Artificial Intelligence
ENCS 434

Learning Methods

Aziz M. Qaroush - Birzeit University

What is Learning?

- Most often heard criticisms of AI is that machines cannot be called intelligent **until they are able to learn to do new things and adapt to new situations**, rather than simply doing as they are told to do.
- Some critics of AI have been saying that computers cannot learn!
- Definitions of Learning: changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time.
- Learning covers a wide range of phenomenon:
  - Skill refinement : Practice makes skills improve. More you play tennis, better you get
  - Knowledge acquisition: Knowledge is generally acquired through experience

Various learning mechanisms

- Simple storing of computed information or rote learning, is the most basic learning activity.
  - Many computer programs ie., database systems can be said to learn in this sense although most people would not call such simple storage learning.
- Another way we learn if through taking advice from others. Advice taking is similar to rote learning, but high-level advice may not be in a form simple enough for a program to use directly in problem solving.
- People also learn through their own problem-solving experience.
- Learning from examples : we often learn to classify things in the world without being given explicit rules.
- Learning from examples usually involves a teacher who helps us classify things by correcting us when we are wrong.

An example application

- An emergency room in a hospital measures 17 variables (e.g., blood pressure, age, etc) of newly admitted patients.
- A decision is needed: whether to put a new patient in an intensive-care unit.
- Due to the high cost of ICU, those patients who may survive less than a month are given higher priority.
- Problem: to predict high-risk patients and discriminate them from low-risk patients.

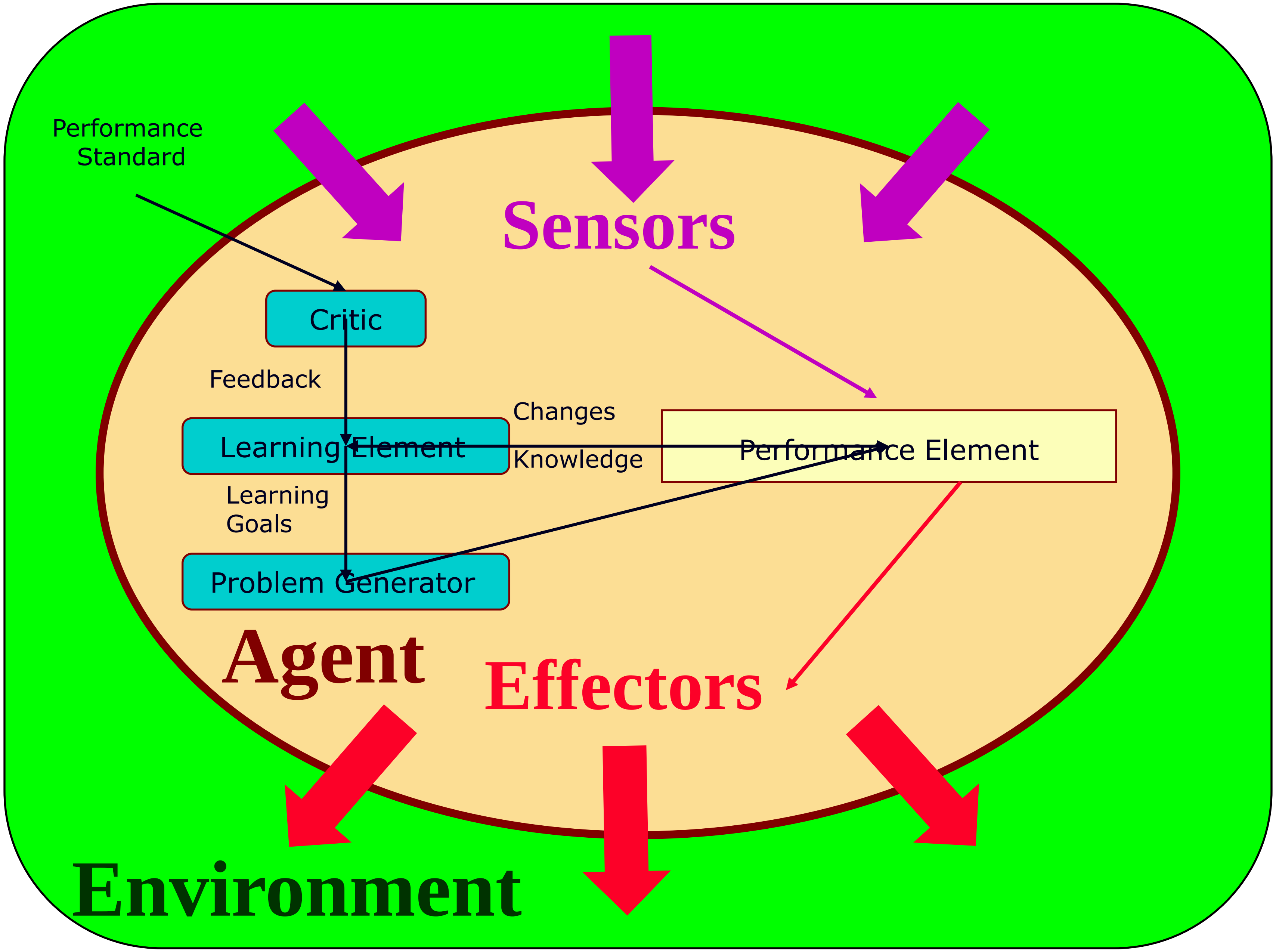
Another application

- A credit card company receives thousands of applications for new cards. Each application contains information about an applicant,
  - age
  - Marital status
  - annual salary
  - outstanding debts
  - credit rating
  - etc.
- Problem: to decide whether an application should approved, or to classify applications into two categories, approved and not approved.

Forms of Learning

- **supervised learning**
  - an agent tries to find a function that matches examples from a sample set
    - each example provides an input together with the correct output
  - a teacher provides feedback on the outcome
    - the teacher can be an outside entity, or part of the environment
- **unsupervised learning**
  - the agent tries to learn from patterns without corresponding output values
- **reinforcement learning**
  - the agent does not know the exact output for an input, but it receives feedback on the desirability of its behavior
    - the feedback can come from an outside entity, the environment, or the agent itself
    - the feedback may be delayed, and not follow the respective action immediately

Learning Agent Model



Learning Element Design Issues

- selections of the components of the performance elements that are to be improved
- representation mechanisms used in those components
- availability of feedback
- availability of prior information

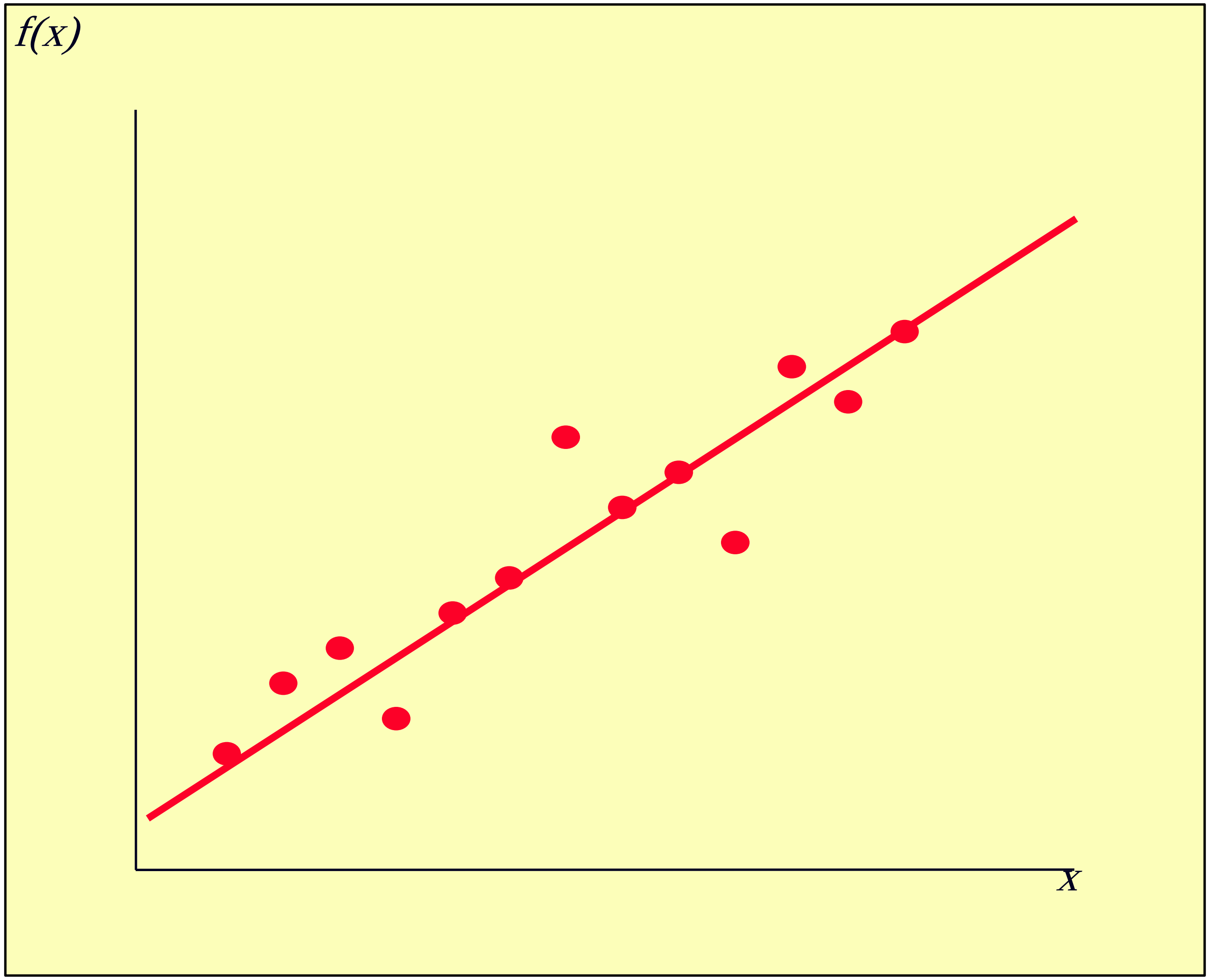
Machine learning

Supervised learning

Supervised learning

- Like human learning from past experiences.
- A computer does not have "experiences".
- A computer system learns from data, which represent some "past experiences" of an application domain.
- Our focus: learn a target function that can be used to predict the values of a discrete class attribute, e.g., approve or not-approved, and high-risk or low risk.
- The task is commonly called: Supervised learning, classification, or inductive learning.

Example Inductive Learning

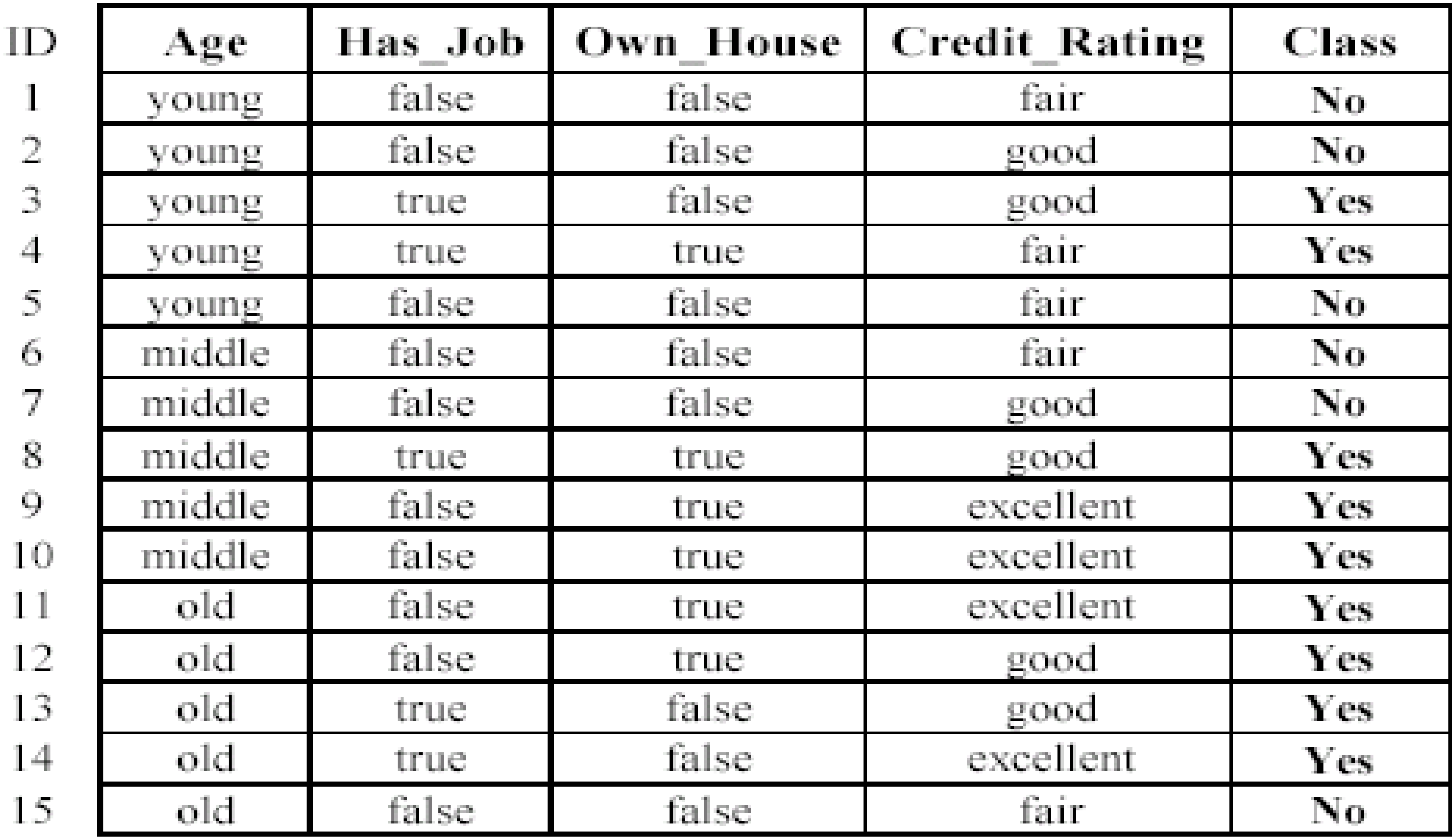


The data and the goal

- Data: A set of data records (also called examples, instances or cases) described by
  - k attributes: $A_1, A_2, \ldots A_k$.
  - a class: Each example is labelled with a pre-defined class.
- Goal: To learn a classification model from the data that can be used to predict the classes of new (future, or test) cases/instances.

An example: data (loan application)

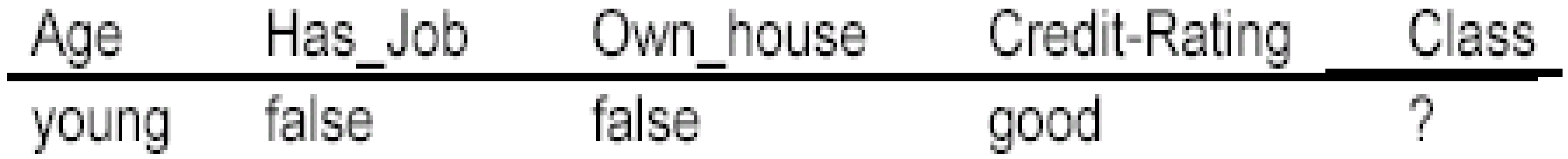
Approved or not



ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

An example: the learning task

- Learn a classification model from the data
- Use the model to classify future loan applications into
  - Yes (approved) and
  - No (not approved)
- What is the class for following case/instance?



Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	?

Supervised vs. unsupervised Learning

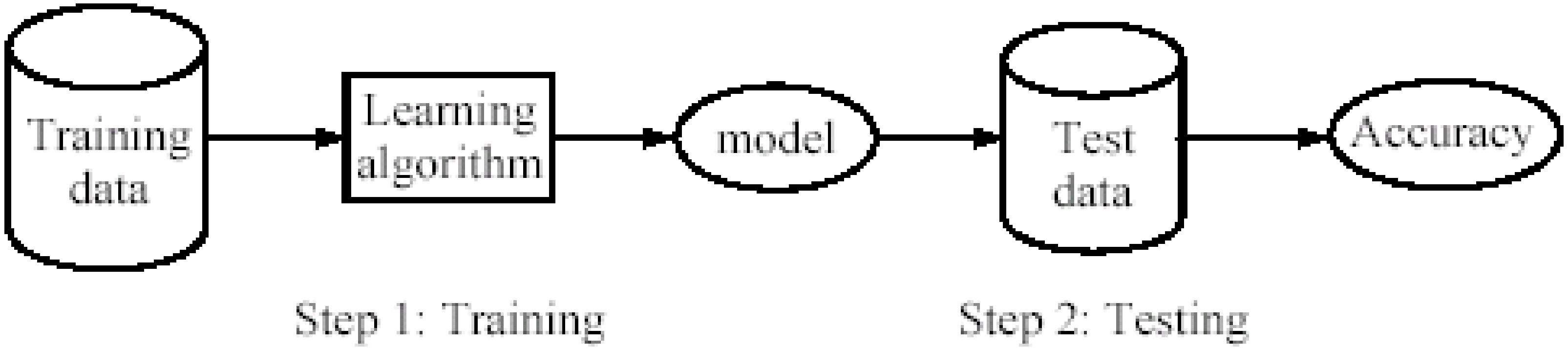
- Supervised learning: classification is seen as supervised learning from examples.
  - Supervision: The data (observations, measurements, etc.) are labeled with pre-defined classes. It is like that a "teacher" gives the classes (supervision).
  - Test data are classified into these classes too.
- Unsupervised learning (clustering)
  - Class labels of the data are unknown
  - Given a set of data, the task is to establish the existence of classes or clusters in the data

Supervised learning process: two steps

Learning (training): Learn a model using the training data

Testing: Test the model using unseen test data to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$



What do we mean by learning?

- Given
  - a data set *D*,
  - a task *T*, and
  - a performance measure *M*,

  a computer system is said to **learn** from *D* to perform the task *T* if after learning the system's performance on *T* improves as measured by *M*.
- In other words, the learned model helps the system to perform *T* better as compared to no learning.

An example

- Data: Loan application data
- Task: Predict whether a loan should be approved or not.
- Performance measure: accuracy.

No learning: classify all future applications (test data) to the majority class (i.e., Yes):

Accuracy = 9/15 = 60%.

- We can do better than 60% with learning.

Decision Trees

Introduction

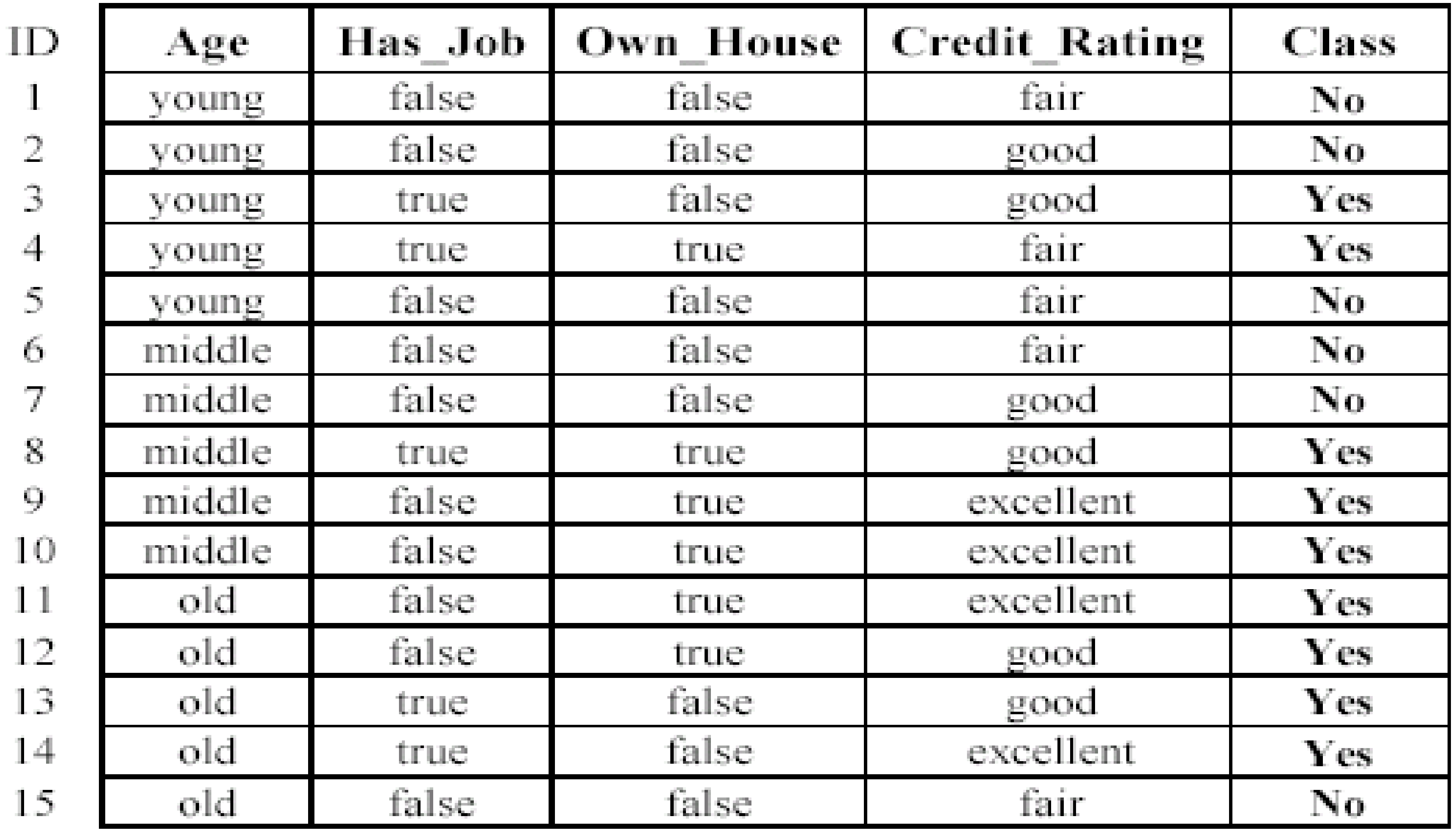
- Decision tree learning is one of the most widely used techniques for classification.
  - Its classification accuracy is competitive with other methods, and
  - it is very efficient.
- The classification model is a tree, called decision tree.
- C4.5 by Ross Quinlan is perhaps the best known system. It can be downloaded from the Web.

Boolean Decision Trees

- compute yes/no decisions based on sets of desirable or undesirable properties of an object or a situation
  - each node in the tree reflects one yes/no decision based on a test of the value of one property of the object
    - the root node is the starting point
    - leaf nodes represent the possible final decisions
  - branches are labeled with possible values
- the learning aspect is to predict the value of a *goal predicate* (also called goal concept)
  - a hypothesis is formulated as a function that defines the goal predicate

The loan data (reproduced)

Approved or not



ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

Learning Decision Trees

- problem: find a decision tree that agrees with the training set
- trivial solution: construct a tree with one branch for each sample of the training set
    - works perfectly for the samples in the training set
    - may not work well for new samples (generalization)
    - results in relatively large trees
- better solution: find a concise tree that still agrees with all samples
    - corresponds to the simplest hypothesis that is consistent with the training set

Constructing Decision Trees - Ockham's Razor

The most likely hypothesis is the simplest one that is consistent with all observations.

- general principle for inductive learning
- a simple hypothesis that is consistent with all observations is more likely to be correct than a complex one

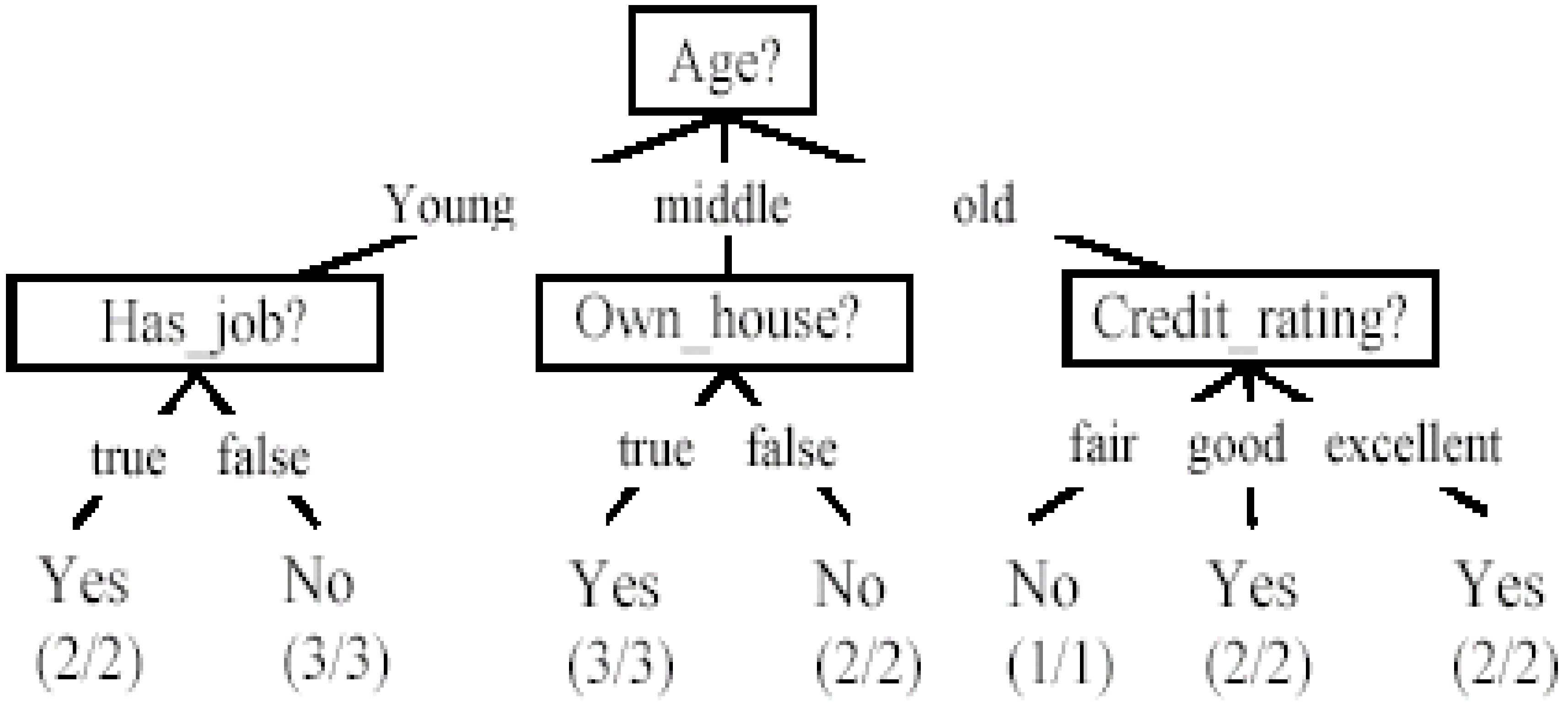
- in general, constructing the smallest possible decision tree is an intractable problem
- algorithms exist for constructing reasonably small trees
- basic idea: test the most important attribute first
  - attribute that makes the most difference for the classification of an example
    - can be determined through information theory
  - hopefully will yield the correct classification with few tests

Decision Tree Algorithm

- recursive formulation
    - select the best attribute to split positive and negative examples
    - if only positive or only negative examples are left, we are done
    - if no examples are left, no such examples were observers
        - return a default value calculated from the majority classification at the node's parent
    - if we have positive and negative examples left, but no attributes to split them we are in trouble
        - samples have the same description, but different classifications
        - may be caused by incorrect data (noise), or by a lack of information, or by a truly non-deterministic domain

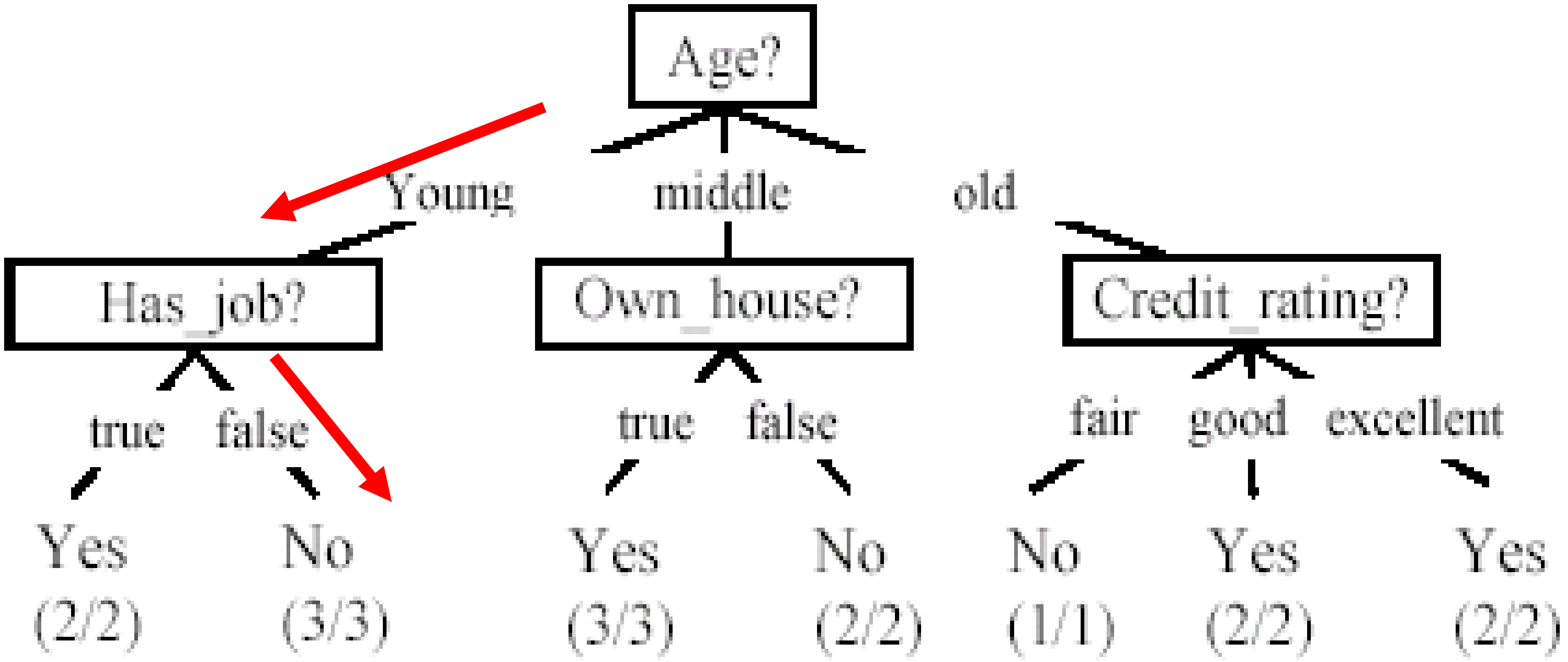
A decision tree from the loan data

Decision nodes and leaf nodes (classes)



Use the decision tree

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	? No

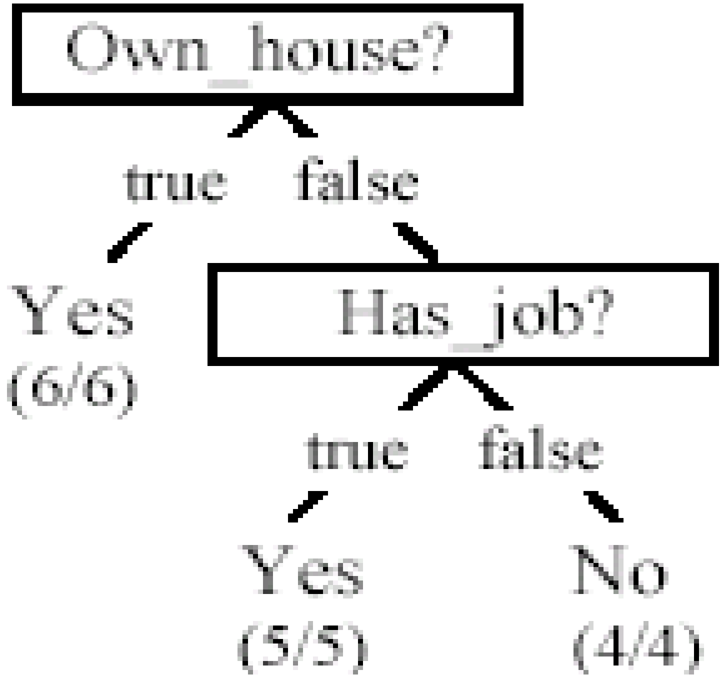


Is the decision tree unique?

No. Here is a simpler tree.

We want smaller tree and accurate tree.

Easy to understand and perform better.



Finding the best tree is NP-hard.

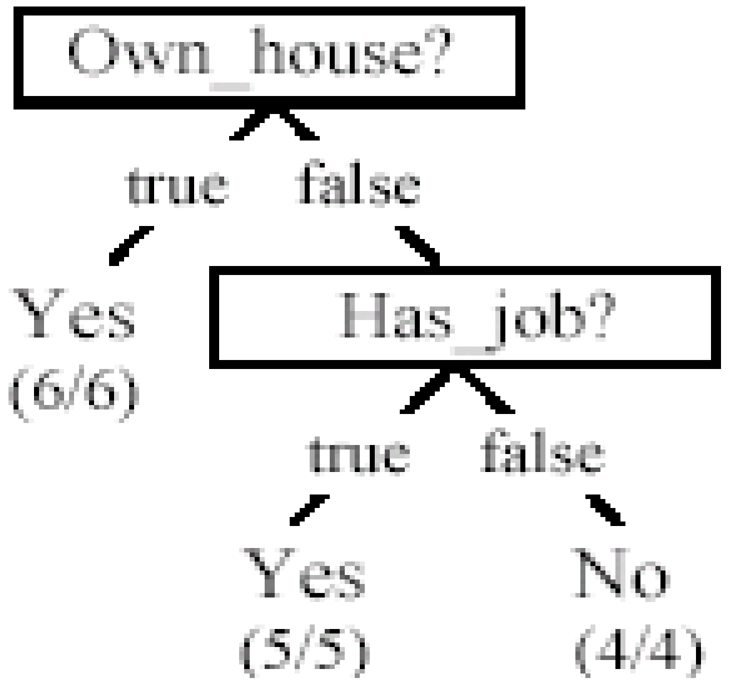
All current tree building algorithms are heuristic algorithms

Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"

From a decision tree to a set of rules

A decision tree can be converted to a set of rules

Each path from the root to a leaf is a rule.

