Generative Models and GANs

Yuan YAO
HKUST
Based on Feifei Li, Tong Zhang, et al.

Supervised vs Unsupervised Learning

Supervised Learning

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

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

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

Unsupervised Learning

Data: x
Just data, no labels!

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

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

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 $p_{data}(x)$

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 $p_{data}(x)$

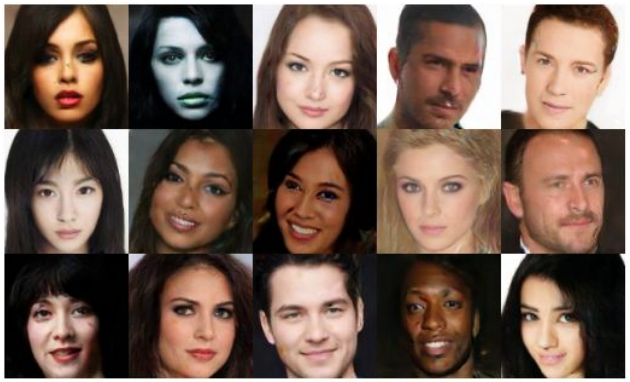
Addresses density estimation, a core problem in unsupervised learning

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

Why Generative Models?

- Realistic samples for artwork, super-resolution, colorization, etc.



- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
- Training generative models can also enable inference of latent representations that can be useful as general features

Taxonomy of Generative Models

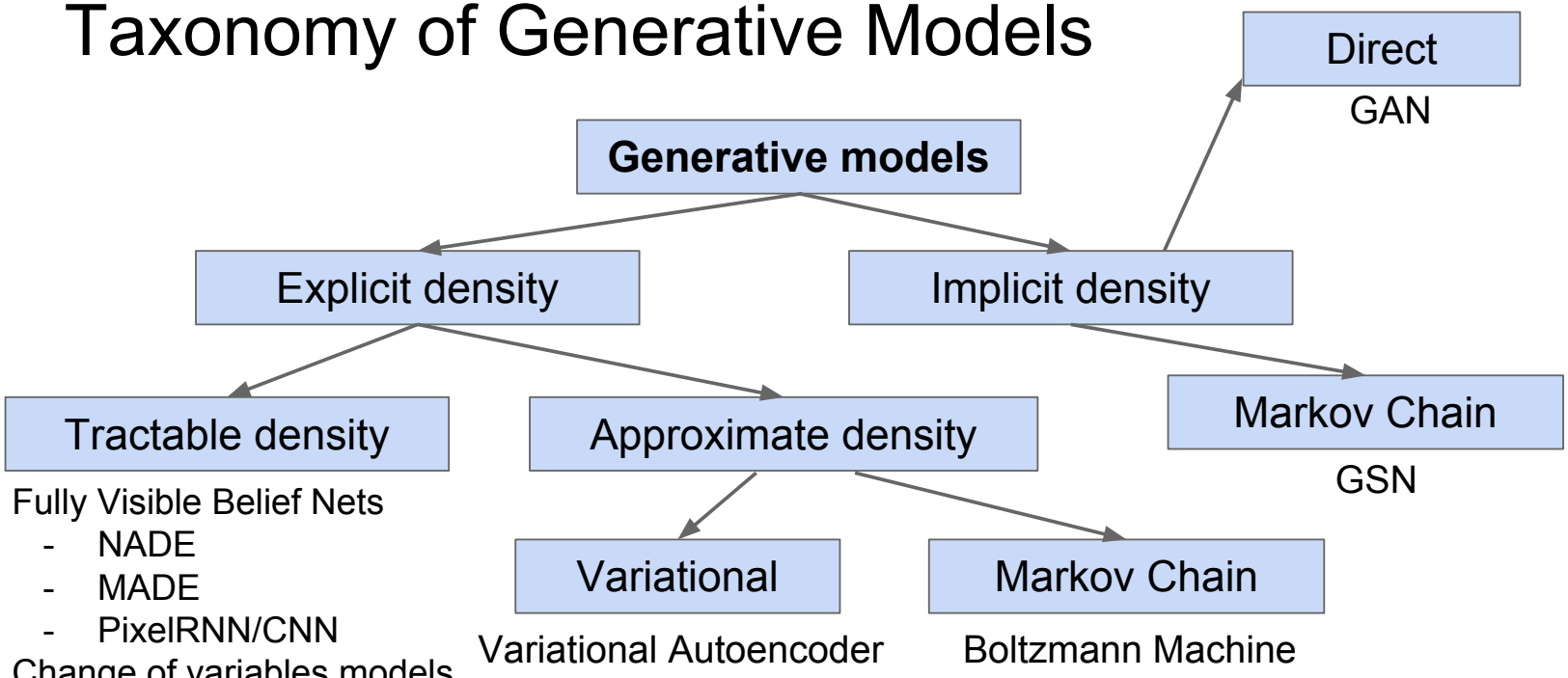


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

Today: discuss 3 most popular types of generative models today

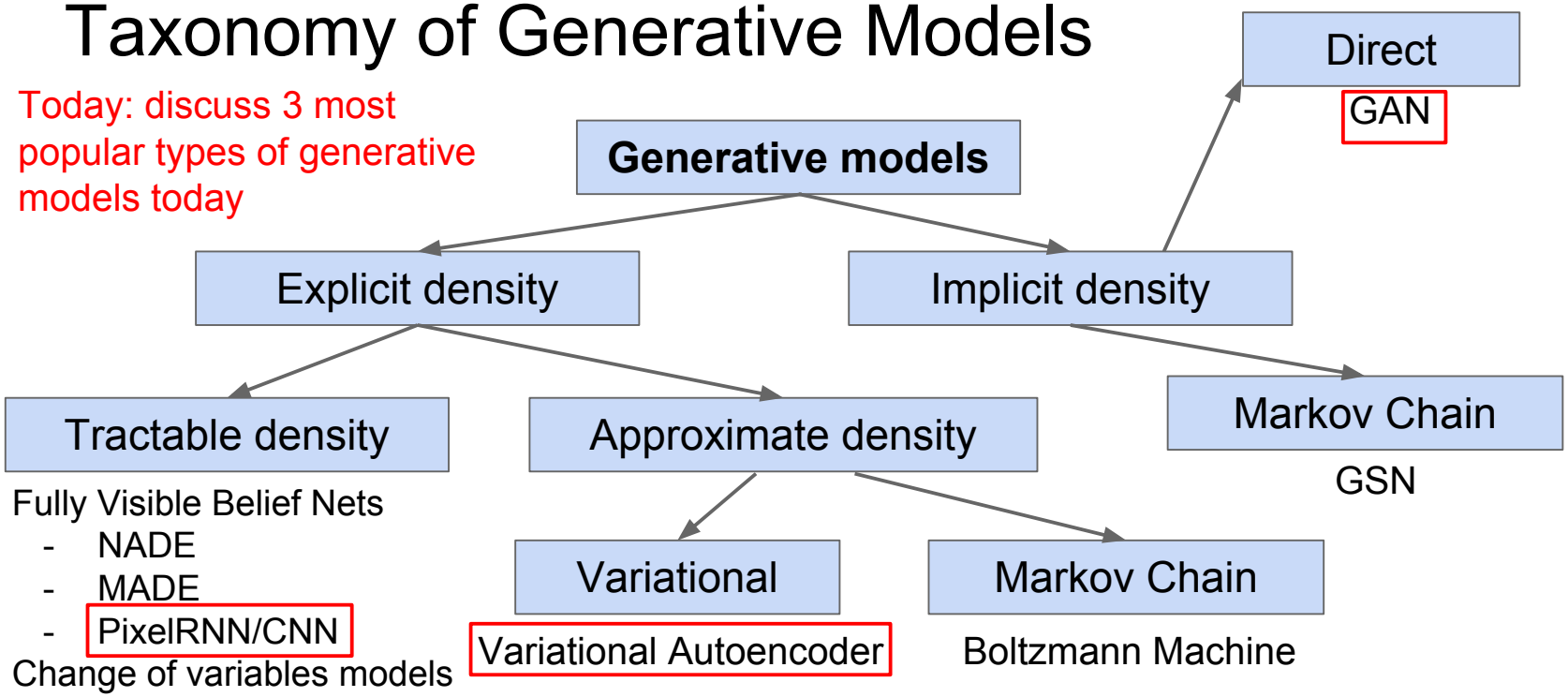


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

PixelRNN and Pixel CNN

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

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

Will need to define ordering of "previous pixels"

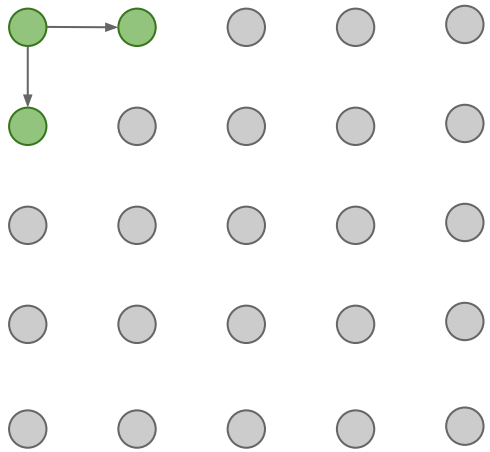
Then maximize likelihood of training data

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

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

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

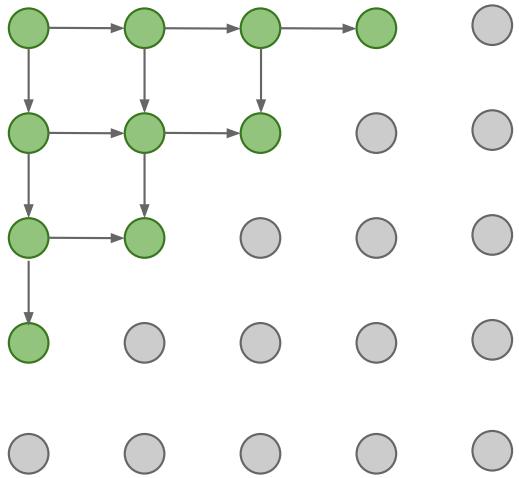


PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow!



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

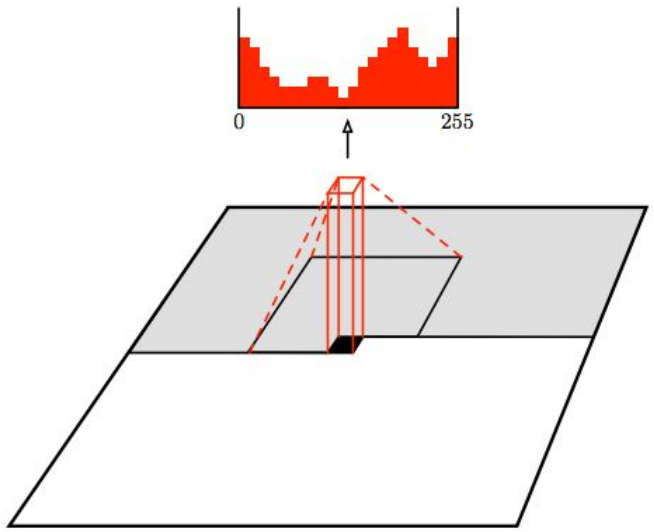
Still generate image pixels starting from corner

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

Training: maximize likelihood of training images

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

Softmax loss at each pixel



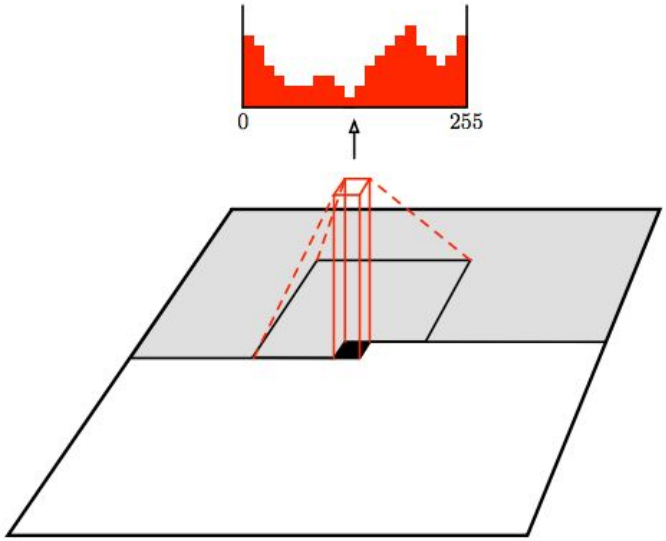
PixelCNN *[van der Oord et al. 2016]*

Still generate image pixels starting from corner

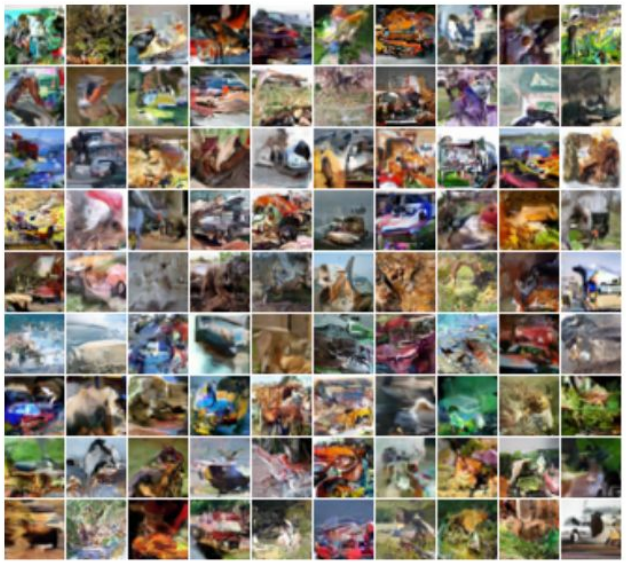
Dependency on previous pixels now modeled using a CNN over context region

