Large Scale Learning and Neural Networks

Week 9

Daniele Bianchi[1]

whitesphd.com

[1] School of Economics and Finance
Queen Mary, University of London

Summary

This week we discuss the use of machine learning methods for large dimensional data as well as highly non-linear neural network models.

Contents

Working with large-scale data

Working with large-scale data

- Increasingly, data is no longer on daily / monthly / quarterly format.

- Alternative data sources are non-homogeneous, (non-numeric, which we will cover later today or in the next session), and are very large databases.

- High-dimensional. For example, take a standard computer monitor with a resolution of 1920 x 1080.

- 2073600 pixels. If we restrict the colours to be just white or black for each pixel, we have $2^{2073600}$ potential images.

- Add one more dimension: Each pixel can do 2^{24} potential colours (three, 8-bit channels for Red, Green and Blue).

Working with large-scale data

- Searching through such datasets for outcomes is an enormous task.
- Key question: Is there a way to find lower-dimensional subspaces to work with?
- Here is where large scale learning makes use of unsupervised learning.
- Goal: Reduce dimensionality, but retain the "signal" in the data.
- Finance data is very high-dimensional in almost all sub-fields. Unsupervised learning increasingly relevant in all aspects of finance.

Supervised Learning

- So far, we have only looked at methods under the broad umbrella of "Supervised Learning" methods.
- Broadly, all regression and classification tools fall under this category.
- In this set up, we have a set of predictors $X_1, X_2, X_3 ... X_p$, measured for N observations.
- We aim to predict Y – the response that we also observe – using these predictors.
- All our examples such as predicting default, returns, and other financial variables use supervised learning tools.

Unsupervised Learning

- In "Unsupervised Learning", we do not have an associated "response variable", Y.
- The goal with unsupervised learning is to find patterns in the data that discovers subgroups and other ways in which data may be "clustered".
- Unlike supervised learning, unsupervised learning is a lot more subjective.
- You can consider unsupervised learning as a way to explore data before using supervised learning methods.
- Particularly useful when you have very large datasets.

Unsupervised Learning

- A big challenge: There is no universally accepted way to validate the results from using an unsupervised learning method.
- In other words, we *do not know the true answer*.
- For example, unsupervised learning is potentially a useful tool with COVID-19 pandemic in today's world to understand whom it affects before asking *why*.
- We have briefly looked at one such approached under "Dimension reduction methods" early on in the course – Principal Component Analysis.
- We now focus on another broad class of methods for discovering unknown subgroups: Clustering.

Clustering

Clustering Methods

- Clustering refers to the process of dividing observations in a dataset into distinct groups.
- Two properties:
  1. Similarity within groups: Observations in each group are similar to each other.
  2. Distinct across groups: Observations in different groups are different from each other.
- This is an "unsupervised" problem because we are discovering patterns in the data.
- Two approaches we will focus on: $K-$means clustering, and hierarchical clustering.

$K-$means clustering

- Goal: Partition a dataset into K distinct, *non-overlapping* clusters.
- Before using a procedure, an analyst has to decide how many clusters she wants.
- Intuitive mathematical problem:
- Let $C_1, ..., C_K$ represent the different clusters of data. All observations $1, ..., n$ belong to one or the other set. No set contains an observation that also belongs to another set.
- Mathematically,
  1. $C_1 \cup C_2 \cup ... \cup C_K = 1, ..., n$
  2. $C_k \cap C_{k'} = \emptyset$
- Lastly, a property of a good $K-$means clustering procedure is one that has low *within* cluster variation.

Minimal Within-Cluster Variation

- Let $W(C_k)$ measure the within-cluster variation in the cluster C_k.
- Mathematically, we therefore want to solve the following problem:

$$\min_{C_1,\ldots,C_K} \left(\sum_{k=1}^{K} W(C_k) \right)$$

- We want to partition the observations into K clusters such that the total within-cluster variation is as small as possible.

Measures of Within-Cluster Variation

- Most common choice: *squared Euclidean Distance*.

$$W(C_k) = \frac{1}{N_k} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

- N_k is the number of observations in cluster C_k. In this set up, the optimization problem will therefore be:

$$\min_{C_1,\ldots,C_K} \left(\sum_{k=1}^{K} \frac{1}{N_k} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right)$$

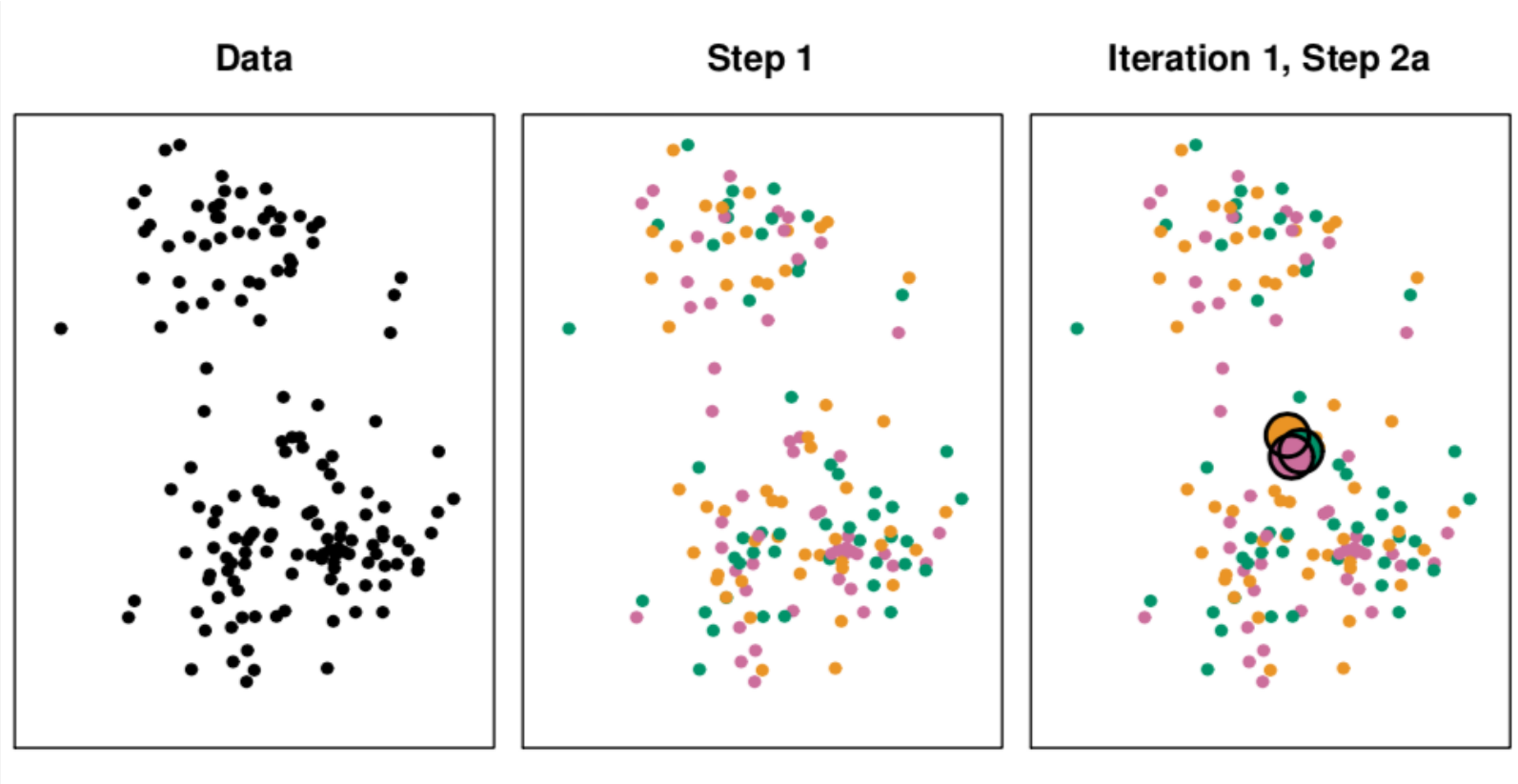
Measures of Within-Cluster Variation

- The computation challenge: We need an algorithm that can solve this equation. There are almost K^n ways to partition all N observations into K clusters!

