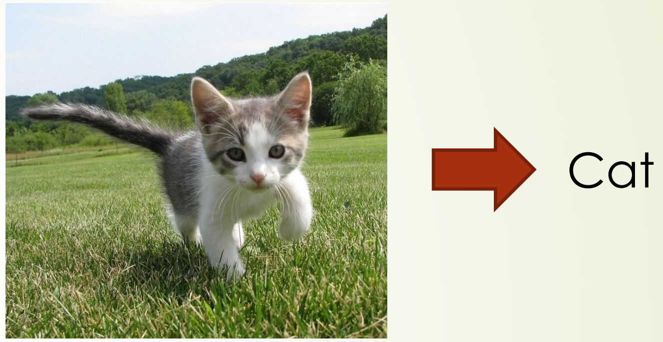
Unsupervised Learning: PCA, Clustering, AutoEncoder, and Generative Adversarial Networks

Yuan YAO

HKUST

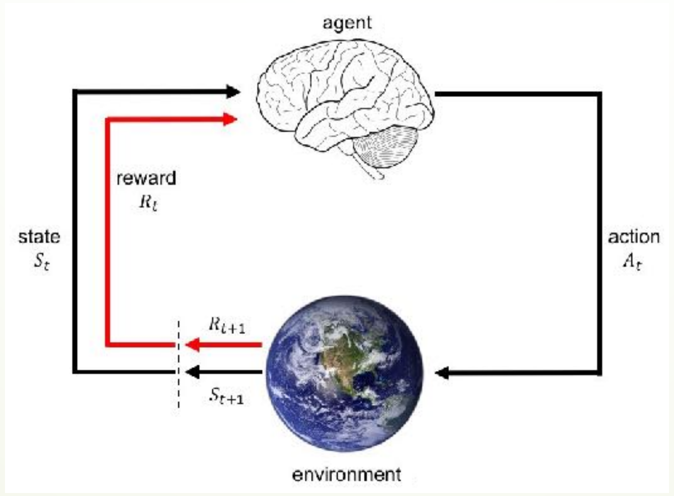
Supervised Learning

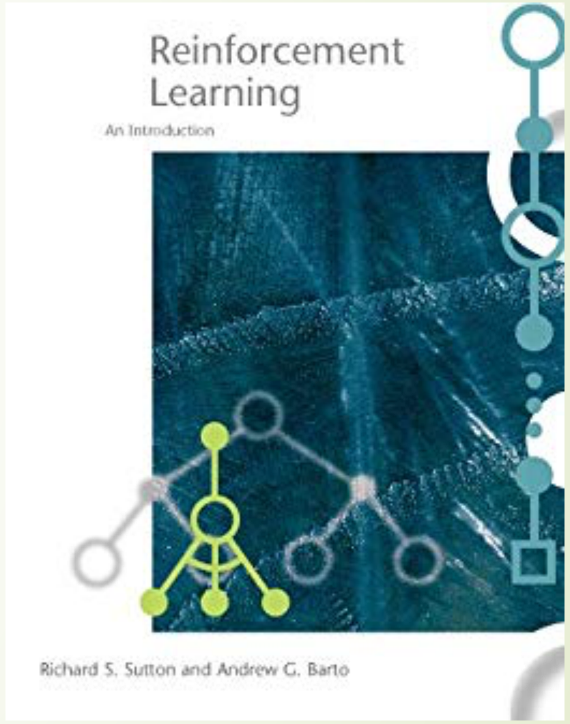
- **Data**: (x, y)
  x is input, y is output/response (label)
- **Goal**: Learn a *function* to map x -> y
- **Examples**:
  - Classification,
  - regression,
  - object detection,
  - semantic segmentation,
  - image captioning, etc.



Reinforcement Learning

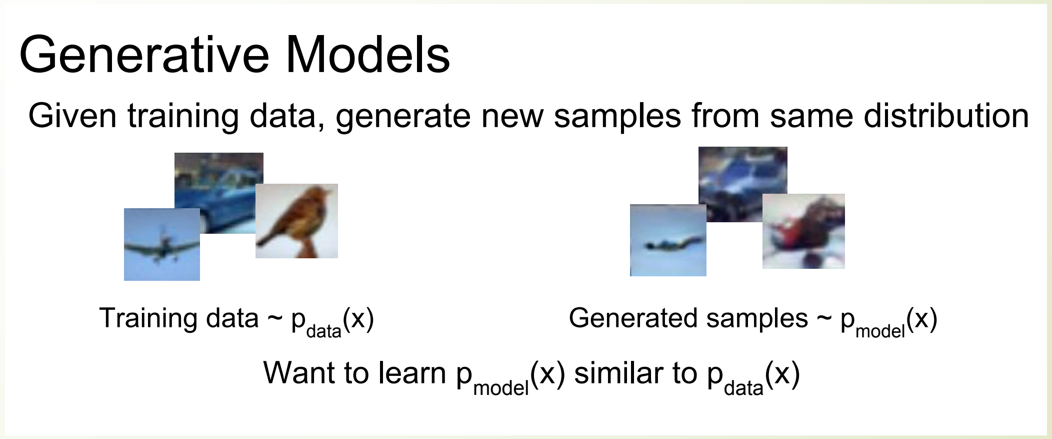
- Problems involving an **agent**
- interacting with an **environment**,
- which provides numeric **reward** signals
- **Goal**:
  - Learn how to take actions in order to maximize reward in dynamic scenarios





Today: Unsupervised Learning

- **Data**: x
  Just input data, no output labels!
- **Goal**: Learn some underlying hidden *structure* of the data
- **Examples**:
  - Clustering,
  - dimensionality reduction (manifold learning),
  - Density (probability) estimation,
  - Generative models:
    - Autoencoder
    - GANs, etc.



Generative Models

Given training data, generate new samples from same distribution

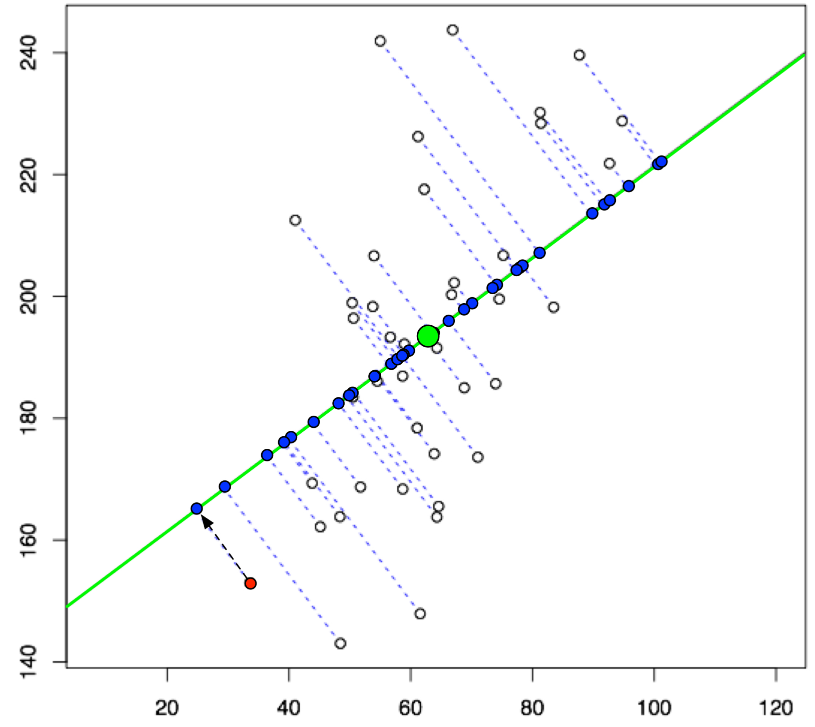
Training data ~ $p_{data}(x)$

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

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

PCA: Principal Component Analysis

Can you find a low dimensional affine representation?



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

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

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

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

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, $z_{ik} = x_{i1}e_{1k} + x_{i2}e_{2k} + \ldots + x_{ip}e_{pk}$ are the principle component scores of the i-th observation.
- $\hat{\lambda}_k$ measures the importance of the k-th PC.
- $\hat{\lambda}_k / (\hat{\lambda}_1 + \ldots + \hat{\lambda}_p) = \hat{\lambda}_k / trace(\mathbf{S})$ 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 $(\hat{\mathbf{e}}_k)$ 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, $\hat{\mathbf{e}}_1$ and $\hat{\mathbf{e}}_2$, for the USArrests data. These are also displayed in Figure 10.1.

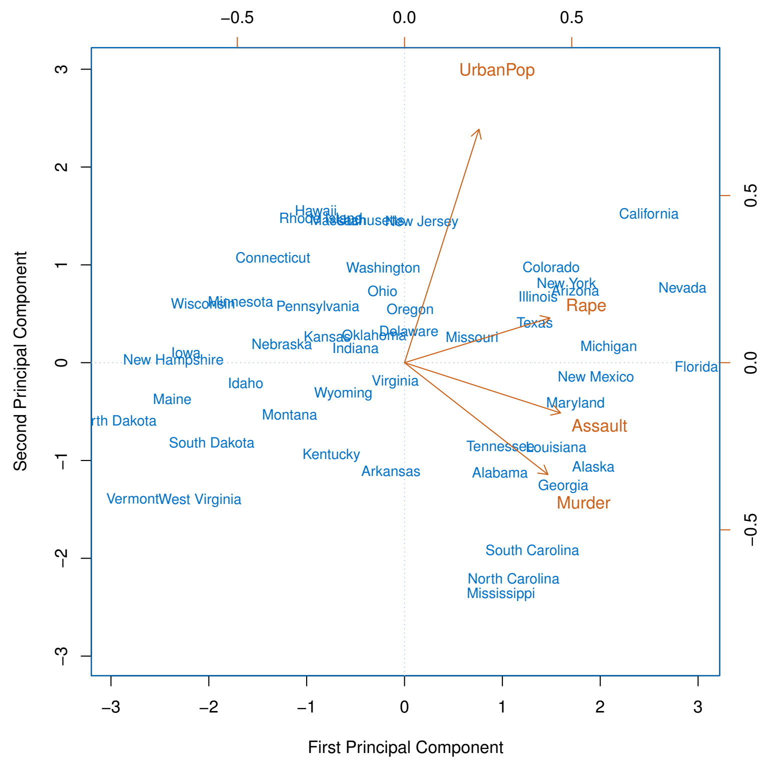


Figure: 10.1. Next page

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 kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.
  2. Assign each observation to the cluster whose centroid is closest (where closest is defined using Euclidean distance).

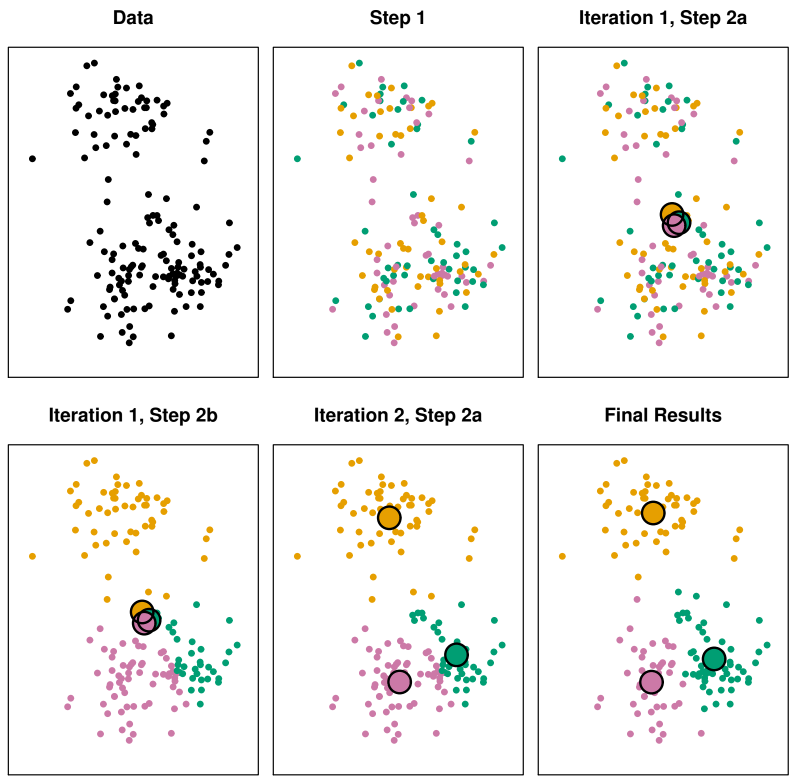
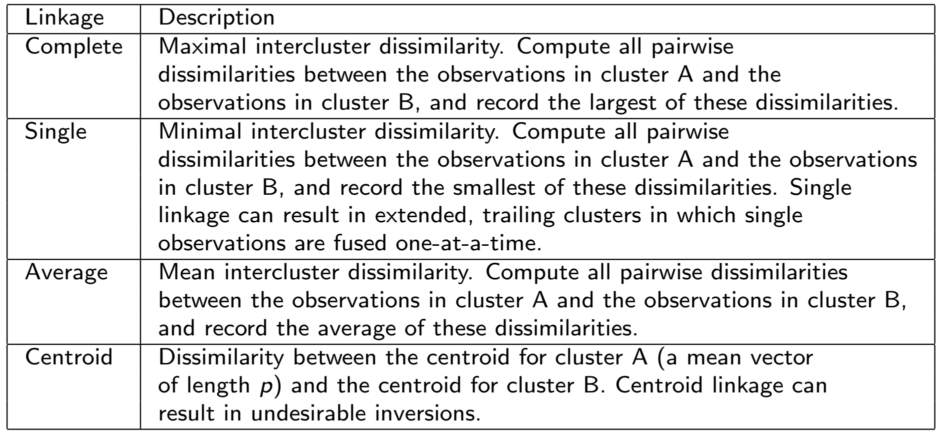


Figure: 10.6

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.

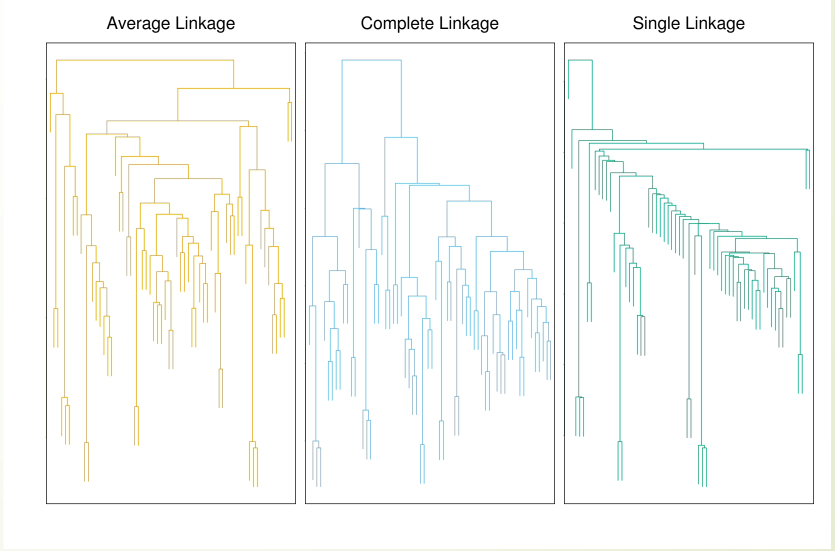
Hierarchical Clustering Algorithms (Agglomerative)

- 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 $i = n, n-1, \ldots 2$:
    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

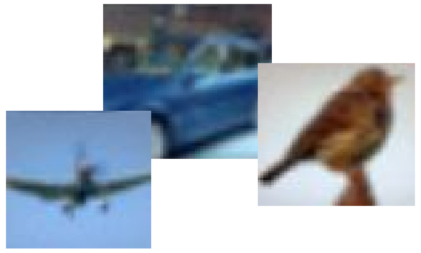


Manifold Learning: Nonlinear Dimensionality Reduction

- 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



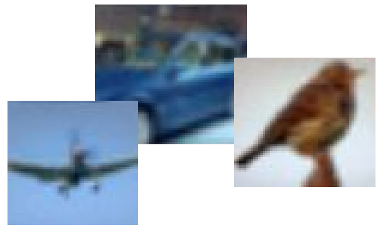
Training data ~ $p_{data}(x)$

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 ~ $p_{data}(x)$

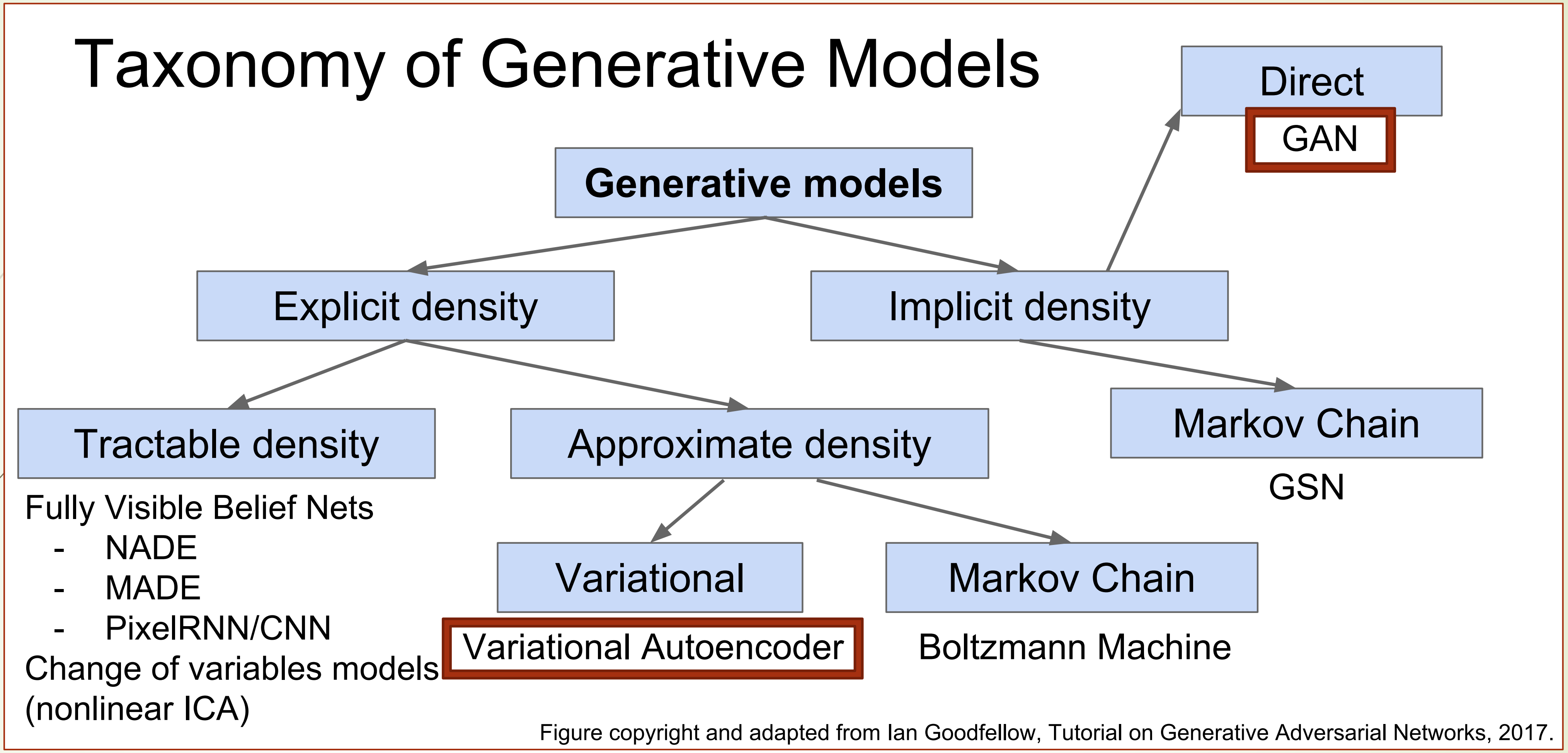
Generated samples ~ $p_{model}(x)$

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



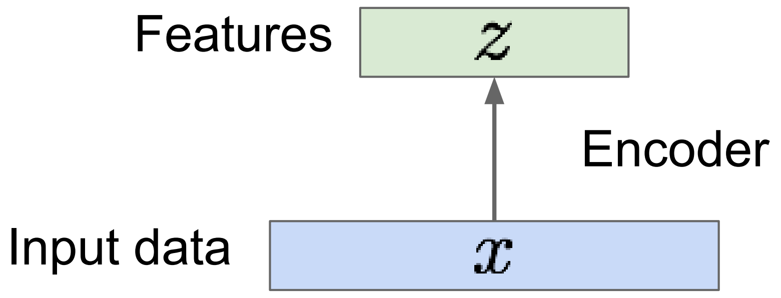
- We are going to focus on:
  - Variational AutoEncoder (VAE)
  - Generative Adversarial Network (GAN)

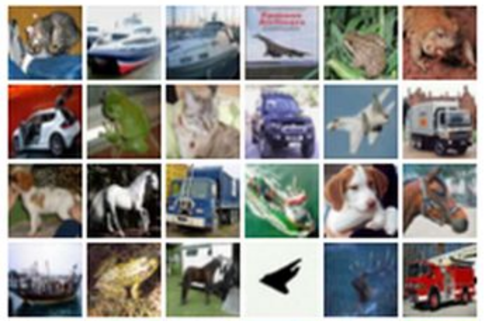
Variational Autoencoders (VAE)

Some background first: Autoencoders

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

e.g. PCA, Manifold Learning, Dictionary Learning

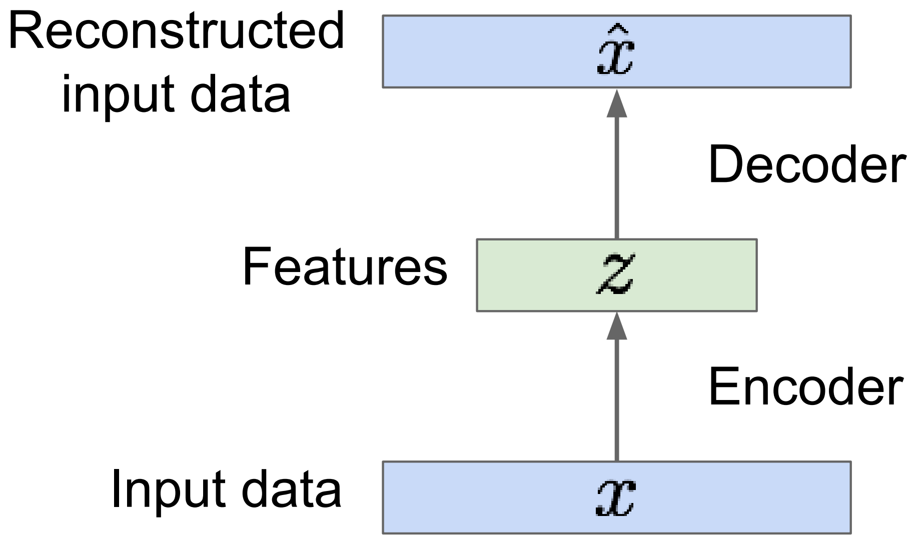




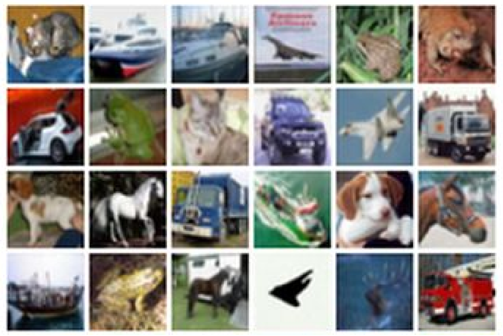
How to learn this feature representation?

