Artificial Neural Networks: Lecture 5

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Error landscape and optimization methods for deep networks

Objectives for today:
- Error function landscape: minima and saddle points
- Momentum
- Adam
- No Free Lunch
- Shallow versus Deep Networks

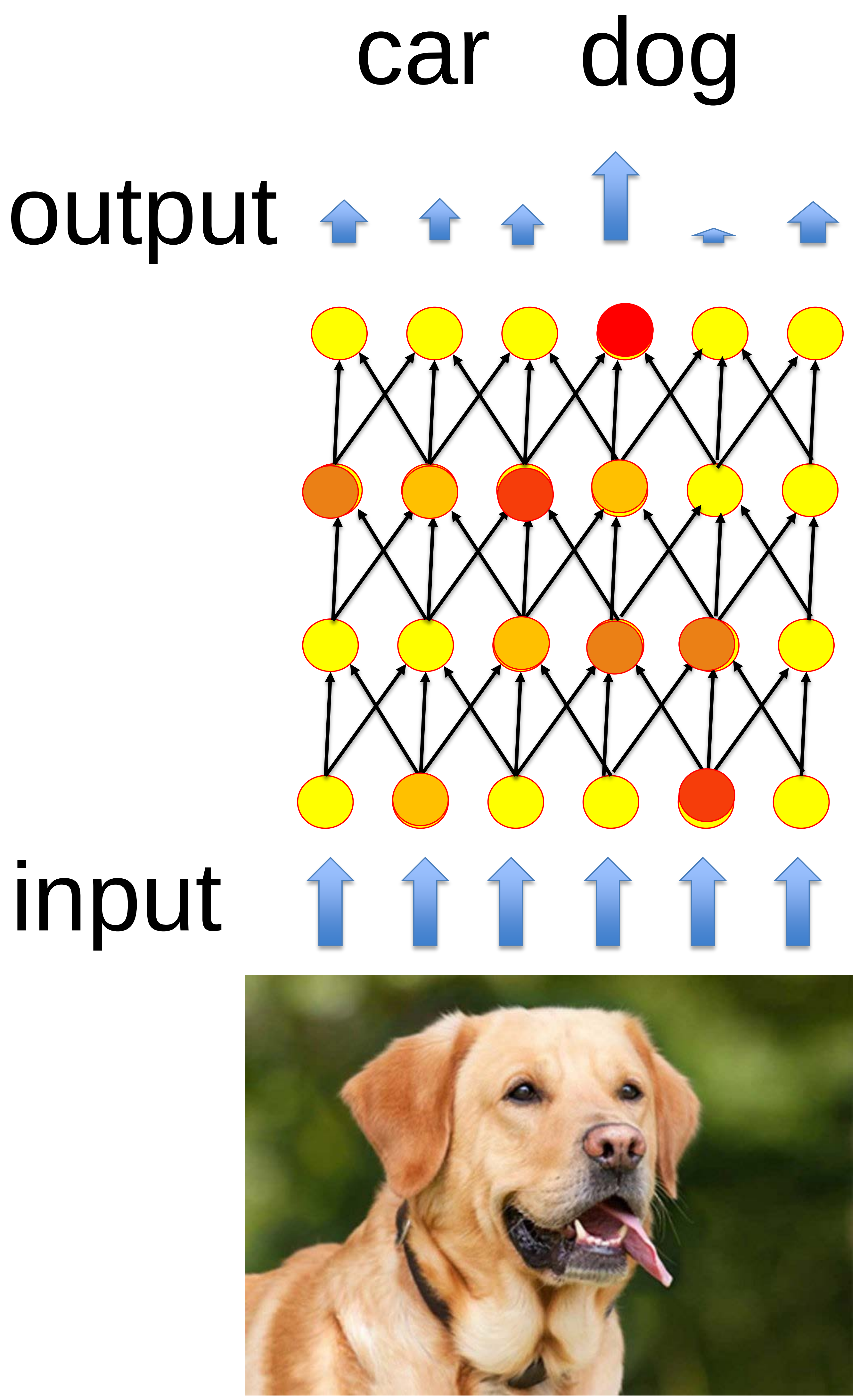
Reading for this lecture:

Goodfellow et al.,**2016** *Deep Learning*

- Ch. 8.2, Ch. 8.5
- Ch. 4.3
- Ch. 5.11, 6.4, Ch. 15.4, 15.5

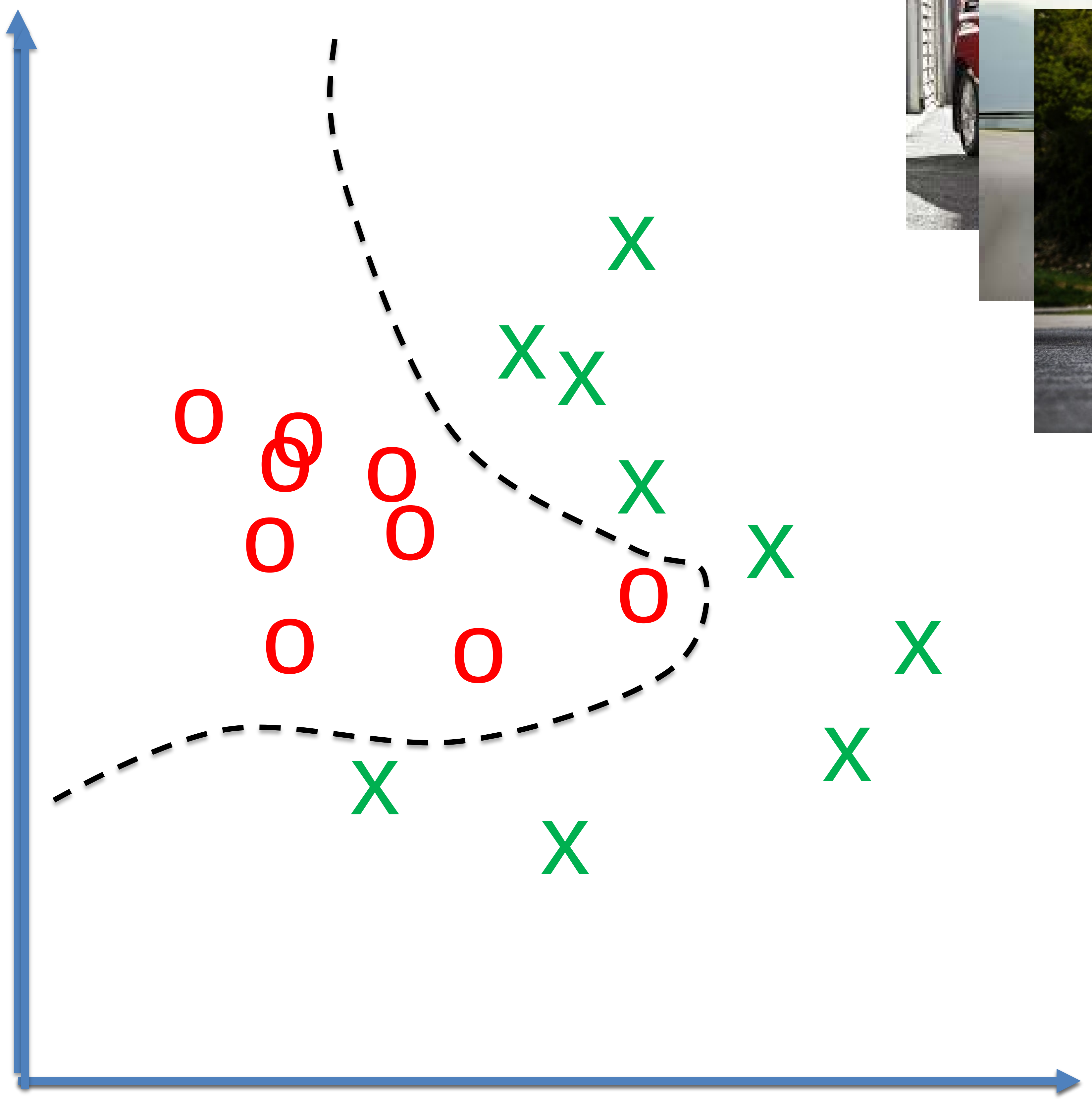
Further Reading for this Lecture:

review: Artificial Neural Networks for classification

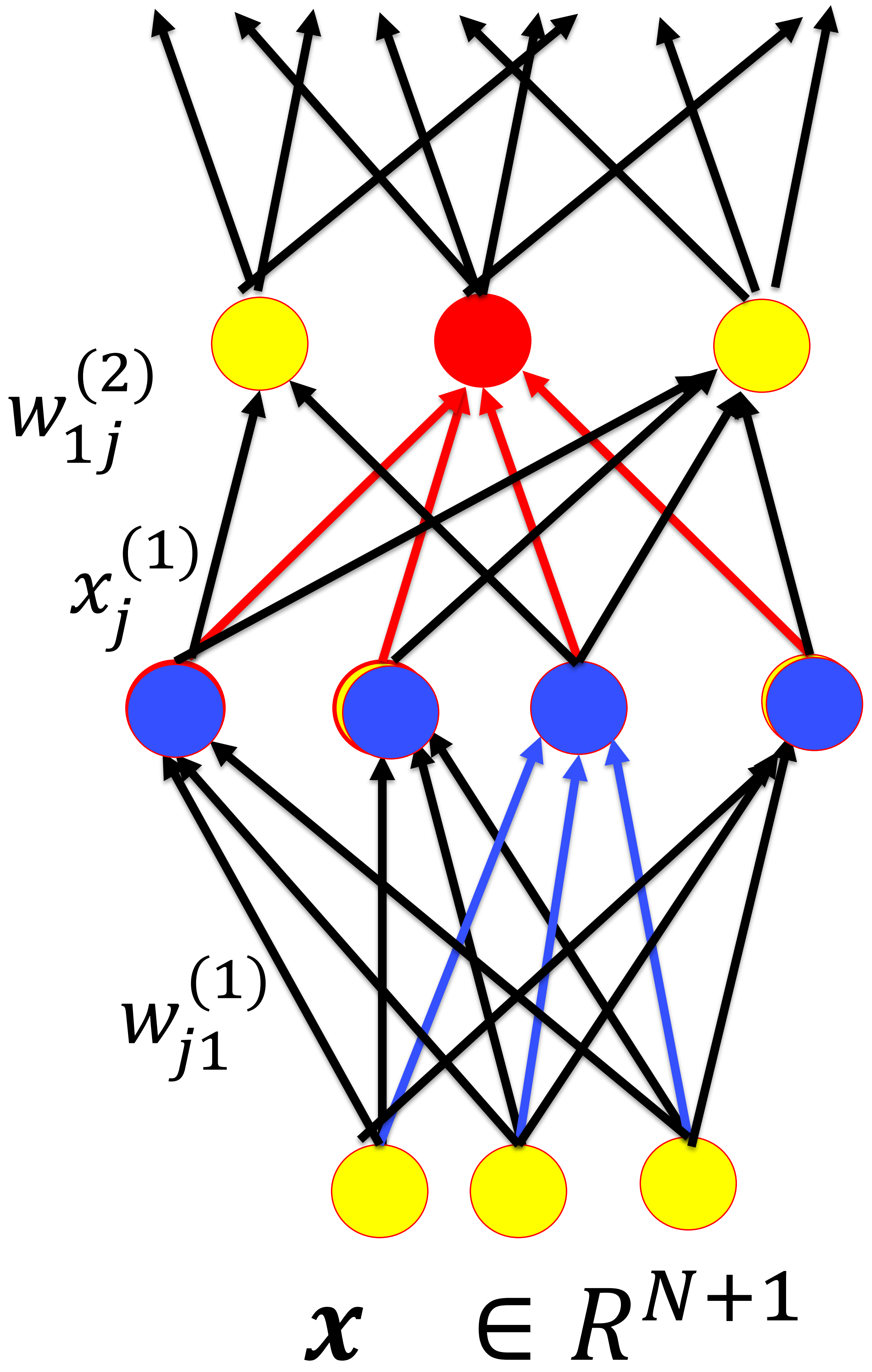


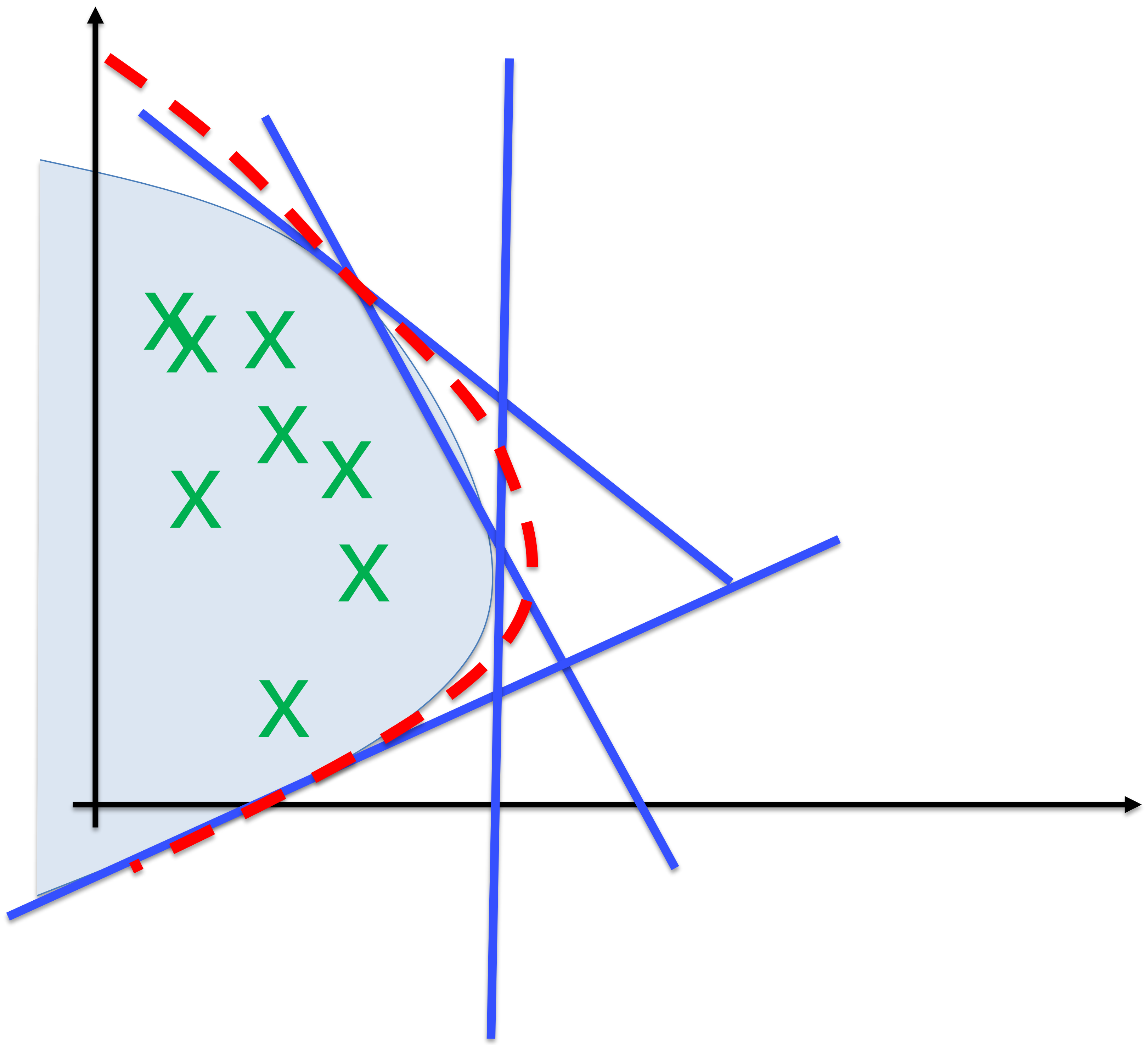
Aim of learning:
Adjust connections such that output is correct (for each input image, **even new ones**)

Review: Classification as a geometric problem

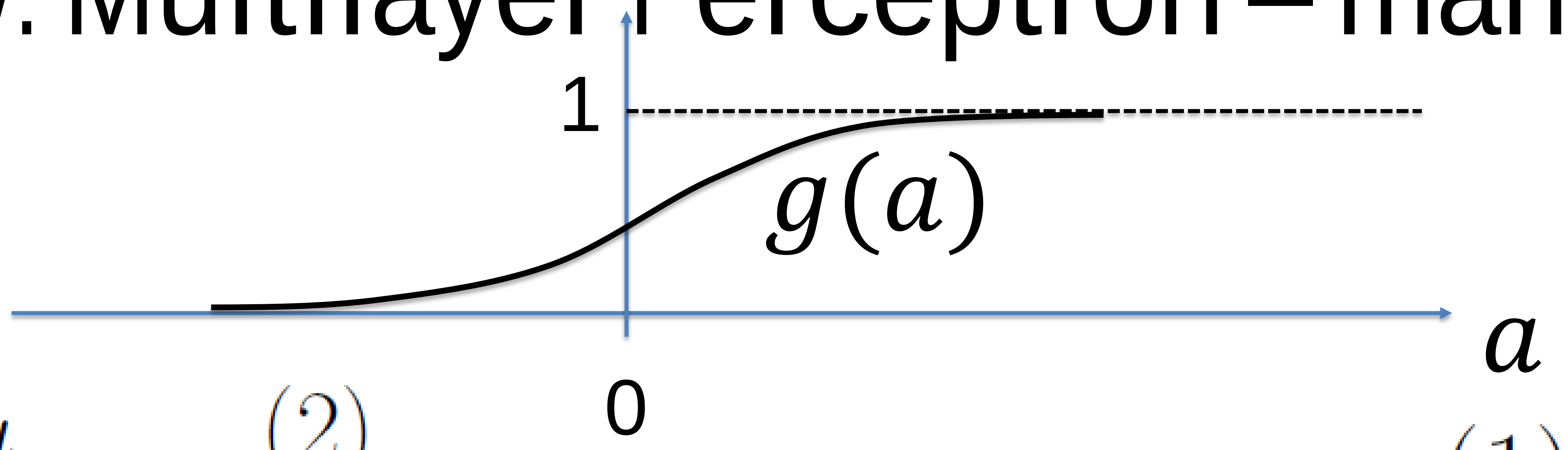


Review: task of hidden neurons (blue)





Review: Multilayer Perceptron – many parameters



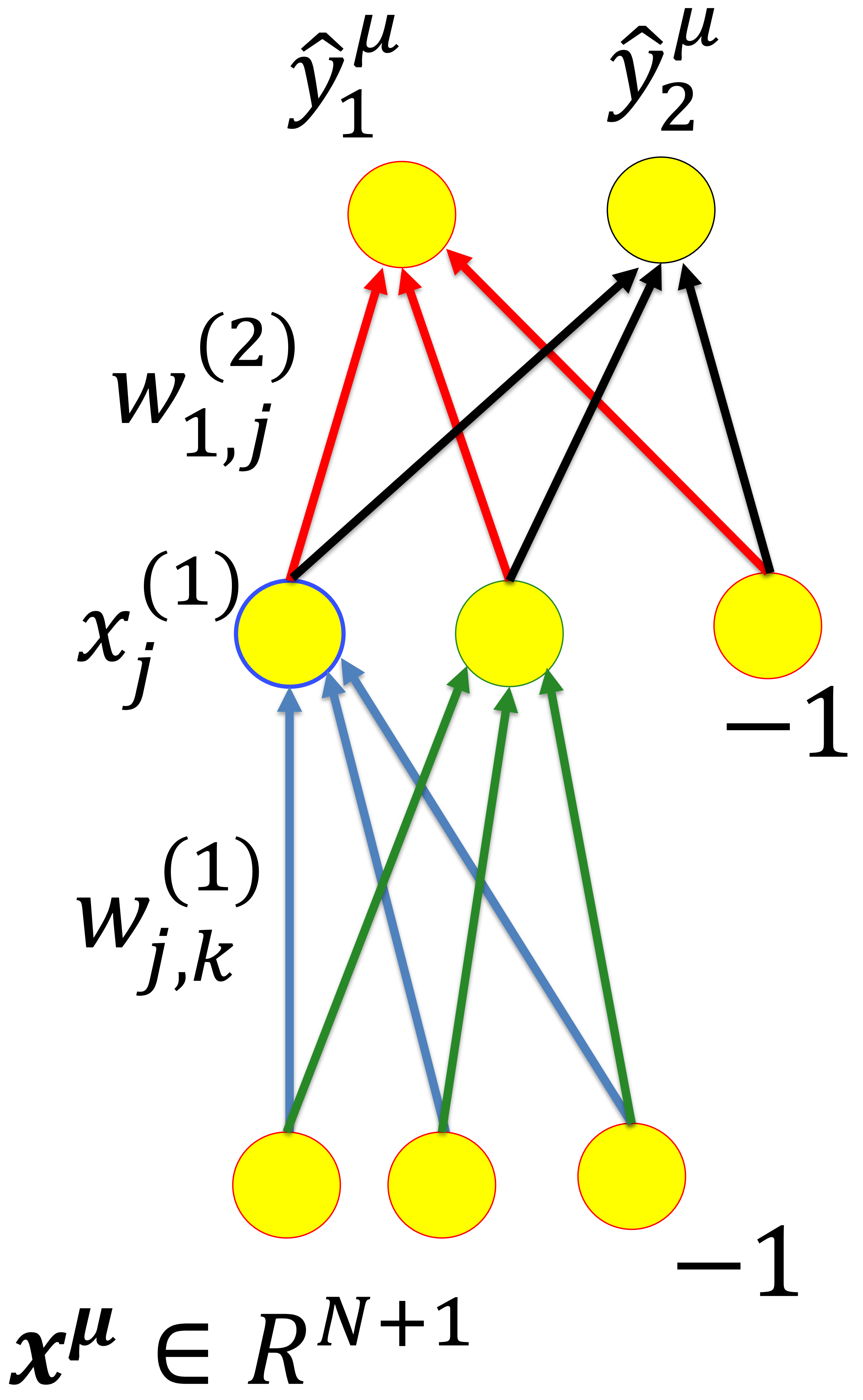
$$\hat{y}_i^{\mu} = x_i^{(2)} \quad (1)$$

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

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

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

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



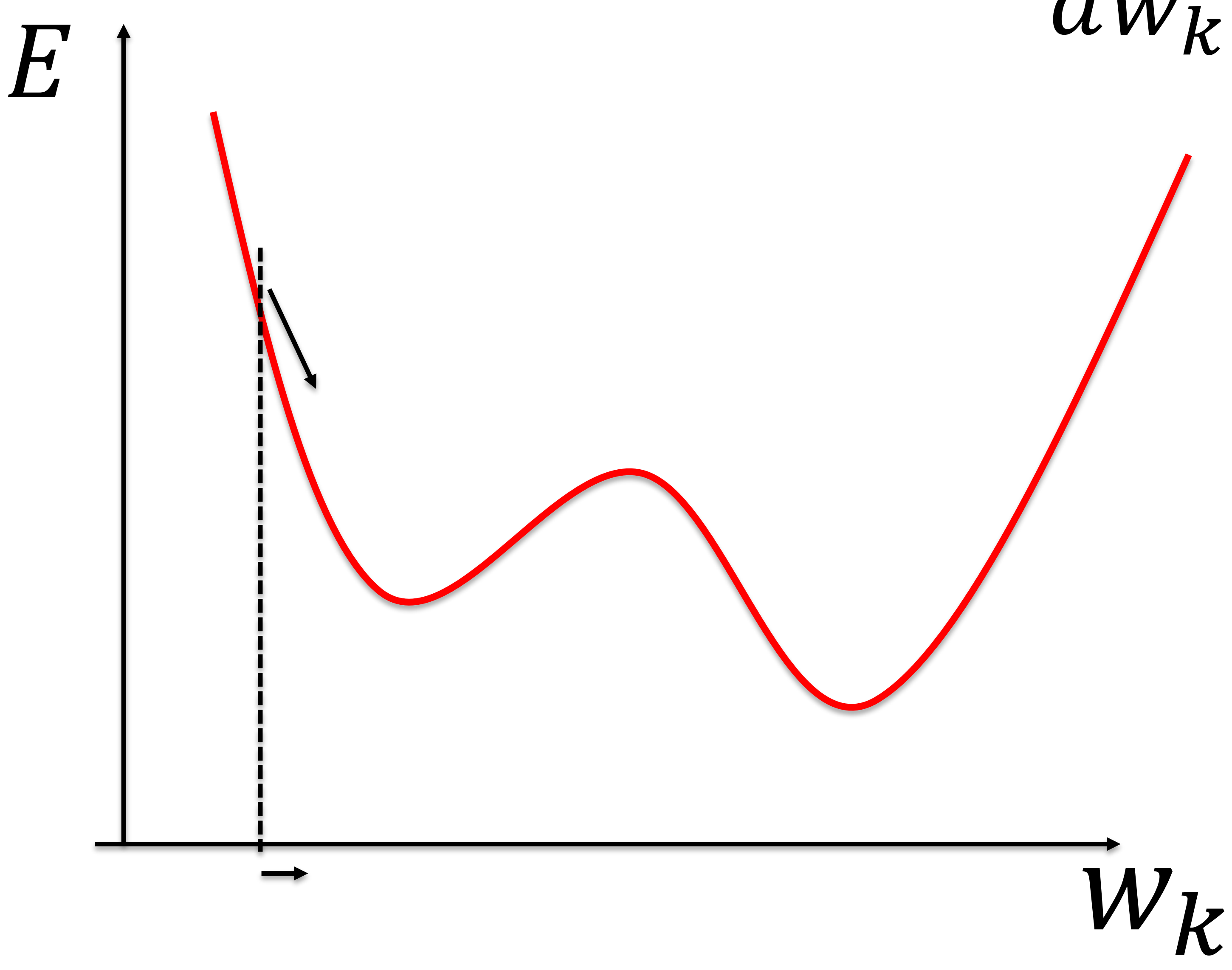
Review: gradient descent

Quadratic error

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

gradient descent

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



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

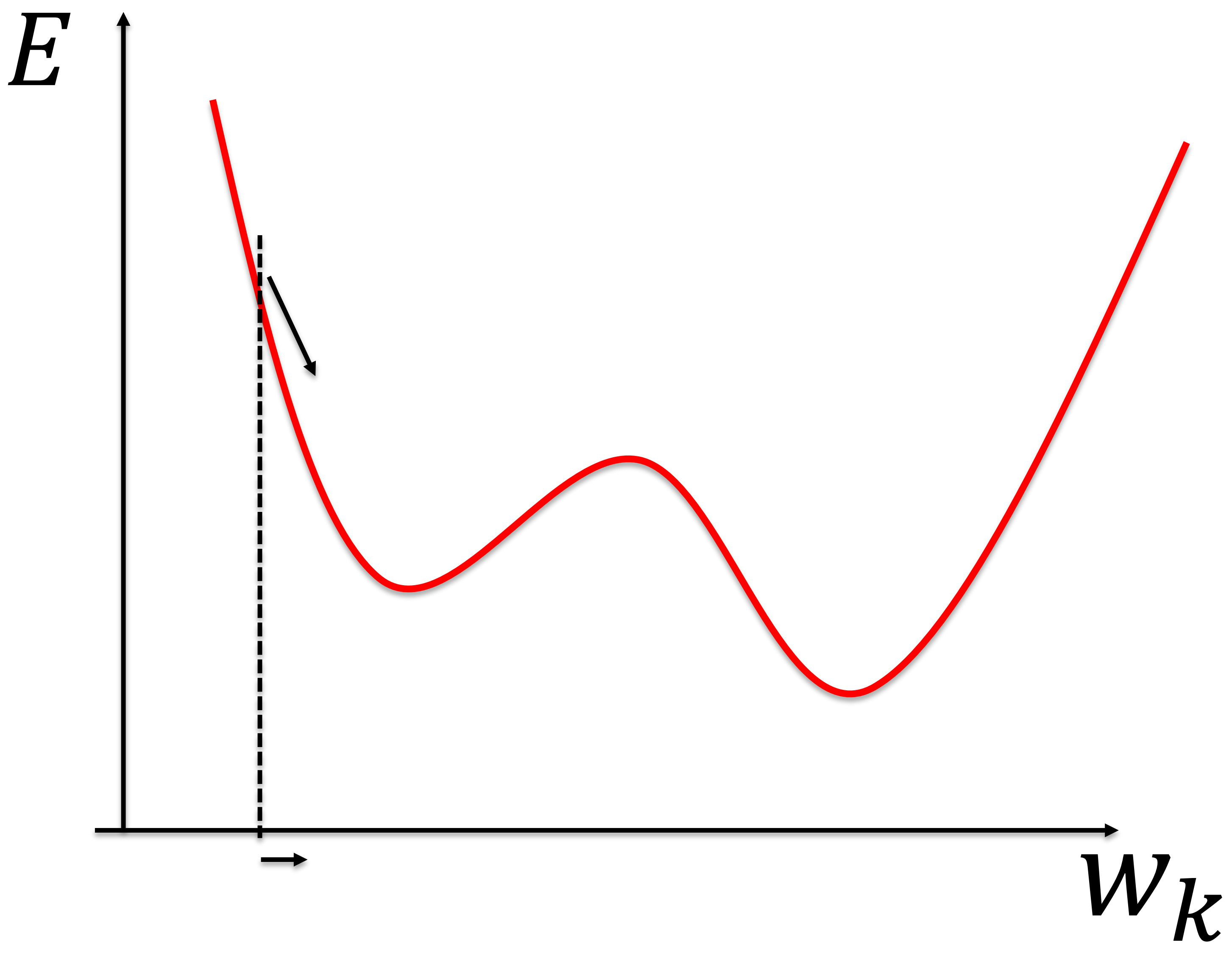
Online rule:
one update after one pattern
(stochastic gradient descent)

Same applies to all loss functions, e.g., **Cross-entropy error**

Three Big questions for today

- How does the error landscape (as a function of the weights) look like?
- How can we quickly find a (good) minimum?

- Why do deep networks work well?



