

# Unsupervised machine learning

## Statistical Natural Language Processing

Çağrı Çöltekin

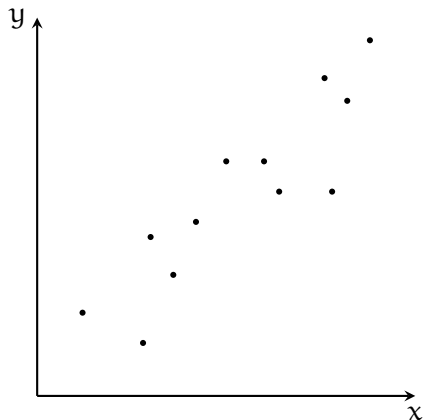
University of Tübingen  
Seminar für Sprachwissenschaft

Summer Semester 2021

# Supervised learning

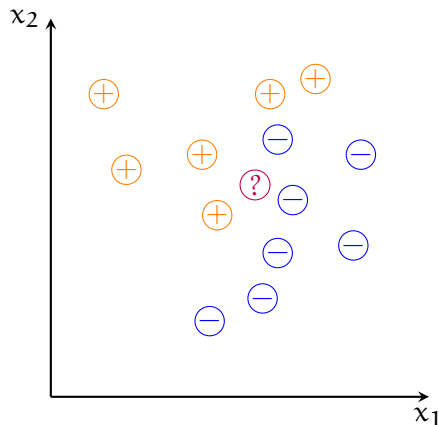
- The methods we studied so far are instances of supervised learning
- In supervised learning, we have a set of predictors  $\mathbf{x}$ , and want to predict a response or outcome variable  $\mathbf{y}$
- During training, we have both input and output variables
- Training consist of estimating parameters  $\mathbf{w}$  of a model
- During prediction, we are given  $\mathbf{x}$  and make predictions based on model we learned

# Supervised learning: regression



- The response (outcome) variable ( $y$ ) is a quantitative variable.
- Given the features ( $x$ ) we want to predict the value of  $y$

# Supervised learning: classification



- The response (outcome) is a label. In the example: positive  $\oplus$  or negative  $\ominus$
- Given the features ( $x_1$  and  $x_2$ ), we want to predict the label of an unknown instance  $?$

# Supervised learning

how do we learn?

- The aim is to estimate a set of parameters  $\mathbf{w}$
- We define an *objective function*, and find the parameter values that minimize the objective
- The objective typically involves reducing the training error defined based on the true labels in the training data

# Unsupervised learning

- In unsupervised learning, we do not have labels in our training data
- Our aim is to find useful patterns/structure in the data
  - for exploratory study of the data
  - for augmenting / complementing supervised methods
- Close relationships with ‘data mining’, ‘data science / analytics’, ‘knowledge discovery’
- Most unsupervised methods can be cast as graphical models with hidden variables
- Evaluation is difficult: we do not have ‘true’ labels/values

# Today's lecture

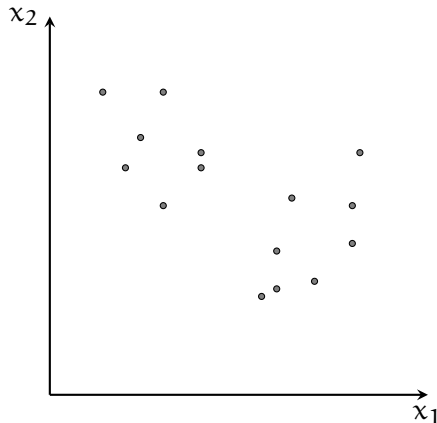
- *Clustering*: find related groups of instances
- *Density estimation*: find a probability distribution that explains the data
- *Dimensionality reduction*: find an accurate/useful lower dimensional representation of the data
- Unsupervised learning in ANNs (RBMs, autoencoders)

# Clustering: why do we do it?

- The aim is to find groups of instances/items that are similar to each other
- Applications include
  - Clustering languages, dialects for determining their relations
  - Clustering (literary) texts, for e.g., authorship attribution
  - Clustering words for e.g., better parsing
  - Clustering documents, e.g., news into topics
  - ...

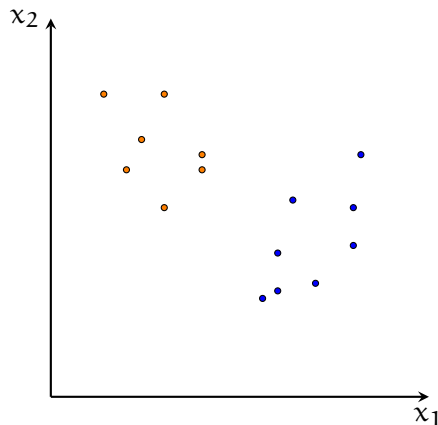


# Clustering in two dimensional space



- Unlike classification, we do not have labels

# Clustering in two dimensional space



- Unlike classification, we do not have labels
- We want to find 'natural' groups in the data
- Intuitively, similar or closer data points are grouped together

# Similarity and distance

- The notion of distance (similarity) is important in clustering. A distance measure  $D$ ,
  - is symmetric:  $D(a, b) = D(b, a)$
  - non-negative:  $D(a, b) \geq 0$   
for all  $a, b$ , and it  $D(a, b) = 0$  iff  $a = b$
  - obeys triangle inequality:  $D(a, b) + D(b, c) \geq D(a, c)$
- The choice of distance is application specific
- We will often face with defining distance measures between linguistic units (letters, words, sentences, documents, ...)

# Distance measures in Euclidean space

- Euclidean distance:

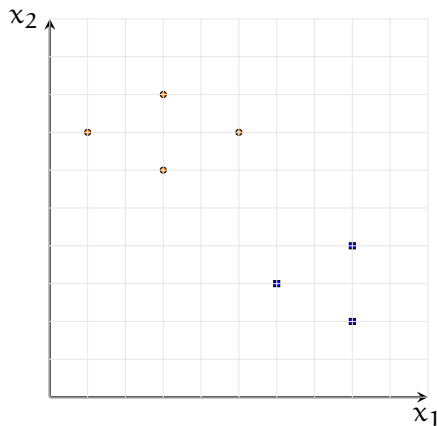
$$\|\mathbf{a} - \mathbf{b}\| = \sqrt{\sum_{j=1}^k (a_j - b_j)^2}$$

- Manhattan distance:

$$\|\mathbf{a} - \mathbf{b}\|_1 = \sum_{j=1}^k |a_j - b_j|$$

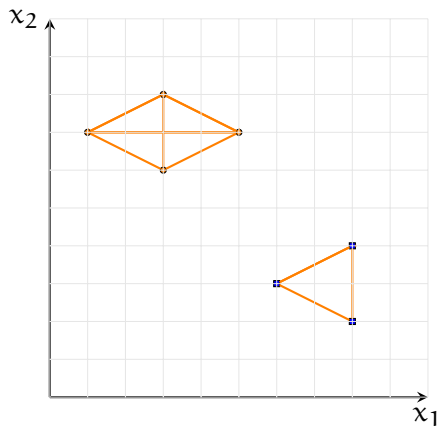
# How to do clustering

Most clustering algorithms try to minimize the scatter **within** each cluster. Which is equivalent to maximizing the scatter **between** clusters.



