Rechnen mit Zahlenpaketen (Tensor-Operationen)

Kapitel online

Zahlenpakete addieren, multiplizieren und umformen — Addition, Multiplikation, Reshape, Flatten sind die Basis für alles Weitere.

Grundlagen · Kapitel 10 von 50 · ca. 20 Min. · 100 XP

Einfach gesagt: Zahlenpakete werden umsortiert und miteinander verrechnet — genau diese Handgriffe stecken hinter jedem Rechenschritt einer KI.

Im Tensor-Kapitel hast du gesehen: Alles wird zu Zahlenpaketen. Aber ein Paket, das nur herumliegt, nützt niemandem — die eigentliche KI-Arbeit ist, Pakete **umzuformen** und **miteinander zu verrechnen**, millionenfach pro Sekunde. Genau diese Handgriffe lernst du jetzt. Und es sind erstaunlich wenige.

Denk an eine Tabelle in Excel: Dieselben Zahlen kannst du als lange Spalte, als 2×3-Block oder quer anordnen — der *Inhalt* bleibt gleich, nur die **Form** ändert sich. Und manche Rechnungen klappen nur, wenn die Form stimmt. Bei Tensoren ist es genauso: erst formen (reshape, flatten, transpose), dann rechnen (add, mul, matmul).

Praxis-Relevanz: Jedes neuronale Netz nutzt diese Operationen ständig! Ein Bild (28×28) wird geflattet zu [784], dann multipliziert mit Gewichten [784, 128] = [128]. Das ist Training - nur Tensor-Operationen!

HerrKI-Tipp: matmul = Matrix-Multiplikation, multiply = Element-weise! Das ist DER Unterschied, den du kennen musst! [2,3] × [3,4] = [2,4] mit matmul!

Was du hier lernst

- Wie du Tensoren **umformst** (reshape, flatten, transpose)
- Wie Tensoren **miteinander verrechnet** werden (add, multiply, matmul)
- Warum **flatten** für neuronale Netze wichtig ist
- Was ist der Unterschied zwischen **multiply** und **matmul**?

Für den Unterricht

Einstieg (4 Min): Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

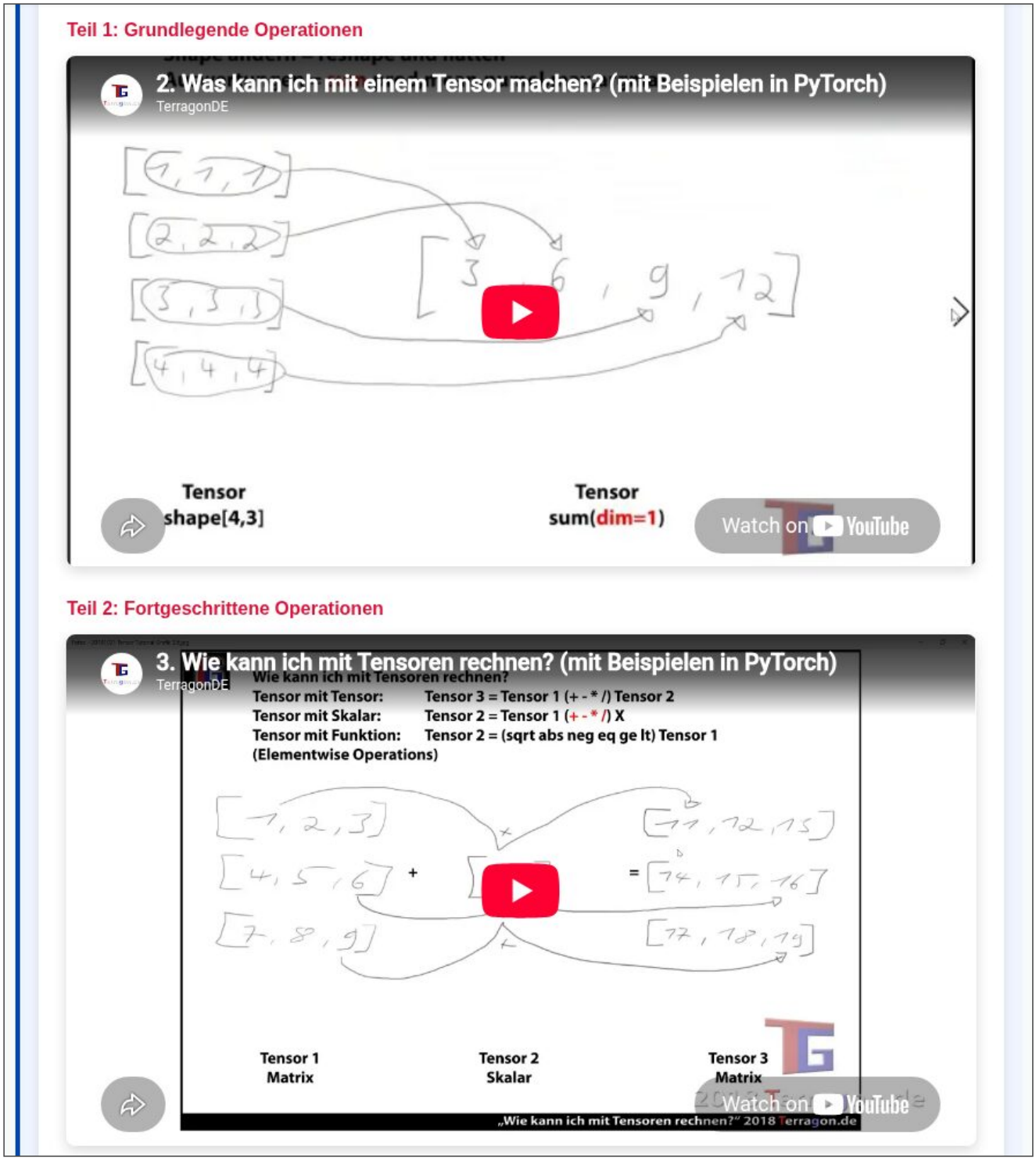
Erkunden am Gerät (12 Min): die interaktiven Module auf herrki.de/060_tensor_operationen ausprobieren.

