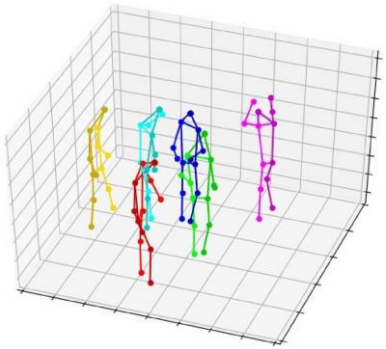
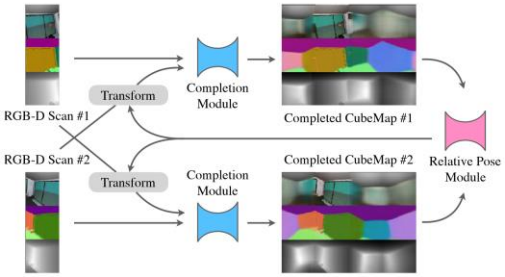
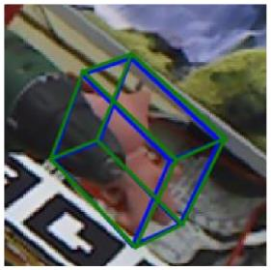
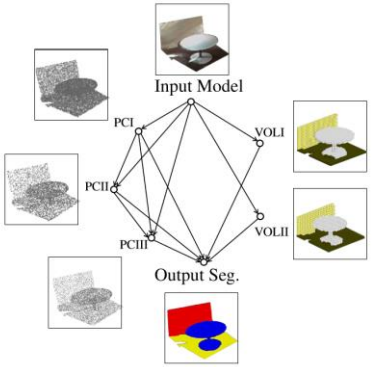
CS376 Computer Vision
Lecture 22: Generative Models



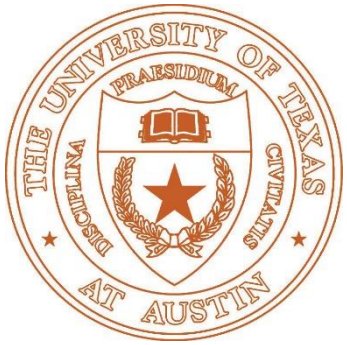






Qixing Huang

Nov. 9th 2021



Why generative models

- From the machine learning perspective, they share many techniques for visual recognition
- They appear as sub-modules for the networks used in visual recognition
- They provide training data for supervising neural networks

Overview

- Unsupervised Learning

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

Supervised vs Unsupervised Learning

- **Supervised Learning**

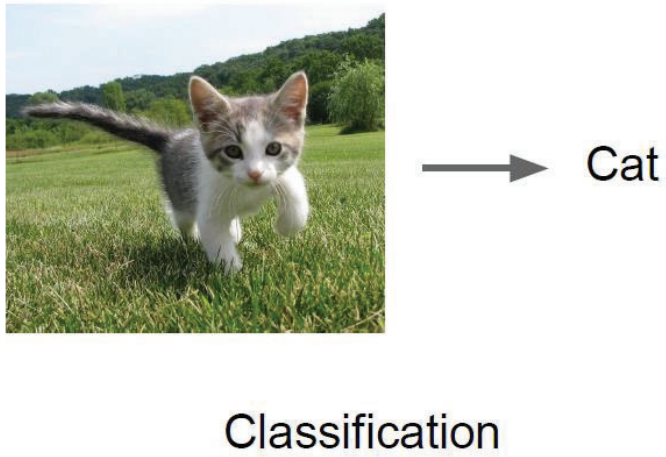
- **Data**: (x, y), x is data, y is label

- **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

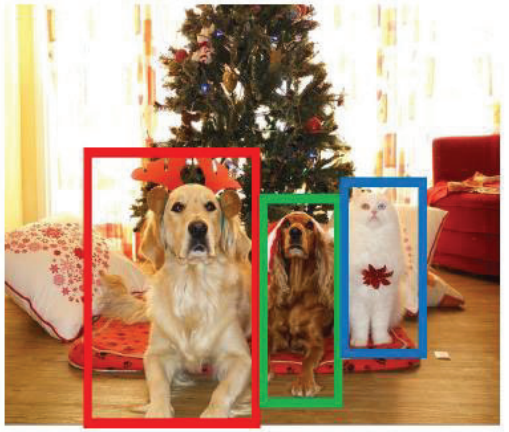
- **Examples**: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Supervised vs Unsupervised Learning

- **Supervised Learning**

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

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



Object Detection

- **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

Semantic Segmentation

- **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.

A cat sitting on a suitcase on the floor

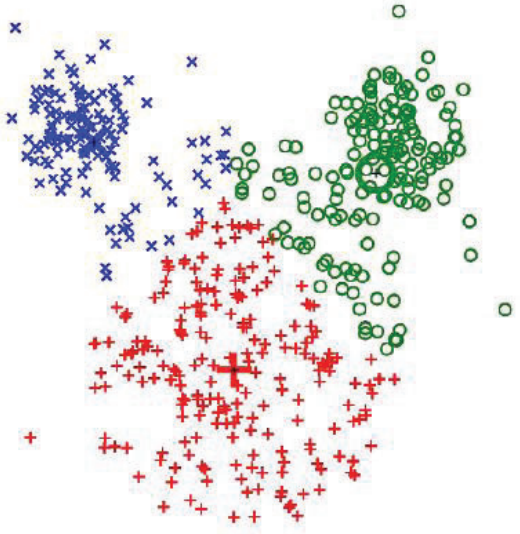
Image captioning

Supervised vs Unsupervised Learning

- **Unsupervised Learning**
- **Data**: (x, y), 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

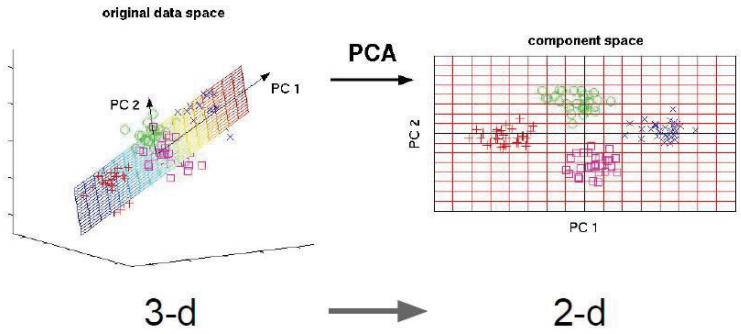
- **Unsupervised Learning**
- **Data**: (x, y), just data, no labels!
- **Goal**: Learn some underlying hidden *structure* of the data
- **Examples**: Clustering, dimensionality reduction, feature learning, density estimation, etc.



K-means clustering

Supervised vs Unsupervised Learning

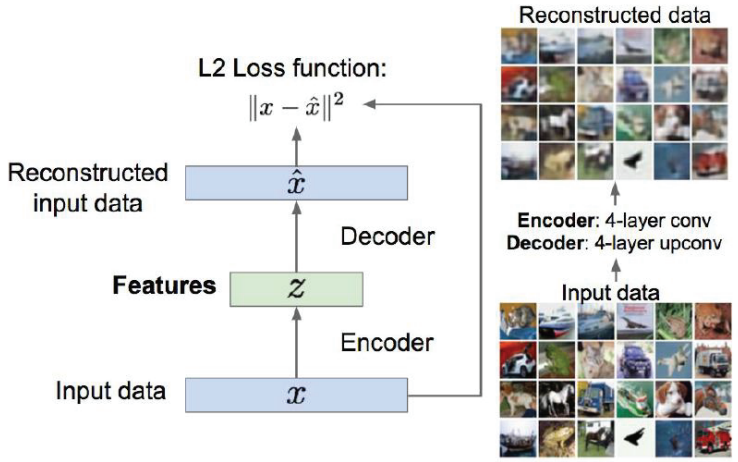
- **Unsupervised Learning**
- **Data**: (x, y), just data, no labels!
- **Goal**: Learn some underlying hidden *structure* of the data
- **Examples**: Clustering, dimensionality reduction, feature learning, density estimation, etc.



