



# Generative Models and GANs

1

Yuan YAO

HKUST



# Supervised vs Unsupervised Learning

## Supervised Learning

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

**Goal:** Learn a *function* to map  $x \rightarrow y$

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

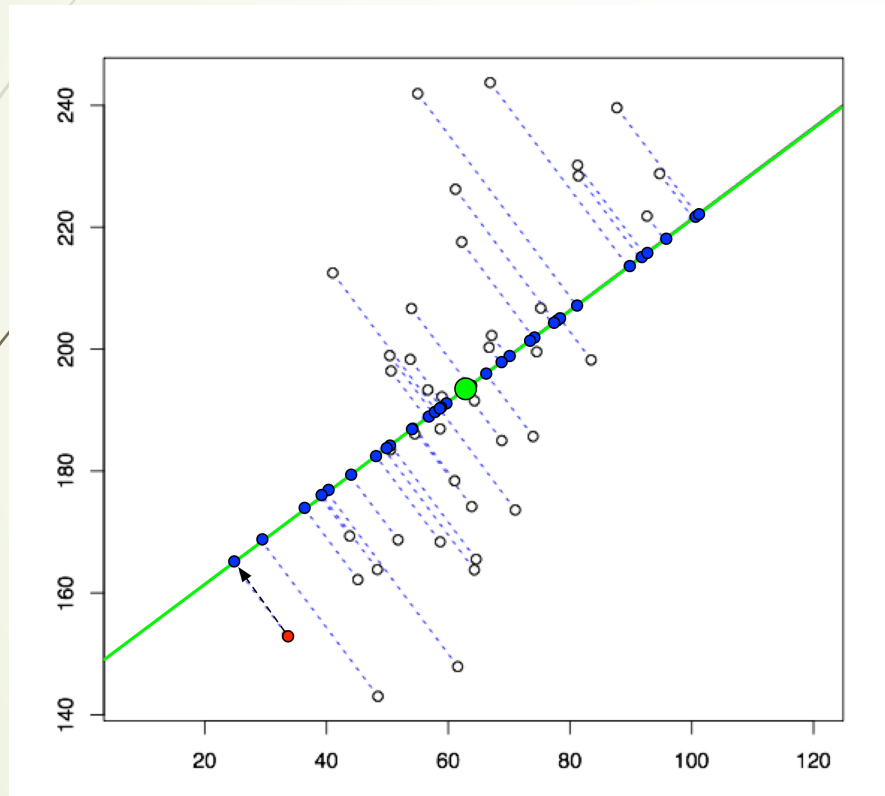
## Unsupervised Learning

**Data:** x  
Just data, no labels!

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

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

# PCA: Principal Component Analysis



- ▶ Data:  $\mathbf{x}_i = (x_{i1}, \dots, x_{ip})$ ,  $i = 1, \dots, n$ .
- ▶ Compute sample covariance matrix, e.g.  
$$\mathbf{S} = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \hat{\boldsymbol{\mu}})^T (\mathbf{x}_i - \hat{\boldsymbol{\mu}}).$$
- ▶ Decompose into eigenvalue-eigenvector pairs:

$$\mathbf{S} = \hat{\mathbf{e}} \hat{\boldsymbol{\Lambda}} \hat{\mathbf{e}}^T = (\hat{\mathbf{e}}_1 \dots \hat{\mathbf{e}}_p) \hat{\boldsymbol{\Lambda}} \begin{pmatrix} \hat{\mathbf{e}}_1 \\ \vdots \\ \hat{\mathbf{e}}_p \end{pmatrix}$$

where  $\hat{\boldsymbol{\Lambda}} = \text{diag}(\hat{\lambda}_1, \dots, \hat{\lambda}_p)$ .

- ▶  $(\hat{\lambda}_k, \hat{\mathbf{e}}_k)$  are eigen-value-eigenvector pairs,  $\hat{\lambda}_1 \geq \dots \geq \hat{\lambda}_p$ .

• Can you find a low dimensional affine representation?

# PCA

- ▶ The  $k$ -th sample PC.s:

$$Z_k = \begin{pmatrix} z_{1k} \\ \vdots \\ z_{nk} \end{pmatrix} = \mathbf{X}\hat{\mathbf{e}}_k$$

- ▶ Component-wise,  $z_{ik} = x_{i1}e_{1k} + x_{i2}e_{2k} + \dots + x_{ip}e_{pk}$  are the principle component scores of the  $i$ -th observation.
- ▶  $\hat{\lambda}_k$  measures the importance of the  $k$ -th PC.
- ▶  $\hat{\lambda}_k / (\hat{\lambda}_1 + \dots + \hat{\lambda}_p) = \hat{\lambda}_k / \text{trace}(\mathbf{S})$  is interpreted as percentage of the total variation explained by  $Y_k$ .
- ▶ Usually retain the first few PCs.
- ▶ PCs are uncorrelated with each other.



# Example: USArrests Data

For each of the 50 states in the United States, the data set contains the number of arrests per 100,000 residents for each of three crimes: Assault, Murder, and Rape.

We also record UrbanPop (the percent of the population in each state living in urban areas).

The principal component score vectors  $Z_k$  have length  $n = 50$ , and the principal component loading vectors ( $\hat{e}_k$ ) have length  $p = 4$ . PCA was performed after standardizing each variable to have mean zero and standard deviation one.

	PC1	PC2
Murder	0.5358995	0.4181809
Assault	0.5831836	0.1879856
UrbanPop	0.2781909	0.8728062
Rape	0.5434321	0.1673186

Table 10.1. The principal component loading vectors,  $\hat{e}_1$  and  $\hat{e}_2$ , for the USArrests data. These are also displayed in Figure 10.1.

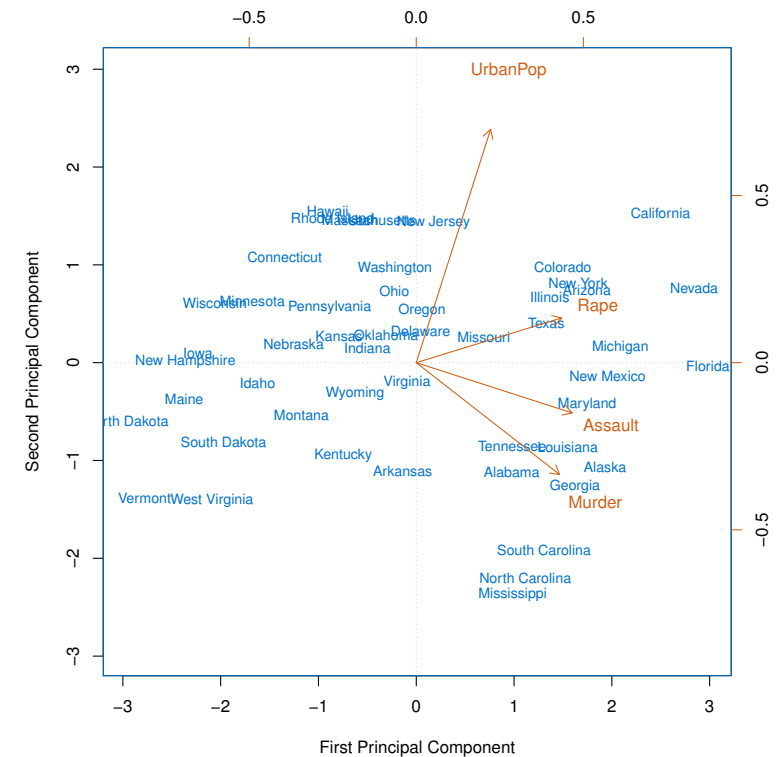


Figure: 10.1. Next page

# K-Means Clustering

## Algorithm 10.1 K-Means Clustering

- ▶ 1. Randomly assign a number, from 1 to  $K$ , to each of the observations. These serve as initial cluster assignments for the observations.
- ▶ 2. Iterate until the cluster assignments stop changing:
  1. For each of the  $K$  clusters, compute the cluster centroid. The  $k$ th cluster centroid is the vector of the  $p$  feature means for the observations in the  $k$ th cluster.
  2. Assign each observation to the cluster whose centroid is closest (where closest is defined using Euclidean distance).

FIGURE 10.6. The progress of the K-means algorithm on the example of Figure 10.5 with  $K = 3$ . Top left: the observations are shown. Top center: in Step 1 of the algorithm, each observation is randomly assigned to a cluster. Top right: in Step 2(a), the cluster centroids are computed. These are shown as large colored disks. Initially the centroids are almost completely overlapping because the initial cluster assignments were chosen at random. Bottom left: in Step 2(b), each observation is assigned to the nearest centroid. Bottom center: Step 2(a) is once again performed, leading to new cluster centroids. Bottom right: the results obtained after ten iterations.

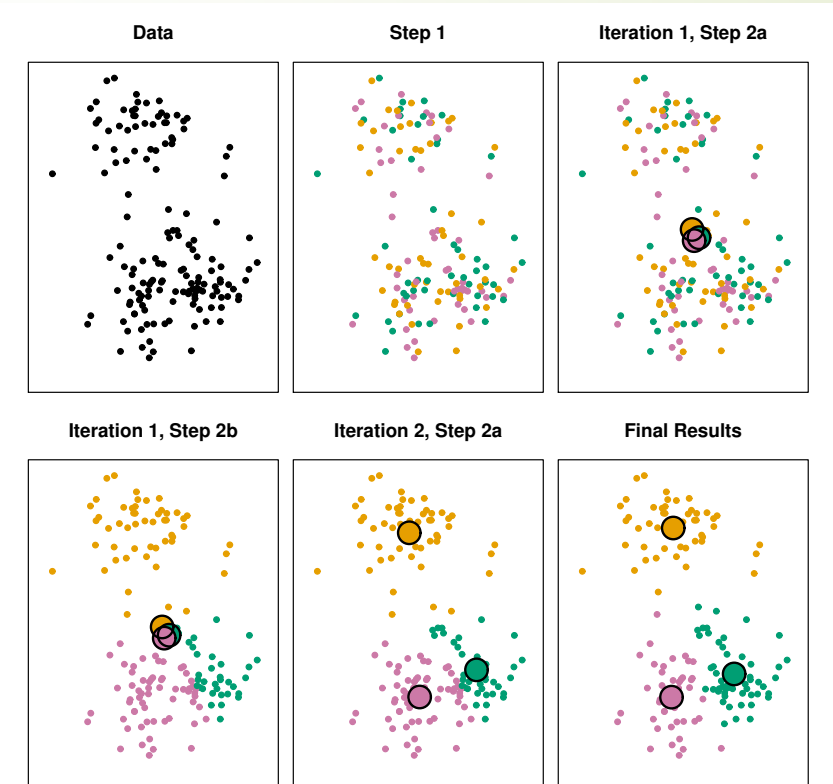


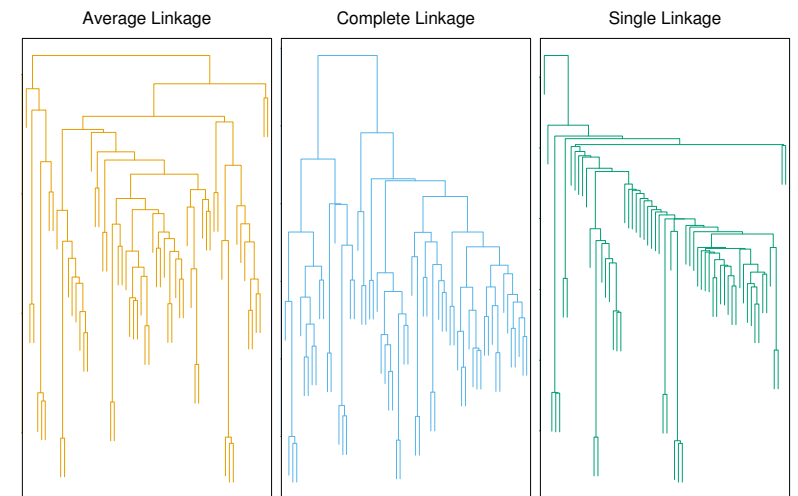
Figure: 10.6

# Hierarchical Clustering Algorithms

- ▶ 1. Begin with  $n$  observations and a measure (such as Euclidean distance) of all the  $\binom{n}{2} = n(n-1)/2$  pairwise dissimilarities. Treat each observation as its own cluster.
- ▶ 2. For  $i = n, n-1, \dots, 2$ :
  1. Examine all pairwise inter-cluster dissimilarities among the  $i$  clusters and identify the pair of clusters that are least dissimilar (that is, most similar). Fuse these two clusters. The dissimilarity between these two clusters indicates the height in the dendrogram at which the fusion should be placed.
  2. Compute the new pairwise inter-cluster dissimilarities among the  $i-1$  remaining clusters.

