Machine Learning
CSE2510 – Lecture 6.2

Non-linear classification – Multi-layer perceptrons

Odette Scharenborg

Welcome to week 6 - lecture 1

- Administrative questions?
- Recap previous lecture

- Perceptron
- Multi-layer perceptrons
- Combining classifiers
- Class imbalance problem

Administrative questions?

Recap of the previous lecture

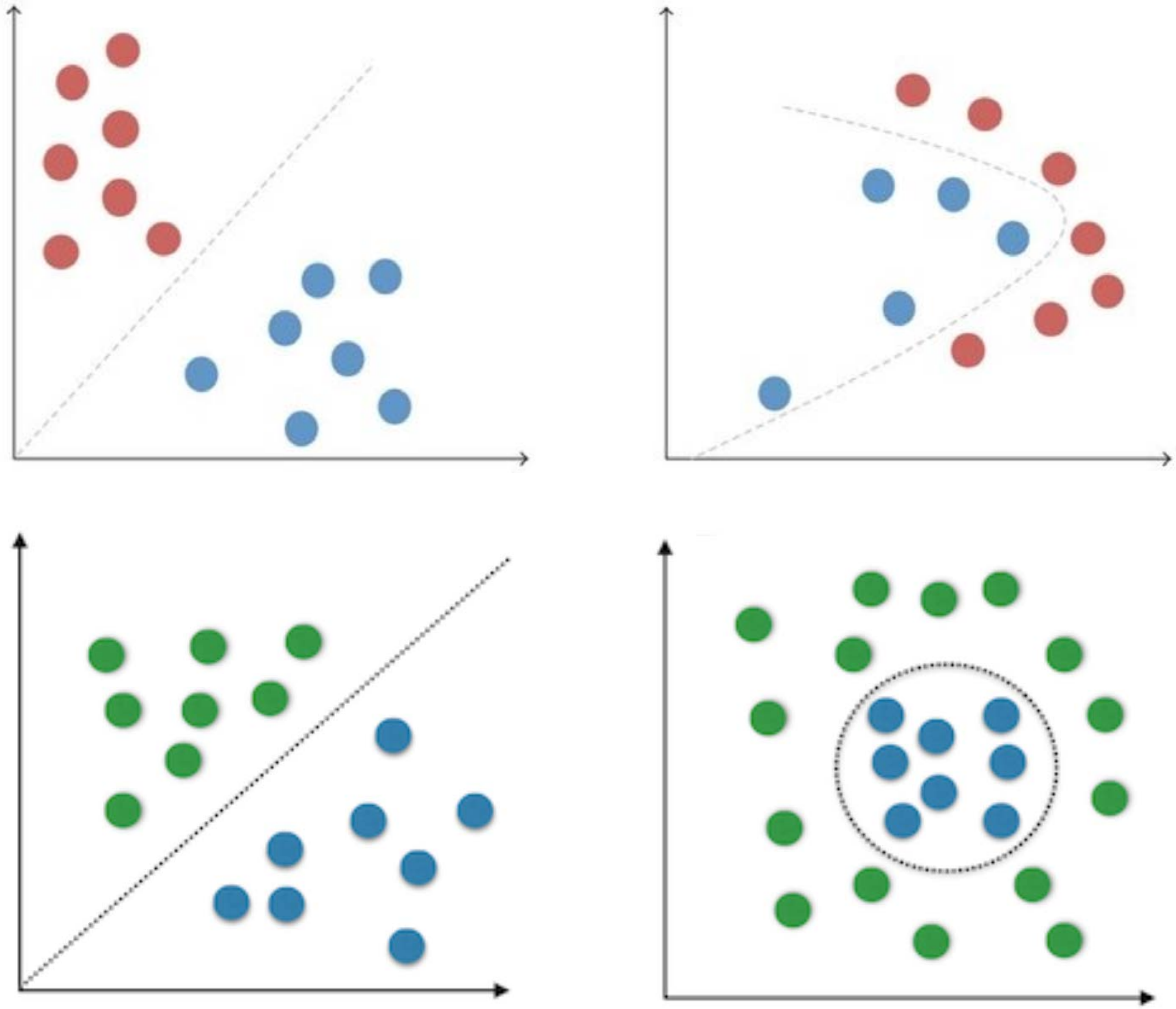
Different types of classifiers

Machine learning:

- Supervised and unsupervised classifiers
- Parametric and non-parametric classifiers
- Generative and discriminative classifiers
- Linear and non-linear classifiers

This week: supervised & non-parametric & discriminative & non-linear classification

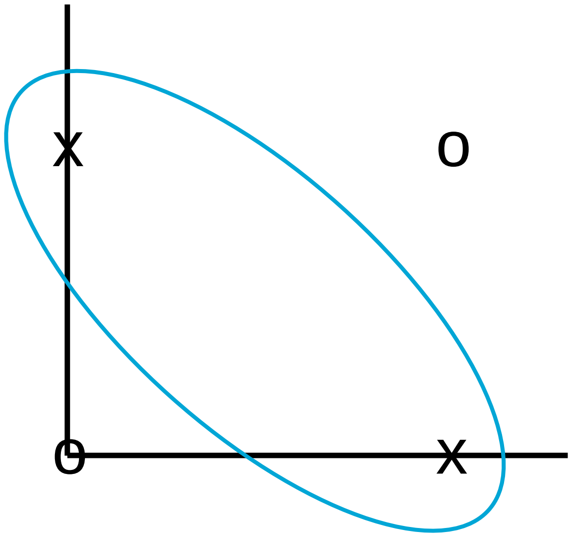
Linear vs. non-linear classification problems



XOR problem

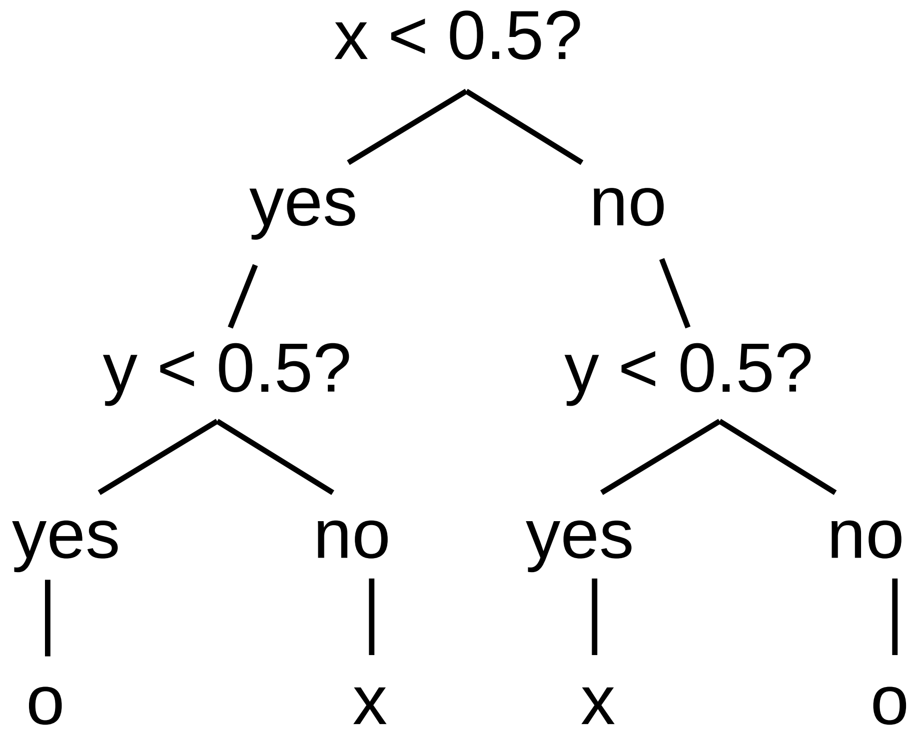
Where do you put the decision boundary to separate the two classes?

