Artificial Intelligence

Machine Learning Basics

Prof. Dr. habil. Jana Koehler

Dr. Sophia Saller, M. Sc. Annika Engel

Summer 2020

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

Agenda

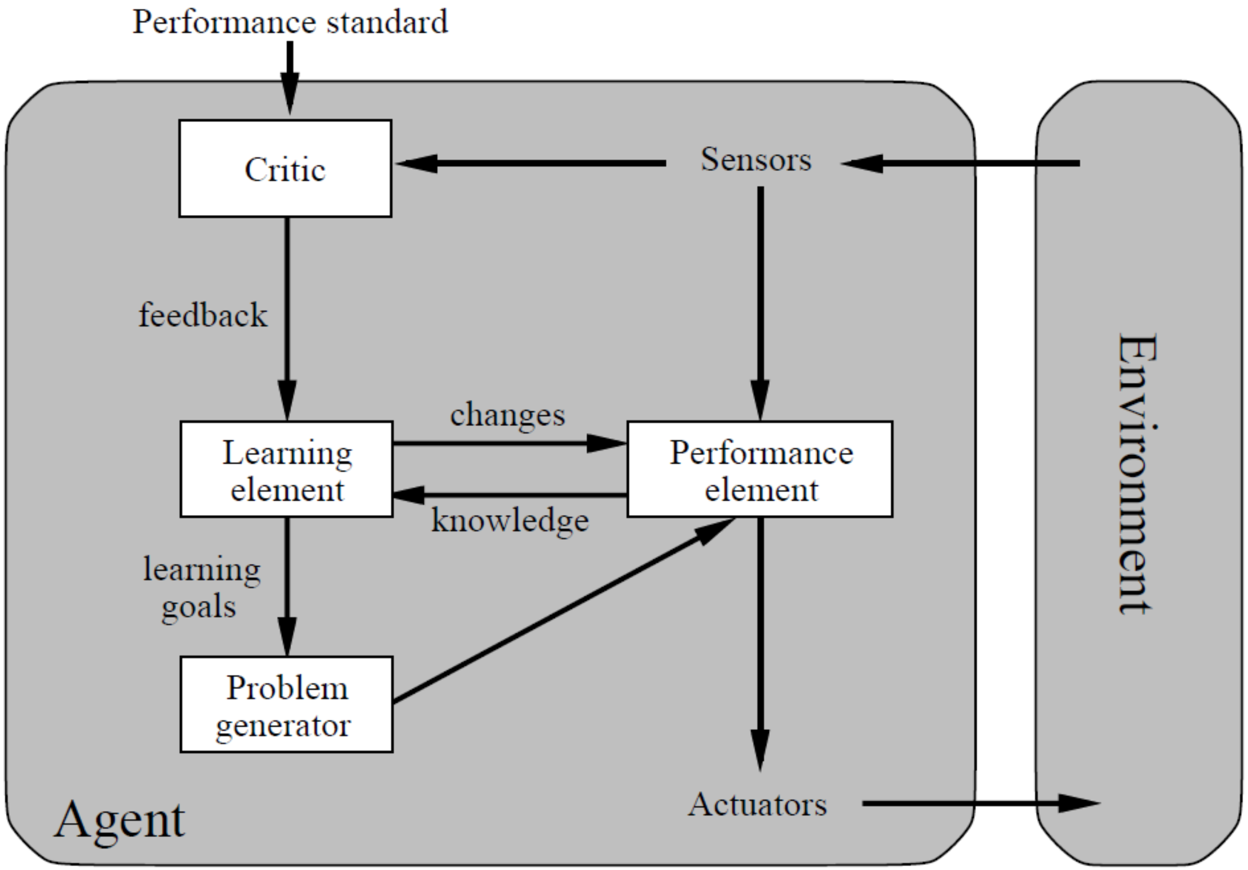
- The learning agent revisited
- Basic concepts of machine learning
- Decision Trees
  - An algorithm to find small trees
  - Entropy and information gain
- Evaluating a learned classifier
- Neural Networks
  - Perceptron
  - Multi-Layer Perceptron and backpropagation
  - Deep Learning

Recommended Reading

- AIMA Chapter 18: Learning from Examples
  - 18.1 Forms of Learning
  - 18.3 Learning Decision Trees
  - 18.7 Artificial Neural Networks
- AIMA Chapter 22: Natural Language Processing
  - 22.3.2 IR (Information Retrieval) System Evaluation

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



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

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?

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

Supervised Learning

- An example is a pair $(x, f(x))$
- The complete set of examples is called the **training set**

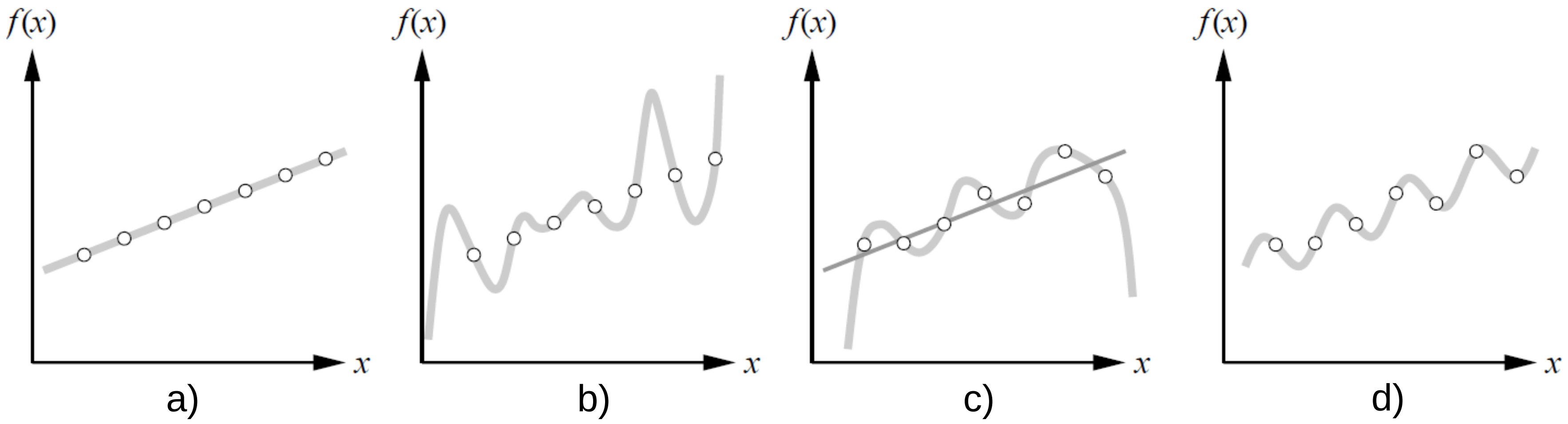
- Pure inductive inference: for a collection of examples for f, return a function h (**hypothesis**) that approximates f

- The function h is member of a **hypothesis space** H
  - good hypothesis should **generalize** the data well, i.e., predicts unseen examples correctly
  - a hypothesis is **consistent** with the data set if it agrees with all the data

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

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

Which is preferred by Ockham´s razor?

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 to learn f (marketing, sales)

Decision Tree Learning

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
- Discrete-valued functions lead 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

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

Boolean Decision Tree when Predicting

Input

A vector of input attributes $A = \{a_1, \ldots, a_n\}$ and a decision tree D

Output

Value of the goal predicate for vector A

- **Boolean decision tree D is a Boolean function**

$$D\colon\ A \mapsto \{\mathrm{TRUE}, \mathrm{FALSE}\}$$

Properties of (Boolean) Decision Trees

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

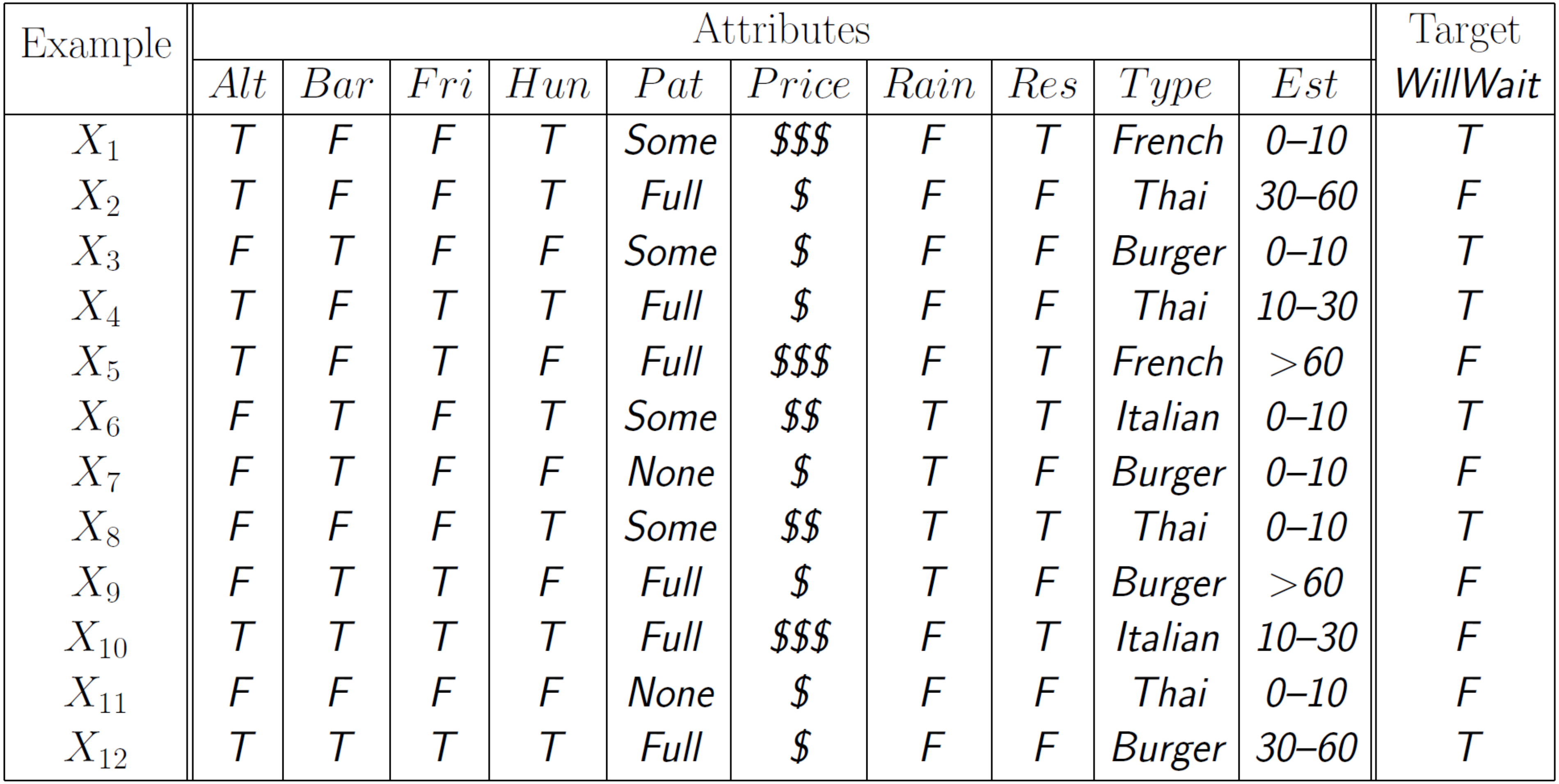
The Restaurant Example

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? (True/False)
- *Hungry*: Am I hungry? (True/False)
- *Reservation*: Have I made a reservation? (True/False)
- *Bar*: Does the restaurant have a bar to wait in? (True/False)
- *Fri/Sat*: Is it Friday or Saturday? (True/False)
- *Raining*: Is it raining outside? (True/False)
- *Price*: How expensive is the food? ($, $$, $$$)
- *Type*: What kind of restaurant is it? (French, Italian, Thai, Burger)

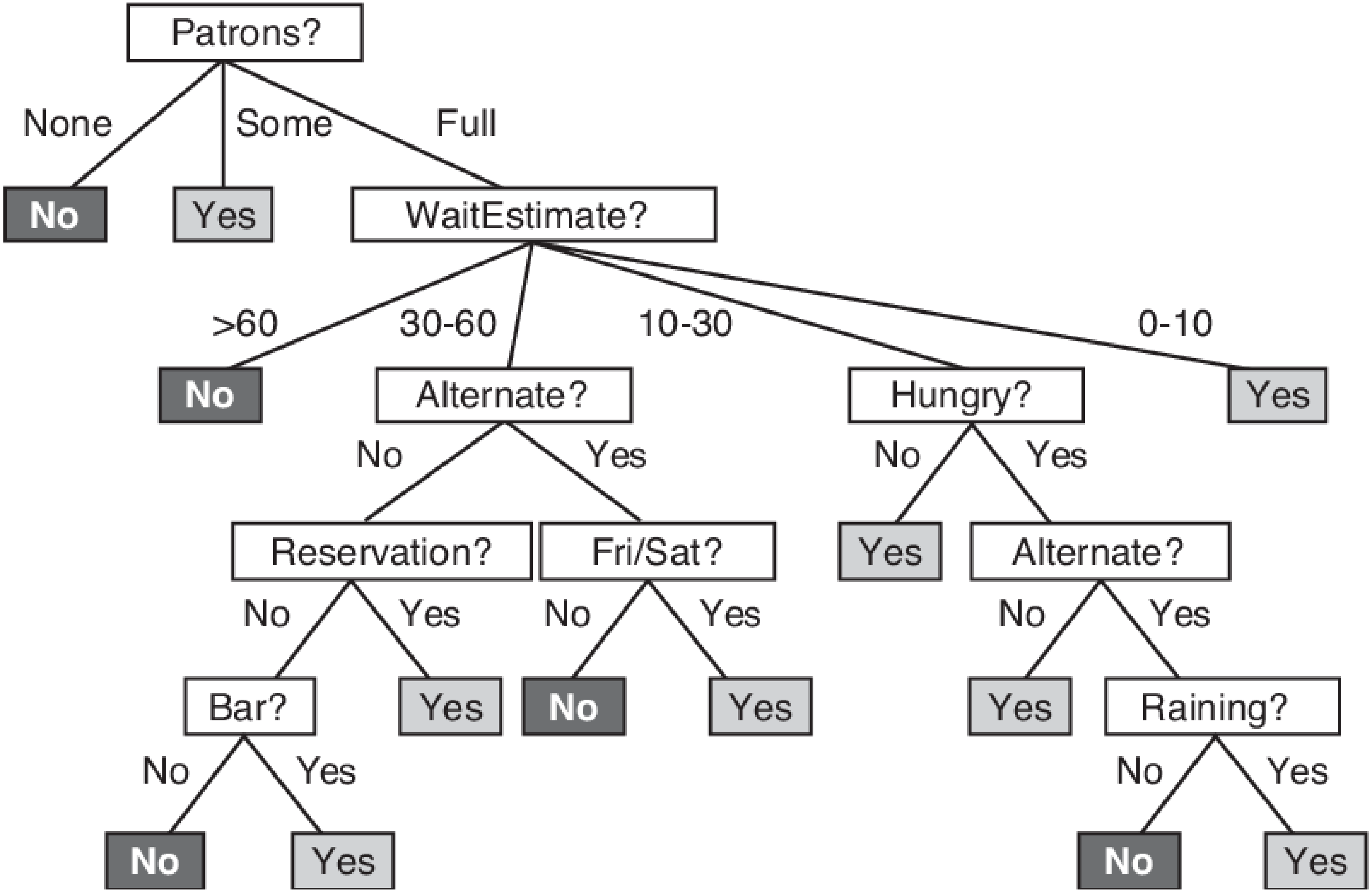
The Training Set



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)

A Boolean Decision Tree



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

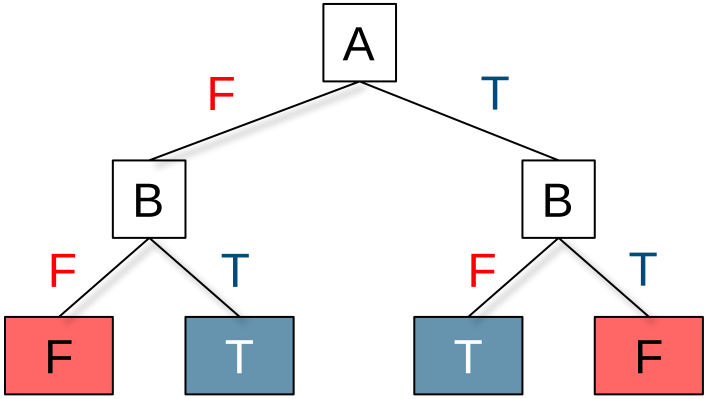
- Whereas the resulting tree will correctly classify all given examples, it will not say much about other cases

- It just memorizes the observations and does not generalize
  - “Overfitting”

Memorizing 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

Hypotheses Spaces

- How many distinct decision trees can we built with n Boolean variables (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 $n = 2^{2^n}$

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

➢ For the 10 attributes of the restaurant example, we have $2^{1024} = 10^{308}$ potential candidate decision trees

Expressiveness of Decision Trees

- Each decision tree hypothesis for the *WillWait* goal predicate can be seen as 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

- Any Boolean decision tree is logically equivalent to the assertion that the goal predicate is true if and only if the input attributes satisfy one of the paths leading to a true leaf node
- Limitation:

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

  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

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

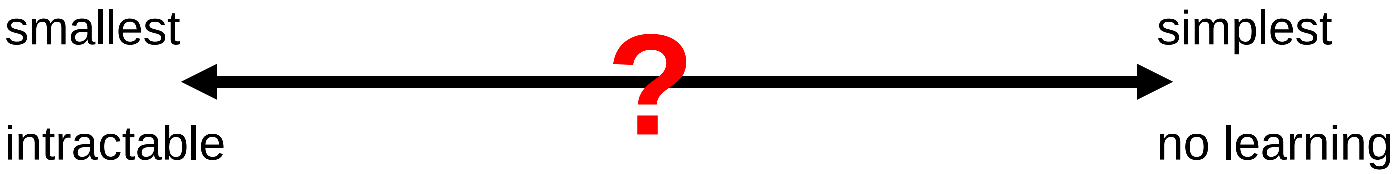
  Majority function: $p(x) = \begin{cases} 1, & \text{half of inputs are 1} \\ 0, & \text{otherwise} \end{cases}$
- There is no consistent representation that is compact for all possible Boolean functions

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:



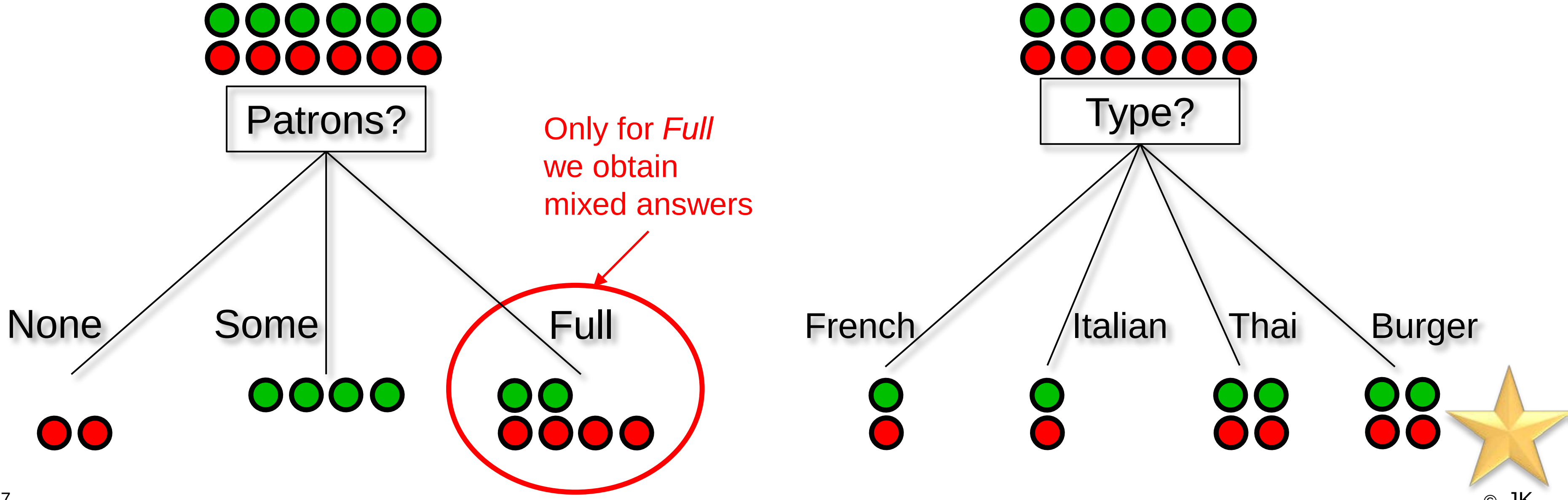
➢ We can devise a decision tree learning algorithm that generates “smallish“ trees

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

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

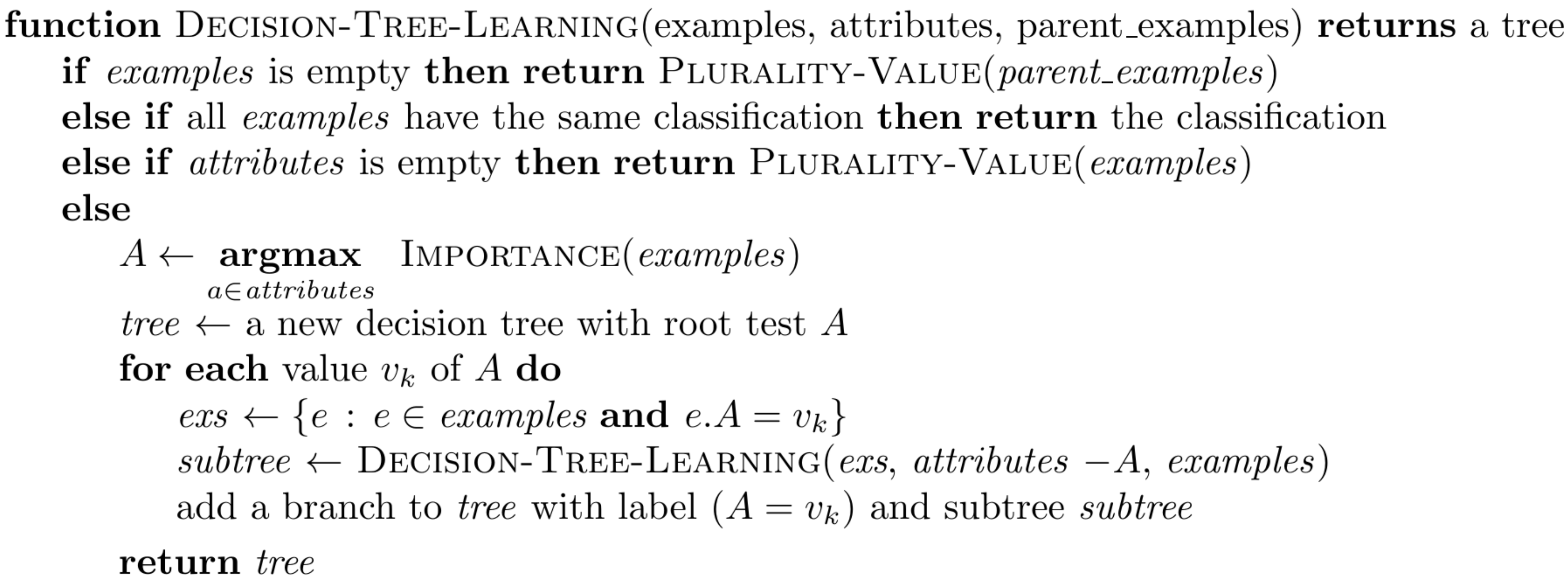


Recursive Learning Process

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*

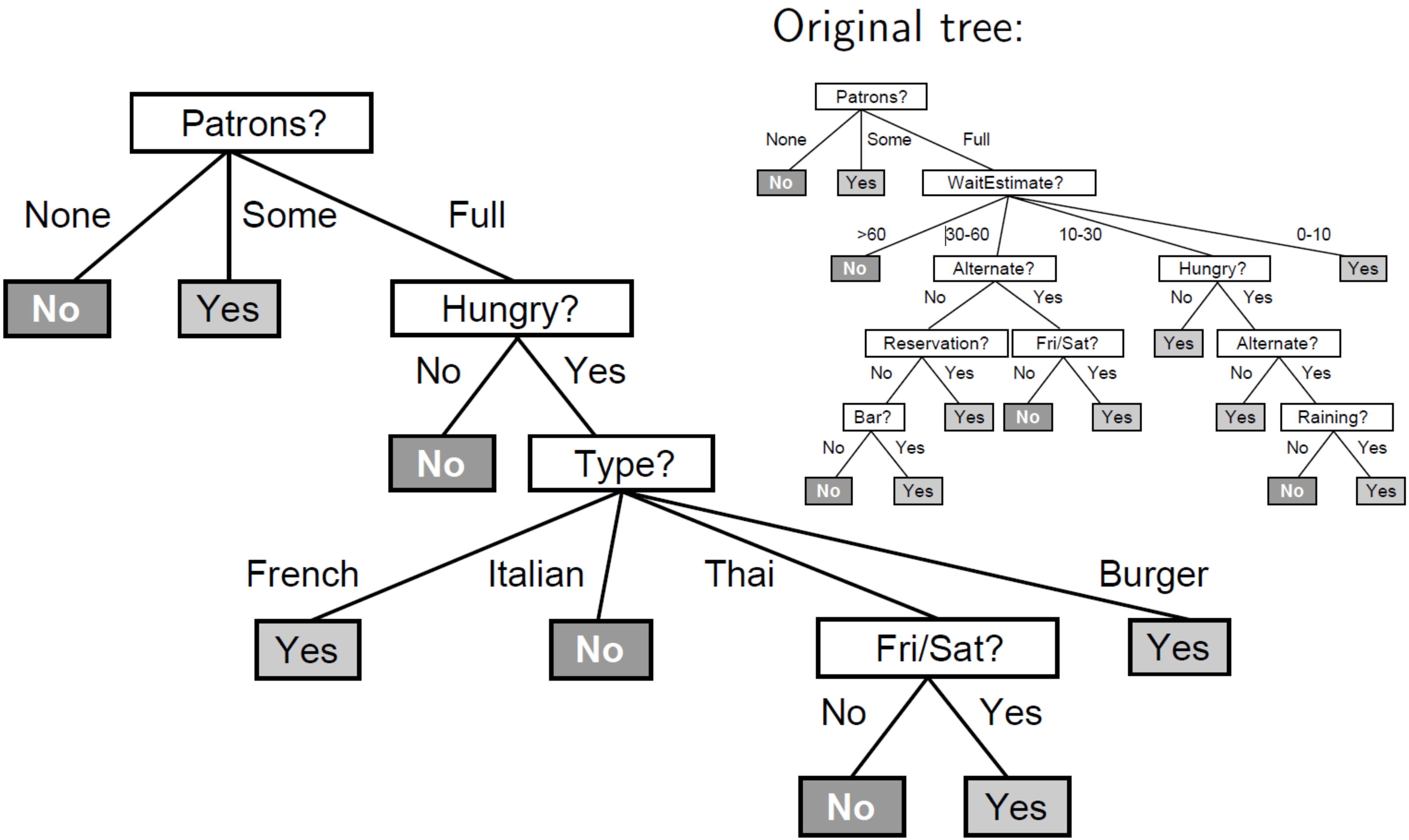
The Decision-Tree-Learning Algorithm



```
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
        $A \leftarrow \operatorname{argmax}_{a \in attributes}$ IMPORTANCE(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 $tree$
```

- 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



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

- Some tests (Raining, Reservation) are not included since the algorithm can classify the examples without them

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 “rather useless" (Type?) to implement the IMPORTANCE function

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

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 the four equally probable outcomes

Entropy

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

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

$\log_n\left(\frac{a}{b}\right) = \log_n(a) - \log_n(b)$, 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:

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

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

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

Influence of Testing an Attribute on Entropy

- The restaurant example has $p = n = 6$, $B(WillWait)$ = 1 bit
  - ➢ $\frac{p}{p+n} = 0.5$ since 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

Information Provided by an Attribute

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

- Each subset E_k has p_k positive and n_k negative examples, so along this branch we need an additional
$$B(p_k/(p_k + n_k))$$
bits of information to answer the question

- A randomly chosen example from the training set has the k^{th} value for the attribute with probability
$$(p_k + n_k)/(p + n)$$

Expected Entropy Remaining after Testing Attribute A

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

$k = 1, .. d$ distinct values of attribute A with number of positive outcomes p_k and number of negative outcomes n_k

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

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

IMPORTANCE Function uses Information Gain

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

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

3 different values of the Patrons attribute
None:2 negative, Some:4 positive, Full:6 (2 positive+4 negative)

B(0.5)

$$Gain(Patrons) = \overset{B(0.5)}{1} - \left[\overbrace{\frac{2}{12}B\left(\frac{0}{2}\right) + \frac{4}{12}B\left(\frac{4}{4}\right) + \frac{6}{12}B\left(\frac{2}{6}\right)}\right] \approx 0.541 \text{ bits}$$

$$Gain(Type) = 1 - \left[\frac{2}{12}B\left(\frac{1}{2}\right) + \frac{2}{12}B\left(\frac{1}{2}\right) + \frac{4}{12}B\left(\frac{2}{4}\right) + \frac{4}{12}B\left(\frac{2}{4}\right)\right] \approx 0 \text{ bits}$$

Advantages of Decision Trees

- + One possibility for representing (Boolean) functions
- + Small decision tree 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 only little data preparation
- + Perform well even on larger datasets

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 decision tree
- 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 decision trees leads us to Random Forests Learning, which addresses these issues

Evaluating a Learned Classifier

Training and Testing

1. Collect a large number of examples

2. Divide it into two disjoint sets: the training set and the test set

3. Use the training set to generate h

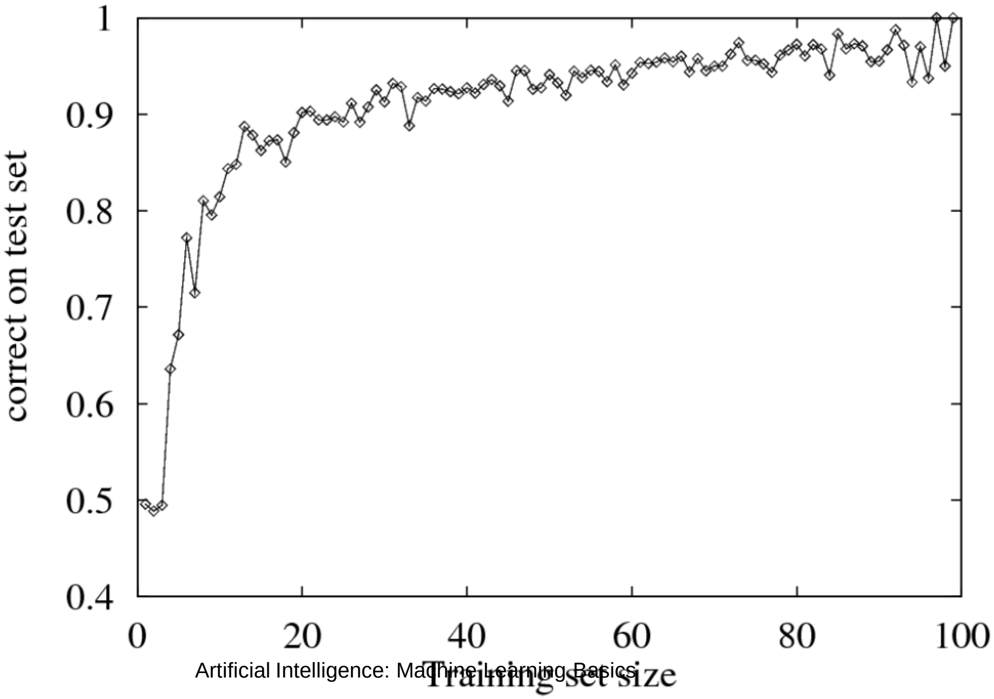
4. Measure the percentage of examples of the test set that are correctly classified by h

5. Repeat the process for randomly-selected training sets of different sizes

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 corresponds to how much of a test set is correctly classified as a function of training set size



Separation of Training and Test Set

- The training and test sets must be kept separate!

- **Common error: Retraining an algorithm using examples from the test set and then repeating the tests**

- By doing this, knowledge about the test set gets stored in the algorithm, and the training and test sets are no longer independent

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

Evaluating the Quality of Learning Algorithms

4 possible outcomes for each test example

1) **TRUE Positive** (TP)

   is an outcome where the model correctly predicts the positive class

2) **TRUE Negative** (TN)

   is an outcome where the model correctly predicts the negative class

3) **FALSE Positive** (FP)

   is an outcome where the model incorrectly predicts the positive class

4) **FALSE Negative** (FN)

   is an outcome where the model incorrectly predicts the negative class

Accuracy – Performance of Correct Classifications

Accuracy is given by $$\frac{\mathbf{TP+TN}}{\mathbf{Total}}$$

- What is the proportion of correct classifications compared to the total number of tests?

Recall – Performance on YES Instances

$$\textbf{Recall is given by} \quad \frac{\textbf{TP}}{\textbf{Actual Yes}} = \frac{\textbf{TP}}{\textbf{TP+FN}}$$

- When the true value is YES, how often does the classifier predict YES?
- For example, recall is the estimated probability that a randomly selected relevant document is retrieved in a search
- Recall is also called True-Positive Rate or Sensitivity

Specificity – Performance on NO Instances

Specificity is given by $\dfrac{\textbf{TN}}{\textbf{Actual NO}} = \dfrac{\textbf{TN}}{\textbf{TN+FP}}$

- When the true value is NO, how often does the classifier predict NO?
- True negative rate: proportion of actual negatives that the classifier correctly identifies

Precision – Performance when TN cannot be assessed

Precision is given by $$\frac{\textbf{TP}}{\textbf{Predicted Yes}} = \frac{\textbf{TP}}{\textbf{TP}+\textbf{FP}}$$

- Precision is the estimated probability that a randomly selected retrieved document is relevant
- 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?

Example

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

- Accuracy is given by $\frac{\text{TP+TN}}{\text{Total}} = \frac{150}{165} = 91\%$
- Recall is given by $\frac{\text{TP}}{\text{Actual Yes}} = \frac{\text{TP}}{\text{TP+FN}} = \frac{100}{100+5} = 95\%$
- Specificity is given by $\frac{\text{TN}}{\text{Actual NO}} = \frac{\text{TN}}{\text{TN+FP}} = \frac{50}{60} = 83\%$
- Precision is given by $\frac{\text{TP}}{\text{Predicted Yes}} = \frac{\text{TP}}{\text{TP+FP}} = \frac{100}{100+10} = 91\%$

A Probabilistic Interpretation

- **Recall** 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
  Type II error   1 - recall
- A formal explanation is given on the following slide

A Probabilistic Interpretation

Prediction by the classifier: Does a patient have cancer?

Recall is given by $\frac{\text{TP}}{\text{Actual Yes}} = \frac{p(\text{Classifier} = \text{YES}, \text{Cancer} = \text{YES})}{p(\text{Cancer} = \text{YES})}$

$= p(\text{Classifier} = \text{YES} \mid \text{Cancer} = \text{YES})$

Specificity is given by $\frac{\text{TN}}{\text{Actual NO}} = \frac{p(\text{Classifier} = \text{NO}, \text{Cancer} = \text{NO})}{p(\text{Cancer} = \text{NO})}$

$= p(\text{Classifier} = \text{NO} \mid \text{Cancer} = \text{NO})$

Precision is given by $\frac{\text{TP}}{\text{Predicted YES}} = p(\text{Cancer} = \text{YES} \mid \text{Classifier} = \text{YES})$

F1 Score

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

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

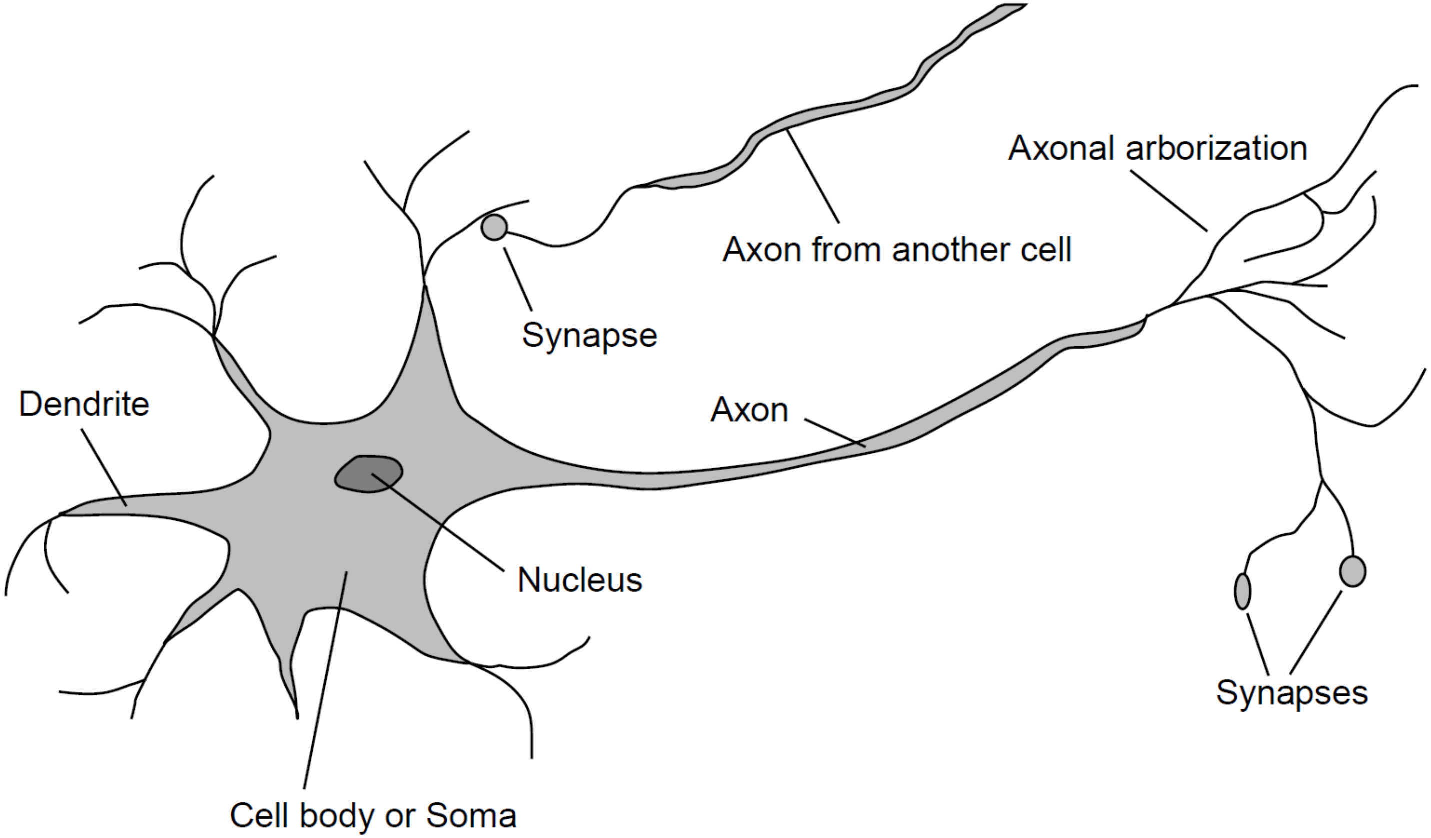
$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- F1 is the harmonic mean between precision and recall
- Maximum value of F1 is 1 (perfect precision and recall)
- F1 by design does not take TN into account !

