CSC2515 Lecture 7: PCA and K-Means

David Duvenaud

Based on Materials from Roger Grosse, University of Toronto

Overview

- Today: first examples of unsupervised learning algorithms.

Overview

- Today: first examples of unsupervised learning algorithms.
- Two traditional kinds of unsupervised learning:
  - Dimensionality reduction: map high-dimensional inputs to a lower-dimensional space that summarizes the important factors of variation.
    - Principal Component Analysis (PCA): mapping is a linear projection
    - Deep autoencoders: mapping is nonlinear

Overview

- Today: first examples of unsupervised learning algorithms.
- Two traditional kinds of unsupervised learning:
  - Dimensionality reduction: map high-dimensional inputs to a lower-dimensional space that summarizes the important factors of variation.
    - Principal Component Analysis (PCA): mapping is a linear projection
    - Deep autoencoders: mapping is nonlinear
  - Clustering: group the data points into discrete clusters
    - K-means (today): choose a set of cluster centers that minimize the Euclidean distance to the data points
    - Mixture of Gaussians (future lecture): learn a more flexible set of clusters that fit the data distribution well
  - Newer approach: Generative models: learn $p(x)$ and maybe sample from it, or $p(x_1|x_2)$ (e.g. GTP3, Dall-E)

Overview

- Today: first examples of unsupervised learning algorithms.
- Two traditional kinds of unsupervised learning:
  - Dimensionality reduction: map high-dimensional inputs to a lower-dimensional space that summarizes the important factors of variation.
    - Principal Component Analysis (PCA): mapping is a linear projection
    - Deep autoencoders: mapping is nonlinear
  - Clustering: group the data points into discrete clusters
    - K-means (today): choose a set of cluster centers that minimize the Euclidean distance to the data points
    - Mixture of Gaussians (future lecture): learn a more flexible set of clusters that fit the data distribution well
  - Newer approach: Generative models: learn $p(x)$ and maybe sample from it, or $p(x_1|x_2)$ (e.g. GTP3, Dall-E)
- We'll end by introducing maximum likelihood, a general approach for fitting $p(x)$ to data samples.

Dimensionality Reduction

- Images are intrinsically low-dimensional. Consider MNIST.
- Input space: $28 \times 28 = 784$ pixel values
- A lower dimensional representation: describe the strokes using 20 or so control points, plus a few more parameters for thickness, etc.

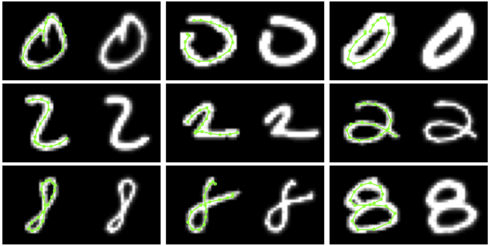
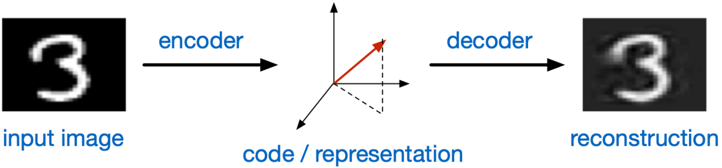


Image credit: Nair and Hinton (2006)

- Can we learn low-dimensional representations directly from the data?

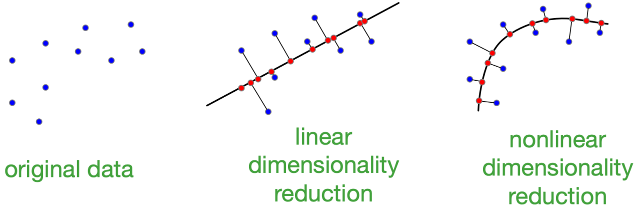
Dimensionality Reduction



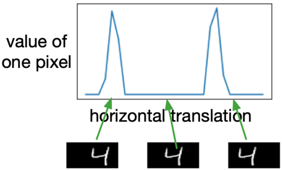
- In dimensionality reduction, we try to learn a mapping to a lower dimensional space that preserves as much information as possible about the input.
- Motivations
  - Save computation/memory
  - Reduce overfitting
  - Visualize in 2 dimensions

Dimensionality Reduction

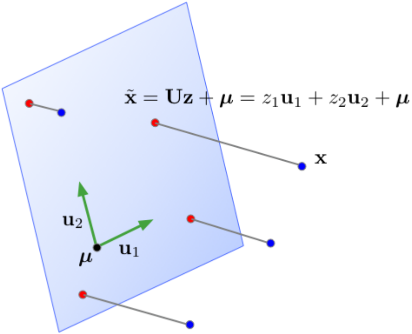
Can be linear or nonlinear:



- Linear dimensionality reduction methods (e.g. PCA) are much simpler, and easier to get to work.
- But many kinds of transformations behave nonlinearly in image space (e.g. translation of an image).



Projection onto a Subspace

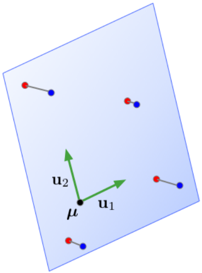


$$\mathbf{z} = \mathbf{U}^\top(\mathbf{x} - \boldsymbol{\mu})$$

- Here, the columns of $\mathbf{U}$ form an orthonormal basis for a subspace $\mathcal{S}$.
- The projection of a point $\mathbf{x}$ onto $\mathcal{S}$ is the point $\tilde{\mathbf{x}} \in \mathcal{S}$ closest to $\mathbf{x}$. In machine learning, $\tilde{\mathbf{x}}$ is also called the reconstruction of $\mathbf{x}$.
- $\mathbf{z}$ is its representation, or code.

