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:

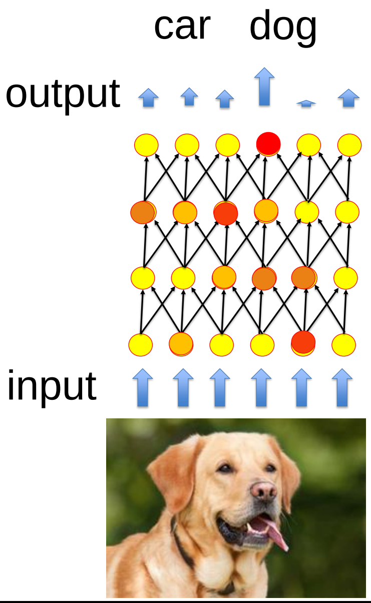
Goodfellow et al.,2016 *Deep Learning*

- 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

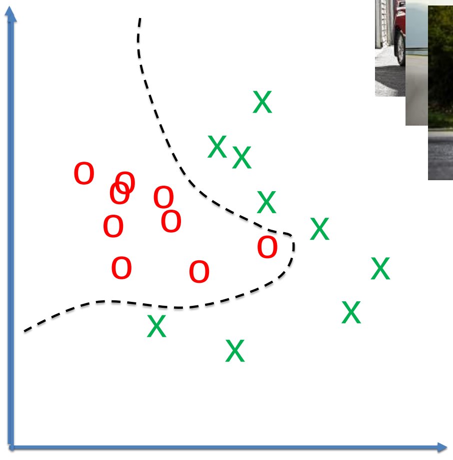
Aim of learning:
Adjust connections such that output is correct (for each input image, **even new ones**)



Previous slide.

A multilayer perceptron for classification

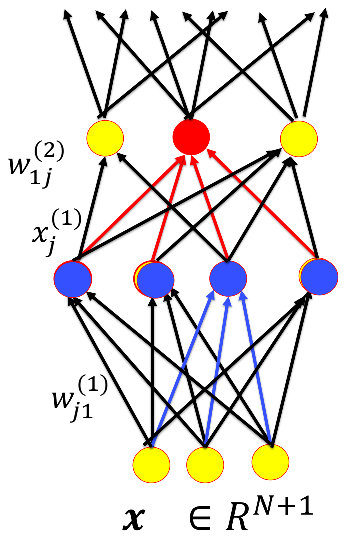
Review: Classification as a geometric problem

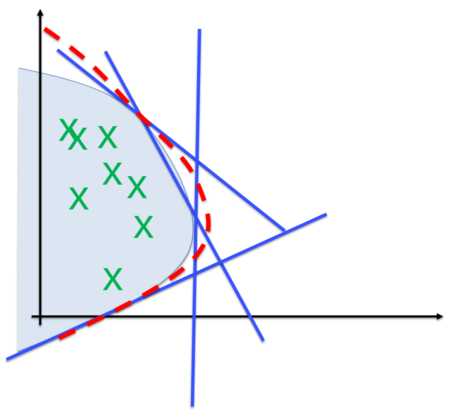


Previous slide.

… will implement a separating surface …

Review: task of hidden neurons (blue)

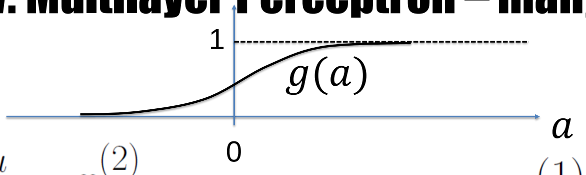




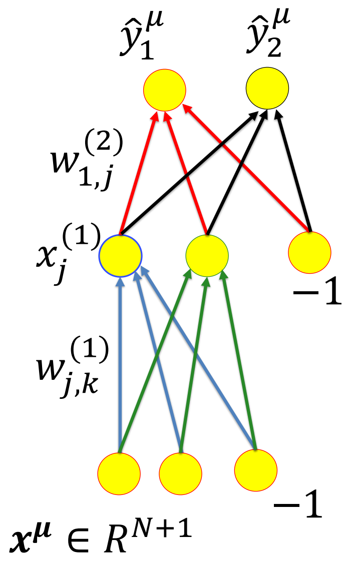
Previous slide.

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

Review: Multilayer Perceptron – many parameters



$$
\begin{aligned}
\hat{y}_i^\mu &= x_i^{(2)} && (1)\\
&= g^{(2)}[a_i^{(2)}] && (2)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} x_j^{(1)}] && (3)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(a_j^{(1)})] && (4)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(\sum_k w_{jk}^{(1)} x_k^\mu)] && (5)
\end{aligned}
$$



Previous slide.
The hyperplanes are defined by the weight vector

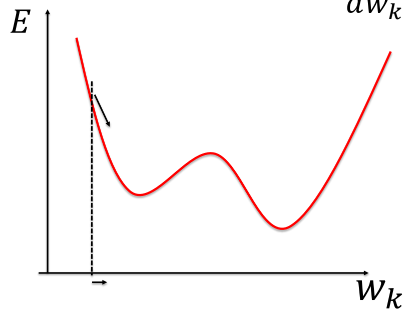
Review: gradient descent

Quadratic **error**

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

gradient descent

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



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

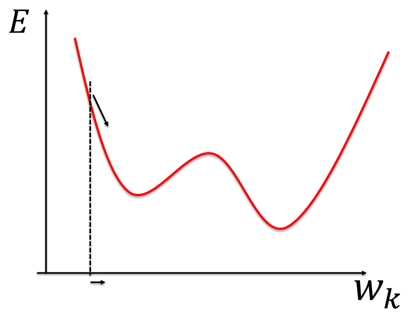
Previous slide.

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

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?



Previous slide.

We address three important questions today.

1. What is the shape of the error function, as a function of the weights?

2. How can we quickly find a good minimum?

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

Artificial Neural Networks: Lecture 5

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Error function and optimization methods for deep networks

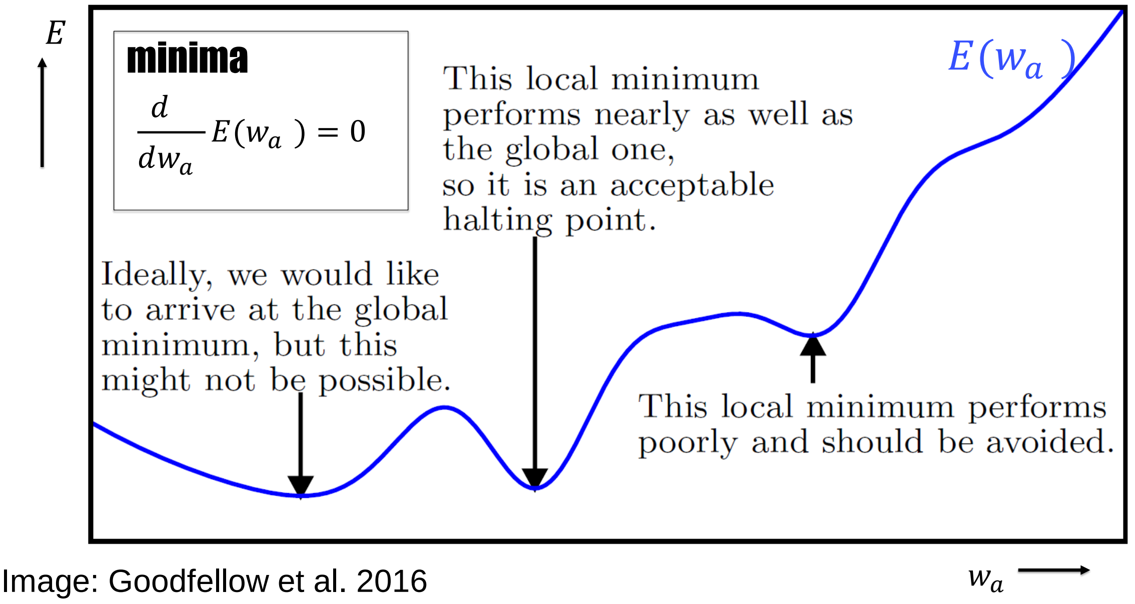
Objectives for today:

- Error function: minima and saddle points

Previous slide.

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

1. Error function: minima



E
minima
$\frac{d}{dw_a} E(w_a) = 0$
This local minimum performs nearly as well as the global one, so it is an acceptable halting point.
$E(w_a)$
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. 2016

Previous slide.

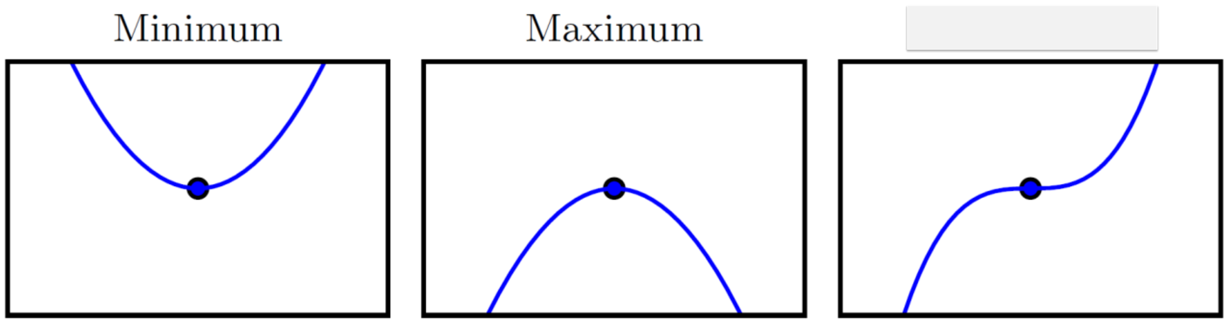
Often we see hand-drawn sketches of one-dimensional plots like the one here, with several local minima.

