

Artificial Neural Networks: Lecture 2

Backprop and multilayer perceptrons

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
- training base, validation and test base: the need to predict well for future data

Your Semester planning

Wulfram Gerstner
EPFL, Lausanne, Switzerland

The course '**Deep Learning**' (Fleuret)
and the course '**Artificial Neural Networks**' (Gerstner)
have more than 50 percent overlap.

**I suggest to take either one or the other but not both (XOR),
but the CS section director allows you to take both in 2018**

The course '**Unsupervised and Reinforcement L.**' (Gewaltig)
and the course '**Artificial Neural Networks**' (Gerstner)
have about 50 percent overlap.

I suggest to take either one or the other but not both (XOR).

Your semester planning

Wulfram Gerstner
EPFL, Lausanne, Switzerland

The course '**Deep Learning**' (Fleuret)
and the course '**Unsupervised and Reinforcement L.**' (Gewaltig)
have less than 5 percent overlap.

You can take one or the other or both (OR).

- +The course '**Unsupervised and Reinforcement L.**' (Gewaltig)
is oriented towards biological questions
- +The course '**Deep Learning**' (Fleuret) is an applied course.
It has no prerequisites and does not cover reinforcement learning
- + The course '**Artificial Neural Networks**' (Gerstner) is a course
aimed at IC students. **Prerequisite: Machine Learning** (Jaggi)

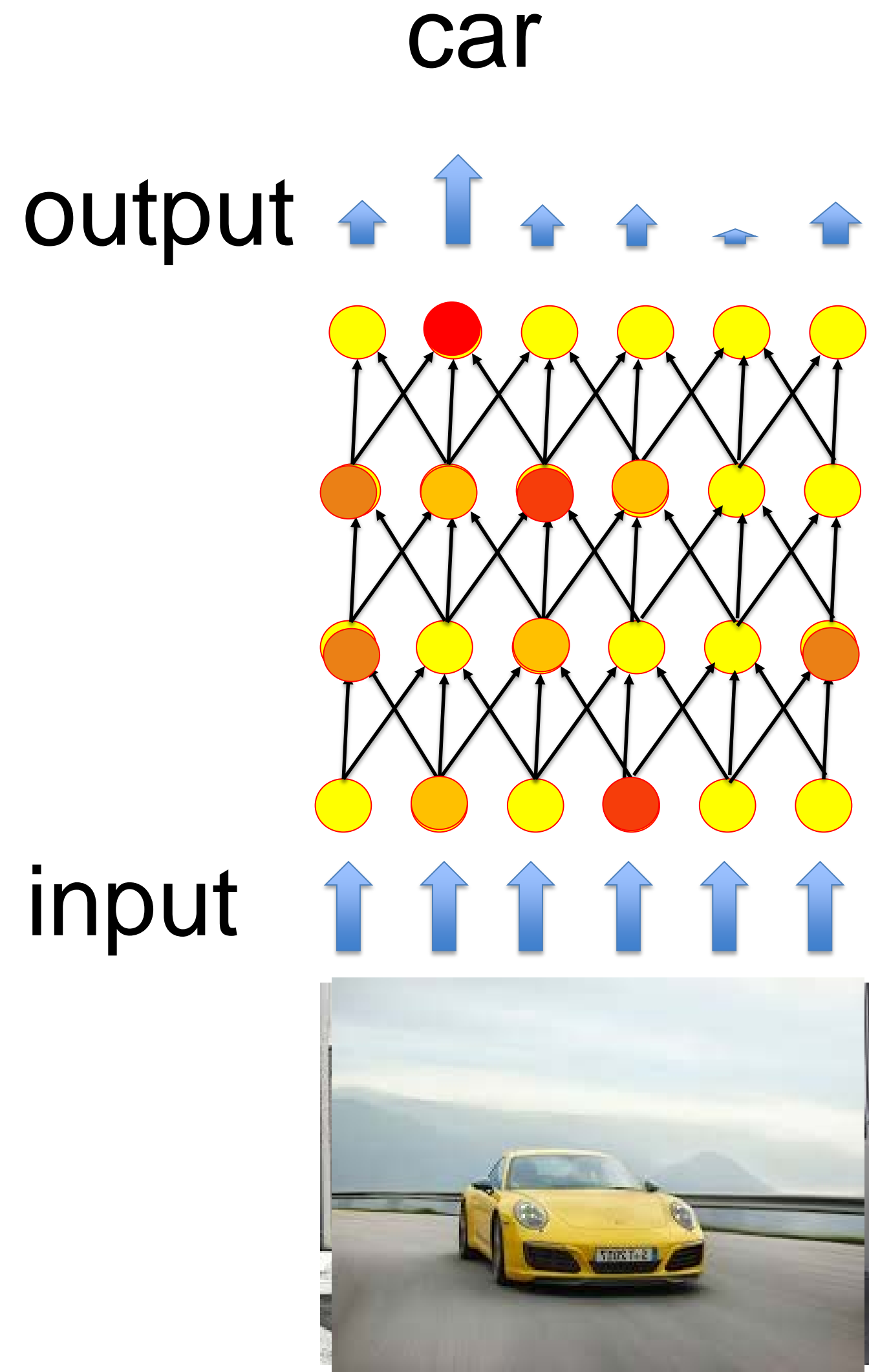
Artificial Neural Networks: Lecture 2

Backprop and multilayer perceptrons

Objectives for today:

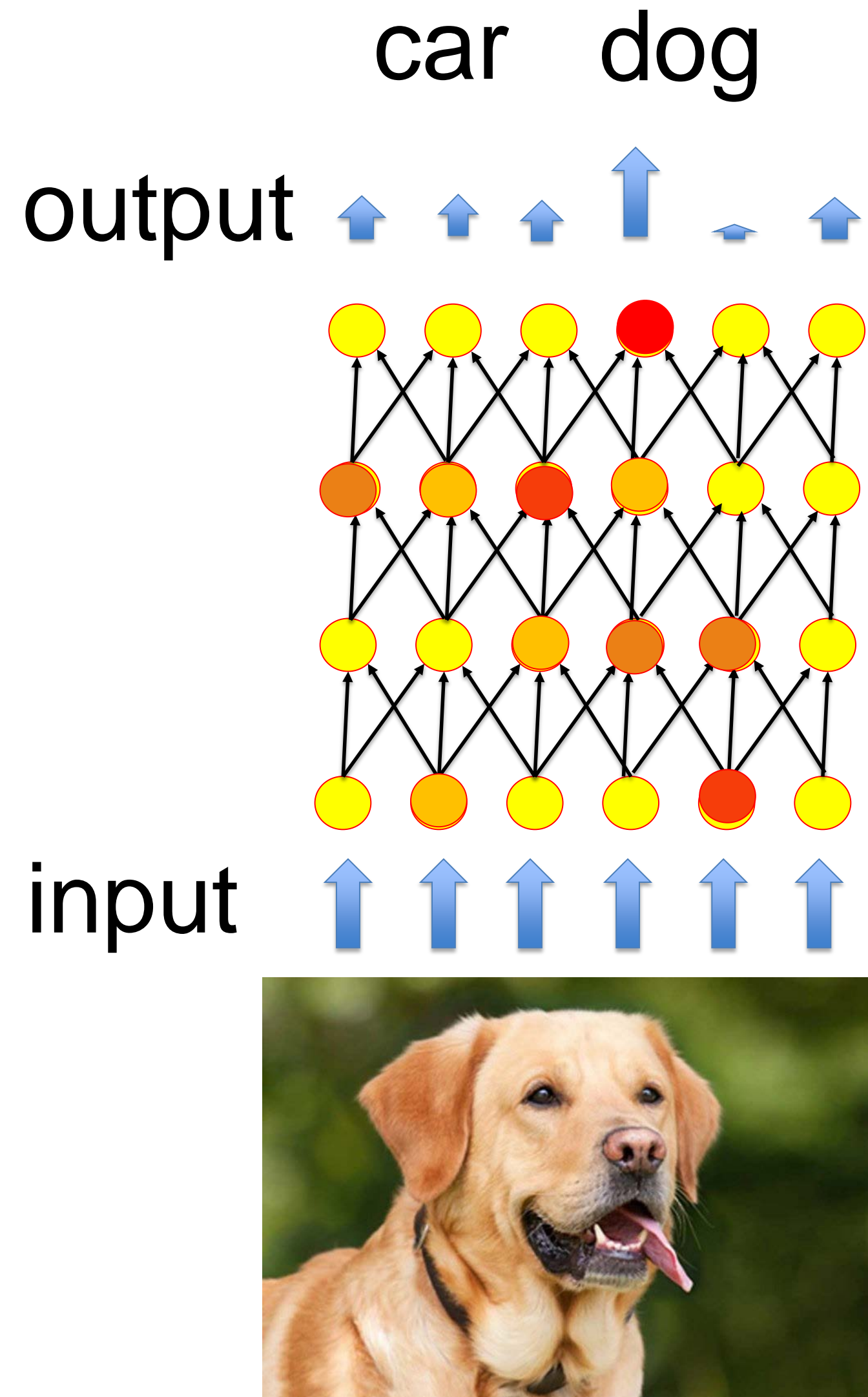
- XOR problem and the need for multiple layers
- understand backprop as a smart algorithmic implementation of the chain rule
- hidden neurons add flexibility, but flexibility is not always good
- training base, validation and test base: the need to predict well for future data

Review: Artificial Neural Networks for classification



review: Artificial Neural Networks for classification

Aim of learning:
Adjust connections such
that output is correct
(for each input image,
even new ones)



Review: Data base for Supervised learning (single output)

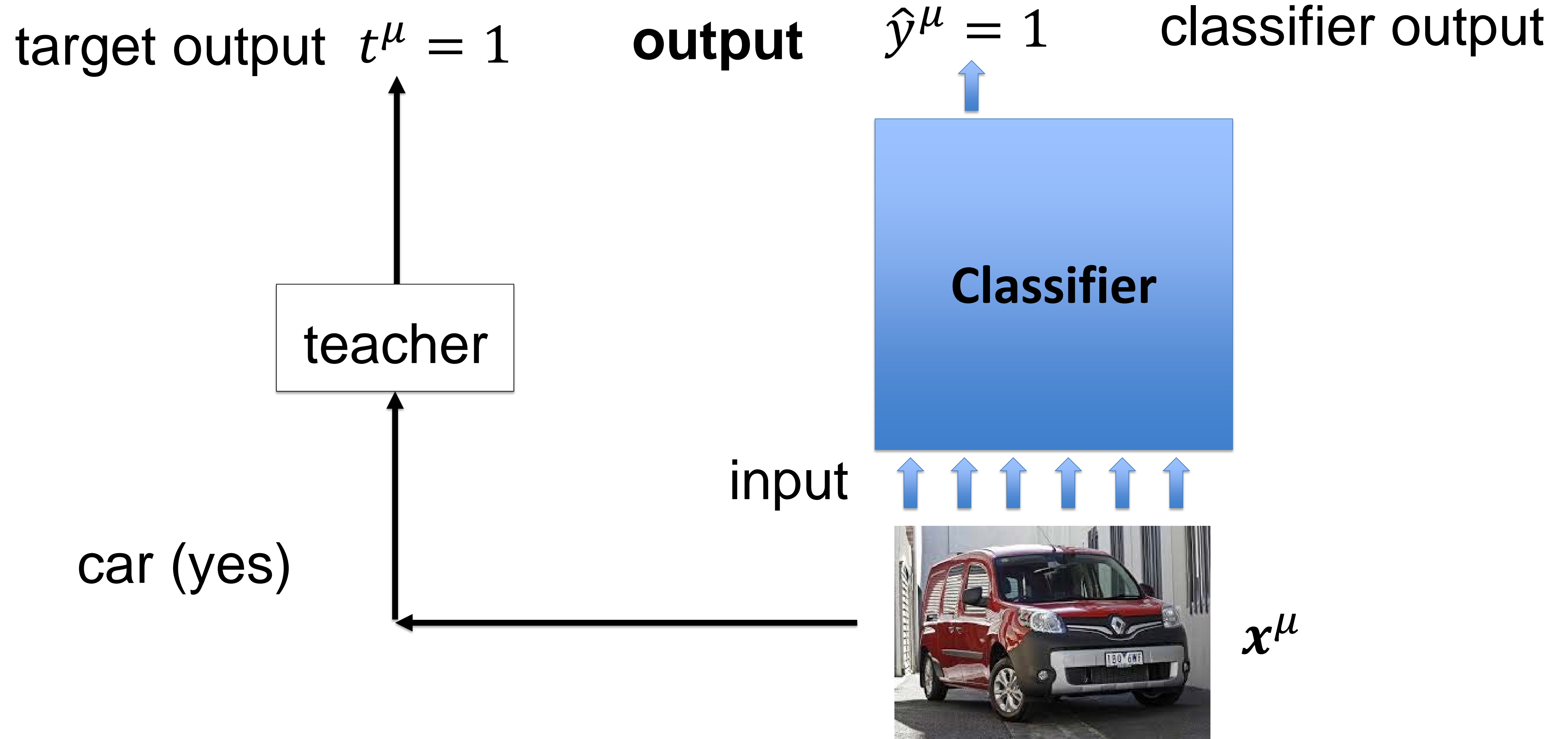
P data points $\{ (\mathbf{x}^\mu, t^\mu) , \quad 1 \leq \mu \leq P \};$


input target output

$t^\mu = 1$ car =yes

$t^\mu = 0$ car =no

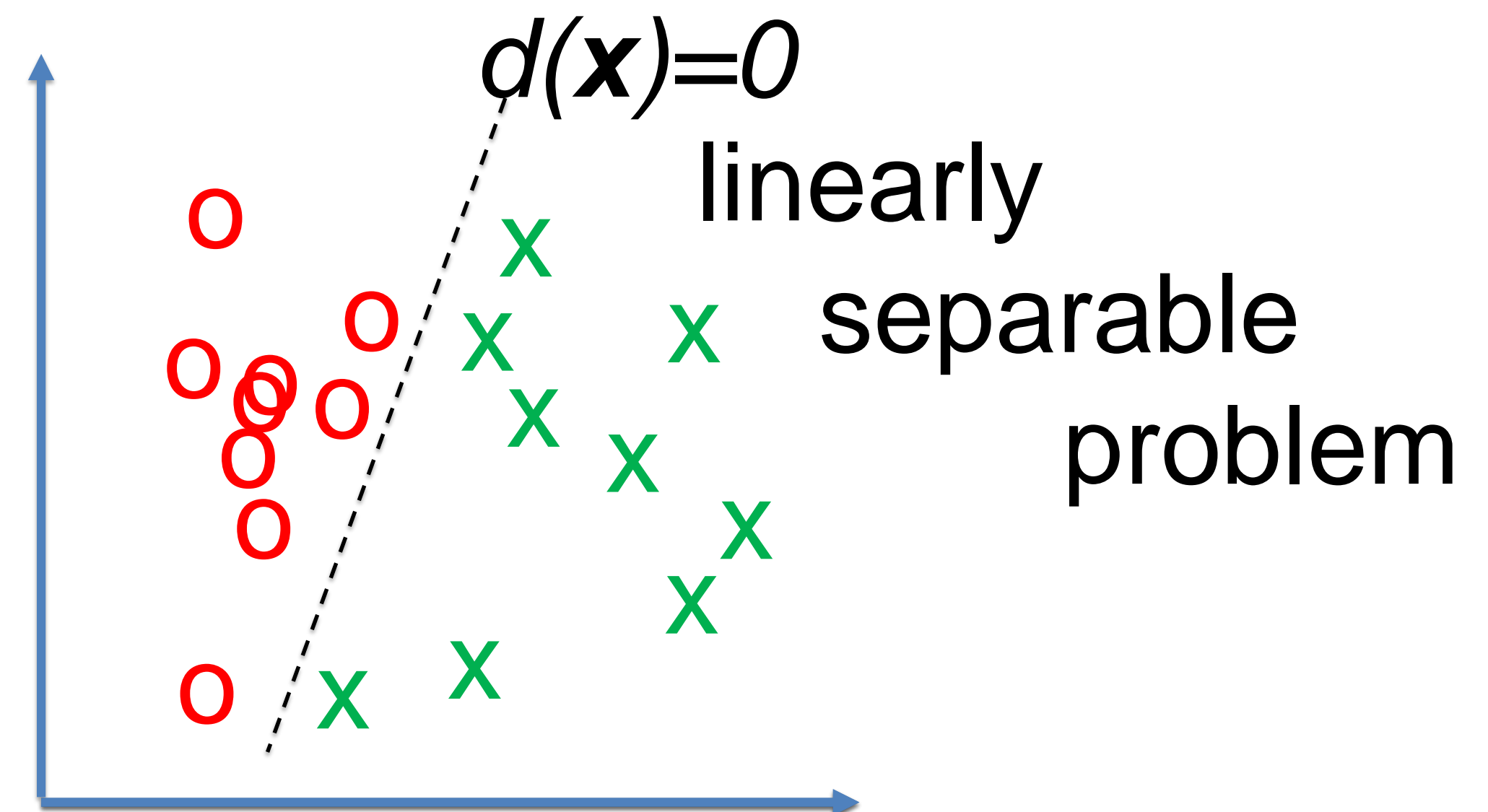
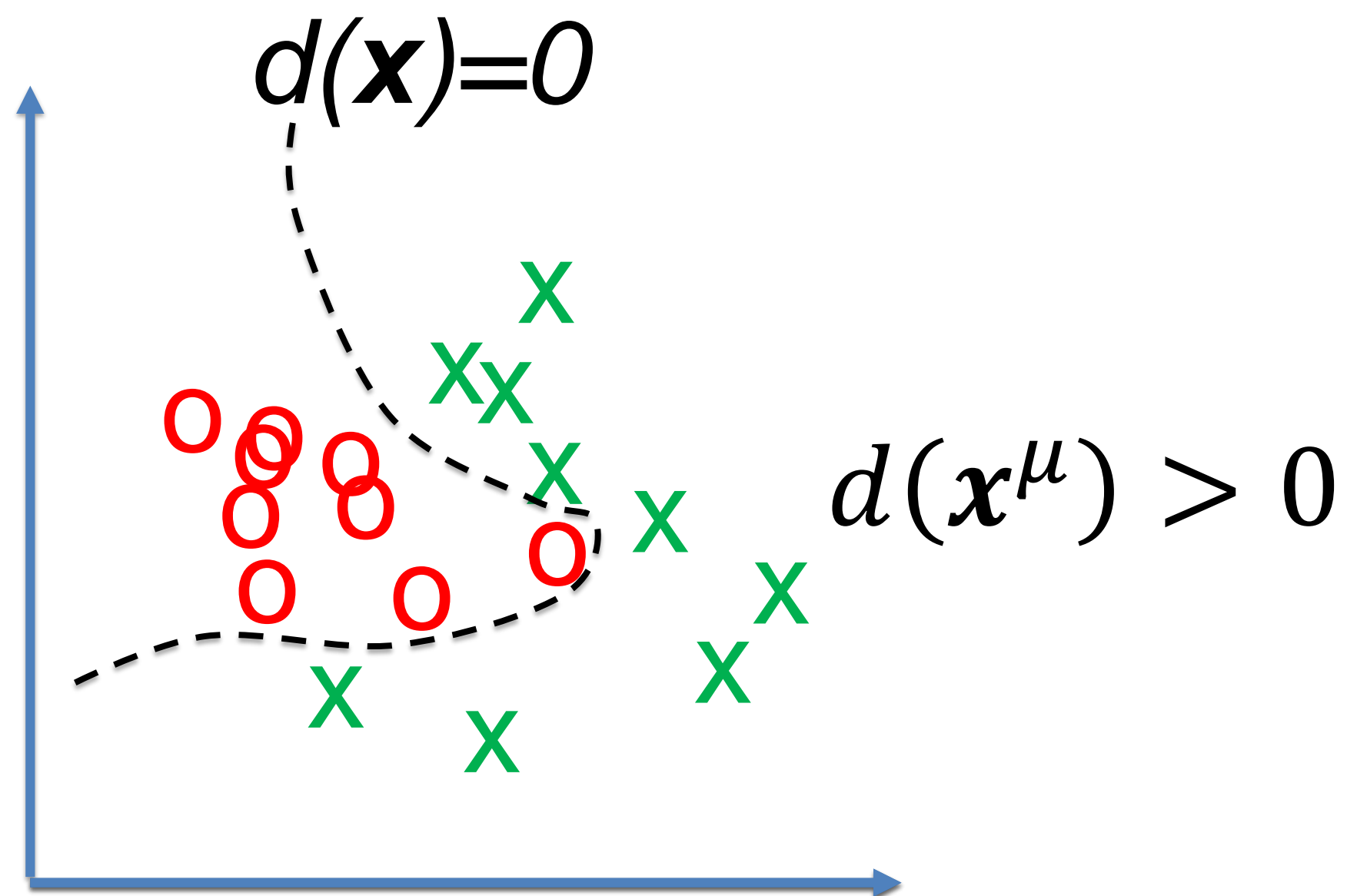
review: Supervised learning



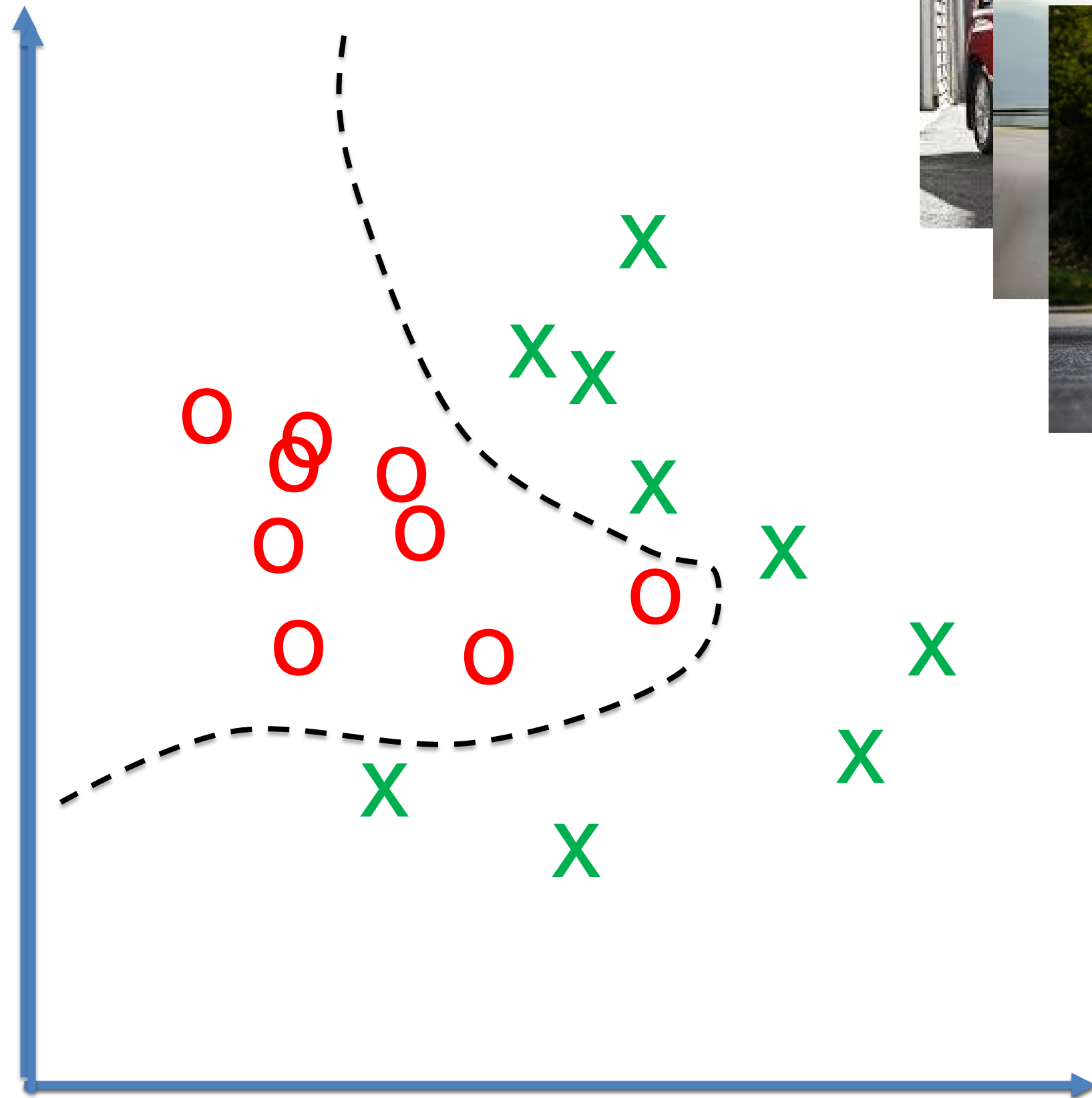
Review: Classification as a geometric problem

