Bergab zum kleinsten Fehler (Gradient Descent)

Kapitel online

Im Nebel den Berg hinab: Lernrate einstellen, Schritt für Schritt ins Tal — und einmal zusehen, wie eine zu große Schrittweite die Kugel aus dem Bild schleudert.

Praxis · Kapitel 13 von 50 · ca. 10 Min. · 150 XP

Einfach gesagt: Wie im dichten Nebel ertastet die KI das Gefälle unter sich und geht Schritt für Schritt bergab, bis der Fehler minimal ist.

Du weißt aus „Wie falsch ist falsch?“, dass der Loss eine Zahl ist. Aber wie findet ein Modell die Einstellung, bei der diese Zahl am kleinsten ist — bei **Millionen** von Gewichten? Die Antwort ist ein einziger, wunderbar simpler Trick. Er heißt **Gradient Descent**, und du hast ihn in Kapitel „Was heißt Lernen?“ beim Zahlenraten schon selbst benutzt.

Was du hier lernst

- Warum **Gradient Descent** nichts anderes ist als bergab gehen im Nebel
- Was der **Gradient** ertastet — die Steigung des Loss an genau dieser Stelle
- Wieso die **Lernrate** der wichtigste Drehknopf ist — und woher „Loss: NaN“ kommt
- Was **Plateau** und **lokales Minimum** im Training wirklich bedeuten
- Dass der komplette Lernschritt aus **einer Zeile** besteht — auch bei einer Million Gewichte

Für den Unterricht

Einstieg (2 Min): Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

Erkunden am Gerät (6 Min): die interaktiven Module auf herrki.de/118_gradient_descent ausprobieren.

Sichern mit Quiz (2 Min): „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

Die Berg-Metapher — und sie ist wörtlich gemeint

Stell dir vor: Du stehst an einem Berghang, im **dichtesten Nebel** deines Lebens. Du willst ins Tal. Du siehst: nichts. Aber du kannst mit den Füßen **das Gefälle unter dir ertasten**. Also: spüren, in welche Richtung es bergab geht → einen Schritt in diese Richtung → wieder spüren → wieder ein Schritt. Genau so — und *nicht* anders — findet jedes neuronale Netz dieser Welt seine Gewichte.

1. **Der Berg** = die Loss-Landschaft: für jede Gewichts-Einstellung die Höhe des Fehlers
2. **Das Gefälle ertasten** = den **Gradienten** berechnen (die Steigung des Loss)
3. **Die Schrittweite** = die **Lernrate** — DER Drehknopf des ganzen Trainings

Am Gerät ausprobieren: herrki.de/118_gradient_descent#abschnitt-2

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Die drei Landschaften — drei Lektionen fürs Leben

Schüssel: Der Traumfall — funktioniert mit fast jeder Lernrate. Aber stell mal **> 0,9** ein: Die Kugel springt über das Tal hinweg, schaukelt sich auf und **fliegt aus dem Bild**. In echt heißt das: „mein Loss ist NaN“ — der häufigste Anfänger-Frust überhaupt. Jetzt weißt du für immer, was da passiert.

Plateau: Fast kein Gefälle → winzige Schritte → gefühlter Stillstand. *Deshalb* dauert Training manchmal ewig, obwohl nichts kaputt ist.

Zwei Täler: Je nach Startpunkt landet die Kugel im **falschen Tal** (lokales Minimum) —

würfle ein paarmal neu! Auch echte Trainings finden nie „das perfekte“, sondern „ein gutes“ Tal. Meistens reicht das völlig.

Runter zum Tensor — Schritt für Schritt

Bis hierhin lief unsere Kugel auf **einer** Achse — als hätte das Netz nur ein einziges Gewicht. Echte Netze haben viele. Was ändert sich dann? Erstaunlich wenig. Rechnen wir es einmal mit **zwei** Gewichten von Hand durch:

1 Die Position der Kugel IST ein Tensor.
Unser Mini-Netz habe zwei Stellschrauben (Tensoren: Kapitel 010):

```
gewichte = [0.30, 1.20] // wo die Kugel gerade steht – jetzt eine LANDKARTE statt einer Linie
```

2 Der Gradient hat dieselbe Form — eine Steigung PRO Gewicht.
Das Netz „ertastet“ für *jedes* Gewicht einzeln: „Wird der Fehler größer oder kleiner, wenn ich hier ein winziges bisschen drehe?“ Heraus kommt:

```
gradient = [+2.0, -0.5] // Steigung unter Gewicht 1 und unter Gewicht 2
```

So liest du das: **+2.0** = „bei Gewicht 1 geht es steil *bergauf*, wenn du es erhöhst — also runter damit, und zwar deutlich.“ **−0.5** = „bei Gewicht 2 geht es leicht bergab, wenn du erhöhst — also ruhig etwas hoch damit, aber sachte.“ Das Vorzeichen sagt die *Richtung*, die Größe sagt die *Dringlichkeit*.