Principal Component Analysis
(Dimensionality reduction)

Supervised vs Unsupervised Learning

- **Unsupervised Learning**
- **Data**: (x, y), just data, no labels!
- **Goal**: Learn some underlying hidden *structure* of the data
- **Examples**: Clustering, dimensionality reduction, feature learning, density estimation, etc.



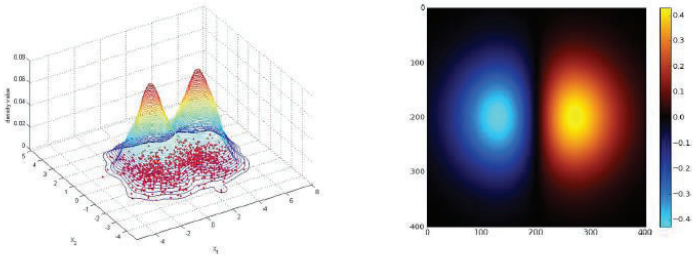
Autoencoders
(Feature learning)

Supervised vs Unsupervised Learning

- **Unsupervised Learning**
- **Data**: (x, y), just data, no labels!
- **Goal**: Learn some underlying hidden *structure* of the data
- **Examples**: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

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**

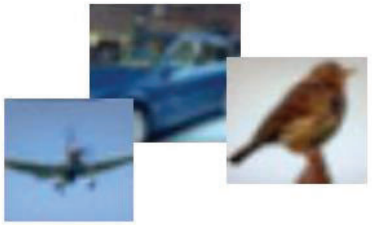
Training data is cheap

- **Data**: (x, y), just data, no labels!
- **Goal**: Learn some underlying hidden *structure* of the data
- **Examples**: C dimensional feature lear estimation,

Solve unsupervised learning => understand structure of visual world

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 $p_{data}(x)$

Addresses density estimation, a core problem in unsupervised learning

Several flavors:

- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it

Taxonomy of Generative Models

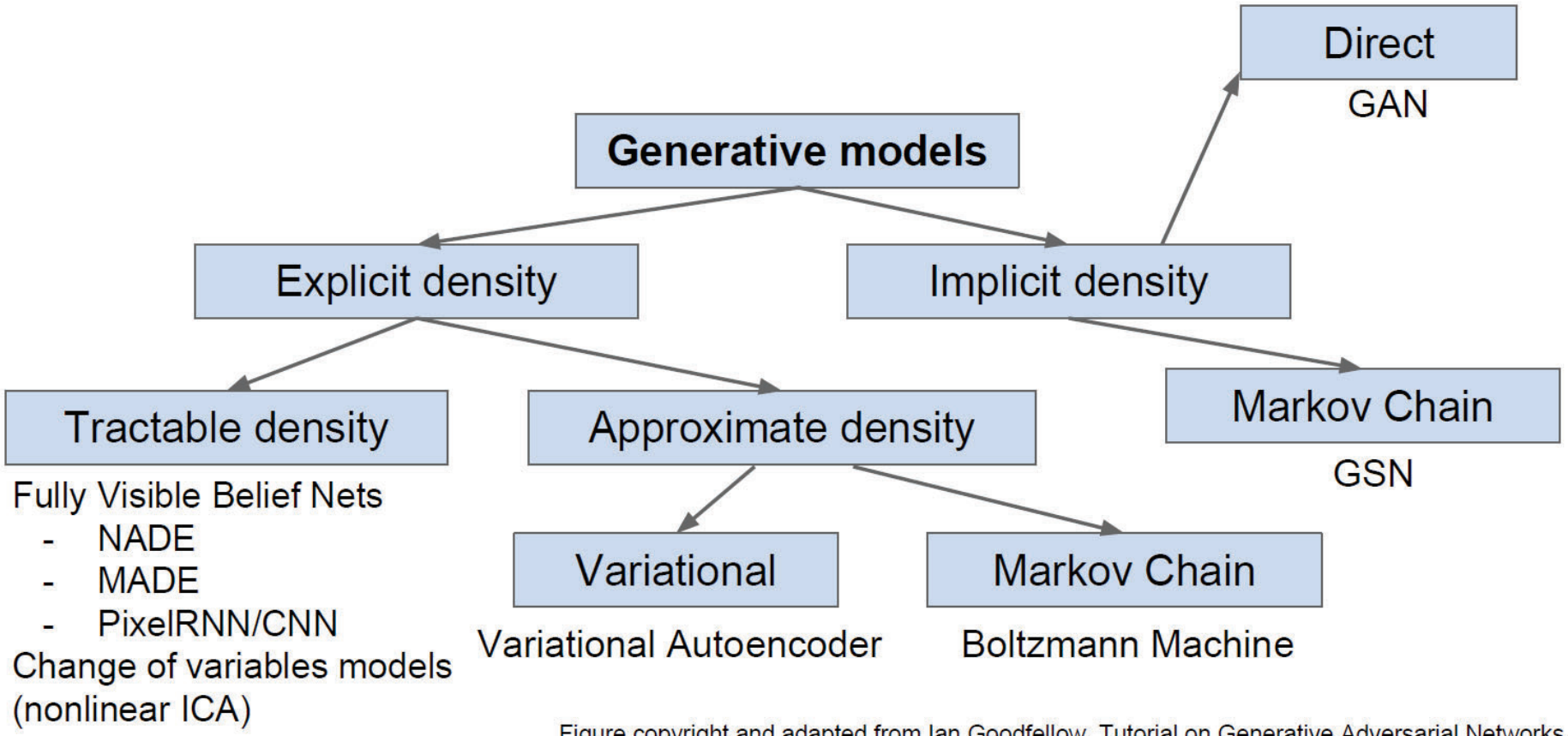


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

PixelRNN and PixelCNN

Fully visible belief network

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

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

Likelihood of image x

Probability of i'th pixel value given all previous pixels

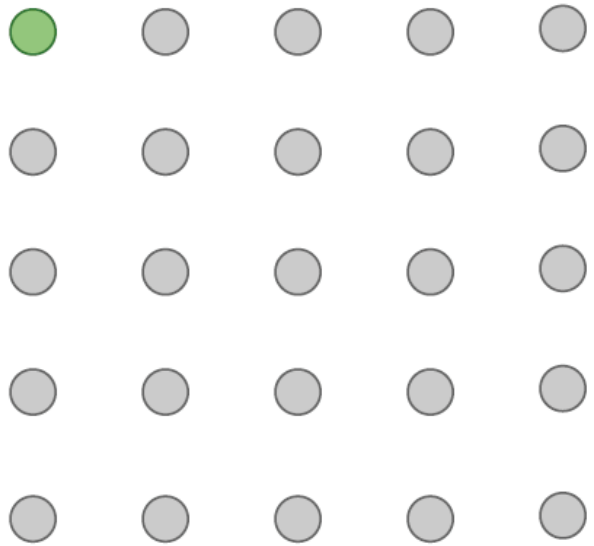
- Then maximize likelihood of training data

Complex distribution over pixe values => Express using a neu network!

PixelRNN *[van der Oord et al. 2016]*

- Generate image pixels starting from corner

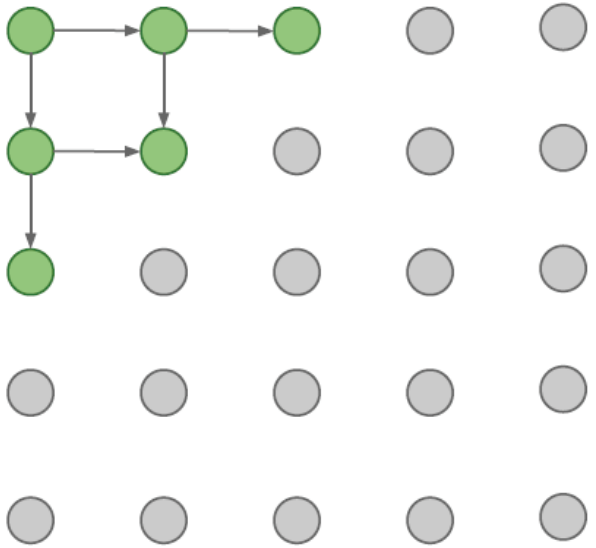
- Dependency on previous pixels modeled using an RNN (LSTM)



