UNIVERSITÄT DES SAARLANDES

Artificial Intelligence

Machine Learning Basics

Dr. Sophia Saller

Summer 2022

Thank you to
Prof. Jana Koehler and
Annika Engel
for their contribution to the slides.

UNIVERSITÄT DES SAARLANDES

Agenda

- The learning agent revisited
- Basic concepts of machine learning
- Decision Trees
  - Entropy and information gain
- Evaluating a learned classifier
- Neural Networks
  - Perceptron and Multi-Layer Perceptron
  - Training with Backpropagation algorithm
- Risks and Challenges of applications using machine learning

UNIVERSITÄT DES SAARLANDES

Recommended Reading

Artificial Intelligence: A Modern Approach
by Stuart J. Russell and Peter Norvig (2016)

- Chapter 18: Learning from Examples
  - 18.1 Forms of Learning
  - 18.3 Learning Decision Trees
  - 18.7 Artificial Neural Networks

- Chapter 22: Natural Language Processing
  - 22.3.2 IR (Information Retrieval) System Evaluation

Further Reading

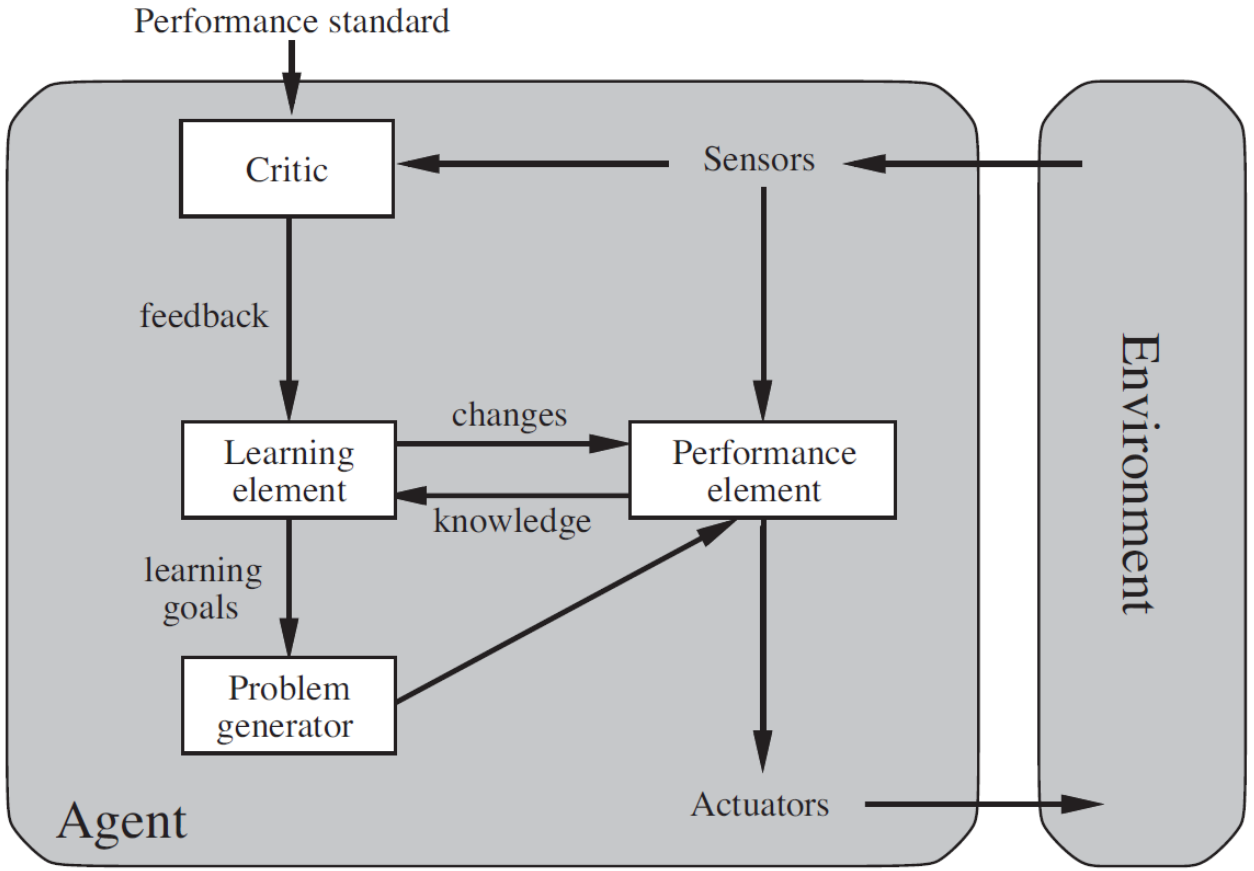
- Christopher M. Bishop: Pattern Recognition and Machine Learning, Springer 2006 https://www.microsoft.com/en-us/research/people/cmbishop/prml-book/
- Ovidiu Calin: Deep Learning Architectures, Springer 2020

More resources to learn neural networks:

- Andrew Ng´s machine learning and neural networks courses on Coursera https://www.coursera.org/instructor/andrewng
- Intuitive introduction (including example) (and watch the further videos on this list for a more thorough introduction to backpropagation) https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=1
- A nice toolbox to visualize Neural Nets in Latex: https://github.com/HarisIqbal88/PlotNeuralNet

The Learning Agent

So far, an agent's percepts have only served to help the agent choose its actions - now they also serve to improve future behavior



Russell & Norvig, Fig. 2.15

Types of Learning

The type of feedback available determines the nature of the learning problem:

- Unsupervised learning
  - No feedback is given, the agent detects patterns in the sensory input data, e.g., clustering and association algorithms
- Supervised learning
  - Agent observes example input–output pairs and learns a function that maps from input to output, e.g., decision trees and neural networks
- Reinforcement learning
  - Agent learns from a series of reinforcements (rewards or punishments) returned from the environment when the agent executes actions, e.g., AlphaZero

UNIVERSITÄT DES SAARLANDES

Supervised Learning

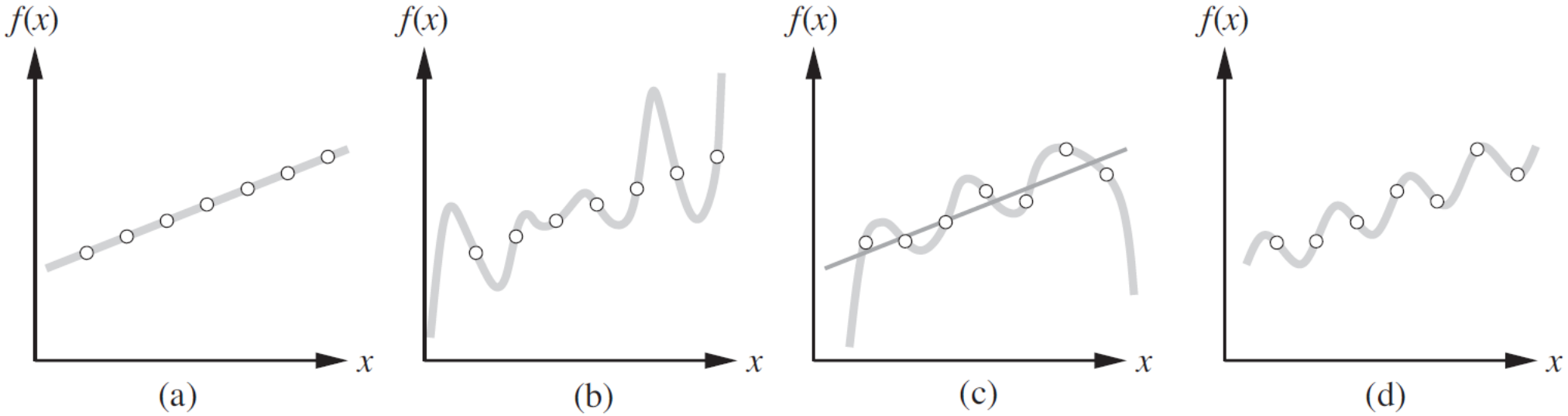
- A training example is a pair $(x, f(x))$
- The complete set of examples is called the **training set**
- Pure inductive inference: for a collection of examples for f, return a function h (**hypothesis**) that approximates f
- The function h is member of a **hypothesis space** H
- A learning problem is **realizable** if the hypothesis space H contains the true function f

Accuracy of a Learned Function and Ockham's Razor

- A learned hypothesis h is given a **test set** of examples that are distinct from the training set
  - h is **consistent** with the training set if it agrees with all examples from the training set
  - h **generalizes** well if it correctly predicts value of $y = f(x)$ for examples from the test set

- How do we choose from among multiple consistent hypotheses?
  - Ockham's Razor: prefer the simplest hypothesis consistent with the data
  - In general, there is a tradeoff between complex hypotheses that fit the training data well and simpler hypotheses that may generalize better
  - Maximize a combination of consistency and simplicity

Example: Which Hypothesis is Preferred by Ockham´s Razor?



Russell & Norvig, Fig. 18.1

a) Consistent hypothesis that agrees with all the data

b) Degree-7 polynomial that is also consistent with the data set

c) Data set that can be approximated consistently with a degree-6 polynomial

d) Sinusoidal exact fit to the same data

Supervised Learning is a Form of Inductive Learning

- Supervised learning is a simplified model of real learning, which ignores prior knowledge of the agent, and that induces a hypothesis from input-output example pairs
  - Assumes that such example pairs exists
- Deductive learning learns a new rule from an existing rule, which is entailed by the data and more specific (and thus useful, because more efficient to apply)
- Supervised learning shown impressive results over the last years
  - Training data occur in massive amounts through digitization (observable environments)
  - Annotation of input data (labeling) to add the expected output is possible, but labor-intensive and risks that human bias is encoded into the data

UNIVERSITÄT DES SAARLANDES

Decision Tree Learning

UNIVERSITÄT DES SAARLANDES

Decision Tree

Input

Description of an example object or a situation through a fixed set of attributes
$A = \{a_1, \ldots, a_n\}$

Output

A single value, a final “decision”, of the example based on its attributes

- Both, input values and output values can be discrete or continuous
- Learning a discrete-valued functions leads to classification problems
- If the classification is Yes/No → Boolean Decision Tree
- Learning a continuous function is called **regression**

Boolean Decision Tree when Training

Input

A set S of vectors of input attributes $A_i = \{a_1, \dots, a_n\}$ and for each vector a single Boolean output value y of the goal predicate (function to be learned)

Output

Definition of the goal predicate in the form of a decision tree

$$S = \{A_i, y_i\}_{i \in \{1,\dots,m\}} \mapsto D$$

where $D: \{a_j\}_{j \in J} \mapsto \{\text{TRUE}, \text{FALSE}\}$ and $J \subseteq \{1, \dots, n\}$.

Properties of (Boolean) Decision Trees

- Each root or internal node of the decision tree represents a test of an attribute
- Branches are labeled with the possible outcome attribute values of the test
- Each leaf node specifies the Boolean value to be returned if this leaf is reached
- A classification of an instance means to traverse the tree from the root node to one of the leaf nodes

The Restaurant Example

- **Goal predicate:** WillWait
- **Test predicates:**
  1. *Patrons*: How many guests are there? (none, some, full)
  2. *WaitEstimate*: How long do we have to wait? (0-10, 10-30, 30-60, >60)
  3. *Alternate*: Is there an alternative? (True/False)
  4. *Hungry*: Am I hungry? (True/False)
  5. *Reservation*: Have I made a reservation? (True/False)
  6. *Bar*: Does the restaurant have a bar to wait in? (True/False)
  7. *Fri/Sat*: Is it Friday or Saturday? (True/False)
  8. *Raining*: Is it raining outside? (True/False)
  9. *Price*: How expensive is the food? ($, $$, $$$)
  10. *Type*: What kind of restaurant is it? (French, Italian, Thai, Burger)

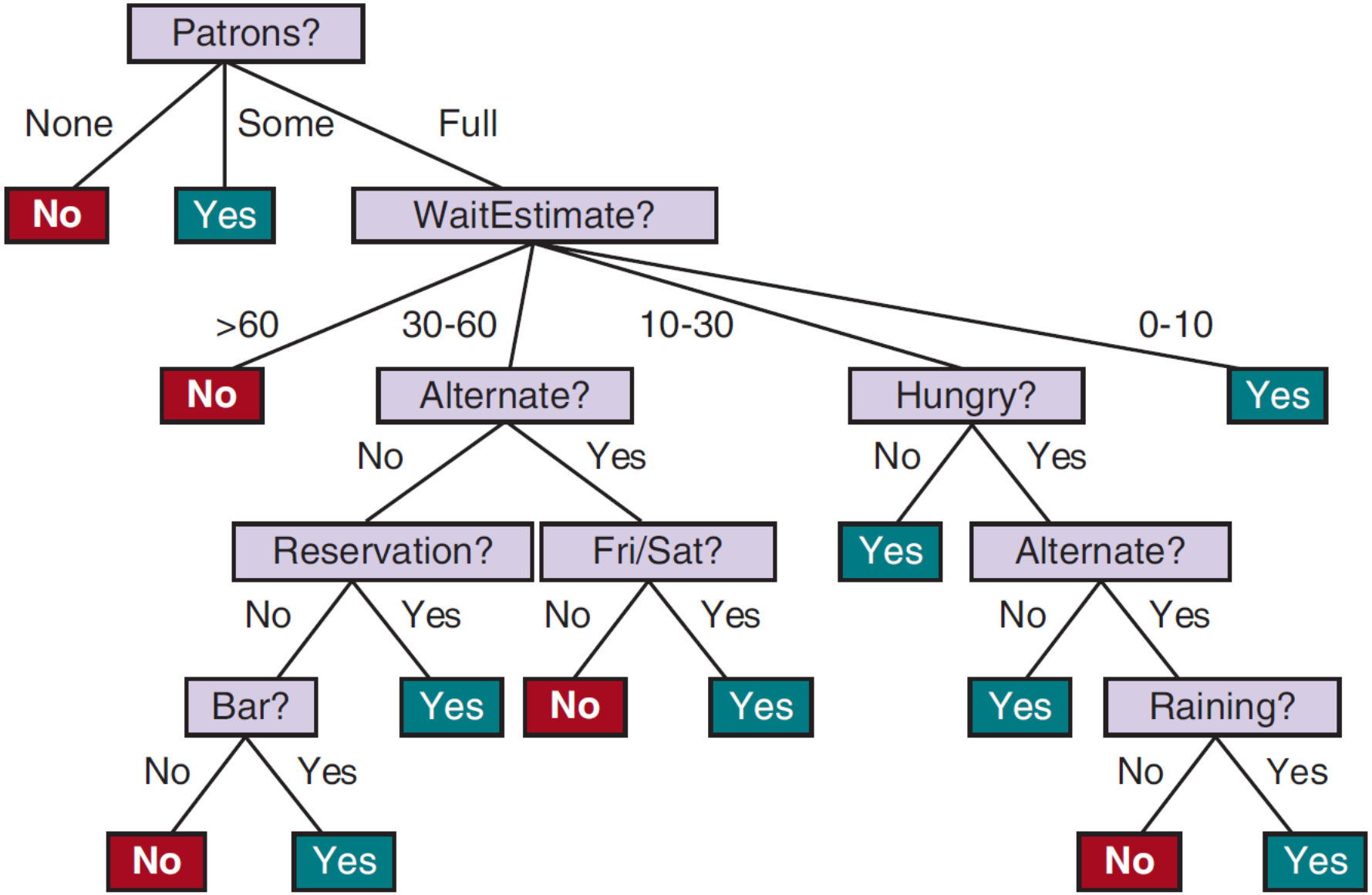
The Training Set

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Russell & Norvig, Fig. 18.3

Classification of examples (and value of the goal predicate) is positive (T) or negative (F)

A Boolean Decision Tree for deciding whether to wait for a Table



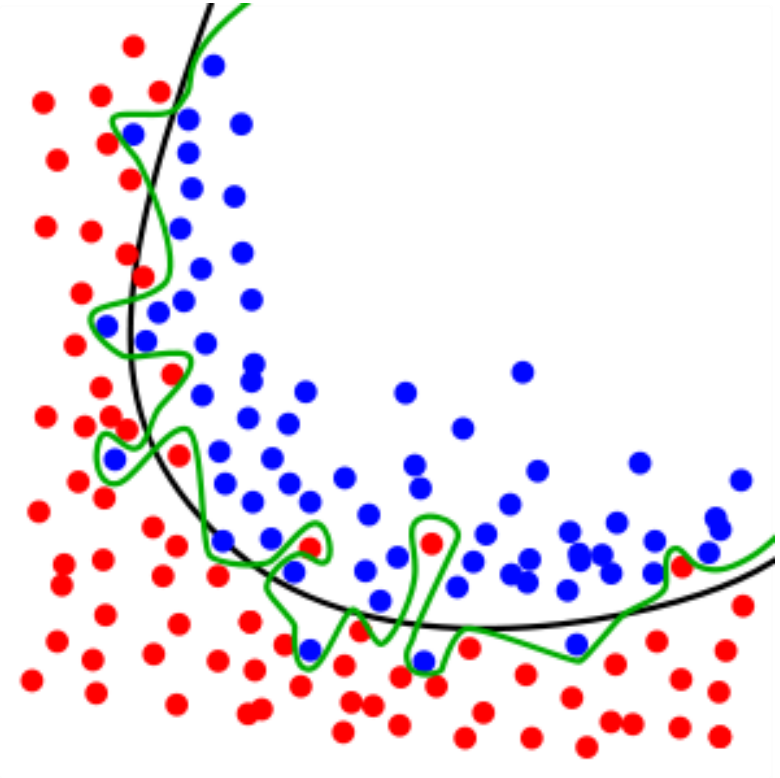
Russell & Norvig, 4th edition
Fig. 19.3

Inducing Decision Trees from Examples

- Naive solution: we simply construct a tree with one path to a leaf for each example
- In this case, we test all the attributes along the path and attach the classification of the example to the leaf
- Whereas the resulting tree will correctly classify all given examples, it will not say much about other cases
- It just memorizes the observations and does not generalize
  - "Overfitting"

