Machine Learning Basics

Prof. Dr. habil. Jana Koehler

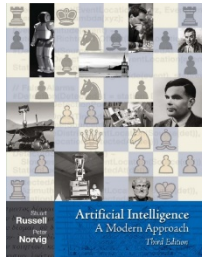
Artificial Intelligence - Summer 2019

Deep thanks go to Profs. Bernhard Nebel, Marc Pouly, Stuart Russell, Ismini Lourentzou and others for sharing their course material directly or via the web.

© DFKI - JK

Agenda

- The learning agent revisited
- Basic concepts of machine learning
- Decision Trees
  - An algorithm to find small trees
  - Entropy and information gain
- Neural Networks
  - Perceptron
  - Multi-Layer Perceptron and backpropagation
  - Deep Learning
- Evaluating the quality of learning algorithms

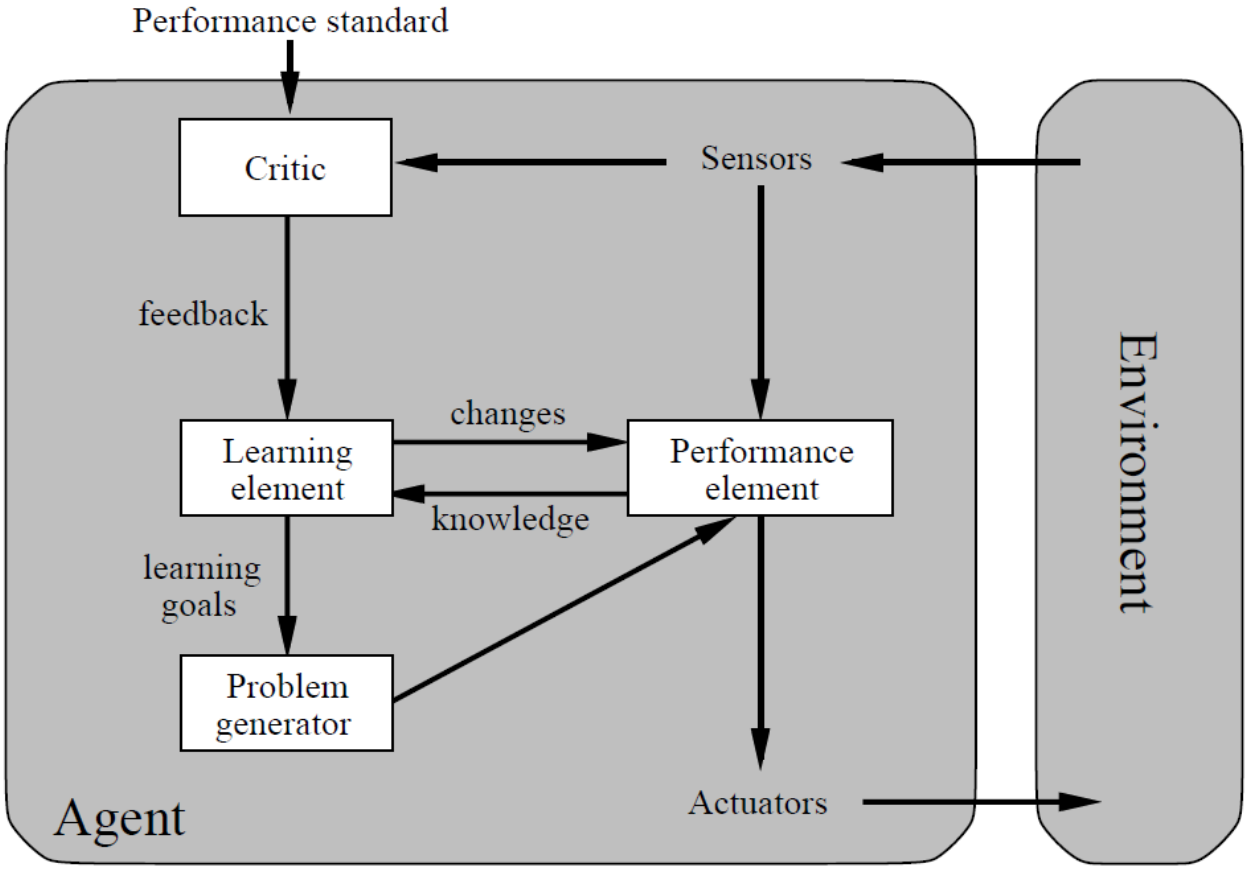


A small subset of chapters 18 and 21

© DFKI - JK

The Learning Agent

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



© DFKI - JK

Building Blocks of the Learning Agent

- Performance element: Processes percepts and chooses actions
  - corresponds to the agent model we have studied so far
- Learning element: Carries out improvements
  - requires self knowledge and feedback on how the agent is doing in the environment
- Critic: Evaluation of the agent's behavior based on a given external behavioral measure
- Problem generator: Suggests explorative actions that lead the agent to new experiences

© DFKI - JK

The Learning Element

Its design is affected by four major issues:

1. Which components of the performance element are to be learned?
2. What representation should be chosen?
3. What form of feedback is available?
4. Which prior information is available?

© DFKI - JK

Types of Feedback During Learning

- The type of feedback available is usually the most important factor in determining the nature of the learning problem
- Supervised learning
  - Involves learning a function from examples of its inputs and outputs
- Unsupervised learning
  - The agent has to learn patterns in the input when no specific output values are given
- Reinforcement learning
  - The most general form of learning in which the agent is not told what to do by a teacher
  - Rather it must learn from a reinforcement or reward
  - It typically involves learning how the environment works

© DFKI - JK

Supervised Learning

- An 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
  - good hypothesis should **generalize** the data well, i.e., will predict unseen examples correctly
  - hypothesis is **consistent** with the data set if it agrees with all the data

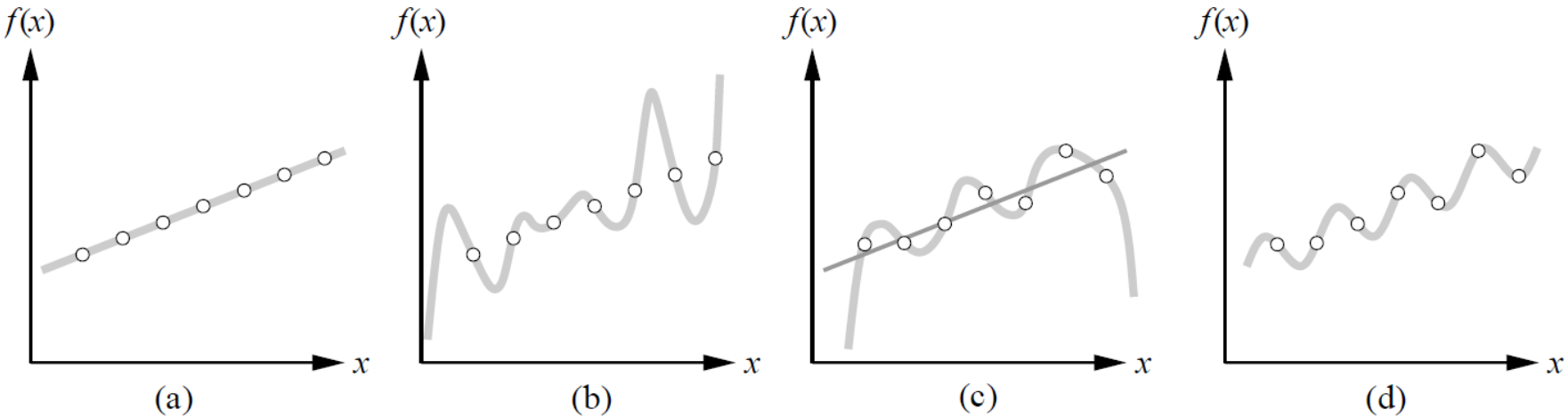
© DFKI - JK

Ockham's Razor

- How do we choose from among multiple consistent hypotheses?
  - prefer the simplest hypothesis consistent with the data
  - maximize a combination of consistency and simplicity

© DFKI - JK

Example: Fitting a Function to a Data Set



(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

Ockham´s razor?

© DFKI - JK

Inductive/Supervised Learning of *f*

- Is a highly simplified model of real learning that ignores prior knowledge
  - assumes a deterministic, observable environment
  - assumes examples are given
  - assumes that the agent wants to learn *f* - why?

- But has shown impressive results over the last years
  - training data occur in massive amounts
  - many applications possible in deterministic, observable environments (online shopping, social networks)
  - high financial incentives (marketing, sales) to learn *f*

© DFKI - JK

Decision Tree Learning

© DFKI - JK

Decision Tree

Input

Description of an object or a situation through a set of attributes

Output

Decision as the predicted output value for the input

- Both, input and output can be discrete or continuous
- Discrete-valued functions lead to classification problems
- Learning a continuous function is called *regression*

© DFKI - JK

Boolean Decision Tree

Input

Set of vectors of input attributes X and a single Boolean output value *y* (goal predicate)

Output

Yes/No decision based on a goal predicate

- Goal of the learning process: Definition of the goal predicate in the form of a decision tree
- Boolean decision trees represent Boolean functions

© DFKI - JK

Properties of (Boolean) Decision Trees

- An internal node of the decision tree represents a test of a property
- Branches are labeled with the possible values of the test
- Each leaf node species the Boolean value to be returned if that leaf is reached

© DFKI - JK

The Restaurant Example

Goal predicate: $WillWait$
Test predicates:

- $Patrons$: How many guests are there? ($none$, $some$, $full$)
- $WaitEstimate$: How long do we have to wait? (0-10, 10-30, 30-60, >60)
- $Alternate$: Is there an alternative? (T/F)
- $Hungry$: Am I hungry? (T/F)
- $Reservation$: Have I made a reservation? (T/F)
- Bar: Does the restaurant have a bar to wait in? (T/F)
- Fri/Sat: Is it Friday or Saturday? (T/F)
- $Raining$: Is it raining outside? (T/F)
- $Price$: How expensive is the food? ($, $$, $$$)
- $Type$: What kind of restaurant is it? ($French$, $Italian$, $Thai$, $Burger$)

