Artificial Neural Networks

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Deep Nets 3: Loss landscape and optimization methods

Part 1: Questions and Aims of this Lecture

Objectives for today:

- Error function landscape: minima and saddle points
- Momentum
- ADAM
- No Free Lunch Theorem
- Shallow versus Deep Networks

Reading for this lecture:

Goodfellow et al.,**2016** *Deep Learning* *(MIT Press)*
- Ch. 8.2, Ch. 8.5
- Ch. 4.3
- Ch. 5.11, 6.4, Ch. 15.4, 15.5

Further Reading for this Lecture:

Johanni Brea et al. (2019), Weight space symmetry in deep networks gives rise to …
arXiv https://arxiv.org/pdf/1907.02911.pdf

Berfin Simsek et al., Geometry of the loss landscape in overparameterized neural networks …
ICML 2021, PMLR 139:9722-9732, 2021.

Artificial Neural Networks
week of April 12 (next week):

EPFL

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Inverse classroom setting:

- Watch videos at home:

https://lcnwww.epfl.ch/gerstner/VideoLecturesANN-Gerstner.html

Deep Learning Lecture 4. **Statistical Classification by Neural Networks** (90 min) Part 1 - The statistical view: generative models (6 min)
Part 2 - The likelihood of data under a model (12 min)
Part 3A - Statistical interpretation of artificial neural networks: the cross-entropy loss function (19 min)
Part 3B - Statistical interpretation of artificial neural networks: can we interpret the output as a probability? (18 min)
Part 4 - Sigmoidal units as natural output functions (9 min)
Part 5 - Multi-class problems (19 min)
Part 6 - Statistical approach: Summary and Quiz (7 min)

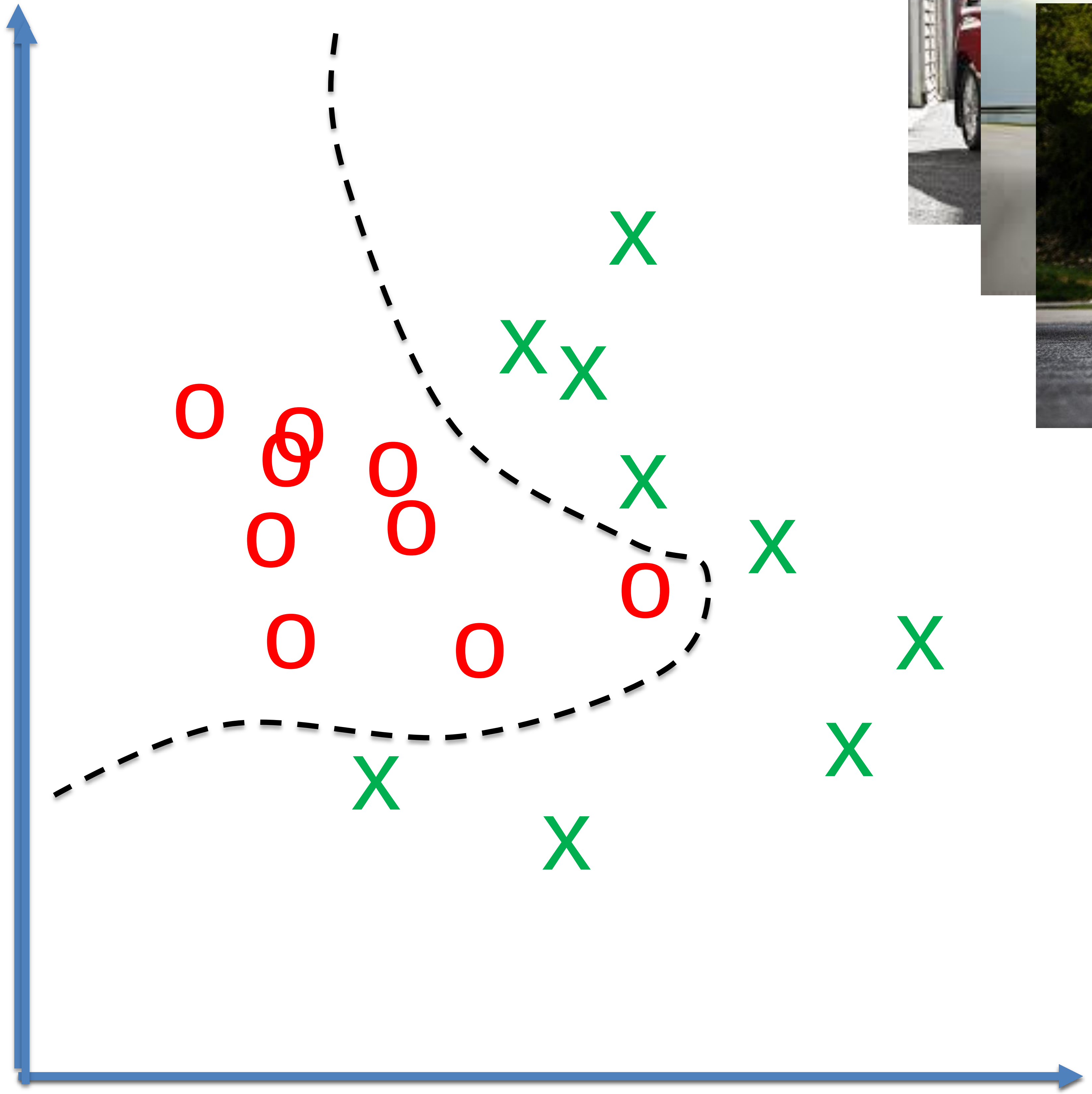
Come to class at 14h15:

- Introduction to, and hand-out of, miniprojects
- Q&A session
- Exercise session (week 8/statistical classification)

Previous slide.

A multilayer perceptron for classification

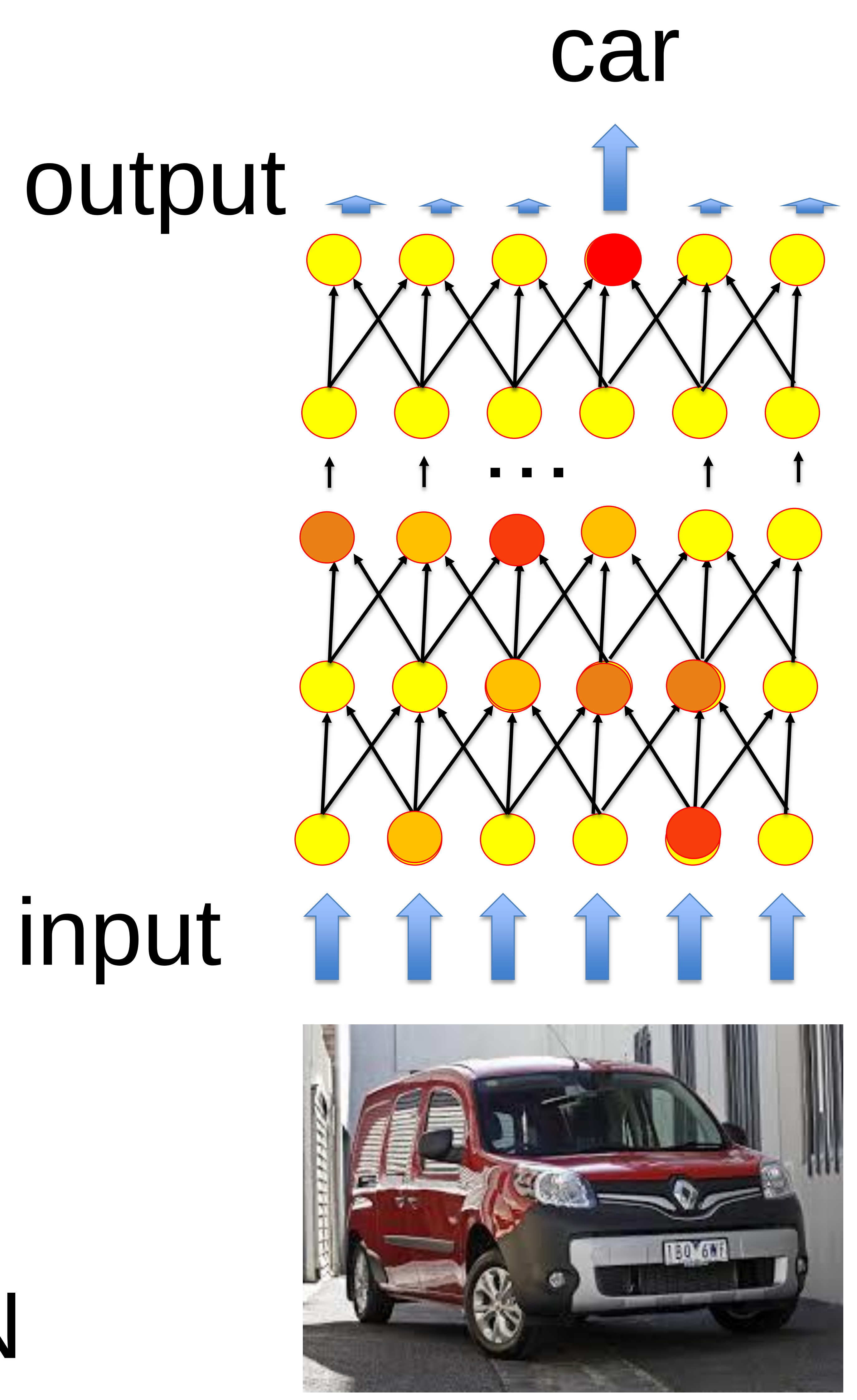
Review: Classification as a geometric problem



Previous slide.

A multilayer perceptron for classification

Review: Deep Neural Networks for classification



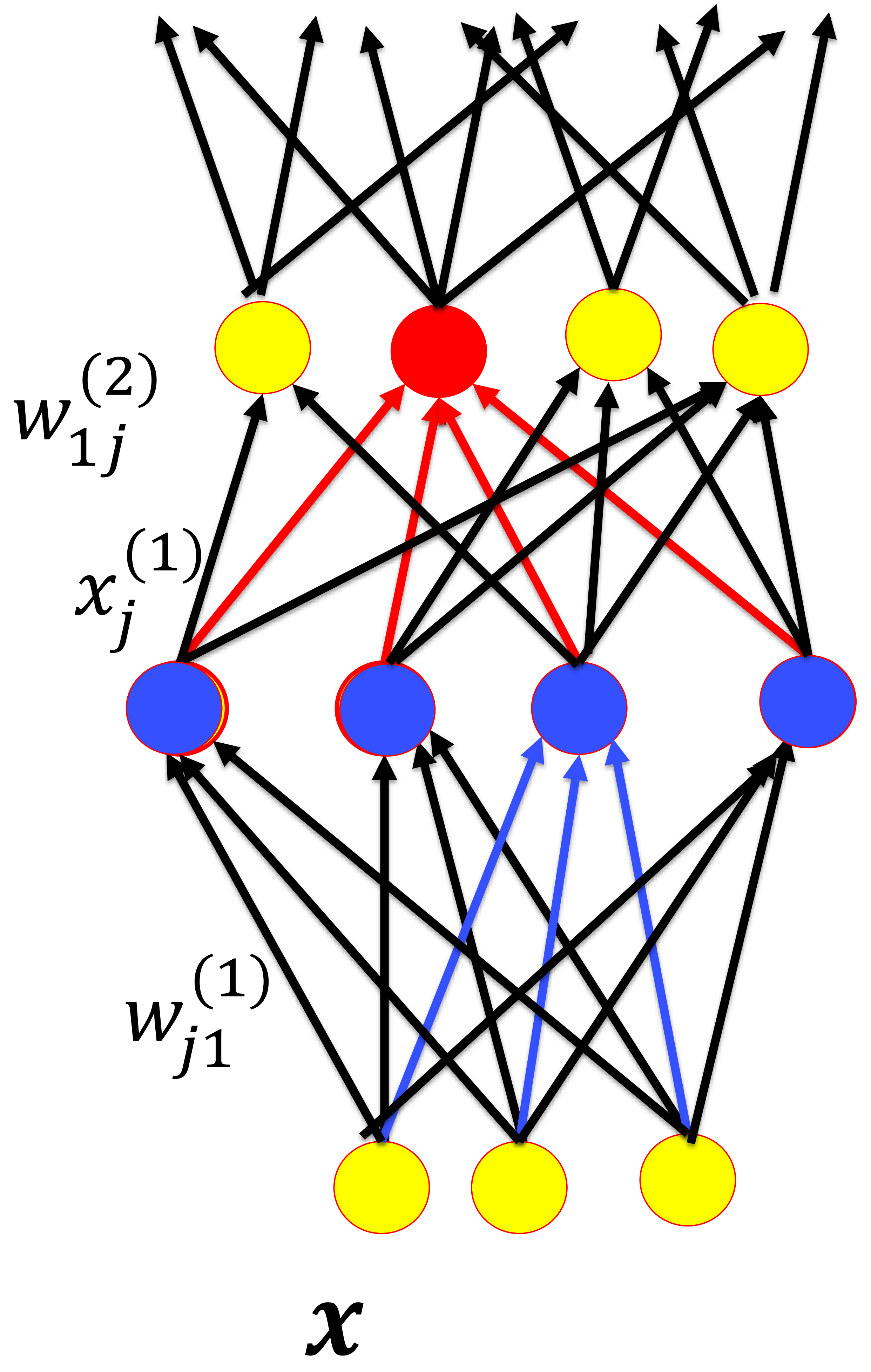
Aim of learning:
Adjust connection weights such that output is correct.

Total number of parameters: N

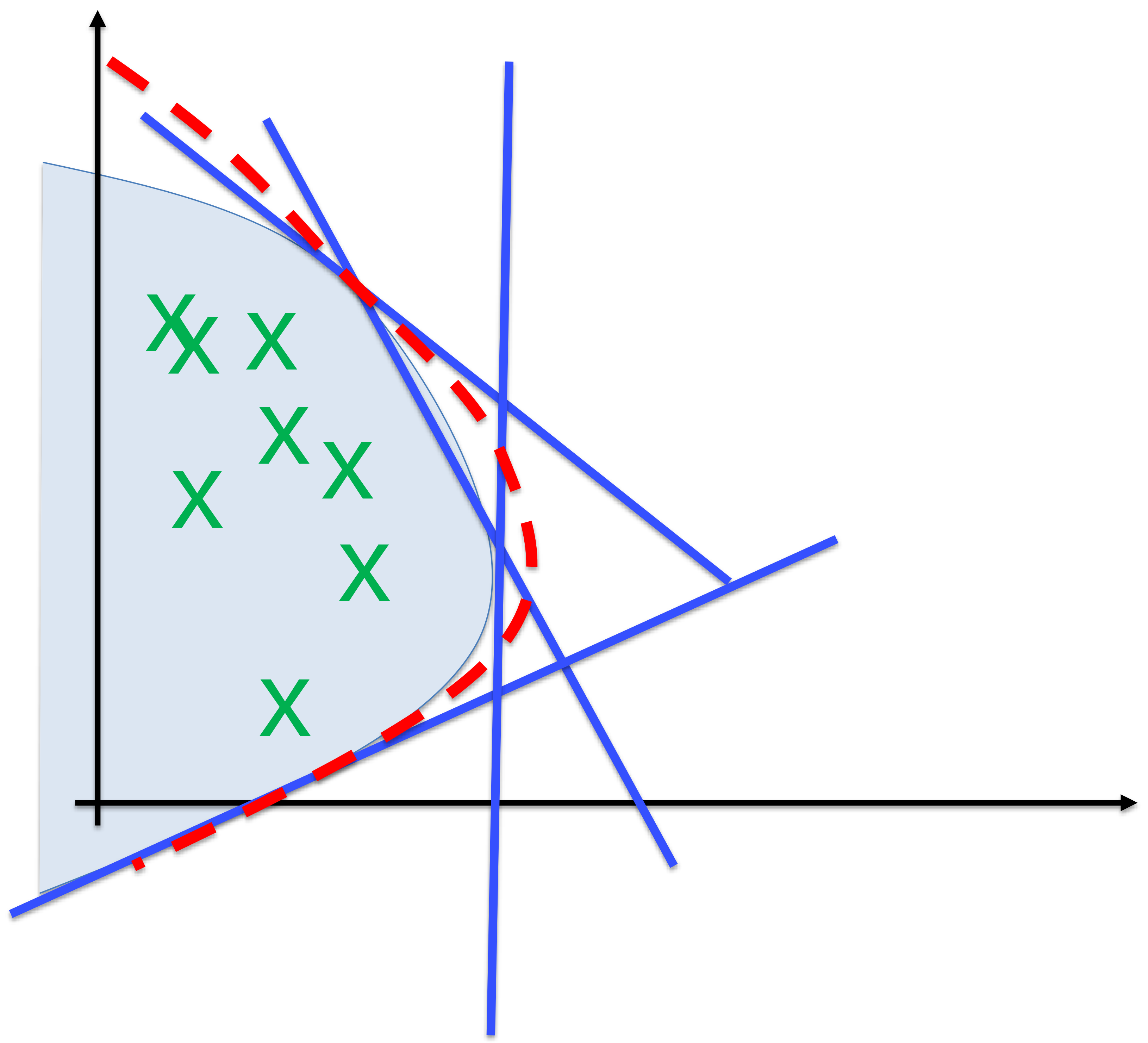
Previous slide.

… will implement a separating surface …
… by stacking neurons over several layers. Each neuron implements a hyperplane in the space of activites one layer below.

Review: task of hidden neurons (blue)



hidden neurons implement hyperplanes



Previous slide.

… by stacking neurons over several layers. Each neuron implements a hyperplane in the space of activites one layer below. Hyperplanes are defined by weight vectors.

Review: gradient descent

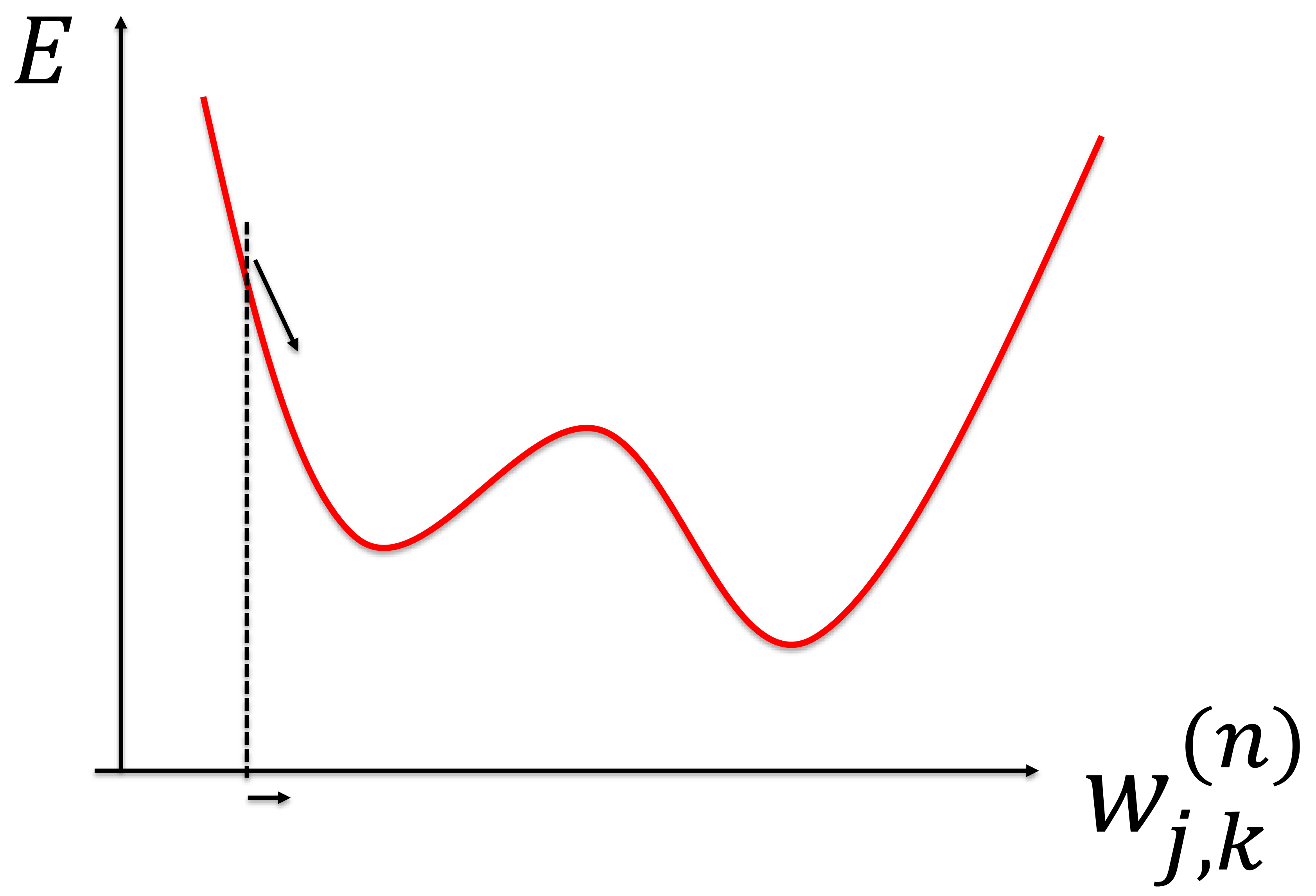
error function (loss function)

$E(\boldsymbol{w})$

gradient descent

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

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



Previous slide.

And the weight vector is updated by gradient descent, using either a batch rule or an online rule.

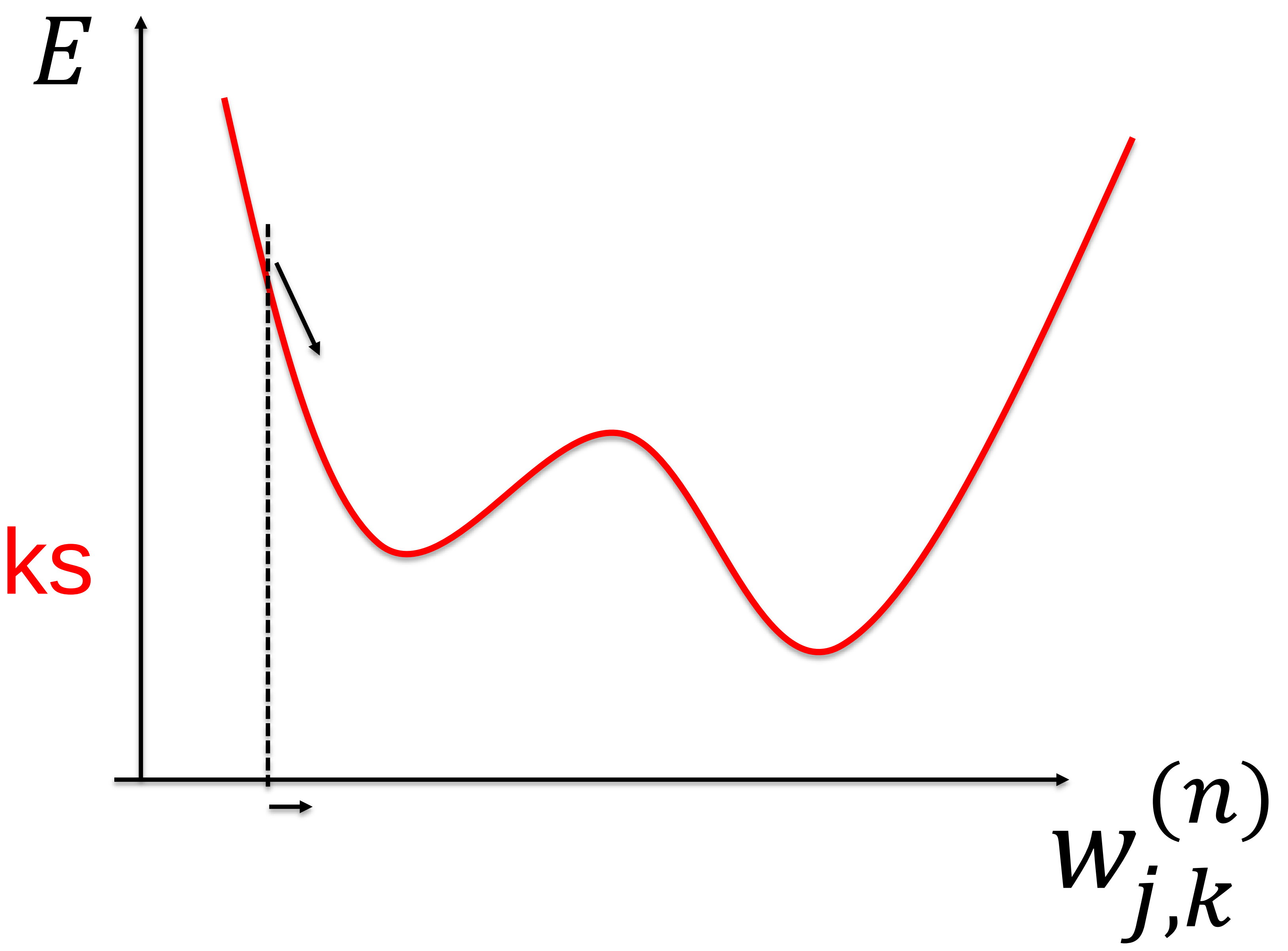
We discuss

Three Big questions for today

- How does the error landscape (as a function of the weights) look like? In high dimension?
  - → Count the minima and saddles (lower bound)

- How can we quickly find a (good) minimum?
  - → Momentum term
  - → ADAM optimizer

- Why do deep networks work well?
  - → No Free Lunch Theorem
  - → Deep Networks versus Shallow Networks



Previous slide.

We address three important questions today.

1. What is the shape of the error function, as a function of the weights?
   → Count the minima and saddles (lower bound)

2. How can we quickly find a good minimum?

   → Momentum Term, ADAM optimizer

3. Why do deep networks work so well in practice?

   → No free lunch theorem; and shallow versus deep networks

Artificial Neural Networks

Wulfram Gerstner
EPFL, Lausanne, Switzerland

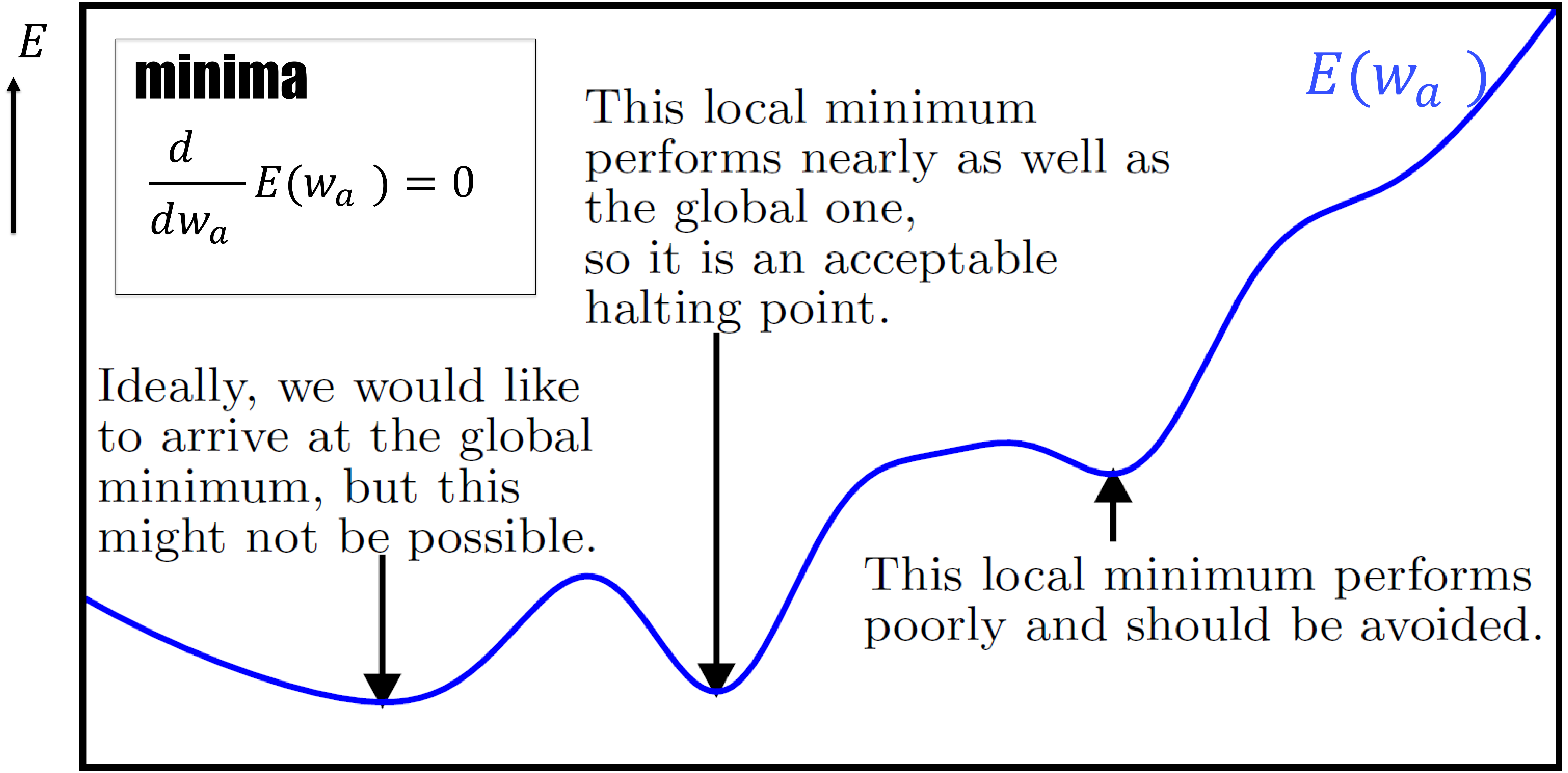
Loss landscape and optimization methods for deep learning

Part 2: Error function: minima and saddle points

Previous slide.

We start with the first question and focus on the error function.

Error function: minima


E
minima
$\frac{d}{dw_a} E(w_a) = 0$
$E(w_a)$
This local minimum performs nearly as well as the global one, so it is an acceptable halting point.
Ideally, we would like to arrive at the global minimum, but this might not be possible.
This local minimum performs poorly and should be avoided.

w_a

Image: Goodfellow et al., Deep Learning, 2016

Previous slide.

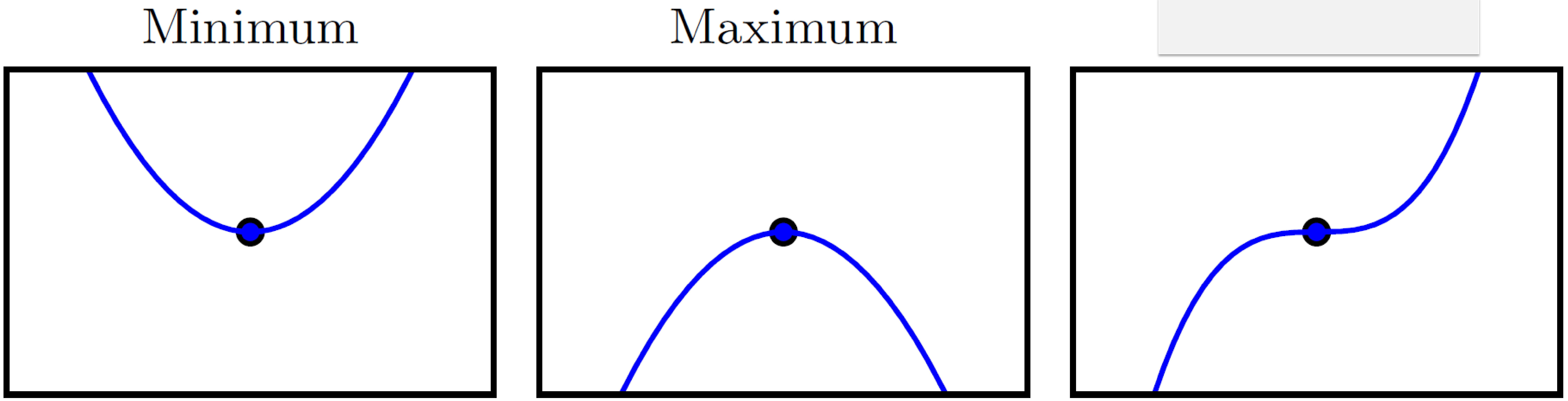
Often we see hand-drawn sketches of one-dimensional plots like the one here, with several local minima.

Error function: minima

minima

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

How many minima are there in a deep network?



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

Image: Goodfellow et al., Deep Learning, 2016

Previous slide.

Both minima and maxima are characterized by a zero derivate:

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

In one dimension, minima can be distinguished from maxima by their second derivative (curvature).
For minima the curvature is positive (left):

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

Transient plateaus where both first and second derivative are zero are the exception (right)

Error function: minima and saddle points

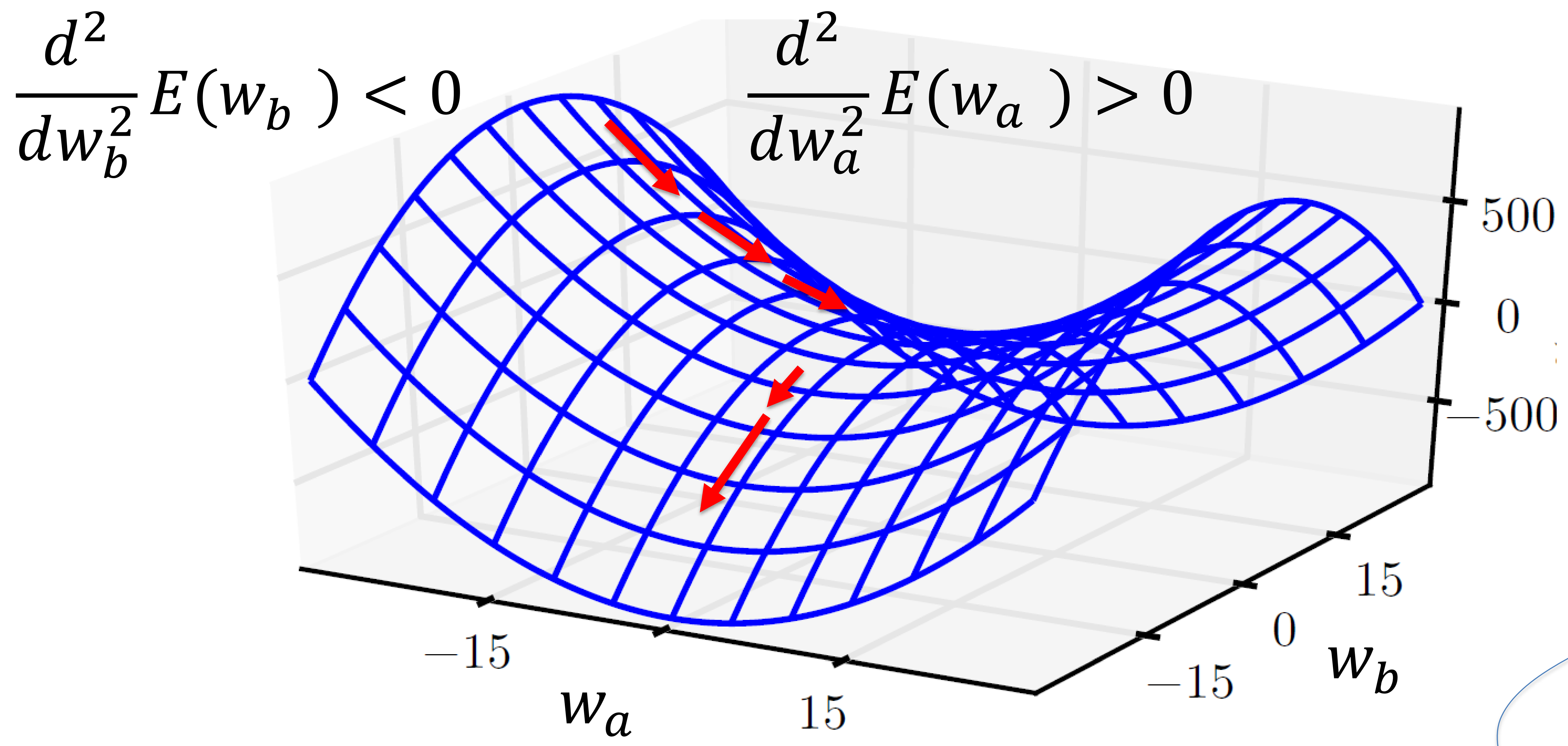
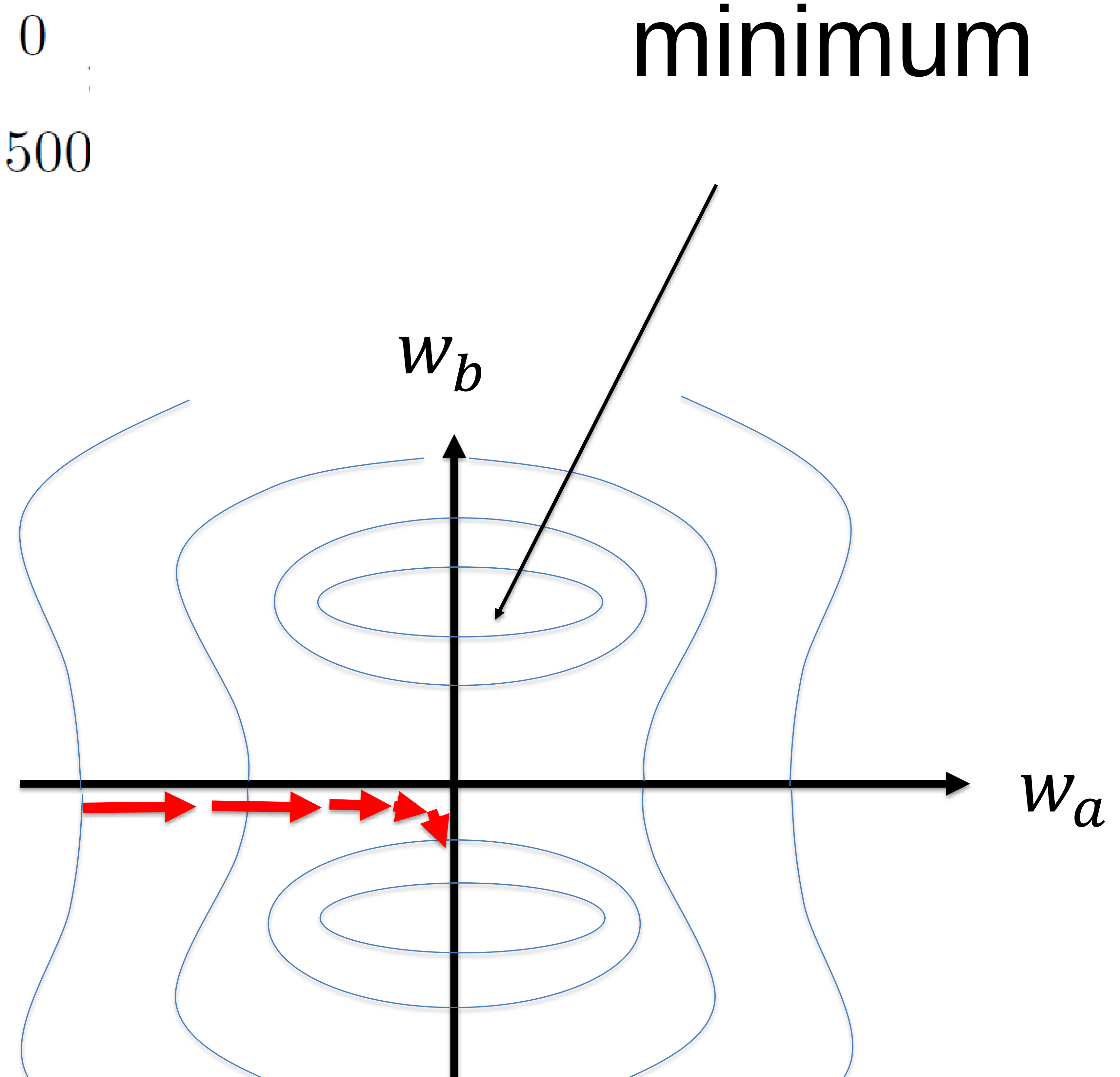


Image: Goodfellow et al., Deep Learning, 2016



2 minima, separated by
1 saddle point

Previous slide.

In two and more dimensions it is possible that in the curvature is positive in one direction, yet negative in the other direction.
This is called a saddle point.

Lower right: contour lines connect points of the same error (niveau lines). The red arrows indicate a path toward a minimum. The two minima are separated by a saddle.

