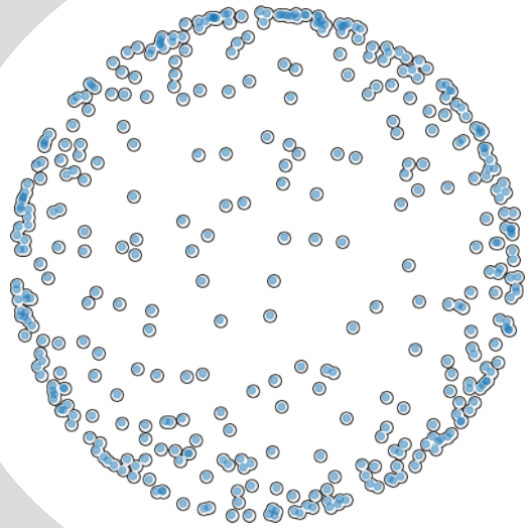
WiSe 2024/25

Machine Learning for Data Science

Lecture by G. Montavon

FREIE UNIVERSITÄT BERLIN



Lecture 11a | **Neural Networks**

Outline

Motivations

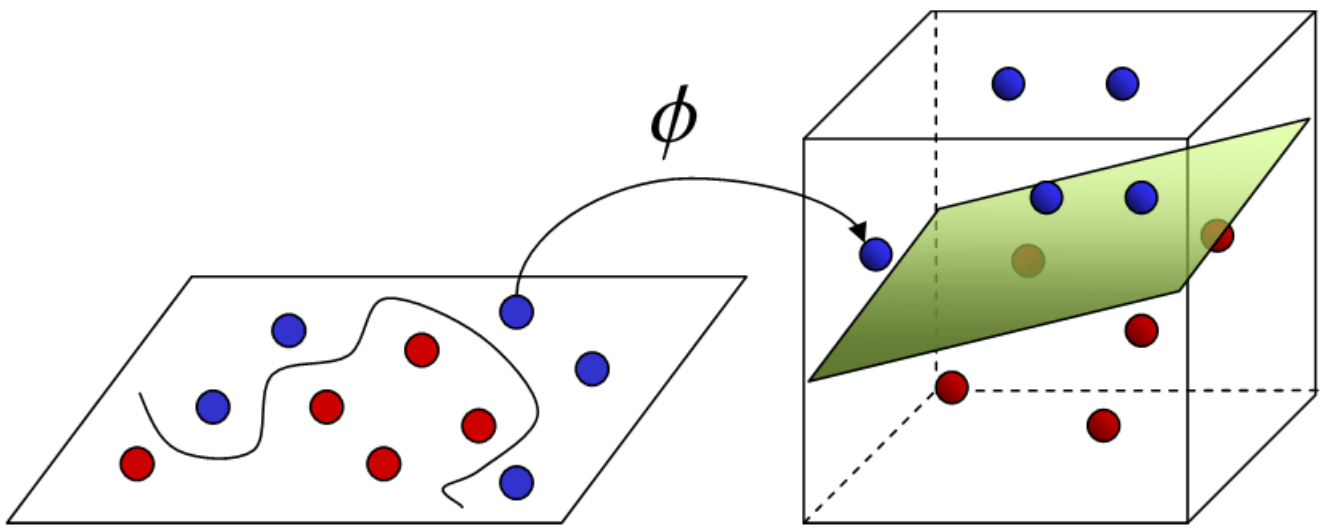
- From linear models/kernels to neural networks
- From biological to artificial neural networks
- Compactness

Neural Networks

- Training via gradient descent
- The forward pass
- The backward pass

ML Review: From Linear to Nonlinear Models

Most problems are however not linearly separable, and we need a way to enable ML models to learn nonlinear decision boundaries. A simple approach consists of nonlinearly mapping $\boldsymbol{x}$ to some high-dimensional feature space $\phi(\boldsymbol{x})$, and classify linearly in that space. The decision boundary becomes nonlinear in input space.



Question: How to choose the feature map ϕ?

Feature Expansions / Kernels

Idea: Generate new features automatically (without expert knowledge) using an expansion procedure.

- *Polynomial expansion:* Expand the input representation with product of features:

$$\begin{aligned}\phi(\boldsymbol{x}) &= \left(x_1, x_2, \ldots, x_d, x_1^2, x_2^2, \ldots, x_d^2, x_1x_2, x_1x_3, x_2x_3, \ldots\right) \\ &= \left(\boldsymbol{x}, \boldsymbol{x}\boldsymbol{x}^\top, \ldots\right)\end{aligned}$$

- *Random features expansion:* Expand the input representation with random projections $\boldsymbol{w}$ and nonlinear functions:

$$\phi(\boldsymbol{x}) = \left(\rho(\boldsymbol{w}_1^\top \boldsymbol{x}), \rho(\boldsymbol{w}_2^\top \boldsymbol{x}), \ldots, \rho(\boldsymbol{w}_H^\top \boldsymbol{x})\right)$$

  with $\boldsymbol{w}_j \in \mathcal{N}(0, \sigma I)$.

The problem is more likely to be linearly separable in higher dimensions than in the original d dimensions, however, computing a sufficiently large expansion might be expensive.

Part 1

Neural Networks

Motivations

Empirical Observation:

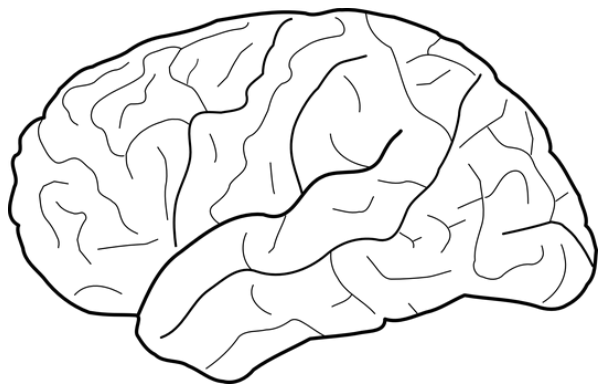
- Humans have shown capable of mastering tasks such as visual recognition, motion, speech, games, etc. All these tasks are highly nonlinear (i.e. they somehow need some nonlinear feature representation $\phi(\boldsymbol{x})$).

Question:

- Can machine learning models take inspiration of some mechanisms in the human's brain in order to learn the needed feature representation $\phi(\boldsymbol{x})$?