PixelRNN *[van der Oord et al. 2016]*

- Generate image pixels starting from corner

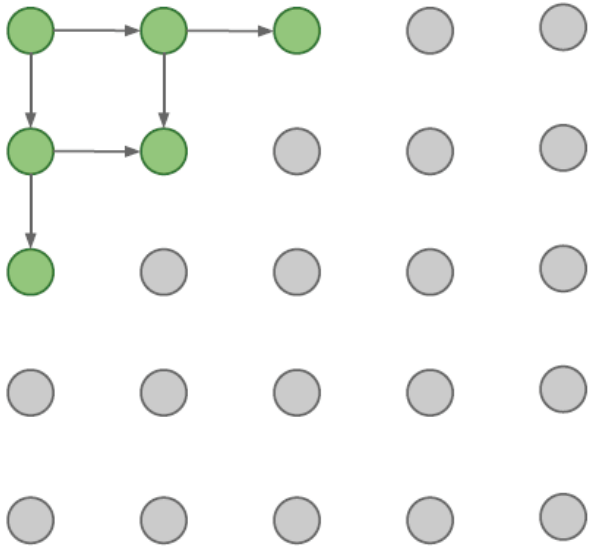
- Dependency on previous pixels modeled using an RNN (LSTM)



PixelRNN *[van der Oord et al. 2016]*

- Generate image pixels starting from corner

- Dependency on previous pixels modeled using an RNN (LSTM)

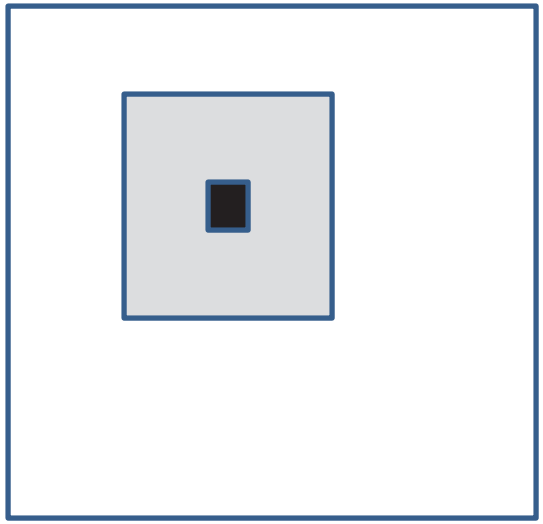


Drawback: sequential generation is slow!

PixelCNN *[van der Oord et al. 2016]*

- Still generate image pixels starting from corner

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



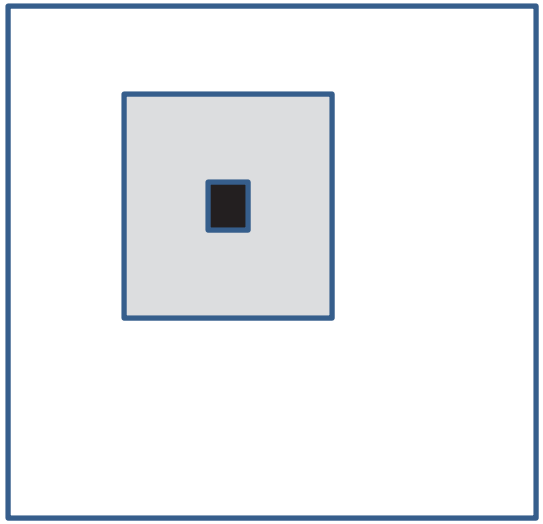
Training: maximize likelihood of training images

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

PixelCNN *[van der Oord et al. 2016]*

- Still generate image pixels starting from corner

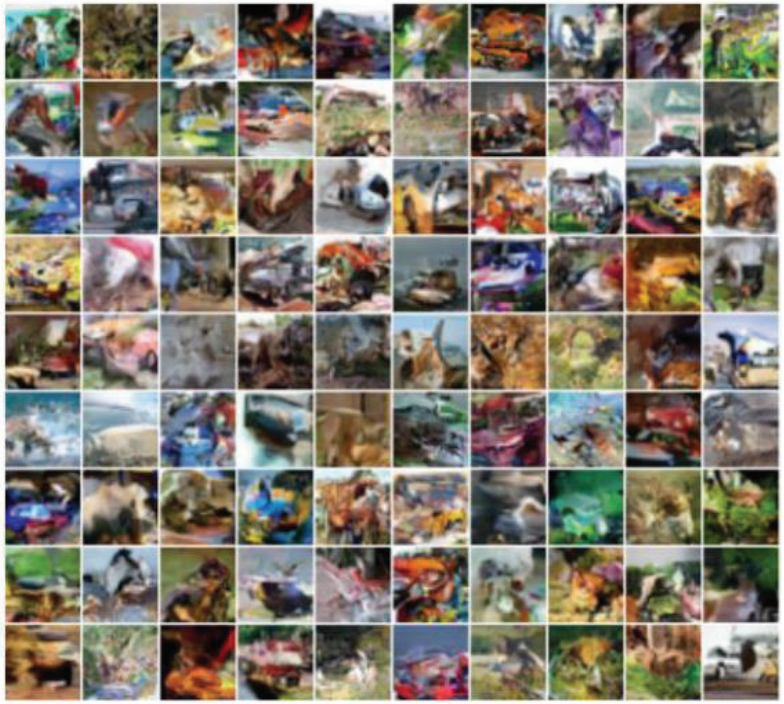
- Dependency on previous pixels now modeled using a CNN over context region



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

Generation must still proceed sequentially
=> still slow

Generation Samples



32x32 CIFAR-10

32x32 ImageNet

Generative Adversarial Networks (GAN)

Generative Adversarial Networks

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

- Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

- Q: What can we use to represent this complex transformation?

