# Was ist ein Neuronales Netz?

Neuronen, Gewichte, Bias und Aktivierungsfunktionen — interaktiv erklärt.

**Praxis · Kapitel 11 von 50 · ca. 15 Min. · 150 XP**

Kapitel online

**Einfach gesagt:** Ein Neuron ist kein Gehirn, sondern ein Taschenrechner: es multipliziert deine Eingabe mit einem Gewicht und addiert eine Grundzahl.

Du weißt jetzt, wie Daten zu Tensoren werden und wie man damit rechnet. Aber WER rechnet da eigentlich? Zeit, das berühmteste Bauteil der KI kennenzulernen — und zu entdecken, dass ein „Neuron“ weniger Gehirn und mehr Taschenrechner ist: **eine Multiplikation, eine Addition — fertig**.

Heute schaust du ihm live dabei zu: Du drehst selbst an **Gewicht** und **Bias** und siehst sofort, was mit der Zahl passiert, die hindurchfließt. Wenn das einmal sitzt, ist der Rest des Kurses nur noch Zusammenbauen.

**Keine Angst vor Mathe!** Ich zeige dir alles visuell und interaktiv. Du siehst, wie Zahlen durch das Netz fließen, wie Gewichte angepasst werden und wie am Ende eine Vorhersage rauskommt. Learning by Seeing!

**HerrKI-Tipp: Neuron = Eingabe × Gewicht + Bias → Aktivierung**. Das ist die Grundformel! Alles andere ist nur mehr davon.

**Was du hier lernst**

- Was sind **Neuronen**? (Eingabe × Gewicht + Bias)
- Was sind **Gewichte** und warum sind sie wichtig?
- Was macht eine **Aktivierungsfunktion**? (ReLU, Sigmoid)
- Wie werden Neuronen zu **Layern** verbunden?

**Für den Unterricht**

**Einstieg (3 Min):** Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

**Erkunden am Gerät (9 Min):** die interaktiven Module auf herrki.de/110_neuronale_netze ausprobieren.

**Sichern mit Quiz (3 Min):** „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

# Runter zum Tensor: Tensoren treffen auf Neuronen

Du hast in den vorherigen Modulen gelernt, dass **alle Daten als Tensoren** dargestellt werden - ob Bilder, Text oder Tabellen. Jetzt lernst du, wie **Neuronen diese Tensoren verarbeiten**!

### Die Verbindung: Tensoren & Neuronen

### Input-Tensor

| 100 |
|---|

[1]

Wassermenge: 100 ml

→

### Neuron

**× 0.05** (weight)
**+ 0.0** (bias)

Verarbeitung

→

### Output-Tensor

| 5.0 |
|---|

[1]

Wachstum: 5.0 cm

**Was passiert hier?**

1. **Input-Tensor:** Die Daten kommen als Tensor rein - hier ein Skalar [1] mit dem Wert 100
2. **Neuron verarbeitet:** Das Neuron multipliziert den Wert mit seinem **Gewicht** (0.05) und addiert den **Bias** (0.0)
3. **Output-Tensor:** Das Ergebnis ist wieder ein Tensor [1] mit dem Wert 5.0

**Die große Erkenntnis:**

**Neuronen sind Tensor-Transformatoren!**
Sie nehmen Tensoren als Eingabe, verarbeiten sie mit Gewichten und Bias, und geben transformierte Tensoren aus. In komplexen Netzen fließen diese Tensoren durch **viele Neuronen in vielen Schichten** - aber das Grundprinzip bleibt gleich!

**Mehrere Eingaben**

z.B. Wasser, Sonne, Temp

| 100 | 6 | 22 |
|---|---|---|

[3]

→ Ein Neuron kann viele Eingaben verarbeiten!

**Layer von Neuronen**

Mehrere Neuronen parallel

→ Viele Neuronen = viele Transformationen!

