

Artificial Neural Networks

Wulfram Gerstner

EPFL, Lausanne, Switzerland

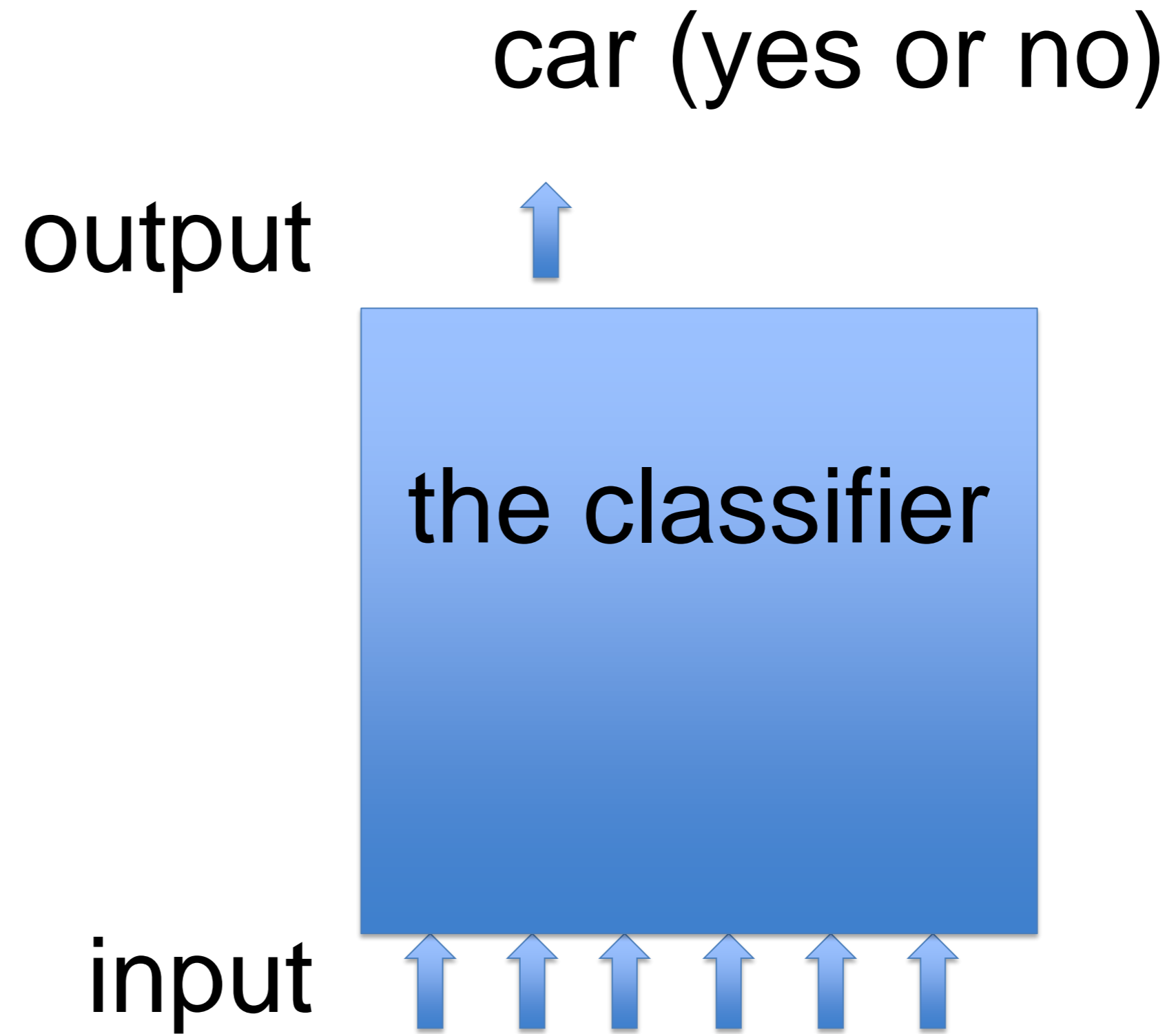
Supervised learning, classification, simple perceptron

1. Classification as a geometric problem

Previous slide.

... and now we really start.

The problem of Classification



Previous slide.

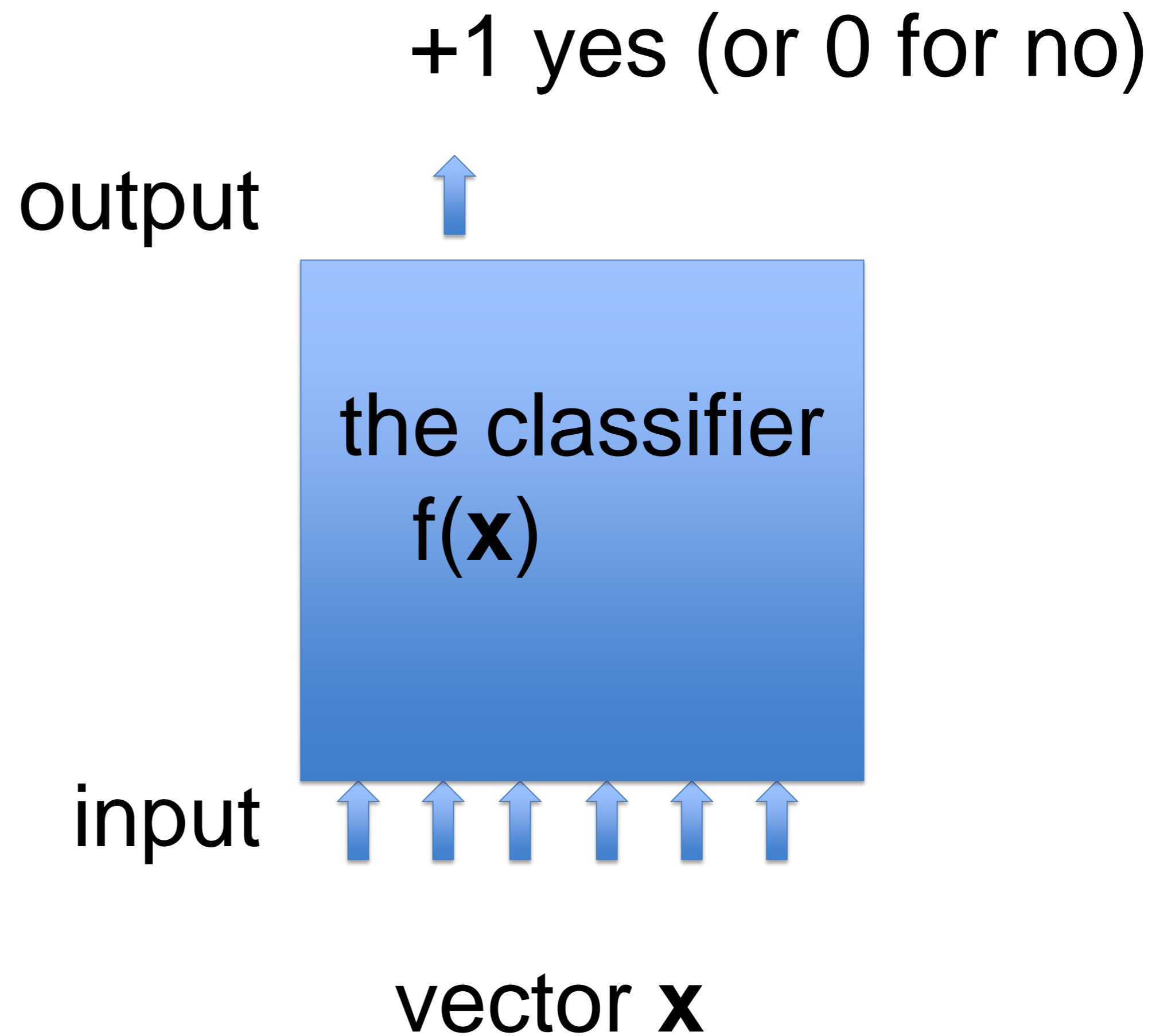
We focus on the task of classification.

To be concrete we consider images. The task of the classifier is to say: yes or no.

In the concrete example: 'yes' means that there is a car on the image

Even though we visualize the input as a two-dimensional image, the input to the networks really just is a vector \mathbf{x} of pixel values (Blackboard 1)

The problem of Classification



Previous slide.

The input is a vector \mathbf{x}

The classifier is a function $f(\mathbf{x})$
that maps the input to the output

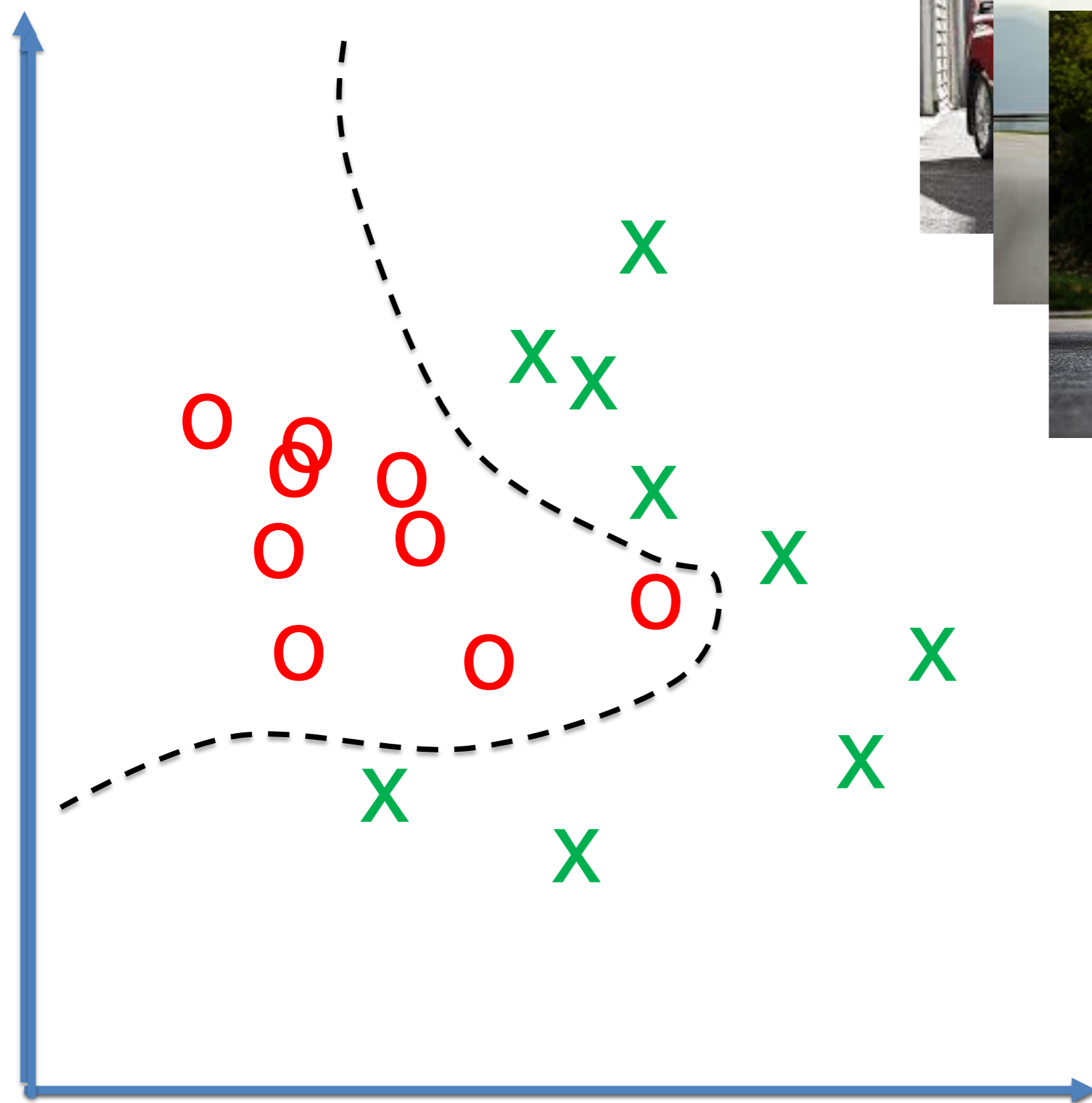
The output is binary: +1 or 0.

Blackboard 1:
from images to vector

Blackboard 2:

from vectors to classification

Classification as a geometric problem



Blackboard 1:
from images to vector

Blackboard 2:
from vectors to classification

Previous slide.

Classification means: assigning a +1 to some inputs (e.g. cars) and 0 to other inputs (not cars).

Classification corresponds to a separating (by some surface) the positive examples (green crosses) from the negative ones (red circles).

