Fachhochschule Graubünden
University of Applied Sciences

Methods & Algorithms

(formerly known as Introduction to Data Science)

FS 2023

Prof. Dr. rer. nat. habil. Ralf-Peter Mundani

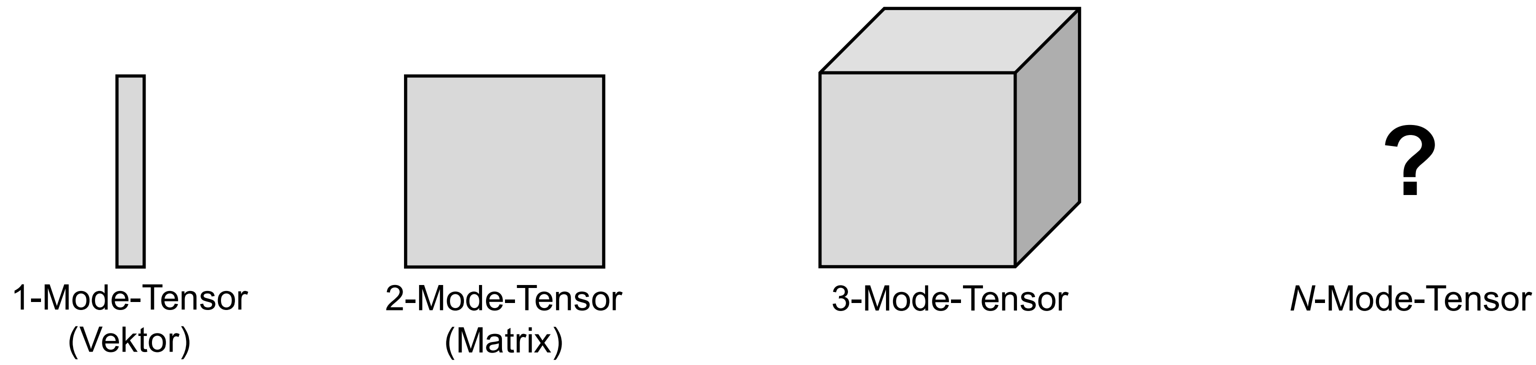
DAViS

Tensoren (nicht TensorFlow ☺)

- so etwas wie eine Definition
  - Tensoren sind mehrdimensionale Reihungen (Arrays) und damit eine Generalisierung von Matrizen in höhere Dimensionen
  - erstes Auftauchen im Bereich der Psychometrie
  - nicht zu verwechseln mit (Spannungs)Tensoren aus der Physik und dem Ingenieurwesen
- typische Einsatzgebiete
  - Dimensionsreduktion; Kompression von Bild- / Videodaten
  - Klassifikation
  - Deep Learning
  - *Offenlegung* latenter Information / latenter Beziehungen
  - …

Tensoren (nicht TensorFlow ☺)

- Tensorordnung
  - Anzahl der Dimensionen (auch als Mode bezeichnet)



- Tensorelemente
  - i-tes Element eines Vektors $\vec{\mathbf{v}} \rightarrow v_i$
  - Element (i, j) einer Matrix $\mathbf{A} \rightarrow a_{ij}$
  - Element (i, j, k) eines 3-Mode-Tensors $\mathfrak{X} \rightarrow x_{ijk}$

Lineare Algebra: Matrix Reloaded ☺

Kennen wir schon.

- dyadisches (äusseres) Produkt $\vec{\mathbf{v}} \cdot \vec{\mathbf{w}}^T$

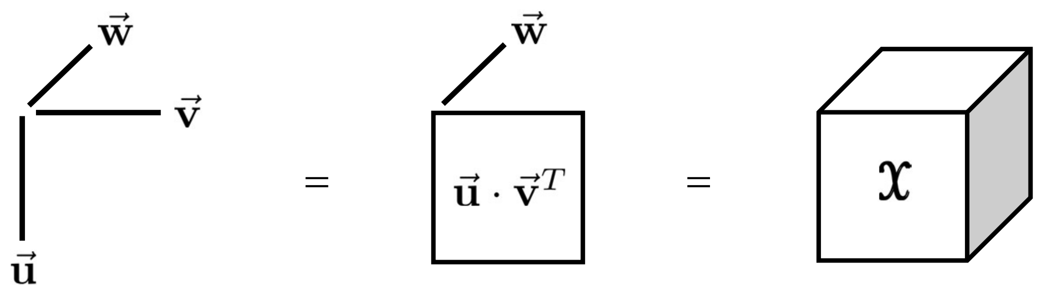
$$\vec{\mathbf{v}} \cdot \vec{\mathbf{w}}^T = \begin{pmatrix} v_{x_1} \\ v_{x_2} \\ \vdots \\ v_{x_m} \end{pmatrix} \cdot \begin{pmatrix} w_{x_1} & w_{x_2} & \cdots & w_{x_n} \end{pmatrix} = \underbrace{\begin{pmatrix} v_{x_1} \cdot w_{x_1} & v_{x_1} \cdot w_{x_2} & \cdots & v_{x_1} \cdot w_{x_n} \\ v_{x_2} \cdot w_{x_1} & v_{x_2} \cdot w_{x_2} & \cdots & v_{x_2} \cdot w_{x_n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{x_m} \cdot w_{x_1} & v_{x_m} \cdot w_{x_2} & \cdots & v_{x_m} \cdot w_{x_n} \end{pmatrix}}_{|\, m \times n \,|} = \mathbf{A}$$

- Beispiel: $\vec{\mathbf{v}} = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}, \vec{\mathbf{w}} = \begin{pmatrix} 4 \\ 0 \\ 3 \end{pmatrix}$

$$\vec{\mathbf{v}} \cdot \vec{\mathbf{w}}^T = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix} \cdot \begin{pmatrix} 4 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 3 \cdot 4 & 3 \cdot 0 & 3 \cdot 3 \\ 1 \cdot 4 & 1 \cdot 0 & 1 \cdot 3 \\ 2 \cdot 4 & 2 \cdot 0 & 2 \cdot 3 \end{pmatrix} = \begin{pmatrix} 12 & 0 & 9 \\ 4 & 0 & 3 \\ 8 & 0 & 6 \end{pmatrix}$$

