FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
NATURWISSENSCHAFTLICHE
FAKULTÄT

Review

Lecture “Mathematics of Learning” 2022/23

Wigand Rathmann
Friedrich-Alexander-Universität Erlangen-Nürnberg

WIRTSCHAFTS
MATHEMATIK

Module evaluation: **Dead line:** 10.02.2023

https://eva.fau.de/evasys/online.php?pswd=YR5NA

Loss function

Idea:
We measure the performance of a trained system using a metric $d\colon Y \times Y \to \mathbb{R}^+$.

- many metrics are possible, which lead to different realizations of the free parameters Θ
- typical examples: mean squared error, cross-entropy, ...
- based on a chosen metric d one defines the loss function C of f_Θ with respect to the free parameters Θ as:

$$C(\Theta) := \sum_{i=1}^{N} d[\, f_\Theta(\vec{x}^{(i)}), \vec{y}^{(i)}].$$

Clustering Algorithm

Given

- N number of data points
- M number of variables (i.e “mass”, “price”, “color”, ...)
- Data $X = \{x_1, \dots, x_N\}$, where $x_n \in \mathbb{R}^M$ for all $n = 1, \dots, N$
- K number of *assumed* clusters

Want

- *Assignment:* $x_n \mapsto k_n \in \{1, \dots, K\}$ for all $n = 1, \dots, N$
- *Assignment rule:* $\mathbf{x} \mapsto k(\mathbf{x}) \in \{1, \dots, K\}$ for all $x \in \mathbb{R}^M$
- *Reconstruction rule (’representative’):* $k \mapsto m_k \in \mathbb{R}^M$

On an abstract level:

- Determination of best possible clustering (w.r.t. some objective) is a classical combinatorial optimization problem
- K-means clustering: Determine K points, i.e., centers, that minimize the sum of the squared Euclidean distance to its closest center.

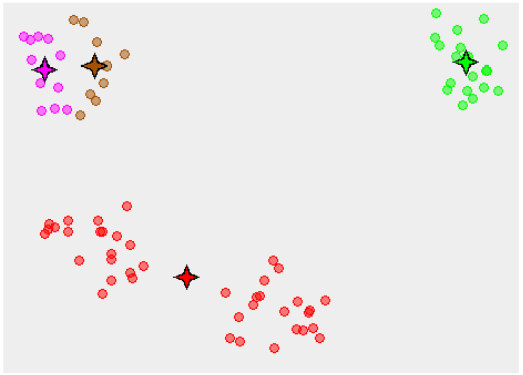
K-means clustering as optimization problem

Find clustering $\underline{C} = \{C_1, \ldots, C_K\}$ into sets $C_k \subset X$ and centers $\underline{m} = \{m_1, \ldots, m_K\}$ with $m_k \in C_k$, which minimize the clustering energy

$$E(\underline{C}, \underline{m}) := \frac{1}{2} \sum_{k=1}^{K} \sum_{x \in C_k} \|x - m_k\|^2 .$$

Observations

- The clustering energy has local minima



(picture from: https://upload.wikimedia.org/wikipedia/commons/7/7c/K-means_convergence_to_a_local_minimum.png, modified)

Derivation of the K-means algorithm

Let us fix the clustering $\underline{C}$ in

$$E(\underline{C}, \underline{m}) := \frac{1}{2} \sum_{k=1}^{K} \sum_{x \in C_k} \|x - m_k\|^2 .$$

necessary first-order optimality condition: gradient with respect to m_k is zero, i.e., a critical point.
Taking the gradient with respect to m_k we obtain the first-order optimality condition:

$$0 = \nabla_{m_k} E(\underline{C}, \underline{m}) = \sum_{x \in C_k} (x - m_k) = \sum_{x \in C_k} x - |C_k| m_k$$

and hence

$$m_k = \frac{1}{|C_k|} \sum_{x \in C_k} x \mathrel{\widehat{=}} \text{mean of the cluster}$$