Sichern mit Quiz (4 Min): „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

Video-Erklärungen von HerrKI

Video ansehen: herrki.de/060_tensor_operationen#abschnitt-2

Direkt zum Modul



Video auf der Kapitelseite – am Gerät ansehen

Teil 1: Tensor-Transformationen

Tensoren können ihre Form ändern, ohne dass sich die Daten ändern. Das ist wichtig, um sie für verschiedene Operationen vorzubereiten!

1.1 Reshape - Form ändern

Mit `reshape` kannst du die Form eines Tensors ändern, solange die Gesamtzahl der Elemente gleich bleibt.

Original: Vektor mit 6 Elementen

1	2	3	4	5	6

[6]

↓ reshape([2, 3])

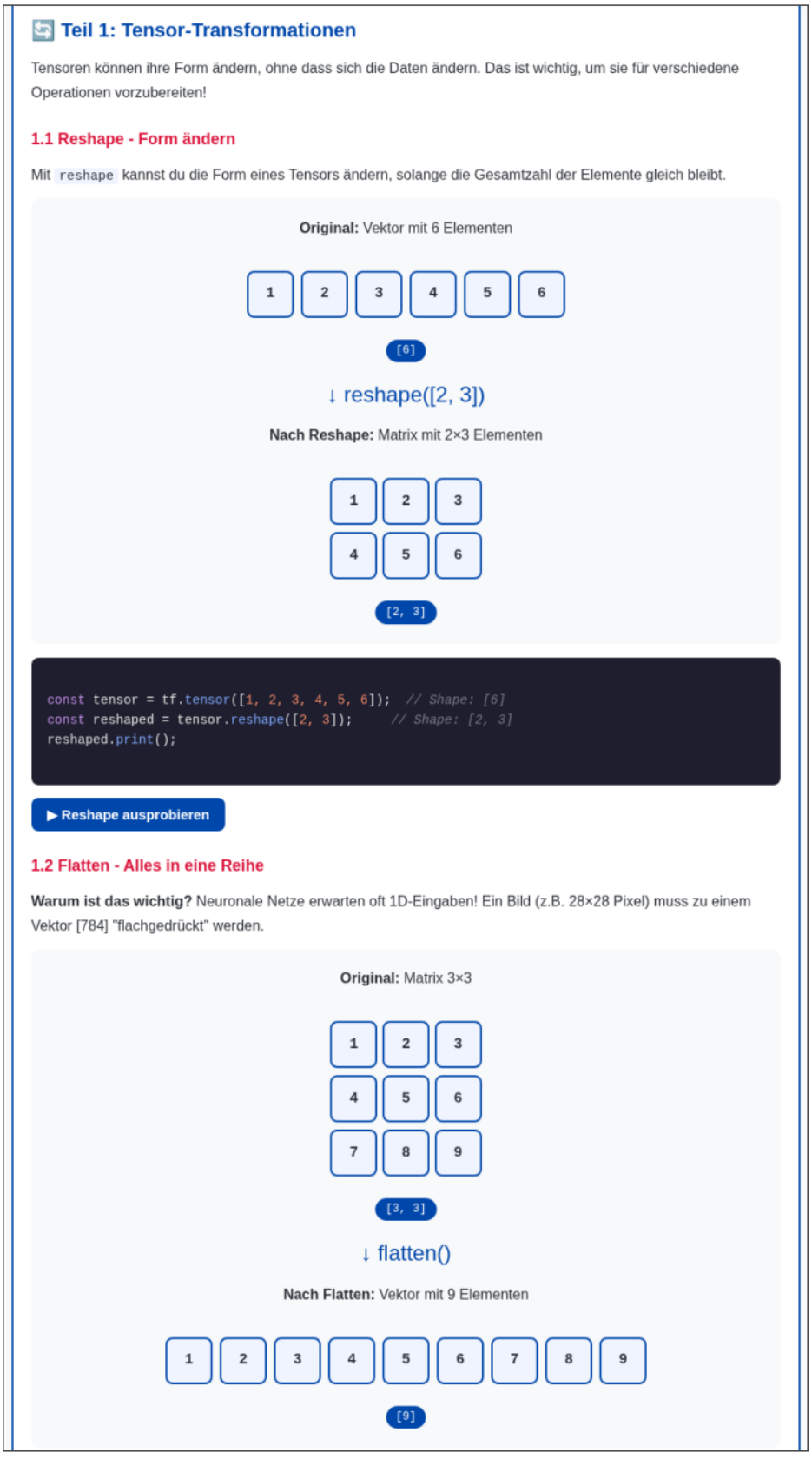
Nach Reshape: Matrix mit 2×3 Elementen

1	2
4	5

[2, 3]

Am Gerät ausprobieren: herrki.de/060_tensor_operationen#abschnitt-3

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

```
const tensor = tf.tensor([1, 2, 3, 4, 5, 6]); // Shape: [6]
const reshaped = tensor.reshape([2, 3]); // Shape: [2, 3]
reshaped.print();
```

1.2 Flatten - Alles in eine Reihe

Warum ist das wichtig? Neuronale Netze erwarten oft 1D-Eingaben! Ein Bild (z.B. 28×28 Pixel) muss zu einem Vektor [784] "flachgedrückt" werden.

Original: Matrix 3×3

1	2
4	5
7	8

[3, 3]

↓ flatten()

Nach Flatten: Vektor mit 9 Elementen

1	2	3	4	5	6	7	8	9

[9]

Flatten für Neuronale Netze