3 Der Schritt: elementweise rechnen — genau wie in „Tensor-Operationen“ gelernt.
Mit Lernrate 0.1 heißt „einen Schritt bergab machen“:

```
neu = [ 0.30 − 0.1·(+2.0), 1.20 − 0.1·(−0.5) ]
    = [ 0.10, 1.25 ] // Gewicht 1 kraeftig runter, Gewicht 2 minimal hoch
```

Beide Gewichte bewegen sich gleichzeitig — jedes nach seiner eigenen Steigung: wo es steil ist, macht die Kugel einen großen Schritt, wo es flach ist, nur einen Trippelschritt. Deshalb „rollt“ sie wie von selbst ins Tal.

Und bei einer Million Gewichte? Exakt dasselbe — die beiden Listen sind dann bloß eine Million Einträge lang. Kein Gewicht sieht je die ganze Landschaft (unser Nebel!): jedes kennt nur die Steigung direkt unter sich, macht seinen kleinen Schritt — und aus einer Million Trippelschritten pro Runde wird Lernen. Die Lern-Regel bleibt dabei EINE Zeile:

```
// Der komplette "Lern"-Schritt jeder KI, von der Foto-App bis zum Chatbot:
gewichte = gewichte.sub( gradient.mul(lernrate) );
// neuer Wert = alter Wert − Schrittweite × Steigung … millionenfach, tausende Schritte
```

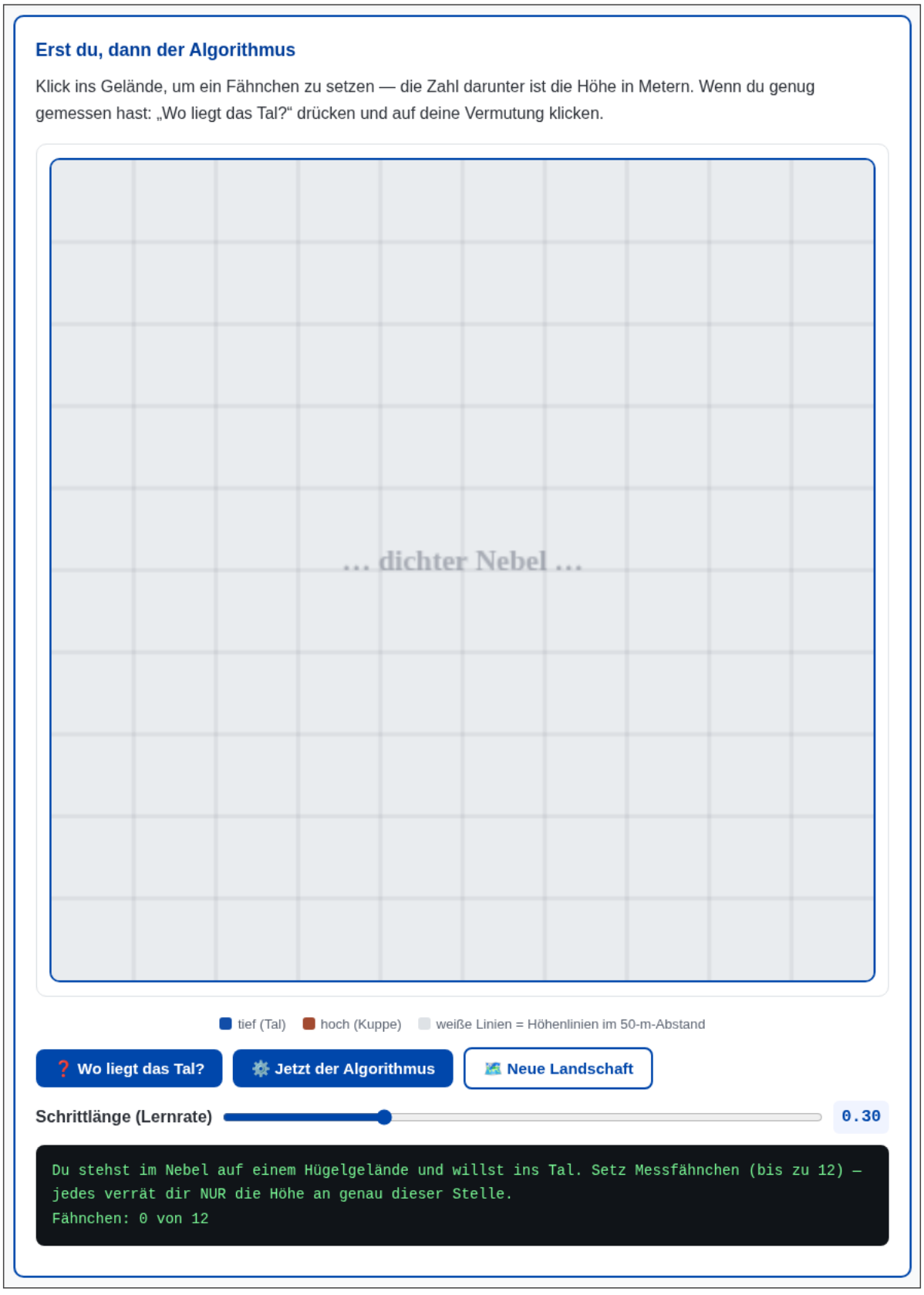
Im Nebel bergab: wie findest du das Tal, ohne die Karte zu sehen?

