Data Science Academy

Data Science for Business – Becoming a Data Science Expert (D)

Pilot Presentation:
for participants of and use in the pilot only

07.09.2021

Boehringer Ingelheim

Agenda week two

Schedule week two

Week 2			
	Day 1 Monday, 06.09.2021		Day 2 Tuesday, 07.09.2021
Start: 12:00	Recap	Start: 12:00	Recap
	5 – Unsupervised Learning (Part 1)		6 – Gradient Descent (Part 1)
14:00 – 15:00	Break	14:00 – 15:00	Break
	5 – Unsupervised Learning (Part 2)		6 – Gradient Descent (Part 2)
End: 18:00	Q&A and Feedback	End: 18:00	Q&A and Feedback

We will also have several short coffee breaks in between.

Feedback for pilot training

We aim to provide a great training experience for you and are looking forward to receiving your feedback!

You will have three different ways to give us your feedback on each training day:

1. We will have an **anonymized** feedback collection **after the last session** of each day per **Myforms.**
2. We will have an **open feedback round and discussion** at the **end of each training day.**
3. Please also **take notes** regarding your ideas during the sessions: **locally or via the Mural Board** which you can reach via LINK.

Quiz: Recap week two day one

Please join at slido.com with #031 077.

Let’s go through some questions together.

Let’s see what you think. All answers will be anonymous.

Module 6
Gradient Descent

Agenda week one

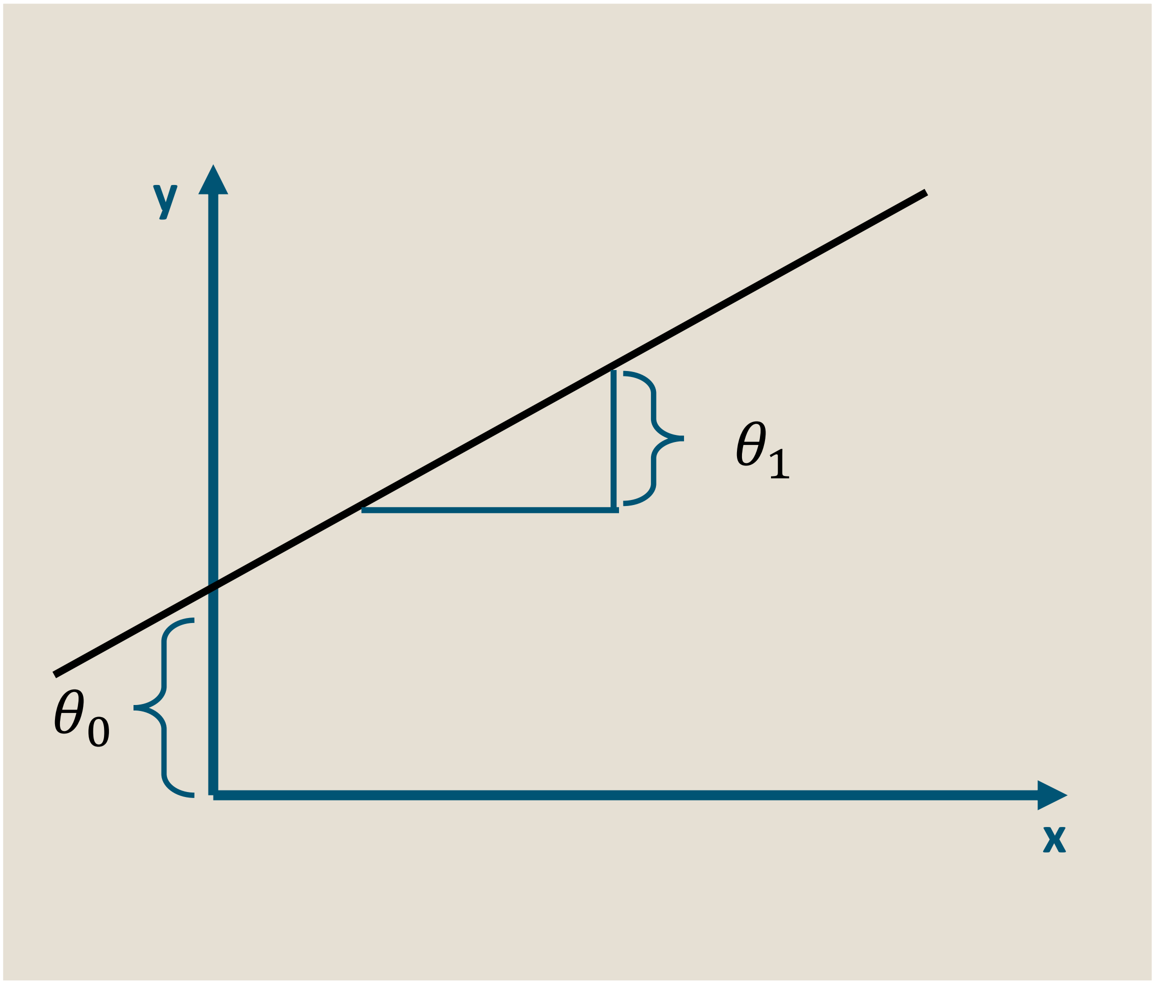
Linear Regression

Two-dimensional example

- Parameters slope and intercept θ=(b,a)
- The linear function with input x is $f(x;\theta) = \theta_1 x + \theta_0$

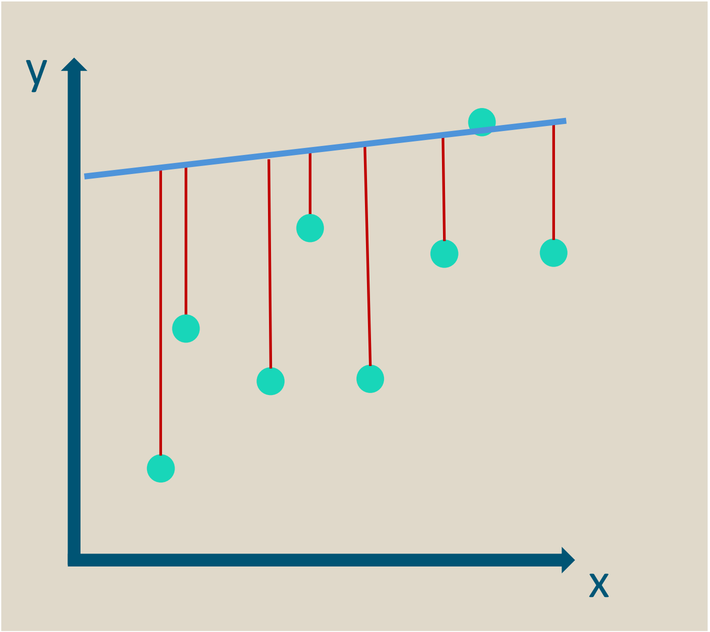
For the d dimensional case

- Parameters $\theta = (\theta_0, \theta_1, .., \theta_d)$
- Input: $x_i = \left(x_i^{(1)}, \ldots, x_i^{(d)}\right)$
- The linear function is $f(x;\theta) = \theta_0 + \theta_1 x^{(1)} + \ \ldots + \theta_d x^{(d)}$



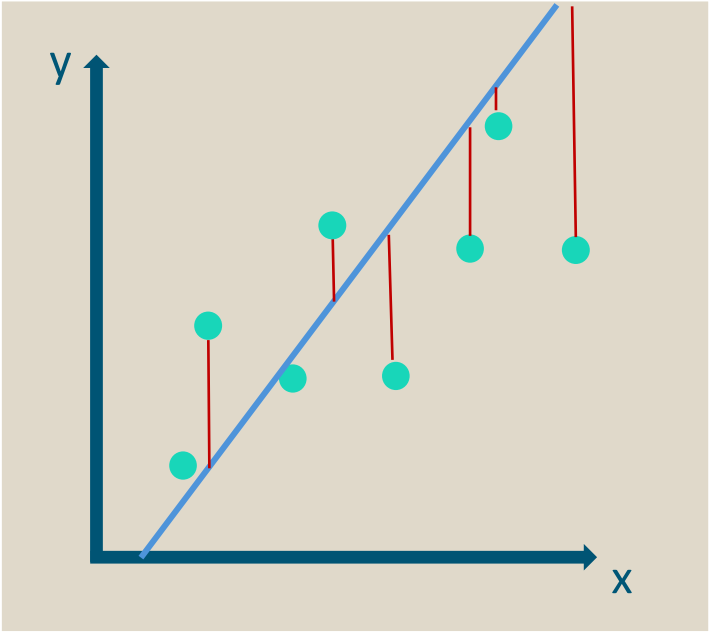
How does the algorithm find the best fit?