© DFKI - JK

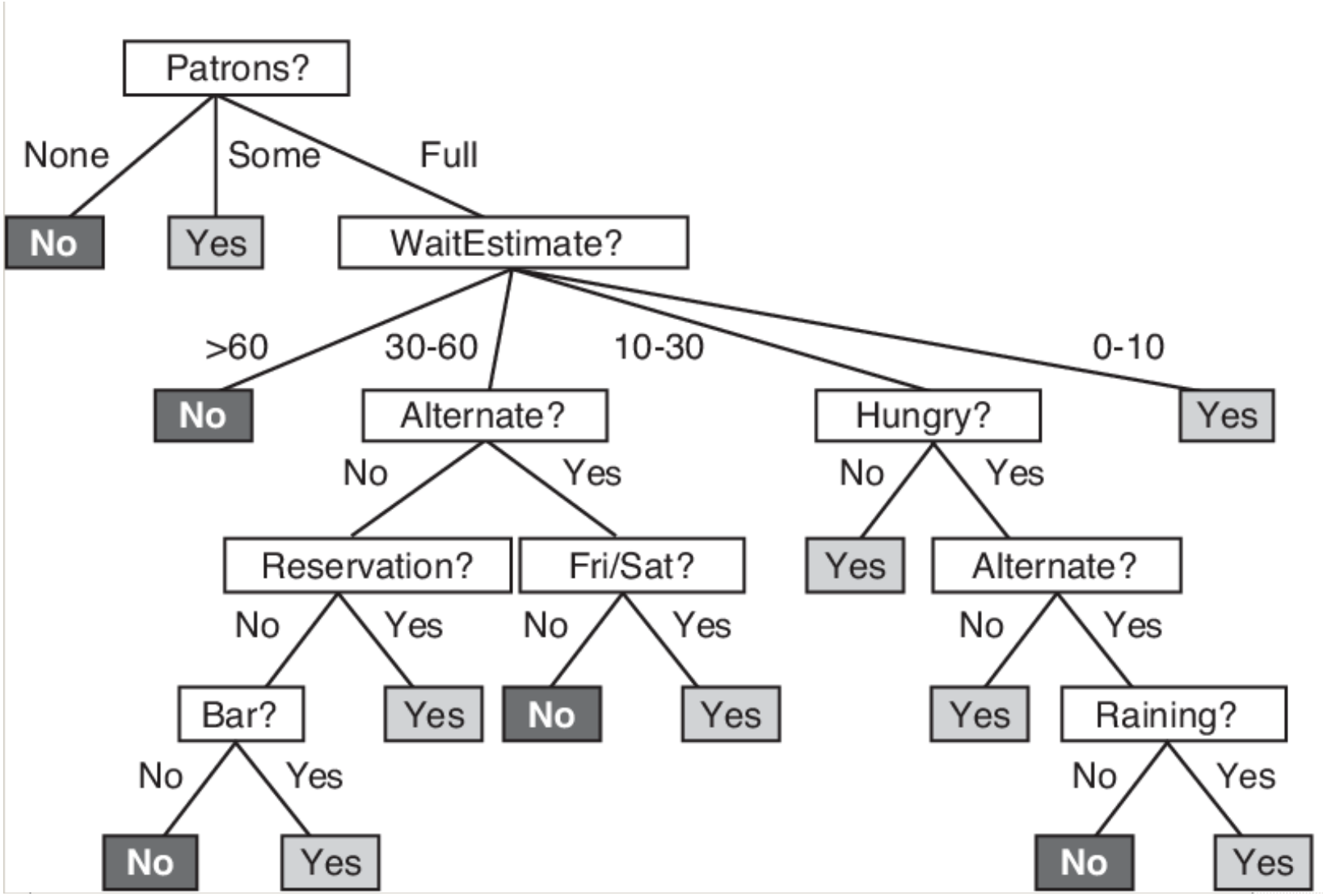
The Training Set

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Classification of examples is positive (T) or negative (F)

© DFKI - JK

A Boolean Decision Tree



© DFKI - JK

Inducing Decision Trees from Examples

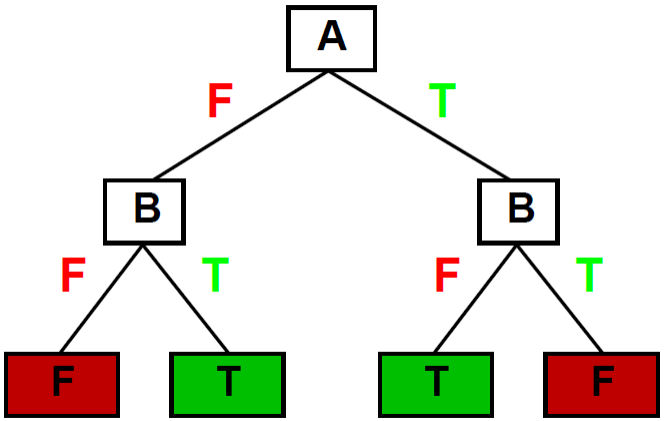
- Naive solution: we simply construct a tree with one path to a leaf for each example
- In this case, we test all the attributes along the path and attach the classification of the example to the leaf
- Whereas the resulting tree will correctly classify all given examples, it will not say much about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

© DFKI - JK

Memoizing Examples

- A trivial consistent decision tree for any training set has one path from root to leaf for each example (= row in the corresponding table of training sets)

A	B	A xor B
F	F	F
F	T	T
T	F	T
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

© DFKI - JK

Hypotheses Spaces

- How many distinct decision trees can we built with *n* Boolean variables (attributes)?

  = number of Boolean functions

  = number of distinct truth tables with 2^n rows = 2^{2^n}

- With 6 Boolean attributes, we have 18,446,744,073,709,551,616 trees

© DFKI - JK

Expressiveness of Decision Trees

- Each decision tree hypothesis for the *WillWait* goal predicate can be seen as an assertion of the form

$$\forall s\, WillWait(s) \Leftrightarrow (P_1(s) \vee P_2(s) \vee \ldots \vee P_n(s))$$

where each *Pi(s)* is the conjunction of tests along a path from the root of the tree to a leaf with a positive outcome

- Any Boolean function can be represented by a decision tree
- Limitation: All tests always involve only <u>one object</u> and the language of traditional decision trees is inherently <u>propositional</u>

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

cannot be represented as a test. We could always add another test called *CheaperRestaurantNearby*, but a decision tree with all such attributes would grow exponentially

© DFKI - JK

Compact Representations

- For every Boolean function we can construct a decision tree by translating every row of a truth 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: $$m(x) = \begin{cases} 1 & \text{half of the inputs are 1} \\ 0 & \text{otherwise} \end{cases}$$

- There is no consistent representation that is compact for all possible Boolean functions

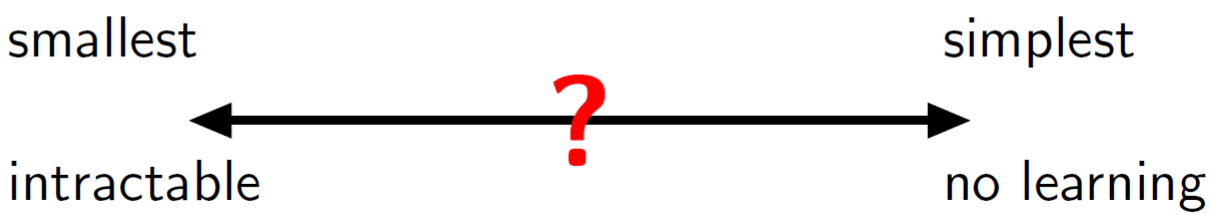
© DFKI - JK

Finding a Smallest Tree

- Applying Ockham's razor we should instead find the smallest decision tree that is consistent with the training set

- Unfortunately, for any reasonable definition of “smallest” finding the smallest tree is intractable

Dilemma:



➢ We can devise a decision tree learning algorithm that generates “smallish“ trees

© DFKI - JK

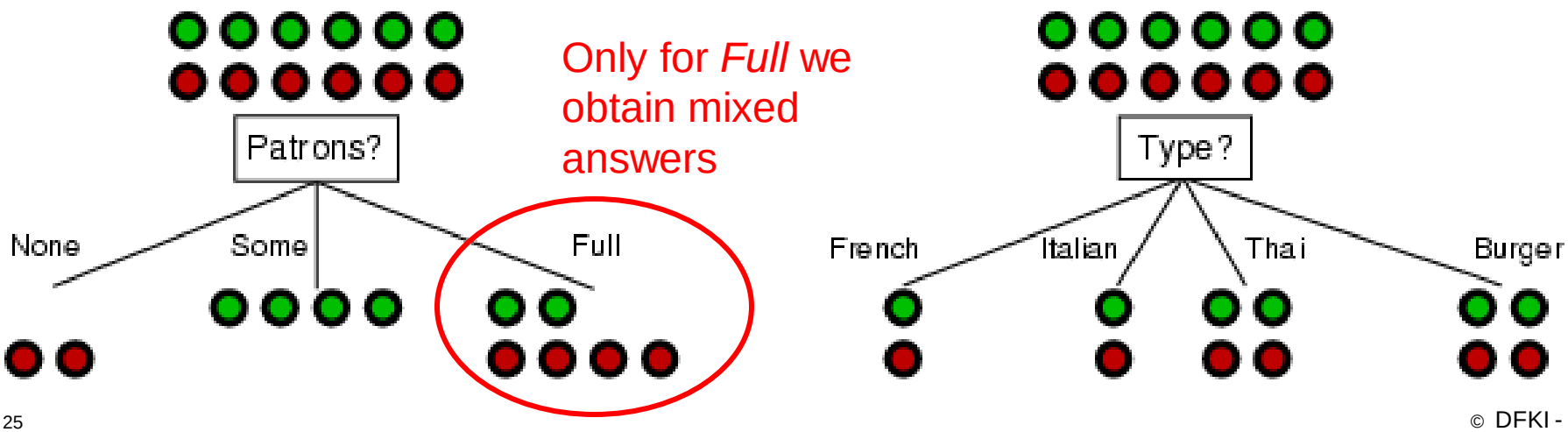
Idea of Decision Tree Learning

- Divide and Conquer approach:
  - Choose an (or better: the best) attribute
- Split the training set into subsets each corresponding to a particular value of that attribute
- Now that we have divided the training set into several smaller training sets, we can recursively apply this process to the smaller training sets

© DFKI - JK

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, since it leaves us with four subsets each of them containing the same number of positive and negative examples - no reduction in problem complexity



© DFKI - JK

Recursive Learning Process

In each recursive step there are four cases to consider:

1) Positive and negative examples
   - choose a new attribute
2) Only positive (or only negative) examples
   - done (answer is Yes or No)
3) No examples
   - there was no example with the desired property
   - Answer *Yes* if the majority of the parent node's examples is positive, otherwise *No*
4) No attributes left, but there are still examples with different classifications
   - there were errors in the data (! NOISE) or the attributes do not give sufficient information. Answer *Yes* if the majority of examples is positive, otherwise *No*