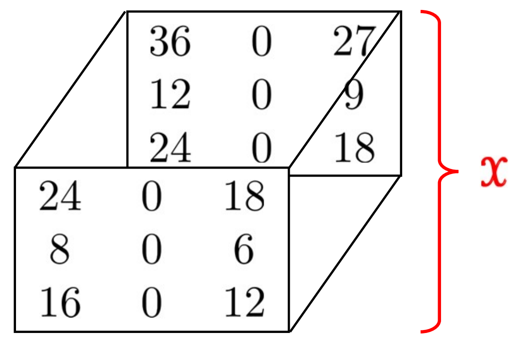
Lineare Algebra: Matrix Reloaded ☺

- was passiert bei drei (oder mehr) Vektoren…?
  - äusseres Produkt dreier Vektoren $\vec{\mathbf{u}} \circ \vec{\mathbf{v}} \circ \vec{\mathbf{w}} = \mathfrak{X}$



- Beispiel

$$\vec{\mathbf{u}} = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}, \vec{\mathbf{v}} = \begin{pmatrix} 4 \\ 0 \\ 3 \end{pmatrix}, \vec{\mathbf{w}} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \rightarrow \underbrace{\begin{pmatrix} 12 & 0 & 9 \\ 4 & 0 & 3 \\ 8 & 0 & 6 \end{pmatrix}}_{\vec{\mathbf{u}} \cdot \vec{\mathbf{v}}^T} \circ \begin{pmatrix} 2 \\ 3 \end{pmatrix} \rightarrow$$



Lineare Algebra: Matrix Reloaded ☺

- Berechnung des dyadischen Produkts

```
np.outer(vec1, vec2)
```

- Berechnung des äusseren Produkts

```
np.einsum('i,j,... -> ij...', vec1, vec2, ...)
```

- Beispiel: äusseres Produkt dreier Vektoren $\vec{\mathbf{u}} \circ \vec{\mathbf{v}} \circ \vec{\mathbf{w}} = \mathfrak{X}$

```
u = [3, 1, 2]
v = [4, 0, 3]
w = [2, 3, 4]
X = np.einsum('i,j,k->ijk', u, v, w)
```

Lineare Algebra: Matrix Revolutions ☺

- weitere Matrixprodukte
  - Kronecker-Produkt $\mathbf{A} \otimes \mathbf{B}$ mit $\mathbf{A}$ ($m \times n$) und $\mathbf{B}$ ($k \times l$)

$$\mathbf{A} \otimes \mathbf{B} = \underbrace{\begin{pmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2n}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & a_{m2}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{pmatrix}}_{|\, mk \times nl \,|}$$

- Beispiel

$$\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 3 & 3 \\ -2 & 4 \end{pmatrix} \quad \rightarrow \quad \mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} 6 & 6 & 3 & 3 \\ -4 & 8 & -2 & 4 \\ 3 & 3 & 9 & 9 \\ -2 & 4 & -6 & 12 \end{pmatrix}$$

Lineare Algebra (Teil 3): Matrix Revolutions ☺

- weitere Matrixprodukte
  - Khatri-Rao-Produkt $\mathbf{A} \odot \mathbf{B}$ mit $\mathbf{A}$ ($m \times n$) und $\mathbf{B}$ ($k \times n$)
  - spaltenweises Kronecker-Produkt mit $\mathbf{a}_i$ und $\mathbf{b}_i$ als i-te Spalte von $\mathbf{A}$ bzw. $\mathbf{B}$

$$\mathbf{A} \odot \mathbf{B} = \underbrace{\begin{pmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_n \otimes \mathbf{b}_n \end{pmatrix}}_{|\, mk \times n \,|}$$

- Beispiel: $\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$, $\mathbf{B} = \begin{pmatrix} 3 & 3 \\ -2 & 4 \end{pmatrix}$

$$\mathbf{A} \odot \mathbf{B} = \left(\begin{pmatrix} 2 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 3 \\ -2 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 3 \end{pmatrix} \otimes \begin{pmatrix} 3 \\ 4 \end{pmatrix} \right) = \begin{pmatrix} 6 & 3 \\ -4 & 4 \\ 3 & 9 \\ -2 & 12 \end{pmatrix}$$

Lineare Algebra: Matrix Revolutions ☺

- weitere Matrixprodukte
  - Hadamard-Produkt $\mathbf{A} * \mathbf{B}$ mit $\mathbf{A}$ und $\mathbf{B}$ jeweils ($m \times n$)

$$\mathbf{A} * \mathbf{B} = \underbrace{\begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2n}b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & a_{m2}b_{m2} & \cdots & a_{mn}b_{mn} \end{pmatrix}}_{|\, m \times n \,|}$$

- Beispiel

$$\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 3 & 3 \\ -2 & 4 \end{pmatrix} \quad \rightarrow \quad \mathbf{A} * \mathbf{B} = \begin{pmatrix} 6 & 3 \\ -2 & 12 \end{pmatrix}$$

Lineare Algebra: Matrix Revolutions ☺

- Paket **`tensorly`**
  - Kronecker-Produkt $\mathbf{A} \otimes \mathbf{B}$

```
import tensorly as tl
tl.tenalg.kronecker((A, B))
```

  - Khatri-Rao-Produkt $\mathbf{A} \odot \mathbf{B}$

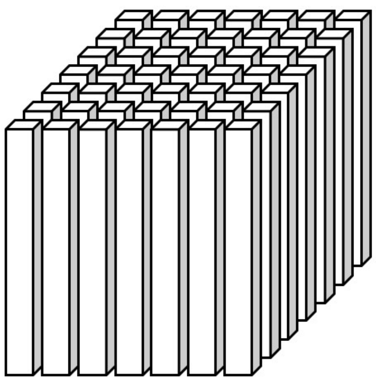
```
tl.tenalg.khatri_rao((A, B))
```

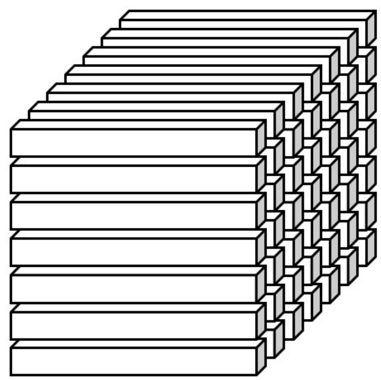
  - Hadamard-Produkt $\mathbf{A} * \mathbf{B}$ ☺

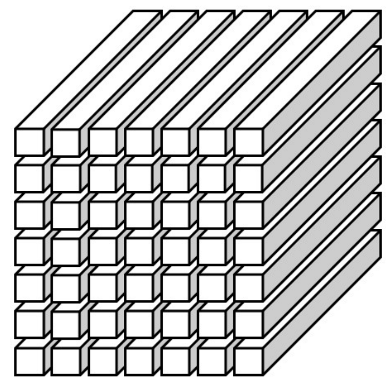
```
A*B
```

Von Fasern, Schichten und anderen Ballaststoffen…

- Tensorfasern
  - höherdimensionales Analogon zu Zeilen / Spalten einer Matrix
  - Entstehung durch 'Festhalten' aller Indizes bis auf einen
- Beispiel: 3-Mode Tensor $\mathcal{X}$
  - Spalten-, Zeilen und Röhrenfasern als $\mathbf{x}_{:jk}$, $\mathbf{x}_{i:k}$ bzw. $\mathbf{x}_{ij:}$