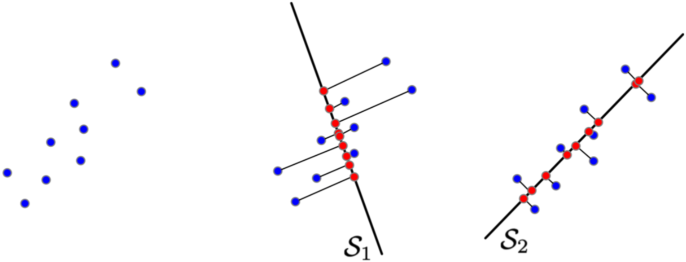
Projection onto a Subspace

- If we have a K-dimensional subspace in a D-dimensional input space, then $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{z} \in \mathbb{R}^K$.
- If the data points $\mathbf{x}$ all lie close to the subspace, then we can approximate distances, dot products, etc. in terms of these same operations on the code vectors $\mathbf{z}$.
- If $K \ll D$, then it's much cheaper to work with $\mathbf{z}$ than $\mathbf{x}$.



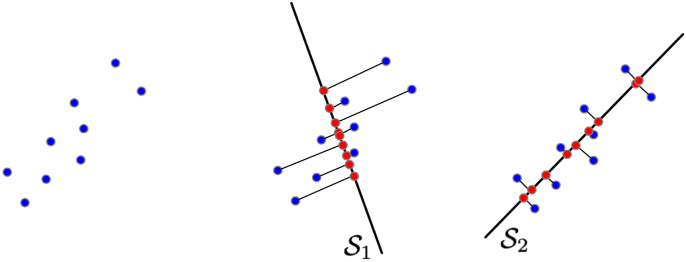
Learning a Subspace

- Which of the following subspaces is a better representation of the dataset?



Learning a Subspace

- Which of the following subspaces is a better representation of the dataset?



- On average, the data points are closer to $\mathcal{S}_2$ than to $\mathcal{S}_1$.
- The projections onto $\mathcal{S}_2$ are more spread out than the projections onto $\mathcal{S}_1$.

Learning a Subspace

- How to choose a good subspace $\mathcal{S}$?
  - Need to choose a vector $\boldsymbol{\mu}$ and a $D \times K$ matrix $\mathbf{U}$ with orthonormal columns.
- Set $\boldsymbol{\mu}$ to the mean of the data, $\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)}$

Learning a Subspace

- How to choose a good subspace $\mathcal{S}$?
  - Need to choose a vector $\boldsymbol{\mu}$ and a $D \times K$ matrix $\mathbf{U}$ with orthonormal columns.
- Set $\boldsymbol{\mu}$ to the mean of the data, $\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)}$
- Two criteria:
  - Minimize the reconstruction error

$$\min \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\|^2$$

Learning a Subspace

- How to choose a good subspace $\mathcal{S}$?
  - Need to choose a vector $\boldsymbol{\mu}$ and a $D \times K$ matrix $\mathbf{U}$ with orthonormal columns.
- Set $\boldsymbol{\mu}$ to the mean of the data, $\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)}$
- Two criteria:
  - Minimize the reconstruction error

$$\min \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\|^2$$

  - Maximize the variance of the code vectors

$$\begin{aligned} \max \sum_j \mathrm{Var}(z_j) &= \frac{1}{N} \sum_j \sum_i (z_j^{(i)} - \bar{z}_j)^2 \\ &= \frac{1}{N} \sum_i \|\mathbf{z}^{(i)} - \bar{\mathbf{z}}\|^2 \\ &= \frac{1}{N} \sum_i \|\mathbf{z}^{(i)}\|^2 \qquad \text{Exercise: show } \bar{\mathbf{z}} = 0 \end{aligned}$$

  - Note: here, $\bar{\mathbf{z}}$ denotes the mean, not a derivative.

Learning a Subspace

- These two criteria are equivalent! I.e., we'll show

$$\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}^{(i)}-\tilde{\mathbf{x}}^{(i)}\|^2 = \mathrm{const} - \frac{1}{N}\sum_{i}\|\mathbf{z}^{(i)}\|^2$$

Learning a Subspace

- These two criteria are equivalent! I.e., we'll show

$$\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}^{(i)}-\tilde{\mathbf{x}}^{(i)}\|^2 = \text{const} - \frac{1}{N}\sum_{i}\|\mathbf{z}^{(i)}\|^2$$

- Observation: by unitarity,

$$\|\tilde{\mathbf{x}}^{(i)}-\boldsymbol{\mu}\| = \|\mathbf{U}\mathbf{z}^{(i)}\| = \|\mathbf{z}^{(i)}\|$$

Learning a Subspace

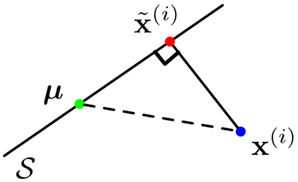
- These two criteria are equivalent! I.e., we'll show

$$\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}^{(i)}-\tilde{\mathbf{x}}^{(i)}\|^2 = \text{const} - \frac{1}{N}\sum_{i}\|\mathbf{z}^{(i)}\|^2$$

- Observation: by unitarity,

$$\|\tilde{\mathbf{x}}^{(i)}-\boldsymbol{\mu}\| = \|\mathbf{U}\mathbf{z}^{(i)}\| = \|\mathbf{z}^{(i)}\|$$

- By the Pythagorean Theorem,



$$\underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\tilde{\mathbf{x}}^{(i)}-\boldsymbol{\mu}\|^2}_{\text{projected variance}} + \underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}^{(i)}-\tilde{\mathbf{x}}^{(i)}\|^2}_{\text{reconstruction error}}$$

$$= \underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}^{(i)}-\boldsymbol{\mu}\|^2}_{\text{constant}}$$

Principal Component Analysis

Choosing a subspace to maximize the projected variance, or minimize the reconstruction error, is called principal component analysis (PCA).

Recall:

- Spectral Decomposition: a symmetric matrix $\mathbf{A}$ has a full set of eigenvectors, which can be chosen to be orthogonal. This gives a decomposition

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top,$$

  where $\mathbf{Q}$ is orthogonal and $\mathbf{\Lambda}$ is diagonal. The columns of $\mathbf{Q}$ are eigenvectors, and the diagonal entries λ_j of $\mathbf{\Lambda}$ are the corresponding eigenvalues.
- I.e., symmetric matrices are diagonal in some basis.
- A symmetric matrix $\mathbf{A}$ is positive semidefinite iff each $\lambda_j \geq 0$.