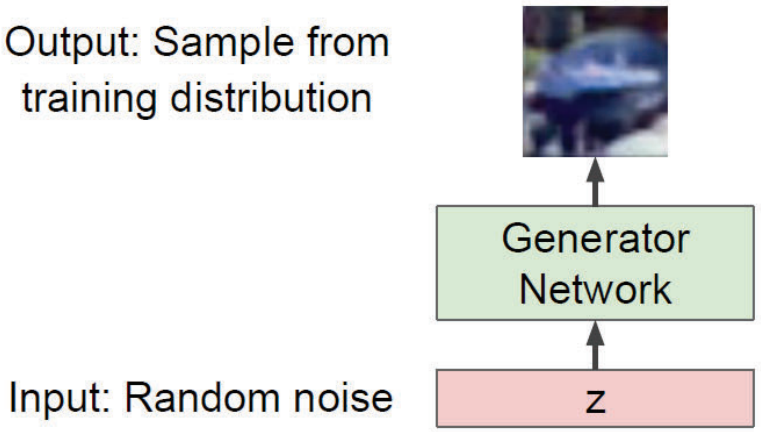
Generative Adversarial Networks

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

- Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

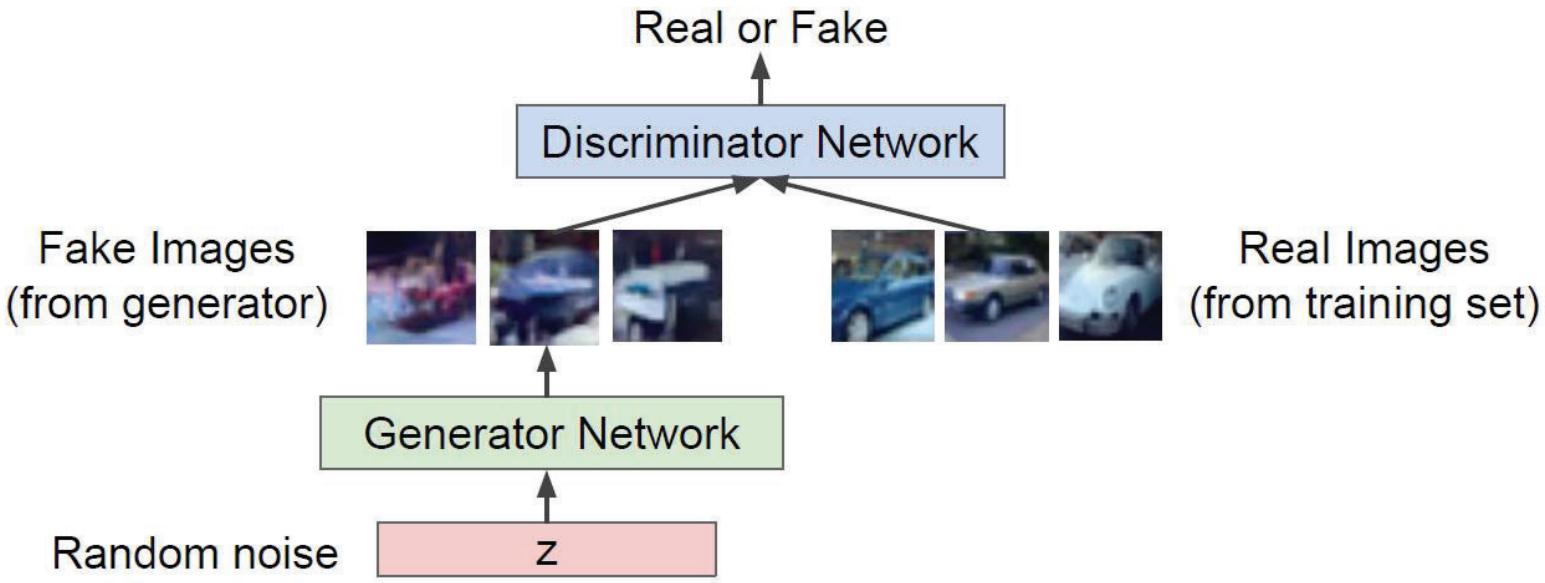
Q: What can we use to represent this complex transformation?

A: A neural network!



Training GANs: Two-player game

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



Training GANs: Two-player game

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

Train jointly in **minimax game**

Minimax objective function:

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

Training GANs: Two-player game

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

Train jointly in **minimax game**

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

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- 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

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

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:

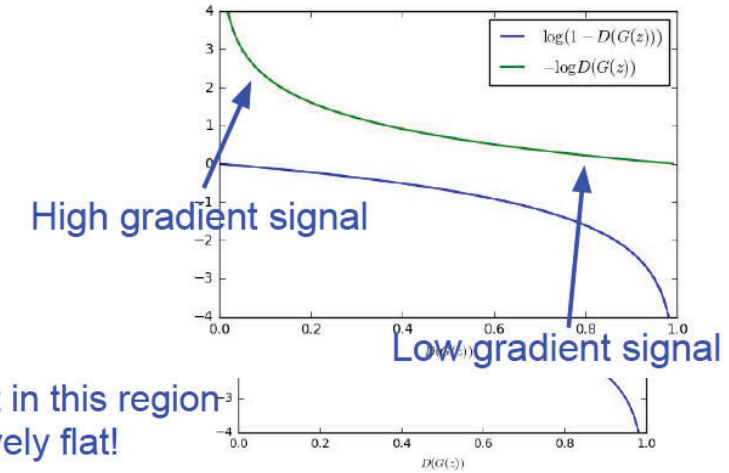
1. **Gradient ascent** on discriminator

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

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

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:

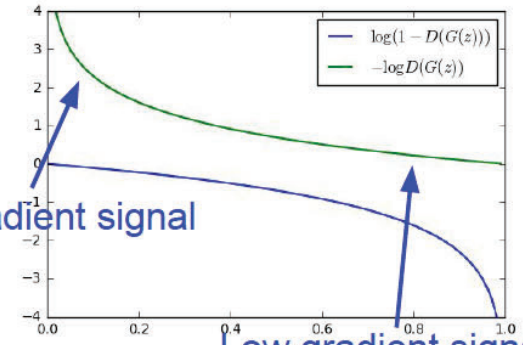
1. **Gradient ascent** on discriminator

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

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

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

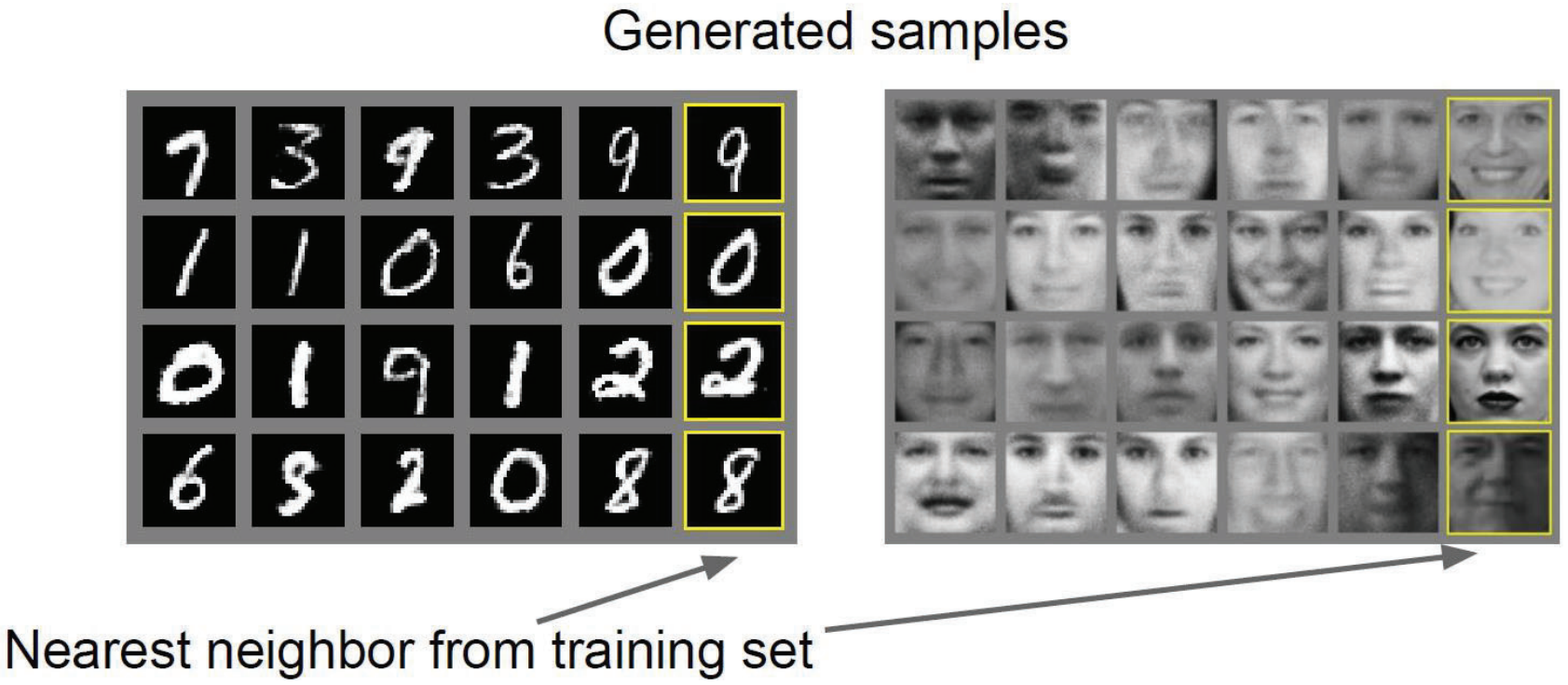
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by ascending its stochastic gradient (improved objective):

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

end for

Generative Adversarial Nets

[Ian Goodfellow et al. 14]