# How to do clustering

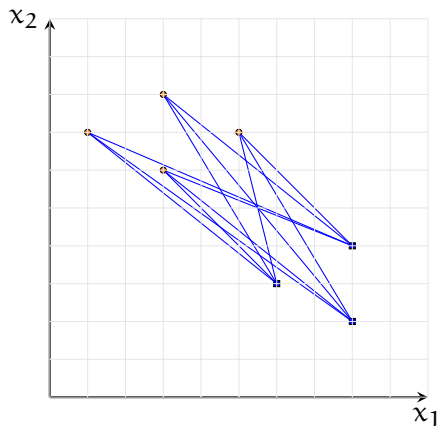
Most clustering algorithms try to minimize the scatter **within** each cluster. Which is equivalent to maximizing the scatter **between** clusters.



$$\sum_{k=1}^K \sum_{a \in C_k} \sum_{b \in C_k} d(a, b)$$

# How to do clustering

Most clustering algorithms try to minimize the scatter **within** each cluster. Which is equivalent to maximizing the scatter **between** clusters.

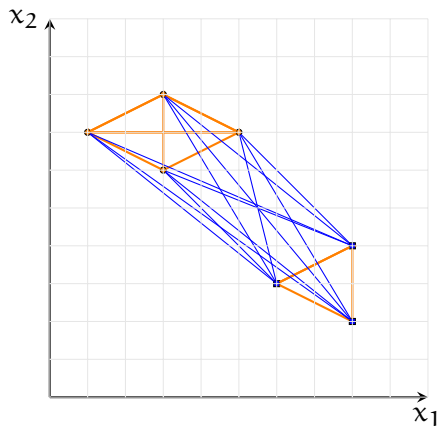


$$\sum_{k=1}^K \sum_{a \in C_k} \sum_{b \in C_k} d(a, b)$$

$$\sum_{k=1}^K \sum_{a \in C_k} \sum_{b \notin C_k} d(a, b)$$

# How to do clustering

Most clustering algorithms try to minimize the scatter **within** each cluster. Which is equivalent to maximizing the scatter **between** clusters.



$$\sum_{k=1}^K \sum_{a \in C_k} \sum_{b \in C_k} d(a, b)$$

$$\sum_{k=1}^K \sum_{a \in C_k} \sum_{b \notin C_k} d(a, b)$$



# K-means algorithm

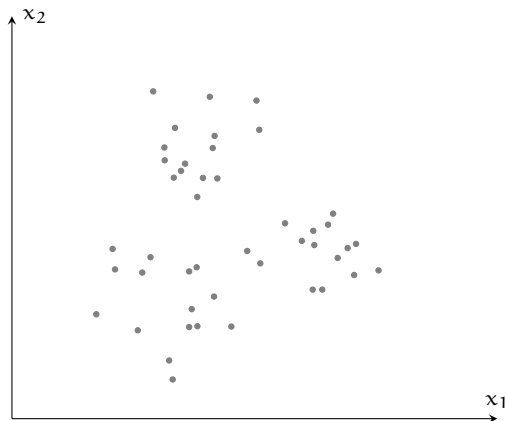
K-means is a popular method for clustering.

1. Randomly choose *centroids*,  $m_1, \dots, m_K$ , representing K clusters
2. Repeat until convergence
  - Assign each data point to the cluster of the nearest centroid
  - Re-calculate the centroid locations based on the assignments

Effectively, we are finding a *local minimum* of the sum of squared Euclidean distance within each cluster

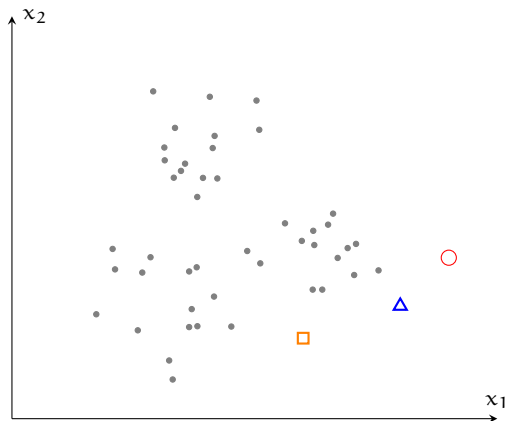
$$\frac{1}{2} \sum_{k=1}^K \sum_{a \in C_k} \sum_{b \in C_k} \|a - b\|^2$$

# K-means clustering: visualization



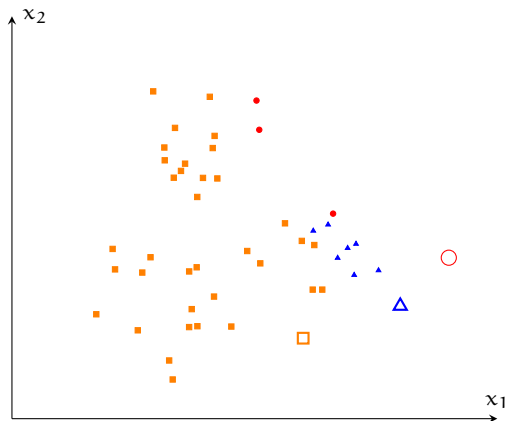
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

# K-means clustering: visualization



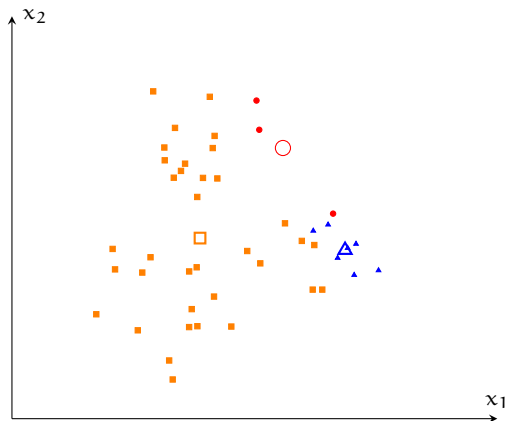
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

