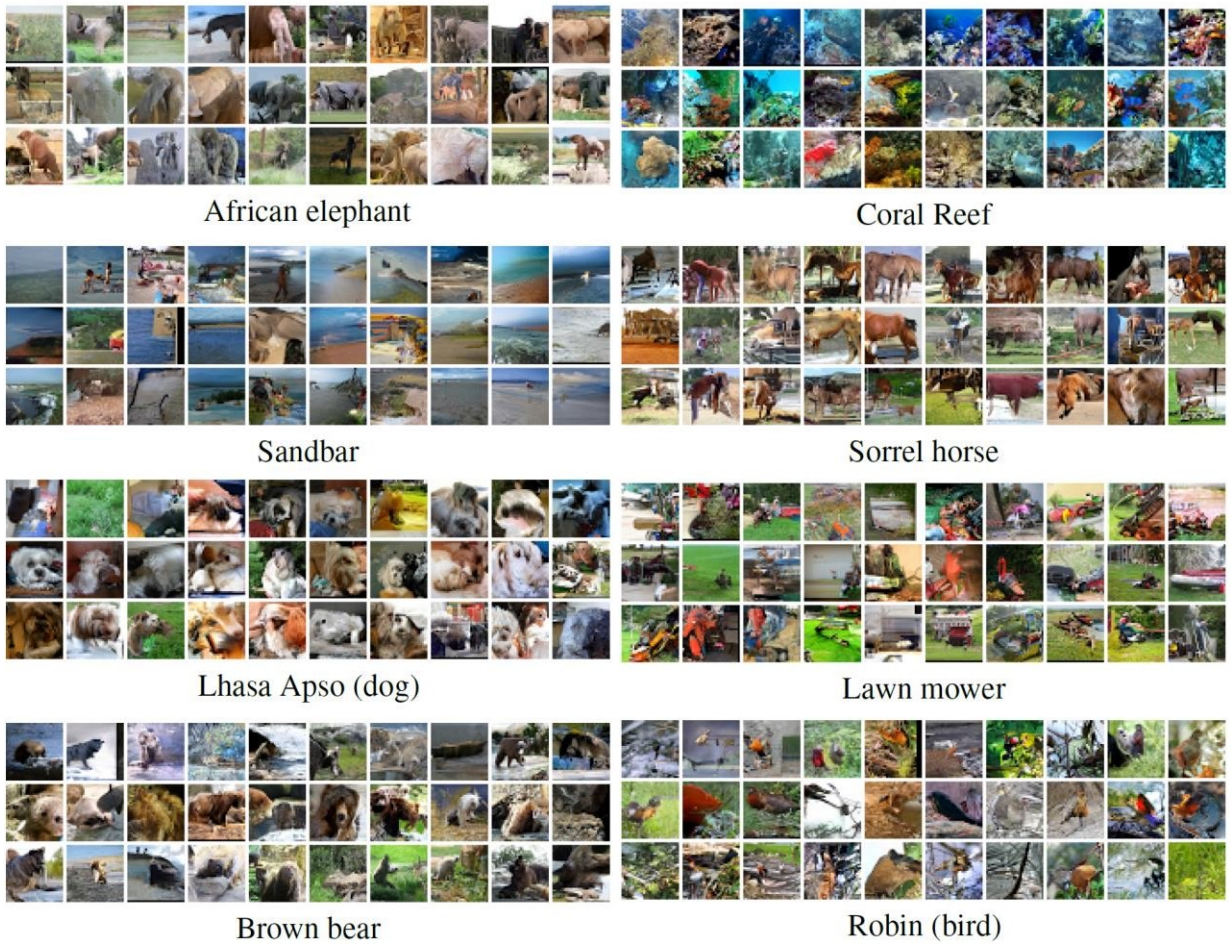


Figure 3: Class-Conditional samples from the Conditional PixelCNN.

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

Tradeoffs and considerations

- Autoregressive generative models are “language models” for other types of data
  - Though more accurate to say that language models are just a special type of autoregressive generative model

- Can represent autoregressive models in many different ways
  - RNNs (e.g., LSTMs)
  - Local context models like PixelCNNs
  - Transformers

- Tradeoffs compared to other models we’ll learn about:

  + provide full distribution with probabilities

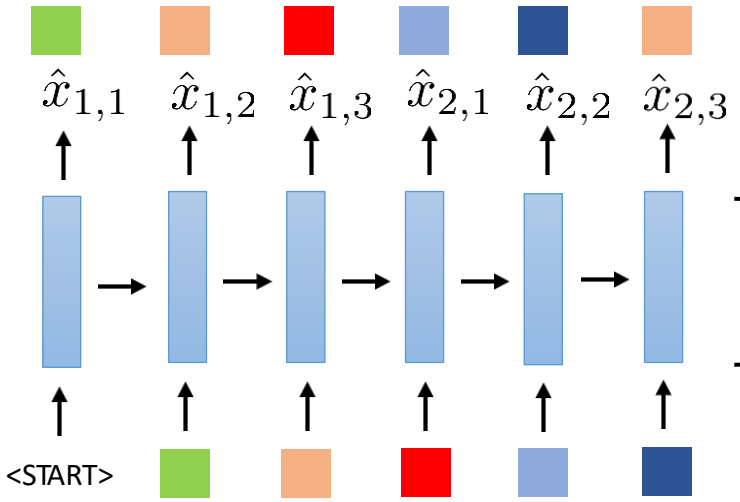
  + conceptually very simple

  - very slow for large datapoints (e.g., images)

  - generally limited in image resolution

Autoencoders

A 30,000 ft view…

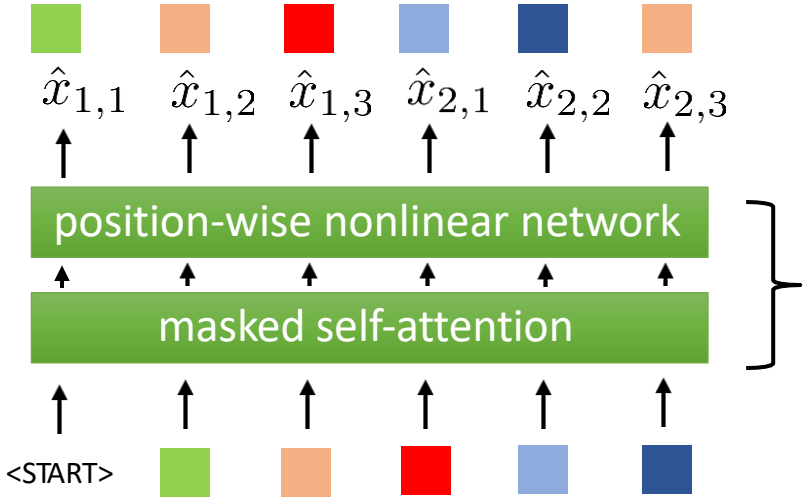


goal is to produce the **same** image

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

In this case, it's the fact that each pixel must be constructed from **preceding** pixels

image goes in as input



structure comes from **masking** in the self-attention operator (otherwise this would be trivial!)

remember BERT…

A 30,000 ft view…

A general design 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 sequence models that must predict a pixel's value based only on "previous" pixels
- "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

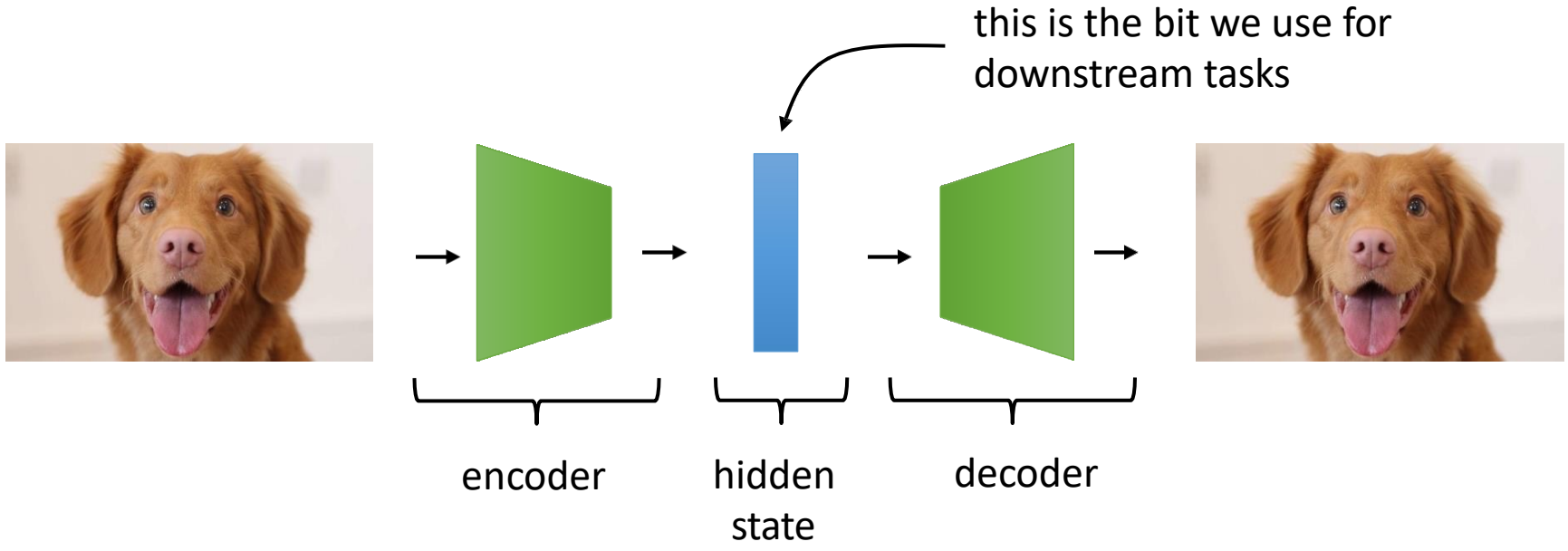
This is all **spatial** structure, can we use more abstract structure instead?

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 in the data processing or regularization, must force the autoencoder to learn a **structured** representation



The types of autoencoders

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

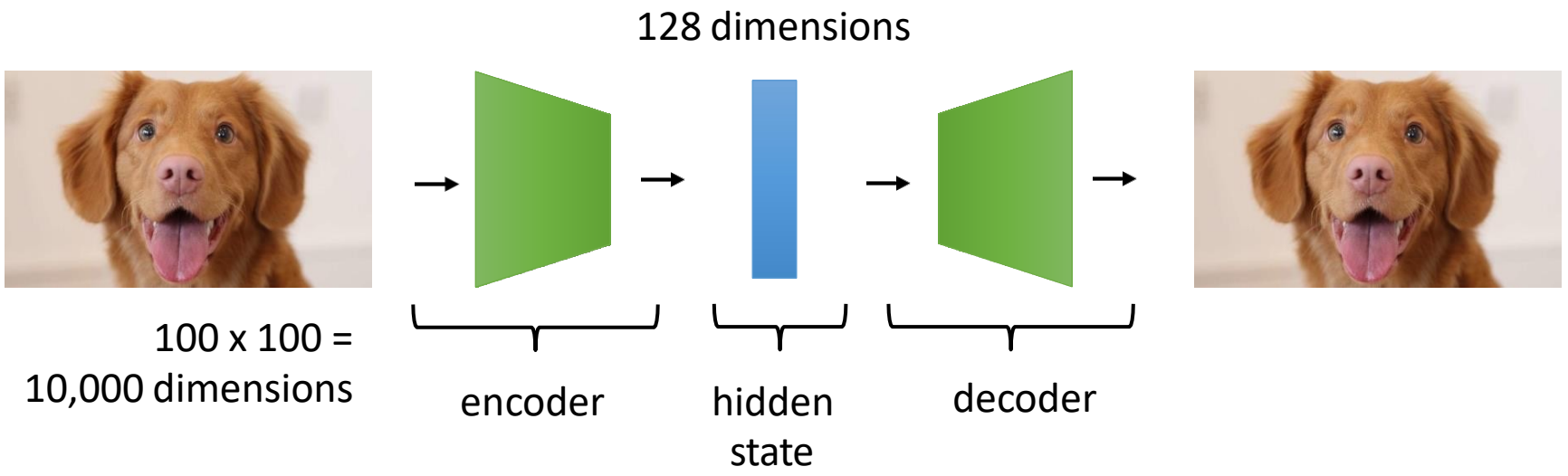
Dimensionality: make the **hidden state** smaller than the **input/output**, so that the network must **compress** it

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** (this will be covered next time)

(Classic) Bottleneck autoencoder



This has some interesting properties:

- If both encoder and decoder are **linear** (which is usually not very interesting), this exactly recovers PCA
- Can be viewed as "non-linear dimensionality reduction" – could be useful simply because dimensionality is lower and we can use various algorithms that are only tractable in low-dimensional spaces (e.g., discretization)

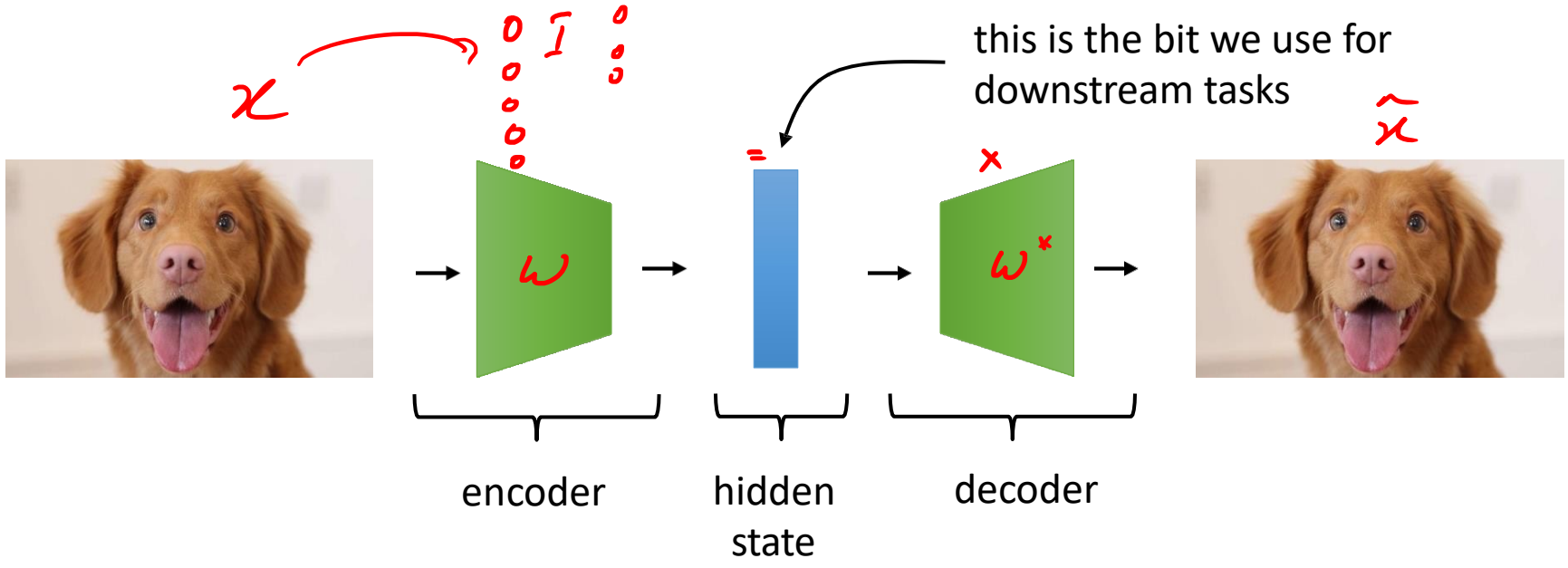
Today, this design is rather antiquated and rarely used, but good to know about historically

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**

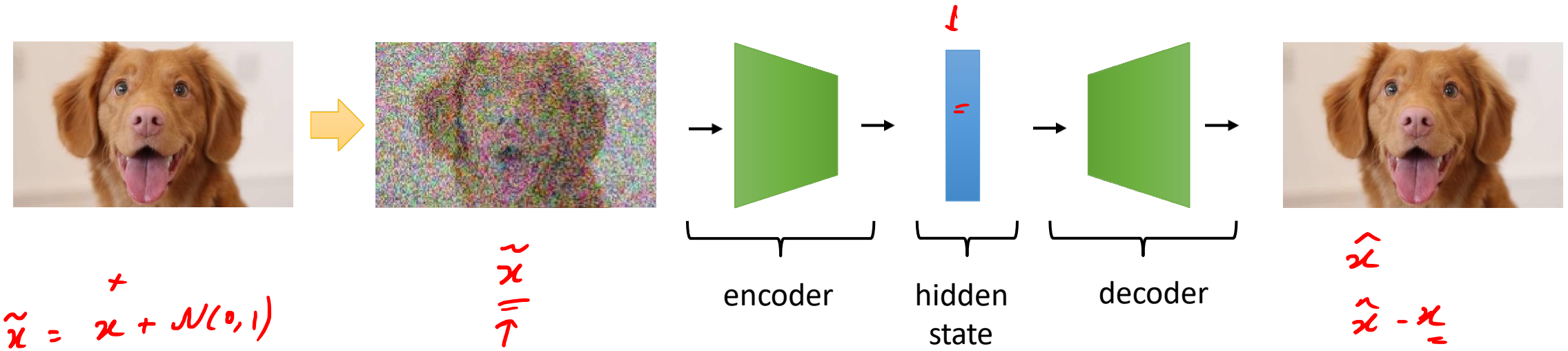
Such a network is called an **autoencoder**

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



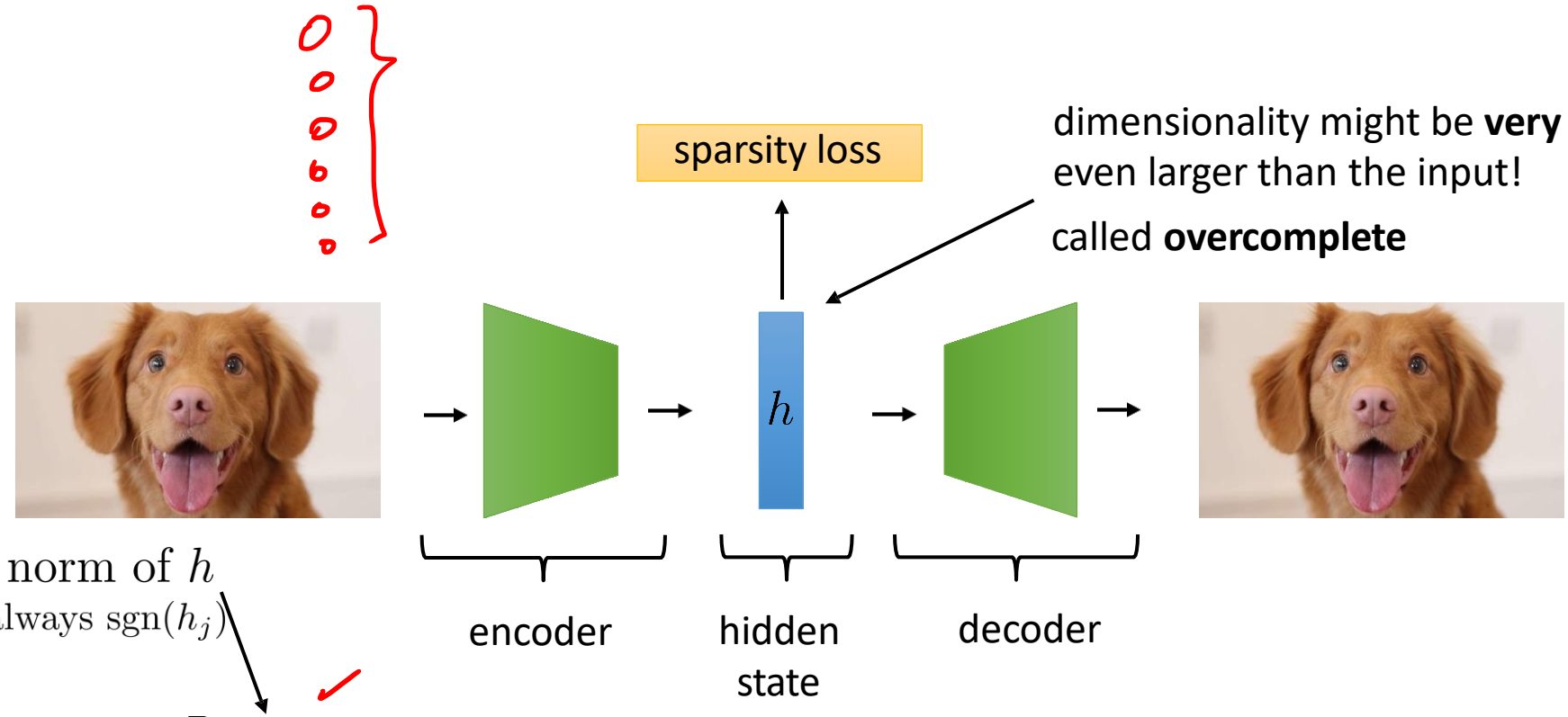
Denoising autoencoder

Idea: a good model that has learned meaningful structure should "fill in the blanks"



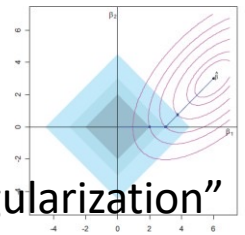
There are **many variants** on this basic idea, and this is one of the most widely used simple autoencoder designs

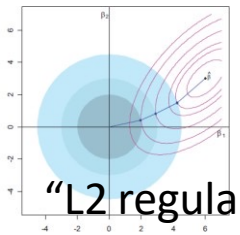
Sparse autoencoder



ℓ_1 norm of h
gradient is always $\text{sgn}(h_j)$

simple sparsity loss: $\sum_{j=1}^{D} |h_j|$


"L1 regularization"


"L2 regularization"

There are other kinds of sparsity losses/models:

- Lifetime sparsity
- Spike and slab models

Sparse autoencoder

Idea: can we describe the input with a small set of "attributes"?

This might be a more **compressed** and **structured** representation

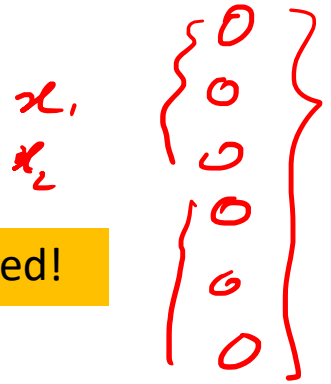
Aside:

This idea originated in neuroscience, where researchers believe that the brain uses **sparse** representations (see "sparse coding")

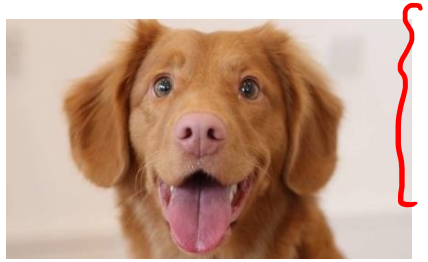
Pixel (0,0): #FE057D
Pixel (0,1): #FD0263
Pixel (0,2): #E1065F

NOT structured

"dense": most values non-zero



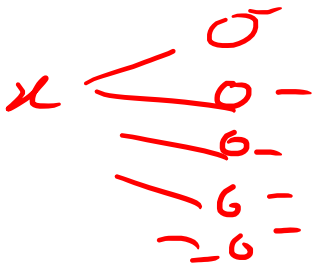
Idea: "sparse" representations are going to be more structured!



has_ears: 1
has_wings: 0
has_wheels: 0

very structured!