Linkage	Description
Complete	Maximal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the largest of these dissimilarities.
Single	Minimal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the smallest of these dissimilarities. Single linkage can result in extended, trailing clusters in which single observations are fused one-at-a-time.
Average	Mean intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the average of these dissimilarities.
Centroid	Dissimilarity between the centroid for cluster A (a mean vector of length $p$ ) and the centroid for cluster B. Centroid linkage can result in undesirable inversions.

TABLE 10.2. A summary of the four most commonly-used types of linkage





# Manifold Learning: Nonlinear Dimensionality Reduction

- MDS
- ISOMAP
- LLE: Locally linear Embedding
- Laplacian Eigenmap
- Hessian Eigenmap
- Diffusion Map
- LTSA: Local Tangent Space Alignment
- \*MDS-SDP (Sensor-Network-Localization)
- t-SNE
- <https://scikit-learn.org/stable/modules/manifold.html>

# Generative Models

Given training data, generate new samples from same distribution



Training data  $\sim p_{\text{data}}(x)$

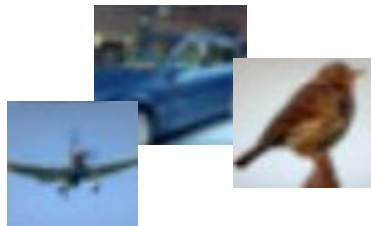


Generated samples  $\sim p_{\text{model}}(x)$

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

# Generative Models

Given training data, generate new samples from same distribution



Training data  $\sim p_{\text{data}}(x)$



Generated samples  $\sim p_{\text{model}}(x)$

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

Addresses density estimation, a core problem in unsupervised learning

## Several flavors:

- Explicit density estimation: explicitly define and solve for  $p_{\text{model}}(x)$
- Implicit density estimation: learn model that can sample from  $p_{\text{model}}(x)$  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

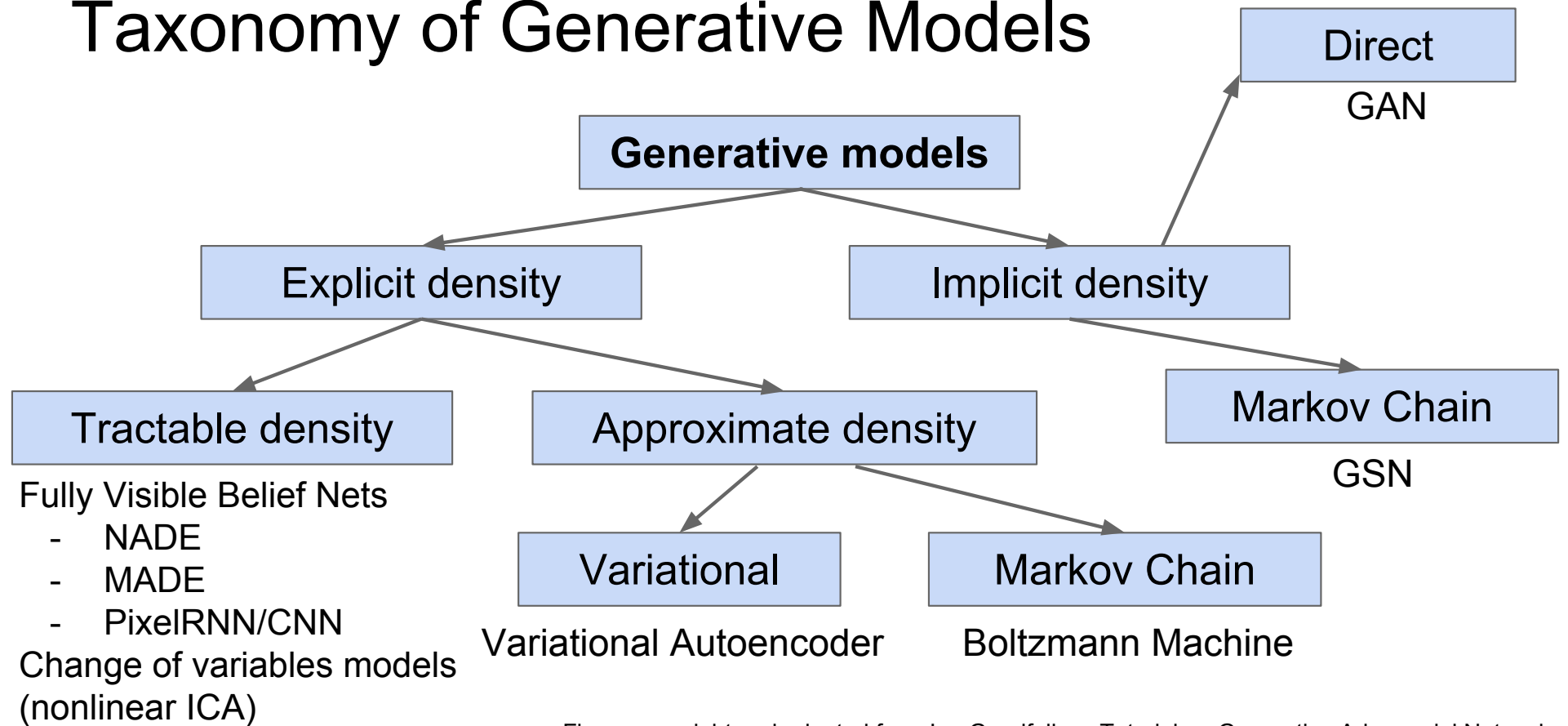


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

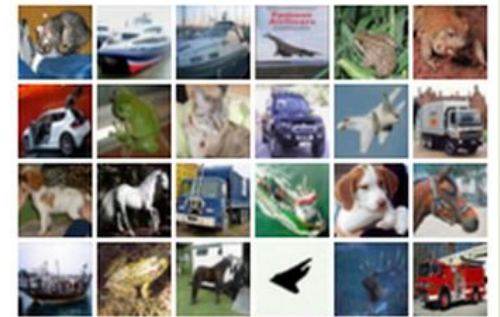
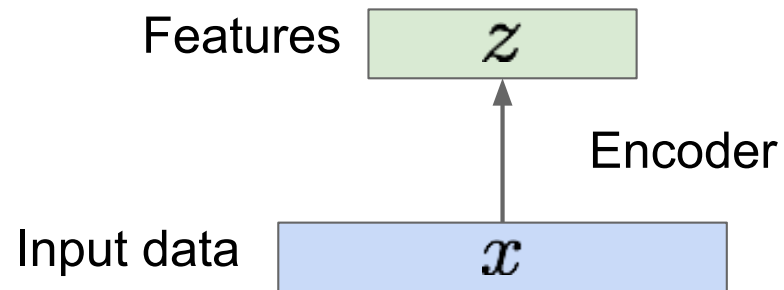


# Variational Autoencoders (VAE)

# Some background first: Autoencoders

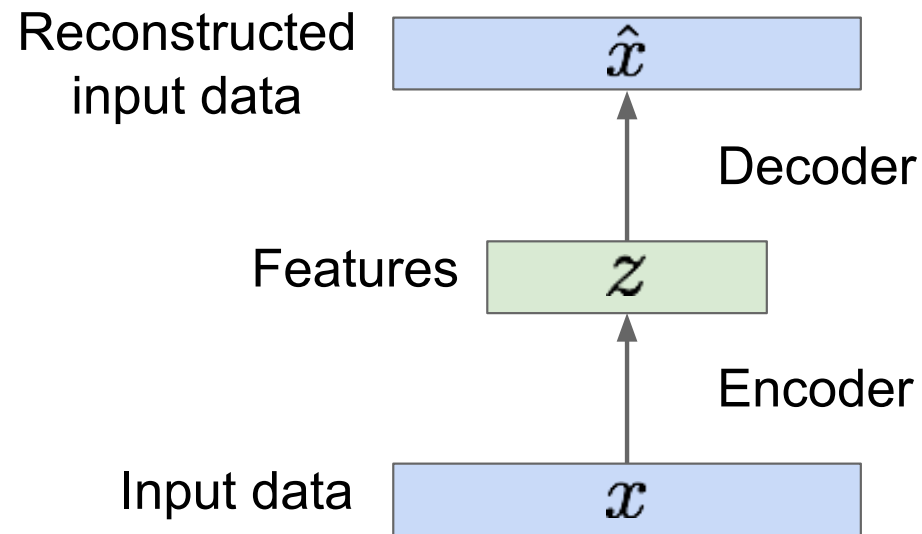
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

e.g. PCA, Manifold Learning, Dictionary Learning

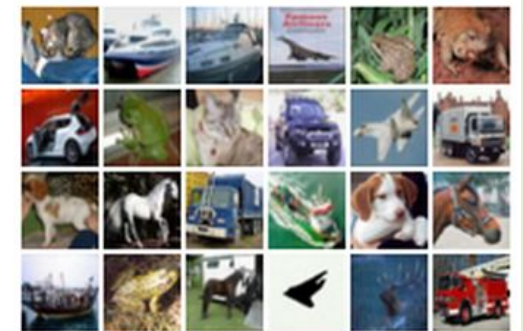


## How to learn this feature representation?

Train such that features can be used to reconstruct original data  
“Autoencoding” - encoding itself