**Tiefe Netze**

Layer stapeln

Input → Layer 1 →
→ Layer 2 →
→ Layer 3 → Output

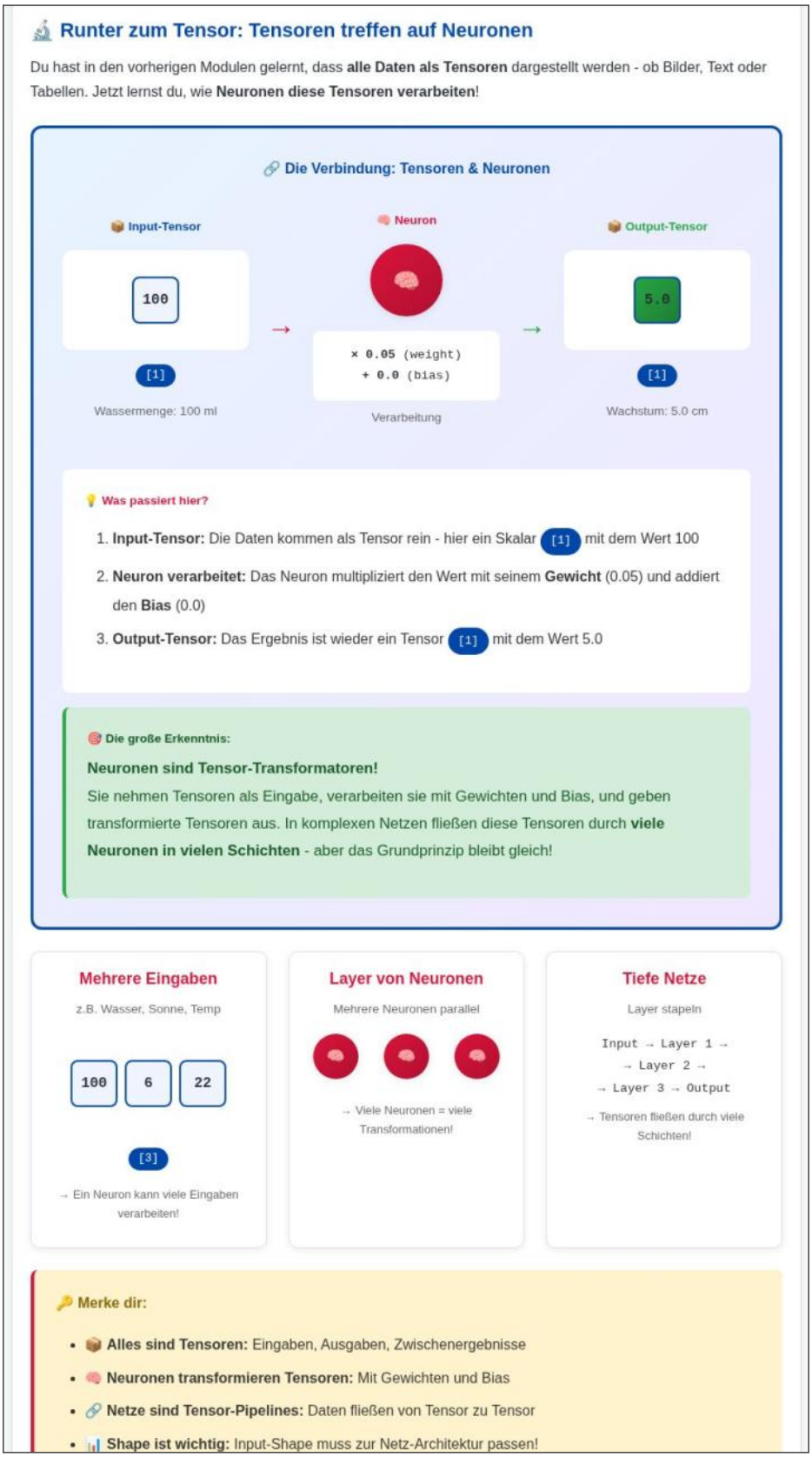
→ Tensoren fließen durch viele Schichten!

**Merke dir:**

- **Alles sind Tensoren:** Eingaben, Ausgaben, Zwischenergebnisse
- **Neuronen transformieren Tensoren:** Mit Gewichten und Bias
- **Netze sind Tensor-Pipelines:** Daten fließen von Tensor zu Tensor
- **Shape ist wichtig:** Input-Shape muss zur Netz-Architektur passen!

Am Gerät ausprobieren: herrki.de/110_neuronale_netze#abschnitt-2

Direkt zum Modul




*Interaktives Modul – am Gerät ausprobieren*

# Experiment 1: Ein einzelnes Neuron - Pflanzenwachstum vorhersagen

**Praktisches Beispiel:** Ein Neuron berechnet das Wachstum unserer roten Pflanze! Es nimmt die **Wassermenge** (ml pro Tag), multipliziert sie mit dem **Wachstumsfaktor**, und addiert das **Grundwachstum**.

