Die erste Vorhersage: eine Gerade durch die Punkte (Lineare Regression)

Kapitel online

Dein erstes echtes Training mit TensorFlow.js: eine Gerade so durch echte Punkte legen, dass sie am besten passt.

Praxis · Kapitel 15 von 50 · ca. 25 Min. · 150 XP

Einfach gesagt: Aus zehn Messpunkten findet die KI selbst die Gerade, die am besten passt — indem sie ihren Fehler immer wieder verkleinert.

Erinnerst du dich an das Neuron aus dem Neuronen-Kapitel? Da hast DU an Gewicht und Bias gedreht, bis das Pflanzen-Wachstum stimmte. Heute drehen wir den Spieß um: **Das Netz dreht selbst** — und du schaust zu, wie es aus 10 Messungen die Gieß-Regel unserer Pflanze herausfindet.

Das Modell ist absichtlich das einfachste der Welt: eine Gerade ($y = m \cdot x + b$), ein einziges Neuron. Genau deshalb kannst du hier ALLES beobachten, was auch in GPT & Co. passiert — Loss, Optimizer, Epochen — nur eben mit zwei Zahlen statt Milliarden.

Die Ausgangslage: Du hast 10 Einträge aus deinem Gieß-Tagebuch — Wassermenge rein, Wachstum raus. Die Regel dahinter ist „verloren gegangen“. Stell dir vor, du druckst die 10 Punkte auf Papier und legst ein Lineal so hindurch, dass es allen Punkten möglichst nahe kommt: *genau das* macht das Netz gleich — nur dass es das Lineal nicht mit der Hand schiebt, sondern mit dem Bergab-Schritt aus Kapitel 118. Zwei Stellschrauben hat es dafür: **m** (wie steil das Lineal liegt) und **b** (wo es startet — das „Grundwachstum“ ganz ohne Wasser).

HerrKI-Tipp: Training ist exakt der Kreislauf aus „Was heißt Lernen?“: **raten → Fehler messen (Loss) → nachjustieren (Gradient Descent) → wiederholen**. Gleich siehst du ihn zum ersten Mal in echt.

Was du hier lernst

- Ein **Modell definieren** — „Sequential“ + „Dense“ klingt wichtig, heißt aber nur: *Schichten hintereinander, hier genau eine mit einem Neuron*
- Die **Loss-Funktion** — der Fehler-Zollstock aus Kapitel 115 (hier: mittlerer quadratischer Fehler)
- Der **Optimizer** — der Bergab-Läufer aus Kapitel 118, der die Stellschrauben dreht
- **Epochen** — wie oft das Netz den kompletten Datensatz durcharbeitet (hier: 100 Runden)