The space is the space of input vectors \mathbf{x}

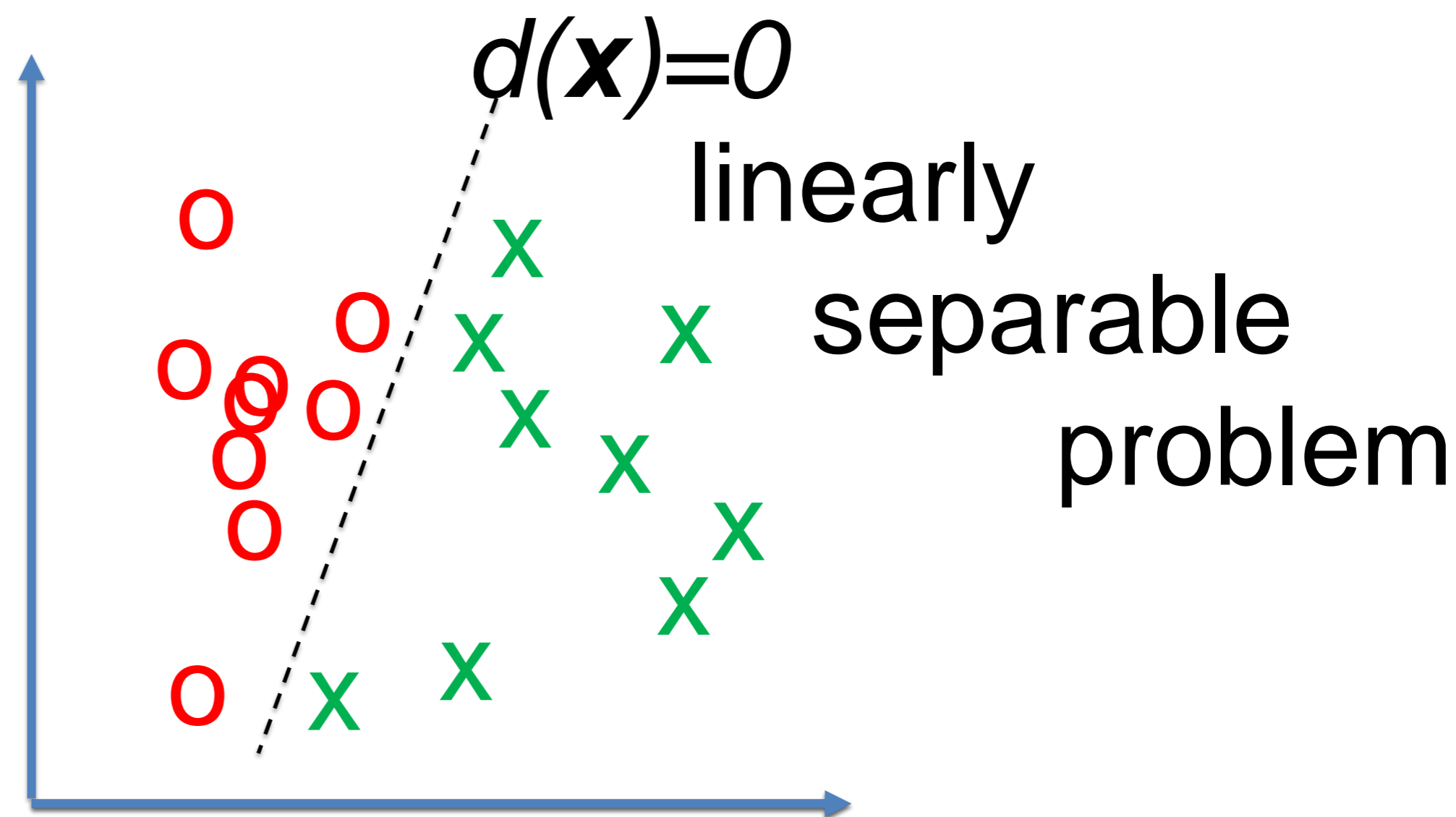
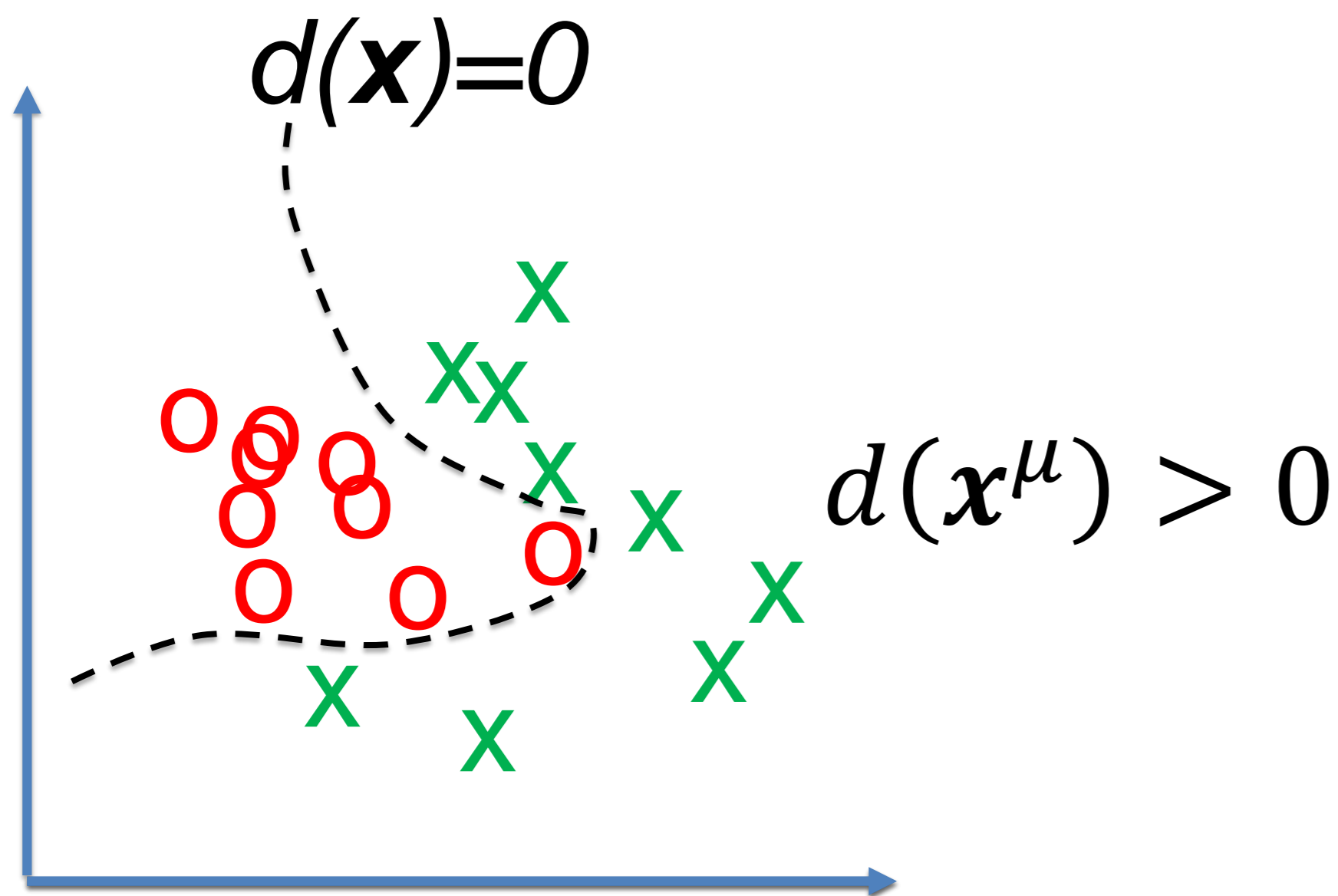
Classification as a geometric problem

Task of Classification

= find a **separating surface** in the high-dimensional input space

Classification by **discriminant function** $d(\mathbf{x})$

→ $d(\mathbf{x})=0$ on this surface; $d(\mathbf{x})>0$ for all positive examples \mathbf{x}
 $d(\mathbf{x})<0$ for all counter examples \mathbf{x}



Previous slide.

The discriminant function $d(\mathbf{x})$ takes inputs \mathbf{x} and maps these to:

$d(\mathbf{x}) > 0$ for all positive examples \mathbf{x}

$d(\mathbf{x}) < 0$ for all counter examples \mathbf{x}

$d(\mathbf{x}) = 0$ on the separating surface

Solving a classification problem therefore is equivalent to finding a discriminant function.

A linear discriminant function will become important in the next section.

Artificial Neural Networks

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Supervised learning, classification, simple perceptron

1. Classification as a geometric problem
2. Supervised learning

Previous slide.

We now turn to supervised learning in order to learn a classifier. Remember that a classifier implements a discriminant function.

Data base for Supervised learning

target output $t^\mu = 1$

output

$\hat{y}^\mu = 1$

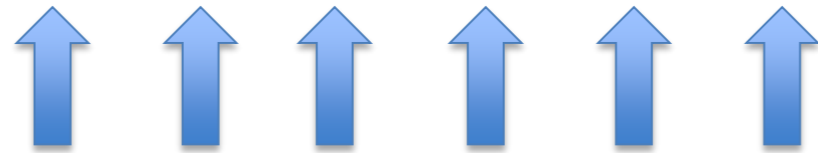
classifier output

car (yes)

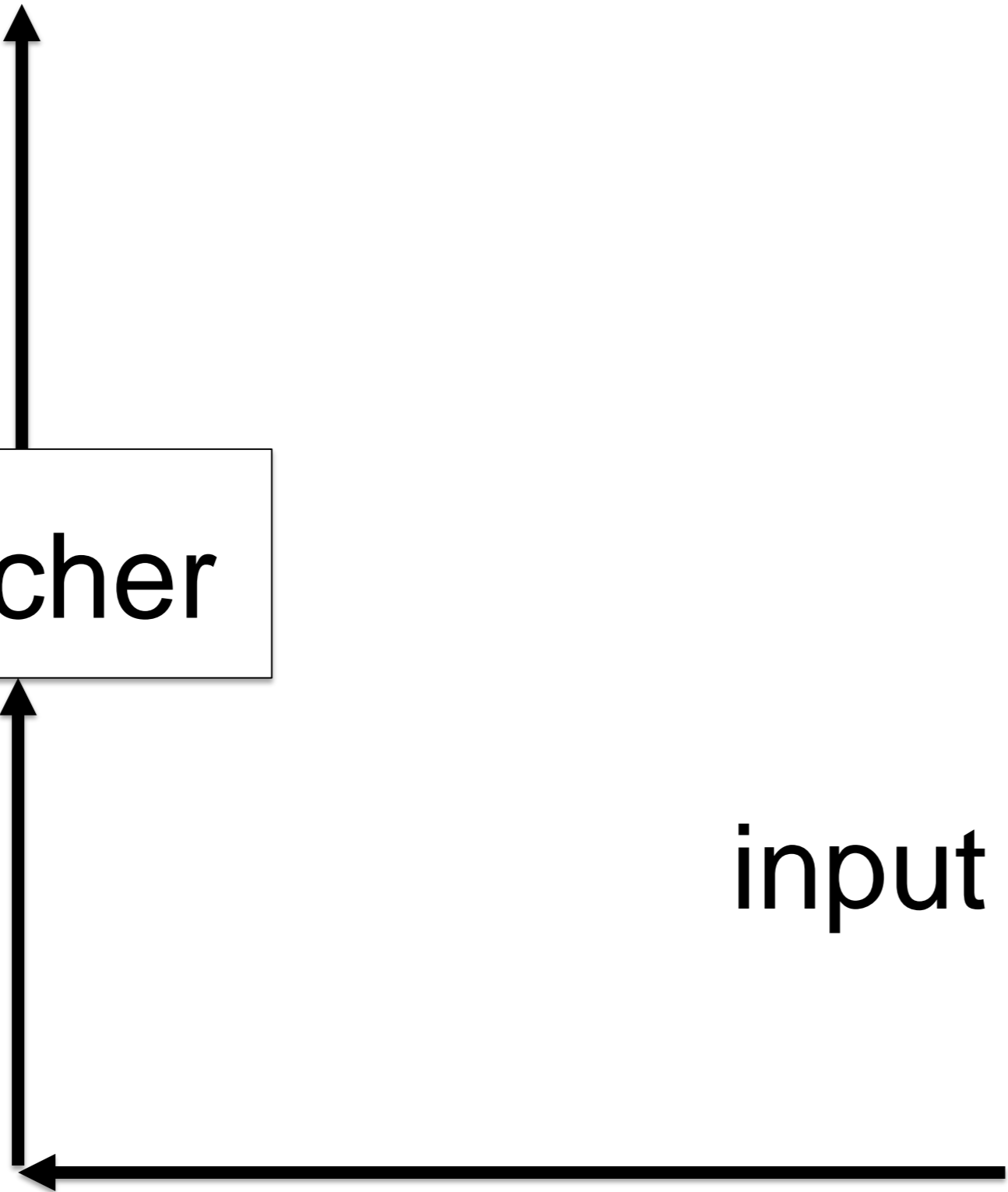
teacher



input



x^μ



Previous slide.

To construct such a discriminant function we need a data base for supervised learning.

The problem is called supervise learning because we assume that a teacher has previously looked at the examples and assigned labels.