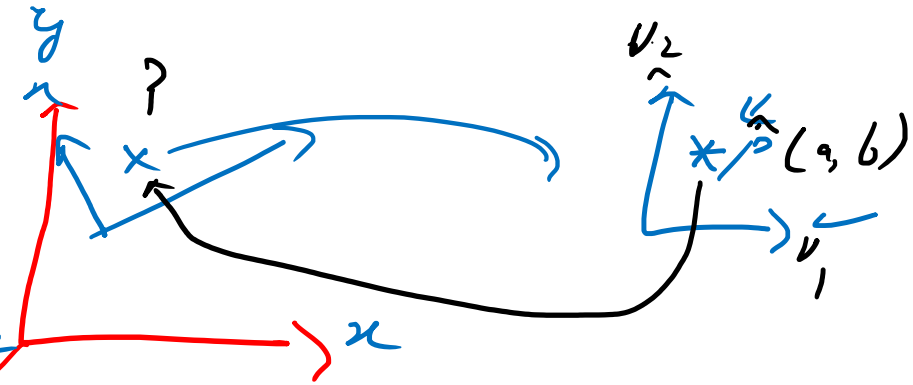
"sparse": most values are zero



there are many possible "attributes," and most images don't have most of the attributes

$V^T V = I$ $\quad V$

orthonormal



P $\quad$ PCA $\quad$ [ ] $\quad$ [ ] $\rightarrow$

$\hat{P} = V^T P$

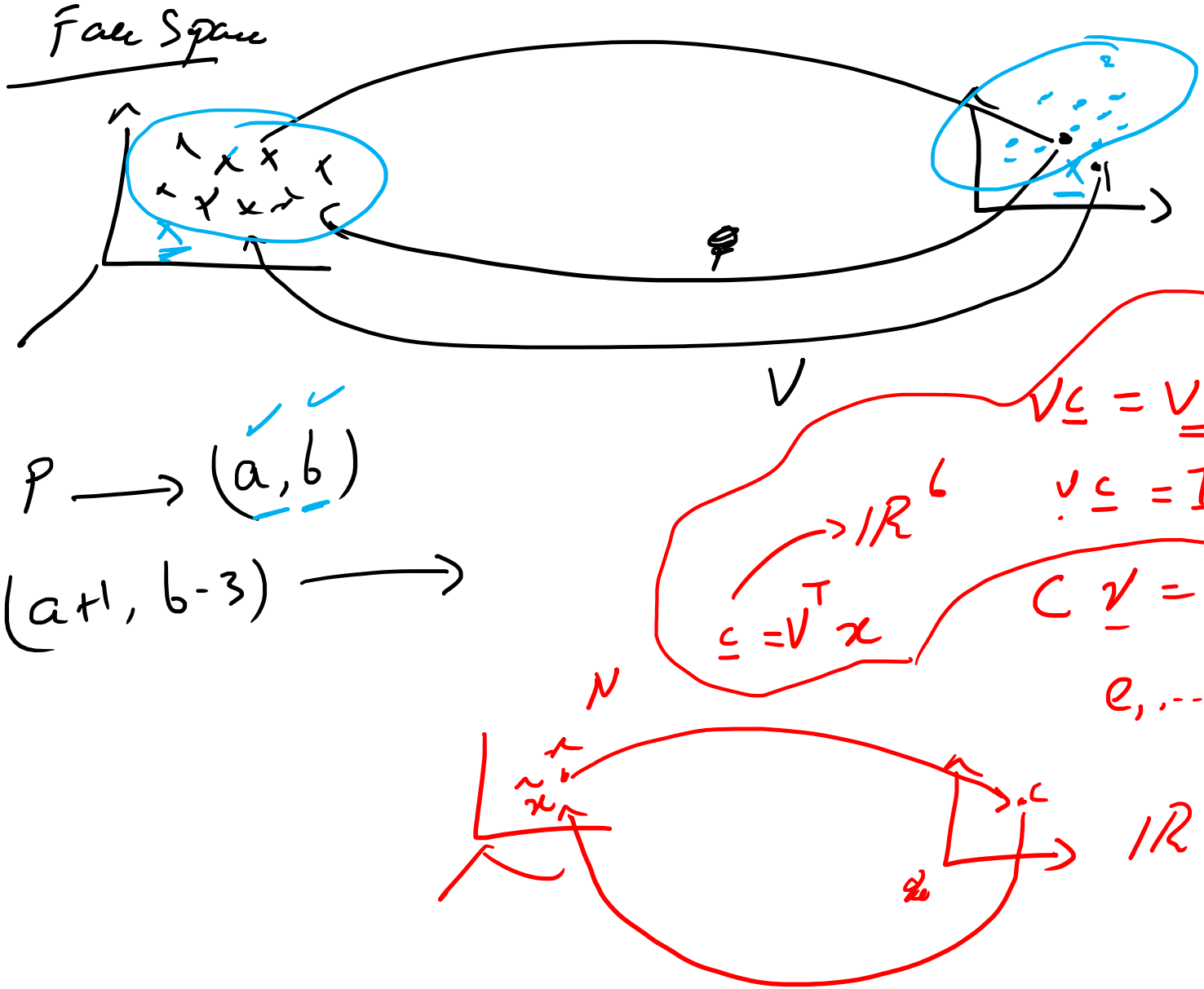
$V\tilde{P} = \tilde{P}$

$$\begin{bmatrix} 2 \\ 8 \\ 6 \end{bmatrix} = 2\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + 8\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + 6\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

x $\quad$ y $\quad$ z

$\hat{p} = (a, b)$

$\tilde{p} = a v_1 + b v_2$


Face Space
V
P → (a, b)
(a+1, b-3) →
$\mathbb{R}^6$
$c = V^T x$
$V c = V V^T x$
$x \in \mathbb{R}^N$
$V c = I x \rightarrow V c = \tilde{x}$
$C v = e_i v$
$e_1 \dots e_6$ $[v_1 \dots v_6]$
$\in \mathbb{R}^N$
N
$\tilde{x}$
c
$\mathbb{R}^6$

The types of autoencoders

Forcing structure: something about the design of the model, or in 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

+ very simple to implement

- simply reducing dimensionality often does not provide the structure we want

Sparsity: force the **hidden state** to be sparse (most entries are zero), so that the network must **compress** the input

+ principled approach that can provide a "disentangled" representation

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

Denoising: corrupt the **input** with **noise**, forcing the autoencoder to learn to distinguish **noise from signal**

+ very simple to implement

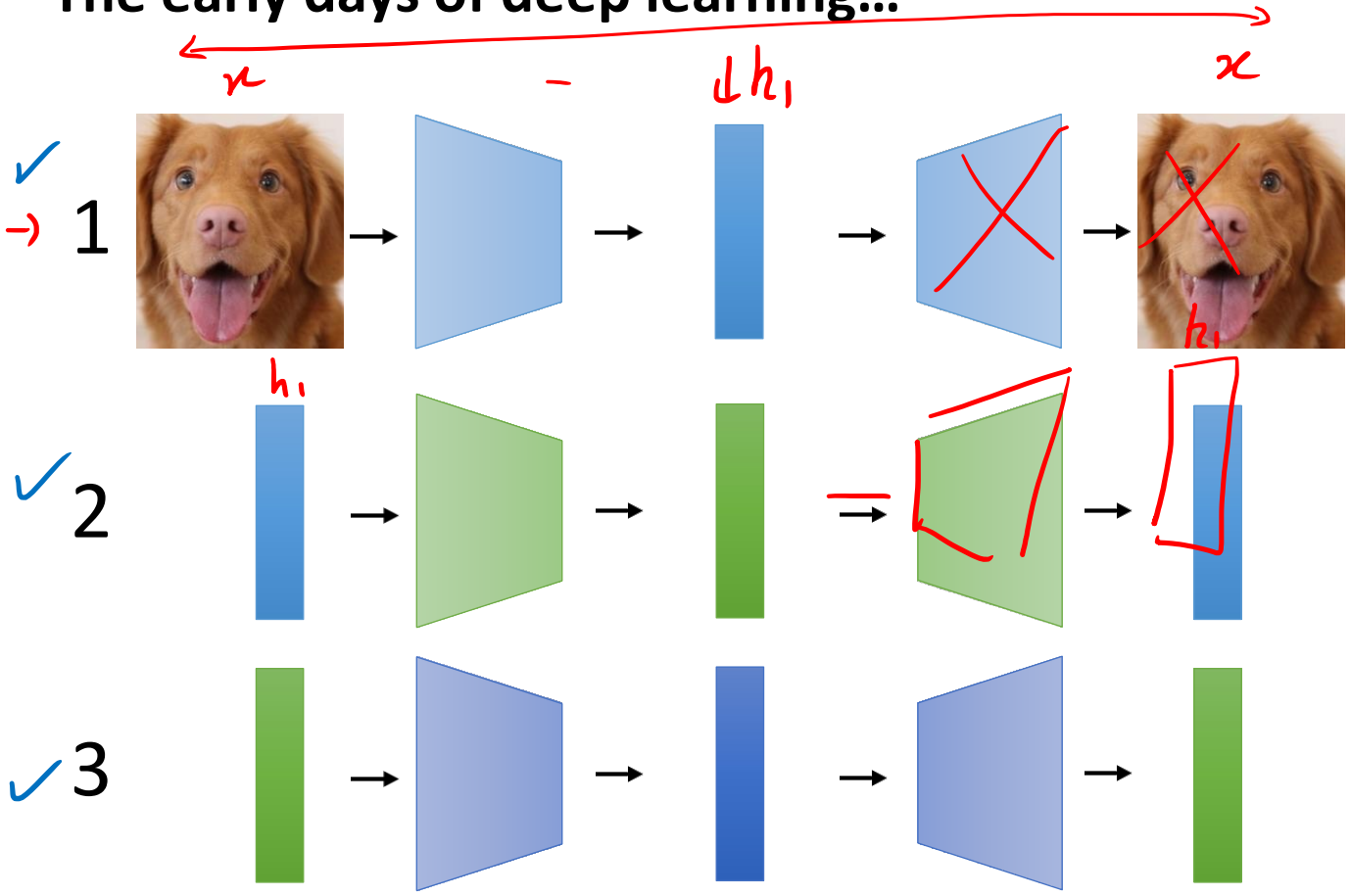
- not clear which layer to choose for the bottleneck, many ad-hoc choices (e.g., how much noise to add)

Probabilistic modeling: force the **hidden state** to agree with a **prior distribution** (this will be covered next time)

We'll discuss this design in much more detail in the next lecture!

Layerwise pretraining

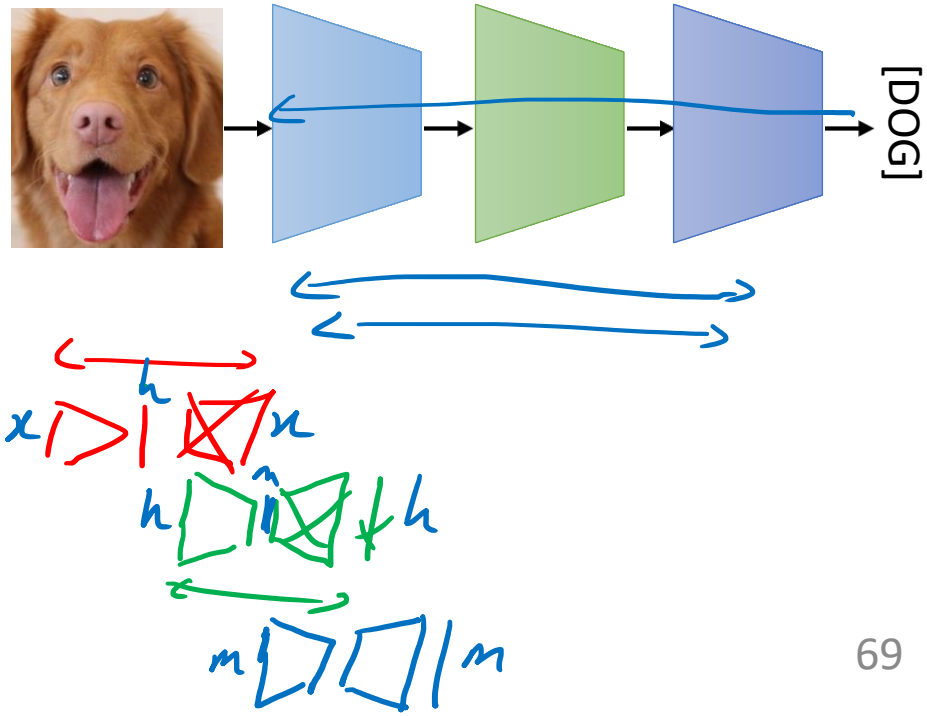
The early days of deep learning…



For a while (2006-2009 or so), this was one of the dominant ways to train **deep** networks

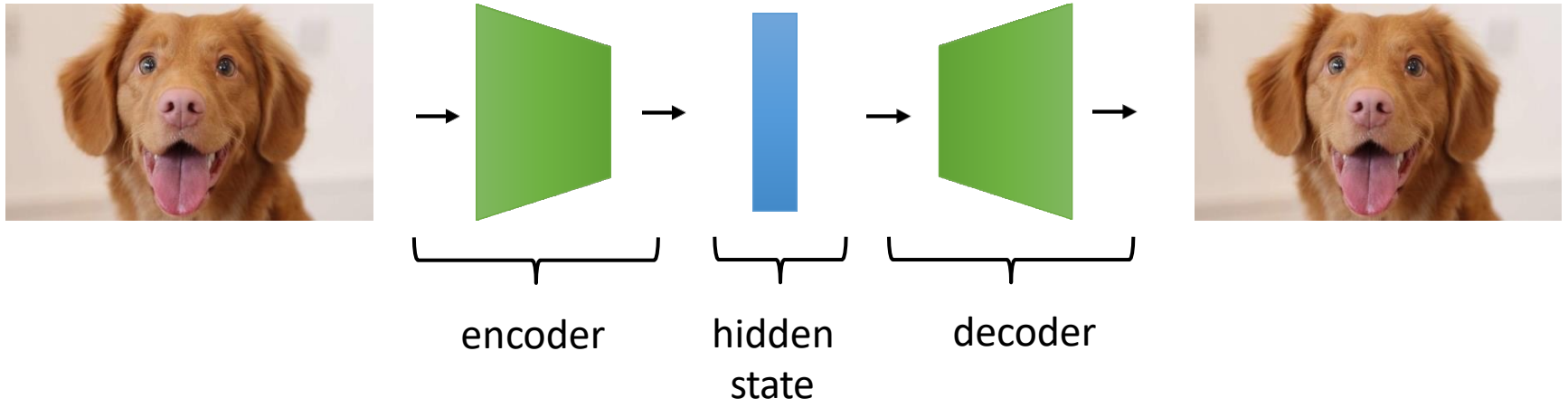
Then we got a lot better at training deep networks end-to-end (ReLU, batch norm, better hyperparameter tuning), and largely stopped doing this

Correspondingly, autoencoders became less important, but they are still useful!

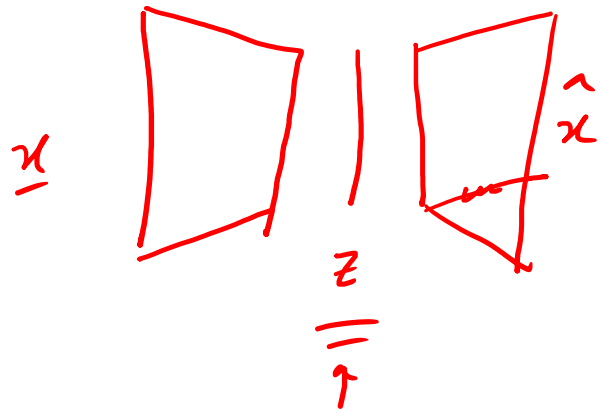


Autoencoders today

- Much **less** widely used these days because there are better alternatives
  - **Representation learning:** VAEs, contrastive learning
  - **Generation:** GANs, VAEs, autoregressive models
- Still a viable option for "quick and dirty" representation learning that is very fast and can work OK
- **Big problem:** sampling (generation) from an autoencoder is hard, which limits its uses
  - The variational autoencoder (VAE) addresses this, and is the most widely used autoencoder today – we will cover this next time!



variational inference



$I = -\log p(x)$ ↓ x is an event

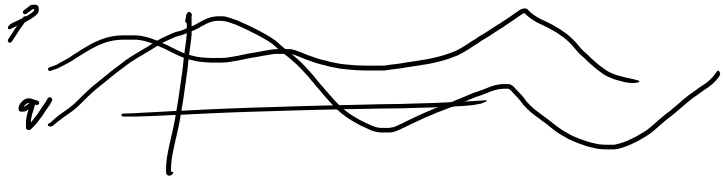
$p(x_1) = 0.9$ I_1 $I_1 < I_2$

$p(x_2) = 0.1$ I_2

I is large when prob is low ?

$H = \sum_x p(x) \{-\log p(x)\} = -\sum p(x) \log p(x)$

$p \mid q$



w.r.t p

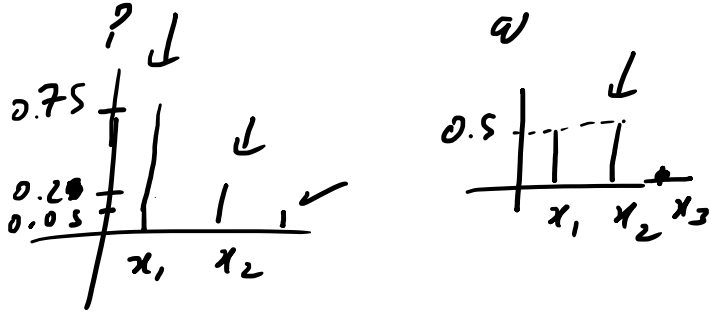
$KL(p||q) \simeq -\sum q(x) \log q(x) - (-\sum p(x) \log p(x))$

Div

$= -\sum p(x) \log q(x) + \sum p(x) \log p(x) = -\sum p(x) \log \frac{q(x)}{p(x)}$

Average w.r.t p(x)

$\not\geq KL(p||q) \neq KL(q||p)$



I, H, KL

$-(0.75 \log 0.5 + 0.25 \log 0.5) + 0.75 \log(0.75) + 0.25 \log(0.25)$

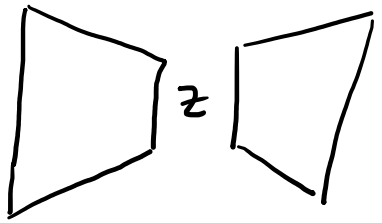
$$\Rightarrow p(z/x) = \frac{p(x/z)\,p(z)}{p(x)} = \frac{p(x,z)}{p(x)}$$

$$p(x) = \int_z p(x/z)\,p(z)\,dz \Big\} \rightarrow \text{Variational}$$

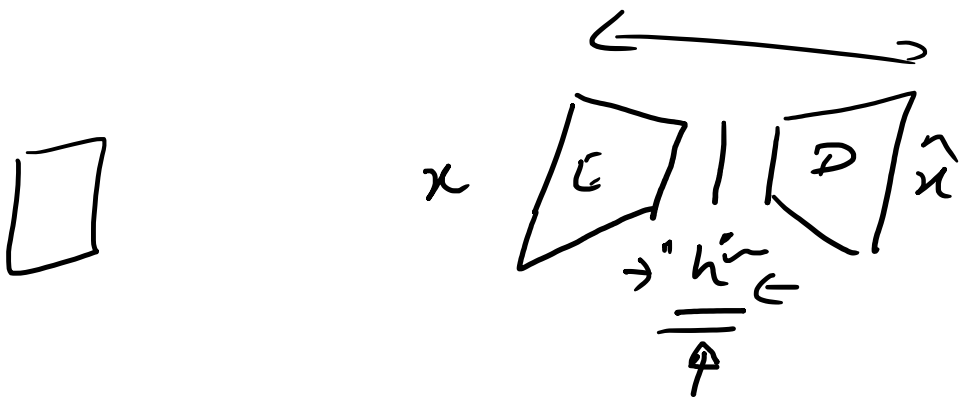
$z \in \mathbb{R}^{\text{very high}}$

$p(z/x)$ Approx by $q(z)$ → tractible distribution

$= q(z/x)$



$p(z/x)$ $q(z)$



$$KL(q\|p) = -\sum q(z) \log \frac{p(z|x)}{q(z)} = -\sum q(z) \log \frac{p(x,z)/p(x)}{q(z)}$$

$p(z|x)$

$$= -\sum_z q(z)\left\{\log \frac{p(x,z)}{q(z)} + \log \frac{1}{p(x)}\right\} = -\sum q(z) \log \frac{p(x,z)}{q(z)} + \log p(x) \underbrace{\sum_z q(z)}_{1}$$

$$KL(q\|p(z|x)) = -\sum q(z) \log \frac{p(x,z)}{q(z)} + \log p(x)$$

≥ 0 ; const ; c

min ; $\mathcal{L}$

$\max \mathcal{L}$ variational lower bound

$$KL(q(z)\|p(z|x)) + \mathcal{L} = \log p(x)$$

$KL(q(z)\|p(z|x)) \geq 0$

constant if x is given

$\mathcal{L} \leq c$

e.g. $m = -n + g$, $m \geq 0$

$m = -n + 10$, $m \geq 0$

$$\log p(x) = KL(q(z) \| p(z|x)) + \sum q(z) \log \frac{p(x,z)}{q(z)}$$

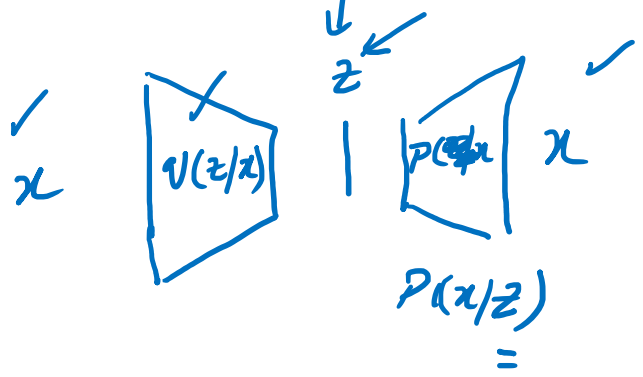
$\log p(x)$ = c ✓

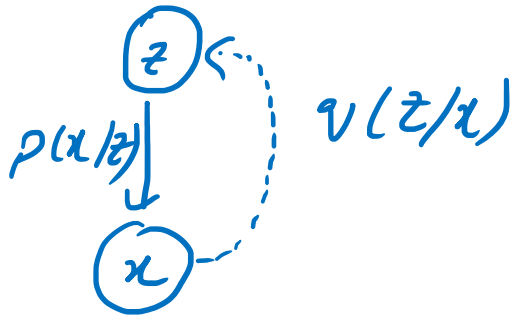
$\sum q(z) \log \frac{p(x,z)}{q(z)}$ ↑ $\mathcal{L}$ ← variational Lower bound.

$$\mathcal{L} = \sum q(z) \log \frac{p(x,z)}{q(z)}$$

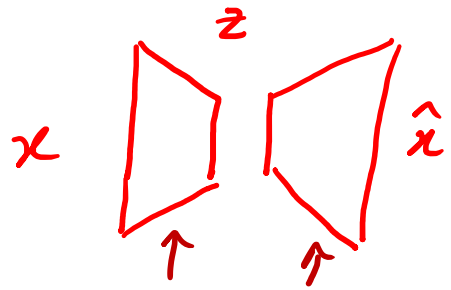
$$= \sum q(z) \log \frac{p(x|z)\,p(z)}{q(z)} = \sum q(z) \log p(x|z) - \sum q(z) \log \frac{p(z)}{q(z)}$$

$$= E_q\{\log p(x|z)\} - KL(q(z) \| p(z))$$





VAE



$$\rightarrow KL(q_{\theta'}(z/x) \,||\, p_{\theta}(z/x)) = -\sum q(z/x) \log \frac{p(x,z)}{q(z)} + \log p(x)$$

+ve

$$\rightarrow \log p(x) = KL(q(z/x)\,||\,p(z/x)) + \sum q(z/x) \log \frac{p(x,z)}{q(z/x)}$$

$\log p(x)$: c

$KL(q(z/x)\,||\,p(z/x)) \geq 0$

$\mathcal{L}$: variational Lower bound.

