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.

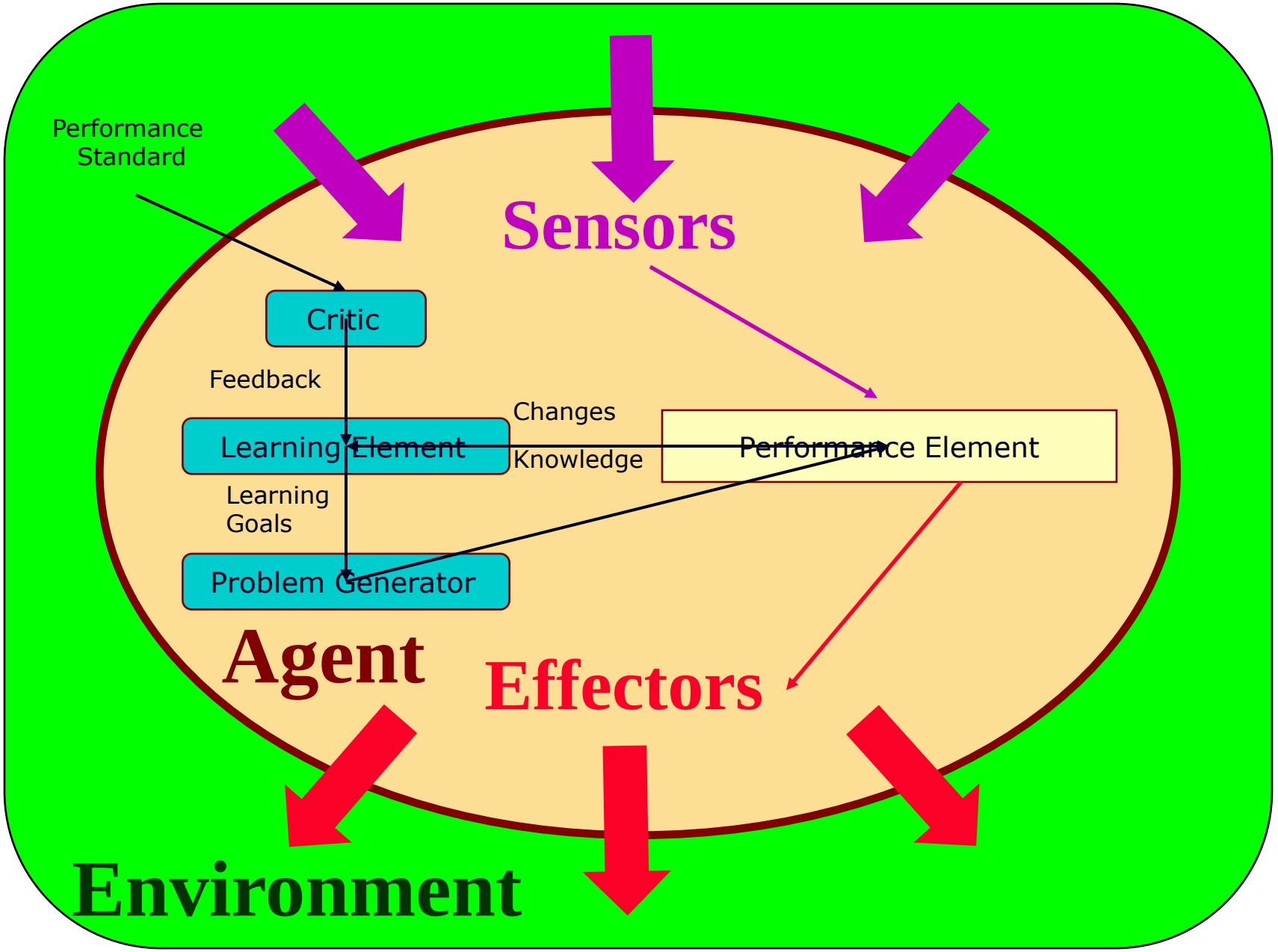
Another application

- 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

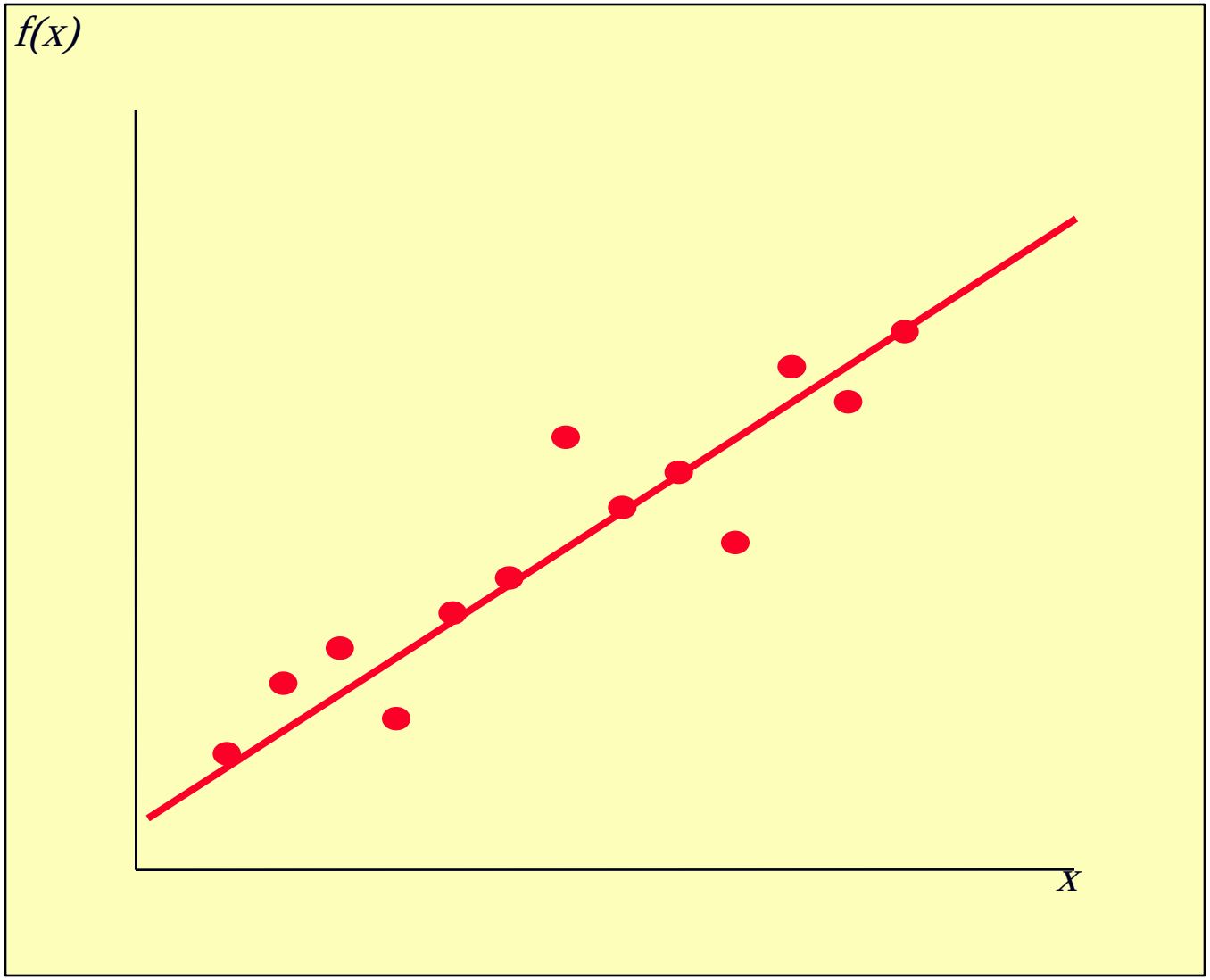
Machine learning

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

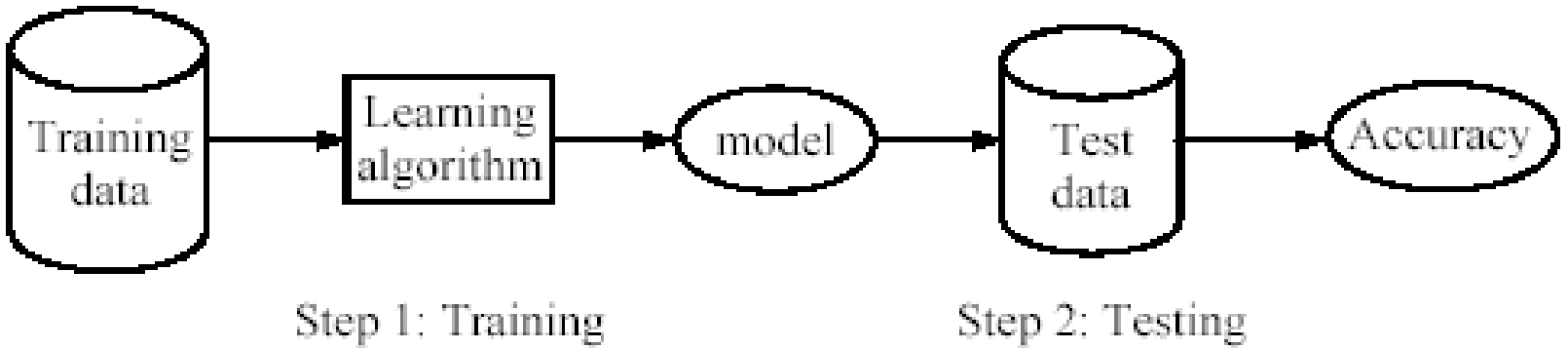
- Supervised learning: classification is seen as supervised learning from examples.
  - Supervision: The data (observations, measurements, etc.) are labeled with pre-defined classes. It is like that a "teacher" gives the classes (supervision).
  - Test data are classified into these classes too.
- Unsupervised 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

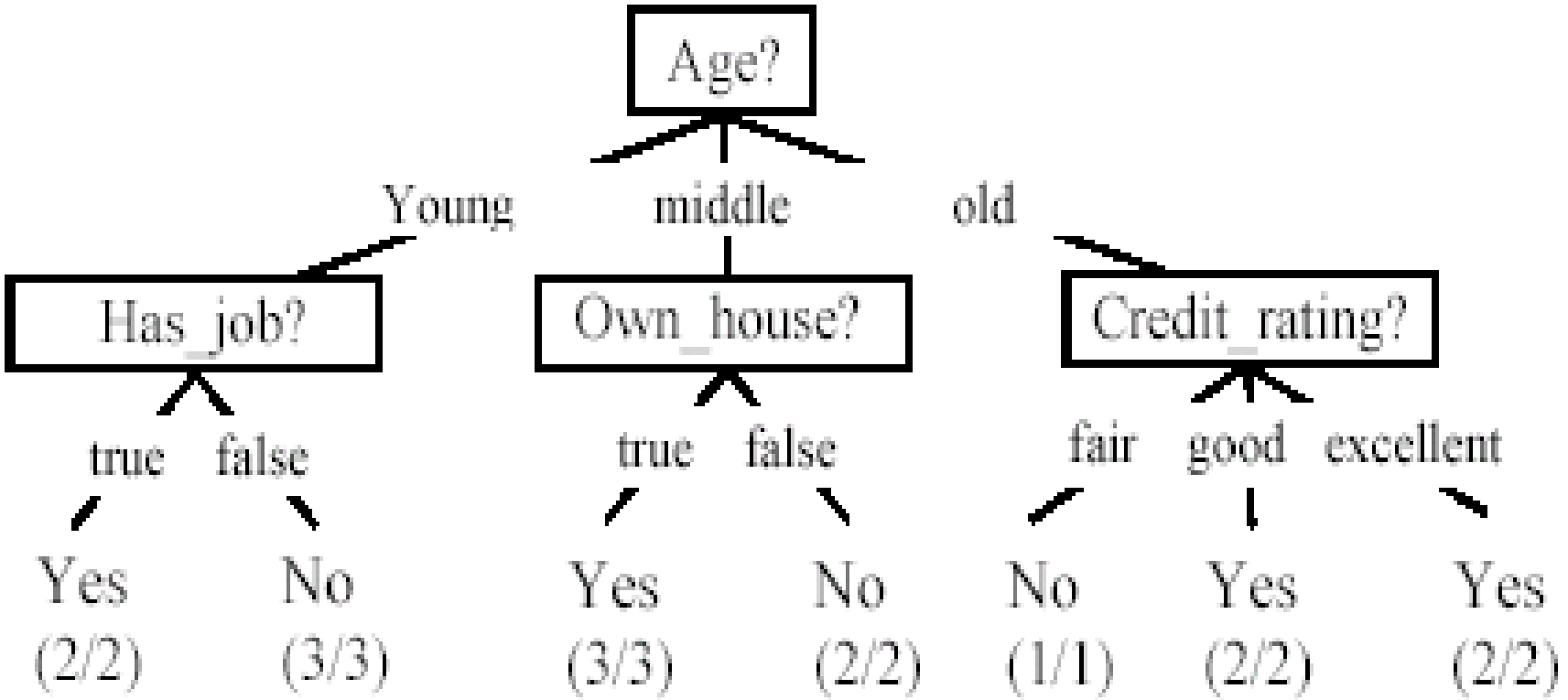
- in general, constructing the smallest possible decision tree is an intractable problem
- algorithms exist for constructing reasonably small trees
- basic idea: test the most important attribute first
  - attribute that makes the most difference for the classification of an example
    - can be determined 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

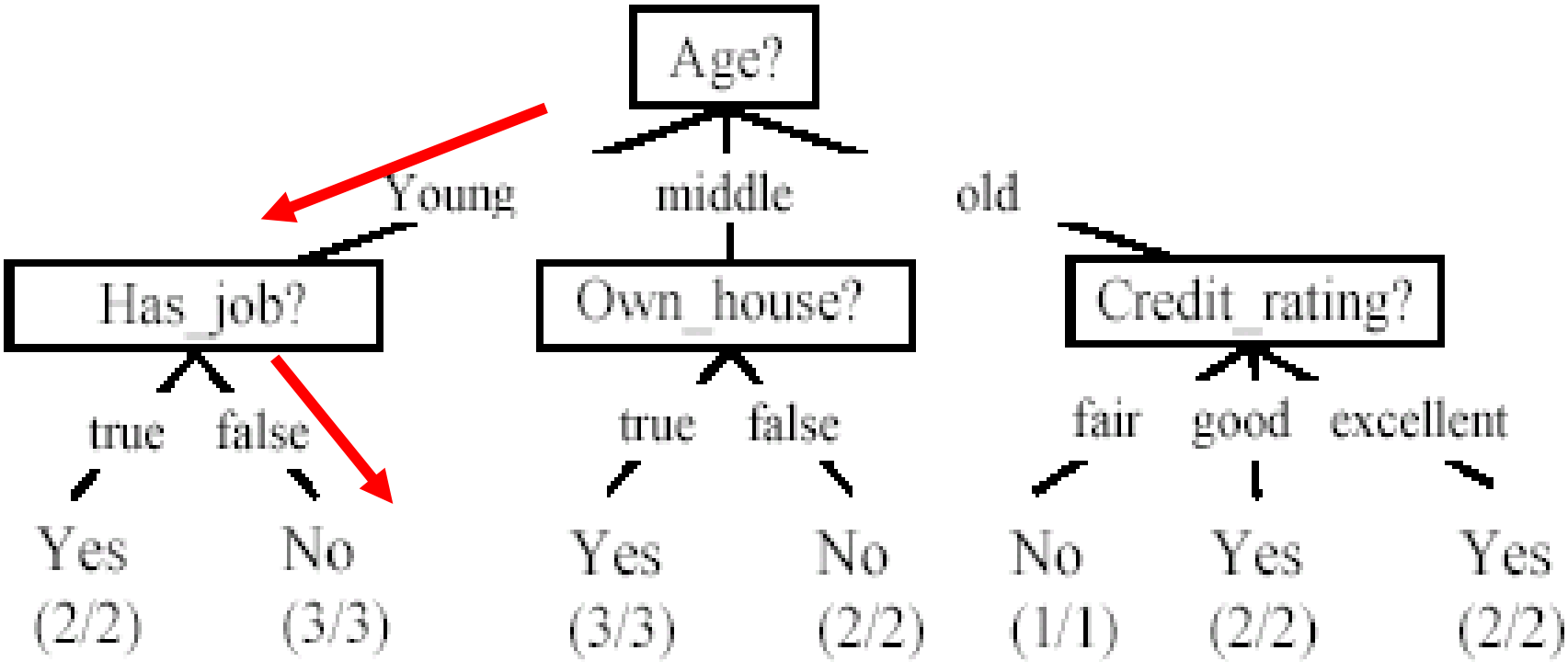
Decision nodes and leaf nodes (classes)



Use the decision tree

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

No

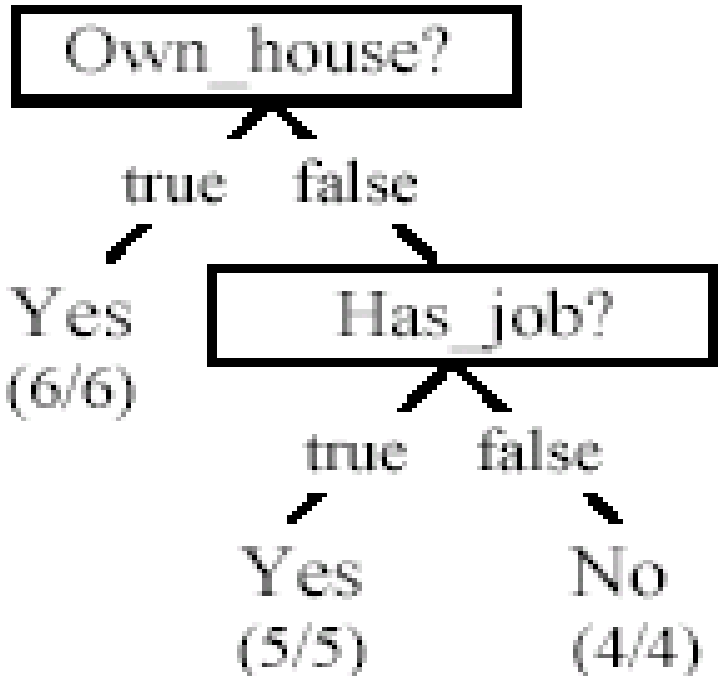


Is the decision tree unique?

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.

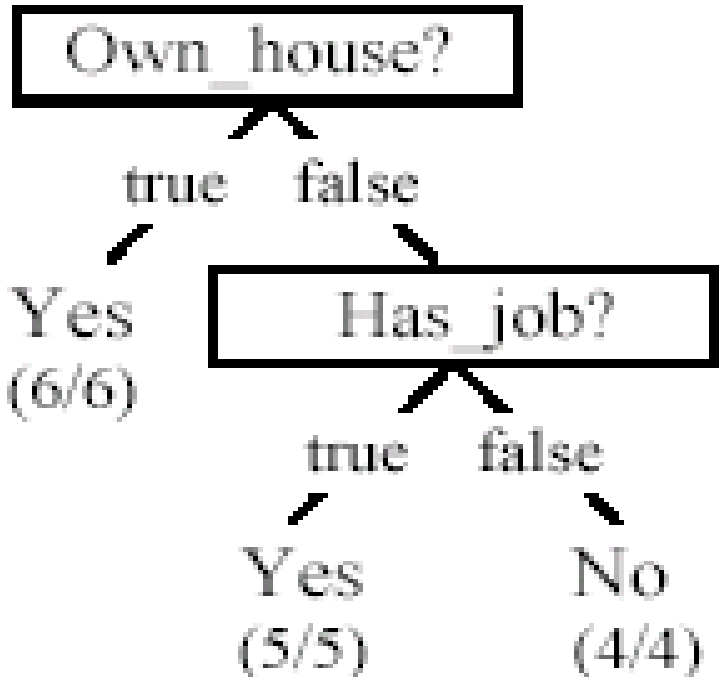
All current tree building algorithms are heuristic algorithms