Artificial Neural Networks: Lecture 5

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Error function and optimization methods for deep networks

Objectives for today:

- Error function: minima and saddle points

1. Error function: minima

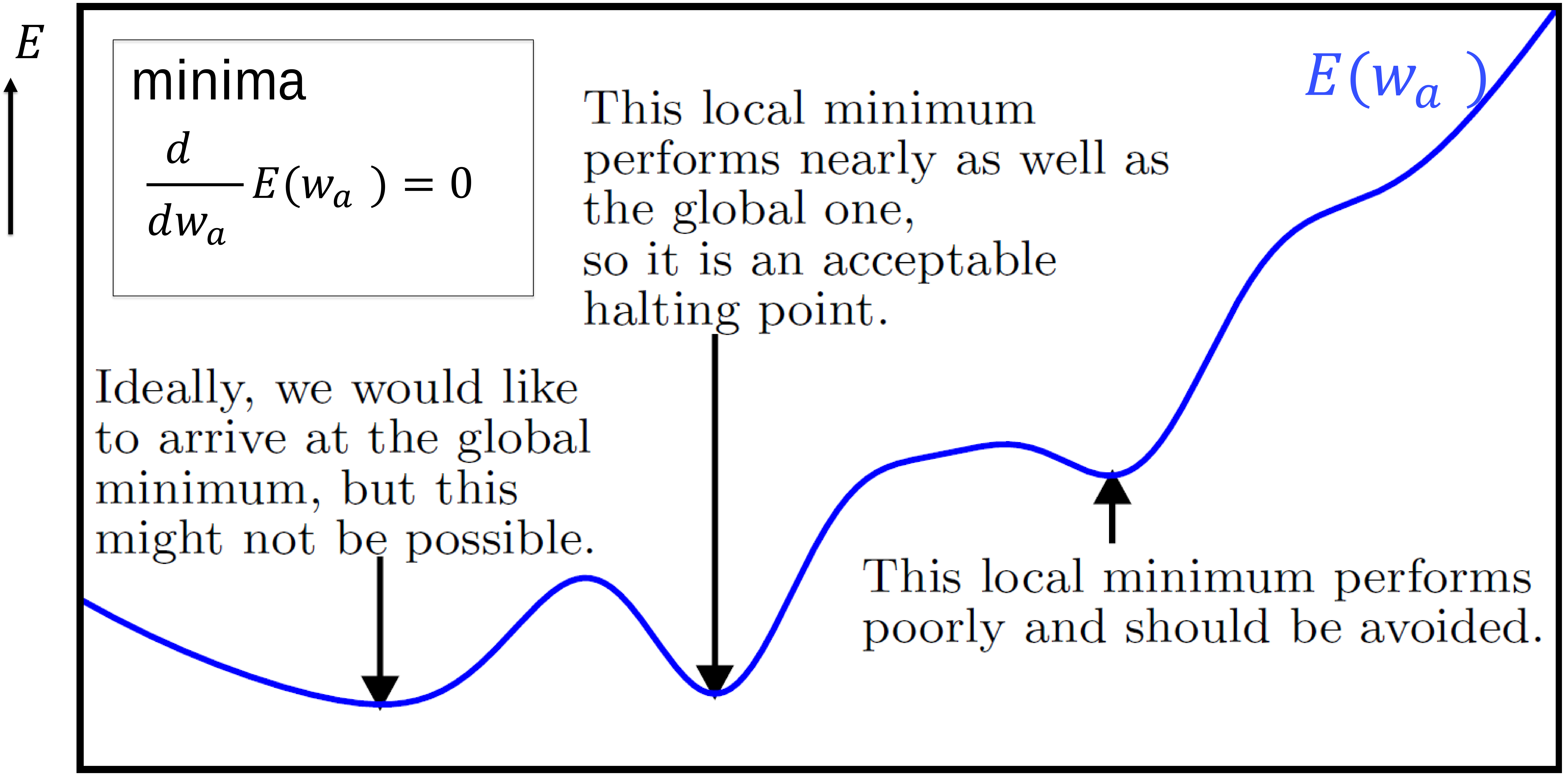


Image: Goodfellow et al. 2016

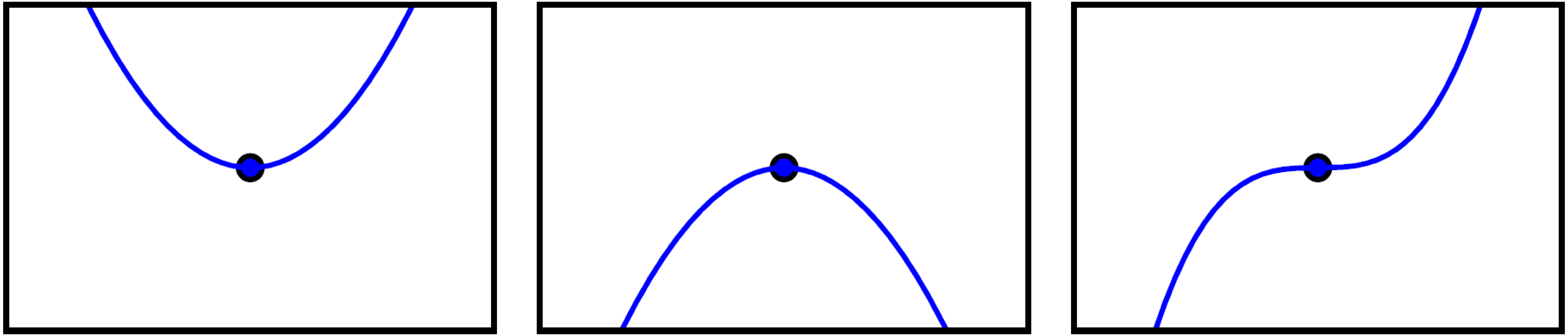
1. Error function: minima

How many minima are there in a deep network?

minima

$$\frac{d}{dw_a} E(w_a) = 0$$

Minimum　　　　　　Maximum



$$\frac{d^2}{dw_a^2} E(w_a) > 0 \qquad \frac{d^2}{dw_a^2} E(w_a) < 0 \qquad \frac{d^2}{dw_a^2} E(w_a) = 0$$

Image: Goodfellow et al. 2016

1. Error function: minima and saddle points

$$\frac{d^2}{dw_b^2} E(w_b) < 0 \qquad \frac{d^2}{dw_a^2} E(w_a) > 0$$

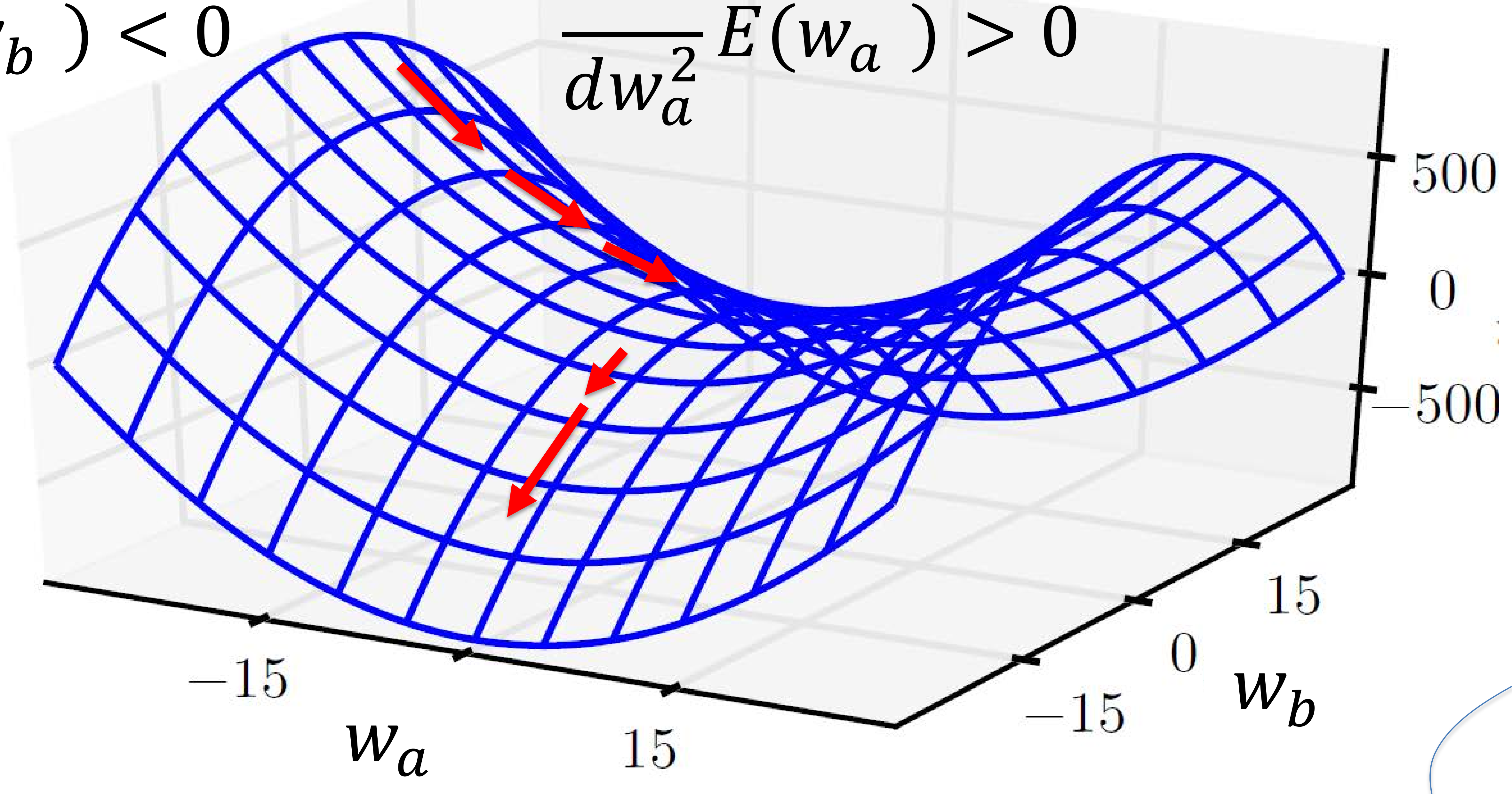
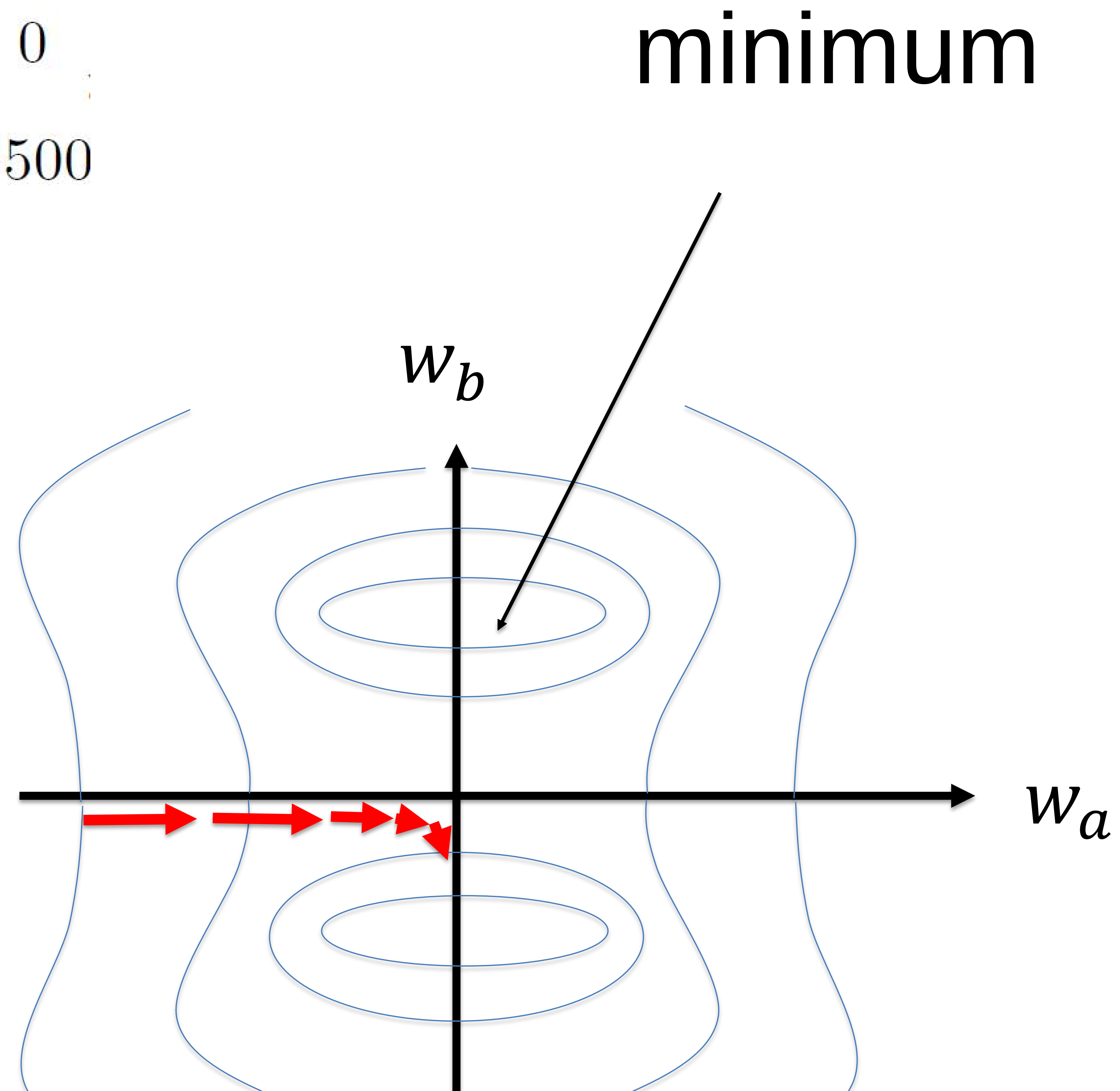


Image: Goodfellow et al. 2016



2 minima, separated by
1 saddle point

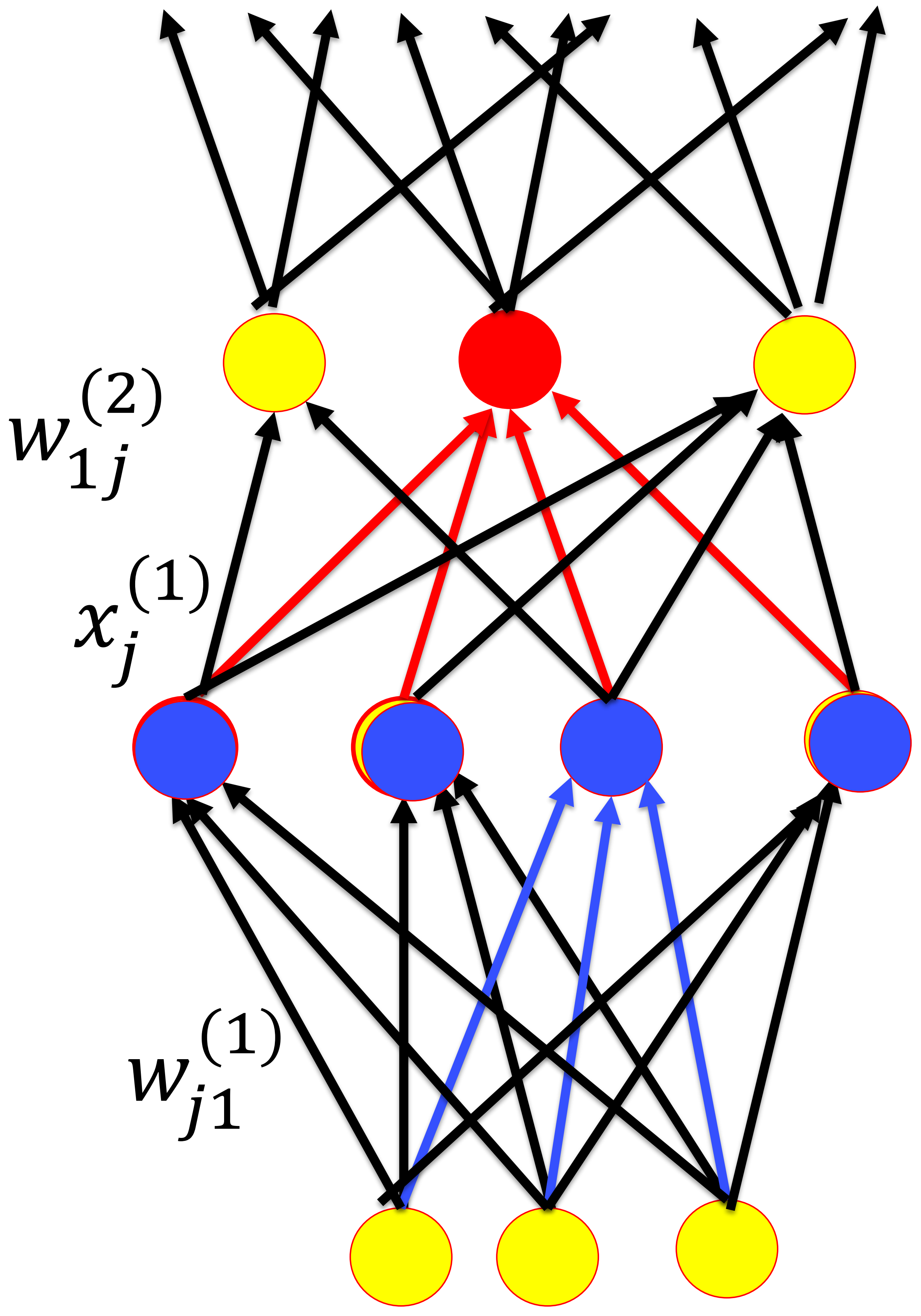
Quiz: Strengthen your intuitions in high dimensions

1. A deep neural network with 9 layers of 10 neurons each

[ ] has typically between 1 and 1000 minima (global or local)
[ ] has typically more than 1000 minima (global or local)

2. A deep neural network with 9 layers of 10 neurons each
[ ] has many minima and in addition a few saddle points
[ ] has many minima and about as many saddle points
[ ] has many minima and even many more saddle points

1. Error function

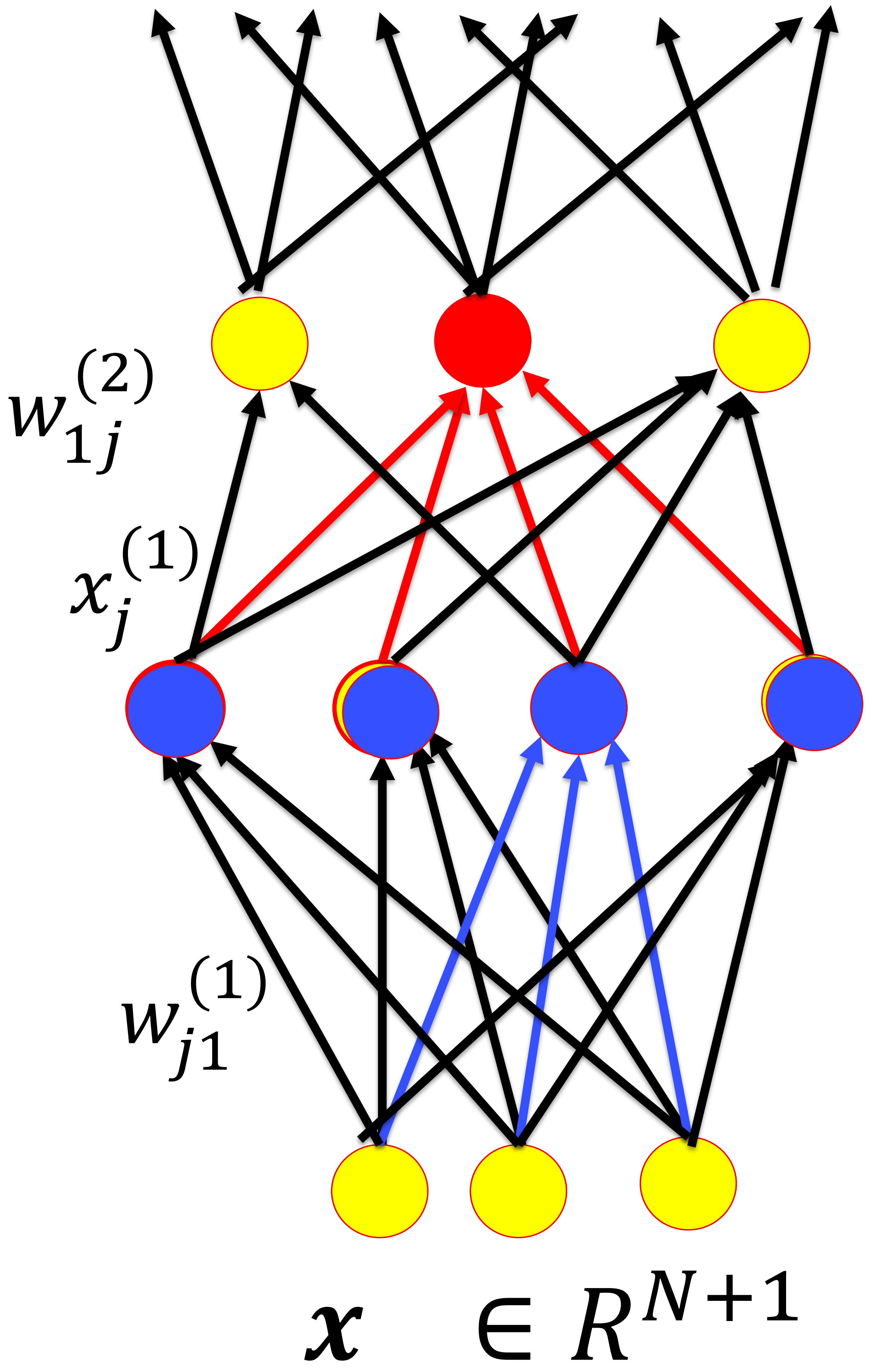


$\boldsymbol{x} \in R^{N+1}$

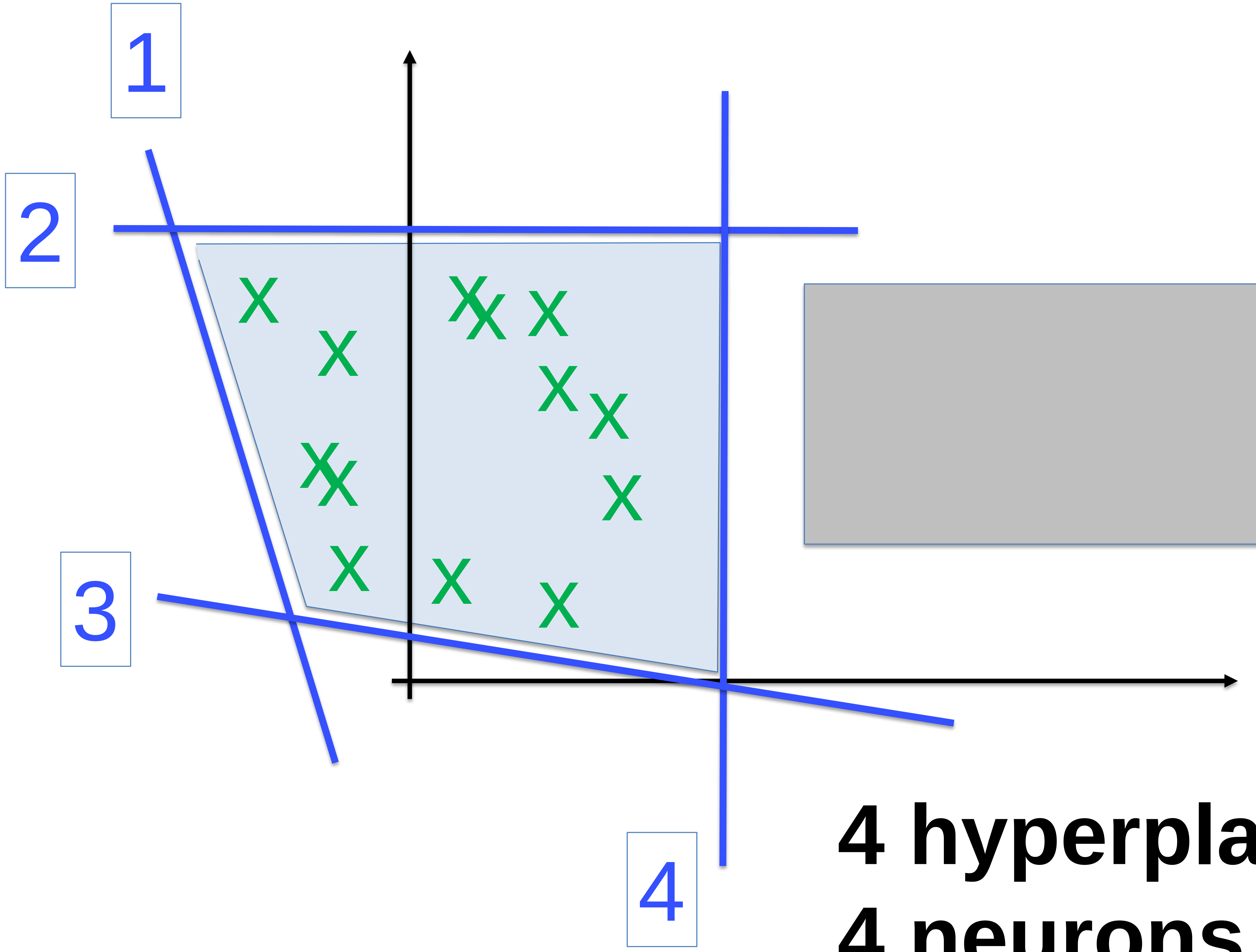
How many minima are there?