© DFKI - JK

The Algorithm

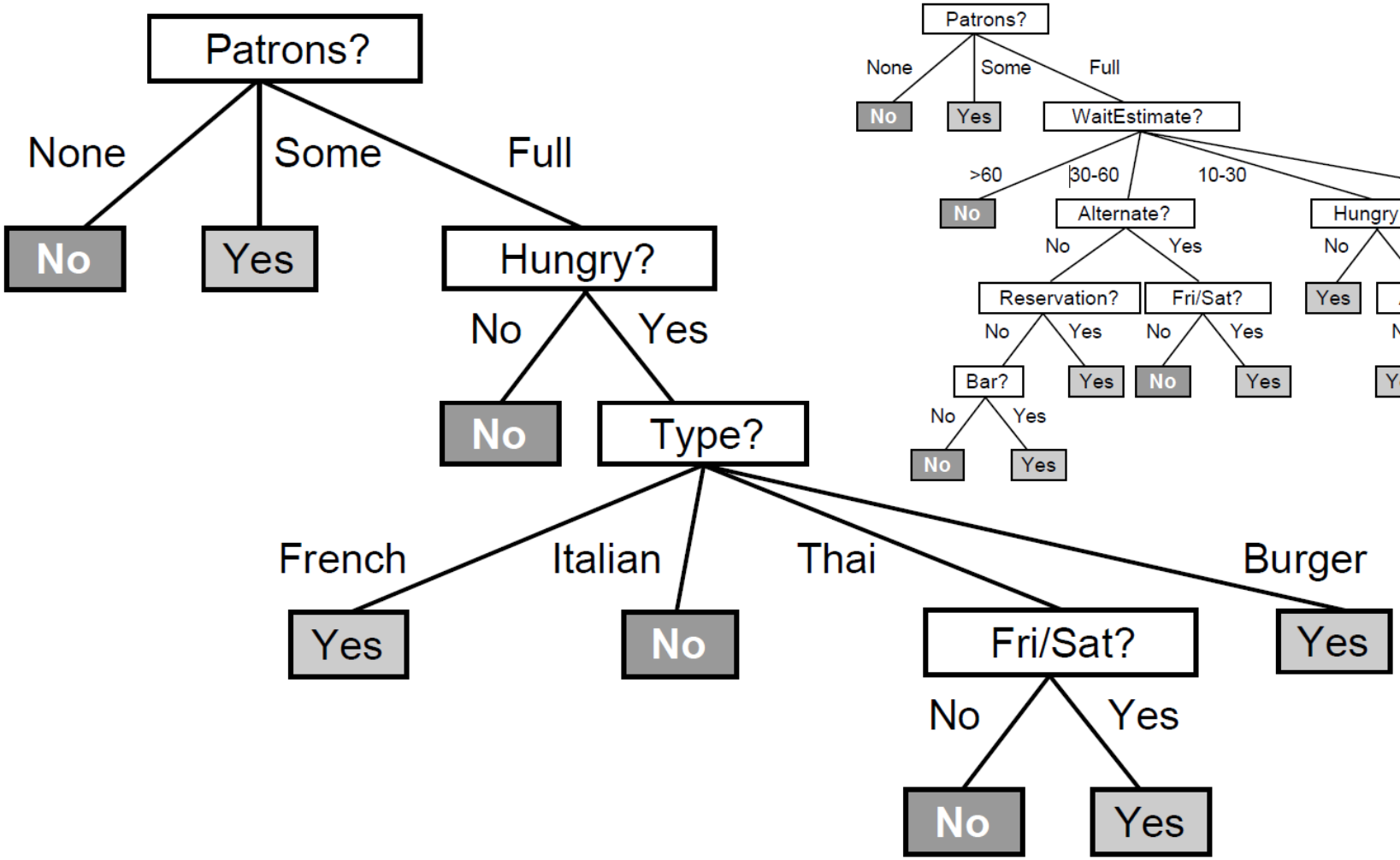
```
function DECISION-TREE-LEARNING(examples, attributes, parent_examples) returns
a tree

  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 ← argmax_{a ∈ attributes} IMPORTANCE(a, examples)
     tree ← a new decision tree with root test A
     for each value v_k of A do
        exs ← {e : e ∈ examples and e.A = v_k}
        subtree ← DECISION-TREE-LEARNING(exs, attributes − A, examples)
        add a branch to tree with label (A = v_k) and subtree subtree
     return tree
```

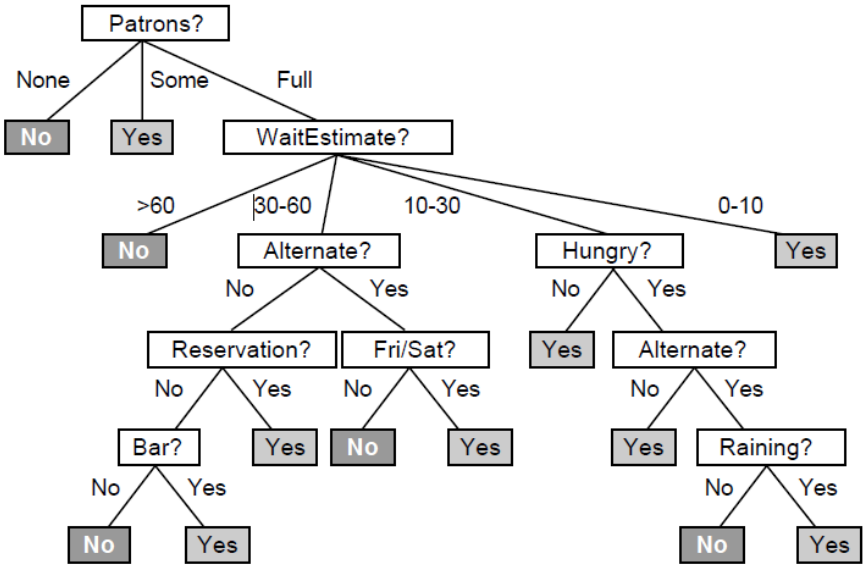
Figure 18.5 The decision-tree learning algorithm. The function IMPORTANCE is described in Section 18.3.4. The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly.

© DFKI - JK

The Learned Tree



Original tree:



© DFKI - JK

Properties of the Resulting Tree

- The learned tree is considerably simpler than the one originally given (and from which the training examples were generated)
- The learning algorithm outputs a tree that is consistent with all examples it has seen
- The tree does not necessarily agree with the correct function: For example, it suggests not to wait if we are not hungry. If we are, there are cases in which it tells us to wait
- Some tests (Raining, Reservation) are not included since the algorithm can classify the examples without them

© DFKI - JK

The Function IMPORTANCE

- The greedy search used in decision tree learning is designed to approximately minimize the depth of the final tree
- The idea is to pick the attribute that goes as far as possible towards providing an exact classification of the examples
- A perfect attribute divides the examples into sets, each of which are all positive or all negative and thus will be leaves of the tree

➢ Formal measure of "fairly good“ (e.g. Patrons?) and "really useless" (Type?) to implement the IMPORTANCE function

© DFKI - JK

The Notion of Information Gain

- Information gain is defined in terms of **entropy,** the fundamental quantity in information theory (Shannon and Weaver, 1949)

- Measure the uncertainty of a random variable

- Acquisition of information corresponds to a reduction in entropy

➢ The less we know about the outcome, the more valuable the prior information

© DFKI - JK

Example: Tossing a Coin

- A random variable with only one value - a coin that always comes up heads - has no uncertainty
  - its entropy is defined as zero
  - we gain no information by observing its value

- A flip of a fair coin is equally likely to come up heads or tails, 0 or 1
  - this counts as **1 bit of entropy**
- The roll of a fair *four-sided* die has **2 bits of entropy**
  - it takes 2 bits to describe one of four equally probable choices

© DFKI - JK

Entropy

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

$$\text{Entropy:} \quad H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

- Entropy of a fair coin:

$$H(Fair) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1$$

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

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

© DFKI - JK

Entropy of a Boolean Variable

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

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

- E.g. $H(Loaded) = B(0.99) \approx 0.08$

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

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

© DFKI - JK

Influence of Testing an Attribute on Entropy

- The restaurant example has p=n=6, so B(0.5) is 1 bit
  - 0.5 because we do wait or not (2 possible outcomes)

- A test on a single attribute can give us only a part (not more) information gain than 1 bit

- We can measure exactly how much by measuring the entropy <u>after</u> performing the test on this attribute
- The value of an attribute A depends on the additional information that we still need to collect <u>after</u> we selected it

© DFKI - JK

Information Provided by an Attribute

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

- Each subset E_k has p_k positive and n_k negative examples, so along this branch we will need an additional

$$B(p_k/(p_k + n_k))$$

bits of information to answer the question

- A randomly chosen variable from the training set has the kth value for the attribute with probability

$$(p_k + n_k)/(p + n)$$

© DFKI - JK

Expected Entropy Remaining after Testing Attribute A

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

- The **information gain** from the attribute test on A is the expected reduction in Entropy

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

© DFKI - JK

IMPORTANCE Function

- IMPORTANCE function computes the information gain and chooses the attribute with the largest value

$$Gain(Patrons) = 1 - [\tfrac{2}{12}I(0,1) + \tfrac{4}{12}I(1,0) + \tfrac{6}{12}I(\tfrac{2}{6},\tfrac{4}{6})] \approx 0.541$$

$$Gain(Type) = 1 - [\tfrac{2}{12}I(\tfrac{1}{2},\tfrac{1}{2}) + \tfrac{2}{12}I(\tfrac{1}{2},\tfrac{1}{2}) + \tfrac{4}{12}I(\tfrac{2}{4},\tfrac{2}{4}) + \tfrac{4}{12}I(\tfrac{2}{4},\tfrac{2}{4})] = 0$$

Remark: In this formula, I=B.

© DFKI - JK

Advantages of Decision Trees

- One possibility for representing (Boolean) functions
- Small DT can be generated by ranking the attributes based on the information gain
- Are simple to understand and interpret for humans
- Can handle both numeric and categorical data
- Require almost no data preparation
- Perform well even on larger datasets

© DFKI - JK

Disadvantages of Decision Trees

- Often not as accurate as other approaches
  - Can be exponential in the number of attributes
  - Often too difficult to find the minimal DT
- Learners tend to create over-complex trees that do not generalize well (overfitting)
  - Pruning is very important when training decision trees
- A small change in the training data can result in a big change in the tree
  - Tree models are very sensitive to data quality
- Generalizing DT leads us to Random Forests Learning, which address these issues

