max planck institut informatik

SIC Saarland Informatics Campus

High Level Computer Vision

Generative Models

@ July 19, 2023

Bernt Schiele

cms.sic.saarland/hlcvss23/

Max Planck Institute for Informatics & Saarland University,
Saarland Informatics Campus Saarbrücken

Overview of Today's Lecture

- Motivation
  - supervised vs unsupervised learning
  - discriminative vs generative models

- Generative Models
  - Autoregressive Models
  - Variational Autoencoder (VAE)
  - Generative Adversarial Network (GAN)

  - (Diffusion Models - https://www.youtube.com/watch?v=cS6JQpEY9cs)

Supervised vs. Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

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

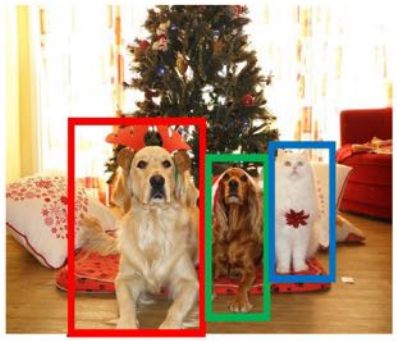
Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Classification

Cat

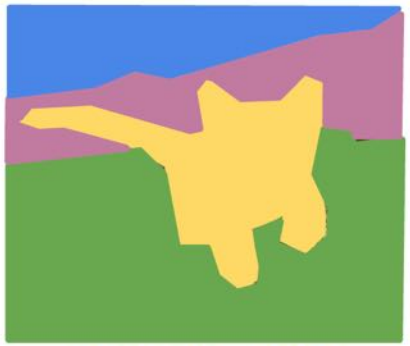
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Supervised Learning Examples



DOG, DOG, CAT

Object Detection



GRASS, CAT,
TREE, SKY

Semantic Segmentation

A cat sitting on a suitcase on the floor

Image captioning

Caption generated using neuraltalk2
Image is CC0 Public domain

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

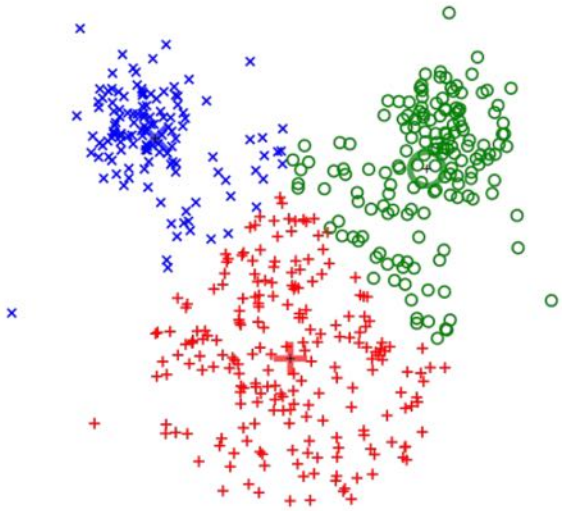
Supervised vs. Unsupervised Learning

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.



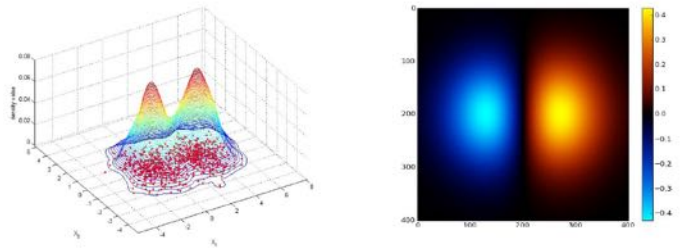
K-means clustering

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Unsupervised Learning Examples

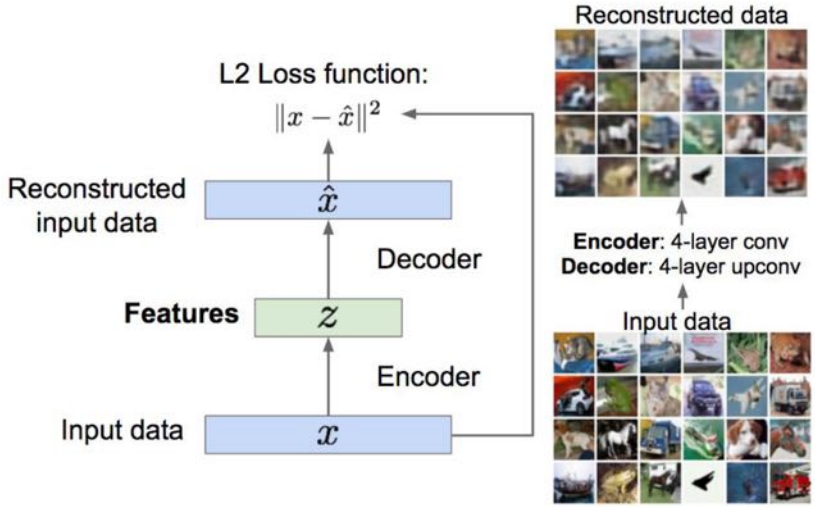
Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

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



Autoencoders
(Feature learning)

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Discriminative vs. Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Data: x

Label: y

Cat

Density Function

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

Density functions are **normalized**:

$$\int_X p(x)dx = 1$$

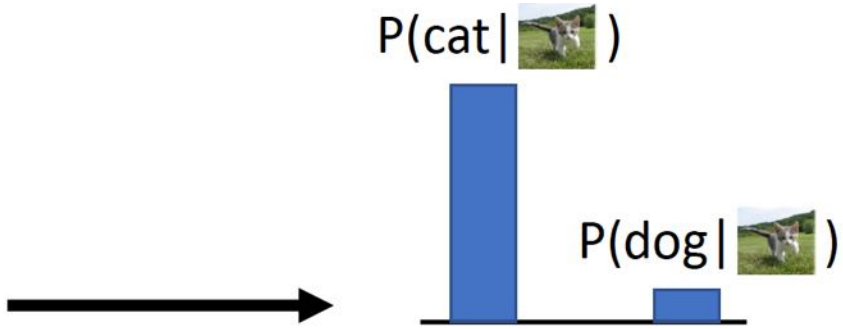
Different values of x **compete** for density

slide credit: Justin Johnson

Discriminative vs. Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Data: x



Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Density Function

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

Density functions are **normalized**:

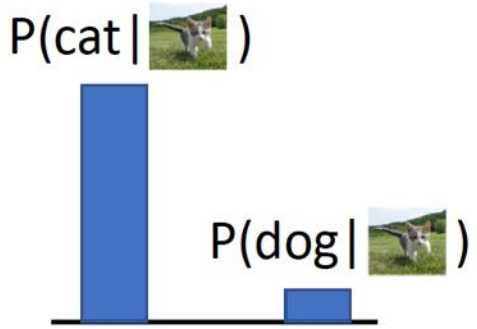
$$\int_X p(x)dx = 1$$

Different values of x **compete** for density

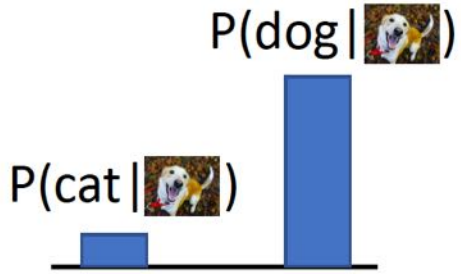
slide credit: Justin Johnson

Discriminative vs. Generative Models

Discriminative Model: Learn a probability distribution p(y|x)



Generative Model: Learn a probability distribution p(x)



Conditional Generative Model: Learn p(x|y)

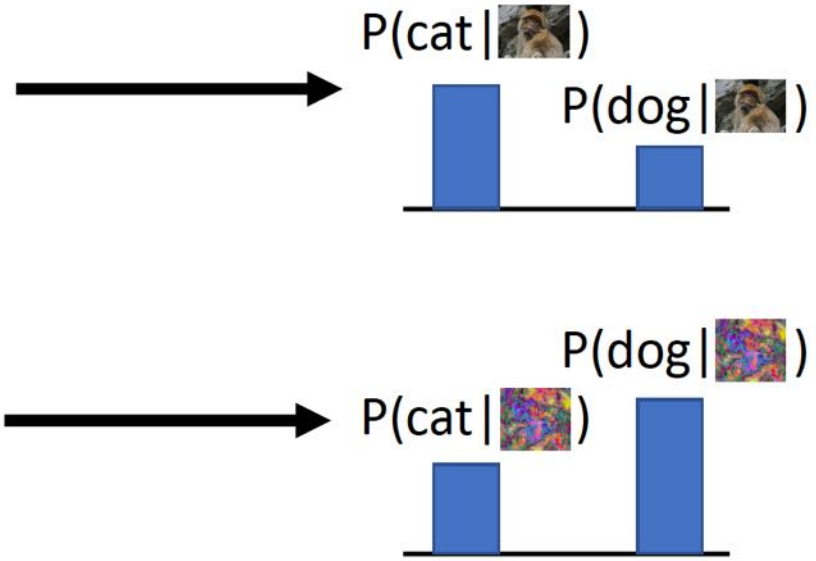
Discriminative model: the possible labels for each input "compete" for probability mass. But no competition between **images**

slide credit: Justin Johnson

Discriminative vs. Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)



Conditional Generative Model: Learn p(x|y)

Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

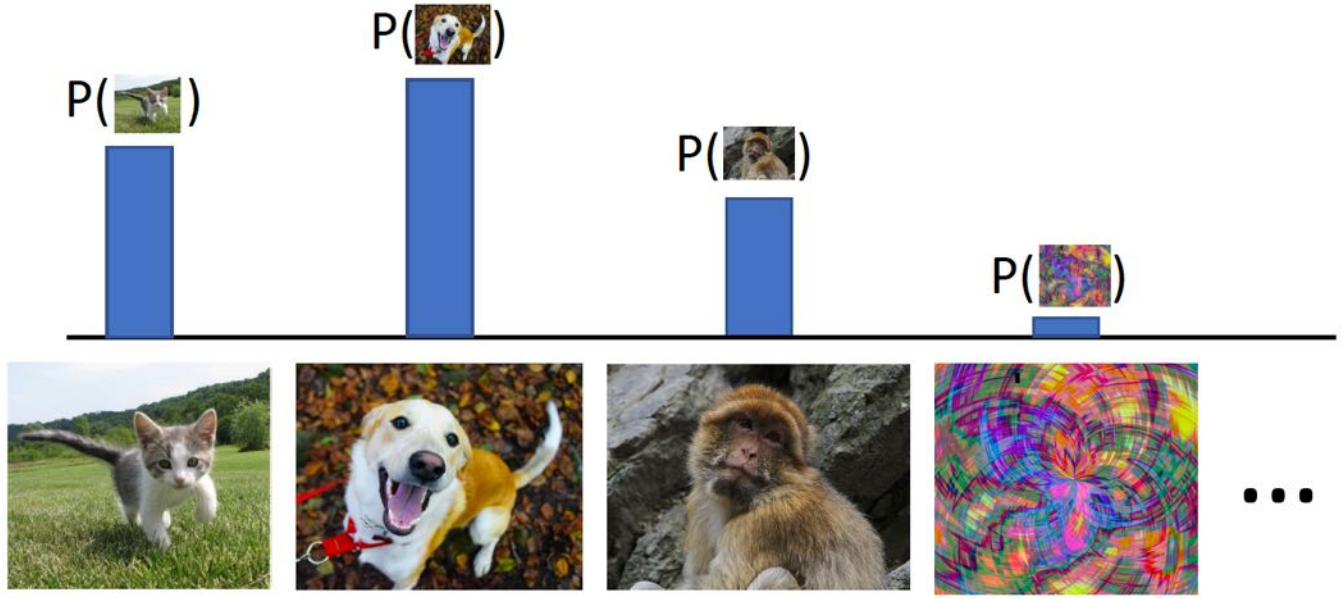
slide credit: Justin Johnson

Discriminative vs. Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Generative model: All possible images compete with each other for probability mass

Model can “reject” unreasonable inputs by assigning them small values

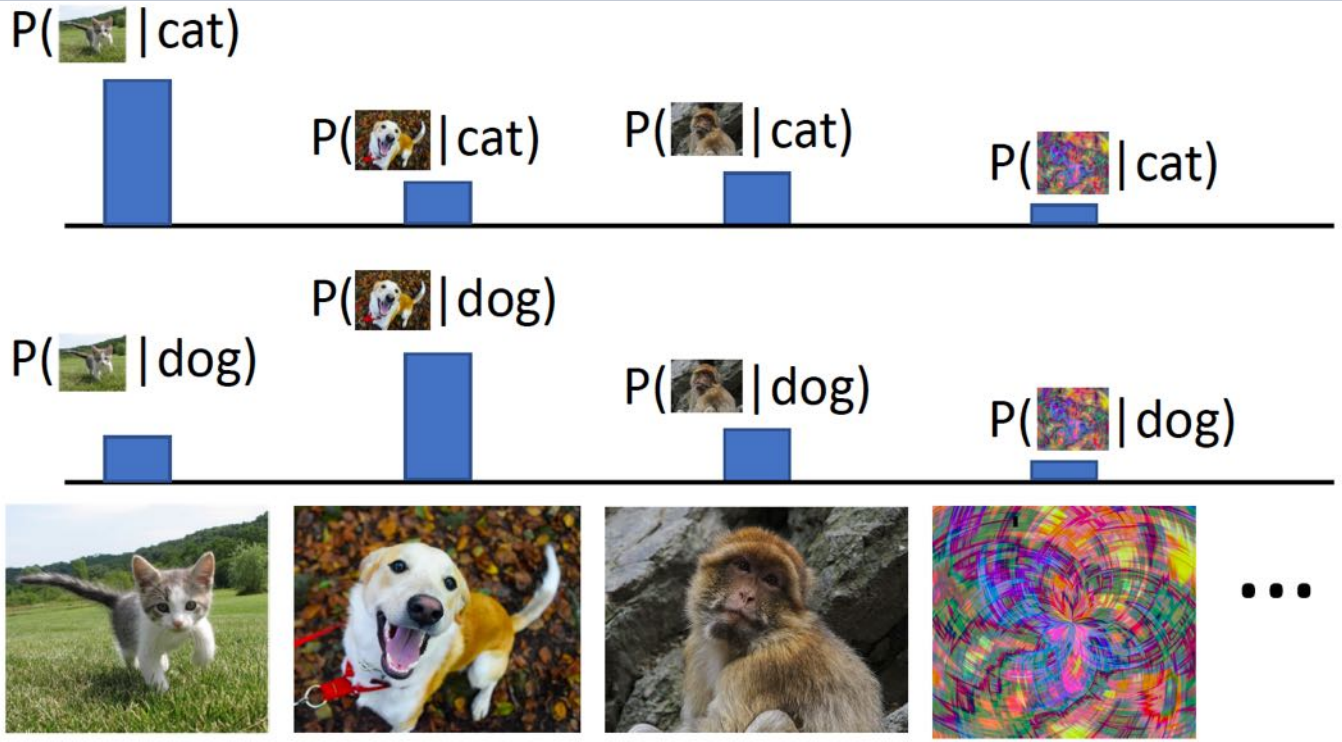
slide credit: Justin Johnson

Discriminative vs. Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Conditional Generative Model: Each possible label induces a competition among all images

slide credit: Justin Johnson

Discriminative vs. Generative Models

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Recall **Bayes' Rule:**

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

Conditional Generative Model

Discriminative Model

Prior over labels

(Unconditional) Generative Model

We can build a conditional generative model from other components!

slide credit: Justin Johnson

What can we do with these models?

Discriminative Model:
Learn a probability distribution p(y|x)

→ Assign labels to data
Feature learning (with labels)

Generative Model:
Learn a probability distribution p(x)

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

Conditional Generative Model: Learn p(x|y)

→ Assign labels, while rejecting outliers!
Generate new data conditioned on input labels

slide credit: Justin Johnson

Taxonomy of Generative Models

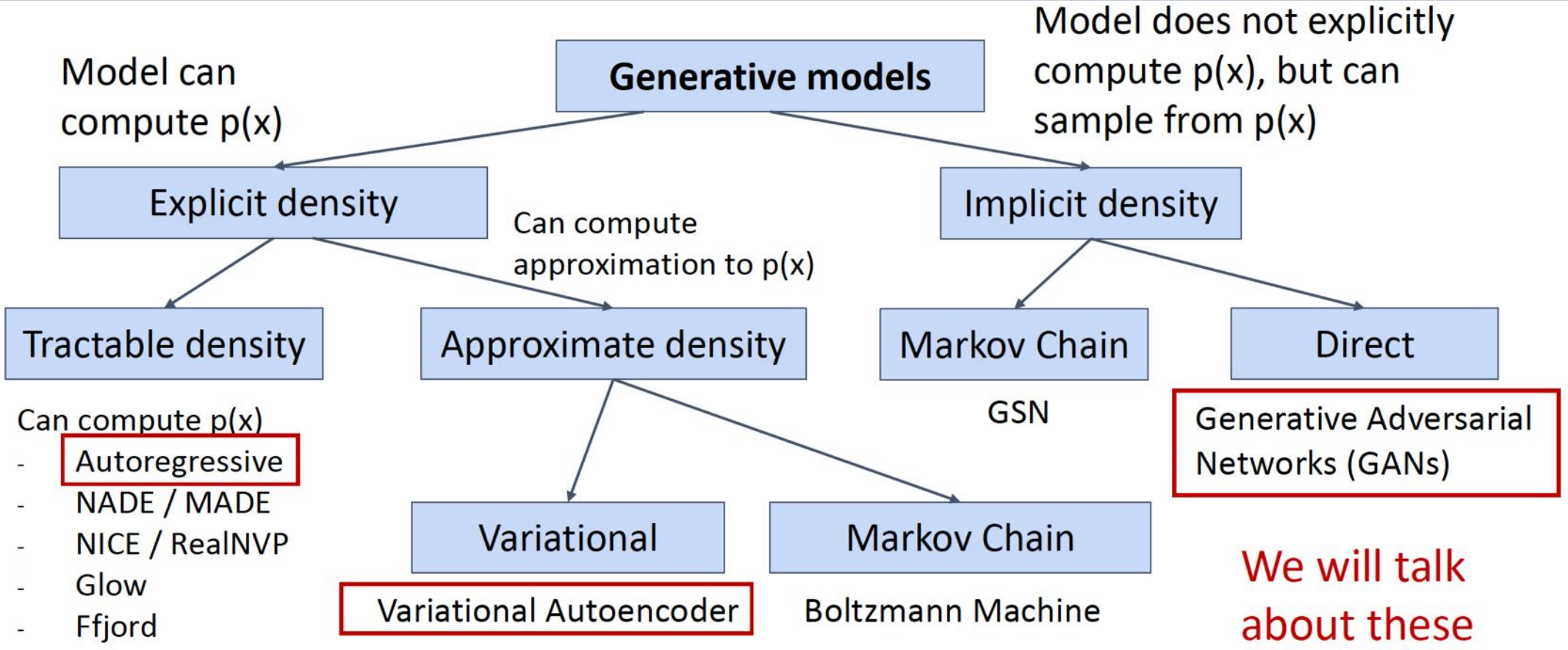


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.
slide credit: Justin Johnson

Autoregressive models

Explicit Density Estimation

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

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

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

Maximize probability of training data
(Maximum likelihood estimation)

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

Log trick to exchange product for sum

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

This will be our loss function!
Train with gradient descent

slide credit: Justin Johnson

Explicit Density: Autoregressive Models

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

Assume x consists of multiple subparts:

$$x = (x_1, x_2, x_3, \ldots, x_T)$$

Break down probability using the chain rule:

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

Probability of the next subpart given all the previous subparts

slide credit: Justin Johnson

Explicit Density: Autoregressive Models

Explicit density model

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

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

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Will need to define ordering of "previous pixels"

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

Then maximize likelihood of training data

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

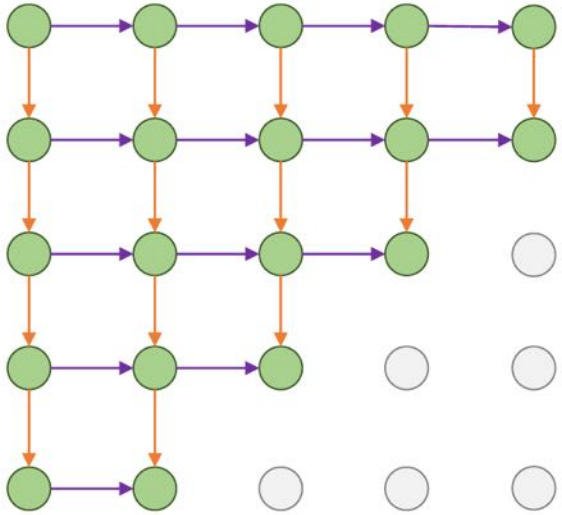
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