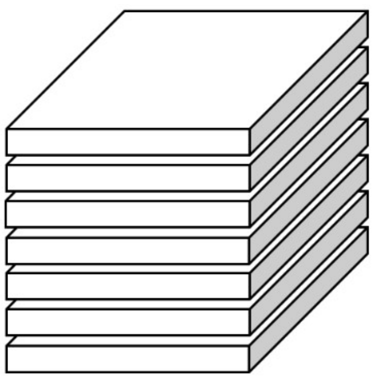

Mode-0-Fasern $\mathbf{x}_{:jk}$


Mode-1-Fasern $\mathbf{x}_{i:k}$

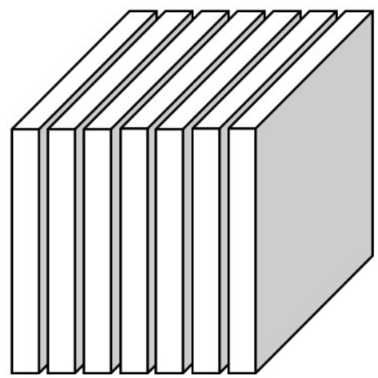

Mode-2-Fasern $\mathbf{x}_{ij:}$

Von Fasern, Schichten und anderen Ballaststoffen…

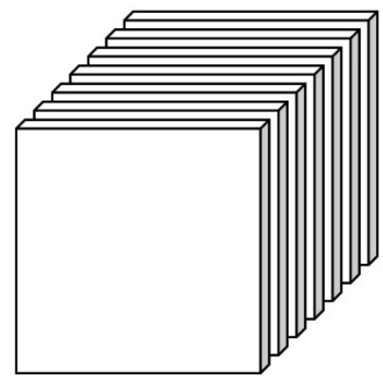
- Tensorschichten
  - zweidimensionaler Ausschnitt eines Tensors
  - Entstehung durch 'Festhalten' aller Indizes bis auf zwei
- Beispiel: 3-Mode Tensor $\mathcal{X}$
  - horizontale, laterale und frontale Schichten als $\mathbf{X}_{i::}$, $\mathbf{X}_{:j:}$ bzw. $\mathbf{X}_{::k}$



horizontale Schichten $\mathbf{X}_{i::}$



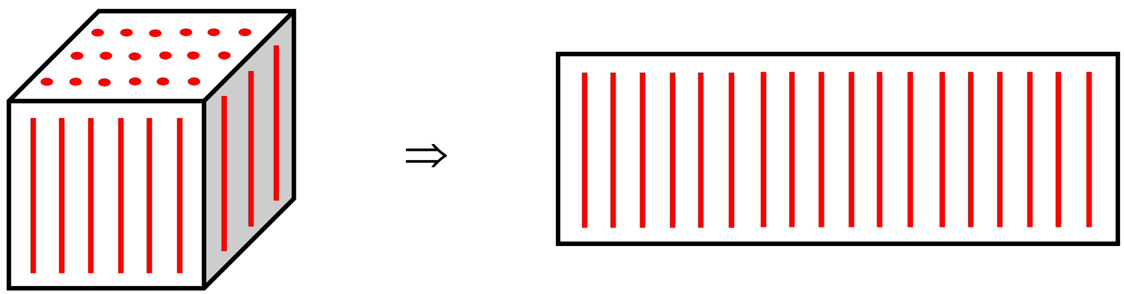
laterale Schichten $\mathbf{X}_{:j:}$



frontale Schichten $\mathbf{X}_{::k}$

Von Fasern, Schichten und anderen Ballaststoffen…

- Matrifizierung (Entfaltung / Ebnung eines Tensors)
  - Idee: Umordnung der Elemente eines *N*-Mode Tensors in eine Matrix
  - Mode-*n*-Matrifizierung $\mathbf{X}_{(n)} \rightarrow$ Umordnung der Mode-*n*-Fasern als Spalten



Von Fasern, Schichten und anderen Ballaststoffen…

- Matrifizierung (Entfaltung / Ebnung eines Tensors)
  - Beispiel: 3-Mode Tensor $\mathcal{X}$ ($3 \times 4 \times 2$) als frontale Schichten

$$\mathbf{X}_{::0} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 9 & 11 & 13 & 15 \\ 17 & 19 & 21 & 23 \end{bmatrix} \qquad \mathbf{X}_{::1} = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 10 & 12 & 14 & 16 \\ 18 & 20 & 22 & 24 \end{bmatrix}$$

$\Rightarrow$ Mode-0-Entfaltung

```
zuerst X0 erste Spalte als Spalte, dann X1 erste Spalte als Spalte,
X0 zweite Spalte....
```

$$\mathbf{X}_{(0)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{bmatrix}$$

Von Fasern, Schichten und anderen Ballaststoffen…

- Matrifizierung (Entfaltung / Ebnung eines Tensors)
  - Beispiel: 3-Mode Tensor $\mathcal{X}$ ($3 \times 4 \times 2$) als frontale Schichten

$$\mathbf{X}_{::0} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 9 & 11 & 13 & 15 \\ 17 & 19 & 21 & 23 \end{bmatrix} \qquad \mathbf{X}_{::1} = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 10 & 12 & 14 & 16 \\ 18 & 20 & 22 & 24 \end{bmatrix}$$

$\Rightarrow$ Mode-1-Entfaltung

```
zuerst X0 erste Zeile als Spalte, dann X1 erste Zeile als Spalte,
X0 zweite Zeile als Spalte....
```

$$\mathbf{X}_{(1)} = \begin{bmatrix} 1 & 2 & 9 & 10 & 17 & 18 \\ 3 & 4 & 11 & 12 & 19 & 20 \\ 5 & 6 & 13 & 14 & 21 & 22 \\ 7 & 8 & 15 & 16 & 23 & 24 \end{bmatrix}$$

Von Fasern, Schichten und anderen Ballaststoffen…

- Matrifizierung (Entfaltung / Ebnung eines Tensors)
  - Beispiel: 3-Mode Tensor $\mathfrak{X}$ ($3 \times 4 \times 2$) als frontale Schichten

$$\mathbf{X}_{::0} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 9 & 11 & 13 & 15 \\ 17 & 19 & 21 & 23 \end{bmatrix} \qquad \mathbf{X}_{::1} = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 10 & 12 & 14 & 16 \\ 18 & 20 & 22 & 24 \end{bmatrix}$$

```
räumliche 3. DImension
```

