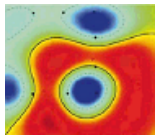
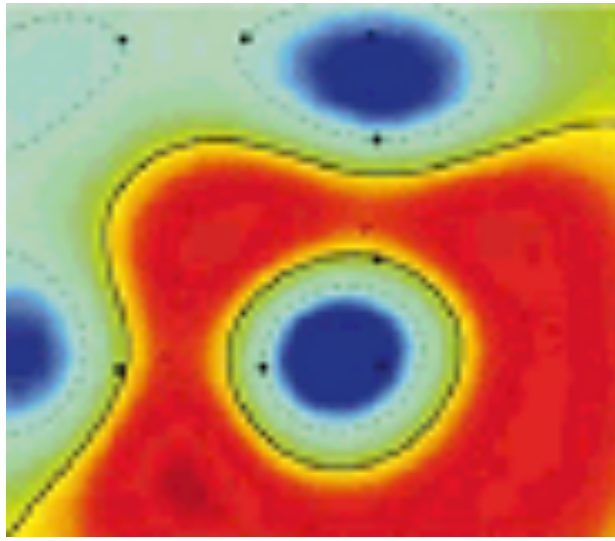


WiSe 2024/25

**Deep Learning 1**

Technische Universität Berlin



Lecture 2

# Error Backpropagation

# Outline

- The Perceptron algorithm
- Perceptron as gradient descent
- How to compute the gradient in a neural network
- Numerical gradient computation
- The error backpropagation algorithm
  - Chain rule and multivariate chain rule
  - Worked through example
  - General equation for backpropagation
  - Vectorization of backpropagation
  - Automatic differentiation

Part 1

# The Perceptron

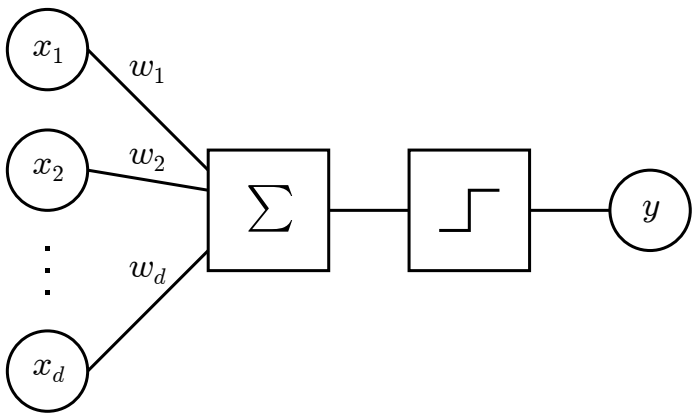
# The Perceptron


F. Rosenblatt (1928–1971)

- Algorithm proposed in 1958 by F. Rosenblatt to train single-layer neural networks.
- The algorithm produces classifiers that prefectly separates training data (if the data is linearly separable).
- The algorithm consists of simple and cheap iterative procedure.

# The Perceptron

**Structure of the perceptron:**




- A weighted sum of the input features:

$$z = \sum_{i=1}^{d} w_i x_i + b = \boldsymbol{w}^\top \boldsymbol{x} + b$$

followed by the sign function

$$y = \text{sign}(z)$$

**Problem formulation:** Let

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \in \mathbb{R}^d$$

be our input data and $t^{(1)}, t^{(2)}, \ldots, t^{(N)} \in \{-1, 1\}$ our corresponding labels (aka. targets). The goal of the perceptron is to learn a collection of parameters $(\boldsymbol{w}, b)$ such that all points are correctly classified, i.e.

$$\forall_{k=1}^{N} : \quad y^{(k)} = t^{(k)}$$

# The Perceptron Algorithm

Recall that for each data point, predictions of our perceptron are computed as:

$$z^{(k)} = \boldsymbol{w}^\top \boldsymbol{x}^{(k)} + b$$
$$y^{(k)} = \text{sign}(z^{(k)})$$

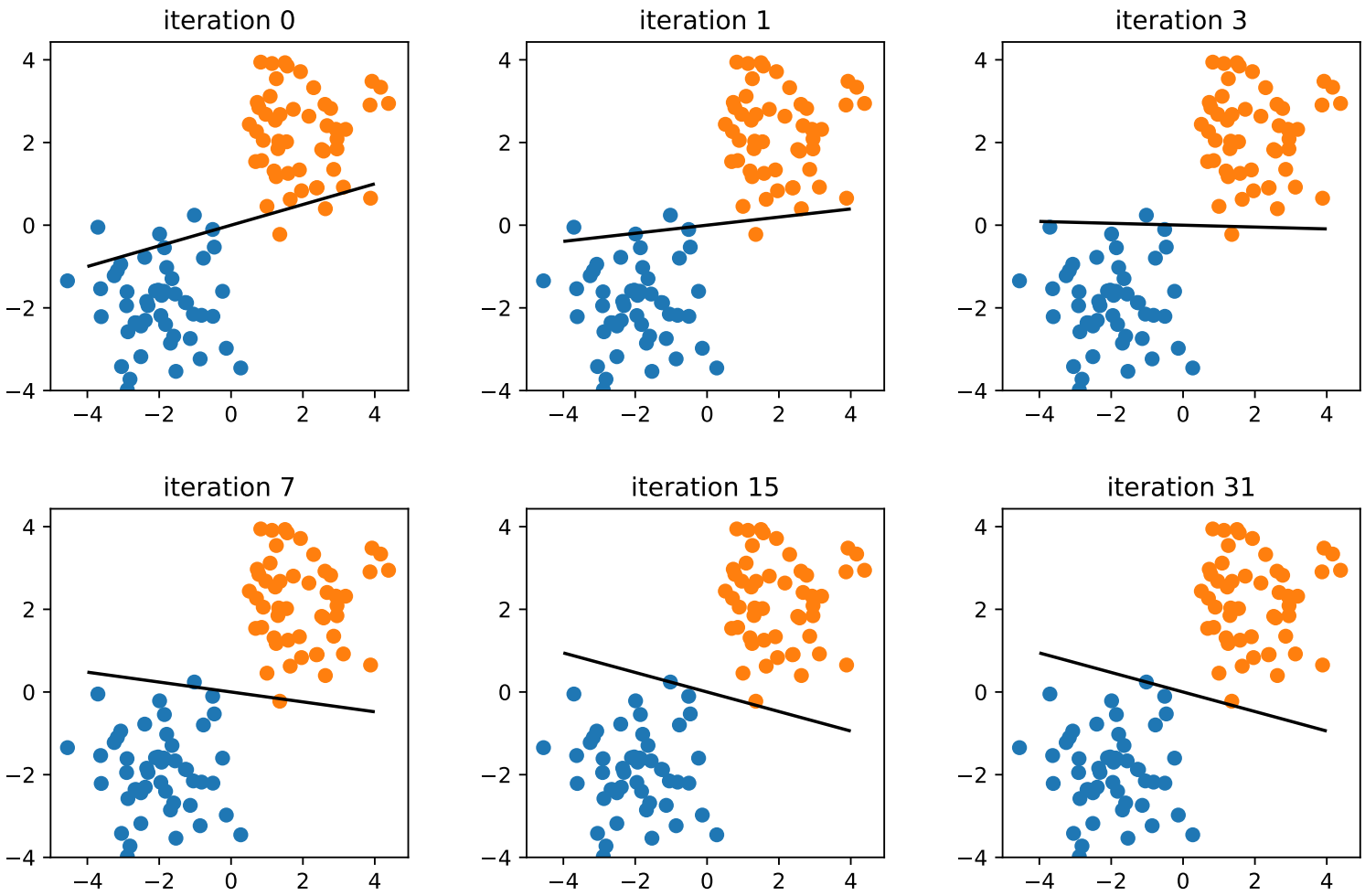
**Perceptron algorithm**

- Iterate (multiple times from $k = 1 \dots N$).
  - If $\boldsymbol{x}^{(k)}$ is correctly classified ($y^{(k)} = t^{(k)}$), continue.
  - If $\boldsymbol{x}^{(k)}$ is wrongly classified ($y^{(k)} \neq t^{(k)}$), update the perceptron:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \cdot \boldsymbol{x}^{(k)} t^{(k)}$$
$$b \leftarrow b + \eta \cdot t^{(k)}$$

    where $\eta$ is a learning rate.
- Stop once all examples are correctly classified.

