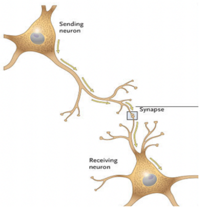
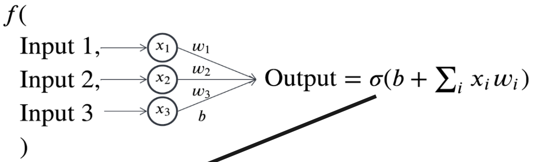
What expects us today

- Neural network training - 2h
- 15min break
- Tour of modern neural networks 1:45

Quick recap



https://teachmephysiology.com/nervous-system/synapses/synaptic-transmission/



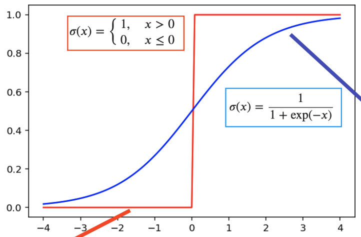
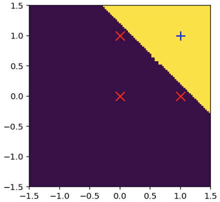
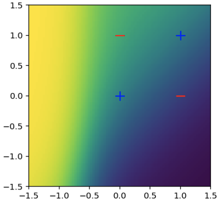


Figure 59



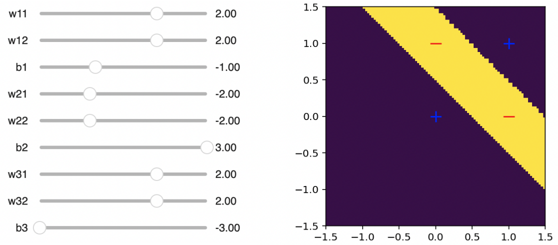


Let's get things moving

Training a NN

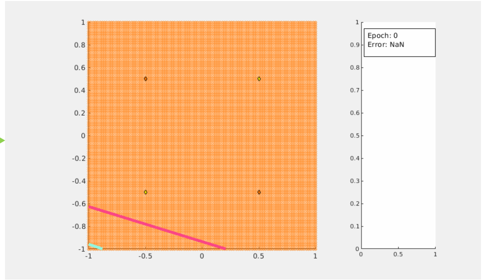
Coming up with the correct weights for the XOR problem is not an easy task: Requires to explore 9 grid values simultaneously!

Manual trial and error



Training a NN is a process in which we **learn** this parameters in a way that "best fits" the data. We learn the parameters with a **training dataset**.

Automated NN training



Training a NN

The training dataset has instances of **features** and **responses**. For example, the XOR problem has the following training dataset:

For features (x1,x2)=(1,0) we have a 0 response

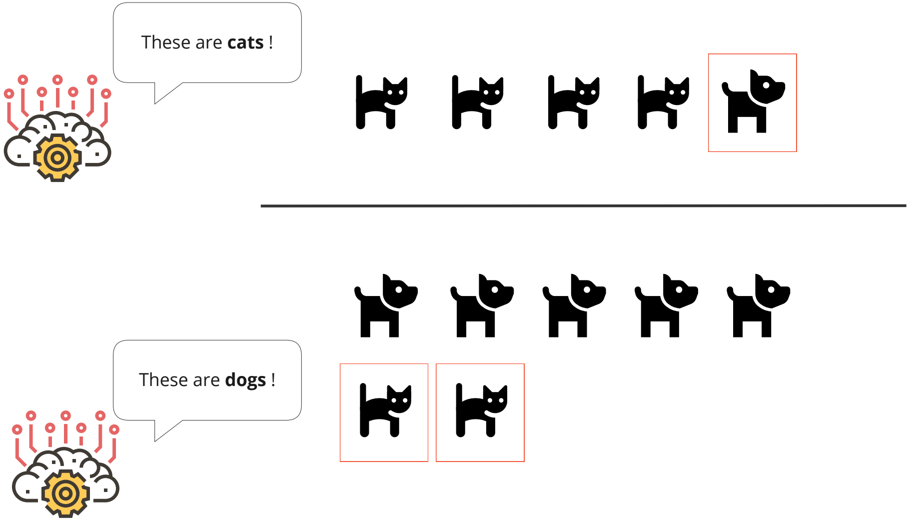
INPUTS

GROUND TRUTH = DESIRED OUTPUT

Value for x1	Value for x2	Class = XOR(x1,x2)
1	1	1
1	0	0
0	1	0
0	0	1

The training dataset is crucial in learning which parameters from the NN actually follow the response variable