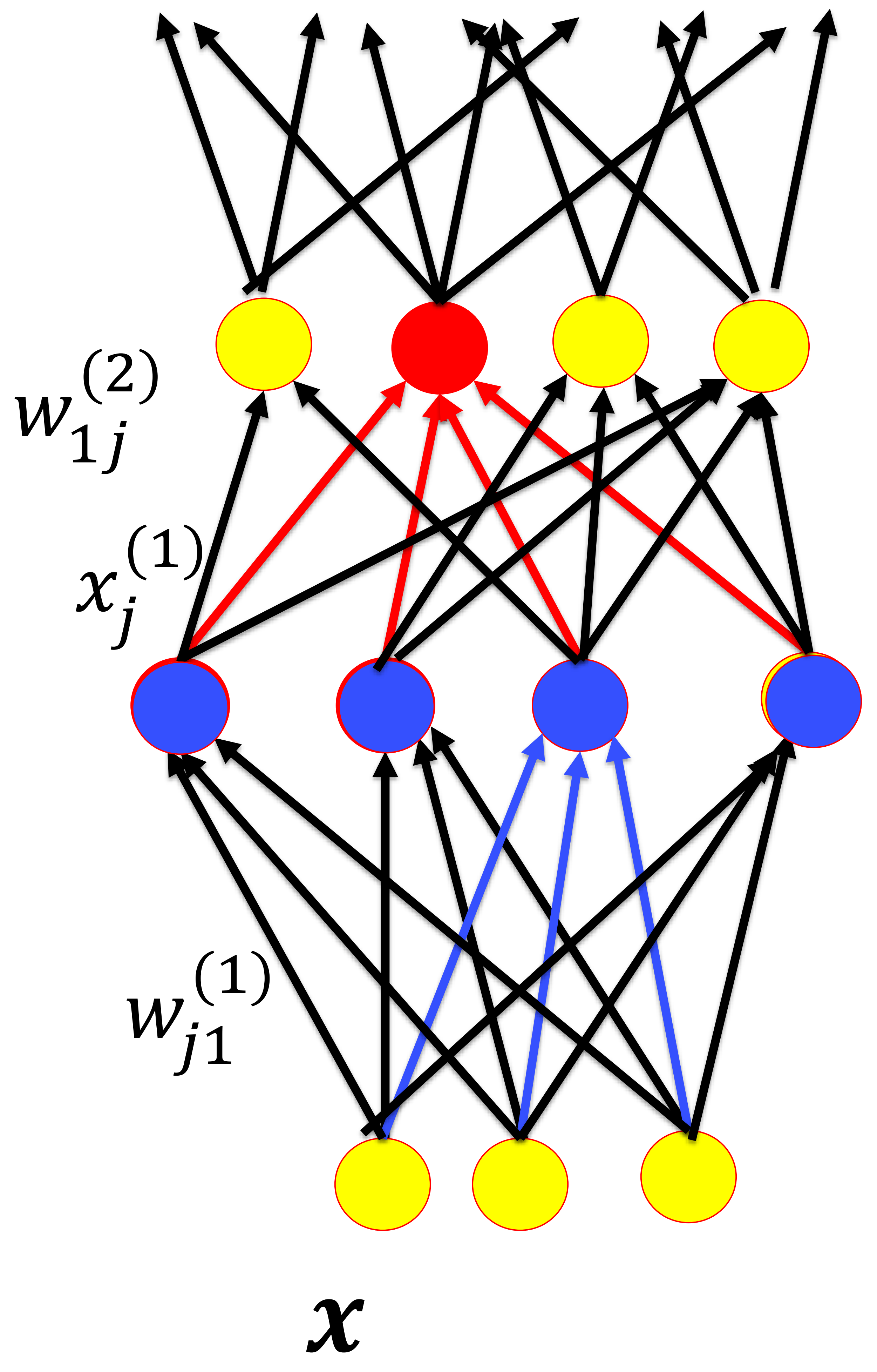
Quiz: Strengthen your intuitions in high dimensions

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

Your notes.

Minima of loss function



How many minima are there?

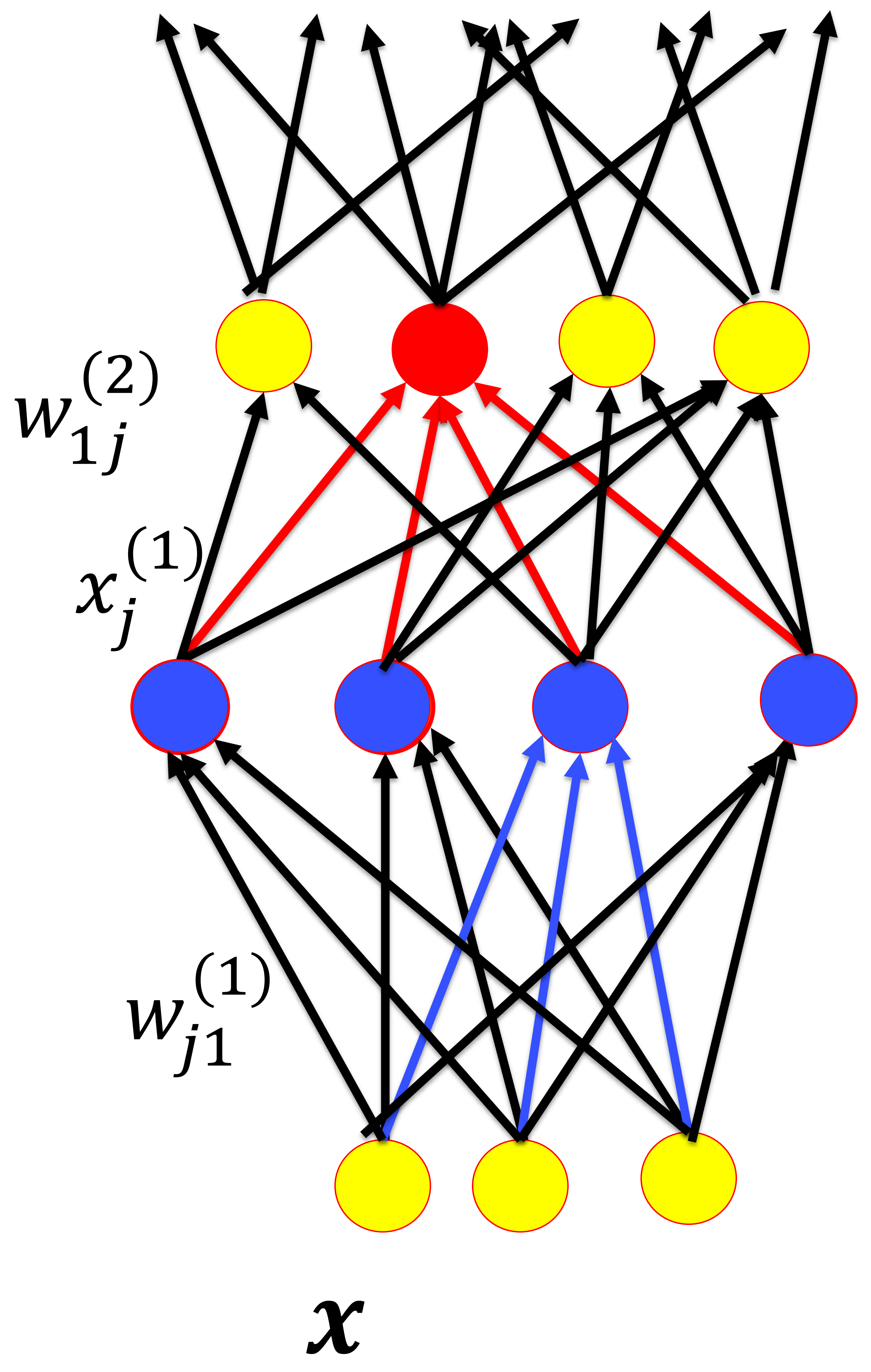
Answer:
In a network with m hidden layers and n neurons per hidden layer,

Previous slide.

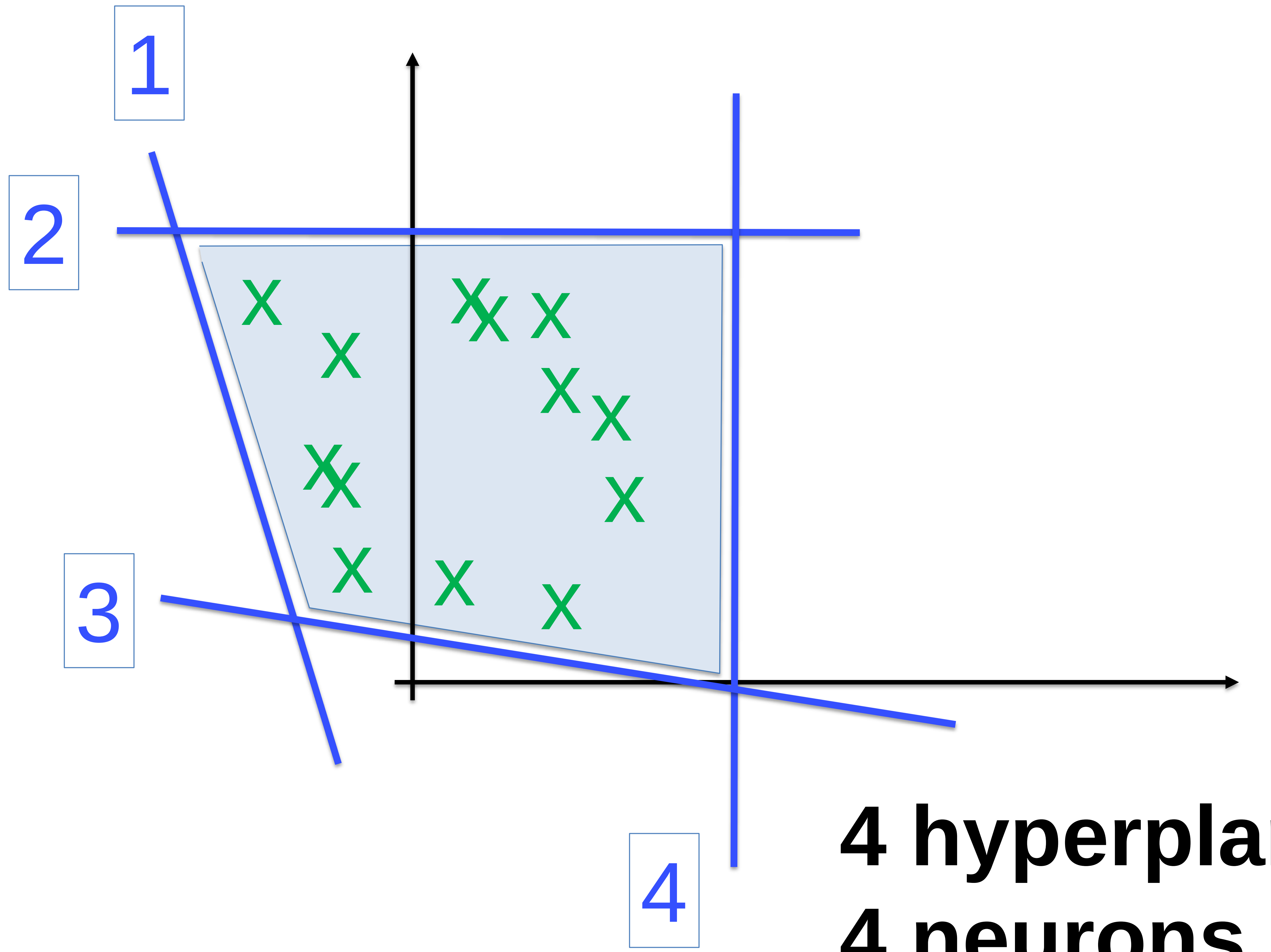
Because of the permutation symmetry, there are many equivalent minima.

(See Exercises)
.

Error function and weight space symmetry



many assignments
of hyperplanes to neurons



4 hyperplanes for 4 neurons

Previous slide.

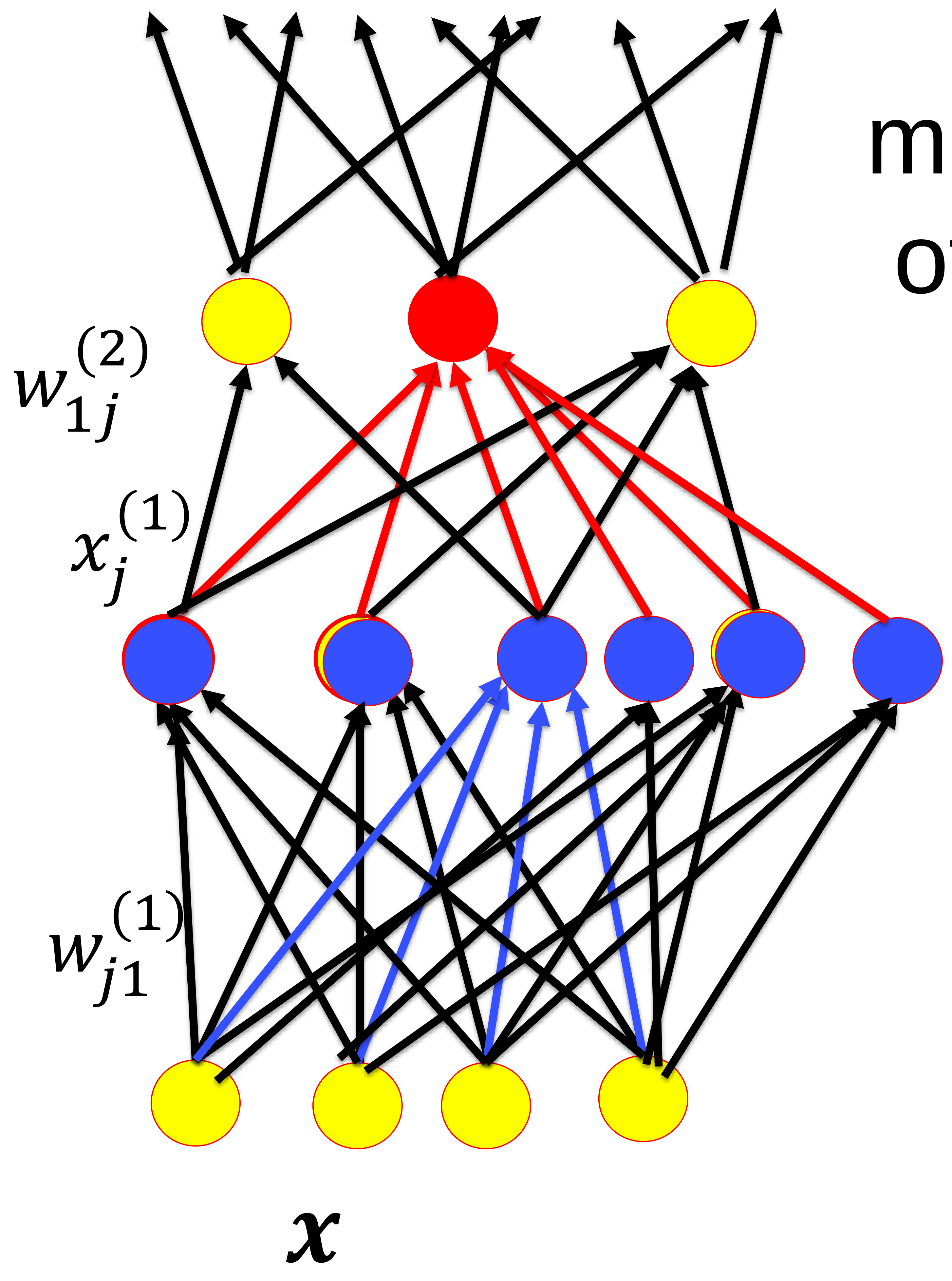
For example, with 4 neurons in a given layer, we have 4! different ways to implement the same 4 hyperplanes.

In total, in a network of m hidden layers with n neurons each there are

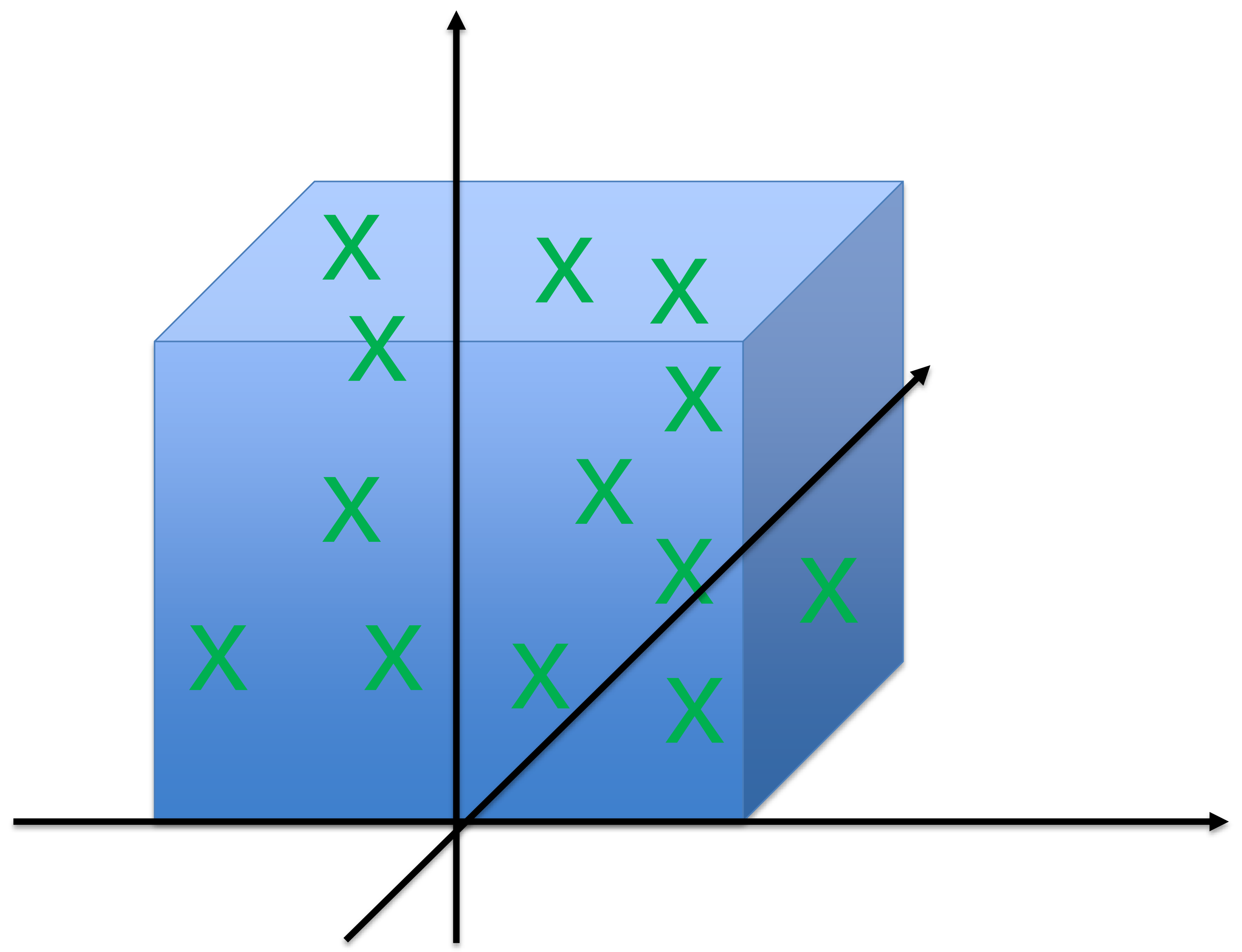
$$\boldsymbol{n}!^{\boldsymbol{m}}$$

equivalent solutions. Therefore there are many permutation symmetries in the weights space.

Error function and weight space symmetry



many assignments of hyperplanes to neurons



even more permutations

6 hyperplanes for 6 hidden neurons

Previous slide.
Suppose all the positive examples lie inside a the blue box.

We need 6 neurons in the first layer to define this box. Each neuron implements one hyperplane. Therefore there are 6! = 240 different, but completely equivalent solutions.

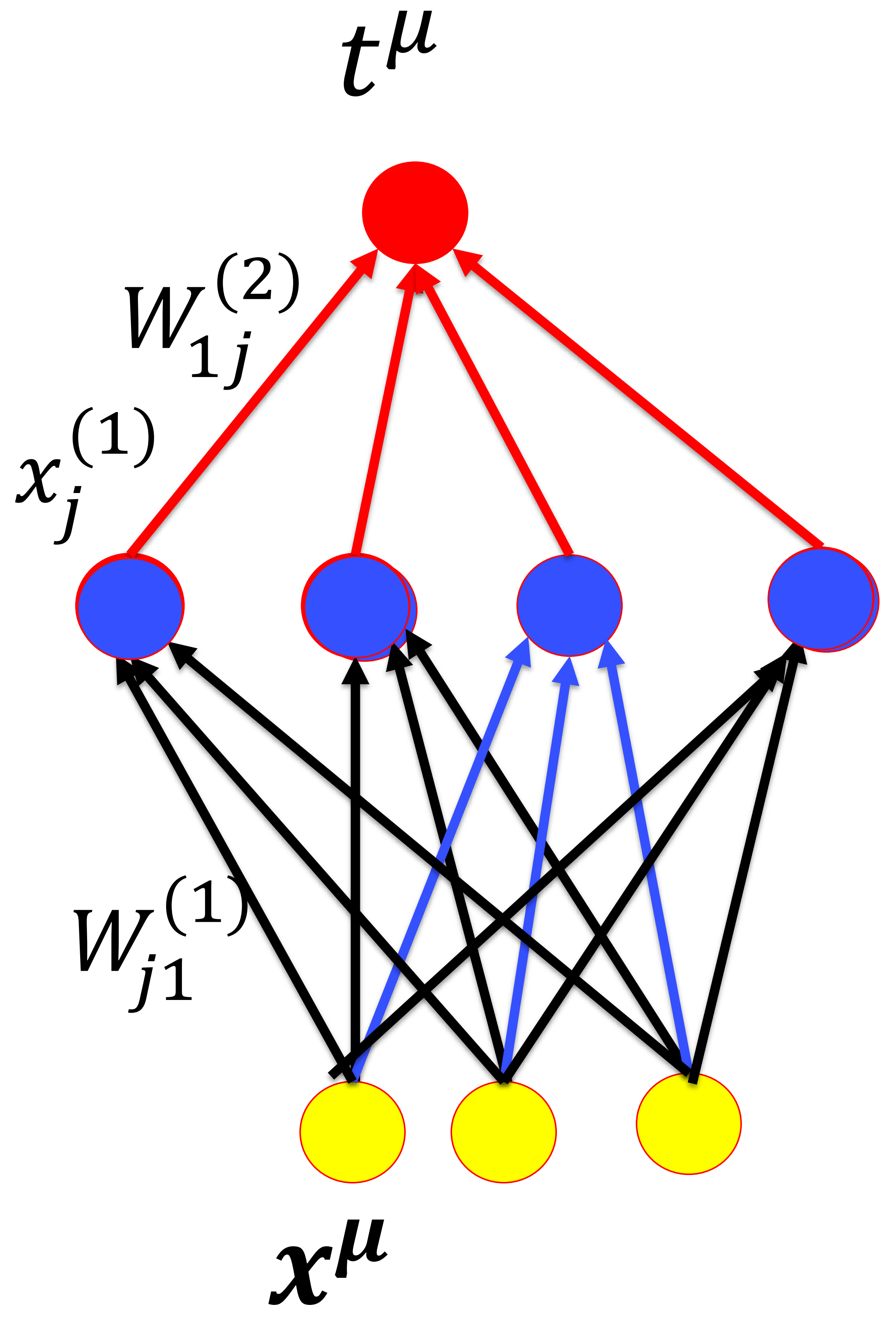
Note that the result does not depend on the input dimensionality. It is three in the graph, but it could be 2 or 7.

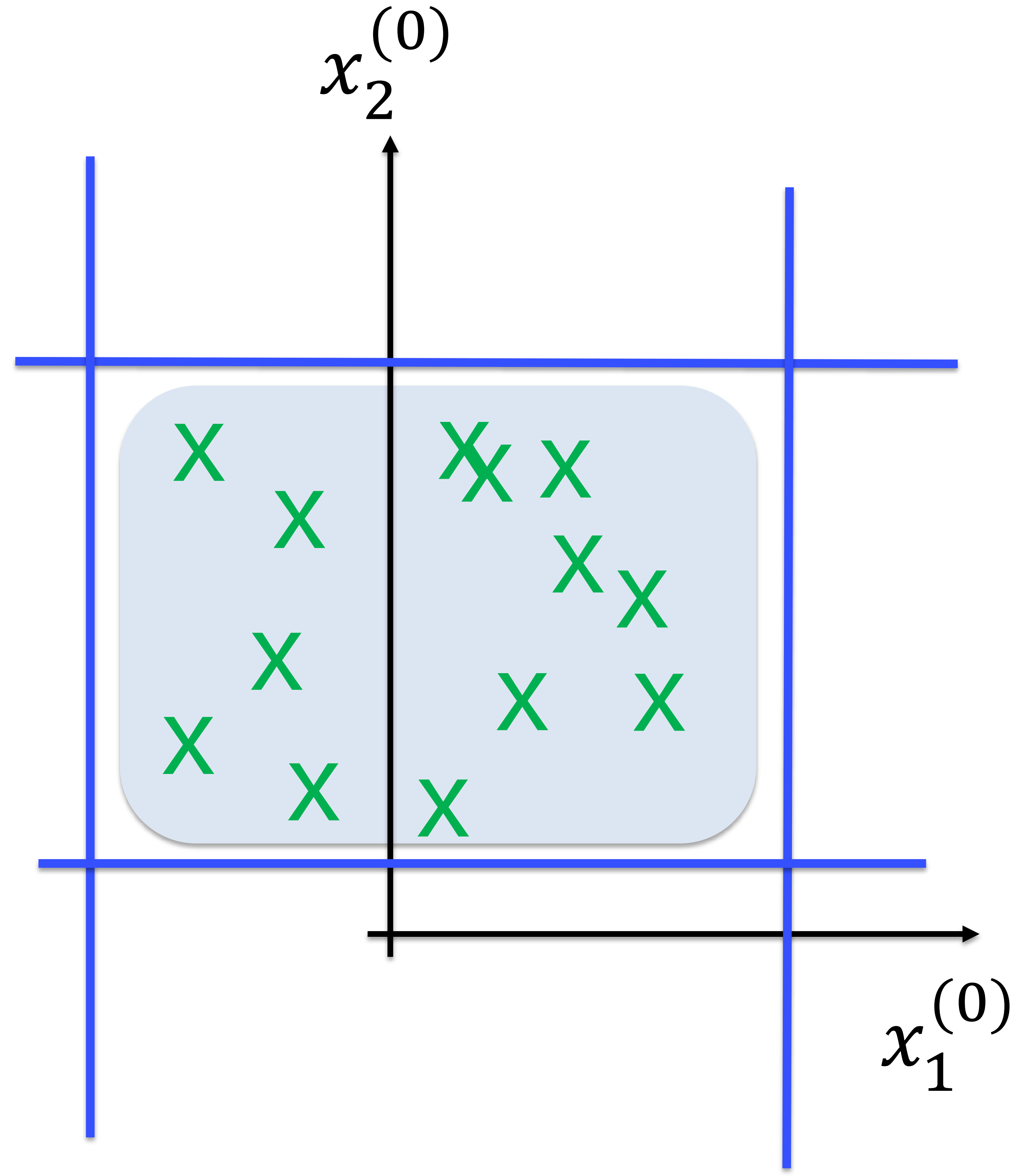
Teacher-Student setup: Global minima guaranteed

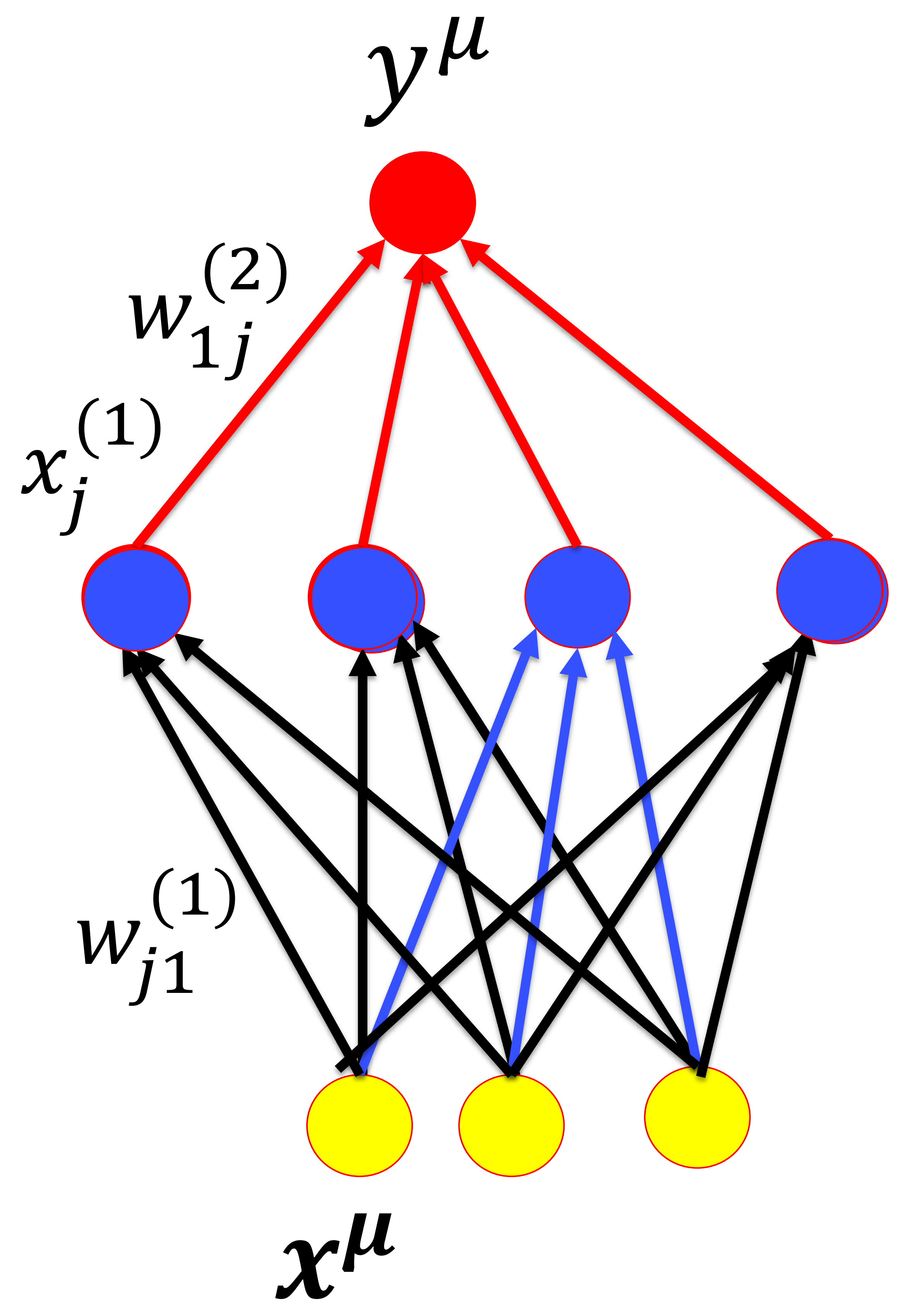
teacher network:
generates labels

4 neurons
4 hyperplanes

student network:
learns outputs



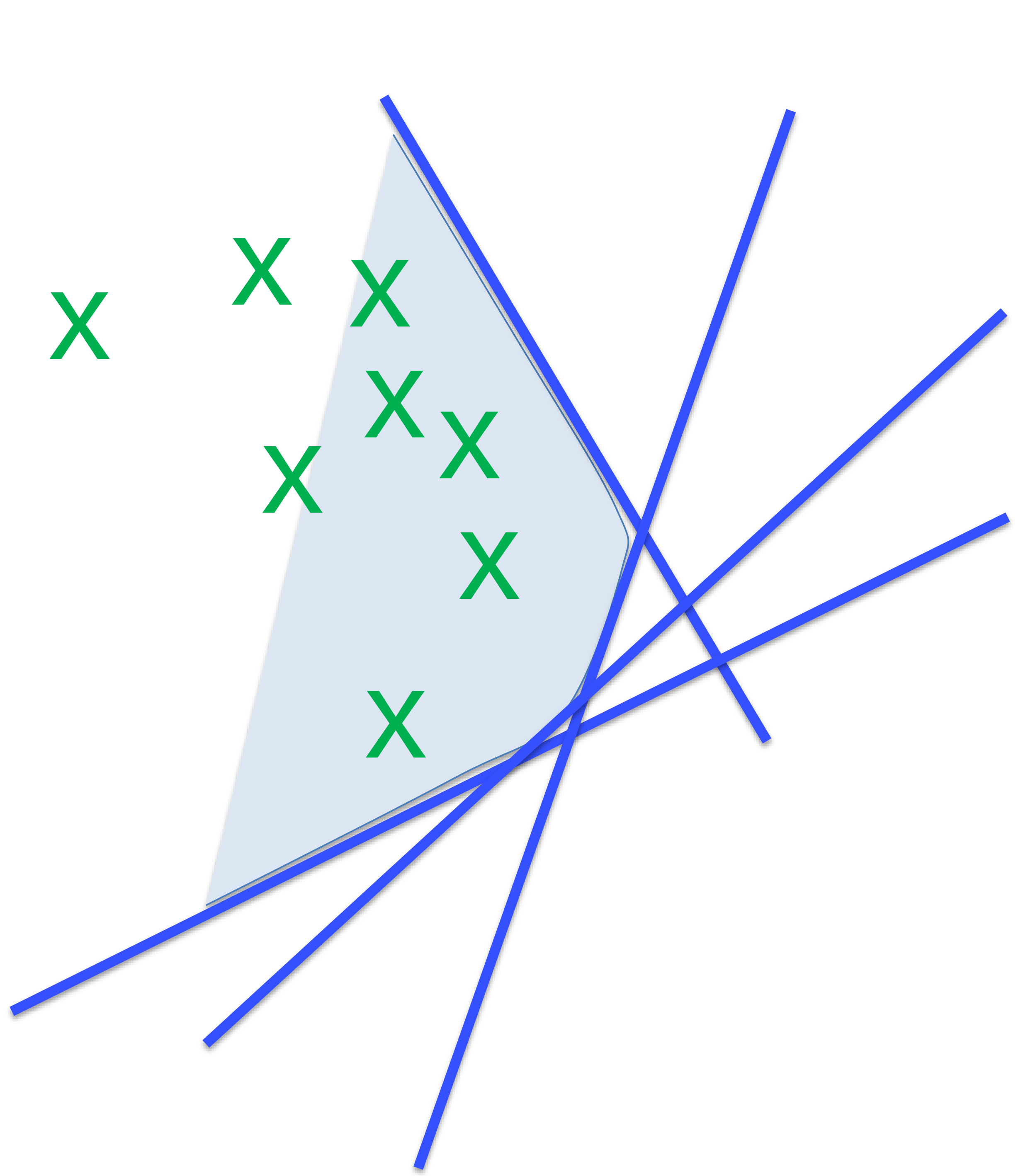


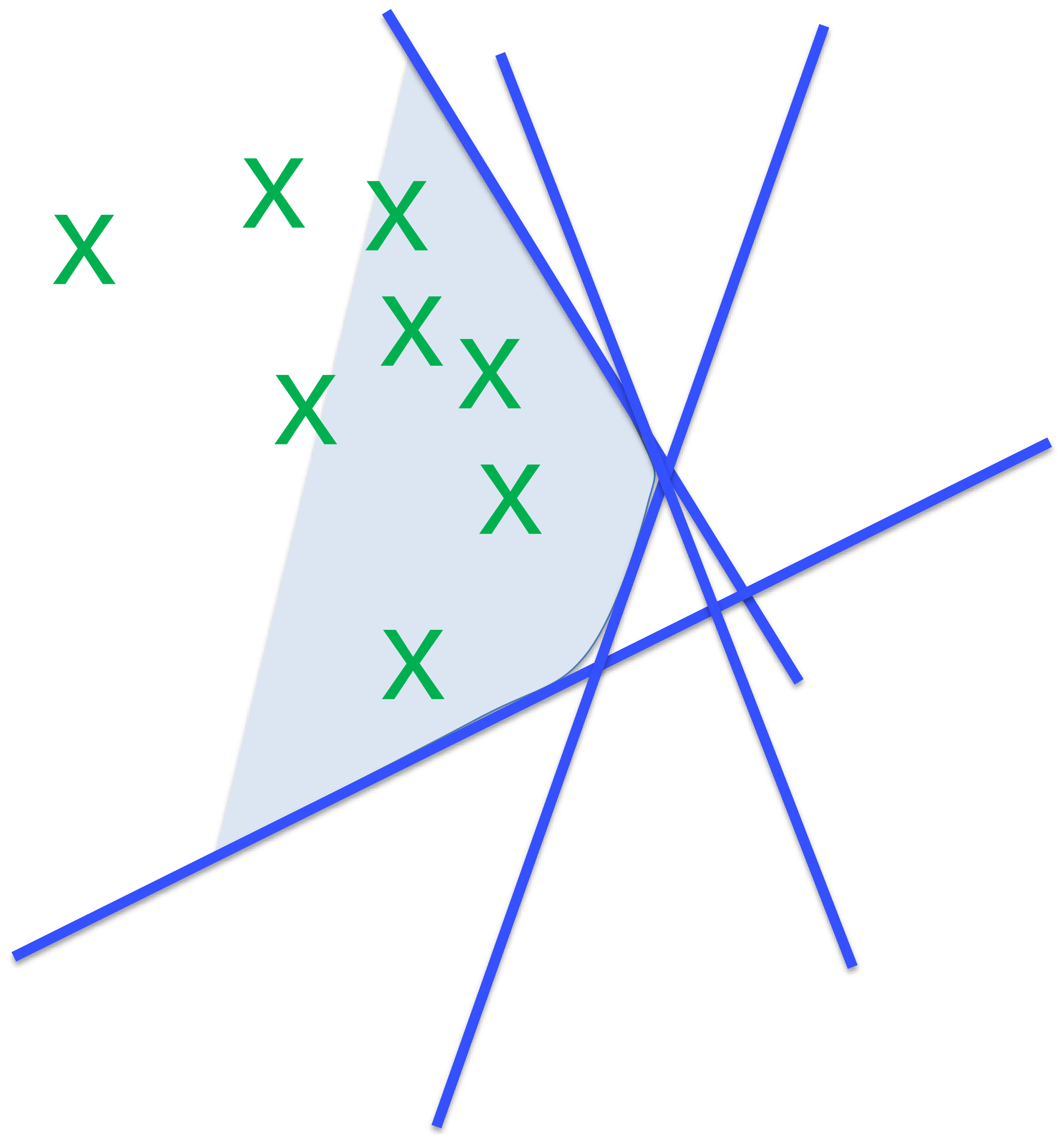


Previous slide.

So far we focused on the ‘best’ minima: in a teacher-student situation where the student network has exactly the same architecture as the teacher, the best minima are those where the student has the same weight vectors (apart from permutations).

Arbitrary data: Many near-equivalent 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

Previous slide.

However, real data is not generated from a teacher network of known architecture. Therefore all solutions are approximate solutions and finding zero-loss solutions is the exception.

Then you will typically find many near-equivalent reasonably good solutions.
For an example, suppose that the data (positive examples) lie in the shaded area. There are several near-equivalent solutions of modeling the boundaries of this shaded area with 4 hyperplanes.

If you increase to 8 hyperplanes even more near-equivalent solutions appear.

Summary Quiz: Number of minima in deep networks

A deep neural network with many neurons

[ ] has always **many** equivalent ‘optimal’ solutions (global minima)

[ ] has typically **many** near-optimal solutions, not related by permutation symmetry

Previous slide.

Artificial Neural Networks

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Loss landscape and optimization methods for deep learning

Part 3: Why are there so many saddle points?

1. Questions and Aims of this Lecture
2. Error function: minima and saddle points
3. **Why are there so many saddle points?**

Previous slide.

We now look at saddles

Loss function: why are saddle points relevant?