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

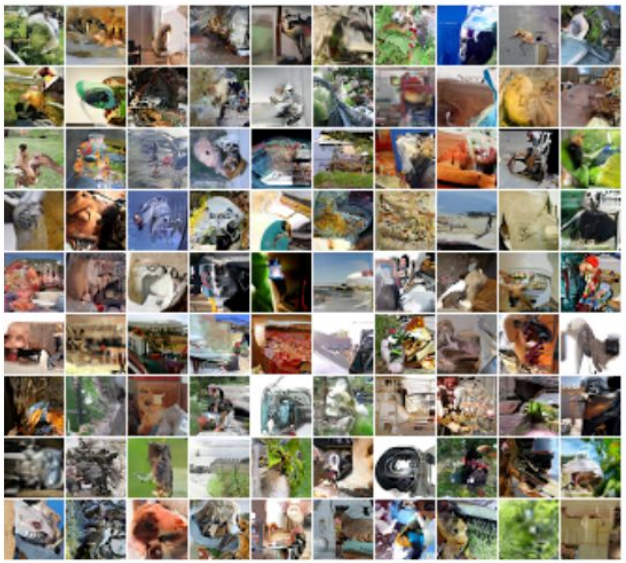
Generation must still proceed sequentially
=> still slow



Generation Samples



32x32 CIFAR-10

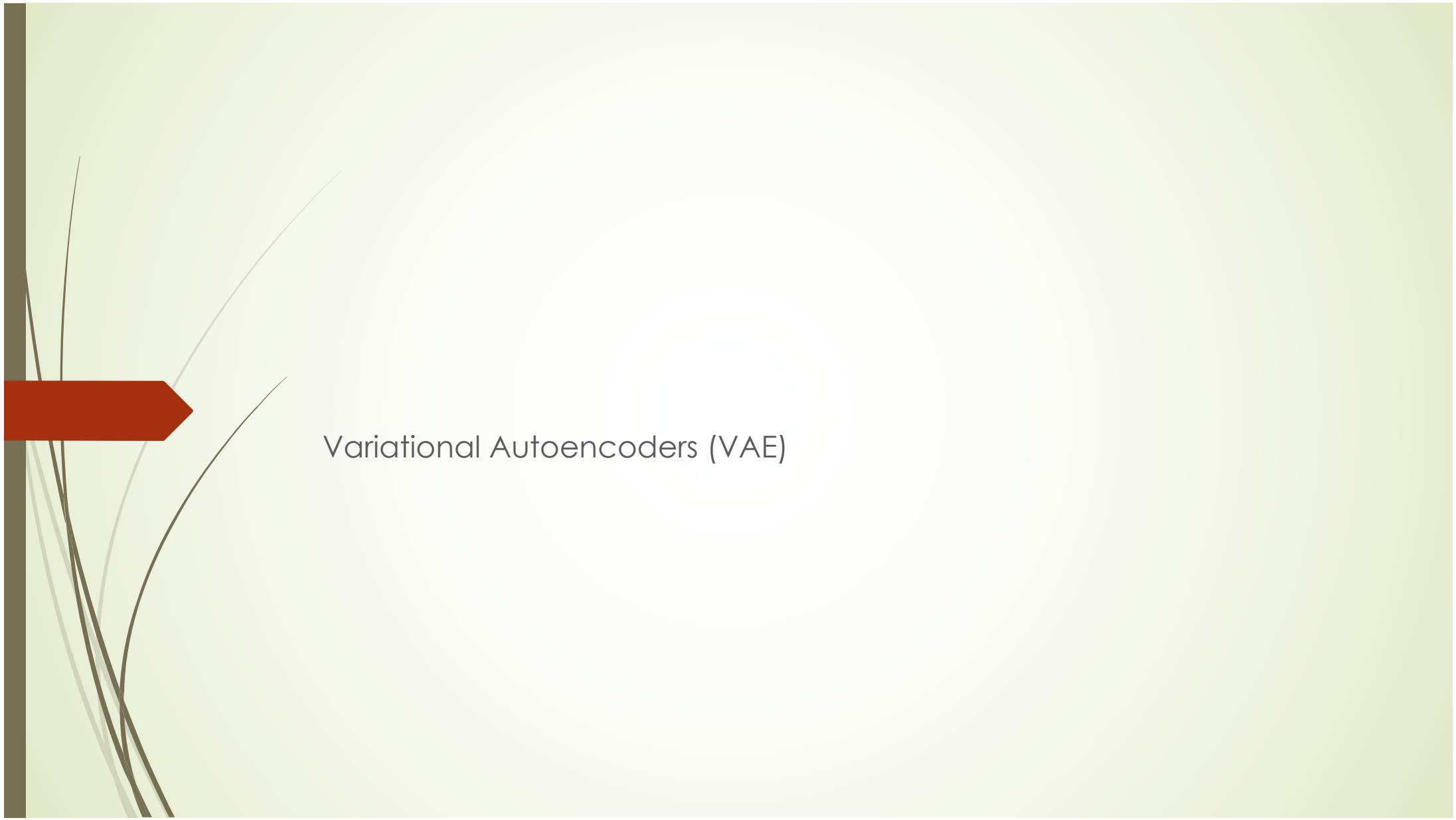


32x32 ImageNet

PixelRNN and 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++)


Variational Autoencoders (VAE)

From Likelihood to Mixture Likelihood

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**:

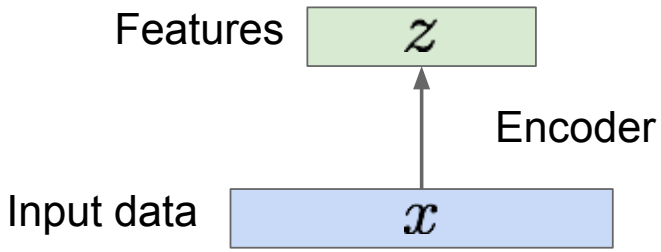
$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

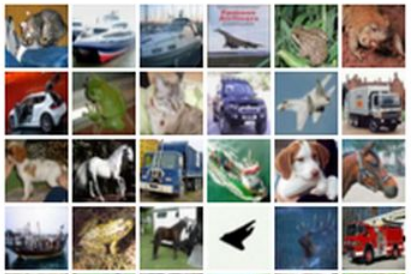
Cannot optimize directly, derive and optimize lower bound on likelihood instead

Some background first: Autoencoders

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

e.g. PCA, Manifold Learning, Dictionary Learning

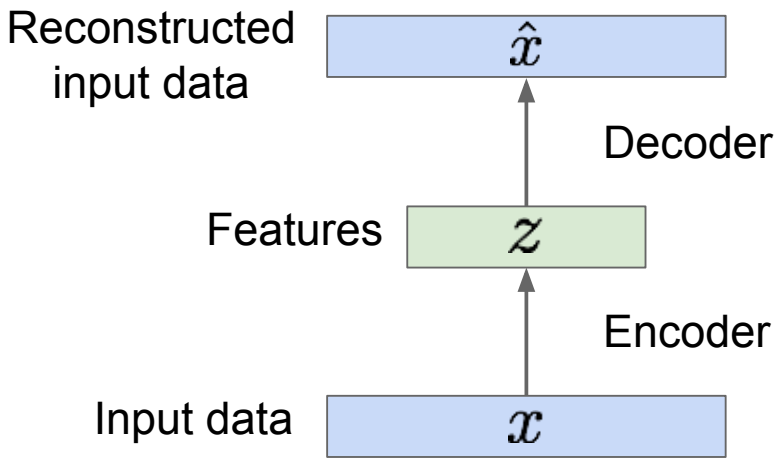




How to learn this feature representation?

Train such that features can be used to reconstruct original data

"Autoencoding" - encoding itself



e.g. PCA, Manifold Learning, Dictionary Learning, Matrix Factorization: D = E'

Deep Learning for encoders

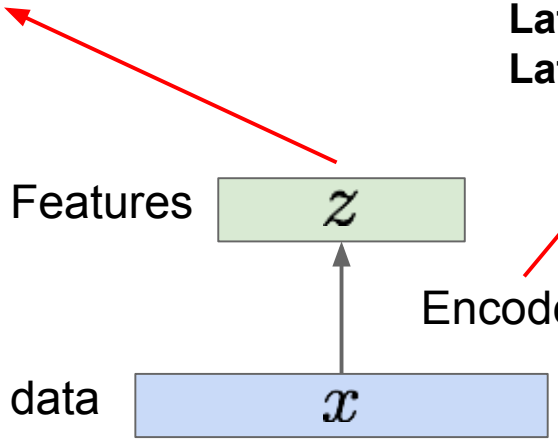
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

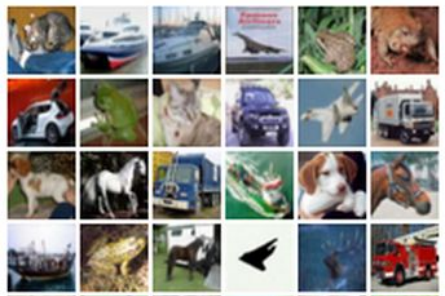
z usually smaller than **x** (dimensionality reduction)

Q: Why dimensionality reduction?

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

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



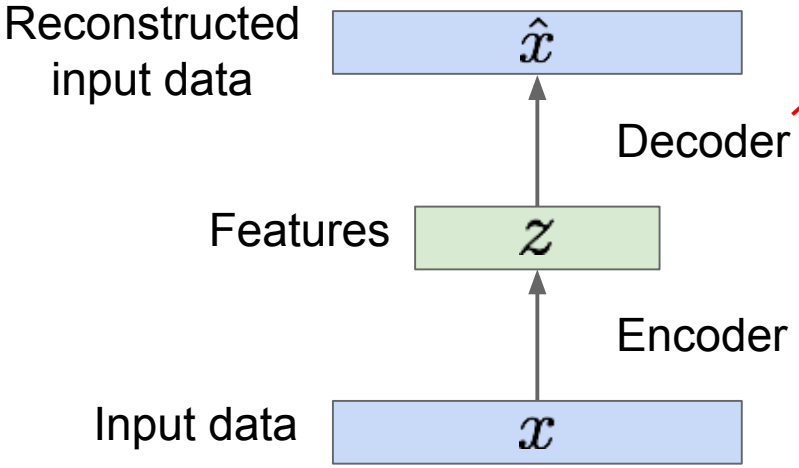


Deep Learning for decoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data

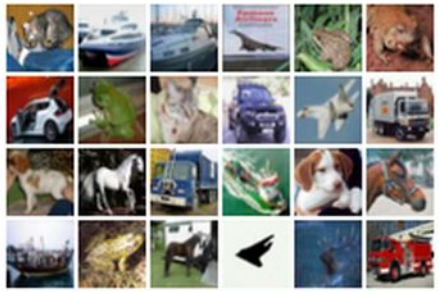
“Autoencoding” - encoding itself



Originally: Linear + nonlinearity (sigmoid)

Later: Deep, fully-connected

Later: ReLU CNN (upconv)