$\mathcal{L} = c$

$\mathcal{L} \leq c$

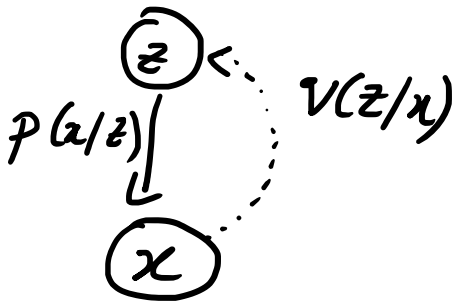
$$\mathcal{L} = \sum q(z|x) \log \frac{p(x,z)}{q(z|x)}$$

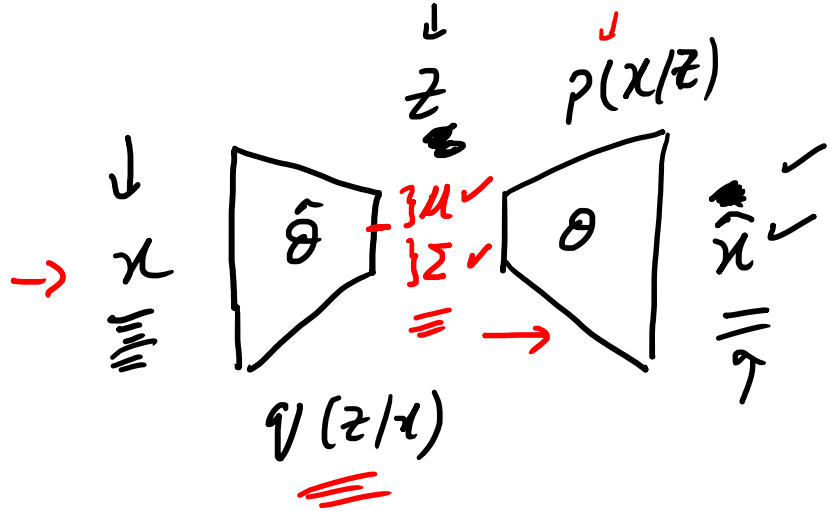
$$= \sum q(z|x) \log \frac{p(x|z)\,p(z)}{q(z|x)} = \sum_z q(z|x) \log p(x|z) + \sum q(z|x) \log \frac{p(z)}{q(z|x)}$$

$$= \mathbb{E}_{q(z|x)} \log p(x|z)$$

$$= \mathbb{E}_{q(z|x)} \{ \log p(x|z) \} - KL(q(z|x) \,||\, p(z))$$

$-KL(q(z|x) \,||\, p(z))$





$$\mathcal{L} = E_{q_\theta(z/x)} \log p(x/z) - KL(q_\theta(z/x) \| p(z))$$

$$= -|x - \hat{x}|^2 - KL(q_\theta(z/x) \| p(z))$$

$$p(x/\hat{x}) = \text{const} \exp(-|x-\hat{x}|^2)$$

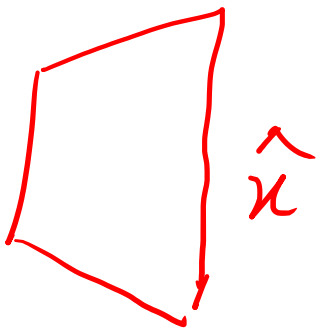
$$\log p(x/\hat{x}) = \text{const} - |x-\hat{x}|^2$$

$$\min |x - \hat{x}| + KL(q(z/x) \| N(0, I))$$

$$\Sigma = \begin{bmatrix} \cdot & 0 \\ 0 & \cdot \end{bmatrix}$$

$$N(\mu_{z/x}, \Sigma_{z/x})$$

x 0 0 0 0 } μ
0 0 0 0
0 0 0 0 } σ_1
0 0 0 0 } σ_2
0 0 0
0

$N(\mu, \sigma_1, \sigma_2) \sim z \rightarrow$ 

$\hookrightarrow$

$x = \mu_1 + d$

$\hookleftarrow$

$\lambda = N(0,1)$

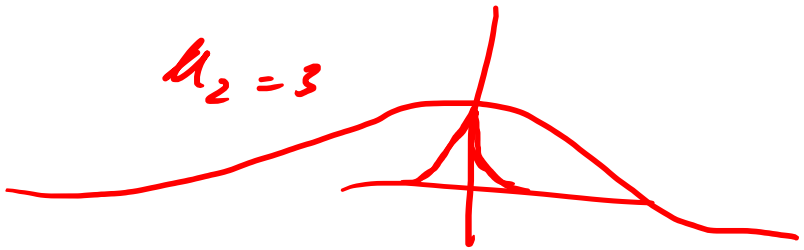
$\hookrightarrow x = \mu_1 + \lambda\sigma_1$

$\overline{\overline{T}}$

$-\sigma_1, \sigma_1$

$\mu_1 = 2$

$\mu_2 = 3$



Variational Autoencoders (VAE)

So far...

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

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

So far...

PixelCNNs define tractable density function, optimize likelihood of training data:

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

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

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

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

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

So far...

PixelCNNs define tractable density function, optimize likelihood of training data:

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

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

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

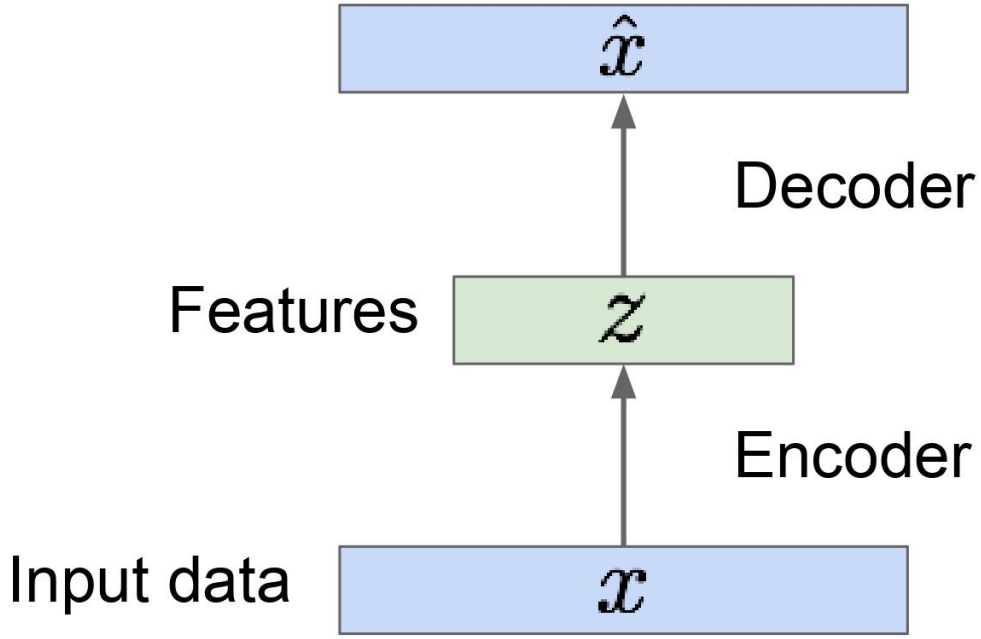
No dependencies among pixels, can generate all pixels at the same time!

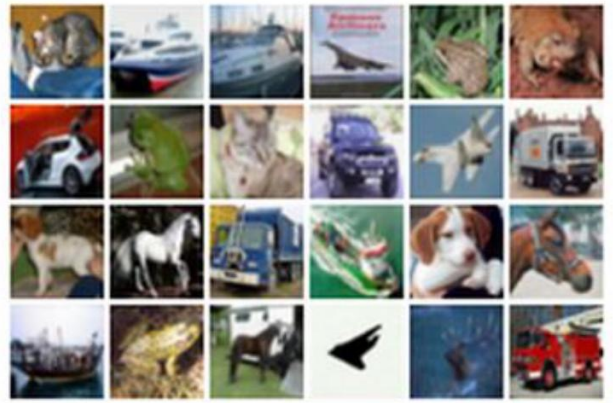
Cannot optimize directly, derive and optimize lower bound on likelihood instead

Why latent z?

Some background first: Autoencoders

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



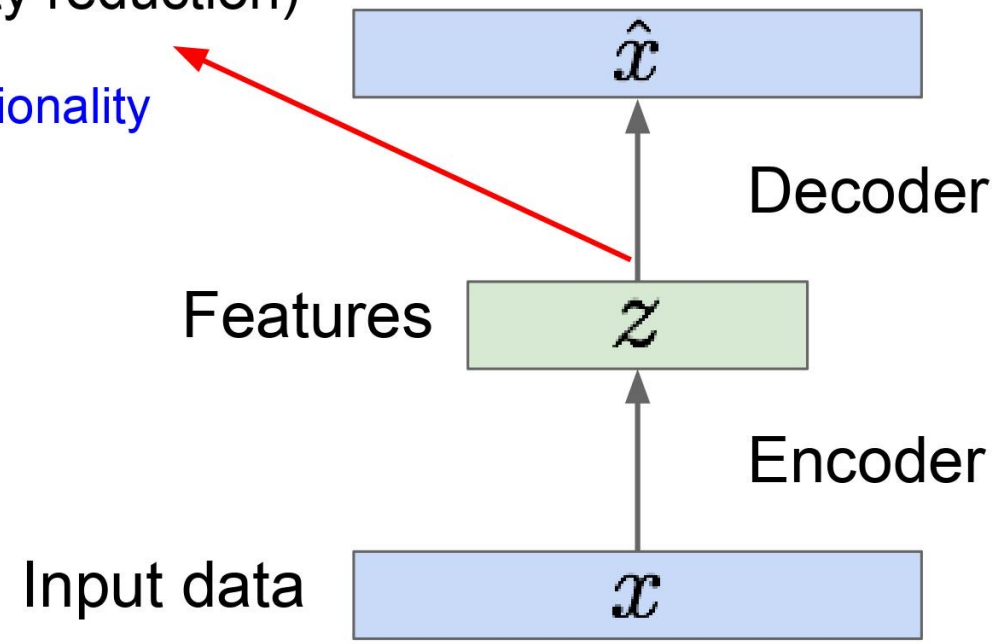


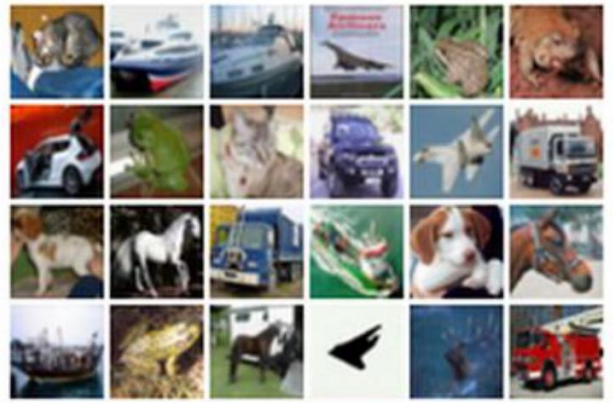
Some background first: Autoencoders

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

z usually smaller than **x**
(dimensionality reduction)

Q: Why dimensionality reduction?





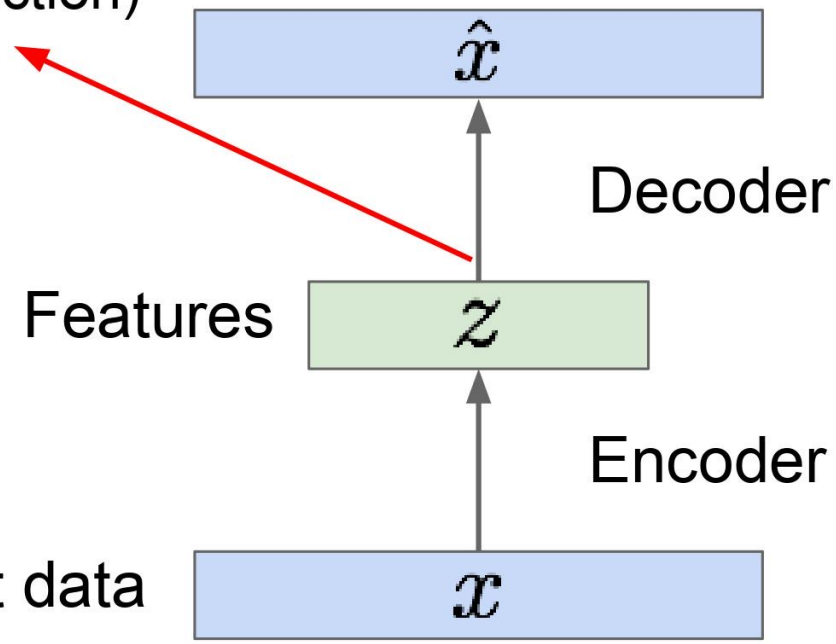
Some background first: Autoencoders

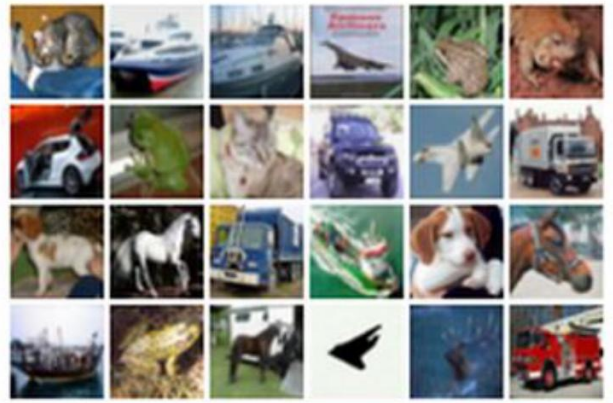
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x**
(dimensionality reduction)

Q: Why dimensionality reduction?

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

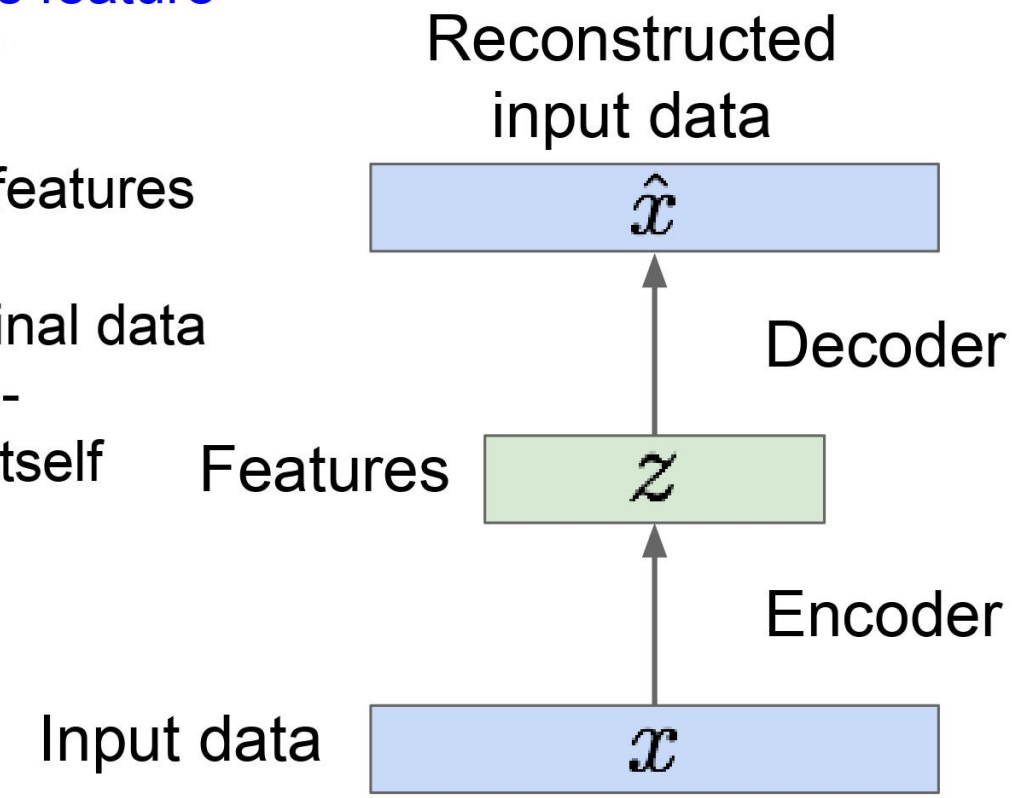


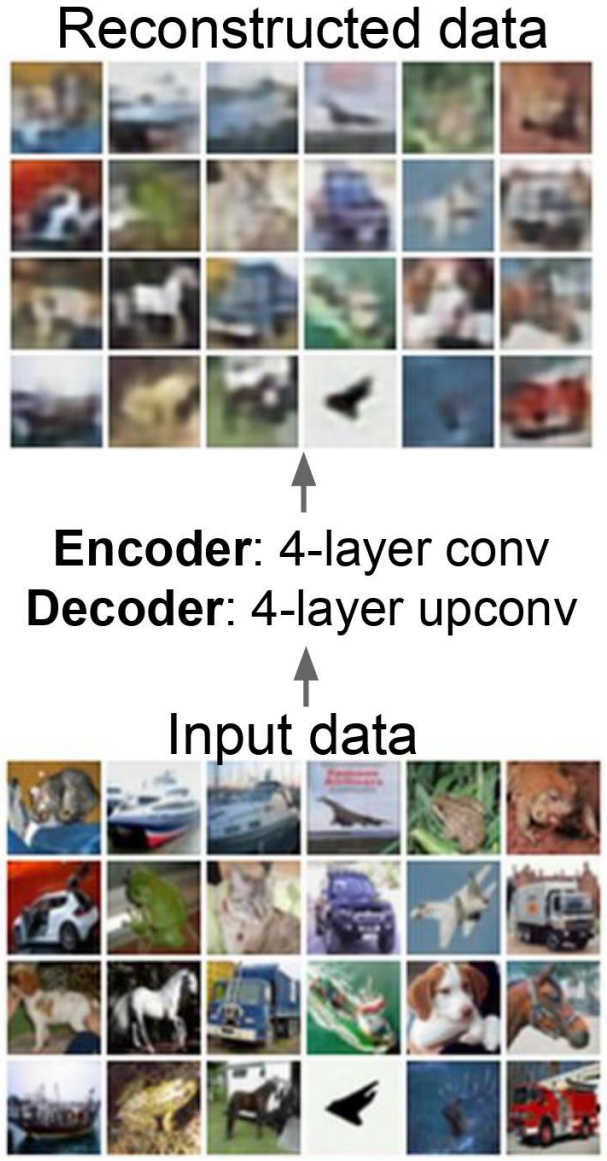


Some background first: Autoencoders

How to learn this feature representation?

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

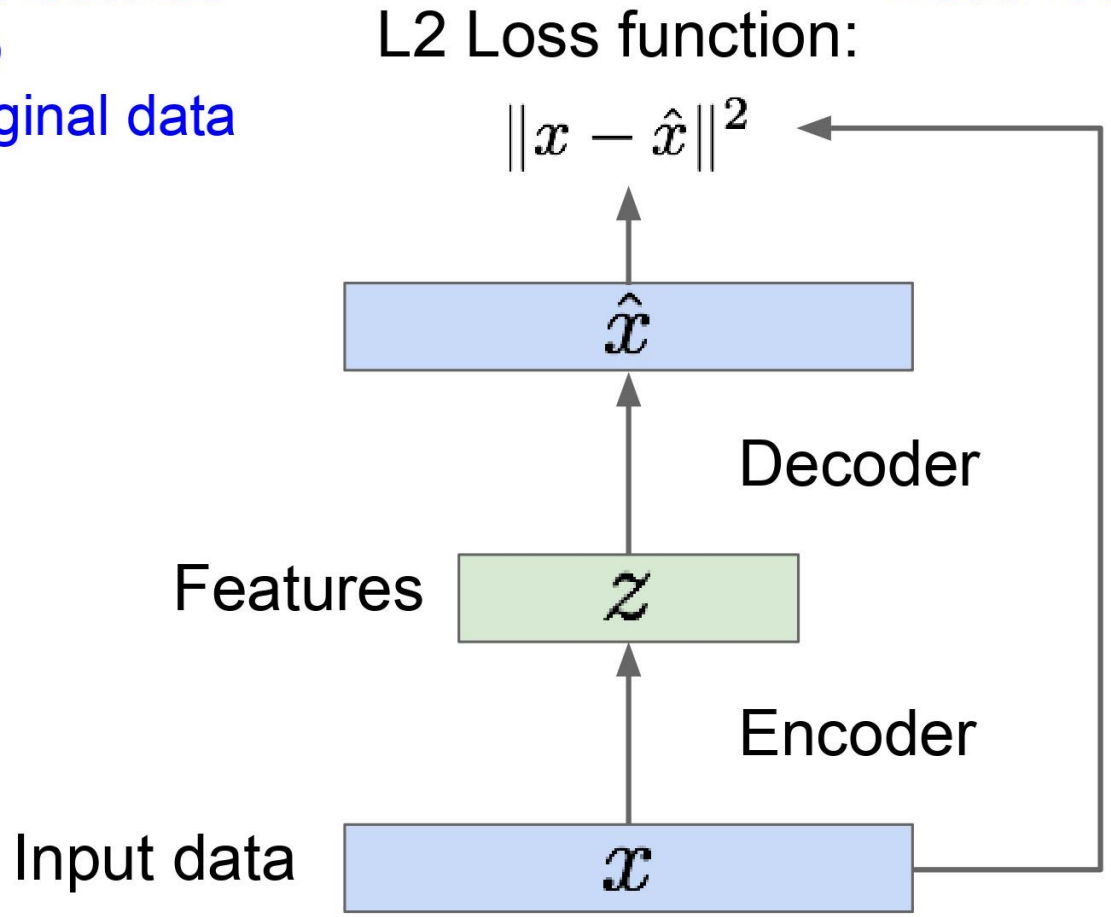


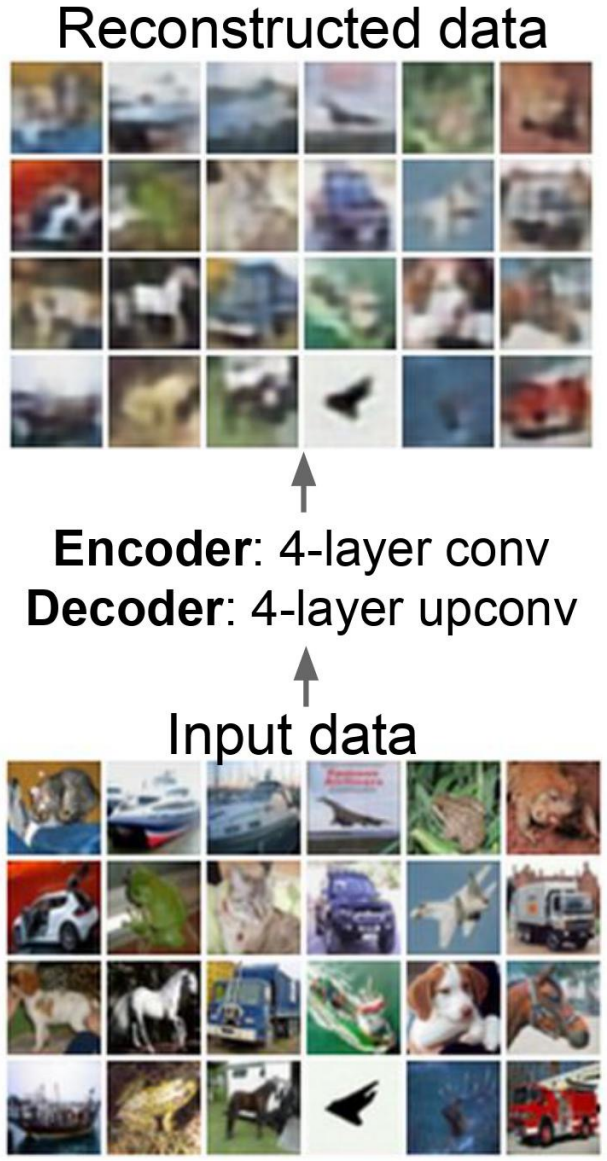


Some background first: Autoencoders

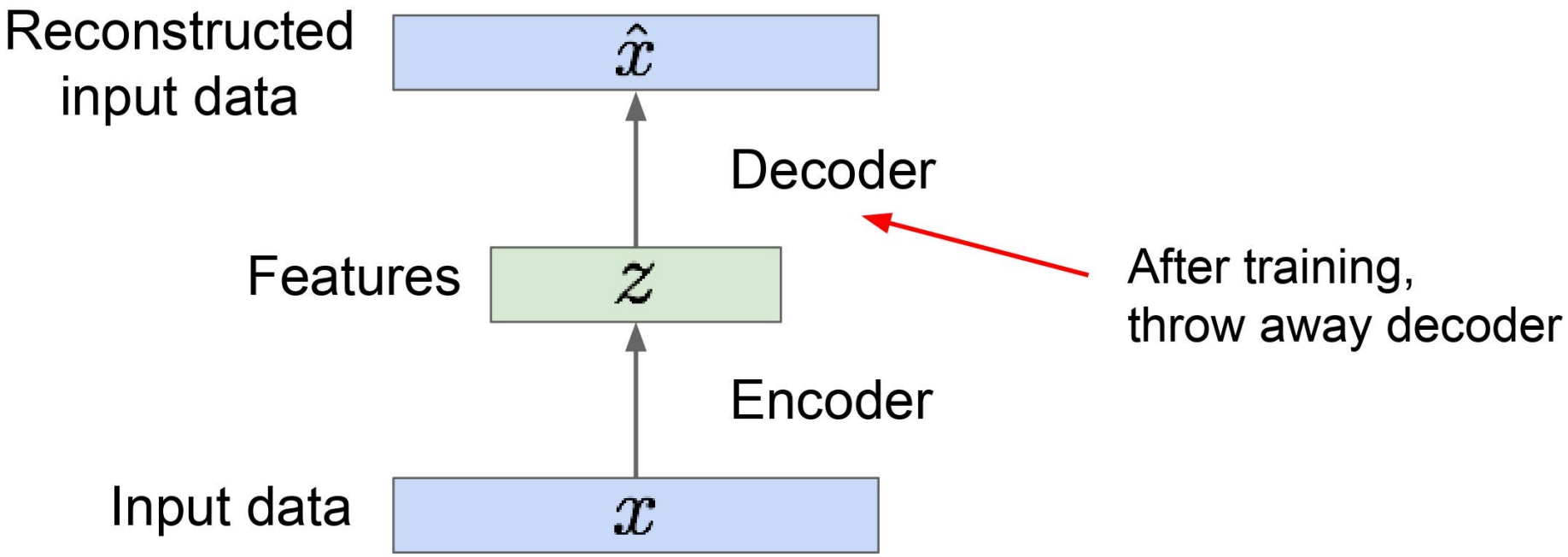
Train such that features can be used to reconstruct original data

Doesn't use labels!