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

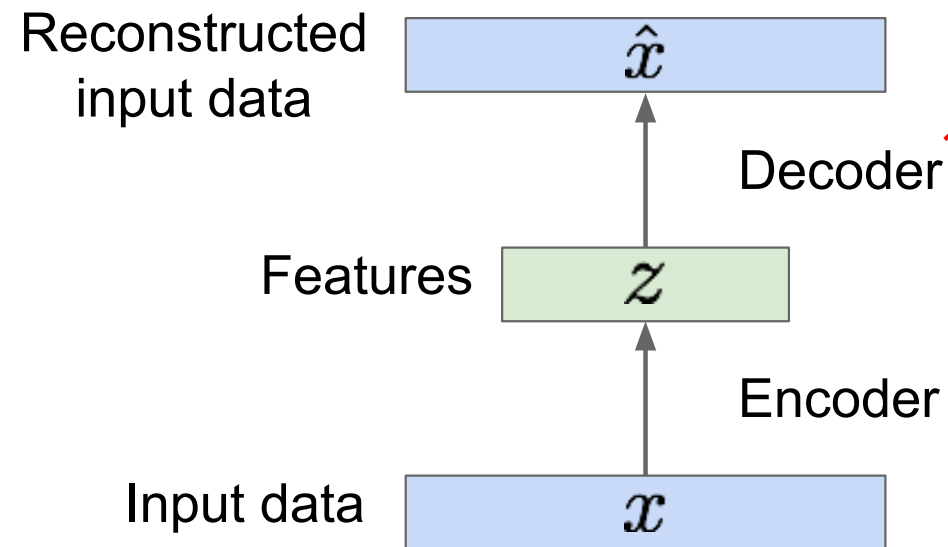




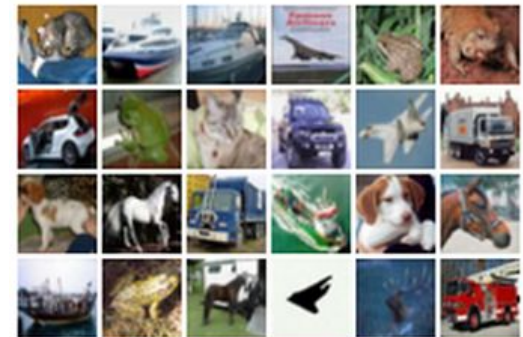
# Deep Learning for decoders

## 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)



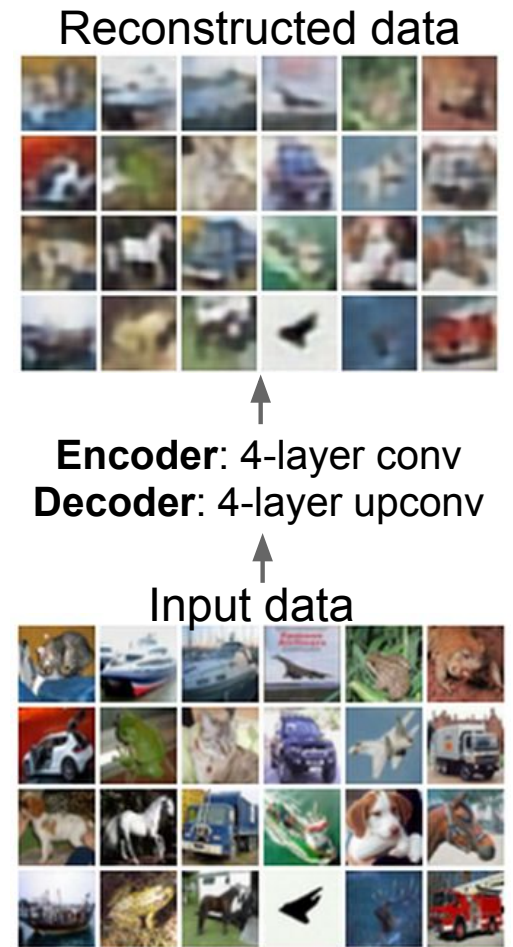
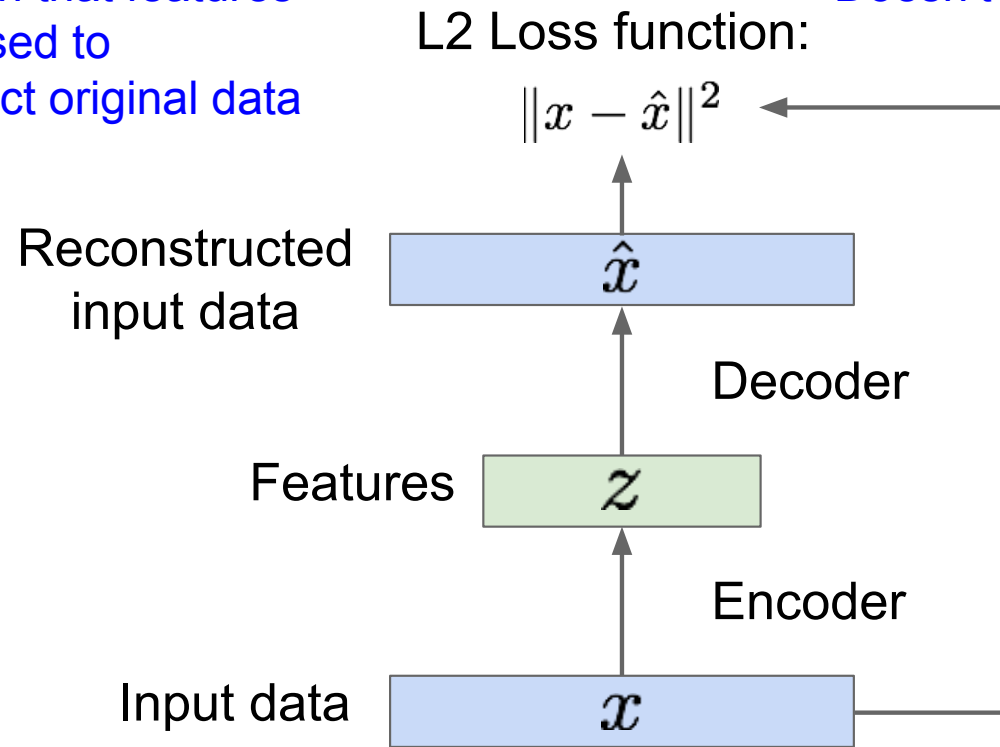


# L2 Loss functions

## Some background first: Autoencoders

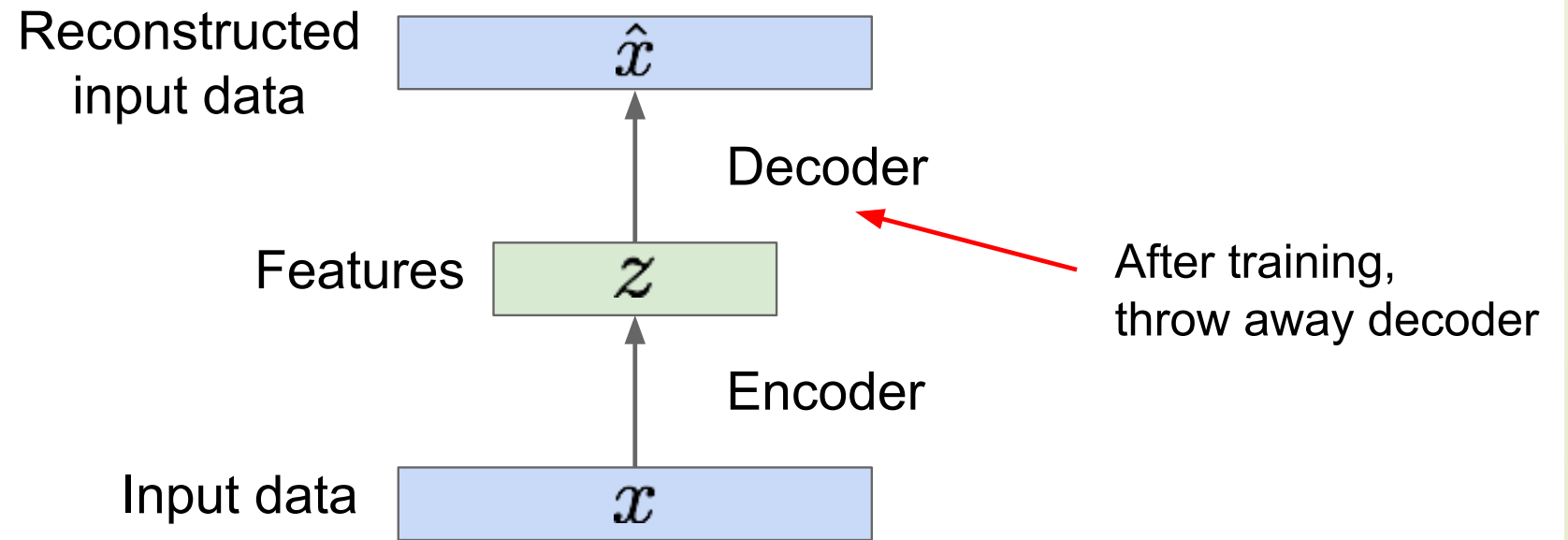
Train such that features can be used to reconstruct original data

Doesn't use labels!

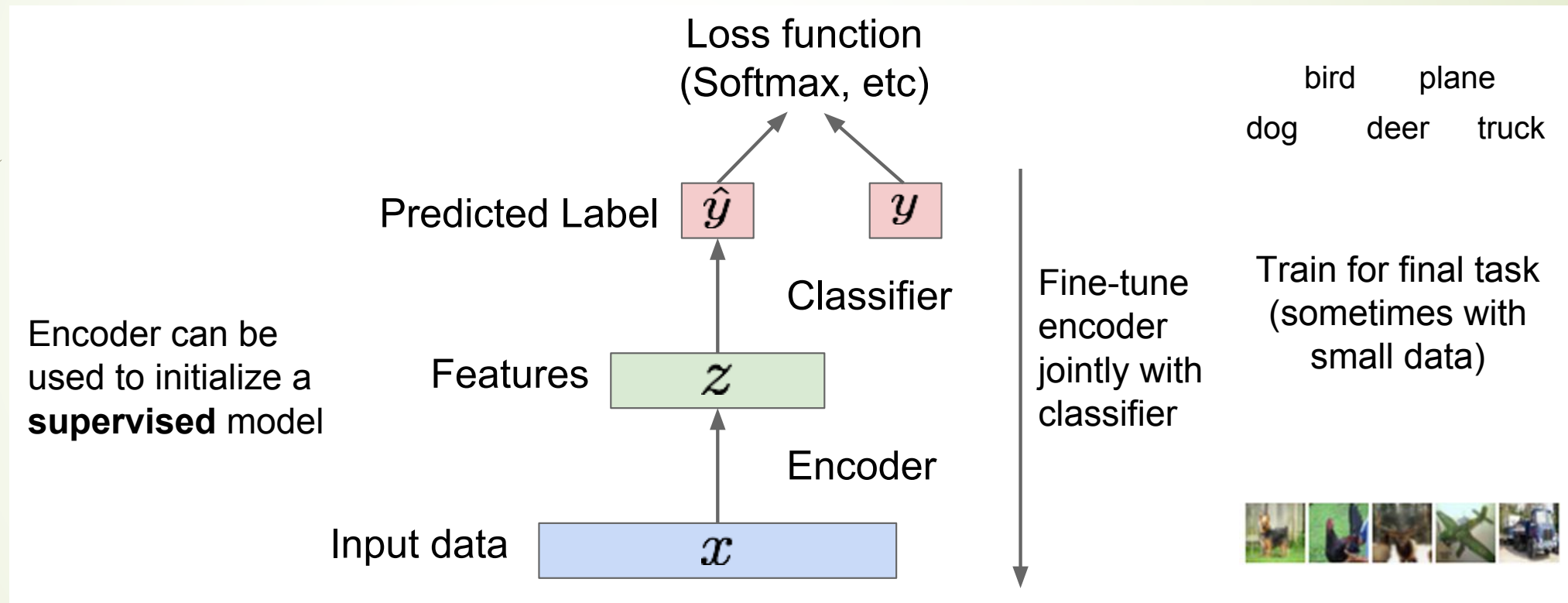


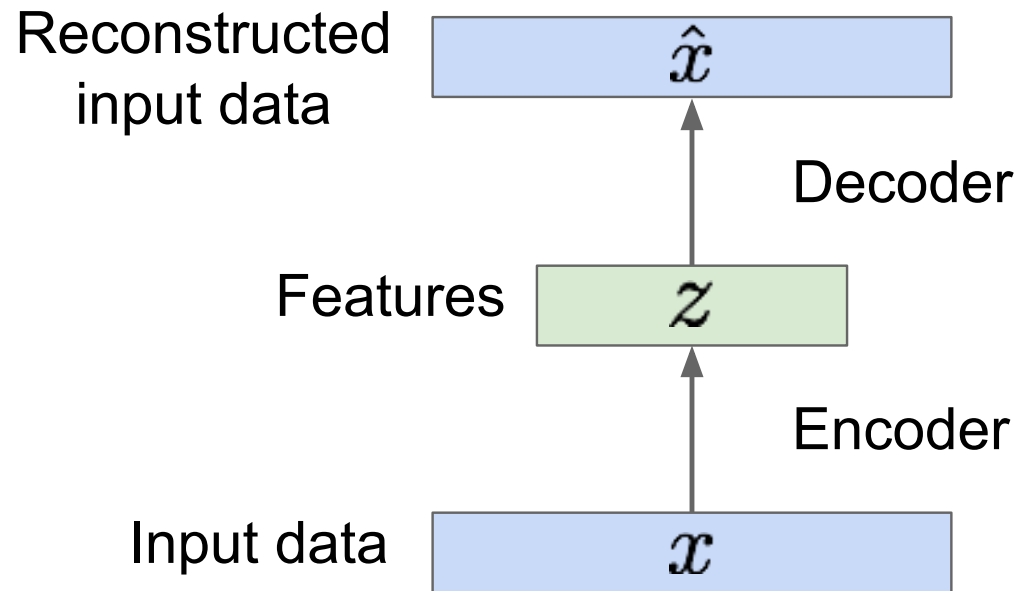


# Some background first: Autoencoders



# Autoencoders for Transfer Learning





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

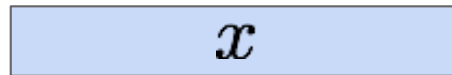
Features capture factors of variation in training data. Can we generate new images from an autoencoder?