Principal Component Analysis

- Consider the empirical covariance matrix:

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}^{(i)} - \boldsymbol{\mu})(\mathbf{x}^{(i)} - \boldsymbol{\mu})^\top$$

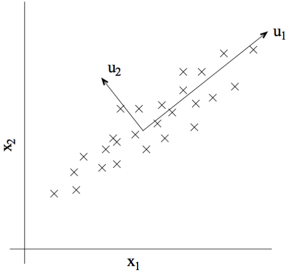
- Recall: Covariance matrices are symmetric and positive semidefinite.

Principal Component Analysis

- Consider the empirical covariance matrix:

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}^{(i)} - \boldsymbol{\mu})(\mathbf{x}^{(i)} - \boldsymbol{\mu})^\top$$

- Recall: Covariance matrices are symmetric and positive semidefinite.
- The optimal PCA subspace is spanned by the top K eigenvectors of $\boldsymbol{\Sigma}$.
  - More precisely, choose the first K of any orthonormal eigenbasis for $\boldsymbol{\Sigma}$.
  - The general case is tricky, but we'll show this for $K = 1$.
- These eigenvectors are called principal components, analogous to the principal axes of an ellipse.



Deriving PCA

- For $K = 1$, we are fitting a unit vector $\mathbf{u}$, and the code is a scalar $z = \mathbf{u}^\top(\mathbf{x} - \boldsymbol{\mu})$.

$$
\begin{aligned}
\frac{1}{N}\sum_i [z^{(i)}]^2 &= \frac{1}{N}\sum_i (\mathbf{u}^\top(\mathbf{x}^{(i)} - \boldsymbol{\mu}))^2 \\
&= \frac{1}{N}\sum_{i=1}^{N} \mathbf{u}^\top(\mathbf{x}^{(i)} - \boldsymbol{\mu})(\mathbf{x}^{(i)} - \boldsymbol{\mu})^\top \mathbf{u} \\
&= \mathbf{u}^\top \left[\frac{1}{N}\sum_{i=1}^{N} (\mathbf{x}^{(i)} - \boldsymbol{\mu})(\mathbf{x}^{(i)} - \boldsymbol{\mu})^\top\right] \mathbf{u} \\
&= \mathbf{u}^\top \boldsymbol{\Sigma} \mathbf{u} \\
&= \mathbf{u}^\top \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top \mathbf{u} && \text{Spectral Decomposition} \\
&= \mathbf{a}^\top \boldsymbol{\Lambda} \mathbf{a} && \text{for } \mathbf{a} = \mathbf{Q}^\top \mathbf{u} \\
&= \sum_{j=1}^{D} \lambda_j a_j^2
\end{aligned}
$$

Deriving PCA

- Maximize $\mathbf{a}^\top \mathbf{\Lambda} \mathbf{a} = \sum_{j=1}^{D} \lambda_j a_j^2$ for $\mathbf{a} = \mathbf{Q}^\top \mathbf{u}$.
  - This is a change-of-basis to the eigenbasis of $\mathbf{\Sigma}$.
- Assume the λ_i are in sorted order. For simplicity, assume they are all distinct.

Deriving PCA

- Maximize $\mathbf{a}^\top \boldsymbol{\Lambda} \mathbf{a} = \sum_{j=1}^{D} \lambda_j a_j^2$ for $\mathbf{a} = \mathbf{Q}^\top \mathbf{u}$.
  - This is a change-of-basis to the eigenbasis of $\boldsymbol{\Sigma}$.
- Assume the λ_i are in sorted order. For simplicity, assume they are all distinct.
- Observation: since $\mathbf{u}$ is a unit vector, then by unitarity, $\mathbf{a}$ is also a unit vector. I.e., $\sum_j a_j^2 = 1$.

Deriving PCA

- Maximize $\mathbf{a}^\top \boldsymbol{\Lambda} \mathbf{a} = \sum_{j=1}^{D} \lambda_j a_j^2$ for $\mathbf{a} = \mathbf{Q}^\top \mathbf{u}$.
  - This is a change-of-basis to the eigenbasis of $\boldsymbol{\Sigma}$.
- Assume the λ_i are in sorted order. For simplicity, assume they are all distinct.
- Observation: since $\mathbf{u}$ is a unit vector, then by unitarity, $\mathbf{a}$ is also a unit vector. I.e., $\sum_j a_j^2 = 1$.
- By inspection, set $a_1 = \pm 1$ and $a_j = 0$ for $j \neq 1$.
- Hence, $\mathbf{u} = \mathbf{Q}\mathbf{a} = \pm \mathbf{q}_1$ (the top eigenvector).

Deriving PCA

- Maximize $\mathbf{a}^\top \mathbf{\Lambda} \mathbf{a} = \sum_{j=1}^{D} \lambda_j a_j^2$ for $\mathbf{a} = \mathbf{Q}^\top \mathbf{u}$.
  - This is a change-of-basis to the eigenbasis of $\mathbf{\Sigma}$.
- Assume the λ_i are in sorted order. For simplicity, assume they are all distinct.
- Observation: since $\mathbf{u}$ is a unit vector, then by unitarity, $\mathbf{a}$ is also a unit vector. I.e., $\sum_j a_j^2 = 1$.
- By inspection, set $a_1 = \pm 1$ and $a_j = 0$ for $j \neq 1$.
- Hence, $\mathbf{u} = \mathbf{Q}\mathbf{a} = \pm \mathbf{q}_1$ (the top eigenvector).

- A similar argument shows that the kth principal component is the kth eigenvector of $\mathbf{\Sigma}$. If you're interested, look up the Courant-Fischer Theorem.