Task of Classification

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

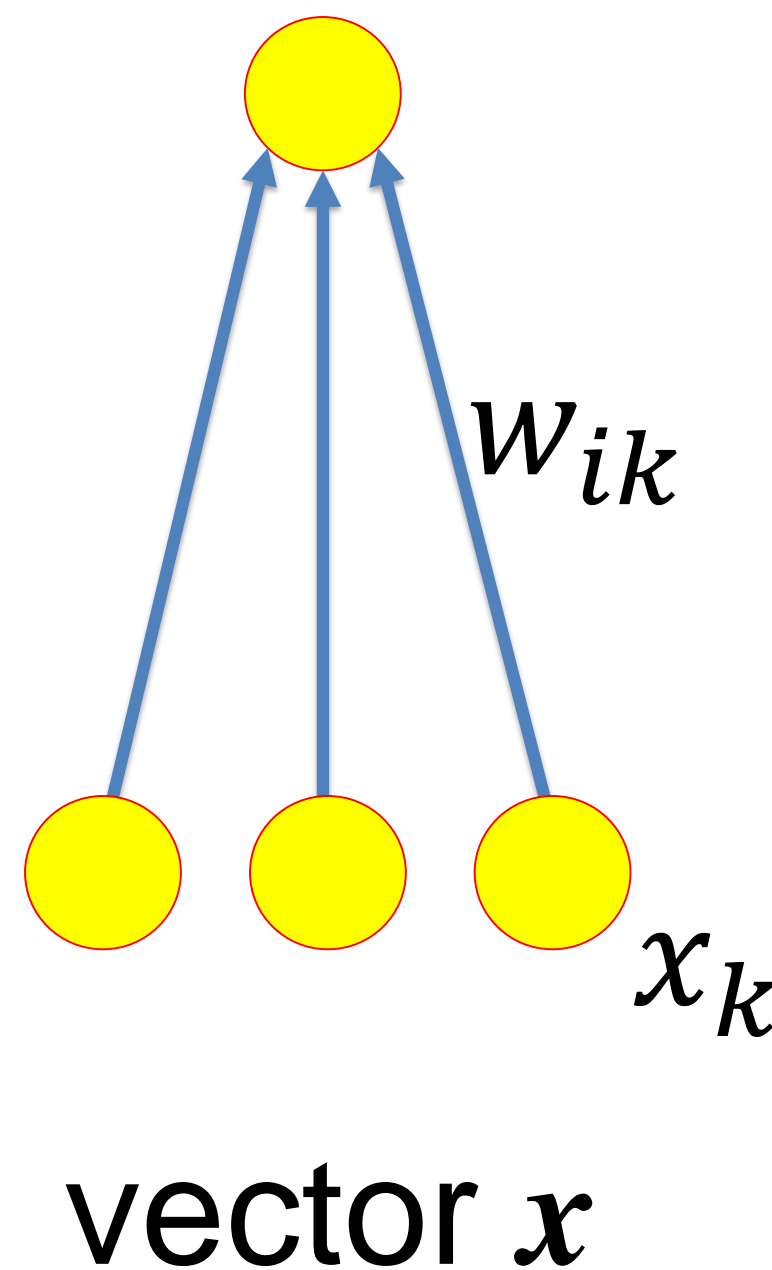


Review: Classification as a geometric problem

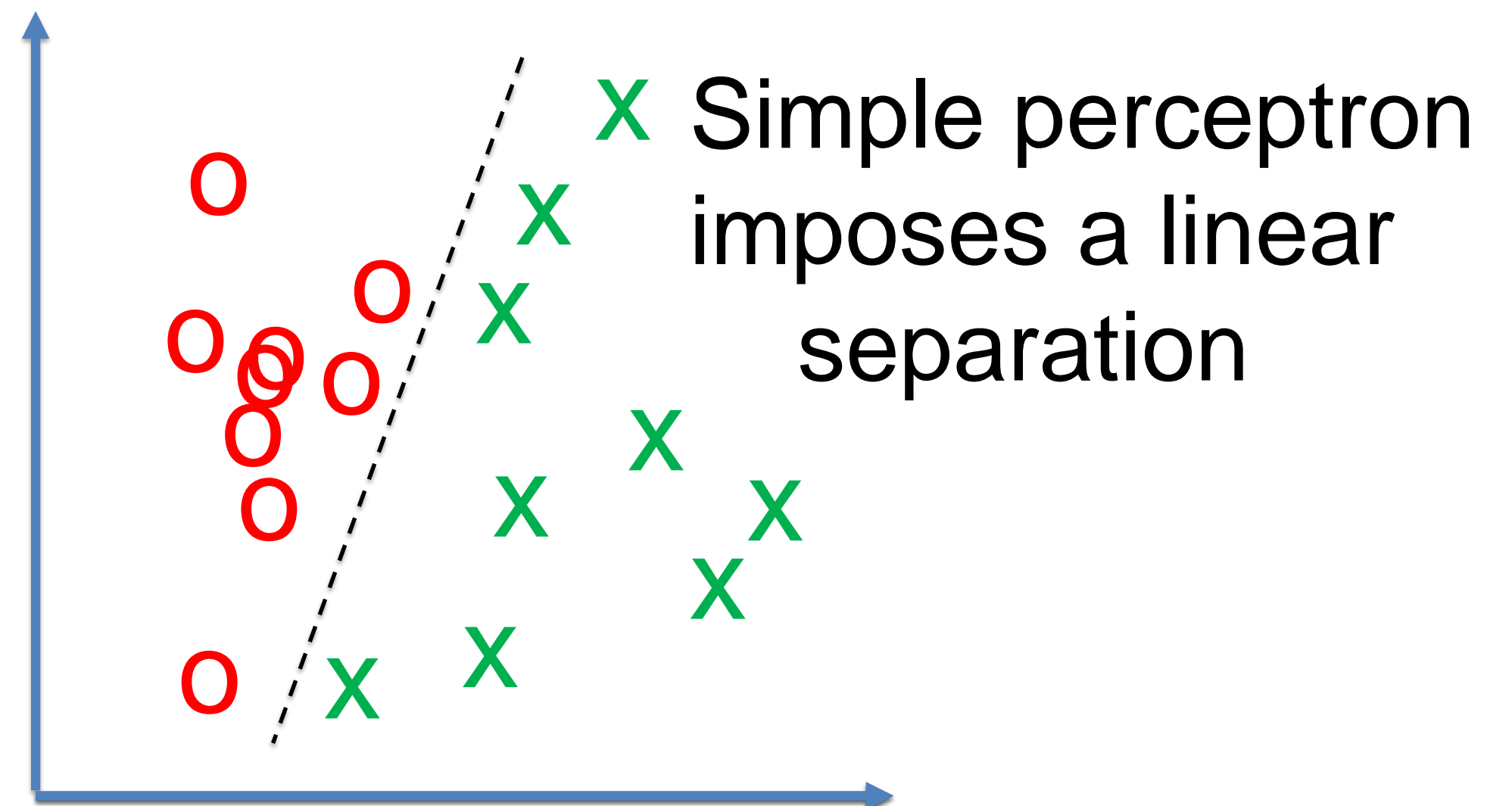


Review: Single-Layer networks: simple perceptron

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

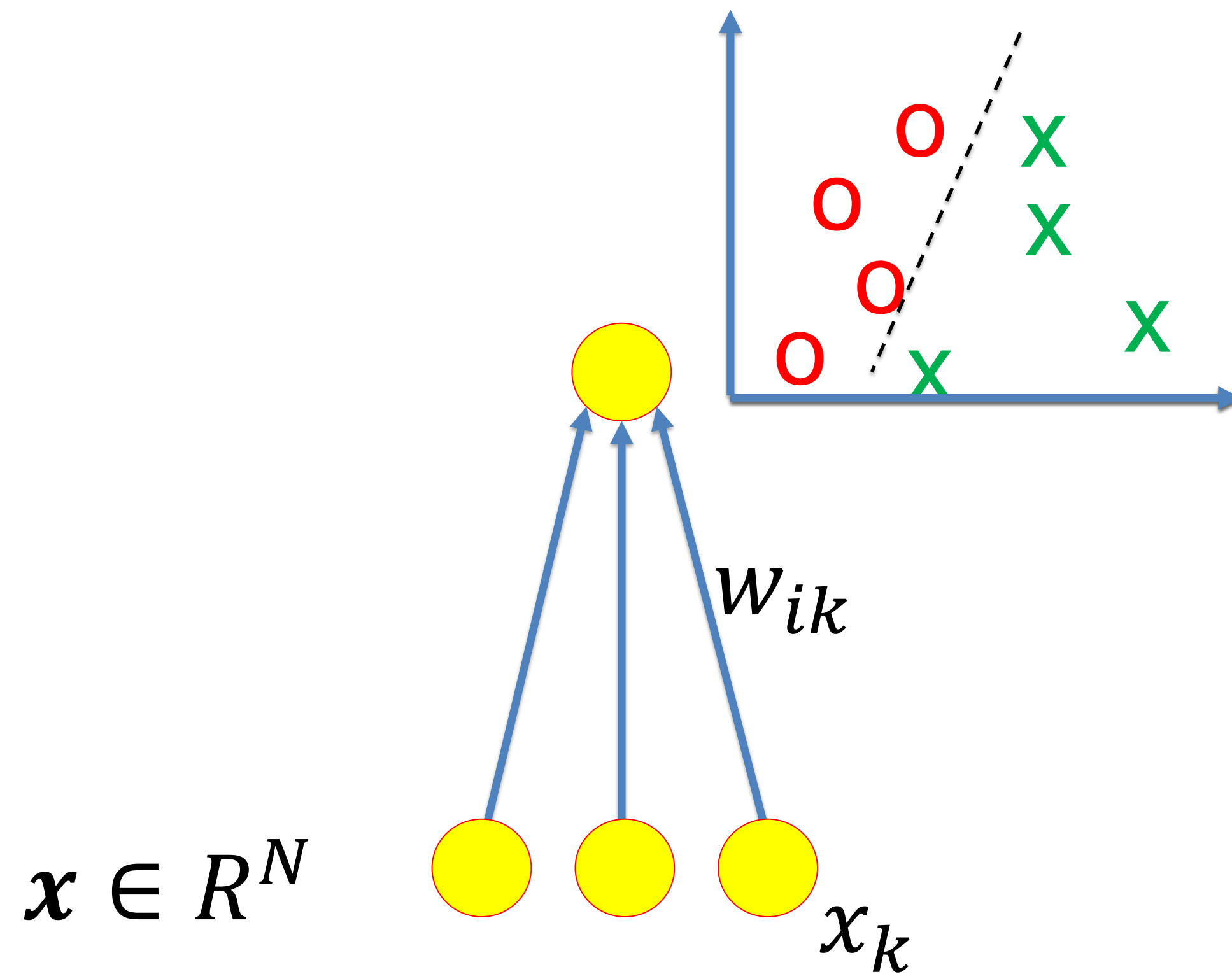


$$d(x) = \sum_k w_k x_k - \vartheta = 0$$

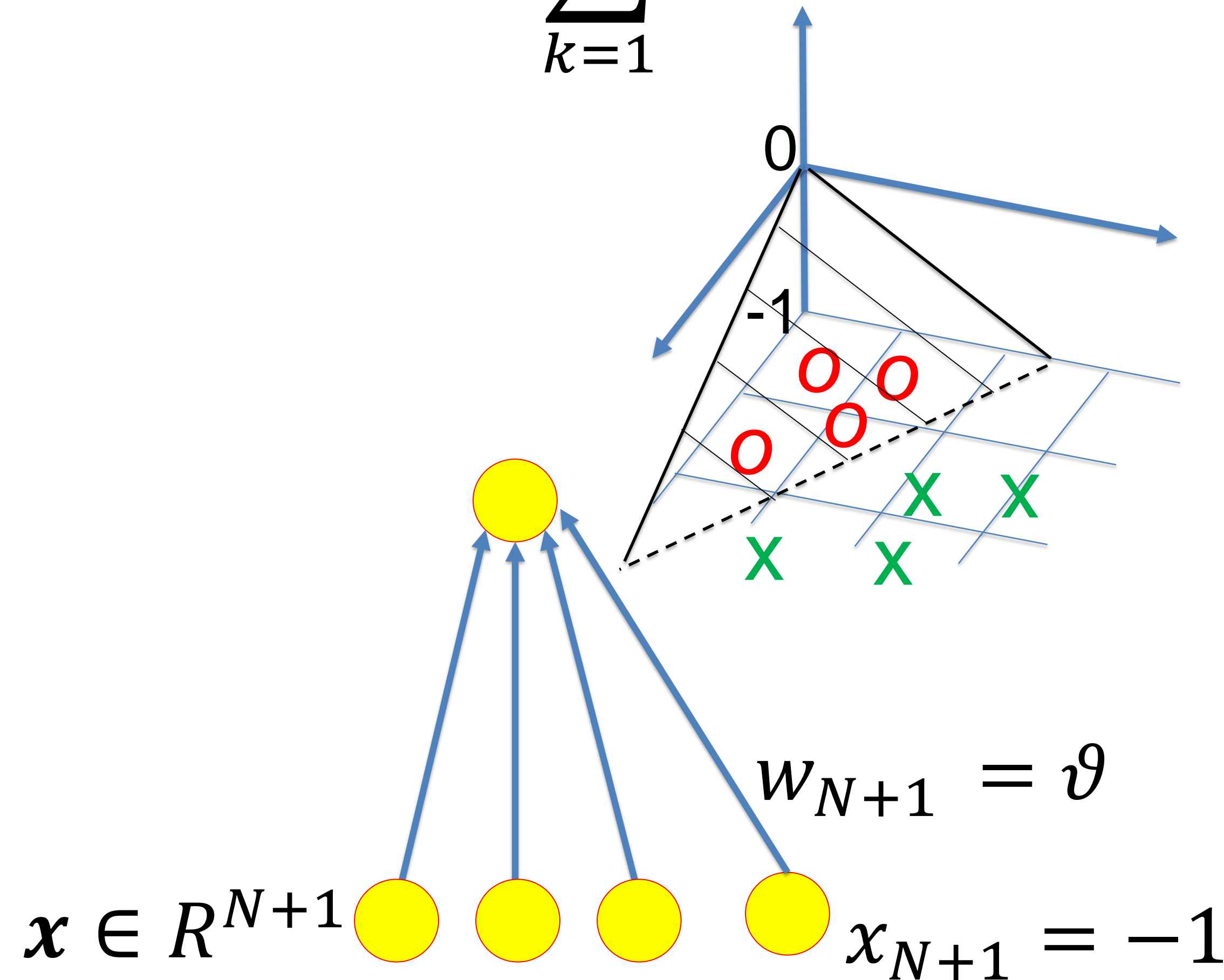


Review: remove threshold: add extra input

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



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



Review: Single-Layer networks

a simple perceptron

- can only solve linearly separable problems
- imposes a separating hyperplane
- in **$N+1$** dimensions hyperplane always goes through origin
- Adapt weights by gradient descent
(perceptron algo and other algos)

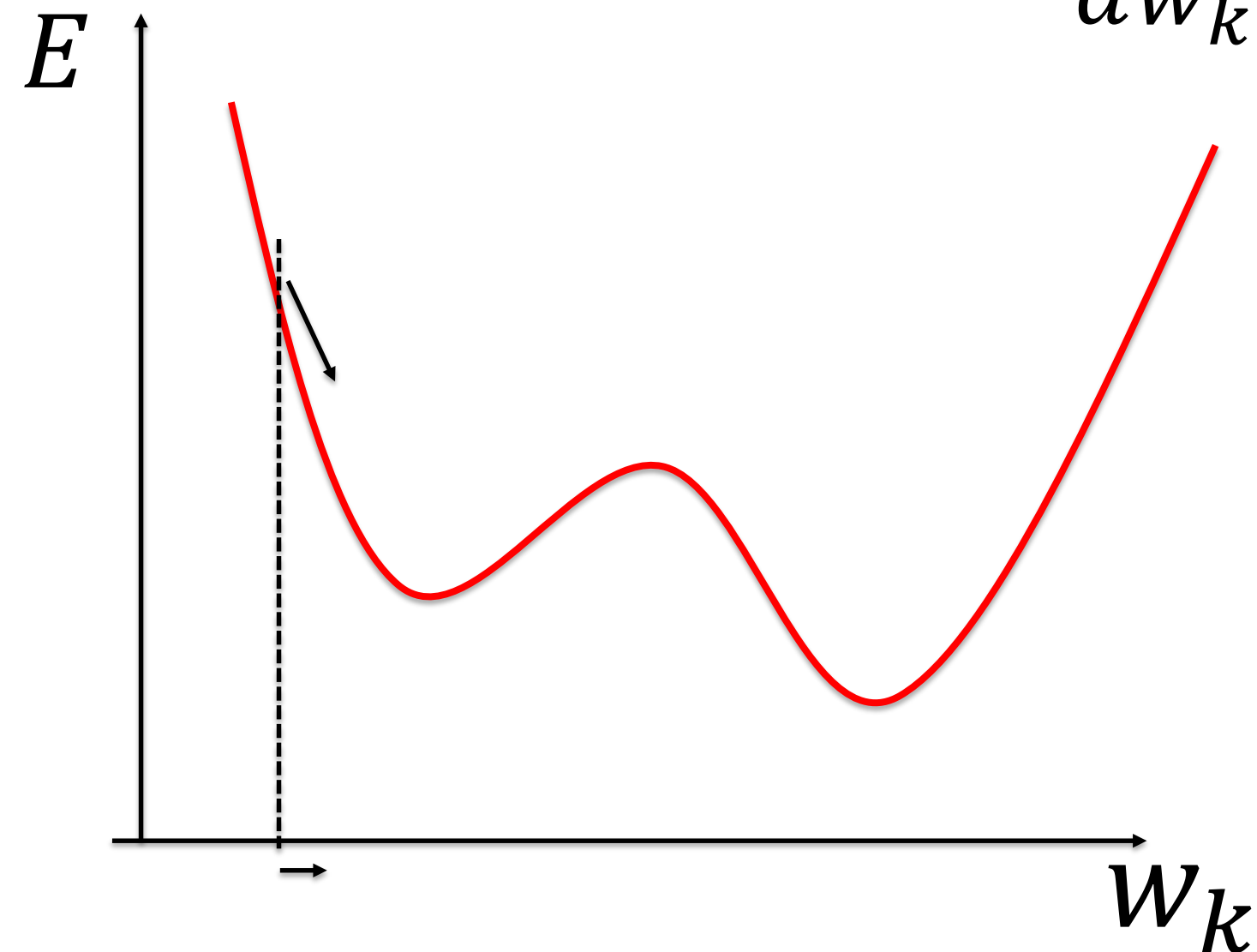
Review: gradient descent

Quadratic error

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

gradient descent

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



Batch rule:

one update after all patterns

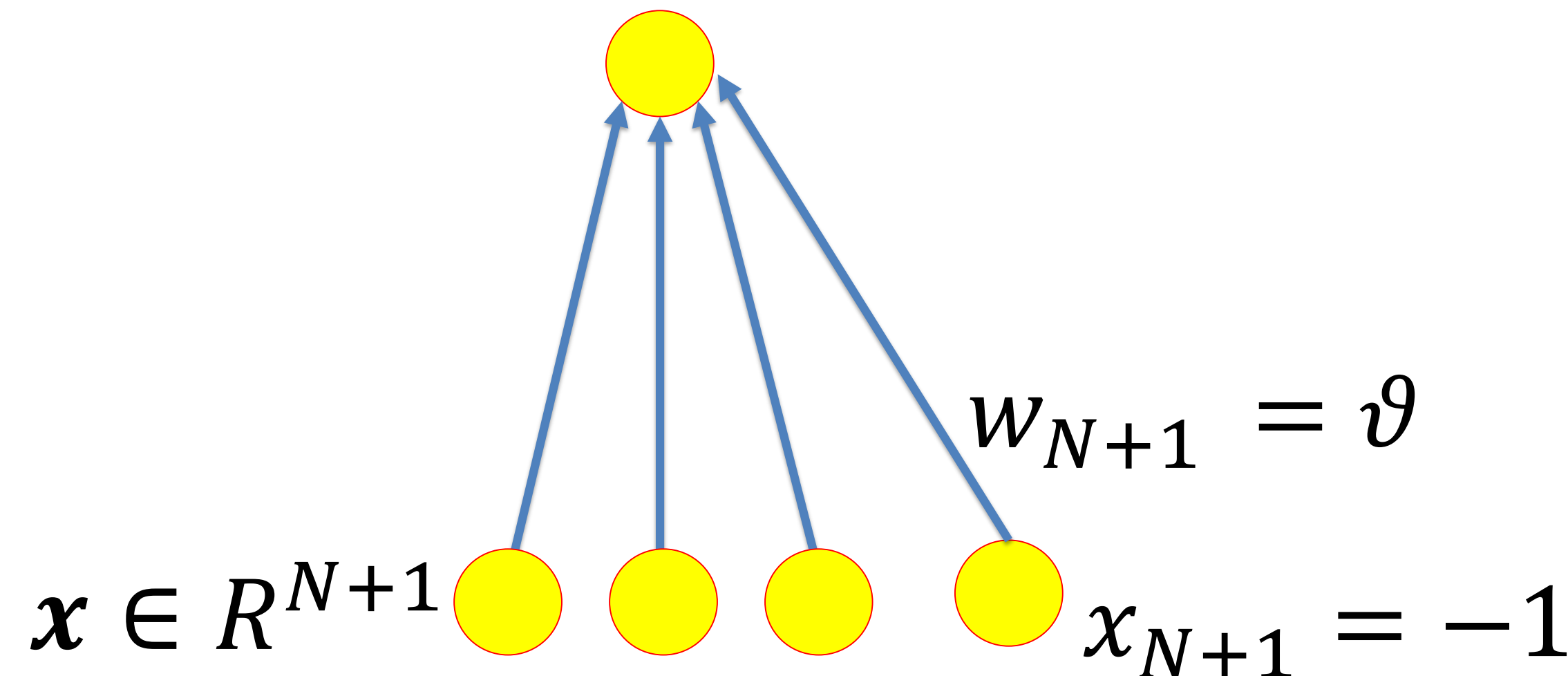
(normal gradient descent)

Online rule:

one update after one pattern

(stochastic gradient descent)

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



Artificial Neural Networks: Lecture 2

Backprop and multilayer perceptrons

Wulfram Gerstner
EPFL, Lausanne, Switzerland

1. Modern Gradient Descent Methods

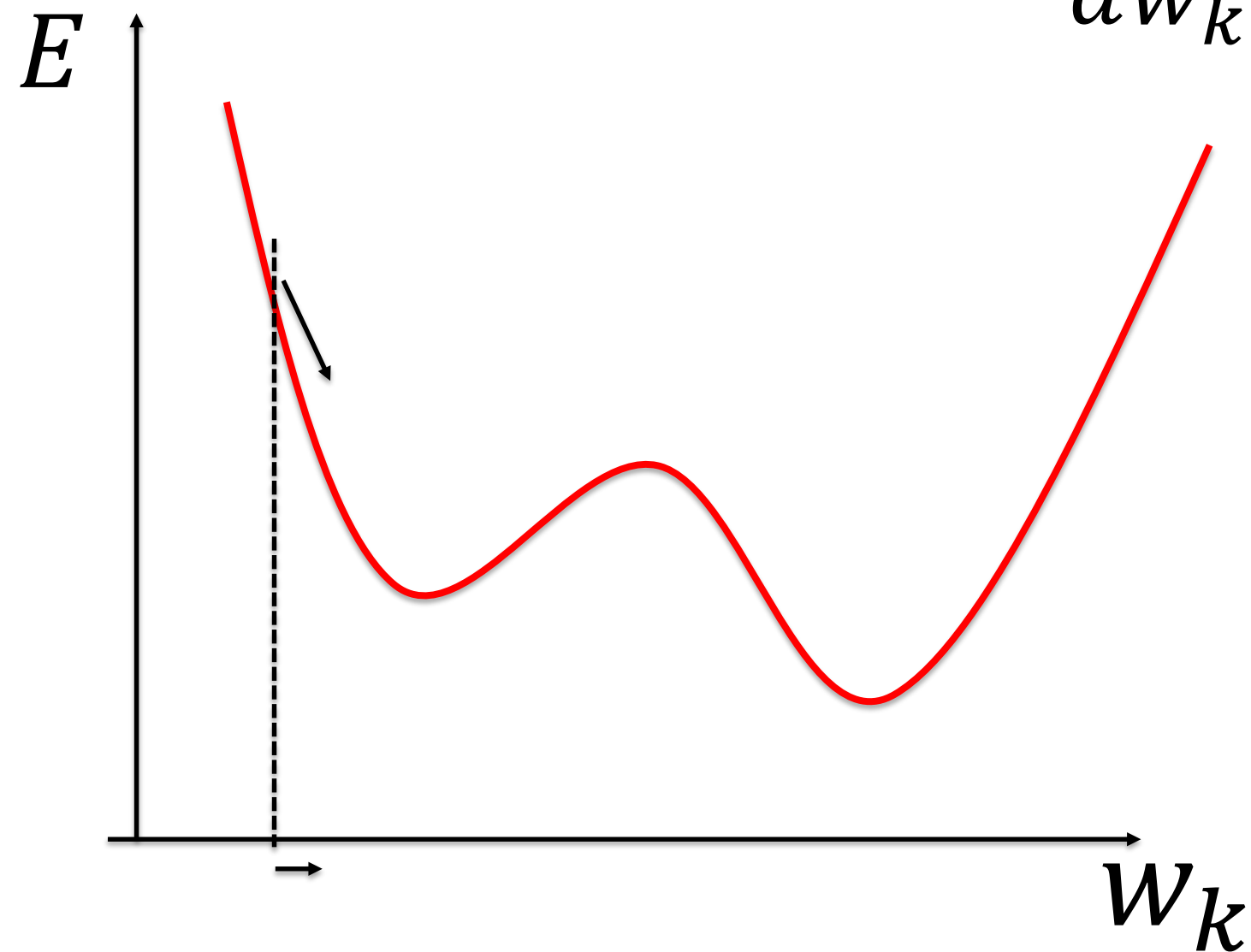
Modern gradient descent

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 **N** patterns

(minibatch update)

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

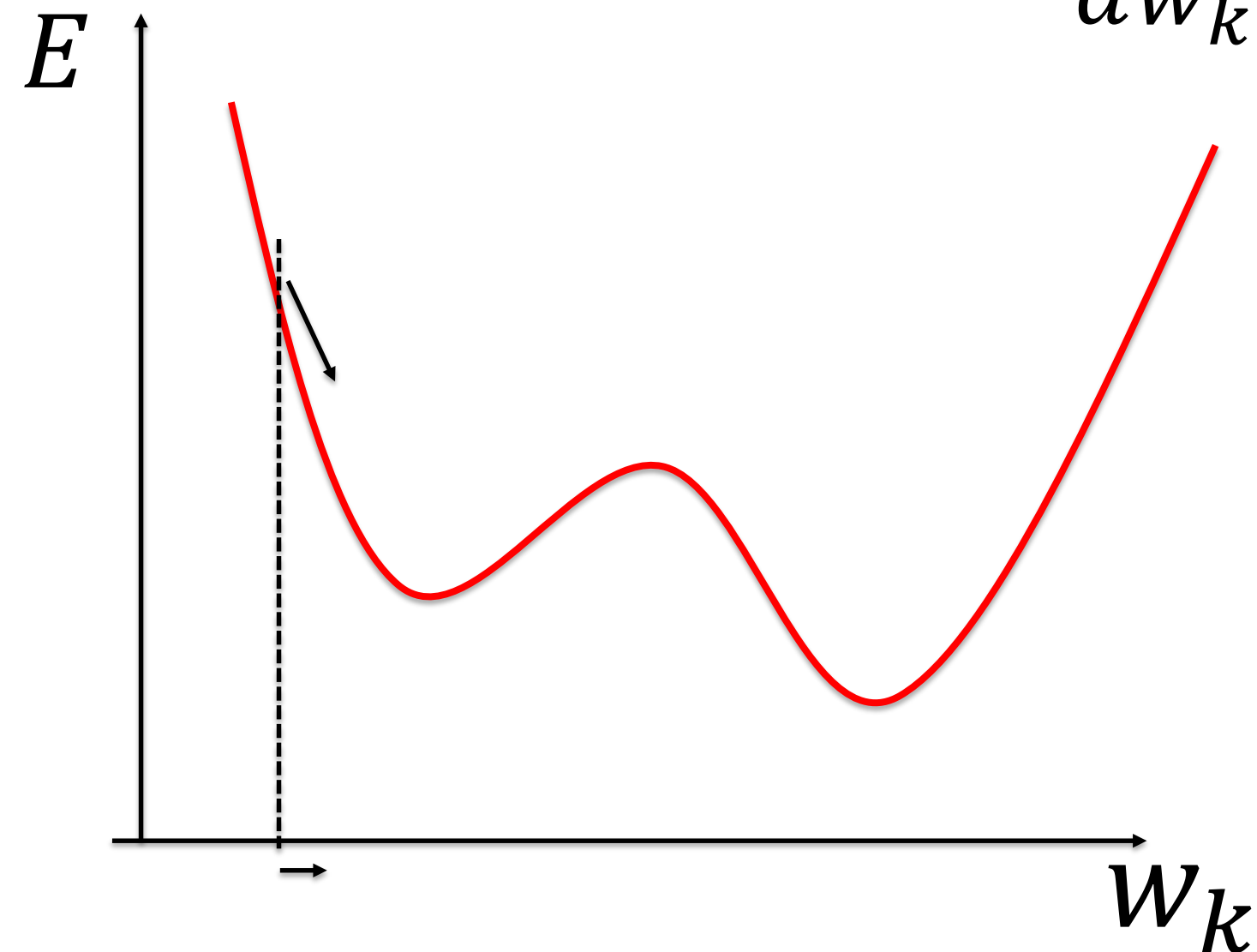
Modern gradient descent

Some **error function**, also called **loss function**

$E(\mathbf{w})$

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

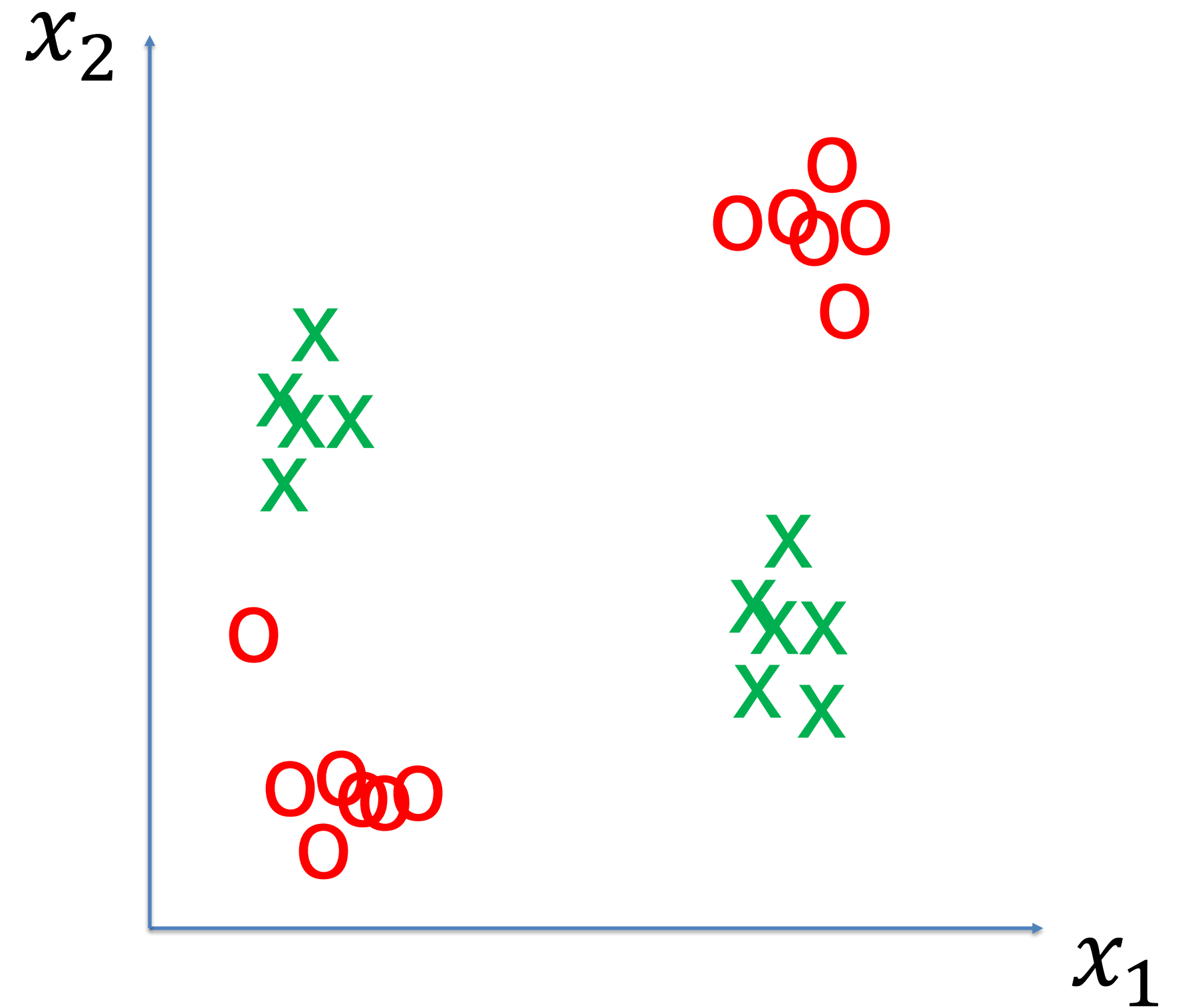
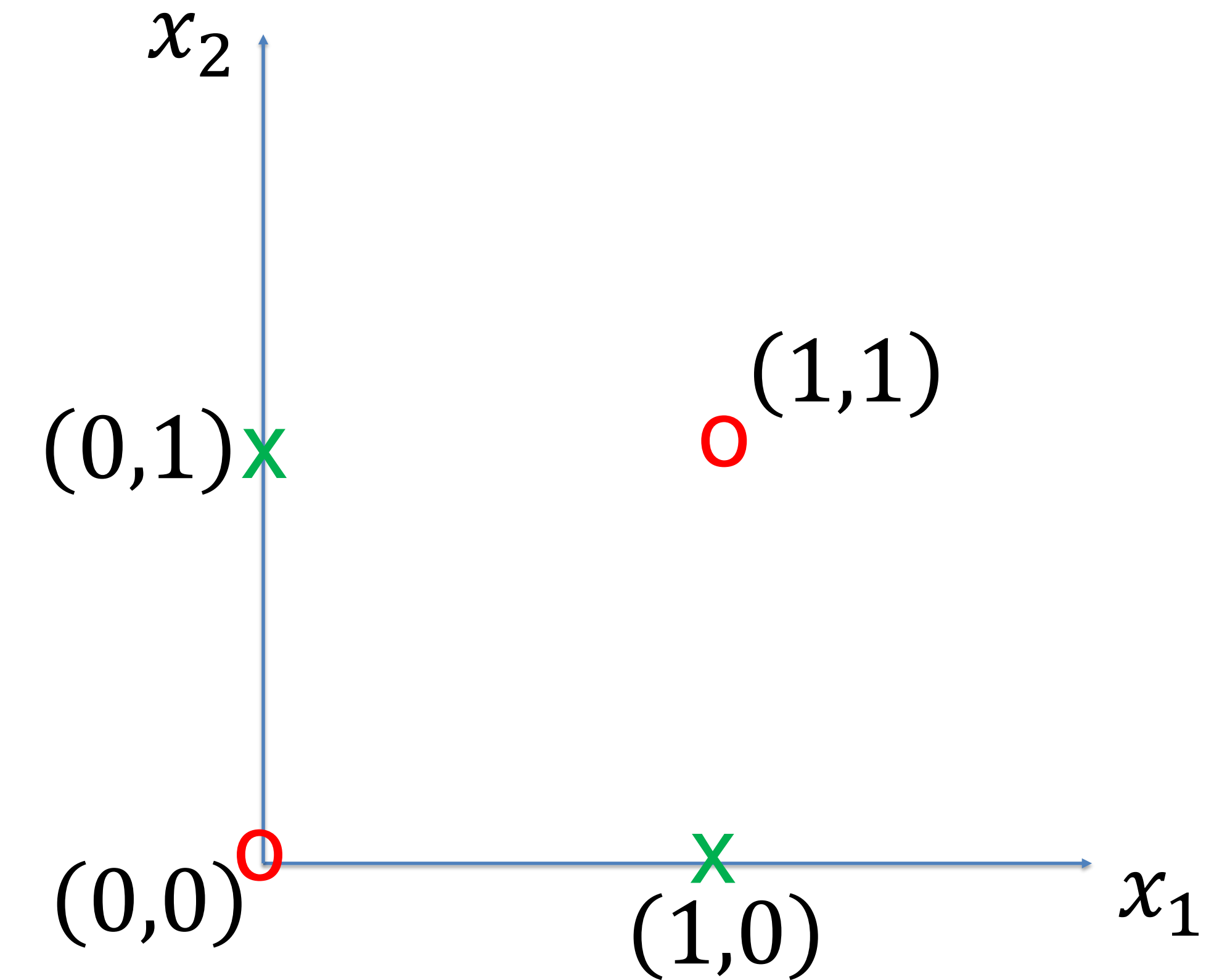
1. Modern Gradient Descent Methods
2. **XOR problem**