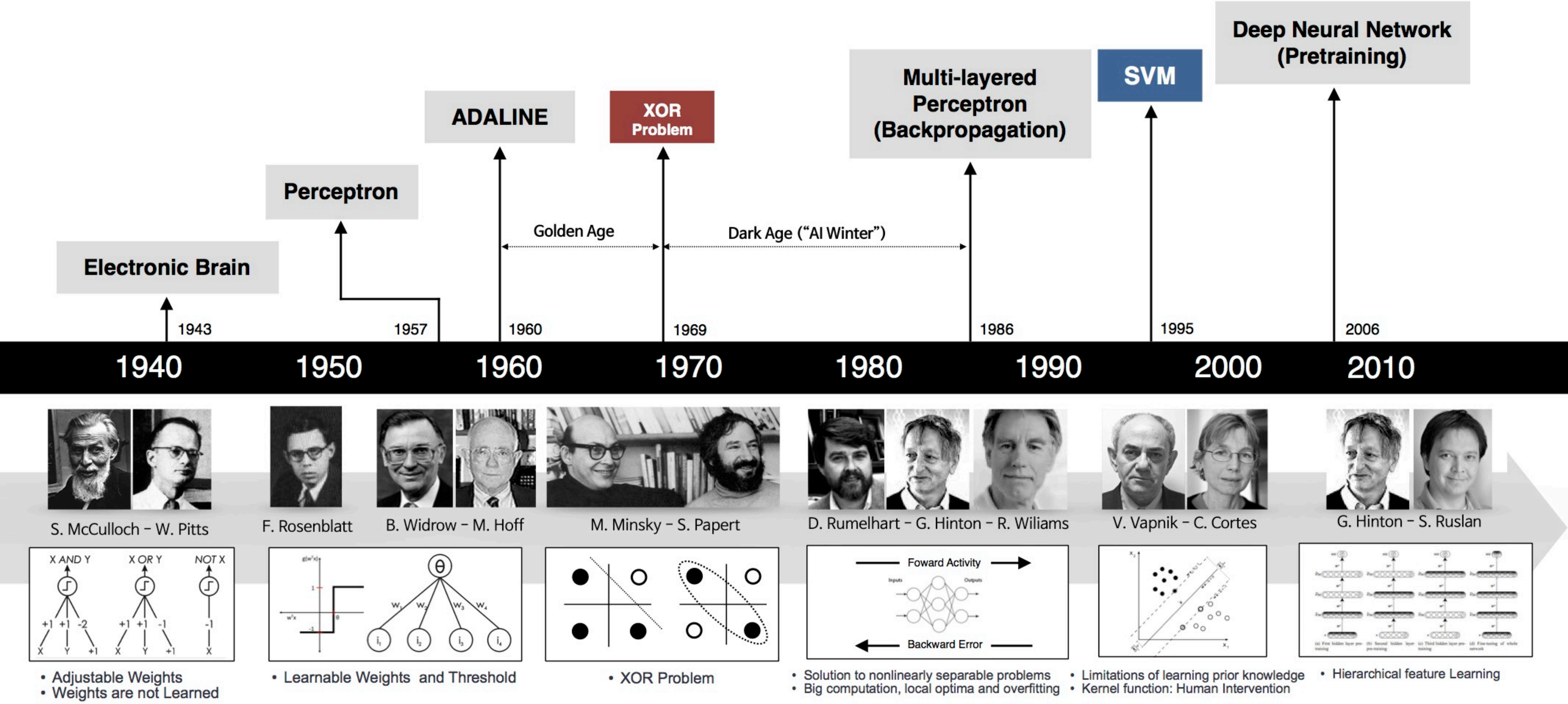
Neural Networks

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



Neural Networks Research



Electronic Brain — 1943

Perceptron — 1957

ADALINE — 1960

XOR Problem — 1969

Multi-layered Perceptron (Backpropagation) — 1986

SVM — 1995

Deep Neural Network (Pretraining) — 2006

Golden Age

Dark Age ("AI Winter")

1940 1950 1960 1970 1980 1990 2000 2010

S. McCulloch - W. Pitts
- Adjustable Weights
- Weights are not Learned

F. Rosenblatt
- Learnable Weights and Threshold

B. Widrow - M. Hoff

M. Minsky - S. Papert
- XOR Problem

D. Rumelhart - G. Hinton - R. Wiliams
- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting

V. Vapnik - C. Cortes
- Limitations of learning prior knowledge
- Kernel function: Human Intervention

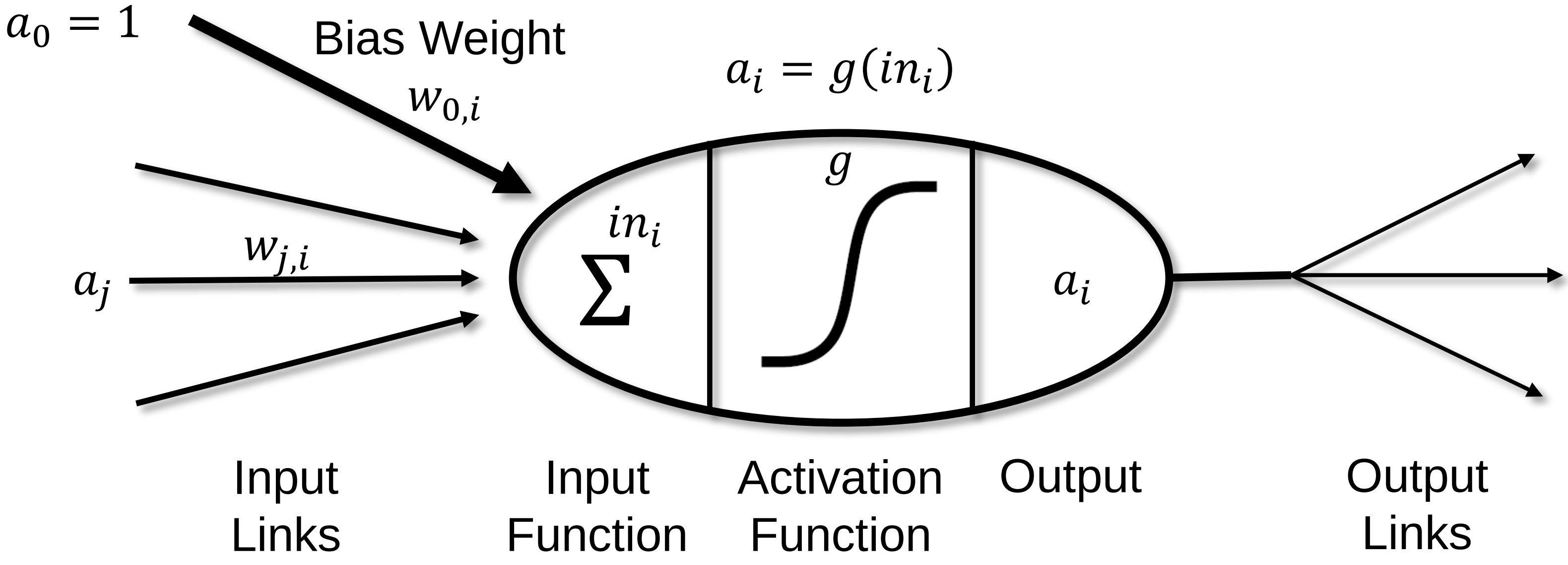
G. Hinton - S. Ruslan
- Hierarchical feature Learning

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

McCulloch-Pitts “Unit”

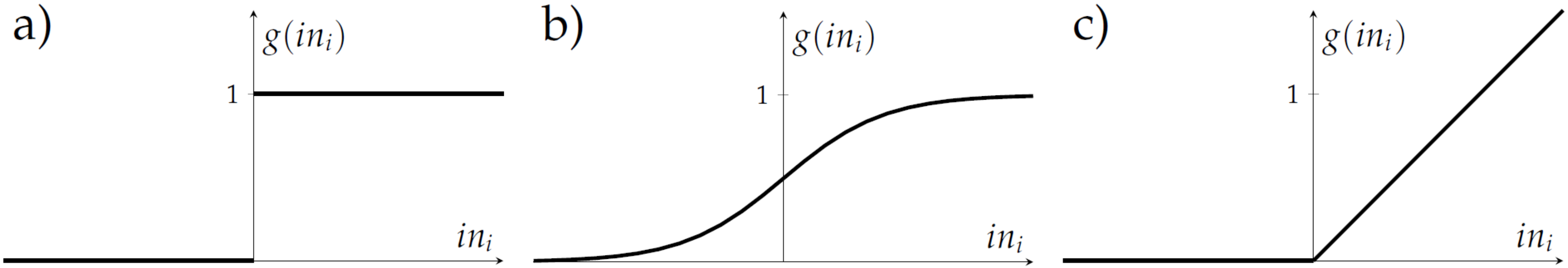
- Output is a “squashed” linear function of the inputs

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



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

Activation Functions



a) is a step function or threshold function

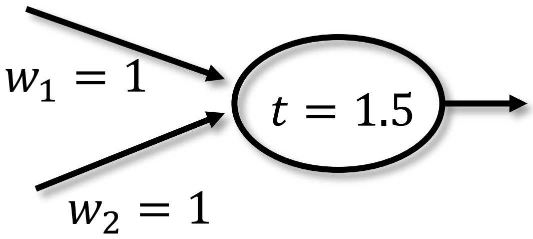
b) is a sigmoid function $1/(1 + e^{-x})$

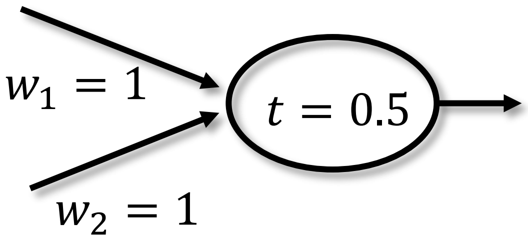
c) is a ReLu function $\max(0, x)$

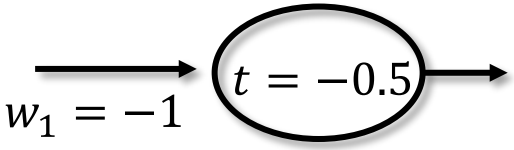
Implementing Logical Functions

- McCulloch and Pitts: every Boolean function can be implemented

- Using a step function with step value t, we obtain







AND

$$1 \cdot 1 + 1 \cdot 1 \;\; => 1$$
$$1 \cdot 1 + 1 \cdot 0 \;\; => 0$$

OR

$$1 \cdot 0 + 1 \cdot 0 \;\; => 0$$
$$1 \cdot 0 + 1 \cdot 1 \;\; => 1$$

NOT

$$-1 \;\cdot 0 => 1$$
$$-1 \;\cdot 1 => 0$$

Network Structures

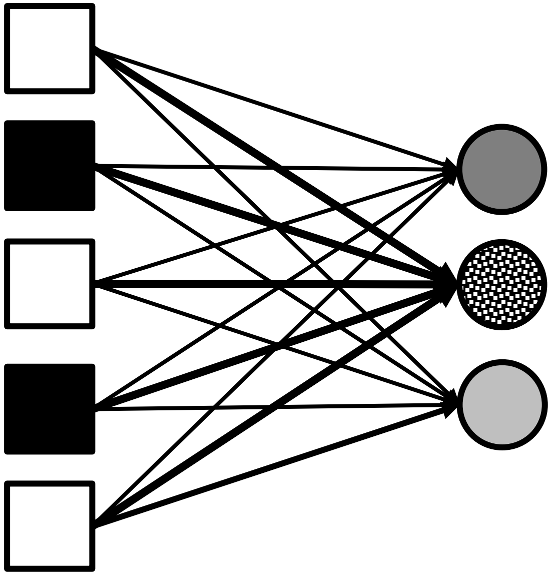
- Feed-forward networks:
  - Single-layer perceptron
  - Multi-layer perceptron

- Feed-forward networks are directed acyclic graphs
  - they have no internal state other than the weights

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

Single-Layer Perceptron

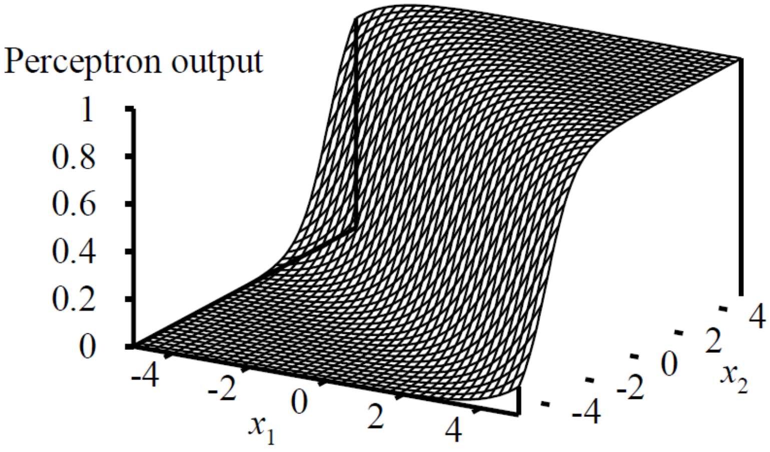
- Output units all operate separately - no shared weights
- Adjusting weights moves the location, orientation, and steepness of cliff



Input Units

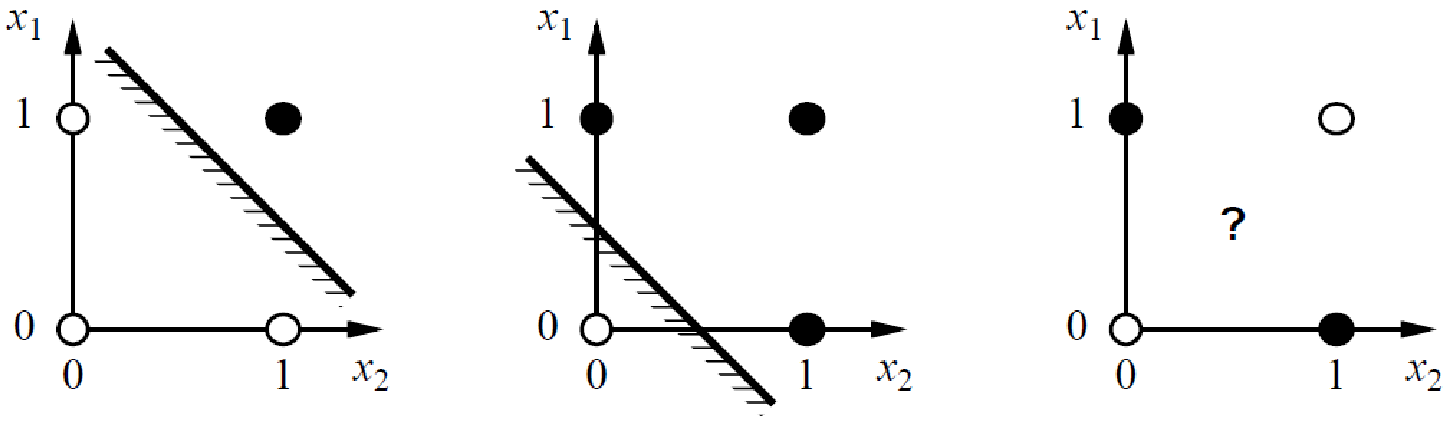
$W_{j,i}$

Output Units



Expressiveness of Perceptrons

- Consider a perceptron with the step function g (Rosenblatt 1957/60)
  - which can represent AND, OR, NOT, majority, etc., but not XOR
  - which represents a linear separator in input space $\sum_j w_j x_j > 0 \text{ or } W \cdot x > 0$