Van den Oord et al, “Pixel Recurrent Neural Networks”, ICML 2016

PixelRNN

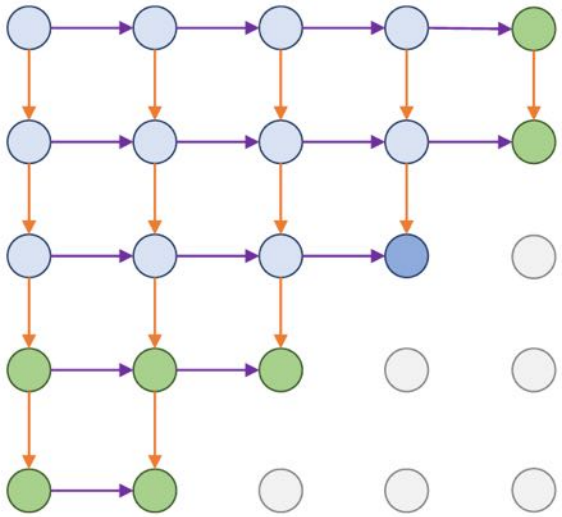
Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]

Each pixel depends **implicity** on all pixels above and to the left:



Van den Oord et al, "Pixel Recurrent Neural Networks", ICML 2016

slide credit: Justin Johnson

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

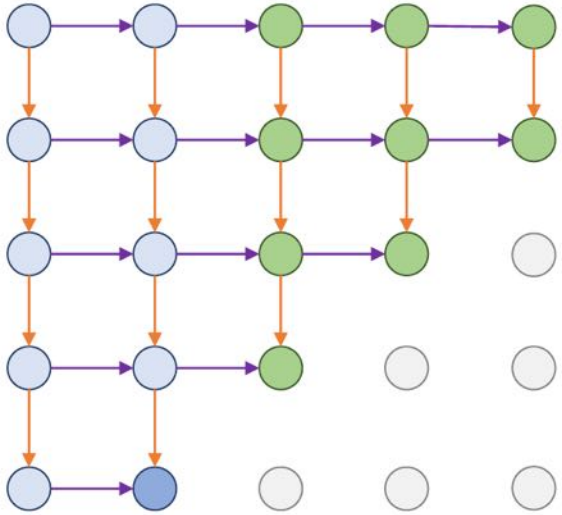
Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]

Each pixel depends **implicity** on all pixels above and to the left:

Problem: Very slow during both training and testing; N x N image requires 2N-1 sequential steps



Van den Oord et al, "Pixel Recurrent Neural Networks", ICML 2016

slide credit: Justin Johnson

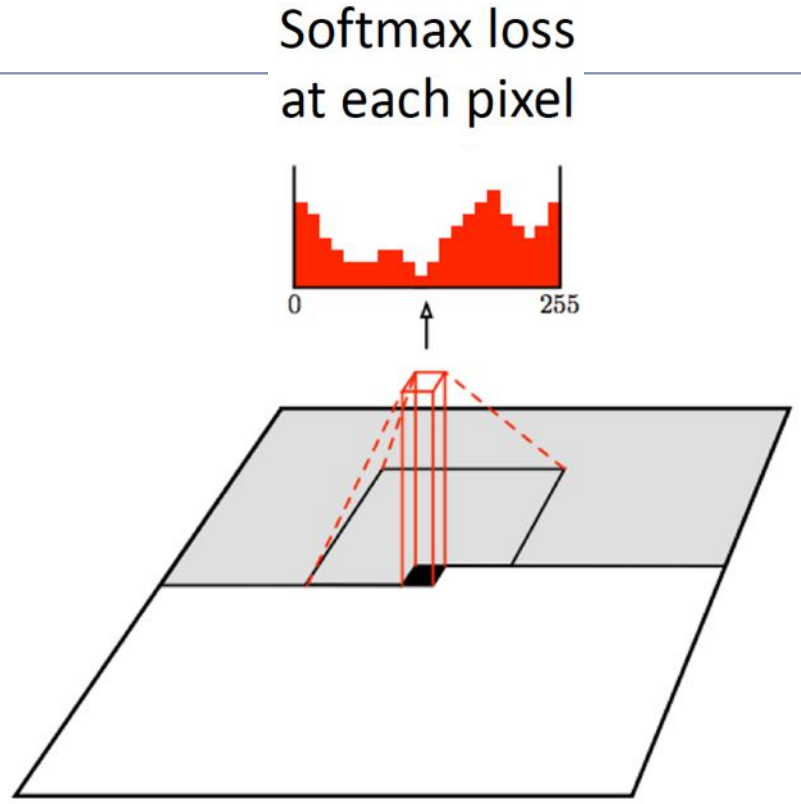
PixelCNN

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training: maximize likelihood of training images

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



Van den Oord et al, "Conditional Image Generation with PixelCNN Decoders", NeurIPS 2016

slide credit: Justin Johnson

PixelCNN

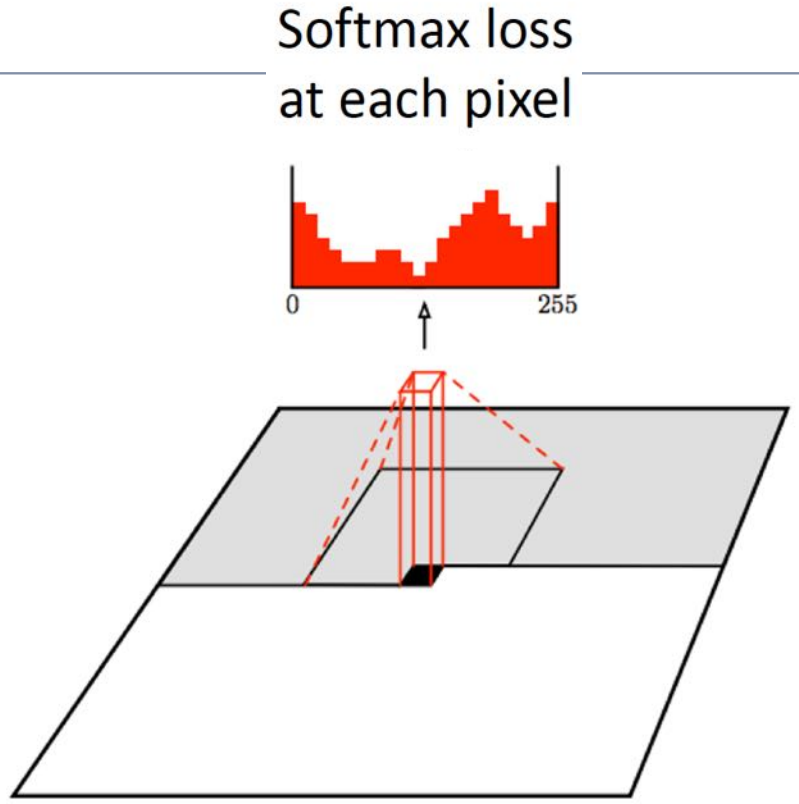
Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training: maximize likelihood of training images

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

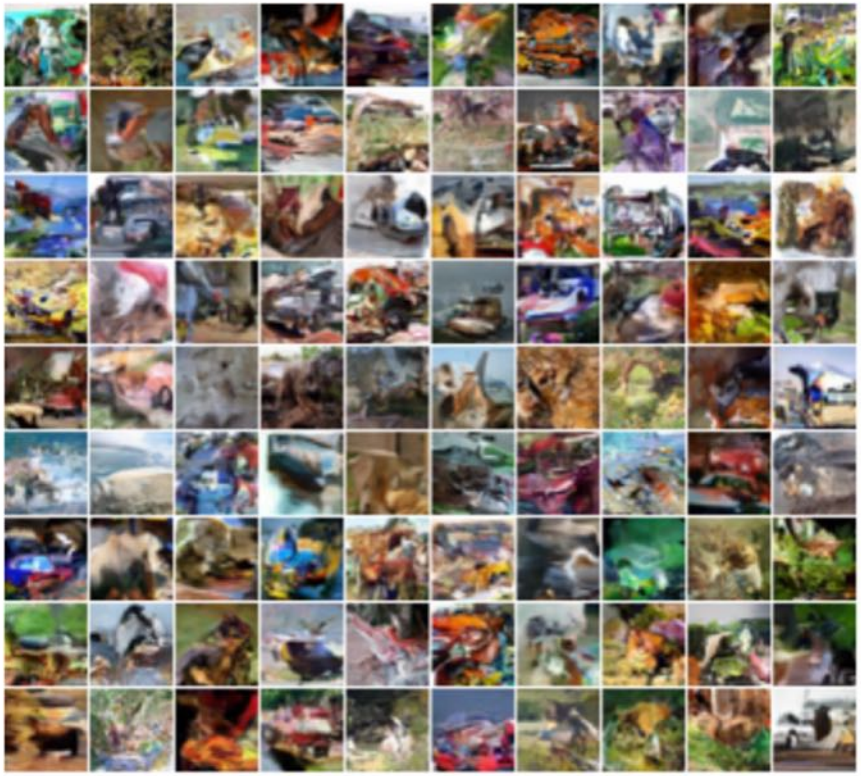
Generation must still proceed sequentially
=> still slow



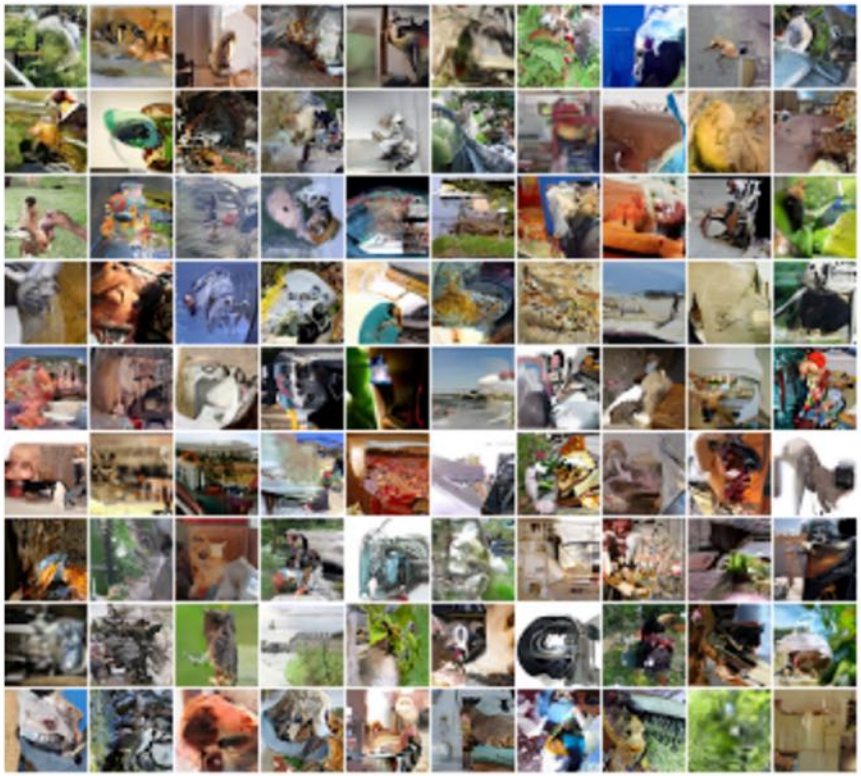
Van den Oord et al, "Conditional Image Generation with PixelCNN Decoders", NeurIPS 2016

slide credit: Justin Johnson

PixelRNN: Generated Samples



32x32 CIFAR-10



32x32 ImageNet

Van den Oord et al, “Pixel Recurrent Neural Networks”, ICML 2016

slide credit: Justin Johnson

Autoregressive Models: PixelRNN vs PixelCNN

Pros:

- Can explicitly compute likelihood p(x)
- Explicit likelihood of training data gives good evaluation metric
- Good samples

Con:

- Sequential generation => slow

Improving PixelCNN performance

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

See

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

slide credit: Justin Johnson

Variational Autoencoders

Variational Autoencoders

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

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

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

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

slide credit: Justin Johnson

Variational Autoencoders

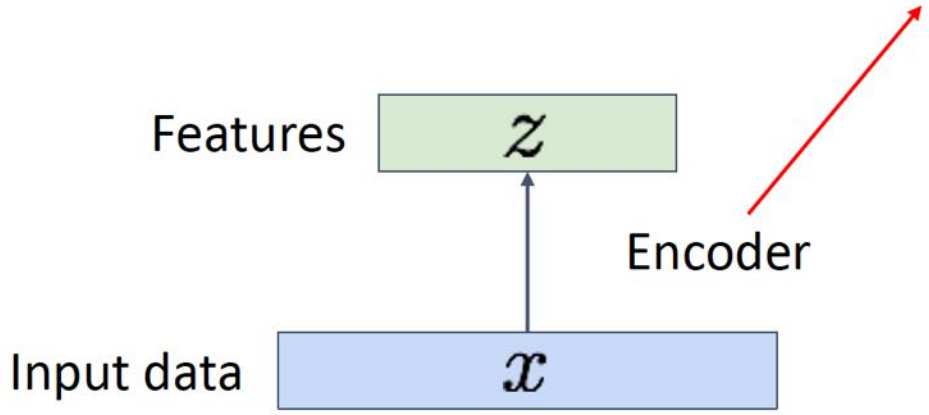
slide credit: Justin Johnson

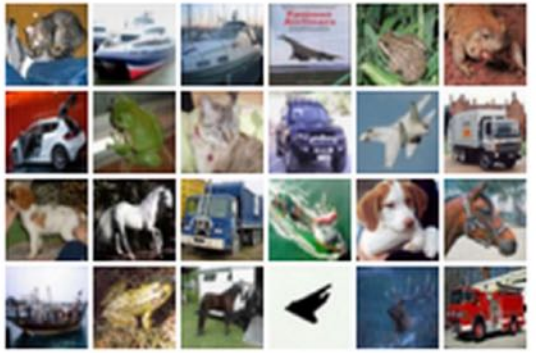
Regular (non-variational) Autoencoders

Unsupervised method for learning feature vectors from raw data x, without any labels

Features should extract useful information (maybe object identities, properties, scene type, etc) that we can use for downstream tasks

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN





Input Data

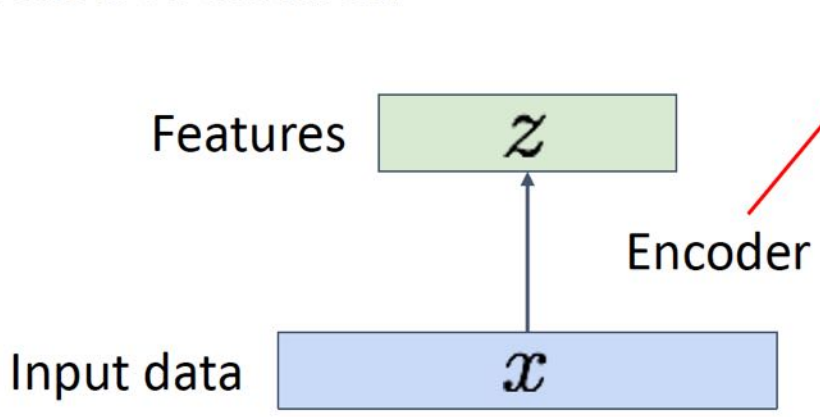
slide credit: Justin Johnson

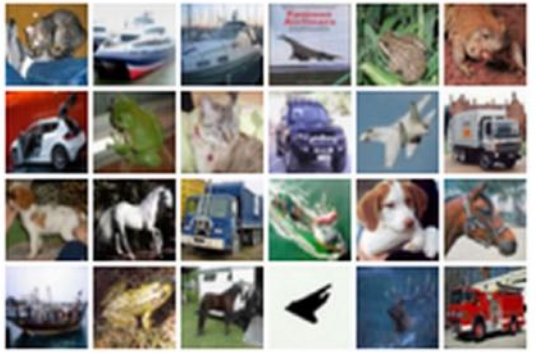
Regular (non-variational) Autoencoders

Problem: How can we learn this feature transform from raw data?

Features should extract useful information (maybe object identities, properties, scene type, etc) that we can use for downstream tasks
But we can't observe features!

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN





Input Data

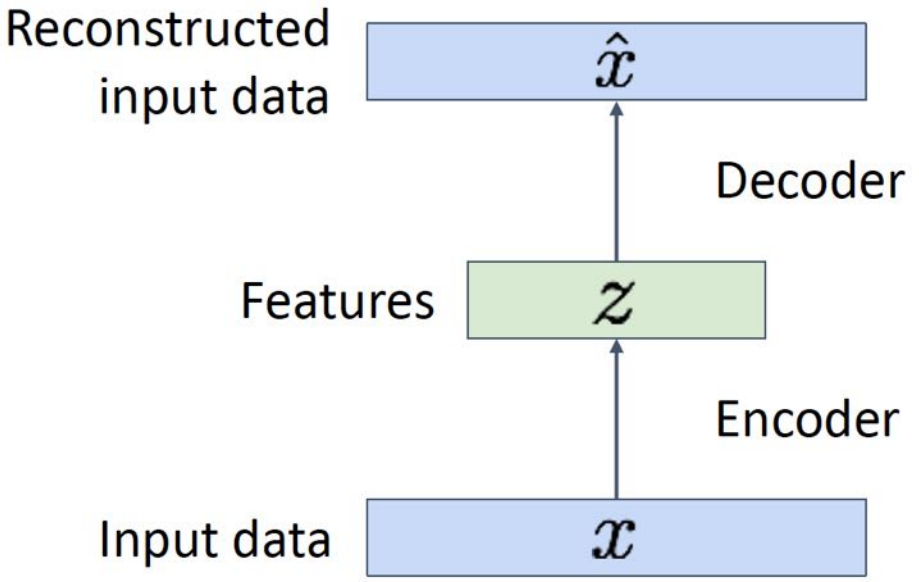
slide credit: Justin Johnson

Regular (non-variational) Autoencoders

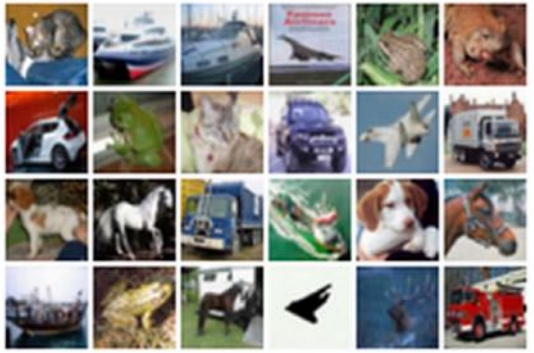
Problem: How can we learn this feature transform from raw data?

Idea: Use the features to reconstruct the input data with a **decoder**

"Autoencoding" = encoding itself



Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN (upconv)



Input Data

slide credit: Justin Johnson

(Regular, non-variational) Autoencoders

Loss: L2 distance between input and reconstructed data.

Does not use any labels! Just raw data!