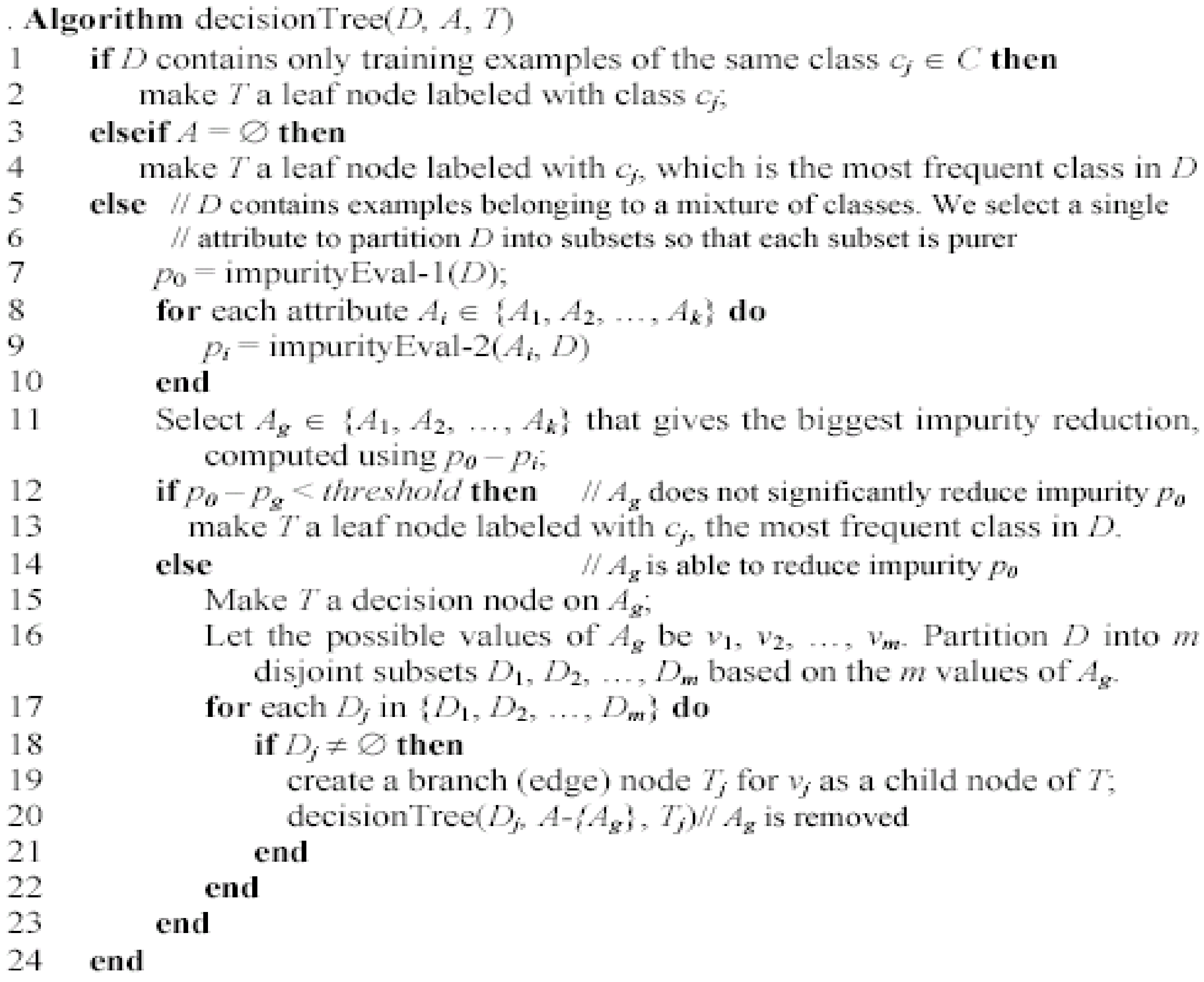


Own_house = true → Class =Yes [sup=6/15, conf=6/6]
Own_house = false, Has_job = true → Class = Yes [sup=5/15, conf=5/5]
Own_house = false, Has_job = false → Class = No [sup=4/15, conf=4/4]

Algorithm for decision tree learning

- Basic algorithm (a greedy **divide-and-conquer** algorithm)
  - Assume attributes are categorical now (continuous attributes can be handled too)
  - Tree is constructed in a top-down recursive manner
  - At start, all the training examples are at the root
  - Examples are partitioned recursively based on selected attributes
  - Attributes are selected on the basis of an impurity function (e.g., information gain)
- Conditions for stopping partitioning
  - All examples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority class is the leaf
  - There are no examples left

Decision tree learning algorithm



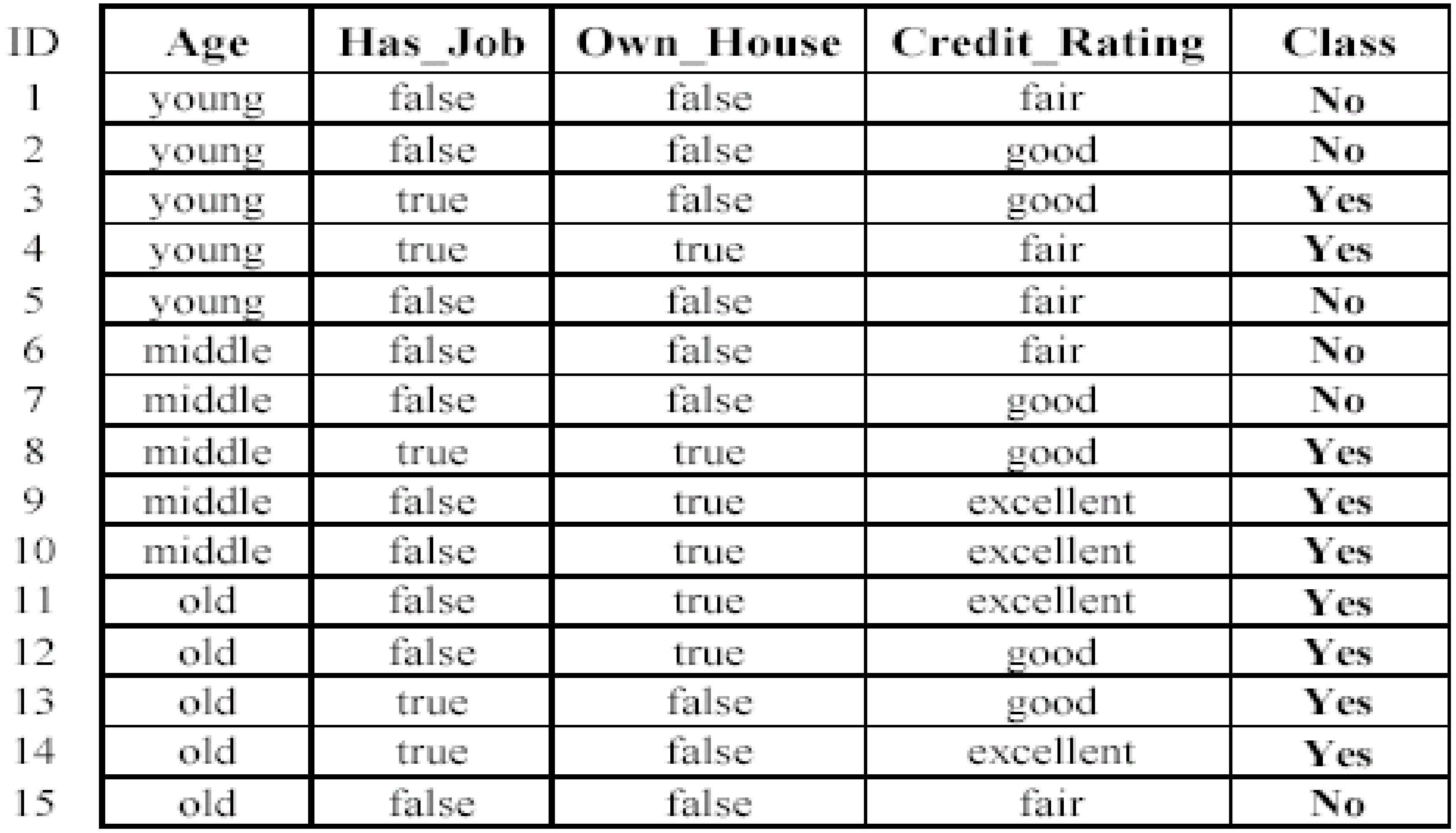
```
. Algorithm decisionTree($D$, $A$, $T$)
if $D$ contains only training examples of the same class $c_j \in C$ then
    make $T$ a leaf node labeled with class $c_j$;
elseif $A = \emptyset$ then
    make $T$ a leaf node labeled with $c_j$, which is the most frequent class in $D$
else  // $D$ contains examples belonging to a mixture of classes. We select a single
      // attribute to partition $D$ into subsets so that each subset is purer
     $p_0$ = impurityEval-1($D$);
     for each attribute $A_i \in \{A_1, A_2, \ldots, A_k\}$ do
         $p_i$ = impurityEval-2($A_i$, $D$)
     end
     Select $A_g \in \{A_1, A_2, \ldots, A_k\}$ that gives the biggest impurity reduction,
         computed using $p_0 - p_i$;
     if $p_0 - p_g < threshold$ then     // $A_g$ does not significantly reduce impurity $p_0$
        make $T$ a leaf node labeled with $c_j$, the most frequent class in $D$.
     else                                // $A_g$ is able to reduce impurity $p_0$
         Make $T$ a decision node on $A_g$;
         Let the possible values of $A_g$ be $v_1, v_2, \ldots, v_m$. Partition $D$ into $m$
             disjoint subsets $D_1, D_2, \ldots, D_m$ based on the $m$ values of $A_g$.
         for each $D_j$ in $\{D_1, D_2, \ldots, D_m\}$ do
             if $D_j \neq \emptyset$ then
                create a branch (edge) node $T_j$ for $v_j$ as a child node of $T$;
                decisionTree($D_j$, $A$-$\{A_g\}$, $T_j$)// $A_g$ is removed
             end
         end
     end
end
```

Choose an attribute to partition data

- The *key* to building a decision tree - which attribute to choose in order to branch.
- The objective is to reduce impurity or uncertainty in data as much as possible.
  - A subset of data is pure if all instances belong to the same class.
- The *heuristic* in C4.5 is to choose the attribute with the maximum Information Gain or Gain Ratio based on information theory.

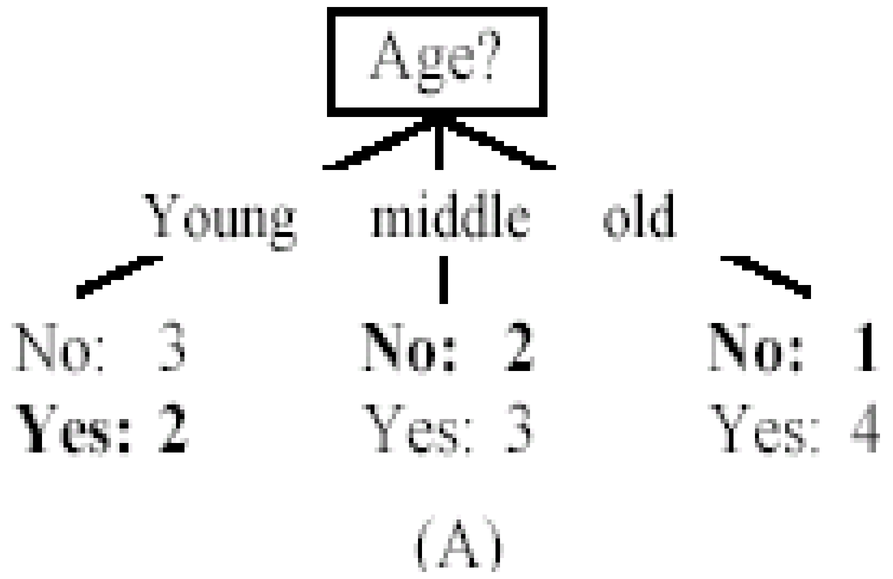
The loan data (reproduced)

Approved or not



ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

Two possible roots, which is better?



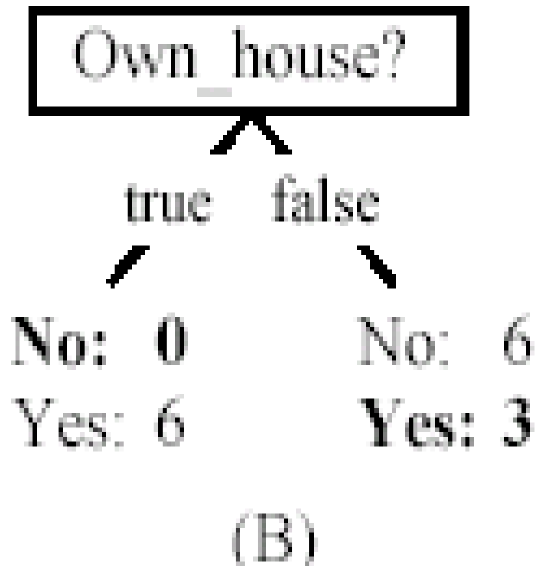


Fig. (B) seems to be better.

Information theory

- Information theory provides a mathematical basis for measuring the information content.
- To understand the notion of information, think about it as providing the answer to a question, for example, whether a coin will come up heads.
  - If one already has a good guess about the answer, then the actual answer is less informative.
  - If one already knows that the coin is rigged so that it will come with heads with probability 0.99, then a message (advanced information) about the actual outcome of a flip is worth less than it would be for a honest coin (50-50).

Information theory (cont ...)

- For a fair (honest) coin, you have no information, and you are willing to pay more (say in terms of \$) for advanced information - less you know, the more valuable the information.
- Information theory uses this same intuition, but instead of measuring the value for information in dollars, it measures information contents in **bits**.
- One bit of information is enough to answer a yes/no question about which one has no idea, such as the flip of a fair coin

Information theory: Entropy measure

- The entropy formula,

$$entropy(D) = -\sum_{j=1}^{|C|} \Pr(c_j) \log_2 \Pr(c_j)$$

$$\sum_{j=1}^{|C|} \Pr(c_j) = 1,$$

- $\Pr(c_j)$ is the probability of class c_j in data set D
- We use entropy as a measure of impurity or disorder of data set D. (Or, a measure of information in a tree)

Entropy measure: let us get a feeling

1. The data set D has 50% positive examples ($\Pr(positive) = 0.5$) and 50% negative examples ($\Pr(negative) = 0.5$).

$$entropy(D) = -0.5 \times \log_2 0.5 - 0.5 \times \log_2 0.5 = 1$$

2. The data set D has 20% positive examples ($\Pr(positive) = 0.2$) and 80% negative examples ($\Pr(negative) = 0.8$).

$$entropy(D) = -0.2 \times \log_2 0.2 - 0.8 \times \log_2 0.8 = 0.722$$

3. The data set D has 100% positive examples ($\Pr(positive) = 1$) and no negative examples, ($\Pr(negative) = 0$).

$$entropy(D) = -1 \times \log_2 1 - 0 \times \log_2 0 = 0$$

As the data become purer and purer, the entropy value becomes smaller and smaller. This is useful to us!

Using information theory

- To implement `Choose-Attribute` in the DTL algorithm
- Information Content (Entropy):

$$I(P(v_1), \dots , P(v_n)) = \Sigma_{i=1} -P(v_i) \log_2 P(v_i)$$

- For a training set containing *p* positive examples and *n* negative examples:

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n}\log_2 \frac{p}{p+n} - \frac{n}{p+n}\log_2 \frac{n}{p+n}$$

Information gain

- A chosen attribute *A* divides the training set *E* into subsets E_1, … , E_v according to their values for *A*, where *A* has *v* distinct values.

$$remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$

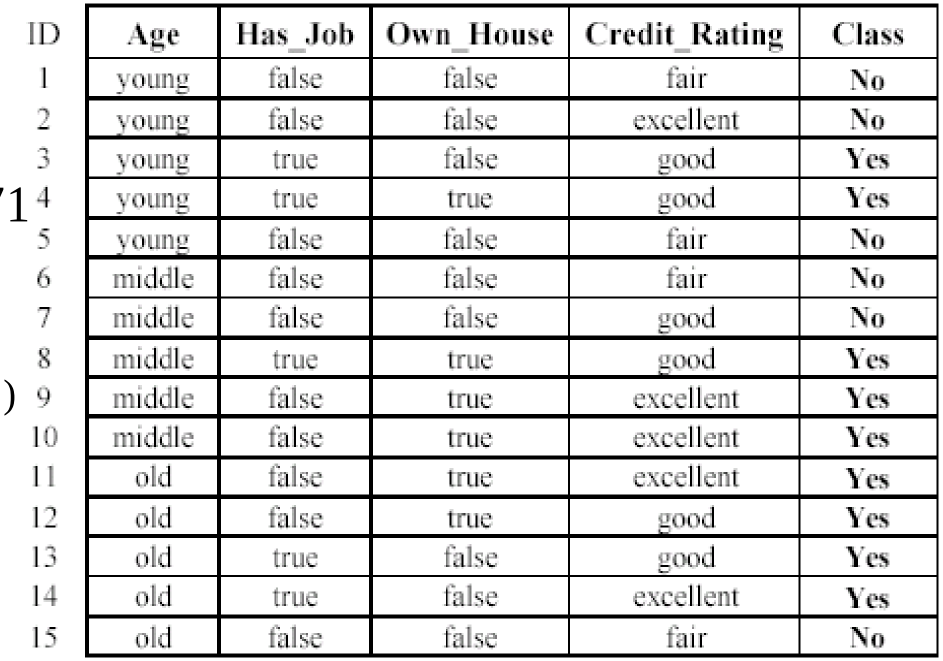
- Information Gain (IG) or reduction in entropy from the attribute test:

$$IG(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - remainder(A)$$

- Choose the attribute with the largest IG

An example

$$entropy(D) = -\frac{6}{15}\times\log_2\frac{6}{15} - \frac{9}{15}\times\log_2\frac{9}{15} = 0.971$$



ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	excellent	No
3	young	true	false	good	Yes
4	young	true	true	good	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

$$entropy_{Own\_house}(D) = -\frac{6}{15}\times entropy(D_1) - \frac{9}{15}\times entropy(D_2)$$
$$= \frac{6}{15}\times 0 + \frac{9}{15}\times 0.918$$
$$= 0.551$$

$$entropy_{Age}(D) = -\frac{5}{15}\times entropy(D_1) - \frac{5}{15}\times entropy(D_2) - \frac{5}{15}\times entropy(D_3)$$
$$= \frac{5}{15}\times 0.971 + \frac{5}{15}\times 0.971 + \frac{5}{15}\times 0.722$$
$$= 0.888$$

Age	Yes	No	entropy(Di)
young	2	3	0.971
middle	3	2	0.971
old	4	1	0.722

$gain(D, \text{Age}) = 0.971 - 0.888 = 0.083$

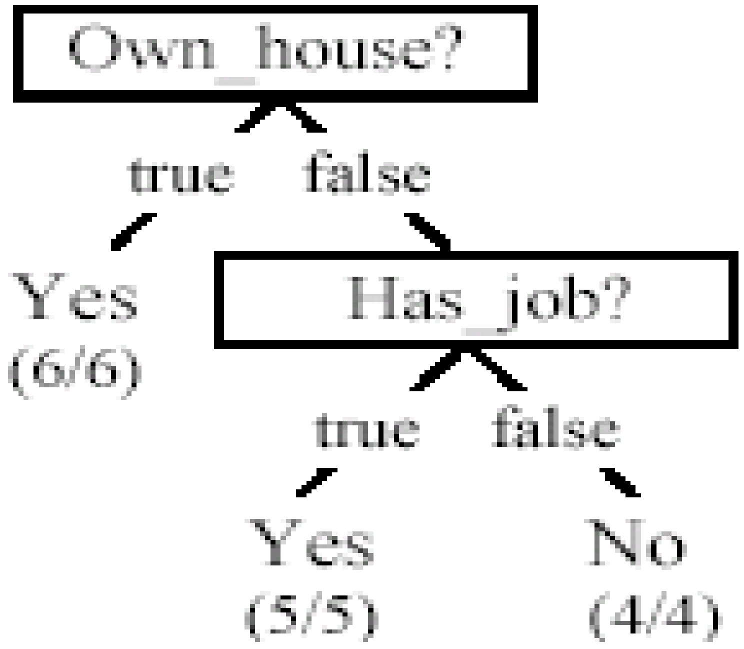
$gain(D, \text{Own\_house}) = 0.971 - 0.551 = 0.420$

$gain(D, \text{Has\_Job}) = 0.971 - 0.647 = 0.324$

$gain(D, \text{Credit\_Rating}) = 0.971 - 0.608 = 0.363$

Own_house is the best choice for the root.

We build the final tree

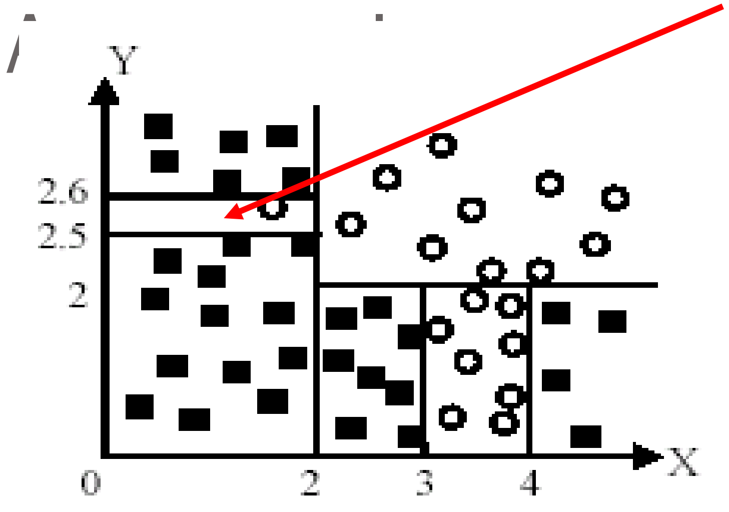


Avoid overfitting in classification

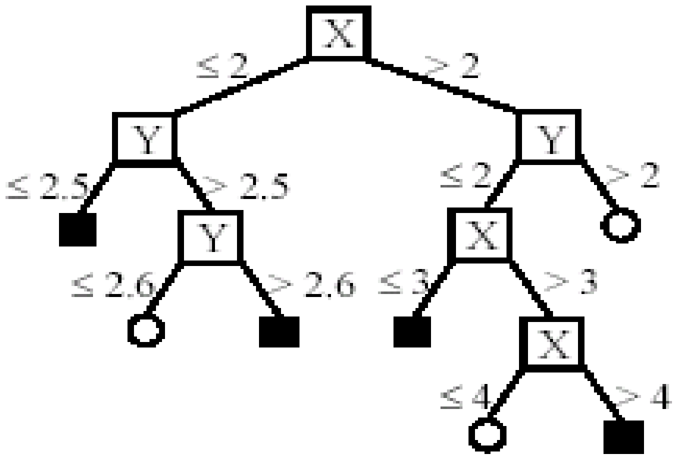
- Ideal goal of classification: Find the simplest decision tree that fits the data and generalizes to unseen data

- Overfitting: A tree may overfit the training data
  - Good accuracy on training data but poor on test data
  - Symptoms: tree too deep and too many branches, some may reflect anomalies due to noise or outliers
  - Overfitting results in decision trees that are more complex than necessary
  - Trade-off full consistency for compactness
    - Larger decision trees can be more consistent
    - Smaller decision trees generalize better

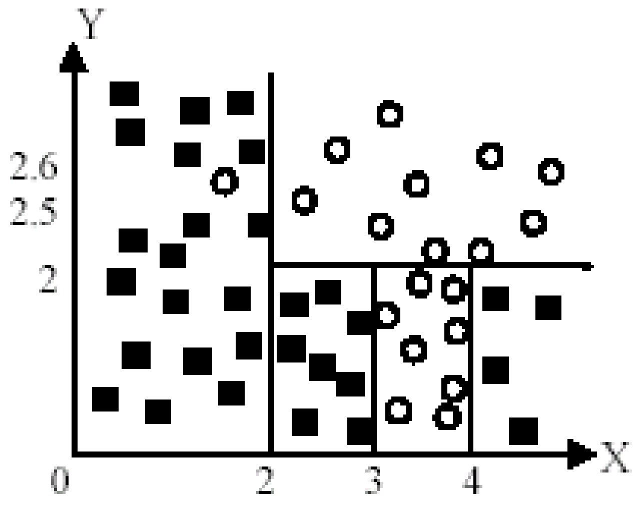
Likely to overfit the data

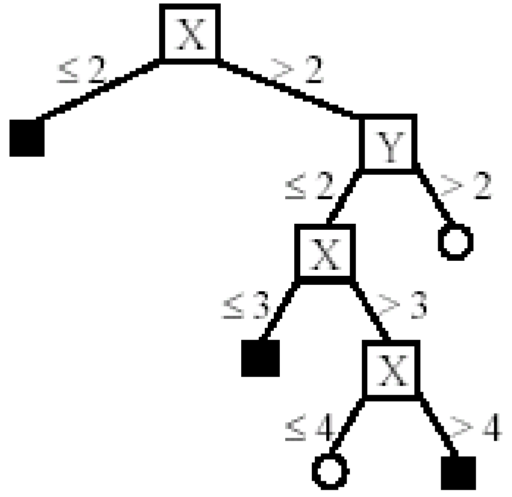


(A) A partition of the data space

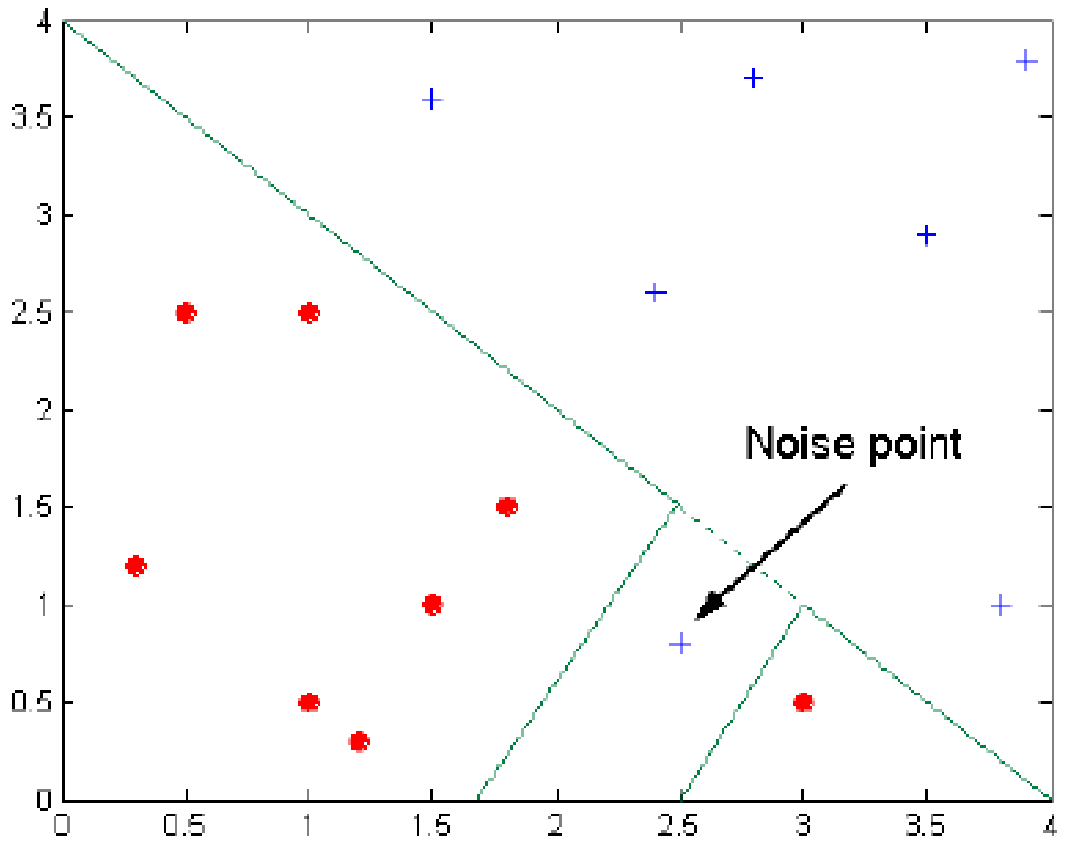


(B). The decision tree



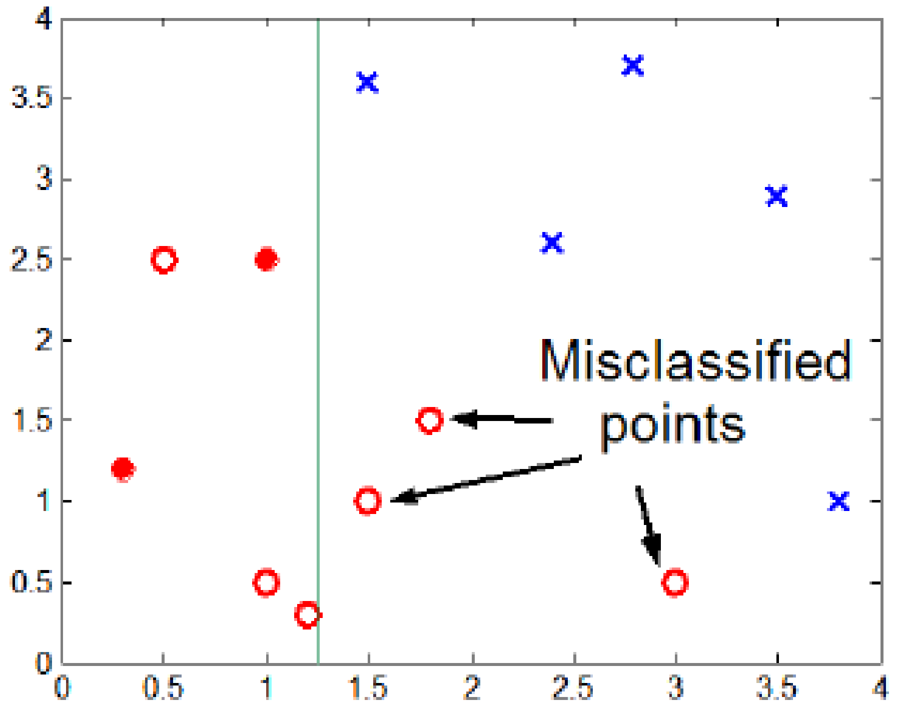


Overfitting due to Noise



Decision boundary is distorted by noise point

Overfitting due to Insufficient Examples

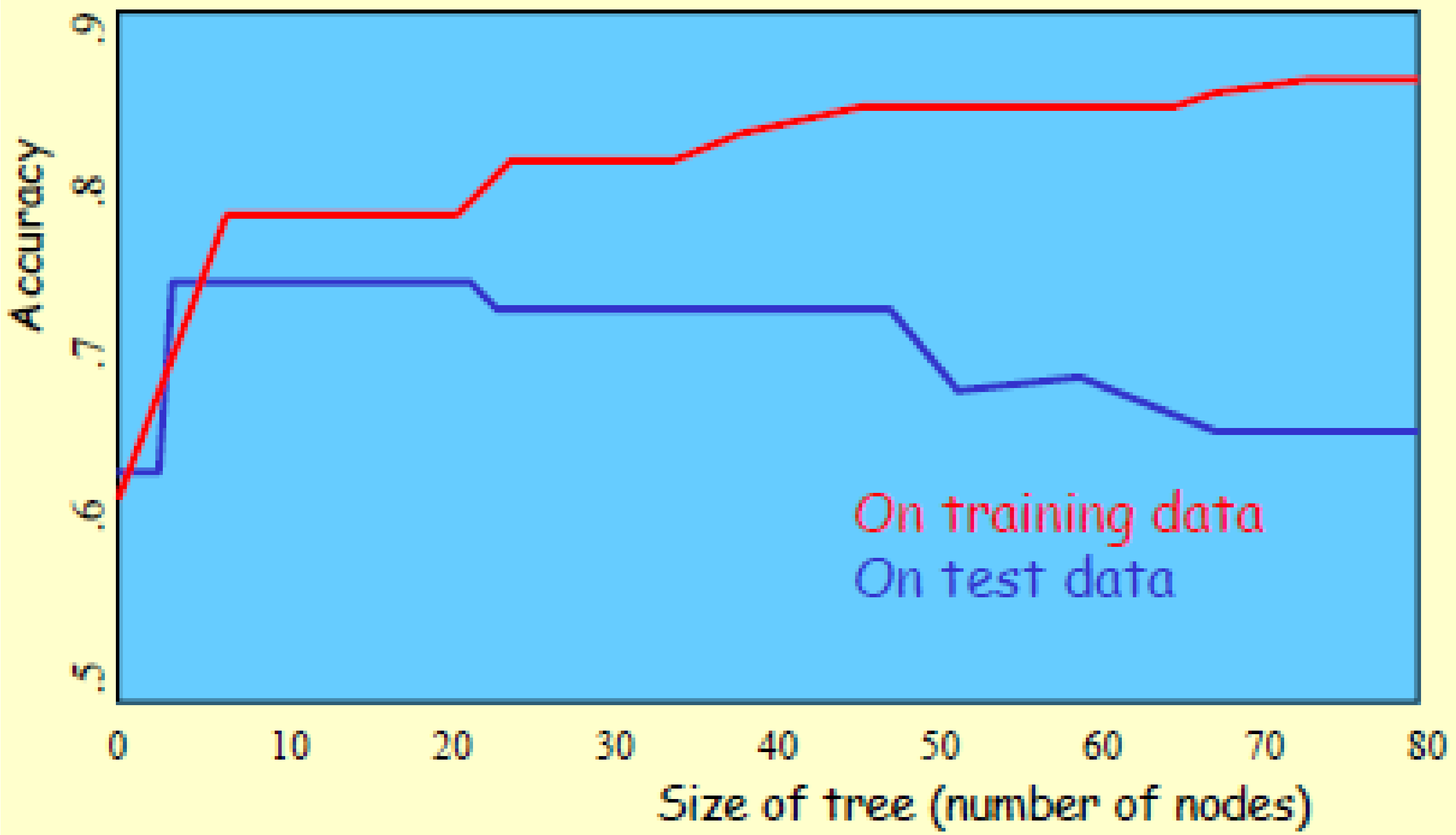


Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region

- Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task

Overfitting and accuracy

- Typical relation between tree size and accuracy:



Pruning to avoid overfitting

- **Prepruning**: Stop growing the tree when there is not enough data to make reliable decisions or when the examples are acceptably homogenous (ID3)
  - Do not split a node if this would result in the goodness measure falling below a threshold (e.g. InfoGain)
  - Difficult to choose an appropriate threshold
  - Since we use a hill-climbing search, looking only one step ahead, pre-pruning might stop too early.