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 Autoencoder

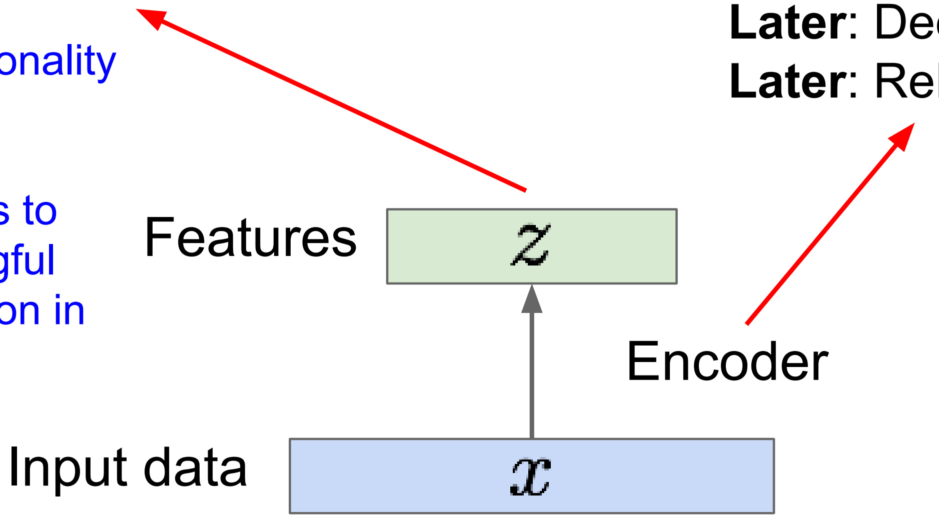
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

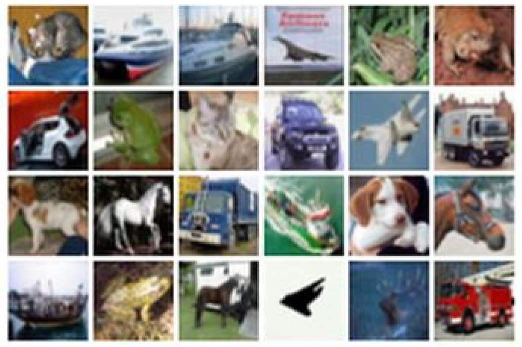
z usually smaller than **x** (dimensionality reduction)

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



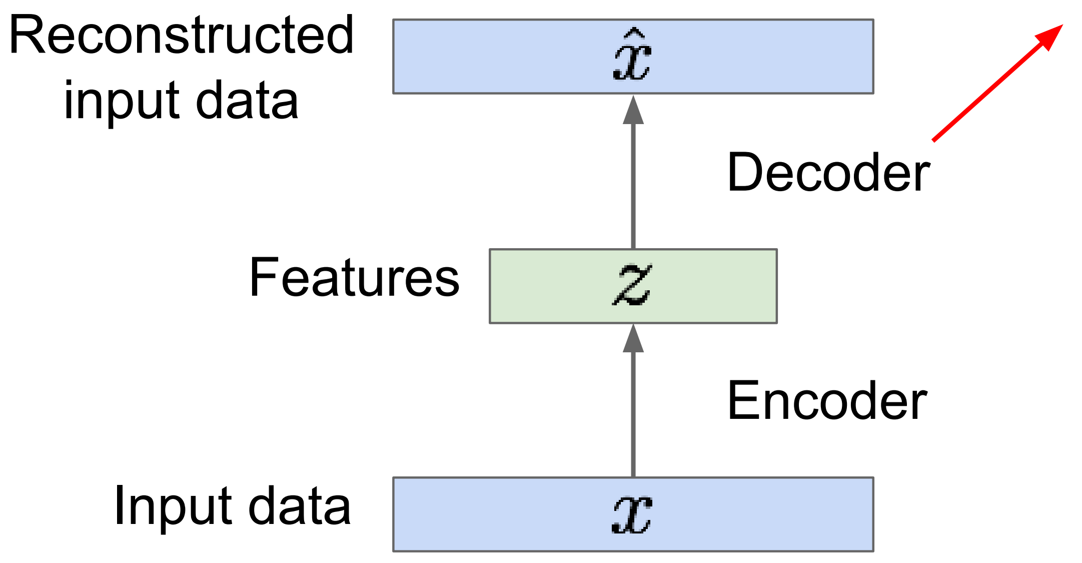


Deep Learning for decoders

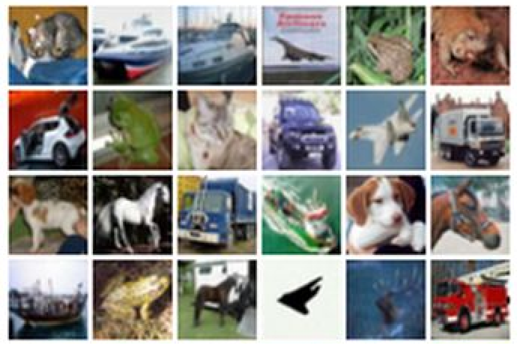
How to learn this feature representation?

Train such that features can be used to reconstruct original data

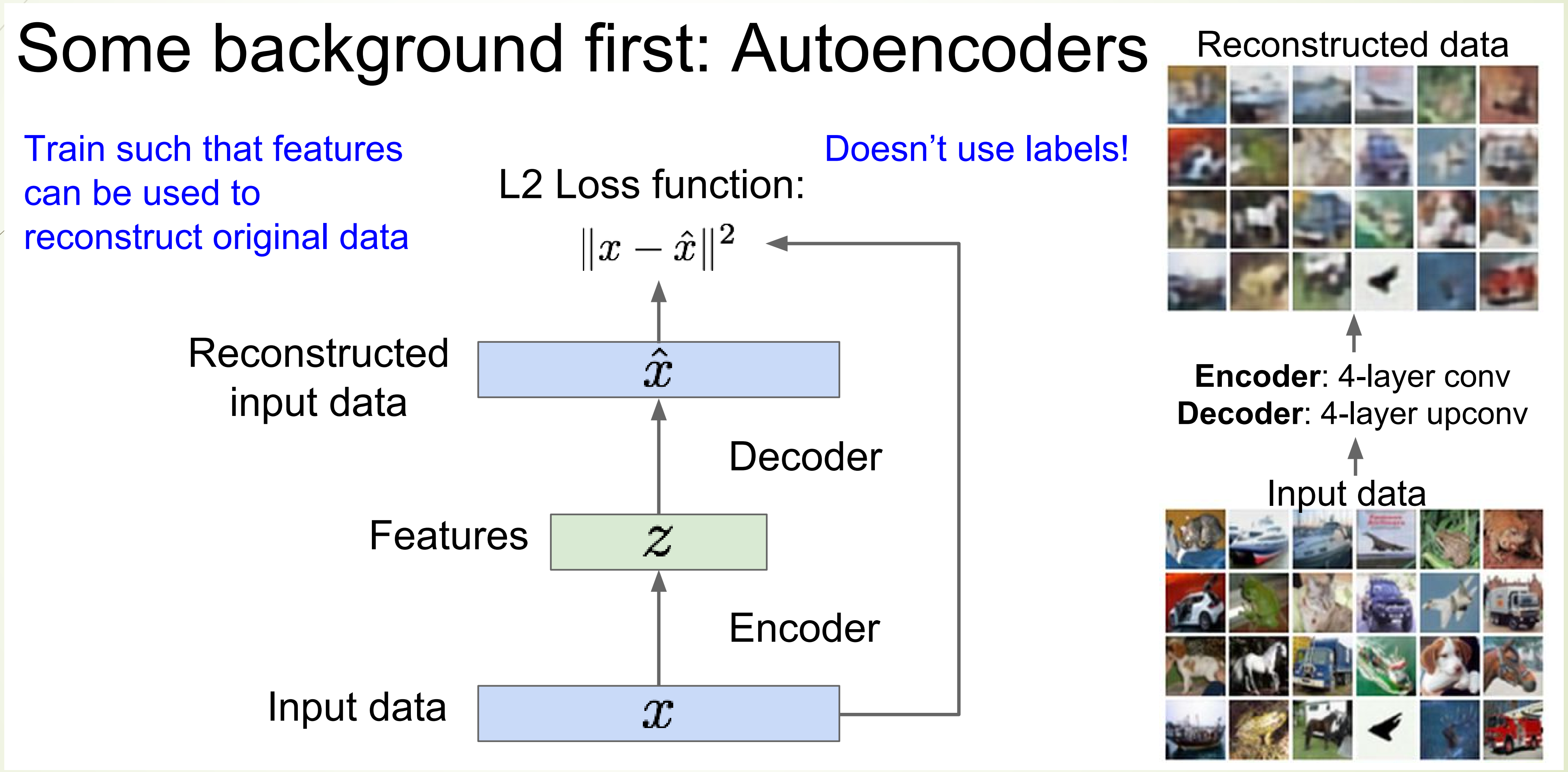
“Autoencoding” - encoding itself



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



L2 Loss functions



Some background first: Autoencoders

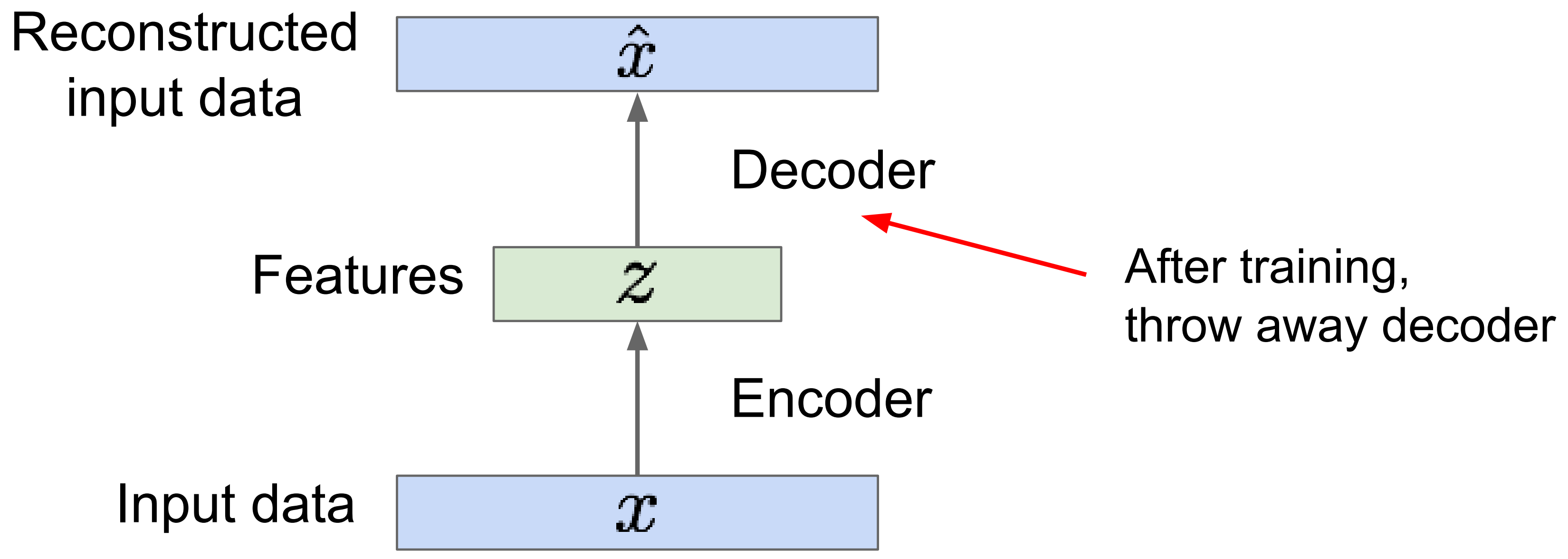
Train such that features can be used to reconstruct original data

Doesn't use labels!

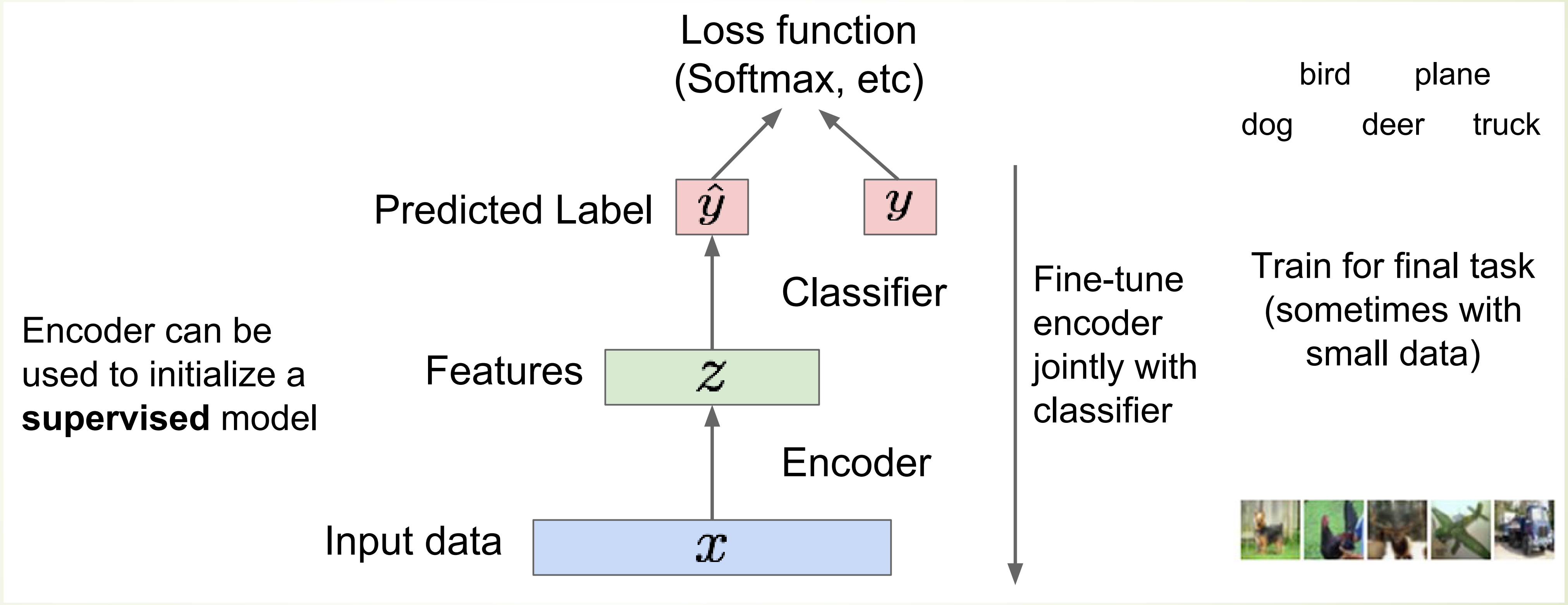
L2 Loss function:

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

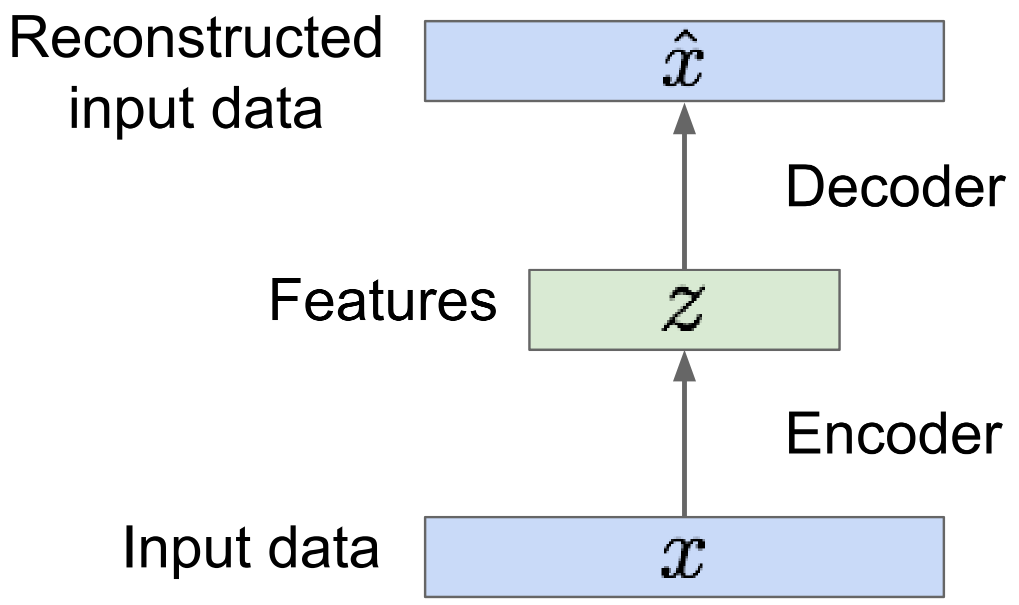
Some background first: Autoencoders



Autoencoders for Transfer Learning



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



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

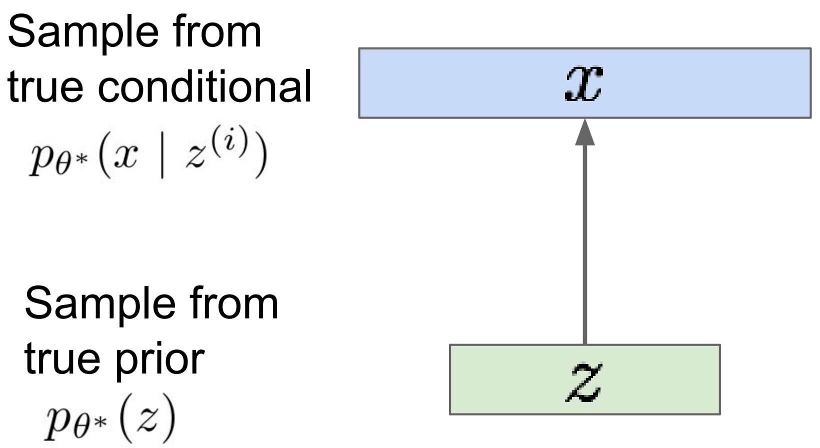
Features capture factors of variation in training data. Can we generate new images from an autoencoder?