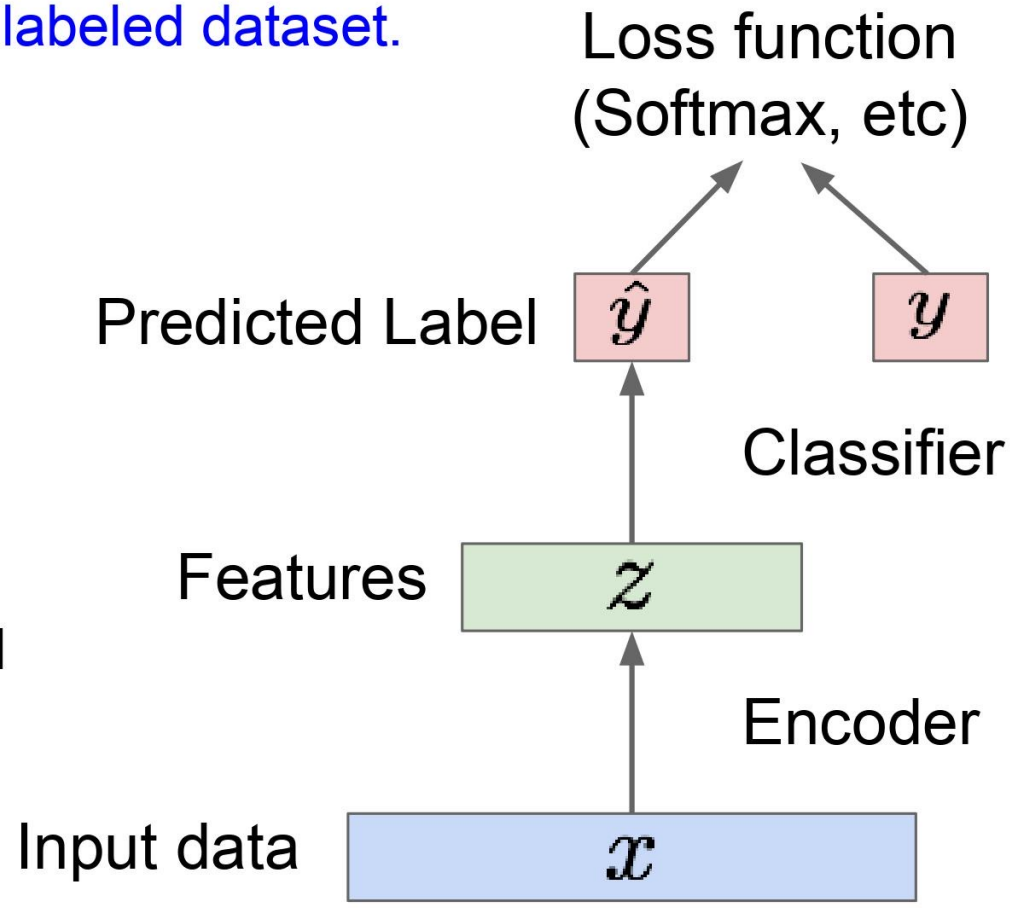


Some background first: Autoencoders



Some background first: Autoencoders

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



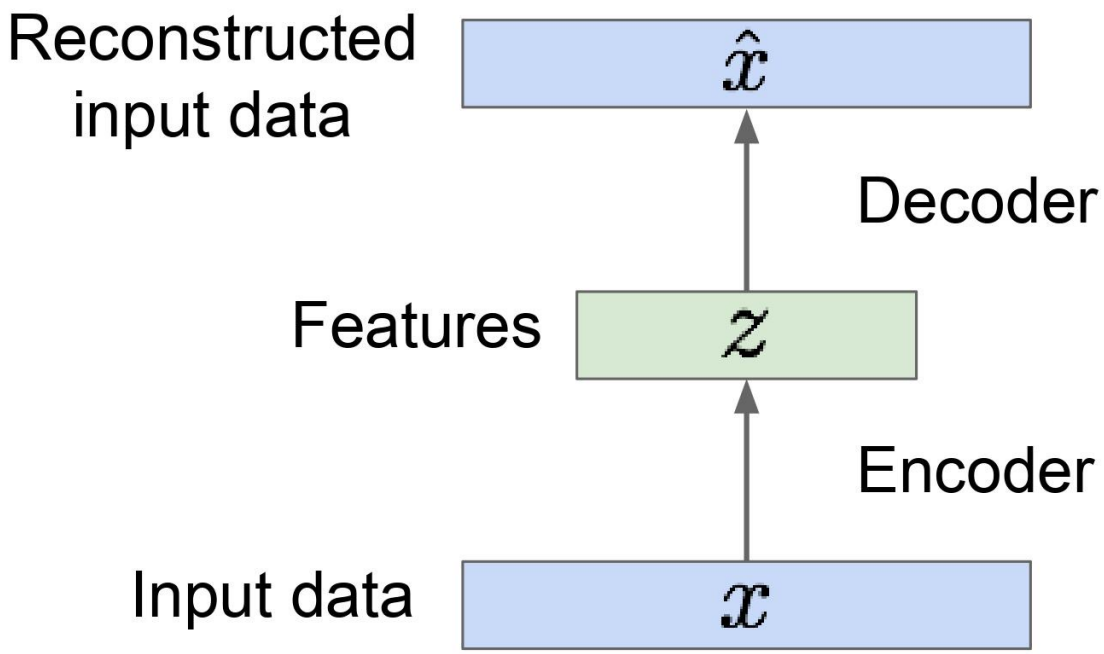
Encoder can be used to initialize a **supervised** model

Fine-tune encoder jointly with classifier

Train for final task (sometimes with small data)

bird plane
dog deer truck

Some background first: Autoencoders



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

Features capture factors of variation in training data.

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

How do we make autoencoder a **generative model**?

Variational Autoencoders

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

Variational Autoencoders

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

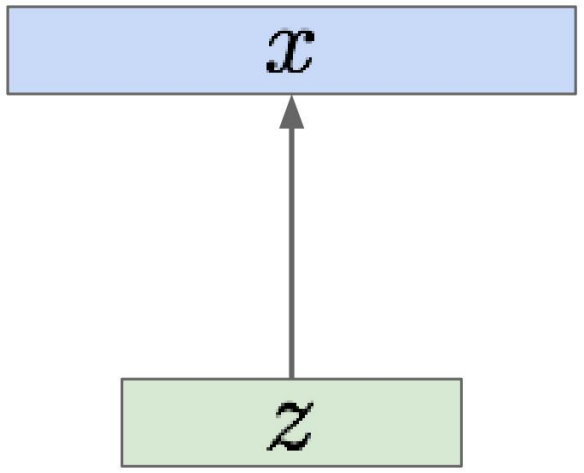
Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from the distribution of unobserved (latent) representation **z**

Sample from true conditional

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

Sample from true prior

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



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

Variational Autoencoders

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

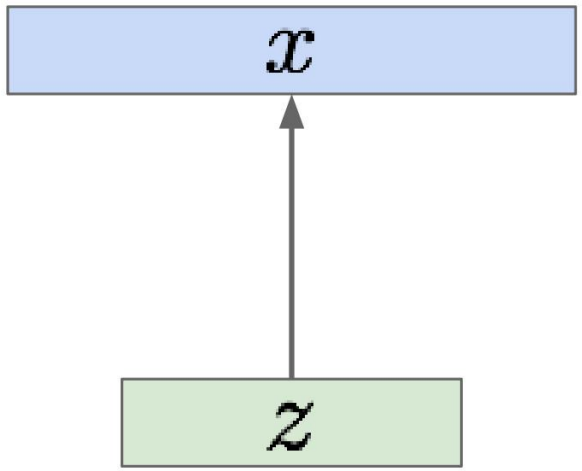
Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from the distribution of unobserved (latent) representation **z**

Sample from true conditional

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

Sample from true prior

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



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

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

Variational Autoencoders

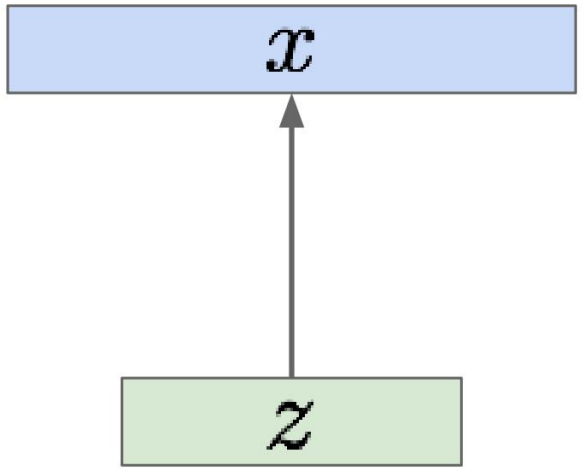
We want to estimate the true parameters θ^* of this generative model given training data x.

Sample from true conditional

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

Sample from true prior

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



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

Variational Autoencoders

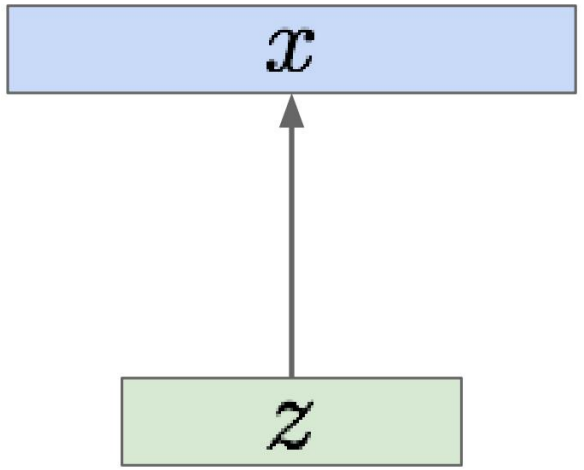
We want to estimate the true parameters θ^* of this generative model given training data x.

Sample from true conditional

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

Sample from true prior

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



How should we represent this model?

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

Variational Autoencoders

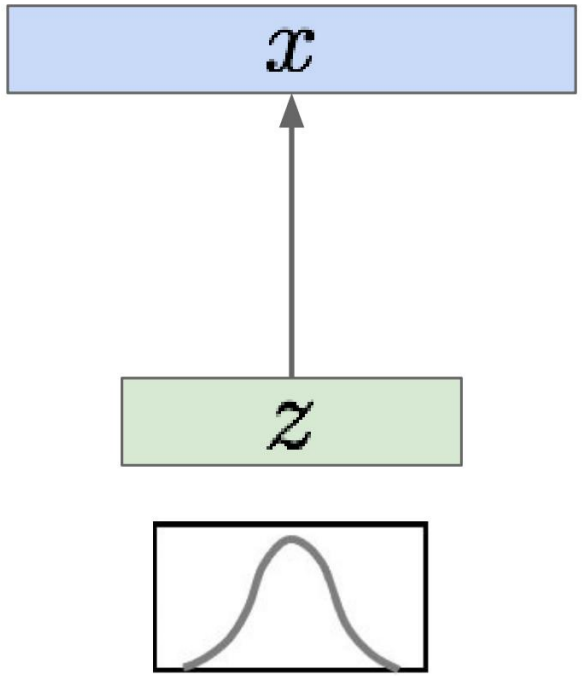
We want to estimate the true parameters θ^* of this generative model given training data x.

Sample from true conditional

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

Sample from true prior

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



How should we represent this model?

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

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

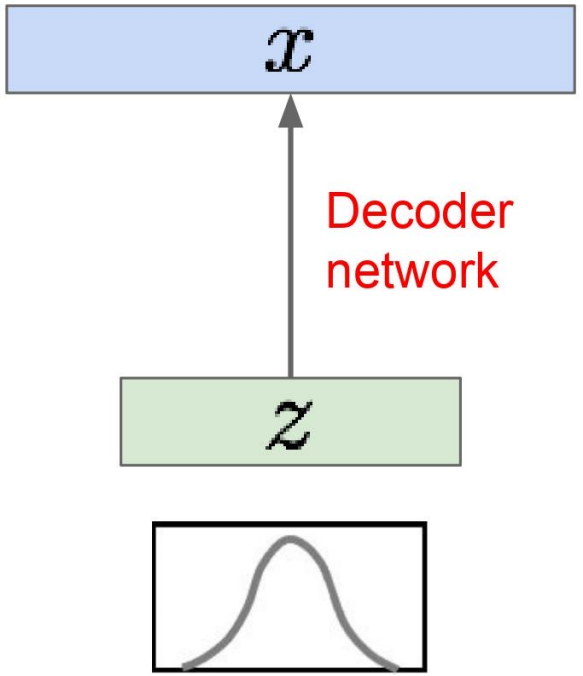
Variational Autoencoders

Sample from true conditional

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

Sample from true prior

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



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

How should we represent this model?

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

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

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

Variational Autoencoders

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

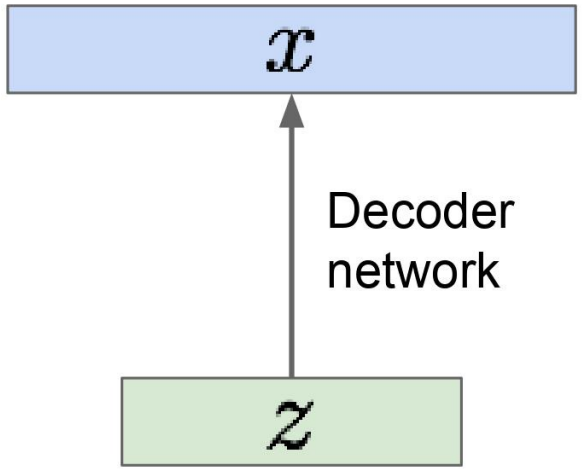
How to train the model?

Sample from true conditional

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

Sample from true prior

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



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

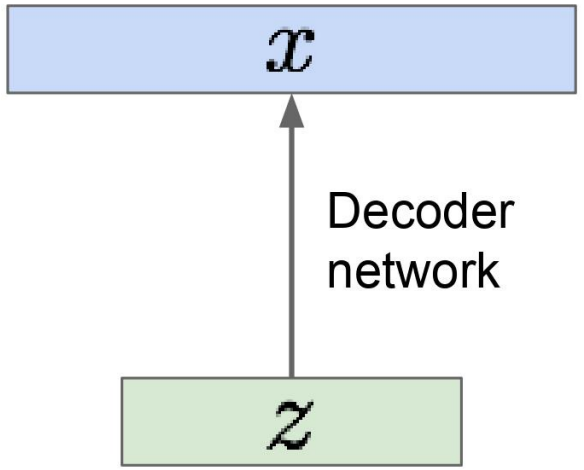
Variational Autoencoders

Sample from true conditional

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

Sample from true prior

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



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

How to train the model?

Learn model parameters to maximize likelihood of training data

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

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

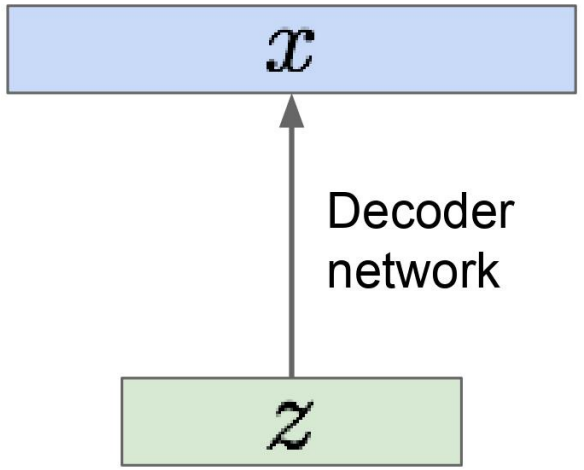
Variational Autoencoders

Sample from true conditional

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

Sample from true prior

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



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

How to train the model?

Learn model parameters to maximize likelihood of training data

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

Q: What is the problem with this?

Intractable!

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

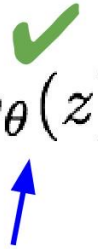
Variational Autoencoders: Intractability

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

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

Variational Autoencoders: Intractability

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

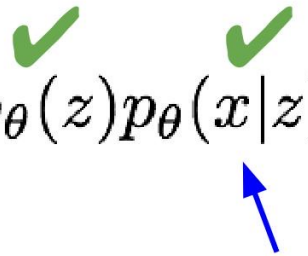


Simple Gaussian prior

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

Variational Autoencoders: Intractability

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

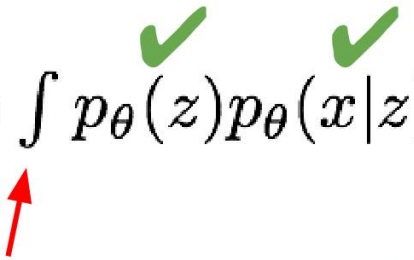


Decoder neural network

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

Variational Autoencoders: Intractability

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

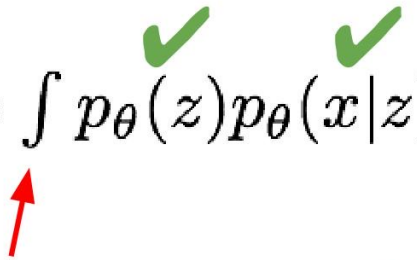


Intractable to compute p(x|z) for every z!

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

Variational Autoencoders: Intractability

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



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

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

Monte Carlo estimation is too high variance

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

Variational Autoencoders: Intractability

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

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

Intractable data likelihood

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

Variational Autoencoders: Intractability

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

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

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

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

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

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

Variational Autoencoders

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

Variational Autoencoders

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

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

Variational Autoencoders

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

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

Variational Autoencoders

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

Variational Autoencoders

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

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

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

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

Variational Autoencoders

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

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

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

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

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

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

Variational Autoencoders

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

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

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

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

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

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

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

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

Variational Autoencoders

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

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

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

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

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

We want to maximize the data likelihood

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

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

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

Variational Autoencoders

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

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

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

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

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

We want to maximize the data likelihood

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

Variational Autoencoders

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

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

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

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

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

Decoder: reconstruct the input data

Encoder: make approximate posterior distribution close to prior

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

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

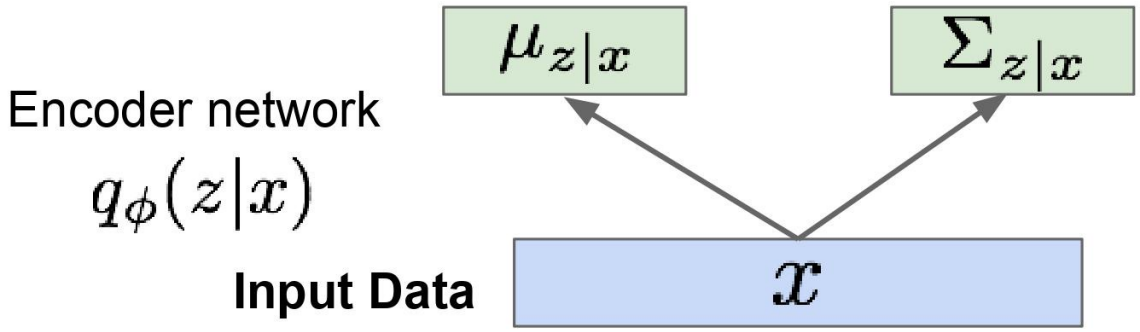
$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

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

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

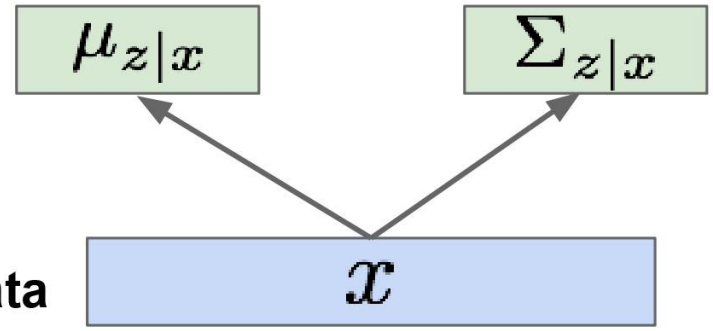
$$D_{KL}(\mathcal{N}(\mu_{z|x}, \Sigma_{z|x})||\mathcal{N}(0, I))$$

Have analytical solution

Make approximate posterior distribution close to prior

Encoder network

$q_\phi(z|x)$



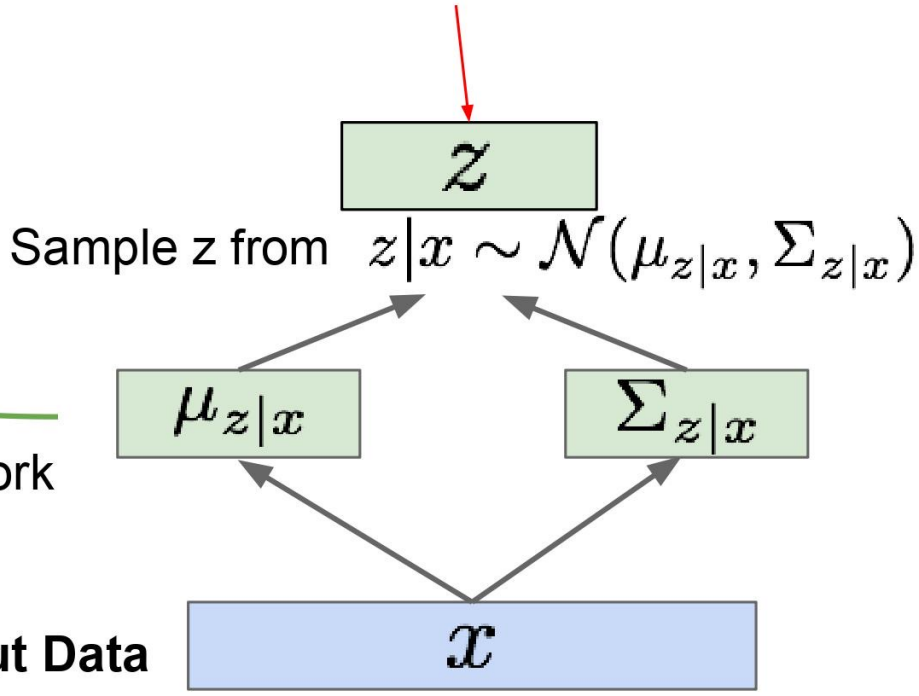
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Make approximate posterior distribution close to prior

Not part of the computation graph!