An Intuitive Characterization of Overfitting

- A machine learning algorithm that suffers from overfitting will generate a hypothesis h, which is consistent with all training examples
- However, h is not generalizing beyond the training data
  - It will unlikely predict unseen data well
  - It often does not correspond to Ockham´s razor as it is not a simpler description of the training data than the training data themselves

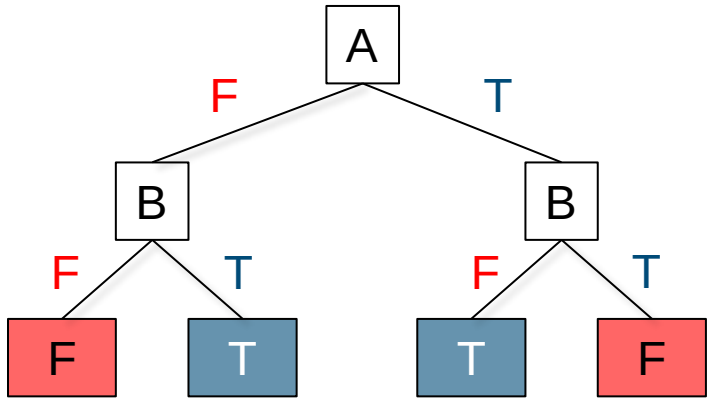


https://en.wikipedia.org/wiki/Overfitting
(June 2021)

Memoization of the Training Set

- A trivial consistent decision tree for any training set has one path from the root node to a leaf node for each example attribute vector (row in the table)

Instance	Attr. A	Attr. B	Goal: A **XOR** B
X1	F	F	F
X2	F	T	T
X3	T	F	T
X4	T	T	F



- Prefer to find a small tree consistent with the training examples
  - (Recursively) choose the most “significant" attribute as root of the (sub)tree that is constructed in the next step of the learning algorithm

Size of the Hypothesis Space for Decision Tree Learning

- How many distinct decision trees can we built with n Boolean attributes?
  - Number of distinct truth tables with n rows : 2^n
  - "Answer" column of a truth table: 2^n-bit number that defines the function
  - Number of Boolean functions over $n = 2^{2^n}$ (2^{2^n} functions from 2^n to $\{0,1\}$)
  - There are more than that number of trees, since more than one tree can compute the same function

- 6 Boolean attributes yield a hypothesis space of at least 18,446,744,073,709,551,616 trees
- For the 10 attributes of the restaurant example, there are more than $2^{1024} = 10^{308}$ potential candidate decision trees

Expressiveness of Decision Trees

- Each decision tree hypothesis for the *WillWait* goal predicate is equivalent to an assertion of the form

$$WillWait \Leftrightarrow Path_1 \vee Path_2 \vee \cdots \vee Path_n$$

where each $Path_i$ is the conjunction of tests along a path from the root of the tree to a leaf with the value true, e.g., $Path = Patrons = Full \wedge WaitEstimate = 0\text{–}10$

- Limitation:

$$\exists\, r_2 \colon NearBy(r_2, r) \wedge Price(r, p) \wedge Price(r_2, p_2) \wedge Cheaper(p_2, p)$$

cannot be represented as a test. Adding a new attribute *CheaperRestaurantNearby* makes the decision tree grow exponentially

Compact Representations of Boolean Functions

- For every Boolean function we can construct a decision tree by translating every row of a truth value table to a path in the tree
  - a tree of size exponential in the number of attributes
- There are functions that require an exponentially large decision tree:

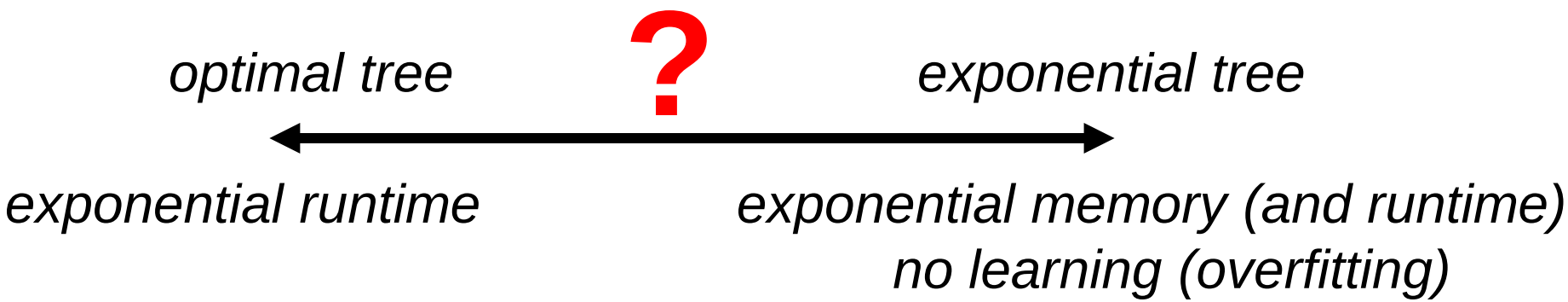
Parity function: $p(x) = \begin{cases} 1, & \text{even number of inputs are 1} \\ 0, & \text{otherwise} \end{cases}$

Majority function: $p(x) = \begin{cases} 1, & \text{half of inputs are 1} \\ 0, & \text{otherwise} \end{cases}$

- There is no consistent and compact (non-exponential) representation for all possible Boolean functions

Finding a Smallest Tree is Intractable

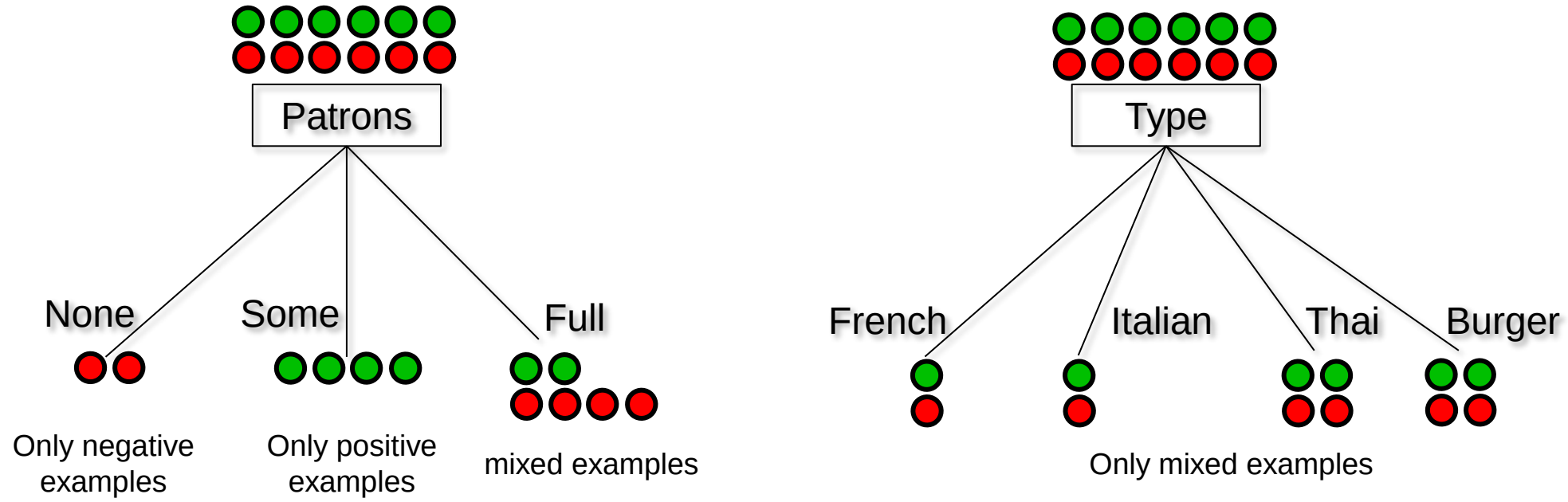
- Applying Ockham's Razor, we would like to find the simplest decision tree that is consistent with the training set
  - An optimal decision tree minimizes the expected number of tests
- Finding an optimal decision tree is NP-complete [Hyafil & Rivest, 1976]
  - No efficient algorithm to construct an optimal decision tree exists, unless P=NP



➢ Can we devise an efficient decision tree learning algorithm that generates "smallish" trees?

Splitting Examples by Choosing an Attribute

- **Idea:** a good attribute splits the examples into subsets that are (ideally) all *positive* or all *negative: Patrons* is a better choice than *Type*
- *Type* is a poor attribute - it leaves us with four subsets each containing positive and negative examples



Idea for a Recursive Decision Tree Learning Algorithm

- Divide and Conquer approach:
  - Choose a good (or better: the best) attribute
- Split the training set into subsets each corresponding to a particular value of that attribute
- Once the training set is divided into several smaller training sets, the algorithm can recursively apply this process to the smaller training sets
  - Select the next best attribute and split the remaining training sets further
  - In general, after an attribute test splits up the examples, each outcome is a new decision tree learning problem with fewer examples and one less attribute

Four Cases to consider for a recursive Decision Tree Learning Algorithm

1. If the remaining examples are all positive (or all negative), then terminate and return Yes (or No)
2. If there are some positive and some negative examples, then choose the best attribute to split them
3. If there are no examples left, then no example has been observed for this combination of attribute values:
   - Return a default value calculated from the plurality classification of all the examples that were used in constructing the node's parent
   - Answer *Yes* if the majority of the parent node's examples is positive, otherwise *No*
4. If there are no attributes left, but both positive and negative examples, then these examples have exactly the same description, but different classifications:
   - Do a plurality classification of the remaining examples: Answer *Yes* if the majority of the remaining examples is positive, otherwise *No*

Root Causes for Inconsistent Classifications

- If there are no attributes left, but both positive and negative examples, then these examples have exactly the same description, but different classifications

- Possible root causes:
  - There is an error or noise in the data
  - The domain is nondeterministic
  - The agent cannot observe an attribute that would distinguish the examples

The Decision-Tree-Learning Algorithm

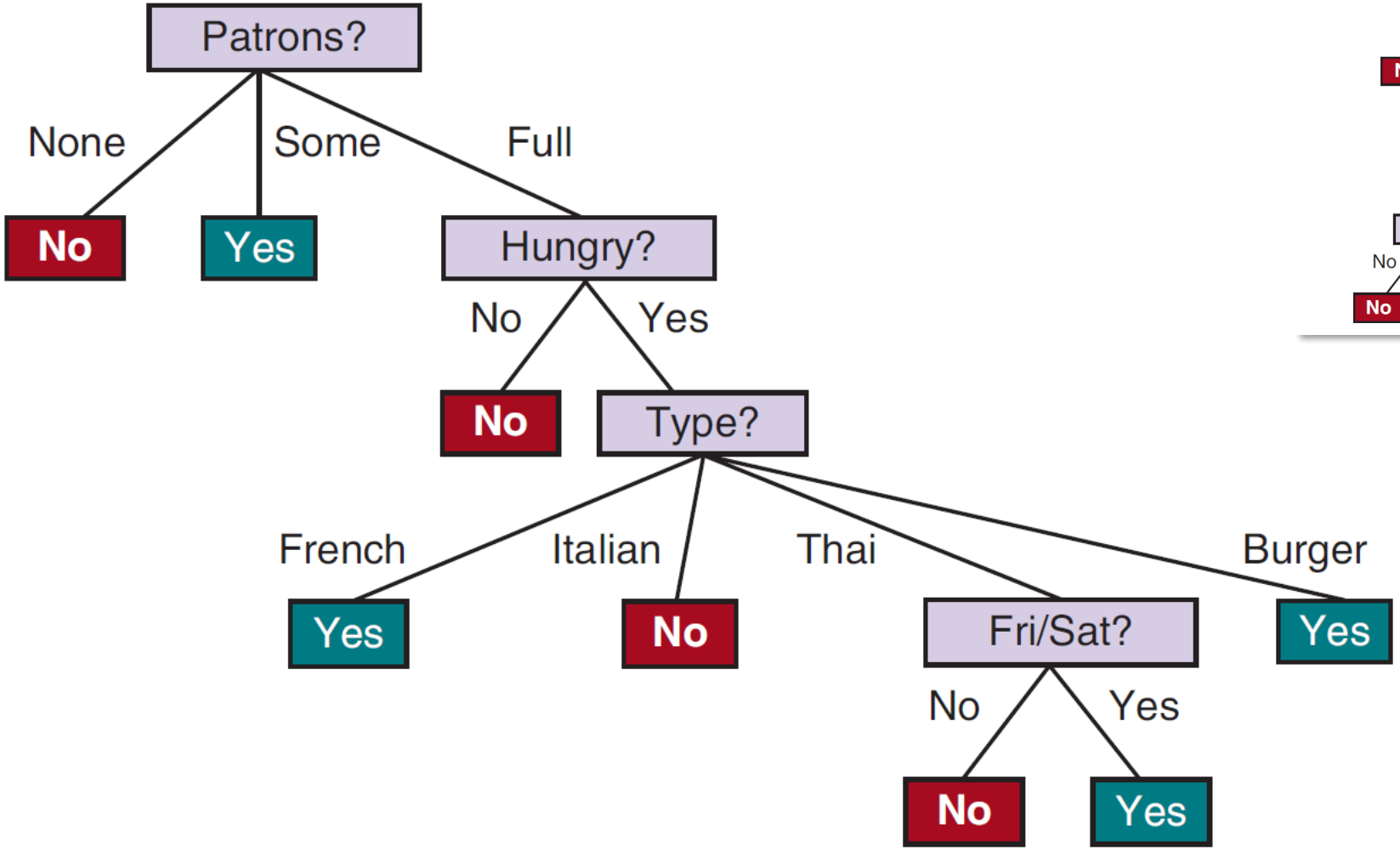
function DECISION-TREE-LEARNING($examples, attributes, parent\_examples$) **returns** a tree

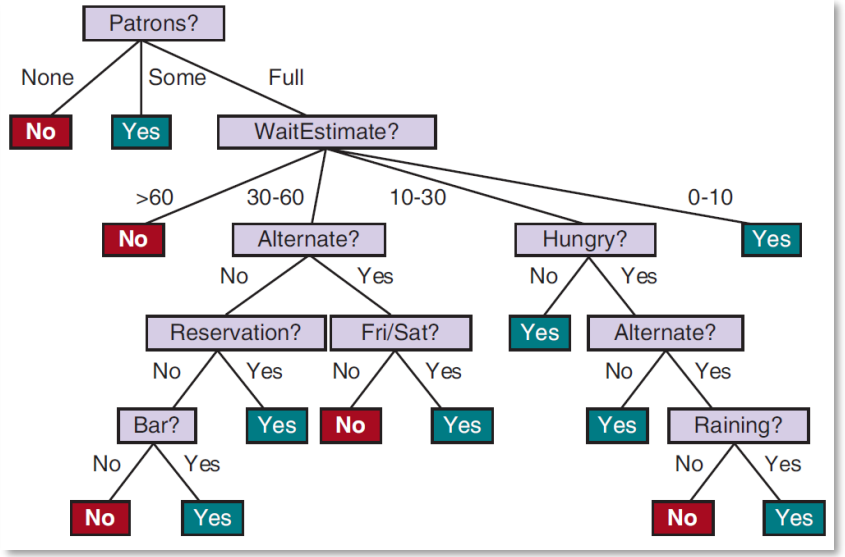
if $examples$ is empty **then return** PLURALITY-VALUE($parent\_examples$)
else if all $examples$ have the same classification **then return** the classification
else if $attributes$ is empty **then return** PLURALITY-VALUE($examples$)
else
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE($a, examples$)
  $tree \leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e : e \in examples$ **and** $e.A = v_k\}$
    $subtree \leftarrow$ DECISION-TREE-LEARNING($exs, attributes - A, examples$)
    add a branch to $tree$ with label $(A = v_k)$ and subtree $subtree$
  return $tree$

Russell & Norvig, Fig. 18.5

- IMPORTANCE assigns importance measure to each attribute (precise definition see later slides)
- PLURALITY-VALUE selects the most common output value among a set of examples breaking ties randomly

The Learned Tree





first version of the tree

Russell & Norvig, 4th edition
Fig. 19.6

Properties of the Decision Tree Learning Algorithm

- The algorithm returns a tree that is consistent with all training examples (in general: if no plurality decision needed to be taken during construction)
  - Some attributes tests are not included since the algorithm can classify the examples without them
- The learned tree is considerably simpler than the one originally given
  - The greedy search used is designed to approximately minimize the depth of the final tree
- Ideally, the Importance function selects the perfect attribute, which divides the examples into sets of only positive or only negative examples to construct a leaf
  - A formal measure of the importance of an attribute can be based on the information gain of this attribute

Properties of Decision Trees

+ Can represent any (Boolean) function
+ Can handle both numeric and categorical data
+ Require only little data preparation
+ Perform well on smaller and larger data sets
+ Small decision tree can be generated by ranking attributes based on information gain
+ Easy to understand and interpret for humans (explainability)

- Finding a minimal decision tree is NP-hard
- Size of the tree can be exponential in the number of attributes
- Algorithms tend to create complex trees that do not generalize well (overfitting)
- As any learning algorithm, DT learning is sensitive to data quality
- Small change in training data can result in a big change in the tree
- Less predictive power than other machine learning algorithms

UNIVERSITÄT DES SAARLANDES

Information Gain and Entropy

The Notion of Information Gain and Entropy

- The IMPORTANCE function will be based on the information gain provided by an attribute
  - Information gain is defined based on **entropy,** the fundamental quantity in information theory (Shannon and Weaver, 1949)