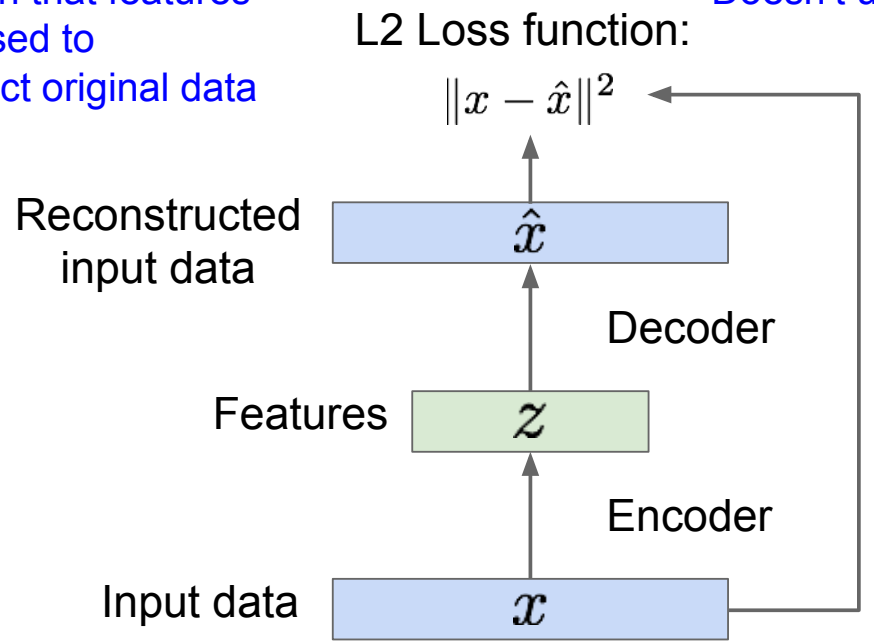


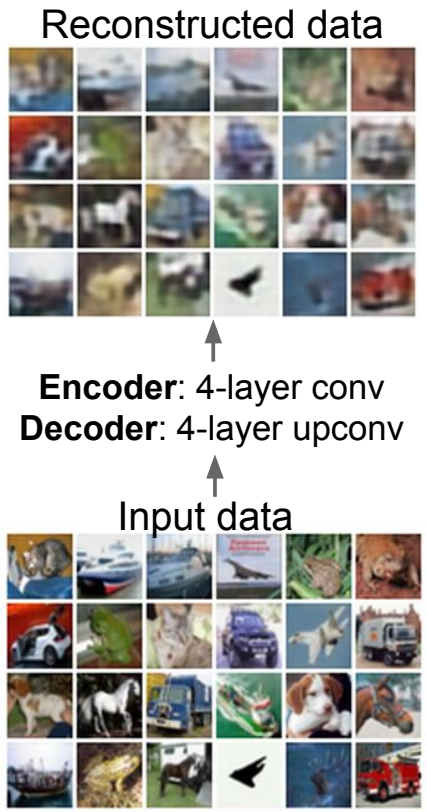
L2 Loss functions

Some background first: Autoencoders

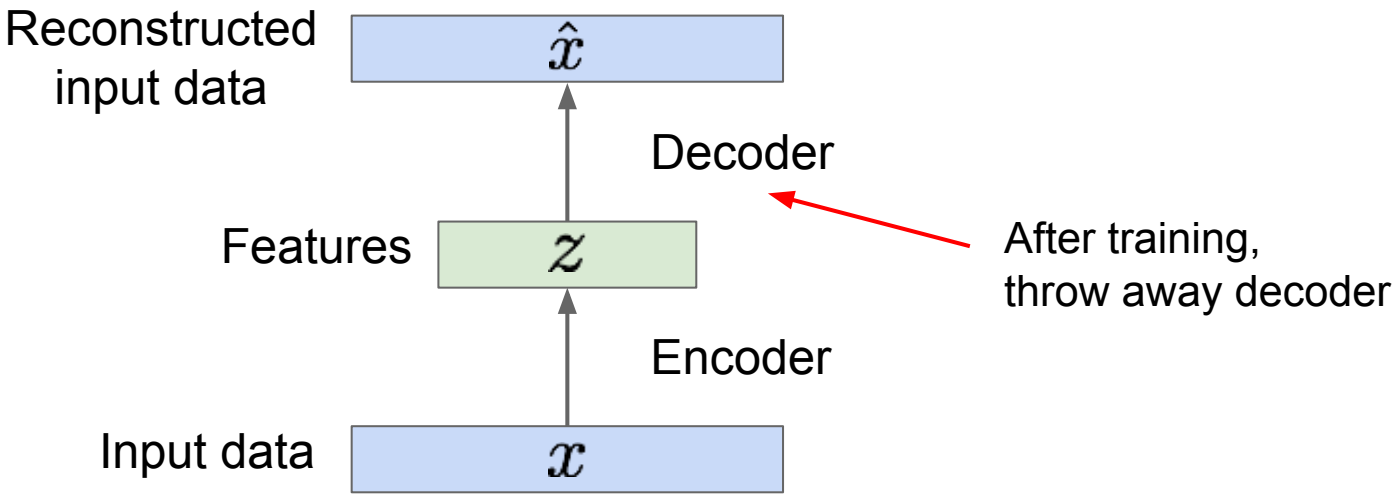
Train such that features can be used to reconstruct original data

Doesn't use labels!

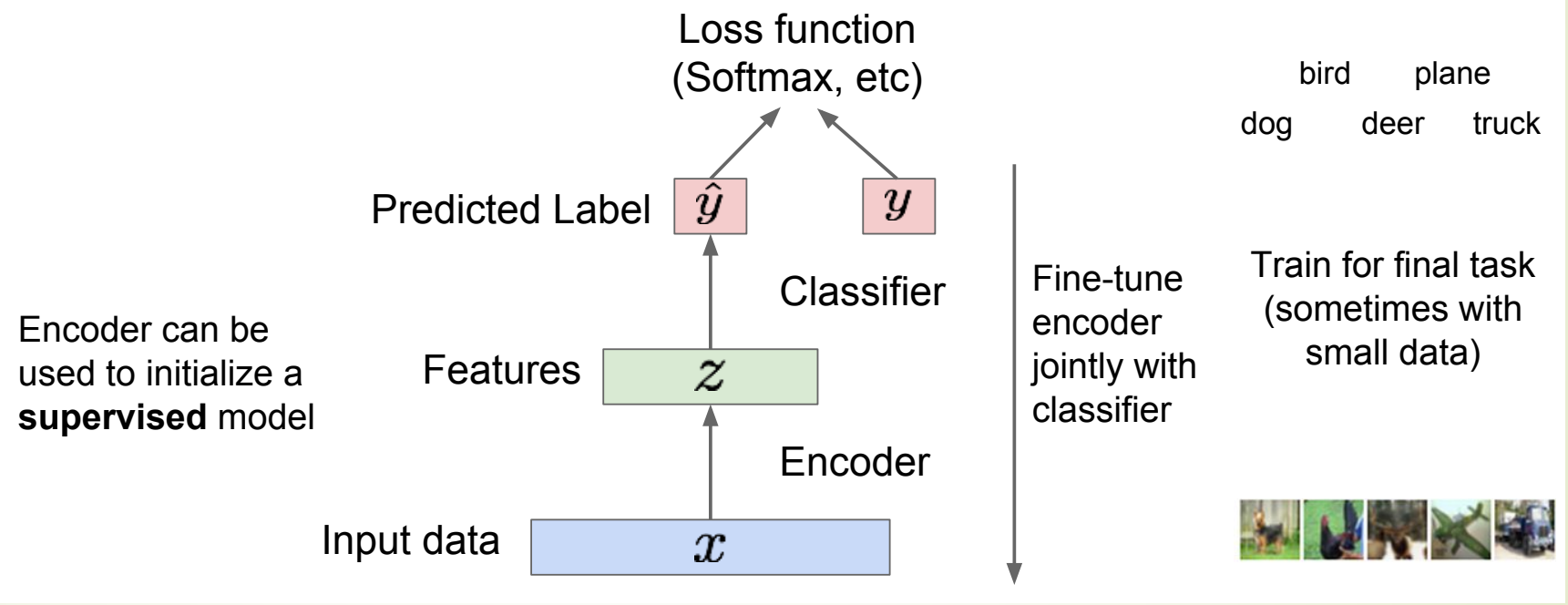


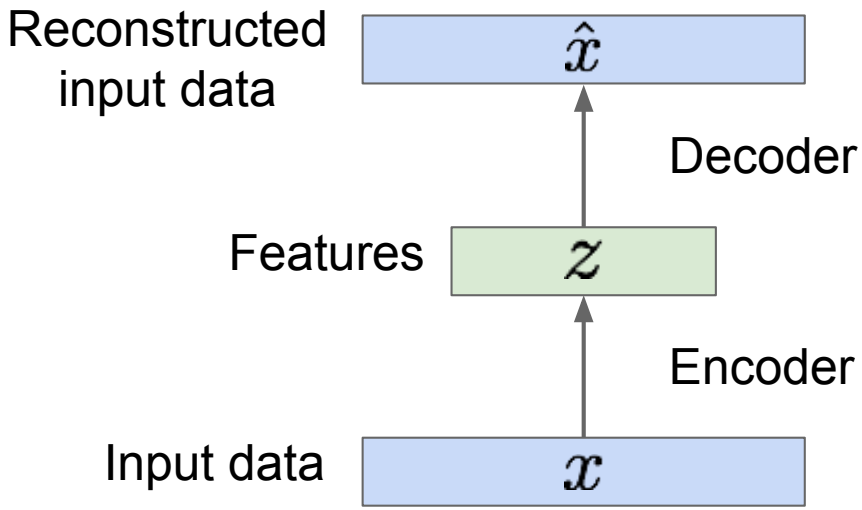


Some background first: Autoencoders



Autoencoders for Transfer Learning





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?

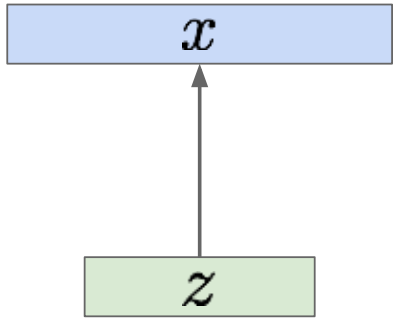
Variational Autoencoders

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



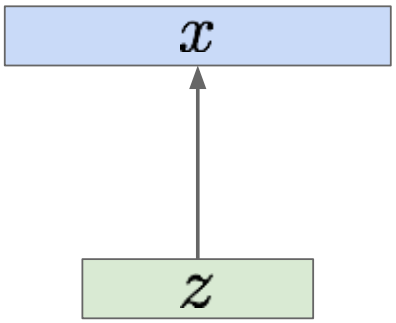
Intuition (remember from autoencoders!): **x** is an image, **z** is latent factors used to generate **x:** attributes, orientation, etc.

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?

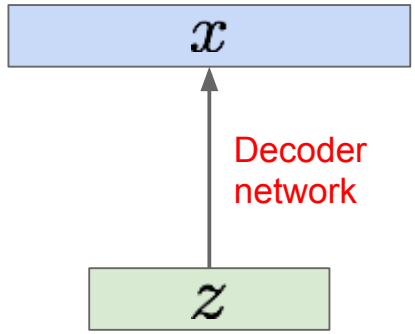
Choose prior p(z) to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

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

Variational Autoencoders

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

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



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.

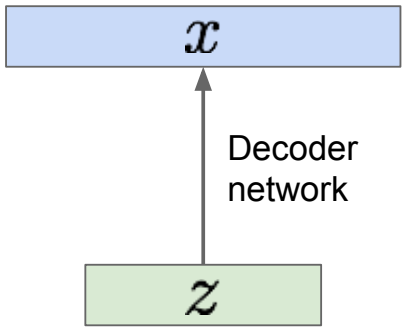
Conditional p(x|z) is complex (generates image) => represent with neural network

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?

Remember strategy for training generative models from FVBNs. Learn model parameters to maximize likelihood of training data

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

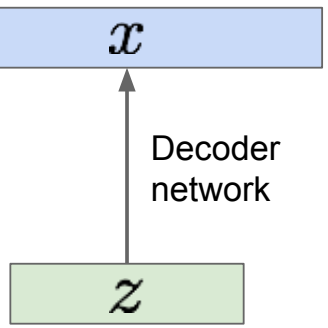
Now with latent z

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

Variational Autoencoders

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

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



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

How to train the model?

Remember strategy for training generative models from FVBNs. Learn model parameters to maximize likelihood of training data

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

Q: What is the problem with this?

Intractable!

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

Variational Autoencoders: Intractability

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

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

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

Intractable data likelihood

Variational Lower Bounds

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 $p_\theta(z|x)$

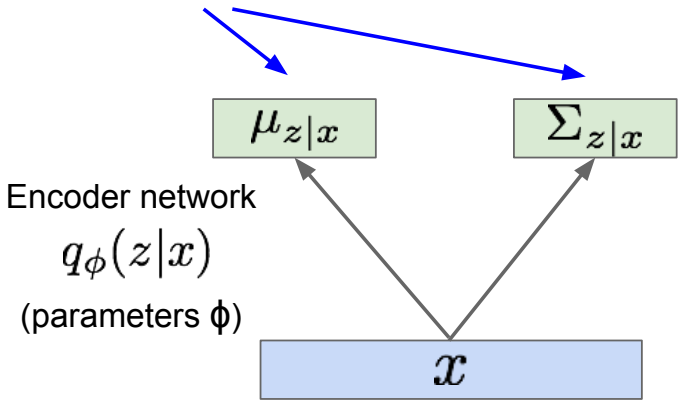
Will see that this allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize

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

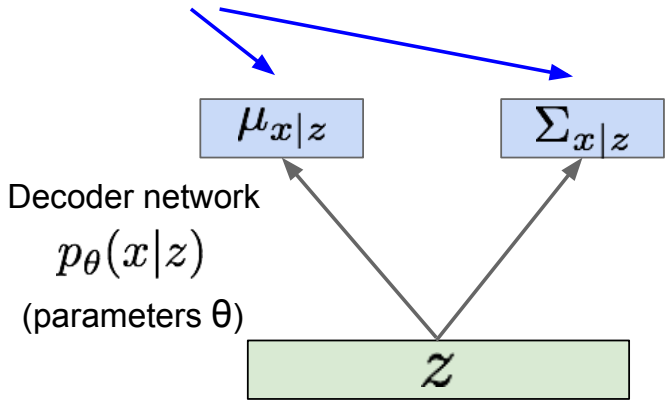
Variational Autoencoders

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

Mean and (diagonal) covariance of **z | x**



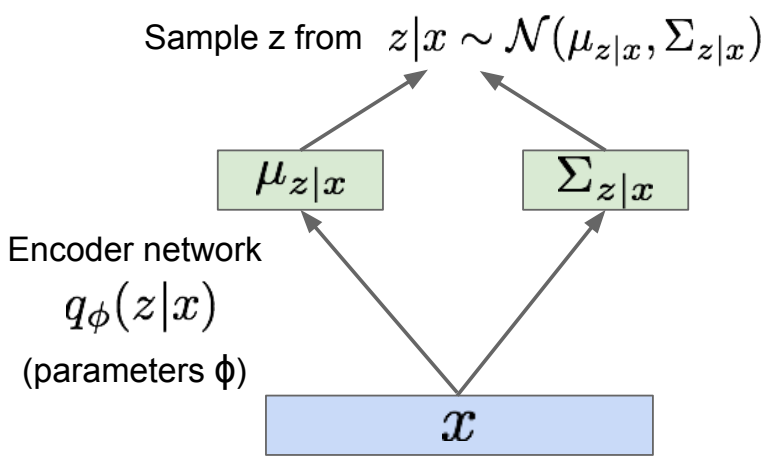
Mean and (diagonal) covariance of **x | z**