# Variational Autoencoders

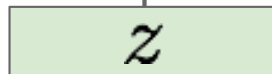
Probabilistic spin on autoencoders - will let us sample from the model to generate data!

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

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



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



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

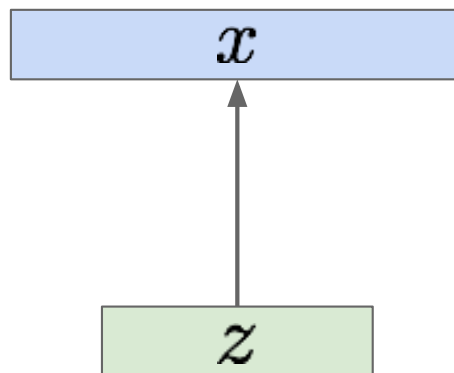
# Variational Autoencoders

Sample from  
true conditional

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

Sample from  
true prior

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



We want to estimate the true parameters  $\theta^*$  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.

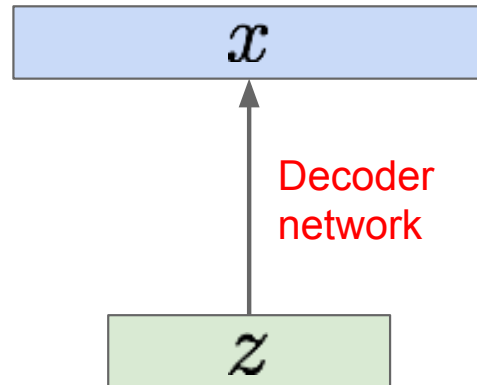
# Variational Autoencoders

Sample from  
true conditional

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

Sample from  
true prior

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



We want to estimate the true parameters  $\theta^*$  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

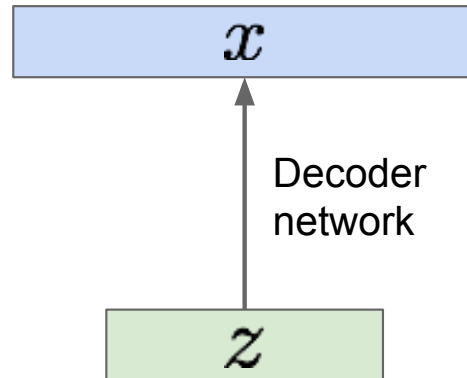
# Variational Autoencoders

Sample from  
true conditional

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

Sample from  
true prior

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



We want to estimate the true parameters  $\theta^*$  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  $z$



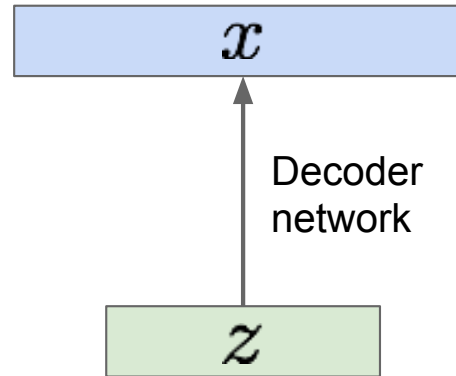
# Variational Autoencoders

Sample from  
true conditional

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

Sample from  
true prior

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



We want to estimate the true parameters  $\theta^*$  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:  $p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$

Intractable 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:  $p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$

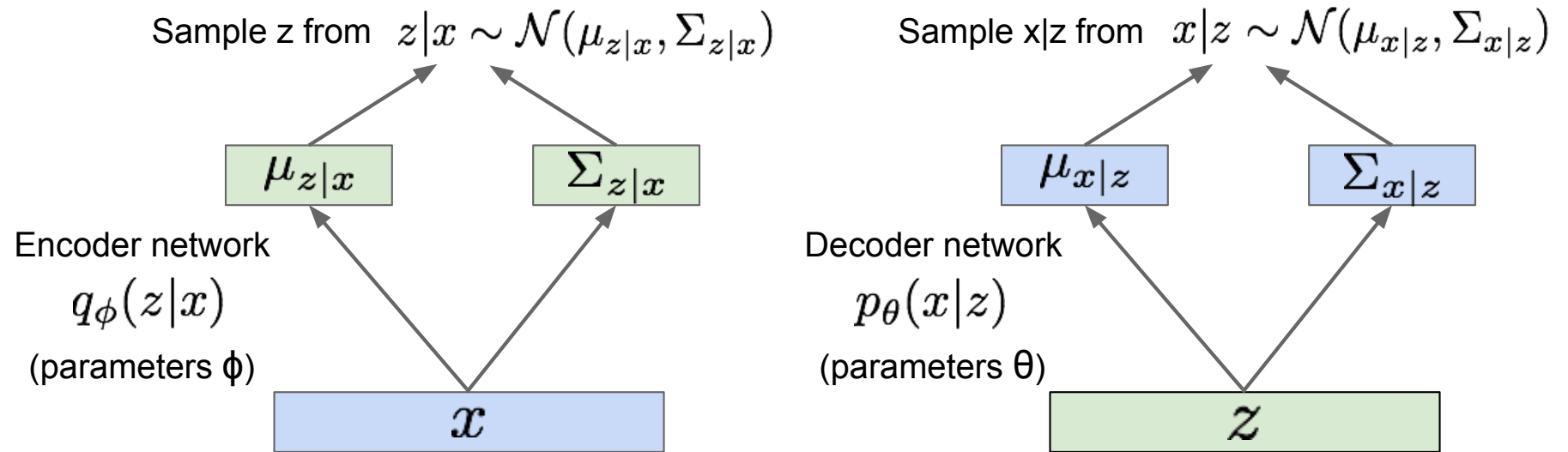
Posterior density also intractable:  $p_{\theta}(z|x) = p_{\theta}(x|z) p_{\theta}(z) / p_{\theta}(x)$

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

# 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

Assume that  $\Sigma_{x|z}$  and  $\Sigma_{z|x}$  are both diagonal, *i.e.* conditional independence.

# Variational Autoencoders

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

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

↑  
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  $\geq 0$ .

# Variational Autoencoders

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

$$\begin{aligned}\log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] && (p_{\theta}(x^{(i)})) \text{ Does not depend on } z \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] && (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] && (\text{Multiply by constant}) \\ &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] && (\text{Logarithms}) \\ &= \underbrace{\mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))}_{\geq 0}\end{aligned}$$

**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:

$$\begin{aligned}
 \log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] && (p_{\theta}(x^{(i)})) \text{ Does not depend on } z \\
 &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] && (\text{Bayes' Rule}) \\
 \text{Reconstruct the input data} &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] && (\text{Multiply by constant}) \\
 &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] && (\text{Logarithms}) \\
 &= \underbrace{\mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{> 0} + \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))}_{> 0}
 \end{aligned}$$

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

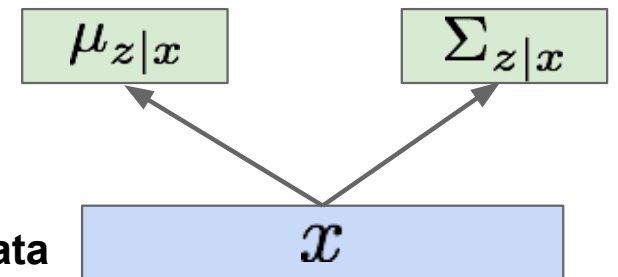
$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Make approximate posterior distribution close to prior