a) x_1 AND x_2 b) x_1 OR x_2 c) x_1 XOR x_2

Multi-Layer Perceptron / Feed-forward Network

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

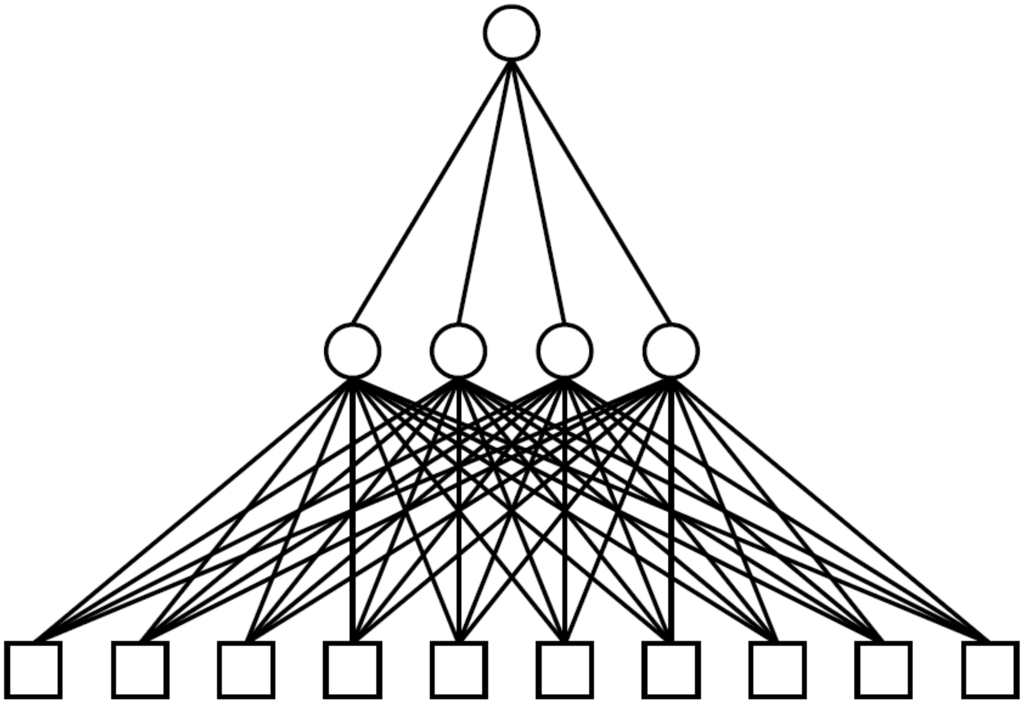
Output units a_k

$w_{j,k}$

Hidden units a_j

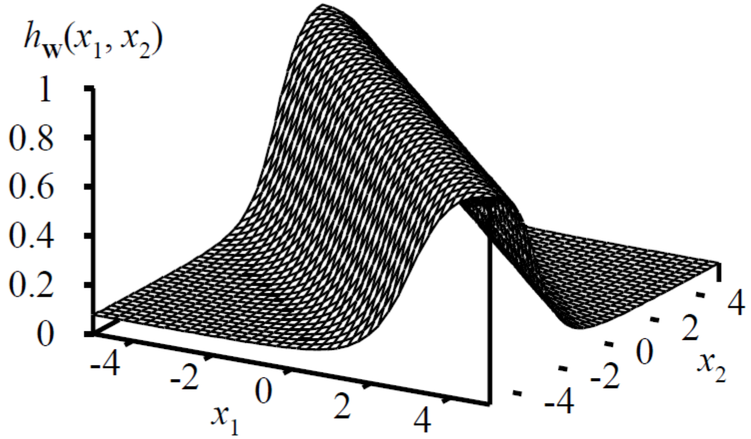
$w_{i,j}$

Input units a_i

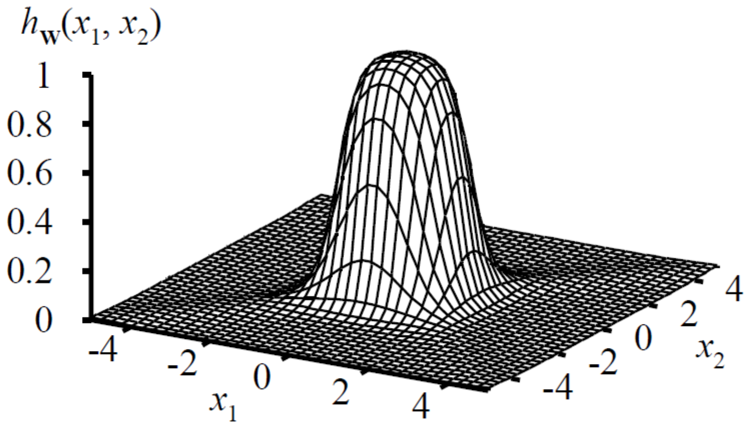


Expressiveness of Multi-Layer Perceptron

- With a single, sufficiently large hidden layer, it is possible to represent any continuous function of the inputs with arbitrary accuracy
- With two hidden layers, discontinuous functions can be represented



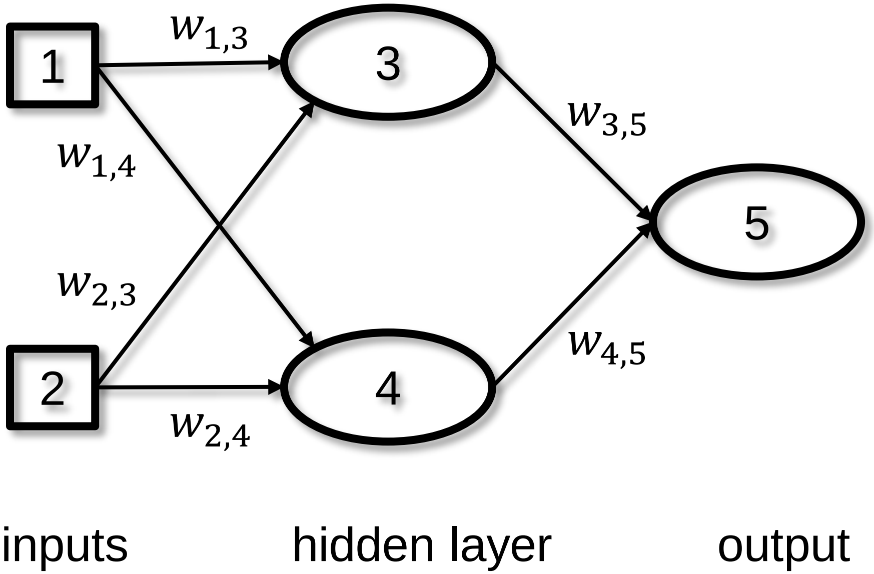
(a) The result of combining two opposite-facing soft threshold functions to produce a ridge.



(b) The result of combining two ridges to produce a bump.

Learning in Feed-forward Network

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



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

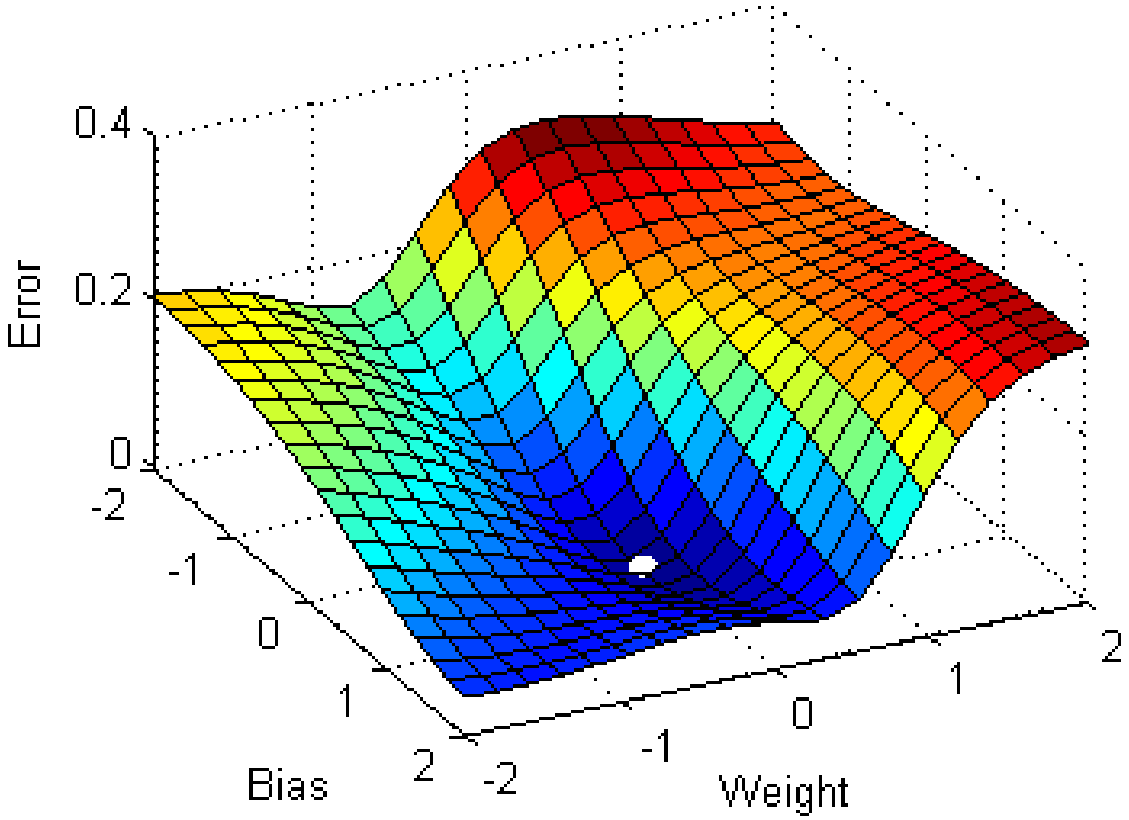
Idea of Backpropagation Learning for Multi-Layer Networks

- Learning = minimizing the squared error „cost“ function by performing optimization search using **gradient descent**

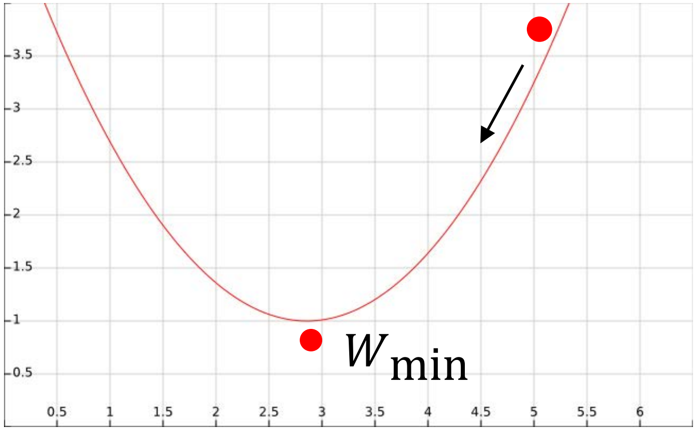
- Split error across hidding units: node *j* is responsible for some fraction of the error in each of the output nodes to which it connects
  - error is divided according to the strength of the connection between a hidden node and the output node

- Continue to propagate the error to the previous layer and adjust weights between two layers until input layer is reached

Illustration of Gradient Descent



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



Derivation of the Error and the “Delta”

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

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

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

Then, it yields

$$\frac{\delta E}{\delta w_{j,k}} = -(y_k - a_k)\frac{\delta a_k}{\delta w_{j,k}} = -(y_k - a_k)\frac{\delta g(in_k)}{\delta w_{j,k}} = -(y_k - a_k)g'(in_k) \cdot \frac{\delta in_k}{\delta w_{j,k}}$$

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

Backpropagation Learning

- Adjust weights leading to output layer:

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

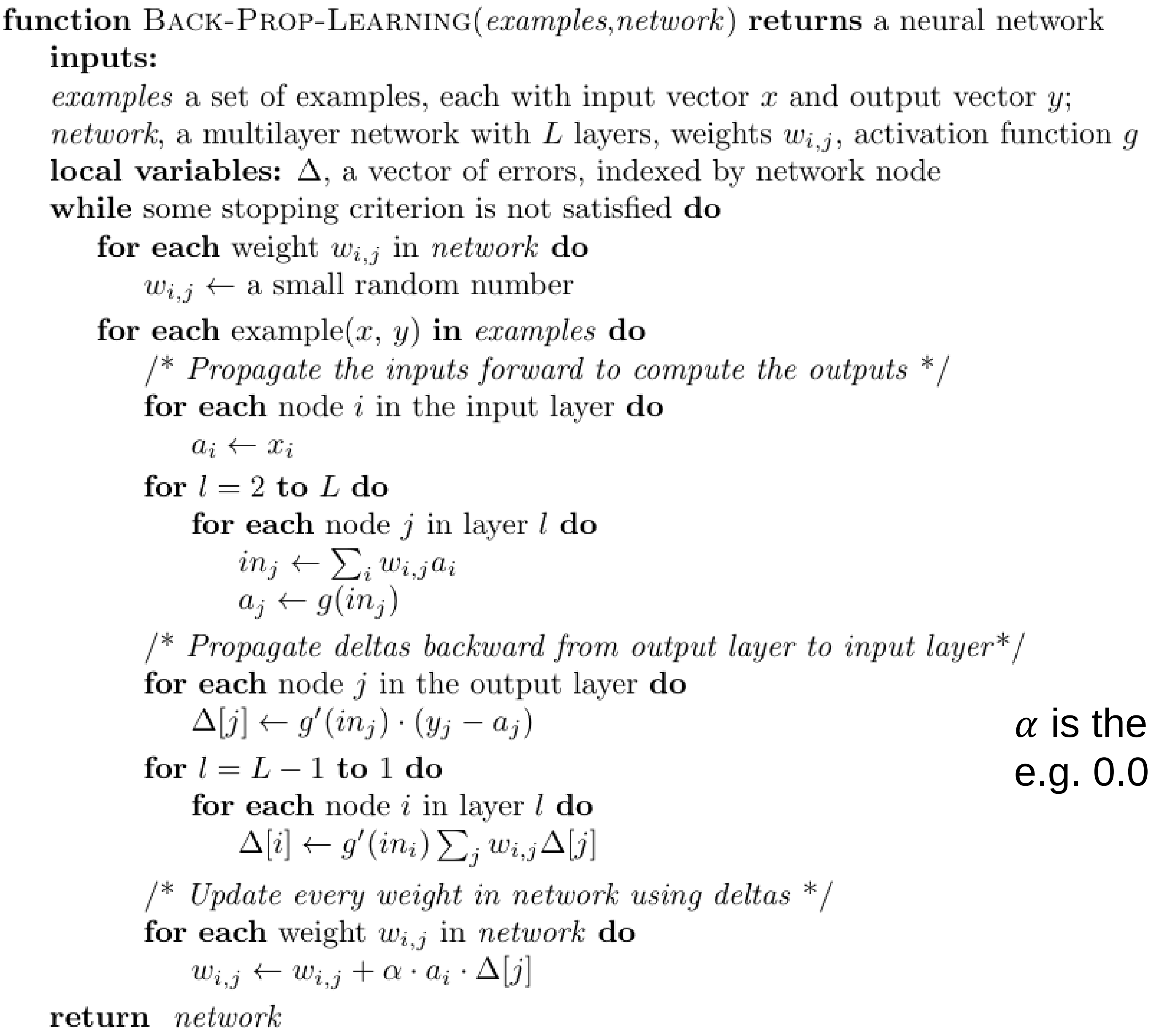
α - learning rate, a small constant e.g. 0.02
$g´$ - derivative of activation function g

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

- Adjust weights for each node j in hidden layer:

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

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

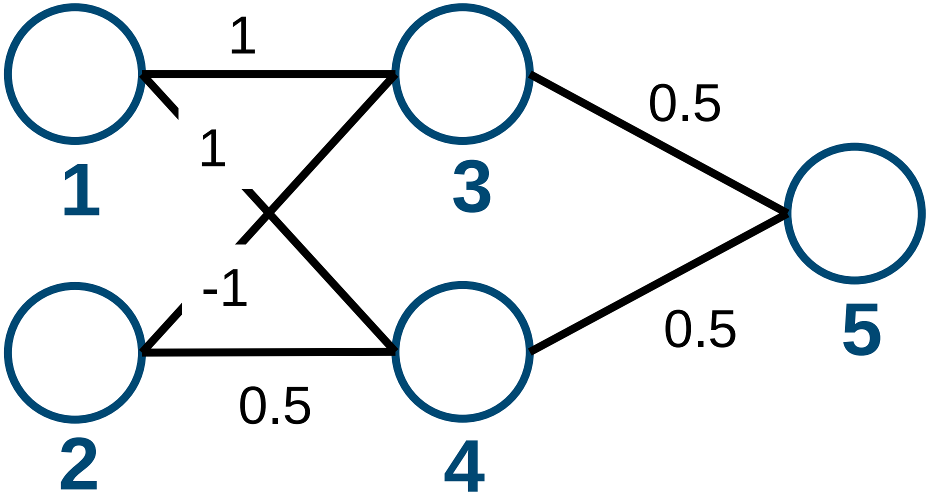


```
function BACK-PROP-LEARNING(examples, network) returns a neural network
    inputs:
    examples a set of examples, each with input vector $x$ and output vector $y$;
    network, a multilayer network with $L$ layers, weights $w_{i,j}$, activation function $g$
    local variables: $\Delta$, a vector 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
```

α is the learning rate, e.g. 0.02

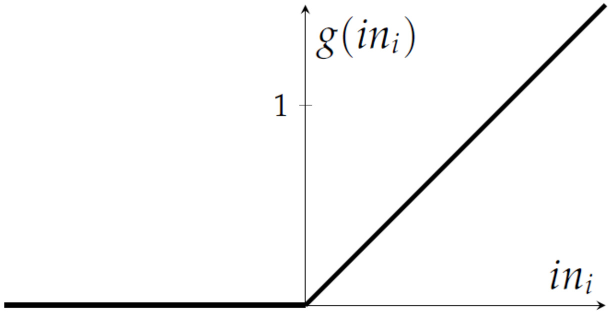
Example Neural Network Learning

Network structure



Learning rate: α = 1

g-function



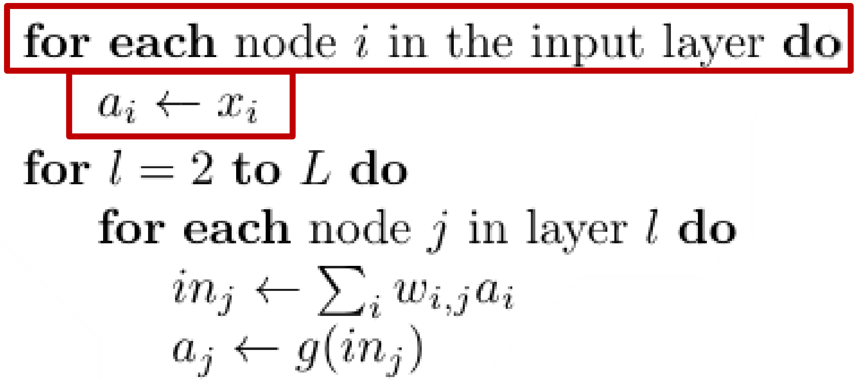
Derivative of the ReLu function $\max(0, x)$

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

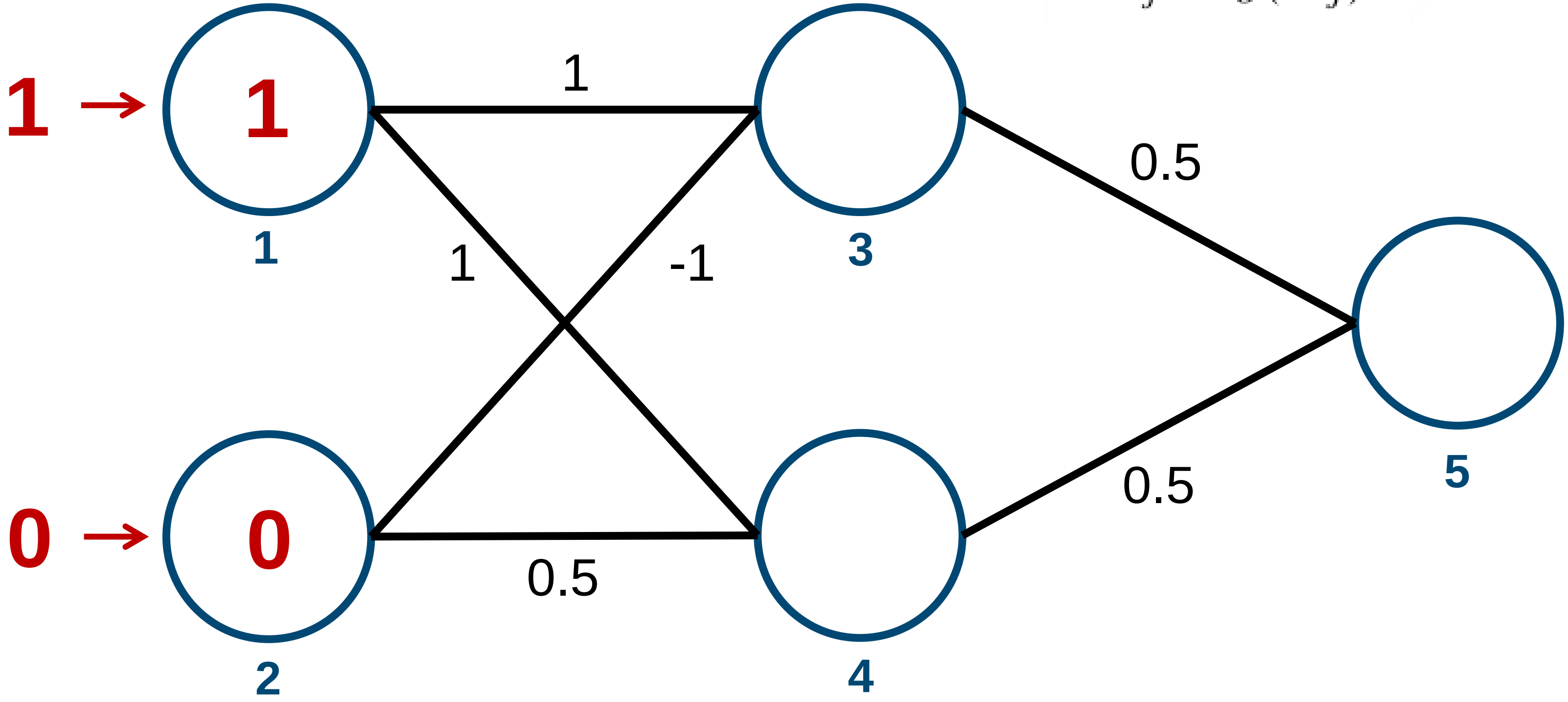
Note that when $x = 0$, the derivative does not exist, but one can define a value, common practice 0, 0.5, or 1.