Variational Autoencoders

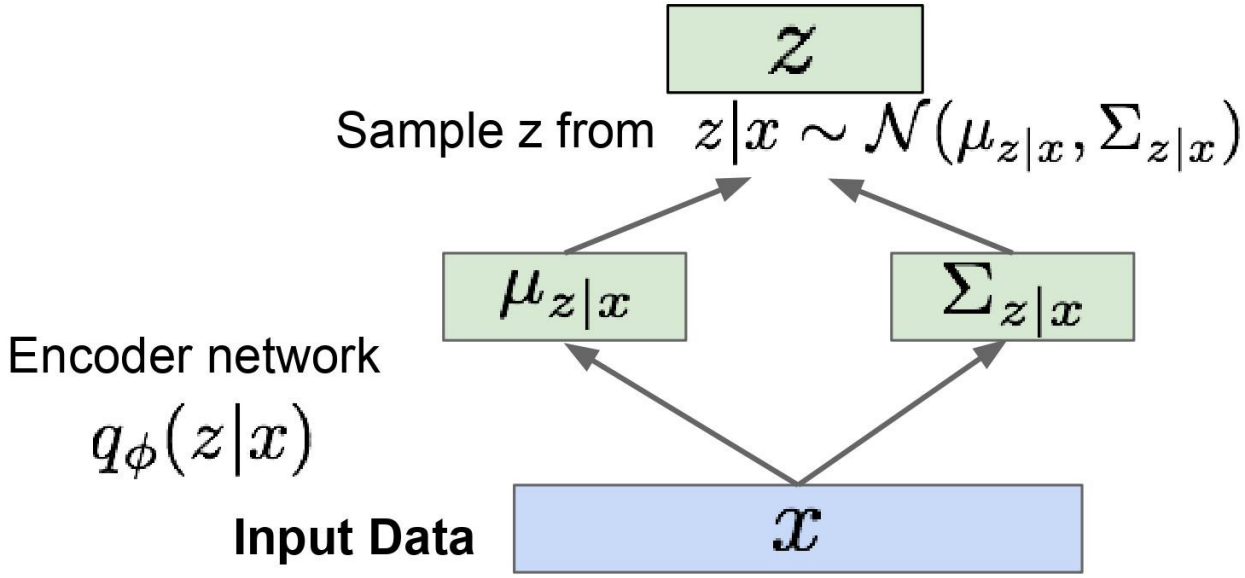
Putting it all together: maximizing the likelihood lower bound

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

Reparameterization trick to make sampling differentiable:

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

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



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

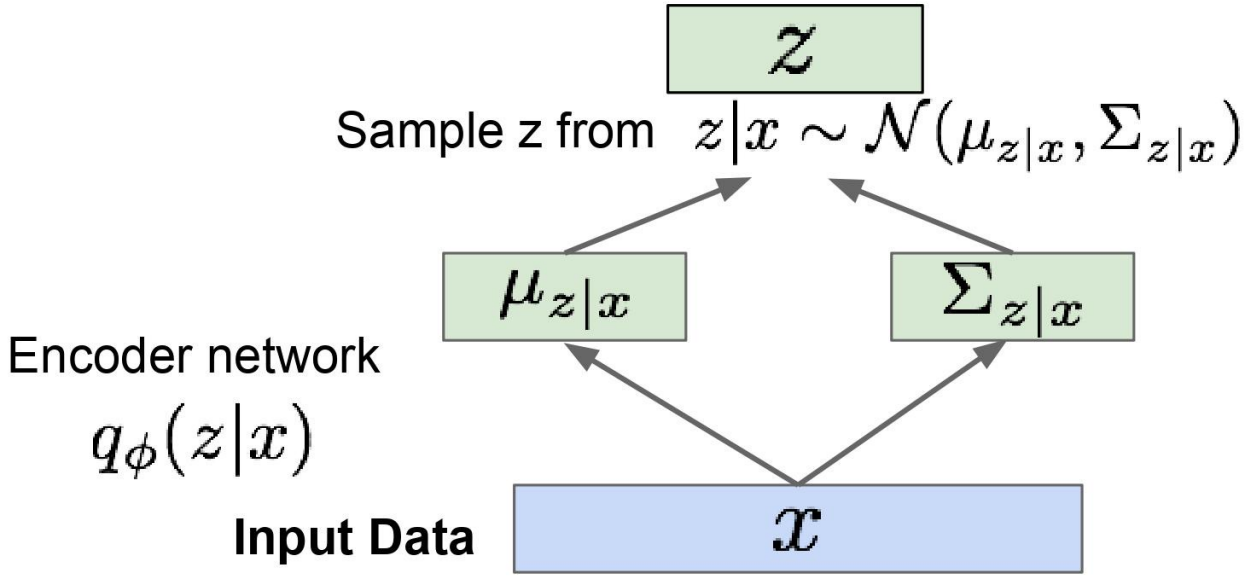
Reparameterization trick to make sampling differentiable:

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

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

Input to the graph

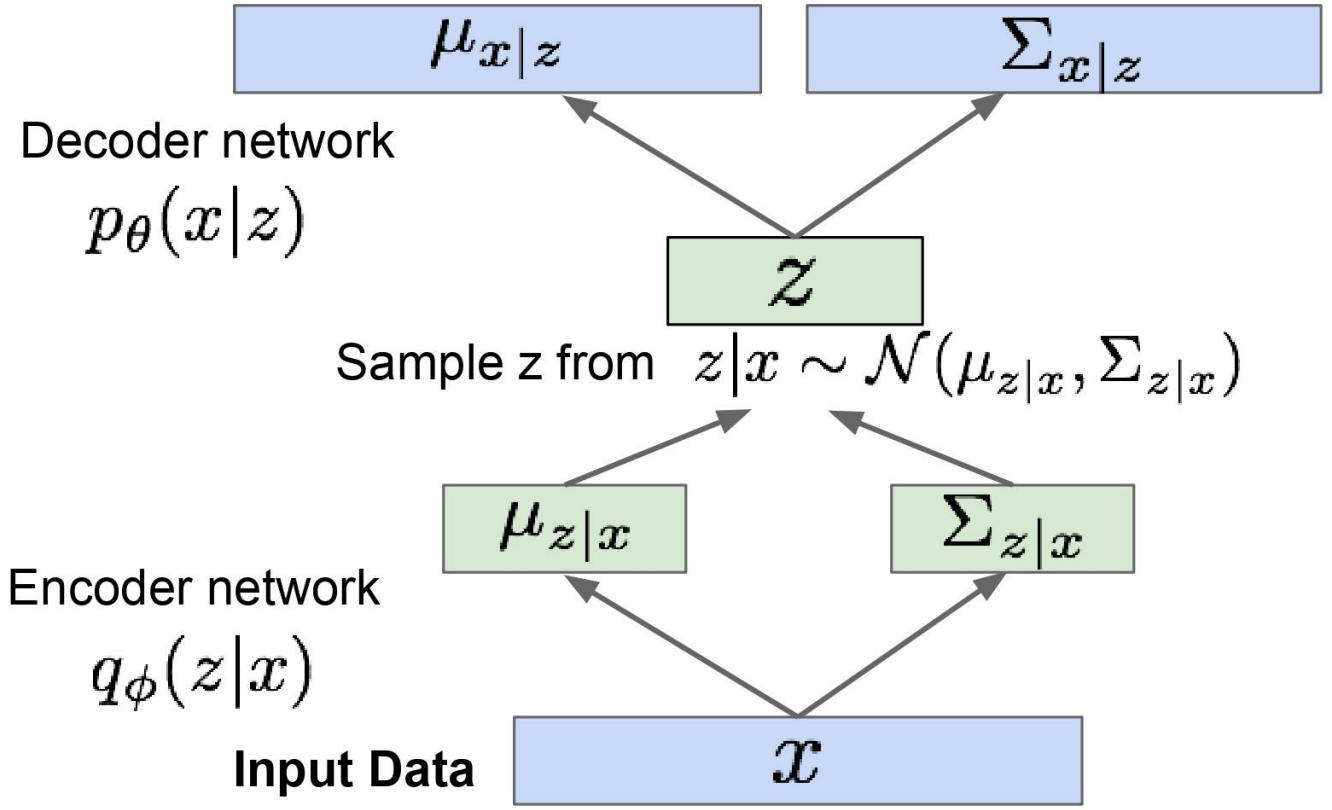
Part of computation graph



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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



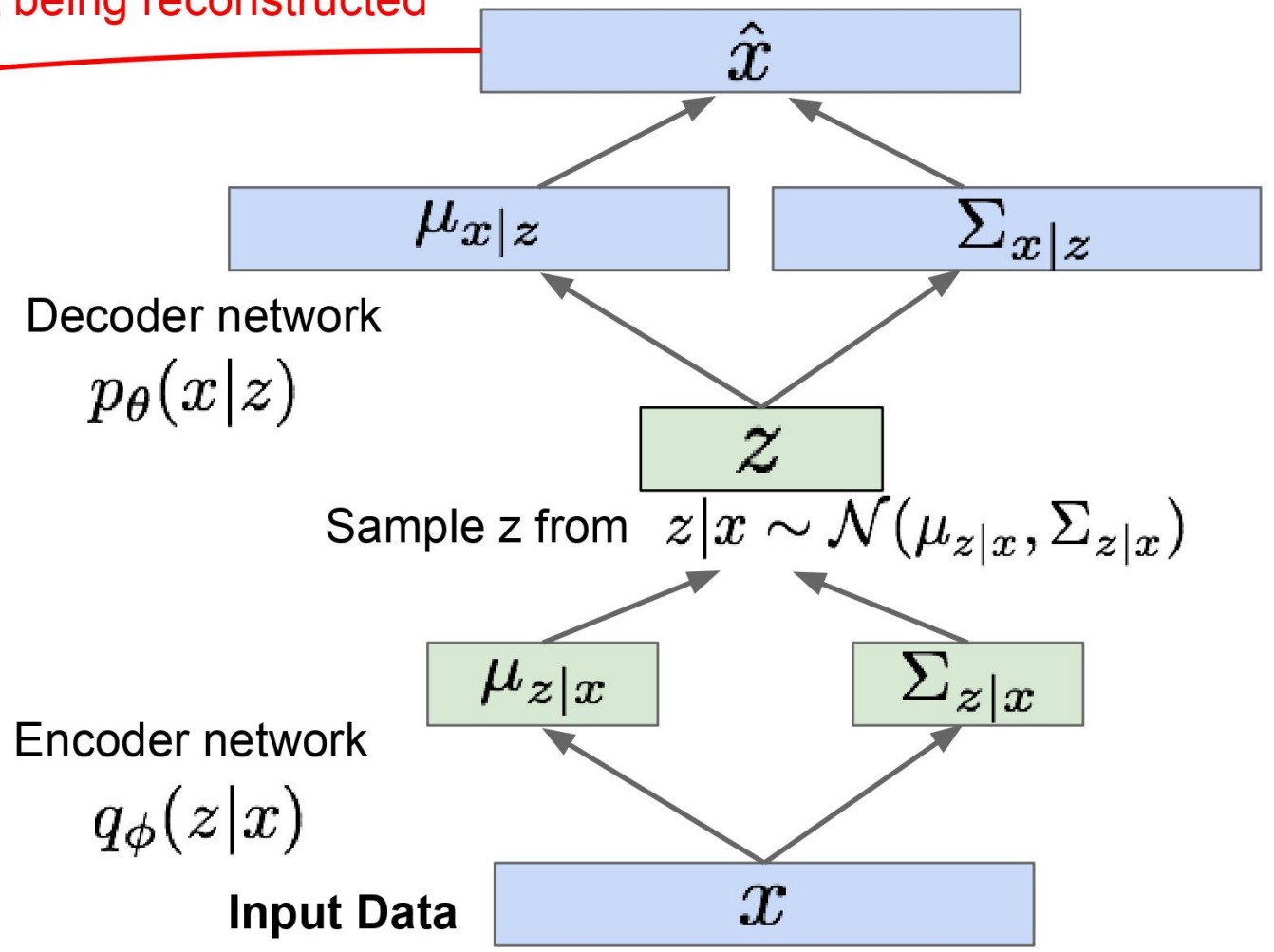
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

Maximize likelihood of original input being reconstructed

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

$$\mathcal{L}(x^{(i)}, \theta, \phi)$$

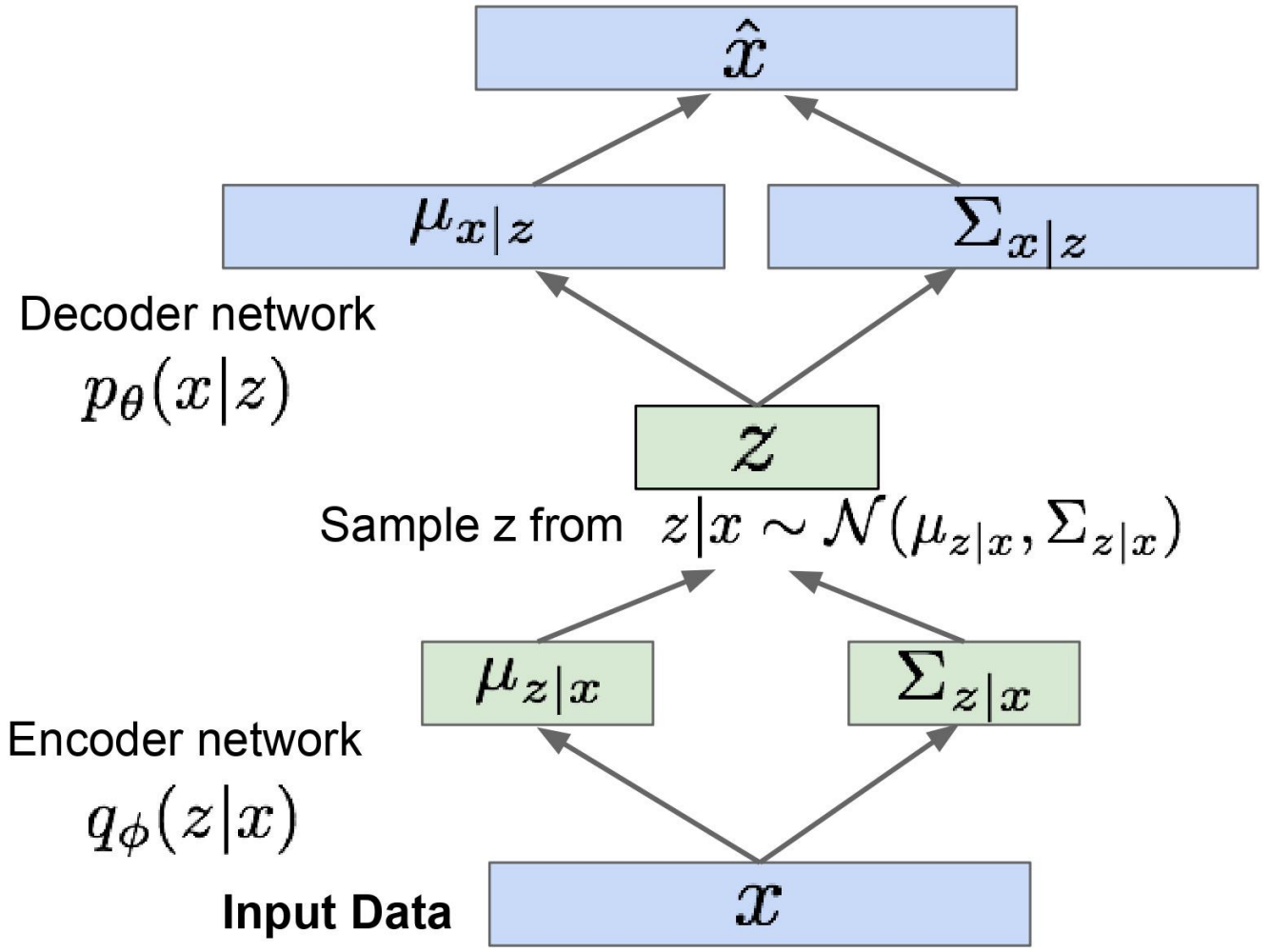


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

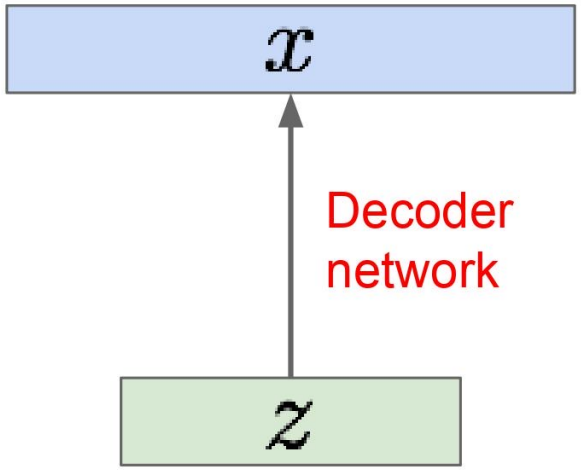


Variational Autoencoders: Generating Data!

Our assumption about data generation process

Sample from true conditional

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



Sample from true prior

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

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

Variational Autoencoders: Generating Data!

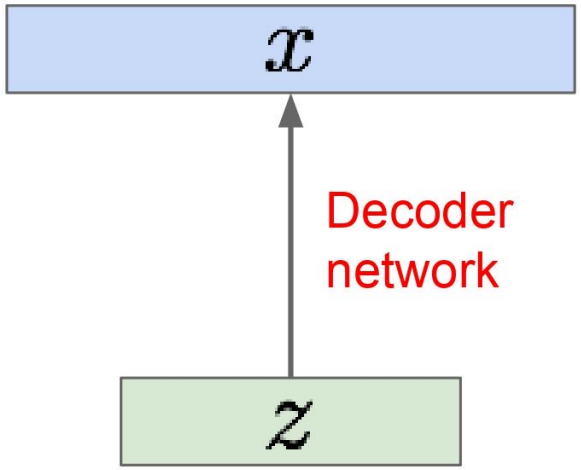
Our assumption about data generation process

Sample from true conditional

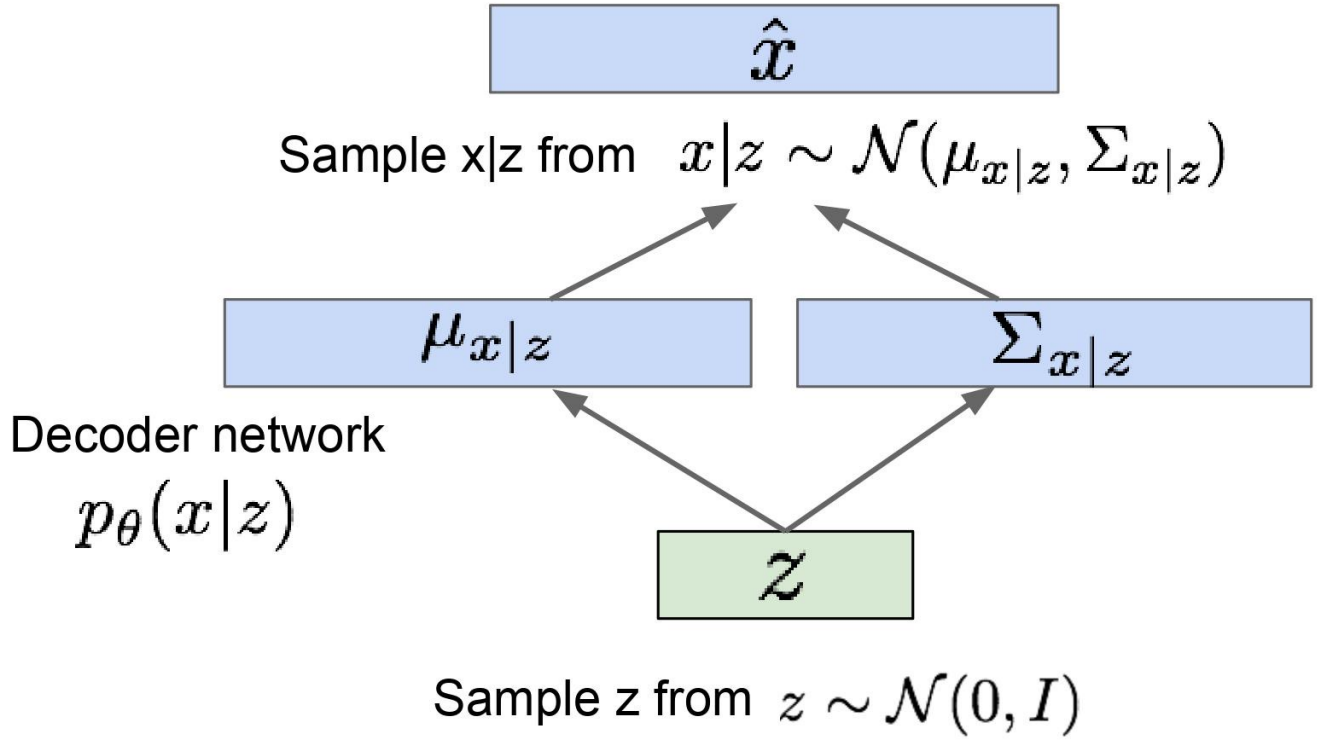
$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

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



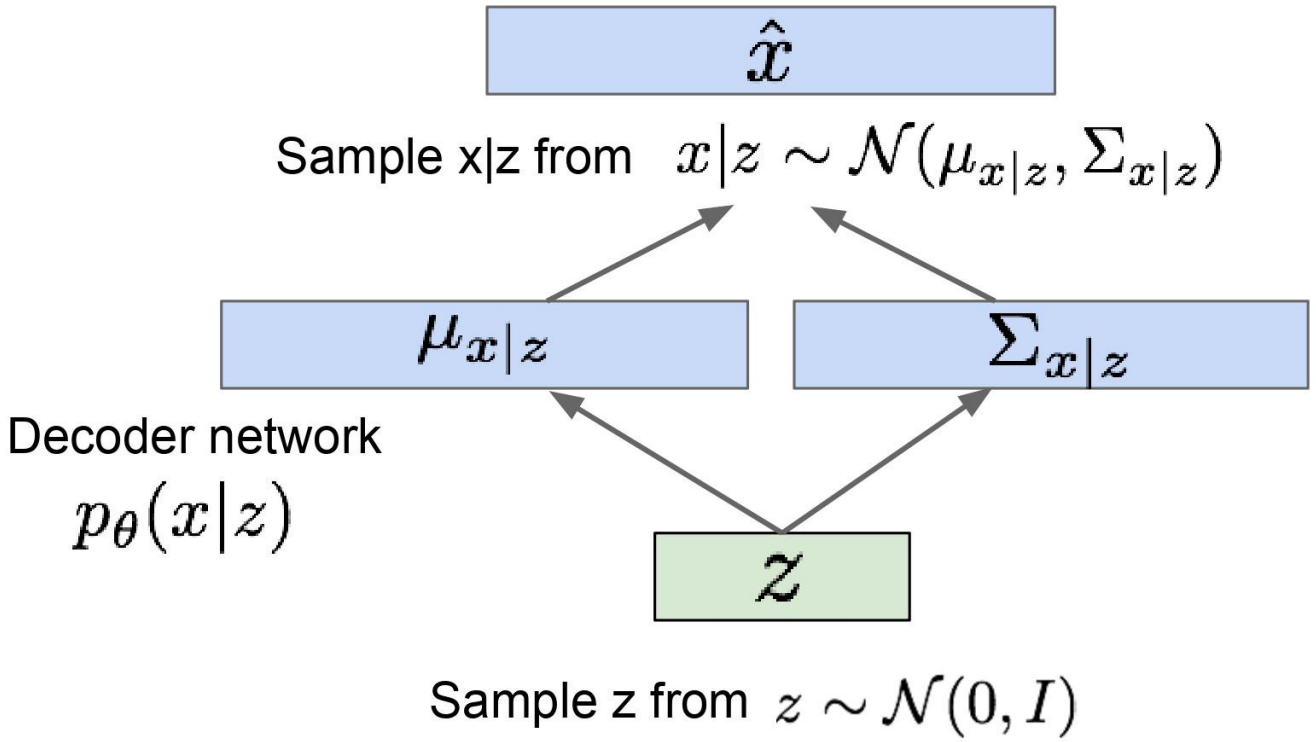
Now given a trained VAE:
use decoder network & sample z from prior!