Network error (loss)



Out of **12** animals, our network classified **3 incorrectly.**

Our network has an error of

$$\frac{3}{12} = 25\ \%$$

Other error measures

Mean Squared Error (MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2.$$

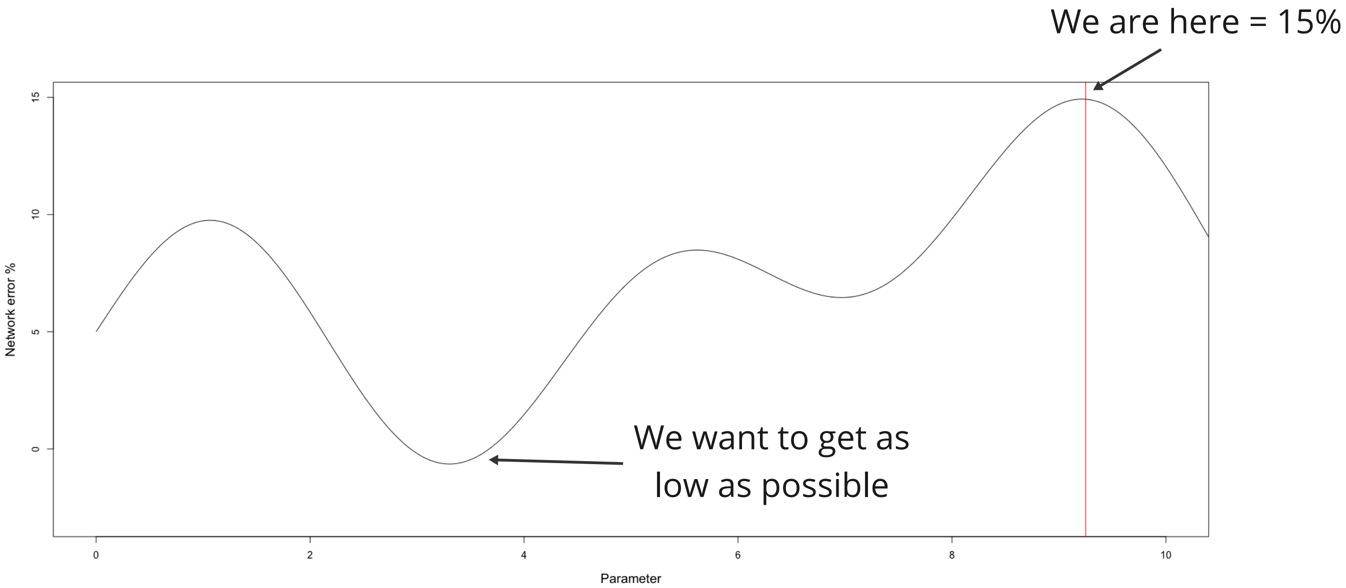
Used in regression problems

Cross-entropy

$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i)$$

Used in classification problems