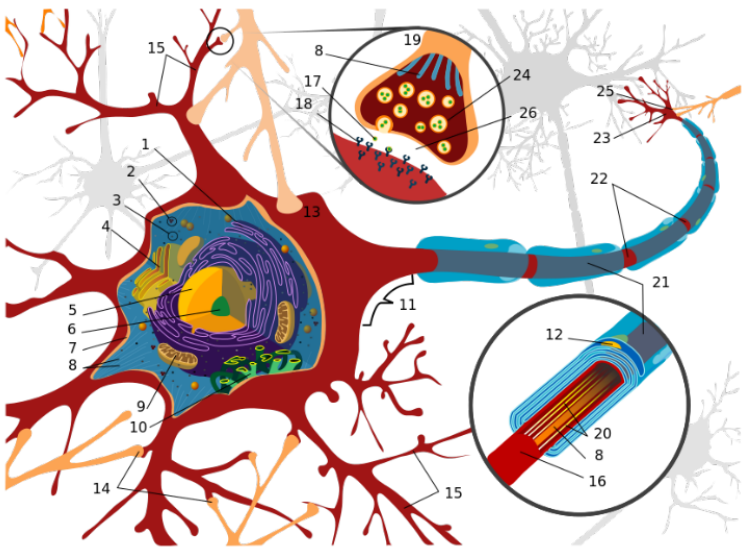
The Human Brain as a Model for Machine Learning

- The human brain is a highly complex (and so far scarcely understood) system.
- Scientific research in the past century has however provided some understanding of what might enable these systems to learn successfully:
  - Complex abstract representations result from the *interconnection* of many simple nonlinear neurons.
  - The property of these neurons to *modify* their response when exposed repeatedly to a certain stimuli enables the brain to learn.

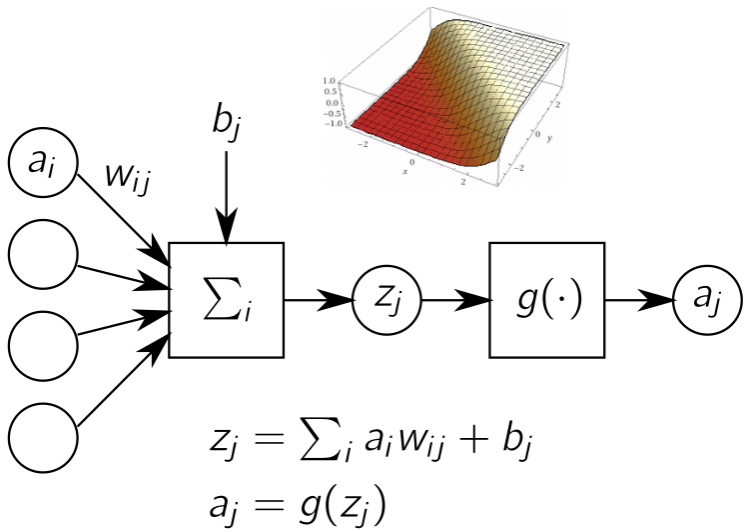


Biological vs. Artificial Neurons

Biological neuron

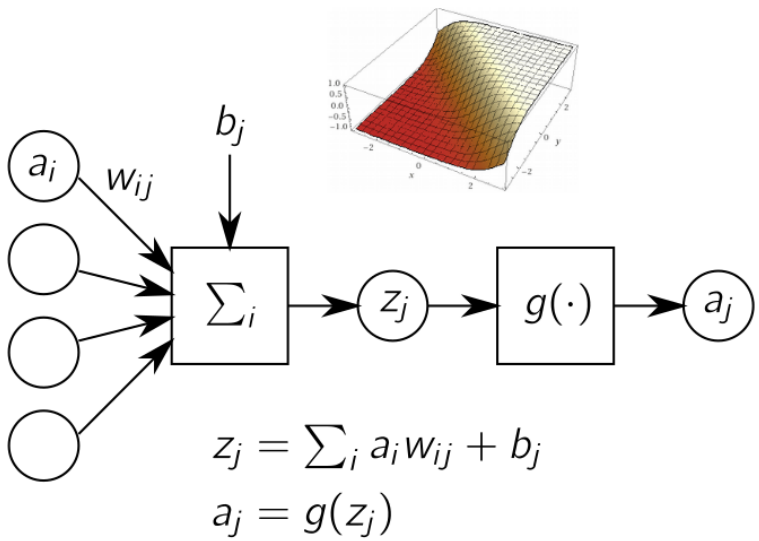


Artificial neuron



- The biological neuron is a highly sophisticated physical system with complex spatio-temporal dynamics that transfers signal received by dendrites to the axon.
- Artificial neurons only retain the most essential components of the biological neuron for practical purposes: *nonlinearity* and ability to *learn*.

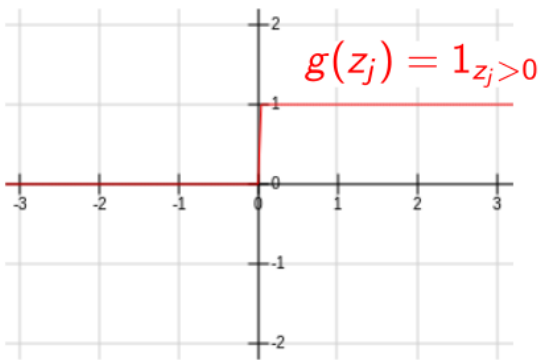
The Artificial Neuron



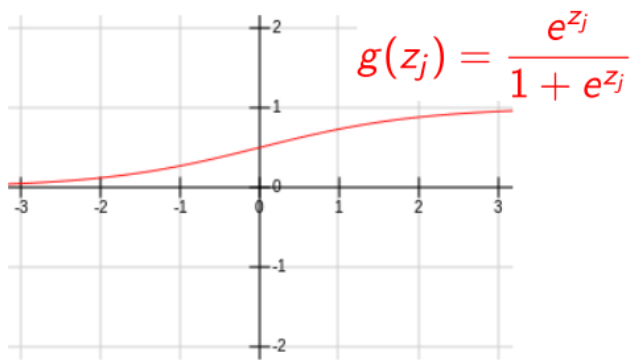
- Simple multivariate, nonlinear and differentiable function.
- Ultra-simplification of the biological neuron that retains two key properties: (1) ability to produce complex nonlinear representations when many neurons are interconnected (2) ability to learn from the data.