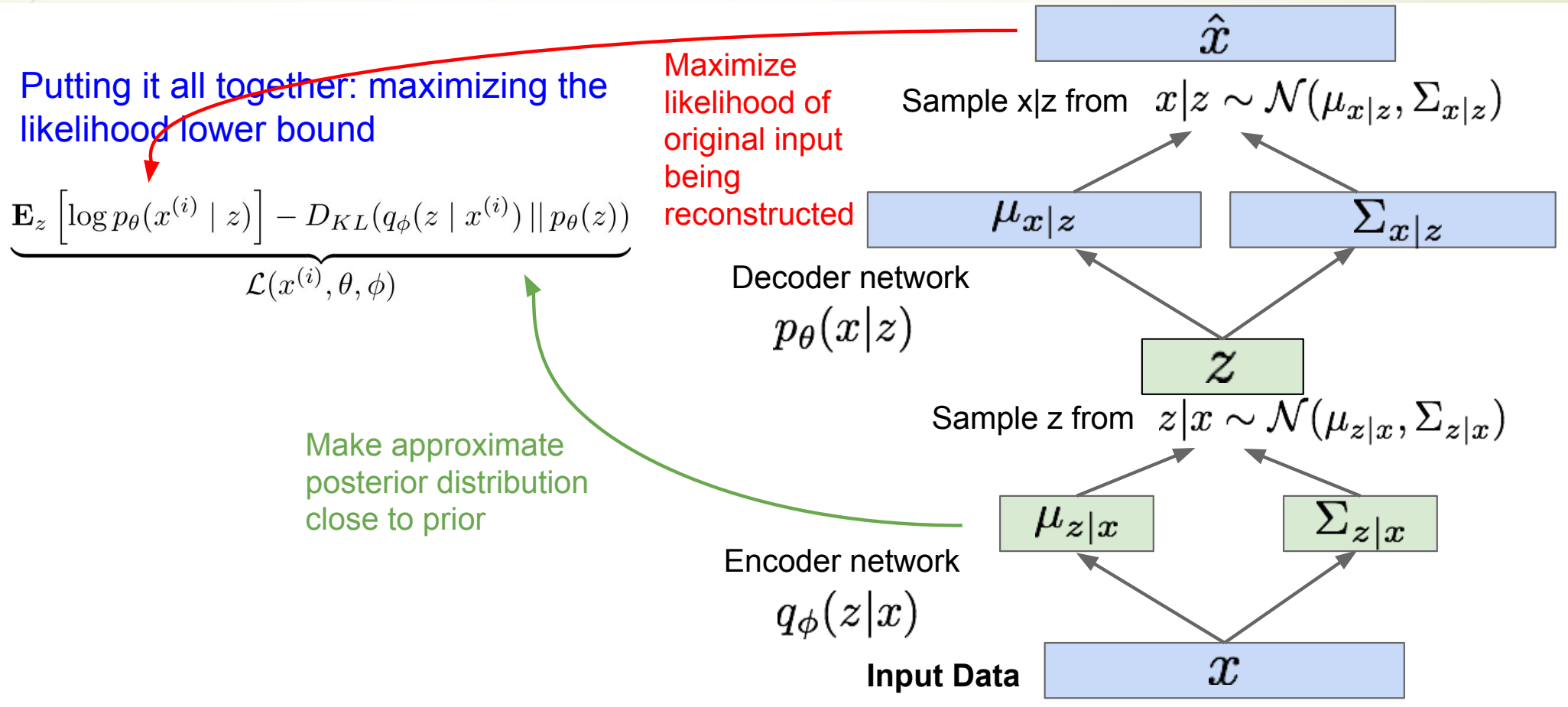
Encoder network

$$q_\phi(z | \mathbf{x})$$

Input Data



# Stage II in forward pass



# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

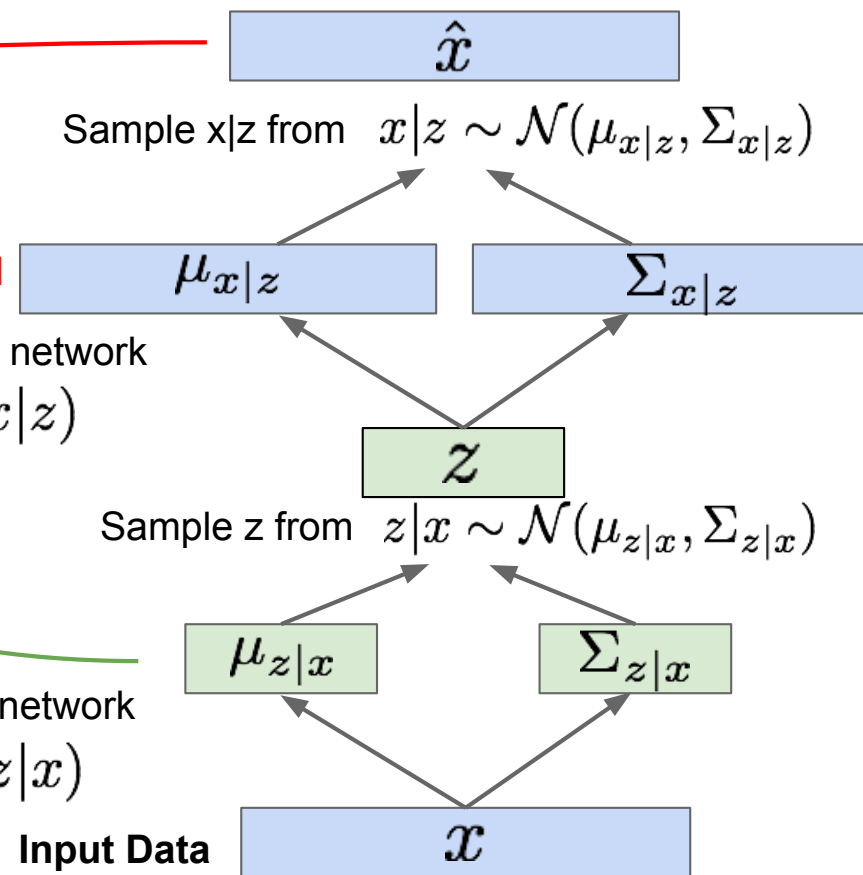
Make approximate posterior distribution close to prior

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

Maximize likelihood of original input being reconstructed

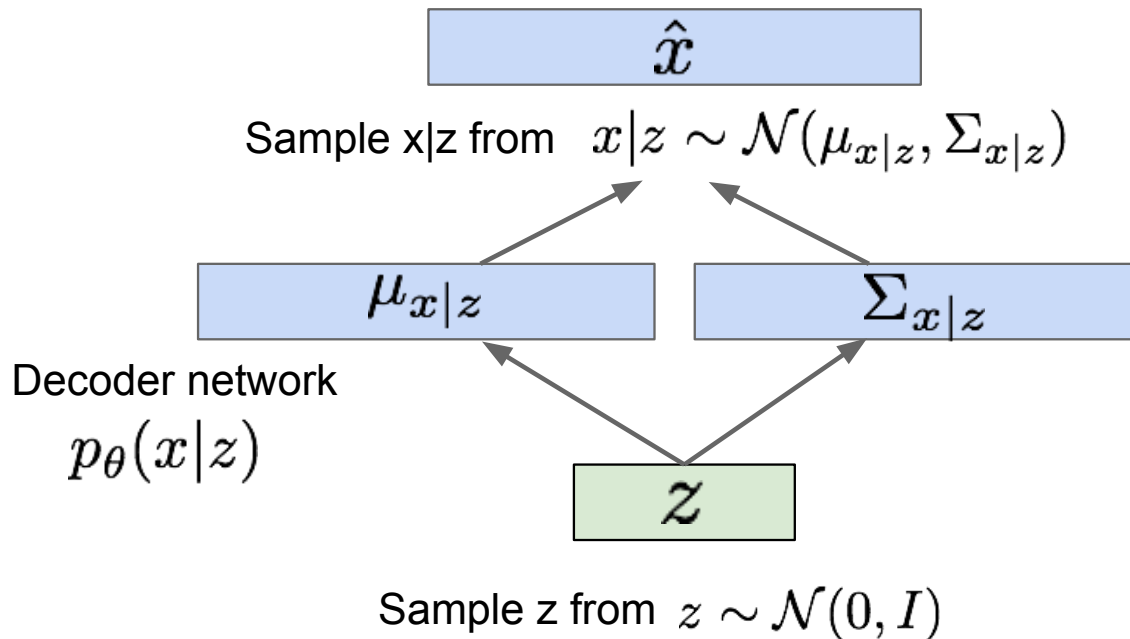
Decoder network  
 $p_\theta(x|z)$

Encoder network  
 $q_\phi(z|x)$

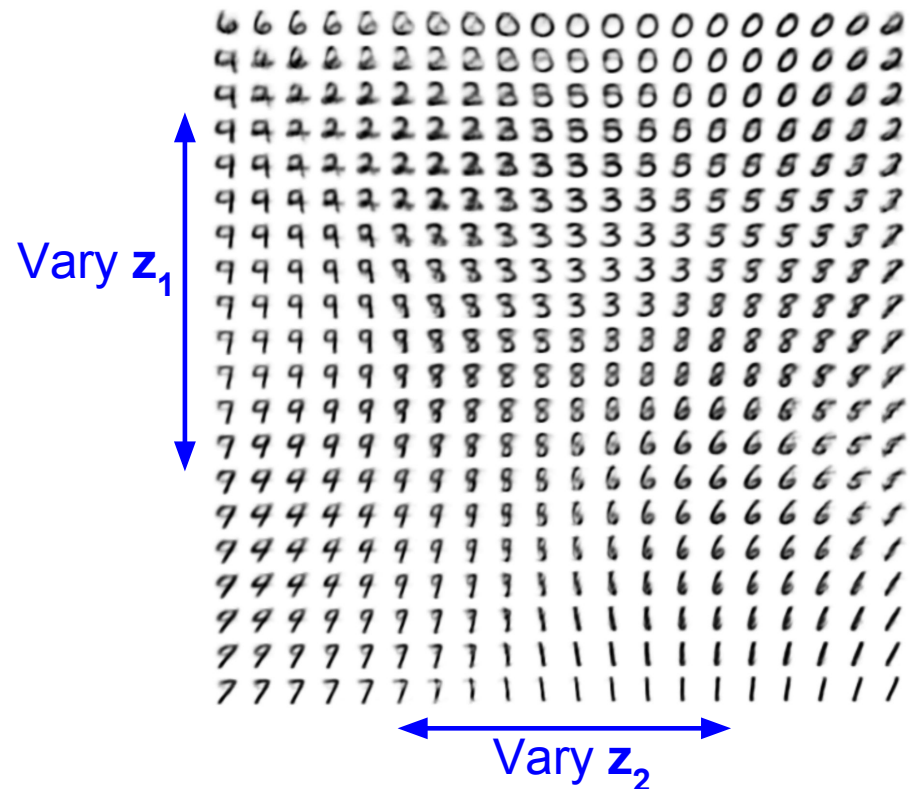


# VAE: generating data

Use decoder network. Now sample  $z$  from prior!



Data manifold for 2-d  $z$



# VAE: generating data

Diagonal prior on  $\mathbf{z}$   
=> independent  
latent variables

Different  
dimensions of  $\mathbf{z}$   
encode  
interpretable factors  
of variation

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

Degree of smile

Vary  $z_1$



Vary  $z_2$

Head pose



# VAE: Generating Data



32x32 CIFAR-10



Labeled Faces in the Wild

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



# Variational Autoencoders



- ▶ Probabilistic spin to traditional autoencoders => allows generating data  
Defines an intractable density => derive and optimize a (variational) lower bound
- ▶ **Pros:**
  - ▶ Principled approach to generative models
  - ▶ Allows inference of  $q(z|x)$ , can be useful feature representation for other tasks
- ▶ **Cons:**
  - ▶ Maximizes lower bound of likelihood
  - ▶ 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:

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

VAEs define intractable density function with latent  $\mathbf{z}$ :

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

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!

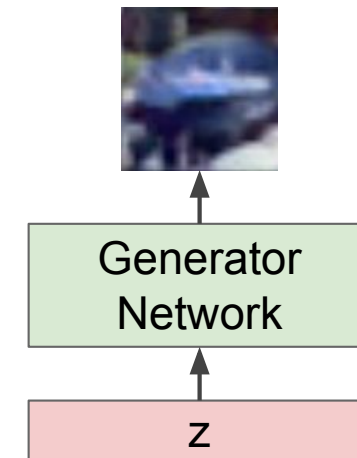
Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

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

A: A neural network!

Output: Sample from training distribution

Input: Random noise

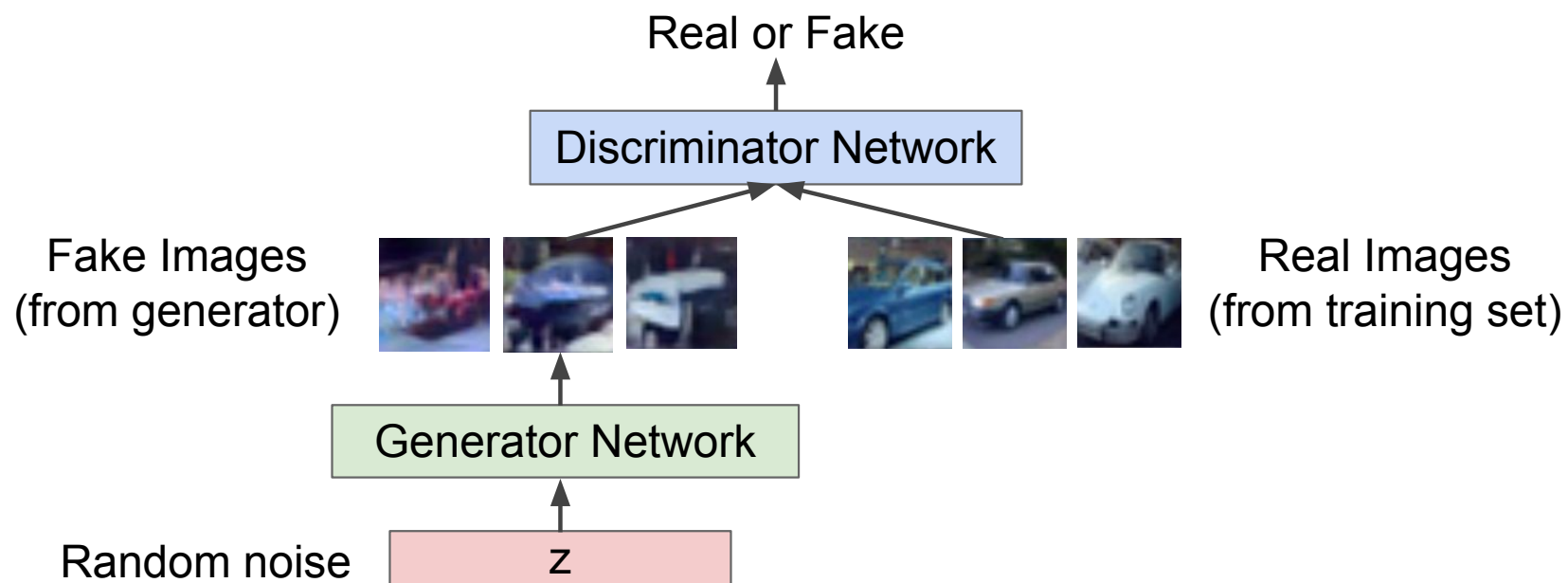


# Training GANs: Two-player game

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



# 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:

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

- Discriminator ( $\theta_d$ ) wants to **maximize objective** such that  $D(x)$  is close to 1 (real) and  $D(G(z))$  is close to 0 (fake)
- Generator ( $\theta_g$ ) 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