Answer:
In a network with n hidden layers and m neurons per hidden layer,

1. Error function and weight space symmetry

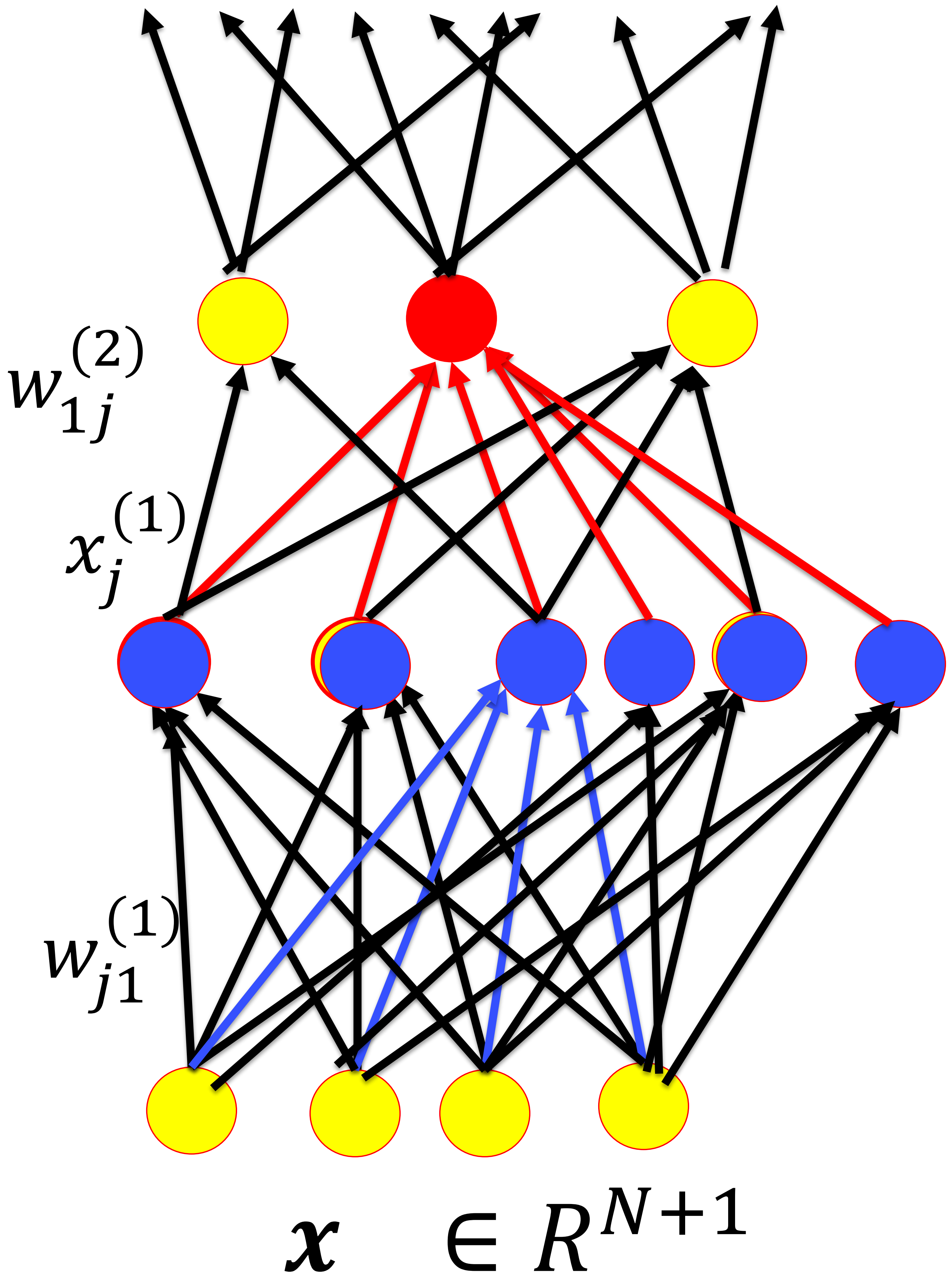


many assignments
of hyperplanes to neurons

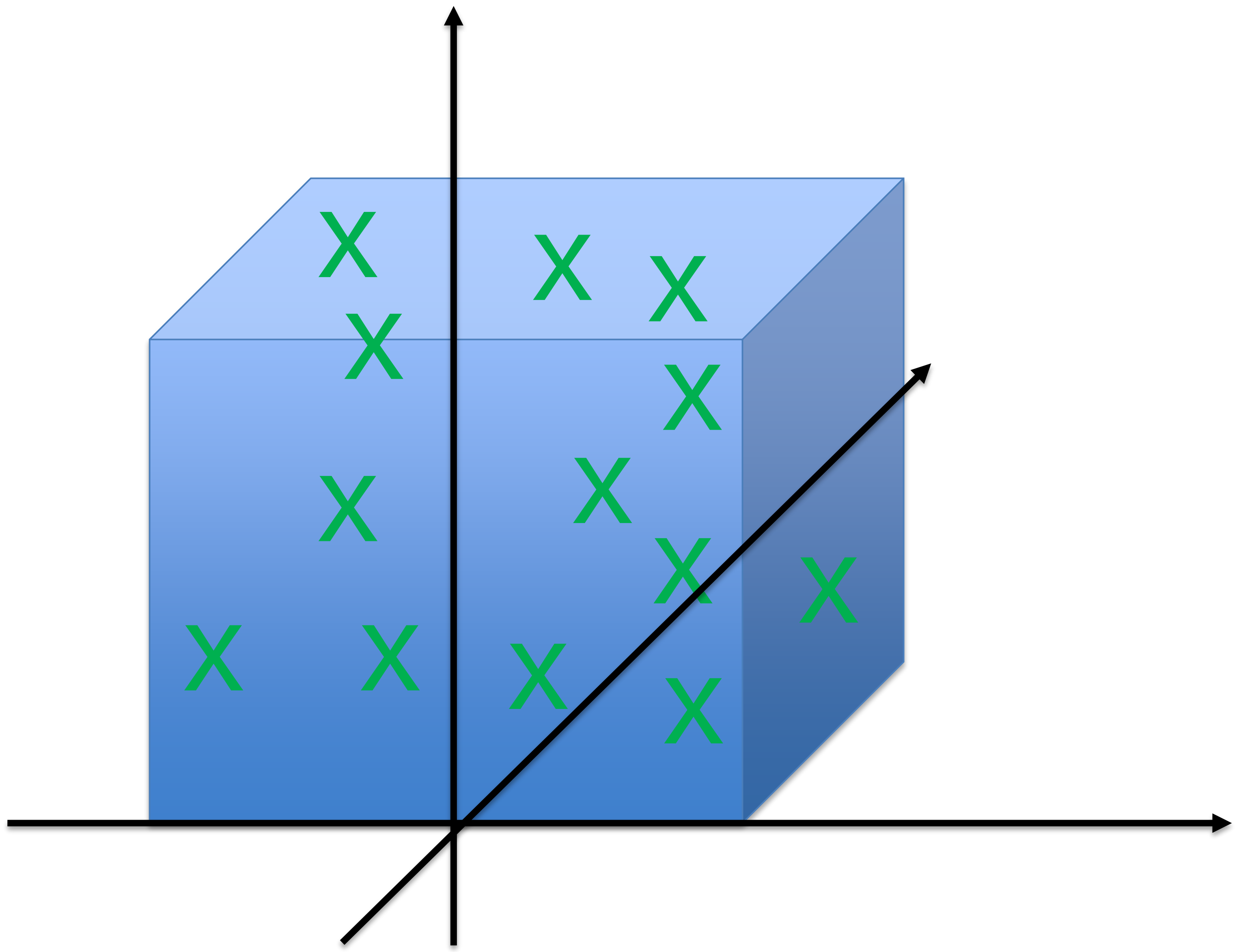


4 hyperplanes for
4 neurons

1. Error function and weight space symmetry



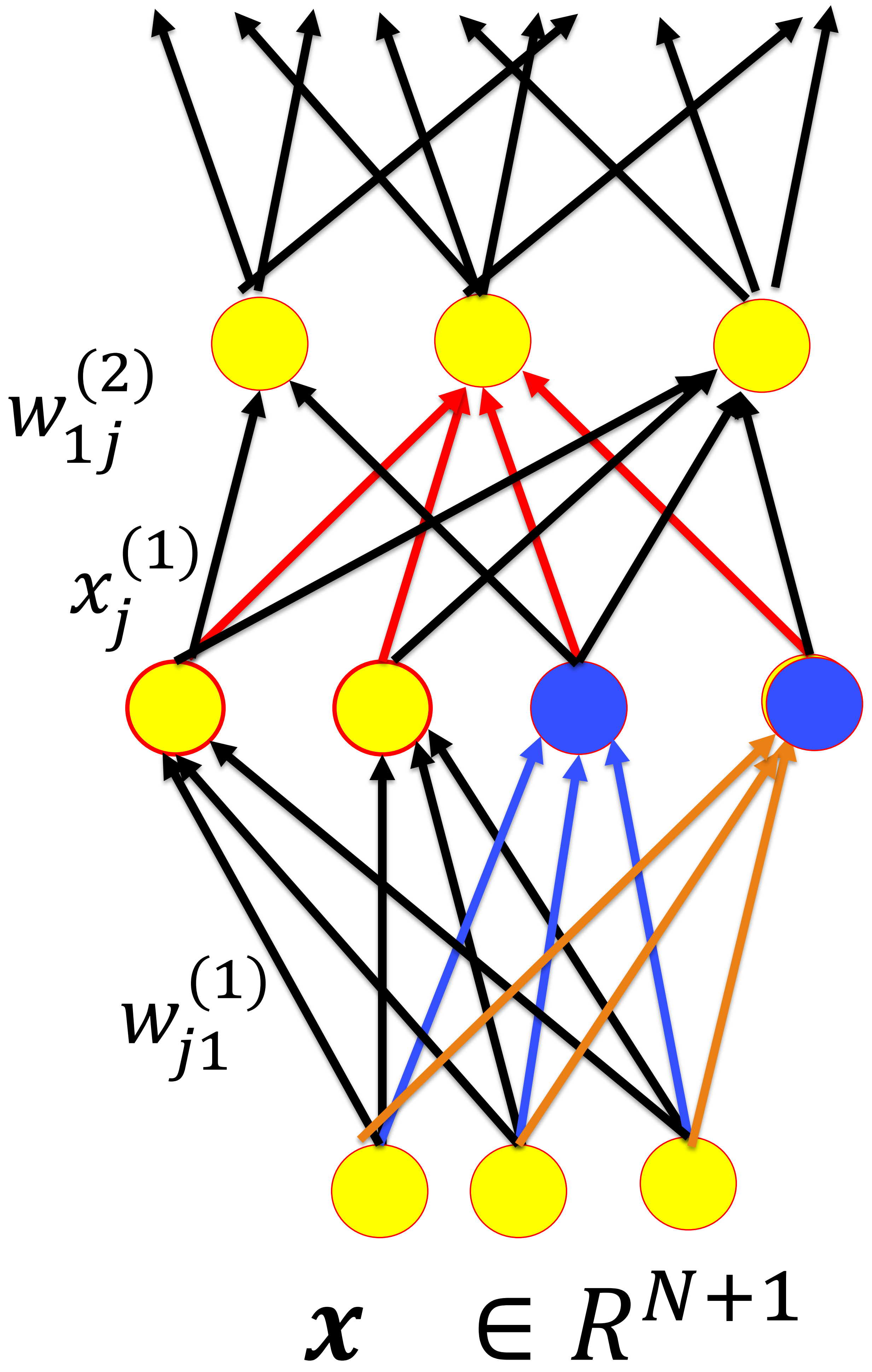
many assignments
of hyperplanes to neurons



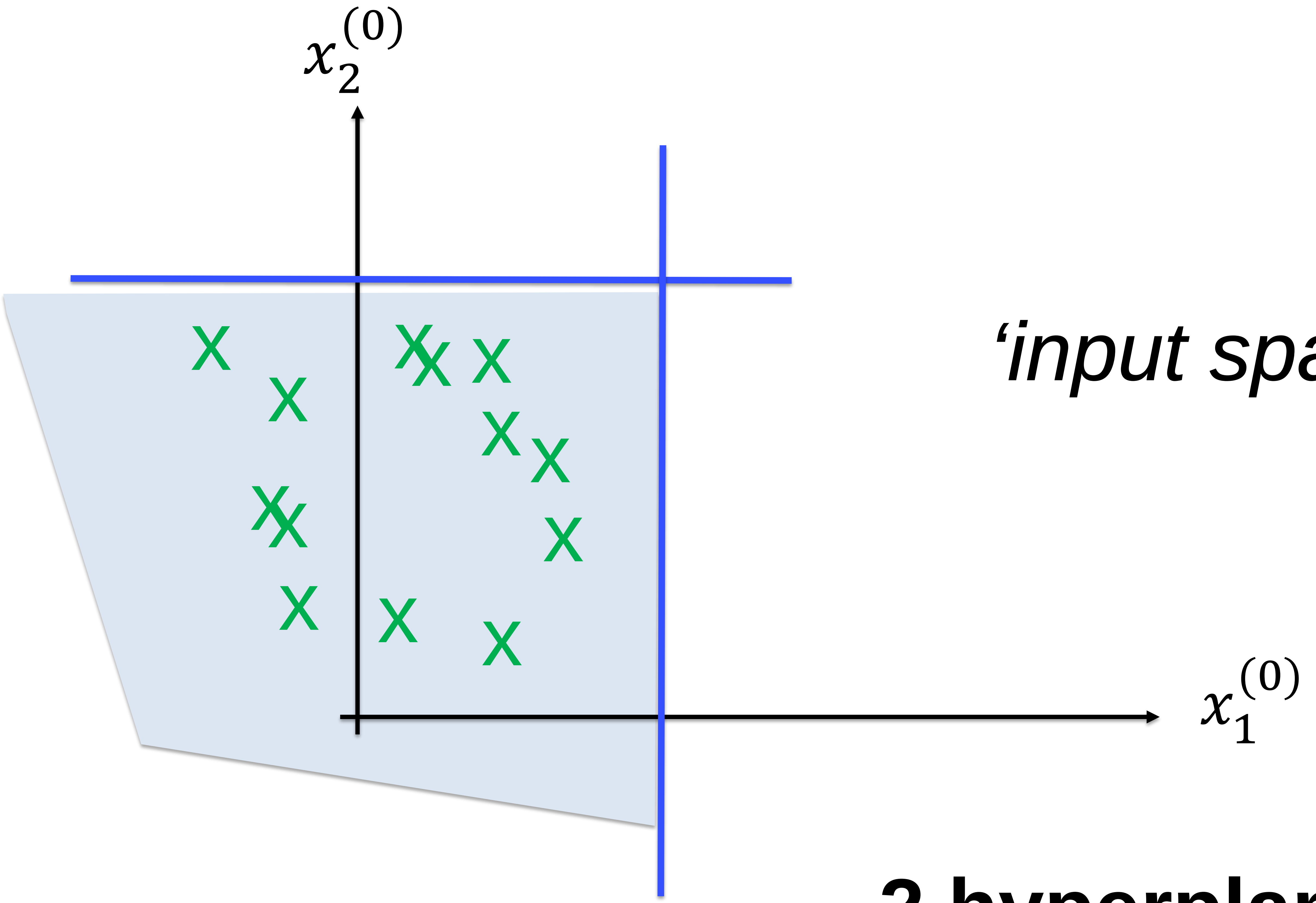
even more
permutations

6 hyperplanes for
6 hidden neurons

1. Minima and saddle points



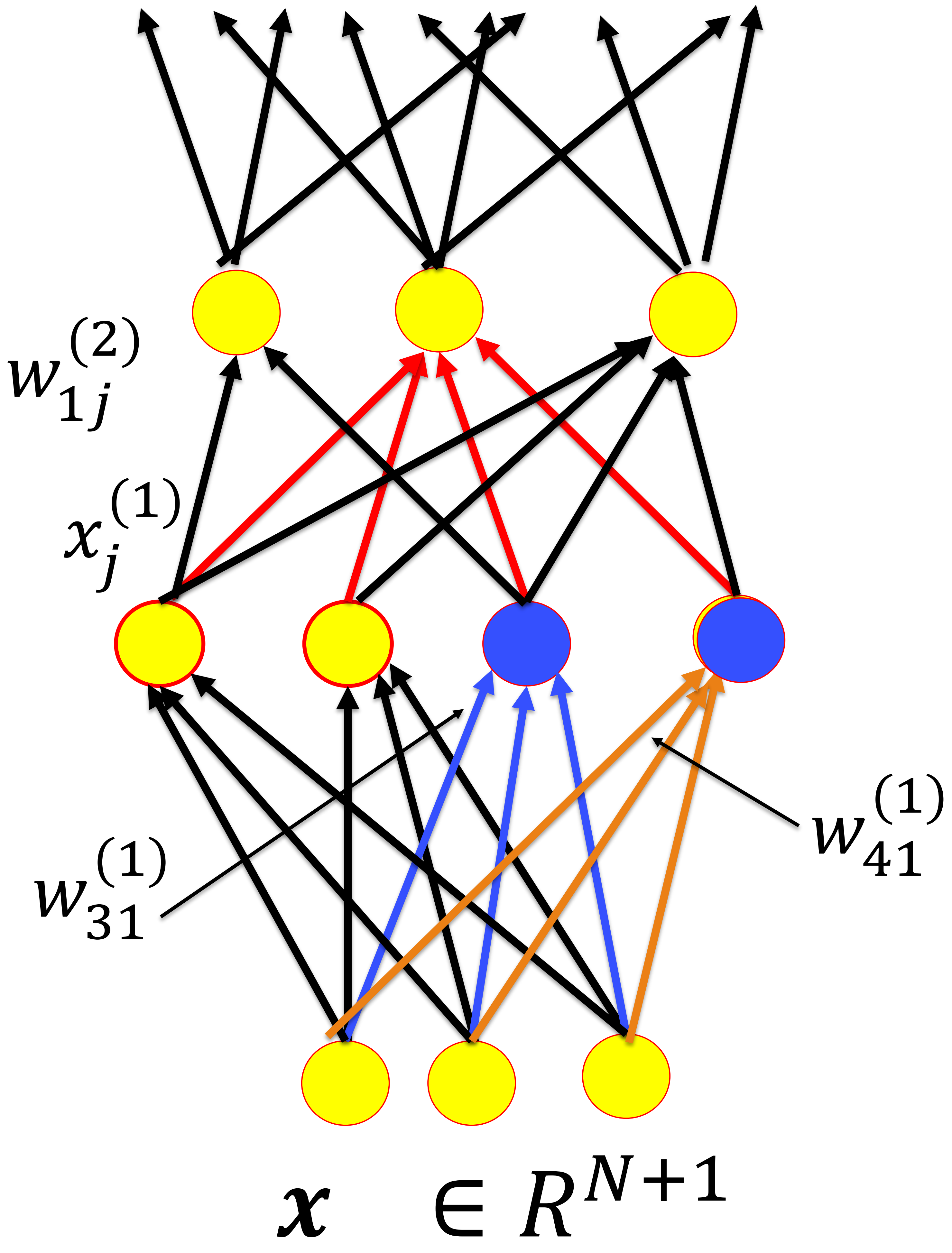
2 blue neurons
2 hyperplanes in input space



'input space'

2 hyperplanes

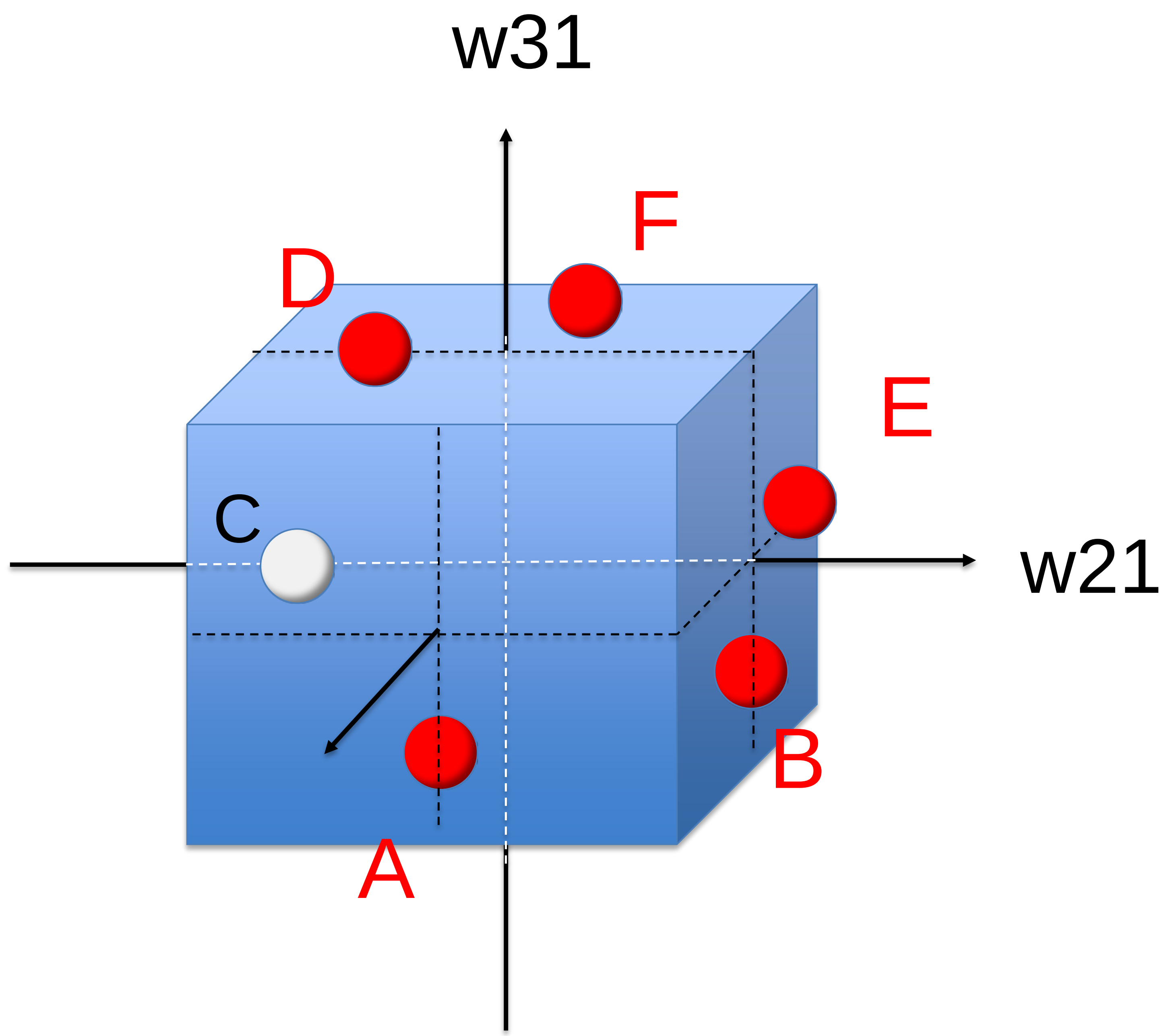
1. Error function and weight space symmetry



Blackboard 1

Solutions in weight space

1. Minima and saddle points in weight space



A = (1,0,-.7); C = (1,-.7,0)

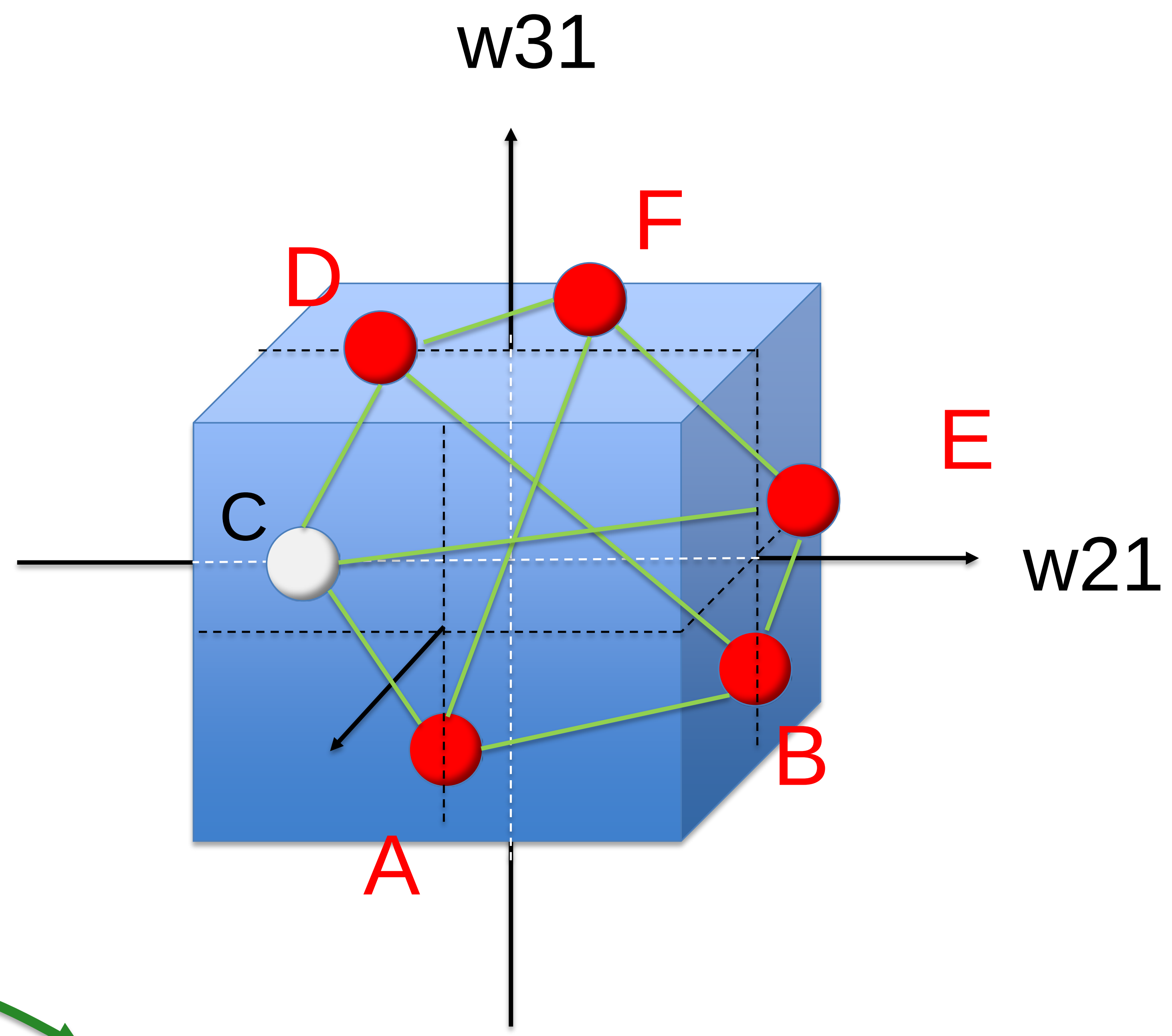
B = (0,1,-.7); D = (0,-.7,1)

E= (-.7,1,0); F = (-.7,0,1)

Algo for plot:

- Pick w11,w21,w31
- Adjust other parameters to minimize E

1. Minima and saddle points in weight space



Red (and white):
Minima

Green lines:
Run through saddles

Saddles:

A = (1,0,-.7); C = (1,-.7,0)

B = (0,1,-.7); D = (0,-.7,1)

E= (-.7,1,0); F = (-.7,0,1)

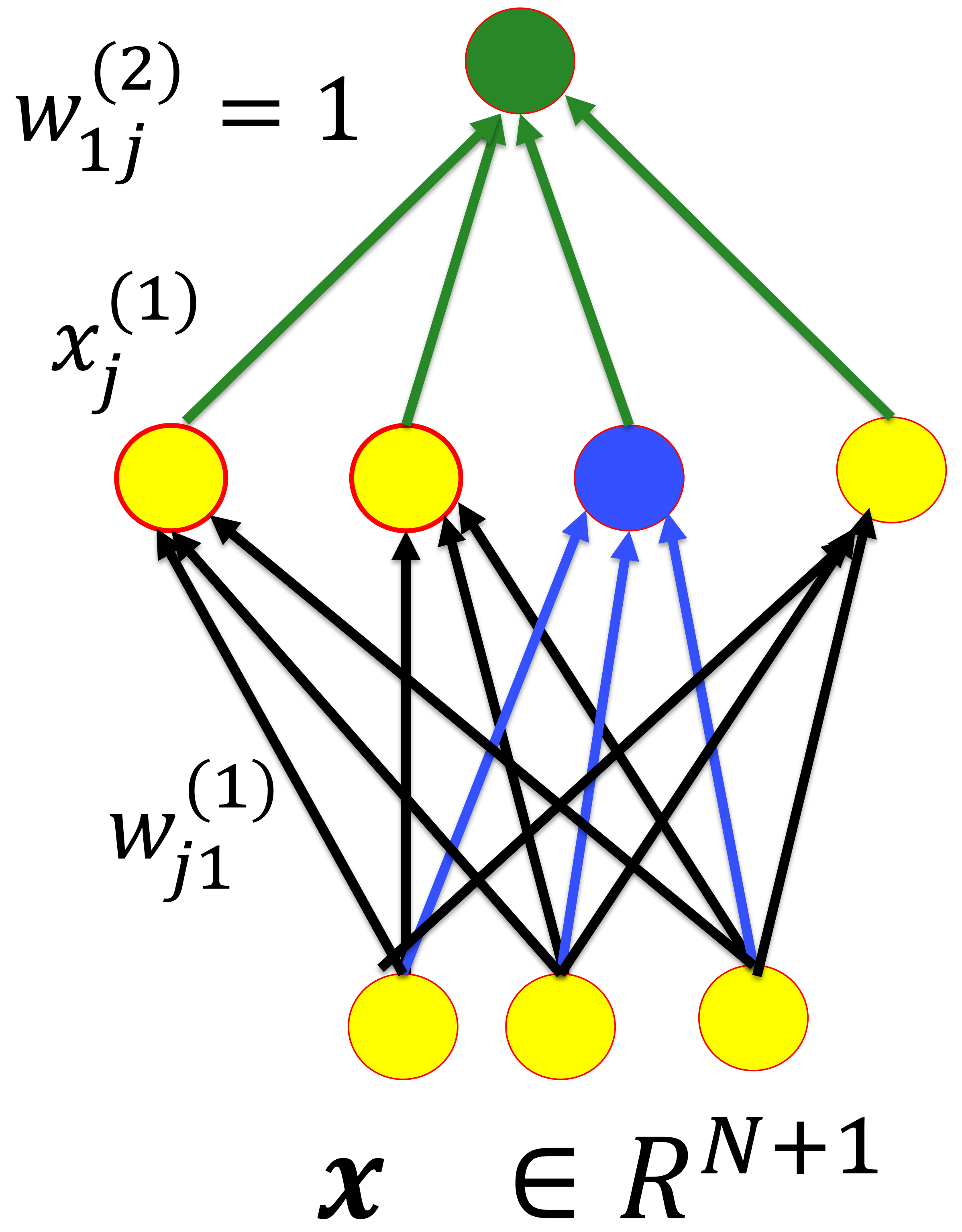
6 minima but >6 saddle points

1. Minima and saddle points: Example

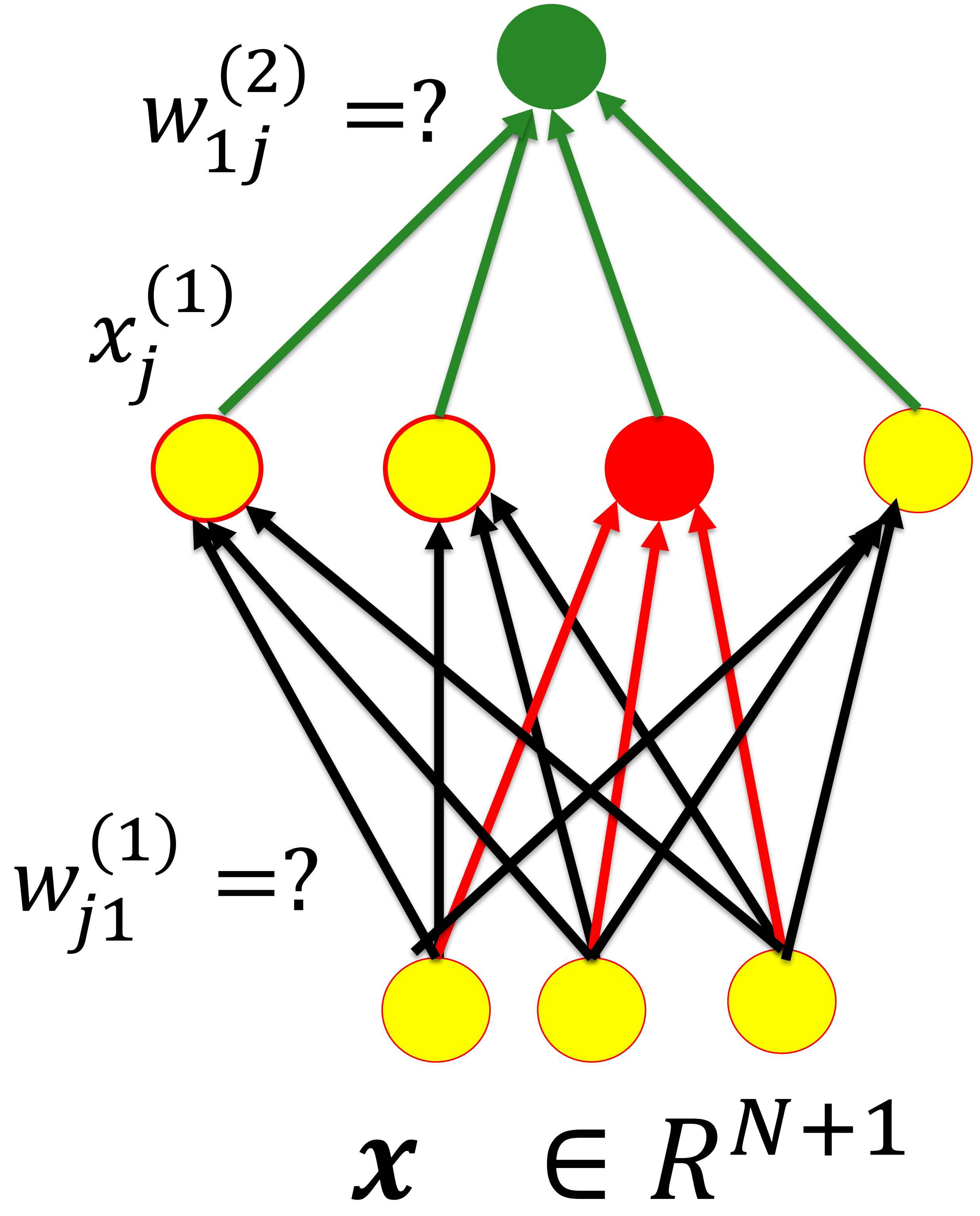
4 hyperplanes

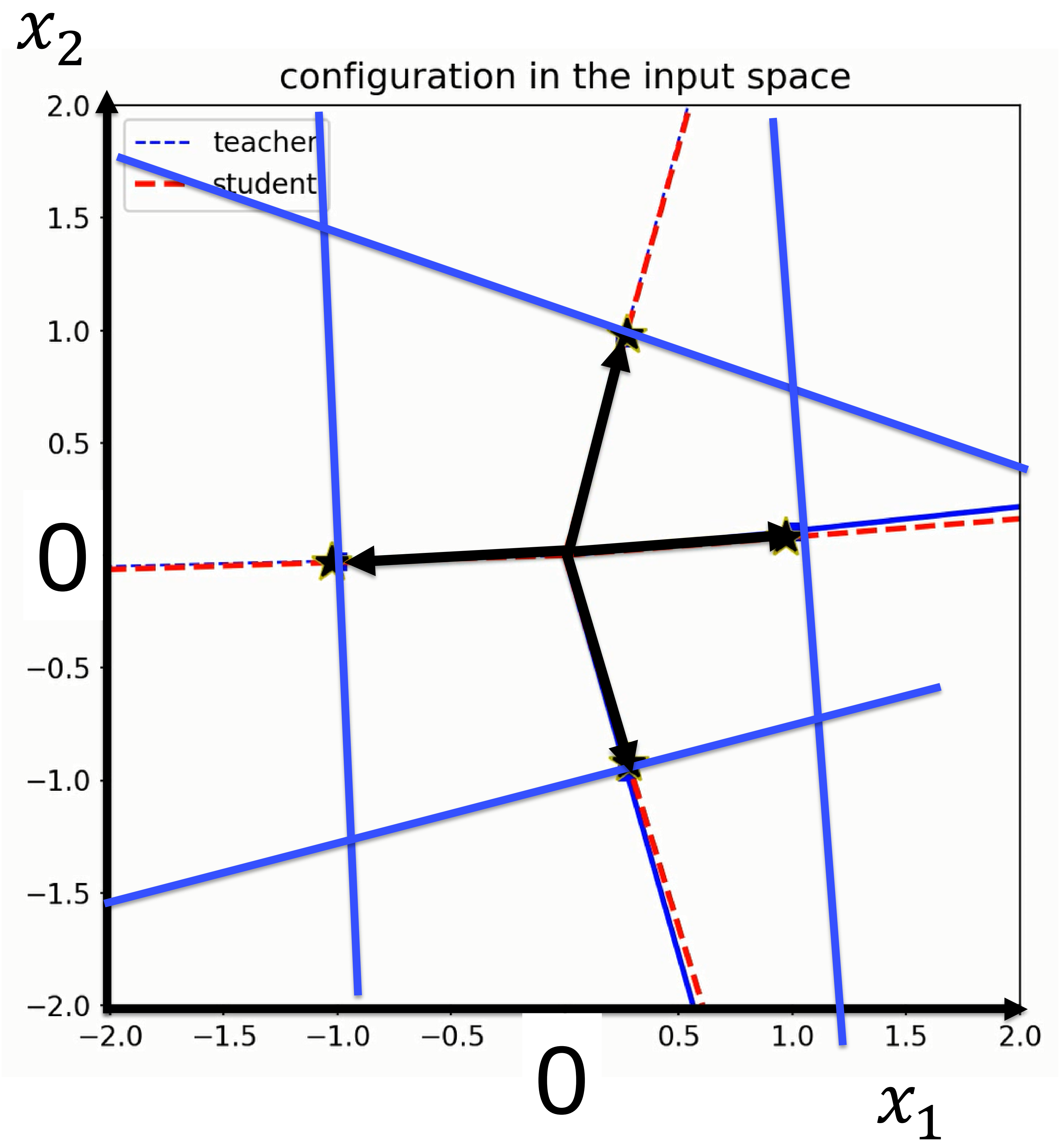
'input space'

