*INF5390 – Kunstig intelligens*

# Learning from Examples

Roar Fjellheim

# Outline

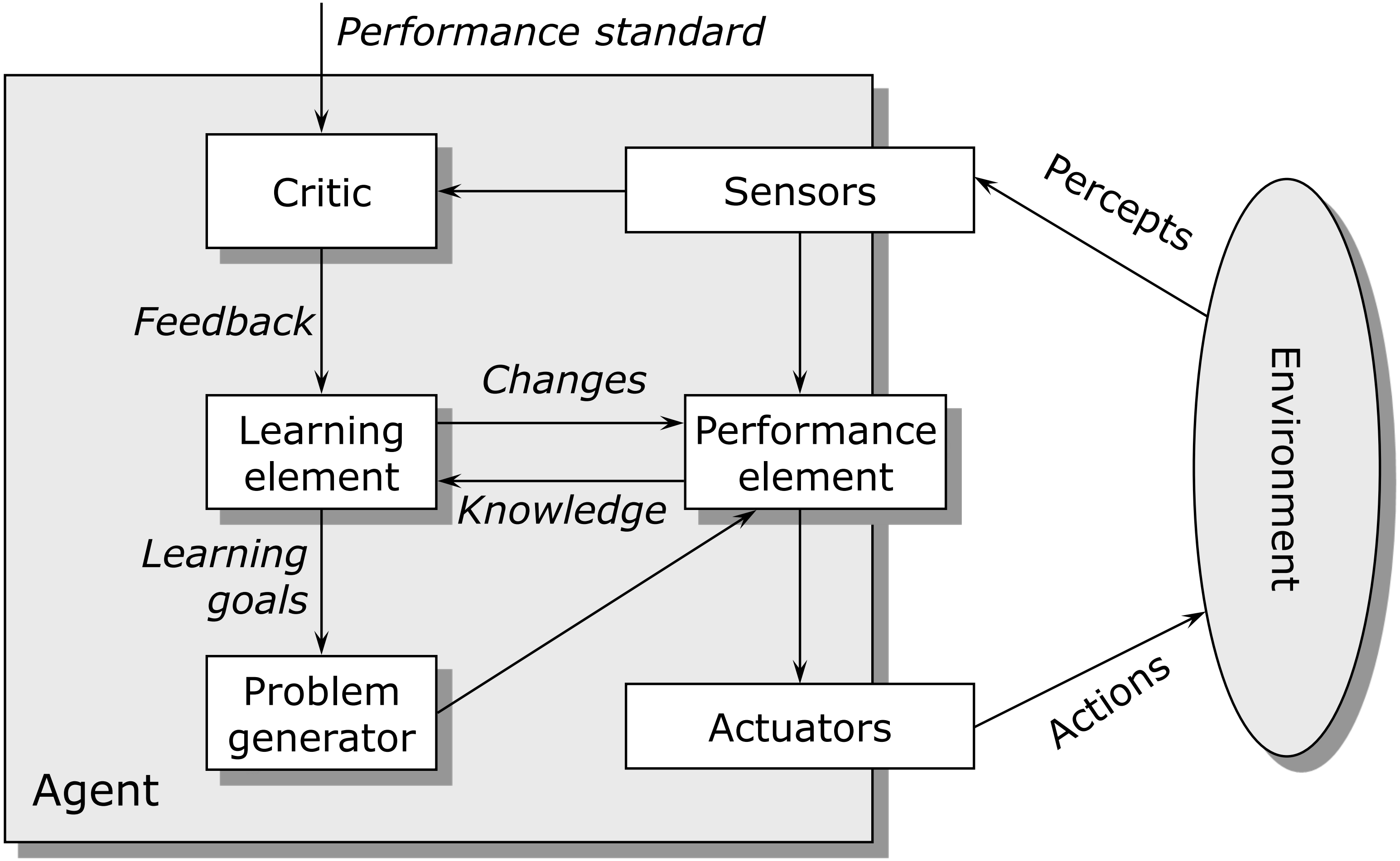
- General model
- Types of learning
- Learning decision trees
- Neural networks
- Perceptrons
- Summary

AIMA Chapter 18: Learning From Examples

# Why should agents learn?

- Agents in previous lectures have assumed "built-in" knowledge, provided by designers
- In order to handle incomplete knowledge and changing knowledge requirements, agents must *learn*
- Learning is a way of achieving agent *autonomy* and the ability to *improve performance* over time
- The field in AI that deals with learning is called *machine learning*, and is very active

# General model of learning agents



# Elements of the general model

- Performance element
  - √ Carries out the task of the agent, i.e. processes percepts and decides on actions
- Learning element
  - √ Proposes improvements of the performance element, based on previous knowledge and feedback
- Critic
  - √ Evaluates performance element by comparing results of its actions with imposed performance standards
- Problem generator
  - √ Proposes exploratory actions to increase knowledge

# Aspects of the learning element

- Which *components* of the performance element are to be improved
  - Which parts of the agent's knowledge base is targeted
- What *feedback* is available
  - Supervised, unsupervised or reinforcement learning differ in type of feedback agent receives
- What *representation* is used for the components
  - E.g. logic sentences, belief networks, utility functions, etc.
- What *prior information* (*knowledge*) is available

# Performance element components

- Possible components that can be improved
  - Direct mapping from states to actions
  - Means to infer world properties from percept sequences
  - Information about how the world evolves
  - Information about the results of possible actions
  - Utility information about the desirability of world states
  - Desirability of specific actions in specific states
  - Goals describing states that maximize utility
- In each case, learning can be sees as learning an unknown *function* $y = f(x)$