X = (0,1);(1,0)
O = (0,0);(1,1)



Decision trees

- X = (0,1);(1,0)
- O = (0,0);(1,1)



Decision trees

- Split the training data into unique regions *sequentially*
- Structure of the tree is not predefined but depends on the complexity and structure of the training data
- Structure grows from the root down
- At every node, a decision is made which splits the training data into smaller subsets

Predict if John will play tennis

Training examples: **9 yes / 5 no**

Day	**Outlook**	**Humidity**	**Wind**	**Play**
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

New data:

D15	Rain	High	Weak	?

ID3 algorithm

- Split (node, {examples}):
  1. A ← the best attribute for splitting the {examples}
  2. Decision attribute for this node ← A
  3. For each value of A, create new child node
  4. Split training {examples} to child nodes
  5. If examples perfectly classified: STOP
     else: iterate over new child nodes
     Split (child_node, {subset of examples})

ID3 algorithm

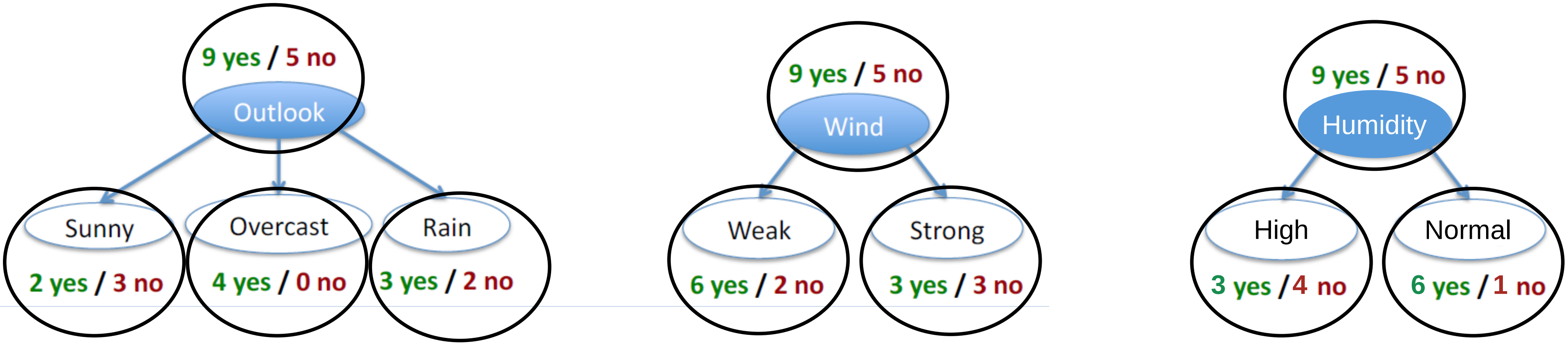
- Split (node, {examples}):
  1. A ← the best attribute for splitting the {examples}
  2. Decision attribute for this node ← A
  3. For each value of A, create new child node
  4. Split training {examples} to child nodes
  5. If examples perfectly classified: STOP
     else: iterate over new child nodes

     Split (child_node, {subset of examples})

The best attribute for splitting?

- For each attribute (i.e., *outlook, humidity, wind*):
- Calculate the **entropy** over the root and all attributes’ values:

$$H(S) = -\, p_{(+)} \log_2 p_{(+)} - p_{(-)} \log_2 p_{(-)} \text{ bits}$$



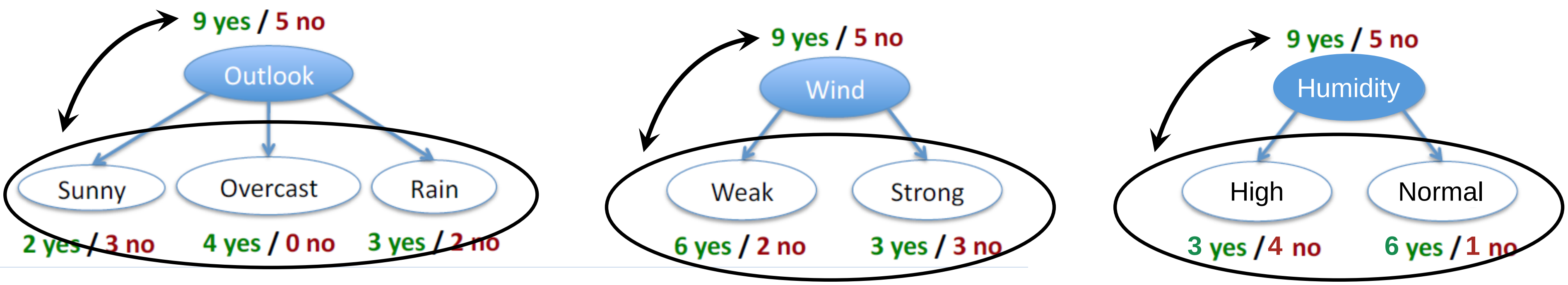
$$H(S) = -\, \Sigma_c\, p_{(c)} \log_2 p_{(c)}$$

The best attribute for splitting?

- Calculate **average weighted entropy**
- Calculate the **information gain**

$$Gain(S,A) = H(S) - \sum_{V \in Values(A)} \frac{|S_V|}{|S|} H(S_V)$$

V ... possible values of A
S ... set of examples {X}
S_V ... subset where X_A = V



TUDelft

The best attribute for splitting?