© DFKI - JK

Assessing the Performance of the Learning Algorithm

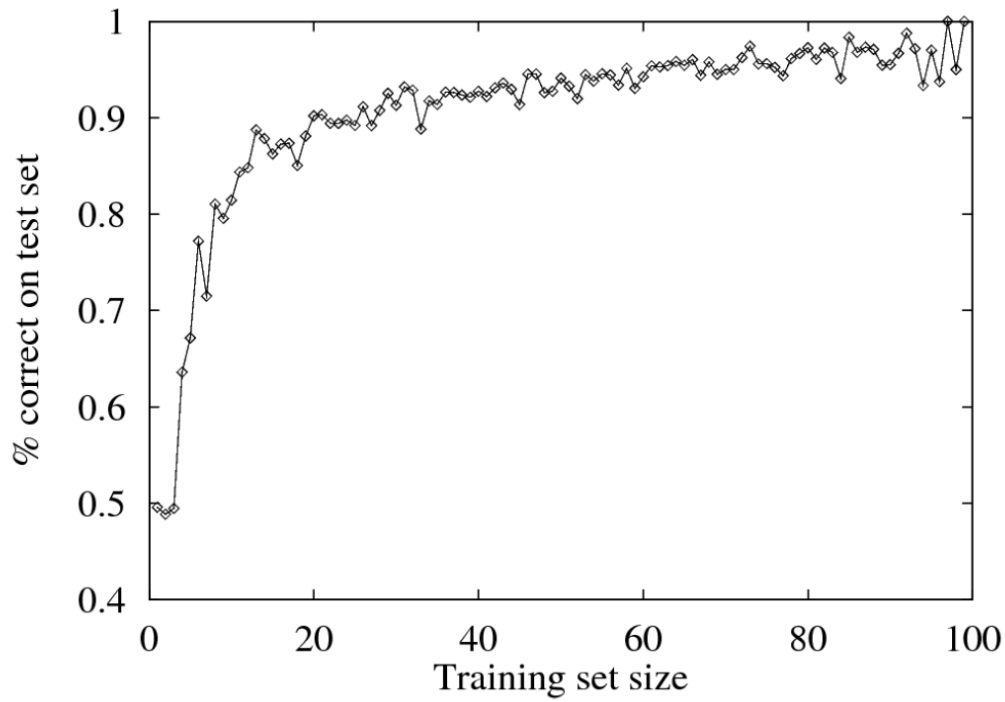
1. Collect a large number of examples
2. Divide it into two <u>disjoint</u> sets: the training set and the test set
3. Use the training set to generate *h*
4. Measure the percentage of examples of the test set that are correctly classified by *h*
5. Repeat the process for randomly-selected training sets of different sizes

© DFKI - JK

Learning Curve for the Restaurant Example

- How do we know that $h \approx f$?
  1) Use theorems of computational/statistical learning theory
  2) Try h on a new test set of examples
     (use same distribution over example space as training set)

Learning curve = % correct on test set as a function of training set size



© DFKI - JK

Separation of Training and Test Set

- The training and test sets must be kept separate!
- Common error: Changing the algorithm after running a test, and then testing it with training and test sets from the same basic set of examples
- By doing this, knowledge about the test set gets stored in the algorithm, and the training and test sets are no longer independent

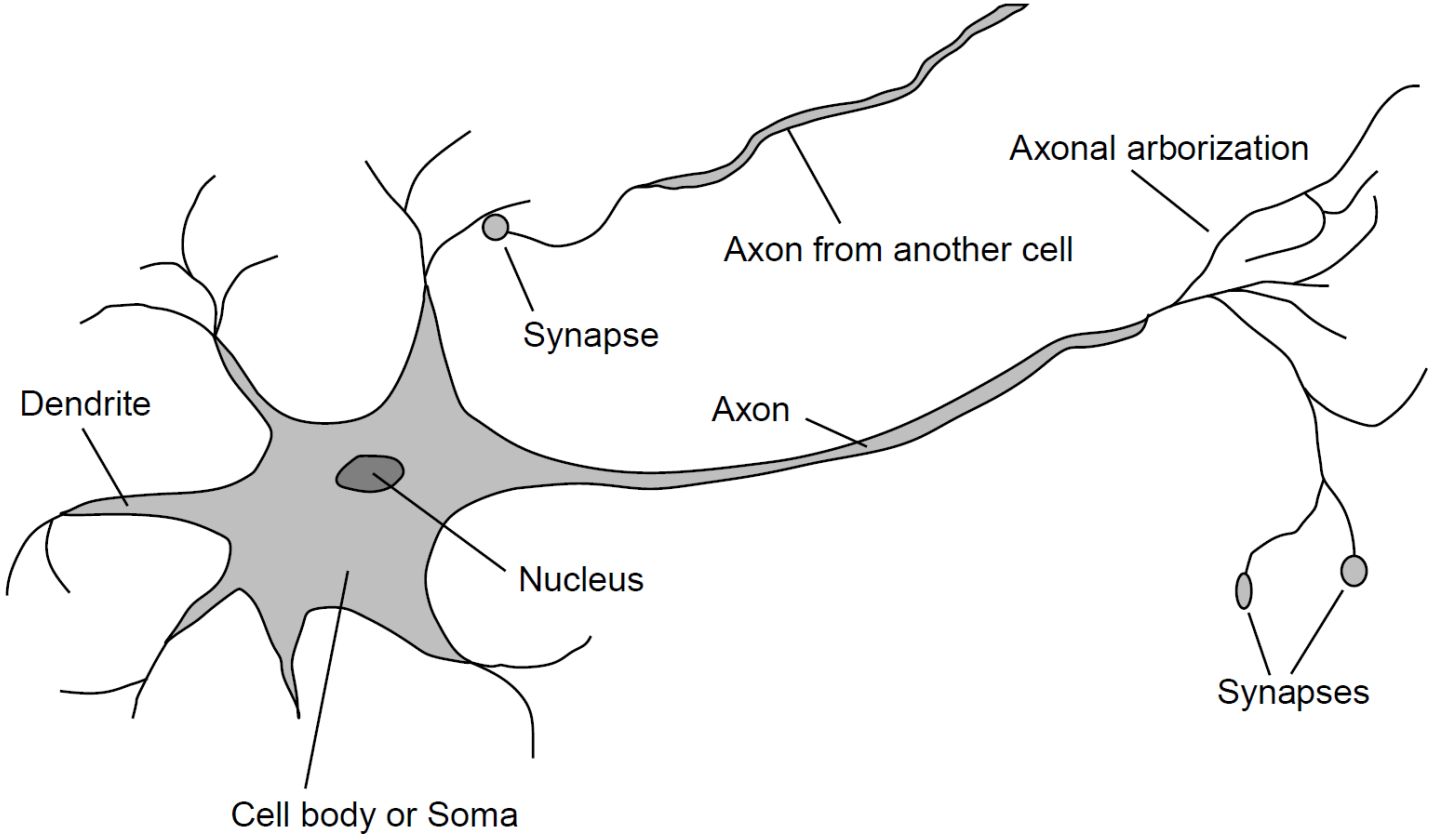
© DFKI - JK

Neural Networks

© DFKI - JK

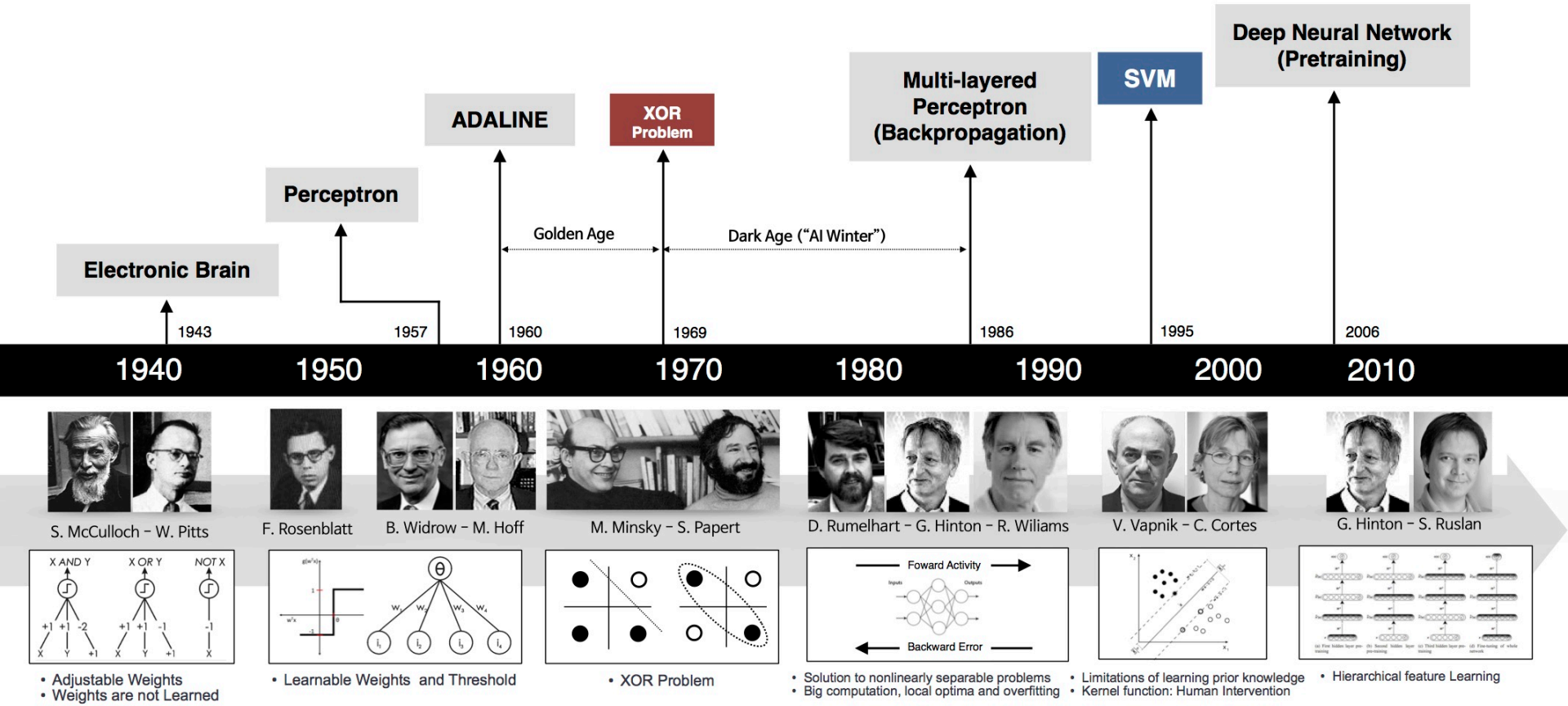
A Neuron in the Human Brain

- 10^{11} neurons of > 20 types, 10^{14} synapses
- 1ms - 10ms cycle time
- Signals are noisy “spike trains" of electrical potential