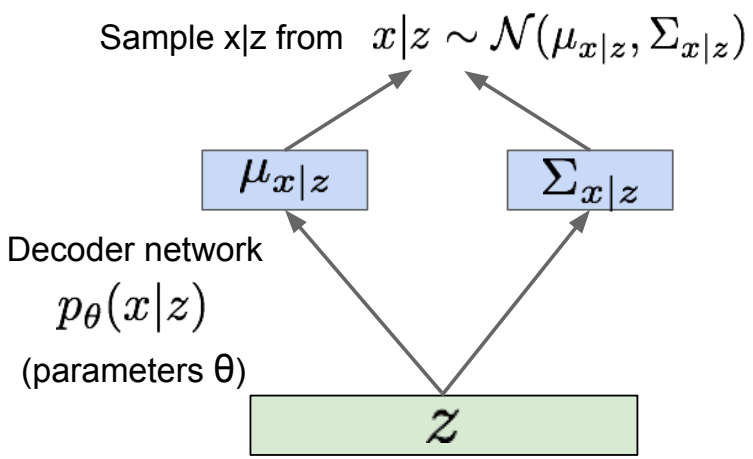


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

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





Encoder and decoder networks also called "recognition"/"inference" and "generation" networks

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

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

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

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

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

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

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

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

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)

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

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

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

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

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

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

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

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

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

Reconstruct the input data

Make approximate posterior distribution close to prior

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

Variational lower bound ("ELBO")

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: Maximize lower bound

Stage I in Forward Pass

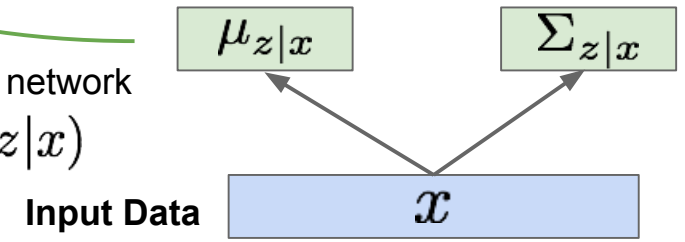
Putting it all together: maximizing the likelihood lower bound

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

Make approximate posterior distribution close to prior

Encoder network

$q_\phi(z|x)$

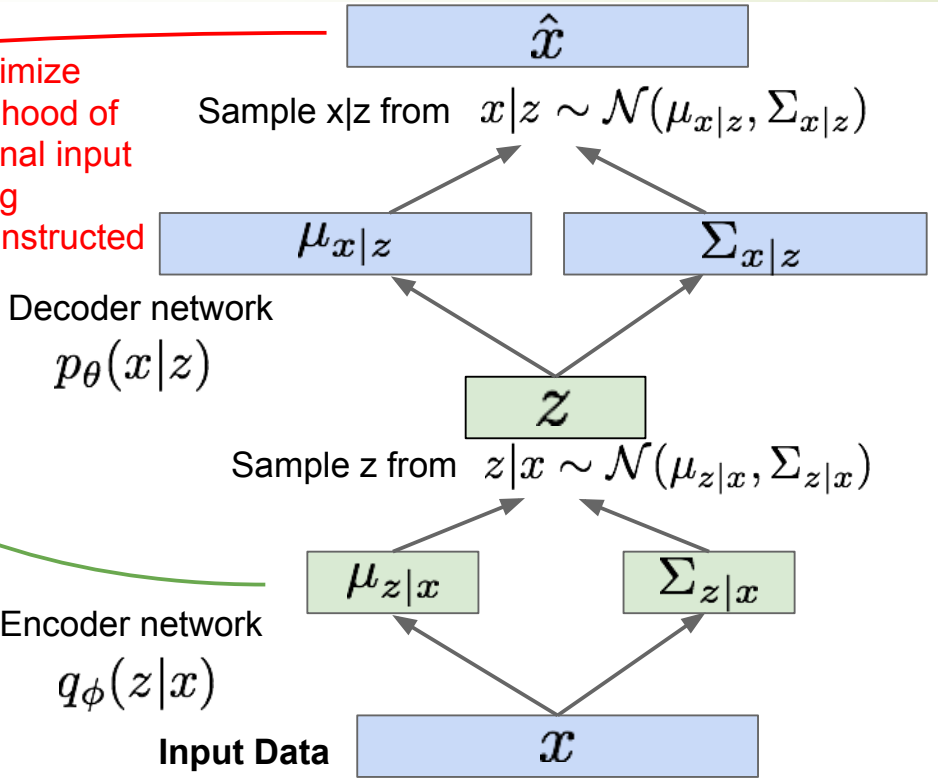


Stage II in forward pass

Putting it all together: maximizing the likelihood lower bound

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

Maximize likelihood of original input being reconstructed



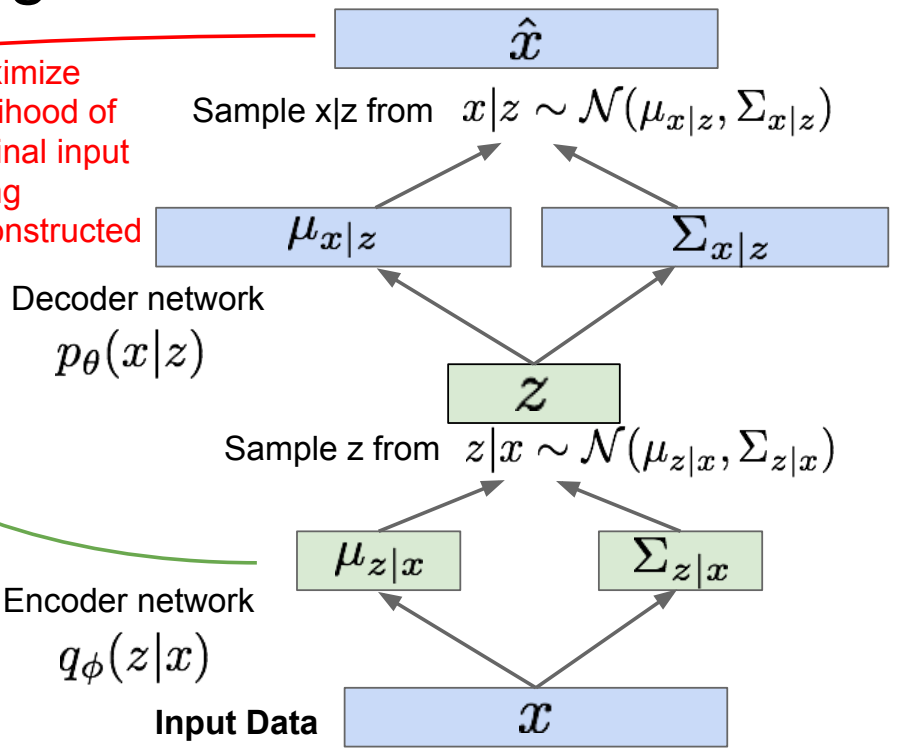
Make approximate posterior distribution close to prior

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Maximize likelihood of original input being reconstructed

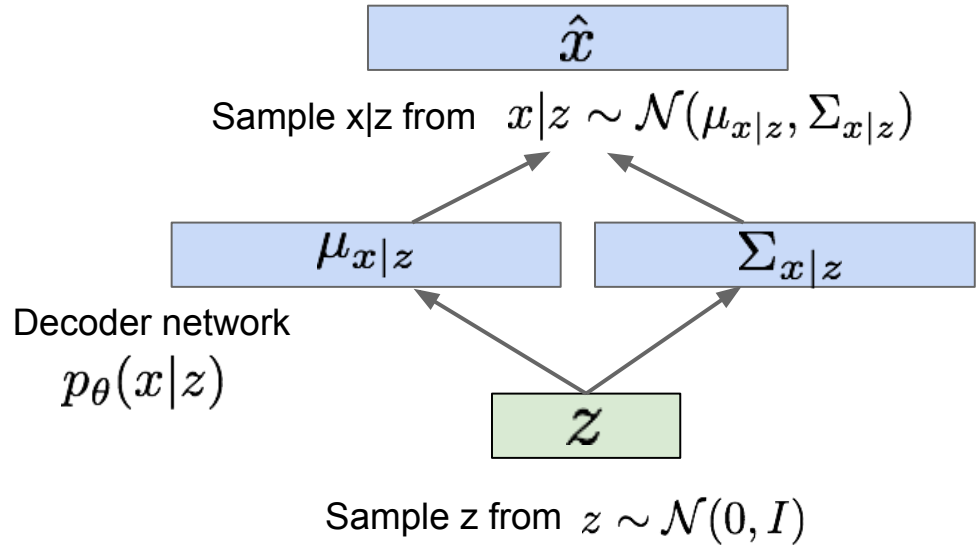


Make approximate posterior distribution close to prior

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

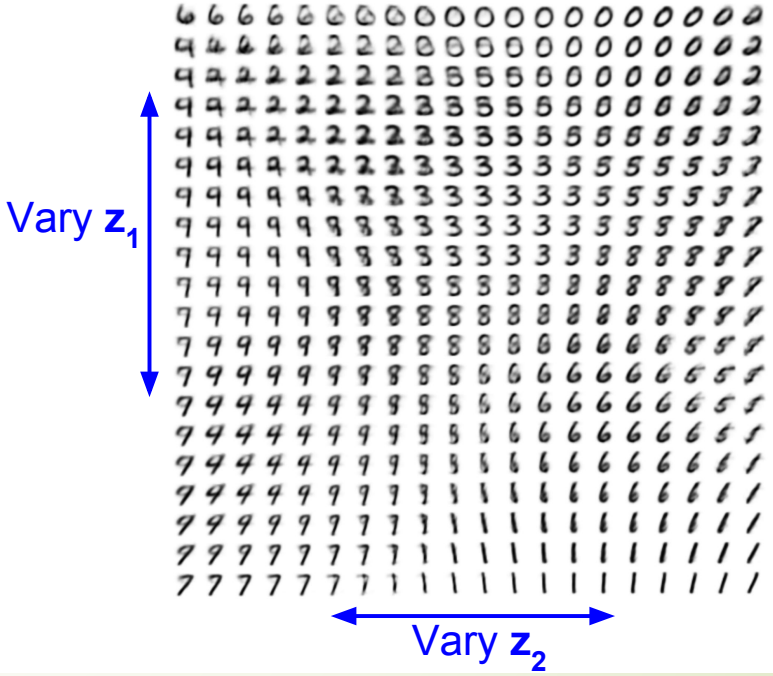
VAE: generating data

Use decoder network. Now sample z from prior!



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

Data manifold for 2-d **z**



VAE: generating data

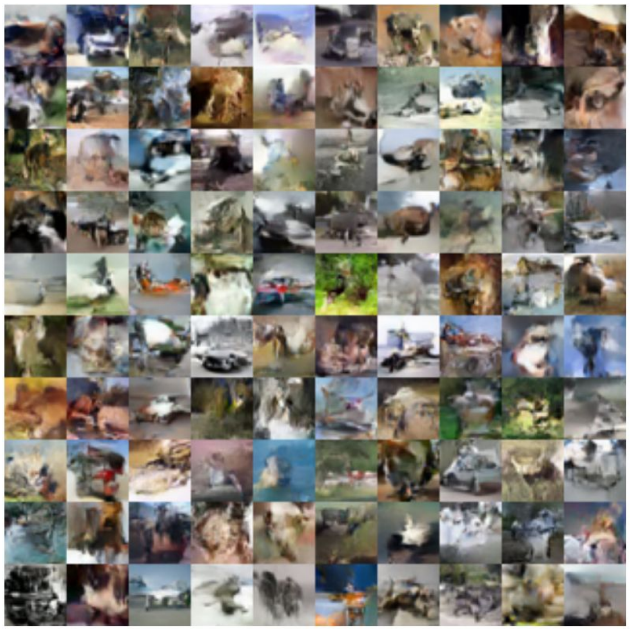
Diagonal prior on **z**
=> independent
latent variables

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

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

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

VAE: Generating Data



32x32 CIFAR-10

Labeled Faces in the Wild

Variational Autoencoders

- Probabilistic spin to traditional autoencoders => allows generating data
  Defines an intractable density => derive and optimize a (variational) lower bound
- Pros:
  - Principled approach to generative models
  - 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
  - Incorporating structure in latent variables

Generative Adversarial Networks (GAN)

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

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

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

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

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

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

GANs: don't work with any explicit density function!
Instead, take game-theoretic approach: learn to generate from training distribution through 2-player game

Generative Adversarial Networks

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

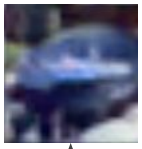
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

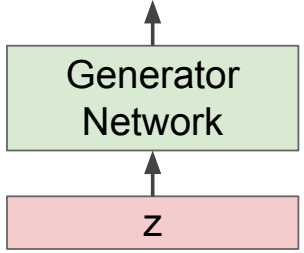
Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

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

A: A neural network!

