Mehrere Einflüsse gleichzeitig (Multiple Regression)

Kapitel online

Mehrere Eingaben gleichzeitig verarbeiten und daraus Wohnungspreise vorhersagen — multiple lineare Regression.

Praxis · Kapitel 16 von 50 · ca. 15 Min. · 150 XP

Einfach gesagt: Miete hängt nicht von einer Zahl ab, sondern von Fläche, Zimmern und Entfernung zugleich — dein Netz lernt für jede ein eigenes Gewicht.

Öffne ein Immobilienportal: Warum kostet die eine Wohnung 800 € und die andere 1.400 €? Nicht wegen EINER Zahl — Größe, Zimmerzahl und Baujahr spielen zusammen hinein. Fast jedes echte Problem sieht so aus. Solche Einflussgrößen nennt man **Features** — und heute lernt dein Netz zum ersten Mal, mehrere davon gleichzeitig zu verrechnen.

Aus Tensor-Sicht ändert sich dabei nur eines: Statt Shape [1] hat jede Eingabe jetzt Shape [3] — drei Zahlen pro Wohnung. Das Neuron macht daraus wie immer eine gewichtete Summe. Alles, was du in „Lineare Regression“ gelernt hast, gilt unverändert weiter — und obendrauf findest du im Vergleichs-Experiment heraus, mit welchen Einstellungen ein Netz am besten lernt.

Unser Fall heute: 10 Wohnungen mit *Fläche, Zimmerzahl und Entfernung zum Zentrum* — und ihren Kaltmieten. In den Daten steckt eine versteckte Preisregel. Du darfst gleich erst selbst raten (und wetten!), dann treten 4 Netze mit verschiedenen Einstellungen im Wettrennen an: Wer findet die Regel am besten?

HerrKI-Tipp: Input Shape = Anzahl Features. 3 Features? inputShape: [3]. Jedes Neuron im ersten Layer bekommt alle 3 Werte!

Die Trainingsdaten: 10 Wohnungen aus dem Portal

Tensor-Perspektive: Diese Tabelle wird zu X mit Shape **[10, 3]** (10 Wohnungen × 3 Features) und Y mit Shape **[10, 1]** (die Mieten).

Wohnung	Fläche (m²)	Zimmer	Entfernung (km)	Kaltmiete
1	45	2	10	640 €
2	60	2	5	920 €
3	30	1	15	310 €
4	80	3	8	1.150 €
5	100	4	3	1.540 €
6	55	2	12	720 €
7	70	3	20	790 €
8	120	5	2	1.850 €

9	40	1	6	610 €
10	90	4	18	1.120 €

Auch hier steckt eine exakte Regel drin — aber diesmal wirst selbst DU sie nicht durch Hingucken finden. Drei Ursachen gleichzeitig sind für Menschenaugen schwer zu entwirren. Für ein Neuron nicht.

⚖ **Eine Sache noch, bevor es losgeht:** Fläche läuft von 30–120, Zimmer nur von 1–5 — völlig verschiedene Skalen! Damit die große Fläche die kleinen Zimmer nicht überbrüllt, rechnen wir alle Features vor dem Training auf 0–1 um (*Normalisierung* — das ganze Warum samt Live-Duell: „Alles auf eine Skala“). Die Ergebniskarten rechnen die gelernten Gewichte für dich wieder in €/m², €/Zimmer und €/km zurück.

Am Gerät ausprobieren: herrki.de/130_multiple_regression#abschnitt-1

Direkt zum Modul

👋 HerrKI hier — heute wird es realistisch: mehrere Ursachen gleichzeitig

🧭 **Einfach gesagt:** Miete hängt nicht von einer Zahl ab, sondern von Fläche, Zimmern und Entfernung zugleich — dein Netz lernt für jede ein eigenes Gewicht.

Öffne ein Immobilienportal: Warum kostet die eine Wohnung 800 € und die andere 1.400 €? Nicht wegen EINER Zahl — Größe, Zimmerzahl und Baujahr spielen zusammen hinein. Fast jedes echte Problem sieht so aus. Solche Einflussgrößen nennt man **Features** — und heute lernt dein Netz zum ersten Mal, mehrere davon gleichzeitig zu verrechnen.