Beispiel: Ein 28×28 Graustufenbild hat **Shape [28, 28]** mit 784 Pixeln.

Ein neuronales Netz mit 784 Eingabe-Neuronen erwartet einen Vektor mit **Shape [784]**.

→ Du musst `flatten()` verwenden: **[28, 28] → [784]**

```
const image = tf.ones([28, 28]); // Bild 28×28
const flattened = image.flatten(); // Vektor [784]
console.log(flattened.shape); // [784]
```

1.3 Transpose - Zeilen ↔ Spalten tauschen

Mit `transpose` kannst du die Dimensionen eines Tensors vertauschen.

Original: Matrix 2×3

1	2
4	5

[2, 3]

↓ transpose()

Nach Transpose: Matrix 3×2

1
2
3

[3, 2]

```
const matrix = tf.tensor2d([[1, 2, 3], [4, 5, 6]]); // [2, 3]
const transposed = matrix.transpose(); // [3, 2]
transposed.print();
```

Runter zum Tensor: Rechnen mit Tensoren

Jetzt wird es spannend! Wie können wir Tensoren miteinander verrechnen?

> **Moment — warum sollte man mit Tensoren überhaupt rechnen wollen?**
> Erinnerst du dich an „Text & LLMs“? Dort hast du gesehen, dass Wörter zu **Vektoren** werden — und dass Paare wie *König* ↔ *Königin* oder *Paris* ↔ *Frankreich* im Zahlenraum ähnliche Abstände haben. Das heißt: **Bedeutung verschieben = Vektoren verrechnen.** Genau die Operationen, die du gleich lernst, sind das Handwerkszeug dafür — und dieselben Operationen treiben jedes neuronale Netz an. Kein Rechnen um des Rechnens willen: jede Zeile hier hat einen Job in der KI.