Variational Autoencoders

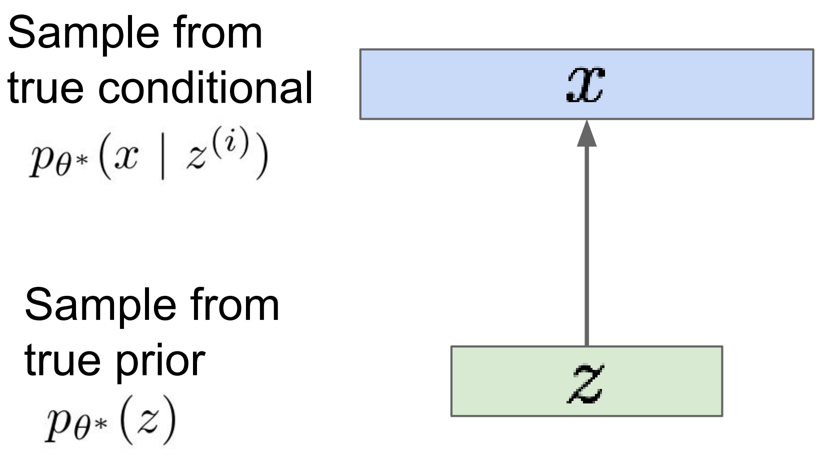
Probabilistic spin on autoencoders - will let us sample from the model to generate data!

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

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



Variational Autoencoders



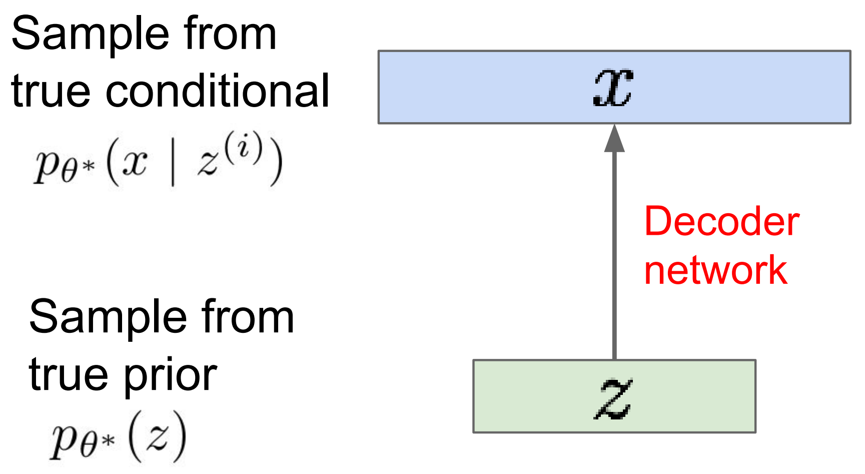
We want to estimate the true parameters θ^* of this generative model.

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



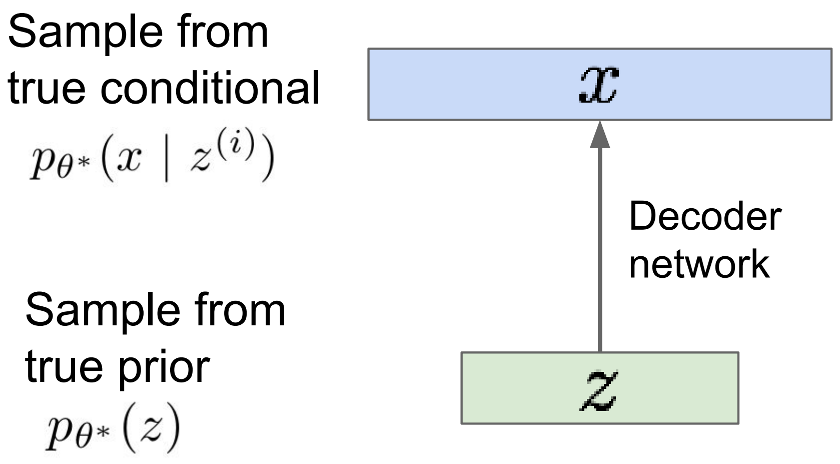
We want to estimate the true parameters θ^* of this generative model.

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

Variational Autoencoders



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

How to train the model?

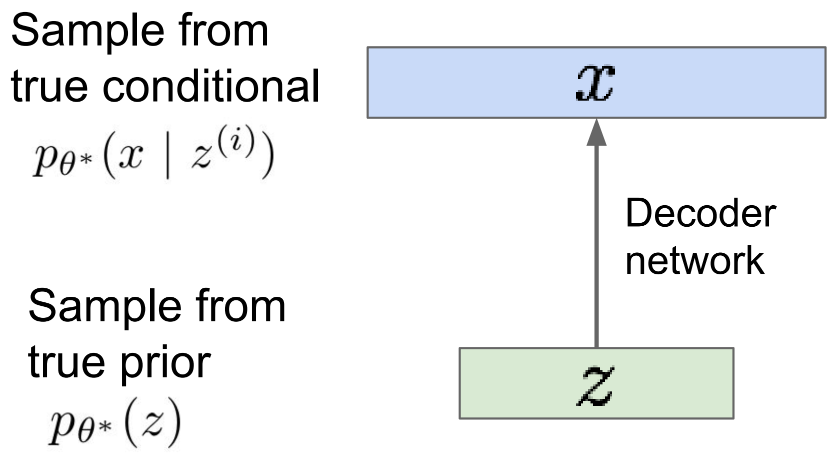
Remember strategy for training generative models from FVBNs. Learn model parameters to maximize likelihood of training data

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

Now with latent z

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

Variational Autoencoders



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: $p_\theta(x) = \overset{\text{☹}}{\int} \overset{\checkmark}{p_\theta(z)} \overset{\checkmark}{p_\theta(x|z)} dz$

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

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

Intractable data likelihood

Variational Lower Bounds

Data likelihood: $p_\theta(x) = \overset{\text{😟}}{\int} \overset{\checkmark}{p_\theta(z)}\,\overset{\checkmark}{p_\theta(x|z)}\,dz$

Posterior density also intractable: $p_\theta(z|x) = \overset{\checkmark}{p_\theta(x|z)}\,\overset{\checkmark}{p_\theta(z)}/\overset{\text{😟}}{p_\theta(x)}$

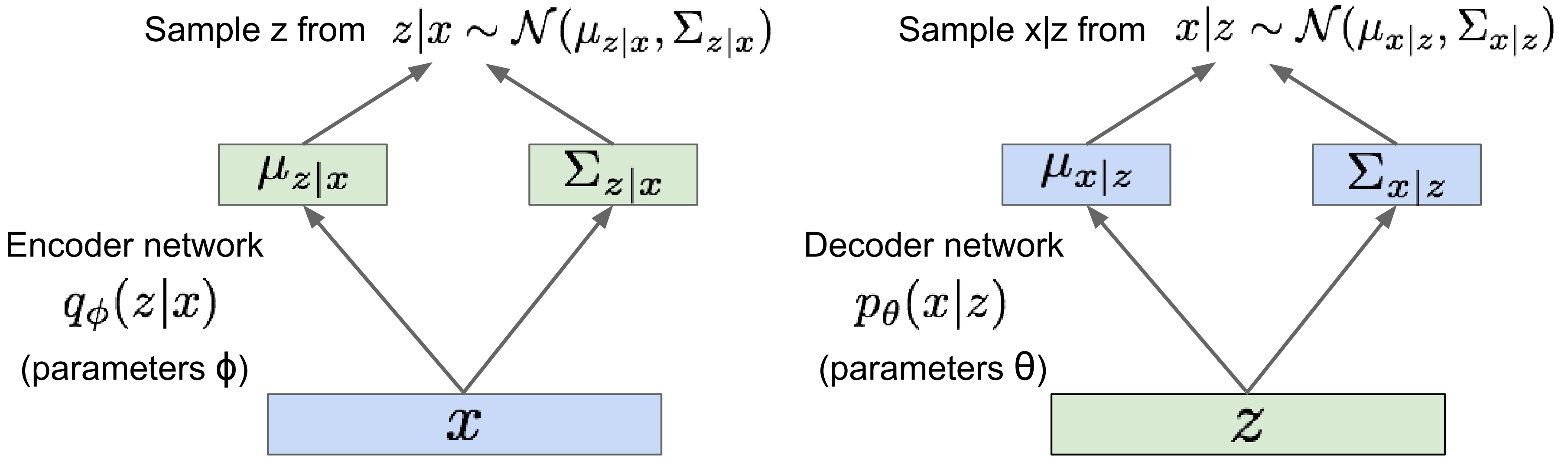
Solution: In addition to decoder network modeling $p_\theta(x|z)$, define additional encoder network $q_\phi(z|x)$ that approximates $p_\theta(z|x)$

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



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:

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

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

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

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

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

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

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

$\uparrow$ $p_\theta(z|x)$ 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:

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

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

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

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

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

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:

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

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

Make approximate posterior distribution close to prior

Reconstruct the input data

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

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

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

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

Variational lower bound ("ELBO")

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

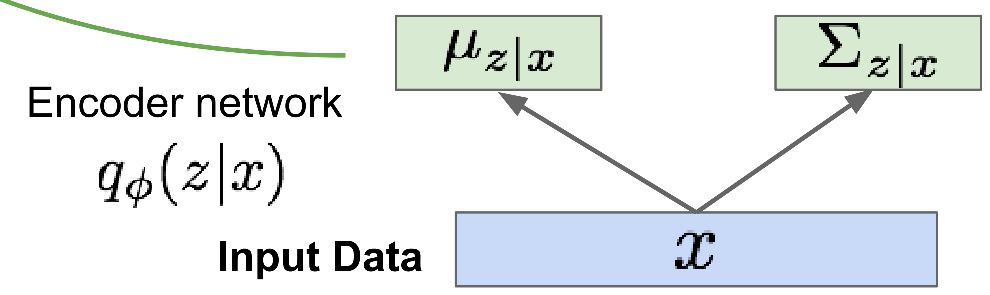
Training: Maximize lower bound

Stage I: Encoder

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



Stage II: Decoder.

Variational Autoencoders

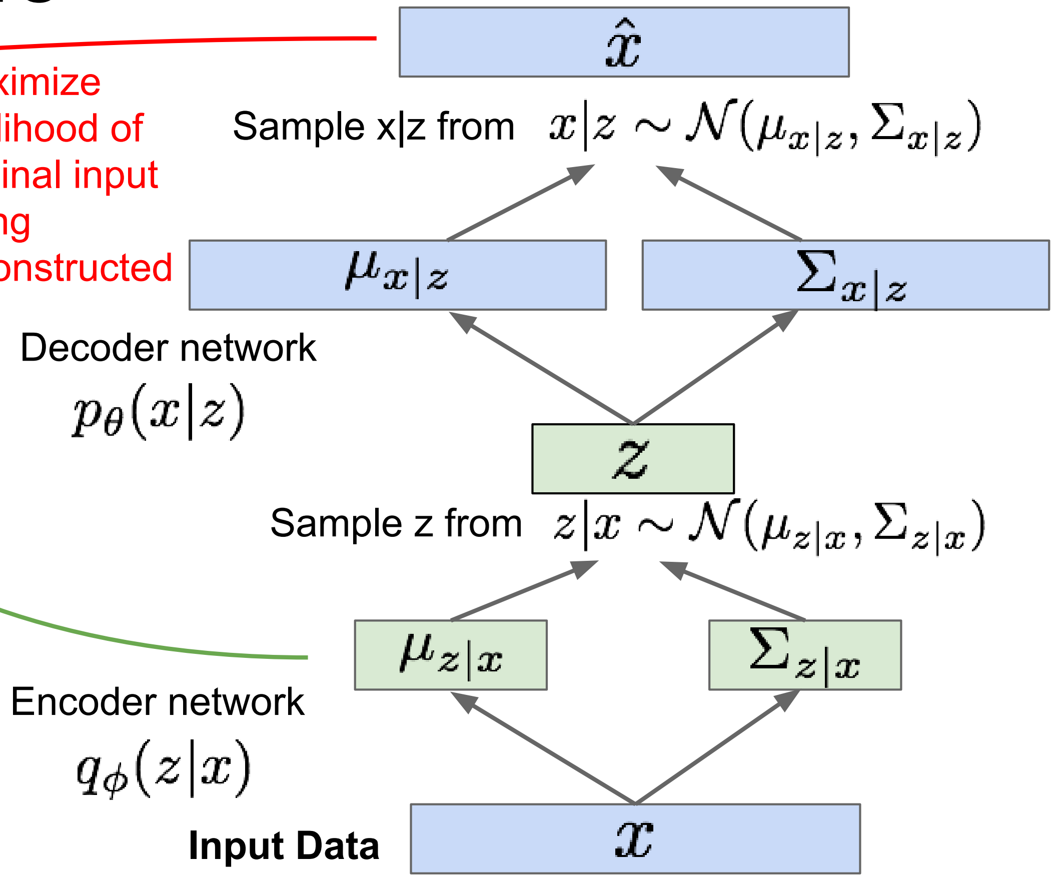
Putting it all together: maximizing the likelihood lower bound

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

Maximize likelihood of original input being reconstructed

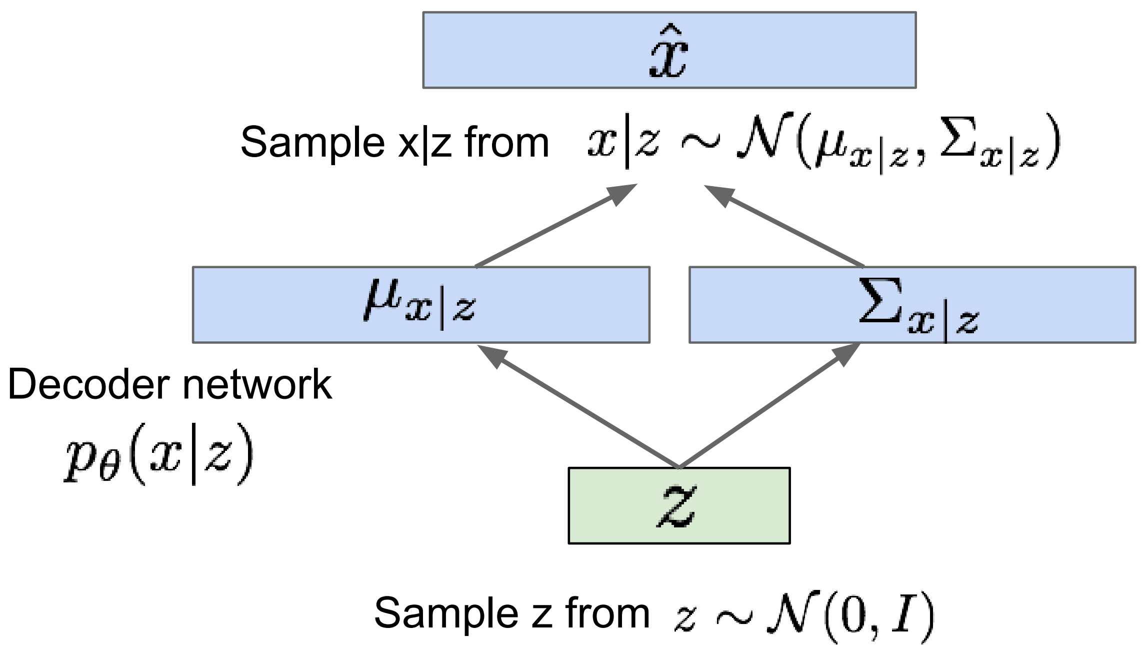
Make approximate posterior distribution close to prior

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


$\hat{x}$
Sample x|z from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$
$\mu_{x|z}$
$\Sigma_{x|z}$
Decoder network $p_\theta(x|z)$
z
Sample z from $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$
$\mu_{z|x}$
$\Sigma_{z|x}$
Encoder network $q_\phi(z|x)$
Input Data
x

VAE: generating data

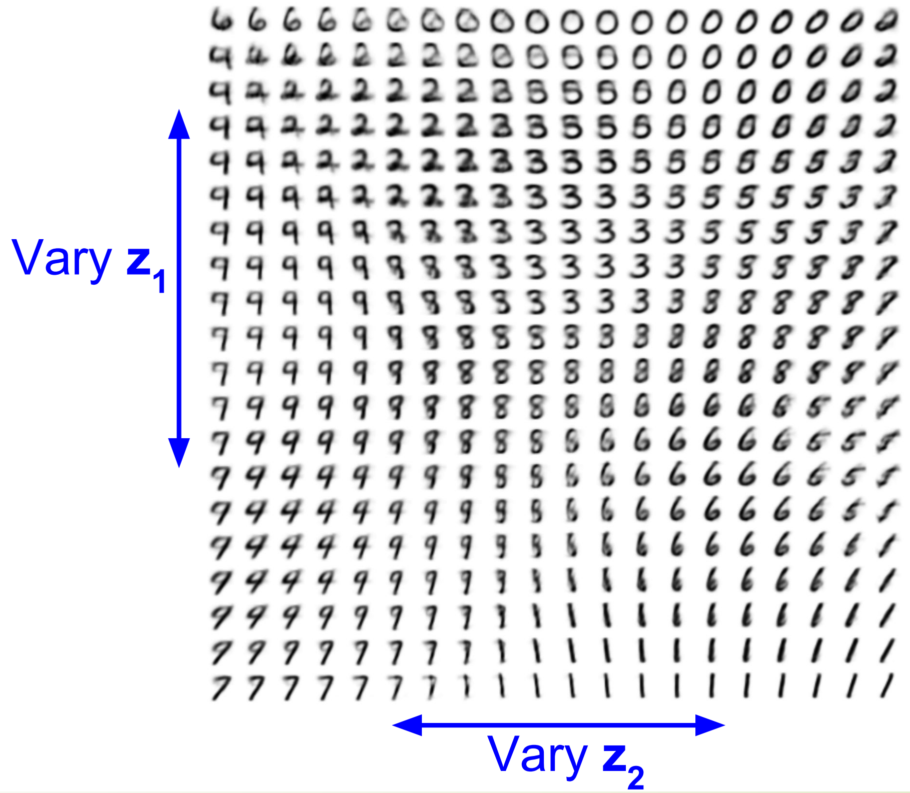
Use decoder network. Now sample z from prior!



$\hat{x}$
Sample x|z from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$
$\mu_{x|z}$
$\Sigma_{x|z}$
Decoder network
$p_\theta(x|z)$
z
Sample z from $z \sim \mathcal{N}(0, I)$

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

Data manifold for 2-d **z**

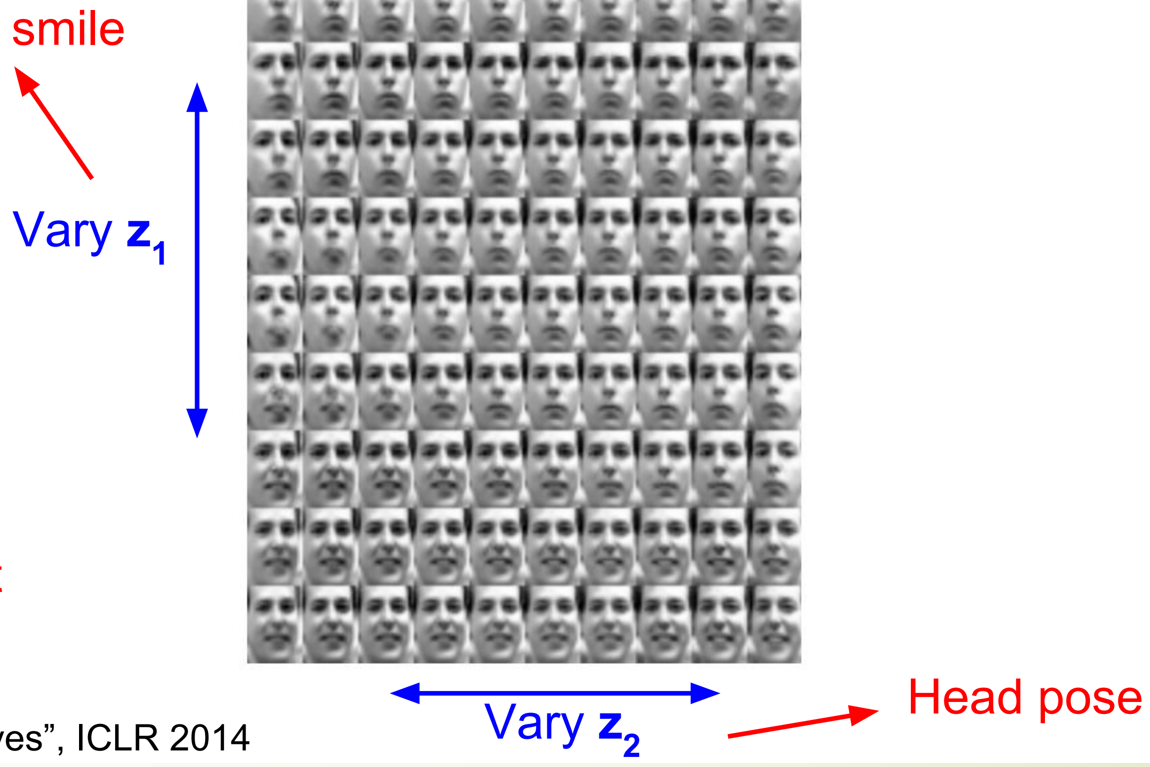


VAE: generating data

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

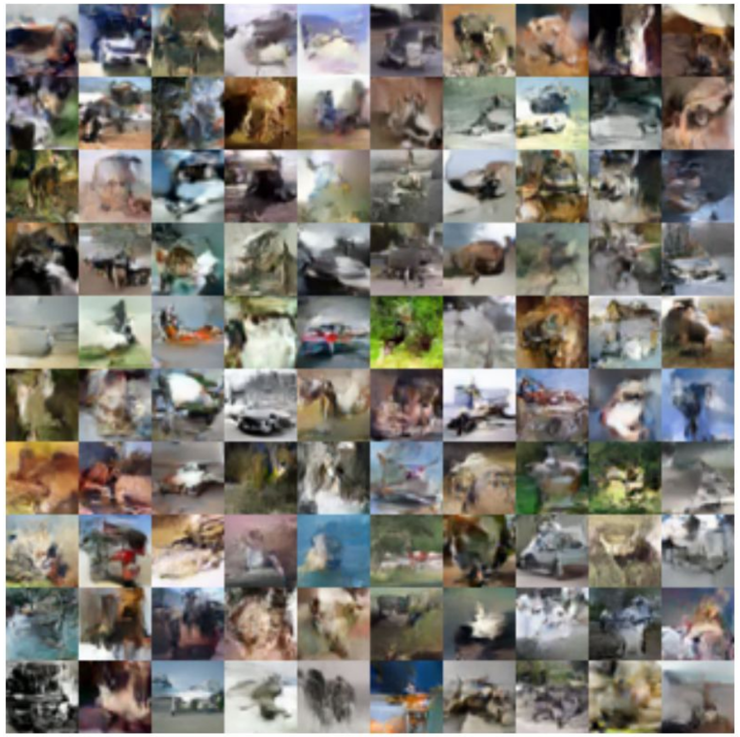
Different
dimensions of **z**
encode
interpretable factors
of variation