# Perceptron at Work



# The Perceptron: Optimization View

**Proposition**

*The perceptron can be seen as a gradient descent of the error function*

$$\mathcal{E}(\boldsymbol{w}, b) = \frac{1}{N} \sum_{k=1}^{N} \underbrace{\max(0, -z^{(k)} t^{(k)})}_{\mathcal{E}_k(\boldsymbol{w}, b)}$$

*Also known as the Hinge Loss.*

**Proof.**

$$\begin{aligned}
\boldsymbol{w} - \eta \frac{\partial \mathcal{E}_k}{\partial \boldsymbol{w}} &= \boldsymbol{w} - \eta \cdot 1_{-z^{(k)} t^{(k)} > 0} \cdot \Big( - \frac{\partial z^{(k)}}{\partial \boldsymbol{w}} t^{(k)} \Big) \\
&= \boldsymbol{w} - \eta \cdot 1_{y^{(k)} \neq t^{(k)}} \cdot \Big( - \frac{\partial z^{(k)}}{\partial \boldsymbol{w}} t^{(k)} \Big) \\
&= \boldsymbol{w} + \eta \cdot 1_{y^{(k)} \neq t^{(k)}} \cdot \boldsymbol{x}^{(k)} t^{(k)}
\end{aligned}$$

which is the parameter update equation of the perceptron algorithm. We proceed similarly for the parameter $b$. □

# Perceptron vs. Nearest Mean Classifier

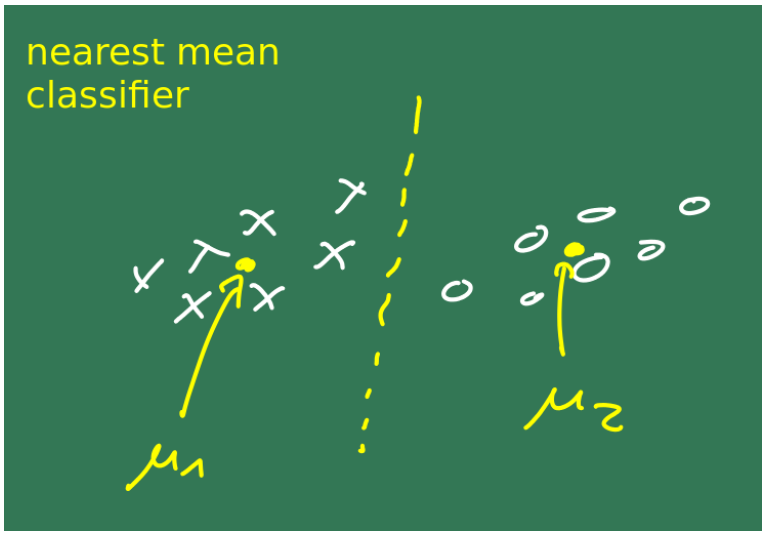
The nearest mean classifier first builds the means of two classes:

$$\boldsymbol{\mu}_1 = \frac{1}{|\mathcal{C}_1|} \sum_{k \in \mathcal{C}_1} \boldsymbol{x}^{(k)} \quad \text{and} \quad \boldsymbol{\mu}_2 = \frac{1}{|\mathcal{C}_2|} \sum_{k \in \mathcal{C}_2} \boldsymbol{x}^{(k)}$$

Then it predicts:

class 1 if $\|\boldsymbol{x} - \boldsymbol{\mu}_1\| < \|\boldsymbol{x} - \boldsymbol{\mu}_2\|$

class 2 if $\|\boldsymbol{x} - \boldsymbol{\mu}_1\| > \|\boldsymbol{x} - \boldsymbol{\mu}_2\|$



# Perceptron vs. Nearest Mean Classifier

Equivalent formulation of the nearest mean classifier:

$$\begin{aligned} &\text{class 1} \quad \text{if} \|\boldsymbol{x} - \boldsymbol{\mu}_1\|^2 < \|\boldsymbol{x} - \boldsymbol{\mu}_2\|^2 \\ &\text{class 2} \quad \text{if} \|\boldsymbol{x} - \boldsymbol{\mu}_1\|^2 > \|\boldsymbol{x} - \boldsymbol{\mu}_2\|^2 \end{aligned}$$

(raising distances to the square does not change the decision).

The nearest mean classifier can then be further developed to become expressible as a linear classifier:

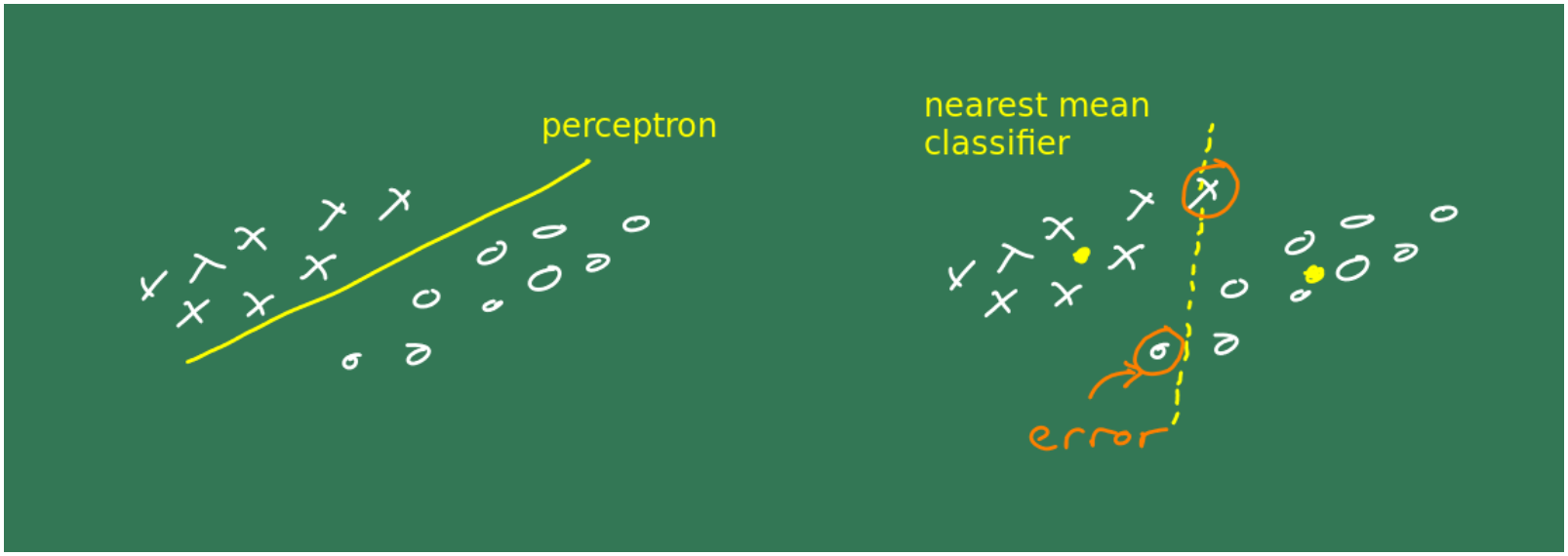
$$\begin{aligned} y &= \text{sign}(\|\boldsymbol{x} - \boldsymbol{\mu}_2\|^2 - \|\boldsymbol{x} - \boldsymbol{\mu}_1\|^2) \\ &= \text{sign}(\|\boldsymbol{x}\|^2 - 2\boldsymbol{\mu}_2^\top \boldsymbol{x} + \|\boldsymbol{\mu}_2\|^2 - \|\boldsymbol{x}\|^2 + 2\boldsymbol{\mu}_1^\top \boldsymbol{x} - \|\boldsymbol{\mu}_1\|^2) \\ &= \text{sign}(\underbrace{2(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)}_{\boldsymbol{w}}^\top \boldsymbol{x} + \underbrace{\|\boldsymbol{\mu}_2\|^2 - \|\boldsymbol{\mu}_1\|^2}_{b}) \end{aligned}$$