Aus Tensor-Sicht ändert sich dabei nur eines: Statt Shape [1] hat jede Eingabe jetzt Shape [3] — drei Zahlen pro Wohnung. Das Neuron macht daraus wie immer eine gewichtete Summe. Alles, was du in „Lineare Regression" gelernt hast, gilt unverändert weiter — und obendrauf findest du im Vergleichs-Experiment heraus, mit welchen Einstellungen ein Netz am besten lernt.

🎯 **Was du hier lernst:**

- ✅ Wie man **mehrere Features** als Input nutzt (Input Shape)
- ✅ Was ist **Batch-Training**? (Mehrere Beispiele gleichzeitig)
- ✅ Wie man **Feature-Normalisierung** macht (wichtig!)
- ✅ Wie man **Vorhersagen** mit mehreren Inputs macht

Unser Fall heute: 10 Wohnungen mit *Fläche, Zimmerzahl und Entfernung zum Zentrum* — und ihren Kaltmieten. In den Daten steckt eine versteckte Preisregel. Du darfst gleich erst selbst raten (und wetten!), dann treten 4 Netze mit verschiedenen Einstellungen im Wettrennen an: Wer findet die Regel am besten? 🏁

💡 ***HerrKI-Tipp: Input Shape = Anzahl Features.*** *3 Features? inputShape: [3]. Jedes Neuron im ersten Layer bekommt alle 3 Werte!* 🎯

🏠 Die Trainingsdaten: 10 Wohnungen aus dem Portal

📦 **Tensor-Perspektive:** Diese Tabelle wird zu `X` mit Shape **[10, 3]** (10 Wohnungen × 3 Features) und `Y` mit Shape **[10, 1]** (die Mieten).

Wohnung	📐 Fläche (m²)	🚪 Zimmer	📍 Entfernung (km)	💶 Kaltmiete
1	45	2	10	640 €
2	60	2	5	920 €
3	30	1	15	310 €
4	80	3	8	1.150 €
5	100	4	3	1.540 €
6	55	2	12	720 €
7	70	3	20	790 €
8	120	5	2	1.850 €
9	40	1	6	610 €
10	90	4	18	1.120 €

🤔 *Auch hier steckt eine exakte Regel drin — aber diesmal wirst selbst DU sie nicht durch Hingucken finden. Drei Ursachen gleichzeitig sind für Menschenaugen schwer zu entwirren. Für ein Neuron nicht.*

Interaktives Modul – am Gerät ausprobieren

Was du hier lernst

- Wie man **mehrere Features** als Input nutzt (Input Shape)
- Was ist **Batch-Training**? (Mehrere Beispiele gleichzeitig)
- Wie man **Feature-Normalisierung** macht (wichtig!)
- Wie man **Vorhersagen** mit mehreren Inputs macht

Für den Unterricht

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

Erkunden am Gerät (9 Min): die interaktiven Module auf herrki.de/130_multiple_regression ausprobieren.

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

Die wichtigsten Hyperparameter

Epochen (Epochs)

Was ist das?

Eine Epoche = Ein kompletter Durchlauf durch alle Trainingsdaten.

Beispiel:

Bei uns: „50 Epochen“ = das Netz blättert 50-mal durch alle 10 Wohnungen und dreht nach jeder Runde an seinen 3 Gewichten. Modell 2 bekommt 200 Runden — viermal mehr Übung.

Effekt:

- **Zu wenig:** Modell lernt nicht genug
- **Perfekt:** Modell lernt optimal
- **Zu viel:** Overfitting — auswendig gelernt statt verstanden (das ganze Drama dazu: Kapitel „Overfitting“)

Lernrate (Learning Rate)

Was ist das?

Wie groß die Schritte sind, mit denen das Modell seine Gewichte anpasst.

Analogie:

Der Schritt aus Kapitel 118, in Zahlen: Steigung +2,0 und LR 0,01 → Gewicht 0,30 → 0,28 (Trippelschritt). Mit LR 0,05 → 0,30 → 0,20 (Riesenschritt — schneller, aber wackliger).

Effekt:

- **Zu klein (0.0001):** Training dauert ewig

- **Perfekt (0.001-0.01):** Schnell & stabil
- **Zu groß (0.1+):** Modell "springt" und konvergiert nie

Optimizer

Was ist das?

Die Strategie, wie das Modell lernt.