# K-means clustering: visualization



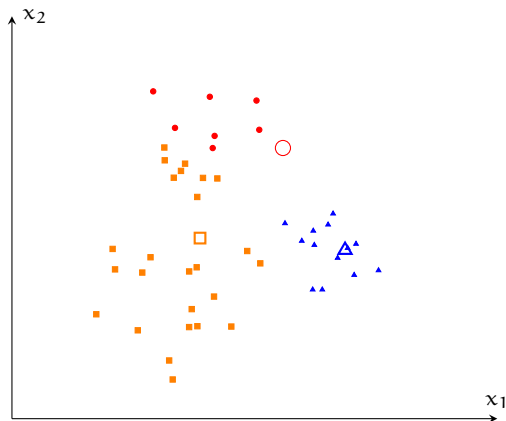
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

# K-means clustering: visualization



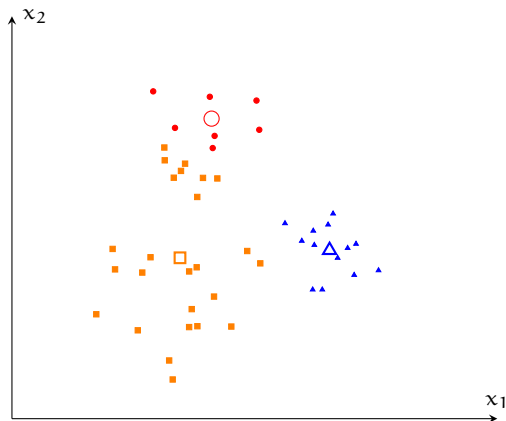
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

# K-means clustering: visualization



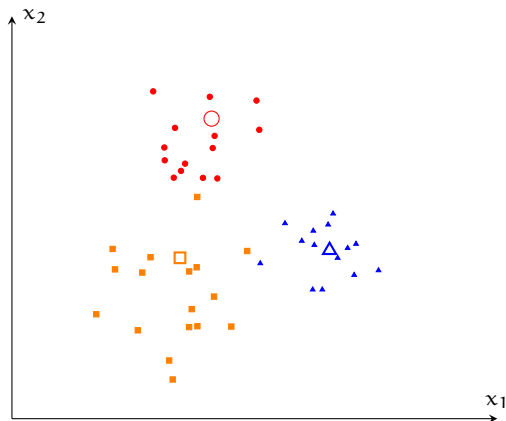
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

# K-means clustering: visualization



- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

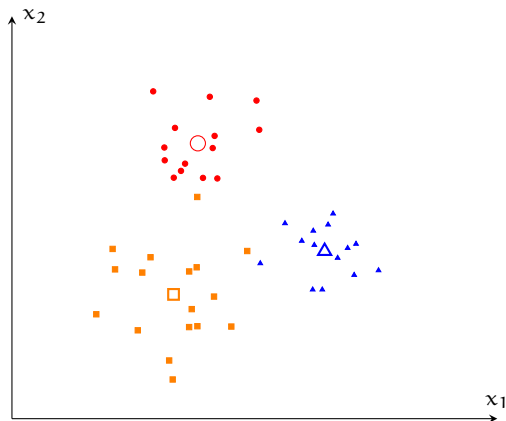
# K-means clustering: visualization



- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

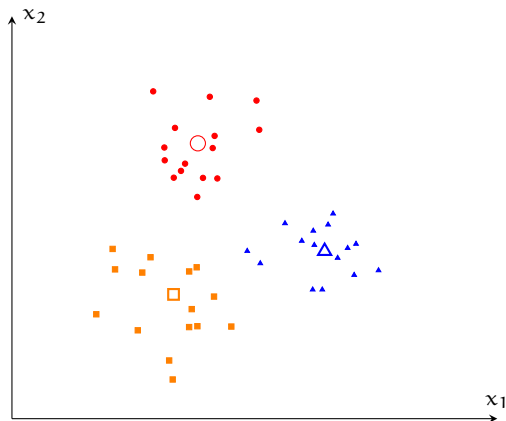


# K-means clustering: visualization



- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

# K-means clustering: visualization



- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

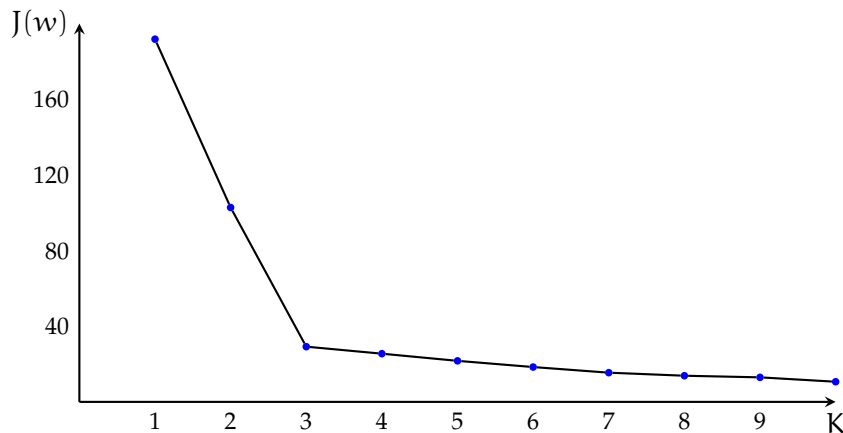
## K-means: some issues

- K-means requires the data to be in an Euclidean space
- K-means is sensitive to outliers
- The results are sensitive to initialization
  - There are some smarter ways to select initial points
  - One can do multiple initializations, and pick the best (with lowest within-group squares)
- It works well with approximately equal-size round-shaped clusters
- We need to specify number of clusters in advance

# How many clusters?

- The number of clusters is defined for some problems, e.g., classifying news into a fixed set of topics/interests
- For others, there is no clear way to select the best number of clusters
- The error (within cluster scatter) decreases with increasing number of clusters, using a test set or cross validation is not useful either
- A common approach is clustering for multiple  $K$  values, and picking where there is an 'elbow' in the graph of the error function

# How many clusters?



This plot is sometimes called a *scree plot*.