$\Rightarrow$ Mode-2-Entfaltung

```
erste Wert in Zeile aus X0 in 1. Spalte 1. Zeile,
erster Wert in Zeile aus X1 in 1. Spalte 2. Zeile
zweiter Wert in Zeile aus X0 in 2. Spalte 1. Zeile...
```

$$\mathbf{X}_{(2)} = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 & 18 & 20 & 22 & 24 \end{bmatrix}$$

Von Fasern, Schichten und anderen Ballaststoffen…

- Erzeugen eines Tensors $\mathcal{X}$ $(3 \times 4 \times 2)$

```
X = np.arange(24).reshape((3, 4, 2))
X = X + 1   // Zahlen liegen in [0,23] → Zahlen liegen in [1,24]
```

- Darstellen der frontalen Schichten $\mathbf{X}_{::0}$ und $\mathbf{X}_{::1}$

```
X[:,:,0]
X[:,:,1]
```

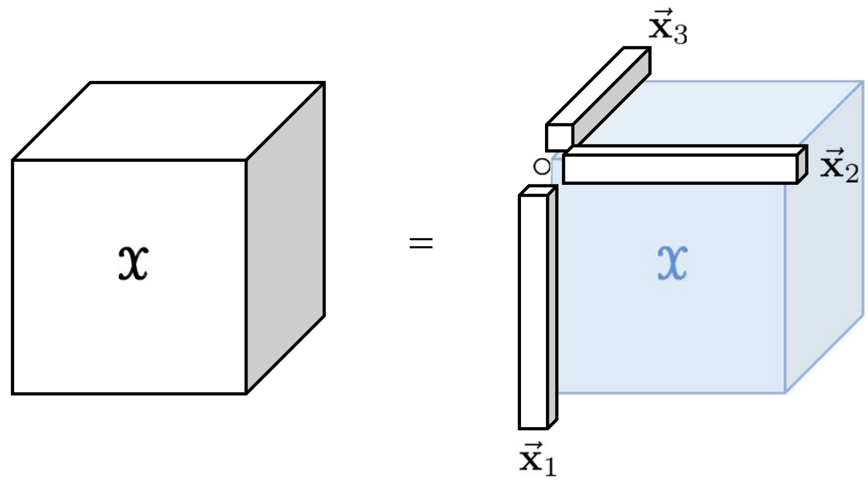
- Mode-*n*-Entfaltung $\mathbf{X}_{(0)}$, $\mathbf{X}_{(1)}$ und $\mathbf{X}_{(2)}$

```
tl.unfold(X,0)
tl.unfold(X,1)
tl.unfold(X,2)
```

Tensorfaktorisierung

- Rang-1-Tensoren (kurz R1-Tensor)
  - ein *N*-Mode-Tensor $\mathfrak{X}$ ist vom Rang 1 $\Leftrightarrow \mathfrak{X} = \vec{\mathbf{x}}_1 \circ \vec{\mathbf{x}}_2 \circ \cdots \circ \vec{\mathbf{x}}_N$

- Beispiel: 3-Mode-R1-Tensor

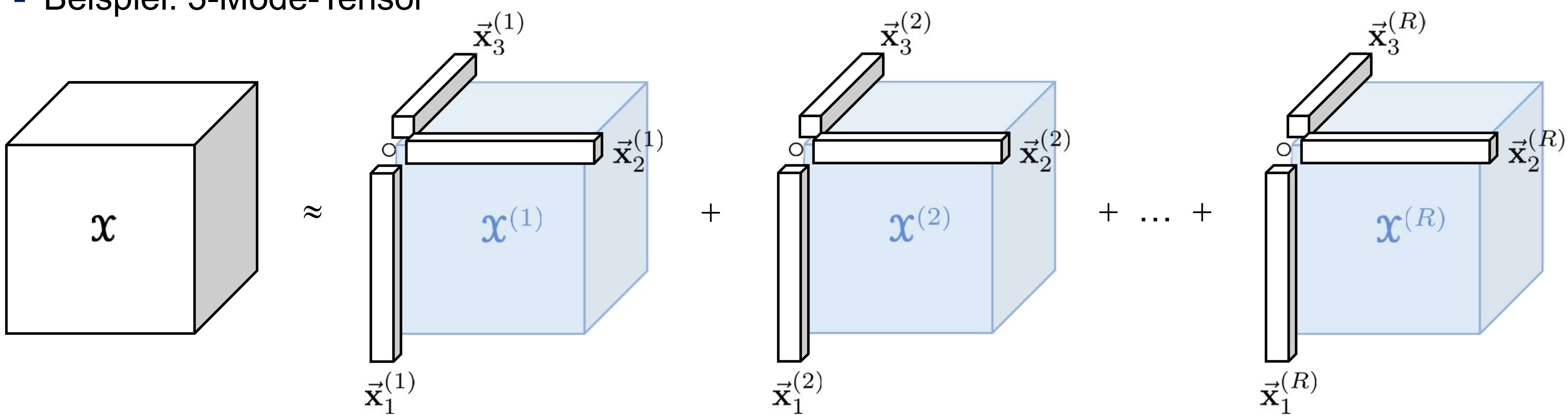


- Berechnung der Elemente eines *N*-Mode-R1-Tensors als $x_{i_1 i_2 \cdots i_N} = x_{1,i_1} \cdot x_{2,i_2} \cdot \cdots \cdot x_{N,i_N}$

Tensorfaktorisierung

- CANDECOMP / PARAFAC (*canonical decomposition* / *parallel factors*; kurz CP)
  - Idee: Zerlegung eines *N*-Mode-Tensors $\mathfrak{X}$ in *R* Rang-1-Tensoren

- Beispiel: 3-Mode-Tensor



- Problem: Rang($\mathfrak{X}$) $=$ *R* ist a-priori nicht bekannt

Tensorfaktorisierung

- CANDECOMP / PARAFAC
  - CP berechnet für einen *N*-Mode-Tensor $\mathfrak{X}$ folgende Zerlegung

$$\mathfrak{X} \approx \sum_{r=1}^{R} \vec{\mathbf{x}}_1^{(r)} \circ \vec{\mathbf{x}}_2^{(r)} \circ \cdots \circ \vec{\mathbf{x}}_N^{(r)}$$