100

**Wassermenge**

ml pro Tag

→

**Neuron**

Wasser × Faktor + Basis

→

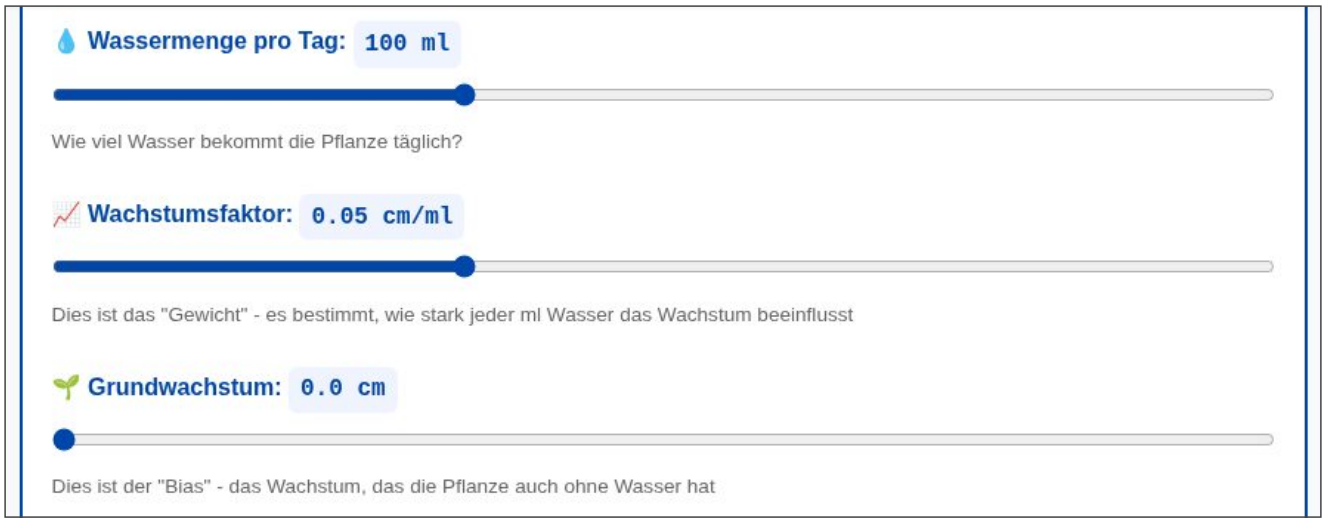
5.0

**Wachstum**

cm pro Woche

Am Gerät ausprobieren: herrki.de/110_neuronale_netze#abschnitt-3

Direkt zum Modul




*Interaktives Modul – am Gerät ausprobieren*

Wie viel Wasser bekommt die Pflanze täglich?

Dies ist das "Gewicht" - es bestimmt, wie stark jeder ml Wasser das Wachstum beeinflusst

Dies ist der "Bias" - das Wachstum, das die Pflanze auch ohne Wasser hat

**Berechnung des Pflanzenwachstums:**

```
Wachstum = 0.05 cm/ml × 100 ml + 0.0 cm = 5.0 cm
```

**Experimentiere:**

- Wie wächst die Pflanze bei nur 50ml Wasser pro Tag?
- Was passiert mit einem höheren Wachstumsfaktor (0.10)?

- Wie wirkt sich ein Grundwachstum von 2cm aus?

# Experiment 2: Aktivierungsfunktionen

Aktivierungsfunktionen entscheiden, ob ein Neuron „feuert“ oder nicht — sie machen neuronale Netze erst richtig leistungsfähig. Warum ausgerechnet diese kleinen Knicke aus Geraden beliebige Kurven machen, nimmst du später in „Warum eine Gerade nicht reicht“ auseinander.

**Linear**

y = x

Keine Veränderung

**ReLU**

y = max(0, x)

Negative Werte → 0

**Sigmoid**

y = 1/(1+e^-x)