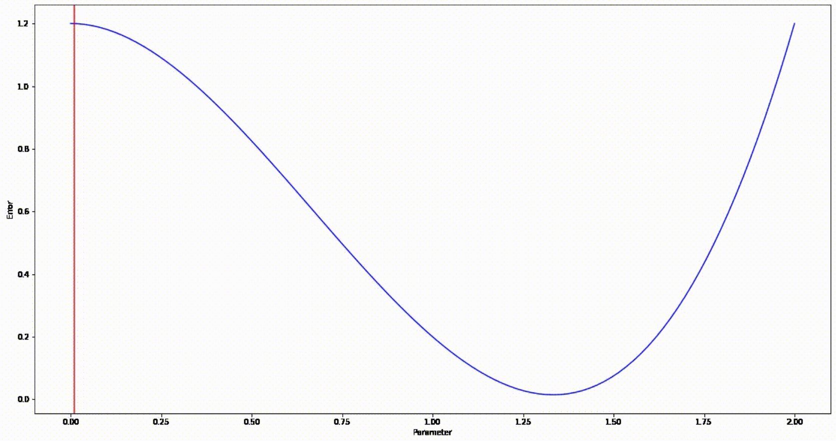
Error changes with parameters



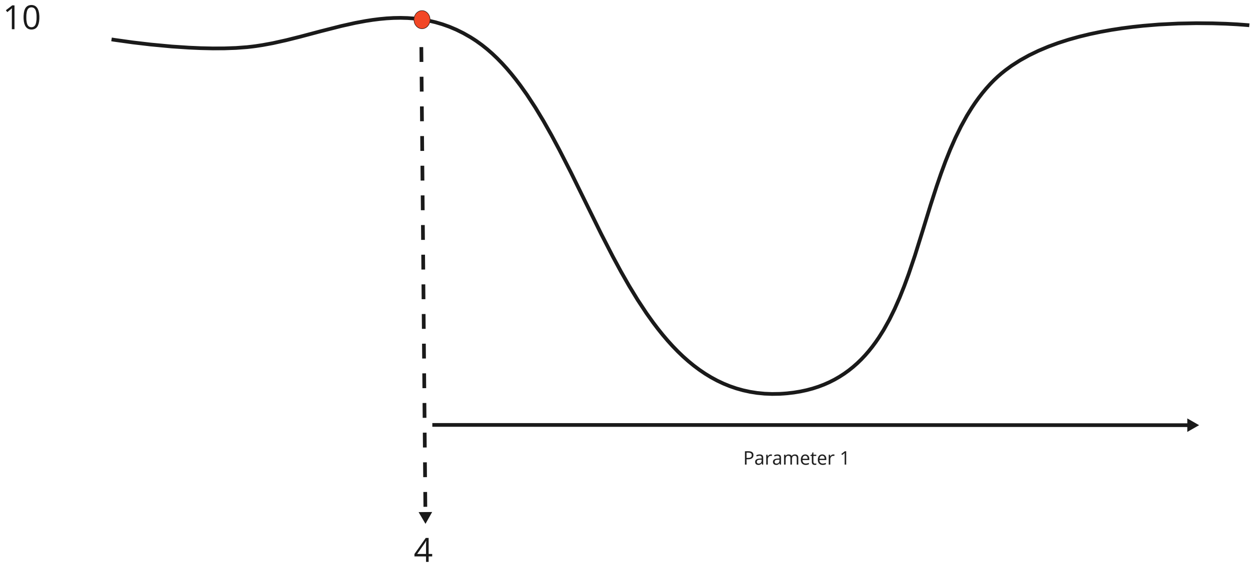
Error changes with parameter values = error is a **function** of parameters

Training strategy: Descent

- We start with initial random value
- Update the value in steps (descent)
- Size of the step changes



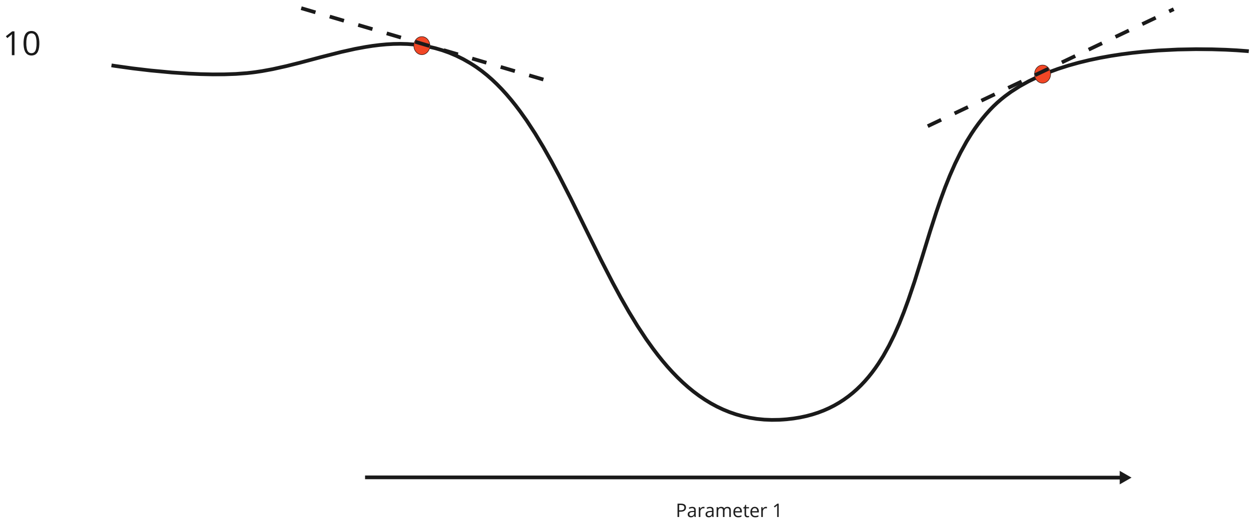
How do you know whether to go to left or to right ?