  - Vektoren $\vec{\mathbf{x}}_i^{(1)}$, $\vec{\mathbf{x}}_i^{(2)}$, …, $\vec{\mathbf{x}}_i^{(R)}$ der *i*-ten Dimension definieren die Spalten einer Matrix $\mathbf{A}_i$

$$\mathbf{A}_1 = \begin{pmatrix} \vec{\mathbf{x}}_1^{(1)} & \vec{\mathbf{x}}_1^{(2)} & \cdots & \vec{\mathbf{x}}_1^{(R)} \end{pmatrix}, \mathbf{A}_2 = \begin{pmatrix} \vec{\mathbf{x}}_2^{(1)} & \vec{\mathbf{x}}_2^{(2)} & \cdots & \vec{\mathbf{x}}_2^{(R)} \end{pmatrix}, \ldots, \mathbf{A}_N = \begin{pmatrix} \vec{\mathbf{x}}_N^{(1)} & \vec{\mathbf{x}}_N^{(2)} & \cdots & \vec{\mathbf{x}}_N^{(R)} \end{pmatrix}$$

  - mithilfe der Faktormatrizen $\mathbf{A}_i$ lässt sich Tensor $\mathfrak{X}$ auch schreiben als

$$\mathfrak{X} \approx [\![\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N]\!] \quad \text{bzw.} \quad \mathfrak{X} \approx [\![\vec{\lambda}; \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N]\!]$$

R-dimensionaler Vektor mit Gewichten (Normierung)

Tensorfaktorisierung

- Erzeugen eines Tensors $\mathcal{X}$ $(3 \times 4 \times 2)$

```
X = np.arange(24.).reshape((3, 4, 2))
X = X + 1.    // Zahlen liegen in [1.,24.]
```

- CP-Faktorisierung $\mathcal{X} \rightarrow [\![\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3]\!]$

```
from tensorly.decomposition import parafac
factors = parafac(X, rank)
```

- Tensor aus Faktormatrizen rekonstruieren $[\![\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3]\!] \rightarrow \widehat{\mathcal{X}}$

```
X_rec = tl.kruskal_to_tensor(factors)
tl.norm(X - X_rec)
```

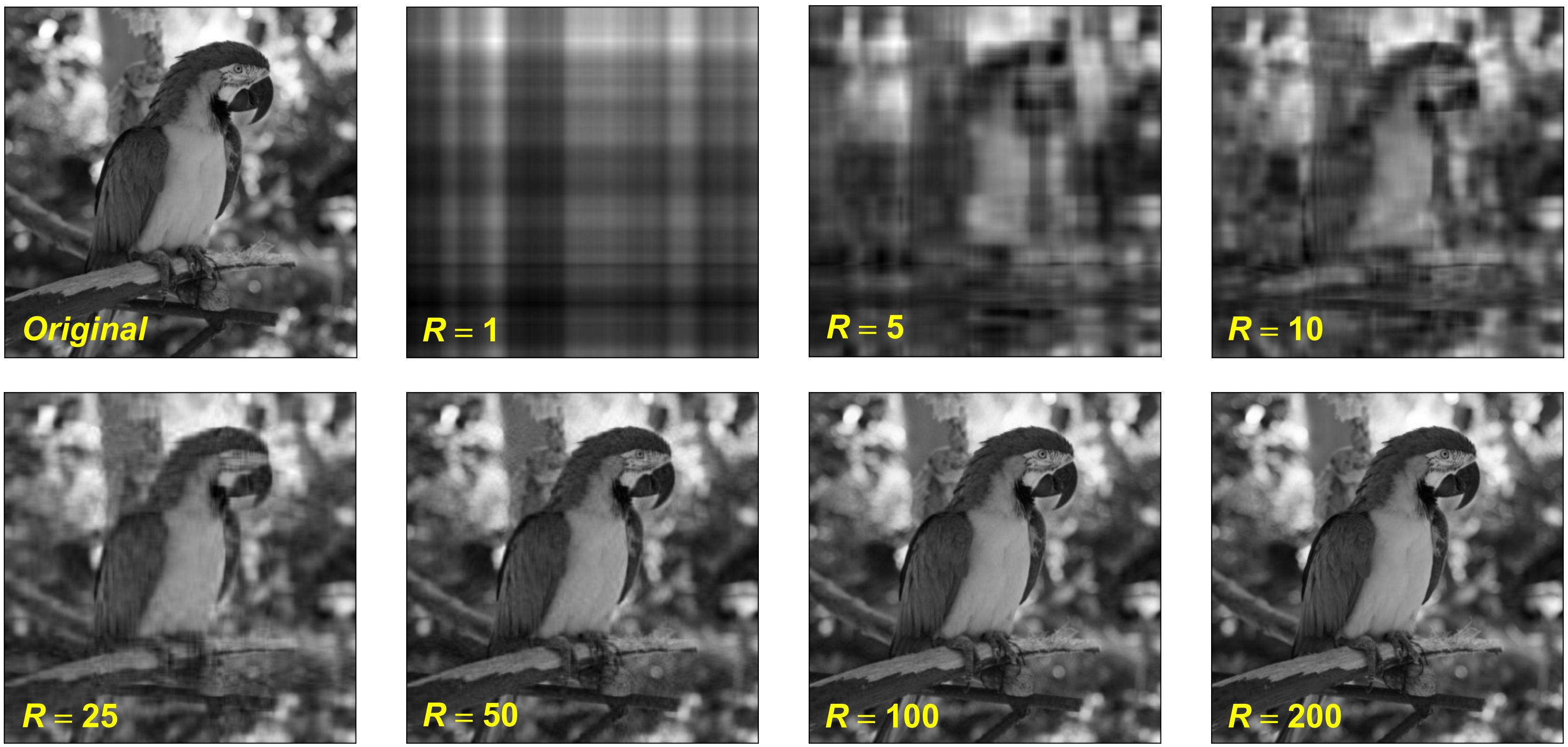
Tensorfaktorisierung

- mal wieder Lora… (Bildkompression durch CP-Faktorisierung)

```
import tensorly as tl
from tensorly.decomposition import parafac

img = plt.imread('papa.png')

factors = parafac(img, rank ∈ {1,5,10,25,50,100,200})
img_rec = tl.kruskal_to_tensor(factors)

plt.imshow(img_rec, cmap='gray')
plt.show()
```

Tensorfaktorisierung

- mal wieder Lora… (Bildkompression durch CP-Faktorisierung)



Tensorfaktorisierung

- was passiert hier…?
  - Vorüberlegung: Matrifizierung eines 3-Mode-Tensors $\mathcal{X} \approx [\![\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3]\!]$

$$\mathbf{X}_{(1)} \approx \mathbf{A}_1(\mathbf{A}_3 \odot \mathbf{A}_2)^T$$

$$\mathbf{X}_{(2)} \approx \mathbf{A}_2(\mathbf{A}_3 \odot \mathbf{A}_1)^T$$

$$\mathbf{X}_{(3)} \approx \mathbf{A}_3(\mathbf{A}_2 \odot \mathbf{A}_1)^T$$

  - oder allgemein für einen *N*-Mode-Tensor $\mathcal{X} \approx [\![\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N]\!]$

$$\mathbf{X}_{(n)} \approx \mathbf{A}_n(\mathbf{A}_N \odot \mathbf{A}_{N-1} \odot \cdots \odot \mathbf{A}_{n+1} \odot \mathbf{A}_{n-1} \odot \cdots \odot \mathbf{A}_2 \odot \mathbf{A}_1)^T$$

