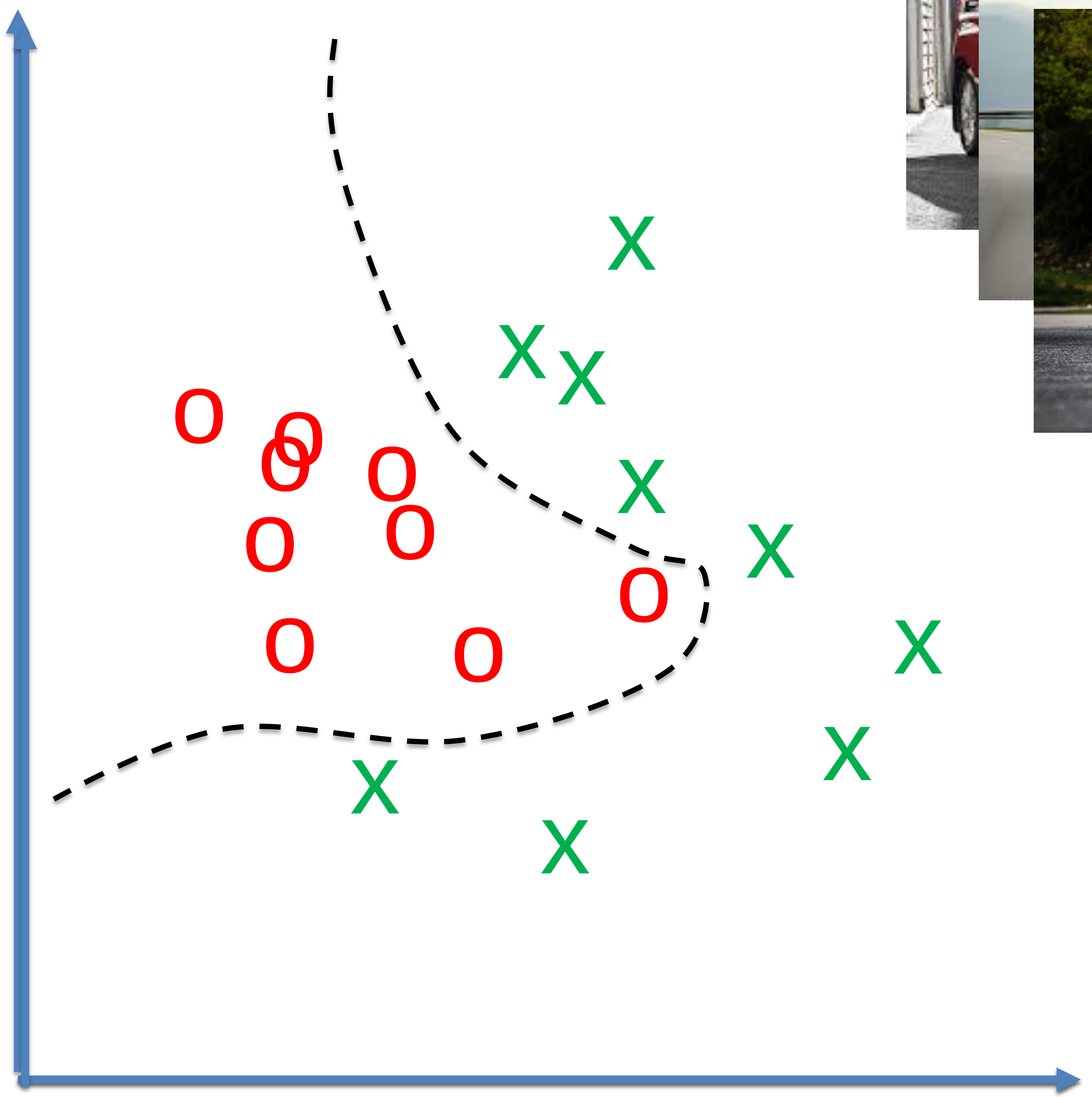
Artificial Neural Networks: Lecture 2
Backprop and multilayer perceptrons

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Objectives for today:

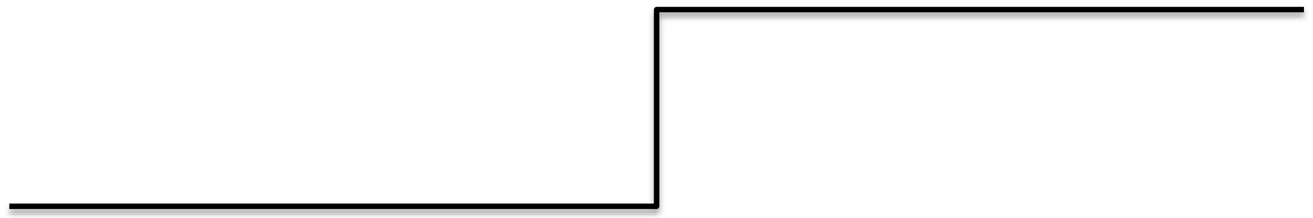
- XOR problem and the need for multiple layers
- understand backprop as a smart algorithmic implementation of the chain rule
- hidden neurons add flexibility, but flexibility is not always good: the problem of generalization
- training base and validation base: the need to predict well for future data

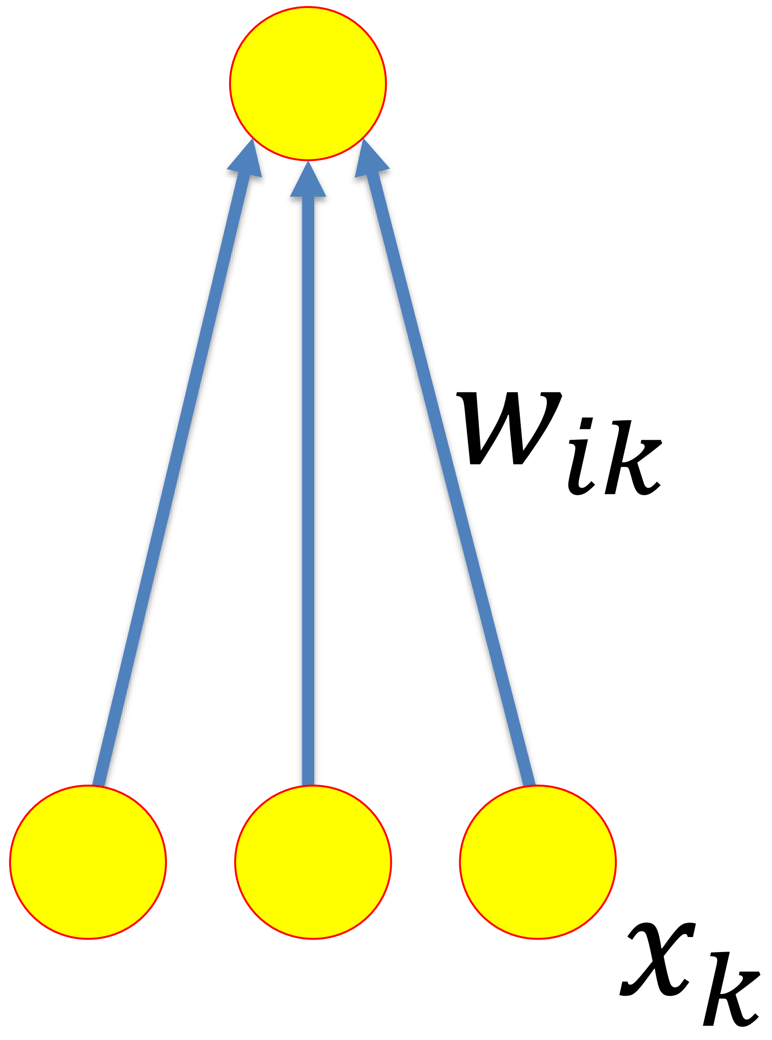
Review: Classification as a geometric problem



Review: Single-Layer networks: simple perceptron

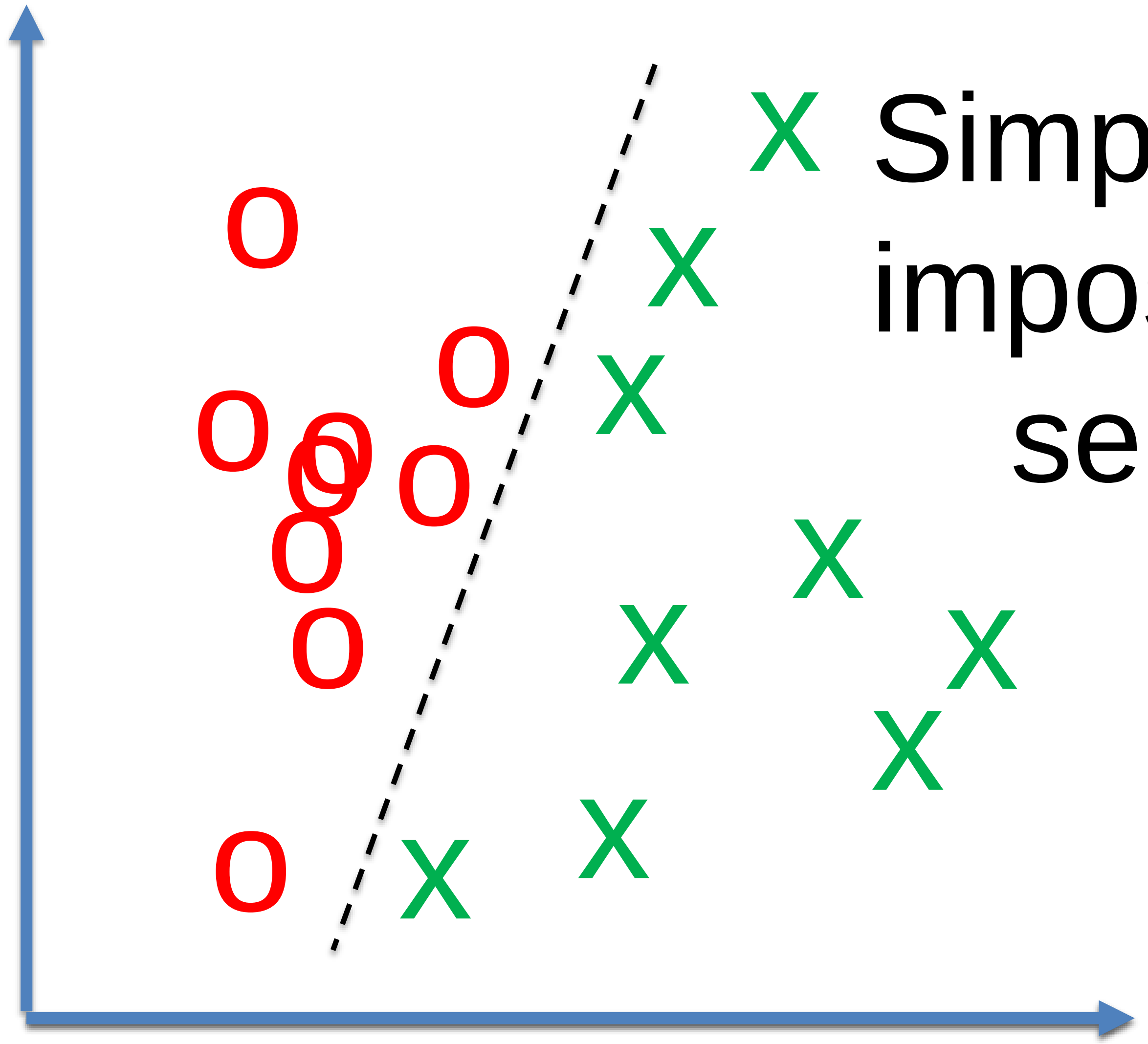
$$\hat{y} = 0.5[1 + sgn(\sum_k w_k x_k - \vartheta)]$$





vector $\boldsymbol{x}$

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



Simple perceptron imposes a linear separation

Artificial Neural Networks: Lecture 2
Backprop and multilayer perceptrons

Wulfram Gerstner
EPFL, Lausanne, Switzerland

1. Modern Gradient Descent Methods

Review: gradient descent

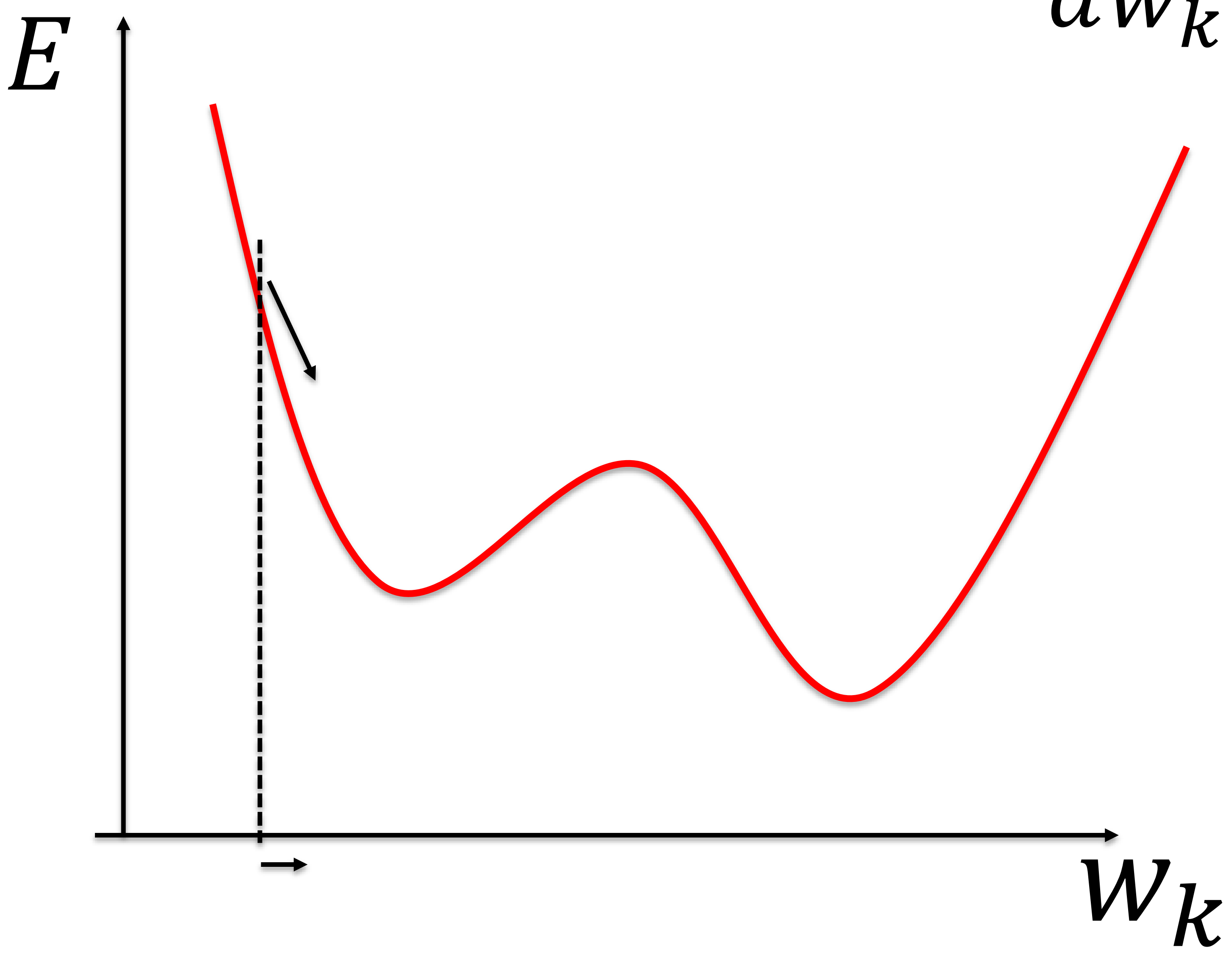
Batch rule:
one update after all patterns
(normal gradient descent)
Online rule:
one update after one pattern
(stochastic gradient descent)