1. Error function: minima

How many minima are there in a deep network?

minima

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



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

Image: Goodfellow et al. 2016

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)

1. Error function: minima and saddle points

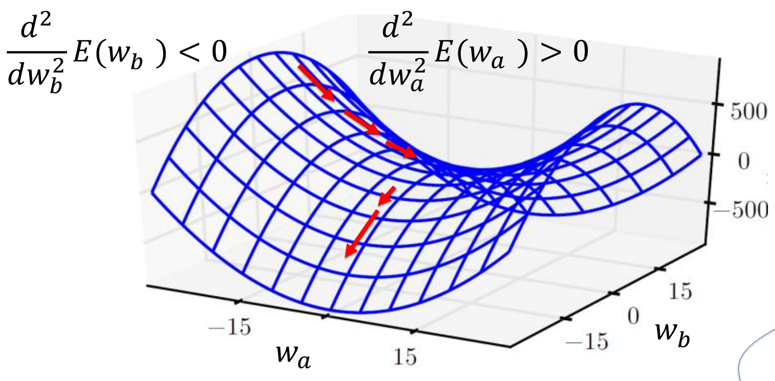
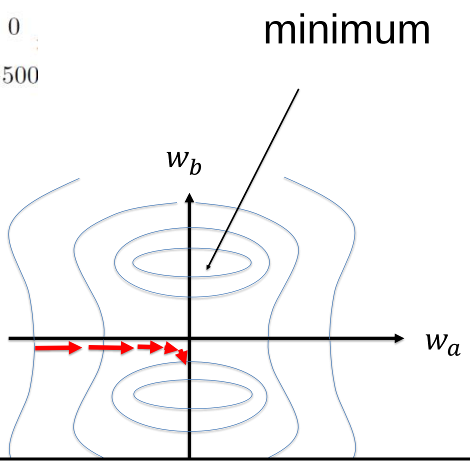


Image: Goodfellow et al. 2016

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

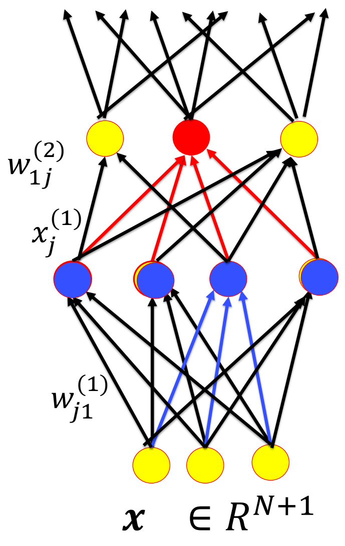
1. A deep neural network with 9 layers of 10 neurons each

[ ] 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.

1. Error function



How many minima are there?

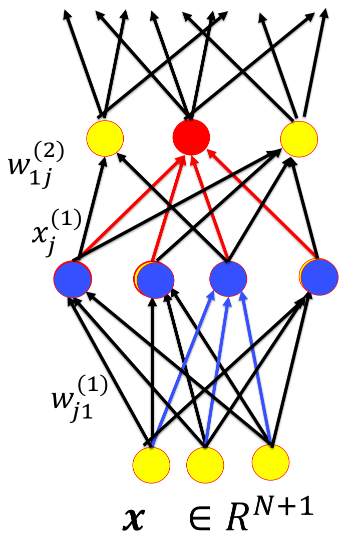
Answer:
In a network with n hidden layers and m neurons per hidden layer,

Previous slide.

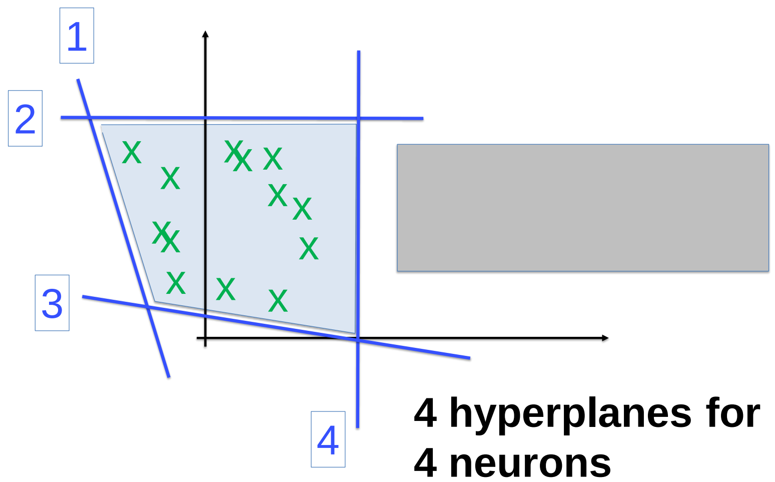
Because of the permutation symmetry, there are many equivalent minima.

(See Exercises)
.

1. Error function and weight space symmetry



many assignments of hyperplanes to neurons



4 hyperplanes for 4 neurons

Previous slide.

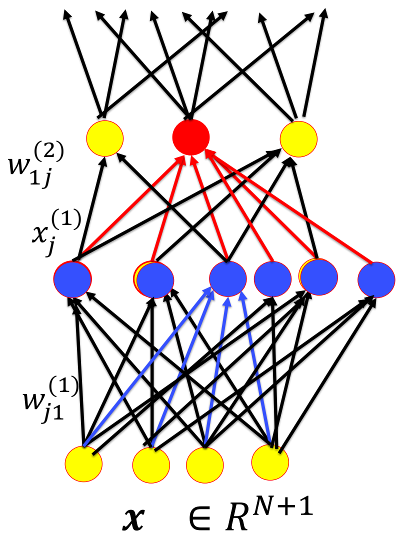
For example, with 4 neurons in a given layer, we have 4! different ways to implement the same 4 hyperplanes.

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

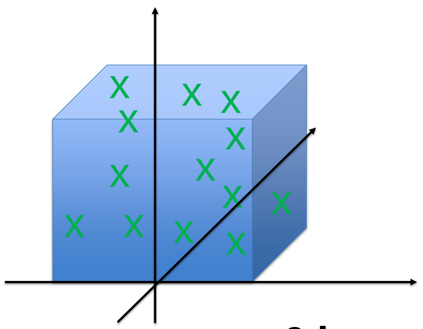
$$\boldsymbol{m}!^{\boldsymbol{n}}$$

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

1. Error function and weight space symmetry



many assignments
of hyperplanes to neurons

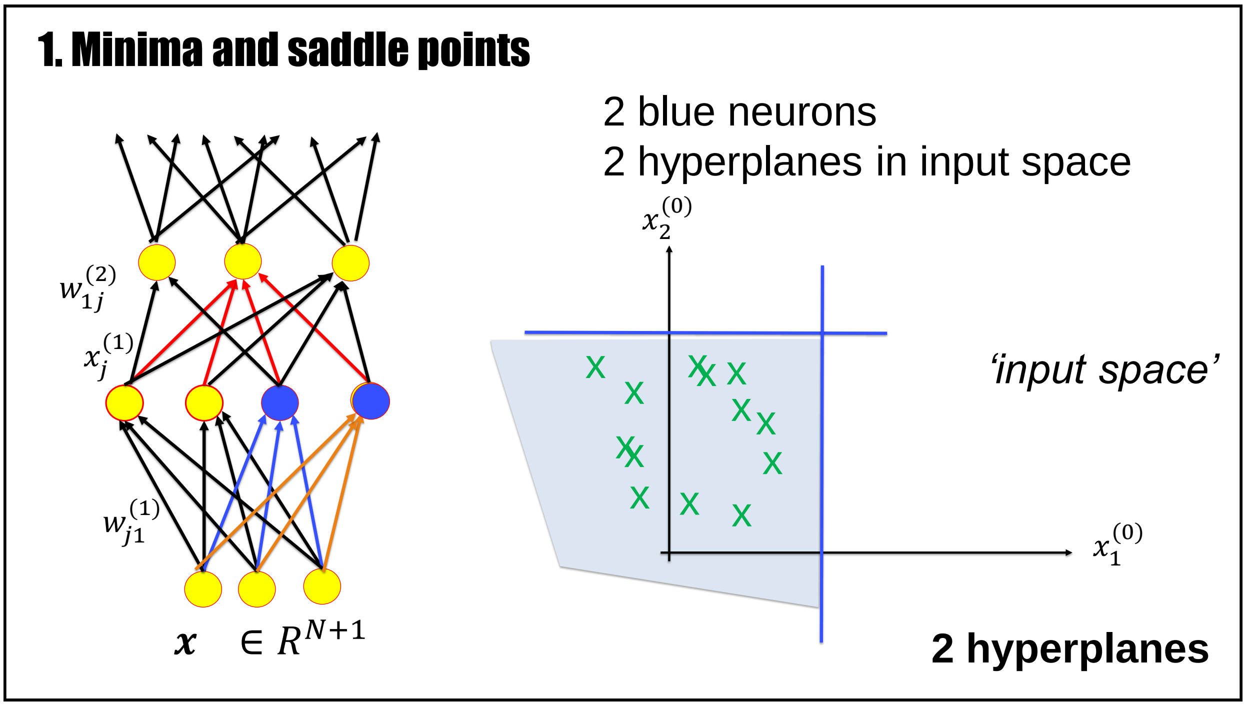


even more
permutations

**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.



1. Minima and saddle points

2 blue neurons

2 hyperplanes in input space

2 hyperplanes

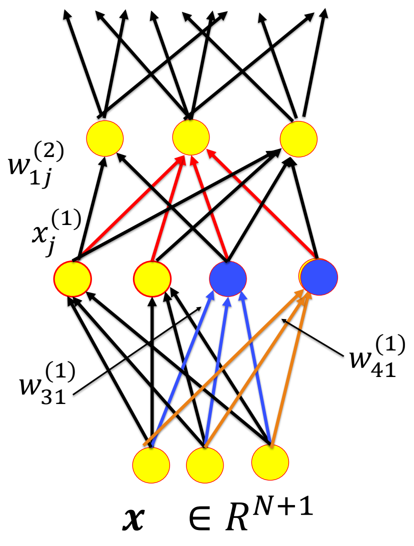
Previous slide.