2. The XOR problem

just 4 data points

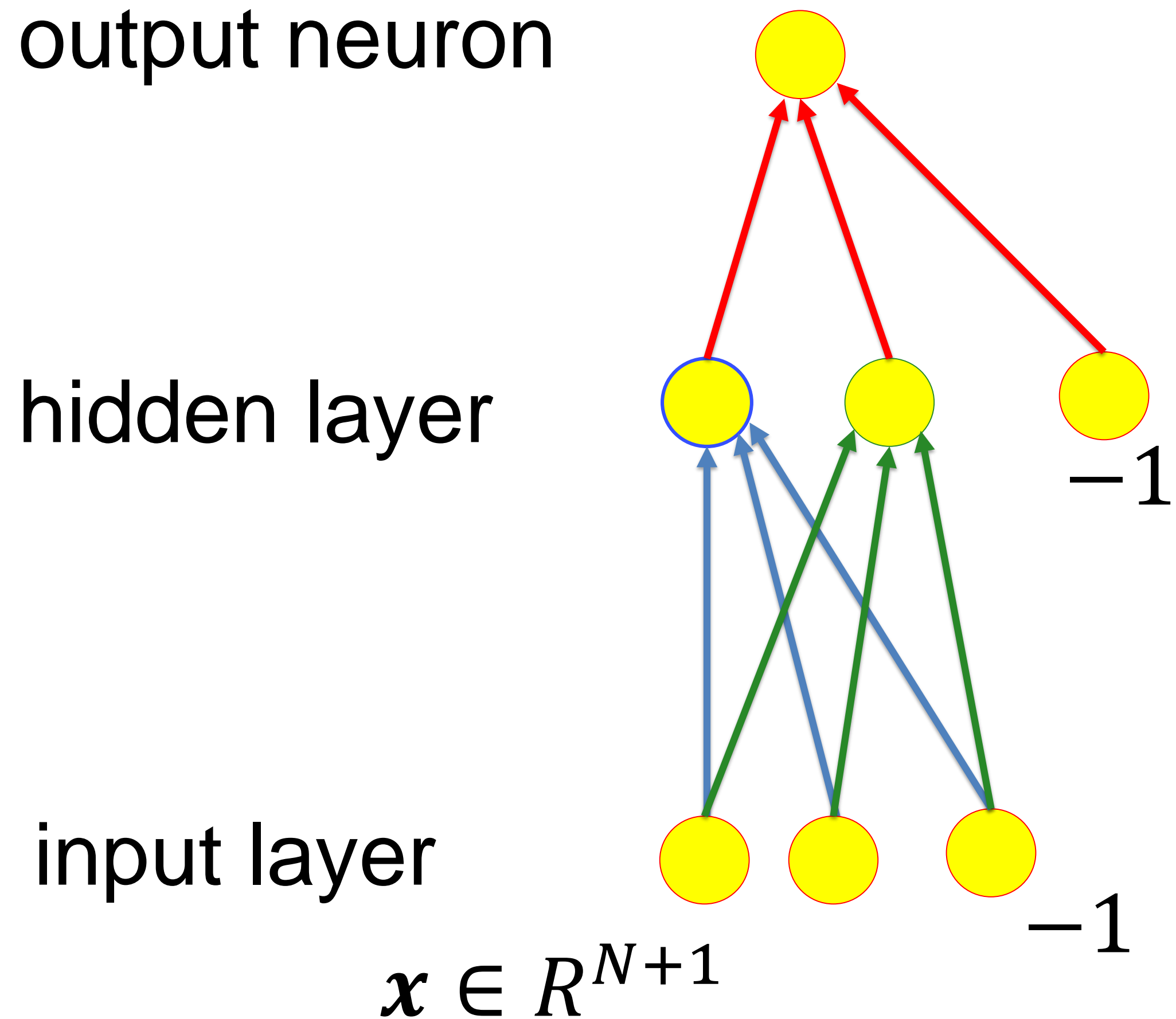
(or many)

Blackboard 1:
solution of XOR



Blackboard 1:
solution of XOR

2. Solution of XOR problem



Artificial Neural Networks: Lecture 2

Backprop and multilayer perceptrons

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

3. Multi-layer perceptron

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

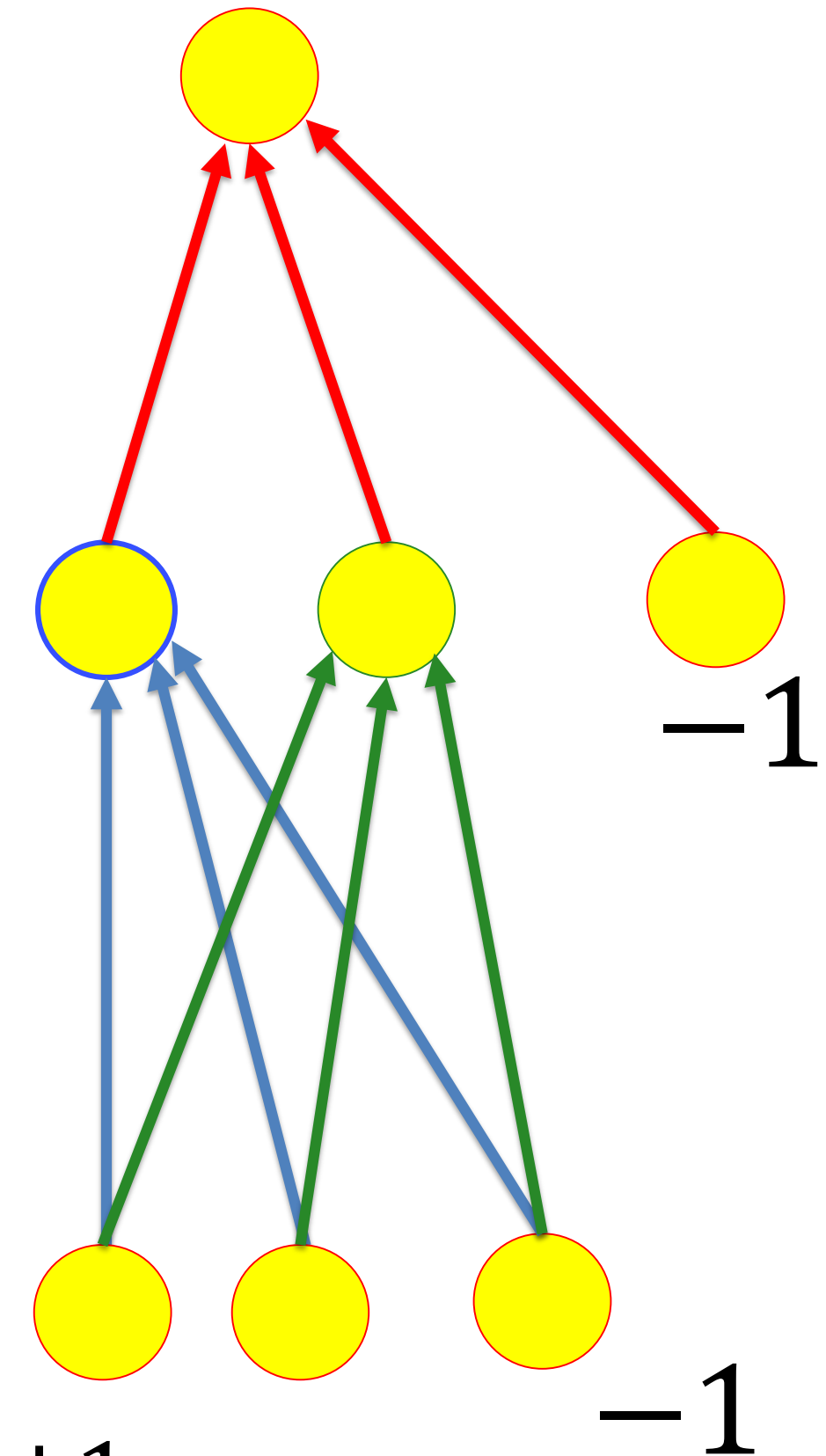
- But is there an **algorithm to find the weights** in more complicated cases?

output neuron

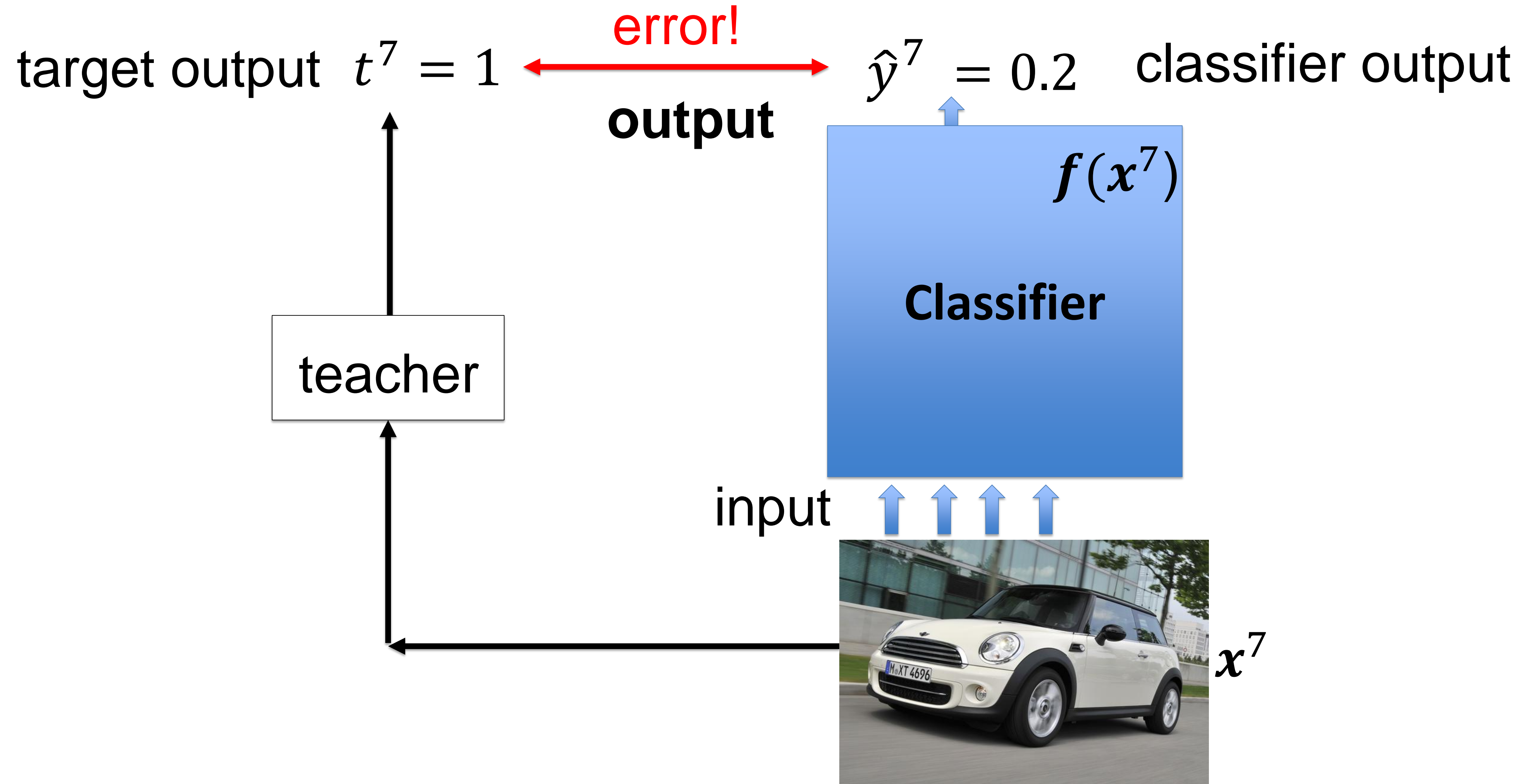
hidden layer

input layer

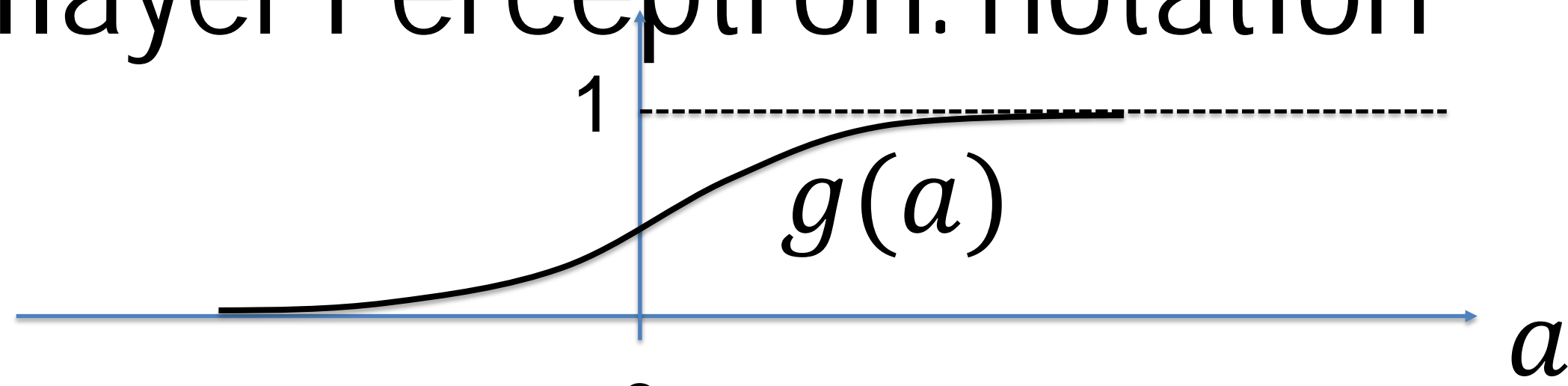
$$x \in R^{N+1}$$



3. Supervised learning with sigmoidal output



3. Multilayer Perceptron: notation



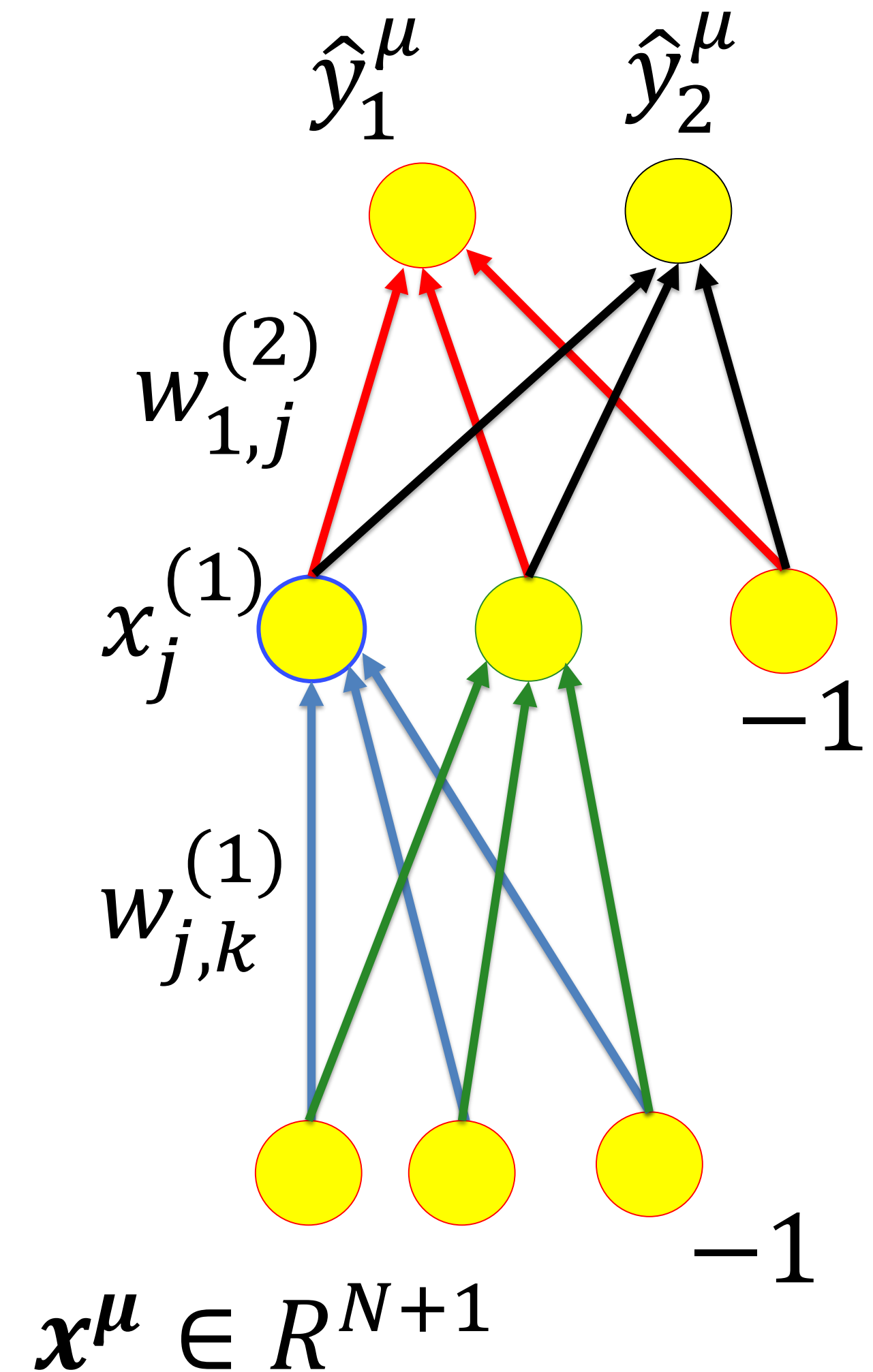
$$\hat{y}_i^\mu = x_i^{(2)} \quad (1)$$

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

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

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

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



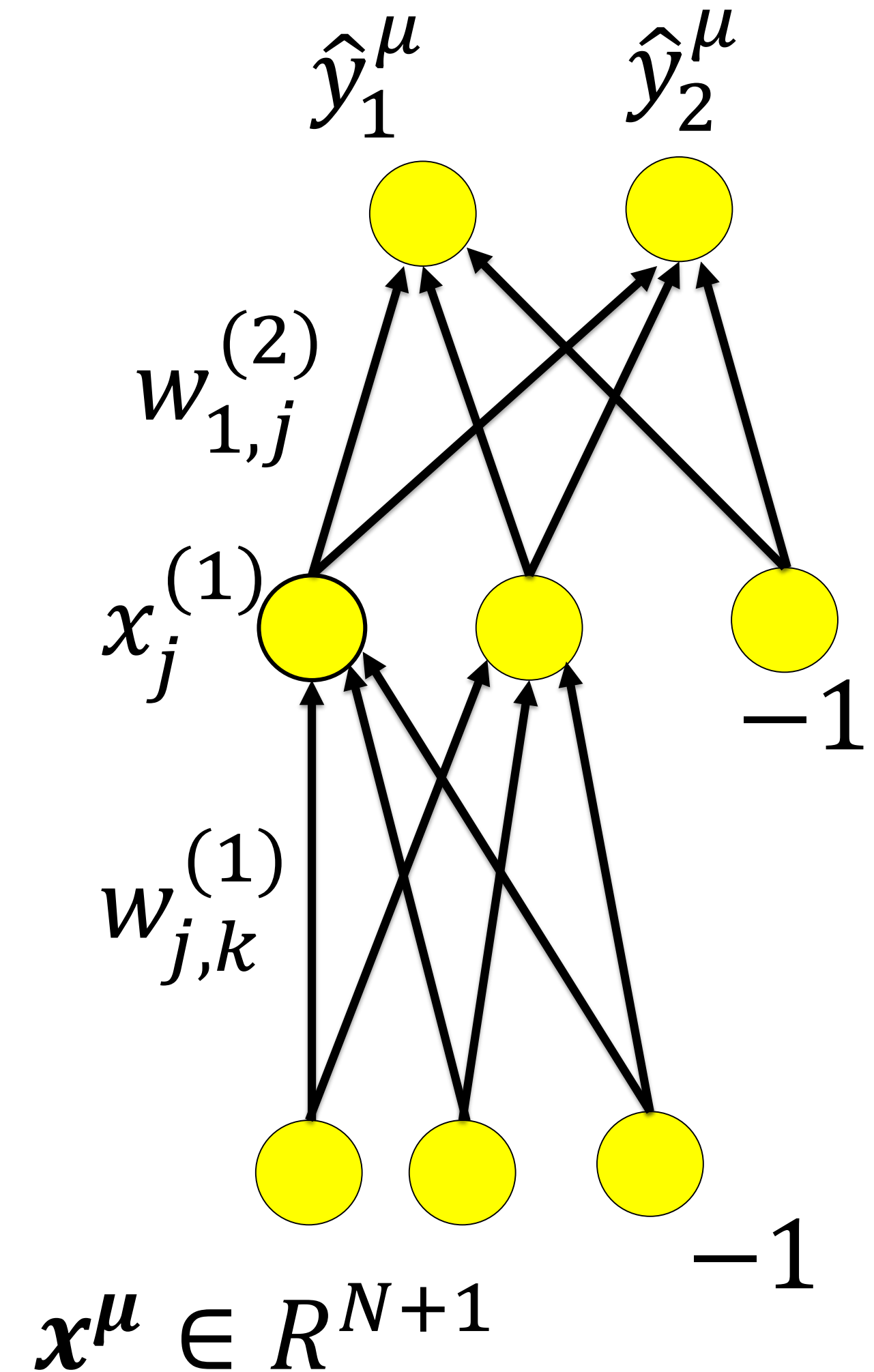
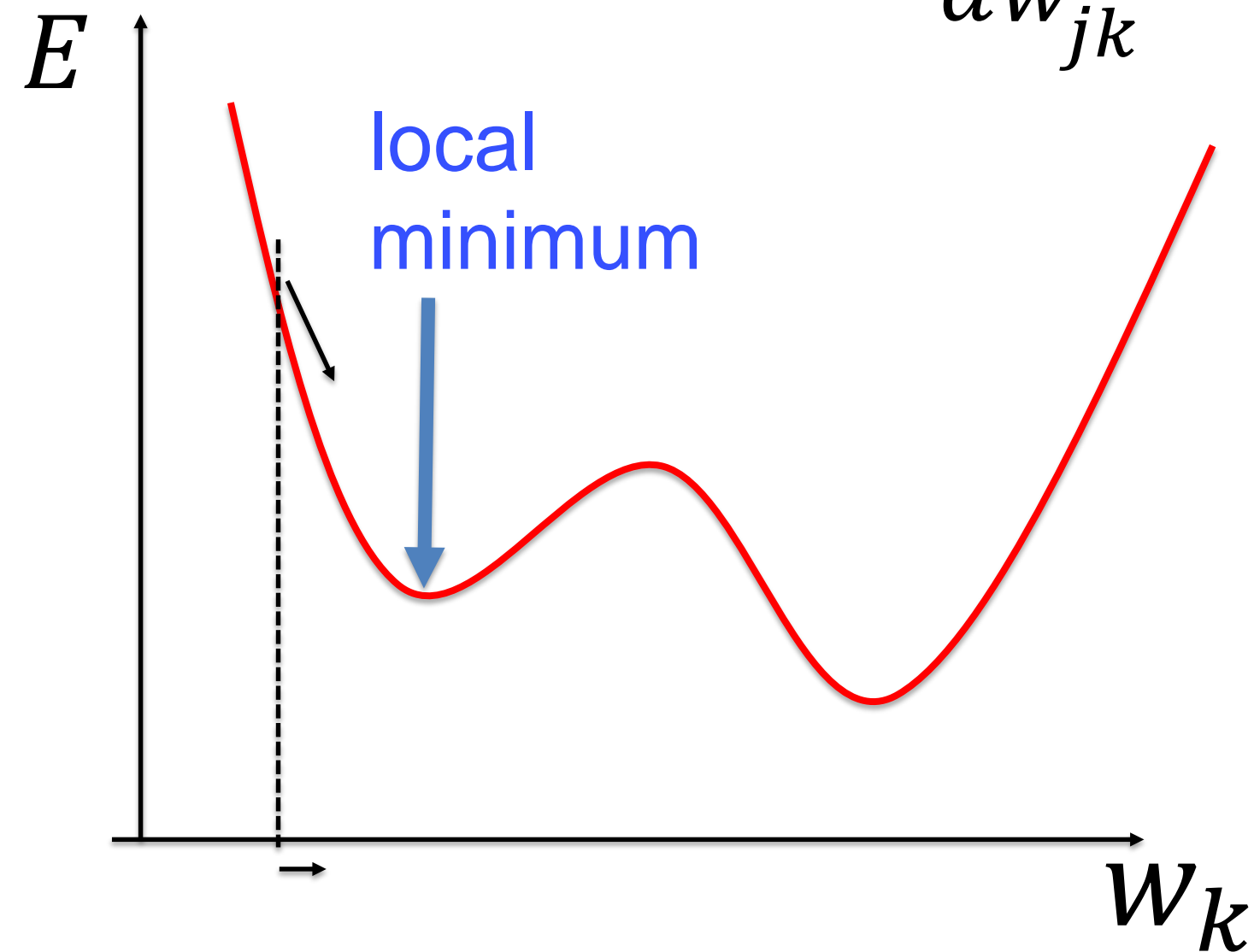
3. Multilayer Perceptron: gradient descent