- Entropy is a measure of the uncertainty of a random variable
  - Acquisition of information corresponds to a reduction in entropy
  - The less one knows about the outcome of an action or event, the more valuable the information when the outcome can be observed
  - The higher the information gain when observing the outcome

Defining Entropy: Example of Tossing a Coin

- A random variable with only one value - an unfair coin that always comes up heads - has no uncertainty
  - Entropy is defined as zero
  - No gain in information by observing its value, because the outcome is known in advance

- A flip of a fair coin is equally likely to come up heads or tails, 0 or 1
  - Counts as **1 bit of entropy**

- The outcome of the roll of a fair *six-sided* die can be encoded using **3 bits**
  - It takes 3 bits to describe one of the six outcomes of equal probability
  - Its entropy will be a value between 2 and 3

Definition of (Bitwise) Entropy

- Entropy H of a random variable V with values v_k, each with probability $P(v_k)$, is defined as

$$\mathrm{H}(V) = \sum_k P(v_k) \log_2 \left(\frac{1}{P(v_k)} \right) = -\sum_k P(v_k) \log_2 P(v_k)$$

$\log_n \left(\frac{a}{b}\right) = \log_n(a) - \log_n(b)$, here $a = 1$ and thus $\log_n(1) = 0$ more at CMS>Information>Materials>Supplementary Materials

$$\sum_k P(v_k) \log_2 \left(\frac{1}{P(v_k)} \right) = \underbrace{\sum_k P(v_k) \underbrace{\log_2(1)}_{0}}_{0} - \sum_k P(v_k) \log_2(P(v_k)) = -\sum_k P(v_k) \log_2\big(P(v_k)\big)$$

- Entropy of a fair coin:

$$H(Fair) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1 \text{ bit}$$

- Entropy of an unfair ("loaded") coin that gives 99% heads:

$$H(Loaded) = -(0.99 \log_2 0.99 + 0.01 \log_2 0.01) \approx 0.08 \text{ bits}$$

UNIVERSITÄT DES SAARLANDES

Example: Entropy of a Fair Die

$$\mathrm{H}(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

$P(v_k) = \frac{1}{6} = 0.166$

Sum up 6 times for the 6 possible outcomes:

$$H(Die) = -6 \times (0.166 \log_2 0.166)$$
$$\approx -6 \times (0.166 \times -2.59074485332) \text{ bits}$$
$$\approx -6 \times -0.43006364565$$
$$\approx 2.5803818739$$

Entropy of a Boolean Variable

- $B(q)$ is the entropy of a Boolean variable that is true with probability q

$$B(q) = -(q \log_2 q + (1-q) \log_2 (1-q))$$

- For example,

$$H(Loaded) = B(0.99) = -(0.99\ log_2 0.99 + 0.01\ log_2 0.01\ ) \approx 0.08\ bits$$

- If a training set contains p positive and n negative examples, then the entropy of the goal attribute on the whole set is

$$H(Goal) = B\left(\frac{p}{p+n}\right)$$

$\frac{p}{p+n}$ is the fraction of positive outcomes

Influence of Testing an Attribute on Entropy

- The restaurant example has $p = n = 6$, $\frac{p}{p+n} = 0.5$ as there are 50 % positive examples where a guest waits
- The entropy of the goal attribute is $B(WillWait) = 1\ bit$, which encodes the two possible outcomes (wait or do not wait)

- A test on a single attribute cannot give more than this entropy
- Measure this value precisely by determining the entropy after performing the test on this attribute
  - The value of an attribute A depends on the additional information that is still required to classify the remaining examples after A was applied

Entropy of the Goal Predicate w.r.t. a specific Attribute Value

- An attribute A with d distinct values divides the training set E into subsets $E_1, \ldots, E_d$

- Each subset E_k with $k = 1, \ldots, d$ has p_k positive and n_k negative examples, so along this branch we need an additional

$$B\left(\frac{p_k}{p_k + n_k}\right)$$

Type=Italian: $B\left(\frac{1}{2}\right)$

bits of information to classify the remaining examples

- A randomly chosen example from the training set has the k^{th} value for the attribute with probability

$$\frac{p_k + n_k}{p + n}$$

Type=Italian: $\left(\frac{1+1}{6+6}\right)$

Expected Entropy after Testing an Attribute

- Multiplying the entropy of an outgoing branch for a specific attribute value (1) with the probability of its occurrence (2) and summing this up over all possible attribute values $k = 1, \ldots, d$ with number of positive outcomes p_k and number of negative outcomes n_k yields the expected entropy after the test on attribute A

(2) $\frac{p_k + n_k}{p + n}$

(1) $B\left(\frac{p_k}{p_k + n_k}\right)$

$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p + n} B\left(\frac{p_k}{p_k + n_k}\right)$$

Information Gain

- The **information gain** from the attribute test on A is the reduction in entropy of the goal predicate by the expected entropy after testing on A
  - IMPORTANCE function computes the information gain and chooses the attribute with the largest value

$$Gain(A) = B\left(\frac{p}{p+n}\right) - Remainder(A)$$

$$= B\left(\frac{p}{p+n}\right) - \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right)$$

Example: Information Gain of Attributes in the Restaurant Example

$B\left(\frac{p}{p+n}\right)$ with $p = 6$ and $n = 6$

$$\sum_{k=1}^{d} \frac{p_k + n_k}{p + n} B\left(\frac{p_k}{p_k + n_k}\right)$$

3 different values of the Patrons attribute
None: 0 positive + 2 negative
Some: 4 positive + 0 negative
Full: 2 positive + 4 negative

$$Gain(Patrons) = 1 - \left[\underbrace{\frac{2}{12}B\left(\frac{0}{2}\right)}_{\text{None}} + \underbrace{\frac{4}{12}B\left(\frac{4}{4}\right)}_{\text{Some}} + \underbrace{\frac{6}{12}B\left(\frac{2}{6}\right)}_{\text{Full}}\right] \approx 0.541 \text{ bits}$$

$$Gain(Type) = 1 - \left[\underbrace{\frac{2}{12}B\left(\frac{1}{2}\right)}_{\text{French}} + \underbrace{\frac{2}{12}B\left(\frac{1}{2}\right)}_{\text{Italian}} + \underbrace{\frac{4}{12}B\left(\frac{2}{4}\right)}_{\text{Thai}} + \underbrace{\frac{4}{12}B\left(\frac{2}{4}\right)}_{\text{Burger}}\right] = 1 - \frac{12}{12} = 0 \text{ bits}$$

4 different values of the Type attribute
French: 1 positive + 1 negative
Italian: 1 positive + 1 negative
Thai: 2 positive + 2 negative
Burger: 2 positive + 2 negative

UNIVERSITÄT DES SAARLANDES

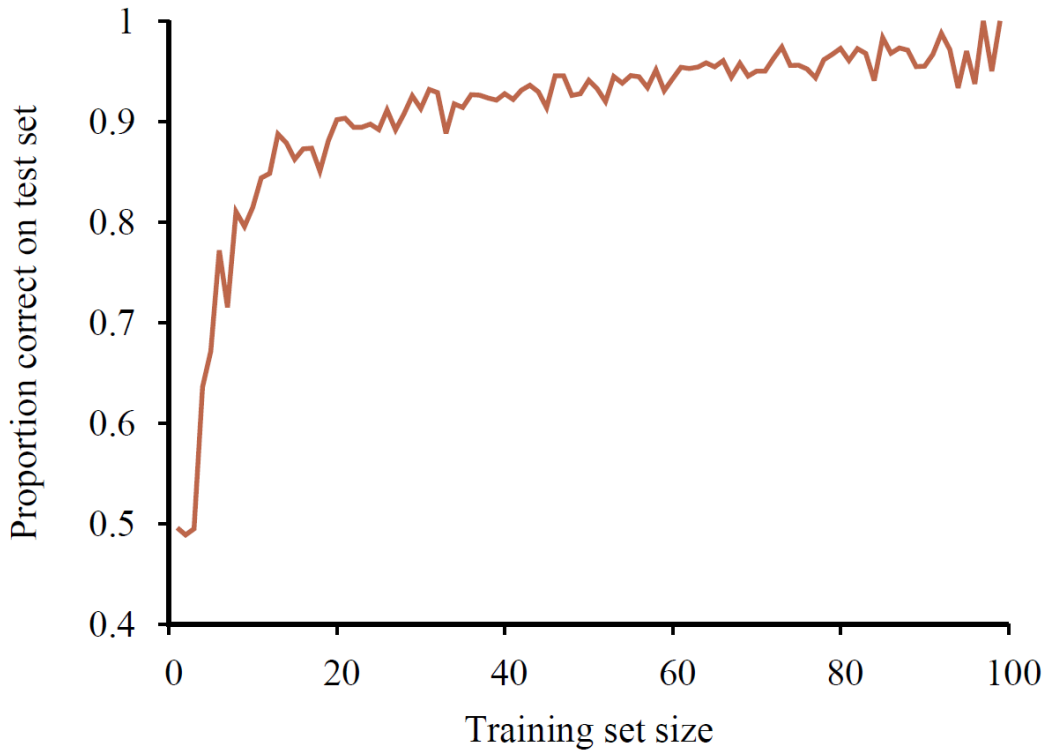
Evaluating a Learned Classifier

Machine Learning Workflow: Training and Testing

1. Collect a large number of examples with known outcome (ground truth)
   - The data should be representative for the application domain, of high quality and free of noise, errors, and bias
2. Divide the examples into two disjoint sets: the training set and the test set
3. Use the training set to generate h
4. Evaluate h based on the test set
   - For example, measure the percentage of examples of the test set that are correctly classified by h
5. Repeat the process for randomly-selected training/test sets of different sizes
   - Result is a learning curve for the learning algorithm

Learning Curve for the Decision Tree Learner on the Restaurant Example

- Learning curve (aka happy graph) evaluates the accuracy of a learning algorithm based on a given example set: 100 examples in the restaurant problem
  - Start with training set size 1 and increase in each training by 1 until 99
- For each size, repeat the process of randomly splitting 20 times, and average the test results of the 20 trials
- The curve shows that as the training set size grows, the accuracy increases



Russell & Norvig, 4th edition
Fig. 19.7

Separation of Training and Test Set

- Training and test sets must be kept separate!
- Common error: Retraining an algorithm using examples from the test set and then repeating the evaluation using the same test set
  - Knowledge about the test set is learned by the algorithm and represented in the resulting classifier
  - Training and test sets are no longer independent

- Note the limitation: the accuracy of machine learning is only demonstrated using a set of example data - how can we know that $h \approx f$ for all data?
  - No generalization beyond the sample data is possible with certainty
  - Lab testing results rarely transferrable to the real-world

UNIVERSITÄT DES SAARLANDES

Systematic Splitting of Example Data using k-fold Cross-Validation

- Basic idea: each example can serve as training data and test data

1. Split data into k equal subsets
2. Perform k rounds of learning on $k-1$ sets of training data
3. Test on remaining $1/k$ of data

- Average test set score of the k rounds should then be a better estimate than a single score
  - Popular values for k are 5 and 10

UNIVERSITÄT DES SAARLANDES

Possible Outcomes of a Boolean Classifier on a Test Example

- **TRUE Positive** $(\mathrm{T}p)$
  - Classifier correctly predicts a positive example as being positive
- **TRUE Negative** $(\mathrm{T}n)$
  - Classifier correctly predicts a negative example as being negative
- **FALSE Positive** $(\mathrm{F}p)$
  - Classifier incorrectly predicts a negative example as being positive
- **FALSE Negative** $(\mathrm{F}n)$
  - Classifier incorrectly predicts a positive example as being negative

Example: False Positives and False Negatives in Medical Diagnosis

Lilly has measles

ground truth

Lilly has scarlet fever

Classifier Output → **Lilly does <u>not</u> have measles**
(false negative: the virus infection is not recognized)
➢ No treatment

Classifier Output → **Lilly has measles**
false positive: the bacterial infection is not recognized
➢ Wrong treatment

UNIVERSITÄT DES SAARLANDES

Example Testing Scenario

- Test set size 165 examples
  - Positive examples $p_T = 105$
  - Negative examples $n_T = 60$

- Outcomes of the classifier
  - TRUE Positive $\mathrm{T}p = 100$
  - TRUE Negative $\mathrm{T}n = 50$
  - FALSE Positive $\mathrm{F}p = 10$
  - FALSE Negative $\mathrm{F}n = 5$

- Note that $p_T = \mathrm{T}p + \mathrm{F}n$ and $n_T = \mathrm{T}n + \mathrm{F}p$

UNIVERSITÄT DES SAARLANDES

Accuracy: Percentage of Correct Classifications

- Accuracy shows the proportion of correct classifications compared to the total number of tests

$$accuracy = \frac{Tp + Tn}{p_T + n_T}$$

- A perfect classifier that has learned the correct function $h = f$ has accuracy 1 (or 100%) as $Tp = pT$ and $Tn = nT$ hold

Recall/Sensitivity: Performance when Answering YES

- Recall measures how often a classifier returns YES in comparison to the positive examples
- Recall is also called True-Positive Rate or Sensitivity
  - For example, i information retrieval, recall is the fraction of the relevant documents that are successfully retrieved

$$recall = \frac{Tp}{Tp + Fn}$$

- A perfect classifier has recall = 1 (or 100%) because $Fn = 0$

Specificity: Performance when Answering NO

- Specificity defines how often a classifier returns NO in comparison to the negative examples

- Specificity is also called True-Negative Rate:
  - For example, specificity is the estimated probability that a randomly selected patient is correctly identified as not having a tested disease

$$specificity = \frac{Tn}{Tn + Fp}$$

- A perfect classifier has specificity = 1 (or 100%) because $Fp = 0$

UNIVERSITÄT DES SAARLANDES

Precision: Performance when Answering YES and Tn cannot be assessed

- Often, the size of Tn cannot be determined in an application domain
  - For example: how many web pages are not correct matches for a Google search query?
  - Accuracy and specificity cannot be calculated in these cases

- Instead, use precision: When the classifier predicts YES, how often is it correct?

$$precision = \frac{Tp}{Tp + Fp}$$

- A perfect classifier has precision = 1 (or 100%) because $Fp = 0$

F_1 Score

- The F_1 score provides a single measure that summarizes the overall performance combining precision and recall
  - F_1 is the harmonic mean between precision and recall
  - F_1 is independent of Tn

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

- A perfect classifier has an F_1 score of 1 due perfect precision and recall
- The lowest possible value is 0 if either precision or recall is zero

Example

- Test set size 165 examples
  - Positive examples $p_T = 105$
  - Negative examples $n_T = 60$
- Outcomes of the classifier
  - TRUE Positive T$p = 100$
  - TRUE Negative T$n = 50$
  - FALSE Positive F$p = 10$
  - FALSE Negative F$n = 5$

- $accuracy = \frac{Tp+Tn}{p_T+nT} = \frac{150}{165} = 0.909$
- $recall = \frac{Tp}{Tp+Fn} = \frac{100}{100+5} = 0.952$
- $specificity = \frac{Tn}{Tn+Fp} = \frac{50}{60} = 0.833$
- $precision = \frac{Tp}{Tp+Fp} = \frac{100}{100+10} = 0.909$
- $F_1 = 2 \cdot \frac{precision \cdot recall}{precision+recall} = 2 \cdot \frac{91 \cdot 95}{91+95} = 92{,}95$

A Probabilistic Interpretation of Precision and Recall

Let us consider a medical diagnosis system for cancer

- Recall is the probability of a positive test given that the patient has cancer:

$$\frac{Tp}{Tp+Fn} = \frac{\text{p(Classifier = YES, Cancer = YES)}}{\text{p(Cancer = YES)}} = \text{p(Classifier = YES | Cancer = YES)}$$