Tensorfaktorisierung

- Idee für Algorithmus
  - Beispiel: 3-Mode-Tensor $\mathcal{X}$ mit Rang($\mathcal{X}$) = R

wiederhole

fixiere $\mathbf{A}_2, \mathbf{A}_3 \Rightarrow$ löse für $\mathbf{A}_1$

fixiere $\mathbf{A}_1, \mathbf{A}_3 \Rightarrow$ löse für $\mathbf{A}_2$ mit $\widehat{\mathcal{X}} = [\![\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3]\!]$

fixiere $\mathbf{A}_1, \mathbf{A}_2 \Rightarrow$ löse für $\mathbf{A}_3$

bis $\| \mathcal{X} - \widehat{\mathcal{X}} \| < \varepsilon$

  - Lösung für $\mathbf{A}_i$ mittels Verfahren der kleinsten Fehlerquadrate (Stichwort: Regression), z.B.

$$\min_{\widehat{\mathbf{A}}_1} \| \mathbf{X}_{(1)} - \widehat{\mathbf{A}}_1 (\mathbf{A}_3 \odot \mathbf{A}_2)^T \| \quad \Leftrightarrow \quad \widehat{\mathbf{A}}_1 = \mathbf{X}_{(1)} (\mathbf{A}_3 \odot \mathbf{A}_2)(\mathbf{A}_3^T \mathbf{A}_3 * \mathbf{A}_2^T \mathbf{A}_2)^+$$

Tensorfaktorisierung

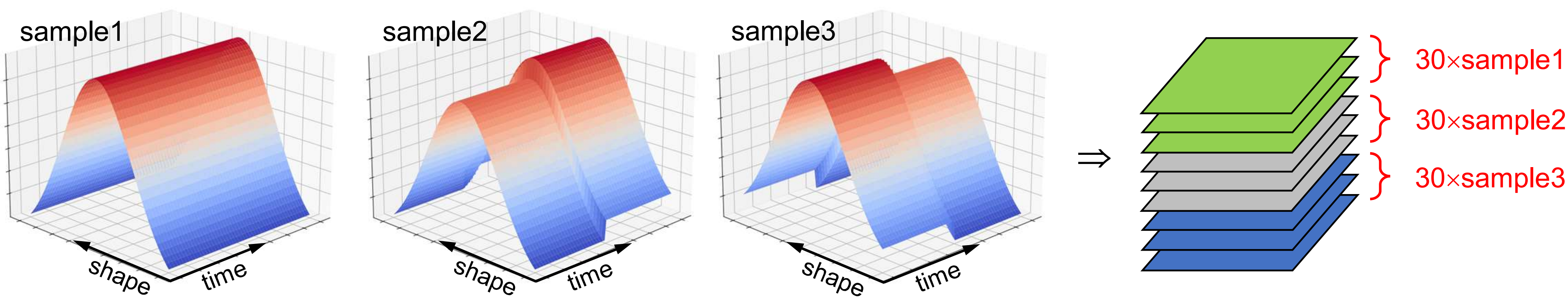
- zwei Probleme bleiben
  1) wie werden die $\mathbf{A}_i$ initialisiert…?
  2) wie wird Rang($\mathfrak{X}$) bestimmt…?

- Problem (1): Initialisierung der $\mathbf{A}_i$
  - berechne SVD($\mathbf{X}_{(i)}) = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T$ und initialisiere $\mathbf{A}_i$ mit den linken *R* Spalten von $\mathbf{U}$

- Problem (2): Bestimmung Rang($\mathfrak{X}$)
  - simpler, ABER ineffizienter Algorithmus
    i. setze $R = 1$
    ii. berechne Zerlegung $\widehat{\mathfrak{X}}$ und bestimme Fehler $\| \mathfrak{X} - \widehat{\mathfrak{X}} \|$
    iii. erhöhe *R* um 1 und wiederhole Schritt (ii) bis sich Fehler nicht mehr (gross) verändert

Tensorfaktorisierung

- Beispiel zur Analyse mehrdimensionaler Daten (1)
  - 3-Mode-Tensor mit
    - einem geometrischen Profil (shape)
    - einer zeitlichen Veränderung (time)
    - drei verrauschten Datensätzen (sample)