2.1 Addition - Element für Element

Tensoren mit **gleicher Shape** können addiert werden. Jedes Element wird mit dem entsprechenden Element addiert.

1	2	3

[3]

+

10	20	30

[3]

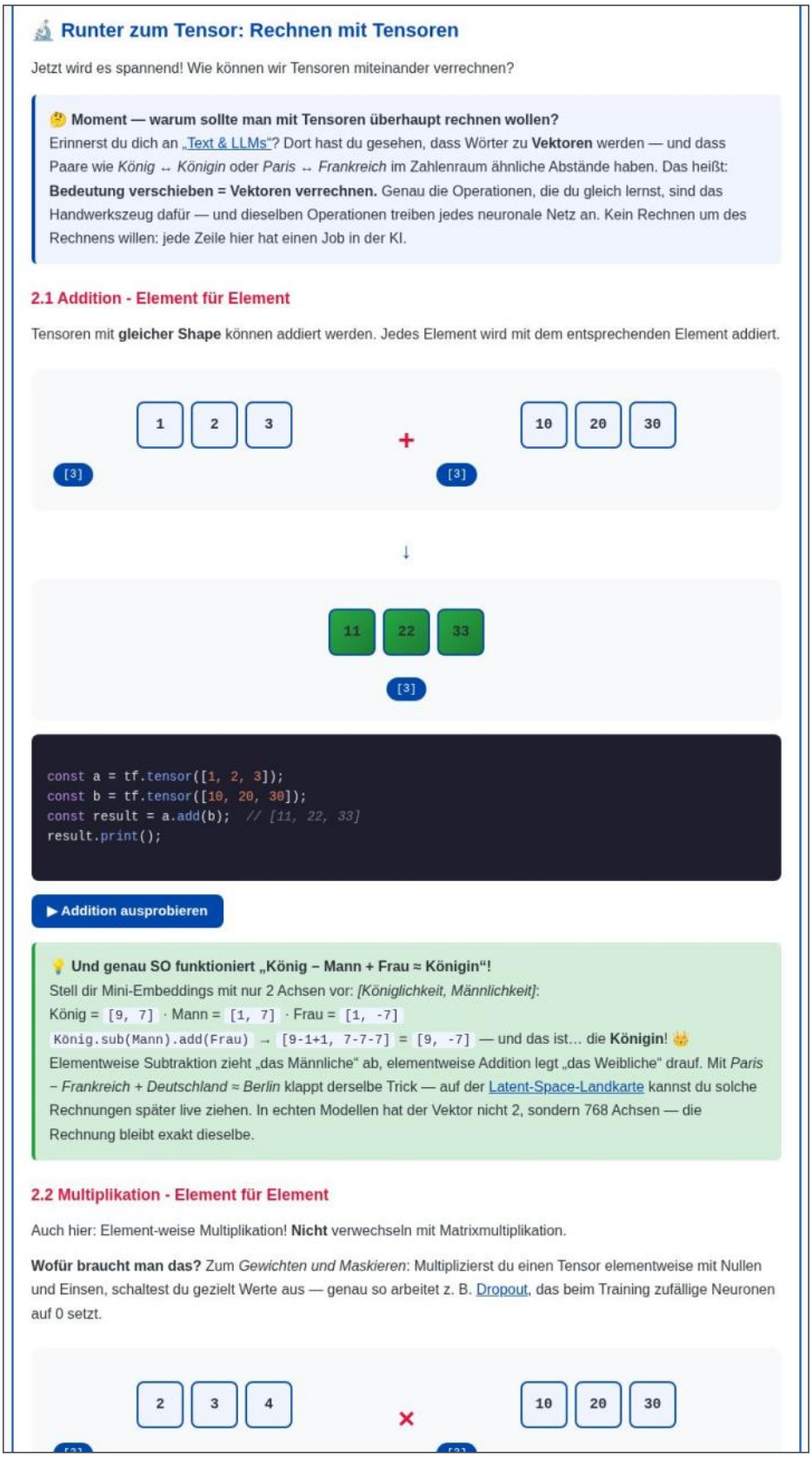
↓

11	22	33

[3]

Am Gerät ausprobieren: herrki.de/060_tensor_operationen#abschnitt-4

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

```
const a = tf.tensor([1, 2, 3]);
const b = tf.tensor([10, 20, 30]);
const result = a.add(b); // [11, 22, 33]
result.print();
```