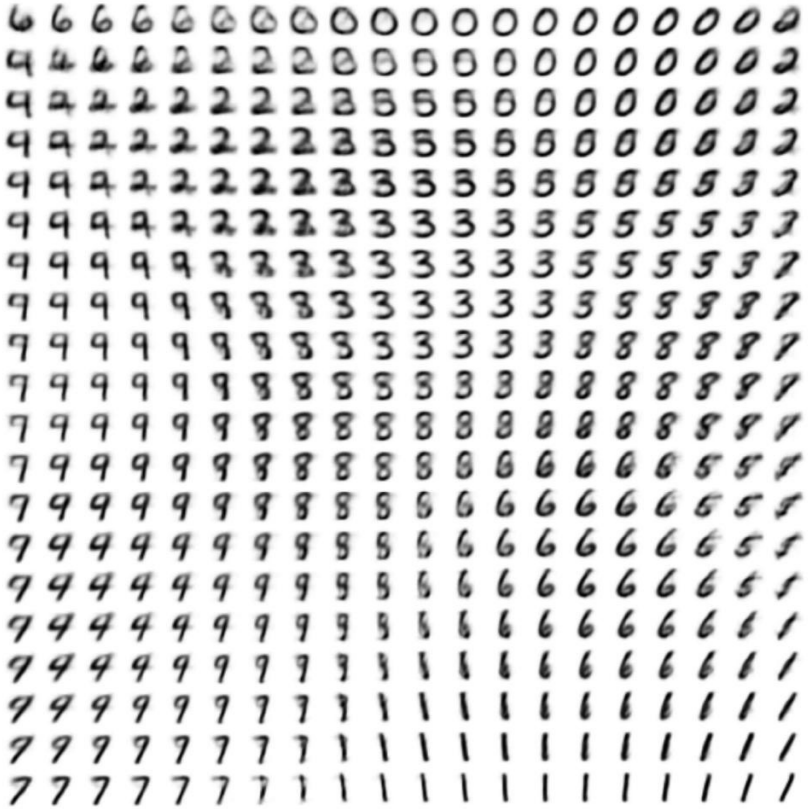


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!

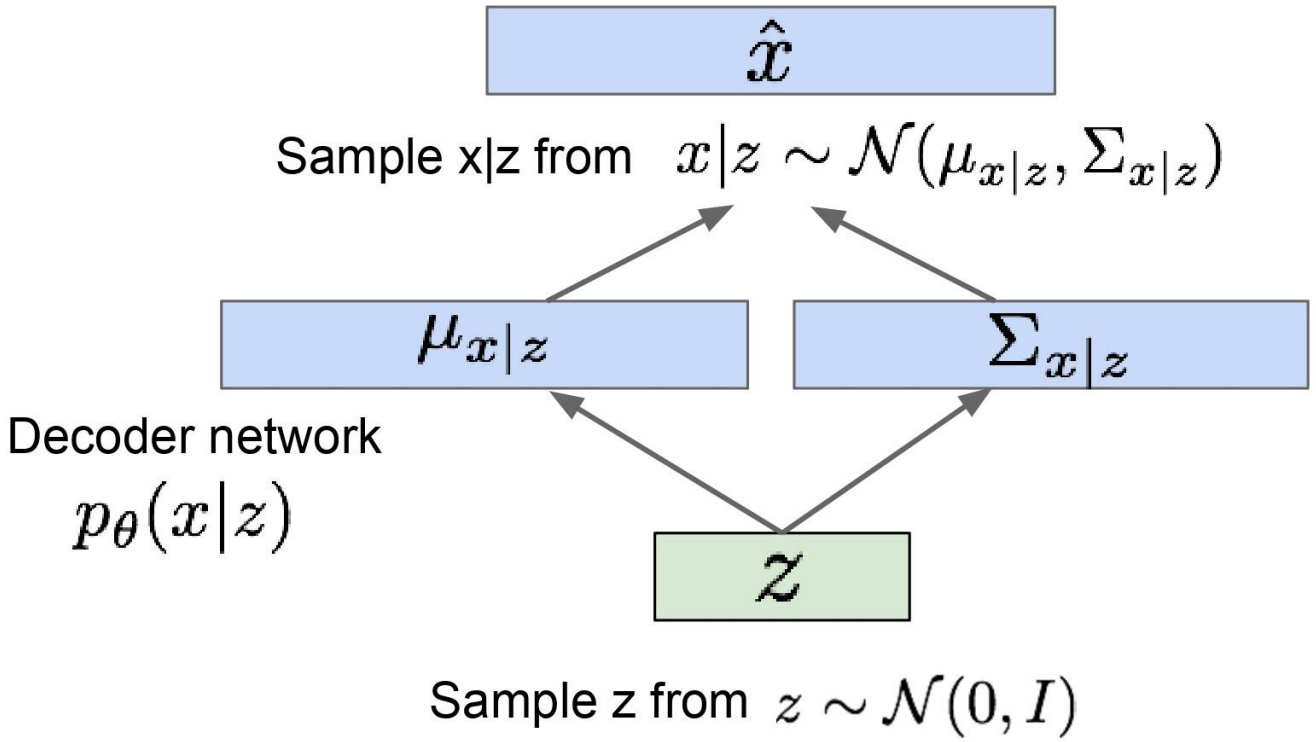




Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

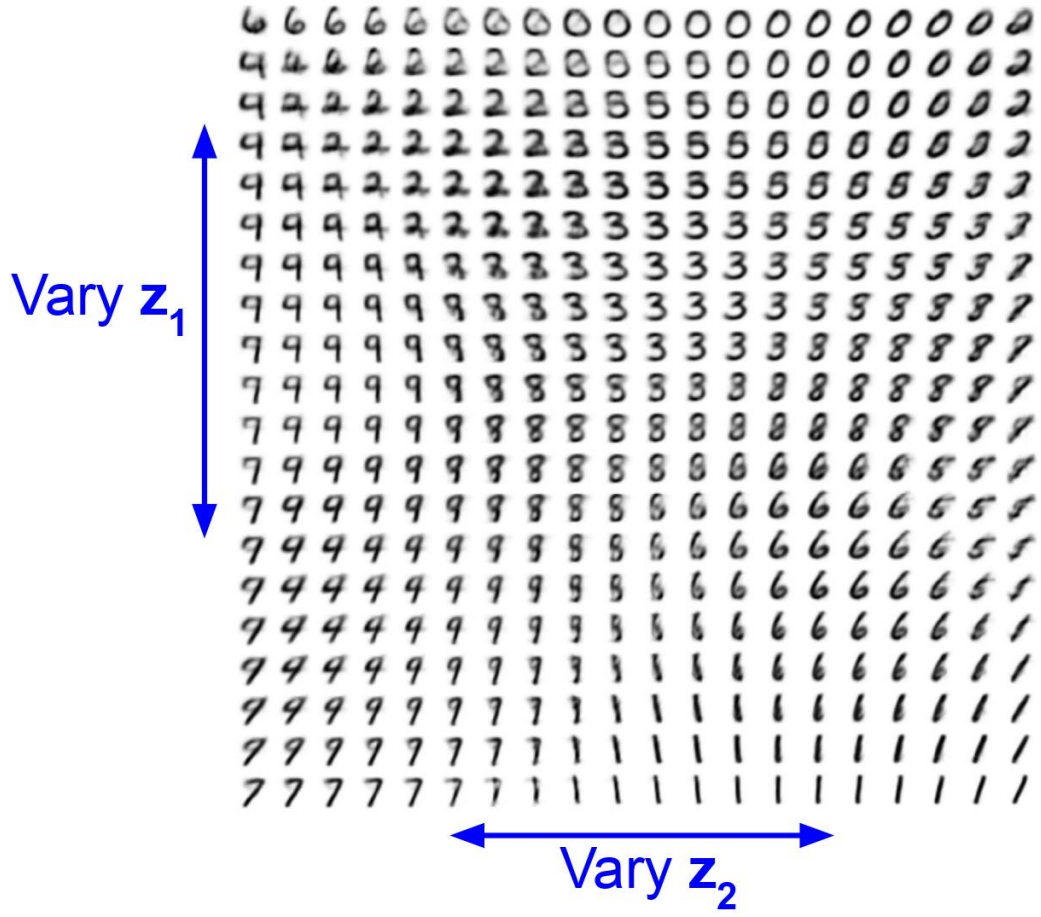
Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!



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

Data manifold for 2-d **z**



Variational Autoencoders: Generating Data!

Diagonal prior on **z**
=> independent
latent variables

Different
dimensions of **z**
encode
interpretable factors
of variation

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

Variational Autoencoders: Generating Data!

Diagonal prior on **z**
=> independent
latent variables

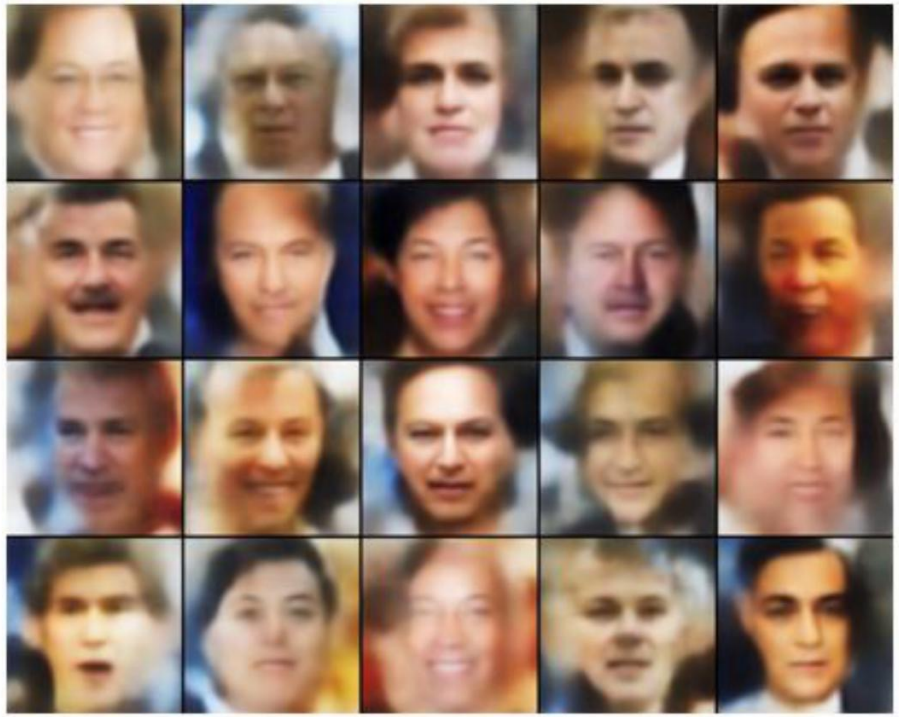
Different
dimensions of **z**
encode
interpretable factors
of variation

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

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

Variational Autoencoders: Generating Data!

32x32 CIFAR-10



Labeled Faces in the Wild

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

Variational Autoencoders

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

Pros:

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

Cons:

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

Active areas of research:

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

Taxonomy of Generative Models

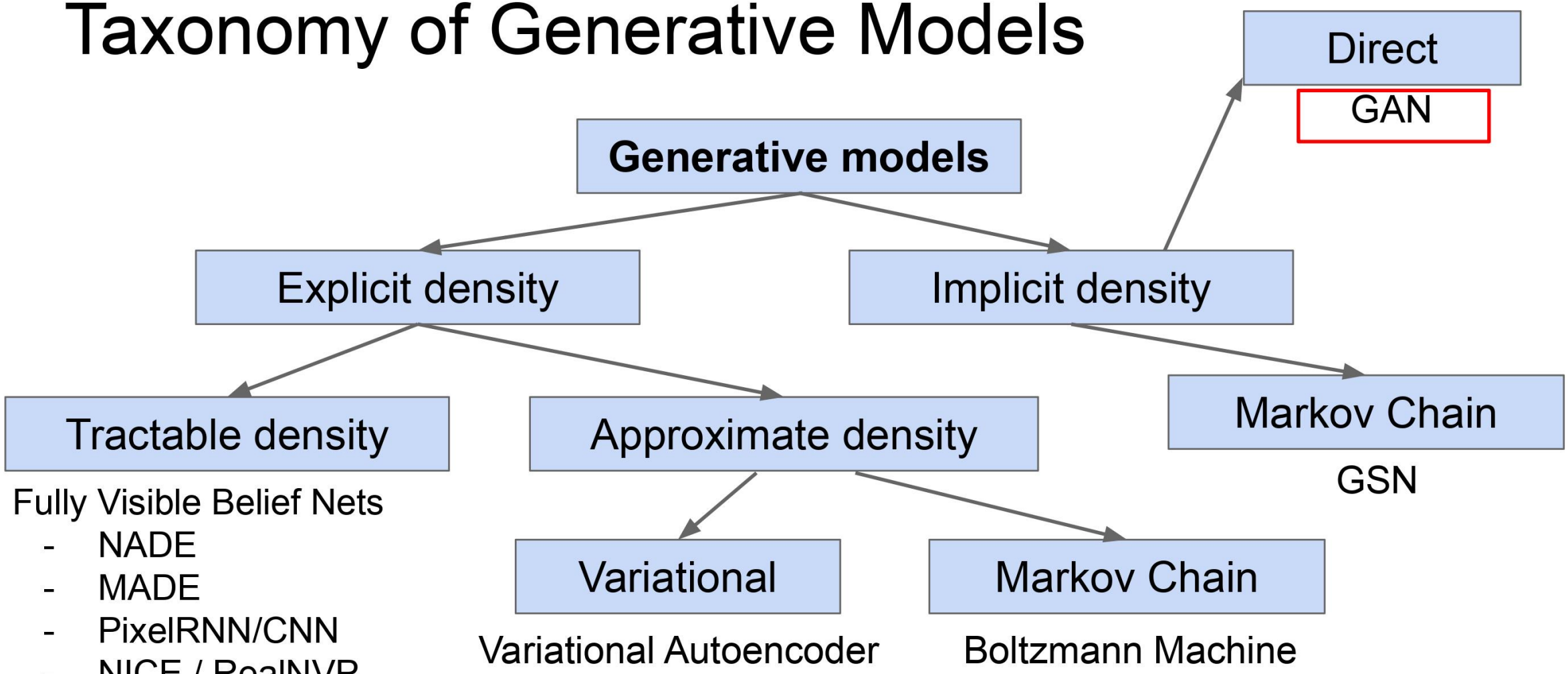


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Generative Adversarial Networks (GANs)

So far...

PixelCNNs define tractable density function, optimize likelihood of training data:

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

VAEs define intractable density function with latent **z**:

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

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

So far...

PixelCNNs define tractable density function, optimize likelihood of training data:

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

VAEs define intractable density function with latent **z**:

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

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

What if we give up on explicitly modeling density, and just want ability to sample?

So far...

PixelCNNs define tractable density function, optimize likelihood of training data:

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

VAEs define intractable density function with latent **z**:

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

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

What if we give up on explicitly modeling density, and just want ability to sample?

GANs: not modeling any explicit density function!

Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

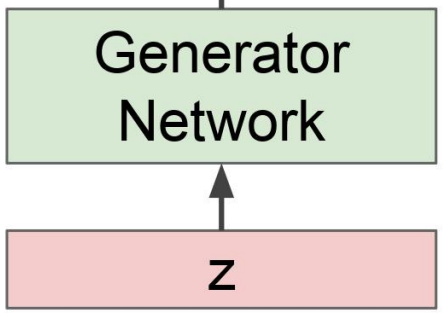
Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

Output: Sample from training distribution



Input: Random noise

Generative Adversarial Networks

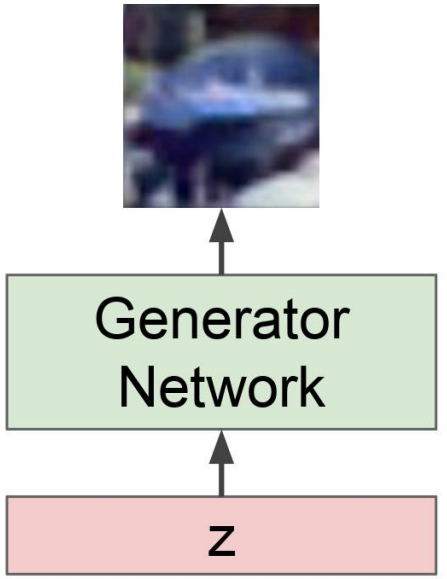
Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Output: Sample from training distribution



Input: Random noise

Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

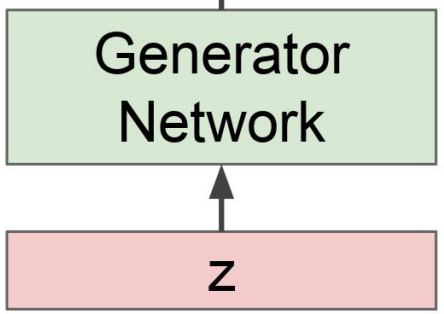
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Output: Sample from training distribution

Objective: generated images should look "real"



Input: Random noise

Generative Adversarial Networks

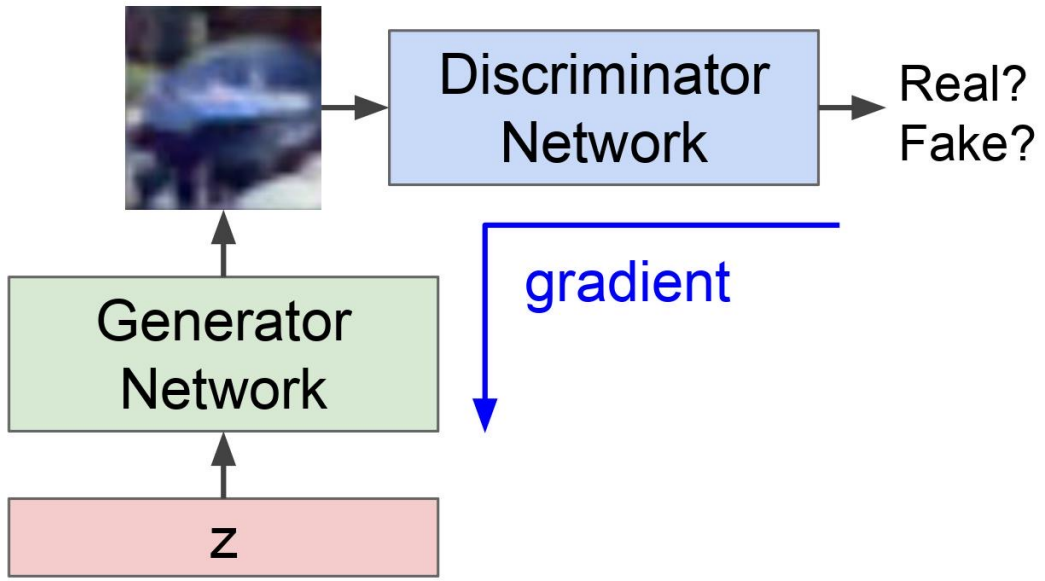
Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Solution: Use a discriminator network to tell whether the generate image is within data distribution ("real") or not



Training GANs: Two-player game

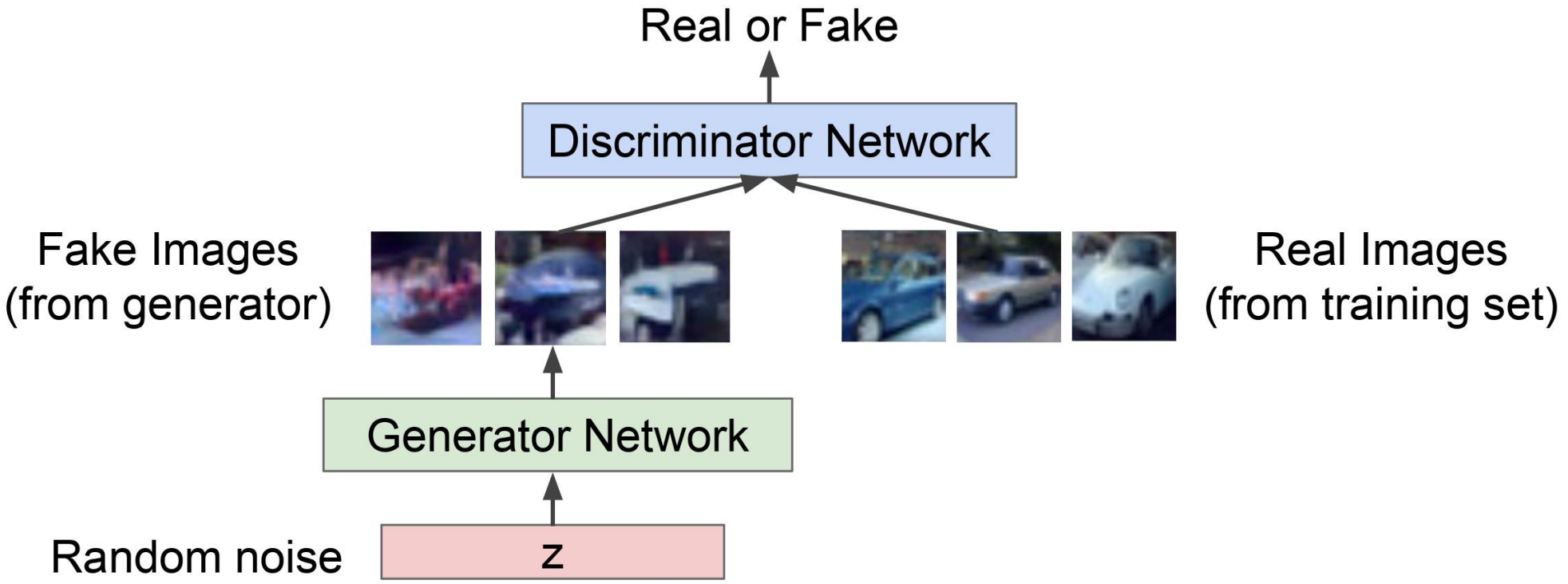
Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