© DFKI - JK

Neural Networks Research

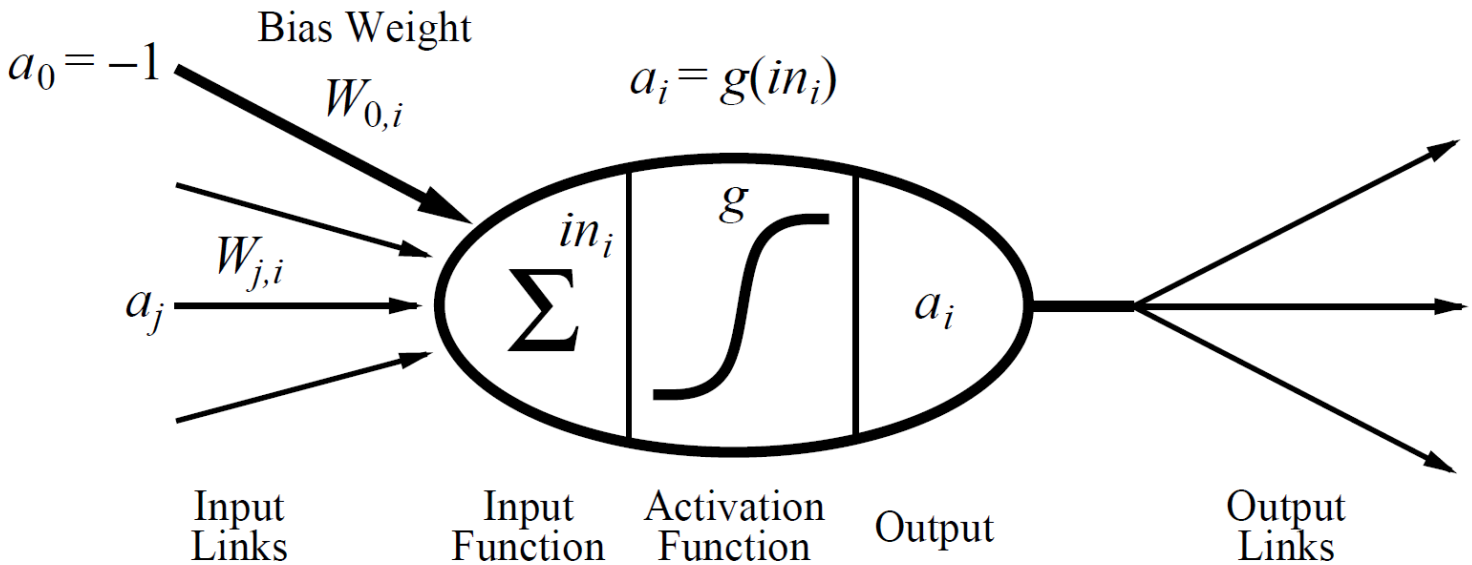


https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

McCulloch-Pitts "Unit"

- Output is a "squashed" linear function of the inputs

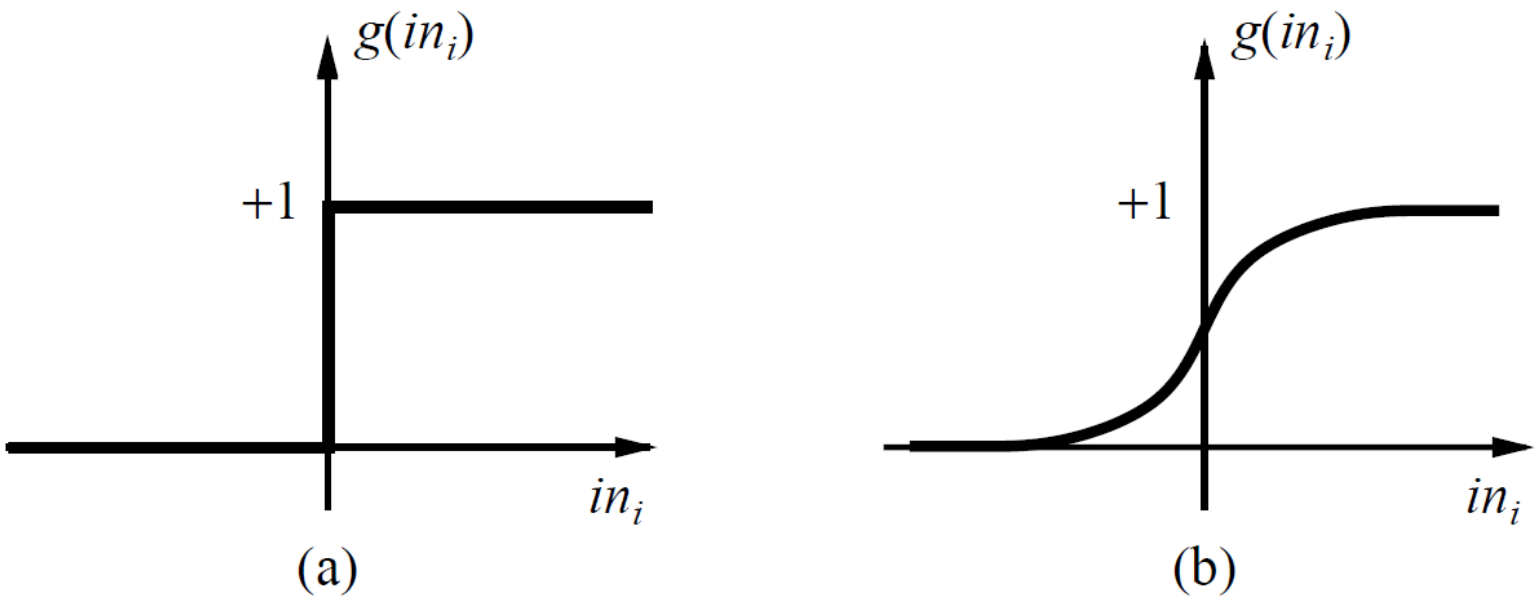
$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$



- An oversimplification of real neurons, but its purpose is to develop an understanding of what networks of simple units can do

© DFKI - JK

Activation Functions



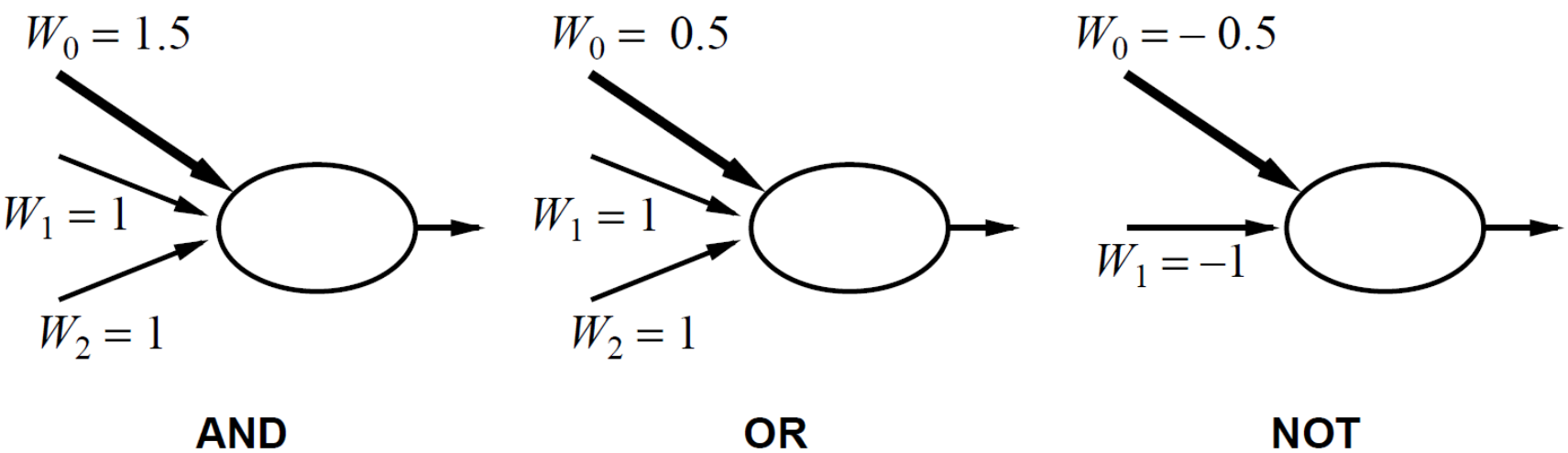
(a) is a step function or threshold function

(b) is a sigmoid function $1/(1+e^{-x})$

© DFKI - JK

Implementing Logical Functions

- McCulloch and Pitts: every Boolean function can be implemented



© DFKI - JK

Network Structures

- Feed-forward networks:
  - Single-layer perceptron
  - Multi-layer perceptron

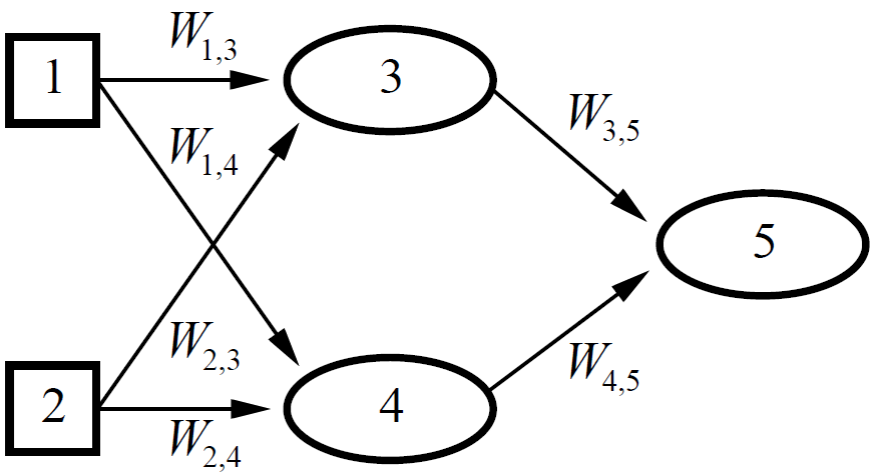
- Feed-forward networks implement functions, have no internal state

- Recurrent networks:
  - Hopfield networks have symmetric weights ($W_{ij} = W_{ji}$)
  - Boltzmann machines use stochastic activation functions
  - Recurrent neural nets have directed cycles with delays

© DFKI - JK

Feed-forward Example

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

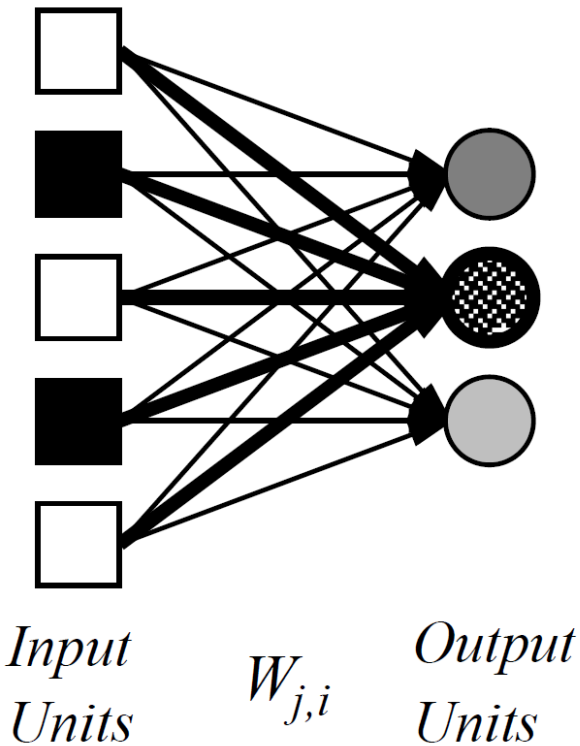


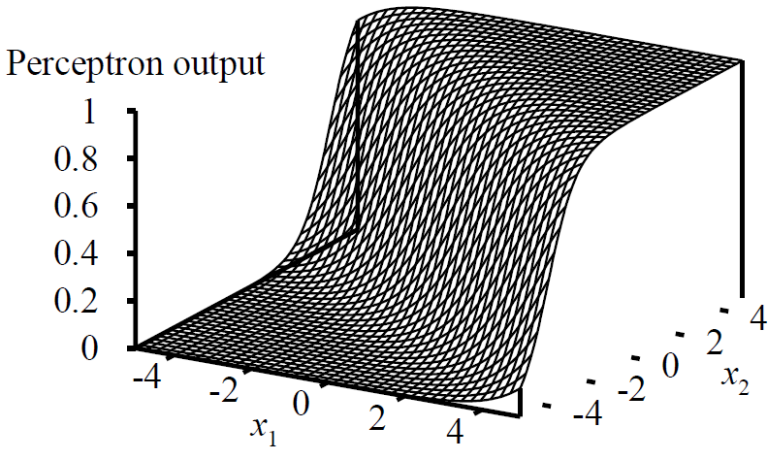
$$
\begin{aligned}
a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\
&= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2))
\end{aligned}
$$