Wir nutzen Adam:

- **Adam:** Intelligent, adaptiv, robust
- Passt die Lernrate automatisch an
- Funktioniert mit verschiedenen Learning Rates

Effekt:

Beispiel: Ein naiver Schritt wollte 0,03 → 2,03 springen — Adam bremst auf ≈ 0,04. Deshalb überlebt Modell 4 gleich (vermutlich!) sogar die riskante LR 0,05.

Batch Size (nicht heute)

Was ist das?

Wie viele Beispiele gleichzeitig verarbeitet werden.

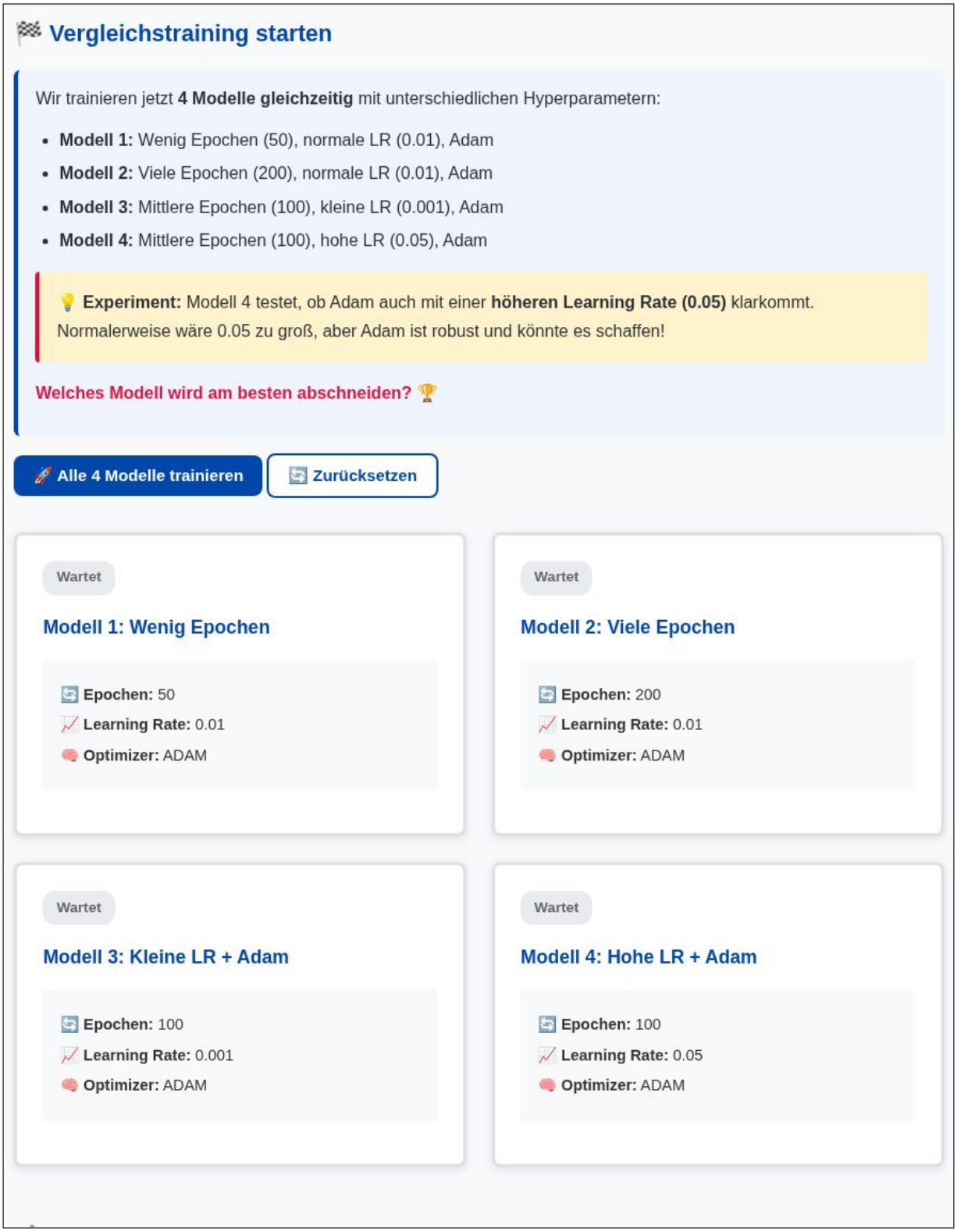
Heute:

Bei uns passen alle 10 Wohnungen auf einmal in einen Tensor [10, 3] — ein einziger „Batch“. Bei Millionen Datensätzen füttert man häppchenweise (Mini-Batches).