# Hypothesis space H

- H: the set of hypothesis functions *h* to be considered in searching for *f(x)*
- *Consistent* hypothesis: Fits with all data
  - If several consistent hypotheses – choose simplest one! (Occam's razor)
- *Realizability* of learning problem:
  - *Realizable* if H contains the "true" function
  - *Unrealizable* if not
  - We do normally know what the true function is
- Why not choose H as large as possible?
  - May be very inefficient in learning and in applying

# Types of learning - Knowledge

- *Inductive* learning
  - Given a collection of *examples* ($x$, $f$($x$))
  - Return a function $h$ that approximates $f$
  - Does not rely on prior knowledge ("just data")
- *Deductive* (or analytical) learning
  - Going from known general $f$ to a new $f'$ that is logically entailed
  - Based on prior knowledge ("data+knowledge")
  - Resemble more human learning

# Types of learning - Feedback

- *Unsupervised* learning
  - Agent learns patterns in data even though no feedback is given, e.g. via clustering
- *Reinforcement* learning
  - Agent gets reward or punishment at the end, but is not told which particular action led to the result
- *Supervised* learning
  - Agent receives learning examples and is explicitly told what the correct answer is for each case
- Mixed modes, e.g. *semi-supervised* learning
  - Correct answers for some but not all examples

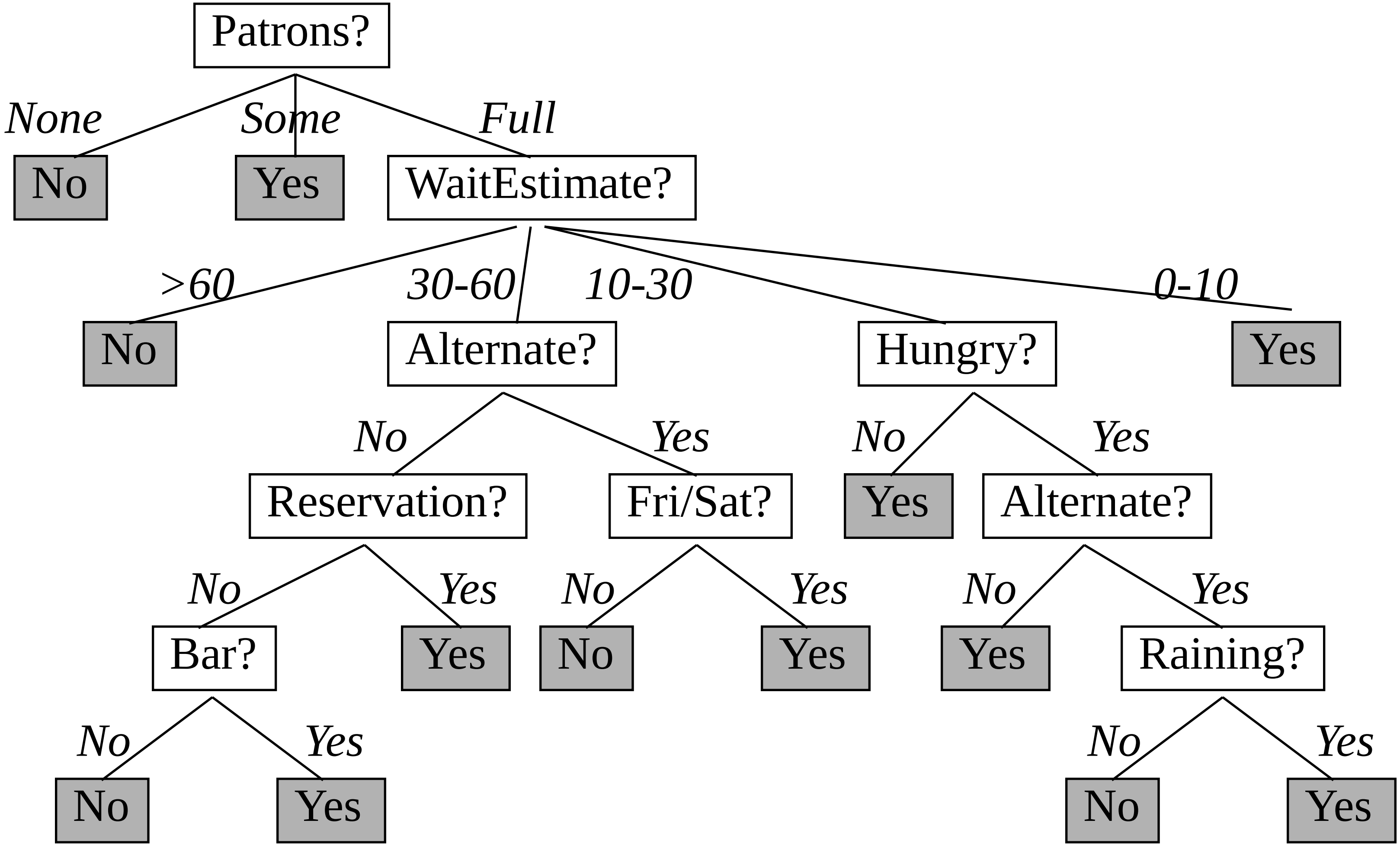
# Learning decision trees

- A *decision situation* can be described by
  - A number of *attributes*, each with a set of possible values
  - A *decision* which may be Boolean (yes/no) or multivalued
- A *decision tree* is a tree structure where
  - Each internal node represents a *test* of the value of an attribute, with one branch for each possible attribute value
  - Each leaf node represents the value of the *decision* if that node is reached
- *Decision tree learning* is one of simplest and most successful forms of machine learning
- An example of *inductive* and *supervised* learning