Last week: Quadratic **error**

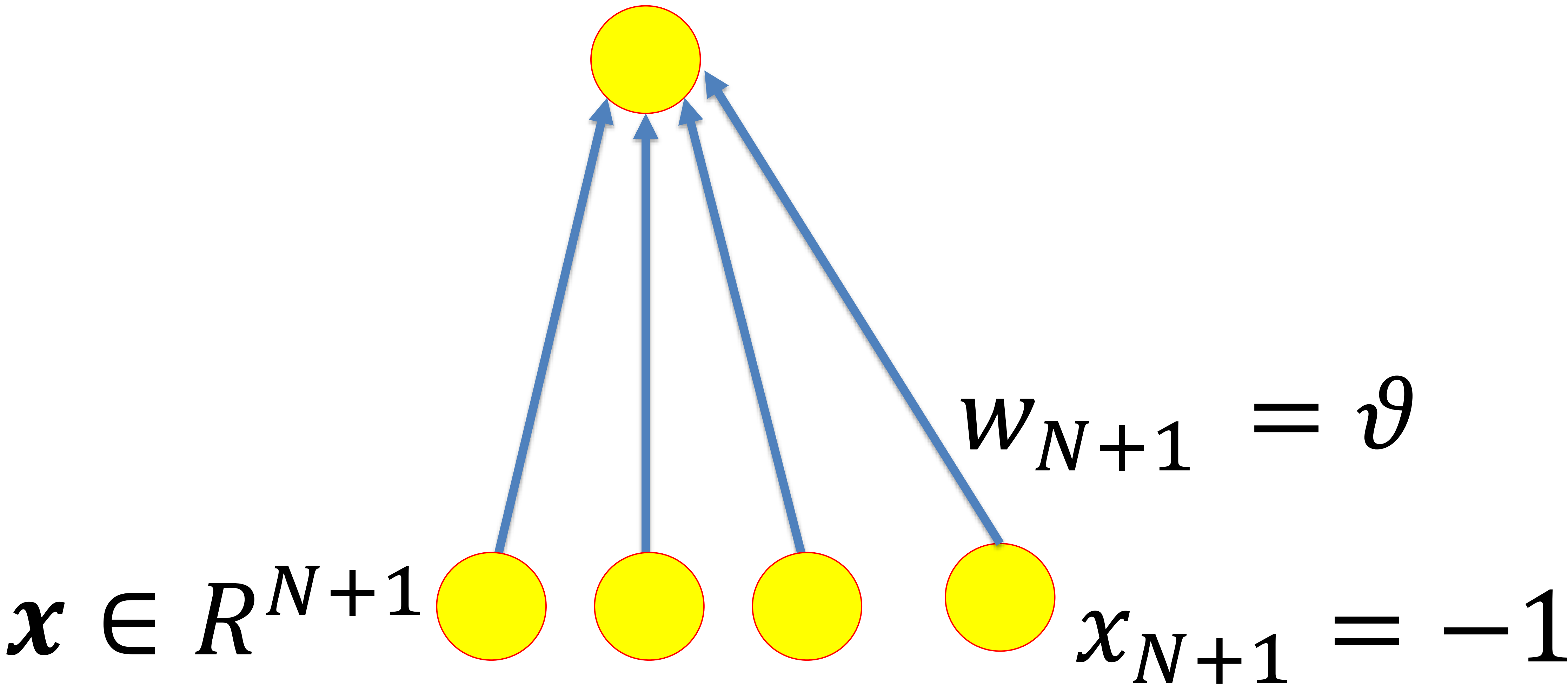
$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{P}\left[t^{\mu} - \hat{y}^{\mu}\right]^2$$

gradient descent

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



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



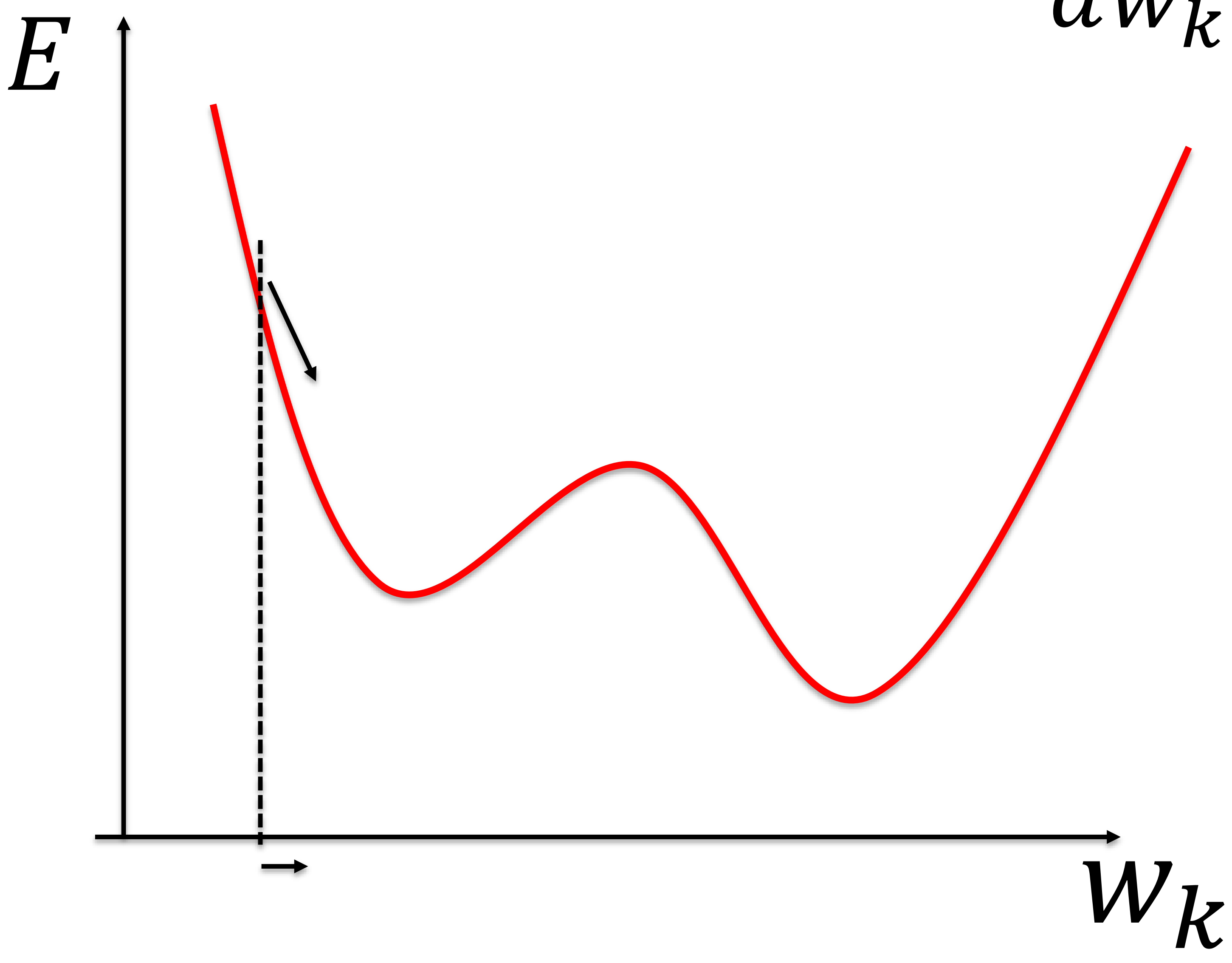
Modern gradient descent

Take some **error function,**
also called **loss function**

$$E(\mathbf{w})$$

gradient descent

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



Batch rule:
one update after all ***P* patterns**
(normal gradient descent)

Online rule:
one update after **one pattern**
(stochastic gradient descent)

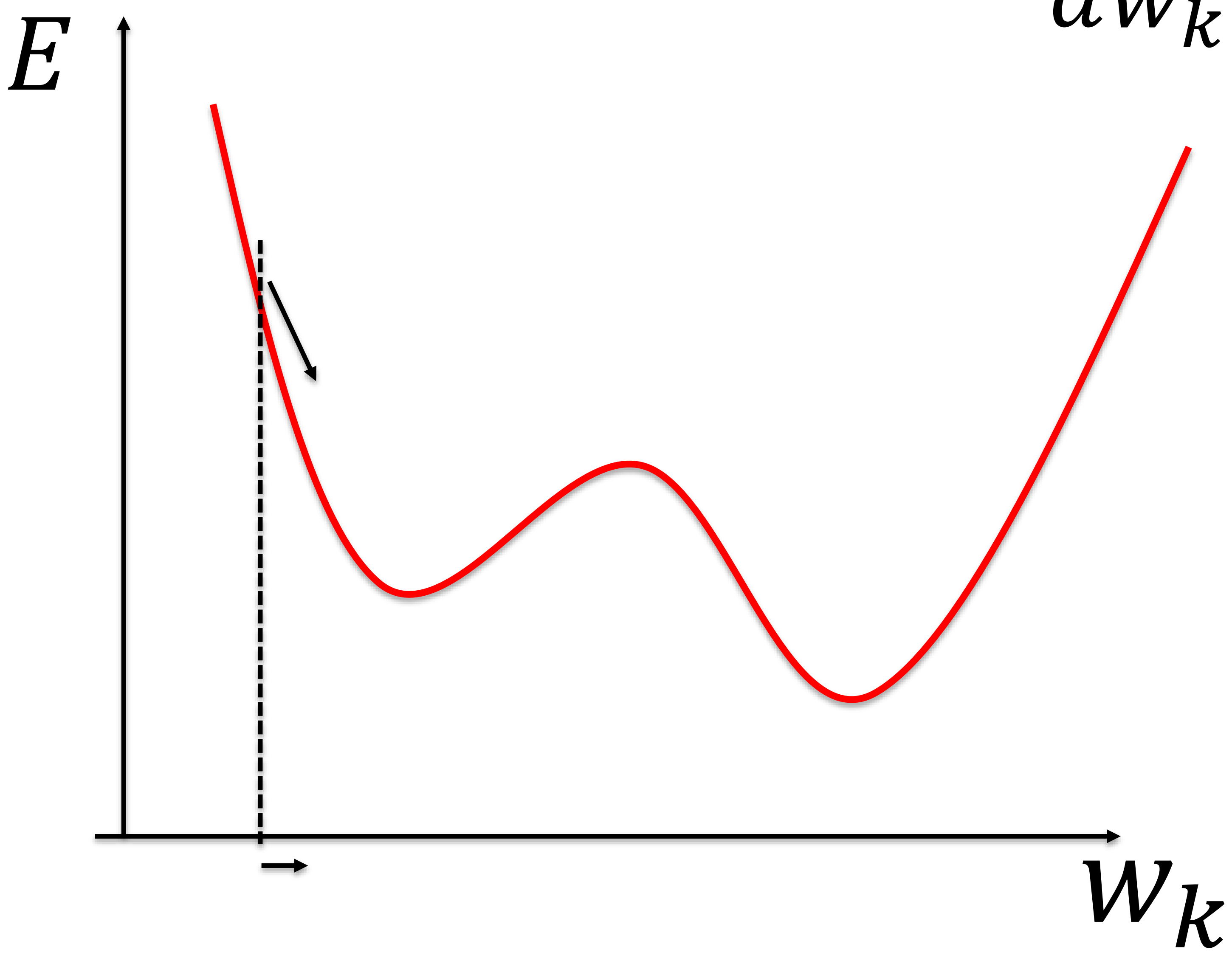
Mini Batch rule:
one update after ***P'=P/K* patterns**
(minibatch update)

1 epoch = all patterns applied once.
Training over many epochs

Properties of gradient descent

gradient descent

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



Convergence

- To local minimum
- No guarantee to find global minimum
- Learning rate needs to be sufficiently small
- Learning rate can be further decreased once you are close to convergence

→ See course: *Machine Learning* (Jaggi-Urbanke)

