Unsupervised Learning

Wenhu Chen
Lecture 21

Outline

Learning Goals

- Understanding what is unsupervised Learning
- Understanding K-Means clustering algorithm
- Knowing how to perform PCA
- Understanding the basic idea of Auto-Encoder and GAN

Unsupervised Learning Tasks

2 major types of tasks:

- *Representation learning*: learning low-dimensional representations of examples
- *Generative modelling*: learning probability distribution from which new examples can be drawn as samples

Unsupervised Learning - Clustering

Clustering is a common unsupervised representation learning task

→ Goal is to group training examples into *clusters*.

→ Clusters can be thought of as classes/categories.

Unsupervised Learning - Clustering

2 types of clustering tasks

- *Hard clustering*: each example is assigned to 1 cluster with certainty
  $\rightarrow \texttt{class}(x) = c$
- *Soft clustering*: each example has a probability distribution over all clusters
  $\rightarrow \texttt{class}(x) \sim P(C|x)$

k-Means Clustering - Overview

- A hard clustering algorithm
- Learns to definitively assign examples to classes
- Input: number of clusters k, training examples X
- Goal is to learn a representation that assigns examples to the appropriate class $c \in \{1, 2, \ldots, k\}$

k-Means Clustering - Centroids

Suppose each example contains n features: $x = \langle x_1, x_2, \ldots, x_n \rangle$

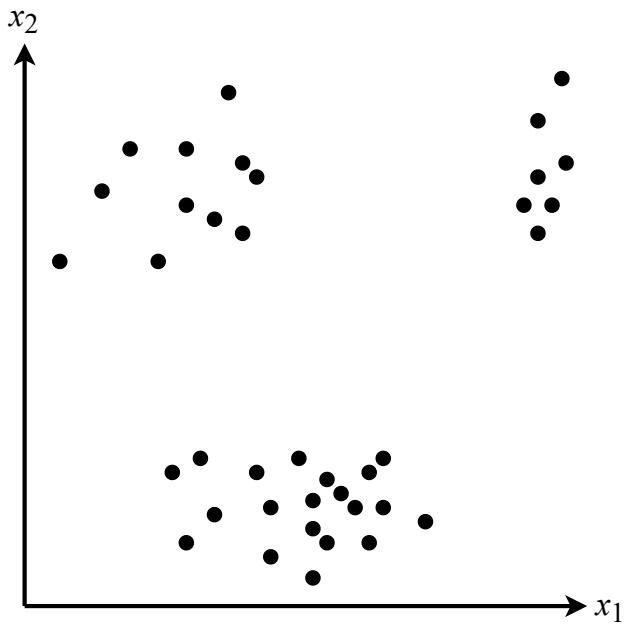
Each feature x_j is real-valued.

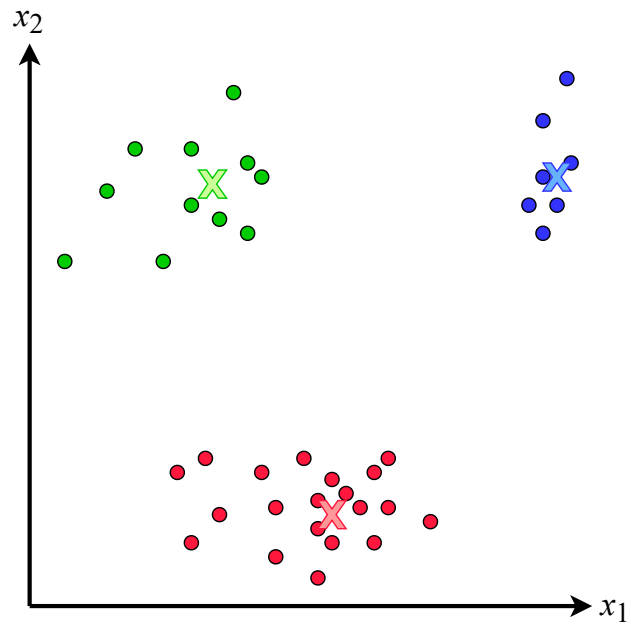
k-Means learns a *centroid* for each cluster and assigns examples to the closest centroid