# Example: Wait for restaurant table

- Goal predicate: *WillWait* (for restaurant table)
- Domain attributes
  - Alternate (other restaurants nearby)
  - Bar (to wait in)
  - Fri/Sat (day of week)
  - Hungry (yes/no)
  - Patrons (none, some, full)
  - Price (range)
  - Raining (outside)
  - Reservation (made before)
  - Type (French, Italian, ..)
  - WaitEstimate (minutes)

# One decision tree for the example



# Expressiveness of decision trees

- The tree is equivalent to a conjunction of implications

$$\forall r\, Patrons(r, Full) \wedge WaitEstimate(r, 10-30) \wedge Hungry(r, No) \Rightarrow WillWait(r)$$

- Cannot represent tests on two or more objects, restricted to testing attributes of one object
- Fully expressive as propositional language, e.g. any Boolean function can be written as a decision tree
- For some functions, exponentially large decision trees are required
- E.g. decision trees are good for some functions and bad for others

# Inducing decision trees from examples

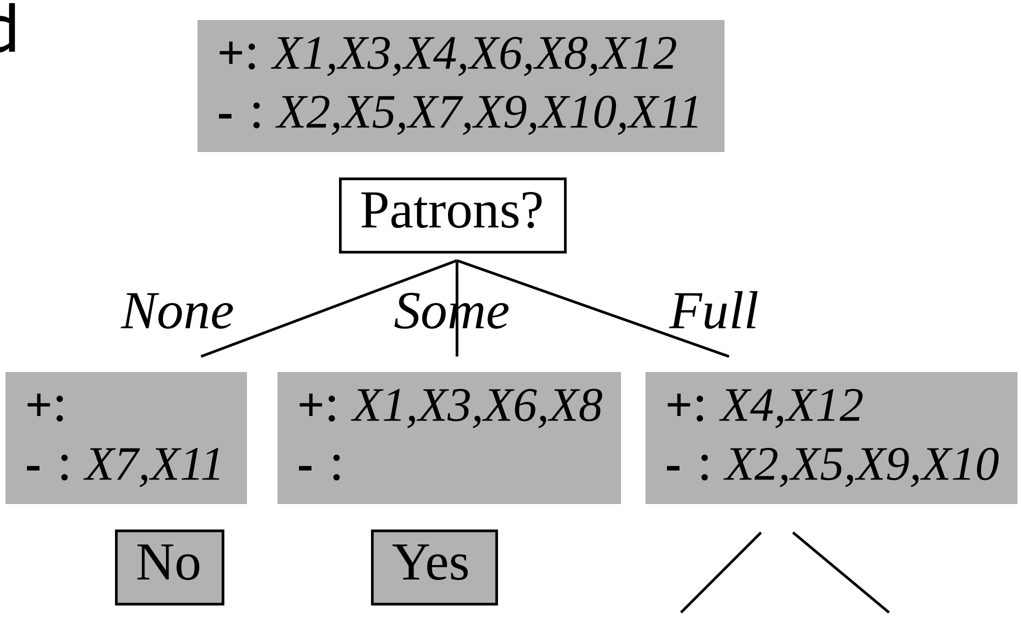
- Terminology
  - *Example* - Specific values for all attributes, plus goal predicate
  - *Classification* - Value of goal predicate of the example
  - *Positive/negative example* - Goal predicate is true/false
  - *Training set* - Complete set of examples
- The task of inducing a decision tree from a training set is to *find the simplest tree that agrees with the examples*
- The resulting tree should be more *compact* and *general* than the training set itself

# A training set for the restaurant example

| Example | Attributes | | | | | | | | | | Will wait |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Alt* | *Bar* | *Fri* | *Hun* | *Pat* | *Price* | *Rain* | *Res* | *Type* | *Est* | |
| *X1* | *Yes* | *No* | *No* | *Yes* | *Some* | *$$$* | *No* | *Yes* | *French* | *0-10* | *Yes* |
| *X2* | *Yes* | *No* | *No* | *Yes* | *Full* | *$* | *No* | *No* | *Thai* | *30-60* | *No* |
| *X3* | *No* | *Yes* | *No* | *No* | *Some* | *$* | *No* | *No* | *Burger* | *0-10* | *Yes* |
| *X4* | *Yes* | *No* | *Yes* | *Yes* | *Full* | *$* | *No* | *No* | *Thai* | *10-30* | *Yes* |
| *X5* | | | | | | | | | | | |
| *X6* | | | | | | | | | | | |
| *X7* | | | | | | | | | | | |
| *X8* | | | | | | ETC. | | | | | |
| *X9* | | | | | | | | | | | |
| *X10* | | | | | | | | | | | |
| *X11* | | | | | | | | | | | |
| *X12* | | | | | | | | | | | |

# General idea of induction algorithm

- Test the most important attribute first, i.e. the one that makes the most difference to the classification
- *Patrons?* is a good choice for the first attribute, because it allows early decisions
- Apply same principle recursively