Suppose that the two blue neurons implement two hyperplanes in the input space.
We now make an important conceptual transition from hyperplanes in the input space to weight vectors in the weight space.

1. Error function and weight space symmetry

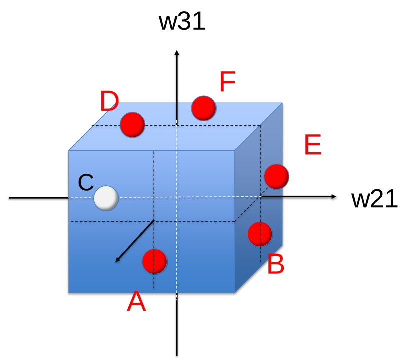
Blackboard 1

Solutions in weight space



Your notes.

1. Minima and saddle points in weight space



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

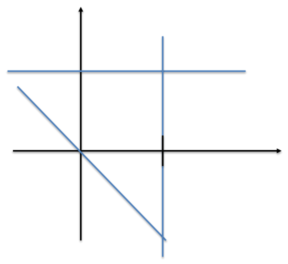
B = (0,1,-.7); D = (0,-.7,1)

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

Algo for plot:
- Pick w11,w21,w31
- Adjust other parameters to minimize E

Previous slide.

Imagine that in the input space you want to implement three hyperplanes, two of these parallel to the axes, and the third one diagonal.

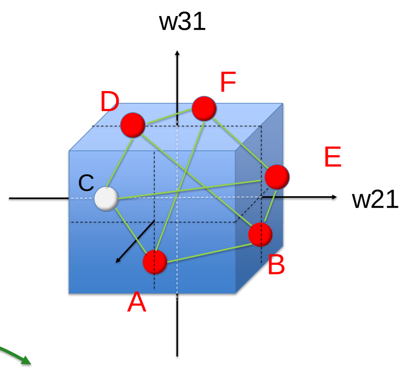


the weight vectors are:
$\boldsymbol{w_1} = (1,0)$ and threshold 1
$\boldsymbol{w_2} = (0,1)$ and threshold 1
$\boldsymbol{w_3} = (-1,-1)/\sqrt{2}$ and threshold 0

If we now plot just the FIRST component of each weight vector (previous slide), we have a first solution, labeled A.

But because of the perturbation symmetry, there are 5 other equivalent solutions, labeled B – F.
In the slide we approximate $1/\sqrt{2}$ = 0.7

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

Previous slide.

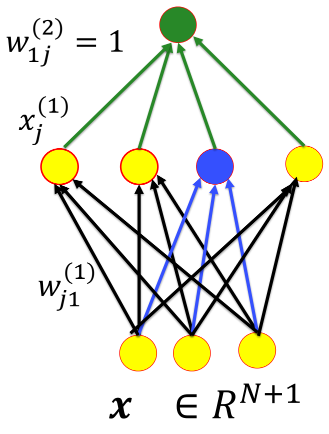
If I perform pairwise permutations of the weight vectors, I move on paths linking for example A with C, or A with F, etc.

On each of these paths, I expect a saddle point. Therefor I expect more than 6 saddle points, just from the symmetries in the weights space!

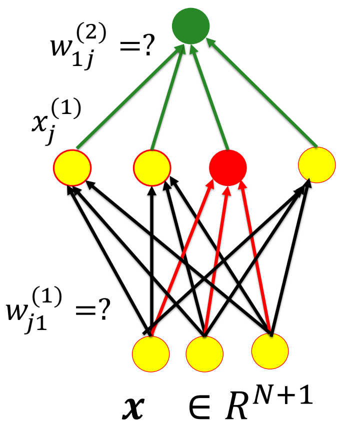
I do not give a specific number, because some permutations paths might cross through the same point close to the origin, so that they might actually be the 'same' saddle.

1. Minima and saddle points: Example

Teacher Network:
Committee machine

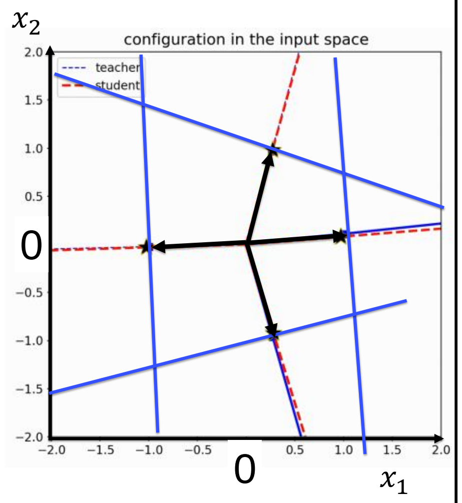


Student Network:



4 hyperplanes

'input space'



Previous slide.

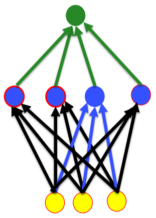
Data is generated from a teacher network (left).
Neurons in the first hidden layer implement hyperplanes (e.g., blue neuron).
The green neuron in the second layer sums up all contributions with equal weight. Such a configuration is called a committee machine ('all votes count equally').

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.

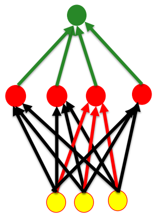
The student network has the same architecture, but freely adaptable weights in both layers.

1. Minima and saddle points

Teacher Network: Blue

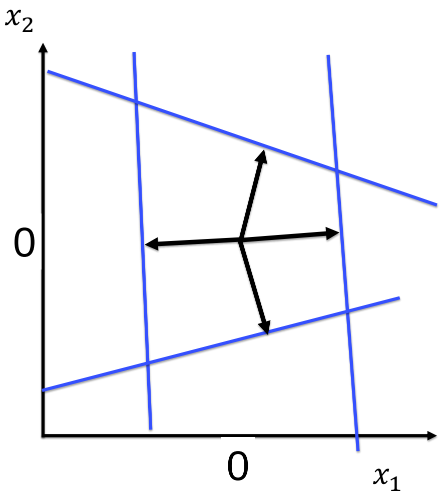


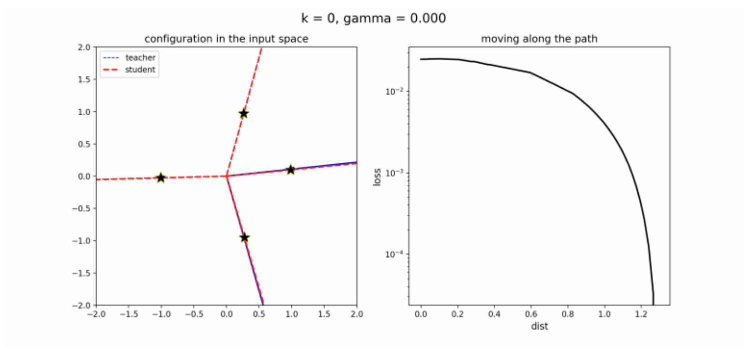
Student Network: Red



4 hyperplanes

'input space'



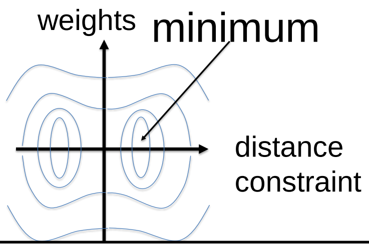


Previous slide.

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 labels of the two vectors can be exchanged at not cost. Thereafter we can relax back to the original position, but with exchanged labels of the vector.

The point at dist=0 is a saddle because all other weights have been minimized.



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)

Previous.

Permutation minima are connected by saddles. There are many more saddles than minima.

Some of the saddles are connected with each other. 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!

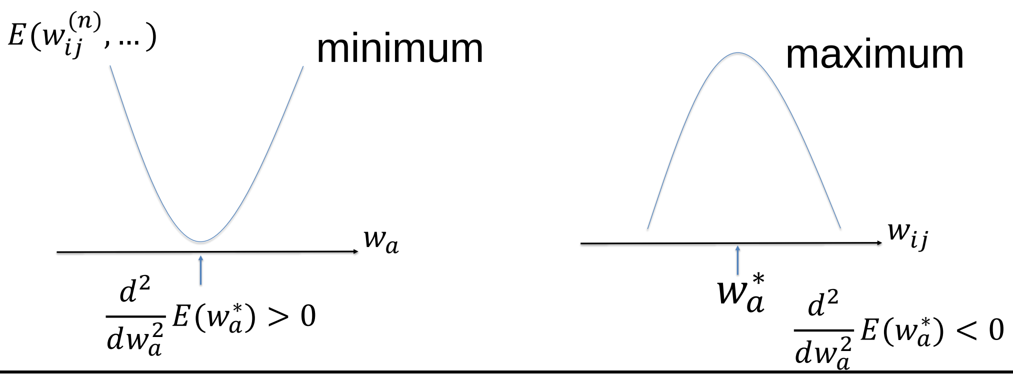
Thus the barrier of the saddle point between permutation minima is the lowest one of all possible pairs.

1. Minima and saddle points

There are many more saddle points than minima

Two arguments

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


$E(w_{ij}^{(n)}, \dots)$
minimum
w_a
$\frac{d^2}{dw_a^2} E(w_a^*) > 0$
maximum
w_{ij}
w_a^*
$\frac{d^2}{dw_a^2} E(w_a^*) < 0$

Previous slide.

There is also a completely different argument about the number and arrangement of saddles. It focuses on the Hessian matrix of second derivatives evaluated at the location where the first derivative vanishes.

1. Minima and saddle points

In 1dim: at a point with vanishing gradient

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

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.

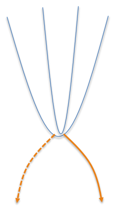
1. Minima and saddle points

in N dim: **Hessian**

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

Diagonalize:

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



$\lambda_1 > 0$

...