- By "closest" we mean the centroid that is the shortest distance from x
- Need to define a distance function $d(c, x)$
  $\rightarrow$ E.g. Euclidean distance (L2): $d(c, x) = \sqrt{\sum_{j=1}^{n} (c_j - x_j)^2}$

k-Means Clustering - Centroids

Example: $k = 3$, $x = \langle x_1, x_2 \rangle$





k-Means Clustering - Algorithm Overview

k-means alternates between 2 steps:

1. *Centroid update:* Set the centroid of each cluster as the feature-wise mean of each example currently assigned to the cluster.
2. *Cluster assignment:* Assign each training example x to the cluster with the closest centroid.

k-Means Clustering - Algorithm

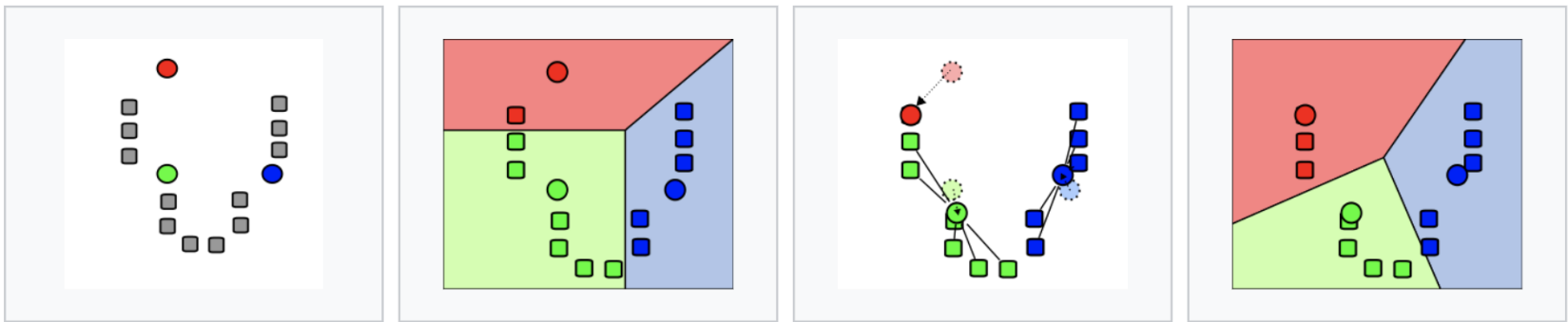
Input: $X \in \mathbb{R}^{m \times n}, k \in \mathbb{N}, d(c, x)$

1. Initialization:
   Randomly initialize k centroids: $C \in \mathbb{R}^{k \times n}$
2. While not converged, do:
   - Assign each example to the cluster whose centroid is closest.
     $Y[i] \leftarrow \arg\min_c d(C[c], X[i])$
   - Calculate the centroid for each cluster c by calculating the average feature value for each exmaple currently classified as cluster c.
     $C[c] \leftarrow \frac{1}{n_c} \sum_{j=1}^{n_c} X_c[j]$

Visualization of Clustering Algorithm

The clustering algorithm visualization:

Demonstration of the standard algorithm



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

k-Means Clustering - Example Iteration

Let's perform 1 iteration of k-means with $k = 2$, using Euclidean distance. Use the following dataset:

Example	x_1,	x_2	x_3
1	0.2	0.5	0
2	-0.6	2.1	1.2
3	-0.5	1.9	1.3
4	0.1	0.5	-0.3

Assume the current values for the centroids are as follows:

c	c_1,	c_2	c_3
1	0.3	0.8	-0.5
2	-0.1	-0.5	1.0

k-Means Clustering - Example Iteration

Let's perform 1 iteration of k-means with $k = 2$, using Euclidean distance. Use the following dataset:

Example	x_1,	x_2	x_3
1	0.2	0.5	0
2	-0.6	2.1	1.2
3	-0.5	1.9	1.3
4	0.1	0.5	-0.3

$c1 = [0.3, 0.8, -0.5]$, $c2 = [-0.1, -0.5, 1.0]$

Example 1 to $c1$: $0.1^2 + 0.3^2 + 0.5^2$, to $c2$: $0.3^2 + 1.0^2 + 1.0^2$: $c1$
Example 2 to $c1$: $0.9^2 + 1.3^2 + 1.7^2$, to $c2$: $0.4^2 + 2.6^2 + 0.2^2$: $c1$
Example 3 to $c1$: $0.8^2 + 1.1^2 + 1.8^2$, to $c2$: $0.4^2 + 2.4^2 + 0.3^2$: $c1$
Example 4 to $c1$: $0.2^2 + 0.3^2 + 0.2^2$, to $c2$: $0.2^2 + 1.0^2 + 1.3^2$: $c1$

k-Means Clustering - Example Iteration

Let's perform 1 iteration of k-means with $k = 2$, using Euclidean distance. Use the following dataset:

Example	x_1,	x_2	x_3
1	0.2	0.5	0
2	-0.6	2.1	1.2
3	-0.5	1.9	1.3
4	0.1	0.5	-0.3

Computing the new centroid:

$c1 = [0.2, 1.25, 0.55]$ $c2 = []$

$\rightarrow$ ou need to re-initialize the centroid.

k-Means Clustering - Finding the Best Solution

- k-means is guaranteed to converge (with L2 distance)
- Solution not guaranteed to be optimal
- To increase chance of finding better solution, you could:
  - Run multiple times with different random initial cluster assignments
  - Scale the features so that their domains are similar

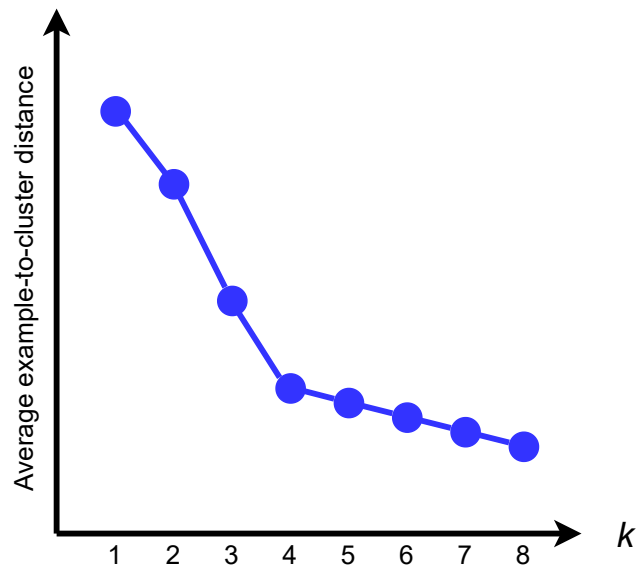
k-Means Clustering - Choosing proper k

The choice of k greatly determines the outcome of the clustering.

- As long as there are $\leq k+1$ examples, running k-means with $k+1$ clusters will result in lower error than running with k clusters
- But using too large k will defeat the point of representation learning...

k-Means Clustering - The Elbow Method

1. Execute k-means with multiple values of $k \in \{1, 2, \ldots, k_{max}\}$.
2. Plot average distance across all examples and assigned clusters.
3. Select k where there is drastic reduction in error improvement on the plot (i.e. "elbow point")



$\rightarrow$ Can be ambiguous, since it is manual

k-Means Clustering - Silhouette Analysis

1. Execute k-means with multiple values of $k \in \{1, 2, \ldots, k_{max}\}$.
2. Calculate average silhouette score $s(x)$ for each k across the dataset
3. Select k that maximizes average $s(x)$

$$s(x) = \begin{cases} \frac{b(x)-a(x)}{\max(a(x),b(x))} & \text{if } |C_x| > 1 \\ 0 & \text{if } |C_x| = 1 \end{cases}$$

- $a(x)$ is the average distance from example x to all other examples in its own cluster
- $b(x)$ is the smallest of the average distance of x to examples in any other cluster