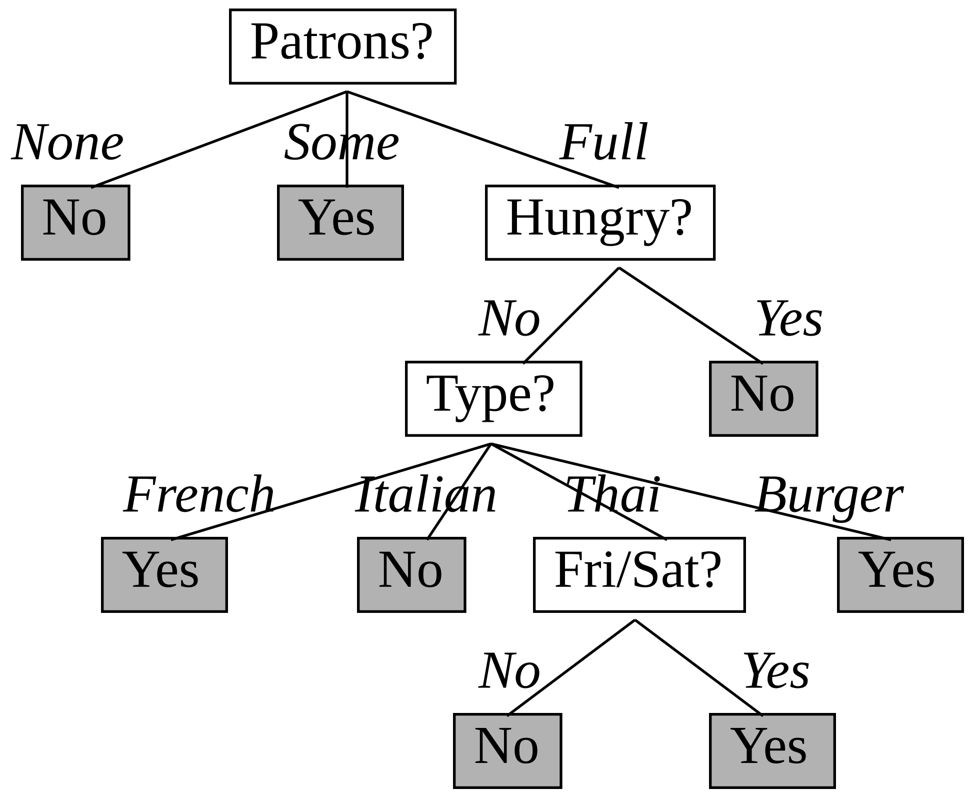


# Recursive step of induction algorithm

- The attribute test splits the tree into smaller decision trees, with fewer examples and one attribute less
- Four cases to consider for the smaller trees
  - √ If some positive and some negative examples, choose best attribute to split them
  - √ If examples are all positive (negative), answer *Yes* (*No*)
  - √ If no examples left, return a default value (no example observed for this case)
  - √ If no attributes left, but both positive and negative examples: Problem! (same description, different classifications - *noise*)

# Induced tree for the example set

- The induced tree is *simpler* than the original "manual" tree
- It captures some *regularities* that the original creator was unaware of

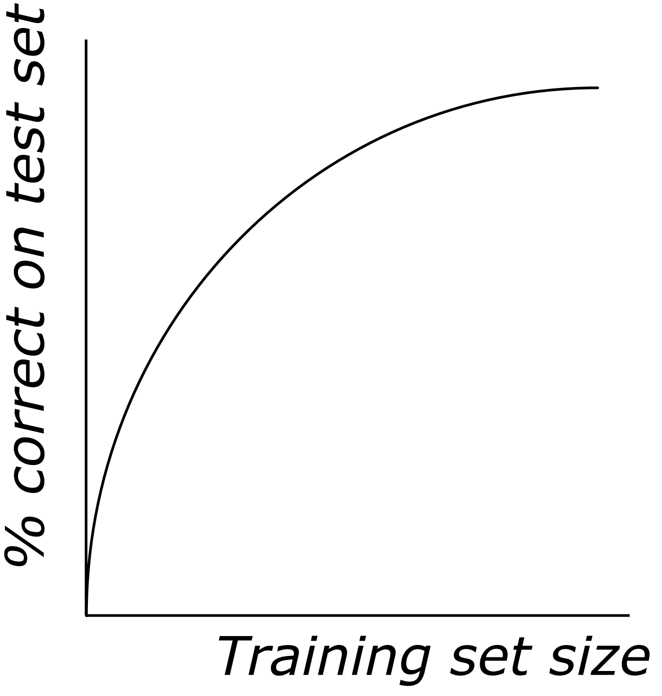


# Broaden applicability of decision trees

- Missing data
  - How to handle training samples with partially missing attribute values
- Multi/many-valued attributes
  - How to treat attributes with many possible values
- Continuous or integer-valued input attributes
  - How to branch the decision tree when attribute has a continuous value range
- Continuous-valued output attributes
  - Requires *regression tree* rather than a decision tree, i.e. output value is a linear function of input variables rather than a point value

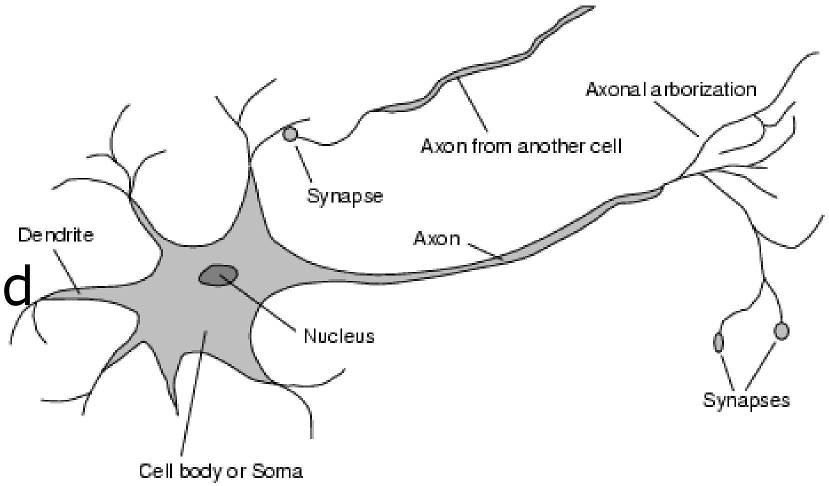
# Assessing learning performance

- Collect large set of examples
- Divide into two disjoint sets, *training set* and *test set*
- Use learning algorithm on training set to generate hypothesis *h*
- Measure percentage of examples in test set that are correctly classified by *h*
- Repeat steps above for differently sized training sets