Vergleichstraining starten

Am Gerät ausprobieren: herrki.de/130_multiple_regression#abschnitt-3

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Wir trainieren jetzt **4 Modelle gleichzeitig** mit unterschiedlichen Hyperparametern:

- **Modell 1:** Wenig Epochen (50), normale LR (0.01), Adam
- **Modell 2:** Viele Epochen (200), normale LR (0.01), Adam
- **Modell 3:** Mittlere Epochen (100), kleine LR (0.001), Adam
- **Modell 4:** Mittlere Epochen (100), hohe LR (0.05), Adam

Experiment: Modell 4 testet, ob Adam auch mit einer **höheren Learning Rate (0.05)** klarkommt. Normalerweise wäre 0.05 zu groß, aber Adam ist robust und könnte es schaffen!

Welches Modell wird am besten abschneiden?

Und der Gewinner ist...

Runter zum Tensor: Wie funktioniert das im Detail?

Learning Rate - Die Schrittgröße beim Lernen

Stell dir vor, du stehst auf einem Berg im Nebel und willst ins Tal (= bester Gewichtswert). Du kannst nicht sehen, wo genau das Tal ist, aber du fühlst, in welche Richtung es abwärts geht. Dieses Bild kennst du schon: In „Gradient Descent zum Anfassen“ bist du den Berg selbst hinuntergestapft — hier siehst du, was die Schrittgröße im ECHTEN Training anrichtet.

Die Berg-Analogie

LR = 0.0001 (zu klein)

Winzige Schritte

Effekt: Du bewegst dich millimeterweise. Sicher, aber es dauert ewig bis zum Tal!

Training: Sehr langsam

LR = 0.001-0.01 (perfekt)

Normale Schritte

Effekt: Du gehst mit vernünftigen Schritten. Schnell und sicher ins Tal!

✔ Training: Optimal!

LR = 0.1+ (zu groß)

Riesige Sprünge

Effekt: Du springst wild umher, fliegst über das Tal hinweg und findest es nie!

✘ Training: NaN/Fehler

Was passiert konkret mit den Gewichten?

Bei jedem Trainingsschritt berechnet das Modell: **"In welche Richtung muss ich die Gewichte ändern?"**
Die Learning Rate bestimmt: **"Wie weit gehe ich in diese Richtung?"**

```
Neues Gewicht = Altes Gewicht - (Learning Rate × Gradient)
```

Adam Optimizer - Die intelligente Lernstrategie

Der Optimizer entscheidet, **wie intelligent** das Modell die Gewichte anpasst. Wir nutzen **Adam** - einer der besten Optimizer!

Wie Adam funktioniert

Learning Rate Effekte

⚖

Was passiert bei verschiedenen LR?

- **LR 0.001:** Sehr langsam, aber sicher

- **LR 0.01:** Optimal für die meisten Fälle
- **LR 0.05:** Schnell, aber riskant
- **LR 0.1+:** Meist NaN!

Formel bleibt gleich:

```
w = w - (LR × gradient)
```

Nur die LR-Zahl ändert sich!

Tipp: Adam kann mit größeren LR besser umgehen als andere Optimizer!

Adam (Adaptive Moment Estimation)

Wie es funktioniert:

1. Berechne Fehler (Loss)
2. Berechne Gradient (Richtung)
3. **Passe Schrittgröße intelligent an:**
    - Große Schritte am Anfang
    - Kleine Schritte nahe am Ziel
    - Merkt sich vorherige Gradienten
4. Wiederhole

Formel (vereinfacht):

```
w = w - (adaptierte_LR × gradient)
```

✔ **Vorteil:** Intelligent, adaptiv, meist schneller
✘ **Nachteil:** Komplexer, braucht mehr Speicher

Konkrete Beispiel-Rechnung mit Adam

Stellen wir uns vor, unser Gewicht ist gerade `w = 0.03` und wir wollen `w = 0.05` erreichen.