A label $t^\mu = 1$ for an input vector x^μ means that this input pattern belongs to the class (positive example)

A label $t^\mu = 0$ for an input vector x^μ means that this input pattern does not belong to the class (counter-example)

Supervised learning

P data points $\{ (x^\mu, t^\mu) , \quad 1 \leq \mu \leq P \};$

input target output

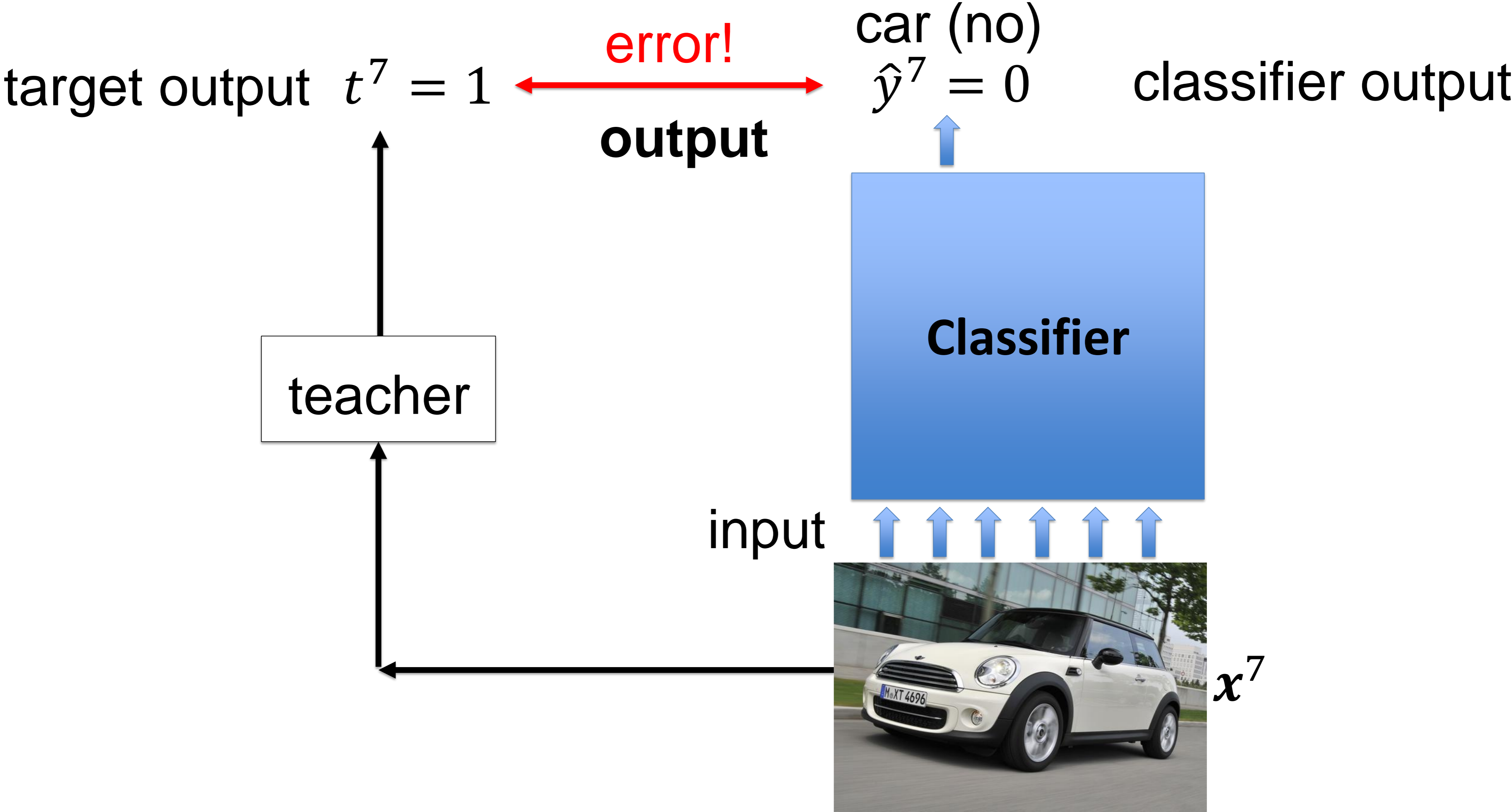
$t^\mu = 1$ car =yes

$t^\mu = 0$ car =no

Previous slide.

The data base for supervised learning contains P data points, each consisting of a pair of input and target output.

Data base for Supervised learning




Previous slide and next slide.

The basic idea of supervised learning is that the actual output of the classifier is compared with the target output. If there is a mismatch, then the error can be used to optimize the function $f(x)$ of the classifier.

Error in Supervised learning

P data points $\{ (x^\mu, t^\mu) , \quad 1 \leq \mu \leq P \};$


input target output

for each data point x^μ , the classifier gives an output \hat{y}^μ

→ use errors $\hat{y}^\mu \neq t^\mu$ for optimization of classifier

Remark: Errors can be used to define a ‘Loss function’.

Remark: for multi-class problems y and t are vectors

Previous slide.

A single-class classifier has a single binary target output $t^\mu = 0$ or 1 .

For a multi-class classifier the target output is a vector.

Summary: Supervised learning

1. Data base $\{ (x^\mu, t^\mu) , \quad 1 \leq \mu \leq P \}$;

\uparrow \uparrow
input target output

2. A way to measure errors

for x^μ compare classifier output \hat{y}^μ with t^μ

$\sum_{\mu} E (\hat{y}^\mu, t^\mu)$ Error function/Loss function

3. A method to minimize the errors

Previous slide.

Supervised learning needs three things:

A data base with labels t^μ

A way to measure the differences between target t^μ and actual output y^μ

This is usually done with a loss function (also called error function)

A method to minimize the errors by changing the parameters of the classifier. In the case of Neural Networks: change the parameters of the network. We will see a first method in the next section

Artificial Neural Networks

Wulfram Gerstner

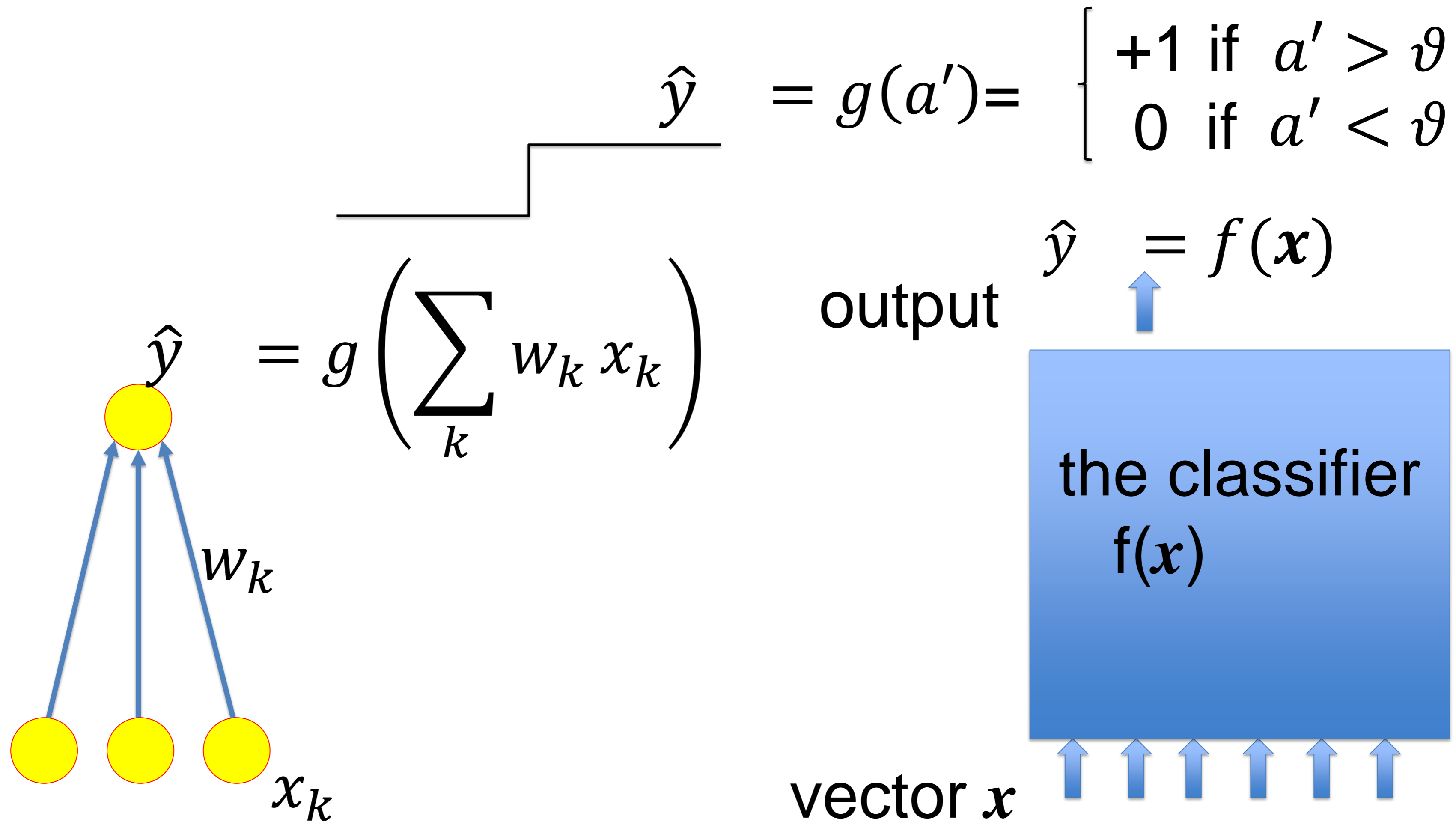
EPFL, Lausanne, Switzerland

Supervised learning, classification, simple perceptron

1. Classification as a geometric problem
2. Supervised learning
3. Simple Perceptron

Your notes.

3. Single-Layer networks: simple perceptron



Previous slide.

So far we have not specified the function $f(x)$ of the classifier.

Now we assume that the classifier consists of a single artificial model neuron.

Each component x_k of the input vector is multiplied by a weight w_k .

The function $g(\)$ is some nonlinear function.

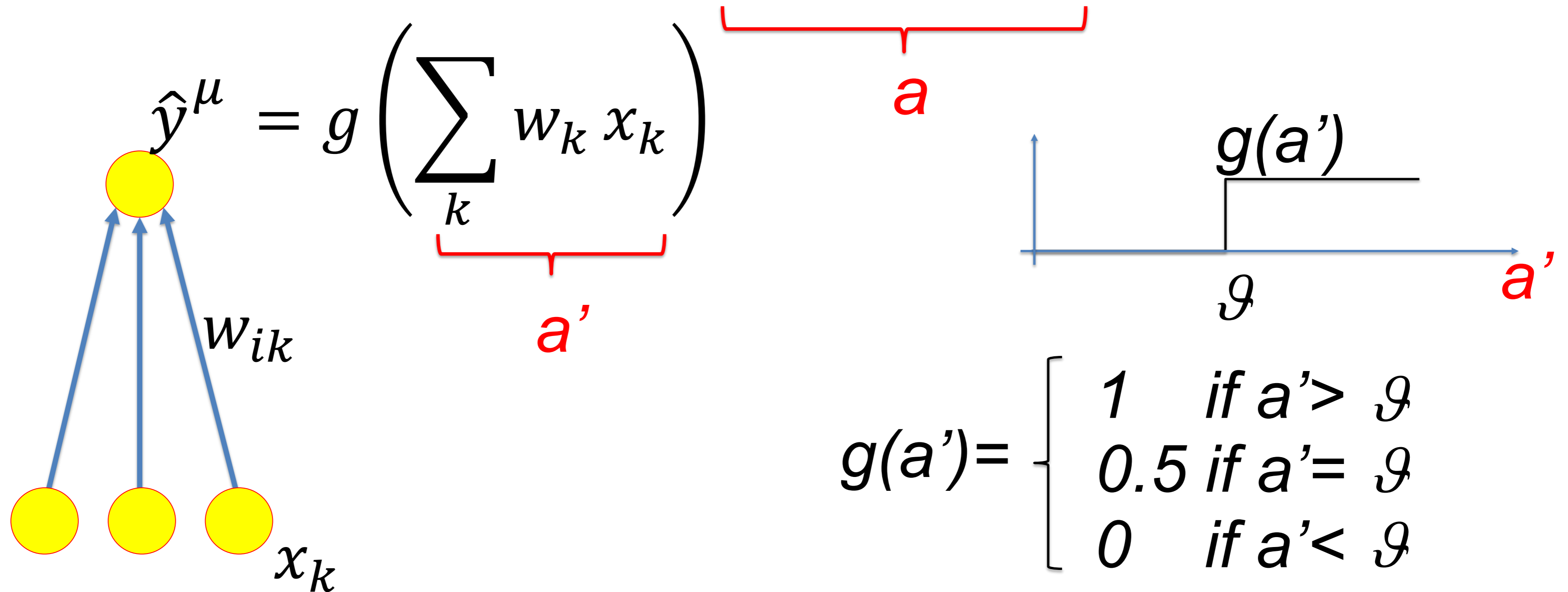
*Blackboard 3: Geometry of
perceptron: hyperplane*

Your notes.

Single-Layer networks: simple perceptron

$$\hat{y}^\mu = 0.5[1 + \text{sgn}(\sum_k w_k x_k - \vartheta)]$$

output



input

vector x

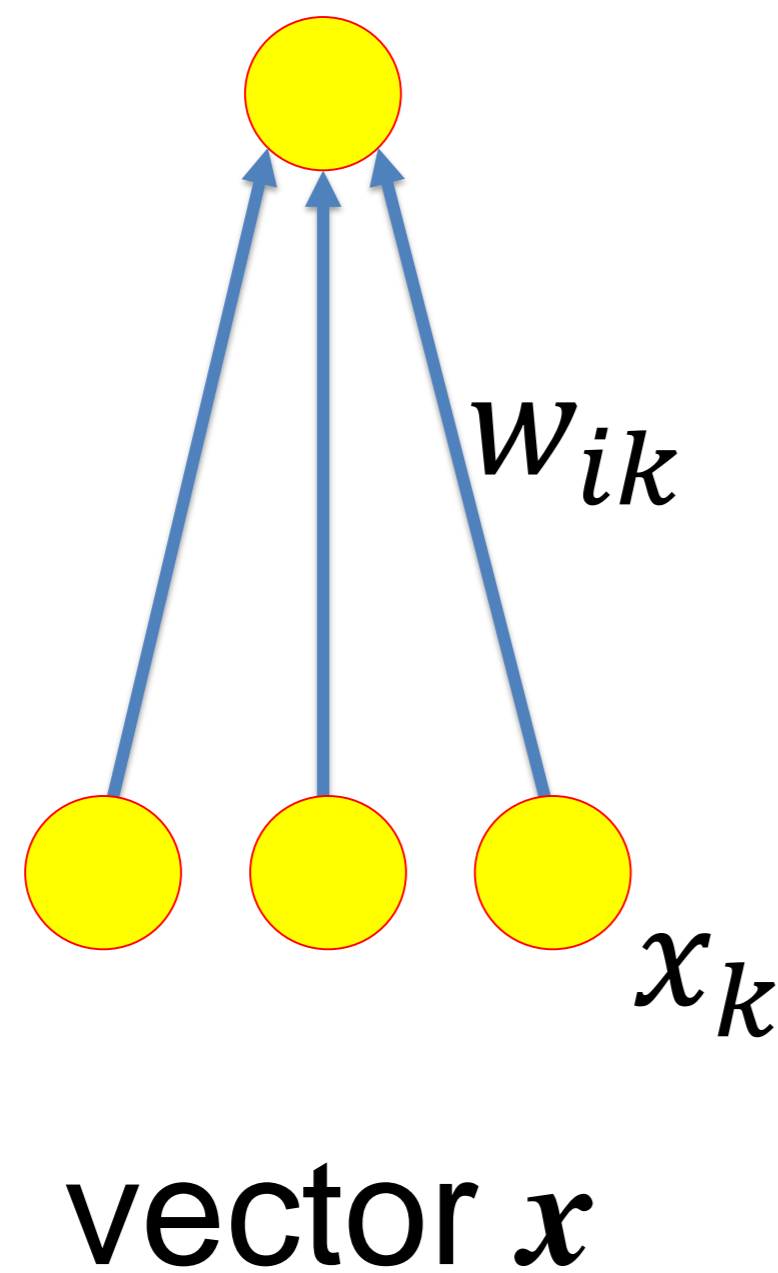
Previous slide.