# Neural networks in AI

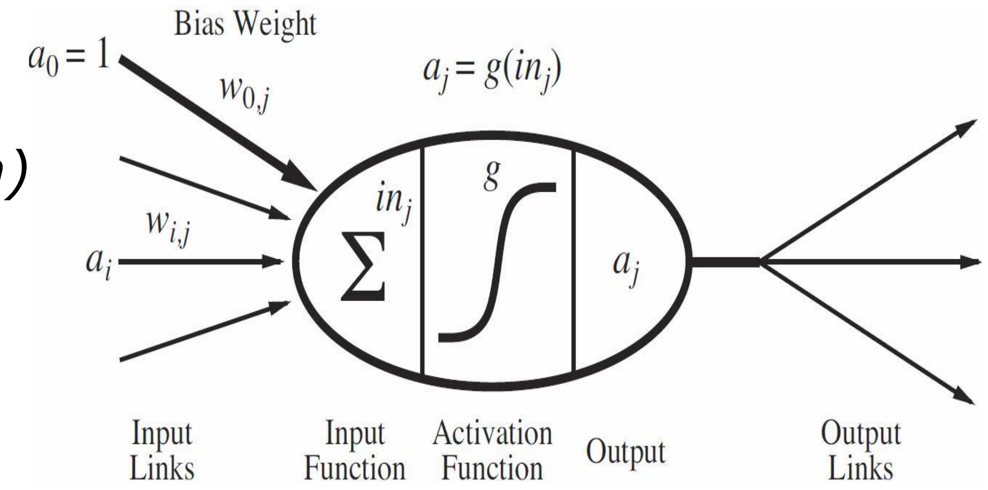
- The human brain is a huge network of *neurons*
  - A neuron is a basic processing unit that collects, processes and disseminates electrical signals




- Early AI tried to imitate the brain by building *artificial neural networks* (ANN)
  - Met with theoretical limits and ”disappeared”
- In the 1980-90’es, interest in ANNs resurfaced
  - New theoretical development
  - Massive industrial interest&applications

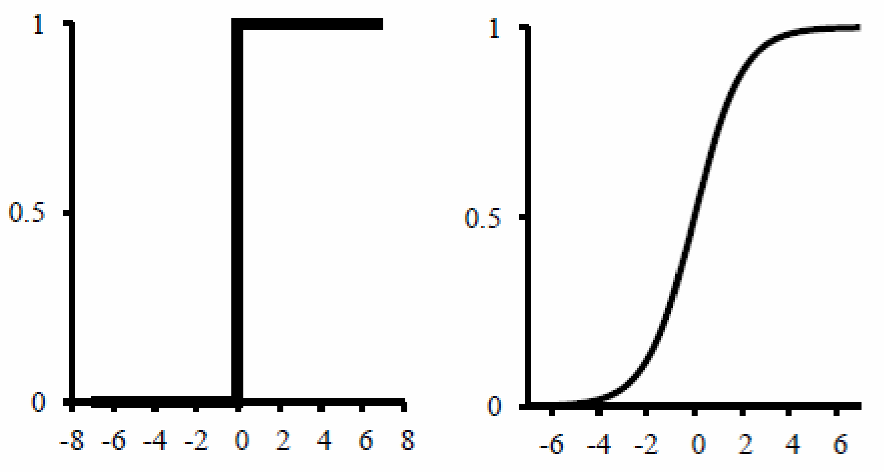
# The basic unit of neural networks

- The network consists of *units* (nodes, "neurons") connected by *links*
  - √ Carries an *activation* $a_i$ from unit $i$ to unit $j$
  - √ The link from unit $i$ to unit $j$ has a *weight* $W_{i,j}$
  - √ *Bias* weight $W_{0,j}$ to fixed input $a_0 = 1$
- *Activation* of a unit $j$
  - √ Calculate input
    $in_j = \sum W_{i,j}\, a_i \quad (i=0..n)$
  - √ Derive output
    $a_j = g(in_j)$ where
    g is the *activation function*