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

Forward Propagation of Inputs
Input Layer



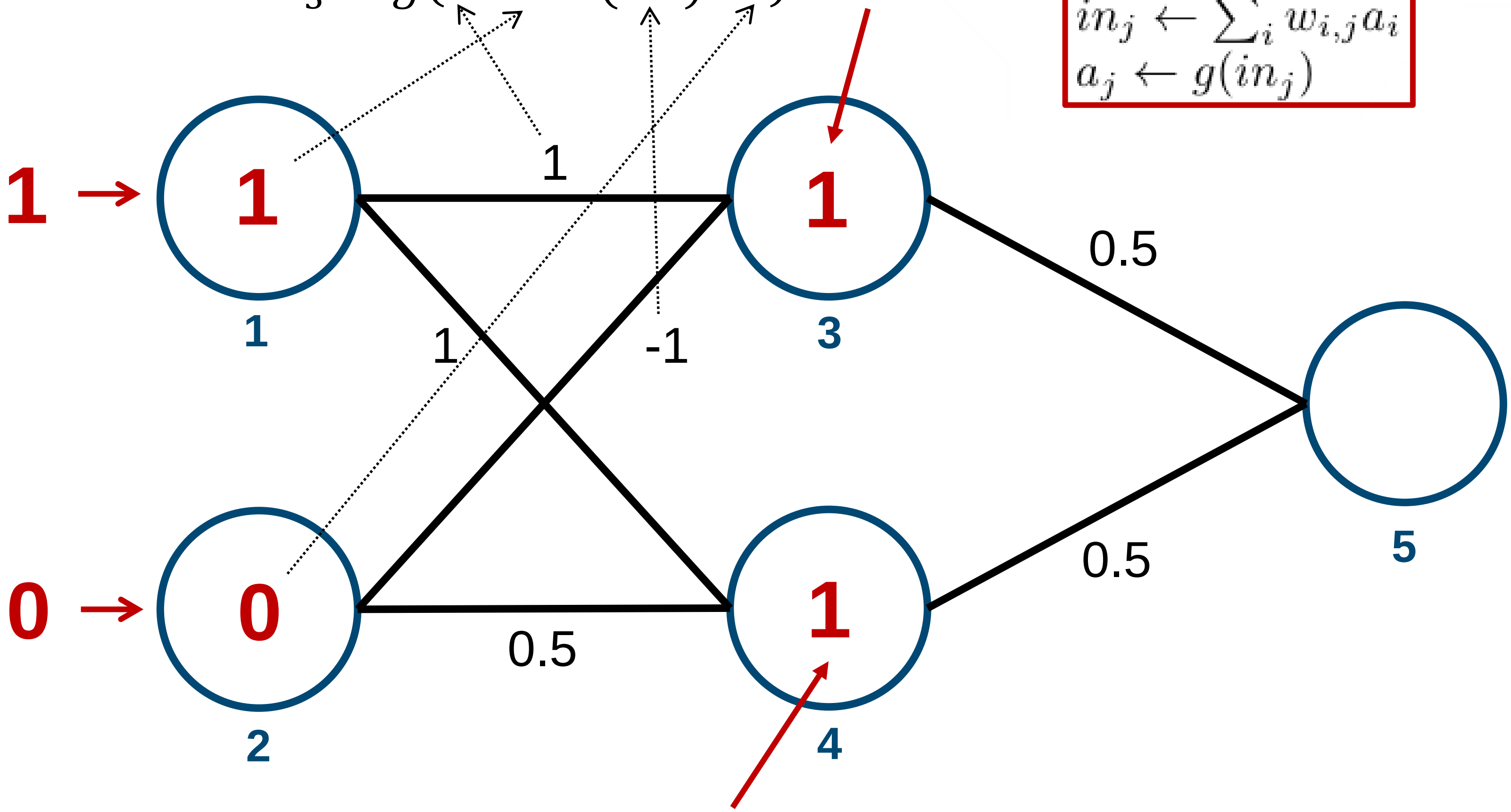
```
for each node $i$ in the input layer do
    $a_i \leftarrow x_i$
for $l = 2$ to $L$ do
    for each node $j$ in layer $l$ do
        $in_j \leftarrow \sum_i w_{i,j} a_i$
        $a_j \leftarrow g(in_j)$
```



Forward Propagation of Inputs Hidden Layer

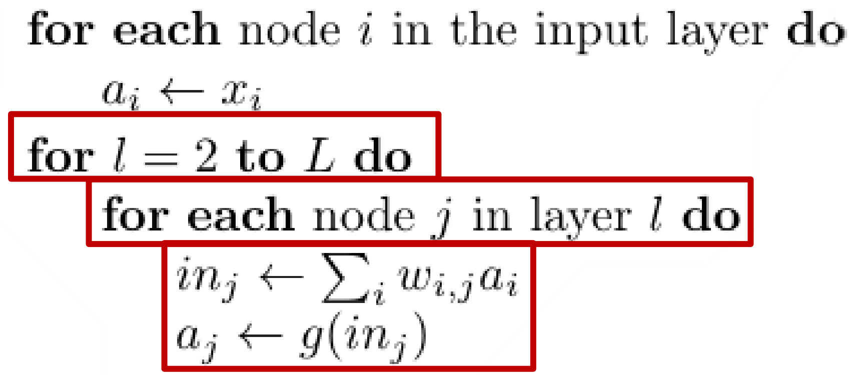
```
for each node $i$ in the input layer do
    $a_i \leftarrow x_i$
for $l = 2$ to $L$ do
    for each node $j$ in layer $l$ do
        $in_j \leftarrow \sum_i w_{i,j} a_i$
        $a_j \leftarrow g(in_j)$
```

$$a_3 = g(1 \cdot 1 + (-1) \cdot 0) = 1$$

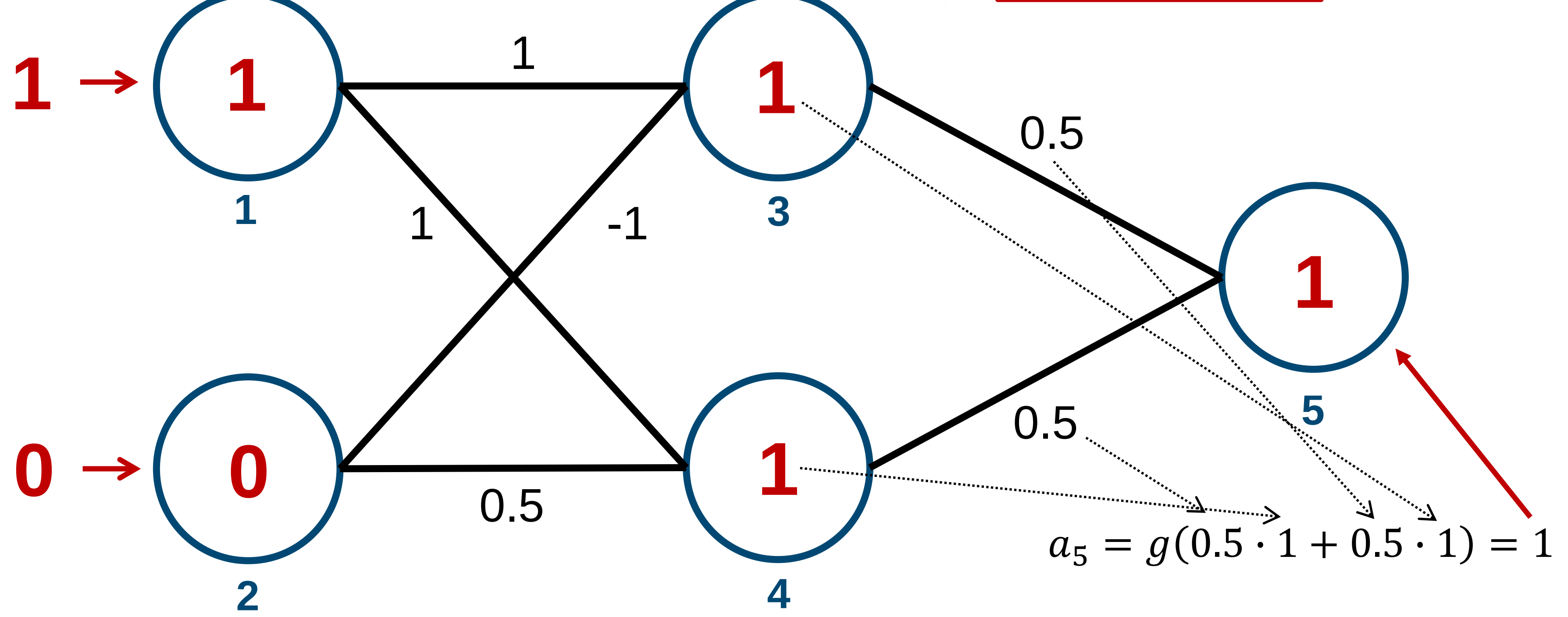


$$a_4 = g(1 \cdot 1 + 0.5 \cdot 0) = 1$$

Forward Propagation of Inputs Output Layer



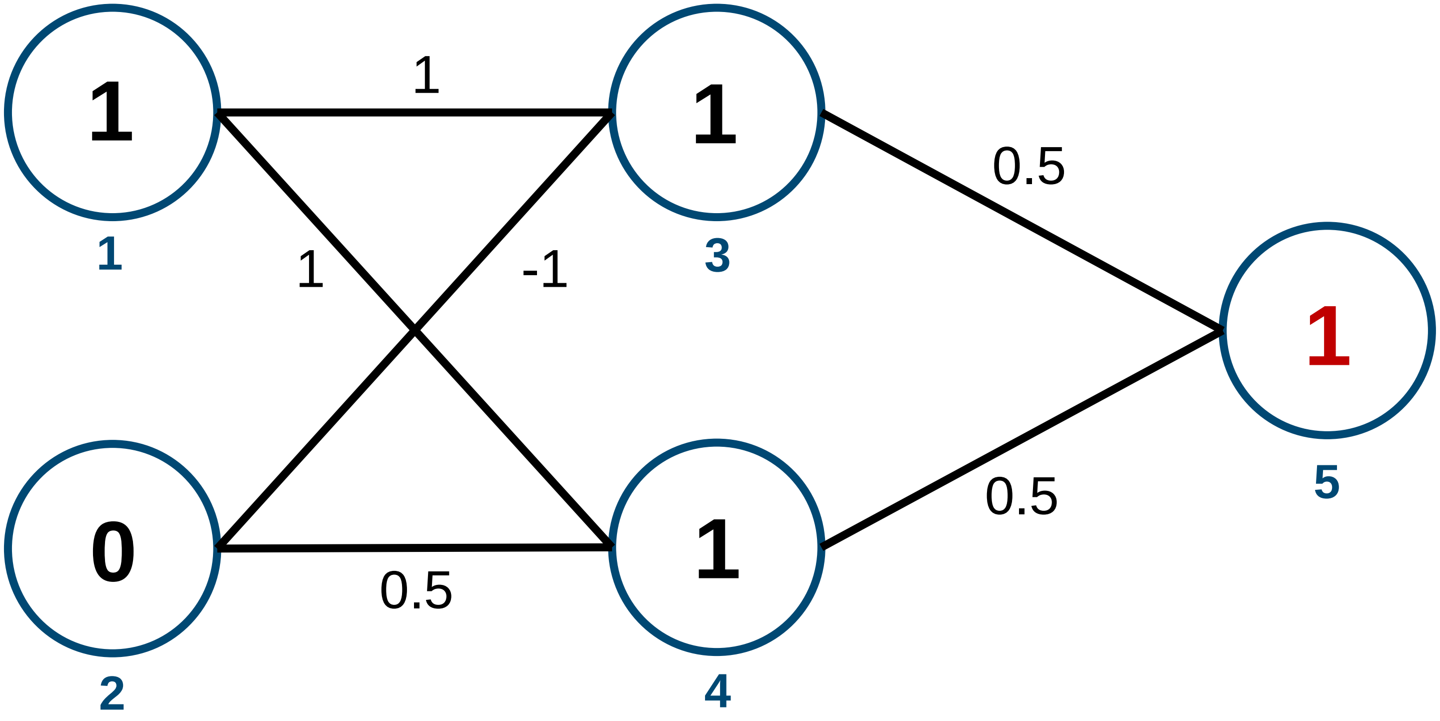
```
for each node $i$ in the input layer do
    $a_i \leftarrow x_i$
for $l = 2$ to $L$ do
    for each node $j$ in layer $l$ do
        $in_j \leftarrow \sum_i w_{i,j} a_i$
        $a_j \leftarrow g(in_j)$
```



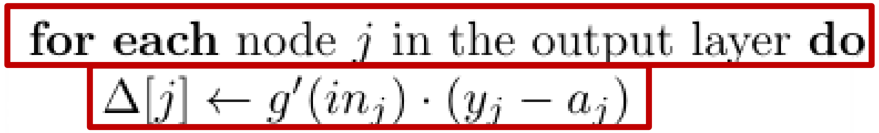
$$a_5 = g(0.5 \cdot 1 + 0.5 \cdot 1) = 1$$