Quadratic error

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

gradient descent

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



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

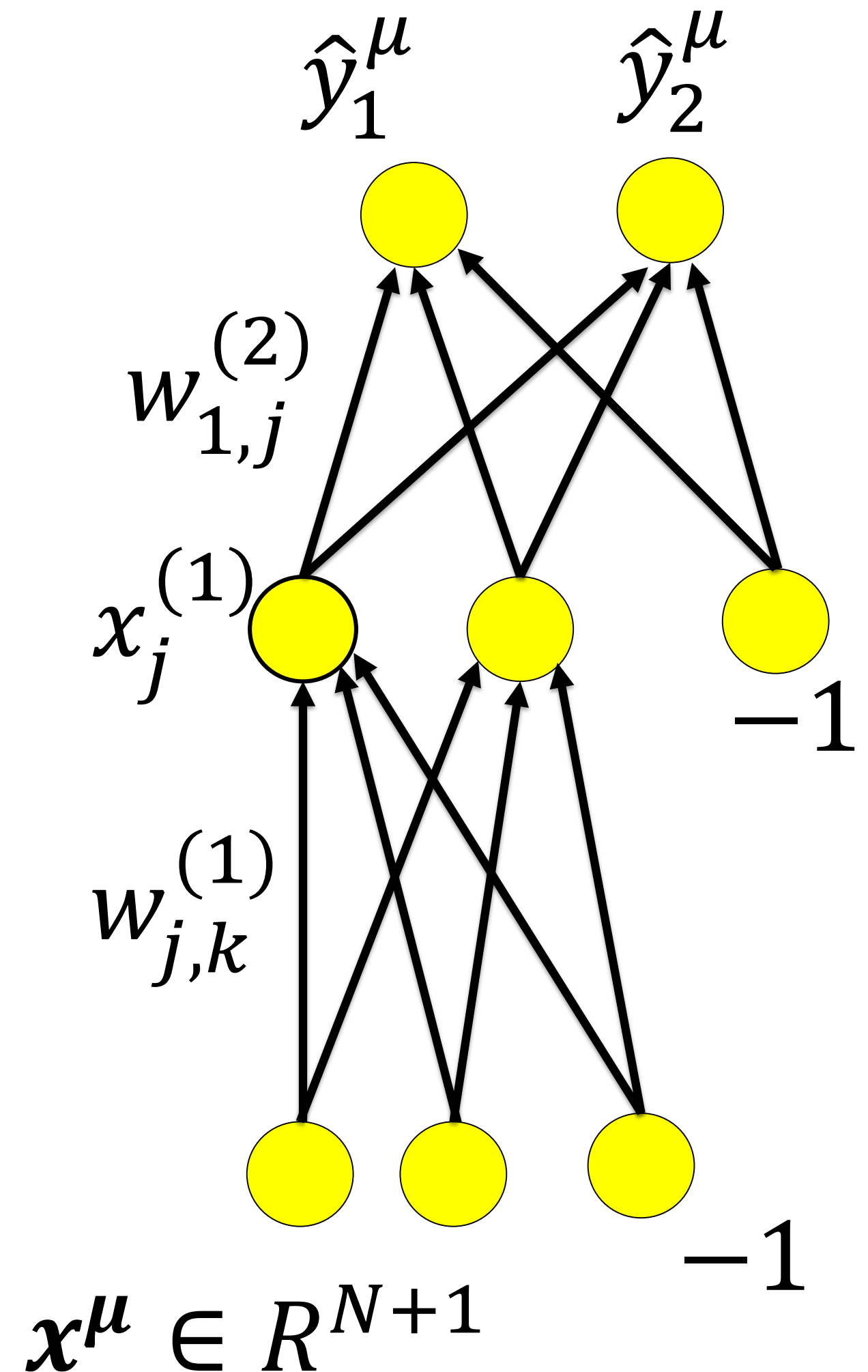
We continue in **8 minutes!**

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

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

with

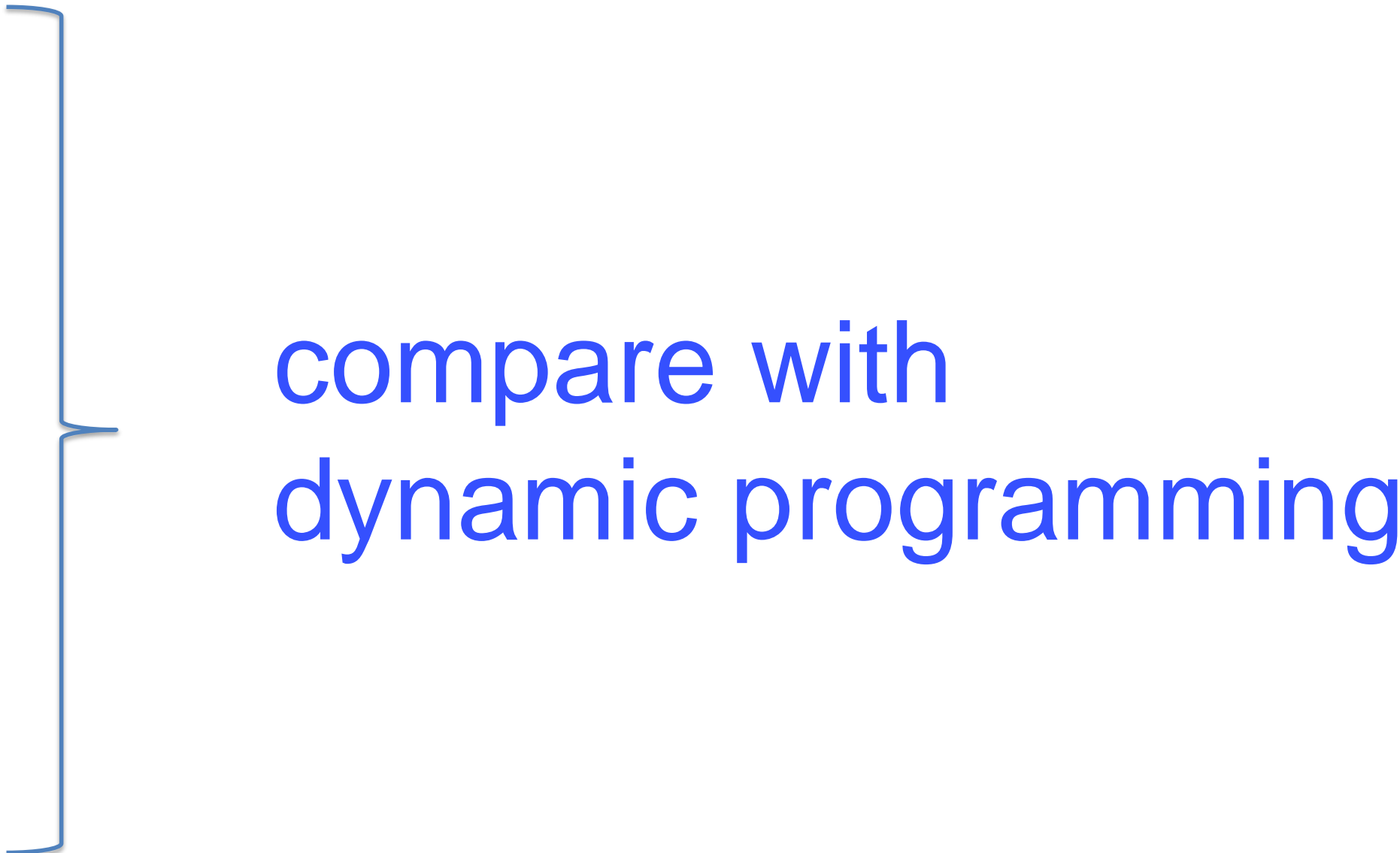
$$\hat{y}_i^\mu = g^{(2)} \left[\sum_j w_{ij}^{(2)} g^{(1)} \left(\sum_k w_{jk}^{(1)} x_k^\mu \right) \right]$$



Blackboard 2:
calculate gradient

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

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

BackProp

0. Initialization of weights

1. Choose pattern \mathbf{x}^μ

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

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

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

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

3. Computation of errors in output

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

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

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

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

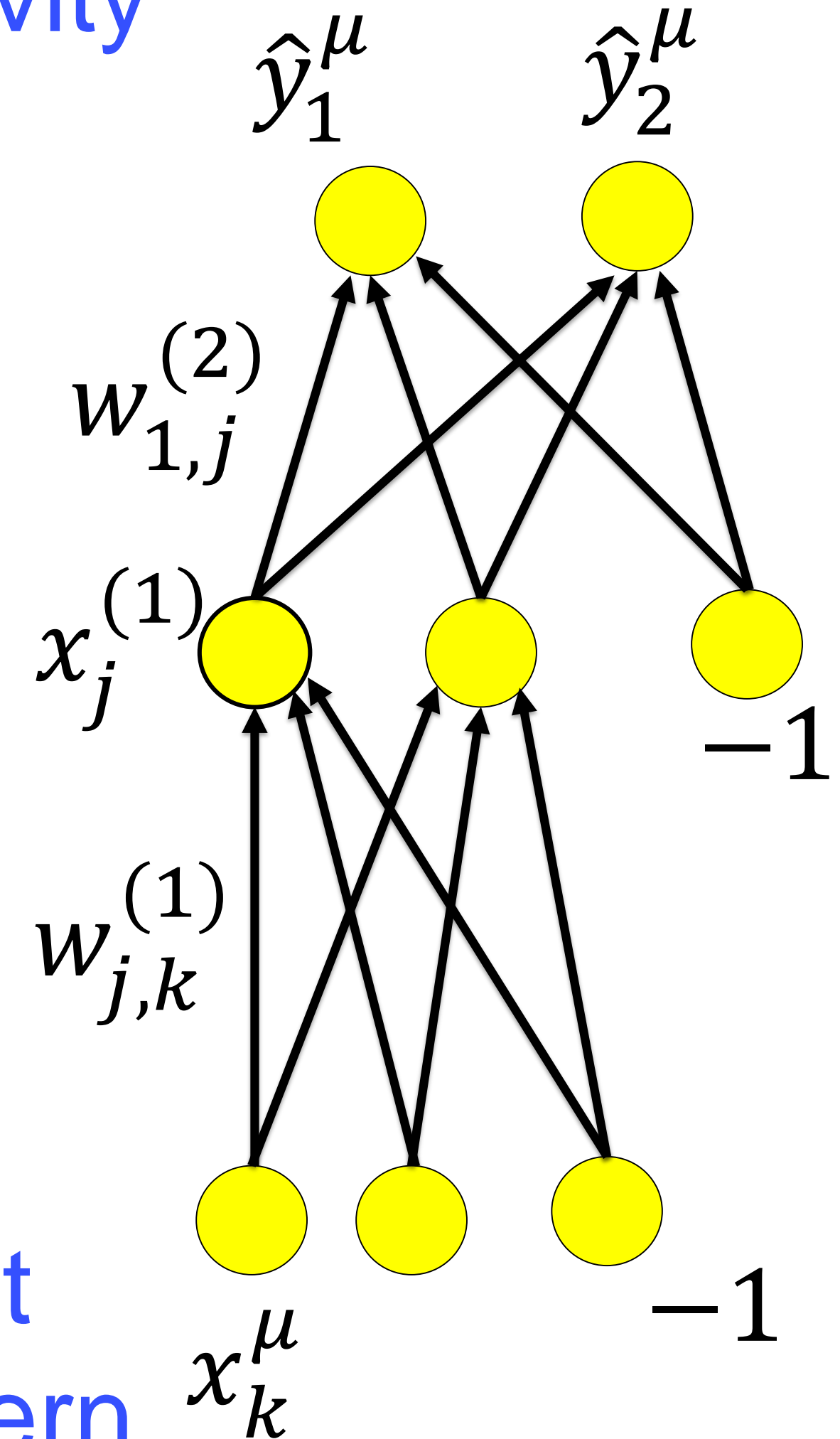
$$\Delta w_{ij}^{(n)} = \eta \delta_i^{(n)} x_j^{(n-1)} \quad (4)$$

6. Return to step 1.

output
activity



input
pattern



BackProp

0. Initialization of weights

1. Choose pattern \mathbf{x}^μ

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

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

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

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

3. Computation of errors in output

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

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

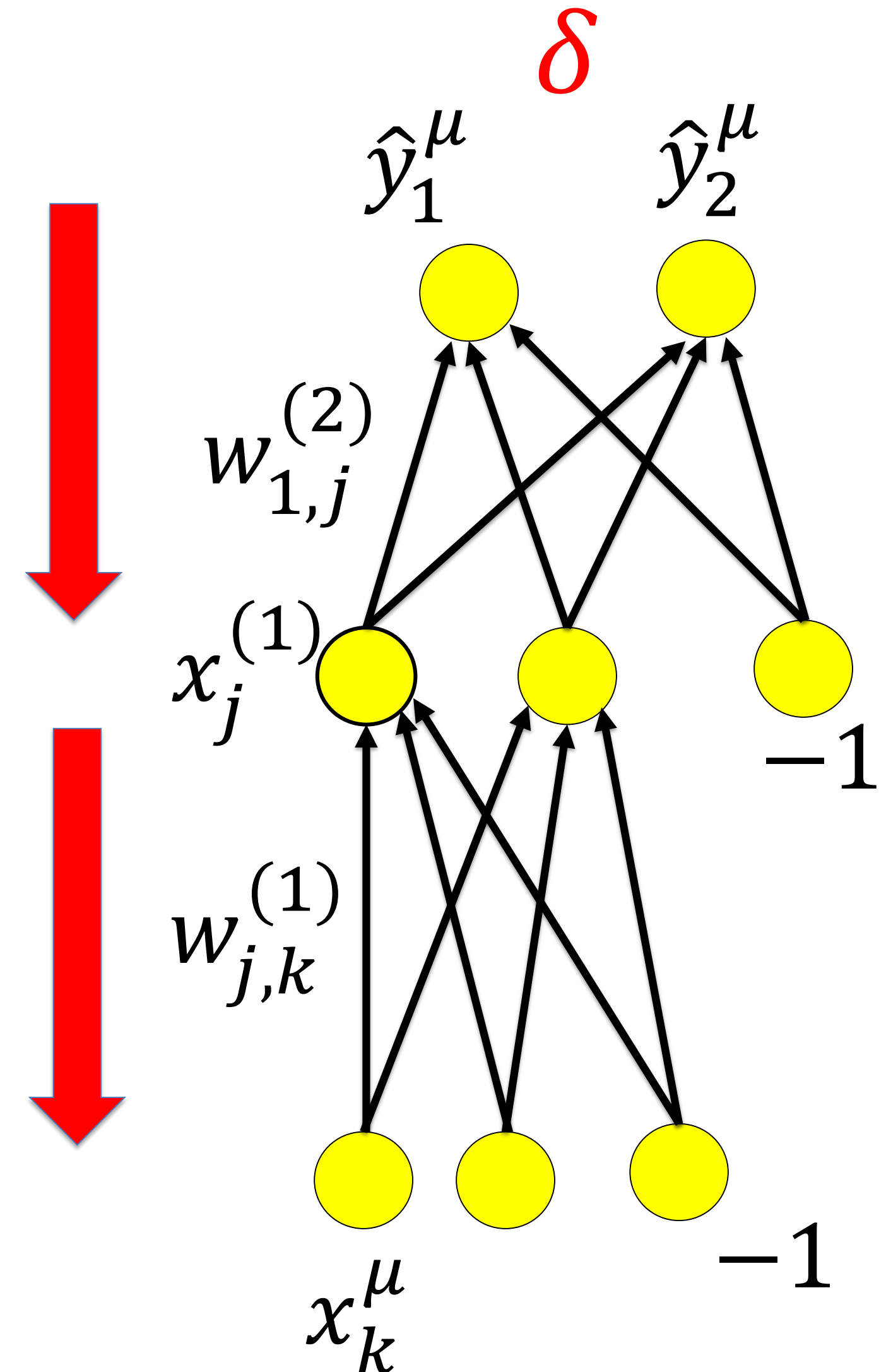
$$\delta_j^{(n-1)} = g'^{(n-1)}(a_j^{(n-1)}) \sum_i w_{ij} \delta_i^{(n)} \quad (3)$$

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

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

6. Return to step 1.

Calculate output error



BackProp

0. Initialization of weights

1. Choose pattern \mathbf{x}^μ

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

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

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

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

3. Computation of errors in output

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

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

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

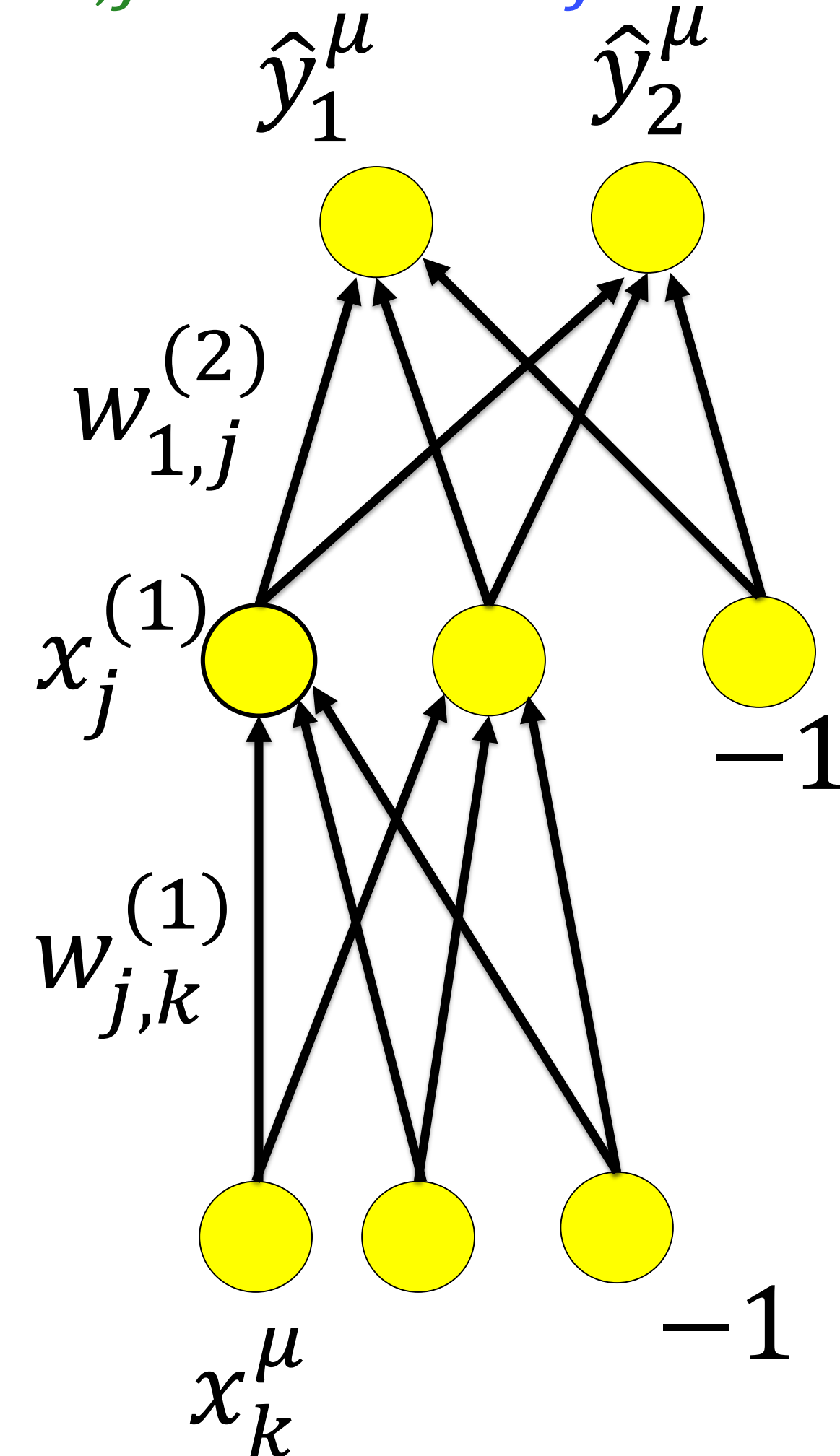
5. Update weights (for each (i, j) and all layers (n))

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

6. Return to step 1.

update all weights

$$w_{i,j}^{(n)} = \delta_i^{(n)} x_j^{(n-1)}$$



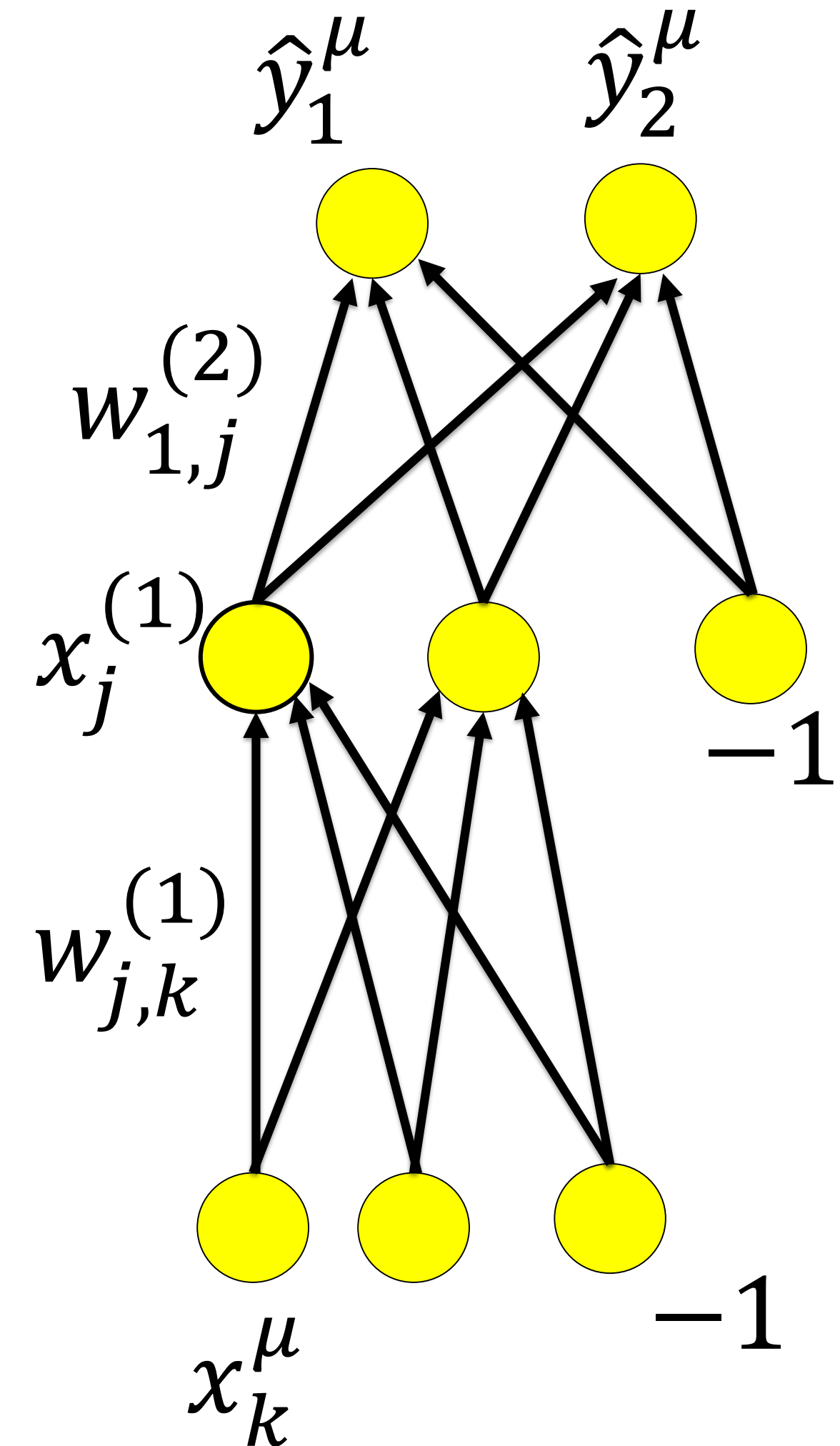
4. Backprop versus direct numerical evaluation

$$\begin{aligned}\Delta w_{jk}^{(1)} &= -\gamma \frac{dE}{dw_{jk}^{(1)}} \\ &= -\gamma \frac{E(w_{jk}^{(1)} + \varepsilon) - E(w_{jk}^{(1)} - \varepsilon)}{2\varepsilon}\end{aligned}$$

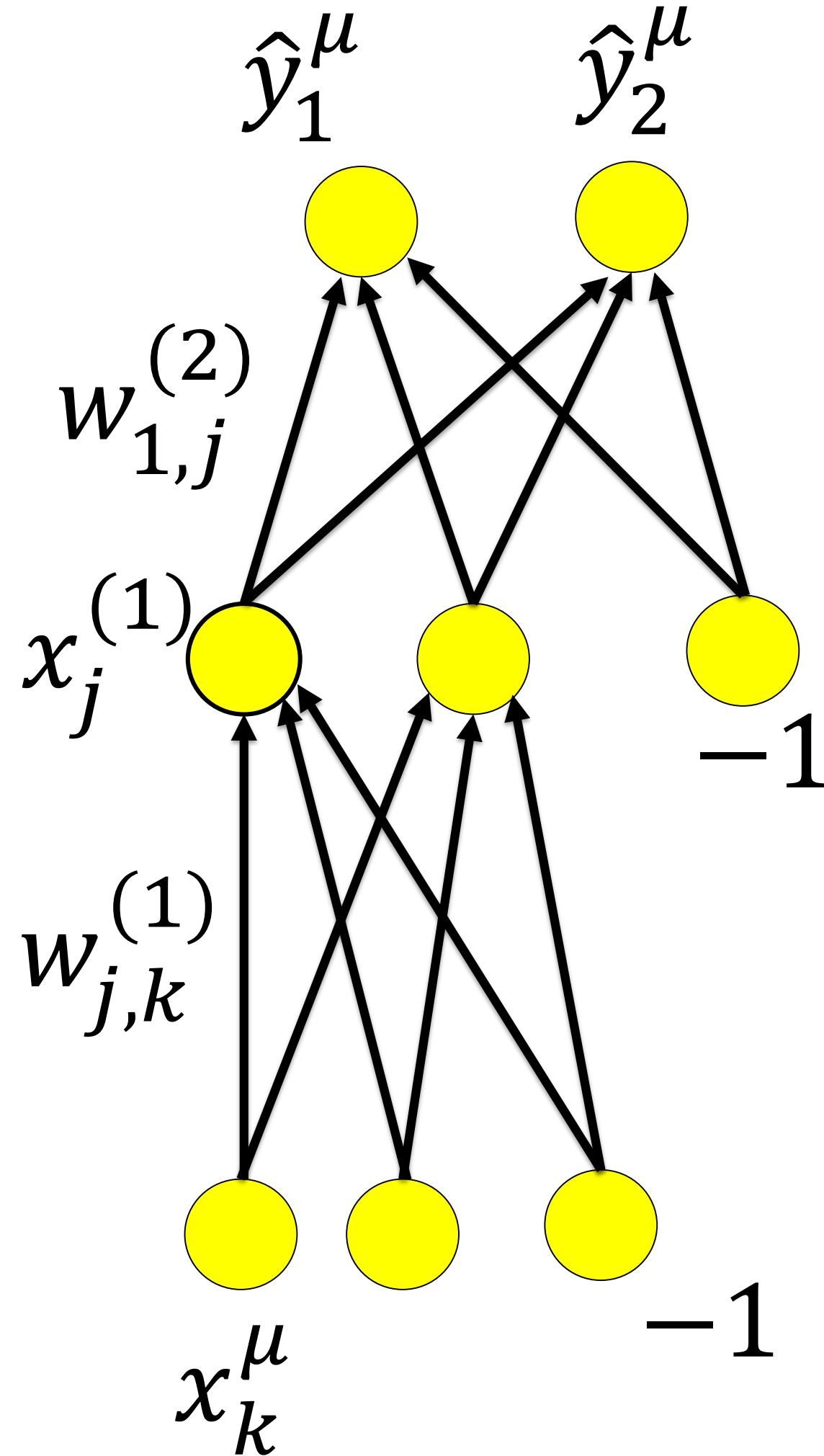
calculate $E(\mathbf{w}) = \frac{1}{2} \sum_{\mu=1}^P \sum_i [t_i^\mu - \hat{y}_i^\mu]^2$

→ calculate \hat{y}_i^μ for one pattern

Blackboard 3:
algorithmic complexity



Blackboard 3:
algorithmic complexity



$$\frac{E(w_{jk}^{(1)} + \varepsilon) - E(w_{jk}^{(1)} - \varepsilon)}{2\varepsilon}$$

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

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

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

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

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

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

4. Direct numerical evaluation: complexity

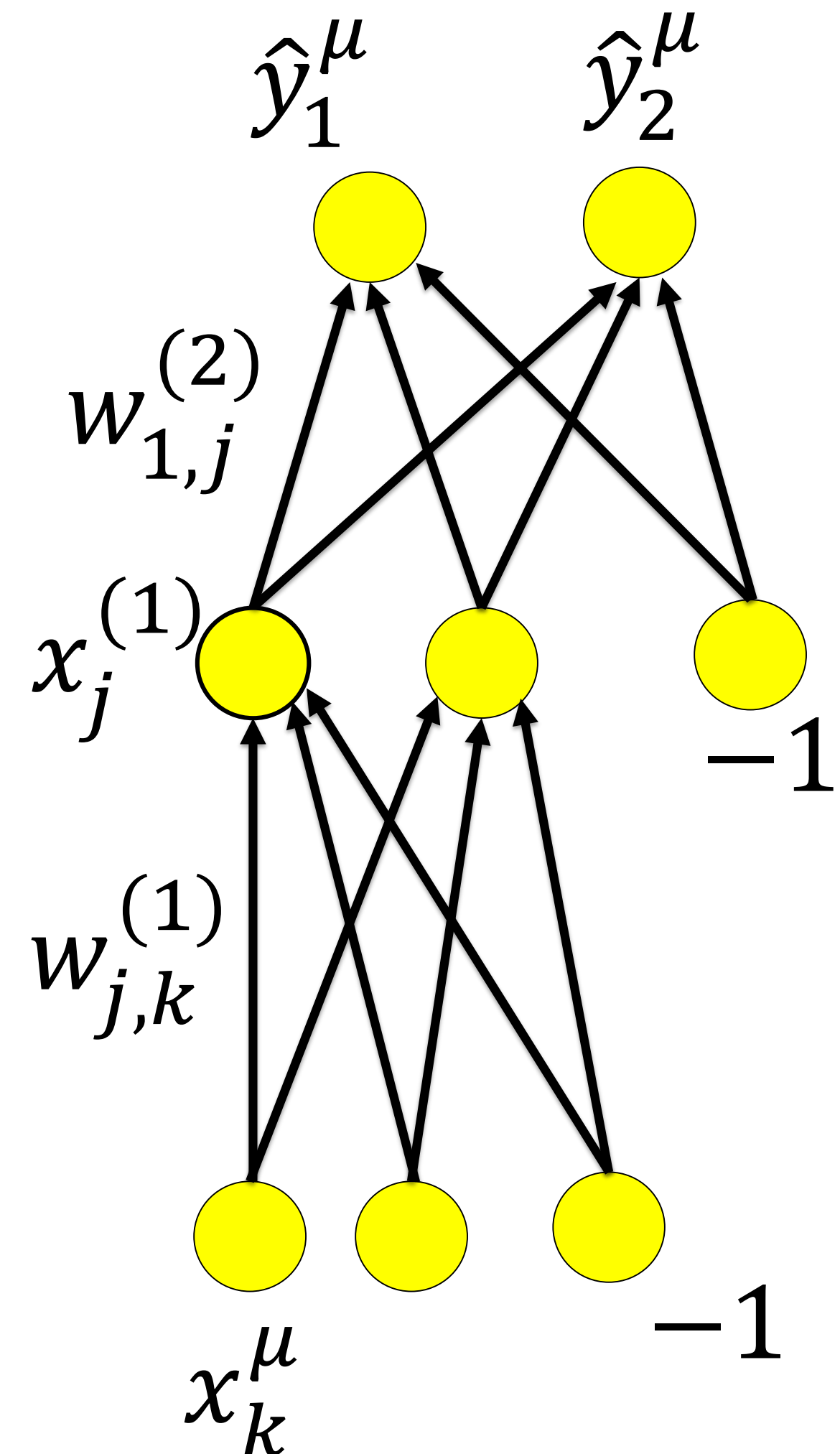
calculate $E(\mathbf{w}) = \frac{1}{2} \sum_{\mu=1}^P \sum_i [t_i^\mu - \hat{y}_i^\mu]^2$

1) calculate \hat{y}_i^μ for one pattern
→ each weight is touched once

2) for each change of weight,
evaluate E twice

$$\Delta w_{jk}^{(1)} = -\gamma \frac{dE(w_{jk}^{(1)} + \varepsilon) - dE(w_{jk}^{(1)} - \varepsilon)}{2\varepsilon}$$

3) For n weights, order n -square!!!