where $y = 1$ corresponds to class 1 and $y = -1$ corresponds to class 2.

# Perceptron vs. Nearest Mean Classifier

**Question:**

- Both the perceptron and the nearest mean classifier are linear classifiers. Then, how do the perceptron and nearest mean classifier differ?




**Observation**

- Perceptron always separates data when it is linearly separable. This is not always the case for the nearest mean classifier, especially when the data of each class is elongated.

Nearest mean classifier describes the data, perceptron minimizes the error.

Part 2

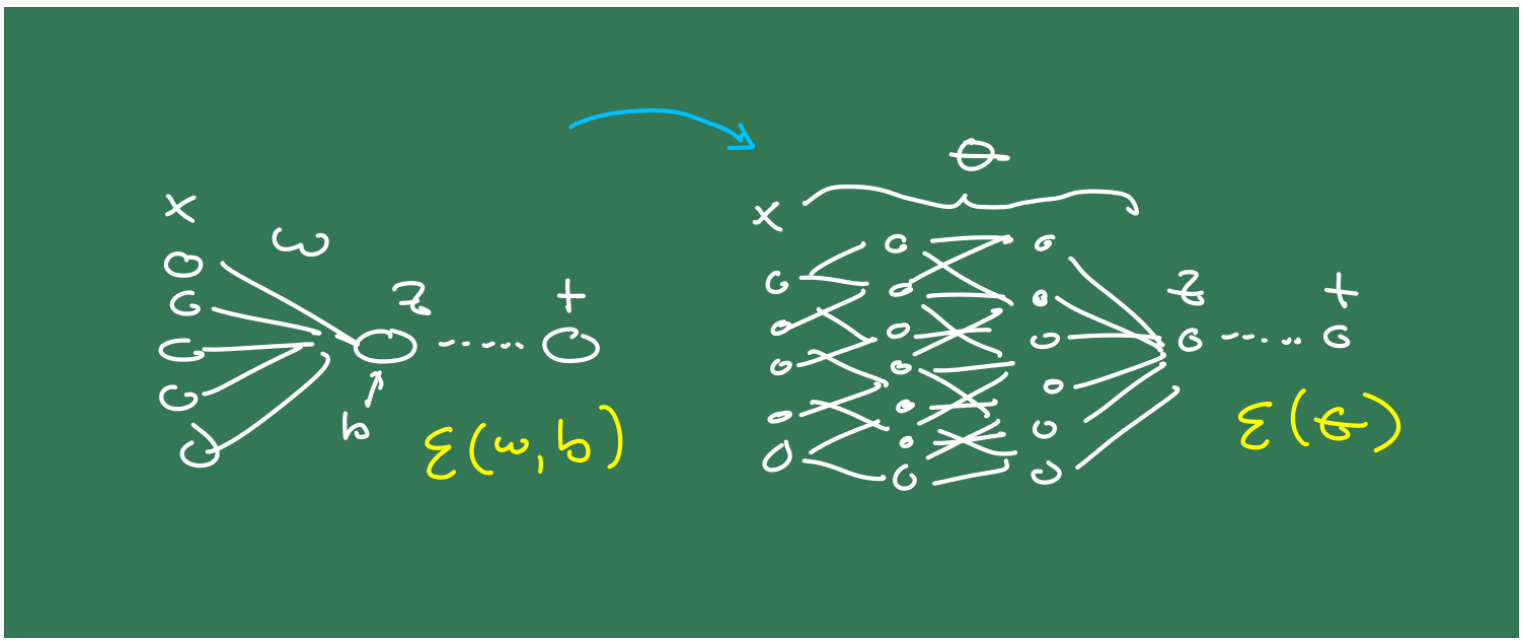
# Training Multilayer Networks

# From Perceptrons to Training Deep Networks

**Recap:** The perceptron performs a gradient descent of the error function

$$\mathcal{E}(\boldsymbol{w}, b) = \frac{1}{N} \sum_{k=1}^{N} \underbrace{\max(0, -z^{(k)} t^{(k)})}_{\mathcal{E}_k(\boldsymbol{w}, b)}$$

**Idea:** Generalize the formulation where $z$ is not the output of the perceptron, but of any multilayer neural network.




**Question:** How to compute $\partial\mathcal{E}/\partial\theta$, i.e. the gradient of the newly defined error function w.r.t. the model parameters?

# Naive Approach: Numerical Differentiation

Formula for numerical differentiation:

$$\forall_t : \frac{\partial \mathcal{E}}{\partial \theta_t} = \lim_{\epsilon \to 0} \frac{\mathcal{E}(\theta + \epsilon \cdot \delta_t) - \mathcal{E}(\theta)}{\epsilon}$$

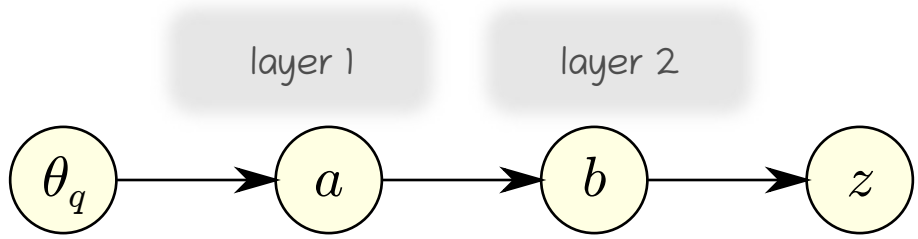
where $\delta_t$ is an indicator vector for the parameter $t$.

**Properties:**

- Can be applied to any error function $\mathcal{E}$ (not necessary the error of a neural network).
- Need to evaluate the function as many times as there are parameters ($\rightarrow$ slow when the number of parameters is large).
- A neural network typically has between $10^3$ and $10^9$ parameters ($\rightarrow$ numerical differentiation unfeasible).
- Still useful as a unit-test for verifying gradient computation.
- Because $\epsilon$ and the numerator are very small, for numerical differentiation to work, one must use high-precision (e.g. float64 rather than float32).