Top line: Often we choose for g a step function with threshold ϑ .

The total effective input activation of the neuron is called \mathbf{a} (including the threshold) or \mathbf{a}' (before the threshold is subtracted).

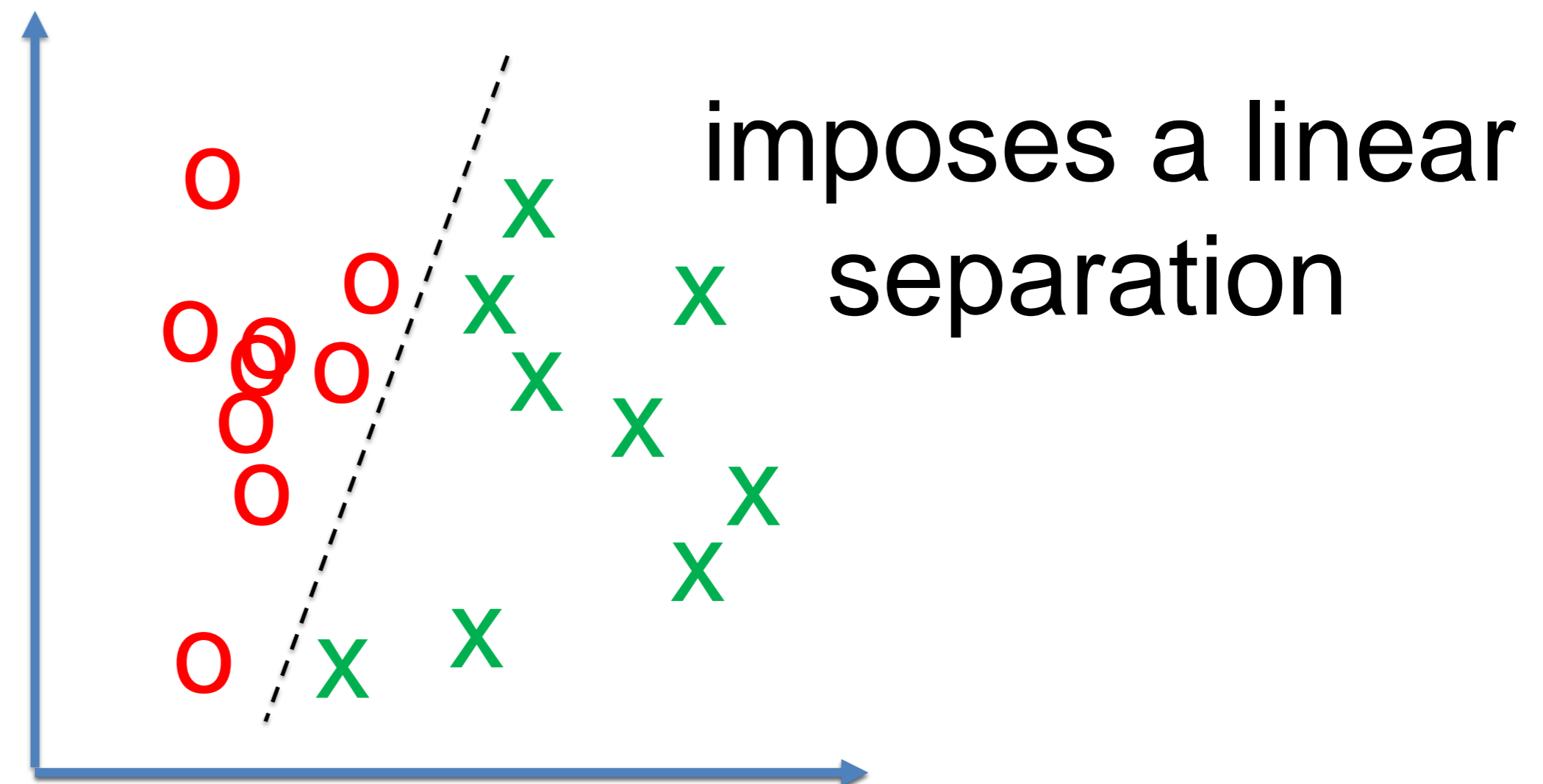
Single-Layer networks: simple perceptron

$$\hat{y} = 0.5[1 + \text{sgn}(\sum_k w_k x_k - \vartheta)]$$



Discriminant function

$$d(\mathbf{x}) = \sum_k w_k x_k - \vartheta = 0$$



Previous slide.

A single artificial model neuron implements a linear separation of the positive and negative examples. Thus the discriminant function is a hyperplane.

Consider as a discriminant function

$$d(\mathbf{x}) = \sum_{k=1}^N w_k x_k - \vartheta = 0$$

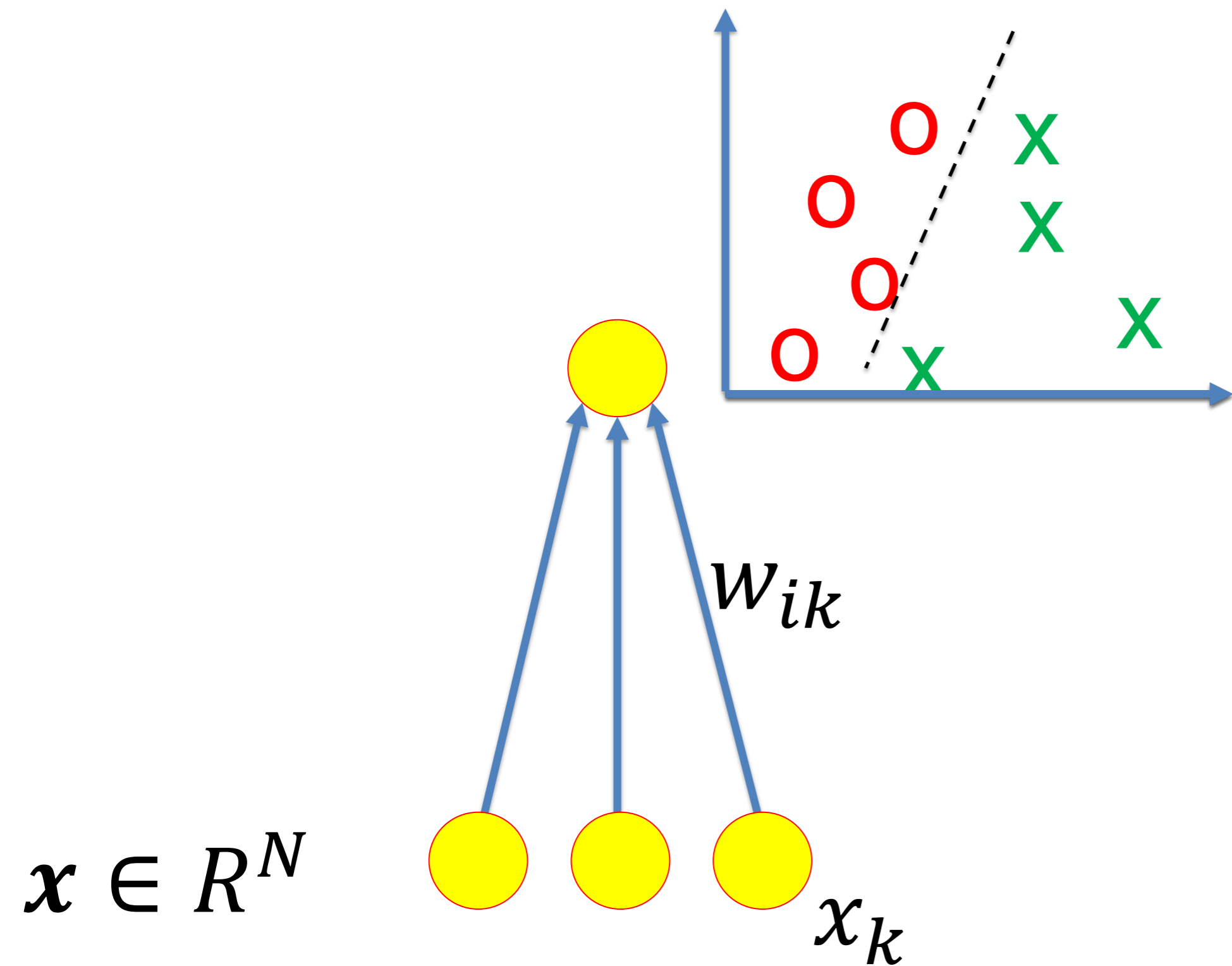
The output of the simple perceptron is a nonlinear function (step function) applied on $d(\mathbf{x})$. The Critical case is $d(\mathbf{x})=0$.

Positive examples have $d(\mathbf{x}) > 0$.

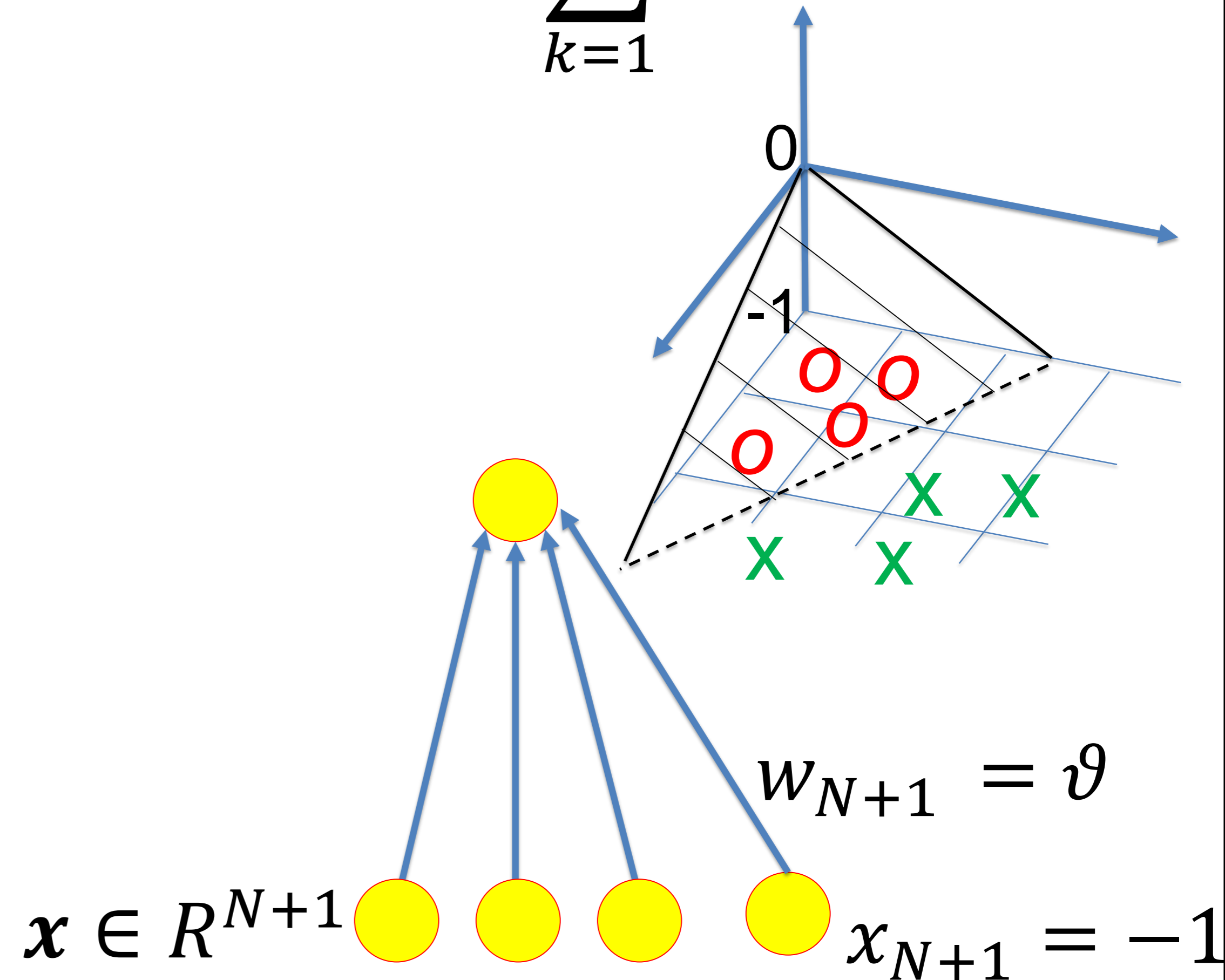
Negative examples have $d(\mathbf{x}) < 0$.

remove threshold: add a constant input

$$d(\mathbf{x}) = \sum_{k=1}^N w_k x_k - \vartheta = 0$$



$$d(\mathbf{x}) = \sum_{k=1}^{N+1} w_k x_k = 0$$



Previous slide.

