Künstliche Intelligenz
Feedforward Neural Networks

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

Artificial Neural Network

- An *artificial neural network* is a graph of artificial neurons.
- Some units receive external inputs (*input units*)
- Every unit computes its *potential* based on the outputs of its predecessors and its incoming weights
- Every unit computes an *output value* by applying a non-linear function to the potential and a bias value

- Different neural architectures differ with respect to:
  - Type of input function used (e.g., weighted sum)
  - Type of output function (e.g., threshold, sigmoidal, relu, ...)
  - Connection structure (acyclic = feed forward, cyclic = recurrent, layered = groups of units, convolutional, ...)
  - Dynamics (synchronous, asynchronous, probabilistic, ...)

Nice introductory videos

Below you will find a list of nicely animated introductory videos:

- But what is a Neural Network? Deep learning, chap.1:
  [YouTube aircAruvnKk]
- How Deep Neural Networks Work (up to Min. 12):
  [YouTube ILsA4nyG7I0]
- Artificial Neural Networks - Fun and Easy Machine Learning (up to Min. 10):
  [YouTube GQVL1ORqpSs]
- A Visual And Interactive Look at Basic Neural Network Math:
  jalammar.github.io/feedforward-neural-networks-visual-interactive

A larger Network in Action

`adamharley.com/nn_vis/cnn/2d.html`

Take-away-messages of section: *"Dynamics of Artificial Neural Networks in General"*

E You should now be able to ...

- explain the general structure of an artificial neural networks as a directed graph of neurons

Take-away-messages of section: *"Dynamics of Artificial Neural Networks in General"*

E You should now be able to ...

- explain the general structure of an artificial neural networks as a directed graph of neurons
- describe different connection architectures

Agenda

Dynamics of Artificial Neural Networks in General

Simple Feed Forward Network

Hands On

Training Artificial Neural Networks

Learning objective of section: *"Simple Feed Forward Network"*

In this section we will ...

- focus on a certain subset of neural networks - in which units are organised in layers
- discuss the universal approximation capabilities of feed-forward networks
- introduce online and batch learning

Feed-Forward Artificial Neural Network

- An *Artificial Neural Network* consist of ...
  - a set U of units
  - a set of connections $C \subseteq U \times U$,
    each labelled with a weight $w_{i,j} \in \mathbb{R}$
  - ...

Feed-Forward Artificial Neural Network

- An *Artificial Neural Network* consist of ...
  - a set U of units
  - a set of connections $C \subseteq U \times U$,
    each labelled with a weight $w_{i,j} \in \mathbb{R}$
  - ...

- In a *Feed-Forward Artificial Neural Network* ...
  - the units are organised in a sequence of n disjoint sub-sets $U_1, \ldots, U_n$ (called *layers*) with

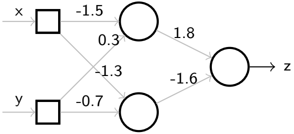
$$U = \bigcup_i U_i \quad \text{and} \quad U_i \cap U_j = \emptyset \quad \text{for all } i \neq j$$

  - all units from layer i are connected to all units in layer $i+1$:

$$\begin{aligned} C &= C_1 \cup \ldots \cup C_{n-1}, \quad \text{with} \\ C_i &= U_i \times U_{i+1} \end{aligned}$$

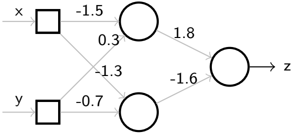
A Simple Feed-Forward Network

- The following network consists of 3 layers:
  - U_1 - called the input layer (with two units labelled x and y)
  - U_2 - called the hidden layer (with two unlabelled units)
  - U_3 - called the output layer (with the single unit labelled z)



A Simple Feed-Forward Network

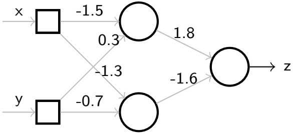
- The following network consists of 3 layers:
  - U_1 - called the input layer (with two units labelled x and y)
  - U_2 - called the hidden layer (with two unlabelled units)
  - U_3 - called the output layer (with the single unit labelled z)



- What does this network compute?

A Simple Feed-Forward Network

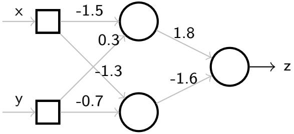
- The following network consists of 3 layers:
  - U_1 - called the input layer (with two units labelled x and y)
  - U_2 - called the hidden layer (with two unlabelled units)
  - U_3 - called the output layer (with the single unit labelled z)



- What does this network compute?
  - We do not know yet, because it is not specified how inputs, weights, etc. are combined

A Simple Feed-Forward Network

- The following network consists of 3 layers:
  - U_1 - called the input layer (with two units labelled x and y)
  - U_2 - called the hidden layer (with two unlabelled units)
  - U_3 - called the output layer (with the single unit labelled z)



- What does this network compute?
  - We do not know yet, because it is not specified how inputs, weights, etc. are combined
  - But we know the signature of the overall network function:

$$\mathcal{N} : \mathbb{R}^2 \to \mathbb{R}$$

Units of Connectionist Systems – Input Function

- *Input function* I (aka *activation function*): maps n-dimensional inputs $\vec{x}$ and n-dimensional weights $\vec{w}$ to an activation potential p:

$$I : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

$$(\vec{x}, \vec{w}) \mapsto \sum_{i=1}^{n} (x_i - w_i)^2 \qquad \textit{(squared distance)}$$

Units of Connectionist Systems – Input Function

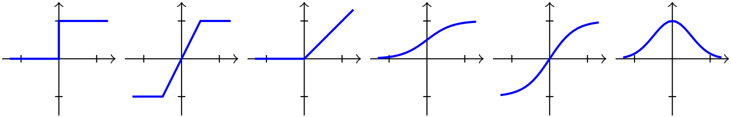
- *Input function* I (aka *activation function*): maps n-dimensional inputs $\vec{x}$ and n-dimensional weights $\vec{w}$ to an activation potential p:

$$
\begin{aligned}
I : \mathbb{R}^n \times \mathbb{R}^n &\to \mathbb{R} \\
(\vec{x}, \vec{w}) &\mapsto \sum_{i=1}^{n} (x_i - w_i)^2 && \textit{(squared distance)} \\
(\vec{x}, \vec{w}) &\mapsto \sum_{i=1}^{n} x_i \cdot w_i && \textit{(weighted sum)}
\end{aligned}
$$

- Most frameworks (TensorFlow, Keras , PyTorch) use the weighted sum by default

Units of Connectionist Systems – Output Function

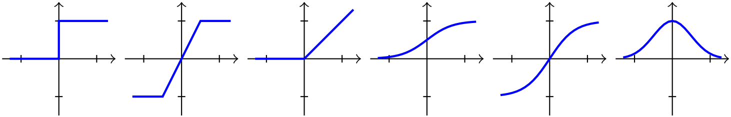
- *Output function* (aka *activation function*): maps the potential p and bias b to the output o. I.e. computes $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$



threshold	ramp	ReLU	sigmoidal	tanh	Gaussian
$(b = 0)$	$(b = 0)$	$(b = 0)$	$(b = 0)$	$(b = 0)$	$(b = 1)$

Units of Connectionist Systems – Output Function

- *Output function* (aka *activation function*): maps the potential p and bias b to the output o. I.e. computes $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$

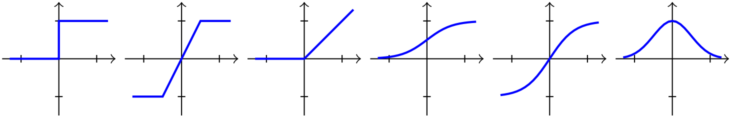


threshold ($b = 0$) | ramp ($b = 0$) | ReLU ($b = 0$) | sigmoidal ($b = 0$) | tanh ($b = 0$) | Gaussian ($b = 1$)

- Until about 2000, the sigmoidal function (or the tanh) was the most prominent activation function

Units of Connectionist Systems – Output Function

- *Output function* (aka *activation function*): maps the potential p and bias b to the output o. I.e. computes $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$

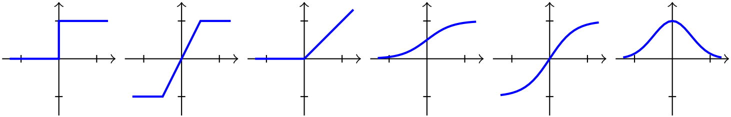


threshold	ramp	ReLU	sigmoidal	tanh	Gaussian
$(b = 0)$	$(b = 0)$	$(b = 0)$	$(b = 0)$	$(b = 0)$	$(b = 1)$

- Until about 2000, the sigmoidal function (or the tanh) was the most prominent activation function
- A [`Radial basis function network`] relies on the gaussian function (with the euclidean distance as input function)

Units of Connectionist Systems – Output Function

- *Output function* (aka *activation function*): maps the potential p and bias b to the output o. I.e. computes $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$

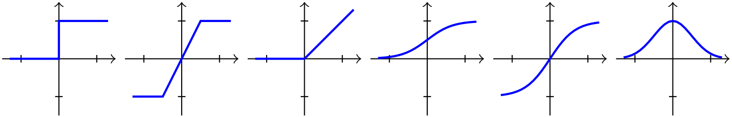


threshold ($b = 0$) | ramp ($b = 0$) | ReLU ($b = 0$) | sigmoidal ($b = 0$) | tanh ($b = 0$) | Gaussian ($b = 1$)

- Until about 2000, the sigmoidal function (or the tanh) was the most prominent activation function
- A [`Radial basis function network`] relies on the gaussian function (with the euclidean distance as input function)
- State of the art systems mostly use the ReLU function

Units of Connectionist Systems – Output Function

- *Output function* (aka *activation function*): maps the potential p and bias b to the output o. I.e. computes $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$



threshold	ramp	ReLU	sigmoidal	tanh	Gaussian
$(b = 0)$	$(b = 0)$	$(b = 0)$	$(b = 0)$	$(b = 0)$	$(b = 1)$