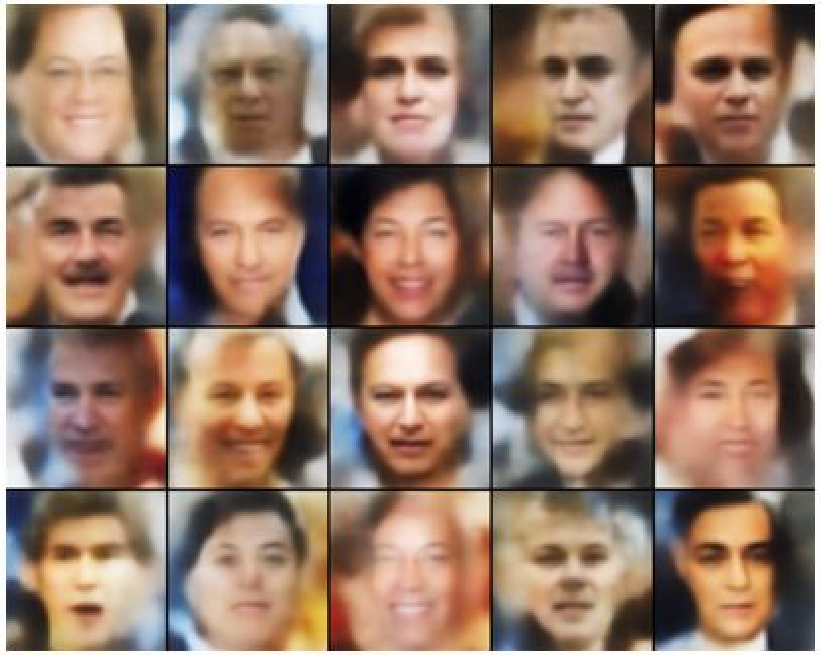
Also good feature representation that
can be computed using $q_\phi(z|x)$!



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
  - 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

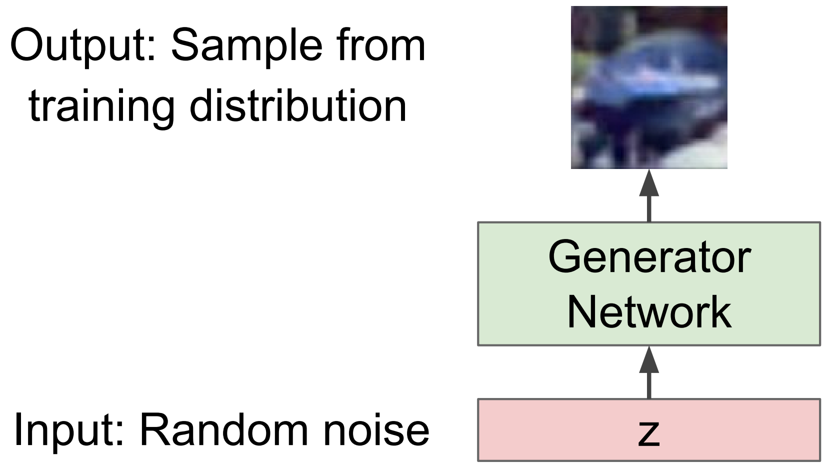
Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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?

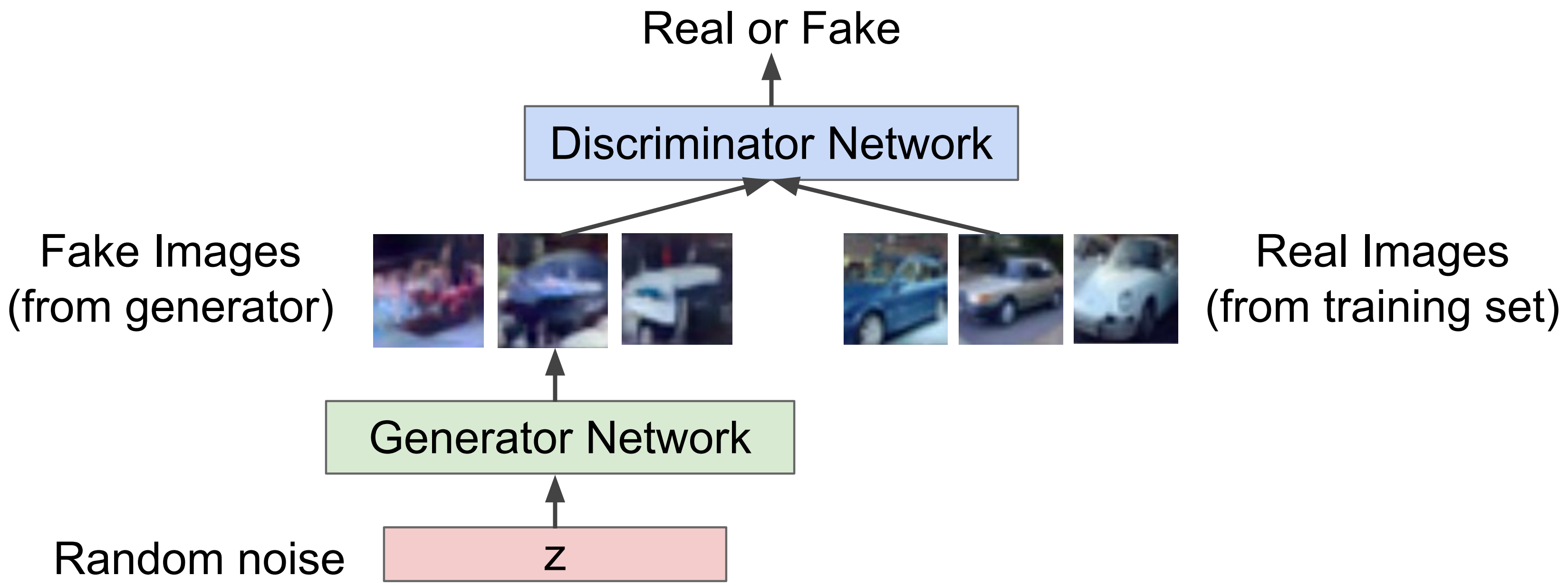
A: A neural network!



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**

Minimax objective function:

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

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

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

Training GANs

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

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

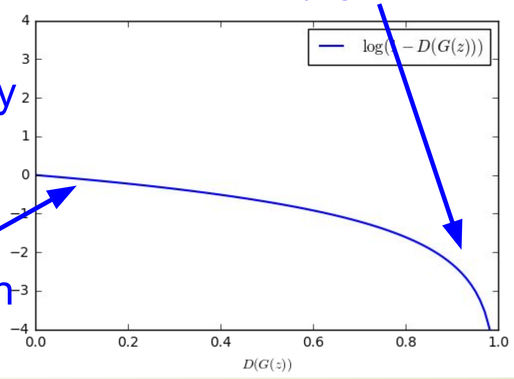
2. **Gradient descent** on generator

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

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

Gradient signal dominated by region where sample is already good

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



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:

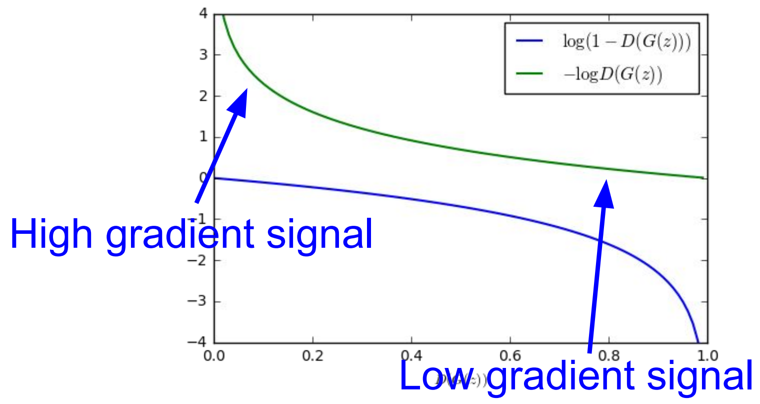
1. **Gradient ascent** on discriminator

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

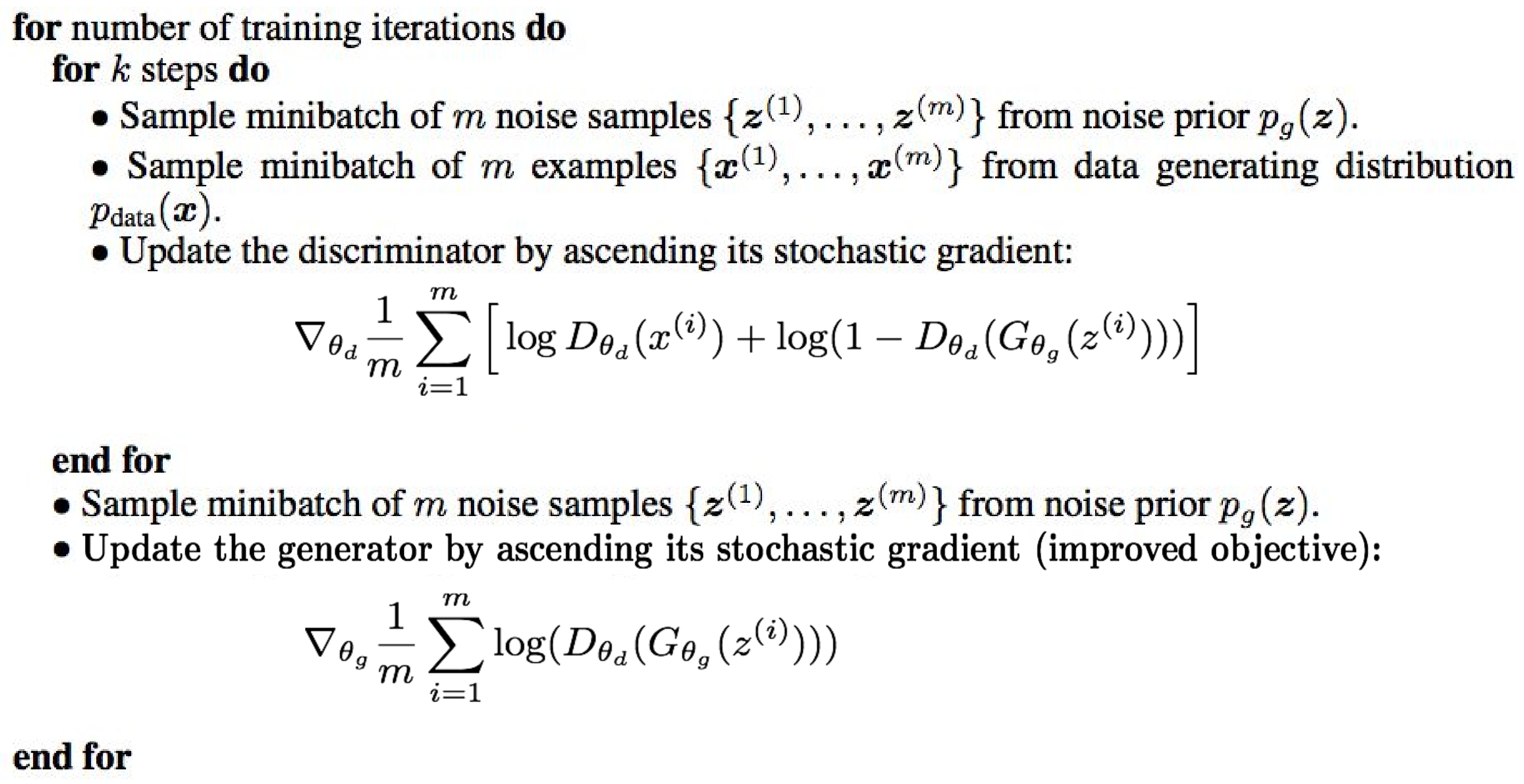
2. **Instead: Gradient ascent** on generator, different objective

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

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.