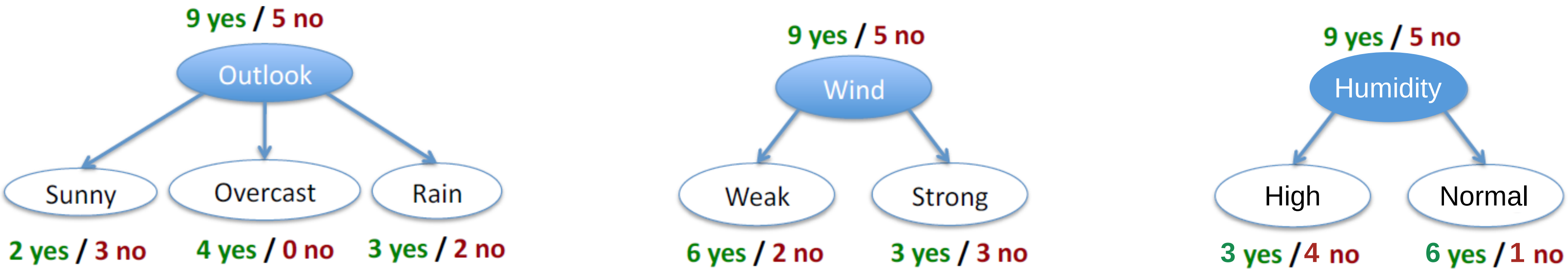
- Calculate the **gain ratio** to take into account attributes with more/fewer values

$$SplitEntropy(S,A) = -\sum_{V \in Values(A)} \frac{|S_V|}{|S|} \log \frac{|S_V|}{|S|}$$

A … candidate attribute
V … possible values of A
S … set of examples {X}
S_V … subset where X_A = V

$$GainRatio(S,A) = \frac{Gain(S,A)}{SplitEntropy(S,A)}$$

penalizes attributes with many values



Best attribute: the one with the highest gain ratio

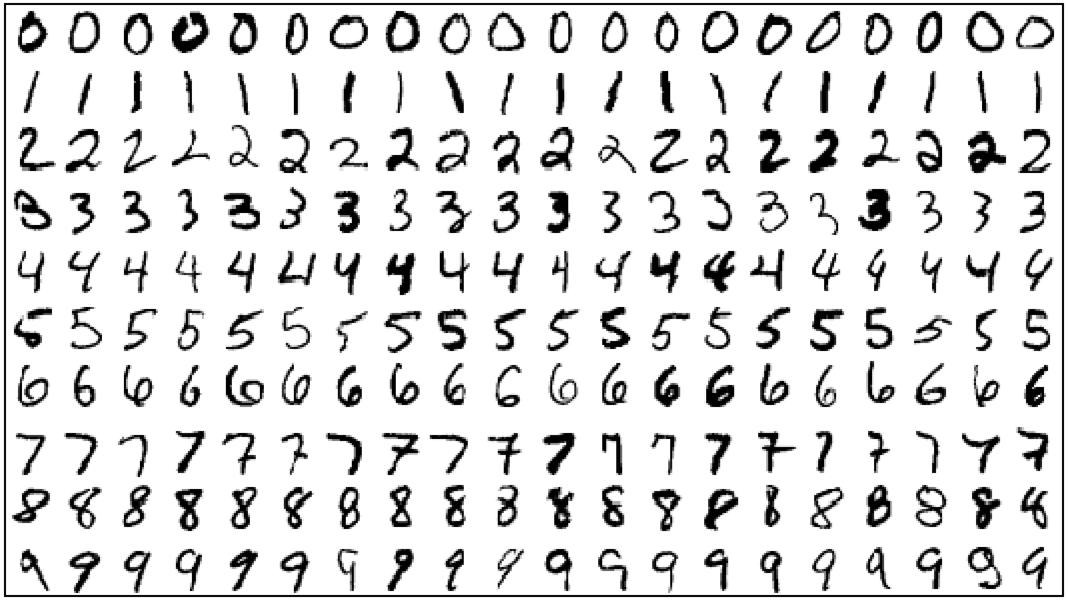
Avoid overfitting

- Split training data into training and validation set
- Grow tree to pure leaves
- Prune by removing subtrees and leaves
- Test on validation set
- Repeat until accuracy on validation set goes down

Today's learning objectives

After practicing with the concepts of today's lecture you are able to:

- Explain the underlying algorithm of **multi-layer perceptrons**



0
1
2
3
4
5
6
7
8
9

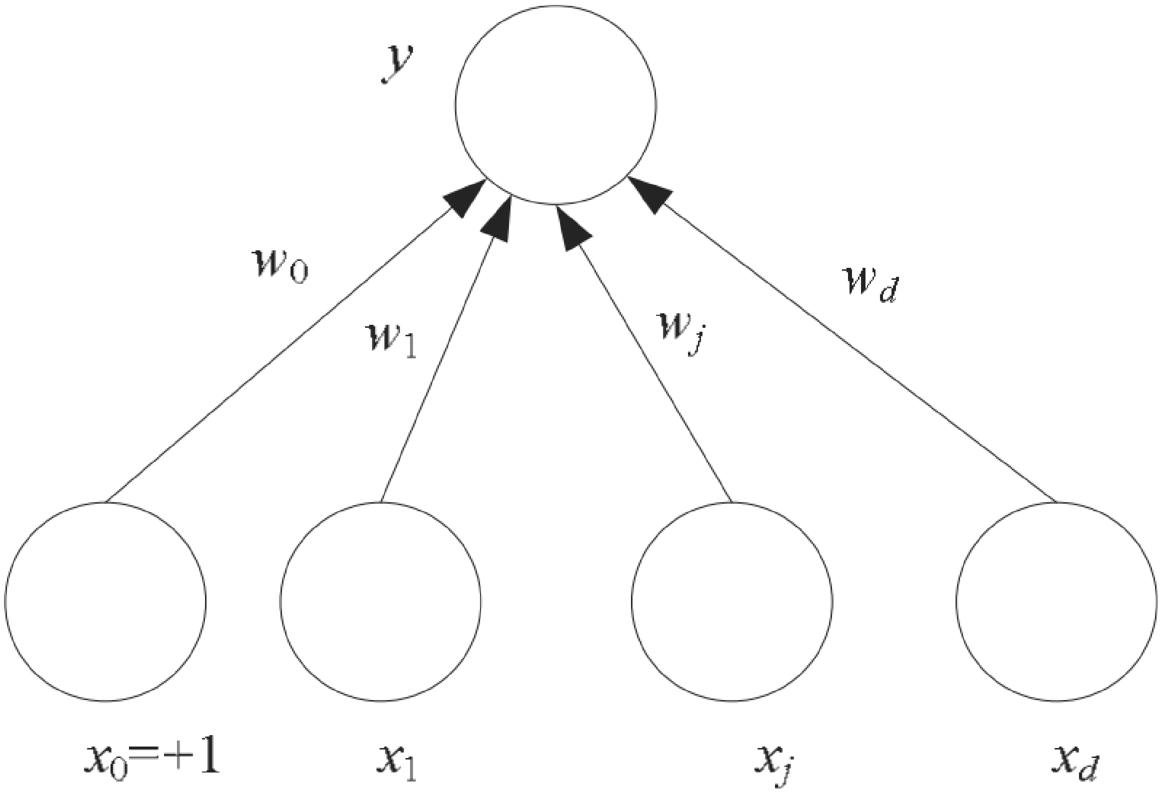
Artificial neural networks

- Inspired by the human brain

- Neurons → Perceptron
  - Basic processing unit
- Synapses → Connections
  - Connects the neurons/perceptrons