Observing the Error

Expected output: **0**

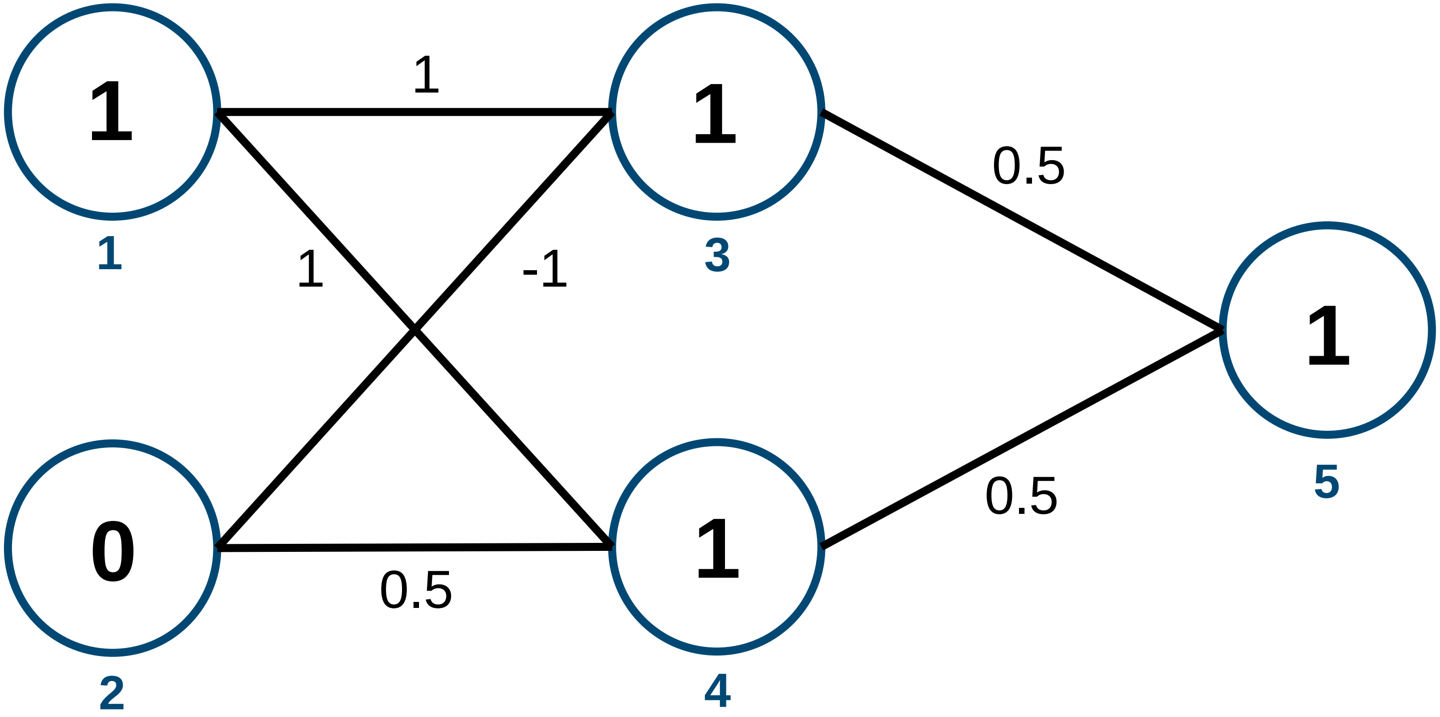


Backpropagation of Deltas Output Layer



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

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

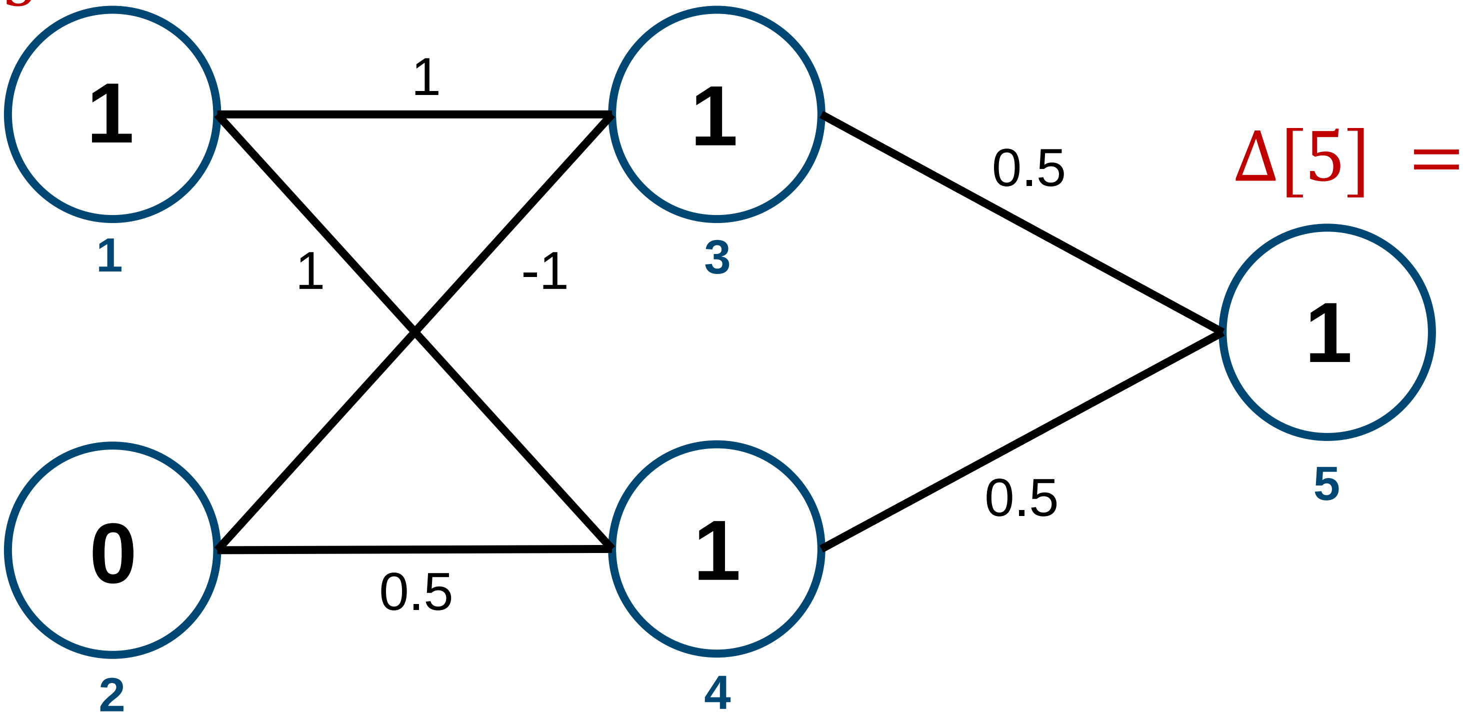


$$\Delta[5] = g'(0.5 \cdot 1 + 0.5 \cdot 1) \cdot (0\text{-}1)$$
$$= -1$$

Backpropagation of Deltas Hidden Layer

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

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



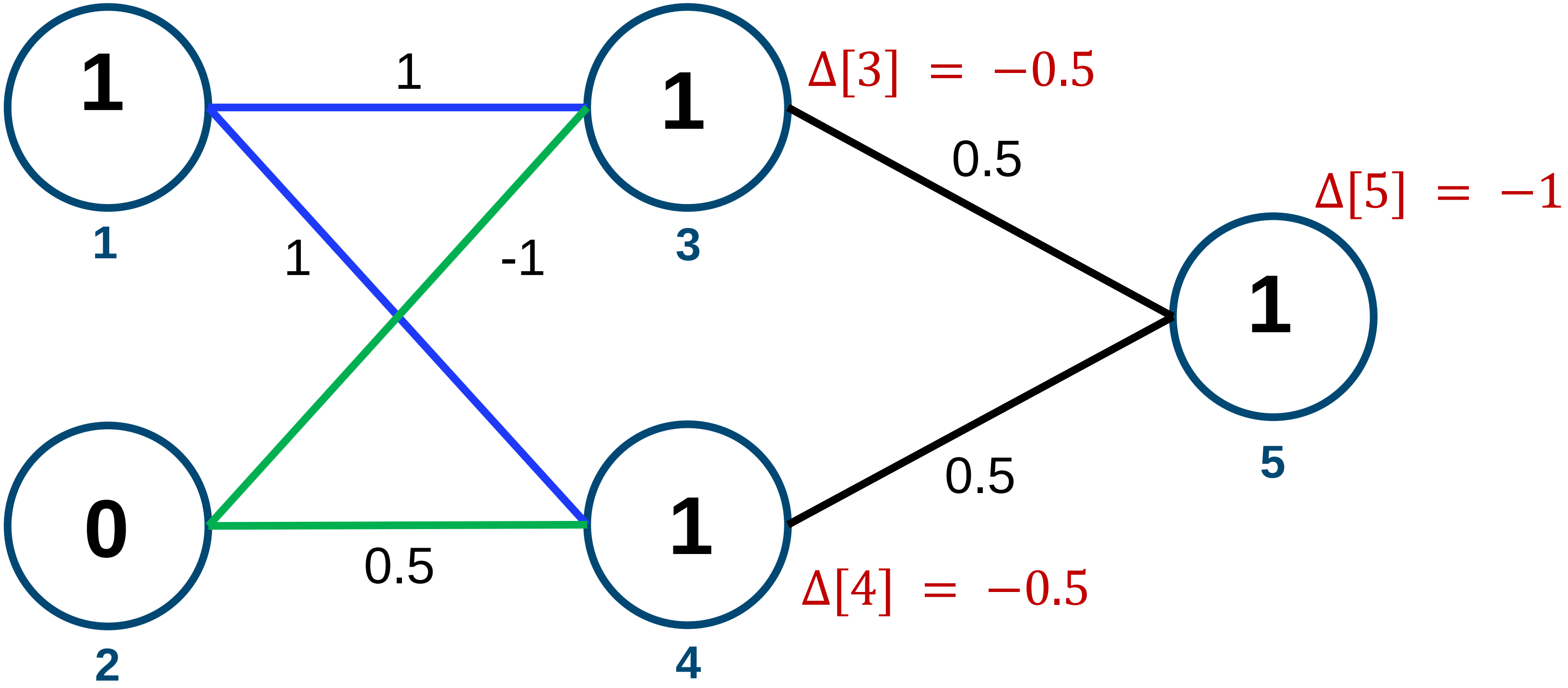
$$\Delta[5] = -1$$

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

Backpropagation of Deltas Input Layer(*)

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

$$\Delta[1] = g'(1) \cdot \big(1 \cdot (-0.5) + 1 \cdot (-0.5)\big)$$
$$= 0.25$$



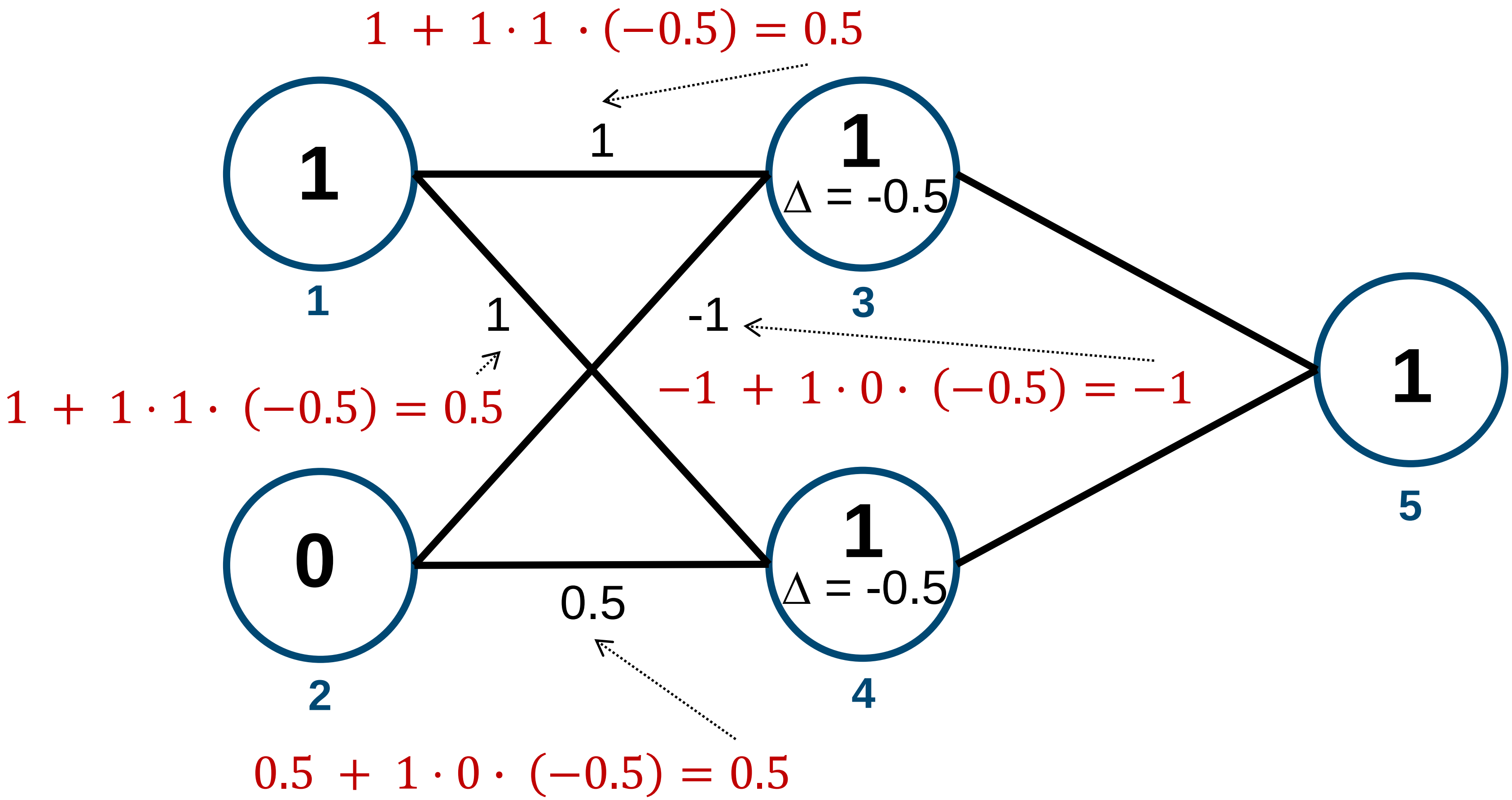
$$\Delta[2] = g'(0) \cdot \big(-1 \cdot (-0.5) + 0.5 \cdot (-0.5)\big)$$
$$= 0$$

(*) only shown to have another example computation of the delta computation, Deltas only required for hidden layers. We take the input value as in_i

Updating Weights
Input to Hidden Layer

```
for each weight $w_{i,j}$ in $network$ do
    $w_{i,j} \leftarrow w_{i,j} + \alpha \cdot a_i \cdot \Delta[j]$
```