Generative Adversarial Nets: Convolutional Architectures

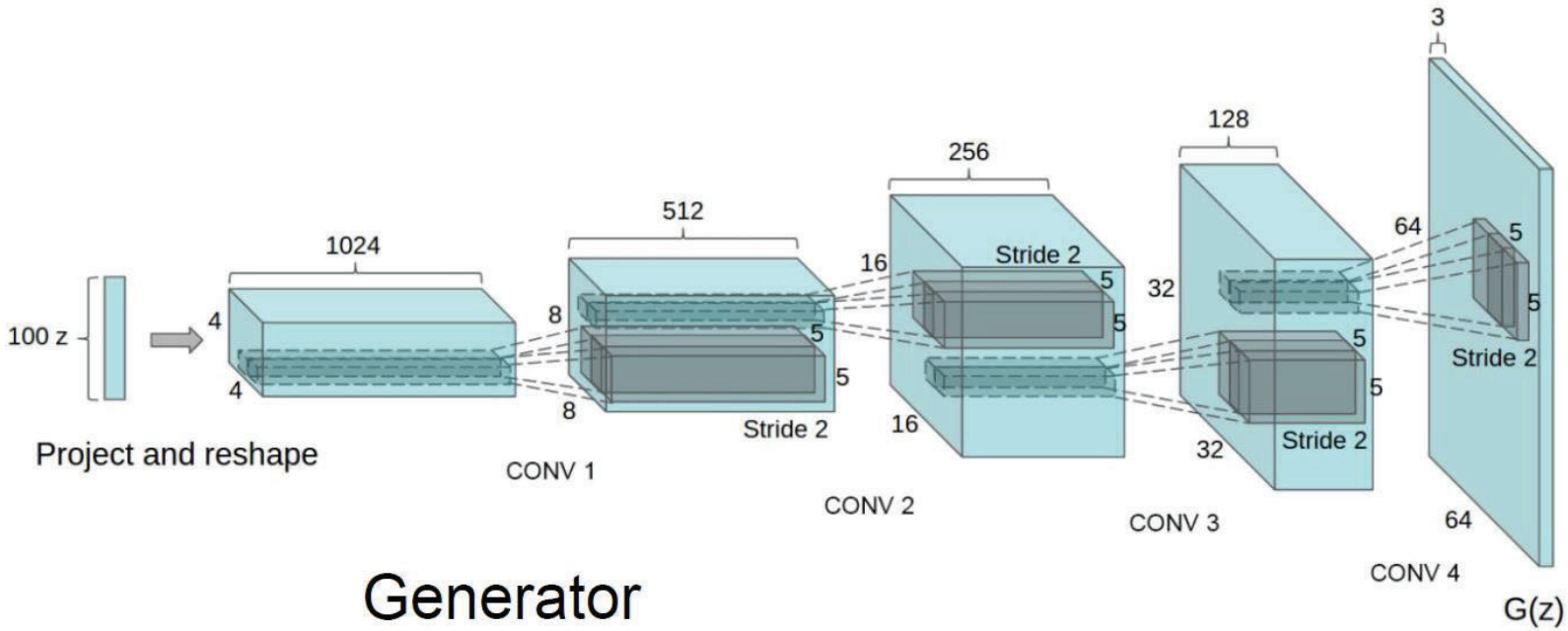
Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

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

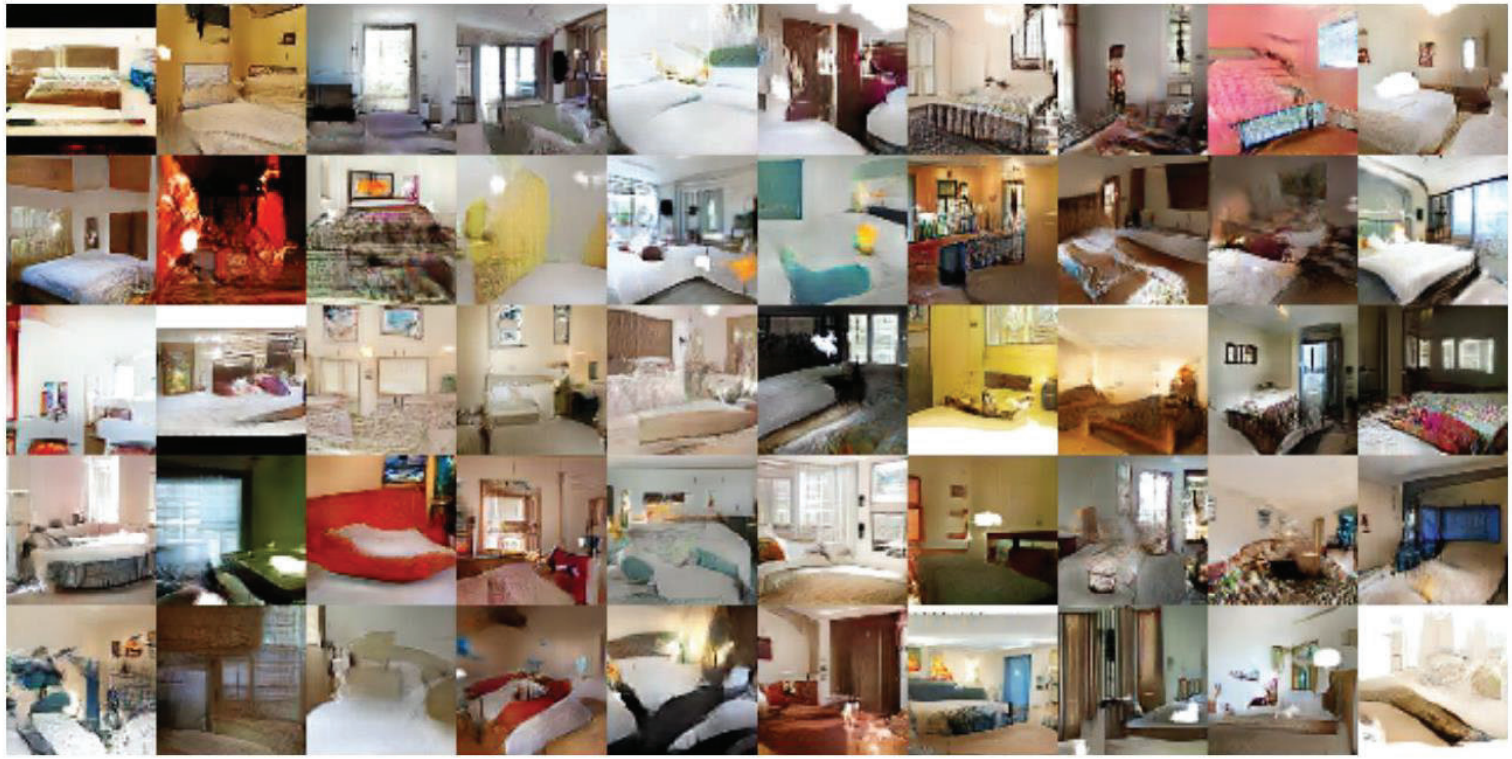
Generative Adversarial Nets: Convolutional Architectures



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

Generative Adversarial Nets: Convolutional Architectures

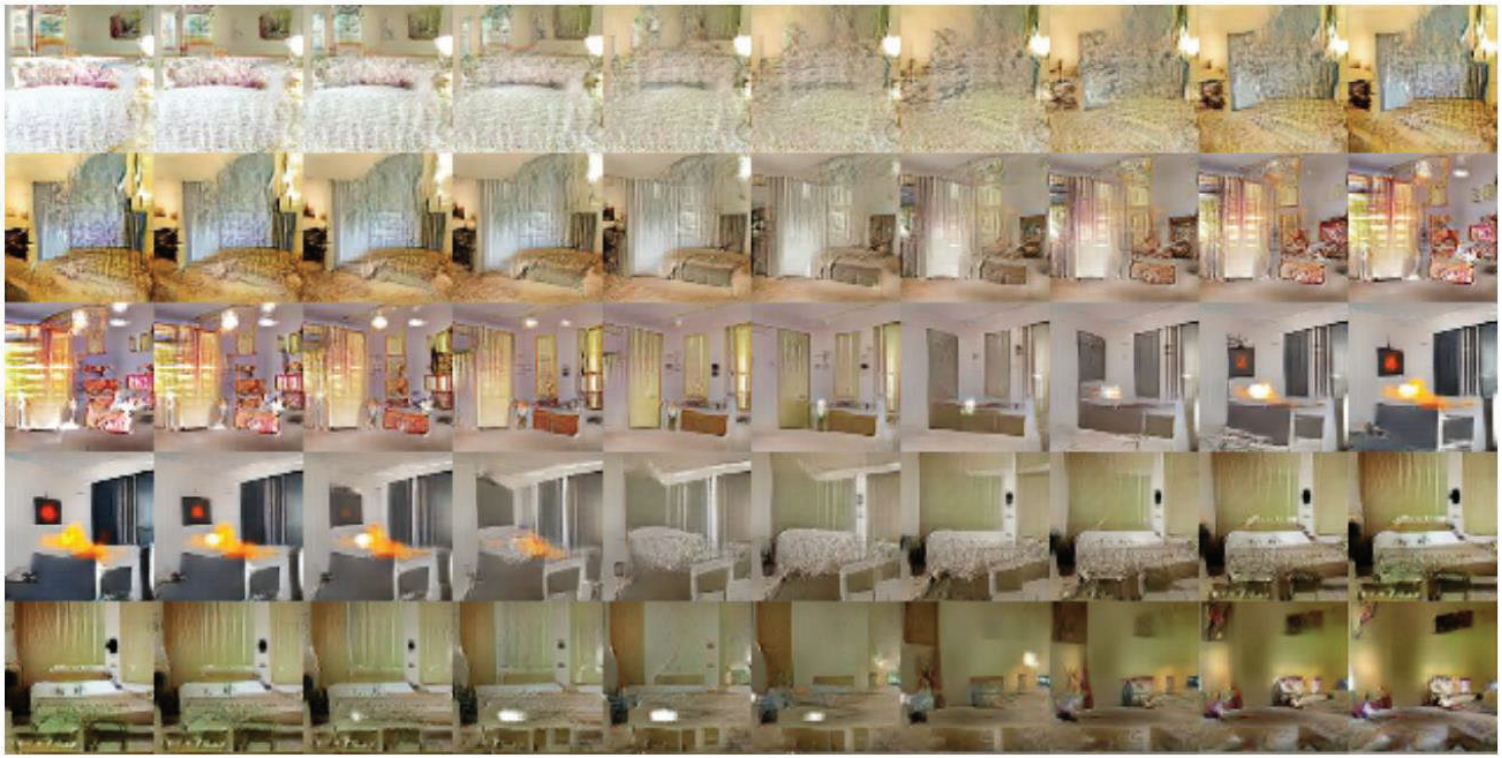
Samples from the model look much better!