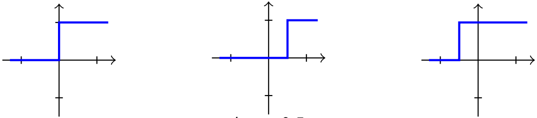
- Until about 2000, the sigmoidal function (or the tanh) was the most prominent activation function
- A [`Radial basis function network`] relies on the gaussian function (with the euclidean distance as input function)
- State of the art systems mostly use the ReLU function
- List of possible functions: [`Activation function`]

Output-functions and Bias

- *Output function A* maps potential p and bias b to output o
- Note: bias $\neq$ threshold
  - A threshold has to be exceed in order to activate a unit
  - A bias is added to the input
  - I.e., bias $\approx$ -threshold
- Threshold function:

$$\theta(p, b) = \begin{cases} 1 & \text{if } p + b \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

- Shape for different values of b:



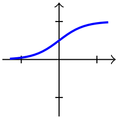
$b = 0$ $b = -0.5$ $b = 0.5$

Output-functions and Bias

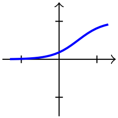
- Sigmoidal function:

$$f(x, b) = \mathrm{sig}(x + b) = \frac{1}{1.0 + e^{-(x+b)}}$$

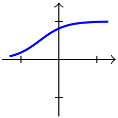
- Shape for different values of b:



$b = 0$



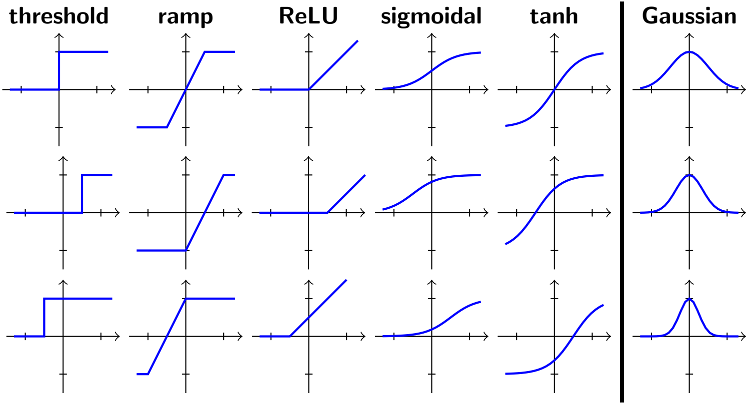
$b = -0.5$



$b = 0.5$

Output-functions and Bias

- Shape of activation functions (as function over p) for three different values for b:



Ridge output functions

A *ridge function* is some univariate function g applied to a linear combination of the inputs: $f = g(a \cdot x + b)$.

- *Linear activation:*

$$o(p, b) = p + b$$

- *ReLU activation:*

$$o(p, b) = \max(0, p + b)$$

- *Sigmoidal functions*, e.g., the *logistic function*:

$$o(p, b) = \frac{1}{1 + e^{-p-b}}$$

Ridge output functions

A *ridge function* is some univariate function g applied to a linear combination of the inputs: $f = g(a \cdot x + b)$.
Used together with the weighted sum input function.

- *Linear activation:*

$$o(p, b) = p + b \qquad o(\vec{x}, \vec{w}, b) = \vec{w} \cdot \vec{x} + b$$

- *ReLU activation:*

$$o(p, b) = \max(0, p + b) \qquad o(\vec{x}, \vec{w}, b) = \max(0, \vec{w} \cdot \vec{x} + b)$$

- *Sigmoidal functions*, e.g., the *logistic function*:

$$o(p, b) = \frac{1}{1 + e^{-p-b}} \qquad o(\vec{x}, \vec{w}, b) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{x} - b}}$$

Radial basis output functions

A *radial basis function* is some real-valued function whose value at each point depends only on the distance between that point and some other fixed point:

- *Gaussian:*

$$o(p, b) = e^{-\frac{p^2}{b^2}}$$

Radial basis output functions

A *radial basis function* is some real-valued function whose value at each point depends only on the distance between that point and some other fixed point:

- *Gaussian:*

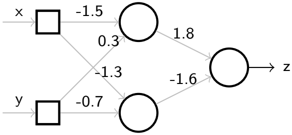
$$o(p, b) = e^{-\frac{p^2}{b^2}} \qquad o(\vec{x}, \vec{w}, b) = e^{-\frac{(\vec{x}-\vec{w})^2}{b^2}}$$

Used together with (squared) distance input function.

Folding output functions

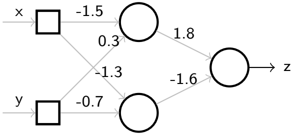
- A *fold function* (reduce, aggregate, compress) is a function which combines it's inputs recursively.
- Fold functions combine the outputs of multiple units $p_1, \ldots, p_n$ within a layer. Instead of $o : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, they compute a function $\mathbb{R}^n \to \mathbb{R}^m$.
- Examples:
  - *Pooling:* Maximum, Minimum, Mean, ... of a group of units
  - *Softmax:* Renormalise all outputs to form a probability distribution: $o_i = \frac{e^{o_i}}{\sum_j e^{o_j}}$
- They are either implemented as activation function or as additional layer:
  - `tf.keras.layers.Dense(10, activation = "softmax")`
  - `tf.keras.layers.MaxPooling2D(pool_size=(3, 3))`

Dynamics of a Network



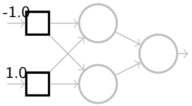
	Input F.	Output F.
input	p set from outside	$o = p$
hidden	$p = \sum_n (i_n - w_n)^2$	$o = e^{-p^2}$
output	$p = \sum_n (i_n * w_n)$	$o = p$

Dynamics of a Network

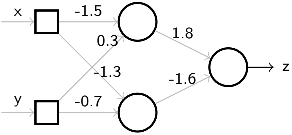


	Input F.	Output F.
input	p set from outside	$o = p$
hidden	$p = \sum_n (i_n - w_n)^2$	$o = e^{-p^2}$
output	$p = \sum_n (i_n * w_n)$	$o = p$

set input:

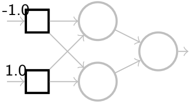


Dynamics of a Network

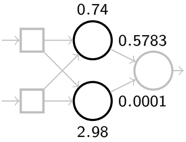


	Input F.	Output F.
input	p set from outside	$o = p$
hidden	$p = \sum_n (i_n - w_n)^2$	$o = e^{-p^2}$
output	$p = \sum_n (i_n * w_n)$	$o = p$

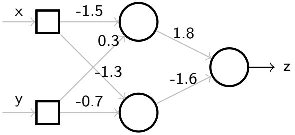
set input:



hidden layer:



Dynamics of a Network

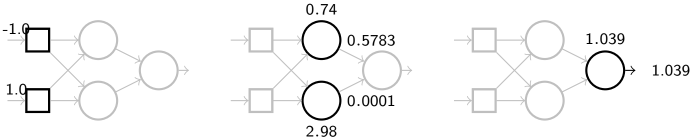


	Input F.	Output F.
input	p set from outside	$o = p$
hidden	$p = \sum_n (i_n - w_n)^2$	$o = e^{-p^2}$
output	$p = \sum_n (i_n * w_n)$	$o = p$

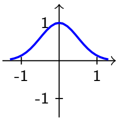
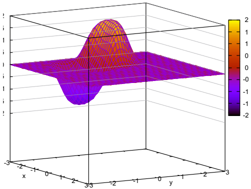
set input:

hidden layer:

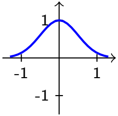
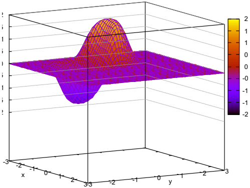
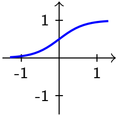
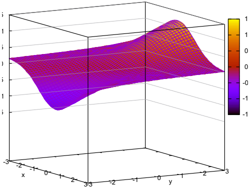
output:



Dynamics of a Network

Activation function	**Output function**	**Result**
$p = \sum_n (i_n - w_n)^2$		

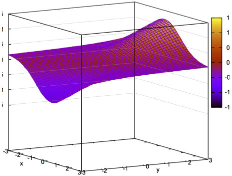
Dynamics of a Network

Activation function	**Output function**	**Result**
$p = \sum_n (i_n - w_n)^2$		
$p = \sum_n i_n \cdot w_n$		

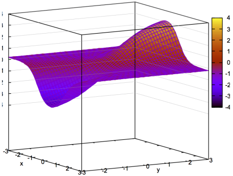
Dynamics of a Network (ctd.)

- Using the weighted sum as input function, the behaviour only depends on the activation function:

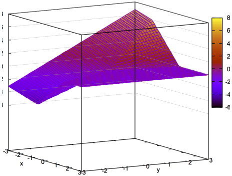
Sigmoidal



TanH

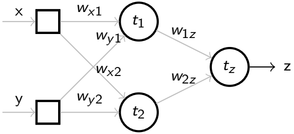


ReLU



A Simple Example

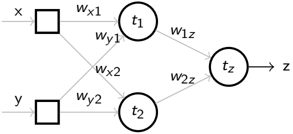
- Let the following 3-layer feed-forward network be given:



- Input units:
  - *Input function:* input value set from outside
  - *Output function:* identity
- Hidden units:
  - *Input function:* weighted sum
  - *Output function:* sigmoidal
- Output units:
  - *Input function:* weighted sum
  - *Output function:* sigmoidal

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:

$$o_x = x \qquad\qquad o_y = y$$

- Hidden units:

$$i_1 = w_{x1} * o_x + w_{y1} * o_y \qquad o_1 = \mathrm{sig}(i_1 + t_1)$$

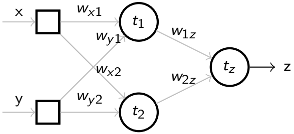
$$i_2 = w_{x2} * o_x + w_{y2} * o_y \qquad o_1 = \mathrm{sig}(i_2 + t_2)$$