Werte zwischen 0 und 1

**Mit Aktivierungsfunktion:**

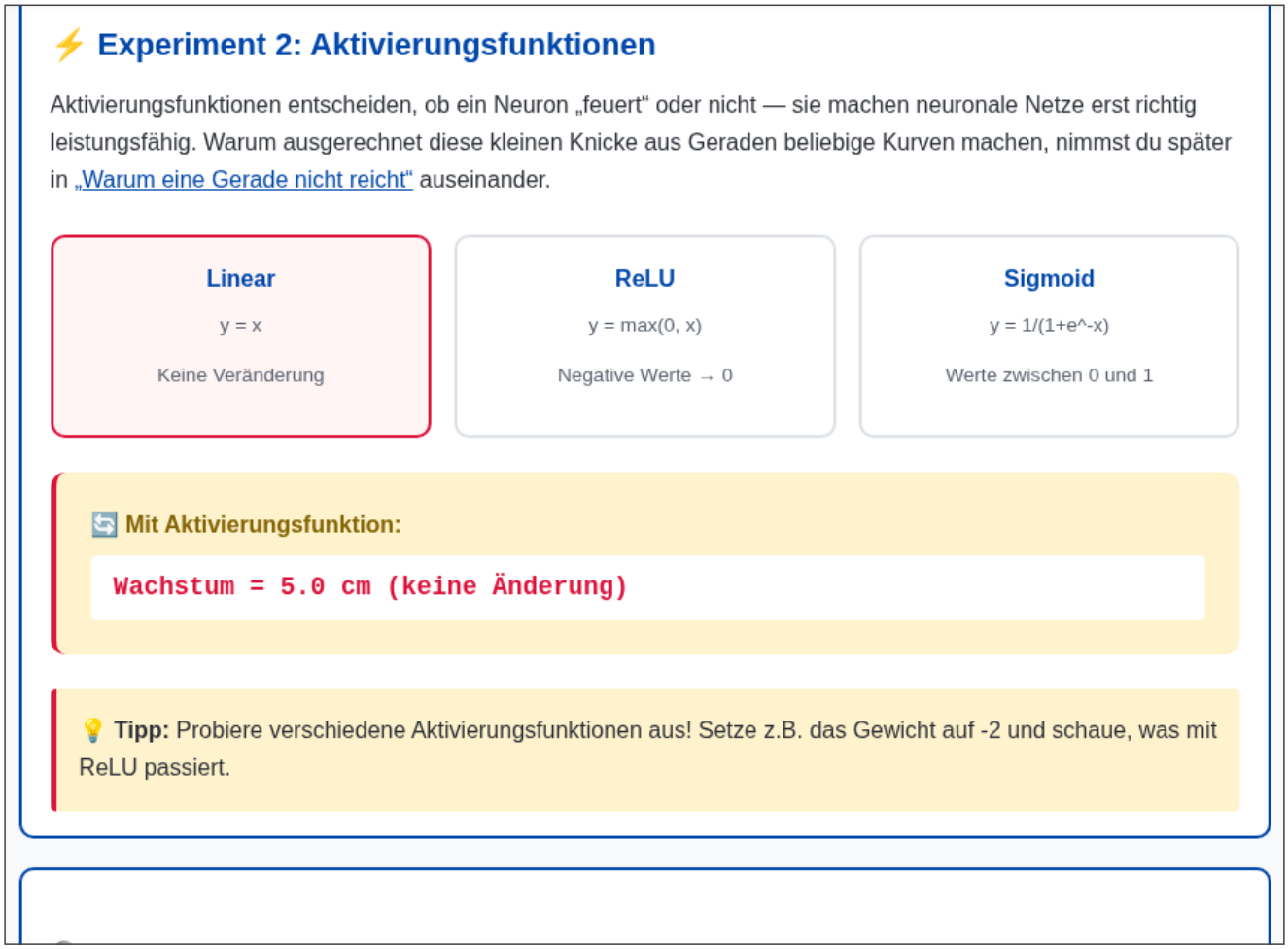
```
y = 10.0 (linear)
```

**Tipp:** Probiere verschiedene Aktivierungsfunktionen aus! Setze z.B. das Gewicht auf -2 und

schaue, was mit ReLU passiert.

Am Gerät ausprobieren: herrki.de/110_neuronale_netze#abschnitt-4

Direkt zum Modul




*Interaktives Modul – am Gerät ausprobieren*

# Ein winziges Netz, bei dem du jede Zahl siehst

Oben hattest du **ein** Neuron mit **einem** Gewicht. Jetzt bauen wir das kleinstmögliche echte **Netz**: 15 Eingänge, 3 Ausgabe-Neuronen — und wir zeigen dir *jede einzelne Zahl*, die darin steckt. Keine versteckte Schicht, kein Ausblenden: 45 Gewichte plus 3 Bias-Werte, alle sichtbar, alle nachrechenbar.