- Multi-layer perceptron
  - Several layers of connected perceptrons

Perceptron

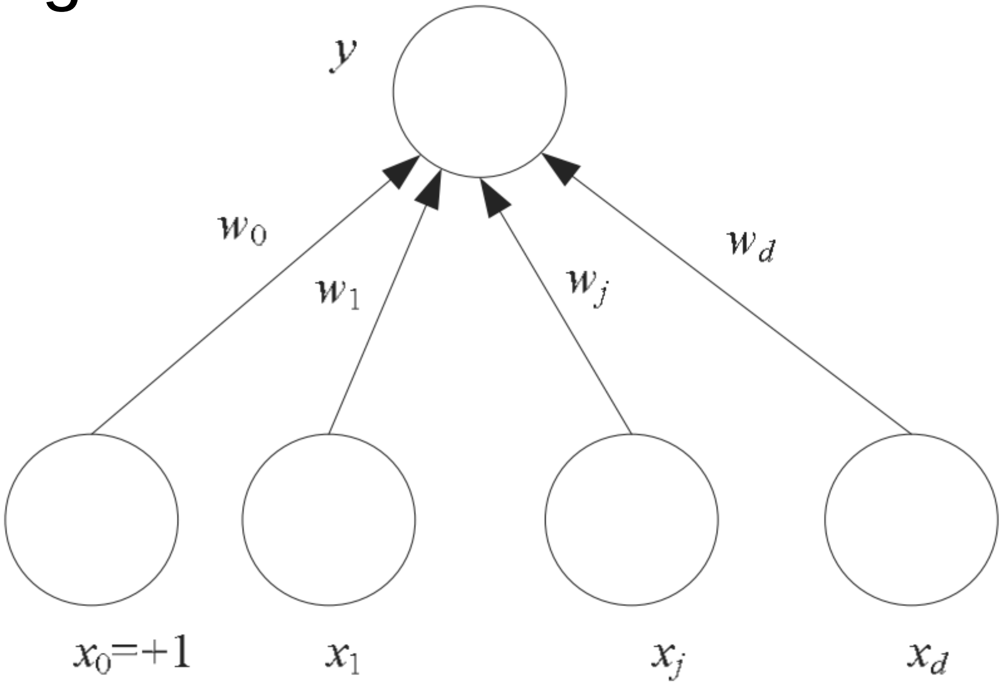


A simple perceptron

- x_j = input unit $\in R, j = 1,..,d$
- x_0 = bias (always 1)
- w_j = connection weight
- y = output unit

Simplest form of y

$$y = \sum_{j=1}^{d} w_j x_j + w_0$$



Output of a perceptron

- $y = w^T x + b$

- We have seen this before → linear classifier

Can be used to separate two classes

- Threshold θ

$$y = \begin{cases} 1 \ \text{if } w^T x + b >= \theta \\ 0 \ \text{otherwise} \end{cases}$$

= Perceptron algorithm

➔ Perceptron is a linear classifier

Posterior probability

- A perceptron outputs the code of the winning class

- $o = \ w^T x + b$
- $y = sigmoid(o) = \ \frac{1}{1+\exp\left[-w^T\right]x+b}$

Multi-class classifier

- K parallel perceptrons

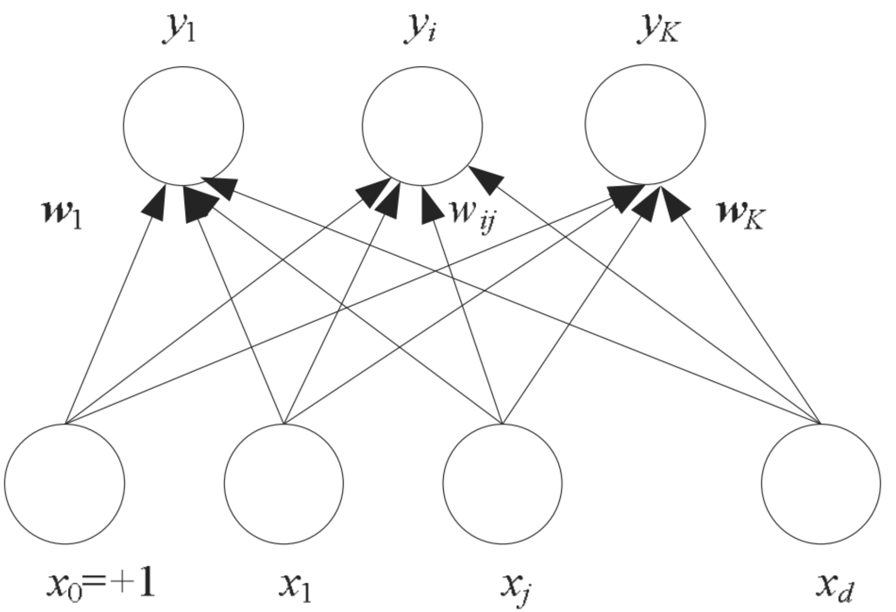
$$y_i = \sum_{j=1}^{d} w_{ij} x_j + w_{i0} = \mathbf{w}_i^T \mathbf{x} + b$$

where i = 1,..., K outputs

- w_{ij} = weight of the connection from input x_j to output y_i

- $\boldsymbol{y} = \mathbf{W}\boldsymbol{x} + b$,

where $\mathbf{W} = K$ x d vector



TU Delft

Multi-class classification

- Choose C_i if $y_i = \max_k y_k$

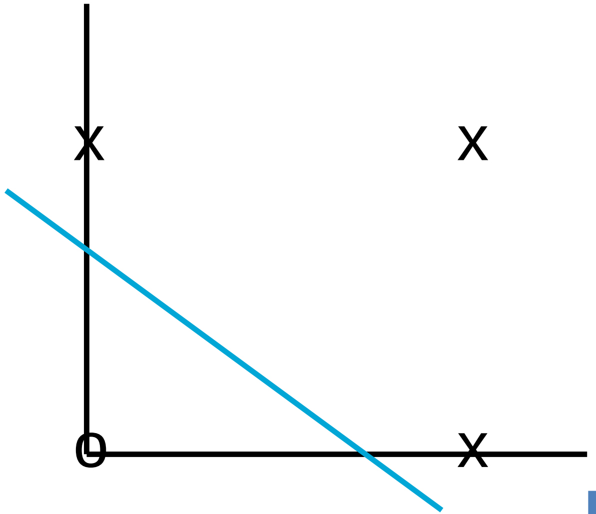
- Posterior probabilities: two-stage process (but considered 1 layer)
  - Calculate the weighted sums (= activations)
  - Calculate the softmax for each output node i