Und genau SO funktioniert „König − Mann + Frau ≈ Königin“!
Stell dir Mini-Embeddings mit nur 2 Achsen vor: *[Königlichkeit, Männlichkeit]*:
König = `[9, 7]` · Mann = `[1, 7]` · Frau = `[1, -7]`
`König.sub(Mann).add(Frau)` → `[9-1+1, 7-7-7]` = `[9, -7]` — und das ist… die **Königin**!
Elementweise Subtraktion zieht „das Männliche“ ab, elementweise Addition legt „das Weibliche“ drauf. Mit *Paris − Frankreich + Deutschland ≈ Berlin* klappt derselbe Trick — auf der Latent-Space-Landkarte kannst du solche Rechnungen später live ziehen. In echten Modellen hat der Vektor nicht 2, sondern 768 Achsen — die Rechnung bleibt exakt dieselbe.

2.2 Multiplikation - Element für Element

Auch hier: Element-weise Multiplikation! **Nicht** verwechseln mit Matrixmultiplikation.

Wofür braucht man das? Zum *Gewichten und Maskieren*: Multiplizierst du einen Tensor elementweise mit Nullen und Einsen, schaltest du gezielt Werte aus — genau so arbeitet z. B. Dropout, das beim Training zufällige Neuronen auf 0 setzt.

2	3	4

[3]

×

10	20	30

[3]

↓

20	60	120

[3]

```
const a = tf.tensor([2, 3, 4]);
const b = tf.tensor([10, 20, 30]);
const result = a.mul(b); // [20, 60, 120]
result.print();
```

2.3 Matrixmultiplikation (matmul)

Das ist die wichtigste Operation für neuronale Netze! Sie kombiniert Zeilen und Spalten auf mathematische Weise.

Warum ausgerechnet matmul? Weil *jede Schicht eines neuronalen Netzes genau EINE Matrixmultiplikation ist:* dein Eingabe-Vektor (z. B. das flachgedrückte Bild `[784]` aus Teil 1) trifft auf die **Gewichts-Matrix** der Schicht `[784, 128]` — heraus kommt `[128]`: die Antworten von 128 Neuronen *auf einen Schlag*. Jede **Spalte** der Gewichts-Matrix ist dabei ein komplettes Neuron aus „Was ist ein Neuronales Netz?“ (Gewichte × Eingaben, aufsummiert). Wenn du in Kapitel „Lineare Regression“ gleich dein erstes Netz trainierst, dreht das Training an genau diesen Matrix-Zahlen.

Regel für Matrixmultiplikation

Matrix A mit Shape **[m, n]** kann mit Matrix B mit Shape **[n, p]** multipliziert werden.

Das Ergebnis hat Shape **[m, p]**.

Beispiel: [2, 3] × [3, 2] = [2, 2]

Matrix A: [2, 3]

1	2
4	5

matmul

Matrix B: [3, 2]

1
3
5

↓

Ergebnis: [2, 2]

22
49

```
const A = tf.tensor2d([[1, 2, 3], [4, 5, 6]]); // [2, 3]
const B = tf.tensor2d([[1, 2], [3, 4], [5, 6]]); // [3, 2]
```

```
const result = tf.matMul(A, B); // [2, 2]
result.print();
```

Warum ist matmul so wichtig?

In neuronalen Netzen werden **Inputs × Weights** mit matmul berechnet!

Beispiel: Ein Batch von 32 Bildern [32, 784] × Gewichte [784, 128] = Output [32, 128]

Merk dir dieses Bild: Wenn du in Kapitel „Lineare Regression“ zum ersten Mal trainierst, ist jede einzelne Vorhersage genau so ein matmul.

Zurück zur Anwendung

Erinnerst du dich an die Excel-Tabelle vom Anfang? Genau das hast du hier die ganze Zeit gemacht: dieselben Zahlen mal als lange Spalte, mal als 2×3-Block, mal quer — `reshape`, `flatten` und `transpose` sind nichts anderes als Umsortieren, ohne dass ein einziger Wert verloren geht. Und dann kommt das Rechnen dazu: `add` und `mul` gehen Zelle für Zelle durch, `matmul` verknüpft ganze Zeilen mit ganzen Spalten. Das ist der Unterschied, an dem in der Praxis die meisten Fehlermeldungen hängen — und deshalb die wichtigste Faustregel des Kapitels: erst die Form prüfen, dann rechnen.

Ein 28×28-Bild wird zu [784] flachgedrückt, mit Gewichten [784, 128] multipliziert — und heraus kommt [128]. Das ist keine Fingerübung, das ist wörtlich eine Schicht eines neuronalen Netzes.