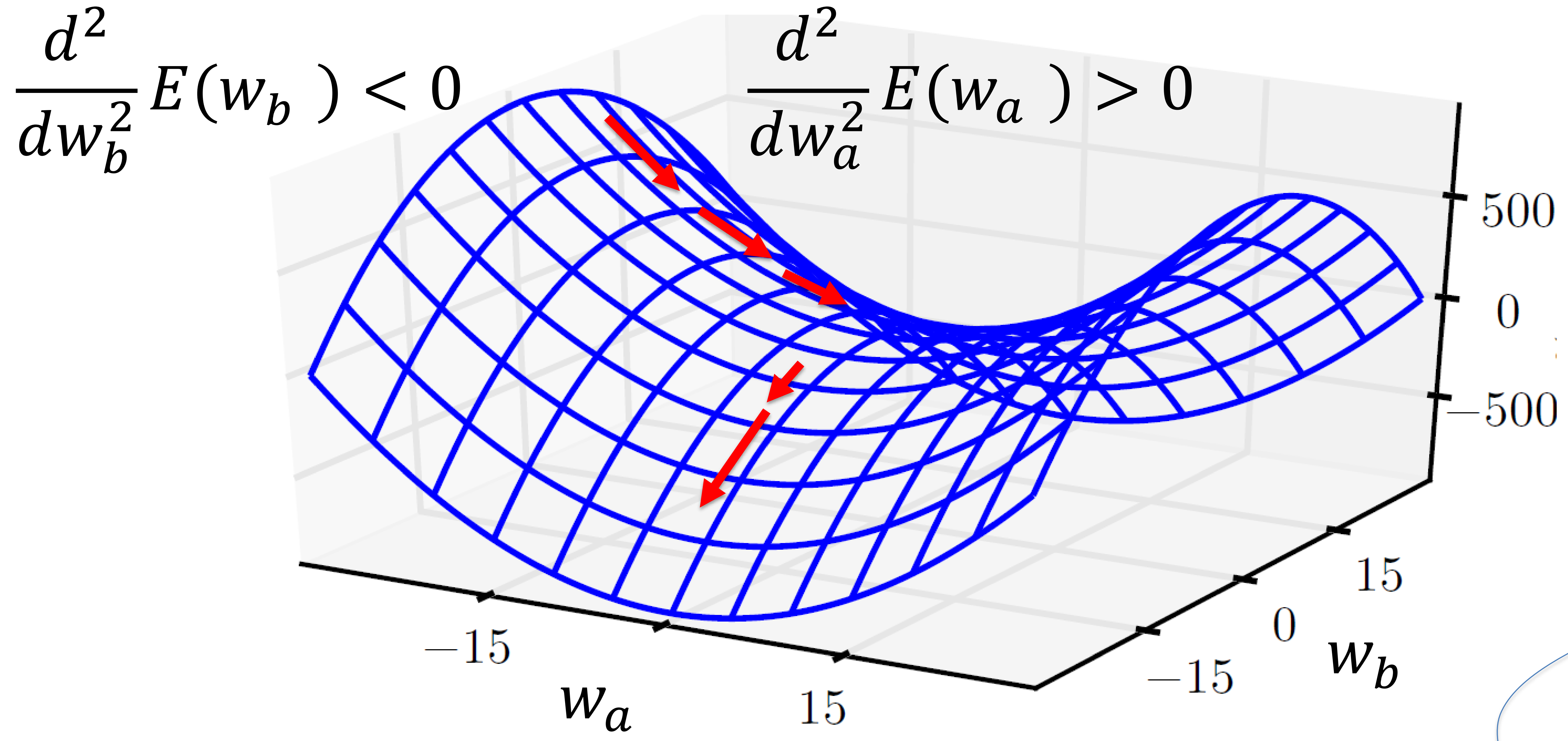
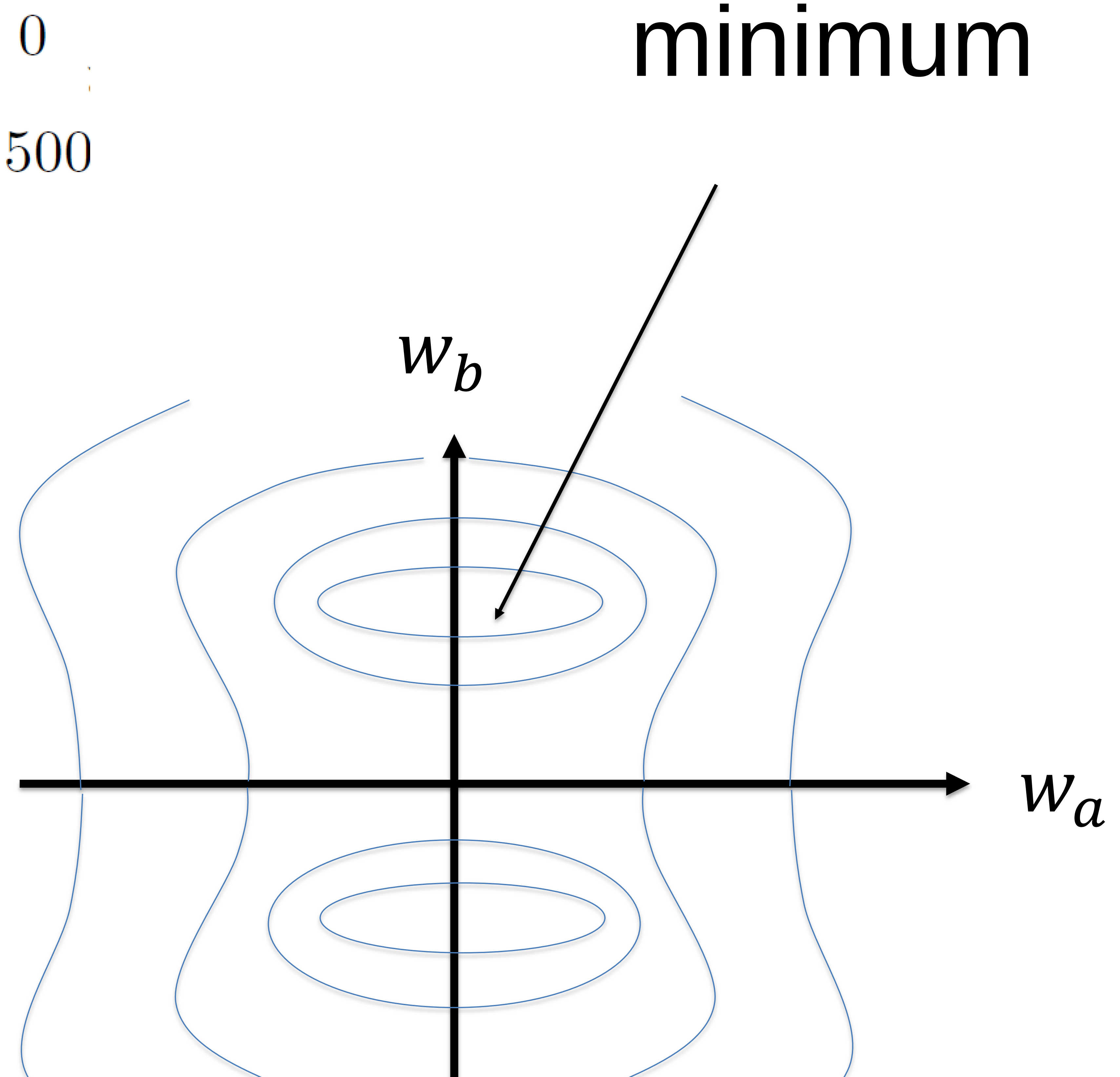


Image: Goodfellow et al. 2016

Gradient descent is slow close to saddle



Your notes.

We are interested in saddles because these are critical points where gradient descent is slow.

Minima and saddle points

Claim:

There are many more saddle points than minima

Two arguments

(i) Statistical argument

→ Hessian Matrix

(ii) Geometric argument

→ Permutations

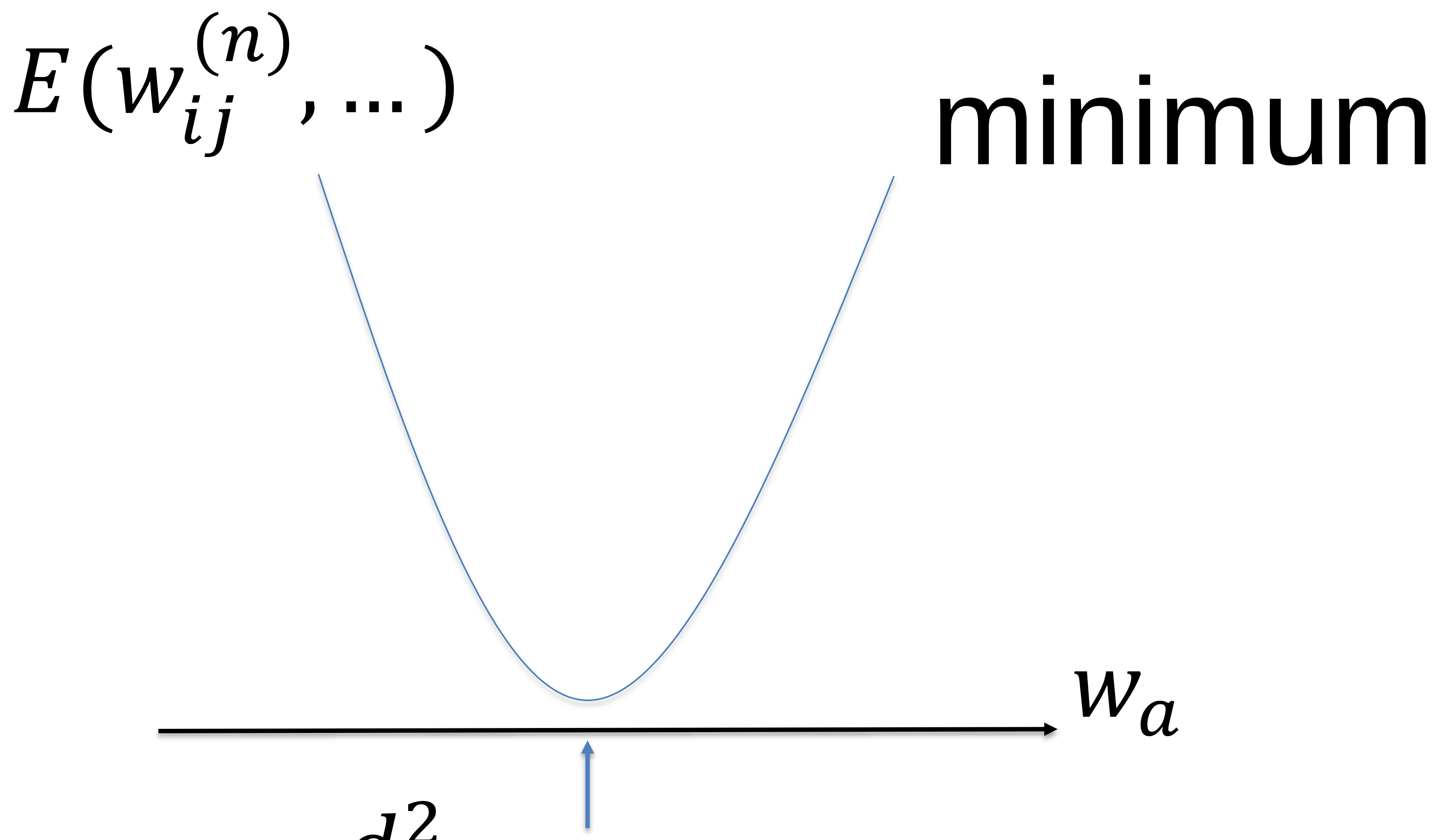
Previous slide.
The claim is that there are many more saddle points than minima.

There are two different arguments. We start with the first one.

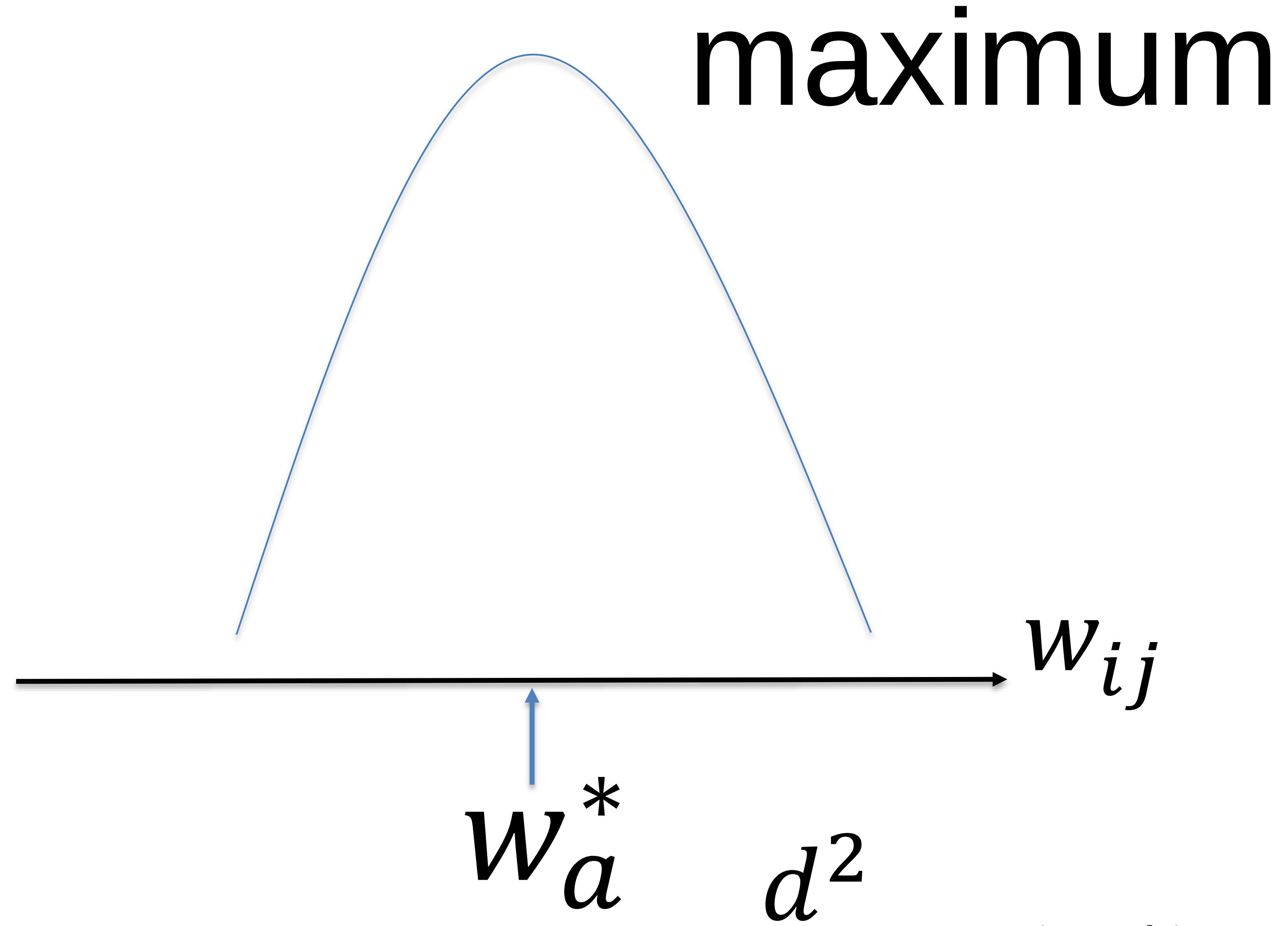
Minima and saddle points

There are many more saddle points than minima

(i) Statistical argument on second derivative
(Hessian matrix) at gradient zero



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



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

Previous slide.

The first argument focuses on the Hessian matrix of second derivatives evaluated at the location where the first derivative vanishes.

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!

Previous slide.

Since the Hessian matrix is symmetric, it is diagonalizable and has real Eigenvalues.

A point is stable only of ALL eigenvalues are positive.

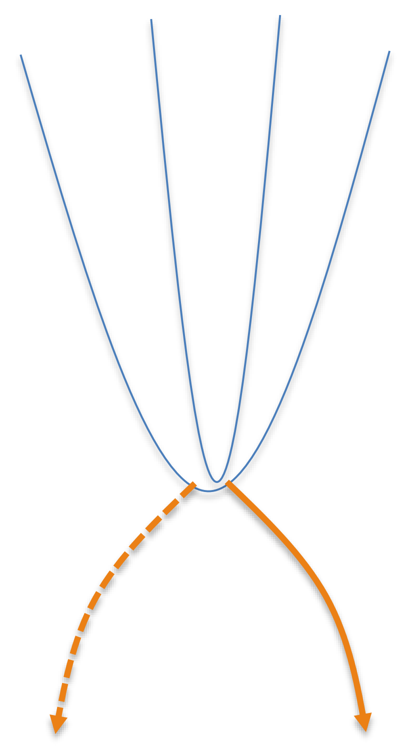
Minima and saddle points

in N dim: **Hessian**

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

Diagonalize:

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



$\lambda_1 > 0$

$\dots$

$\lambda_{N-1} > 0$

$\lambda_N < 0$

In N-1 dimensions surface goes up,
In 1 dimension it goes down

Previous slide.

If N-1 Eigenvalues are positive, but one is negative, we have a first-order saddle.

Minima and saddle points: Second-order saddle

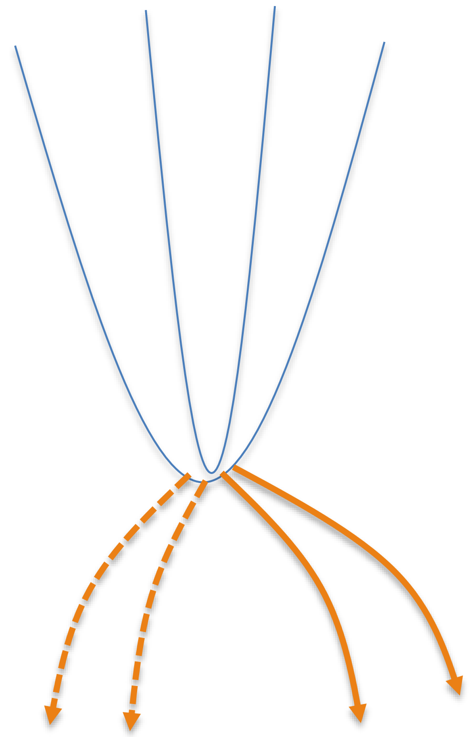
in N dim: **Hessian**

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

Diagonalize:

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

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



$\lambda_1 > 0$

$\ldots$

$\lambda_{N-2} > 0$
$\lambda_{N-1} < 0$
$\lambda_N < 0$

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

Humans visualize
In three dimensions

Previous slide.

If N-2 Eigenvalues are positive, but two are negative, we have a second-order saddle.

Kant: humans necessarily think in 3 dimensions.

Therefore it is hard to imagine that I have 2 dimensions in which the error goes down and N-2 orthogonal directions in which the error goes up. The drawing is very schematic.

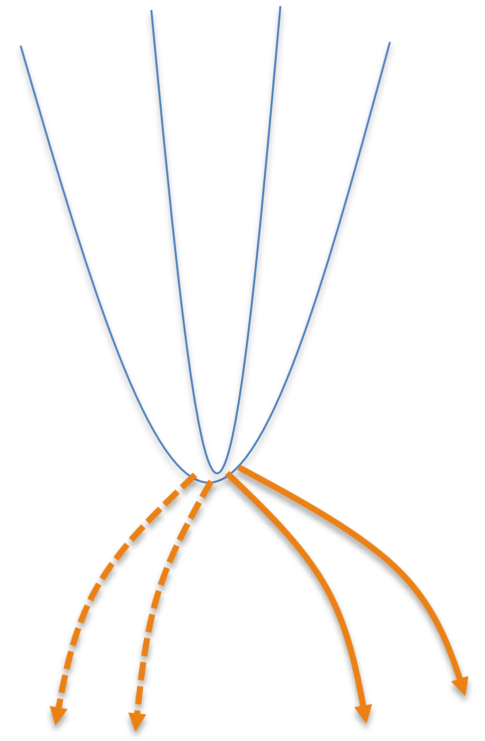
General saddle point: k_{th}-order saddle

in N dim: **Hessian**

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

Diagonalize:

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



$\lambda_1 > 0$

$\ldots$

$\lambda_{N-k} > 0$

$\lambda_{N-k+1} < 0$

$\lambda_N < 0$

In *N*-*k* dimensions surface goes up,
In *k* dimension it goes down

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

Previous slide.

Analogously, we define a general saddle.

Minima and saddle points: statistical argument

Suppose you create eigenvalues randomly with mean zero.

It is (statistically) 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 ones with surface going down

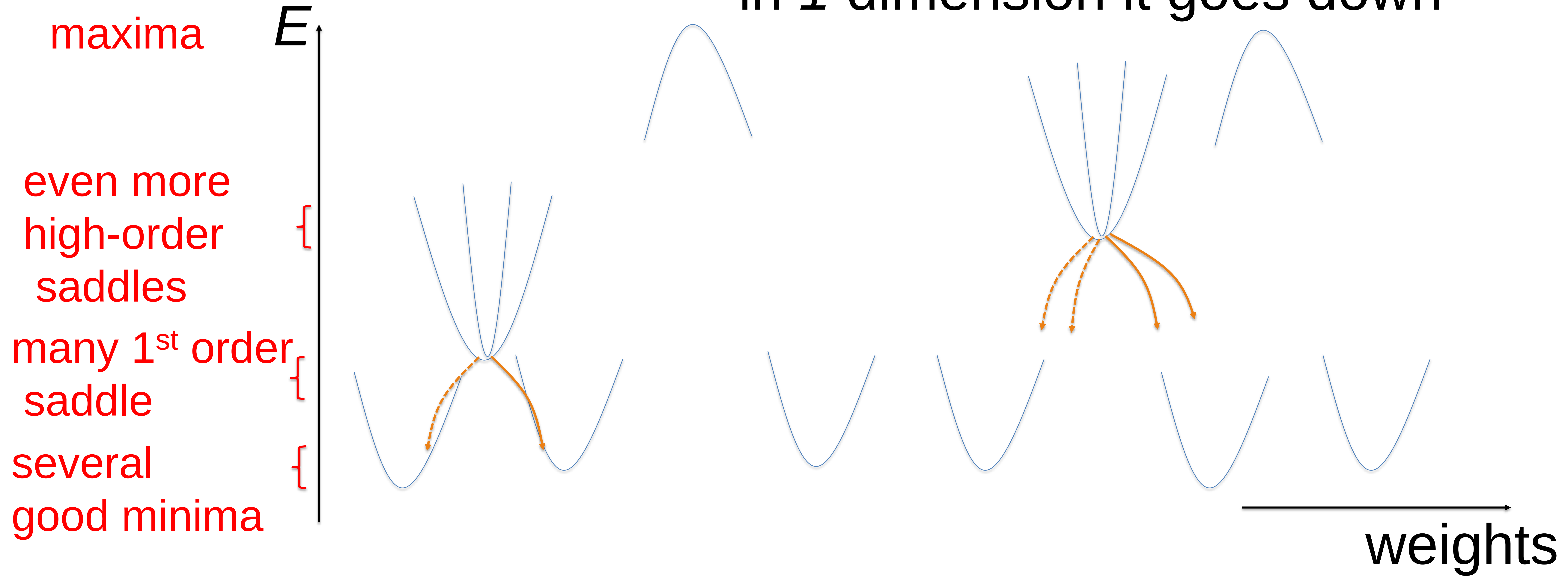
Previous slide.

The core of the argument is a statistical one. If you were to create Eigenvalues randomly with zero mean, then it would be very rare that all eigenvalues are positive. Most likely is a mix of positive and negative Eigenvalues. Therefore we expect to find more saddles than maxima or minima.

Minima and saddle points: modern statistical view

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

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



Previous slide.

Specific mathematical and physical models, linked to random matrix theory, Gaussian processes, and spin glasses, lead to a statistical picture where a few minima are at the lowest energies,
But most points with vanishing gradient are saddles of various order.

It is, however, not clear whether these models can be linked to deep neural networks because the specific weight space symmetries of deep network (e.g., permutation of neurons) are neglected.

Minima and saddle points

Review: Goodfellow et al., Deep Learning, 2016

(i) Statistical argument (Random Matrix Theory/Spin Glass)

For balanced random systems, eigenvalues will be randomly distributed with zero mean:
draw N random numbers (for N eigenvalues)

→ 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 dimensions of escape**

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

Previous slide.

For these random matrix or spin glass arguments, and similarly Gaussian process models, the question arises where the randomness stems from. The answer is that, when we design the neural network, we did not yet look at the data. Therefore, the data points can be considered as random constraints on the possible configuration of weights. This notion can be formalized but this is not the topic here. See Goodfellow et al. (2016) for references.

Minima and saddle points

Claim:

There are many more saddle points than minima

Two arguments

(i) Statistical argument

→ Eigenvalues of Hessian Matrix

(ii) Geometric argument

→ Permutations (between global minima)

Previous slide.

Remember that there are two different arguments.
So far we discussed the statistical argument. Let us now look at the second one.

Minima and saddle points

There are many more saddle points than minima

Second argument

(ii) Geometric counting argument using weight space symmetry

→ number of saddle points increases rapidly with number of parameters (even more rapidly than the number of equivalent global minima that arise from permutations)

Previous.

We focus on global minima to keep the argument simple. Permutation minima are connected with each other by saddles. We claim that there are many more saddles than global minima.

To connect the global minima with each other we imagine that we decrease the distance between two weight vector positions. Once the distance between two weight vectors is zero, I can remove one of them and shift its output weight to his partner. I can then turn it and make it identical to any other weight vector in the same layer, and exchange with that one, at no extra cost!

Thus the barrier of the saddle point between permutation minima is the lowest one of all possible pairs.

Error function: minima and saddle points

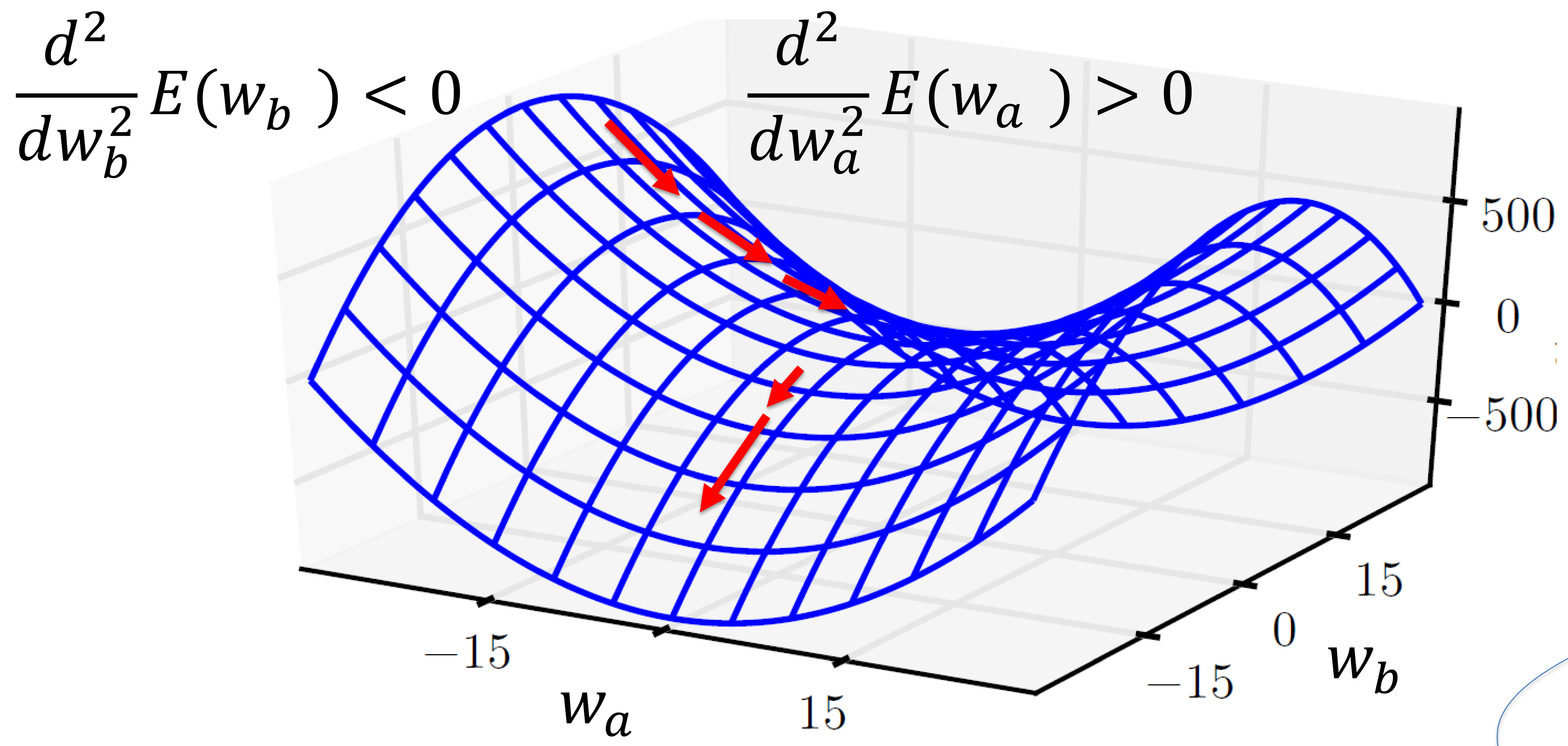
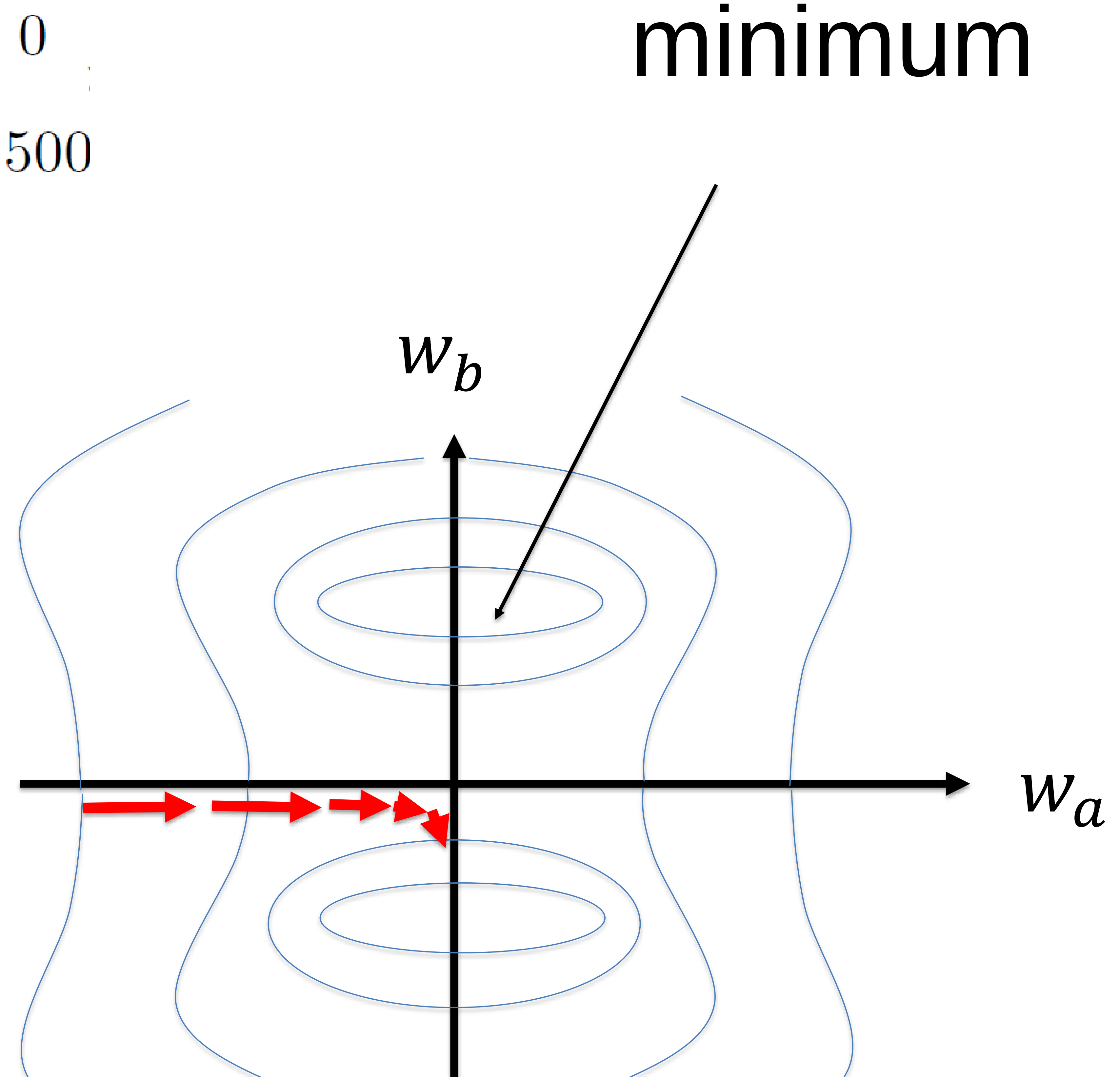


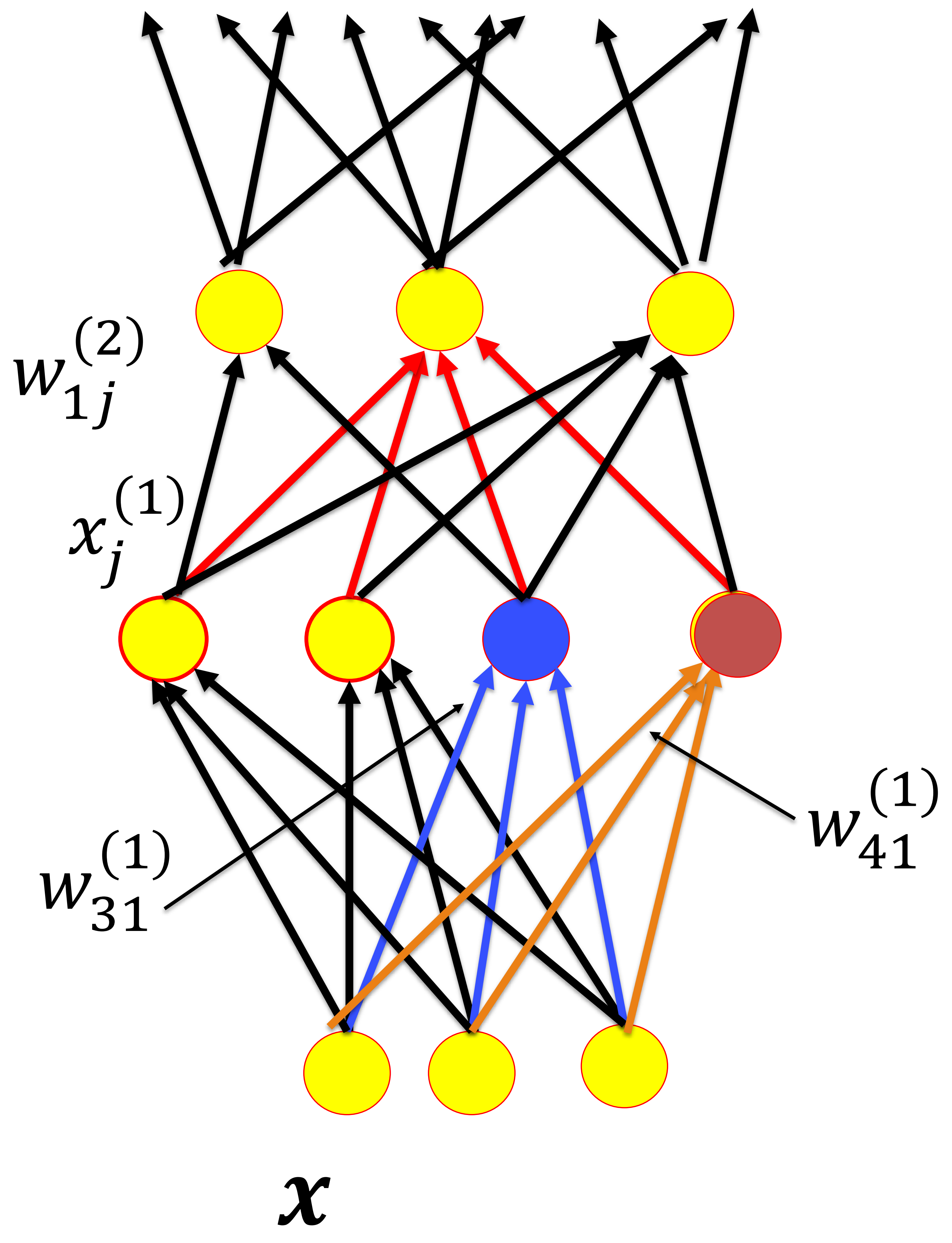
Image: Goodfellow et al. 2016



2 minima, separated by
1 saddle point

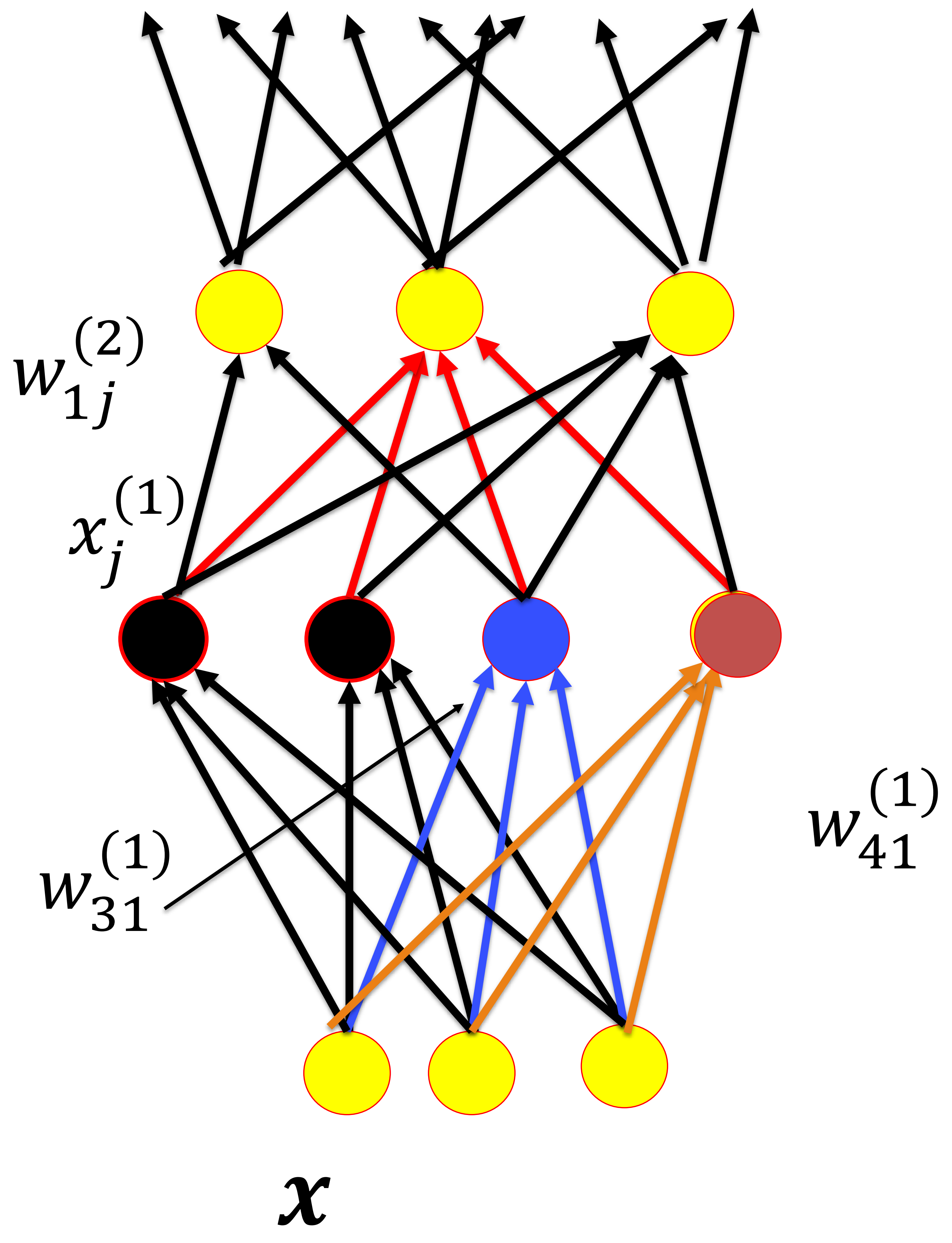
Your notes.
Just a reminder how a saddle separating two minima looks in two dimensions.
We now imagine that the two minima are global minima that represent different permutations of the hidden neurons.