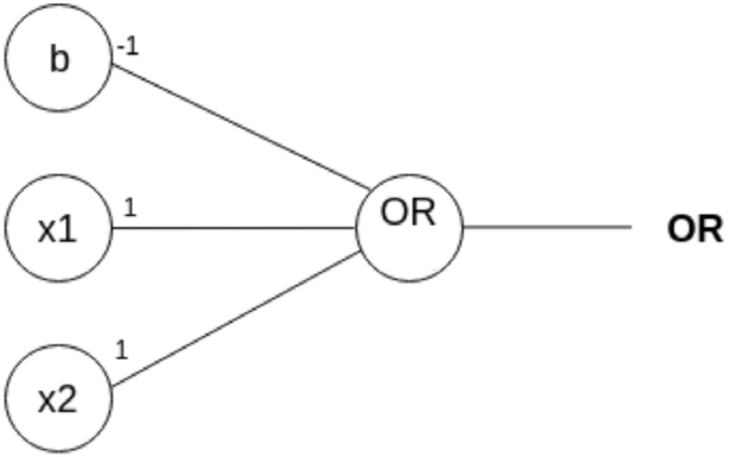
$$o_i = w_i^T x + b$$

$$y_i = \frac{\exp o_i}{\sum_k \exp o_k}$$

TU Delft

An example



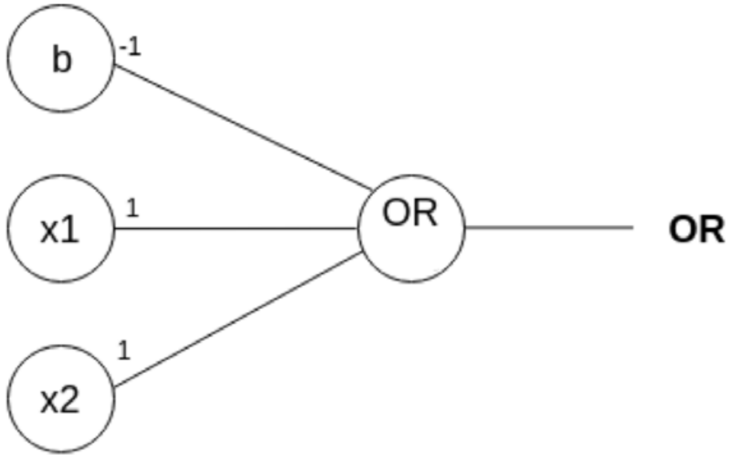


x	*y*	**Label**
0	0	o = 0
0	1	x = 1
1	0	x = 1
1	1	x = 1

https://medium.com/@stanleydukor/neural-representation-of-and-or-not-xor-and-xnor-logic-gates-perceptron-algorithm-b0275375fea1

An example: the OR gate

- $\boldsymbol{y} = \mathbf{W}\boldsymbol{x} + b$
- $y = 1 \times x_1 + 1 \times x_2 - 1$



x_1	x_2	Label	Calculation	Result
0	0	0	1 x 0 + 1 x 0 - 1	-1
0	1	1	1 x 0 + 1 x 1 - 1	0
1	0	1	1 x 1 + 1 X 0 - 1	0
1	1	1	1 x 1 + 1 x 1 - 1	1

https://medium.com/@stanleydukor/neural-representation-of-and-or-not-xor-and-xnor-logic-gates-perceptron-algorithm-b0275375fea1

An example: the OR gate

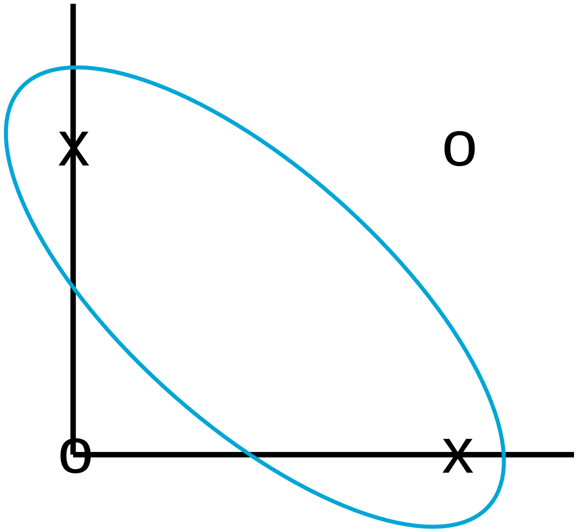
$$y = \begin{cases} 1 \text{ if } w^T x + b >= 0 \\ 0 \text{ otherwise} \end{cases}$$

= Perceptron algorithm

x_1	x_2	Label	Calculation	Result	y
0	0	0	1 x 0 + 1 x 0 - 1	-1	0
0	1	1	1 x 0 + 1 x 1 - 1	0	1
1	0	1	1 x 1 + 1 x 0 - 1	0	1
1	1	1	1 x 1 + 1 x 1 - 1	1	1

☑

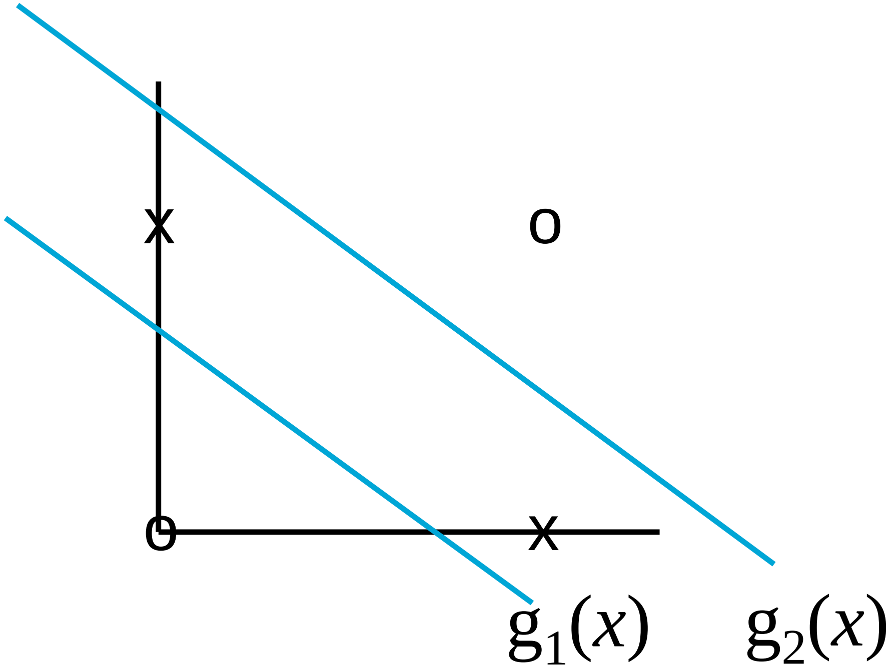
Let's go back to the XOR problem



- $y = \ 1 \times x_1 + 1 \times x_2 - 1$
- $y = \begin{cases} 1 \ \text{if } w^T x + b >= 0 \\ 0 \ \text{otherwise} \end{cases}$