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)

```
1     if D contains only training examples of the same class c_j ∈ C then
2        make T a leaf node labeled with class c_j;
3     elseif A = ∅ then
4        make T a leaf node labeled with c_j, which is the most frequent class in D
5     else  // D contains examples belonging to a mixture of classes. We select a single
6           // attribute to partition D into subsets so that each subset is purer
7        p_0 = impurityEval-1(D);
8        for each attribute A_i ∈ {A_1, A_2, ..., A_k} do
9           p_i = impurityEval-2(A_i, D)
10       end
11       Select A_g ∈ {A_1, A_2, ..., A_k} that gives the biggest impurity reduction,
            computed using p_0 − p_i;
12       if p_0 − p_g < threshold then    // A_g does not significantly reduce impurity p_0
13          make T a leaf node labeled with c_j, the most frequent class in D.
14       else                             // A_g is able to reduce impurity p_0
15          Make T a decision node on A_g;
16          Let the possible values of A_g be v_1, v_2, ..., v_m. Partition D into m
               disjoint subsets D_1, D_2, ..., D_m based on the m values of A_g.
17          for each D_j in {D_1, D_2, ..., D_m} do
18             if D_j ≠ ∅ then
19                create a branch (edge) node T_j for v_j as a child node of T;
20                decisionTree(D_j, A-{A_g}, T_j)// A_g is removed
21             end
22          end
23       end
24    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?

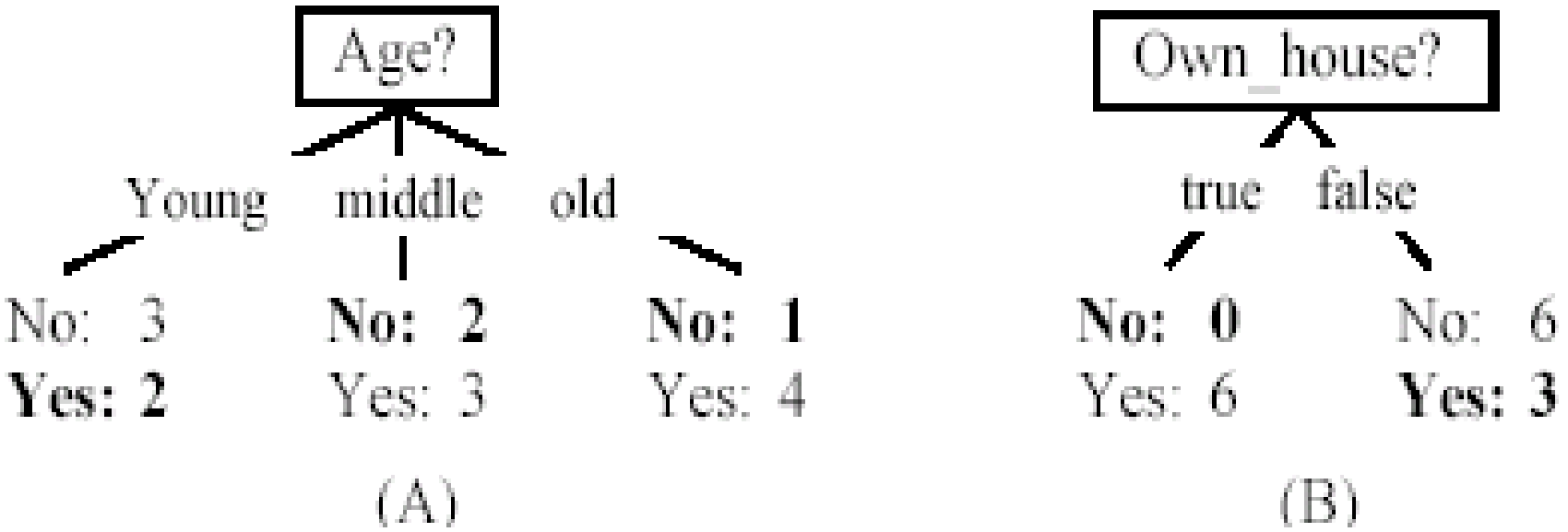


Fig. (B) seems to be better.

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!

Information gain

- Given a set of examples D, we first compute its entropy:

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

- If we make attribute A_i, with v values, the root of the current tree, this will partition D into v subsets $D_1, D_2 \ldots, D_v$. The expected entropy if A_i is used as the current root:

$$entropy_{A_i}(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times entropy(D_j)$$

Information gain (cont ...)

- Information gained by selecting attribute A_i to branch or to partition the data is

$$gain(D, A_i) = entropy(D) - entropy_{A_i}(D)$$

- We choose the attribute with the highest gain to branch/split the current tree.

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

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

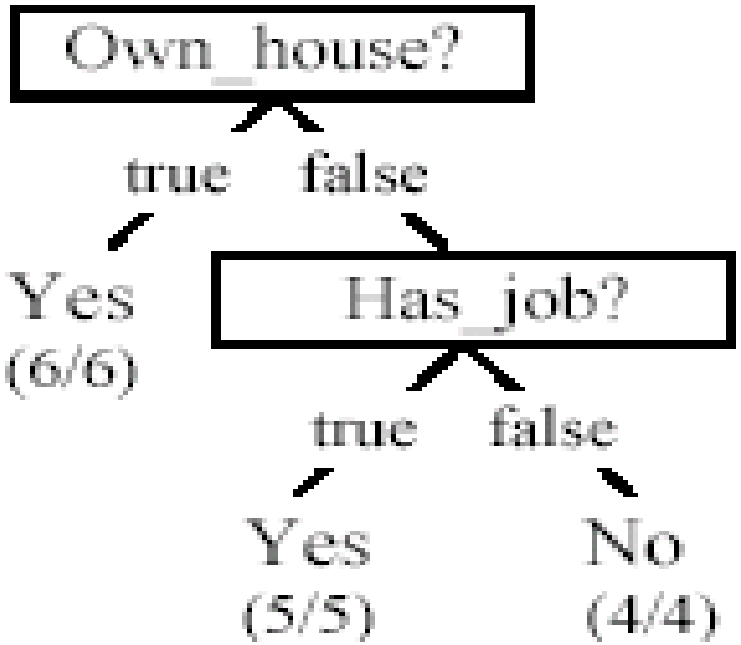
$$entropy_{Age}(D) = -\frac{5}{15} \times entropy(D_1) - \frac{5}{15} \times entropy(D_2) - \frac{5}{15} \times entropy(D_3)$$
$$= \frac{5}{15} \times 0.971 + \frac{5}{15} \times 0.971 + \frac{5}{15} \times 0.722$$
$$= 0.888$$

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

Own_house is the best choice for the root.

$gain(D, \text{Age}) = 0.971 - 0.888 = 0.083$
$gain(D, \text{Own\_house}) = 0.971 - 0.551 = 0.420$
$gain(D, \text{Has\_Job}) = 0.971 - 0.647 = 0.324$
$gain(D, \text{Credit\_Rating}) = 0.971 - 0.608 = 0.363$

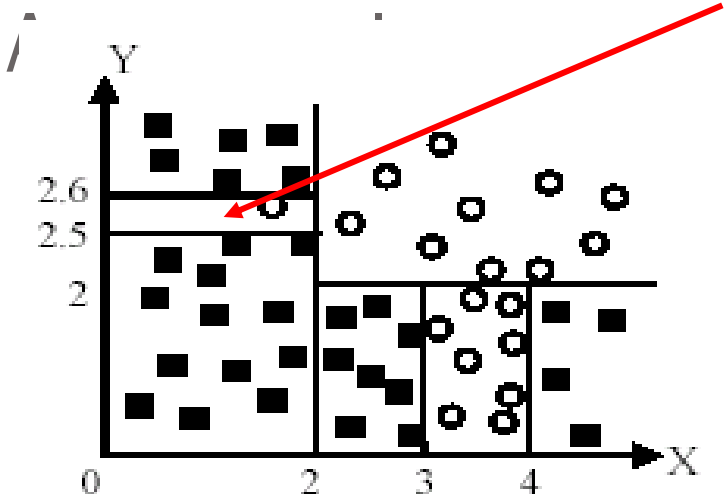
We build the final tree



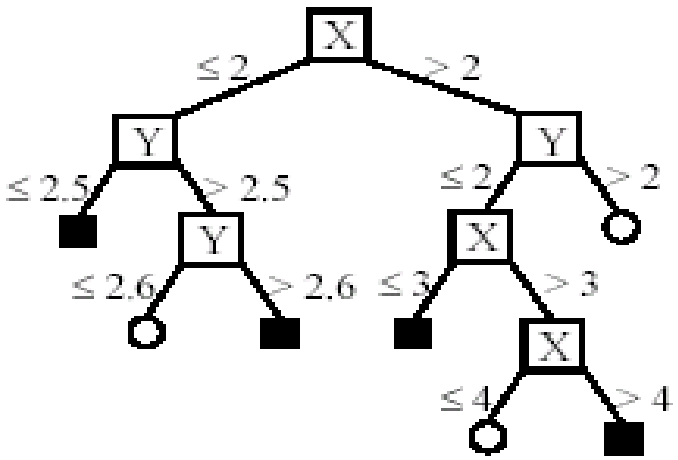
Avoid overfitting in classification

- 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
  - Trade-off full consistency for compactness
    - Larger decision trees can be more consistent
    - Smaller decision trees generalize better
- Two approaches to avoid overfitting
  - Pre-pruning: Halt tree construction early
    - Difficult to decide because we do not know what may happen subsequently if we keep growing the tree.
  - Post-pruning: Remove branches or sub-trees from a "fully grown" tree.
    - This method is commonly used. C4.5 uses a statistical method to estimates the errors at each node for pruning.
    - A validation set may be used for pruning as well.

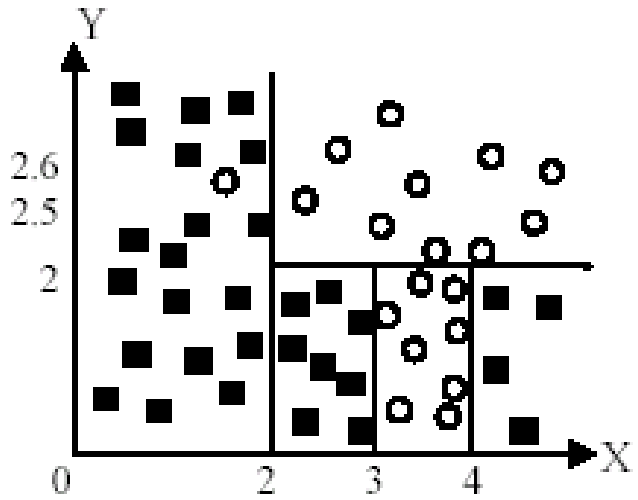
Likely to overfit the data

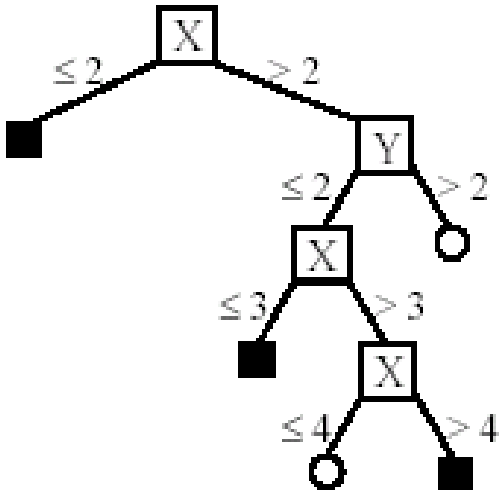


(A) A partition of the data space



(B). The decision tree





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

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
- The Hopfield network
- Bidirectional associative memories (BAM)
- Summary

Introduction, or how the brain works

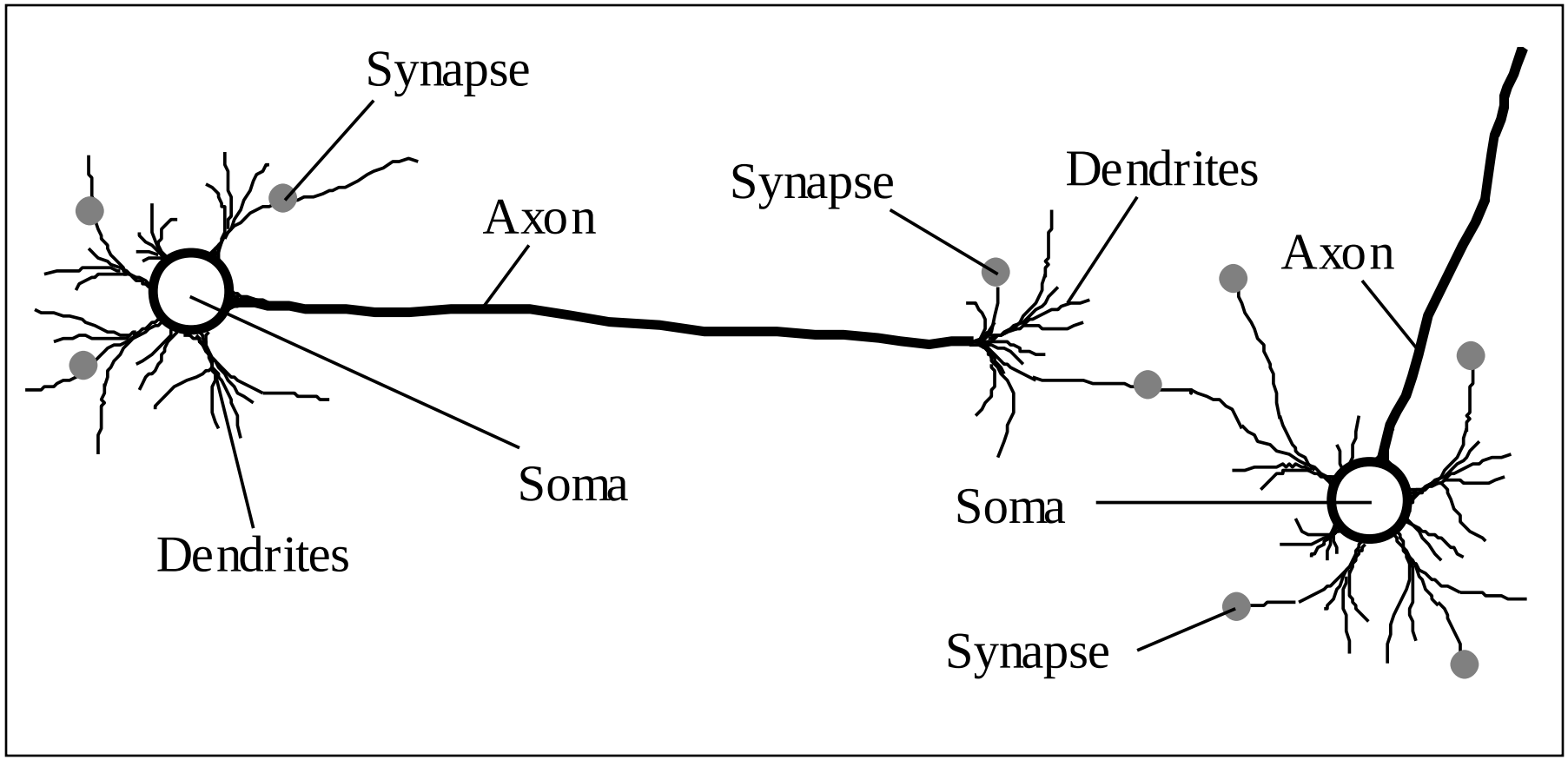
- Machine learning involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by analogy.
- Learning capabilities can improve the performance of an intelligent system over time.