- Output units:

$$i_z = w_{1z} * o_1 + w_{2z} * o_2 \qquad o_z = \mathrm{sig}(i_z + t_z)$$

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:

$$z = o_z = \text{sig}\left(i_z + t_z\right)$$

- Hid

$$i_1 = w_{x1} * o_x + w_{y1} * o_y \qquad o_1 = \text{sig}(i_1 + t_1)$$

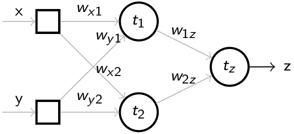
$$i_2 = w_{x2} * o_x + w_{y2} * o_y \qquad o_1 = \text{sig}(i_2 + t_2)$$

- Output units:

$$i_z = w_{1z} * o_1 + w_{2z} * o_2 \qquad o_z = \text{sig}(i_z + t_z)$$

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:
- Hid

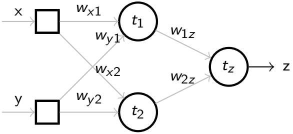
$$z = \mathrm{sig}(w_{1z} * o_1 + w_{2z} * o_2 + t_z)$$

- Output units:

$$i_z = w_{1z} * o_1 + w_{2z} * o_2 \qquad o_z = \mathrm{sig}(i_z + t_z)$$

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:

- Hid

$$z = \mathrm{sig}(w_{1z} * \mathrm{sig}(i_1 + t_1) + w_{2z} * \mathrm{sig}(i_2 + t_2) + t_z)$$

)

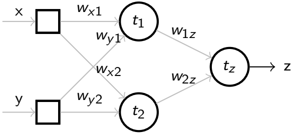
)

- Output units:

$$i_z = w_{1z} * o_1 + w_{2z} * o_2 \qquad o_z = \mathrm{sig}(i_z + t_z)$$

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:
- Hid

$$z = \mathrm{sig}(w_{1z} * \mathrm{sig}\left(w_{x1} * o_x + w_{y1} * o_y + t_1\right) + w_{2z} * \mathrm{sig}\left(w_{x2} * o_x + w_{y2} * o_y + t_2\right) + t_z)$$

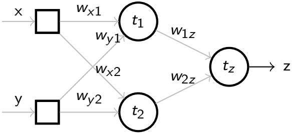
)
)

- Output units:

$$i_z = w_{1z} * o_1 + w_{2z} * o_2 \qquad o_z = \mathrm{sig}(i_z + t_z)$$

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:
- Hid
- Ou

$$z = \mathrm{sig}(w_{1z} * \mathrm{sig}\left(w_{x1} * o_x + w_{y1} * o_y + t_1\right) + w_{2z} * \mathrm{sig}\left(w_{x2} * o_x + w_{y2} * o_y + t_2\right) + t_z)$$

$$\text{with } \mathrm{sig}(x) = \frac{1}{1.0 + e^{-x}}$$

Network Function

- The *Network input-output function* $\mathcal{N}$ computes the output for a given input vector $\vec{x}$:

$$\mathcal{N}(\vec{x}) := f_{w_{ij},t_i,\dots}(\vec{x})$$

- It depends on:
  - *hyper-parameters* of the network: structure, input and output functions of all units, update schema
  - *(trainable) parameters* of the network: weights and bias values
- The network function $\mathcal{N}$ is well defined for feed-forward networks

Feed-Forward Artificial Neural Network

recap

- An *Artificial Neural Network* consist of ...
  - a set U of units
  - a set of connections $C \subseteq U \times U$, each labelled with a weight $w_{i,j} \in \mathbb{R}$
  - ...

- In a *Feed-Forward Artificial Neural Network* (FFN) ...
  - the units are organised in a sequence of n disjoint sub-sets $U_1, \ldots, U_n$ (called *layers*) with

$$U = \bigcup_i U_i \quad \text{and} \quad U_i \cap U_j = \emptyset \quad \text{for all } i \neq j$$

  - all units from layer i are connected to all units in layer $i + 1$:

$$C = C_1 \cup \ldots \cup C_{n-1}, \quad \text{with}$$
$$C_i = U_i \times U_{i+1}$$

Network Function for FFNs

- The *Network input-output function* $\mathcal{N}$ computes the output for a given input vector $\vec{x}$:

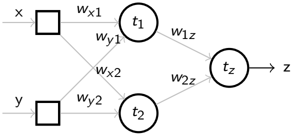
$$\mathcal{N}(\vec{x}) := \mathcal{L}_n(\vec{x})$$
$$\mathcal{L}_1(\vec{x}) := \vec{x}$$
$$\mathcal{L}_i(\vec{x}) := \vec{A}_i(\vec{I}_i(W_i, \mathcal{L}_{i-1}(\vec{x})), \vec{t}_i)$$

with

- n being the number of layers, and n_i being the size of layer i
- W_i being the weight matrix between layer $i-1$ and layer i, i.e., a matrix of shape $n_{i-1} \times n_i$
- t_i being the vector of biases for layer i
- $\vec{I}_i : \mathbb{R}^{n_{i-1}} \times \mathbb{R}^{n_{i-1} * n_i} \to \mathbb{R}^{n_i}$ being the vectorised version of the *input function* I_i for layer i
- $\vec{A}_i : \mathbb{R}^{n_i} \times \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ being the vectorised version of the *activation function* A_i for layer i

Network Function for weighted sum FFNs

- Let the following 3-layer feed-forward network be given:



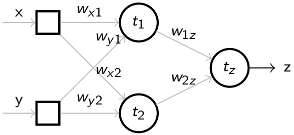
- The network function can be computed as follows:

$$I_1 \qquad \begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix}$$

$$\begin{pmatrix} x & y \end{pmatrix}$$

Network Function for weighted sum FFNs

- Let the following 3-layer feed-forward network be given:

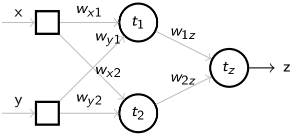


- The network function can be computed as follows:

$$\begin{matrix} & \begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix} \\ \begin{pmatrix} x & y \end{pmatrix} & \begin{pmatrix} i_1 & i_2 \end{pmatrix} \end{matrix}$$

Network Function for weighted sum FFNs

- Let the following 3-layer feed-forward network be given:

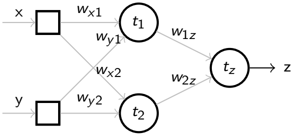


- The network function can be computed as follows:

$$\begin{array}{cccc} & \begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix} & & \\ \begin{pmatrix} x & y \end{pmatrix} & \begin{pmatrix} i_1 & i_2 \end{pmatrix} & \rightsquigarrow_{A_1} & \begin{pmatrix} o_1 & o_2 \end{pmatrix} \end{array}$$

Network Function for weighted sum FFNs

- Let the following 3-layer feed-forward network be given:

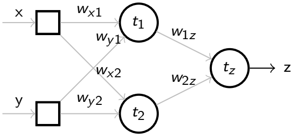


- The network function can be computed as follows:

$$\begin{array}{ccccc} & \begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix} & & I_2 & \begin{pmatrix} w_{1z} \\ w_{2z} \end{pmatrix} \\ \begin{pmatrix} x & y \end{pmatrix} & \begin{pmatrix} i_1 & i_2 \end{pmatrix} & \rightsquigarrow_{A_1} & \begin{pmatrix} o_1 & o_2 \end{pmatrix} & \end{array}$$

Network Function for weighted sum FFNs

- Let the following 3-layer feed-forward network be given:

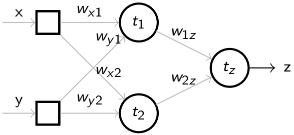


- The network function can be computed as follows:

$$\begin{pmatrix} x & y \end{pmatrix} \qquad \overset{\begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix}}{\begin{pmatrix} i_1 & i_2 \end{pmatrix}} \qquad \rightsquigarrow_{A_1} \quad \begin{pmatrix} o_1 & o_2 \end{pmatrix} \quad \overset{\begin{pmatrix} w_{1z} \\ w_{2z} \end{pmatrix}}{\begin{pmatrix} i_z \end{pmatrix}}$$

Network Function for weighted sum FFNs

- Let the following 3-layer feed-forward network be given:



- The network function can be computed as follows:

$$\begin{array}{ccccccc} & \begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix} & & & \begin{pmatrix} w_{1z} \\ w_{2z} \end{pmatrix} & & \\ \begin{pmatrix} x & y \end{pmatrix} & \begin{pmatrix} i_1 & i_2 \end{pmatrix} & \rightsquigarrow_{A_1} & \begin{pmatrix} o_1 & o_2 \end{pmatrix} & \begin{pmatrix} i_z \end{pmatrix} & \rightsquigarrow_{A_2} & \begin{pmatrix} o_z \end{pmatrix} \end{array}$$

Take-away-messages of section: *"Simple Feed Forward Network"*

E You should now be able to ...

- describe the architecture of a feed-forward network

Take-away-messages of section: *"Simple Feed Forward Network"*

E You should now be able to ...

- describe the architecture of a feed-forward network
- describe and explain the differences between different input and output functions

Take-away-messages of section: *“Simple Feed Forward Network”*

E You should now be able to ...

- describe the architecture of a feed-forward network
- describe and explain the differences between different input and output functions
- name, explain and draw some prominent input and output functions

Take-away-messages of section: *“Simple Feed Forward Network”*

E You should now be able to ...

- describe the architecture of a feed-forward network
- describe and explain the differences between different input and output functions
- name, explain and draw some prominent input and output functions
- describe the intuition behind computing the network function of a FFN in matrix formulation

Agenda

Dynamics of Artificial Neural Networks in General

Simple Feed Forward Network

Hands On

Training Artificial Neural Networks

https://playground.tensorflow.org

Take-away-messages of section: *"Hands On"*

E You should now be able to ...

- describe the dynamics of a neural network during the training phase

Take-away-messages of section: *"Hands On"*

E You should now be able to ...