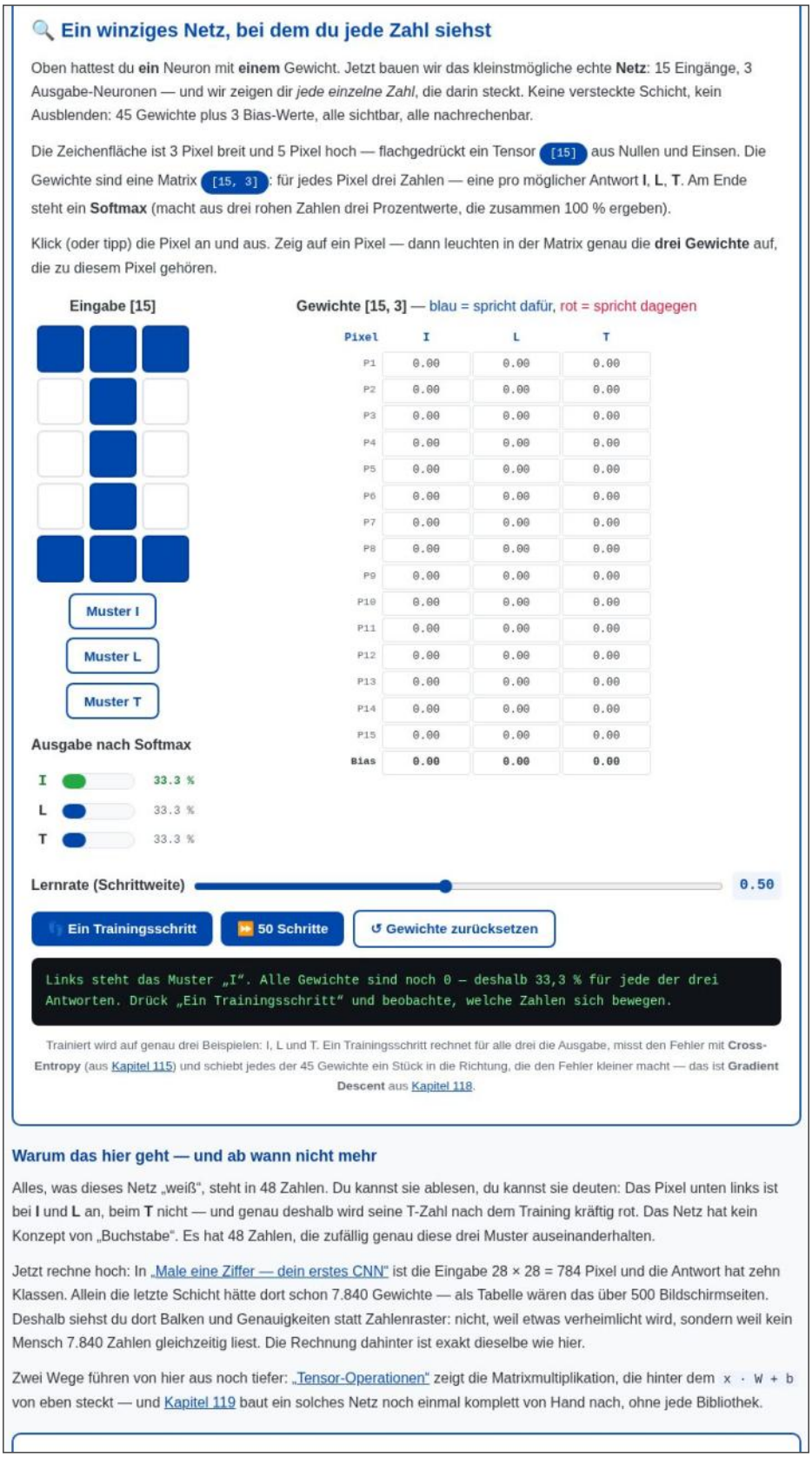
Die Zeichenfläche ist 3 Pixel breit und 5 Pixel hoch — flachgedrückt ein Tensor [15] aus Nullen und Einsen. Die Gewichte sind eine Matrix [15, 3]: für jedes Pixel drei Zahlen — eine pro möglicher Antwort **I**, **L**, **T**. Am Ende steht ein **Softmax** (macht aus drei rohen Zahlen drei Prozentwerte, die zusammen 100 % ergeben).