Decorrelation

- Interesting fact: the dimensions of $\mathbf{z}$ are decorrelated. For now, let Cov denote the empirical covariance.

$$
\begin{aligned}
\mathrm{Cov}(\mathbf{z}) &= \mathrm{Cov}(\mathbf{U}^\top(\mathbf{x} - \boldsymbol{\mu})) \\
&= \mathbf{U}^\top \mathrm{Cov}(\mathbf{x}) \mathbf{U} \\
&= \mathbf{U}^\top \boldsymbol{\Sigma} \mathbf{U} \\
&= \mathbf{U}^\top \mathbf{Q} \boldsymbol{\Lambda} \mathbf{Q}^\top \mathbf{U} \\
&= \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} \boldsymbol{\Lambda} \begin{pmatrix} \mathbf{I} \\ \mathbf{0} \end{pmatrix} && \text{by orthogonality} \\
&= \text{top left } K \times K \text{ block of } \boldsymbol{\Lambda}
\end{aligned}
$$

- If the covariance matrix is diagonal, this means the features are uncorrelated.
- This is why PCA was originally invented (in 1901!).

Recap

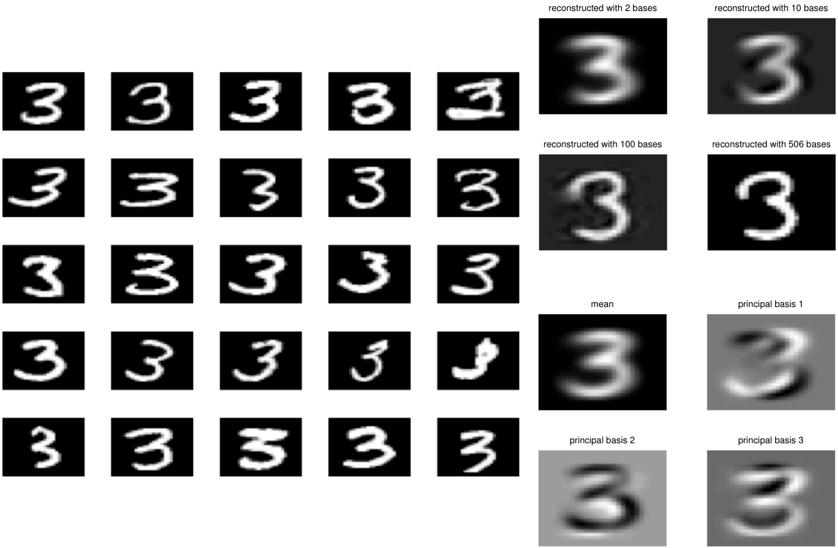
Recap:

- Dimensionality reduction aims to find a low-dimensional representation of the data.
- PCA projects the data onto a subspace which maximizes the projected variance, or equivalently, minimizes the reconstruction error.
- The optimal subspace is given by the top eigenvectors of the empirical covariance matrix.
- PCA gives a set of decorrelated features.

Applying PCA to faces: Learned basis

Principal components of face images ("eigenfaces")

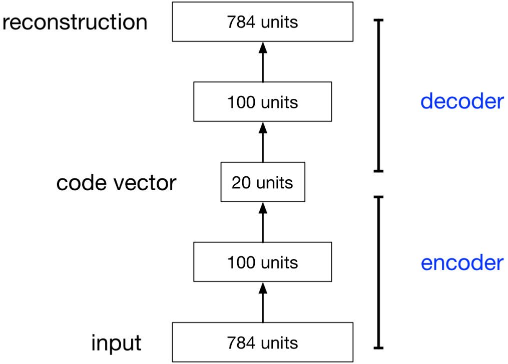
Applying PCA to digits



Autoencoders and Nonlinear Dimensionality Reduction

Autoencoders

- An autoencoder is a feed-forward neural net whose job it is to take an input $\mathbf{x}$ and predict $\mathbf{x}$.
- To make this non-trivial, we need to add a bottleneck layer whose dimension is much smaller than the input.



Linear Autoencoders

Why autoencoders?

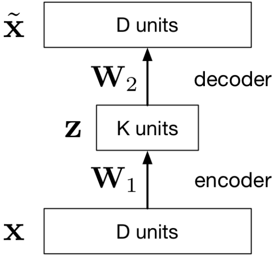
- Map high-dimensional data to two dimensions for visualization
- Learn abstract features in an unsupervised way so you can apply them to a supervised task
  - Unlabled data can be much more plentiful than labeled data

Linear Autoencoders

- The simplest kind of autoencoder has one hidden layer, linear activations, and squared error loss.

$$\mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|^2$$

- This network computes $\tilde{\mathbf{x}} = \mathbf{W}_2\mathbf{W}_1\mathbf{x}$, which is a linear function.
- If $K \geq D$, we can choose $\mathbf{W}_2$ and $\mathbf{W}_1$ such that $\mathbf{W}_2\mathbf{W}_1$ is the identity matrix. This isn't very interesting.



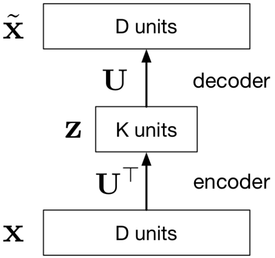
- But suppose $K < D$:
  - $\mathbf{W}_1$ maps $\mathbf{x}$ to a K-dimensional space, so it's doing dimensionality reduction.

Linear Autoencoders

- Observe that the output of the autoencoder must lie in a K-dimensional subspace spanned by the columns of $\mathbf{W}_2$.
- We saw that the best possible K-dimensional subspace in terms of reconstruction error is the PCA subspace.

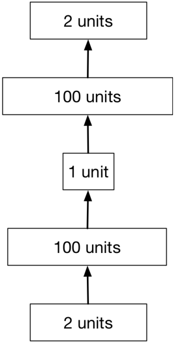
Linear Autoencoders

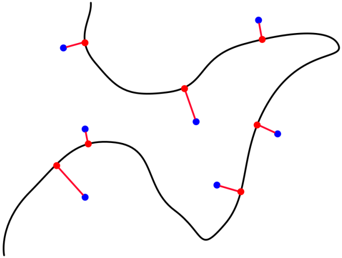
- Observe that the output of the autoencoder must lie in a K-dimensional subspace spanned by the columns of $\mathbf{W}_2$.
- We saw that the best possible K-dimensional subspace in terms of reconstruction error is the PCA subspace.
- The autoencoder can achieve this by setting $\mathbf{W}_1 = \mathbf{U}^\top$ and $\mathbf{W}_2 = \mathbf{U}$.
- Therefore, the optimal weights for a linear autoencoder are just the principal components!