$\rightarrow$ Significantly more objective than the Elbow Method

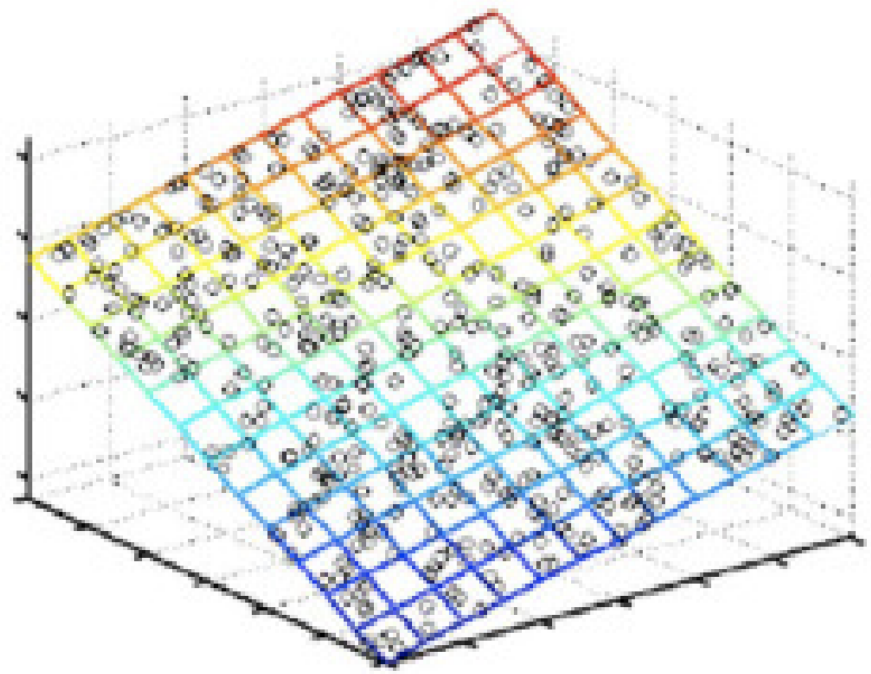
Dimension Reduction

Dimensionality reduction simply refers to the process of reducing the number of attributes in a dataset while keeping as much of the variation in the original dataset as possible.

- High Dimension Data actually resides in an inherent low-dimensional space.
- Additional dimensions are just random noise.
- Goal is to recover these inherent dimension and discard noise dimension.

Dimension Reduction

The observed data point dimensionality is not necessarily the intrinsic dimension of the data.

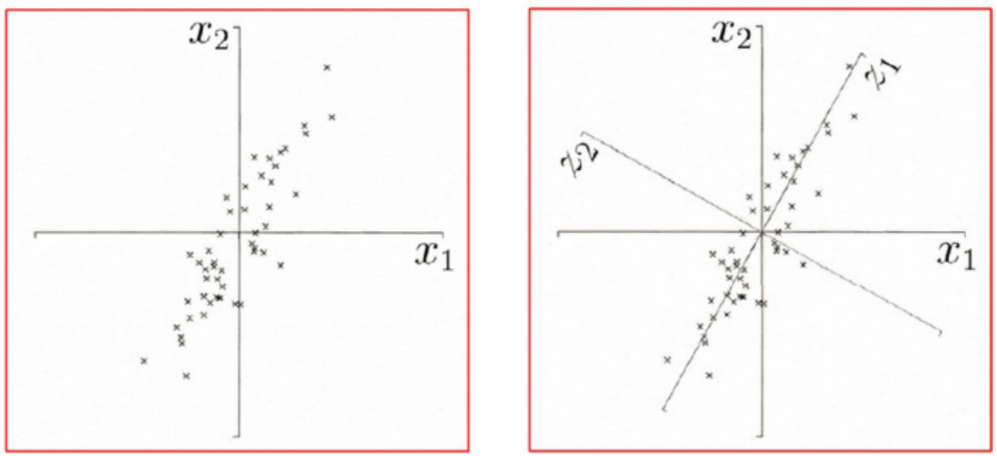


$$3d \Rightarrow 2d$$

By finding the intrinsic dimension, the problem becomes simpler.

Principal Component Analysis

- Widely used method for unsupervised dimensionality reduction
- account for variance of data in as few dimensions as possible
- First PC is the project of direction that maximizes the variance of projected data
- Second PC is the project of direction that is orthogonal to the first PC that maximizes the variance of projected data



PCA Terminology

Consider a set of random variables of dimension p. We observe n data points of $x_i \in \mathbb{R}^p$, where each of the n rows represents a different repetition of the experiment, and each of the p columns gives a particular kind of feature.

Therefore, we can write the entire dataset as $X \in \mathbb{R}^{n \times p}$

PCA aims to project the dataset to a lower dimension $p' < p$, which can be written as:

$$T = XW \tag{1}$$

where $W \in \mathbb{R}^{p \times p'}$ projection matrix.