Loss function and weight space symmetry



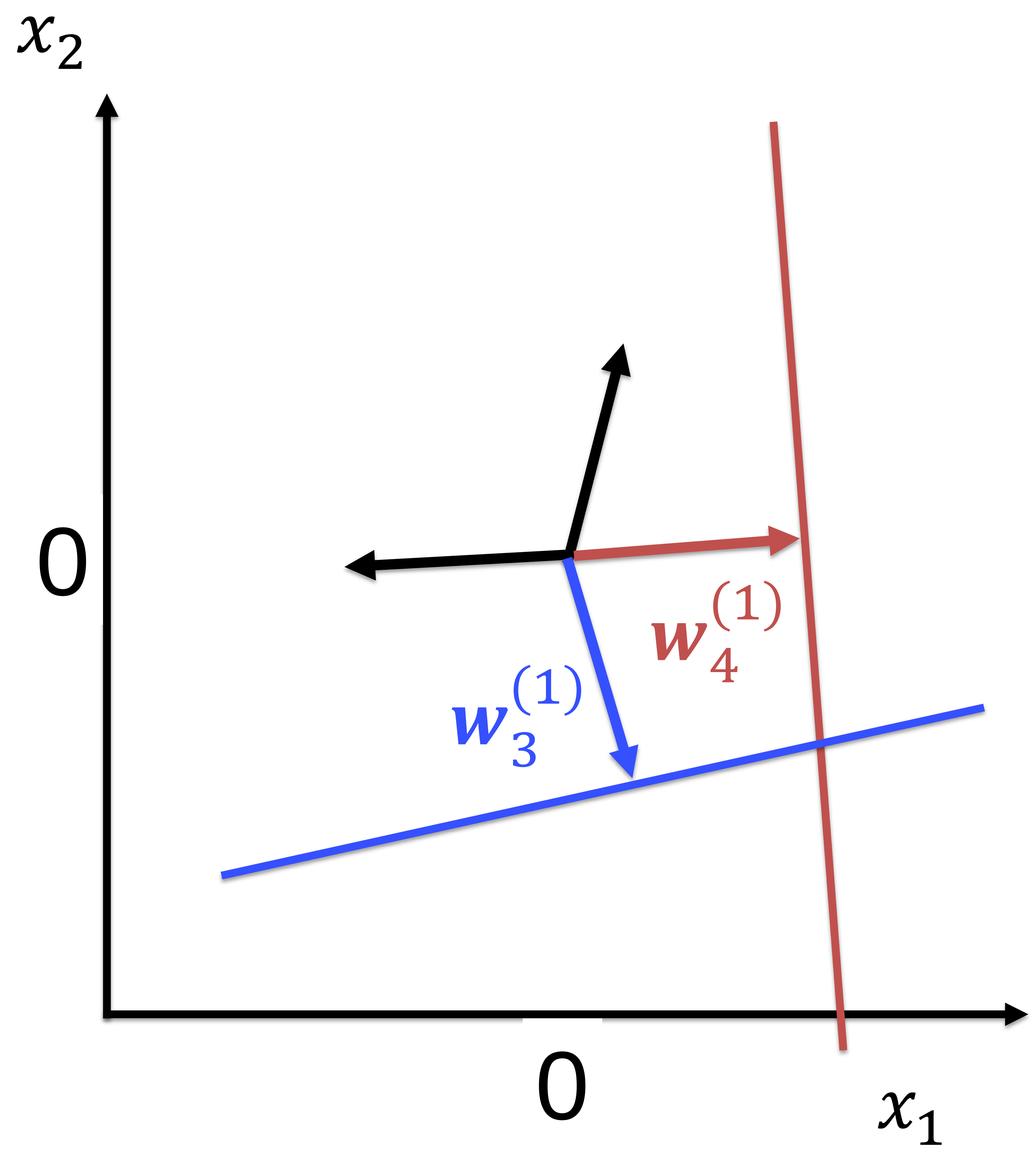
Solutions in weight space

Minima and saddle points: Example

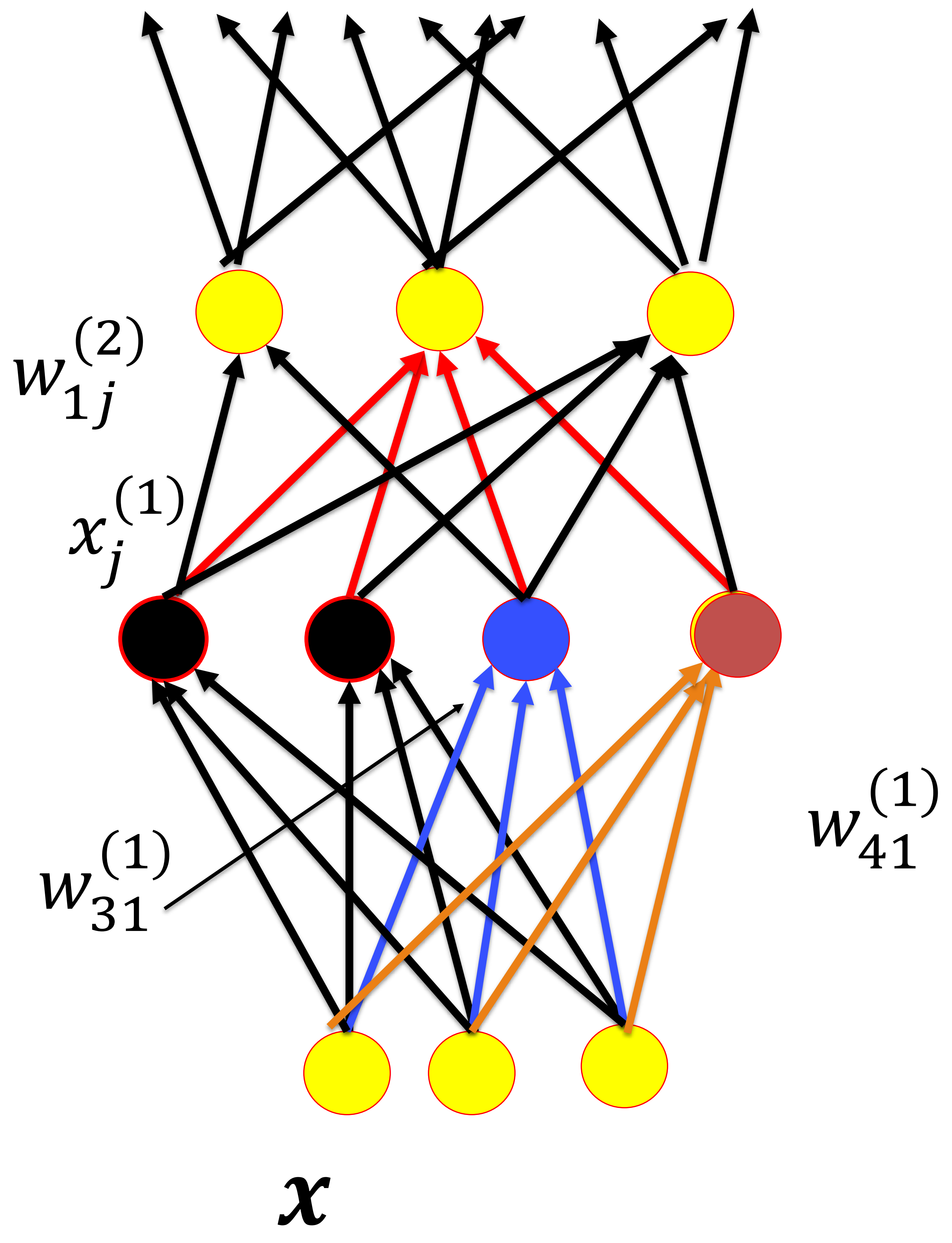


4 hyperplanes

'input space'

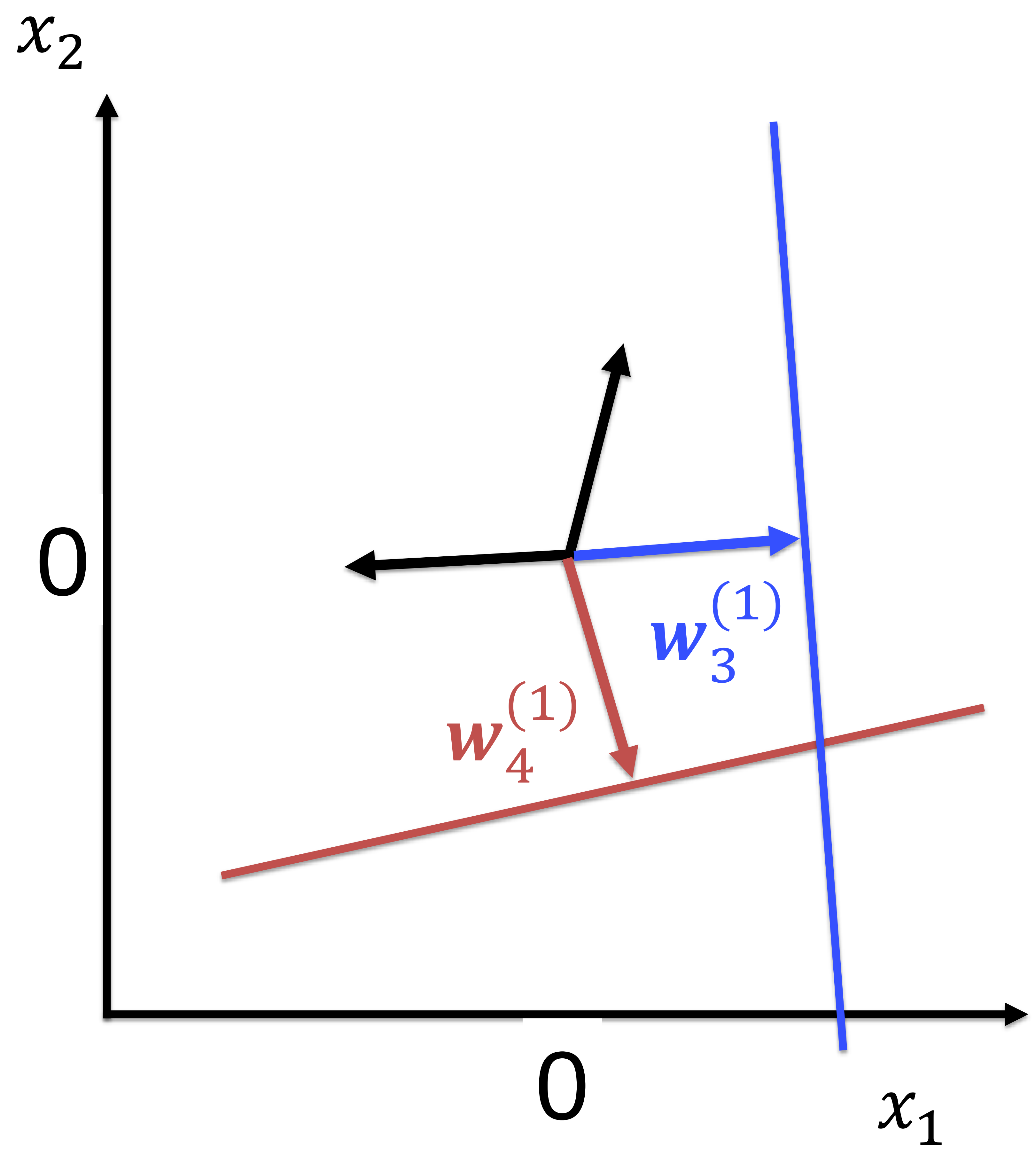


Minima and saddle points: Example after permutation



4 hyperplanes

'input space'



Previous three slides:
We have seen previously that if we found one global minimum there is a huge number of equivalent minima arising from permutations.

Concretely, in this example
permutations of neurons 3 and 4 in the first layer give exactly the same solution.

3 slides:
1. Brown neuron (i=4) and blue neuron (i=3
2. Hyperplanes of both
3. Now we want to exchange these neurons
    After Permutation

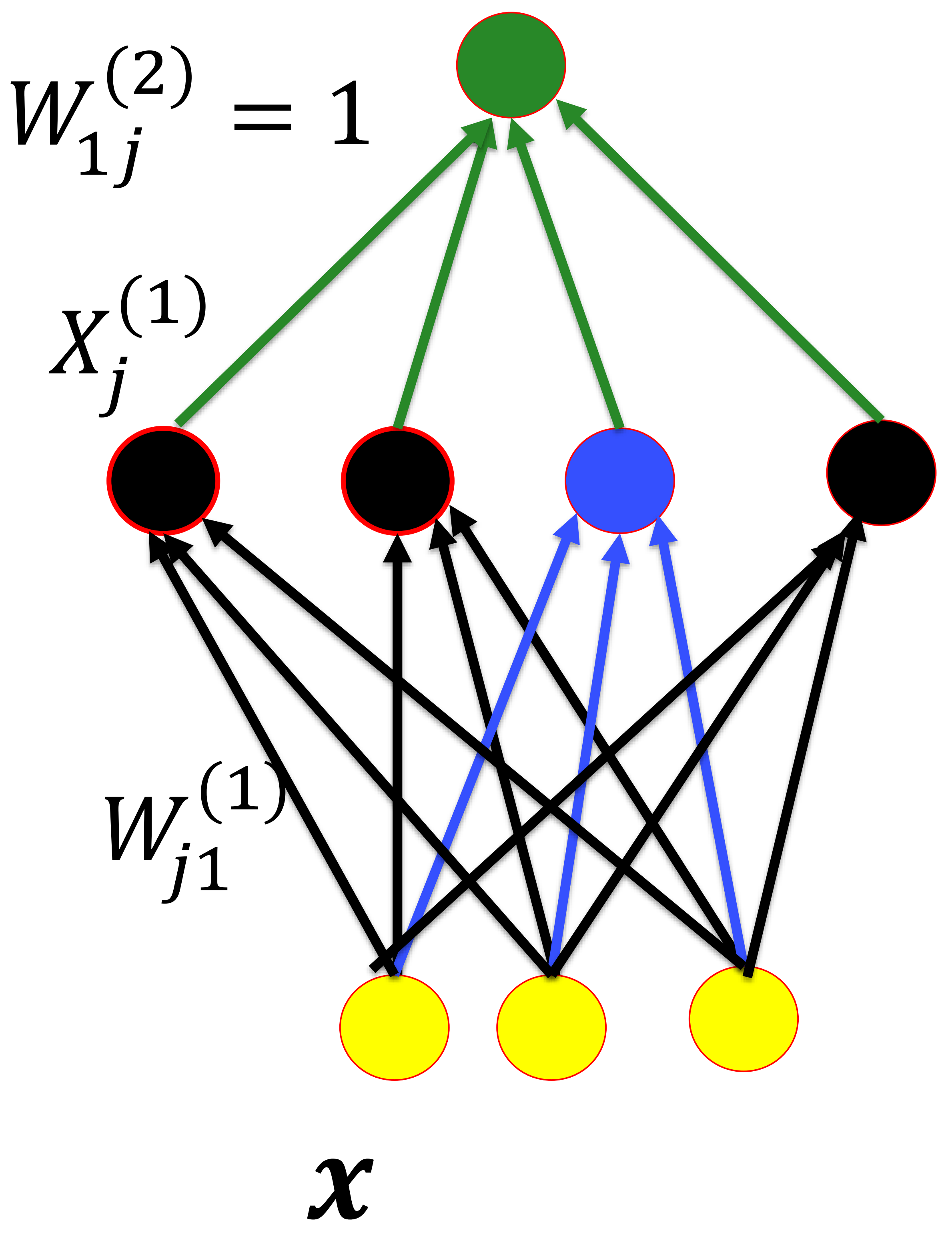
Minima and saddle points: Example

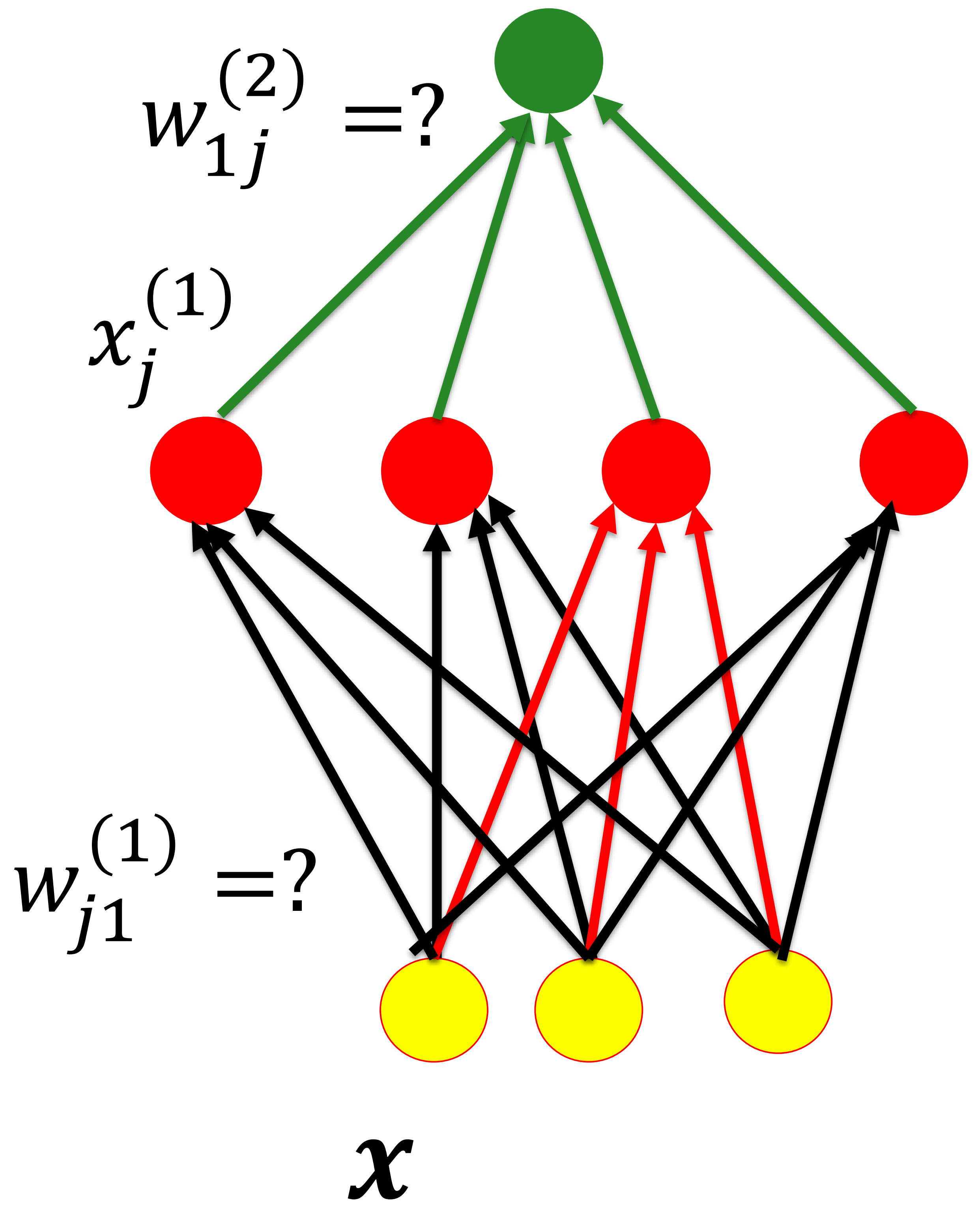
4 hyperplanes

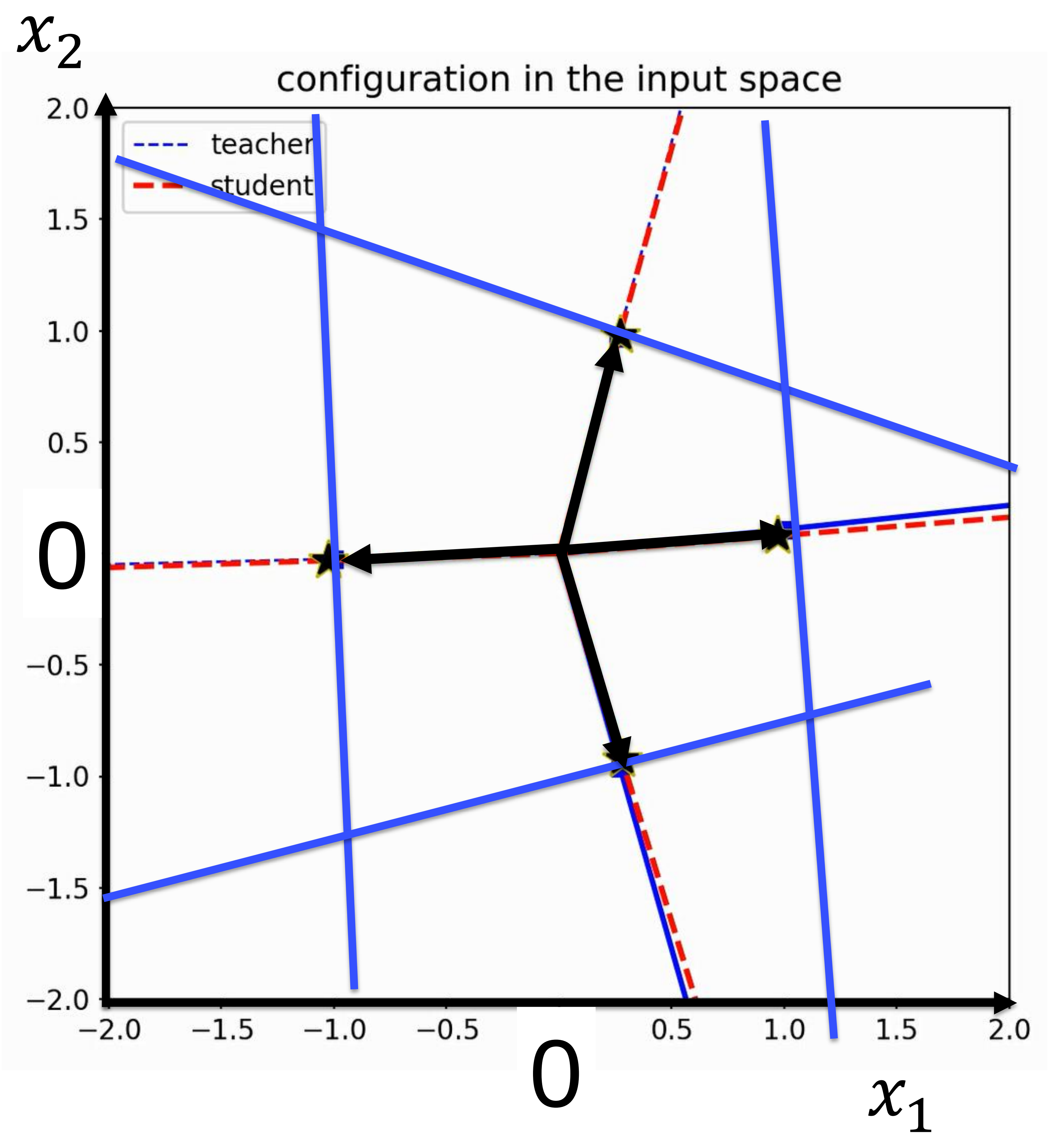
'input space'

Teacher Network:
Committee machine

Student Network:







Previous slide.

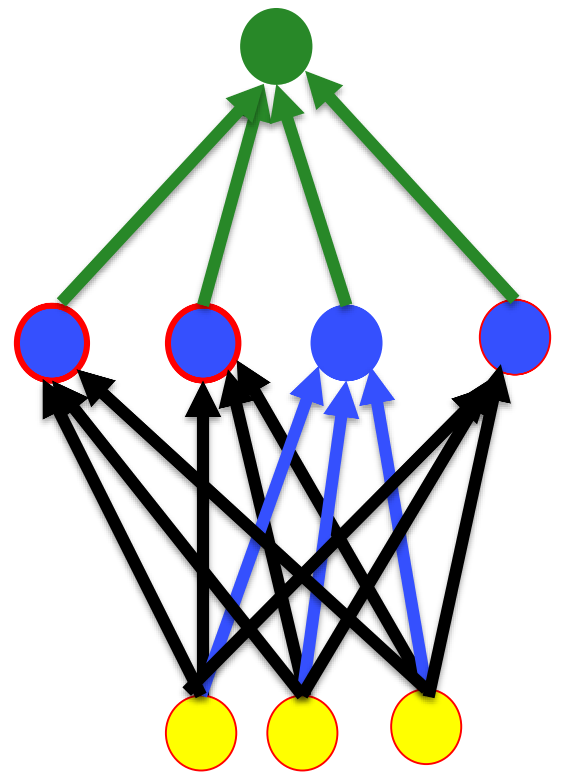
Data is generated from a teacher network (left).
Neurons in the first hidden layer implement hyperplanes (e.g., blue neuron).
The green neuron in the second layer sums up all contributions with equal weight. Such a configuration is called a committee machine (‘all votes count equally’).

The hyperplanes in input space are shown as blue lines on the right-hand side. They are characterized by their weight vectors (black). The end point of the weight vector indicates the location of the hyperplane.

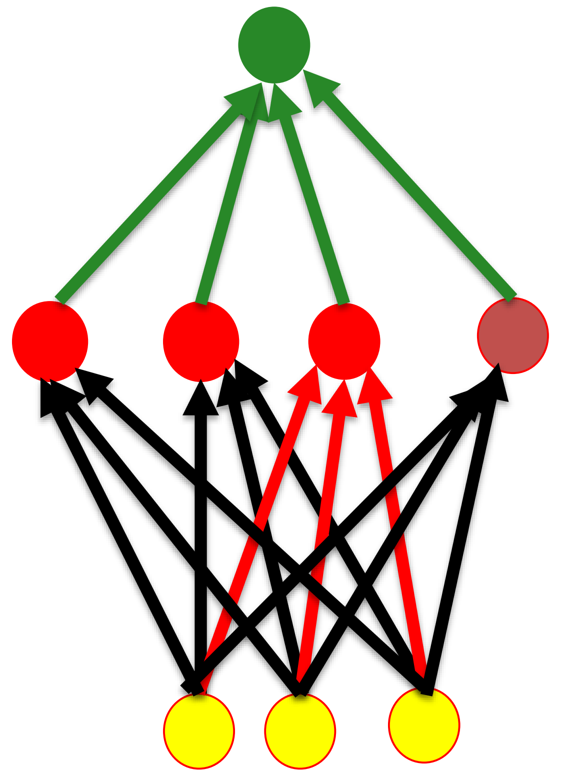
The student network has the same architecture, but freely adaptable weights in both layers.

Permutation Minima are connected by Saddle Points

Teacher Network: Blue

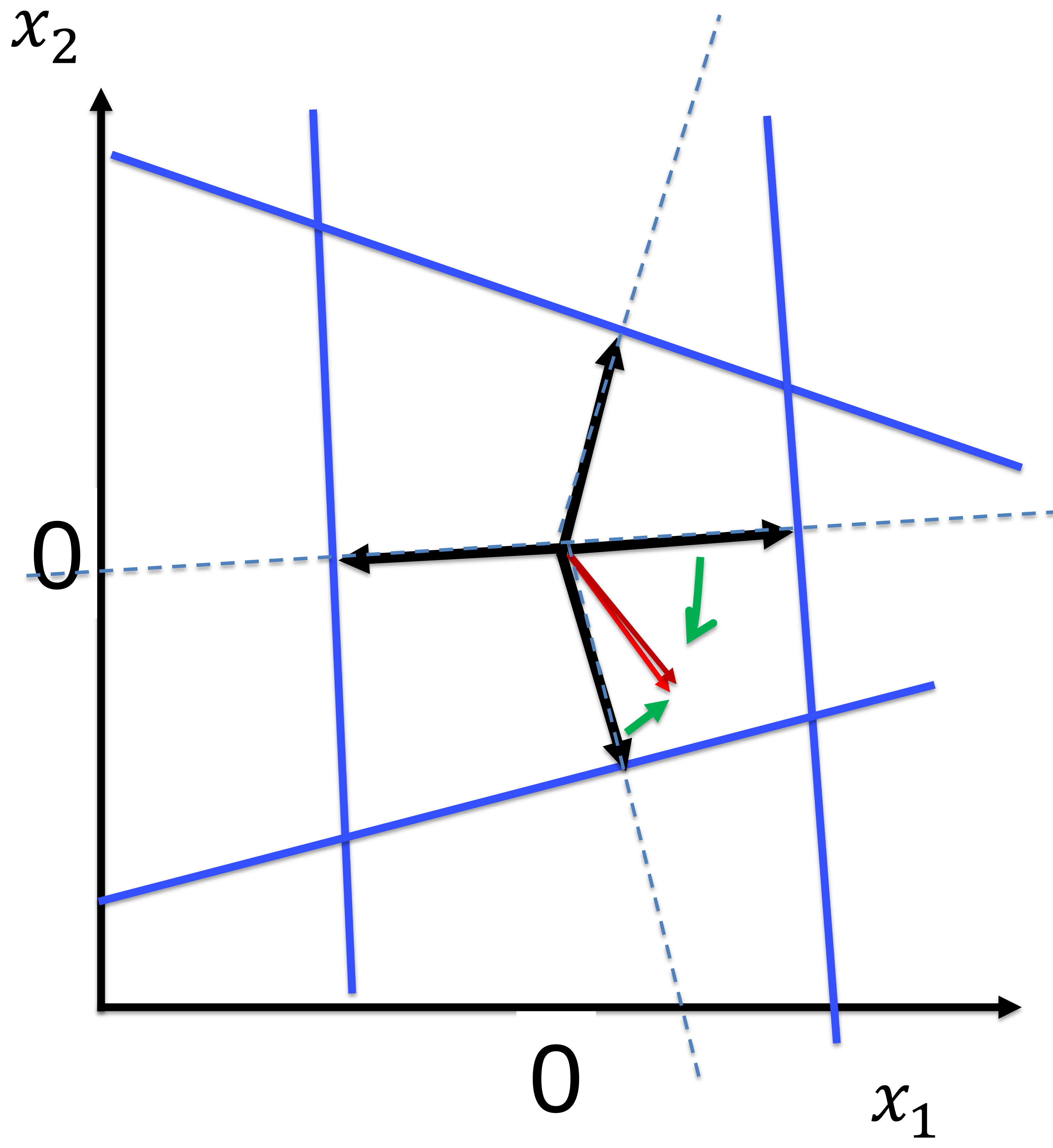


Student Network: Red



4 hyperplanes

'input space'

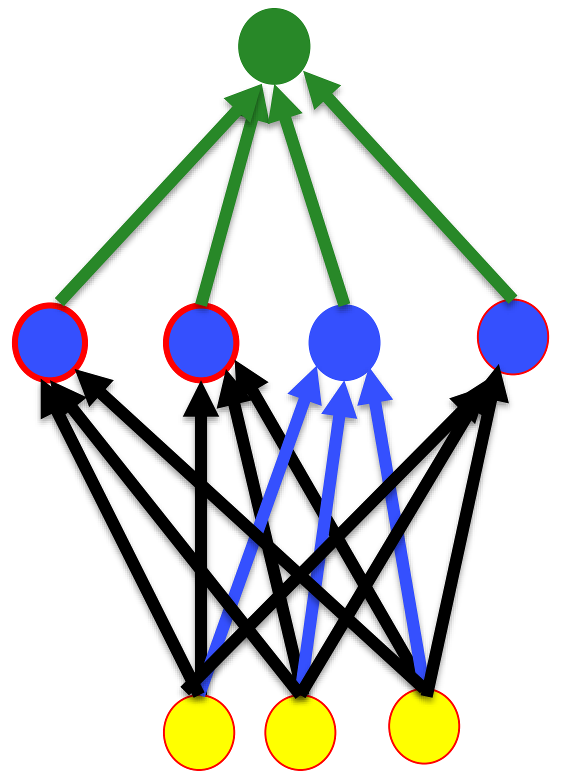


Approach

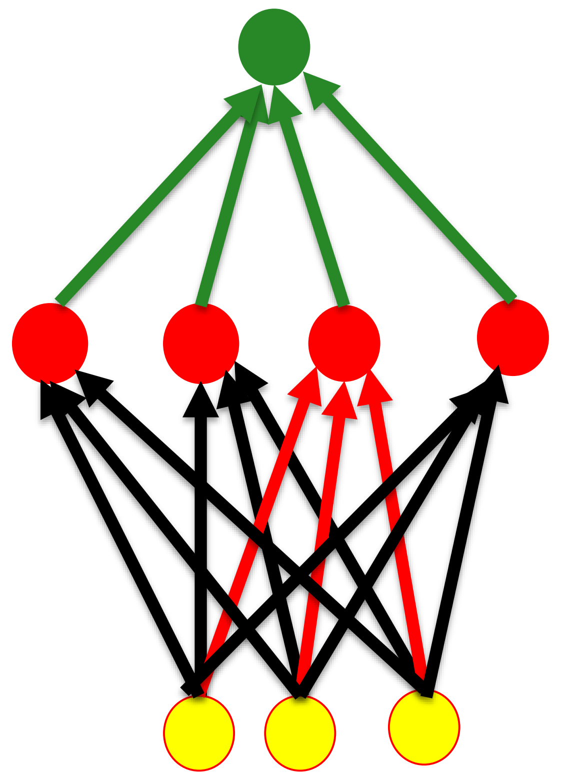
- Student starts aligned with teacher
- Slowly decrease distance between two weight vectors
- Let other weight vectors equilibrate to (nearby) minimum-loss configuration

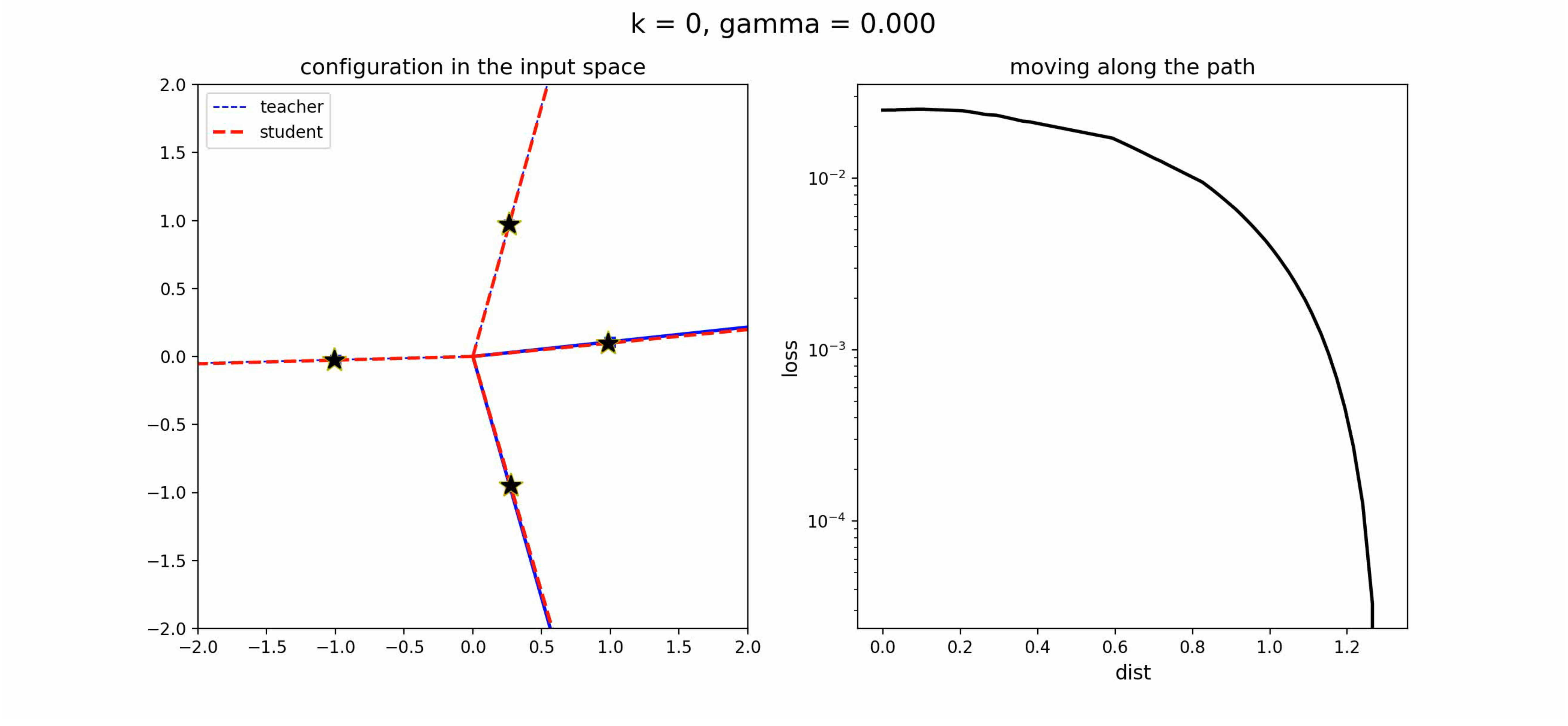
Permutation Minima are connected by Saddle Points

Teacher Network: Blue

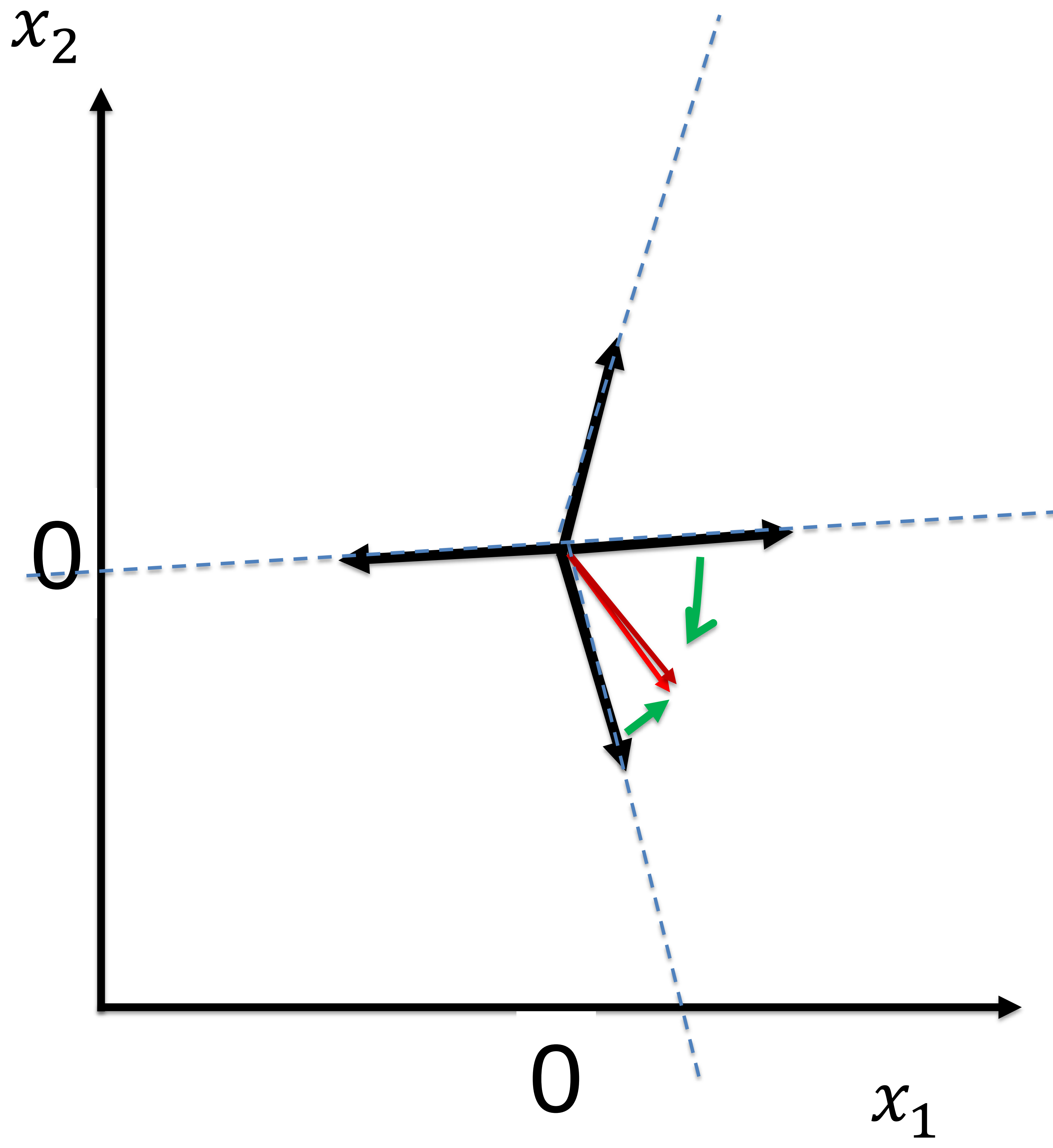


Student Network: Red



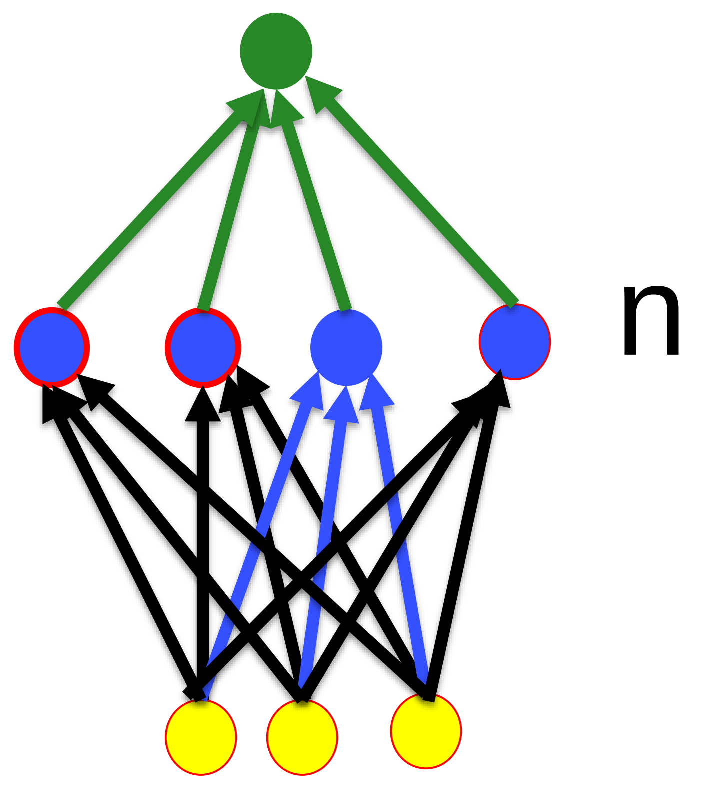


'blue=teacher'