$\lambda_{N-1} > 0$

$\lambda_N < 0$

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.

1. Minima and saddle points

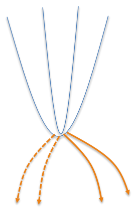
in N dim: **Hessian**

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

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!

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.

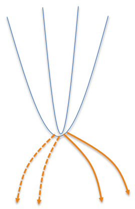
1. General saddle point

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

Previous slide.

Analogously, we define a general saddle.

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

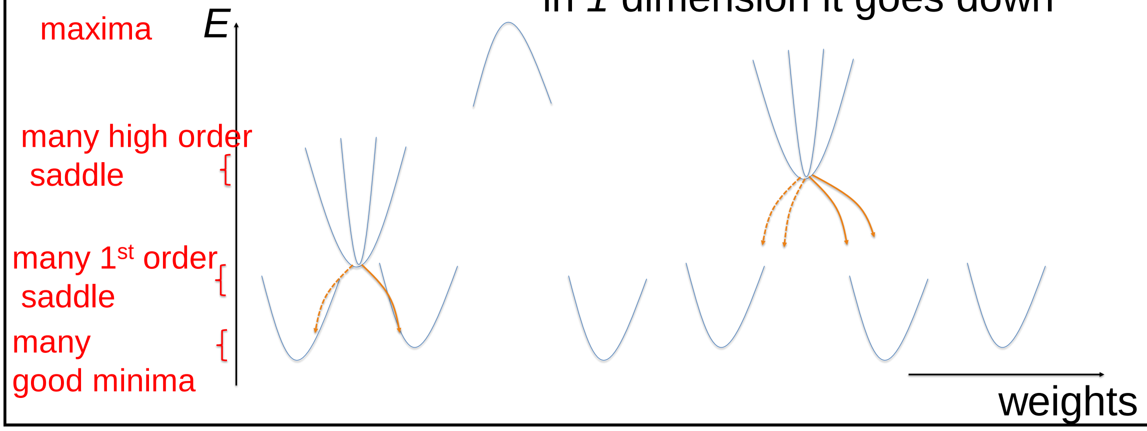
Previous slide.

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.

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



Previous slide.

Specific mathematical and physical models, linked to random matrix theory 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.

1. Minima and saddle points

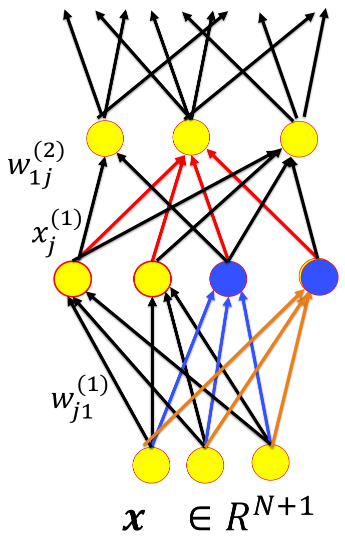
(ii) For balance random systems, eigenvalues will be randomly distributed with zero mean:
draw N random numbers
→ rare to have all positive or all negative
→ Rare to have maxima or minima
→ **Most points of vanishing gradient are saddle points**
→ **Most high-error saddle points have multiple directions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

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

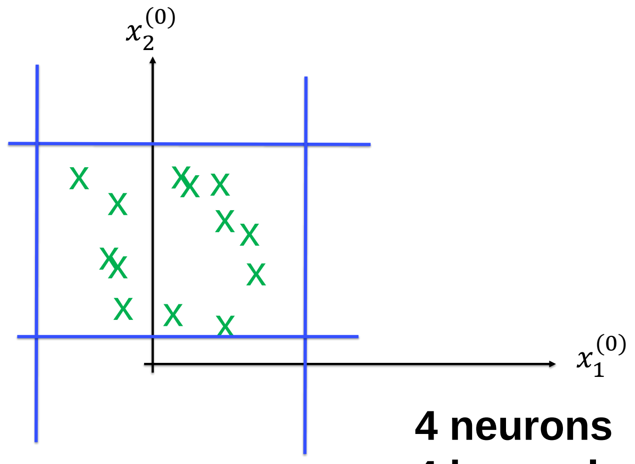
Previous slide.

For these random matrix or spin glass arguments, 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.

1. Minima = good solutions



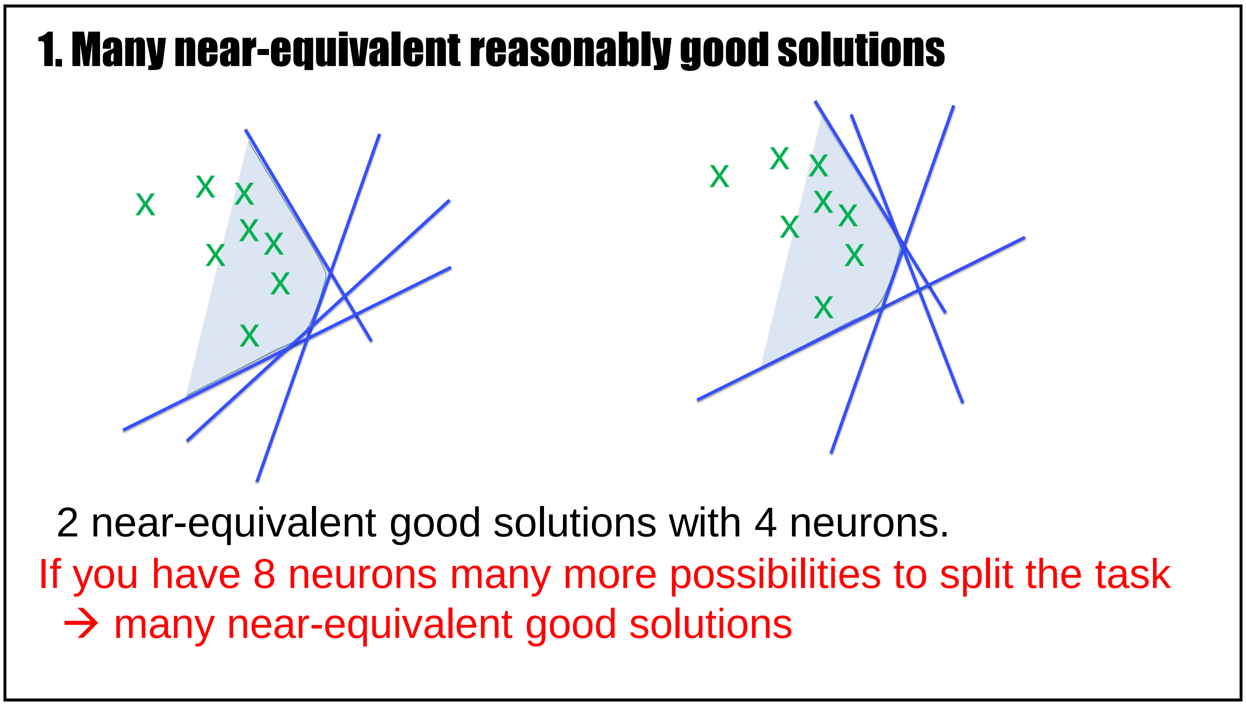
2 blue neurons
2 hyperplanes in input space



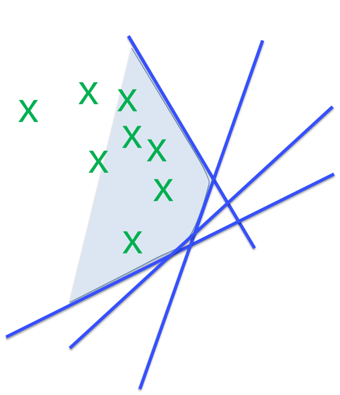
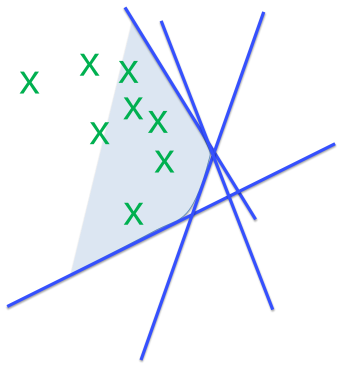
4 neurons
4 hyperplanes

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).



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

Previous slide.

However, real data is not generated from a teacher network of known architecture.
Therefore all solutions are approximate solutions.

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.

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

Your notes.

Artificial Neural Networks: Lecture 5

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Error function and optimization methods for deep networks

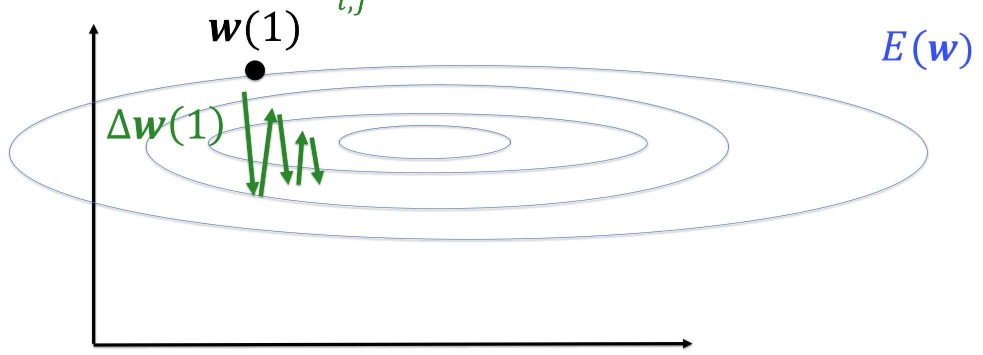
Objectives for today:

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

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.