Line 1



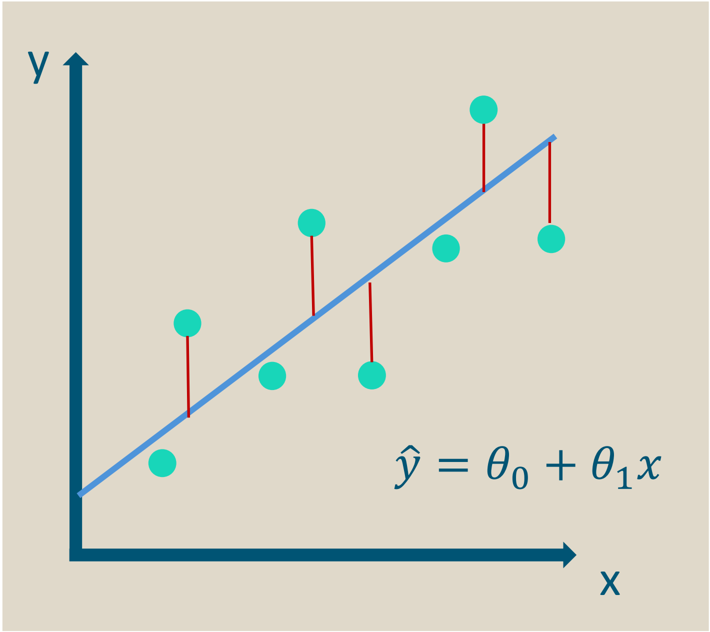
- We **randomly initialize a line**
- The total error will be high
- The **error** indicates that we should **move the line** downward and increase the slope

Line 2



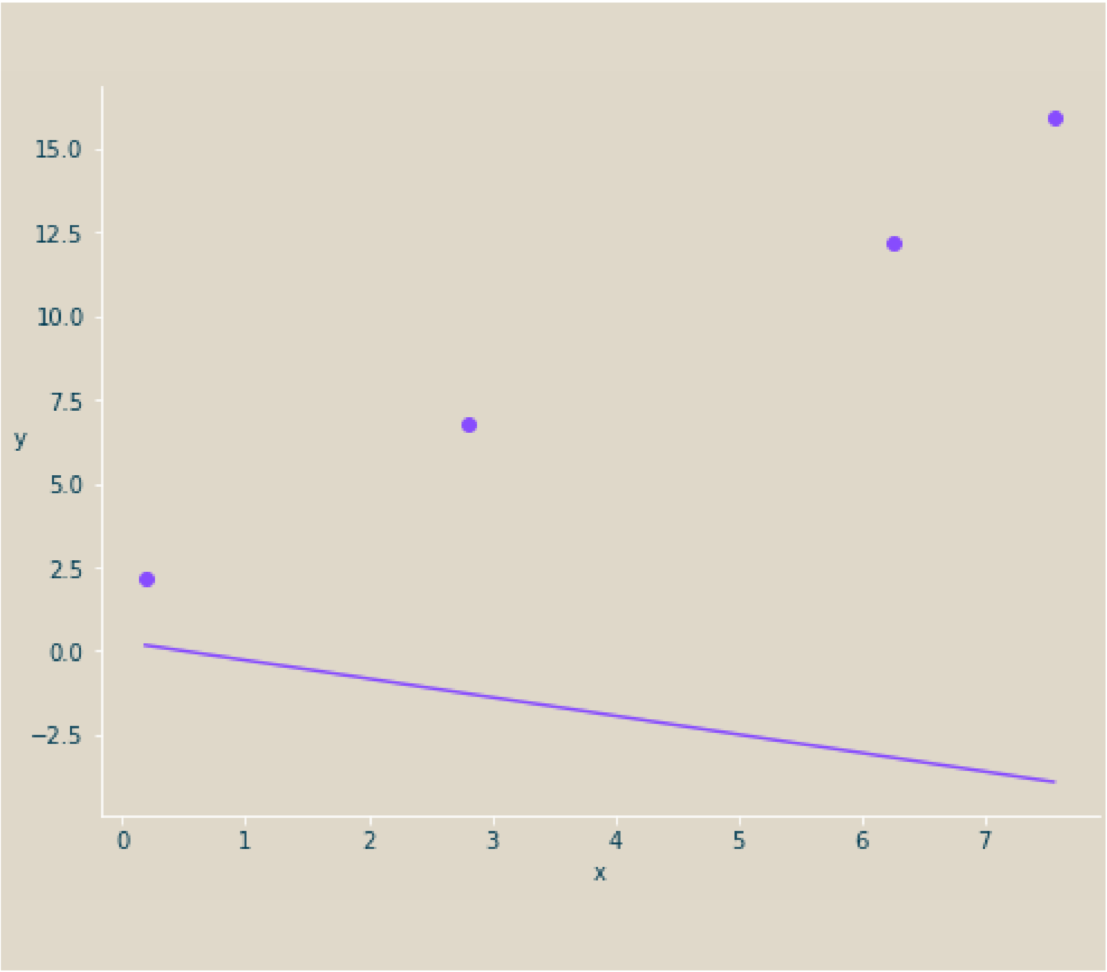
- The **error has decreased**, but we overshot the goal
- Due to the lower error, we will change the parameters of the line less

Line 3



- We achieved the **smallest possible error**
- The **error is evenly distributed** between above and below points

Randomly initialize a line



Randomly initialize the models' parameters $\boldsymbol{\theta}$:

$$\theta = (0.28, \quad -0.55)$$

Build the model $\boldsymbol{f(x;\theta)}$:

$$f(x;\theta) = \theta_0 + \theta_1 \boldsymbol{x}$$

Use the function to compute the predicted values for the points. Example $x = 3$:

$$f(3;\theta) = 0.28 + (-0.55) * 3 \approx -1.5$$

How good is the current line?

Loss function

Given a true value y and a prediction of a 'current' model $\hat{y}$,

$l(y, \hat{y})$ = "how far is $\hat{y}$ away from y?

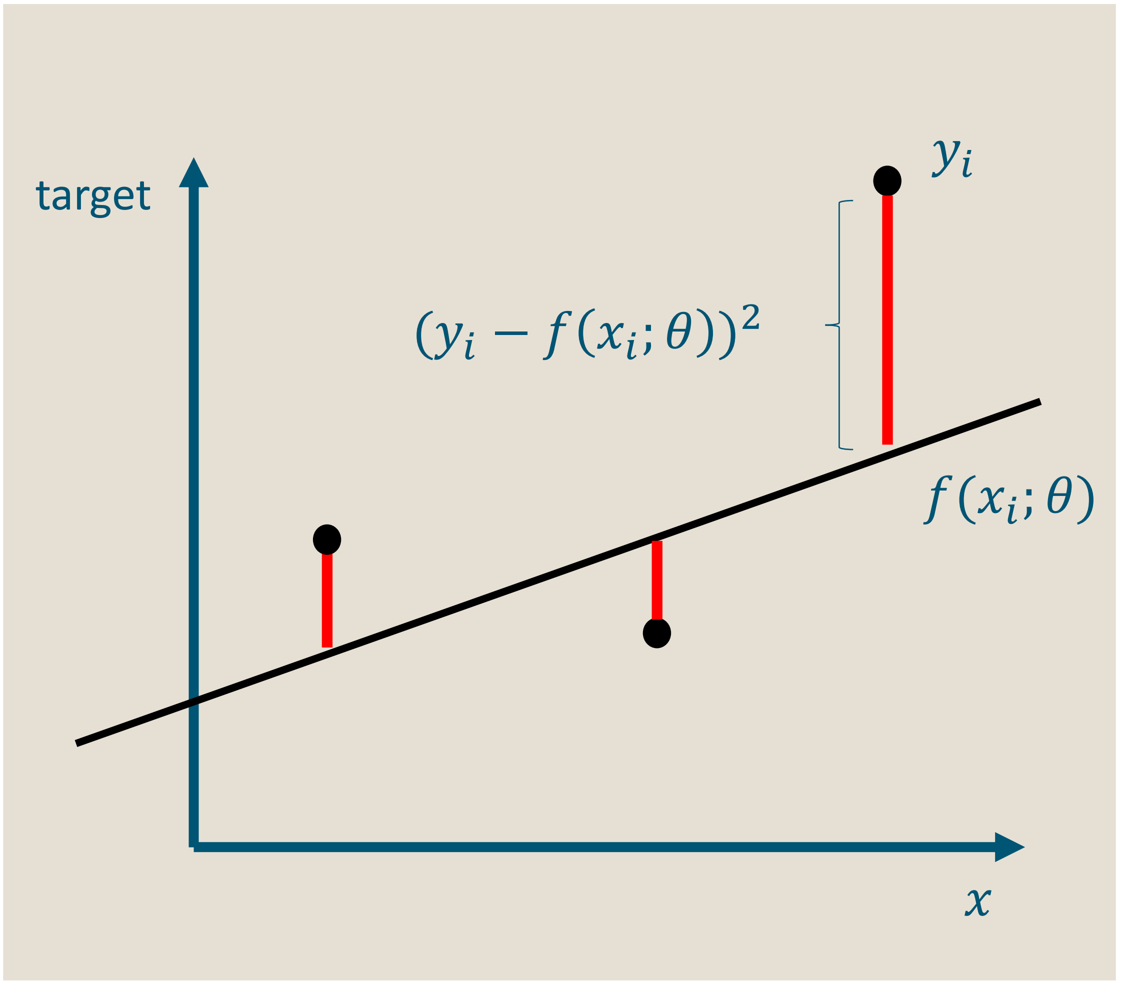
- In our case: $l(y, \hat{y}) = (y - \hat{y})^2$