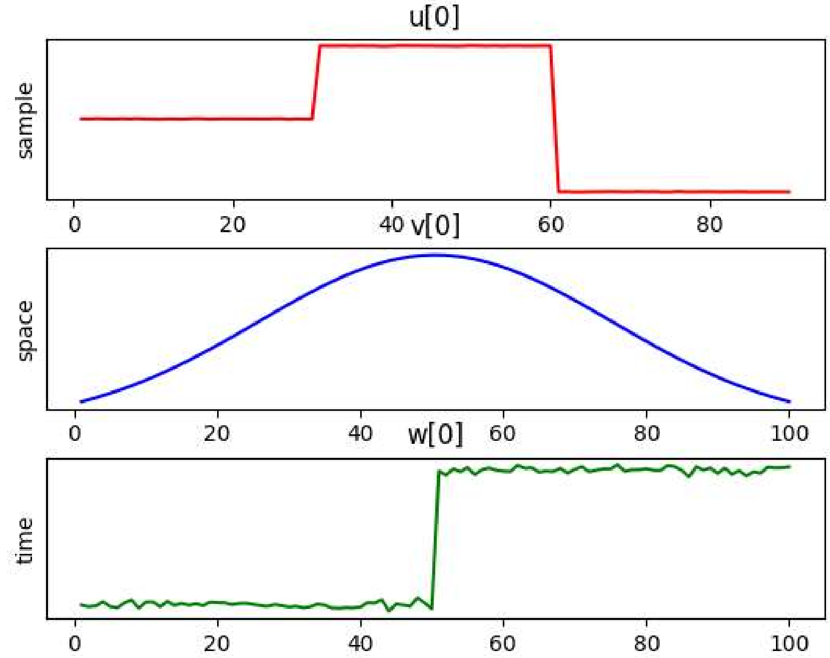


Tensorfaktorisierung

- Beispiel zur Analyse mehrdimensionaler Daten (1)
  - Datei **`mytensor.py`**

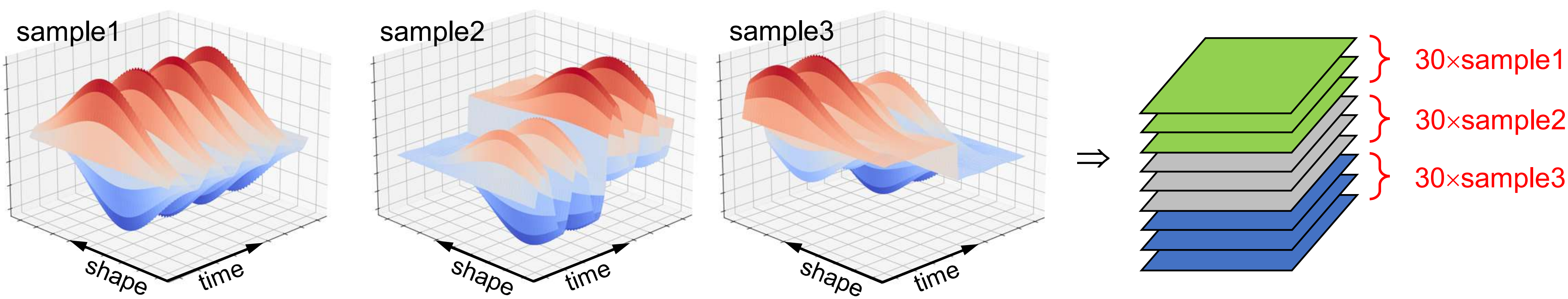
```
from mytensor import gen_tensor_one_feature as gen1f
from mytensor import plot_uvw_one_feature as plot1f
```

```
X = gen1f()
w_, fac = parafac(X, 1)
plot1f(fac)
```



Tensorfaktorisierung

- Beispiel zur Analyse mehrdimensionaler Daten (2)
  - 3-Mode-Tensor mit
    - verschiedenen geometrischen Profilen (shape)
    - wellenförmiger Ausbreitung und einer zeitlichen Veränderung (time)
    - drei verrauschten Datensätzen (sample)

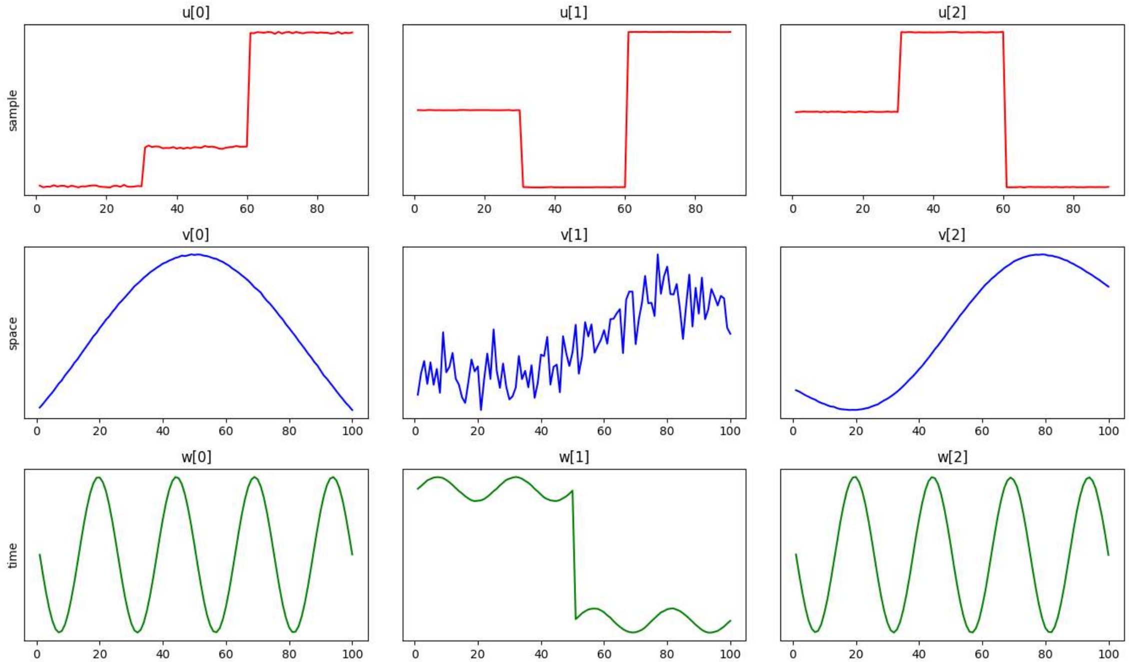


Tensorfaktorisierung

- Beispiel zur Analyse mehrdimensionaler Daten (2)

```
from mytensor import gen_tensor_three_feature as gen3f
from mytensor import plot_uvw_three_feature as plot3f
```

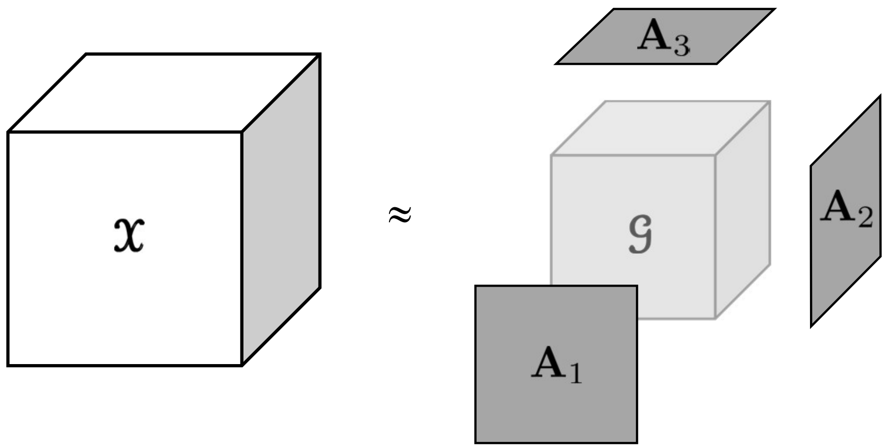
```
X = gen3f()
w_, fac = parafac(X, 3)
plot3f(fac)
```



Tensorfaktorisierung

- weitere Faktorisierung: Tucker
  - Idee: Zerlegung eines *N*-Mode-Tensors $\mathcal{X}$ in einen Kerntensor $\mathcal{G}$ plus Faktormatrizen $\mathbf{A}_i$

- Beispiel: 3-Mode-Tensor



$\Rightarrow$ allgemeine Zerlegung eines *N*-Mode-Tensors $\mathcal{X} \approx [\![\, \mathcal{G}\,; \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N]\!]$