Artificial Neural Networks: Lecture 2

Backprop and multilayer perceptrons

Wulfram Gerstner

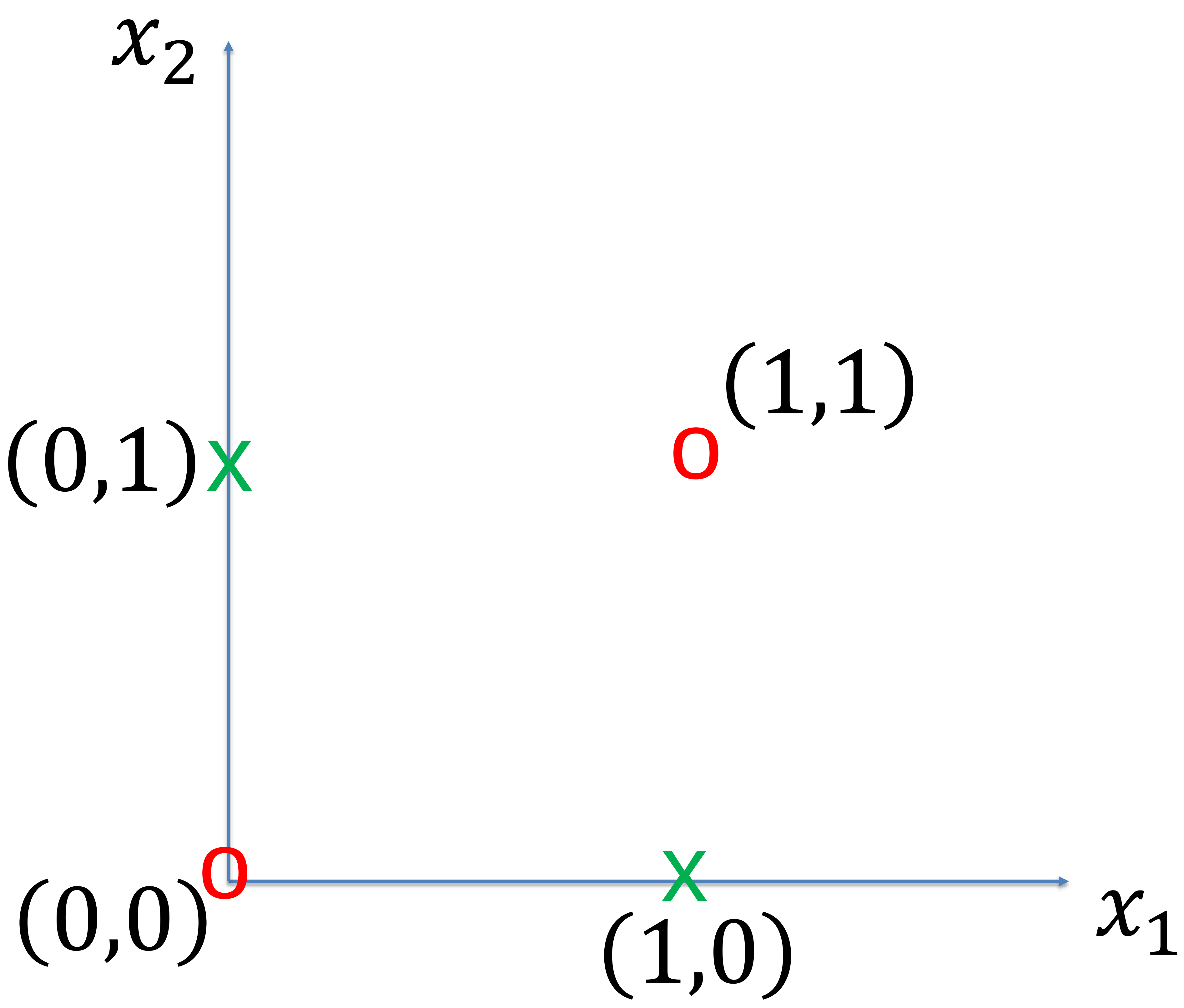
EPFL, Lausanne, Switzerland

1. Modern Gradient Descent Methods
2. **XOR problem**

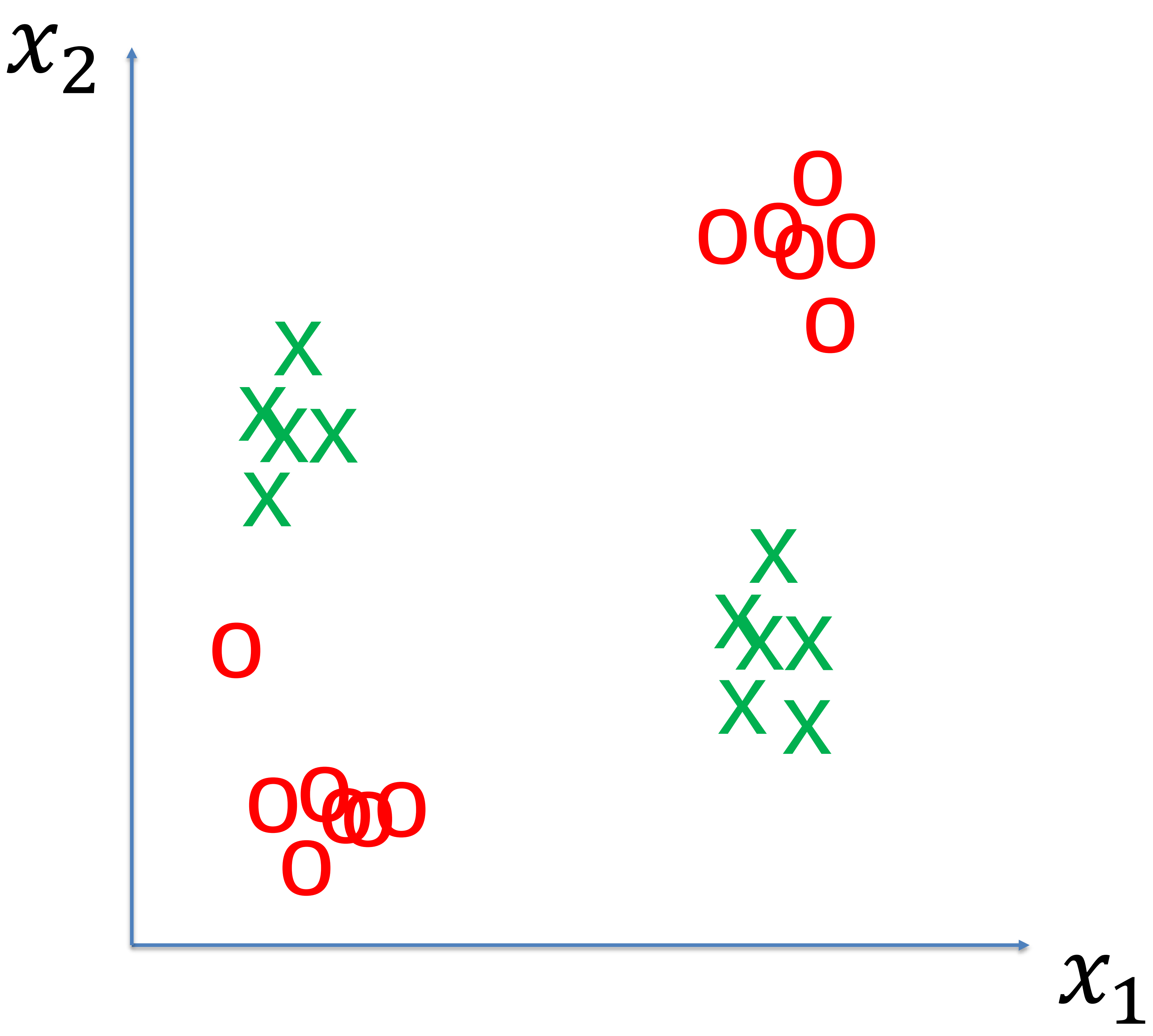
2. The XOR problem

Blackboard 1: solution of XOR

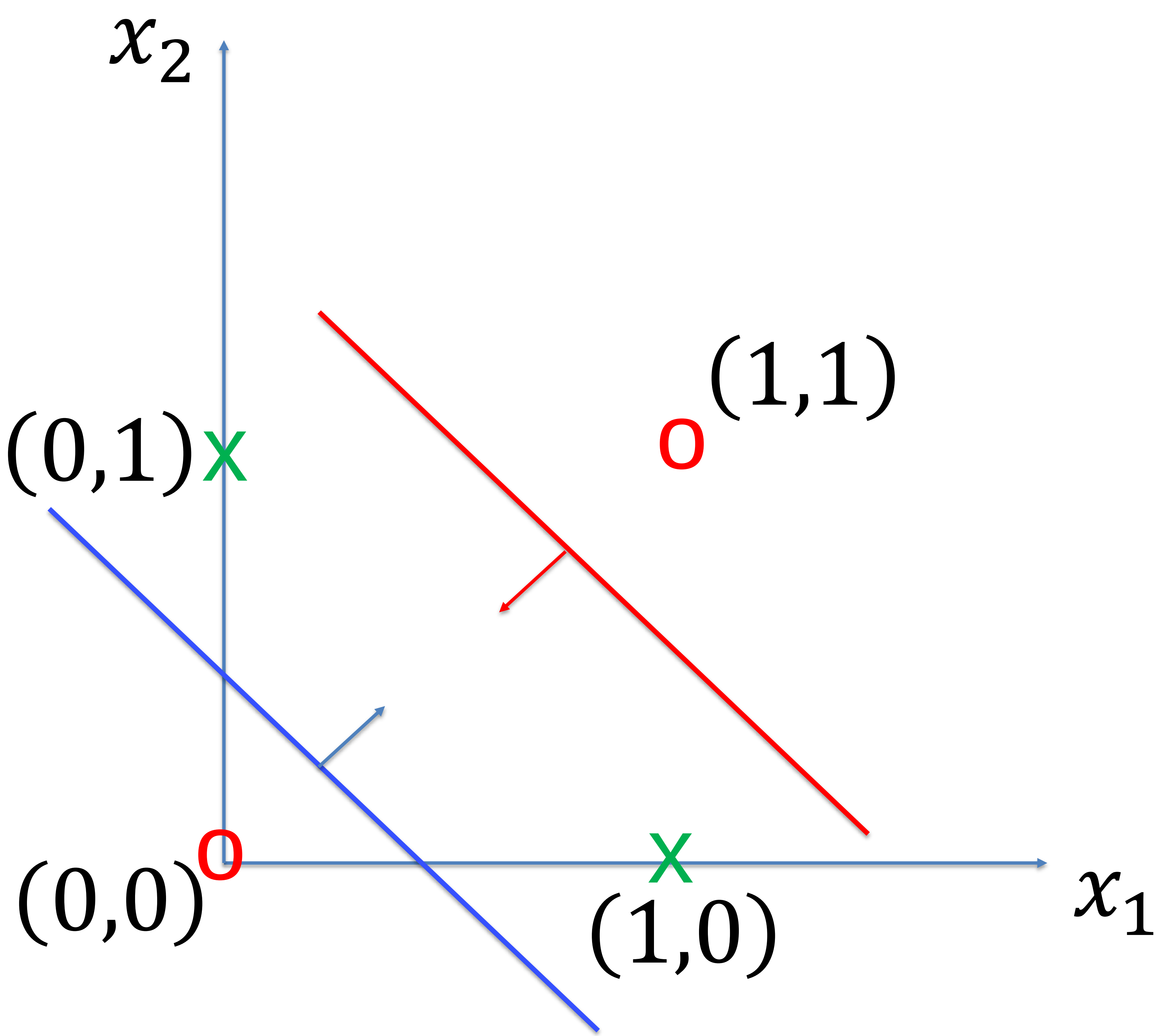
just 4 data points

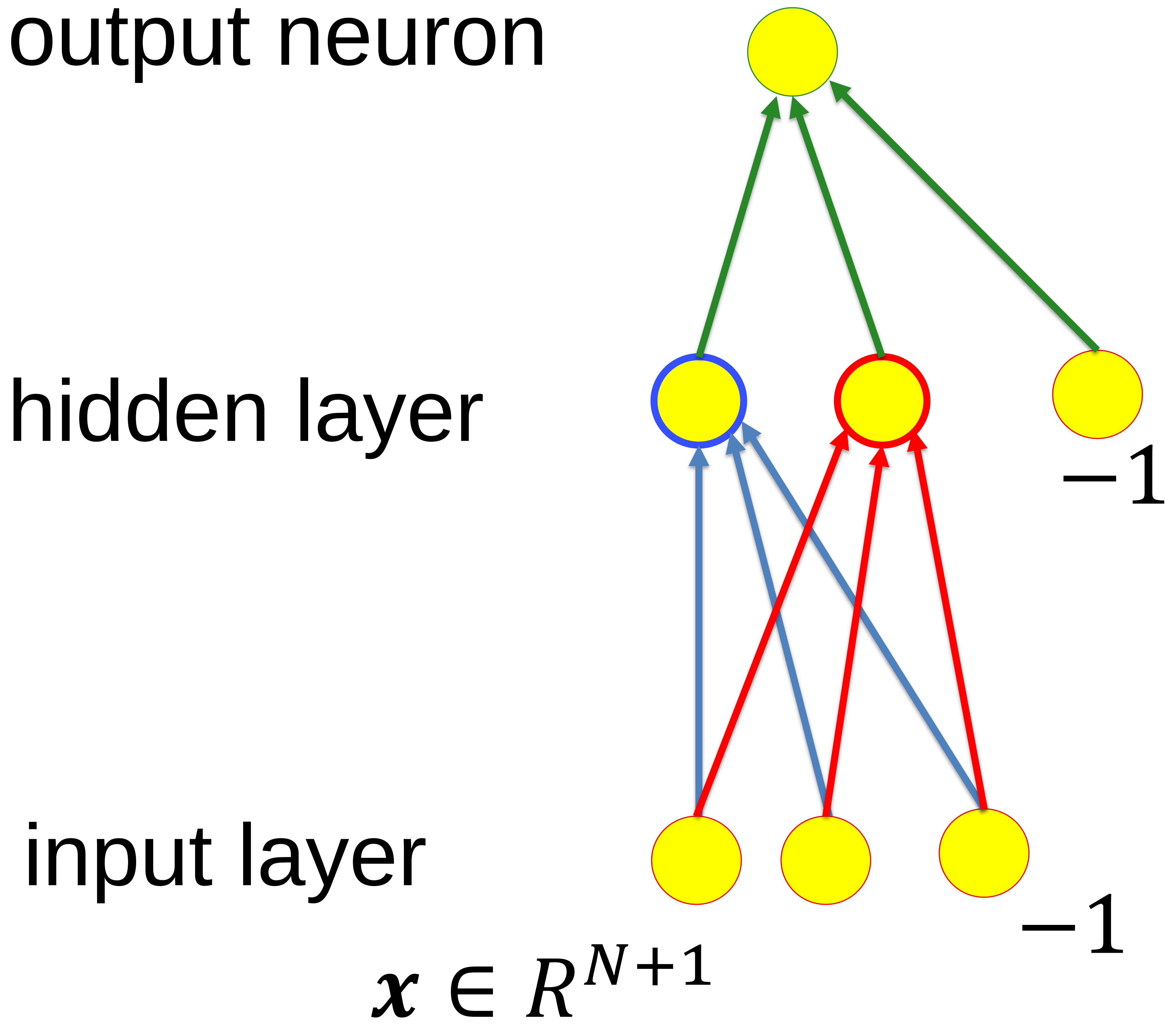


(or many)



2. Solution of XOR problem

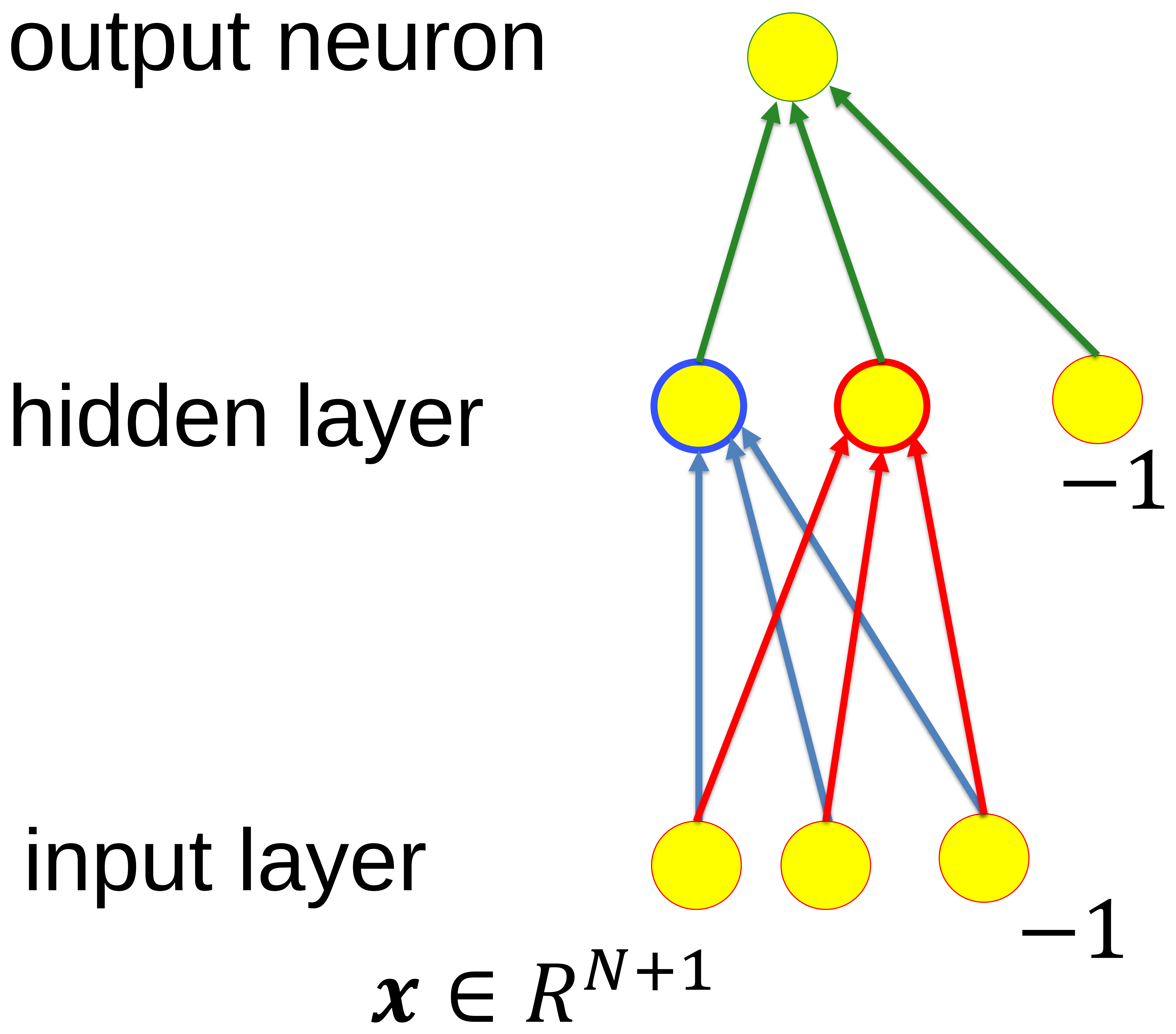




3. Multi-layer perceptron

- OK, can solve the XOR problem (by construction)

- But is there an **algorithm to find the weights** in more complicated cases?



