2018 CS420, Machine Learning, Lecture 9

Unsupervised Learning

Weinan Zhang

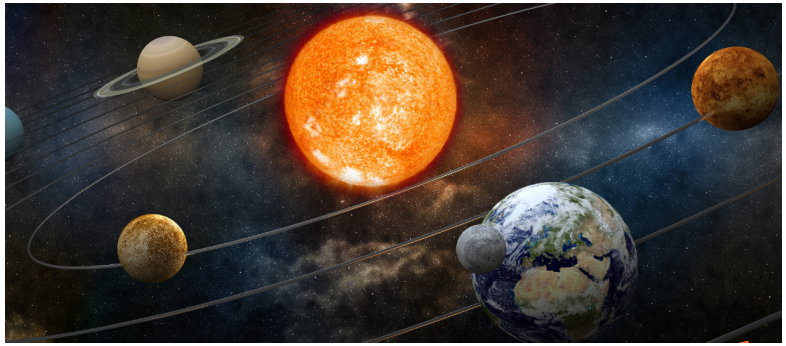
Shanghai Jiao Tong University

http://wnzhang.net

http://wnzhang.net/teaching/cs420/index.html

What is Data Science

- Physics
  - Goal: discover the underlying Principal of the world



  - Solution: build the model of the world from observations

$$F = G\frac{m_1 m_2}{r^2}$$

- Data Science
  - Goal: discover the underlying Principal of the data

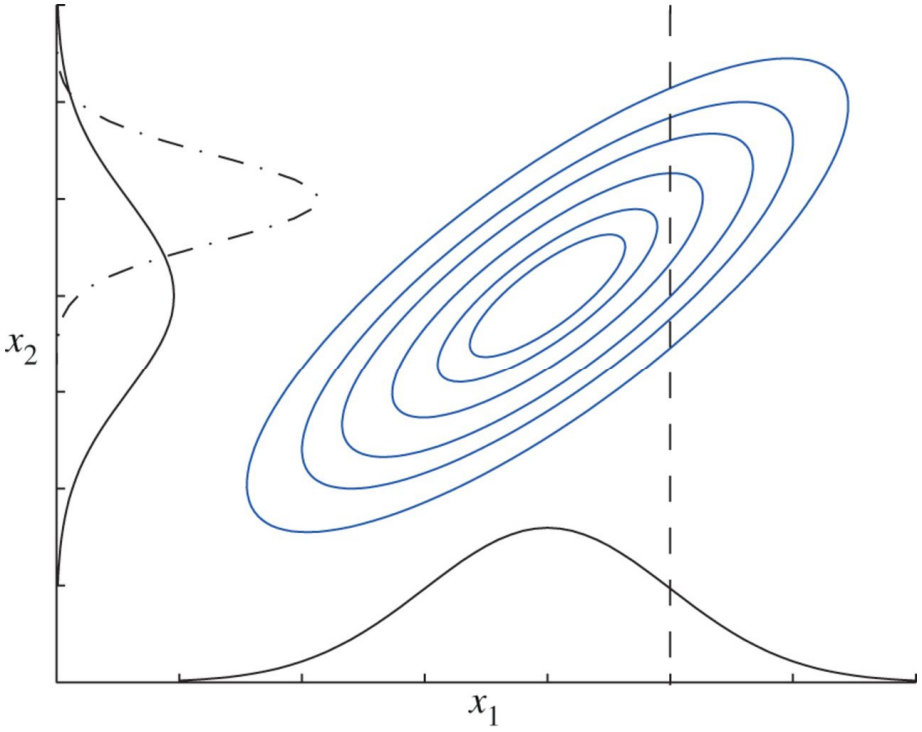
  - Solution: build the model of the data from observations

$$p(x) = \frac{e^{f(x)}}{\sum_{x'} e^{f(x')}}$$

Data Science

- Mathematically
  - Find joint data distribution $p(x)$
  - Then the conditional distribution $p(x_2|x_1)$
- Gaussian distribution
  - Multivariate

$$p(x) = \frac{e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)}}{\sqrt{|2\pi\Sigma|}}$$



- Univariate

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

A Simple Example in User Behavior Modelling

Interest	Gender	Age	BBC Sports	PubMed	Bloomberg Business	Spotify
Finance	Male	29	Yes	No	Yes	No
Sports	Male	21	Yes	No	No	Yes
Medicine	Female	32	No	Yes	No	No
Music	Female	25	No	No	No	Yes
Medicine	Male	40	Yes	Yes	Yes	No

- Joint data distribution

p(Interest=Finance, Gender=Male, Age=29, Browsing=BBC Sports,Bloomberg Business)

- Conditional data distribution

p(Interest=Finance | Browsing=BBC Sports,Bloomberg Business)

p(Gender=Male | Browsing=BBC Sports,Bloomberg Business)

Problem Setting

- First build and learn $p(x)$ and then infer the conditional dependence $p(x_t|x_i)$
  - Unsupervised learning
  - Each dimension of x is equally treated

- Directly learn the conditional dependence $p(x_t|x_i)$
  - Supervised learning
  - x_t is the label to predict

Definition of Unsupervised Learning

- Given the training dataset

$$D = \{x_i\}_{i=1,2,...,N}$$

  let the machine learn the data underlying patterns

  - Latent variables

$$z \rightarrow x$$

  - Probabilistic density function (p.d.f.) estimation

$$p(x)$$

  - Good data representation (used for discrimination)

$$\phi(x)$$

Uses of Unsupervised Learning

- Data structure discovery, data science
- Data compression
- Outlier detection
- Input to supervised/reinforcement algorithms (causes may be more simply related to outputs or rewards)
- A theory of biological learning and perception

Slide credit: Maneesh Sahani

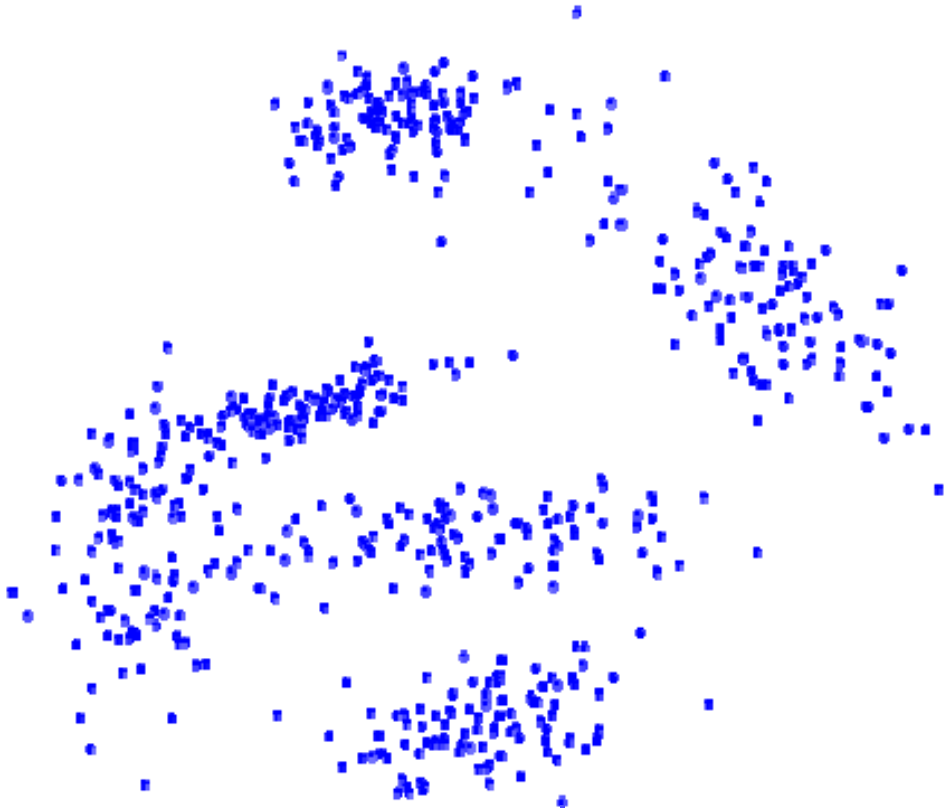
Content

- Fundamentals of Unsupervised Learning
  - K-means clustering
  - Principal component analysis
- Probabilistic Unsupervised Learning
  - Mixture Gaussians
  - EM Methods
- Deep Unsupervised Learning
  - Auto-encoders
  - Generative adversarial nets

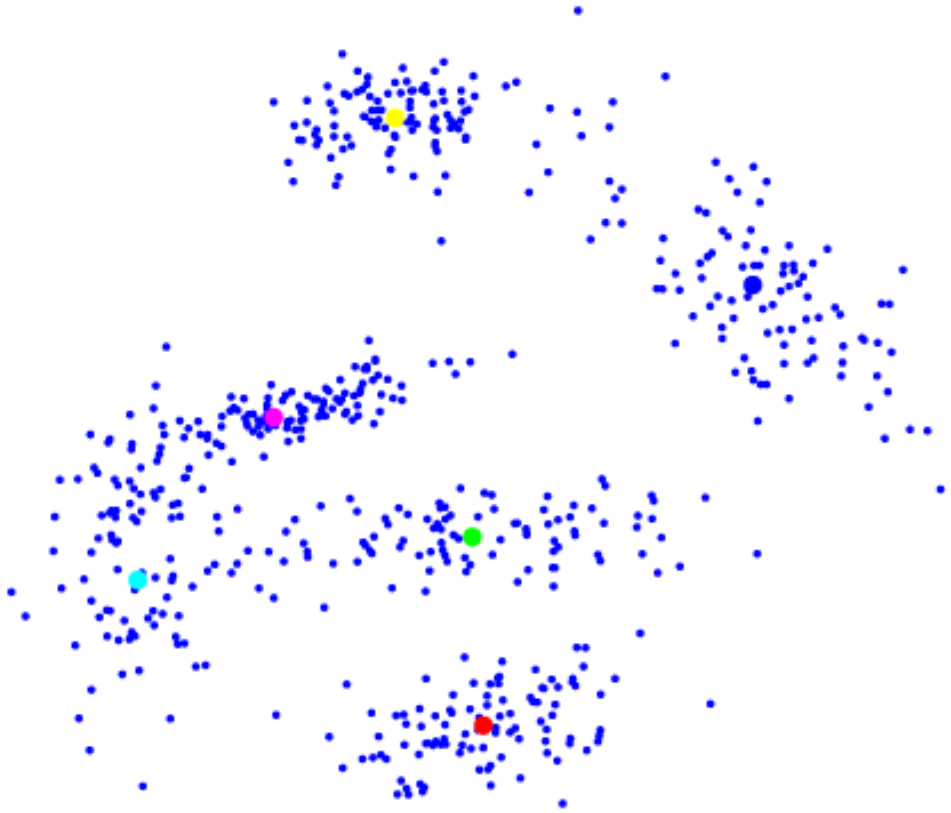
Content

- Fundamentals of Unsupervised Learning
  - K-means clustering
  - Principal component analysis
- Probabilistic Unsupervised Learning
  - Mixture Gaussians
  - EM Methods
- Deep Unsupervised Learning
  - Auto-encoders
  - Generative adversarial nets

K-Means Clustering



K-Means Clustering



K-Means Clustering

- Provide the number of desired clusters *k*
- Randomly choose *k* instances as seeds, one per each cluster, i.e. the centroid for each cluster
- Iterate
  - Assign each instance to the cluster with the closest centroid
  - Re-estimate the centroid of each cluster
- Stop when clustering converges
  - Or after a fixed number of iterations

Slide credit: Ray Mooney

K-Means Clustering: Centriod

- Assume instances are real-valued vectors

$$x \in \mathbb{R}^d$$

- Clusters based on centroids, center of gravity, or mean of points in a cluster C_k

$$\mu^k = \frac{1}{C_k} \sum_{x \in C_k} x$$

Slide credit: Ray Mooney

K-Means Clustering: Distance

- Distance to a centroid $L(x, \mu^k)$
- Euclidian distance (L2 norm)

$$L_2(x, \mu^k) = \|x - \mu^k\| = \sqrt{\sum_{m=1}^{d} (x_i - \mu_m^k)^2}$$

- Euclidian distance (L1 norm)

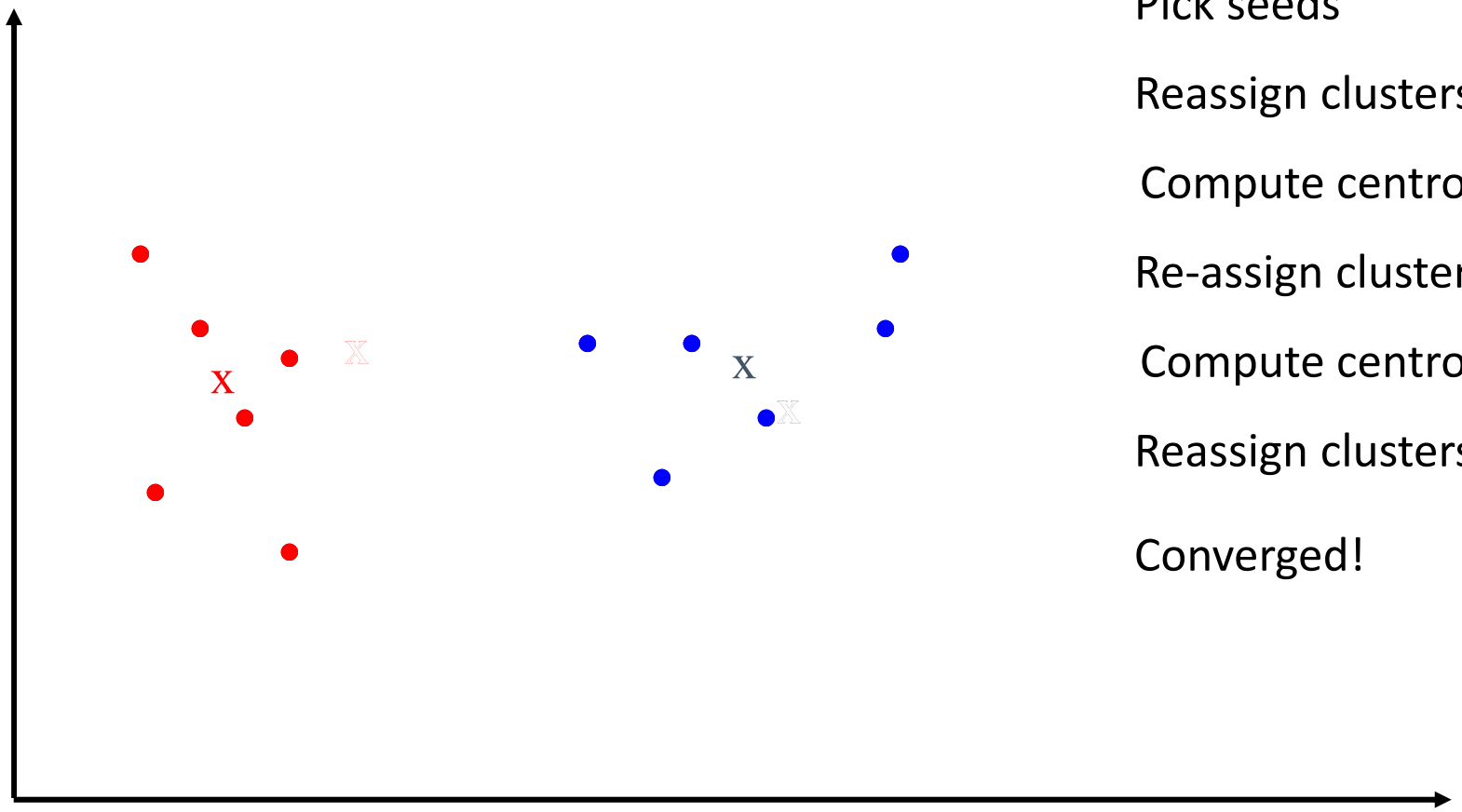
$$L_1(x, \mu^k) = |x - \mu^k| = \sum_{m=1}^{d} |x_i - \mu_m^k|$$

- Cosine distance

$$L_{\cos}(x, \mu^k) = 1 - \frac{x^\top \mu^k}{|x| \cdot |\mu^k|}$$

Slide credit: Ray Mooney

K-Means Example (K=2)



Pick seeds

Reassign clusters

Compute centroids

Re-assign clusters

Compute centroids

Reassign clusters

Converged!