You can construct the original input by:

$$X' = TW^T = XWW^T \tag{2}$$

Principal Component Analysis (Algorithm I)

- Given the centered data, compute the principle vectors:

$$w_1 = argmax_{|w|=1} \frac{1}{m} \sum_{i=1}^{m} \{w^T x_i\}^2 \tag{3}$$

Principal Component Analysis (Algorithm I)

- Given the centered data, compute the principle vectors:

$$w_1 = argmax_{|w|=1} \frac{1}{m} \sum_{i=1}^{m} \{w^T x_i\}^2 \tag{3}$$

- We maximize the variance of the projection of x

$$w_k = argmax_{|w|=1} \frac{1}{m} \sum_{i=1}^{m} \{w^T [x_i - \sum_{j=1}^{k-1} w_j w_j^T x_i]\}^2 \tag{4}$$

Principal Component Analysis (Algorithm I)

- Given the centered data, compute the principle vectors:

$$w_1 = argmax_{|w|=1} \frac{1}{m} \sum_{i=1}^{m} \{w^T x_i\}^2 \tag{3}$$

- We maximize the variance of the projection of x

$$w_k = argmax_{|w|=1} \frac{1}{m} \sum_{i=1}^{m} \{w^T [x_i - \sum_{j=1}^{k-1} w_j w_j^T x_i]\}^2 \tag{4}$$

- We maximize the variance of the projection in the residual subspace

Principal Component Analysis (Algorithm II)

- Mean center the data.
- Compute Covariance Matrix $\Sigma = X^T X$.
- Calculate the eigen values and eigen vectors of Σ.
  - Eigenvector with largest eigen value λ_1 is the first PC.
  - Eigenvector with k_{th} largest eigenvaluve λ_k is the k-th PC.
  - $\lambda_k / \sum_k \lambda_k$ is the proportion of variance captured by k-th PC.

Principal Component Analysis (Algorithm II)

The goal of PCA can be written as follows:

$$\text{Max} \quad u^T X^T X u \tag{5}$$

$$\text{s.t} \quad u^T u = 1 \tag{6}$$

Construct the Lagrangian multiplier to maximize $u^T X^T X u - \lambda u^T u$

The partial derivatives $\frac{\partial u^T X^T X u - \lambda u^T u}{\partial u}$ is set to zero:

$$2X^T X u - 2\lambda u = 2(X^T X - \lambda I)u = 0 \tag{7}$$

Therefore, u must be an eigenvalue of $X^T X$ with eigenvalue of λ.

Principal Component Analysis (Algorithm II)

- For symmetric matrices, eigenvectors for distinct eigenvalues are orthogonal.

$$X^T X u = \lambda u, \quad \text{and} \quad \lambda_1 \neq \lambda_2 \rightarrow u_1 \cdot u_2 = 0 \tag{8}$$

- All eigenvalues of a real symmetric matrix are real.

$$|X^T X - \lambda I| = 0 \quad \text{and} \quad \lambda \in \mathcal{R} \tag{9}$$

- All eigenvalues of a positive semidefinite matrix are non-negative

$$w^T X^T X w \geq 0, \quad \text{then} \quad \lambda > 0 \tag{10}$$

Autoencoders - Overview

- A representation learning algorithm
- Learn to map examples to low-dimensional representation

Autoencoders - Components

2 main components

1. *Encoder* $e(x)$: maps x to low-dimensional representation $\hat{z}$
2. *Decoder* $d(\hat{z})$: maps $\hat{z}$ to its original representation x

Autoencoder implements $\hat{x} = d(e(x))$

- $\hat{x}$ is the *reconstruction* of original input x
- Encoder and decoder learned such that $\hat{z}$ contains as much information about x as needed to reconstruct it

Minimize sum of squares of differences between input and prediction:

$$E = \sum_i (x_i - d(e(x_i)))^2$$

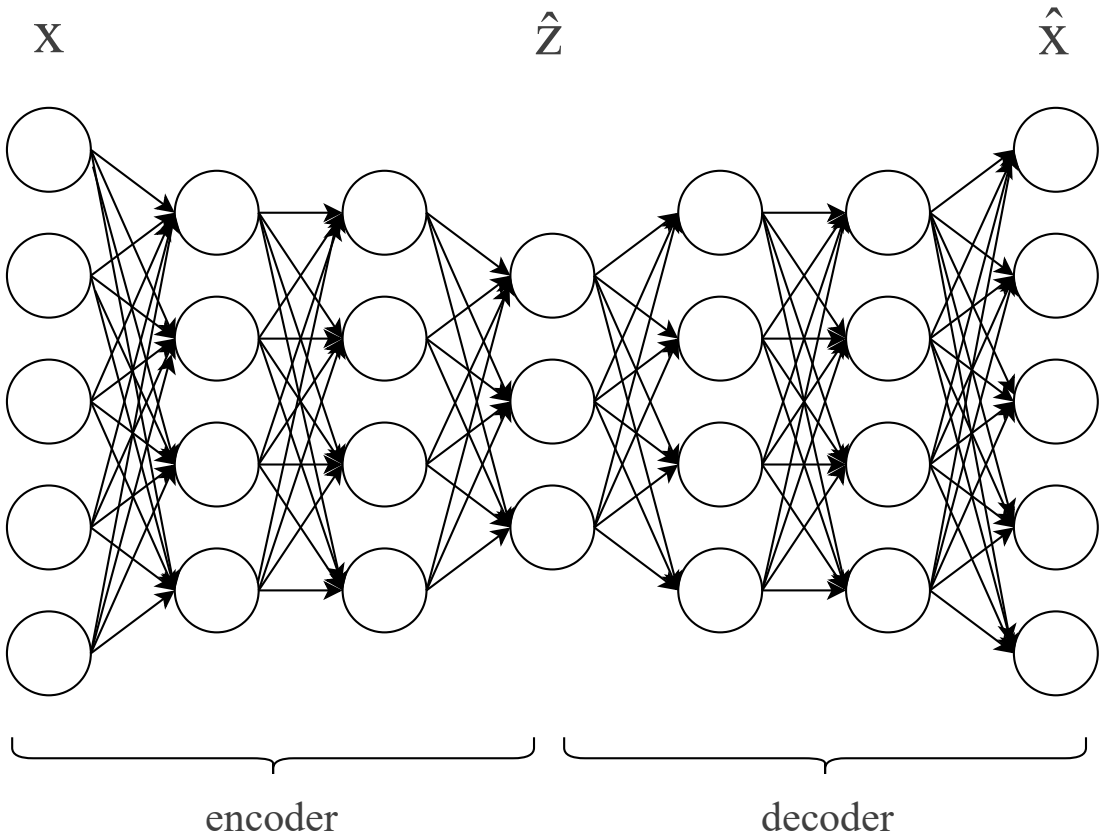
Linear Autoencoders

- Simplest form of autoencoder
- e and d are linear functions with shared weight matrix W

$$\hat{z} = Wx$$
$$\hat{x} = W^\top \hat{z}$$

Deep Neural Network Autoencoders

- Good for complex inputs
- e and d are feedforward neural networks, joined in series
- Train with backpropagation