Loss Function

$$\|\hat{x} - x\|_2^2$$

Reconstructed input data

$\hat{x}$

Decoder

Features need to be **lower dimensional** than the data

Features

z

Encoder

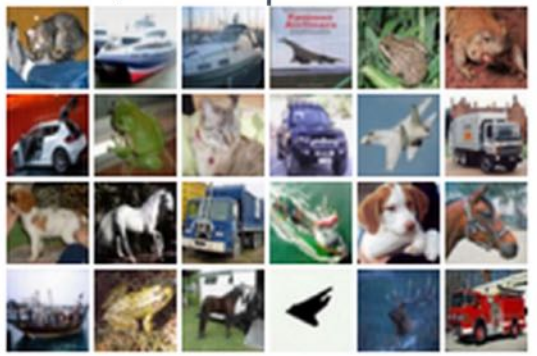
Input data

x

Reconstructed data

Decoder:
4 tconv layers

Encoder:
4 conv layers

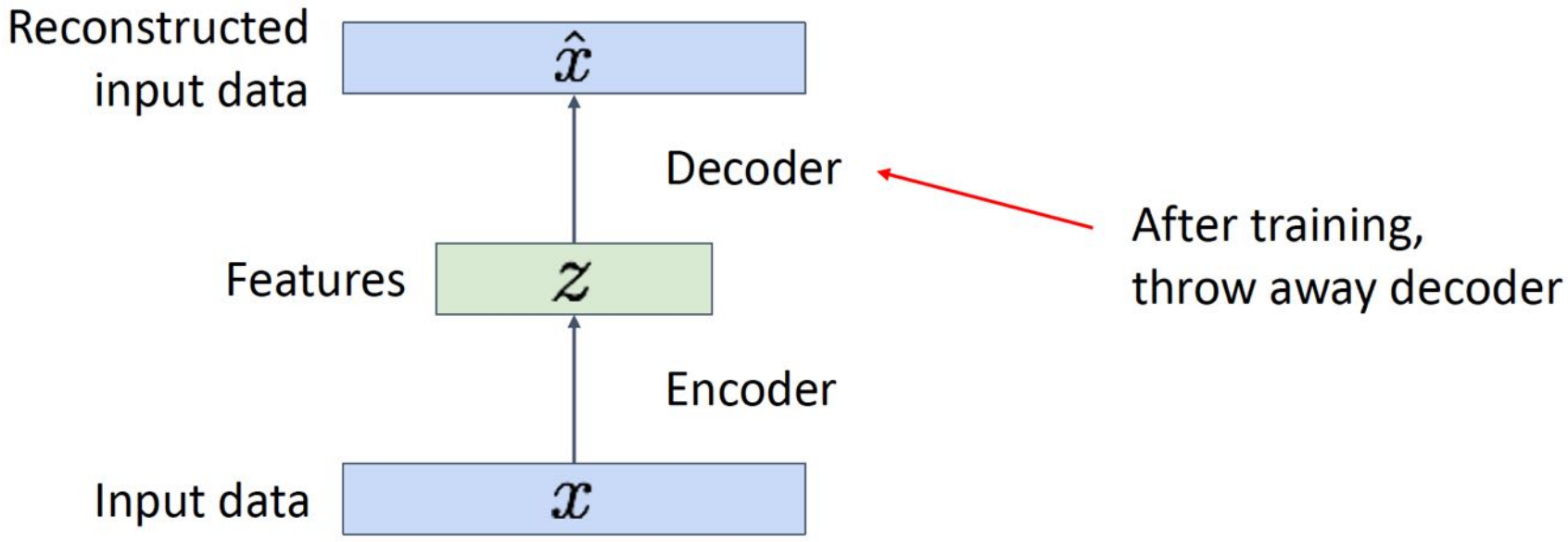


Input Data

slide credit: Justin Johnson

Regular (non-variational) Autoencoders

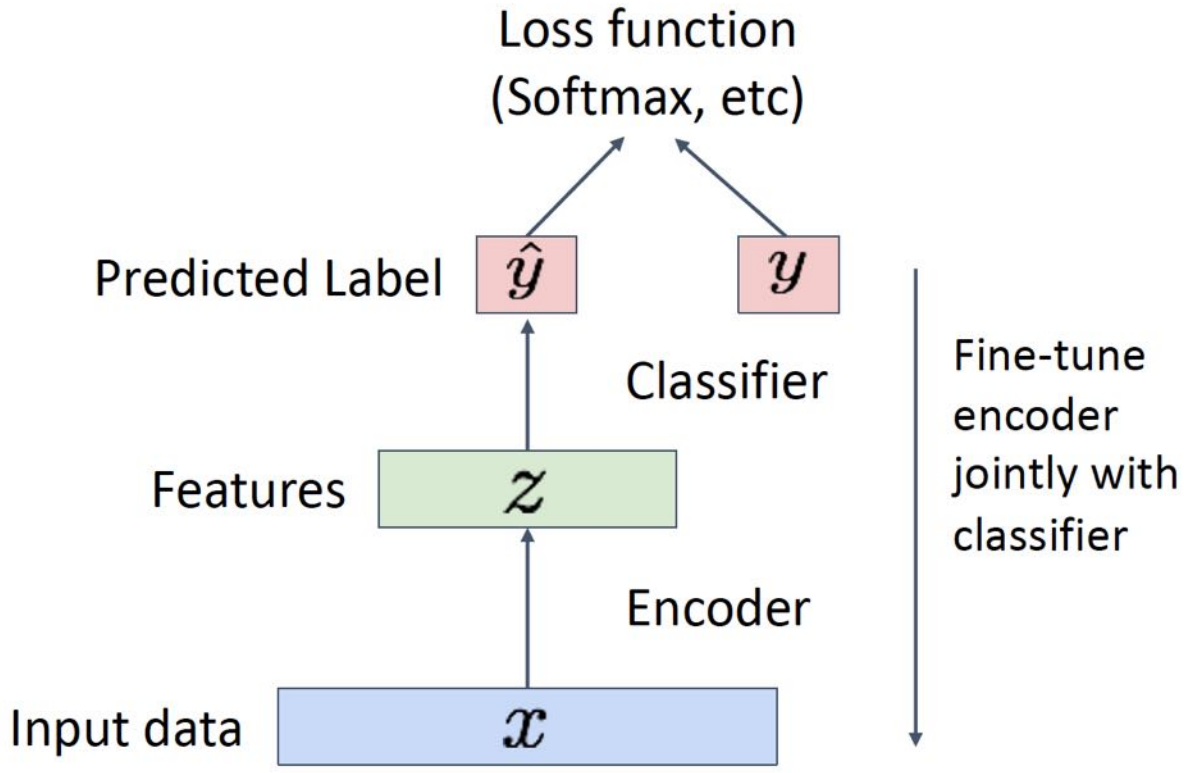
After training, **throw away decoder** and use encoder for a downstream task



slide credit: Justin Johnson

Regular (non-variational) Autoencoders

After training, **throw away decoder** and use encoder for a downstream task



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

Train for final task (sometimes with small data)

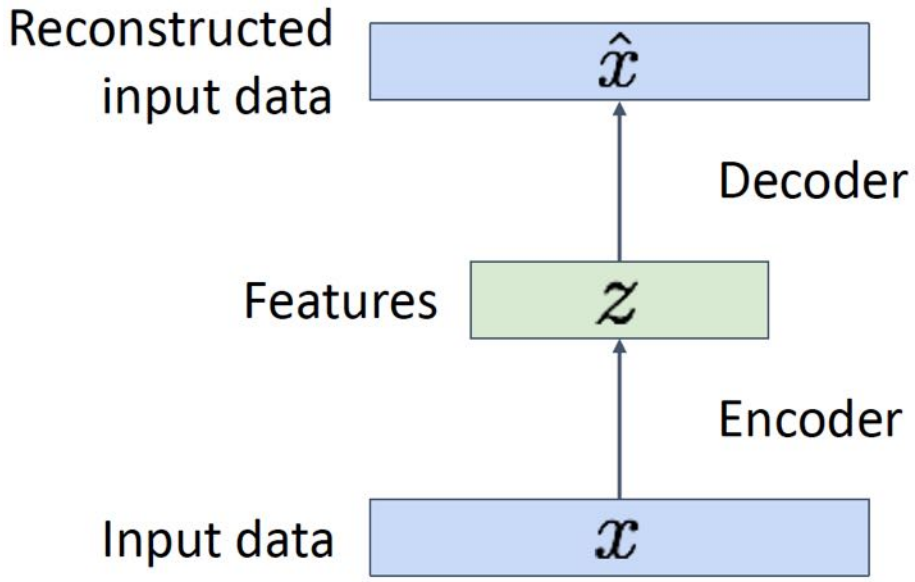
slide credit: Justin Johnson

Regular (non-variational) Autoencoders

Autoencoders learn **latent features** for data without any labels!

Can use features to initialize a **supervised** model

Not probabilistic: No way to sample new data from learned model



slide credit: Justin Johnson

Variational Autoencoders

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

Variational Autoencoders

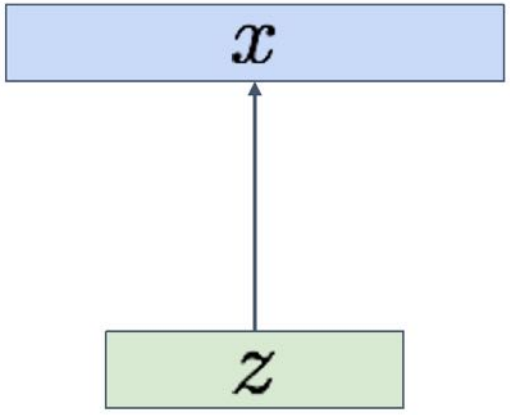
Probabilistic spin on autoencoders:

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

After training, sample new data like this:

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

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



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

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

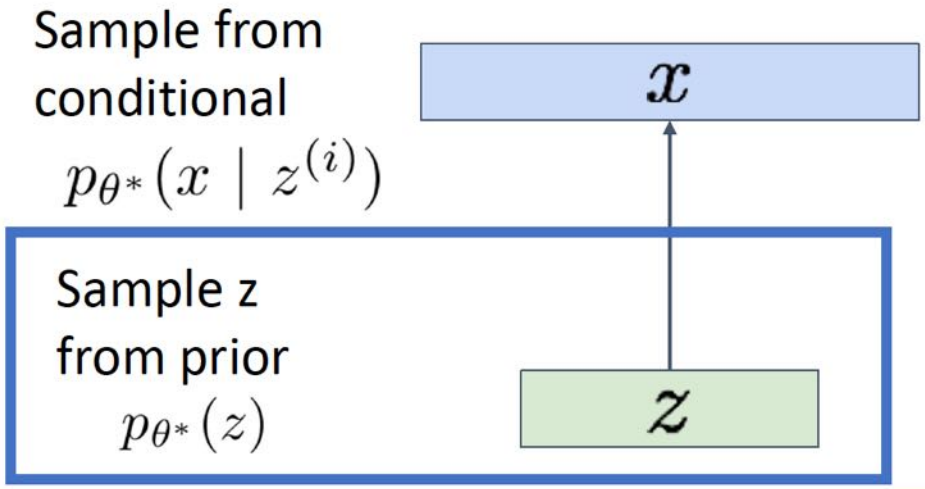
slide credit: Justin Johnson

Variational Autoencoders

Probabilistic spin on autoencoders:

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

After training, sample new data like this:



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

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

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

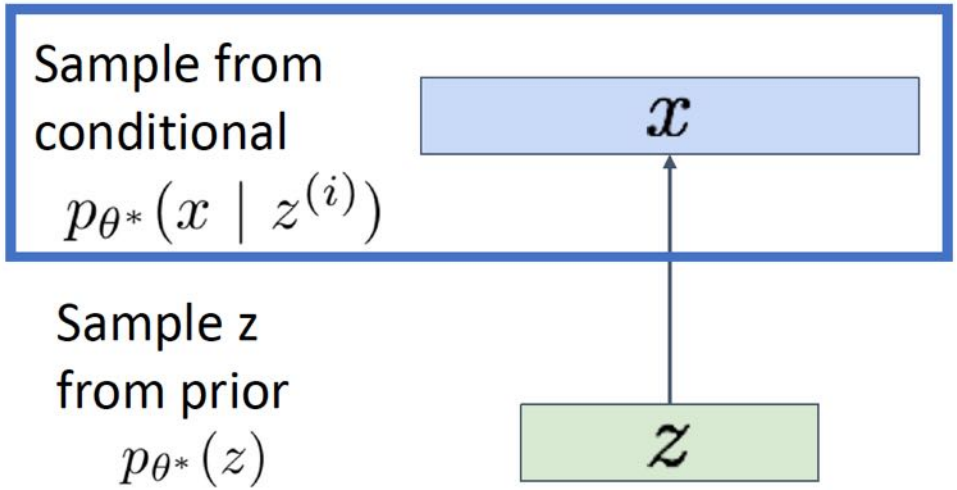
slide credit: Justin Johnson

Variational Autoencoders

Probabilistic spin on autoencoders:

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

After training, sample new data like this:



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

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

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

Represent p(x|z) with a neural network (Similar to **decoder** from autencoder)

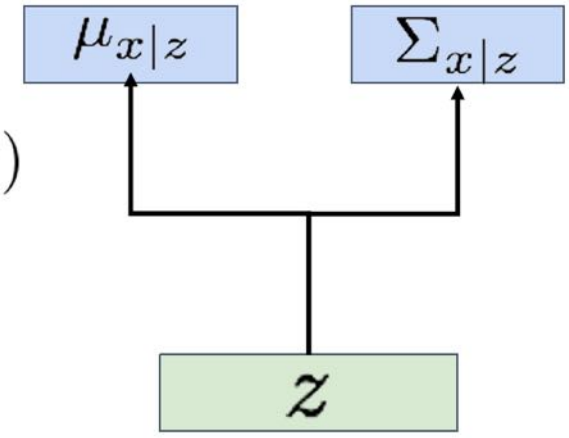
slide credit: Justin Johnson

Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

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



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

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

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

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

Represent p(x|z) with a neural network (Similar to **decoder** from autencoder)

slide credit: Justin Johnson

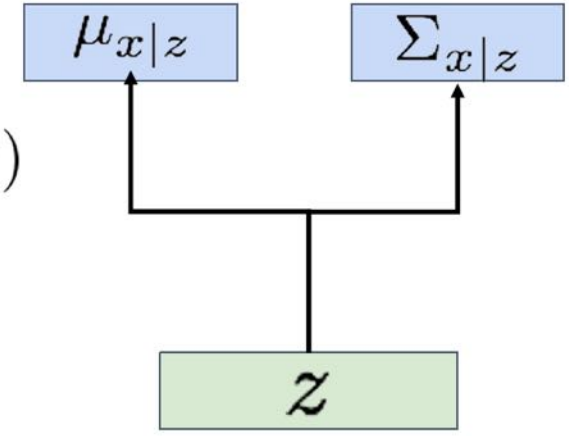
Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

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

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

If we could observe the z for each x, then could train a *conditional generative model* p(x|z)

slide credit: Justin Johnson

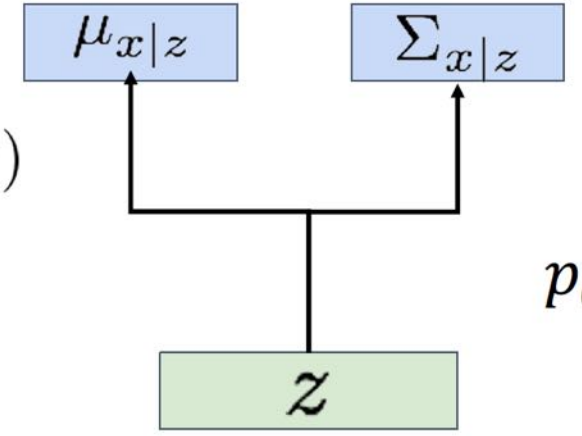
Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

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

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

We don't observe z, so need to marginalize:

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

Ok, can compute this with decoder network

slide credit: Justin Johnson

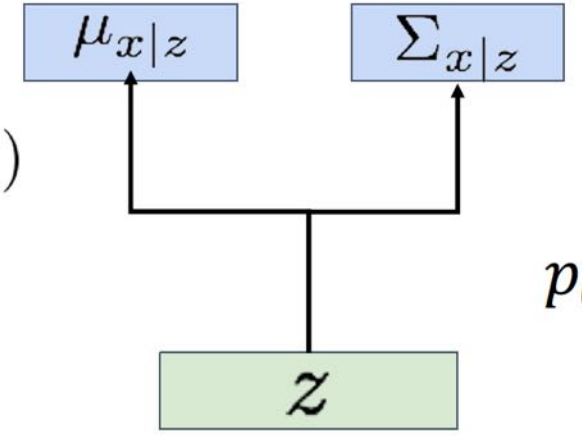
Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

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

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

We don't observe z, so need to marginalize:

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

Ok, we assumed Gaussian prior for z

slide credit: Justin Johnson

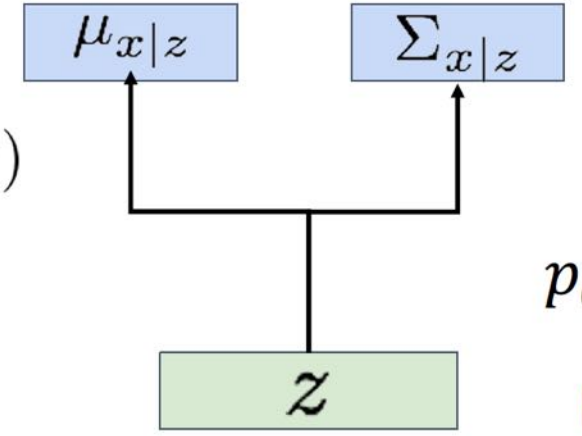
Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

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

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

We don't observe z, so need to marginalize:

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

Problem: Impossible to integrate over all z!

slide credit: Justin Johnson

Variational Autoencoders

Recall $p(x, z) = p(x \mid z)p(z) = p(z \mid x)p(x)$

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

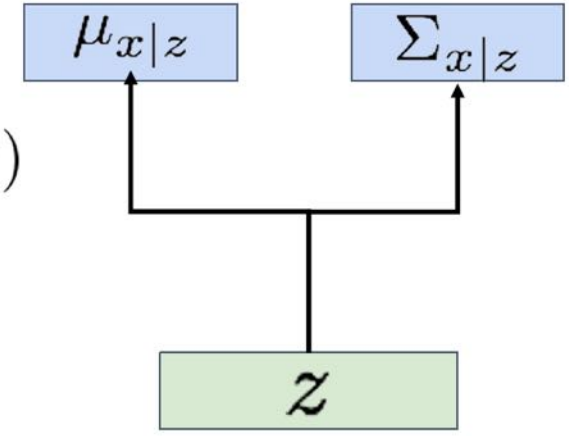
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

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

How to train this model?

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

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



Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

Ok, compute with decoder network

slide credit: Justin Johnson

Variational Autoencoders

Recall $p(x, z) = p(x \mid z)p(z) = p(z \mid x)p(x)$

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

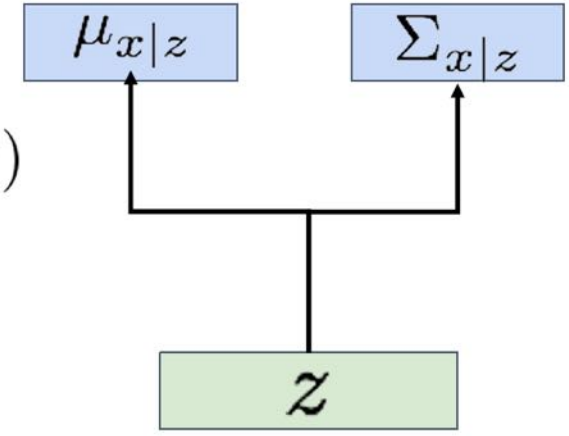
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

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

How to train this model?

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

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



Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

Ok, we assumed Gaussian prior

slide credit: Justin Johnson

Variational Autoencoders

Recall $p(x, z) = p(x \mid z)p(z) = p(z \mid x)p(x)$

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

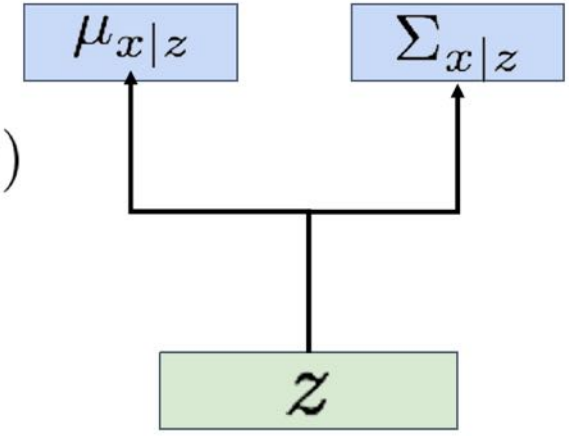
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

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

How to train this model?

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

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



Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

Problem: No way to compute this!

slide credit: Justin Johnson

Variational Autoencoders

Recall $p(x, z) = p(x \mid z)p(z) = p(z \mid x)p(x)$

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

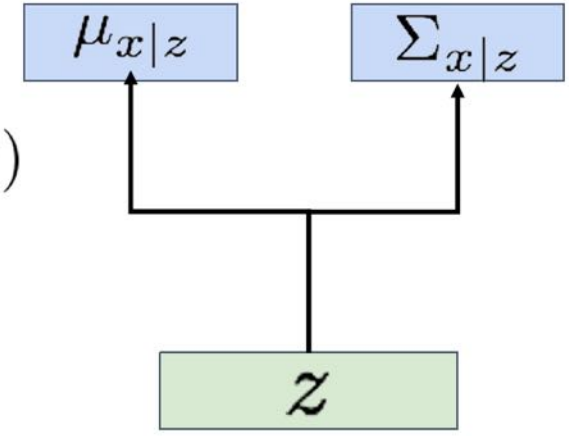
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

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

How to train this model?

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

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



Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

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

slide credit: Justin Johnson

Variational Autoencoders

Recall $p(x, z) = p(x \mid z)p(z) = p(z \mid x)p(x)$

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

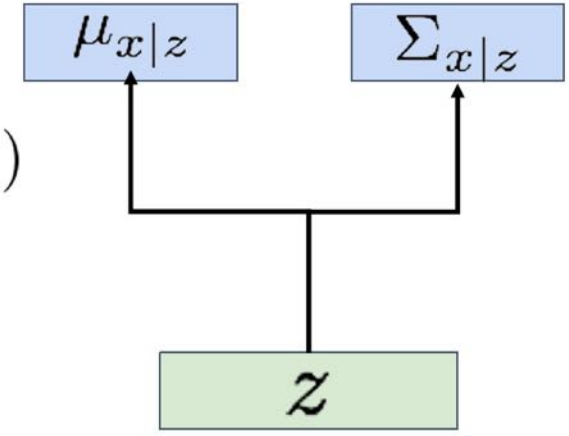
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\sum_{x|z}$

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

How to train this model?