Slide credit: Ray Mooney

K-Means Time Complexity

- Assume computing distance between two instances is $O(d)$ where d is the dimensionality of the vectors
- Reassigning clusters: $O(knd)$ distance computations
- Computing centroids: Each instance vector gets added once to some centroid: $O(nd)$
- Assume these two steps are each done once for I iterations: $O(Iknd)$

Slide credit: Ray Mooney

K-Means Clustering Objective

- The objective of *K*-means is to minimize the total sum of the squared distance of every point to its corresponding cluster centroid

$$\min_{\{\mu^k\}_{k=1}^K} \sum_{k=1}^{K} \sum_{x \in C_k} L(x - \mu^k) \qquad \mu^k = \frac{1}{C_k} \sum_{x \in C_k} x$$

- Finding the global optimum is NP-hard.
- The *K*-means algorithm is guaranteed to converge to a local optimum.

Seed Choice

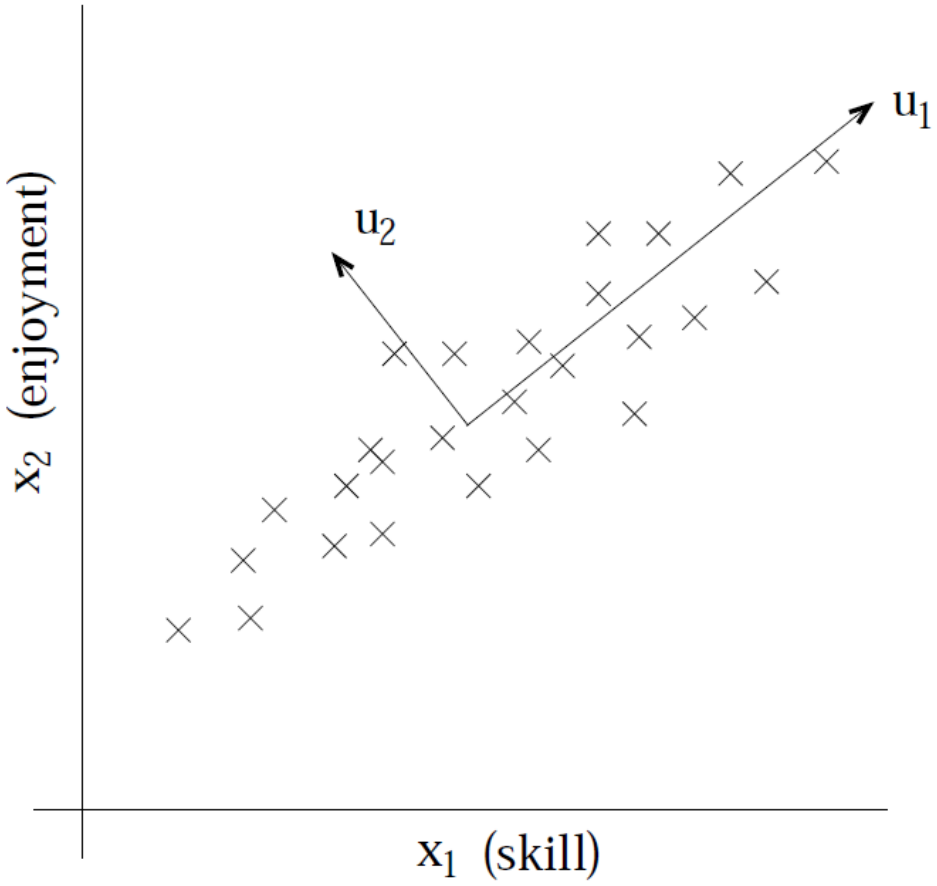
- Results can vary based on random seed selection.
- Some seeds can result in poor convergence rate, or convergence to sub-optimal clusterings.
- Select good seeds using a heuristic or the results of another method.

Clustering Applications

- Text mining
  - Cluster documents for related search
  - Cluster words for query suggestion
- Recommender systems and advertising
  - Cluster users for item/ad recommendation
  - Cluster items for related item suggestion
- Image search
  - Cluster images for similar image search and duplication detection
- Speech recognition or separation
  - Cluster phonetical features

Principal Component Analysis (PCA)

- An example of 2-dimensional data
  - x_1: the piloting skill of pilot
  - x_2: how much he/she enjoys flying
- Main components
  - u_1: intrinsic piloting "karma" of a person
  - u_2: some noise



Example credit: Andrew Ng

Principal Component Analysis (PCA)

- PCA tries to identify the subspace in which the data approximately lies

- PCA uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.
  - The number of principal components is less than or equal to the smaller of the number of original variables or the number of observations.

$$\mathbb{R}^d \rightarrow \mathbb{R}^k \qquad k \ll d$$

PCA Data Preprocessing

- Given the dataset

$$D = \{x^{(i)}\}_{i=1}^{m}$$

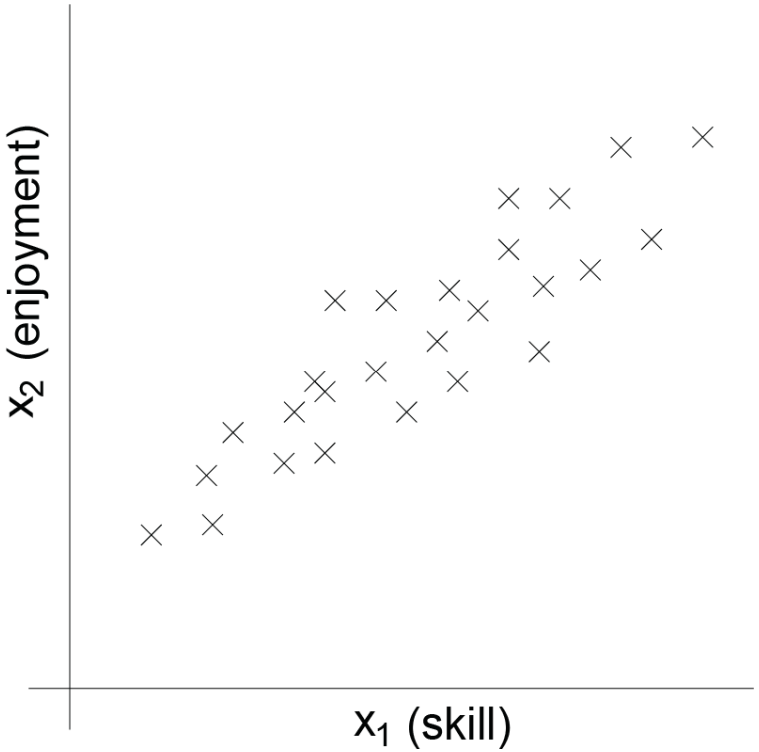
- Typically we first pre-process the data to normalize its mean and variance
  1. Move the central of the data set to 0

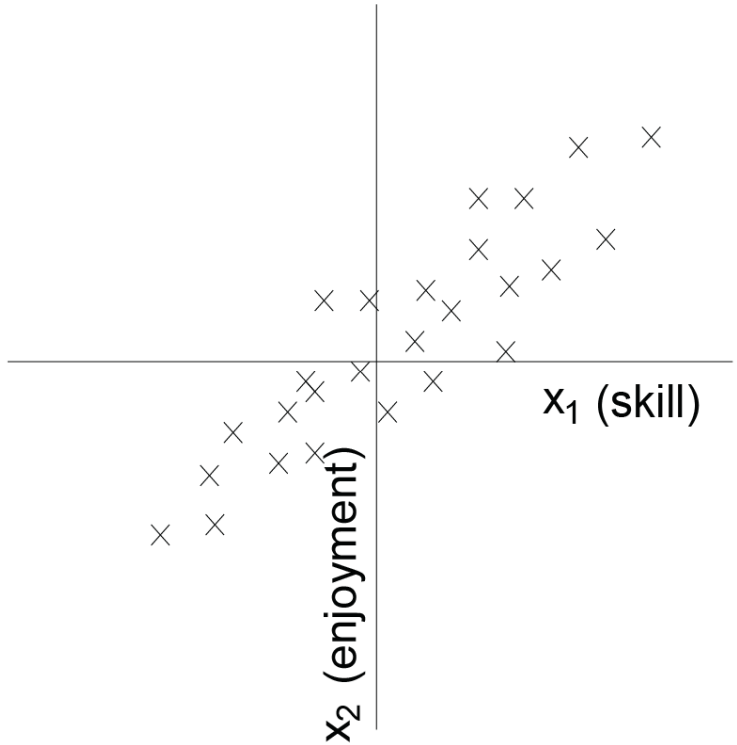
$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \qquad\qquad x^{(i)} \leftarrow x^{(i)} - \mu$$

  2. Unify the variance of each variable

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m} (x_j^{(i)})^2 \qquad\qquad x^{(i)} \leftarrow x^{(i)}/\sigma_j$$

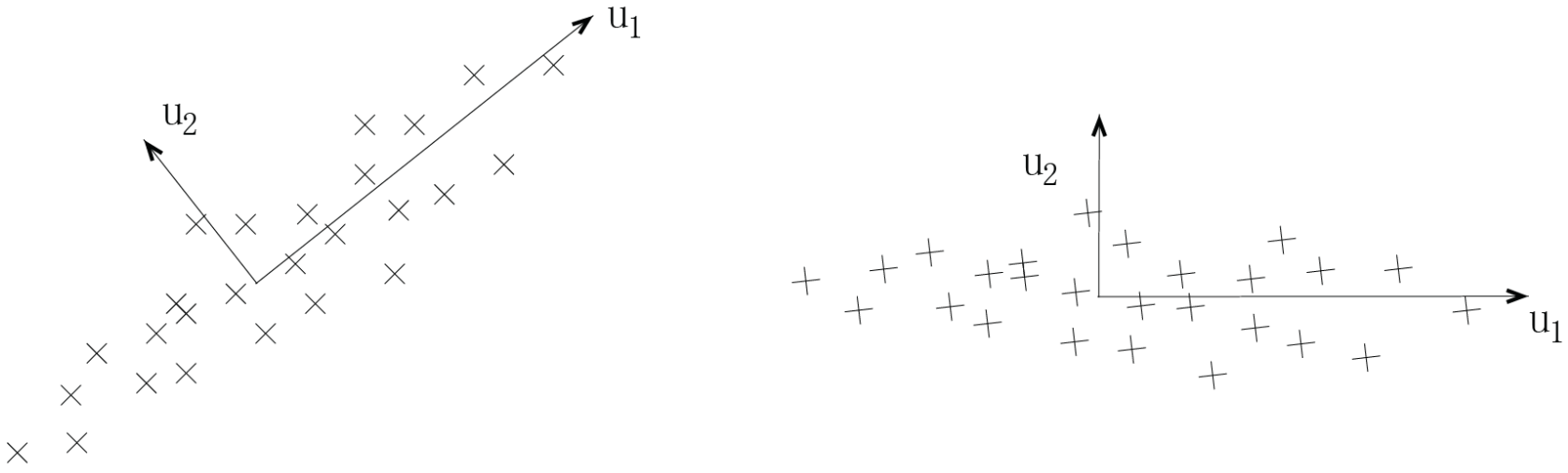
PCA Data Preprocessing





- Zero out the mean of the data
- Rescale each coordinate to have unit variance, which ensures that different attributes are all treated on the same “scale”.

PCA Solution



- PCA finds the directions with the largest variable variance
  - which correspond to the eigenvectors of the matrix $X^T X$ with the largest eigenvalues

PCA Solution: Data Projection

- The projection of each point $x^{(i)}$ to a direction u $\quad (\|u\| = 1)$

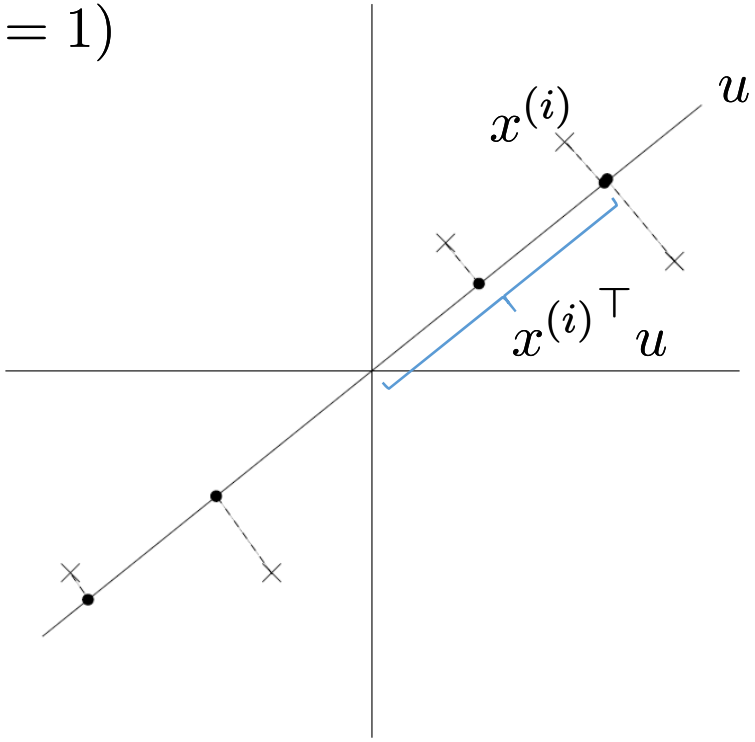
$$x^{(i)\top} u$$

- The variance of the projection

$$\frac{1}{m}\sum_{i=1}^{m}(x^{(i)\top}u)^2 = \frac{1}{m}\sum_{i=1}^{m} u^\top x^{(i)} x^{(i)\top} u$$

$$= u^\top \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)\top}\right) u$$

$$\equiv u^\top \Sigma u$$



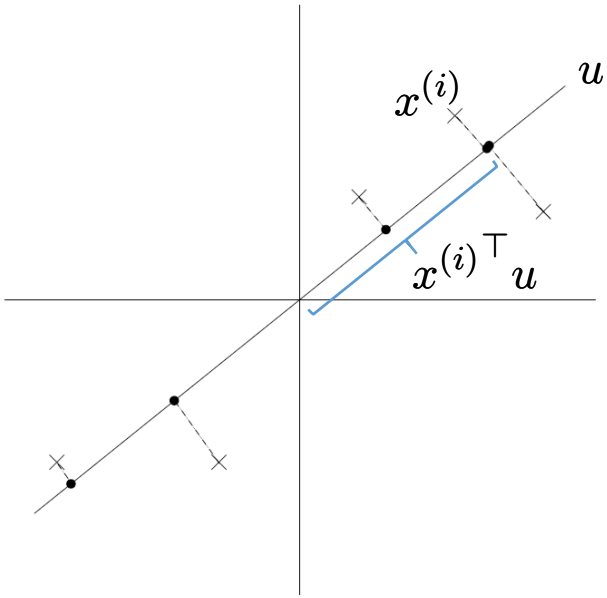
PCA Solution: Largest Eigenvalues

$$\max_u \quad u^\top \Sigma u \qquad\qquad \Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)\top}$$

$$\text{s.t.} \quad \|u\| = 1$$

- Find *k* principal components of the data is to find the *k* principal eigenvectors of Σ
  - i.e. the top-*k* eigenvectors with the largest eigenvalues
- Projected vector for $x^{(i)}$

$$y^{(i)} = \begin{bmatrix} u_1^\top x^{(i)} \\ u_2^\top x^{(i)} \\ \vdots \\ u_k^\top x^{(i)} \end{bmatrix} \in \mathbb{R}^k$$