Examples of Nonlinear Functions

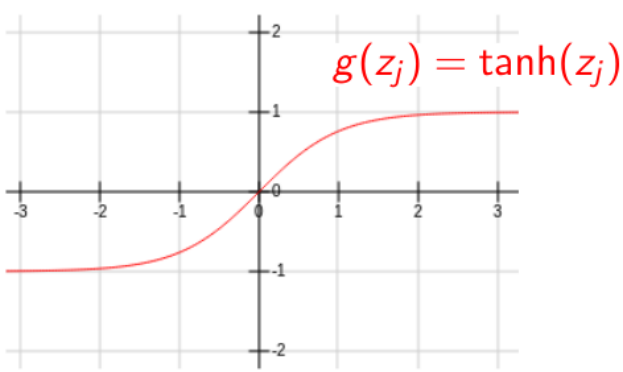
threshold function



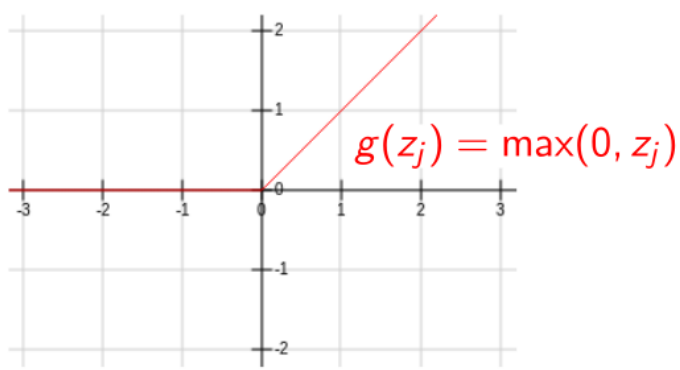
logistic sigmoid



hyperbolic tangent

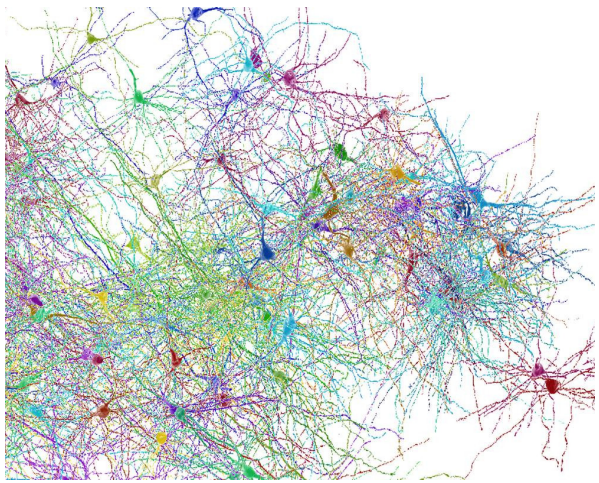


rectified linear unit

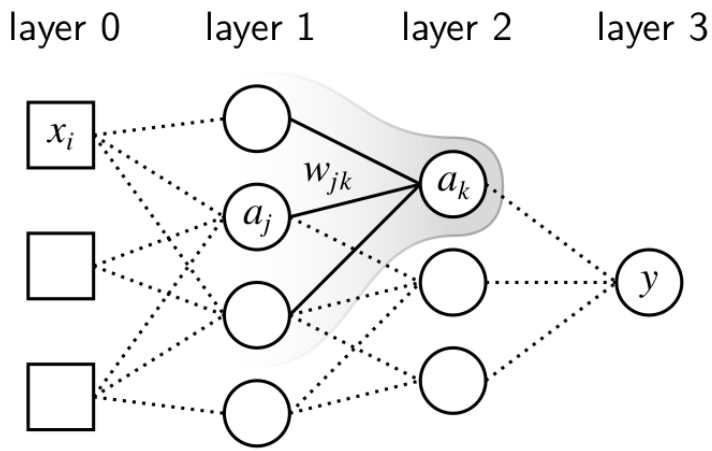


Interconnecting Multiple Neurons

Biological network

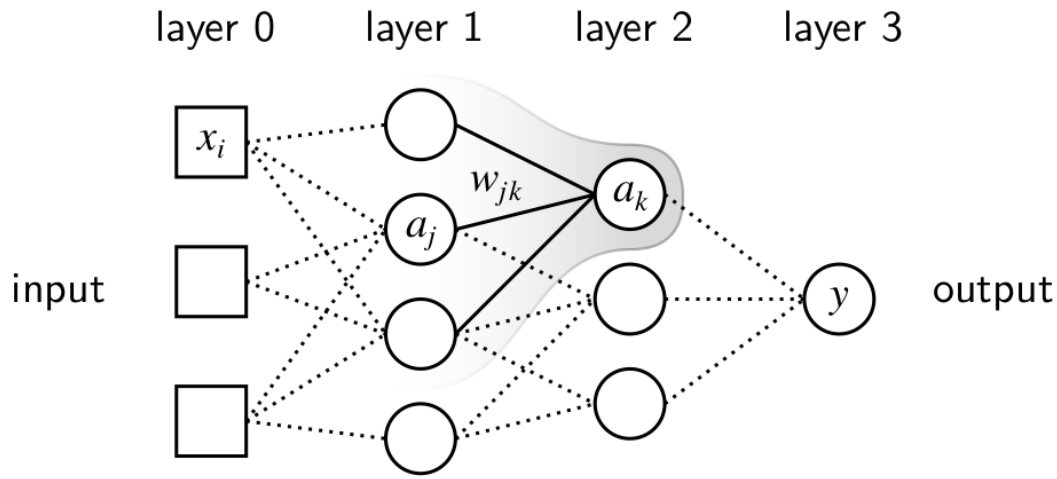


Artificial network



- The human brain is composed of a very large number of neurons (approx. 86 billions) that are interconnected (150 trillions synapses).
- An artificial neural network mimicks the way biological neurons are connected in the brain by composing many artificial neurons. For practical purposes, neurons of an artificial neural network can be organized in a layered structure.