- **Postpruning**: Grow the full tree, then remove nodes for which there is not sufficient evidence (C4.5)
  - Replace a split (subtree) with a leaf if the *predicted validation error is* no worse than the more complex tree (use dataset)

- Prepruning easier, but postpruning works better
- Prepruning - hard to know when to stop

Advantages/Disadvantages of Decision Trees

- Advantages:
  - Easy to understand (Doctors love them!)
  - Easy to generate rules
- Disadvantages:
  - May suffer from overfitting.
  - Classifies by rectangular partitioning (so does not handle correlated features very well).
  - Can be quite large – pruning is necessary.
  - Does not handle streaming data easily

Supervised learning

Artificial Neural Networks

Artificial neural networks:
Supervised learning

- Introduction, or how the brain works
- The neuron as a simple computing element
- The Perceptron
- Multilayer neural networks
- Accelerated learning in multilayer neural networks

Introduction, or how the brain works

- Machine learning involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by analogy.
- Learning capabilities can improve the performance of an intelligent system over time.
- The most popular approache to machine learning is **artificial neural networks**

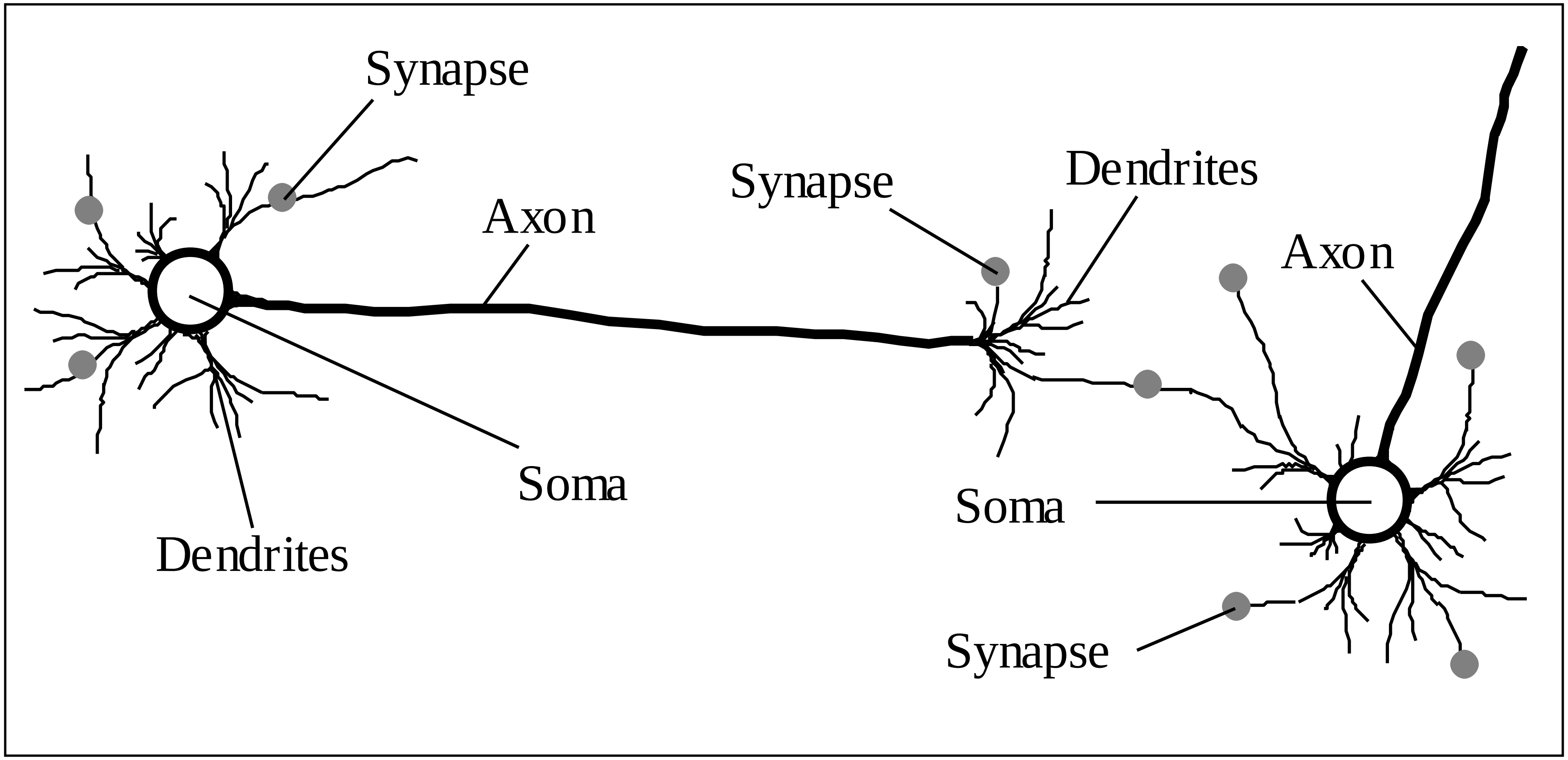
Artificial Neural Networks

- A **neural network** can be defined as a model of reasoning based on the human brain. The brain consists of a densely interconnected set of nerve cells, or basic information-processing units, called **neurons**.
- The human brain incorporates nearly 10 billion neurons and 60 trillion connections, *synapses*, between them. By using multiple neurons simultaneously, the brain can perform its functions much faster than the fastest computers in existence today.

Artificial Neural Networks

- Each neuron has a very simple structure, but an army of such elements constitutes a tremendous processing power.
- A neuron consists of a cell body, **soma**, a number of fibers called **dendrites**, and a single long fiber called the **axon**.

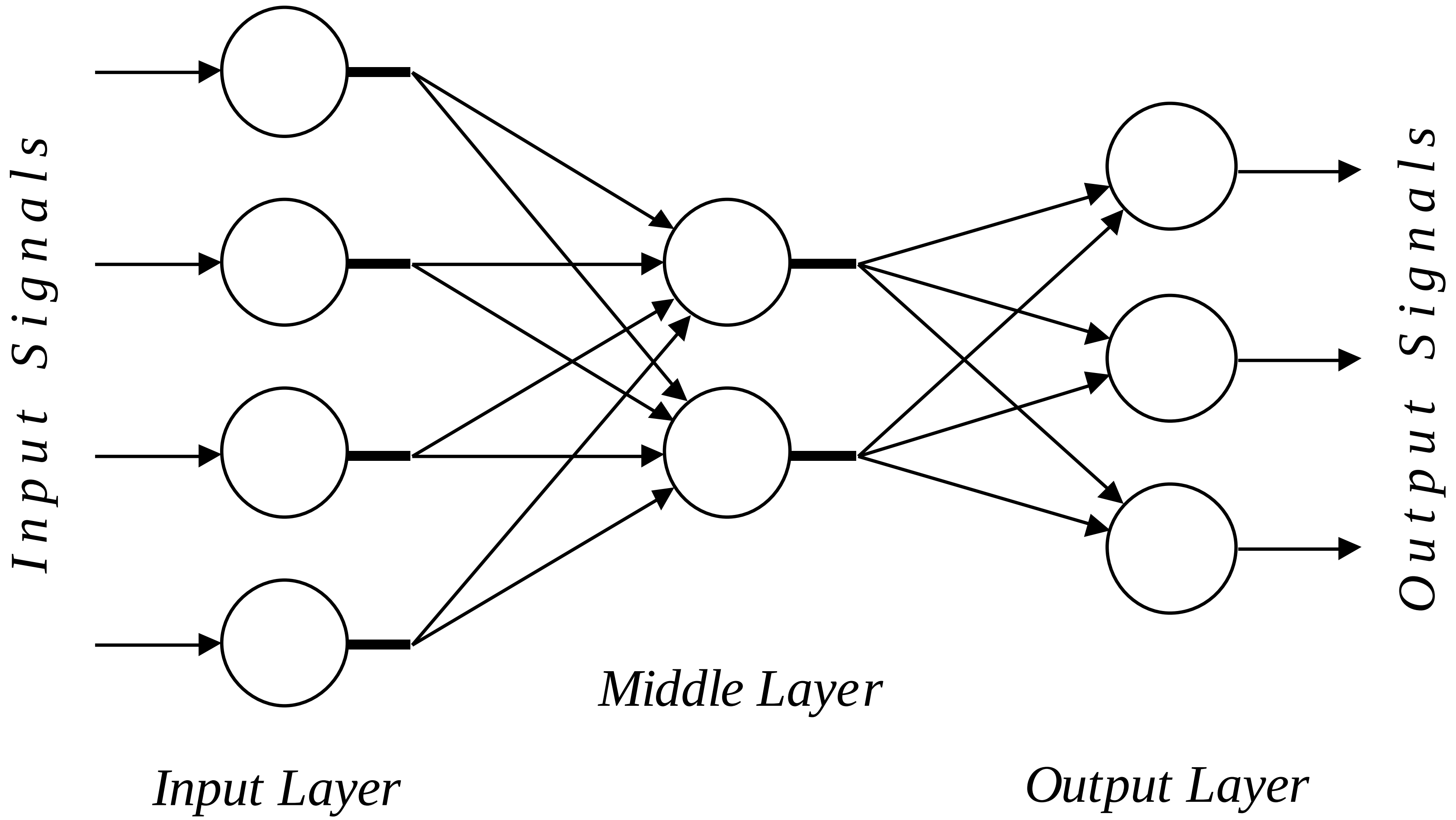
Biological neural network



Artificial Neural Network

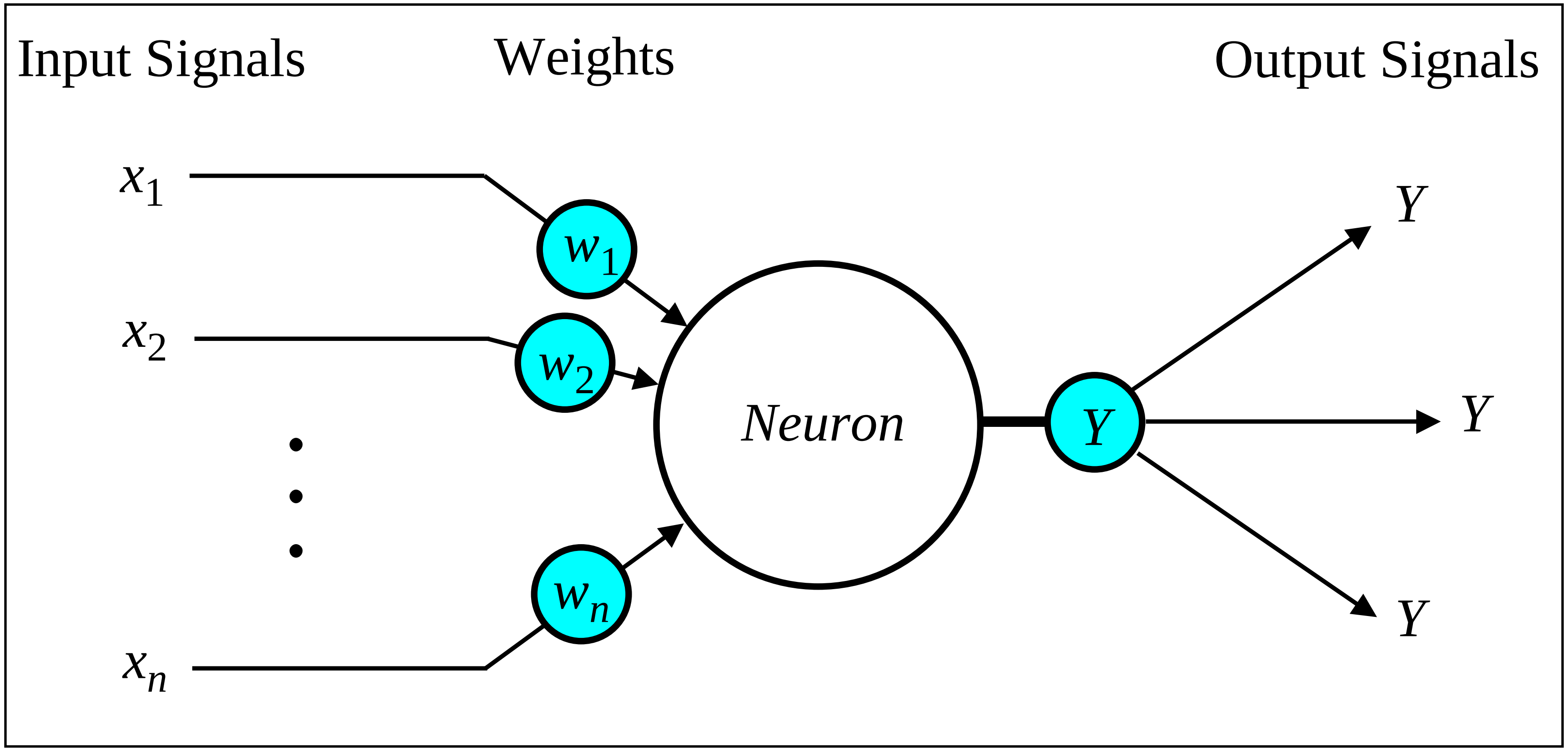
- An artificial neural network consists of a number of very simple processors, also called **neurons**, which are analogous to the biological neurons in the brain.
- The neurons are connected by weighted links passing signals from one neuron to another.
- The output signal is transmitted through the neuron's outgoing connection. The outgoing connection splits into a number of branches that transmit the same signal. The outgoing branches terminate at the incoming connections of other neurons in the network.

Architecture of a typical artificial neural network



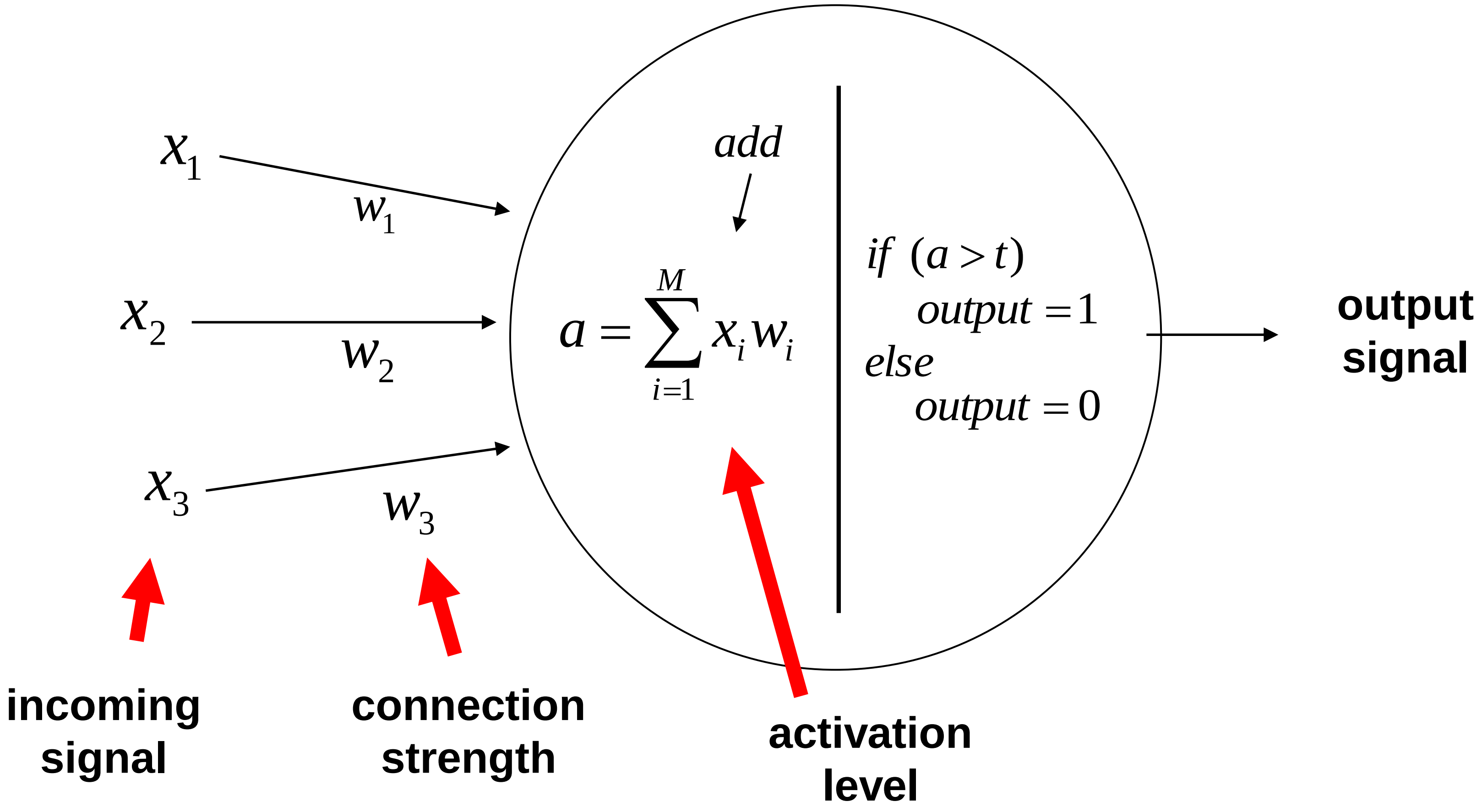
The neuron as a simple computing element

Diagram of a neuron



The neuron as a simple computing element

- input signals ‘x’ and coefficients ‘w’ are multiplied
- weights correspond to connection strengths
- signals are added up – if they are enough, FIRE!



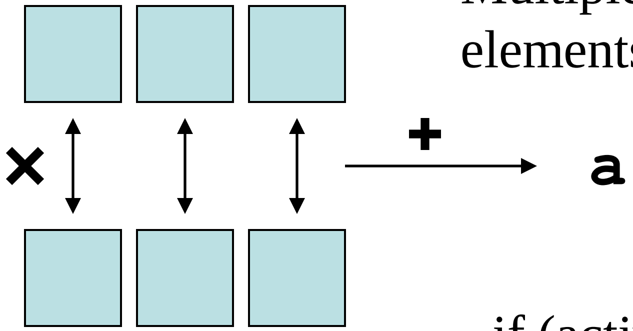
Calculation…

$$a = \sum_{i=1}^{M} x_i w_i$$

Sum notation

(just like a loop from 1 to M)

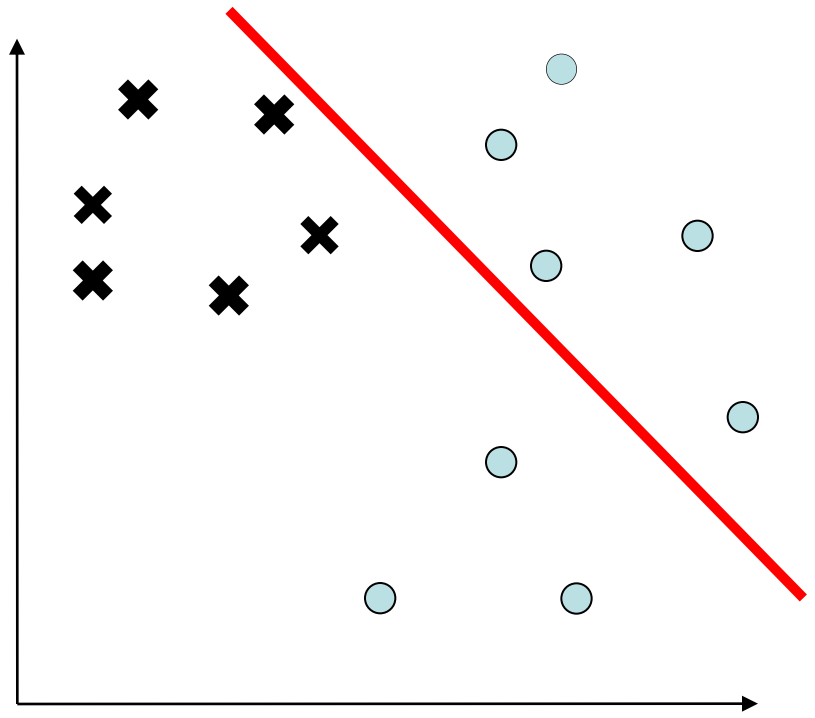
```
double[] x =

double[] w =
```



Multiple corresponding elements and add them up

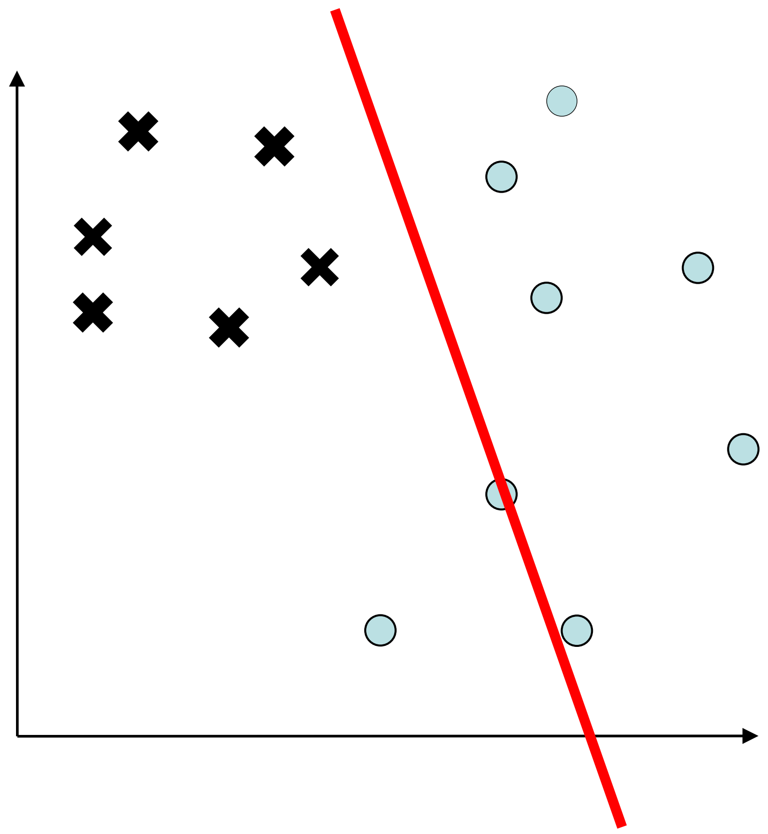
(activation)

if (activation > threshold) FIRE !



Is this a good decision boundary?

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$

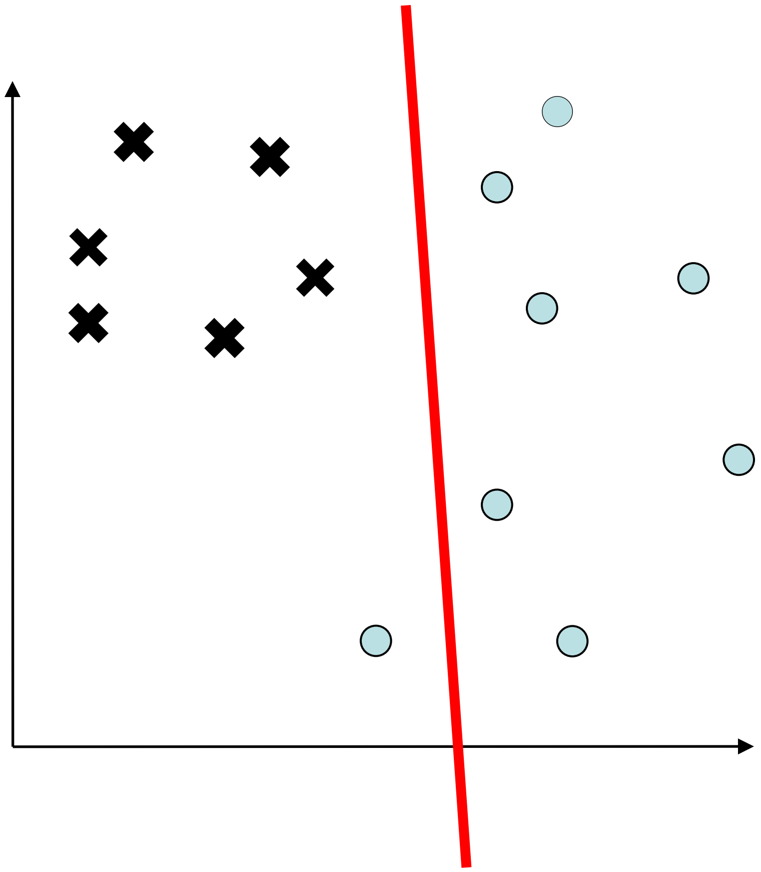


$w_1 = 2.1$

$w_2 = 0.2$

$t = 0.05$

$$\textit{if} \left(\sum_{i=1}^{M} x_i w_i \right) > t \quad \text{then } output = 1, \text{ else } output = 0$$

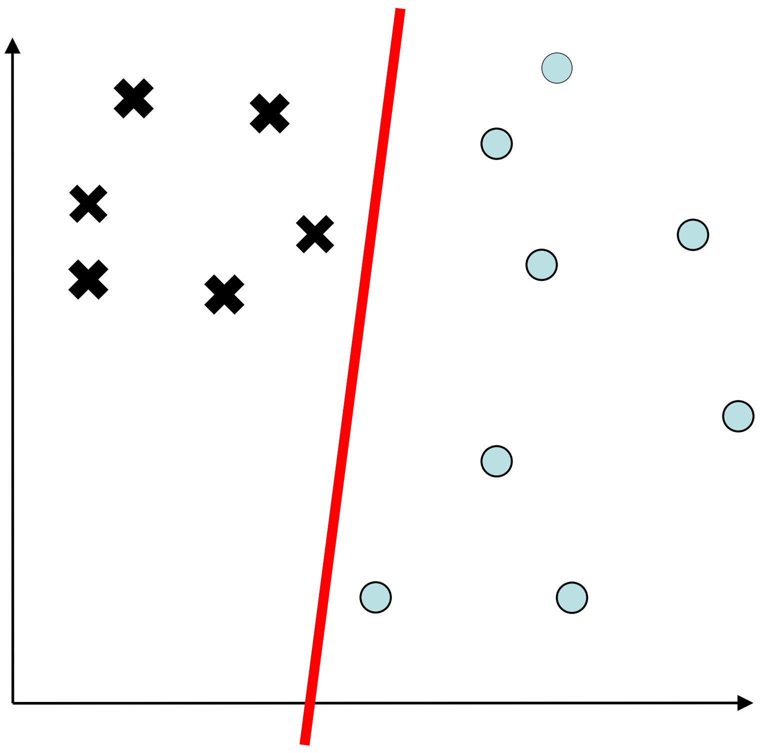


$w_1 = 1.9$

$w_2 = 0.02$

$t = 0.05$

$$\mathit{if}\ \left(\sum_{i=1}^{M} x_i w_i\right) > t \quad \text{then } output = 1, \ \text{else } output = 0$$



$w_1 = -0.8$

$w_2 = 0.03$

$t = 0.05$

Changing the weights/threshold makes the decision boundary move.

Pointless / impossible to do it by hand – only ok for simple 2-D case.

We need an algorithm….

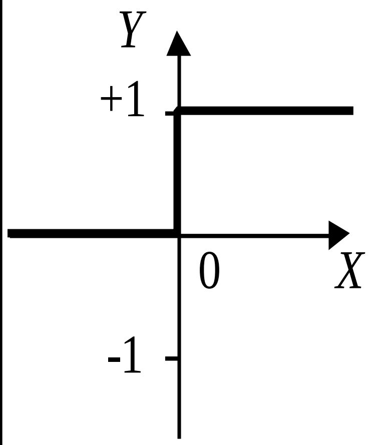
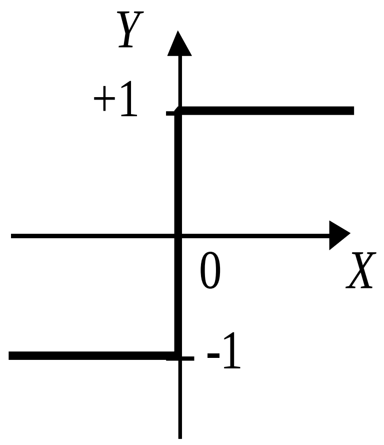
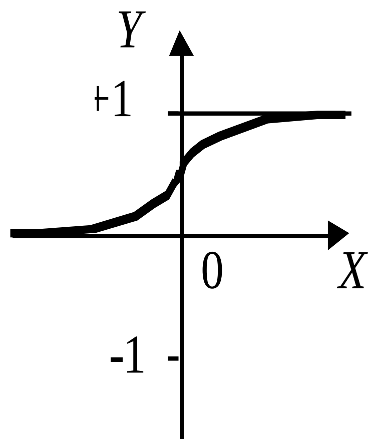
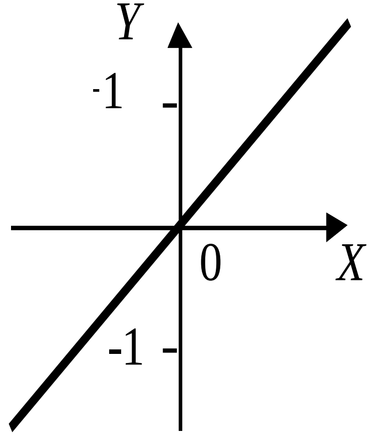
The neuron as a simple computing element

- The neuron computes the weighted sum of the input signals and compares the result with a **threshold value**, θ. If the net input is less than the threshold, the neuron output is –1. But if the net input is greater than or equal to the threshold, the neuron becomes activated and its output attains a value +1.
- The neuron uses the following transfer or **activation function**:

$$\boxed{X = \sum_{i=1}^{n} x_i w_i \qquad\qquad Y = \begin{cases} +1, & \text{if } X \geq \theta \\ -1, & \text{if } X < \theta \end{cases}}$$

- This type of activation function is called a **sign function**.

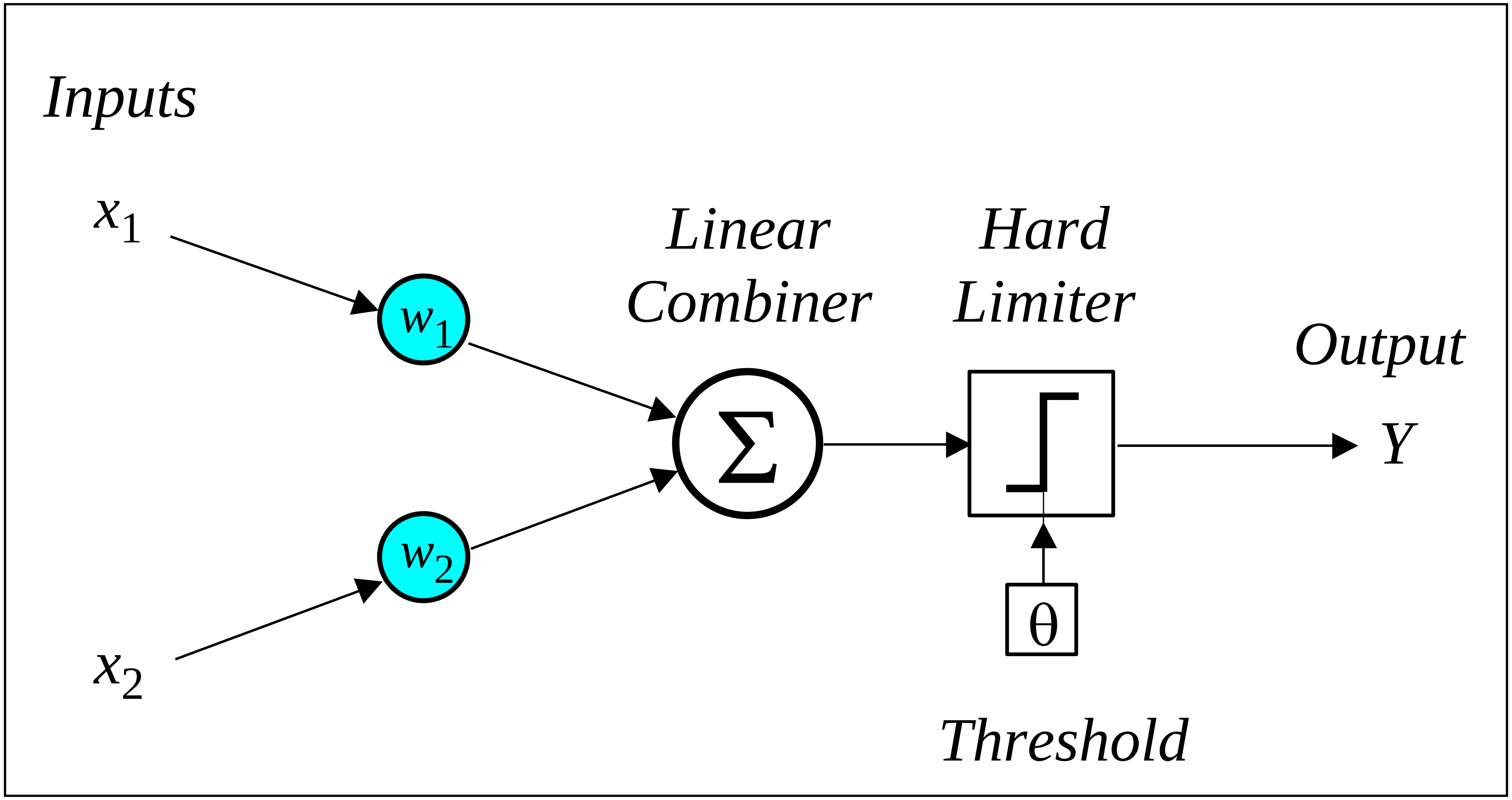
Activation functions of a neuron

Step function	*Sign function*	*Sigmoid function*	*Linear function*
			
$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{if } X < 0 \end{cases}$	$Y^{sign} = \begin{cases} +1, \text{if } X \geq 0 \\ -1, \text{if } X < 0 \end{cases}$	$Y^{sigmoid} = \frac{1}{1+e^{-X}}$	$Y^{linear} = X$

Can a single neuron learn a task?

- In 1958, **Frank Rosenblatt** introduced a training algorithm that provided the first procedure for training a simple ANN: a **perceptron**.
- The perceptron is the simplest form of a neural network. It consists of a single neuron with *adjustable* synaptic weights and a *hard limiter*.

Single-layer two-input perceptron



The Perceptron

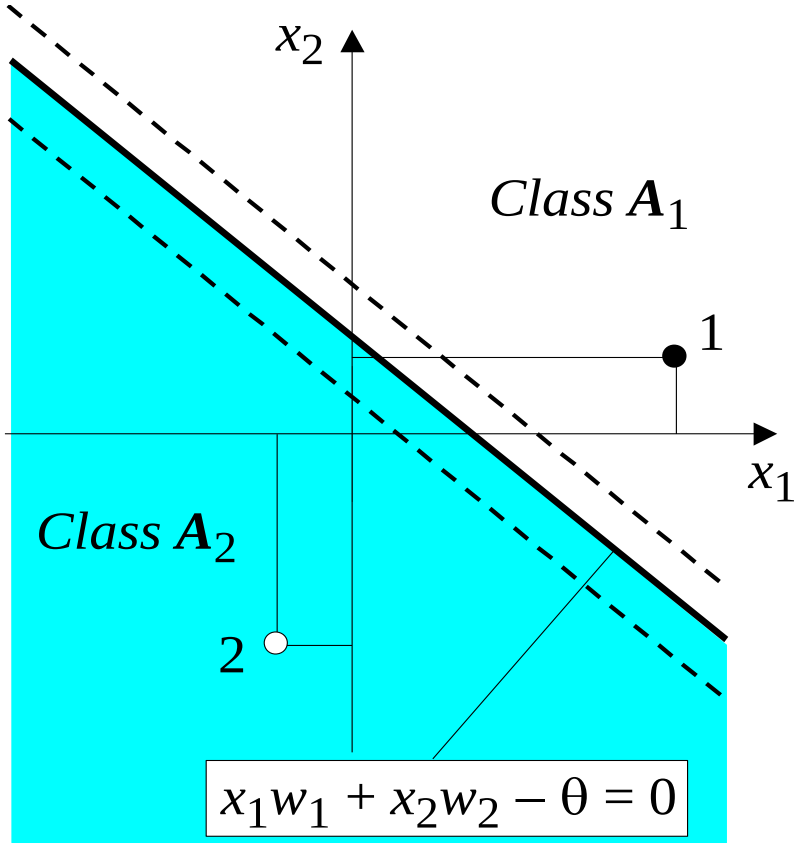
- The operation of Rosenblatt’s perceptron is based on the **McCulloch and Pitts neuron model**. The model consists of a linear combiner followed by a hard limiter.
- The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and –1 if it is negative.