# Better Approach: The Chain Rule

Suppose that some parameter of interest $\theta_q$ (one element of the parameter vector $\theta$) is linked to the output of the network through some sequence of functions.



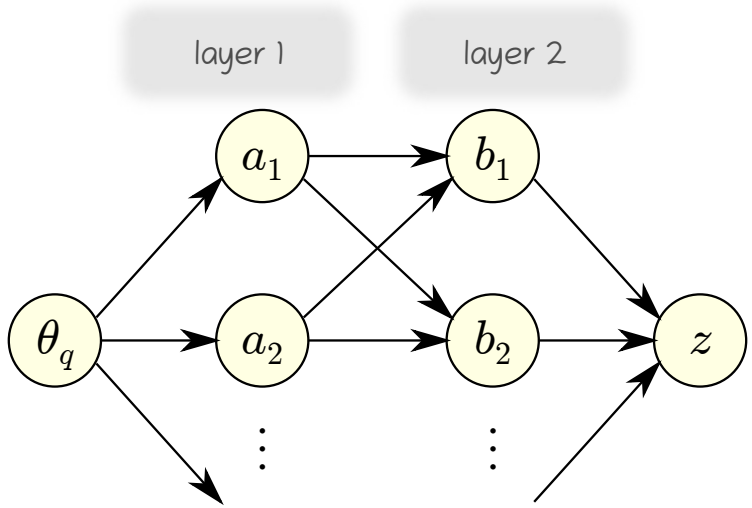

The chain rule for derivatives states that

$$\frac{\partial z}{\partial \theta_q} = \frac{\partial z}{\partial b} \frac{\partial b}{\partial a} \frac{\partial a}{\partial \theta_q}$$

i.e. the derivative w.r.t. the parameter of interest is the product of local derivatives along the path connecting $\theta_q$ to $z$.

# The Multivariate Chain Rule

In practice, some parameter of interest may be linked to the output of the network through multiple paths (formed by all neurons between them):

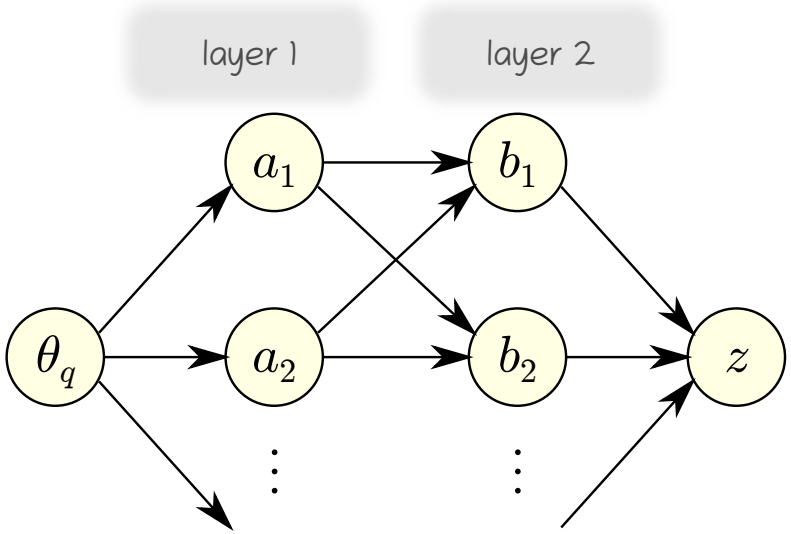



The chain rule can be extended to this multivariate scenario by enumerating all the paths between $\theta_q$ and $z$

$$\frac{\partial z}{\partial \theta_q} = \sum_i \sum_j \frac{\partial z}{\partial b_j} \frac{\partial b_j}{\partial a_i} \frac{\partial a_i}{\partial \theta_q}$$

where $\sum_i$ and $\sum_j$ run over the indices of the neurons in the corresponding layers. Its complexity grows exponentially with the number of layers.

# Factor Structure in the Multivariate Chain Rule



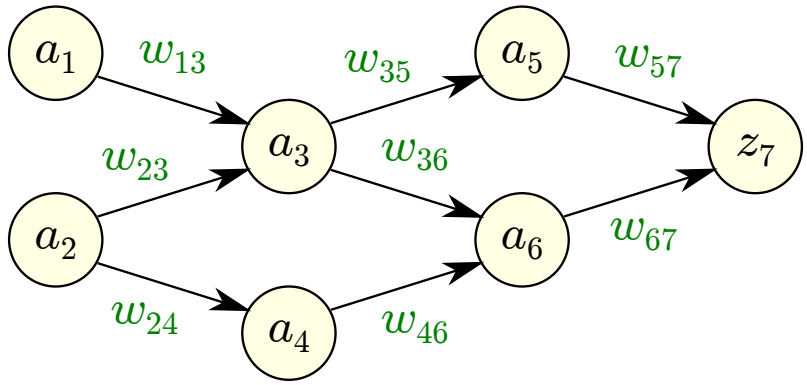

$$\frac{\partial z}{\partial \theta_q} = \sum_i \frac{\partial a_i}{\partial \theta_q} \overbrace{\sum_j \frac{\partial b_j}{\partial a_i} \underbrace{\frac{\partial z}{\partial b_j}}_{\delta_j}}^{\delta_i}$$

- Computation can be rewritten in a way that summing operation can be performed incrementally.
- Intermediate computation can be reused for different paths, and for different parameters for which we would like to compute the gradient.
- Overall, the resulting gradient computation (w.r.t. of all parameters in the network) becomes linear with the size of the network ($\Rightarrow$ fast!)
- The algorithm is known as the Error Backpropagation algorithm (Rumelhart 1986).

Part 3

# Worked-Through Example and Formalization

# Worked through Example

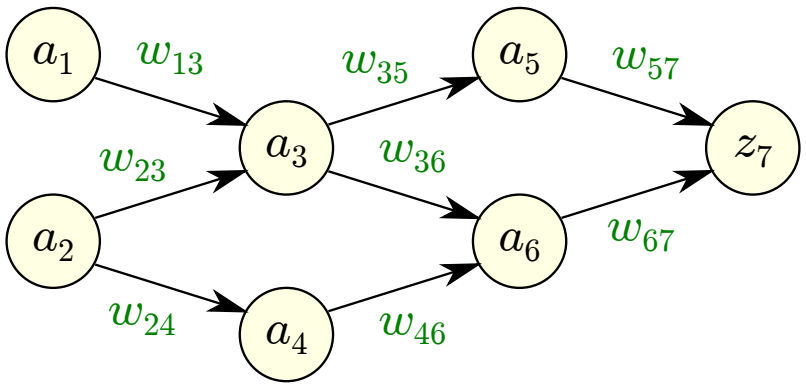



**Forward pass:**

$$
\begin{aligned}
z_3 &= a_1 w_{13} + a_2 w_{23} \\
z_4 &= a_2 w_{24} \\
z_5 &= a_3 w_{35} \\
z_6 &= a_3 w_{36} + a_4 w_{46} \\
z_7 &= a_5 w_{57} + a_6 w_{67} \\
\mathcal{E} &= \max(0, -z_7 t)
\end{aligned}
$$

$$
\begin{aligned}
a_1 &= x_1 \\
a_2 &= x_2 \\
a_3 &= g(z_3) \\
a_4 &= g(z_4) \\
a_5 &= g(z_5) \\
a_6 &= g(z_6)
\end{aligned}
$$