The hyperplane has a distance $\vartheta / |w|$ from the origin.

Formally, we can represent the threshold by an additional weight $w_{N+1} = \vartheta$ which is multiplied with a constant input $x_{N+1} = -1$.

In this $(N+1)$ -dimensional space, the hyperplane passes through the origin.

Single-Layer networks: simple perceptron

a simple perceptron

- can only solve linearly separable problems
- imposes a separating hyperplane
- for $\vartheta = 0$ hyperplane goes through origin
- threshold parameter ϑ can be removed by adding an input dimension
- in **$N+1$** dimensions hyperplane always goes through origin
- we can **adapt the weight vector** to the problem: this is called 'learning'

Previous slide.

Thus, a simple perceptron can only solve linearly separable problems. Important for the following is that the positioning of the hyperplane in the high-dimensional space can be changed by adapting the weight vector to the data base.

Artificial Neural Networks

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Supervised learning, classification, simple perceptron

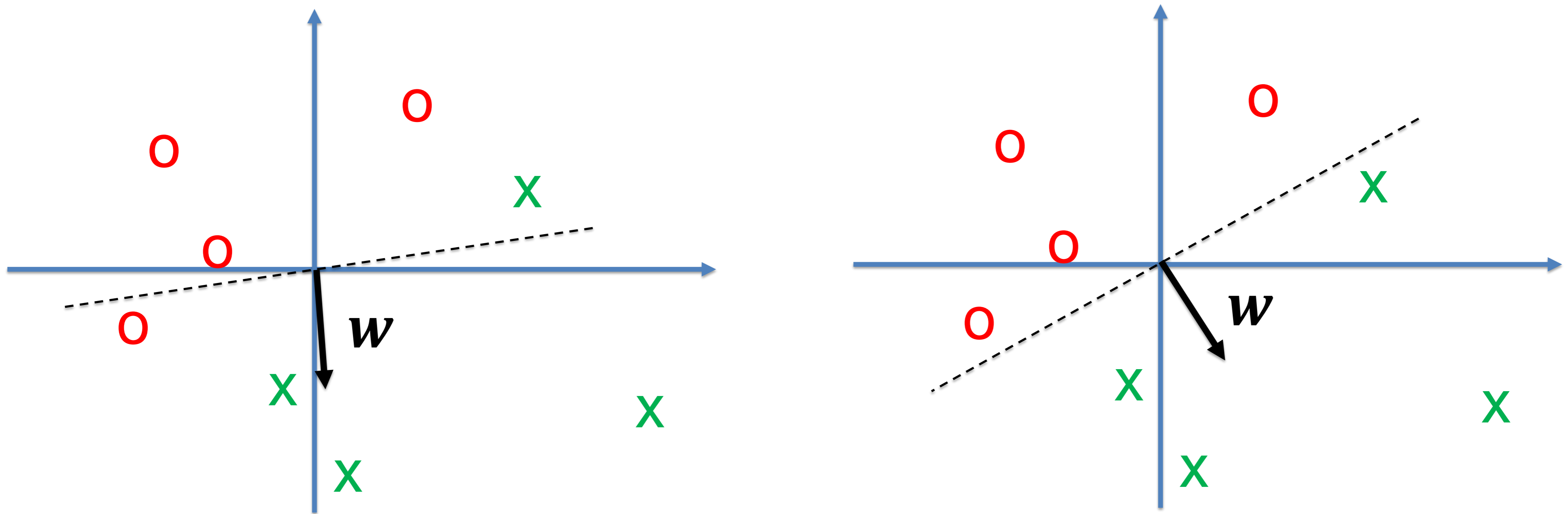
1. Classification as a geometric problem
2. Supervised learning
3. Simple Perceptron
4. Perceptron Algorithm

Previous slide.

We now study a first algorithm to minimize errors.

Perceptron algorithm: turn weight vector (in N+1 dim.)

$$\text{hyperplane: } d(\mathbf{x}) = \sum_{k=1}^{N+1} w_k x_k = \mathbf{w}^T \mathbf{x} = 0$$



idea: 'turn weight vector'

Previous slide.

In the following we always work in $N+1$ dimensions and exploit that the hyperplane goes through the origin.

Left: one of the examples is misclassified.

Right: all examples are correctly classified.

Idea: Turn weight vector in appropriate direction to go from the situation on the left to the situation on the right.

Perceptron algorithm

geometry of perceptron algorithm:

turn weight vector $\Delta \mathbf{w} \sim \mathbf{x}^\mu$

Perceptron algo (in $N+1$ dimensions):

- set $\gamma = 0.1$

(1) cycle many times through all patterns

- choose pattern μ

- calculate output

$$\hat{y}^\mu = 0.5[1 + \text{sgn}(\mathbf{w}^T \mathbf{x}^\mu)]$$

- update by

$$\Delta \mathbf{w} = \gamma [t^\mu - \hat{y}^\mu] \mathbf{x}^\mu$$

- iterate $\mu \leftarrow (\mu + 1) \bmod P$, back to (1)

(2) stop if no changes for all P patterns

Previous slide.

A change of the weight vector (during the update step) happens only if the actual output \hat{y}^μ for pattern x^μ is not equal to the target output t^μ .

And the change of the vector w is proportional to x^μ of the misclassified pattern.

parallel if target +1, antiparallel if target 0.

Classic Perceptron algo:

The cycling can be deterministic (loop through patterns in a fixed order)

Alternative:

You cycle stochastically (during a given cycle, pick each pattern once, but in random order).

*Blackboard 4: geometry of
the perceptron algorithm:
Turn weight vector*

output

$$\hat{y}^{\mu} = 0.5[1 + \text{sgn}(\mathbf{w}^T \mathbf{x}^{\mu})]$$

update

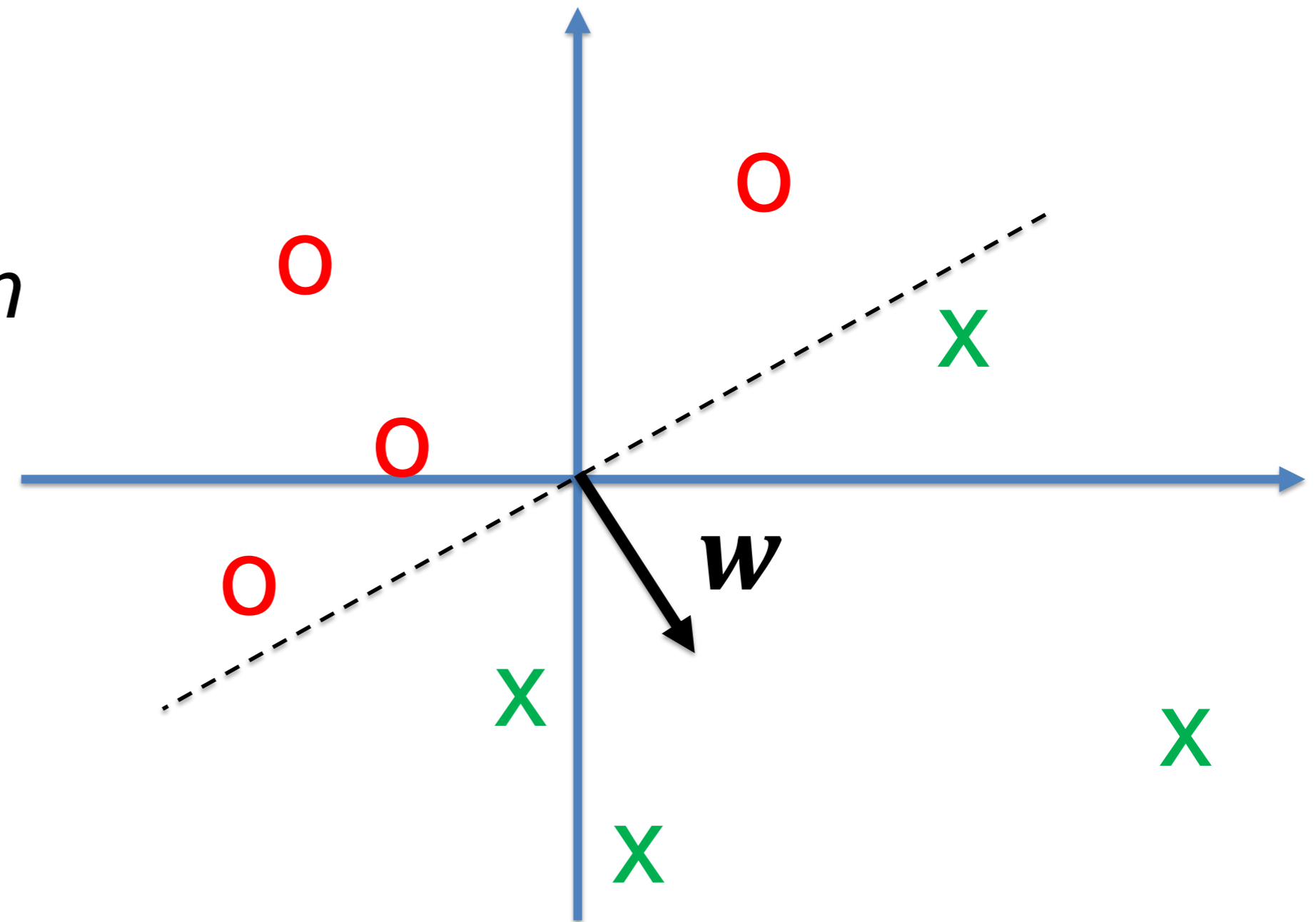
$$\Delta \mathbf{w} = \gamma [t^{\mu} - \hat{y}^{\mu}] \mathbf{x}^{\mu}$$

Your notes.

Perceptron algorithm: theorem

If the problem is linearly separable, the perceptron algorithm converges in a finite number of steps.

Proof: in many books, e.g.,
Bishop, 1995,
Neural Networks for Pattern Recognition



Previous slide.

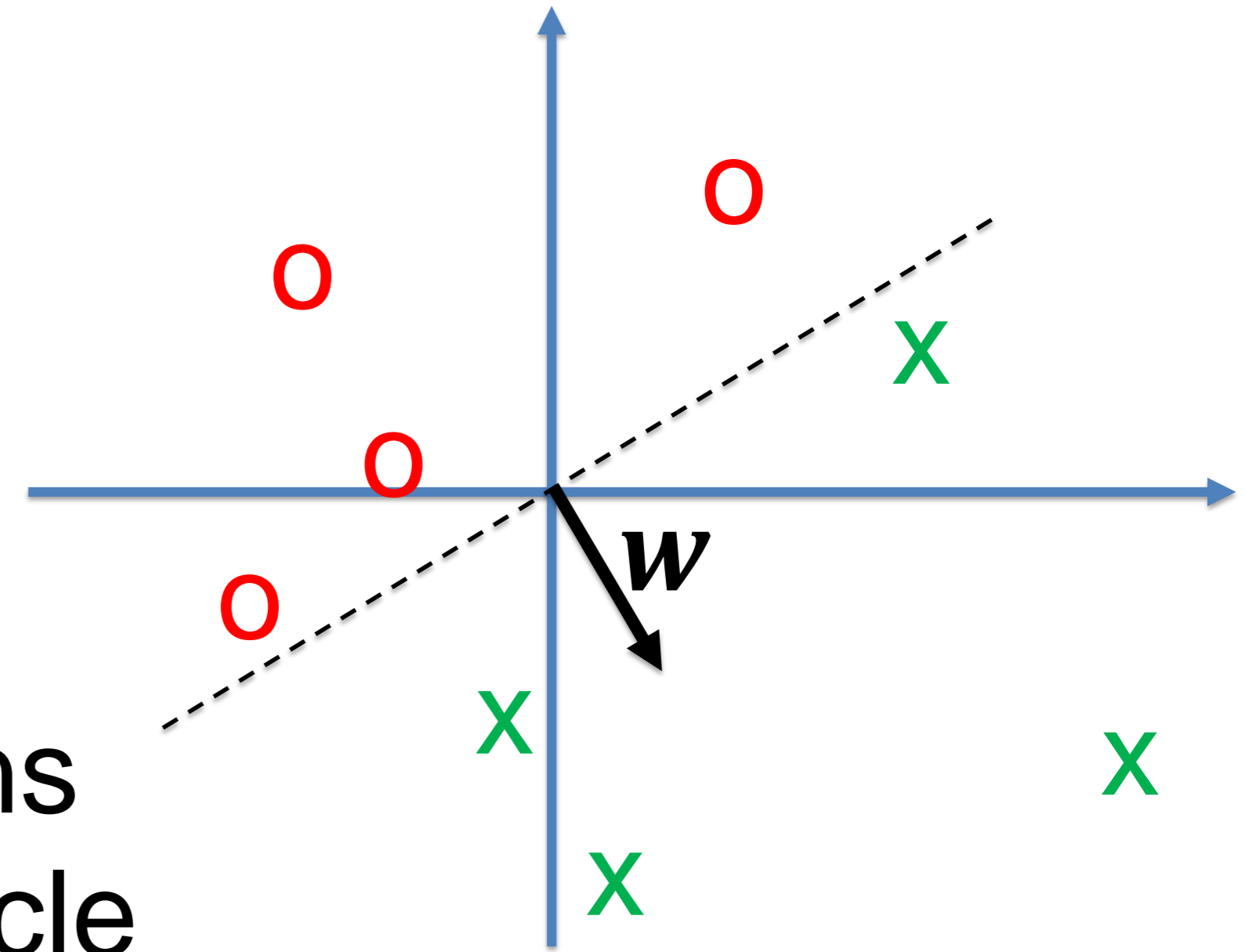
Important: Convergence is only guaranteed if the problem is linearly separable.