Nonlinear Autoencoders

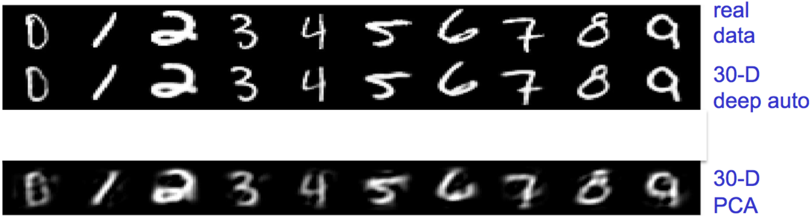
- Deep nonlinear autoencoders learn to project the data, not onto a subspace, but onto a nonlinear manifold
- This manifold is the image of the decoder.
- This is a kind of nonlinear dimensionality reduction.





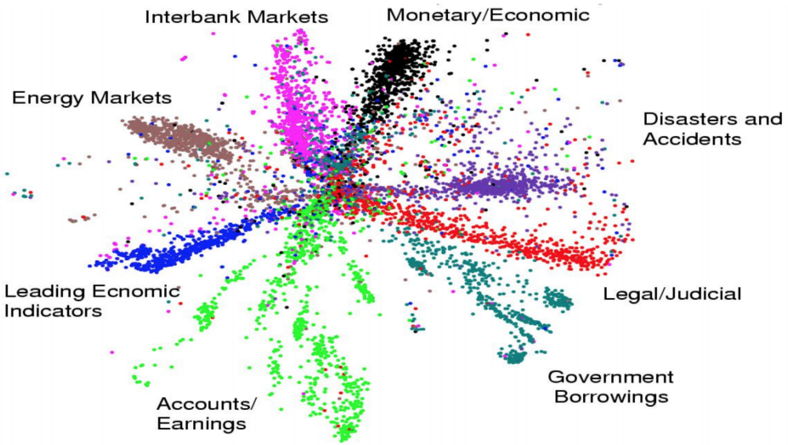
Nonlinear Autoencoders

- Nonlinear autoencoders can learn more powerful codes for a given dimensionality, compared with linear autoencoders (PCA)



Nonlinear Autoencoders

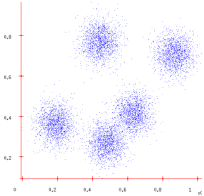
Here's a 2-dimensional autoencoder representation of newsgroup articles. They're color-coded by topic, but the algorithm wasn't given the labels.



Clustering and K-Means

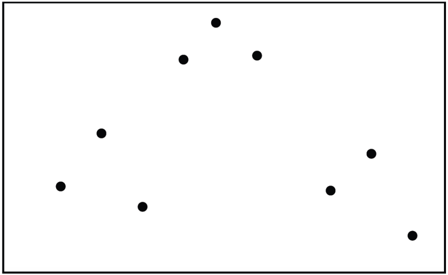
Clustering

- Sometimes the data form clusters, where examples within a cluster are similar to each other, and examples in different clusters are dissimilar:



- Such a distribution is multimodal, since it has multiple modes, or regions of high probability mass.
- Grouping data points into clusters, with no labels, is called clustering
- E.g. clustering machine learning papers based on topic (deep learning, Bayesian models, etc.)
  - This is an overly simplistic model — more on that later

Clustering



- Assume the data $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$ lives in a Euclidean space, $\mathbf{x}^{(n)} \in \mathbb{R}^d$.
- Assume the data belongs to K classes (patterns)
- Assume the data points from same class are similar, i.e. close in Euclidean distance.
- How can we identify those classes (data points that belong to each class)?

K-means intuition

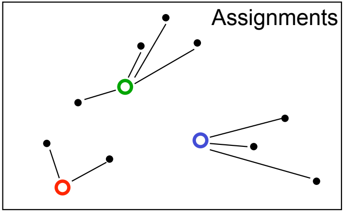
- K-means assumes there are k clusters, and each point is close to its cluster center (the mean of points in the cluster).
- If we knew the cluster assignment we could easily compute means.
- If we knew the means we could easily compute cluster assignment.
- Chicken and egg problem!
- Can show it is NP hard.
- Very simple (and useful) heuristic - start randomly and alternate between the two!

K-means

- Initialization: randomly initialize cluster centers

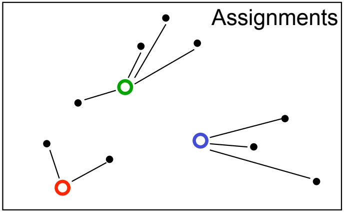
K-means

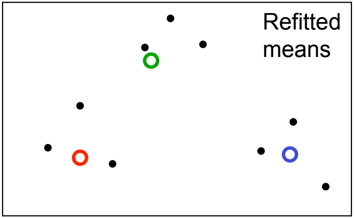
- Initialization: randomly initialize cluster centers
- The algorithm iteratively alternates between two steps:
  - Assignment step: Assign each data point to the closest cluster



K-means

- Initialization: randomly initialize cluster centers
- The algorithm iteratively alternates between two steps:
  - Assignment step: Assign each data point to the closest cluster
  - Refitting step: Move each cluster center to the center of gravity of the data assigned to it