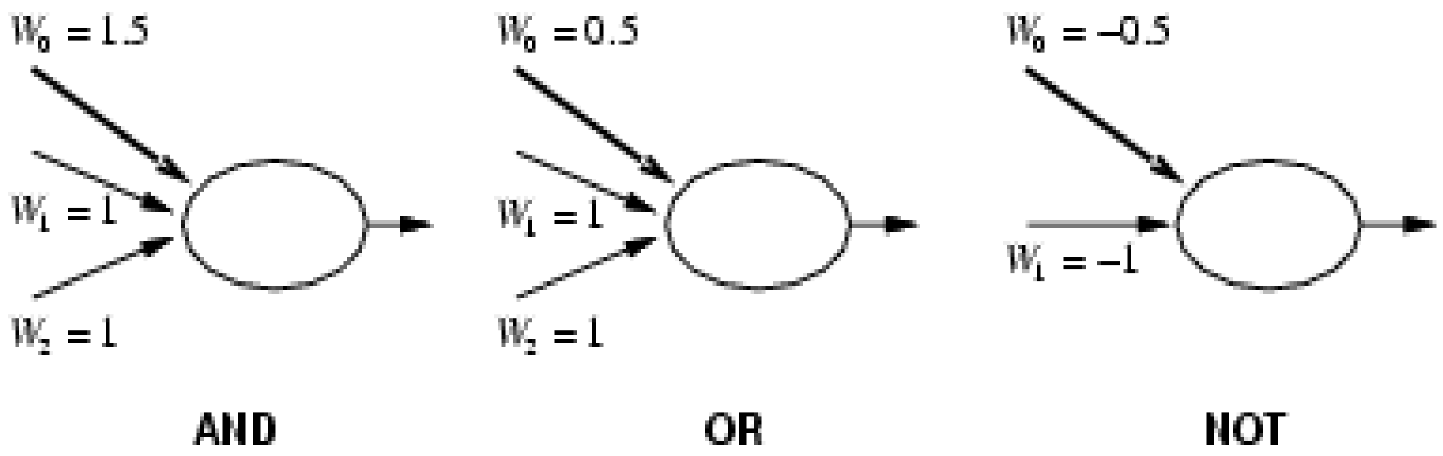
# Activation functions

- Activation function should separate well
  - √ ”Active” (near 1) for desired input
  - √ ”Inactive” (near 0) otherwise
- It should be *non-linear*
- Most used functions
  - √ *Threshold* function
  - √ *Sigmoid* function



# Neural networks as logical gates

- With proper use of *bias weight* $W_0$ to set thresholds, neural networks can compute standard logical gate functions

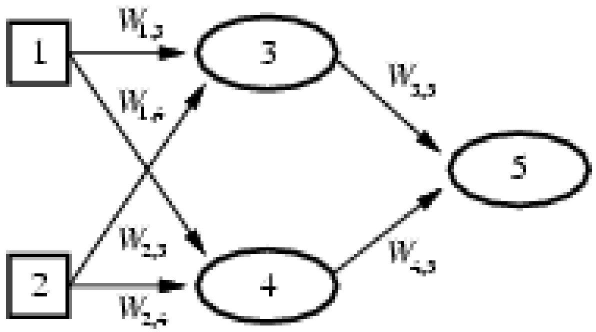


# Neural network structures

- Two main structures
  - *Feed-forward* (acyclic) networks
    - Represents a function of its inputs
    - No internal state
  - *Recurrent* network
    - Feeds outputs back to inputs
    - May be stable, oscillate or become chaotic
    - Output depends on initial state
- Recurrent networks are the most interesting and ”brain-like”, but also most difficult to understand

# Feed-forward networks as functions

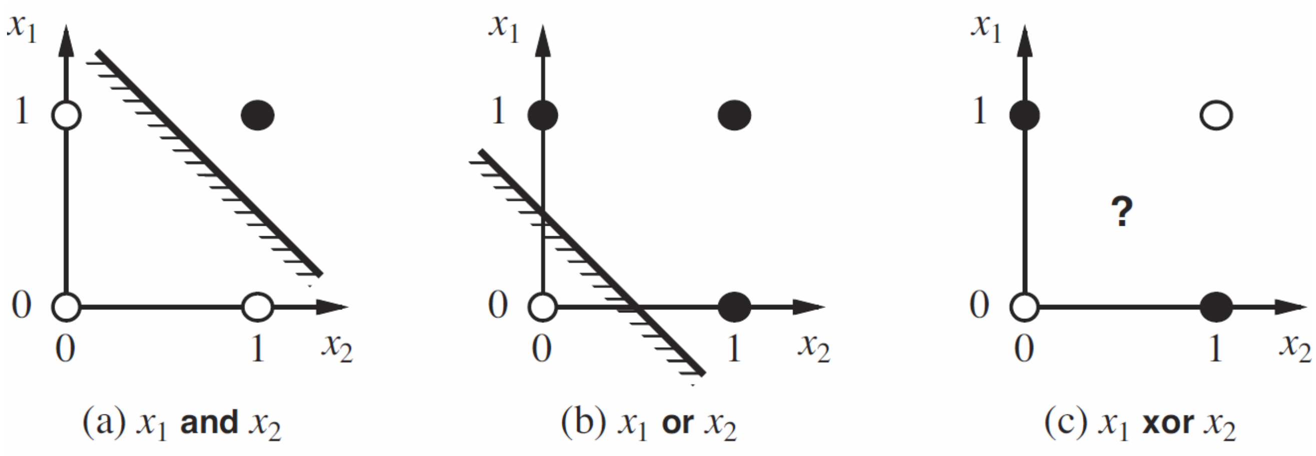
- A FF network calculates a *function* of its inputs
- The network may contain *hidden* units/layers



- By changing #layers/units and their weights, different functions can be realized
- FF networks are often used for *classification*

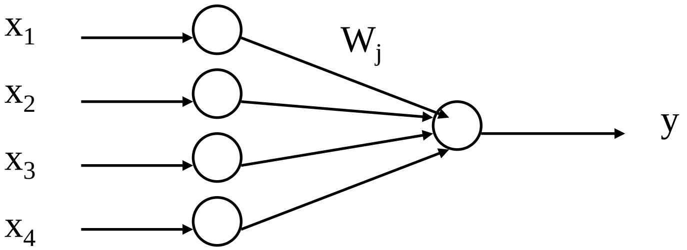
# Perceptrons

- Single-layer feed-forward neural networks are called *perceptrons*, and were the earliest networks to be studied
- Perceptrons can only act as *linear separators*, a small subset of all interesting functions
  - √ This partly explains why neural network research was discontinued for a long time



# Perceptron learning algorithm

- How to train the network to do a certain function (e.g. *classification*) based on *a training set* of input/output pairs?