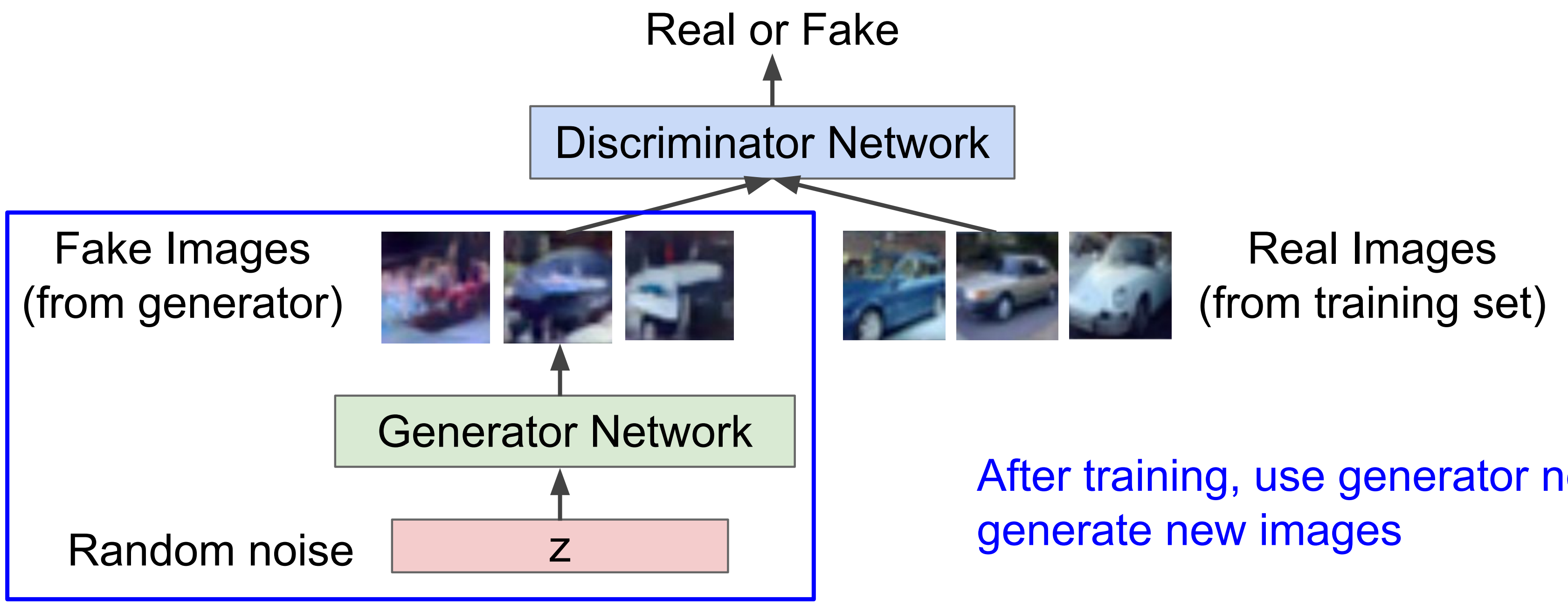
Putting it together: GAN training algorithm



```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
        • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ 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

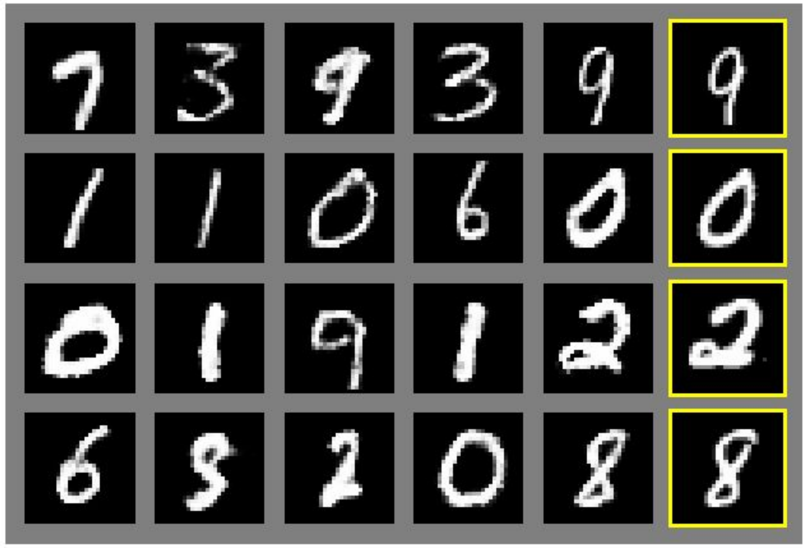


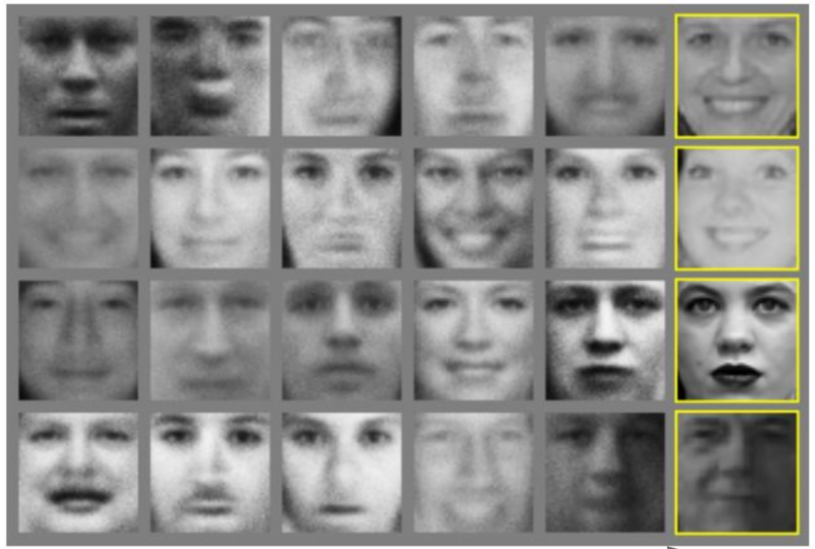
After training, use generator network to generate new images

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

Generative Adversarial Nets

Generated samples



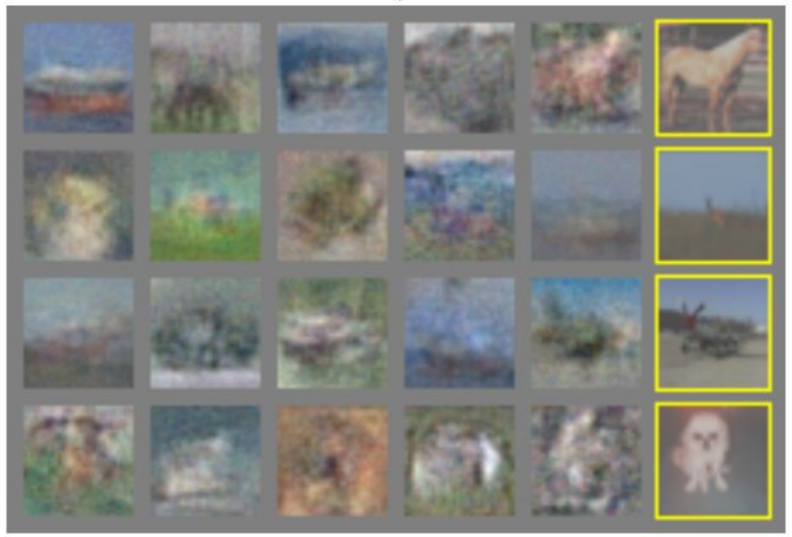


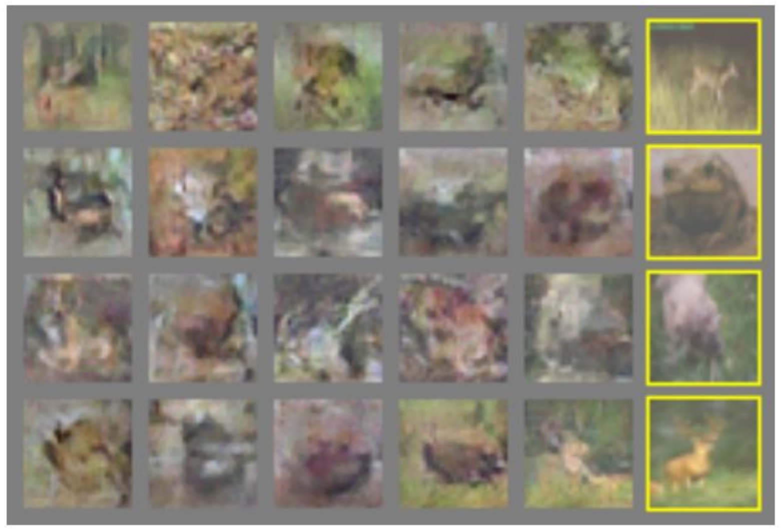
Nearest neighbor from training set

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

Generative Adversarial Nets

Generated samples (CIFAR-10)

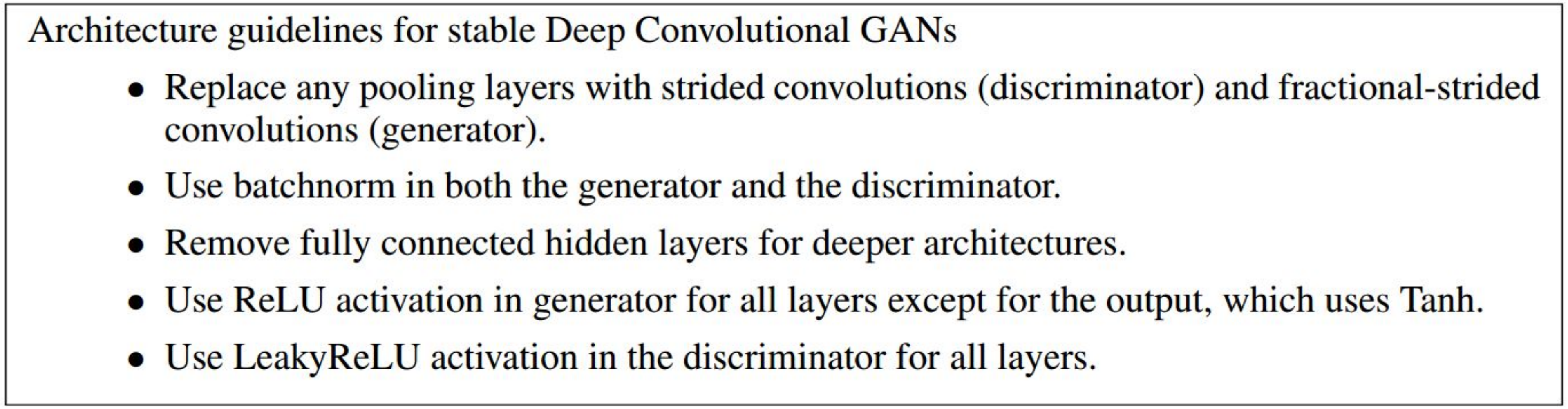




Nearest neighbor from training set

Generative Adversarial Nets: Convolutional Architectures

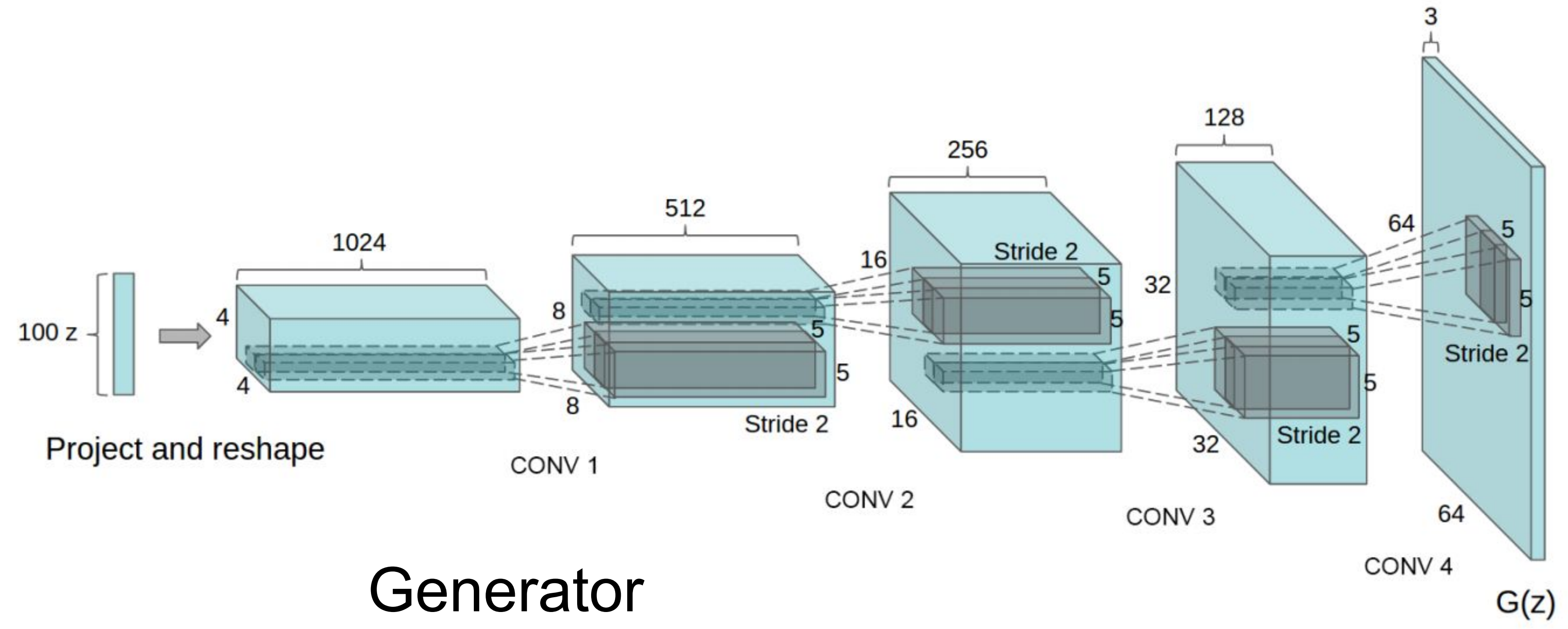
Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network



Architecture guidelines for stable Deep Convolutional GANs

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

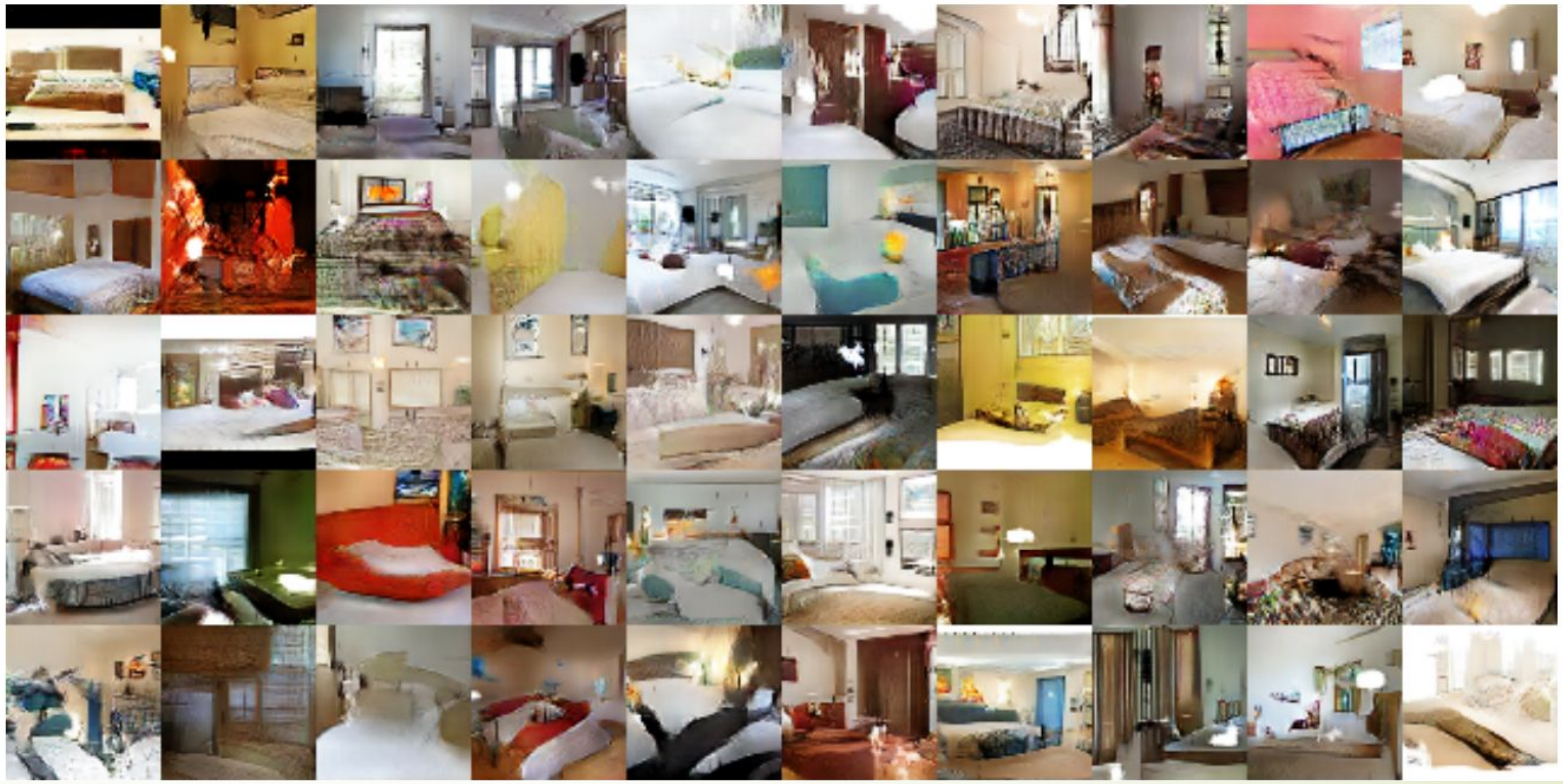
Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016



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

Generative Adversarial Nets: Convolutional Architectures

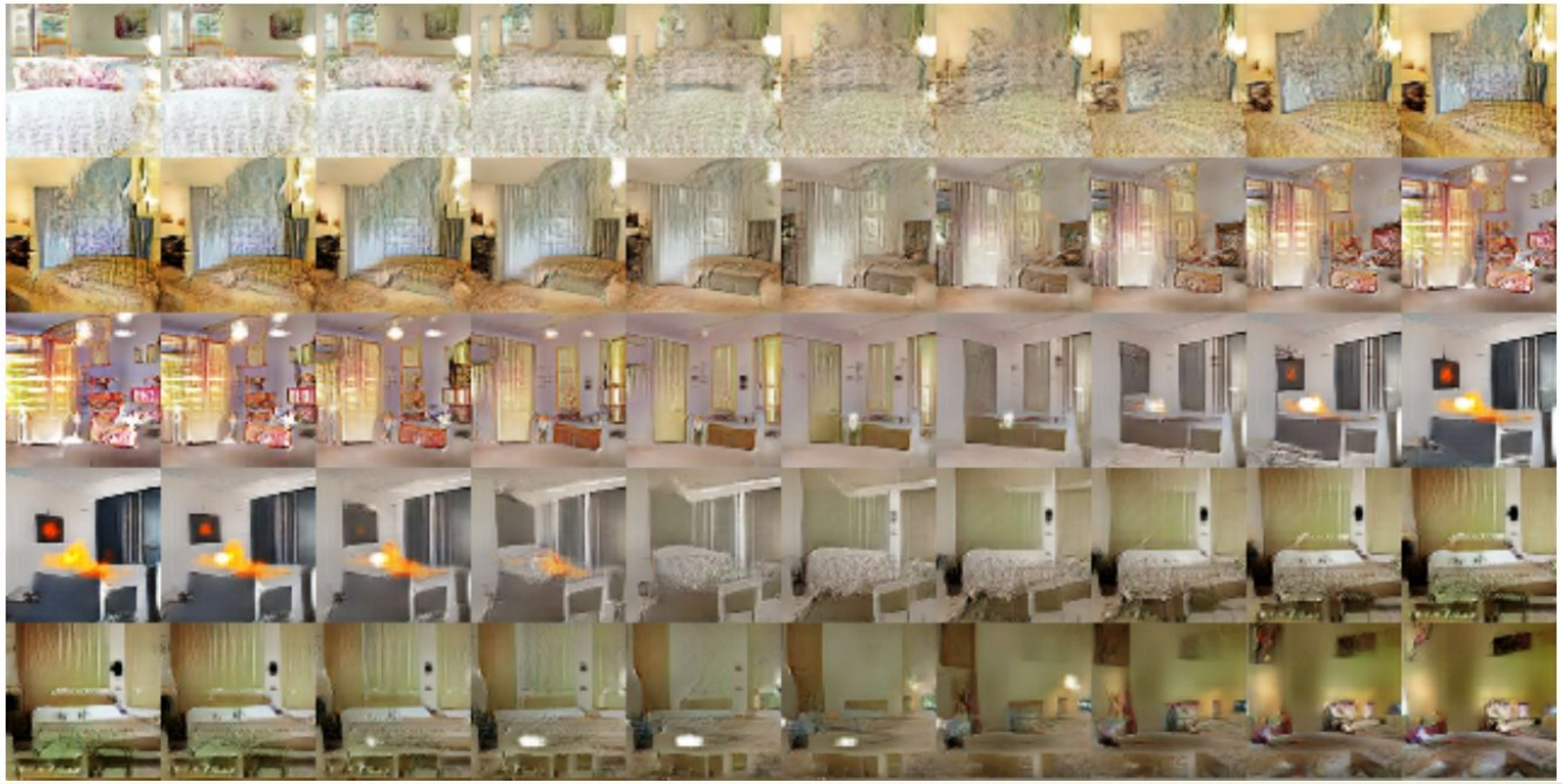
Samples from the model look amazing!



Radford et al, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

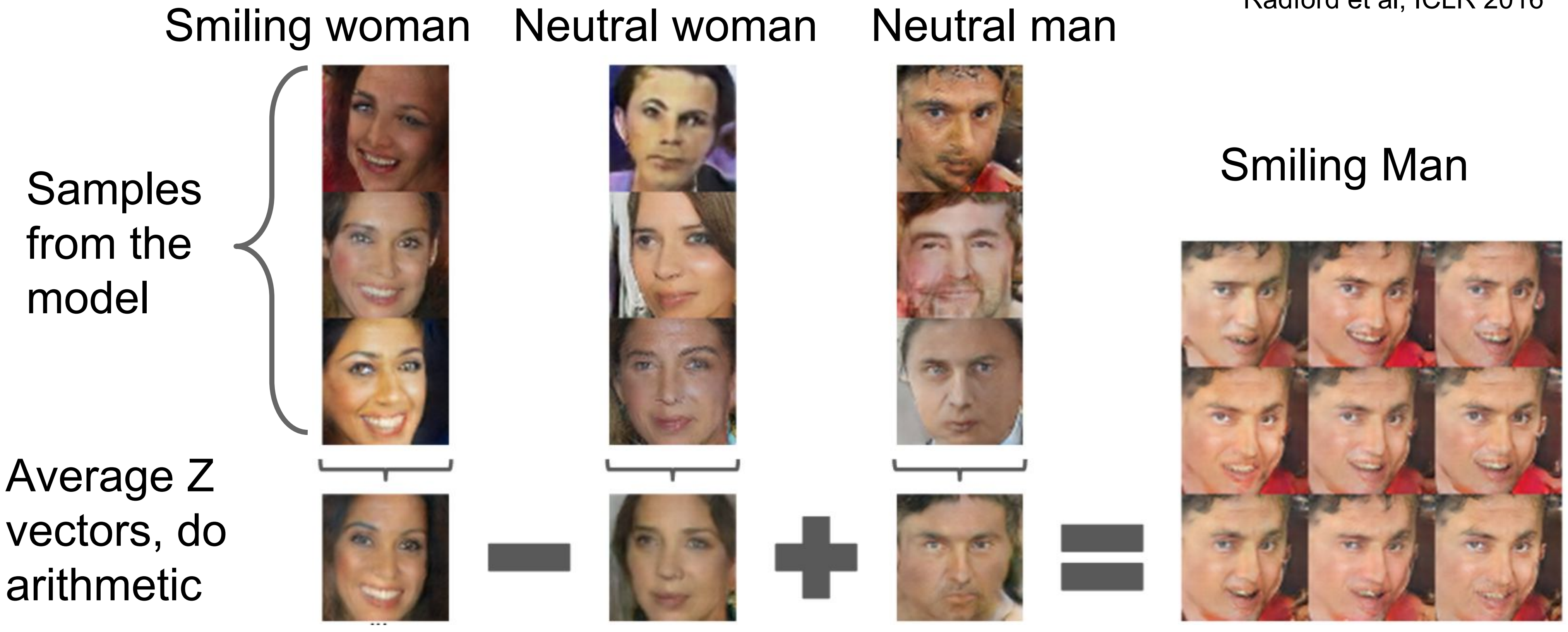
Interpolating between random points in laten space



Radford et al, ICLR 2016

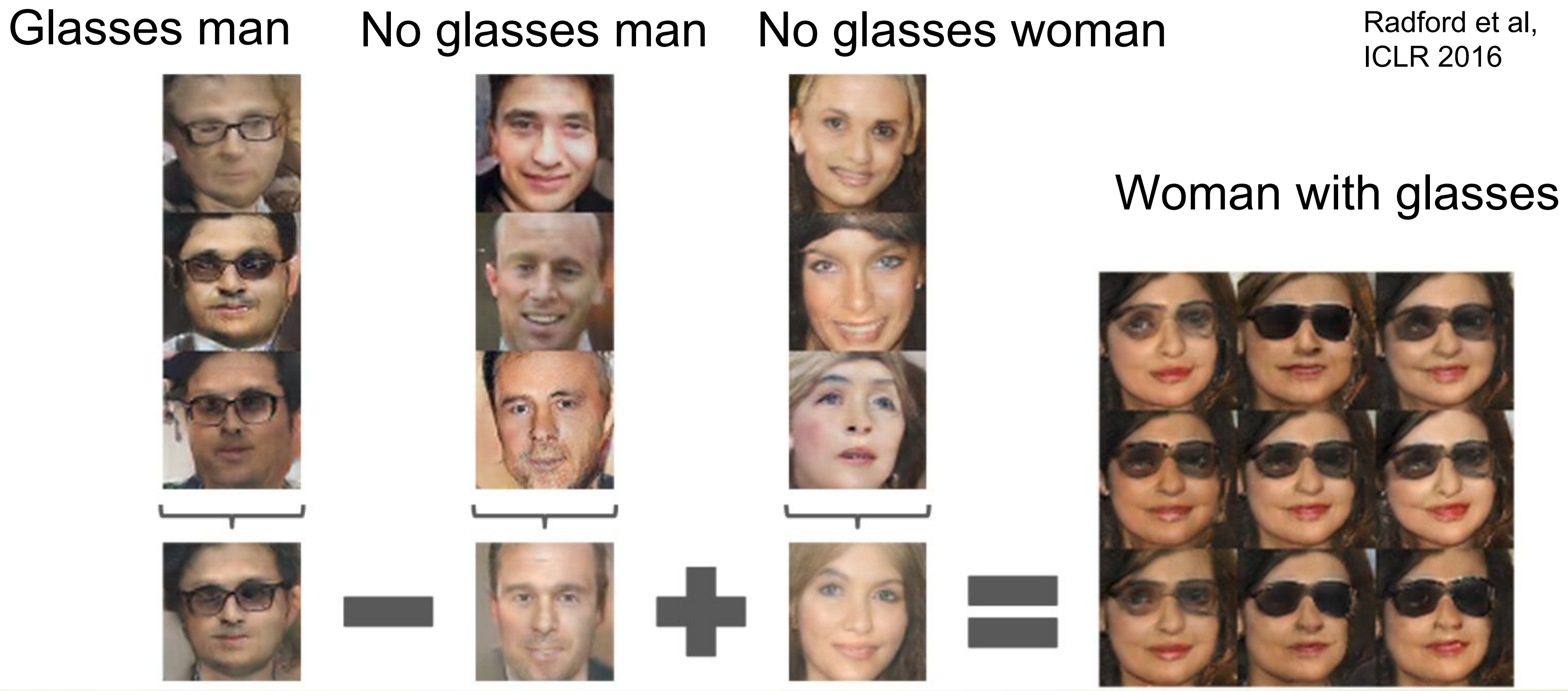
Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016



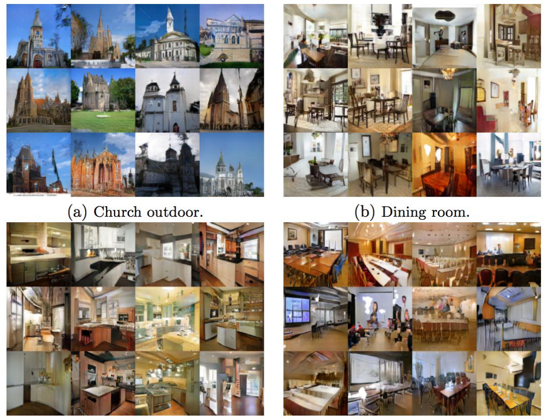
Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016



2017: Year of the GAN

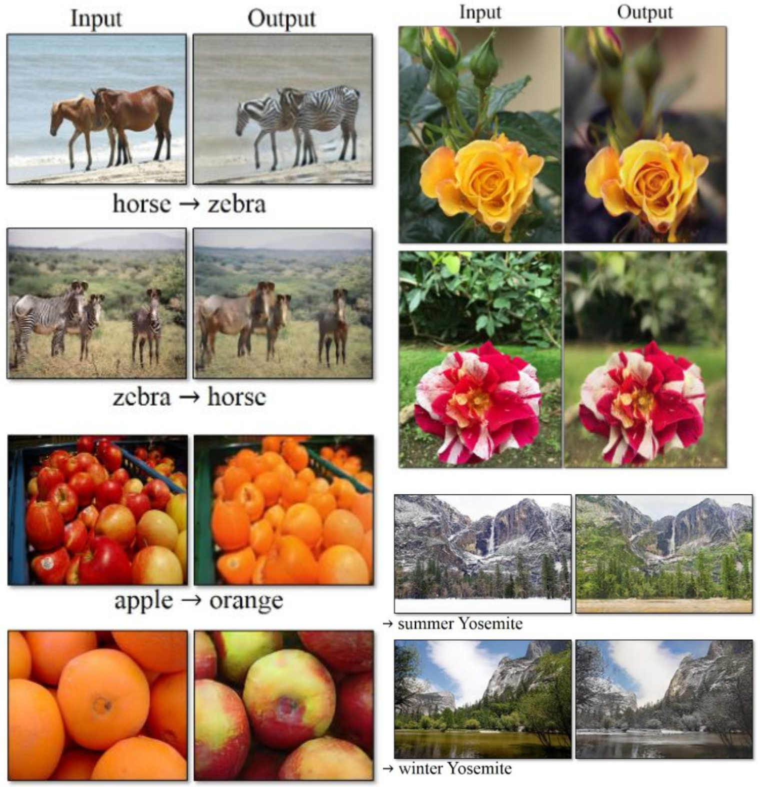
Better training and generation



LSGAN. Mao et al. 2017.

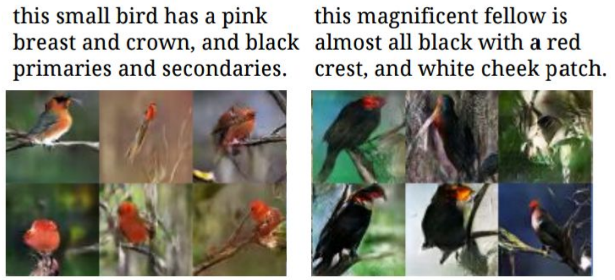
BEGAN. Bertholet et al. 2017.

Source->Target domain transfer



CycleGAN. Zhu et al. 2017.

Text -> Image Synthesis



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

Application: Credit card fraud detection via GAN

Hung Ba, Improving Detection of Credit Card Fraudulent Transactions using Generative Adversarial Networks, arXiv: 1907.03355.

Replicated by Ruoxue Liu
(HKUST)

Brief introduction

- Problem: Imbalanced dataset

  Non-fraud(0): 284315; fraud(1): 492; 0.17% imbalanced

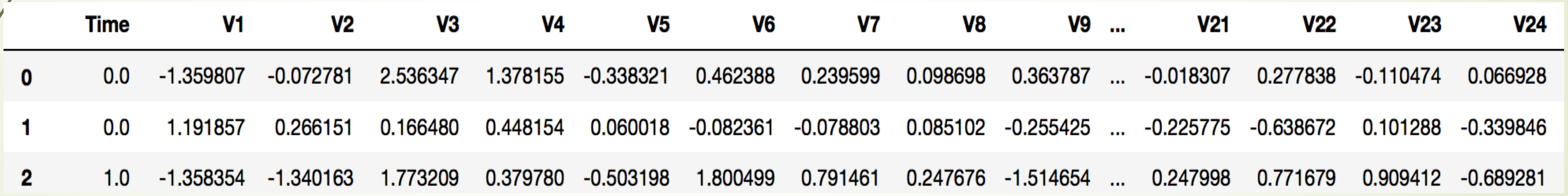
  classification algorithms have difficulties identifying the minority classes