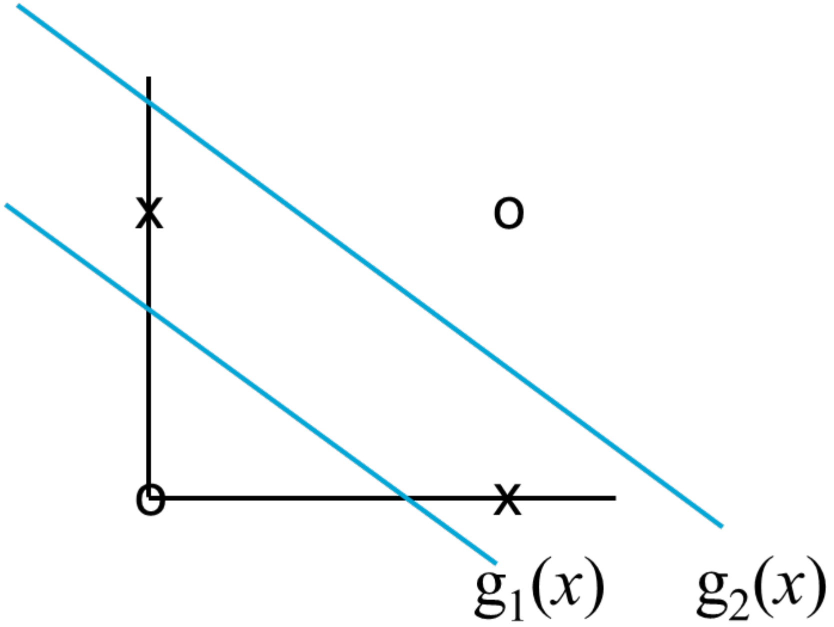
x_1	x_2	Label	Calculation	Result	y
0	0	0	1 x 0 + 1 x 0 - 1	-1	0
0	1	1	1 x 0 + 1 x 1 - 1	0	1
1	0	1	1 x 1 + 1 x 0 - 1	0	1
1	1	0	1 x 1 + 1 x 1 - 1	1	1

Multi-layer perceptrons



Two consecutive steps

1. Calculate the position of the data point to *each* of the decision boundaries
2. Combine the results of 1. to determine position of the data point to *both* decision boundaries and determine class



A multi-layer perceptron

Output layer

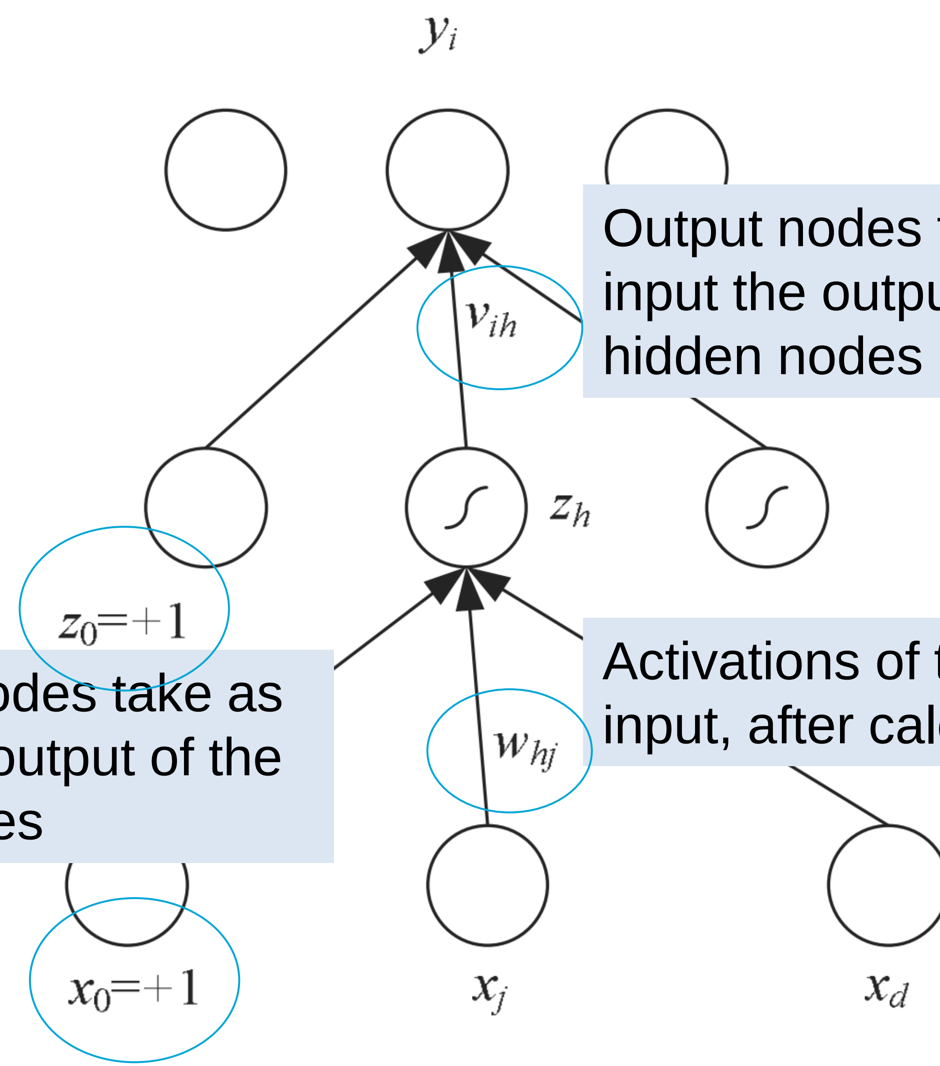
- Computations

Hidden layer

- Computations
- Bias

Input layer:

- No computations
- Bias



Output nodes take as input the output of the hidden nodes

Activations of the input, after calculations

Hidden nodes take as input the output of the input nodes

Activation functions

- So far: $y = \begin{cases} 1 & \text{if } w^T x + b >= 0 \\ 0 & \text{otherwise} \end{cases}$

➔ For linear problems, on/off, 0/1

For non-linear problems, e.g.:

- sigmoid (see figure previous slide)
- tanh

XOR using a multi-layer perceptron

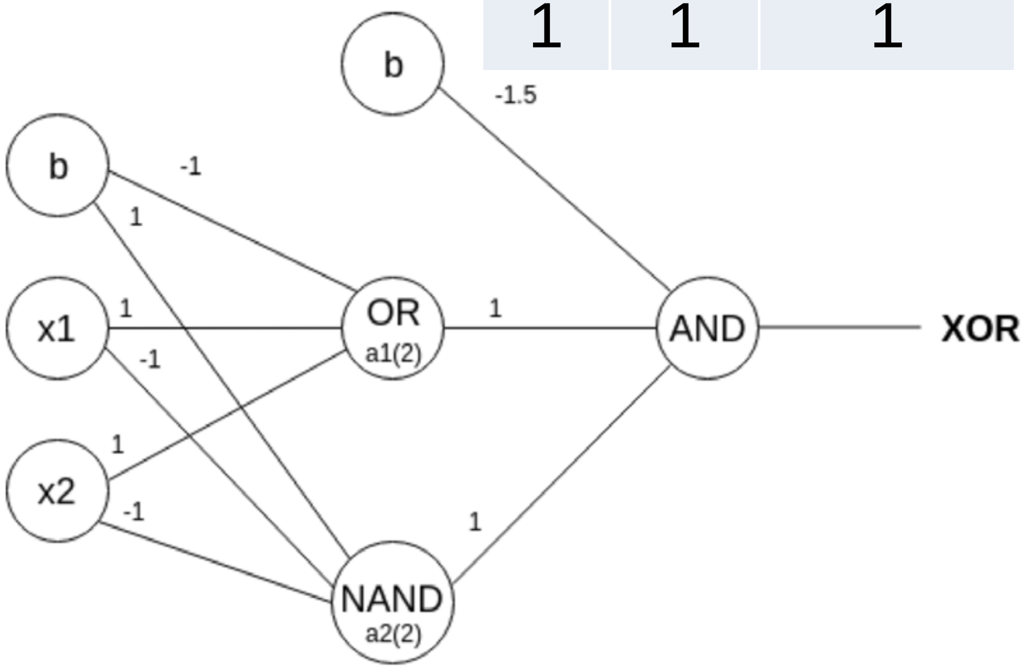
- Truth tabels:

OR

x_1	x_2	Label
0	0	0
0	1	1
1	0	1
1	1	1

NAND

x_1	x_2	Label
0	0	1
0	1	1
1	0	1
1	1	0



Hidden layer: OR + NAND gates

OR

x_1	x_2	Label	Calculation	Result	y
0	0	0	1 x 0 + 1 x 0 - 1	-1	0
0	1	1	1 x 0 + 1 x 1 - 1	0	1
1	0	1	1 x 1 + 1 x 0 - 1	0	1
1	1	1	1 x 1 + 1 x 1 - 1	1	1

NAND

x_1	x_2	Label	Calculation	Result	y
0	0	1	-1 x 0 - 1 x 0 + 1	1	1
0	1	1	-1 x 0 - 1 x 1 + 1	0	1
1	0	1	-1 x 1 - 1 x 0 + 1	0	1
1	1	0	-1 x 1 - 1 x 1 + 1	-1	0

NAND: $y = -1 \times x_1 - 1 \times x_2 + 1$

Output layer: AND gate

x_1	x_2	Label	Calculation	Result	y
0	1	0	1 x 0 + 1 x 1 – 1.5	-0.5	0
1	1	1	1 x 1 + 1 x 1 – 1.5	0.5	1
1	1	1	1 x 1 + 1 x 1 – 1.5	0.5	1
1	0	0	1 x 1 + 1 x 0 – 1.5	-0.5	0

☑

AND: $y = 1 \times x_1 + 1 \times x_2 - 1.5$

Training an MLP

- $y = w^T x + b$

- During training: learn the weights so that the value of y given x is correct

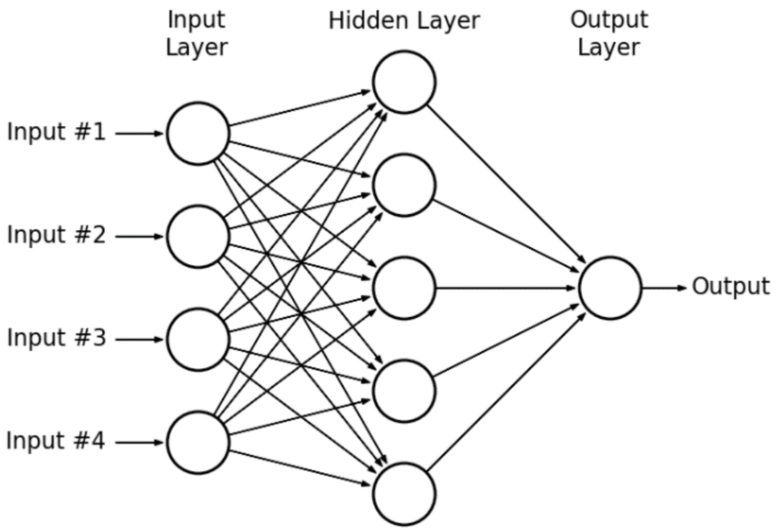
- Supervised learning with the true labels

➔ Stochastic gradient descent

Stochastic gradient descent

Feed-forward pass

- Initialise weights with random value
- Push input through the MLP row by row
- Calculate and push forward the activations
- Produce output value



Stochastic gradient descent

Backpropagation pass

- Output value is compared to the label/target
- Calculate error, using a loss function
- Error is *propagated back* through the network, layer by layer
- Update weights depending on how much they contributed to the error

➔ Find the weights that minimise the loss function

Minimise loss function

- To minimize the error, the *gradients* of the error function with respect to each weight are calculated

➔ Gradient vector indicates the direction of highest *increase* of a function

- We want: direction of the highest *decrease*

Iterative updating of the weights

- Recalculate the gradients at the beginning of each training iteration step
- STOP when error < θ OR max nb of iterations

- General formula to update the weights:

$$w_t \leftarrow w_{t-1} - \eta \frac{\partial E}{\partial w}$$

Partial derivatives of the error function ***E*** with respect to each weight of the array ***w***

Direction of the highest decrease

Learning rate of the network (step size)

TU Delft

- Epoch:
  - All data used once for updating weights
- Batch training:
  - Update weights after all training data are processed
- Online training:
  - Update weights after each training sample
- The amount that weights are updated: learning rate