Wie Adam das Gewicht anpasst:

Epoche 1:

w = 0.0300 - (0.01 × -0.5) = **0.0350** ⚡

Gradient: -0.5 → Gewicht erhöhen!

Epoche 2:

w = 0.0350 + 0.0032 = **0.0382** ⚡

Adam passt die Schrittgröße an

Nach mehreren Epochen:

w → **0.05**

Perfekt gelernt!

⚡ Adam macht intelligente, adaptive Schritte und erreicht das Ziel schnell und stabil!

Was passiert mit den Tensoren?

Der komplette Trainingsablauf

Schritt für Schritt:

1

Forward Pass (Vorhersage)

Input-Tensor `[100]` fließt durch das Netz:
`output = input × gewicht + bias = 100 × 0.03 + 0 = 3.0`
Output-Tensor `[3.0]`

2

Loss berechnen (Fehler messen)

Erwarteter Output: `5.0`, Tatsächlicher Output: `3.0`
`Loss = (5.0 - 3.0)² = 4.0` ← Der Fehler!

3

Backward Pass (Gradient berechnen)

Berechne: **"In welche Richtung muss ich das Gewicht ändern, um den Fehler zu verringern?"**
`Gradient für Gewicht = -200` (negative = Gewicht erhöhen!)

4

Adam Update (Gewicht anpassen)

Der rohe Gradient ist riesig (−200): Ein naiver Schritt `0.03 − 0.01 × (−200) = 2.03` würde meilenweit über das Ziel (0.05) hinausschießen!
Adam begrenzt deshalb die Schrittweite auf ungefähr Lernraten-Größe: `neues_gewicht ≈ 0.03 + 0.01 = 0.04`
Genau das heißt „adaptiv“: Die Richtung kommt vom Gradienten, die Schrittgröße bleibt unter Kontrolle.

5

Wiederhole (nächste Epoche)

Mit dem neuen Gewicht starten wir wieder bei Schritt 1. Nach vielen Epochen konvergiert das Gewicht zu `≈0.05`!

Zusammenfassung: Was macht der Optimizer mit Tensoren?

- **Liest** die aktuellen Gewichts-Tensoren
- **Berechnet** Gradienten-Tensoren (Richtung der Änderung)
- **Wendet** die Learning Rate an (Schrittgröße)
- **Aktualisiert** die Gewichts-Tensoren
- **Bei Adam:** Passt die Schrittgröße intelligent pro Gewicht an!

Zurück zur Anwendung

Zurück zum Wohnungsportal vom Anfang — dieselben 10 Wohnungen, dieselbe Frage: Warum kostet die eine 800 € und die andere 1.400 €? Dein Netz beantwortet das jetzt nicht mehr mit einer einzigen Stellschraube, sondern mit **drei Gewichten nebeneinander**: eines für die Fläche, eines für die Zimmerzahl, eines für die Entfernung zum Zentrum. Diese drei Zahlen kannst du ab sofort *lesen* — ein großer Betrag heißt „zieht den Preis kräftig“, ein **Minus** heißt „drückt ihn“, so wie jeder weitere Kilometer Richtung Stadtrand. Genau so arbeitet jeder Miet- und Preisrechner, jede Kredit-Scorecard und jede Verbrauchsprognose: eine gewichtete Summe über mehrere Merkmale (Features) — nur mit 50 Spalten statt mit drei.

Ein Warnschild gehört dazu: Ein Gewicht ist ein *Zusammenhang in diesen Daten*, kein Beweis für eine Ursache (Scheinkausalität) — wer eine Wohnung 5 km näher ans Zentrum trägt, macht sie damit nicht teurer, und in einem anderen Datensatz kann dasselbe Vorzeichen kippen. Und dass das Modul die drei Spalten vor dem Training auf eine gemeinsame Skala rechnet, war kein Detail am Rande: Fläche 30–120 gegen Zimmer 1–5 — ungleiche Größenordnungen bringen jedes Training aus dem Tritt. Das ganze Warum steht im nächsten Kapitel: „Alles auf eine Skala (Normalisierung)“.