Tensorfaktorisierung

- Tucker-Zerlegung $\mathcal{X} \rightarrow [\![\mathcal{G}; \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N]\!]$

```
from tensorly.decomposition import tucker
G, factors = tucker(X, (rank_1, rank_2, ..., rank_N))
```

- Tensor rekonstruieren $[\![\mathcal{G}; \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N]\!] \rightarrow \widehat{\mathcal{X}}$

```
X_rec = tl.tucker_to_tensor((G, factors))
tl.norm(X - X_rec)
```

- und zum Abschluss nochmal ein bisschen 'Gesichtserkennung'… ☺

Tensorfaktorisierung

- Gesichtserkennung: Olivetti-Datensatz (Paket **`sklearn`**) – Teil 2

```
from sklearn.datasets import fetch_olivetti_faces
faces = fetch_olivetti_faces()
```

- Tucker-Zerlegung der Bilder mit Kerntensor $32 \times 32 \times 32$

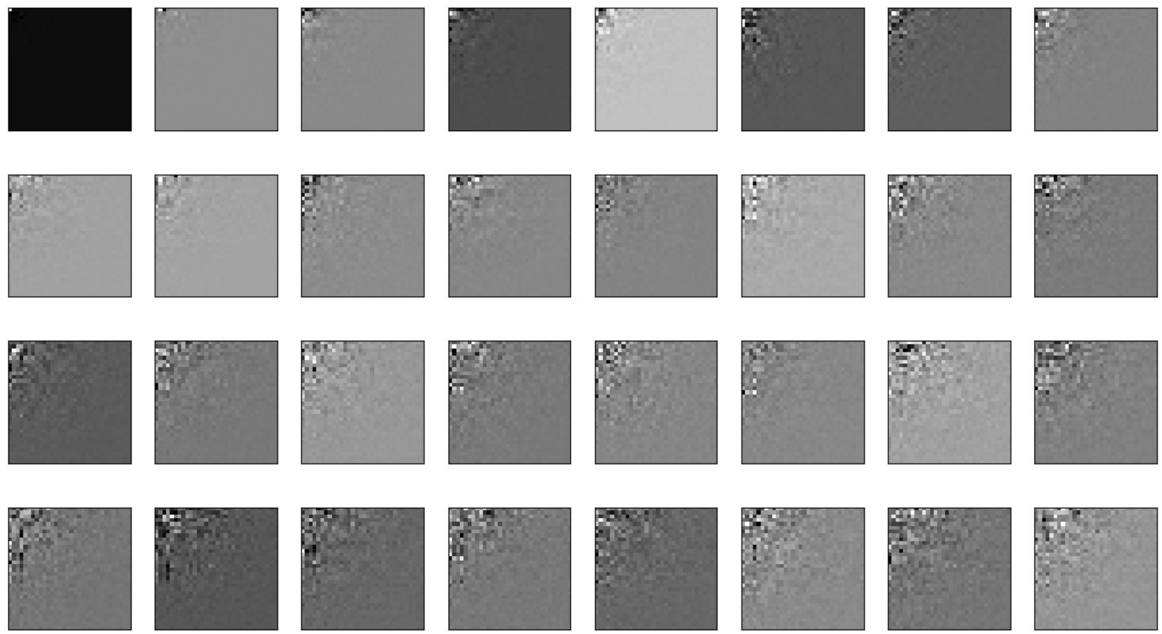
```
G, fac = tucker(faces.data, (32,32,32))
```

- kurze Rechnung
  - Originaldatensatz: $40 \times 10 \times 64 \times 64 = 1'638'400$ Datenpunkte ($\sim$ 6.25 MByte)
  - komprimierter Datensatz: $32 \times 32 \times 32 = 32'768$ Datenpunkte ($\sim$ 128 KByte)
  - d.h. Kompressionsrate von 1:50 $\rightarrow$ entspricht 2% des Originalaufwands
  - Frage: ist die verbliebene (latente) Information brauchbar…?

Tensorfaktorisierung

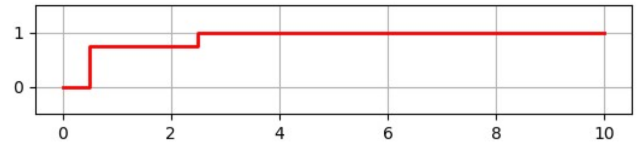
- Gesichtserkennung: Olivetti-Datensatz (Paket **`sklearn`**) – Teil 2
  - was steckt eigentlich im Kerntensor…?

```
G, fac = tucker(faces.images, (32,32,32))
plot_faces(G, 4, 8, rnd=False)
```

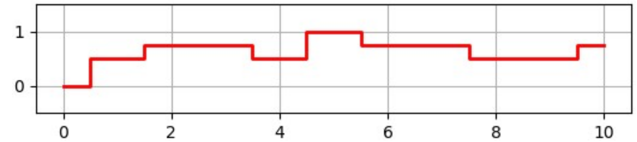


Tensorfaktorisierung

- Gesichtserkennung: Olivetti-Datensatz (Paket **`sklearn`**) – Teil 2
  - Vorgehen
    - Tucker-Zerlegung des Bilddatensatzes **`faces.data`** ($\rightarrow$ Kompression auf 2%)
    - Rekonstruktion der komprimierten Daten als **`x_rec`**
    - Rest analog zur Aufgabe LDA (Teil 4)
      - Aufteilung von **`x_rec`** und **`y_true`** in Training- und Testdaten (4 Bilder $\rightarrow$ 1%)
      - **`fit(`**$n\_components = 3$**`)`** und **`transform()`** der Trainings-/Testdaten mittels LDA
      - Anlernen (**`fit()`**) eines *k-NN* auf den projizierten Trainingsdaten für $k \in [1,10]$
      - Klassifikation (**`predict()`**) und Ausgabe der Treffsicherheit (**`accuracy_score()`**)



$n\_components = 3$ (Tucker + LDA)



$n\_components = 3$ (LDA)

Tensorfaktorisierung

- Gesichtserkennung: Olivetti-Datensatz (Paket **`sklearn`**) – Teil 2
  - Rücktransformation der komprimierten & projizierten Trainingsdaten (*Fisherfaces*)

Zusammenfassung

- was haben wir gesehen / gemacht…
  - Bestimmung der Tendenz eines Datensatzes (Lineare Regression)
  - Klassifikation
    - *k*-Nächste-Nachbarn (überwachtes Lernen)
    - *k*-Mitten (unüberwachtes Lernen)
  - Dimensionsreduktion
    - Hauptkomponentenanalyse (PCA)
    - Lineare Diskriminanzanalyse (LDA)
  - Tensorfaktorisierung
    - Datenanalyse
    - Kompression

Quelle: ampro.fr

Danke!