Teacher Network:
Committee machine



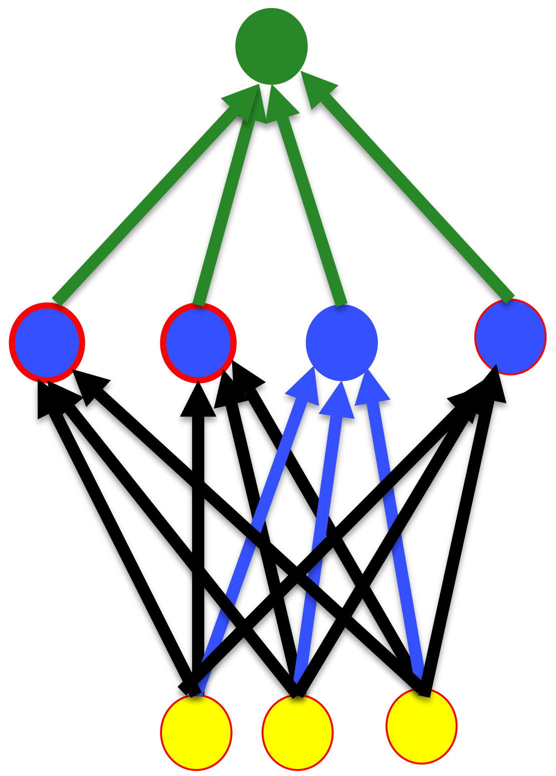
Student Network:



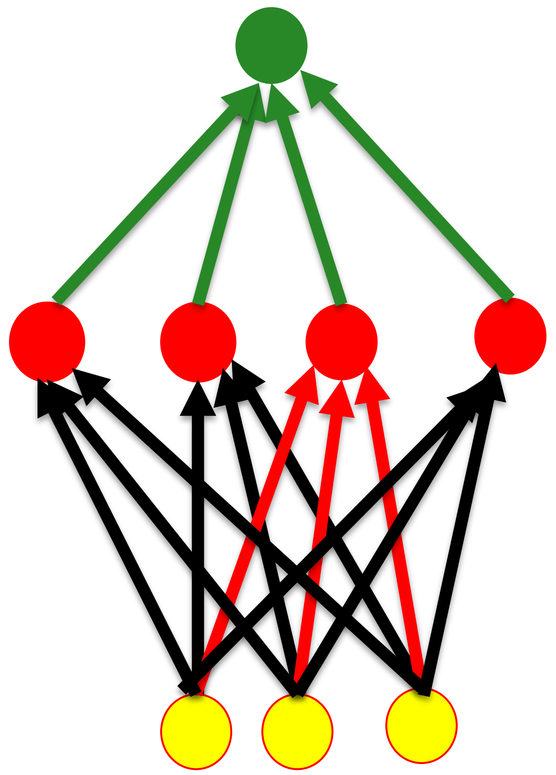


1. Minima and saddle points

Teacher Network: Blue

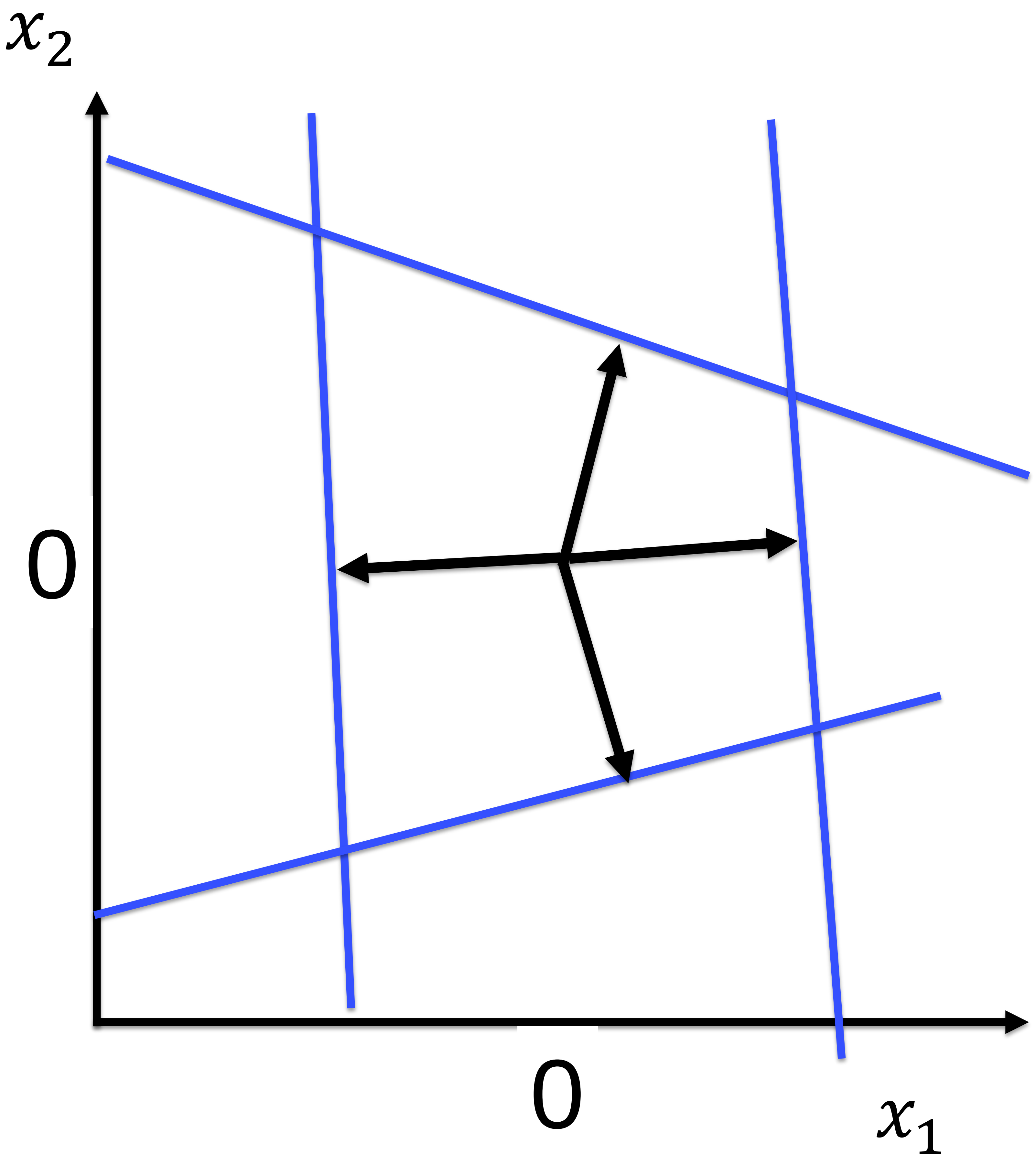


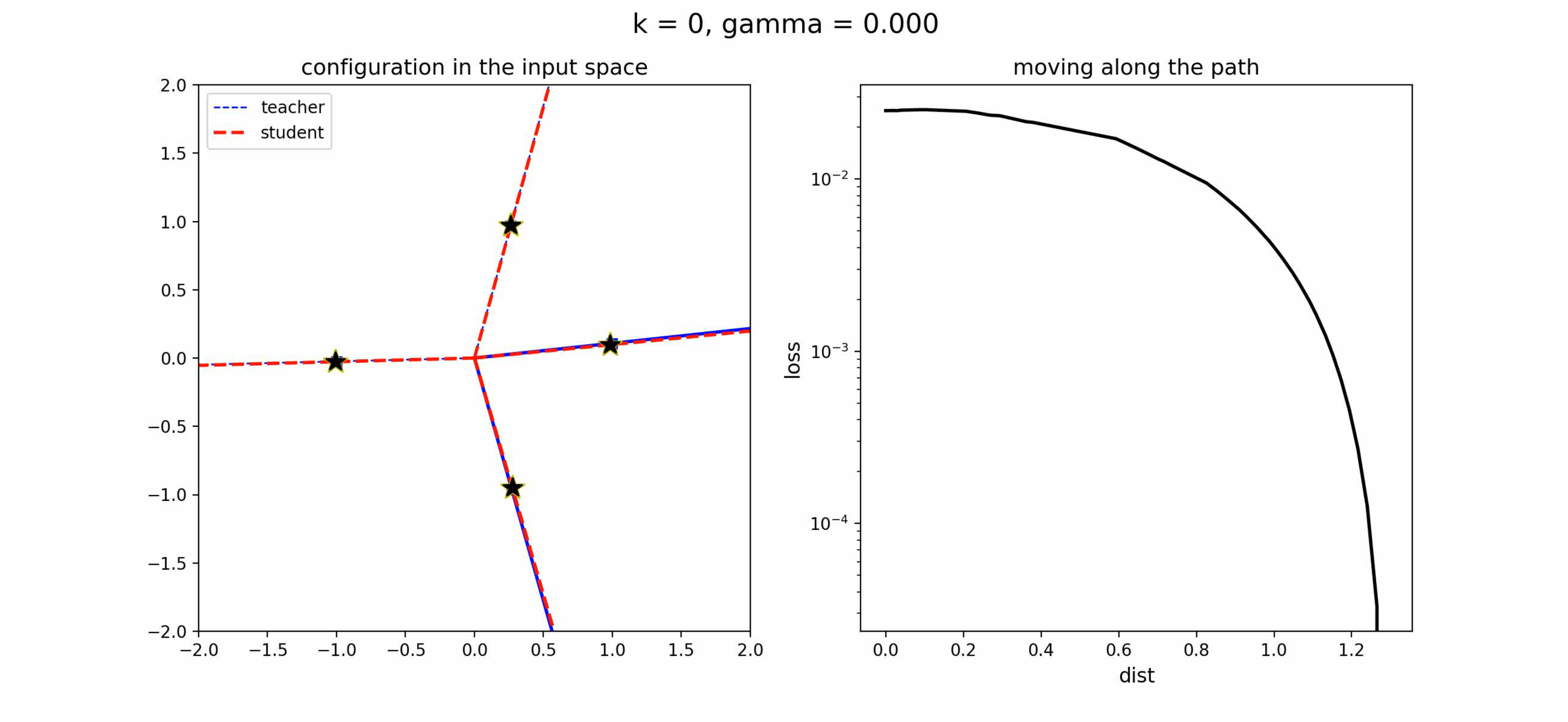
Student Network: Red



4 hyperplanes

'input space'





1. Minima and saddle points

There are many more saddle points than minima

Two arguments

(i) Geometric argument and weight space symmetry

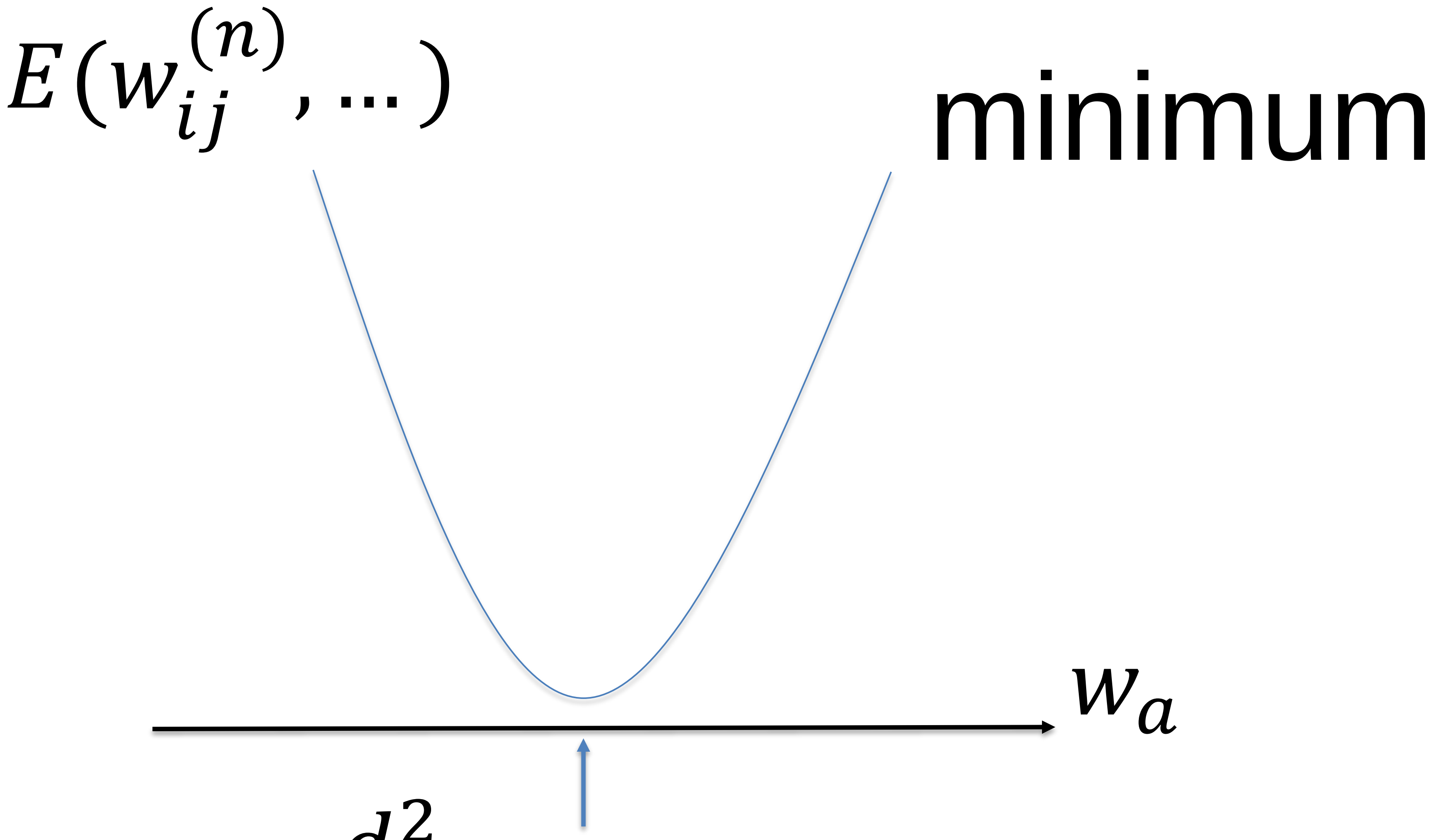
→ number of saddle points increases
rapidly with dimension
(much more rapidly than the number of minima)

1. Minima and saddle points

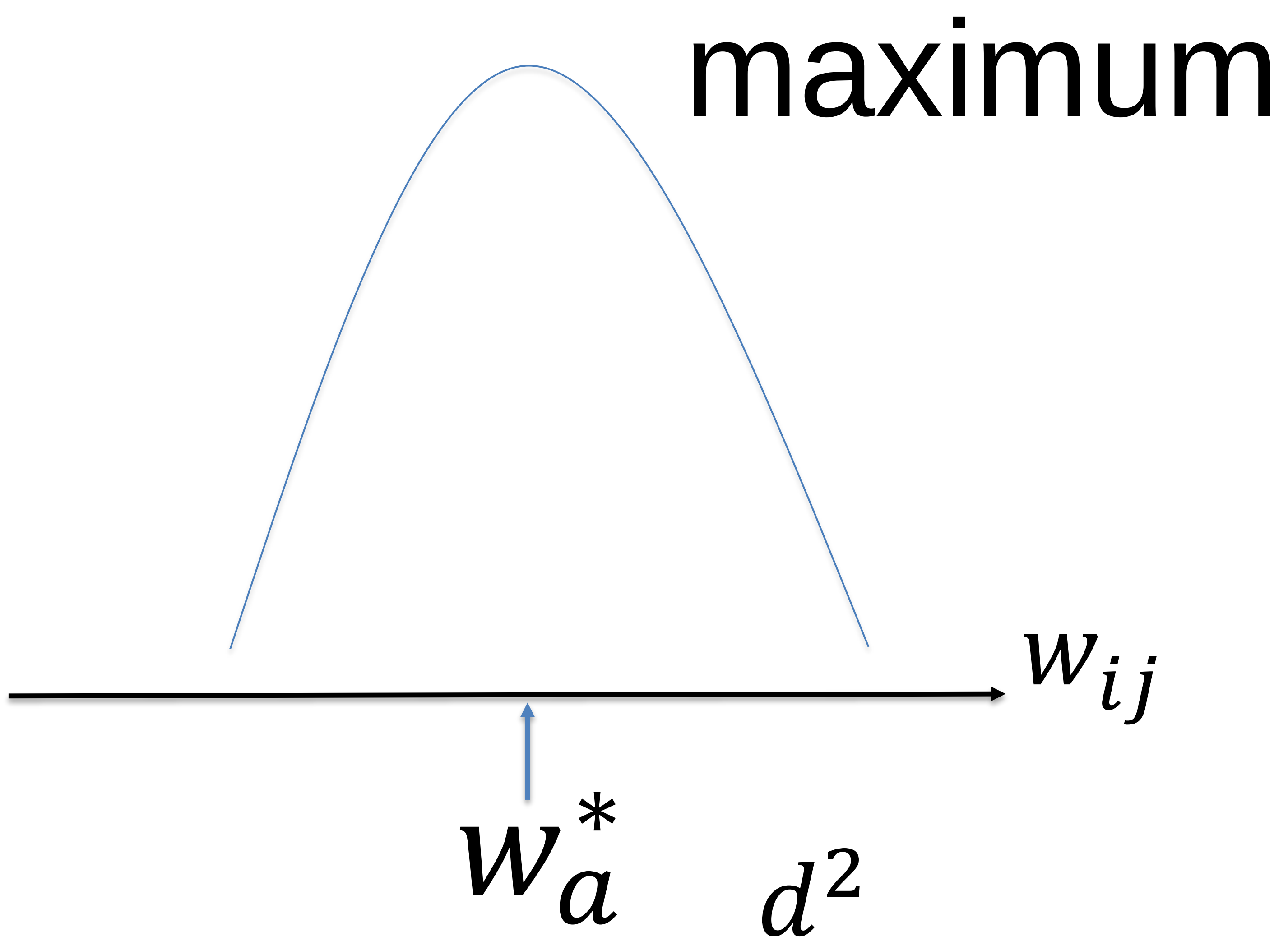
There are many more saddle points than minima

Two arguments

(ii) Second derivative (Hessian matrix) at gradient zero



$$\frac{d^2}{dw_a^2} E(w_a^*) > 0$$



$$\frac{d^2}{dw_a^2} E(w_a^*) < 0$$

1. Minima and saddle points

In 1dim: at a point with vanishing gradient

$$\frac{d^2}{dw_a^2} E(w_a) > 0 \qquad \rightarrow \text{minimum}$$

Minimum in N dim: study **Hessian**

$$\mathrm{H} = \frac{d}{dw_a} \frac{d}{dw_b} E(w_a, w_b)$$

Diagonalize: minimum if all eigenvalues positive.
But for N dimensions, this is a strong condition!

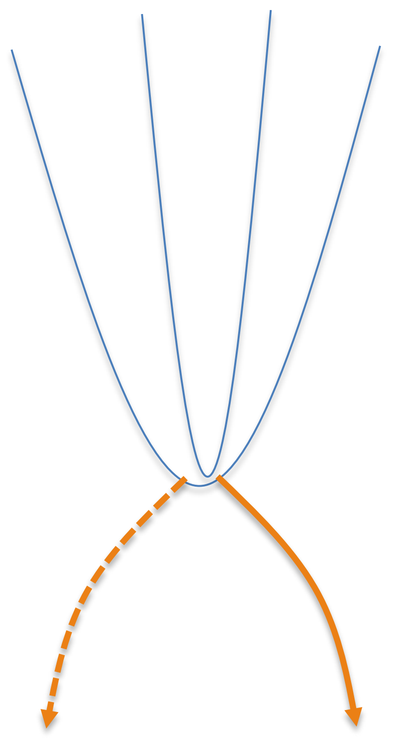
1. Minima and saddle points

in N dim: **Hessian**

$$H = \frac{d}{dw_a}\frac{d}{dw_b}E(w_a, w_b)$$

Diagonalize:

$$H = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_N \end{pmatrix}$$



$\lambda_1 > 0$

$\ldots$

$\lambda_{N-1} > 0$

$\lambda_N < 0$

In N-1 dimensions surface goes up,

In 1 dimension it goes down

1. Minima and saddle points

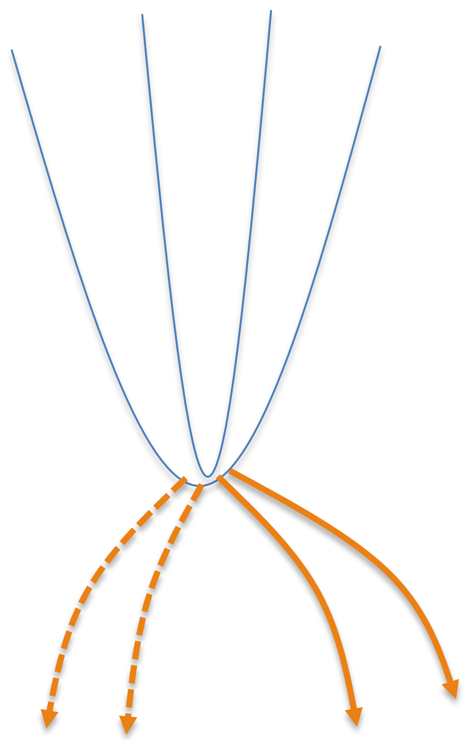
in N dim: **Hessian**

$$H = \frac{d}{dw_a}\frac{d}{dw_b}E(w_a, w_b)$$

Diagonalize:

$$H = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_N \end{pmatrix}$$

In N-m dimensions
surface goes up,
In m dimension it goes down



$\lambda_1 > 0$

...

$\lambda_{N-2} > 0$

$\lambda_{N-1} < 0$

$\lambda_N < 0$

In N-2 dimensions
surface goes up,
In 2 dimension it goes down

Kant!

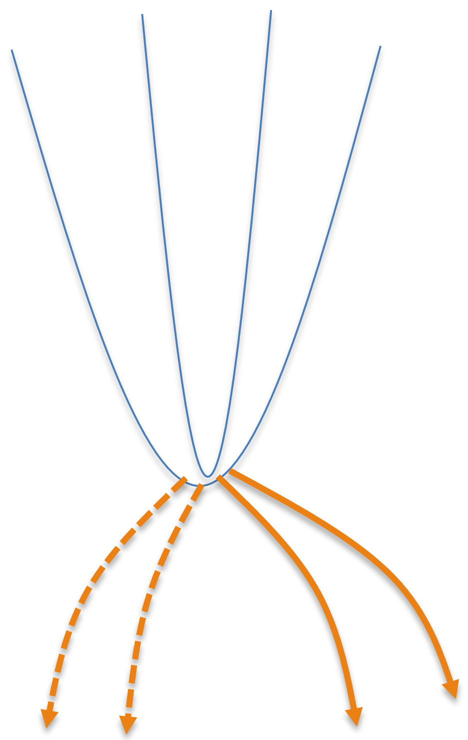
1. General saddle point

in N dim: **Hessian**

$$H = \frac{d}{dw_a}\frac{d}{dw_b}E(w_a, w_b)$$

Diagonalize:

$$H = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_N \end{pmatrix}$$



$\lambda_1 > 0$

...

$\lambda_{N-m+1} > 0$

$\lambda_{N-m} < 0$

$\lambda_N < 0$

In N-m dimensions surface goes up,
In m dimension it goes down

General saddle:
In N-m dimensions surface goes up,
In m dimension it goes down

1. Minima and saddle points

It is rare that all eigenvalues of the Hessian have same sign

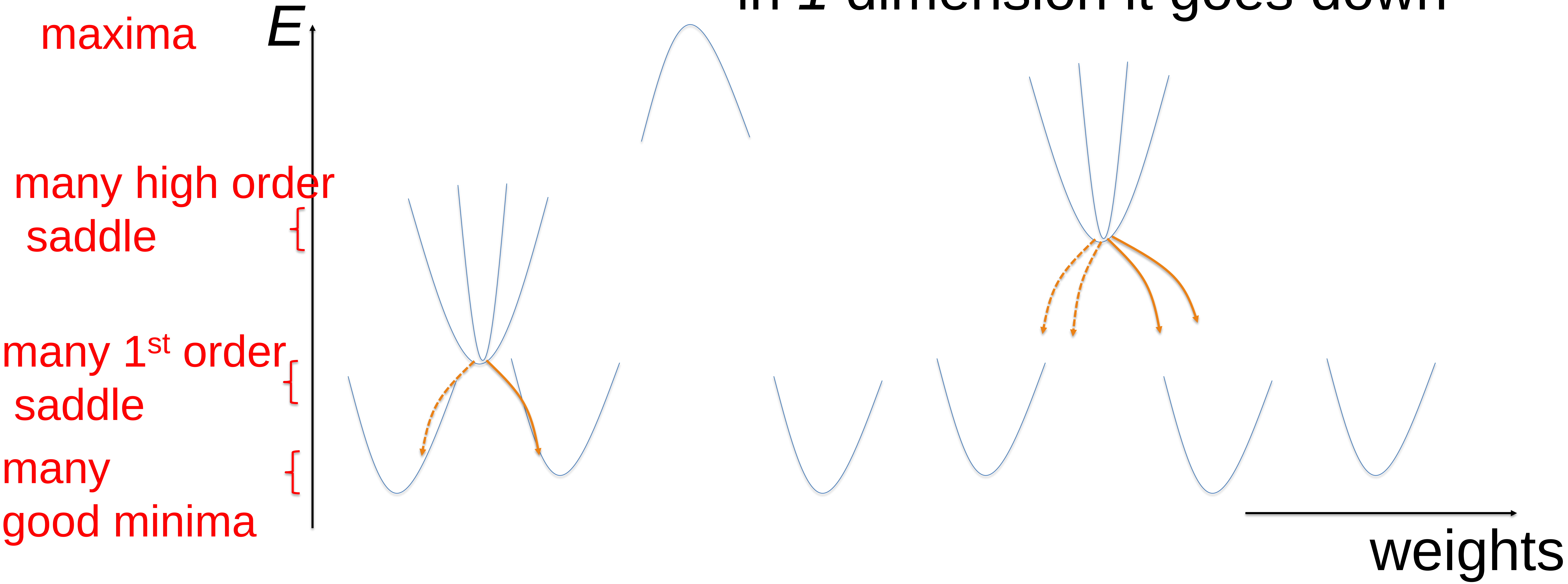
It is fairly rare that only one eigenvalue has a different sign than the others

→ Most saddle points have multiple dimensions with surface up and multiple with surface going down

1. Minima and saddle points: modern view

General saddle points: In N-m dimensions surface goes up,
in m dimension it goes down

1st-order saddle points: In N-1 dimensions surface goes up,
in *1* dimension it goes down