problem: do not know the means, thus heuristically search for good means

Derivation of the K-means algorithm

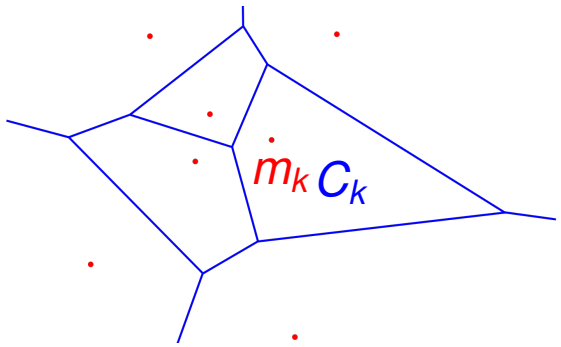
Conversely, let us fix the means $\underline{m}$ in

$$E(\underline{C}, \underline{m}) := \frac{1}{2} \sum_{k=1}^{K} \sum_{x \in C_k} \|x - m_k\|^2 .$$

perform the simple assignment step

$$C_k = \{x \in X : \|x - m_k\| \leq \left\|x - m_j\right\| \text{ for all } j = 1, \ldots, K\}$$

$\widehat{=}$ Voronoi cell of m_k



Computing PCA: Covariance matrix

Computation of covariance matrix $C \in \mathbb{R}^{M \times M}$:

$$\boxed{C := \frac{1}{N} \sum_{i=1}^{N} y^{(i)} y^{(i)^T}}$$

$$\begin{aligned} C_{k,l} &= \frac{1}{N} \sum_{i=1}^{N} y_k{}^{(i)} y_l{}^{(i)} = \frac{1}{N} \sum_{i=1}^{N} (y_k{}^{(i)} - 0)(y_l{}^{(i)} - 0) \\ &= \frac{1}{N} \sum_{i=1}^{N} (y_k{}^{(i)} - \overline{Y_k})(y_l{}^{(i)} - \overline{Y_l}) =: Cov(y_k, y_l) \end{aligned}$$

The actual PCA

Define transformation matrix:

$$T := (v^{(1)}, \dots, v^{(k)}) \in \mathbb{R}^{M \times k},$$

for which $v^{(1)}, \dots, v^{(k)}$ are the respective eigenvectors of the $1 \leq k \leq M$ largest eigenvalues.

Principal component analysis:

- transform the data: $z^{(i)} := T^T y^{(i)} = T^T (x^{(i)} - \overline{X})$ for $i = 1, \dots, N$
- $z^{(i)} \in \mathbb{R}^k$ contains the most relevant information (features) of the input data
- The components $z_j^{(i)}, j = 1, \dots, k$ are called **principal components**
- If T is quadratic ($k = M$) $\Rightarrow$ PCA is simply a rotation in $\mathbb{R}^M$

The principal components of the input data are typically used as (cluster) representatives in **clustering tasks**.

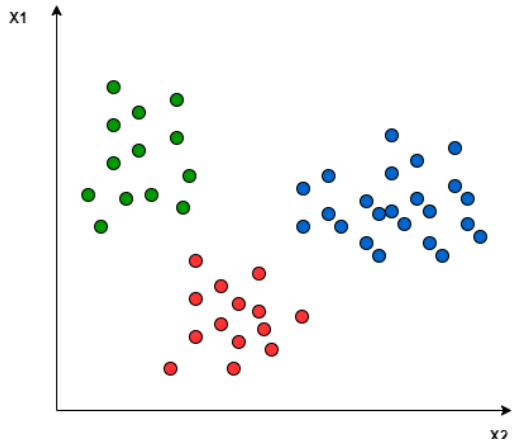
Summary of PCA

For given input data $x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^M$ the PCA can be computed as