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

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



Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

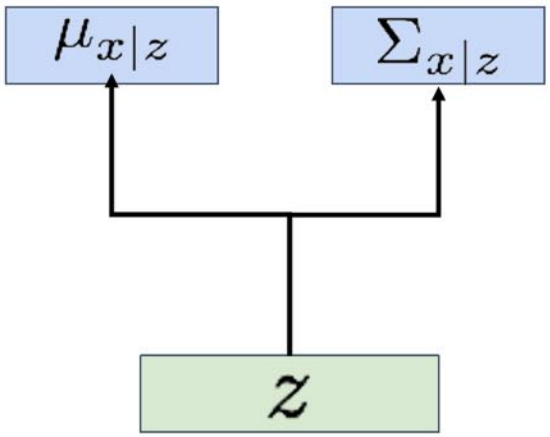
Use **encoder** to compute $q_\phi(z \mid x) \approx p_\theta(z \mid x)$

slide credit: Justin Johnson

Variational Autoencoders

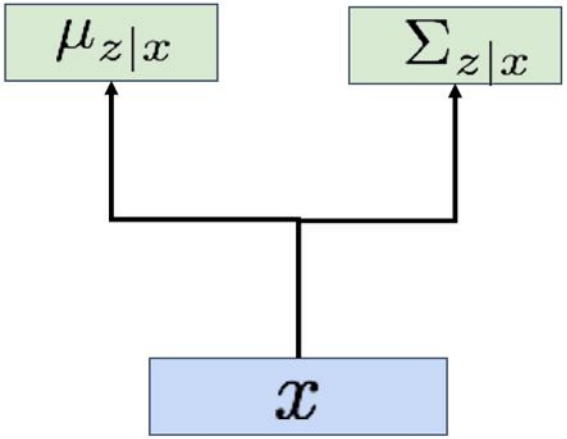
Decoder network inputs latent code z, gives distribution over data x

$$p_\theta(x \mid z) = N(\mu_{x|z}, \Sigma_{x|z})$$



Encoder network inputs data x, gives distribution over latent codes z

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



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

then we can approximate

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

Idea: Jointly train both encoder and decoder

slide credit: Justin Johnson

Variational Autoencoders

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

Bayes' Rule

slide credit: Justin Johnson

Variational Autoencoders

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

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

slide credit: Justin Johnson

Variational Autoencoders

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

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

Split up using rules for logarithms

slide credit: Justin Johnson

Variational Autoencoders

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

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

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

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

slide credit: Justin Johnson

Variational Autoencoders

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

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

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

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

slide credit: Justin Johnson

Variational Autoencoders

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

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

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

Data reconstruction

slide credit: Justin Johnson

Variational Autoencoders

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

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

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

KL divergence between prior, and samples from the encoder network

slide credit: Justin Johnson

Variational Autoencoders

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

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

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

KL divergence between encoder and posterior of decoder

slide credit: Justin Johnson

Variational Autoencoders

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

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

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

KL is >= 0, so dropping this term gives a **lower bound** on the data likelihood:

slide credit: Justin Johnson

Variational Autoencoders

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

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

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

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

slide credit: Justin Johnson

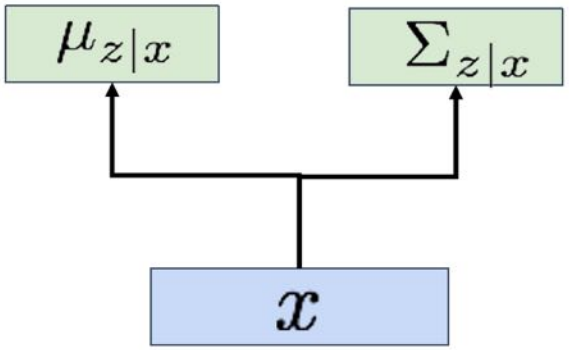
Variational Autoencoders

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

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

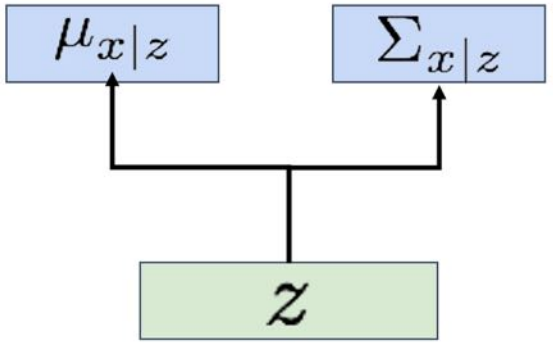
Encoder Network

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



Decoder Network

$$p_\theta(x \mid z) = N(\mu_{x|z}, \Sigma_{x|z})$$



slide credit: Justin Johnson

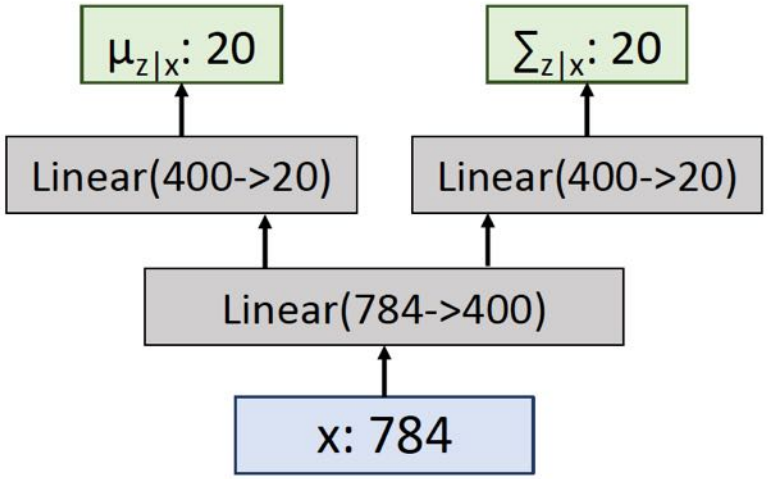
Example: Fully-Connected VAE

x: 28x28 image, flattened to 784-dim vector

z: 20-dim vector

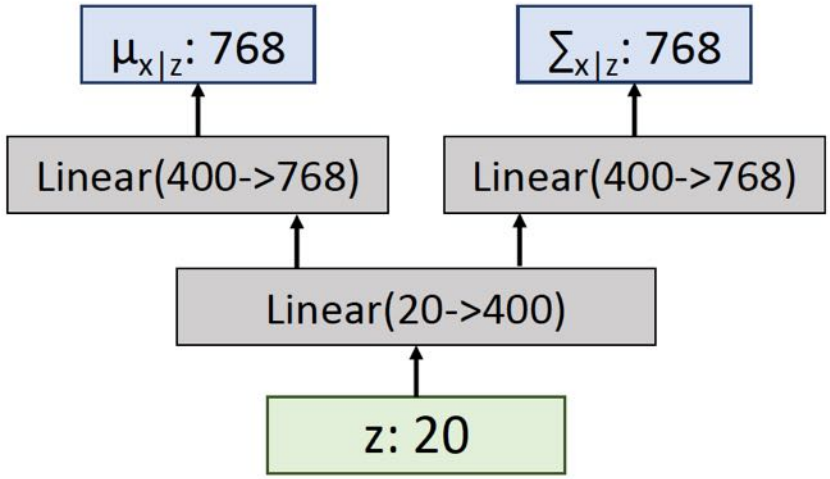
Encoder Network

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



Decoder Network

$$p_\theta(x \mid z) = N(\mu_{x|z}, \Sigma_{x|z})$$



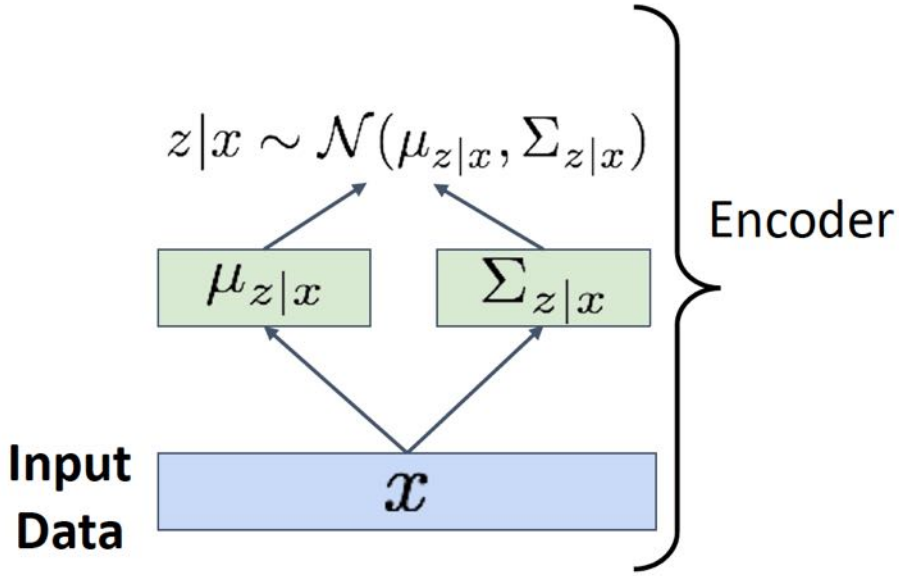
slide credit: Justin Johnson

Variational Autoencoders

Train by maximizing the
variational lower bound

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

1. Run input data through **encoder** to get a distribution over latent codes



slide credit: Justin Johnson

Variational Autoencoders

Train by maximizing the **variational lower bound**

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

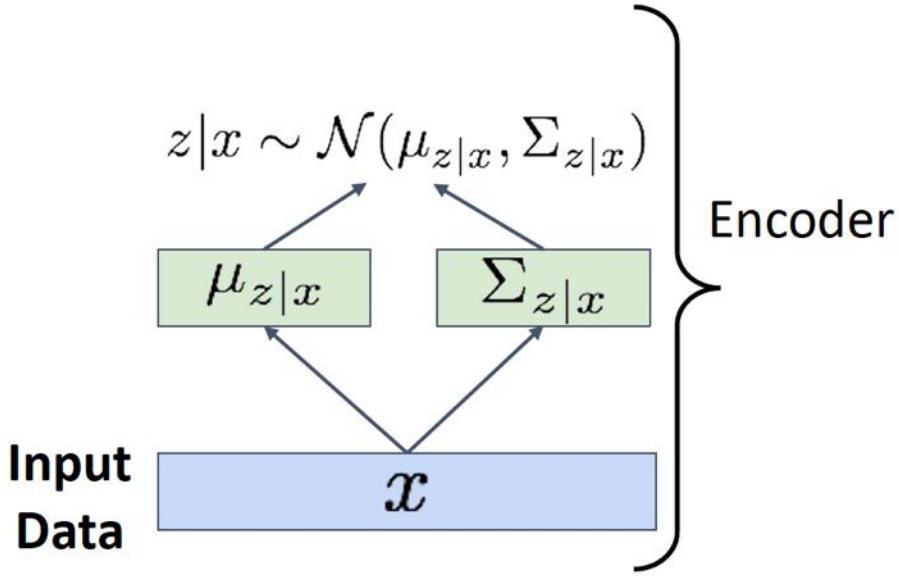
Closed form solution when q_ϕ is diagonal Gaussian and p is unit Gaussian! (Assume z has dimension J)

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior p(z)!**

$$-D_{KL}\left(q_\phi(z|x), p(z)\right) = \int_Z q_\phi(z|x)\log\frac{p(z)}{q_\phi(z|x)}dz$$

$$= \int_Z N\left(z;\mu_{z|x},\Sigma_{z|x}\right)\log\frac{N(z;0,I)}{N\left(z;\mu_{z|x},\Sigma_{z|x}\right)}dz$$

$$= \frac{1}{2}\sum_{j=1}^{J}\left(1+\log\left(\left(\Sigma_{z|x}\right)_j^2\right)-\left(\mu_{z|x}\right)_j^2-\left(\Sigma_{z|x}\right)_j^2\right)$$



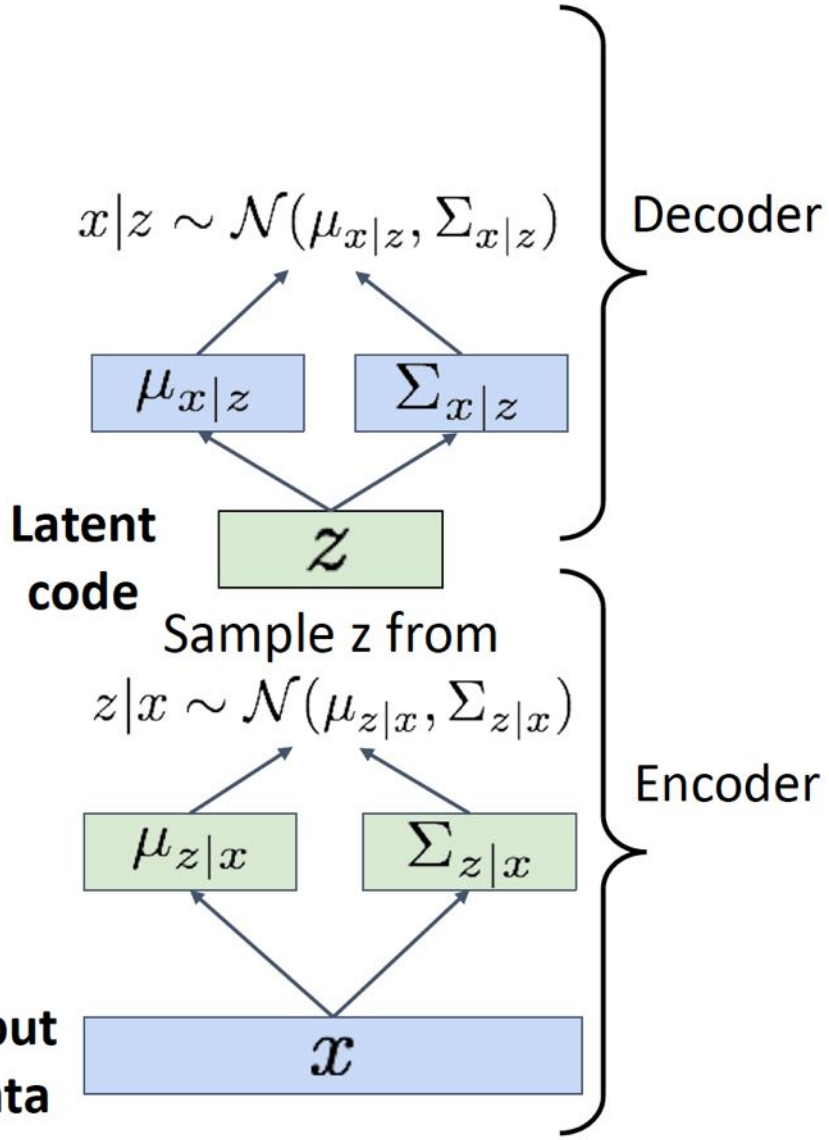
slide credit: Justin Johnson

Variational Autoencoders

Train by maximizing the **variational lower bound**

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

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior p(z)!**
3. Sample code z from encoder output
4. Run sampled code through **decoder** to get a distribution over data samples



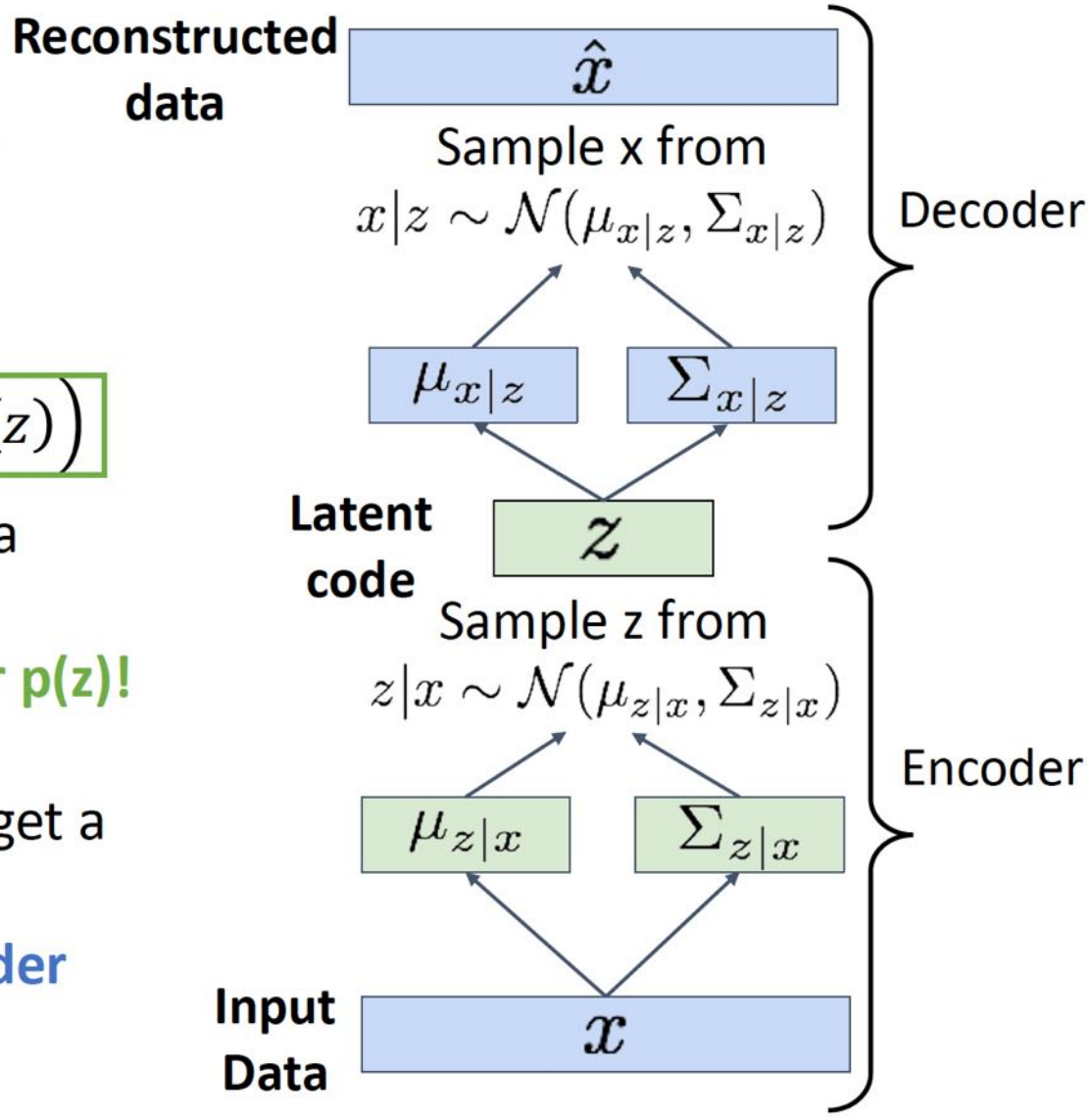
slide credit: Justin Johnson

Variational Autoencoders

Train by maximizing the **variational lower bound**

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

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior p(z)!**
3. Sample code z from encoder output
4. Run sampled code through **decoder** to get a distribution over data samples
5. **Original input data should be likely under the distribution output from (4)!**
6. Can sample a reconstruction from (4)

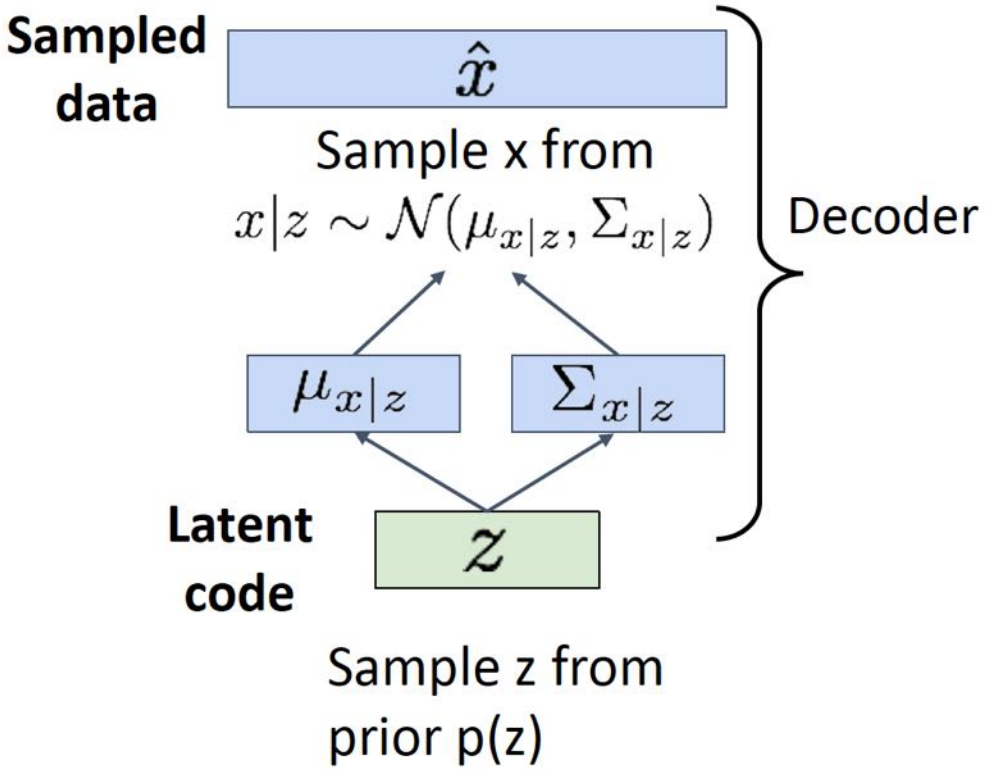


slide credit: Justin Johnson

Variational Autoencoders: Generating Data

After training we can generate new data!