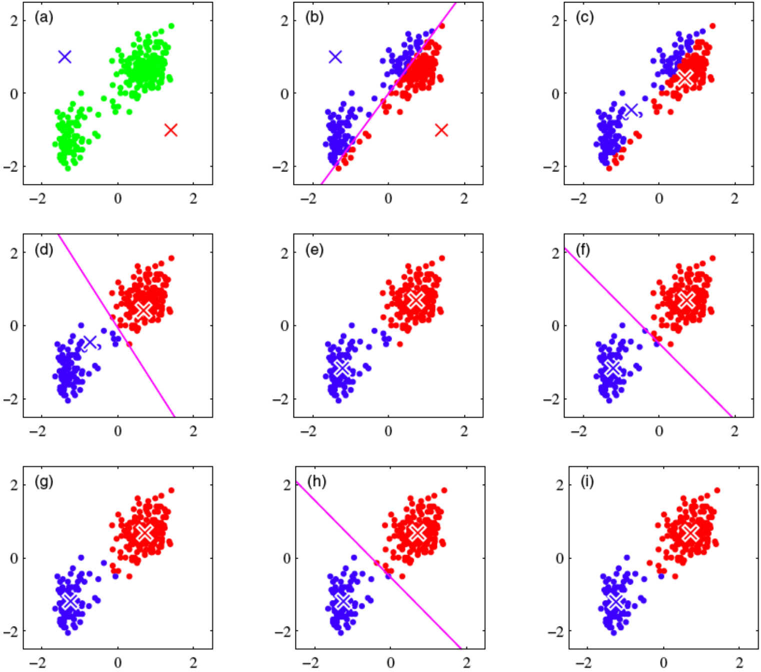


Figure from Bishop

Simple demo: http://syskall.com/kmeans.js/

K-means Objective

What is actually being optimized?

K-means Objective

What is actually being optimized?

K-means Objective:
Find cluster centers $\mathbf{m}$ and assignments $\mathbf{r}$ to minimize the sum of squared distances of data points $\{\mathbf{x}^{(n)}\}$ to their assigned cluster centers

$$\min_{\{\mathbf{m}\},\{\mathbf{r}\}} J(\{\mathbf{m}\},\{\mathbf{r}\}) = \min_{\{\mathbf{m}\},\{\mathbf{r}\}} \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)} ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$$

$$\text{s.t.} \sum_k r_k^{(n)} = 1, \forall n, \quad \text{where} \qquad r_k^{(n)} \in \{0,1\}, \forall k, n$$

where $r_k^{(n)} = 1$ means that $\mathbf{x}^{(n)}$ is assigned to cluster k (with center $\mathbf{m}_k$)

K-means Objective

What is actually being optimized?

K-means Objective:
Find cluster centers $\mathbf{m}$ and assignments $\mathbf{r}$ to minimize the sum of squared distances of data points $\{\mathbf{x}^{(n)}\}$ to their assigned cluster centers

$$\min_{\{\mathbf{m}\},\{\mathbf{r}\}} J(\{\mathbf{m}\},\{\mathbf{r}\}) = \min_{\{\mathbf{m}\},\{\mathbf{r}\}} \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)} ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$$

$$\text{s.t.} \sum_k r_k^{(n)} = 1, \forall n, \quad \text{where} \quad r_k^{(n)} \in \{0,1\}, \forall k, n$$

where $r_k^{(n)} = 1$ means that $\mathbf{x}^{(n)}$ is assigned to cluster k (with center $\mathbf{m}_k$)

- Optimization method is a form of coordinate descent ("block coordinate descent")
  - Fix centers, optimize assignments (choose cluster whose mean is closest)
  - Fix assignments, optimize means (average of assigned datapoints)

The K-means Algorithm

- Initialization: Set K cluster means $\mathbf{m}_1, \ldots, \mathbf{m}_K$ to random values
- Repeat until convergence (until assignments do not change):
  - Assignment: Each data point $\mathbf{x}^{(n)}$ assigned to nearest mean

    $$\hat{k}^n = arg \min_k d(\mathbf{m}_k, \mathbf{x}^{(n)})$$

    (with, for example, L2 norm: $\hat{k}^n = arg \min_k ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$)

    and Responsibilities (1-hot encoding)

    $$r_k^{(n)} = 1 \longleftrightarrow \hat{k}^{(n)} = k$$

  - Refitting: Model parameters, means are adjusted to match sample means of data points they are responsible for:

    $$\mathbf{m}_k = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{\sum_n r_k^{(n)}}$$

K-means for Vector Quantization

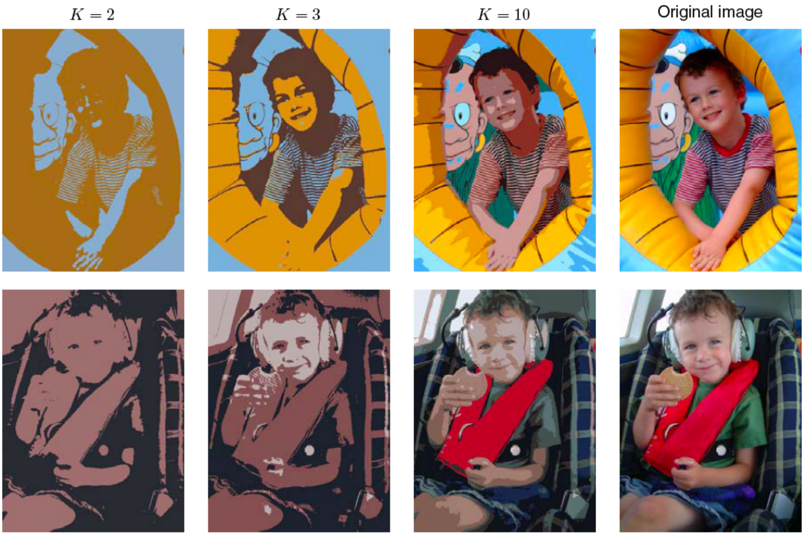
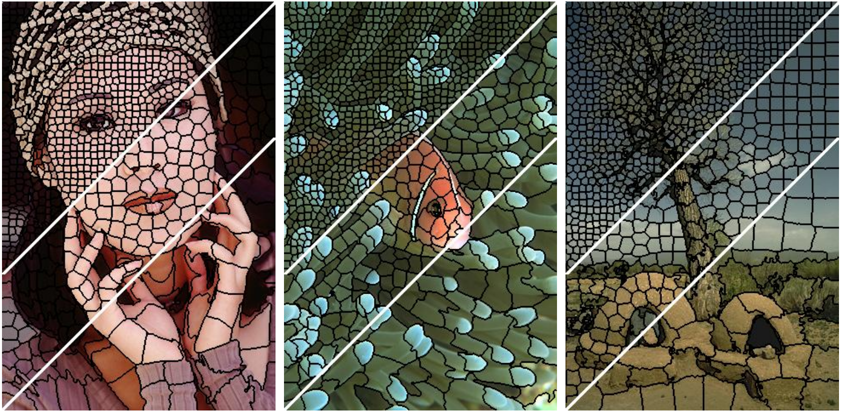