# K-medoids

- K-medoids algorithm is an alternation of K-means
- Instead of calculating centroids, we try to find most typical data point (medoids) at each iteration
- K-medoids can work with distances, does not need feature vectors to be in an Euclidean space
- It is less sensitive to outliers
- It is computationally more expensive than K-means

# Hierarchical clustering

- Instead of a flat division to clusters as in K-means, hierarchical clustering builds a hierarchy based on similarity of the data points
- There are two main 'modes of operation':

Bottom-up or *agglomerative* clustering

- starts with individual data points,
- merges the clusters until all data is in a single cluster

Top-down or *divisive* clustering

- starts with a single cluster,
- and splits until all leaves are single data points

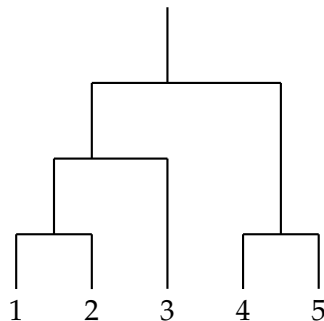
# Hierarchical clustering

- Hierarchical clustering operates on distances (or similarities)
- The result is a binary tree called *dendrogram*
- Dendrograms are easy to interpret (especially if data is hierarchical)
- The algorithm does not commit to the number of clusters  $K$  from the start, the dendrogram can be 'cut' at any height for determining the clusters

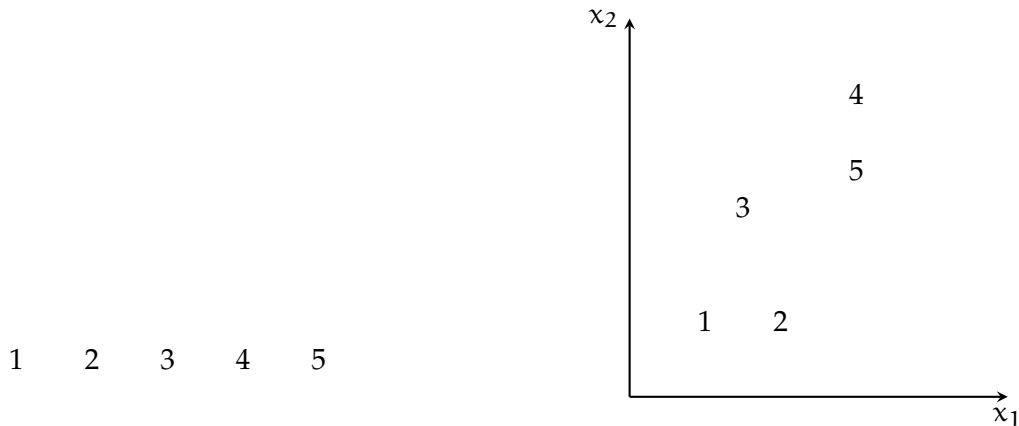


# Agglomerative clustering

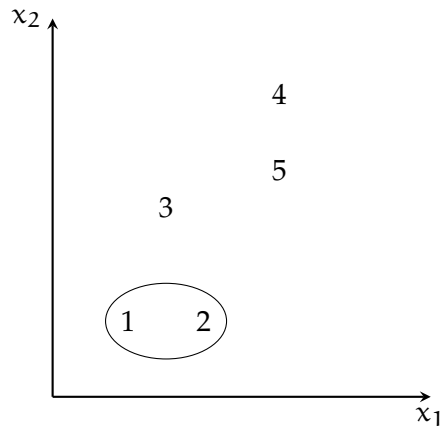
1. Compute the similarity/distance matrix
2. Assign each data point to its own cluster
3. Repeat until no clusters left to merge
  - Pick two clusters that are most similar to each other
  - Merge them into a single cluster



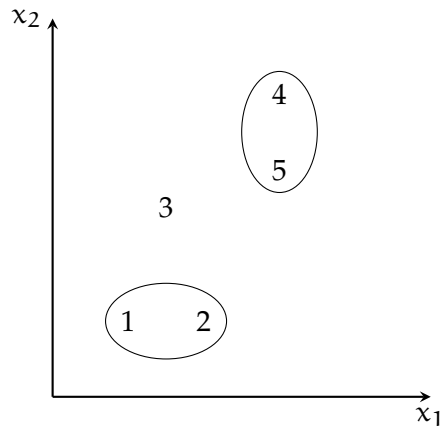
# Agglomerative clustering demonstration



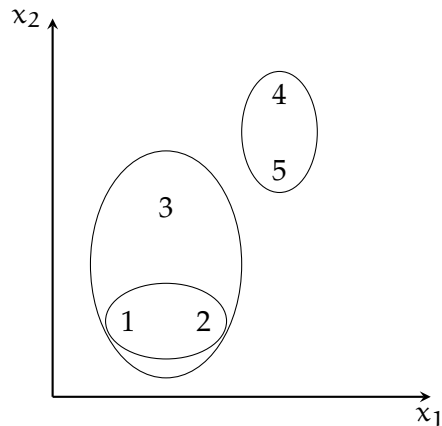
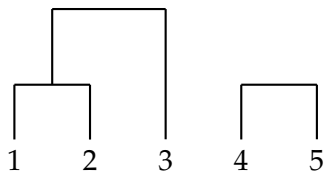
# Agglomerative clustering demonstration



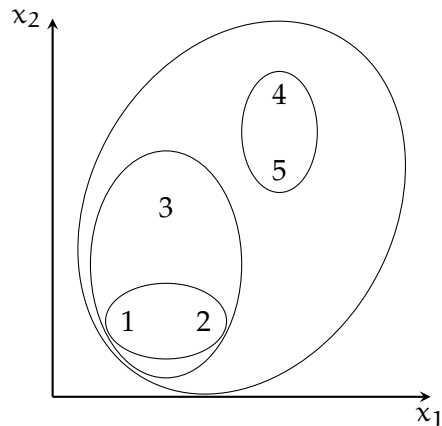
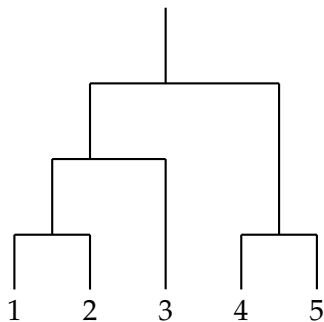
# Agglomerative clustering demonstration