The Perceptron

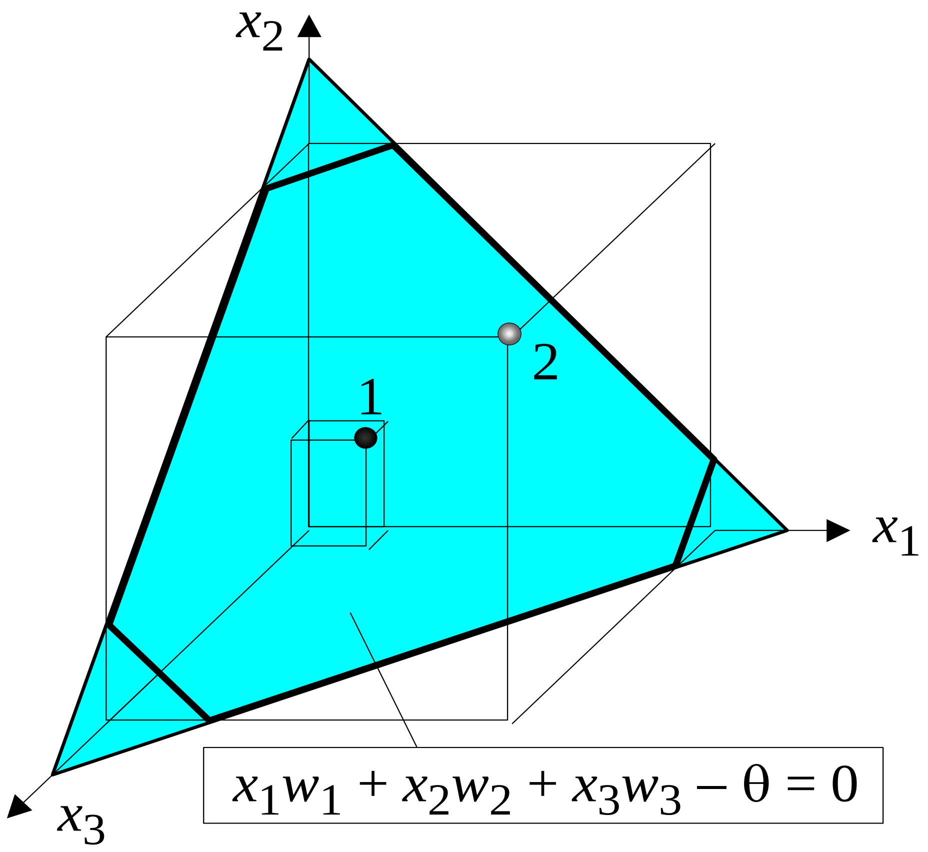
- The aim of the perceptron is to classify inputs, $x_1, x_2, \ldots, x_n$, into one of two classes, say $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$.
- In the case of an elementary perceptron, the n-dimensional space is divided by a *hyperplane* into two decision regions. The hyperplane is defined by the ***linearly separable* function**:

$$\boxed{\sum_{i=1}^{n} x_i w_i - \theta = 0}$$

Linear separability in the perceptrons



(*a*) Two-input perceptron.



(*b*) Three-input perceptron.

How does the perceptron learn its classification tasks?

- This is done by making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron.

- The initial weights are randomly assigned, usually in the range [–0.5, 0.5], and then updated to obtain the output consistent with the training examples.

How does the perceptron learn its classification tasks?

- If at iteration p, the actual output is $Y(p)$ and the desired output is $Y_d(p)$, then the error is given by:

  $$\boxed{e(p) = Y_d(p) - Y(p)} \quad \text{where } p = 1, 2, 3, \ldots$$

  Iteration p here refers to the pth training example presented to the perceptron.

- If the error, $e(p)$, is positive, we need to increase perceptron output $Y(p)$, but if it is negative, we need to decrease $Y(p)$.

The perceptron learning rule

$$\boxed{w_i(p+1) = w_i(p) + \alpha \cdot x_i(p) \cdot e(p)}$$

where $p = 1, 2, 3, \ldots$

α is the **learning rate**, a positive constant less than unity.

The perceptron learning rule was first proposed by **Rosenblatt** in 1960. Using this rule we can derive the perceptron training algorithm for classification tasks.

Perceptron’s tarining algorithm

Step 1: Initialisation

Set initial weights $w_1, w_2, \ldots, w_n$ and threshold θ to random numbers in the range [–0.5, 0.5].

If the error, $e(p)$, is positive, we need to increase perceptron output $Y(p)$, but if it is negative, we need to decrease $Y(p)$.

Perceptron's tarining algorithm (continued)

Step 2: Activation

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p), \ldots, x_n(p)$ and desired output $Y_d(p)$. Calculate the actual output at iteration $p = 1$

$$\boxed{Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]}$$

where n is the number of the perceptron inputs, and *step* is a step activation function.

Perceptron's tarining algorithm (continued)

Step 3: Weight training

Update the weights of the perceptron

$$\boxed{w_i(p+1) = w_i(p) + \Delta w_i(p)}$$

where $\Delta w_i(p)$ is the weight correction at iteration *p*.

The weight correction is computed by the **delta rule**:

$$\boxed{\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)}$$

Step 4: Iteration

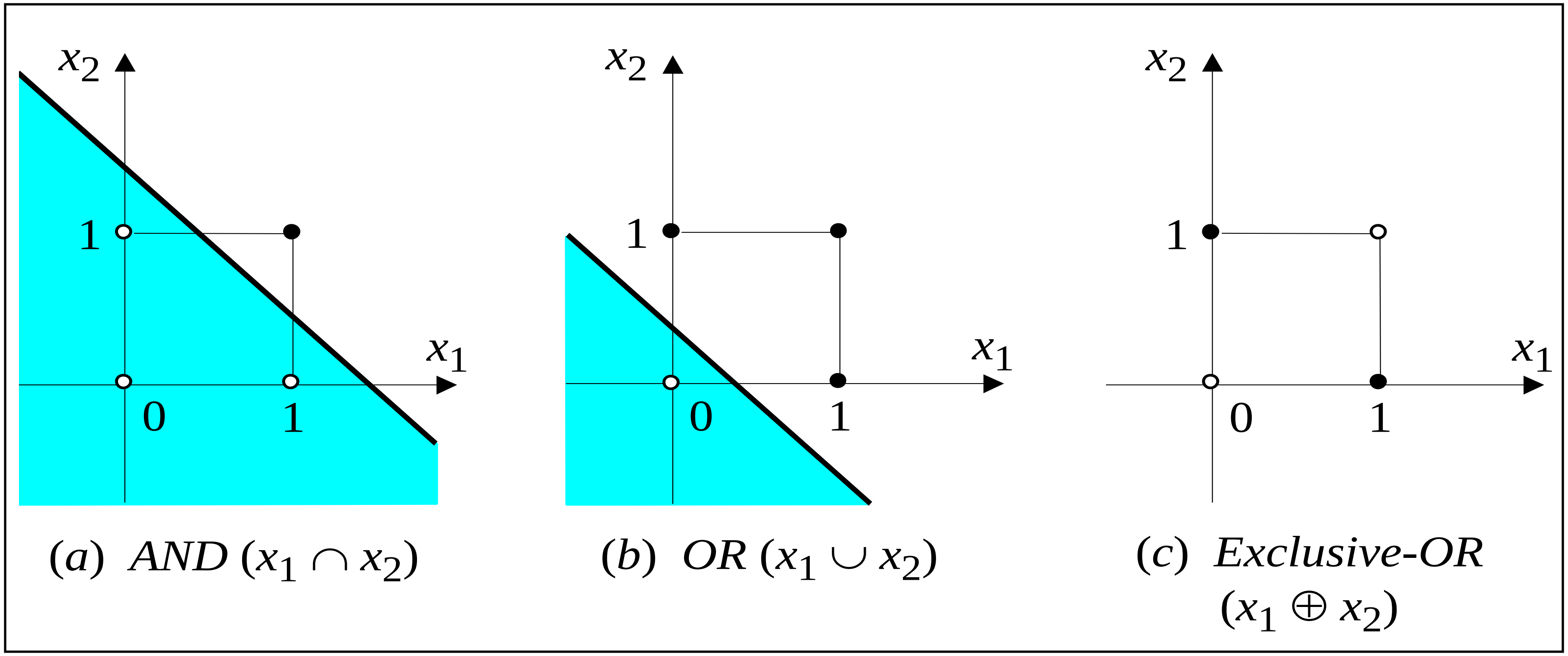
Increase iteration *p* by one, go back to *Step 2* and repeat the process until convergence.

Example of perceptron learning: the logical operation *AND*

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	–0.1	0	0	0.3	–0.1
	0	1	0	0.3	–0.1	0	0	0.3	–0.1
	1	0	0	0.3	–0.1	1	–1	0.2	–0.1
	1	1	1	0.2	–0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	–1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	–1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	–1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

Two-dimensional plots of basic logical operations



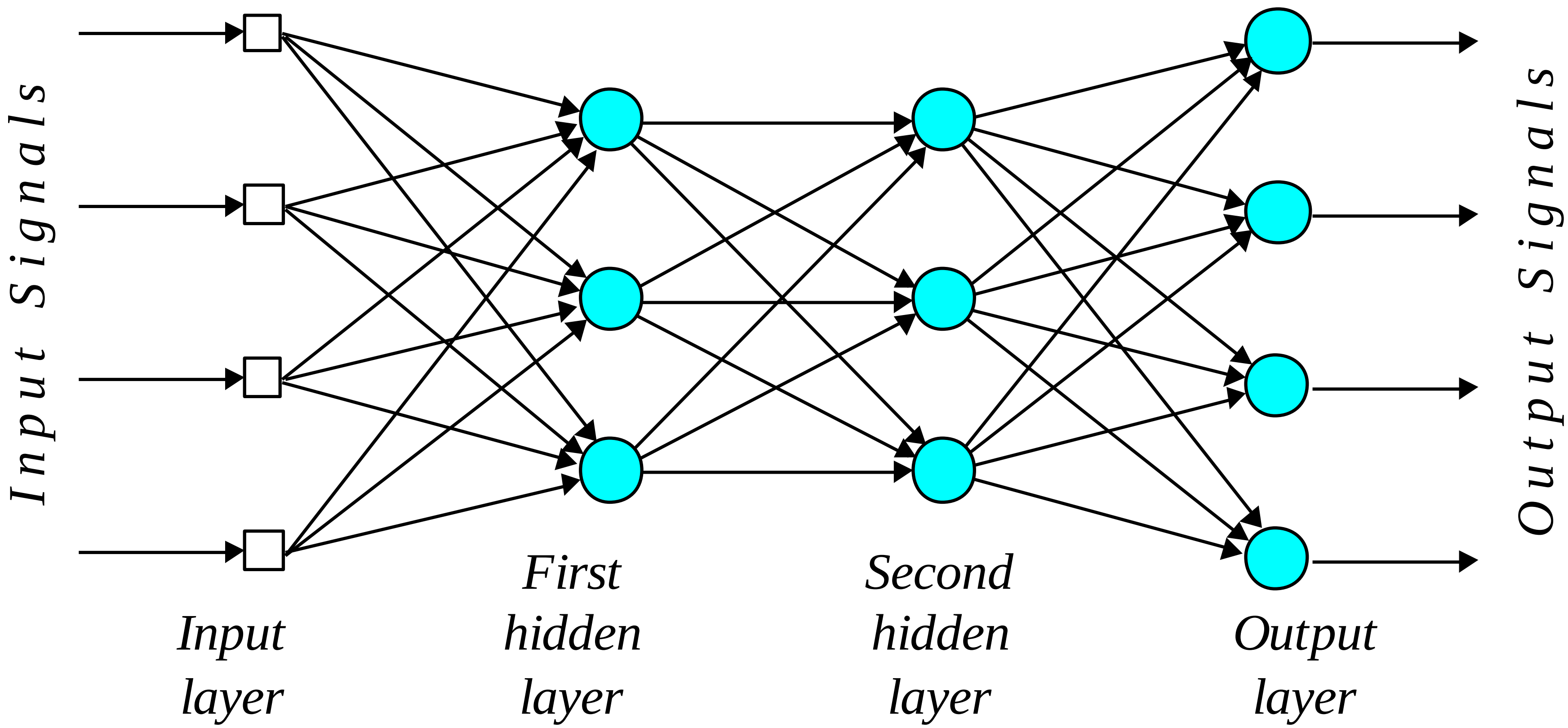
x_2
x_1
1
0
1
(a) *AND* $(x_1 \cap x_2)$
(b) *OR* $(x_1 \cup x_2)$
(c) *Exclusive-OR* $(x_1 \oplus x_2)$

A perceptron can learn the operations *AND* and *OR*, but not *Exclusive-OR*.

Multilayer neural networks

- A multilayer perceptron is a feedforward neural network with one or more hidden layers.
- The network consists of an **input layer** of source neurons, at least one middle or **hidden layer** of computational neurons, and an **output layer** of computational neurons.
- The input signals are propagated in a forward direction on a layer-by-layer basis.

Multilayer perceptron with two hidden layers



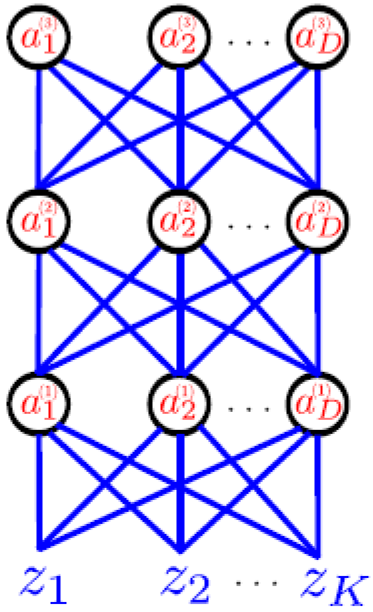
What does the middle layer hide?

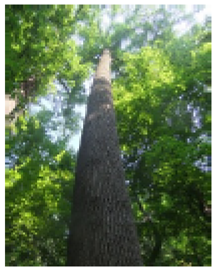
- A hidden layer "hides" its desired output. Neurons in the hidden layer cannot be observed through the input/output behaviour of the network. There is no obvious way to know what the desired output of the hidden layer should be.
- Neurons in the hidden layer detect the features; the weights of the neurons represent the features hidden in the input patterns. These features are then used by the output layer in determining the output pattern.
- Commercial ANNs incorporate three and sometimes four layers, including one or two hidden layers. Each layer can contain from 10 to 1000 neurons. Experimental neural networks may have five or even six layers, including three or four hidden layers, and utilise millions of neurons.

Why use hierarchical multi-layered models?

Argument 1: visual scenes are hierachically organised

$$\begin{tikzcd}
\text{object} & \text{trees} \\
\text{object parts} \arrow[u] & \text{bark, leaves, etc.} \arrow[u] \\
\text{primitive features} \arrow[u] & \text{oriented edges} \arrow[u] \\
\text{input image} \arrow[u] & \text{forest image} \arrow[u]
\end{tikzcd}$$

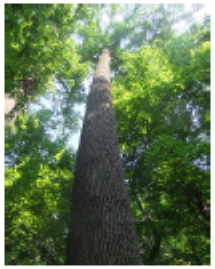

$a_1^{(3)}$ $a_2^{(3)}$ $\cdots$ $a_D^{(3)}$
$a_1^{(2)}$ $a_2^{(2)}$ $\cdots$ $a_D^{(2)}$
$a_1^{(1)}$ $a_2^{(1)}$ $\cdots$ $a_D^{(1)}$
z_1 z_2 $\cdots$ z_K



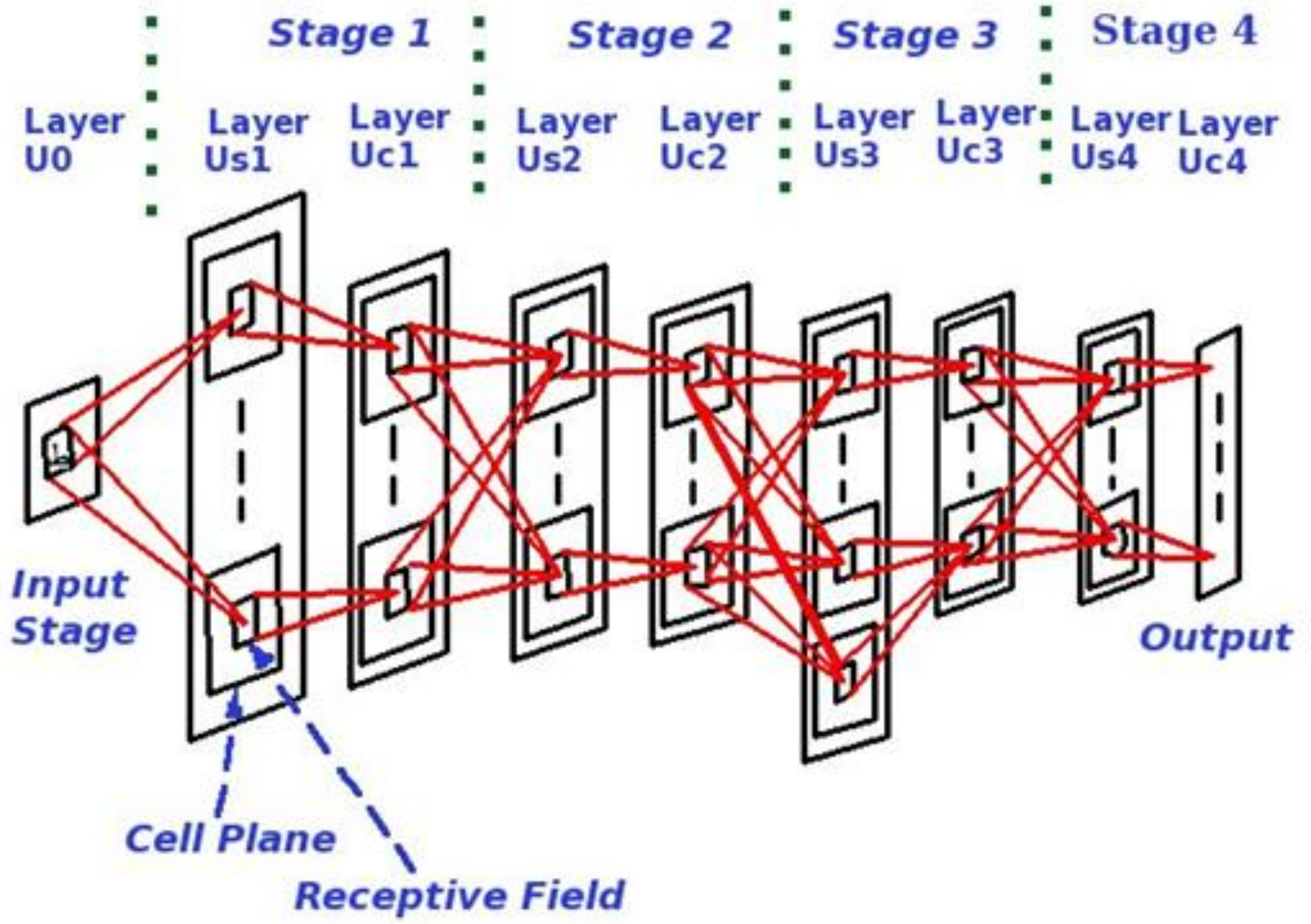
Why use hierarchical multi-layered models?

Argument 2: biological vision is hierachically organised

$$\begin{tikzcd}
\text{object} & \text{trees} & \text{Inferotemporal cortex} \\
\text{object parts} \arrow[u] & \text{bark, leaves, etc.} \arrow[u] & \text{V4: different textures} \\
\text{primitive features} \arrow[u] & \text{oriented edges} \arrow[u] & \text{V1: simple and complex cells} \\
\text{input image} \arrow[u] & \text{forest image} \arrow[u] & \text{photo-receptors retina}
\end{tikzcd}$$



Neocognitron For Handwritten Text



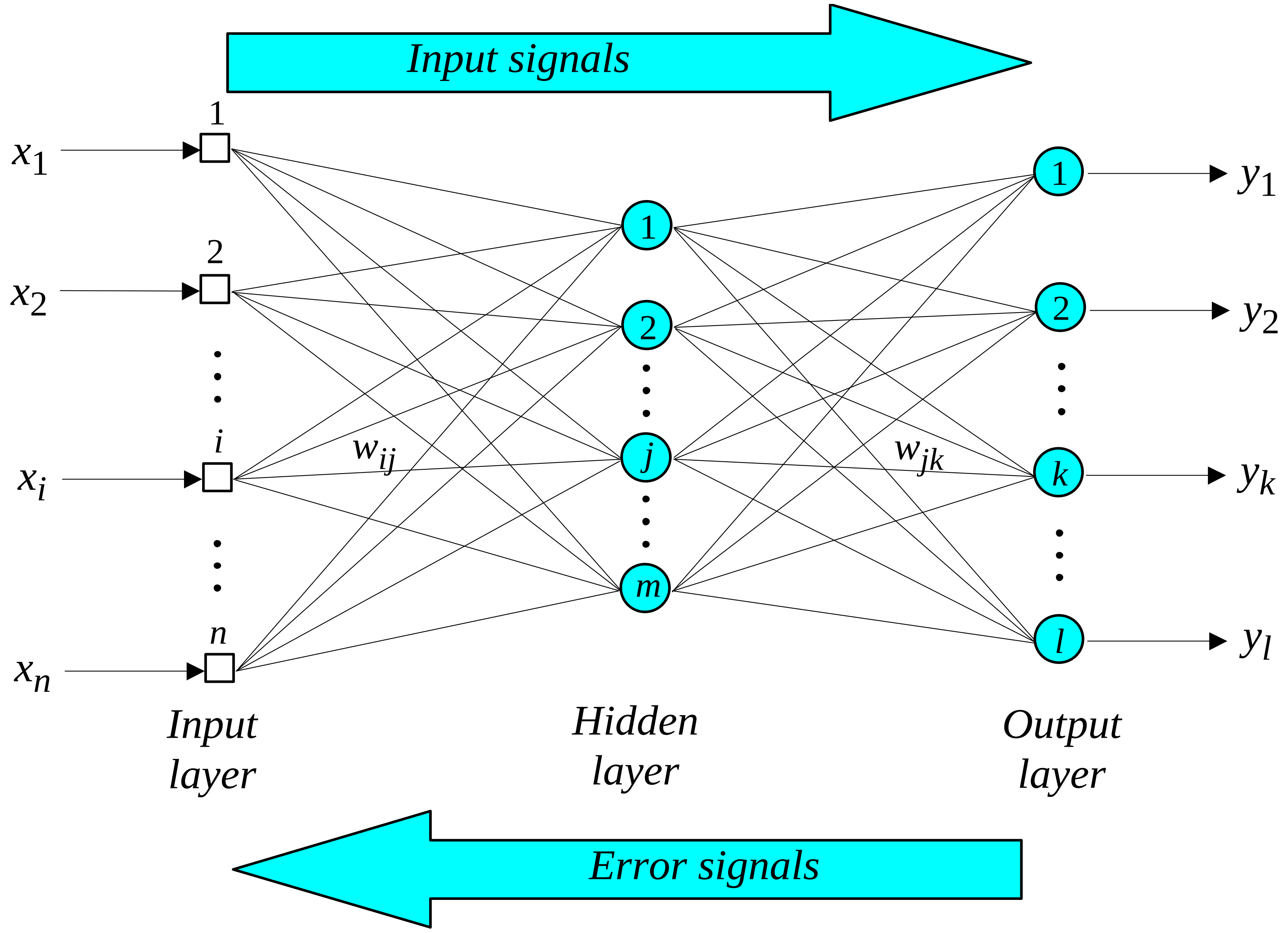
Back-propagation neural network

- Learning in a multilayer network proceeds the same way as for a perceptron.
- A training set of input patterns is presented to the network.
- The network computes its output pattern, and if there is an error – or in other words a difference between actual and desired output patterns – the weights are adjusted to reduce this error.

Back-propagation neural network

- In a back-propagation neural network, the learning algorithm has two phases.
- First, a training input pattern is presented to the network input layer. The network propagates the input pattern from layer to layer until the output pattern is generated by the output layer.
- If this pattern is different from the desired output, an error is calculated and then propagated backwards through the network from the output layer to the input layer. The weights are modified as the error is propagated.

Three-layer back-propagation neural network



The back-propagation training algorithm

Step 1: Initialisation

Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range:

$$\boxed{\left(-\frac{2.4}{F_i},\ +\frac{2.4}{F_i}\right)}$$

where F_i is the total number of inputs of neuron i in the network. The weight initialisation is done on a neuron-by-neuron basis.

Step 2: Activation

Activate the back-propagation neural network by applying inputs $x_1(p)$, $x_2(p)$,…, $x_n(p)$ and desired outputs $y_{d,1}(p)$, $y_{d,2}(p)$,…, $y_{d,n}(p)$.

(*a*) Calculate the actual outputs of the neurons in the hidden layer:

$$\boxed{y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \cdot w_{ij}(p) - \theta_j\right]}$$

where *n* is the number of inputs of neuron *j* in the hidden layer, and *sigmoid* is the *sigmoid* activation function.

Step 2: Activation (continued)

(*b*) Calculate the actual outputs of the neurons in the output layer:

$$\boxed{y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p) \cdot w_{jk}(p) - \theta_k\right]}$$

where *m* is the number of inputs of neuron *k* in the output layer.

Step 3: Weight training

Update the weights in the back-propagation network propagating backward the errors associated with output neurons.

(*a*) Calculate the error gradient for the neurons in the output layer:

$$\boxed{\delta_k(p) = y_k(p) \cdot [1 - y_k(p)] \cdot e_k(p)}$$

where $\boxed{e_k(p) = y_{d,k}(p) - y_k(p)}$

Calculate the weight corrections:

$$\boxed{\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)}$$

Update the weights at the output neurons:

$$\boxed{w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)}$$

Step 3: Weight training (continued)

(*b*) Calculate the error gradient for the neurons in the hidden layer:

$$\boxed{\delta_j(p) = y_j(p) \cdot [1 - y_j(p)] \cdot \sum_{k=1}^{l} \delta_k(p)\, w_{jk}(p)}$$

Calculate the weight corrections:

$$\boxed{\Delta w_{ij}(p) = \alpha \cdot x_i(p) \cdot \delta_j(p)}$$

Update the weights at the hidden neurons:

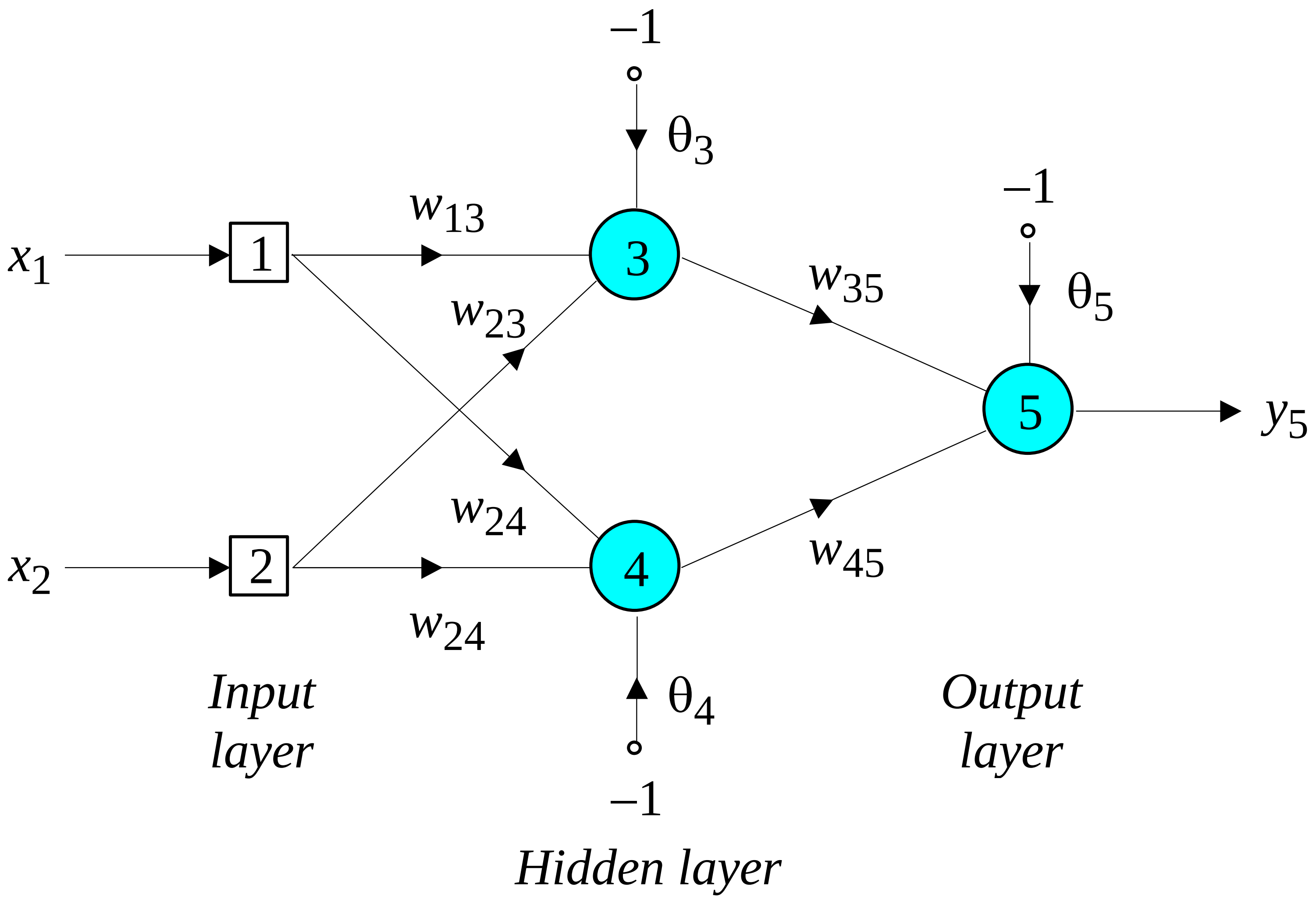
$$\boxed{w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)}$$

Step 4: Iteration

Increase iteration p by one, go back to *Step 2* and repeat the process until the selected error criterion is satisfied.

As an example, we may consider the three-layer back-propagation network. Suppose that the network is required to perform logical operation *Exclusive-OR*. Recall that a single-layer perceptron could not do this operation. Now we will apply the three-layer net.

Three-layer network for solving the Exclusive-OR operation



- The effect of the threshold applied to a neuron in the hidden or output layer is represented by its weight, θ, connected to a fixed input equal to –1.
- The initial weights and threshold levels are set randomly as follows:

  $w_{13} = 0.5$, $w_{14} = 0.9$, $w_{23} = 0.4$, $w_{24} = 1.0$, $w_{35} = -1.2$, $w_{45} = 1.1$, $\theta_3 = 0.8$, $\theta_4 = -0.1$ and $\theta_5 = 0.3$.

- We consider a training set where inputs x_1 and x_2 are equal to 1 and desired output $y_{d,5}$ is 0. The actual outputs of neurons 3 and 4 in the hidden layer are calculated as

$$\boxed{\begin{aligned} y_3 &= sigmoid\,(x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/\left[1 + e^{-(1\cdot 0.5 + 1\cdot 0.4 - 1\cdot 0.8)}\right] = 0.5250 \\ y_4 &= sigmoid\,(x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/\left[1 + e^{-(1\cdot 0.9 + 1\cdot 1.0 + 1\cdot 0.1)}\right] = 0.8808 \end{aligned}}$$

- Now the actual output of neuron 5 in the output layer is determined as:

$$\boxed{y_5 = sigmoid\,(y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/\left[1 + e^{-(-0.5250\cdot 1.2 + 0.8808\cdot 1.1 - 1\cdot 0.3)}\right] = 0.5097}$$

- Thus, the following error is obtained:

$$\boxed{e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097}$$

- The next step is weight training. To update the weights and threshold levels in our network, we propagate the error, *e*, from the output layer backward to the input layer.
- First, we calculate the error gradient for neuron 5 in the output layer:

$$\boxed{\delta_5 = y_5\,(1 - y_5)\,e = 0.5097 \cdot (1 - 0.5097) \cdot (-0.5097) = -0.1274}$$

- Then we determine the weight corrections assuming that the learning rate parameter, α, is equal to 0.1:

$$\boxed{\begin{aligned}\Delta w_{35} &= \alpha \cdot y_3 \cdot \delta_5 = 0.1 \cdot 0.5250 \cdot (-0.1274) = -0.0067\\ \Delta w_{45} &= \alpha \cdot y_4 \cdot \delta_5 = 0.1 \cdot 0.8808 \cdot (-0.1274) = -0.0112\\ \Delta\theta_5 &= \alpha \cdot (-1) \cdot \delta_5 = 0.1 \cdot (-1) \cdot (-0.1274) = -0.0127\end{aligned}}$$

- Next we calculate the error gradients for neurons 3 and 4 in the hidden layer:

$$\boxed{\begin{aligned}\delta_3 &= y_3(1-y_3)\cdot\delta_5\cdot w_{35} = 0.5250\cdot(1-0.5250)\cdot(-0.1274)\cdot(-1.2) = 0.0381\\ \delta_4 &= y_4(1-y_4)\cdot\delta_5\cdot w_{45} = 0.8808\cdot(1-0.8808)\cdot(-0.127\,4)\cdot 1.1 = -0.0147\end{aligned}}$$

- We then determine the weight corrections:

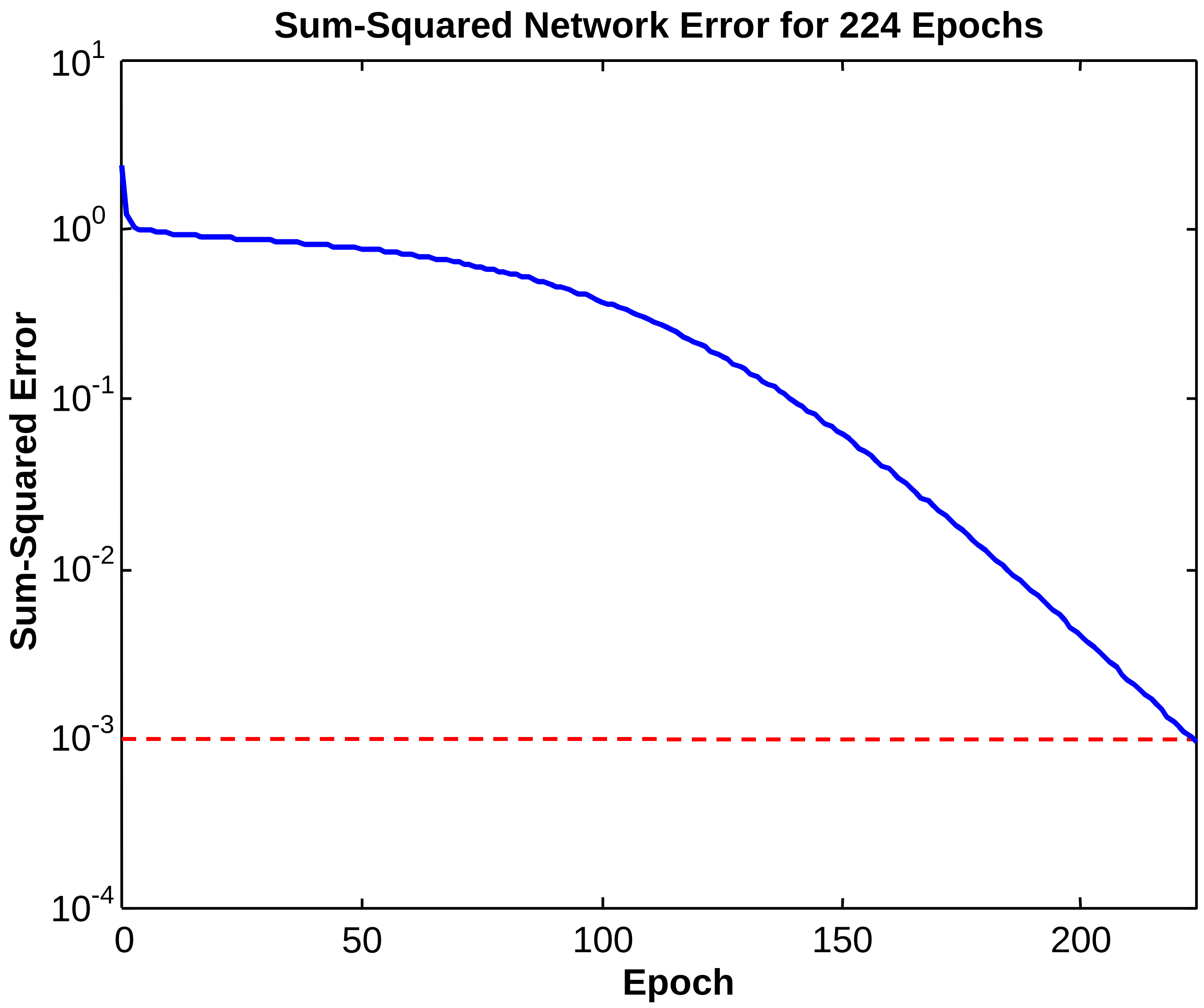
$$\boxed{\begin{aligned}\Delta w_{13} &= \alpha\cdot x_1\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038\\ \Delta w_{23} &= \alpha\cdot x_2\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038\\ \Delta\theta_3 &= \alpha\cdot(-1)\cdot\delta_3 = 0.1\cdot(-1)\cdot 0.0381 = -0.0038\\ \Delta w_{14} &= \alpha\cdot x_1\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015\\ \Delta w_{24} &= \alpha\cdot x_2\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015\\ \Delta\theta_4 &= \alpha\cdot(-1)\cdot\delta_4 = 0.1\cdot(-1)\cdot(-0.0147) = 0.0015\end{aligned}}$$

- At last, we update all weights and threshold:

$$
\boxed{
\begin{aligned}
w_{13} &= w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038 \\
w_{14} &= w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985 \\
w_{23} &= w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038 \\
w_{24} &= w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985 \\
w_{35} &= w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067 \\
w_{45} &= w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888 \\
\theta_3 &= \theta_3 + \Delta\theta_3 = 0.8 - 0.0038 = 0.7962 \\
\theta_4 &= \theta_4 + \Delta\theta_4 = -0.1 + 0.0015 = -0.0985 \\
\theta_5 &= \theta_5 + \Delta\theta_5 = 0.3 + 0.0127 = 0.3127
\end{aligned}
}
$$

- The training process is repeated until the sum of squared errors is less than 0.001.