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

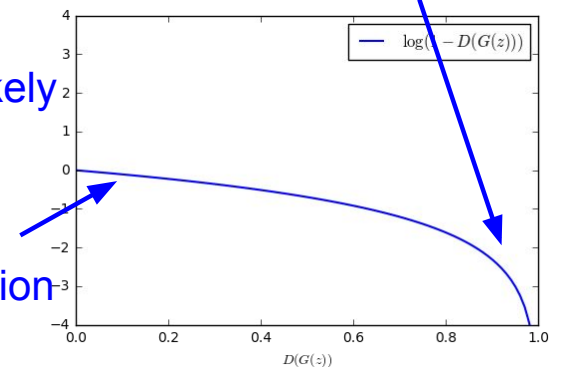
2. **Gradient descent** on generator

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

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

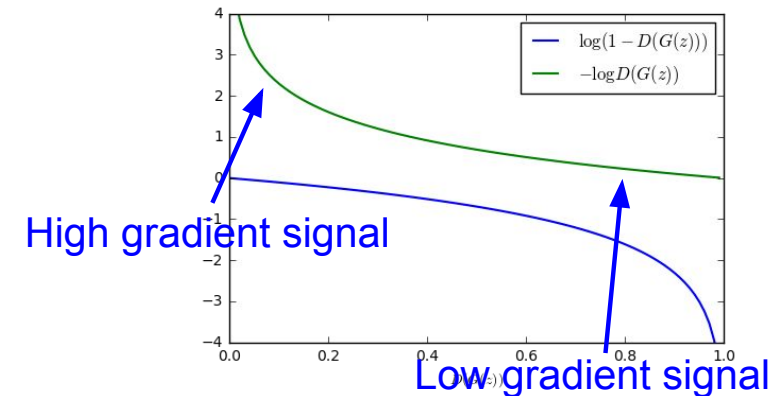
$$\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

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

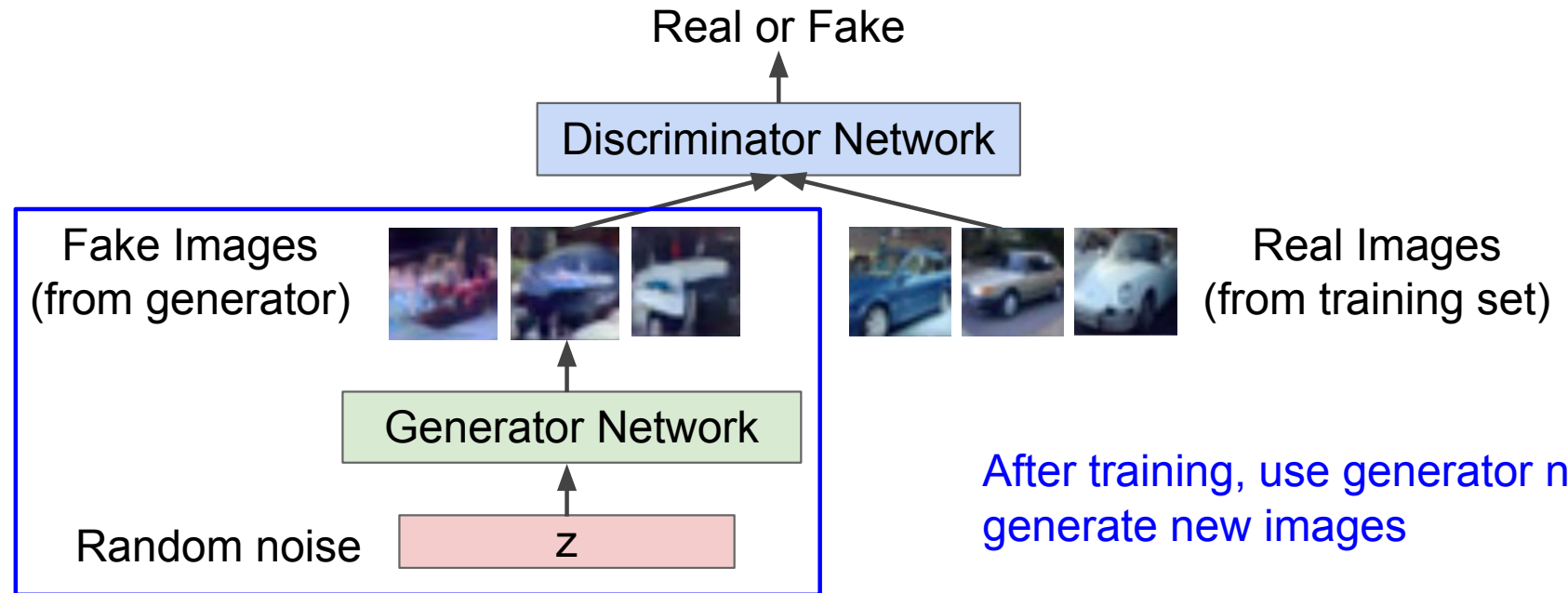
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

**end for**

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

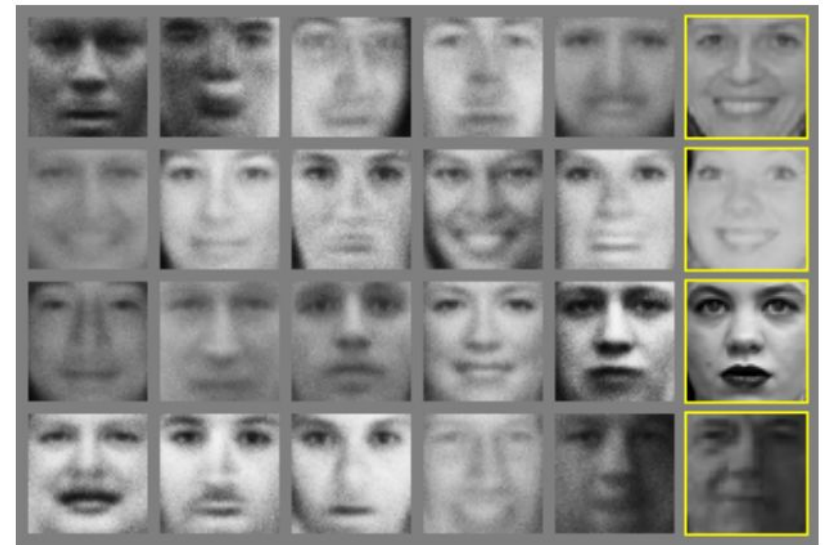
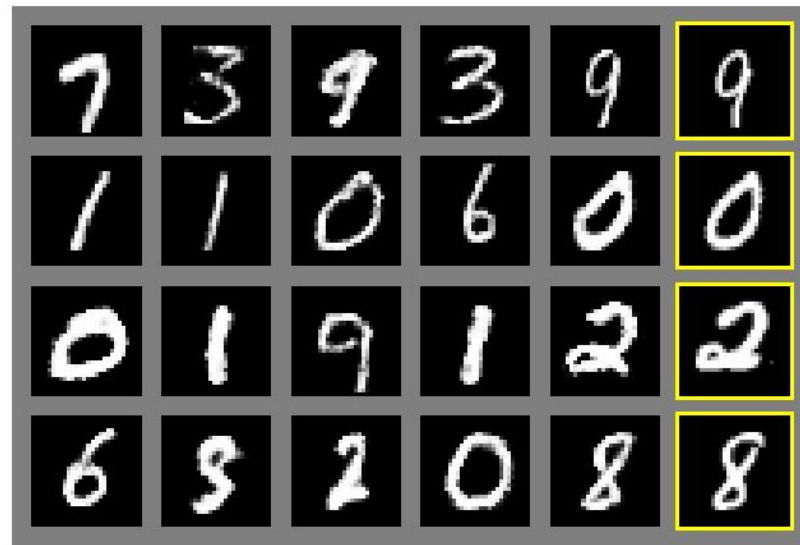
**Generator network:** try to fool the discriminator by generating real-looking images

**Discriminator network:** try to distinguish between real and fake images



# Generative Adversarial Nets

Generated samples



Nearest neighbor from training set



# Generative Adversarial Nets

Generated samples (CIFAR-10)



Nearest neighbor from training set



# Generative Adversarial Nets: Convolutional Architectures

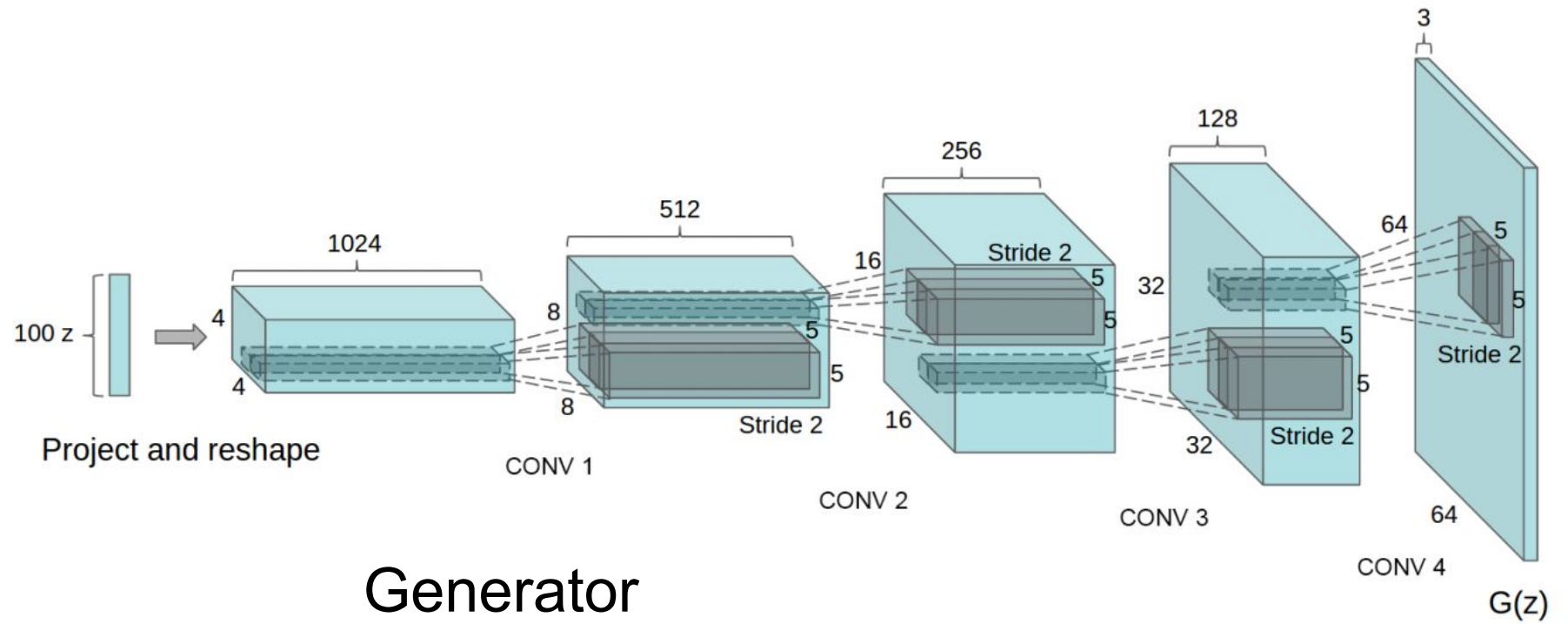
Generator is an upsampling network with fractionally-strided convolutions  
Discriminator is a convolutional network