# Agglomerative clustering demonstration

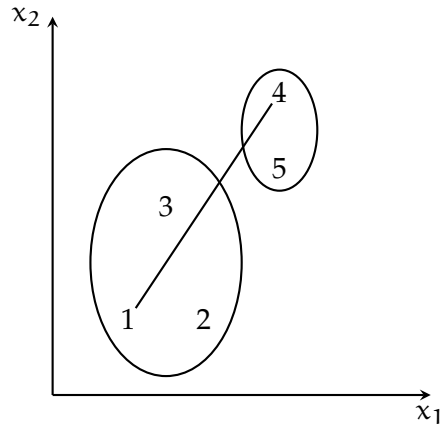


# Agglomerative clustering demonstration



# How to calculate between cluster distances

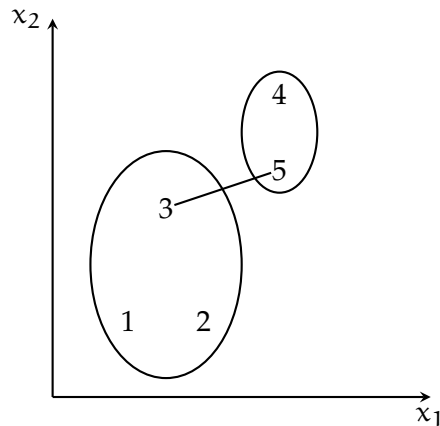
**Complete** maximal inter-cluster distance



# How to calculate between cluster distances

Complete maximal inter-cluster distance

Single minimal inter-cluster distance



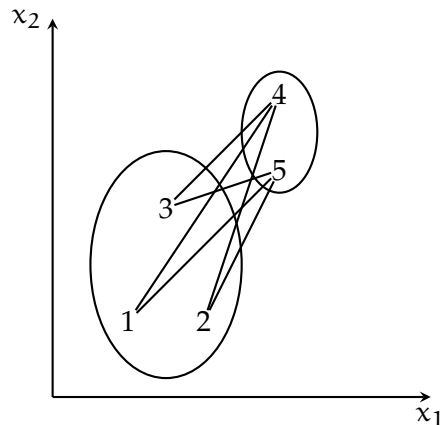


# How to calculate between cluster distances

Complete maximal inter-cluster distance

Single minimal inter-cluster distance

Average mean inter-cluster distance



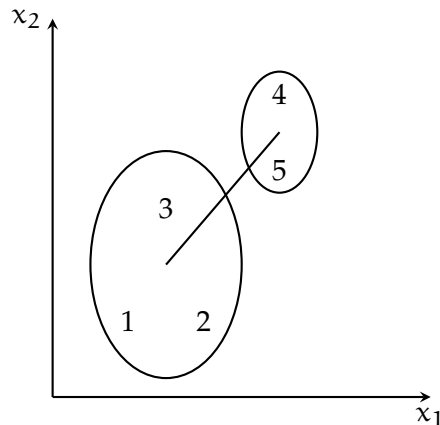
# How to calculate between cluster distances

Complete maximal inter-cluster distance

Single minimal inter-cluster distance

Average mean inter-cluster distance

**Centroid** distance between the  
centroids



Note: we only need distances, (feature) vectors are not necessary

# Clustering evaluation

Evaluating clustering results is often non-trivial

- Internal evaluation is based a metric that aims to indicate ‘good clustering’: e.g., *Dunn index*, *gap statistic*, *silhouette*
- External metrics can be useful if we have labeled *test* data: e.g., *V-measure*, *B<sup>3</sup>ed F-score*
- The results can be tested on the target application: e.g., word-clusters evaluated based on their effect on parsing accuracy
- Human judgments, manual evaluation – ‘looks good to me’

# Clustering evaluation

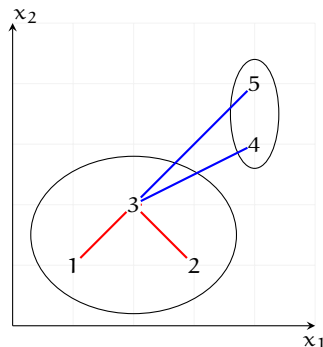
internal metric example: silhouette

$$s_i = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where

$a(i)$  average distance between object  $i$  and objects in the same cluster

$b(i)$  average distance between object  $i$  and objects in the *closest* cluster

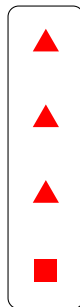


# Clustering evaluation

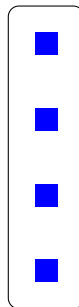
external metrics: general intuition

- We want clusters that contain members of a single gold-standard class (homogeneity)
- We want all members of a class to be in a single cluster (completeness)

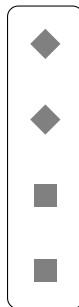
Cluster 1



Cluster 2



Cluster 3



Note the similarity with precision and recall.

## Clustering: some closing notes

- We do not have proper evaluation procedures for clustering results (for unsupervised learning in general)
- Some clustering methods are unstable, slight changes in the data or parameter choices may change the results drastically
- Approaches against instability include some validation methods, or producing 'probabilistic' dendrograms by running clustering with different options

# Density estimation

- K-means treats all data points in a cluster equally
- A 'soft' version of K-means is density estimation for Gaussian mixtures, where
  - We assume the data comes from a mixture of K Gaussian distributions
  - We try to find the parameters of each distribution (instead of centroids) that maximizes the likelihood of the data
- Unlike K-means, mixture of Gaussians assigns probabilities for each data point belonging to one of the clusters
- It is typically estimated using the expectation-maximization (EM) algorithm

# Density estimation using the EM algorithm

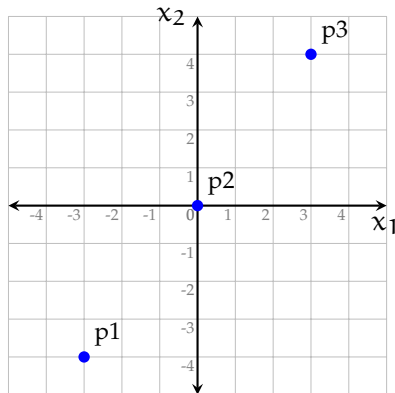
- The EM algorithm (or its variations) is used in learning models with latent/hidden variables
  - It is closely related to the K-means algorithm
1. Initialize the parameters (e.g., randomly) of K multivariate normal distributions ( $\mu, \Sigma$ )
  2. Iterate until convergence:
    - E-step Given the parameters, compute the membership 'weights', the probability of each data point belonging to each distribution
    - M-step Re-estimate the mixture density parameters using the calculated membership weights in the E-step



# Principal component Analysis

- Principal component analysis (PCA) is a method of *dimensionality reduction*
- PCA maps the original data into a lower dimensional space by a linear transformation (rotation)
- The transformed lower-dimensional variables retain most of the variation (=information) in the input
- PCA can be used for
  - visualization
  - data compression
  - reducing dimensionality of features for other machine learning methods
  - eliminating noise

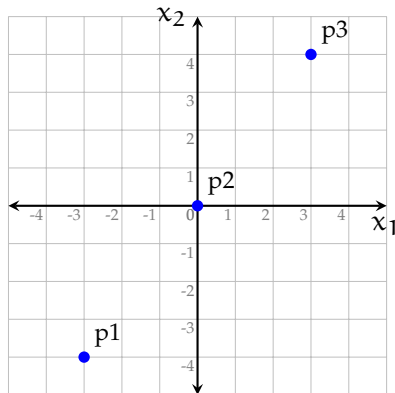
# PCA: a toy example



Questions:

- How many dimensions do we have?
- How many dimensions do we need?