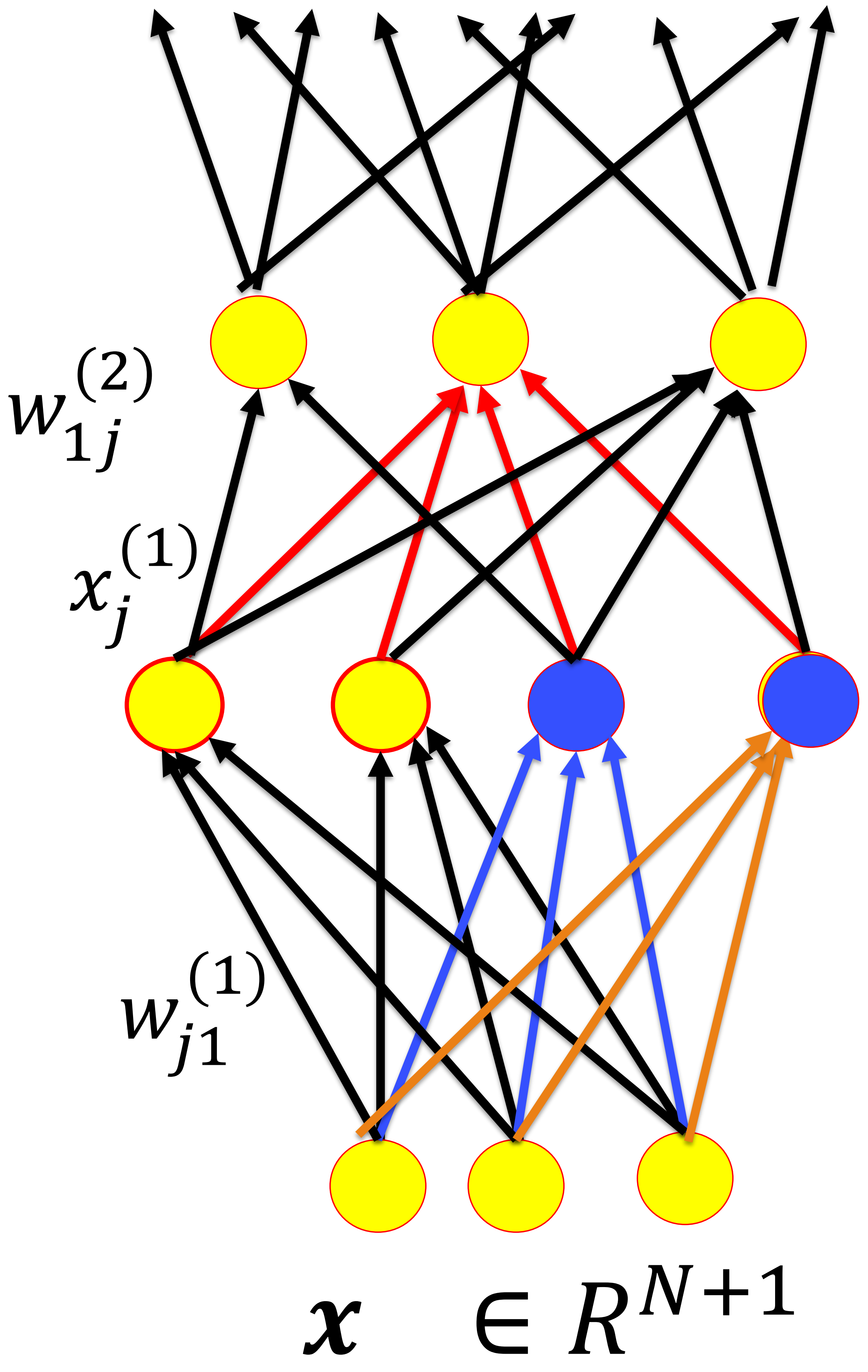


1. Minima and saddle points

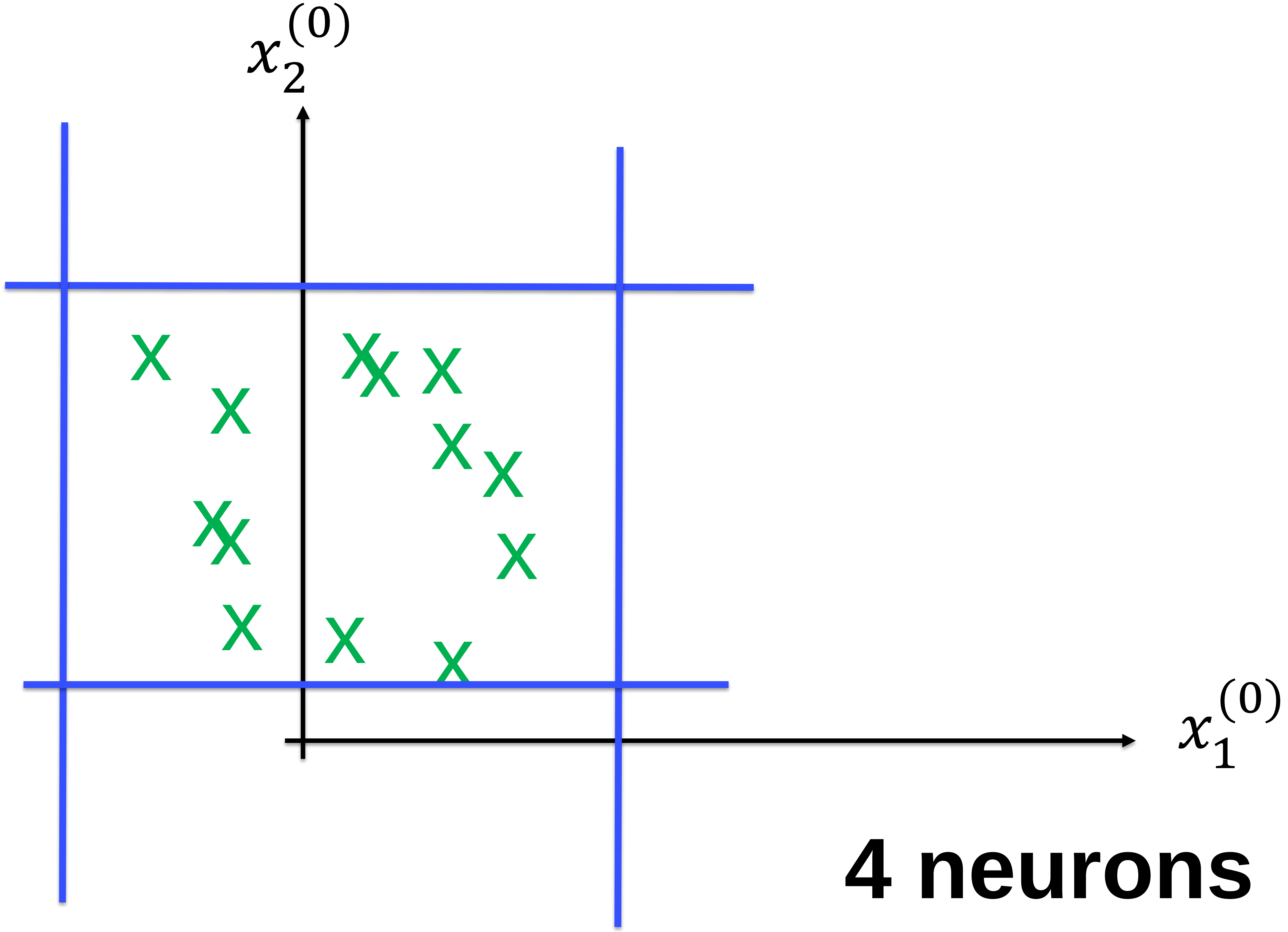
(ii) For balance random systems, eigenvalues will be randomly distributed with zero mean:
draw N random numbers
→ rare to have all positive or all negative
→ Rare to have maxima or minima
→ **Most points of vanishing gradient are saddle points**
→ **Most high-error saddle points have multiple directions of escape**

But what is the random system here?
The data is 'random' with respect to the design of the system!

1. Minima = good solutions

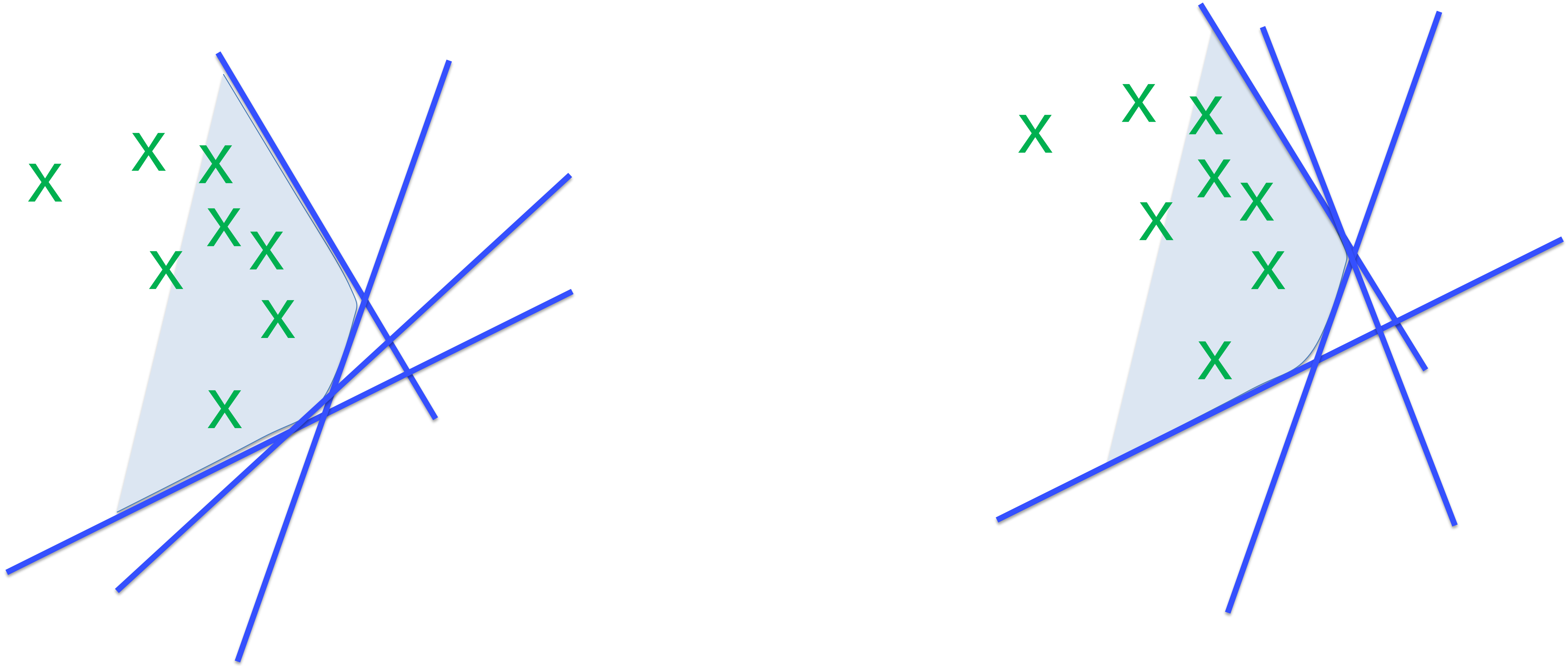


2 blue neurons
2 hyperplanes in input space



4 neurons
4 hyperplanes

1. Many near-equivalent reasonably good solutions



2 near-equivalent good solutions with 4 neurons.

If you have 8 neurons many more possibilities to split the task
→ many near-equivalent good solutions

Quiz: Strengthen your intuitions in high dimensions

A deep neural network with many neurons

[ ] has many minima and a few saddle points
[ ] has many minima and about as many saddle points
[ ] has many minima and even many more saddle points
[ ] gradient descent is slow close to a saddle point
[ ] close to a saddle point there is only one direction to go down
[ ] has typically many equivalent ‘optimal’ solutions
[ ] has typically many near-optimal solutions

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Artificial Neural Networks: Lecture 5

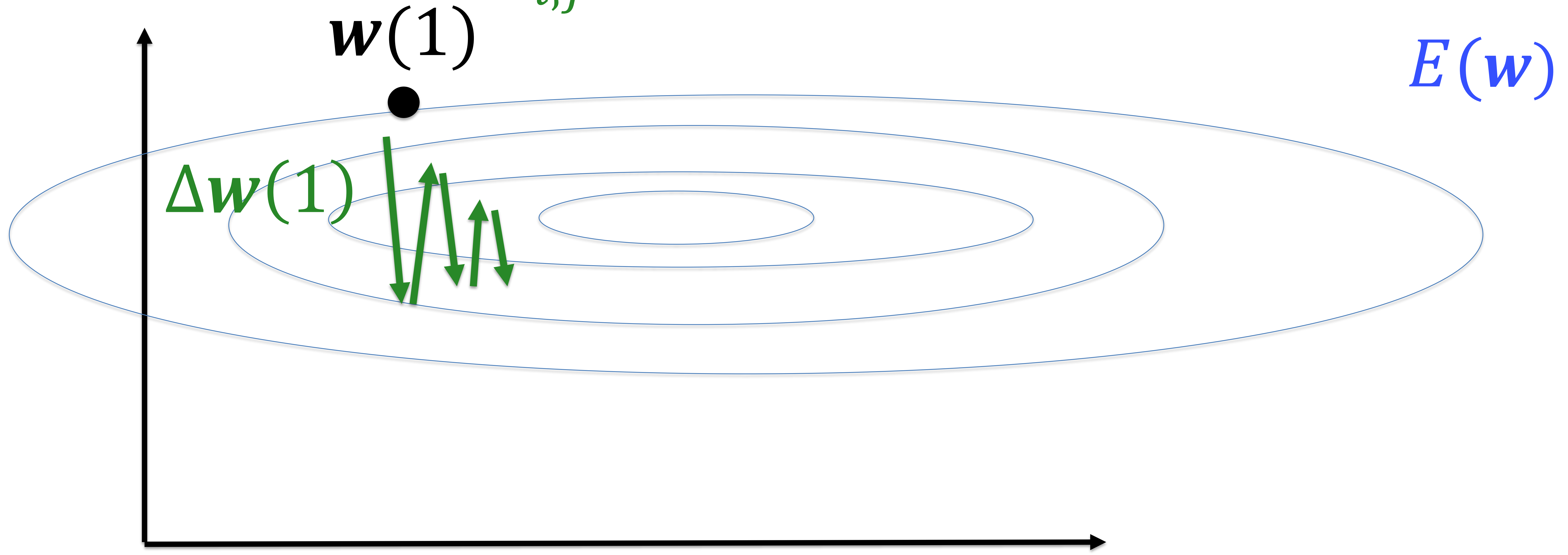
Error function and optimization methods for deep networks

Objectives for today:

- Error function: minima and saddle points
- **Momentum**

Review: Standard gradient descent:

$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\boldsymbol{w}(1))}{dw_{i,j}^{(n)}}$$



2. Momentum: keep previous information

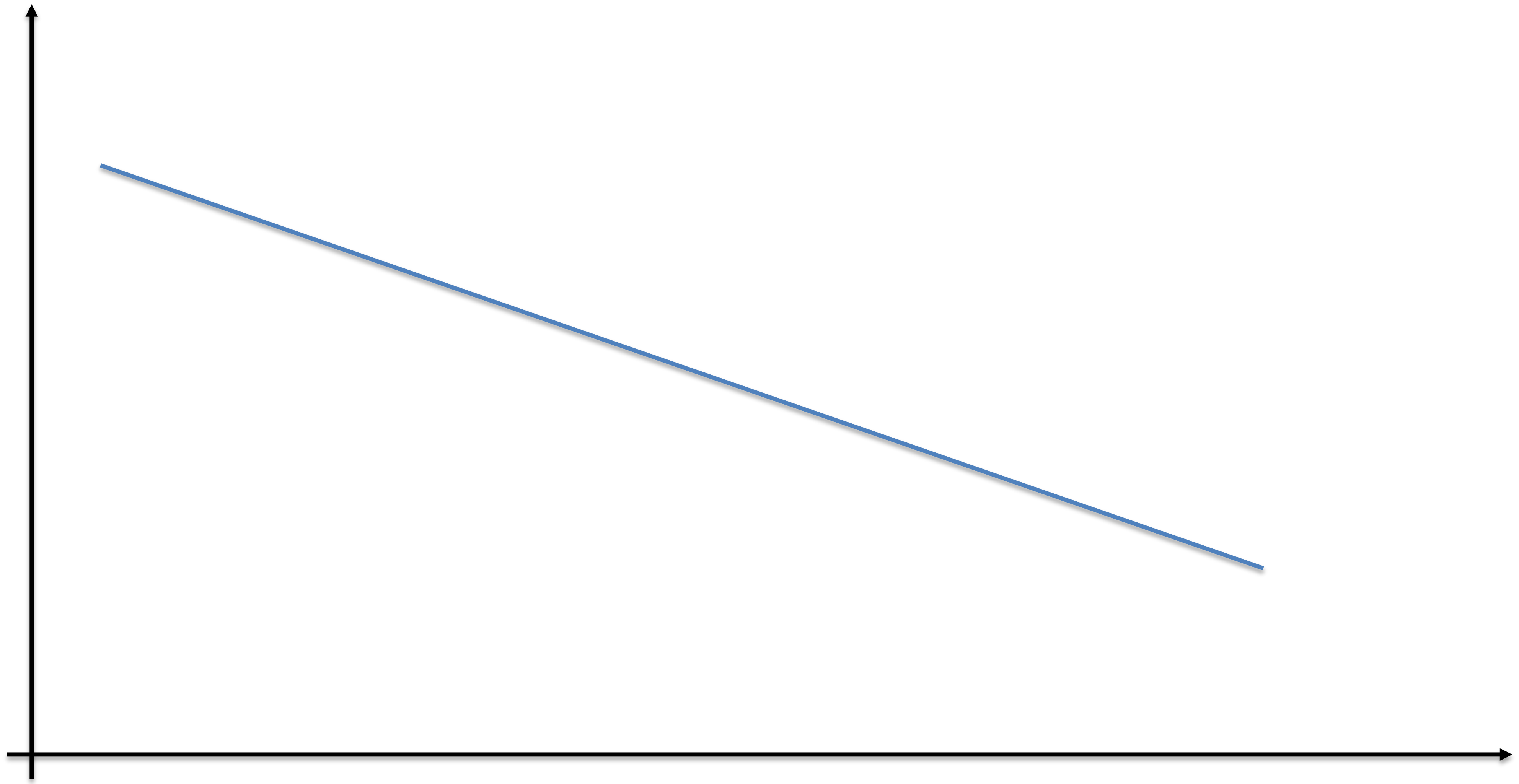
Blackboard2

In first time step: $m=1$

$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\mathbf{w}(1))}{dw_{i,j}^{(n)}}$$

In later time step: m

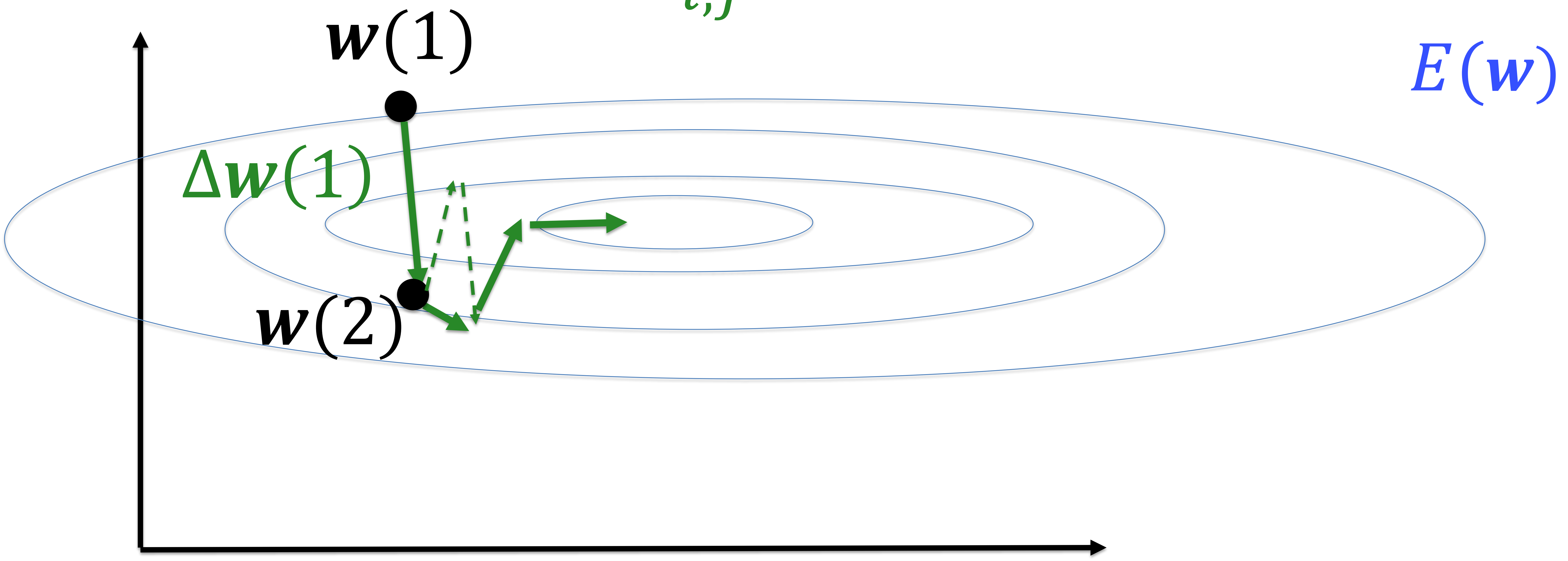
$$\Delta w_{i,j}^{(n)}(m) = -\gamma \frac{dE(\mathbf{w}(m))}{dw_{i,j}^{(n)}} + \alpha \ \Delta w_{i,j}^{(n)}(m-1)$$



Blackboard2

2. Momentum suppresses oscillations

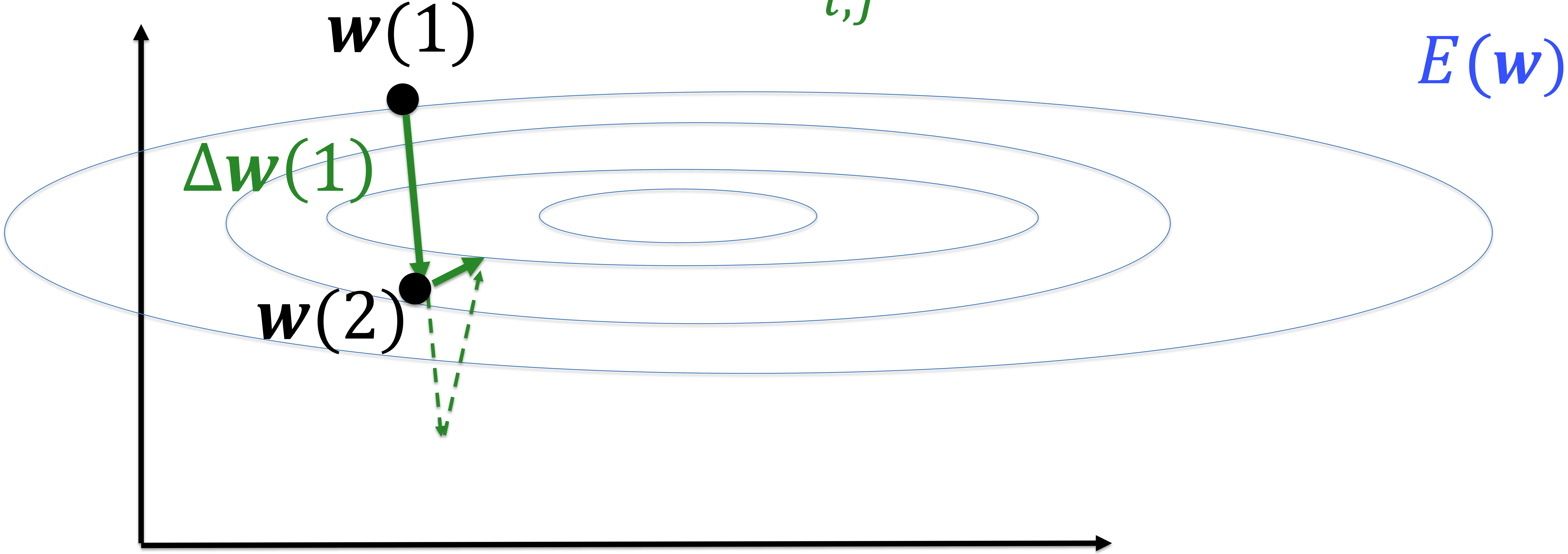
$$\Delta w_{i,j}^{(n)}(2) = -\gamma \frac{dE(\boldsymbol{w}(2))}{dw_{i,j}^{(n)}} + \alpha \; \Delta w_{i,j}^{(n)}(1)$$



good values for α: 0.9 or 0.95 or 0.99 combined with small γ

2. Nesterov Momentum (evaluate gradient at interim location)

$$\Delta w_{i,j}^{(n)}(2) = -\gamma \frac{dE\left(\boldsymbol{w}(\mathbf{2}) + \alpha \Delta w_{i,j}^{(n)}(1)\right)}{dw_{i,j}^{(n)}} + \alpha \ \ \Delta w_{i,j}^{(n)}(1)$$



good values for α: 0.9 or 0.95 or 0.99 combined with small γ

Quiz: Momentum

Momentum

[ ] momentum speeds up gradient descent in ‘boring’ directions

[ ] momentum suppresses oscillations

[ ] with a momentum parameter α=0.9 the maximal speed-up is a factor 1.9

[ ] with a momentum parameter α=0.9 the maximal speed-up is a factor 10

[ ] Nesterov momentum needs twice as many gradient evaluations as standard momentum

Artificial Neural Networks: Lecture 5

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Error function and optimization methods for deep networks

Objectives for today:

- Error function: minima and saddle points
- Momentum
- **RMSprop and ADAM**

3. Error function: batch gradient descent

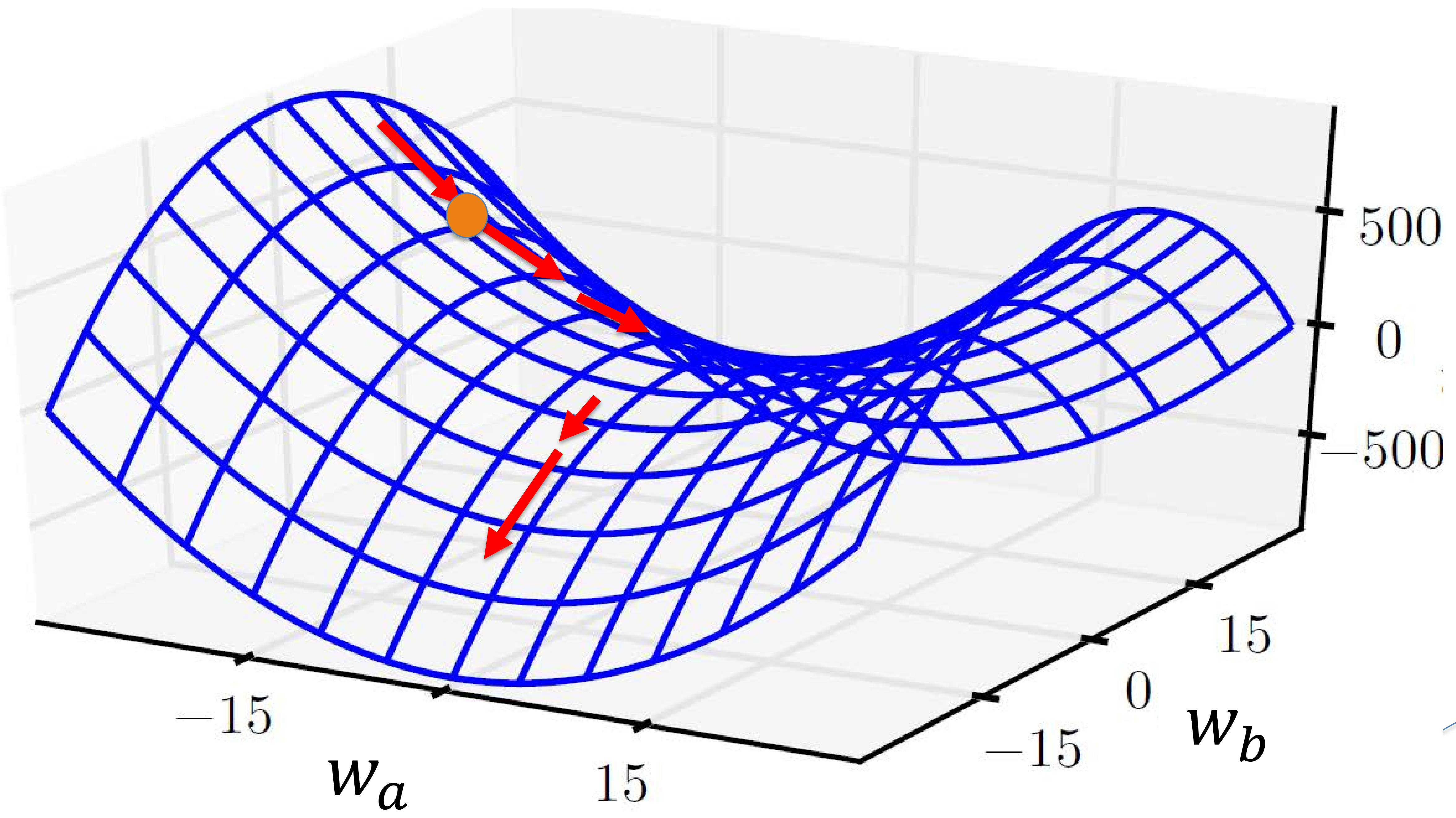
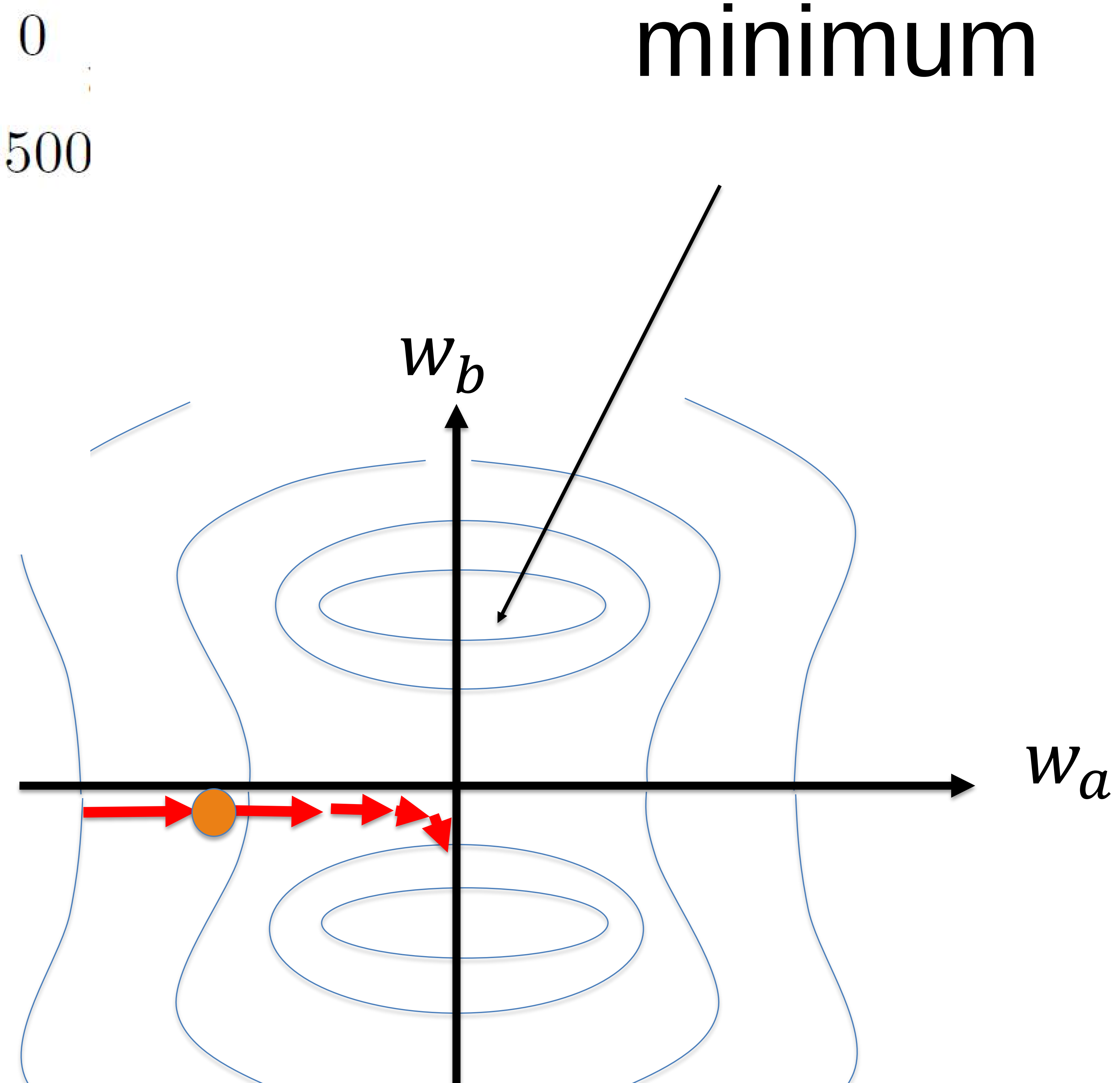