Für den Unterricht

Einstieg (5 Min): Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

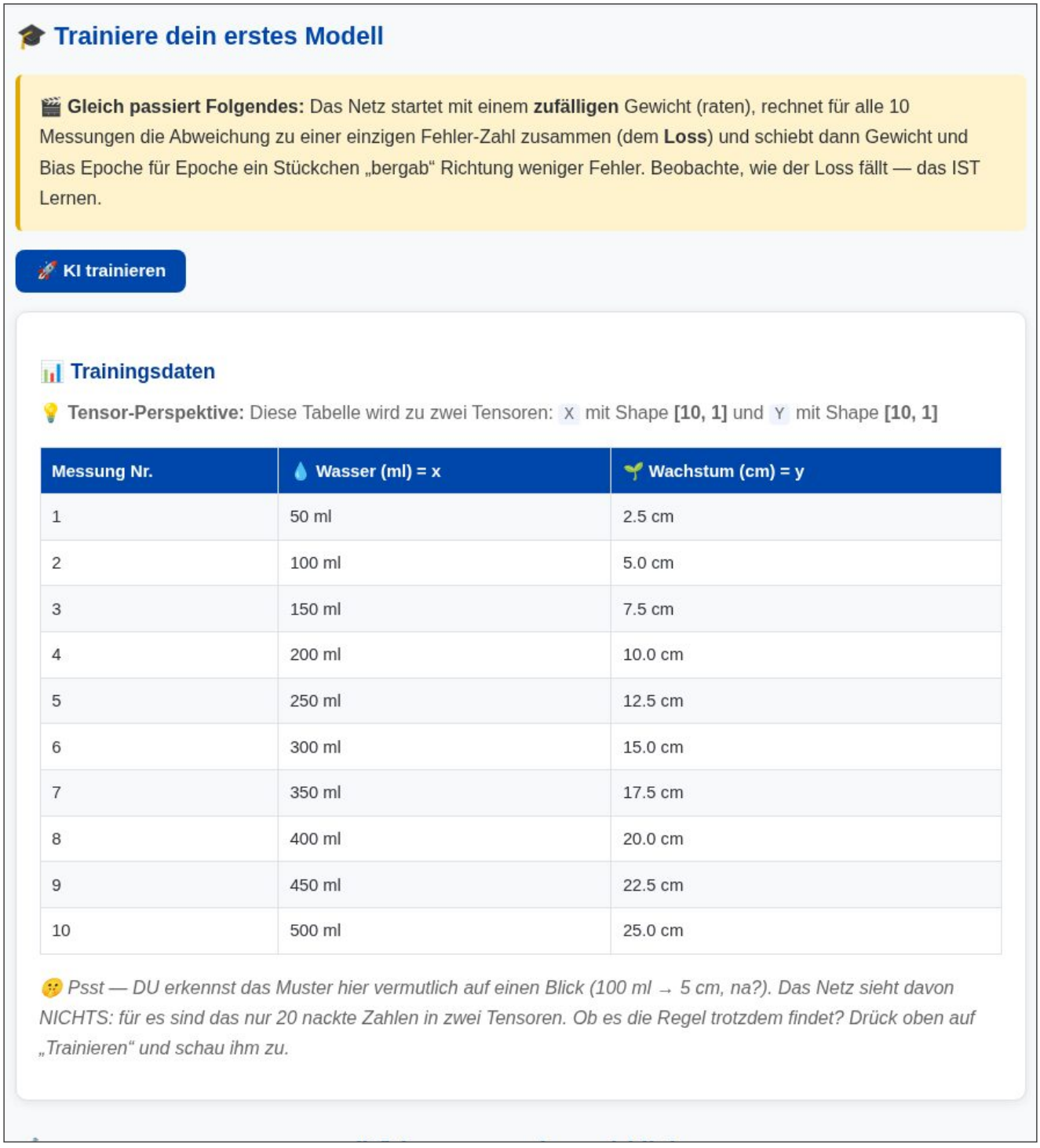
Erkunden am Gerät (15 Min): die interaktiven Module auf herrki.de/120_lineare_regression ausprobieren.

Sichern mit Quiz (5 Min): „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

Trainiere dein erstes Modell

Am Gerät ausprobieren: herrki.de/120_lineare_regression#abschnitt-2

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Gleich passiert Folgendes: Das Netz startet mit einem **zufälligen** Gewicht (raten), rechnet für alle 10 Messungen die Abweichung zu einer einzigen Fehler-Zahl zusammen

(dem **Loss**) und schiebt dann Gewicht und Bias Epoche für Epoche ein Stückchen „bergab“ Richtung weniger Fehler. Beobachte, wie der Loss fällt — das IST Lernen.

0%

Trainingsdaten

Tensor-Perspektive: Diese Tabelle wird zu zwei Tensoren: X mit Shape **[10, 1]** und Y mit Shape **[10, 1]**

Messung Nr.	Wasser (ml) = x	Wachstum (cm) = y
1	50 ml	2.5 cm
2	100 ml	5.0 cm
3	150 ml	7.5 cm
4	200 ml	10.0 cm
5	250 ml	12.5 cm
6	300 ml	15.0 cm
7	350 ml	17.5 cm
8	400 ml	20.0 cm
9	450 ml	22.5 cm
10	500 ml	25.0 cm

Psst — DU erkennst das Muster hier vermutlich auf einen Blick (100 ml → 5 cm, na?). Das Netz sieht davon NICHTS: für es sind das nur 20 nackte Zahlen in zwei Tensoren. Ob es die Regel trotzdem findet? Drück oben auf „Trainieren“ und schau ihm zu.

Modell-Struktur

Trainings-Ergebnis

Eigene Vorhersage machen

Gib eine Wassermenge ein, und die KI sagt dir das Pflanzenwachstum voraus:

Tensor-Perspektive: Deine Eingabe wird zu einem Tensor mit Shape [1, 1] umgewandelt und durch das Neuron transformiert!

Halte kurz inne: Du hast gerade eine KI trainiert. Kein Mensch hat ihr die Gieß-Regel verraten — sie hat *raten → Loss messen → bergab drehen* so lange wiederholt, bis aus zwei Zufallszahlen m ≈ 0,05 und b ≈ 0 wurden. Merke dir dieses Gefühl: In „Multiple Regression“ bekommt dein Netz gleich **drei** Stellschrauben auf einmal — und der Ablauf

bleibt haargenau derselbe.

Runter zum Tensor: Was fit() in 100 Epochen wirklich tut

Ein Klick, ein Fortschrittsbalken, fertig — zu schnell, um etwas zu sehen. Also halten wir die Zeit an und rechnen **einen einzigen Trainingsschritt von Hand nach**, mit den echten Zahlen dieses Moduls.