# Worked through Example

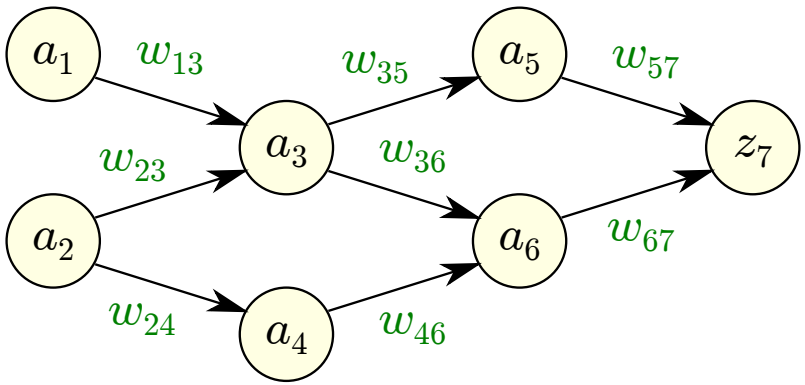



$$
\begin{aligned}
z_3 &= a_1 w_{13} + a_2 w_{23} & a_1 &= x_1 \\
z_4 &= a_2 w_{24} & a_2 &= x_2 \\
z_5 &= a_3 w_{35} & a_3 &= g(z_3) \\
z_6 &= a_3 w_{36} + a_4 w_{46} & a_4 &= g(z_4) \\
z_7 &= a_5 w_{57} + a_6 w_{67} & a_5 &= g(z_5) \\
\mathcal{E} &= \max(0, -z_7 t) & a_6 &= g(z_6)
\end{aligned}
$$

**Backward pass:**

$$\delta_7 = \frac{\partial \mathcal{E}}{\partial z_7} = I(-z_7 \cdot t > 0) \cdot (-t)$$

# Worked through Example

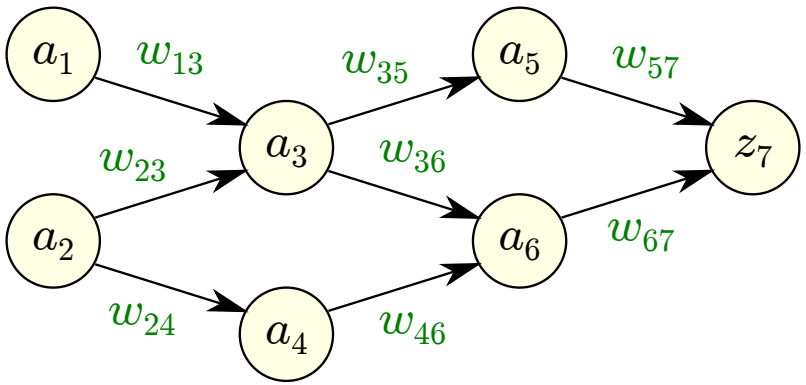



$$
\begin{aligned}
a_1 &= x_1 \\
a_2 &= x_2 \\
z_3 &= a_1 w_{13} + a_2 w_{23} & a_3 &= g(z_3) \\
z_4 &= a_2 w_{24} & a_4 &= g(z_4) \\
z_5 &= a_3 w_{35} & a_5 &= g(z_5) \\
z_6 &= a_3 w_{36} + a_4 w_{46} & a_6 &= g(z_6) \\
z_7 &= a_5 w_{57} + a_6 w_{67} \\
\mathcal{E} &= \max(0, -z_7 t)
\end{aligned}
$$

**Backward pass:**

$$
\begin{aligned}
\delta_7 = \frac{\partial \mathcal{E}}{\partial z_7} &= I(-z_7 \cdot t > 0) \cdot (-t) \\
\frac{\partial \mathcal{E}}{\partial w_{67}} &= \frac{\partial \mathcal{E}}{\partial z_7} \frac{\partial z_7}{\partial w_{67}} = \delta_7 \cdot a_6 \\
\frac{\partial \mathcal{E}}{\partial w_{57}} &= \frac{\partial \mathcal{E}}{\partial z_7} \frac{\partial z_7}{\partial w_{57}} = \delta_7 \cdot a_5
\end{aligned}
$$

# Worked through Example

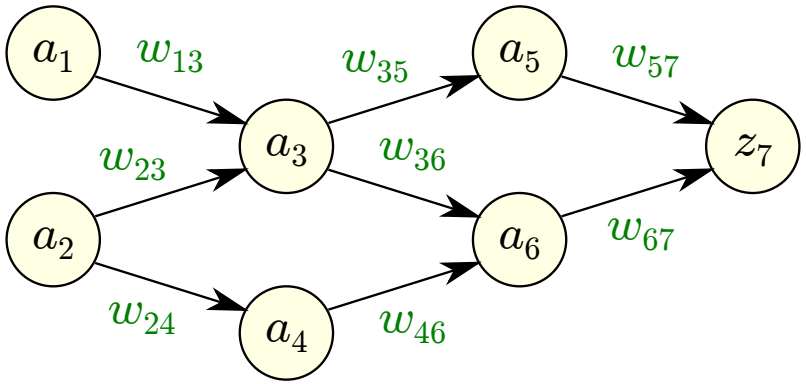



$$
\begin{aligned}
z_3 &= a_1 w_{13} + a_2 w_{23} & a_3 &= g(z_3) \\
z_4 &= a_2 w_{24} & a_4 &= g(z_4) \\
z_5 &= a_3 w_{35} & a_5 &= g(z_5) \\
z_6 &= a_3 w_{36} + a_4 w_{46} & a_6 &= g(z_6) \\
z_7 &= a_5 w_{57} + a_6 w_{67} & & \\
\mathcal{E} &= \max(0, -z_7 t) & &
\end{aligned}
$$

**Backward pass:**

$$
\begin{aligned}
\delta_7 &= \frac{\partial \mathcal{E}}{\partial z_7} = I(-z_7 \cdot t > 0) \cdot (-t) \\
\delta_6 &= \frac{\partial \mathcal{E}}{\partial a_6} = \frac{\partial \mathcal{E}}{\partial z_7} \frac{\partial z_7}{\partial a_6} = \delta_7 \cdot w_{67} \\
\delta_5 &= \frac{\partial \mathcal{E}}{\partial a_5} = \frac{\partial \mathcal{E}}{\partial z_7} \frac{\partial z_7}{\partial a_5} = \delta_7 \cdot w_{57}
\end{aligned}
$$

# Worked through Example

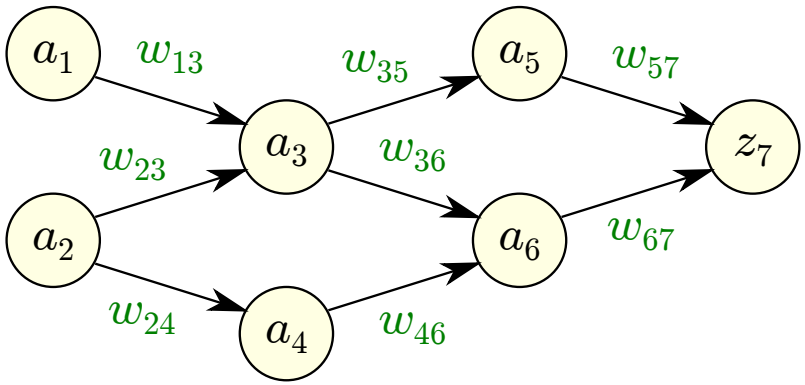



$$
\begin{aligned}
z_3 &= a_1 w_{13} + a_2 w_{23} & a_1 &= x_1 \\
z_4 &= a_2 w_{24} & a_2 &= x_2 \\
z_5 &= a_3 w_{35} & a_3 &= g(z_3) \\
z_6 &= a_3 w_{36} + a_4 w_{46} & a_4 &= g(z_4) \\
z_7 &= a_5 w_{57} + a_6 w_{67} & a_5 &= g(z_5) \\
\mathcal{E} &= \max(0, -z_7 t) & a_6 &= g(z_6)
\end{aligned}
$$