© DFKI - JK

Single-layer Perceptron

- Output units all operate separately - no shared weights
- Adjusting weights moves the location, orientation, and steepness of cliff



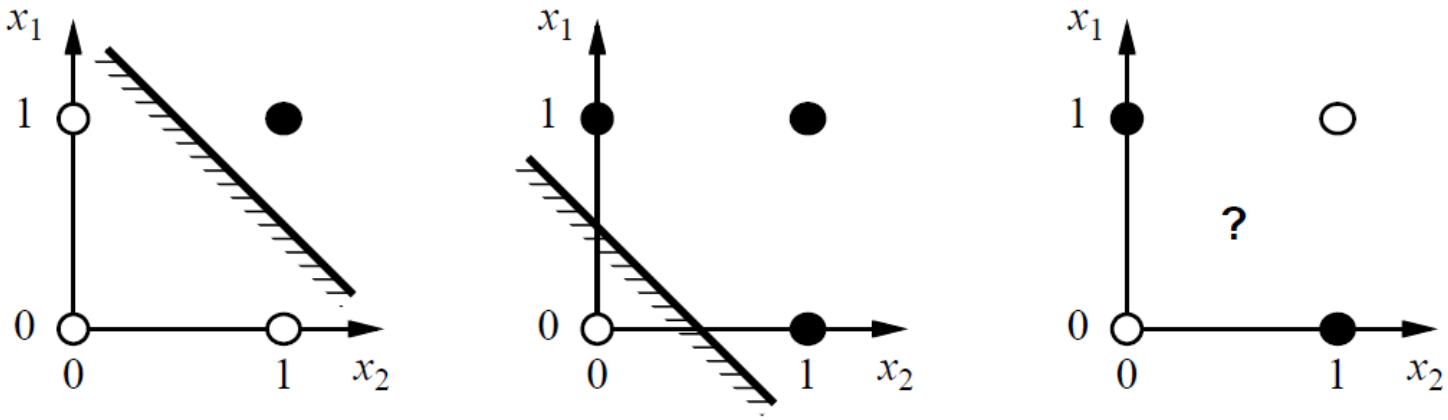


© DFKI - JK

Expressiveness of Perceptrons

- Consider a perceptron with g = step function (Rosenblatt 1957/60)
  - Can represent AND, OR, NOT, majority, etc., but not XOR
  - Represents a linear separator in input space

$\sum_j W_j x_j > 0$ or $\mathbf{W} \cdot \mathbf{x} > 0$



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

Perceptron Learning

- Learn by adjusting weights to reduce error on training set
  - The squared error for an example with input *x* and true output *y* is

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2$$

- Perform optimization search by gradient descent:

$$\begin{aligned} \frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}\left(y - g(\Sigma_{j=0}^{n} W_j x_j)\right) \\ &= -Err \times g'(in) \times x_j \end{aligned}$$

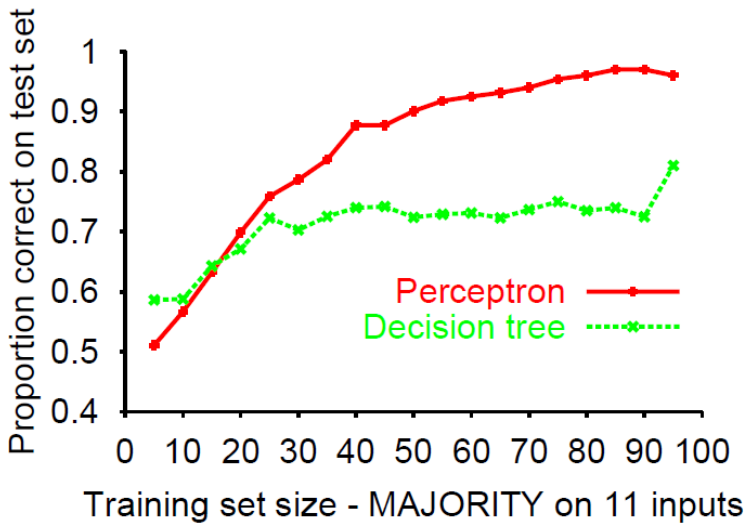
- Simple weight update rule:

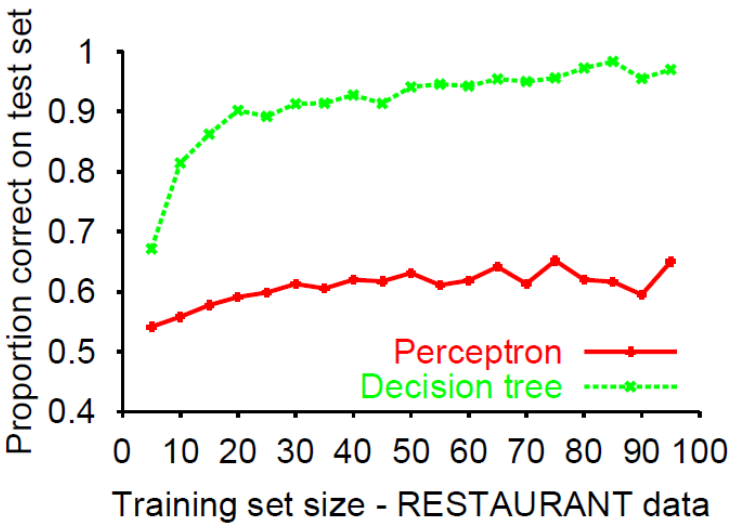
$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

© DFKI - JK

Perceptron Learning Contd.

- Perceptron learning rule converges to a consistent function for any linearly separable data set
  - Perceptron learns majority function easily, DT is hopeless
  - DT learns restaurant function easily, perceptron cannot represent it