Output: Sample from training distribution



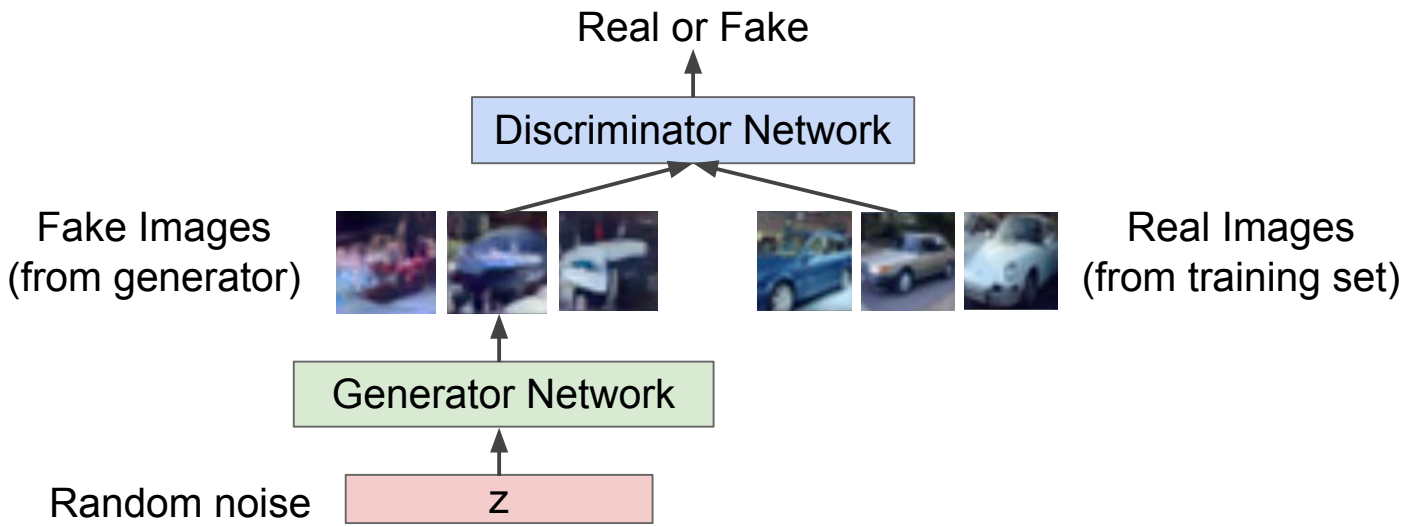


Input: Random noise

Training GANs: Two-player game

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

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



Why Early Stopping?

- GD asymptotically converges to minimal norm least square solution
- The minimal norm least square may overfit the training data if it is noisy
- Can we use early stopping as a regularization?
  - Yes, early stopping plays the same role as Ridge regression (Tikhonov regularization) [Yao-Rosasco-Caponetto'2005]

Training GANs: Minimax 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: Minimax 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:

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

Discriminator output for real data x

Discriminator output for generated fake data G(z)

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

Training GANs

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

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

The Issue in Training GANs

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

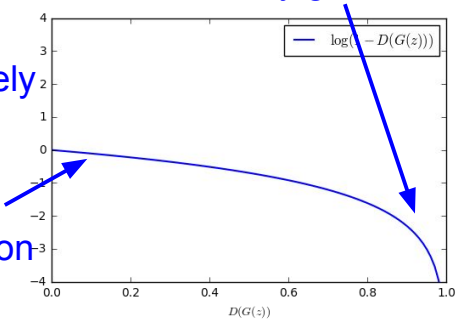
2. **Gradient descent** on generator

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

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

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



The Log D trick

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:

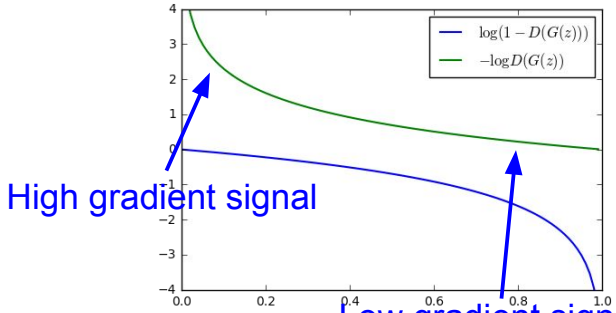
1. **Gradient ascent** on discriminator

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

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.



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

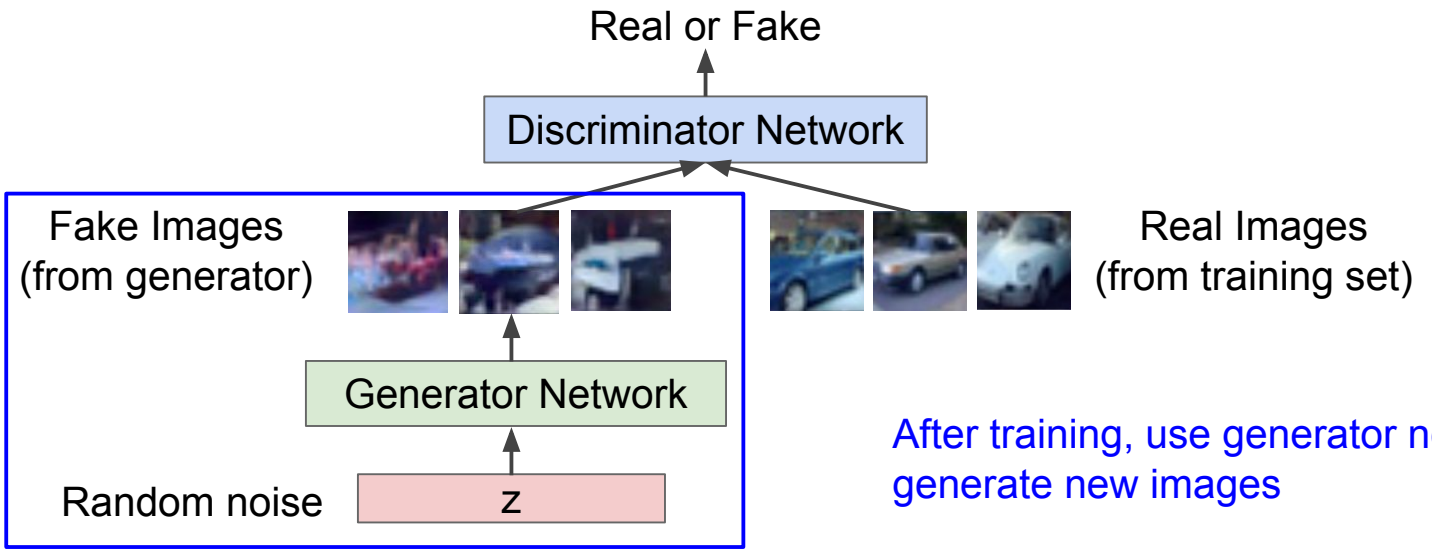
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

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

end for

Other Losses (Wasserstein Distance, KL-divergence) are better in stability!

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

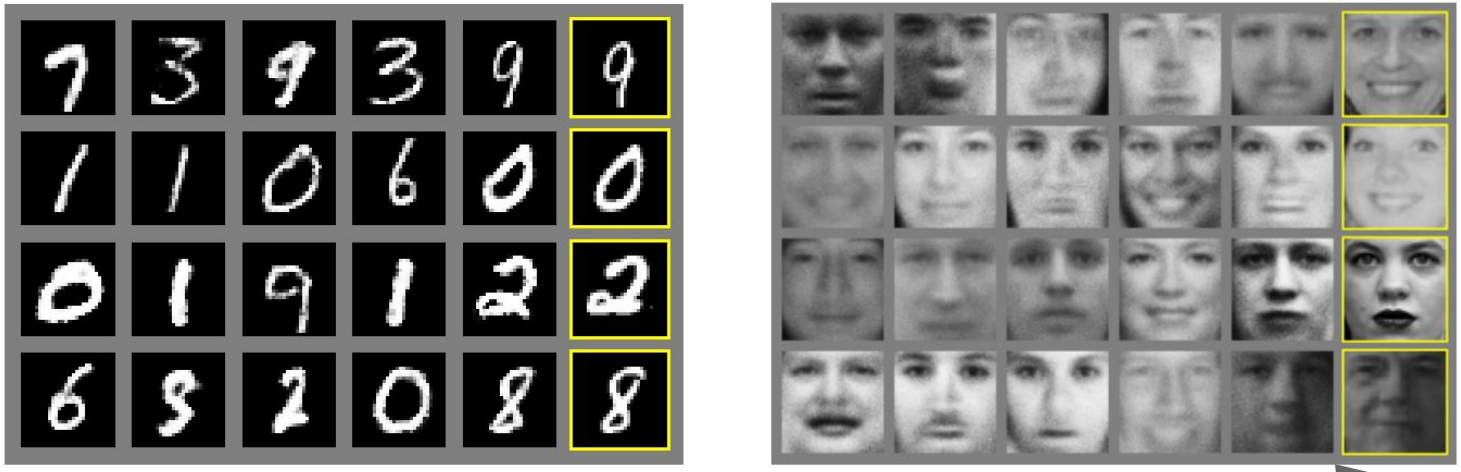


After training, use generator network to generate new images

Generative Adversarial Nets

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

Generated samples

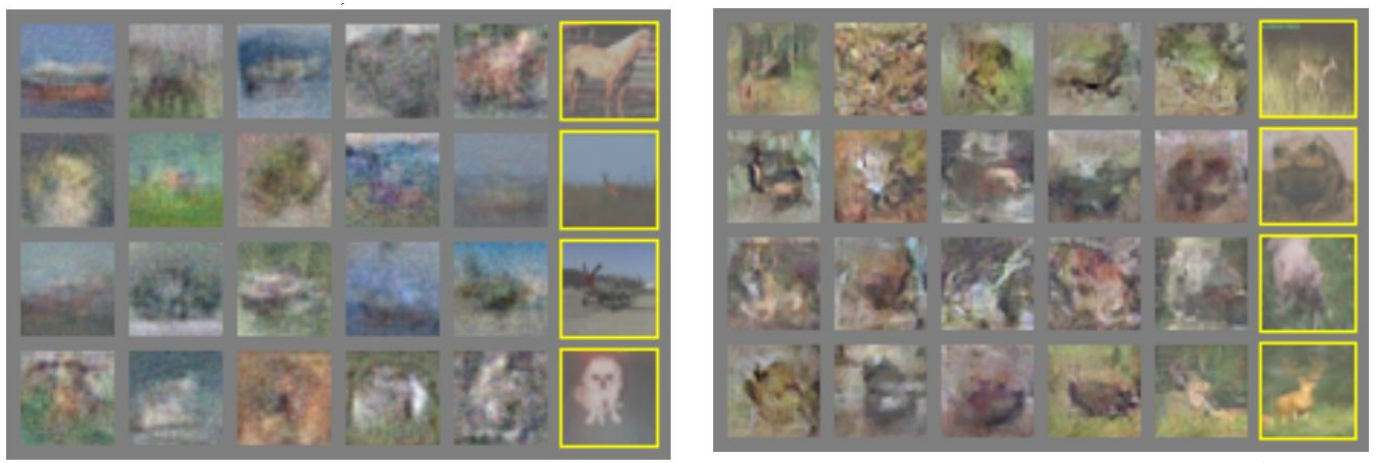


Nearest neighbor from training set

Generative Adversarial Nets

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

Generated samples (CIFAR-10)



Nearest neighbor from training set

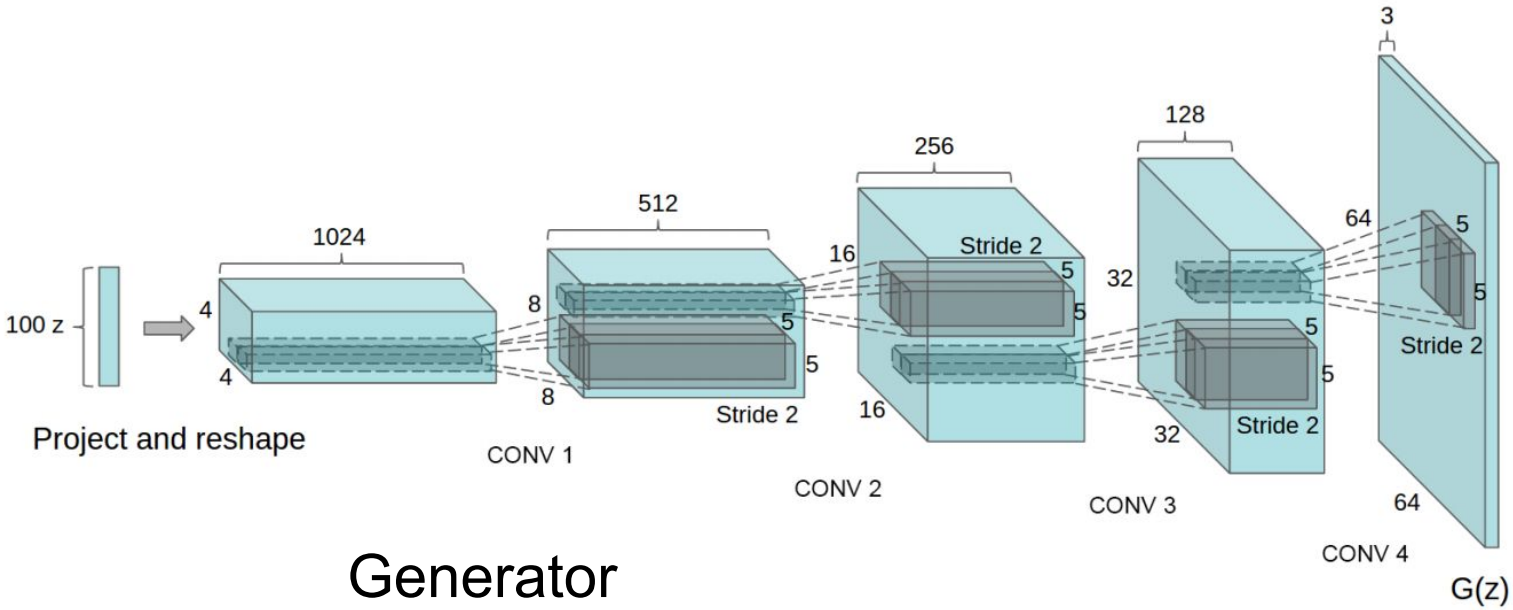
Generative Adversarial Nets: Convolutional Architectures