# PCA: a toy example



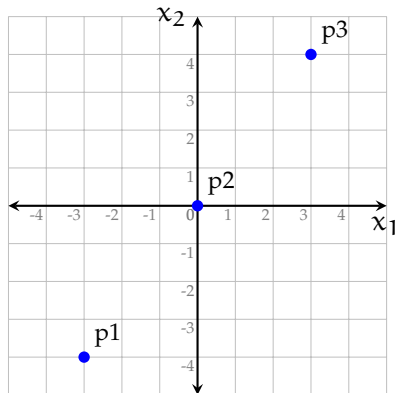
## Questions:

- How many dimensions do we have?
- How many dimensions do we need?
- Short divergence: calculate the covariance matrix

$$\Sigma = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$$

- What is the correlation between  $x_1$  and  $x_2$ ?

# PCA: a toy example



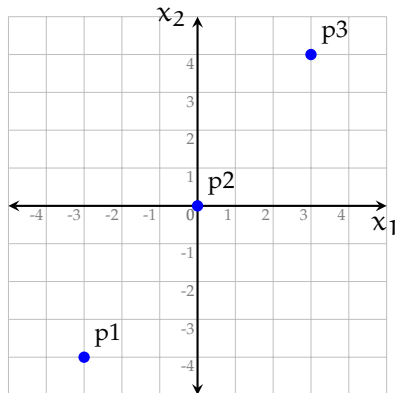
## Questions:

- How many dimensions do we have?
- How many dimensions do we need?
- Short divergence: calculate the covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_{x_1}^2 & \sigma_{x_2, x_1} \\ \sigma_{x_1, x_2} & \sigma_{x_2}^2 \end{bmatrix}$$

- What is the correlation between  $x_1$  and  $x_2$ ?

# PCA: a toy example



## Questions:

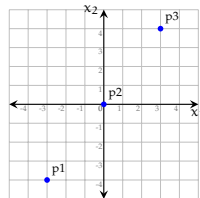
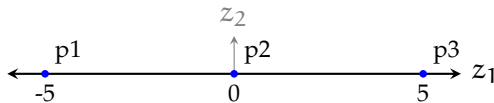
- How many dimensions do we have?
- How many dimensions do we need?
- Short divergence: calculate the covariance matrix

$$\Sigma = \begin{bmatrix} \frac{18}{3} & 8 \\ 8 & \frac{32}{3} \end{bmatrix}$$

- What is the correlation between  $x_1$  and  $x_2$ ?

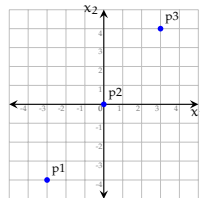
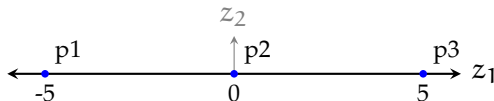
## PCA: A toy example (2)

What if we reduce the data to:



## PCA: A toy example (2)

What if we reduce the data to:

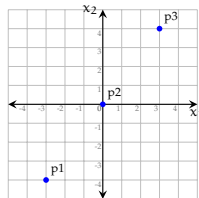
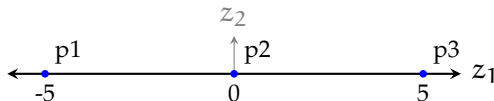


Going back to the original coordinates is easy, rotate using:

$$A = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} = \begin{bmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{bmatrix}$$

## PCA: A toy example (2)

What if we reduce the data to:



Going back to the original coordinates is easy, rotate using:

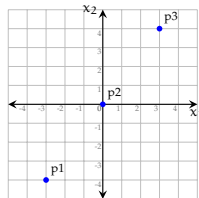
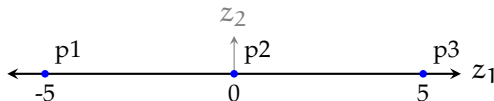
$$A = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} = \begin{bmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{bmatrix}$$

$$p_1 = A \times \begin{bmatrix} -5 \\ 0 \end{bmatrix} = \begin{bmatrix} -3 \\ -4 \end{bmatrix} \quad p_2 = A \times \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad p_3 = A \times \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$



## PCA: A toy example (2)

What if we reduce the data to:



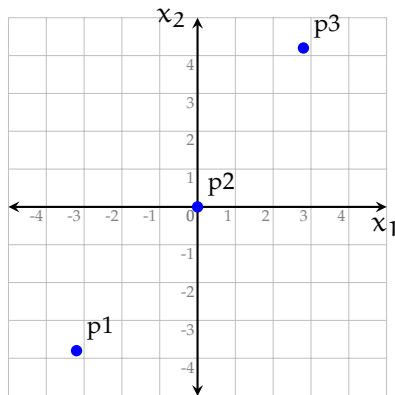
Going back to the original coordinates is easy, rotate using:

$$A = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} = \begin{bmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{bmatrix}$$

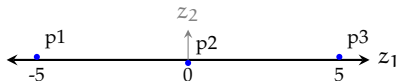
$$p1 = A \times \begin{bmatrix} -5 \\ 0 \end{bmatrix} = \begin{bmatrix} -3 \\ -4 \end{bmatrix} \quad p2 = A \times \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad p3 = A \times \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

We can recover the original points perfectly. In this example the inherent dimensionality of the data is only 1.