- Fortunately there is a simple algorithm that provides a *local optimum* to this problem.

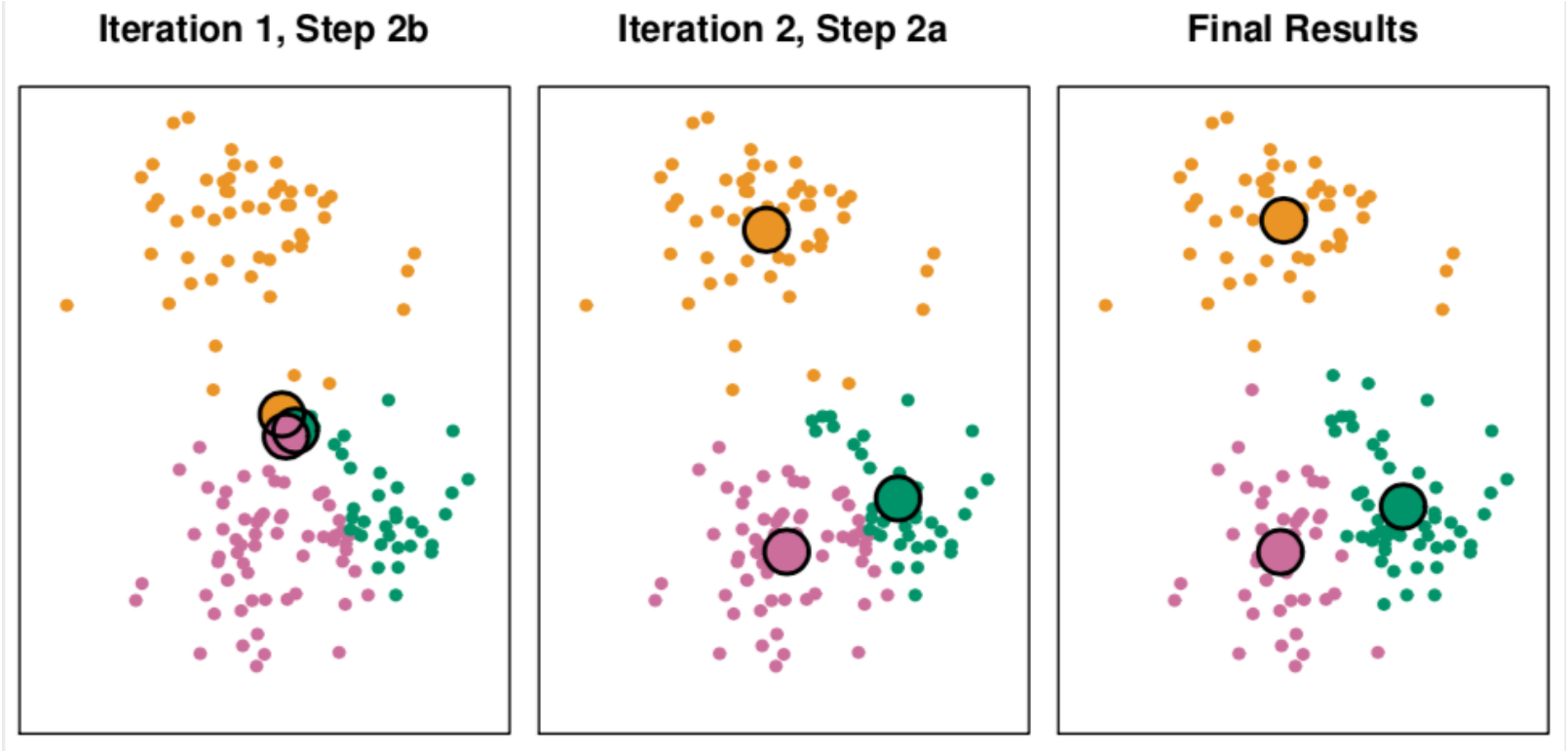
- Algorithm: $K-$**means clustering**
  1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.
  2. Iterate until the cluster assignments stop changing:
     - 2.1 For each of the K clusters, compute the cluster centroid. The k^{th} cluster centroid is the vector of the p feature means for the observations in the kth cluster.
     - 2.2 Assign each observation to the cluster whose centroid is closest (where closest is defined using Euclidean distance).

An Example:



Source: "An introduction to Statistical Learning", James et al. (2013).

An Example:



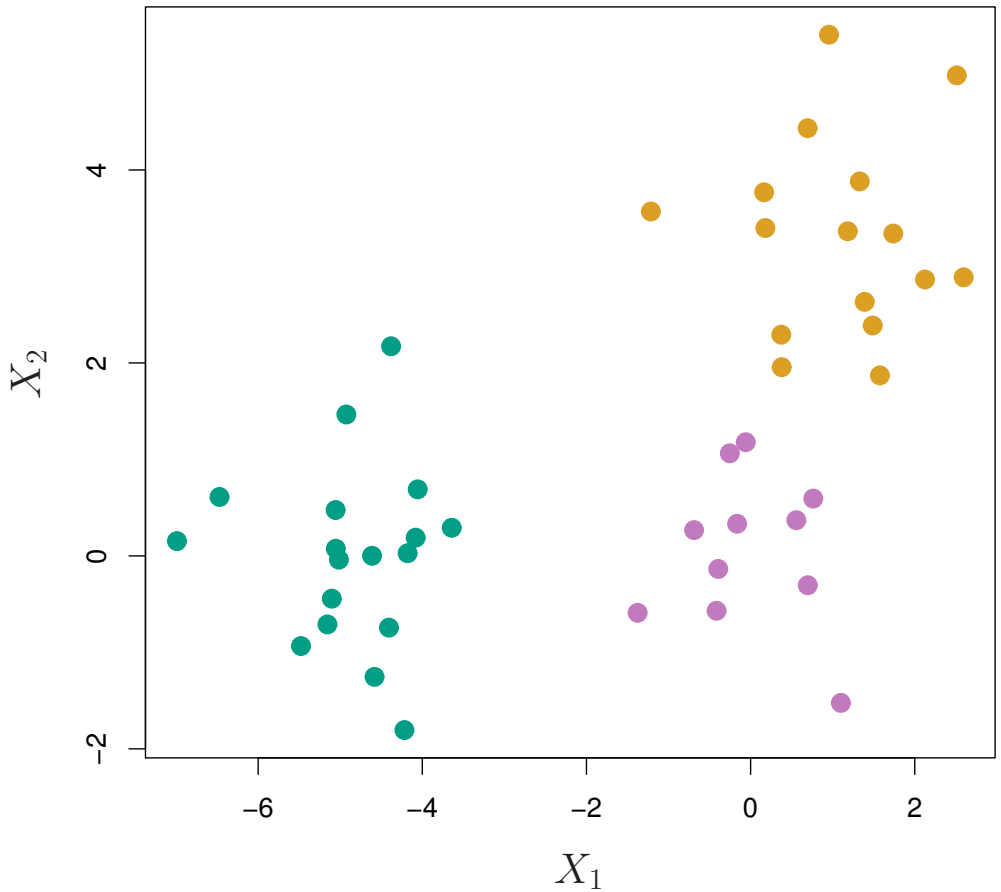
Source: "An introduction to Statistical Learning", James et al. (2013).