This is just one single y_i but we need loss for the whole training set:

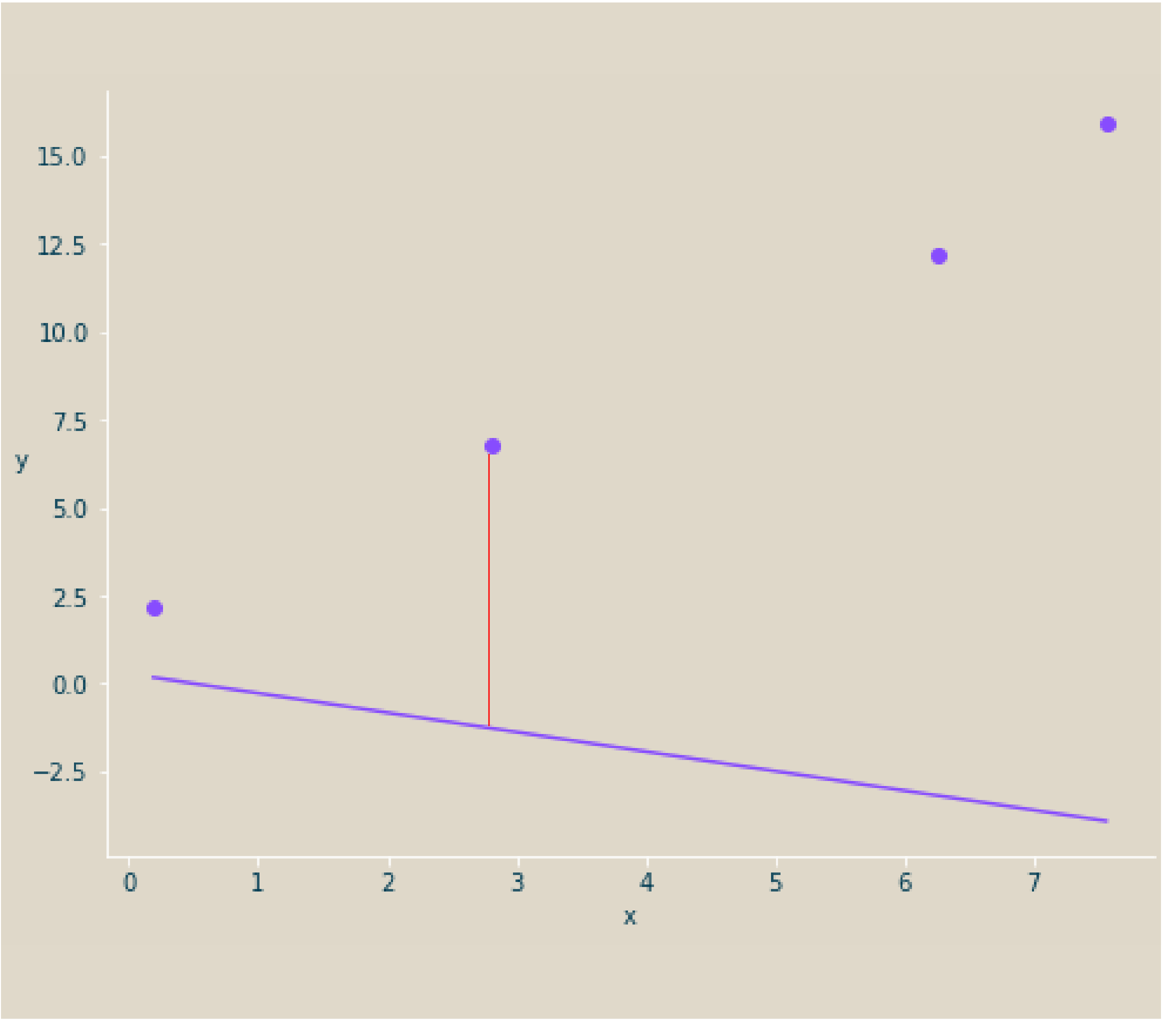
$$L(\theta) = \sum_{i=1}^{n} l(y_i, f(x_i; \theta))$$

Where $\theta = (b, a)$ and $f(x; \theta) = bx + a$

$$L(b, a) = \sum_{i=1}^{n} \left(y_i - (bx_i + a)\right)^2$$



Example: Linear regression



The predicted value for x = 3 is

$$f(3;\theta) = 0.28 + (-0.55) * 3 = -1.496$$

The actual position of y is 7.

The loss of our loss function is described by

$$L(\theta) = \sum_{i=1}^{n} l(y_i, f(x_i;\theta))\text{, where } l(y,\hat{y}) = (y-\hat{y})^2$$

$$l(y,\hat{y})^2 = (7-(-1.5))^2 = 72.25$$

How do we minimize the function $L(\theta)$?

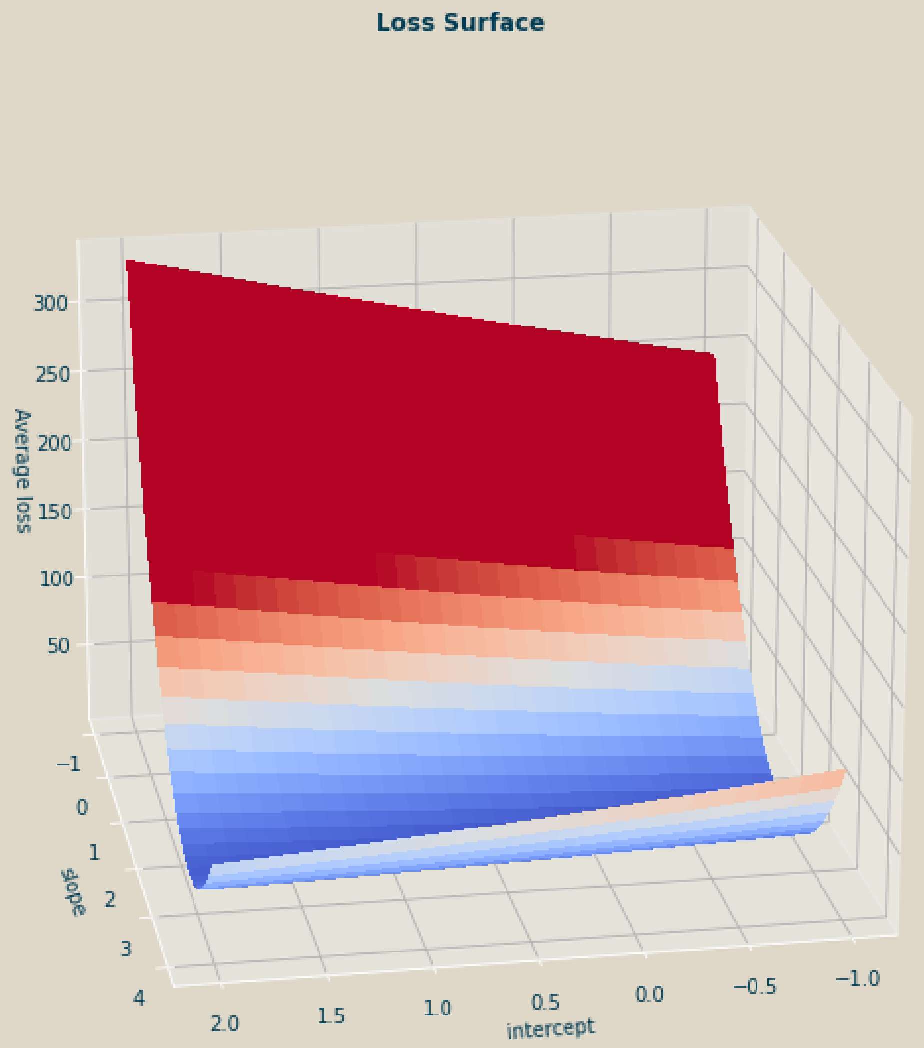
We want to minimize the loss $\mathbf{L}(\boldsymbol{\theta})$

The derivative $\mathbf{L}'(\boldsymbol{\theta})$ gives us the slope of the error

- The error is increasing in the direction of $L'^{(\theta)}$
- Moving in the opposite direction $-L'(\theta)$ reduces the error

For multi-dimensional function $\boldsymbol{\theta}$ is $(\boldsymbol{\theta_0}, \boldsymbol{\theta_1}, .., \boldsymbol{\theta_d})$

- We want to minimize the Loss for every Parameter θ_i, so we need one derivative per parameter
- This is called the gradient of $L(\theta)$ which is the partial derivative for each parameter for L: $\nabla L(\theta) = (\partial_{\theta_0} L, \ldots, \partial_{\theta_d} L)$



How do we minimize the function $L(\theta)$?