- Method: Augment the minority class (fraud) using synthetic data produced via GAN, then conduct classification, e.g. Decision Tree.

- Results: Improved the test classification AUC from 0.93 to 0.97, improved accuracy rate from 0.97 to 0.99.

Dataset

- Credit Card Fraud Detection (Dal Pozzolo et al. (2017)): This data set contains transactions made by credit cards in September 2013 by European cardholders. It presents transactions that occurred in two days, where there are 492 frauds out of 284,315 transactions. The data set is highly imbalanced, the positive class (frauds) account for only 0.172% of all transactions.

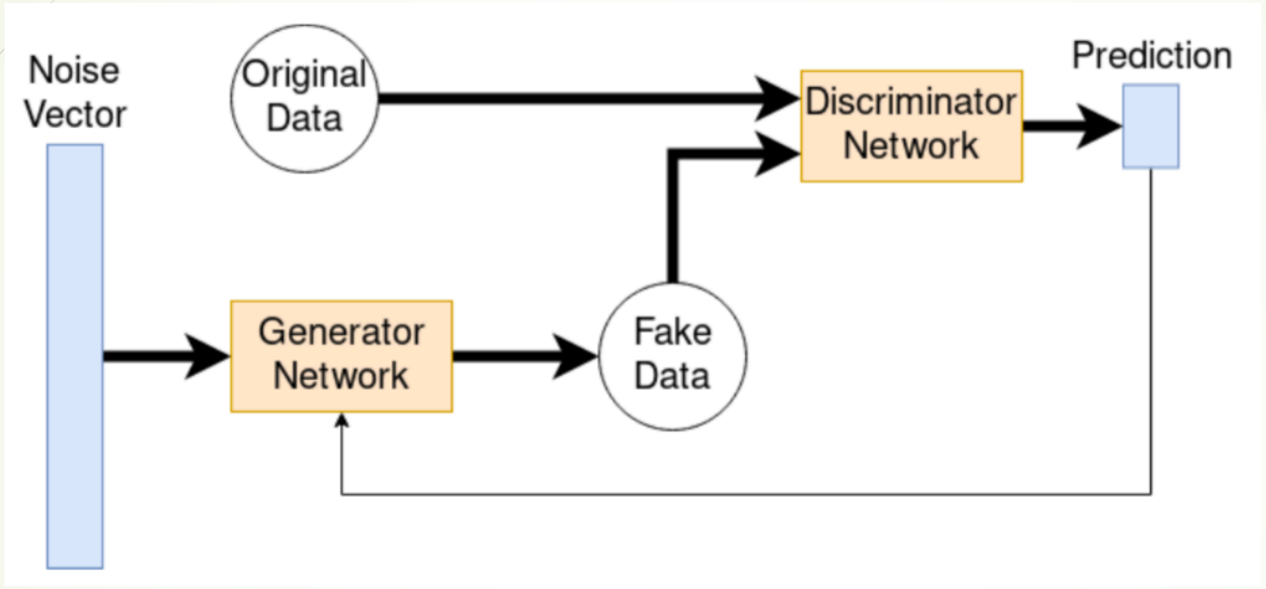


	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281

Experimental method:

- Normalization the features between 0 and 1.
- Separated the training set based on the target class (non-frauds and frauds).
- Trained the GAN using only the minority class data.
- Used the GAN to add new entries to the training data set until it becomes balanced or just mitigate imbalanced.
- Used the newly more balanced training data set to train the classifier.
- The classifier was tested on the original test set.

GAN



Value function: $\min_G \max_D V(D,G) = E_{x\sim Pdata}[logD(x)] + E_{z\sim Pz(Z)}[\log(1-D(G(z)))]$

Cost function: J_D= - $\frac{1}{2m}(\sum_{i=1}^{m} logD(xi) + \sum_{i=1}^{m} \log(1-D(G(zi))))$

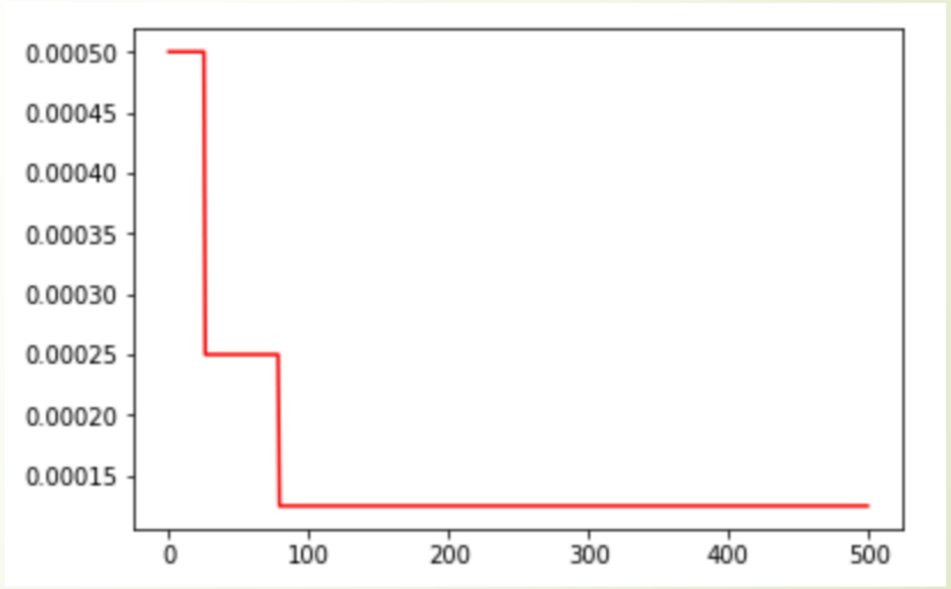
J_G= - $\frac{1}{m}(\sum_{i=1}^{m} logD(G(zi)))$

Tips and parameters to make GANs work

- Normalize data between 0 and 1
- Tanh as the last layer of the generator output; Use dropout; Avoid ReLU, use LeakyReLU
- Optimize G with min (log 1-D) or max log D (log D trick)
- Label Smoothing. E.g. Real=1 and Fake=0, then replace the label with a random number between 0.7 and 1.2 for real, if it is a fake sample, replace it with 0.0 and 0.3
- Train discriminator more/ generator more according to loss
- Gradient clipping
- Learning rate decay

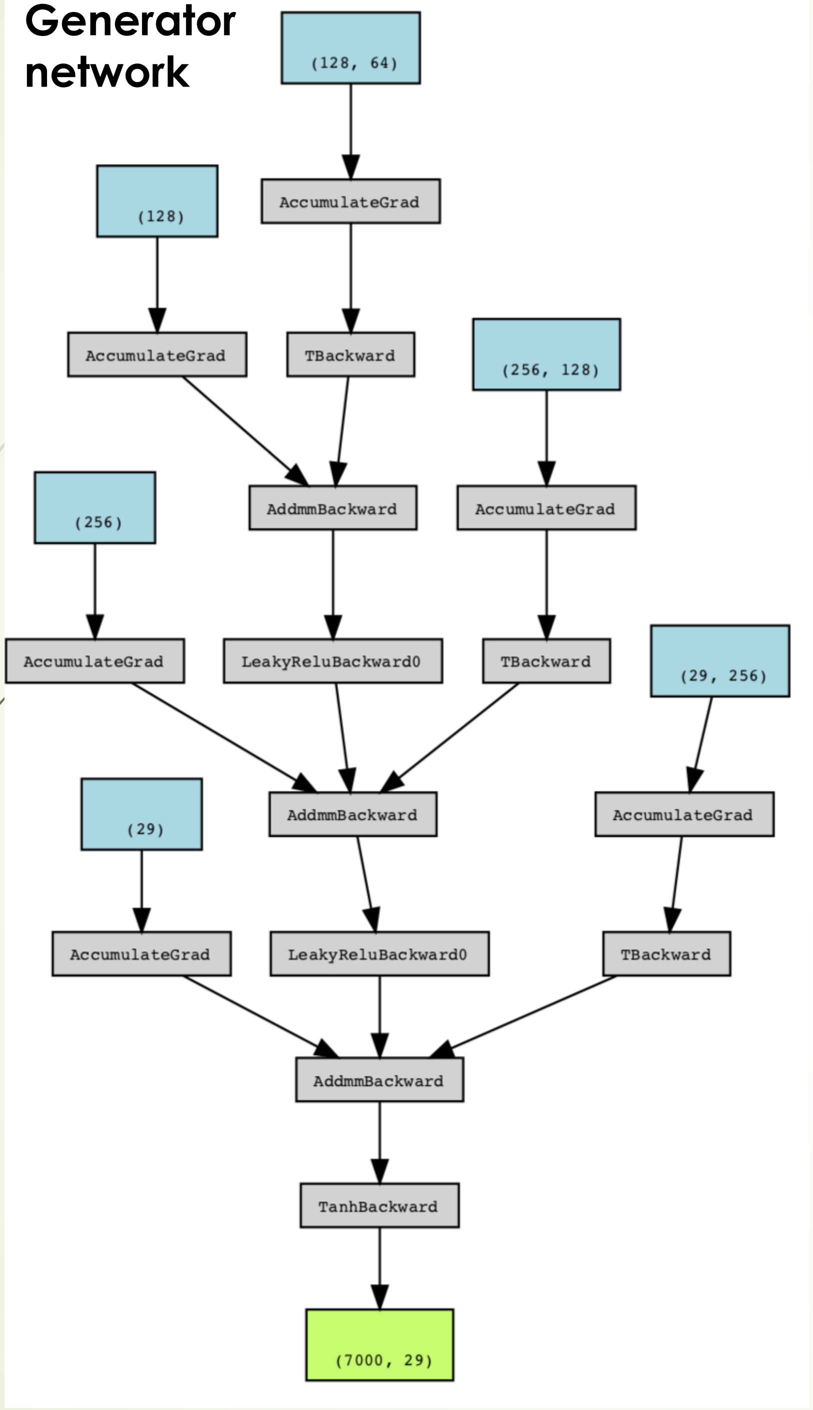
Parameters:

- G hidden layers [128, 256]; D hidden layers [256,128]
- Input noise dimension 64
- batch size = 5
- Learning rate decay
- Adam as the optimizer

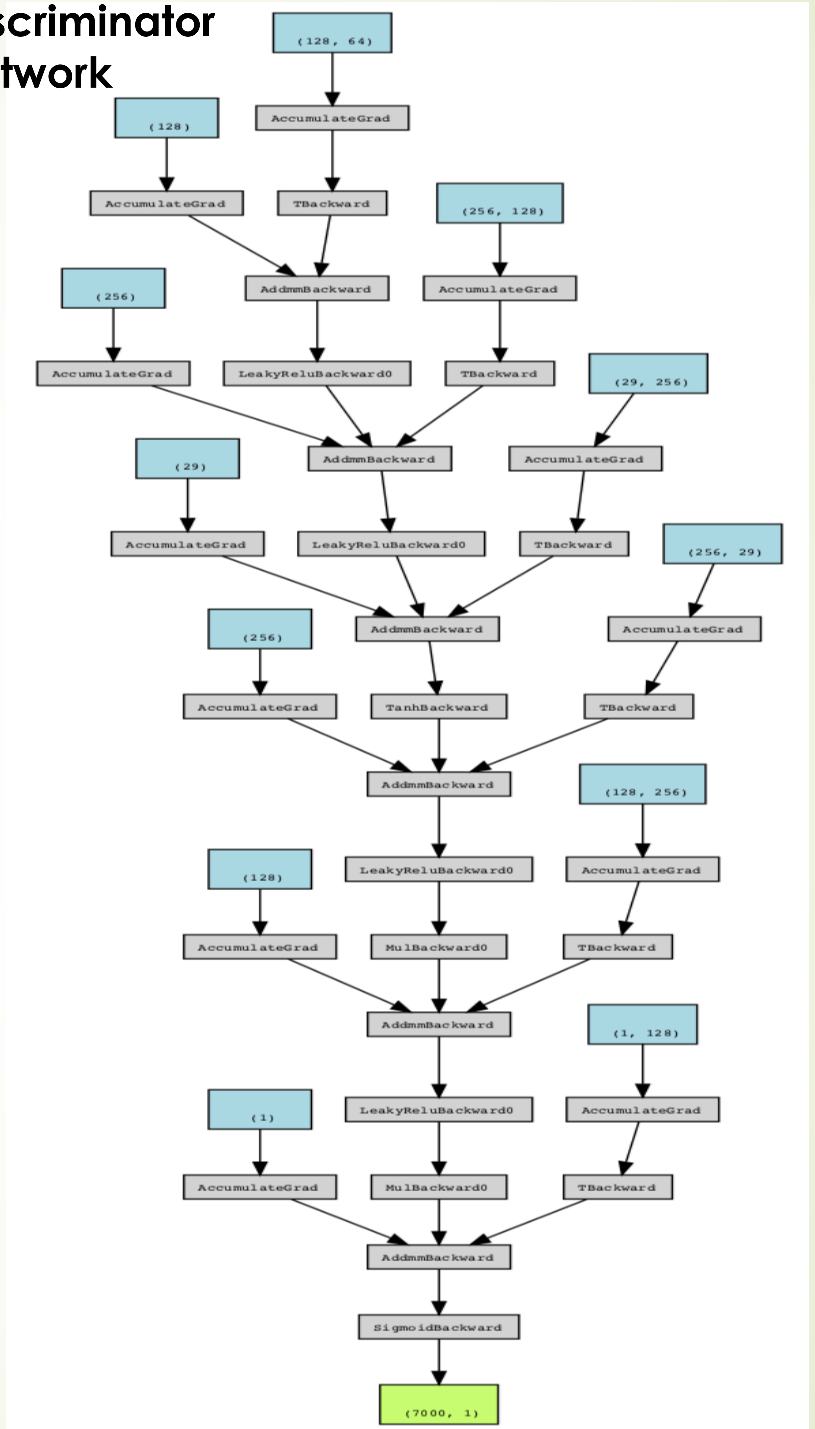


Lr decay

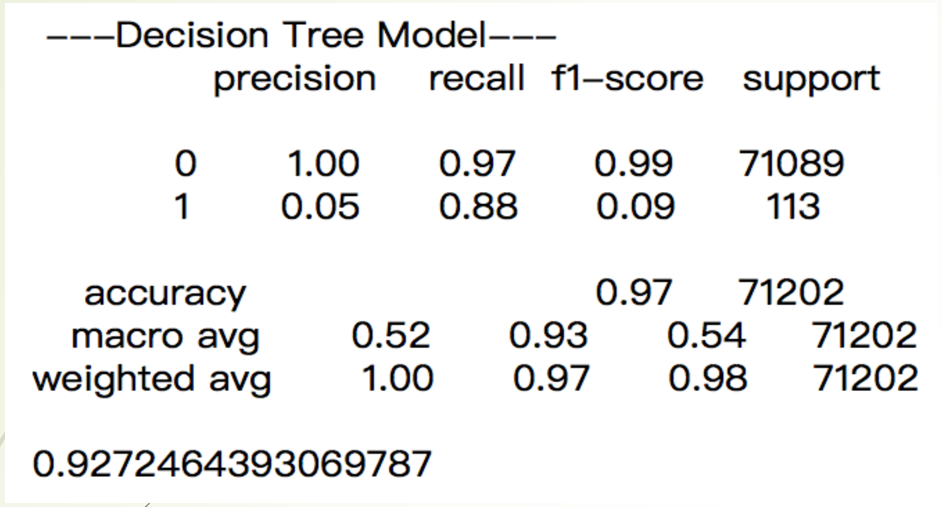
Generator network



Discriminator network

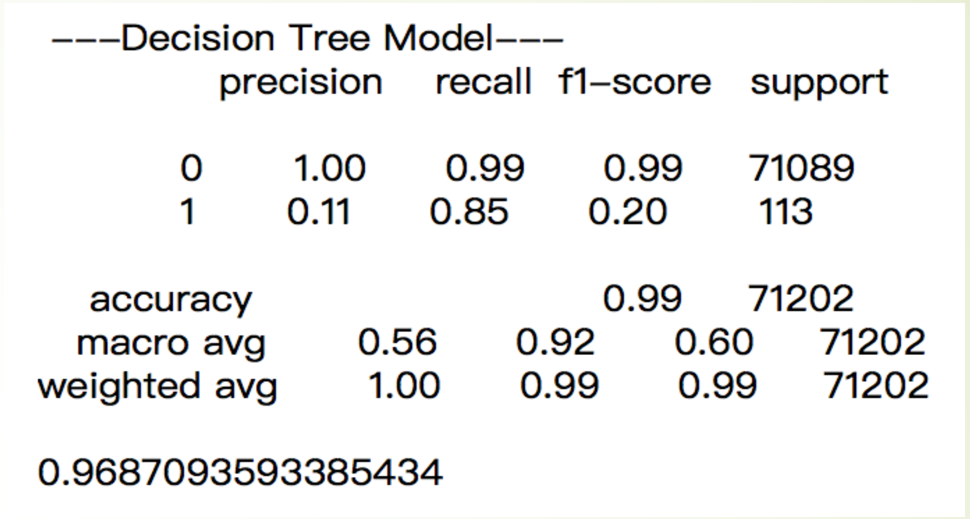


Results



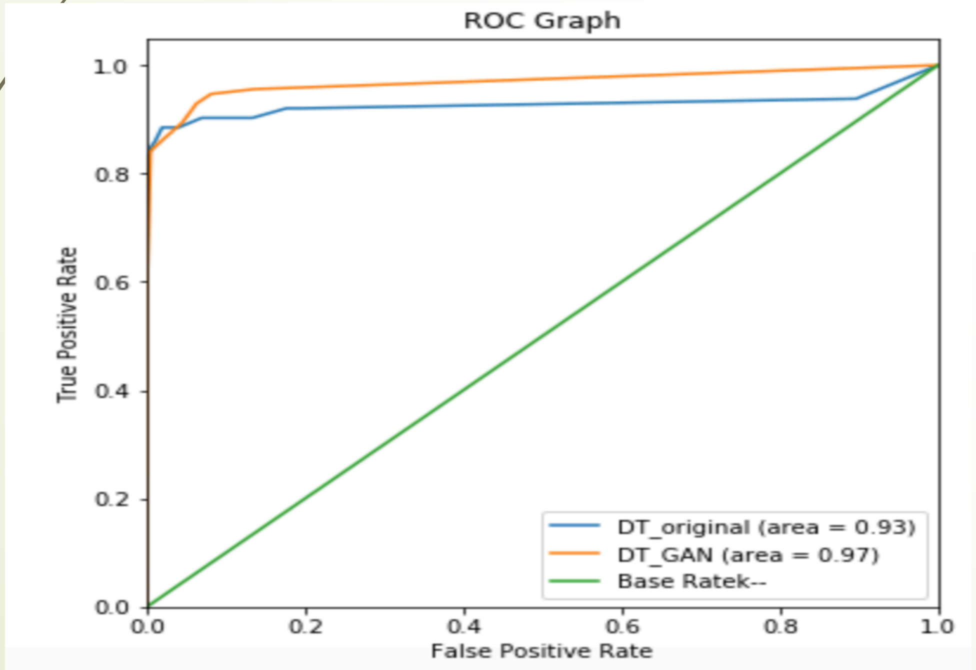
```
---Decision Tree Model---
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     71089
           1       0.05      0.88      0.09       113