Limitations

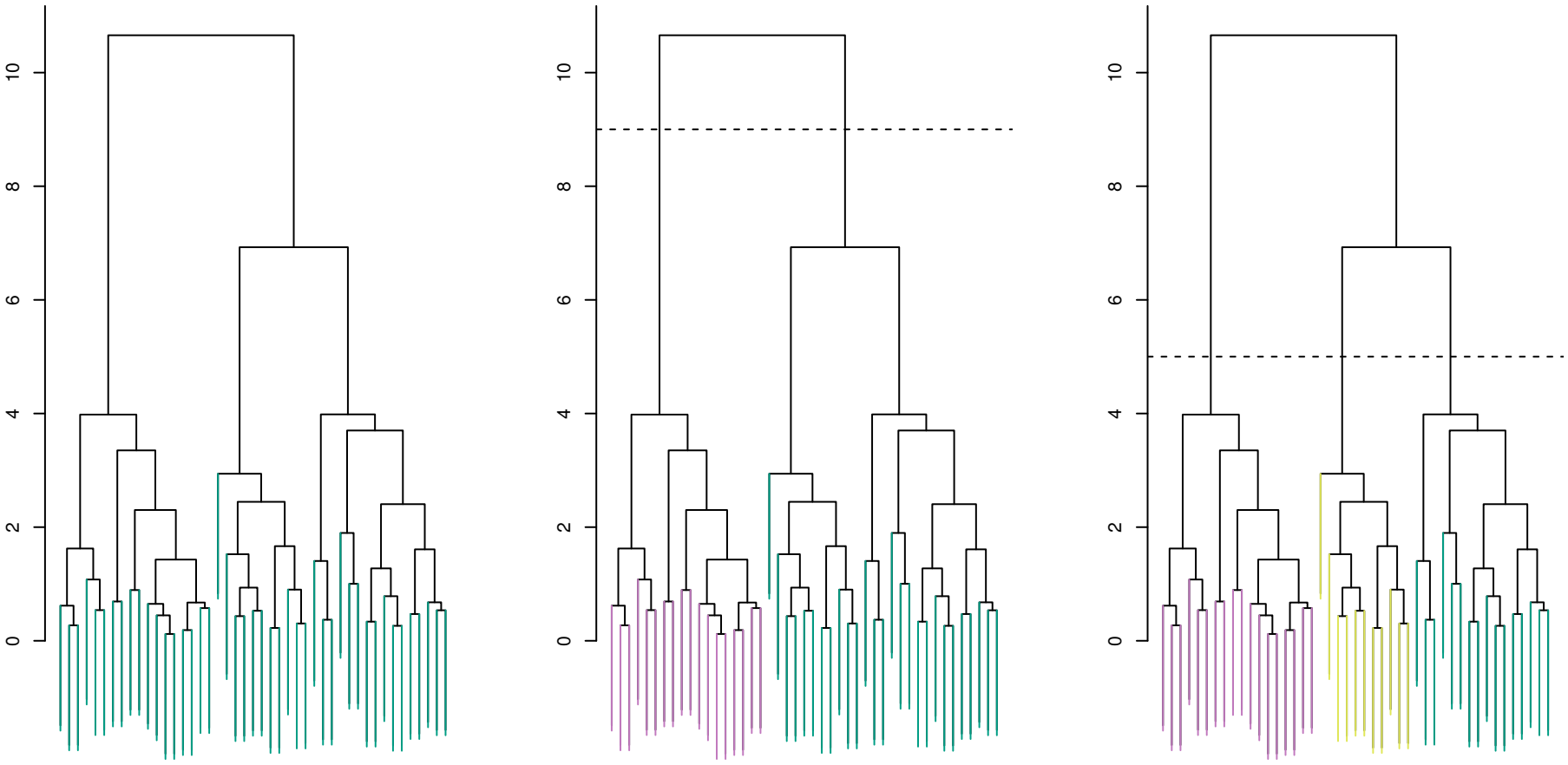
- Two disadvantages of $K-$means clustering are that you have to pre-specify K, and the classification may change depending on the initial configuration!
- Solution to the second problem: Run several $K-$means tests with different initial configurations. Then select the best solution, i.e., that for which the objective function is smallest.
- Solution to the first problem: Hierarchical Clustering.

Hierarchical Clustering

- Bottom-up or agglomerative clustering: Most common type of clustering.
- Can be represented as "Dendrograms" (upside down trees).
- The data:

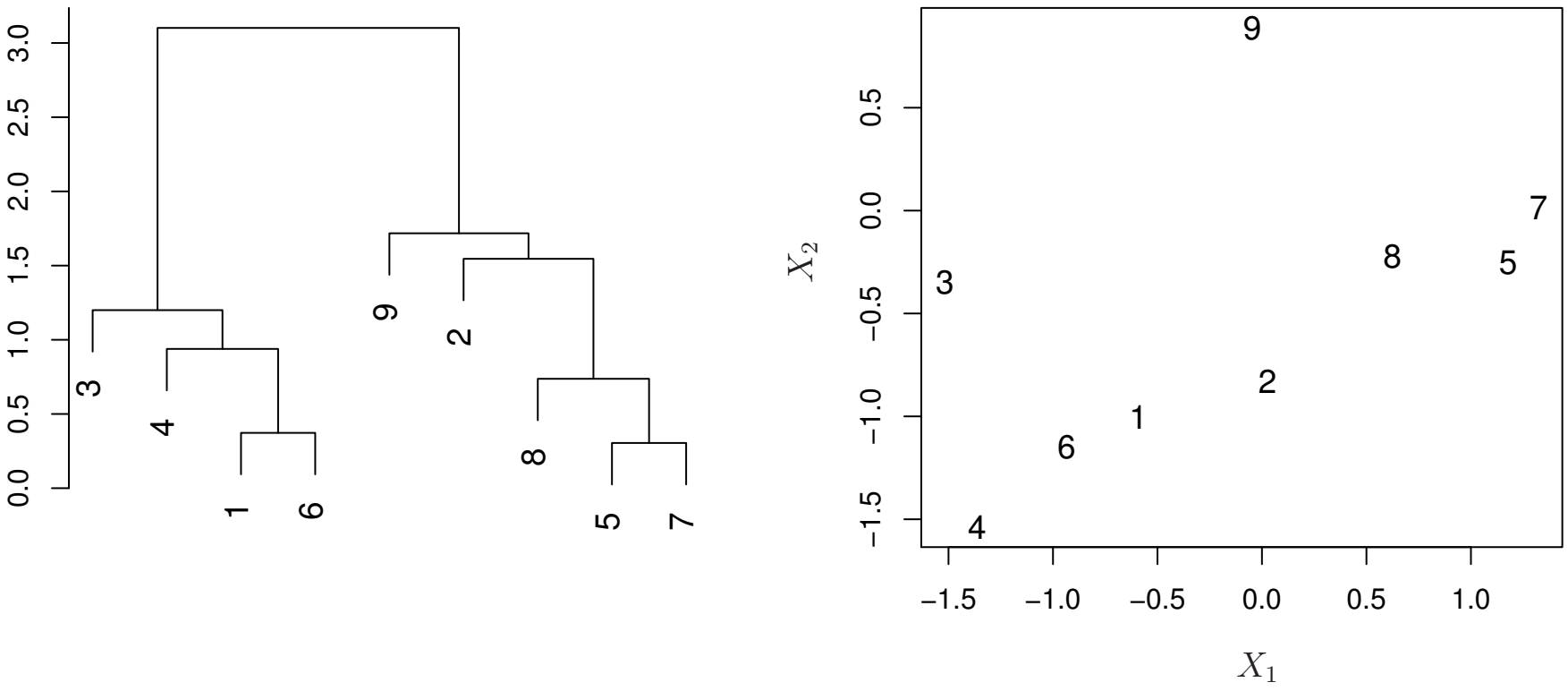


Interpreting a Dendrogram



Source: "An introduction to Statistical Learning", James et al. (2013).

Interpreting a Dendrogram



Source: "An introduction to Statistical Learning", James et al. (2013).

How does this help select clusters?

- "Hierarchical" refers to the fact that you can obtain clusters by cutting the dendrogram at different heights.
- One single dendrogram can be used to obtain any number of clusters.
- Here, the work is more an "art" than a "science". Often the choice of where to cut the dendrogram is not so clear.

Hierarchical Clustering Algorithm

- Core concept: Measure dissimilarity between each pair of observations.
- Algorithm:
  1. Start at bottom of dendrogram: Each observation is its own cluster.
  2. Two observations most similar to each other fused. Result: $n - 1$ clusters.
  3. Next two clusters that are most similar to each other are fused again...
  4. Stop when you have fused all observations into one single cluster.