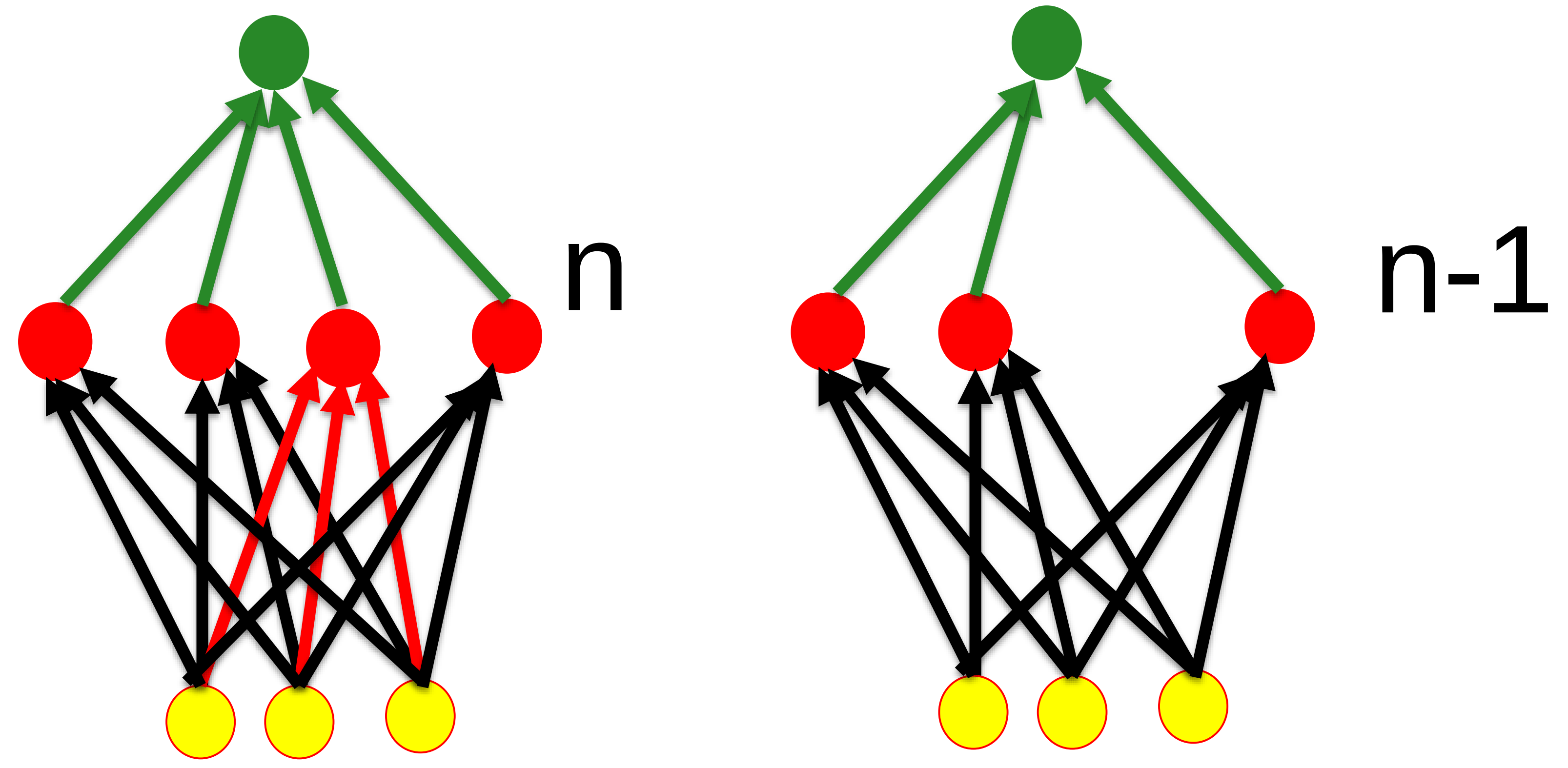


Between Permutation Minima: critical point of smaller network

Teacher
Network:
Blue



Student
Network:
Red



Teacher:

Data generated with
n hidden neurons

Student

→ zero loss possible for student with
n hidden neuron (same configuration)

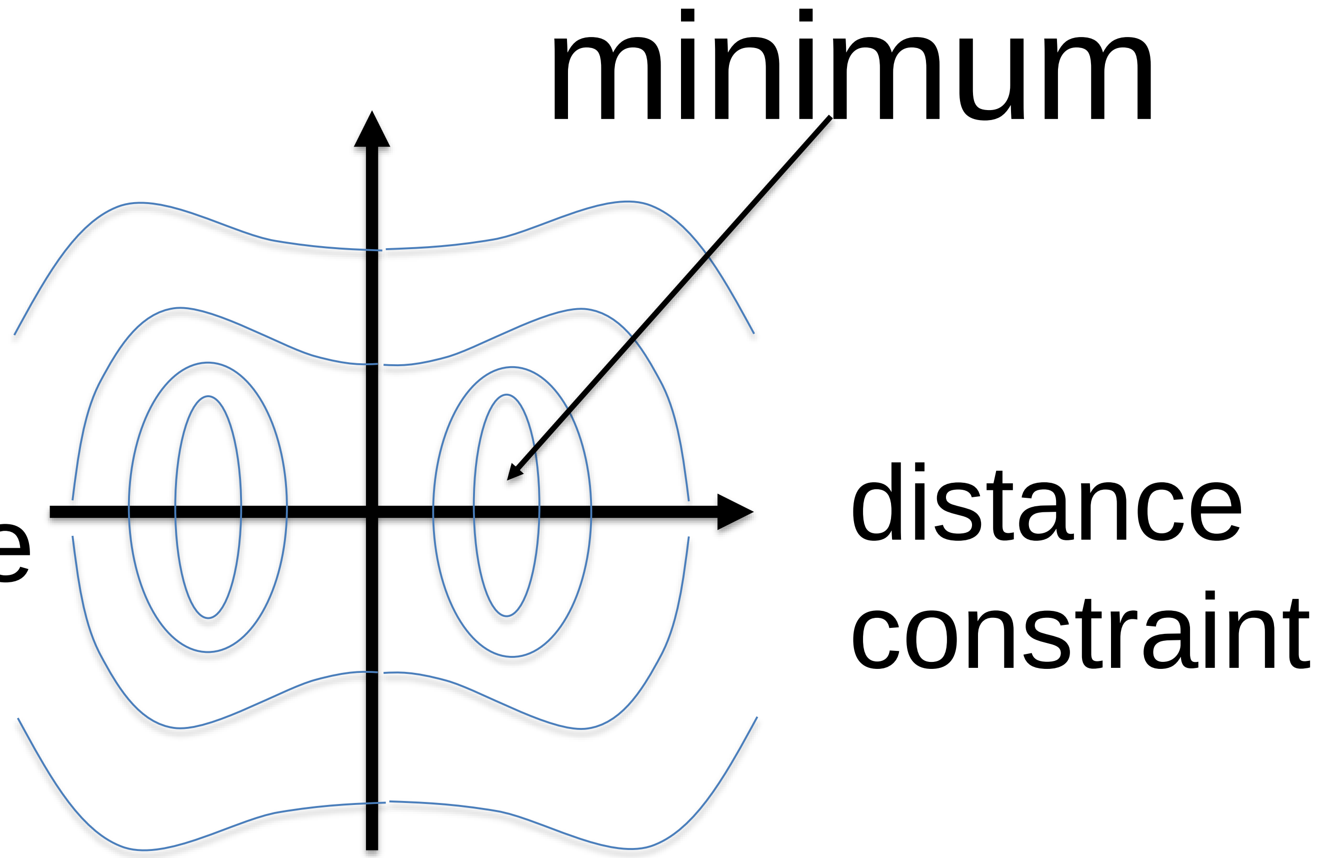
→ Nonzero loss for student with
n-1 hidden neurons.

→ Minima of student with n-1 hidden
neurons = critical point (saddle) of
student with n hidden neuron

Previous slide.
We want to explore the saddle between two equivalent permutation minima.
To do so, we initialize the student with weights perfectly aligned with those of the teacher. Then we force the student to have two weight vectors approach each other. All other weights remain free and are minimized (under the constraint that the two chosen weight vectors have a certain distance (dist, horizontal axis; loss, vertical axis).

As the distance is reduced from the initial configuration, the loss increases. When the distance is zero, the two weight vectors are identical (and implement the same hyperplane). At this moment, the configuration of the network is identical to a network with (n-1) instead of n hidden neurons. Note that the labels of the two vectors can be exchanged at this point for free – thereafter one could move back to the original configuration but with permuted weight vectors.

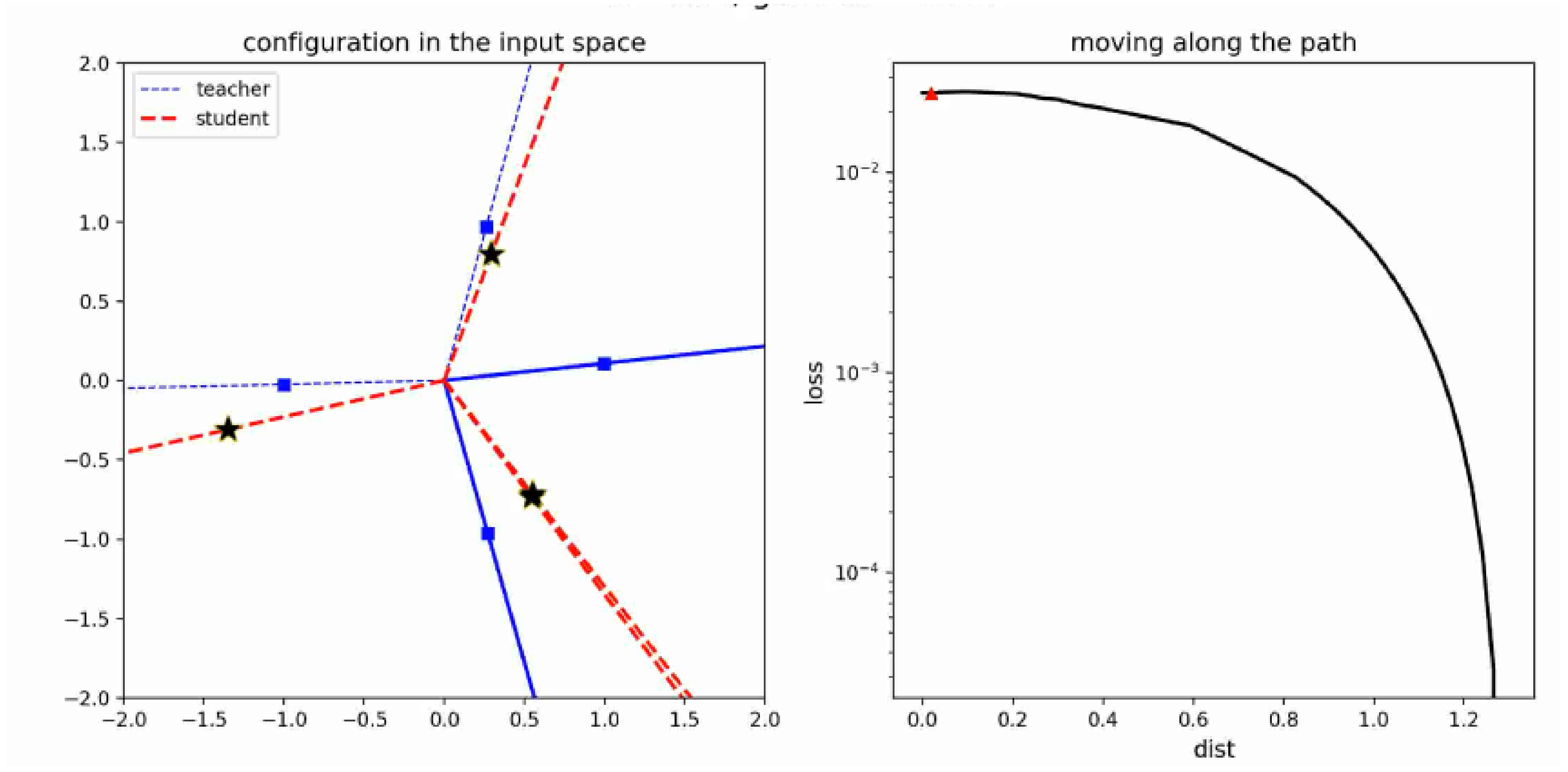
The point at dist=0 is a critical point because all other weights have been minimized. Moreover, it can be a saddle because when going from n-1 to n hidden neurons, we have More parameters and can go down further.



Permutation Minima are connected by Saddle Points

- Slowly decrease distance between two weight vectors
- Let other weight vectors equilibrate to (nearby) minimum-loss configuration.
- Resulting point is a minimum of a smaller network with n-1 neurons
- Every minimum of a network with n-1 neurons is a critical point of the network with n neurons.

Simsek et al.,
Geometry of the loss landscape
ICML 2021
PMLR 139:9722-9732, 2021.



Previous slide.

Two ways of thinking:

(i) We start at the minimum of a smaller network with n-1 neurons. We add one extra neuron and search for the new minimum of the network with n neurons. This minimum will be lower. The initial point (minimum of network with n-1 neurons) is a critical point (gradient-zero point) of the larger network.

(ii) We start from the global minimum of a network with n neurons. We slowly squeeze two weight vectors together and let the other equilibrate. This is a minimum of the smaller network with n-1 neurons. At this point we can exchange the two indices of the two neurons and go back the same way, but with permuted indices. Obviously, either the point of collapse of the two weight vector is a saddle point or we must pass through another saddle point on the way to and back. We go back on the same path, but after the change of the indices.

Minima and saddle points: Geometric argument

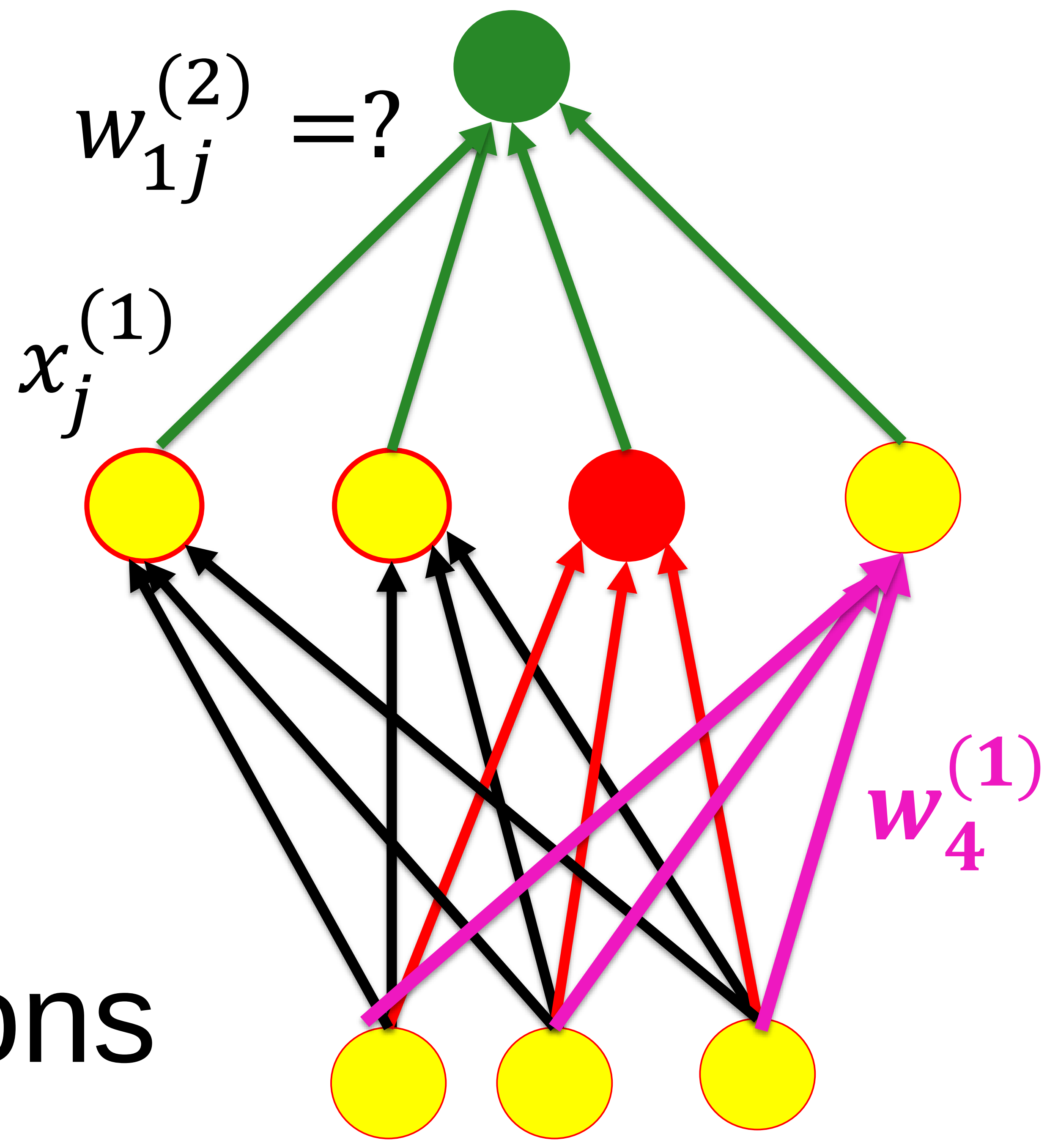
There are many more saddle points than global minima

Geometric argument exploiting the weight space symmetry

→ count number of symmetry-induced saddle points

→ lower bound: **permutation points**

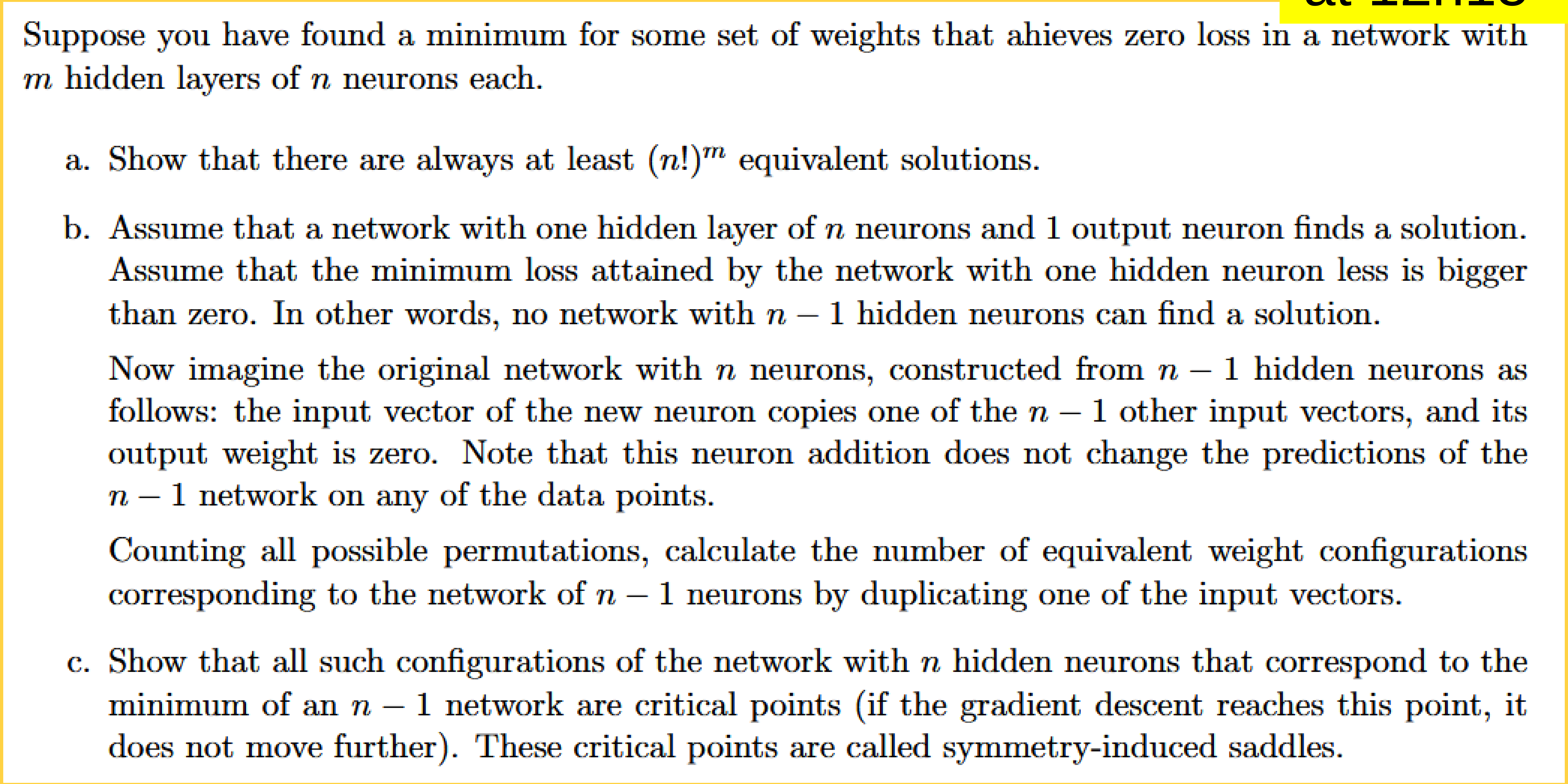
For n=4: map 4 vectors onto 3 positions (each position taken at least once, exactly one twice)



Your notes.
Now we start with the counting argument. 4 hidden neurons (student with n neurons) give 4 weight indices, that we have to place on three vector positions (the minimum of the student with n-1 neurons).

Exercise: Weight space symmetries

Next Lecture at 12h15



Suppose you have found a minimum for some set of weights that ahieves zero loss in a network with m hidden layers of n neurons each.

a. Show that there are always at least $(n!)^m$ equivalent solutions.

b. Assume that a network with one hidden layer of n neurons and 1 output neuron finds a solution. Assume that the minimum loss attained by the network with one hidden neuron less is bigger than zero. In other words, no network with $n-1$ hidden neurons can find a solution.

Now imagine the original network with n neurons, constructed from $n-1$ hidden neurons as follows: the input vector of the new neuron copies one of the $n-1$ other input vectors, and its output weight is zero. Note that this neuron addition does not change the predictions of the $n-1$ network on any of the data points.

Counting all possible permutations, calculate the number of equivalent weight configurations corresponding to the network of $n-1$ neurons by duplicating one of the input vectors.

c. Show that all such configurations of the network with n hidden neurons that correspond to the minimum of an $n-1$ network are critical points (if the gradient descent reaches this point, it does not move further). These critical points are called symmetry-induced saddles.

Your notes.

b1) Suppose you have zero loss for a network with n hidden neurons, but nonzero loss for a network with n-1 hidden neurons. How many permutations are there to assign indices of n neurons to the n-1 INPUT weight vectors of the smaller network?

b2) What can we say about the output weights? So far it one of the weights of the duplicate neuron was zero, and the other had a value of w*, so a tuple (0,w*). Convince yourself that any combination (a,w*-a) would give exactly the same output. Hence, we should not talk about separate critical POINTS but about a LINE OF CRITICAL points, i.e., a 1-dimensional manifold.

c) Show that each configuration with (0,w*) corresponds to a critical point of the network with n neurons (assume that the output weight of one of the duplicated neurons vanishes at the initial condition)

Minima and saddle points: weight space symmetry/geometry

Assumption: Data explained by a network with n hidden neurons, but a network with n-1 neurons has higher loss.

Claim: The loss landscape of network with n hidden neurons has more first-order critical 'points' than global minima

→ count permutation points/global minima of smaller network

Layer with n hidden neurons:

→ n vector indices for (n-1) positions

→ lower bound for first-order saddles

→ Important we count LINES (1-d manifolds), and not points

Previous slide.

For first-order symmetry-induced critical points ('permutation points'), we have to place n vector indices on the n-1 locations that define the configuration with (n-1) neurons in the hidden layer at one of the minima of the smaller network.

We do this placement in the following sequence: :
(i) We have 1 special position with 2 weight vectors and n-2 with one weight vector each. To select the special position that has double weights, we have n-1 possibilities.

(ii) Once we have chosen our special location, we have n!/2! possibilities to distribute the weight indices. The factor 2! arises because it does not matter in which order the two indices are assigned into the special position. The order of the indices in the duplicate weight vector does not matter because we count the number of LINES with output weights (a,w*-a). There are 2 points (0,w*) and (w*,0) but they are connected by 1 line – and hence only counted once.

Note that there might be even MORE first-order symmetry-induced critical points, because depending on which weight vectors we merge, different configurations (=positions of n-1 weight vectors) arise. These configuations will in general not all have the same loss. Some are global, others are local minima of the smaller networks

Saddle points: Geometric argument and weight space symmetry

There are many more saddle points than minima

Lower bound for first-order saddles

→ count permutation points in layer with n hidden neurons

→ n vectors for (n-1) specific positions

→ $\frac{n!}{2!}(n-1)$ lines (1d-manifolds) of first-order critical points

Lower bound for second-order saddles

→ n vectors for (n-2) specific positions

→ $\frac{n!}{3!}\binom{n-2}{1} + \frac{n!}{2!2!}\binom{n-2}{2}$ 2d-manifolds of second-order critical points

Previous slide.
For second-order symmetry-induced critical points, we have to place n vector indices on n-2 positions.
Suppose n=4. We have two overall possibilities:

(i) We have 1 special position with 3 weight vectors and n-3 with one weight vector each. To select the special position that has triple weights, we have n-2 possibilities. Once we have chosen our special location, we have n!/3! possibilities to distribute the weight indices

(ii) We have 2 special positions with 2 weight vectors each and n-4 with one weight vector each. To select the special positions with double weights, we have n(n-1)/2 possibilities. Once we have chosen our special locations we have n!/4 possibilities to distribute the weight indices.

Note that the term in paragraph (ii) dominates largely for large n. We use this observation for an overall lower bound on the next slide.

First-order symmetry induced critical points lie on a 1-dim manifold, whereas second-order symmetry induced critical points lie on a 2-dim manifold (e.g., for a triplet, the output weight condition is that (a,b,w*-a-b) which has two free variables).
Hence, the Hessian of first-order symmetry induced critical points has one zero Eigenvalue and the Hessian of second-order symmetry induced critical points has two zero Eigenvalues!

Saddle points: Geometric argument and weight space symmetry

There are many more saddle points than minima

→ number of saddle point increases rapidly with number n of neurons in hidden layer (much more rapidly than the number of minima)

Theorem: a layer with n neurons generates at least a factor of

$$\frac{1}{2^K}\binom{n-K}{K}$$

more K_{th} - order saddles $(K<n/2)$ than global permutation minima.

Your notes.
Here comes a more precise formulation of the theorem.
The assumption is as before: **:**
Data can be explained by a network with n hidden neurons, but a network with n-1 neurons has higher loss.
Then

Theorem: a layer with n neurons generates at least a factor of

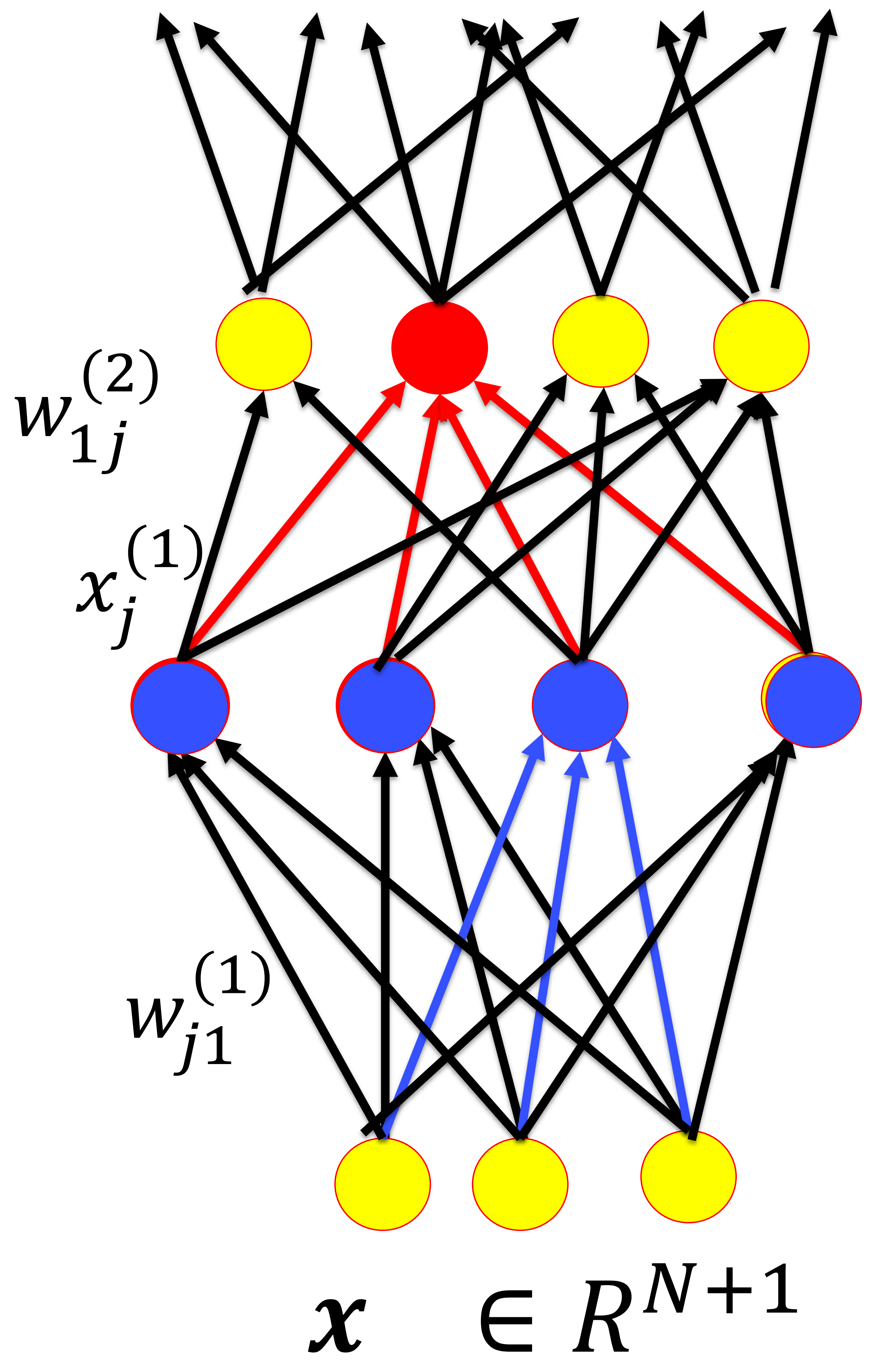
$$\frac{1}{2^K}\binom{n-K}{K}$$

more K_{th}- order saddles $(K<n/2)$ than global permutation minima.

The assumption is clearly satisfied if the data is generated from a teacher network which does not contain any redundant neuron. But this is not the only possible case.

Note that the count does not concern the number of points but the number of manifolds!

Summary: loss landscape in a deep neural network



In a network with m hidden layers and n neurons per hidden layer.
We have found one global minimum.

Then there are at least $\boldsymbol{n}!^{\boldsymbol{m}}$ minima with the same loss

and at least

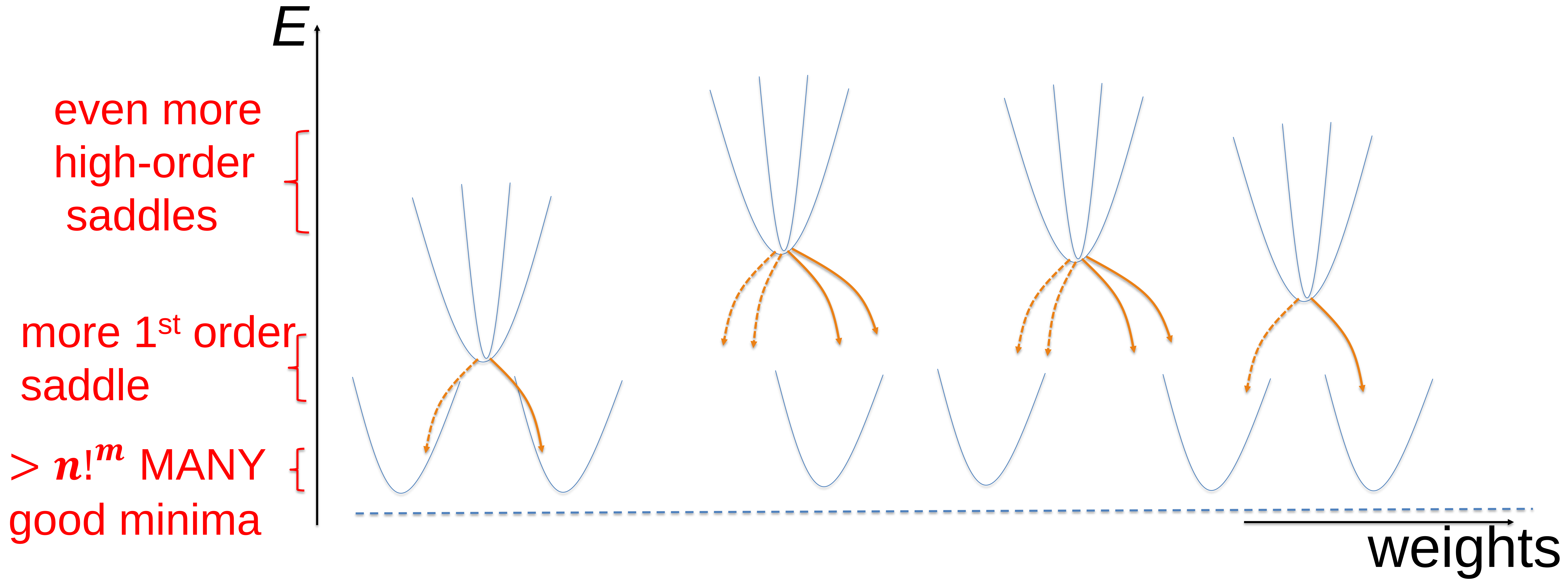
$$\frac{m}{2^K}\binom{n-K}{K}\,\boldsymbol{n}!^{\boldsymbol{m}}$$

K_{th}-order saddle point manifolds (K<n/2) of dimension K.

J. Brea et al. (2019), Weight space symmetry …
https://arxiv.org/pdf/1907.02911.pdf

Minima and saddle points: modern neural network view

Neural network with m hidden layers and n neurons per hidden layer.
Input dimension also n. Output dimension q.
Then dimensionality of weight space: $N = m \cdot n^2 + n \cdot q$


E
even more high-order saddles
more 1st order saddle
$> \boldsymbol{n}!^{\boldsymbol{m}}$ MANY good minima
weights

Quiz: Saddle points

A deep neural network with many neurons

[ ] has a huge number of equivalent minima and even many more saddle points

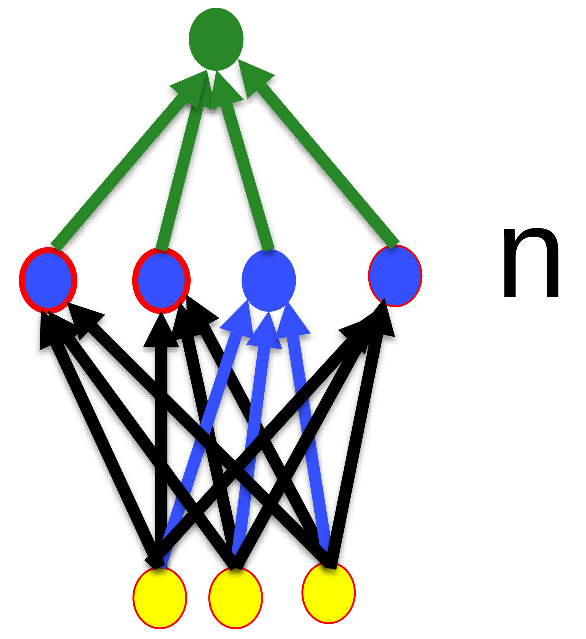
[ ] gradient descent is slow close to a saddle point

[ ] close to a saddle point there is only one dimension to go down

Previous slide.

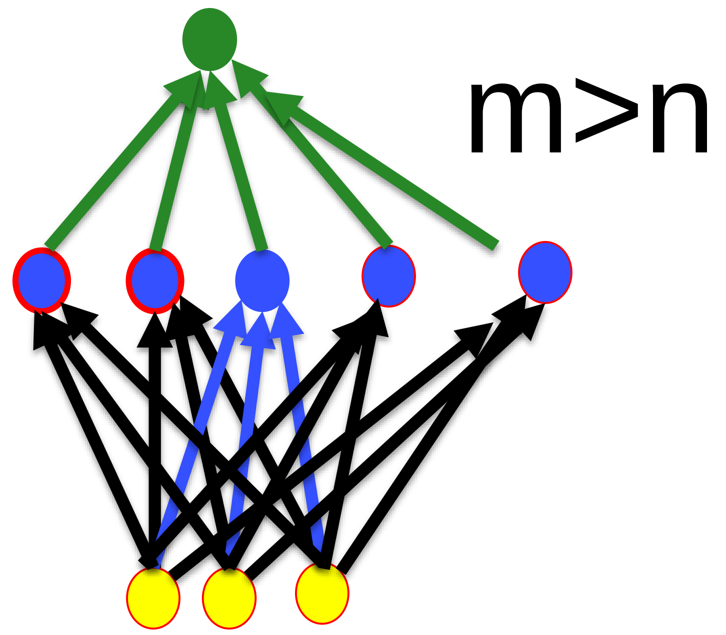
Global minima and saddle points and overparameterization

Teacher Network:



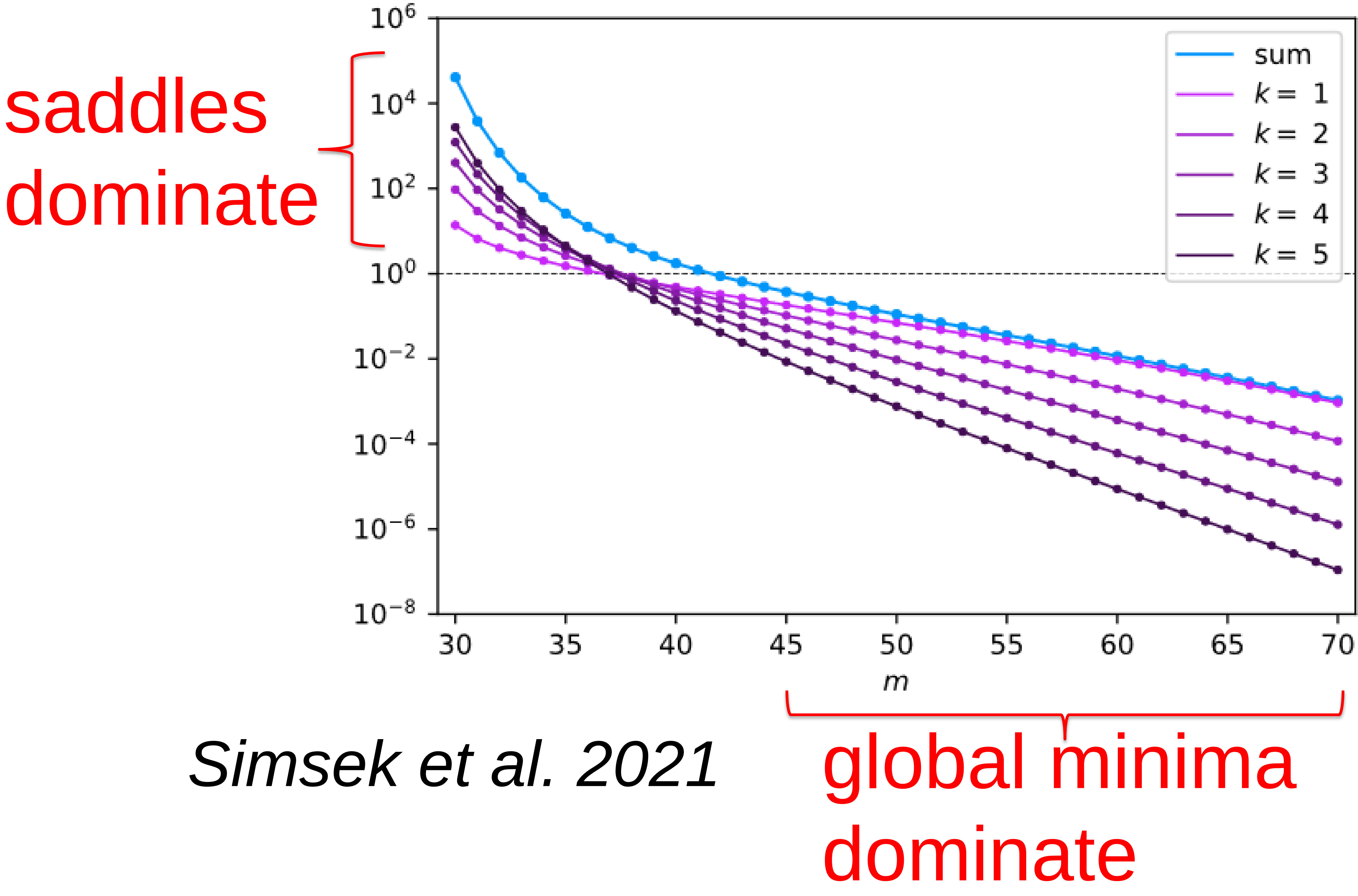
Teacher:
Data generated with n hidden neurons

Student Network:



Student has m=n+k neurons (overparameterized)

Ratio: $\frac{\text{\#saddle points order k}}{\text{\#global minima for n=30}}$



Simsek et al. 2021

Previous slide.