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

Architecture guidelines for stable Deep Convolutional GANs

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

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

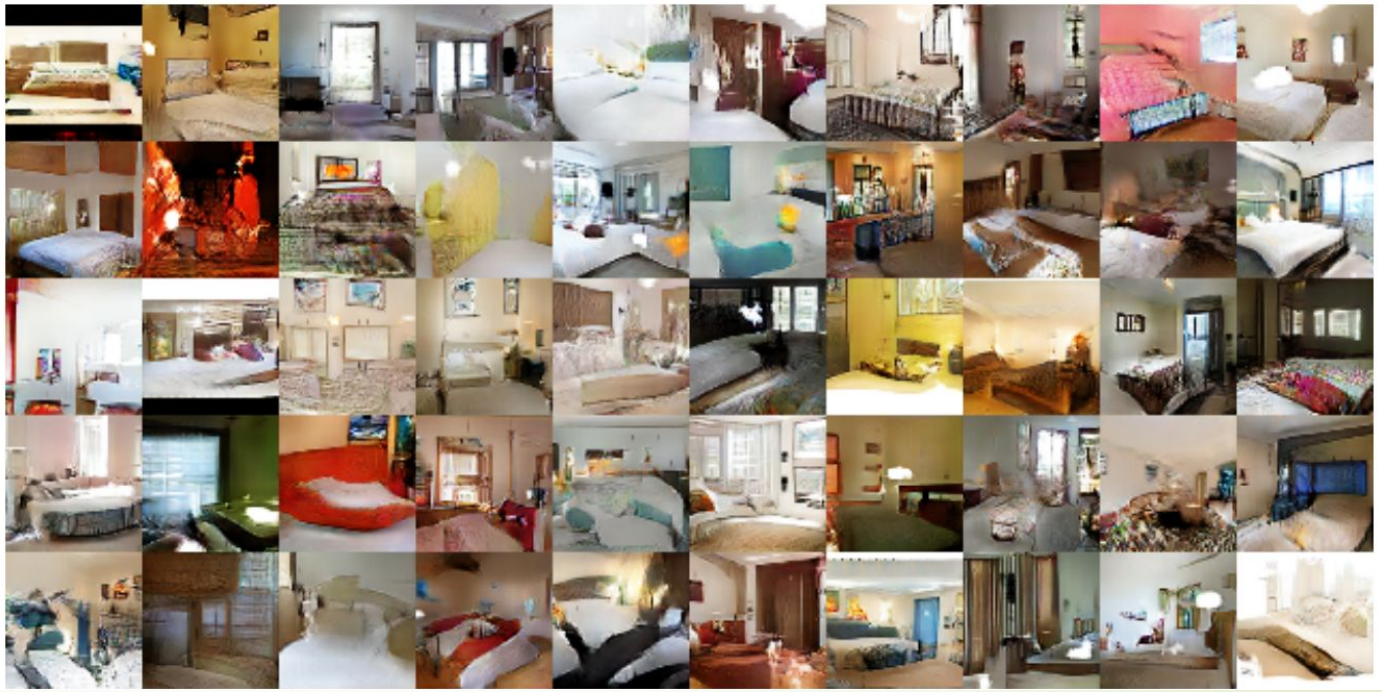


Generator

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

Generative Adversarial Nets: Convolutional Architectures

Samples from the model look amazing!



Radford et al, ICLR 2016

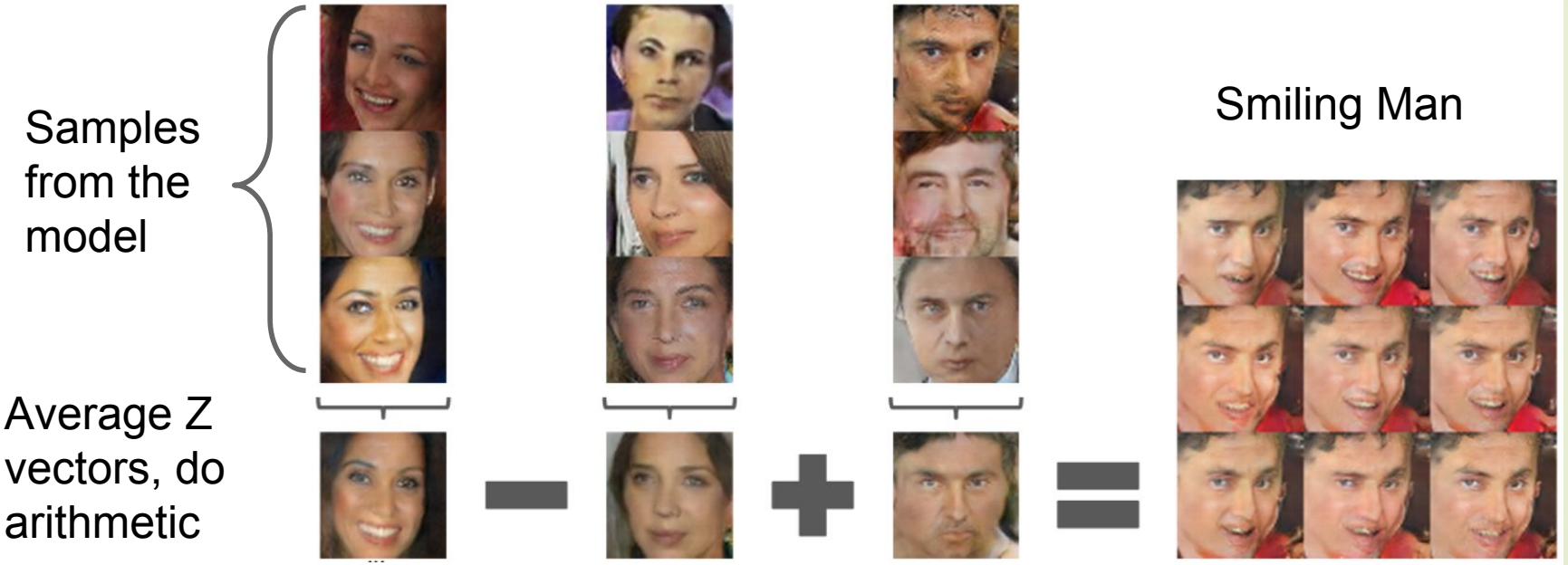
Generative Adversarial Nets: Convolutional Architectures

Interpolating between random points in laten space

Radford et al, ICLR 2016

Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016



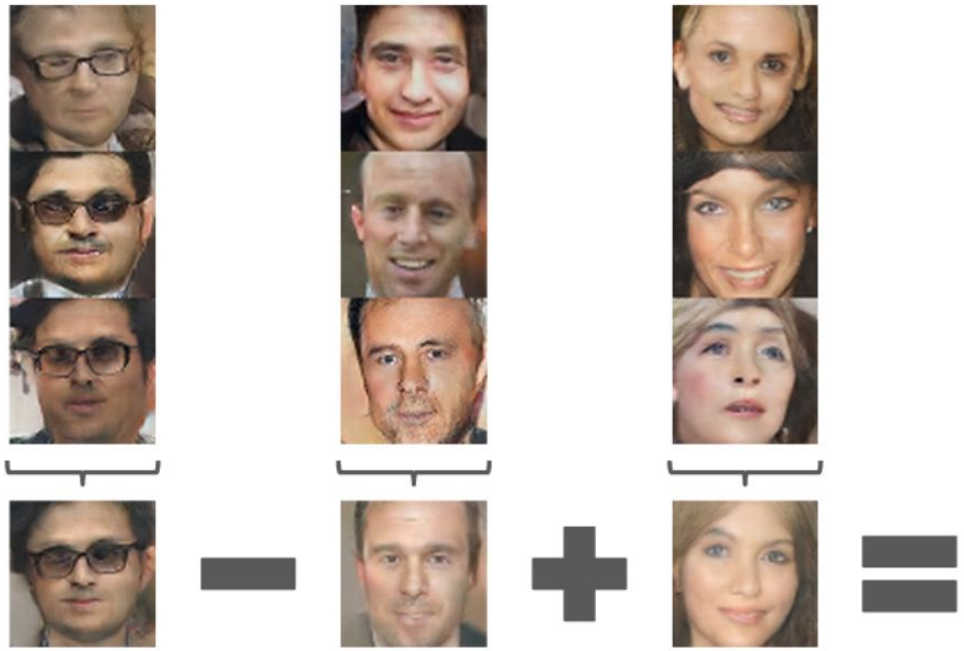
Generative Adversarial Nets: Interpretable Vector Math

Glasses man

No glasses man

No glasses woman

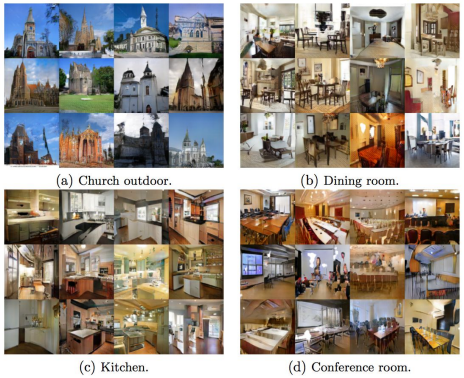
Radford et al, ICLR 2016



Woman with glasses

2017: Year of the GAN

Better training and generation



LSGAN. Mao et al. 2017.

BEGAN. Bertholet et al. 2017.

Source->Target domain transfer

CycleGAN. Zhu et al. 2017.

Text -> Image Synthesis

Reed et al. 2017.

Many GAN applications

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

Reference of GANs

- The GAN zoo: https://github.com/hindupuravinash/the-gan-zoo
- See also: https://github.com/soumith/ganhacks for tips and tricks for trainings GANs

GANs

- Don't work with an explicit density function
  Take game-theoretic approach: learn to generate from training distribution through 2-player minimax zero-sum game
- Pros:
  - Beautiful, state-of-the-art samples!
- Cons:
  - Trickier / more unstable to train
  - Can't solve inference queries such as p(x), p(z|x)
- Active areas of research:
  - Better loss functions, more stable training (Wasserstein GAN, LSGAN, etc. Tong ZHANG next time)
  - Conditional GANs, GANs for all kinds of applications

Composite Functional Gradient Learning of Generative Adversarial Models

Rie JOHNSON, *IBM*, and **Tong ZHANG**, *Tencent AI Lab*

Recall that

Want to draw samples similar to $S = \{x_1, \ldots, x_n\}$

Procedure:

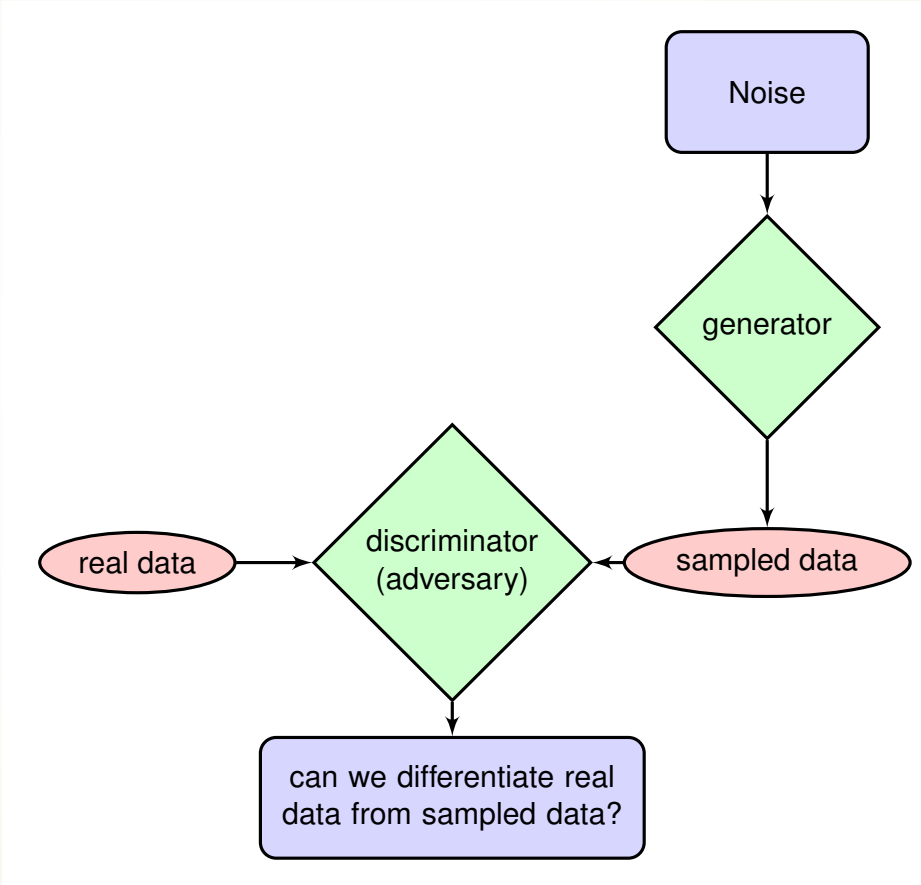
- Generate z from an initial noise distribution p_z
- Transform $z : z \rightarrow G(z)$ so that $G(z)$ has the same distribution as S

Key problem:

- How to estimate the transformation $G(z)$?