1. Sample z from prior p(z)
2. Run sampled z through decoder to get distribution over data x
3. Sample from distribution in (2) to generate data

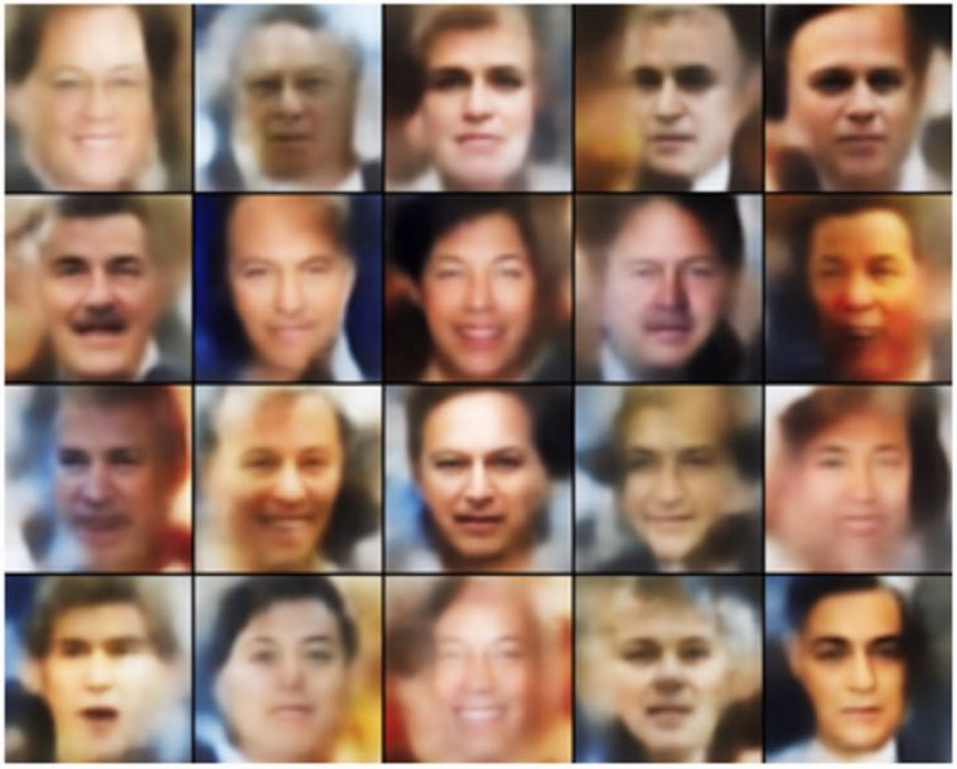


slide credit: Justin Johnson

Variational Autoencoders: Generating Data

32x32 CIFAR-10

Labeled Faces in the Wild



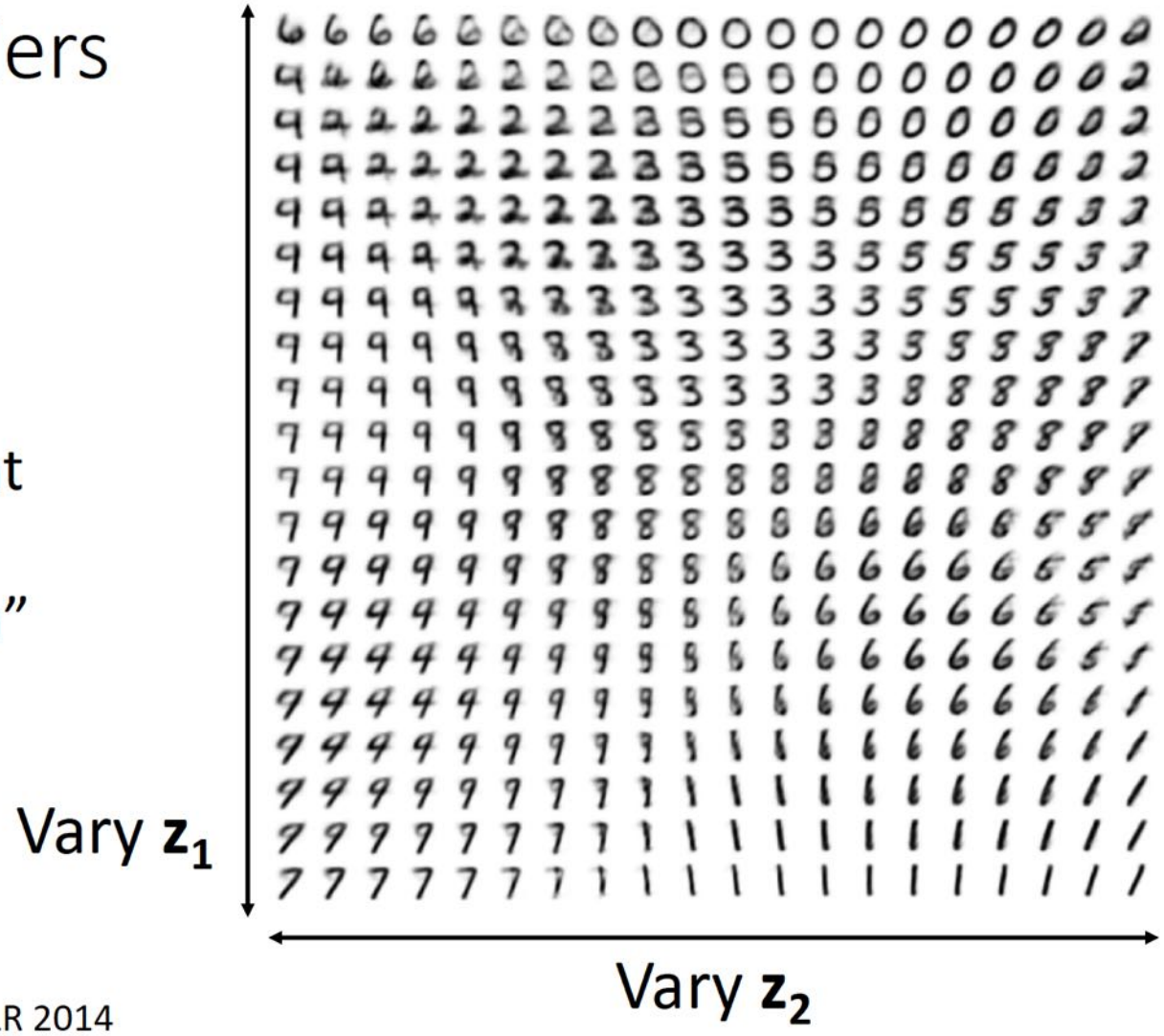
Figures from (L) Dirk Kingma et al. 2016; (R) Anders Larsen et al. 2017.

slide credit: Justin Johnson

Variational Autoencoders

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

"Disentangling factors of variation"



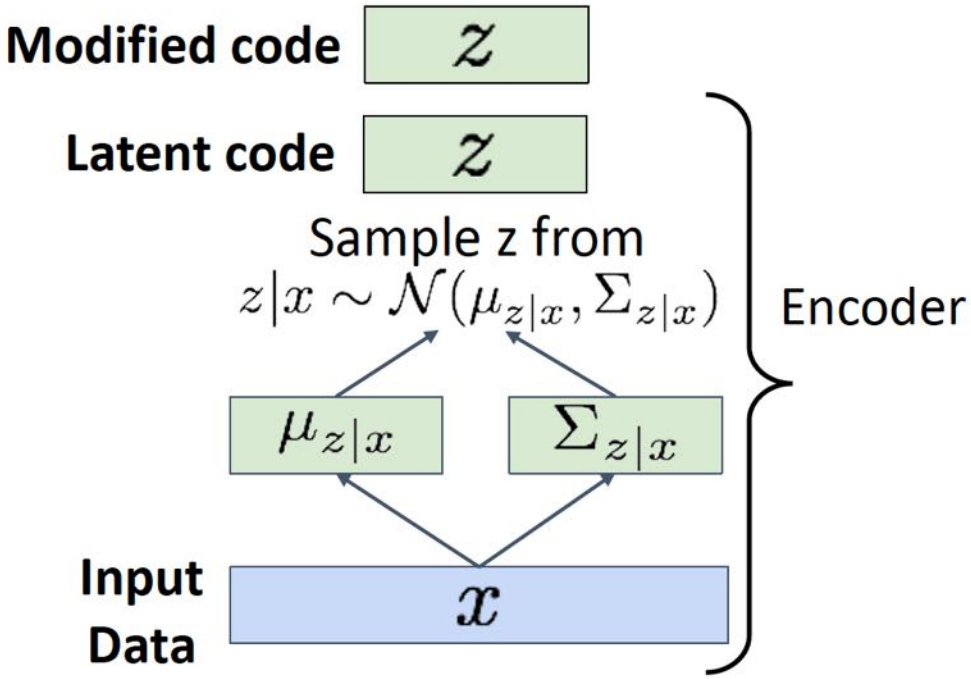
Kingma and Welling, Auto-Encoding Variational Beyes, ICLR 2014

slide credit: Justin Johnson

Variational Autoencoders

After training we can **edit images**

1. Run input data through **encoder** to get a distribution over latent codes
2. Sample code z from encoder output
3. Modify some dimensions of sampled code

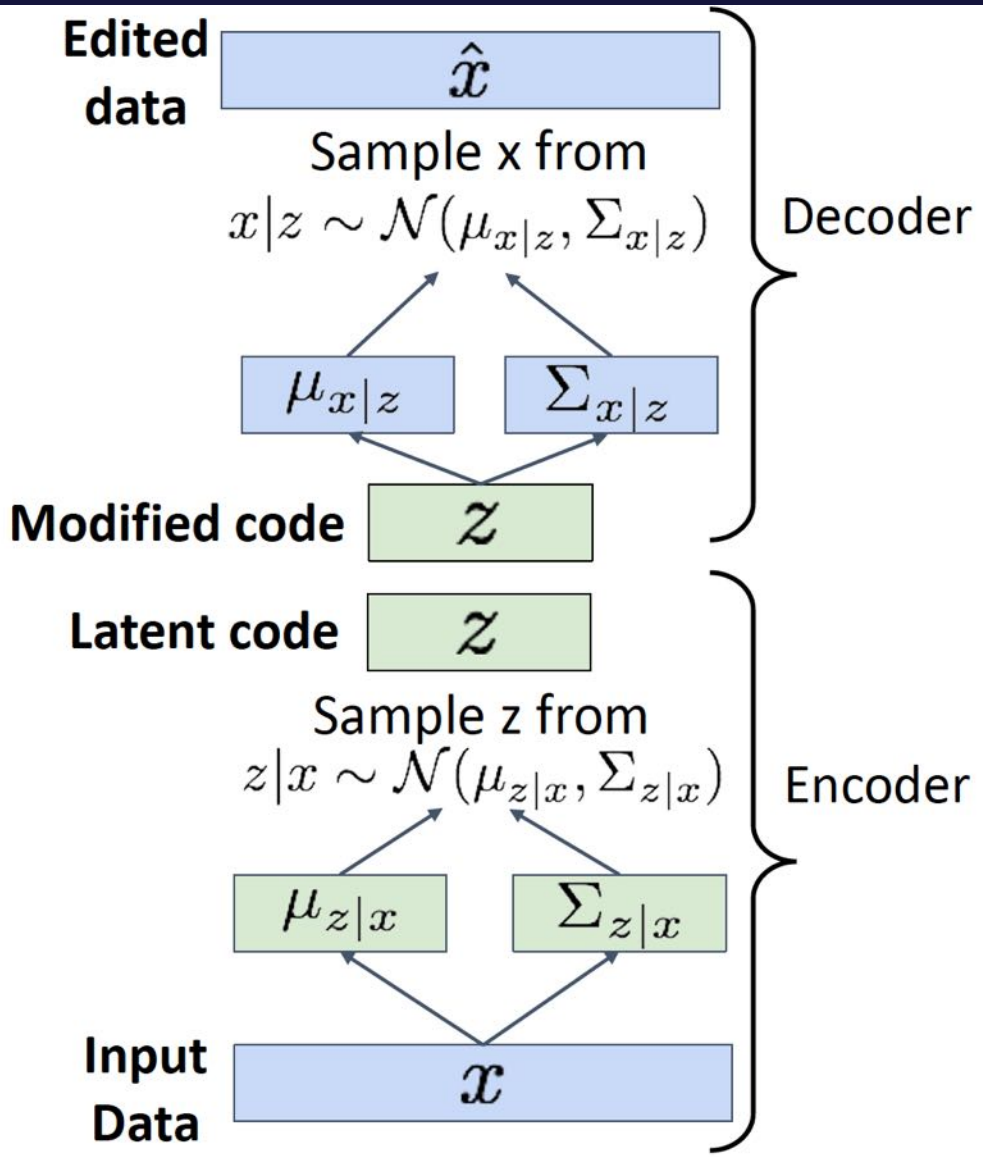


slide credit: Justin Johnson

Variational Autoencoders

After training we can **edit images**

1. Run input data through **encoder** to get a distribution over latent codes
2. Sample code z from encoder output
3. Modify some dimensions of sampled code
4. Run modified z through **decoder** to get a distribution over data samples
5. Sample new data from (4)

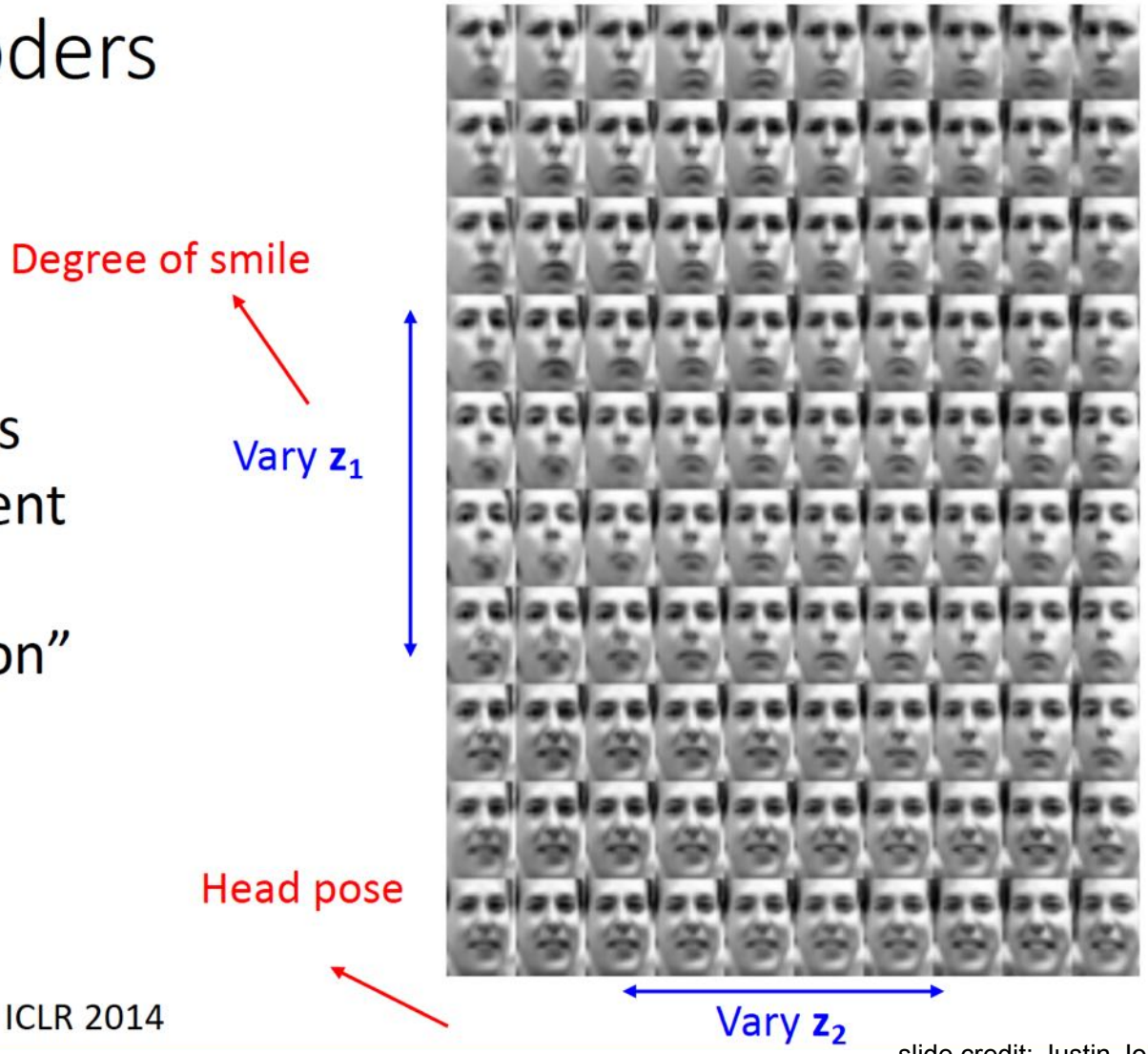


slide credit: Justin Johnson

Variational Autoencoders

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

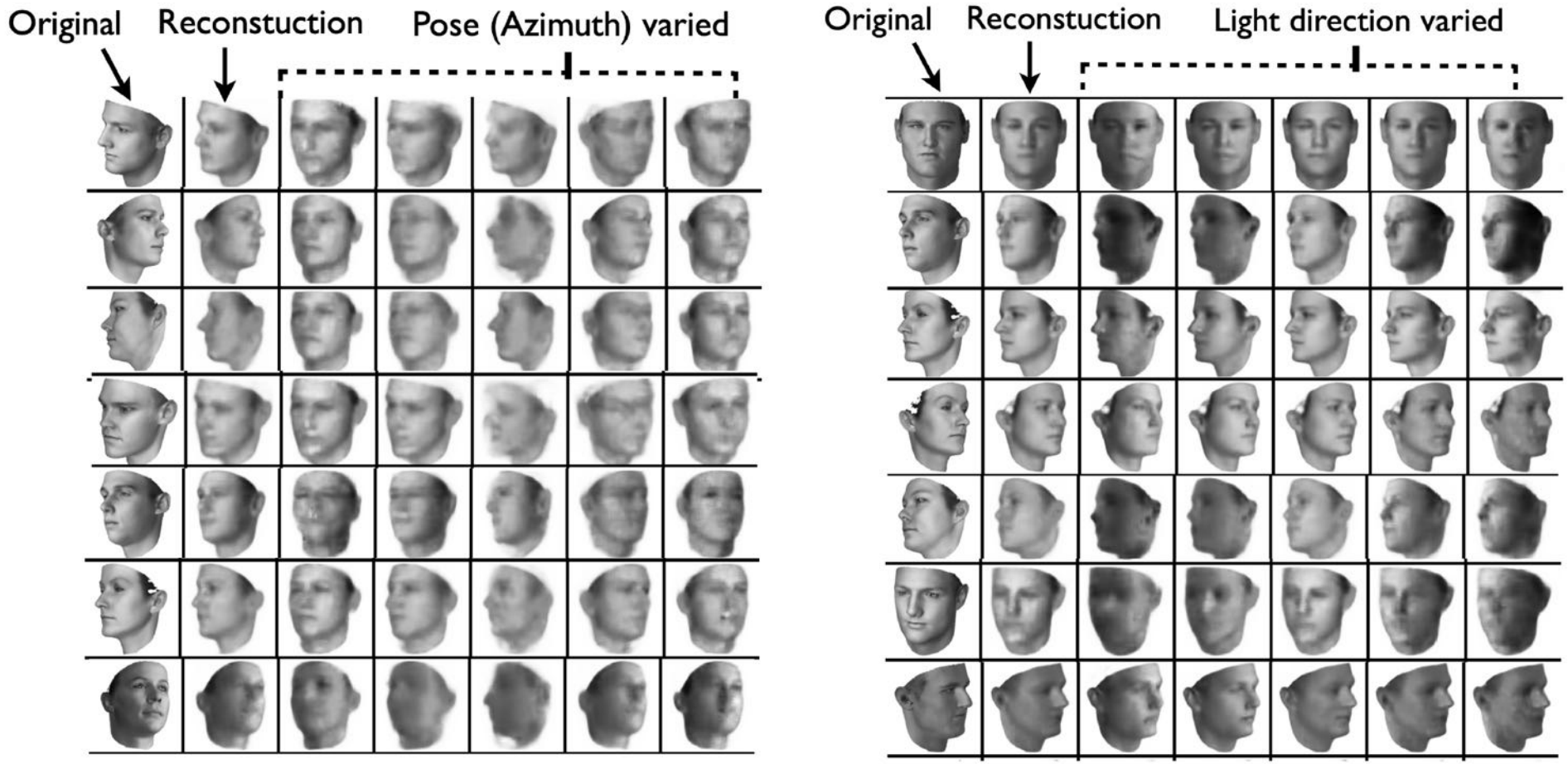
"Disentangling factors of variation"



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

slide credit: Justin Johnson

Variational Autoencoders: Image Edition



Kulkarni et al, "Deep Convolutional Inverse Graphics Networks", NeurIPS 2014

slide credit: Justin Johnson

Variational Autoencoders: Summary

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
- 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)
- Incorporating structure in latent variables, e.g., Categorical Distributions

slide credit: Justin Johnson

So far: Two Types of Generative Models

Autoregressive models
- Directly maximize p(data)
- High-quality generated images
- Slow to generate images
- No explicit latent codes

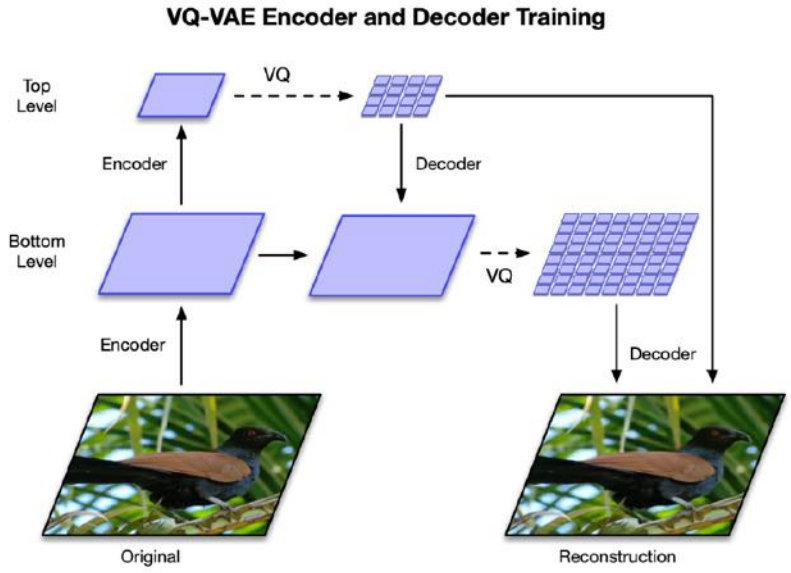
Variational models
- Maximize lower-bound on p(data)
- Generated images often blurry
- Very fast to generate images
- Learn rich latent codes

Can we combine them and get the best of both worlds?