At the beginning, the weight vector may oscillate. However, the norm of the weight vector increases during learning, so that the angle by which it rotates gets smaller and smaller until it no longer changes (for linearly separable problems).

For problems that are not linearly separable, oscillations will continue for ever.

Summary: Perceptron algorithm

- Perceptron algorithm can solve linearly separable problems
- Cycle several times though all patterns until nothing changes during a full cycle
- Update proportional to weight vector $\Delta \mathbf{w} \sim \mathbf{x}^\mu$
- Proof shows:
 - initial value of \mathbf{w} not important
 - learning rate γ not importantReason: length of \mathbf{w} grows, but only direction matters



Your notes.

Quiz: Perceptron algorithm

The **input vector has N dimensions** and we apply a perceptron algorithm.

A change of parameters corresponds always to a rotation of the separating hyperplane in N dimensions.

A change of the separating hyperplane implies a rotation of the hyperplane in $N+1$ dimensions.

In the following change of length means $w + \Delta w = \beta w$ i.e., same direction

An increase of the length of the weight vector implies an increase of the distance of the hyperplane from the origin in N dimensions.

An increase of the length of the weight vector implies that the hyperplane does not change in N dimensions

An increase of the length of the weight vector implies that the hyperplane does not change in $N+1$ dimensions

Your notes.

Artificial Neural Networks

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Supervised learning, classification, simple perceptron

1. Classification as a geometric problem
2. Supervised learning
3. Simple Perceptron
4. Perceptron Algorithm
5. **Gradient descent and sigmoidal output unit**

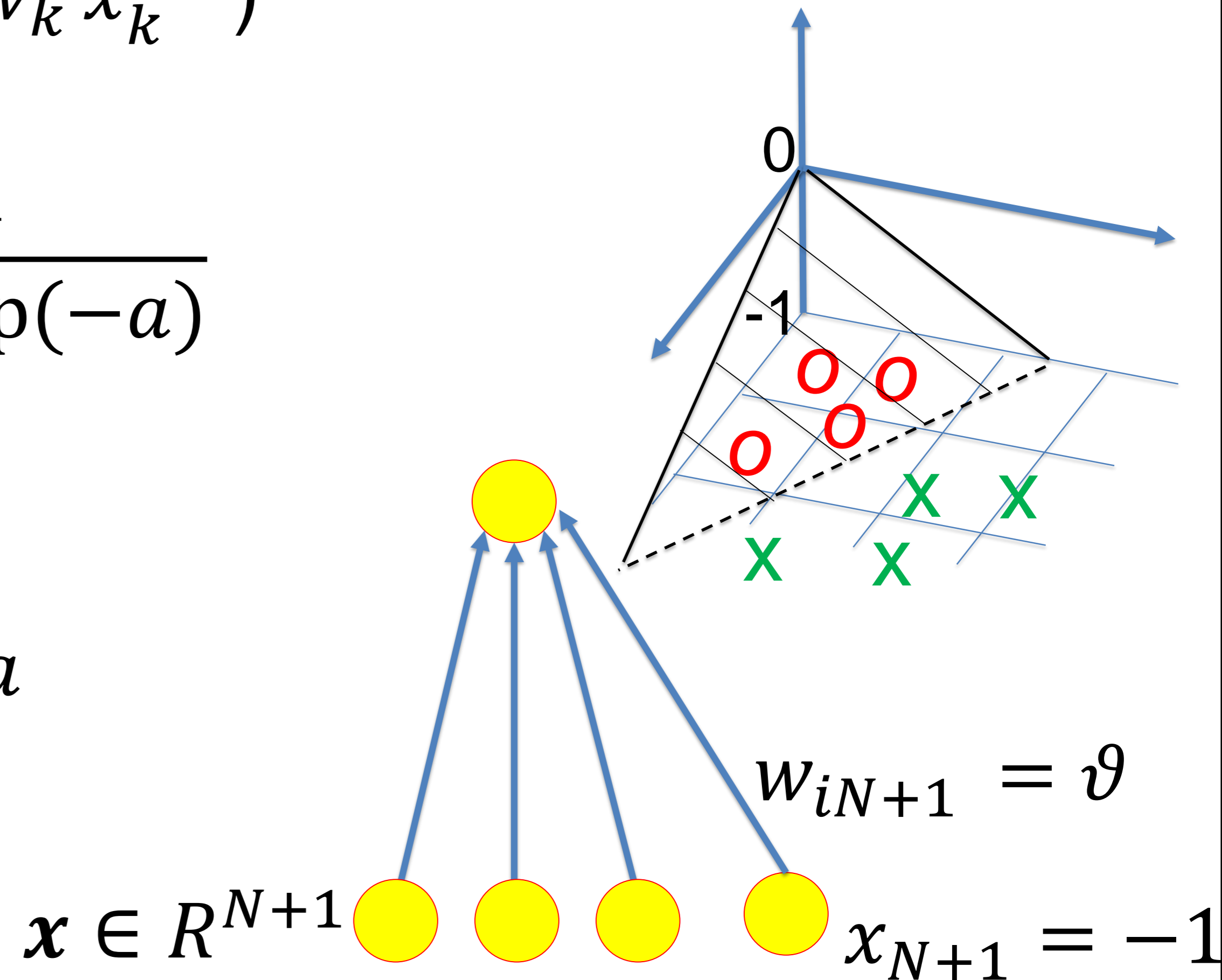
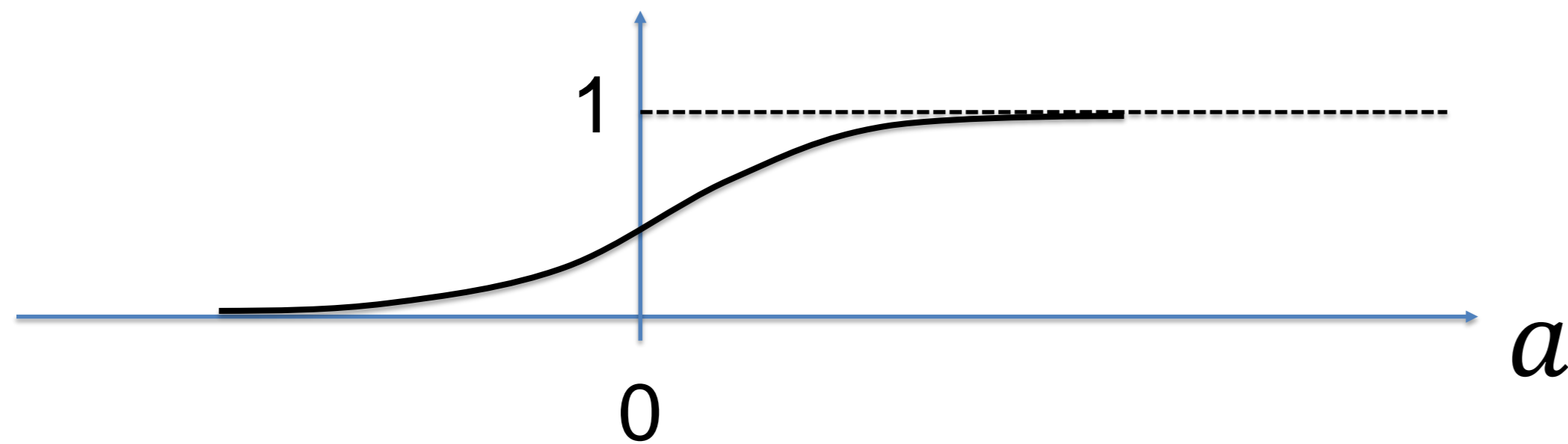
Sigmoidal output unit

A saturating nonlinear function with a smooth transition from 0 to 1.

$$\hat{y}^\mu = g(\mathbf{w}^T \mathbf{x}^\mu) = g\left(\sum_{k=1}^{N+1} w_k x_k^\mu\right)$$

with

$$g(a) = \frac{\exp(a)}{1 + \exp(a)} = \frac{1}{1 + \exp(-a)}$$



Previous slide.

We return to our choice of the nonlinear function $g()$.

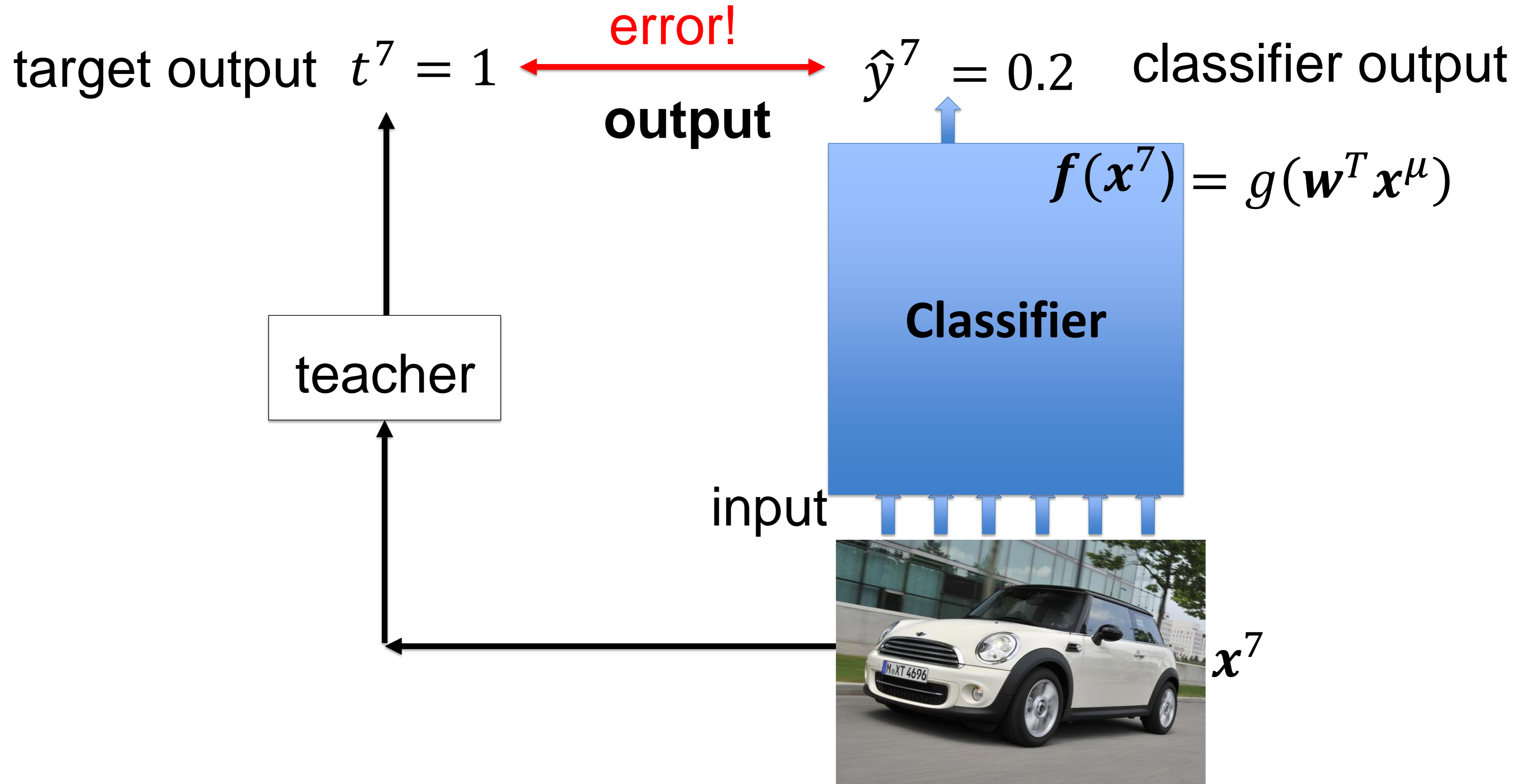
Instead of a threshold function, we can also work with a sigmoidal function.

The definition of the total input activation a is the same as before.

Instead of a step we now have a smooth transition from zero to one.

Note that the discriminant function is the same as for the simple perceptron with step output.