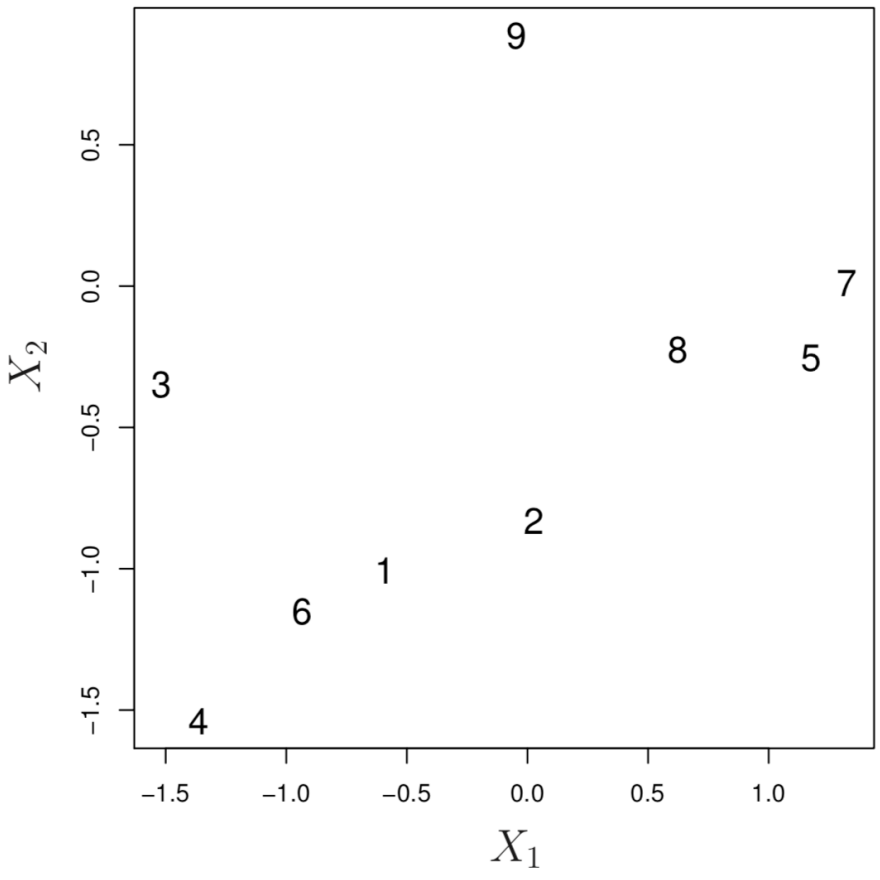
Challenge

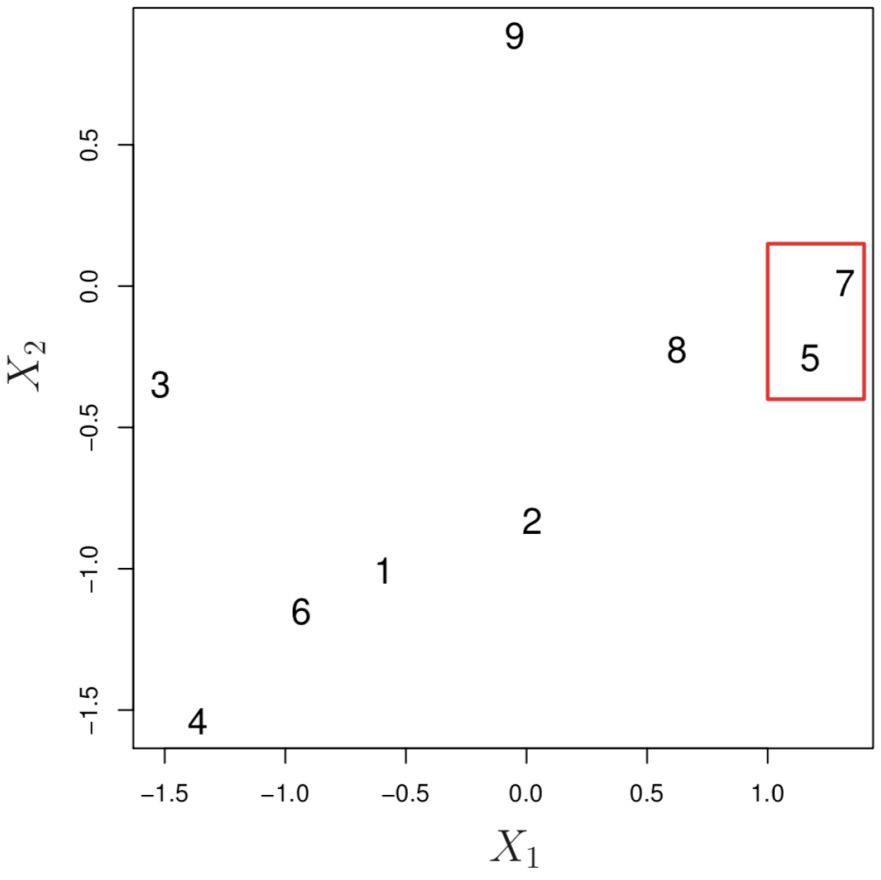
- Easy to measure dissimilarity between pairs. How about dissimilarity between two clusters with multiple observations?
- *Linkage* refers to the concept of dissimilarity extended to a pair of groups of observations.
- Four common types: Complete, Average, Single and Centroid.
- Most popular: Complete and Average.

Linkage in Hierarchical Clustering Algorithm

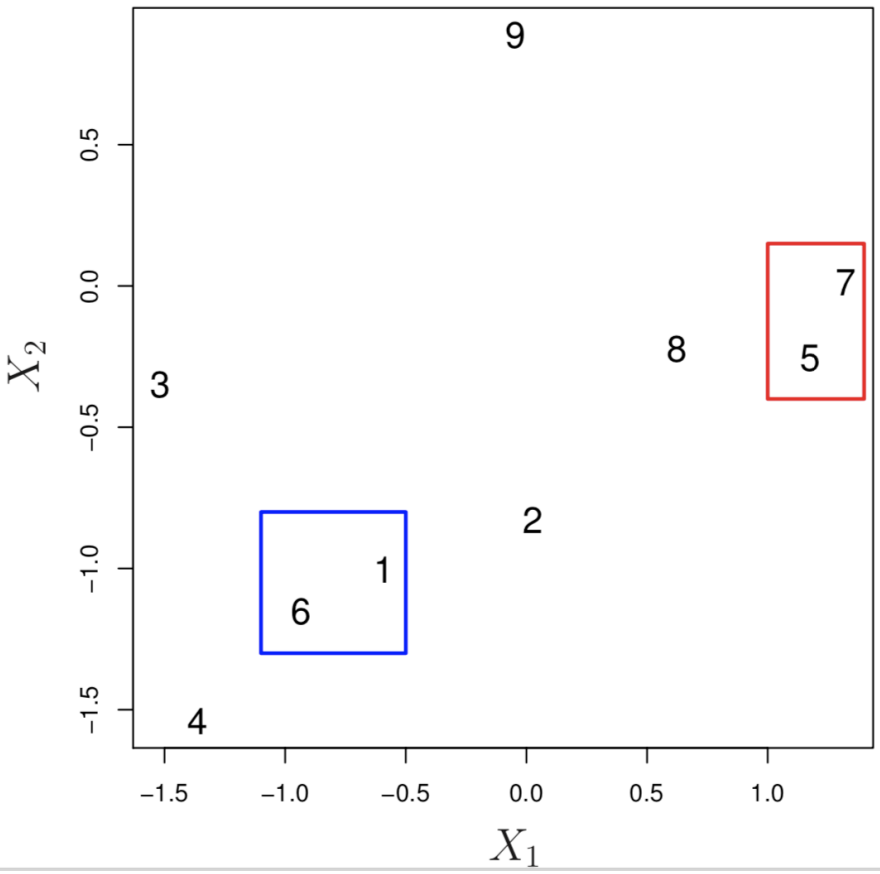
Linkage	Explanation
Complete	Compute all pairwise dissimilarities between observations across two clusters. Record the largest of these dissimilarities.
Single	Compute all pairwise dissimilarities between observations across two clusters. Record the smallest of these dissimilarities.
Average	Compute all pairwise dissimilarities between observations across two clusters. Record the average of these dissimilarities.
Centroid	Dissimilarity between the centroid for two clusters (mean vector of all variables).