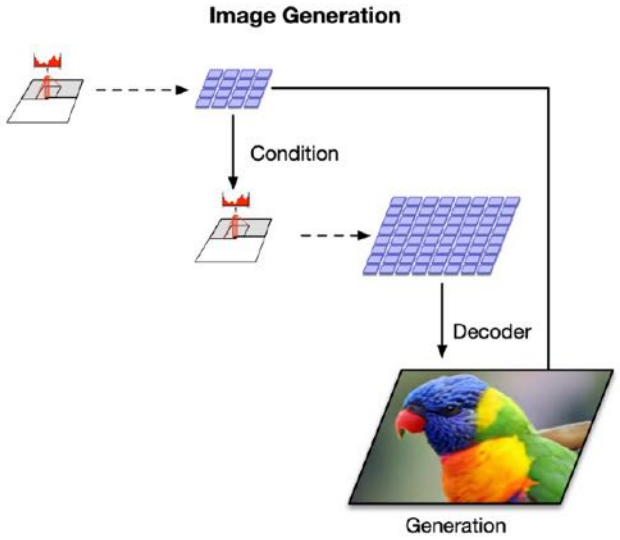
slide credit: Justin Johnson

Combining VAE + Autoregressive: Vector-Quantized Variational Autoencoder (VQ-VAE-2)

Train a VAE-like model to generate multiscale grids of latent codes



Use a multiscale PixelCNN to sample in latent code space



Razavi et al, "Generating Diverse High-Fidelity Images with VQ-VAE-2", NeurIPS 2019

slide credit: Justin Johnson

Combining VAE + Autoregressive: VQ-VAE-2

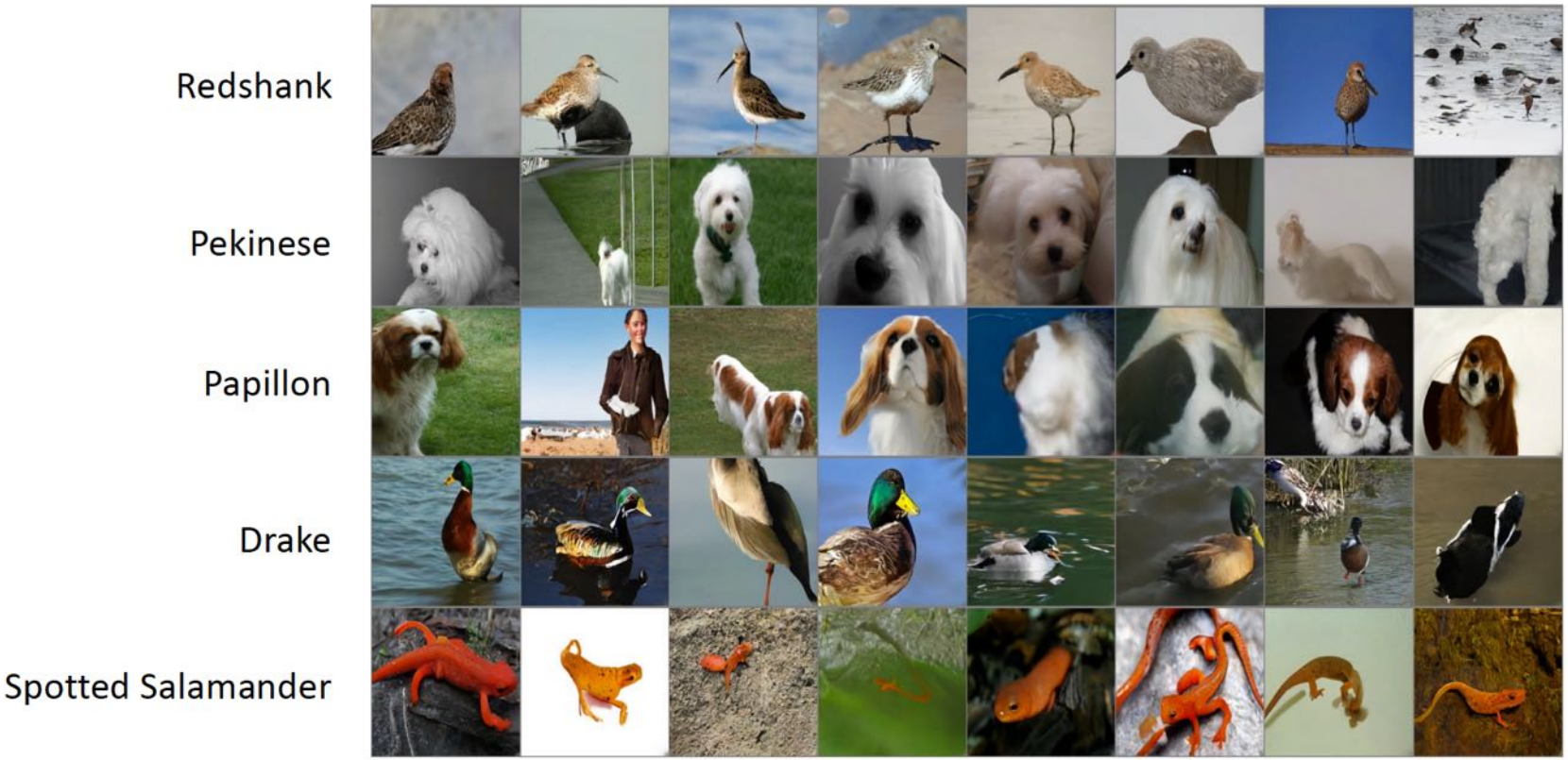
256 x 256 class-conditional samples, trained on ImageNet

Razavi et al, “Generating Diverse High-Fidelity Images with VQ-VAE-2”, NeurIPS 2019

slide credit: Justin Johnson

Combining VAE + Autoregressive: VQ-VAE-2

256 x 256 class-conditional samples, trained on ImageNet



Razavi et al, "Generating Diverse High-Fidelity Images with VQ-VAE-2", NeurIPS 2019

slide credit: Justin Johnson

Combining VAE + Autoregressive: VQ-VAE2

1024 x 1024 generated faces, trained on FFHQ

Razavi et al, “Generating Diverse High-Fidelity Images with VQ-VAE-2”, NeurIPS 2019

slide credit: Justin Johnson

Combining VAE + Autoregressive: VQ-VAE2

1024 x 1024 generated faces, trained on FFHQ

Razavi et al, “Generating Diverse High-Fidelity Images with VQ-VAE-2”, NeurIPS 2019

slide credit: Justin Johnson

Generative Models so Far:

Autoregressive Models directly maximize likelihood of training data:

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

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

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

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

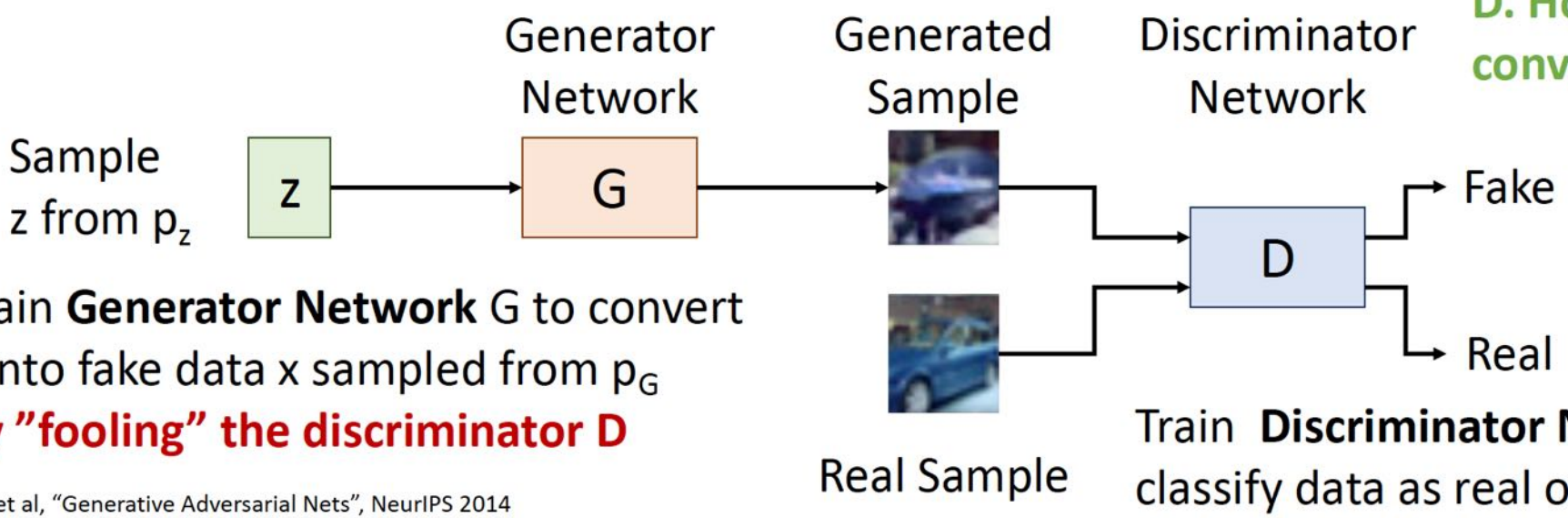
slide credit: Justin Johnson

Generative Adversarial Networks

Setup: Assume we have data x_i drawn from distribution $p_{data}(x)$. Want to sample from p_{data}.

Idea: Introduce a latent variable z with simple prior p(z).
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** p_G. Want $p_G = p_{data}$!

Jointly train G and D. Hopefully p_G converges to p_{data}!



Train **Generator Network** G to convert z into fake data x sampled from p_G
by "fooling" the discriminator D

Train **Discriminator Network** D to classify data as real or fake (1/0)

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

slide credit: Justin Johnson

Generative Adversarial Networks: Training Objective

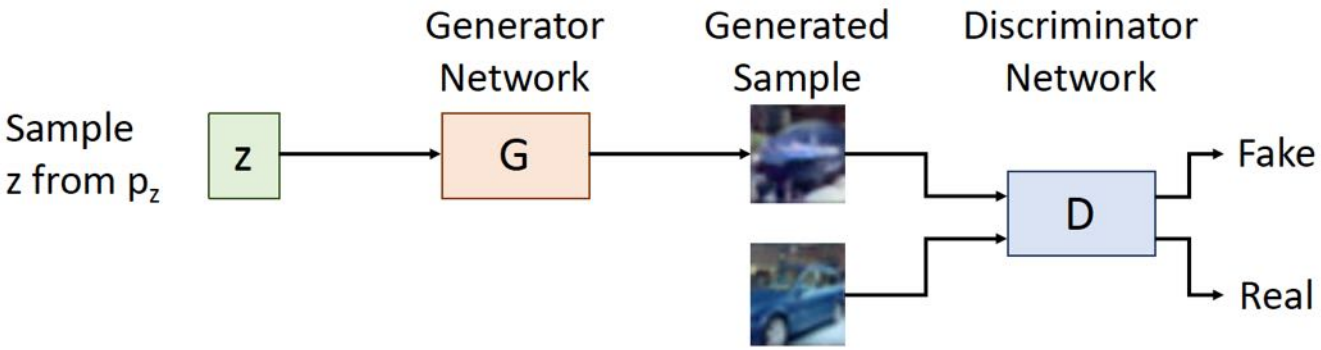
Jointly train generator G and discriminator D with a **minimax game**

Discriminator wants
D(x) = 1 for real data

Discriminator wants
D(x) = 0 for fake data

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

Generator wants
D(x) = 1 for fake data



Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

slide credit: Justin Johnson

Generative Adversarial Networks: Training Objective

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

Train G and D using alternating gradient updates

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

$$= \min_G \max_D V(G, D)$$

For t in 1, ... T:

1. (Update D) $D = D + \alpha_D \frac{\partial V}{\partial D}$
2. (Update G) $G = G - \alpha_G \frac{\partial V}{\partial G}$

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

slide credit: Justin Johnson

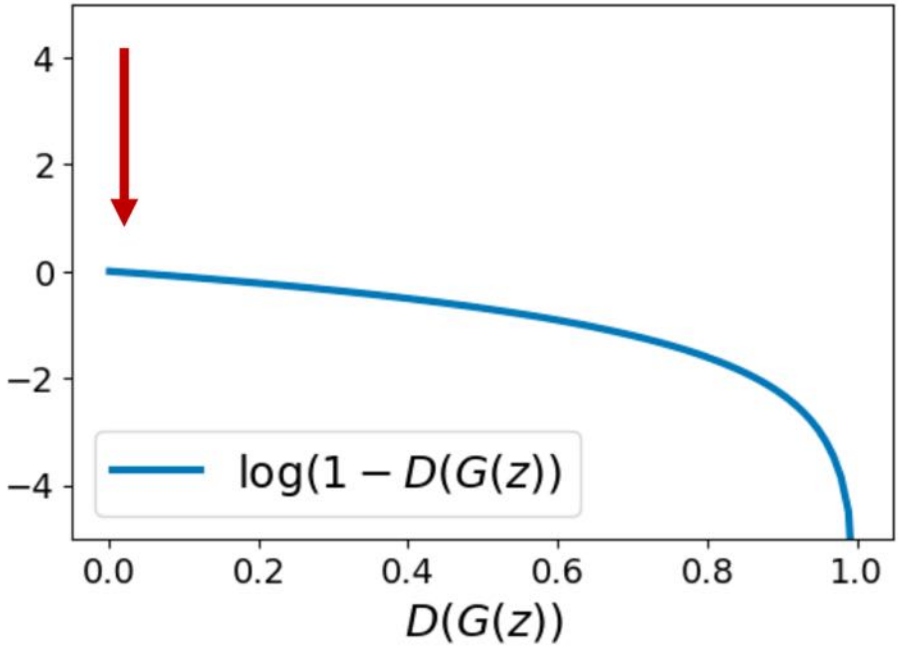
Generative Adversarial Networks: Training Objective

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

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

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

Problem: Vanishing gradients for G



Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

slide credit: Justin Johnson

Generative Adversarial Networks: Training Objective

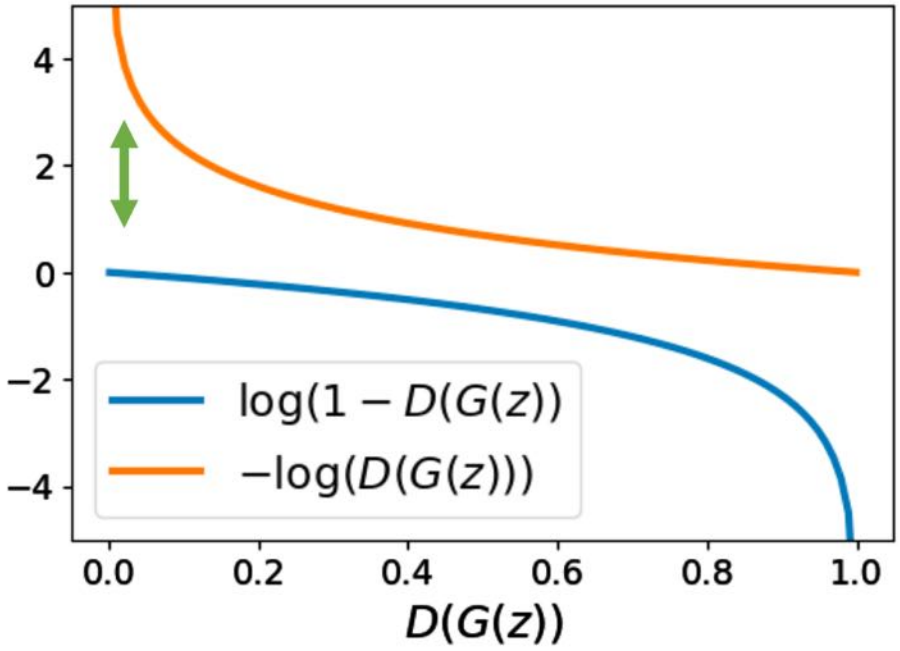
Jointly train generator G and discriminator D with a **minimax game**

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

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

Problem: Vanishing gradients for G

Solution: Right now G is trained to minimize log(1-D(G(z)). Instead, train G to minimize –log(D(G(z)). Then G gets strong gradients at start of training!



Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

slide credit: Justin Johnson

Generative Adversarial Networks: Optimality

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

Why is this particular objective a good idea?

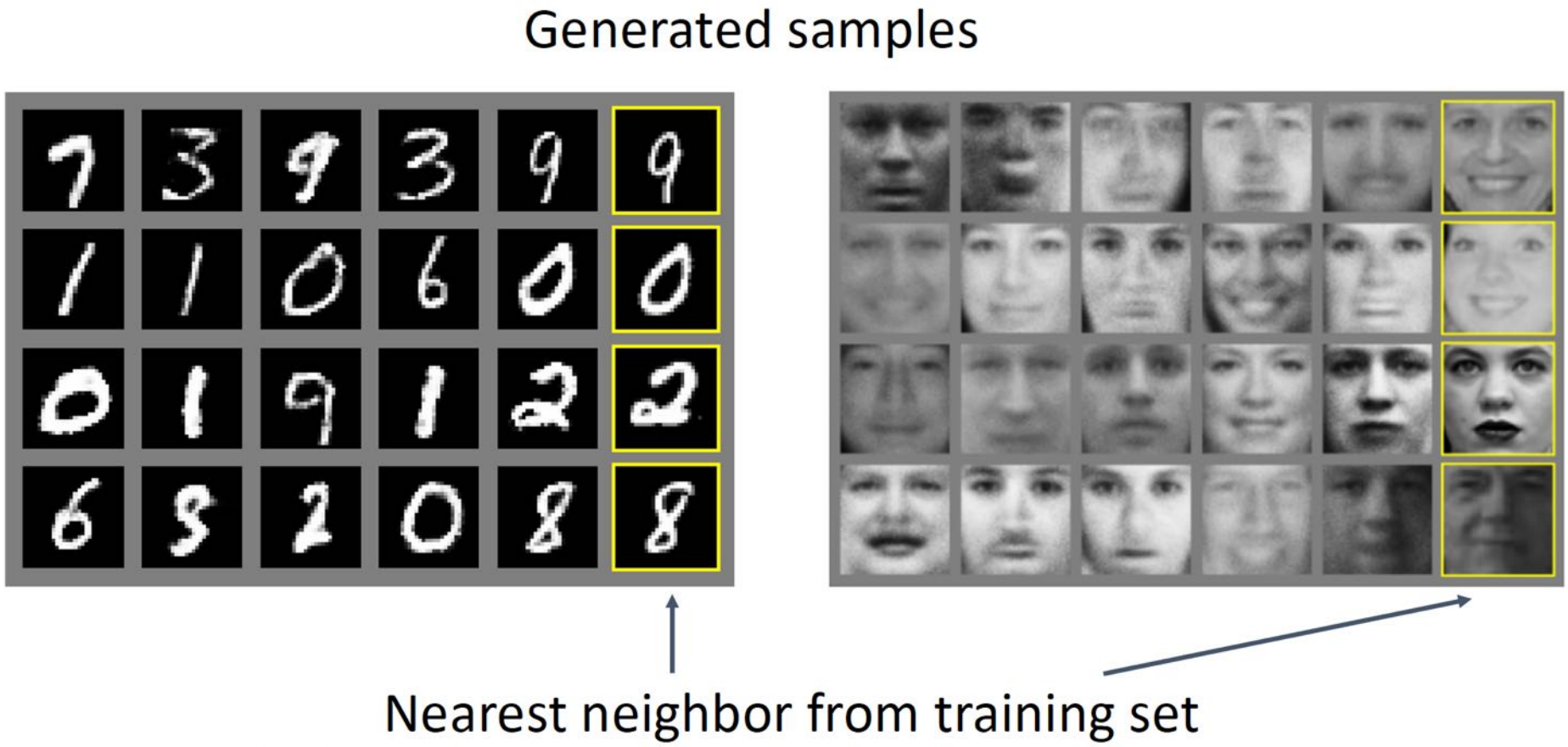
$$\min_{G} \max_{D} \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log \left(1 - D(G(z)) \right) \right] \right)$$

This minimax game achieves its global minimum when $p_G = p_{data}$!