Figure from Bishop

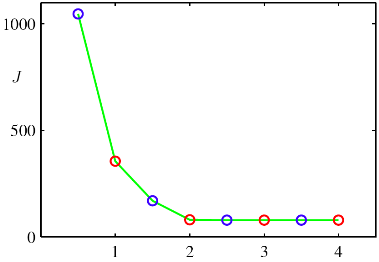
K-means for Image Segmentation



- How would you modify k-means to get superpixels?

Why K-means Converges

- Whenever an assignment is changed, the sum squared distances J of data points from their assigned cluster centers is reduced.
- Whenever a cluster center is moved, J is reduced.
- Test for convergence: If the assignments do not change in the assignment step, we have converged (to at least a local minimum).

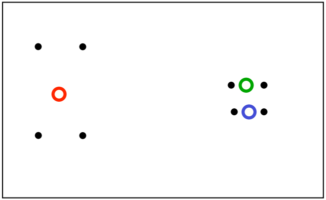


- K-means cost function after each E step (blue) and M step (red). The algorithm has converged after the third M step

Local Minima

- The objective J is non-convex (so coordinate descent on J is not guaranteed to converge to the global minimum)
- There is nothing to prevent k-means getting stuck at local minima.
- We could try many random starting points
- We could try non-local split-and-merge moves:
    - Simultaneously merge two nearby clusters
    - and split a big cluster into two

A bad local optimum



Soft K-means

- Instead of making hard assignments of data points to clusters, we can make soft assignments. One cluster may have a responsibility of .7 for a datapoint and another may have a responsibility of .3.
  - Allows a cluster to use more information about the data in the refitting step.
  - What happens to our convergence guarantee?
  - How do we decide on the soft assignments?

Soft K-means Algorithm

- Initialization: Set K means $\{\mathbf{m}_k\}$ to random values
- Repeat until convergence (until assignments do not change):
  - Assignment: Each data point n given soft "degree of assignment" to each cluster mean k, based on responsibilities

$$r_k^{(n)} = \frac{\exp[-\beta d(\mathbf{m}_k, \mathbf{x}^{(n)})]}{\sum_j \exp[-\beta d(\mathbf{m}_j, \mathbf{x}^{(n)})]}$$

  - Refitting: Model parameters, means, are adjusted to match sample means of datapoints they are responsible for:

$$\mathbf{m}_k = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{\sum_n r_k^{(n)}}$$

Probabilistic Models and Maximum Likelihood

Maximum Likelihood

- PCA and K-Means are procedures that capture particular types of structure.
- Recall: unifying picture of supervised learning in terms of models, loss functions, and optimization algorithms
- Probabilistic models play an analogous role for unsupervised learning (and sometimes supervised learning as well).
  - Treat the quantities of interest as random variables, and specify the form of their probabilistic dependencies.
  - Infer unknown quantities from the observations by performing probabilistic inference.
- Today: maximum likelihood, which is one tool we need for fitting probabilistic models.

Maximum Likelihood

- Motivating example: estimating the parameter of a biased coin
  - You flip a coin 100 times. It lands heads $N_H = 55$ times and tails $N_T = 45$ times.
  - What is the probability it will come up heads if we flip again?
- Model: flips are independent Bernoulli random variables with parameter θ.
  - Assume the observations are independent and identically distributed (i.i.d.)

Maximum Likelihood

- The likelihood function is the probability of the observed data, as a function of θ.
- In our case, it's the probability of a *particular* sequence of H's and T's.
- Under the Bernoulli model with i.i.d. observations,

$$L(\theta) = p(\mathcal{D}) = \theta^{N_H}(1-\theta)^{N_T}$$

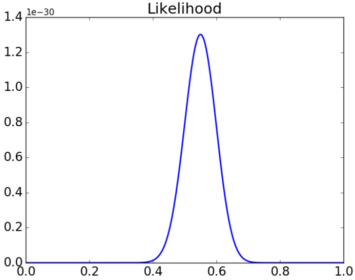
- This takes very small values (in this case, $L(0.5) = 0.5^{100} \approx 7.9 \times 10^{-31}$)
- Therefore, we usually work with log-likelihoods:

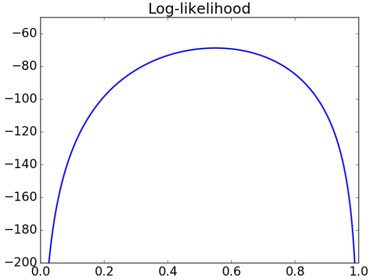
$$\ell(\theta) = \log L(\theta) = N_H \log \theta + N_T \log(1-\theta)$$

- Here, $\ell(0.5) = \log 0.5^{100} = 100 \log 0.5 = -69.31$

Maximum Likelihood

$N_H = 55,\ N_T = 45$





Maximum Likelihood

- Good values of θ should assign high probability to the observed data. This motivates the maximum likelihood criterion.
- Remember how we found the optimal solution to linear regression by setting derivatives to zero? We can do that again for the coin example.

$$\begin{aligned}\frac{\mathrm{d}\ell}{\mathrm{d}\theta} &= \frac{\mathrm{d}}{\mathrm{d}\theta}\left(N_H \log \theta + N_T \log(1-\theta)\right)\\ &= \frac{N_H}{\theta} - \frac{N_T}{1-\theta}\end{aligned}$$

- Setting this to zero gives the maximum likelihood estimate:

$$\hat{\theta}_{\mathrm{ML}} = \frac{N_H}{N_H + N_T},$$

Maximum Likelihood

- This is equivalent to minimizing cross-entropy. Let $t_i = 1$ for heads and $t_i = 0$ for tails.