- Precision is the probability that a randomly selected patient from all patients with a positive test indeed has cancer:

$$\frac{Tp}{Tp+Fp} = \text{p(Cancer = YES | Classifier = YES)}$$

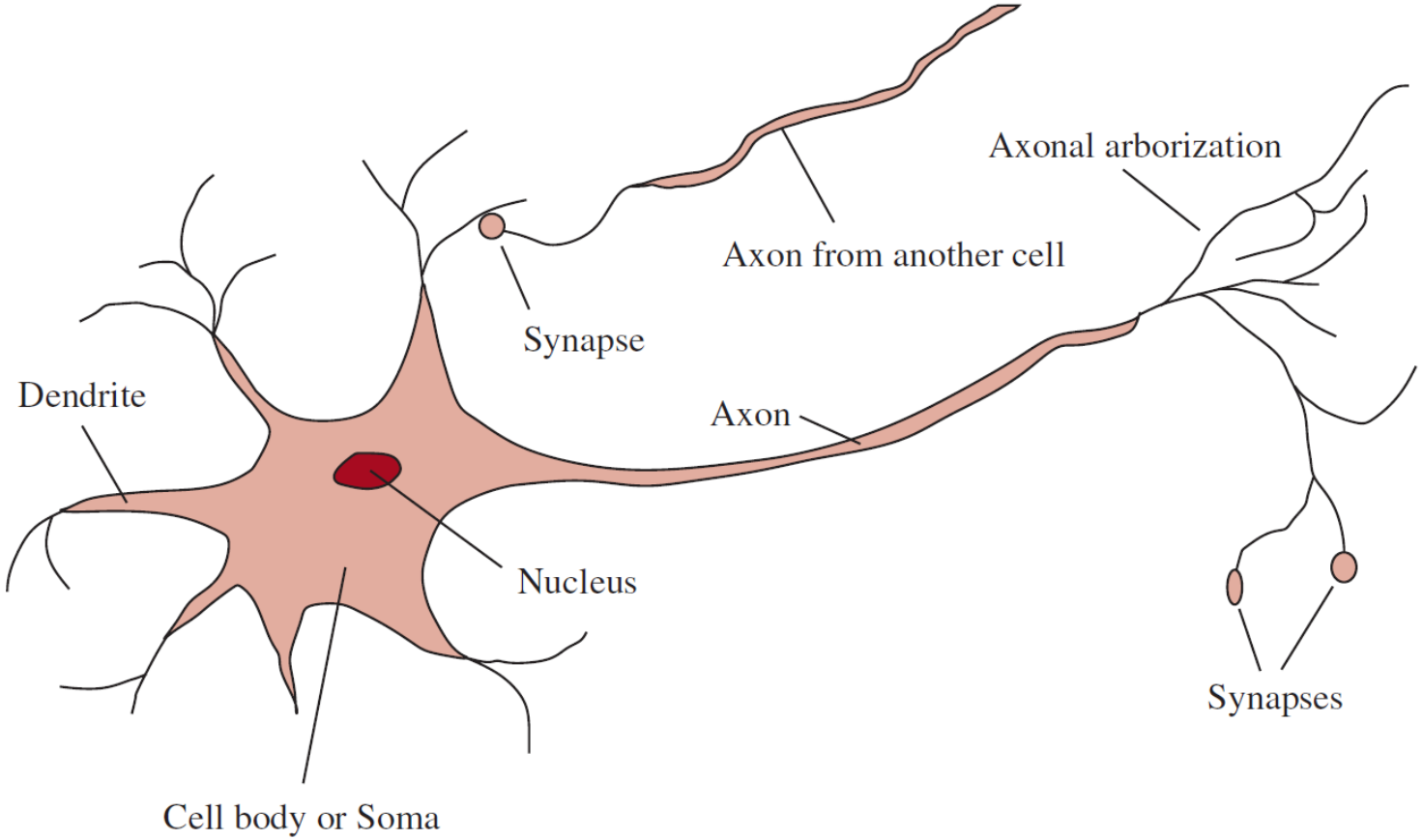
- Relation to statistical errors:
  - Type I error: $1 - specificity$ Type II error: $1 - recall$

UNIVERSITÄT DES SAARLANDES

Neural Networks

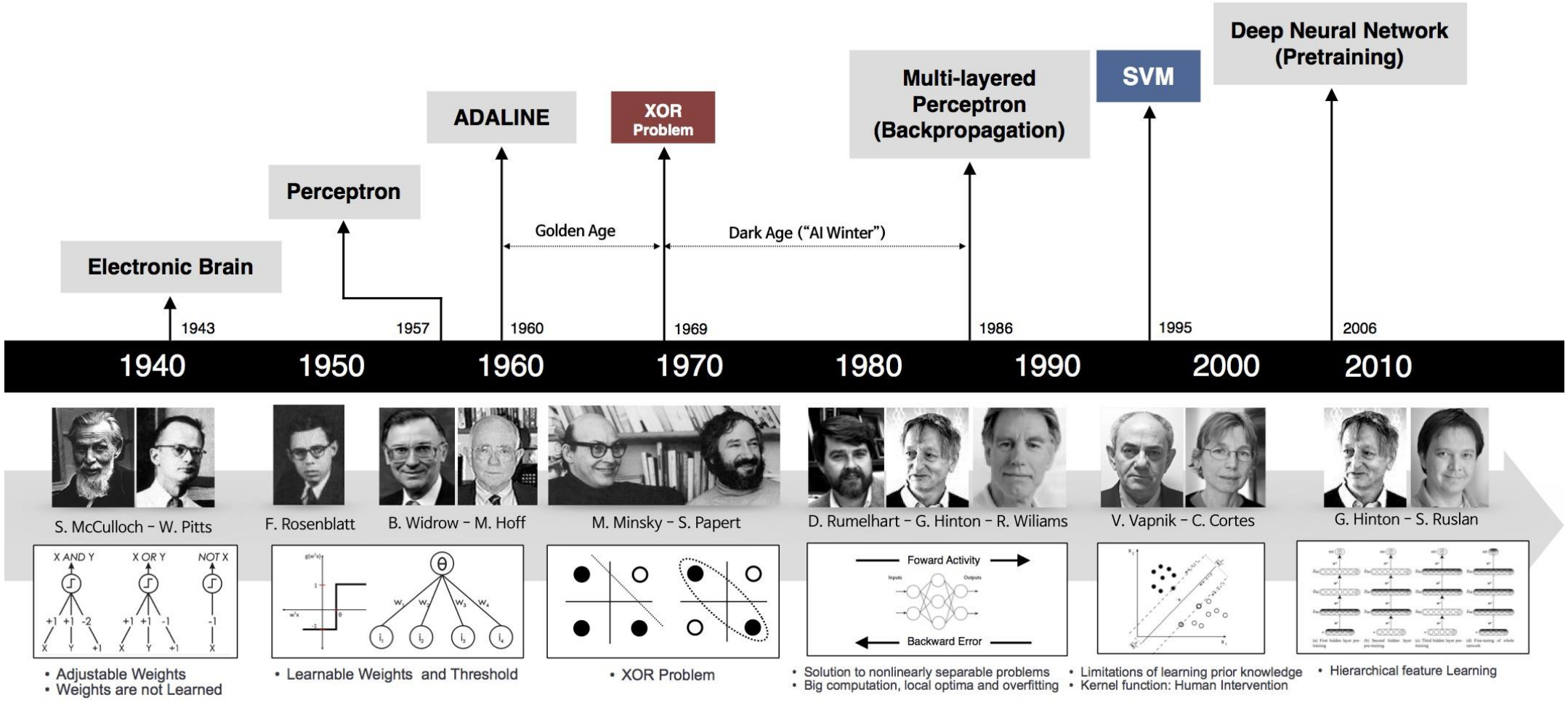
A Neuron in the Human Brain

- 10^{11} neurons of over 20 types
- A neuron can connect to 10-10^5 other neurons via synapses
- Signals are noisy “spike trains" of electrical potential
- A key property of the human brain is neural plasticity
  - The neural network can reorganize its connections



Russell/Norvig, 4th edition, Fig. 1.1

Neural Networks Research

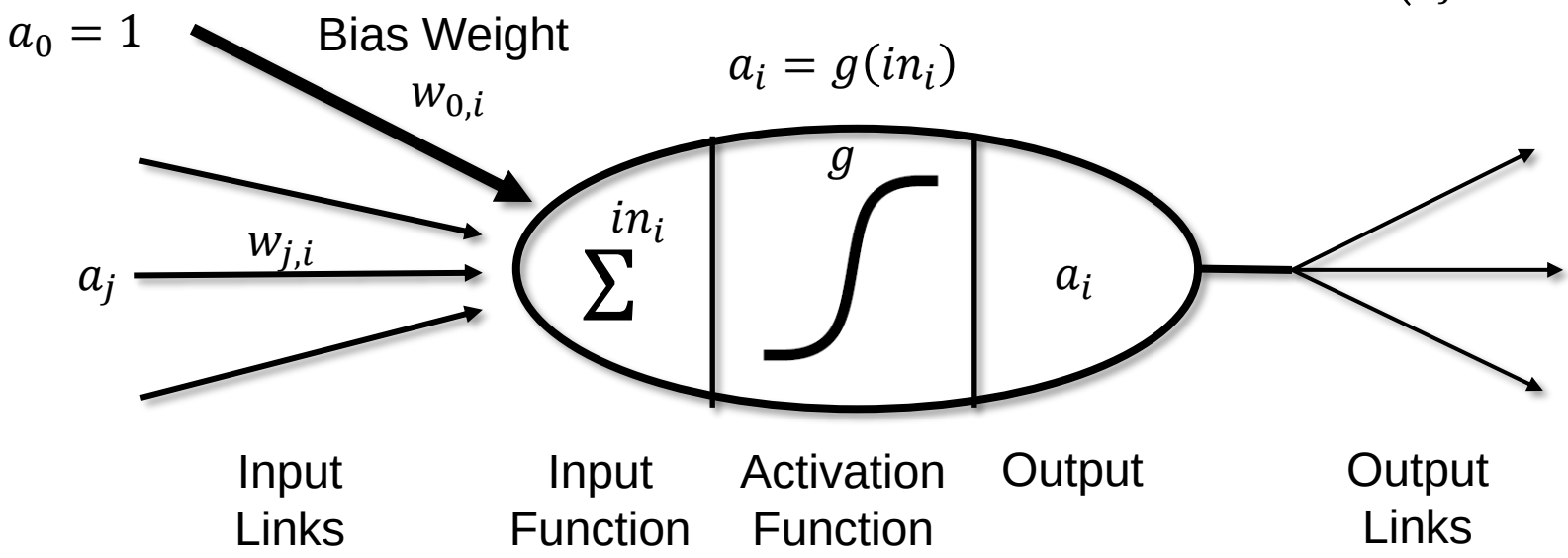


https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html
(accessed in 2017, picture no longer available)

McCulloch-Pitts "Unit" Model of a Neuron

- Neuron fires when a linear combination of its inputs exceeds predefined threshold

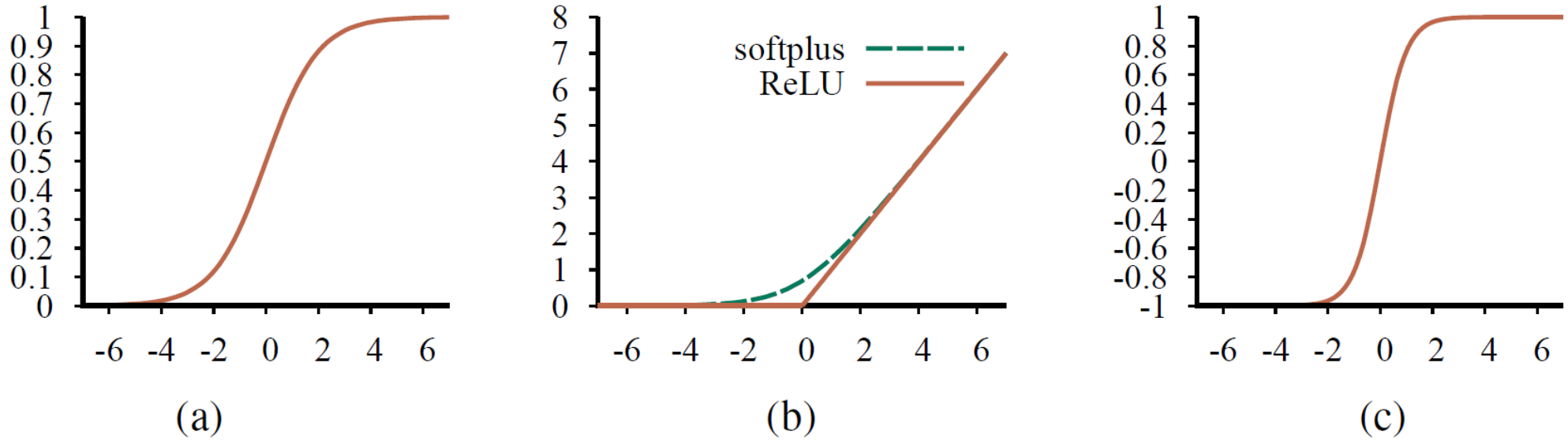
$$a_i = g(in_i) = g\left(\sum_j w_{j,i} a_j\right)$$



- The bias weight (multiplied with input 1.0) is a constant to shift the result of activation function towards the positive or negative side (offset the result)

Activation Functions commonly used in Neural Networks

Russell/Norvig, 4th edition, Figure 21.2



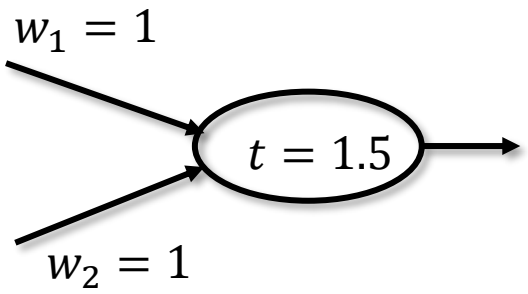
(a) logistic or sigmoid function $1/(1+e^{-x})$

(b) ReLU function $\max(0,x)$ and softplus function $\ln(1+e^{x})$ (smooth approximation of ReLU function to constrain output to always be positive, derivative is the logistic function)

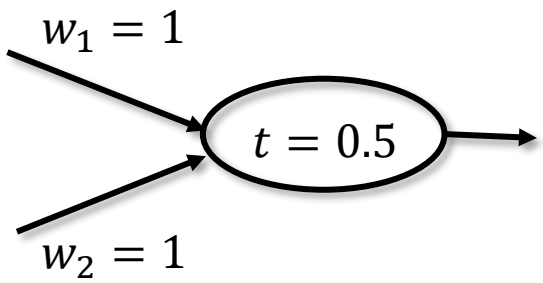
c) the tanh function $\tanh x = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

Neural Networks can implement Boolean Functions

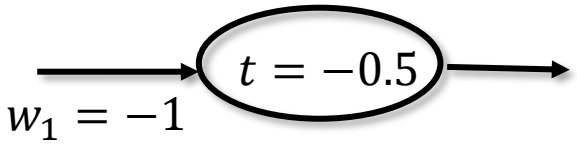
- Using a step function with step value t (the threshold value), the output 1 or 0 is produced depending on whether the weighted input sum is higher/lower than t



AND



OR



NOT

$weighted\ input$:

$1 \cdot 1 + 1 \cdot 1 = 2$

$1 \cdot 1 + 1 \cdot 0 = 1$

$Output\ depending\ on\ threshold$:

$2 > 1.5 \Rightarrow 1$

$1 < 1.5 \Rightarrow 0$

$1 \cdot 0 + 1 \cdot 0 \Rightarrow 0$

$1 \cdot 0 + 1 \cdot 1 \Rightarrow 1$

$-1 \cdot 0 \Rightarrow 1$

$-1 \cdot 1 \Rightarrow 0$

Network Structures

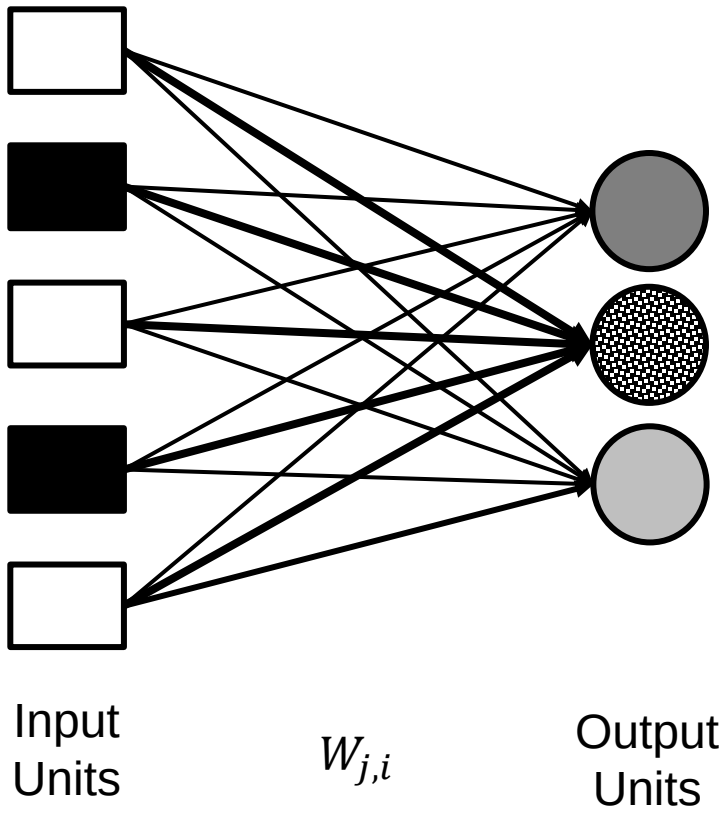
- Feed-forward networks are directed acyclic graphs, they have no internal state other than the weights
  - Single-layer perceptron
  - Multi-layer perceptron

- Recurrent networks:
  - Activation is fed back to causing units (cyclic graphs)
  - Have internal state (short term memory), e.g., LSTM networks
  - Hopfield nets use *bidirectional* connections with *symmetric* weights $w_{i,j} = w_{j,i}$

Single-Layer Perceptron (Rosenblatt 1957)

- Inputs connected directly to the outputs
  - Output units all operate separately: m outputs = m separate networks