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

slide credit: Justin Johnson

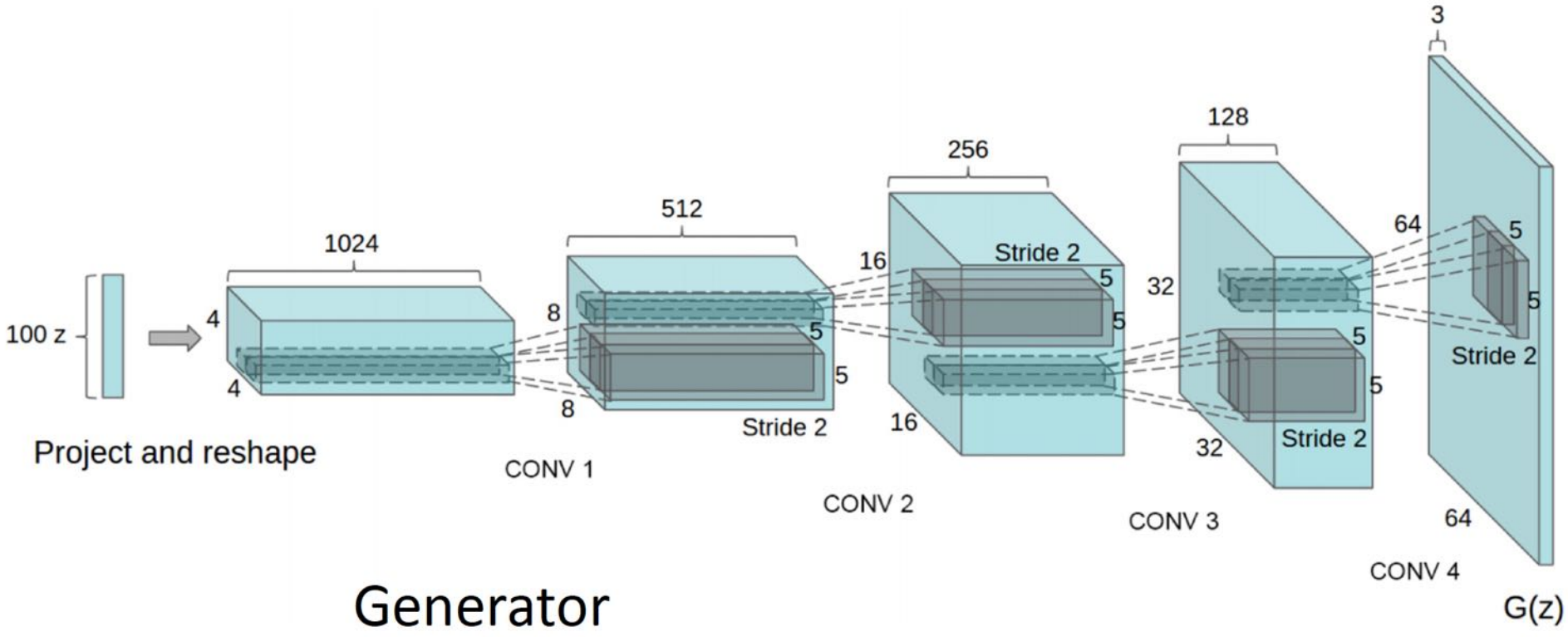
Generative Adversarial Networks: Results



Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

slide credit: Justin Johnson

Generative Adversarial Networks: DC-GAN

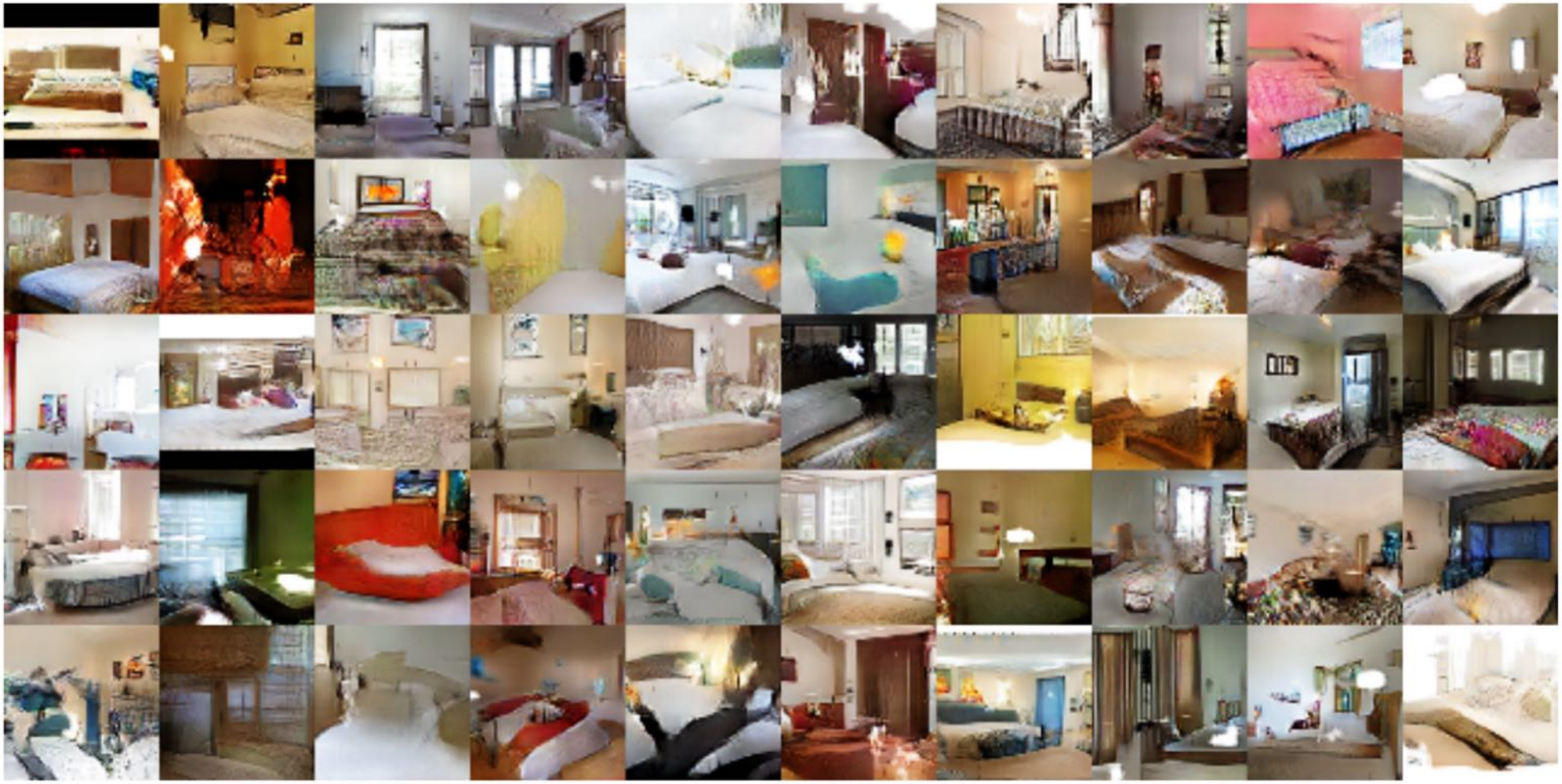


Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

slide credit: Justin Johnson

Generative Adversarial Networks: DC-GAN

Samples from the model look much better!



Radford et al, ICLR 2016

slide credit: Justin Johnson

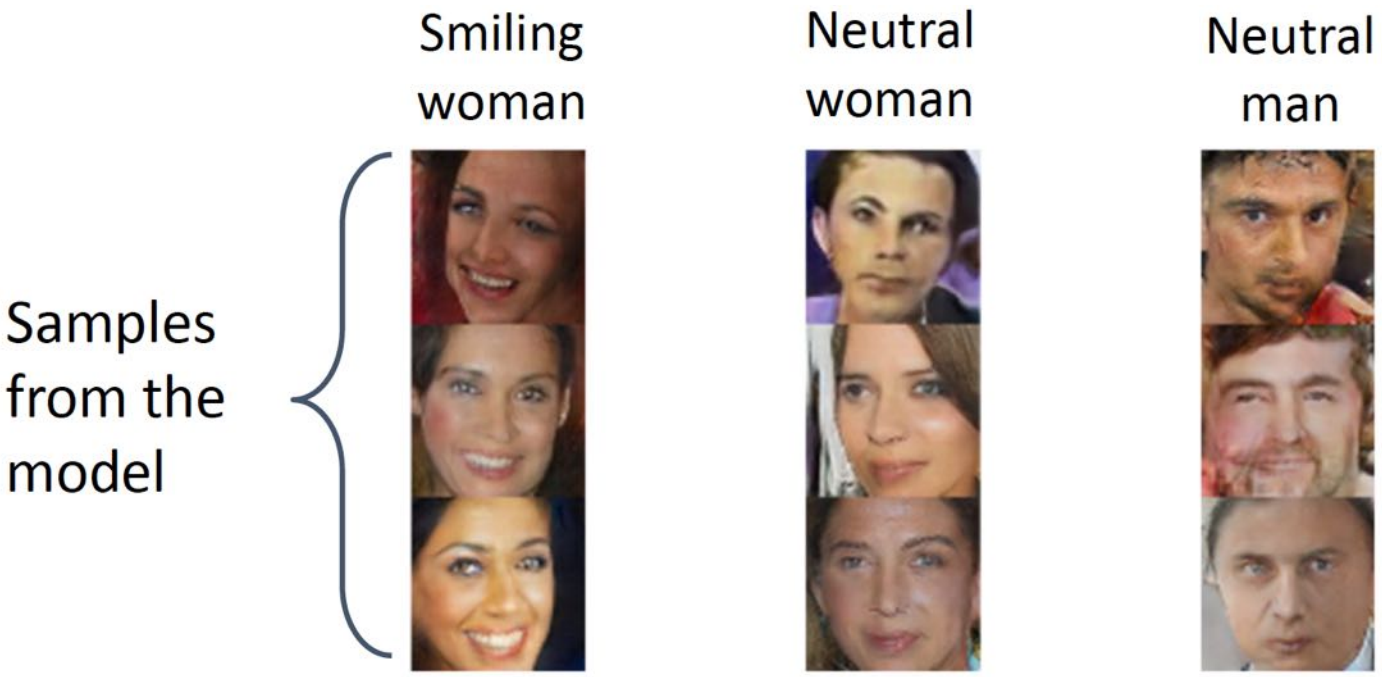
Generative Adversarial Networks: DC-GAN

Interpolating between points in latent z space

Radford et al, ICLR 2016

slide credit: Justin Johnson

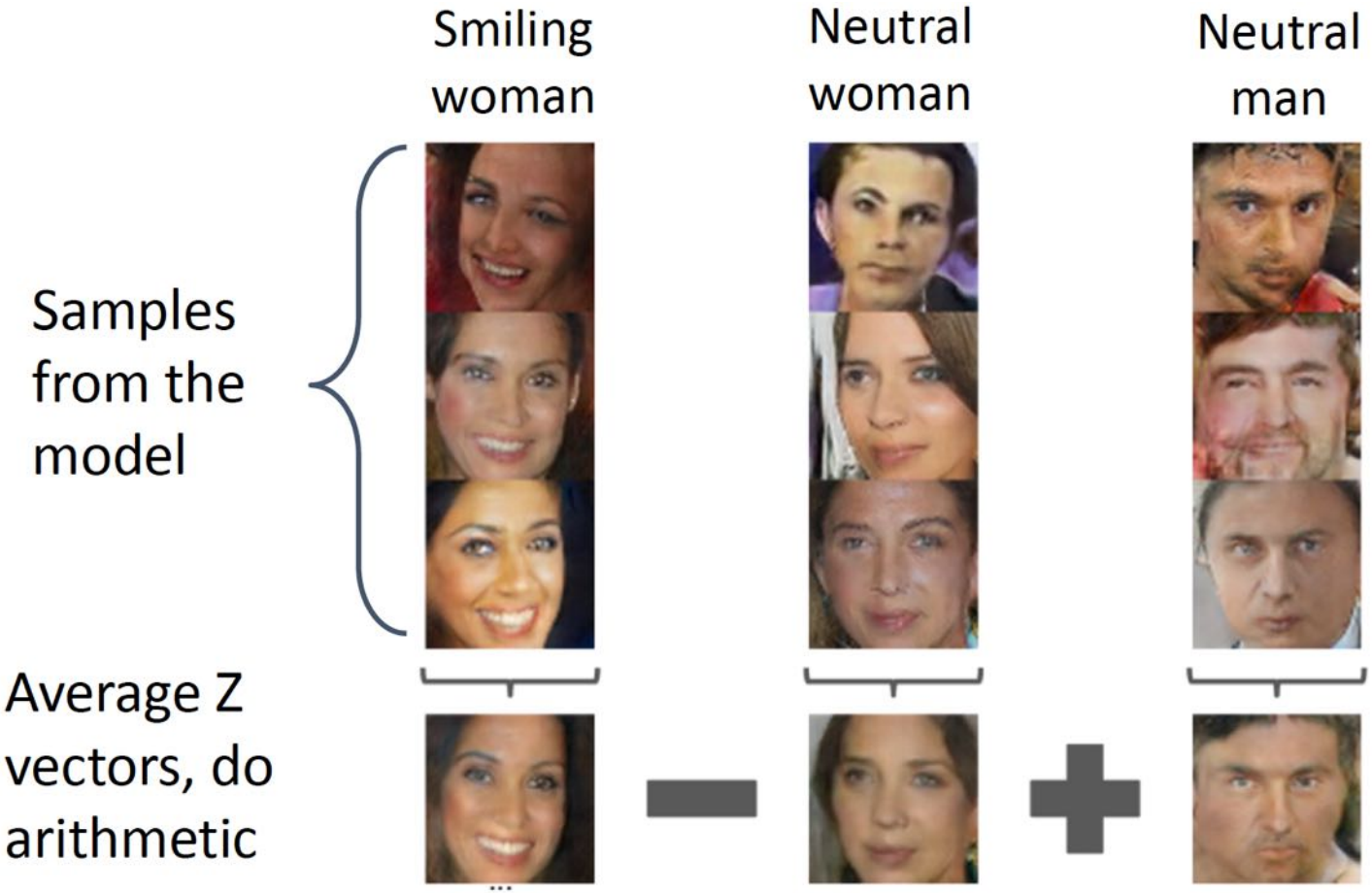
Generative Adversarial Networks: Vector-Math



Radford et al, ICLR 2016

slide credit: Justin Johnson

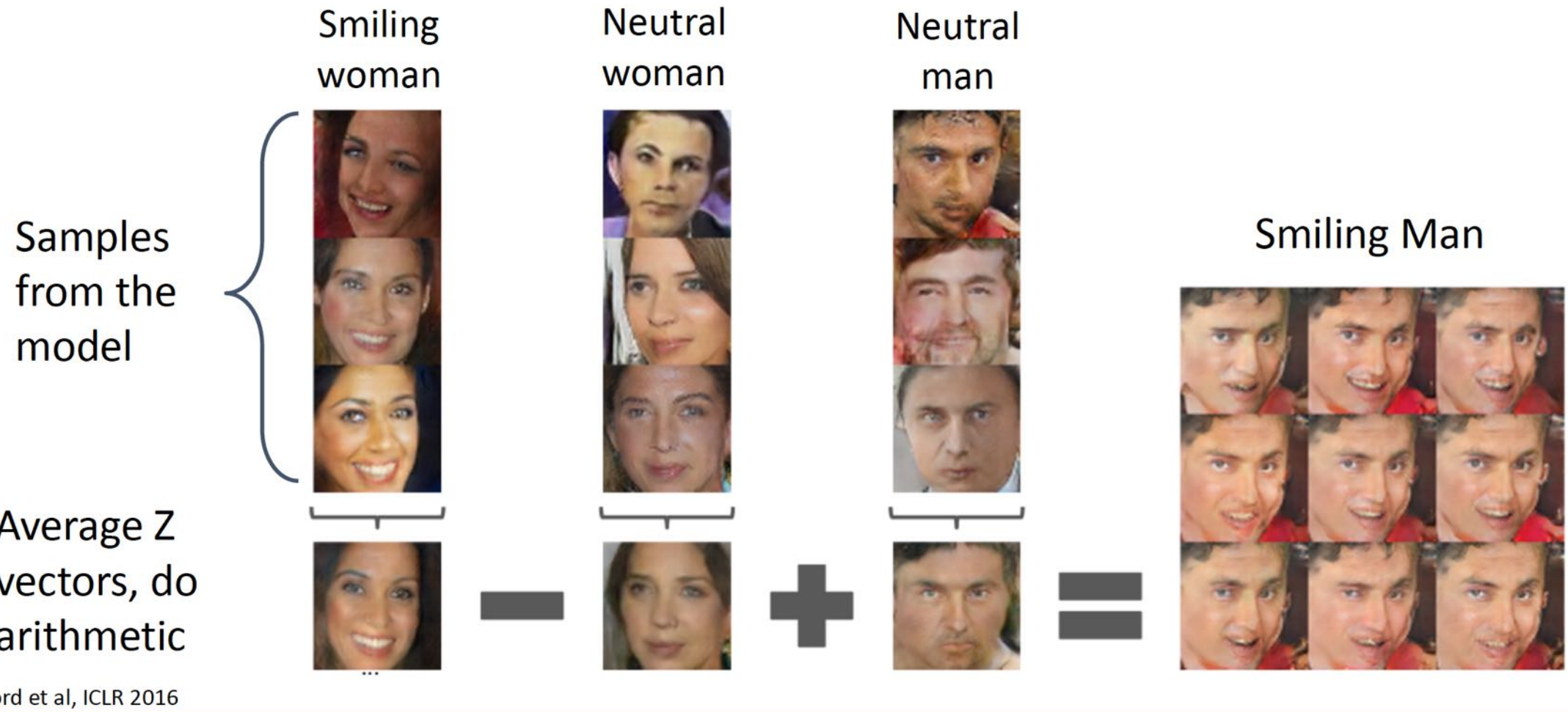
Generative Adversarial Networks: Vector-Math



Radford et al, ICLR 2016

slide credit: Justin Johnson

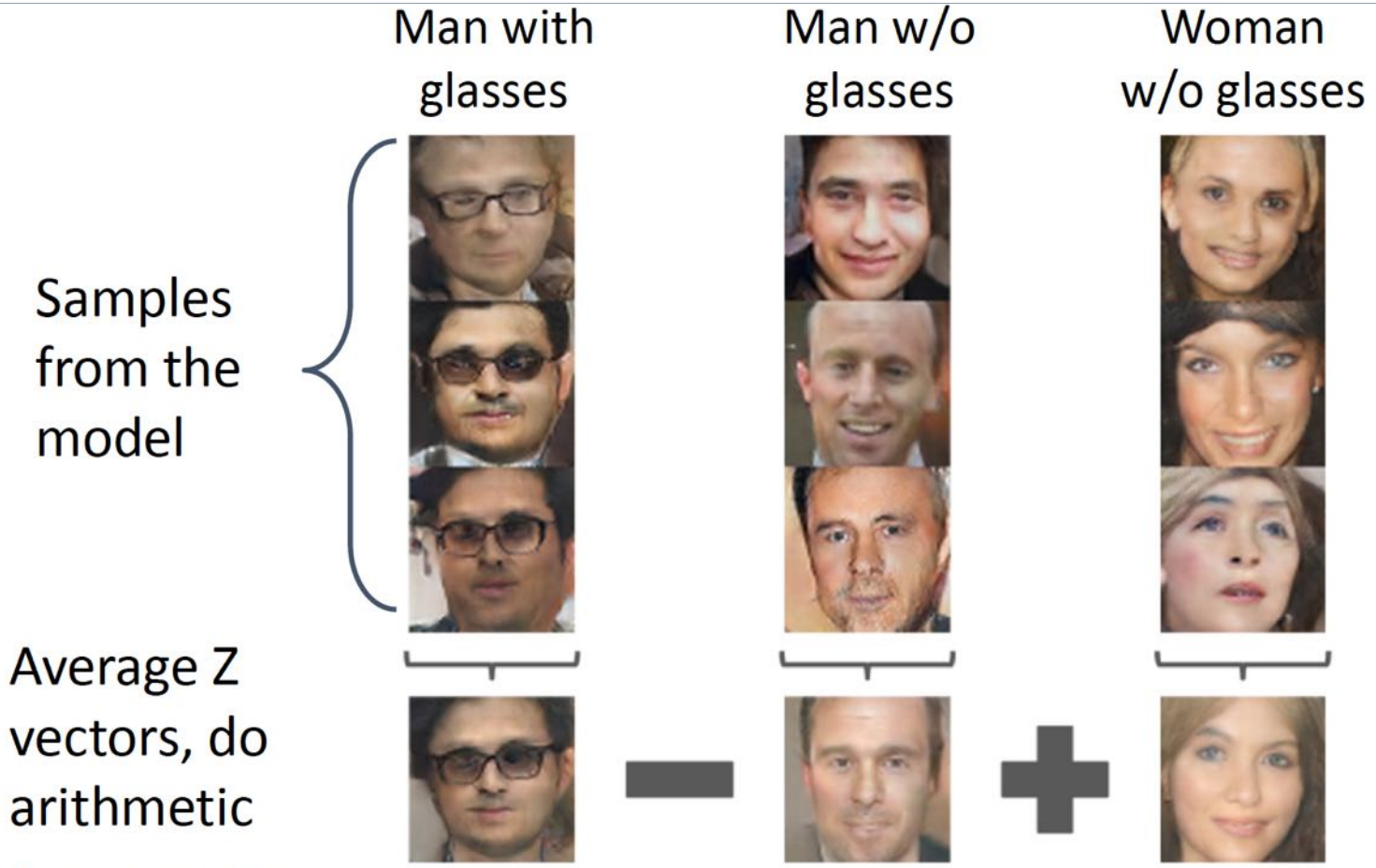
Generative Adversarial Networks: Vector-Math