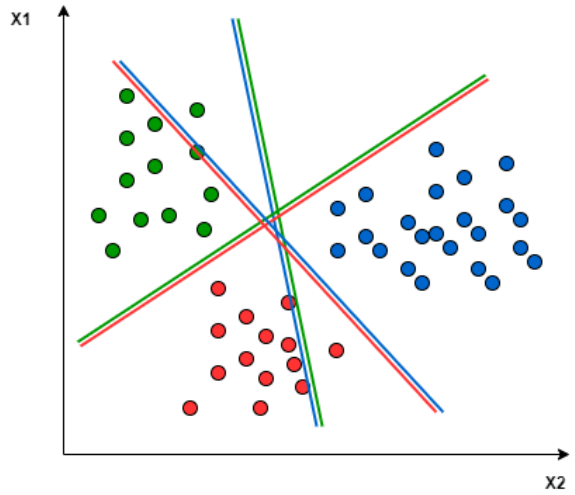
The (linear) PCA algorithm

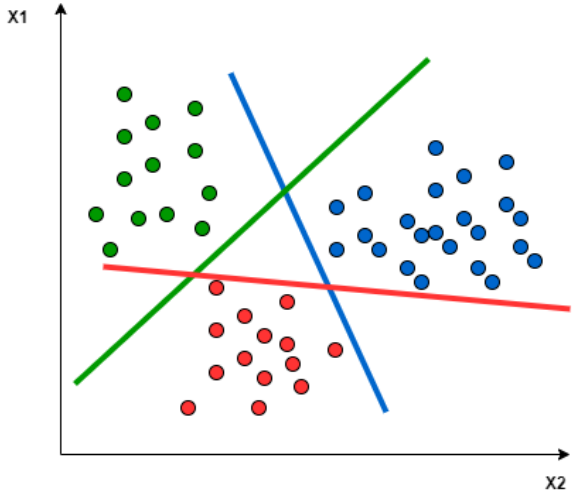
1. Compute mean value of data $\overline{X} = \frac{1}{N} \sum_{i=1}^{N} x^{(i)}$
2. Center data via $y^{(i)} = x^{(i)} - \overline{X}$
3. Compute covariance matrix $C = \frac{1}{N} \sum_{i=1}^{N} y^{(i)} y^{(i)^T}$
4. Determine the M eigenvalues and eigenvectors of C numerically
5. Select $1 \leq k \leq M$ respective eigenvectors $v^{(1)}, \ldots, v^{(k)}$ of the k largest non-vanishing eigenvalues
6. Assemble selected eigenvectors $v^{(1)}, \ldots, v^{(k)}$ columnwise to matrix $T \in \mathbb{R}^{M \times k}$
7. Compute principal components for each centered input point $y^{(i)} \in \mathbb{R}^M$ via:
$$T^T y^{(i)} = z^{(i)} \in \mathbb{R}^k$$

Support Vector Machines



with green points.





One version of the Optimization Problem

$$\min_{\beta,\beta_0,} \frac{1}{2}||\beta|| \quad \text{s.t. } y_i(x_i^\top \beta + \beta_0) \geq 1, i = 1, \ldots, N \tag{1}$$

The constraints define an empty slab or margin around the linear decision boundary of thickness $\frac{1}{||\beta||}$. By minimizing β and β_0, we maximize its thickness.

Lagrange Ansatz

Instead of (1), consider an unconstrained problem where the violation of the constraints is penalized in the objective. The Lagrange function models this: The Lagrange (primal) function, to be minimized w.r.t. β and β_0, is

$$L_P = \frac{1}{2}||\beta||^2 - \sum_{i=1}^{N} \alpha_i [y_i(x_i^\top \beta + \beta_0) - 1]$$

Karush-Kuhn Tucker conditions

Looking for critical points, we set the derivatives to zero and obtain:

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i, 0 = \sum_{i=1}^{N} \alpha_i y_i \tag{2}$$

Substituting this in Lagrange function, we obtain

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i^\top x_k, \alpha_i \geq 0$$

A solution has to satisfy the Karush-Kuhn Tucker conditions (2), $\alpha \geq 0$ and

$$\alpha_i [y_i (x_i^\top \beta + \beta_0) - 1] = 0 \forall i. \tag{3}$$

From this we can see

- if $\alpha_i > 0$, then $y_i (x_i^\top \beta + \beta_0) = 1$ or in other words, x_i is on the boundary of the slab;
- if $y_i (x_i^\top \beta + \beta_0) > 1$, x_i is not on the boundary of the slab, and $\alpha_i = 0$.