Jetzt hast du das komplette Rechen-Werkzeug in der Hand. Als Nächstes setzen wir es zusammen: „Was ist ein Neuronales Netz?“ — dort entdeckst du, dass ein Neuron nichts anderes tut als genau diese Operationen.

Zum Weiterlesen

- **Die Mathematik dahinter, sauber aufgeschrieben:** Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, Kapitel 2 „Linear Algebra“ — deeplearningbook.org (deeplearningbook.org/contents/linear_algebra.html) — dort stehen Transpose und Matrixmultiplikation als Formeln, die du jetzt schon anfassen kannst.
- **Zum Nachschlagen beim Selberbauen:** TensorFlow.js, API-Referenz — js.tensorflow.org/api/latest — jede Operation aus diesem Kapitel mit allen Parametern und Beispielen.
- **Wenn die Shapes doch mal nicht passen:** NumPy, *Broadcasting* — numpy.org (numpy.org/doc/stable/user/basics.broadcasting.html) — die Regeln, nach denen ein `[2, 3]` und ein `[3]` trotzdem zusammen rechnen dürfen.
- **Auf HerrKI:** 010 — Was sind Tensoren? (woher die Zahlenpakete kommen) und 120 — Lineare Regression (wo dein erstes matmul wirklich etwas lernt).

Kurzer Check – mit Lösung

Lösungen für Lehrkräfte – für Lernende die Online-Version nutzen.

Drei schnelle Fragen, dann bist du durch — klick einfach deine Antwort an:

Frage 1: Ändert reshape die Werte eines Tensors?

- Ja, die Werte werden neu berechnet
- ✔ **Nein — gleicher Inhalt, nur andere Anordnung (wie Excel-Zellen umsortieren)**
- Nur bei Matrizen, nicht bei Vektoren

Richtig: reshape packt dieselben Zahlen nur in eine neue Form. Aus [2, 3] wird [3, 2] oder [6] — kein Wert geht verloren, keiner ändert sich.

Frage 2: Ein Tensor mit Shape [2, 3] wird geflattet. Welche Shape kommt heraus?

- ✔ **[6]**
- [3, 2]
- [2, 3, 1]

Genau: flatten legt alle 2·3 = 6 Werte in eine lange Reihe. Das braucht man ständig — etwa um ein Bild in eine Netz-Eingabeschicht zu füttern.

Frage 3: Warum ist die Matrix-Multiplikation die wichtigste Operation der KI?

- ✔ **Sie ist der Rechenkern jeder Netz-Schicht: Eingaben × Gewichte, tausendfach parallel**
- Sie macht Tensoren kleiner und spart Speicher
- Sie ist nur für Mathematiker interessant

Richtig! Wenn im nächsten Kapitel dein erstes Netz trainiert, passiert innen genau das: matmul aus Eingabe-Tensor und Gewichts-Tensor — Millionen Mal pro Sekunde.

Das nimmst du mit

- **Form und Inhalt sind zwei Paar Schuhe.** `reshape` und `transpose` ordnen nur um — kein Wert ändert sich dabei.
- **`flatten` ist die Rampe ins Netz.** Alles, was mehrdimensional ist, wird vor der Eingabeschicht zu einer langen Reihe.
- **`mul` ≠ `matMul`.** Das eine rechnet Zelle gegen Zelle, das andere Zeile gegen Spalte — der häufigste Anfängerfehler überhaupt.
- **Die matmul-Regel:** `[a, b] × [b, c] = [a, c]`. Die inneren Zahlen müssen gleich sein, die äußeren stehen im Ergebnis.
- **Mehr braucht es nicht.** Jede Schicht jedes neuronalen Netzes ist Eingabe-Tensor × Gewichts-Tensor — millionenfach pro Sekunde, aber immer diese eine Operation.

Transformationen

- `reshape` - Form ändern
- `flatten` - Zu Vektor machen
- `transpose` - Dimensionen tauschen

Operationen

- `add` - Element-weise Addition
- `mul` - Element-weise Multiplikation
- `matMul` - Matrixmultiplikation

Für Neuronale Netze

- `flatten` - Für Input Layer
- `matMul` - Für Gewichte
- Shape beachten!

Das Wichtigste in einem Satz:

Tensoren müssen die richtige Form haben, damit Operationen funktionieren - und `flatten` ist der Schlüssel, um komplexe Daten in neuronale Netze zu bringen!

So zitierst du dieses Kapitel

HerrKI: Rechnen mit Zahlenpaketen (Tensor-Operationen). herrki.de/060_tensor_operationen, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.