3. Backprop: complexity

Exercise 2 at home: show algo is of order n

0. Initialization of weights

1. Choose pattern \mathbf{x}^μ

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

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

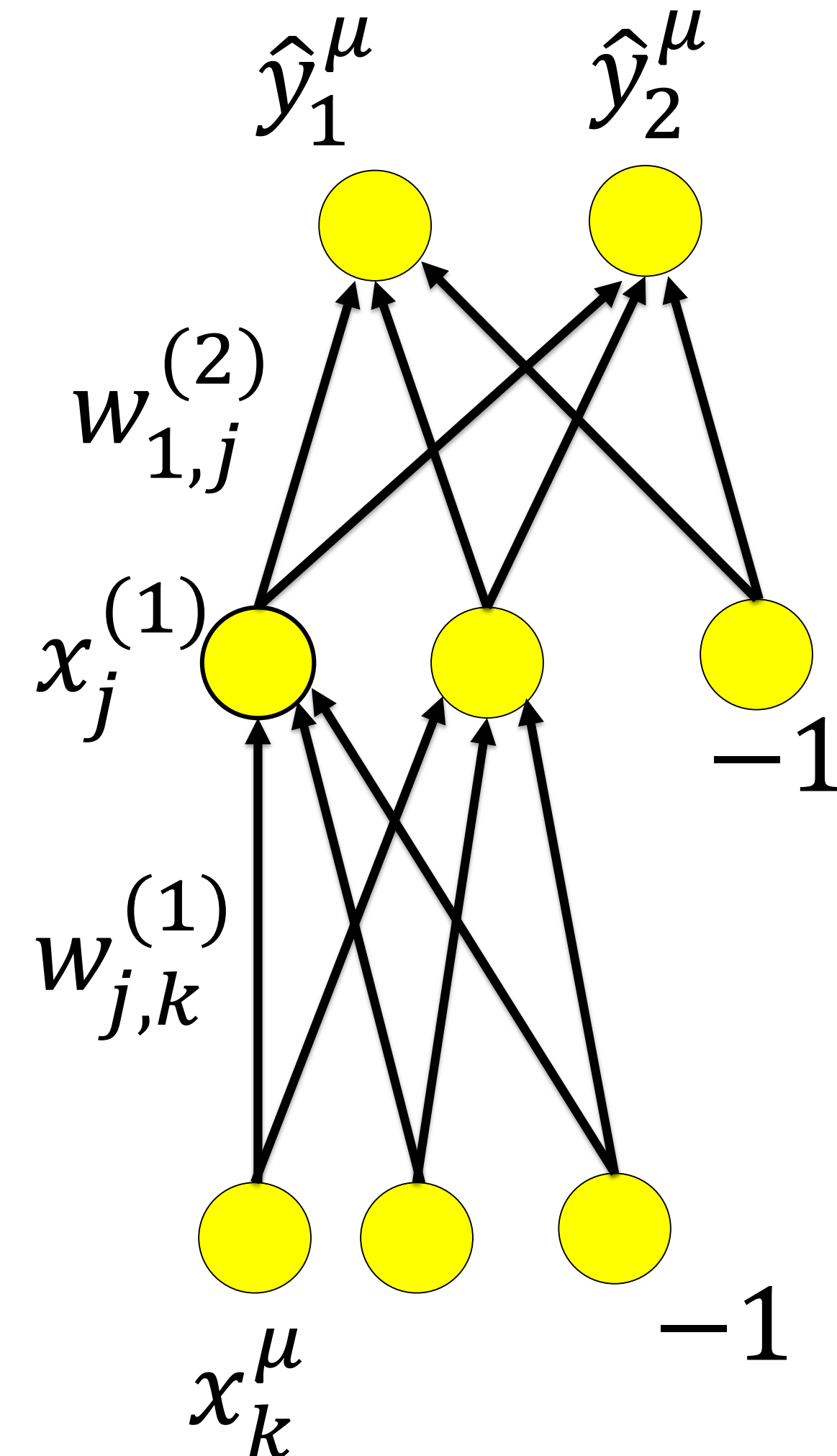
4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

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

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

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

6. Return to step 1.



4. Conclusions: Multilayer Perceptron and Backprop

- A multilayer Perceptron can solve the XOR problem
 - Hidden neurons increase the flexibility of the separating surface
 - Weights 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

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

5. The problem of overfitting

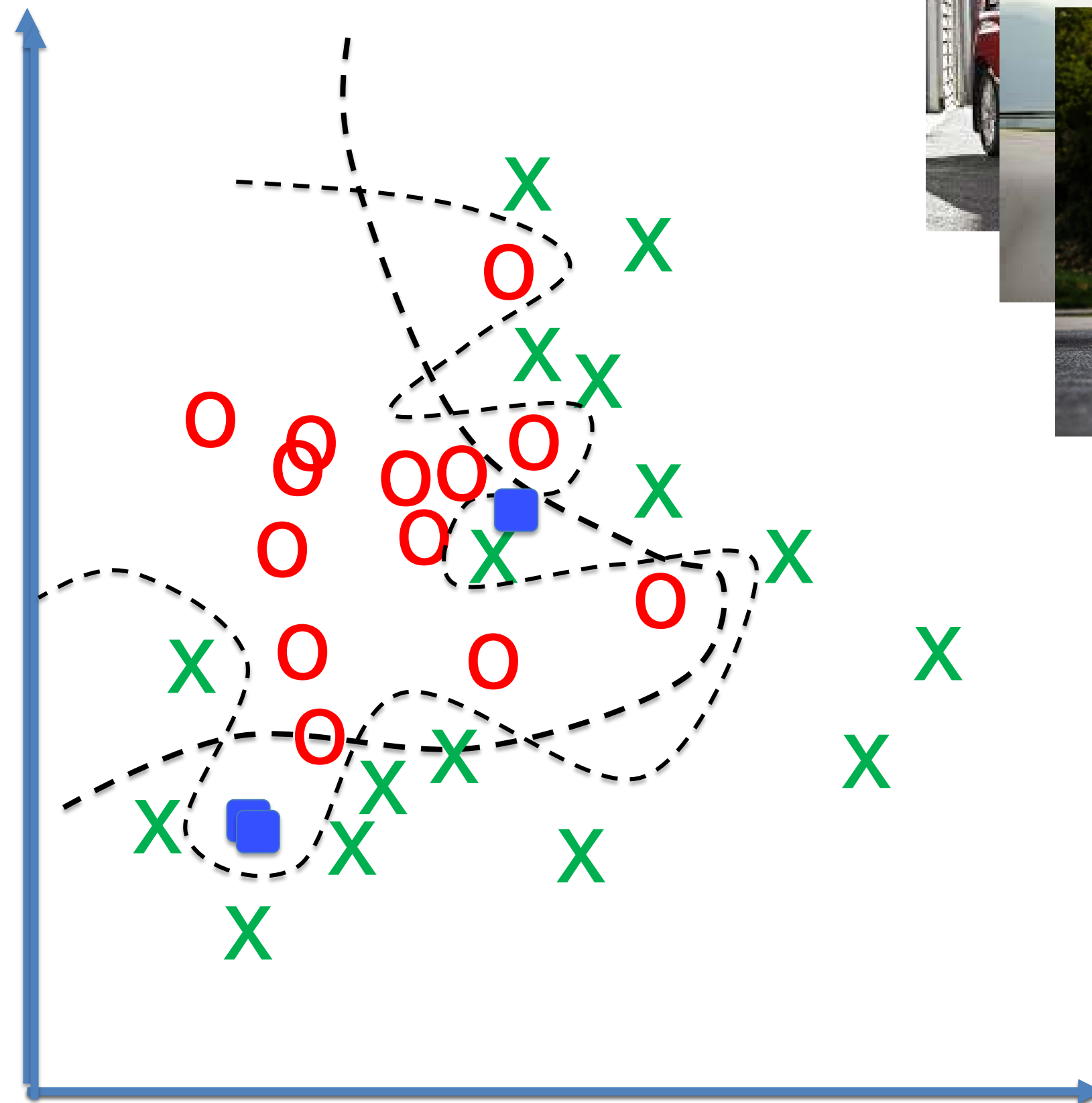
Big Multilayer perceptrons are flexible and can be trained by BackProp to minimize classification error

... but is flexibility always good?

5. Classification of new inputs

X = 'car'

o = 'not car'



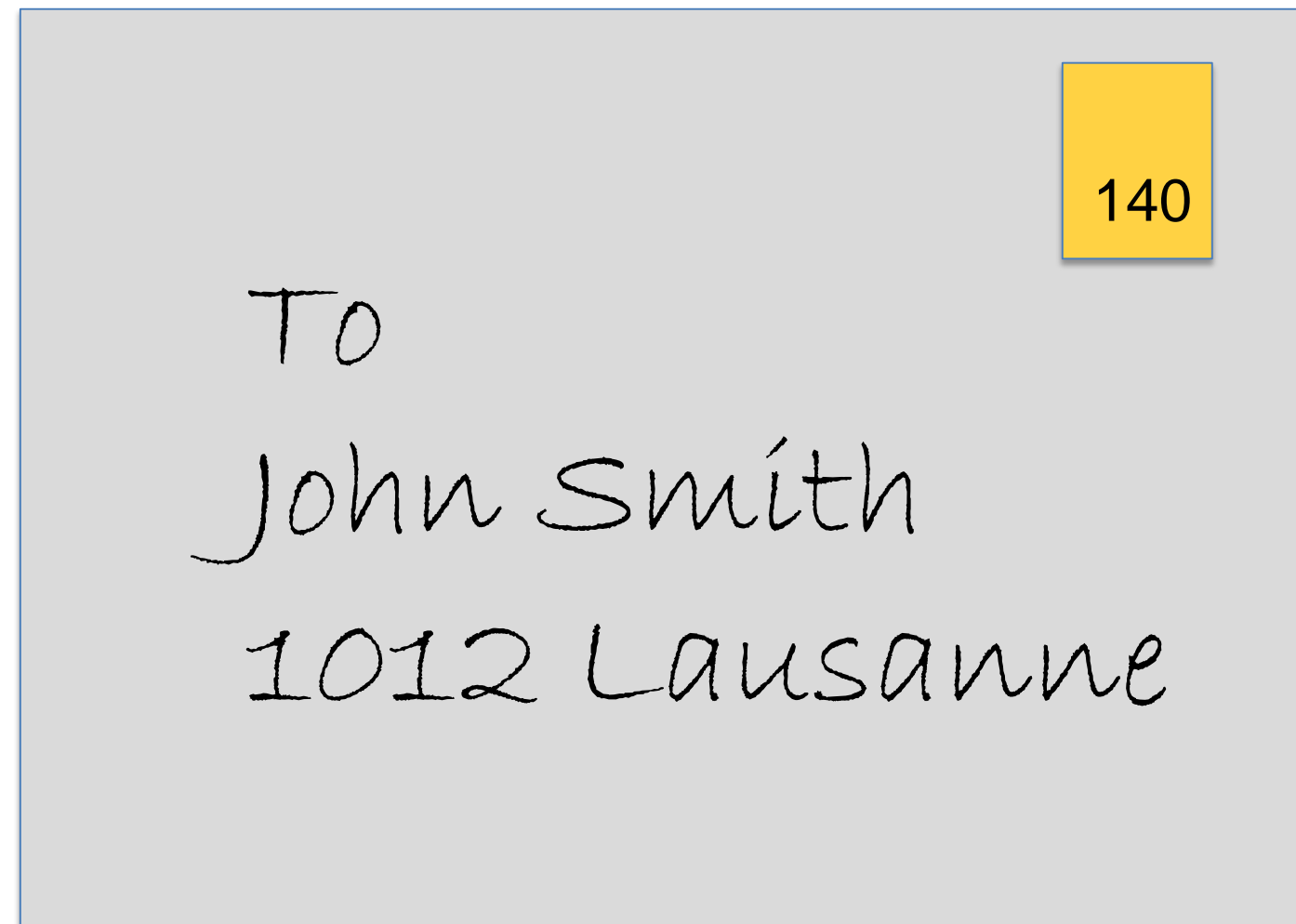
Aim: predict classification for **new** inputs, not seen during training

■ = 'new image'

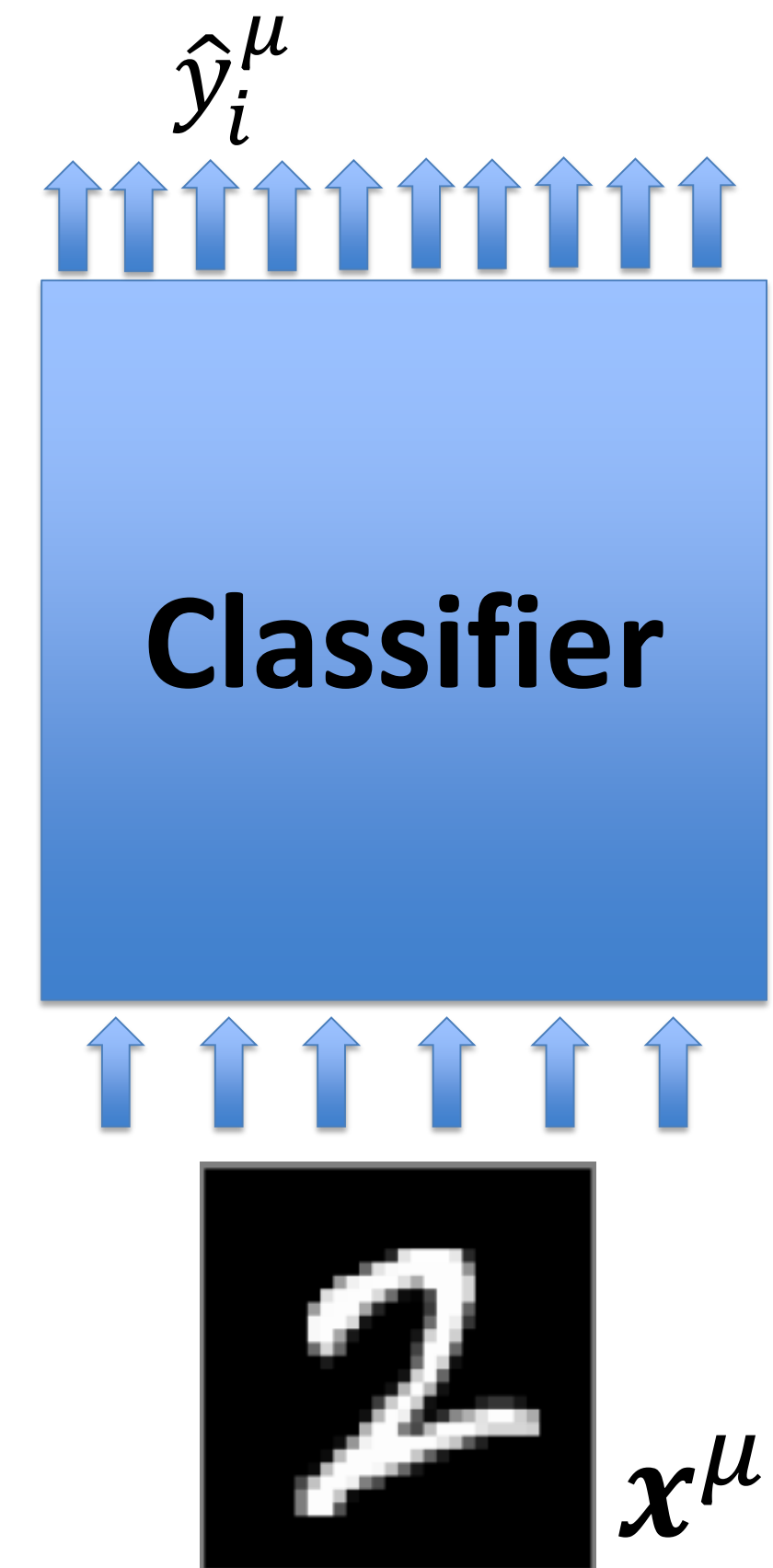


5. Classification of new inputs: Example

Task: Read Postal Code



10 output units



5. Classification of new inputs: Example

MNIST data samples

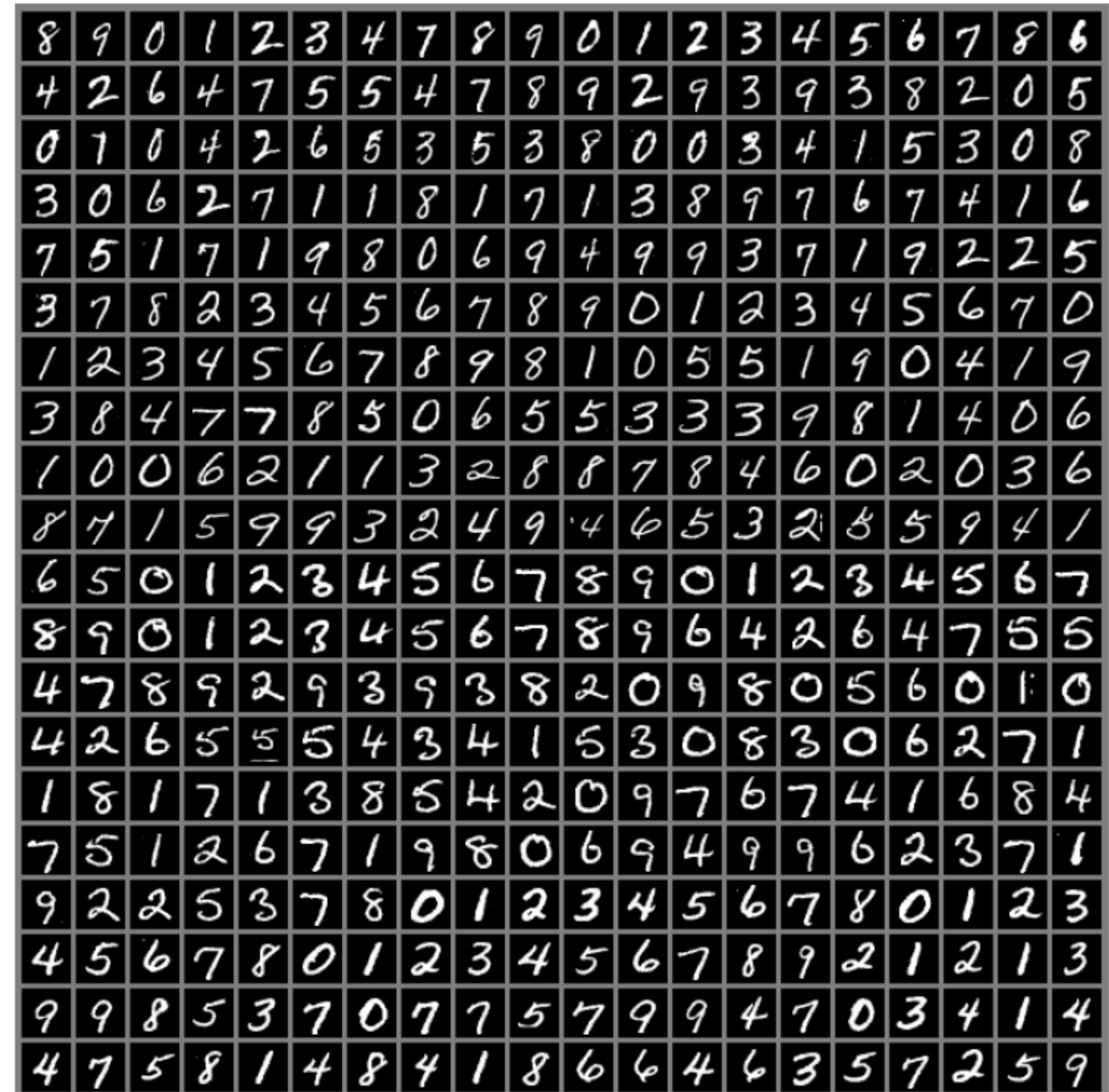


- images 28x28
- Labels: 0, ..., 9
- 250 writers
- 60 000 images in training set

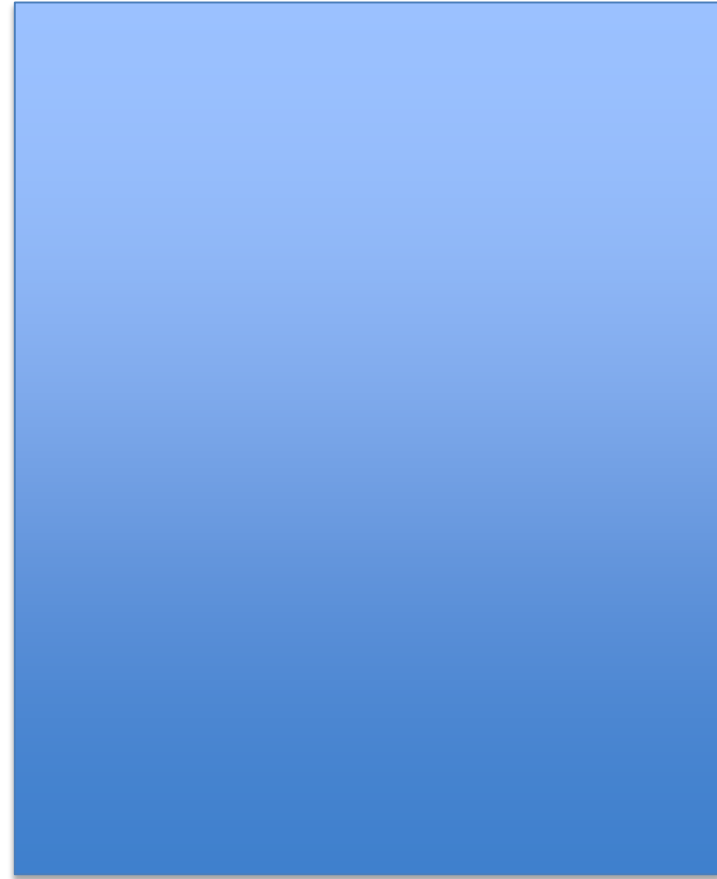
Picture: Goodfellow et al, 2016

Data base:

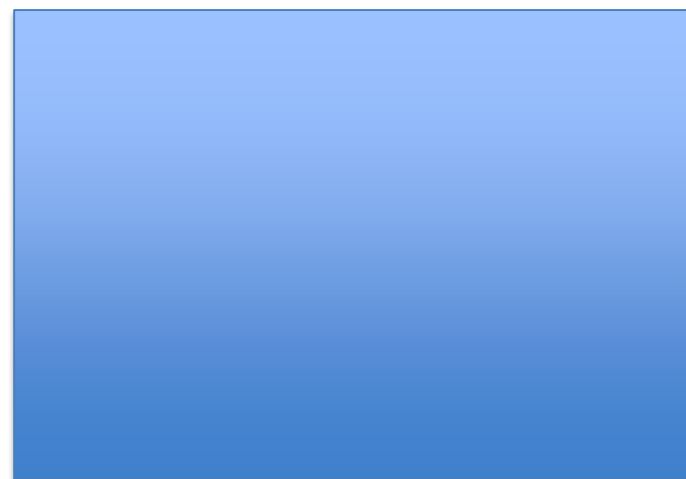
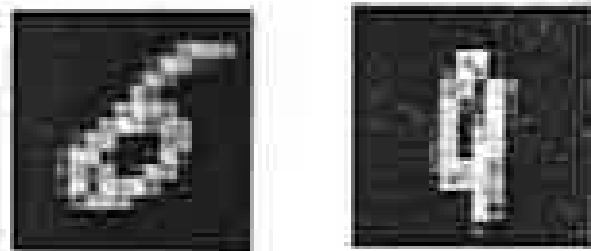
<http://yann.lecun.com/exdb/mnist/>