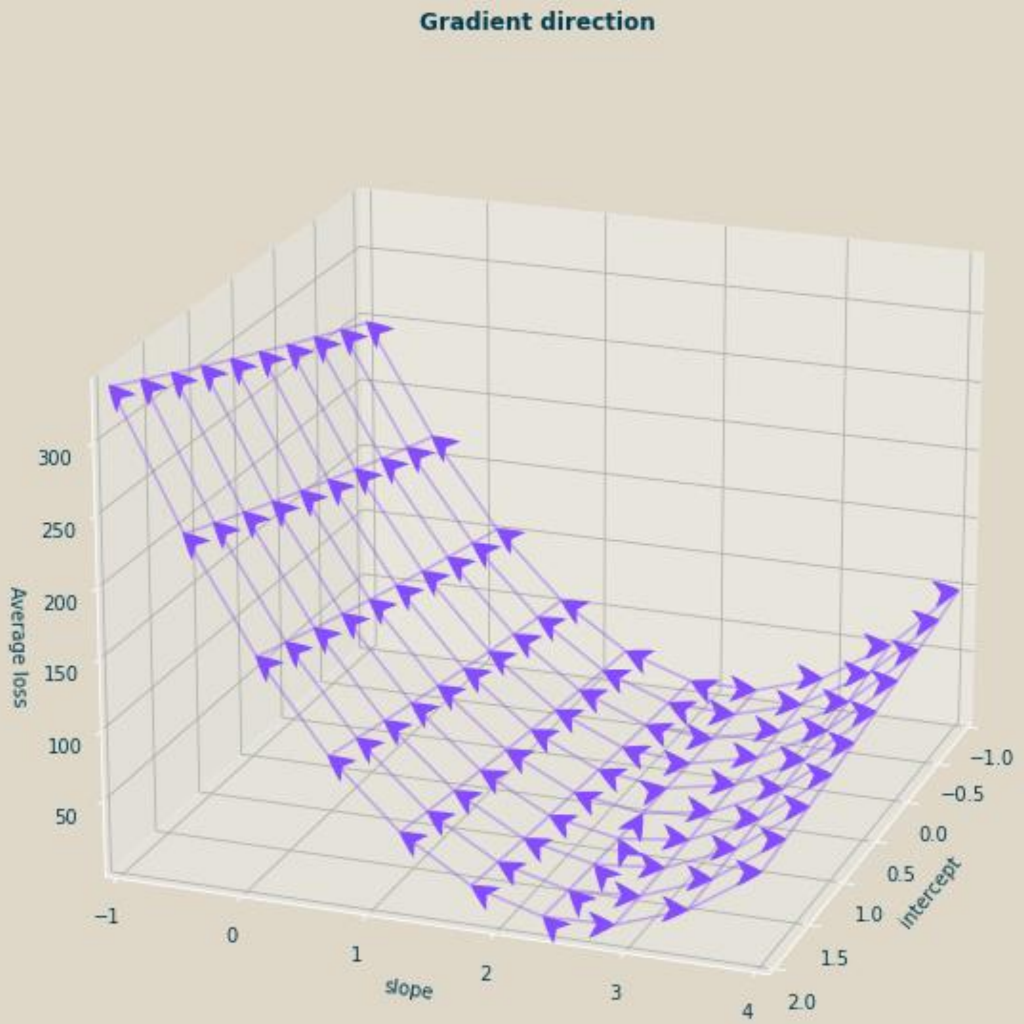
We want to minimize the loss $\mathbf{L(\theta)}$

The derivative $\mathbf{L'(\theta)}$ gives us the slope of the error

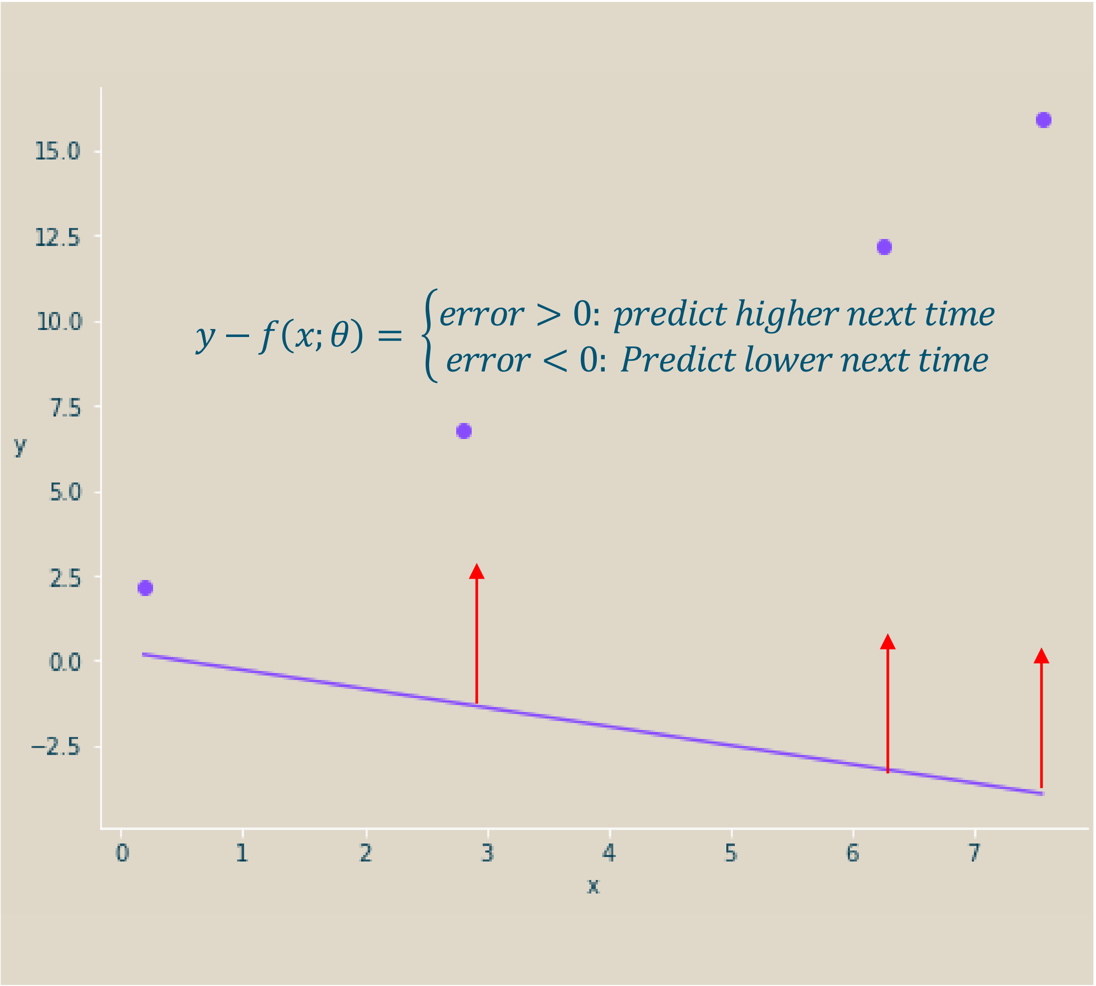
- The error is increasing in the direction of $L'^{(\theta)}$
- Moving in the opposite direction $-L'(\theta)$ reduces the error

For multi-dimensional function $\boldsymbol{\theta}$ is $(\boldsymbol{\theta_0}, \boldsymbol{\theta_1}, .., \boldsymbol{\theta_d})$

- We want to minimize the Loss for every Parameter θ_i, so we need one derivative per parameter
- This is called the gradient of $L(\theta)$ which is the partial derivative for each parameter for L: $\nabla L(\theta) = (\partial_{\theta_0} L, \dots, \partial_{\theta_d} L)$



Example: Linear regression



For the input x = 3 our algorithm genrated a loss of 72.25

Now we create the gradient of the loss function

$$\boldsymbol{L(x,\theta) = l(y,\hat{y}) = \left(y - (\theta_0 + x\theta_1)\right)^2}$$

For this we use the chain rule

$$(f \circ g)'(x) = f'\left(g(x)\right) * g'(x)$$

The derivative of our loss $\mathrm{L}(\theta)$ **with** $\nabla L(\theta) = (\partial_{\theta_0} L, \dots, \partial_{\theta_d} L)$ **is**

$$\nabla L(\theta) = \begin{cases} \dfrac{\delta}{\delta\theta_0} = -2 * \left(\boldsymbol{y} - f(x;\theta)\right) = \boldsymbol{-144,5} \\ \dfrac{\delta}{\delta\theta_1} = -2 * \boldsymbol{x} * \left(\boldsymbol{y} - f(x;\theta)\right) = \boldsymbol{-433,5} \end{cases}$$