**Backward pass:**

$$
\begin{aligned}
\delta_6 = \frac{\partial \mathcal{E}}{\partial a_6} &= \frac{\partial \mathcal{E}}{\partial z_7}\frac{\partial z_7}{\partial a_6} = \delta_7 \cdot w_{67} \\
\delta_5 = \frac{\partial \mathcal{E}}{\partial a_5} &= \frac{\partial \mathcal{E}}{\partial z_7}\frac{\partial z_7}{\partial a_5} = \delta_7 \cdot w_{57} \\
\frac{\partial \mathcal{E}}{\partial w_{46}} &= \frac{\partial \mathcal{E}}{\partial a_6}\frac{\partial a_6}{\partial z_6}\frac{\partial z_6}{\partial w_{46}} = \delta_6 \cdot g'(z_6) \cdot a_4 \\
\frac{\partial \mathcal{E}}{\partial w_{36}} &= \frac{\partial \mathcal{E}}{\partial a_6}\frac{\partial a_6}{\partial z_6}\frac{\partial z_6}{\partial w_{36}} = \delta_6 \cdot g'(z_6) \cdot a_3 \\
\frac{\partial \mathcal{E}}{\partial w_{35}} &= \frac{\partial z_5}{\partial w_{35}}\frac{\partial a_5}{\partial z_5}\frac{\partial \mathcal{E}}{\partial a_5} = \delta_5 \cdot g'(z_5) \cdot a_3
\end{aligned}
$$

# Worked through Example




$$
\begin{aligned}
z_3 &= a_1 w_{13} + a_2 w_{23} & a_1 &= x_1 \\
z_4 &= a_2 w_{24} & a_2 &= x_2 \\
z_5 &= a_3 w_{35} & a_3 &= g(z_3) \\
z_6 &= a_3 w_{36} + a_4 w_{46} & a_4 &= g(z_4) \\
z_7 &= a_5 w_{57} + a_6 w_{67} & a_5 &= g(z_5) \\
\mathcal{E} &= \max(0, -z_7 t) & a_6 &= g(z_6)
\end{aligned}
$$

**Backward pass:**

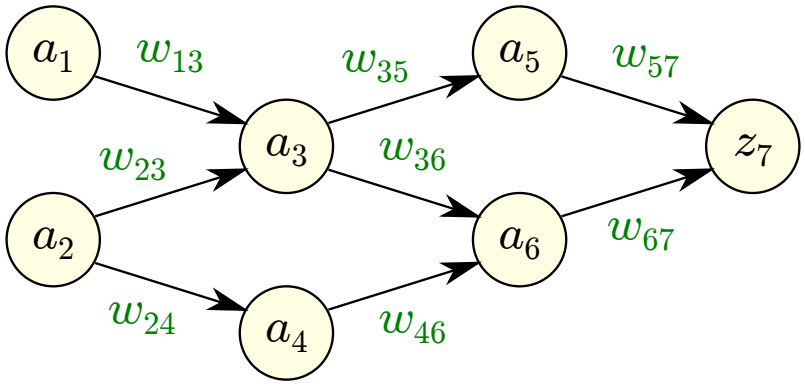
$$\delta_6 = \frac{\partial \mathcal{E}}{\partial a_6} = \frac{\partial \mathcal{E}}{\partial z_7}\frac{\partial z_7}{\partial a_6} = \delta_7 \cdot w_{67}$$

$$\delta_5 = \frac{\partial \mathcal{E}}{\partial a_5} = \frac{\partial \mathcal{E}}{\partial z_7}\frac{\partial z_7}{\partial a_5} = \delta_7 \cdot w_{57}$$

$$\delta_4 = \frac{\partial \mathcal{E}}{\partial a_4} = \frac{\partial \mathcal{E}}{\partial a_6}\frac{\partial a_6}{\partial z_6}\frac{\partial z_6}{\partial a_4} = \delta_6 \cdot g'(z_6) \cdot w_{46}$$

$$\delta_3 = \frac{\partial \mathcal{E}}{\partial a_3} = \frac{\partial \mathcal{E}}{\partial a_6}\frac{\partial a_6}{\partial z_6}\frac{\partial z_6}{\partial a_3} + \frac{\partial \mathcal{E}}{\partial a_5}\frac{\partial a_5}{\partial z_5}\frac{\partial z_5}{\partial a_3} = \delta_6 \cdot g'(z_6) \cdot w_{36} + \delta_5 \cdot g'(z_5) \cdot w_{35}$$

# Worked through Example




$$
\begin{aligned}
z_3 &= a_1 w_{13} + a_2 w_{23} & a_1 &= x_1 \\
z_4 &= a_2 w_{24} & a_2 &= x_2 \\
z_5 &= a_3 w_{35} & a_3 &= g(z_3) \\
z_6 &= a_3 w_{36} + a_4 w_{46} & a_4 &= g(z_4) \\
z_7 &= a_5 w_{57} + a_6 w_{67} & a_5 &= g(z_5) \\
\mathcal{E} &= \max(0, -z_7 t) & a_6 &= g(z_6)
\end{aligned}
$$

**Backward pass:**

$$\delta_4 = \frac{\partial \mathcal{E}}{\partial a_4} = \frac{\partial \mathcal{E}}{\partial a_6}\frac{\partial a_6}{\partial z_6}\frac{\partial z_6}{\partial a_4} = \delta_6 \cdot g'(z_6) \cdot w_{46}$$

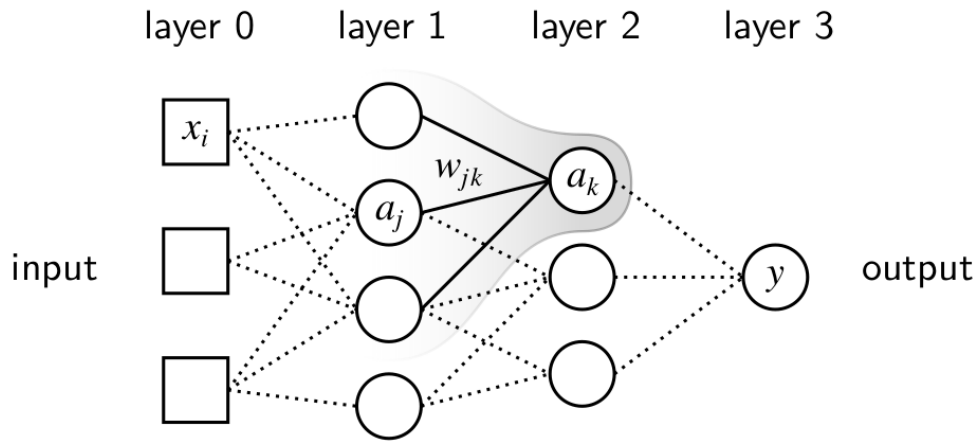
$$\delta_3 = \frac{\partial \mathcal{E}}{\partial a_3} = \frac{\partial \mathcal{E}}{\partial a_6}\frac{\partial a_6}{\partial z_6}\frac{\partial z_6}{\partial a_3} + \frac{\partial \mathcal{E}}{\partial a_5}\frac{\partial a_5}{\partial z_5}\frac{\partial z_5}{\partial a_3} = \delta_6 \cdot g'(z_6) \cdot w_{36} + \delta_5 \cdot g'(z_5) \cdot w_{35}$$