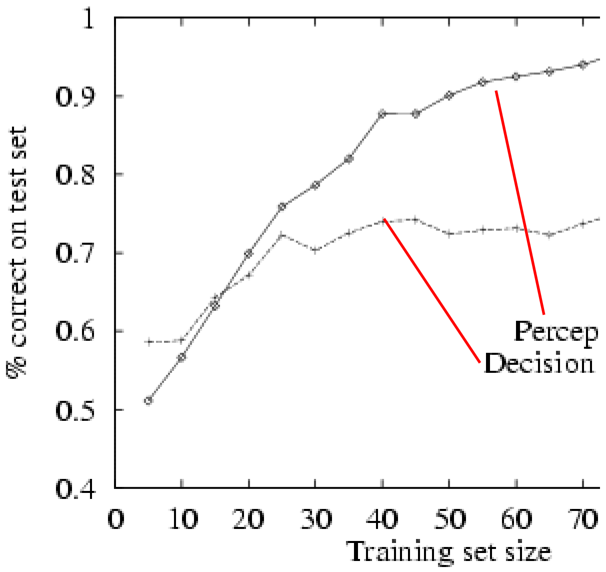

- Basic idea
  - Adjust network link weights to minimize some measure of the error on the training set
  - Adjust weights in direction that minimizes error

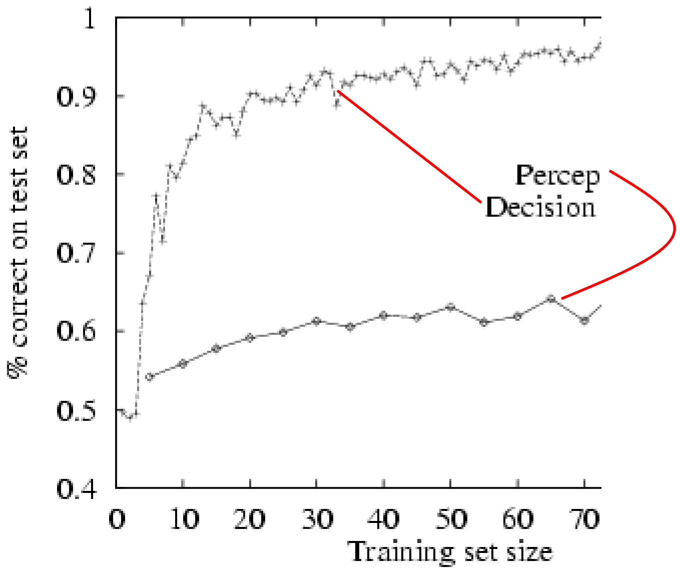
# Perceptron learning algorithm (cont.)

```
function PERCEPTRON-LEARNING(examples, network)
    returns a perceptron hypothesis
    inputs: examples, a set of examples, each with inputs x_1, x_2 ..
                             and output y
            network, a perceptron with weights W_j and act. function g
    repeat
       for each e in examples do
            in = ∑ W_j x_j[e]                    j=0 .. n
            Err = y[e] – g(in)
            W_j = W_j + α Err x_j[e]             α - the learning rate
    until some stopping criterion is satisfied
    return NEURAL-NETWORK-HYPOTHESIS(network)
```

# Performance of perceptrons vs. decision trees

- Perceptrons better at learning separable problem
- Decision trees better at ”restaurant problem”