Gradient Descent Algorithm

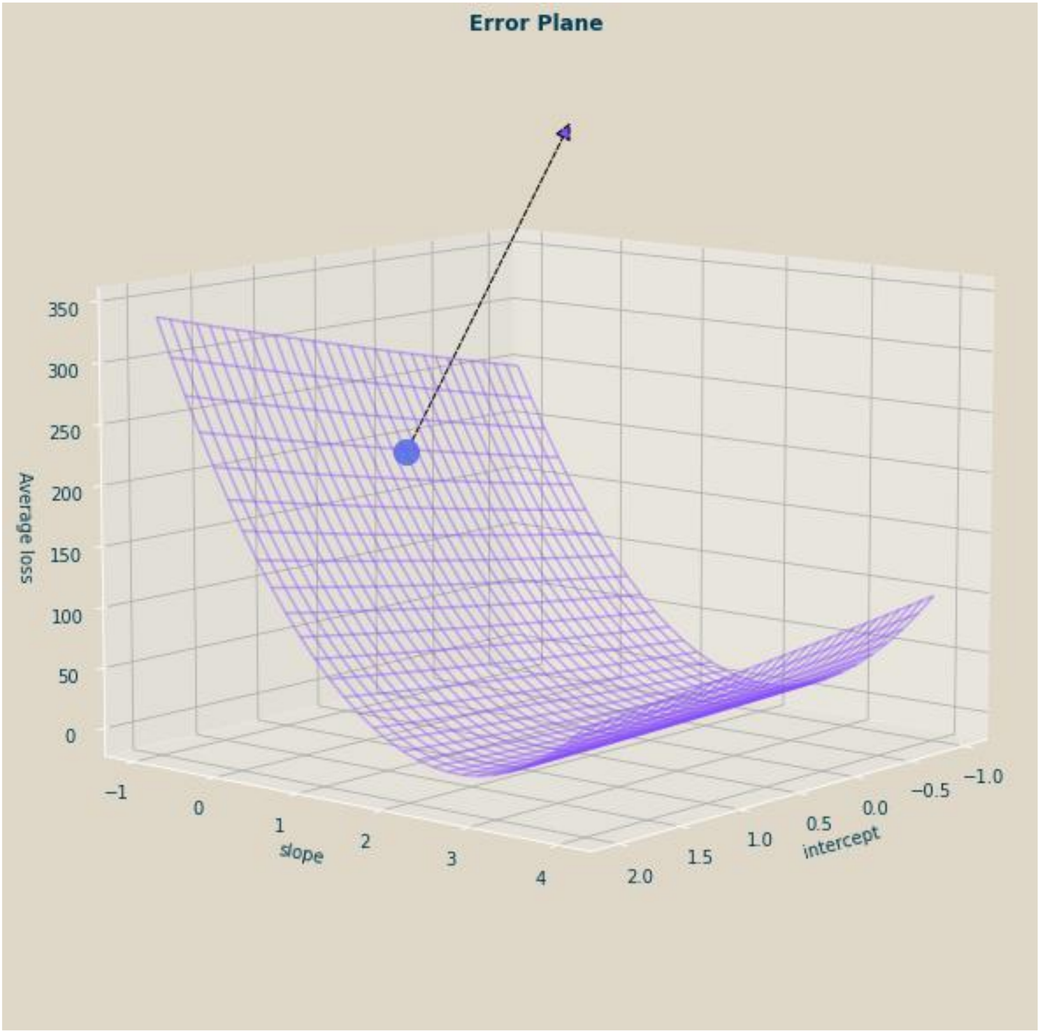
Minimize $L(\theta)$

Input (decide on): A function $f(x;\theta)$, input x, target y, loss $L(\theta)$

1. **Initialize** θ **randomly**
2. **While** θ **still changes**
    1. Build the model $f(x;\theta)$
    2. Evaluate the fit with $L(\theta) = \sum_{i=1}^{n} l(y_i, f(x_i;\theta))$
    3. Update the parameters based on the negative gradient: $\theta = \theta \; - L'(\theta)$
3. **Return fitted model** $f(x;\theta)$

The gradient gives us the direction of the update.

Example: Linear regression



Our randomly initialized parameters are the following

$$\theta = (0.28, -0.55)$$

Our The calculated gradient from the previous step is

$$\nabla L(\theta) = (-144.5, -433.5)$$

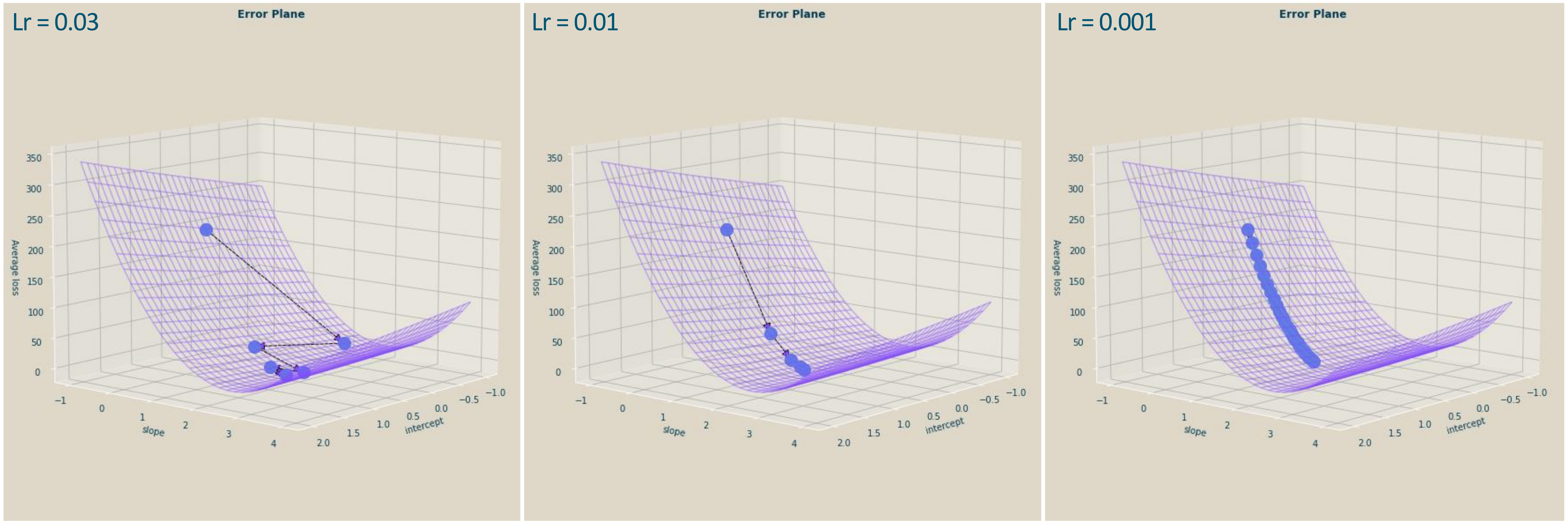
Update the weights accordingly

$$\theta_{new} = \theta - \nabla\theta$$

$$\theta_{new} = (\mathbf{0.28}, -\mathbf{0.55}) - (-\mathbf{144.6}, -\mathbf{433.5})$$