Radford et al, ICLR 2016

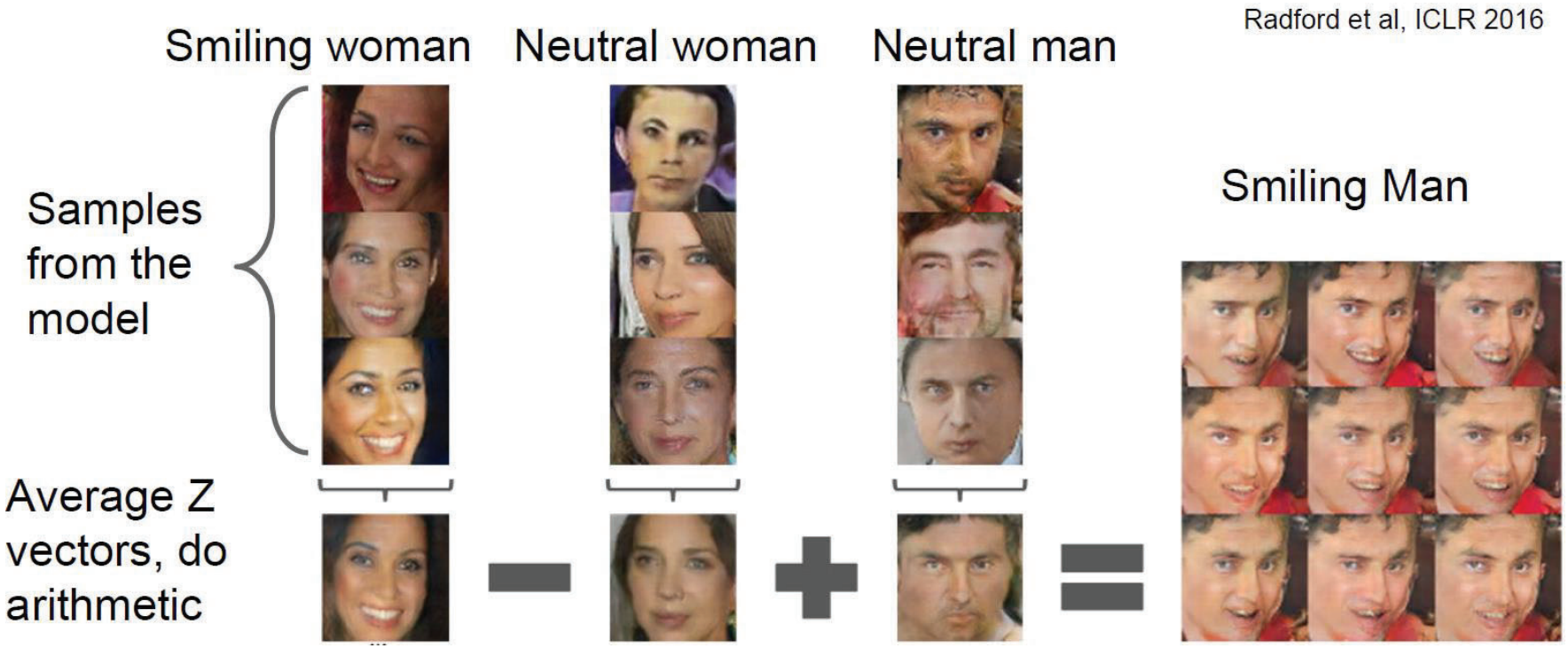
Generative Adversarial Nets: Convolutional Architectures