- The most popular approache to machine learning is **artificial neural networks**

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.

- 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**.

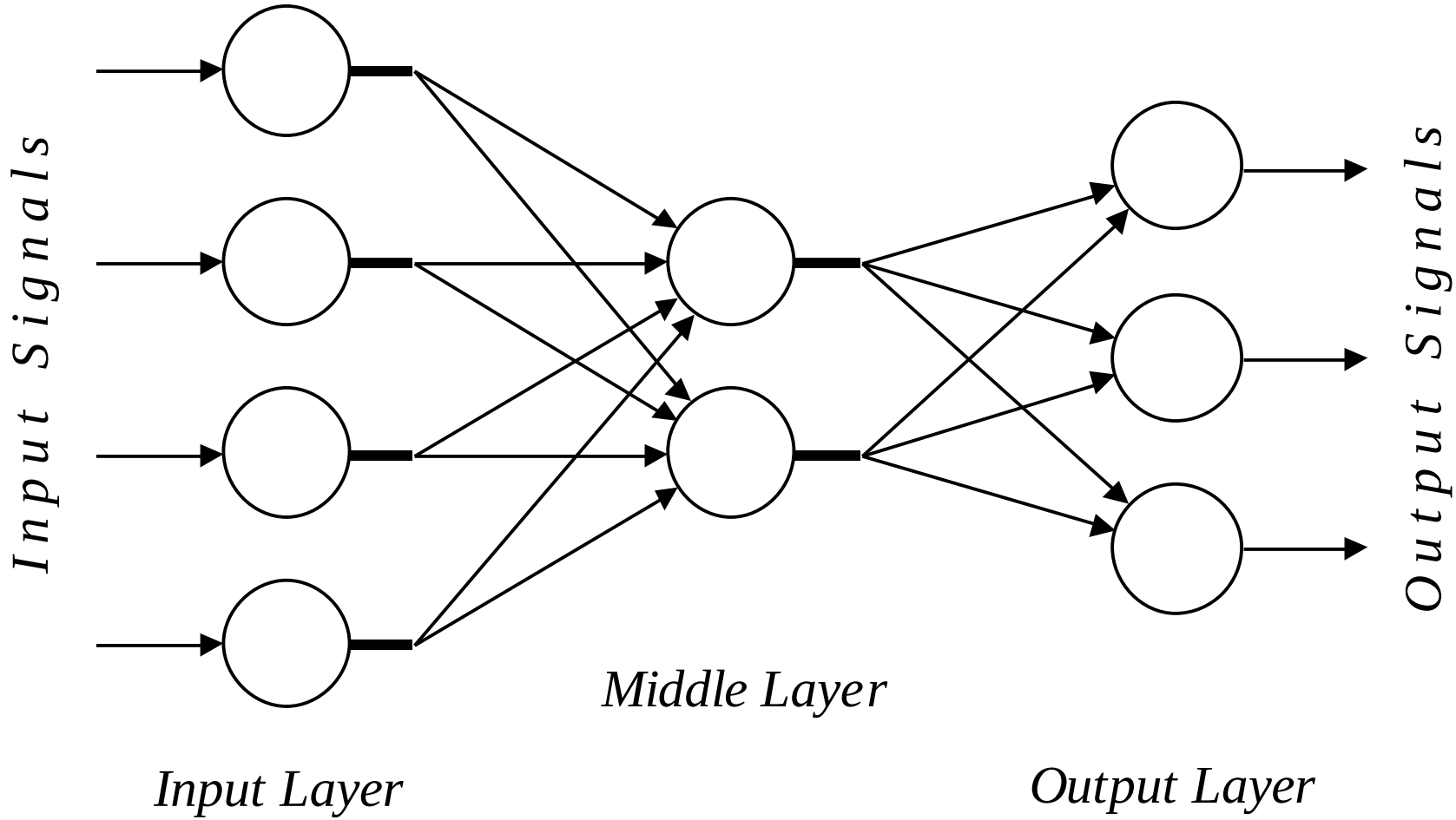
Biological neural network



- Our brain can be considered as a highly complex, non-linear and parallel information-processing system.
- Information is stored and processed in a neural network simultaneously throughout the whole network, rather than at specific locations. In other words, in neural networks, both data and its processing are **global** rather than local.
- Learning is a fundamental and essential characteristic of biological neural networks. The ease with which they can learn led to attempts to emulate a biological neural network in a computer.

- 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

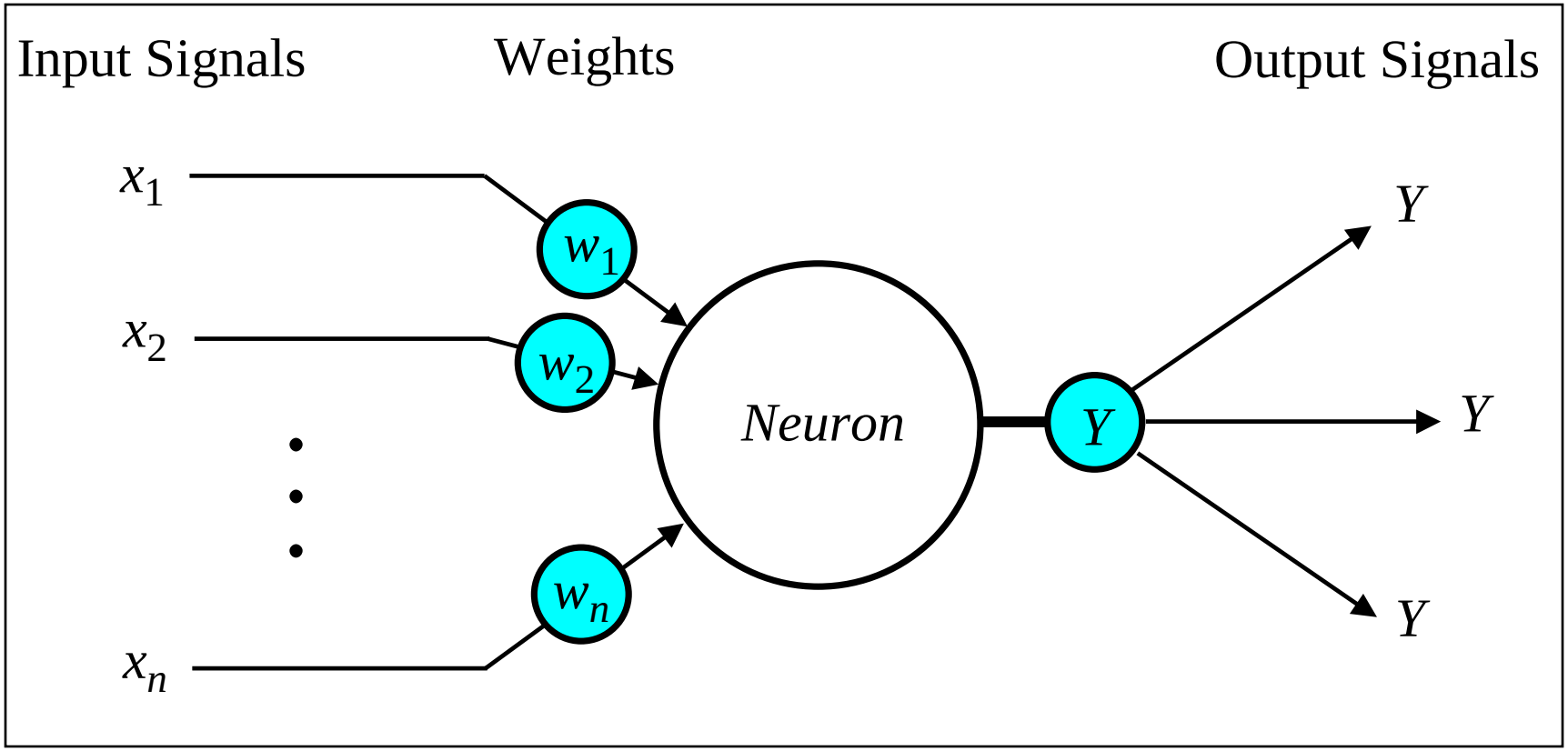


Analogy between biological and artificial neural networks

Biological Neural Network	*Artificial Neural Network*
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight

The neuron as a simple computing element

Diagram of a neuron



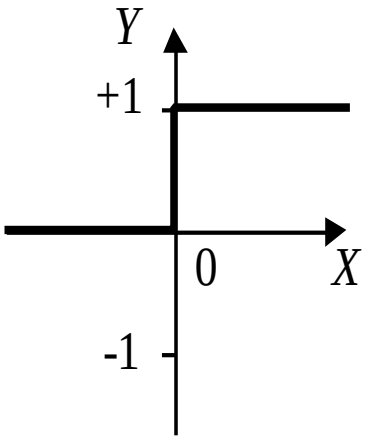
- The neuron computes the weighted sum of the input signals and compares the result with a **threshold value**, θ. If the net input is less than the threshold, the neuron output is –1. But if the net input is greater than or equal to the threshold, the neuron becomes activated and its output attains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

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

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

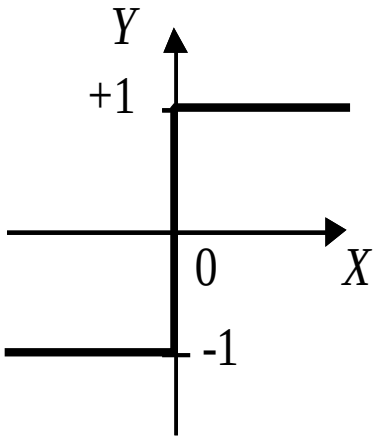
Activation functions of a neuron

Step function



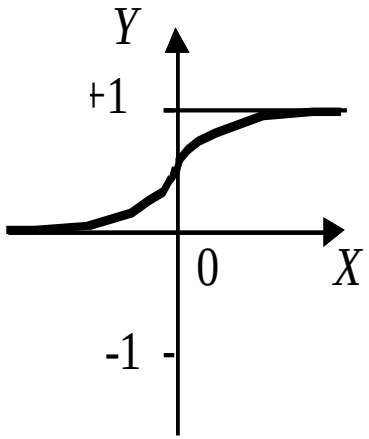
$$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{if } X < 0 \end{cases}$$

Sign function



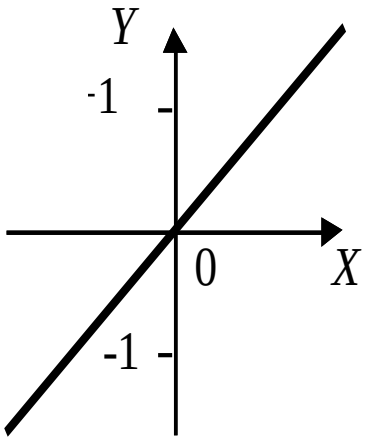
$$Y^{sign} = \begin{cases} +1, \text{if } X \geq 0 \\ -1, \text{if } X < 0 \end{cases}$$

Sigmoid function



$$Y^{sigmoid} = \frac{1}{1+e^{-X}}$$

Linear function

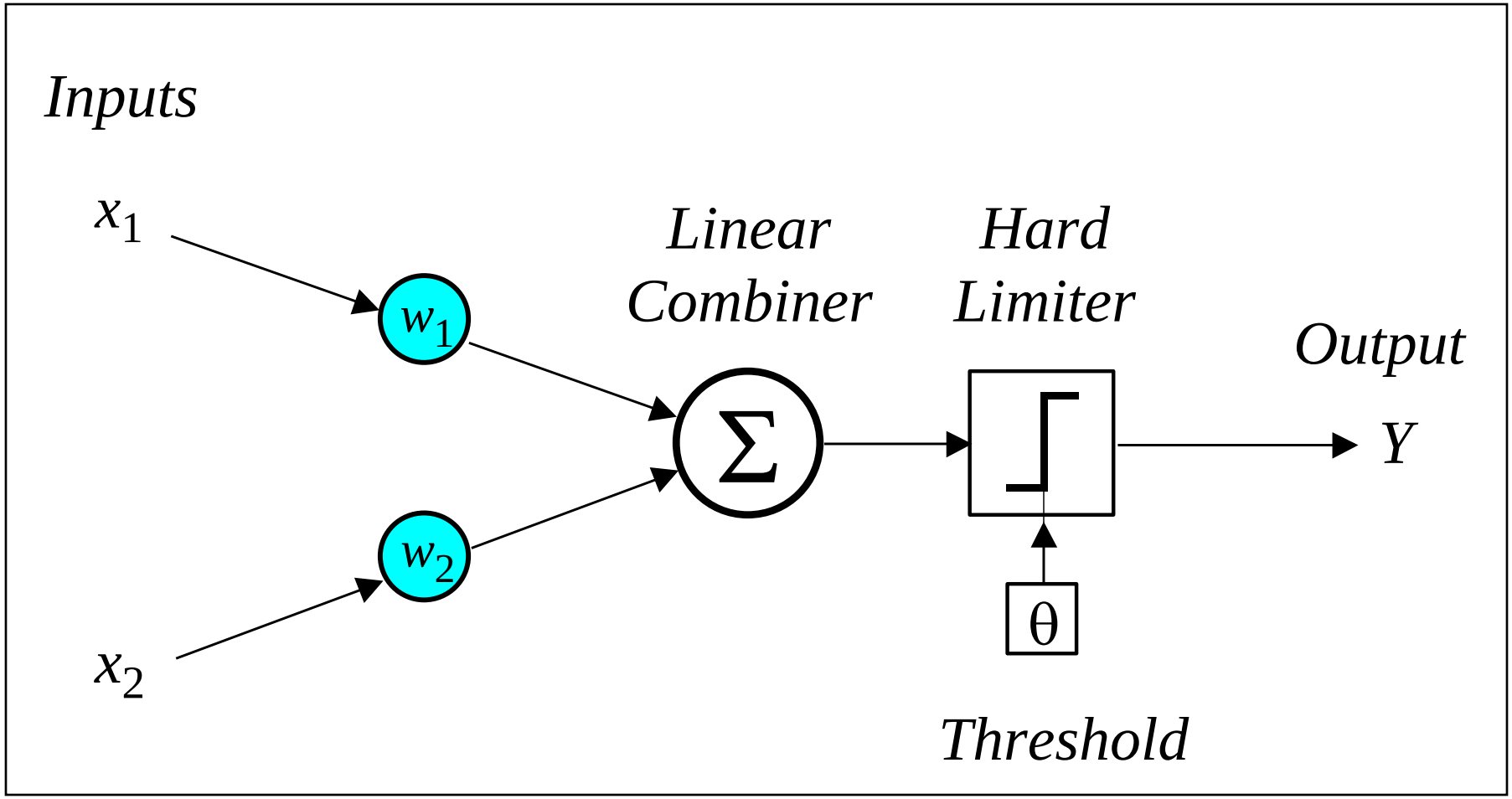


$$Y^{linear} = X$$

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



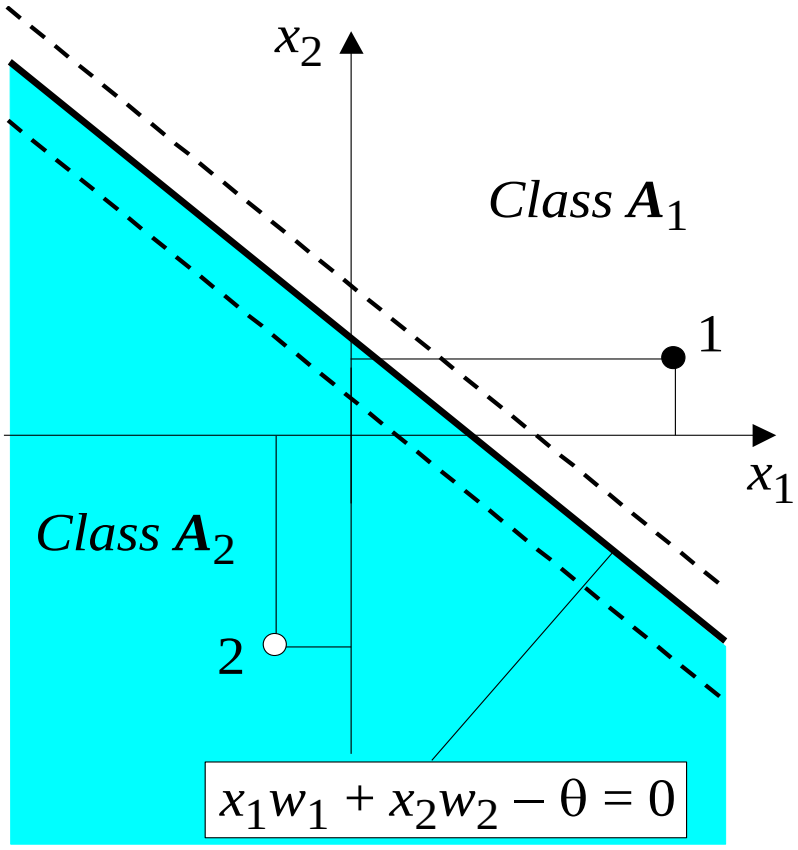
The Perceptron

- 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.

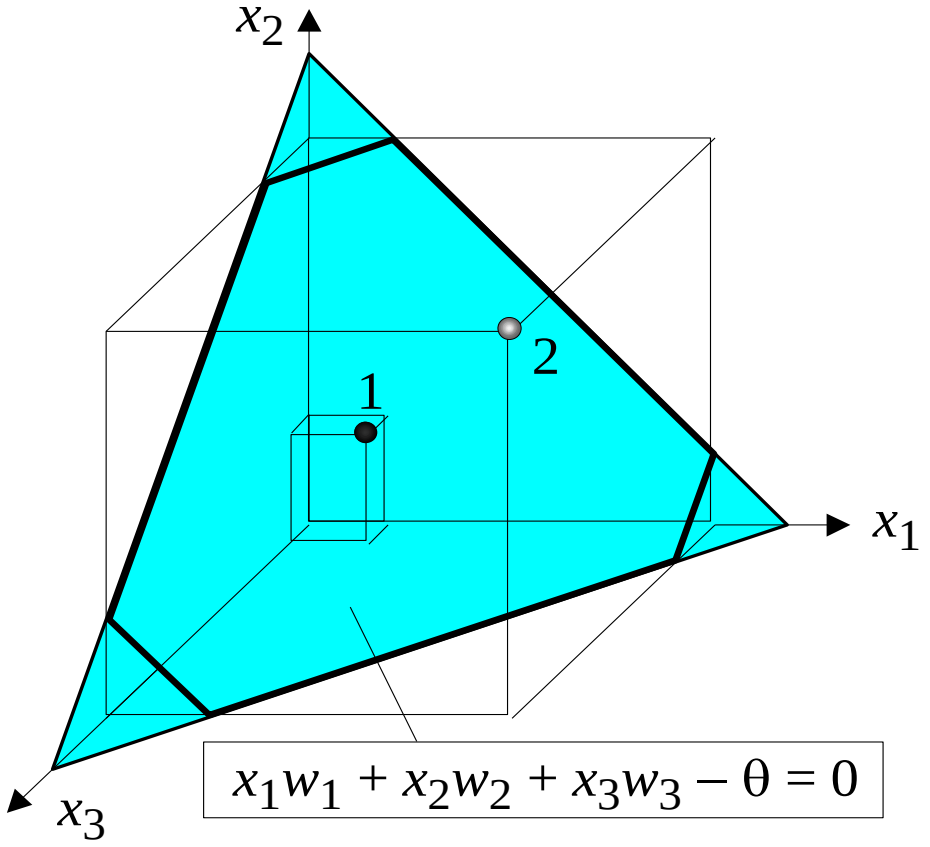
- The aim of the perceptron is to classify inputs, $x_1, x_2, \ldots, x_n$, into one of two classes, say $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$.
- In the case of an elementary perceptron, the n-dimensional space is divided by a *hyperplane* into two decision regions. The hyperplane is defined by the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

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

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.

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

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

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

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

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 $p = 1$

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

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

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

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

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

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

Step 4: Iteration

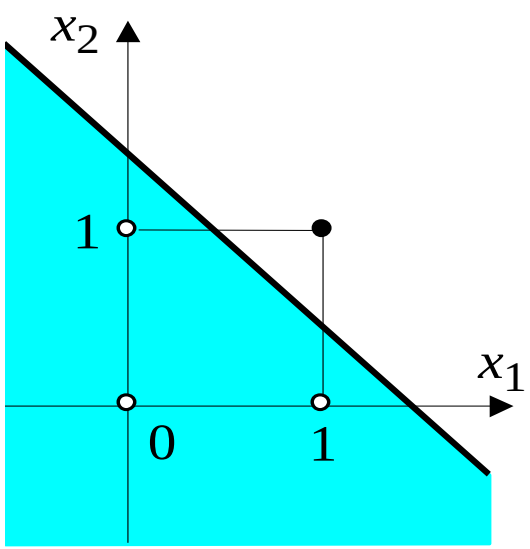
Increase iteration p by one, go back to *Step 2* and repeat the process until convergence.

Example of perceptron learning: the logical operation *AND*

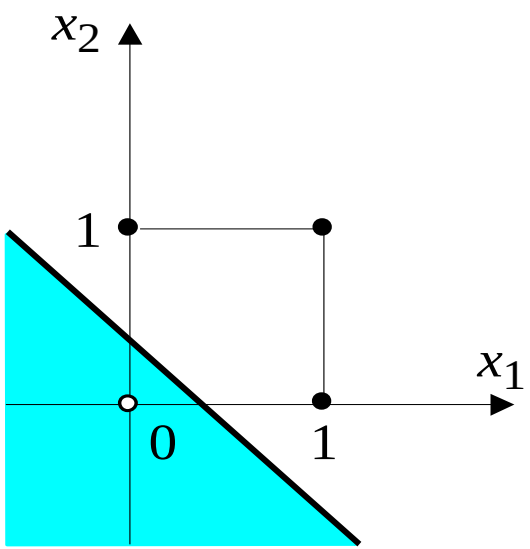
Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	–0.1	0	0	0.3	–0.1
	0	1	0	0.3	–0.1	0	0	0.3	–0.1
	1	0	0	0.3	–0.1	1	–1	0.2	–0.1
	1	1	1	0.2	–0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	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: $\alpha = 0.1$

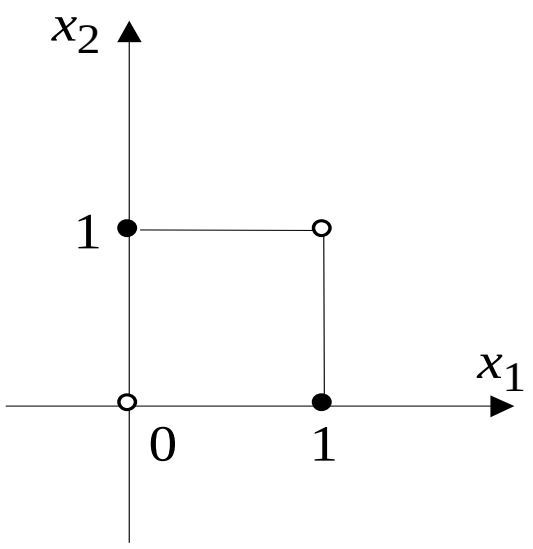
Two-dimensional plots of basic logical operations



(*a*) *AND* ($x_1 \cap x_2$)



(*b*) *OR* ($x_1 \cup x_2$)



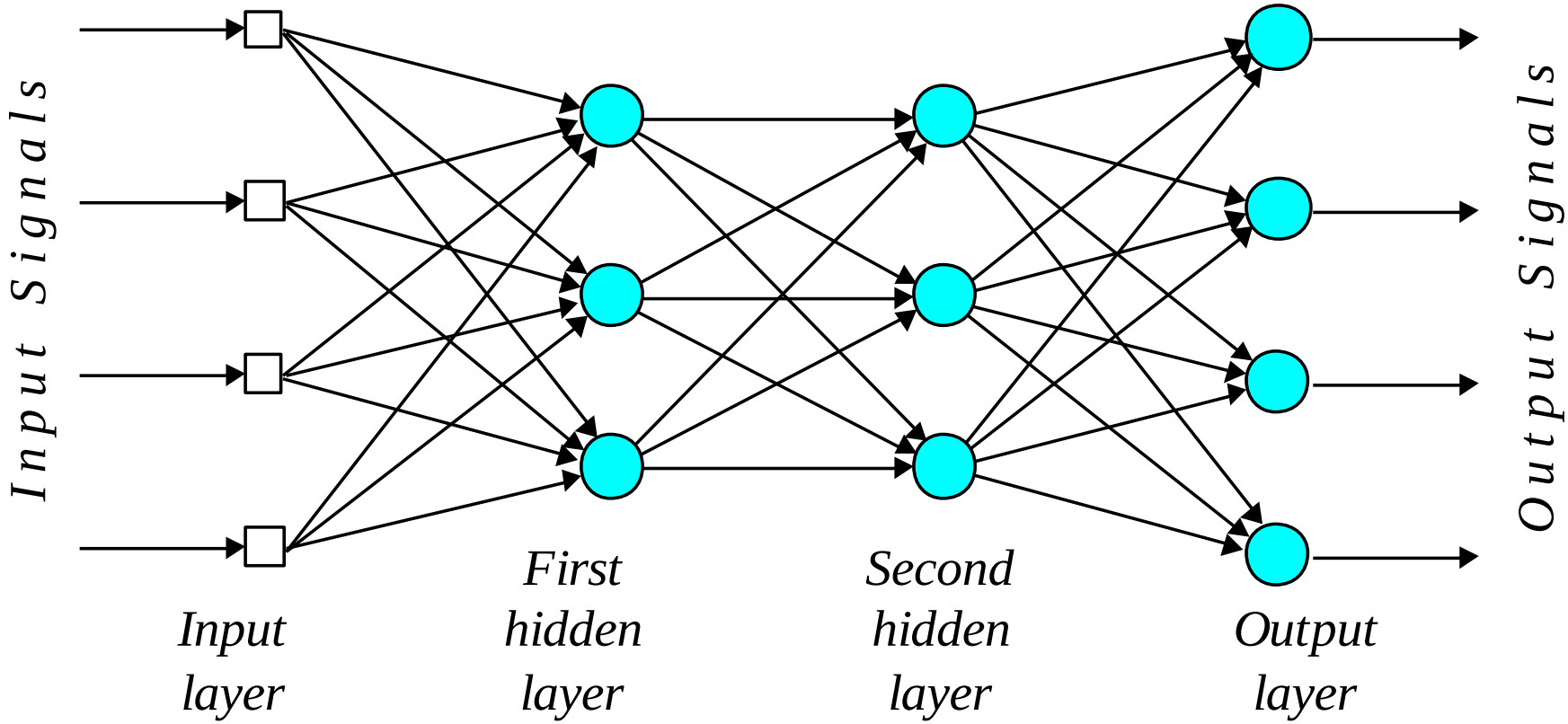
(*c*) *Exclusive-OR* ($x_1 \oplus x_2$)

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?

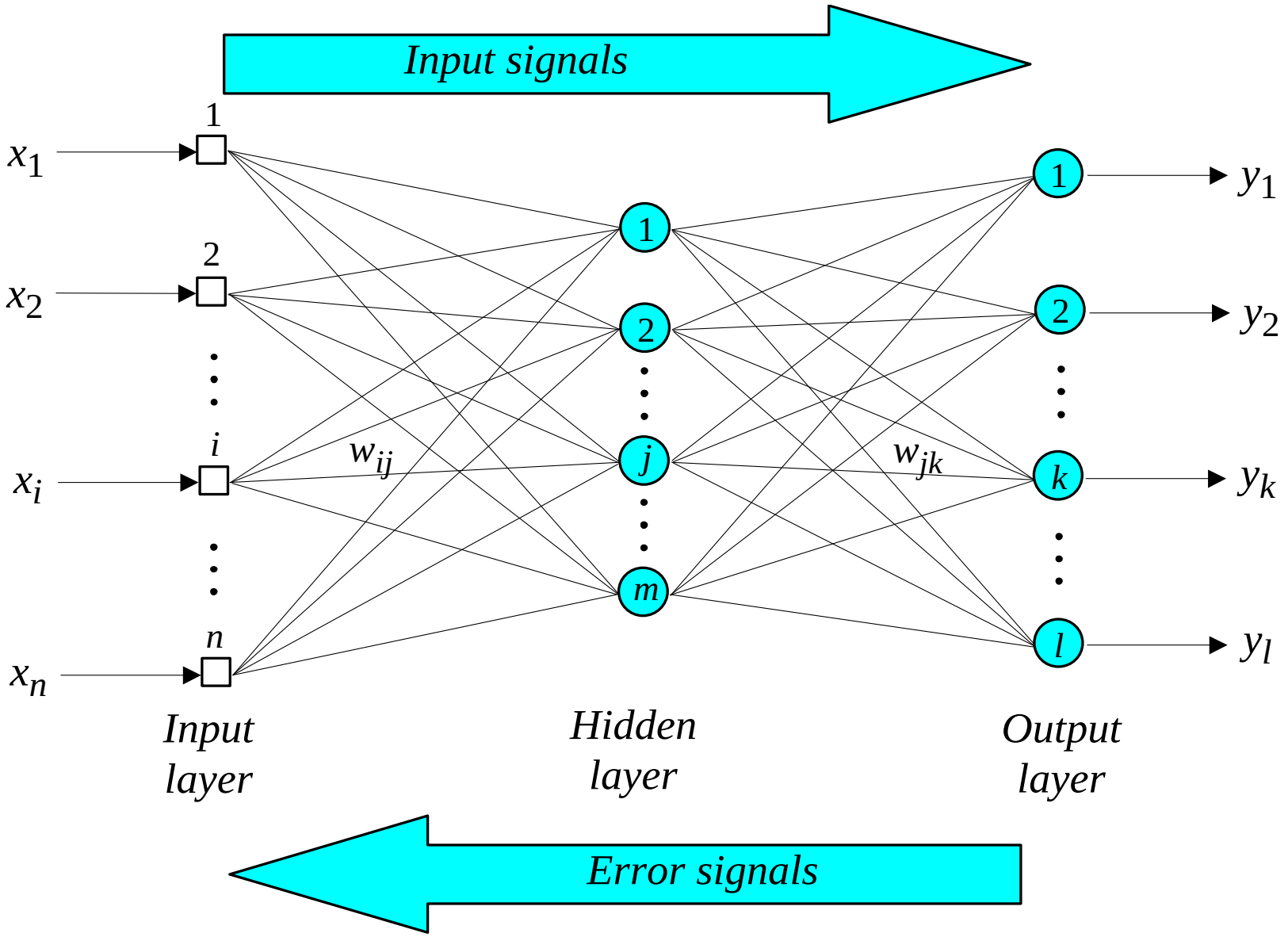
- A hidden layer “hides” its desired output. Neurons in the hidden layer cannot be observed through the input/output behaviour of the network. There is no obvious way to know what the desired output of the hidden layer should be.