Artificial Neural Networks: Lecture 2
Backprop and multilayer perceptrons

Wulfram Gerstner
EPFL, Lausanne, Switzerland

1. Modern Gradient Descent Methods
2. XOR problem
3. **Multilayer Perceptron**

3. Supervised learning with sigmoidal output

error!

target output $t^7 = 1$ ⟷ $\hat{y}^7 = 0.2$ classifier output

output

$f(x^7)$

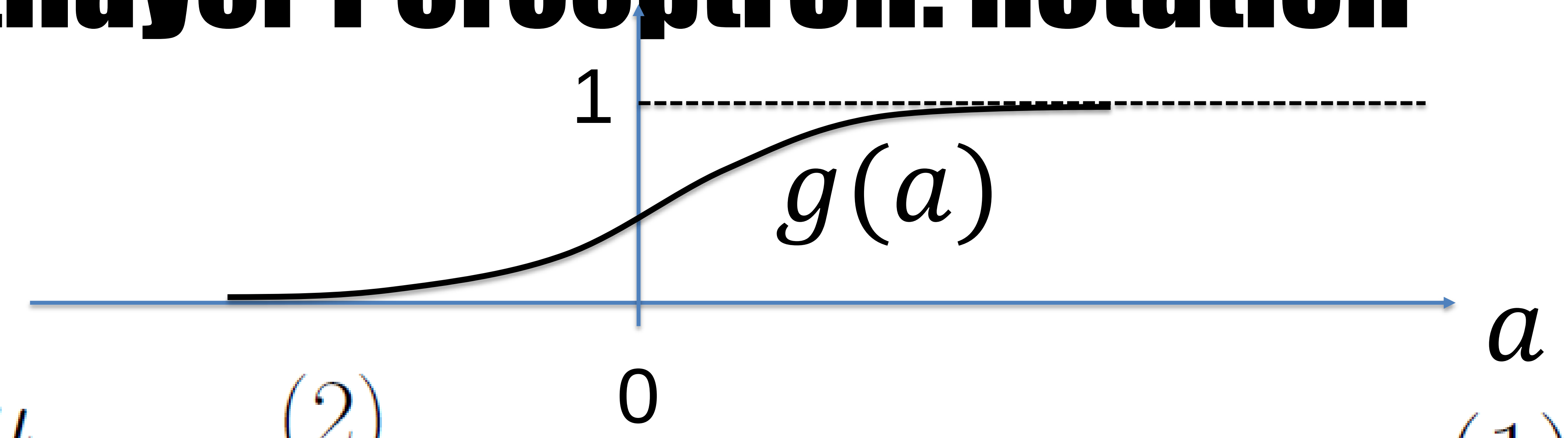
Classifier

teacher

input

x^7

3. Multilayer Perceptron: notation



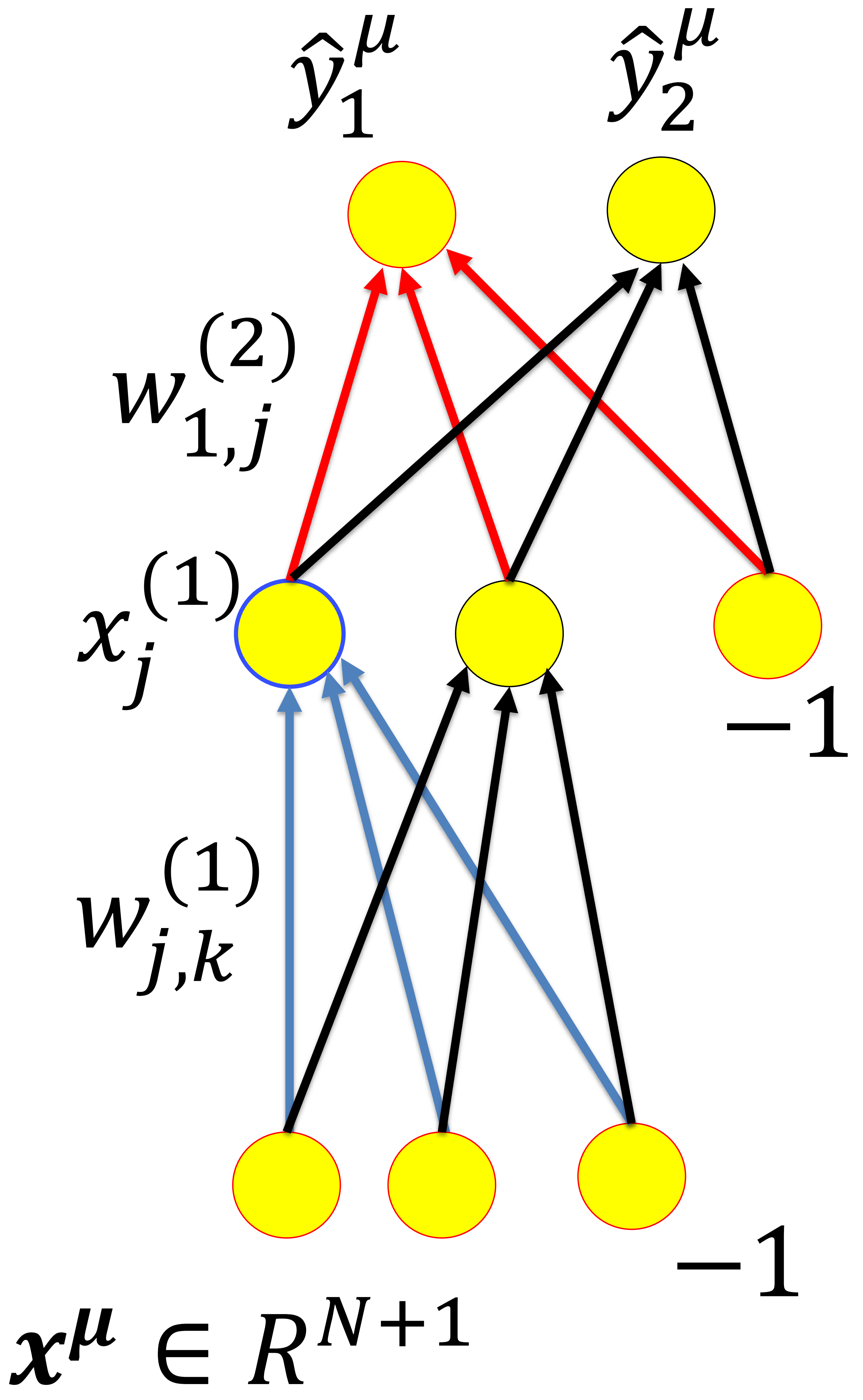
$$\hat{y}_i^{\mu} = x_i^{(2)} \quad (1)$$

$$= g^{(2)}[a_i^{(2)}] \quad (2)$$

$$= g^{(2)}[\sum_j w_{ij}^{(2)} x_j^{(1)}] \quad (3)$$

$$= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(a_j^{(1)})] \quad (4)$$

$$= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(\sum_k w_{jk}^{(1)} x_k^{\mu})] \quad (5)$$



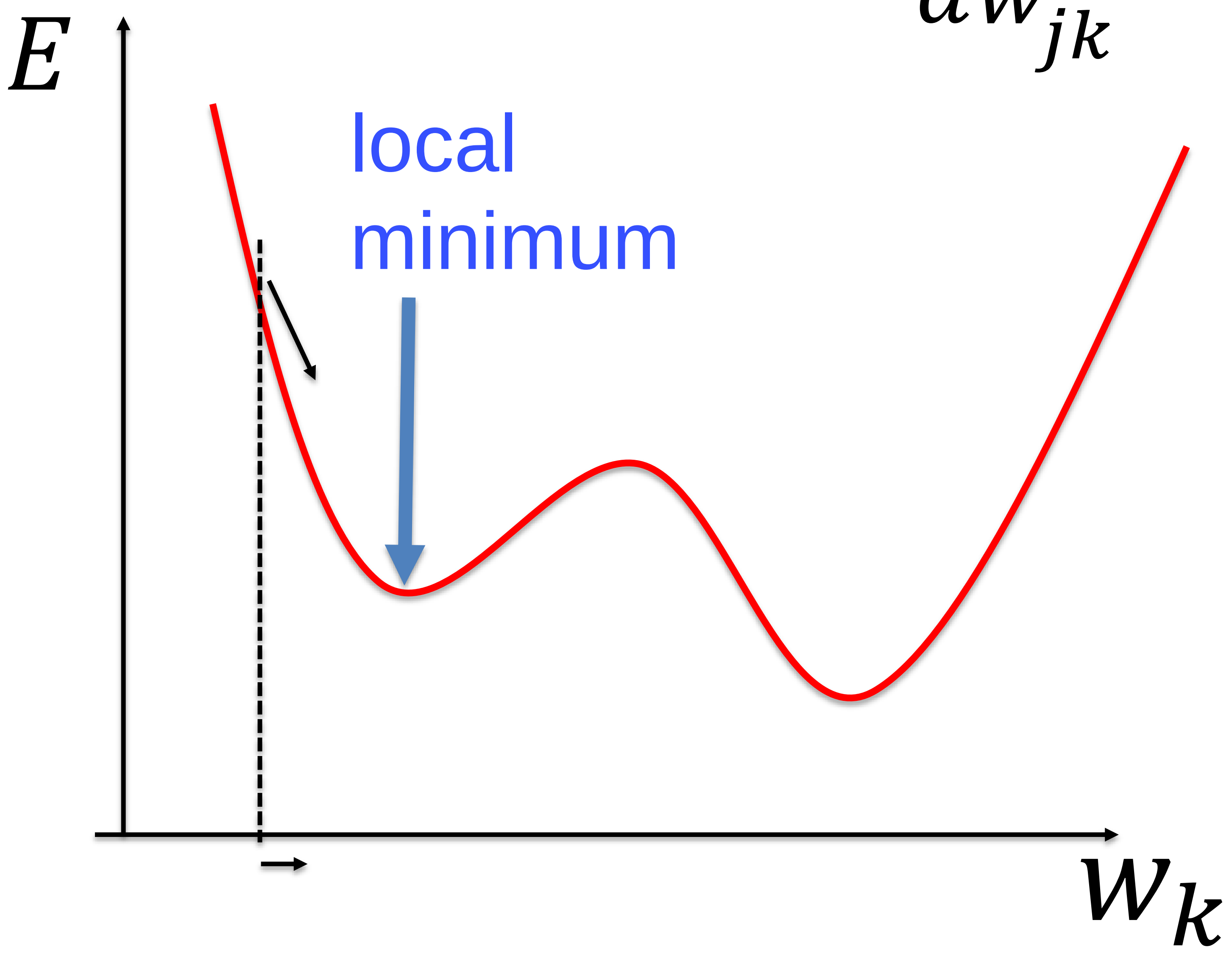
3. Multilayer Perceptron: gradient descent

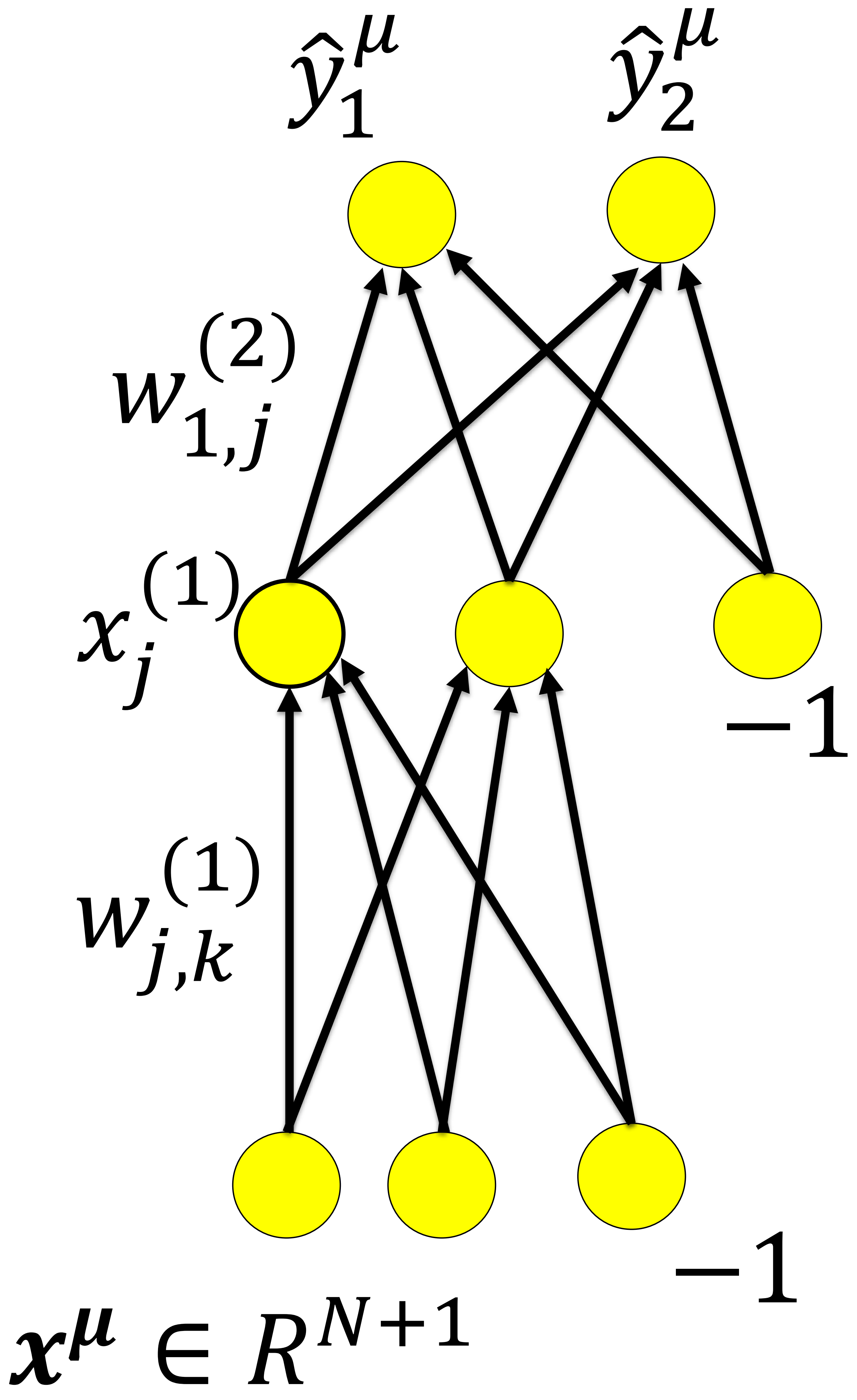
Quadratic **error**

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{P}\sum_{i}[t_i^{\mu} - \hat{y}_i^{\mu}]^2$$

gradient descent

$$\Delta w_{jk}^{(1)} = -\gamma \frac{dE}{dw_{jk}^{(1)}}$$





Exercise 1 now: Calculate gradient!
Use Chain rule, be smart!

We continue in **8 minutes!**

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

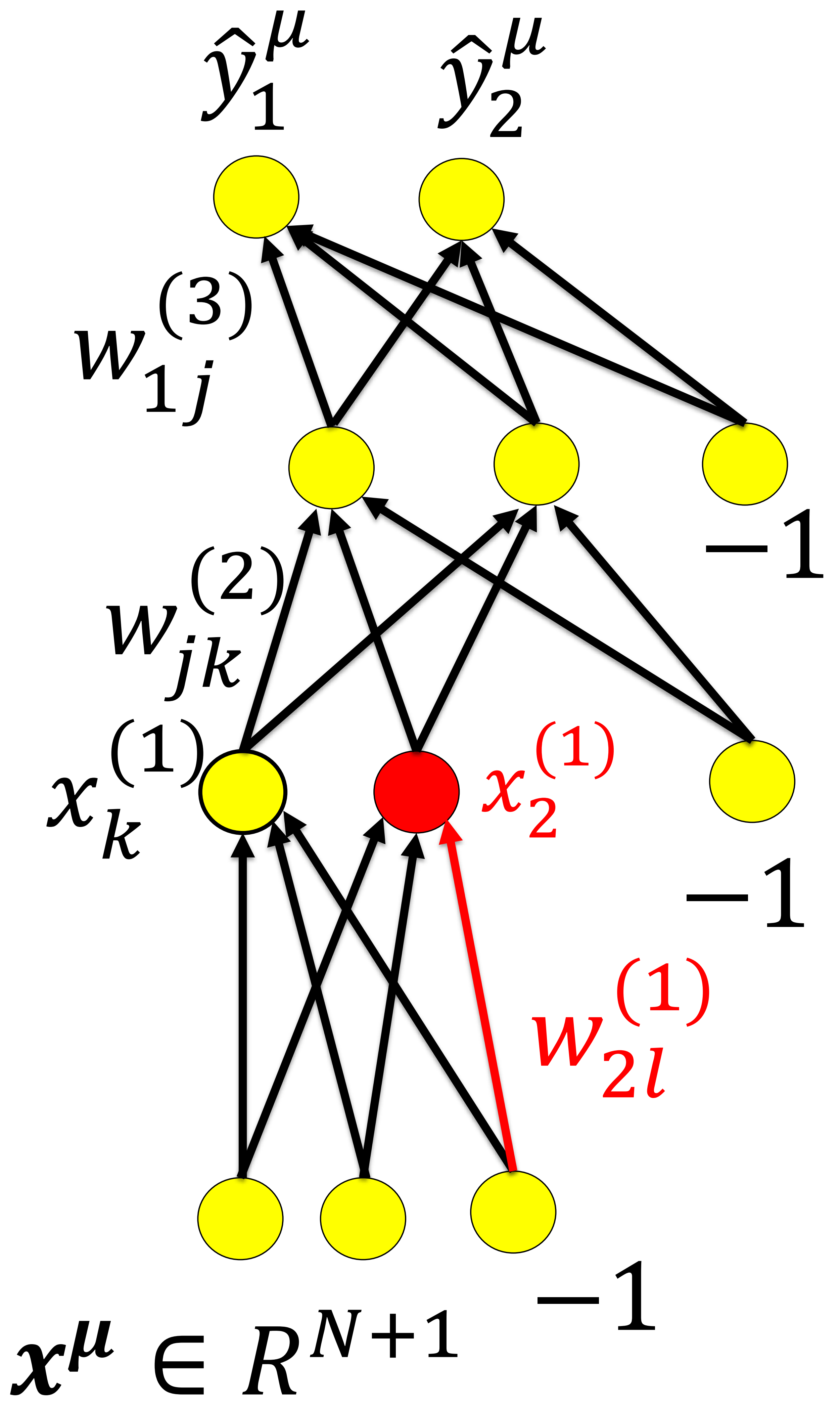
$$\Delta w_{jk}^{(1)} = -\gamma \frac{dE}{dw_{jk}^{(1)}}$$

with

$$\hat{y}_i^\mu = g^{(3)}\Big(\sum_j w_{ij}^{(3)}\, x_j^{(2)}\Big)$$

$$x_j^{(2)} = g^{(2)}\Big(\sum_k w_{jk}^{(2)}\, x_k^{(1)}\Big)$$

$$x_2^{(1)} = g^{(1)}(a_2^{(1)}) = g^{(1)}(\textstyle\sum_l w_{2l}^{(1)}\, x_l^{(0)})$$