We are at parameter value 4
The error is 10

The computer only sees this !

Could a tangent help ?



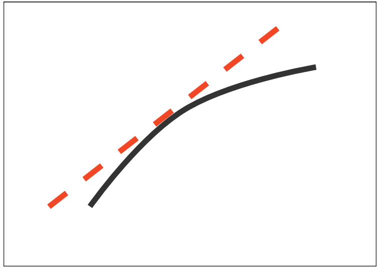
If we could measure the angle of a tangent - we could be able to tell where to go next !

It turns out we can - this is exactly what is called **derivative.**

Descent using derivatives

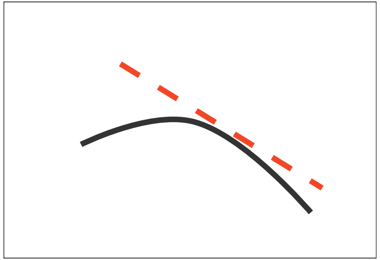
Derivative for a given parameter value is a **single number** that can be

Positive derivative



Tangent pointing upwards
Function is increasing

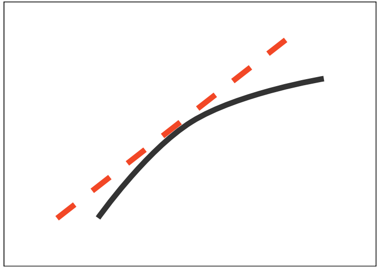
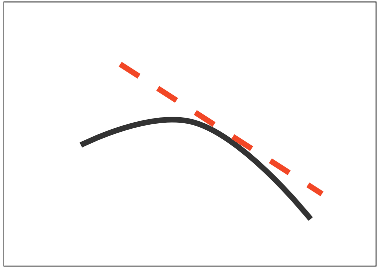
Negative derivative



Tangent pointing downwards
Function is decreasing

Value of the derivative is measuring **how much** is the error increasing/decreasing when we increase the value of parameter by one.

Descent using derivatives

Positive derivative	**Negative derivative**
	

Remember, we are minimizing error !

Derivative sign:	+	-
Parameter increase: consequence:	**Error increases**	**Error decreases**
What to do:	**Decrease parameter**	**Increase parameter**

We are doing the opposite

Gradient Descent

```
w = random()

for i in 1,2,..., MAX_ITERATIONS:
    dw = derivative_of_error(w)
    w = w - a*dw

return w
```

We stop after fixed amount of iterations

Learning rate

Modifications

- we stop when error is no longer decreasing for some time = it **converged**
- we measure error on both, training and testing set and stop when test error starts to increase

How am I supposed to calculate derivatives ?

Short answer: "You don't".

Long answer:

a) You can use mathematical formulas

Works only for specific networks

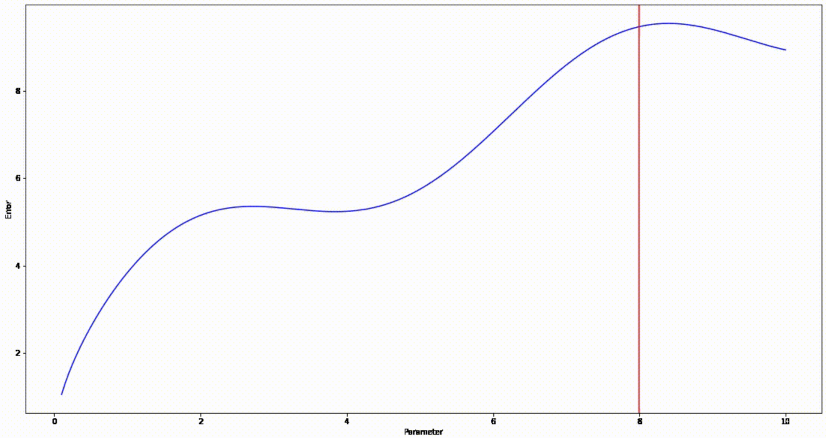
b) You can let deep learning framework do this for you !