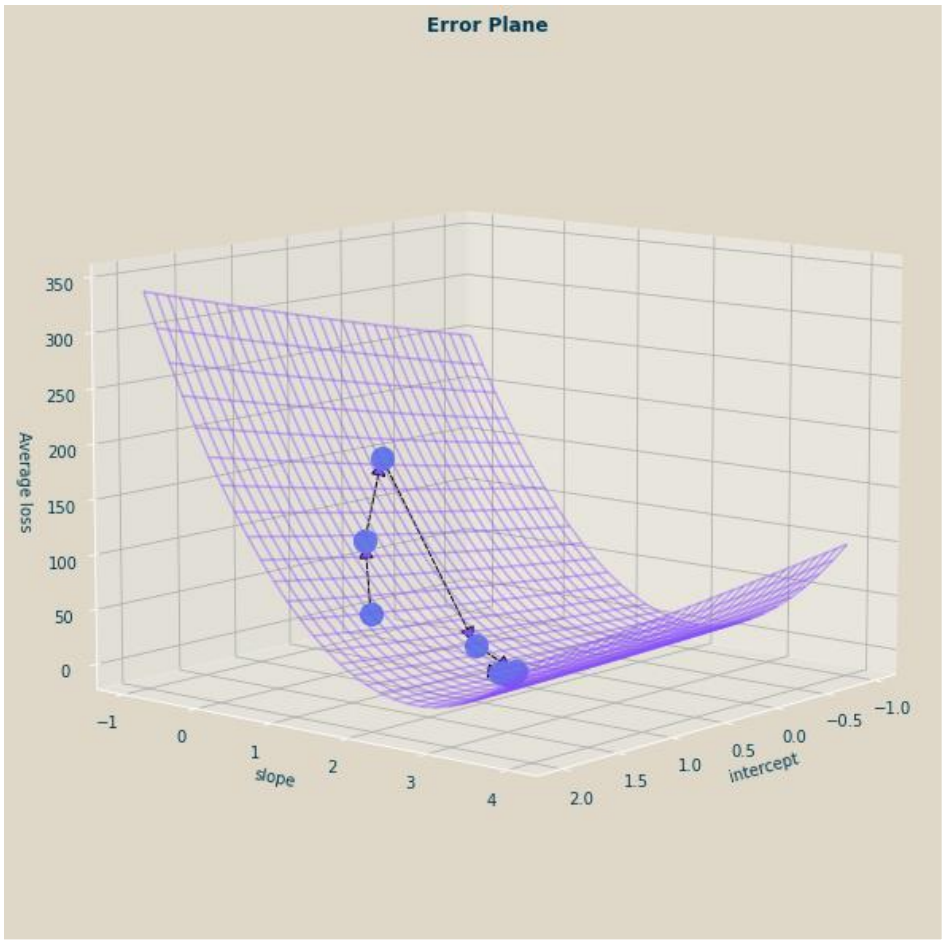
$$\theta_{new} = (144.88, 432{,}95)$$

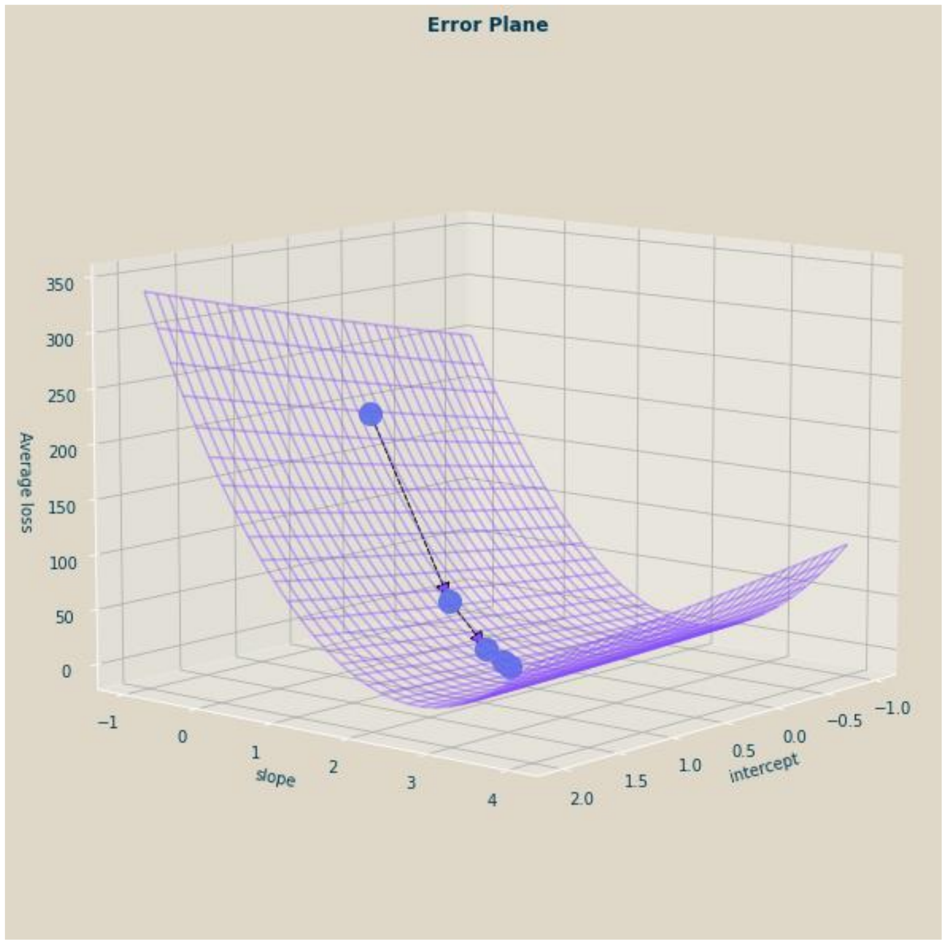
The influence of the learning rate



Choosing the right learning rate is crucial to allow fast convergence of the algorithm.

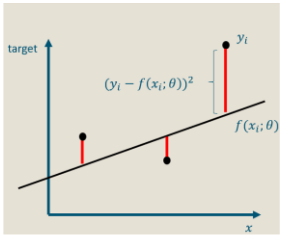
Stochastic gradient descent



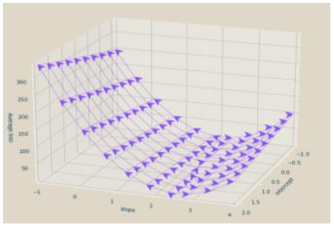


Averaging the gradient updates for each iteration increases the stability of the descent

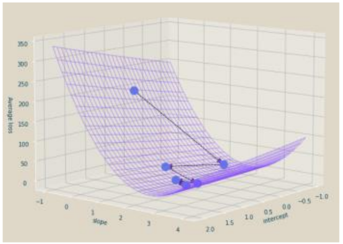
Important takeaways



We can describe our **linear regression** as $f(x;\theta) = \theta_0 + \theta_1 x^{(1)} + \ \dots + \theta_d x^{(d)}$, where each variable is assigned a weight θ_i.
The goodness of fit is evaluated with the **loss function**.



With our loss function we can create a **loss surface**. The **gradient** is a vector that points in the direction of the biggest loss increase. **By following in the opposite direction of the loss we reduce the error in each step**.



The gradient controls the direction of the descent. **The learning rate controls the step size** with which we move.

- Setting learning rate too high leads to overcorrections and increasing loss
- Setting the learning rate too low leads to a really slow descent

The gradient give us the direction of the update. The learning rate controls the step size of the update.

Try it yourself!

In the following exercises

Logistic regression

- Model for classification would then be $\sigma(f(x;\theta))$
  - σ is a 'squash' function $\mathbb{R}$ to the range $[0,1]$
  - $\sigma(t) = \frac{1}{1+e^{-t}}$
  - $\sigma'(t) = \sigma(t) * (1 - \sigma(t))$
- We can calculate the error by

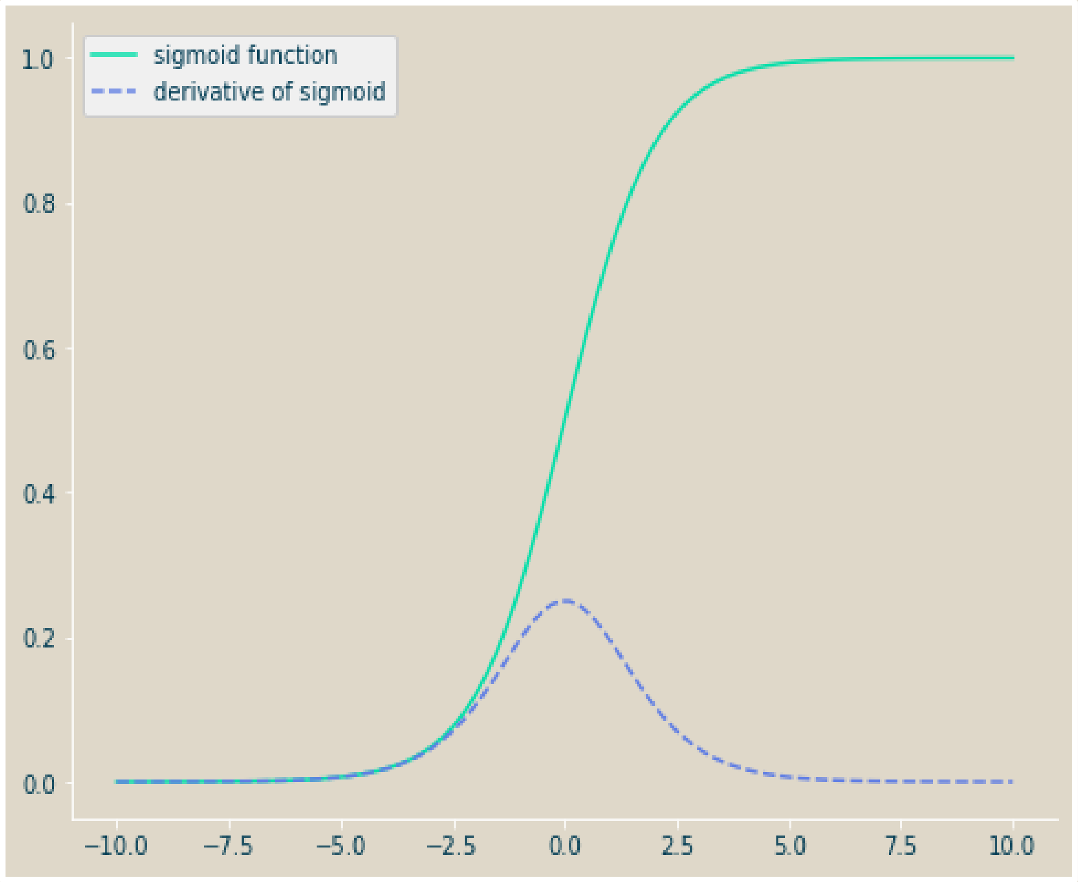
$$L(\theta) = \sum_{i=1}^{n} l(y_i, \sigma(f(x_i;\theta)))$$

And the derivative with

$$\nabla L(\theta) = \sum_{i=1}^{n} \sigma\big(f(x_i;\theta)\big) * \Big(1 - \sigma\big(f(x_i;\theta)\big)\Big) * \nabla_\theta l(y_i, f(x_i;\theta))$$