2. Momentum: keep previous information

Blackboard2

In first time step: $m=1$

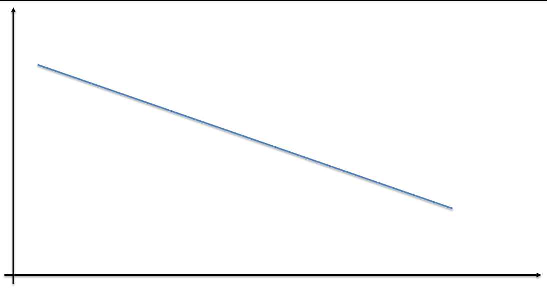
$$\Delta w_{i,j}^{(n)}(1) = -\gamma \frac{dE(\boldsymbol{w}(1))}{dw_{i,j}^{(n)}}$$

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 suppresses these oscillations while giving rise to a ‘speed-up’ in the directions where the gradient does not change

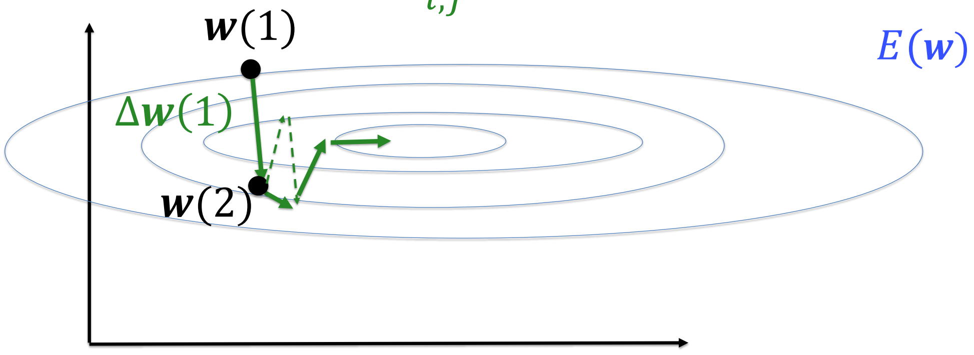


Blackboard2

Your notes.

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

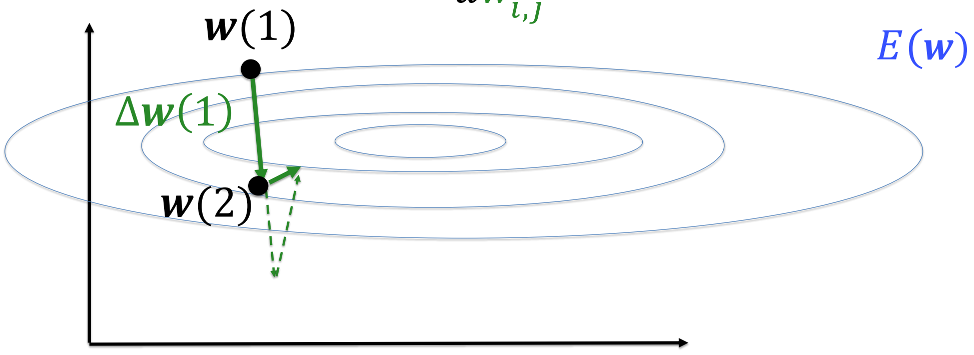
Previous slide.

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.

2. Nesterov Momentum (evaluate gradient at interim location)

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



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 $\boldsymbol{w}(\boldsymbol{n}+\boldsymbol{1})$, but at a hypothetical location

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

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 $\boldsymbol{w}(\boldsymbol{n}+\boldsymbol{1})$.

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.

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**

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

3. Error function: batch gradient descent

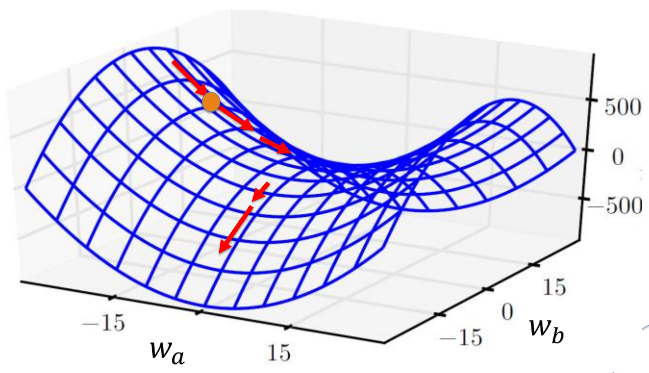
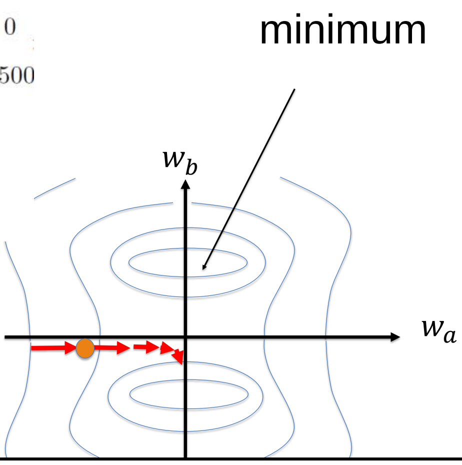


Image: Goodfellow et al. 2016

● $\boldsymbol{w}(1)$

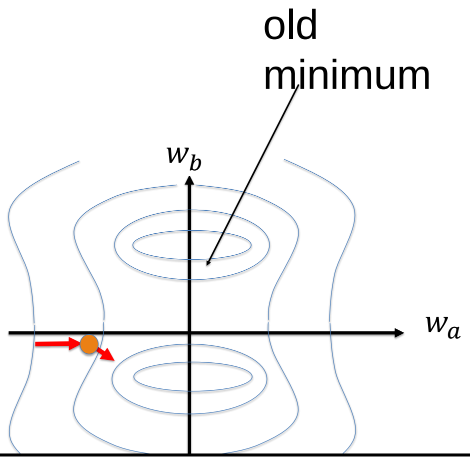


Previous slide.

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.

3. Error function: stochastic gradient descent

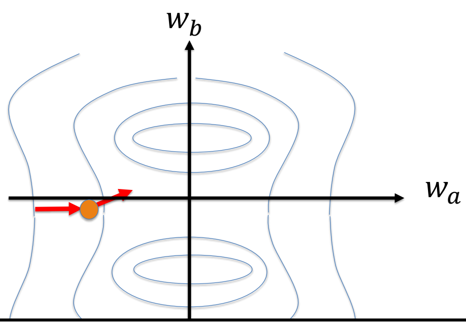
The error function for a small mini-batch
is not identical to the that of the true batch



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.

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



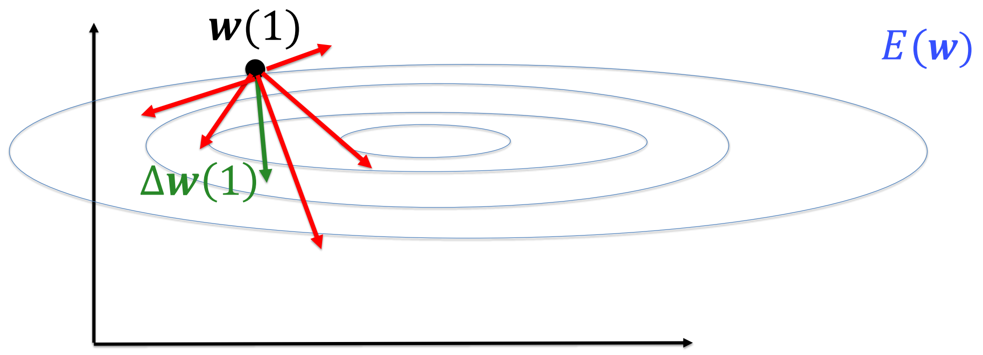
Previous slide.
Therefore, for the first minibatch the gradient would lead to the minimum with positive w_b , and for the second minibatch toward the minimum with negative w_b .

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/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/α examples (see Exercises).

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

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.

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

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

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. Stochastic gradient evaluation

Blackboard 3/Exerc. 1

Example:
consider 3 weights w_1, w_2, w_3

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

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:

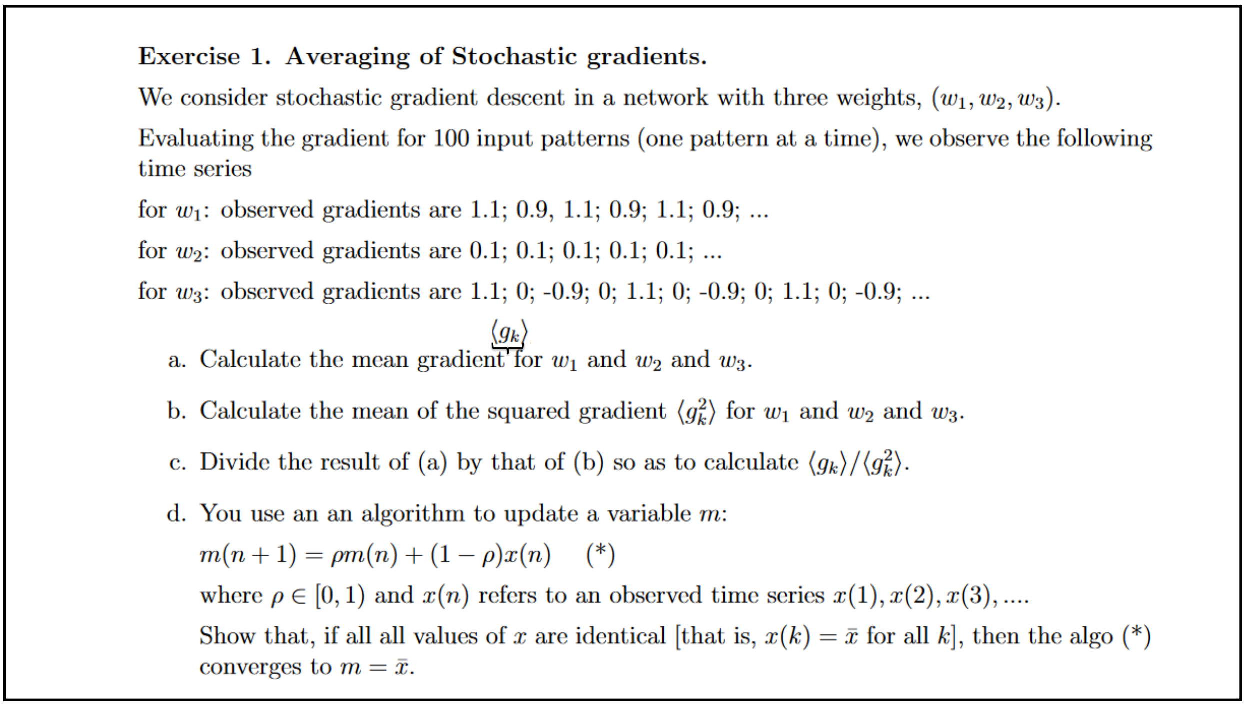
for w_1 : 1.1; 0.9; 1.1; 0.9; …

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

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



Exercise 1. Averaging of Stochastic gradients.