3. Multilayer Perceptron: gradient descent

Calculating a gradient in multi-layer networks:

- write down chain rule
- analyze dependency graph
- store intermediate results
- update intermediate results
  while proceeding through graph
- update all weights together at the end

compare with
dynamic programming

Artificial Neural Networks: Lecture 2

Backprop and multilayer perceptrons

Wulfram Gerstner

EPFL, Lausanne, Switzerland

1. Modern Gradient Descent Methods
2. XOR problem
3. Multilayer Perceptron
4. **BackProp Algorithm**

0. Initialization of weights

BackProp

1. Choose pattern $\mathbf{x}^\mu$

input $x_k^{(0)} = x_k^\mu$

2. Forward propagation of signals $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \quad (1)$$

output $\hat{y}_i^\mu = x_i^{(n_{\max})}$

3. Computation of errors in output

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})}) \ [\hat{y}_i^\mu - t_i^\mu] \quad (2)$$

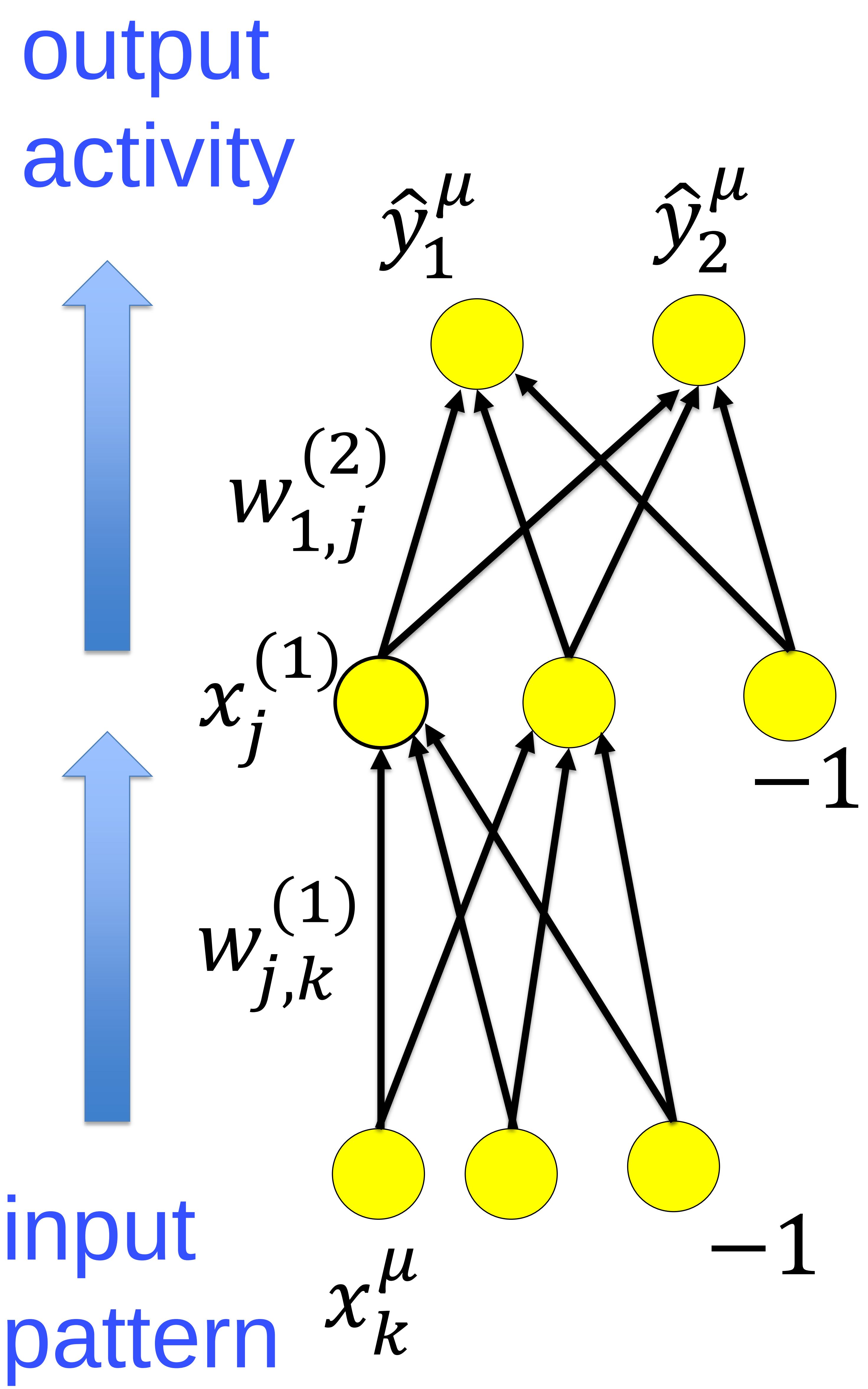
4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij} \, \delta_i^{(n)} \quad (3)$$

5. Update weights (for each (i, j) and all layers (n))

$$\Delta w_{ij}^{(n)} = -\gamma \ \delta_i^{(n)} \, x_j^{(n-1)} \quad (4)$$

6. Return to step 1.



BackProp

0. Initialization of weights

1. Choose pattern $\mathbf{x}^\mu$

$$\text{input } x_k^{(0)} = x_k^\mu$$

2. Forward propagation of signals $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \tag{1}$$

$$\text{output } \hat{y}_i^\mu = x_i^{(n_{\max})}$$

3. Computation of errors in output

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})}) \; [\hat{y}_i^\mu - t_i^\mu] \tag{2}$$

4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

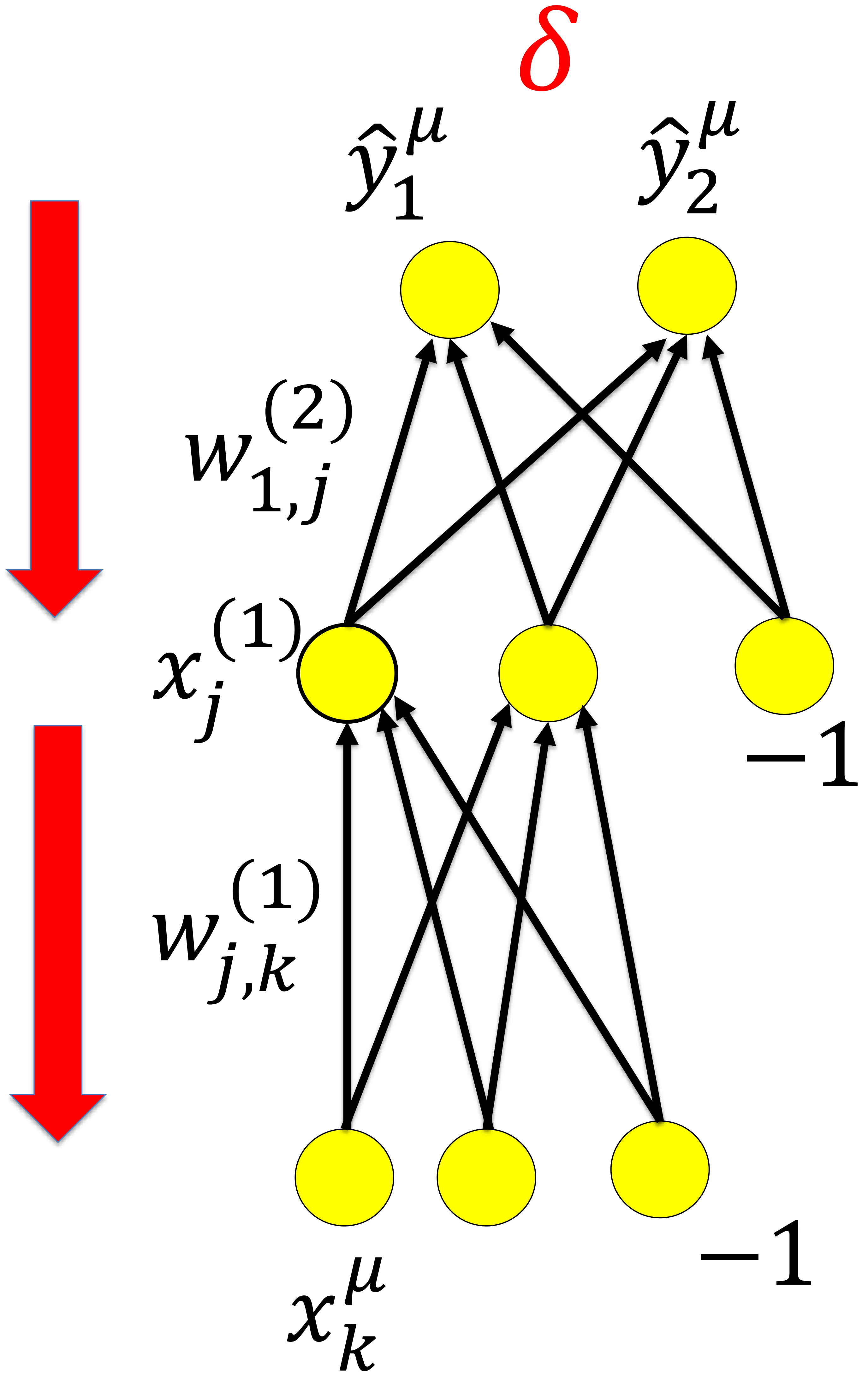
$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij} \, \delta_i^{(n)} \tag{3}$$

5. Update weights (for each (i,j) and all layers (n))

$$\Delta w_{ij}^{(n)} = -\gamma \;\; \delta_i^{(n)} \, x_j^{(n-1)} \tag{4}$$

6. Return to step 1.

Calculate output error



BackProp

0. Initialization of weights
1. Choose pattern $\mathbf{x}^\mu$

   input $x_k^{(0)} = x_k^\mu$

2. Forward propagation of signals $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \tag{1}$$

   output $\hat{y}_i^\mu = x_i^{(n_{\max})}$

3. Computation of errors in output

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})}) \; [\hat{y}_i^\mu - t_i^\mu] \tag{2}$$

4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

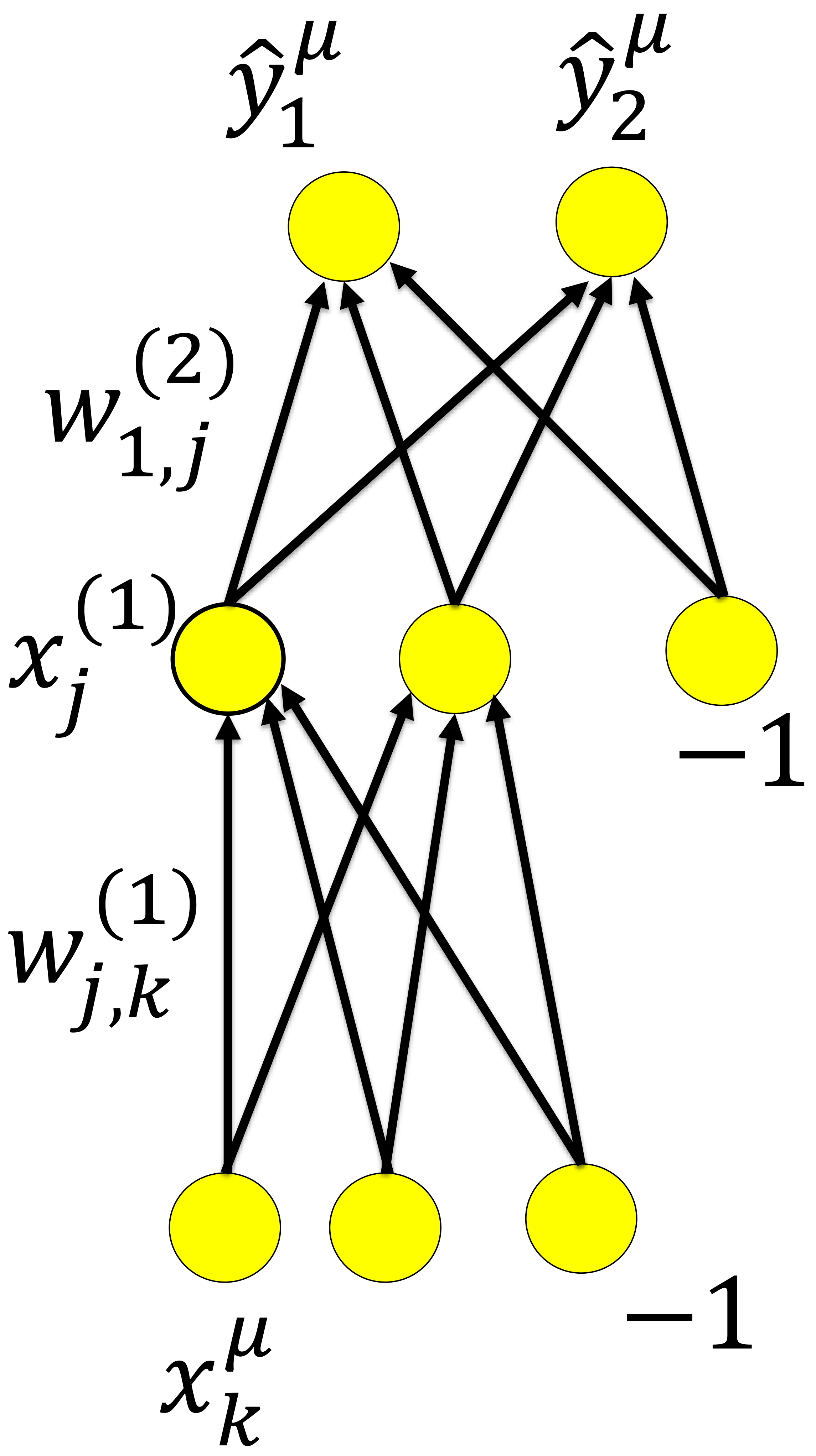
$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \tag{3}$$

5. Update weights (for each (i,j) and all layers (n))

$$\Delta w_{ij}^{(n)} = -\gamma \;\; \delta_i^{(n)} \, x_j^{(n-1)} \tag{4}$$

6. Return to step 1.

update **all** weights



4. Conclusions: Multilayer Perceptron and Backprop

- A multilayer Perceptron can solve the XOR problem
- Hidden neurons increase the flexibility of the separating surface
- Weigths are the parameters of the separating surface
- Weights can be adapted by gradient descent
- Backprop is an implementation of gradient descent
- Gradient descent converges to a local minimum

→ **Big Multilayer perceptrons are flexible and can be trained by BackProp to minimize classification error**

4. Backprop: Quiz

Your friend claims the following; do you agree?

[ ] BackProp is nothing else than the chain rule, handled well.