Eigendecomposition Revisit

- For a semi-positive square matrix $\Sigma_{d\times d}$
  - suppose *u* to be its eigenvector $(\|u\| = 1)$
  - with the scalar eigenvalue *w* $\Sigma u = wu$
  - There are *d* eigenvectors-eigenvalue pairs (u_i, w_i)
  - These *d* eigenvectors are orthogonal, thus they form an orthonormal basis

$$\sum_{i=1}^{d} u_i u_i^\top = I$$

  - Thus any vector *v* can be written as

$$v = \Big(\sum_{i=1}^{d} u_i u_i^\top\Big) v = \sum_{i=1}^{d} (u_i^\top v) u_i = \sum_{i=1}^{d} v_{(i)} u_i$$

  - $\Sigma_{d\times d}$ can be written as

$$\Sigma = \sum_{i=1}^{d} u_i u_i^\top \Sigma = \sum_{i=1}^{d} w_i u_i u_i^\top = UWU^\top$$

$$U = [u_1, u_2, \ldots, u_d]$$

$$W = \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & w_d \end{bmatrix}$$

Eigendecomposition Revisit

- Given the data $X = \begin{bmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_n^\top \end{bmatrix}$ and its covariance matrix $\Sigma = X^\top X$

  (here we may drop *m* for simplicity)

- The variance in direction u_i is

$$\|Xu_i\|^2 = u_i^\top X^\top X u_i = u_i^\top \Sigma u_i = u_i^\top w_i u_i = w_i$$

- The variance in any direction *v* is

$$\|Xv\|^2 = \left\|X\left(\sum_{i=1}^{d} v_{(i)} u_i\right)\right\|^2 = \sum_{ij} v_{(i)} u_i^\top \Sigma u_i v_{(j)} = \sum_{i=1}^{d} v_{(i)}^2 w_i$$

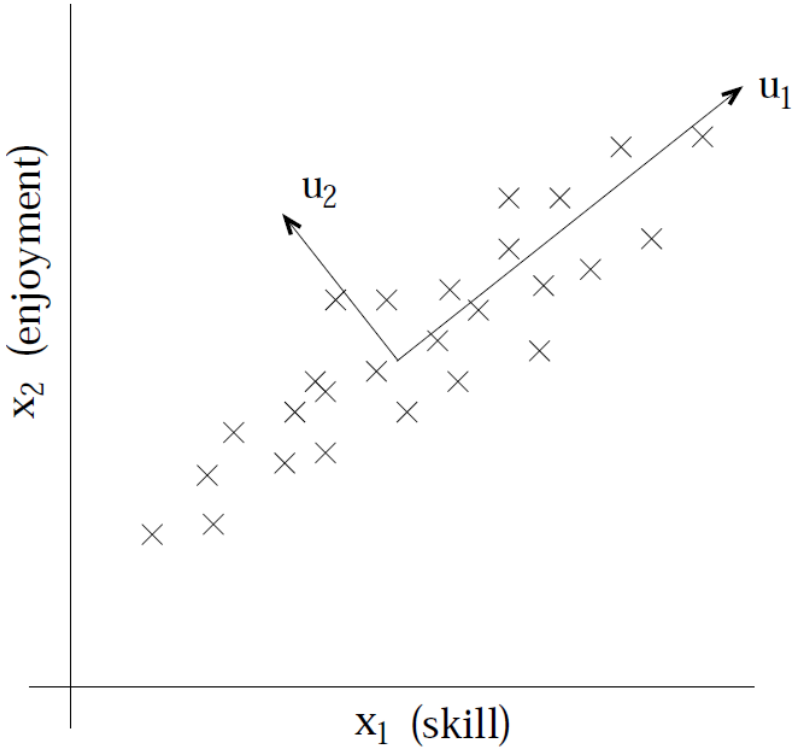
  where $v_{(i)}$ is the projection length of *v* on u_i

- If $v^\top v = 1$, then $\arg\max_{\|v\|=1} \|Xv\|^2 = u_{(\max)}$

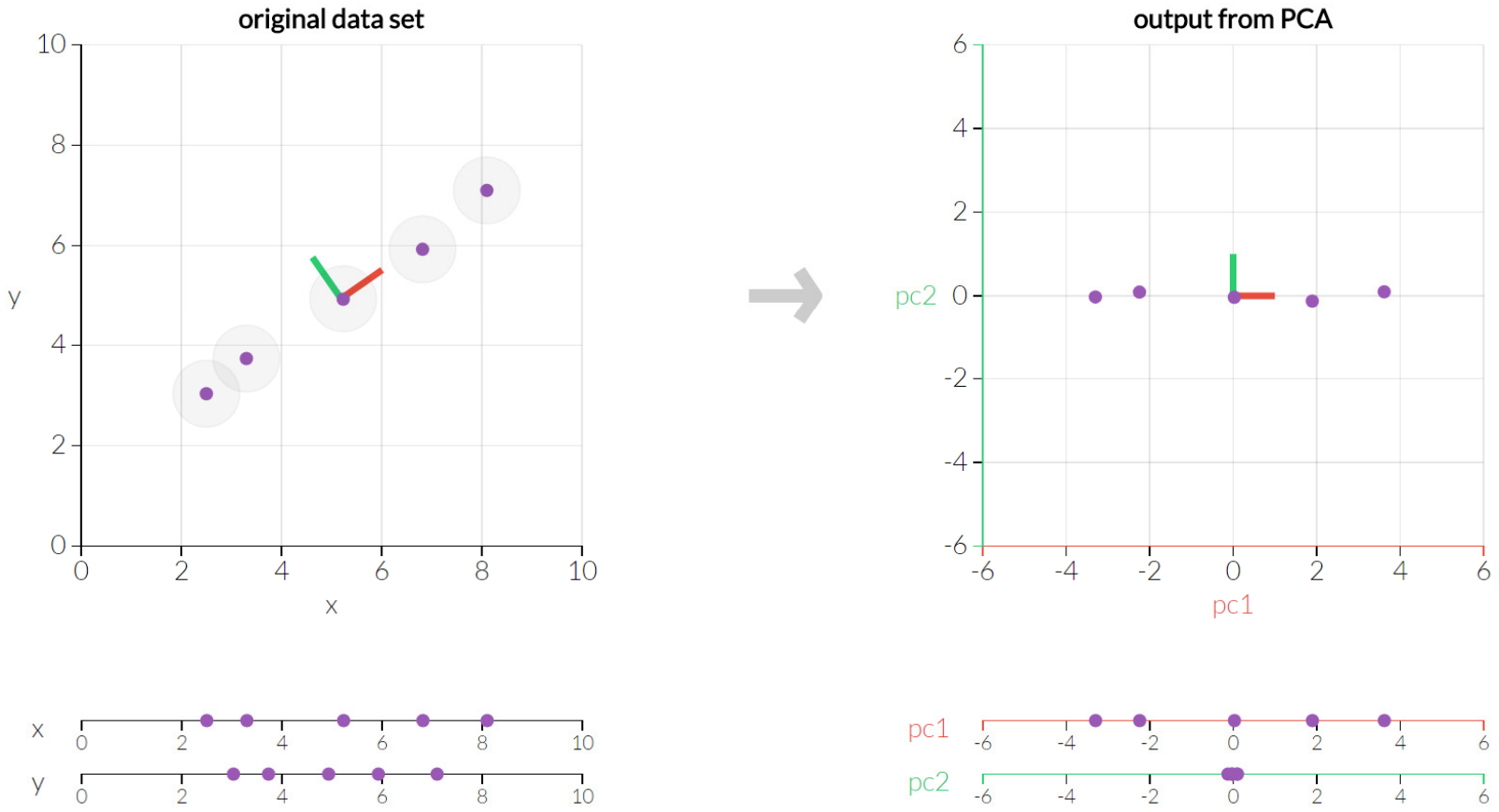
The direction of greatest variance is the eigenvector with the largest eigenvalue

PCA Discussion

- PCA can also be derived by picking the basis that minimizes the approximation error arising from projecting the data onto the *k*-dimensional subspace spanned by them.

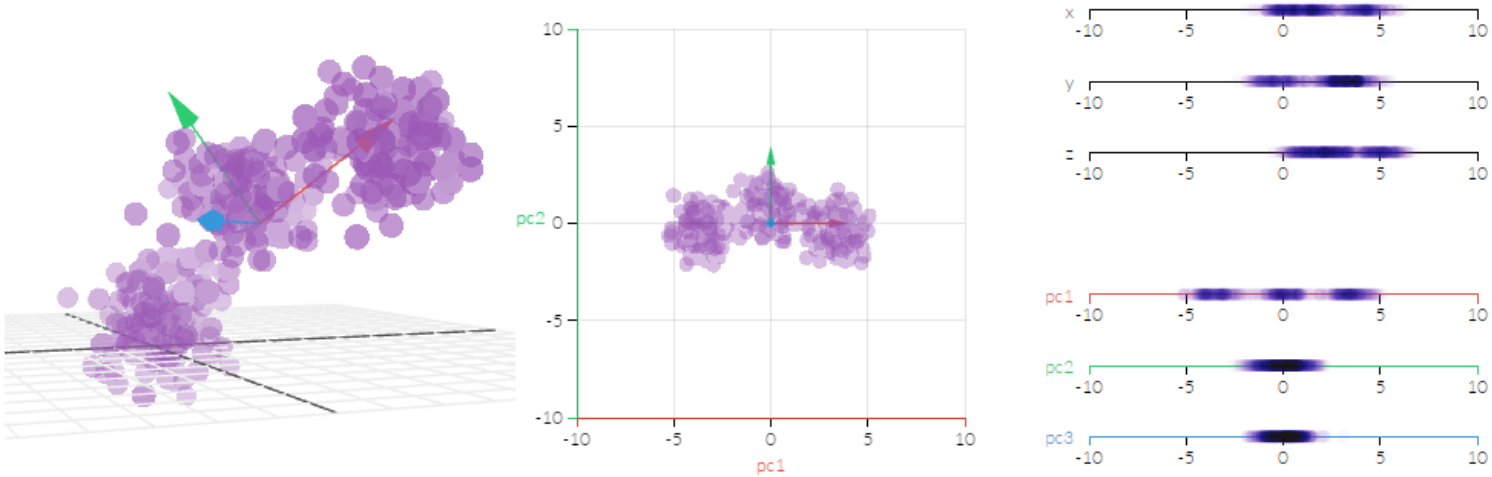


PCA Visualization



http://setosa.io/ev/principal-component-analysis/

PCA Visualization

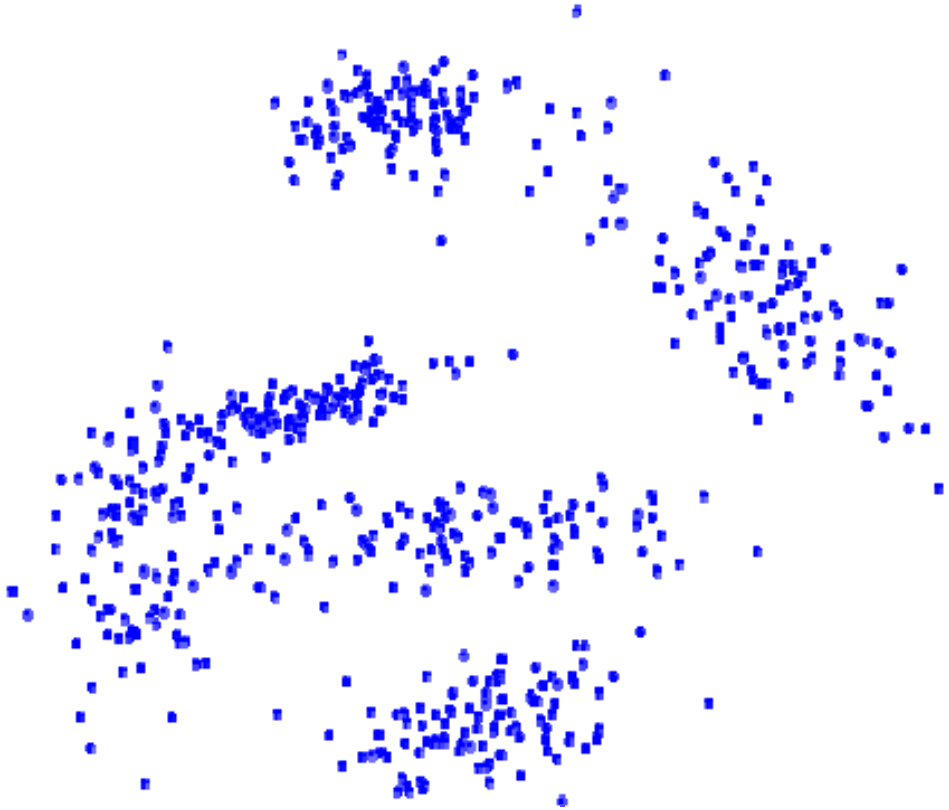


http://setosa.io/ev/principal-component-analysis/

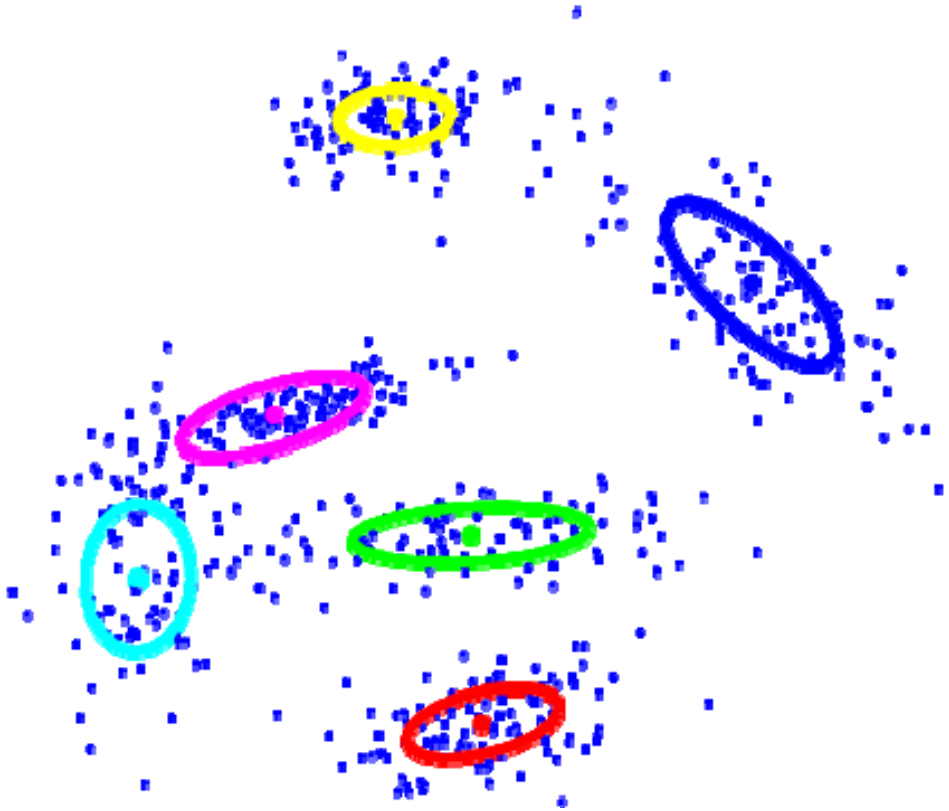
Content

- Fundamentals of Unsupervised Learning
  - K-means clustering
  - Principal component analysis
- Probabilistic Unsupervised Learning
  - Mixture Gaussians
  - EM Methods
- Deep Unsupervised Learning
  - Auto-encoders
  - Generative adversarial nets