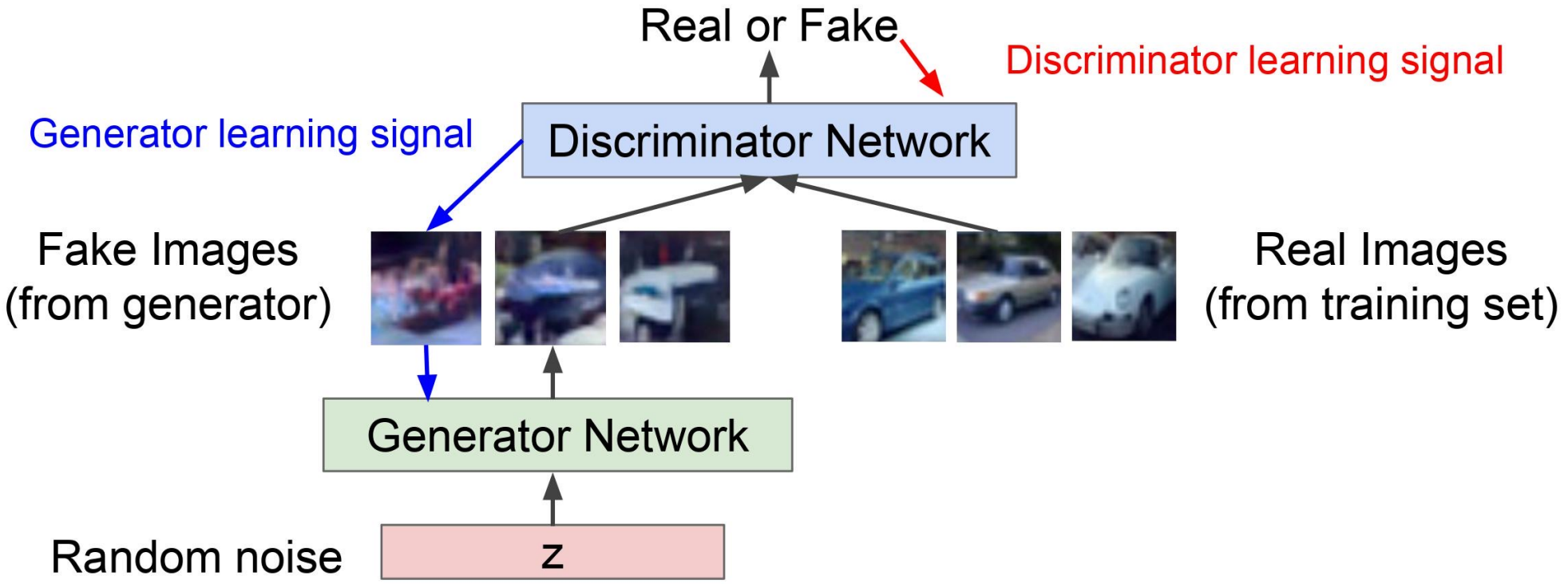


Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images



Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Generator objective

Discriminator objective

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

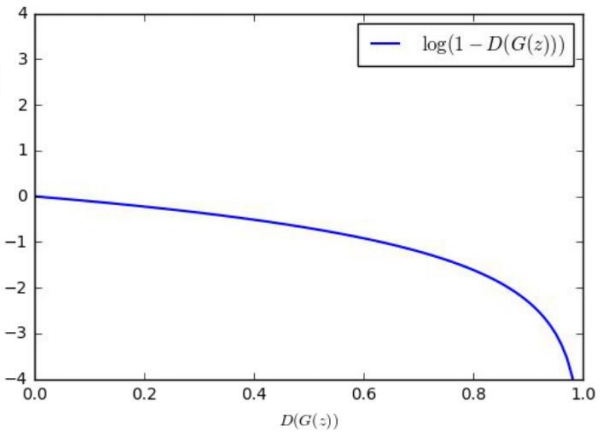
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

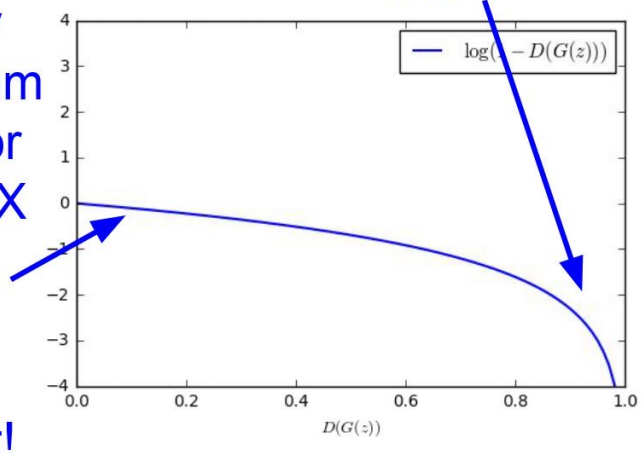
$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).

But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

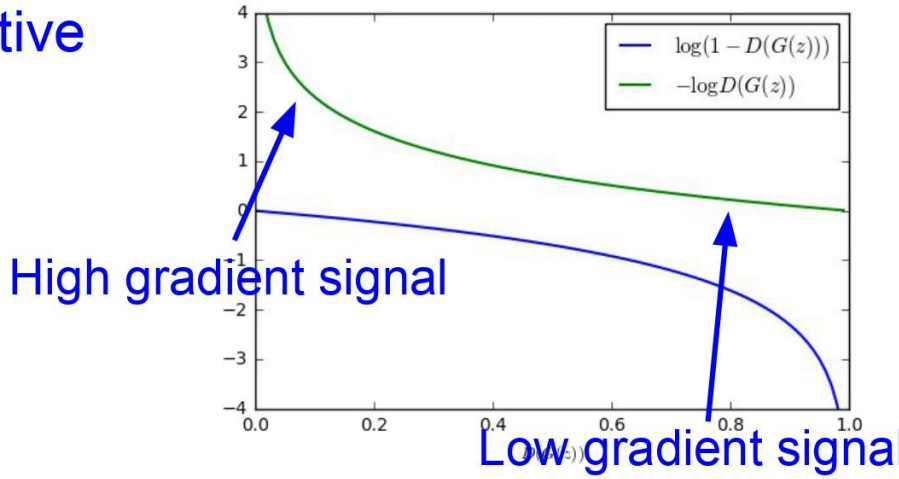
1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Putting it together: GAN training algorithm

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Putting it together: GAN training algorithm

for number of training iterations **do**

  for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Some find k=1 more stable, others use k > 1, no best rule.

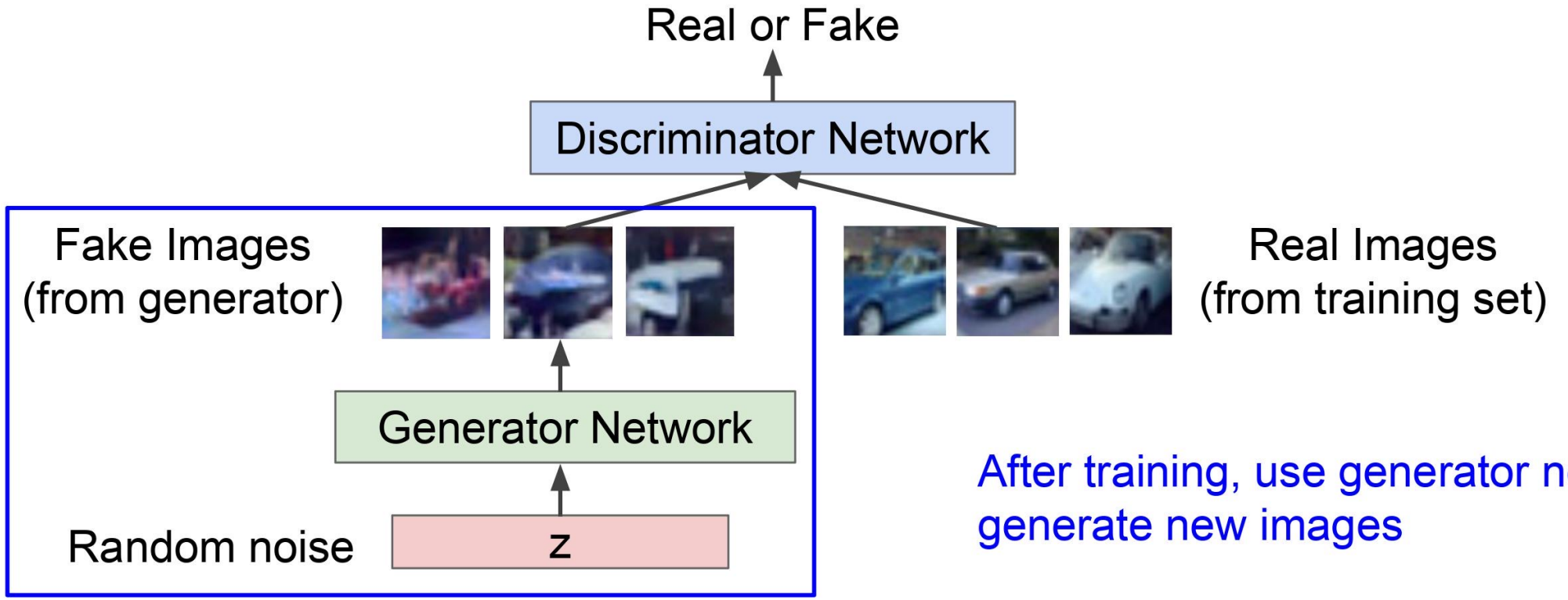
Followup work (e.g. Wasserstein GAN, BEGAN) alleviates this problem, better stability!

Arjovsky et al. "Wasserstein gan." arXiv preprint arXiv:1701.07875 (2017)
Berthelot, et al. "Began: Boundary equilibrium generative adversarial networks." arXiv preprint arXiv:1703.10717 (2017)

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images



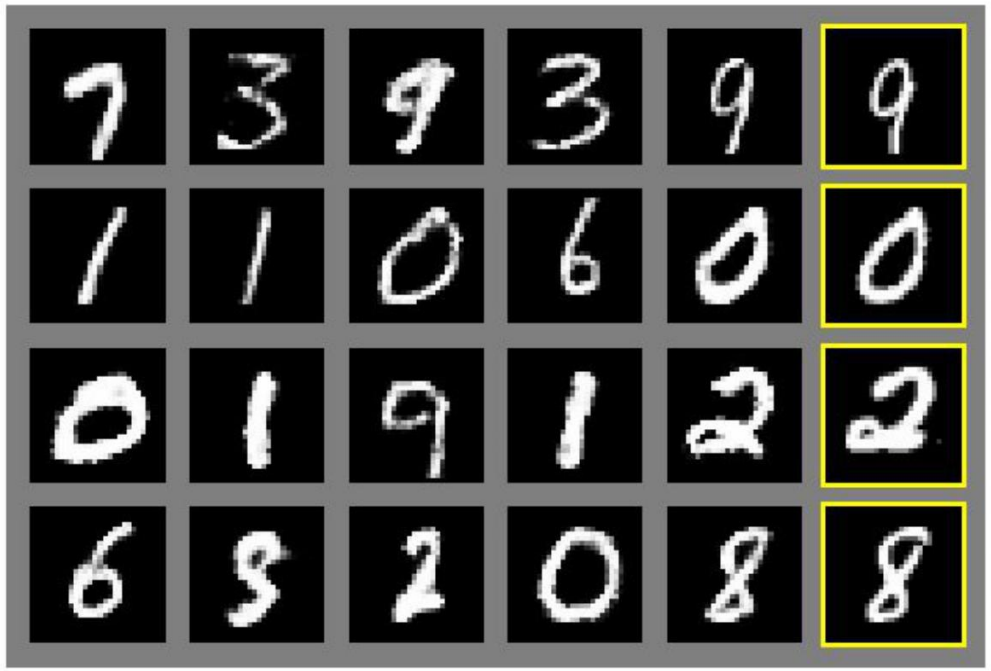
After training, use generator network to generate new images

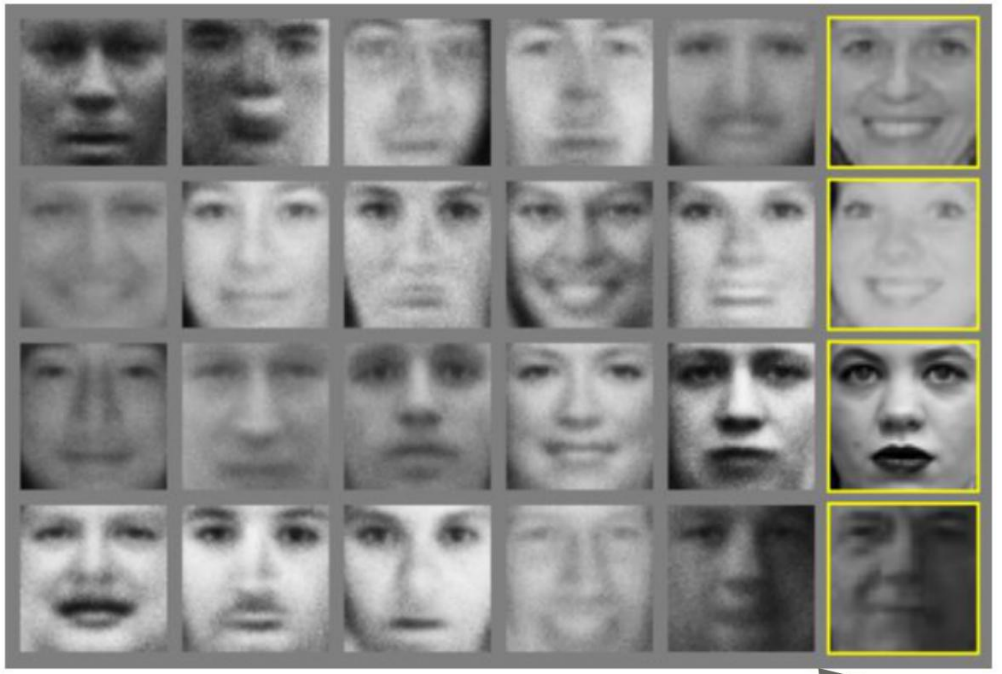
Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generative Adversarial Nets

Generated samples



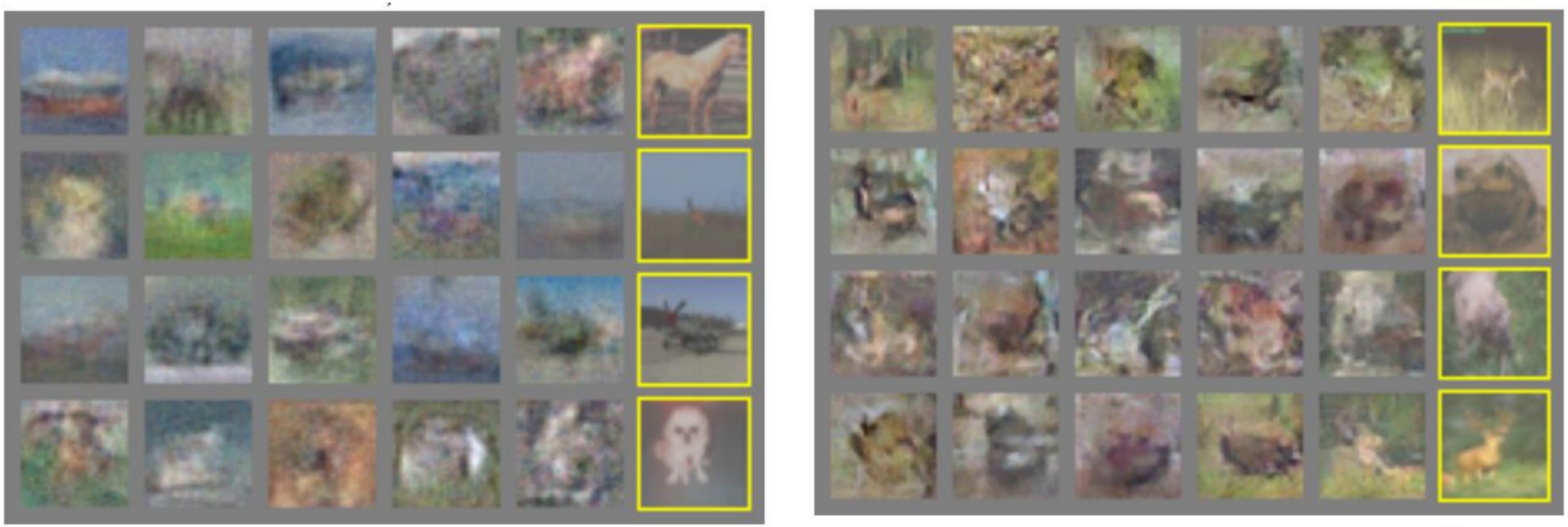


Nearest neighbor from training set

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generative Adversarial Nets

Generated samples (CIFAR-10)



Nearest neighbor from training set

Generative Adversarial Nets: Convolutional Architectures

Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

Samples from the model look much better!

Radford et al, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

Interpolating between random points in laten space

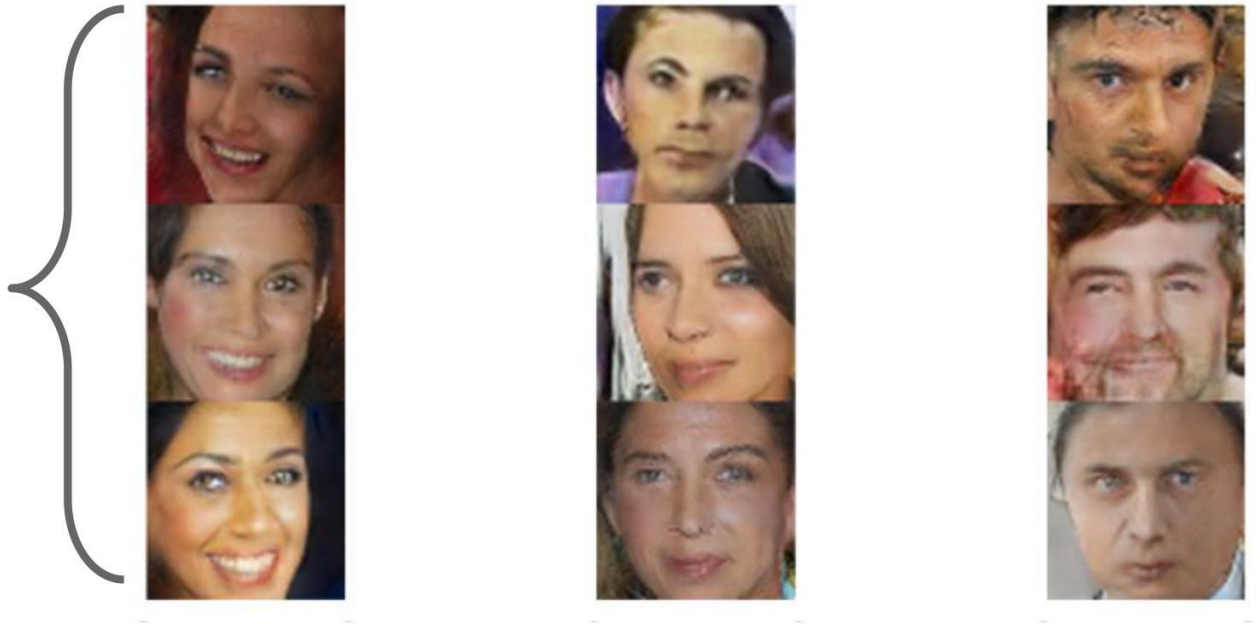
Radford et al, ICLR 2016

Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

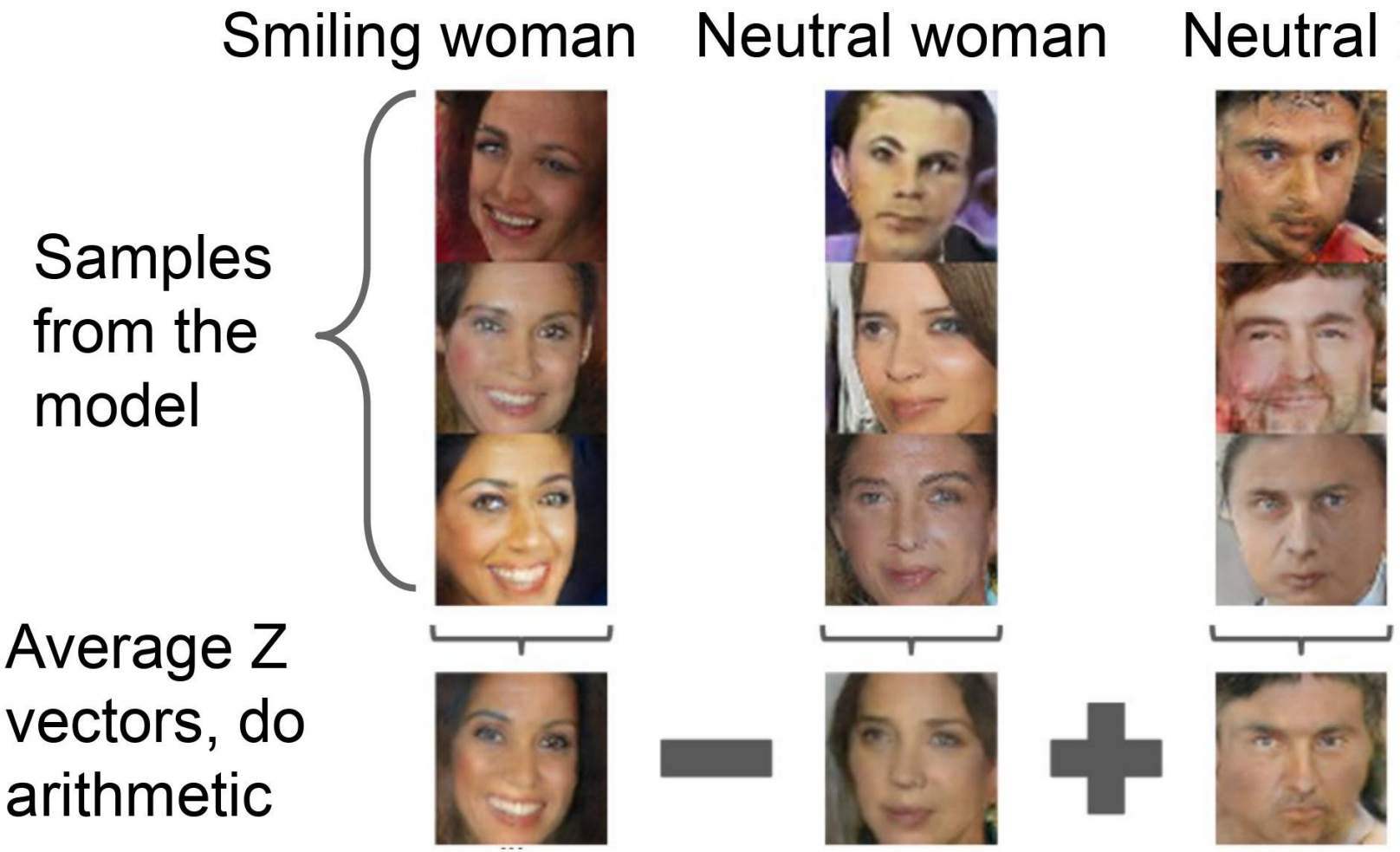
Smiling woman | Neutral woman | Neutral man