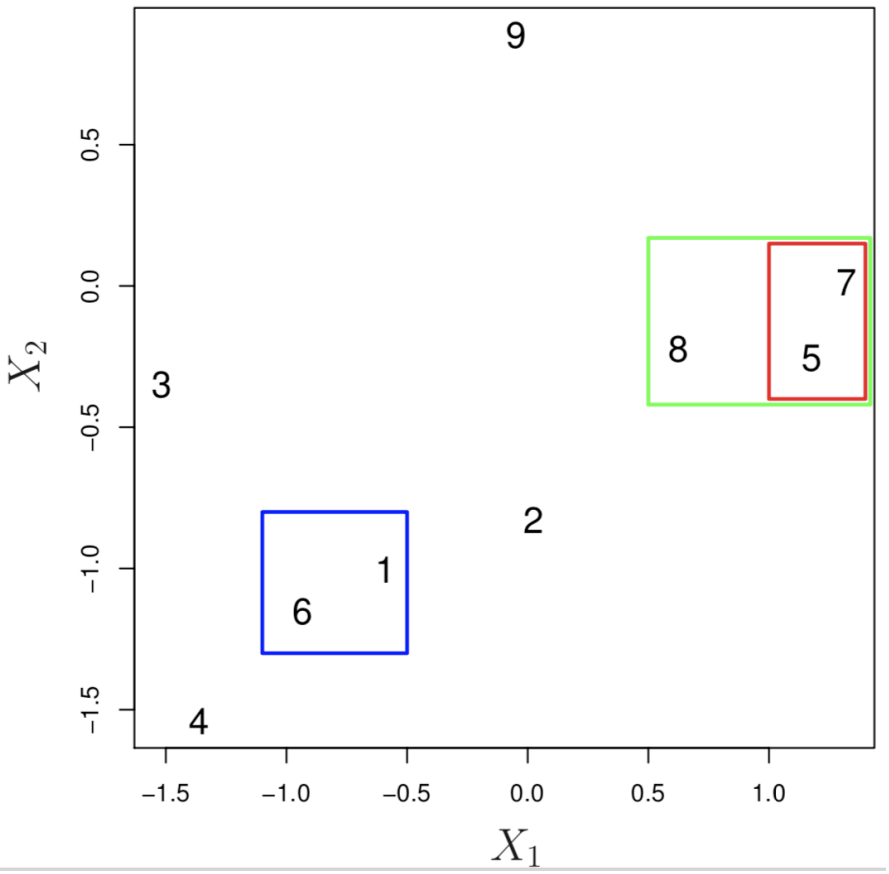
An Illustration



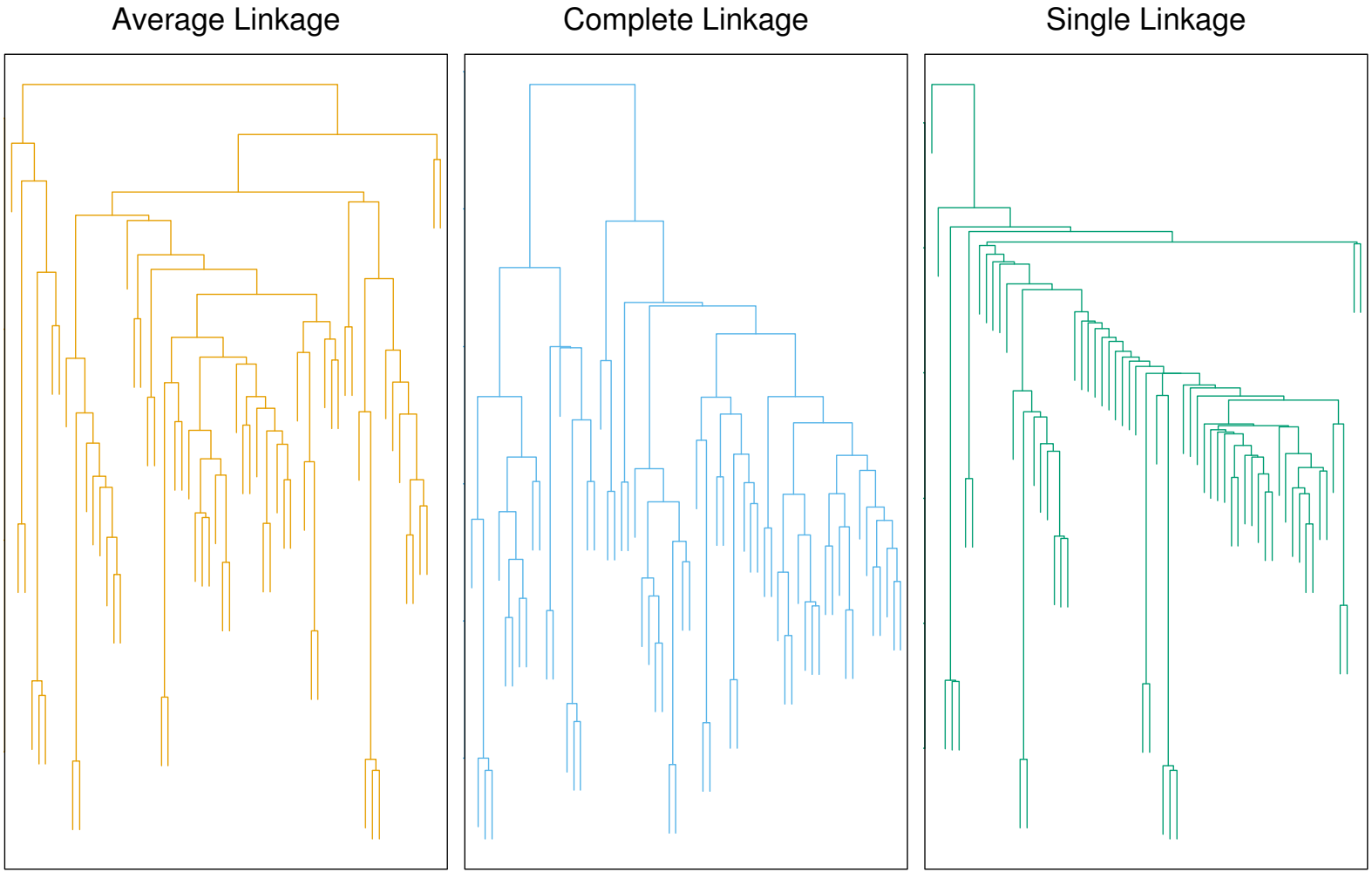


An Illustration





Linkage Matters



Source: "An introduction to Statistical Learning", James et al. (2013).

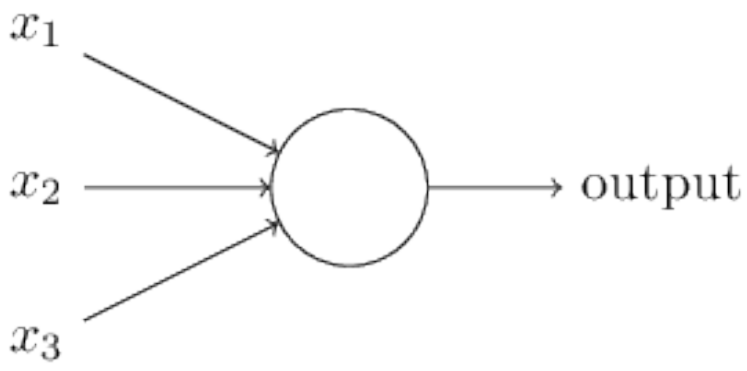
Clustering: Approach with Caution

- Clustering is a very useful tool to approach large data.
- Number of issues that need to be kept in mind:
  1. Standardize observations? For e.g: mean 0, standard deviation of one.
  2. How many clusters K in $K-$means clustering?
  3. If hierarchical, what dissimilarity measure should be used? What type of linkage should be used? Where to draw the line to obtain clusters?
- All very tricky choices as the output is very sensitive to choices made.