Interpolating between random points in laten space

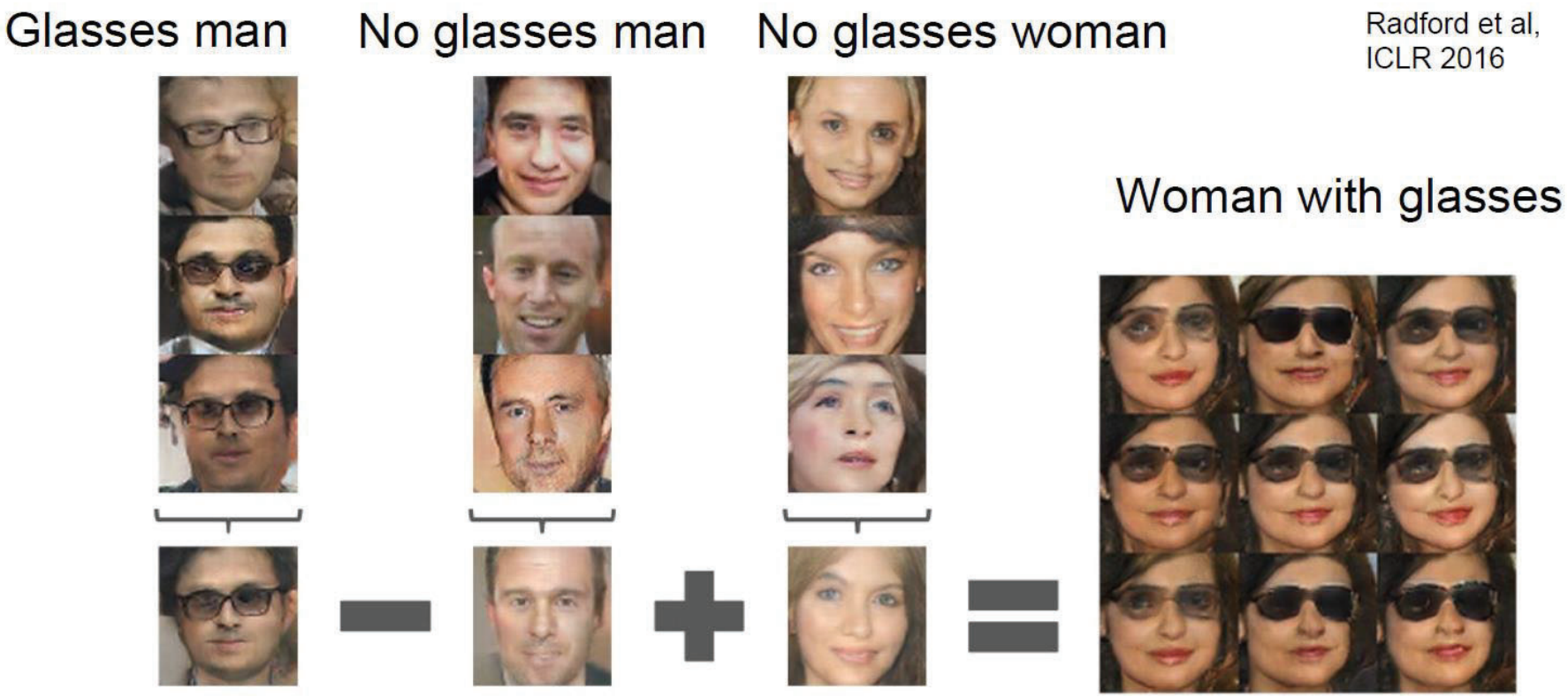


Radford et al, ICLR 2016

Generative Adversarial Nets: Interpretable Vector Math

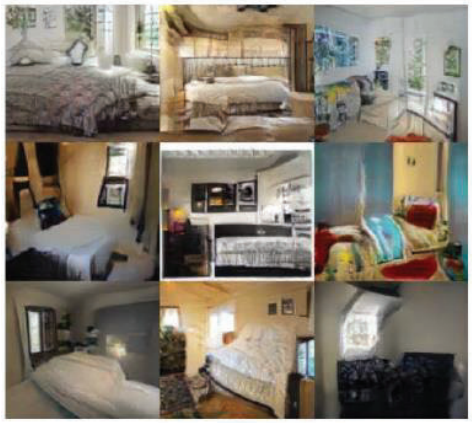


Generative Adversarial Nets: Interpretable Vector Math



Explosion of GANs

Better training and generation



LSGAN, Zhu 2017.

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

Progressive GAN, Karras 2018.

Variational Autoencoders (VAE)

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

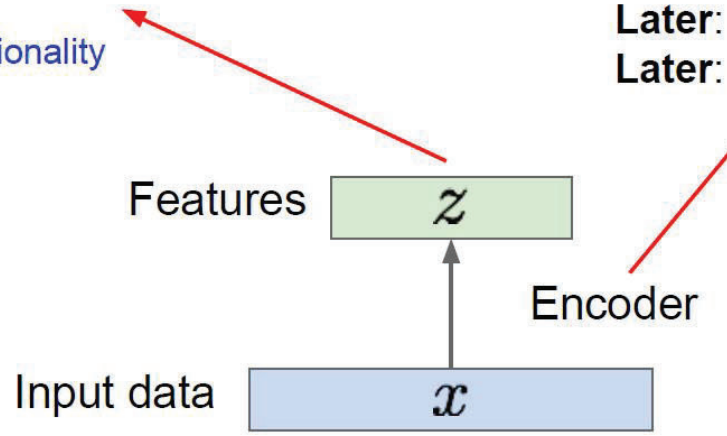
Some background first: Autoencoders

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

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

Q: Why dimensionality reduction?

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

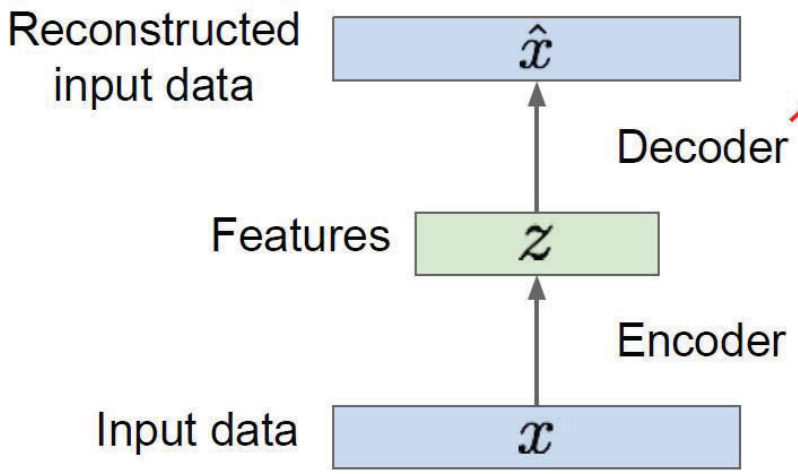


Some background first: Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data

"Autoencoding" - encoding itself



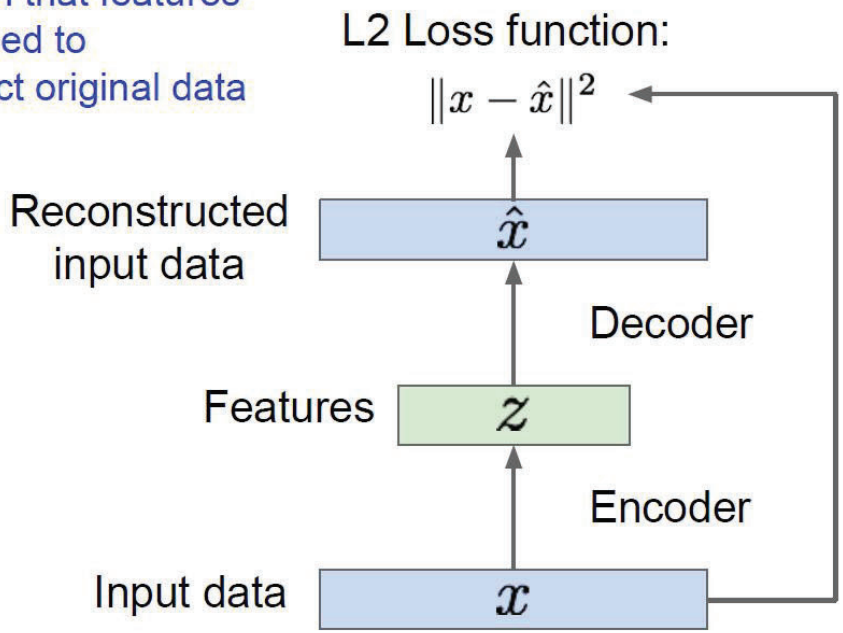
Originally: Linear + nonlinearity (sigmoid)

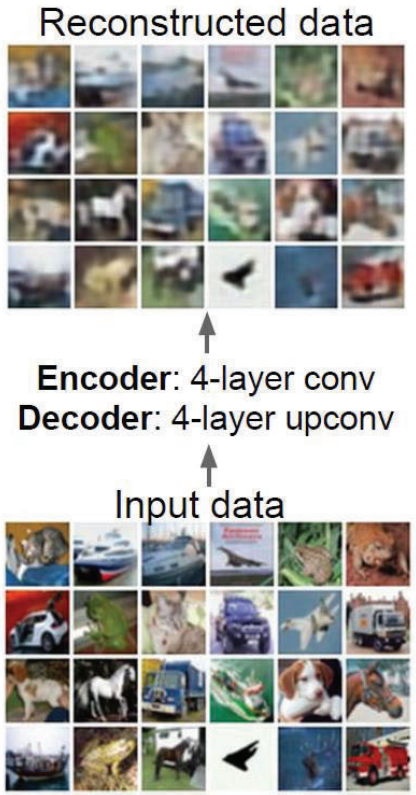
Later: Deep, fully-connected

Later: ReLU CNN (upconv)