Neural Networks: Forward Pass



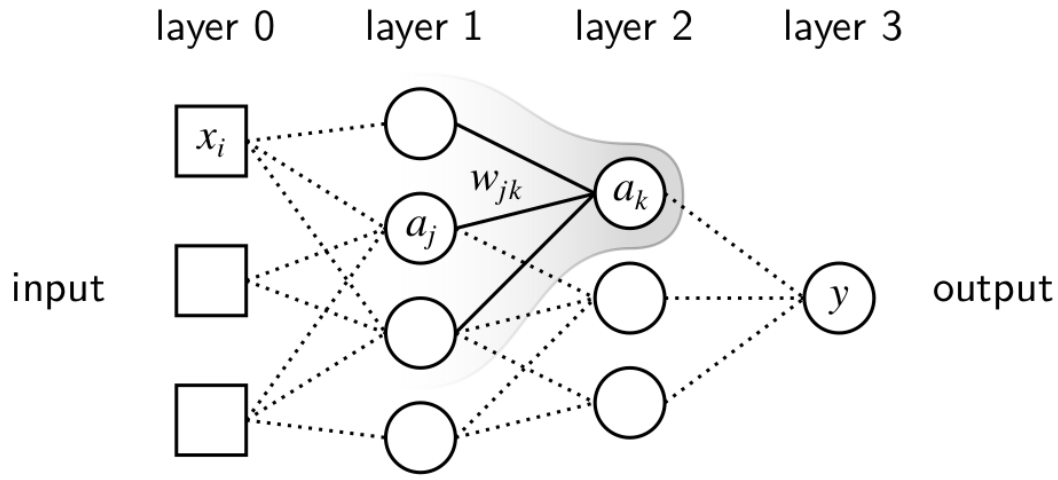
The forward pass mapping the input of the network to the output is given by:

$$z_j = \sum_i x_i w_{ij} + b_j \qquad a_j = g(z_j) \qquad \text{(layer 1)}$$

$$z_k = \sum_j a_j w_{jk} + b_k \qquad a_k = g(z_k) \qquad \text{(layer 2)}$$

$$y = \sum_k a_k v_k + c \qquad \text{(layer 3)}$$

Neural Networks: Forward Pass (Matrix Formulation)



Matrix formulation:

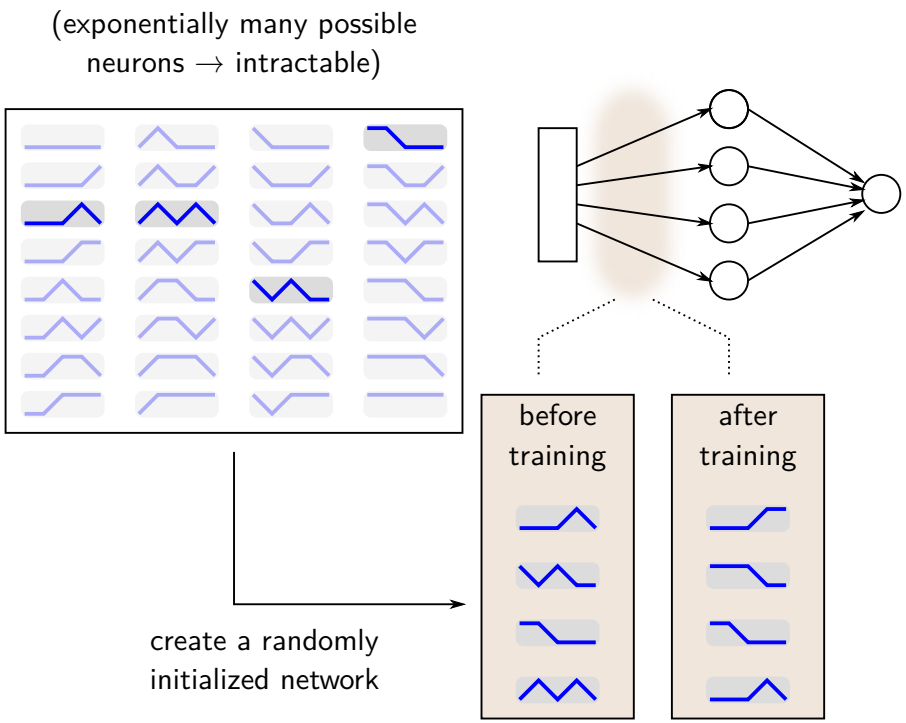
$$z^{(1)} = W^{(1)}x + b^{(1)} \qquad a^{(1)} = g(z^{(1)}) \qquad \text{(layer 1)}$$

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)} \qquad a^{(2)} = g(z^{(2)}) \qquad \text{(layer 2)}$$

$$y = v^\top a^{(2)} + c \qquad \text{(layer 3)}$$

where $[W^{(1)}]_{ji} = w_{ij}$, $[W^{(2)}]_{kj} = w_{jk}$, and where g applies element-wise. The matrix formulation makes it convenient to train neural networks with hundreds, thousands, or more neurons.

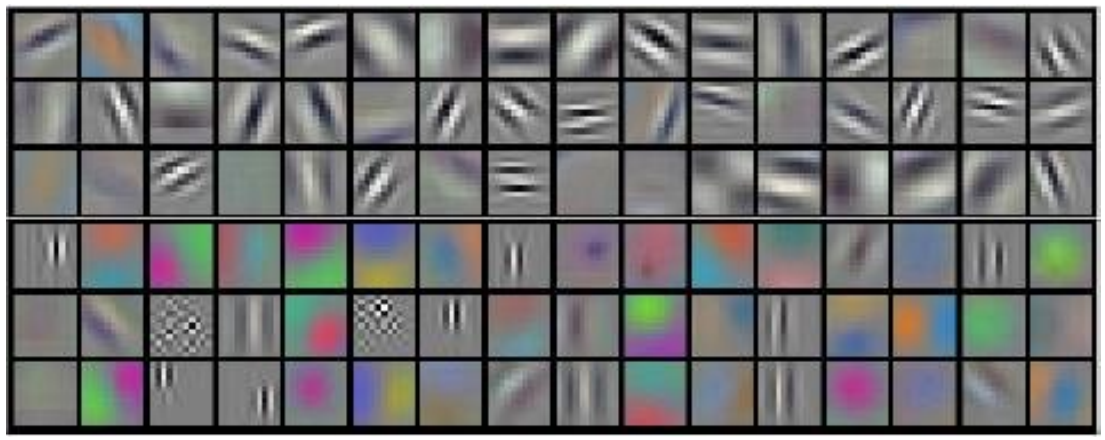
Neural Networks build Compact Representations



- The neural network starts with a finite and typically small set of randomly initialized neurons (i.e. a subset of all possible neurons).
- The compact problem representation is progressively extracted during training under the simultaneous effect of optimization (minimizing the error) and the finite number of neurons in the model.
- The learned representation is almost as predictive as an exhaustive set of neurons, but much more compact.