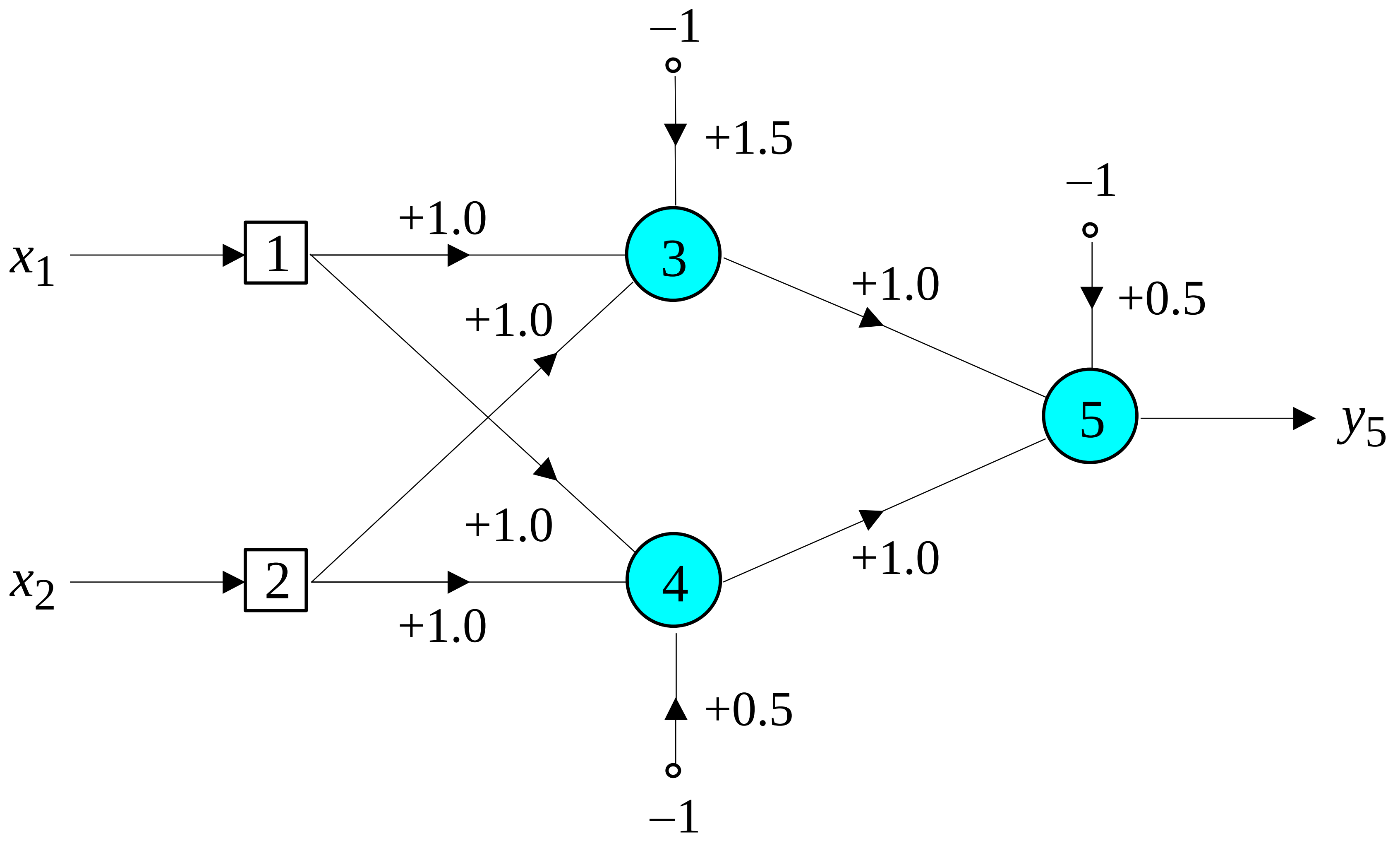
Learning curve for operation *Exclusive-OR*



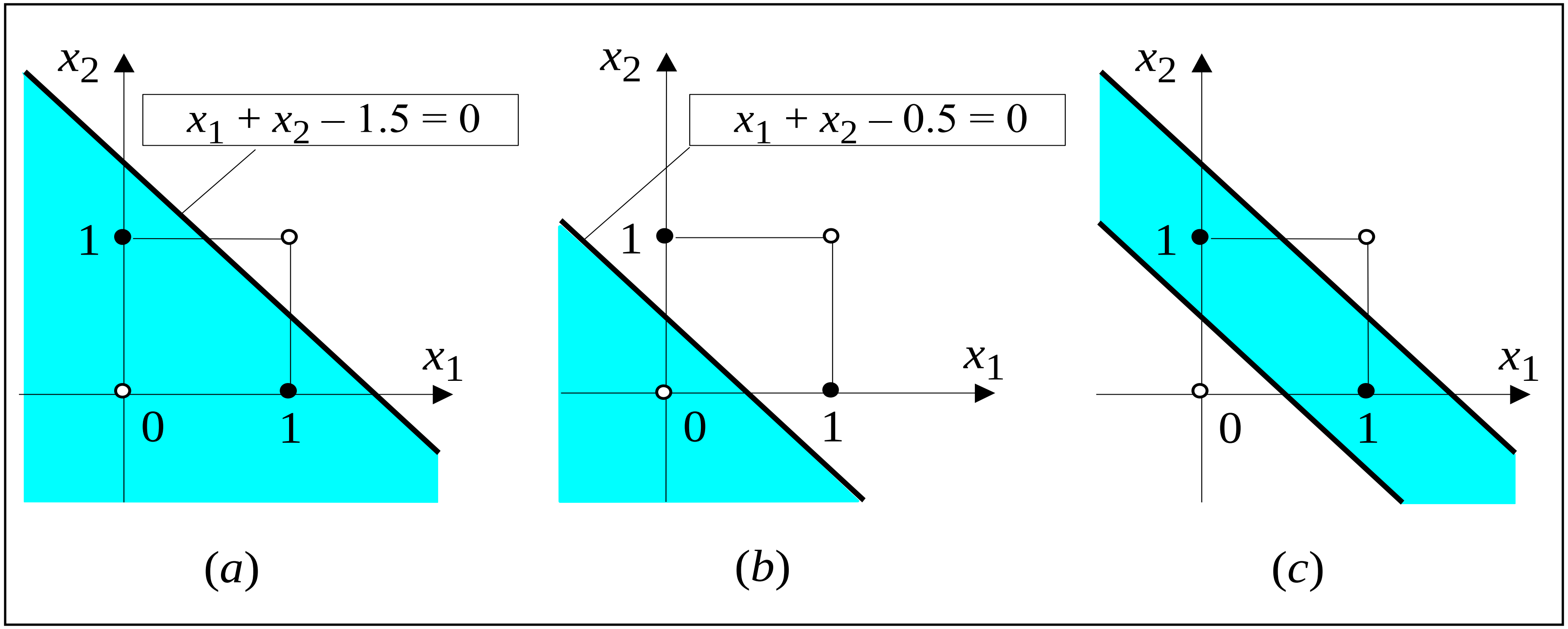
Final results of three-layer network learning

Inputs		Desired output	Actual output	Error	Sum of squared errors
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	–0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	–0.0175	

Network represented by McCulloch-Pitts model for solving the *Exclusive-OR* operation



Decision boundaries



(*a*) Decision boundary constructed by hidden neuron 3;
(*b*) Decision boundary constructed by hidden neuron 4;
(*c*) Decision boundaries constructed by the complete three-layer network

Accelerated learning in multilayer neural networks

- A multilayer network learns much faster when the sigmoidal activation function is represented by a **hyperbolic tangent**:

$$\boxed{Y^{tanh} = \frac{2a}{1 + e^{-bX}} - a}$$

where *a* and *b* are constants.

Suitable values for *a* and *b* are:
$a = 1.716$ and $b = 0.667$

Accelerated learning in multilayer neural networks

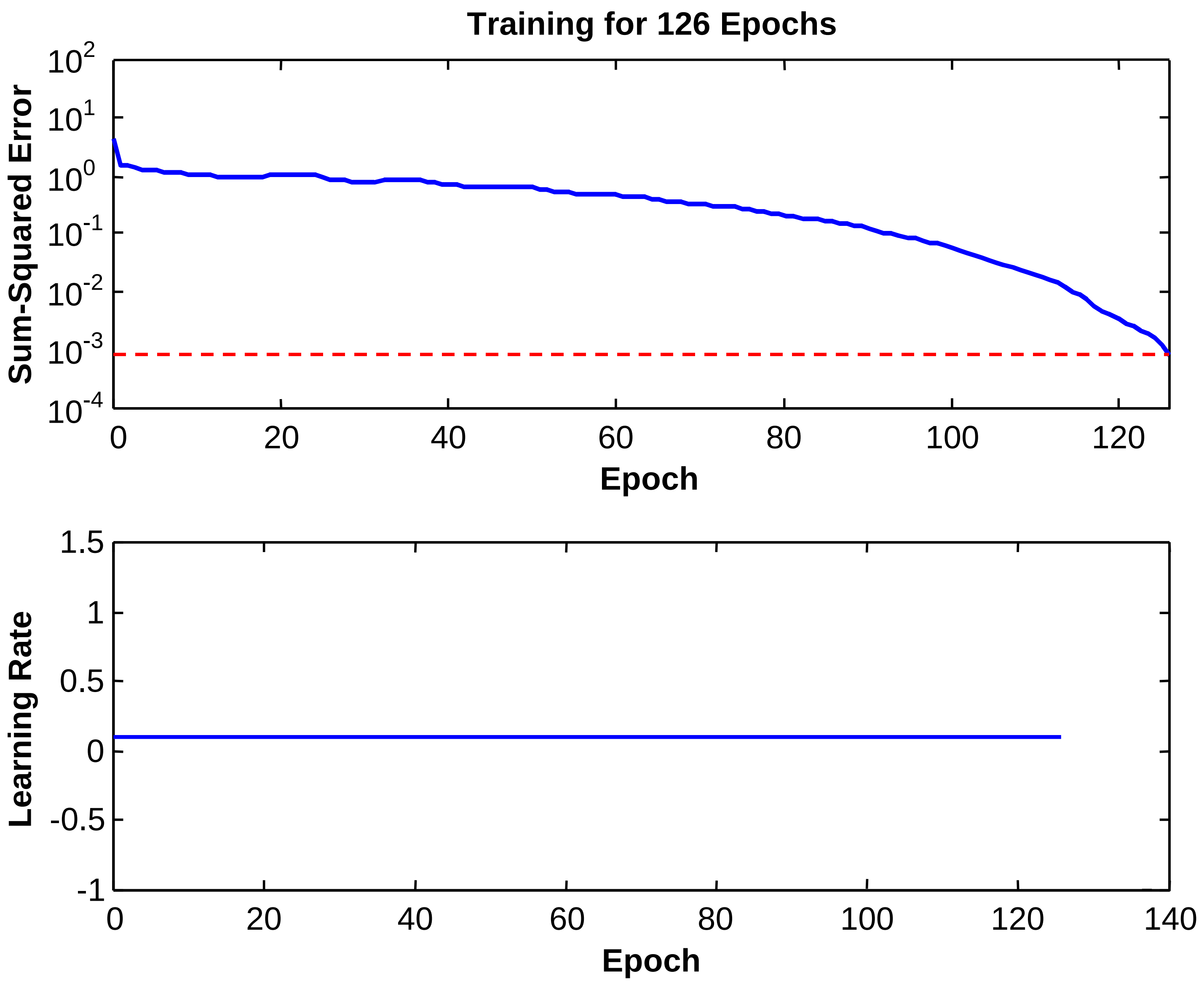
- We also can accelerate training by including a **momentum term** in the delta rule:

$$\boxed{\Delta w_{jk}(p) = \beta \cdot \Delta w_{jk}(p-1) + \alpha \cdot y_j(p) \cdot \delta_k(p)}$$

where β is a positive number ($0 \leq \beta < 1$) called the **momentum constant**. Typically, the momentum constant is set to 0.95.

This equation is called the **generalised delta rule**.

Learning with momentum for operation *Exclusive-OR*



Learning with adaptive learning rate

To accelerate the convergence and yet avoid the danger of instability, we can apply two heuristics:

Heuristic 1

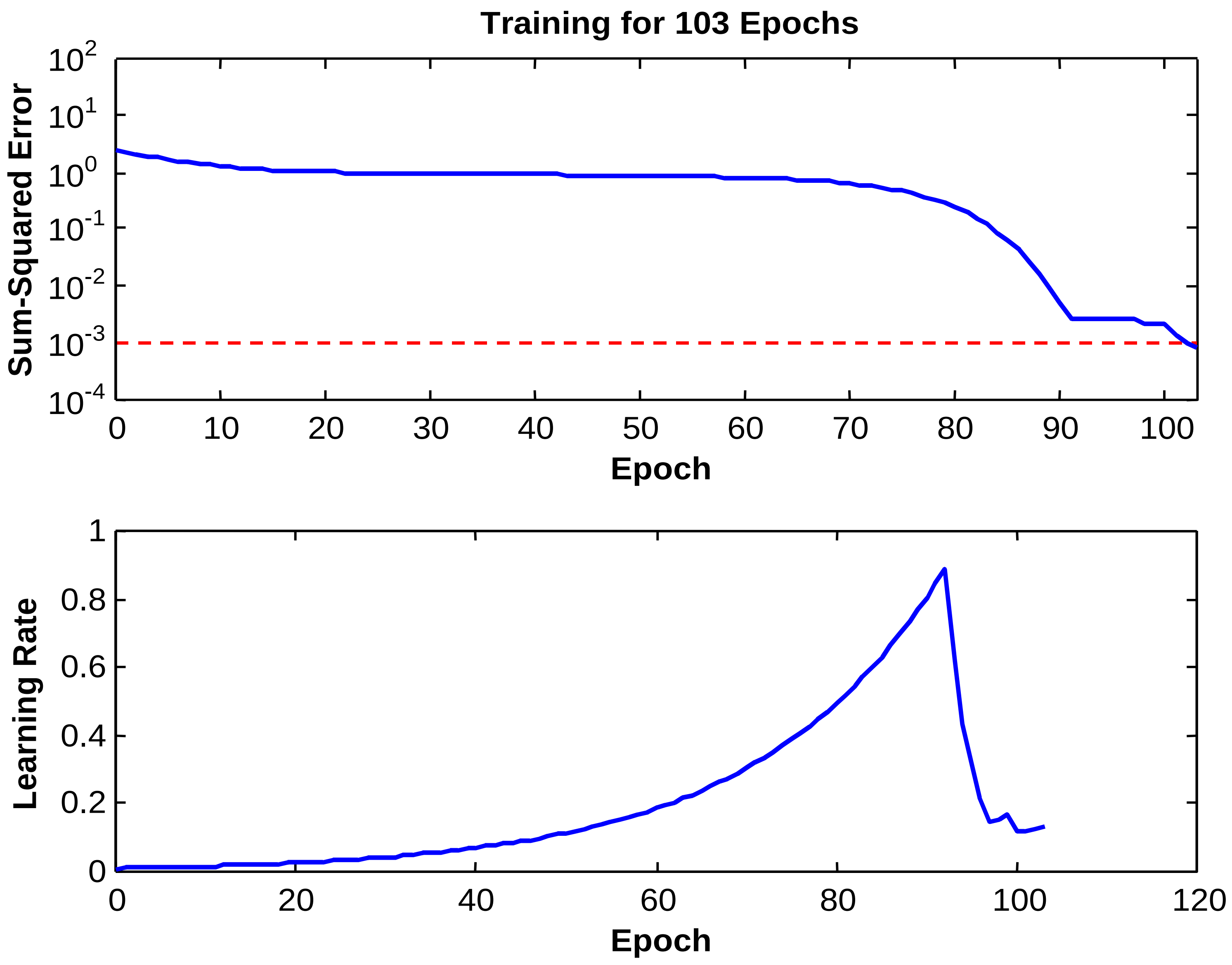
If the change of the sum of squared errors has the same algebraic sign for several consequent epochs, then the learning rate parameter, α, should be increased.

Heuristic 2

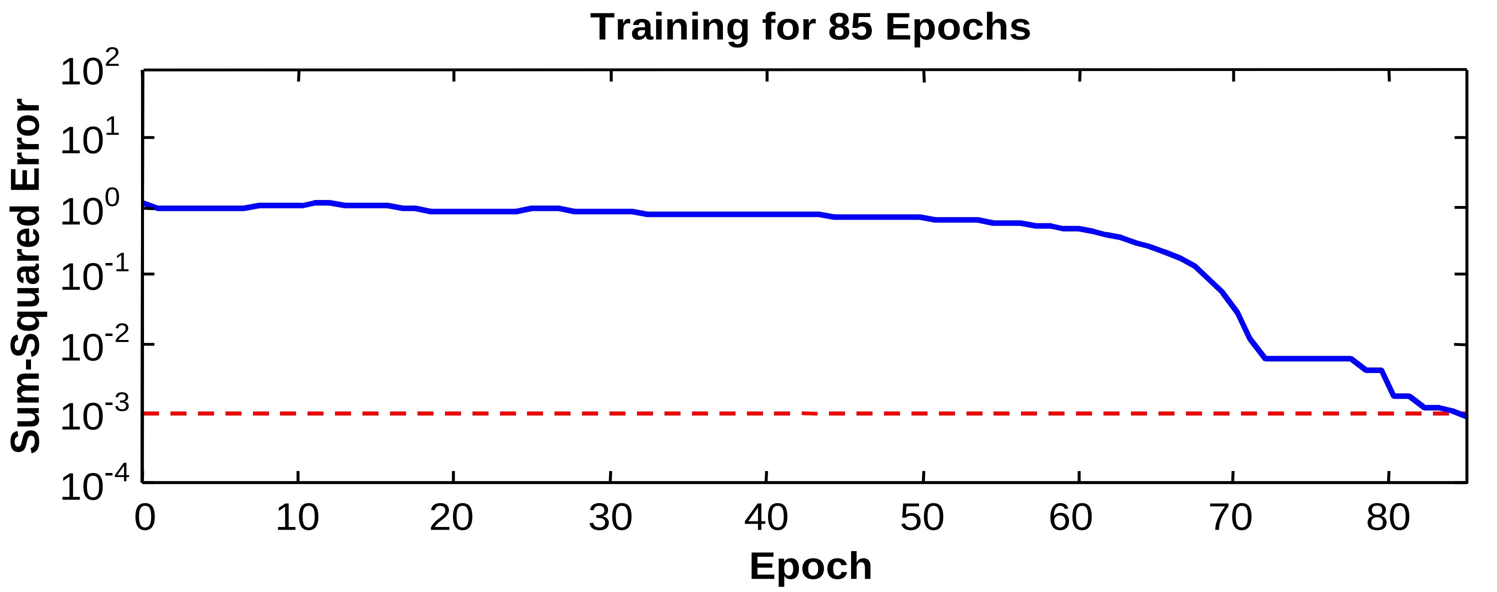
If the algebraic sign of the change of the sum of squared errors alternates for several consequent epochs, then the learning rate parameter, α, should be decreased.

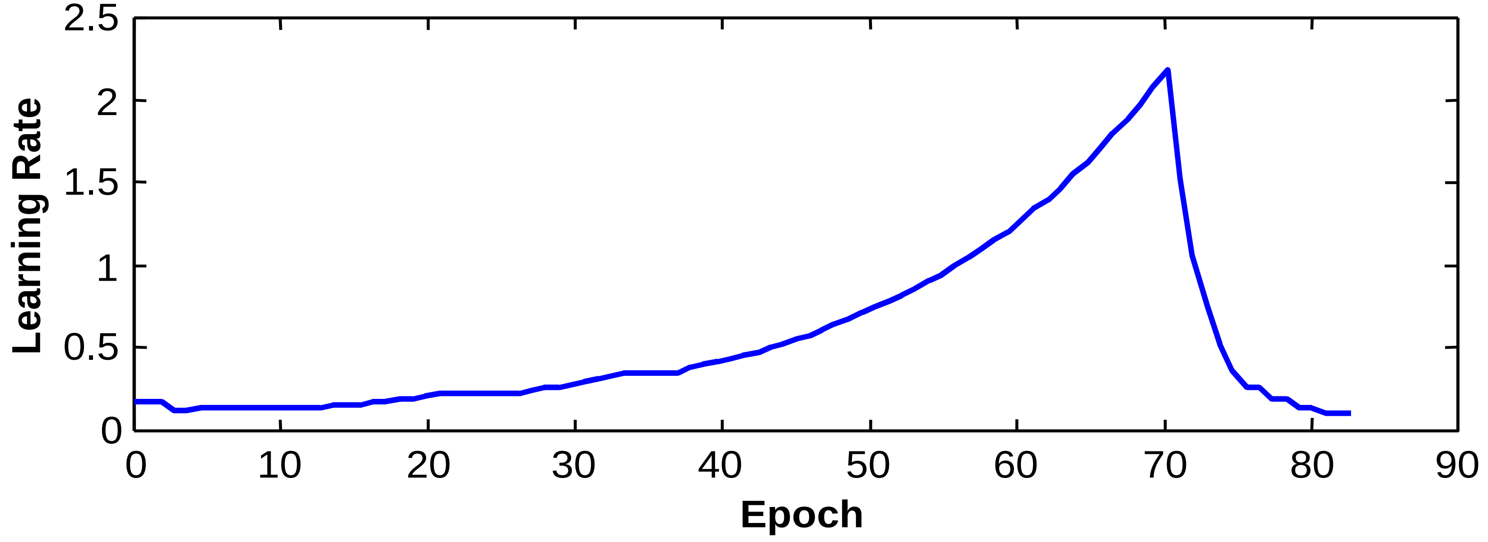
- Adapting the learning rate requires some changes in the back-propagation algorithm.
- If the sum of squared errors at the current epoch exceeds the previous value by more than a predefined ratio (typically 1.04), the learning rate parameter is decreased (typically by multiplying by 0.7) and new weights and thresholds are calculated.
- If the error is less than the previous one, the learning rate is increased (typically by multiplying by 1.05).

Learning with adaptive learning rate



Learning with momentum and adaptive learning rate





Accelerated learning in multilayer neural networks

- Back propagation using gradient descent often converges very slowly or not at all.

- On large-scale problems its success depends on user-specified learning rate and momentum parameters.

- Conjugate gradient algorithm is another approach to adjust weight values using the gradient during the backward propagation of errors through the network.

- Conjugate gradient algorithm takes a more direct path to the optimal set of weight values. Usually, conjugate gradient is significantly faster and more robust than gradient descent. Conjugate gradient also does not require the user to specify learning rate and momentum parameters.

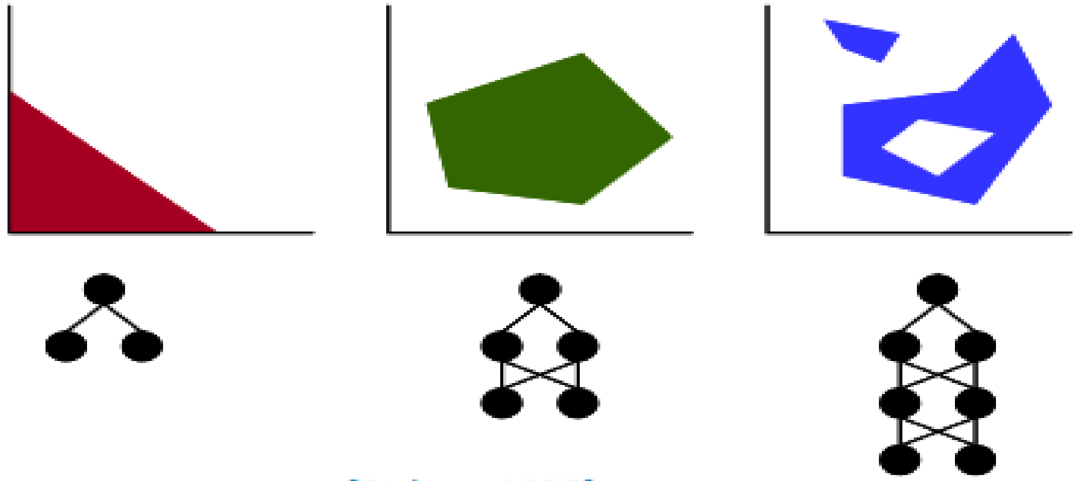
Accelerated learning in multilayer neural networks

- The scaled conjugate gradient algorithm compute the optimal step size in the search direction without having to perform the computationally expensive line search used by the traditional conjugate gradient algorithm.
- Tests performed by Moller show the scaled conjugate gradient algorithm converging up to twice as fast as traditional conjugate gradient and up to 20 times as fast as backpropagation using gradient descent.
- Moller’s tests also showed that scaled conjugate gradient failed to converge less often than traditional conjugate gradient or backpropagation using gradient descent.

Limiting network complexity

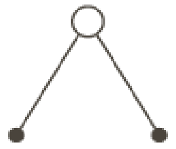
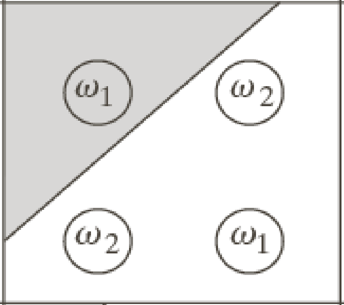
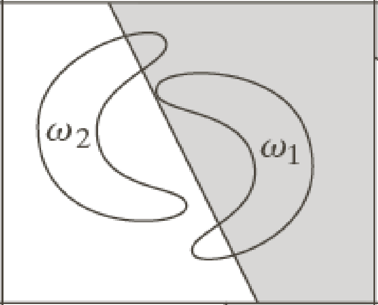
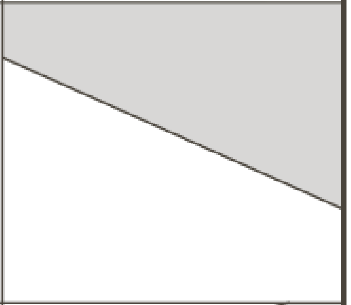
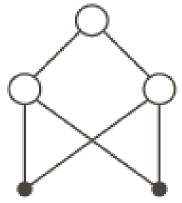
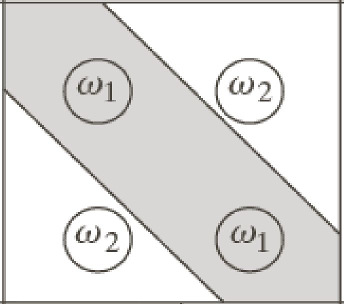
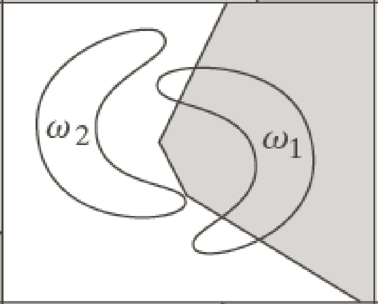
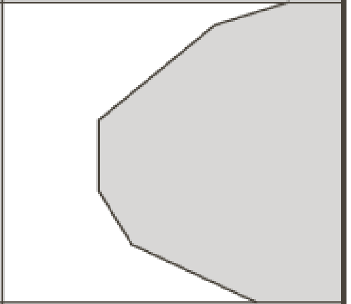
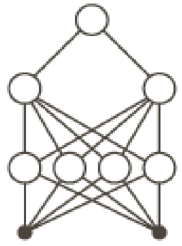
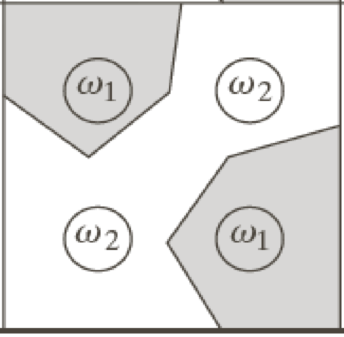
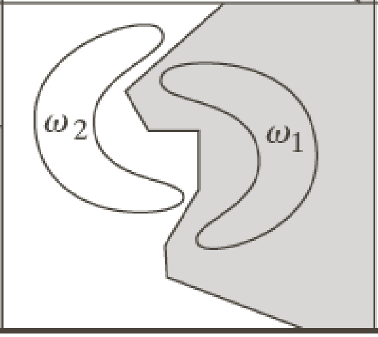
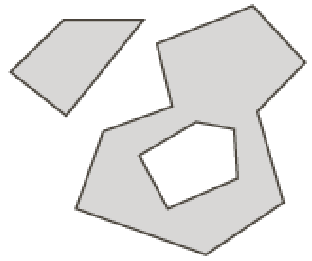
Number of hidden layers

- Given a sufficiently large number of hidden neurons, a two-layer MLP can approximate any continuous function arbitrarily well
  - The example below shows that a combination of four hidden neurons can produce a "bump" at the output space of a two-layered MLP
  - A large number of "bumps" can approximate any surface arbitrarily well
- Nonetheless, the addition of extra hidden layers may allow the MLP to approximate more efficiently, i.e., with fewer weights [Bishop, 1995]



[Bishop, 1995]

Number of Hidden Layer

Network structure	Type of decision region	Solution to exclusive-OR problem	Classes with meshed regions	Most general decision surface shapes
Single layer 	Single hyperplane			
Two layers 	Open or closed convex regions			
Three layers 	Arbitrary (complexity limited by the number of nodes)			

Limiting network complexity

Number of hidden neurons

- While the number of inputs and outputs are dictated by the problem, the number of hidden units N_H is not related so explicitly to the application domain
  - N_H determines the degrees of freedom or expressive power of the model
  - A small N_H may not be sufficient to model complex I/O mappings
  - A large N_H may overfit the training data and prevent the network from generalizing to new examples
- Despite a number of "rules of thumb" published in the literature, a-priori determination of an appropriate N_H is an unsolved problem
  - The "optimal" N_H depends on multiple factors, including number of examples, level of noise in the training set, complexity of the classification problem, number of inputs and outputs, activation functions and training algorithm.
  - In practice, several MLPs are trained and evaluated in order to determine an appropriate N_H
- A number of adaptive approaches have also been proposed
  - **Constructive** approaches start with a small network and incrementally add hidden neurons (e.g., cascade correlation),
  - **Pruning** approaches, which start with a relatively large network and incrementally remove weights (e.g., optimal brain damage)

Tricks of the trade

Weight decay

- To prevent the weights from growing too large (a sign of over-training) it is convenient to add a decay term of the form

$$w(n+1) = (1-\epsilon)w(n)$$

  - Weights that are not needed eventually decay to zero, whereas necessary weights are continuously updated by back-prop
- Weight decay is a simple form of regularization, which encourages smoother network mappings [Bishop, 1995]

Early stopping

- Early stopping can be used to prevent the MLP from over-fitting the training set
- The stopping point may be determined by monitoring the sum-squared-error of the MLP on a validation set during training

Training with noise (jitter)

- Training with noise prevents the MLP from approximating the training set too closely, which leads to improved generalization

Tricks of the trade

Activation function

- An MLP trained with backprop will generally train faster if the activation function is anti-symmetric: $f(-x) = -f(x)$ (e.g., the hyperbolic tangent)

Target values

- Obviously, it is important that the target values are within the dynamic range of the activation function, otherwise the neuron will be unable to produce them
- In addition, it is recommended that the target values are not the asymptotic values of the activation function; otherwise backprop will tend to drive the neurons into saturation, slowing down the learning process
  - The slope of the activation function, which is proportional to Δw, becomes zero at $\pm\infty$

Input normalization

- Input variables should be preprocessed so that their mean value is zero or small compared to the variance
- Input variables should be uncorrelated (use PCA to accomplish this)
- Input variables should have the same variance (use Fukunaga's whitening transform)

Tricks of the trade

Initial weights

- Initial random weights should be small to avoid driving the neurons into saturation
  - HO weights should be made larger than IH weights since they carry the back propagated error
  - If the initial HO weights are very small, the weight changes at the IH layer will initially be very small, slowing the training process