Image: Goodfellow et al. 2016

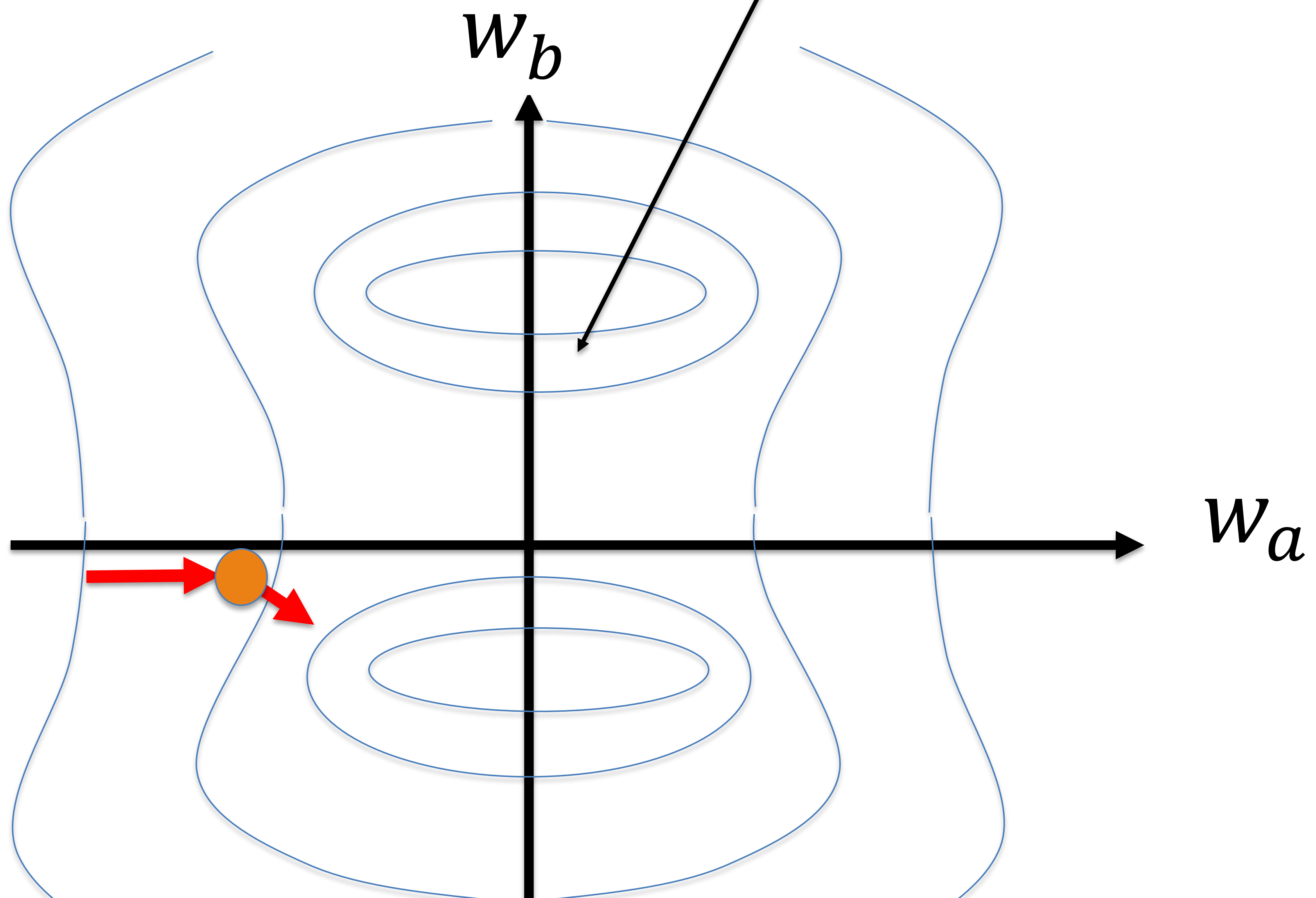
$\boldsymbol{w}(1)$



3. Error function: stochastic gradient descent

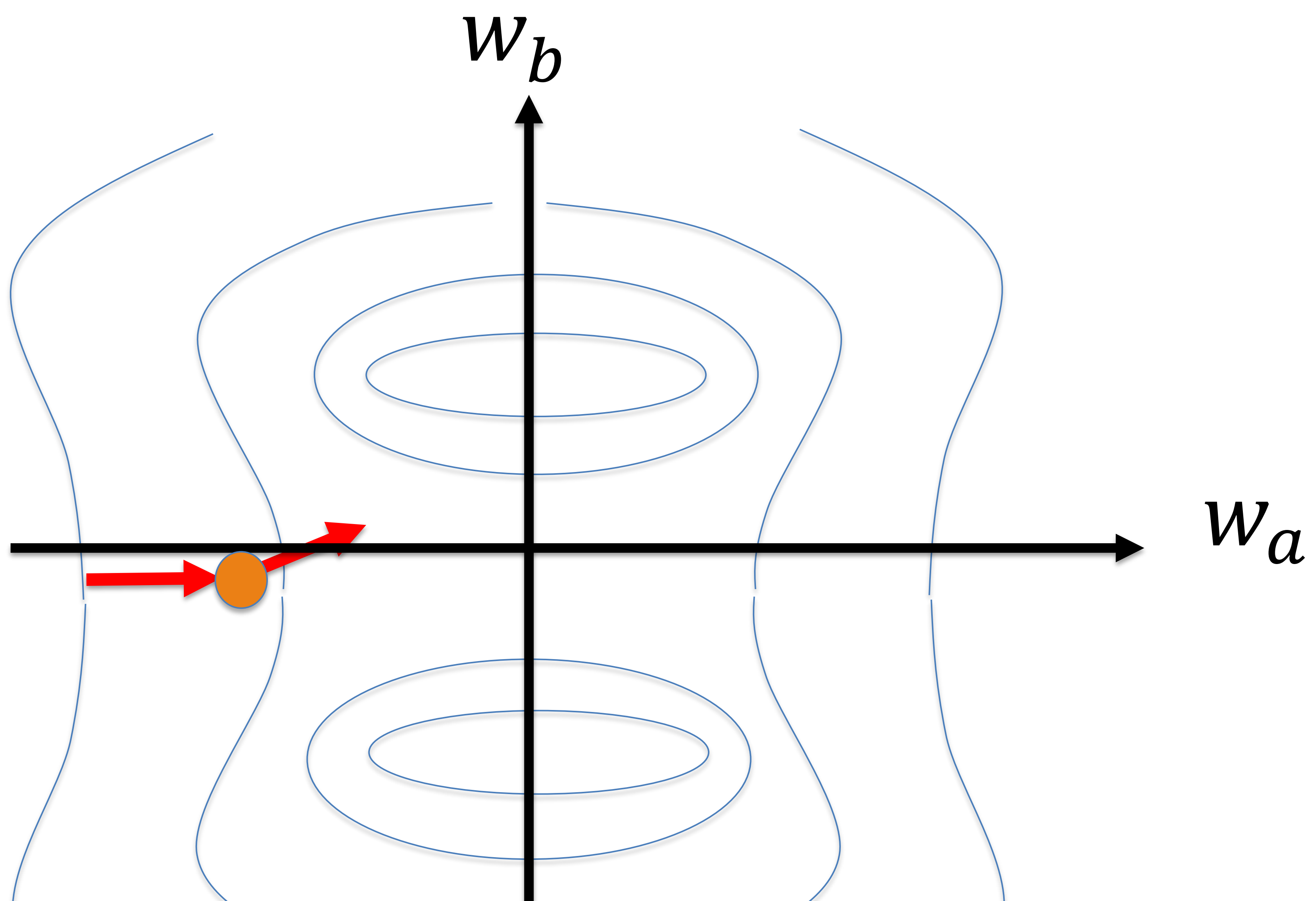
The error function for a small mini-batch
is not identical to the that of the true batch

old
minimum



3. Error function: batch vs. stochastic gradient descent

The error function for a small mini-batch
is not identical to the that of the true batch

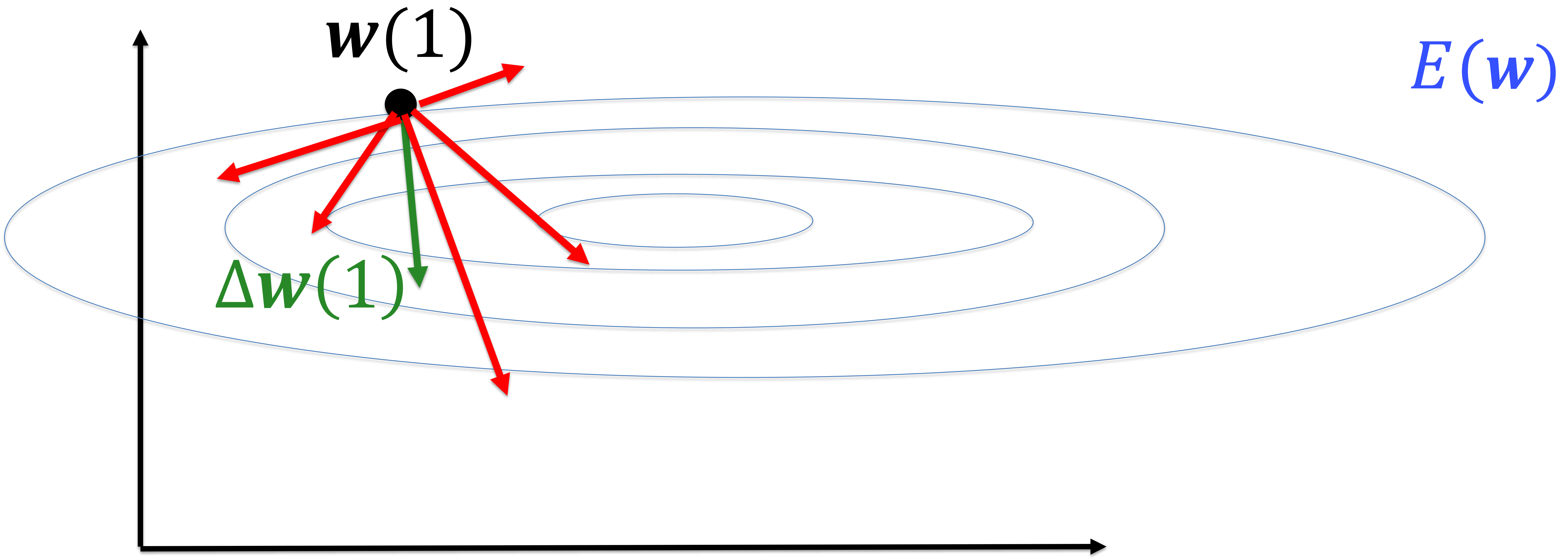


3. Stochastic gradient evaluation

$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\boldsymbol{w}(1))}{dw_{i,j}^{(n)}}$$

real gradient: sum over all samples

stochastic gradient: one sample



Idea: estimate mean and variance from k=1/α samples

Quiz: RMS and ADAM – what do we want?

A good optimization algorithm

[ ] should have different ‘effective learning rate’ for each weight

[ ] should have smaller update steps for noisy gradients

[ ] the weight change should be larger for small gradients and smaller for large ones

[ ] the weight change should be smaller for small gradients and larger for large ones

3. Stochastic gradient evaluation

$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\mathbf{w}(1))}{dw_{i,j}^{(n)}}$$

real gradient: sum over all samples

stochastic gradient: one sample

Idea: estimate mean and variance from k=1/ρ samples

Running Mean: use momentum

$$v_{i,j}^{(n)}(m) = \frac{dE(\mathbf{w}(m))}{dw_{i,j}^{(n)}} + \rho_1 v_{i,j}^{(n)}(m-1)$$

Running second moment: average the squared gradient

$$r_{i,j}^{(n)}(m) = (1-\rho_2)\left(\frac{dE(\mathbf{w}(m))}{dw_{i,j}^{(n)}}\right)\left(\frac{dE(\mathbf{w}(m))}{dw_{i,j}^{(n)}}\right) + \rho_2 r_{i,j}^{(n)}(m-1)$$

3. Stochastic gradient evaluation

Blackboard 3/Exerc. 1

Example:
consider 3 weights w_1, w_2, w_3

Raw Gradient: $\dfrac{dE(\boldsymbol{w}(1))}{dw_{i,j}^{(n)}}$

Running Mean: use momentum

$$v_{i,j}^{(n)}(m) == (1-\rho_1)\frac{dE(\boldsymbol{w}(m))}{dw_{i,j}^{(n)}} + \rho_1 v_{i,j}^{(n)}(m-1)$$

Time series of gradient by sampling:

for w_1 : 1.1; 0.9; 1.1; 0.9; …

for w_2 : 0.1; 0.1; 0.1; 0.1; …

for w_3 : 1.1; 0; -0.9; 0; 1.1; 0; -0.9; .

Running estimate of 2nd moment: average the squared gradient

$$r_{i,j}^{(n)}(m) = (1-\rho_2)\left(\frac{dE(\boldsymbol{w}(m))}{dw_{i,j}^{(n)}}\right)\left(\frac{dE(\boldsymbol{w}(m))}{dw_{i,j}^{(n)}}\right) + \rho_2 r_{i,j}^{(n)}(m-1)$$

3. Adam and variants

The above ideas are at the core of several algos

- RMSprop
- RMSprop with momentum
- ADAM

3. RMSProp

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.

Initialize accumulation variables $\boldsymbol{r} = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. $(\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

end while

Goodfellow et al. 2016

3. RMSProp with Nesterov Momentum

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$ 2nd moment
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

Goodfellow et al. 2016

3. Adam

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
Initialize time step $t = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
$t \leftarrow t + 1$
Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$ (operations applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

Goodfellow et al. 2016

3. Adam and variants

The above ideas are at the core of several algos

- RMSprop
- RMSprop with momentum
- ADAM

Result: parameter movement slower in uncertain directions

(see Exercise 1 above)

Quiz (2nd vote): RMS and ADAM

A good optimization algorithm

[ ] should have different 'effective learning rate' for each weight

[ ] should have a the same weight update step for small gradients and for large ones

[ ] should have smaller update steps for noisy gradients

Objectives for today:

- Momentum:
  - suppresses oscillations (even in batch setting)
  - implicitly yields a learning rate ‘per weight’
  - smooths gradient estimate (in online setting)

- Adam and variants:
  - adapt learning step size to certainty
  - includes momentum

Artificial Neural Networks: Lecture 5

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Error function and optimization methods for deep networks

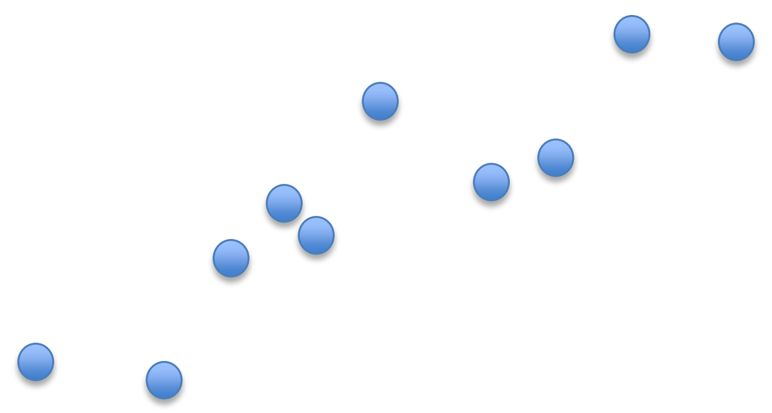
Objectives for today:

- Error function: minima and saddle points
- Momentum
- RMSprop and ADAM
- Complements to Regularization: L1 and L2
- **No Free Lunch Theorem**

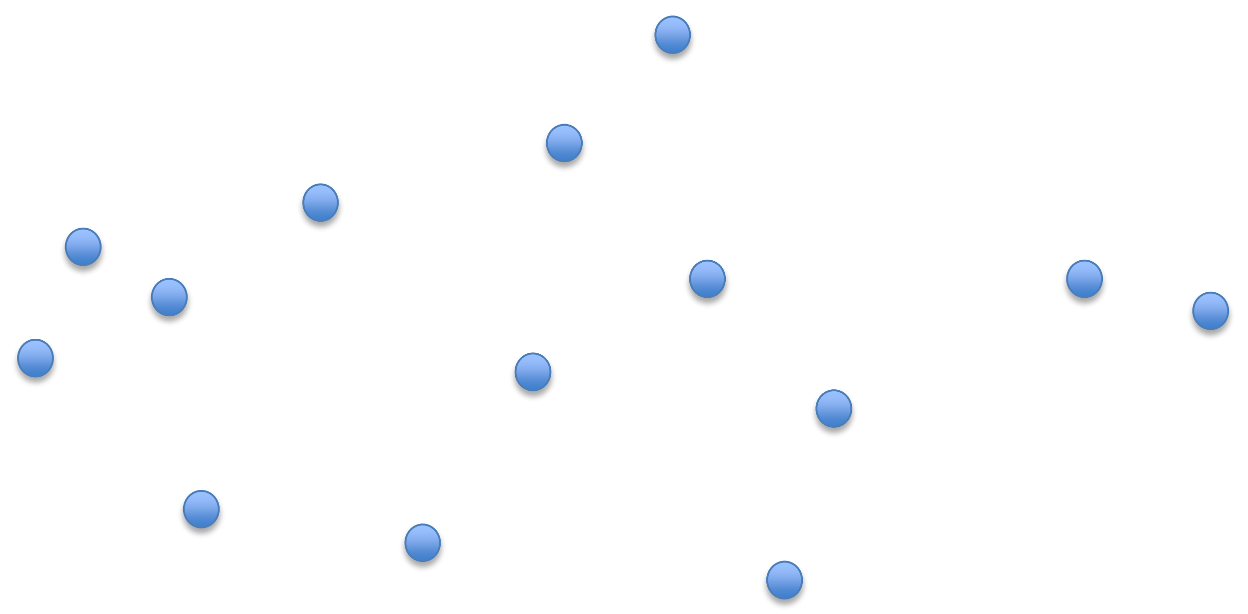
4. No Free Lunch Theorem

Which data set looks more noisy?

A



B



Commitment: Thumbs up

Which data set is easier to fit?

Commitment: Thumbs down

4. No Free Lunch Theorem

5. No Free Lunch Theorem

4. No Free Lunch Theorem

The NO FREE LUNCH THEOREM states
“ *that any two optimization algorithms are equivalent when their performance is averaged across all possible problems"*

See Wikipedia/wiki/No_free_lunch_theorem

•Wolpert, D.H., Macready, W.G. (1997), "No Free Lunch Theorems for Optimization", *IEEE Transactions on Evolutionary Computation* **1**, 67.
•Wolpert, David (1996), "The Lack of *A Priori* Distinctions between Learning Algorithms", *Neural Computation*, pp. 1341-1390.

4. No Free Lunch (NFL) Theorems

The mathematical statements are called

"*NFL theorems because they demonstrate that if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems*"

See Wikipedia/wiki/No_free_lunch_theorem

•Wolpert, D.H., Macready, W.G. (1997), "No Free Lunch Theorems for Optimization", *IEEE Transactions on Evolutionary Computation* **1**, 67.
•Wolpert, David (1996), "The Lack of *A Priori* Distinctions between Learning Algorithms", *Neural Computation*, pp. 1341-1390.

4. Quiz: No Free Lunch (NFL) Theorems

Take neural networks with many layers, optimized by Backprop as an example of deep learning

[ ] Deep learning performs better than most other algorithms on real world problems.

[ ] Deep learning can fit everything.

[ ] Deep learning performs better than other algorithms on all problems.

4. No Free Lunch (NFL) Theorems

- Choosing a deep network and optimizing it with gradient descent is an algorithm

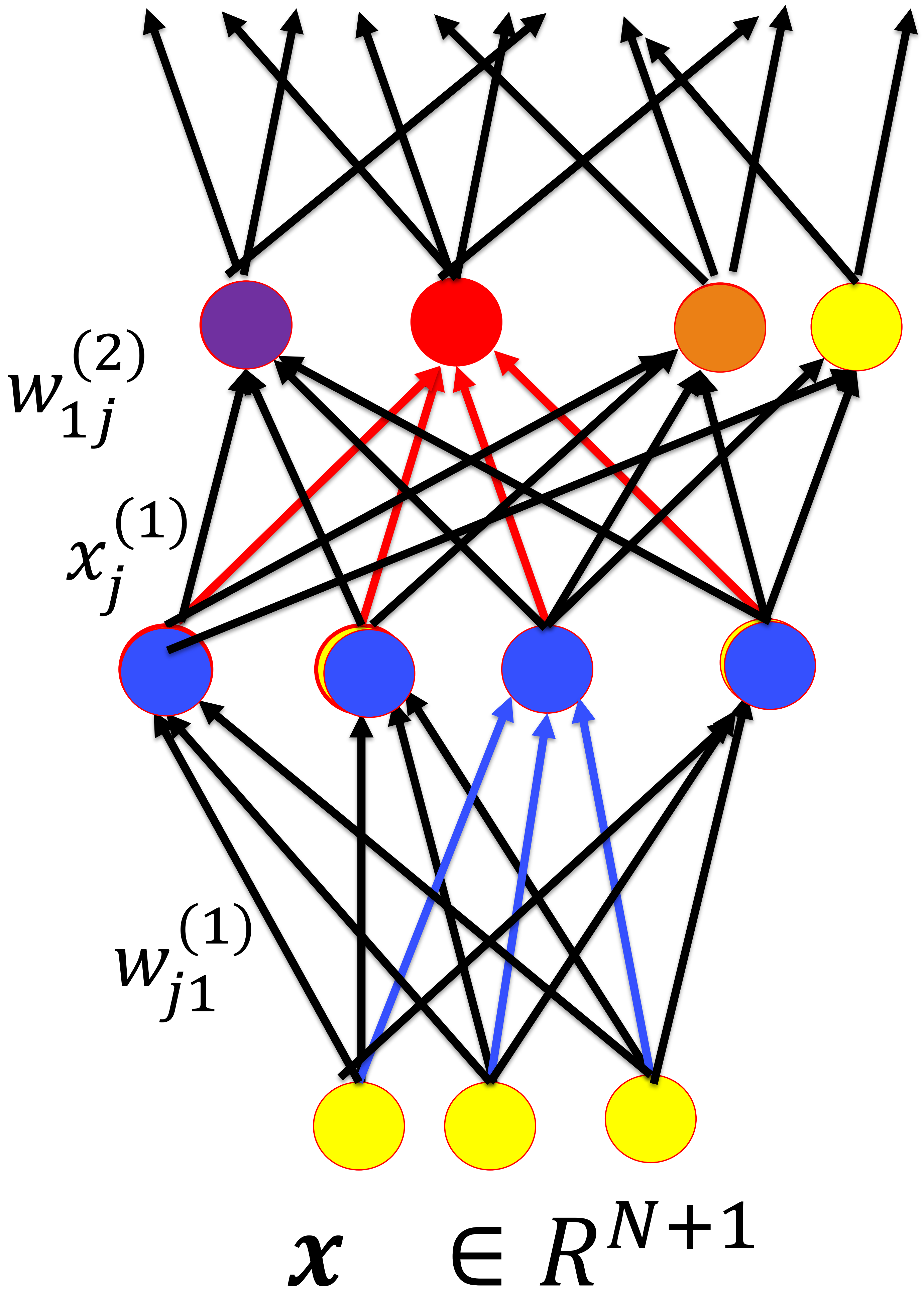
- Deep learning works well on many real-world problems

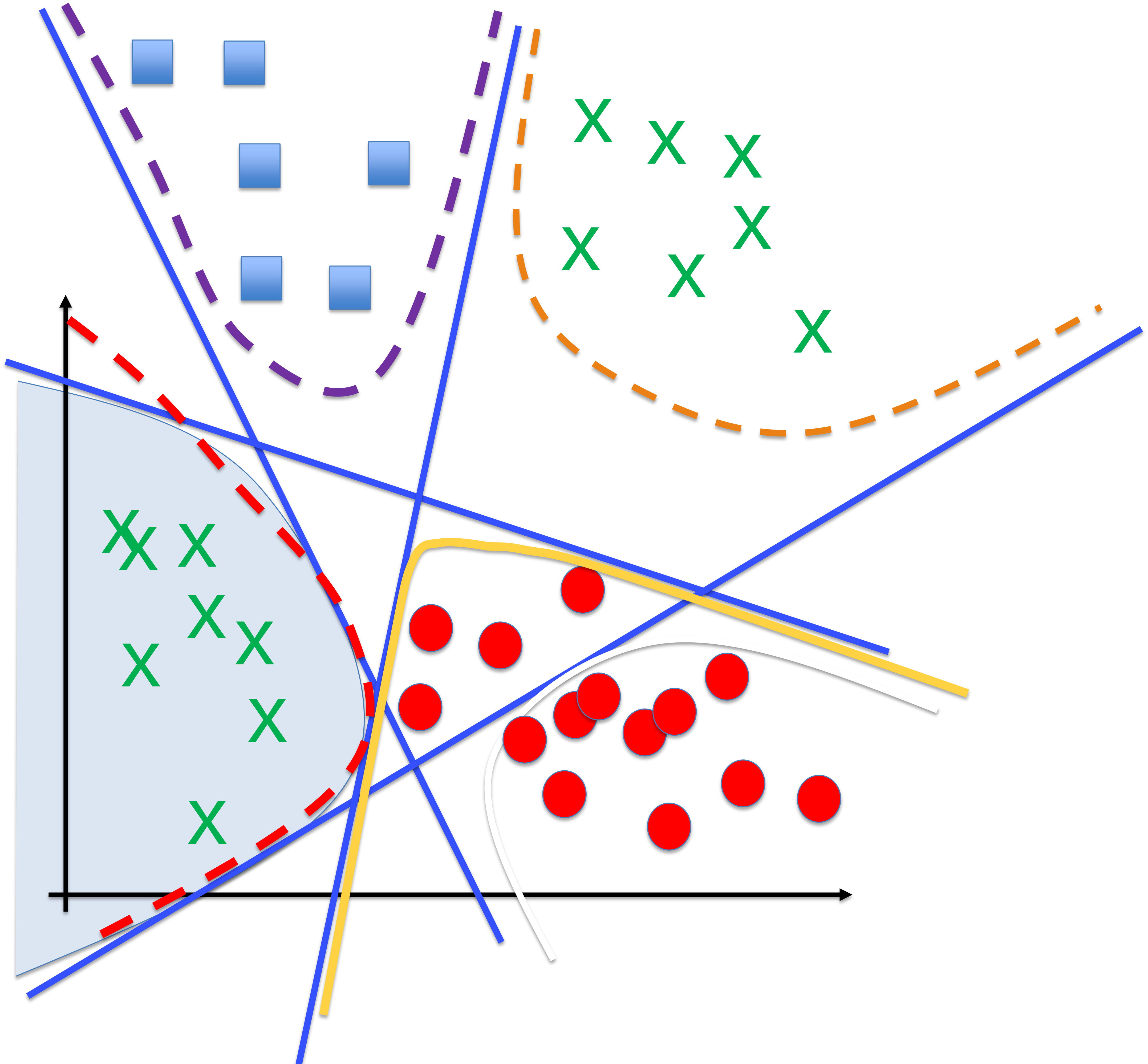
- Somehow the prior structure of the deep network matches the structure of the real-world problems we are interested in.

→ Always use prior knowledge if you have some

4. No Free Lunch (NFL) Theorems

Geometry of the information flow in neural network





4. Reuse of featuers in Deep Networks (schematic)

animals

birds

4 legs

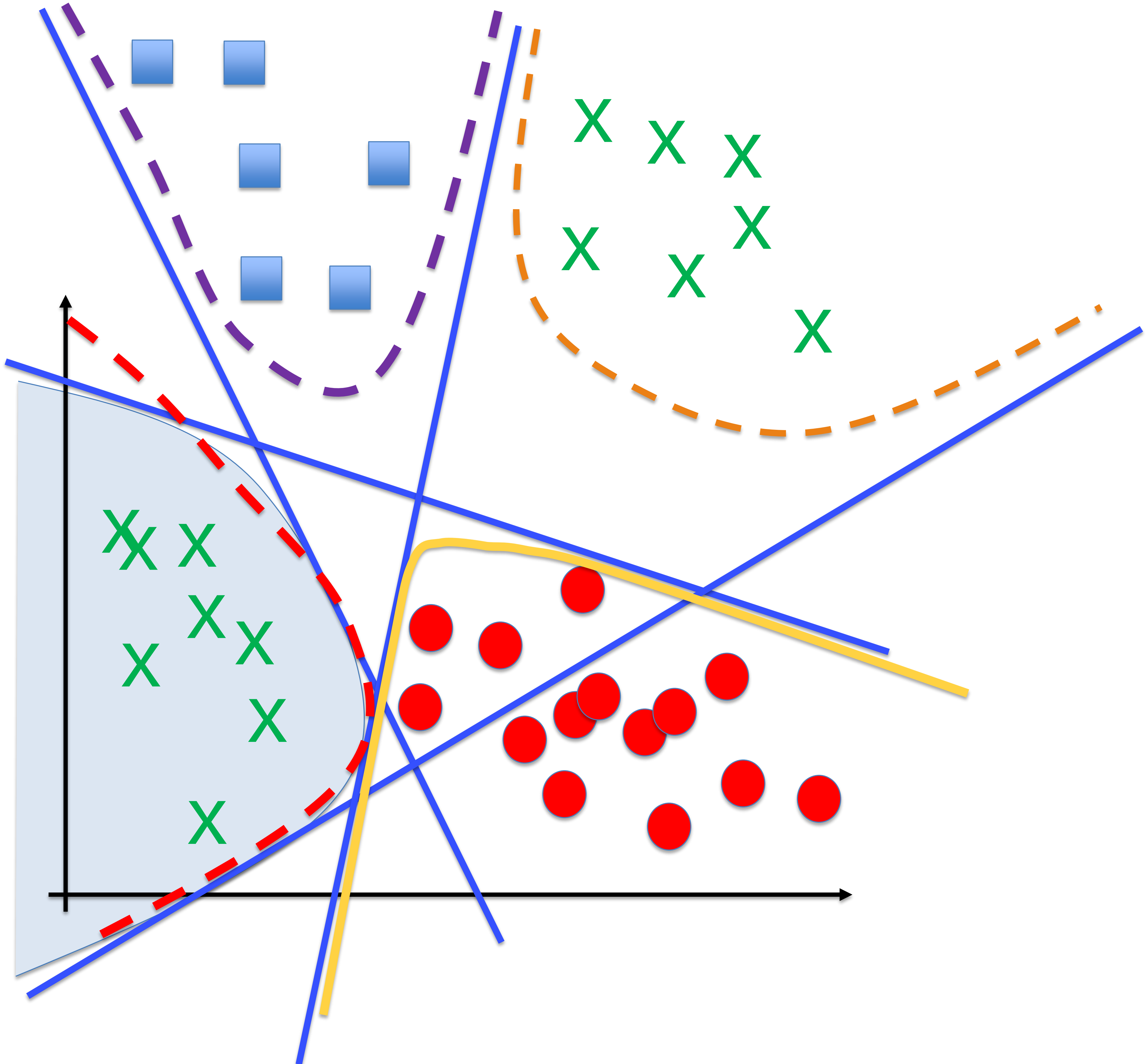
wings

snout

fur

eyes

tail



Artificial Neural Networks: Lecture 5

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Error function and optimization methods for deep networks

Objectives for today:

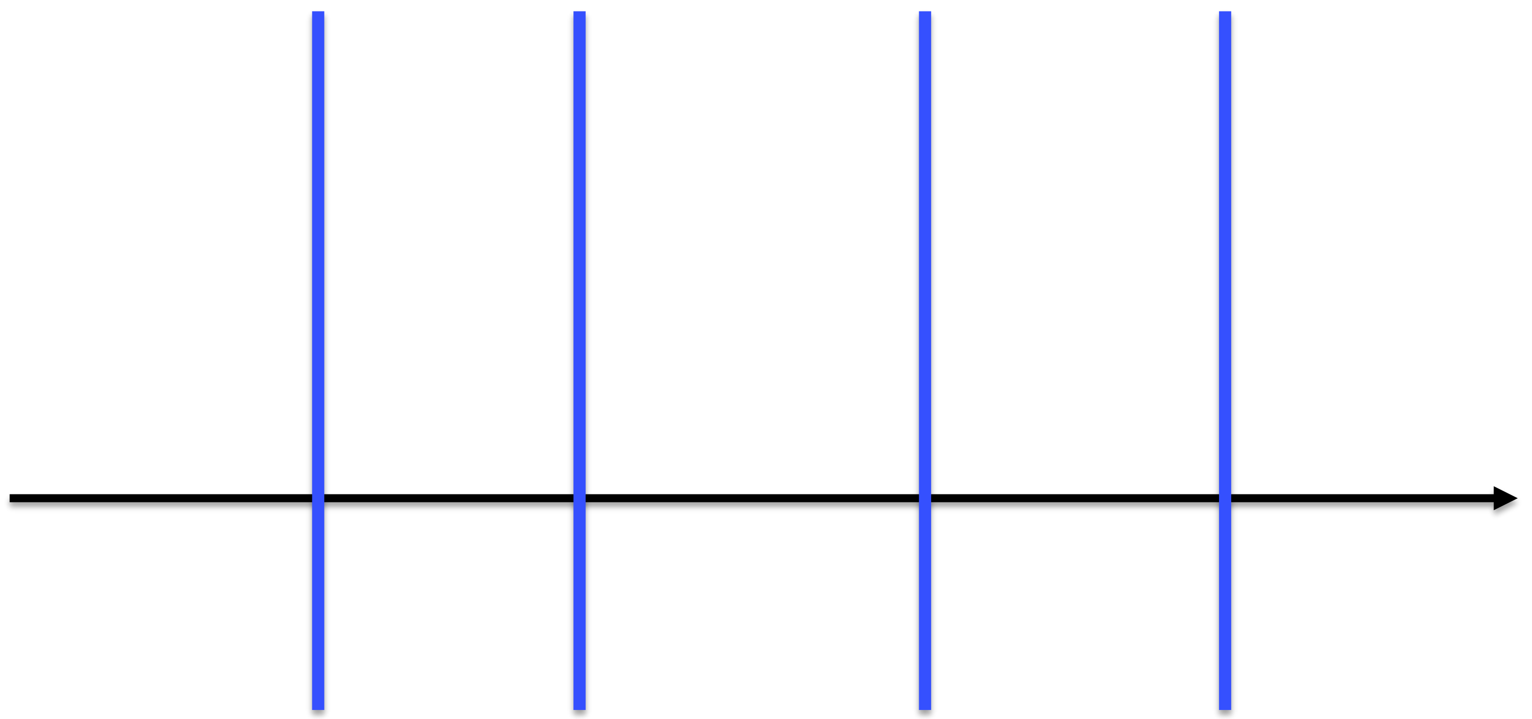
- Error function: minima and saddle points
- Momentum
- RMSprop and ADAM
- Complements to Regularization: L1 and L2
- No Free Lunch Theorem
- **Deep distributed nets versus shallow nets**

5. Distributed representation

How many different regions are carved

In 1dim input space with:

0 hyperplanes
1 hyperplane
2 hyperplanes?
3 hyperplanes?
4 hyperplanes?