- describe the dynamics of a neural network during the training phase
- able to explain the effect of good features on the overall performance

Agenda

Dynamics of Artificial Neural Networks in General

Simple Feed Forward Network

Hands On

Training Artificial Neural Networks

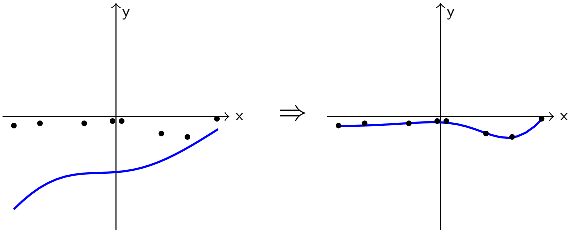
Learning objective of section: *"Training Artificial Neural Networks"*

In this section we will ...

- discuss the idea of adaptation by gradient descent
- define the network function wrt. trainable parameters
- derive the equations to adapt the weights and thresholds of a simple network by hand

Training Artificial Neural Networks

- How can we train a network to represent a function given as a set of samples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$?



- Learning as generalization.

Training a Neural Network

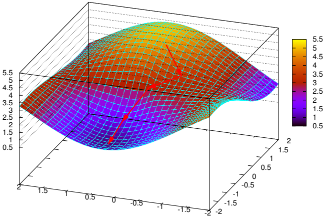
Training a Neural Network

Result: A trained network
Input: A Network N, a set of training data D

```
Let π_N be the set of parameters of N:
  weights and thresholds
Initialise all parameters π_N, randomly
repeat
  | Compute error E wrt. D and current parameters π_N
  | Modify parameters π_N such that the error decreases
until E is acceptable
return The modified network N
```

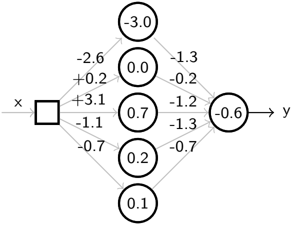
Backpropagation

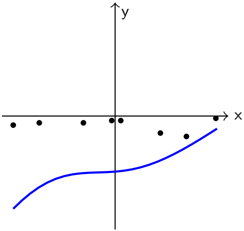
- Let a set of samples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ be given.
- Error of the network: $E = \sum_i (\mathcal{N}(x_i) - y_i)^2$.
  (with $\mathcal{N}(x_i)$ being the output of the network for the input x_i)
- Idea: minimise E by gradient descent.



A sample run ...

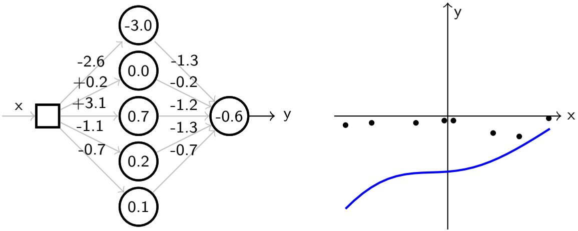
Prior training:



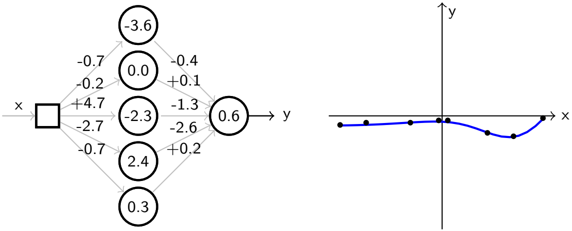


A sample run ...

Prior training:



After training:



Training Schemes

- **Online learning:** All parameters are adapted after presenting a single example
- **Batch learning:** Changes to the parameters are accumulated and parameters are adapted after all samples from a batch have been processed

Desired vz. Actual Input-Output Behaviour

- Actual input-output-behaviour as just derived:

$$
\begin{aligned}
z &= \operatorname{sig}\left(w_{1z} * \operatorname{sig}\left(w_{x1} * o_x + w_{y1} * o_y + t_1\right) + w_{2z} * \operatorname{sig}\left(w_{x2} * o_x + w_{y2} * o_y + t_2\right) + t_z\right) \\
&= \frac{1}{1.0 + e^{-\left(w_{1z} * \frac{1}{1.0 + e^{-(w_{x1} * x + w_{y1} * y + t_1)}} + w_{2z} * \frac{1}{1.0 + e^{-(w_{x2} * x + w_{y2} * y + t_2)}} + t_z\right)}}
\end{aligned}
$$

Desired vz. Actual Input-Output Behaviour

- Actual input-output-behaviour as just derived:

$$
\begin{aligned}
z &= \operatorname{sig}\left(w_{1z} * \operatorname{sig}\left(w_{x1} * o_x + w_{y1} * o_y + t_1\right) + w_{2z} * \operatorname{sig}\left(w_{x2} * o_x + w_{y2} * o_y + t_2\right) + t_z\right) \\
&= \frac{1}{1.0 + e^{-\left(w_{1z} * \frac{1}{1.0 + e^{-(w_{x1} * x + w_{y1} * y + t_1)}} + w_{2z} * \frac{1}{1.0 + e^{-(w_{x2} * x + w_{y2} * y + t_2)}} + t_z\right)}}
\end{aligned}
$$

- What is the desired output of the network?

Desired vz. Actual Input-Output Behaviour

- Actual input-output-behaviour as just derived:

$$z = \text{sig}\left(w_{1z} * \text{sig}\left(w_{x1} * o_x + w_{y1} * o_y + t_1\right) + w_{2z} * \text{sig}\left(w_{x2} * o_x + w_{y2} * o_y + t_2\right) + t_z\right)$$

$$= \frac{1}{1.0 + e^{-\left(w_{1z} * \frac{1}{1.0 + e^{-(w_{x1} * x + w_{y1} * y + t_1)}} + w_{2z} * \frac{1}{1.0 + e^{-(w_{x2} * x + w_{y2} * y + t_2)}} + t_z\right)}}$$

- What is the desired output of the network?
- Usually it is “specified” in form of training samples D:

x	0.0	0.1	0.0	1.0
y	0.0	0.0	1.0	1.0
z	0.0	1.0	1.0	1.0

$$D := \{((0,0),0), ((0,1),1), ((1,0),1), ((1,1),1)\}$$

Desired vz. Actual Input-Output Behaviour

- Actual input-output-behaviour as just derived:

$$
\begin{aligned}
z &= \text{sig}\left(w_{1z} * \text{sig}\left(w_{x1} * o_x + w_{y1} * o_y + t_1\right) + w_{2z} * \text{sig}\left(w_{x2} * o_x + w_{y2} * o_y + t_2\right) + t_z\right) \\
&= \frac{1}{1.0 + e^{-\left(w_{1z} * \frac{1}{1.0 + e^{-(w_{x1} * x + w_{y1} * y + t_1)}} + w_{2z} * \frac{1}{1.0 + e^{-(w_{x2} * x + w_{y2} * y + t_2)}} + t_z\right)}}
\end{aligned}
$$

- What is the desired output of the network?
- Usually it is “specified” in form of training samples D:

x	0.0	0.1	0.0	1.0
y	0.0	0.0	1.0	1.0
z	0.0	1.0	1.0	1.0

$$D := \{((0,0),0), ((0,1),1), ((1,0),1), ((1,1),1)\}$$

- Goal of the training: modify the parameters of the network function, i.e.: weights w_{ij} and bias values t_i

Desired vz. Actual Input-Output Behaviour

- Actual input-output-behaviour as just derived:

$$
\begin{aligned}
z &= \operatorname{sig}\left(w_{1z} * \operatorname{sig}\left(w_{x1} * o_x + w_{y1} * o_y + t_1\right) + w_{2z} * \operatorname{sig}\left(w_{x2} * o_x + w_{y2} * o_y + t_2\right) + t_z\right) \\
&= \frac{1}{1.0 + e^{-\left(w_{1z} * \frac{1}{1.0 + e^{-(w_{x1} * x + w_{y1} * y + t_1)}} + w_{2z} * \frac{1}{1.0 + e^{-(w_{x2} * x + w_{y2} * y + t_2)}} + t_z\right)}}
\end{aligned}
$$

- What is the desired output of the network?
- Usually it is “specified” in form of training samples D:

x	0.0	0.1	0.0	1.0
y	0.0	0.0	1.0	1.0
z	0.0	1.0	1.0	1.0

$$D := \{((0,0),0), ((0,1),1), ((1,0),1), ((1,1),1)\}$$

- Goal of the training: modify the parameters of the network function, i.e.: weights w_{ij} and bias values t_i
- Please note: the hyper-parameter (e.g., structure and functions) are not adjusted as part of the training

Training a Neural Network

recap

Training a Neural Network

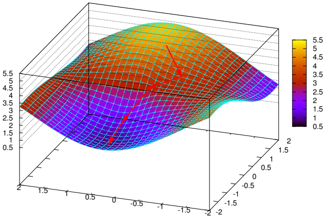
Result: A trained network
Input: A Network N, a set of training data D

Let π_N be the set of parameters of N:
  weights and thresholds
Initialise all parameters π_N, randomly
repeat
  Compute error E wrt. D and current parameters π_N
  Modify parameters π_N such that the error decreases
until E *is acceptable*
return *The modified network* N

Backpropagation

recap

- Let a set of samples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ be given.
- Error of the network defined by a loss function, e.g., quadratic loss $E = \sum_i (\mathcal{N}(x_i) - y_i)^2$.
  (with $\mathcal{N}(x_i)$ being the output of the network for the input x_i)
- Idea: minimise E by gradient descent.



Gradient descent

"Gradient descent is an [...] iterative optimization algorithm for finding a local minimum of a differentiable function. To find a local minimum [...] we take steps proportional to the negative of the gradient [...] of the function at the current point. [...] Gradient descent was originally proposed by Cauchy in 1847."