## PCA: A toy example (3)



- What if the variables were not perfectly but strongly correlated?
- We could still do a similar transformation:



- Discarding  $z_2$  results in a small reconstruction error:

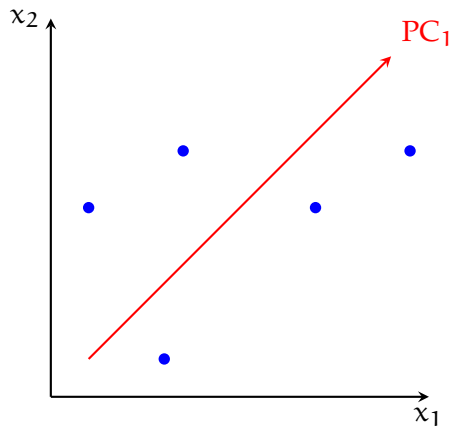
$$p1 = A \times \begin{bmatrix} -5 \\ 0 \end{bmatrix} = \begin{bmatrix} -3 \\ -4 \end{bmatrix}$$

- Note:  $z_1$  (also  $z_2$ ) is a linear combination of original variables

# Why do we want to reduce the dimensionality

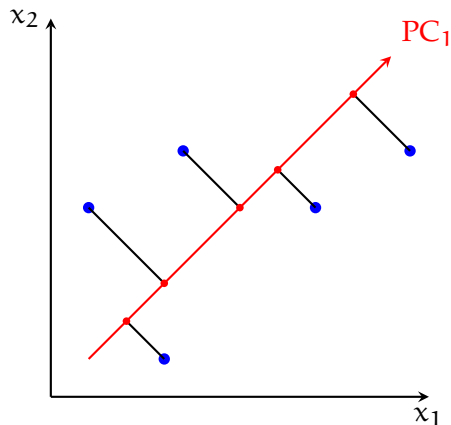
- Visualizing high-dimensional data becomes possible
- If we use the data for other ML methods,
  - we reduce the computation time
  - we may avoid ‘the curse of dimensionality’
- Decorrelation is useful in some applications
- We compress the data (in a lossy way)
- We eliminate noise (assuming a high signal to noise ratio)

# Different views on PCA



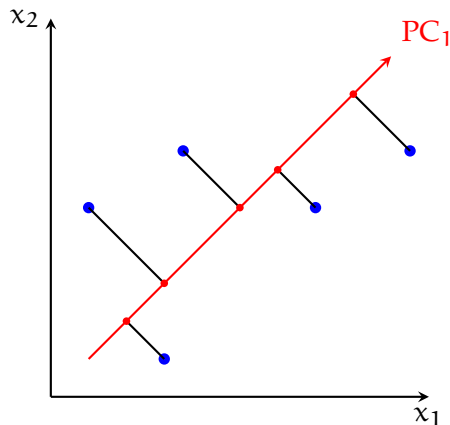
- Find the direction of the largest variance

# Different views on PCA



- Find the direction of the largest variance
- Find the projection with the least reconstruction error

# Different views on PCA



- Find the direction of the largest variance
- Find the projection with the least reconstruction error
- Find a lower dimensional latent Gaussian variable such that the observed variable is a mapping of the latent variable to a higher dimensional space (with added noise)

## How to find PCs

- When viewed as *maximizing variance* or *reducing the reconstruction error*, we can write the appropriate objective function and find the vectors that minimize it
- In latent variable interpretation, we can use EM as in estimating mixtures of Gaussians
- The principal components are the eigenvectors of the correlation matrix, where large eigenvalues correspond to components with large variation
- A numerically stable way to obtain principal components is doing *singular value decomposition* (SVD) on the input data

## PCA as matrix factorization (eigenvalue decomposition)

- One can compute PCA by decomposing the covariance matrix as (note  $\Sigma = \mathbf{X}^T \mathbf{X}$ )

$$\Sigma = \mathbf{U} \Lambda \mathbf{U}^T$$

- the columns of  $\mathbf{U}$  are the principal components (eigenvectors)
  - $\Lambda$  is a diagonal matrix of eigenvalues
- Another option is SVD, which factorizes the input vector ( $k$  variables  $\times$   $n$  data points) as

$$\mathbf{X} = \mathbf{U} \mathbf{D} \mathbf{V}^*$$

- $\mathbf{U}$  ( $k \times k$ ) contains the eigenvectors as before,
  - $\mathbf{D}$  ( $k \times n$ ) diagonal matrix  $\mathbf{D}^2 = \Lambda$
  - $\mathbf{V}^*$  is a  $n \times n$  unitary matrix

\* The above is correct for centered variables, otherwise the formulas get slightly more complicated.



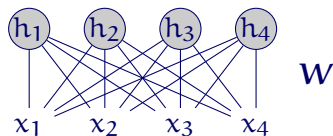
## Some practical notes on PCA

- Variables need to be centered
- Scales of the variables matter, standardizing may be a good idea depending on the units/scales of the individual variables
- The sign/direction of the principal component (vector) is not important
- If there are more variables than the data points, we can still calculate the principal components, but there will be at most  $n - 1$  PCs
- PCA will be successful if variables are correlated, there are extensions for dealing with nonlinearities (e.g., kernel PCA, ICA, t-SNE)

# Unsupervised learning in ANNs

- *Restricted Boltzmann machines* (RBM)  
similar to the latent variable models (e.g., Gaussian mixtures), consider the representation learned by hidden layers as hidden variables ( $\mathbf{h}$ ), and learn  $p(\mathbf{x}, \mathbf{h})$  that maximize the probability of the (unlabeled) data
- *Autoencoders*  
train a constrained feed-forward network to predict its output

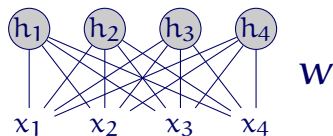
# Restricted Boltzmann machines (RBMs)



- RBMs are unsupervised latent variable models, they learn only from unlabeled data
- They are generative models of the joint probability  $p(\mathbf{h}, \mathbf{x})$
- They correspond to undirected graphical models
- No links within layers
- The aim is to learn useful features ( $\mathbf{h}$ )

\*Biases are omitted in the diagrams and the formulas for simplicity.

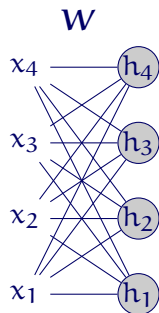
# Restricted Boltzmann machines (RBMs)



- RBMs are unsupervised latent variable models, they learn only from unlabeled data
- They are generative models of the joint probability  $p(\mathbf{h}, \mathbf{x})$
- They correspond to undirected graphical models
- No links within layers
- The aim is to learn useful features ( $\mathbf{h}$ )

\*Biases are omitted in the diagrams and the formulas for simplicity.