Supervised learning with sigmoidal output



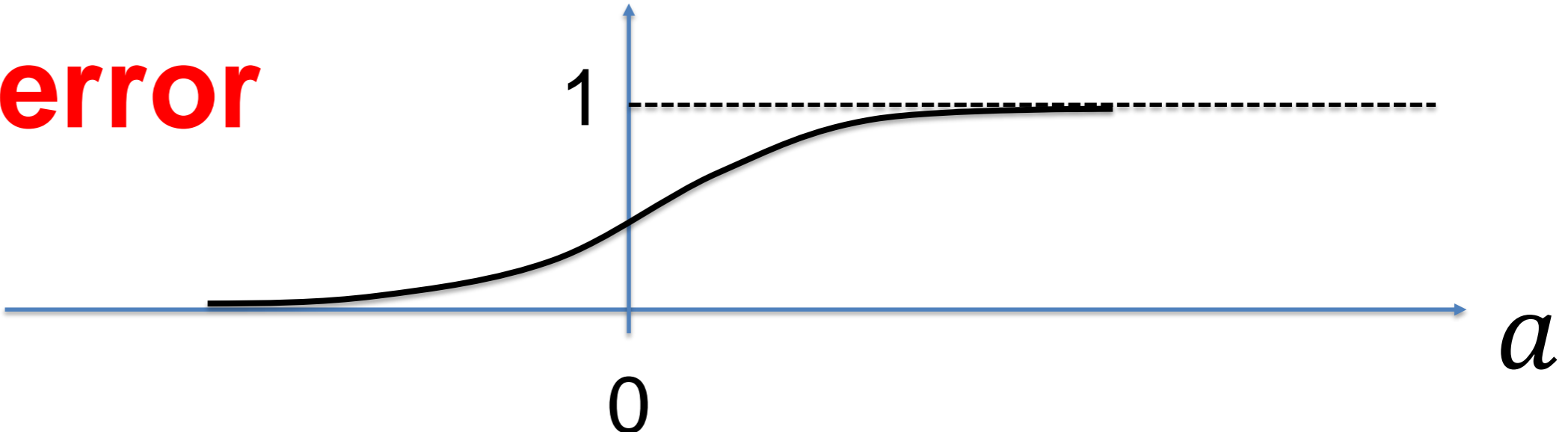
Previous slide.

The notion of mismatch in the output works for the smooth output neuron analogously to the case of the binary one that we have studied before

Supervised learning with sigmoidal output

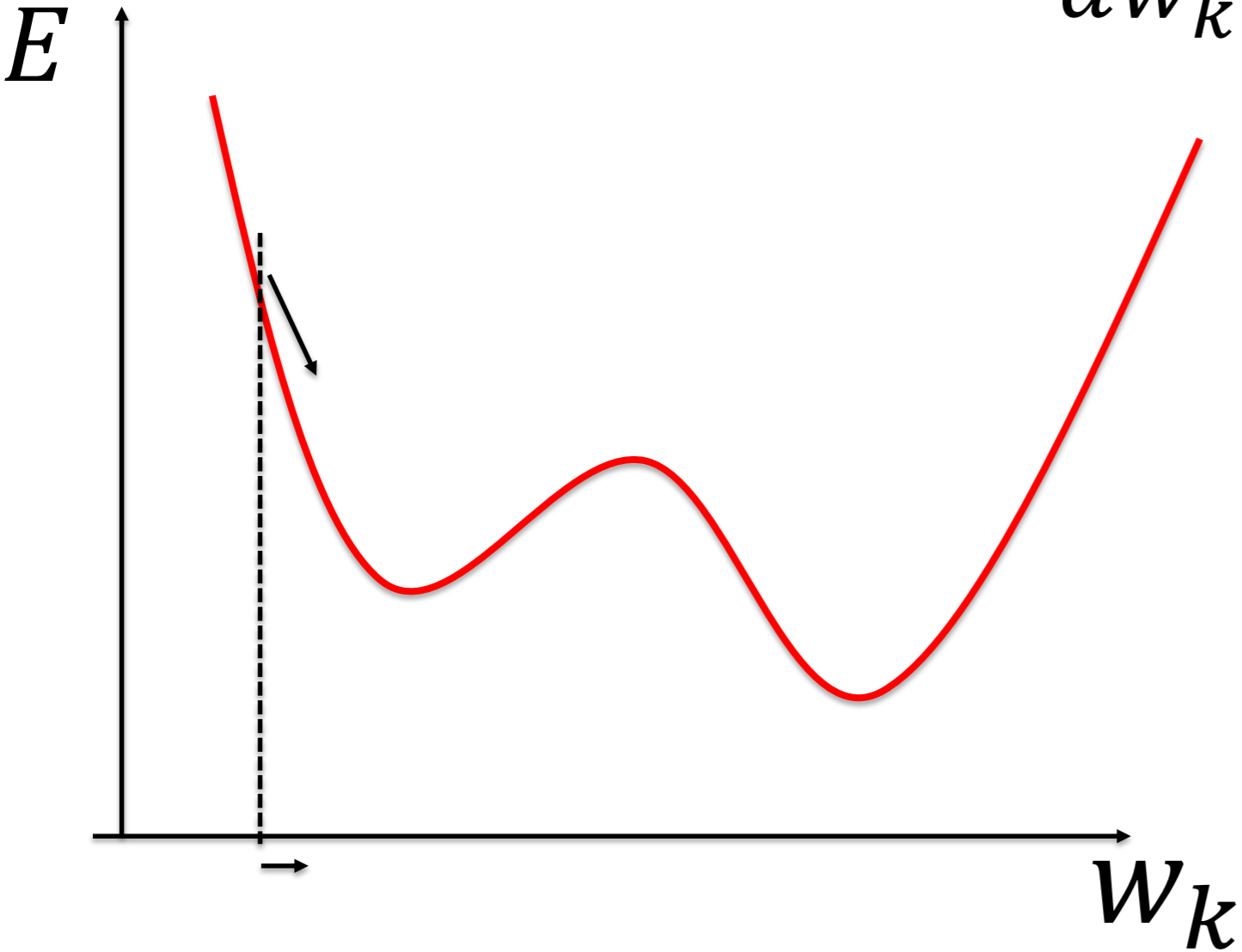
Loss function: define quadratic error

$$E(\mathbf{w}) = \frac{1}{2} \sum_{\mu=1}^P [t^{\mu} - \hat{y}^{\mu}]^2$$

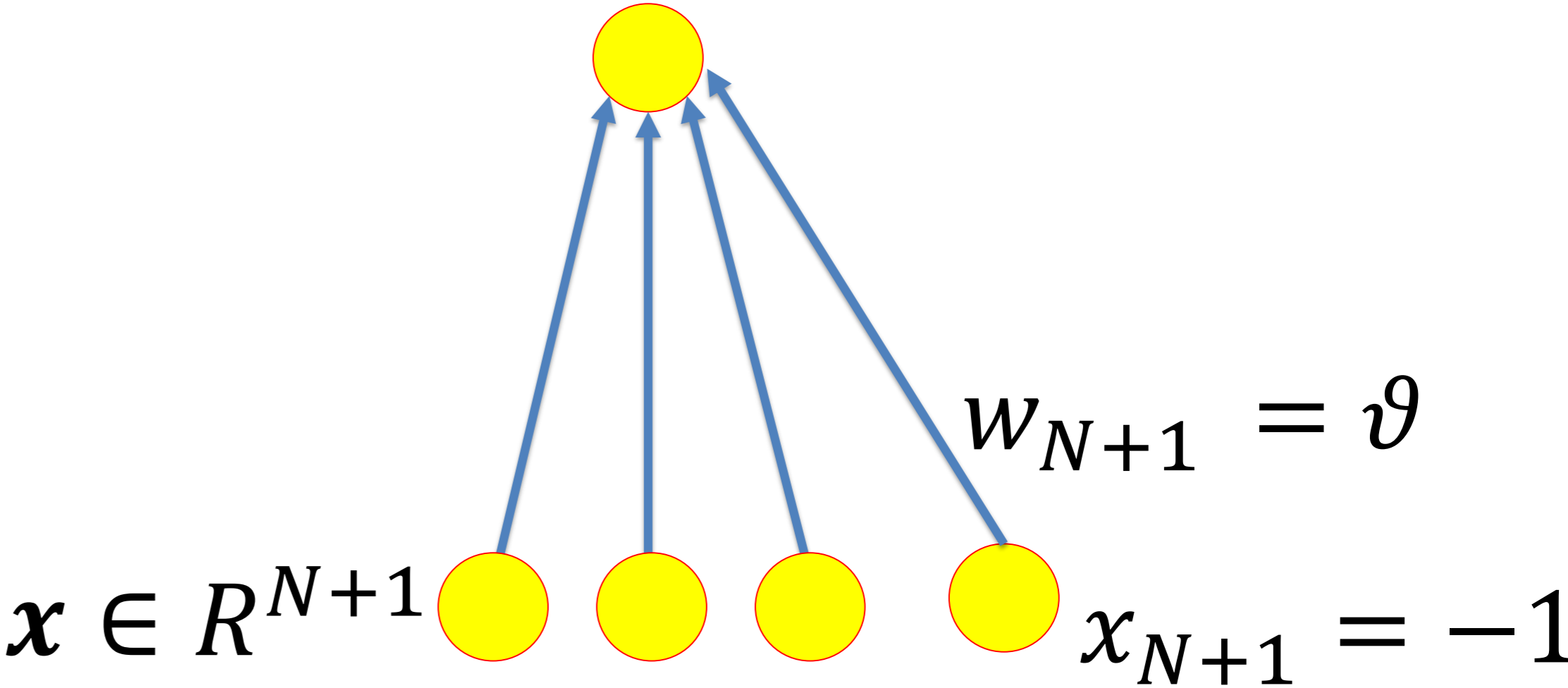


gradient descent

$$\Delta w_k = -\gamma \frac{dE}{dw_k}$$



$$\hat{y}^{\mu} = g(\mathbf{w}^T \mathbf{x}^{\mu})$$



Previous slide.

Since an error measure has to be always positive, we square the difference between actual output and target output.

The error function is this squared difference summed over all patterns in the data base. This error function is called the squared error or the quadratic error function. We will see other error functions in later weeks.

Most of the learning rules that we consider in this class are based on gradient descent:

The weight w_k is updated by an amount Δw_k proportional to the gradient $\frac{dE}{dw_k}$

The amplitude of the update is given by the learning rate gamma.

There is a negative sign because the aim is the REDUCE the error in each step.

With small enough learning rate, the gradient descent algorithm will end up close to a minimum of the error function. It jitters around the minimum because of the finite learning step gamma.

There is no reason that it should end up in a global minimum.