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; -0.9; 0; 1.1; 0; -0.9; 0; 1.1; 0; -0.9; …

a. Calculate the mean $\overset{\langle g_k \rangle}{\text{gradient for}}$ w_1 and w_2 and w_3.

b. Calculate the mean of the squared gradient $\langle g_k^2 \rangle$ for w_1 and w_2 and w_3.

c. Divide the result of (a) by that of (b) so as to calculate $\langle g_k \rangle / \langle g_k^2 \rangle$.

d. You use an an algorithm to update a variable m:

$m(n+1) = \rho m(n) + (1-\rho)x(n) \quad (*)$

where $\rho \in [0,1)$ and $x(n)$ refers to an observed time series $x(1), x(2), x(3), \ldots$.

Show that, if all all values of x are identical [that is, $x(k) = \bar{x}$ for all k], then the algo (*) converges to $m = \bar{x}$.

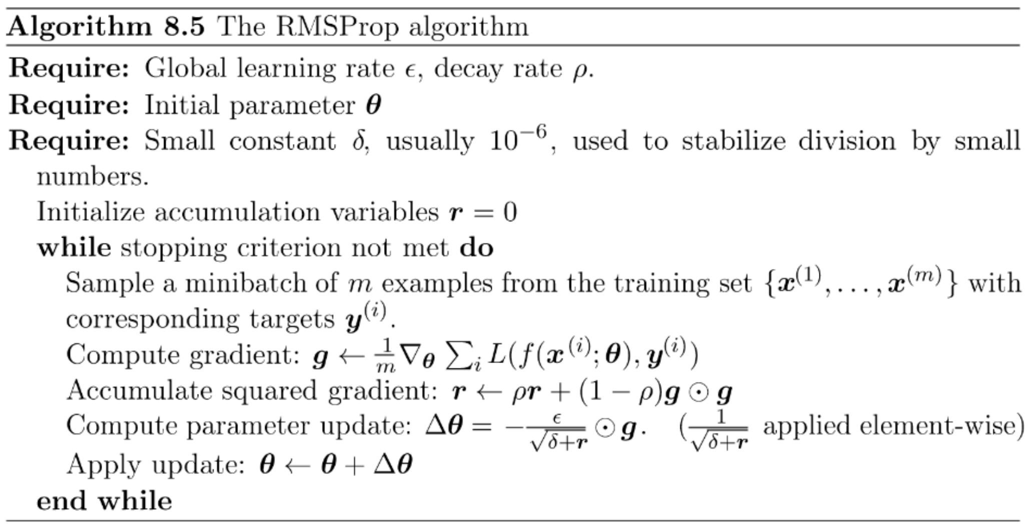
3. Adam and variants

The above ideas are at the core of several algos

- RMSprop
- RMSprop with momentum
- ADAM

Your notes on the exercise.

3. 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. 2016

Previous slide.

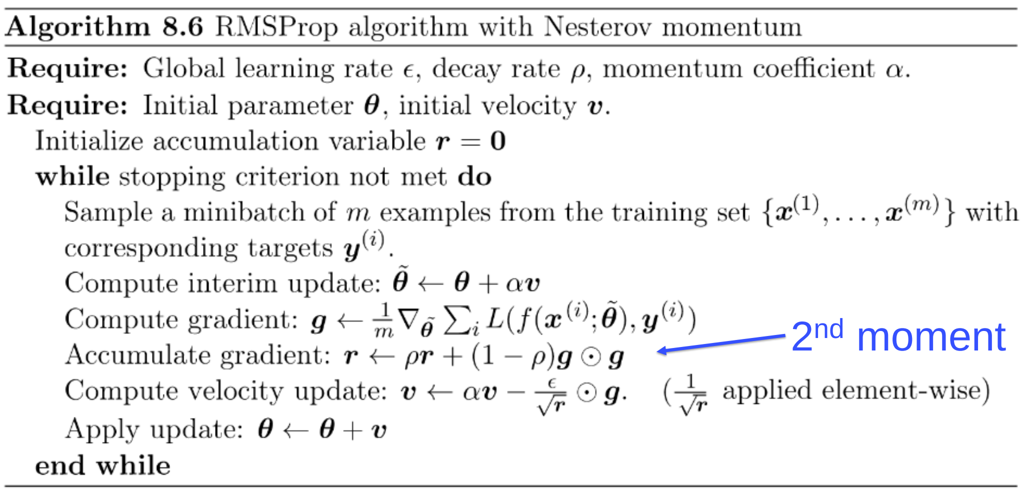
RMSprop algorithm.

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).