TU Delft

Combining classifiers

Many different supervised models

- Combine different classifiers

+ exploit their individual strengths

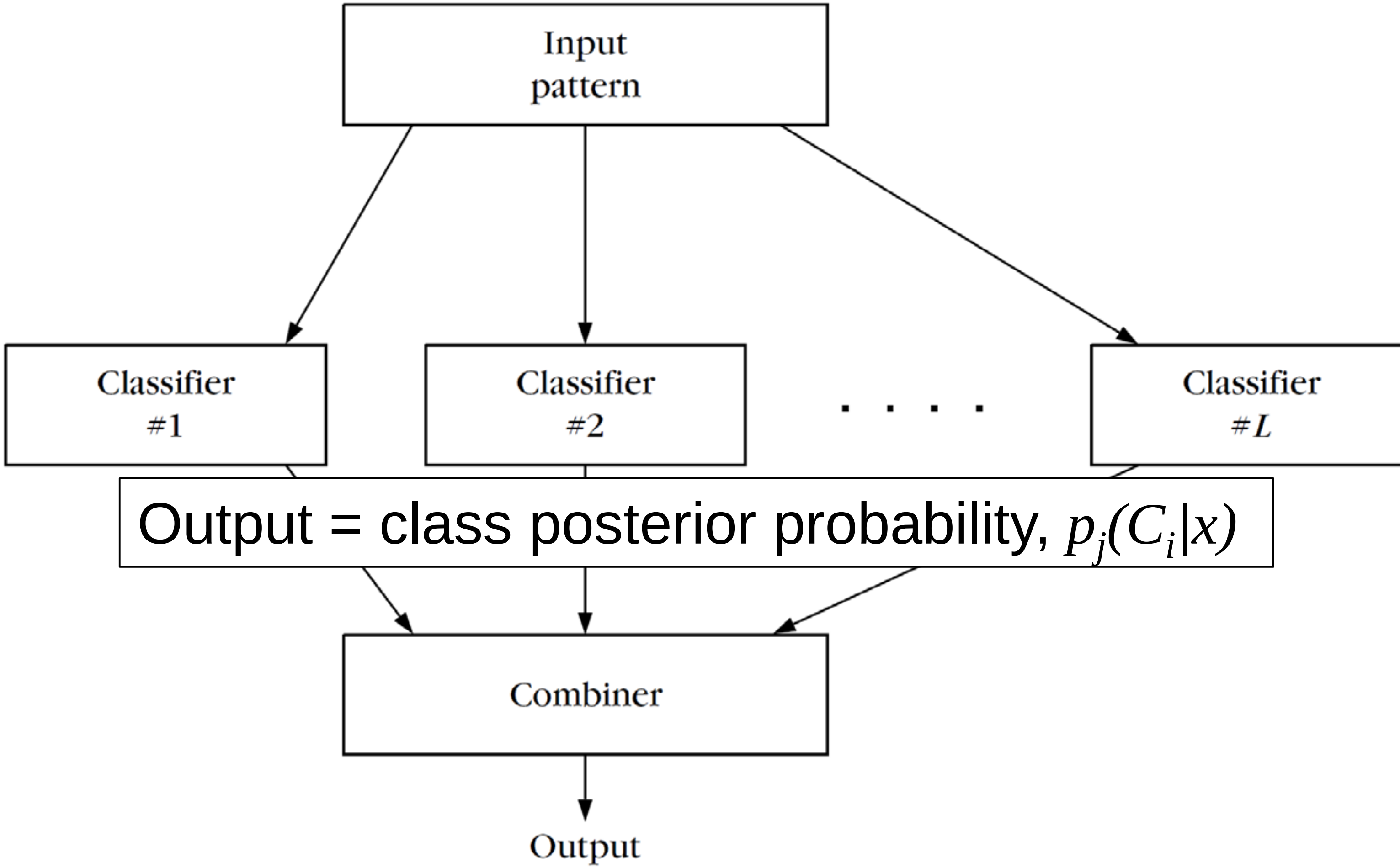
➔ Overall better performance than for individual classifiers

- Two classifiers might have the same accuracy

…. But make errors on different patterns

➔ Use the complementary information that seemingly resides in the different classifiers

General approach



Goal: improved class posterior probability, $P(C_i|x)$

Kullback-Leibler (KL)

- Probability distance measure

➔ Choose $P(C_i|x)$ = minimum average KL distance between the probabilities

➔ Assign unknown pattern to the class that maximises

$$\max_{\omega_i} \prod_{j=1}^{L} P_j(c_i|\boldsymbol{x})$$

(soft-type rule)

Majority voting

- Assign unknown pattern to class for which their is consensus:
  - E.g., in a 2 class problem: assign unknown pattern to class C if ½ + 1 classifiers agree on that class

Assumptions:

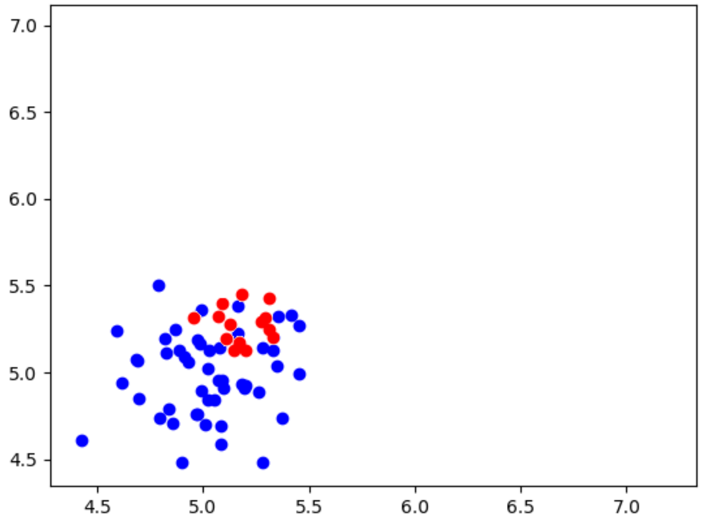
- Number of classifiers is odd
- Each classifier has the same probability of making a correct classification
- Decision by each classifier is taken independently of the other classifiers

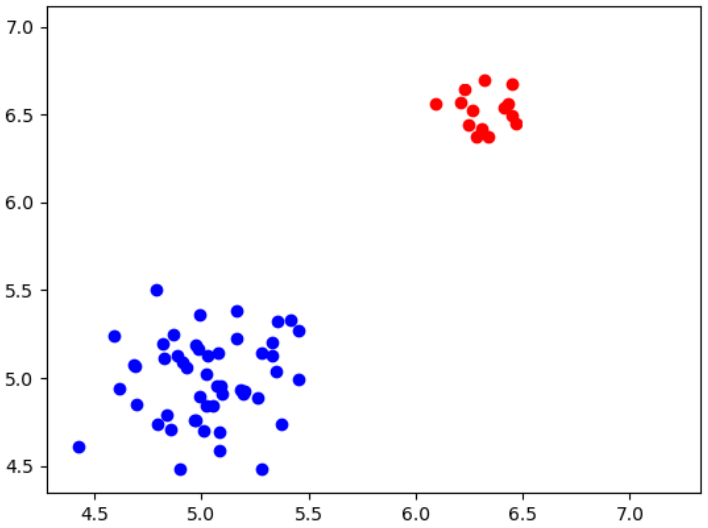
(hard-type rule)

Class imbalance problems

- In training data
- E.g., when wanting to detect a rare event
  - E.g., diagnosis of a medical condition

Q: A problem?





- Yes, it can be, but sometimes it is not

TU Delft

Possible solutions

- Data balance approach: rebalance the classes
  - Oversample the smaller class
  - Undersample the larger class
  - Random
  - Focussed, i.e., specifically points near/far from the boundaries

- Cost sensitive approach: use different parameters for the two classes in the cost function

➔ Happens often, be aware!

Summary

- Perceptron is a linear classifier
- Multi-layer perceptron:
  - A network of perceptrons
  - Each perceptron has an activation function
  - Non-linear classifier
  - Trained using stochastic gradient descent + backpropagation

- Different classifiers can be combined to build better classifiers

- Be aware of large class imbalances in the training data