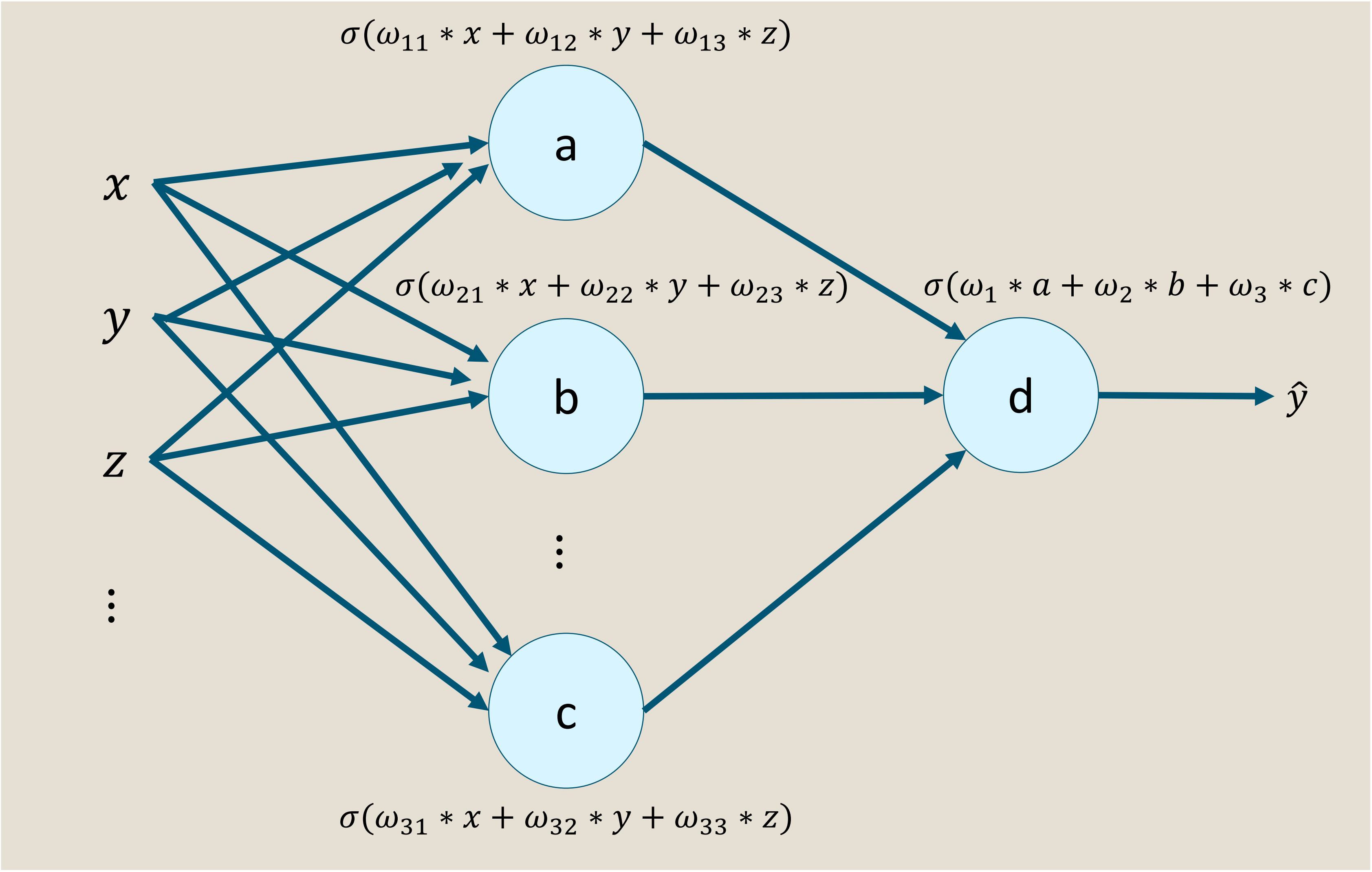
- And then update the weights with

$$\theta = \theta - \nabla L(\theta)$$



Neural Networks forward pass

- Combining multiple neurons into layers and fully connecting the layers leads to a Neural Network
- A neural network is nothing more than multiple logistic regressions stacked on top of each other
- Each input into a neuron is multiplied by a weight ω_{ij}
- The goal is to find the weights that minimize our training error



Neural Networks: Backpropagation

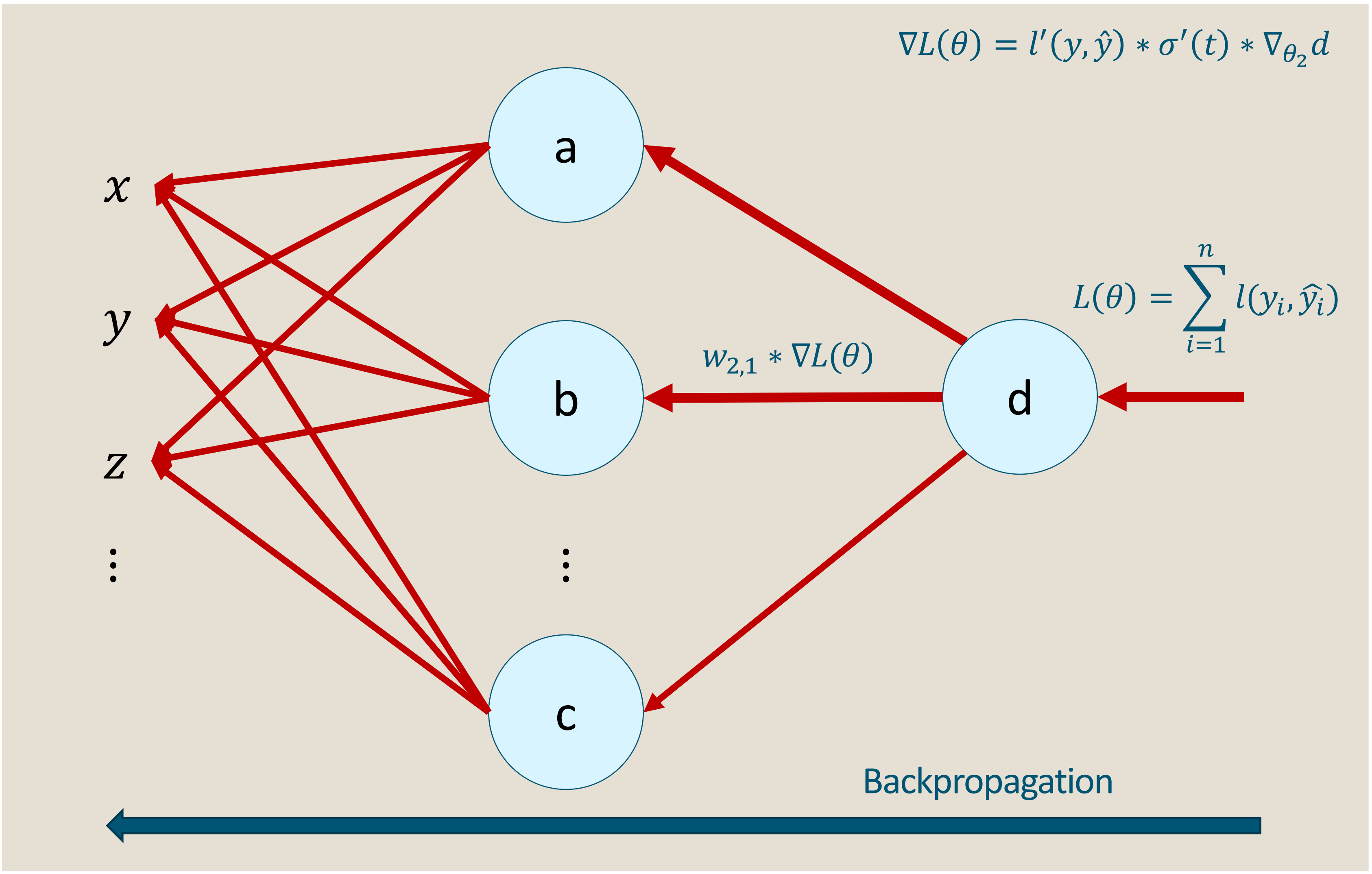
- Calculate the error of the prediction as loss $L(\theta)$
- Based on the loss we want to update the weights responsible for the error
- The first layer is computed as in a logistic regression

  $\nabla L(\theta) = l'(y, \hat{y}) * \sigma'(t) * \nabla_{\theta_2} d$
- Each neuron is responsible for a part of the error. We get the responsibility by

  $\delta_i = (\sum w_{i,j} \delta_j) \sigma'(t)$

  Where δ_j is the error of the neuron in the previous layer
- After calculating the gradients we can update the weights as usual