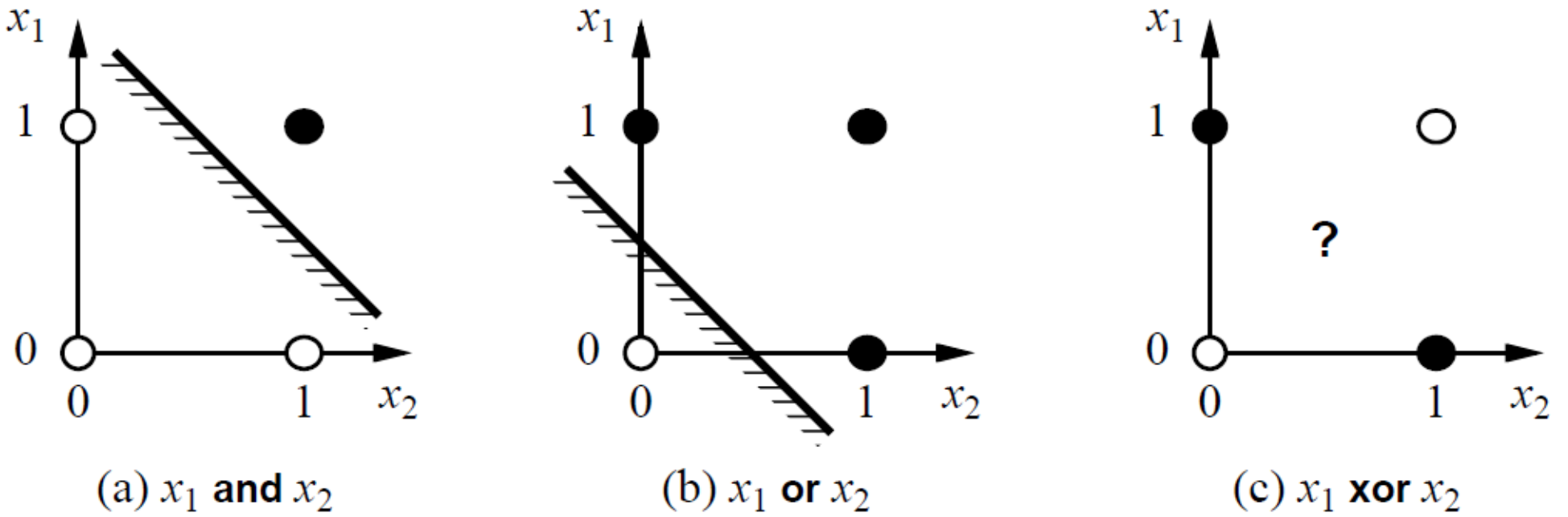
- Minsky and Papert showed in 1969 that perceptrons can only learn linear separable functions
  - M. Minsky, S.Papert: Perceptrons: An Introduction to Computational Geometry MIT Press 1969



Linear Separable Functions

$$\sum_j w_j x_j > t \text{ or } W \cdot x > t$$

- Consider a perceptron with a step function
  - Can represent AND, OR, NOT, majority, etc., but not XOR
  - Represents a linear separator in the input space
  - Only a small fraction of all Boolean functions is linearly separable

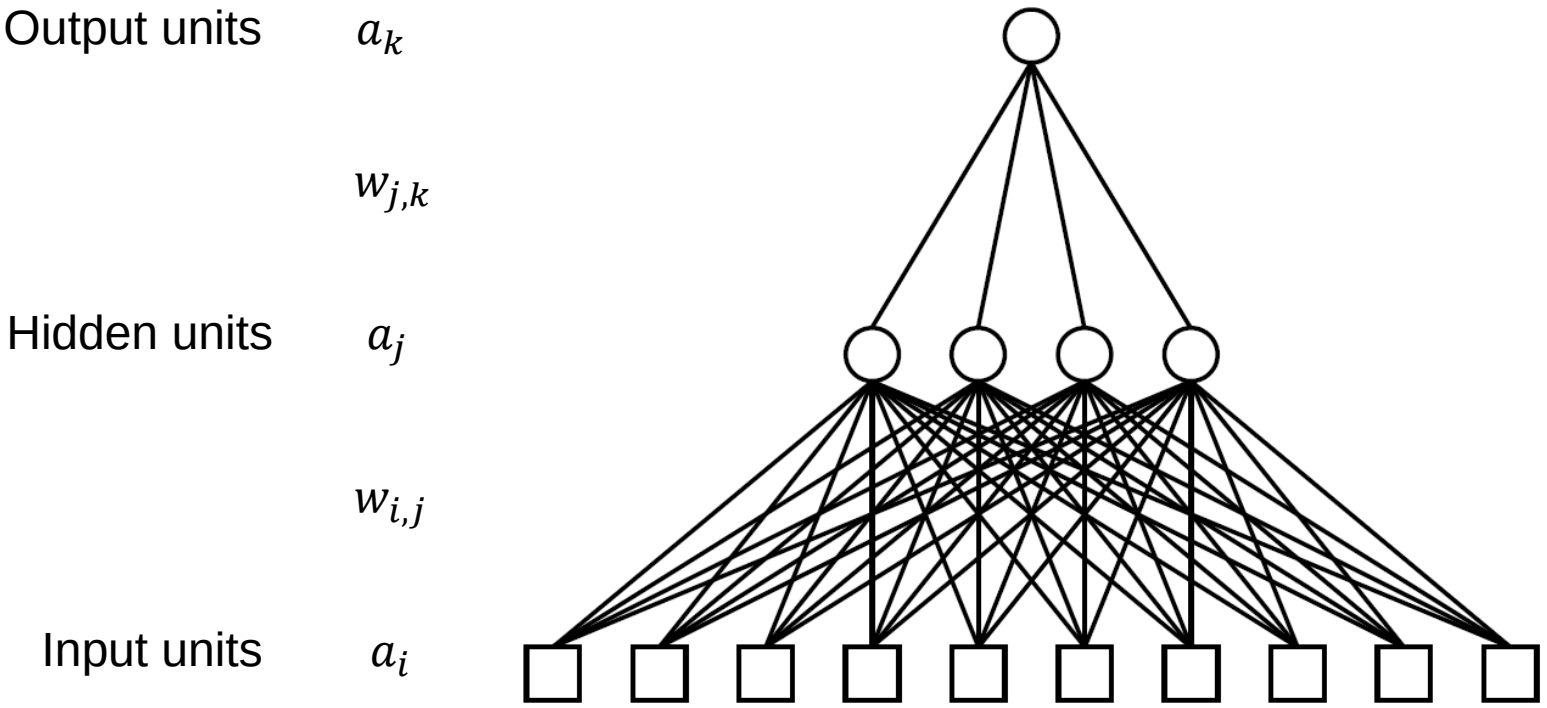


(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Russell/Norvig, Figure 18.21

Multi-Layer Perceptron / Feed-forward Network

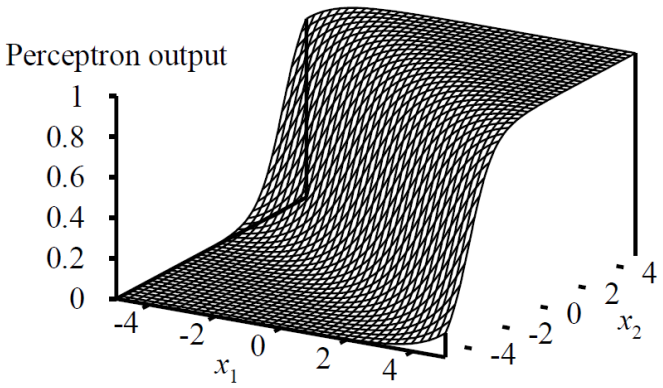
- Input and output layers are usually fully connected via at least one hidden layer
- Network architecture (number of hidden layers/edges typically chosen by hand)



Expressiveness of Feed-Forward Networks with Hidden Layers

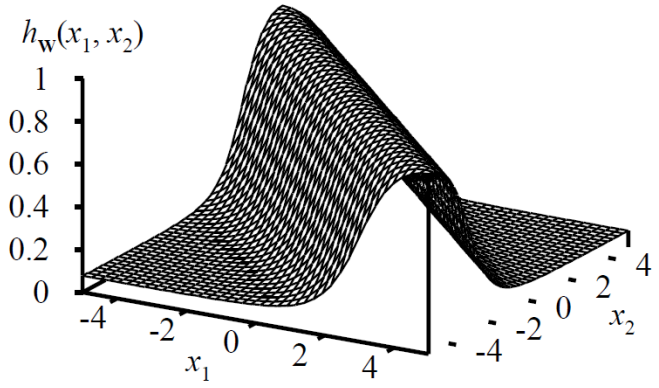
- Feed-Forward network with a single, sufficiently large hidden layer can represent any continuous function of the inputs with arbitrary accuracy
- Feed-Forward with two hidden layers can be represent discontinuous functions

Perceptron

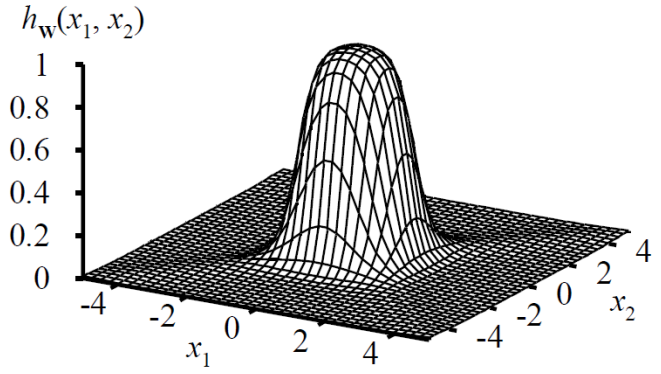


Adjusting weights moves location, orientation, and steepness of the linear separator

Feed-Forward Networks with Hidden Layers



Result of combining two opposite-facing soft threshold functions to produce a ridge

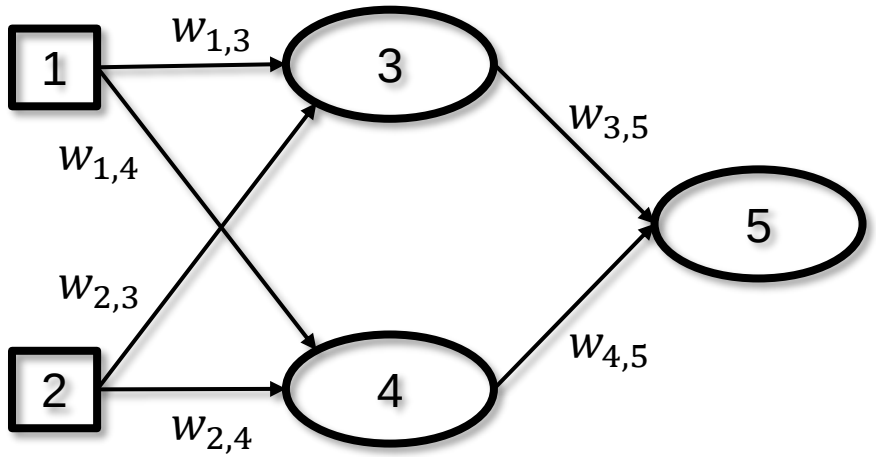


Result of combining two ridges to produce a bump

Russell/Norvig,, Figures 18.17 and 18.23

Learning in Feed-forward Network

- Feed-forward network is a parameterized family of nonlinear functions
- Adjusting weights changes the function: do learning this way!



$$a_5 = g\left(w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4\right)$$
$$= g\left(w_{3,5} \cdot g\left(w_{1,3} \cdot a_1 + w_{2,3} \cdot a_2\right) + w_{4,5} \cdot g\left(w_{1,4} \cdot a_1 + w_{2,4} \cdot a_2\right)\right)$$

Idea of Backpropagation Learning for Multi-Layer Networks

- Output expressed as a function of the inputs and the weights
- When the derivative of this function can be calculated, we can use the gradient-descent loss-minimization method to train the network
- Learning = minimizing the squared error „cost“ function by performing optimization search using gradient descent
  - Split error across hidden units: unit j is responsible for some fraction of the error in each of the output units to which it connects
  - Error is divided according to the strength of the connection between a hidden unit and the output unit
  - Continue to propagate the error to the previous layer and adjust weights between two layers until input layer is reached

Derivation of the Error and the Correcting Error Term

The squared error on a single example with expected output y_k and current output a_k is defined as

$$E = \frac{1}{2}\sum_k (y_k - a_k)^2 \qquad a_k = g(in_k) = g\left(\sum_j w_{j,k} a_j\right)$$

where the sum is over the k units in the output layer

Then, the partial derivative of the error wrt. a specific weight using the chain rule yields

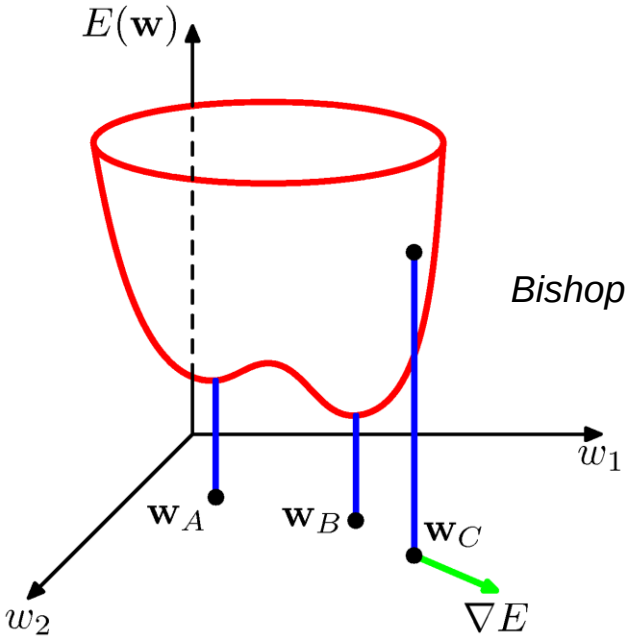
$$\frac{\delta E}{\delta w_{j,k}} = \overbrace{-(y_k - a_k)}^{\text{outer derivative}} \overbrace{\frac{\delta a_k}{\delta w_{j,k}}}^{\text{inner derivative}} = -(y_k - a_k)\frac{\delta g(in_k)}{\delta w_{j,k}} = -(y_k - a_k)g'(in_k)\cdot\frac{\delta in_k}{\delta w_{j,k}}$$

$$= -(y_k - a_k)g'(in_k)\cdot\frac{\delta}{\delta w_{j,k}}\left(\sum_j w_{j,k} a_j\right) = -\underbrace{(y_k - a_k)g'(in_k)}_{\Delta_k}\cdot a_j = -\Delta_k a_j$$

error correcting term = part of derivative

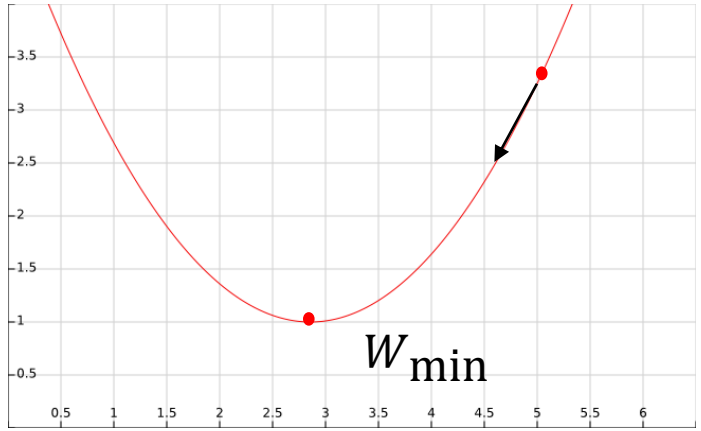
Illustration of Steepest Gradient Descent

- The error is minimized by moving the weights into the opposite direction of the derivative of the squared error
- Backpropagation is an iterative numerical process as there is no analytical solution to the equation $\nabla E(\vec{w}) = 0$



Bishop Figure 5.5

$$\frac{\partial E}{\partial W} = 0 \text{ at } W_{\min}$$



∇E= Vector of derivatives with respect to weights w_1 and w_2

Backpropagation Learning by Adjusting Weights

- Adjust weights of edges leading to unit k in output layer:

$$w_{j,k} \longleftarrow w_{j,k} + \alpha \times a_j \times \Delta_k$$

α - learning rate, a small constant, e.g., 0.02
g' - derivative of activation function g

where $\Delta_k = g'(in_k)(y_k - a_k)$

- Adjust weights of edges leading to unit j in hidden layer:

$$w_{i,j} \longleftarrow w_{i,j} + \alpha \times a_i \times \Delta_j$$

where $\Delta_j = g'(in_j) \sum_k w_{j,k} \Delta_k$

UNIVERSITÄT DES SAARLANDES

The Backpropagation Algorithm

- The book by Calin 2020 contains a comprehensive overview over modern training algorithms using for example, local and stochastic search to find an error-minimizing weight vector

function BACK-PROP-LEARNING(*examples*,*network*) **returns** a neural network
inputs:
examples a set of examples, each with input vector x and output vector y;
network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node
while some stopping criterion is not satisfied **do**
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example(x, y) **in** *examples* **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $l = 2$ **to** L **do**
      for each node j in layer l **do**
        $in_j \leftarrow \sum_i w_{i,j} a_i$
        $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer*/
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \cdot (y_j - a_j)$
    for $l = L - 1$ **to** 1 **do**
      for each node i in layer l **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
    /* *Update every weight in network using deltas* */
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow w_{i,j} + \alpha \cdot a_i \cdot \Delta[j]$
return *network*

Russell/Norvig,, Figure 18.24

Termination of the Learning Process in a Neural Network

- Backpropagation searches for a weight vector $\vec{w}$ such that $E(\vec{w})$ takes its smallest value
- The error function usually has a highly nonlinear dependence on the weights and the bias parameters and there will be man points in the weight space at which the gradient vanishes or is numerically very small
  - 2-layer network with m hidden units: each point in weight space is a member of $m!\,2^m$ equivalent points
  - Multiple inequivalent minima exist
- For a successful application using neural networks it might not be necessary to find the global minimum, but to compare several local minima from different training processes

Neural Network Training - An Intuitive Example

Take a Right Turn?