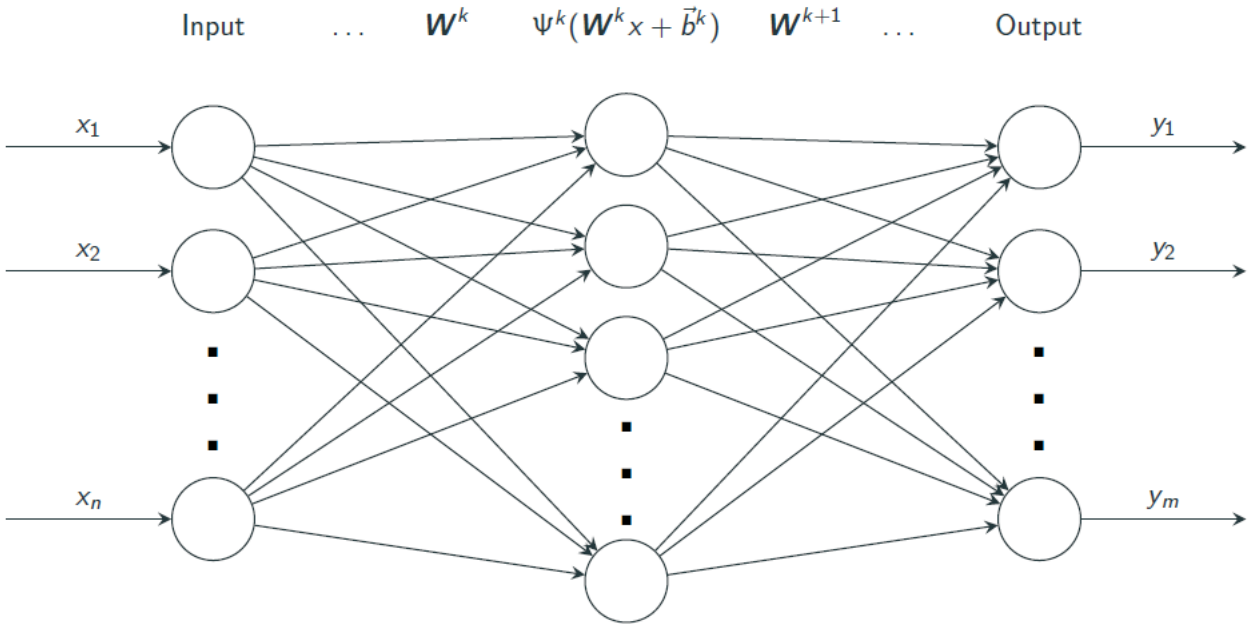
Artificial neural networks

Idea: Combine multiple perceptrons to perform more complex tasks.

- align artificial neurons in consecutive layers
  → convention: use designated input layer and output layer
  → all intermediate layers are called **hidden layer**
  → number of layers is called **depth** of the neural network
  → number of nonzero weights is called **connectivity** of the neural network
- artificial neural networks can be represented by directed graphs
- connections between neurons can be (almost) **arbitrary**
  → often there are no connections within same layer (except in recurrent neural networks)
  → certain network structures have proved to be successful for different applications, e.g., convolutional neural networks

Fully-connected feedforward neural network

Classical representation: Mappings from kth to $(k+1)$st layer:



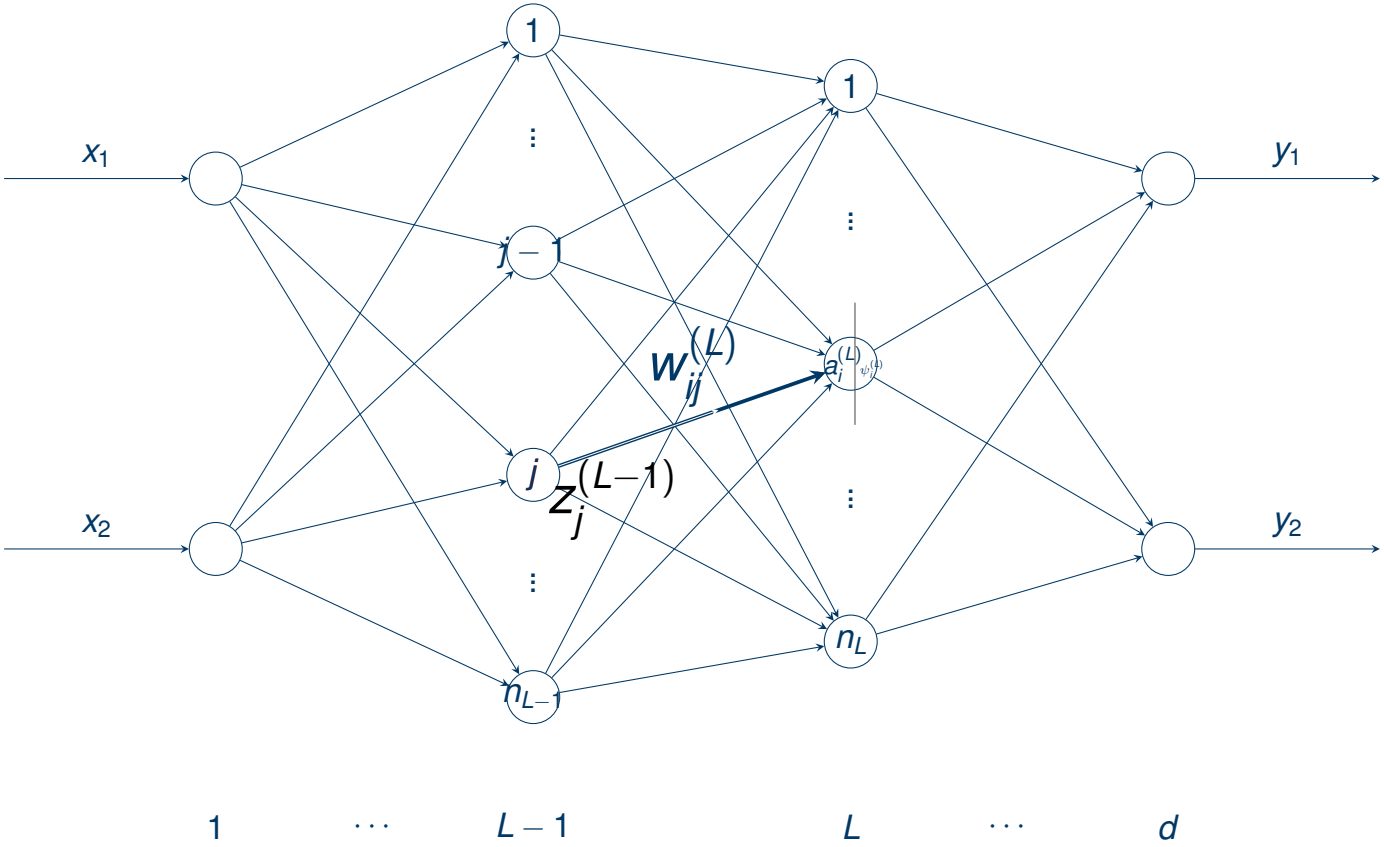
Compact representation:

Fully-connected feedforward neural network

- a fully-connected feedforward neural network can be written as a parametrized map $f_\Theta \colon \mathbb{R}^n \to \mathbb{R}^m$ that is realized by a concatenation of $d \in \mathbb{N}$ perceptron layers via
$$f_\Theta := f^d_{\Theta_d} \circ \ldots \circ f^1_{\Theta_1}$$
- each layer is a map $f^k_{\Theta_k} \colon \mathbb{R}^{n_{k-1}} \to \mathbb{R}^{n_k}$ with $f^k_{\Theta_k}(x) = \Psi^k(\boldsymbol{W}^k x + \vec{b}^k)$
- the free parameters can be written as matrix $\Theta_k = (\boldsymbol{W}^k, \vec{b}^k)$ with weights $\boldsymbol{W}^k \in \mathbb{R}^{n_k \times n_{k-1}}$ and biases $\vec{b}^k \in \mathbb{R}^{n_k}$
- the activation function Ψ^k acts pointwise on the resulting vector of the affine linear map, i.e., $\Psi^k(x_1, \ldots, x_{n_k}) := \big(\psi^k(x_1), \ldots, \psi^k(x_{n^k})\big)$ where $\psi^k \colon \mathbb{R} \to \mathbb{R}$ is the chosen activation function for this layer
- the network is fully-connected if each weight matrix $\boldsymbol{W}^k$ is fully occupied

Notation

Let f_θ be a fully-connected feedforward network of depth $d \in \mathbb{N}$.



Notation

Let f_θ be a fully-connected feedforward network of depth $d \in \mathbb{N}$. Then we denote:

- $L = 1, \ldots, d$ is the **layer index**
- $i = 1, \ldots, n_L$ is the **neuron index** for a layer L
- $j = 1, \ldots, n_{L-1}$ is the **neuron index** for the preceding layer $L-1$
- $w_{ij}^{(L)}$ is the **weight** between neuron j in layer $L-1$ and neuron i in layer L
- $z_j^{(L-1)}$ is the **output** of neuron j in layer $L-1$
- $a_i^{(L)} = \sum_{j=1}^{n_{L-1}} w_{ij}^{(L)} z_j^{(L-1)} + b_i^{(L)}$ is the **affine linear map** of neuron i in layer L
- $\psi_i^{(L)}$ is the **activation function** of neuron i in layer L,

For an index-free notation see the blog article by Dirk Lorenz (TU Braunschweig):

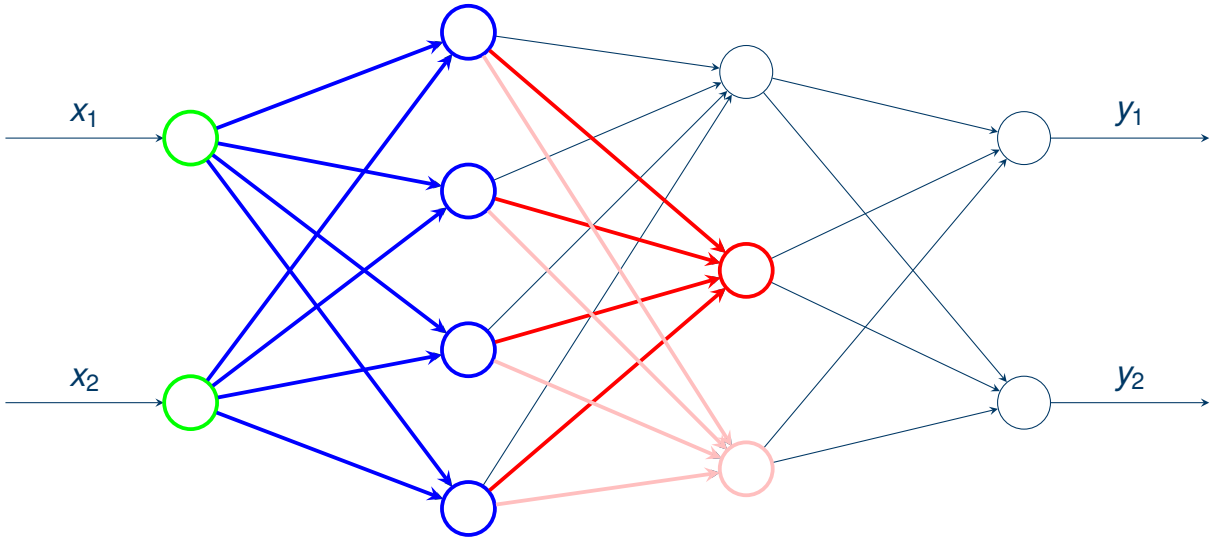
https://regularize.wordpress.com/2018/12/13/an-index-free-way-to-take-the-gradient-of-a-neural-network/