Weight updates

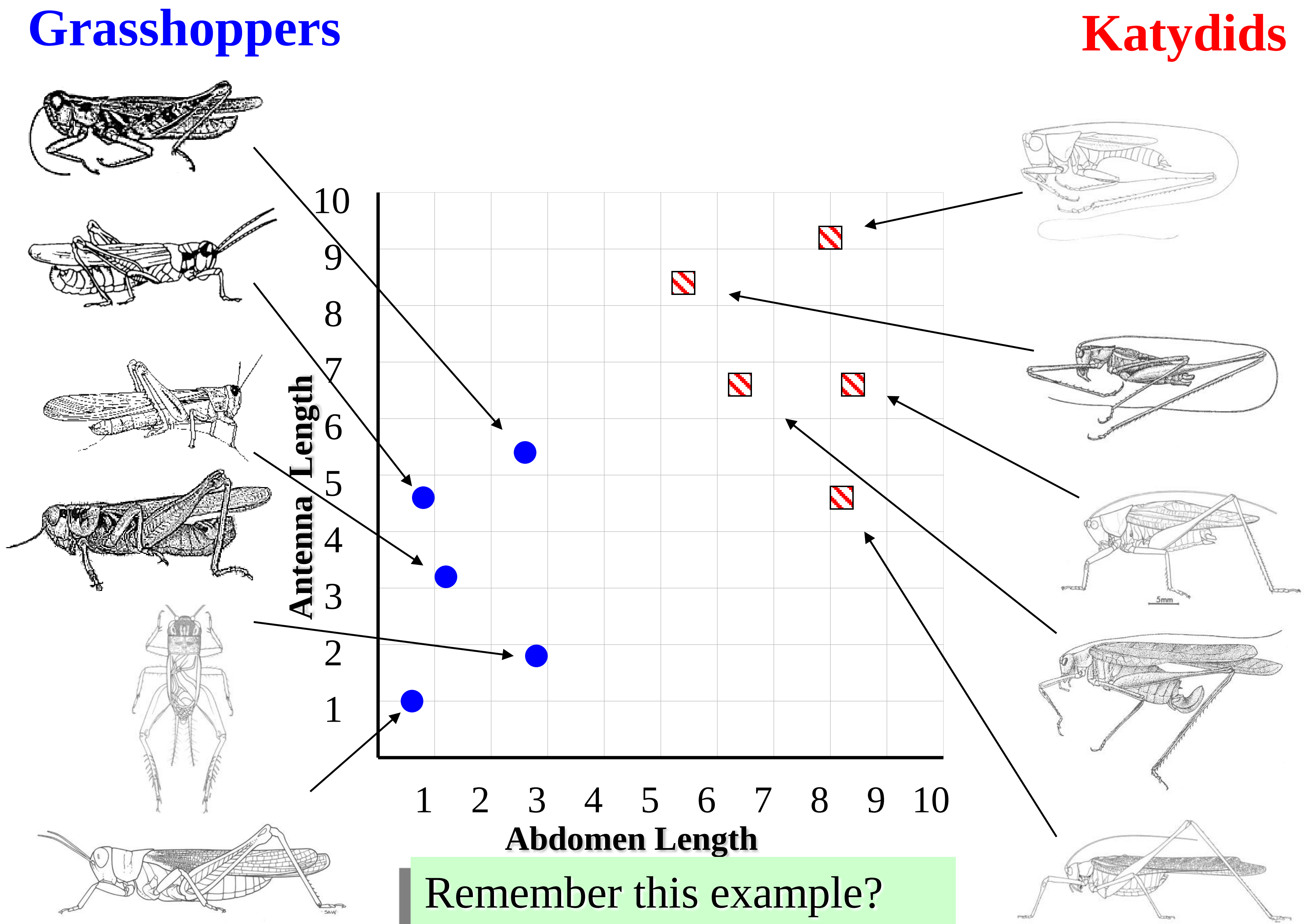
- The derivation of backprop was based on one training example but, in practice, the data set contains a large number of examples
- There are two basic approaches for updating the weights during training
  - On-line training: weights are updated after presentation of each example
  - Batch training: weights are updated after presentation of all the examples (we store the Δw for each example, and add them up to the weight after all the examples have been presented)
- Batch training is recommended
  - Batch training uses the TRUE steepest descent direction
  - On-line training achieves lower errors earlier in training but only because the weights are updated n (# examples) times faster than in batch mode
  - On-line training is sensitive to the ordering of examples

Naïve Bayes Classifier

Thomas Bayes

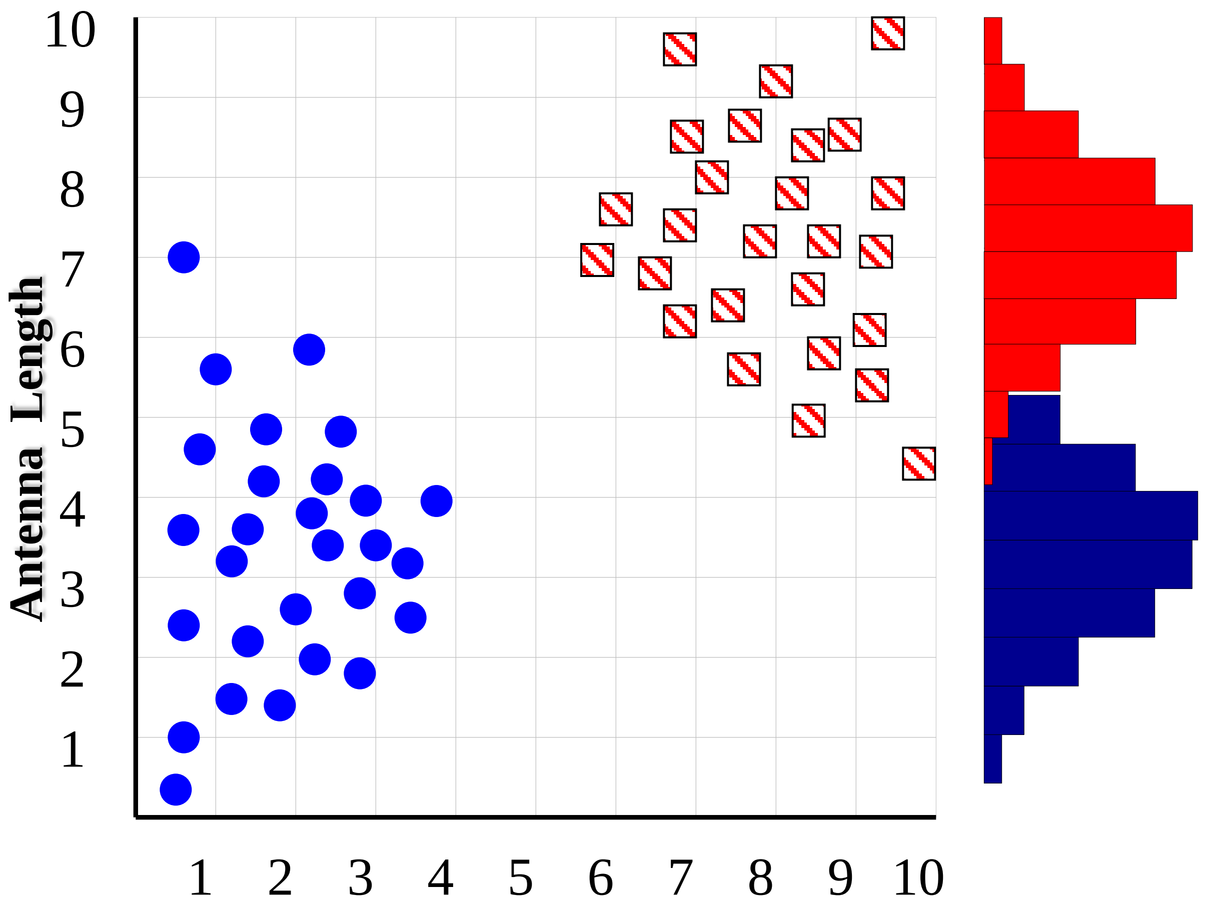
1702 - 1761

We will start off with a visual intuition, before looking at the math…



Remember this example?
Let’s get lots more data…

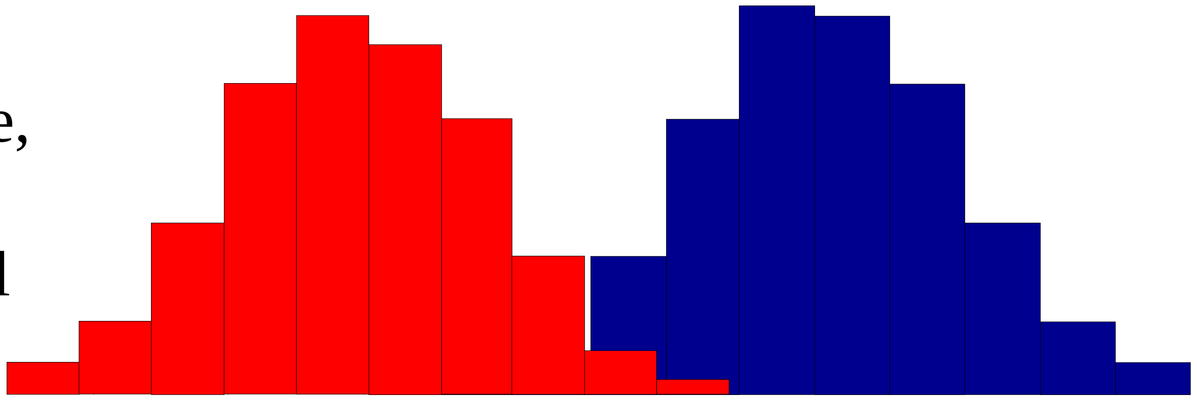
With a lot of data, we can build a histogram. Let us just build one for "Antenna Length" for now…



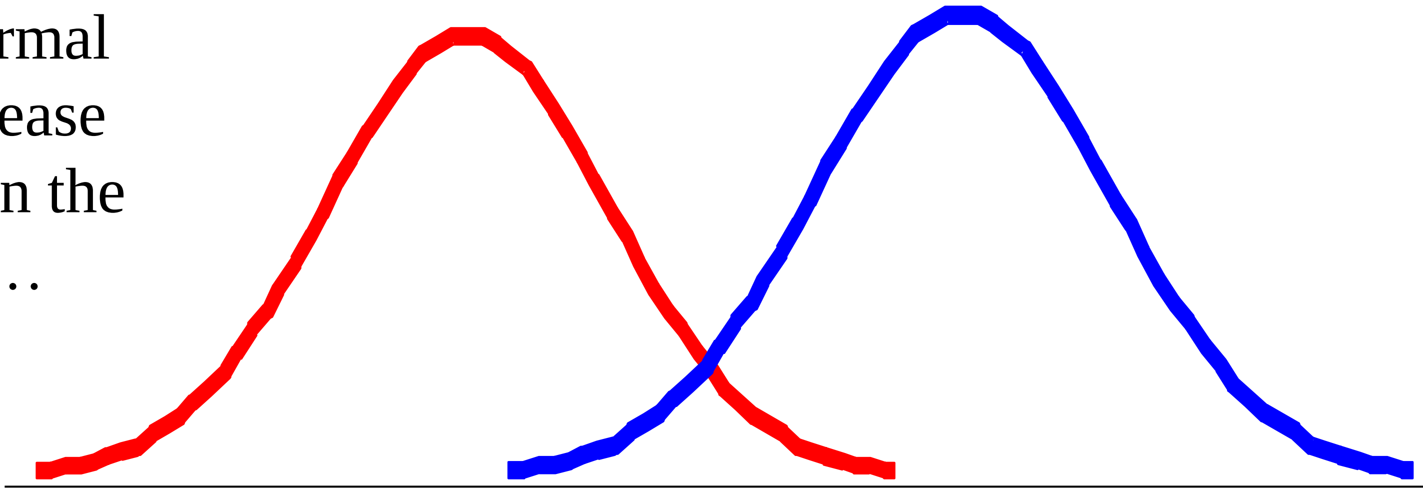
▨ **Katydids**

● **Grasshoppers**

We can leave the histograms as they are, or we can summarize them with two normal distributions.



Let us us two normal distributions for ease of visualization in the following slides…

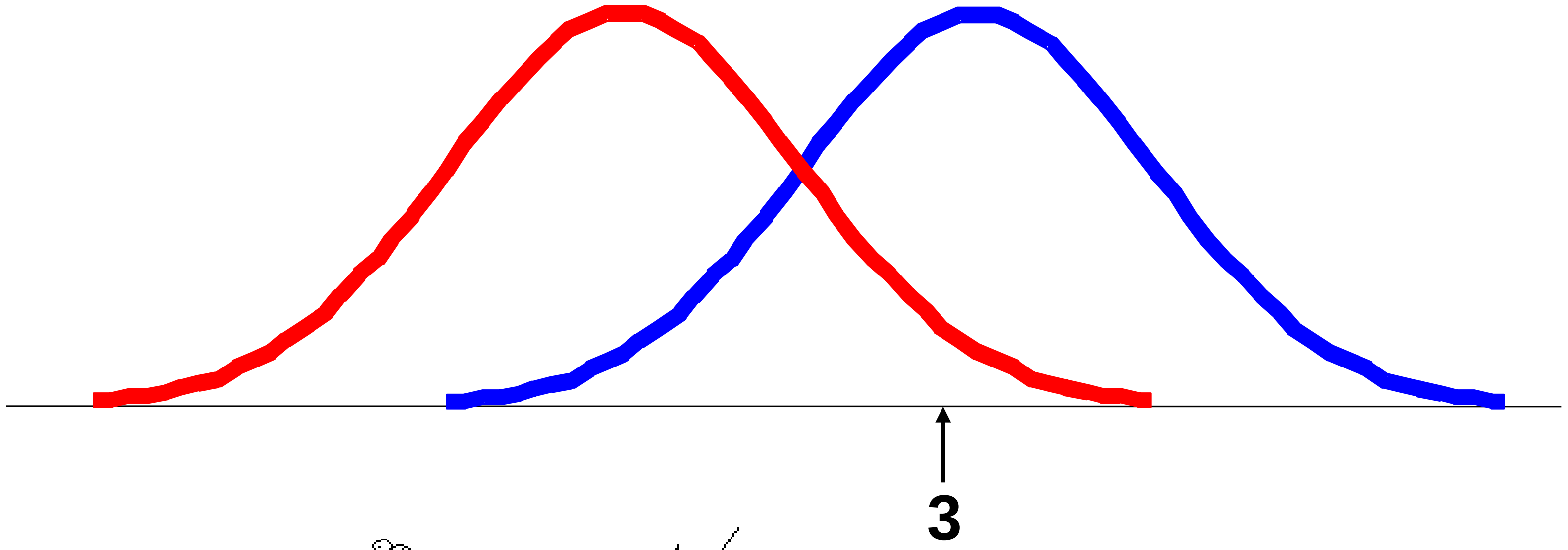


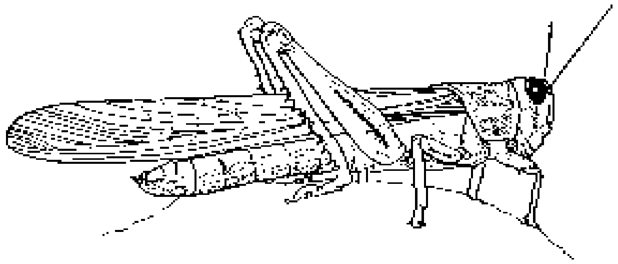
• We want to classify an insect we have found. Its antennae are 3 units long. How can we classify it?

• We can just ask ourselves, give the distributions of antennae lengths we have seen, is it more *probable* that our insect is a **Grasshopper** or a **Katydid**.

• There is a formal way to discuss the most *probable* classification…

$p(c_j \mid d)$ = probability of class c_j, *given* that we have observed d



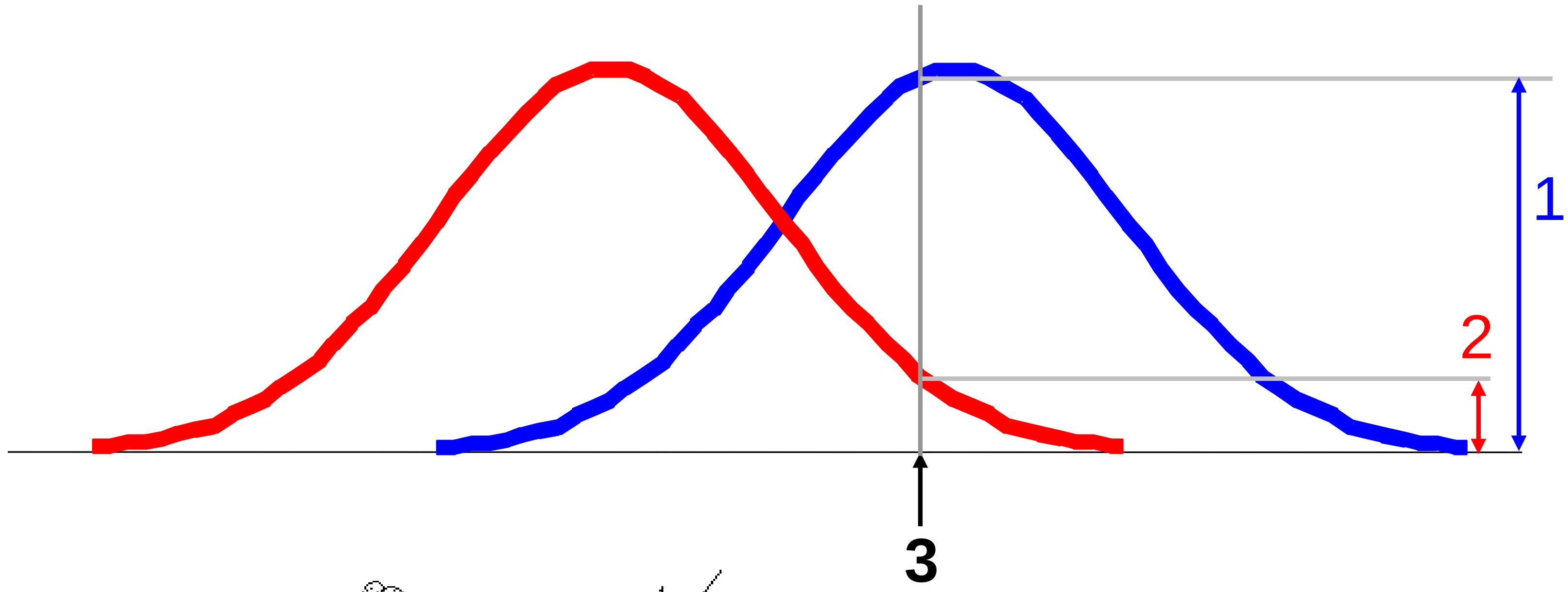


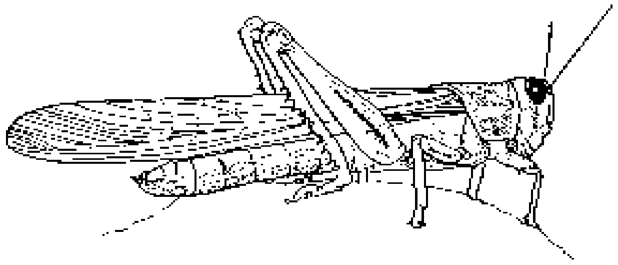
Antennae length is **3**

$p(c_j | d)$ = probability of class c_j, given that we have observed d

P(**Grasshopper** | **3**) = 10 / (10 + 2) = 0.833

P(**Katydid** | **3**) = 2 / (10 + 2) = 0.166



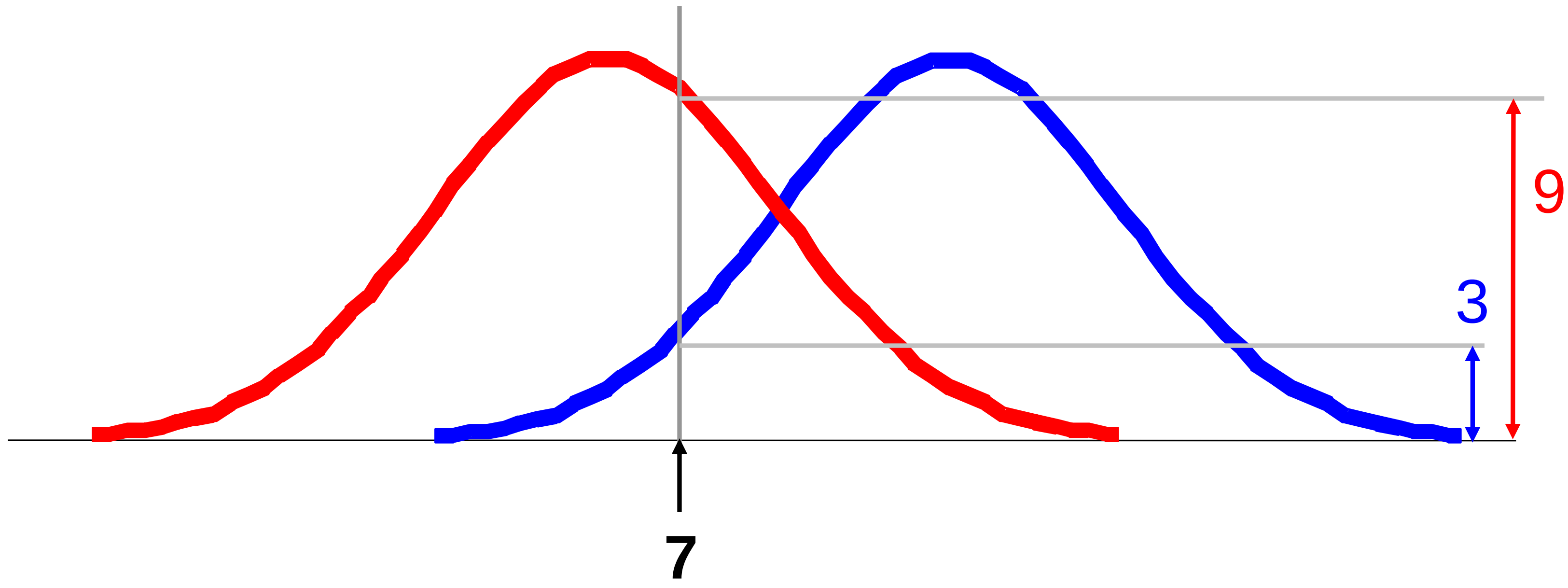


Antennae length is **3**

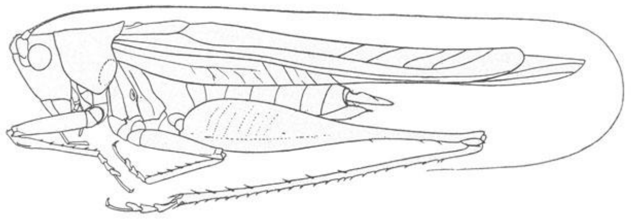
$p(c_j | d)$ = probability of class c_j, given that we have observed d

P(**Grasshopper** | **7**) = 3 / (3 + 9) = 0.250

P(**Katydid** | **7**) = 9 / (3 + 9) = 0.750



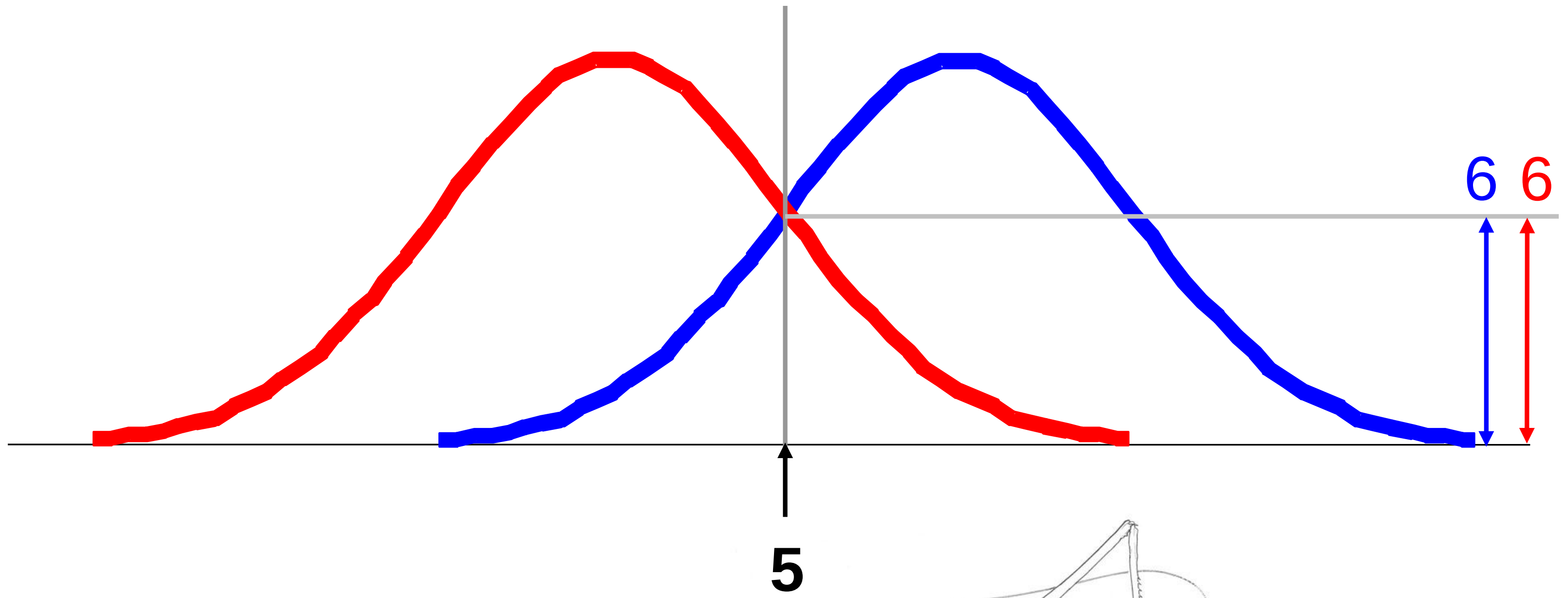
Antennae length is **7**



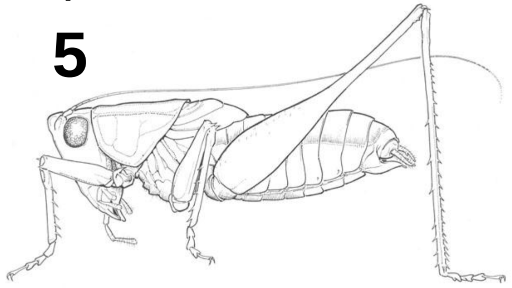
$p(c_j | d)$ = probability of class c_j, given that we have observed d

P(**Grasshopper** | **5**) = 6 / (6 + 6) = 0.500

P(**Katydid** | **5**) = 6 / (6 + 6) = 0.500



Antennae length is **5**



Probability Basics

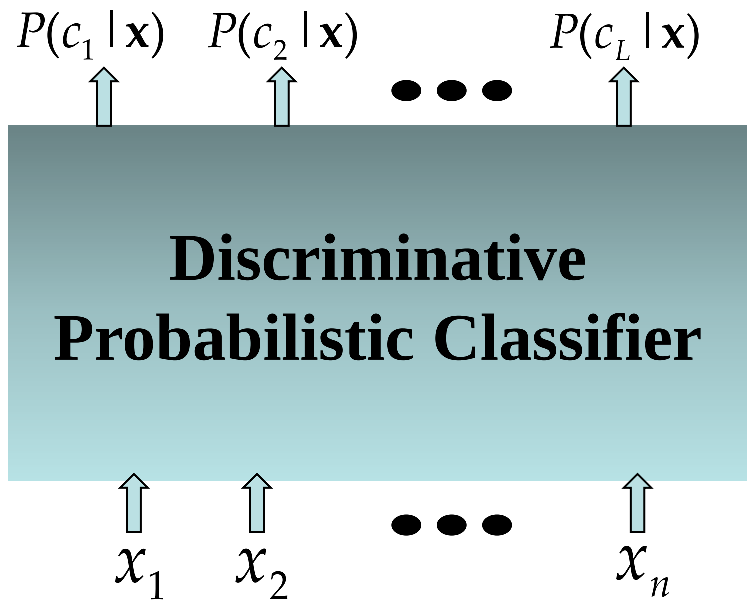
- Prior, conditional and joint probability for random variables
  - Prior probability: $P(X)$
  - Conditional probability: $P(X_1 | X_2), P(X_2 | X_1)$
  - Joint probability: $\mathbf{X} = (X_1, X_2), P(\mathbf{X}) = P(X_1, X_2)$
  - Relationship: $P(X_1, X_2) = P(X_2 | X_1)P(X_1) = P(X_1 | X_2)P(X_2)$
  - Independence: $P(X_2 | X_1) = P(X_2), P(X_1 | X_2) = P(X_1), P(X_1, X_2) = P(X_1)P(X_2)$
- Bayesian Rule

$$P(C | \mathbf{X}) = \frac{P(\mathbf{X} | C)P(C)}{P(\mathbf{X})} \quad Posterior = \frac{Likelihood \times Prior}{Evidence}$$

Probabilistic Classification

- Establishing a probabilistic model for classification
  - **Discriminative model**

$$P(C \mid \mathbf{X}) \quad C = c_1, \cdots, c_L, \mathbf{X} = (X_1, \cdots, X_n)$$

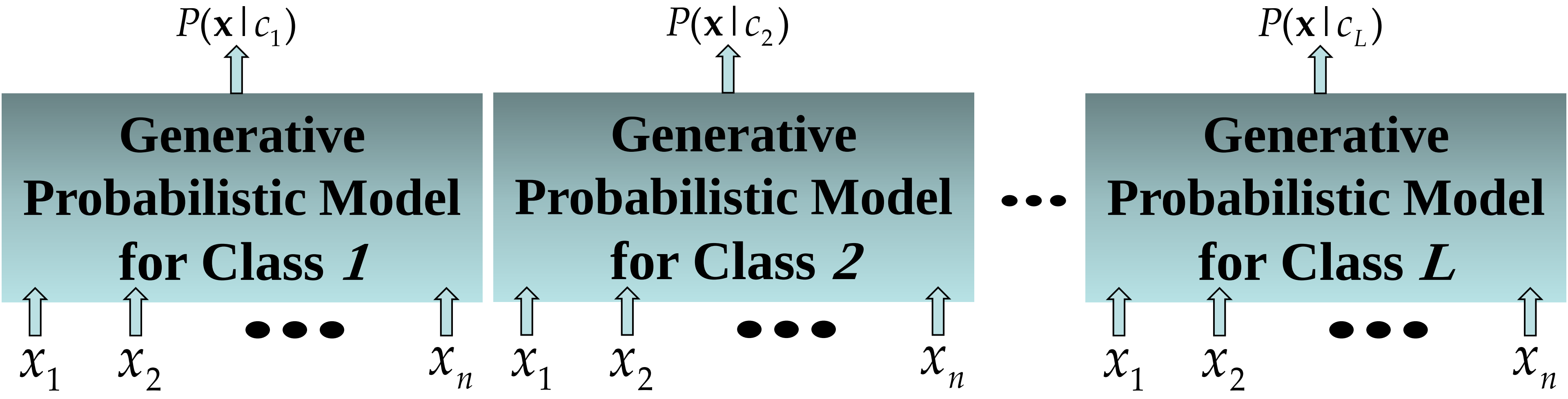


$$\mathbf{x} = (x_1, x_2, \cdots, x_n)$$

Probabilistic Classification

- Establishing a probabilistic model for classification (cont.)
  - **Generative model**

$$P(\mathbf{X}|C) \quad C = c_1, \cdots, c_L, \mathbf{X} = (X_1, \cdots, X_n)$$



$P(\mathbf{x}|c_1)$
$P(\mathbf{x}|c_2)$
$P(\mathbf{x}|c_L)$
Generative Probabilistic Model for Class *1*
Generative Probabilistic Model for Class *2*
Generative Probabilistic Model for Class *L*
x_1 x_2 ••• x_n

$$\mathbf{x} = (x_1, x_2, \cdots, x_n)$$

Probabilistic Classification

- MAP classification rule
  - **MAP**: **M**aximum **A** **P**osterior
  - Assign x to c^* if

$$P(C=c^* \mid \mathbf{X}=\mathbf{x}) > P(C=c \mid \mathbf{X}=\mathbf{x}) \quad c \neq c^*,\ c=c_1,\cdots,c_L$$

- Generative classification with the MAP rule
  - Apply Bayesian rule to convert them into posterior probabilities

$$P(C=c_i \mid \mathbf{X}=\mathbf{x}) = \frac{P(\mathbf{X}=\mathbf{x} \mid C=c_i)P(C=c_i)}{P(\mathbf{X}=\mathbf{x})}$$
$$\propto P(\mathbf{X}=\mathbf{x} \mid C=c_i)P(C=c_i)$$
$$\text{for } i=1,2,\cdots,L$$

  - Then apply the MAP rule

Naïve Bayes

- Bayes classification

$$P(C|\mathbf{X}) \propto P(\mathbf{X}|C)P(C) = P(X_1,\cdots,X_n|C)P(C)$$

Difficulty: learning the joint probability $P(X_1,\cdots,X_n|C)$

- Naïve Bayes classification
  - Assumption that all input features are conditionally independent!

$$\begin{aligned} P(X_1,X_2,\cdots,X_n|C) &= \overline{P(X_1|X_2,\cdots,X_n,C)}P(X_2,\cdots,X_n|C) \\ &= \overline{P(X_1|C)}\,\overline{P(X_2,\cdots,X_n|C)} \\ &= P(X_1|C)\overline{P(X_2|C)\cdots P(X_n|C)} \end{aligned}$$

$$[P(x_1|c^*)\cdots P(x_n|c^*)]P(c^*) > [P(x_1|c)\cdots P(x_n|c)]P(c), \quad c \neq c^*, c = c_1,\cdots,c_L$$

  - MAP classification rule: for

Naïve Bayes

- Algorithm: Discrete-Valued Features
  - Learning Phase: Given a training set S of F features and L classes,

```
For each target value of $c_i$ $(c_i = c_1, \cdots, c_L)$
    $\hat{P}(C = c_i) \leftarrow$ estimate $P(C = c_i)$ with examples in $\mathbf{S}$;
    For every feature value $x_{jk}$ of each feature $X_j$ $(j = 1, \cdots, F; k = 1, \cdots, N_j)$
        $\hat{P}(X_j = x_{jk} | C = c_i) \leftarrow$ estimate $P(X_j = x_{jk} | C = c_i)$ with examples in $\mathbf{S}$;
```

  Output: $F * L$ conditional probabilistic (generative) models

  - Test Phase: Given an unknown instance $\mathbf{X}' = (a'_1, \cdots, a'_n)$

  "Look up tables" to assign the label c^* to $\mathbf{X}'$ if

$$[\hat{P}(a'_1 | c^*) \cdots \hat{P}(a'_n | c^*)]\hat{P}(c^*) > [\hat{P}(a'_1 | c) \cdots \hat{P}(a'_n | c)]\hat{P}(c), \quad c \neq c^*, c = c_1, \cdots, c_L$$

Example

- Example: Play Tennis

PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example

- Learning Phase

Outlook	Play=*Yes*	Play=*No*
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

Temperature	Play=*Yes*	Play=*No*
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

Humidity	Play=*Yes*	Play=*No*
High	3/9	4/5
Normal	6/9	1/5

Wind	Play=*Yes*	Play=*No*
Strong	3/9	3/5
Weak	6/9	2/5

P(Play=*Yes*) = 9/14 P(Play=*No*) = 5/14

Example

- Test Phase
  - Given a new instance, predict its label

    $\mathbf{x}'$=(Outlook=*Sunny*, Temperature=*Cool*, Humidity=*High*, Wind=*Strong*)
  - Look up tables achieved in the learning phrase

P(Outlook=*Sunny*|Play=*Yes*) = 2/9	P(Outlook=*Sunny*|Play=*No*) = 3/5
P(Temperature=*Cool*|Play=*Yes*) = 3/9	P(Temperature=*Cool*|Play==*No*) = 1/5
P(Huminity=*High*|Play=*Yes*) = 3/9	P(Huminity=*High*|Play=*No*) = 4/5
P(Wind=*Strong*|Play=*Yes*) = 3/9	P(Wind=*Strong*|Play=*No*) = 3/5
P(Play=*Yes*) = 9/14	P(Play=*No*) = 5/14

  - Decision making with the MAP rule

$P(Yes|\mathbf{x}') \approx [P(Sunny|Yes)P(Cool|Yes)P(High|Yes)P(Strong|Yes)]P(Play{=}Yes) = 0.0053$

$P(No|\mathbf{x}') \approx [P(Sunny|No)\,P(Cool|No)P(High|No)P(Strong|No)]P(Play{=}No) = 0.0206$

Given the fact $P(Yes|\mathbf{x}') < P(No|\mathbf{x}')$, we label $\mathbf{x}'$ to be "*No*".