Introduction to Neural Networks

Perceptrons

- Consider a simple machine. Converts binary inputs to outputs:



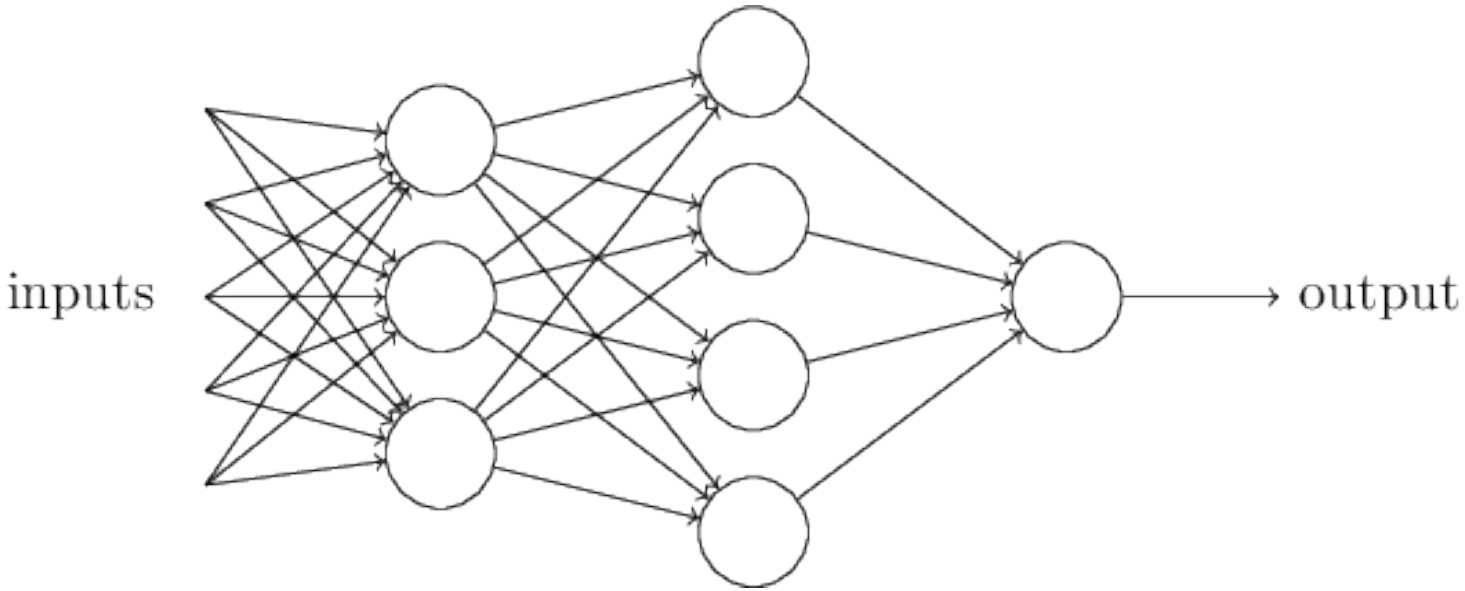
- In the example above, this perceptron has three inputs, but there could be more, or less.

- The output can be computed using a simple weighted sum threshold rule ($w =$ weights and b is bias or threshold):

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j + b \leq 0 \\ 1 & \text{if } \sum_j w_j x_j + b > 0 \end{cases}$$

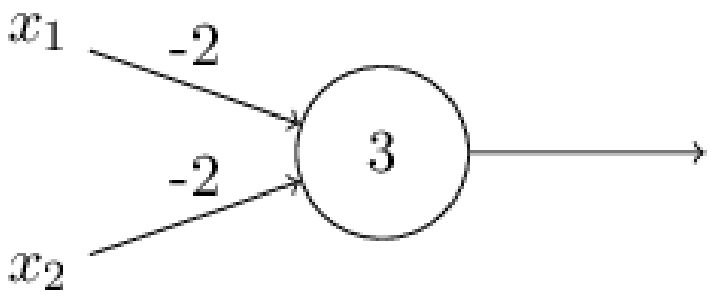
Perceptrons

- This is a very simple concept, but we can build from here



- In the above diagram, perceptrons at every layer are making decisions using inputs from the previous layer.
  - Thus, decisions are using more complex inputs.
  - All perceptrons still have single outputs, but those outputs can be fed as inputs to other perceptrons in deeper layers.

Perceptrons



- In the above diagram, the perceptron has two inputs, each with weight -2, and an overall bias of 3.
  - Input $0, 0$ produces output 1, since $\sum_j w_j x_j + b = 0 + 3 > 0$.
  - Inputs $0, 1$, or $1, 0$ also produce output 1, since $\sum_j w_j x_j + b = -2 + 3 = 1 > 0$.
  - Input $1, 1$ produces output 0, since $\sum_j w_j x_j + b = -4 + 3 = -1 < 0$.
- If you are familiar with Boolean operators, this is known as a NAND gate.

Perceptrons and NAND gates

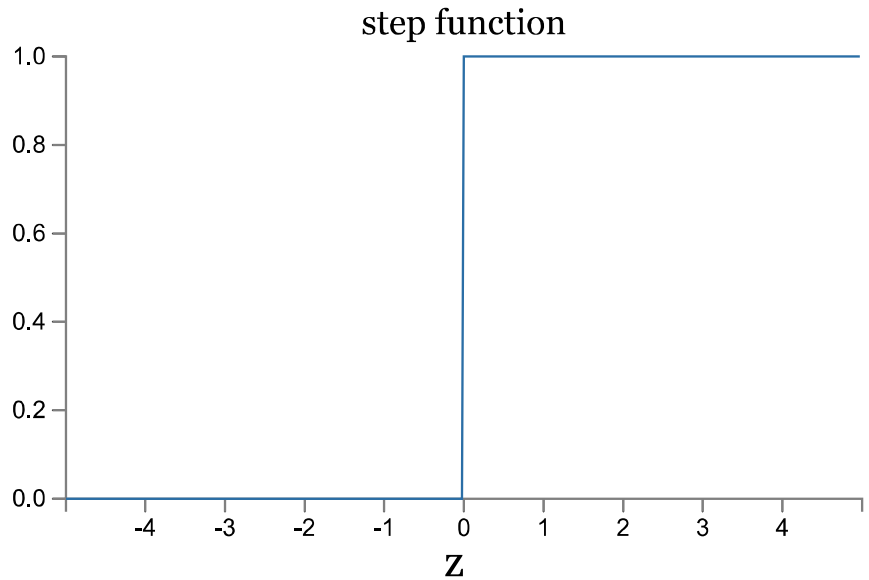
- Why is it important that we can build a simple NAND gate using a perceptron?
- In electronic engineering, these so-called logic gates are the building blocks of digital circuit design, and hence of all computer hardware.
- It is easy to prove that the NAND gate is functionally complete or universal for computation, i.e., that all logic systems/computational operations can be implemented using NAND gates.
- Therefore, these same properties carry over to perceptrons, since they can be used to implement NAND gates.

Perceptrons and NAND gates

- Of course, just being able to implement NAND gates on their own would make perceptrons interesting, but not wildly exciting.
- They would then be just a different way to implement digital circuits.
- However, perceptrons have greater potential than this. The weights and biases can actually be treated as tuning parameters, allowing perceptrons (and ultimately networks of such perceptrons or other types of neurons) to learn.
- This means that neural networks could, in theory, learn to solve difficult problems.

Perceptrons and Sigmoid Neurons

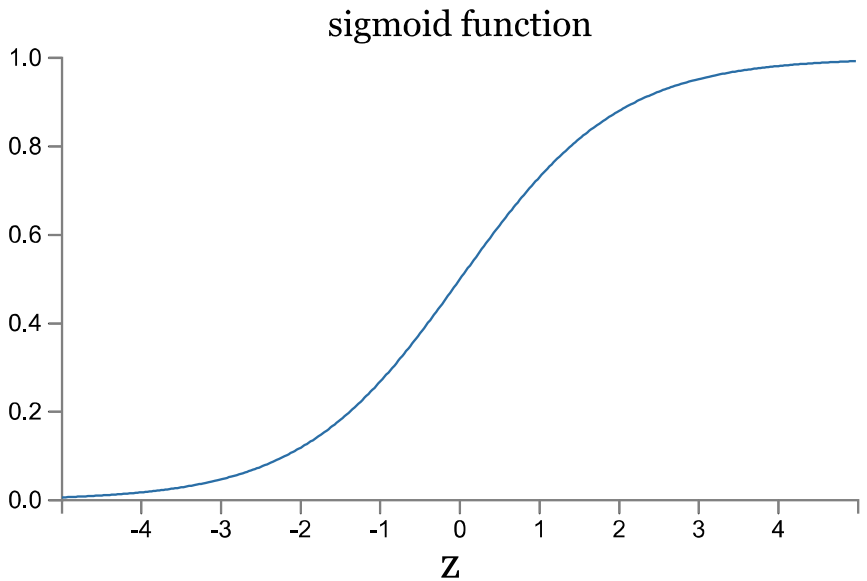
- Perceptrons are conceptually interesting, but they have the property of threshold-based activation, i.e., this activation function is not differentiable. This leads to potentially large changes in outputs for small changes in inputs making it difficult for such neurons to "learn" effectively:



Sigmoid Neurons

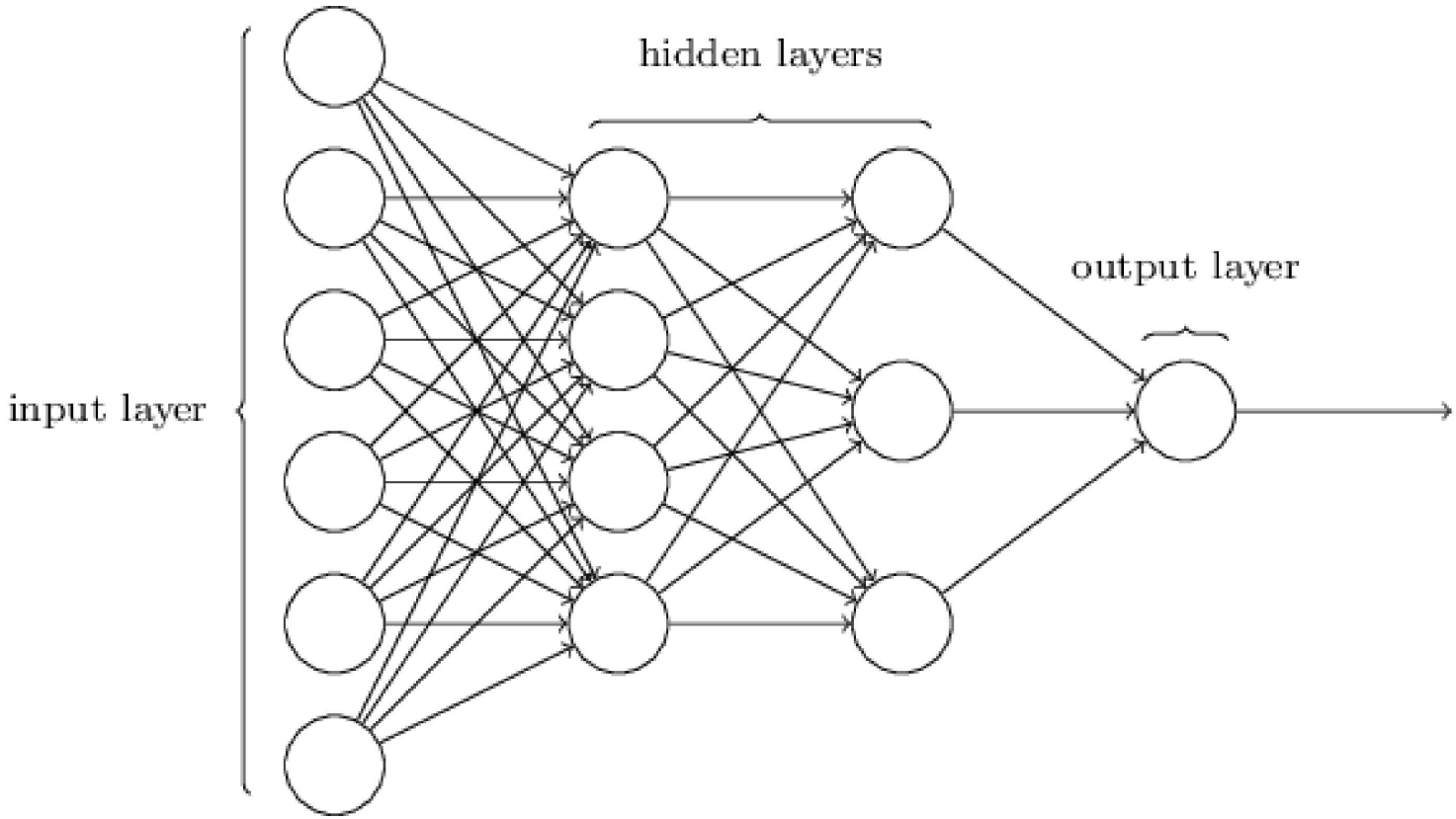
- Instead, we could use the standard logistic activation function which smooths out the response around the threshold. In neural network lingo, this is called a sigmoid activation function:

$$\sigma(z) = \frac{1}{1+e^{-z}}, \text{ where } z = \left(\sum_j w_j x_j - b \right)$$



Architecture of Neural Networks

- Most neural networks have multiple layers (these are sometimes, confusingly, called multilayer perceptrons, although they aren't made up of perceptrons...):



Mapping to what we know

- What are all the layers for?
- They are simply a way (like multiple leaves in tree models), to compute complicated nonlinear transformations and interactions of the various input variables.
- The more layers there are, the more complicated the transformations that can be accommodated.
- The next (difficult) step is to understand how to change the weights and biases to replicate "learning" by the network.

More Neural Network Basics

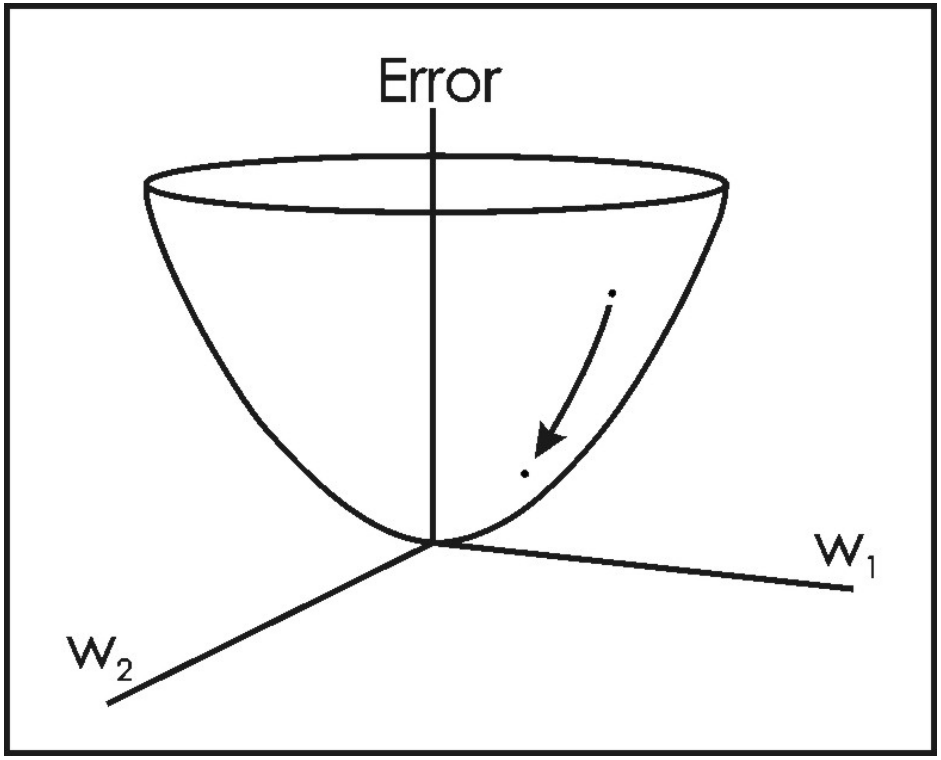
- The first step is to write down a loss (or cost) function. As usual, the loss function has as inputs:
  - n training sample inputs (where x is a specific instance of one of the n training sample observations).
  - parameters, in this case, weights and biases, throughout the network.
  - known training sample outputs (also known as activations) a.

$$C(w, b, x, y) = \frac{1}{2n} \sum_x \left\| y_x - a^L(x) \right\|^2$$

- Here, y_x is the true output associated with input x, and $a^L(x)$ is the final output of the network.