- Traffic Light is Red & Green Arrow exists: Yes
- Traffic Light is Green: Yes
- Traffic Light is Red & No Green Arrow: No

https://www.sueddeutsche.de/auto/abbiegen-rot-radfahrer-gruenpfeil-1.4280175

- Training Example:

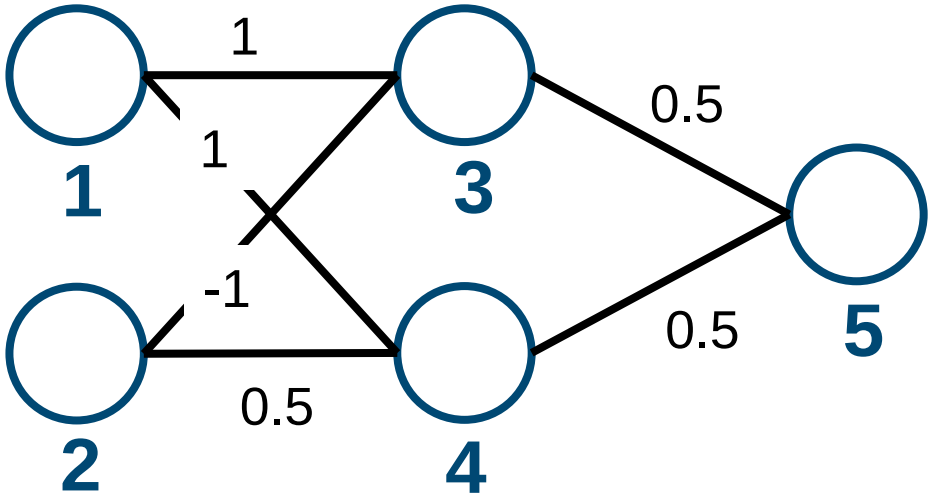
Traffic Light is Red
Green Arrow

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \longrightarrow (0)$$

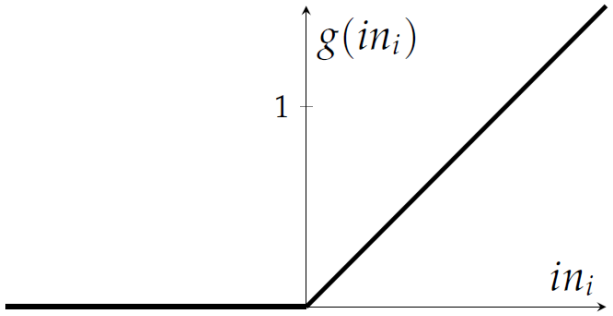
Take a Right Turn? NO

Example Neural Network Learning

Network structure



g-function



Derivative of the ReLu function $\max(0, x)$

$$ReLu'(x) = 0 \text{ for } x \leq 0$$
$$ReLu'(x) = 1 \text{ for } x > 0$$

Note that when $x = 0$, the derivative does not exist, but one can define a value, common practice 0, 0.5, or 1.

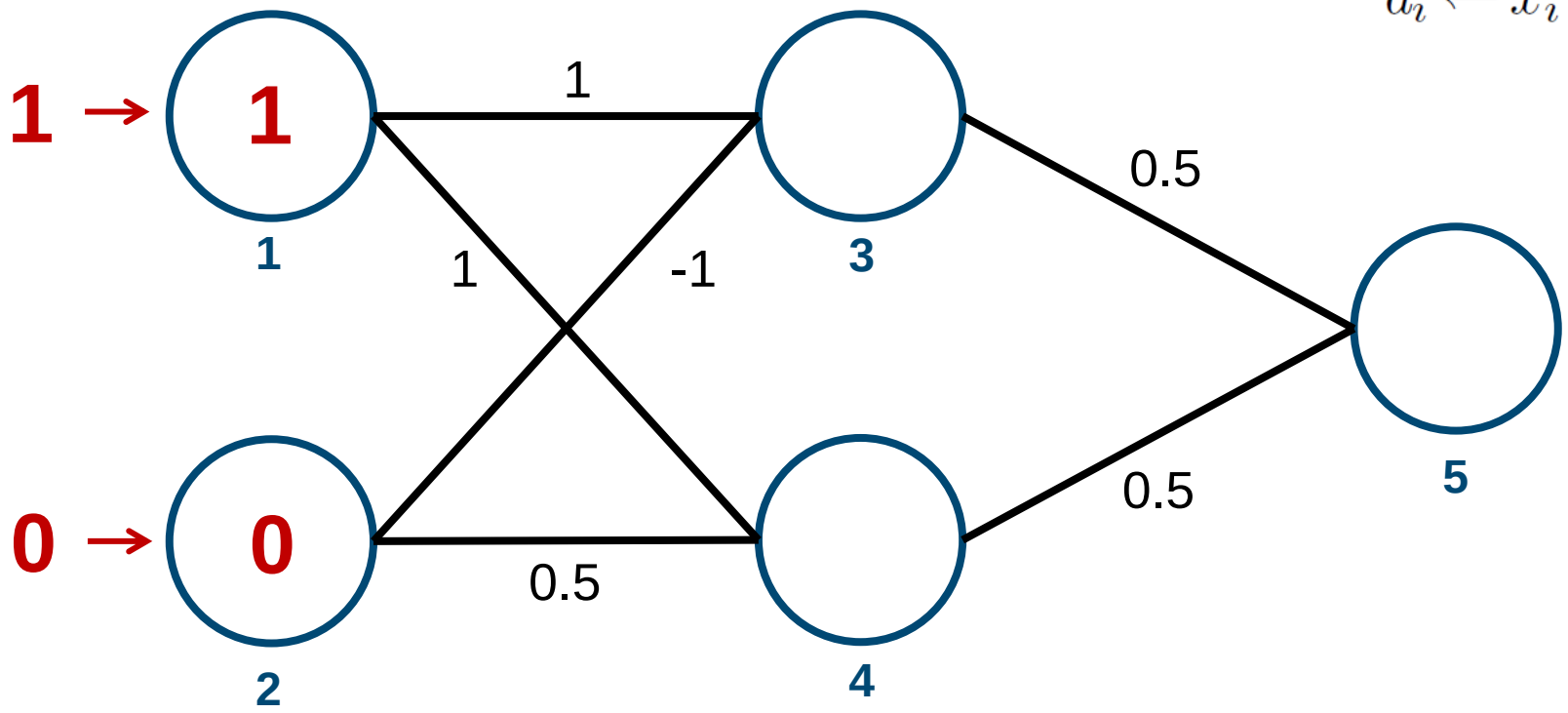
Learning rate: α = 1

Example Instance: input vector = (1,0)
expected output value = 0

Initialization of Input Layer with Training Example Vector

for each node i in the input layer **do**

$a_i \leftarrow x_i$

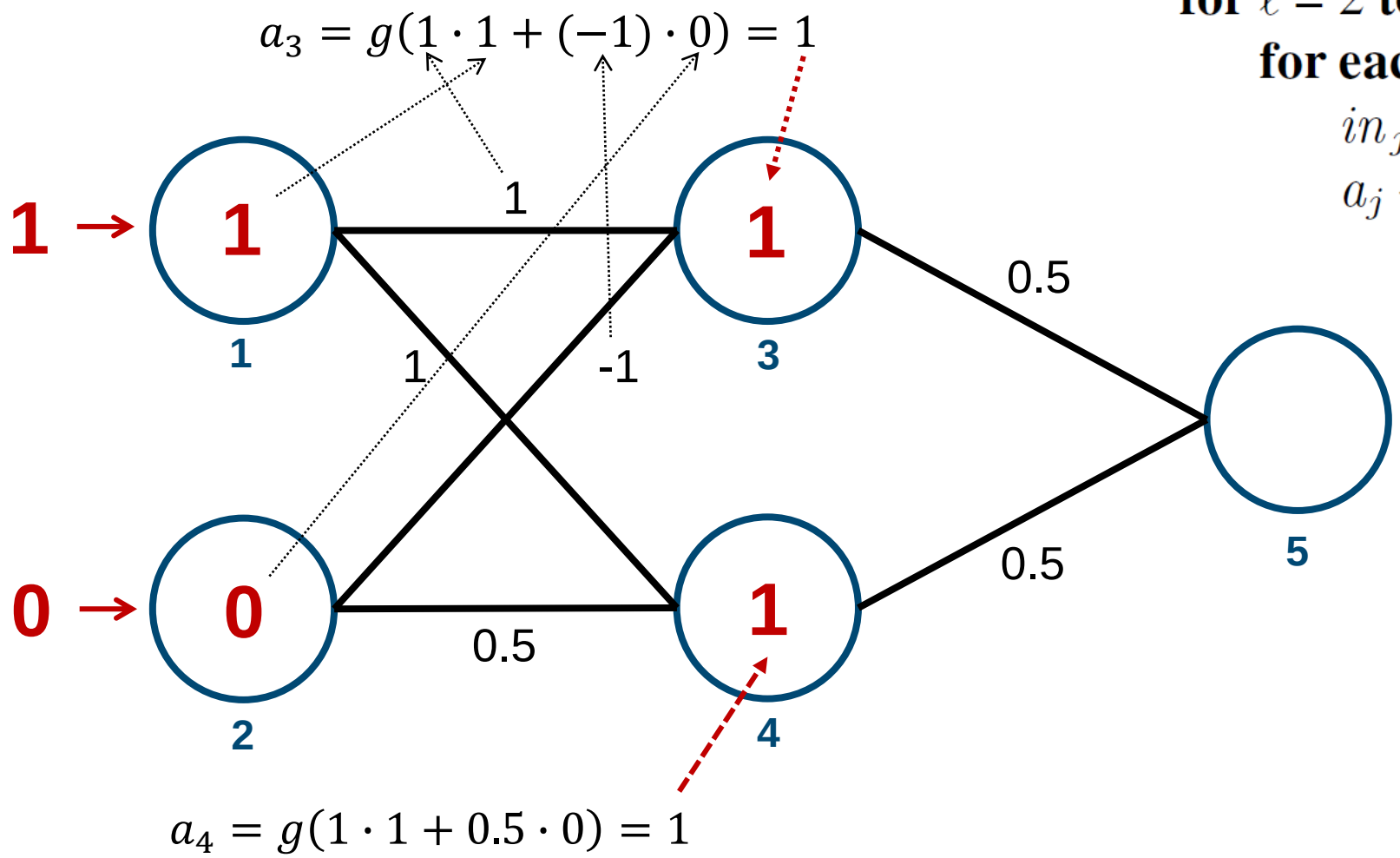


Expected output: 0

Experiment with computations in neural networks:
https://playground.tensorflow.org

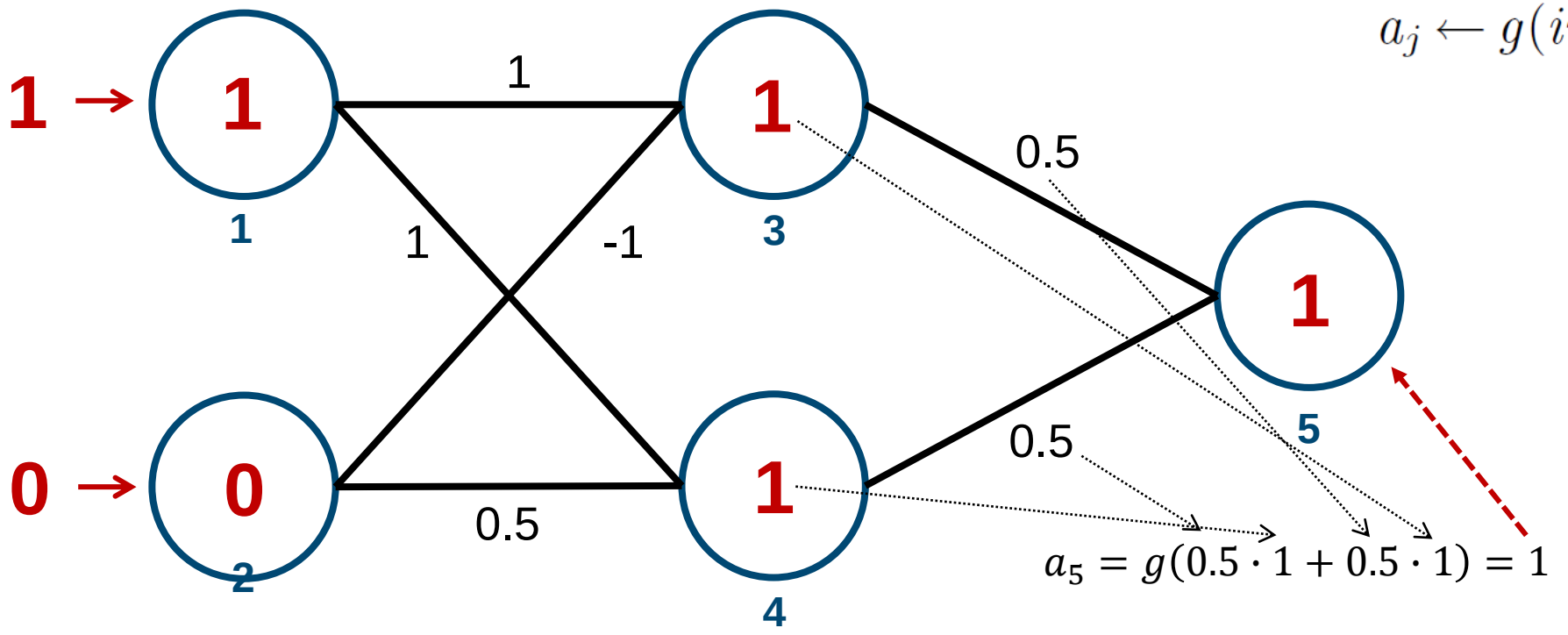
Forward Propagation of Inputs to Hidden Layer

for $\ell = 2$ **to** L **do**
  for each node j in layer ℓ **do**
    $in_j \leftarrow \sum_i w_{i,j} a_i$
    $a_j \leftarrow g(in_j)$



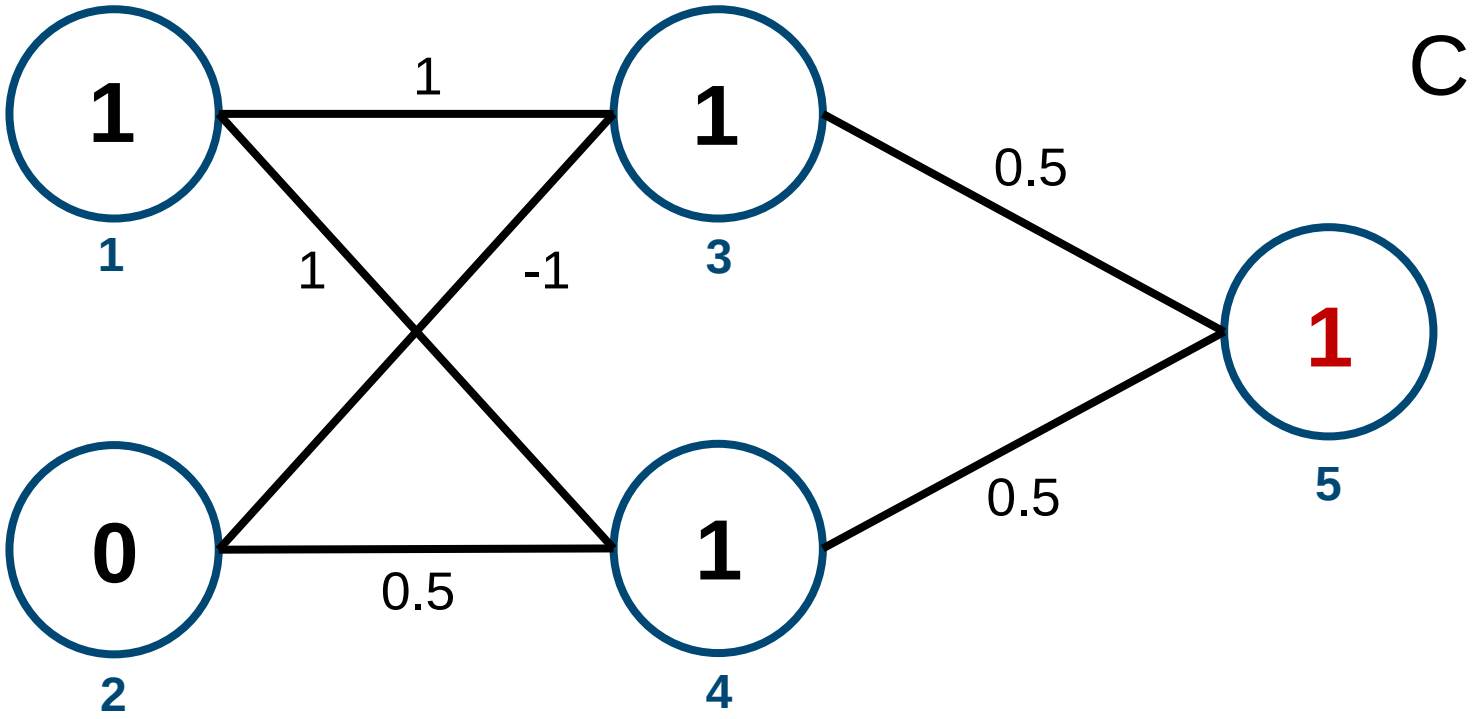
Forward Propagation of Values from Hidden Layer to Output Layer

for $\ell = 2$ **to** L **do**
  for each node j in layer ℓ **do**
    $in_j \leftarrow \sum_i w_{i,j} a_i$
    $a_j \leftarrow g(in_j)$