- Commercial ANNs incorporate three and sometimes four layers, including 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.

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.

- 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:

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

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:

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

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:

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

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:

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

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

Calculate the weight corrections:

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

Update the weights at the output neurons:

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

Step 3: Weight training (continued)

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

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

Calculate the weight corrections:

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

Update the weights at the hidden neurons:

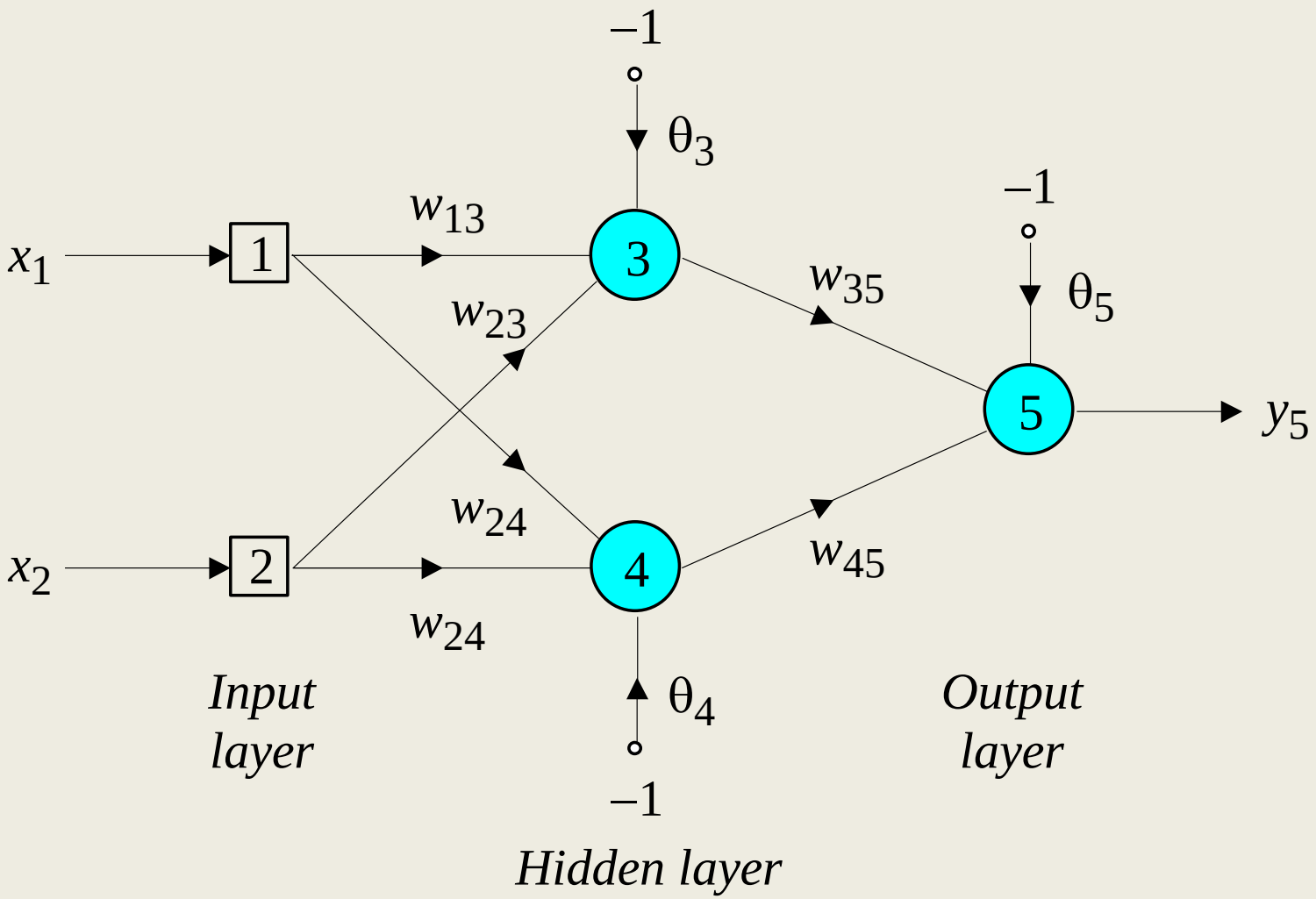
$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

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

$$y_3 = sigmoid\,(x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/\left[1 + e^{-(1\cdot 0.5 + 1\cdot 0.4 - 1\cdot 0.8)}\right] = 0.5250$$

$$y_4 = sigmoid\,(x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/\left[1 + e^{-(1\cdot 0.9 + 1\cdot 1.0 + 1\cdot 0.1)}\right] = 0.8808$$

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

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

- Thus, the following error is obtained:

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

- 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:

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

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

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

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

$$\delta_3 = y_3(1-y_3)\cdot\delta_5\cdot w_{35} = 0.5250\cdot(1-0.5250)\cdot(-0.1274)\cdot(-1.2) = 0.0381$$

$$\delta_4 = y_4(1-y_4)\cdot\delta_5\cdot w_{45} = 0.8808\cdot(1-0.8808)\cdot(-0.127\,4)\cdot 1.1 = -0.0147$$

- We then determine the weight corrections:

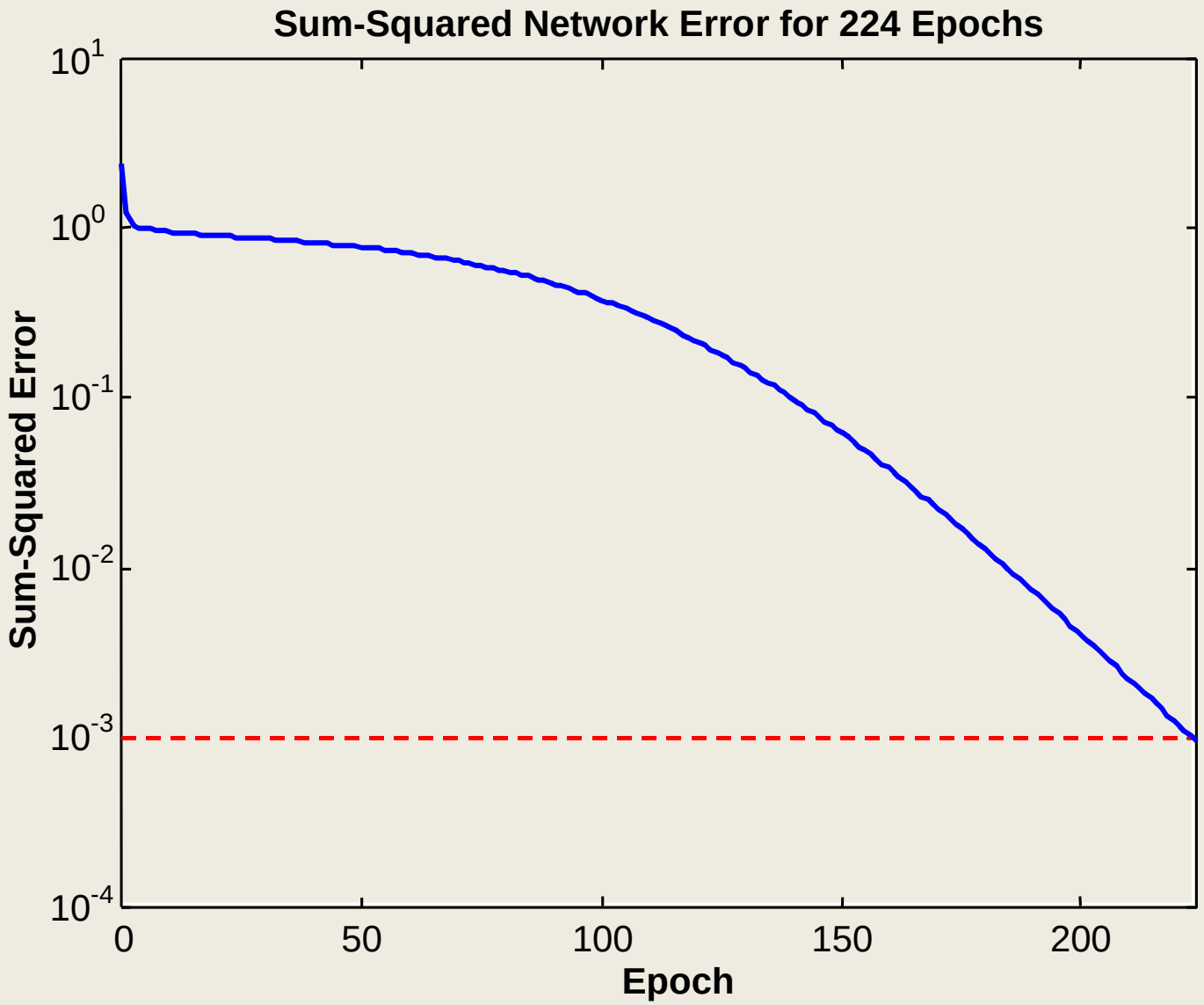
$$\Delta w_{13} = \alpha\cdot x_1\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta w_{23} = \alpha\cdot x_2\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta\theta_3 = \alpha\cdot(-1)\cdot\delta_3 = 0.1\cdot(-1)\cdot 0.0381 = -0.0038$$
$$\Delta w_{14} = \alpha\cdot x_1\cdot\delta_4 =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

- At last, we update all weights and threshold:

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

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

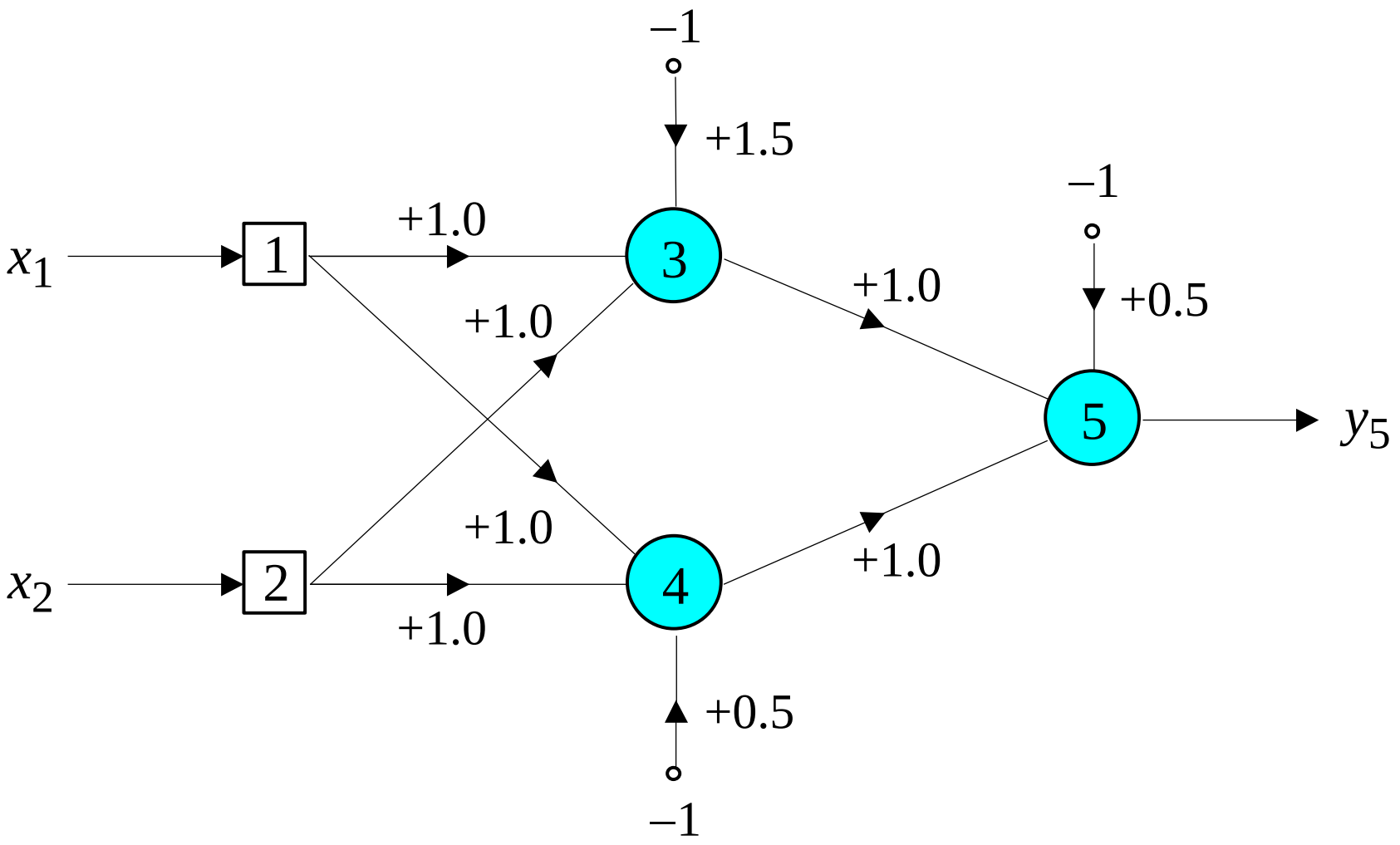
Learning curve for operation *Exclusive-OR*



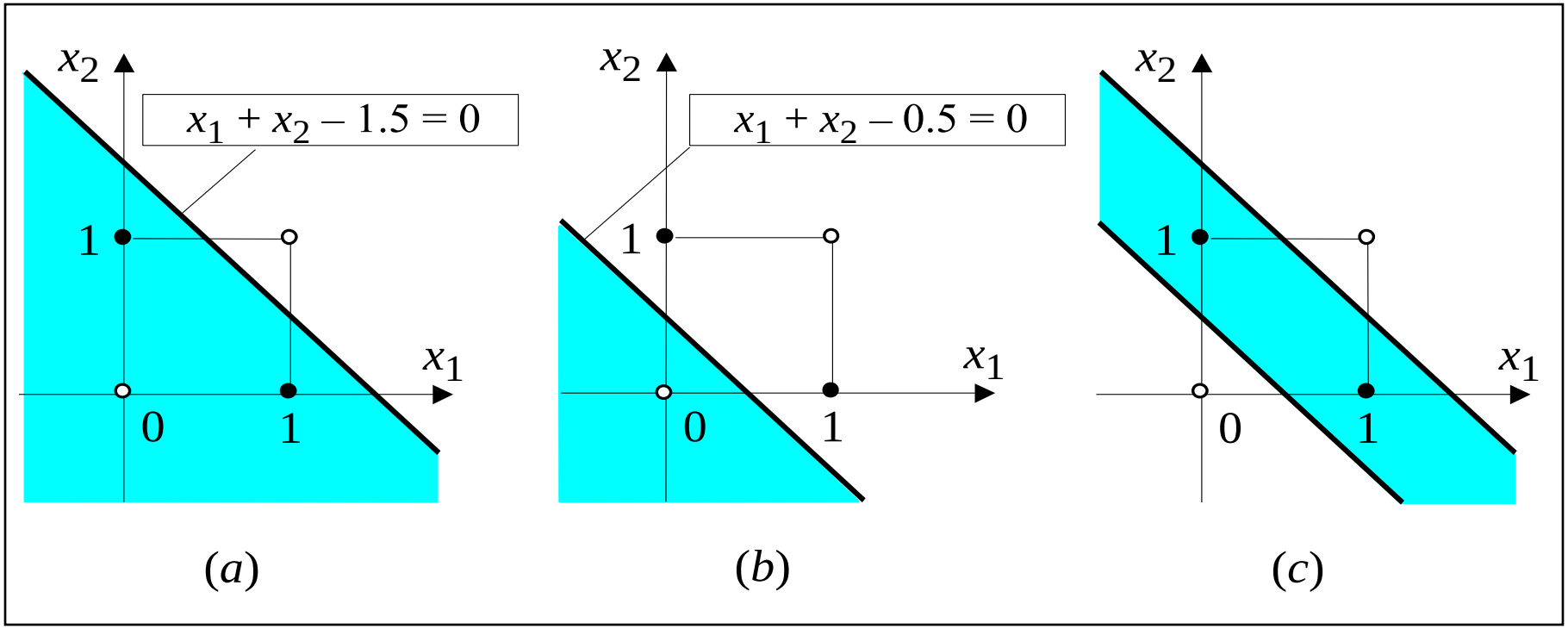
Final results of three-layer network learning

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**:

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

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

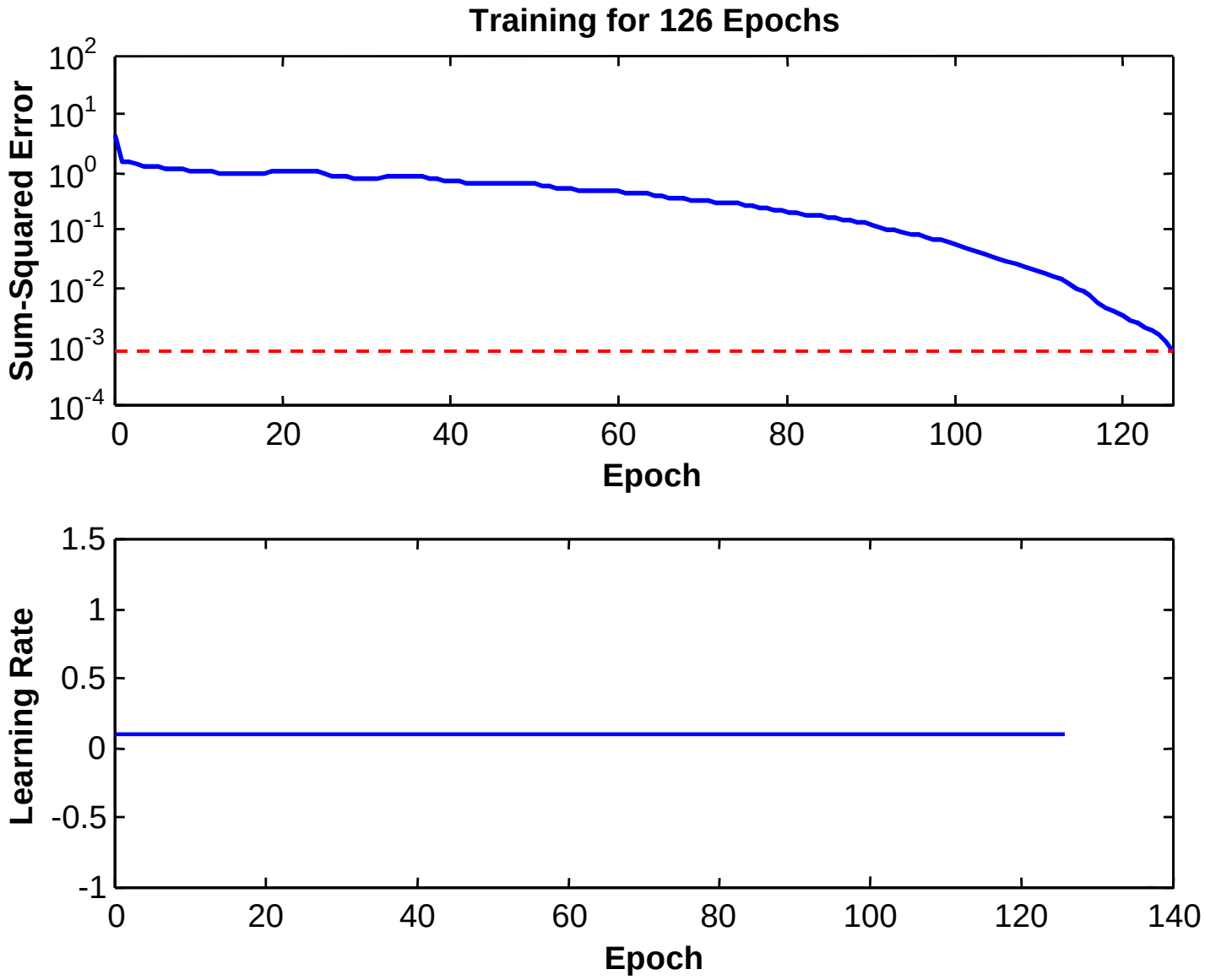
- We also can accelerate training by including a **momentum term** in the delta rule:

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

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

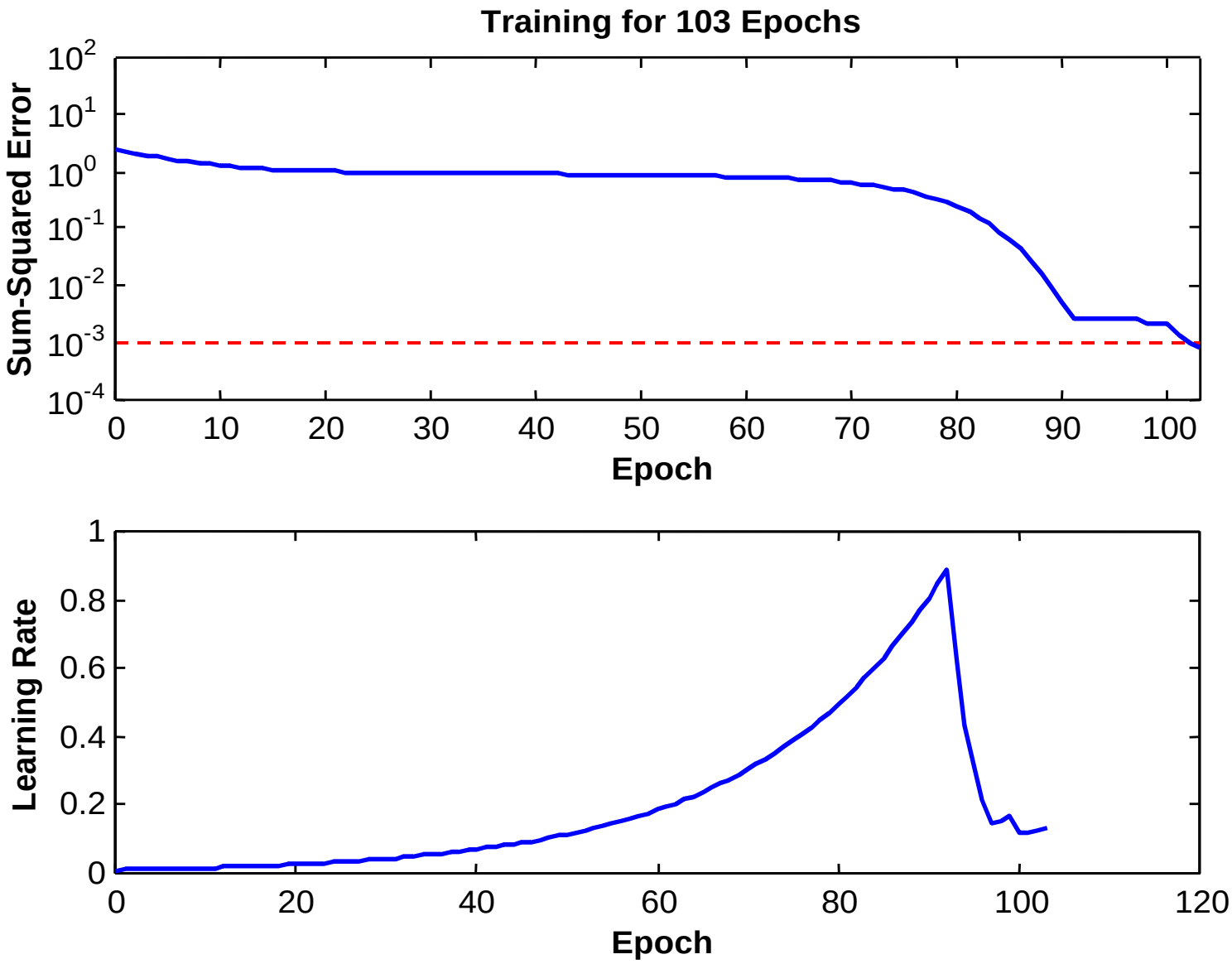
If the change of the sum of squared errors has the same algebraic sign for several consequent epochs, then the learning rate parameter, α, should be increased.

Heuristic 2

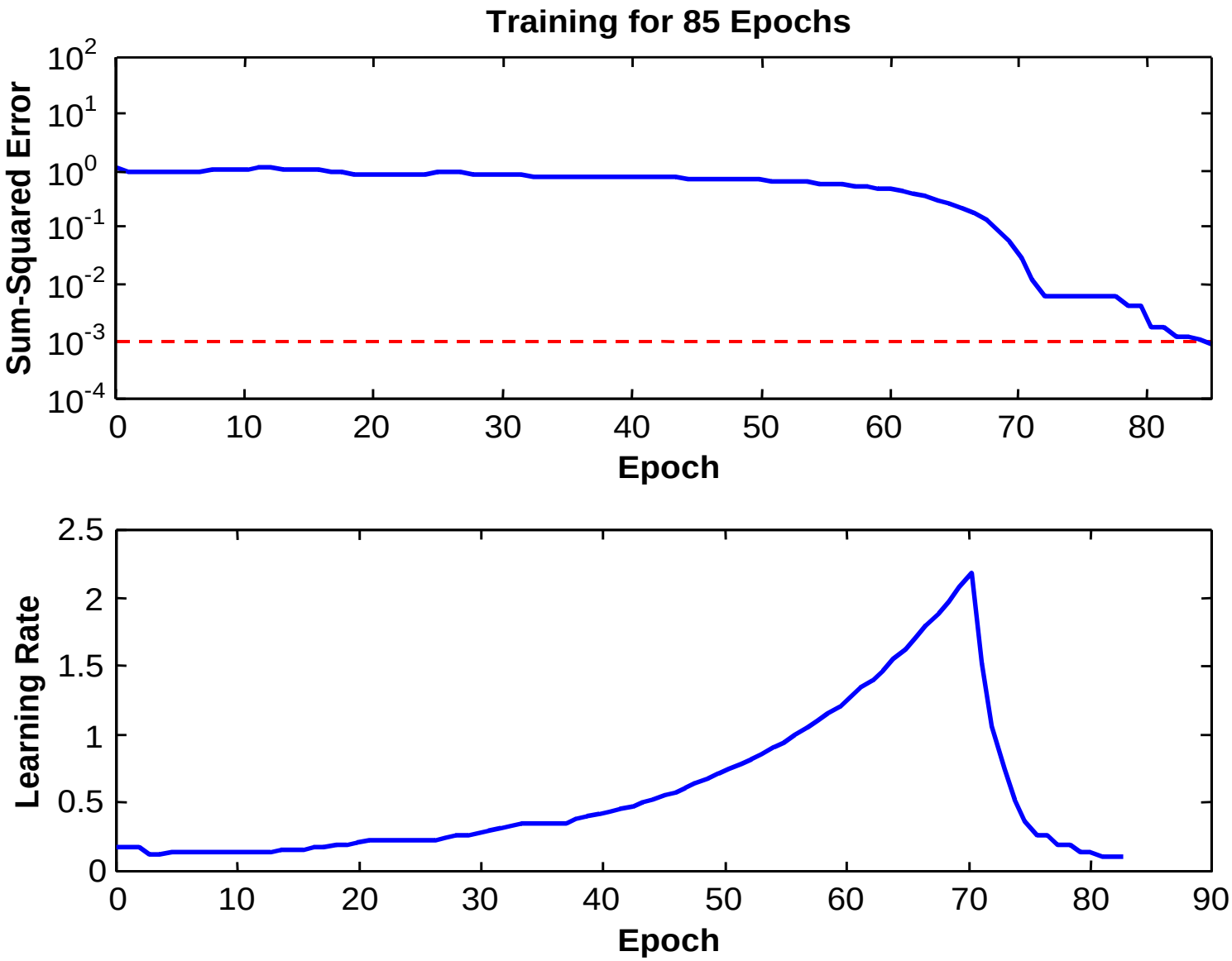
If the algebraic sign of the change of the sum of squared errors alternates for several consequent epochs, then the learning rate parameter, α, should be decreased.

- 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.

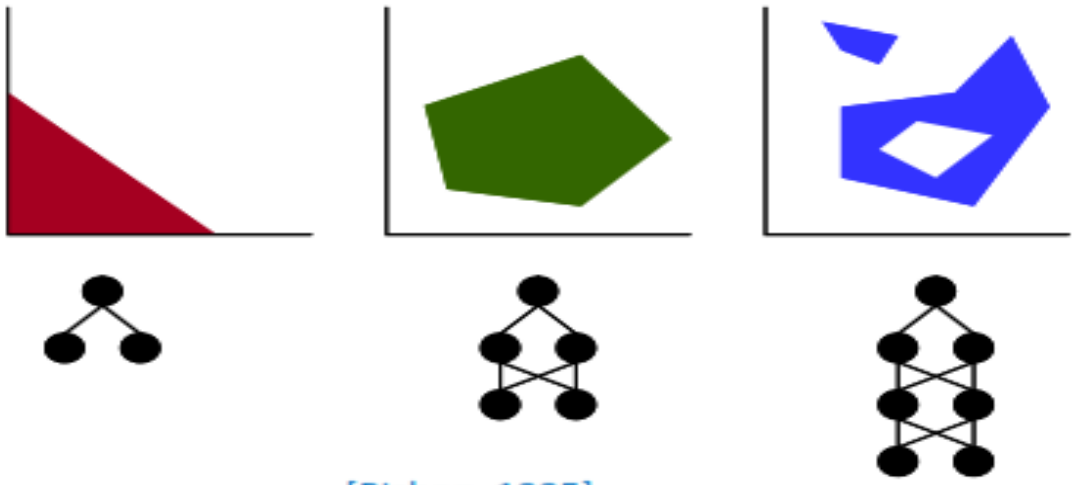
Accelerated learning in multilayer neural networks

- 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

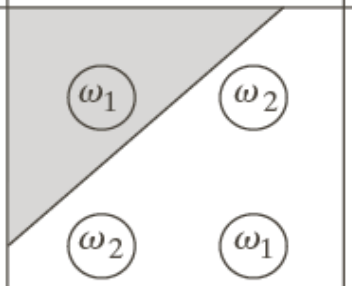
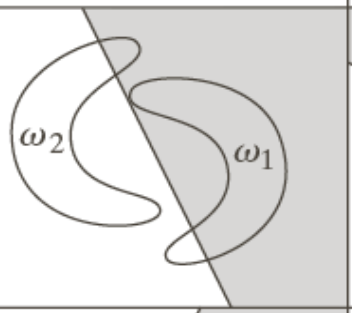
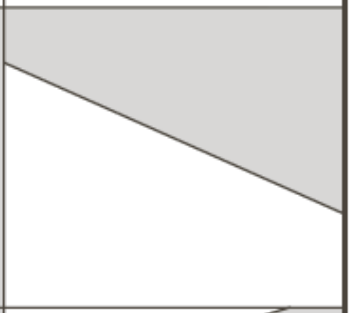
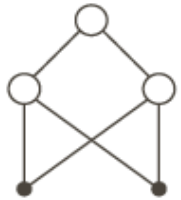
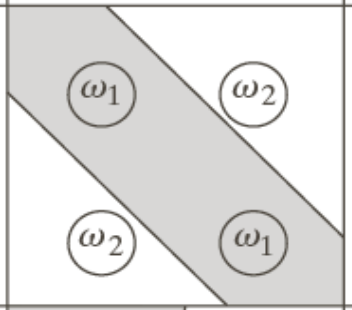
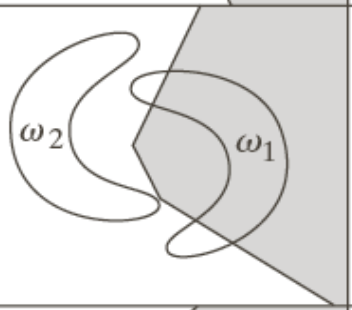