1 Die Daten sind zwei Tensoren — mehr sieht das Netz nie.
Die 10 Zeilen der Tabelle oben (fest im Modul, nichts wird gewürfelt) werden zu zwei Spalten-Tensoren — „Excel/CSV → Tensoren“. Die Shape-Schreibweise [10, 1] aus Kapitel 010 heißt hier: 10 Beispiele, je 1 Merkmal.

```
X // Wasser [ml] y // Wachstum [cm]
[[ 50], [[ 2.5],
 [100], [ 5.0],
 [150], [ 7.5],
   … …
 [500]] [25.0]]

Shape [10, 1] Shape [10, 1]
```

2 Das ganze Modell sind zwei Zahlen.
Eine Schicht, ein Neuron (Kapitel 110): Gewicht `W` mit Shape [1, 1], Bias `b` mit Shape [1]. Zwei Fließkommazahlen — das komplette „Wissen“ deiner ersten KI. Die Vorhersage für alle 10 Messungen auf einen Schlag ist eine Matrixmultiplikation aus Kapitel 060:

```
ŷ = X · W + b

[10,1] · [1,1] → [10,1] // eine Vorhersage pro Zeile
       + b [1] → broadcast // die EINE Bias-Zahl auf alle 10 Zeilen
```

3 Der Fehler schrumpft auf eine einzige Zahl.
Alle 10 Abweichungen quadrieren, Mittelwert bilden — der MSE aus Kapitel 115. Aus [10, 1] wird ein Skalar mit Shape [], ein Tensor ganz ohne Achsen. Das Modul startet mit einem winzigen Zufalls-Gewicht und `b = 0`; wir rechnen mit **W = 0, b = 0**, dann sagt das Netz überall 0 cm voraus:

```
MSE = mean( (ŷ − y)² ) = (2,5² + 5,0² + … + 25,0²) / 10 = 240,6
```

4 Ein Schritt bergab, vorgerechnet.
Die Steigung des Loss in Richtung `W` und `b` (Kapitel 118):

```
dL/dW = 2/N · Σ (ŷ−y)·x = 2/10 · (−48 125) = −9625
dL/db = 2/N · Σ (ŷ−y) = 2/10 · (−137,5) = −27,5

// klassischer Schritt (SGD) mit Lernrate 0,01:
W ← 0 − 0,01 · (−9625) = 96,25 // aua.
b ← 0 − 0,01 · (−27,5) = 0,275
```

Die Richtung stimmt (beide Vorzeichen negativ, also müssen W und b nach oben) — die

Schrittweite ist eine Katastrophe: W landet bei 96,25 statt 0,05, der Loss springt von 240,6 auf rund 890 Millionen, kurz darauf steht da `NaN`. Schuld sind die rohen Wassermengen: x geht bis 500, im Gradienten steckt x^2 — bis 250 000.

Genau deshalb steht hier `tf.train.adam(0.01)` statt SGD: **Adam** merkt sich, wie groß die Gradienten typischerweise sind, und teilt jeden Schritt durch diese Größe. Die Richtung bleibt die von oben, die Schrittweite passt Adam selbst an — der erste Schritt ist exakt 0,01 groß, egal ob der Gradient −9625 oder −27,5 war. Die saubere Lösung kommt in „Alles auf eine Skala“.

5 Und jetzt 100 Mal dasselbe.

Vorhersagen, Loss messen, Gradient, Schritt — **100 Epochen** lang über alle 10 Punkte (die passen in einen Rutsch, also ein Schritt pro Epoche). Nachgerechnet:

```
Epoche 1: W = 0,010 b = 0,010 Loss 153,8
Epoche 5: W = 0,047 b = 0,047 Loss 0,68
Epoche 10: W = 0,070 b = 0,070 Loss 40,0 // erst schießt Adam übers Ziel
Epoche 20: W = 0,044 b = 0,044 Loss 2,85 // dann pendelt es sich ein
Epoche 100: W = 0,050 b = 0,047 Loss 0,005
```

Ungefähr **W ≈ 0,050** und ein **b ≈ 0,05 cm**, das neben 25 cm Wachstum praktisch null ist — vergleich das mit den Werten oben im Modul. Deine Nachkommastellen weichen ab, weil dein Lauf mit einem *zufälligen* Gewicht startet; die Gieß-Regel findet er trotzdem.

Zurück zur Anwendung

Und wieder hoch: Dieselbe Rechnung wie in Kapitel 060 und Kapitel 110 von Hand — nur 100 Mal und ohne dich. In „Multiple Regression“ bekommt X mehrere Spalten, aus W wird [k, 1] — sonst ändert sich an diesen fünf Schritten *nichts*. Deshalb funktioniert derselbe Ablauf auch noch bei 175 Milliarden Zahlen.