UNIVERSITÄT DES SAARLANDES

Observing the Error at the Output Unit



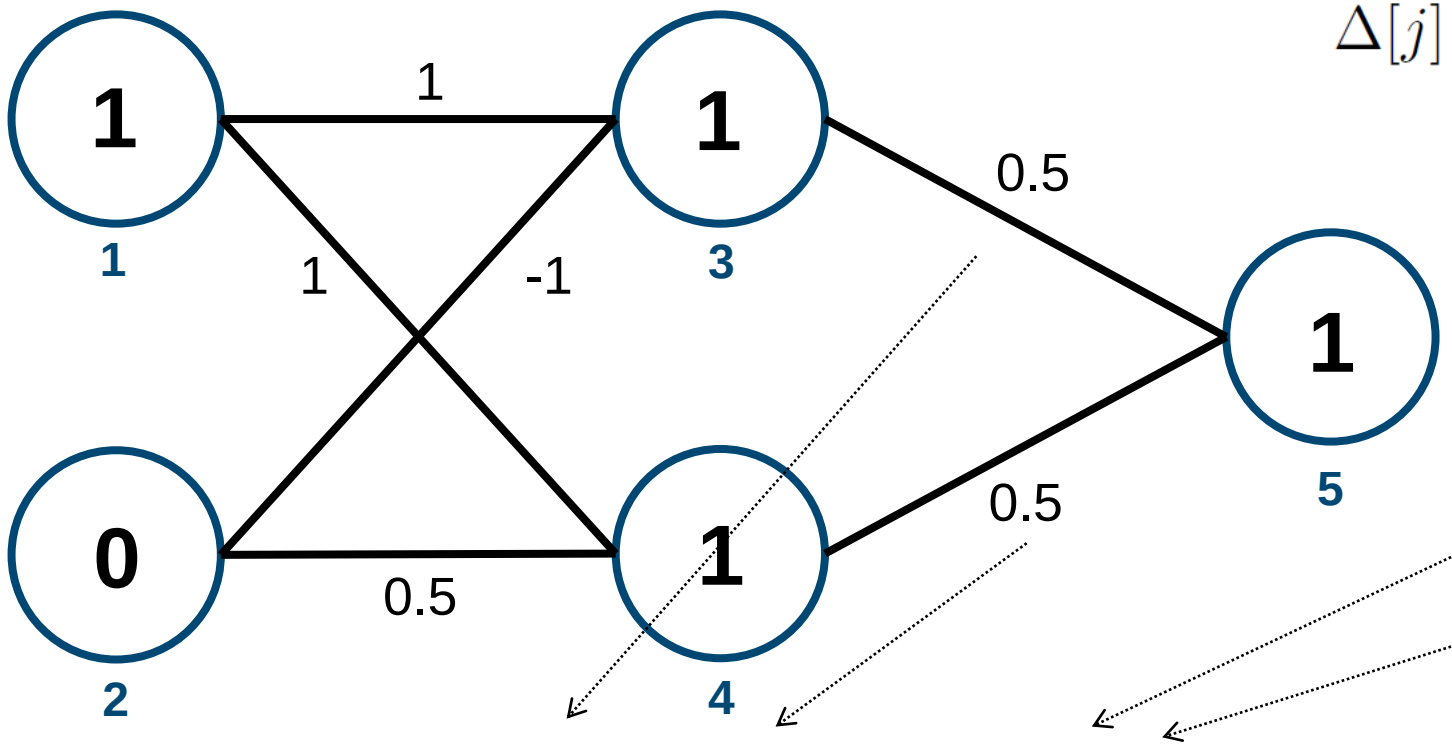
Expected output: 0

Computed output: 1

Computation of Error Correcting Term Delta in Output Layer

for each node j in the output layer **do**

$$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$$



expected output value

computed output value

$ReLu'(x) = 0$ for $x \leq 0$

$ReLu'(x) = 1$ for $x > 0$

$$\Delta[5] = g'(0.5 \cdot 1 + 0.5 \cdot 1) \cdot (0\text{-}1)$$
$$= g'(1) \cdot -1$$
$$= -1$$

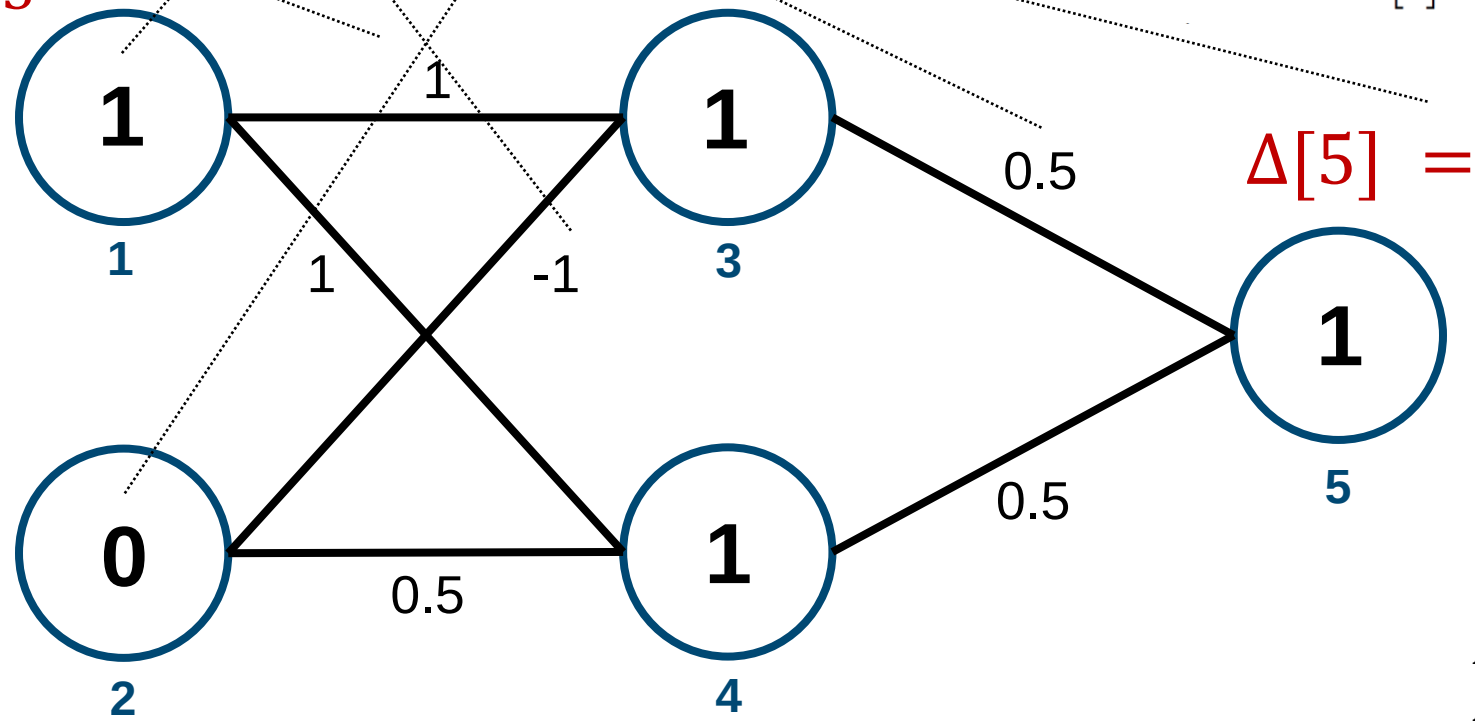
Backpropagation and Distribution of Delta Value to Units in Hidden Layer

$g'(in_3) = g'(1) = 1$

$\Delta[3] = g'(1 \cdot 1 + (-1) \cdot 0) \cdot (0.5 \cdot (-1)$
$= -0.5$

for $\ell = L - 1$ **to** 1 **do**
  for each node i in layer ℓ **do**
    $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$

$\Delta[5] = -1$



$\Delta[4] = g'(1 \cdot 1 + 0.5 \cdot 0) \cdot (0.5 + (-1))$
$= -0.5$

$ReLu'(x) = 0$ for $x \leq 0$
$ReLu'(x) = 1$ for $x > 0$

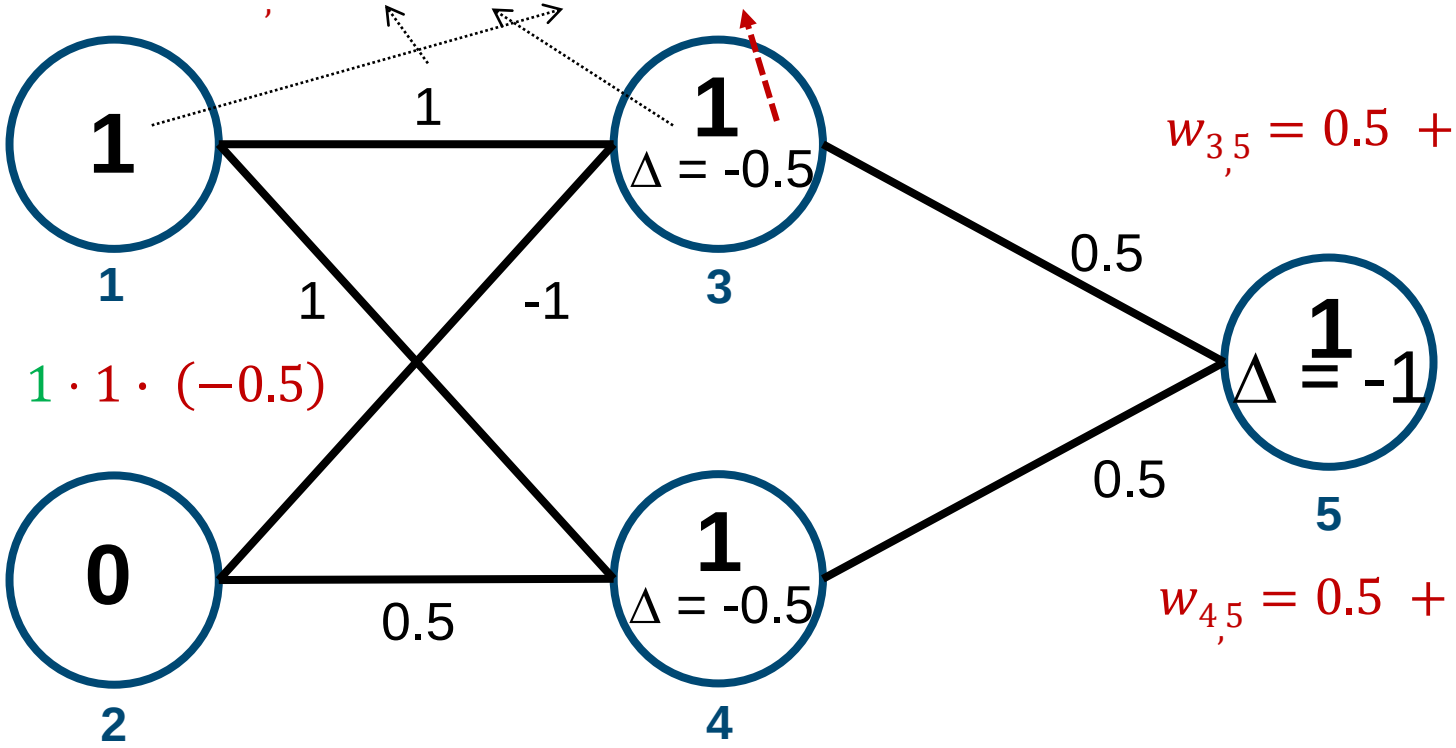
Update Weights

for each weight $w_{i,j}$ in *network* **do**

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$$

$\alpha = 1$

$w_{1,3} = 1 + 1 \cdot 1 \cdot (-0.5) = 0.5$



$w_{3,5} = 0.5 + 1 \cdot 1 \cdot (-1) = -0.5$

$w_{1,4} = 1 + 1 \cdot 1 \cdot (-0.5) = 0.5$

$w_{4,5} = 0.5 + 1 \cdot 1 \cdot (-1) = -0.5$

$w_{2,4} = 0.5 + 1 \cdot 0 \cdot (-0.5) = 0.5$

$w_{2,3} = -1 + 1 \cdot 0 \cdot (-0.5) = -1$

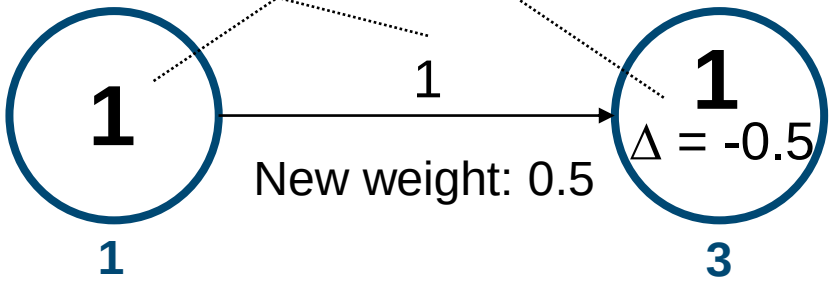
Impact of Learning Rate on Weight Update