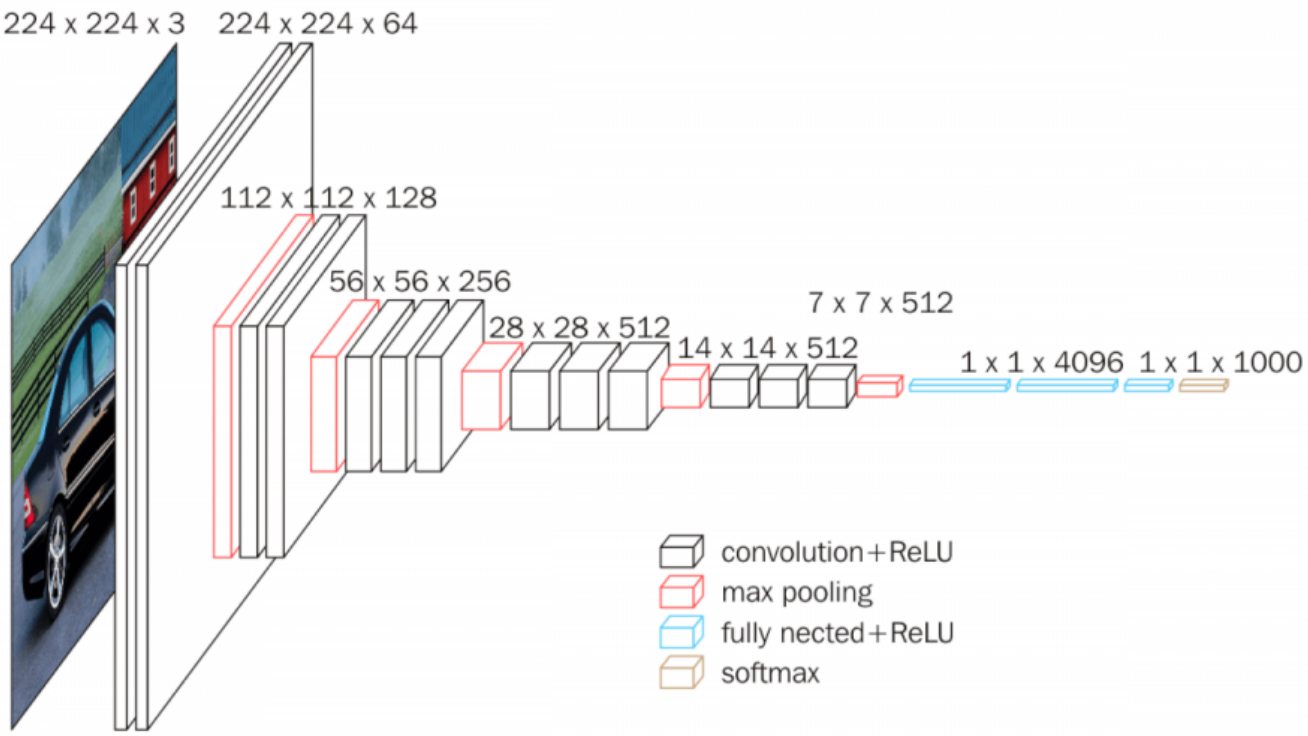
Neural Networks build Compact Representations

Example of the set of first-layer filters learned by a neural network trained on image classification (AlexNet):



These 96 filters capture most of the important low-level signal for image classification, and are much more compact than the exhaustive set of all possible filters (potentially thousands or millions of possible filters).

Neural Networks build Compact Representations



- Progressive tradeoff between spatial resolution and semantic resolution ensures that the representation remains compact at every step.

Part 2

Learning a Neural Network

Kernel models vs. neural networks

Kernel models

$$f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{w}$$

- Many problem formulations have their solutions given in closed form (e.g. ridge regression, Fisher discriminant, or take the simple form of a convex problem with a quadratic objective and linear constraints.

Deep models

$$f(\boldsymbol{x}) = f_3(f_2(f_1(\boldsymbol{x}, \theta_1), \theta_2), \theta_3)$$

- Objective is a complicated function of the parameters θ, and there is no closed form solution for θ, nor convexity.
- Commonly trained using gradient descent.

Simple algorithm for training a neural network

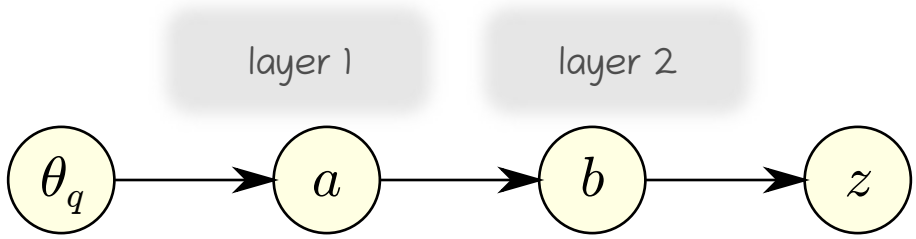
Basic gradient descent algorithm:

- Initialize vector of parameters θ at random.
- Iterate for T steps:
  - Compute the forward pass and the prediction error $\mathcal{E}$.
  - Extract the gradient $\partial\mathcal{E}/\partial\theta$ using backpropagation
  - Perform a gradient step $\theta = \theta - \gamma\partial\mathcal{E}/\partial\theta$

where γ is a learning rate that needs to be set by the user.

Better Approach: The Chain Rule

Suppose that some parameter of interest θ_q (one element of the parameter vector θ) is linked to the output of the network through some sequence of functions.