[Gradient descent]

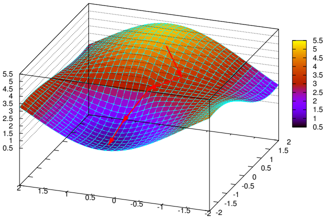
Augustin-Louis Cauchy

- *21 Aug. 1789, †23 May 1857
- French mathematician
- Some of his influential work:
  - Analysis: formal definition of continuity based on infinitesimals, *Cours d'Analyse* (1821)
  - Converging sequences: Cauchy sequences
  - Probability theory: Cauchy distributions

[Augustin-Louis Cauchy]

Gradient Descent

- *Gradient descent* is an optimisation algorithm to find a local minimum of a given function
- Idea:
  1. Select a starting position
  2. Compute the gradient of the function at the current point
  3. Make a step towards the steepest descent (down hill)
  4. Repeat step 2 and 3 until satisfied



Gradient Descent (GD)

Finding a Local Minimum by Gradient Descent

Input: Differentiable function $f : \mathbb{R}^n \to \mathbb{R}$
Input: Starting point $\vec{x} \in \mathbb{R}^n$, Step size $\gamma \in \mathbb{R}^+$
Result: A local minimum

repeat
    Compute $\nabla f(\vec{x})$ with $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$
    Set $\vec{x} := \vec{x} - \gamma \nabla f(\vec{x})$
until $\nabla f(\vec{x}) = \vec{0}$ *or given number of iterations*
return $\vec{x}$, *which is a minimum iff* $\nabla f(\vec{x}) = \vec{0}$

Gradient Descent (GD)

Finding a Local Minimum by Gradient Descent

Input: Differentiable function $f : \mathbb{R}^n \to \mathbb{R}$
Input: Starting point $\vec{x} \in \mathbb{R}^n$, Step size $\gamma \in \mathbb{R}^+$
Result: A local minimum

repeat
  Compute $\nabla f(\vec{x})$ with $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$
  Set $\vec{x} := \vec{x} - \gamma \nabla f(\vec{x})$
until $\nabla f(\vec{x}) = \vec{0}$ *or given number of iterations*
return $\vec{x}$, *which is a minimum iff* $\nabla f(\vec{x}) = \vec{0}$

- for certain function classes (e.g., convex and Lipschitz continuous) and suitable γ, GD converges to a local minimum

Gradient Descent (GD)

Finding a Local Minimum by Gradient Descent

Input: Differentiable function $f : \mathbb{R}^n \to \mathbb{R}$
Input: Starting point $\vec{x} \in \mathbb{R}^n$, Step size $\gamma \in \mathbb{R}^+$
Result: A local minimum

repeat
  Compute $\nabla f(\vec{x})$ with $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$
  Set $\vec{x} := \vec{x} - \gamma \nabla f(\vec{x})$
until $\nabla f(\vec{x}) = \vec{0}$ *or given number of iterations*
return $\vec{x}$, *which is a minimum iff* $\nabla f(\vec{x}) = \vec{0}$

- for certain function classes (e.g., convex and Lipschitz continuous) and suitable γ, GD converges to a local minimum
- the step size γ can be different for every iteration

Gradient Descent (GD)

Finding a Local Minimum by Gradient Descent

Input: Differentiable function $f : \mathbb{R}^n \to \mathbb{R}$
Input: Starting point $\vec{x} \in \mathbb{R}^n$, Step size $\gamma \in \mathbb{R}^+$
Result: A local minimum

repeat
  Compute $\nabla f(\vec{x})$ with $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$
  Set $\vec{x} := \vec{x} - \gamma \nabla f(\vec{x})$
until $\nabla f(\vec{x}) = \vec{0}$ *or given number of iterations*
return $\vec{x}$, *which is a minimum iff* $\nabla f(\vec{x}) = \vec{0}$

- for certain function classes (e.g., convex and Lipschitz continuous) and suitable γ, GD converges to a local minimum
- the step size γ can be different for every iteration
- if f is convex, every local minimum is a global minimum

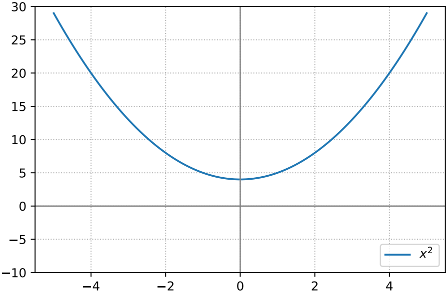
Rules for Partial Derivatives

do you recall?

Rule	F	$\partial F / \partial x$
Constant	c	0
Factor	$c \cdot f(x)$	$c \frac{\partial f(x)}{\partial x}$
Power rule	x^n	$n \cdot x^{n-1}$
Sum rule	$f(x) + g(x)$	$\frac{\partial f(x)}{\partial x} + \frac{\partial g(x)}{\partial x}$
Product rule	$f(x) \cdot g(x)$	$f \frac{\partial g(x)}{\partial x} + g \frac{\partial f(x)}{\partial x}$
Chain rule	$f(g(x))$	$\frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$
Exponential	e^x	e^x

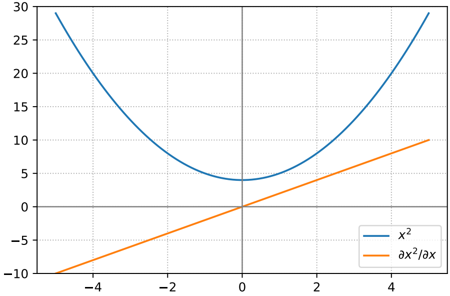
Gradient Descent by Example

- How to find the minimum of a function?
- Let the following function be given:



Gradient Descent by Example

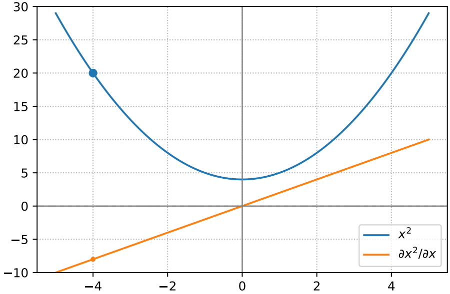
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

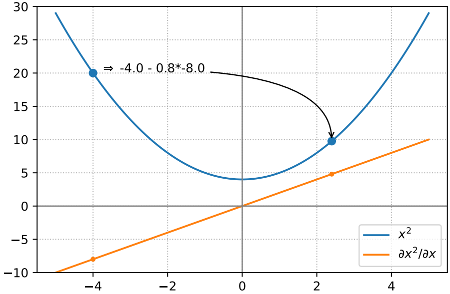
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

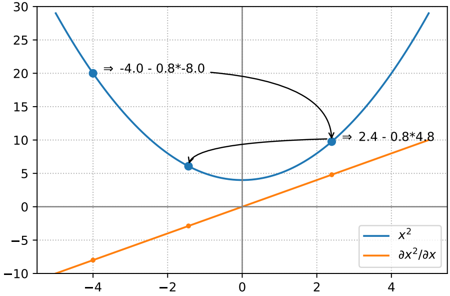
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

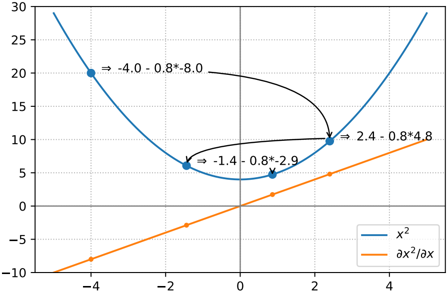
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

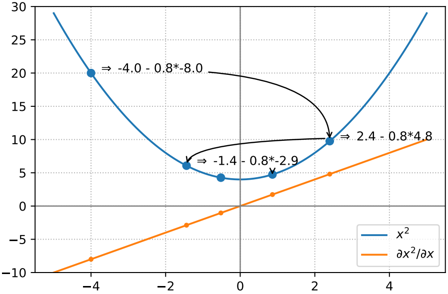
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

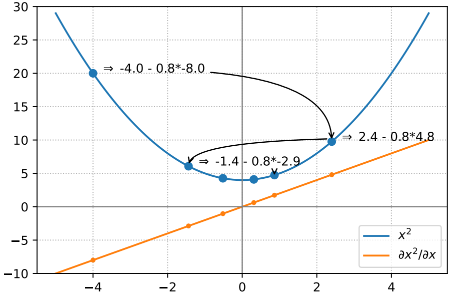
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

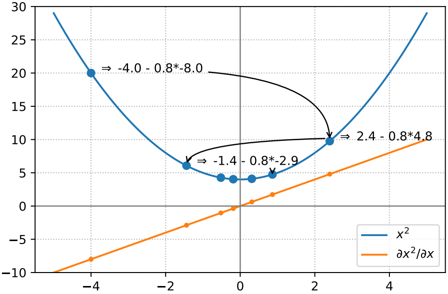
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

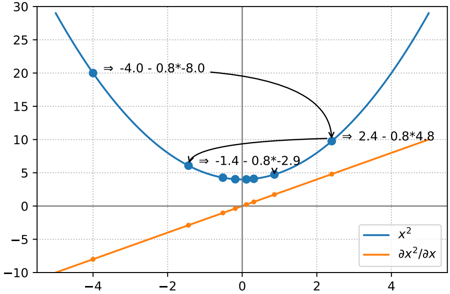
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

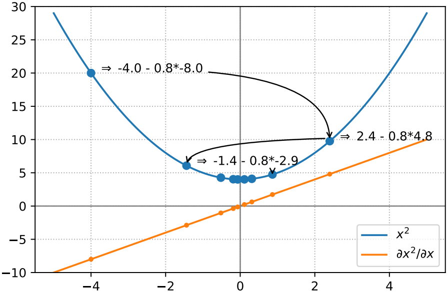
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

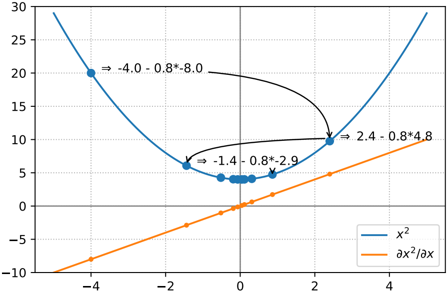
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

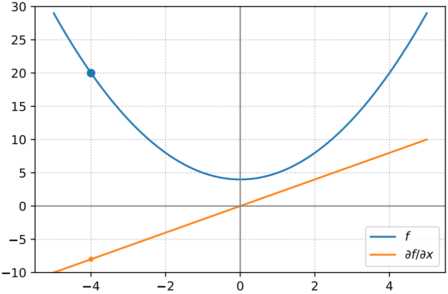
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