$\alpha = 1$



$1 + 1 \cdot 1 \cdot (-0.5) = 0.5$

$1 + 1 \cdot 1 \cdot (-0.5) = 0.5$

$-1 + 1 \cdot 0 \cdot (-0.5) = -1$

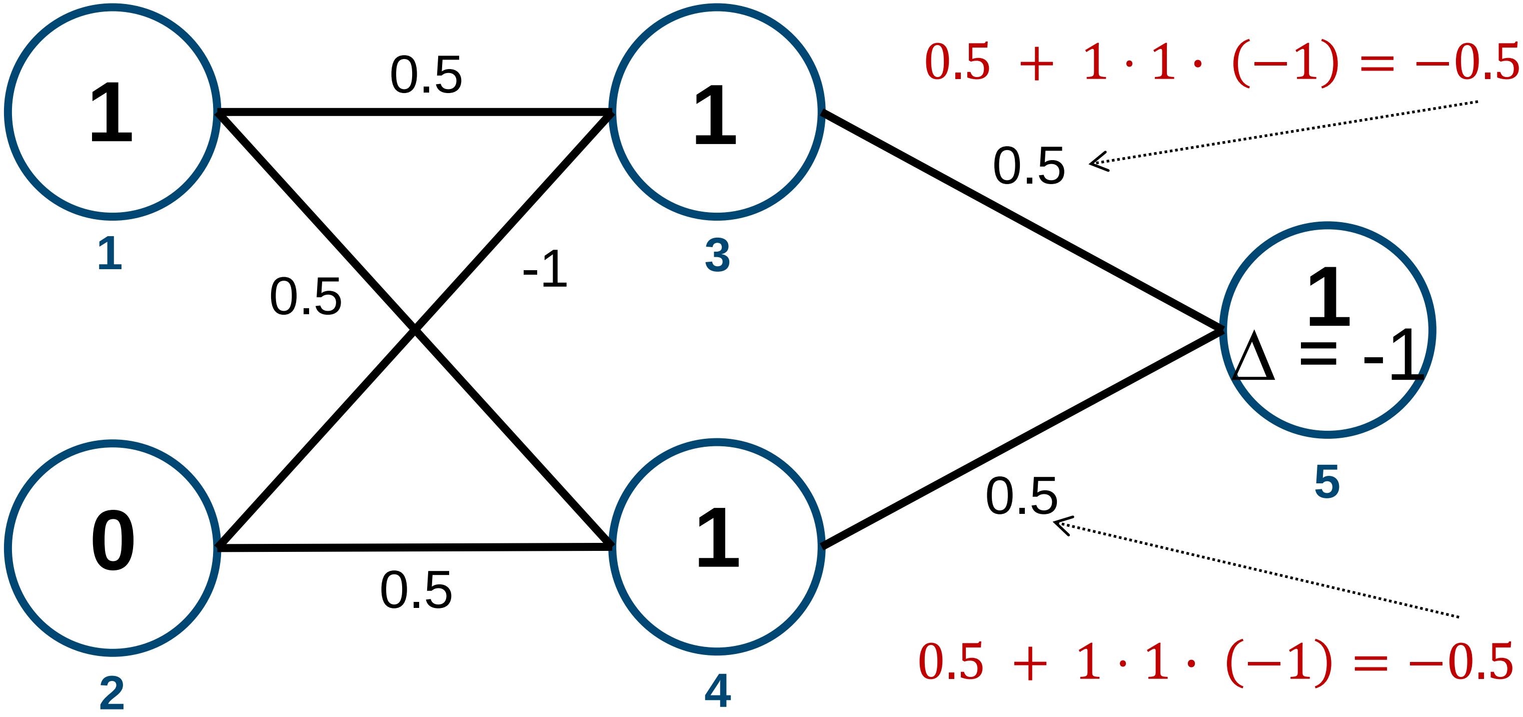
$0.5 + 1 \cdot 0 \cdot (-0.5) = 0.5$

Updating Weights
Hidden Layer to Output Layer

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

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

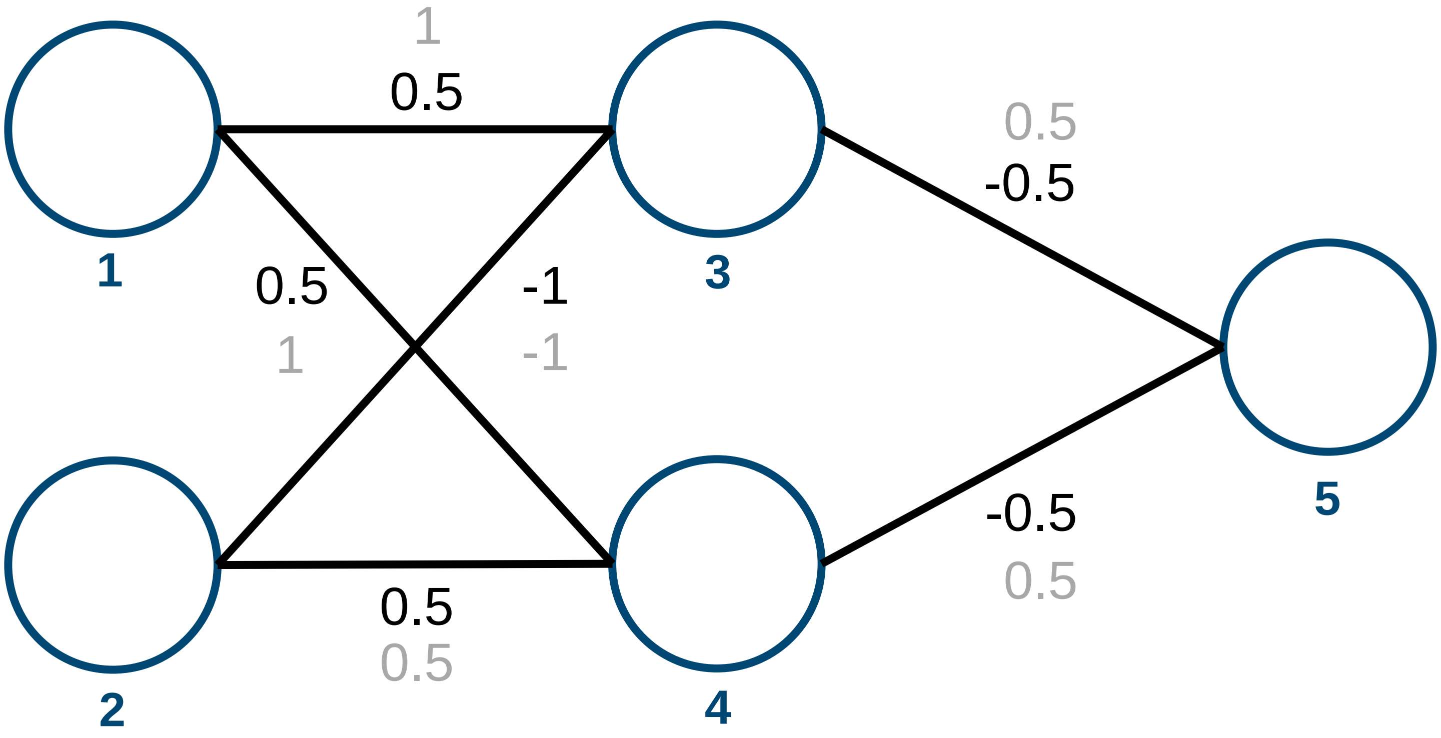
$\alpha = 1$



$0.5 + 1 \cdot 1 \cdot (-1) = -0.5$

$0.5 + 1 \cdot 1 \cdot (-1) = -0.5$

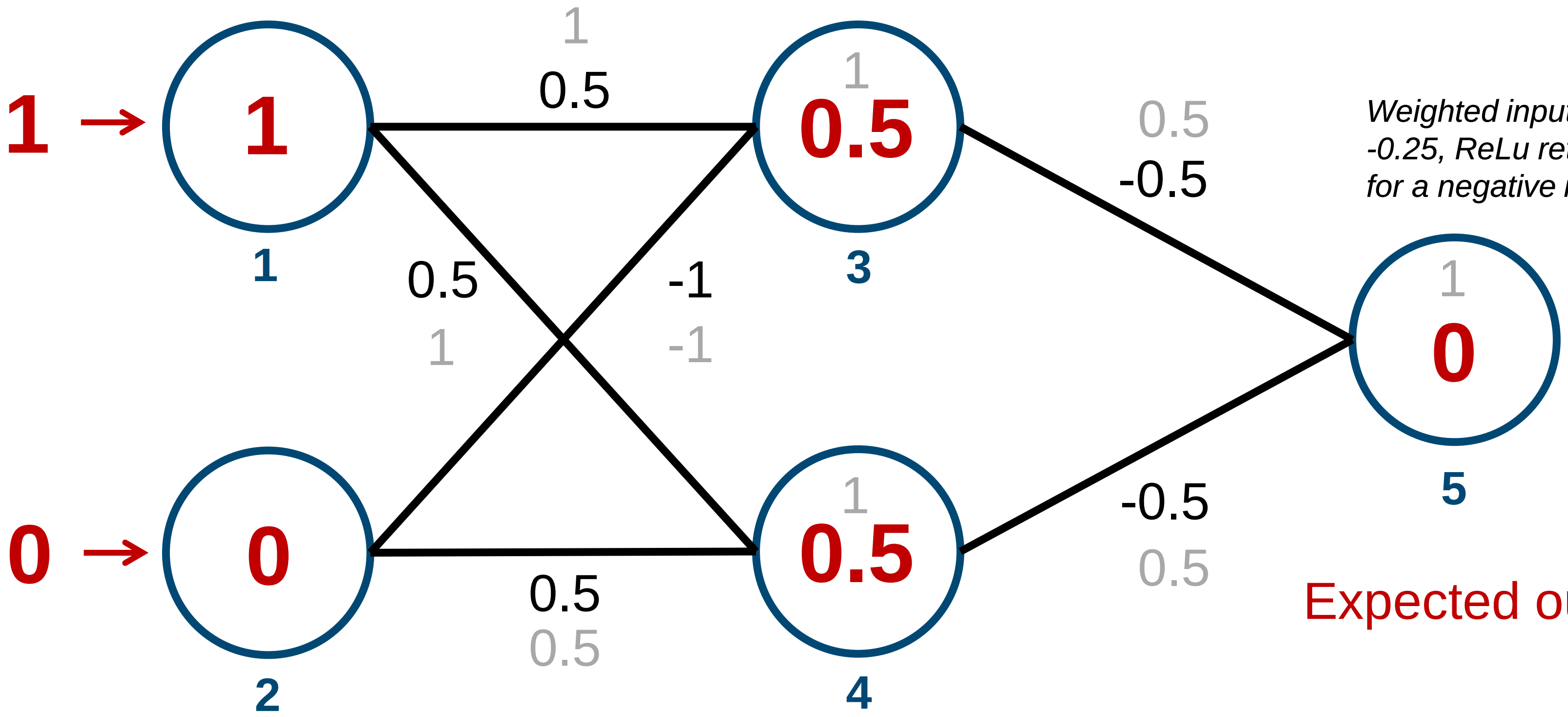
Network with Updated Weights



Old weights in grey for comparison

Comparison of Networks

Same input into the new network → Squared error is getting 0



Weighted input sum is -0.25, ReLu returns 0 for a negative input

Expected output: 0

Squared Error: $\frac{1}{2}(\mathbf{0} - \boldsymbol{a_5})^2$

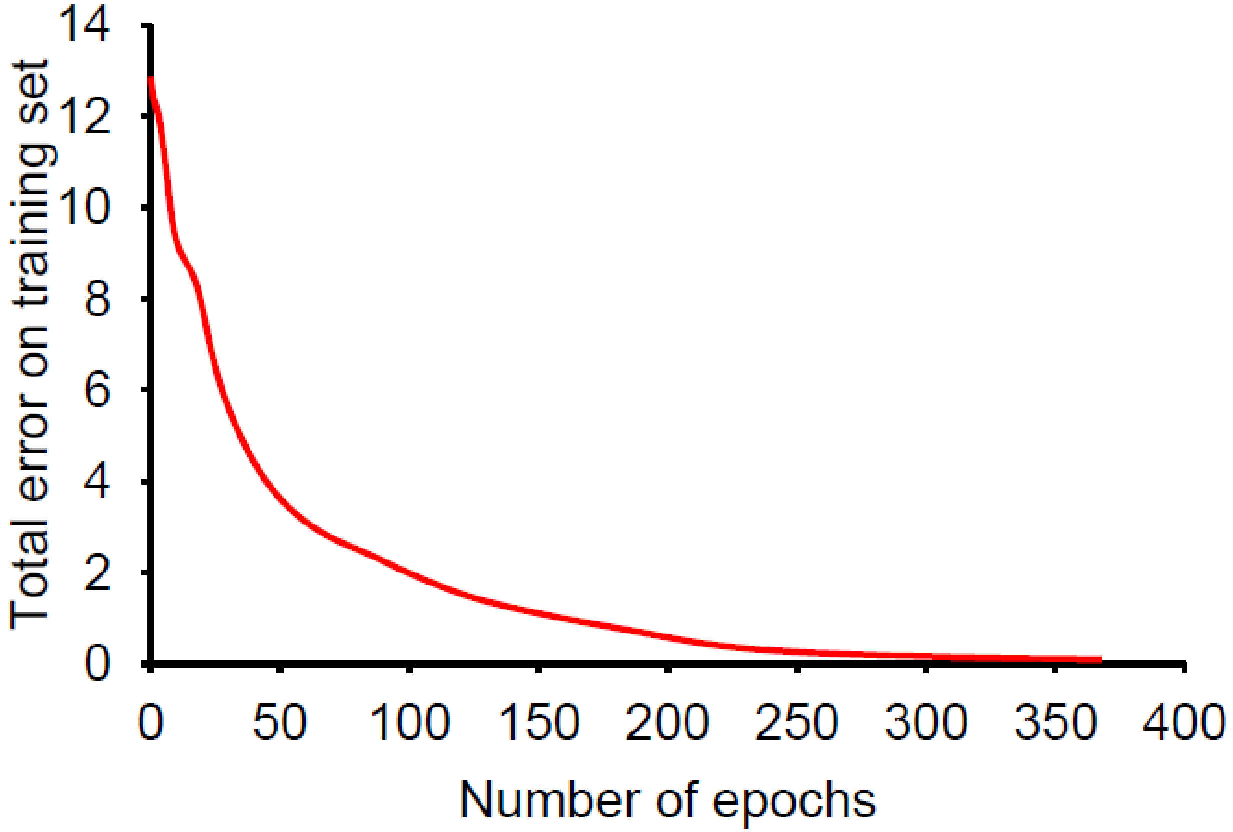
Squared Error old network: $\frac{1}{2}(0 - 1)^2 = 0.5$

Squared Error new network: $\frac{1}{2}(0 - (0))^2 = 0$

No error: The network has learned to recognize the example correctly.

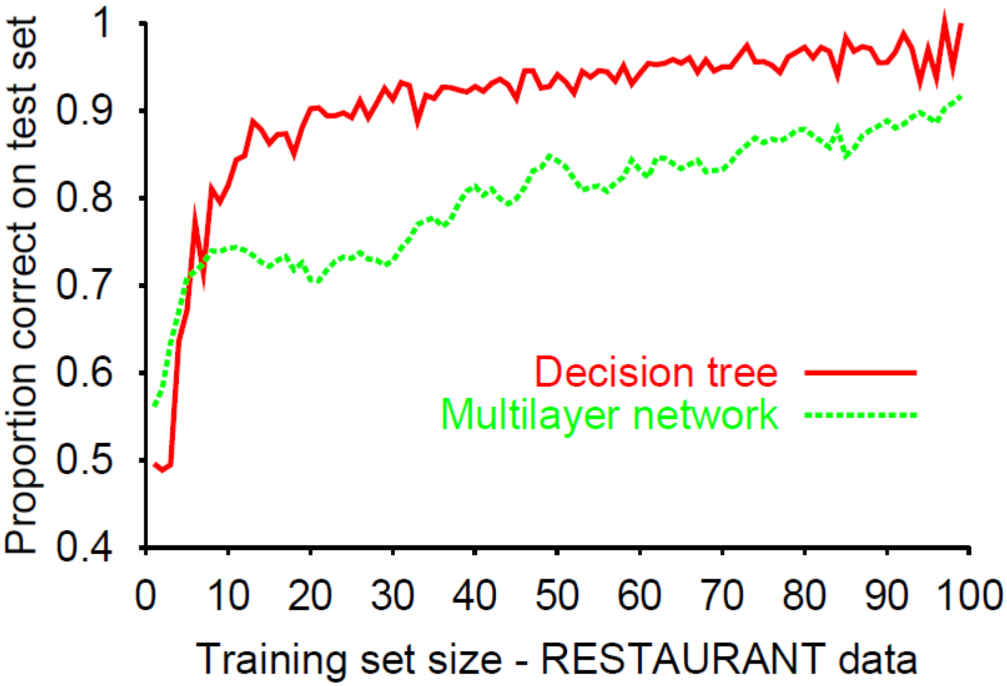
Convergence of Backpropagation Learning

- Training curve showing the gradual reduction in error as weights are modified over several epochs, for a given set of examples in the restaurant domain

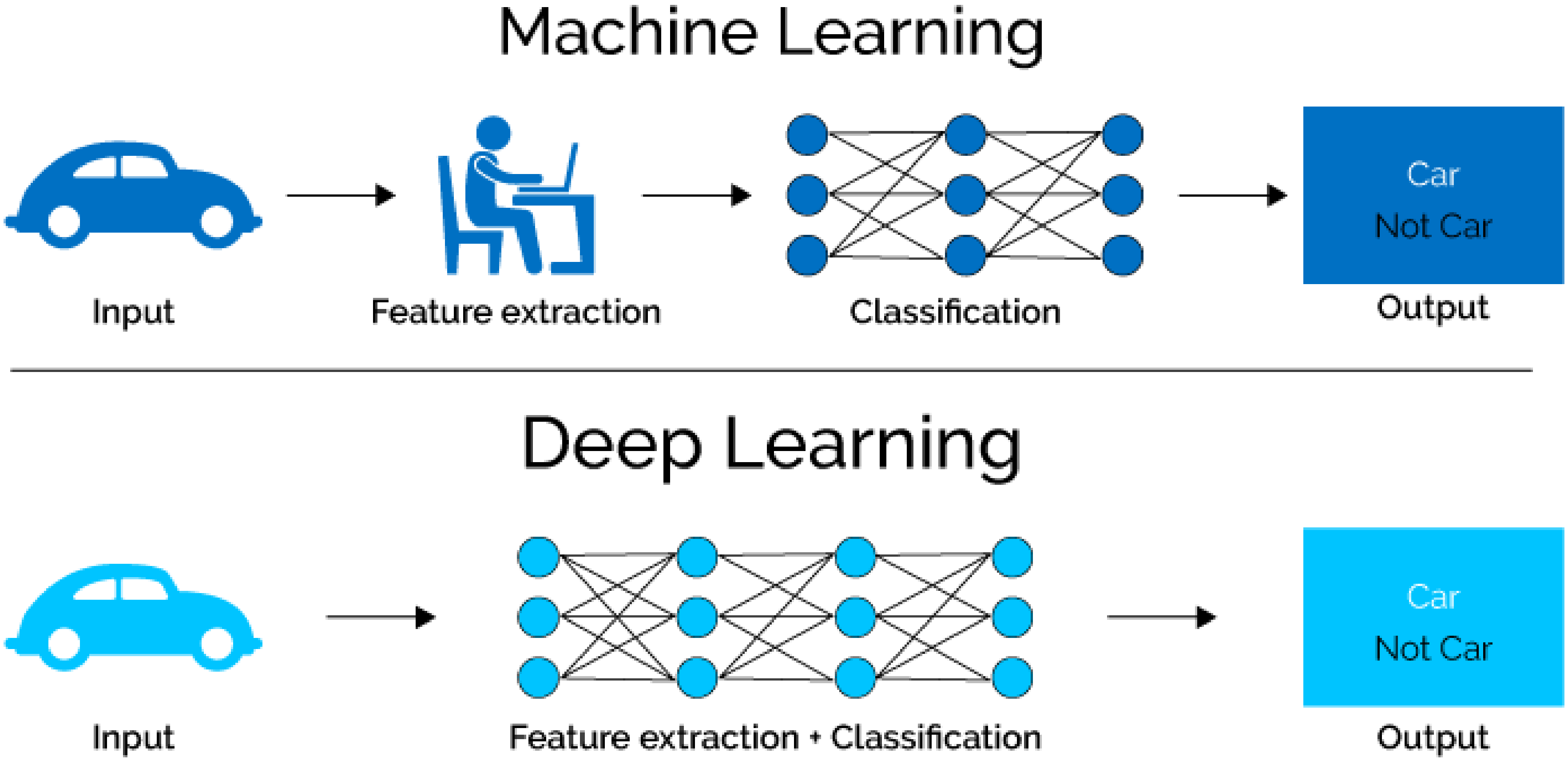


Comparison Multi-Layer Network - Decision Tree

- Comparative learning curves showing that decision-tree learning does slightly better on the restaurant problem than back-propagation in a multi-layer network
- Decision trees work well for small data sets, but do not scale well for multi-dimensional data



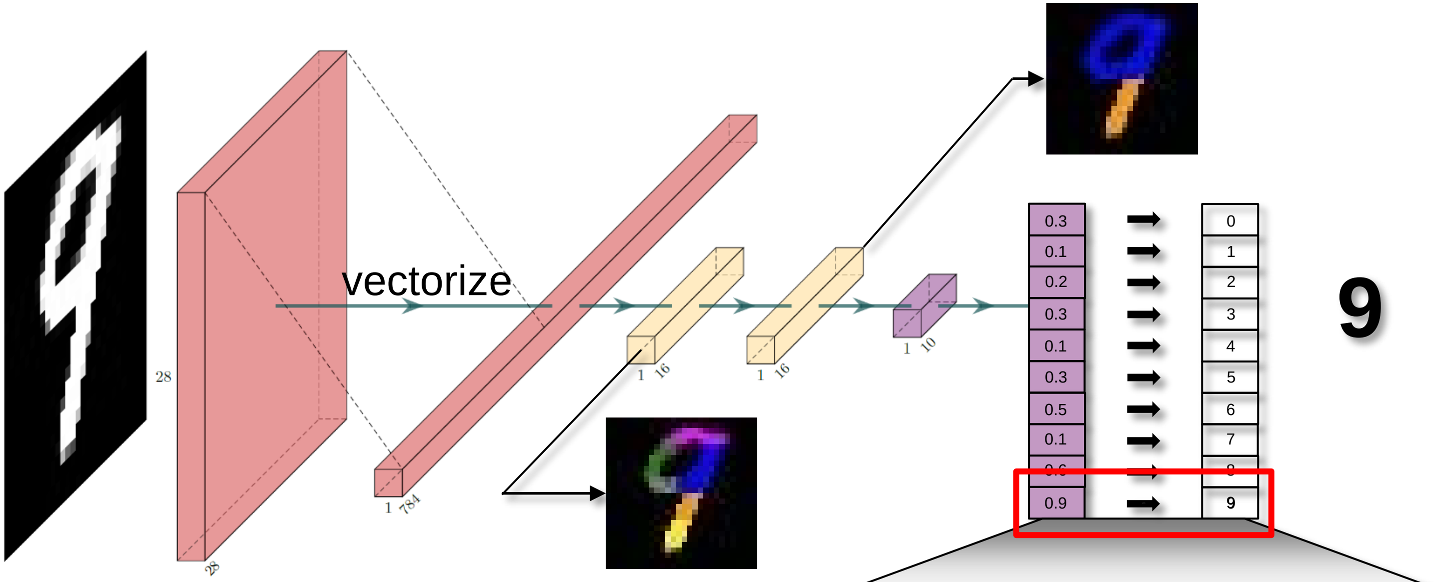
Deep Learning



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
- Utilize large amounts of training data

Exemplary Network for Digit Recognition



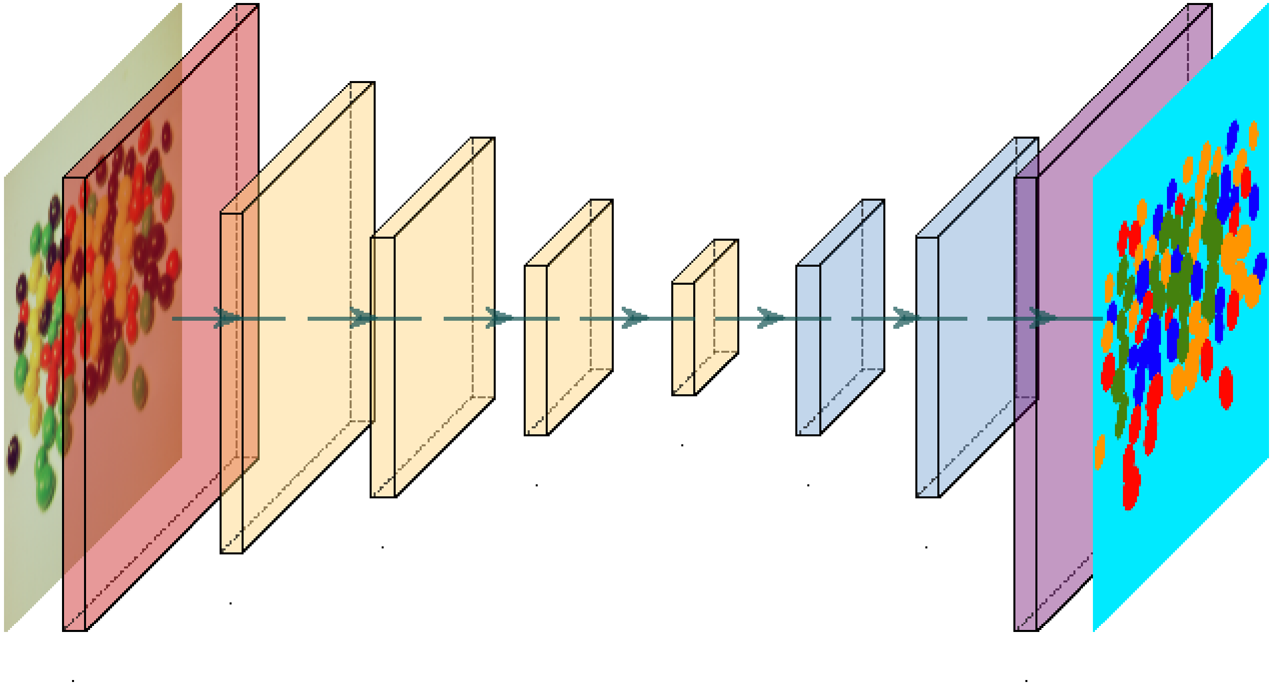
0.3	➡	0
0.1	➡	1
0.2	➡	2
0.3	➡	3
0.1	➡	4
0.3	➡	5
0.5	➡	6
0.1	➡	7
0.6	➡	8
0.9	➡	9

Measure of the network's confidence that the input is a certain number. Thus, the digit with the highest measure is the prediction.