    accuracy                           0.97     71202
   macro avg       0.52      0.93      0.54     71202
weighted avg       1.00      0.97      0.98     71202

0.9272464393069787
```

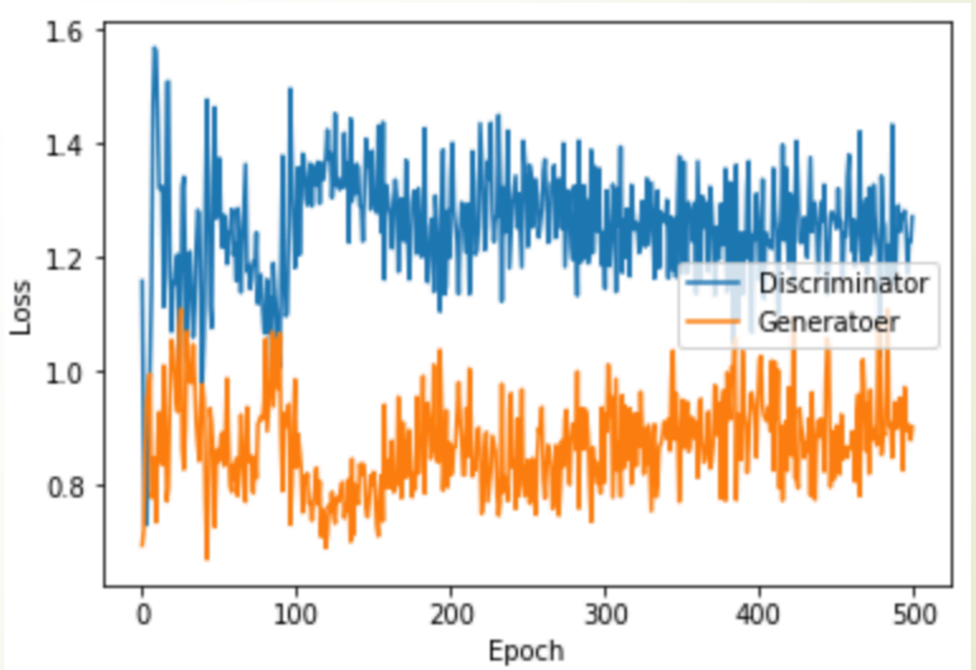
DT performance on original data



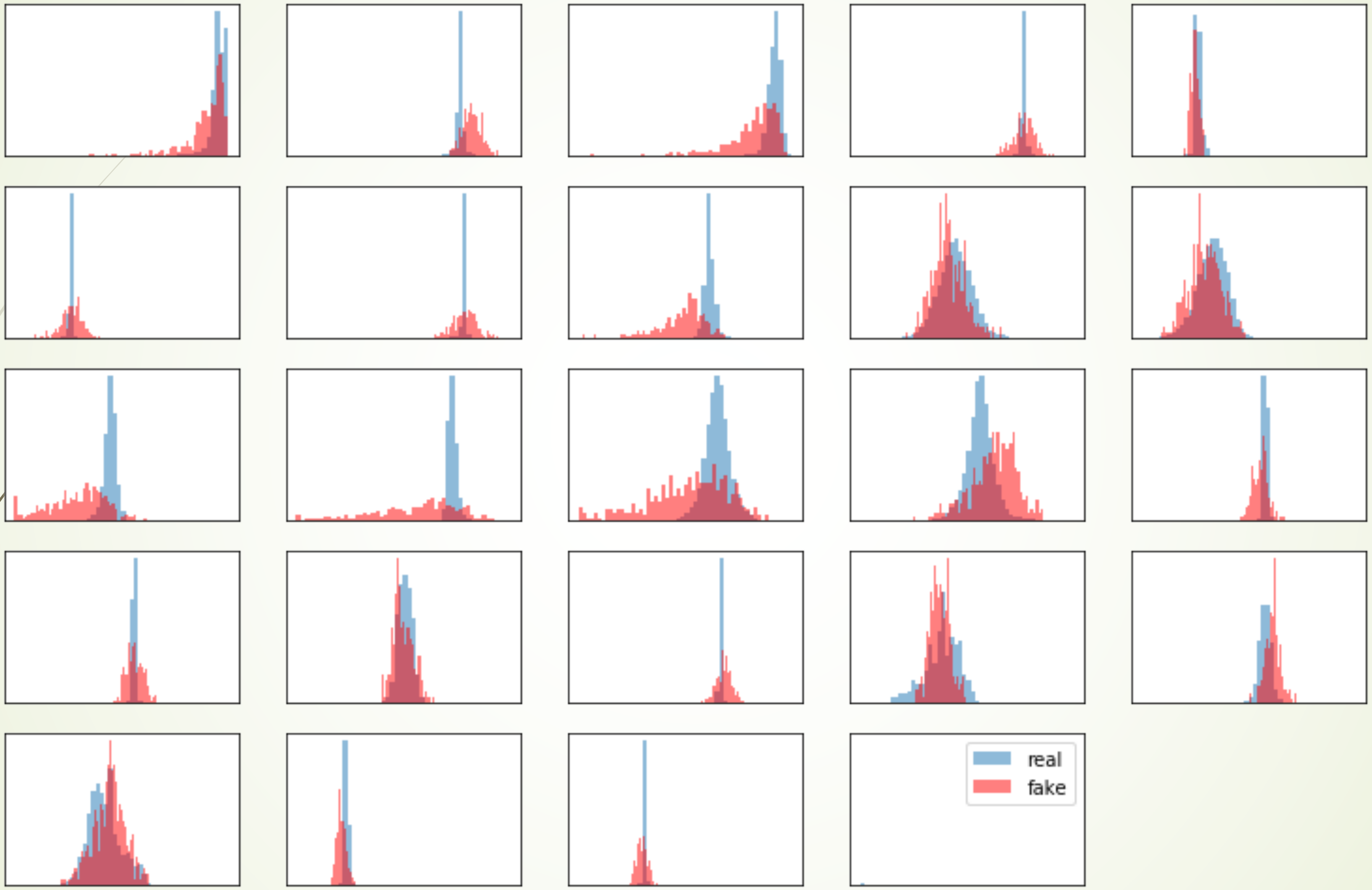
```
---Decision Tree Model---
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     71089
           1       0.11      0.85      0.20       113

    accuracy                           0.99     71202
   macro avg       0.56      0.92      0.60     71202
weighted avg       1.00      0.99      0.99     71202