Modern answers: GAN (and related, VAE)

Generative Adversarial Network (GAN)



Mathematical Formulation of GAN

- Real data $S = \{x_1, \ldots, x_n\}$
- Noise $z_1, \ldots, z_m$
- Generator $G(\theta; z)$
- Discriminator $d(\psi; x)$

Nonconvex Minimax Saddle Point Optimization Problem:

$$\max_{\psi} \min_{\theta} \left[\frac{1}{n} \sum_i \log d(\psi; x_i) + \frac{1}{m} \sum_j \log(1 - d(\psi; G(\theta; z_j))) \right].$$

We have:

$$d(\psi; x) = \frac{1}{1 + \exp(-D(\psi; x))}$$

Solving Saddle Point Optimization

$$\max_{\psi} \min_{\theta} \left[\frac{1}{n} \sum_{i} \log d(\psi; x_i) + \frac{1}{m} \sum_{j} \log(1 - d(\psi; G(\theta; z_j))) \right].$$

Using SGD as follows.

- Update discriminator (primal variable SGD):

$$\psi \leftarrow \psi + \eta \nabla_{\psi} \left[\log d(\psi; x_i) + \log(1 - d(\psi; G(\theta; z_j))) \right]$$

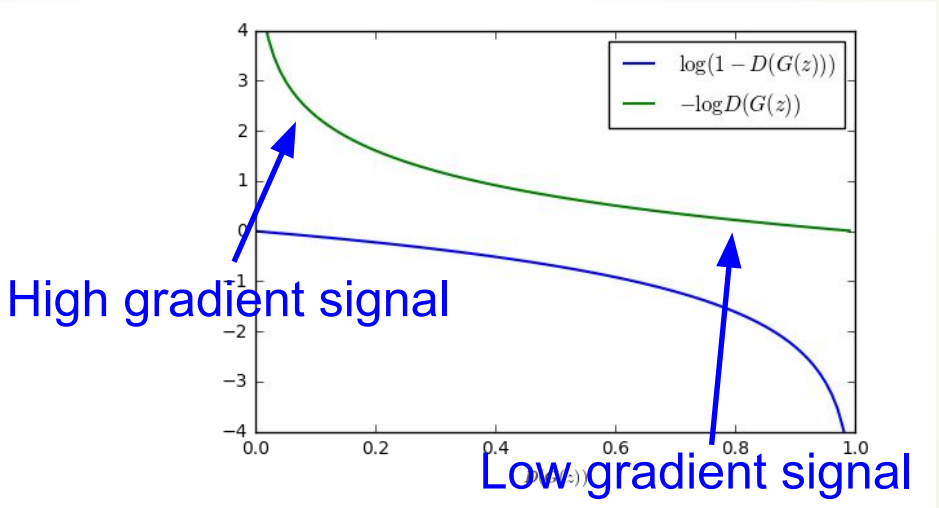
- Update generator (dual variable SGD):

$$\theta \leftarrow \theta - \eta' \nabla_{\theta} \log(1 - d(\psi; G(\theta; z_j)))$$

The Log D trick

Strange phenomenon: generator update with logD trick is practically better

$$\theta \leftarrow \theta + \eta' \nabla_\theta \log(d(\psi; G(\theta; z_j))$$



Mathematical Theory of GAN

The population version of GAN tries to find optimal d to maximize

$$\int \log d(x) p_{\text{real}}(x) dx + \int \log(1 - d(x)) p_{\text{gen}}(x) dx,$$

where

$p_{\text{gen}}(x)$ is the density of $x = G(z)$

The optimal d is given by

$$d_{\text{optimal}}(x) = \frac{p_{\text{real}}(x)}{p_{\text{real}}(x) + p_{\text{gen}}(x)}.$$

Generator Minimizes JS-divergence

$$\max_{d} \min_{G} \left[\mathbf{E}_x \log d(x) + \mathbf{E}_z \log(1 - d(G(z))) \right].$$

Substitute d by d_{optimal}, we get Jensen-Shanon (JS) divergence minimization:

$$\min_{G} \left[\int p_{\text{real}}(x) \log \frac{2p_{\text{real}}(x)}{p_{\text{real}}(x) + p_{\text{gen}}(x)} dx + \int p_{\text{gen}}(x) \log \frac{2p_{\text{gen}}(x)}{p_{\text{real}}(x) + p_{\text{gen}}(x)} dx \right],$$

where

$p_{\text{gen}}(x)$ is the density of $x = G(z)$

Some Issues of GAN

- The optimization procedure is unstable
  - one proposed improvement is WGAN (changing loss function)
- Practical implementation with logD trick is only a heuristic
  - inconsistent with the minimax formulation, implying the theory is flawed
- Minimizes JS divergence, not KL divergence

We want to design a procedure that addresses the above points.

- based on KL divergence minimization
- stable (by modifying optimization process of GAN)
- can explain the logD trick

New Approach

- Learn $G(z)$ to minimize the KL-divergence between the distributions of real data and generated data:

$$\int p_{\text{real}}(x) \log \frac{p_{\text{real}}(x)}{p_{\text{gen}}(x)} dx$$

- Procedure uses functional gradient learning greedily, similar to gradient boosting.
- Learning procedure uses functional compositions in the form

$$G_t(z) = G_{t-1}(z) + \eta_t g_t(G_{t-1}(z)), \qquad (t = 1, \ldots, T)$$

gradient descent in function space

Adversarial Learner

Given training examples $\{(x_i, y_i)\}$ (where $y_i = \pm 1$).

Assume there is an oracle adversarial learner $\mathcal{A}$ that can solve the logistic regression problem:

$$D(x) \approx \arg\min_D \sum_i \ln(1 + \exp(-D(x_i)y_i)).$$

Using

- real sample $S = \{x_1, \ldots, x_n\}$ with label $y = 1$
- generated sample $\{G(z_1), \ldots, G(z_m)\}$ with label $y = -1$

We can find approximately

$$D(x) \approx \ln \frac{p_*(x)}{p_{\text{gen}}(x)}$$

Johnson-Zhang'2018

Theorem

Consider variable transformation $X' = X + \eta g(X)$.
Let p *be the probability density of random variable* X.
Let p' *be the probability density of random variable* X'.
Let p_* *be the probability density of the real data.*
Let $\mathcal{D}(x) := \ln \frac{p_*(x)}{p(x)}$.
Assume $D_\epsilon(x) \approx \mathcal{D}(x)$ *is learned from adversarial learner:*

$$\int p_*(x) \max(1, \|\nabla \ln p_*(x)\|) \left(|D_\epsilon(x) - \mathcal{D}(x)| + \left| e^{D_\epsilon(x)} - e^{\mathcal{D}(x)} \right| \right) dx \leq \epsilon .$$

Then for some constant $c > 0$*:*

$$KL(p_*||p') \leq KL(p_*||p) - \eta \int p_*(x)\, g(x)^\top \nabla D_\epsilon(x)\, dx + c\eta^2 + c\eta\epsilon.$$

Implication of the Theorem

The result is

$$KL(p_*||p') \leq KL(p_*||p) - \eta \int p_*(x) g(x)^\top \nabla D(x)\, dx + O(\eta^2).$$

If we further take

$$g(x) = s(x)\nabla D(x),$$

where $s(x) > 0$ is a scaling factor, then

$$KL(p_*||p') \leq KL(p_*||p) - \eta \int p_*(x) s(x)\, \|\nabla D(x)\|_2^2\, dx + O(\eta^2).$$

Implication:

$$\int p_*(x) s(x)\, \|\nabla D(x)\|_2^2\, dx \to 0,$$

which means

$$D(x) = \ln \frac{p_*(x)}{p_{\text{gen}}(x)} = \text{constant}$$

Johnson-Zhang'2018

Algorithm 1 CFG: Composite Functional Gradient Learning

Require: real data $x_1^*, \ldots, x_n^*$, initial generator $G_0(z)$

1: **for** $t = 1, 2, \ldots, T$ **do**

2: $D_t(x) \leftarrow \arg\min_D \left[\frac{1}{n}\sum_{i=1}^n \ln(1 + e^{-D(x_i^*)}) + \frac{1}{m}\sum_{i=1}^m \ln(1 + e^{D(G_{t-1}(z_i))})\right]$

3: $g_t(x) \leftarrow s_t(x)\nabla D_t(x)$ (usually $s_t(x) = 1$)

4: $G_t(z) \leftarrow G_{t-1}(z) + \eta_t g_t(G_{t-1}(z))$, for some $\eta_t > 0$.

5: **end for**

6: **return** generator $G_T(z)$

Theory: as $n \to \infty$ and $T \to \infty$

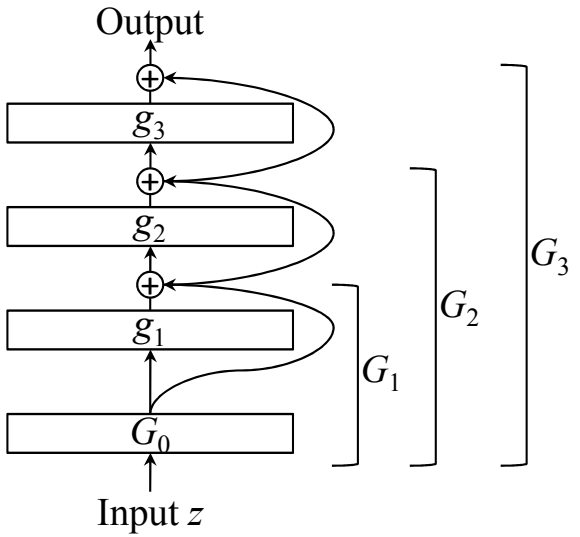
$$G_T(z) \sim p_*(\cdot)$$

Algorithm 2 ICFG: Incremental Composite Functional Gradient Learning

Require: training examples S_*, prior p_z, initial generator G_0, discriminator D

for $t = 1, 2, \ldots, T$ **do**
  for U steps **do**
    Sample $x_1^*, \ldots, x_b^*$ from S_*, and $z_1, \ldots, z_b$ according to p_z.
    Update discriminator D by SGD with minibatch gradient:
    $\nabla_{\theta_D} \frac{1}{b} \sum_{i=1}^{b} \left[\ln(1 + \exp(-D(x_i^*))) + \ln(1 + \exp(D(G_{t-1}(z_i))))\right]$
  end for
  $g_t(x) \leftarrow s_t(x) \nabla D(x)$ (most simply $s_t(x) = 1$)
  $G_t(z) \leftarrow G_{t-1}(z) + \eta_t g_t(G_{t-1}(z))$, for some $\eta_t > 0$.
end for
return generator G_T

Graphical Illustration



Generator network automatically derived by CFG
$T = 3$ and '$\oplus$' indicates addition

Problem: network depth grows when T increases
Solution: use a fixed depth approximator

Algorithm 3 xICFG: Approximate Incremental CFG Learning

Require: a set of training examples S_*, prior p_z, approximator $\widetilde{G}$ at its initial state, discriminator D.

1: **loop**
2: $S_z \leftarrow$ an input pool of the given size, sampled according to p_z.
3: $q_z \leftarrow$ the uniform distribution over S_z.
4: $G, D \leftarrow$ output of ICFG using $S_*, q_z, \widetilde{G}, D$ as input.
5: **if** some exit criteria is met **then**
6: **return** generator G
7: **end if**
8: Update the approximator $\widetilde{G}$ to minimize $\frac{1}{2}\sum_{z \in S_z} \|\widetilde{G}(z) - G(z)\|^2$
9: **end loop**

$\widetilde{G}(z)$ is a network of the fixed size
$G(z)$ is of growing but limited size (initialized from $\widetilde{G}(z)$ every time)

Approximator Network

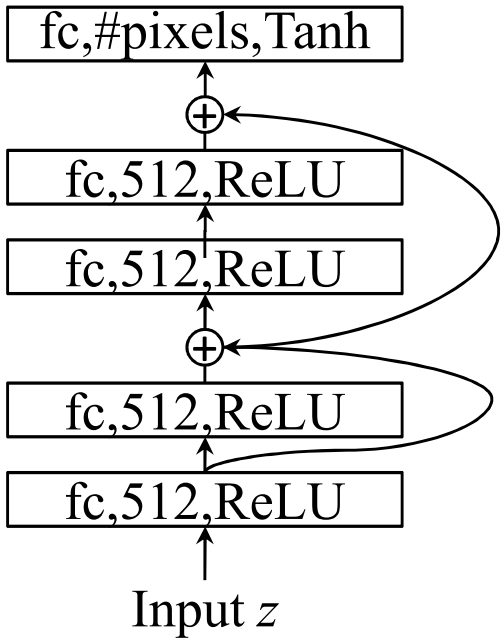


Figure: Fully-connected networks with shortcuts for use as a xICFG approximator or a GAN generator.

Comparison to Original GAN (GAN0)

Algorithm 4 GAN0(Generative Adversarial Nets)

Require: training examples S_*, prior p_z, discriminator D, generator G.