[ ] BackProp is just **numerical** differentiation

[ ] BackProp is a special case of automatic **algorithmic** differentiation

[ ] BackProp is an order of magnitude faster than numerical differentiation

Artificial Neural Networks: Lecture 2
Backprop and multilayer perceptrons

Wulfram Gerstner
EPFL, Lausanne, Switzerland

1. Modern Gradient Descent Methods
2. XOR problem
3. Multilayer Perceptron
4. BackProp Algorithm
5. **The problem of overfitting**

5. The problem of overfitting

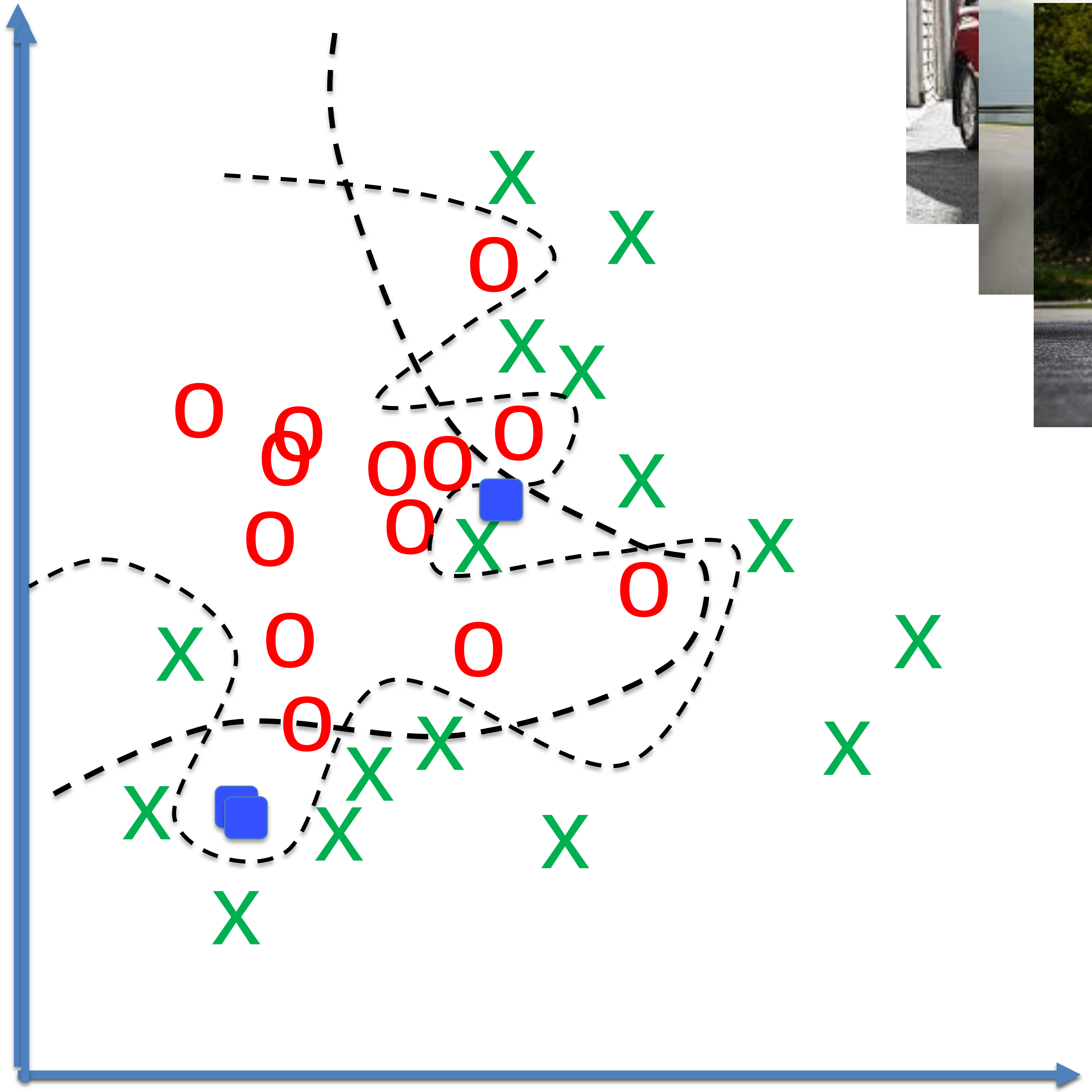
Big Multilayer perceptrons are flexible and can be trained by BackProp to minimize classification error

… but is flexibility always good?

5. Classification of new inputs

X = ‘car’

o = ‘not car’



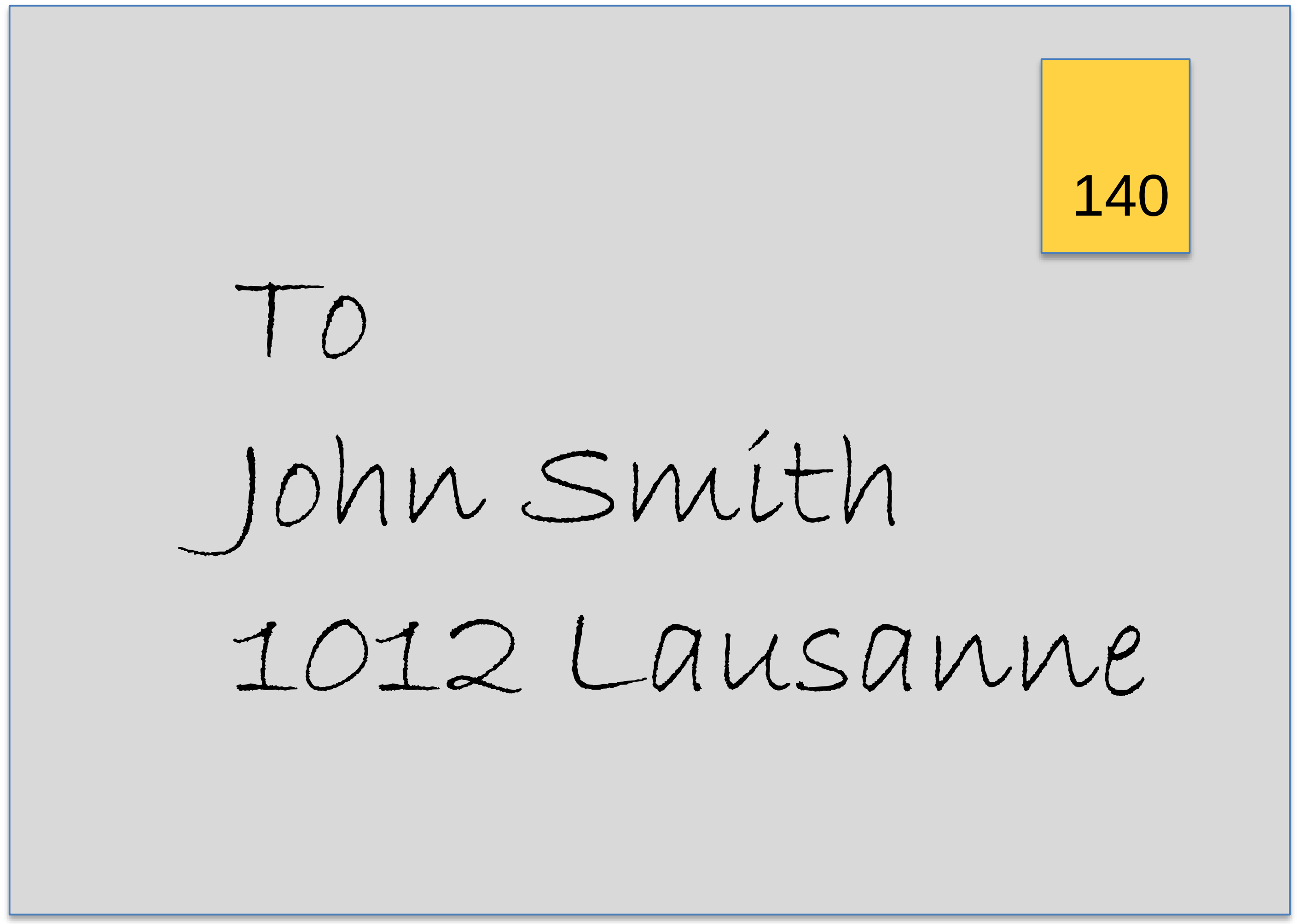
more data to learn from

Aim: predict classification for **new** inputs, not seen during training

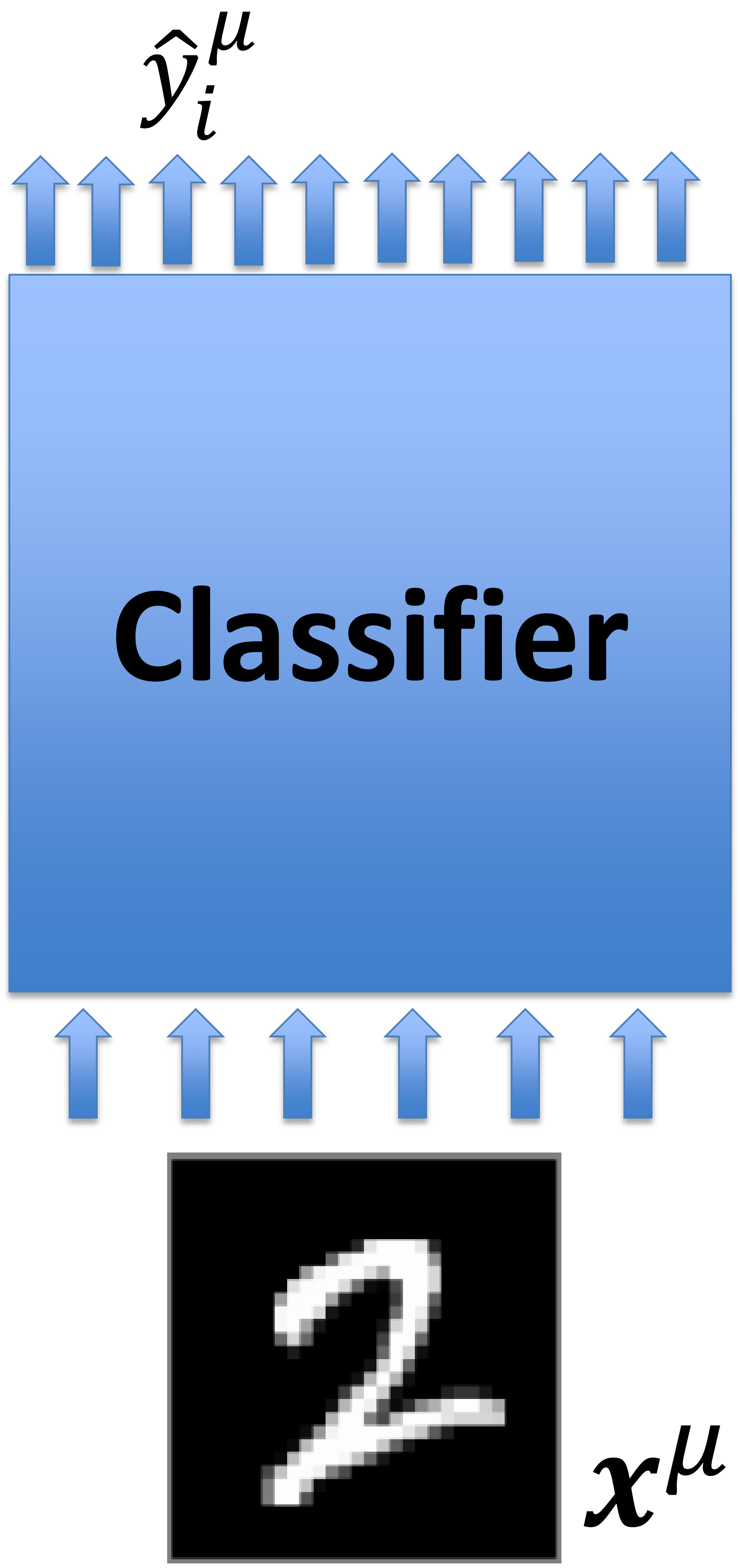
■ = ‘new image’

5. Classification of new inputs: Example

Task: Read Postal Code



10 output units



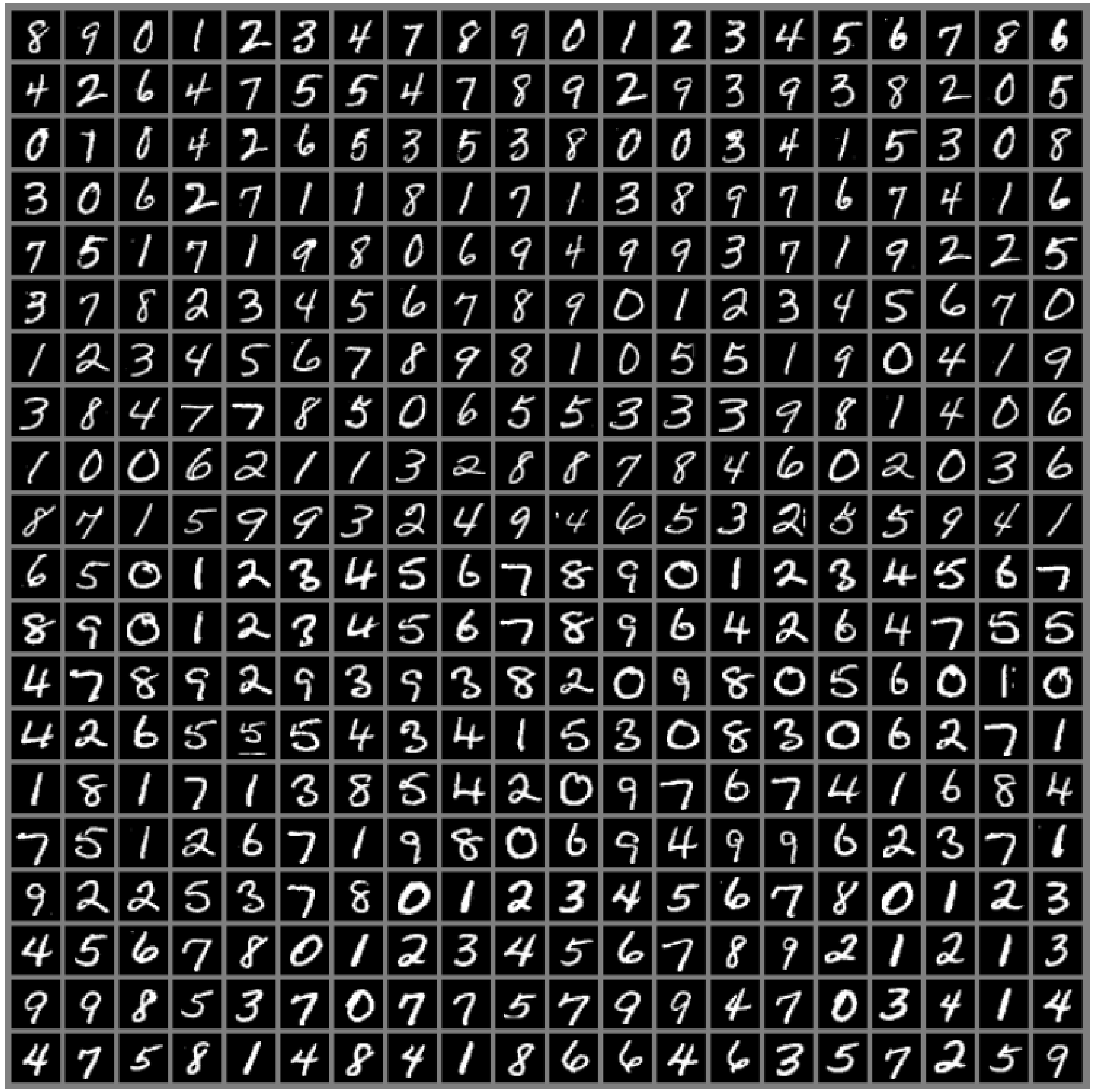
must work on future data!

5. Classification of new inputs: Example

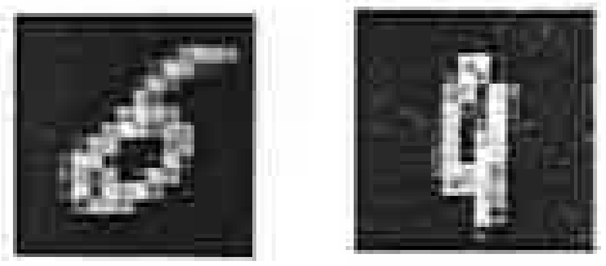
- images 28x28
- Labels: 0, …, 9
- 250 writers
- 60 000 images in training set

Picture: Goodfellow et al, 2016
Data base:
http://yann.lecun.com/exdb/mnist/

MNIST data samples



5. Classification of new inputs: Example



- training data is always noisy
- the future data has different noise

- Classifier must extract the essence
  → **do not fit the noise!!**

5. The problem of overfitting

Big Multilayer perceptrons are flexible and can be trained by BackProp to minimize classification error

… but is flexibility always good?