TensorFlow

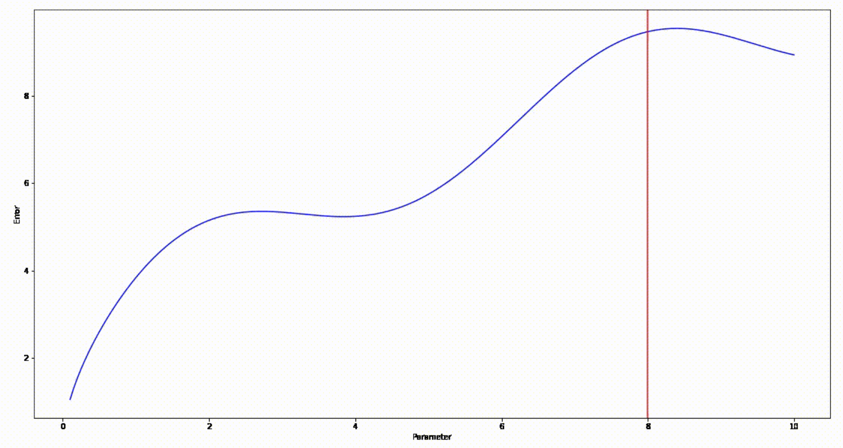
PyTorch

Works for all networks !

Gradient Descent - problem



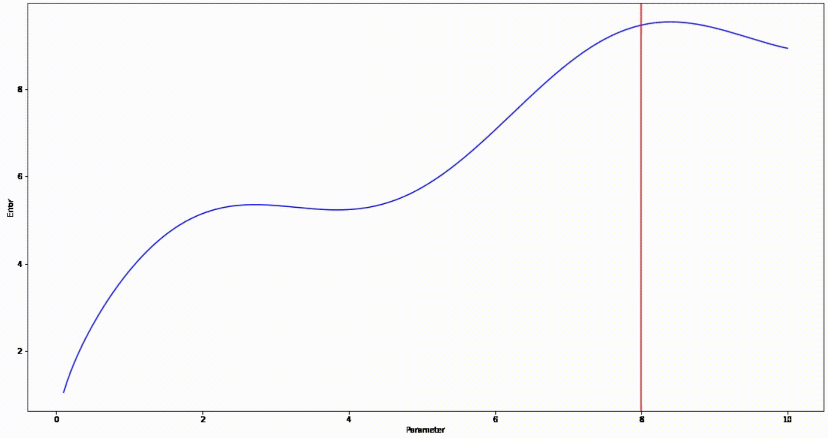
Gradient Descent - problem



Gradient descent can get stuck in **local minimum** !

How to solve this ?

Gradient Descent - problem

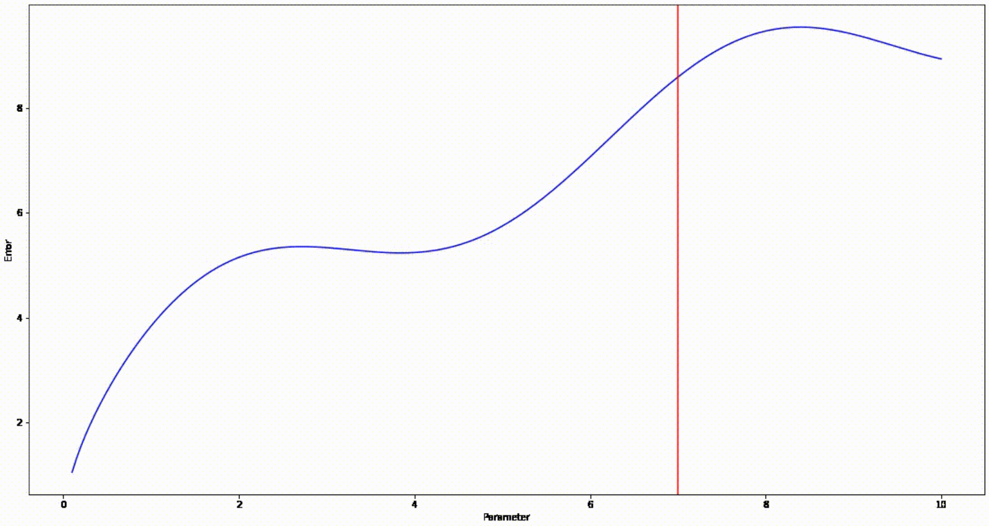


Gradient descent can get stuck in **local minimum** !

How to solve this ?

- change step size
- start again with different initialization
- **give it a whack - stochastic descent**
- treat it as a ball - momentum

Stochastic Gradient Descent



If we could give our ball a little whack from time to time ...