Klick (oder tipp) die Pixel an und aus. Zeig auf ein Pixel — dann leuchten in der Matrix genau die **drei Gewichte** auf, die zu diesem Pixel gehören.

**Eingabe [15]**

Am Gerät ausprobieren: herrki.de/110_neuronale_netze#abschnitt-5

Direkt zum Modul




*Interaktives Modul – am Gerät ausprobieren*

**Ausgabe nach Softmax**

**Gewichte [15, 3]** — blau = spricht dafür, rot = spricht dagegen

Trainiert wird auf genau drei Beispielen: I, L und T. Ein Trainingsschritt rechnet für alle drei die Ausgabe, misst den Fehler mit **Cross-Entropy** (aus Kapitel 115) und schiebt jedes der 45 Gewichte ein Stück in die Richtung, die den Fehler kleiner macht — das ist **Gradient Descent** aus Kapitel 118.

### Warum das hier geht — und ab wann nicht mehr

Alles, was dieses Netz „weiß“, steht in 48 Zahlen. Du kannst sie ablesen, du kannst sie deuten: Das Pixel unten links ist bei **I** und **L** an, beim **T** nicht — und genau deshalb wird seine T-Zahl nach dem Training kräftig rot. Das Netz hat kein Konzept von „Buchstabe“. Es hat 48 Zahlen, die zufällig genau diese drei Muster auseinanderhalten.

Jetzt rechne hoch: In „Male eine Ziffer — dein erstes CNN“ ist die Eingabe 28 × 28 = 784 Pixel und die Antwort hat zehn Klassen. Allein die letzte Schicht hätte dort schon 7.840 Gewichte — als Tabelle wären das über 500 Bildschirmseiten. Deshalb siehst du dort Balken und Genauigkeiten statt Zahlenraster: nicht, weil etwas verheimlicht wird, sondern weil kein Mensch 7.840 Zahlen gleichzeitig liest. Die Rechnung dahinter ist exakt dieselbe wie hier.

Zwei Wege führen von hier aus noch tiefer: „Tensor-Operationen“ zeigt die Matrixmultiplikation, die hinter dem `x · W + b` von eben steckt — und Kapitel 119 baut ein solches Netz noch einmal komplett von Hand nach, ohne jede Bibliothek.

# Zurück zur Anwendung: Warum Neuronale Netze?

Ein einzelnes Neuron ist wie eine einfache Formel. Aber wenn wir **viele Neuronen** in **mehreren Schichten** kombinieren, können sie komplexe Muster lernen!

**Und das Verrückte: Es ist immer DIESELBE Maschine.**

Egal welches Beispiel — unter der Haube passiert exakt das, was du gerade mit dem Regler gemacht hast, nur millionenfach:

- **Wohnungspreis schätzen:** Eingabe-Tensor [3] (Fläche, Zimmer, Lage) → Schichten → Preis. Baust du selbst in „Multiple Regression“.
- **Handschrift erkennen:** das 28×28-Pixel-Bild aus „Bilder als Tensoren“, flachgedrückt zu [784] → Schichten → „das ist eine 7“.
- **Mit dir chatten:** Token-Tensoren aus „Text & LLMs“ → sehr viele Schichten → nächstes Wort. Ja — auch ChatGPT ist am Ende dieser kleine Tensor-Transformator, nur gestapelt.
- **Spiele meistern:** Spielfeld als Tensor → Schichten → bester Zug.

Eine „Schicht“ ist dabei genau die **Matrixmultiplikation** aus „Tensor-Operationen“: viele Neuronen rechnen auf einen Schlag.

Und damit zurück zum Anfang: Die Blackbox, die wir aufgemacht haben, ist ein Taschenrechner. Als du am Regler für den Wachstumsfaktor gedreht hast, hast du ein **Gewicht** verstellt — dieselbe Zahl, die in einem echten Netz milliardenfach vorkommt. Das Grundwachstum war der **Bias**, und die Aktivierungsfunktion hat entschieden, ob aus dem Ergebnis überhaupt etwas herauskommt. Ein Neuron ist also `Eingabe × Gewicht + Bias → Aktivierung`, mehr nicht; alles Weitere im Kurs ist Stapeln und Wiederholen.

Einen Haken hat die Sache noch: Bisher hast du die Gewichte *von Hand* gedreht, bis das Ergebnis passte. Ein Netz mit Millionen Gewichten kann das niemand mehr — es muss selbst merken, wie falsch es liegt.