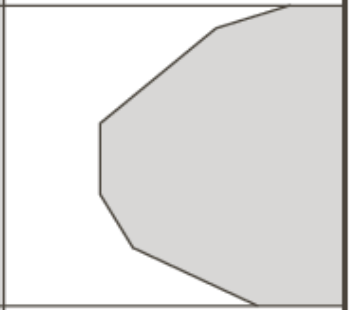
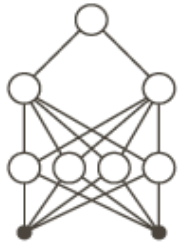
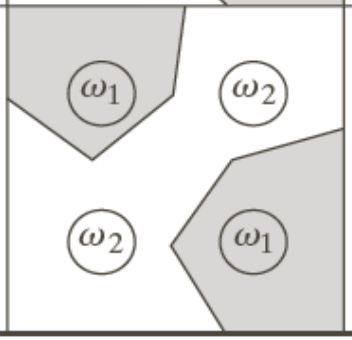
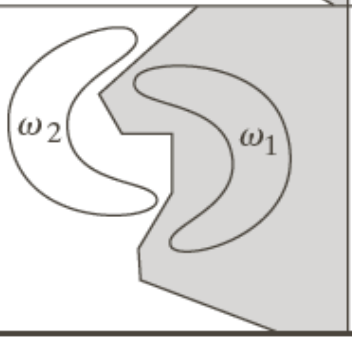
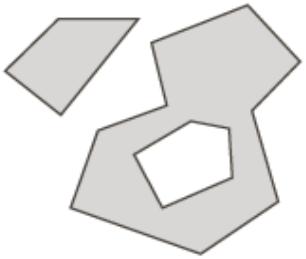
Number of hidden layers

- 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

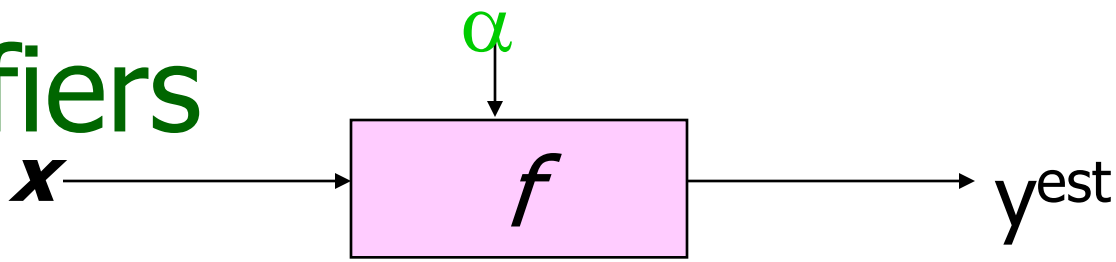
- Initial random weights should be small to avoid driving the neurons into saturation
  - HO weights should be made larger than IH weights since they carry the back propagated error
  - If the initial HO weights are very small, the weight changes at the IH layer will initially be very small, slowing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

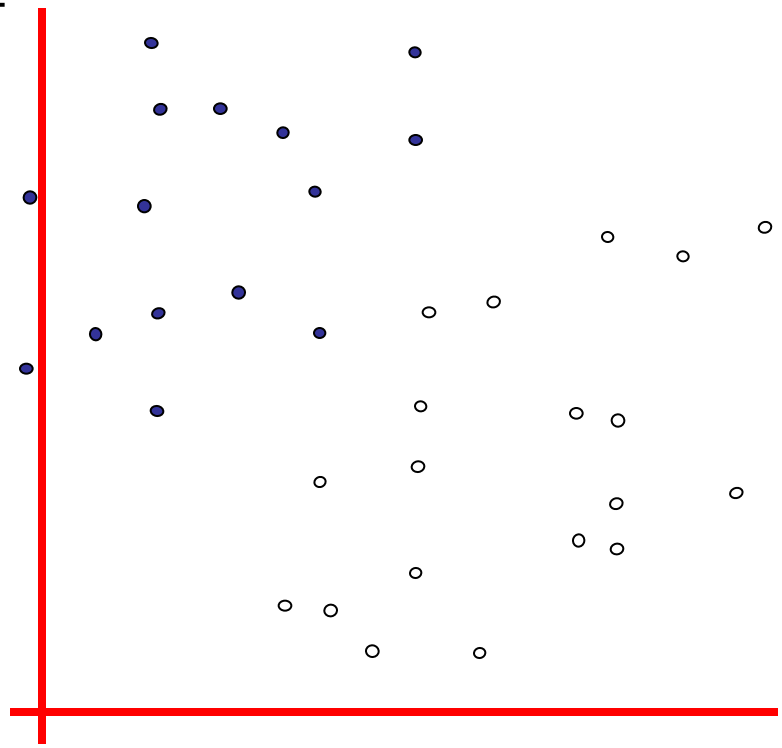
Support Vector Machines

Linear Classifiers



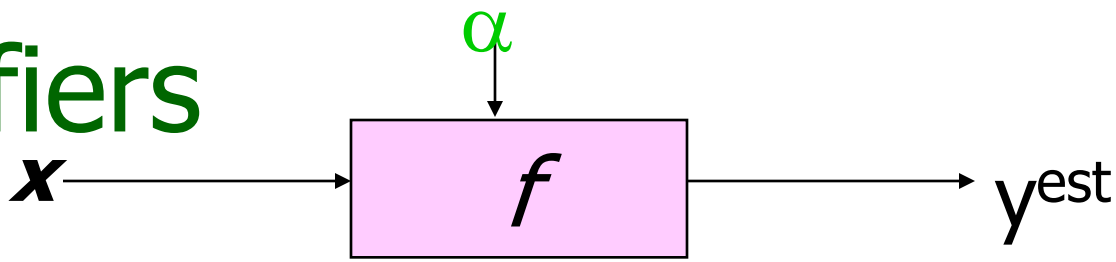
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

- denotes +1
- denotes -1



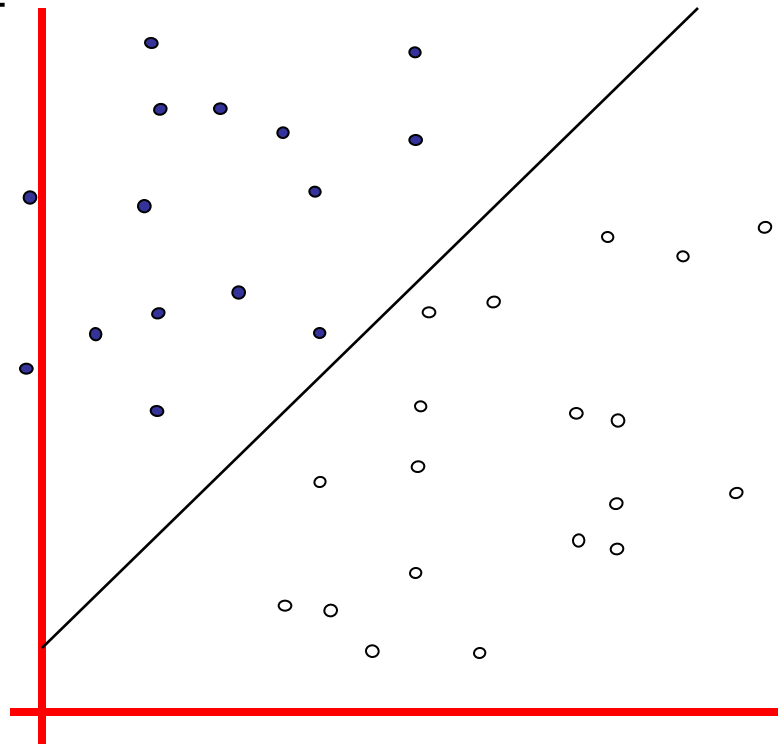
How would you classify this data?

Linear Classifiers



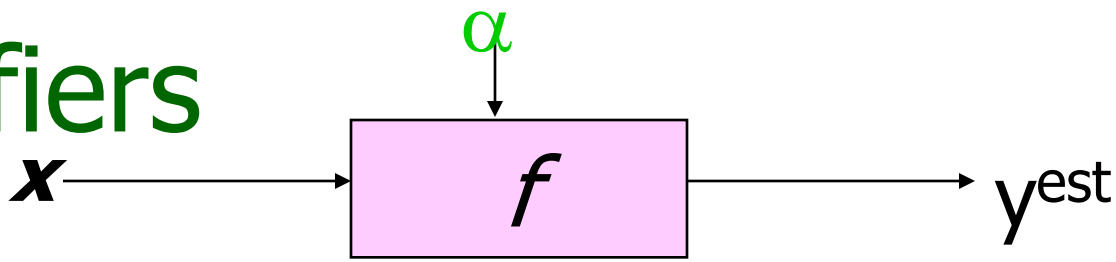
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

- denotes +1
- denotes -1



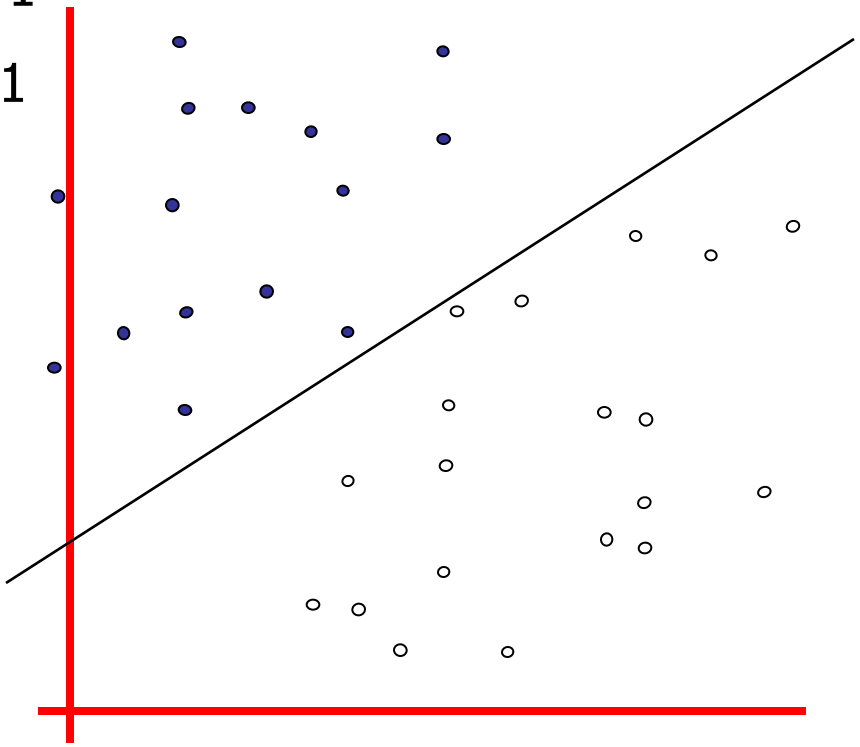
How would you classify this data?

Linear Classifiers



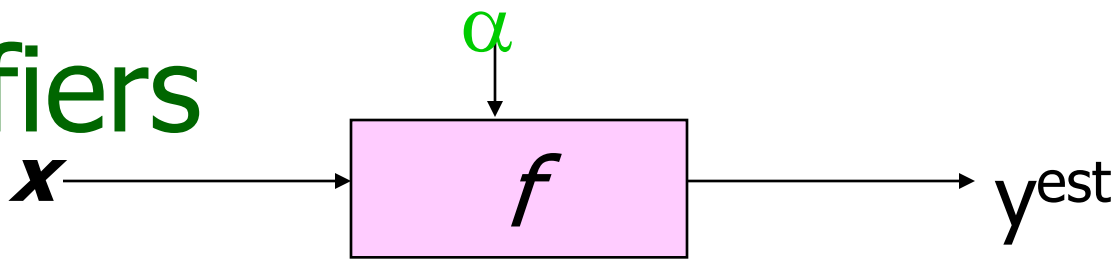
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

- denotes +1
- denotes -1



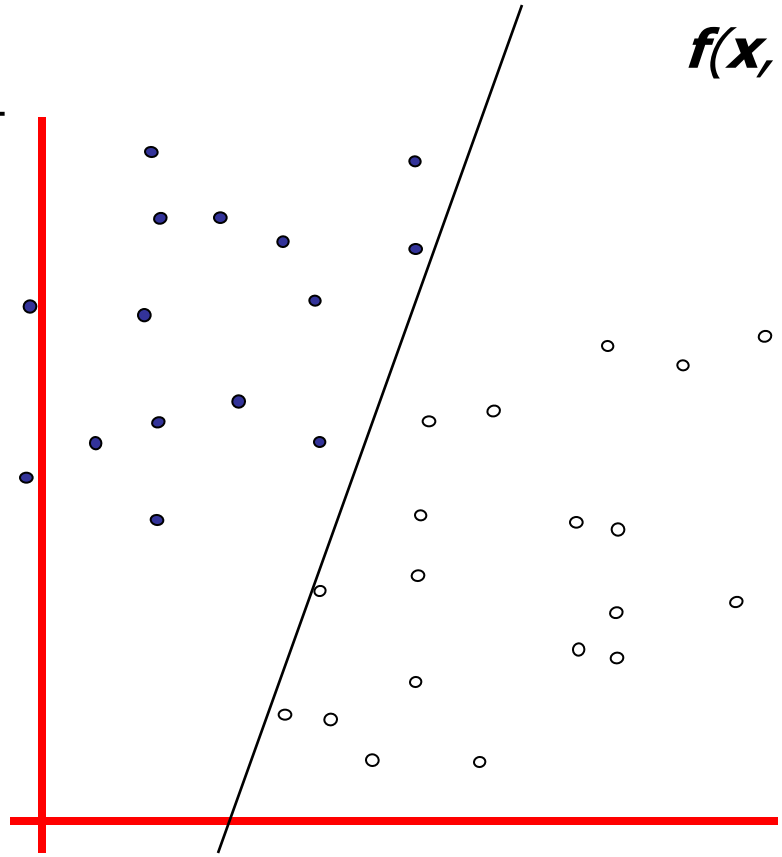
How would you classify this data?

Linear Classifiers



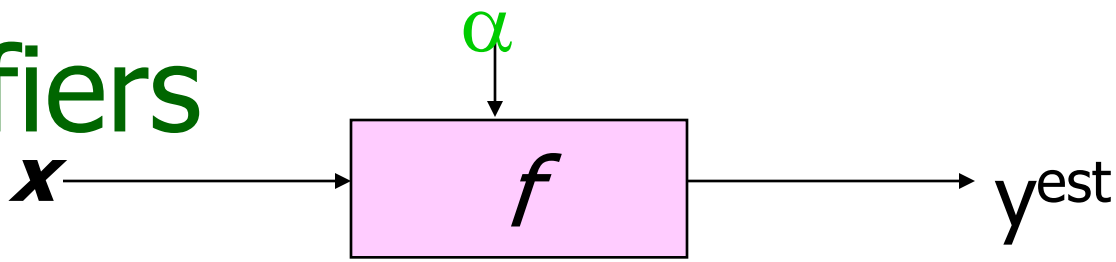
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

- denotes +1
- denotes -1



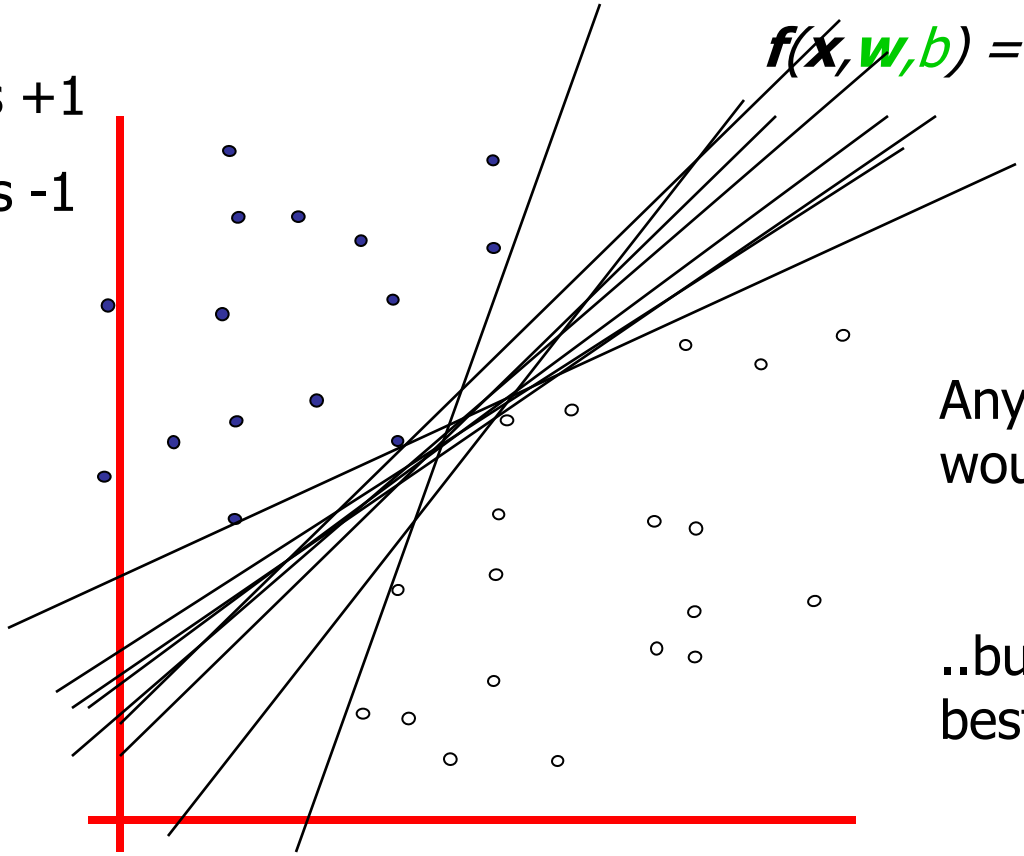
How would you classify this data?

Linear Classifiers



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

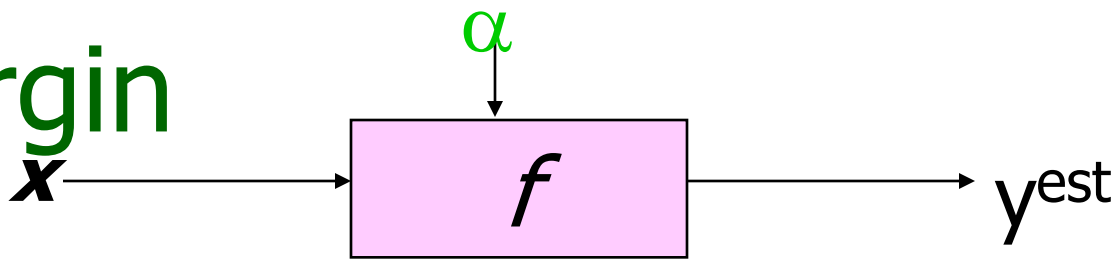
- denotes +1
- denotes -1



Any of these would be fine..

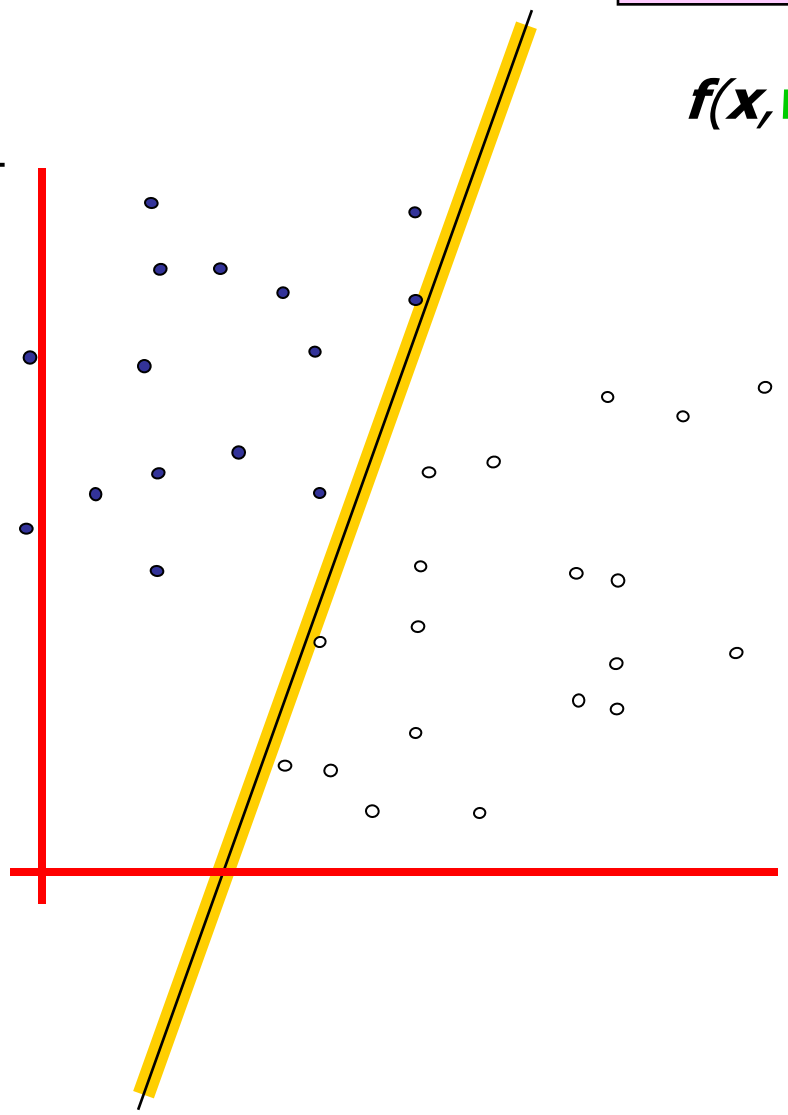
..but which is best?

Classifier Margin



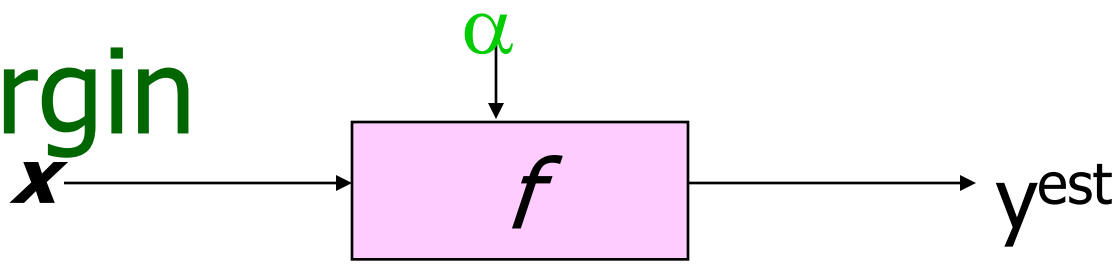
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

- denotes +1
- denotes -1



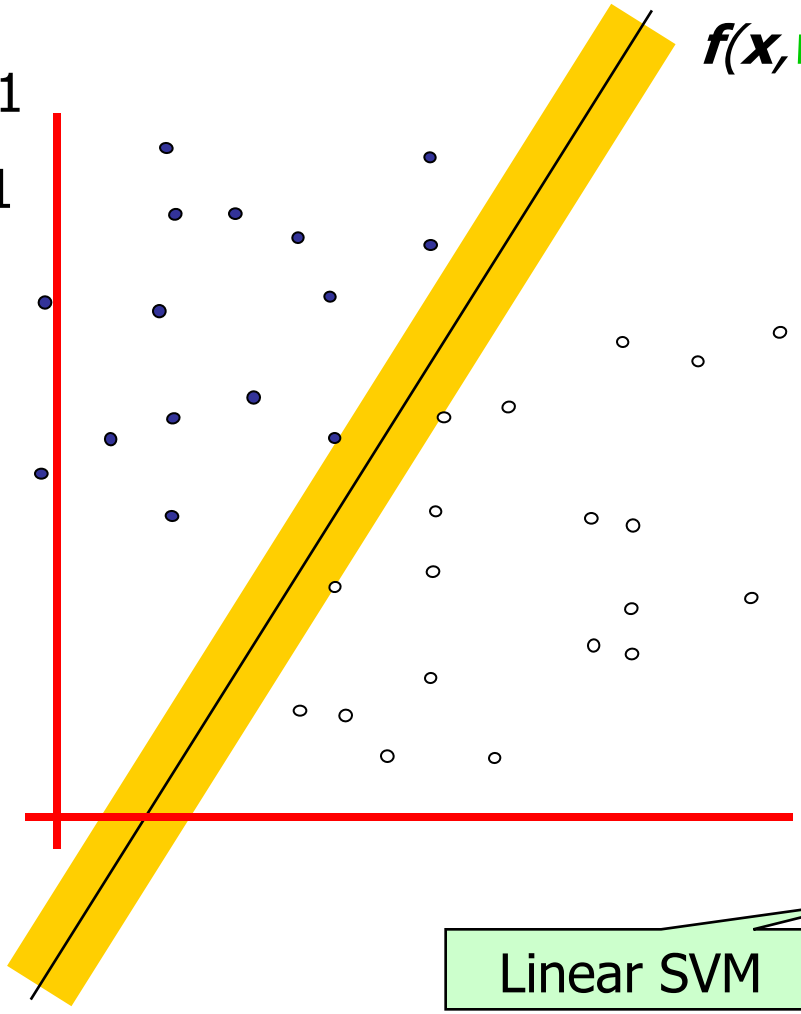
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

- denotes +1
- denotes -1

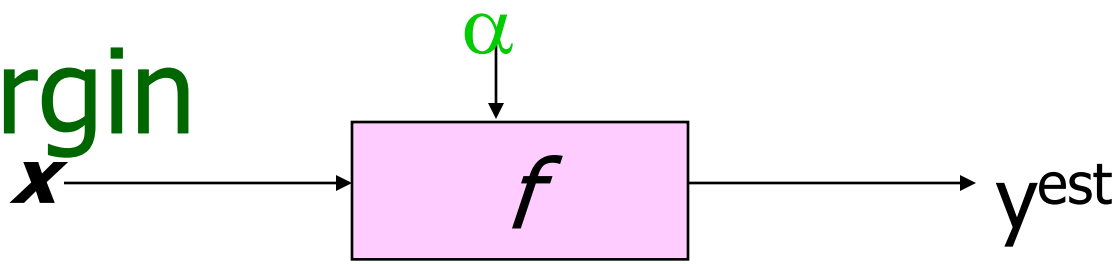


The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Linear SVM

Maximum Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

- denotes +1
- denotes -1

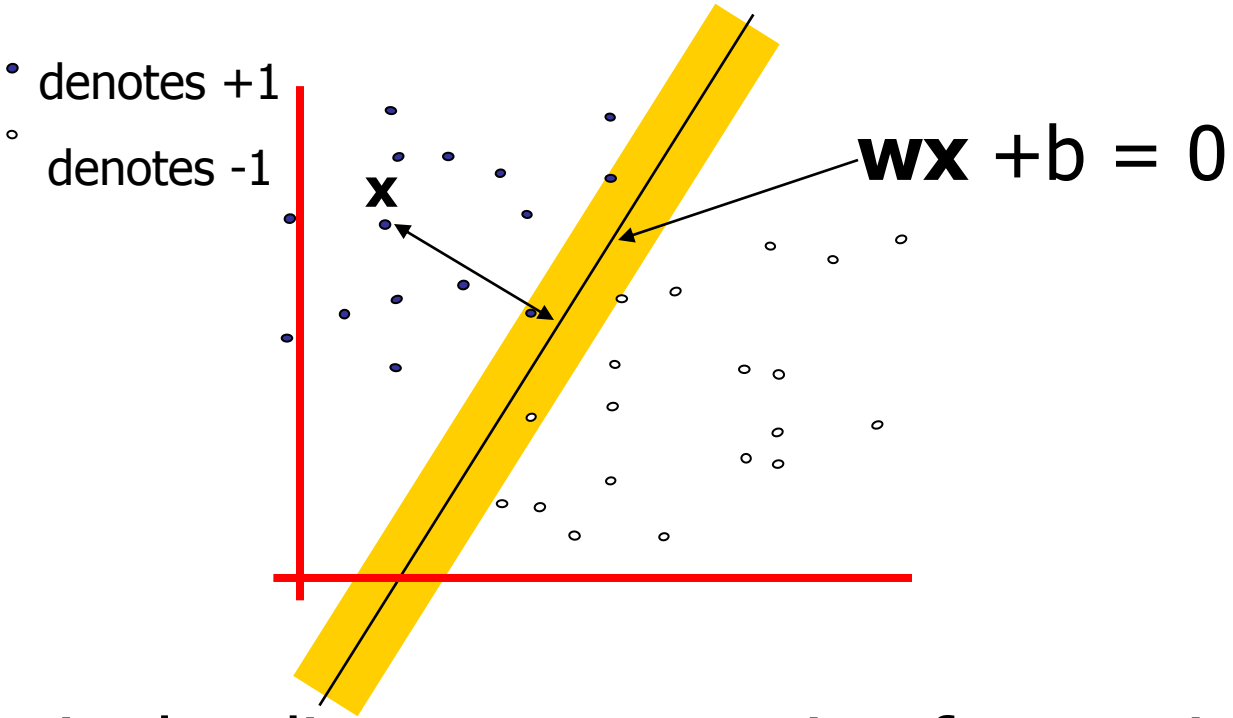
Support Vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Linear SVM

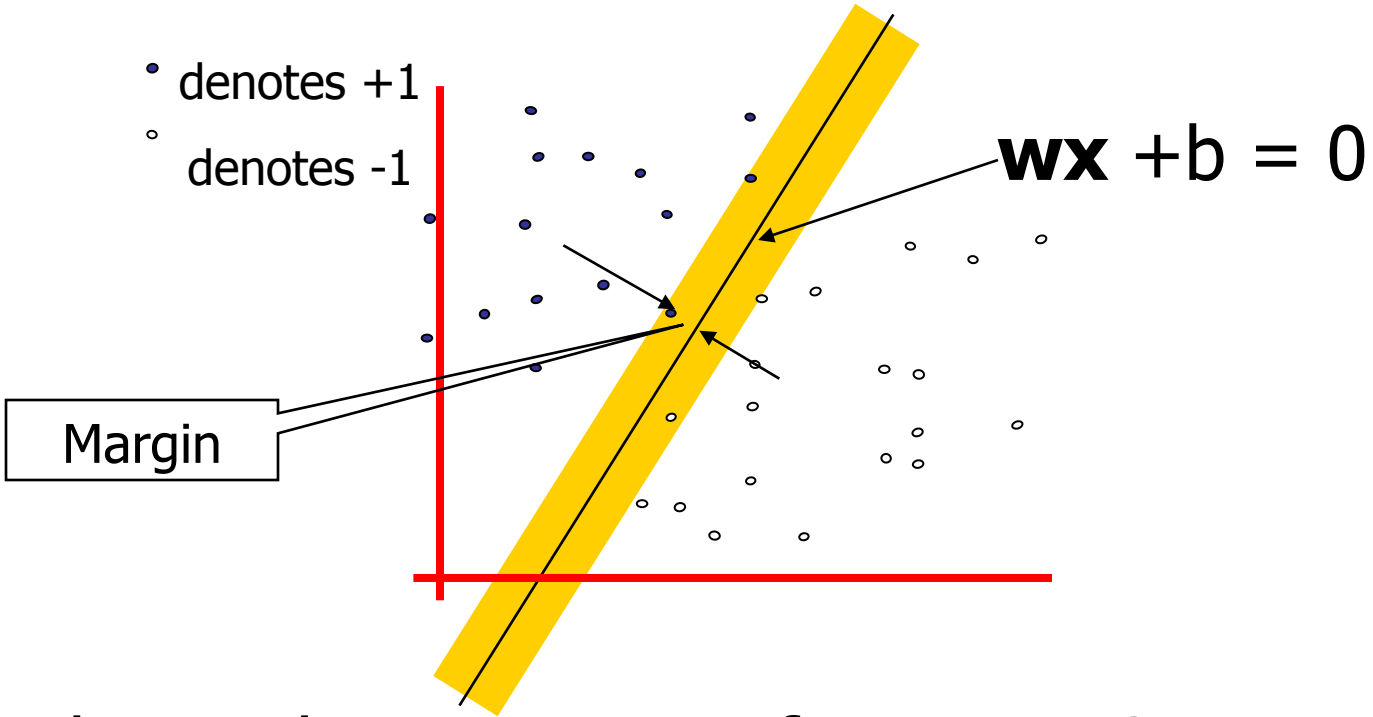
Estimate the Margin



- What is the distance expression for a point **x** to a line **wx**+b= 0?

$$d(\mathbf{x}) = \frac{\left|\mathbf{x}\cdot\mathbf{w}+b\right|}{\sqrt{\|\mathbf{w}\|_2^2}} = \frac{\left|\mathbf{x}\cdot\mathbf{w}+b\right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

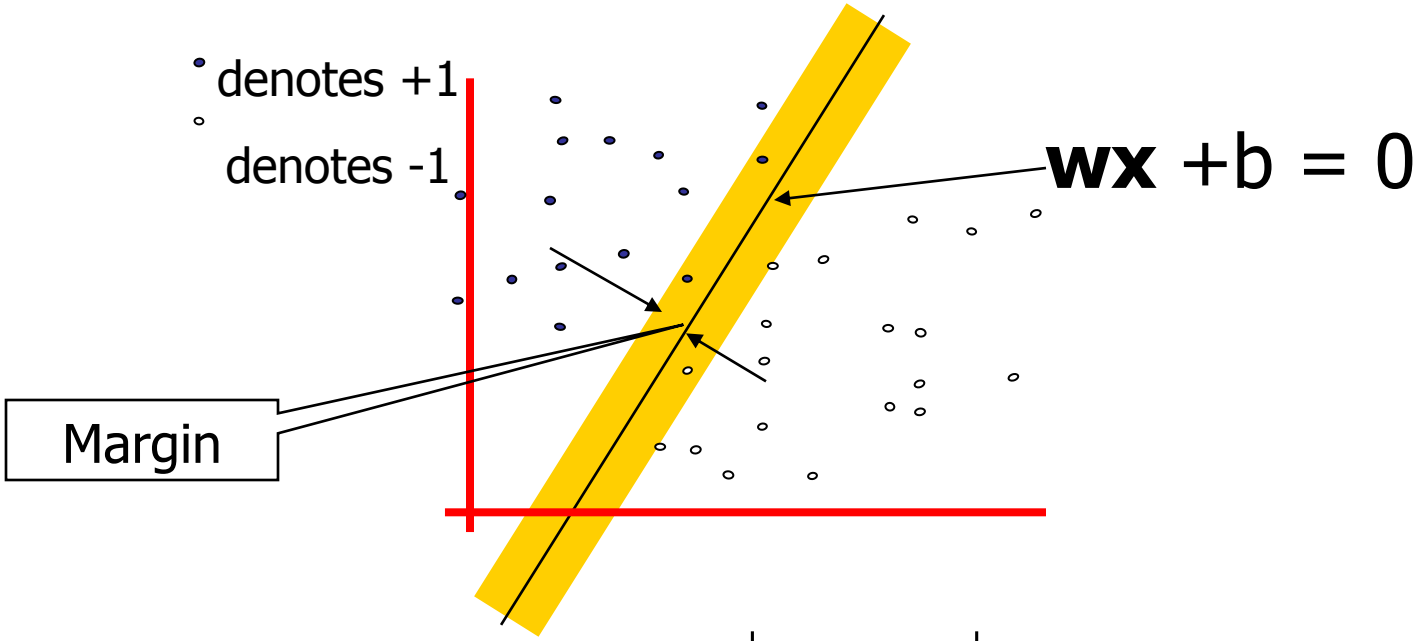
Estimate the Margin



- What is the expression for margin?

$$\text{margin} \equiv \min_{\mathbf{x}\in D} d(\mathbf{x}) = \min_{\mathbf{x}\in D} \frac{\left|\mathbf{x}\cdot\mathbf{w}+b\right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

Maximize Margin

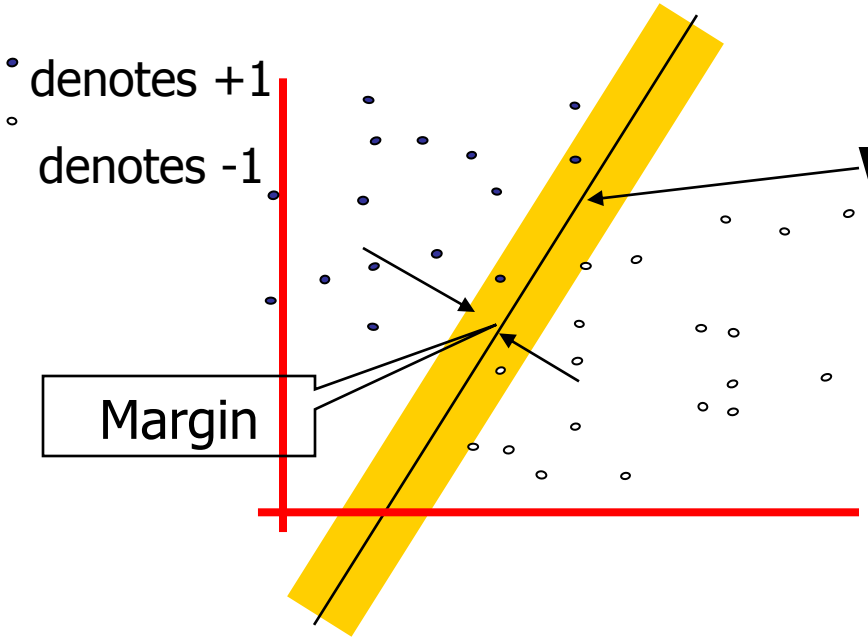


$$\underset{\mathbf{w},b}{\operatorname{argmax}} \min_{\mathbf{x}_i \in D} \frac{\left| b + \mathbf{x}_i \cdot \mathbf{w} \right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

$$\text{subject to } \forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) > 0$$

- Min-max problem → game problem

Maximize Margin



wx +b = 0

$$\underset{\mathbf{w},b}{\operatorname{argmax}} \min_{\mathbf{x}_i \in D} \frac{\left|b + \mathbf{x}_i \cdot \mathbf{w}\right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

subject to $\forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) \geq 0$

Strategy:

$$\forall \mathbf{x}_i \in D : \left|b + \mathbf{x}_i \cdot \mathbf{w}\right| \geq 1$$

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \sum_{i=1}^{d} w_i^2$$

subject to $\forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) \geq 1$

Maximum Margin Linear Classifier

$$\{\vec{w}^*, b^*\} = \underset{\vec{w},b}{\operatorname{argmin}} \sum_{k=1}^{d} w_k^2$$

subject to

$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1$$

$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1$$

....

$$y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1$$

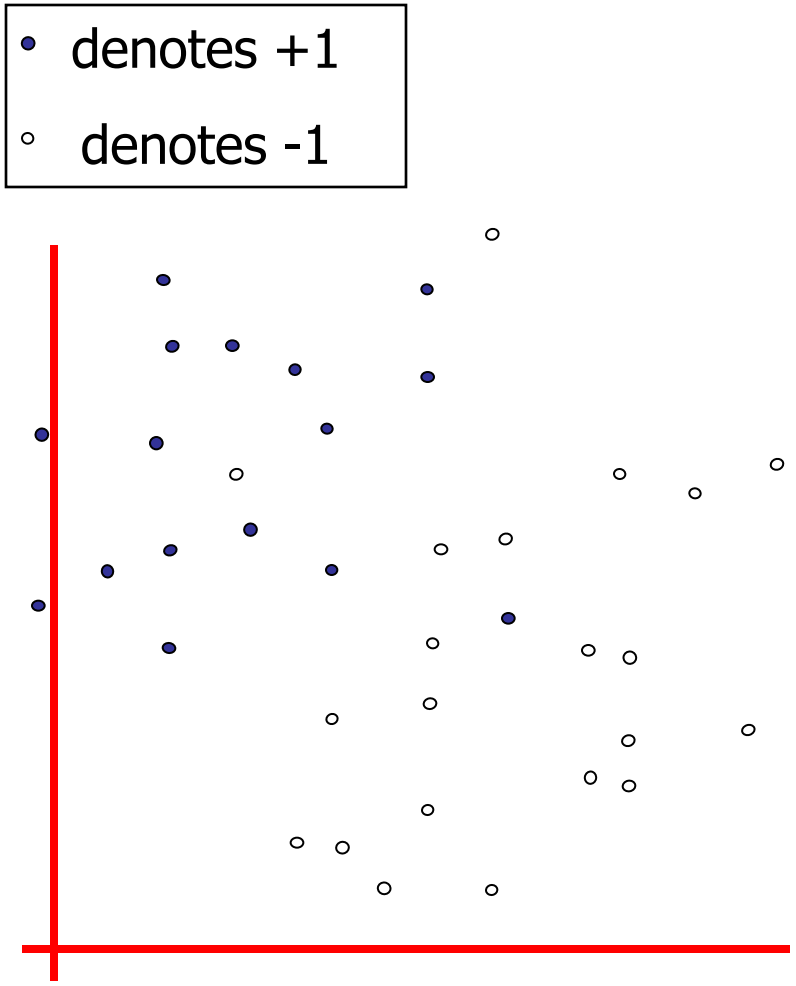
- How to solve it?

  Quadratic Programming

Uh-oh!