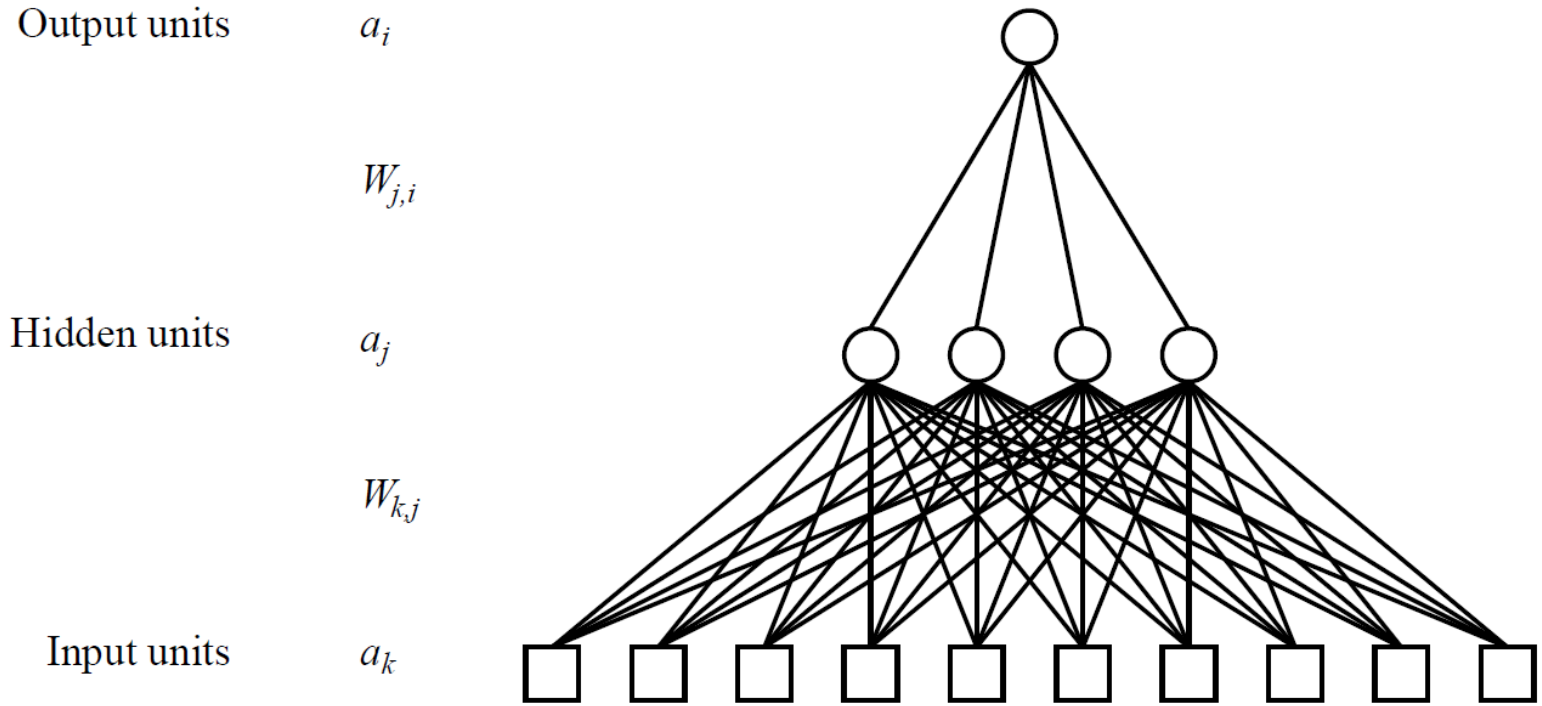




© DFKI - JK

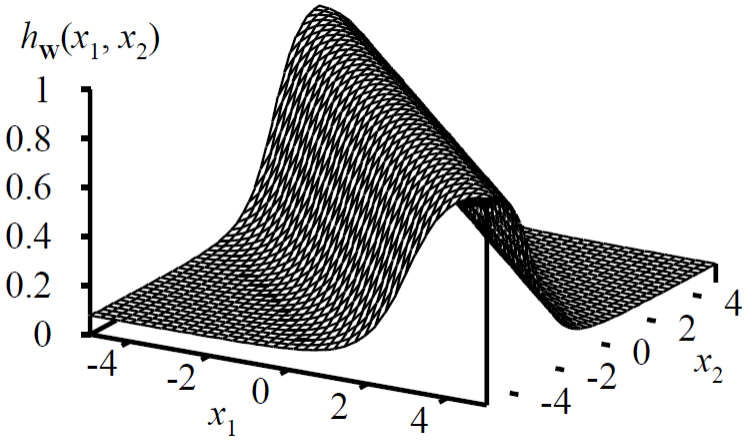
Multilayer Perceptron

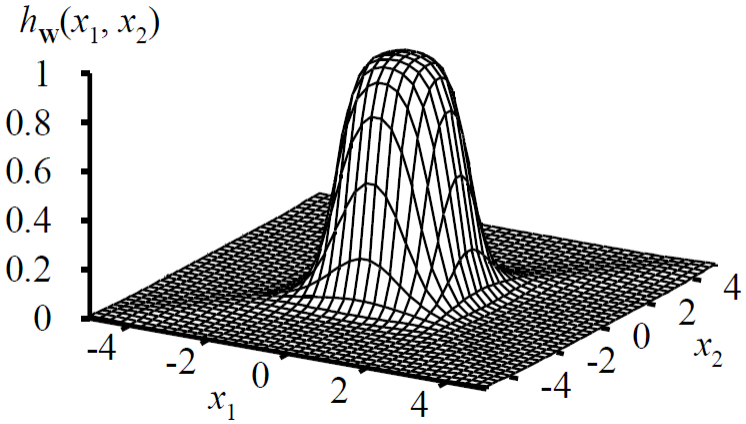
- Layers are usually fully connected
- Numbers of hidden units typically chosen by hand



Expressiveness of Multilayer Perceptron

- All continuous functions w/ 2 layers, all functions w/ 3 layers
  - Combine two opposite-facing threshold functions to make a ridge
  - Combine two perpendicular ridges to make a bump
  - Add bumps of various sizes and locations to t any surface





© DFKI - JK

Backpropagation Learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where $\Delta_i = Err_i \times g'(in_i)$

Hidden layer: **back-propagate** the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \ .$$

Update rule for weights in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

(Most neuroscientists deny that back-propagation occurs in the brain)

© DFKI - JK

Backpropagation Derivation

The squared error on a single example is defined as

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2 \ ,$$

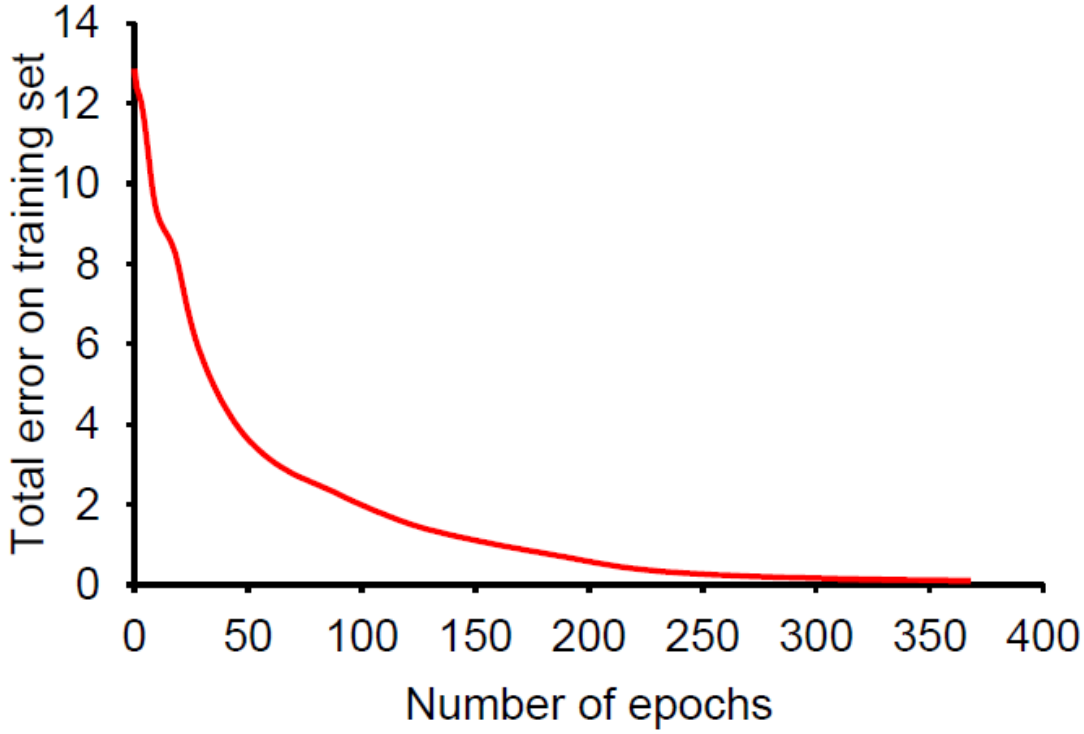
where the sum is over the nodes in the output layer.

$$\begin{aligned}
\frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}} \\
&= -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right) \\
&= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i
\end{aligned}$$

© DFKI - JK

Backpropagation Learning Contd.

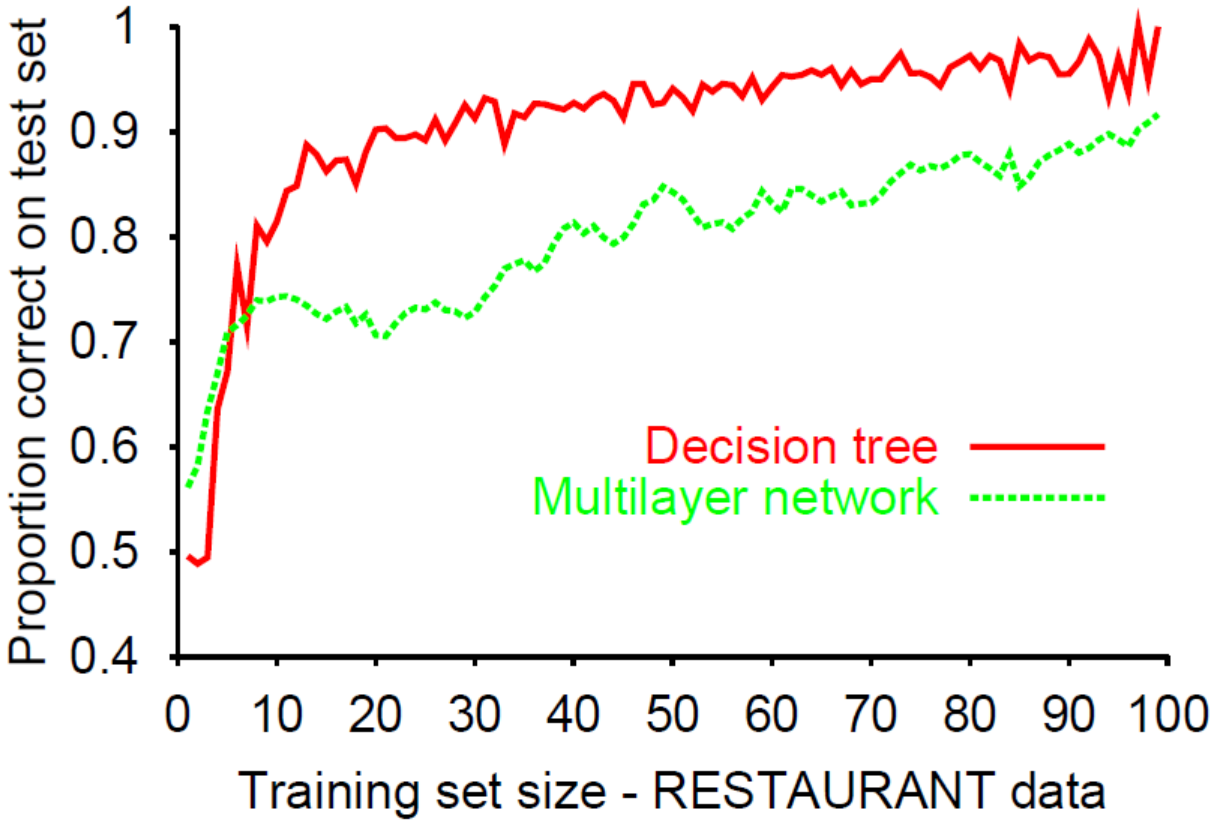
- At each epoch, sum gradient updates for all examples and apply
  - Training curve for 100 restaurant examples: finds exact *t*
  - Typical problems: slow convergence, local minima



© DFKI - JK

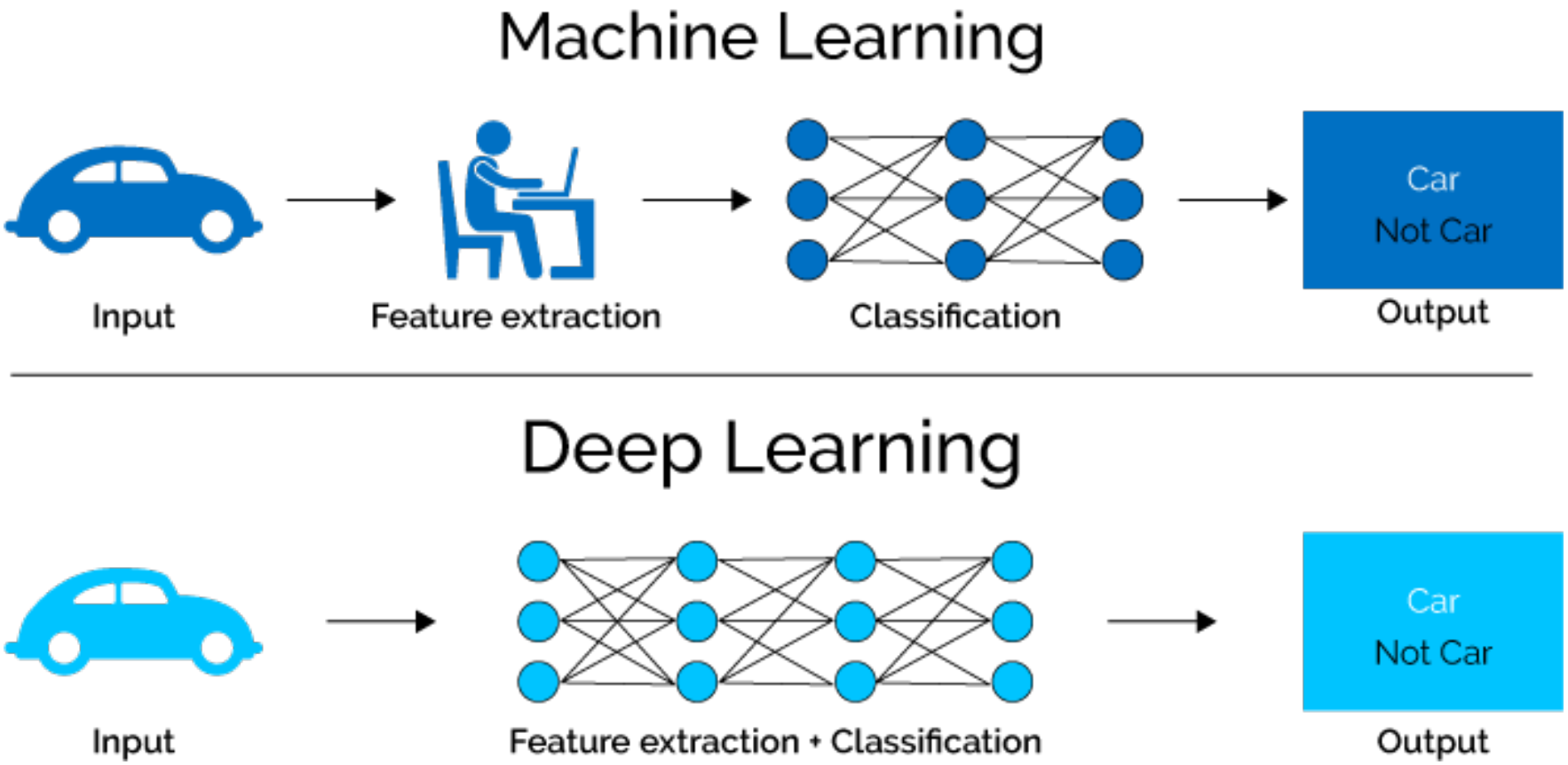
Backpropagation Learning Contd.