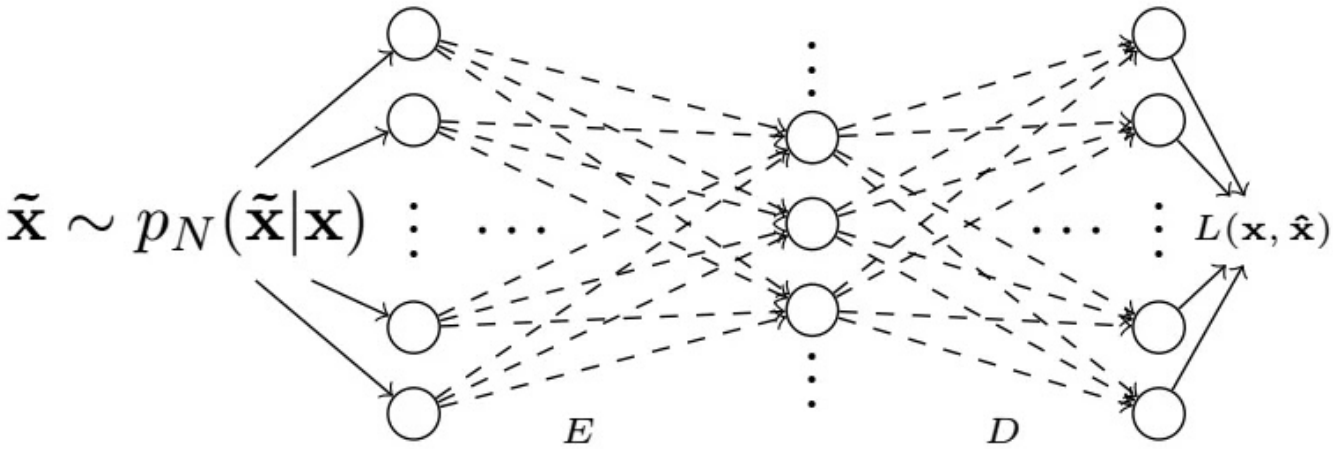
Deep Neural Network Autoencoders

- The bottleneck is the key of the effectiveness of Autoencoders. We map our input vector to bottleneck: the bottleneck keeps the 'latent informations' of input x.
- In other words, it behaves like a approximative compression algorithm. The encoding parameters are learned in training process. Then we map bottleneck information h into same dimension as input x.

Issues of Autoencoders

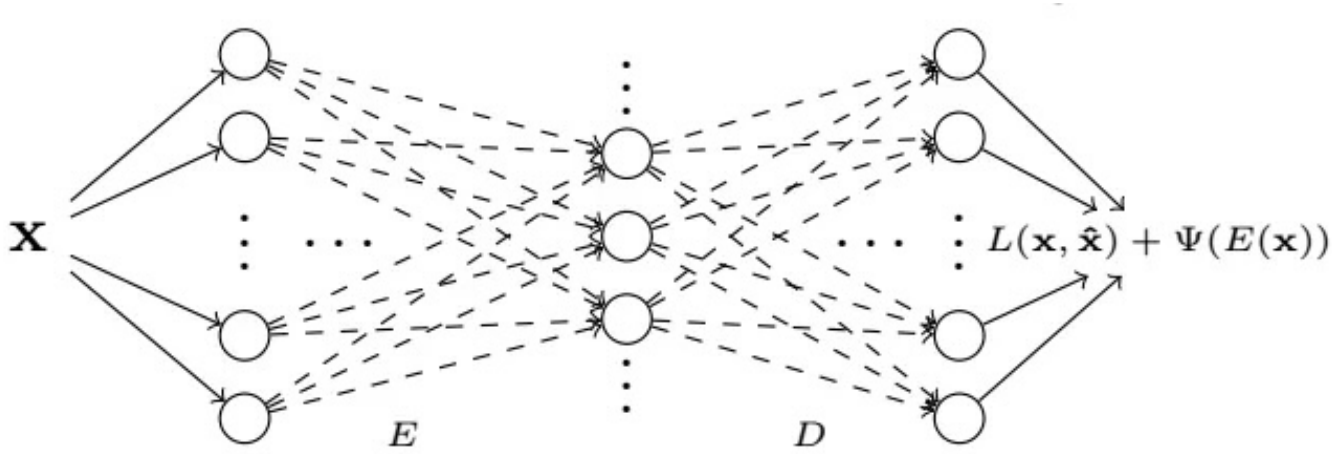
- When we said that main idea behind autoencoders is to copy input to its output, the key idea is that not to copy without extracting useful informations about the distribution of the data. Autoencoders are allowed too much capacity, easy to be trained to the copying the task with learning anything useful about the dataset
- As we said, autoencoders are allowerd too much capacity. Regularized autoencoders can give us the task that find the latent features of input, instead of copying the input.

Denoising Autoencoders



To perform the denoising, the input x is corrupted into $\tilde{x}$ through stochastic mapping of $\tilde{x} \sim p_N(\tilde{x}|x)$. Then the noisy (corrupted) input is used for encoding and decoding parts.

Sparse Autoencoders



The minimization objective involves an additional term

$$E = \sum_i (x_i - d(e(x_i)))^2 + \Phi(e(x_i)) \tag{11}$$

We can apply sparsity constraints on the encoded vector. For example, we can use zero-mean Laplacean prior, where:

$$Lap(e(x_i)|\mu = 0, \lambda) = \frac{\lambda}{2} \exp(-\lambda|h_i|) \tag{12}$$