Artificial Neural Networks

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Loss landscape and optimization methods for deep learning

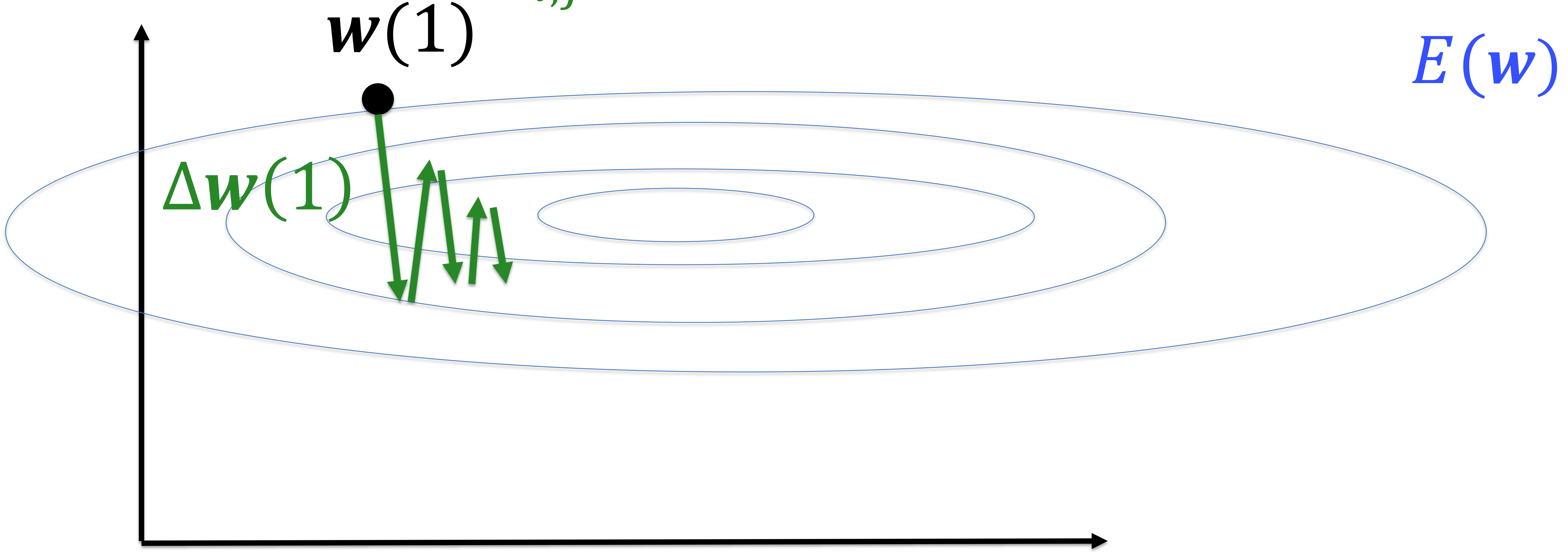
Part 4: Gradient Descent with Momentum

Previous slide.

The next question is: how do we find the minima?

Review: Standard gradient descent:

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



Previous slide.

The contour lines (niveau lines) of the error function $E(\boldsymbol{w})$ are shown as a function of two arbitrarily chosen weights. Gradient descent corresponds (with standard Euclidian metrics) to a movement downward perpendicular to the niveau lines, starting from the weight vector $\boldsymbol{w}(1)$ at time t=1

If the step size (learning rate γ) is too large, the movement shows oscillations.

Momentum: keep previous information

In first time step: *m=1*

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

In later time step: *m*

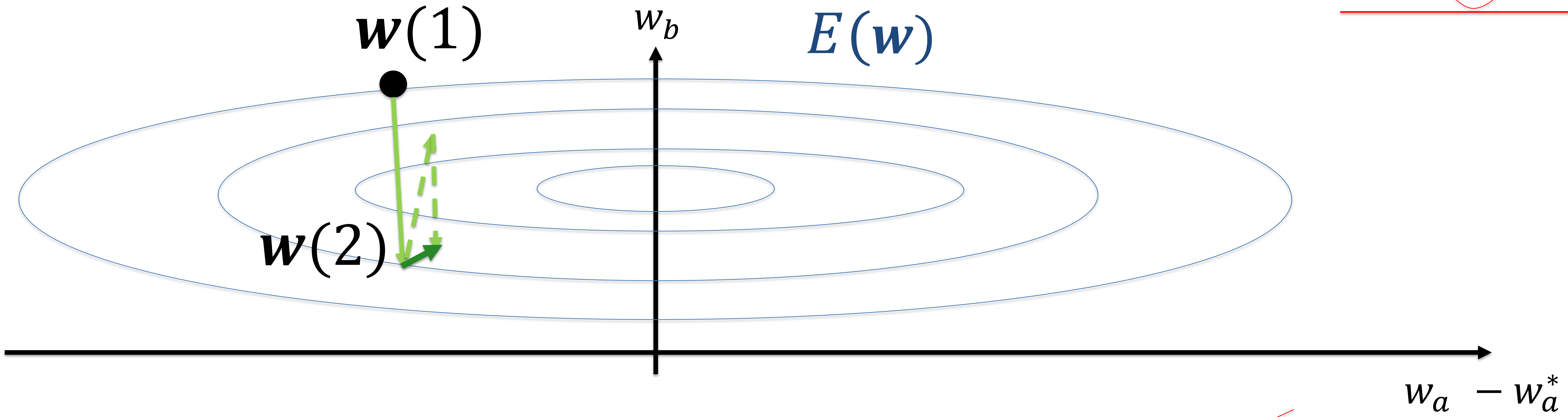
$$\Delta w_{i,j}^{(n)}(m) = -\gamma \frac{dE(\boldsymbol{w}(m))}{dw_{i,j}^{(n)}} + \alpha \; \Delta w_{i,j}^{(n)}(m-1)$$

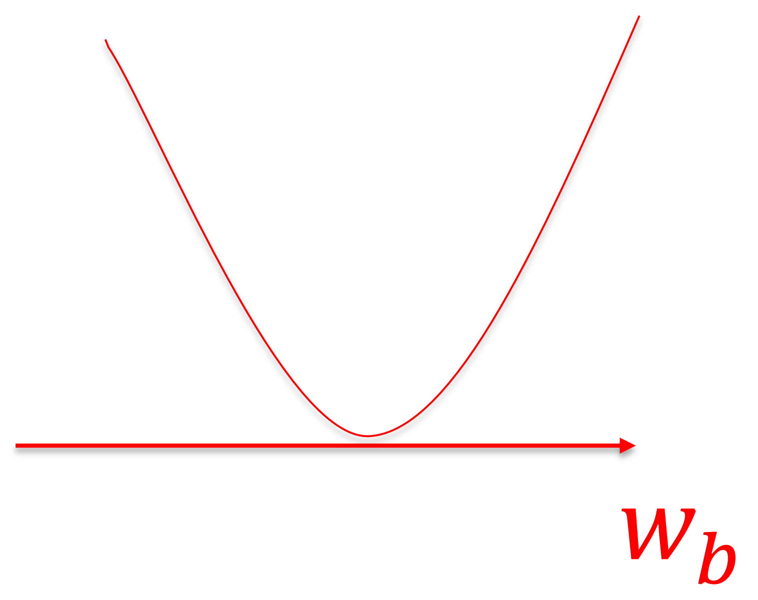
Previous slide.

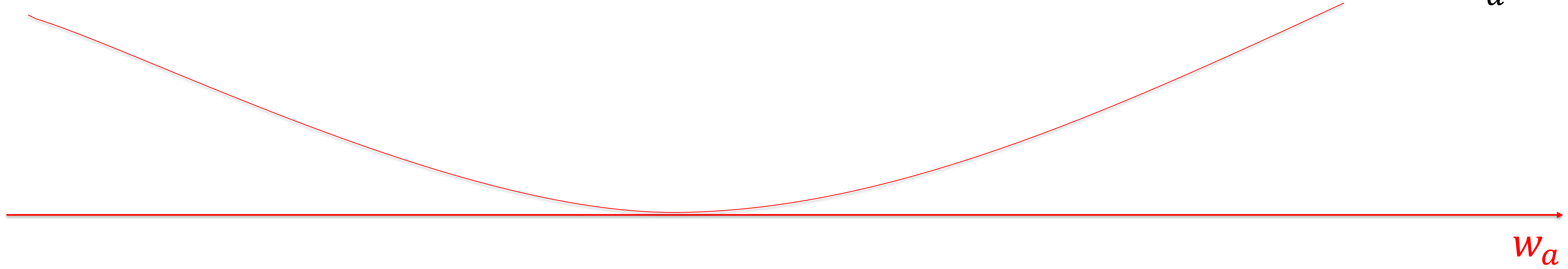
A momentum term
keeps information about the previous direction.
It suppresses therefore these oscillations while giving rise to a ‘speed-up’ in the directions where the gradient does not change

gradient descent with momentum

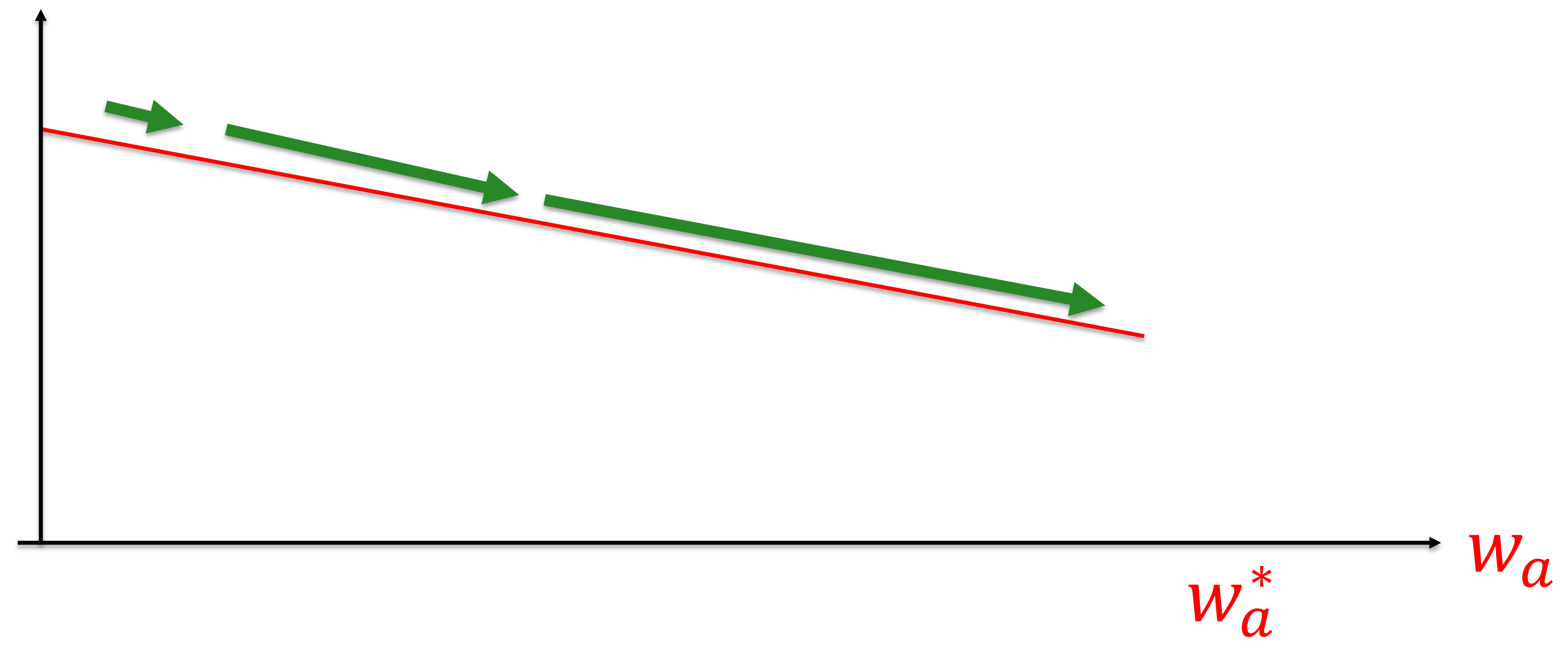
$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\boldsymbol{w}(1))}{dw_{i,j}^{(n)}} \qquad \Delta w_{i,j}^{(n)}(m) = -\gamma \frac{dE(\boldsymbol{w}(m))}{dw_{i,j}^{(n)}} + \alpha \ \Delta w_{i,j}^{(n)}(m-1)$$





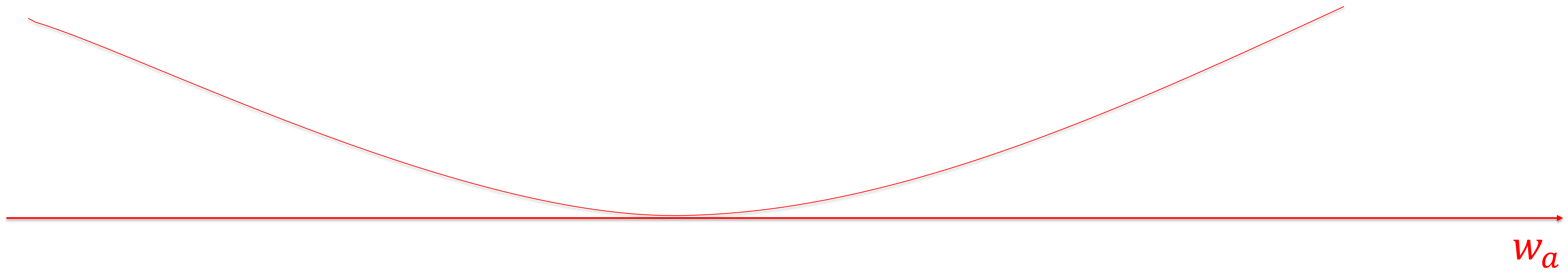


gradient descent with momentum (constant slope)

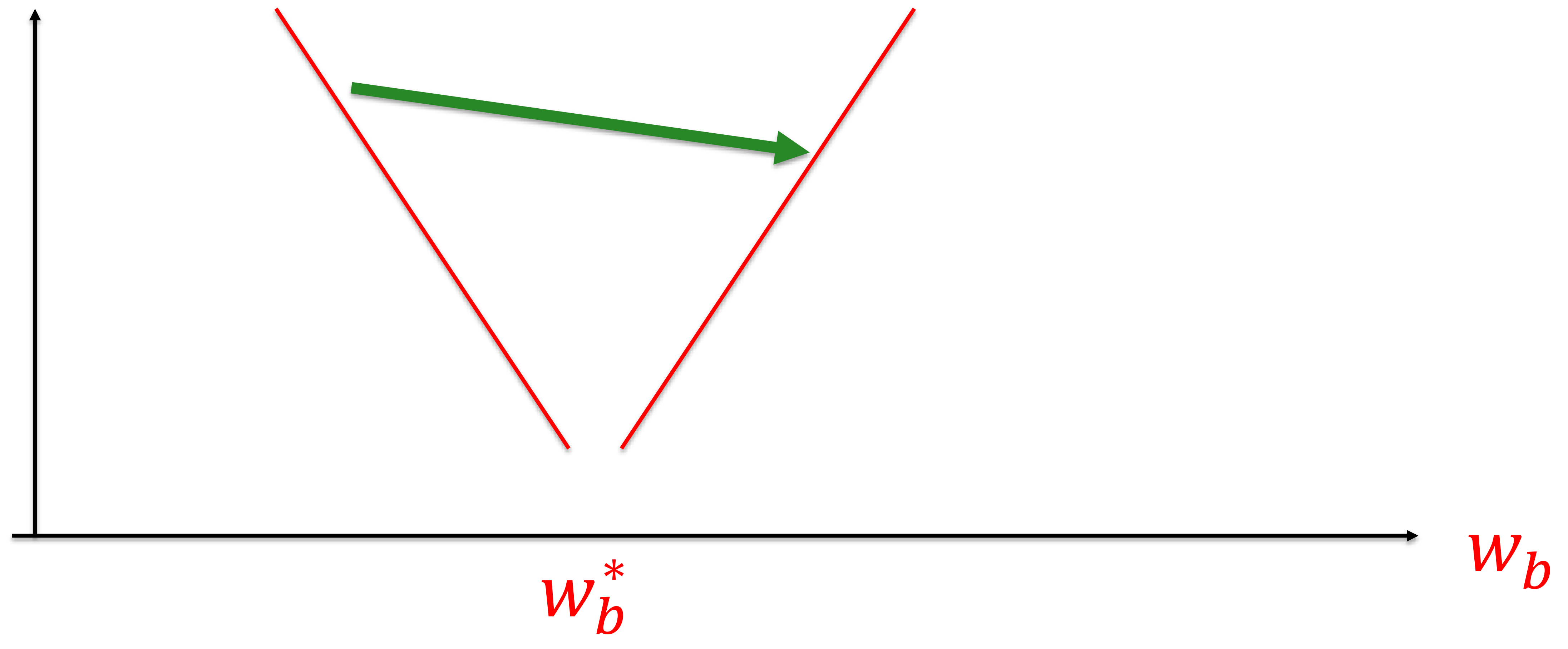


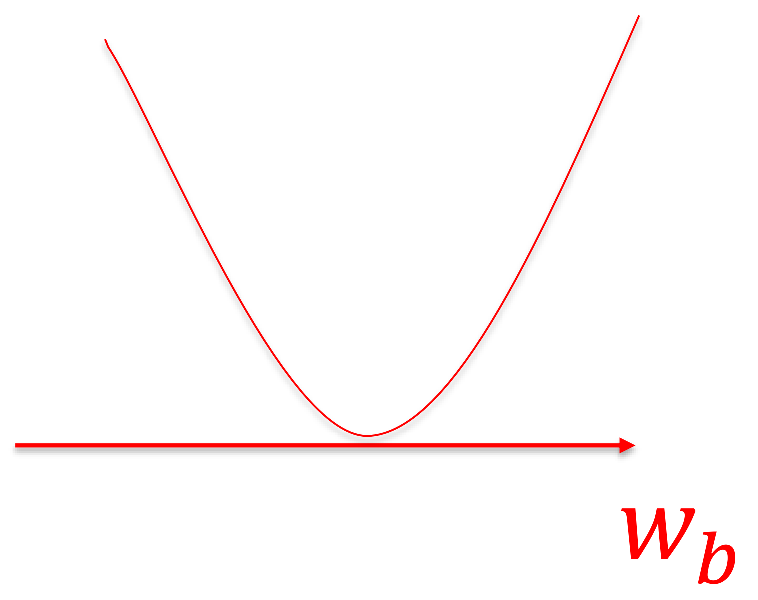
$$\Delta w_{i,j}^{(n)}(m) = -\gamma \frac{dE(\mathbf{w}(m))}{dw_{i,j}^{(n)}} + \alpha \;\; \Delta w_{i,j}^{(n)}(m-1)$$

Blackboard2



gradient descent with momentum (steep valley)





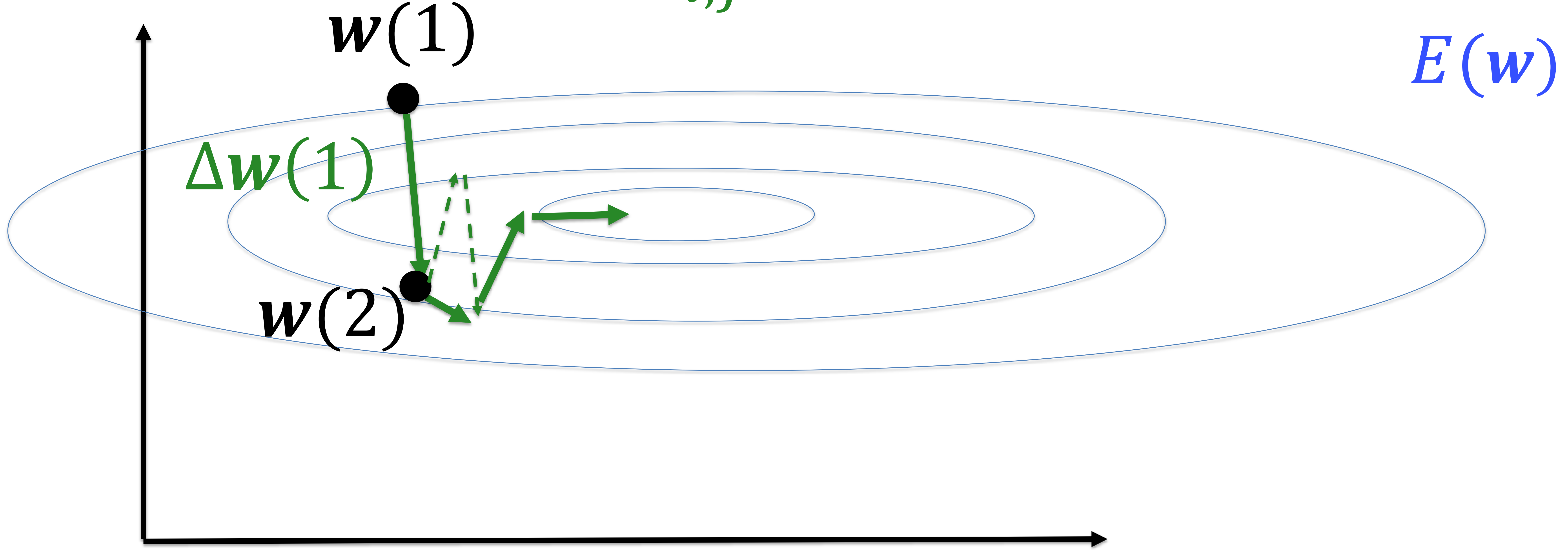
See exercise 2.

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

Your notes. (Calculation of the speed increase and speed decrease, Exercise 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

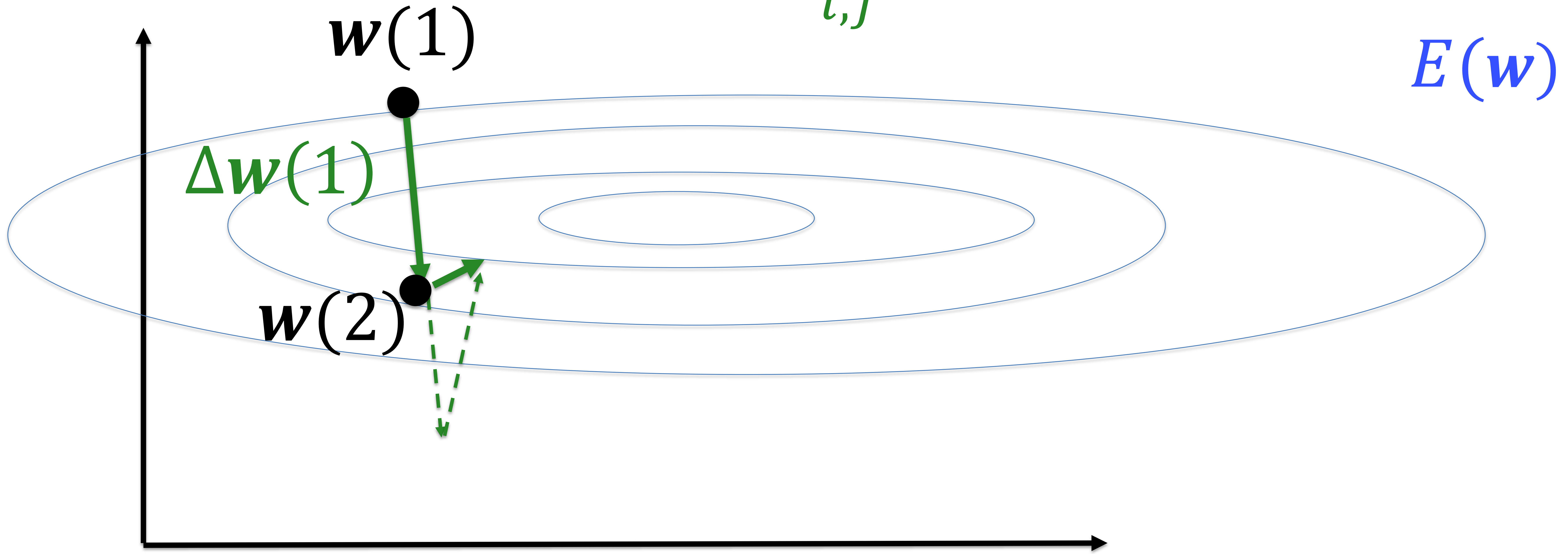
Previous slide.

Graphical illustration of how the momentum term suppresses oscillations.

The direction of changes of the weight vector in time step t=2 adds to the local gradient (perpendicular to the contour lines)
$\alpha\Delta\boldsymbol{w}(1)$
in the direction of the update in time step t=1.
The factor α of the momentum term can be close to 1.

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

Previous slide.

The Nesterov momentum evaluates the gradient at time step t=n+1, not directly at the momentary location $\boldsymbol{w(n+1)}$, but at a hypothetical location

$$\boldsymbol{w(n+1)} + \alpha \Delta w_{i,j}^{(n)}(n)$$

that would be reached by using the momentum term from time step n.

It then combines the local gradient at this hypothetical location with the momentum term, starting (just as in the simple momentum scheme) from the actual location $\boldsymbol{w(n+1)}$.

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

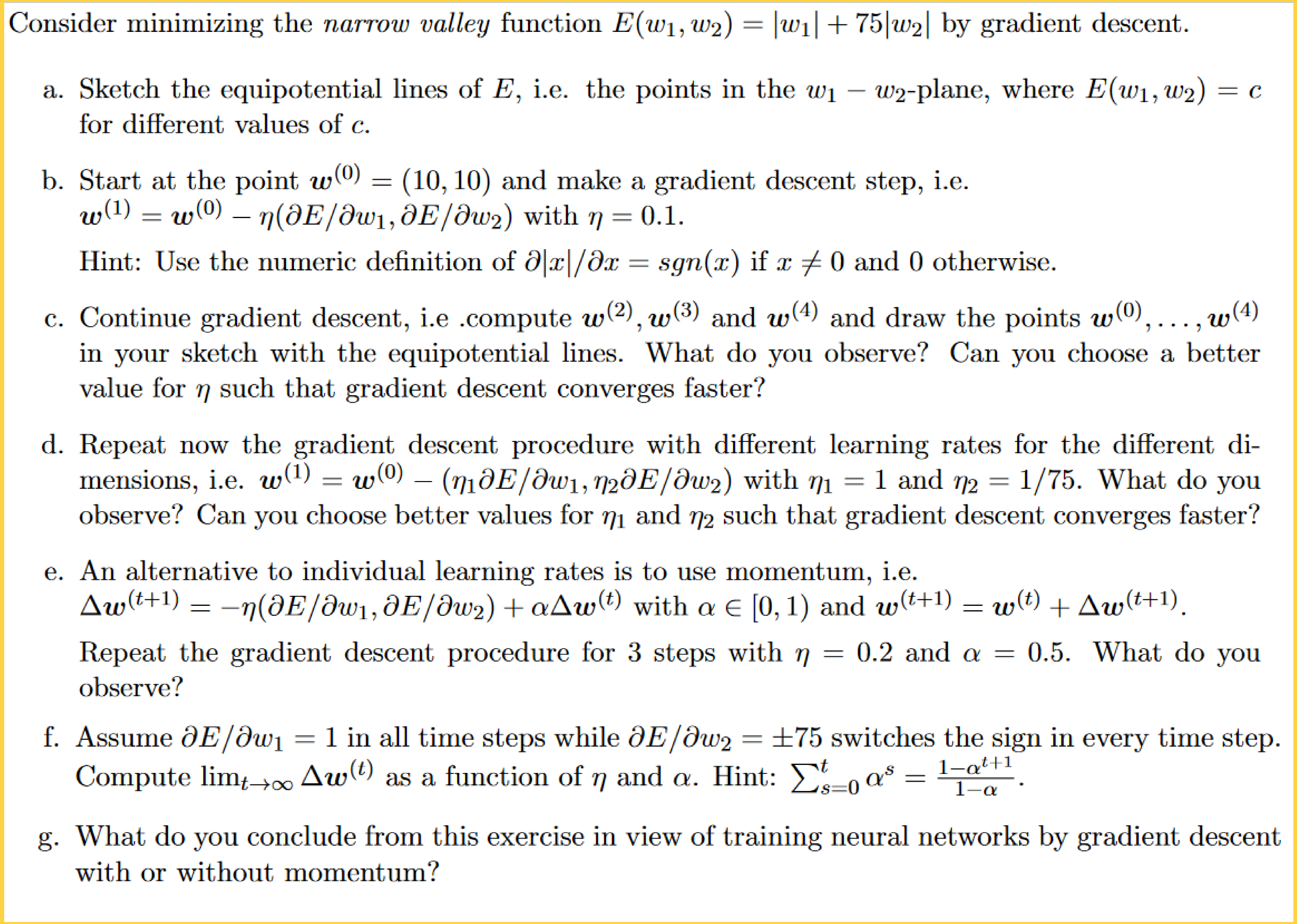
[ ] with a momentum parameter α=0.9 the maximal speed-up is a factor 10

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

Your notes.

Exercise: Momentum and Unitwise learning rates

Next Lecture at 14h15



Consider minimizing the *narrow valley* function $E(w_1, w_2) = |w_1| + 75|w_2|$ by gradient descent.

a. Sketch the equipotential lines of E, i.e. the points in the $w_1 - w_2$-plane, where $E(w_1, w_2) = c$ for different values of c.

b. Start at the point $\boldsymbol{w}^{(0)} = (10, 10)$ and make a gradient descent step, i.e. $\boldsymbol{w}^{(1)} = \boldsymbol{w}^{(0)} - \eta(\partial E/\partial w_1, \partial E/\partial w_2)$ with $\eta = 0.1$.

   Hint: Use the numeric definition of $\partial|x|/\partial x = sgn(x)$ if $x \neq 0$ and 0 otherwise.

c. Continue gradient descent, i.e .compute $\boldsymbol{w}^{(2)}, \boldsymbol{w}^{(3)}$ and $\boldsymbol{w}^{(4)}$ and draw the points $\boldsymbol{w}^{(0)}, \ldots, \boldsymbol{w}^{(4)}$ in your sketch with the equipotential lines. What do you observe? Can you choose a better value for η such that gradient descent converges faster?

d. Repeat now the gradient descent procedure with different learning rates for the different dimensions, i.e. $\boldsymbol{w}^{(1)} = \boldsymbol{w}^{(0)} - (\eta_1 \partial E/\partial w_1, \eta_2 \partial E/\partial w_2)$ with $\eta_1 = 1$ and $\eta_2 = 1/75$. What do you observe? Can you choose better values for η_1 and η_2 such that gradient descent converges faster?

e. An alternative to individual learning rates is to use momentum, i.e. $\Delta\boldsymbol{w}^{(t+1)} = -\eta(\partial E/\partial w_1, \partial E/\partial w_2) + \alpha\Delta\boldsymbol{w}^{(t)}$ with $\alpha \in [0, 1)$ and $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \Delta\boldsymbol{w}^{(t+1)}$.

   Repeat the gradient descent procedure for 3 steps with $\eta = 0.2$ and $\alpha = 0.5$. What do you observe?

f. Assume $\partial E/\partial w_1 = 1$ in all time steps while $\partial E/\partial w_2 = \pm 75$ switches the sign in every time step. Compute $\lim_{t\to\infty} \Delta\boldsymbol{w}^{(t)}$ as a function of η and α. Hint: $\sum_{s=0}^{t} \alpha^s = \frac{1-\alpha^{t+1}}{1-\alpha}$.

g. What do you conclude from this exercise in view of training neural networks by gradient descent with or without momentum?

Your notes.

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Artificial Neural Networks

Loss landscape and optimization methods for deep learning

Part 5: RMSprop and ADAM

1. Questions and Aims of this Lecture
2. Error function: minima and saddle points
3. Why are there so many saddle points?
4. Momentum
5. RMSprop and ADAM

Previous slide.
RMSprop and ADAM are two widely used methods for minibatch updates that combine momentum with further information.

Error function: batch gradient descent

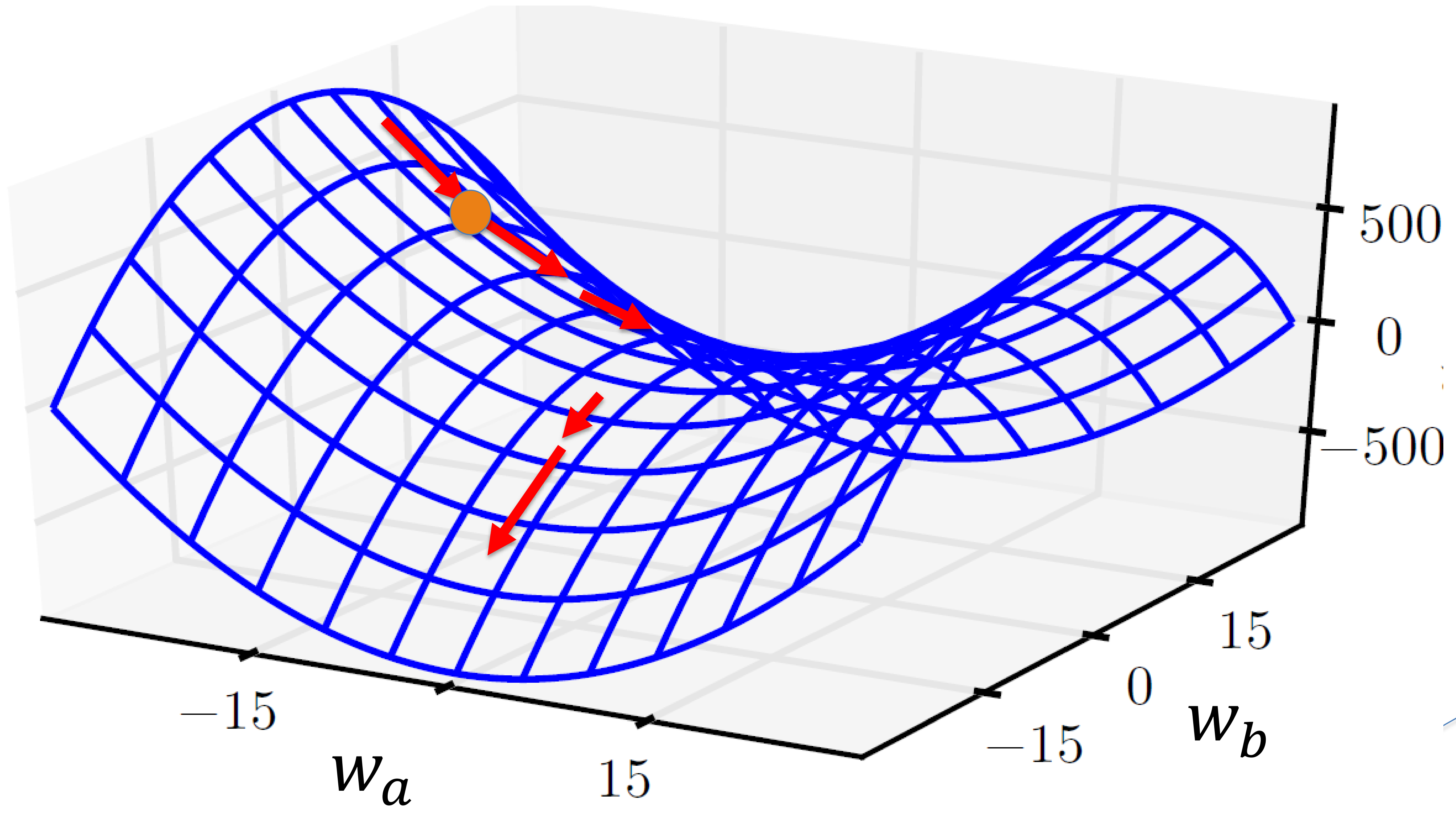
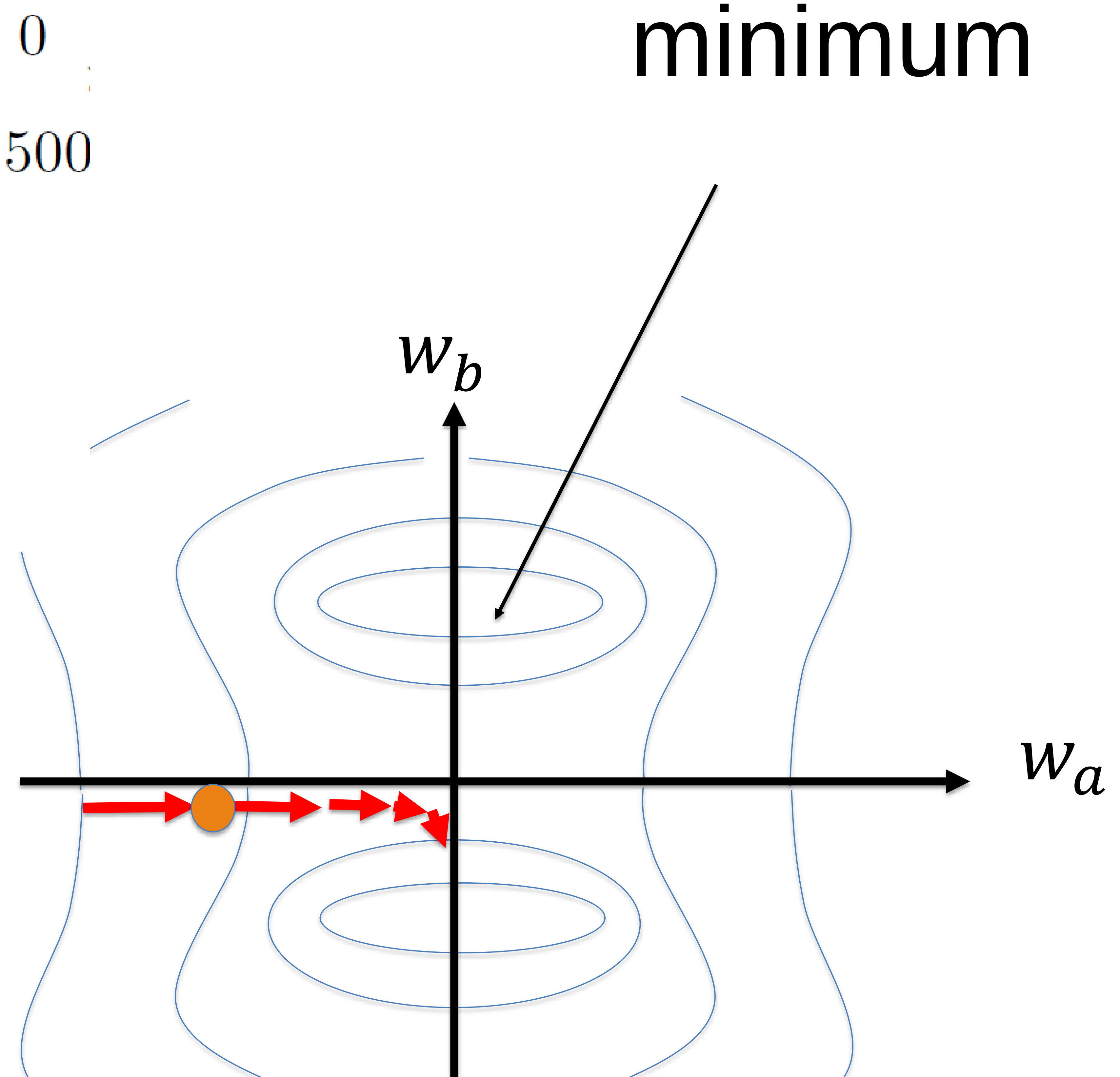


Image: Goodfellow et al. 2016

● $\boldsymbol{w}(1)$

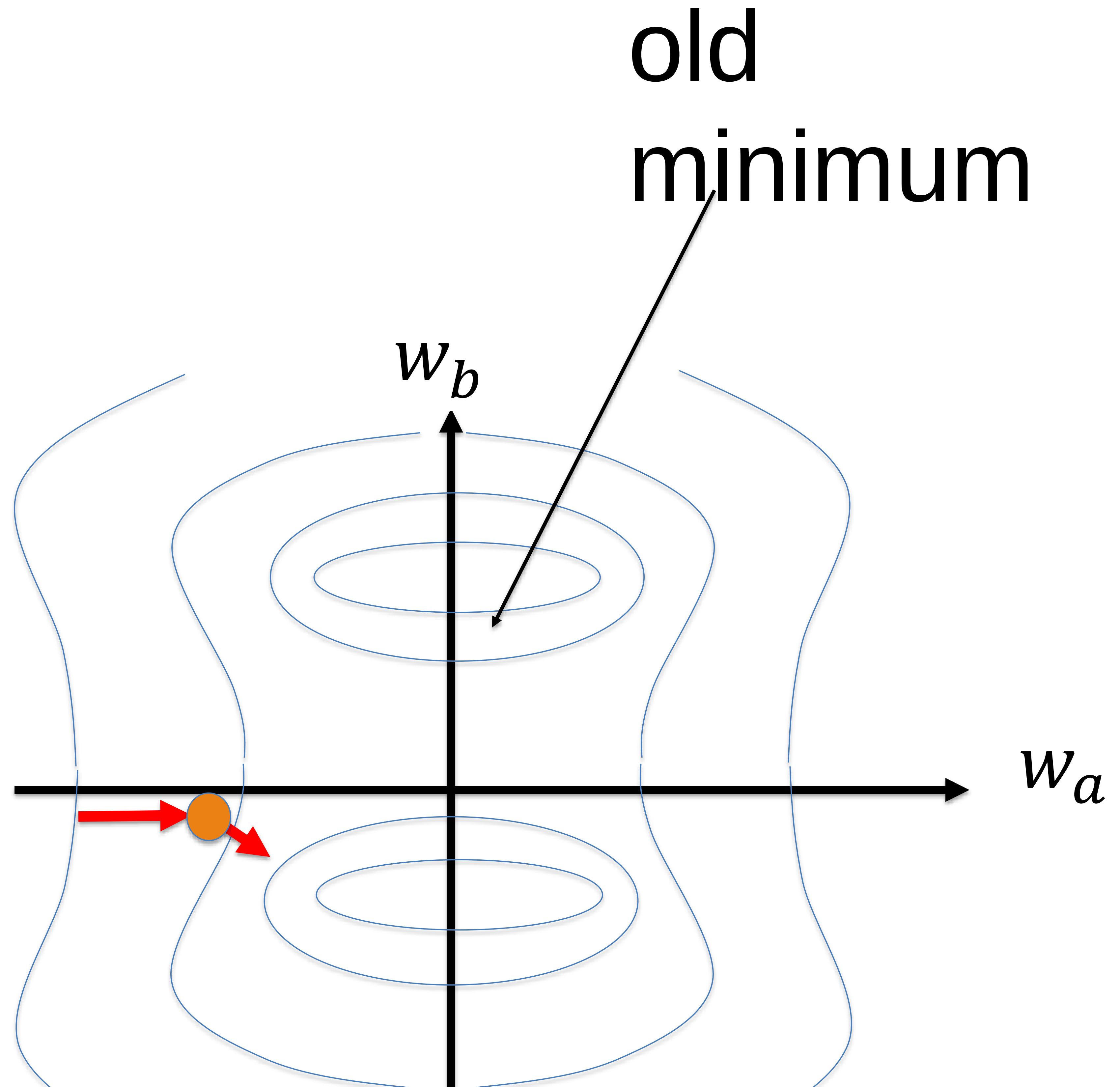


Previous slide.