**Gleich** trainierst du so ein Netz selbst — vorher klären wir nur noch, woher es weiß, in welche Richtung es seine Gewichte drehen muss: in „Wie falsch ist falsch?“ wird aus „irgendwie daneben“ eine messbare Zahl.

Du verstehst jetzt, was ein Neuron ist: ein kleiner Tensor-Transformator. Als Nächstes klären wir, woher das Netz weiß, WIE es seine Gewichte verstellen muss — „Wie falsch ist falsch?“ und „Gradient Descent zum Anfassen“ — und dann trainierst du in „Lineare Regression“ dein erstes echtes Modell.

**Zum Weiterlesen**

- **Das Standardwerk zum Selberlesen:** Michael Nielsen, *Neural Networks and Deep Learning*, Kapitel 1 — neuralnetworksanddeeplearning.com (

neuralnetworksanddeeplearning.com/chap1.html) — er baut dasselbe Neuron auf, geht aber gleich weiter bis zum handgeschriebenen Ziffern-Erkenner.
- **Zum Selberdrehen:** TensorFlow Playground — playground.tensorflow.org — dort schaltest du Neuronen und Schichten dazu und siehst live, welche Muster das Netz dann lernt.
- **Zum Anschauen:** 3Blue1Brown, *Neural Networks* — 3blue1brown.com (3blue1brown.com/topics/neural-networks) — die vermutlich schönste visuelle Erklärung davon, was in den Schichten passiert.
- **Auf HerrKI:** 060 — Tensor-Operationen (die Matrixmultiplikation hinter jeder Schicht) und 145 — Warum eine Gerade nicht reicht (warum es ohne Aktivierungsfunktion nicht geht).

# 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: Was rechnet ein einzelnes Neuron?**

- Es speichert ein Stück Wissen als Text
- **✔ Gewichtete Summe der Eingaben + Bias, dann durch die Aktivierungsfunktion — weniger Gehirn, mehr Taschenrechner**
- Es vergleicht die Eingabe mit allen Trainingsdaten

*Genau: Ausgabe = Aktivierung($w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + b$). Klingt unspektakulär — aber Tausende davon in Schichten ergeben die Netze, die Gesichter erkennen.*

**Frage 2: Welche Rolle spielen die Gewichte?**

- **✔ Sie sind die lernbaren Stellschrauben — Lernen heißt: Gewichte nachjustieren, bis der Fehler sinkt**
- Sie geben an, wie schwer der Computer ist
- Sie werden einmal vom Programmierer festgelegt und bleiben fix

*Richtig — das ist der Kreislauf aus dem allerersten Kapitel: raten → Fehler messen → nachjustieren. Genau diese Gewichte dreht das Training automatisch in die richtige Richtung.*

**Frage 3: Wozu braucht ein Netz eine Aktivierungsfunktion?**

- Sie schaltet das Netz ein und aus
- **✔ Sie macht das Netz nichtlinear — ohne sie könnte es nur Geraden lernen, egal wie viele Schichten es hat**
- Sie beschleunigt die Grafikkarte

*Genau! Ohne Aktivierung kollabieren beliebig viele Schichten zu einer einzigen Geraden. Warum das so ist — und wie ReLU Kurven aus Knicken baut — zeigt dir das Kapitel „Warum eine Gerade nicht reicht“.*

# Das nimmst du mit

- **Ein Neuron ist eine Formel, kein Gehirn:** `Eingabe × Gewicht + Bias → Aktivierung`. Eine Multiplikation, eine Addition, fertig.
- **Das Gewicht ist die Stellschraube.** Es entscheidet, wie stark eine Eingabe zählt — und genau diese Zahlen justiert das Training später automatisch.
- **Der Bias verschiebt das Ganze.** Er ist der Grundwert, der auch dann herauskommt, wenn die Eingabe null ist.
- **Ohne Aktivierungsfunktion bleibt alles eine Gerade.** Beliebig viele lineare Schichten ergeben zusammen wieder nur eine einzige Gerade — erst der Knick (ReLU, Sigmoid) macht Kurven möglich.
- **Eine Schicht ist eine Matrixmultiplikation.** Viele Neuronen auf einen Schlag — genau das `matMul` aus dem Tensor-Kapitel, nur millionenfach.

### So zitierst du dieses Kapitel

HerrKI: Was ist ein Neuronales Netz?. herrki.de/110_neuronale_netze, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.