This is going to be a problem!

What should we do?



Support Vector Machine (SVM) for Noisy Data

$$\{\vec{w}^*, b^*\} = \min_{\vec{w},b,\vec{\varepsilon}} \sum_{i=1}^{\mathrm{d}} w_i^2 + c\sum_{j=1}^{N} \varepsilon_j$$

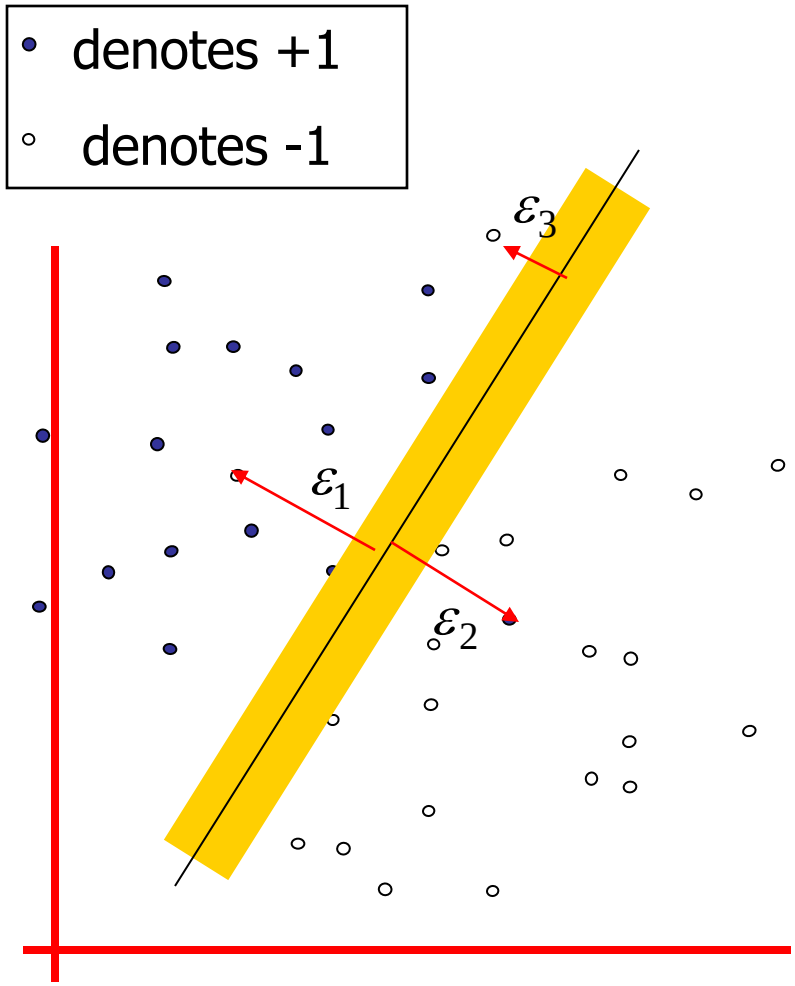
$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1-\varepsilon_1, \varepsilon_1 \geq 0$$

$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1-\varepsilon_2, \varepsilon_2 \geq 0$$

...

$$y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1-\varepsilon_N, \varepsilon_N \geq 0$$

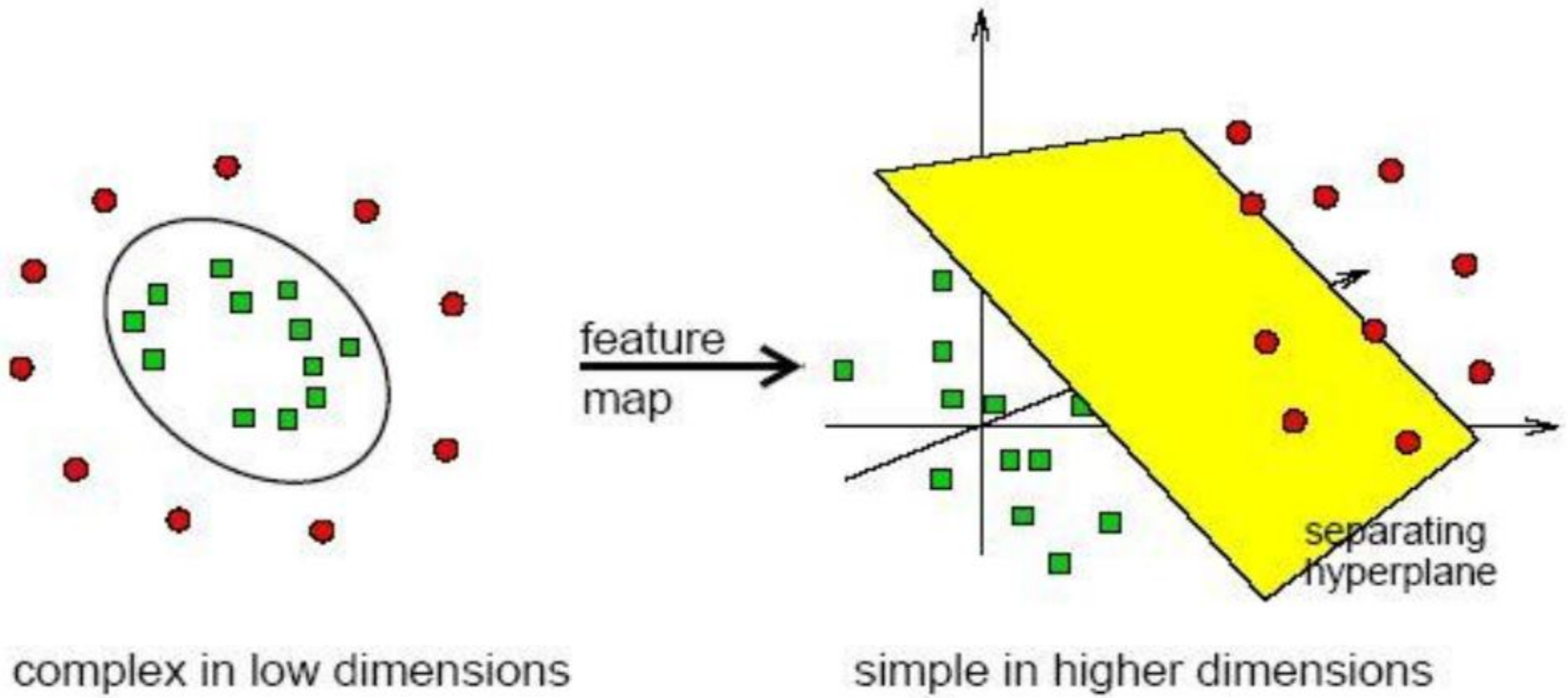
- Balance the trade off between margin and classification errors



Nonlinear SVM - Kernels

The concept of a kernel mapping function is very powerful. It allows SVM models to perform separations even with very complex boundaries such as shown below.

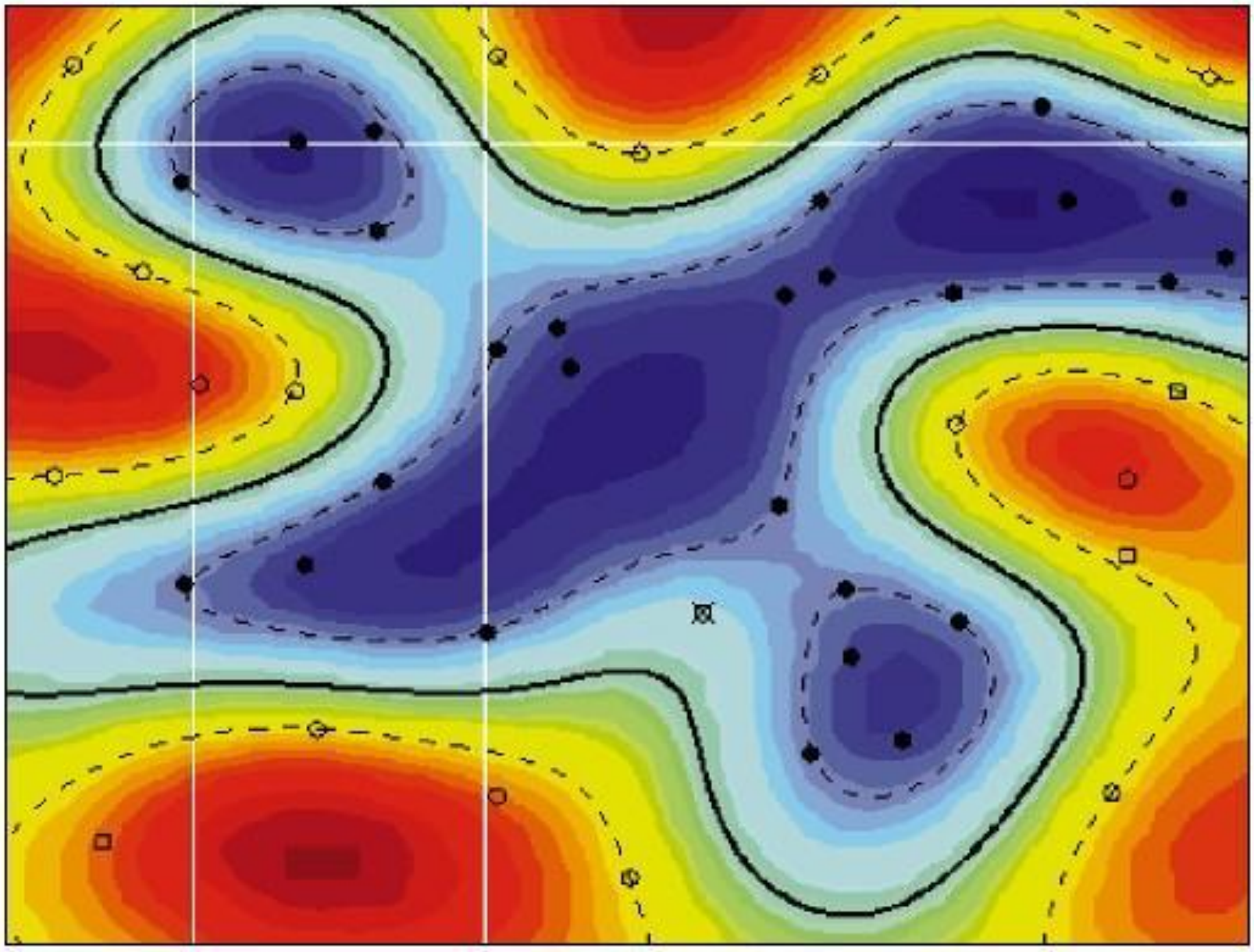
Separation may be easier in higher dimensions



kernel mapping

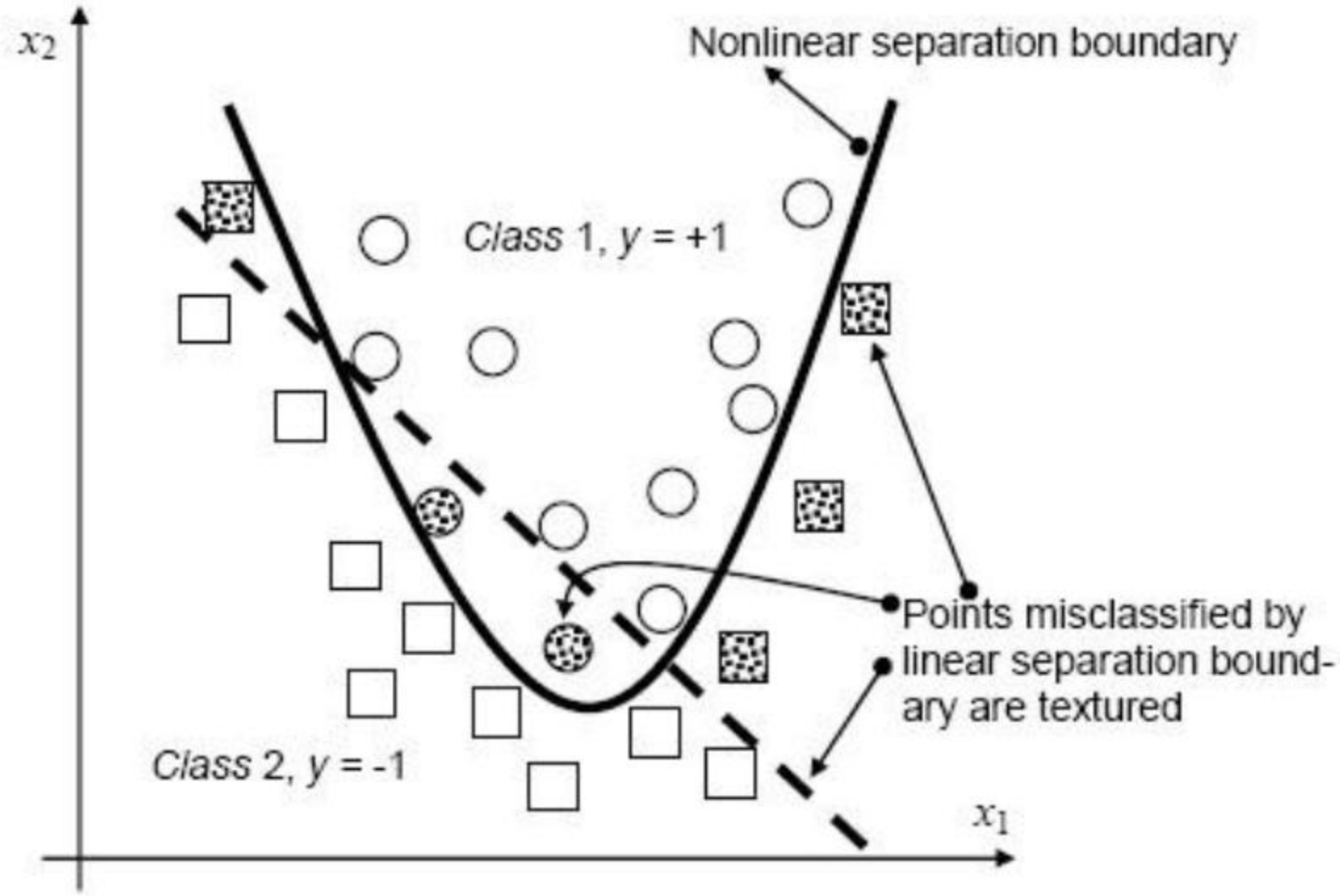
- **The Kernel Trick**
  - Many kernel mapping functions can be used – probably an infinite number. But a few kernel functions have been found to work well in for a wide variety of applications. The default and recommended kernel function is the Radial Basis Function (RBF).

kernel mapping



Kernel Examples

- **Polynomial**

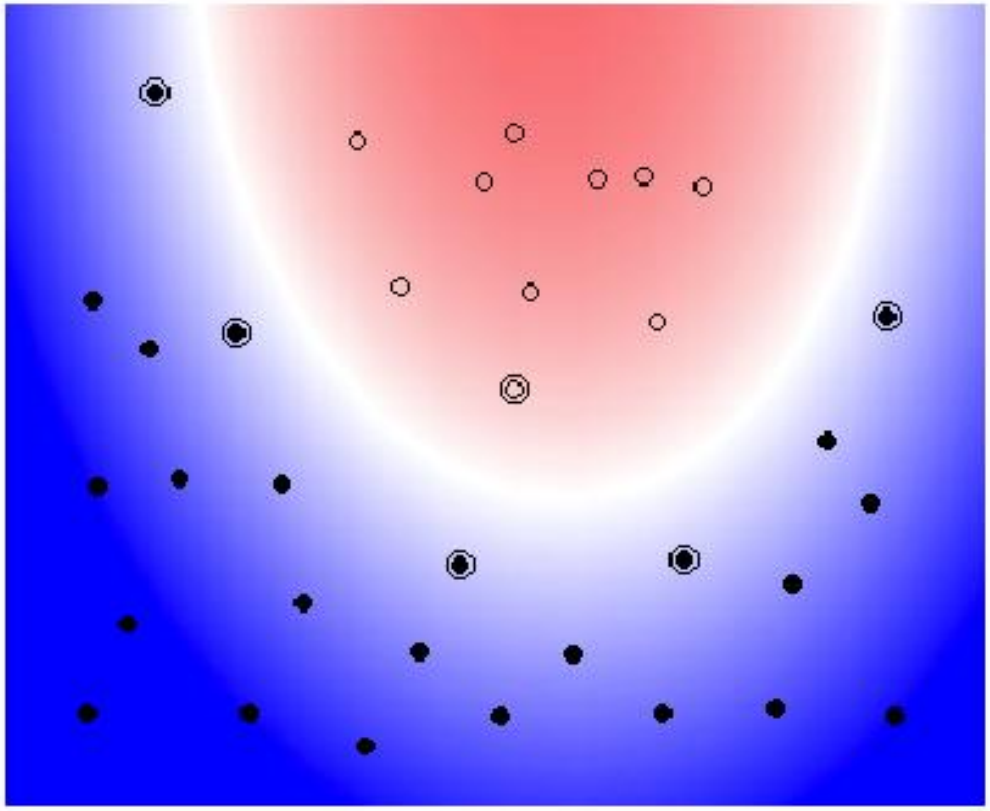


Kernel Examples

Example: SVM with Polynomial of Degree 2

Kernel: $K(\vec{x}_i,\vec{x}_j) = [\vec{x}_i \cdot \vec{x}_j + 1]^2$

plot by Bell SVM applet

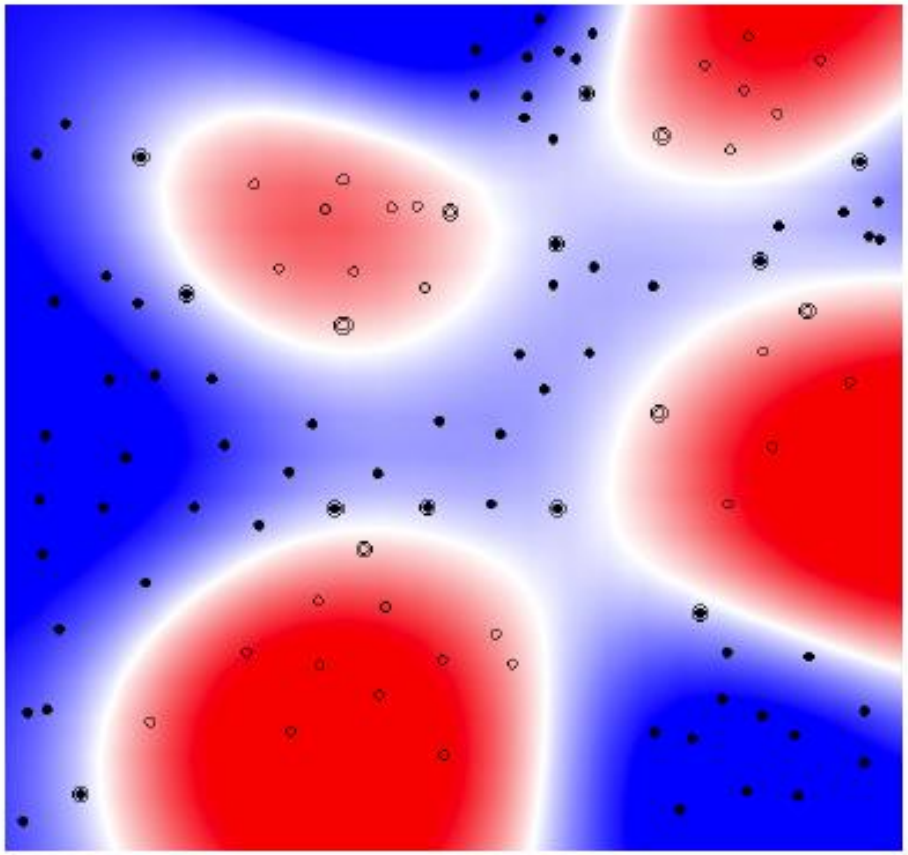


Nonlinear Kernel (II)

Example: SVM with RBF-Kernel

Kernel: $K(\vec{x}_i,\vec{x}_j) = \exp(-|\vec{x}_i - \vec{x}_j|^2/\sigma^2)$

plot by Bell SVM applet



SVM applications

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- SVM techniques have been extended to a number of tasks such as regression [Vapnik *et al.* '97], principal component analysis [Schölkopf *et al.* '99], etc.
- Most popular optimization algorithms for SVMs use *decomposition* to hill-climb over a subset of α_i's at a time, e.g. SMO [Platt '99] and [Joachims '99]
- Tuning SVMs remains a black art: selecting a specific kernel and parameters is usually done in a try-and-see manner.

Unsupervised Learning

Supervised learning vs. unsupervised learning

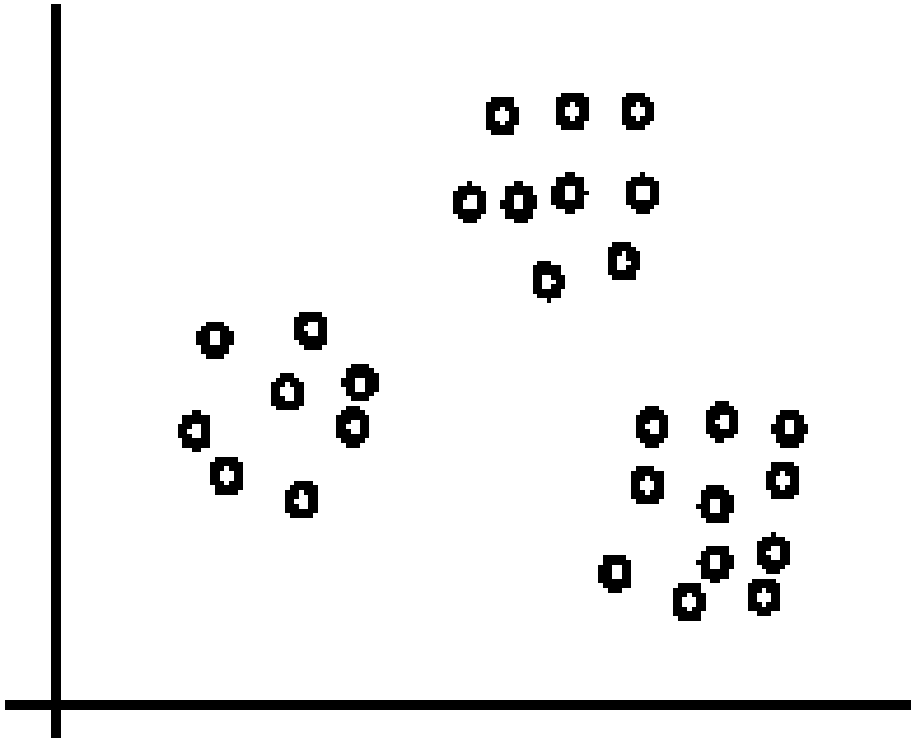
- Supervised learning: discover patterns in the data that relate data attributes with a target (class) attribute.
  - These patterns are then utilized to predict the values of the target attribute in future data instances.
- Unsupervised learning: The data have no target attribute.
  - We want to explore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

Clustering

- Clustering is a technique for finding similarity groups in data, called **clusters**. I.e.,
  - it groups data instances that are similar to (near) each other in one cluster and data instances that are very different (far away) from each other into different clusters.
- Clustering is often called an **unsupervised learning** task as no class values denoting an *a priori* grouping of the data instances are given, which is the case in supervised learning.
- Due to historical reasons, clustering is often considered synonymous with unsupervised learning.
  - In fact, association rule mining is also unsupervised
- This chapter focuses on clustering.

An illustration

- The data set has three natural groups of data points, i.e., 3 natural clusters.



What is clustering for?

- Let us see some real-life examples
- Example 1: groups people of similar sizes together to make "small", "medium" and "large"T-Shirts.
  - Tailor-made for each person: too expensive
  - One-size-fits-all: does not fit all.
- Example 2: In marketing, segment customers according to their similarities
  - To do targeted marketing.

What is clustering for? (cont...)

- Example 3: Given a collection of text documents, we want to organize them according to their content similarities,
  - To produce a topic hierarchy
- In fact, clustering is one of the most utilized data mining techniques.
  - It has a long history, and used in almost every field, e.g., medicine, psychology, botany, sociology, biology, archeology, marketing, insurance, libraries, etc.
  - In recent years, due to the rapid increase of online documents, text clustering becomes important.

Aspects of clustering

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

K-means clustering