Mixture Gaussian



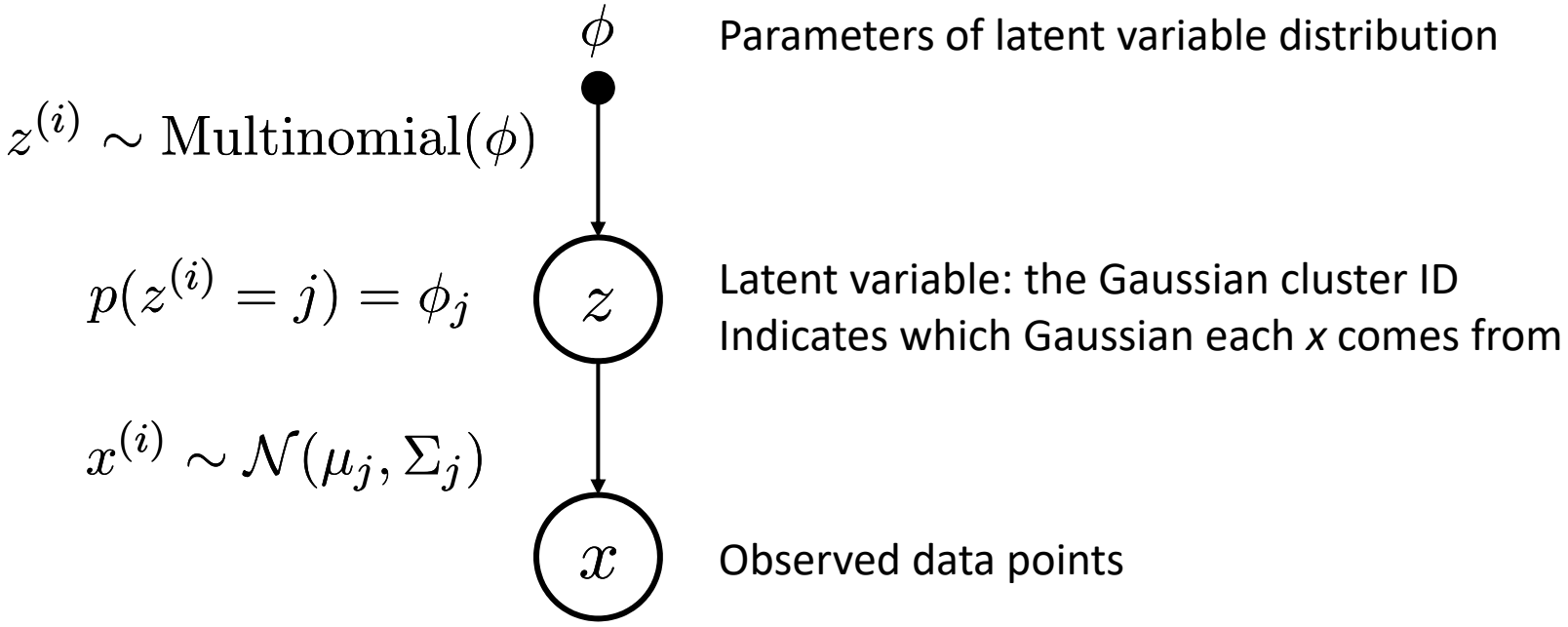
Mixture Gaussian



Graphic Model for Mixture Gaussian

- Given a training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$
- Model the data by specifying a joint distribution

$$p(x^{(i)}, z^{(i)}) = p(x^{(i)}|z^{(i)})p(z^{(i)})$$



Data Likelihood

- We want to maximize

$$
\begin{aligned}
l(\phi, \mu, \Sigma) &= \sum_{i=1}^{m} \log p(x^{(i)}; \phi, \mu, \Sigma) \\
&= \sum_{i=1}^{m} \log \sum_{z^{(i)}=1}^{k} p(x^{(i)}|z^{(i)}; \mu, \Sigma) p(z^{(i)}; \phi) \\
&= \sum_{i=1}^{m} \log \sum_{j=1}^{k} \mathcal{N}(x^{(i)}|\mu_j, \Sigma_j) \phi_j
\end{aligned}
$$

- No closed form solution by simply setting

$$
\frac{\partial l(\phi, \mu, \Sigma)}{\partial \phi} = 0 \qquad \frac{\partial l(\phi, \mu, \Sigma)}{\partial \mu} = 0 \qquad \frac{\partial l(\phi, \mu, \Sigma)}{\partial \Sigma} = 0
$$

Data Likelihood Maximization

- For each data point $x^{(i)}$, latent variable $z^{(i)}$ indicates which Gaussian it comes from
- If we knew $z^{(i)}$, the data likelihood

$$
\begin{aligned}
l(\phi, \mu, \Sigma) &= \sum_{i=1}^{m} \log p(x^{(i)}; \phi, \mu, \Sigma) \\
&= \sum_{i=1}^{m} \log p(x^{(i)} | z^{(i)}; \mu, \Sigma) p(z^{(i)}; \phi) \\
&= \sum_{i=1}^{m} \log \mathcal{N}(x^{(i)} | \mu_{z^{(i)}}, \Sigma_{z^{(i)}}) + \log p(z^{(i)}; \phi)
\end{aligned}
$$

Data Likelihood Maximization

- Given $z^{(i)}$, maximize the data likelihood

$$\max_{\phi,\mu,\Sigma} l(\phi,\mu,\Sigma) = \max_{\phi,\mu,\Sigma} \sum_{i=1}^{m} \log \mathcal{N}(x^{(i)}|\mu_{z^{(i)}},\Sigma_{z^{(i)}}) + \log p(z^{(i)};\phi)$$

- It is easy to get the solution

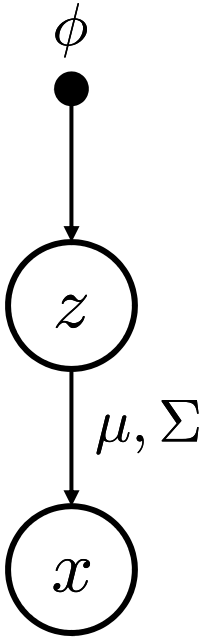
$$\phi_j = \frac{1}{m}\sum_{i=1}^{m} 1\{z^{(i)}=j\}$$

$$\mu_j = \frac{\sum_{i=1}^{m} 1\{z^{(i)}=j\}x^{(i)}}{\sum_{i=1}^{m} 1\{z^{(i)}=j\}}$$

$$\Sigma_j = \frac{\sum_{i=1}^{m} 1\{z^{(i)}=j\}(x^{(i)}-\mu_j)(x^{(i)}-\mu_j)^\top}{\sum_{i=1}^{m} 1\{z^{(i)}=j\}}$$

Latent Variable Inference

- Given the parameters μ, Σ, ϕ, it is not hard to infer the posterior of the latent variable $z^{(i)}$ for each instance



$$p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(z^{(i)} = j, x^{(i)}; \phi, \mu, \Sigma)}{p(x^{(i)}; \phi, \mu, \Sigma)}$$

$$= \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{\sum_{l=1}^{k} p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)}$$

where

- The prior of $z^{(i)}$ is $p(z^{(i)} = j; \phi)$
- The likelihood is $p(x^{(i)} | z^{(i)} = j; \mu, \Sigma)$

Expectation Maximization Methods

- E-step: infer the posterior distribution of the latent variables given the model parameters
- M-step: tune parameters to maximize the data likelihood given the latent variable distribution

- EM methods
  - Iteratively execute E-step and M-step until convergence

EM Methods for Mixture Gaussians

- Mixture Gaussian example

Repeat until convergence: {

(E-step) For each *i*, *j*, set

$$w_j^{(i)} = p(z^{(i)} = j, x^{(i)}; \phi, \mu, \Sigma)$$

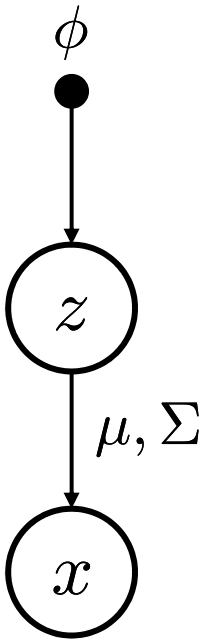
(M-step) Update the parameters

$$\phi_j = \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)}$$

$$\mu_j = \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}}$$

$$\Sigma_j = \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^\top}{\sum_{i=1}^{m} w_j^{(i)}}$$

}



General EM Methods

- Claims:

1. After each E-M step, the data likelihood will not decrease.
2. The EM algorithm finds a (local) maximum of a latent variable model likelihood

- Now let's discuss the general EM methods and verify its effectiveness of improving data likelihood and its convergence

Jensen's Inequality

Theorem. Let *f* be a convex function, and let *X* be a random variable.

Then:

$$\mathbb{E}[f(X)] \geq f(\mathbb{E}[X])$$

- Moreover, if *f* is strictly convex, then

  $$\mathbb{E}[f(X)] = f(\mathbb{E}[X])$$

  holds true if and only if

  $$X = \mathbb{E}[X]$$

  with probability 1 (i.e., if *X* is a constant).