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

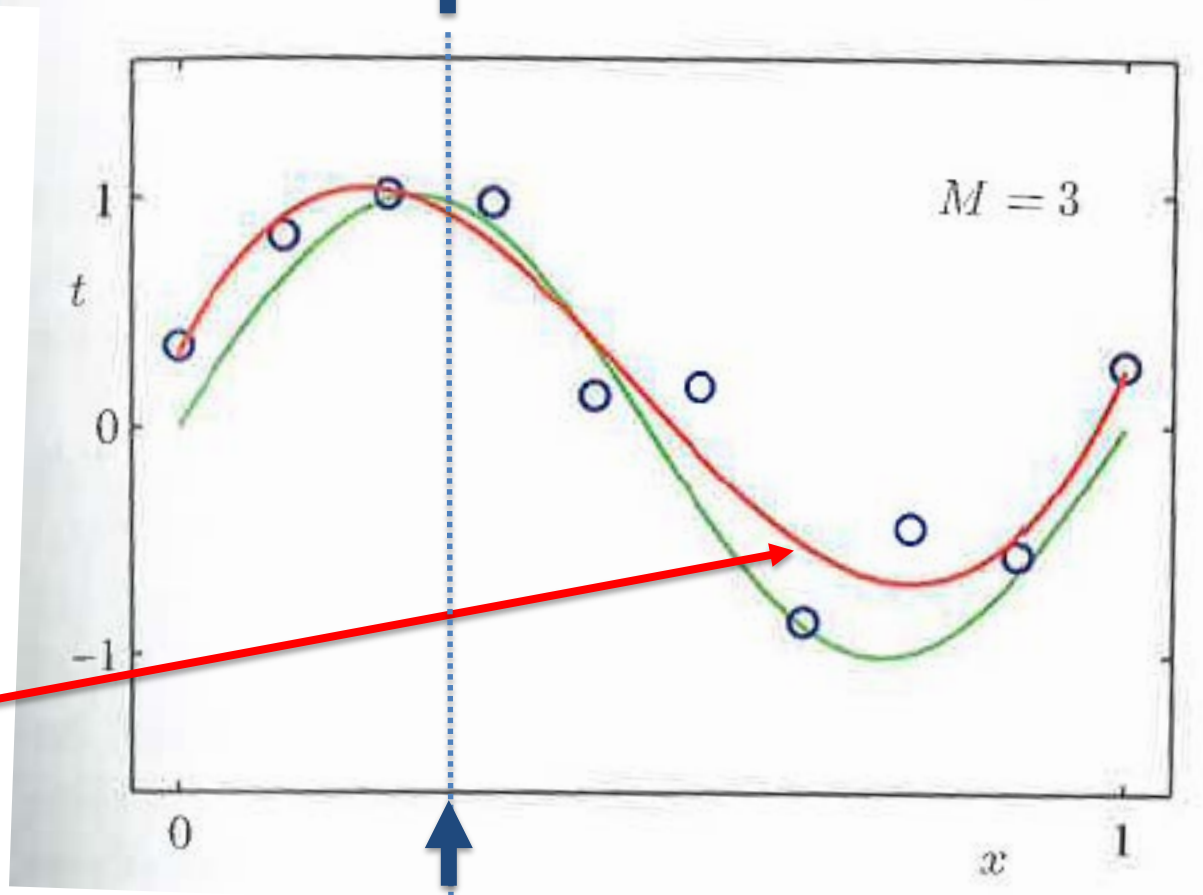
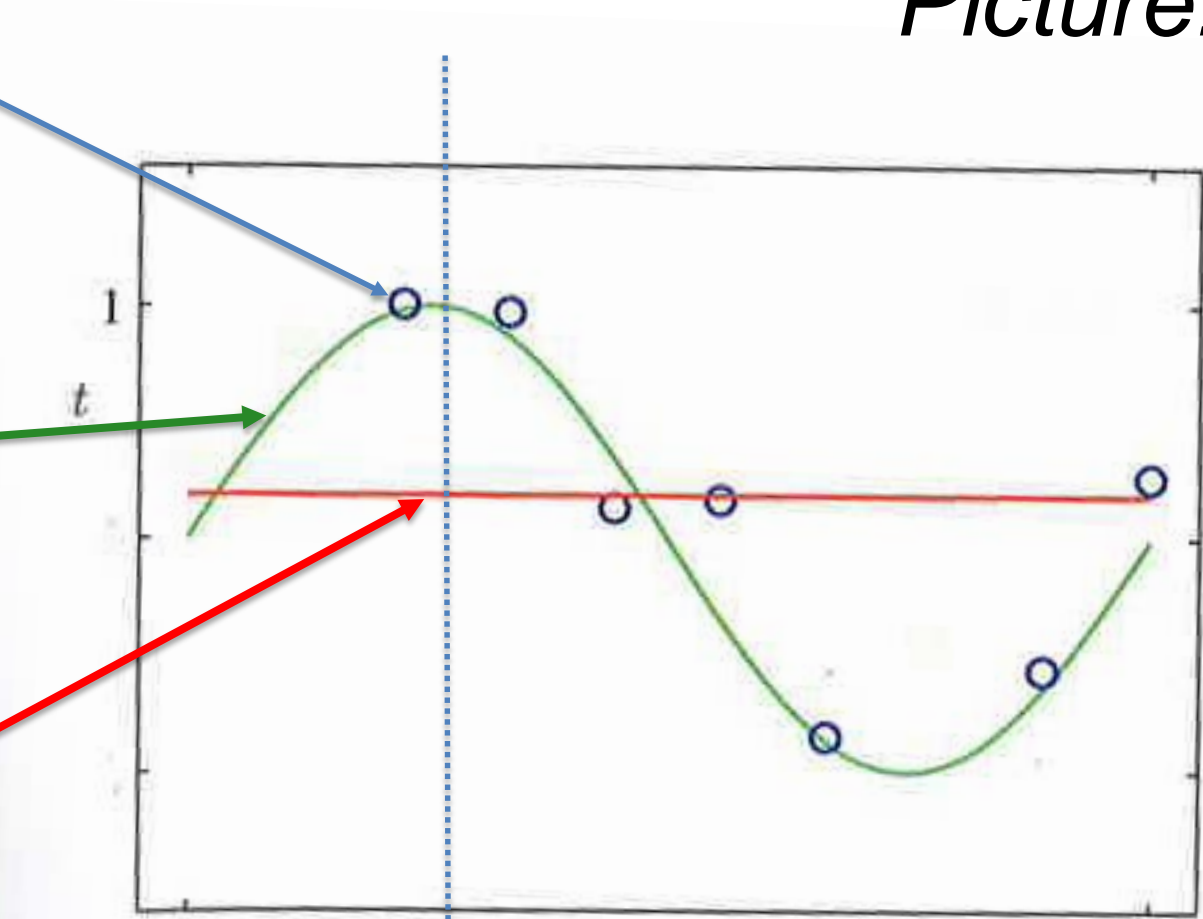
Picture: Bishop, 2006

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

$f(x) = \sin(x)$

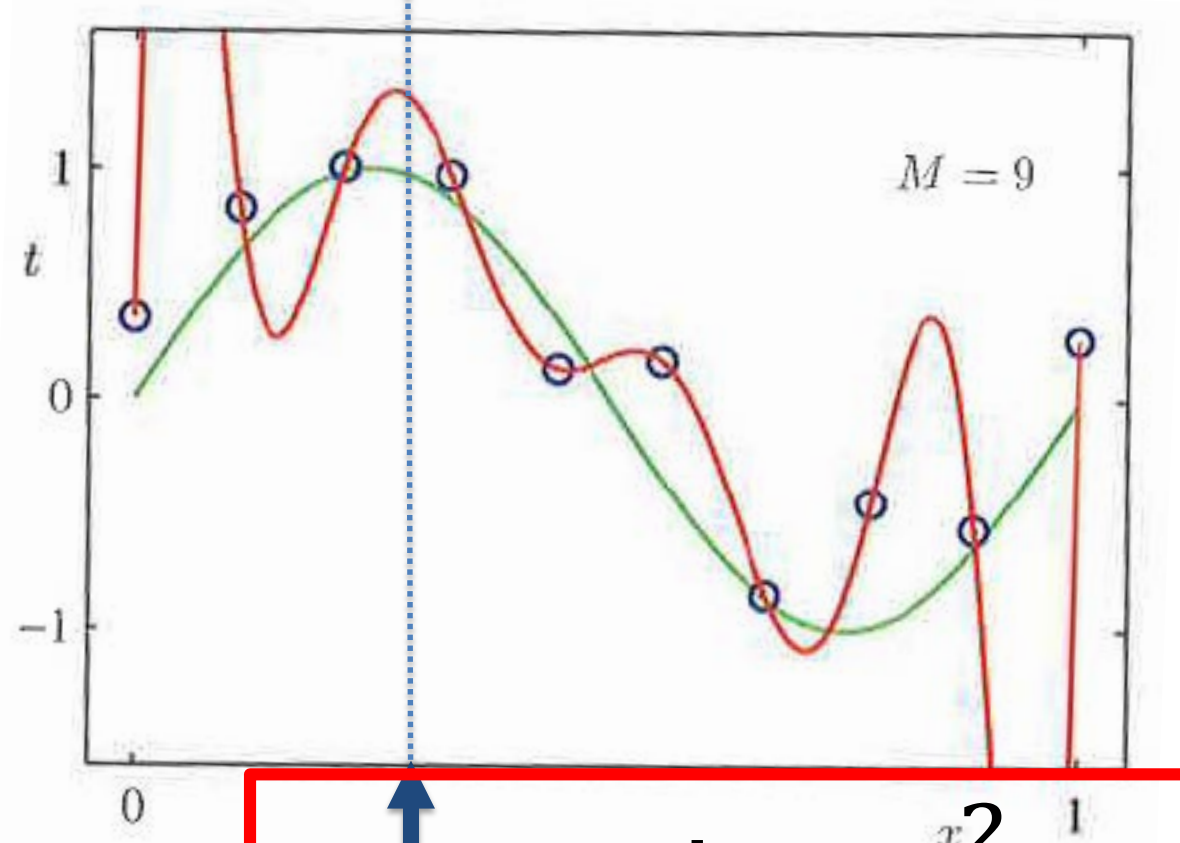
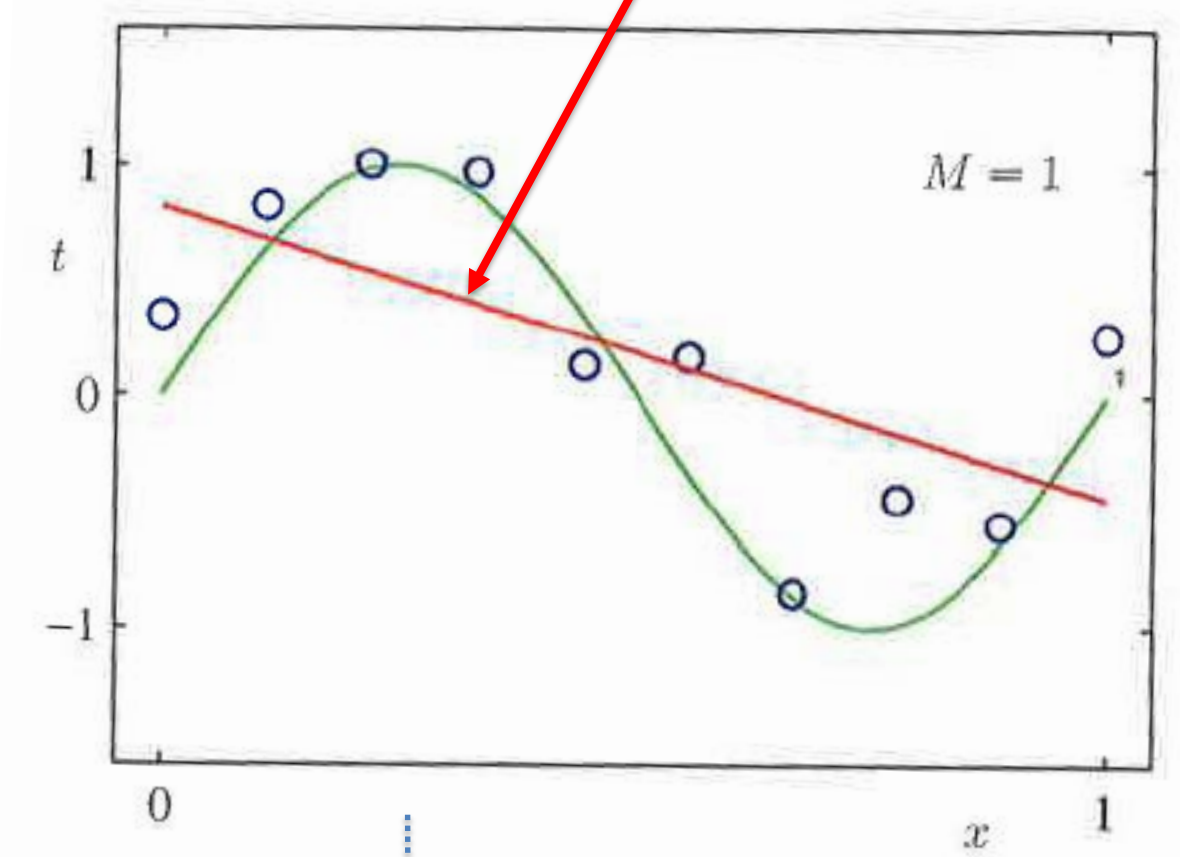
fit with

$y = w_0$



$y = w_0 + w_1x + w_2x^2 + w_3x^3$
4 parameters

$y = w_0 + w_1x$



$y = w_0 + w_1x + w_2x^2 + \dots + w_9x^9$
10 parameters

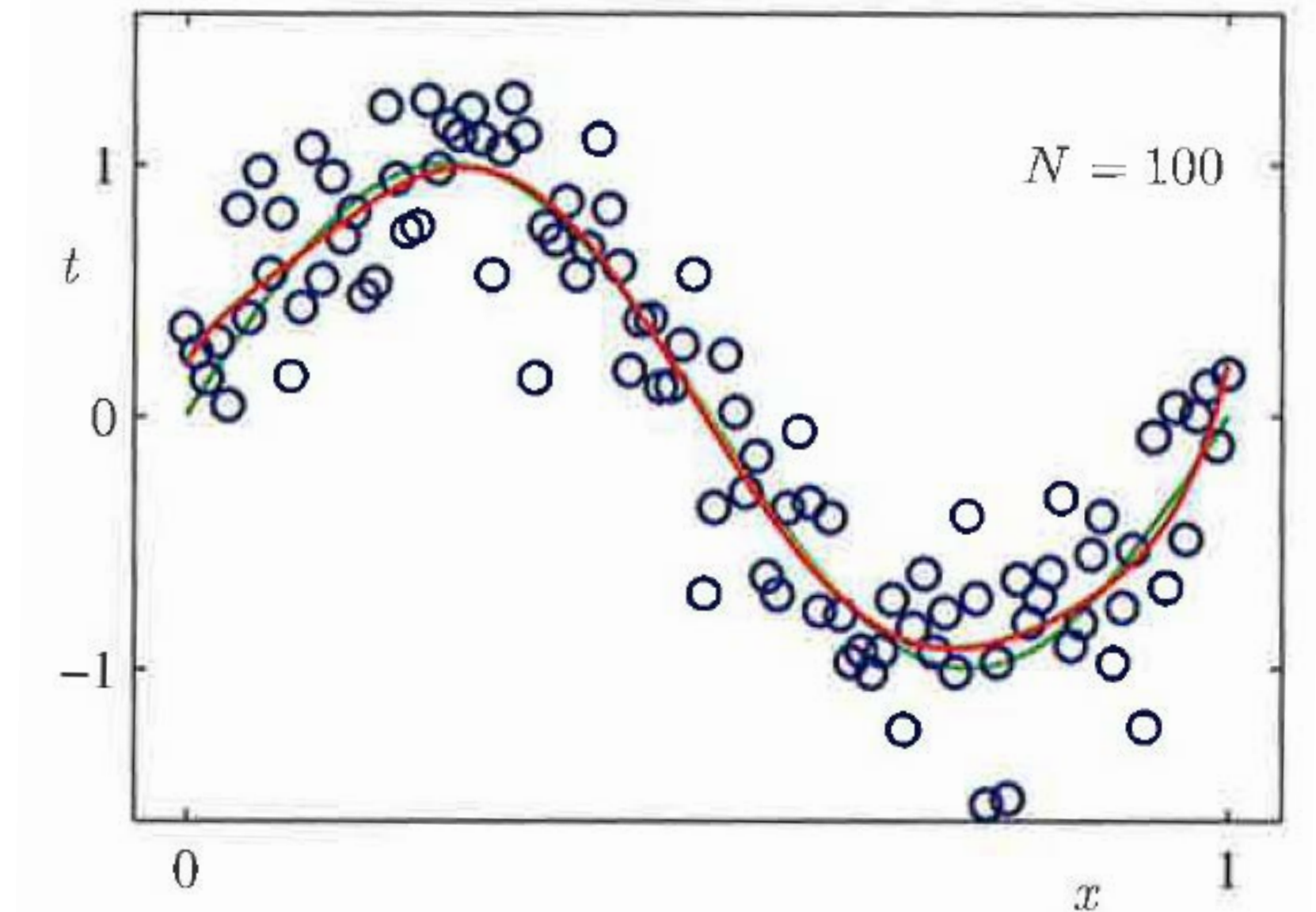
5. Curve fitting: Quiz

- ☐ A polynomial with 10 parameters can always be perfectly well fit 10 data points
- ☐ 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

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

10 parameters

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!

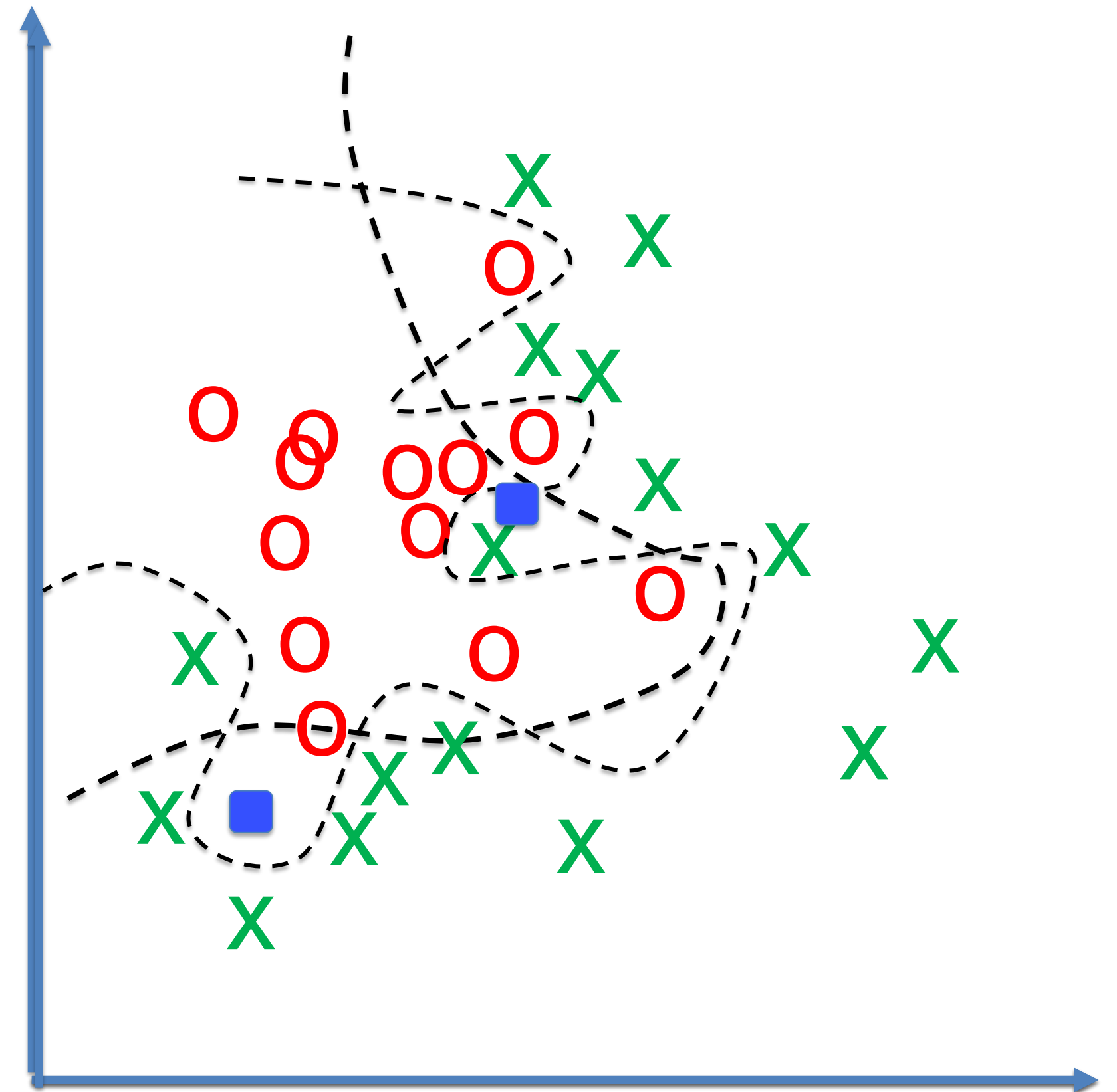
→ See course: *Machine Learning*
(Jaggi-Urbanke)

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!



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

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

input target output

Split data base

$$P = P1 + P2$$

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

Training base, used
to optimize parameters

Validation base, used
to mimic 'future data'

6. Error function on training data and validation data

$$\hat{y} = w_0 + w_1 x^2 \dots + w_9 x^9$$

10 parameters

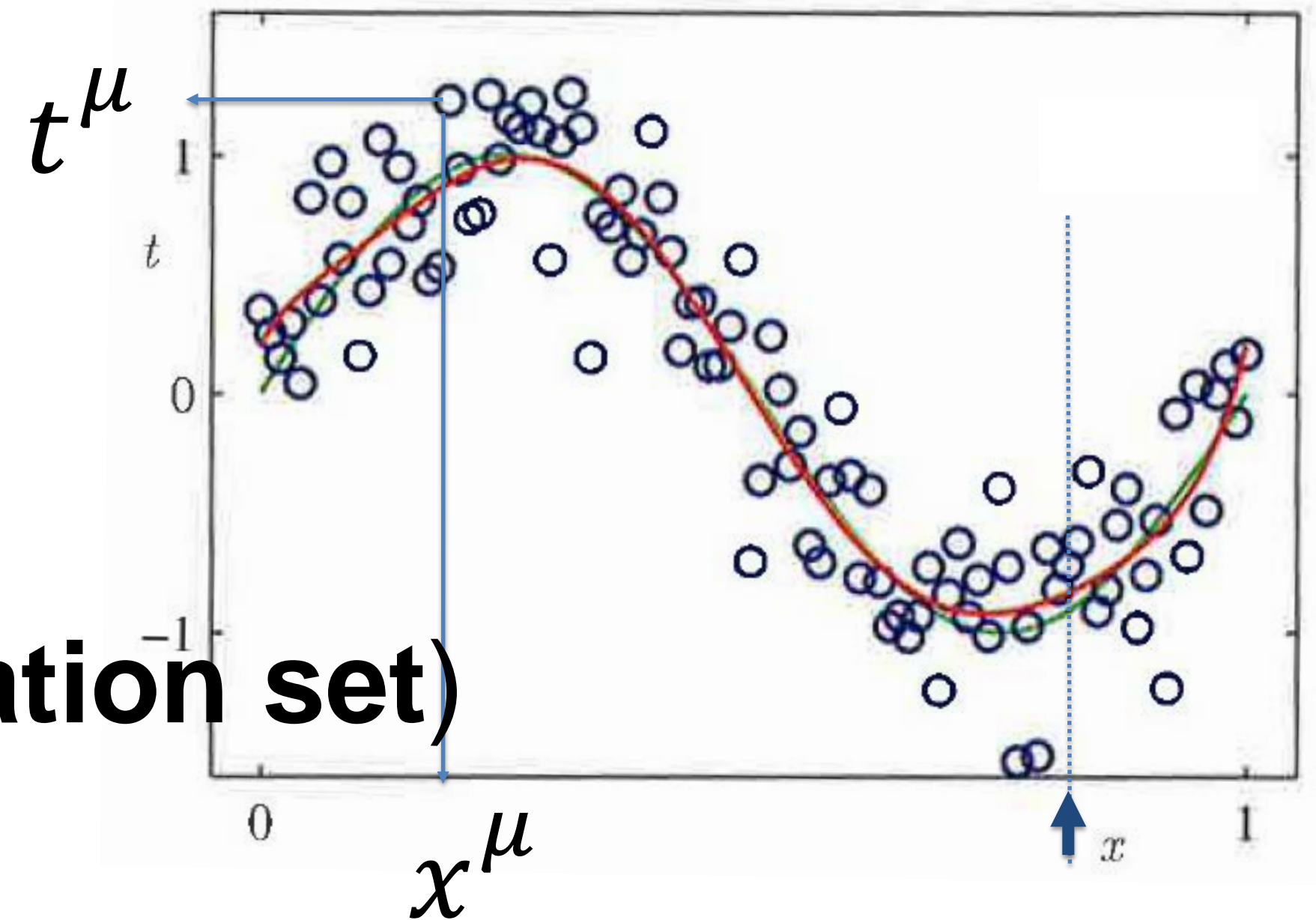
Fit with $P=100$ data points

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

Minimize error on **training set**

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

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

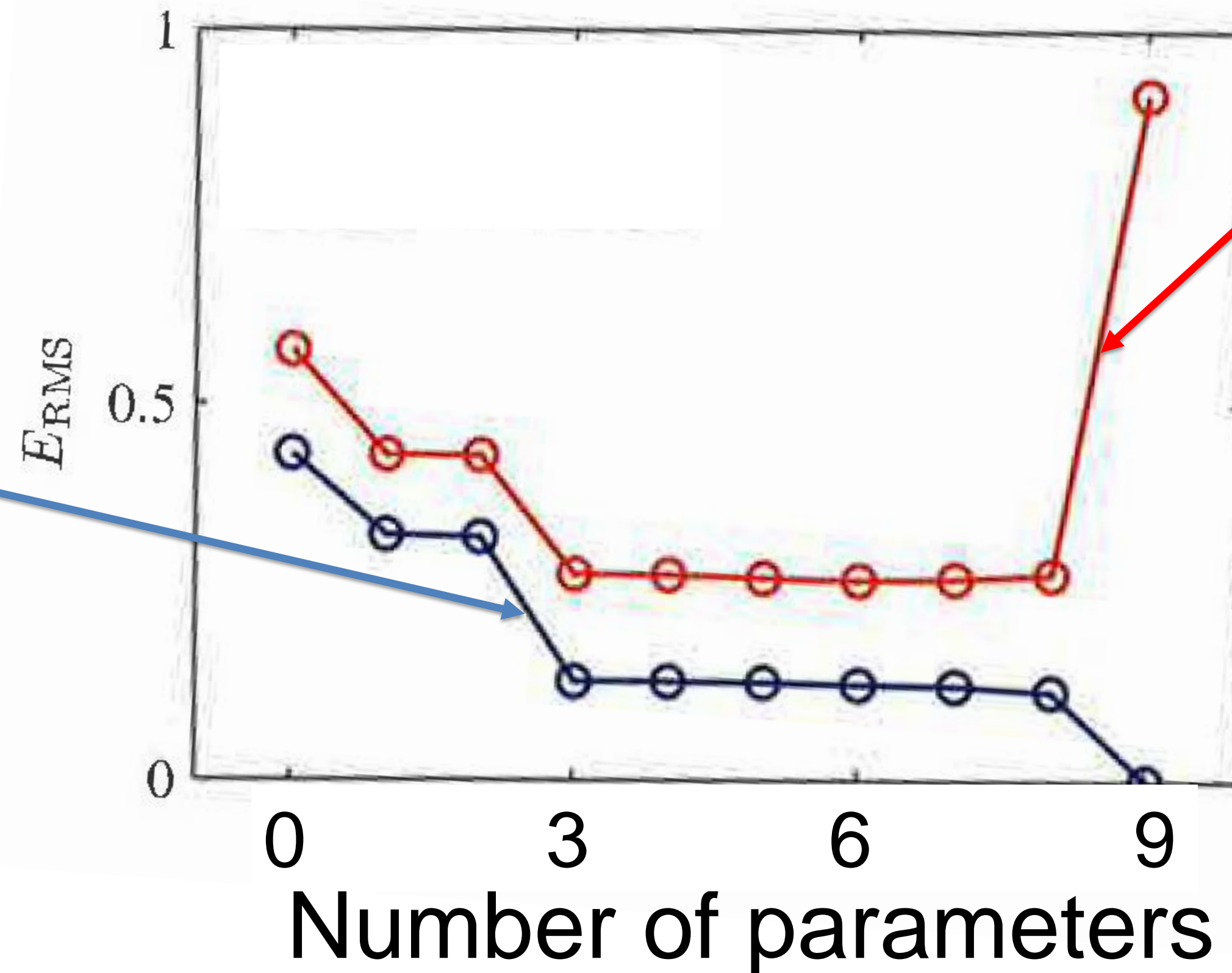


Picture: Bishop, 2006

6. Error function on training data and validation data

Example: polynomial curve fitting with $P1=10$ (training data size)