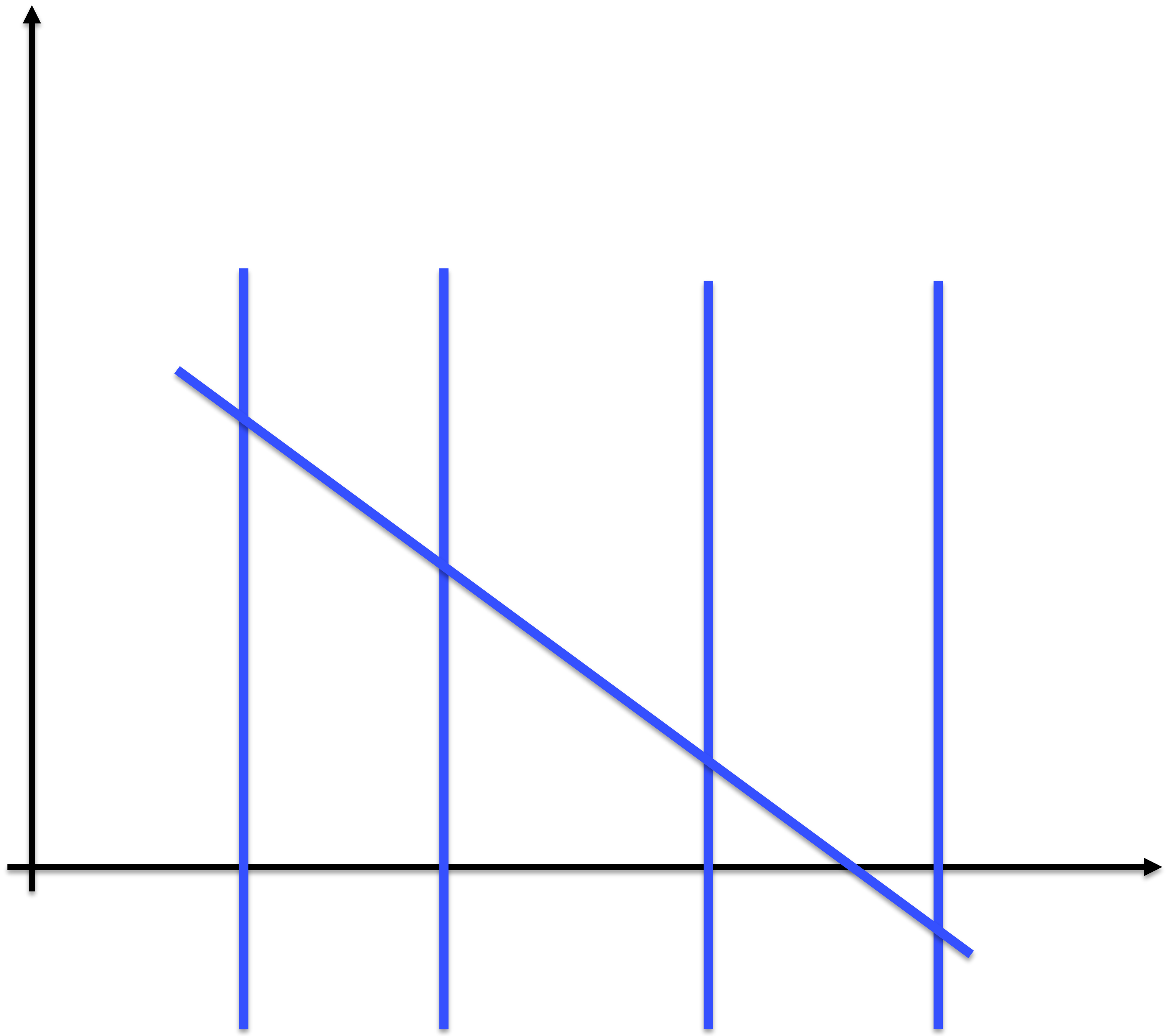
5. Distributed representation

How many different regions are carved

In 2dim input space with:

3 hyperplanes?
4 hyperplanes?

Increase dimension
= turn hyperplane
= new crossing
= new regions



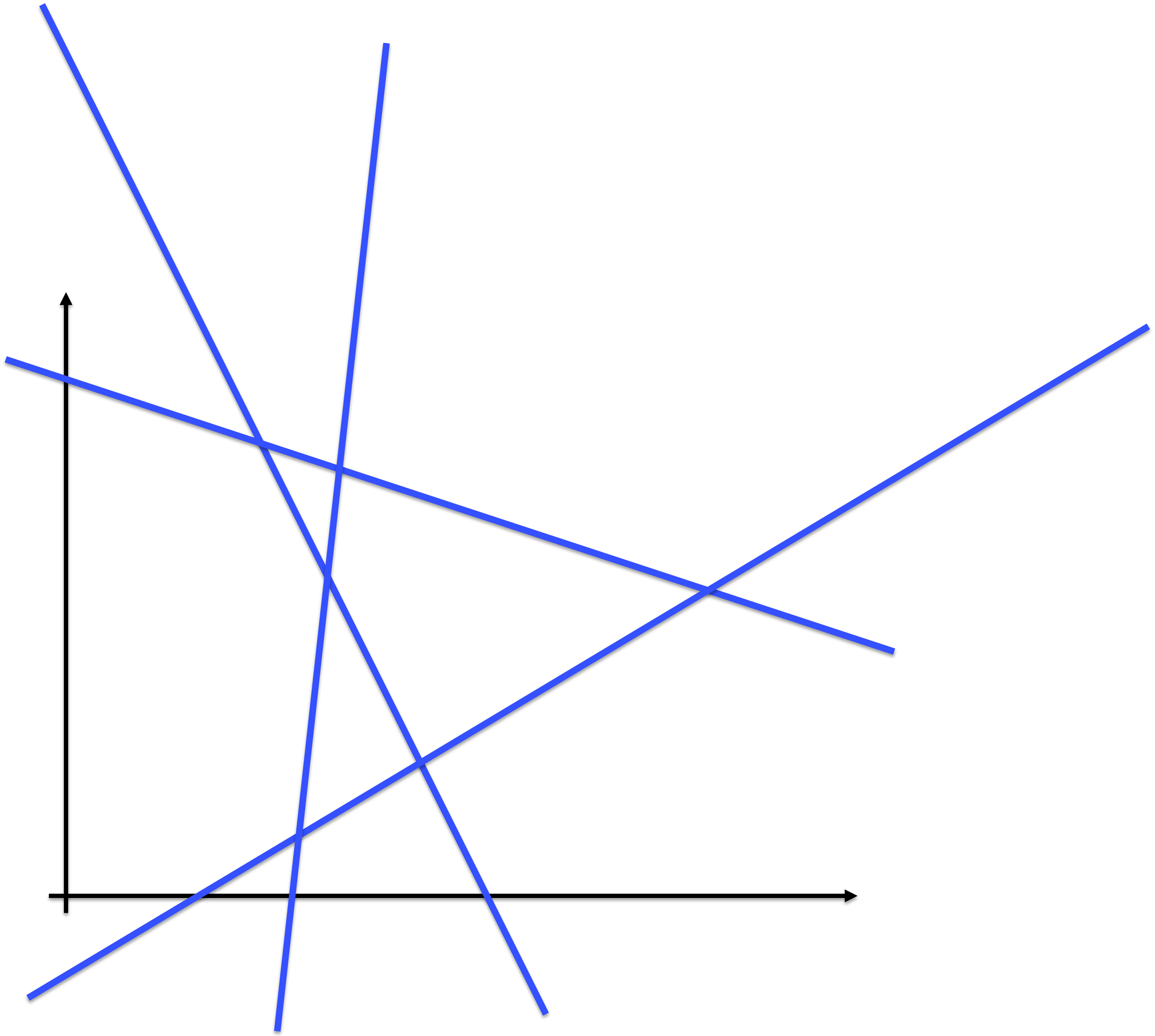
5. Distributed multi-region representation

How many different regions are carved

In 2dim input space by:

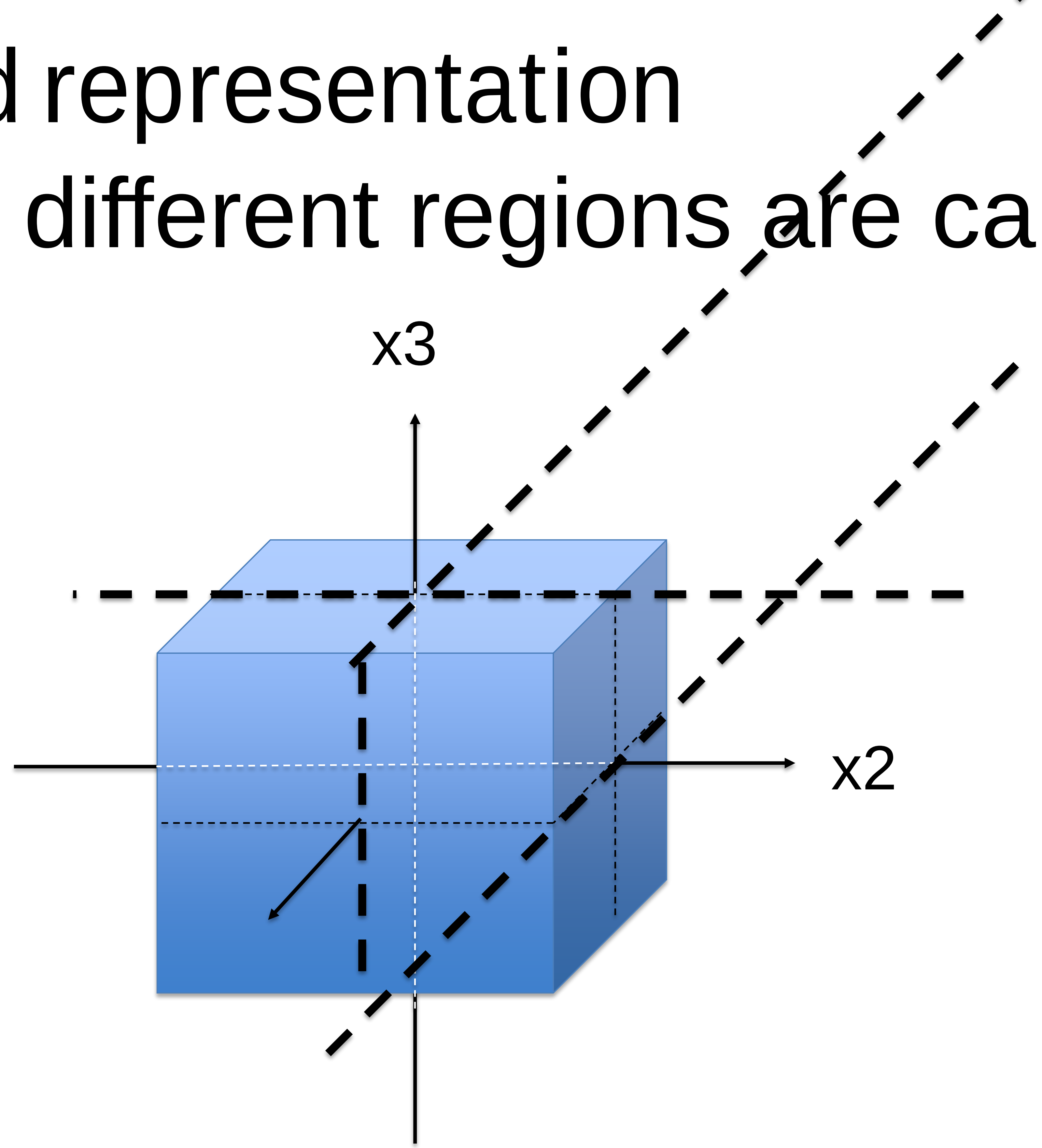
1 hyperplane
2 hyperplanes

3 hyperplanes?
4 hyperplanes?



5. Distributed representation

How many different regions are carved



In 3d input space by:

1 hyperplane
2 hyperplanes

3 hyperplanes?

4 hyperplanes?

5. Distributed multi-region representation

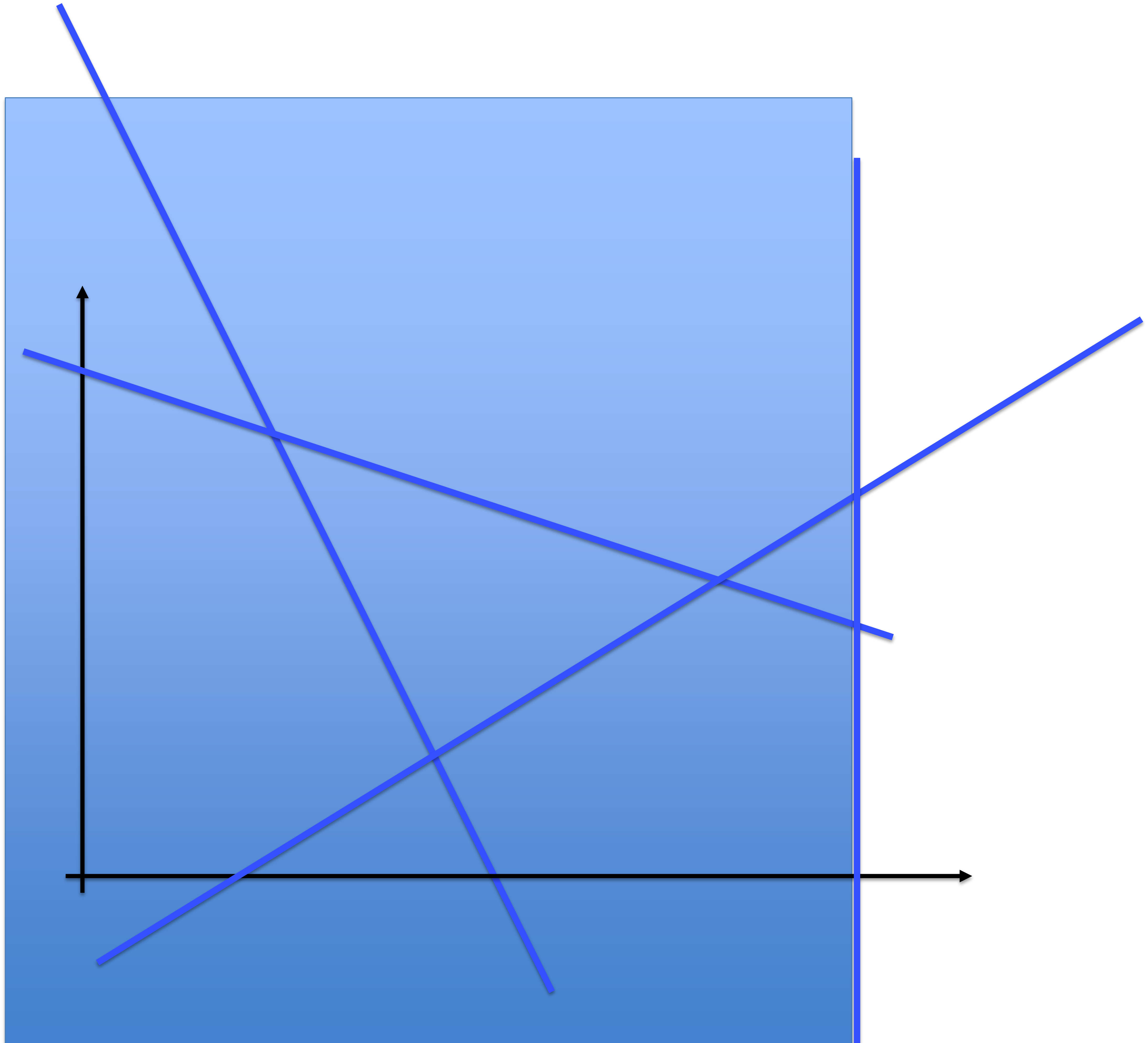
How many different regions are carved

In 3 dim input space by:

3 hyperplanes?
4 hyperplanes?

we look at 4 vertical planes
from the top (birds-eye view)

Keep 3 fixed, but
then tilt 4th plane



5. Distributed multi-region representation

Number of regions cut out by n hyperplanes
In d –dimensional input space:

$$number = \sum_{j=0}^{d} \binom{n}{j}$$

$$number \sim O(n^d)$$

But, we cannot learn arbitrary targets,
by assigning arbitrary class labels {+1,0} to each region,
unless exponentially many hidden neurons:
generalized XOR problem

5. Distributed multi-region representation

There are many, many regions!

But there is a strong prior that we do not need
(for real-world problems) arbitrary labeling of these regions.

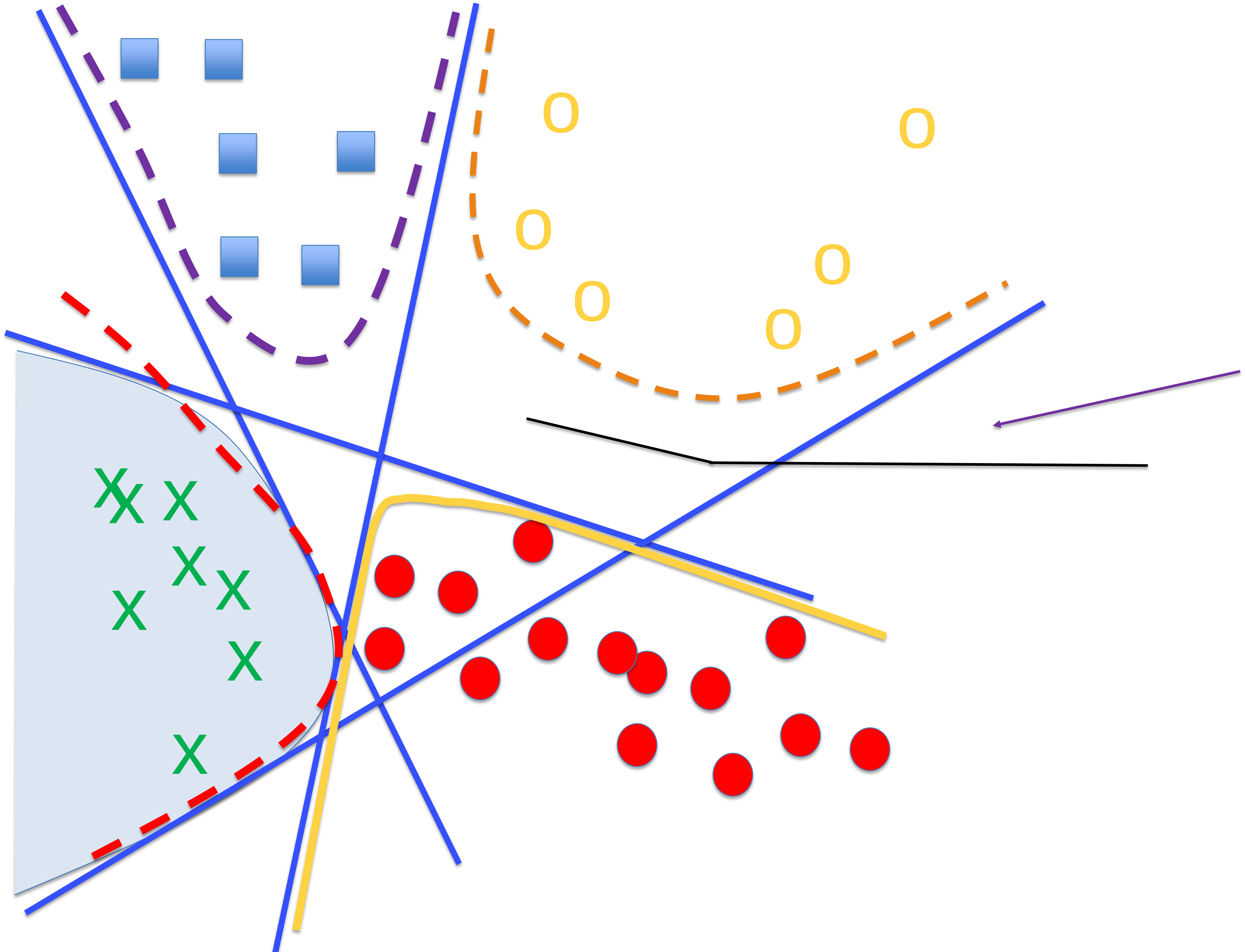
With polynomial number of hidden neurons:

→ classes are automatically assigned for many regions
    where we have no labeled data

→ generalization

5. Distributed representation vs local representation

Example: nearest neighbor representation



Nearest neighbor
Does not create
A new region here

5. Deep networks versus shallow networks

Performance as a function of number of layers on an address classification task

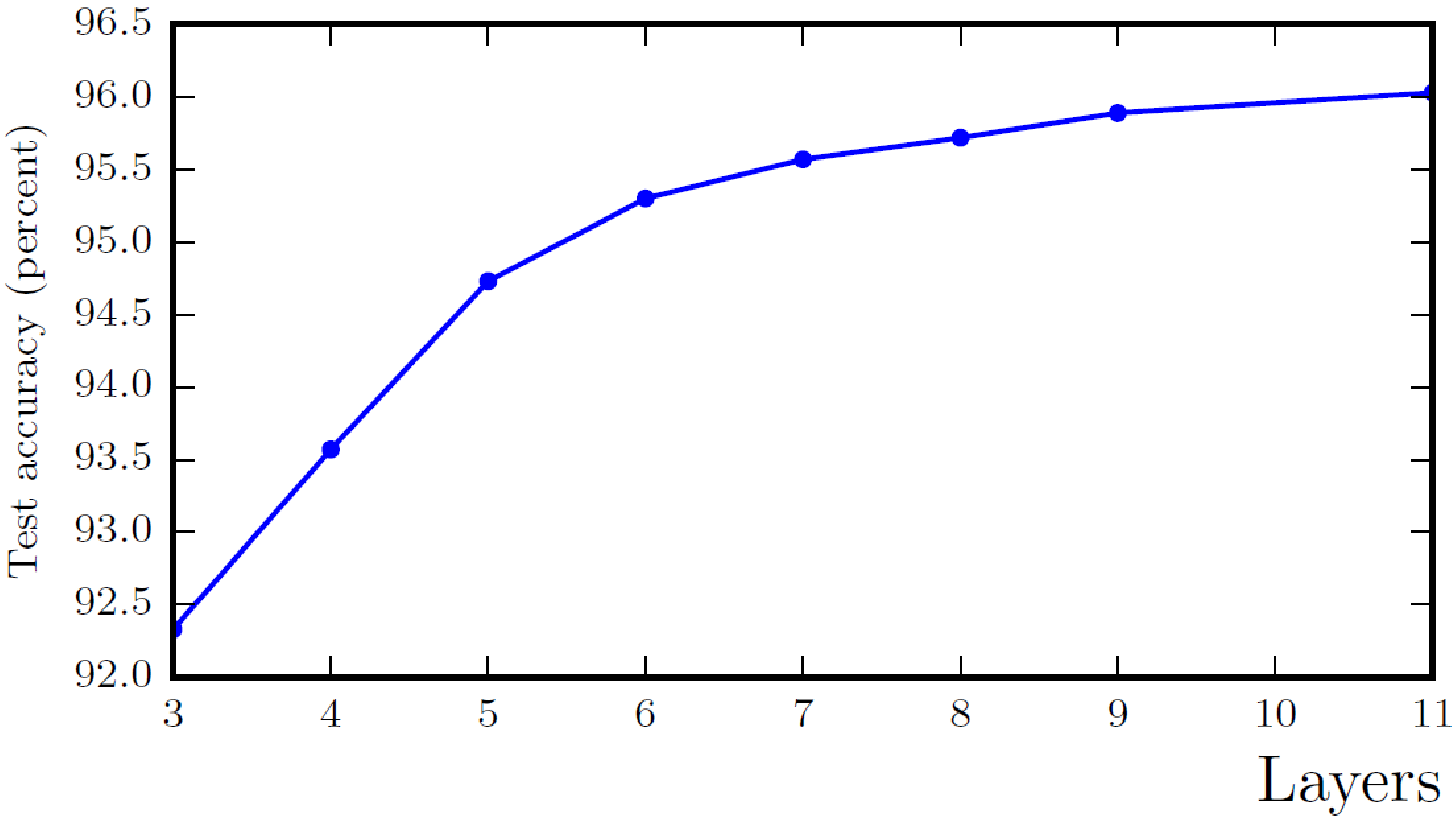


Image: Goodfellow et al. 2016

5. Deep networks versus shallow networks

Performance as a function of number of parameters on an address classification task

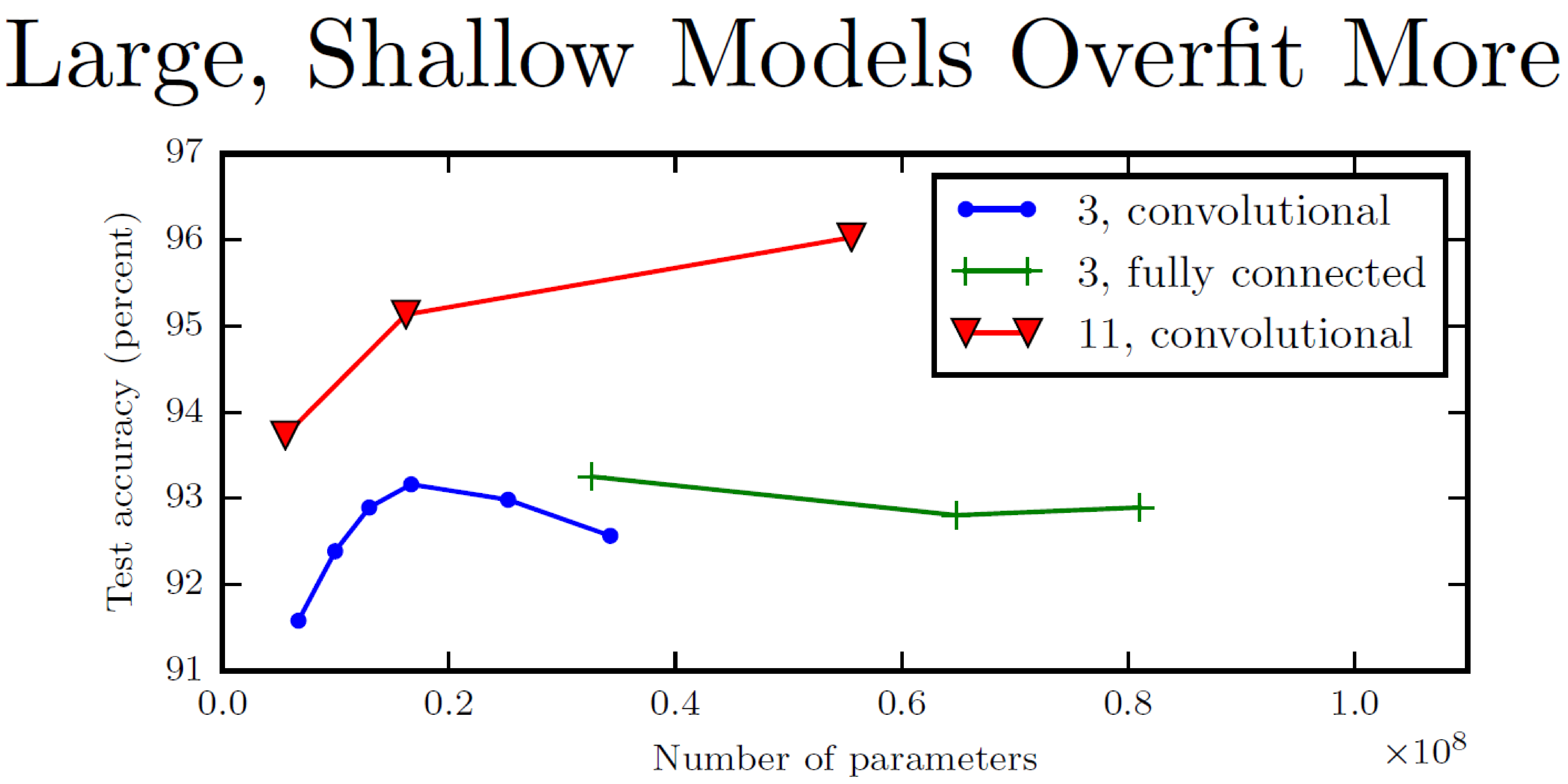


Image: Goodfellow et al. 2016

5. Deep networks versus shallow networks

- Somehow the prior structure of the deep network matches the structure of the real-world problems we are interested in.