Radford et al, ICLR 2016

slide credit: Justin Johnson

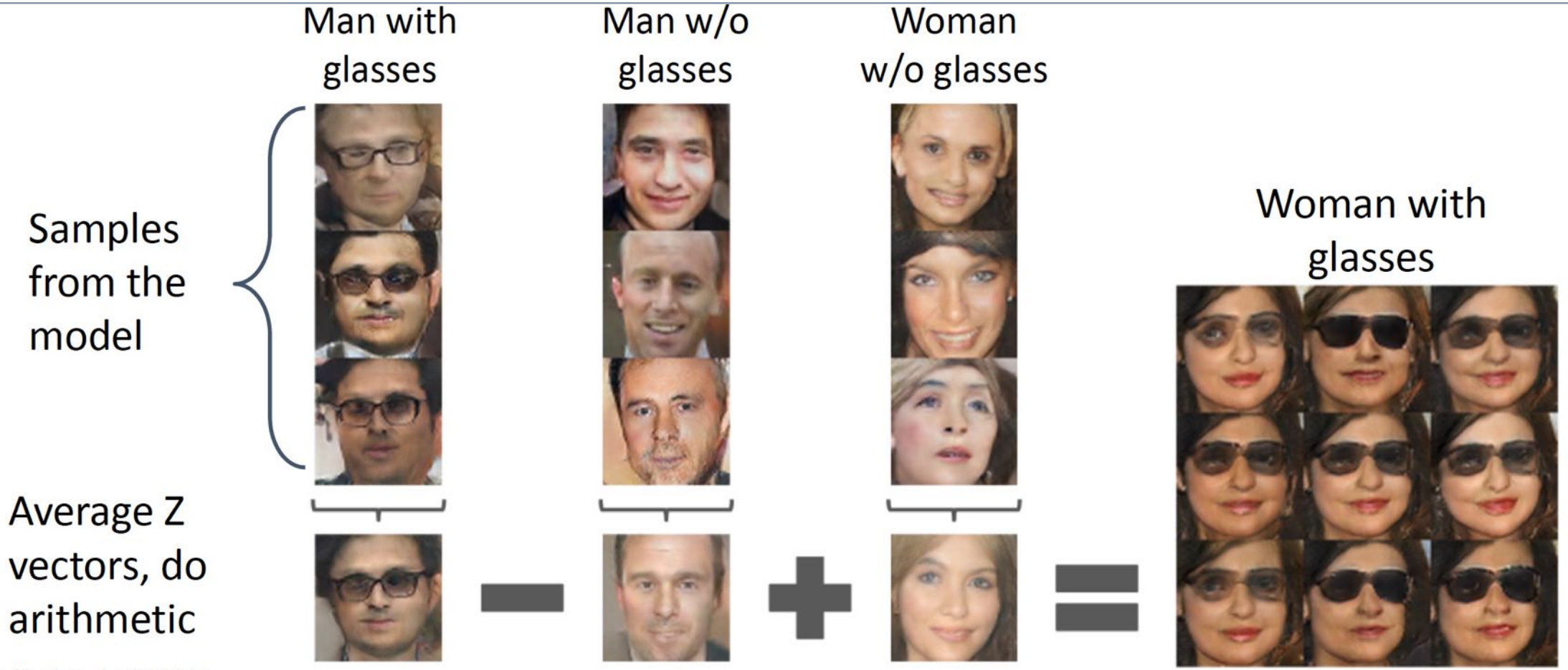
Generative Adversarial Networks: Vector-Math



Radford et al, ICLR 2016

slide credit: Justin Johnson

Generative Adversarial Networks: Vector-Math

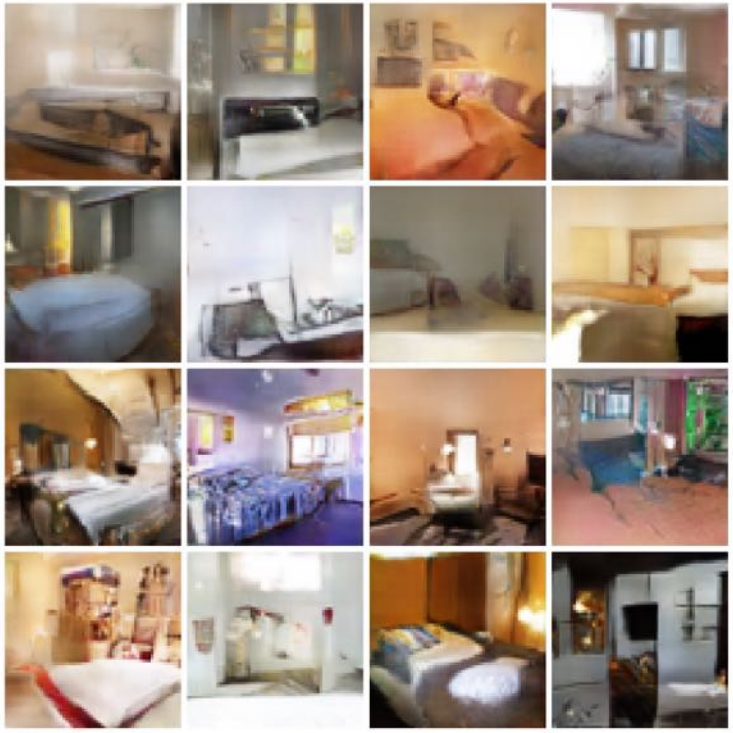


Radford et al, ICLR 2016

slide credit: Justin Johnson

Generative Adversarial Networks: Improved Loss Functions

Wasserstein GAN (WGAN)



Arjovsky, Chintala, and Bouttou, “Wasserstein GAN”, 2017

WGAN with Gradient Penalty (WGAN-GP)

Gulrajani et al, “Improved Training of Wasserstein GANs”, NeurIPS 2017

slide credit: Justin Johnson

Generative Adversarial Networks: Higher Resolution

512 x 384 cars

1024 x 1024 faces

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

slide credit: Justin Johnson

Generative Adversarial Networks: StyleGan2

Karras et al, “Analyzing and Improving the Image Quality of StyleGAN”, CVPR 2020

slide credit: Justin Johnson

Conditional GANs

Recall: Conditional Generative Models learn p(x|y) instead of p(x)

Make generator and discriminator both take label y as an additional input!

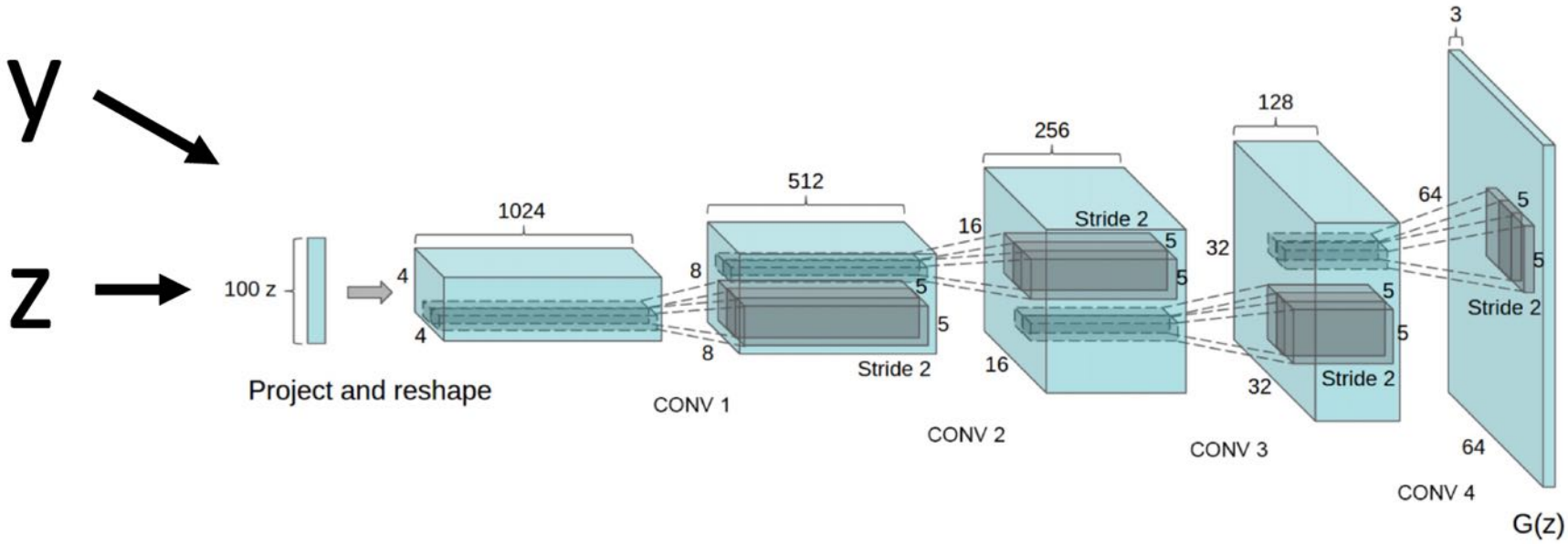


Figure credit: Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

slide credit: Justin Johnson

Conditional GANs: BigGAN

Brock et al, “Large Scale GAN Training for High Fidelity Natural Image Synthesis”, ICLR 2019

512x512 images on ImageNet

slide credit: Justin Johnson

Conditioning on More than Labels: Text to Image

Zhang et al, "StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks.", TPAMI 2018
Zhang et al, "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks.", ICCV 2017
Reed et al, "Generative Adversarial Text-to-Image Synthesis", ICML 2016

slide credit: Justin Johnson

Text-to-Image: DALL-E

Step 1: Train VQ-VAE (discrete grid of latent codes)

Step 2: Train autoregressive Transformer model to predict sequence of latent codes (Giant model on 250M image/text pairs)

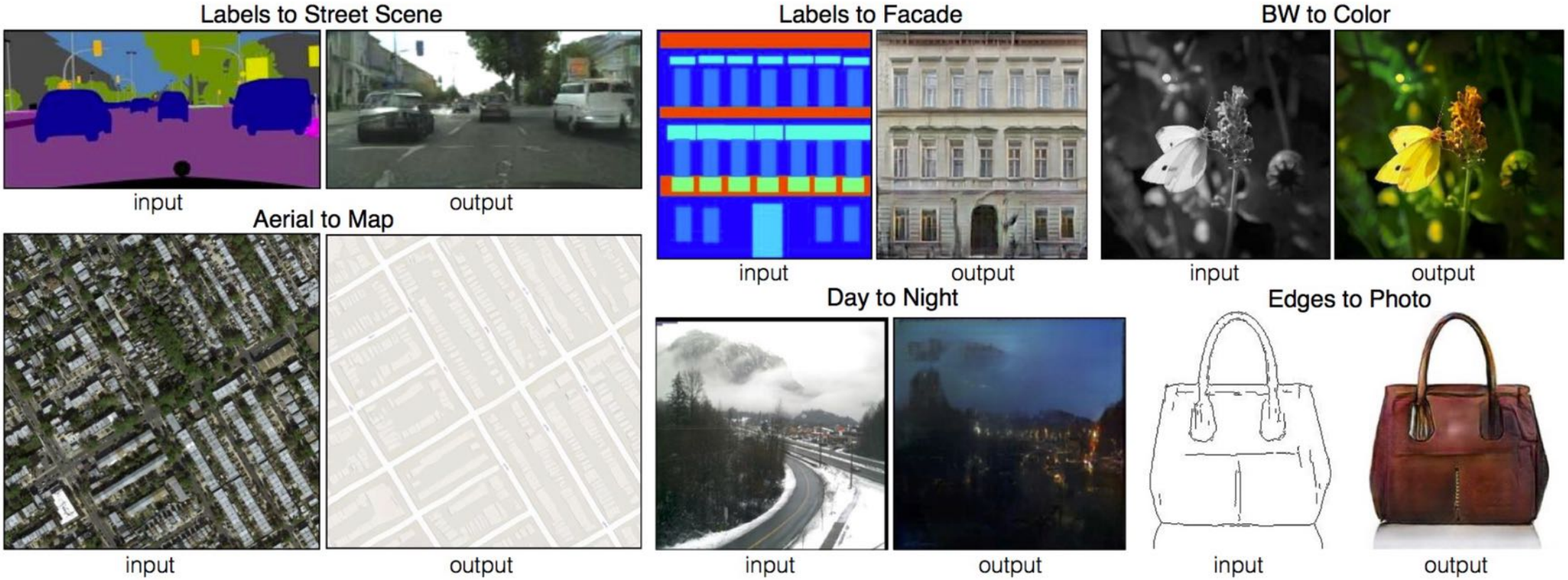
Step 3: Given text prompt, sample new image codes; pass through VQ-VAE decoder to generate images

an illustration of a baby hedgehog in a christmas sweater walking a dog

Ramesh et al, "Zero-Shot Text-to-Image Generation", ICML 2021

slide credit: Justin Johnson

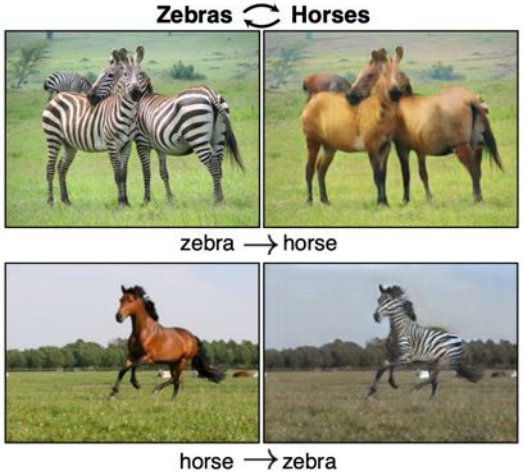
Image-to-Image Translation: Pix2Pix

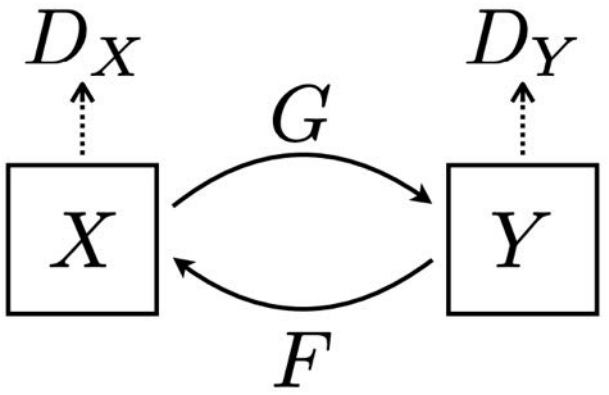


Isola et al, "Image-to-Image Translation with Conditional Adversarial Nets", CVPR 2017

slide credit: Justin Johnson

Cycle GAN: Unpaired Image-to-Image Translation

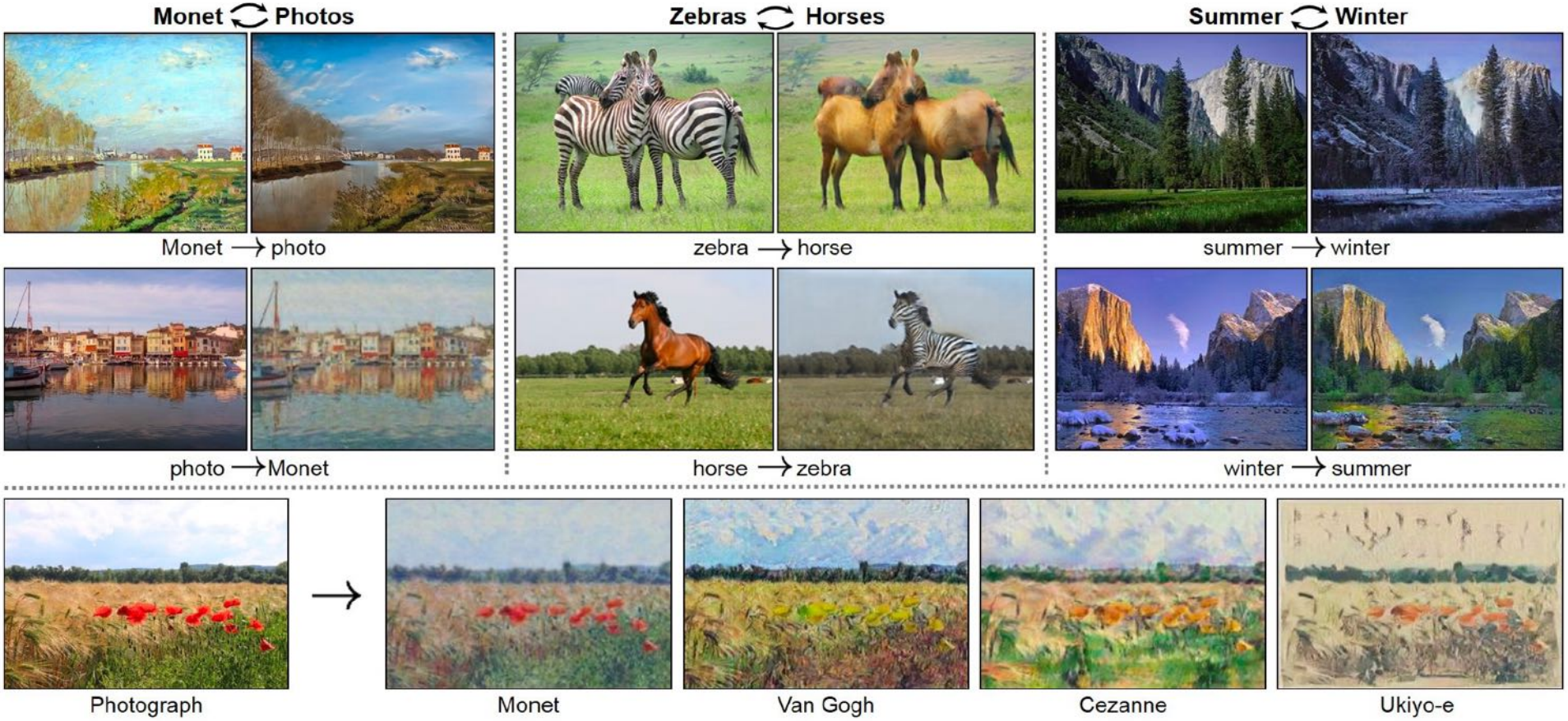




Zhu, Jun-Yan, et al. “Unpaired image-to-image translation using cycle-consistent adversarial networks.” CVPR 2017.

slide credit: Justin Johnson

Cycle GAN: Unpaired Image-to-Image Translation

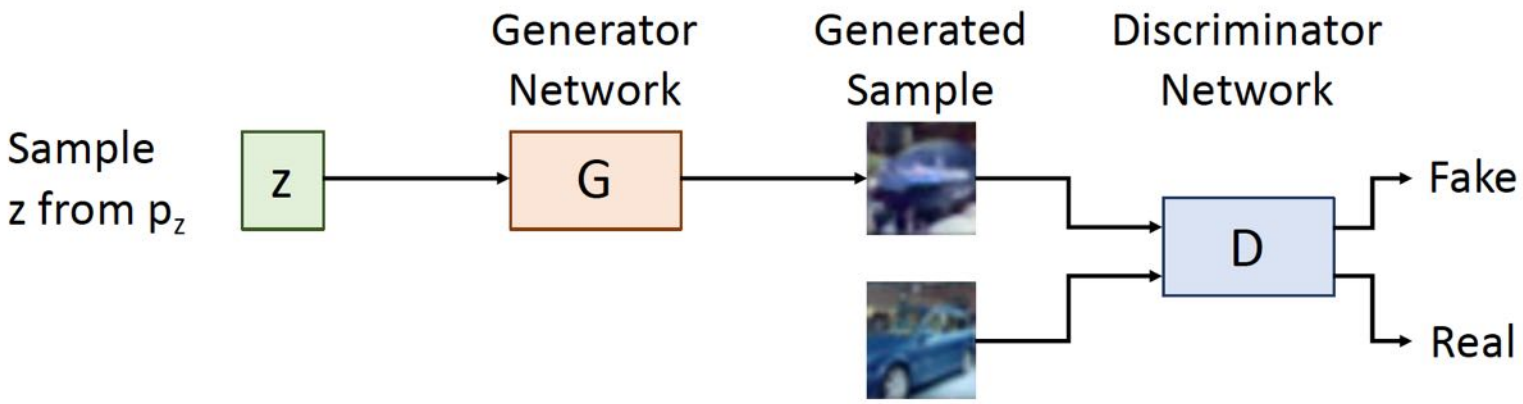


Zhu et al, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", ICCV 2017

slide credit: Justin Johnson

GAN Summary

Jointly train two networks:
Discriminator: Classify data as real or fake
Generator: Generate data that fools the discriminator



Under some assumptions, generator converges to true data distribution
Many applications! Very active area of research!

slide credit: Justin Johnson

Generative Model Summary

Autoregressive Models directly maximize likelihood of training data:

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

Good image quality, can evaluate with perplexity. Slow to generate data, needs tricks to scale up.

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

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

Latent z allows for powerful interpolation and editing applications.

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

slide credit: Justin Johnson