# The distribution defined by RBMs



$$p(\mathbf{h}, \mathbf{x}) = \frac{e^{\mathbf{h}^T \mathbf{W} \mathbf{x}}}{Z}$$

This calculation is intractable ( $Z$  is difficult to calculate).  
But conditional distributions are easy to calculate

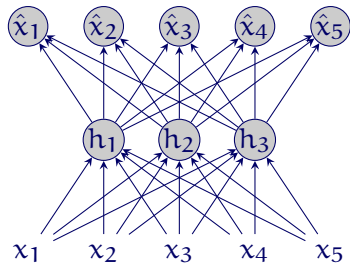
$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x}) = \frac{1}{1 + e^{\mathbf{W}_j \mathbf{x}}}$$

$$p(\mathbf{x}|\mathbf{h}) = \prod_k p(x_k|\mathbf{h}) = \frac{1}{1 + e^{\mathbf{W}_k^T \mathbf{h}}}$$

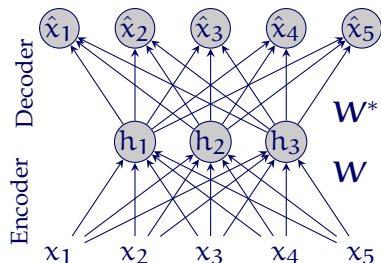
# Learning in RBMs

- We want to maximize the probability the model assigns to the input,  $p(x)$ , or equivalently minimize  $-\log p(x)$
- In general, this is computationally expensive
- *Contrastive divergence algorithm* is a well known algorithm that efficiently finds an approximate solution

# Autoencoders



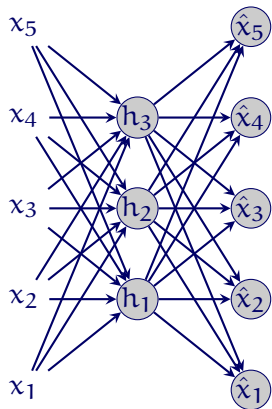
# Autoencoders



- Autoencoders are standard feed-forward networks
- The main difference is that they are trained to predict their input (they try to learn the identity function)
- The aim is to learn useful representations of input at the hidden layer
- The weights are often shared/tied ( $W^* = W^T$ )

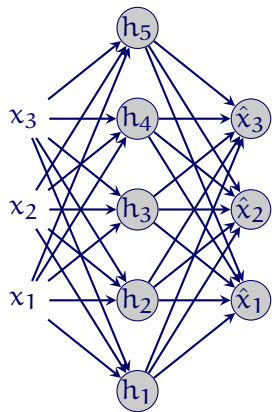


# Under-complete autoencoders



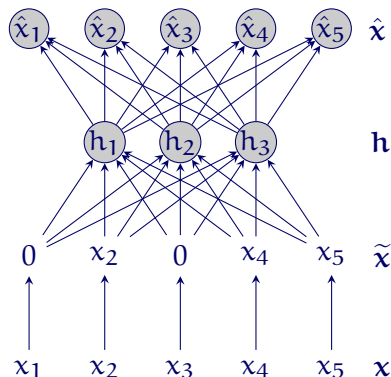
- An autoencoder is said to be *under-complete* if there are fewer hidden units than inputs
- The network is forced to learn a compact representation of the input (compress)
- An autoencoder with a single hidden layer approximates the PCA
- We need multiple layers for learning non-linear features

# Over-complete autoencoders



- An autoencoder is said to be *over-complete* if there are more hidden units than inputs
- The network can normally memorize the input perfectly
- This type of networks are useful if trained with a regularization term resulting in sparse hidden units (e.g., L1 regularization)

# Denoising autoencoders



- Instead of providing the exact input, we introduce noise by
  - randomly setting some inputs to 0 (dropout)
  - adding random (Gaussian) noise
- Network is still expected to reconstruct the original input (without noise)

# Unsupervised pre-training

- A common use case for RBMs and autoencoders are as pre-training methods for supervised networks
- Autoencoders or RBMs are trained using unlabeled data
- The weights learned during the unsupervised learning is used for initializing the weights of a supervised network
- This approach has been one of the reasons for success of deep networks

# Summary

- In unsupervised learning, we do not have labels. Our aim is to find/exploit (latent) structure in the data
- Unsupervised methods try to discover 'hidden' structure in the data
  - Clustering finds groups in the data
  - Density estimation estimates parameters of latent probability distributions
  - Dimensionality reduction transforms the data in a low dimensional space while keeping most of the information in the original data

# Summary

- In unsupervised learning, we do not have labels. Our aim is to find/exploit (latent) structure in the data
- Unsupervised methods try to discover 'hidden' structure in the data
  - Clustering finds groups in the data
  - Density estimation estimates parameters of latent probability distributions
  - Dimensionality reduction transforms the data in a low dimensional space while keeping most of the information in the original data

After the break:

- More ML: sequence learning
- Common CL tasks: tokenization, morphology, syntactic parsing, (lexical) semantics,
- Some NLP applications: text classification, and maybe more

# Derivation of PCA by maximizing the variance

- We focus on the first PC ( $z_1$ ), which maximizes the variance of the data onto itself
- We are interested only on the direction, so we choose  $z_1$  to be a unit vector ( $\|z_1\| = 1$ )
- Remember that to project a vector onto another, we simply use dot product, So the projected data points are  $z_1^T x_i$  for  $i = 1, \dots, N$ .
- The variance of the projected data points (that we want to maximize) is,

$$\sigma_{z_1}^2 = \frac{1}{N} \sum_i^N (z_1^T x_i - z_1^T \bar{x})^2 = z_1^T \Sigma z_1$$

where  $\Sigma_x$  is the covariance matrix of the unprojected data

## Derivation of PCA by maximizing the variance (cont.)

- The problem becomes maximize

$$z_1^T \Sigma z$$

with the constraint  $\|z_1\| = z_1^T z_1 = 1$

- Turning it into a unconstrained optimization problem with Lagrange multipliers, we minimize

$$z_1^T \Sigma z + \lambda_1 (1 - z_1^T z_1)$$

- Taking the derivative and setting it to 0 gives us

$$\Sigma z_1 = \lambda_1 z_1$$

Note: by definition,  $z_1$  is an eigenvector of  $\Sigma$ , and  $\lambda_1$  is the corresponding eigenvalue

- $z_1$  is the first principal component, we can now compute the second principal component with the constraint that it has to be orthogonal to the first one