→ Flexibility is not good for noisy data
→ Danger of overfitting!
→ Control of overfitting by ‘regularization’

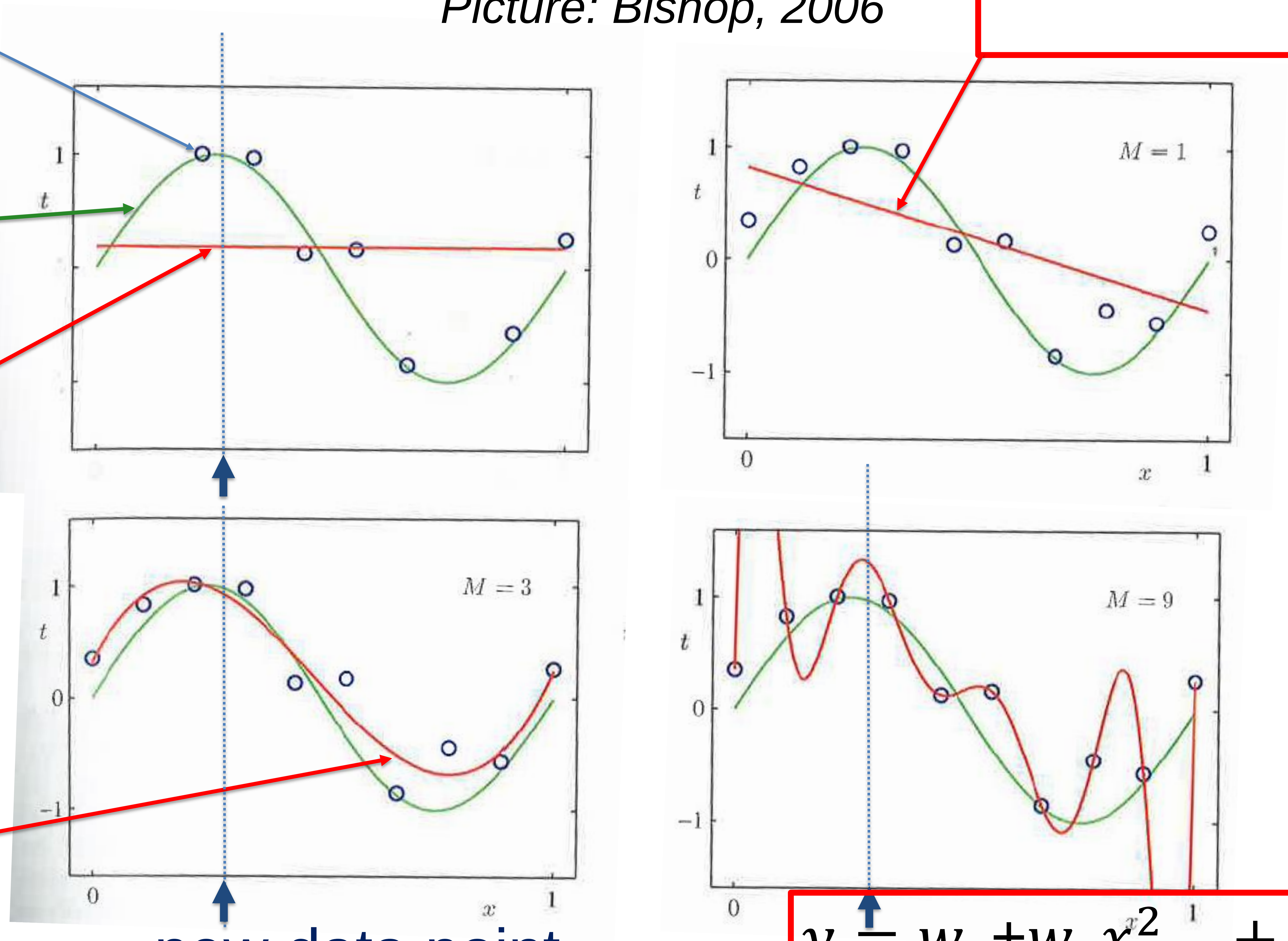
5. Detour: polynomial curve fitting

target data points
= f(x) + *noise*

f(x) = sin(x)

Picture: Bishop, 2006

$y = w_0+w_1x$



fit with $y = w_0$

$y = w_0+w_1x \quad +w_2\, x^2 +w_3x^3$

4 parameters

new data point

$y = w_0+w_1x^2 \ldots + w_9\, x^9$

10 parameters

5. Curve fitting: Quiz

[ ] 20 data points can always be perfectly well fit by a polynomial with 20 parameters

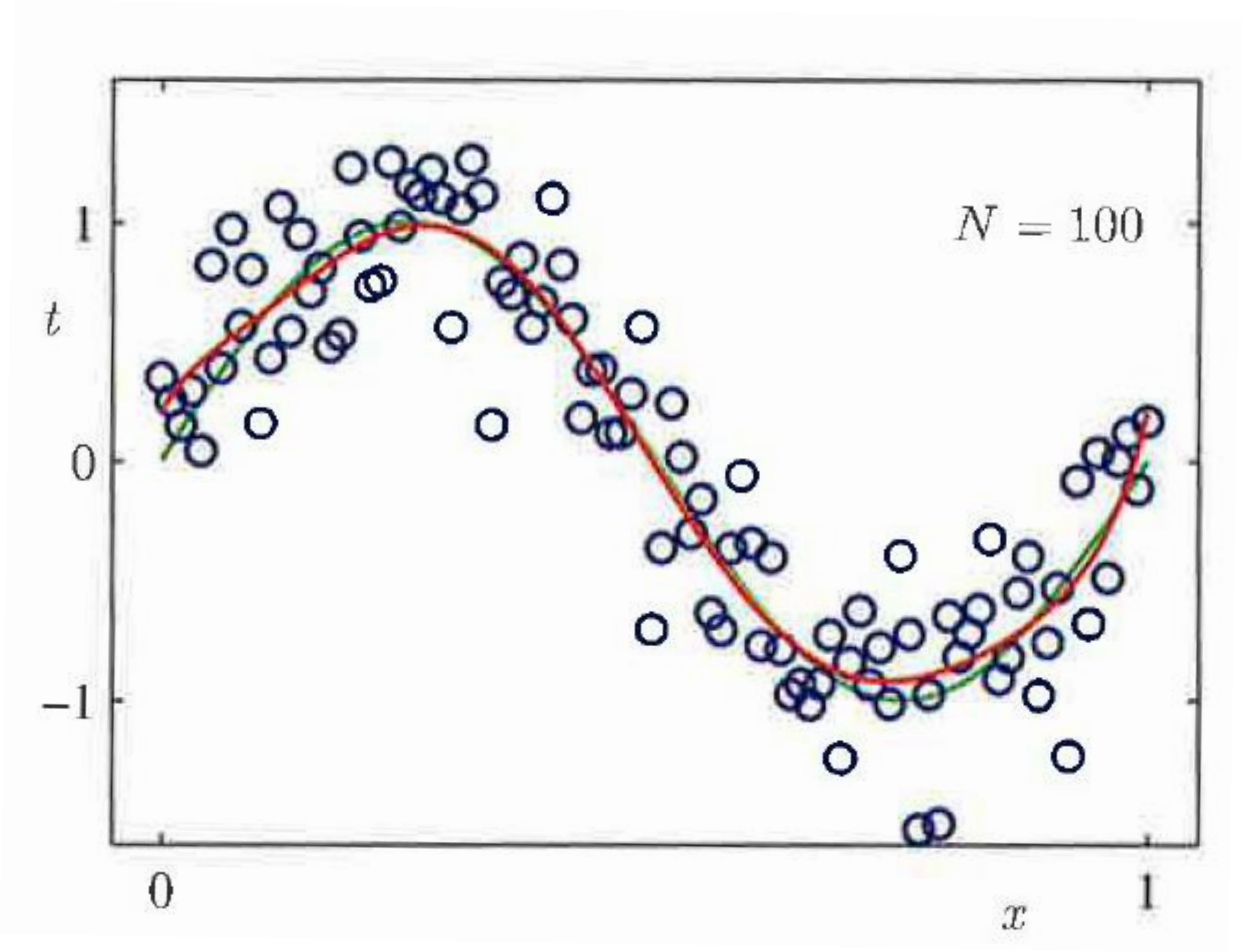
[ ] The prediction for future data is best if the past data is perfectly fit

[ ] A sin-function on $[0,2\pi]$ can be well approximated by a polynomial with 10 parameters

5. Detour: polynomial curve fitting

Fit with P=100 data points

If we have enough data points,
10 parameters are not too much!



Picture: *Bishop, 2006*

$$y = w_0 + w_1 x^2 \ldots + w_9\, x^9$$

10 paramters

5. Detour: curve fitting

- flexibility increases with number of parameter
- flexibility is bad for noisy data
- flexibility OK if we have LARGE amounts of data
- for finite amounts of data, we need to control flexibility!

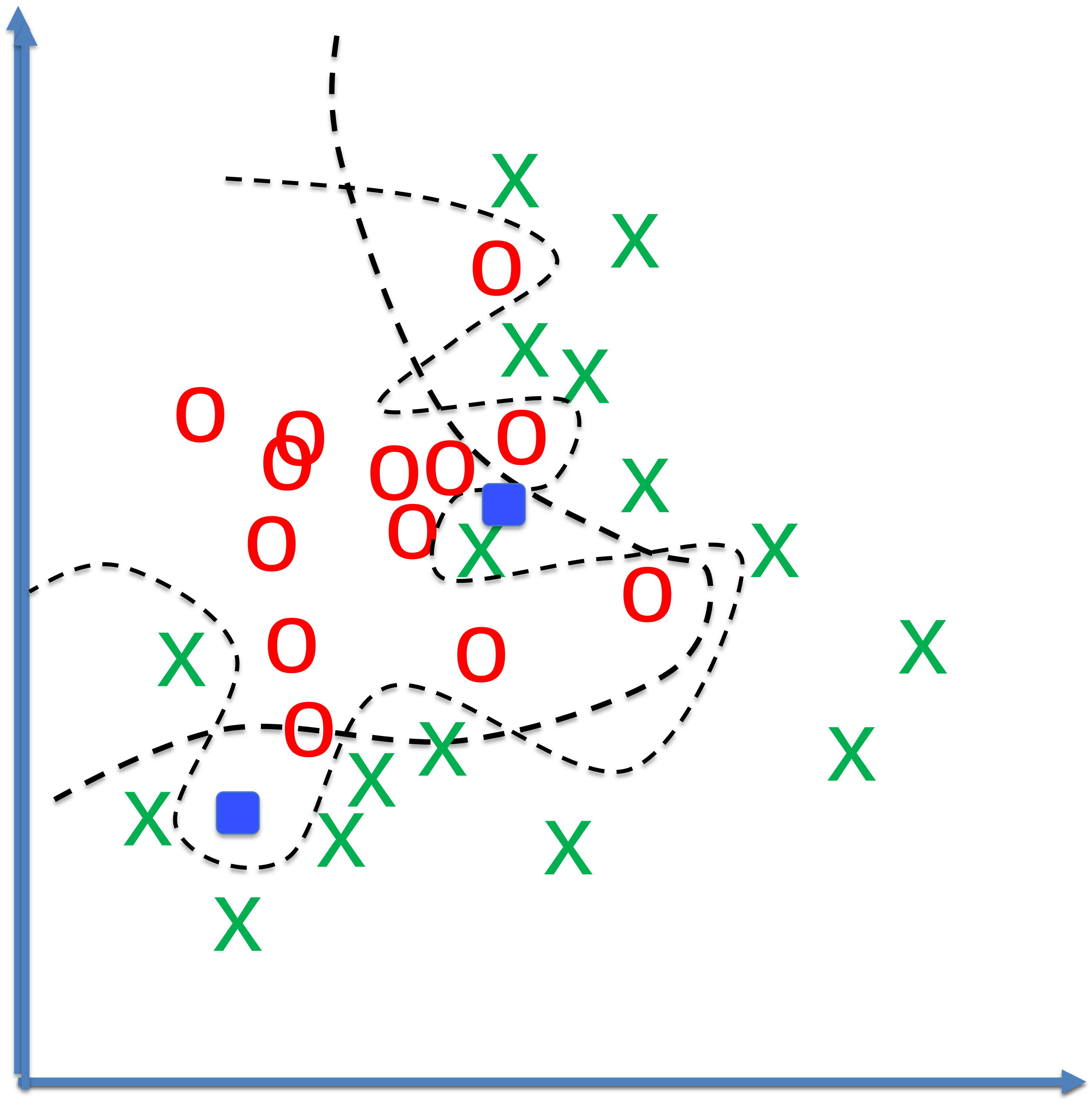
→ See any course on *Machine Learning*

5. The problem of overfitting

Big Multilayer perceptrons are flexible and can be trained by BackProp to minimize classification error

… but is flexibility always good?

→ Flexibility is bad for noisy data
→ Danger of overfitting!
→ Control flexibility!



Wulfram Gerstner
EPFL, Lausanne, Switzerland

Artificial Neural Networks: Lecture 2
Backprop and multilayer perceptrons

1. Modern Gradient Methods
2. XOR problem
3. Multilayer Perceptron
4. BackProp Algorithm
5. The problem of overfitting
6. **Training base and Validation base**

6. Training base and validation base

Our data base contains

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

input target output

Split data base $P = P1 + P2$

$\{(\boldsymbol{x}^{\mu}, t^{\mu}), \quad 1 \leq \mu \leq P1\};$

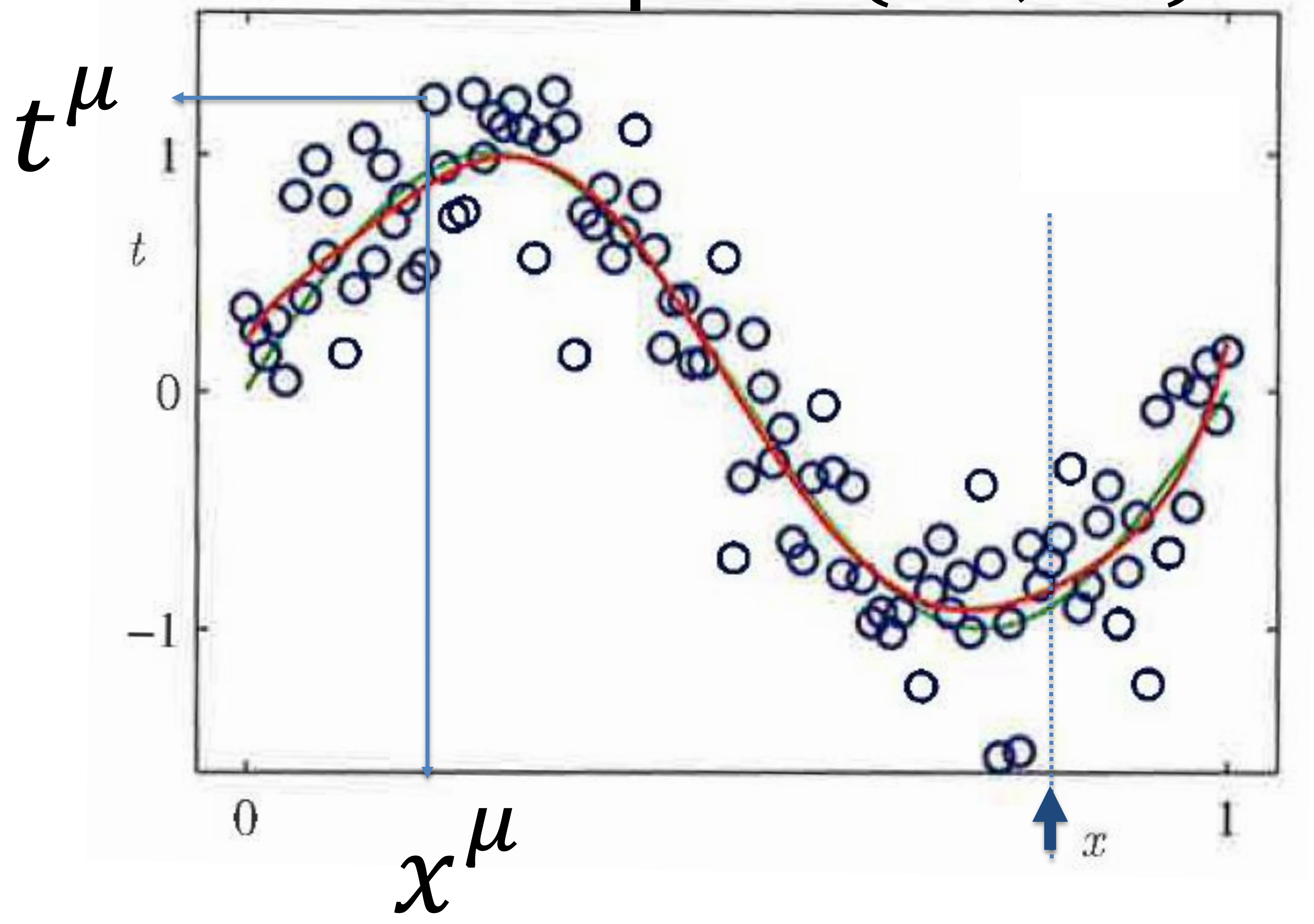
Training base, used
to optimize parameters