Zum Weiterlesen

- **Mehrere Features im Lehrbuch:** James, Witten, Hastie, Tibshirani, *An Introduction to Statistical Learning*, Kapitel 3 (PDF gratis) — statlearning.com — was es heißt, wenn mehrere Merkmale gleichzeitig auf eine Zielgröße wirken — und welche davon wirklich etwas beitragen.
- **Der Gewinner im Original:** Diederik Kingma & Jimmy Ba, *„Adam: A Method for Stochastic Optimization“* (2014) — arxiv.org/abs/1412.6980 — das Paper zu dem Optimizer, der hier in allen vier Modellen steckt.
- **Echte Daten zum Üben:** Kaggle, *„House Prices — Advanced Regression Techniques“* — kaggle.com/competitions/house-prices... (kaggle.com/competitions/house-prices-advanced-regression-techniques) — 79 Merkmale zu rund 1.500 Häusern als CSV: genau die Aufgabe dieses Kapitels, nur in groß (auf Englisch, kostenlose Anmeldung).
- **Auf HerrKI:** 135 — Alles auf eine Skala (warum wir die Features vorher auf 0–1 gerechnet haben) und 140 — Polynomiale Regression (die erste Zwischenschicht)

Kurzer Check – mit Lösung

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

Frage 1: Für Fläche, Zimmerzahl und Entfernung lernt dein Netz einen Gewichts-Tensor mit Shape [3]. Warum drei?

- Weil drei Epochen trainiert werden.
- **✓ Ein Gewicht pro Eingabe-Feature — das Netz lernt, wie stark jedes einzelne Merkmal den Preis beeinflusst.**
- Weil der Preis dreistellig ist.

Richtig! Jedes Feature bekommt seinen eigenen „Lautstärkeregler". Bei 10 Features wäre der Shape [10] — das Prinzip bleibt exakt gleich.

Frage 2: Nach dem Training sind die Gewichte (0,5 · 10 · −5) für (m² · Zimmer · km Entfernung). Was bedeutet das Minus?

- Die Entfernung war ein unwichtiges Feature.
- Beim Training ist ein Fehler passiert — Gewichte sind nie negativ.
- **✓ Mehr Entfernung senkt den Preis: Das Netz hat den Zusammenhang „weiter draußen = günstiger" selbst gefunden.**

Negative Gewichte sind völlig normal und oft die spannendsten: Sie zeigen Merkmale, die entgegengesetzt wirken. Das Netz hat das aus den Beispielen gelernt — niemand hat es ihm gesagt.

Frage 3: Was macht der Adam-Optimizer besser als ein stur fester Lernschritt?

- **✓ Er passt die Schrittgröße für jedes Gewicht einzeln an und verhindert Überschießen.**
- Er lässt das Training ganz ohne Loss-Funktion laufen.
- Er macht das Netz größer, wenn der Fehler steigt.

Genau — statt blind einen Riesenschritt (0,03 → 2,03) zu machen, bleibt Adam bei kontrollierten Schritten (≈ 0,03 → 0,04) und kommt damit zuverlässig ins Tal. Mehr Gefühl dafür bekommst du im Kapitel „Gradient Descent zum Anfassen".

Das nimmst du mit

- **Input Shape = Anzahl Features.** Drei Zahlen pro Wohnung heißt `inputShape: [3]` — aus der Tabelle wird X mit Shape [10, 3].
- **Ein Gewicht pro Feature.** Jedes Merkmal bekommt seinen eigenen „Lautstärkeregler“; bei 10 Merkmalen wäre der Gewichts-Tensor eben [10].
- **Negative Gewichte sind die spannenden.** Das Minus bei der Entfernung heißt „weiter draußen = günstiger“ — selbst aus 10 Zeilen gelernt, niemand hat es gesagt.
- **Ungleiche Skalen müssen raus.** Fläche 30–120 gegen Zimmer 1–5: vor dem Training alles auf 0–1 rechnen, sonst überbrüllt die große Spalte die kleine.
- **Hyperparameter entscheiden das Rennen.** Gleiche Daten, vier Einstellungen, sichtbar verschiedene Ergebnisse — und Adam passt die Schrittgröße für jedes Gewicht einzeln an.

Du verstehst jetzt, wie Hyperparameter im Detail funktionieren! Weiter geht es mit „Polynomiale Regression“ — dort bekommt dein Netz seinen ersten Hidden Layer und lernt Kurven.

So zitierst du dieses Kapitel

HerrKI: Mehrere Einflüsse gleichzeitig (Multiple Regression). herrki.de/130_multiple_regression, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.