Samples from the model



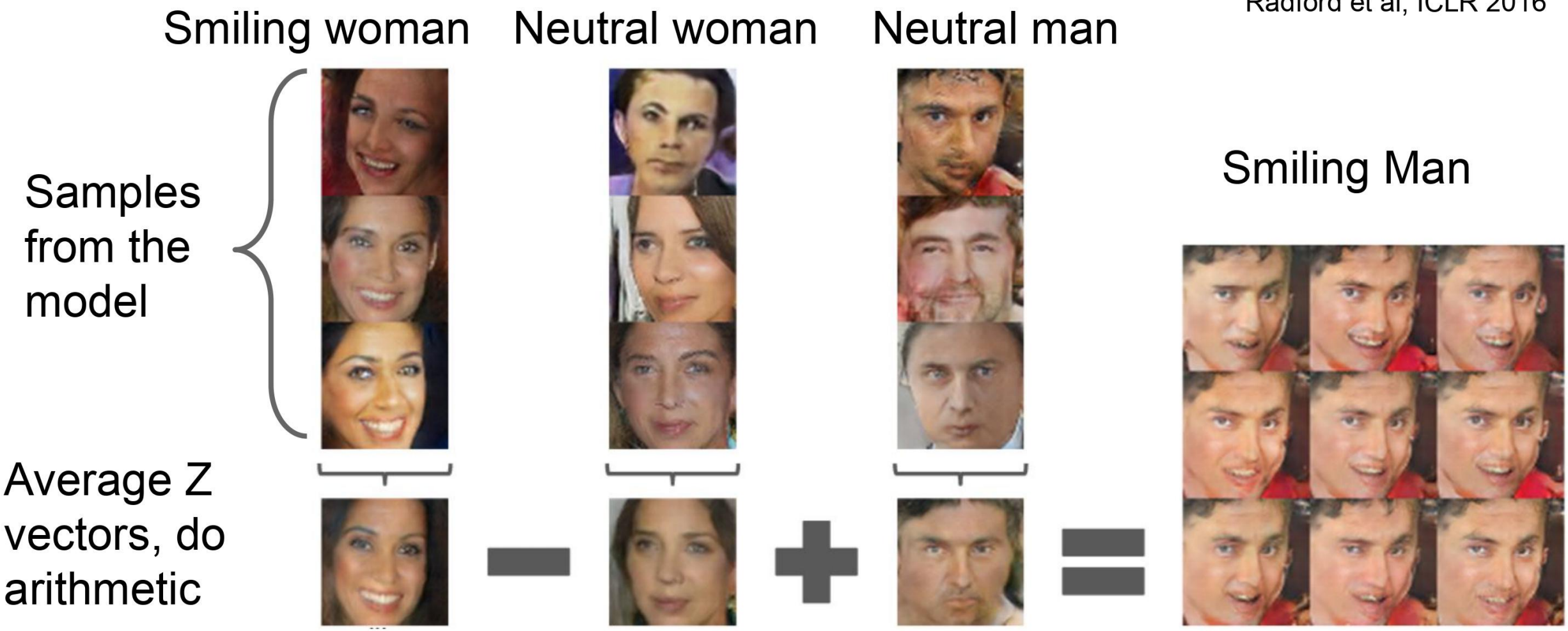
Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016



Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016



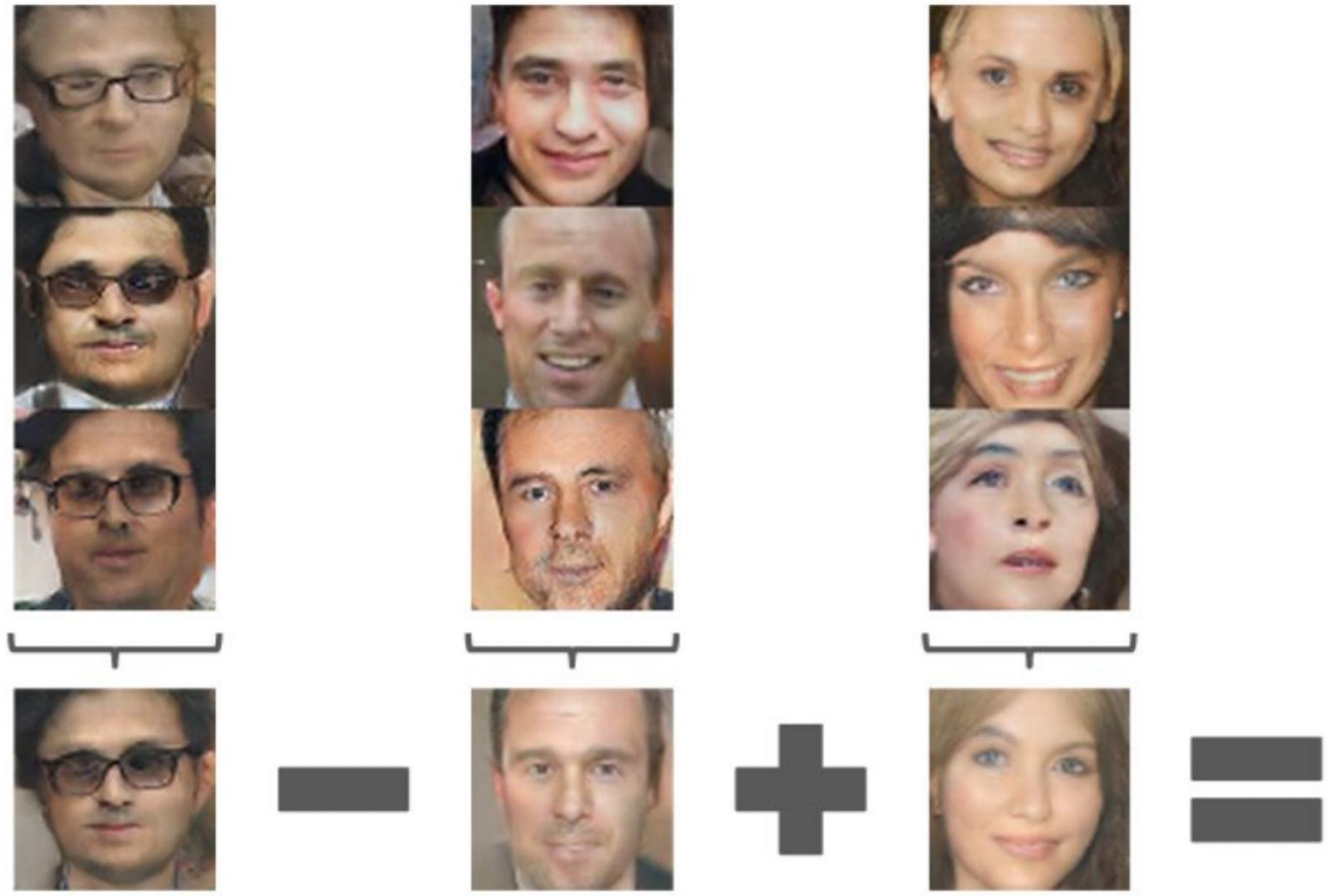
Generative Adversarial Nets: Interpretable Vector Math

Glasses man

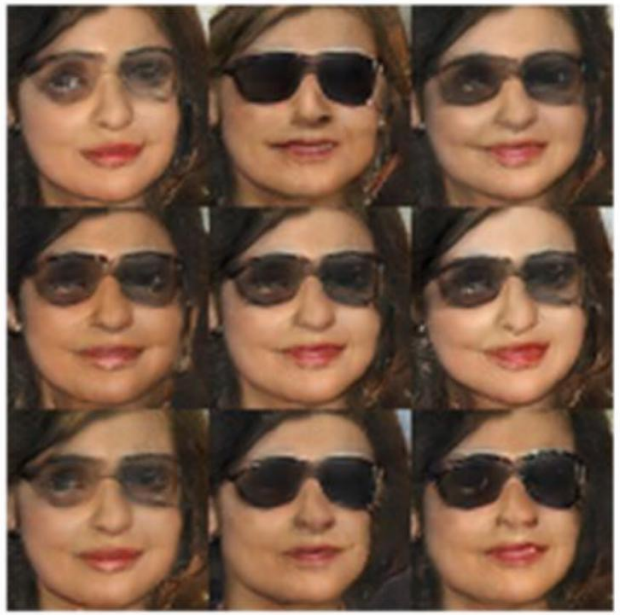
No glasses man

No glasses woman

Radford et al, ICLR 2016



Woman with glasses



2017: Explosion of GANs

See also: https://github.com/soumith/ganhacks for tips and tricks for trainings GANs

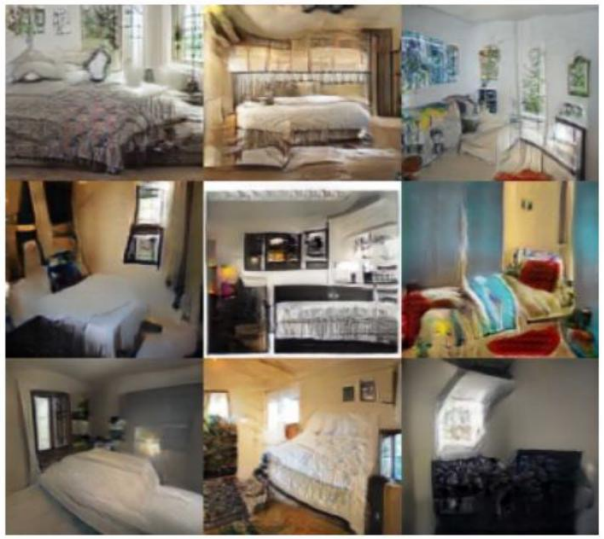
"The GAN Zoo"

- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorial GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

https://github.com/hindupuravinash/the-gan-zoo

2017: Explosion of GANs

Better training and generation



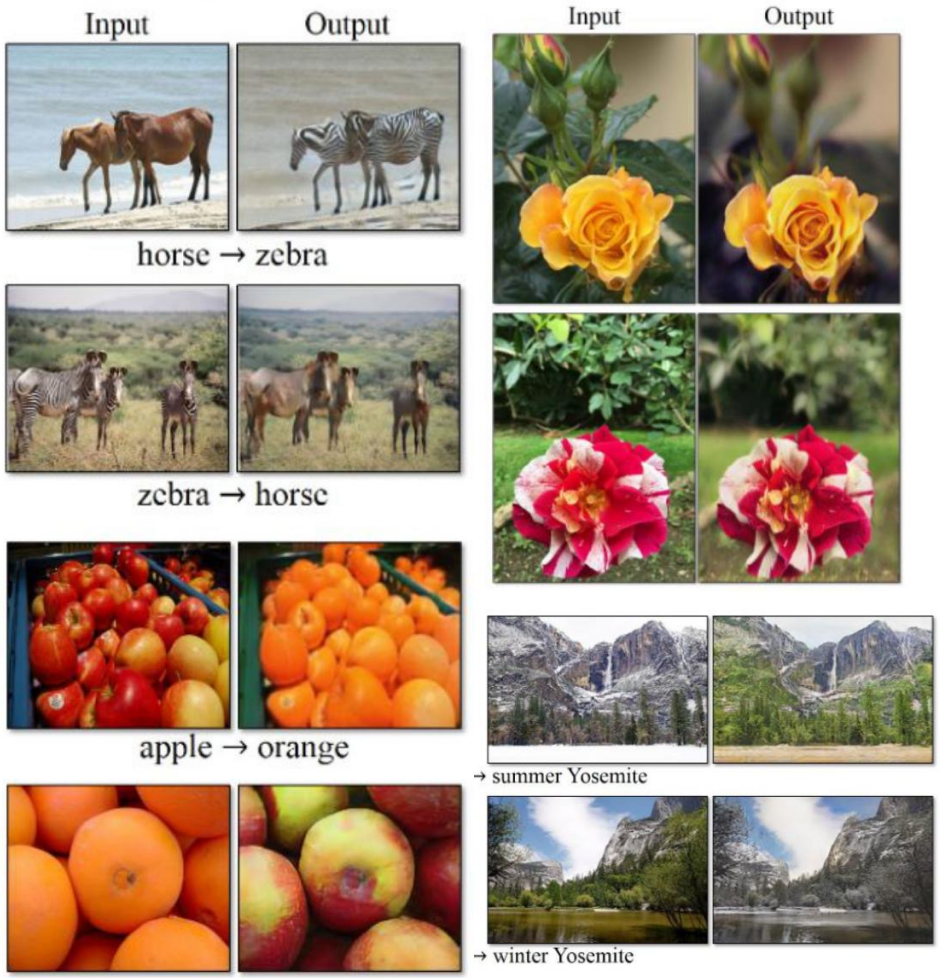
LSGAN, Zhu 2017.

Wasserstein GAN, Arjovsky 2017. Improved Wasserstein GAN, Gulrajani 2017.

Progressive GAN, Karras 2018.

2017: Explosion of GANs

Source->Target domain transfer



CycleGAN. Zhu et al. 2017.

Text -> Image Synthesis

Reed et al. 2017.

Many GAN applications

Pix2pix. Isola 2017. Many examples at https://phillipi.github.io/pix2pix/

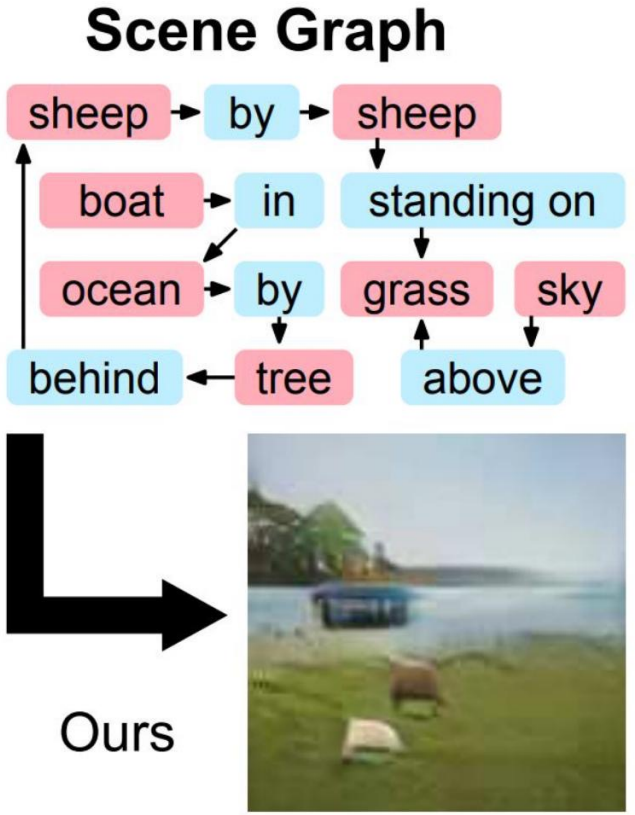
2019: BigGAN

Brock et al., 2019

Scene graphs to GANs

Specifying exactly what kind of image you want to generate.

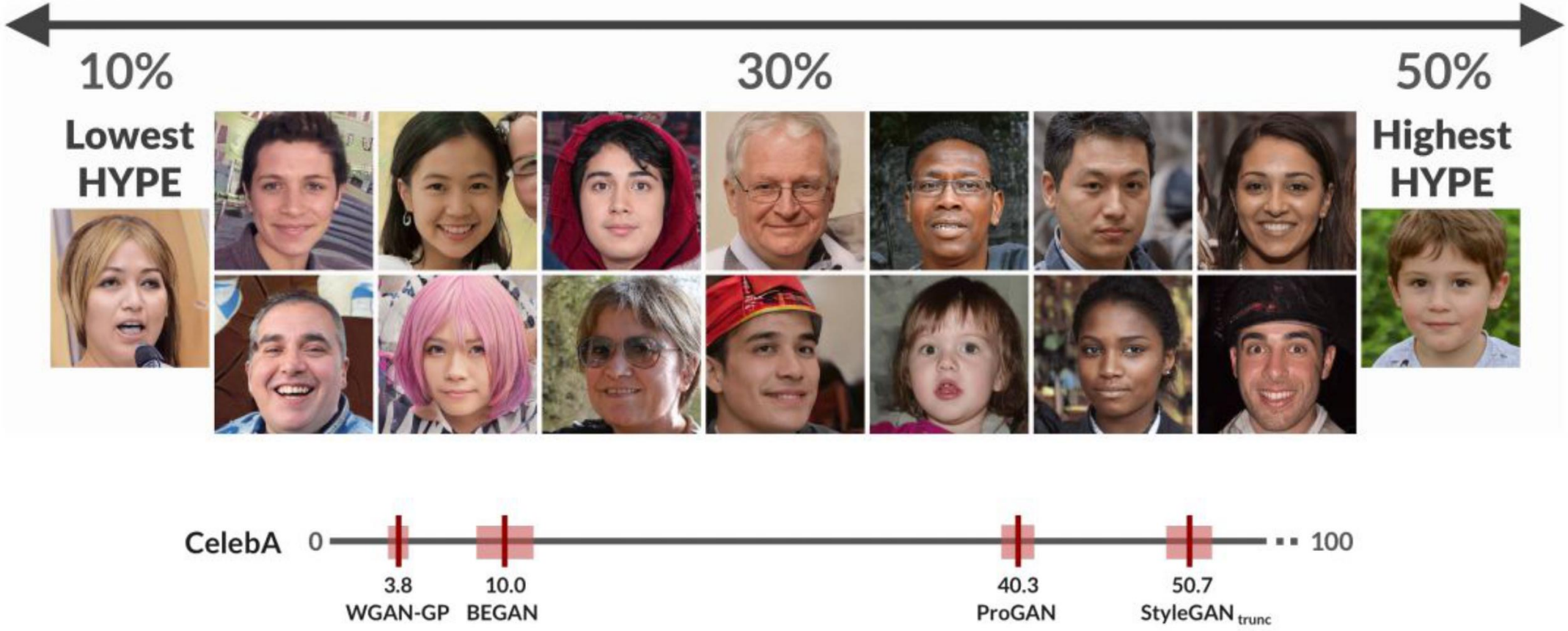
The explicit structure in scene graphs provides better image generation for complex scenes.



Johnson et al. Image Generation from Scene Graphs, CVPR 2019

HYPE: Human eYe Perceptual Evaluations

hype.stanford.edu



Zhou, Gordon, Krishna et al. HYPE: Human eYe Perceptual Evaluations, NeurIPS 2019

Summary: GANs

Don’t work with an explicit density function
Take game-theoretic approach: learn to generate from training distribution through 2-player game

Pros:

- Beautiful, state-of-the-art samples!

Cons:

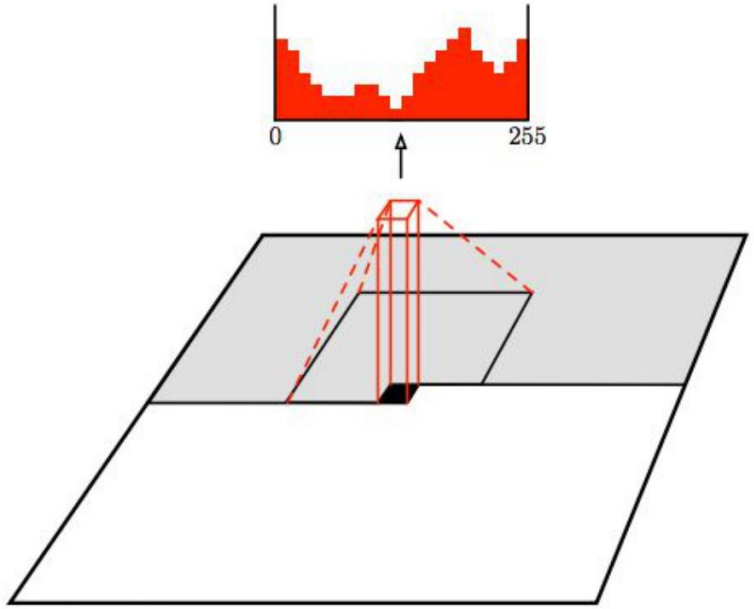
- Trickier / more unstable to train
- Can’t solve inference queries such as p(x), p(z|x)

Active areas of research:

- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications

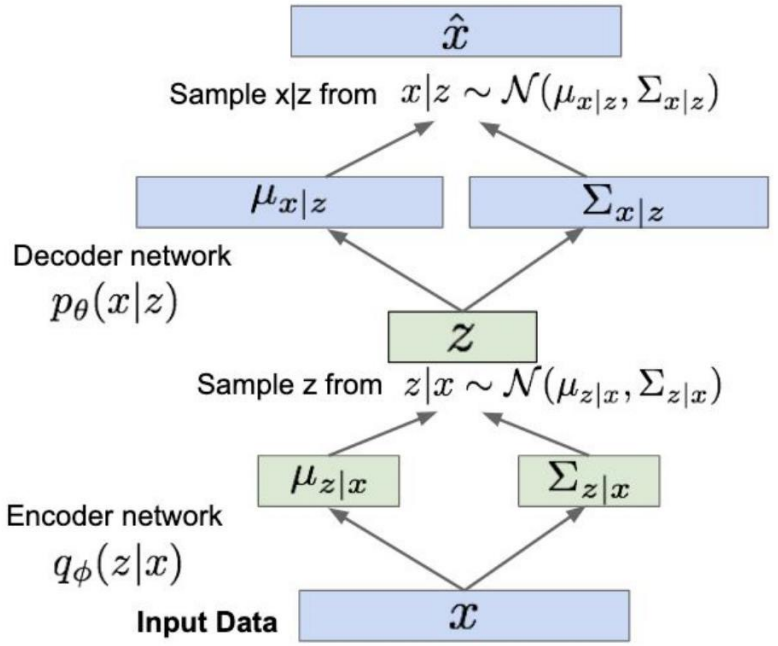
Summary

Autoregressive models:
PixelRNN, PixelCNN



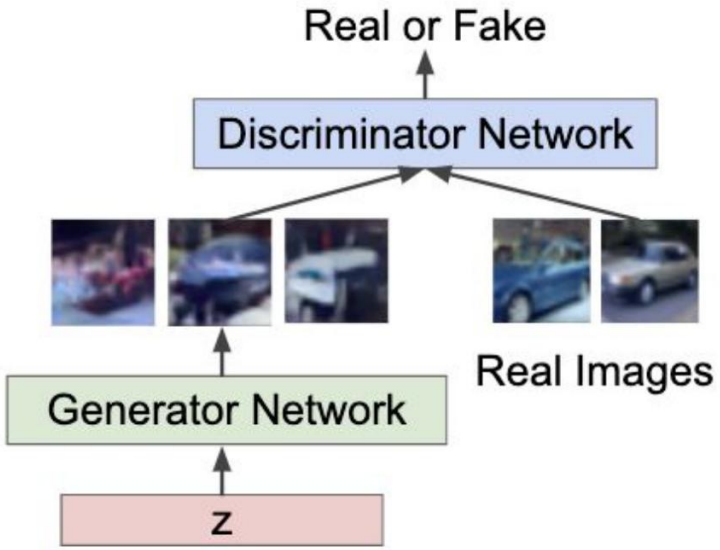
Van der Oord et al, “Conditional image generation with pixelCNN decoders”, NIPS 2016

Variational Autoencoders



Kingma and Welling, “Auto-encoding variational bayes”, ICLR 2013

Generative Adversarial Networks (GANs)



Goodfellow et al, “Generative Adversarial Nets”, NIPS 2014

Useful Resources on Generative Models

CS 236: Deep Generative Models (Stanford)

CS 294-158 Deep Unsupervised Learning (Berkeley)