$$\frac{\partial \mathcal{E}}{\partial w_{24}} = \frac{\partial \mathcal{E}}{\partial a_4}\frac{\partial a_4}{\partial z_4}\frac{\partial z_4}{\partial w_{24}} = \delta_4 \cdot g'(z_4) \cdot a_2$$

$$\frac{\partial \mathcal{E}}{\partial w_{23}} = \frac{\partial \mathcal{E}}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial w_{23}} = \delta_3 \cdot g'(z_3) \cdot a_2$$

$$\frac{\partial \mathcal{E}}{\partial w_{13}} = \frac{\partial \mathcal{E}}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial w_{13}} = \delta_3 \cdot g'(z_3) \cdot a_1$$

# Formalization for a Standard Neural Network




The error gradient can be propagated from layer to layer using the chain rule:

$$\underbrace{\frac{\partial \mathcal{E}}{\partial a_j}}_{\delta_j} = \sum_k \underbrace{\frac{\partial \mathcal{E}}{\partial a_k}}_{\delta_k} \cdot \underbrace{\frac{\partial a_k}{\partial z_k}}_{g'(z_k)} \cdot \underbrace{\frac{\partial z_k}{\partial a_j}}_{w_{jk}}$$

And gradients w.r.t. parameters at each layer can be extracted as:

$$\frac{\partial \mathcal{E}}{\partial w_{jk}} = \underbrace{\frac{\partial \mathcal{E}}{\partial a_k}}_{\delta_k} \cdot \underbrace{\frac{\partial a_k}{\partial z_k}}_{g'(z_k)} \cdot \underbrace{\frac{\partial z_k}{\partial w_{jk}}}_{a_j}$$

# Matrix Formulation

**Observation:**

- Backpropagation equations can be written as matrix-vector products, or outer products:

| Neuron-wise | Layer-wise |
|---|---|
| $\delta_j = \sum_k \delta_k g'(z_k) w_{jk}$ | $\boldsymbol{\delta}^{(l-1)} = W^{(l-1,l)} \cdot (g'(\boldsymbol{z}^{(l)}) \odot \boldsymbol{\delta}^{(l)})$ |
| $\frac{\partial \mathcal{E}}{\partial w_{jk}} = \delta_k g'(z_k) a_j$ | $\frac{\partial \mathcal{E}}{\partial W^{(l-1,l)}} = \boldsymbol{a}^{(l-1)} \cdot (g'(\boldsymbol{z}^{(l)}) \odot \boldsymbol{\delta}^{(l)})^\top$ |

where:

- $j$ *and* $k$ *are indices for neurons at layer* $l-1$ *and* $l$ *respectively,*
- $\odot$ *is an element-wise multiplication,*
- $g'(\cdot)$ *is the derivative of* $g$ *applied element-wise,*
- $W^{(l-1,l)}$ *is a matrix of size* $(d_{l-1} \times d_l)$*, where* $d_{l-1}$ *and* $d_l$ *indicate the number of neurons at layers* $l-1$ *and* $l$ *respectively.*

**Note:**

- Further vectorization can be achieved by computing the gradient for multiple data points at once, in which case, we have matrix-matrix products.

Part 4

# Further Remarks / Advanced Topics

# Choice of Nonlinear Activation Function

In practice, for training to proceed, the nonlinear function must be chosen in a way that:

1. Its gradient is defined (almost) everywhere.
2. There is a significant portion of the input domain where the gradient is non-zero.
3. Gradient must be informative i.e. indicate decrease/increase of the activation function.

**Common activation functions:**

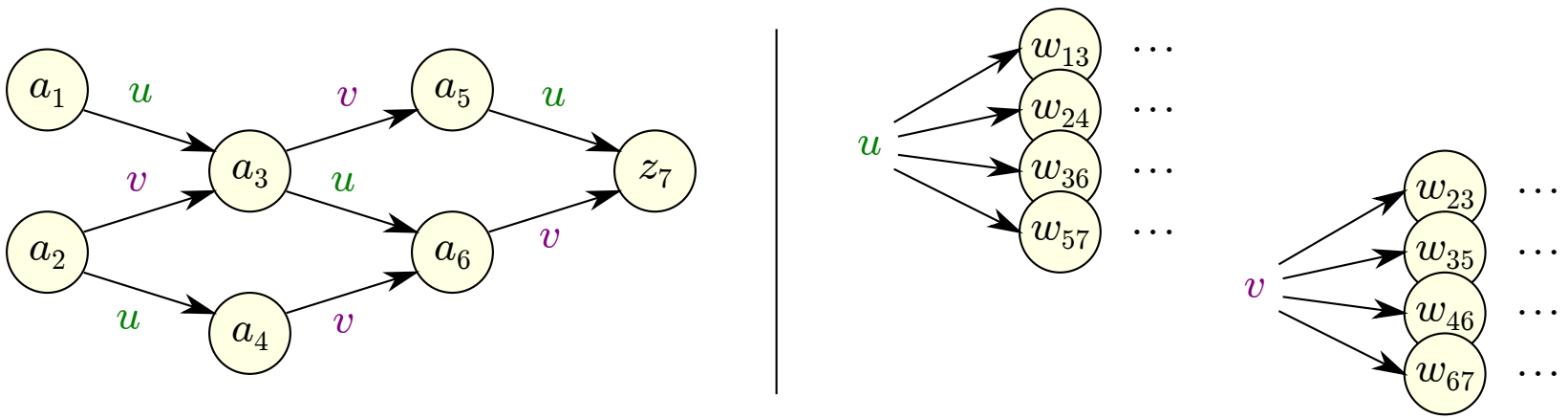
- $g(z) = \exp(z)/(1 + \exp(z))$
- $g(z) = \tanh(z)$
- $g(z) = \max(0, z)$

**Problematic activation functions:**

- $g(z) = \max(0, z - 100)$
- $g(z) = 1_{z>0}$
- $g(z) = \sin(100 \cdot z)$

# Neural Networks with Shared Parameters

Shared parameters can be handled by treating original parameters as "neurons" generated from the new parameters, and appling the chain rule one step further:




Chain rule equations:

$$\frac{\partial z_7}{\partial u} = \frac{\partial z_7}{\partial w_{13}} \underbrace{\frac{\partial w_{13}}{\partial u}}_{1} + \frac{\partial z_7}{\partial w_{24}} \underbrace{\frac{\partial w_{24}}{\partial u}}_{1} + \frac{\partial z_7}{\partial w_{36}} \underbrace{\frac{\partial w_{36}}{\partial u}}_{1} + \frac{\partial z_7}{\partial w_{57}} \underbrace{\frac{\partial w_{57}}{\partial u}}_{1}$$

$$\frac{\partial z_7}{\partial v} = \frac{\partial z_7}{\partial w_{23}} \underbrace{\frac{\partial w_{23}}{\partial v}}_{1} + \frac{\partial z_7}{\partial w_{35}} \underbrace{\frac{\partial w_{35}}{\partial v}}_{1} + \frac{\partial z_7}{\partial w_{46}} \underbrace{\frac{\partial w_{46}}{\partial v}}_{1} + \frac{\partial z_7}{\partial w_{67}} \underbrace{\frac{\partial w_{67}}{\partial v}}_{1}$$

# Automatic Differentiation

**Automatic Differentiation:**

- Generates automatically backpropagation equations from the forward equations.
- Automatic differentiation became widely available in neural network libraries (PyTorch, Tensorflow, JAX, etc.)