## Architecture guidelines for stable Deep Convolutional GANs

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

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





## 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!

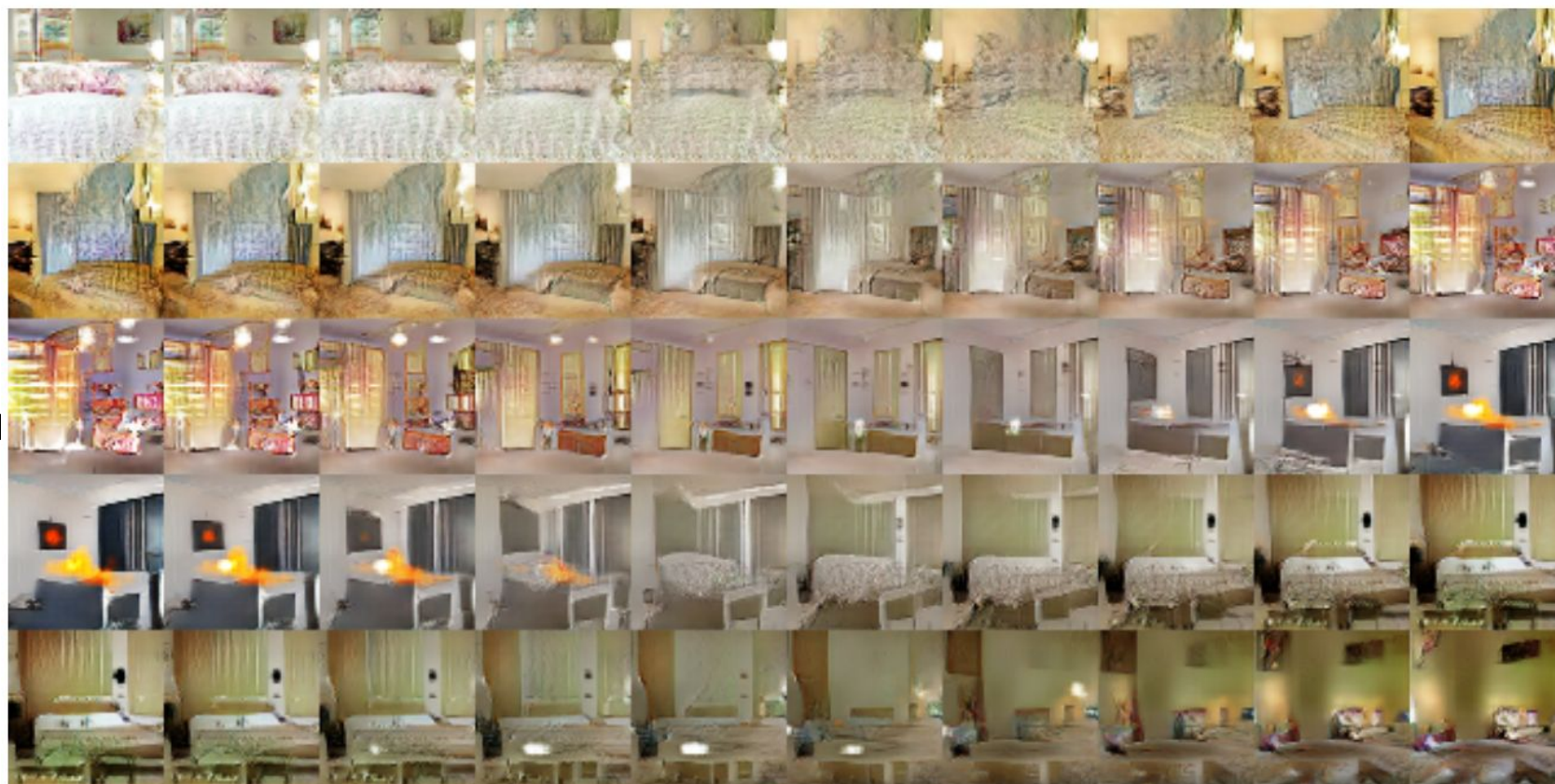


Radford et al,  
ICLR 2016



# Generative Adversarial Nets: Convolutional Architectures

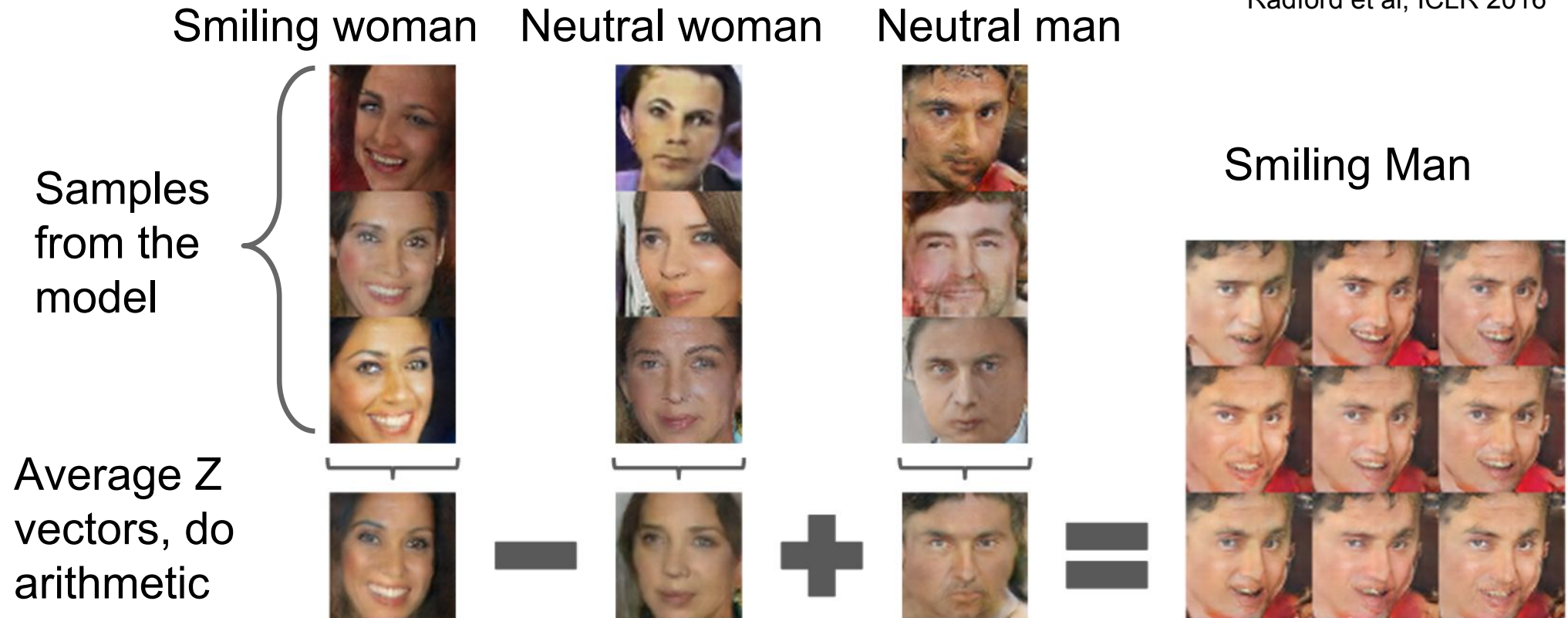
Interpolating  
between  
random  
points in latent  
space



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



-

+

=

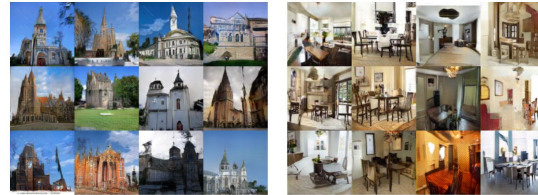
Woman with glasses



Radford et al,  
ICLR 2016

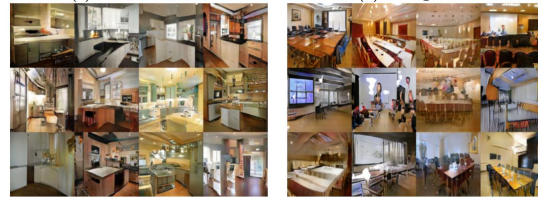
# 2017: Year of the GAN

## Better training and generation



(a) Church outdoor.

(b) Dining room.



(c) Kitchen.

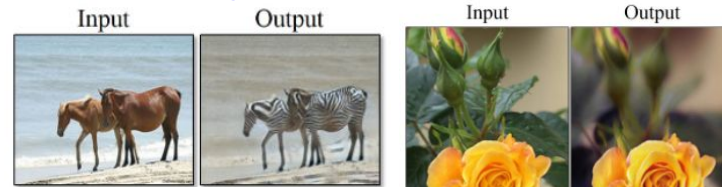
(d) Conference room.

LSGAN. Mao et al. 2017.



BEGAN. Bertholet et al. 2017.

## Source->Target domain transfer



horse → zebra



zebra → horse



apple → orange



→ summer Yosemite

→ winter Yosemite

CycleGAN. Zhu et al. 2017.

## Text -> Image Synthesis

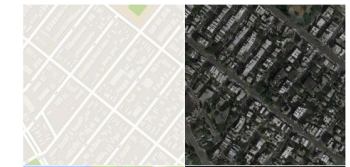
this small bird has a pink breast and crown, and black primaries and secondaries.

this magnificent fellow is almost all black with a red crest, and white cheek patch.



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



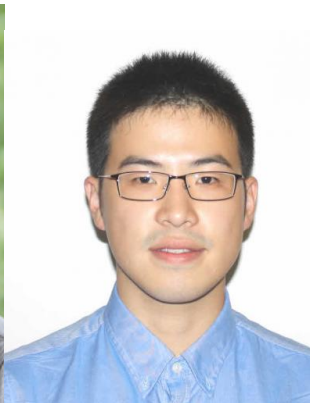
- ▶ Don't work with an explicit density function  
Take game-theoretic approach: learn to generate from training distribution through 2-player minimax zero-sum game
- ▶ **Pros:**
  - ▶ Beautiful, state-of-the-art samples!
- ▶ **Cons:**
  - ▶ Trickier / more unstable to train
  - ▶ Can't solve inference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

# Robust Estimation and GANs

Chao GAO, Jiyi LIU, Y.Y., and Weizhi ZHU



*Chao Gao* (Chicago)

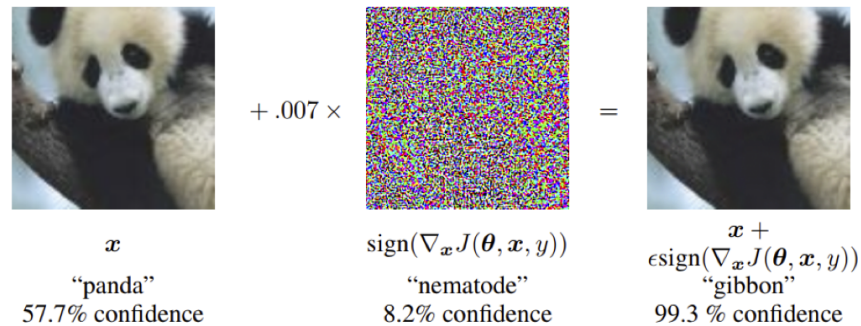


*Jiyu Liu* (Yale)



*Weizhi Zhu* (HKUST)

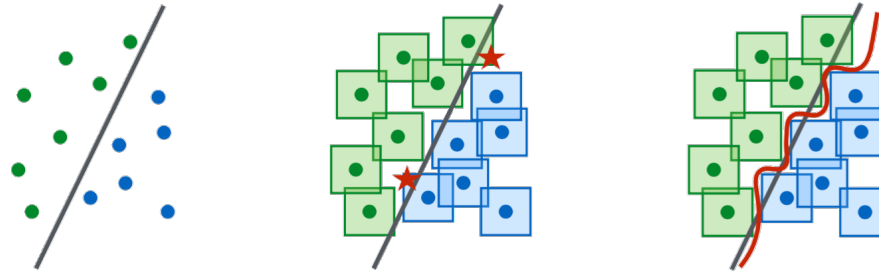
# Deep Learning is Notoriously Not Robust!