Main information:

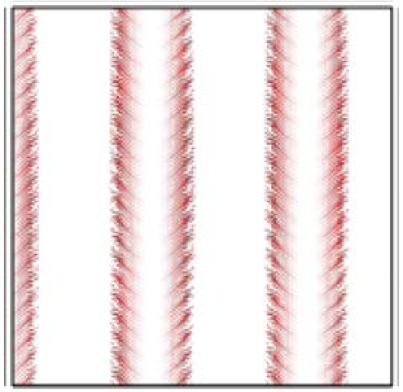
- Depending on the problem you encounter, you have to adjust the network
- The size of the hidden layers can be adapted
- Different networks can solve the same problem with different or equal performance

Exemplary Network for Image Segmentation

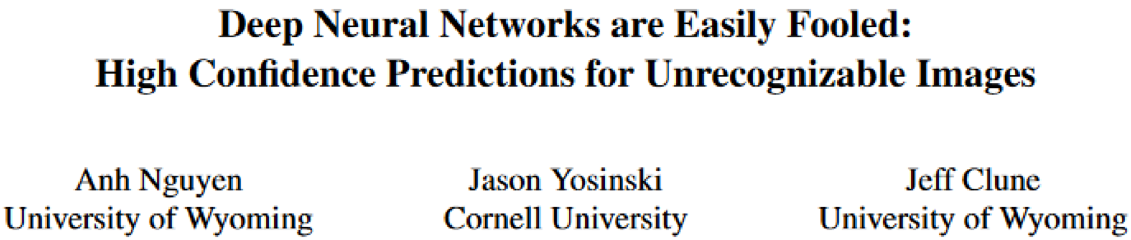


- Here we see an example network for the segmentation of a color image according to the colors (4 smartie-colors and the background)
- First the layers are reduced in size to categorize information
- Afterwards the results obtained are re-calculated for all pixels - this means that the layers must be enlarged to their original size

Challenges with Deep Neural Networks



A baseball with 99.6 % confidence

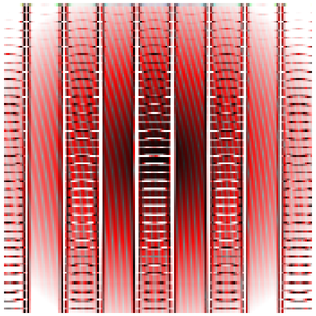


Deep Neural Networks are Easily Fooled:
High Confidence Predictions for Unrecognizable Images

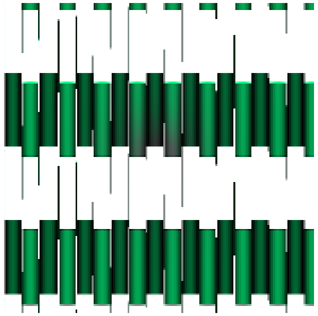
Anh Nguyen
University of Wyoming

Jason Yosinski
Cornell University

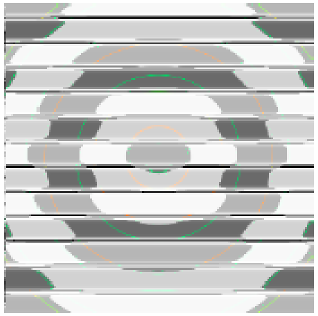
Jeff Clune
University of Wyoming



accordion

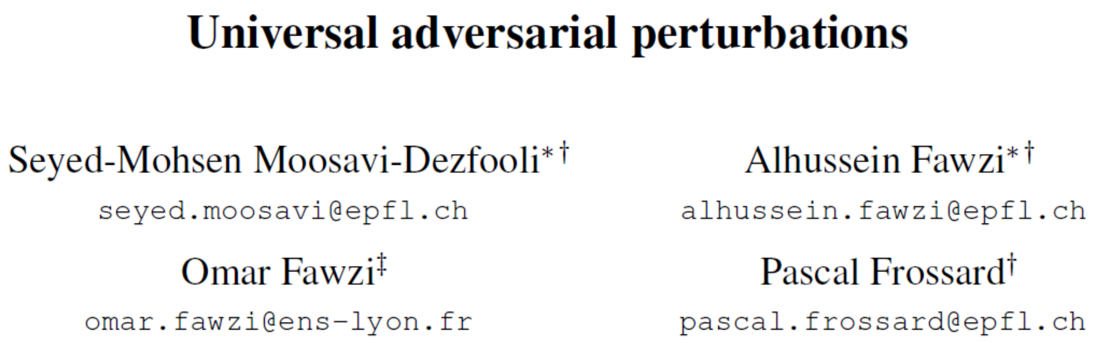


screwdriver



photocopier

Challenges with Deep Neural Networks Cntd.



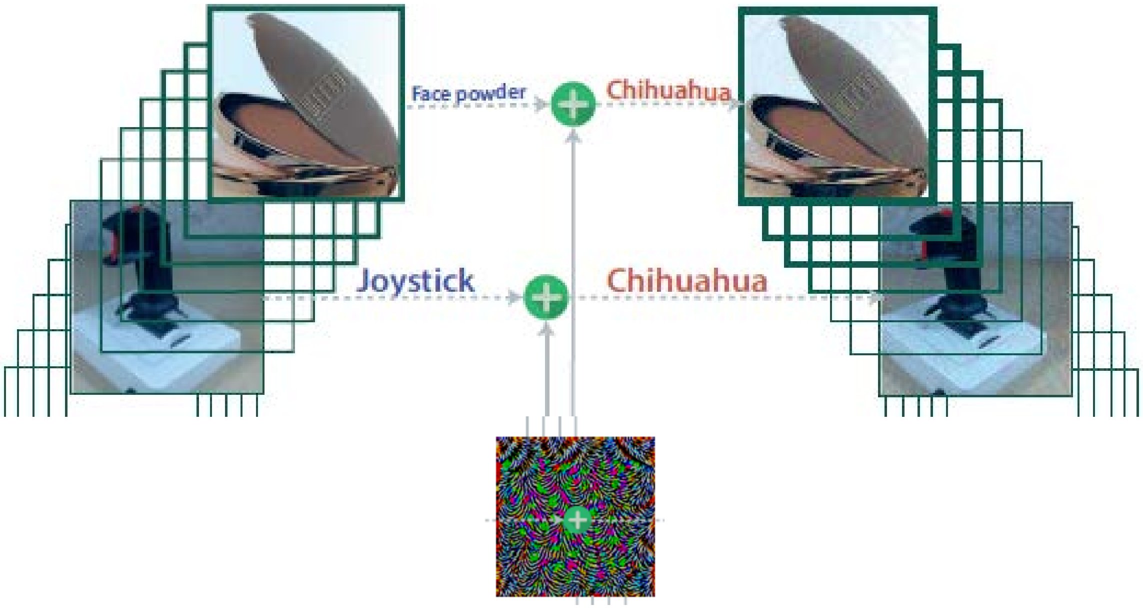
Universal adversarial perturbations

Seyed-Mohsen Moosavi-Dezfooli*[†]
seyed.moosavi@epfl.ch

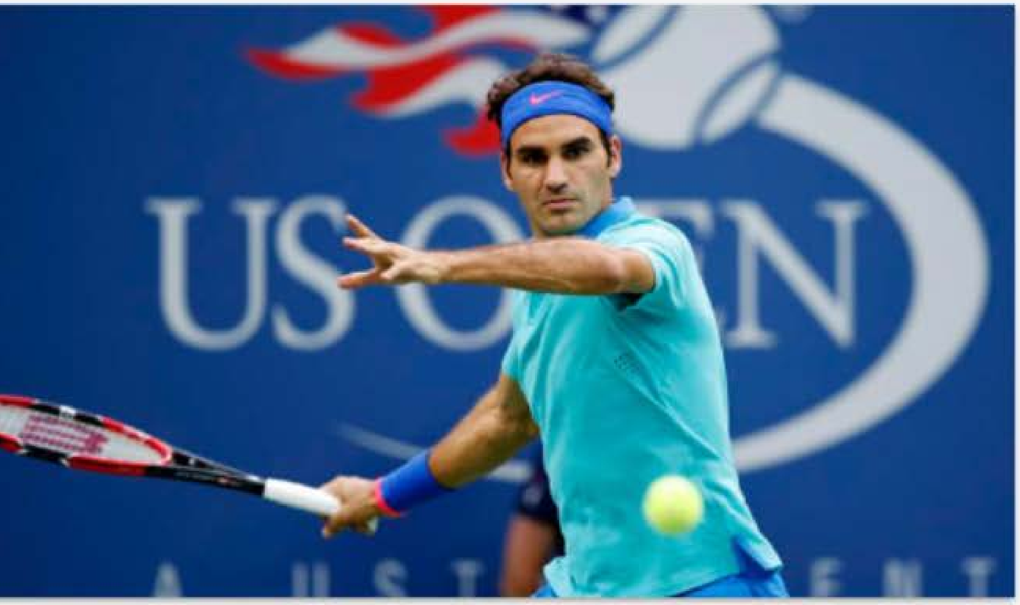
Alhussein Fawzi*[†]
alhussein.fawzi@epfl.ch

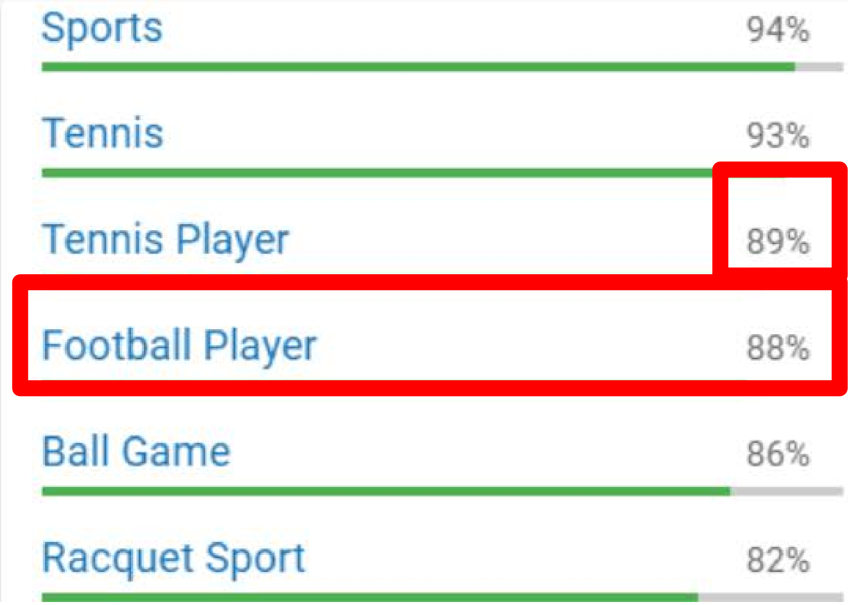
Omar Fawzi[‡]
omar.fawzi@ens-lyon.fr

Pascal Frossard[†]
pascal.frossard@epfl.ch

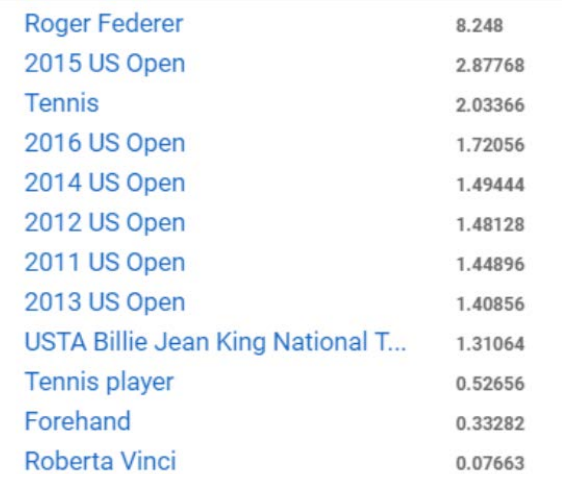


Pattern Similarity can lead to Conceptual Inconsistency

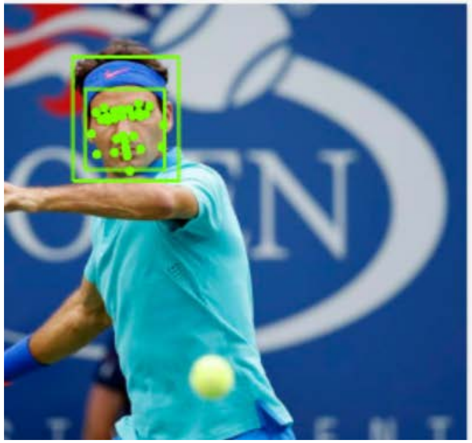


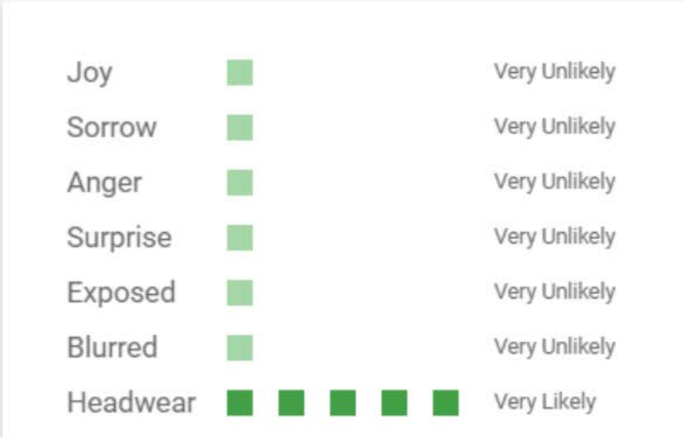


Sports	94%
Tennis	93%
Tennis Player	89%
Football Player	88%
Ball Game	86%
Racquet Sport	82%

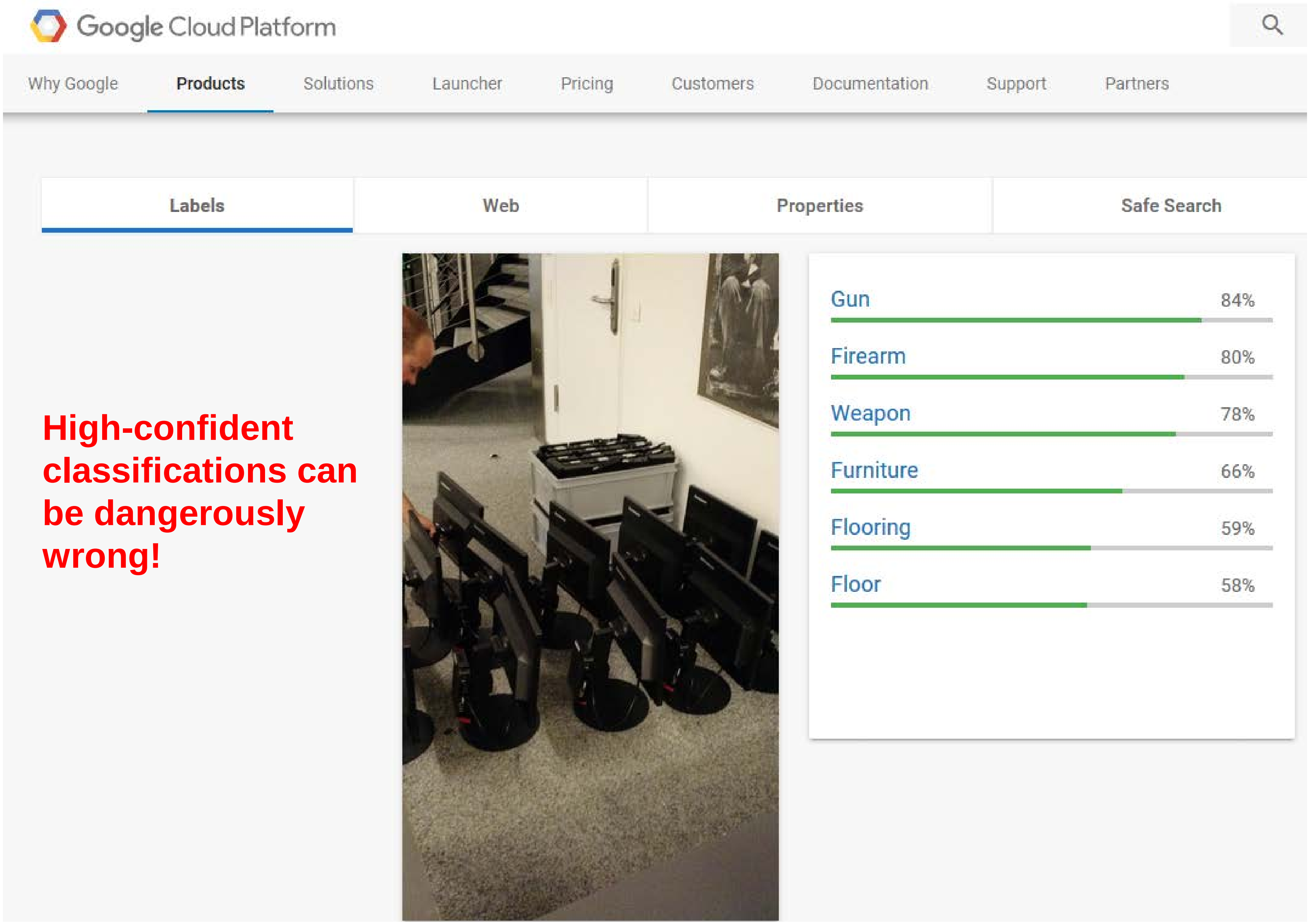


Roger Federer	8.248
2015 US Open	2.87768
Tennis	2.03366
2016 US Open	1.72056
2014 US Open	1.49444
2012 US Open	1.48128
2011 US Open	1.44896
2013 US Open	1.40856
USTA Billie Jean King National T...	1.31064
Tennis player	0.52656
Forehand	0.33282
Roberta Vinci	0.07663





Joy	🟩	Very Unlikely
Sorrow	🟩	Very Unlikely
Anger	🟩	Very Unlikely
Surprise	🟩	Very Unlikely
Exposed	🟩	Very Unlikely
Blurred	🟩	Very Unlikely
Headwear	🟩 🟩 🟩 🟩 🟩	Very Likely



Gun	84%
Firearm	80%
Weapon	78%
Furniture	66%
Flooring	59%
Floor	58%

High-confident classifications can be dangerously wrong!

For a more In-Depth Introduction to Neural Networks

- Andrew Ng´s machine learning and neural networks courses on coursera https://www.coursera.org/instructor/andrewng
- This nice intuitive introduction (including example) on youtube (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
- A good book for more mathematical details on Machine Learning and Neural Networks: Christopher M. Bishop: Pattern Recognition and Machine Learning, 2006

Summary

- Machine learning is a very important and huge research area in artificial intelligence
- Supervised learning takes a set of annoted training data (input-output pairs) as input and generates a generalized function to correctly predict the output for a given input
- Decision tree learning is an effective method for smaller data sets
- Neural networks are layered graphs of processing units and trained by backpropagation
- Neural networks have shown impressive performance and are widely used for image and speech recognition tasks or automatic translation today - despite inherent limitations
- Information theory provides an important basis for learning and the evaluation of learning algorithms
- Measures such as precision, recall, accuracy, specifity, F1 are used to compare the performance of learning algorithms on specific tasks

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 evaluate the results of a learning algorithm?