for each weight $w_{i,j}$ in $network$ **do**

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$$

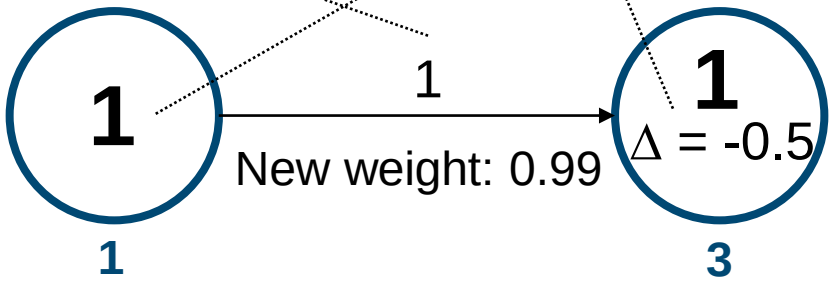
$\alpha = 1$

$w_{1,3} = 1 + 1 \cdot 1 \cdot (-0.5) = 1 + (-0.5) = 0.5$



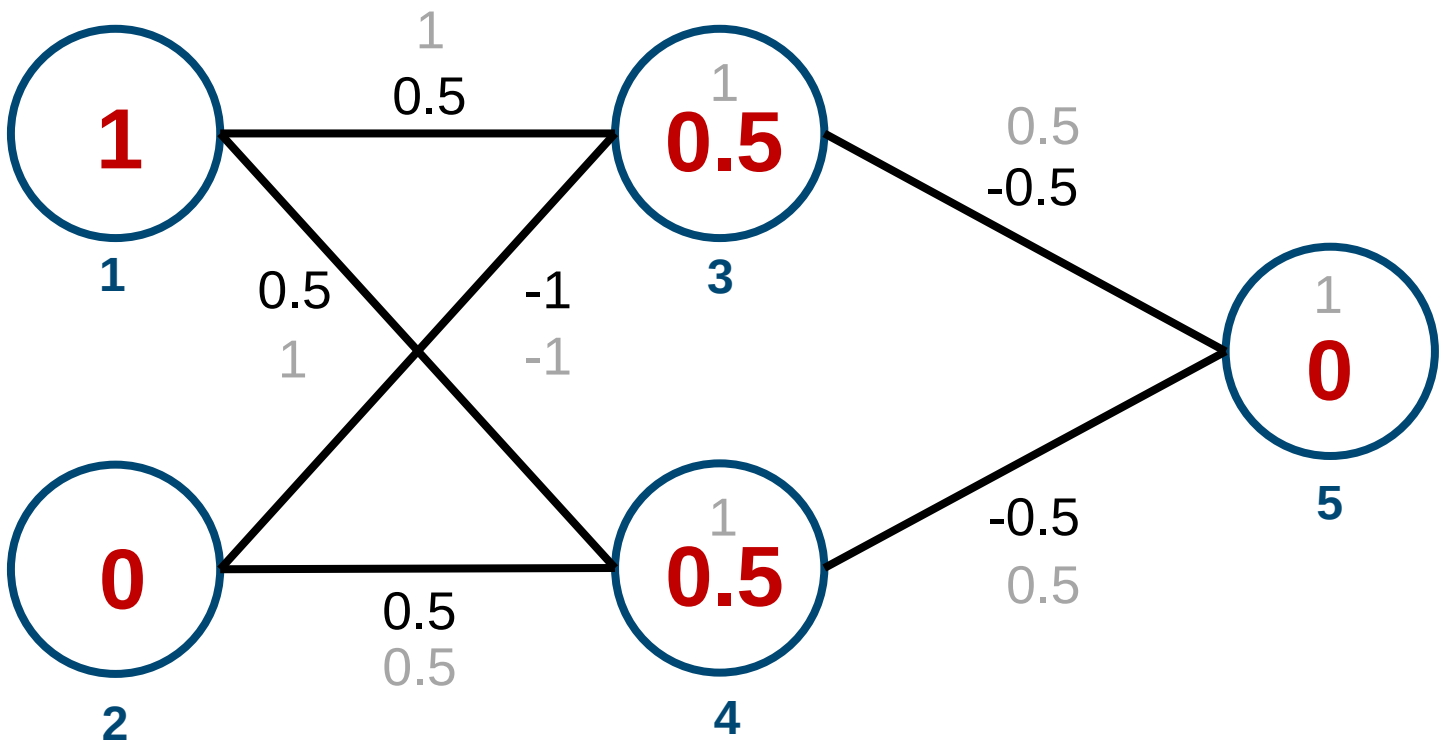
$\alpha = 0.02$

$w_{1,3} = 1 + 0.02 \cdot 1 \cdot (-0.5) = 1 + (-0.01) = 0.99$



Network with Updated Weights

Old values in grey for comparison

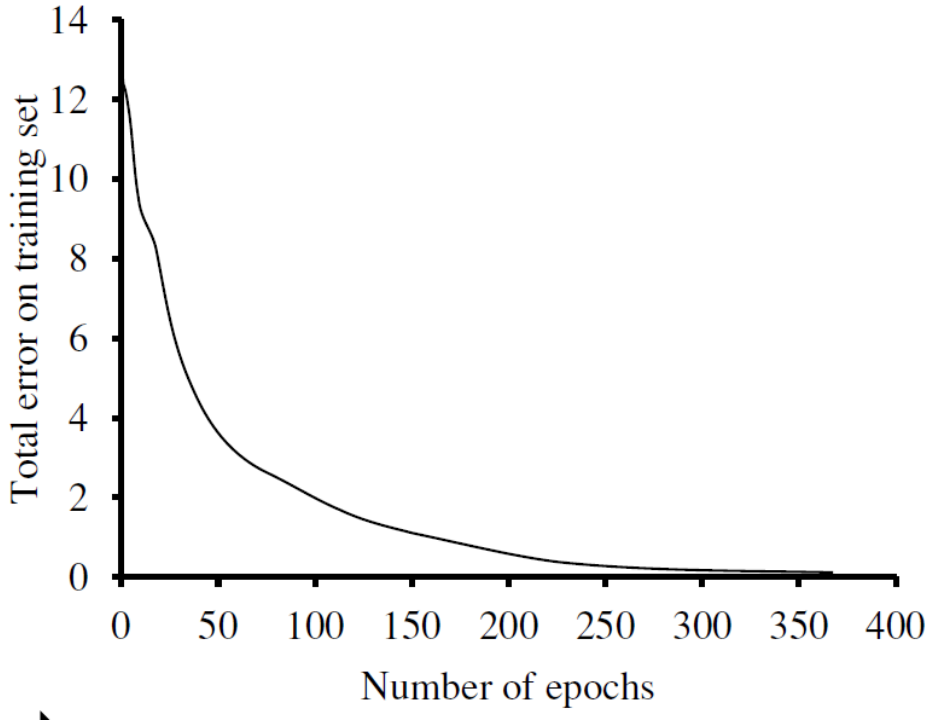


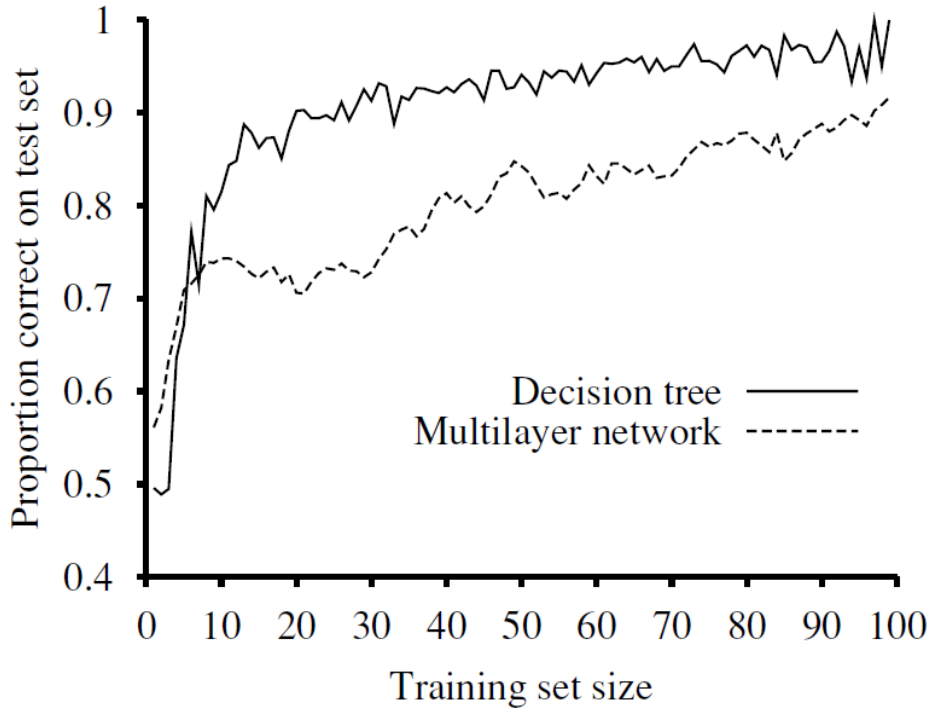
weighted input sum is -0.25, ReLu returns 0 for a negative input expected output 0 is now computed on this training example

- Same input into the new network → Squared error is reduced to 0
  - The network has learned to recognize the example correctly
- **Squared Error:** $\frac{1}{2}(\mathbf{0} - \boldsymbol{a_5})^2$**:** old network = $\frac{1}{2}(0-1)^2 = 0.5$ new network = $\frac{1}{2}(0-(0))^2 = 0$

Comparison Multi-Layer Network - Decision Tree in the Restaurant Domain

- Training curve on the left shows the gradual reduction in error by backpropagation over several epochs (number of complete passes through the training set)
- Comparative learning curves on the right that decision-tree learning does slightly better on the restaurant problem than backpropagation in a multi-layer network



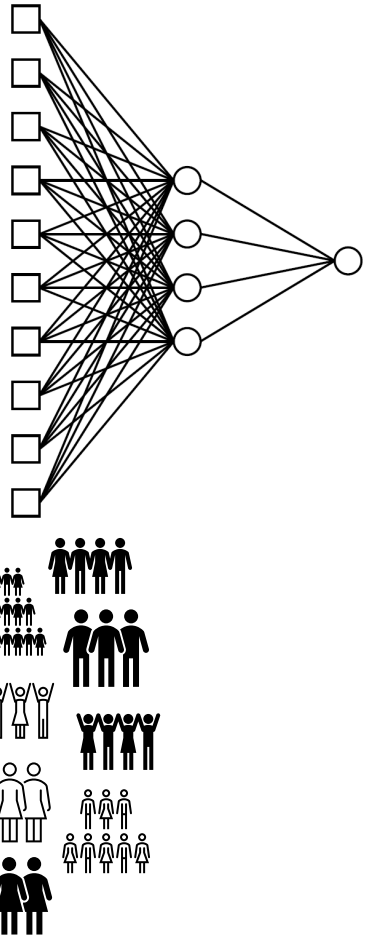


Russell/Norvig,, Figure 18.25

UNIVERSITÄT DES SAARLANDES

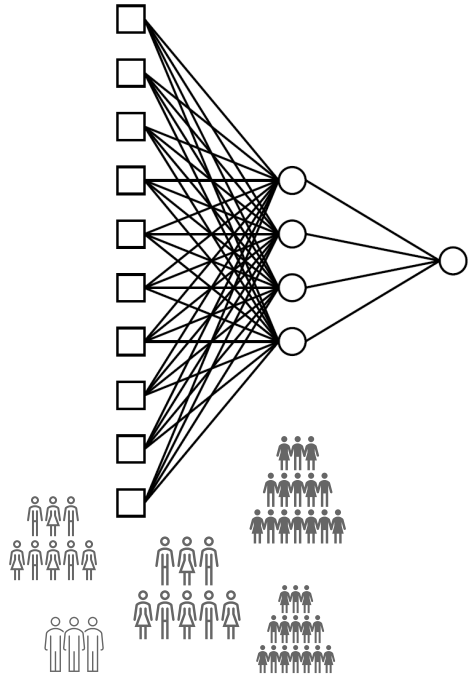
Risks and Challenges of Machine Learning Applications

Review of the Machine Learning Workflow to build Applications



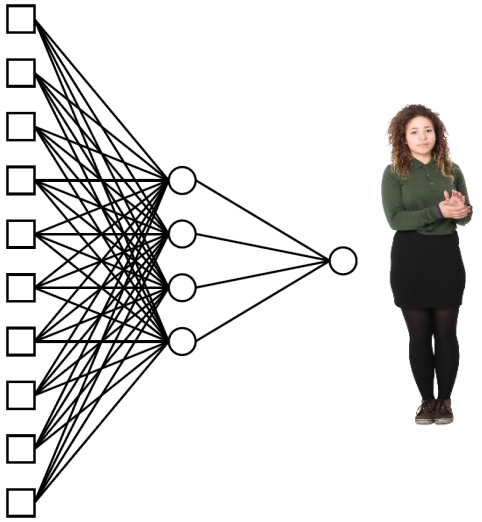
Train on Training Set

- Cases of measles and non-measles
- Extract statistical pattern



Test on Test Set

- Assess prediction quality of statistical pattern



Apply to single concrete case

- Predict membership of this case in trained class with certain confidence
- “Lilly has measles with 99 % confidence”

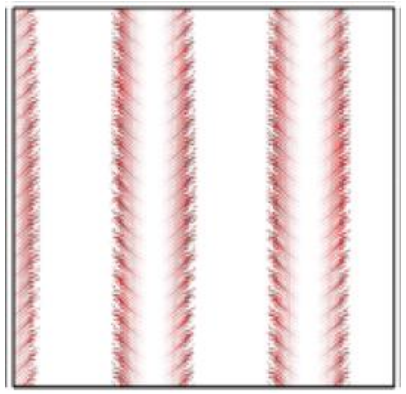
There is no solid mathematical basis that supports the correctness of the prediction on a single case
CORRELATION ≠ CAUSATION

A Statistical Model is NOT a CAUSAL Model

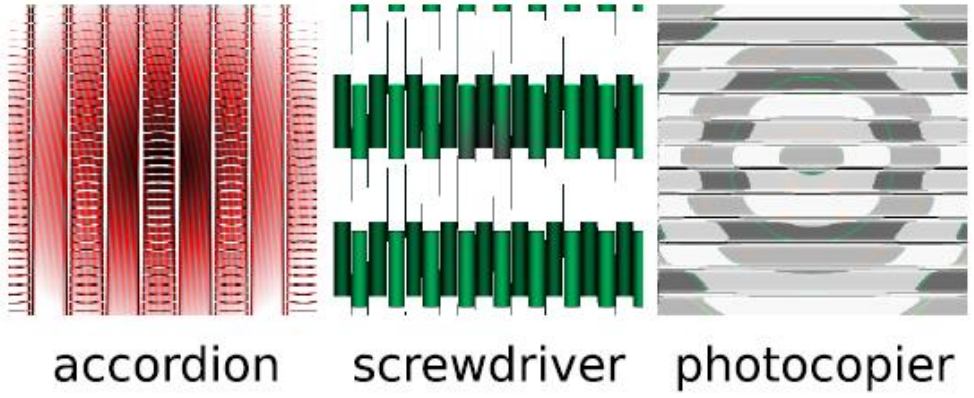
- Machine learning algorithms extract statistical patterns, which are valid for a large population of specific examples
- There is no guarantee that the statistical pattern represents a causal model for the application domain and the question to be answered.
- Furthermore, the learned model is often inaccessible to human understanding
  - In neural networks, the model is described by the network architecture and the weight vector
  - Its semantic interpretation in terms of causality is unknown and an open question for AI research
- This means when we apply these models to a single case, the prediction may be correct or wrong independently of the confidence value a machine learning algorithm might have returned (the error on the single case can be high even if it is low on a larger population)

Wrong Predictions with High Confidence

- Neural networks can make predictions with high confidence on totally meaningless input patterns
  - The patterns were generated from real object pictures using genetic algorithms
- Nguyen, Anh, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 427-436. 2015.

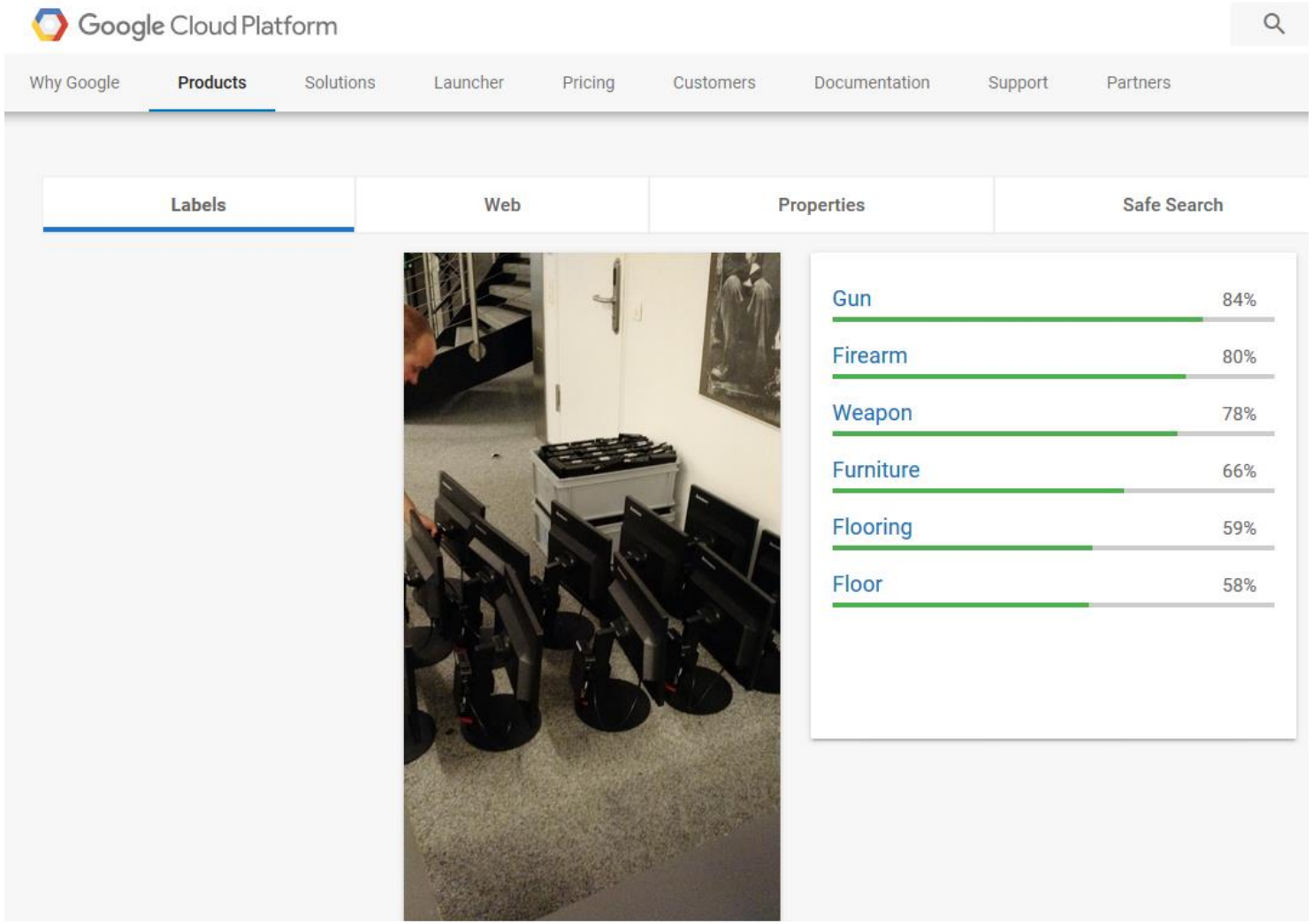


A baseball with 99.6 % confidence



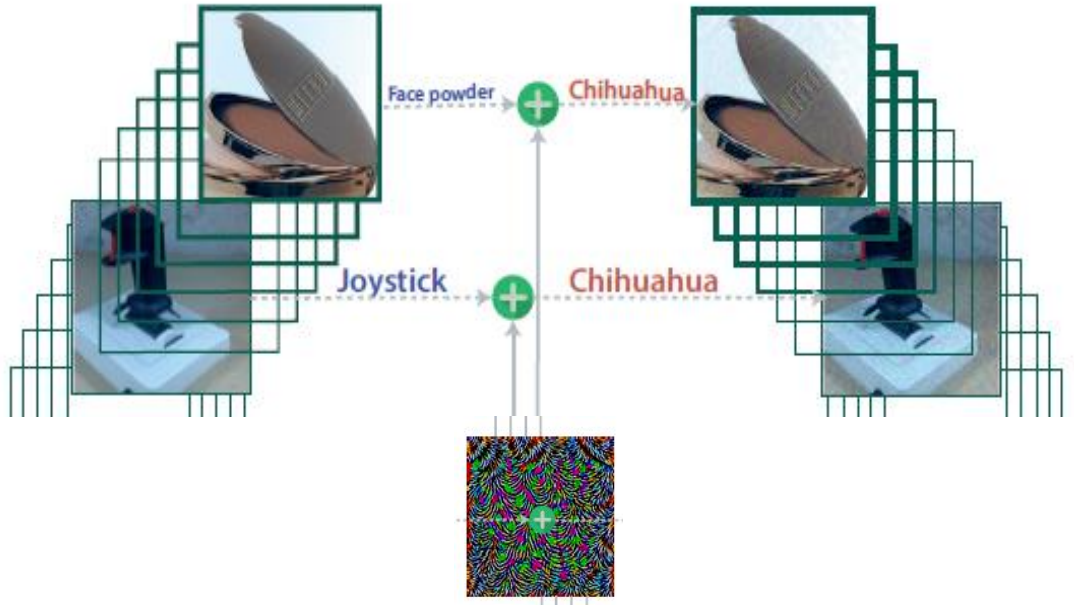
Incorrect Prediction with High Confidence by Google Vision Service in 2018

- Incorrect high-confidence predictions can occur anywhere at anytime leading to uncontrollable risks in an AI application



Universal Adversarial Perturbations in Neural Networks

- A small universal (image-agnostic) perturbation vector can be automatically computed that causes natural images to be misclassified with high probability
- Perturbations are quasi-imperceptible to the human eye and generalize very well across state-of-the-art neural networks
  - Attacks succeed even when the network architecture in an application is unknown



- Moosavi-Dezfooli, Seyed-Mohsen, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. “Universal adversarial perturbations.” In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1765-1773. 2017.

Summary

- Machine learning is a very important and huge research area in AI
- Supervised learning takes a set of annotated training data (input-output pairs) as input and computes a function to correctly predict the output for a given input
- Information theory provides an important basis for learning and the evaluation of learning algorithms
- Decision tree learning is an effective learning method for smaller data sets
- Feed-forward neural networks are directed layered graphs of units and can be trained with backpropagation
- Machine learning algorithms extract statistical models, not causal models - these models must be applied with extreme caution, they should not be applied to take decisions on human individuals

Working Questions

1. What is the architecture of a learning agent?
2. What is supervised learning?
3. How does Decision Tree learning work?
4. How can we compute the information gain of an attribute?
5. How can we evaluate the results of a learning algorithm?
6. What is the architecture of a neural network?
7. What is the difference between a perceptron and a multi-layer perceptron?
8. How does the training of neural networks work with backpropagation?