**Consequences:**

- In practice we do not need to do backpropagation anymore. We just need to program the forward pass, and the backward pass comes for free.
- This has enabled researchers to develop neural networks that are way more complex, and with much more heterogeneous structures (e.g. ResNet, Yolo, transformers, etc.).
- Only in few cases, it is still useful to express the gradient analytically (e.g. to analyze theoretically the stability of a gradient descent procedure, such as the vanishing/exploding gradients problem in recurrent neural networks).

# Simple algorithm for training a neural network

**Basic gradient descent algorithm:**

Initialize vector of parameters $\theta$ at random.
**for** $t = 1 \ldots T$ **do**
  Compute for all data points the forward pass
  Compute the error function $\mathcal{E}(\theta)$
  Extract the gradient $\nabla\mathcal{E}(\theta)$ using backpropagation
  Perform a gradient step, i.e.

$$\theta \leftarrow \theta - \gamma \cdot \nabla\mathcal{E}(\theta)$$

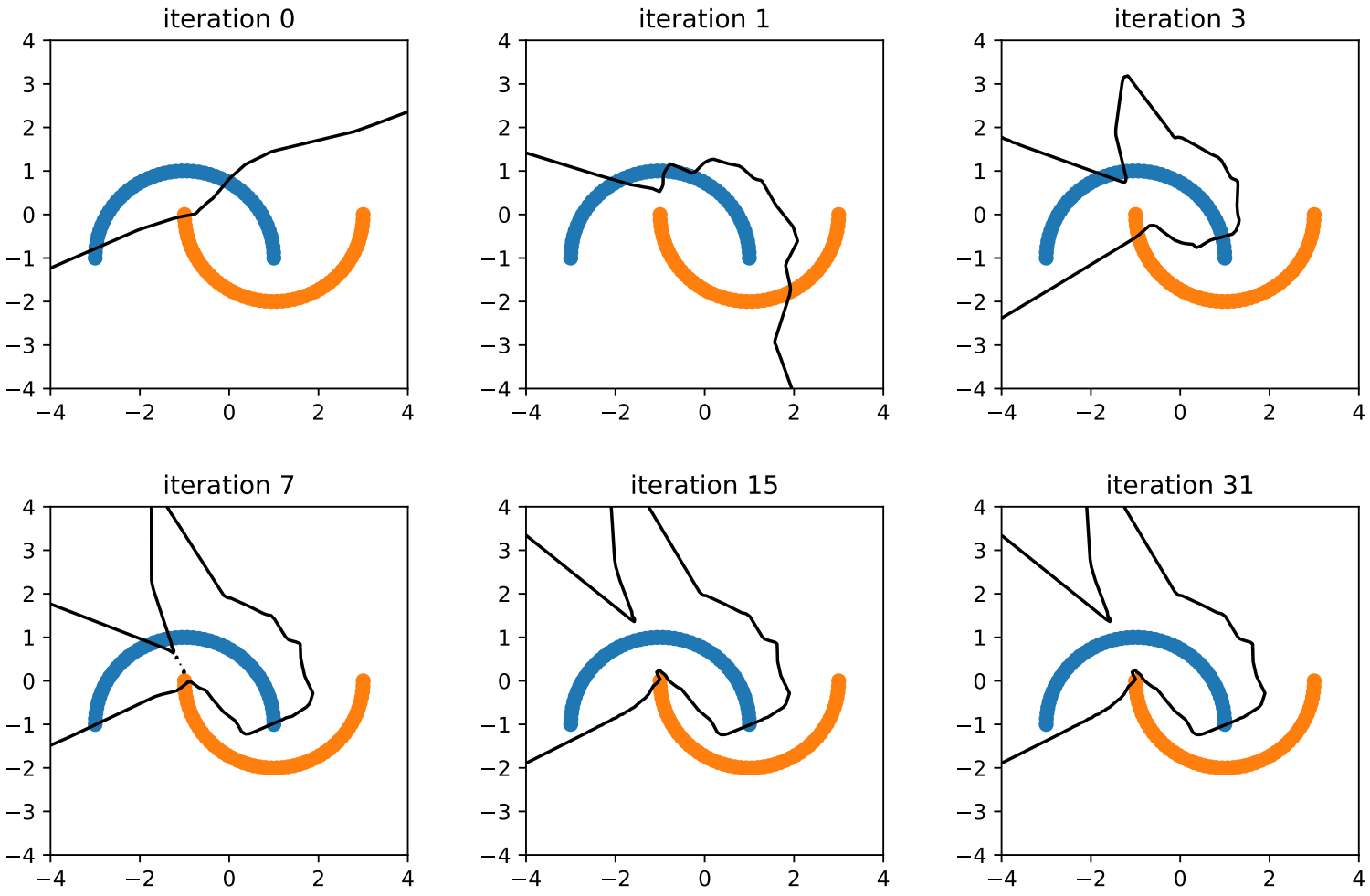
**end for**

- The parameter $\gamma$ is a learning rate that needs to be set by the user.

Part 5

# Neural Network at Work

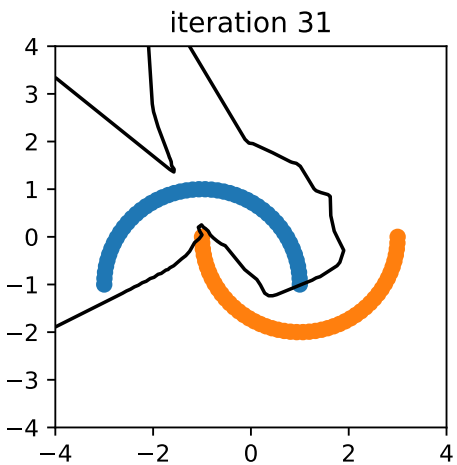
# Neural Network at Work




**Observation:**

- After enough iterations, all points are on the correct side of the decision boundary (like for the perceptron, but even if the data is not linearly separable).

# Neural Network at Work




... still not very fast, and not an optimal decision boundary.

**Consideration for next lectures:**

*Optimization (Lectures 3–4)*

- How to make training faster? (especially important if we consider large problems with many input variables.)

*Regularization (Lectures 5–6)*

- Decision function doesn't look nice. Unlikely to work well for new data points. Can we introduce a mechanism in the learning proceedure that promotes more regular and well-generalizing decision functions?

# Summary

# Summary

- Error of a classifier can be minimized using gradient descent (e.g. Perceptron, neural network + backpropagation).
- Error backpropagation is computationally efficient way of computing the gradient (much faster than using the limit formulation of the derivative).
- Error backpropagation is an application of the multivariate chain rule, where the different terms can be factored due to the structure of the neural network graph.
- In practice, we most of the time do not need to program error backpropagation manually, and we can instead use automatic differentation techniques available in most modern neural network libraries.
- Error backpropagation only extracts the gradient. This is not enough for training a neural network successfully, one still needs to make sure the gradient descent is carried out efficiently (lectures 3–4) and that the learned model is robust to generalize well to new data points (lectures 5–6).