$\{(\boldsymbol{x}^{\mu}, t^{\mu}), \quad P1 + 1 \leq \mu \leq P\};$

Validation base, used
to mimic ‘future data’

6. Error function on training data and validation data

Definition of pair $(\boldsymbol{x}^{\mu}, t^{\mu})$



Minimize error on **training set**

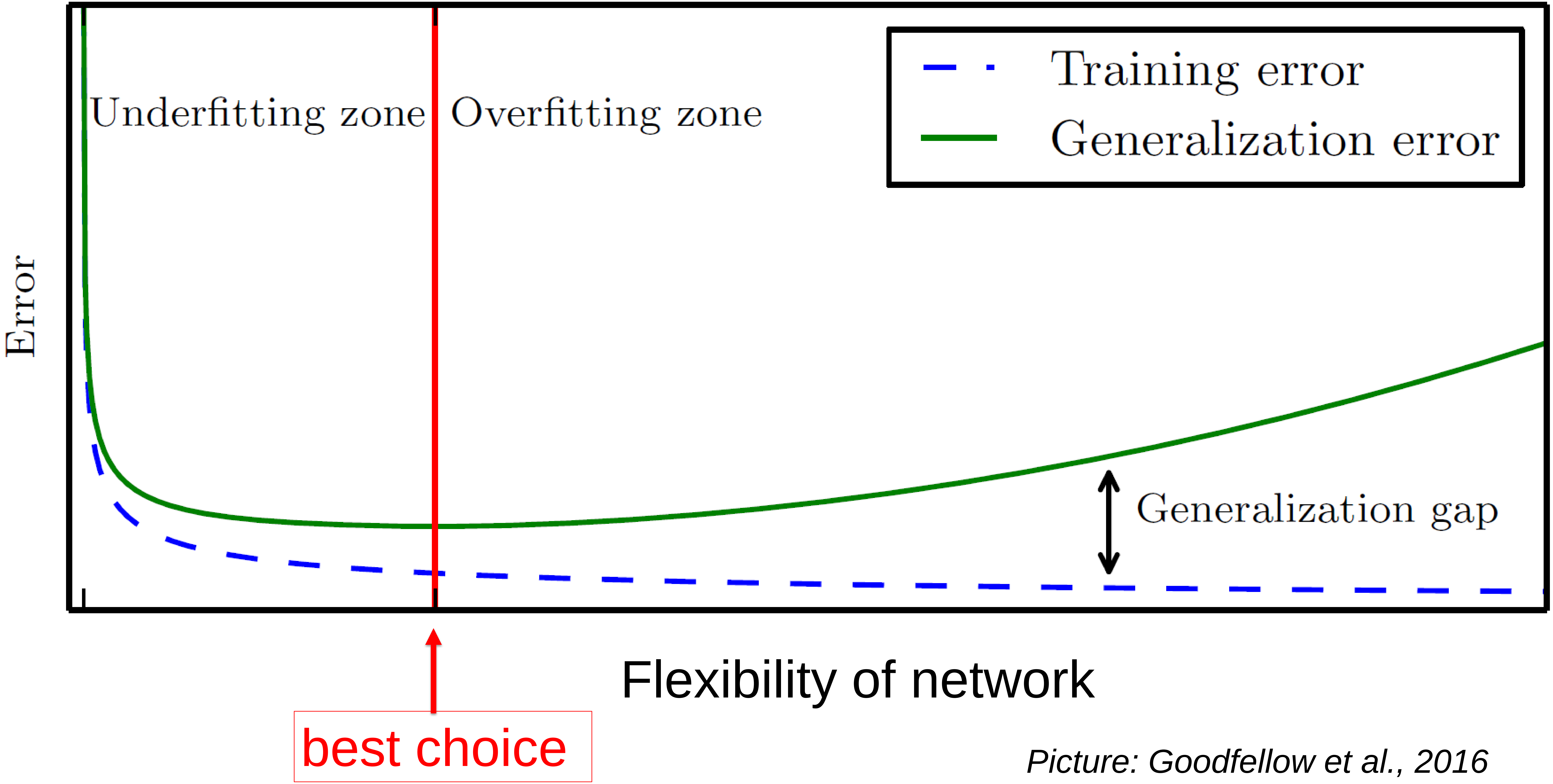
$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{P1} \left[t^{\mu} - \hat{y}^{\mu}\right]^2$$

Evaluate validation error on new data (**validation set**)

$$E^{\text{val}}(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=P1+1}^{P} \left[t^{\mu} - \hat{y}^{\mu}\right]^2$$

Picture: Bishop, 2006

6. Error function on training data and validation data



Picture: Goodfellow et al., 2016

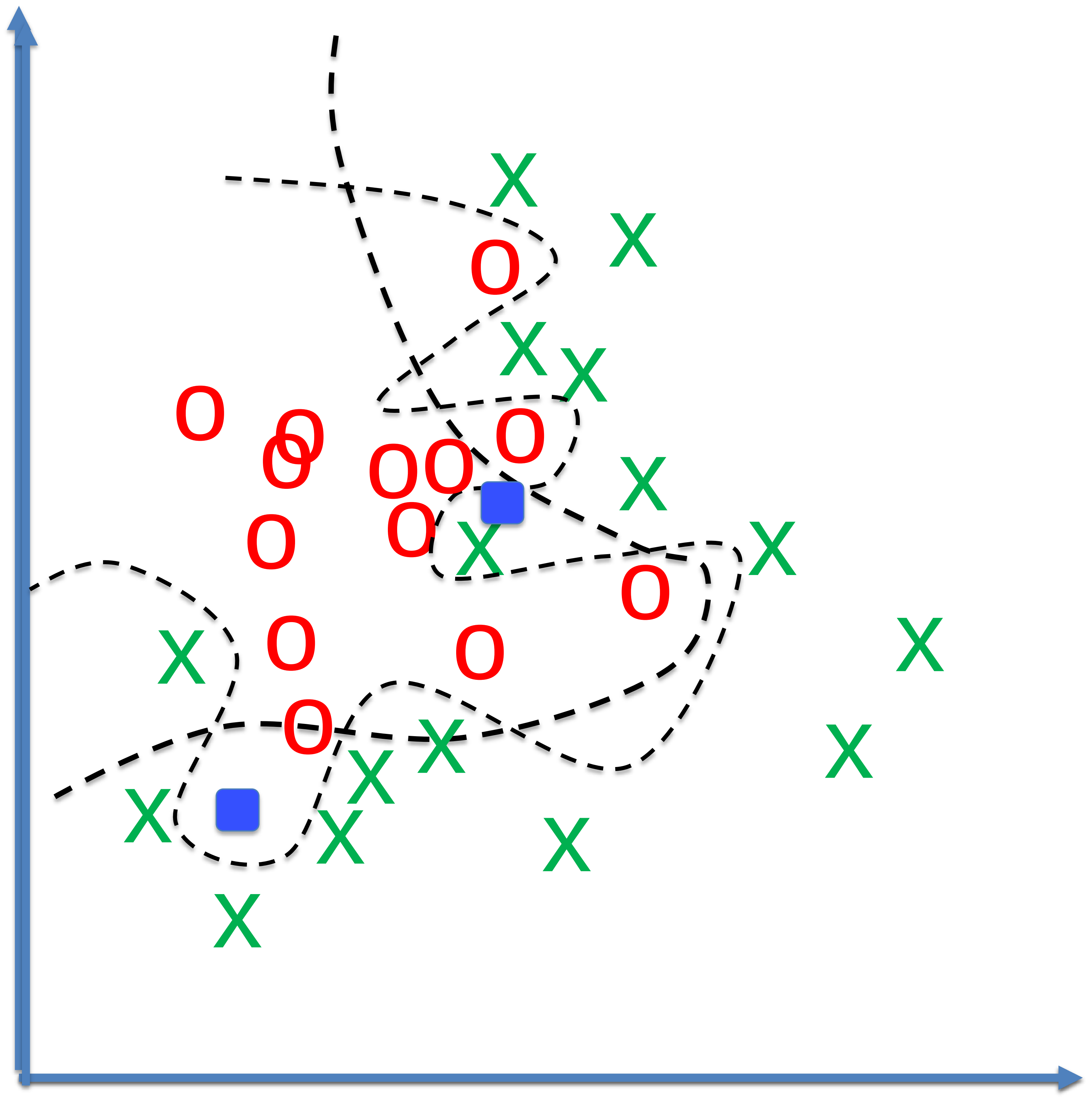
6. The problem of overfitting (revisited)

Big Multilayer perceptrons are flexible and can be trained by BackProp to minimize classification error

… but is flexibility always good?

→ Flexibility is bad for noisy data
→ Danger of overfitting!
→ Control flexibility!

We can control overfitting by splitting into training base and validation base



6. Control of flexibility with Artificial Neural networks

1 **Change flexibility** (several times)

Choose number of hidden neurons and number of layers

2 **Split data base into training base and validation base**

3 **Optimize parameters** (several times):

Initialize weights

4 **Iterate until convergence**

Gradient descent on training error

Report training error and validation error

Report mean training and validation error and standard dev.

Plot mean training and validation error

Pick optimal number of layers and hidden neurons

Artificial Neural Networks: Lecture 2
Backprop and multilayer perceptrons

Wulfram Gerstner
EPFL, Lausanne, Switzerland

1. Modern Gradient Methods
2. XOR problem
3. Multilayer Perceptron
4. BackProp Algorithm
5. The problem of overfitting
6. Training base and validation base
7. **Simple Regularization**

7. Controling Flexibility

Flexibility = number of free parameters

→ Change flexibility = change network structure or number of hidden neurons

Flexibility = ‘**effective**’ number of free parameters

→ Change flexibility = regularization of network

7. Regularization by a penalty term

Minimize on **training set a modified Error function**

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{P1} \left[t^{\mu} - \hat{y}^{\mu}\right]^2 \quad + \ \lambda \text{ penalty}$$

assigns an 'error'
to flexible solutions

check 'normal' error on separate data (**validation set**)

$$E^{\text{val}}(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=P1+1}^{P} \left[t^{\mu} - \hat{y}^{\mu}\right]^2$$

7. Regularization by a weight decay (L2 regularization)

Specific example: L2-regularization

Minimize on **training set a modified Error function**

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{P1} \left[t^{\mu} - \hat{y}^{\mu}\right]^2 + \lambda \sum_{k} (w_k)^2$$

assigns an 'error' to solutions with large pos. or neg. weights

check 'normal' error on separate data (**validation set**)

$$E^{val}(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=P1+1}^{P} \left[t^{\mu} - \hat{y}^{\mu}\right]^2$$

7. Regularization: Quiz

If we **increase the penalty parameter** λ

[ ] the flexibility of the fitting procedure **increases**

[ ] the flexibility of the fitting procedure **decreases**

[ ] the ‘**effective**’ number of free parameters **decreases**

[ ] the ‘**effective**’ number of free parameters **remains the same**

[ ] the **‘explicit’** number of parameters **remains the same**

9. Example: Noisy XOR problem, as a function of training time

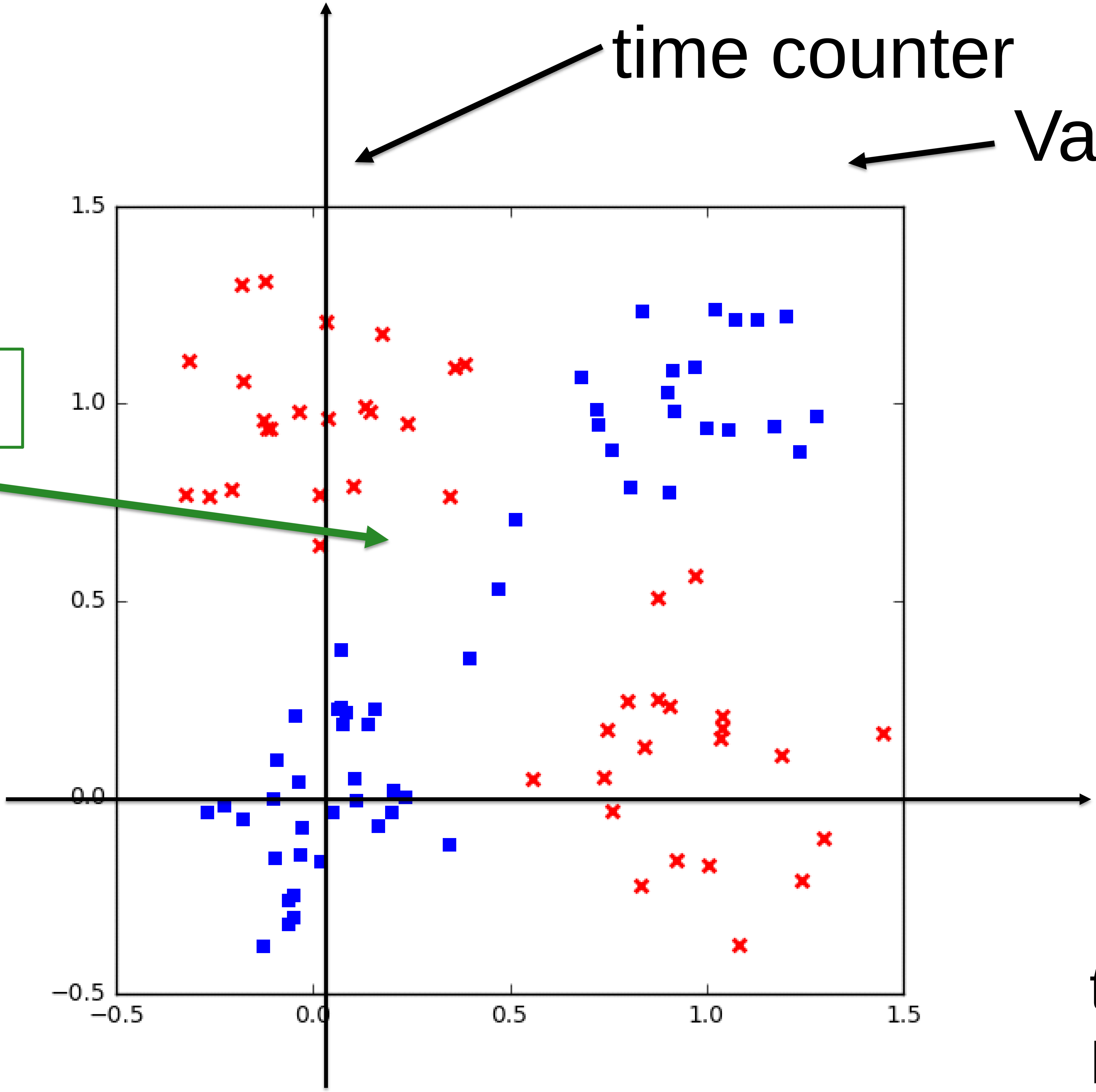
100 data points

low noise
high noise

interesting region

time counter

Validation error



thanks to
Florian Colombo

Objectives for today:

- XOR problem and the need for multiple layers
  → hidden layer provide flexibility
- understand backprop as a smart algorithmic implementation of the chain rule
  → algorithmic differentiation is better than numeric differentiation
- hidden neurons add flexibility, but flexibility is not always good
  → control flexibility by regularizati
  use validation data to find hyperparameters
- training base and validation base: the need to predict well for future data
  →test Error