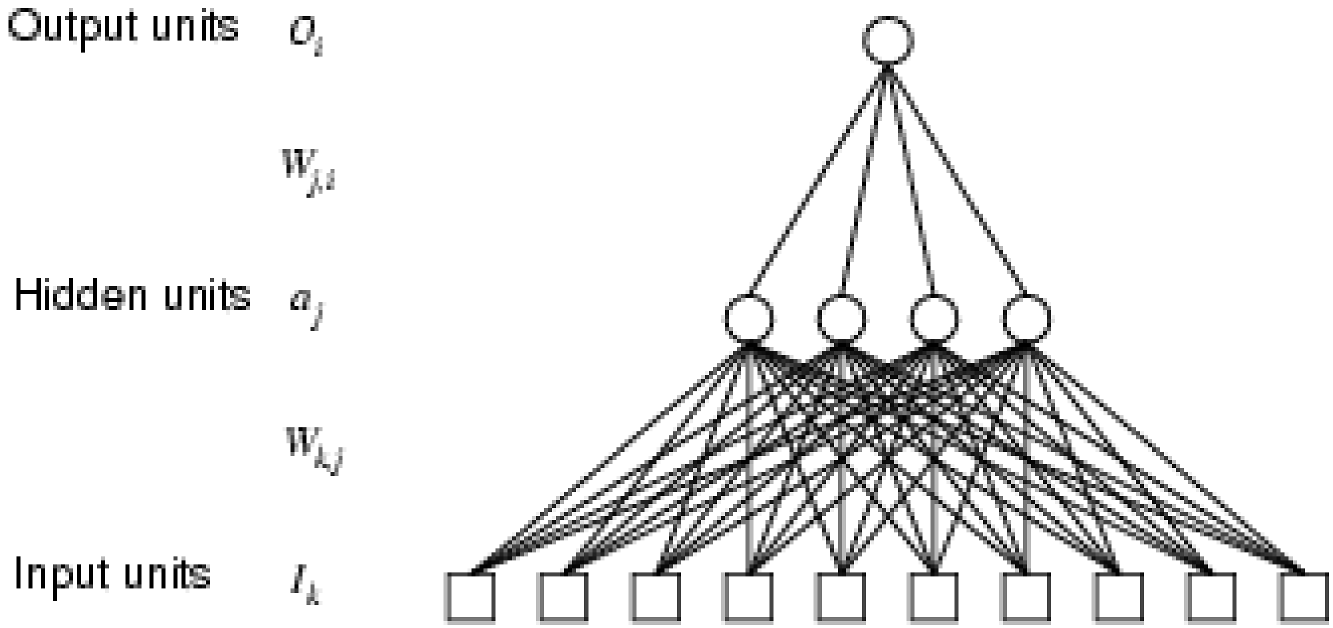



# Multi-layer feed-forward networks

- Adds *hidden* layers
  - The most common is one extra layer
  - The advantage is that more function can be realized, in effect by combining several perceptron functions
- It can be shown that
  - A feed-forward network with a single sufficiently large hidden layer can  represent any *continuous* function
  - With two layers, even *discontinuous* functions can be represented
- However
  - Cannot easily tell which functions a particular network is able to represent
  - Not well understood how to choose structure/number of layers for a particular problem

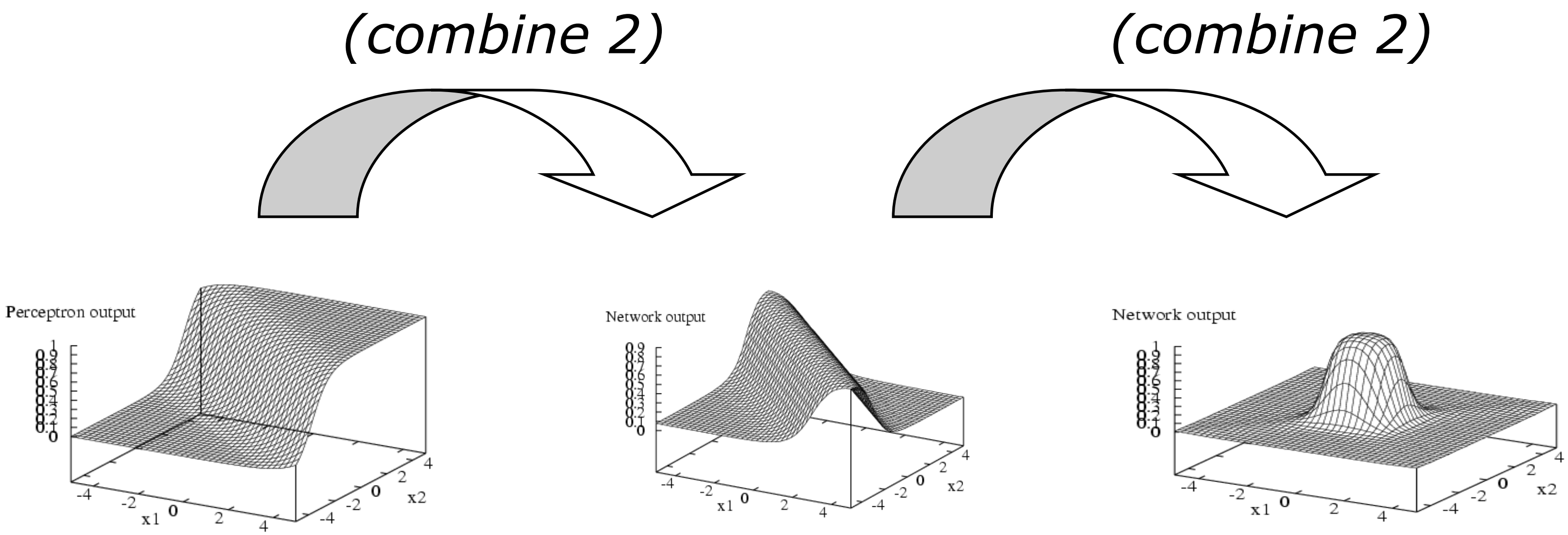
# Example network structure

- Feed-forward network with 10 inputs, one output and one hidden layer – suitable for ”restaurant problem”



# More complex activation functions

- Multi-layer networks can combine simple (linear separation) perceptron activation functions into more complex functions



# Learning in multi-layer networks

- In principle as for perceptrons – adjusting weights to minimize error
- The main difference is what ”error” at internal nodes mean – nothing to compare to
- Solution: *Propagate* error at output nodes back to hidden layers
  - Successively propagate backwards if the network has several hidden layers
- The resulting *Back-propagation algorithm* is the standard learning method for neural networks

# Learning neural network structure

- Need to learn network structure
  - Learning algorithms have assumed fixed network structure
  - However, we do not know in advance what structure will be necessary and sufficient
- Solution approach
  - Try different configurations, keep the best
  - Search space is very large (# layers and # nodes)
  - ”Optimal brain damage”: Start with full network , remove nodes selectively (optimally)
  - ”Tiling”: Start with minimal network that covers subset of training set, expand incrementally

# Summary

- Learning agents have a *performance* element and a *learning* element
- The learning element tries to improve various parts of the performance element, generally seen as *functions y = f(x)*
- Learning can be *inductive* (from examples) or *deductive* (based on knowledge)
- Differ in types of *feedback* to the agent: un-supervised, reinforcement or supervised learning
- Learning a function from examples of inputs and outputs is inductive/supervised learning
- Learning *decision trees* is an important variant

# Summary (cont.)

- Neural networks (NN) are inspired by human brains, and are complex nonlinear functions with many parameters learned from noisy data
- A perceptron is a feed-forward network with no hidden layers and can only represent linearly separable functions
- Multi-layer feed-forward NN can represent arbitrary functions, and be trained efficiently using the back-propagation algorithm