- The network reuses features learned in other contexts

Example: green car, red car, green bus, red bus,
tires, window, lights, house,
→ generalize to red house with lights

Artificial Neural Networks: Lecture 5

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Error landscape and optimization methods for deep networks

Objectives for today:

- Error function landscape:

  there are many good minima and even more saddle points
- Momentum

  gives a faster effective learning rate in boring directions
- Adam

  gives a faster effective learning rate in low-noise directions
- No Free Lunch: no algo is better than others
- Deep Networks: are better than shallow ones on real-world problems due to feature sharing

Previous slide.

THE END

Artificial Neural Networks: Lecture 5

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Error function and optimization methods for deep networks

Objectives of this Appendix:

- **Complements to Regularization: L1 and L2**

L2 acts like a spring.
L1 pushes some weights exactly to zero.
L2 is related to early stopping (for quadratic error surface)

Review: Regularization by a penalty term

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

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) \qquad + \lambda \text{ penalty}$$

Loss function

assigns an 'error'
to flexible solutions

Gradient descent at location $\boldsymbol{w}(1)$ yields

$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE\big(\boldsymbol{w}(1)\big)}{dw_{i,j}^{(n)}} \quad - \gamma\, \lambda \frac{d(penalty)}{dw_{i,j}^{(n)}}$$

4. Regularization by a weight decay (L2 regularization)

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

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) \quad + \lambda \sum_k (w_k)^2$$

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

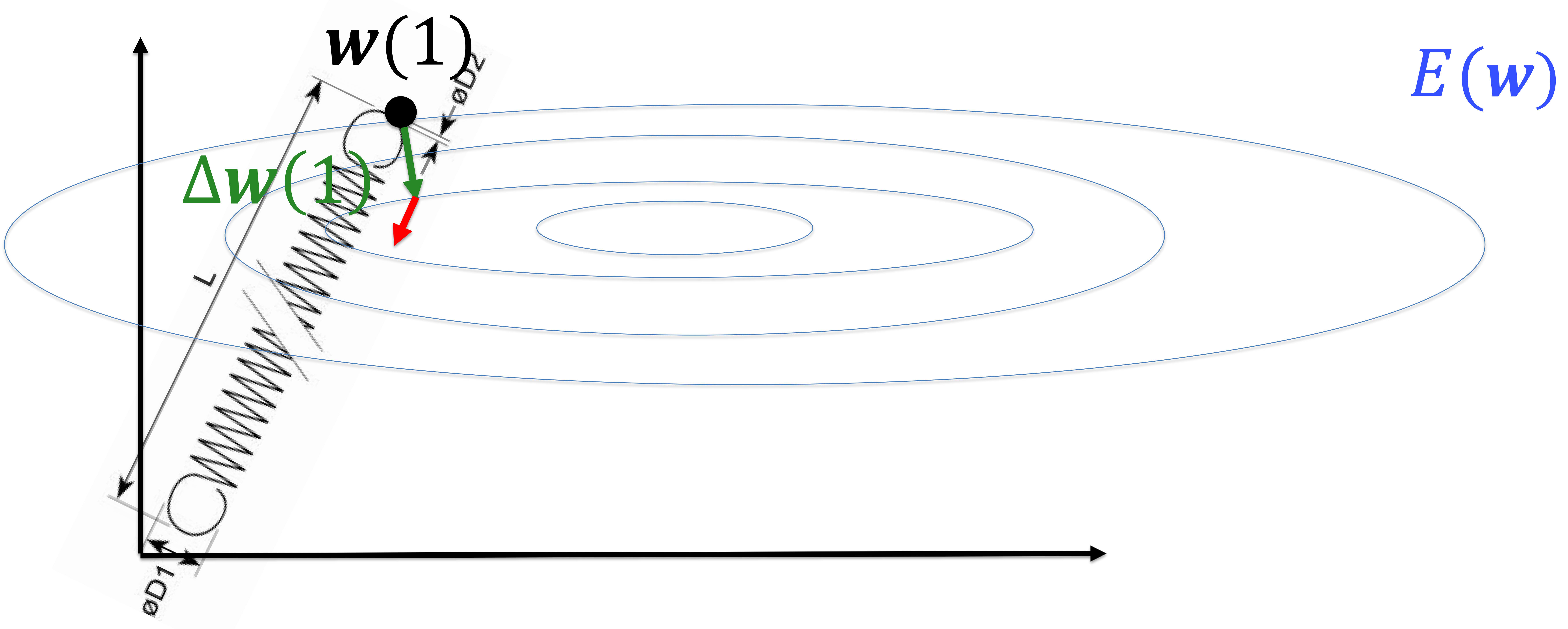
Gradient descent yields

$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\boldsymbol{w}(1))}{dw_{i,j}^{(n)}} \quad -\gamma\, \lambda w_{i,j}^{(n)}(1)$$

See also ML class of Jaggi-Urbanke

4. L2 Regularization

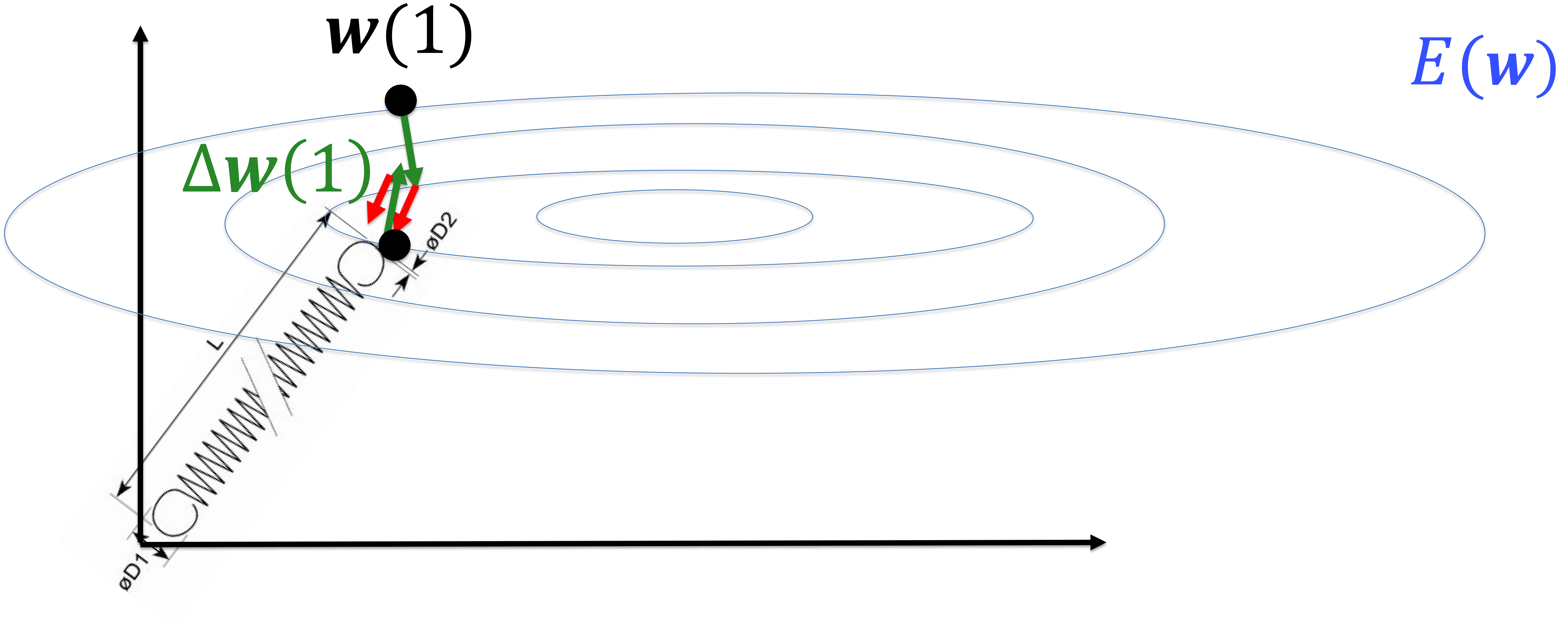
$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\boldsymbol{w}(1))}{dw_{i,j}^{(n)}} - \gamma\, \lambda\, w_{i,j}^{(n)}(1)$$



L2 penalty acts like a spring pulling toward origin

4. L2 Regularization

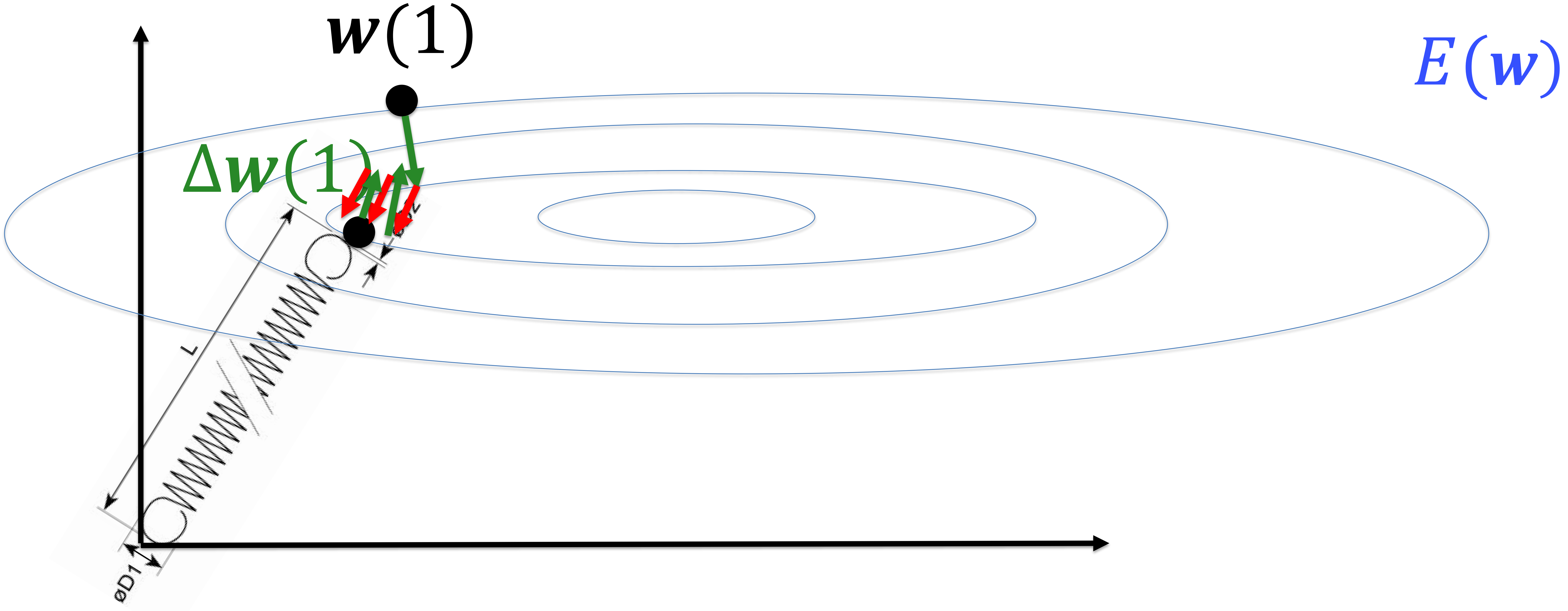
$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\boldsymbol{w}(1))}{dw_{i,j}^{(n)}} - \gamma \, \lambda \, w_{i,j}^{(n)}(1)$$



L2 penalty acts like a spring pulling toward origin

4. L2 Regularization

$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\boldsymbol{w}(1))}{dw_{i,j}^{(n)}} - \gamma\, \lambda\, w_{i,j}^{(n)}(1)$$



L2 penalty acts like a spring pulling toward origin

4. L1 Regularization

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

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) \qquad + \lambda \sum_k |w_k \;|$$

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

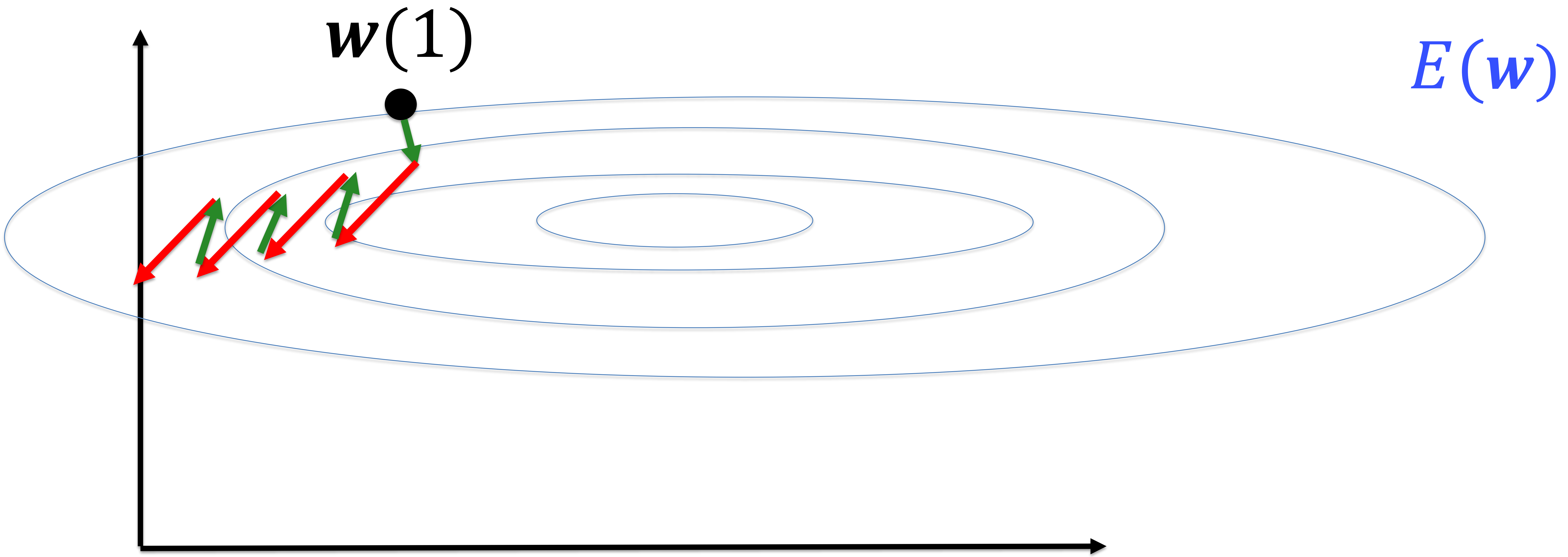
$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\boldsymbol{w}(1))}{dw_{i,j}^{(n)}} \quad -\gamma\,\lambda\, sgn(w_{i,j}^{(n)})$$

See also ML class of Jaggi-Urbanke

4. L1 Regularization

Blackboard4

$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\boldsymbol{w}(1))}{dw_{i,j}^{(n)}} - \gamma \quad \lambda\, sgn[w_{i,j}^{(n)}(1)]$$

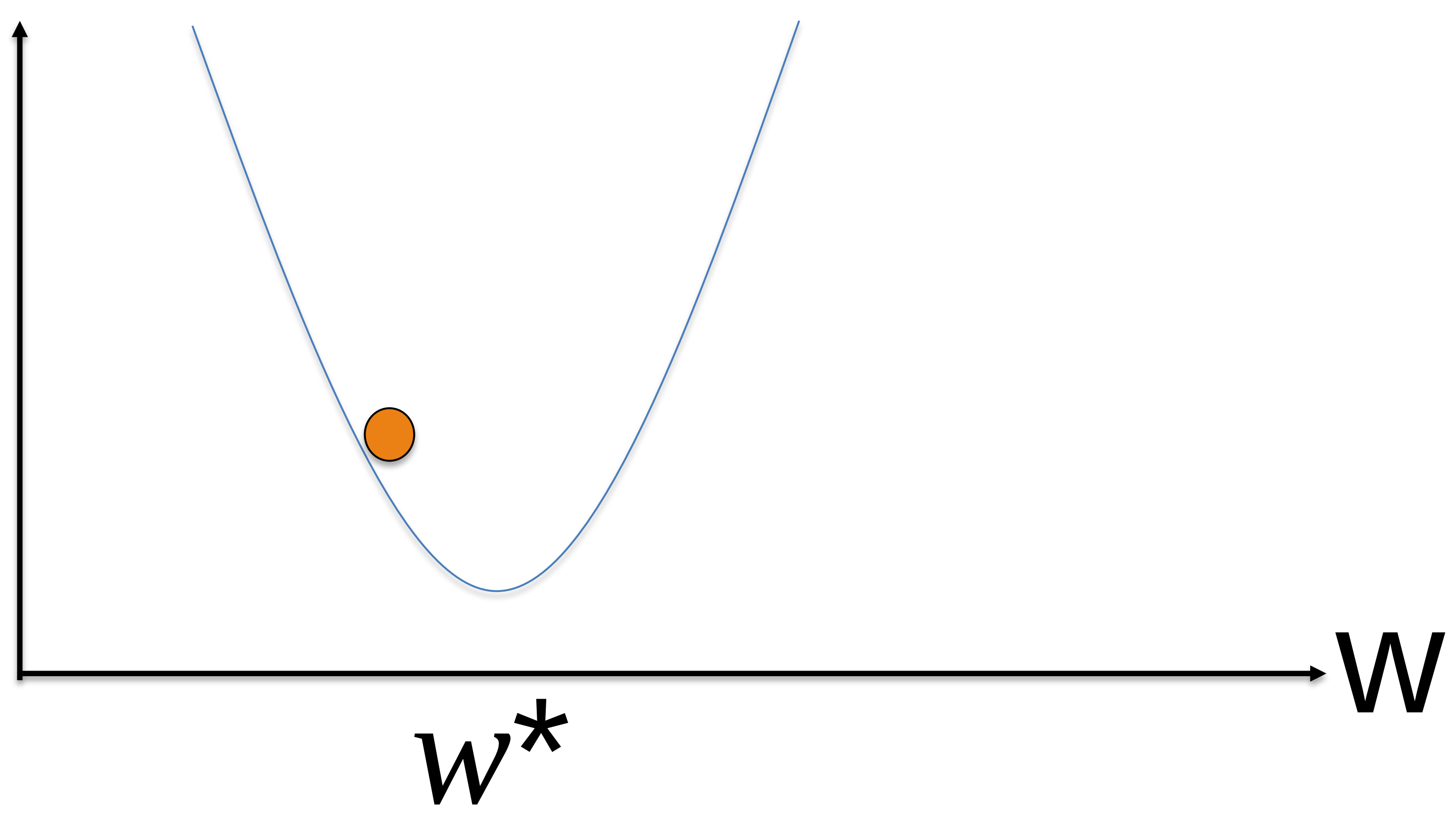


Movement caused by penalty is always diagonal
(except if one compenent vanishes: $w_a=0$

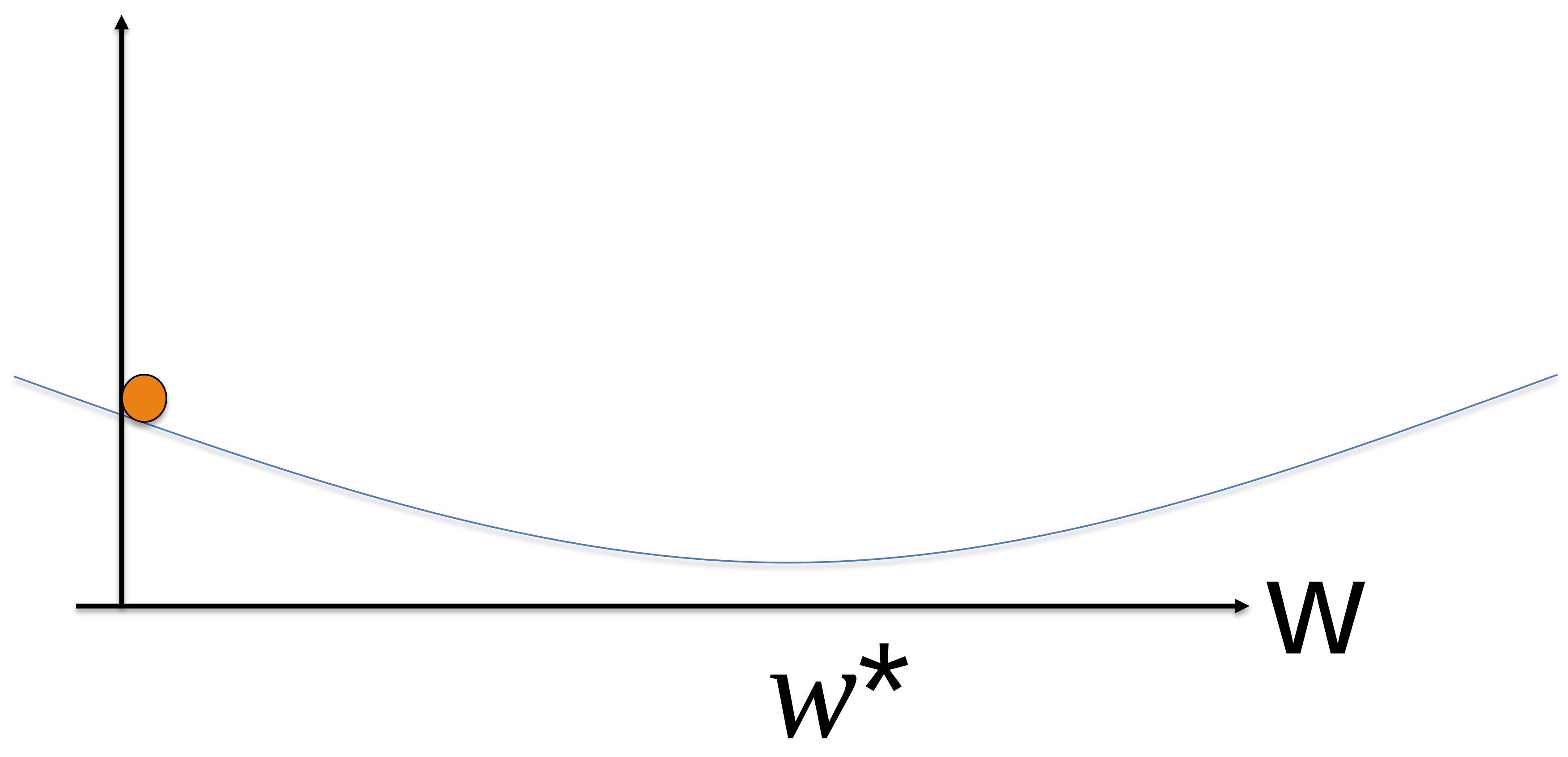
4. L1 Regularization

Blackboard4

4. L1 Regularization (quadratic function)



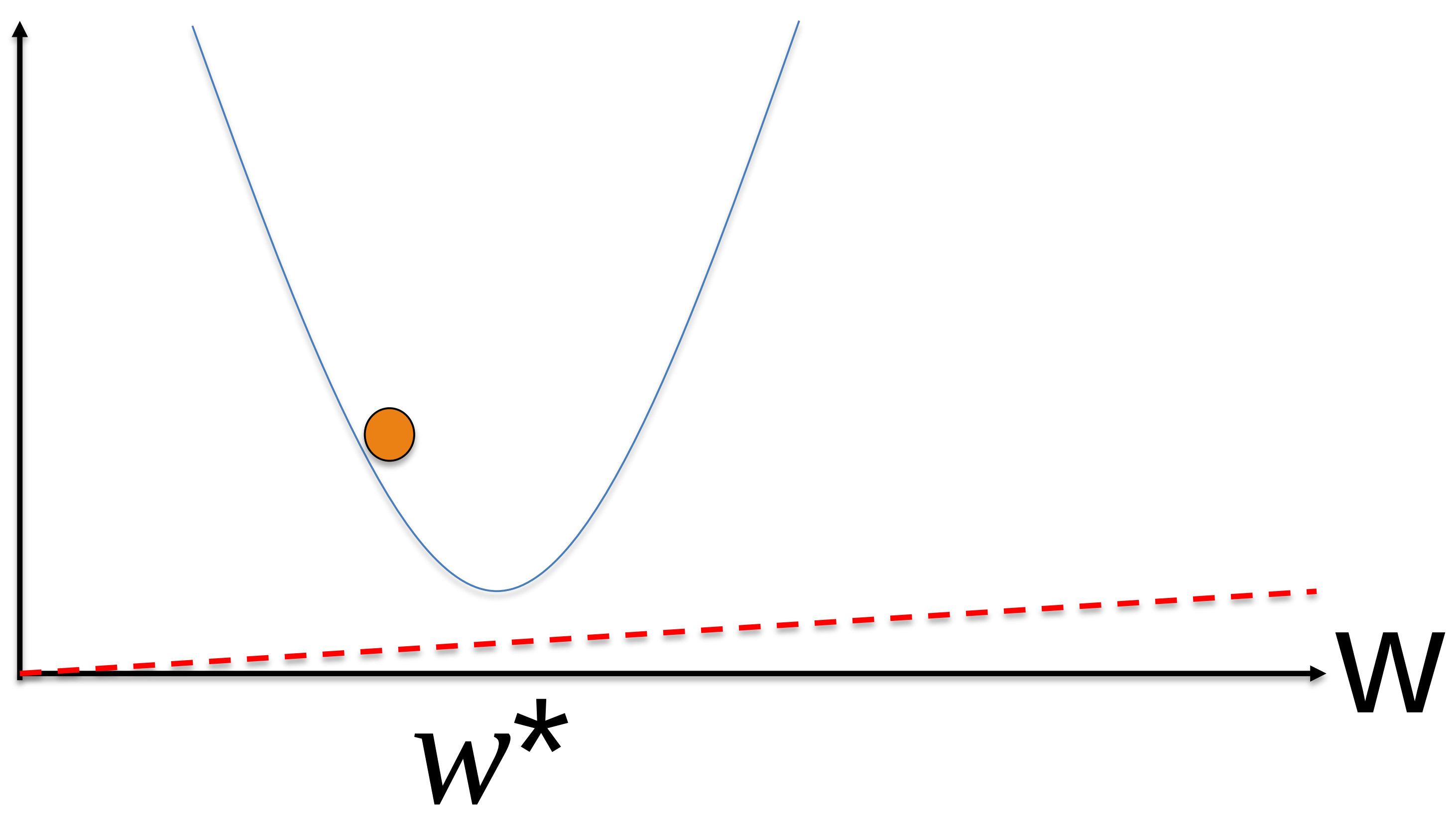
Big curvature β
OR small λ :
Solution at
$w = w^* - \lambda/\beta$

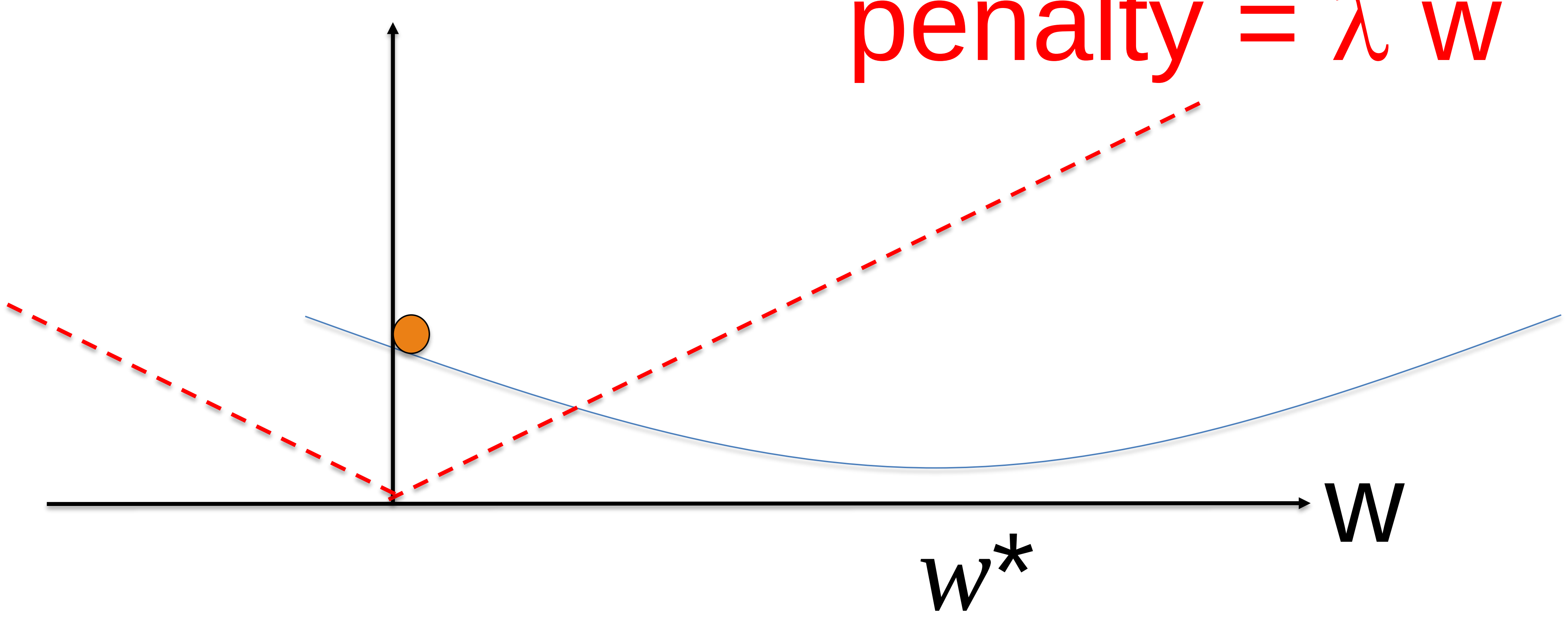


Small curvature β
OR big λ :
Solution at $w=0$

See also ML class of Jaggi-Urbanke

4. L1 Regularization (general)





Big curvature β
OR small λ :
Solution at
w= w^* - λ/β

slope of E at w=0< slope of penalty
→solution at w=0

See also ML class of Jaggi-Urbanke

4. L1 Regularization and L2 Regularization

L1 regularization puts some weights to exactly zero
→ connections ‘disappear’
→ ‘sparse network’

L2 regularization shifts all weights a bit to zero
→ full connectivity remains
→ Close to a minimum and without momentum:
L2 regularization = early stopping
(see exercises)