- MLPs are quite good for complex pattern recognition tasks
  - Learning curve for MLP with 4 hidden units:



© DFKI - JK

Deep Learning



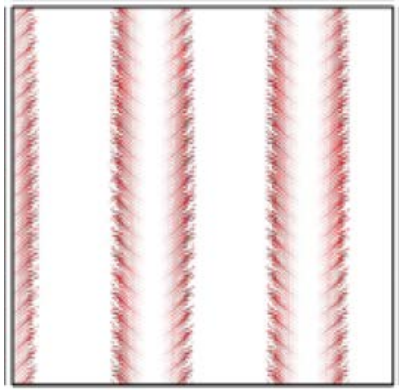
© DFKI - JK

Why is Deep Learning Successful?

- Manually designed features are often over-specified, incomplete and take a long time to design and validate
- Learned features are easy to adapt, fast to learn
- Deep learning provides a very flexible, almost universal, learnable framework for representing world, visual and linguistic information
- Can learn both unsupervised and supervised
- Effective end-to-end joint system learning
- Utilize large amounts of training data

© DFKI - JK

Challenges with Deep Neural Networks



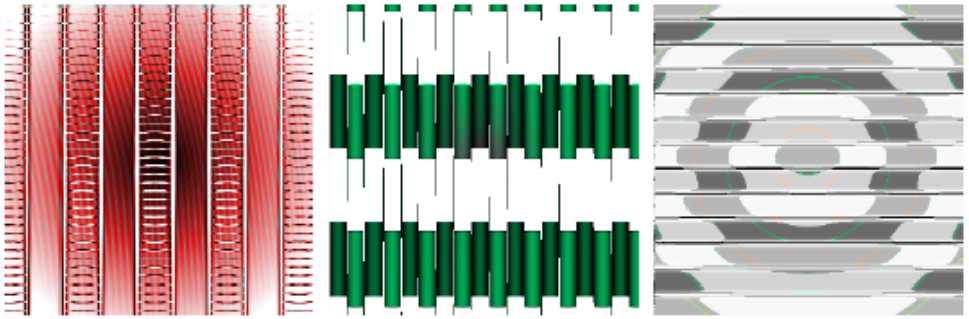
A baseball with 99.6 % confidence

Deep Neural Networks are Easily Fooled:
High Confidence Predictions for Unrecognizable Images

Anh Nguyen
University of Wyoming

Jason Yosinski
Cornell University

Jeff Clune
University of Wyoming

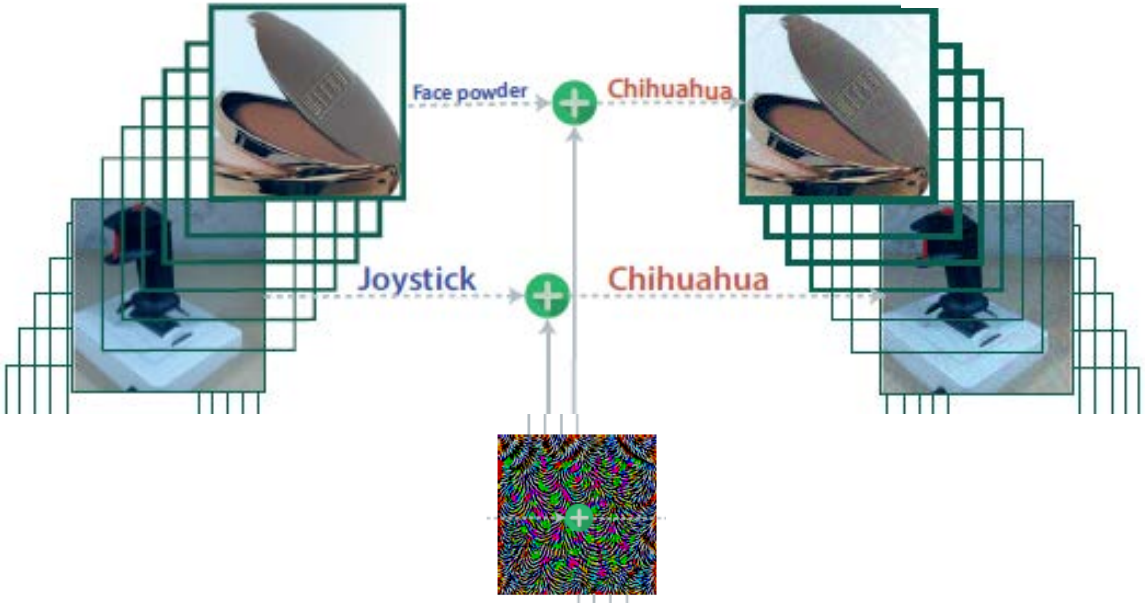


accordion screwdriver photocopier

Challenges with Deep Neural Networks Cntd.

Universal adversarial perturbations

Seyed-Mohsen Moosavi-Dezfooli*†
seyed.moosavi@epfl.ch
Alhussein Fawzi*†
alhussein.fawzi@epfl.ch
Omar Fawzi‡
omar.fawzi@ens-lyon.fr
Pascal Frossard†
pascal.frossard@epfl.ch



© DFKI - JK

Evaluating the Quality of Learning Algorithms

© DFKI - JK

Evaluating the Quality of Learning Algorithms

- 4 possible outcomes for each test example

1) TRUE Positive (TP)

2) TRUE Negative (TN)

3) FALSE Positive (FP)

4) FALSE Negative (FN)

© DFKI - JK

Ac

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}}$$

- Accuracy is easy to interpret but has severe drawbacks
  - It does not work for imbalanced data
  - It does not distinguish between FP and FN which makes debugging hard
- Here accuracy = 91%

© DFKI - JK

Sensitivity – Performance on YES instances

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

$$\text{Sensitivity} = \frac{\text{TP}}{\text{Actual YES}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- When the true value is YES, how often does the model predict YES
- Sensitivity is also called True-Positive Rate or Recall
- Here: Sensitivity = 95%

© DFKI - JK

Specificity – Performance on NO instances

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

$$\text{Specificity} = \frac{\text{TN}}{\text{Actual NO}} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

- When the true value is NO, how often does the model predict NO

- Here: Specificity = 83%

© DFKI - JK

Precision – When TN cannot be assessed

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

$$\text{Precision} = \frac{\text{TP}}{\text{Predicted YES}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- An issue with accuracy and specificity is that TN cannot always be counted
  - Example: TN of a Google search query ?
- Use precision instead: When the model predicts YES, how often is it correct
  - Here: Precision = 91%

© DFKI - JK

A Probabilistic Interpretation

- **Sensitivity** is the probability of a positive test given that the patient has cancer: p(Classifier = YES | Cancer = YES)
- **Specificity** is the probability of a negative test given that the patient does not have cancer: p(Classifier = NO | Cancer = NO)
- **Precision** is the probability that a random patient from all those with a positive test indeed has cancer: p(Cancer = YES | Classifier = YES)

- For statistics geeks: Type I error = 1 – specificity and Type II error = 1 – sensitivity

- A formal explanation is given the following slide

© DFKI - JK

A Probabilistic Interpretation (more formally)

$$\begin{aligned}
\text{Sensitivity} &= \frac{\text{TP}}{\text{Actual YES}} \\
&= \frac{p(\text{Classifier} = \text{YES}, \text{Cancer} = \text{YES})}{p(\text{Cancer} = \text{YES})} \\
&= p(\text{Classifier} = \text{YES} | \text{Cancer} = \text{YES})
\end{aligned}$$

$$\begin{aligned}
\text{Specificity} &= \frac{\text{TN}}{\text{Actual NO}} \\
&= \frac{p(\text{Classifier} = \text{NO}, \text{Cancer} = \text{NO})}{p(\text{Cancer} = \text{NO})} \\
&= p(\text{Classifier} = \text{NO} | \text{Cancer} = \text{NO})
\end{aligned}$$

$$\begin{aligned}
\text{Precision} &= \frac{\text{TP}}{\text{Predicted YES}} \\
&= p(\text{Cancer} = \text{YES} | \text{Classifier} = \text{YES})
\end{aligned}$$

© DFKI - JK

F1 Score

- We must consider pairs of measures such as sensitivity / specificity or sensitivity / precision for interpreting the performance of a classifier

- The F1 score provides a single measure that summarizes the overall performance

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

- F1 is the harmonic mean between precision and recall (= sensitivity)
- F1 by design does not take TN into account !

© DFKI - JK

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. Explain the basic idea behind neural networks!
6. What is the difference between perceptrons and multi-layer perceptrons?
7. How does training of neural networks work?
8. How can we assess the performance of a learning algorithm?

© DFKI - JK

Recommended Reading

- S. Russell, P. Norvig: Artificial Intelligence – A Modern Approach, 3rd Edition, Chapters 18-21 (we only cover a small subset here, mostly sections 18.1-18.3, 18.7, 21.1., 21.2, http://aima.cs.berkeley.edu/

- Andrew Ng´s machine learning courses on coursera https://www.coursera.org/instructor/andrewng

- M. Deru, A. Ndiaye (from DFKI): Deep Learning mit TensorFlow, Keras und TensorFlow.js: Einstieg, Konzepte und Projekte mit Python, JavaScript und HTML5, Rheinwerk Computing, 2019.

© DFKI - JK