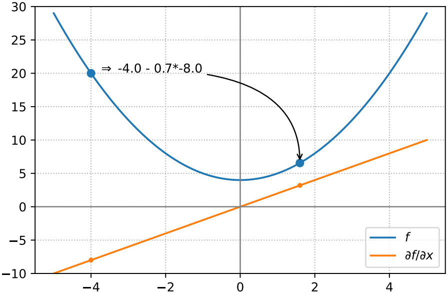
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

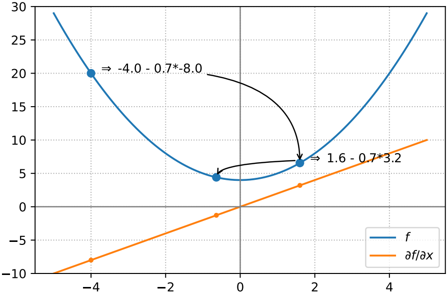
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

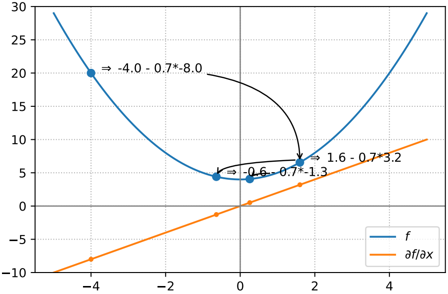
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

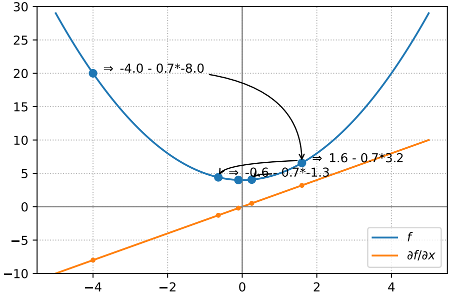
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

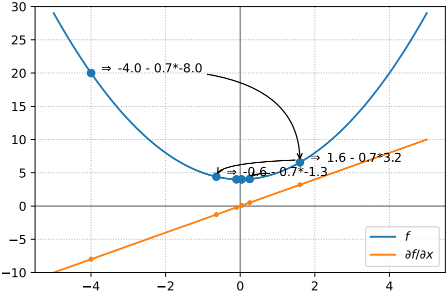
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

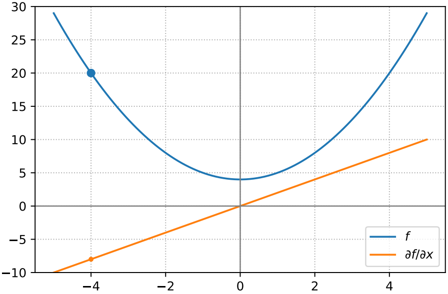
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

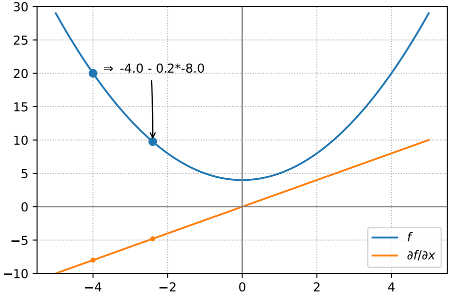
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

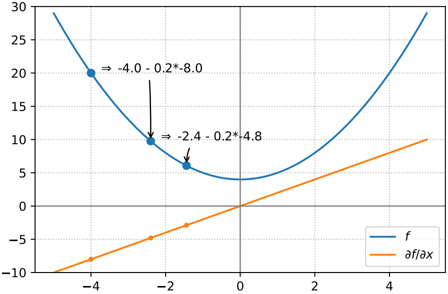
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

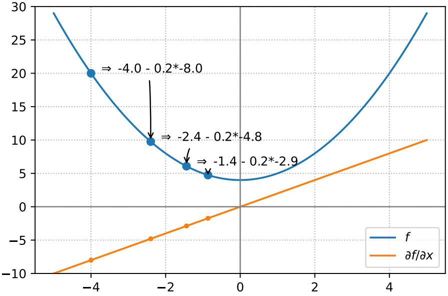
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

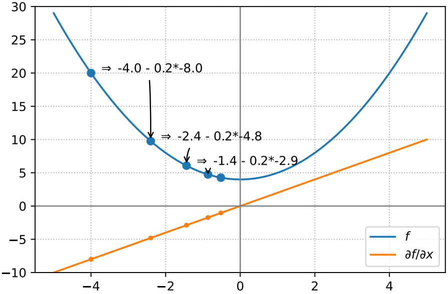
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

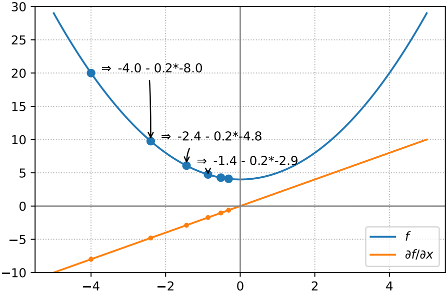
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

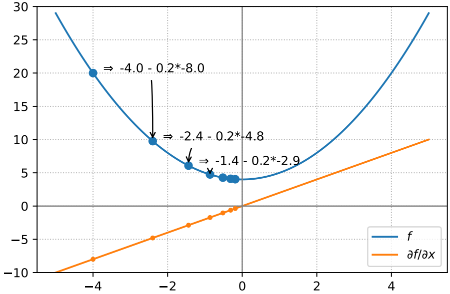
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

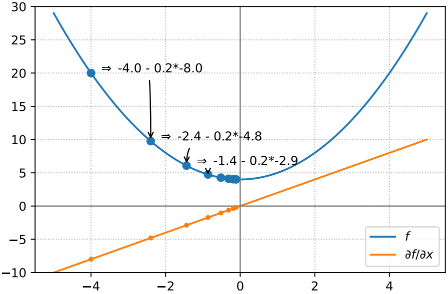
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$)

Gradient Descent by Example

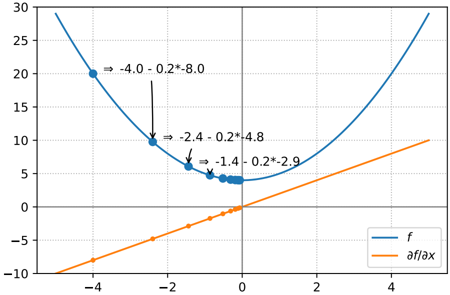
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

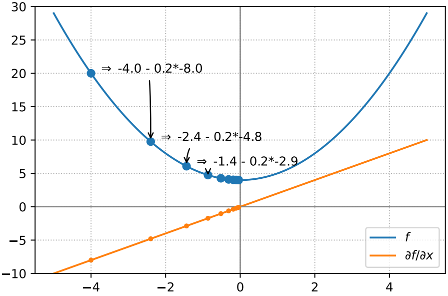
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

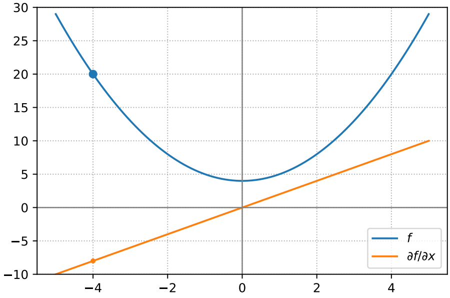
- How to find the minimum of a function?
- Let the following function be given:



- Lets find the minimum by going downhill
  - For this we compute the derivative
  - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example - ping pong

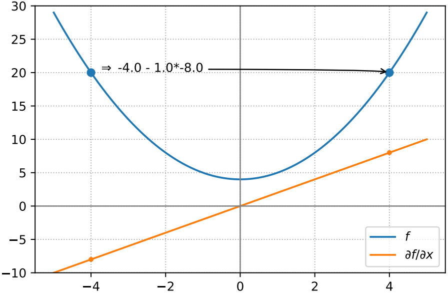
- But there might be combinations of function and step size for which this fails:



- There are strategies, e.g., stochastic gradient descent which solve this and similar problems (discussed later)

Gradient Descent by Example - ping pong

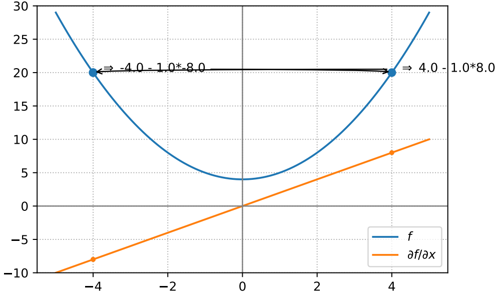
- But there might be combinations of function and step size for which this fails:



- There are strategies, e.g., stochastic gradient descent which solve this and similar problems (discussed later)

Gradient Descent by Example - ping pong

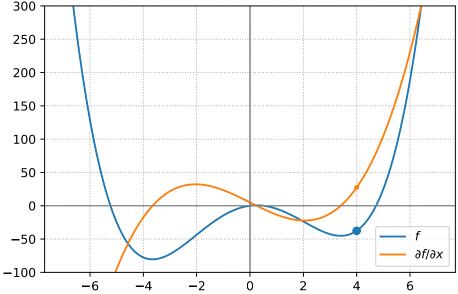
- But there might be combinations of function and step size for which this fails:



- There are strategies, e.g., stochastic gradient descent which solve this and similar problems (discussed later)

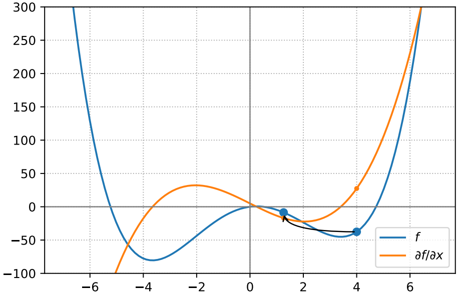
Gradient Descent by Example - exploding gradients

- But there are very steep functions with large derivatives:



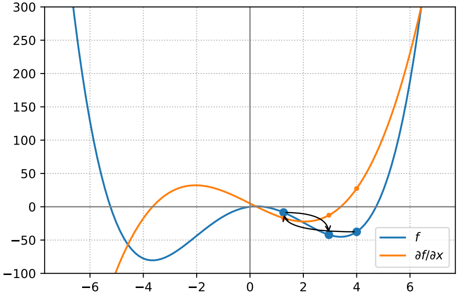
Gradient Descent by Example - exploding gradients

- But there are very steep functions with large derivatives:



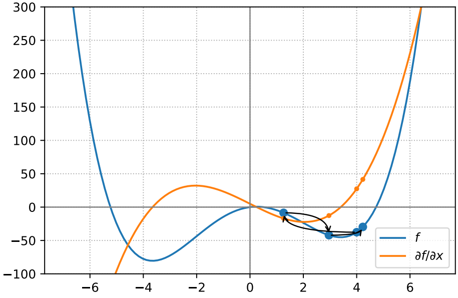
Gradient Descent by Example - exploding gradients

- But there are very steep functions with large derivatives:



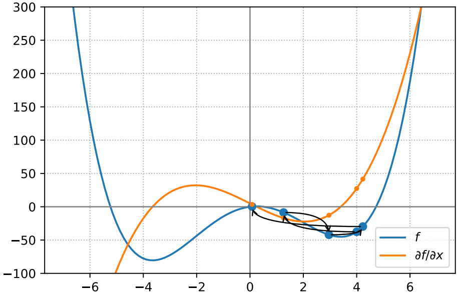
Gradient Descent by Example - exploding gradients

- But there are very steep functions with large derivatives:



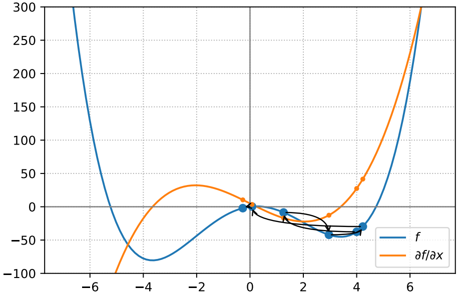
Gradient Descent by Example - exploding gradients

- But there are very steep functions with large derivatives:



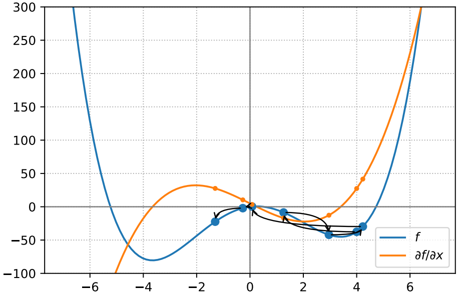
Gradient Descent by Example - exploding gradients

- But there are very steep functions with large derivatives:



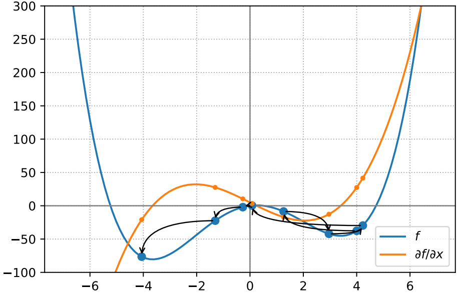
Gradient Descent by Example - exploding gradients

- But there are very steep functions with large derivatives:



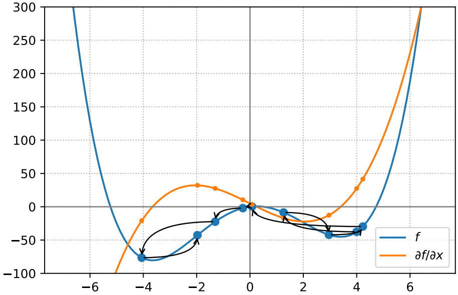
Gradient Descent by Example - exploding gradients

- But there are very steep functions with large derivatives:



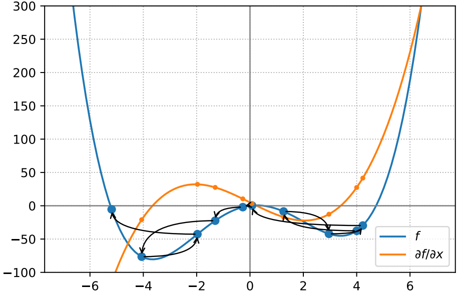
Gradient Descent by Example - exploding gradients

- But there are very steep functions with large derivatives:



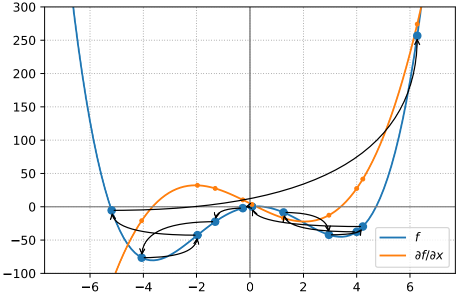
Gradient Descent by Example - exploding gradients

- But there are very steep functions with large derivatives:



Gradient Descent by Example - exploding gradients

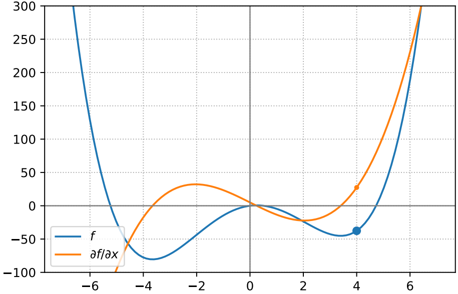
- But there are very steep functions with large derivatives:



- In certain situations, the gradients "explode"

Gradient Descent by Example - exploding gradients

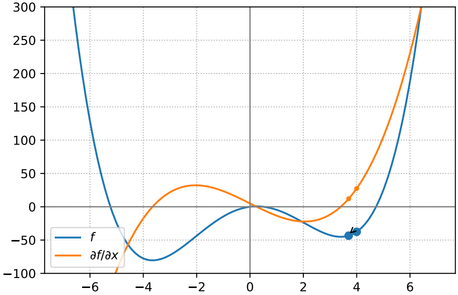
- But there are very steep functions with large derivatives:



- In certain situations, the gradients "explode"
- There are strategies, e.g., gradient clipping, which solve this and similar problems (discussed later)

Gradient Descent by Example - exploding gradients

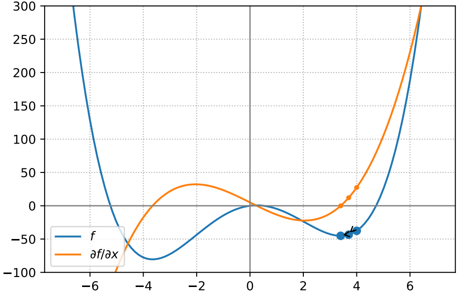
- But there are very steep functions with large derivatives:



- In certain situations, the gradients "explode"
- There are strategies, e.g., gradient clipping, which solve this and similar problems (discussed later)
- Limit the magnitude of the gradient to e.g., 0.3

Gradient Descent by Example - exploding gradients

- But there are very steep functions with large derivatives:



- In certain situations, the gradients "explode"
- There are strategies, e.g., gradient clipping, which solve this and similar problems (discussed later)
- Limit the magnitude of the gradient to e.g., 0.3

Gradient Descent by Example

GradientDescent.ipynb

The Sigmoidal function and it's derivative

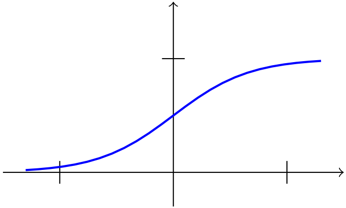
- First we need to compute the derivative of the sigmoidal

$$\mathrm{sig}(x) = \frac{1}{1.0 + e^{-x}}$$

$$\mathrm{sig}'(x) = \frac{\partial\, \mathrm{sig}(x)}{\partial x} =$$

T Compute the derivative of the sigmoidal function by hand!

- Shape of the sigmoidal



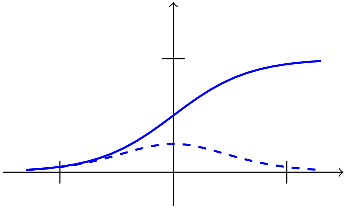
The Sigmoidal function and it's derivative

- First we need to compute the derivative of the sigmoidal

$$\mathrm{sig}(x) = \frac{1}{1.0 + e^{-x}}$$
$$\mathrm{sig}'(x) = \frac{\partial\,\mathrm{sig}(x)}{\partial x} = \mathrm{sig}(x) \cdot (1 - \mathrm{sig}(x))$$

T Compute the derivative of the sigmoidal function by hand!

- Shape of the sigmoidal and it's derivative (dashed line):



The Sigmoidal function and it's derivative

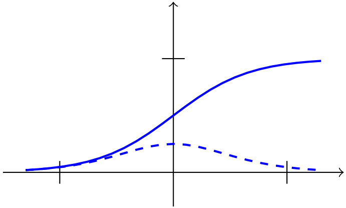
- First we need to compute the derivative of the sigmoidal

$$\mathrm{sig}(x) = \frac{1}{1.0 + e^{-x}}$$

$$\mathrm{sig}'(x) = \frac{\partial\, \mathrm{sig}(x)}{\partial x} = \mathrm{sig}(x) \cdot (1 - \mathrm{sig}(x))$$

T Compute the derivative of the sigmoidal function by hand!