- K-means is a partitional clustering algorithm
- Let the set of data points (or instances) D be

$$\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\},$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ir})$ is a vector in a real-valued space $X \subseteq R^r$, and r is the number of attributes (dimensions) in the data.

- The k-means algorithm partitions the given data into k clusters.
  - Each cluster has a cluster **center**, called **centroid**.
  - k is specified by the user

K-means algorithm

- Given *k*, the *k-means* algorithm works as follows:

  1)Randomly choose *k* data points (seeds) to be the initial centroids, cluster centers

  2)Assign each data point to the closest centroid

  3)Re-compute the centroids using the current cluster memberships.

  4)If a convergence criterion is not met, go to 2).

K-means algorithm – (cont ...)

Algorithm *k*-means(*k*, *D*)

1 Choose *k* data points as the initial centroids (cluster centers)
2 **repeat**
3 **for** each data point $\mathbf{x} \in D$ do
4 compute the distance from **x** to each centroid;
5 assign **x** to the closest centroid // a centroid represents a cluster
6 **endfor**
7 re-compute the centroids using the current cluster memberships
8 **until** the stopping criterion is met

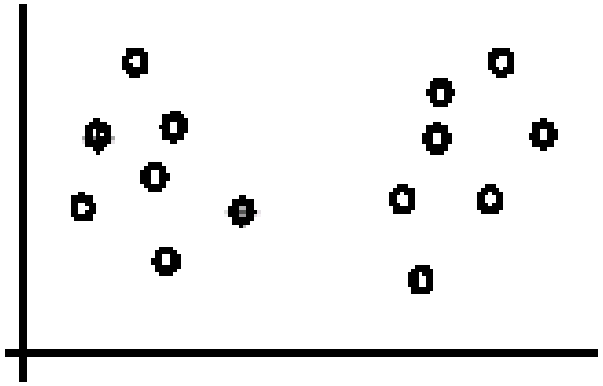
Stopping/convergence criterion

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),

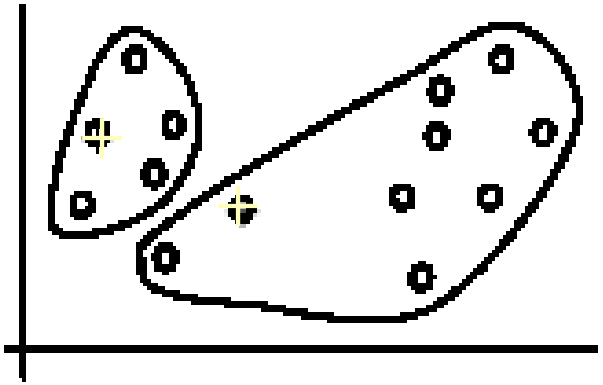
$$SSE = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2 \qquad (1)$$

- 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 $\mathbf{m}_j$.

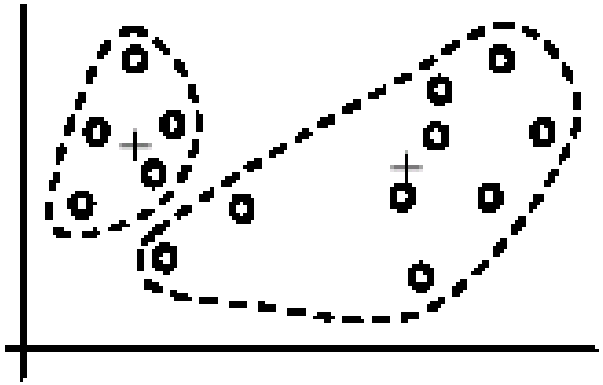
An example



(A). Random selection of k centers

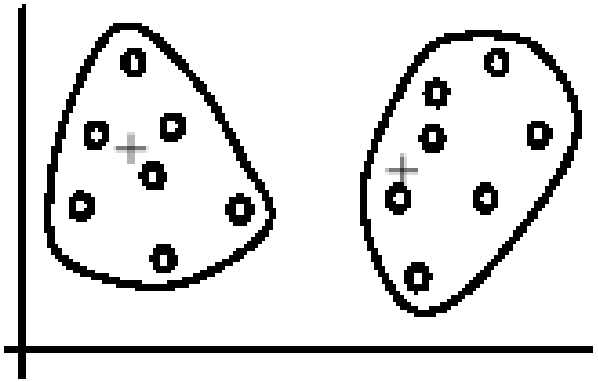


Iteration 1: (B). Cluster assignment

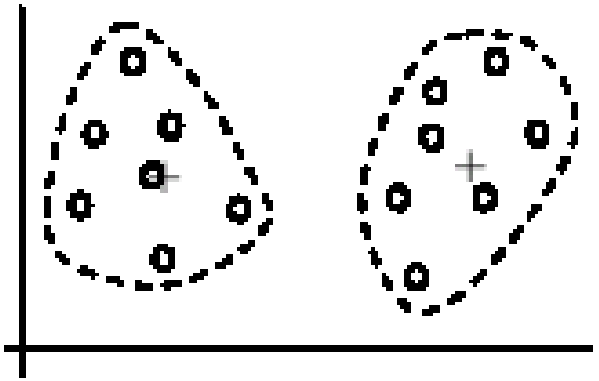


(C). Re-compute centroids

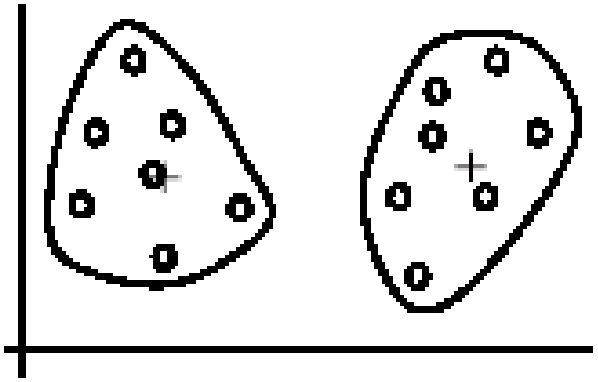
An example (cont …)



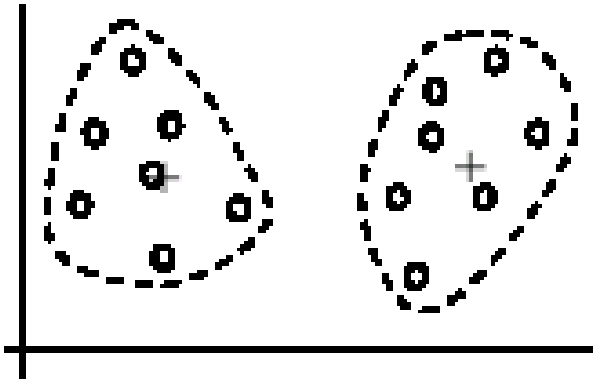
Iteration 2: (D). Cluster assignment



(E). Re-compute centroids



Iteration 3: (F). Cluster assignment



(G). Re-compute centroids

An example distance function

The *k*-means algorithm can be used for any application data set where the **mean** can be defined and computed. In the **Euclidean space**, the mean of a cluster is computed with:

$$\mathbf{m}_j = \frac{1}{|C_j|} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i \qquad (2)$$

where $|C_j|$ is the number of data points in cluster C_j. The distance from one data point $\mathbf{x}_i$ to a mean (centroid) $\mathbf{m}_j$ is computed with

$$dist(\mathbf{x}_i, \mathbf{m}_j) = \| \mathbf{x}_i - \mathbf{m}_j \| \qquad (3)$$

$$= \sqrt{(x_{i1} - m_{j1})^2 + (x_{i2} - m_{j2})^2 + \ldots + (x_{ir} - m_{jr})^2}$$

A disk version of *k*-means

- K-means can be implemented with data on disk
  - In each iteration, it scans the data once.
  - as the centroids can be computed incrementally
- It can be used to cluster large datasets that do not fit in main memory
- We need to control the number of iterations
  - In practice, a limited is set (< 50).
- Not the best method. There are other scale-up algorithms, e.g., BIRCH.

A disk version of k-means (cont ...)

Algorithm disk-k-means(k, D)

1 Choose k data points as the initial centriods $\mathbf{m}_j$, $j = 1, \ldots, k$;
2 **repeat**
3 initialize $\mathbf{s}_j = \mathbf{0}$, $j = 1, \ldots, k$; // $\mathbf{0}$ is a vector with all 0's
4 initialize $n_j = 0$, $j = 1, \ldots, k$; // n_j is the number points in cluster j
5 **for** each data point $\mathbf{x} \in D$ **do**
6 $j = \arg\min_j dist(\mathbf{x}, \mathbf{m}_j)$;
7 assign $\mathbf{x}$ to the cluster j;
8 $\mathbf{s}_j = \mathbf{s}_j + \mathbf{x}$;
9 $n_j = n_j + 1$;
10 **endfor**
11 $\mathbf{m}_i = \mathbf{s}_j / n_j$, $i = 1, \ldots, k$;
12 **until** the stopping criterion is met

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,

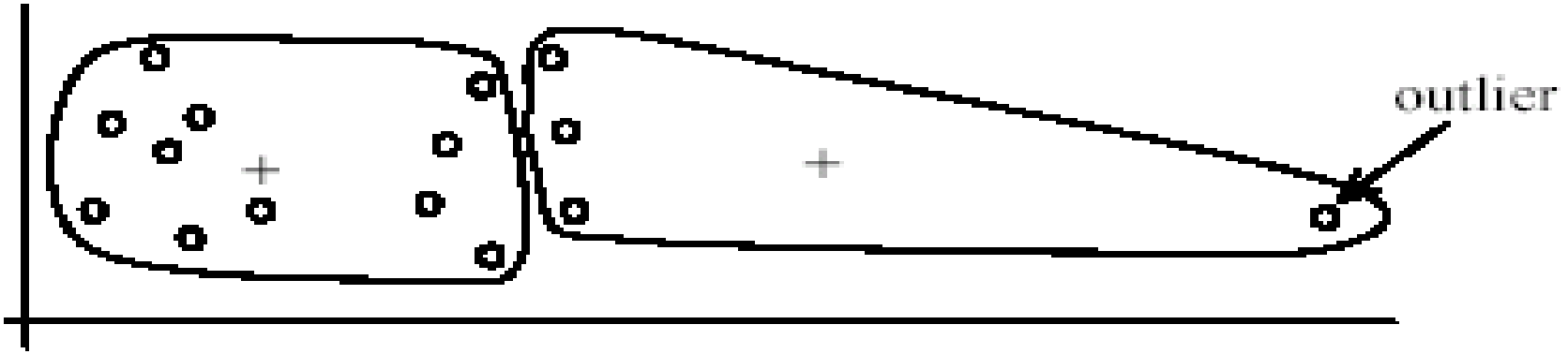
    k is the number of clusters, and

    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.

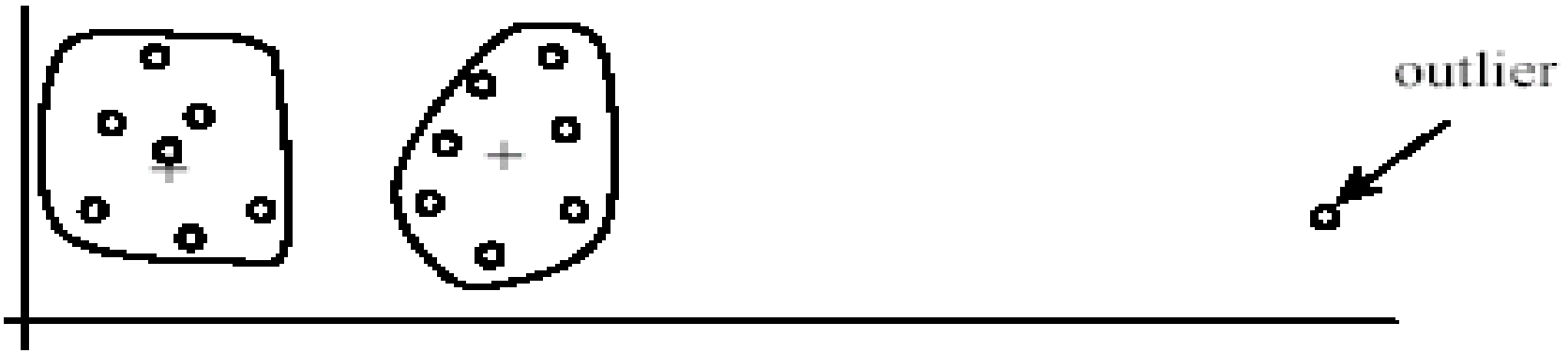
Weaknesses of k-means

- 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



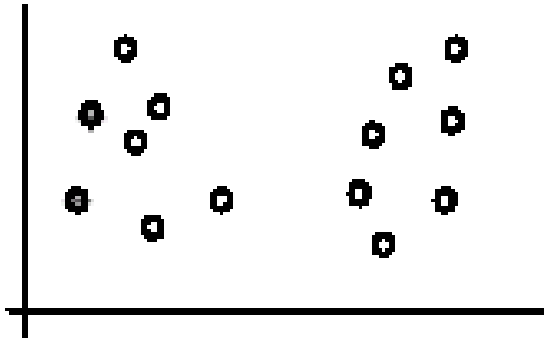
(B): Ideal clusters

Weaknesses of k-means: To deal with outliers

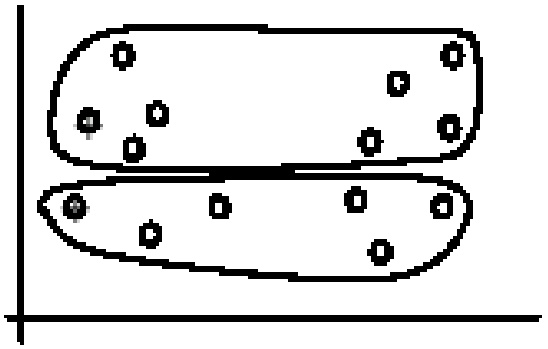
- One method is to remove some data points in the clustering process that are much further away from the centroids than other data points.