  $\omega_{i,j} = \omega_{i,j} - \delta_{i,j}$

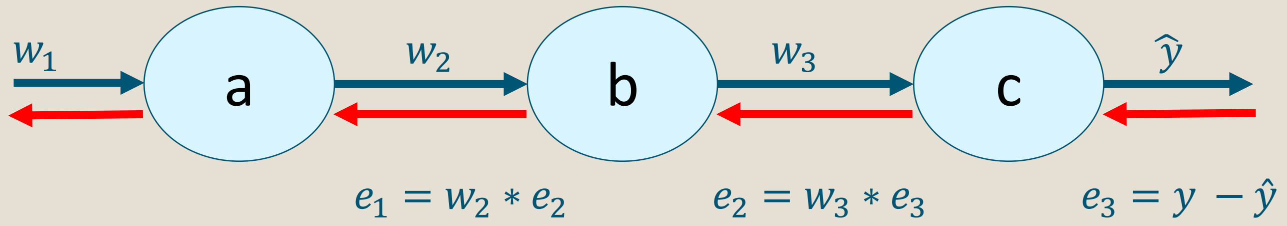


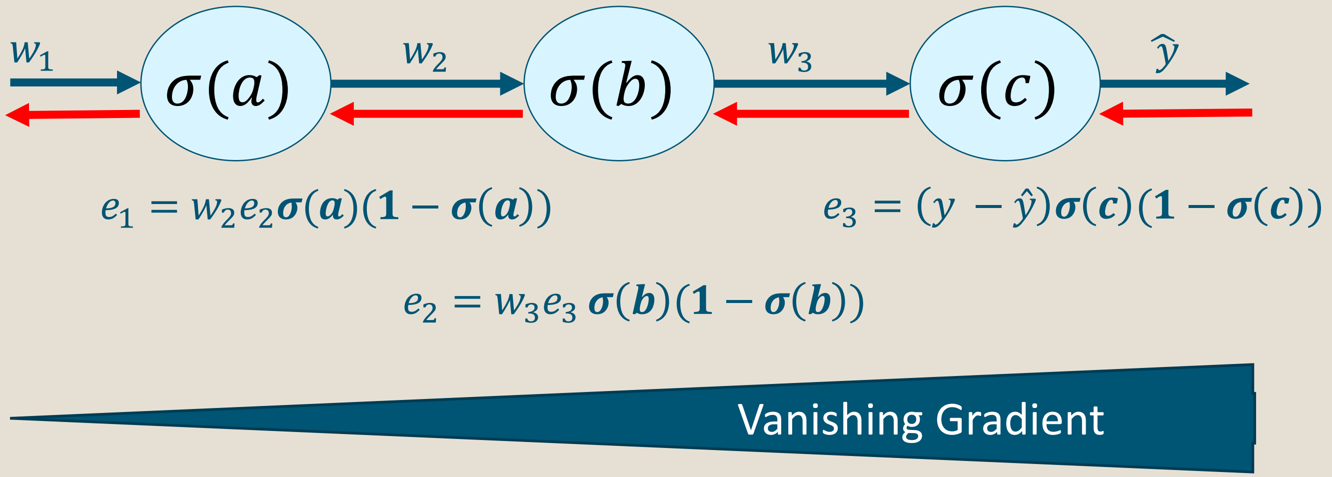
Vanishing Gradient

- The loss is propagated backwards through the layers
- The contribution of each neuron is determined by the connected weight
- Because the chain rule, the error of a layer is dependent on the loss coming from all previous layers

$$(f \circ g)'(x) = f'\big(g(x)\big) * g'(x)$$

- The derivative of the sigmoid activation function is 0.2 at max
- Multiplying several times by a number much smaller than 1 leads to the **vanishing gradient** problem





ReLu for Deep Learning

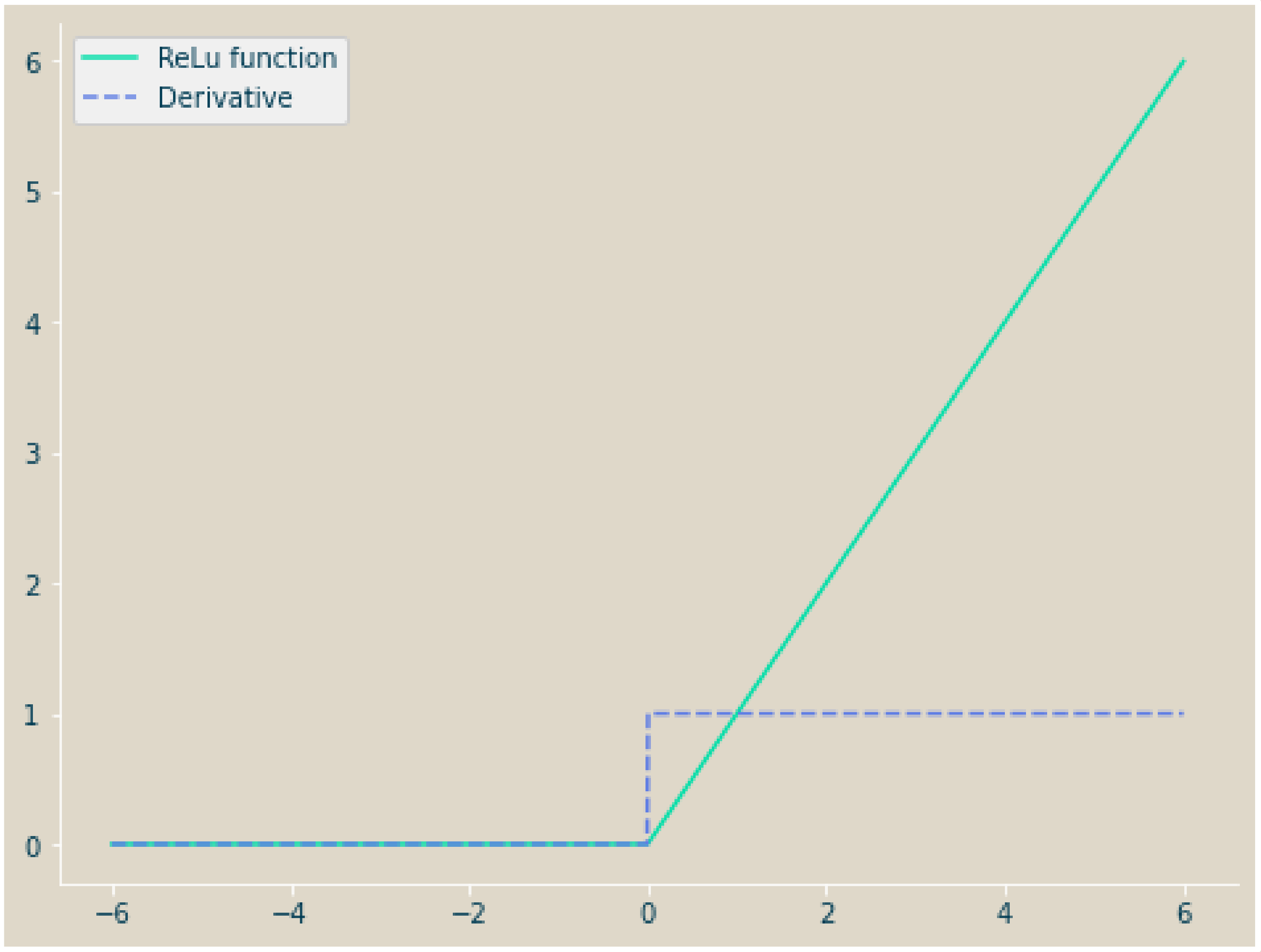
- The solution to the vanishing gradient problem is to exchange the sigmoid activation function with the ReLu activation function

$$ReLu(x) = \begin{cases} x \; for \; x > 0 \\ 0 \; for \; x \leq 0 \end{cases}$$

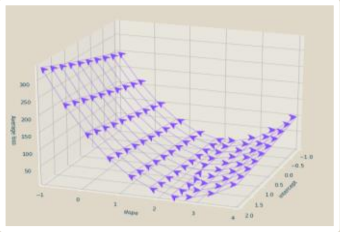
- The derivative of the ReLu

$$ReLu'(x) = \begin{cases} 1 \; for \; x > 0 \\ 0 \; for \; x \leq 0 \end{cases}$$

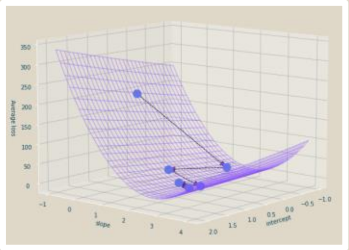
- Multiplying by a number smaller than 1 and reducing the gradient in each layer
- The derivative of ReLu is one, which leaves the gradient from layer to layer unchanged



Important takeaways

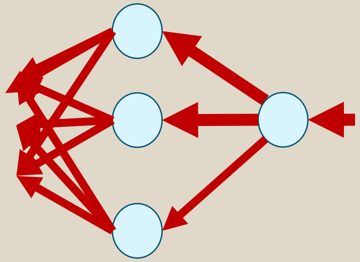


With our loss function we can create a **loss surface**. The **gradient** is a vector that points in the direction of the biggest loss increase. **By following in the opposite direction of the loss we reduce the error in each step**.



The gradient controls the direction of the descent. **The learning rate controls the step size** with which we move.

- Setting learning rate too high leads to overcorrections and increasing loss
- Setting the learning rate too low leads to a really slow descent



Neural Networks can be seen as stacked logistic regressions. Using **backpropagation**, we can propagate the error through the network and update all weights.

- The vanishing gradient occurs when we have many layers and the error is multiplied several times by derivatives smaller than 1
- The ReLu activation function can be used to solve that problem

The gradient descent method can optimize any function which has a derivative.

Quiz: Gradient Descent

Please join at slido.com with #031 077.

Let’s go through some questions together.

Let’s see what you think. All answers will be anonymous.

Try it yourself!

In the following exercises

Agenda

Feedback and Q&A

Thank you

If you would like any further
information please contact
Werner,Dr.,Fabian_Georg (BI X) BIX-DE-I
<fabian_georg.werner@boehringer-ingelheim.com>