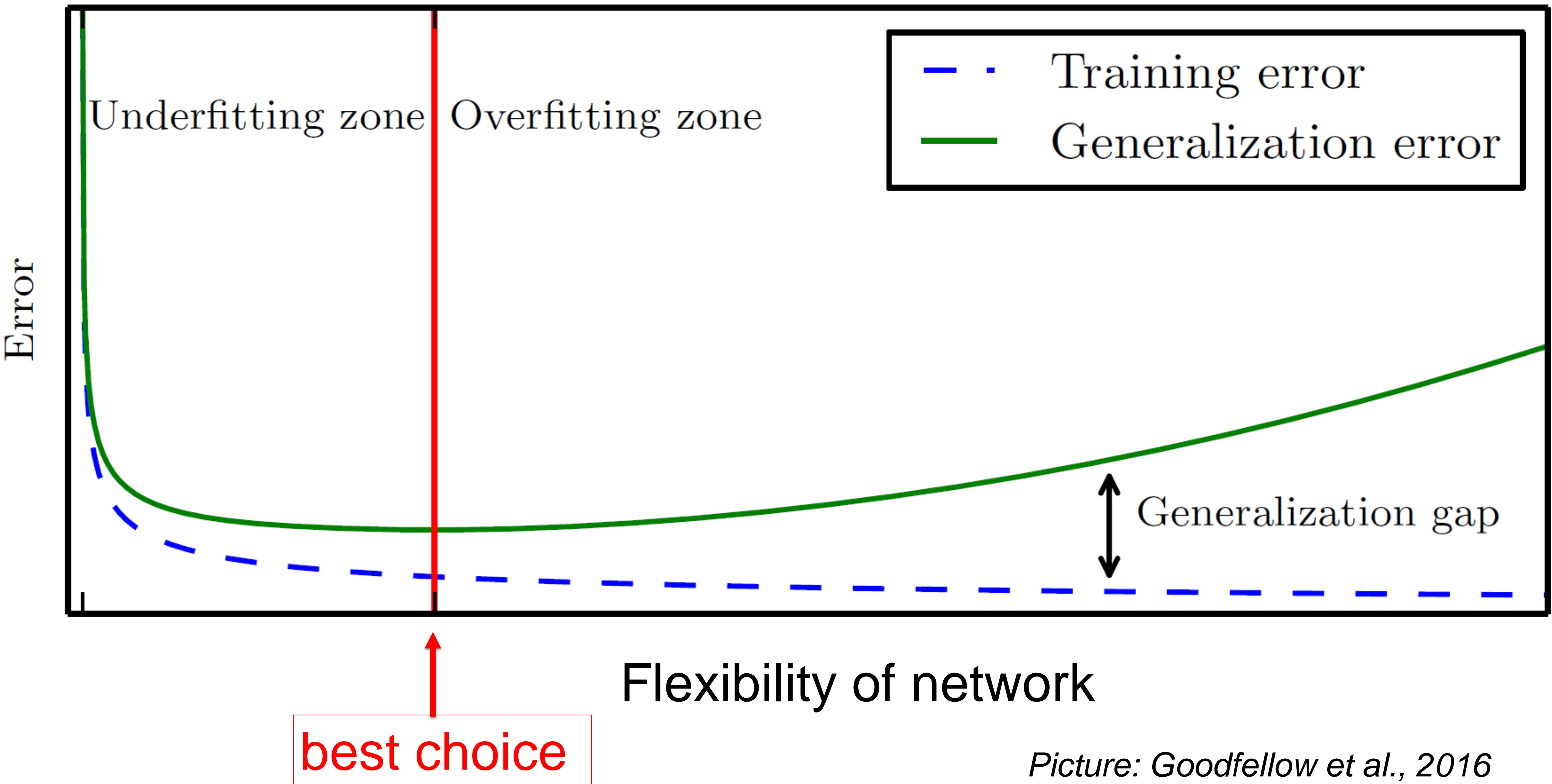
parameters
optimized to
minimize training
Error E



generalization
measured as
Val. error E^{val}

Picture: Bishop, 2006

6. Error function on training data and validation data



Picture: Goodfellow et al., 2016

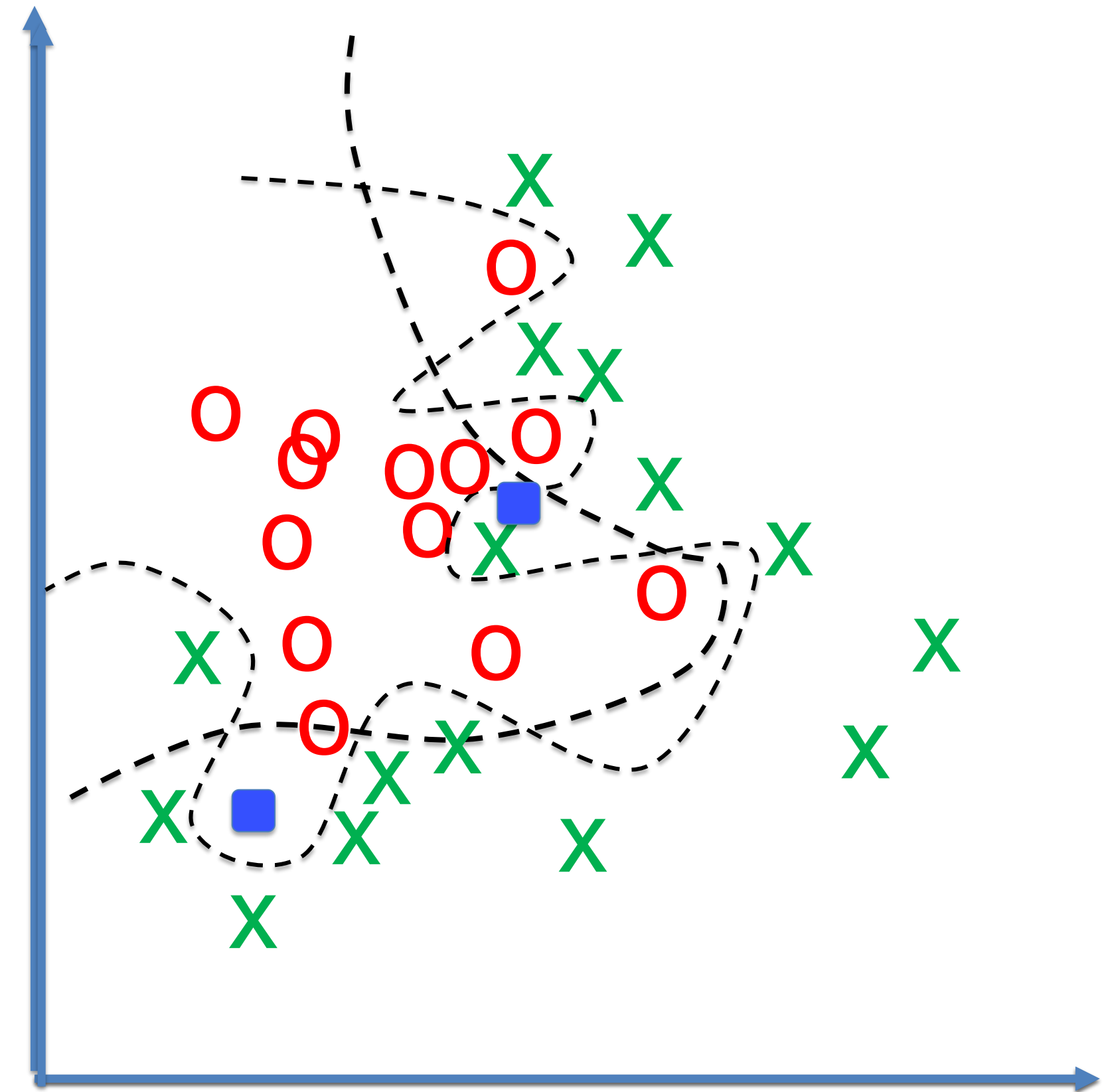
6. The problem of overfitting (revisited)

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!

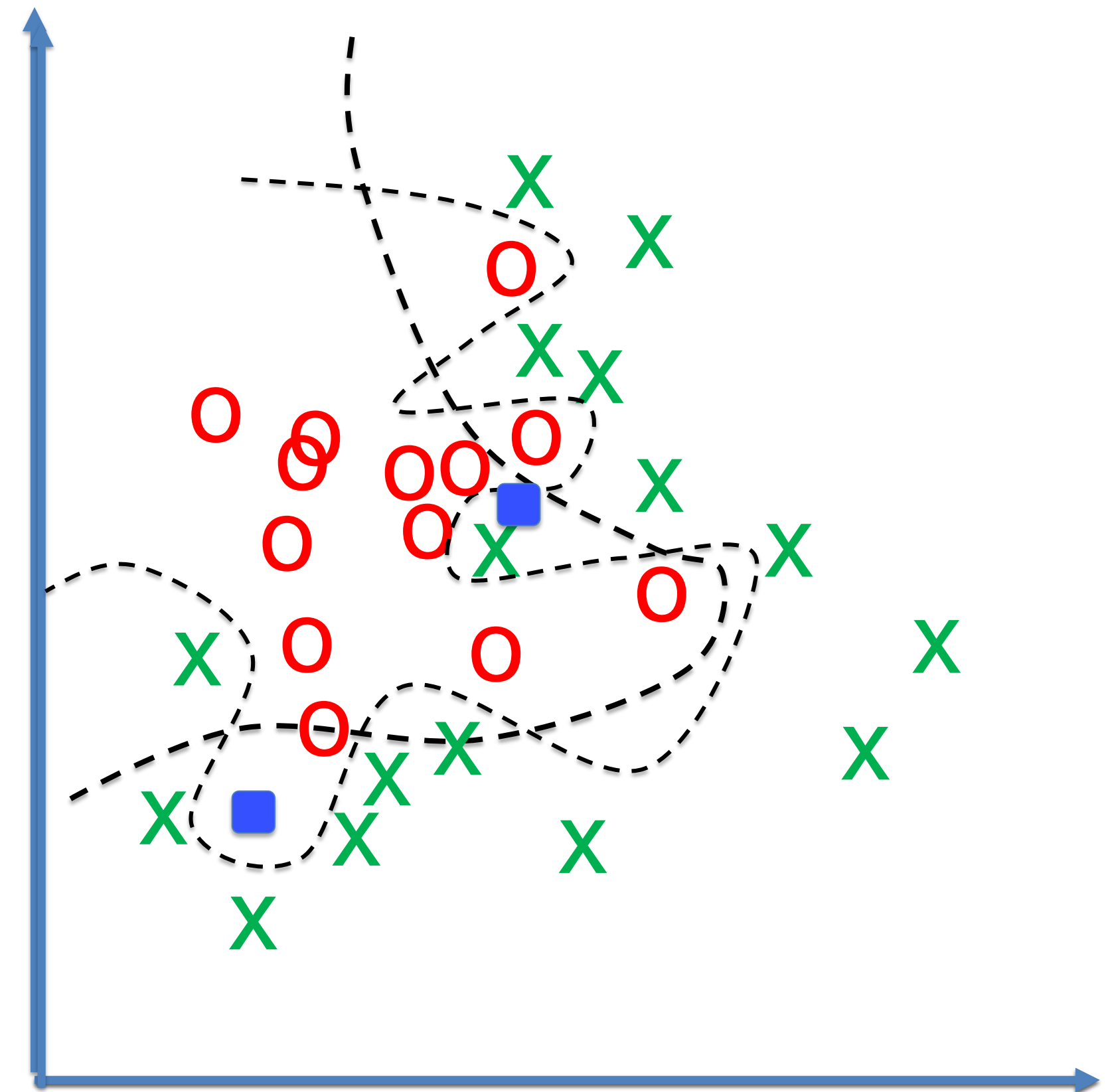
We can control overfitting by splitting into training base and validation base



6. The problem of overfitting (revisited)

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

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

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: use validation data

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
7. **Simple Regularization**

7. Controlling 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**

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

↑
assigns an 'error'
to flexible solutions

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

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

7. Regularization by a weight decay (L2 regularization)

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

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

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

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

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

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**

7. Regularization by a weight decay: curve fitting

Curve fitting, 10 data points, 10 parameters (as before)

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

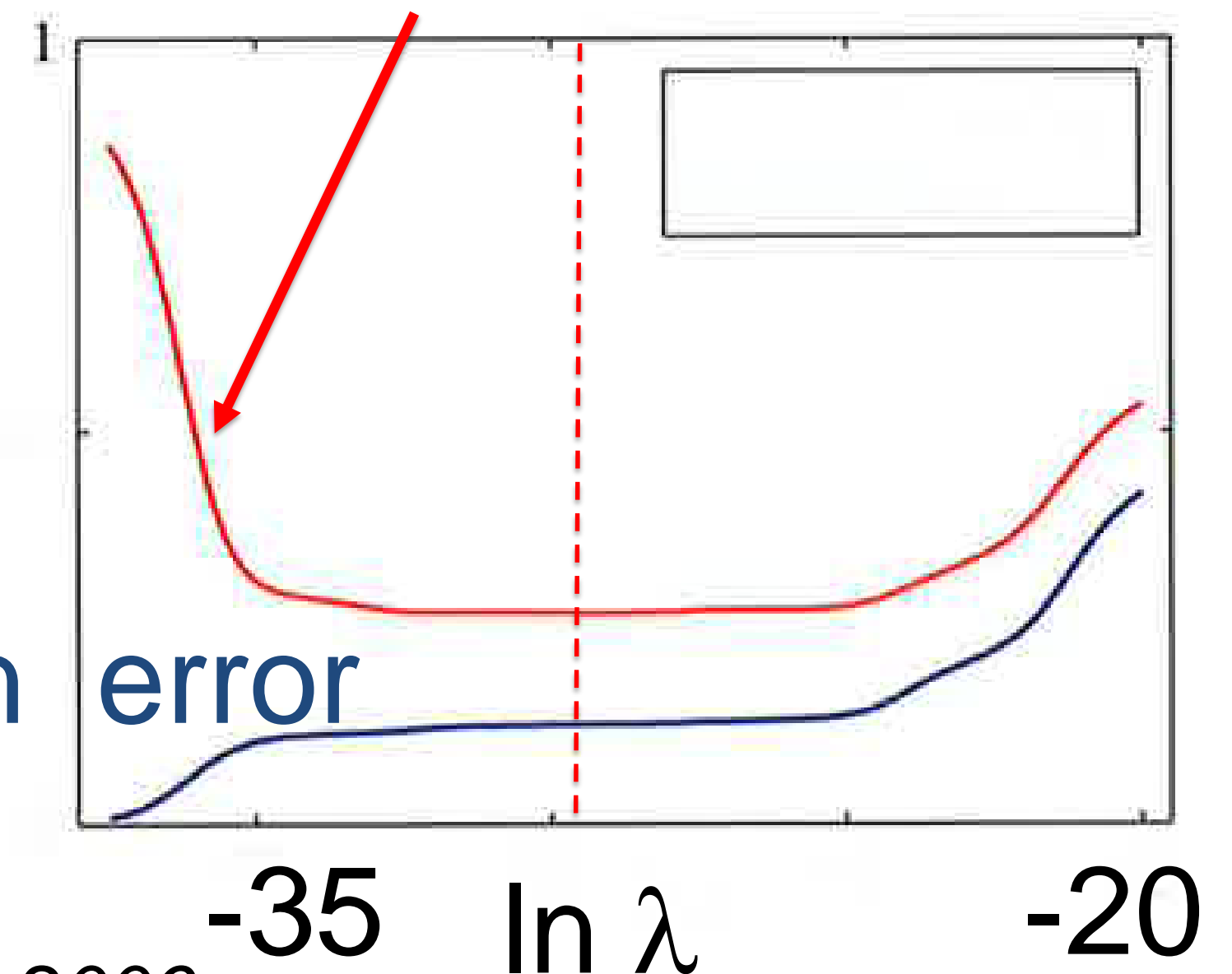
$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{\mu=1}^{P1} [t^{\mu} - \hat{y}^{\mu}]^2 + \lambda \sum_k (w_k)^2$$

If we decrease λ ,
Test error increases
(overfitting)

plot 'normal' error for both data sets

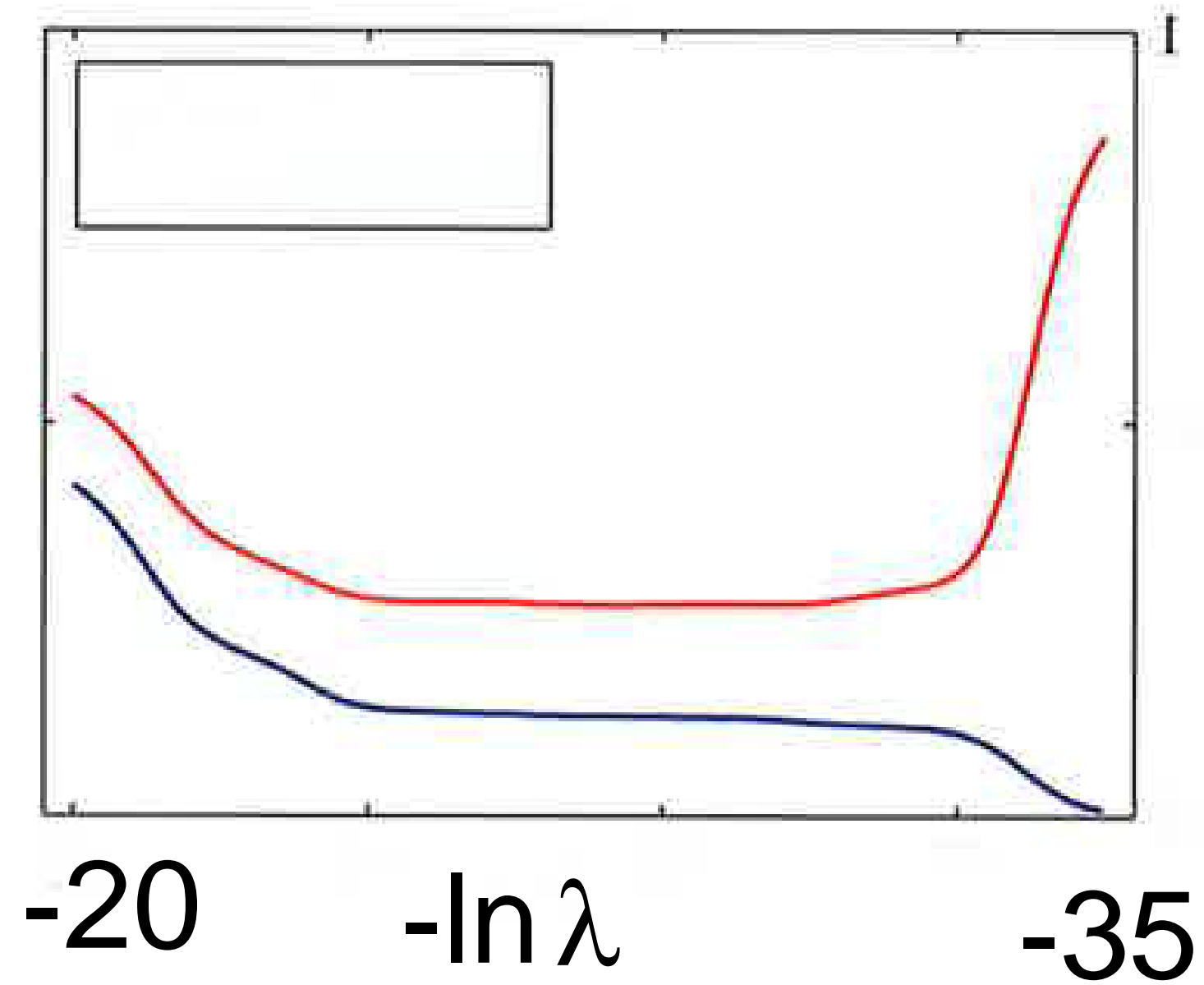
$$E(\mathbf{w}) = \frac{1}{2} \sum_{\mu=P1+1}^P [t^{\mu} - \hat{y}^{\mu}]^2$$

Validation error



Picture: Bishop, 2006

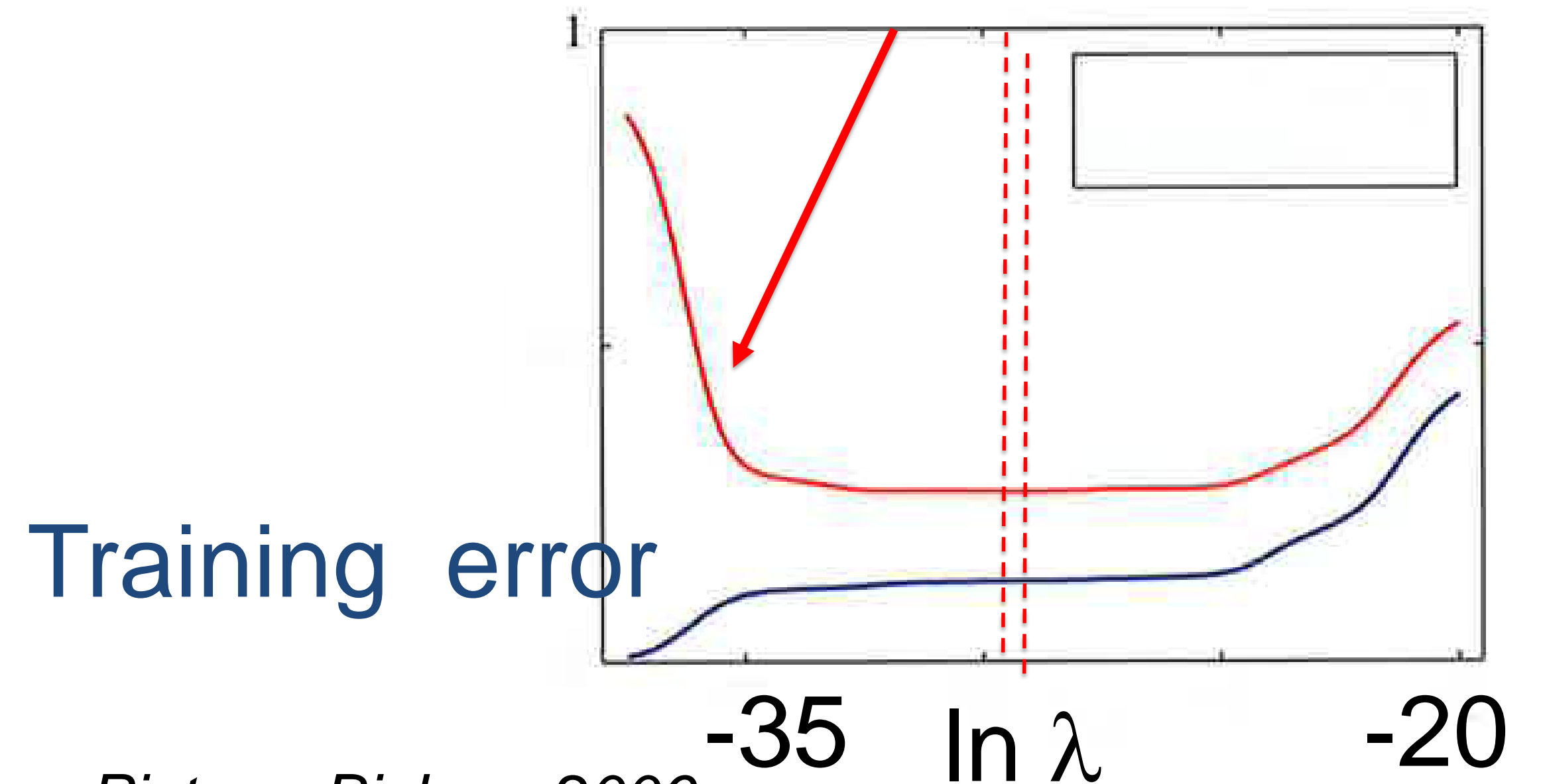
7. Regularization by a weight decay: curve fitting



decreasing $\lambda \rightarrow$

decreasing $\lambda \leftarrow$

If we decrease λ ,
Validation error increases
(overfitting)



Training error

Picture: Bishop, 2006

7. Regularization

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

Conclusion:

- we can keep the real number of parameters fixed and large, and still change flexibility via λ

Application to Artificial Neural Networks:

- we can work with fixed (large) number of hidden neurons and fixed (deep) network structure and control flexibility via regularization

7. Control of flexibility by regularizer

1 **Change flexibility** (several times)

Choose λ

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

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

Initialize weights

4 **Iterate until convergence**

Gradient descent on **modified**
training error $\tilde{E}(w)$

Report training error E and test error E^{val} on validation set

Report mean training and test error and SD

Plot mean training and test error

Pick weights for results with optimal λ

7. Control of flexibility by regularizer

- weights are parameters
- λ is also a parameter (hyperparameter)

BUT ATTENTION:

→ Test set/validation set is no longer 'future data' because we have used it to optimize λ

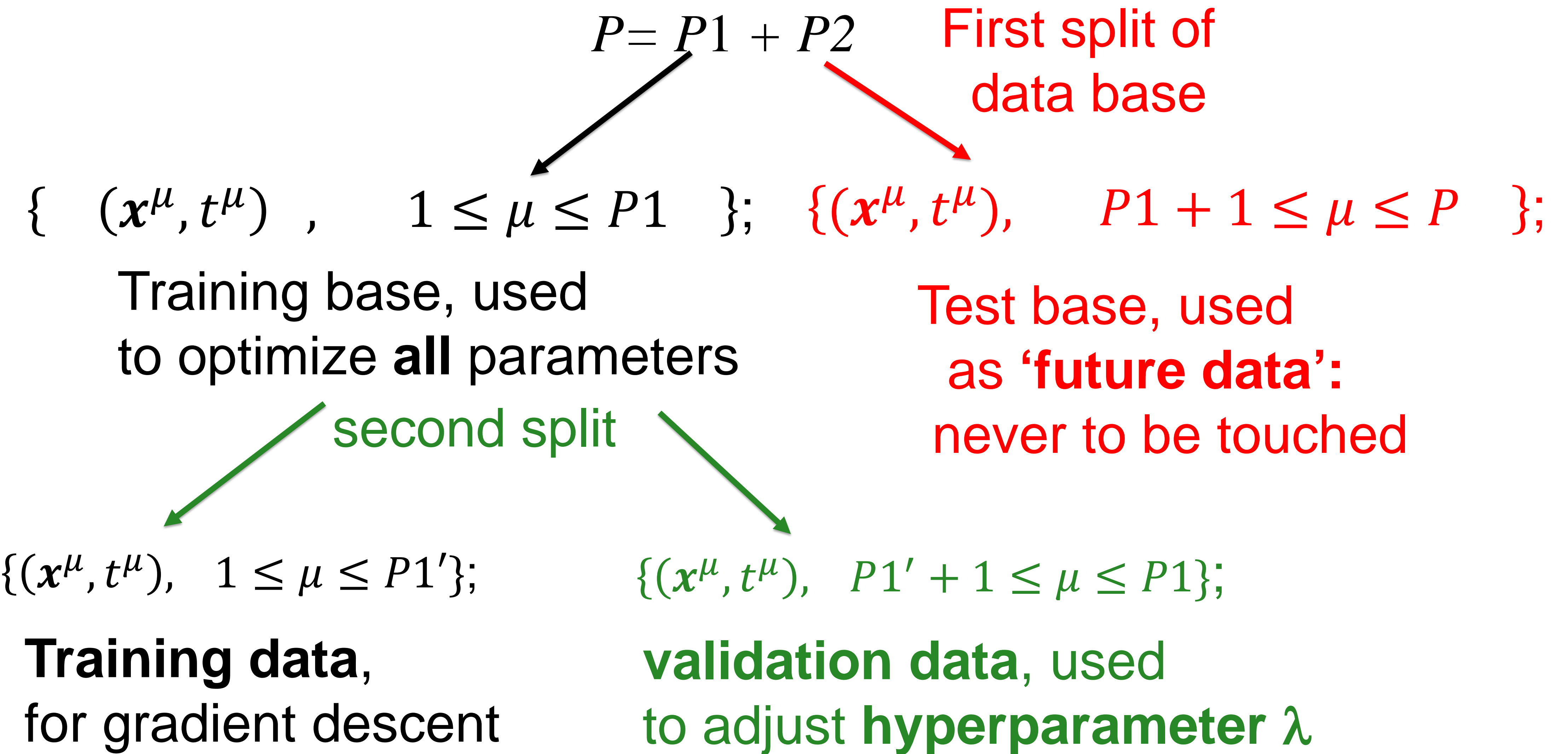
**If you have enough data, and many hyperparameters,
use a double-split**