Advantages/Disadvantages of Naïve Bayes

- Advantages:
  - Fast to train (single scan). Fast to classify
  - Not sensitive to irrelevant features
  - Handles real and discrete data
  - Handles streaming data well
- Disadvantages:
  - Assumes independence of features

Performance Evaluation

Evaluating classification methods

- **Predictive accuracy**

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}}$$

- Efficiency
  - time to construct the model
  - time to use the model
- Robustness: handling noise and missing values
- Scalability: efficiency in disk-resident databases
- Interpretability:
  - understandable and insight provided by the model
- Compactness of the model: size of the tree, or the number of rules.

Evaluation methods

- **Holdout set**: The available data set *D* is divided into two disjoint subsets,
  - the *training set* D_{train} (for learning a model)
  - the *test set* D_{test} (for testing the model)
- **Important:** training set should not be used in testing and the test set should not be used in learning.
  - Unseen test set provides a unbiased estimate of accuracy.
- The test set is also called the holdout set. (the examples in the original data set *D* are all labeled with classes.)
- This method is mainly used when the data set *D* is large.

Evaluation methods (cont…)

- **n-fold cross-validation**: The available data is partitioned into *n* equal-size disjoint subsets.
- Use each subset as the test set and combine the rest *n*-1 subsets as the training set to learn a classifier.
- The procedure is run *n* times, which give *n* accuracies.
- The final estimated accuracy of learning is the average of the *n* accuracies.
- 10-fold and 5-fold cross-validations are commonly used.
- This method is used when the available data is not large.

Evaluation methods (cont...)

- **Leave-one-out cross-validation**: This method is used when the data set is very small.
- It is a special case of cross-validation
- Each fold of the cross validation has only a single test example and all the rest of the data is used in training.
- If the original data has *m* examples, this is *m*-fold cross-validation

Evaluation methods (cont…)

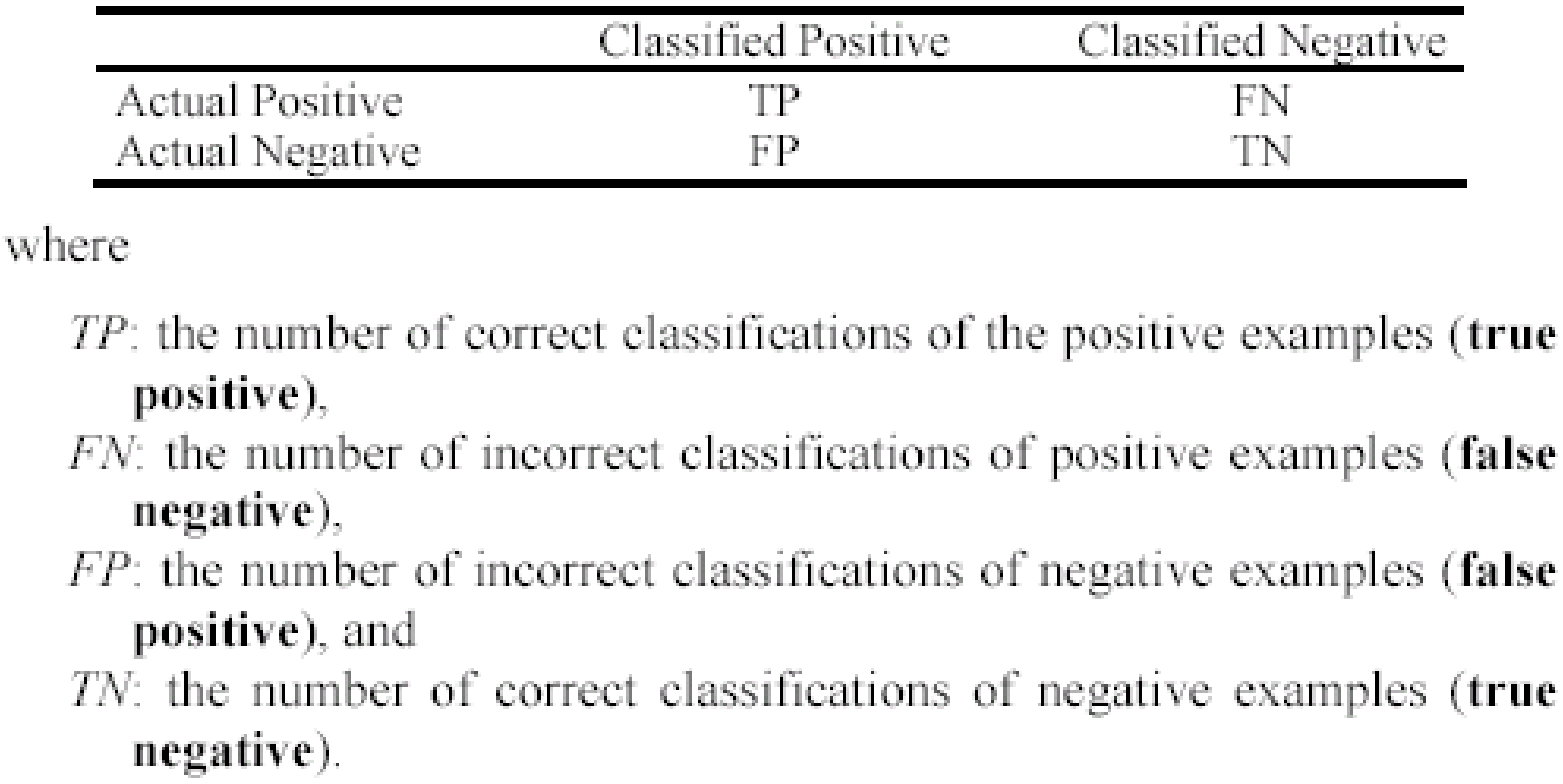
- **Validation set**: the available data is divided into three subsets,
  - a training set,
  - a validation set and
  - a test set.
- A validation set is used frequently for estimating parameters in learning algorithms.
- In such cases, the values that give the best accuracy on the validation set are used as the final parameter values.
- Cross-validation can be used for parameter estimating as well.

Classification measures

- Accuracy is only one measure (error = 1-accuracy).
- **Accuracy is not suitable in some applications**.
- In text mining, we may only be interested in the documents of a particular topic, which are only a small portion of a big document collection.
- In classification involving skewed or highly imbalanced data, e.g., network intrusion and financial fraud detections, we are interested only in the minority class.
  - High accuracy does not mean any intrusion is detected.
  - E.g., 1% intrusion. Achieve 99% accuracy by doing nothing.
- The class of interest is commonly called the **positive class**, and the rest **negative classes**.

Precision and **recall** measures

- Used in information retrieval and text classification.
- We use a confusion matrix to introduce them.



	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

where

TP: the number of correct classifications of the positive examples (**true positive**),

FN: the number of incorrect classifications of positive examples (**false negative**),

FP: the number of incorrect classifications of negative examples (**false positive**), and

TN: the number of correct classifications of negative examples (**true negative**).

Precision and **recall** measures (cont…)

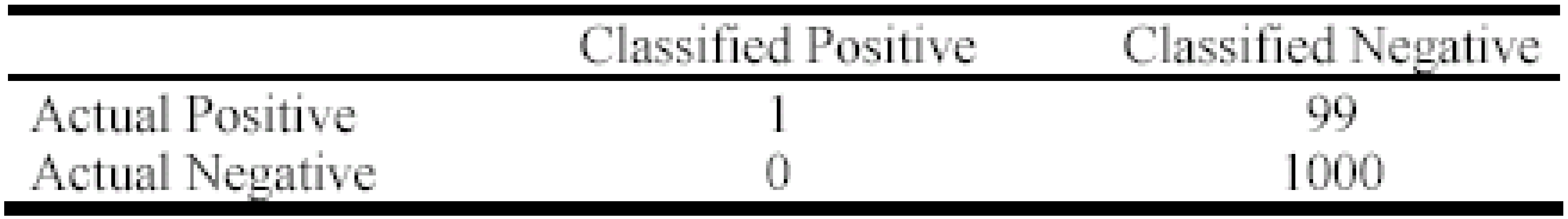
	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

$$p = \frac{TP}{TP + FP}. \qquad r = \frac{TP}{TP + FN}.$$

Precision p is the number of correctly classified positive examples divided by the total number of examples that are classified as positive.

Recall r is the number of correctly classified positive examples divided by the total number of actual positive examples in the test set.

An example



	Classified Positive	Classified Negative
Actual Positive	1	99
Actual Negative	0	1000

- This confusion matrix gives
  - precision $p = 100\%$ and
  - recall $r = 1\%$

  because we only classified one positive example correctly and no negative examples wrongly.
- Note: precision and recall only measure classification on the positive class.

F_1-value (also called F_1-score)

- It is hard to compare two classifiers using two measures. F_1 score combines precision and recall into one measure

$$F_1 = \frac{2pr}{p + r}$$

F_1-score is the harmonic mean of precision and recall.

$$F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

- The harmonic mean of two numbers tends to be closer to the smaller of the two.
- For F_1-value to be large, both *p* and *r* much be large.

Supervised learning vs. unsupervised learning

- Supervised learning: discover patterns in the data that relate data attributes with a target (class) attribute.
  - These patterns are then utilized to predict the values of the target attribute in future data instances.
- Unsupervised learning: The data have no target attribute.
  - We want to explore the data to find some intrinsic structures in them.

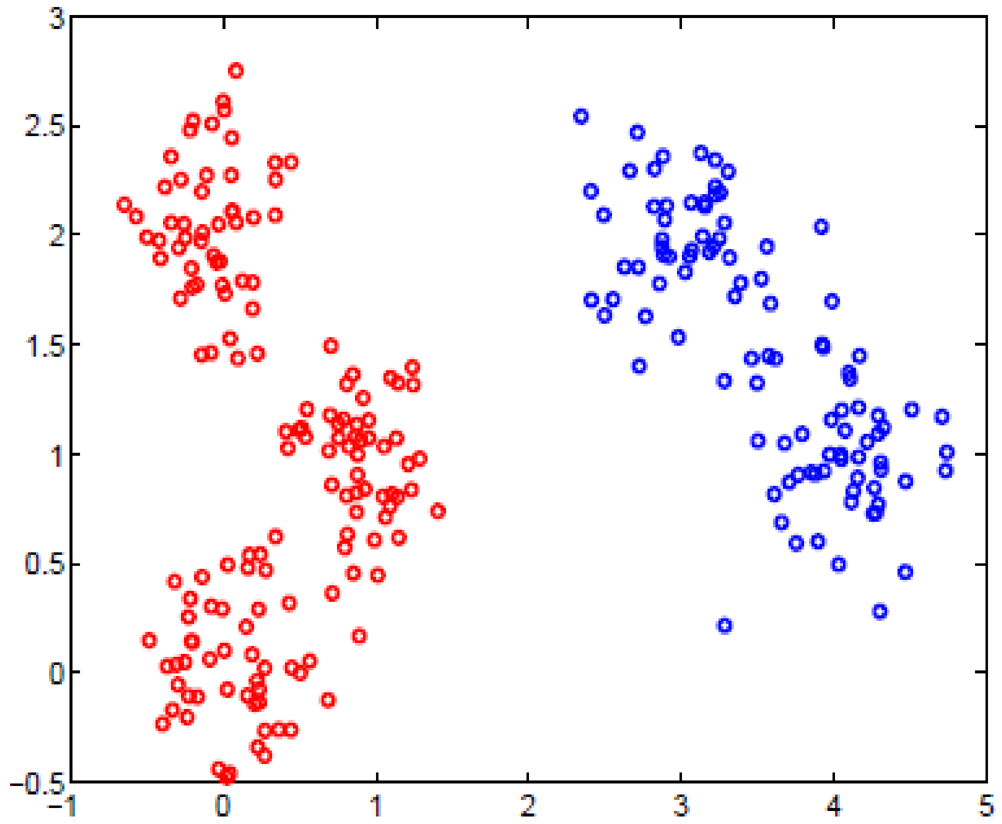
Clustering

Introduction

- Cluster: A collection/group of data objects/points
  - similar (or related) to one another within the same group
  - dissimilar (or unrelated) to the objects in other groups
- Cluster analysis
  - find *similarities* between data according to characteristics underlying the data and grouping similar data objects into clusters
- Clustering Analysis: Unsupervised learning
  - no predefined classes for a training data set
  - Two general tasks: identify the "natural" clustering number and properly grouping objects into "sensible" clusters
- Typical applications
  - as a stand-alone tool to gain an insight into data distribution
  - as a preprocessing step of other algorithms in intelligent systems

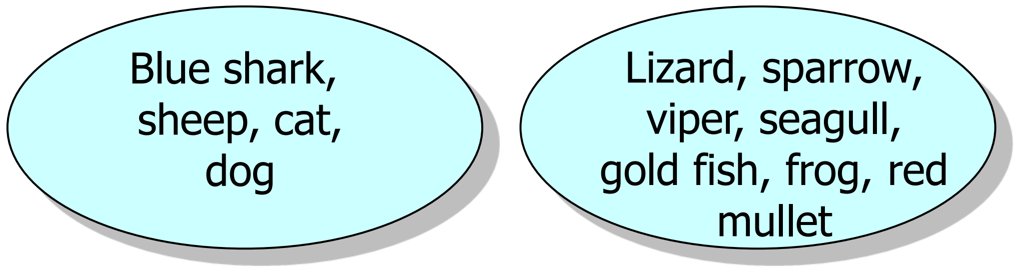
Introduction

- Illustrative Example 1: how many clusters?

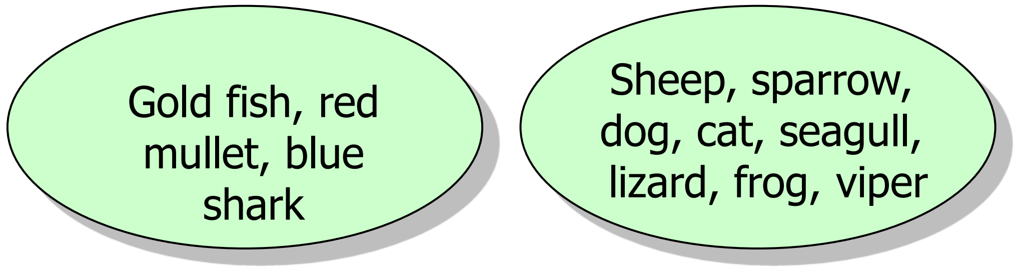


Introduction

- Illustrative Example 2: are they in the same cluster?


Blue shark, sheep, cat, dog
Lizard, sparrow, viper, seagull, gold fish, frog, red mullet

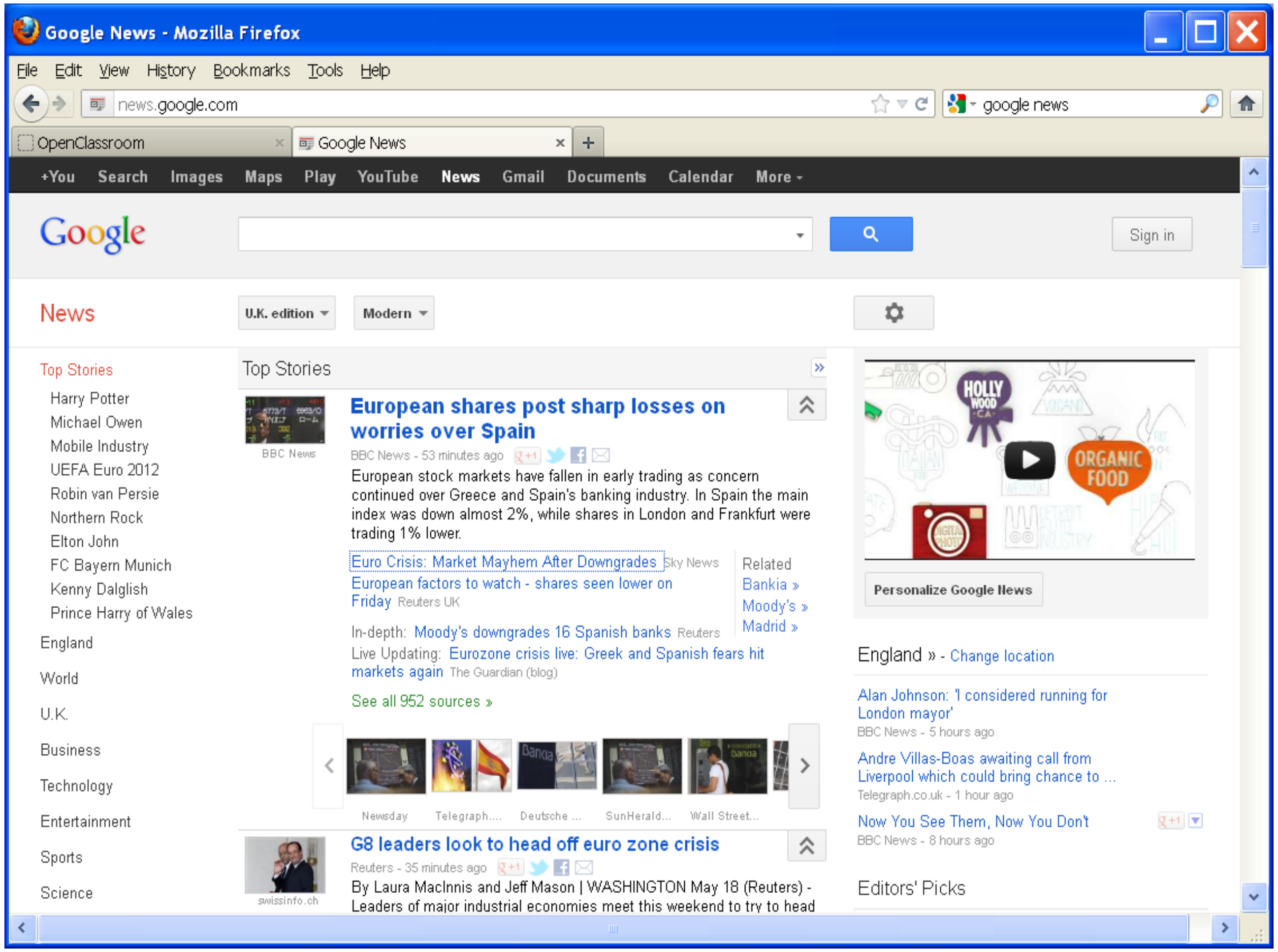
1. Two clusters
2. Clustering criterion: How animals bear their progeny


Gold fish, red mullet, blue shark
Sheep, sparrow, dog, cat, seagull, lizard, frog, viper

1. Two clusters
2. Clustering criterion: Existence of lungs

Introduction

- Real Applications: Google News



Google News - Mozilla Firefox

File Edit View History Bookmarks Tools Help

news.google.com

google news

OpenClassroom | Google News

+You Search Images Maps Play YouTube News Gmail Documents Calendar More

Google

Sign in

News U.K. edition Modern

Top Stories
Harry Potter
Michael Owen
Mobile Industry
UEFA Euro 2012
Robin van Persie
Northern Rock
Elton John
FC Bayern Munich
Kenny Dalglish
Prince Harry of Wales
England
World
U.K.
Business
Technology
Entertainment
Sports
Science

Top Stories

European shares post sharp losses on worries over Spain
BBC News - 53 minutes ago
European stock markets have fallen in early trading as concern continued over Greece and Spain's banking industry. In Spain the main index was down almost 2%, while shares in London and Frankfurt were trading 1% lower.

Euro Crisis: Market Mayhem After Downgrades Sky News
European factors to watch - shares seen lower on Friday Reuters UK
In-depth: Moody's downgrades 16 Spanish banks Reuters
Live Updating: Eurozone crisis live: Greek and Spanish fears hit markets again The Guardian (blog)
See all 952 sources »

Related
Bankia »
Moody's »
Madrid »

Newsday Telegraph.... Deutsche ... SunHerald... Wall Street...

G8 leaders look to head off euro zone crisis
Reuters - 35 minutes ago
By Laura MacInnis and Jeff Mason | WASHINGTON May 18 (Reuters) - Leaders of major industrial economies meet this weekend to try to head

Personalize Google News

England » - Change location

Alan Johnson: 'I considered running for London mayor'
BBC News - 5 hours ago

Andre Villas-Boas awaiting call from Liverpool which could bring chance to ...
Telegraph.co.uk - 1 hour ago

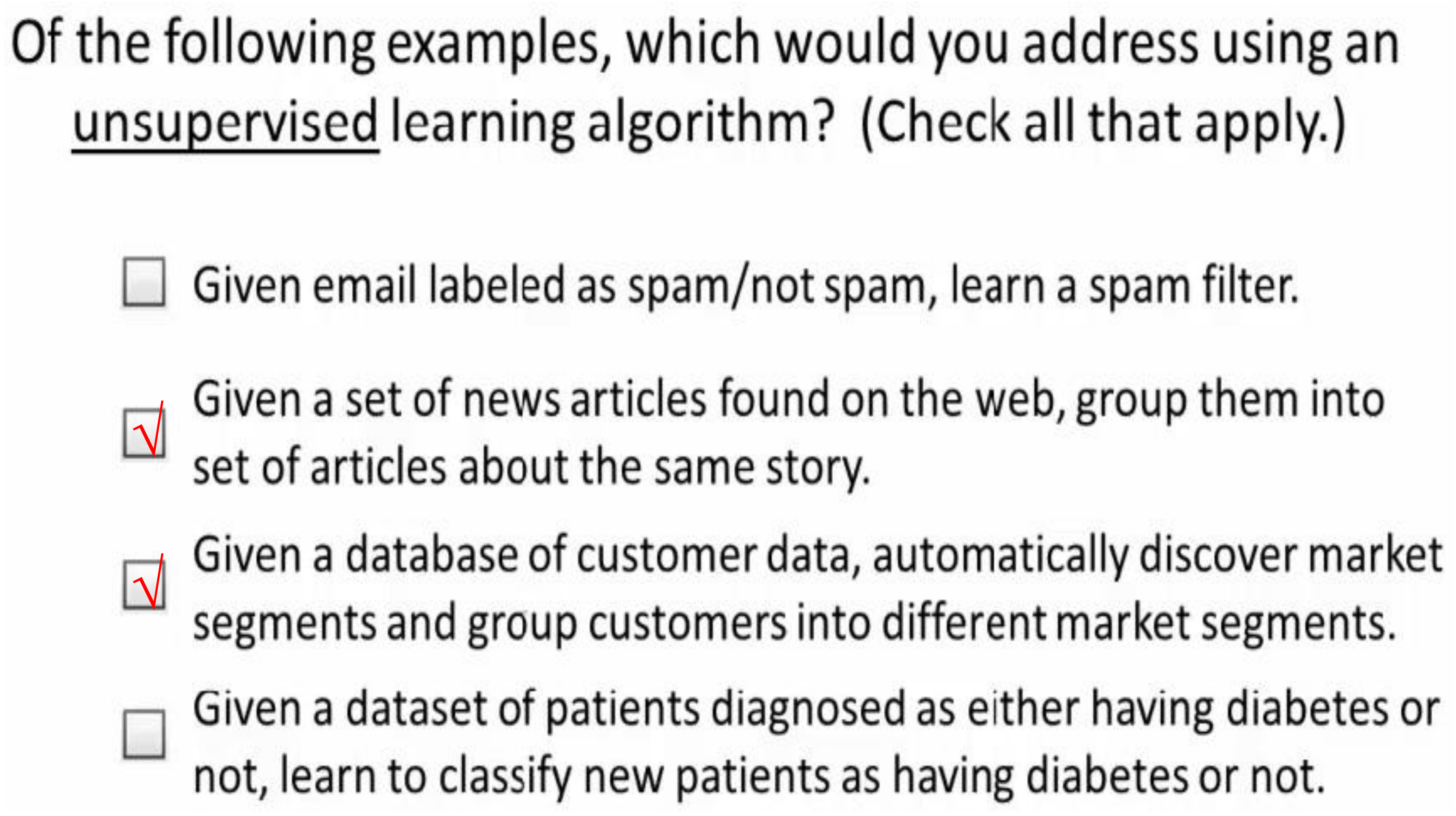
Now You See Them, Now You Don't
BBC News - 8 hours ago

Editors' Picks

Introduction

- A technique demanded by many real world tasks
  - Bank/Internet Security: fraud/spam pattern discovery
  - Biology: taxonomy of living things such as kingdom, phylum, class, order, family, genus and species
  - City-planning: Identifying groups of houses according to their house type, value, and geographical location
  - Climate change: understanding earth climate, find patterns of atmospheric and ocean
  - Finance: stock clustering analysis to uncover correlation underlying shares
  - Image Compression/segmentation: coherent pixels grouped
  - Information retrieval/organisation: Google search, topic-based news
  - Land use: Identification of areas of similar land use in an earth observation database
  - Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
  - Social network mining: special interest group automatic discovery

Quiz



Of the following examples, which would you address using an <u>unsupervised</u> learning algorithm? (Check all that apply.)

- ☐ Given email labeled as spam/not spam, learn a spam filter.
- [✓] Given a set of news articles found on the web, group them into set of articles about the same story.
- [✓] Given a database of customer data, automatically discover market segments and group customers into different market segments.
- ☐ Given a dataset of patients diagnosed as either having diabetes or not, learn to classify new patients as having diabetes or not.

Aspects of clustering

- A clustering algorithm
  - Partitional clustering
  - Hierarchical clustering
  - …
- A distance (similarity, or dissimilarity) function
- Clustering quality
  - Inter-clusters distance $\Rightarrow$ maximized
  - Intra-clusters distance $\Rightarrow$ minimized
- The quality of a clustering result depends on the algorithm, the distance function, and the application.

Data Types and Representations

- Discrete vs. Continuous
  - Discrete Feature
    - Has only a finite set of values

      e.g., zip codes, rank, or the set of words in a collection of documents
    - Sometimes, represented as integer variable
  - Continuous Feature
    - Has real numbers as feature values

      e.g, temperature, height, or weight
    - Practically, real values can only be measured and represented using a finite number of digits
    - Continuous features are typically represented as floating-point variables

Data Types and Representations

- Data representations
  - Data matrix (object-by-feature structure)

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}$$

- **n data points (objects) with p dimensions (features)**
- **Two modes: row and column represent different entities**

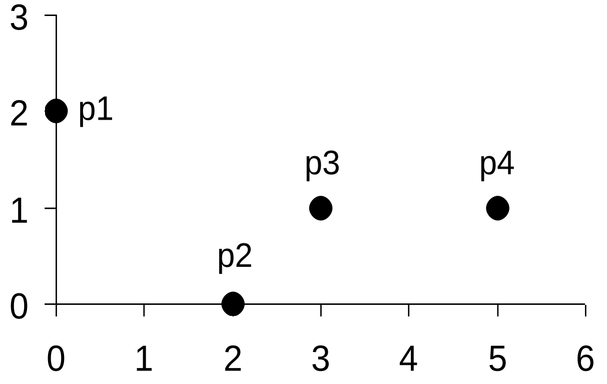
  - Distance/dissimilarity matrix (object-by-object structure)

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix}$$

- **n data points, but registers only the distance**
- **A symmetric/triangular matrix**
- **Single mode: row and column for the same entity (distance)**

Data Types and Representations

- Examples



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

Data Matrix

	p1	**p2**	**p3**	**p4**
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

Distance Matrix (i.e., Dissimilarity Matrix) for Euclidean Distance

Distance Measures

- Minkowski Distance (http://en.wikipedia.org/wiki/Minkowski_distance)

  For $\mathbf{x} = (x_1 \, x_2 \cdots x_n)$ and $\mathbf{y} = (y_1 \, y_2 \cdots y_n)$

$$d(\mathbf{x}, \mathbf{y}) = \left(| x_1 - y_1 |^p + | x_2 - y_2 |^p \cdots + | x_n - y_n |^p \right)^{\frac{1}{p}}, \quad p > 0$$

  – $p = 1$: Manhattan (city block) distance

$$d(\mathbf{x}, \mathbf{y}) = | x_1 - y_1 | + | x_2 - y_2 | \cdots + | x_n - y_n |$$

  – $p = 2$: Euclidean distance

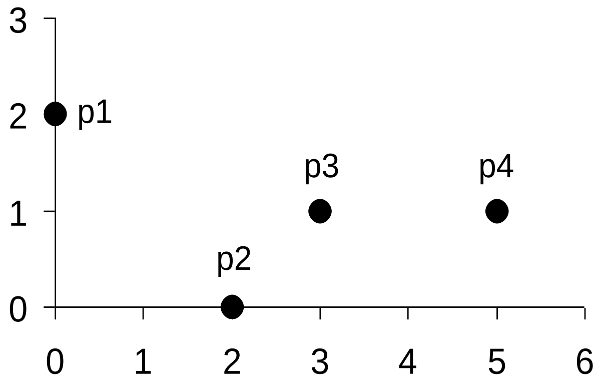
$$d(\mathbf{x}, \mathbf{y}) = \sqrt{| x_1 - y_1 |^2 + | x_2 - y_2 |^2 \cdots + | x_n - y_n |^2}$$

  – Do not confuse p with n, i.e., all these distances are defined based on all numbers of features (dimensions).

  – A generic measure: use appropriate p in different applications

Distance Measures

- Example: Manhatten and Euclidean distances



L1	p1	p2	p3	p4
p1	0	4	4	6
p2	4	0	2	4
p3	4	2	0	2
p4	6	4	2	0

Distance Matrix for Manhattan Distance

point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

Data Matrix

L2	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

Distance Matrix for Euclidean Distance

Distance Measures

- Cosine Measure (Similarity vs. Distance)

  For $\mathbf{x} = (x_1\, x_2 \cdots x_n)$ and $\mathbf{y} = (y_1\, y_2 \cdots y_n)$

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{x_1 y_1 + \cdots + x_n y_n}{\sqrt{x_1^2 + \cdots + x_n^2}\sqrt{y_1^2 + \cdots + y_n^2}}$$

$$d(\mathbf{x}, \mathbf{y}) = 1 - \cos(\mathbf{x}, \mathbf{y})$$

  – Property: $0 \le d(\mathbf{x}, \mathbf{y}) \le 2$
  – Nonmetric vector objects: keywords in documents, gene features in micro-arrays, …
  – Applications: information retrieval, biologic taxonomy, ...

Distance Measures

- Example: Cosine measure

$$\mathbf{x}_1 = (3, 2, 0, 5, 2, 0, 0), \mathbf{x}_2 = (1,0, 0, 0, 1, 0, 2)$$

$$3\times1 + 2\times0 + 0\times0 + 5\times0 + 2\times1 + 0\times0 + 0\times2 = 5$$

$$\sqrt{3^2 + 2^2 + 0^2 + 5^2 + 2^2 + 0^2 + 0^2} = \sqrt{42} \approx 6.48$$

$$\sqrt{1^2 + 0^2 + 0^2 + 0^2 + 1^2 + 0^2 + 2^2} = \sqrt{6} \approx 2.45$$

$$\cos(\mathbf{x}_1, \mathbf{x}_2) = \frac{5}{6.48\times2.45} \approx 0.32$$

$$d(\mathbf{x}_1, \mathbf{x}_2) = 1 - \cos(\mathbf{x}_1, \mathbf{x}_2) = 1 - 0.32 = 0.68$$

K-means Clustering

Introduction

- Partitioning Clustering Approach
  - a typical clustering analysis approach via iteratively partitioning training data set to learn a partition of the given data space
  - learning a partition on a data set to produce several non-empty clusters (usually, the number of clusters given in advance)
  - in principle, optimal partition achieved via minimising the sum of squared distance to its "representative object" in each cluster

$$E = \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} d^2(\mathbf{x}, \mathbf{m}_k)$$

e.g., Euclidean distance $d^2(\mathbf{x}, \mathbf{m}_k) = \sum_{n=1}^{N} (x_n - m_{kn})^2$

Introduction

- Given a *K*, find a partition of *K clusters* to optimise the chosen partitioning criterion (cost function)
  - global optimum: exhaustively search all partitions
- The *K-means* algorithm: a heuristic method
  - K-means algorithm (MacQueen'67): each cluster is represented by the centre of the cluster and the algorithm converges to stable centriods of clusters.
  - K-means algorithm is the simplest partitioning method for clustering analysis and widely used in data mining applications.