[Goodfellow et al., 2014]

- Imperceivable adversarial examples are ubiquitous to fail neural networks
- How can one achieve **robustness**?

# Robust Optimization



- Traditional training:

$$\min_{\theta} J_n(\theta, \mathbf{z} = (x_i, y_i)_{i=1}^n)$$

- e.g. square or cross-entropy loss as negative log-likelihood of logit models

- Robust optimization:

$$\min_{\theta} \max_{\|\epsilon_i\| \leq \delta} J_n(\theta, \mathbf{z} = (x_i + \epsilon_i, y_i)_{i=1}^n)$$

- robust to any distributions, yet perhaps too conservative



# Distributional Robust Optimization

- Distributional Robust Optimization:

$$\min_{\theta} \max_{\epsilon} \mathbb{E}_{\mathbf{z} \sim P_{\epsilon} \in \mathcal{D}} [J_n(\theta, \mathbf{z})]$$

- $\mathcal{D}$  is a set of ambiguous distributions, e.g. Wasserstein ambiguity set
- intermediate approach with statistically contaminated distributions
- *sometimes, contamination might be unstructured...*

# Huber's Model

$$X_1, \dots, X_n \sim (1 - \epsilon)P_\theta + \epsilon Q$$

contamination proportion

arbitrary contamination

parameter of interest

*[Huber 1964]*



# An Example

$$X_1, \dots, X_n \sim (1 - \epsilon)N(\theta, I_p) + \epsilon Q.$$

**how to estimate ?**

# Robust Maximum-Likelihood Does not work!

$$X_1, \dots, X_n \sim (1 - \epsilon)N(\theta, I_p) + \epsilon Q.$$

$$\begin{aligned} \ell(\theta, Q) &= \text{negative log-likelihood} = \sum_{i=1}^n (\theta - X_i)^2 \\ &\sim (1 - \epsilon)\mathbb{E}_{\mathcal{N}(\theta)}(\theta - X)^2 + \epsilon\mathbb{E}_Q(\theta - X)^2 \end{aligned}$$

the sample mean

$$\hat{\theta}_{mean} = \frac{1}{n} \sum_{i=1}^n X_i = \arg \min_{\theta} \ell(\theta, Q)$$

$$\min_{\theta} \max_Q \ell(\theta, Q) \geq \max_Q \min_{\theta} \ell(\theta, Q) = \max_Q \ell(\hat{\theta}_{mean}, Q) = \infty$$

# Medians

## 1. Coordinatewise median

$$\hat{\theta} = (\hat{\theta}_j), \text{ where } \hat{\theta}_j = \text{Median}(\{X_{ij}\}_{i=1}^n);$$

## 2. Tukey's median

$$\hat{\theta} = \arg \max_{\eta \in \mathbb{R}^p} \min_{\|u\|=1} \frac{1}{n} \sum_{i=1}^n \mathbb{I}\{u^T X_i > u^T \eta\}.$$

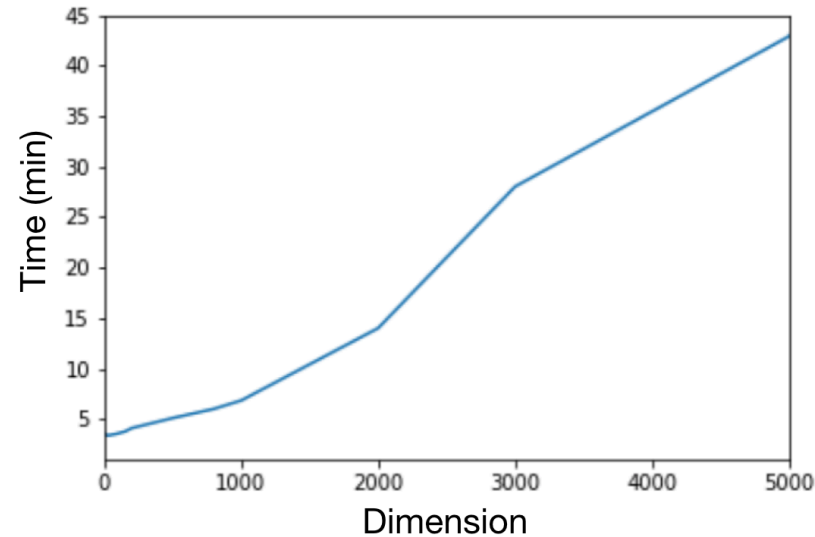
# Comparisons

	Coordinatewise Median	Tukey's Median
breakdown point	$1/2$	$1/3$
statistical precision (no contamination)	$\frac{p}{n}$	$\frac{p}{n}$
statistical precision (with contamination)	$\frac{p}{n} + p\epsilon^2$	$\frac{p}{n} + \epsilon^2$ : minimax [Chen-Gao-Ren'15]
computational complexity	Polynomial	NP-hard [Amenta et al. '00]

Note: R-package for Tukey median can not deal with more than **10** dimensions!

[<https://github.com/ChenMengjie/DepthDescent>]

# Generative Adversarial Networks [Goodfellow et al. 2014]



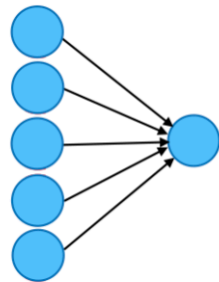
Note: R-package for Tukey median can not deal with more than 10 dimensions [<https://github.com/ChenMengjie/DepthDescent>]

# JS-GAN

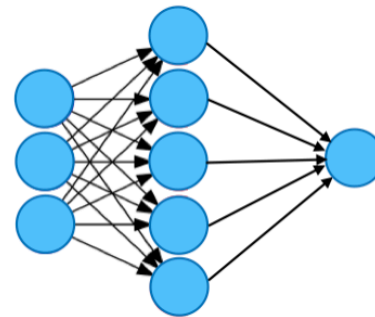
$$\hat{\theta} = \operatorname{argmin}_{\eta \in \mathbb{R}^p} \max_{T \in \mathcal{T}} \left[ \frac{1}{n} \sum_{i=1}^n \log T(X_i) + E_{\eta} \log(1 - T(X)) \right] + \log 4$$

**numerical  
experiment**

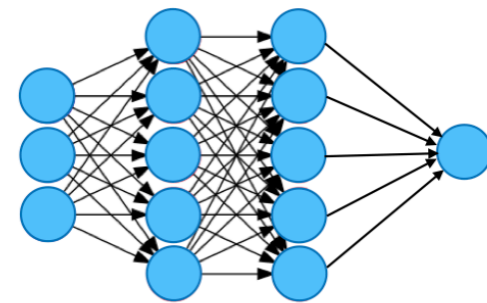
$$X_1, \dots, X_n \sim (1 - \epsilon)N(\theta, I_p) + \epsilon N(\tilde{\theta}, I_p)$$



$$\hat{\theta} \approx (1 - \epsilon)\theta + \epsilon\tilde{\theta}$$



$$\hat{\theta} \approx \theta$$



$$\hat{\theta} \approx \theta$$



# JS-GAN

**A classifier with hidden layers leads to robustness. Why?**

$$\text{JS}_g(\mathbb{P}, \mathbb{Q}) = \max_{w \in \mathbb{R}^d} \left[ \mathbb{P} \log \frac{1}{1 + e^{-w^T g(X)}} + \mathbb{Q} \log \frac{1}{1 + e^{w^T g(X)}} \right] + \log 4.$$

**Proposition.**

$$\text{JS}_g(\mathbb{P}, \mathbb{Q}) = 0 \iff \mathbb{P}g(X) = \mathbb{Q}g(X)$$

# JS-GAN

$$\hat{\theta} = \operatorname{argmin}_{\eta \in \mathbb{R}^p} \max_{T \in \mathcal{T}} \left[ \frac{1}{n} \sum_{i=1}^n \log T(X_i) + E_{\eta} \log(1 - T(X)) \right] + \log 4$$

**Theorem [GLYZ18].** For a neural network class  $\mathcal{T}$  with at least one hidden layer and appropriate regularization, we have

$$\|\hat{\theta} - \theta\|^2 \lesssim \begin{cases} \frac{p}{n} + \epsilon^2 & \text{(indicator/sigmoid/ramp)} \\ \frac{p \log p}{n} + \epsilon^2 & \text{(ReLUs+sigmoid features)} \end{cases}$$

with high probability uniformly over  $\theta \in \mathbb{R}^p, Q$ .

# JS-GAN

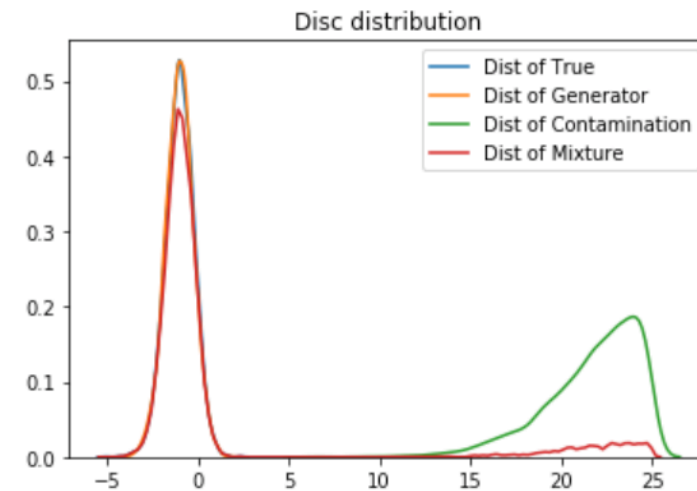
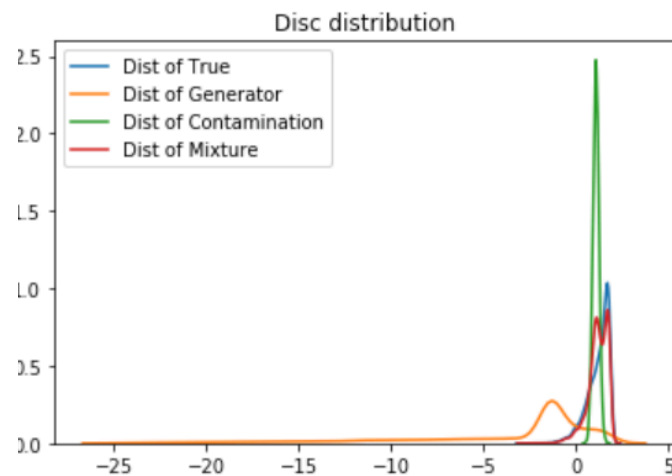
**unknown  
covariance?**

$$X_1, \dots, X_n \sim (1 - \epsilon)N(\theta, \Sigma) + \epsilon Q$$

$$(\hat{\theta}, \hat{\Sigma}) = \operatorname{argmin}_{\eta, \Gamma} \max_{T \in \mathcal{T}} \left[ \frac{1}{n} \sum_{i=1}^n \log T(X_i) + \mathbb{E}_{X \sim N(\eta, \Gamma)} \log(1 - T(X)) \right]$$

no need to change the discriminator class

# Discriminator identifies outliers



$$(1 - \epsilon)N(0_p, I_p) + \epsilon Q$$

$$N(5 * 1_p, I_p)$$

- Discriminator helps identify outliers or contaminated samples
- Generator fits uncontaminated portion of true samples



# Reference



- ▶ One can robustly estimate *mean* and *covariance (scatter)* matrix, for the general family of **Elliptical Distributions** (including Cauchy Distributions whose mean and moments do not exist)
- ▶ **Gao, Liu, Yao, Zhu**, *Robust Estimation and Generative Adversarial Networks*, ICLR 2019, <https://arxiv.org/abs/1810.02030>
- ▶ **Gao, Yao, Zhu**, *Generative Adversarial Networks for Robust Scatter Estimation: A Proper Scoring Rule Perspective*, <https://arxiv.org/abs/1903.01944>



Thank you!