Artificial Neural Networks: Lecture 2

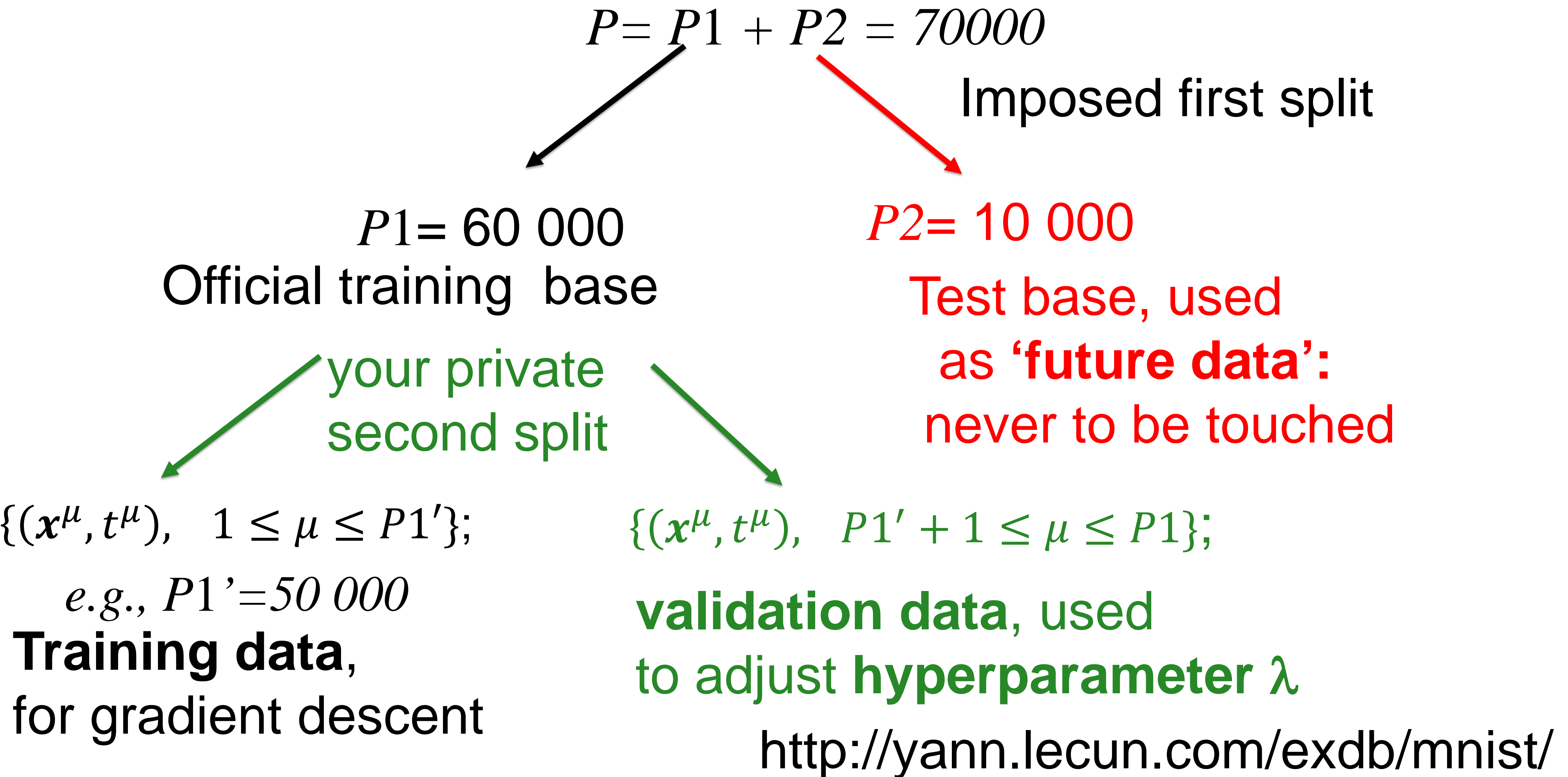
Backprop and multilayer perceptrons

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

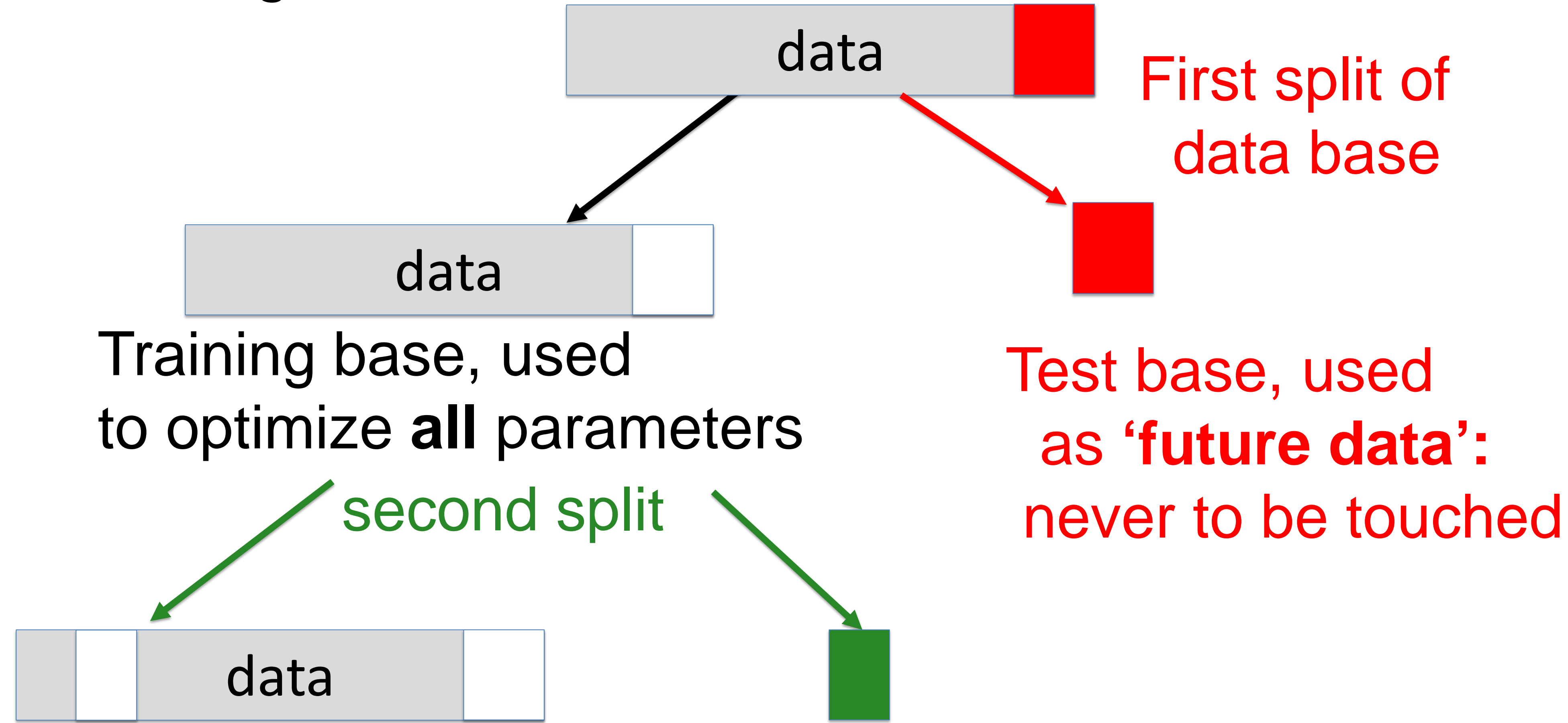
8. Training base, validation base, test base



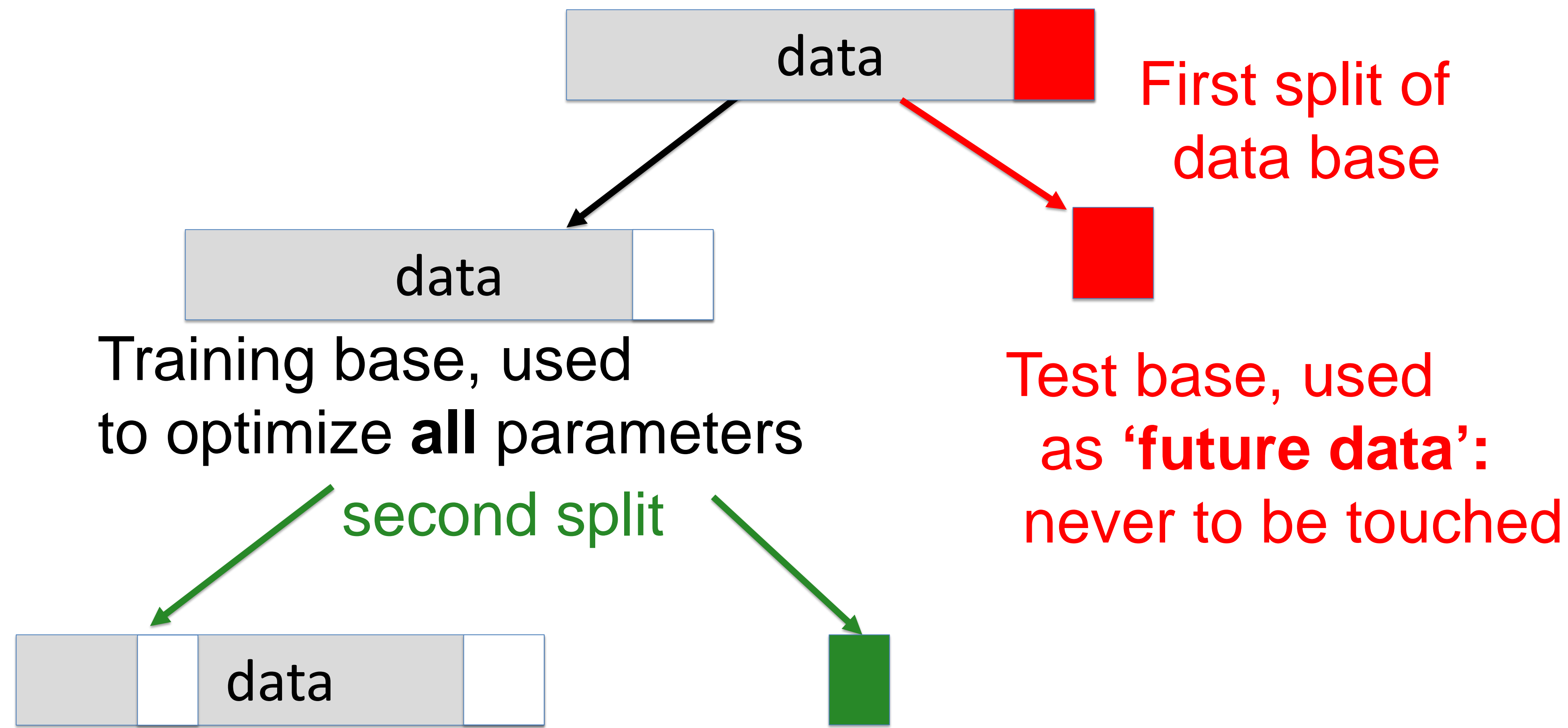
8. Example: MNIST Training base, validation base, test base



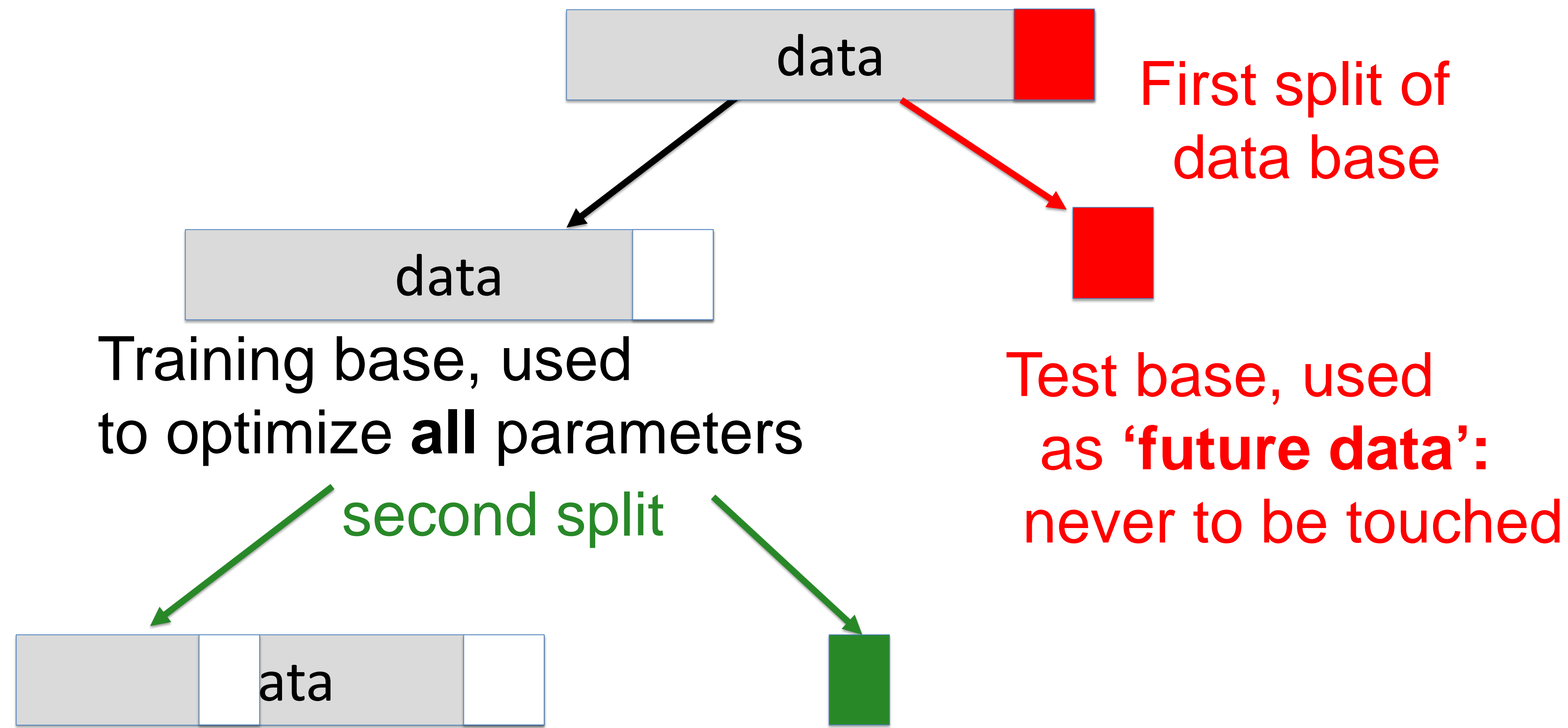
8. Training base, validation base, test base



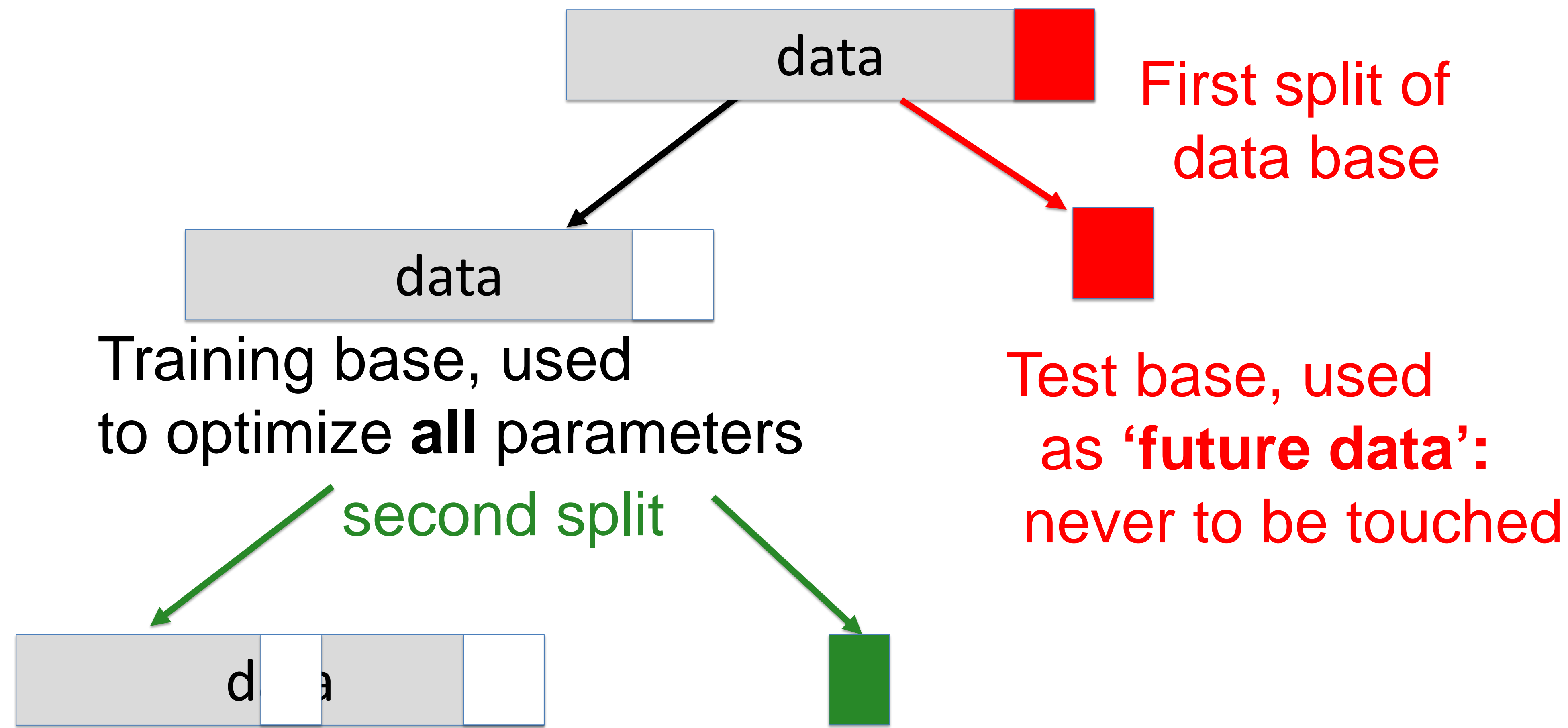
8. k-fold cross-validation



8. k-fold cross-validation



8. k-fold cross-validation



Artificial Neural Networks: Lecture 2

Backprop and multilayer perceptrons

1. Modern Gradient Descent
2. XOR problem
3. Multilayer Perceptron
4. BackProp Algorithm
5. The problem of overfitting
6. Training base and validation base
7. Simple Regularization
8. Careful Cross-validation
9. **Regularization by early stopping**

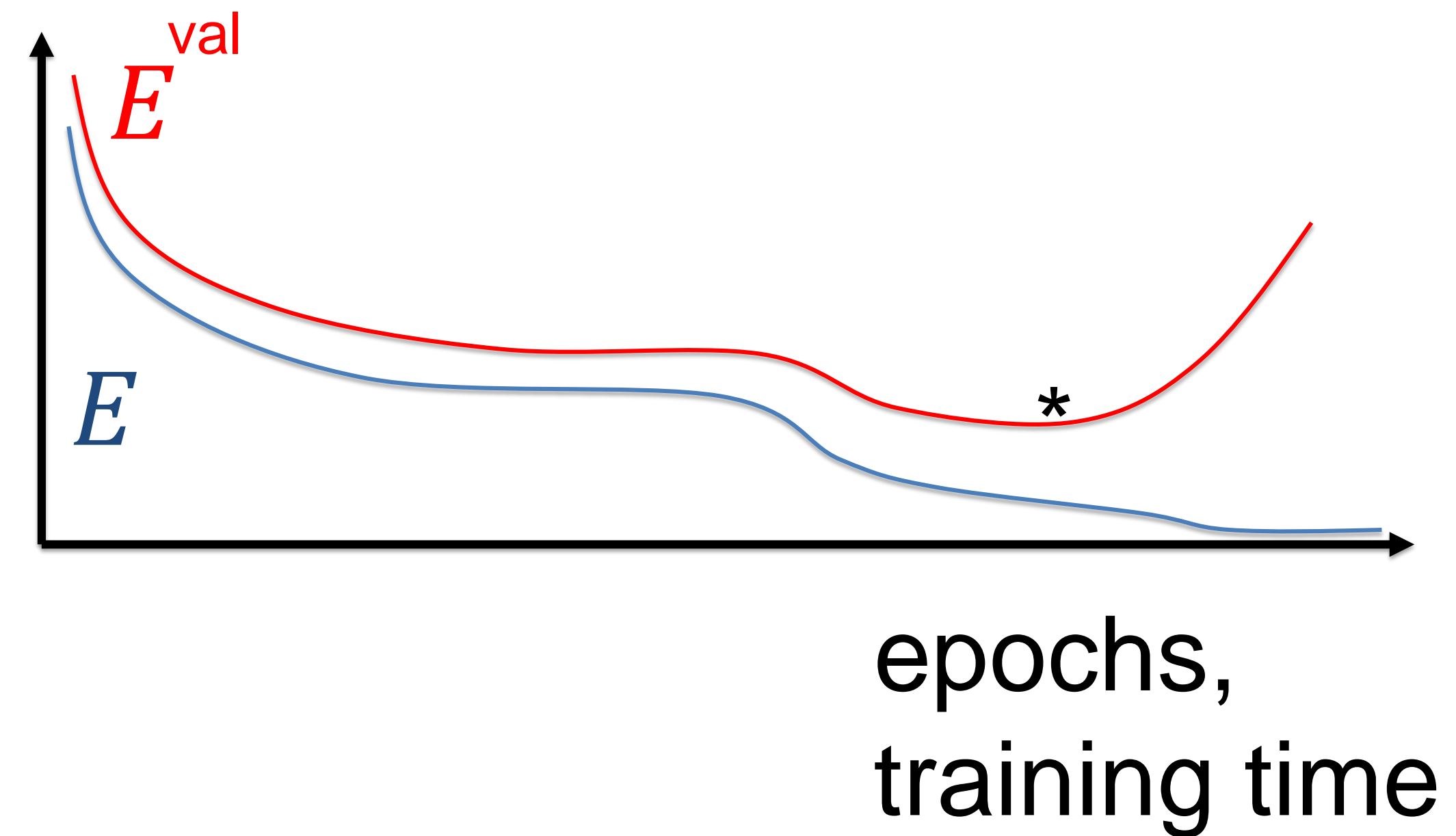
9. Regularization by early stopping

Minimize **training error**
stepwise by gradient descent

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

Every k steps plot error for
both data sets

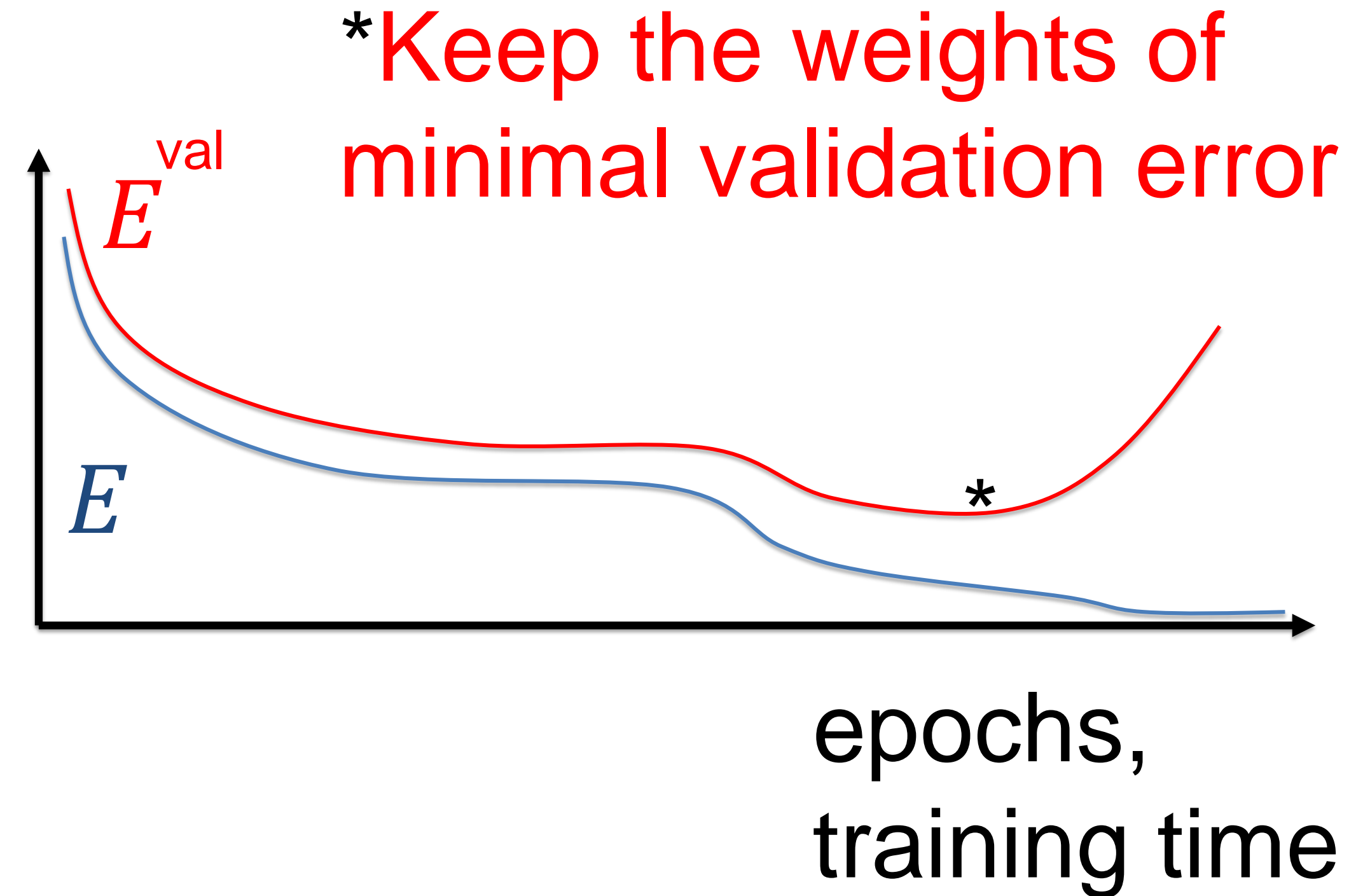
$$E^{\text{val}}(\mathbf{w}) = \frac{1}{2} \sum_{\mu=P1+1}^P [t^{\mu} - \hat{y}^{\mu}]^2$$



* Keep the weights for minimal validation error

9. Regularization by early stopping

- very easy to implement
- control of flexibility via learning time
- network 'uses' its total flexibility only after lengthy optimization
 - go back to 'earlier' solution
 - maximal flexibility not exploited



see also: week 3 and 4

9. Regularization by early stopping

- control of flexibility via learning time

- store weights of previous best solution

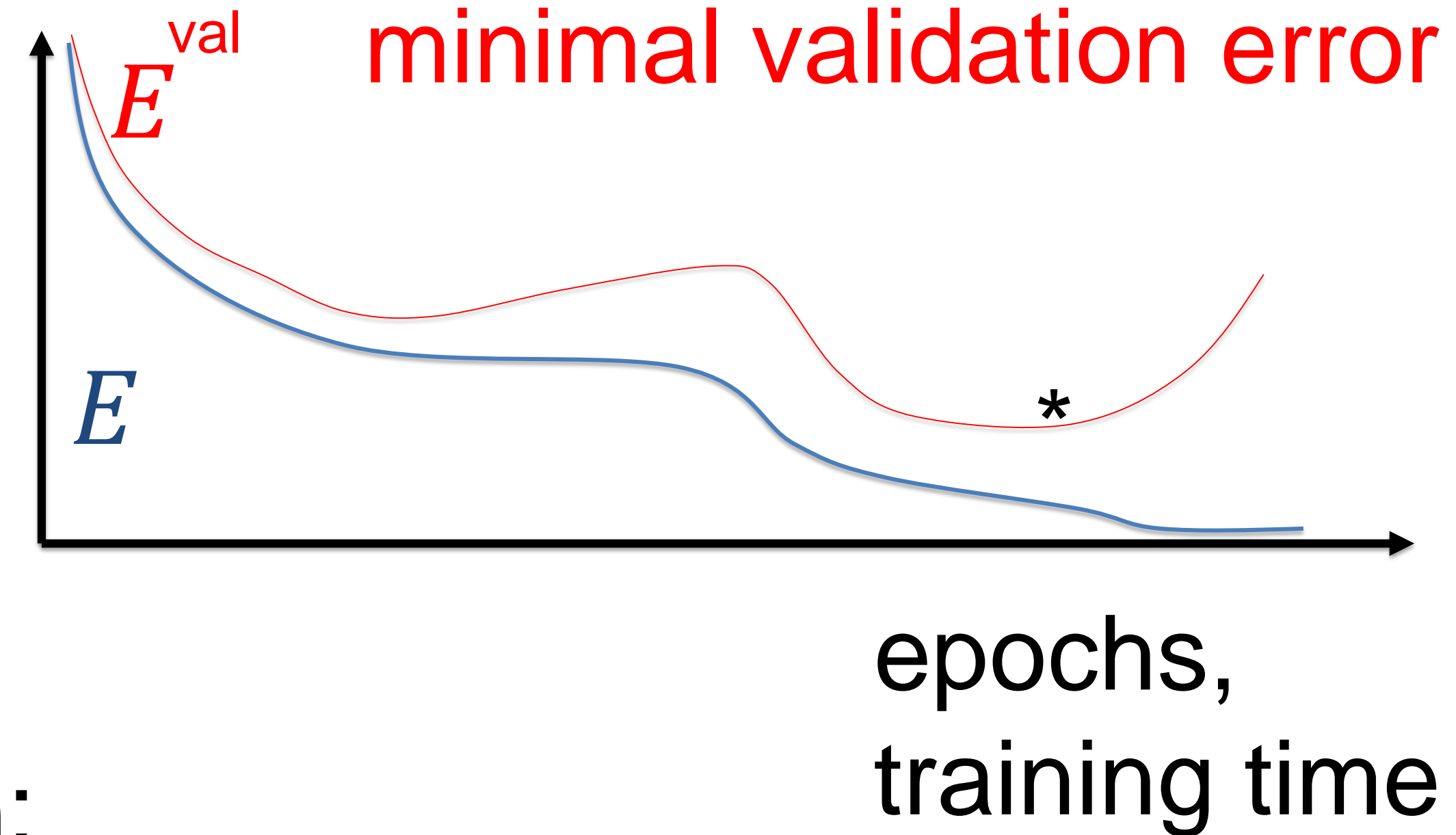
- continue to convergence

→ **go back to ‘earlier’ solution:**

keep weights of minimal validation error

→ maximal flexibility not exploited

***Keep the weights of minimal validation error**



‘Not early stopping, but going back’

see also: week 3 and 4

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 hyperparameters or early stopping: use validation data
- training base and validation and test base: the need predict well for future data
 - test Error

Reading for this lecture:

Bishop 2006, Ch. 1.1 and 5.3 of
Pattern recognition and Machine Learning

or

Bishop 1995, Ch. 1 and 4.8 of
Neural networks for pattern recognition




or

Goodfellow et al., 2016 Ch. 5.1-5.3 and 6.5 of
Deep Learning

Artificial Neural Networks

Wulfram Gerstner

EPFL, Lausanne, Switzerland

1. Simple perceptrons for classification
2. Backprop and multilayer perceptron
3. Statistical Classification by deep networks  [miniproject1](#)
4. Deep learning:
 regularization and tricks of the trade
5. Complements to deep learning
6. Sequence predictions and LSTMs  [miniproject2](#)
7. Convolutional networks
8. Reinforcement learning1: TD learning
9. Reinforcement learning2: SARSA  [miniproject3](#)
10. Reinforcement learning3: Policy Gradient
11. Deep Reinforcement learning
12. Applications