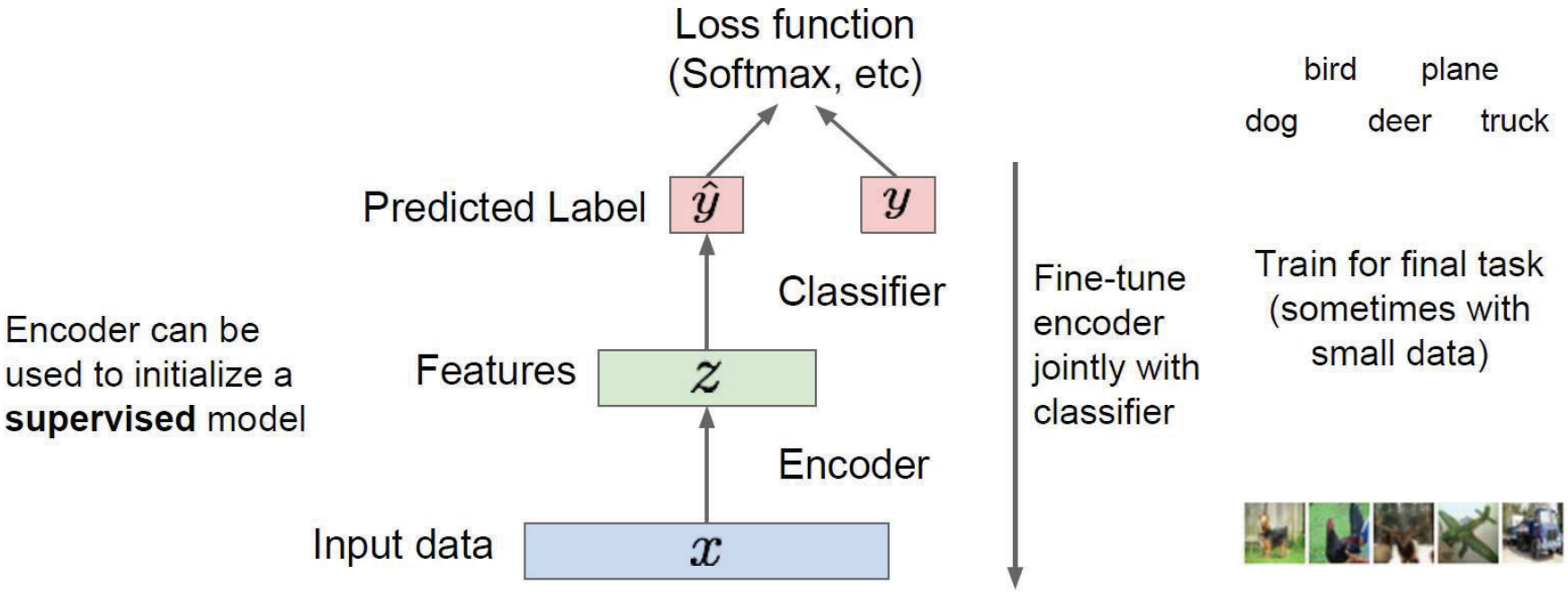
Some background first: Autoencoders

Train such that features can be used to reconstruct original data

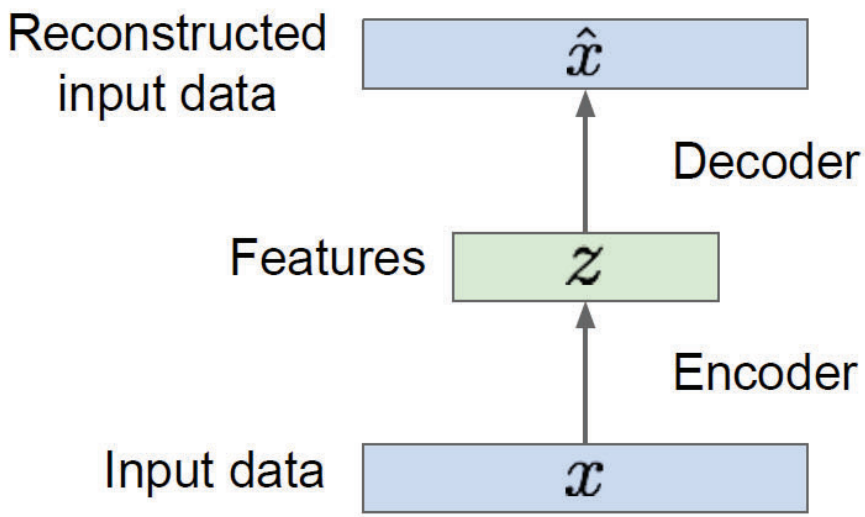




Some background first: Autoencoders



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. Can we generate new images from an autoencoder?

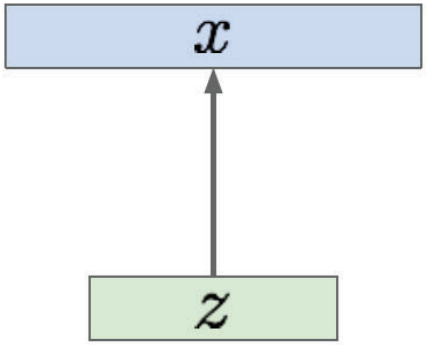
Variational Autoencoders

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 underlying unobserved (latent) representation **z**

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

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



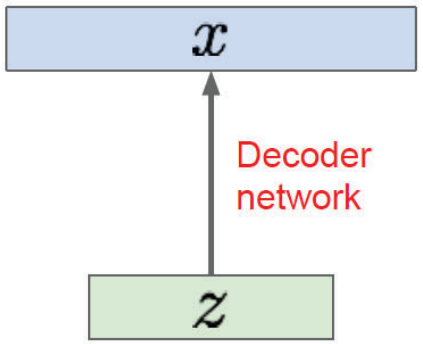
Intuition (remember from autoencoders!): **x** is an image, **z** is latent factors used to generate **x:** attributes, orientation, etc.

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

Variational Autoencoders

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

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



We want to estimate the true parameters θ^* of this generative model.

How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian.

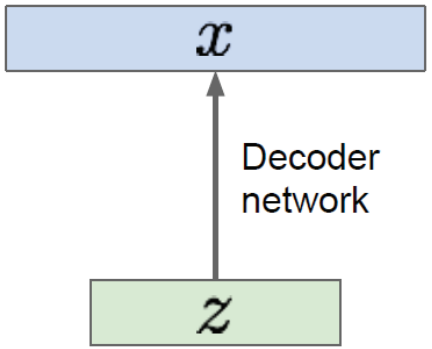
Conditional p(x|z) is complex (generates image) => represent with neural network

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



We want to estimate the true parameters θ^* of this generative model.

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: Intractability

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

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

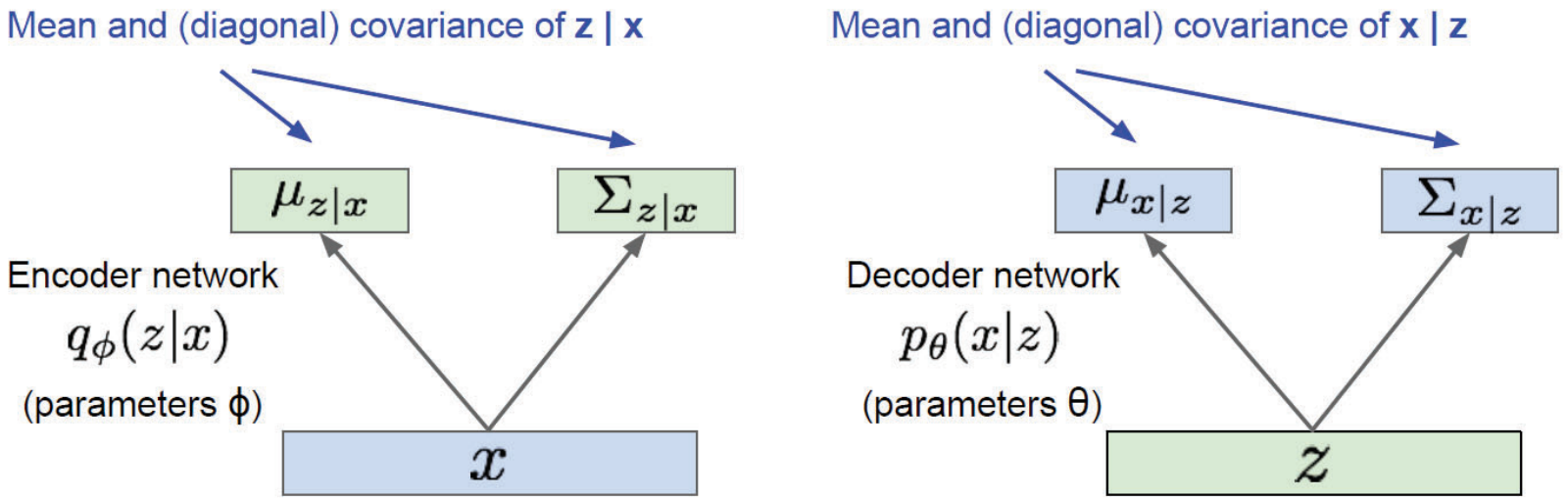
Solution: In addition to decoder network modeling $p_\theta(x|z)$, define additional encoder network $q_\phi(z|x)$ that approximates $p_\theta(z|x)$

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

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

Variational Autoencoders

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic



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

Next Lecture: Continue VAE