Let us consider downward movement on an error function with a saddle. For some initial conditions, the trajectory is first attracted toward the saddle before it moves into one of the two minima, depending on the initial condition.

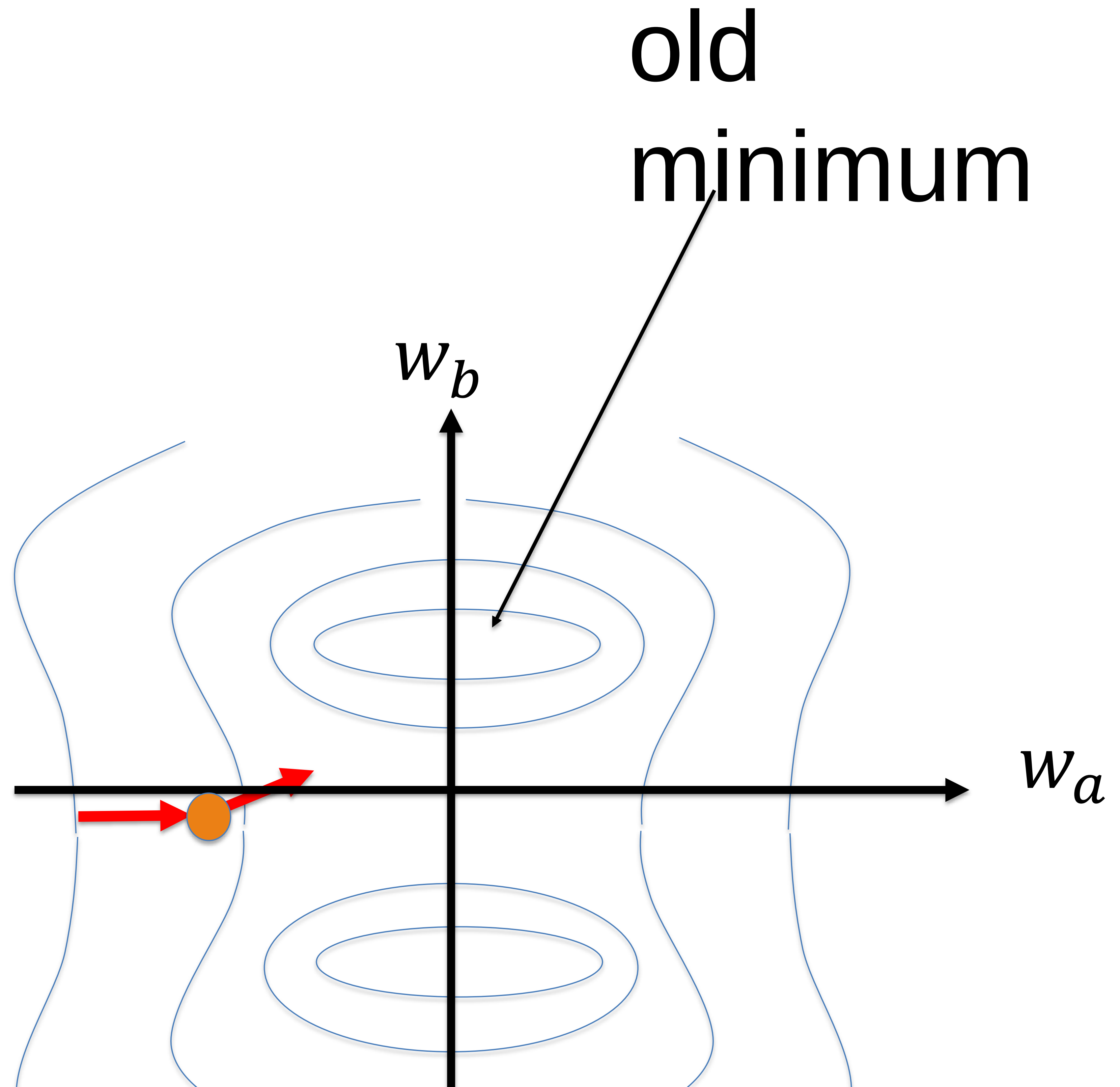
Error function: stochastic gradient descent

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



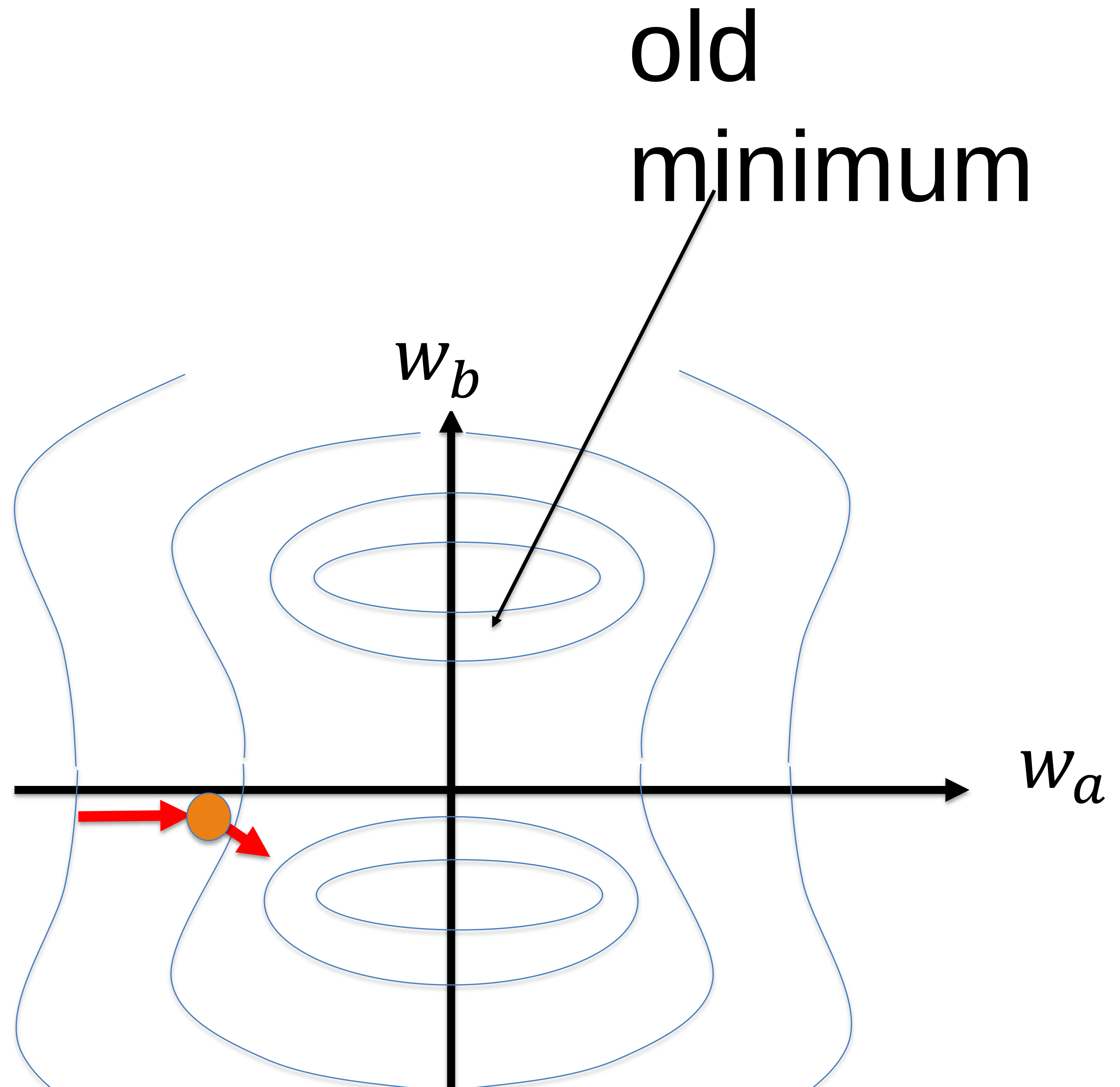
Error function: stochastic gradient descent

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



Error function: stochastic gradient descent

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



Previous slide.
If the error function is evaluated on a minibatch (which means only on part of the data), the exact location of the minima and the saddle is different.

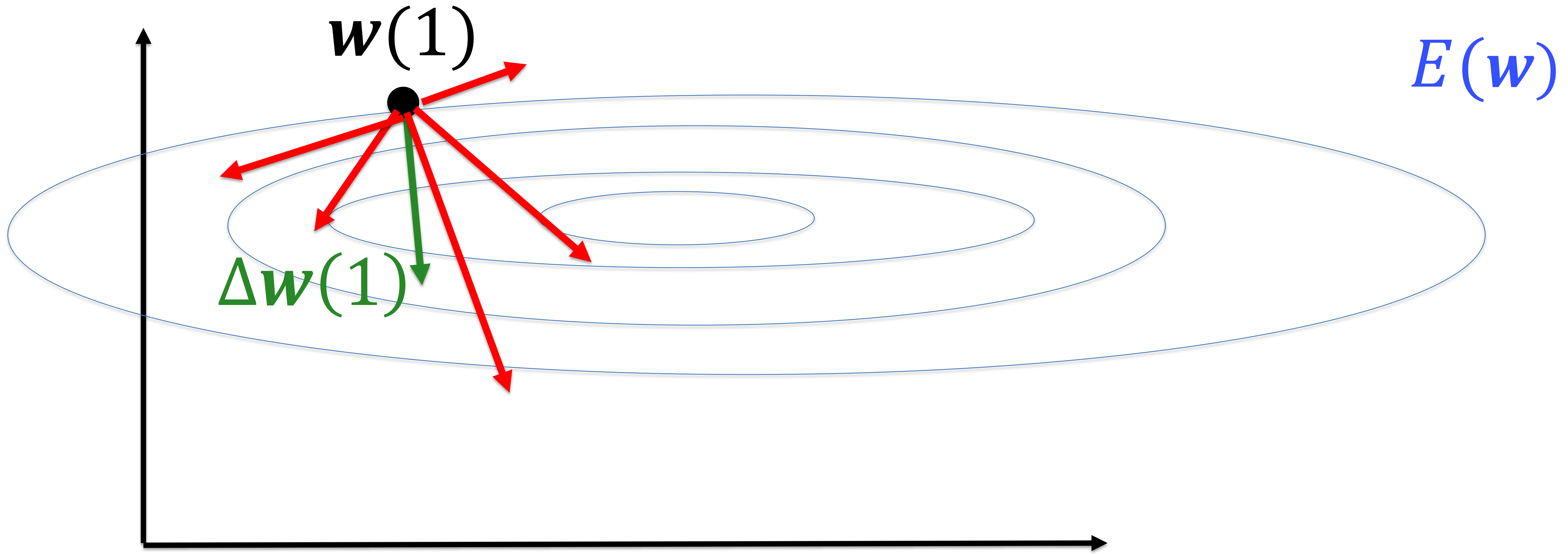
Therefore, for the first minibatch the gradient would lead to the minimum with positive w_b , and for the second minibatch toward the minimum with negative w_b .

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

Previous slide.

The situation is even more extreme with stochastic gradient descent where a single example is evaluated at each time step – whereas the ‘true’ gradient is the one evaluated on all examples (batch update).

The main idea of RMSprop and ADAM is to estimate the ‘mean’ gradient and its variance by a running average.

Note that a momentum term with weight α can be seen as a running average of the gradient of roughly $1/\alpha$ examples (see Exercises).

Recall the role of α in momentum

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

Quiz: RMS and ADAM – what do we want?

A good optimization algorithm

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

[ ] should have smaller update steps for noisy gradients

[ ] the weight change should be smaller for small gradients and larger for large ones, as in standard gradient descent

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

[ ] the weight change should be always the same size (unless gradient is zero)

Previous slide.
Think about what YOU believe would be most useful. Make a commitment by ticking one or several boxes. We will come back to these questions later, at the end of this part.

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 2nd moment from k=1/ρ samples

Running Mean: use momentum

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

Running second moment: average the squared gradient

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

Previous slide.
Hence, the mean of the gradient is estimated using a momentum term (‘online average’) with parameter

$$\rho_1$$

Similarly, the second moment of the gradient is estimated using an online average with parameter

$$\rho_2$$

Note that the second moments form a matrix of correlations. Here we focus on the ‘diagonal terms’ only which are simply the square of one component of the gradient.

Attention: 1. do not confuse this with the Hessian matrix of second derivatives.
2. do not confuse the second moment with the covariance matrix.
3. In the above notation ρ_1 and ρ_2 have slightly different ‘normalizations’ when comparing the two terms on the right-hand side of each equation.

Stochastic gradient evaluation: signal-to-noise-ratio

Example:
consider 3 weights w_1, w_2, w_3

Raw Gradient:

$$\Delta w_i = -\frac{dE(\boldsymbol{w}(m))}{dw_i}$$

Mean over k samples

$$< \frac{dE(\boldsymbol{w}(m))}{dw_i} >$$

average the squared gradient over k samples

$$< \left(\frac{dE(\boldsymbol{w}(m))}{dw_i}\right)\left(\frac{dE(\boldsymbol{w}(m))}{w_i}\right) >$$

Time series of gradient by sampling:

for Δw_1: 1.1; 0.9; 1.1; 0.9; …

for Δw_2: 0.1; 0.1; 0.1; 0.1; …

for Δw_3: 1.1; -0.9; 1.1 -0.9; …

Blackboard 3/Exerc. 1

Exercise 3a-c

We consider stochastic gradient descent in a network with three weights, (w_1, w_2, w_3).

Evaluating the gradient for 100 input patterns (one pattern at a time), we observe the following time series

for w_1: observed gradients are 1.1; 0.9, 1.1; 0.9; 1.1; 0.9; ...

for w_2: observed gradients are 0.1; 0.1; 0.1; 0.1; 0.1; ...

for w_3: observed gradients are 1.1; -0.9; 1.1; -0.9; 1.1; -0.9; ...

a. Calculate the mean gradient (first moment m_1) $\langle g_k \rangle$ for w_k, $k \in [1, 2, 3]$.

b. Calculate the mean of the squared gradient (second moment m_2) $\langle g_k^2 \rangle$ for w_k, $k \in [1, 2, 3]$.

c. Divide the result of (a) by that of (b) so as to calculate $\langle g_k \rangle / \langle g_k^2 \rangle$ as well as $\langle g_k \rangle / \sqrt{\langle g_k^2 \rangle}$ for w_k, $k \in [1, 2, 3]$.

additional questions:

- How would you do an online estimate of $\langle g_k \rangle$ and $\langle g_k^2 \rangle$

- How would you define a ‘signal-to-noise ratio’ of stochastic gradients?

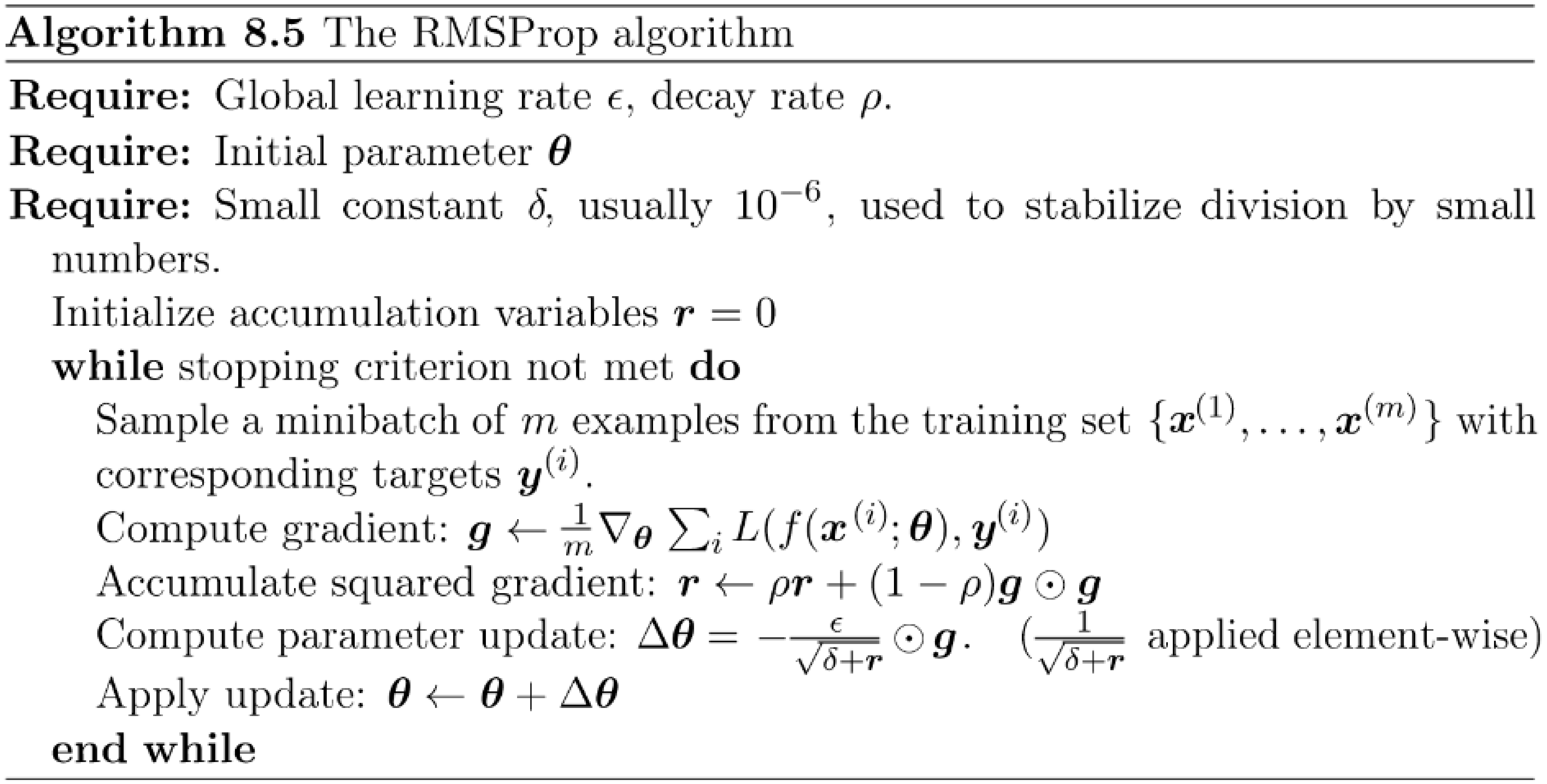
Adam and variants

The above ideas are at the core of several algos

- RMSprop
- RMSprop with momentum
- ADAM

Your notes on the exercise.

RMSProp



Algorithm 8.5 The RMSProp algorithm

```
Require: Global learning rate $\epsilon$, decay rate $\rho$.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, 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., Deep Learning 2016, MIT Press

Previous slide.

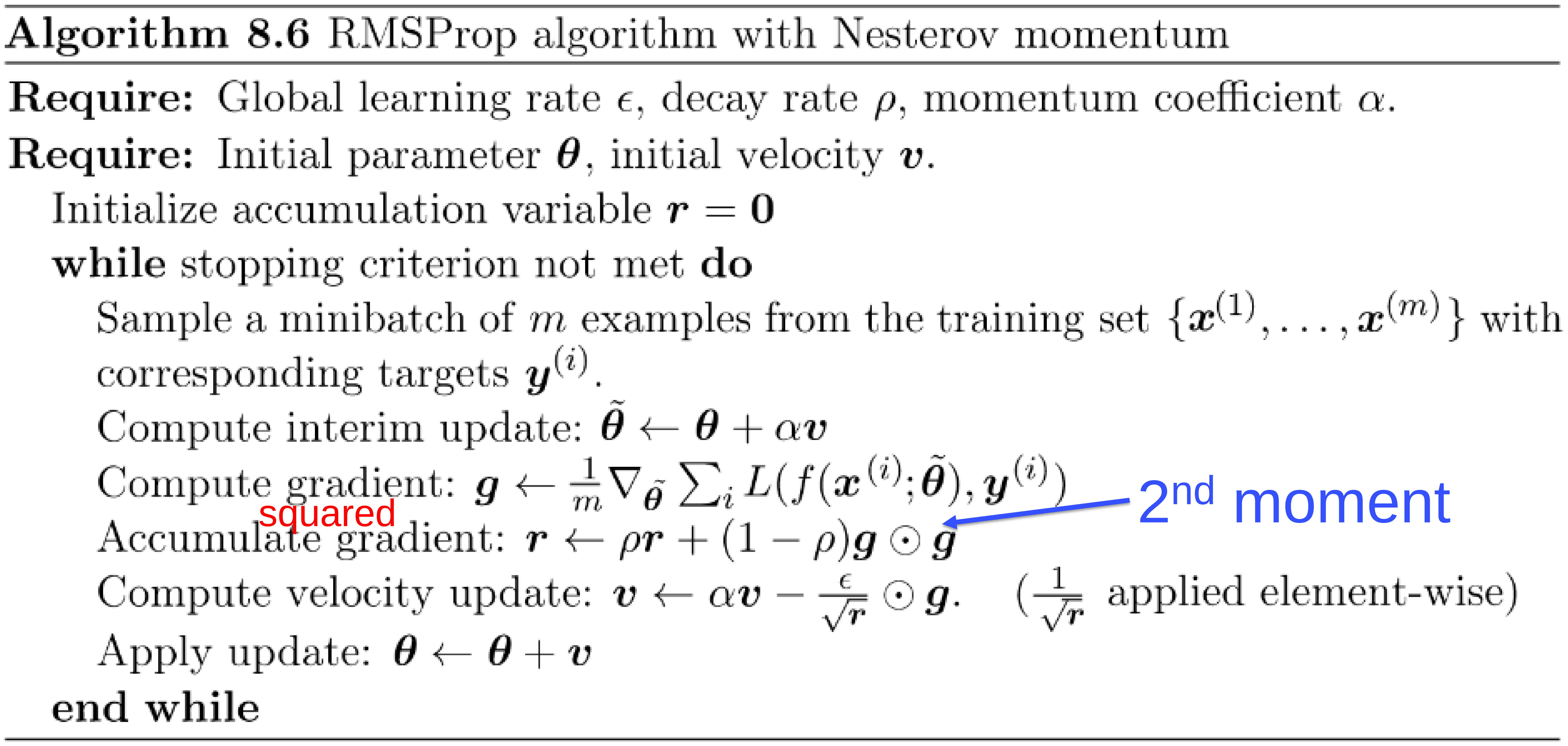
RMSprop algorithm.

The variables r estimate the diagonal elements of the second moment of the gradient.
The operator ‘circle-dot’ indicates elementwise multiplication.

The update step is scaled by the square-root of the second moment.
The delta is a small number to stabilize the division.

There is no smoothing of the gradient itself (no momentum term).

RMSProp with Nesterov Momentum



Algorithm 8.6 RMSProp algorithm with Nesterov momentum

```
Require: Global learning rate $\epsilon$, decay rate $\rho$, momentum coefficient $\alpha$.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  Initialize accumulation variable $\boldsymbol{r} = \mathbf{0}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
    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
```

squared

2nd moment

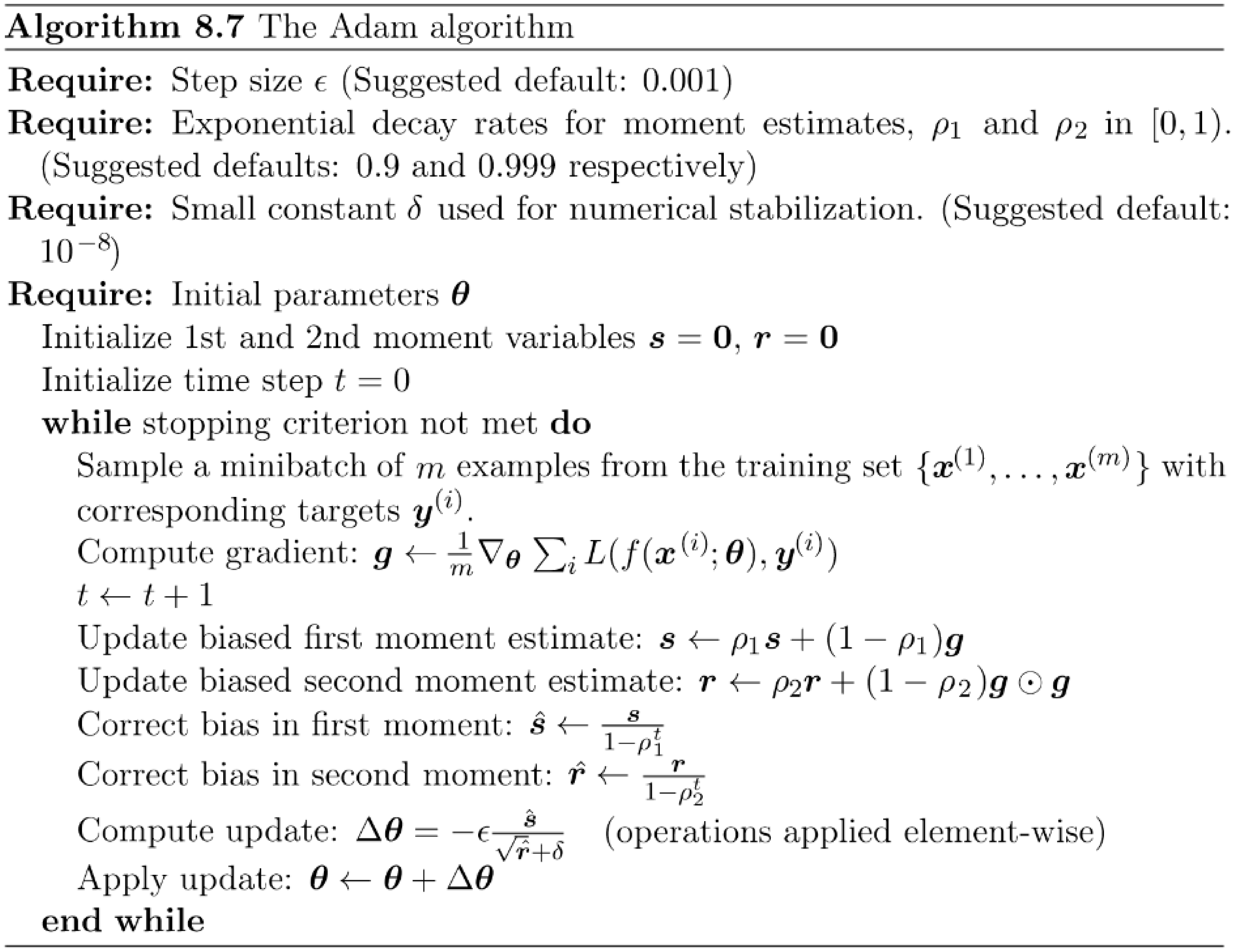
Goodfellow et al., Deep Learning 2016, MIT Press

Previous slide.

This is the version with smoothing (the delta has been suppressed in the notation but should always be kept in practice.)

Note that second moment and variance are not exactly the same (see also exercises). For variance, you subtract the mean before you square.

Adam



Algorithm 8.7 The Adam algorithm

```
Require: Step size $\epsilon$ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in $[0, 1)$.
  (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant $\delta$ used for numerical stabilization. (Suggested default:
  $10^{-8}$)
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  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., Deep Learning 2016, MIT Press

Previous slide.

The first moment is the online average of the mean of the gradient, equivalent to the momentum.

The second moment is similar to the variance. But in contrast to the variance, the mean is not subtracted before squaring.

The bias correction terms are a bit arbitrary. The idea is that (as we have seen for the momentum term earlier) evaluating a constant gradient using a momentum term with parameter ρ gives effectively rise to a factor $1/[1-\rho]$. However, since it takes some time to build up this factor, one could artificially introduce this factor in the first few time steps – and this is what is done in this algorithm. However, this argument makes sense only if the gradient is indeed constant over many steps!

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

Your notes.

Quiz (2nd vote): RMSprop with Momentum and ADAM

A good optimization algorithm (take ADAM as example)

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

[ ] should (in batch mode) have the same weight update step for small gradients and for large ones

[ ] should have smaller update steps for noisy gradients during stochastic gradient descent

Your notes.

Summary:

THE END

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

- Adam and variants:
  - adapt learning step size to certainty
  - include momentum
  - smaller effective learning step for noisy directions

Previous slide.

We can distinguish three main features of momentum:
- it suppresses oscillations. Note that oscillations arise even in the batch setting if the valley of the error function has steep slopes and the learning rate is chosen too big.
- in a narrow valley the effective step size of weight changes aligned with the valley axis increases, whereas those point toward the steep walls of the valley decreases.
- in stochastic online gradient descent, momentum acts as an exponentially shaped averaging filter.

In addition to momentum, Adam (and its variants) also estimate the second moment of the gradient. This estimate can then be used to adapt the step size to the certainty: smaller weight updates if the gradient estimate is noisy (has a large second moment compared to its mean).

Artificial Neural Networks

EPFL

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Loss landscape and optimization methods for deep learning

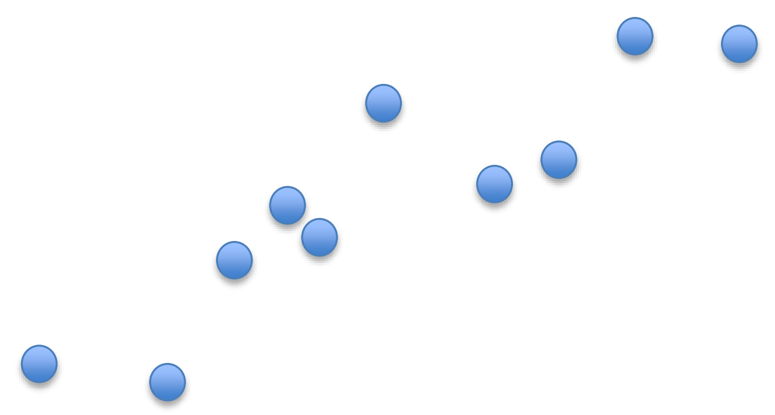
Part 6: No Free Lunch Theorem

Previous slide.
No Free Lunch theorems (there are several variants) are foundational and philosophically important to answer the question: why do deep neural networks work so well?

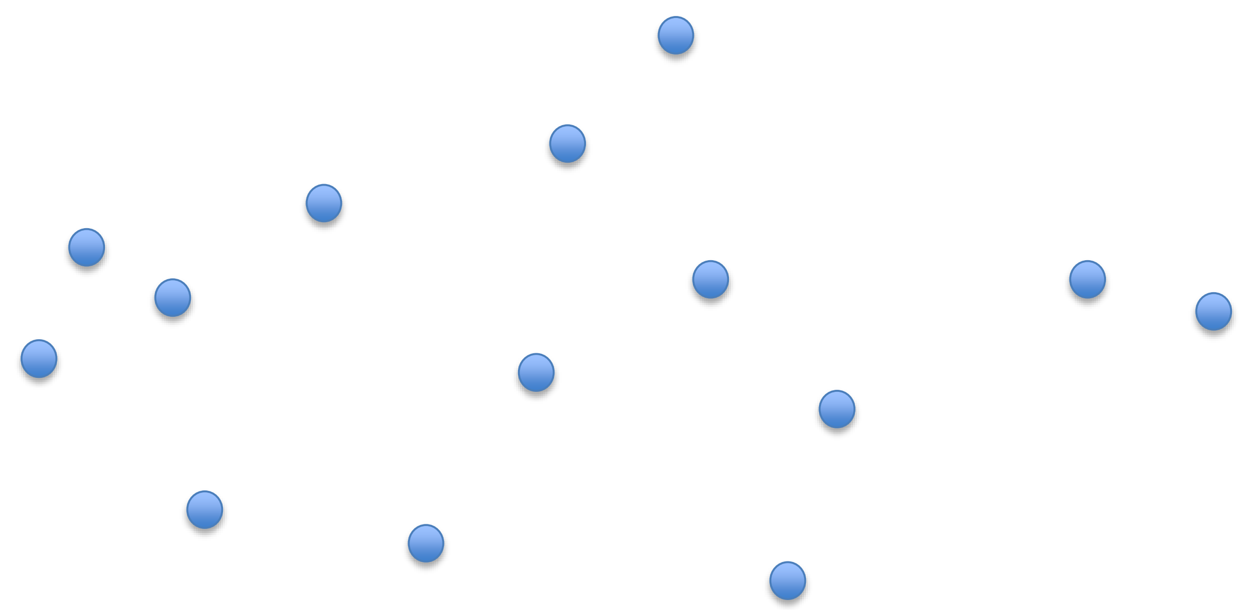
No Free Lunch Theorem

Which data set looks more noisy?

A



B



Commitment:
Thumbs up

Which data set is easier to fit?

Commitment:
Thumbs down

Previous slide.

Let us start with two data sets.

No Free Lunch Theorem

Previous slide.

And here a possible explanation (hidden behind the blue boxes).

No Free Lunch Theorem

Your notes

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.

Previous slide.

The conclusion is: there is no reason to believe that an algorithm that works well on one data set will also work well on an arbitrarily chosen other data set.

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.

Previous slide.

Even worse, if the algo works well on some problem, there must exist another problem on which the algorithm works badly.

Quiz: No Free Lunch (NFL) Theorems

Take neural networks with many layers, optimized by Backprop (with momentum/ADAM) 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.

Your notes.

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

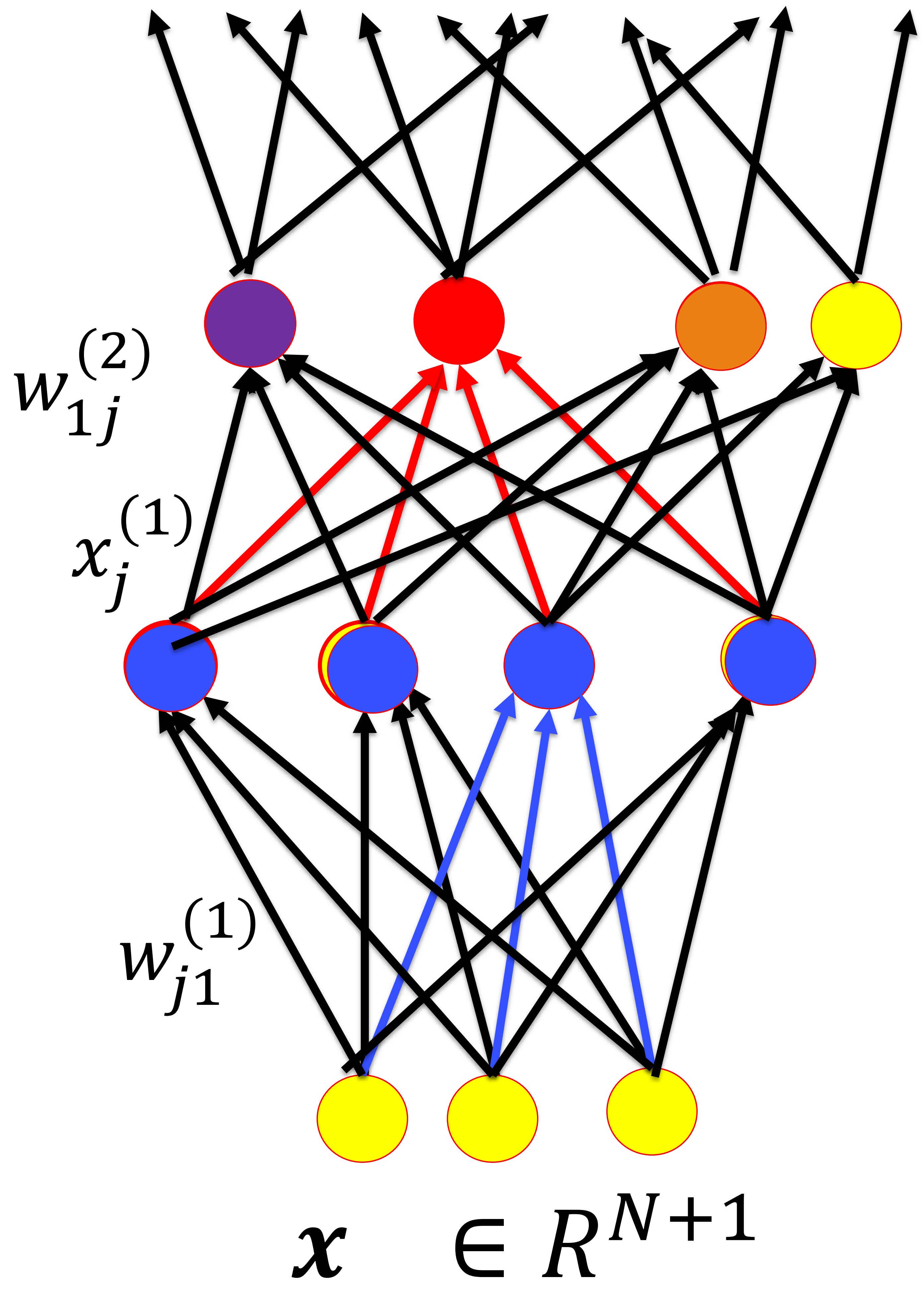
- Somehow the prior structure of the deep network matches the structure of the real-world problems we are interested in.