Jensen's Inequality

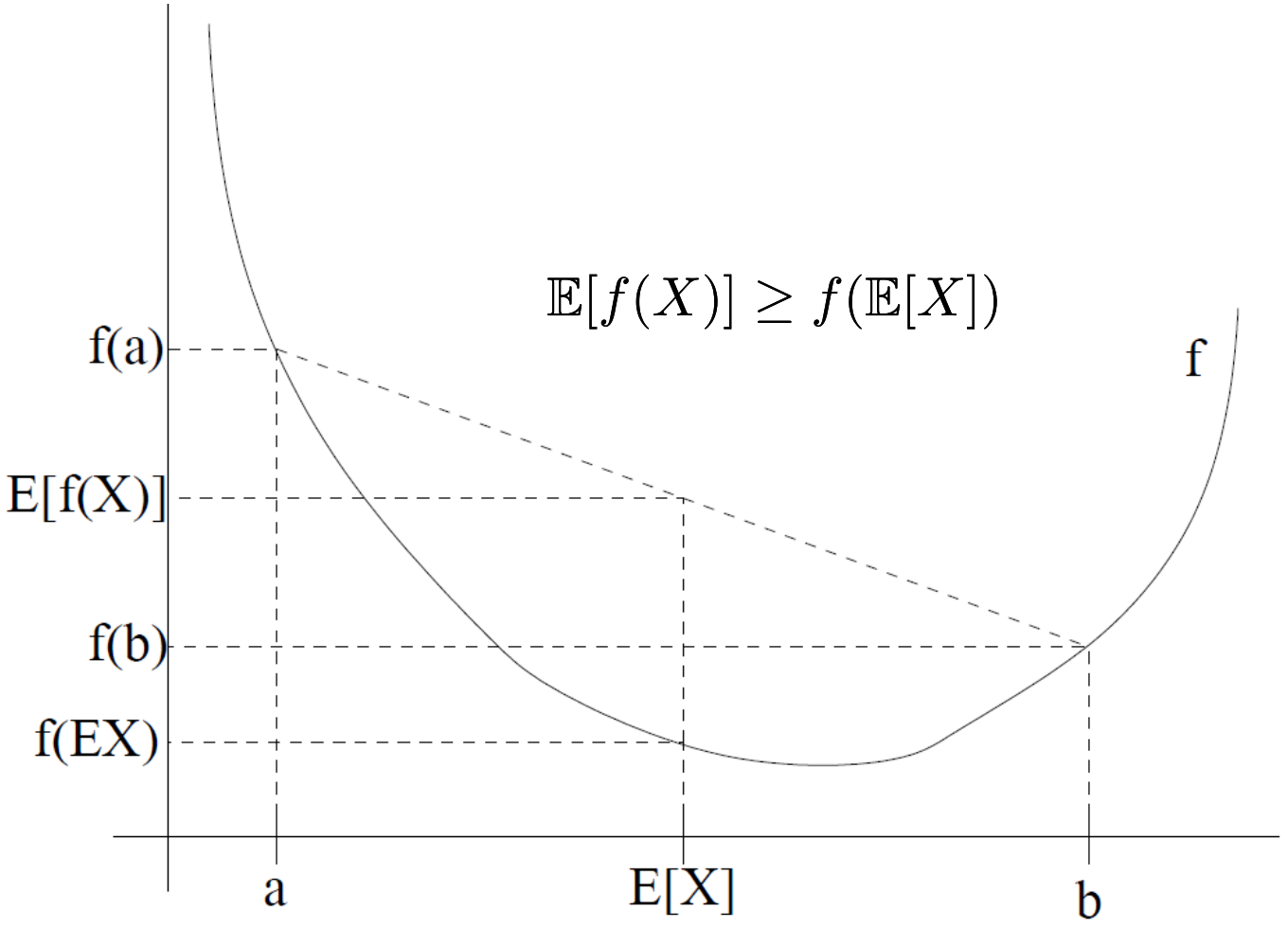


Figure credit: Andrew Ng

Jensen's Inequality

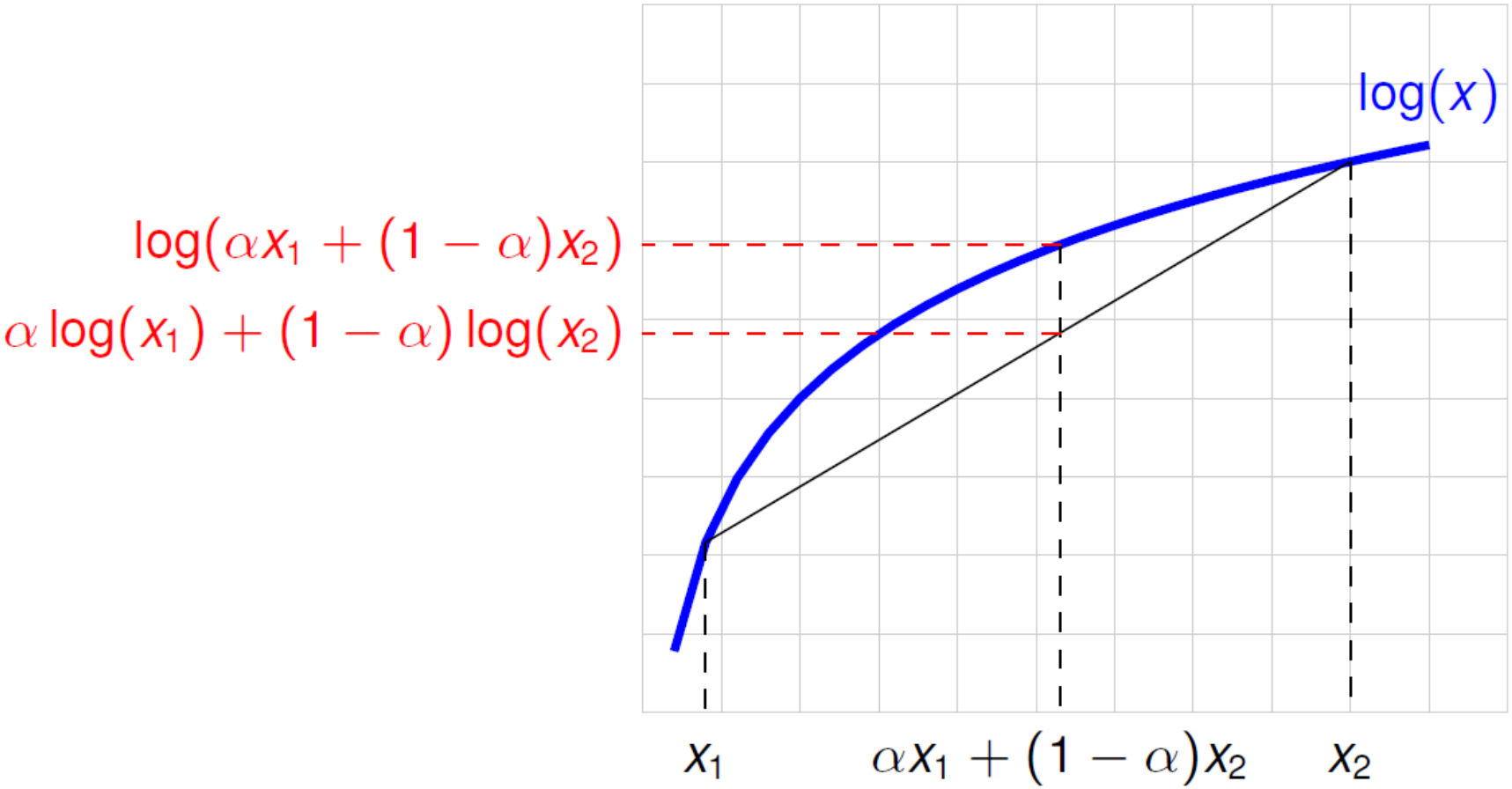


Figure credit: Maneesh Sahani

General EM Methods: Problem

- Given the training dataset

$$D = \{x_i\}_{i=1,2,\ldots,N}$$

  let the machine learn the data underlying patterns

- Assume latent variables

$$z \rightarrow x$$

- We wish to fit the parameters of a model *p*(*x*,*z*) to the data, where the log-likelihood is

$$l(\theta) = \sum_{i=1}^{N} \log p(x;\theta)$$

$$= \sum_{i=1}^{N} \log \sum_{z} p(x,z;\theta)$$

General EM Methods: Problems

- EM methods solve the problems where
  - Explicitly find the maximum likelihood estimation (MLE) is hard

$$\theta^* = \arg\max_\theta \sum_{i=1}^{N} \log \sum_z p(x^{(i)}, z^{(i)}; \theta)$$

  - But given $z^{(i)}$ observed, the MLE is easy

$$\theta^* = \arg\max_\theta \sum_{i=1}^{N} \log p(x^{(i)}|z^{(i)}; \theta)$$

- EM methods give an efficient solution for MLE, by iteratively doing
  - E-step: construct a (good) lower-bound of log-likelihood
  - M-step: optimize that lower-bound

General EM Methods: Lower Bound

- For each instance *i*, let q_i be some distribution of $z^{(i)}$

$$\sum_z q_i(z) = 1, \quad q_i(z) \geq 0$$

- Thus the data log-likelihood

$$\begin{aligned} l(\theta) = \sum_{i=1}^{N} \log p(x^{(i)}; \theta) &= \sum_{i=1}^{N} \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta) \\ &= \sum_{i=1}^{N} \log \sum_{z^{(i)}} q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)}; \theta)}{q_i(z^{(i)})} \\ &\geq \sum_{i=1}^{N} \sum_{z^{(i)}} q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{q_i(z^{(i)})} \end{aligned}$$

Lower bound of $l(\vartheta)$

Jensen's inequality
-log(*x*) is a convex function

General EM Methods: Lower Bound

$$l(\theta) = \sum_{i=1}^{N} \log p(x^{(i)}; \theta) \geq \sum_{i=1}^{N} \sum_{z^{(i)}} q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{q_i(z^{(i)})}$$

- Then what q_i(z) should we choose?

REVIEW

Jensen's Inequality

Theorem. Let *f* be a convex function, and let *X* be a random variable.

Then:

$$\mathbb{E}[f(X)] \geq f(\mathbb{E}[X])$$

- Moreover, if *f* is strictly convex, then

  $$\mathbb{E}[f(X)] = f(\mathbb{E}[X])$$

  holds true if and only if

  $$X = \mathbb{E}[X]$$

  with probability 1 (i.e., if *X* is a constant).

General EM Methods: Lower Bound

$$l(\theta) = \sum_{i=1}^{N} \log p(x^{(i)}; \theta) \geq \sum_{i=1}^{N} \sum_{z^{(i)}} q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{q_i(z^{(i)})}$$

- Then what q_i(z) should we choose?
- In order to make above inequality tight (to hold with equality), it is sufficient that

$$p(x^{(i)}, z^{(i)}; \theta) = q_i(z^{(i)}) \cdot c$$

- We can derive

$$\log p(x^{(i)}; \theta) = \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta) = \log \sum_{z^{(i)}} q(z^{(i)}) c = \sum_{z^{(i)}} q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{q_i(z^{(i)})}$$

- As such q_i(z) is written as the posterior distribution

$$q_i(z^{(i)}) = \frac{p(x^{(i)}, z^{(i)}; \theta)}{\sum_z p(x^{(i)}, z; \theta)} = \frac{p(x^{(i)}, z^{(i)}; \theta)}{p(x^{(i)}; \theta)} = p(z^{(i)} | x^{(i)}; \theta)$$

General EM Methods

Repeat until convergence: {

(E-step) For each *i*, set

$$q_i(z^{(i)}) = p(z^{(i)}|x^{(i)};\theta)$$

(M-step) Update the parameters

$$\theta = \arg\max_{\theta} \sum_{i=1}^{N} \sum_{z^{(i)}} q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)};\theta)}{q_i(z^{(i)})}$$

}

Convergence of EM

- Denote $\vartheta^{(t)}$ and $\vartheta^{(t+1)}$ as the parameters of two successive iterations of EM, we prove that

$$l(\theta^{(t)}) \leq l(\theta^{(t+1)})$$

  which shows EM always monotonically improves the log-likelihood, thus ensures EM will at least converge to a local optimum.

Proof of EM Convergence

- Start from $\vartheta^{(t)}$, we choose the posterior of latent variable

$$q_i^{(t)}(z^{(i)}) = p(z^{(i)}|x^{(i)};\theta^{(t)})$$

- This choice ensures the Jensen's inequality holds with equality

$$l(\theta^{(t)}) = \sum_{i=1}^{N} \log \sum_{z^{(i)}} q_i^{(t)}(z^{(i)}) \frac{p(x^{(i)}, z^{(i)}; \theta^{(t)})}{q_i^{(t)}(z^{(i)})} = \sum_{i=1}^{N} \sum_{z^{(i)}} q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t)})}{q_i^{(t)}(z^{(i)})}$$

- Then the parameters $\vartheta^{(t+1)}$ are then obtained by maximizing the right hand side of above equation
- Thus $l(\theta^{(t+1)}) \geq \sum_{i=1}^{N} \sum_{z^{(i)}} q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t+1)})}{q_i^{(t)}(z^{(i)})}$ [lower bound]

$$\geq \sum_{i=1}^{N} \sum_{z^{(i)}} q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t)})}{q_i^{(t)}(z^{(i)})}$$ [parameter optimization]

$$= l(\theta^{(t)})$$

Remark of EM Convergence

- If we define

$$J(q,\theta) = \sum_{i=1}^{N} \sum_{z^{(i)}} q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{q_i(z^{(i)})}$$

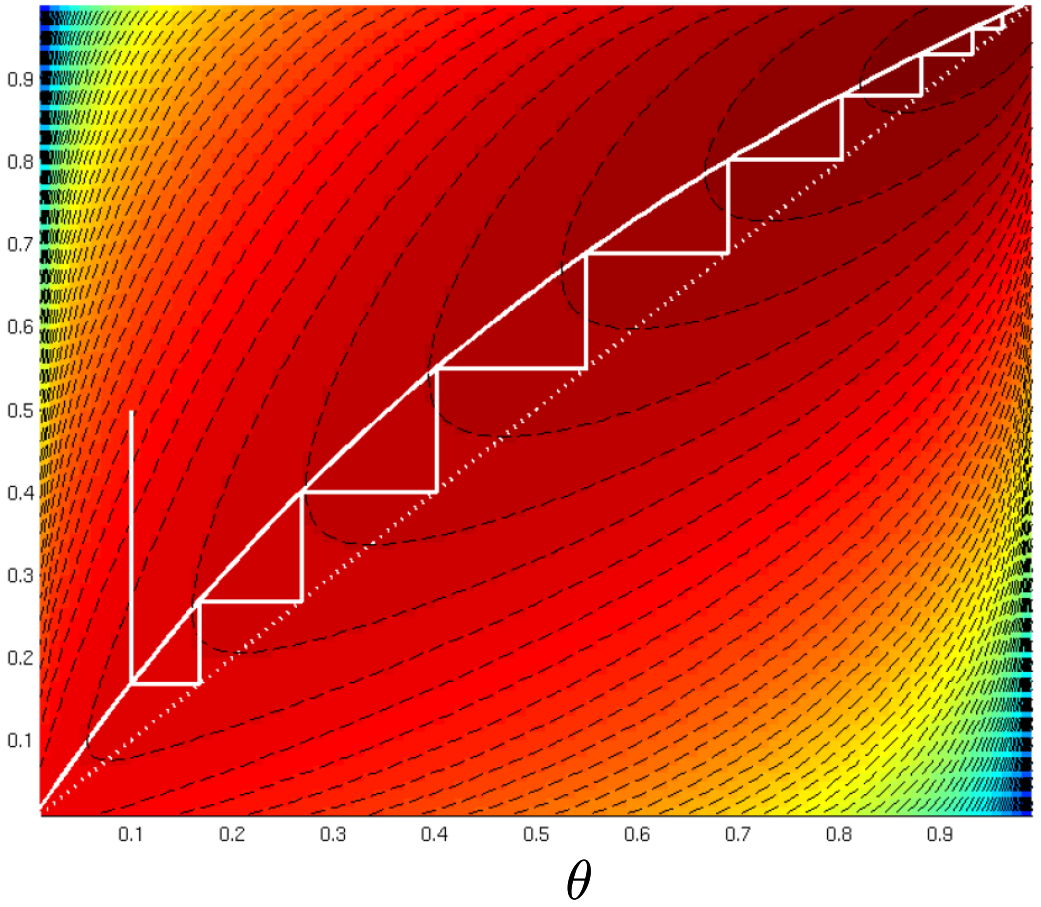
  Then we know

$$l(\theta) \geq J(q,\theta)$$

- EM can also be viewed as a coordinate ascent on *J*
  - E-step maximizes it w.r.t. *q*
  - M-step maximizes it w.r.t. ϑ

Coordinate Ascent in EM

$$q_i^{(t)}(z^{(i)}) = p(z^{(i)}|x^{(i)};\theta^{(t)})$$



$$\theta^{(t+1)} = \arg\max_{\theta} \sum_{i=1}^{N} \sum_{z^{(i)}} q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)};\theta)}{q_i^{(t)}(z^{(i)})}$$

Figure credit: Maneesh Sahani

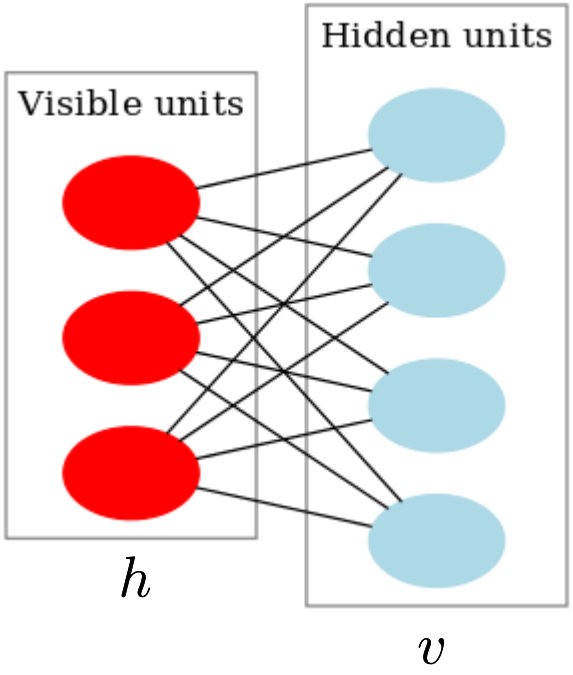
Content

- Fundamentals of Unsupervised Learning
  - K-means clustering
  - Principal component analysis
- Probabilistic Unsupervised Learning
  - Mixture Gaussians
  - EM Methods
- Deep Unsupervised Learning
  - Auto-encoders
  - Generative adversarial nets