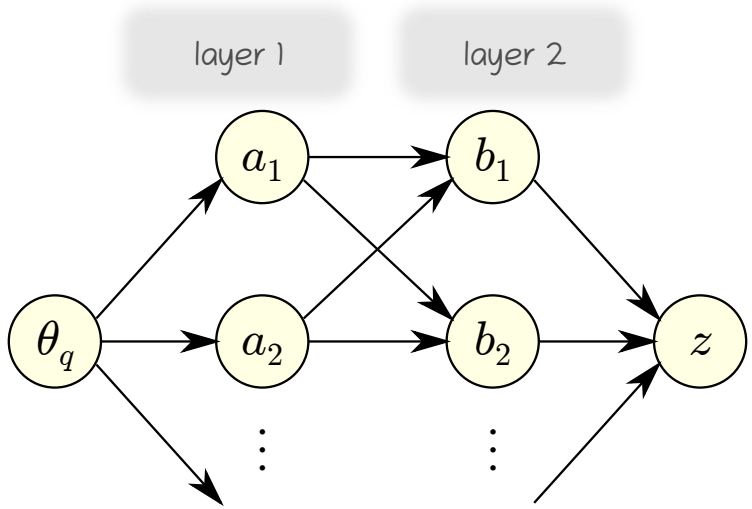
The chain rule for derivatives states that

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

i.e. the derivative w.r.t. the parameter of interest is the product of local derivatives along the path connecting θ_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):

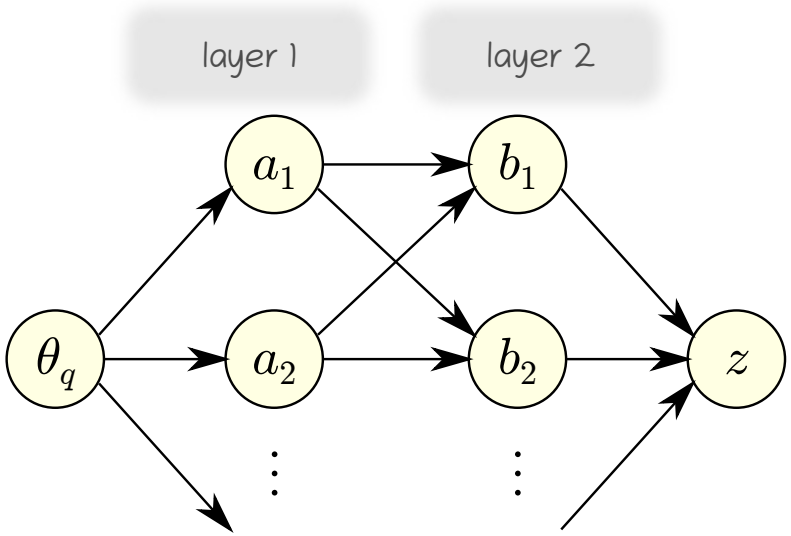


The chain rule can be extended to this multivariate scenario by enumerating all the paths between θ_q and z

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

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.

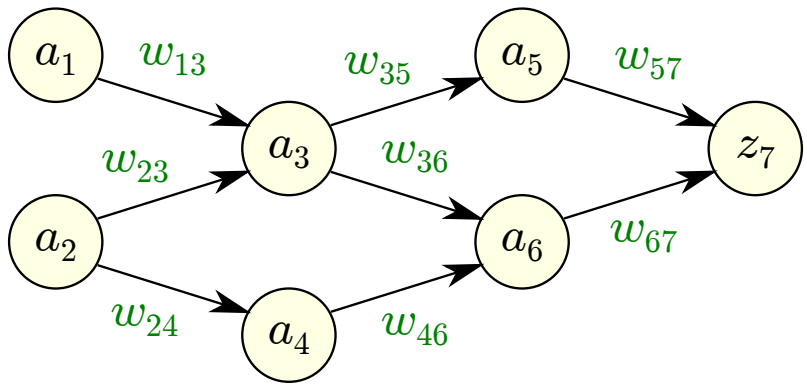
Factor Structure in the Multivariate Chain Rule



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

- 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).

Worked through Example

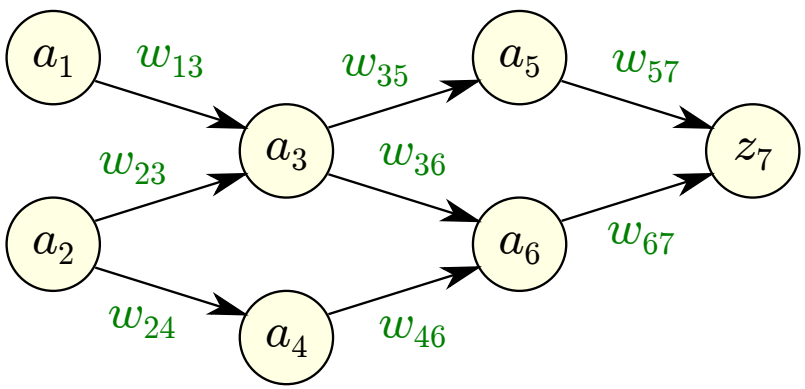


Forward pass:

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

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

Worked through Example

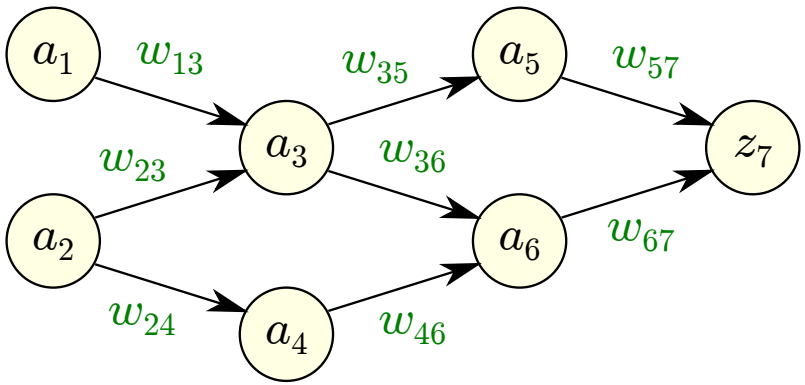


$$
\begin{aligned}
z_3 &= a_1 w_{13} + a_2 w_{23} & a_1 &= x_1 \\
z_4 &= a_2 w_{24} & a_2 &= x_2 \\
z_5 &= a_3 w_{35} & a_3 &= g(z_3) \\
z_6 &= a_3 w_{36} + a_4 w_{46} & a_4 &= g(z_4) \\
z_7 &= a_5 w_{57} + a_6 w_{67} & a_5 &= g(z_5) \\
\mathcal{E} &= \max(0, -z_7 t) & a_6 &= g(z_6)
\end{aligned}
$$

Backward pass:

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

Worked through Example

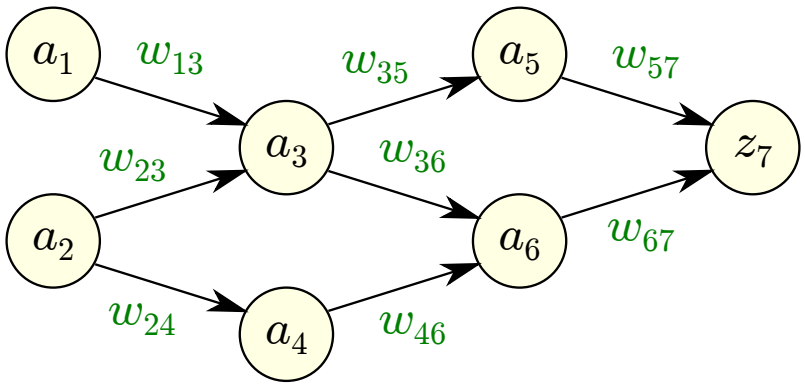


$$
\begin{aligned}
& & a_1 &= x_1 \\
& & a_2 &= x_2 \\
z_3 &= a_1 w_{13} + a_2 w_{23} & a_3 &= g(z_3) \\
z_4 &= a_2 w_{24} & a_4 &= g(z_4) \\
z_5 &= a_3 w_{35} & a_5 &= g(z_5) \\
z_6 &= a_3 w_{36} + a_4 w_{46} & a_6 &= g(z_6) \\
z_7 &= a_5 w_{57} + a_6 w_{67} & & \\
\mathcal{E} &= \max(0, -z_7 t) & &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

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

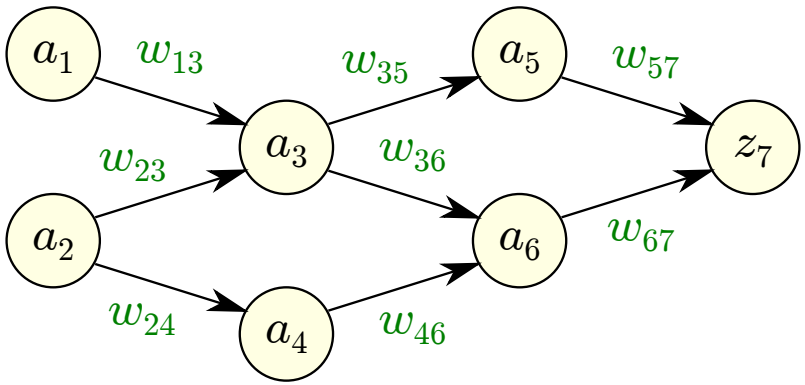


$$
\begin{aligned}
& & a_1 &= x_1 \\
& & a_2 &= x_2 \\
z_3 &= a_1 w_{13} + a_2 w_{23} & a_3 &= g(z_3) \\
z_4 &= a_2 w_{24} & a_4 &= g(z_4) \\
z_5 &= a_3 w_{35} & a_5 &= g(z_5) \\
z_6 &= a_3 w_{36} + a_4 w_{46} & a_6 &= g(z_6) \\
z_7 &= a_5 w_{57} + a_6 w_{67} & & \\
\mathcal{E} &= \max(0, -z_7 t) & &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

Backward pass:

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

Worked through Example

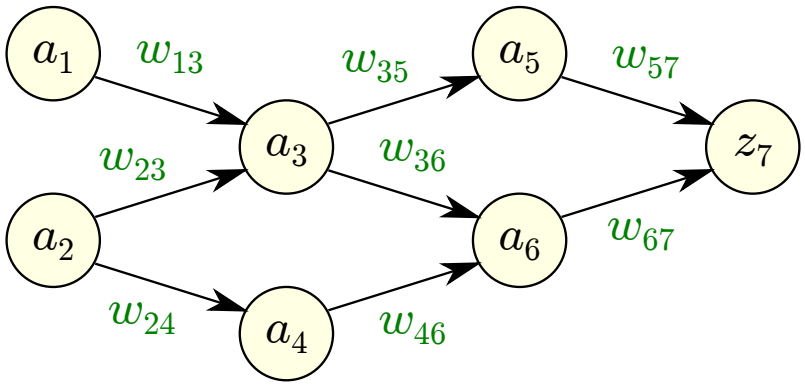


$$
\begin{aligned}
z_3 &= a_1 w_{13} + a_2 w_{23} & a_3 &= g(z_3) \\
z_4 &= a_2 w_{24} & a_4 &= g(z_4) \\
z_5 &= a_3 w_{35} & a_5 &= g(z_5) \\
z_6 &= a_3 w_{36} + a_4 w_{46} & a_6 &= g(z_6) \\
z_7 &= a_5 w_{57} + a_6 w_{67} \\
\mathcal{E} &= \max(0, -z_7 t)
\end{aligned}
$$

Backward pass:

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

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:

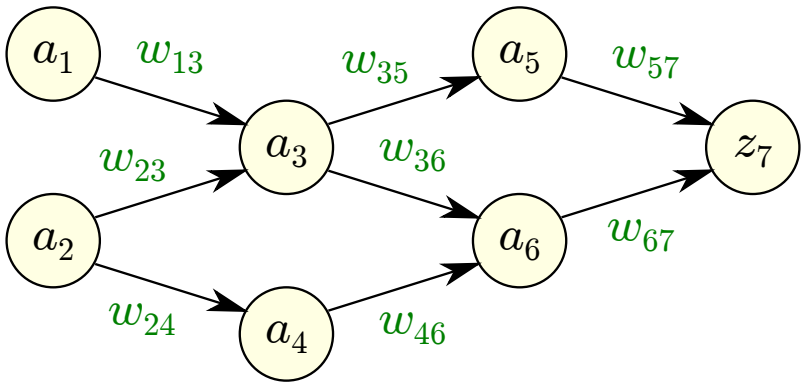
$$\delta_6 = \frac{\partial \mathcal{E}}{\partial a_6} = \frac{\partial \mathcal{E}}{\partial z_7}\frac{\partial z_7}{\partial a_6} = \delta_7 \cdot w_{67}$$

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

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

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

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:

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

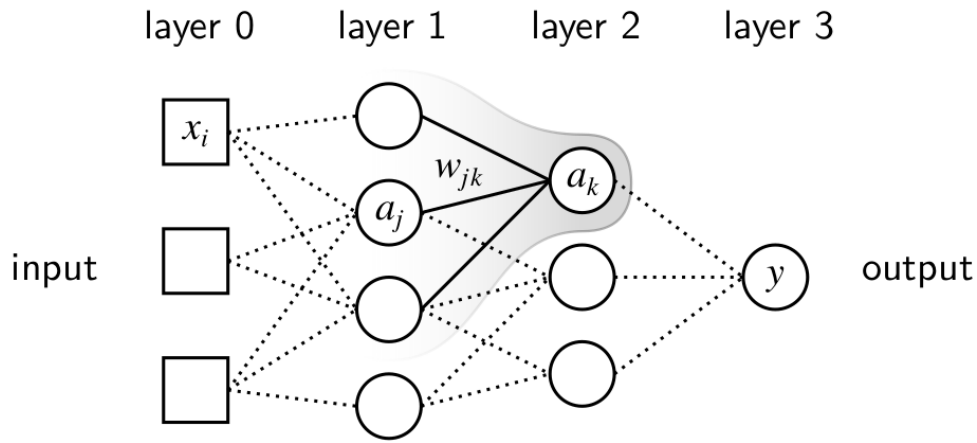
$$\delta_3 = \frac{\partial \mathcal{E}}{\partial a_3} = \frac{\partial \mathcal{E}}{\partial a_6}\frac{\partial a_6}{\partial z_6}\frac{\partial z_6}{\partial a_3} + \frac{\partial \mathcal{E}}{\partial a_5}\frac{\partial a_5}{\partial z_5}\frac{\partial z_5}{\partial a_3} = \delta_6 \cdot g'(z_6) \cdot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

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

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

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

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.

Automatic Differentiation

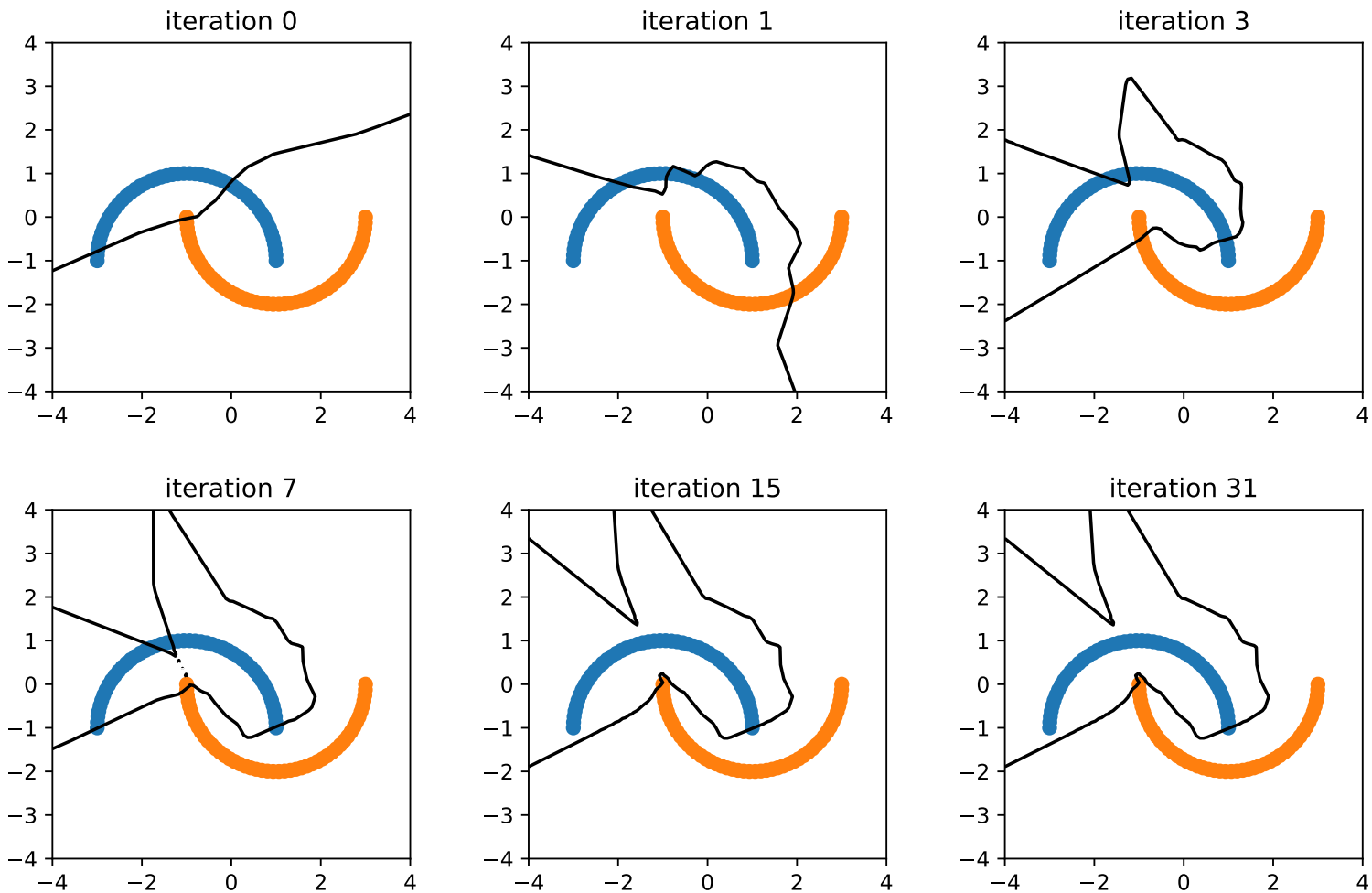
Automatic Differentiation:

- 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).

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 separable).

Summary

- Neural networks is a learning paradigm where both the classifier and the features supporting the classifier are learned from the data.
- Neural networks have the ability to represent and learn complex nonlinear functions through the interconnection of many simple computational units (neurons).
- Because neural networks learn the features, the representation can be made task-specific and reasonably **low-dimensional**.
- Neural networks do not have closed-form solutions or convex/quadratic formulations. They can be instead optimized via **gradient descent**.
- Gradient can be computed via **error backpropagation**, which is based on the **chain rule**. (In practice, backpropagation is readily implemented for us via automatic differentiation available in ML software such as PyTorch and TensorFlow.)