0.9687093593385434
```

DT performance on synthetic data





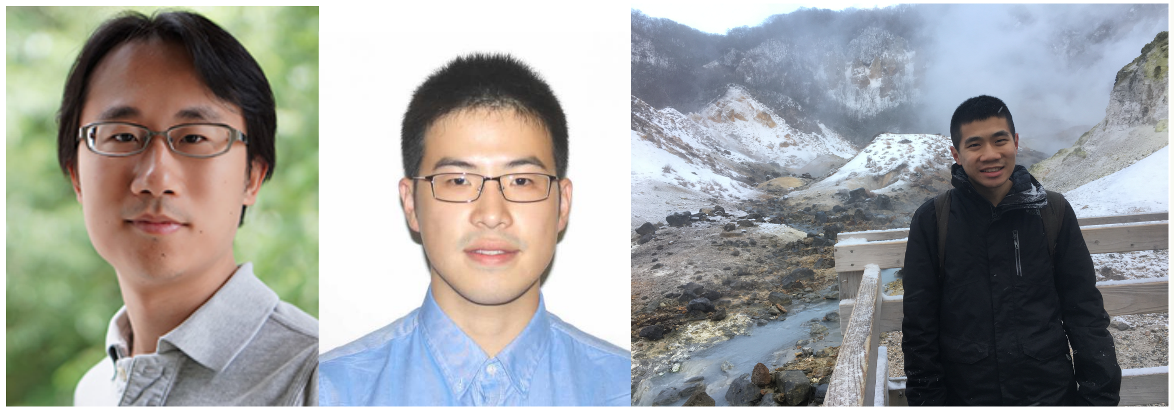
Result: generated feature distributions



Compare real and fake fraud feature distribution

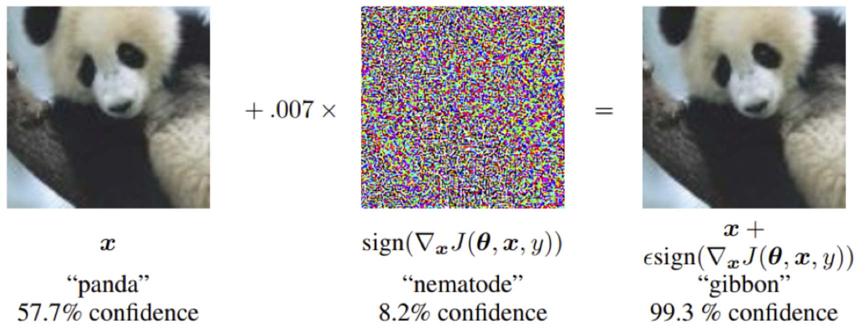
Robust Estimation and GANs

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



Chao Gao (Chicago) *Jiyu Liu* (Yale) *Weizhi Zhu* (HKUST)

Deep Learning is Notoriously Not Robust!

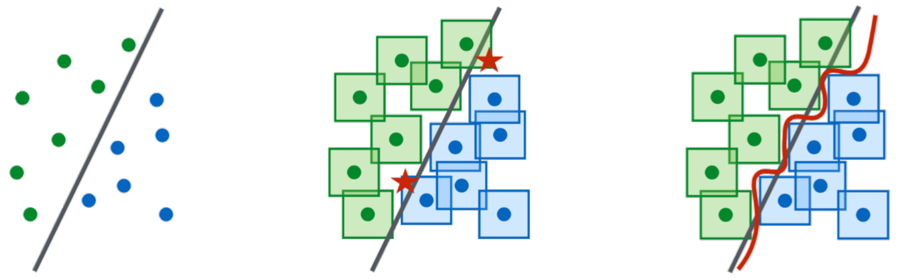


$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

[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 (DRO)

- 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

$$\mathcal{D} = \{P_\epsilon : W_2(P_\epsilon, \text{uniform distribution}) \leq \epsilon\}$$

where DRO may be reduced to regularized maximum likelihood estimates (Shafieezadeh-Abadeh, Esfahani, Kuhn, NIPS'2015) that are convex optimizations and tractable

TV-ambiguity set

- Now how about the TV-uncertainty set?

$$\mathcal{D} = \{P_\epsilon : TV(P_\epsilon, \text{uniform distribution}) \leq \epsilon\}?$$

- an example from *robust statistics* …

Huber's Model

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

contamination proportion

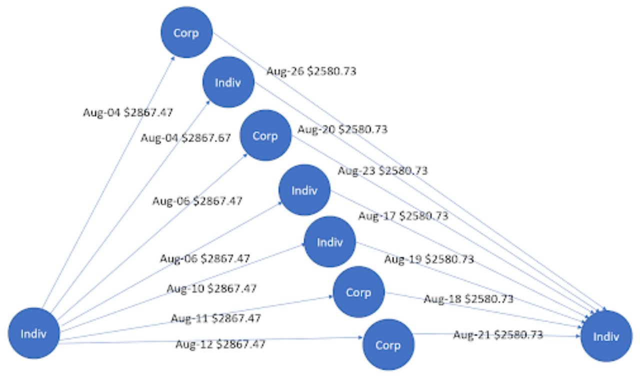
arbitrary contamination

parameter of interest

[Huber 1964]

Example: Financial Fraud

- *P* represent normal transactions
- *Q* represent fraudulent transactions, e.g. money laundering, which is sparse and arbitrarily close to *P*
- Finding *P* and its dual problem in finding *Q*?



X

An Example

$$X_1, \ldots, X_n \sim (1-\epsilon)N(\underset{\text{how to estimate ?}}{\theta}, I_p) + \epsilon Q.$$

Robust Maxmum-Likelihood Does not work!

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

$$\ell(\theta, Q) = \text{negative log-likelihood} = \sum_{i=1}^{n} (\theta - X_i)^2$$

$$\sim (1-\epsilon)\mathbb{E}_{\mathcal{N}(\theta)}(\theta - X)^2 + \epsilon \mathbb{E}_Q(\theta - X)^2$$

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**

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

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]

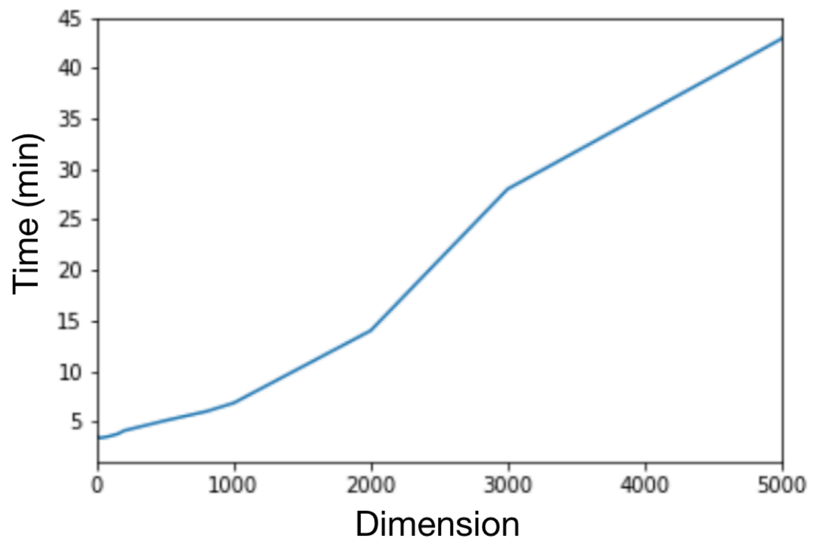
Computational Challenges

- Lai, Rao, Vempala
- Diakonikolas, Kamath, Kane, Li, Moitra, Stewart,
- Balakrishnan, Du, Singh

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

- Polynomial algorithms are proposed [Diakonikolas et al.'16, Lai et al. 16] of minimax optimal statistical precision
  - needs information on second or higher order of moments
  - some priori knowledge about ϵ

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]

Robust Learning of Gaussian Distributions

Q	n	p	ϵ	TV-GAN	JS-GAN	Dimension Halving	Iterative Filtering
$N(0.5 * 1_p, I_p)$	50,000	100	.2	**0.0953 (0.0064)**	0.1144 (0.0154)	0.3247 (0.0058)	0.1472 (0.0071)
$N(0.5 * 1_p, I_p)$	5,000	100	.2	**0.1941 (0.0173)**	0.2182 (0.0527)	0.3568 (0.0197)	0.2285 (0.0103)
$N(0.5 * 1_p, I_p)$	50,000	200	.2	**0.1108 (0.0093)**	0.1573 (0.0815)	0.3251 (0.0078)	0.1525 (0.0045)
$N(0.5 * 1_p, I_p)$	50,000	100	.05	0.0913 (0.0527)	0.1390 (0.0050)	0.0814 (0.0056)	**0.0530 (0.0052)**
$N(5 * 1_p, I_p)$	50,000	100	.2	2.7721 (0.1285)	**0.0534 (0.0041)**	0.3229 (0.0087)	0.1471 (0.0059)
$N(0.5 * 1_p, \Sigma)$	50,000	100	.2	0.1189 (0.0195)	**0.1148 (0.0234)**	0.3241 (0.0088)	0.1426 (0.0113)
Cauchy$(0.5 * 1_p)$	50,000	100	.2	0.0738 (0.0053)	**0.0525 (0.0029)**	0.1045 (0.0071)	0.0633 (0.0042)

Table: Comparison of various robust mean estimation methods. The smallest error of each case is highlighted in bold.

- *Dimension Halving:* [Lai et al.'16]
  `https://github.com/kal2000/AgnosticMeanAndCovarianceCode`.
- *Iterative Filtering:* [Diakonikolas et al.'17]
  `https://github.com/hoonose/robust-filter`.

Robust Learning of Cauchy Distributions whose moments do not exist!

Table 4: Comparison of various methods of robust location estimation under Cauchy distributions. Samples are drawn from $(1-\epsilon)\text{Cauchy}(0_p, I_p) + \epsilon Q$ with $\epsilon = 0.2, p = 50$ and various choices of Q. Sample size: 50,000. Discriminator net structure: 50-50-25-1. Generator $g_\omega(\xi)$ structure: 48-48-32-24-12-1 with absolute value activation function in the output layer.

Contamination Q	JS-GAN (G_1)	JS-GAN (G_2)	Dimension Halving	Iterative Filtering
$\text{Cauchy}(1.5 * 1_p, I_p)$	**0.0664 (0.0065)**	0.0743 (0.0103)	0.3529 (0.0543)	0.1244 (0.0114)
$\text{Cauchy}(5.0 * 1_p, I_p)$	**0.0480 (0.0058)**	0.0540 (0.0064)	0.4855 (0.0616)	0.1687 (0.0310)
$\text{Cauchy}(1.5 * 1_p, 5 * I_p)$	0.0754 (0.0135)	**0.0742 (0.0111)**	0.3726 (0.0530)	0.1220 (0.0112)
$\text{Normal}(1.5 * 1_p, 5 * I_p)$	**0.0702 (0.0064)**	0.0713 (0.0088)	0.3915 (0.0232)	0.1048 (0.0288))

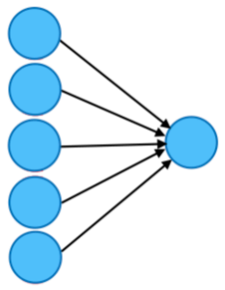
- *Dimension Halving:* [Lai et al.'16]
  https://github.com/kal2000/AgnosticMeanAndCovarianceCode.
- *Iterative Filtering:* [Diakonikolas et al.'17]
  https://github.com/hoonose/robust-filter.

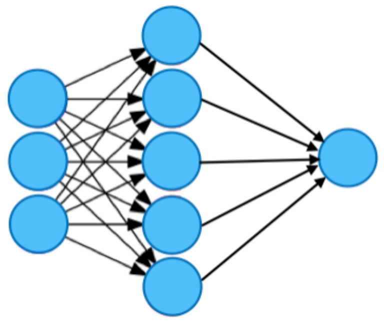
JS-GAN

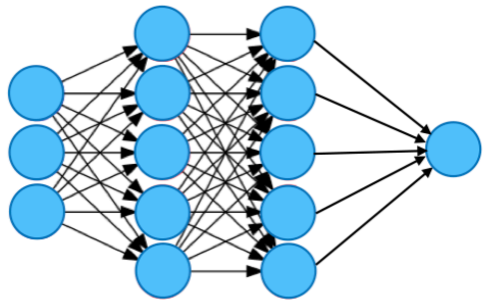
$$\widehat{\theta} = \operatorname*{argmin}_{\eta \in \mathbb{R}^p} \max_{T \in \mathcal{T}} \left[\frac{1}{n} \sum_{i=1}^{n} \log T(X_i) + E_\eta \log(1 - T(X)) \right] + \log 4$$

numerical experiment

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







$\widehat{\theta} \approx (1-\epsilon)\theta + \epsilon\widetilde{\theta}$ $\widehat{\theta} \approx \theta$ $\widehat{\theta} \approx \theta$

JS-GAN

A classifier with hidden layers leads to robustness. Why?

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

Proposition.

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

JS-GAN

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

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

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

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

JS-GAN

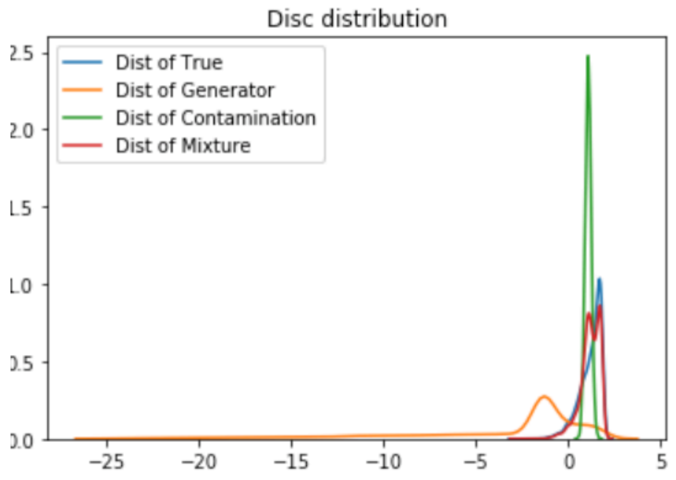
unknown covariance?

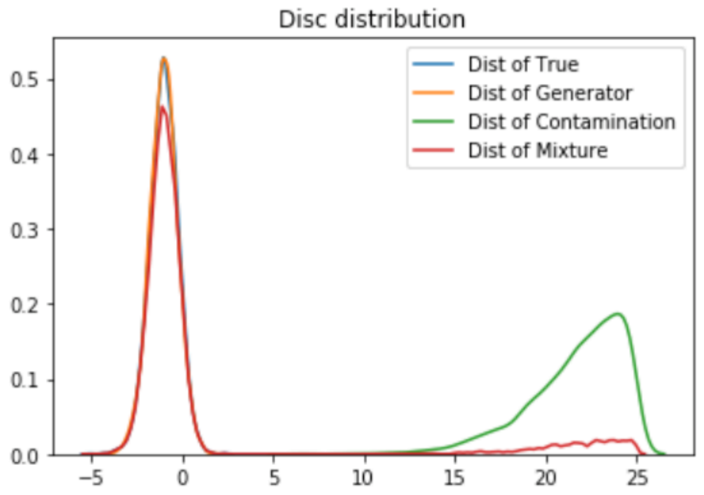
$$X_1, ..., X_n \sim (1-\epsilon)N(\theta, \Sigma) + \epsilon Q$$

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

no need to change the discriminator class

Discriminator identifies outliers



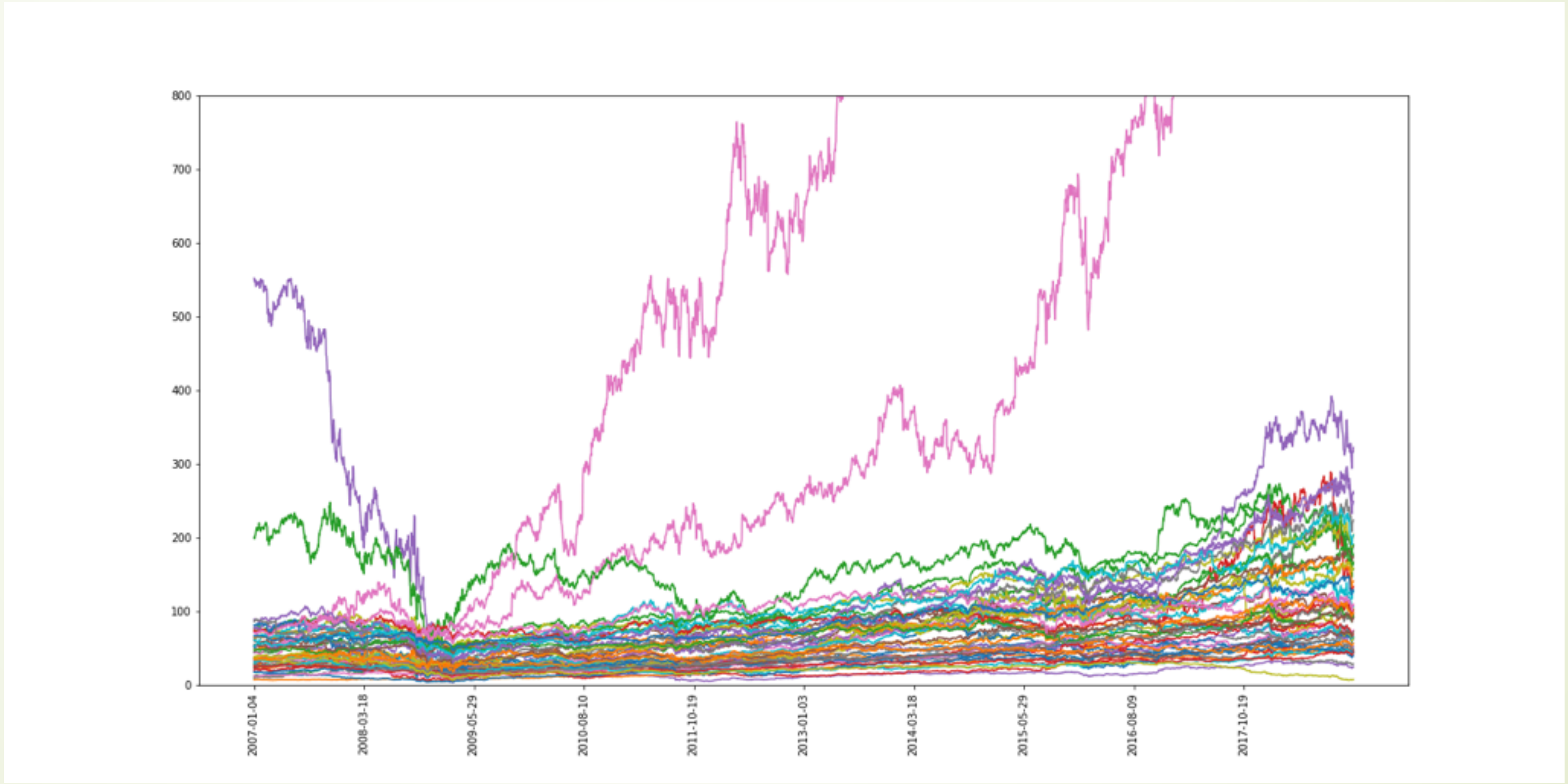


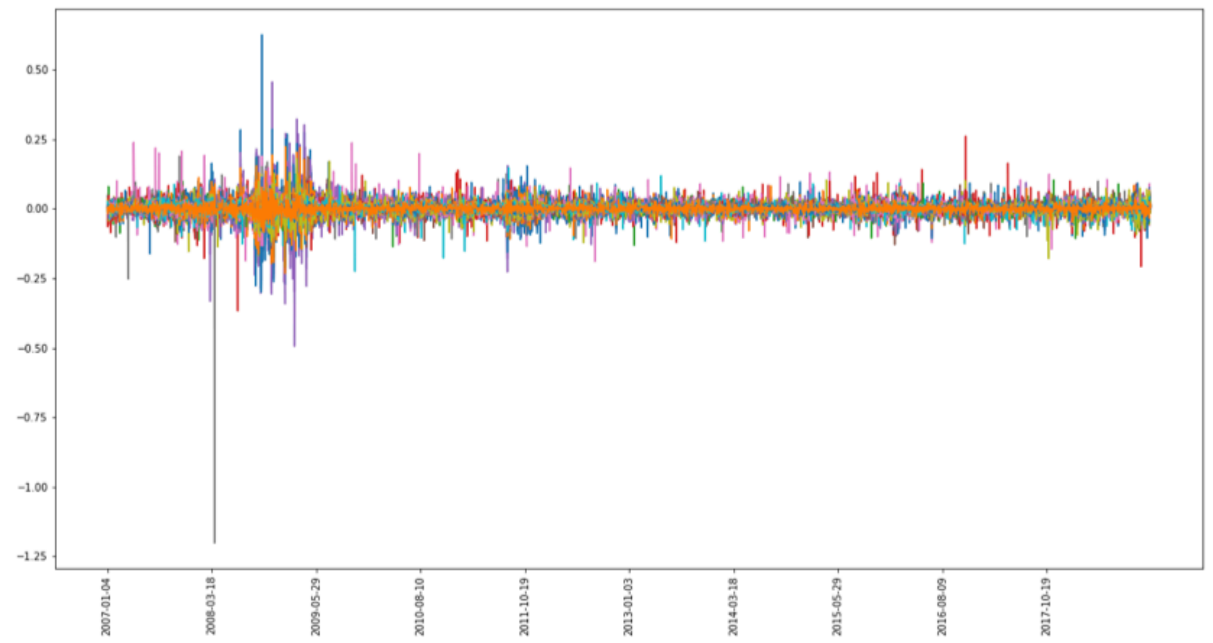
$$(1-\epsilon)N(0_p, I_p) + \epsilon Q$$

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

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

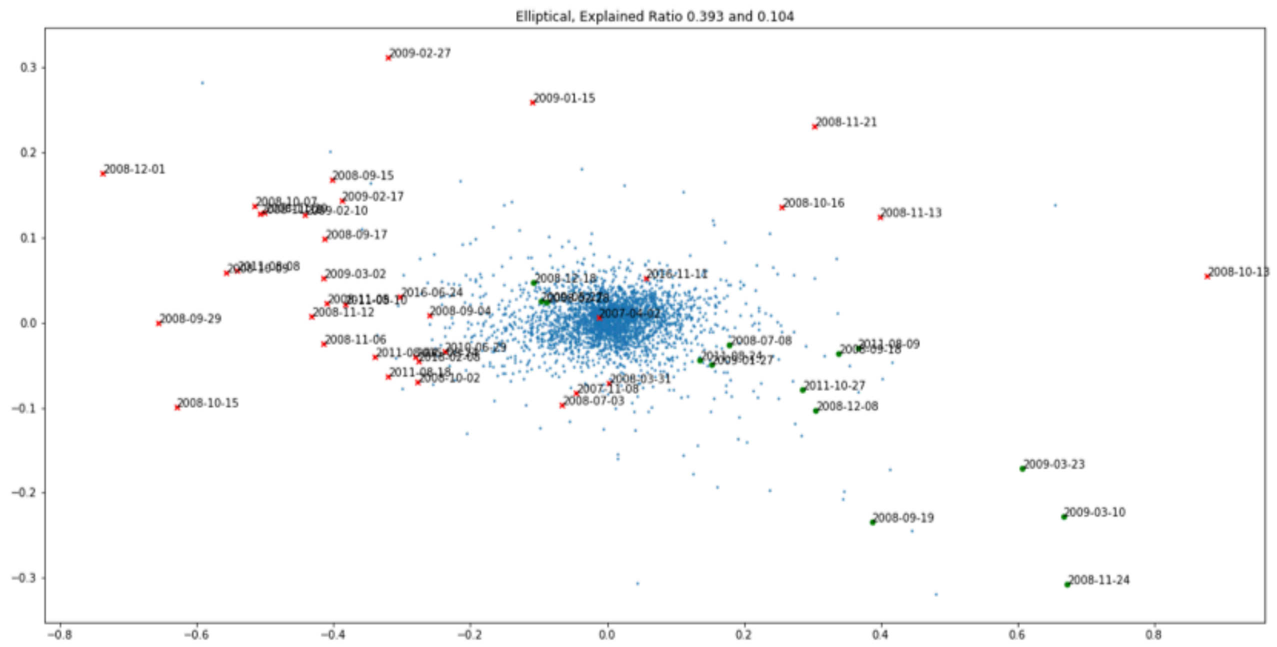
Application: Price of 50 stocks from 2007/01 to 2018/12 Corps are selected by ranking in market capitalization





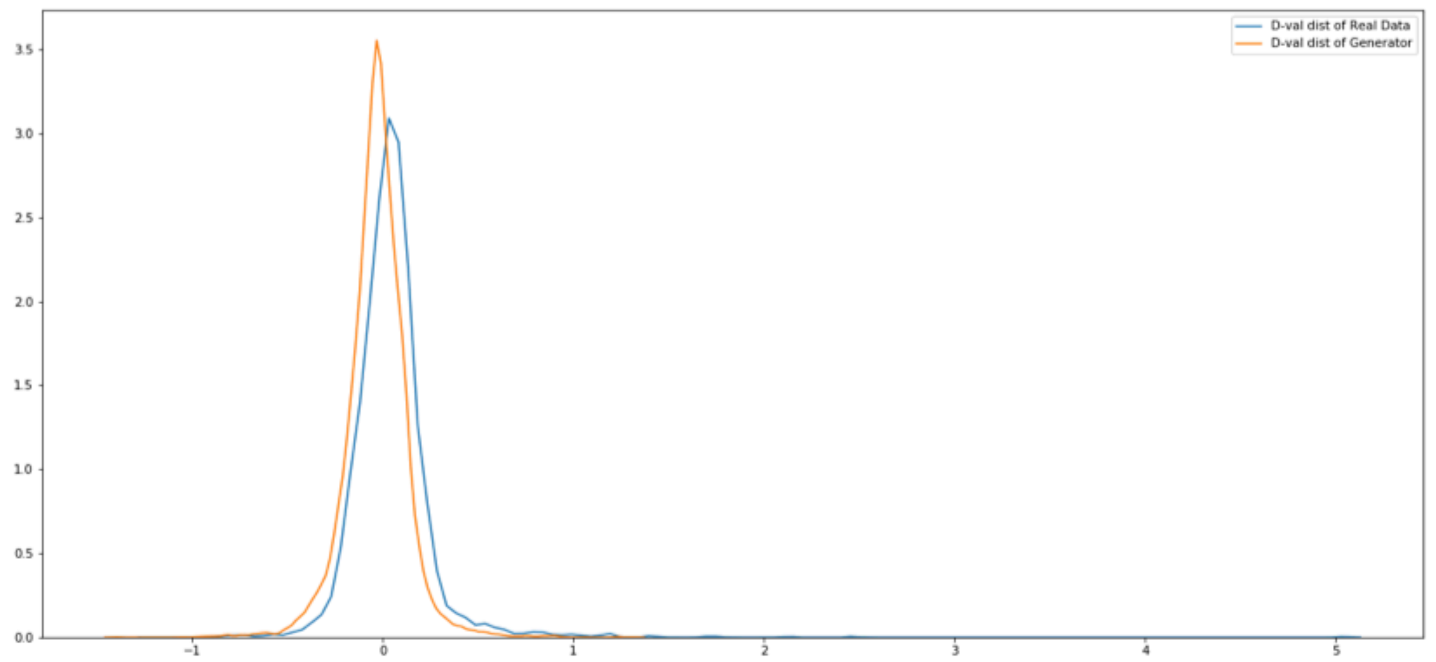
Log-return. y[i] = log(price_{i+1}/price_{i})

Robust PCA by Elliptical-Fitting GAN:



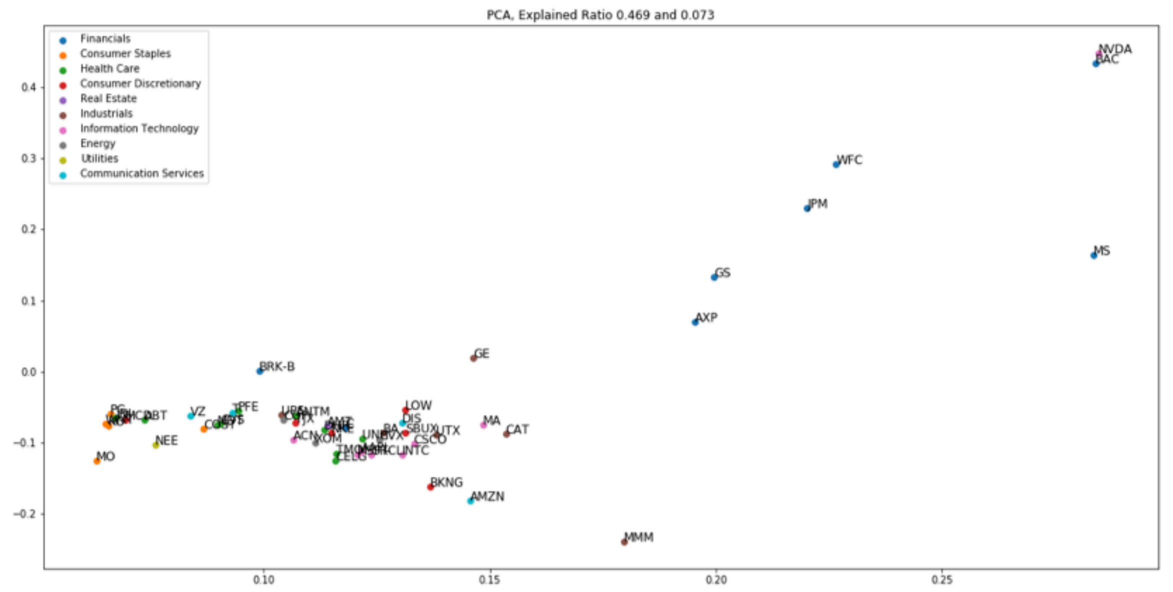
- **Fit data by Elliptical-GAN.**
  Apply SVD on scatter matrix.
  Dimension reduction on R^2.
  outlier x and o are selected from Discriminator value distribution.

Outlier Detection



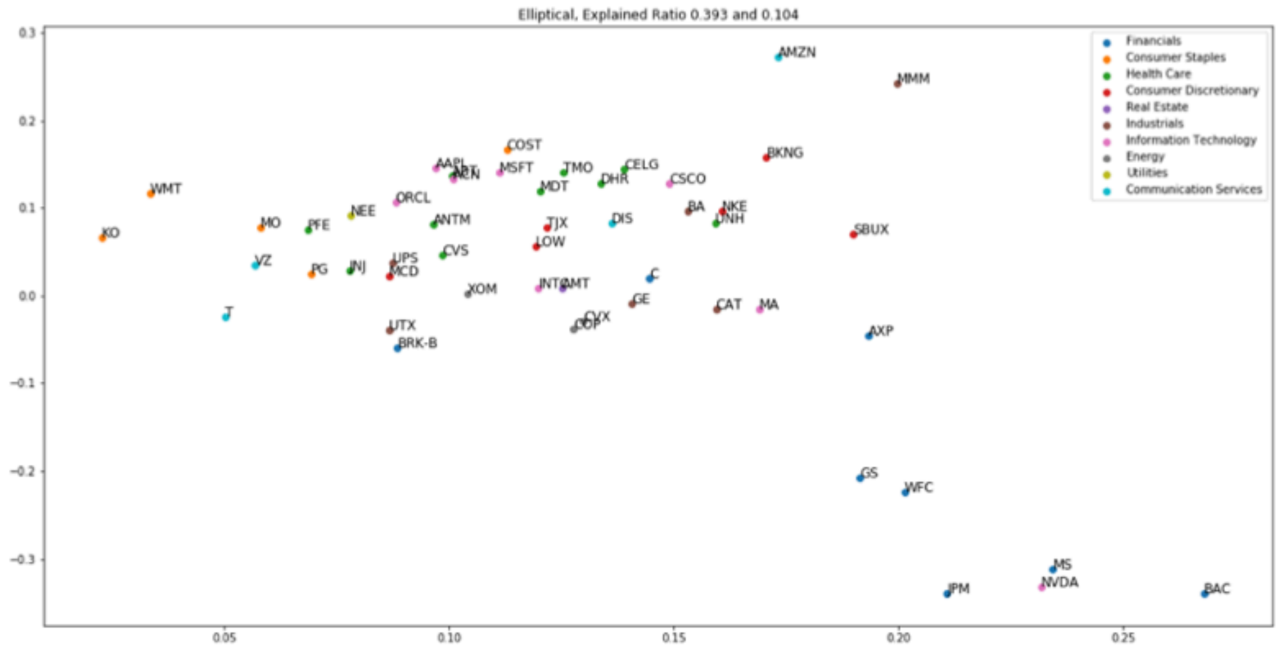
- **Discriminator value distribution from (Elliptical) Generator and real samples. Outliers are chosen from samples larger/ lower than a chosen percentile of Generator distribution**

Standard (non-robust) PCA:



First two direction are dominated by few financial corps —> not robust

Robust PCA by Elliptical GAN:



- **Loadings of Elliptical Scatter**
  Comparing with PCA, it's more robust in the sense that it does not totally dominate by Financial company (JPM, GS)

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!