Random noise !

Stochastic Gradient Descent

```
w = random()

for i in 1,2,..., MAX_ITERATIONS:
    dw = derivative_of_error(w)
    w = w - a*dw + random_small_value()
return w
```

What is the most effective way of doing this ?

Stochastic Gradient Descent - a simple trick

We need to add noise to the derivative

=

We need to make derivative **imprecise**
(but not totally wrong)

Instead of using whole dataset to calculate the error and its derivative, we can use just a **different random sample** in each iteration !

notebooks

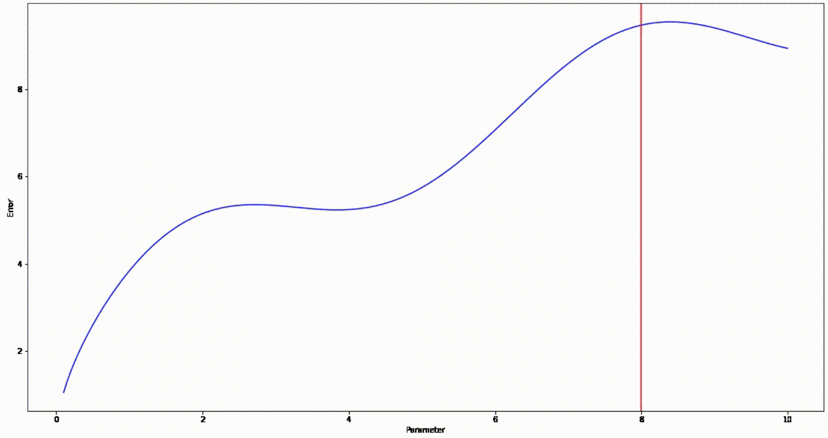
Stochastic Gradient Descent

```
N = sample size
w = random_initialization()

for i in 1,2,..., MAX_ITERATIONS:
    select_random_data(N)
    dw = derivative_of_error(w)
    w = w - a*dw
return w
```

What is more, we save memory ! You only need to hold one sample in memory at a time !

Gradient Descent - problem

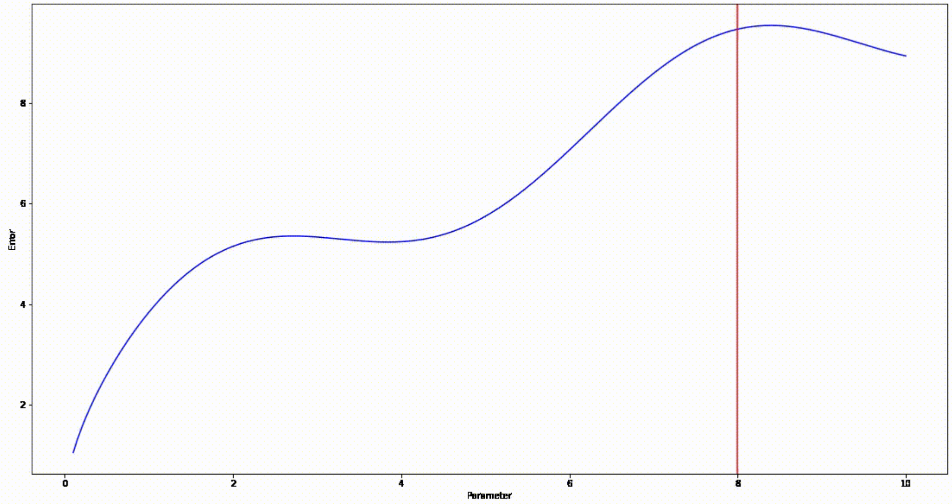


Gradient descent can get stuck in **local minimum** !

How to solve this ?

- change step size
- start again with different initialization
- give it a whack - stochastic descent
- **treat it as a ball - momentum**

Gradient Descent with Momentum



- physical things do not immediately stop moving
- they have a **momentum** and it takes some time and force to stop them
- if our descend was like that, it would not stop immediately in a local minimum

Gradient Descent with Momentum

```
b <0;1>, for example 0.9
w = random()
update = 0

for i in 1,2,..., MAX_ITERATIONS:
    update = b*update - a*derivative_of_error(w)
    w = w + update
return w
```

Algorithms in practice

use combination of momentum,
stochasticity and couple of other tricks ...