Neural Nets for Unsupervised Learning

- Basic idea: use neural networks to recover the data

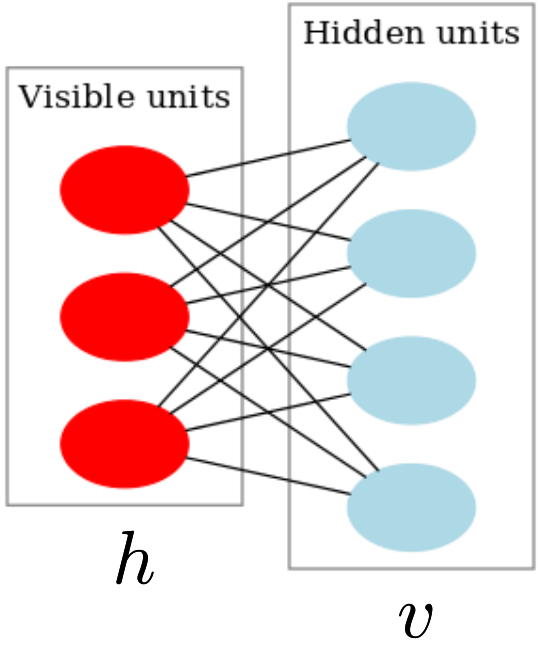
- Restricted Boltzmann Machine



Restricted Boltzmann Machine

- An RBM is an a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs

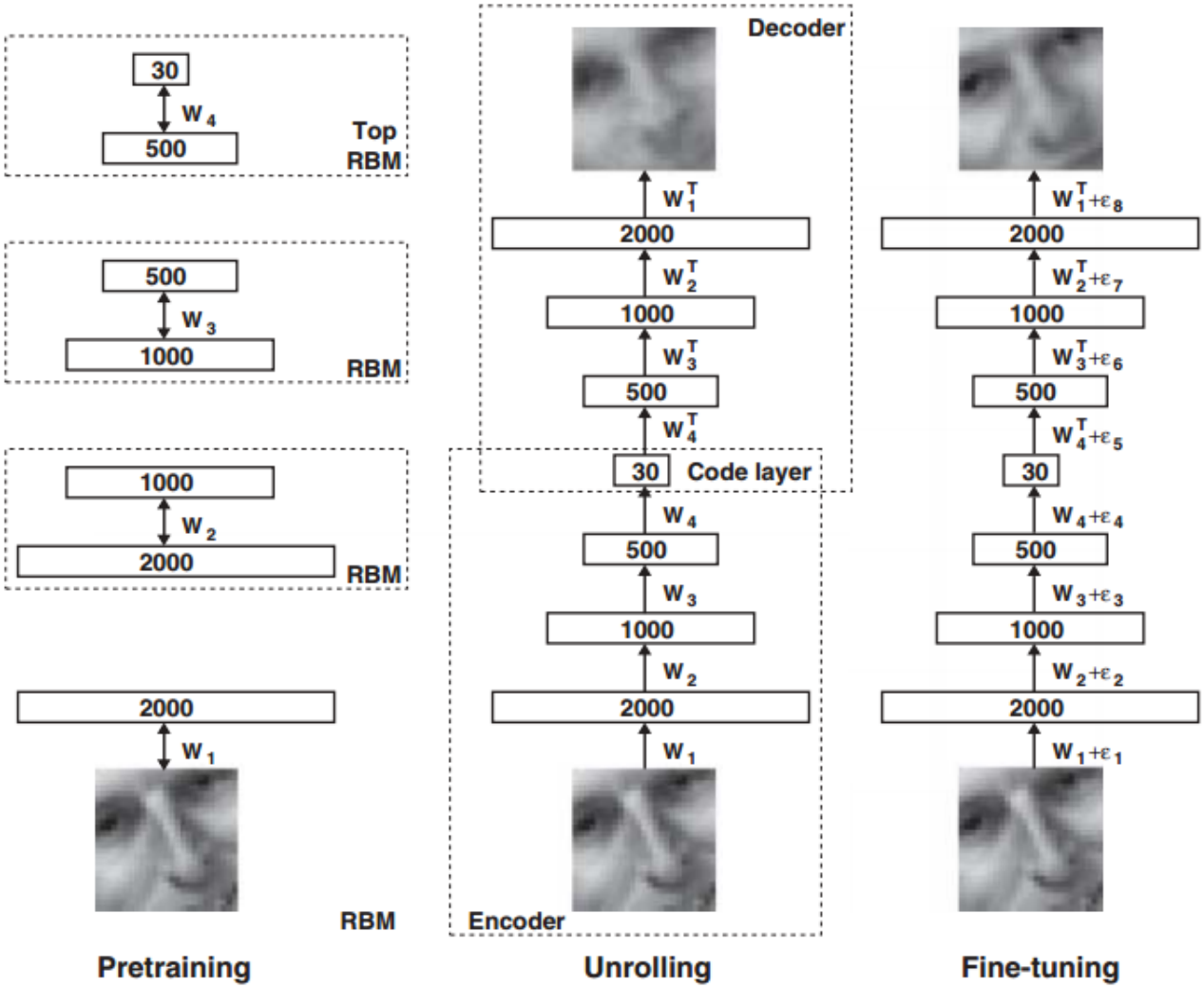
- Undirected graphical model
  - Restricted: Visible (hidden) units are not connected to each other
  - Energy function



$$E(v,h) = -\sum_i b_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i w_{i,j} h_j$$

$$p(v,h) = \frac{1}{Z} e^{-E(v,h)}$$

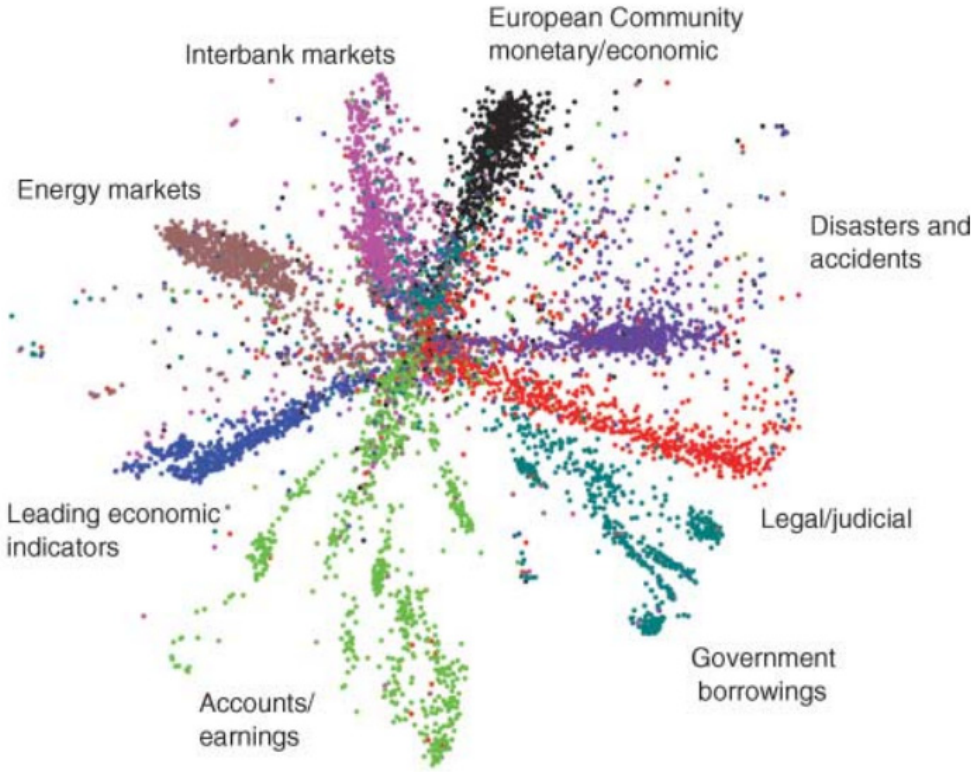
Deep Belief Networks



Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *science* 313.5786 (2006): 504-507.

Performance of Latent Factor Analysis

Latent semantic analysis
based on PCA



A 2000- 500-250-125-2 autoencoder
Trained by DBN

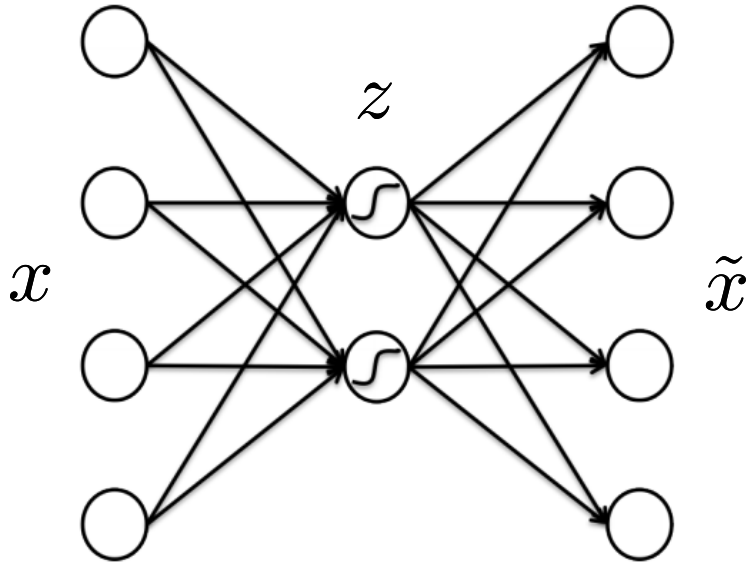
Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *science* 313.5786 (2006): 504-507.

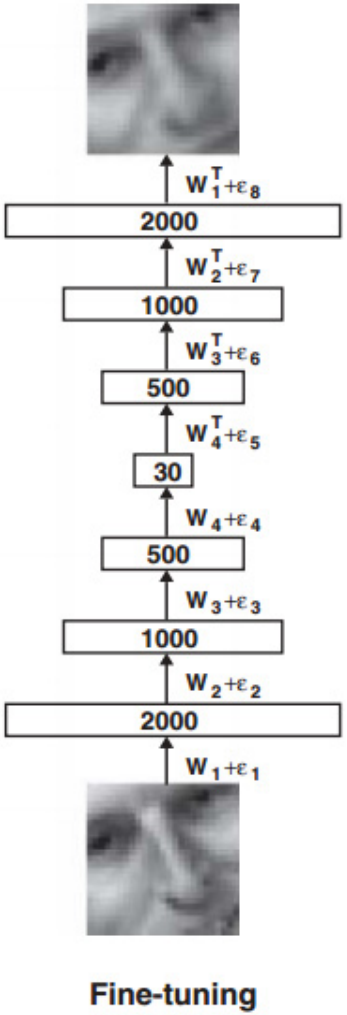
Auto-encoder

- An auto-encoder is an artificial neural net used for unsupervised learning of efficient codings
  - learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction

$$z = \sigma(W_1 x + b_1)$$
$$\tilde{x} = \sigma(W_2 z + b_2)$$

z is regarded as the low dimensional latent factor representation of *x*





Fine-tuning

Learning Auto-encoder

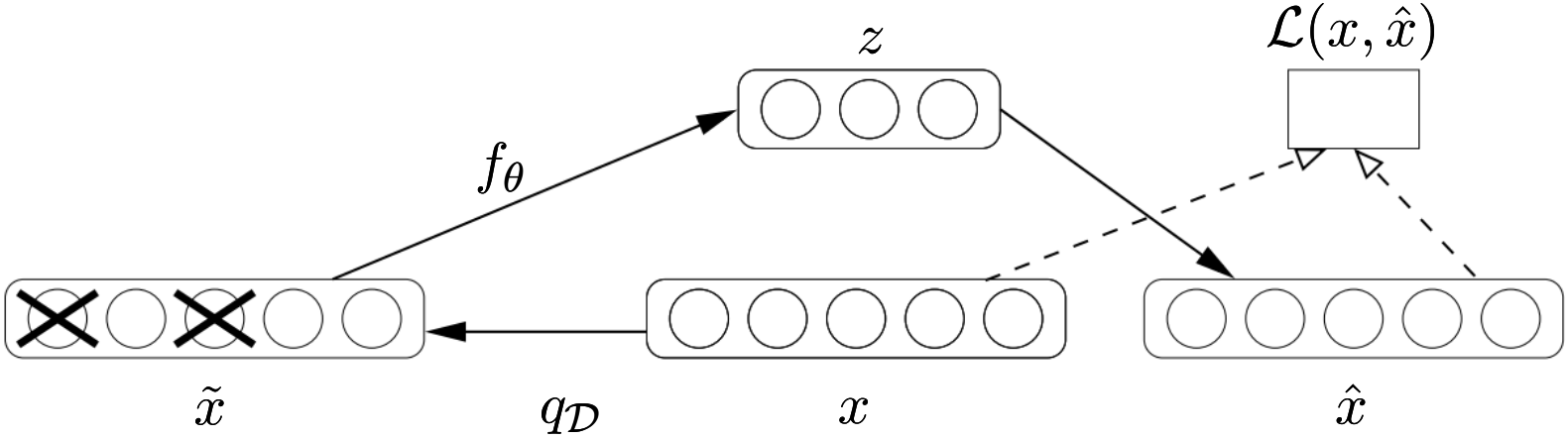
- Objective: squared difference between x and $\tilde{x}$

$$
\begin{aligned}
J(W_1, b_1, W_2, b_2) &= \sum_{i=1}^{m} (\tilde{x}^{(i)} - x^{(i)})^2 \\
&= \sum_{i=1}^{m} (W_2 z^{(i)} + b_2 - x^{(i)})^2 \\
&= \sum_{i=1}^{m} \left(W_2 \sigma(W_1 x^{(i)} + b_1) + b_2 - x^{(i)} \right)^2
\end{aligned}
$$

- Auto-encoder is an unsupervised learning model trained in a supervised fashion

$$\theta \leftarrow \theta - \eta \frac{\partial J}{\partial \theta}$$

Denoising Auto-encoder



- Clean input x is partially destroyed, yielding corrupted input

$$\tilde{x} \sim q_\mathcal{D}(\tilde{x}|x)$$ e.g. Gaussian noise

- The corrupted input $\tilde{x}$ is mapped to hidden representation

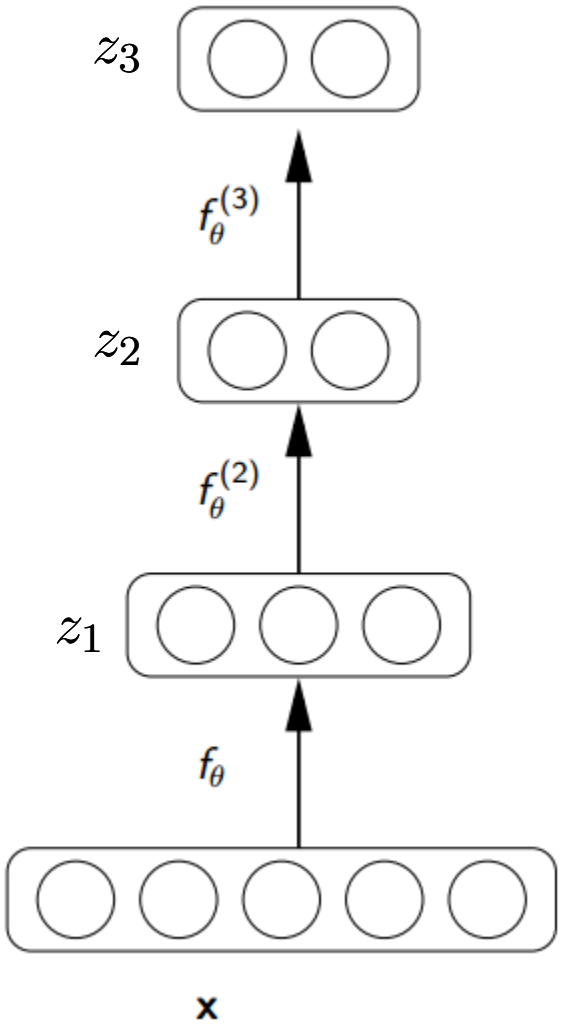
$$z = f_\theta(\tilde{x})$$

- From *z* reconstruct the data

$$\hat{x} = g_{\theta'}(z)$$

Stacked Auto-encoder

- Layer-by-layer training
  1. Train the first layer to use z_1 to reconstruct x
  2. Train the second layer to use z_2 to reconstruct z_1
  3. Train the third layer to use z_3 to reconstruct z_2



Some Denoising AE Examples

Original Corrupted Reconstructed

Generative Adversarial Networks (GANs)

[Goodfellow, I., et al. 2014. Generative adversarial nets. In NIPS 2014.]