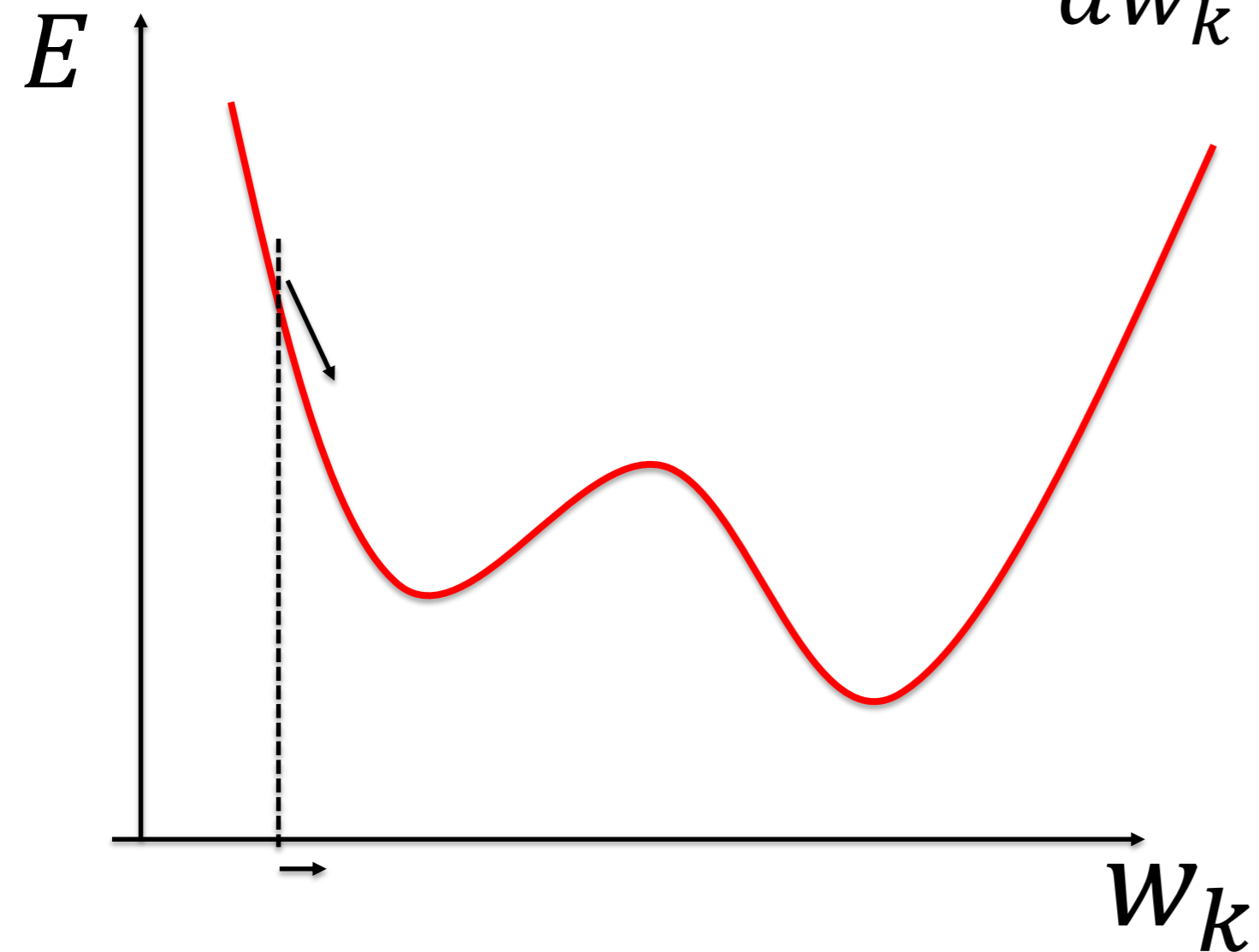
Gradient descent

Quadratic error

$$E(\mathbf{w}) = \frac{1}{2P} \sum_{\mu=1}^P [t^{\mu} - \hat{y}^{\mu}]^2$$

gradient descent

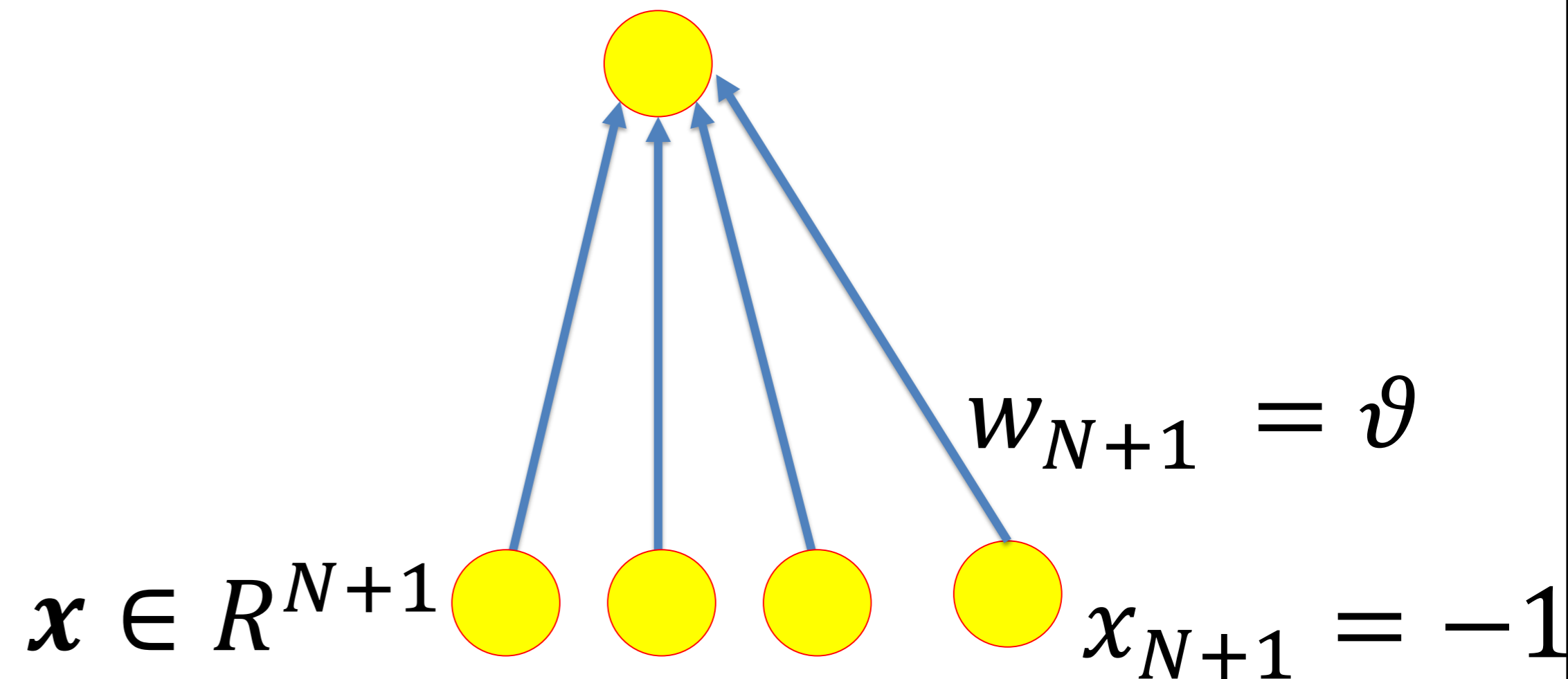
$$w_k = -\gamma \frac{dE}{dw_k}$$



Exercise 1.1 now:

- calculate gradient
- limit to one pattern
- geometric interpretation?

$$\hat{y}^{\mu} = g(\mathbf{w}^T \mathbf{x}^{\mu})$$



Artificial Neural Networks (Gerstner). Exercises for week 1

Week 1: Simple Perceptrons, Geometric interpretation, Discriminant function

1. Gradient of quadratic error function

We define the mean square error in a data base with P patterns as

$$E^{\text{MSE}}(\mathbf{w}) = \frac{1}{2} \frac{1}{P} \sum_{\mu} [t^{\mu} - \hat{y}^{\mu}]^2 \quad (1)$$

where the output is

$$\hat{y}^{\mu} = g(a^{\mu}) = g(\mathbf{w}^T \mathbf{x}^{\mu}) = g\left(\sum_k w_k x_k^{\mu}\right)$$

and the input is the pattern \mathbf{x}^{μ} with components $x_1^{\mu} \dots x_N^{\mu}$.

(a) Calculate the update of weight w_j by gradient descent (batch rule)

$$\Delta w_j = -\eta \frac{dE}{dw_j} \quad (3)$$

Hint: Apply chain rule

(b) Rewrite the formula by taking one pattern at a time (stochastic gradient descent). What is the difference to the batch rule? What is the geometric interpretation? Compare with the perceptron algorithm!

Pause video now

- calculate gradient

- apply only 1 pattern

- geometry/vector?

Exercise 1.1 now:

- calculate gradient
- apply only 1 pattern
- geometry/vector?

Lecture continues
in 10 minutes

Artificial Neural Networks (Gerstner). Exercises for week 1

Week 1: Simple Perceptrons, Geometric interpretation, Discriminant function

1. Gradient of quadratic error function

We define the mean square error in a data base with P patterns as

$$E^{\text{MSE}}(\mathbf{w}) = \frac{1}{2} \frac{1}{P} \sum_{\mu} [t^{\mu} - \hat{y}^{\mu}]^2 \quad (1)$$

where the output is

$$\hat{y}^{\mu} = g(a^{\mu}) = g(\mathbf{w}^T \mathbf{x}^{\mu}) = g\left(\sum_k w_k x_k^{\mu}\right) \quad (2)$$

and the input is the pattern \mathbf{x}^{μ} with components $x_1^{\mu} \dots x_N^{\mu}$.

(a) Calculate the update of weight w_j by gradient descent (batch rule)

$$\Delta w_j = -\eta \frac{dE}{dw_j} \quad (3)$$

Hint: Apply chain rule

(b) Rewrite the formula by taking one pattern at a time (stochastic gradient descent). What is the difference to the batch rule? What is the geometric interpretation? Compare with the perceptron algorithm!

Gradient descent calculation: 'batch' and 'online'

(Σ

18

Your notes.

(Σ

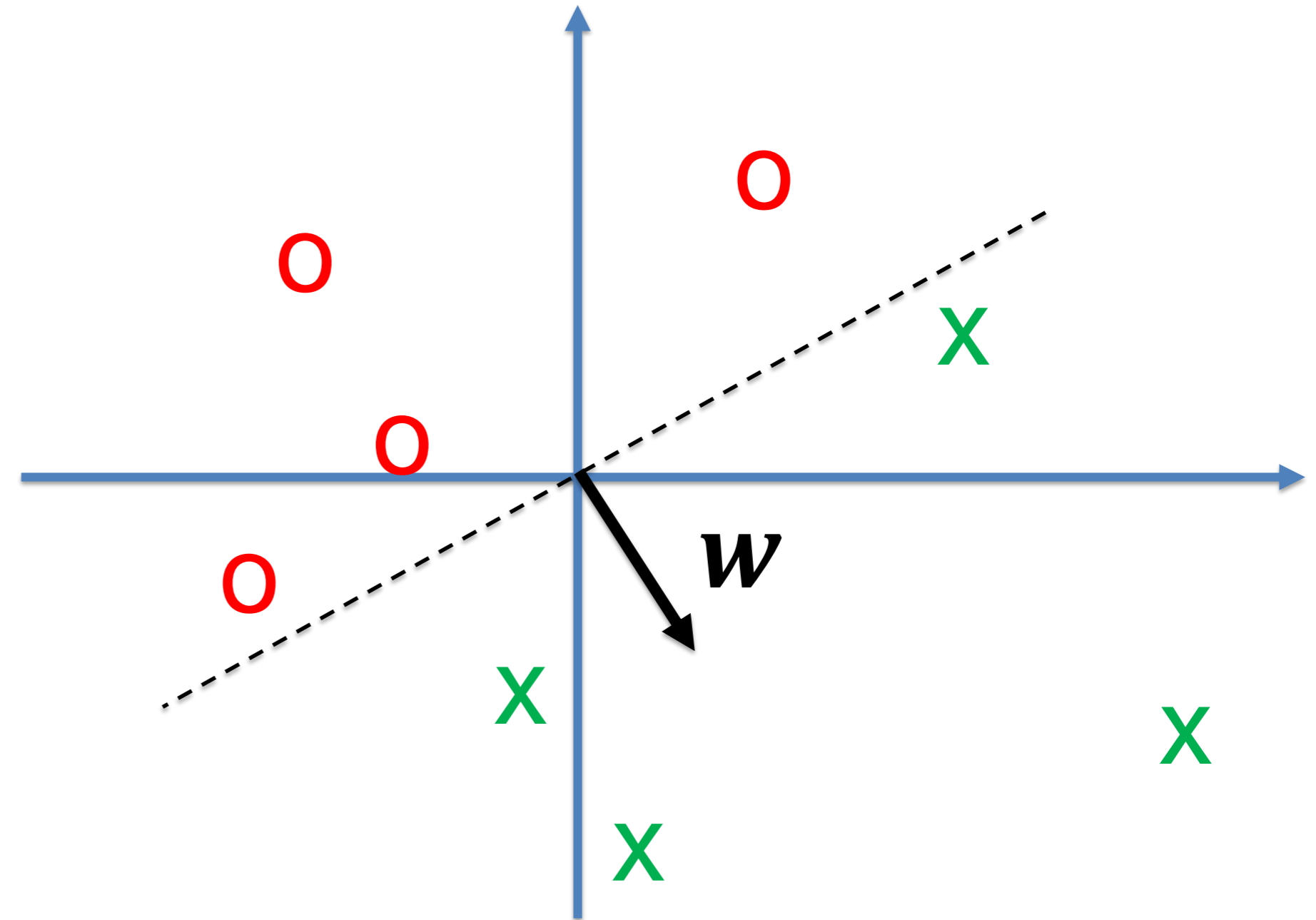
Gradient descent algorithm (stochastic gradient descent)

After presentation of pattern x^μ update the weight vector by

$$\Delta w = \gamma \delta(\mu) x^\mu$$

$$\delta(\mu) = [t^\mu - \hat{y}^\mu] g'$$

- amount of change depends on $\delta(\mu)$, prop. to the (signed) output mismatch for this data point
- change implemented even if 'correctly' classified
- change proportional to x^μ
- similar to perceptron algorithm



Previous slide.

Gradient descent can be done in two different modes:

Batch algorithm: we keep the sum over all patterns →

one update step after all patterns have been presented.

updates are repeated several times.

Online algorithm: one update step after each single pattern.

(patterns can be chosen stochastically or cyclically:

the online algo is also called stochastic gradient descent)

one 'epoch' = all patterns presented once.

repeated for many epochs until convergence

Structure of online algorithm similar to perceptron algorithm.

Main difference: the mismatch $\delta(\mu)$ is smooth here

Similar to perceptron, if a positive example is misclassified, the weight vector turns in direction of this input pattern.

The geometric picture is hence the same as for the Perceptron algorithm.

Stochastic gradient descent algorithm (for simple perceptron)

Gradient Descent: Simple Perceptron (in $N+1$ dimensions)

- set $\gamma = 0.01$ (learning rate; P patterns in total, index μ)
- choose M (number of epochs)

(1) For counter $k < P M$

- randomly choose pattern μ
- calculate output

$$\hat{y}^{\mu} = g(\mathbf{w}^T \mathbf{x}^{\mu})$$

- update by

$$\Delta \mathbf{w} = \gamma [t^{\mu} - \hat{y}^{\mu}] g' \mathbf{x}^{\mu}$$

- increase counter $k \leftarrow k+1$

(2a) stop if change during last P patterns was acceptably small

(2b) else, decrease γ , reset k to $k=1$ and return to (1)

Previous slide.

This is a sketch of the resulting stochastic gradient descent algorithm, i.e., an online algorithm: one update step after each single pattern.

(patterns are chosen stochastically here)

one 'epoch' = all patterns presented once (on average!)

The update steps are repeated for (several times) M epochs until convergence.

The amount of jittering can be reduced by lowering the value of the learning rate γ after M epochs.

Learning outcome and conclusions for today:

- understand classification as a geometrical problem
- discriminant function of classification
- linear versus nonlinear discriminant function
- linearly separable problems
- perceptron algorithm
- gradient descent for simple perceptrons
- understand learning as a geometric problem

Previous slide.

- Classification is equivalent to finding a separating surface in the high-dimensional input space
- This surface can be defined by the condition $d(x)=0$ where d is the discriminant function
- A generic data base for supervised learning requires a nonlinear discriminant function
- A simple perceptron can only implement a linear discriminant function: the separating hyperplane
- The perceptron algorithm turns the separating hyperplane in $N+1$ dimensions
- A quadratic error function gives rise to a stochastic gradient descent algorithm
- Geometrically the stochastic gradient descent algorithm also turns the hyperplane in $N+1$ dimensions, very similar to the perceptron algorithm

Reading for this week:

Bishop, Ch. 4.1.7 of

Pattern recognition and Machine Learning

or

Bishop, Ch. 3.1-3.5 of

Neural networks for pattern recognition

Motivational background reading:

Silver et al. 2017, Archive

*Mastering Chess and Shogi by Self-Play with a
General Reinforcement Learning Algorithm*

Goodfellow et al., Ch. 1 of

Deep Learning



Previous slide.

The suggested reading is important, in particular if you are not able to attend the class in a given week.

In all the following weeks, the suggested reading will always be listed on slide 2, at the beginning of the lecture, so that it is easy to find.