Zum Weiterlesen

- **Dieselbe Gerade, andere Stimme:** Google Machine Learning Crash Course, *Linear Regression* — developers.google.com/machine-learning/crash-course (developers.google.com/machine-learning/crash-course/linear-regression) — kompakter Kurs mit Übungen zu Gewicht, Bias und Loss.
- **Das freie Lehrbuch:** James, Witten, Hastie, Tibshirani, *An Introduction to Statistical Learning* (PDF gratis) — statlearning.com — Kapitel 3 rechnet die lineare Regression statistisch sauber durch, inklusive der Frage, wie sicher m und b eigentlich sind.
- **Zum Nachbauen:** TensorFlow.js-Tutorial, *„Making predictions from 2D data“* — tensorflow.org/js/tutorials (tensorflow.org/js/tutorials/training/linear_regression) — genau dieses `fit()`-Muster als offizielles Tutorial, mit eigenen Daten.
- **Auf HerrKI:** 115 — Wie falsch ist falsch? und 118 — Gradient Descent zum Anfassen — die beiden Hälften dessen, was `fit()` gerade 100 Mal getan hat

Kurzer Check – mit Lösung

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

Frage 1: Was hat dein Netz beim Training wirklich gelernt?

- Es hat die Tabelle mit allen Messwerten auswendig gespeichert.
- **✔ Es hat zwei Zahlen (Gewicht m und Bias b) so lange verstellt, bis der Fehler minimal war.**
- Ein Mensch hat ihm die Gieß-Regel einprogrammiert.

Genau: $y = m \cdot x + b$ — nur zwei gelernte Zahlen, als Tensoren mit Shape [1, 1] gespeichert. Das ganze „Wissen" deiner ersten KI passt in zwei Kästchen!

Frage 2: Dein Netz hat m ≈ 0,05 cm/ml und b ≈ 0 gefunden. Was sagt es für 200 ml Wasser voraus?

- etwa 4 cm
- **✔ etwa 10 cm**
- etwa 0,25 cm

0,05 · 200 = 10 cm. Vorhersagen ist ab jetzt nur noch Einsetzen — genau das passiert bei der Inferenz im Forward Pass.

Frage 3: Woher „weiß" das Netz beim Training, in welche Richtung es m verstellen muss?

- Es probiert stumpf alle möglichen Werte durch.
- **✔ Aus dem Loss: Nach jedem Versuch wird der Fehler gemessen, dann geht es bergab in Richtung kleinerer Fehler.**
- Der Programmierer gibt nach jeder Epoche die Richtung vor.

Der Kreislauf aus Kapitel 5: raten → Fehler messen → nachjustieren. „Bergab" heißt: dem Gradienten des Loss folgen — alle Werte durchprobieren wäre schon bei 3 Gewichten hoffnungslos.

Das nimmst du mit

- **Das ganze Modell sind zwei Zahlen.** Gewicht m (Shape [1, 1]) und Bias b (Shape [1]) — mehr „Wissen“ hat deine erste KI nicht.
- **Training ist der Kreislauf aus Kapitel 005.** Vorhersagen, Loss messen, Gradient, Schritt — hier 100 Epochen lang, ohne dein Zutun.
- **Nichts wurde auswendig gelernt.** Aus 10 Messungen hat das Netz die Gieß-Regel „0,05 cm pro ml“ rekonstruiert, ohne sie je gesagt zu bekommen.
- **Vorhersagen ist danach nur noch Einsetzen.** 0,05 · 200 ml = 10 cm — ein Forward Pass, ganz ohne Gewichts-Updates.
- **Der Optimizer rettet den Start.** Mit rohem SGD springt W beim ersten Schritt auf 96,25 und der Loss ins Unendliche; Adam hält die Schrittweite klein — sauber gelöst wird das erst mit Skalierung.

Halt kurz inne — was ist hier gerade WIRKLICH passiert? Dein Netz hat zwei Zahlen gefunden (Gewicht m ≈ 0,05 und Bias b ≈ 0) und damit die Gieß-Regel aus der Tabelle — 0,05 cm pro ml — aus den Daten **rekonstruiert**, ohne sie je gesagt zu bekommen. Genau das ist Machine Learning. Weiter geht es mit „Multiple Lineare Regression“: mehrere Eingaben gleichzeitig, wie im echten Leben.

So zitierst du dieses Kapitel

HerrKI: Die erste Vorhersage: eine Gerade durch die Punkte (Lineare Regression). herrki.de/120_lineare_regression, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.