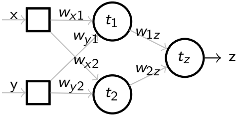
- Shape of the sigmoidal and it's derivative (dashed line):



- A very nice step-by-step explanation can e.g. be found here:

`towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e`

Partial derivative wrt. t_z



- Network function as computed above

$$\mathcal{N}_{xy} = \mathcal{N}(x, y) = \text{sig}\left(i_z(x, y) + t_z\right)$$

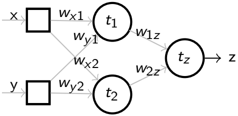
- An error (quadratic loss) function based on a given sample:

$$E(x, y, z) = \left(\mathcal{N}(x, y) - z\right)^2$$

- Compute the derivative of E wrt. t_z

$$\frac{\partial E(x, y, z)}{\partial t_z} = \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_z}$$

Partial derivative wrt. t_z



- Network function as computed above

$$\mathcal{N}_{xy} = \mathcal{N}(x, y) = \text{ sig}\left(i_z(x, y) + t_z\right)$$

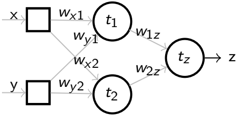
- An error (quadratic loss) function based on a given sample:

$$E(x, y, z) = \left(\mathcal{N}(x, y) - z\right)^2$$

- Compute the derivative of E wrt. t_z

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_z} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_z}\end{aligned}$$

Partial derivative wrt. t_z



- Network function as computed above

$$\mathcal{N}_{xy} = \mathcal{N}(x,y) = \operatorname{sig}\left(i_z(x,y) + t_z\right)$$

- An error (quadratic loss) function based on a given sample:

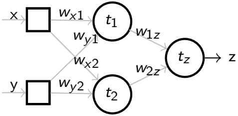
$$E(x,y,z) = (\mathcal{N}(x,y) - z)^2$$

- Compute the derivative of E wrt. t_z

$$\begin{aligned}\frac{\partial E(x,y,z)}{\partial t_z} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_z}\end{aligned}$$

$$\frac{\partial \mathcal{N}_{xy}}{\partial t_z} = \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial i_z(x,y) + t_z}{\partial t_z}$$

Partial derivative wrt. t_z



- Network function as computed above

$$\mathcal{N}_{xy} = \mathcal{N}(x, y) = \operatorname{sig}\left(i_z(x, y) + t_z\right)$$

- An error (quadratic loss) function based on a given sample:

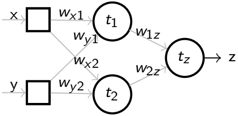
$$E(x, y, z) = (\mathcal{N}(x, y) - z)^2$$

- Compute the derivative of E wrt. t_z

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_z} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_z}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_z} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial i_z(x, y) + t_z}{\partial t_z} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy})\end{aligned}$$

Partial derivative wrt. t_z



- Network function as computed above

$$\mathcal{N}_{xy} = \mathcal{N}(x, y) = \text{ sig}\left(i_z(x, y) + t_z\right)$$

- An error (quadratic loss) function based on a given sample:

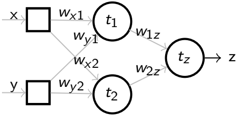
$$E(x, y, z) = \left(\mathcal{N}(x, y) - z\right)^2$$

- Compute the derivative of E wrt. t_z

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_z} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy})\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_z} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial i_z(x, y) + t_z}{\partial t_z} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy})\end{aligned}$$

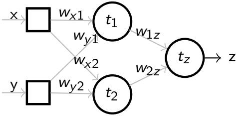
Partial derivative wrt. w_{1z}



$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x,y) &= \operatorname{sig}\left(i_z(x,y) + t_z\right) \\ &= \operatorname{sig}(w_{1z} * o_1(x,y) + w_{2z} * o_2(x,y) + t_z)\end{aligned}$$

$$\frac{\partial E(x,y,z)}{\partial w_{1z}} = \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial w_{1z}}$$

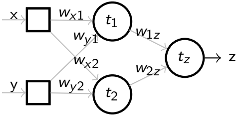
Partial derivative wrt. w_{1z}



$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x,y) &= \mathrm{sig}\left(i_z(x,y) + t_z\right) \\ &= \mathrm{sig}(w_{1z} * o_1(x,y) + w_{2z} * o_2(x,y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x,y,z)}{\partial w_{1z}} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial w_{1z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}}\end{aligned}$$

Partial derivative wrt. w_{1z}

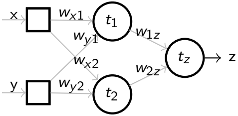


$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x, y) &= \text{sig}\left(i_z(x, y) + t_z\right) \\ &= \text{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial w_{1z}} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial w_{1z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}}\end{aligned}$$

$$\frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}} = \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z}{\partial w_{1z}}$$

Partial derivative wrt. w_{1z}

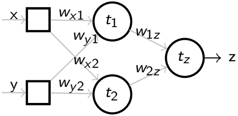


$$
\begin{aligned}
\mathcal{N}_{xy} = \mathcal{N}(x, y) &= \mathrm{sig}\left(i_z(x, y) + t_z\right) \\
&= \mathrm{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial E(x, y, z)}{\partial w_{1z}} &= \frac{\partial (\mathcal{N}_{xy} - z)^2}{\partial w_{1z}} \\
&= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z}{\partial w_{1z}} \\
&= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_1(x, y)
\end{aligned}
$$

Partial derivative wrt. w_{1z}

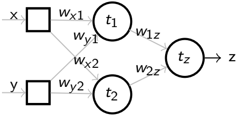


$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x, y) &= \text{sig}\left(i_z(x, y) + t_z\right) \\ &= \text{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial w_{1z}} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial w_{1z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_1(x, y)\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z}{\partial w_{1z}} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_1(x, y)\end{aligned}$$

Partial derivative wrt. w_{2z}

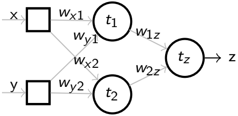


$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x, y) &= \mathrm{sig}\left(i_z(x, y) + t_z\right) \\ &= \mathrm{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial w_{2z}} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial w_{2z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{2z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_2(x, y)\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial w_{2z}} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z}{\partial w_{2z}} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_2(x, y)\end{aligned}$$

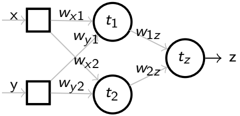
Partial derivative wrt. t_1



$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x,y) &= \mathrm{sig}\left(i_z(x,y) + t_z\right)\\ &= \mathrm{sig}(w_{1z} * \mathrm{sig}(i_1(x,y) + t_1) + w_{2z} * o_2(x,y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x,y,z)}{\partial t_1} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_1}\\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1}\end{aligned}$$

Partial derivative wrt. t_1

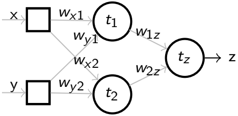


$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x, y) &= \text{sig}\left(i_z(x, y) + t_z\right) \\ &= \text{sig}(w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_1} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1}\end{aligned}$$

$$\frac{\partial \mathcal{N}_{xy}}{\partial t_1} = \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z}{\partial t_1}$$

Partial derivative wrt. t_1

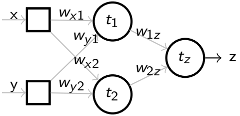


$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x,y) &= \mathrm{sig}\left(i_z(x,y) + t_z\right)\\ &= \mathrm{sig}(w_{1z} * \mathrm{sig}(i_1(x,y) + t_1) + w_{2z} * o_2(x,y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x,y,z)}{\partial t_1} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_1}\\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_1} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * \mathrm{sig}(i_1(x,y) + t_1) + w_{2z} * o_2(x,y) + t_z}{\partial t_1}\\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * \frac{\partial\, \mathrm{sig}(i_1(x,y) + t_1)}{\partial t_1}\end{aligned}$$

Partial derivative wrt. t_1

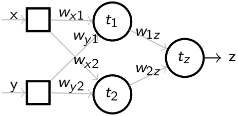


$$
\begin{aligned}
\mathcal{N}_{xy} = \mathcal{N}(x,y) &= \text{sig}\left(i_z(x,y) + t_z\right) \\
&= \text{sig}(w_{1z} * \text{sig}(i_1(x,y) + t_1) + w_{2z} * o_2(x,y) + t_z)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial E(x,y,z)}{\partial t_1} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_1} \\
&= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \mathcal{N}_{xy}}{\partial t_1} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * \text{sig}(i_1(x,y) + t_1) + w_{2z} * o_2(x,y) + t_z}{\partial t_1} \\
&= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * \frac{\partial\, \text{sig}(i_1(x,y) + t_1)}{\partial t_1} \\
&= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x,y) * (1 - o_1(x,y)) * \frac{\partial i_1(x,y) + t_1}{\partial t_1}
\end{aligned}
$$

Partial derivative wrt. t_1

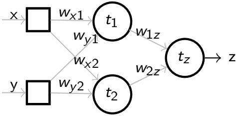


$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x,y) &= \mathrm{sig}\left(i_z(x,y) + t_z\right)\\ &= \mathrm{sig}(w_{1z} * \mathrm{sig}(i_1(x,y) + t_1) + w_{2z} * o_2(x,y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x,y,z)}{\partial t_1} &= \frac{\partial (\mathcal{N}_{xy} - z)^2}{\partial t_1}\\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_1} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * \mathrm{sig}(i_1(x,y) + t_1) + w_{2z} * o_2(x,y) + t_z}{\partial t_1}\\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * \frac{\partial\, \mathrm{sig}(i_1(x,y) + t_1)}{\partial t_1}\\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x,y) * (1 - o_1(x,y)) * \frac{\partial i_1(x,y) + t_1}{\partial t_1}\\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x,y) * (1 - o_1(x,y)) * 1\end{aligned}$$

Partial derivative wrt. t_1



$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x,y) &= \mathrm{sig}\left(i_z(x,y) + t_z\right) \\ &= \mathrm{sig}(w_{1z} * \mathrm{sig}(i_1(x,y) + t_1) + w_{2z} * o_2(x,y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x,y,z)}{\partial t_1} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x,y) * (1 - o_1(x,y))\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_1} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * \mathrm{sig}(i_1(x,y) + t_1) + w_{2z} * o_2(x,y) + t_z}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * \frac{\partial\, \mathrm{sig}(i_1(x,y) + t_1)}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x,y) * (1 - o_1(x,y)) * \frac{\partial i_1(x,y) + t_1}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x,y) * (1 - o_1(x,y)) * 1\end{aligned}$$

Take-away-messages of section: *"Training Artificial Neural Networks"*

E You should now be able to ...

- explain the idea behind gradient descent
- explain how gradient descent can be used to adapt the weights of a neural network
- derive the equations to adapt the weights and thresholds of a simple network by hand