- **Adam**
- Adamax
- RMSProp

Exercise time

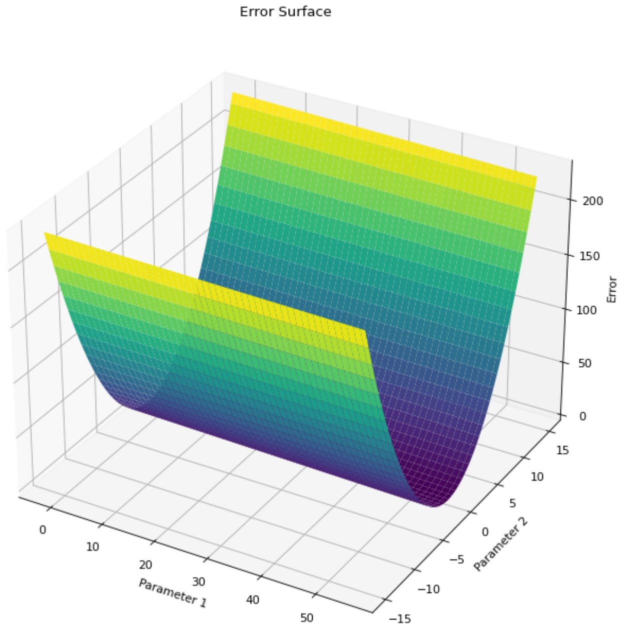
https://github.com/ssmatana/czechitas/blob/main/Copy_of_czechitas_season1_episode3.ipynb

Exercise 3.1

Error with two parameters

- error curve becomes an error surface ("landscape")

- **how will the derivative look like in this case ?**



Derivative with two parameters

One parameter

$$w \longrightarrow \frac{dE}{dw}$$ a single number, like 5

Two parameters

$$w1 \longrightarrow \frac{dE}{dw1}$$

$$w2 \longrightarrow \frac{dE}{dw2}$$

two numbers, like 5 and 42

N parameters means N derivatives

Derivatives in a vector

$$w1 \longrightarrow \frac{dE}{dw1}$$

$$w2 \longrightarrow \frac{dE}{dw2}$$

$$\longrightarrow \mathbf{g} = [\ \frac{dE}{dw1}\ ,\ \frac{dE}{dw2}\ ]$$

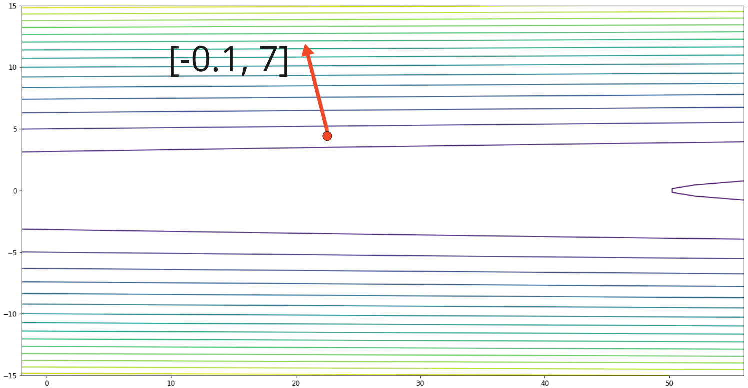
We collect derivatives into a vector and treat them as one entity

The vector of derivatives is called **gradient.**

That's where the "Gradient descent" comes from !

Interpreting gradient

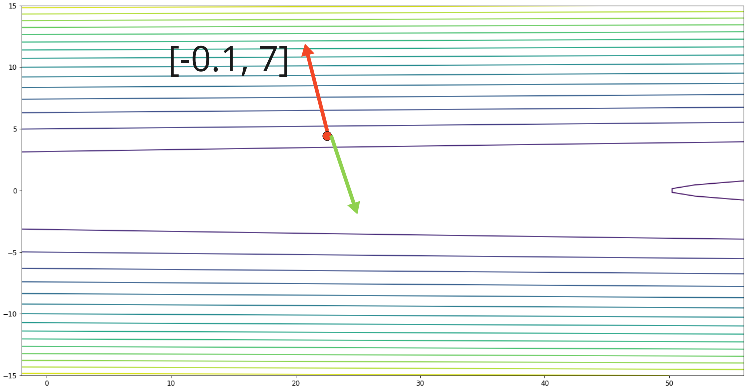
Gradient represents the direction of **steepest** increase.



Where should we go ?