3. 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 $\boldsymbol{r} = \boldsymbol{0}$
  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: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho) \boldsymbol{g} \odot \boldsymbol{g}$
    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
```

2nd moment

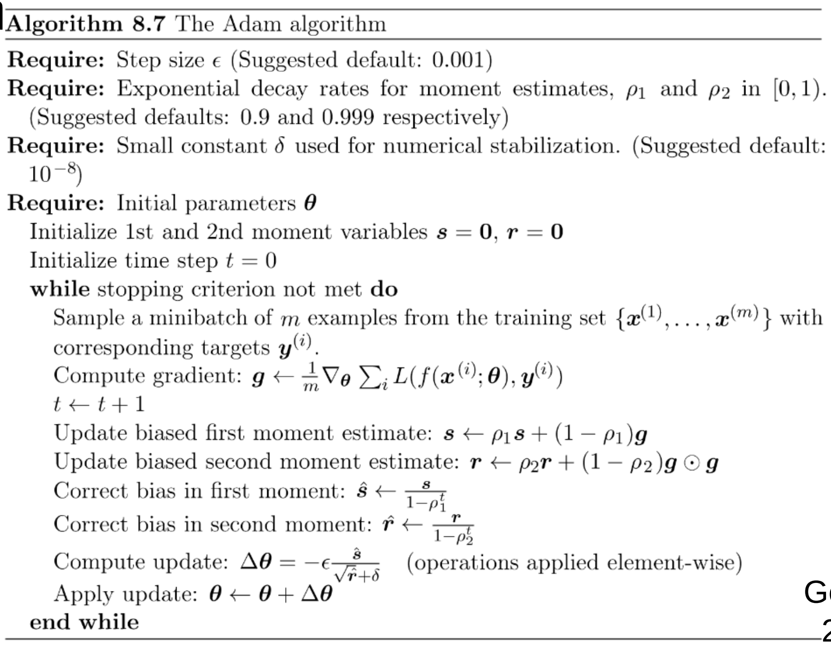
Goodfellow et al. 2016

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.

3. 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 $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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

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!

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)

Your notes.

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

Your notes.

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

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).

Artificial Neural Networks: Lecture 5

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Error function and optimization methods for deep networks

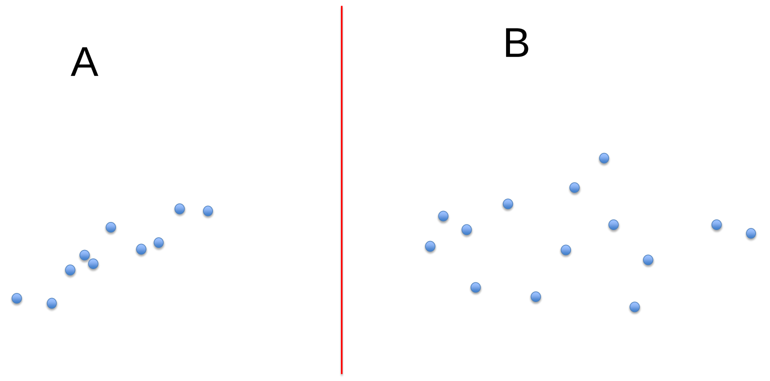
Objectives for today:

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

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?

4. No Free Lunch Theorem

Which data set looks more noisy?



Commitment:
Thumbs up

Which data set is easier to fit?

Commitment:
Thumbs down

Previous slide.

Let us start with two data sets.

4. No Free Lunch Theorem

Previous slide.

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

5. No Free Lunch Theorem

Your notes

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.

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.

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.

Previous slide.

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

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.

Your notes.

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.

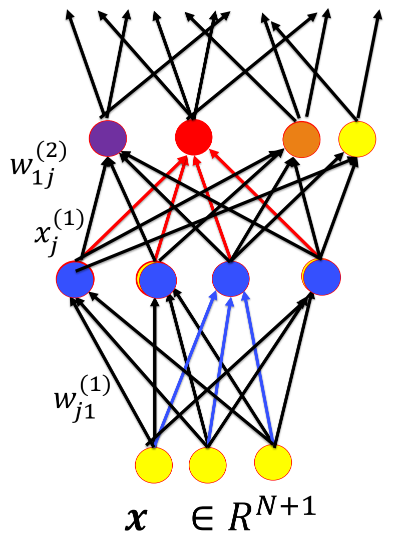
$\longrightarrow$ Always use prior knowledge if you have some

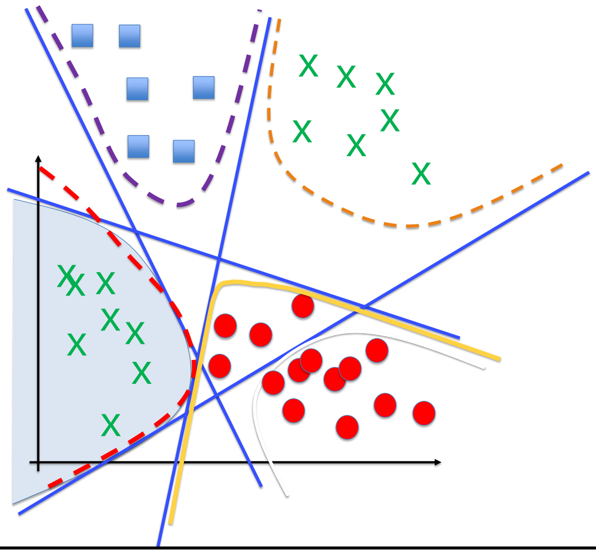
Previous slide.

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

4. 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.

4. Reuse of featuers in Deep Networks (schematic)

animals

birds

4 legs

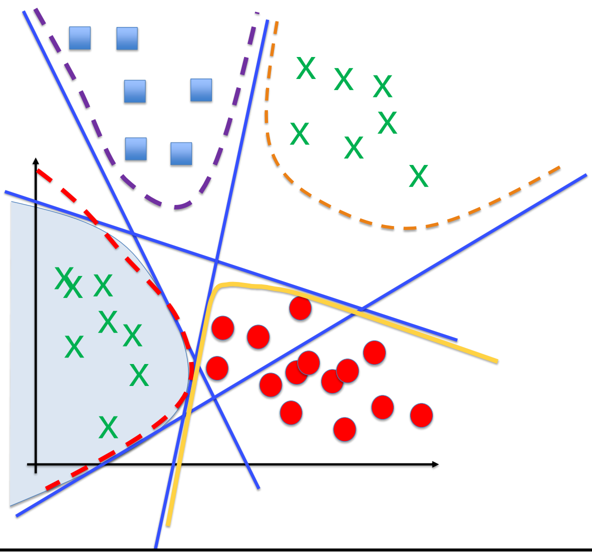
wings

snout

fur

eyes

tail



Previous slide.

A specific illustration of this idea is given here

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**

Previous slide.

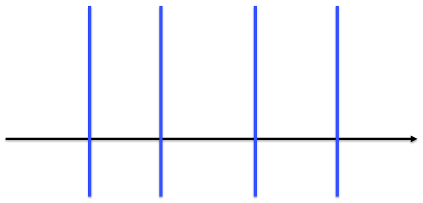
In the following we explore the idea of carving out regions in the space by hyperplanes.

5. Distributed representation

How many different regions are carved

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)

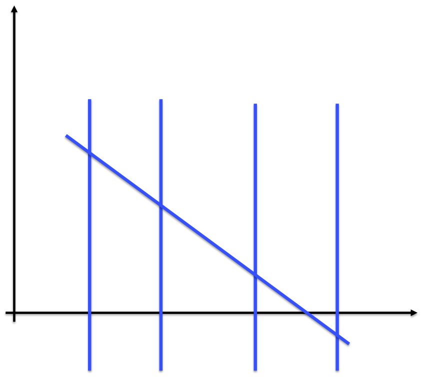
5. Distributed representation

How many different regions are carved

In 2dim input space with:

3 hyperplanes?
4 hyperplanes?

Increase dimension
= turn hyperplane
= new crossing
= new regions



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 next hyperplane. In this case n-2 new regions are created.

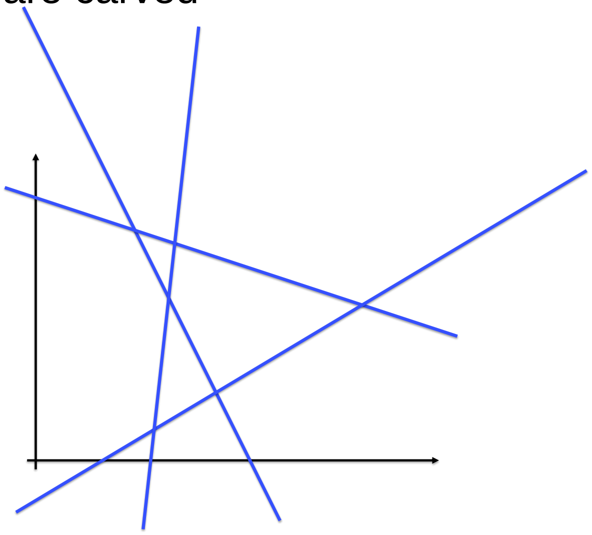
5. Distributed multi-region representation

How many different regions are carved

In 2dim input space by:

1 hyperplane
2 hyperplanes

3 hyperplanes?
4 hyperplanes?



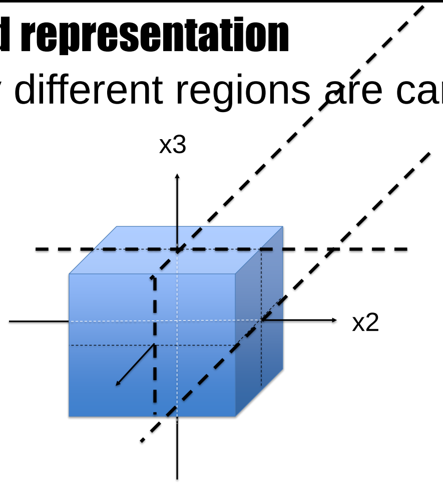
Previous slide.

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

5. 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?

5. Distributed multi-region representation

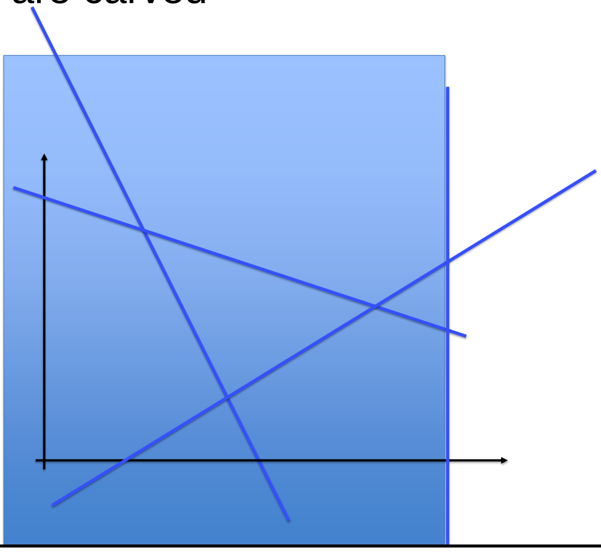
How many different regions are carved

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



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.

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

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.

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.

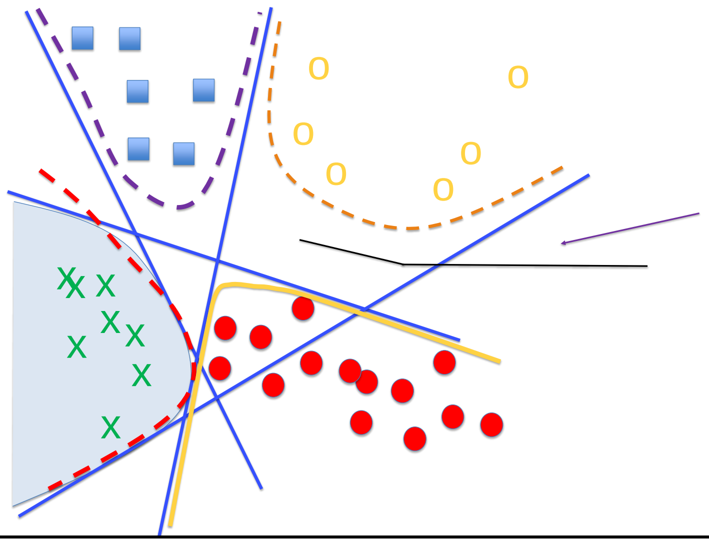
With polynomial number of hidden neurons:
→ classes are automatically assigned for many regions
where we have no labeled data
→ generalization

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

5. 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.

5. Deep networks versus shallow networks

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

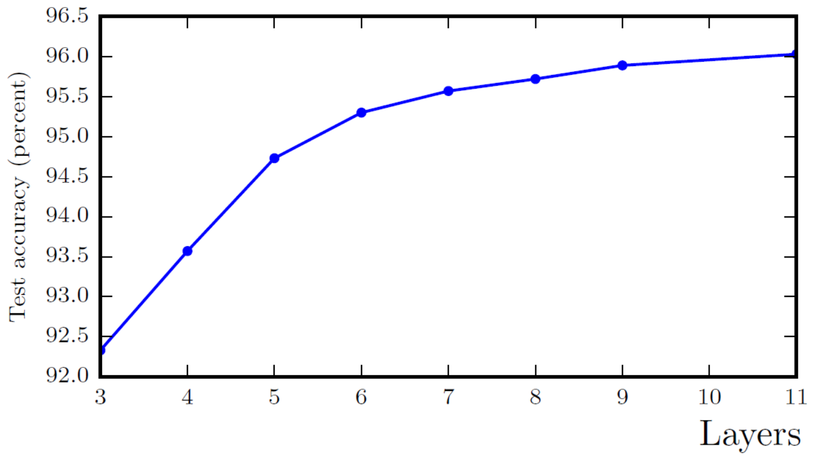


Image: Goodfellow et al. 2016

Previous slide.