A Basic Explanation of Gradient Descent

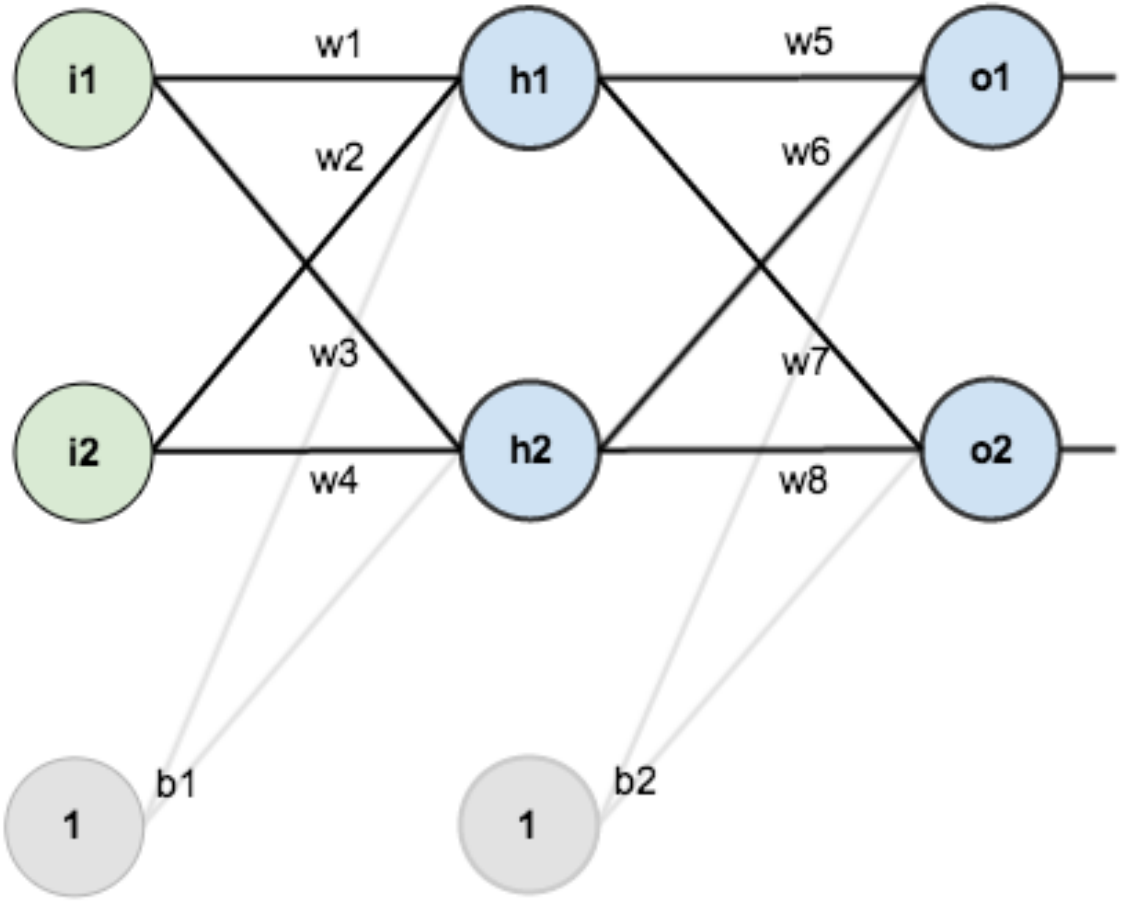
- The approach that is commonly used to generate "learning" in the neural network is the method of gradient descent.
- Just move in the direction in which the first derivatives of the cost function (also known as the "errors") are falling fastest:



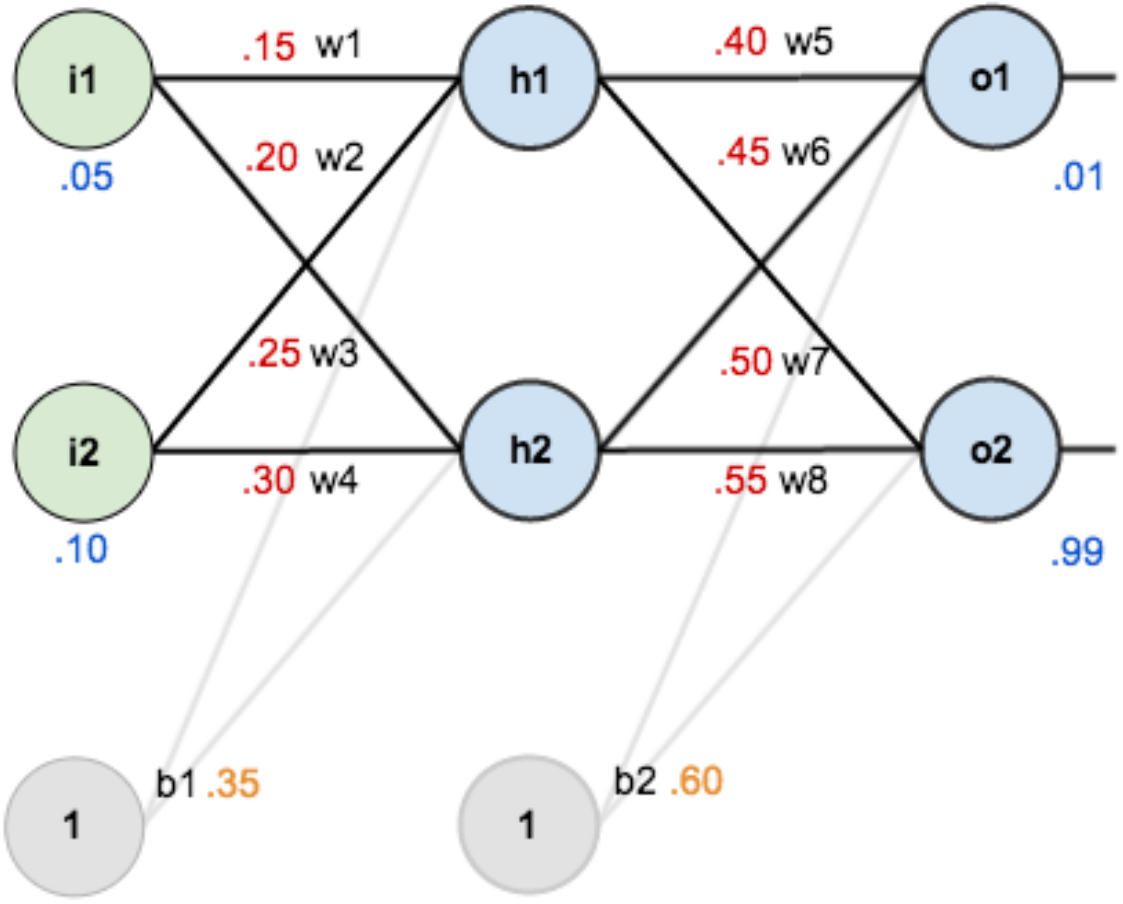
A Basic Explanation of Gradient Descent

- In order to do gradient descent, we need a way to compute the gradient of the cost function with respect to any given training input.
    - Once we know how to compute these, we can average the gradients across a randomly selected batch of training inputs (this is called stochastic gradient descent).
    - The next step is to move the weights and biases in the neural network in the direction of falling costs (i.e., where the gradient with respect to these parameters is negative).
    - Keep repeating until these gradients approach zero. You should then be at a minimum (at least in the training sample!)...
- To compute the gradients, a complicated-seeming (but conceptually simple) algorithm called backpropagation is often used.

Backpropagation with an Example



Backpropagation with an Example



Goal of Backpropogation

- Optimize weights so that the network can learn to map a set of inputs to outputs.
- The mathematics to formally document Backpropogation is not straightforward and is beyond the scope of this introductory course.
- Let's work just with one training set and follow a brilliantly visualized example here:
- This general approach solves many problems in finance!

Examples in Finance

- Forecasting financial time series
- Loan Application Evaluation when the outcome is not known
- Maximize credit card reach to those who would use them (although may not be the best thing for those individuals).
- Any set up where you want an engine to constantly learn about an environment you operate in to help make better decisions.

References

- R. Rojas: Neural Networks: A Systematic Introduction, Springer-Verlag, Berlin, 1996
- Michael Nielsen: Neural Networks and Deep Learning, Online Book
- Mattmazur.com for Backpropogation with an example.
- Introduction to Statistical Learning.