Interpreting gradient

Gradient represents the direction of **steepest** increase.



Where should we go ?

In the opposite direction !

Manipulating gradient

Opposite of a vector: w [-0.1, 7] -w [0.1, -7]

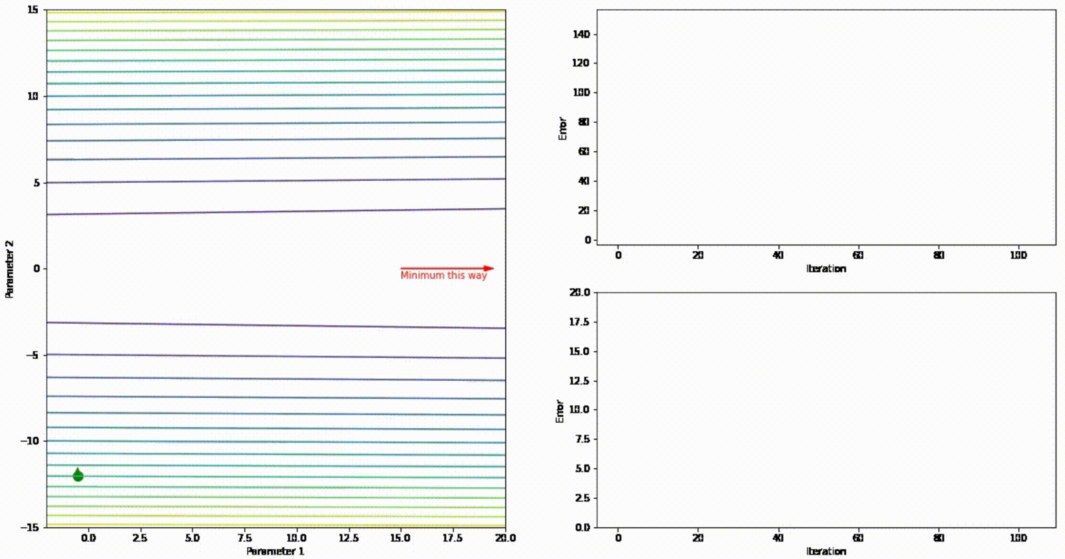
Scaling of a vector: w [-0.05, 3.5] 2w [-0.1, 7]

So what do we do if we want to go in the opposite direction of gradient with some learning rate?

w = **w** - a***g**

Seems familiar ?

Problem with multiple dimensions



You need to normalize your data

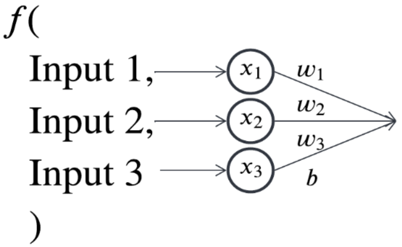
Length travelled in meters	Height in meters
0 - 10000	1.4 - 2.2
↓	↓
0 - 1	0 - 1

minimum ↗ 0 - 1 ↖ maximum

Conclusions

- error changes with parameters, this means that it can be seen as a **function** of parameters
- to find parameters that lead to low error, we use an algorithm called **gradient descent**
- this algorithm exploits properties of derivatives to navigate parameter space effectively
- naive implementation of gradient descent can get stuck in **local minima**, therefore it is beneficial to extend it with the idea of **momentum** and **stochastic gradient descent**
- when we consider more than one parameters, we have multiple derivatives and we collect them into one vector called **gradient**, but this does not change the general algorithm
- if we work with multiple inputs, the inputs must be **normalized** (must have similar range), otherwise we risk a significant slowdown of learning

More fun with vectors



Output $= \sigma(b + \sum_i x_i w_i)$

x1w1 + x2w2 + x3w3 + b

Dot product

w = [b, w1, w2, w3]

x = [1, x1, x2, x3]

+

wx = x1w1 + x2w2 + x3w3 + b

np.dot(x,w)

More fun with vectors - matrices

Matrix = collection of vectors

w=[b, w1, w2, w3]

*

X=

[1, a1, a2, a3] → b + a1w1 + a2w2 + a3w3

[1, b1, b2, b3] → b + b1w1 + b2w2 + b3w3

[1, c1, c2, c3] → b + c1w1 + c2w2 + c3w3

[1, d1, d2, d3] → b + d1w1 + d2w2 + d3w3

All this is denoted by **Xw** and in Python done by

np.matmul(X,w)
or X @ w

Exercise time