Generative Adversarial Networks - Overview

a.k.a. GANs

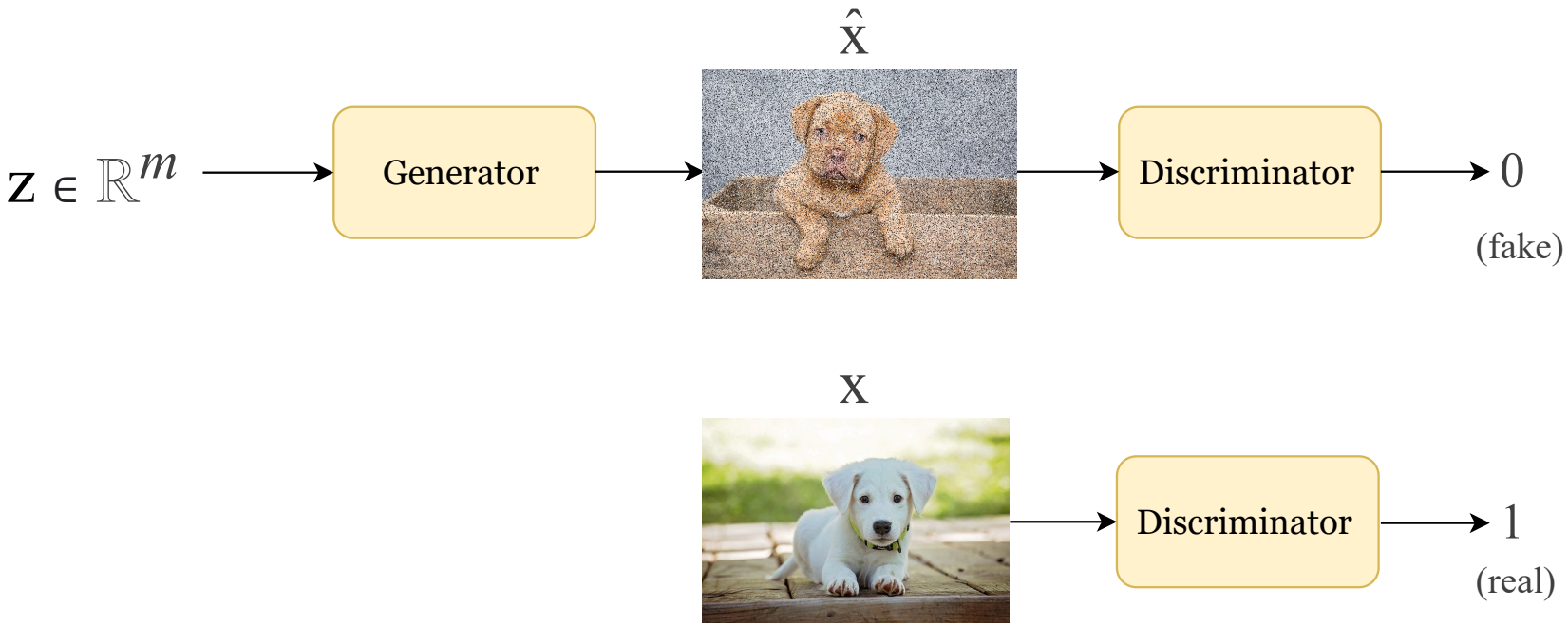
- A generative unsupervised learning algorithm
- Goal is to generate unseen examples that look like training examples

GANs - Components

GANs are actually a pair of neural networks:

- *Generator* $g(z)$: Given vector z in latent space, produces example x drawn from a distribution that approximates the true distribution of training examples
  $\rightarrow$ z usually sampled from a Gaussian distribution
- *Discriminator* $d(x)$: A classifier that predicts whether x is real (from training set) or fake (made by g)

GANs - Illustrative Example



GANs - Training

GANs are trained with a minimax error:

$$E = \mathbb{E}_x[\log(d(x))] + \mathbb{E}_z[\log(1 - d(g(z)))]$$

- Discriminator tries to maximize E
- Generator tries to minimize E

GANs - Training

GANs are trained with a minimax error:

$$E = \mathbb{E}_x[\log(d(x))] + \mathbb{E}_z[\log(1 - d(g(z)))]$$

- Discriminator tries to maximize E
- Generator tries to minimize E

After convergence:

- g should be producing realistic images
- d should output $\frac{1}{2}$, indicating maximal uncertainty

GANs - Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

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

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Revisiting Learning Goals

- Understanding what is unsupervised Learning
- Understanding K-Means clustering algorithm
- Knowing how to perform PCA
- Understanding the basic idea of Auto-Encoder and GAN