Problem Definition

- Given a dataset $D = \{x\}$, build a model $q_\theta(x)$ of the data distribution that fits the true one $p(x)$

- Traditional objective: maximum likelihood estimation (MLE)

$$\max_\theta \frac{1}{|D|} \sum_{x \in D} [\log q_\theta(x)] \simeq \max_\theta \mathbb{E}_{x \sim p(x)} [\log q_\theta(x)]$$

  - Check whether a true data is with a high mass density of the learned model

Inconsistency of Evaluation and Use

- Given a generator *q* with a certain generalization ability

$$\max_{\theta} \mathbb{E}_{x \sim p(x)} [\log q_\theta(x)]$$

Training/evaluation

- Check whether a true data is with a high mass density of the learned model
- Approximated by

$$\max_{\theta} \frac{1}{|D|} \sum_{x \in D} [\log q_\theta(x)]$$

$$\max_{\theta} \mathbb{E}_{x \sim q_\theta(x)} [\log p(x)]$$

Use

- Check whether a model-generated data is considered as true as possible
- More straightforward but it is hard or impossible to directly calculate $p(x)$

Generative Adversarial Nets (GANs)

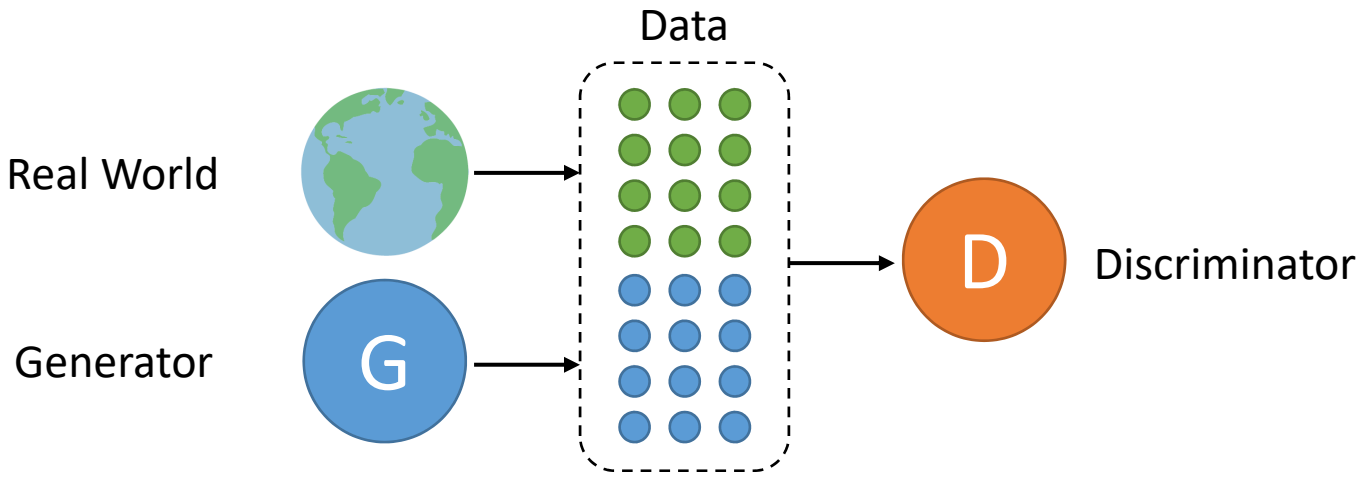
- What we really want

$$\max_{\theta} \mathbb{E}_{x \sim q_\theta(x)} [\log p(x)]$$

- But we cannot directly calculate $p(x)$

- Idea: what if we build a discriminator to judge whether a data instance is true or fake (artificially generated)?
  - Leverage the strong power of deep learning based discriminative models

Generative Adversarial Nets (GANs)

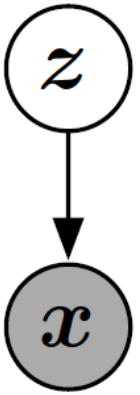


- Discriminator tries to correctly distinguish the true data and the fake model-generated data
- Generator tries to generate high-quality data to fool discriminator
- G & D can be implemented via neural networks
- Ideally, when D cannot distinguish the true and generated data, G nicely fits the true underlying data distribution

Generator Network

$$x = G(z; \theta^{(G)})$$

- Must be differentiable
- No invertibility requirement
- Trainable for any size of *z*
- Can make *x* conditionally Gaussian given *z* but need not do so
  - e.g. Variational Auto-Encoder
- Popular implementation: multi-layer perceptron

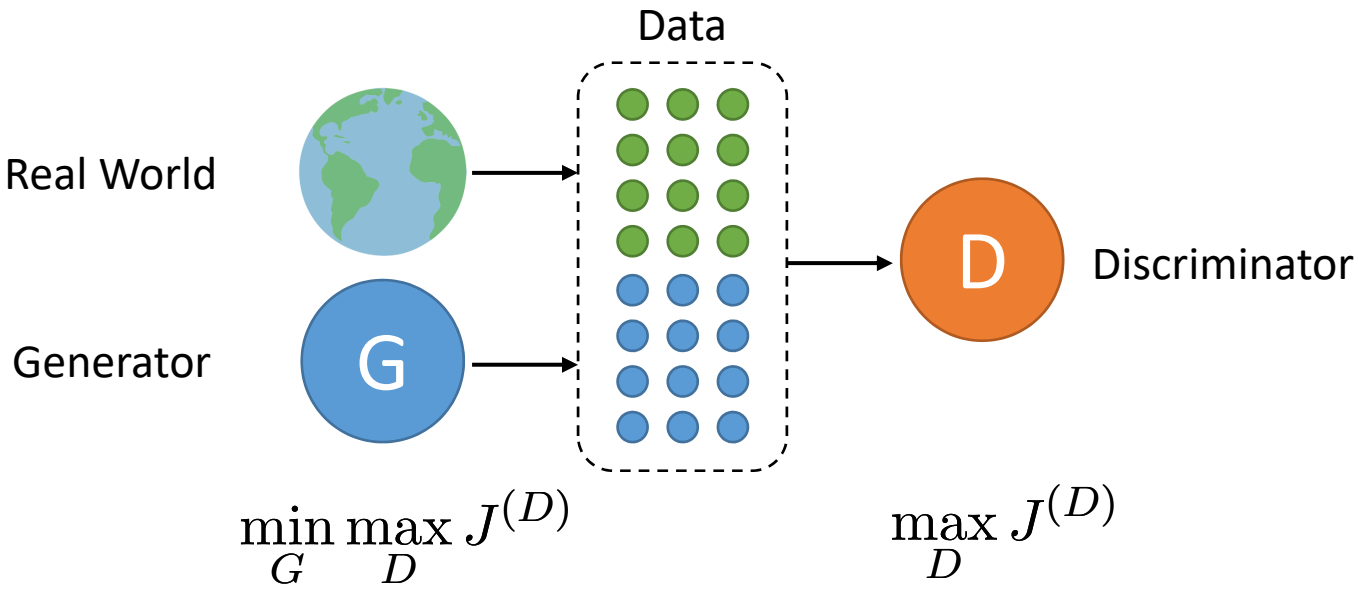


Discriminator Network

$$P(\text{true}|\boldsymbol{x}) = D(\boldsymbol{x}; \boldsymbol{\theta}^{(D)})$$

- Can be implemented by any neural networks with a probabilistic prediction
- For example
  - Multi-layer perceptron with logistic output
  - AlexNet etc.

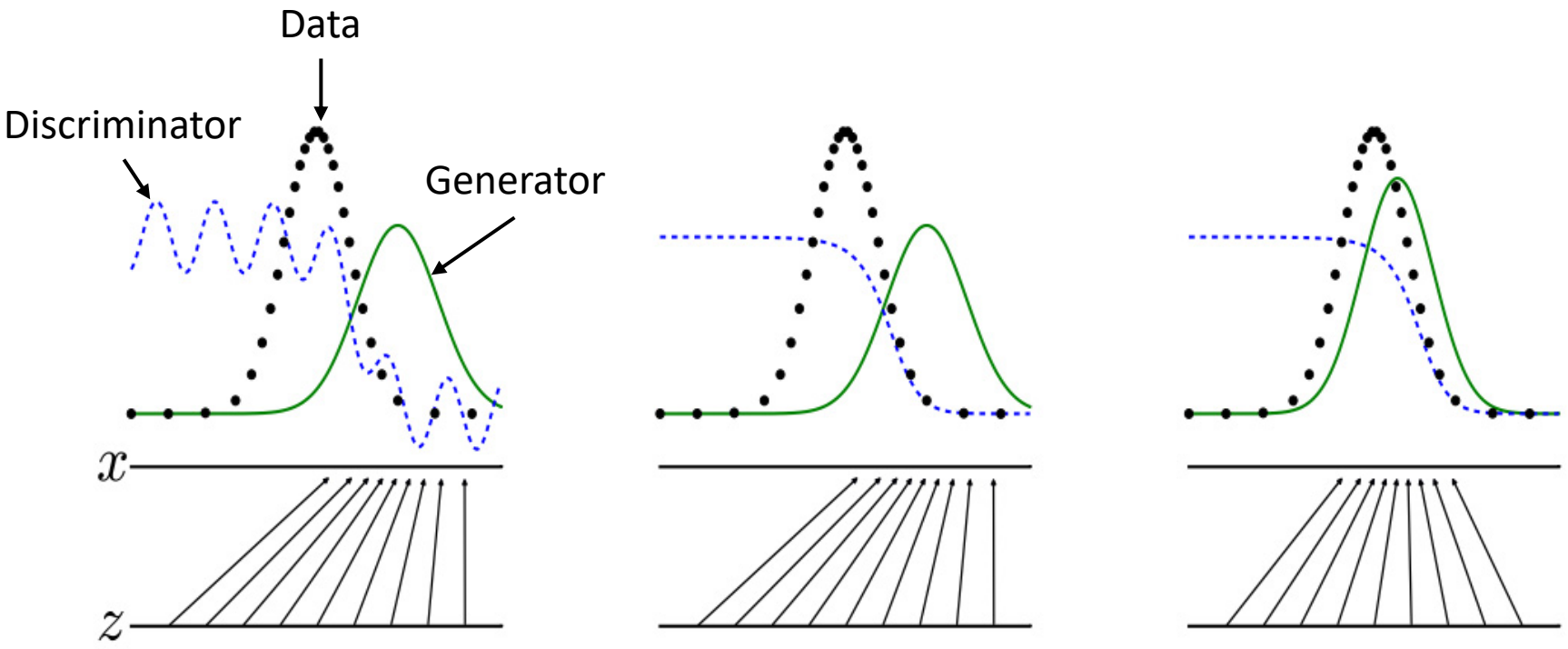
GAN: A Minimax Game



$$J^{(D)} = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Generator $\min_G \max_D J^{(D)}$ Discriminator $\max_D J^{(D)}$

Illustration of GANs

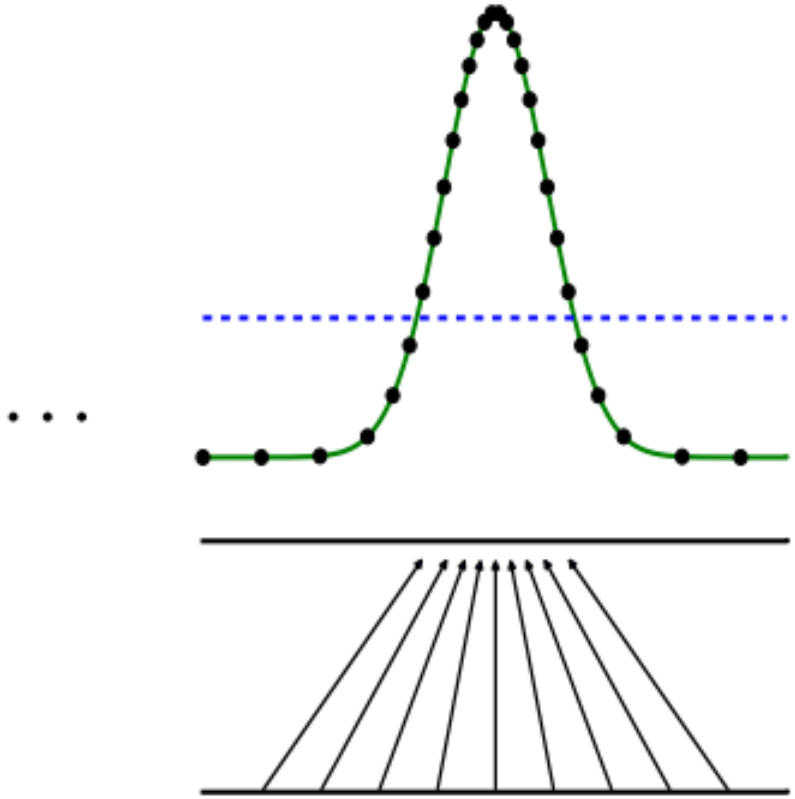


Generator $\min_G \max_D J^{(D)}$ Discriminator $\max_D J^{(D)}$

$$J^{(D)} = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Ideal Final Equilibrium

- Generator generates perfect data distribution
- Discriminator cannot distinguish the true and generated data