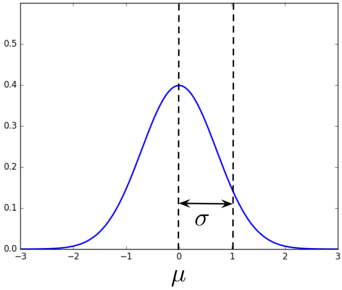
$$\begin{aligned}
\mathcal{L}_{CE} &= -\sum_i t_i \log \theta - (1 - t_i) \log(1 - \theta) \\
&= -N_H \log \theta - N_T \log(1 - \theta) \\
&= -\ell(\theta)
\end{aligned}$$

Maximum Likelihood

- Recall the Gaussian, or normal, distribution:

$$\mathcal{N}(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

- The Central Limit Theorem says that sums of lots of independent random variables are approximately Gaussian.
- In machine learning, we use Gaussians a lot because they make the calculations easy.



Maximum Likelihood

- Suppose we want to model the distribution of temperatures in Toronto in March, and we've recorded the following observations:

  -2.5 -9.9 -12.1 -8.9 -6.0 -4.8 2.4

- Assume they're drawn from a Gaussian distribution with known standard deviation $\sigma = 5$, and we want to find the mean μ.
- Log-likelihood function:

$$\begin{aligned}
\ell(\mu) &= \log \prod_{i=1}^{N} \left[\frac{1}{\sqrt{2\pi} \cdot \sigma} \exp\left(-\frac{(x^{(i)} - \mu)^2}{2\sigma^2} \right) \right] \\
&= \sum_{i=1}^{N} \log \left[\frac{1}{\sqrt{2\pi} \cdot \sigma} \exp\left(-\frac{(x^{(i)} - \mu)^2}{2\sigma^2} \right) \right] \\
&= \sum_{i=1}^{N} \underbrace{-\frac{1}{2} \log 2\pi - \log \sigma}_{\text{constant!}} - \frac{(x^{(i)} - \mu)^2}{2\sigma^2}
\end{aligned}$$

Maximum Likelihood

- Maximize the log-likelihood by setting the derivative to zero:

$$
\begin{aligned}
0 = \frac{\mathrm{d}\ell}{\mathrm{d}\mu} &= -\frac{1}{2\sigma^2} \sum_{i=1}^{N} \frac{\mathrm{d}}{\mathrm{d}\mu} (x^{(i)} - \mu)^2 \\
&= \frac{1}{\sigma^2} \sum_{i=1}^{N} x^{(i)} - \mu
\end{aligned}
$$

- Solving we get $\hat{\mu}_{\mathrm{ML}} = \frac{1}{N} \sum_{i=1}^{N} x^{(i)}$
- This is just the mean of the observed values, or the empirical mean.

Maximum Likelihood

- In general, we don't know the true standard deviation σ, but we can solve for it as well.
- Set the *partial* derivatives to zero, just like in linear regression.

$$0 = \frac{\partial \ell}{\partial \mu} = -\frac{1}{\sigma^2} \sum_{i=1}^{N} x^{(i)} - \mu$$

$$\begin{aligned} 0 = \frac{\partial \ell}{\partial \sigma} &= \frac{\partial}{\partial \sigma} \left[\sum_{i=1}^{N} -\frac{1}{2} \log 2\pi - \log \sigma - \frac{1}{2\sigma^2}(x^{(i)} - \mu)^2 \right] \\ &= \sum_{i=1}^{N} -\frac{1}{2} \frac{\partial}{\partial \sigma} \log 2\pi - \frac{\partial}{\partial \sigma} \log \sigma - \frac{\partial}{\partial \sigma} \frac{1}{2\sigma}(x^{(i)} - \mu)^2 \\ &= \sum_{i=1}^{N} 0 - \frac{1}{\sigma} + \frac{1}{\sigma^3}(x^{(i)} - \mu)^2 \\ &= -\frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_{i=1}^{N} (x^{(i)} - \mu)^2 \end{aligned}$$

$$\hat{\mu}_{\mathrm{ML}} = \frac{1}{N} \sum_{i=1}^{N} x^{(i)}$$

$$\hat{\sigma}_{\mathrm{ML}} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x^{(i)} - \hat{\mu}_{\mathrm{ML}})^2}$$

Maximum Likelihood

- Sometimes there is no closed-form solution. E.g., consider the gamma distribution, whose PDF is

$$p(x) = \frac{b^a}{\Gamma(a)} x^{a-1} e^{-bx},$$

where Γ is the gamma function, a generalization of the factorial function to continuous values.

- There is no closed-form solution, but we can still optimize the log-likelihood using gradient ascent.

Maximum Likelihood

- So far, maximum likelihood has told us to use empirical counts or statistics:
  - **Bernoulli:** $\hat{\theta}_{\mathrm{ML}} = \frac{N_H}{N_H + N_T}$
  - **Gaussian:** $\hat{\mu}_{\mathrm{ML}} = \frac{1}{N} \sum x^{(i)}$, $\hat{\sigma}^2_{\mathrm{ML}} = \frac{1}{N} \sum (x^{(i)} - \hat{\mu}_{\mathrm{ML}})^2$
- This doesn't always happen; the class of probability distributions that have this property is exponential families.

Maximum Likelihood

We've been doing maximum likelihood estimation all along!

- Squared error loss (e.g. linear regression)

$$
\begin{aligned}
p(t|y) &= \mathcal{N}(t; y, \sigma^2) \\
-\log p(t|y) &= \frac{1}{2\sigma^2}(y - t)^2 + \text{const}
\end{aligned}
$$

- Cross-entropy loss (e.g. logistic regression)

$$
\begin{aligned}
p(t = 1|y) &= y \\
-\log p(t|y) &= -t \log y - (1 - t) \log(1 - y)
\end{aligned}
$$