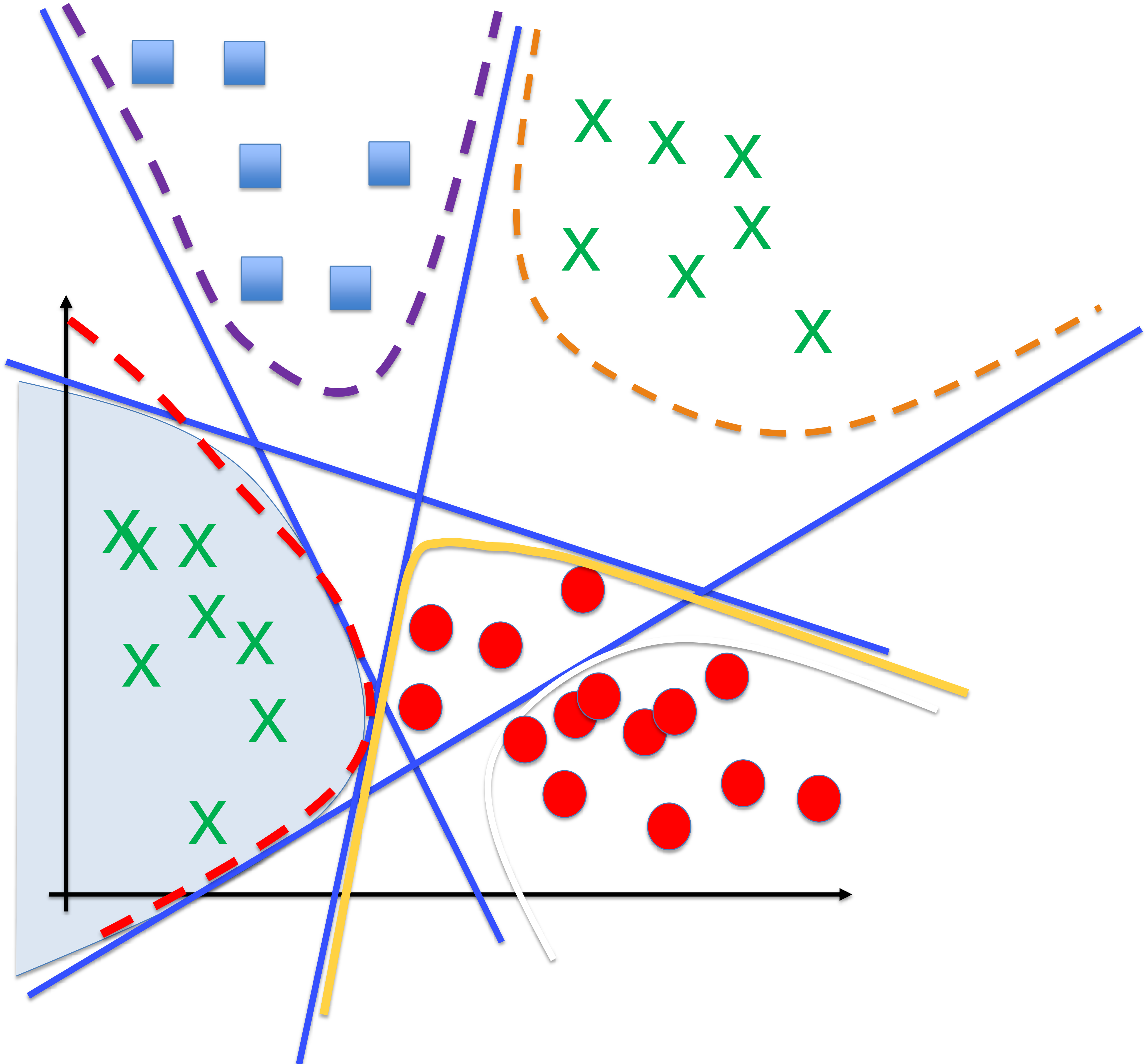
Previous slide.

The reason that deep networks work well must be linked to the type of data on which we test them.

No Free Lunch (NFL) Theorems

Geometry of the information flow in neural network





Previous slide.

One possible explanation of why neural networks work well is the notion of hyperplanes. Even though the data is local, you make a cut through the whole space. This predefines additional ‘compartments’ that can be reused later for other data.

This argument might be applicable in the last few layers before the output.

Reuse of featuers in Deep Networks (schematic)

animals

birds

4 legs

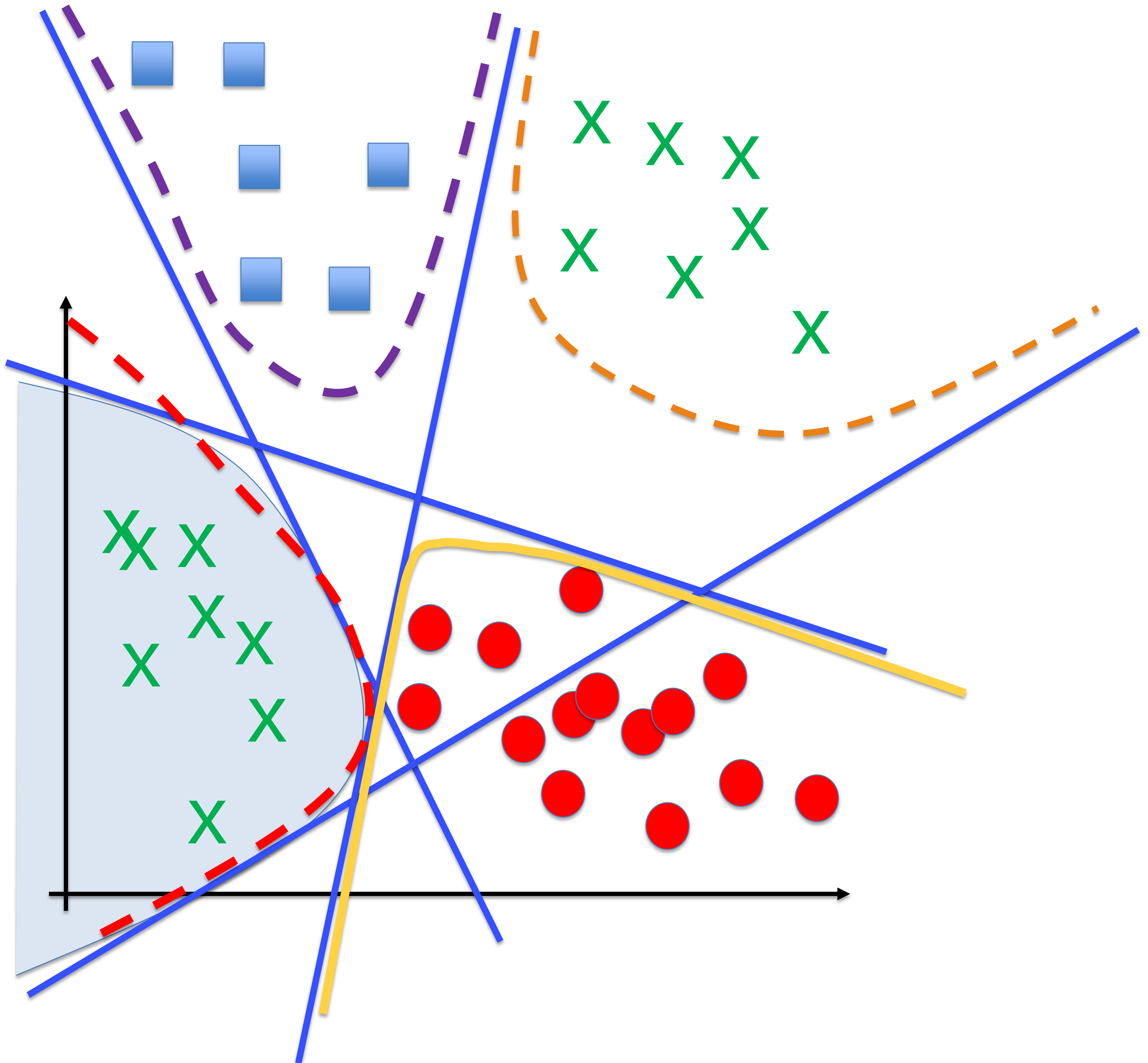
wings

snout

fur

eyes

tail



Previous slide.

A specific illustration of this idea is given here

Summary: No Free Lunch (NFL) Theorems and Deep Networks

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

Example:
- images, translation invariance
- music, tone translation invariance
- known symmetries of tasks

Previous slide.

Prior knowledge is important. We can use prior knowledge when we design the network architecture.

Artificial Neural Networks

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Loss landscape and optimization methods for deep learning

Part 7: Deep Networks versus Shallow Networks

Previous slide.

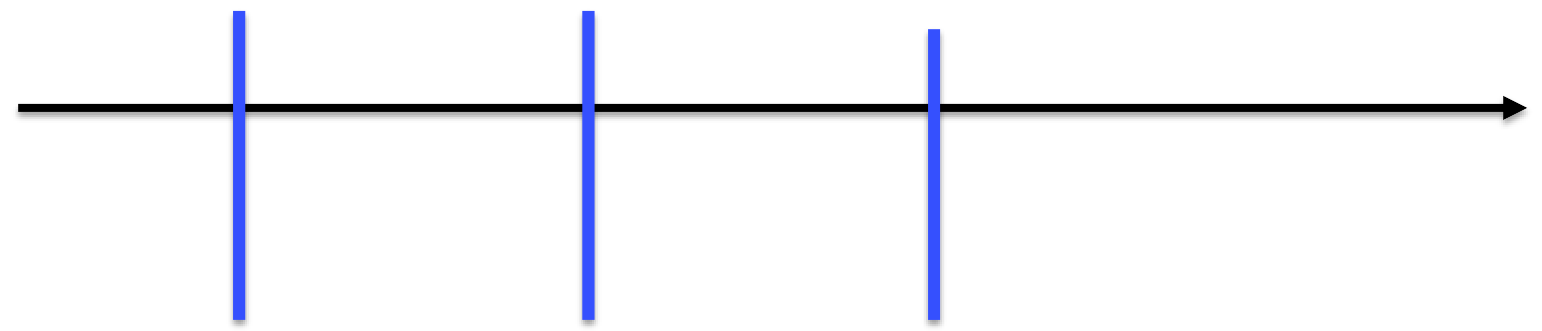
In the following we explore the idea of carving out regions in the space by hyperplanes.

Distributed representation

How many different regions are carved

In 0dim input space

In 1dim input space with:
0 hyperplanes
1 hyperplane
2 hyperplanes?
3 hyperplanes?
4 hyperplanes?



Previous slide.

First we work in zero dimensions. There is only one dot, this is the smallest possible region: d=0 → 1 region

We now work in one dimension (horizontal black axis).
The continuous axis is one connected region.
If we add a first hyperplane, we cut the axis into 2 separate regions. Therefore we have added one extra region.
After adding the nth hyperplane, we have n+1 regions. Each hyperplane adds one ‘crossing’ of the horizontal axis.

d =1 → n+1 regions (where n is the number of hyperplanes in 1d)

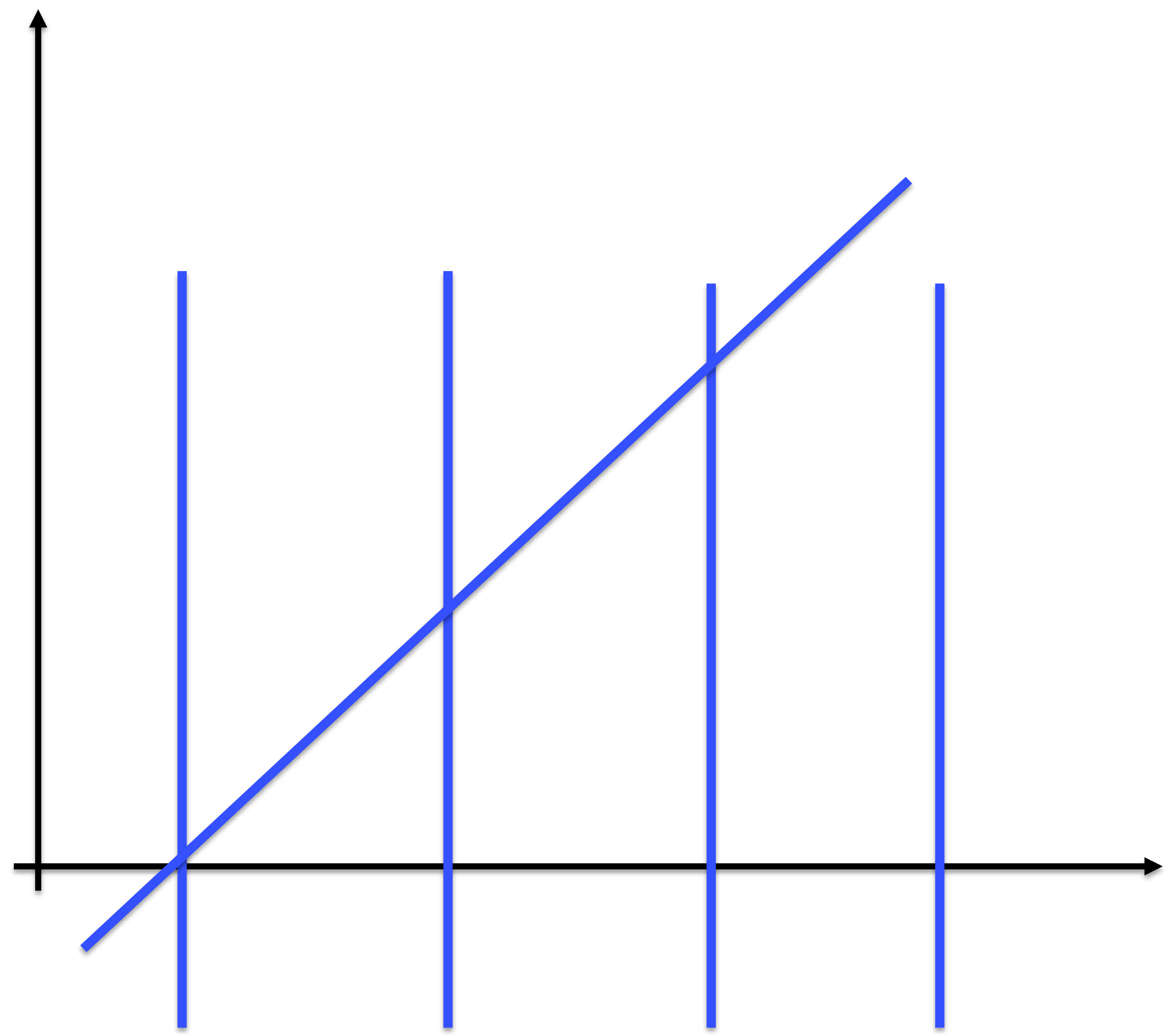
Distributed representation

How many different regions are carved

In 2dim input space with:

3 hyperplanes?
4 hyperplanes?

Increase dimension
= turn hyperplane
= new crossing
= new regions



Previous slide.

Suppose we have n hyperplanes in 1 dimension.
This corresponds to n PARALLEL hyperplanes in 2 dimension. The number of separate regions is still n+1, just as in 1 dimension.

Suppose now we slowly turn one of the hyperplanes into an ARBITRARY position.
Each time it crosses another hyperplane the tilting process creates a new region.
Hence n-1 new regions are created.

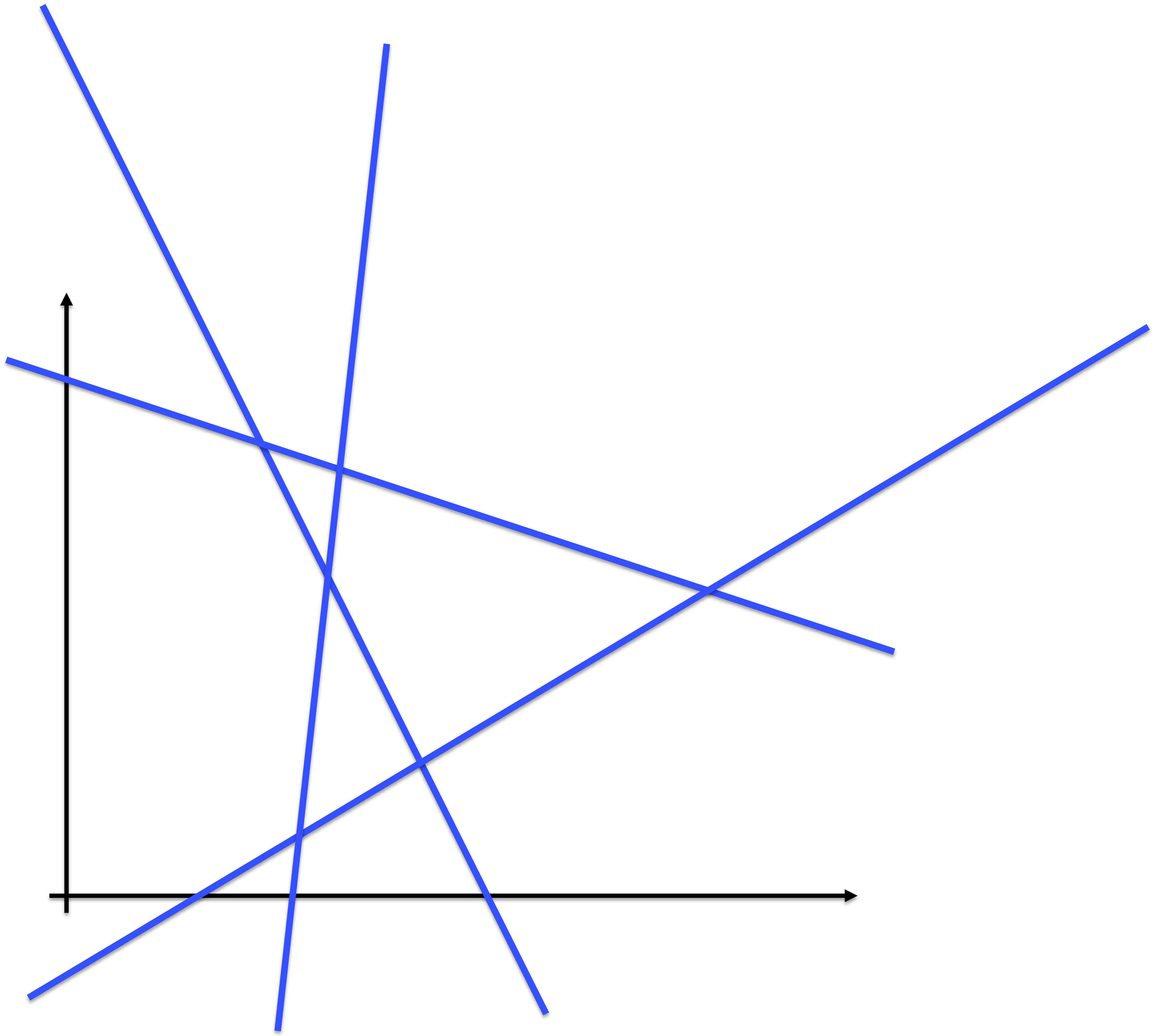
Repeat this with the all n hyperplanes. Each time I create (n-1) new regions – except that I have now overcounted by a factor of 2.

Distributed multi-region representation

How many different regions are carved

in 2dim input space by

n hyperplanes?



$1 + n + n(n-1)/2$

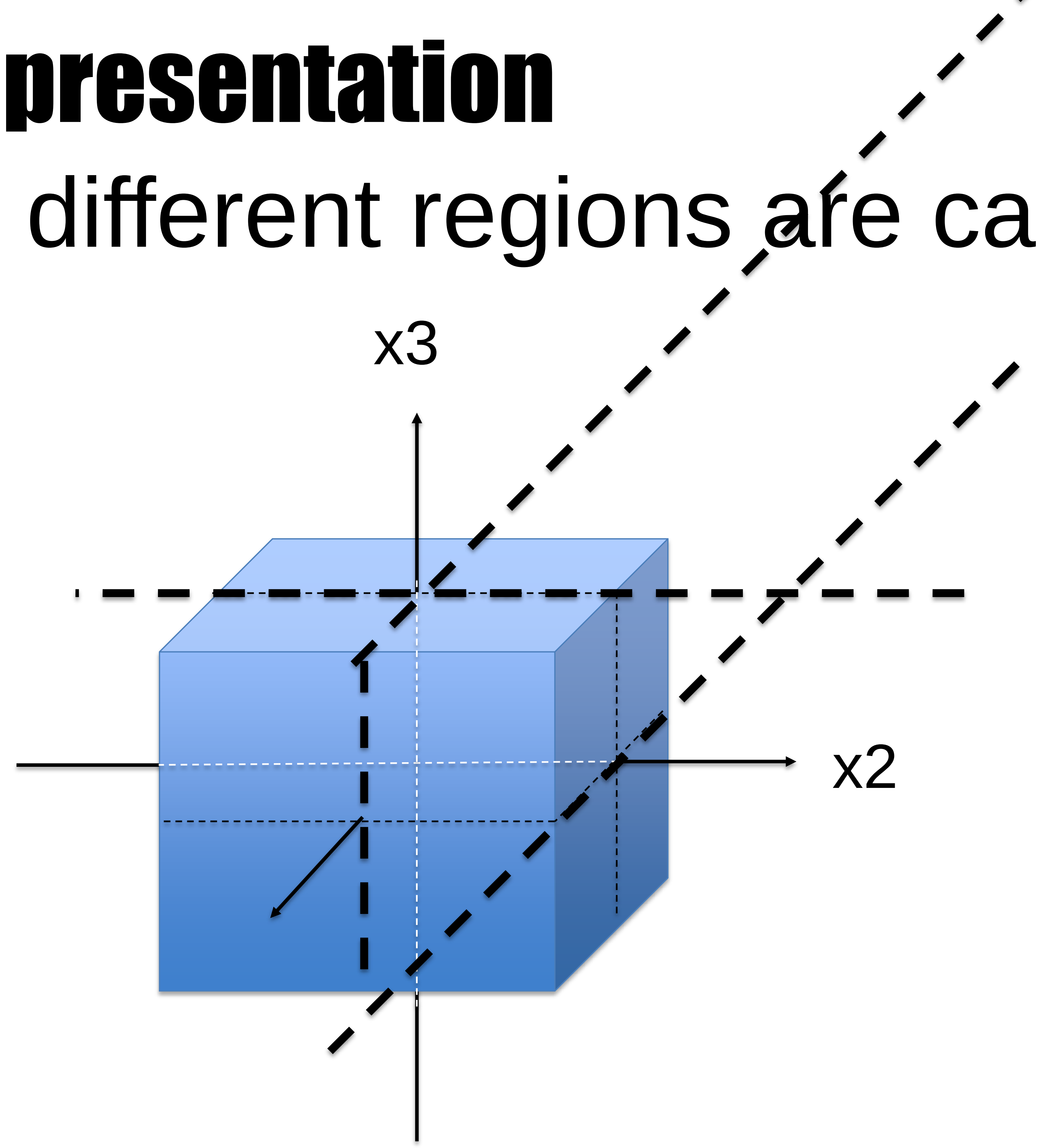
Previous slide.

In 2 dimension:
I have n lines. If I tilt one line → adds n-1 new crossings → adds n-1 new regions.
I can do this for each of the n existing lines: they were parallel in the 1d setting, I turn it = add new crossings.
Total (n)(n-1)/2 new crossings (corrected for counting twice).

But in 1d, I had already n+1 regions. Therefore, total number of regions is given by the formula 1 + n + n(n-1)/2

Distributed representation

How many different regions are carved



In 3d input space by:

1 hyperplane
2 hyperplanes

3 hyperplanes?

4 hyperplanes?

Previous slide.
Let us extend the argument to three dimensions.

At the beginning it is easy, and the number of regions increases exponential.

But how do we treat 4 hyperplanes?

Distributed multi-region representation

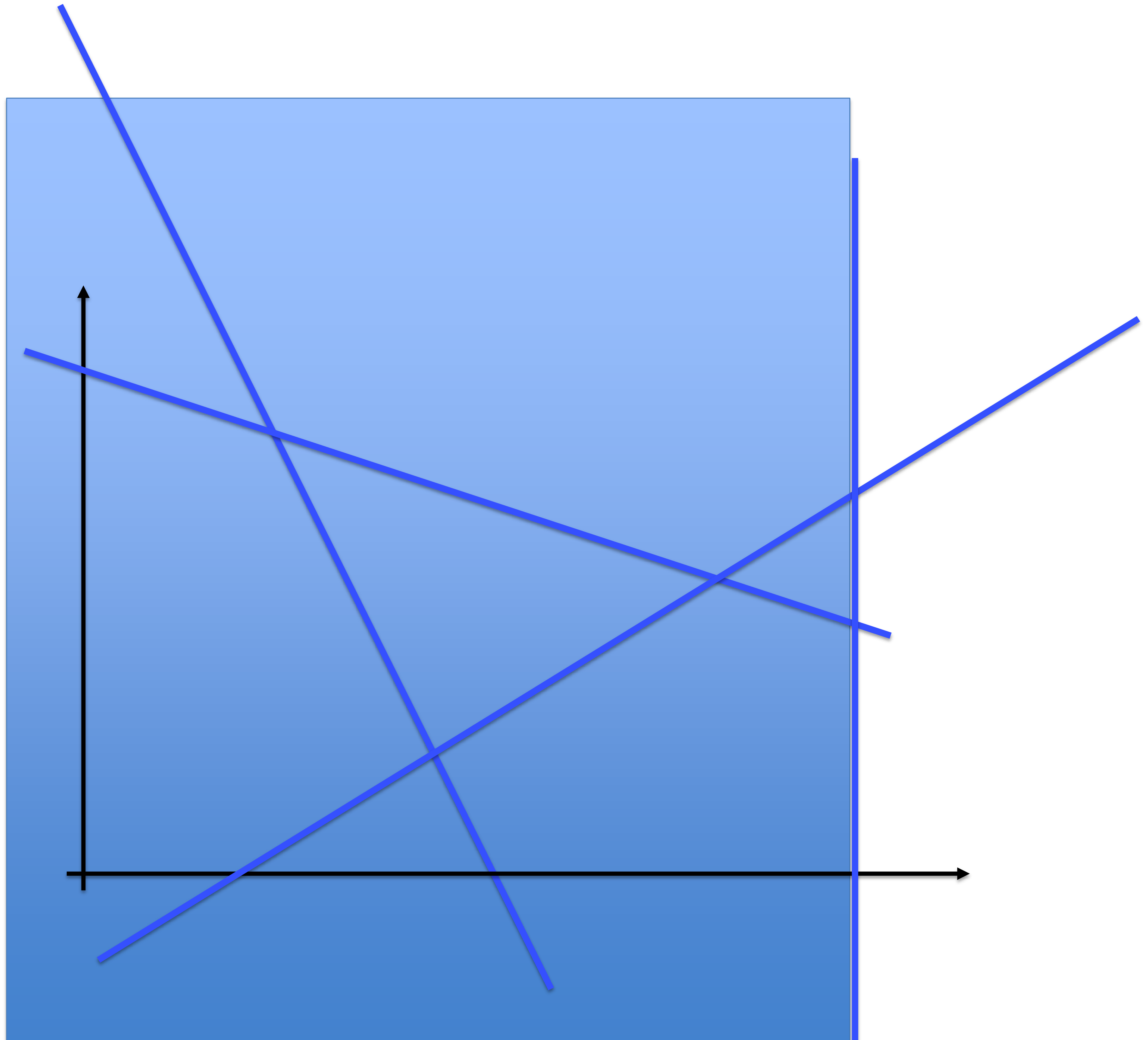
How many different regions are carved

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



Previous slide.

In 3 dimension:
I have n vertical hyperplanes, I look on these from the top. Thus the third dimension is not yet used. Now I tilt one of these hyperplanes.
→ the tilting adds as many new regions as there were **crossings** in 2 dimensions of the remaining n-1 hyperplanes → adds (n-1)(n-2)/2 new regions.

Again, this tilting argument can be repeated for each of the n vertical planes (but avoid double counts!)

So we can build a proof by induction:
The number of NEW regions with n hyperplanes in d dimensions, is linked to the number of crossings with n-1 hyperplanes in d-1 dimensions.

The total number of regions is the NEW regions plus the number of OLD regions with n hyperplanes in d-1 dimensions.

Distributed multi-region representation

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

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

$$number \sim O\left(\frac{n^{d}}{d!}\right)$$

But, without additional layers, we cannot learn arbitrary targets by assigning arbitrary class labels {+1,0} to each region, unless exponentially many hidden neurons:
generalized XOR problem

Your notes.

Conclusion:

1. MANY regions created by a n hyperplanes in d dimension.

2. However, this does not mean that all of these can be assigned to arbitrary classes. For example, 2 hyperplanes carve 4 regions, but an XOR configuration cannot be solved unless we add an extra layer.

3. The argument can then be repeated for all layers. The input dimension in layer n is the number of neurons in layer n-1.

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
in the sense of a generalized XOR problem.

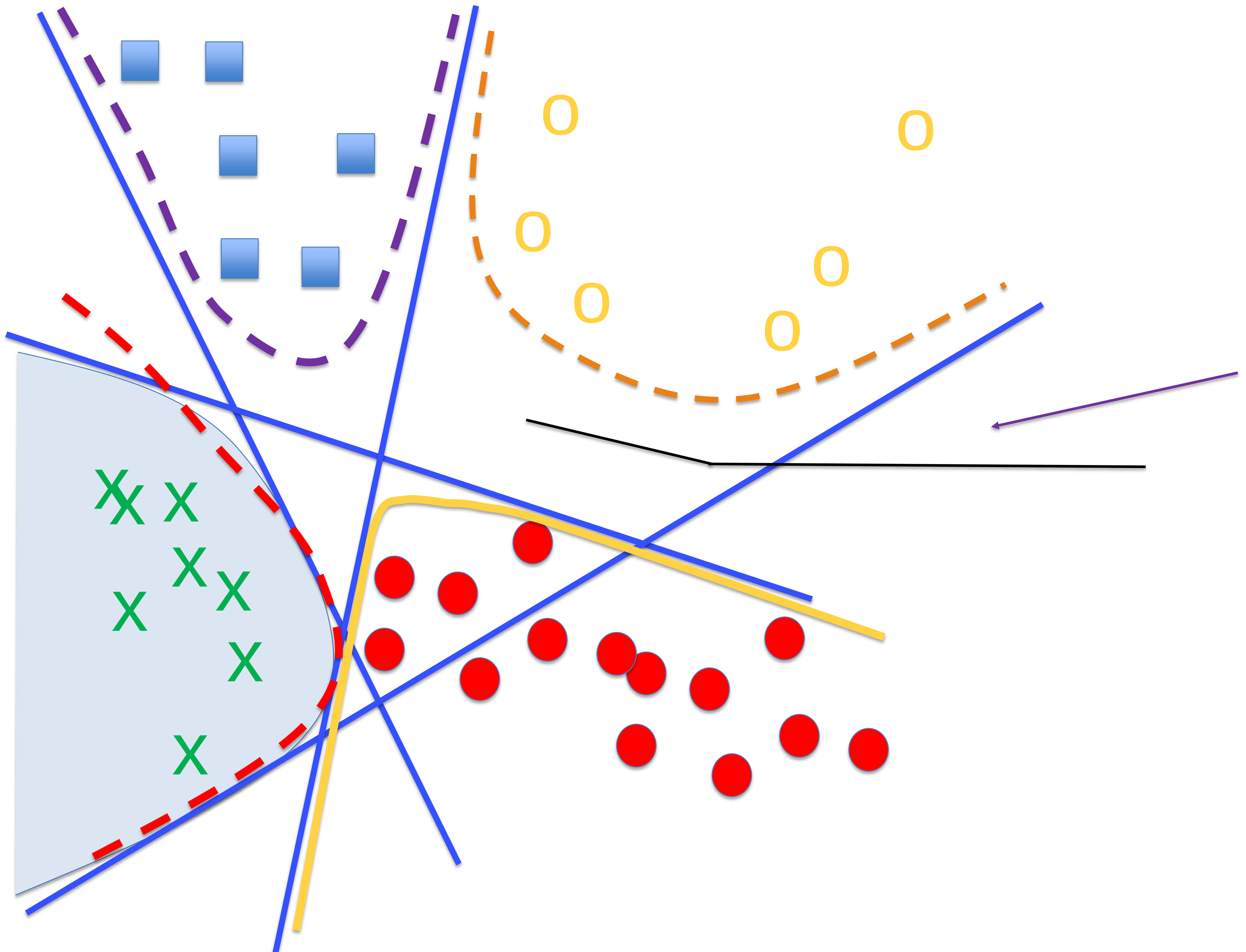
With polynomial number of hidden neurons:
→ Generalization

Previous slide.

Intuitively speaking, hyperplanes can be re-used to assign labels, because the configuration of XOR is rather uncommon in real-world problems.
An example is shown in the next slide

Distributed representation vs local representation

Example: nearest neighbor representation



Nearest neighbor
Does not create
A new region here

Previous slide.
Illustration of the re-use of regions, carved out by hyperplanes, for several classes.

An alternative method to hyperplanes would be nearest-neighbor classification. In this case the assignment to the orange and red classes would be extended, without carving out a new region.

Deep networks versus shallow networks

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

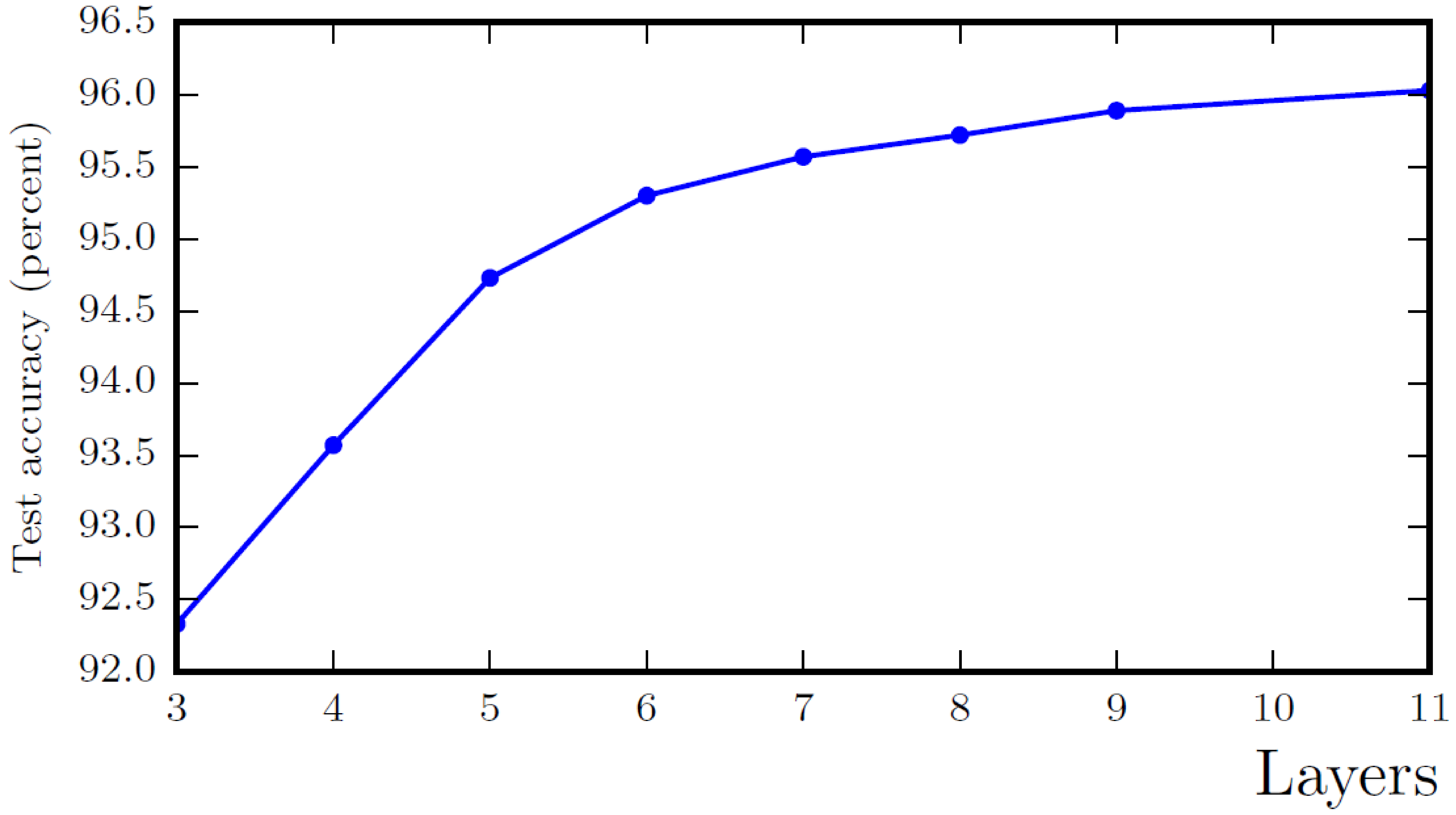


Image: Goodfellow et al., Deep Learning, MIT Press 2016

Previous slide.

Increasing the number of layers increases performance.

Deep networks versus shallow networks

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

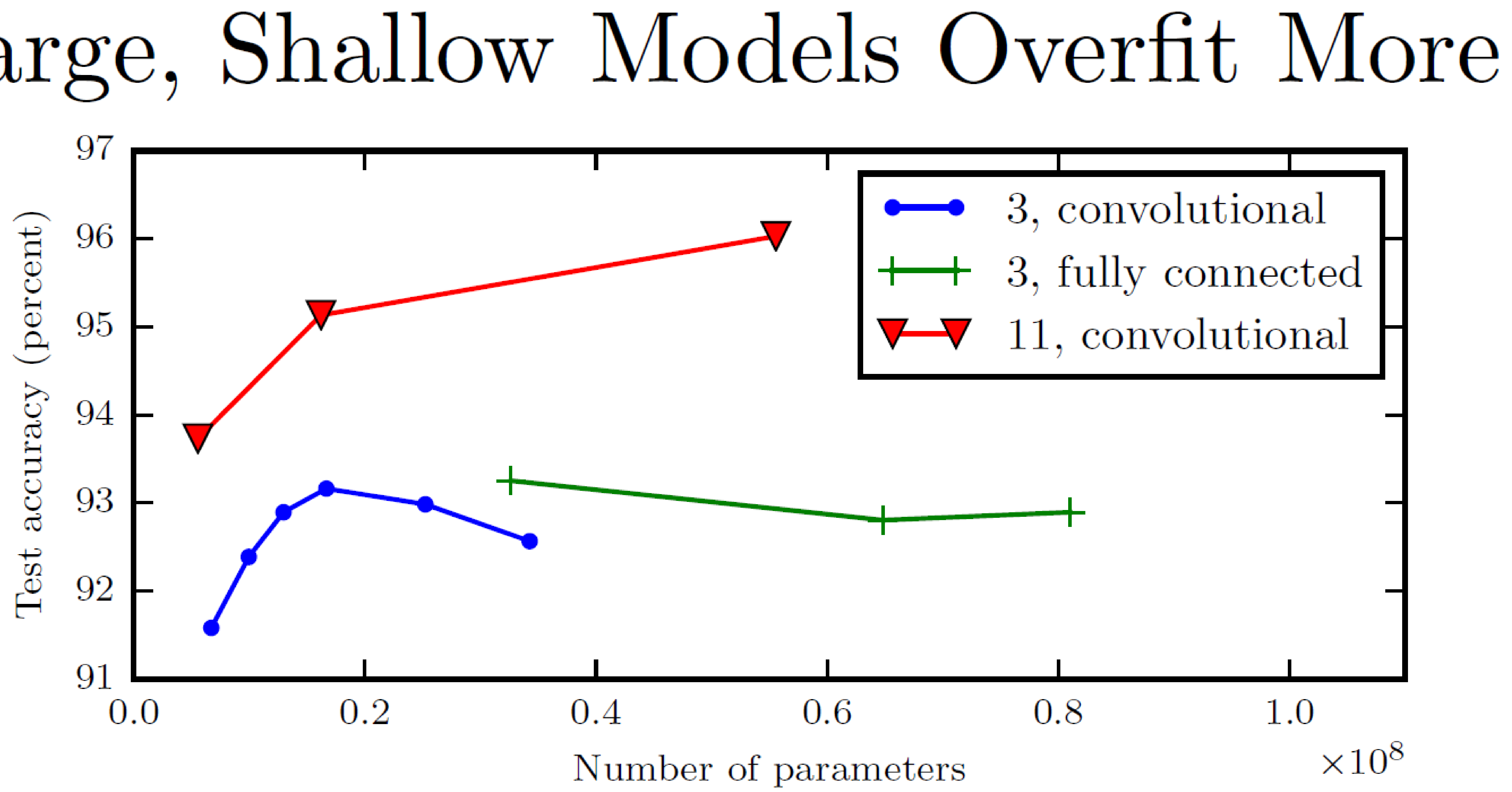


Image: Goodfellow et al., Deep Learning, MIT Press 2016

Previous slide.

For the same number of parameters (weights), a convolutional neural network with 11 layers performs better than a fully connected network with three layers.

For convolutional networks: see lecture ‘week 7’,

Conclusion: experimentally it was found that deep networks perform better than shallow ones.

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

Previous slide.

One potential (non-mathematical) explanation of the success of deep networks is the fact that features in the real world in which we are interested extend over large regions of the data space so that we have seen examples of green trees and green buses, but also red cars, red buses and white houses, we can generalize to red houses.

Artificial Neural Networks

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Loss landscape and optimization methods for deep learning

Objectives for today:

- Error function landscape:
  there are many equivalent 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 algorithm is better than others
- Deep Networks: are better than shallow ones on
  real-world problems due to feature sharing