Training GANs

Training discriminator

for number of training iterations **do**

  for k steps **do**

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

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

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

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Training GANs

for number of training iterations **do**

  for k steps **do**

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

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

Training generator

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

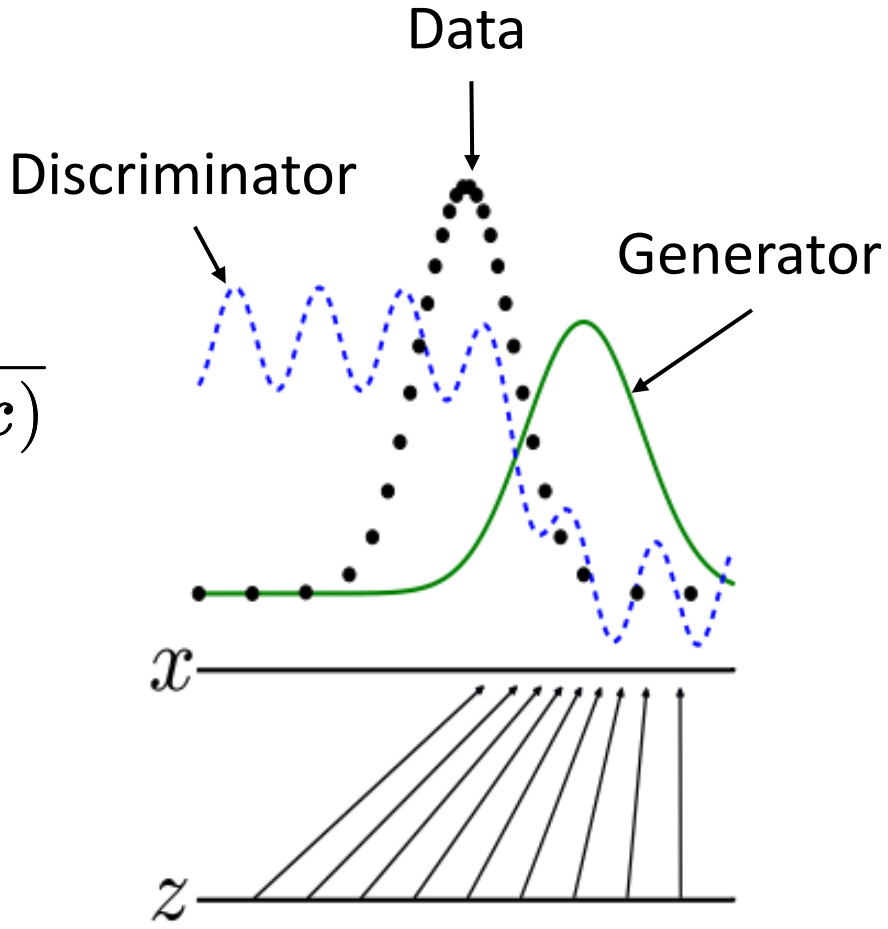
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Optimal Strategy for Discriminator

- Optimal $D(\boldsymbol{x})$ for any $p_{\text{data}}(\boldsymbol{x})$ and $p_G(\boldsymbol{x})$ is always

$$D(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}$$



Reformulate the Minimax Game

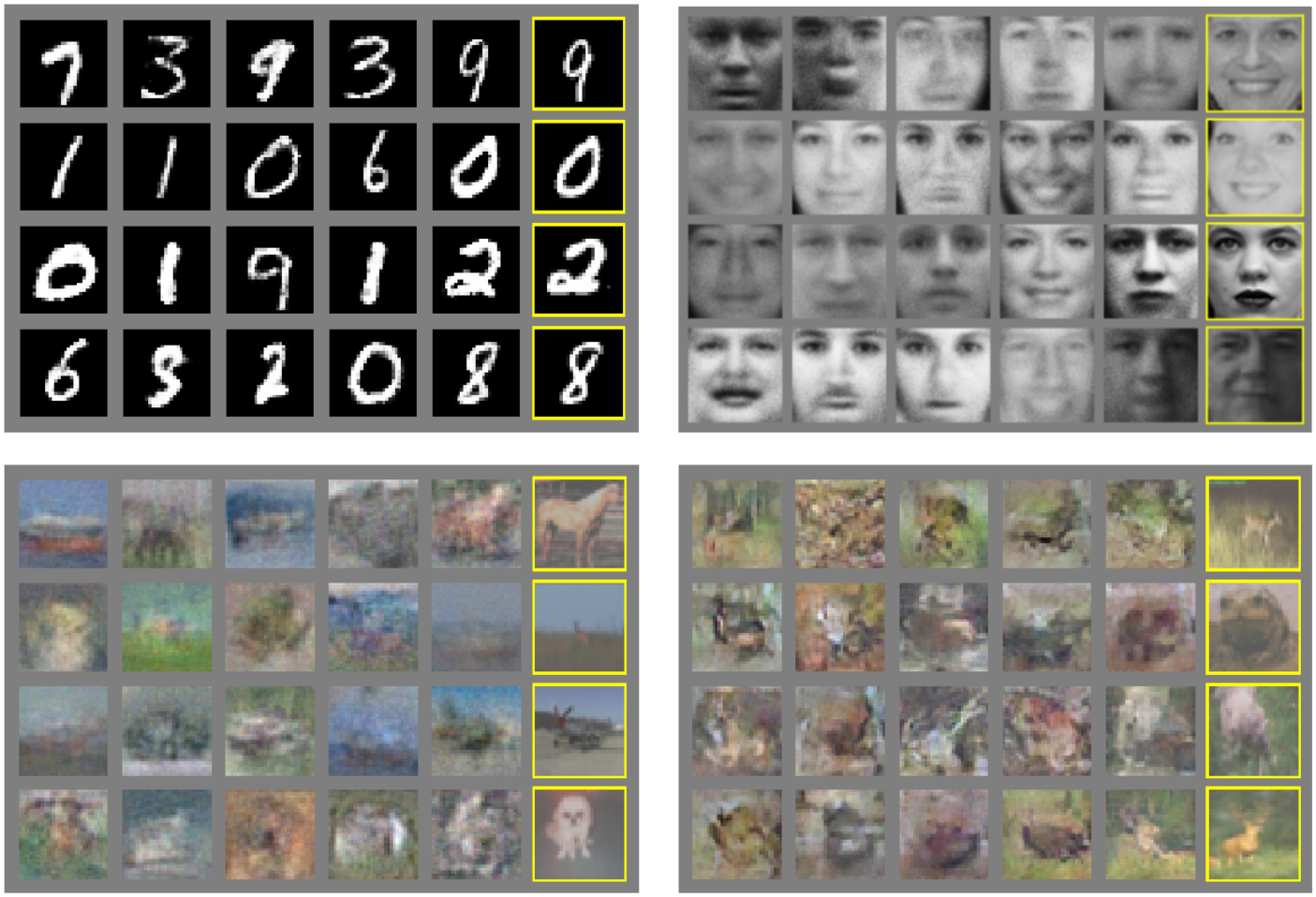
G: $\min_G \max_D J^{(D)}$ D: $\max_D J^{(D)}$

$$
\begin{aligned}
J^{(D)} &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_G(\boldsymbol{x})}[\log(1 - D(\boldsymbol{x}))] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right] \\
&\quad + \mathbb{E}_{\boldsymbol{x}\sim p_G(\boldsymbol{x})}\left[\log \frac{p_G(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right] \\
&= -\log(4) + \mathrm{KL}\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_G}{2}\right) + \mathrm{KL}\left(p_G \middle\| \frac{p_{\text{data}} + p_G}{2}\right)
\end{aligned}
$$

$\min_G J^{(D)}$ is something between $\max_G \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log p_G(\boldsymbol{x})]$ and $\max_G \mathbb{E}_{\boldsymbol{x}\sim p_G}[\log p_{\text{data}}(\boldsymbol{x})]$

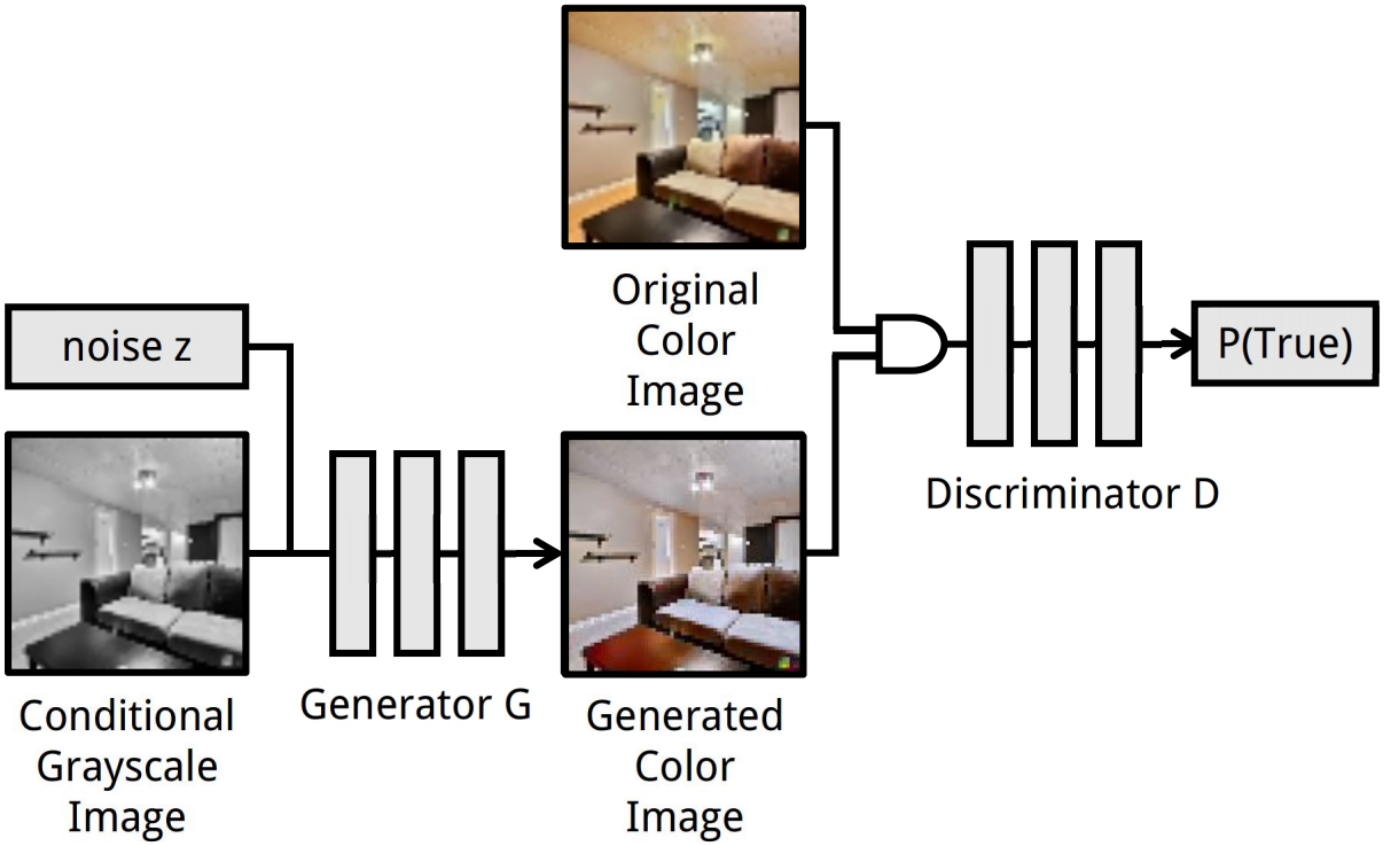
[Huszár, Ferenc. "How (not) to Train your Generative Model: Scheduled Sampling, Likelihood, Adversary?." *arXiv* (2015).]

Case Study of GANs



- The rightmost images in each row is the closest training data images to the neighbor generated ones, which means GAN does not simply memorize training instances

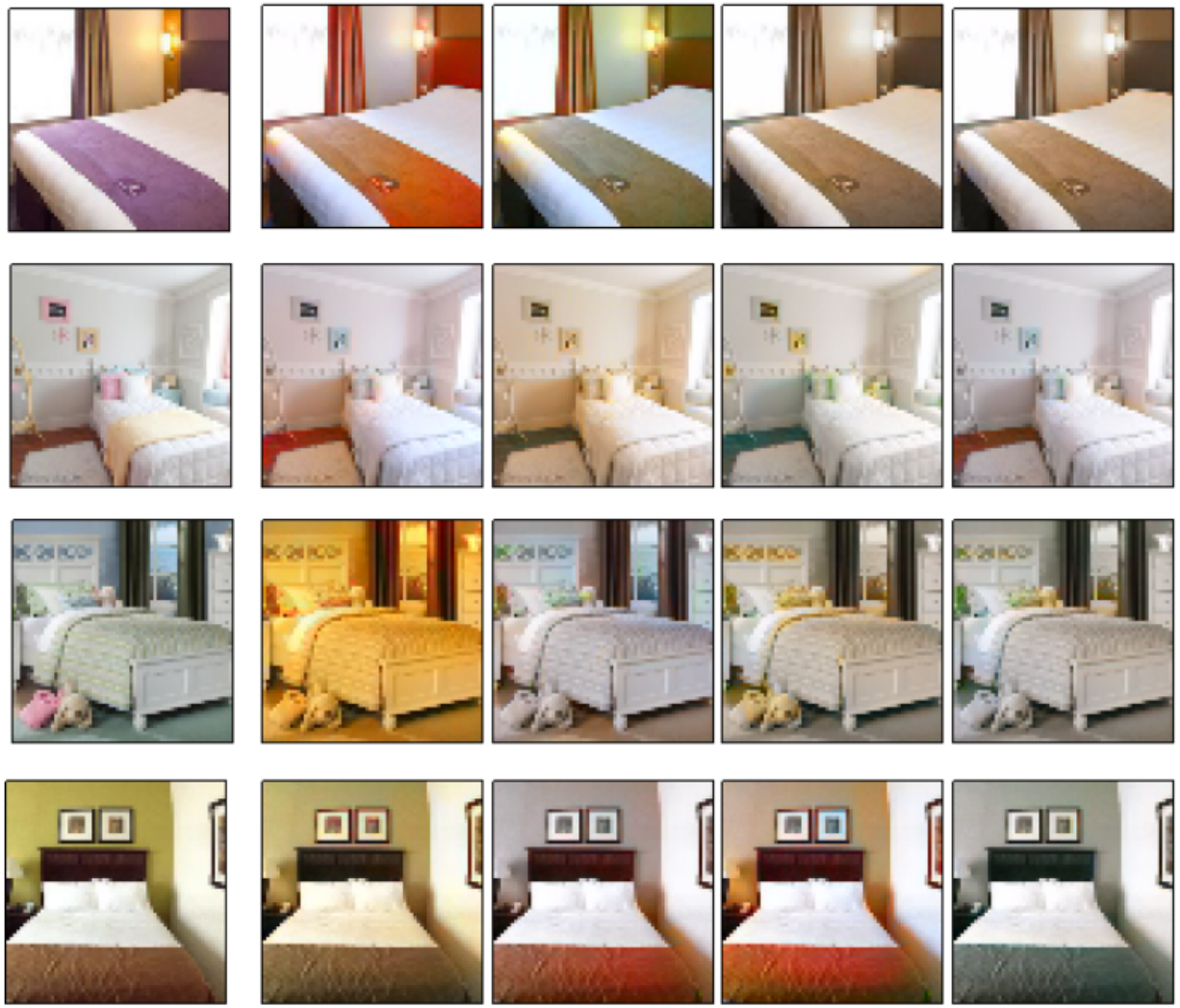
Application: GAN for Image Colorization



- Conditional GAN
  - Input: a grayscale image; output: a naturally colored one

Y Cao, Z Zhou, W Zhang, Y Yu. Unsupervised Diverse Colorization via Generative Adversarial Networks. arXiv:1702.06674, 2017

Examples of GAN for Colorization



Y Cao, Z Zhou, W Zhang, Y Yu. Unsupervised Diverse Colorization via Generative Adversarial Networks. arXiv:1702.06674, 2017