  - To be safe, we may want to monitor these possible outliers over a few iterations and then decide to remove them.
- Another method is to perform random sampling. 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

- The algorithm is sensitive to initial seeds.

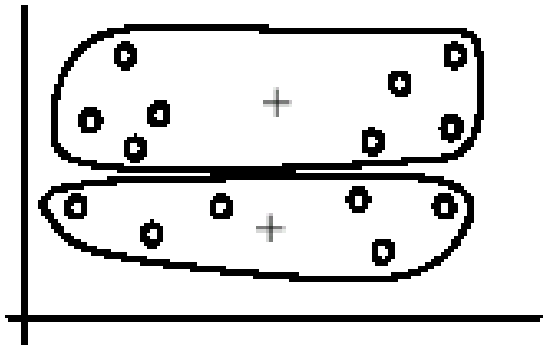
Weaknesses of k-means (cont ...)



(A). Random selection of seeds (centroids)



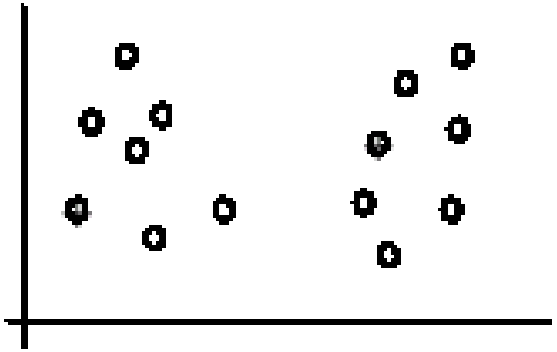
(B). Iteration 1



(C). Iteration 2

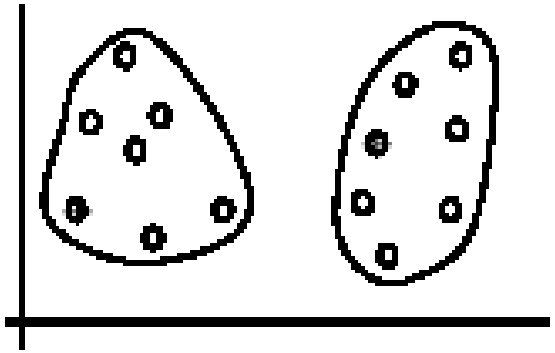
Weaknesses of k-means (cont ...)

- If we use different seeds: good results

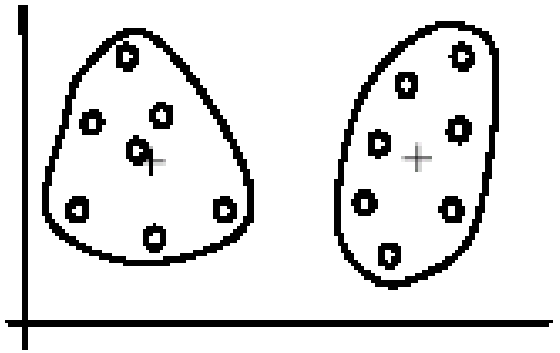


There are some methods to help choose good seeds

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



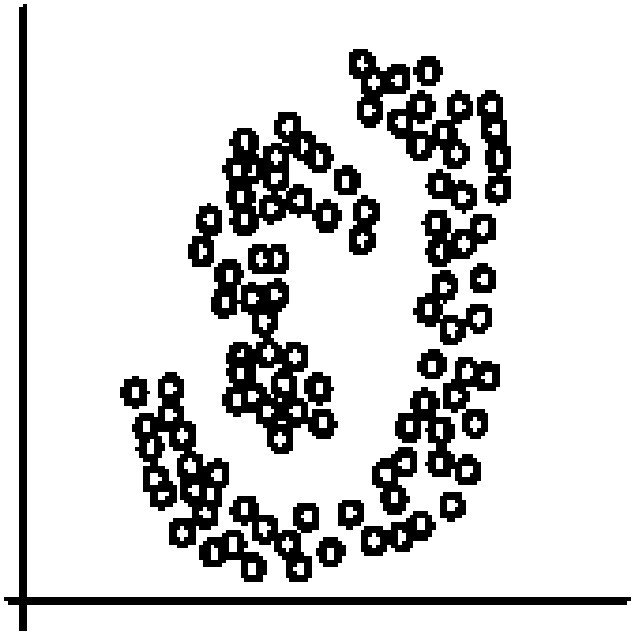
(B). Iteration 1



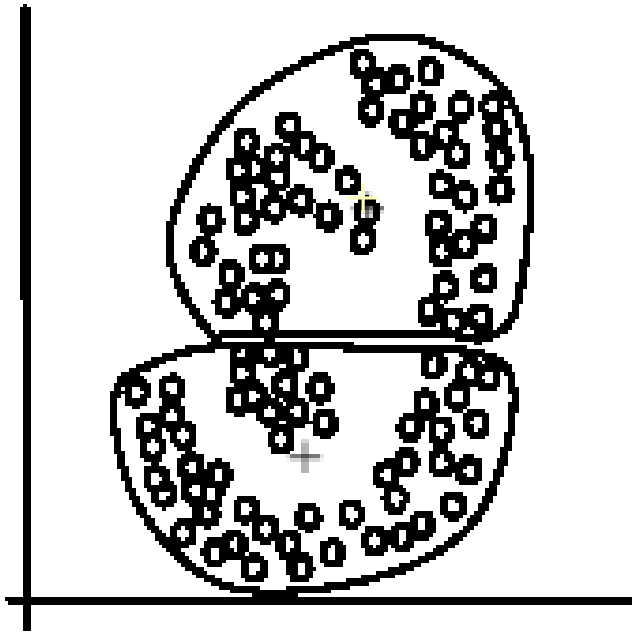
(C). Iteration 2

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, $D_1, D_2, \ldots, D_k$.

Evaluation measures: Entropy

Entropy: For each cluster, we can measure its entropy as follows:

$$entropy(D_i) = -\sum_{j=1}^{k} \Pr_i(c_j) \log_2 \Pr_i(c_j). \quad (29)$$

where $\Pr_i(c_j)$ is the proportion of class c_j data points in cluster i or D_i. The total entropy of the whole clustering (which considers all clusters) is

$$entropy_{total}(D) = \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times entropy(D_i) \quad (30)$$

Evaluation measures: purity

Purity: This again measures the extent that a cluster contains only one class of data. The purity of each cluster is computed with

$$purity(D_i) = \max_j(\Pr_i(c_j)) \tag{31}$$

The total purity of the whole clustering (considering all clusters) is

$$purity_{total}(D) = \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times purity(D_i) \tag{32}$$

An example

Example 14: Assume we have a text collection D of 900 documents from three topics (or three classes), Science, Sports, and Politics. Each class has 300 documents. Each document in D is labeled with one of the topics (classes). We use this collection to perform clustering to find three clusters. Note that class/topic labels are not used in clustering. After clustering, we want to measure the effectiveness of the clustering algorithm.

Cluster	Science	Sports	Politics		Entropy	Purity
1	250	20	10		0.589	0.893
2	20	180	80		1.198	0.643
3	30	100	210		1.257	0.617
Total	300	300	300		1.031	0.711

A remark about ground truth evaluation

- Commonly used to compare different clustering algorithms.
- A real-life data set for clustering has no class labels.
  - Thus although an algorithm may perform very well on some labeled data sets, no guarantee that it will perform well on the actual application data at hand.
- The fact that it performs well on some label data sets does give us some confidence of the quality of the algorithm.
- This evaluation method is said to be based on **external data** or information.

Evaluation based on internal information

- **Intra-cluster cohesion** (compactness):
  - Cohesion measures how near the data points in a cluster are to the cluster centroid.
  - Sum of squared error (SSE) is a commonly used measure.
- **Inter-cluster separation** (isolation):
  - Separation means that different cluster centroids should be far away from one another.
- In most applications, expert judgments are still the key.

Indirect evaluation

- In some applications, clustering is not the primary task, but used to help perform another task.
- We can use the performance on the primary task to compare clustering methods.
- For instance, in an application, the primary task is to provide recommendations on book purchasing to online shoppers.
  - If we can cluster books according to their features, we might be able to provide better recommendations.
  - We can evaluate different clustering algorithms based on how well they help with the recommendation task.
  - Here, we assume that the recommendation can be reliably evaluated.

Chapter Summary

- learning is very important for agents to improve their decision-making process
  - unknown environments, changes, time constraints
- most methods rely on inductive learning
  - a function is approximated from sample input-output pairs
- decision trees are useful for learning deterministic Boolean functions
- neural networks consist of simple interconnected computational elements
- multi-layer feed-forward networks can learn any function
  - provided they have enough units and time to learn