Increasing the number of layers increases performance.

5. Deep networks versus shallow networks

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

Large, Shallow Models Overfit More

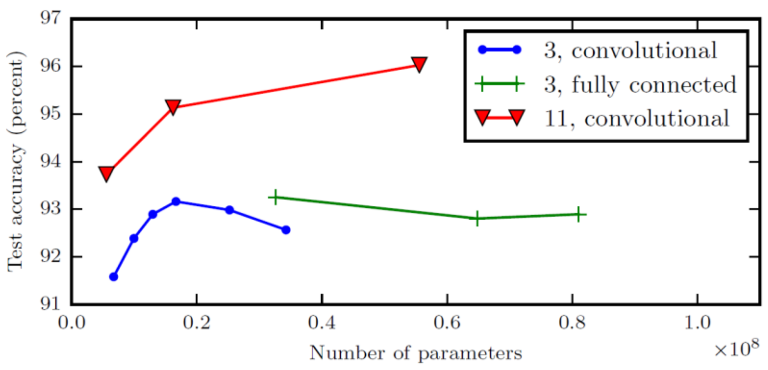


Image: Goodfellow et al. 2016

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.

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

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: 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

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Artificial Neural Networks: Lecture 5
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)

Previous slide.
This section provides a few complements to L2 and L1 regularization methods. It is generic and not limited to deep networks.

This topic has been treated in the class 'Machine Learning' by Profs Jaggi and Urbanke. Therefore it is not repeated during the lectures of the class 'Artificial Neural Networks' but simply added as an appendix in the slides.

Review: Regularization by a penalty term

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

$$\tilde{E}(\boldsymbol{w}) = \underbrace{E(\boldsymbol{w})}_{\text{Loss function}} \quad + \underbrace{\lambda \text{ penalty}}_{\text{assigns an 'error' to flexible solutions}}$$

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

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

Previous slide.

The penalty term acts like a smoothness constraint.

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

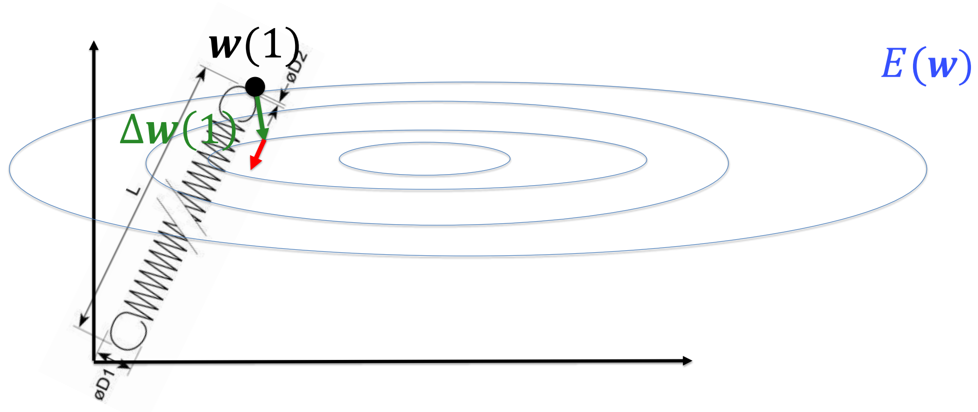
Previous slide.

L2 regularization refers to a penalty term that sums over all squared weights (but not the threshold parameters).

If we take the derivative, we find that the penalty term transforms into a decay term, hence the name weight decay.

4. L2 Regularization

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



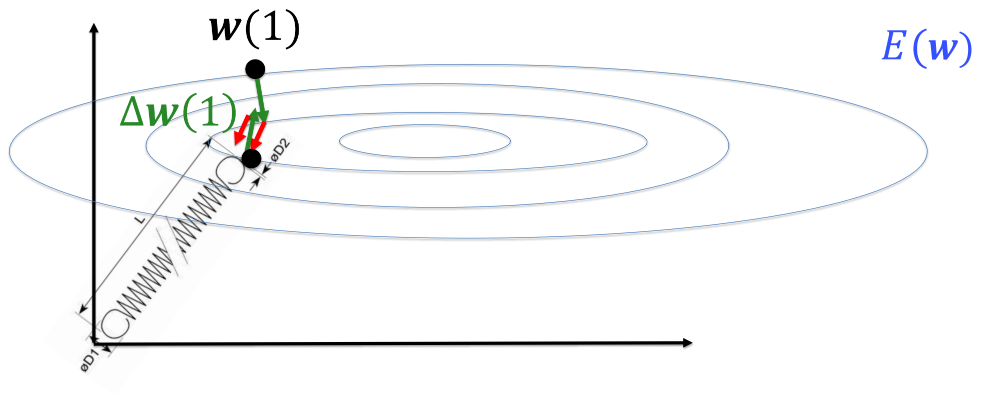
L2 penalty acts like a spring pulling toward origin

Previous slide.

We can interpret the weight decay term as a force that pulls toward the origin: it tries to keep weights small. At the same time the standard error function gives rise to a gradient term that pulls toward the minimum of the error. At equilibrium, the two forces balance each other.

4. L2 Regularization

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



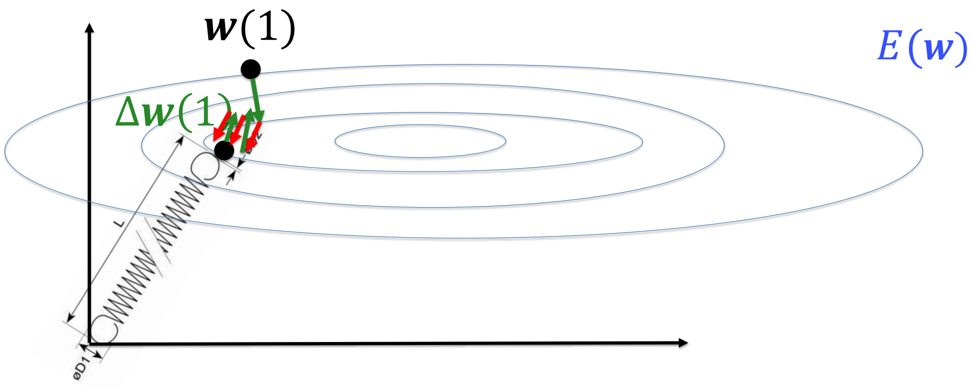
L2 penalty acts like a spring pulling toward origin

Previous slide.

The balance condition means that the size and direction of the red arrows (spring) and the green arrows (local gradient) cancel each other.

4. L2 Regularization

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



L2 penalty acts like a spring pulling toward origin

Previous slide.

Balanced position.

4. L1 Regularization

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

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

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

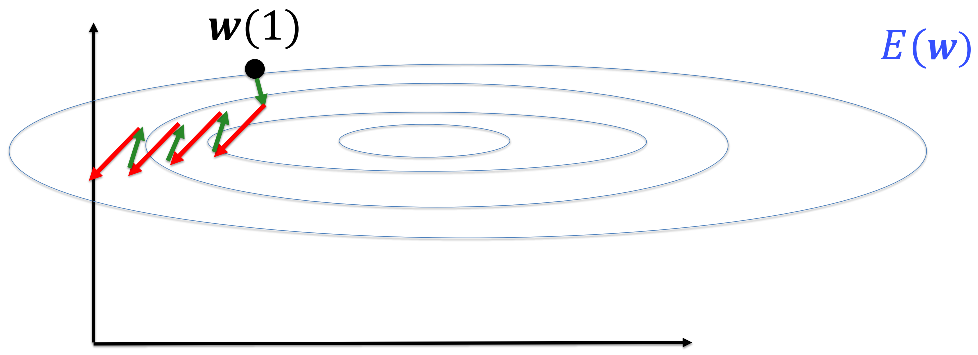
Previous slide.

Instead of L2 regularization, we can also work with L1 regularization.

4. L1 Regularization

Blackboard4

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



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

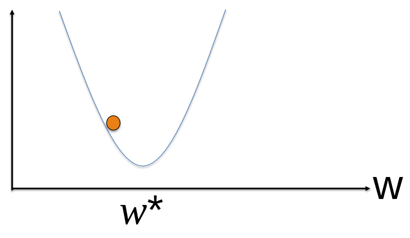
Previous slide.
If we take the derivative, we see that (for positive weights) the penalty cause a movement along a diagonal direction.

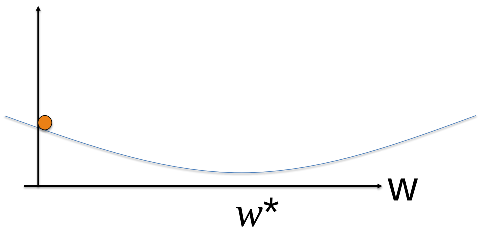
4. L1 Regularization

Blackboard4

Previous slide.

4. L1 Regularization (quadratic function)





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

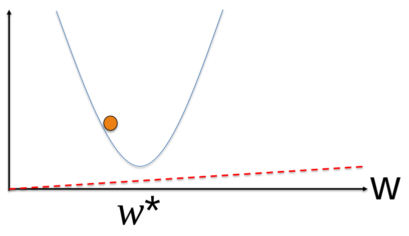
Small curvature β
OR big λ :
Solution at w=0

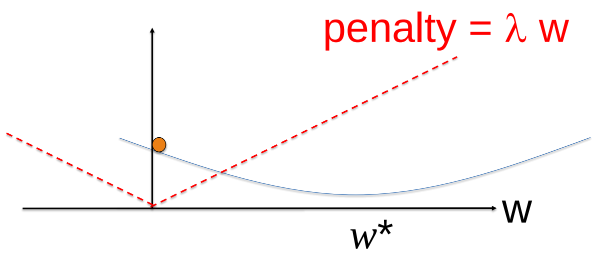
See also ML class of Jaggi-Urbanke

Previous slide.
The exact location of the combined miminum is either slightly shifted toward smaller weight values (left figure), or exactly at zero.

The L1 norm leads to a ‘sparse’ representation where many weights are zero.

4. L1 Regularization (general)





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

slope of E at w=0< slope of penalty
$\rightarrow$solution at w=0

See also ML class of Jaggi-Urbanke

Previous slide.
Sparsity is a generic result for L1 regularization.

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)

Previous slide.

In the exercises we will see that L2 regularization shares features with early stopping.