Forward propagation

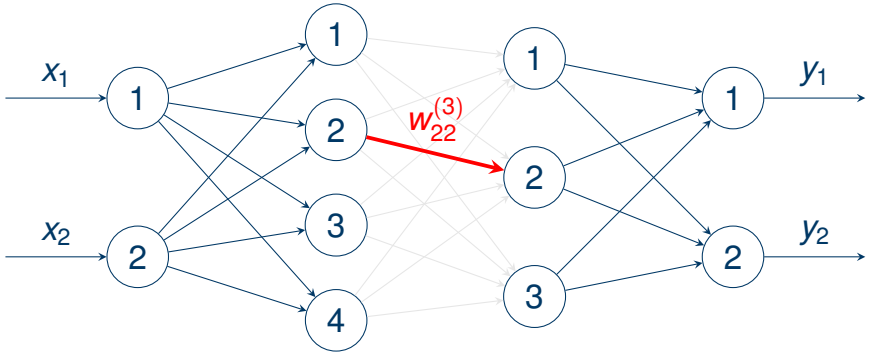
We can now express the output $z_i^{(L)}$ of any neuron i in any layer L as:

$$z_i^{(L)} = \psi_i^{(L)}(a_i^{(L)}) = \psi_i^{(L)}(\sum_{j=1}^{n_{L-1}} w_{ij}^{(L)} z_j^{(L-1)} + b_i^{(L)})$$

To compute this expression one first has to compute the output $z_j^{(L-1)}$ of all neurons in layer $L-1$. $\rightarrow$ **Recursion**



Partial derivative with respect to weight



$$\frac{\partial C}{\partial w_{22}^{(3)}} = \sum_{i=1}^{2} \left(\frac{\partial C}{\partial z_i^{(4)}} \cdot \frac{\partial z_i^{(4)}}{\partial a_i^{(4)}} \cdot \frac{\partial a_i^{(4)}}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial w_{22}^{(3)}} \right) = \left(\sum_{i=1}^{2} \frac{\partial C}{\partial z_i^{(4)}} \cdot \frac{\partial z_i^{(4)}}{\partial a_i^{(4)}} \cdot \frac{\partial a_i^{(4)}}{\partial z_2^{(3)}} \right) \cdot \frac{\partial z_2^{(3)}}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial w_{22}^{(3)}}$$

$$= \underbrace{\left(\sum_{i=1}^{2} (z_i^{(4)} - \vec{y}_i) \cdot (\psi_i'^{(4)}(a_i^{(4)})) \cdot w_{i2}^{(4)} \right) \cdot (\psi_2'^{(3)}(a_2^{(3)}))}_{= \frac{\partial C}{\partial a_2^{(3)}}} \cdot z_2^{(2)}$$

Observation: If we precompute the term $\frac{\partial C}{\partial a_2^{(3)}}$ once, we need only one multiplication for the partial derivative.

Stochastic Gradient Descent (SGD)

One of the most popular algorithms for contemporary data analysis!
Consider fixed network and training set, consider stochastic gradient descent

(1) randomly draw batch B from T

(2) $\theta \leftarrow \theta - \eta \frac{1}{|B|} \sum_{(x,y)\in B} \nabla_\theta L(f_\theta(x), y)$, go back to (1).

We note: This batch can be 'everything' from $|B| = 1$ (then one gradient is randomly chosen in each step) up to all data points in training set.
More abstractly we study

(1) sample gradient estimator g_k

(2) $\theta_{k+1} \leftarrow \theta_k - \eta_k g_k$,

(3) $k \leftarrow k + 1$, go back to (1),