1: Initialize discriminator d and generator G randomly.
2: **for** T steps **do**
3: Update discriminator D by ascending the stochastic gradient:
4: Sample $z_1, \ldots, z_b$ according to p_z.
5: Update the generator by descending the stochastic gradient:
$$\frac{1}{b}\sum_{i=1}^{b} \underbrace{(1+\exp(-D(G(z_i))))^{-1}}_{s(G(z_i))} \nabla_{\theta_G} D(G(z_i))$$
6: **end for**
7: **return** generator G

Problem: $s(G(z)) \approx 0$ when $D(G(z)) \ll 0$, which happens in the beginning.

Comparison to GAN with LogD trick (GAN1)

Algorithm 5 GAN1(Generative Adversarial Nets)

Require: training examples S_*, prior p_z, discriminator D, generator G.

1: Initialize discriminator d and generator G randomly.
2: **for** T steps **do**
3: Update discriminator d by ascending the stochastic gradient:
4: Sample $z_1, \ldots, z_b$ according to p_z.
5: Update the generator by descending the stochastic gradient:

$$\frac{1}{b}\sum_{i=1}^{b} \underbrace{(1+\exp(D(G(z_i))))^{-1}}_{s(S(z_i))} \nabla_{\theta_G} D(G(z_i))$$

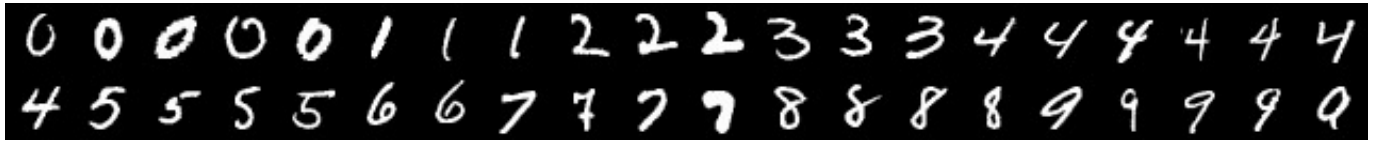
6: **end for**
7: **return** generator G

Good! $s(G(z)) \approx 1$ when $D(G(z)) \ll 0$, which happens in the beginning.

- GAN0 is original GAN
  - SGD optimization of the minimax formulation of GAN
- GAN1 is GAN with log trick, which is a heuristics
  - cannot be explained using the original minimax formulation of GAN can be explained by our theory
- xICFG
  - similar to GAN1
  - grows generator using composite function gradient
  - use approximator to reduce the network to fixed size

Data

- MNIST



- Street View House Numbers

- large-scale scene understanding (LSUN)

- quality *inception score*:

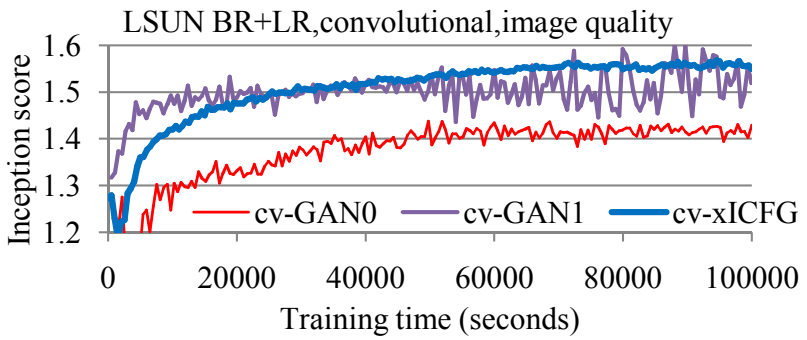
$$\exp(\mathbb{E}_x \mathrm{KL}(\Pr(y|x)||\Pr(y)))$$

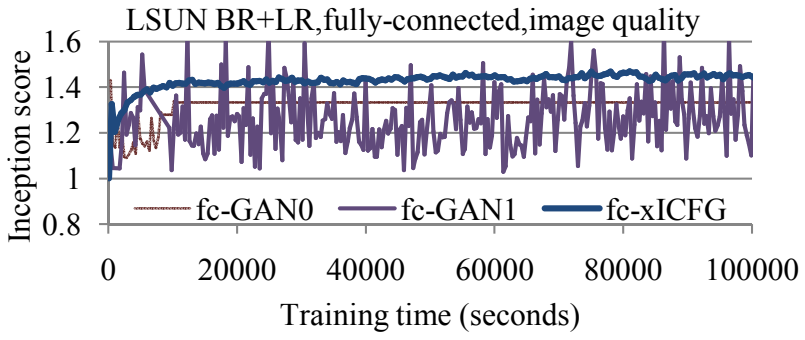
  - high-quality images should lead to high confidence in classification, and high inception score.
- diversity *diversity score*:

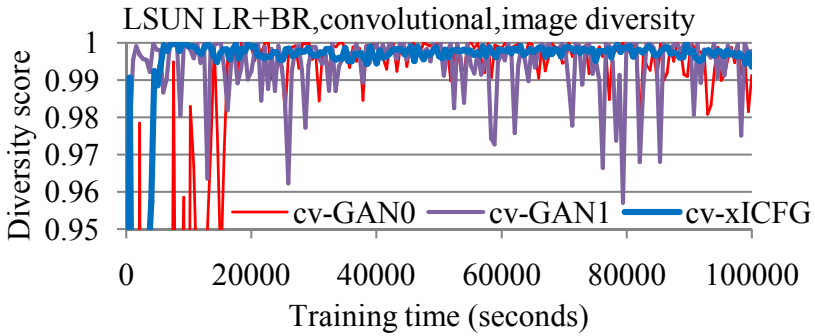
$$\exp(-\mathrm{KL}(\Pr(y)||\Pr_*(y))$$

  - diversity of generated images measured by diversity score becomes large (approaching to 1) when generated class distribution mimics real class distribution.

LSUN Performance Comparisons







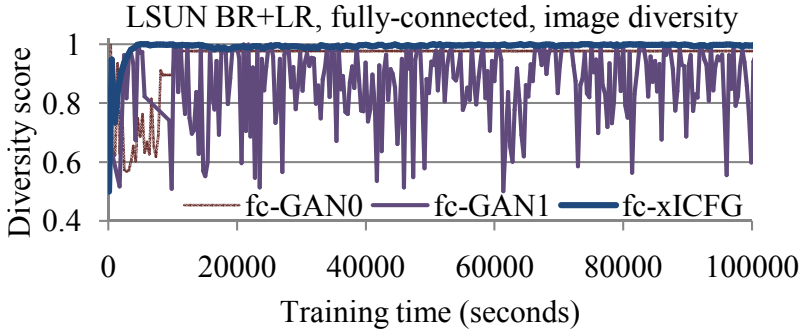


Figure: LSUN BR+LR. Image quality (upper) and diversity (lower). With the convolutional (left) and the fully-connected (right) approximator/generator.

Generated Data (LSUN)

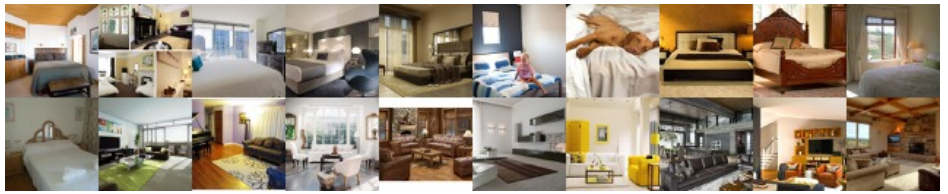


Figure: LSUN bedrooms & living rooms. Real images from the training set.

(a) cv-xICFG (1.55). High quality. No signs of modal collapse.

Wasserstein GAN (WGAN)?

- WGAN uses a different loss function than log-loss
- It claims to improve stability of GAN
- The CFG procedure claims improved stability as well
  - by making optimization easier
- How does CFG compare to WGAN?

Comparison with WGAN: CNN

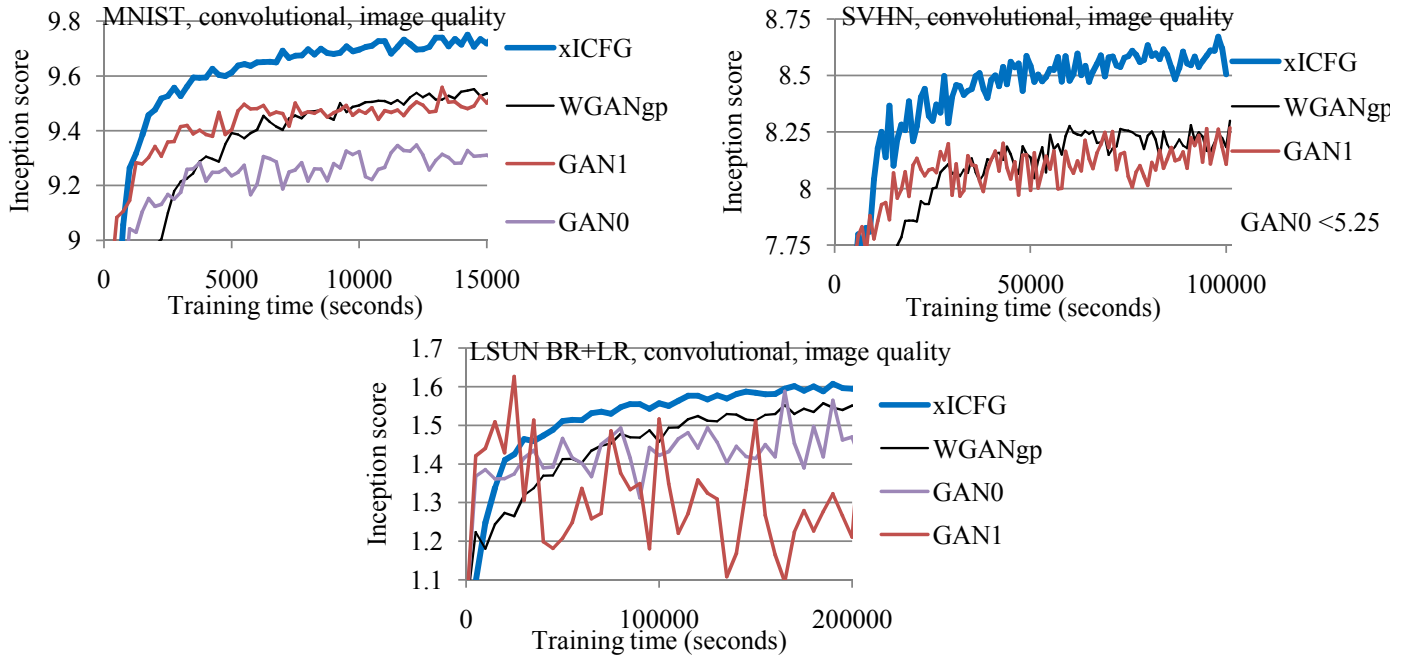


Figure: Image quality. Convolutional networks. The legends are sorted from the best to the worst. xICFG outperforms the others.

Creativity in LSUN (Tower & Bridge)

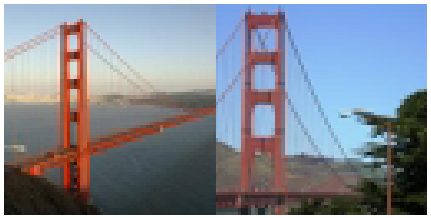
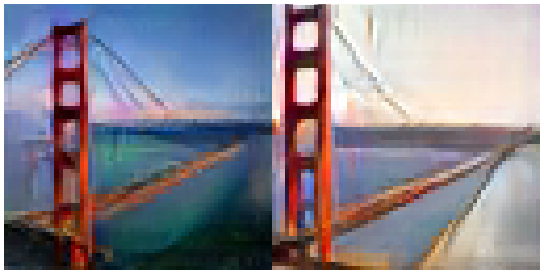
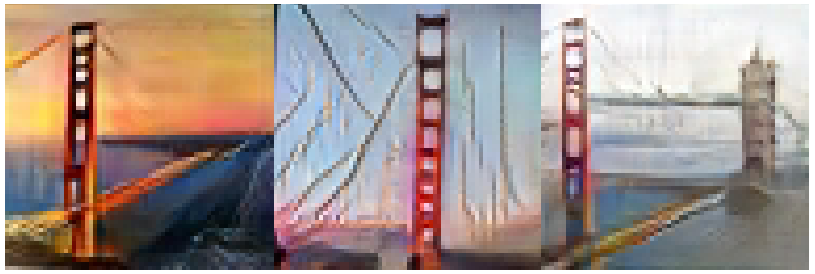


Figure: Real Golden Gate Bridge images: the red tower has 4 grids



(a) "Realistic"



(b) "Creative"

(a) Images generated by xICFG that resemble Golden Gate Bridge

(b) Images generated by xICFG that *modify* Golden Gate Bridge

Summary

- In the generative adversarial learning setting:
  GAN's minimax formulation optimizes JS-divergence
  GAN's algorithm is not stable.
  the practical use of logD trick inconsistent with the minimax formulation
- The optimization problem is hard and unstable
- This work: change optimization, gradient descent in function space
- Learn generator $G(z)$ using CFG (composite functional gradient).
  - Theory: minimizes KL-divergence
  - Theory: lead to the new stable algorithm xICFG.
  - Theory: explains the logD trick of GAN.
  - Experiments: xICFG performs better than GAN/WGAN
- Reference: *Rie Johnson, Tong Zhang*, Composite Functional Gradient Learning of Generative Adversarial Models. [arXiv:1801.06309]

Thank you!