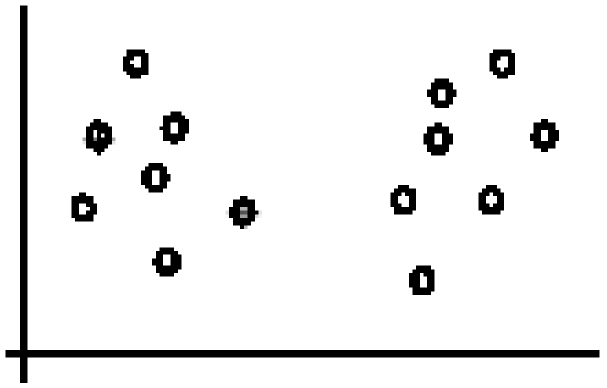
K-means Algorithm

- Given the cluster number *K*, the *K-means* algorithm is carried out in three steps after initialization:

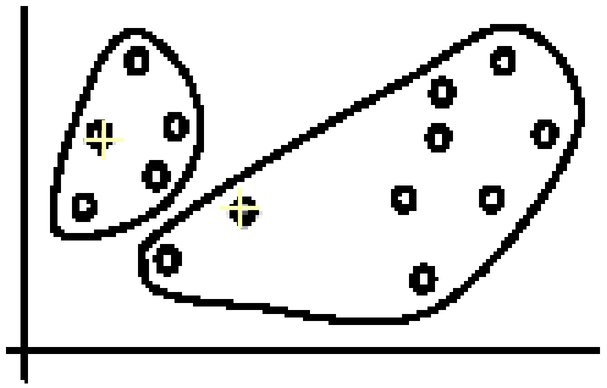
Initialisation: set seed points (randomly)

1) Assign each object to the cluster of the nearest seed point measured with a specific distance metric
2) Compute new seed points as the centroids of the clusters of the current partition (the centroid is the centre, i.e., *mean point*, of the cluster)
3) Go back to Step 1), stop when no more new assignment (i.e., membership in each cluster no longer changes)

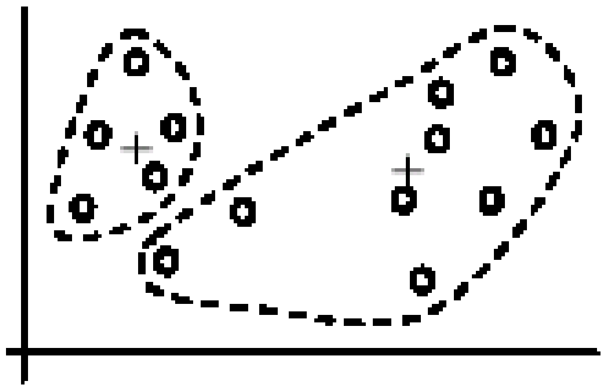
An example



(A). Random selection of *k* centers

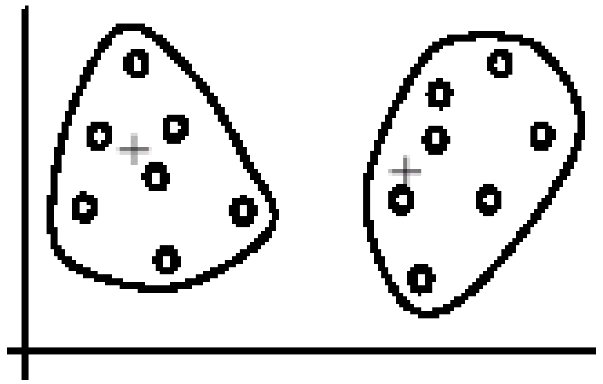


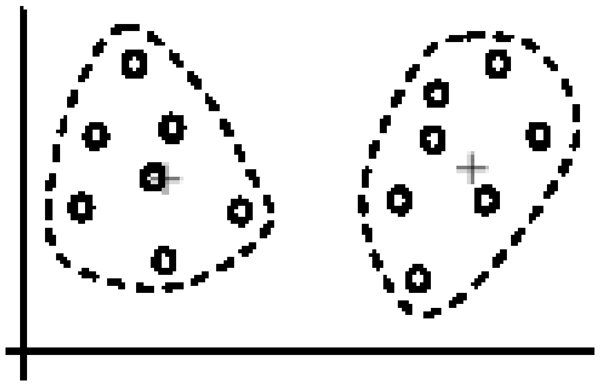
Iteration 1: (B). Cluster assignment

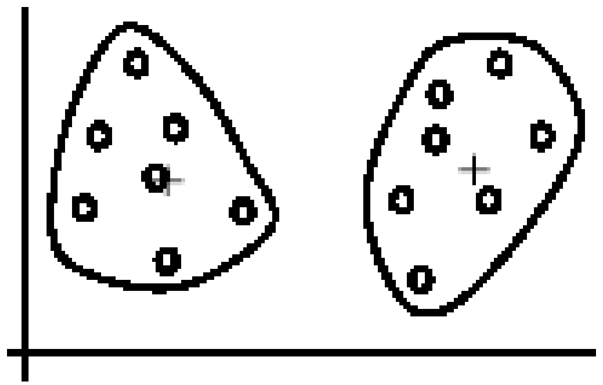


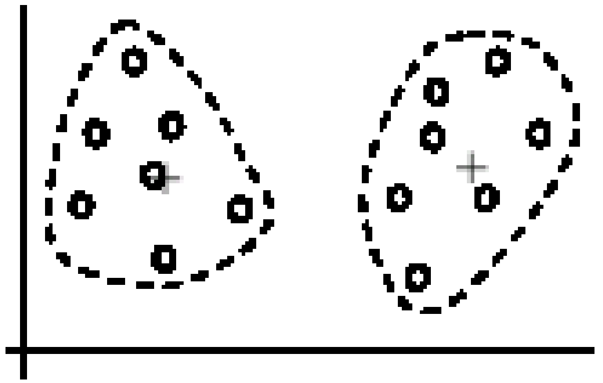
(C). Re-compute centroids

An example (cont …)


Iteration 2: (D). Cluster assignment


(E). Re-compute centroids


Iteration 3: (F). Cluster assignment


(G). Re-compute centroids

Stopping/convergence criterion

1. no (or minimum) re-assignments of data points to different clusters,
2. no (or minimum) change of centroids, or
3. minimum decrease in the **sum of squared error** (SSE),

$$SSE = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2 \qquad (1)$$

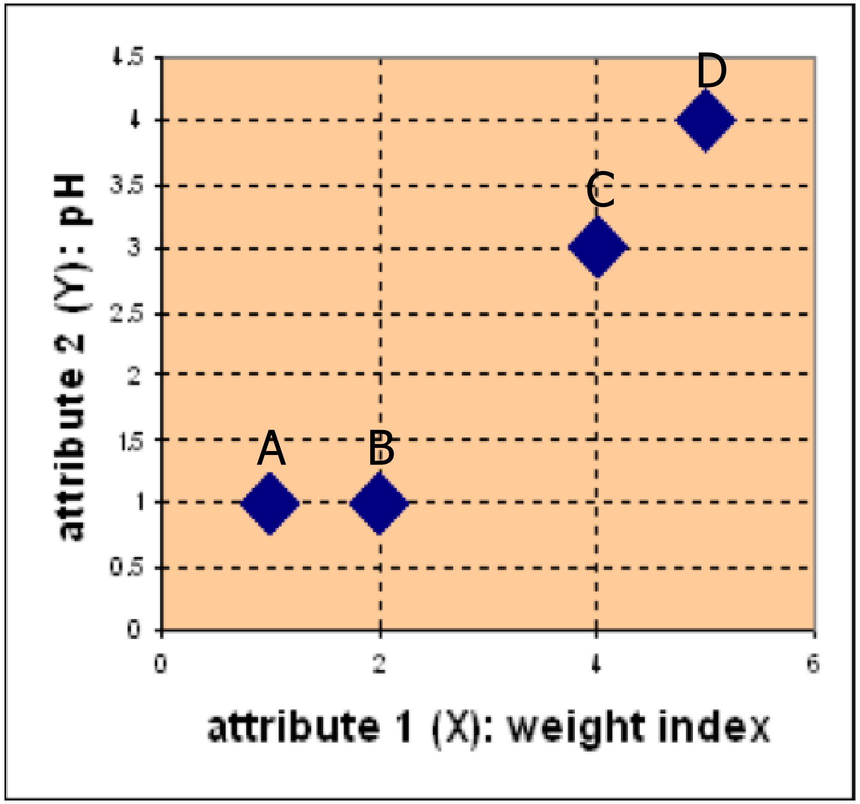
– C_i is the jth cluster, $\mathbf{m}_j$ is the centroid of cluster C_j (the mean vector of all the data points in C_j), and $dist(\mathbf{x}, \mathbf{m}_j)$ is the distance between data point $\mathbf{x}$ and centroid $\mathbf{m}_j$.

Example

- Problem

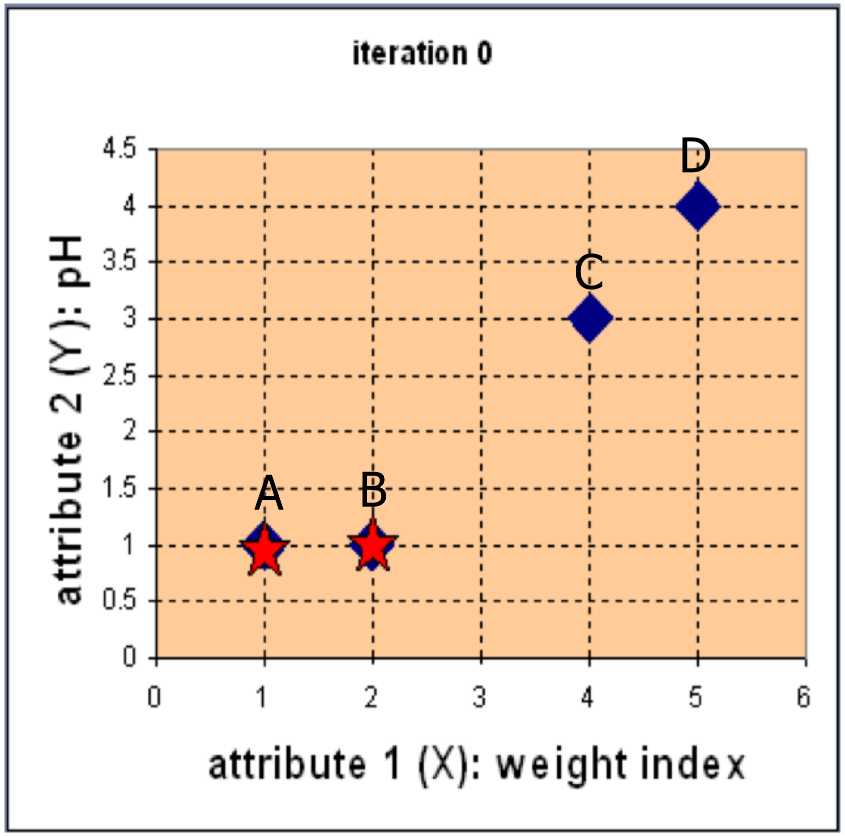
  Suppose we have 4 types of medicines and each has two attributes (pH and weight index). Our goal is to group these objects into *K=2* group of medicine.

Medicine	Weight	pH-Index
A	1	1
B	2	1
C	4	3
D	5	4



Example

- Step 1: Use initial seed points for partitioning



$$c_1 = A, c_2 = B$$

$$\mathbf{D}^0 = \begin{array}{c} \begin{bmatrix} 0 & 1 & 3.61 & \boxed{5} \\ 1 & 0 & 2.83 & \boxed{4.24} \end{bmatrix} \\ \begin{array}{cccc} A & B & C & D \end{array} \end{array} \quad \begin{array}{ll} \mathbf{c}_1 = (1,1) & group-1 \\ \mathbf{c}_2 = (2,1) & group-2 \end{array}$$

Euclidean distance

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \begin{array}{c} X \\ Y \end{array}$$

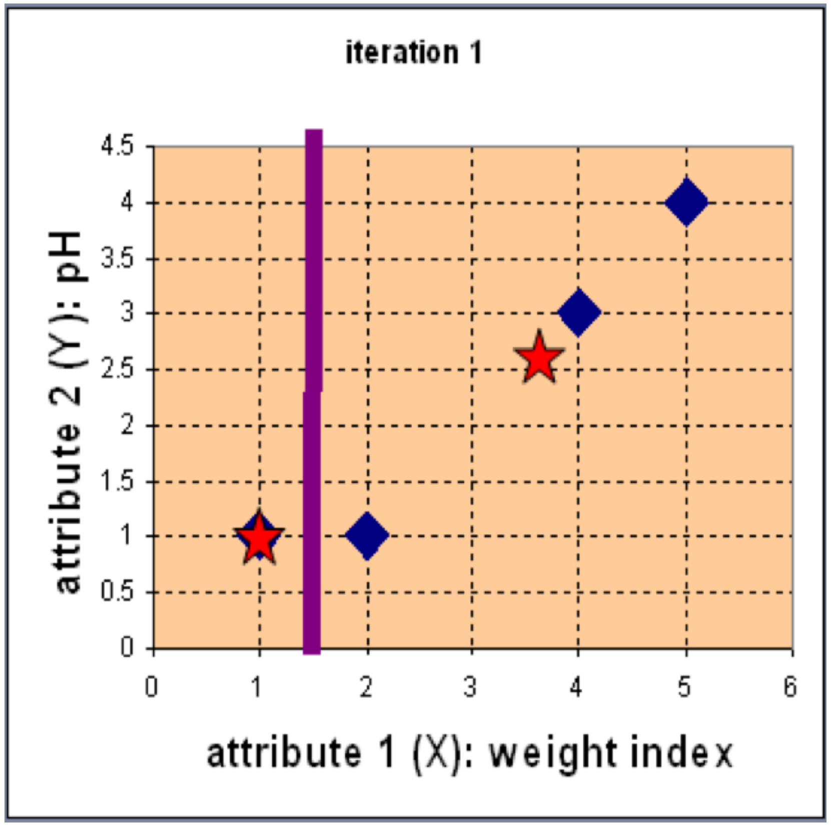
$$d(D,c_1) = \sqrt{(5-1)^2 + (4-1)^2} = 5$$

$$d(D,c_2) = \sqrt{(5-2)^2 + (4-1)^2} = 4.24$$

Assign each object to the cluste
with the nearest seed point

Example

- Step 2: Compute new centroids of the current partition



Knowing the members of each
cluster, now we compute the new
centroid of each group based on
these new memberships.

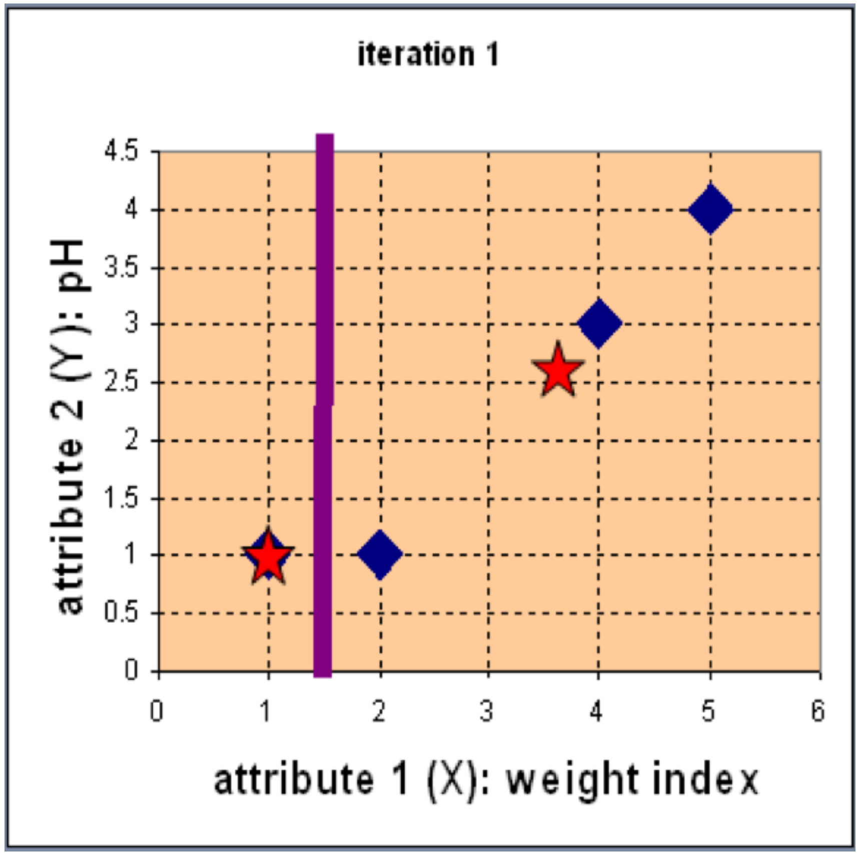
$$c_1 = (1,\ 1)$$

$$c_2 = \left(\frac{2+4+5}{3},\ \frac{1+3+4}{3} \right)$$

$$= (\frac{11}{3},\ \frac{8}{3})$$

Example

- Step 2: Renew membership based on new centroids



Compute the distance of all objects to the new

$$\mathbf{D}^1 = \begin{bmatrix} 0 & 1 & 3.61 & 5 \\ 3.14 & 2.36 & 0.47 & 1.89 \end{bmatrix} \quad \begin{array}{ll} \mathbf{c}_1 = (1,1) & group-1 \\ \mathbf{c}_2 = (\frac{11}{3}, \frac{8}{3}) & group-2 \end{array}$$

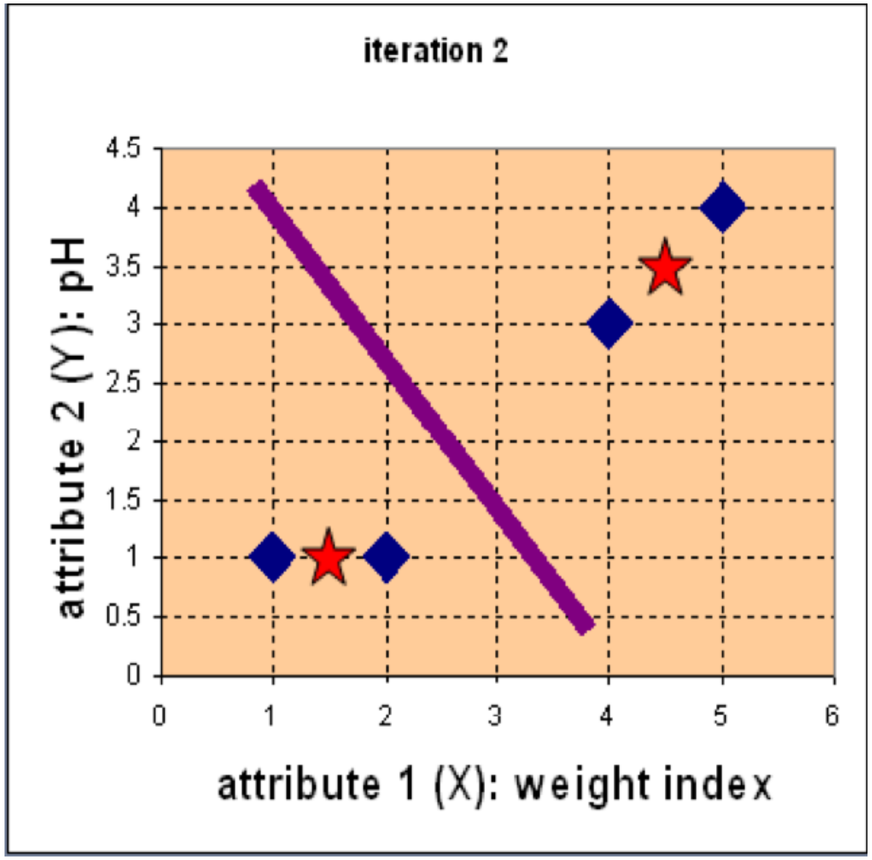
$$\begin{array}{cccc} A & B & C & D \end{array}$$

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \begin{array}{l} X \\ Y \end{array}$$

Assign the membership to objects

Example

- Step 3: Repeat the first two steps until its convergence



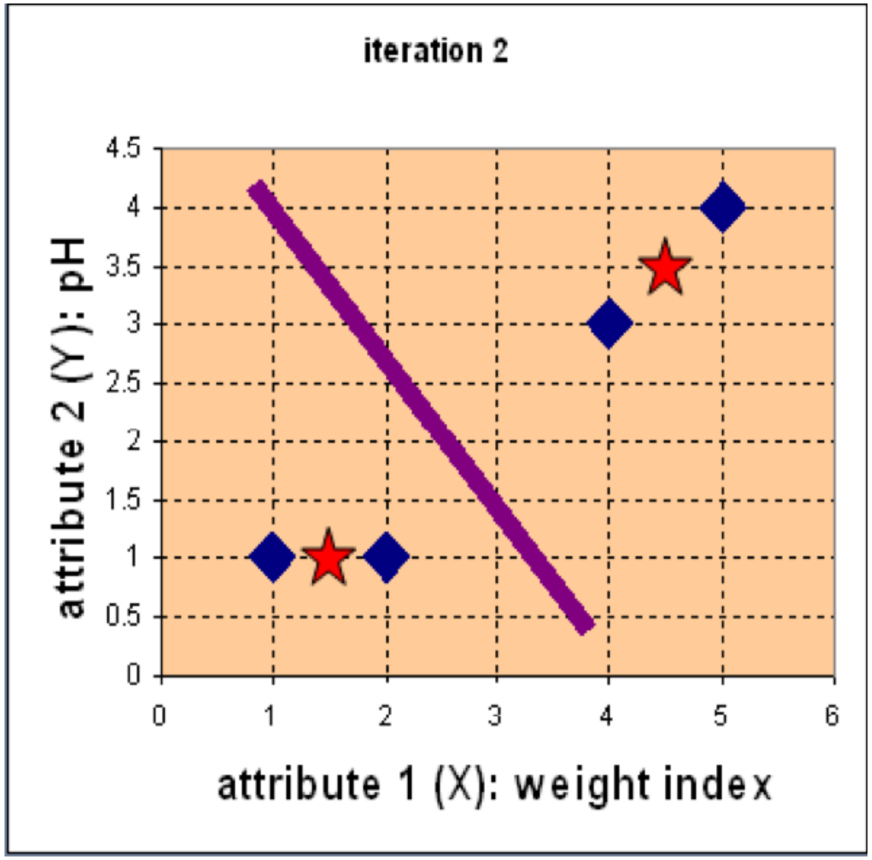
Knowing the members of each cluster, now we compute the new centroid of each group based on these new memberships.

$$c_1 = \left(\frac{1+2}{2}, \ \frac{1+1}{2} \right) = (1\frac{1}{2}, \ 1)$$

$$c_2 = \left(\frac{4+5}{2}, \ \frac{3+4}{2} \right) = (4\frac{1}{2}, \ 3\frac{1}{2})$$

Example

- Step 3: Repeat the first two steps until its convergence



Compute the distance of all objects to the new centroids

$$\mathbf{D}^2 = \begin{bmatrix} 0.5 & 0.5 & 3.20 & 4.61 \\ 4.30 & 3.54 & 0.71 & 0.71 \end{bmatrix} \quad \begin{array}{ll} \mathbf{c}_1 = (1\frac{1}{2}, 1) & group-1 \\ \mathbf{c}_2 = (4\frac{1}{2}, 3\frac{1}{2}) & group-2 \end{array}$$

$$\begin{array}{ccccc} A & B & C & D & \\ \left[\begin{matrix} 1 \\ 1 \end{matrix}\right. & \begin{matrix} 2 \\ 1 \end{matrix} & \begin{matrix} 4 \\ 3 \end{matrix} & \left.\begin{matrix} 5 \\ 4 \end{matrix}\right] & \begin{matrix} X \\ Y \end{matrix} \end{array}$$

Stop due to no new assignment
Membership in each cluster no longer change

Strengths of k-means

- Strengths:
  - Simple: easy to understand and to implement
  - Efficient: Time complexity: $O(tkn)$,

    where n is the number of data points,

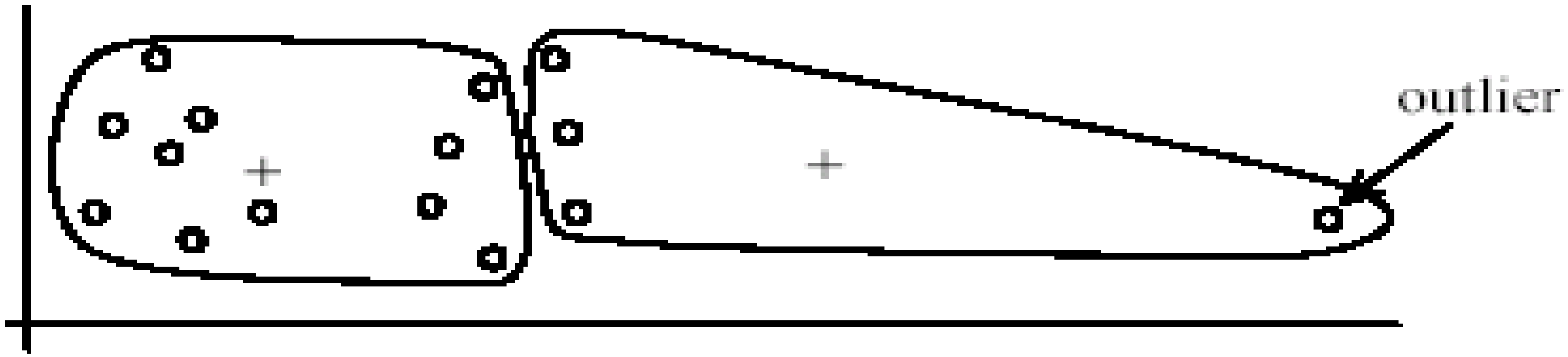
    k is the number of clusters, and

    t is the number of iterations.
  - Since both k and t are small. k-means is considered a linear algorithm.
- K-means is the most popular clustering algorithm.
- Note that: it terminates at a local optimum if SSE is used. The global optimum is hard to find due to complexity.

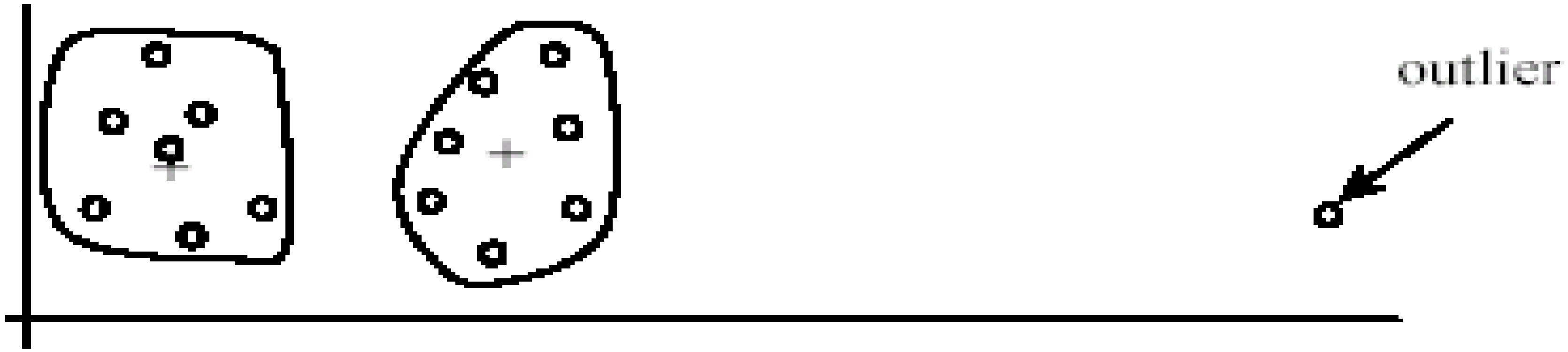
Weaknesses of k-means

- The algorithm is only applicable if the mean is defined.
  - For categorical data, *k*-mode - the centroid is represented by most frequent values.
- The user needs to specify *k*.
- The algorithm is sensitive to **outliers**
  - Outliers are data points that are very far away from other data points.
  - Outliers could be errors in the data recording or some special data points with very different values.

Weaknesses of k-means: Problems with outliers



(A): Undesirable clusters



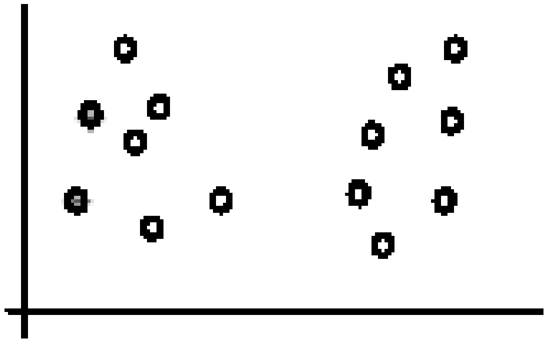
(B): Ideal clusters

Weaknesses of k-means: To deal with outliers

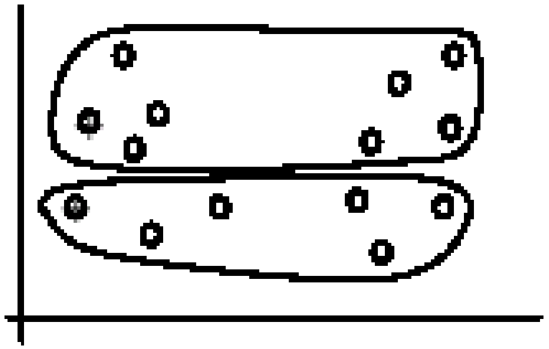
- One method is to remove some data points in the clustering process that are much further away from the centroids than other data points.
  - To be safe, we may want to monitor these possible outliers over a few iterations and then decide to remove them.
- Another method is to perform random sampling. Since in sampling we only choose a small subset of the data points, the chance of selecting an outlier is very small.
  - Assign the rest of the data points to the clusters by distance or similarity comparison, or classification

Weaknesses of k-means (cont …)

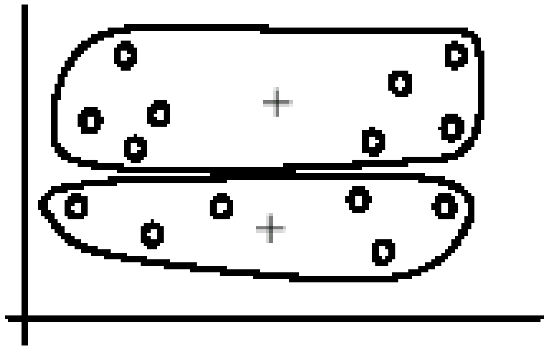
- The algorithm is sensitive to initial seeds.



(A). Random selection of seeds (centroids)



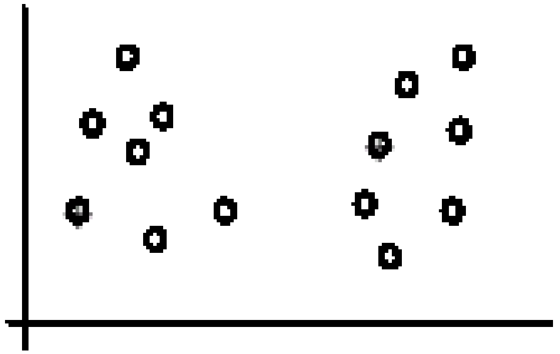
(B). Iteration 1



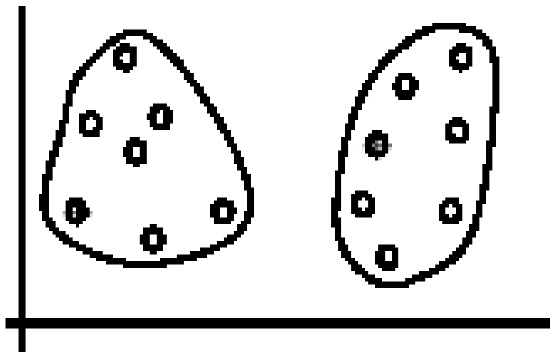
(C). Iteration 2

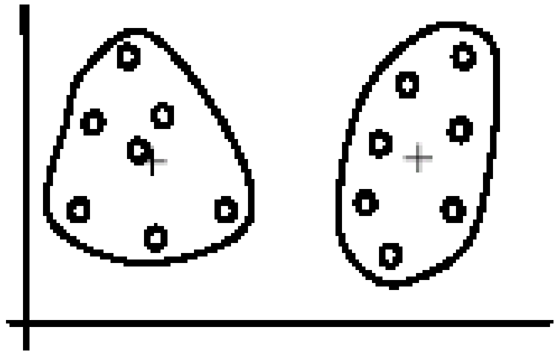
Weaknesses of k-means (cont ...)

- If we use different seeds: good results


(A). Random selection of *k* seeds (centroids)

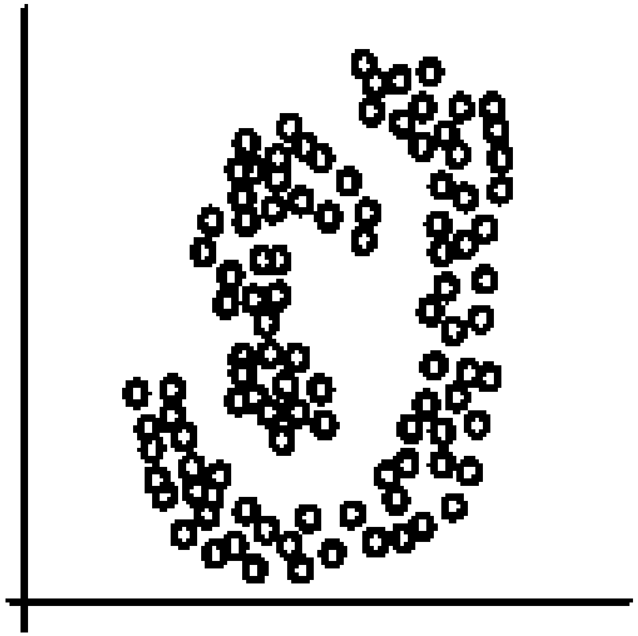
There are some methods to help choose good seeds

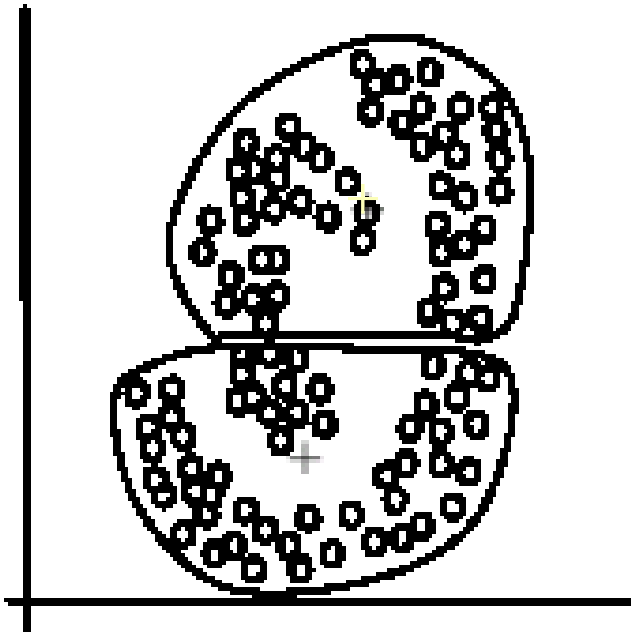

(B). Iteration 1


(C). Iteration 2

Weaknesses of k-means (cont ...)

- The *k*-means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).


(A): Two natural clusters


(B): *k*-means clusters

K-means summary

- Despite weaknesses, *k*-means is still the most popular algorithm due to its simplicity, efficiency and
  - other clustering algorithms have their own lists of weaknesses.
- No clear evidence that any other clustering algorithm performs better in general
  - although they may be more suitable for some specific types of data or applications.
- Comparing different clustering algorithms is a difficult task. No one knows the correct clusters!

Cluster Evaluation: hard problem

- The quality of a clustering is very hard to evaluate because
  - We do not know the correct clusters
- Some methods are used:
  - User inspection
    - Study centroids, and spreads
    - Rules from a decision tree.
    - For text documents, one can read some documents in clusters.

Cluster evaluation: ground truth

- We use some labeled data (for classification)
- Assumption: Each class is a cluster.
- After clustering, a confusion matrix is constructed. From the matrix, we compute various measurements, entropy, purity, precision, recall and F-score.
  - Let the classes in the data D be $C = (c_1, c_2, \ldots, c_k)$. The clustering method produces k clusters, which divides D into k disjoint subsets, $D_1, D_2, \ldots, D_k$.