Zwei Gewichte kann man nicht nur ausrechnen — man kann sie **sehen**. Genau das ist die Landkarte hier unten: die eine Achse ist Gewicht 1, die andere Gewicht 2, und die Höhe an jedem Punkt ist der Loss für diese Einstellung. Nur: Du siehst sie nicht. Du stehst mitten drin, im Nebel, und hast einen **Höhenmesser** dabei. Der sagt dir die Höhe an genau der Stelle, an der du stehst. Sonst nichts.

Du darfst bis zu **12 Messfähnchen** setzen. Danach tippst du, wo das Tal liegt. Und danach lässt du den Algorithmus dieselbe Aufgabe lösen — mit derselben einen Zeile von eben.

Am Gerät ausprobieren: herrki.de/118_gradient_descent#abschnitt-4

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Was du gerade gesehen hast

1**Raten ist teuer.** 12 Messungen auf einem Gelände mit 10.000 möglichen Stellen — ungefähr in diesem Verhältnis kennt ein Netz seine Loss-Landschaft. Nur dass die dort nicht zwei Achsen hat, sondern Millionen.

2**Der Algorithmus rät nicht.** Er misst das Gefälle unter den Füßen und macht einen Schritt bergab — immer dieselbe Regel, immer dieselbe eine Zeile. Meist braucht er dafür weniger Schritte, als du Fähnchen hattest.

3**Er sieht trotzdem nichts.** Startest du auf der falschen Seite des Höhenrückens, landet er im **Seitental** — dem lokalen Minimum — und bleibt dort zufrieden stehen. Er hat keinen Fehler gemacht; ihm fehlt schlicht die Information, dass es nebenan tiefer geht. Das ist der Fall „ Zwei Täler“ von oben, nur eben zweidimensional.

4**Die Schrittlänge entscheidet mit.** Zieh die Lernrate auf 1,00 und setz den Startpunkt neu: Der Wanderer springt über das Tal hinweg und schaukelt sich auf — dieselbe Katastrophe wie bei der Kugel, nur in zwei Richtungen gleichzeitig.

Was man gegen Seitentäler unternimmt — Schwung aus den letzten Schritten mitnehmen (**Momentum**) oder die Schrittlänge für jedes Gewicht einzeln anpassen (**Adam**) — steht in „Multiple Regression“. Ein Wundermittel ist beides nicht; es macht Seitentäler nur seltener. Und weil niemand eine Landschaft mit einer Million Achsen malen kann, bleibt der Nebel für immer da — man lernt nur, gut darin zu laufen.

Zurück zur Anwendung

Bevor wir das erste Netz einem Framework überlassen, bauen wir eines von Hand: In „Ein neuronales Netz in 150 Zeilen PHP — Brainy zum Anfassen“ siehst du Zeile für Zeile, was dahintersteckt. Danach geht es mit „Einfache Lineare Regression“ weiter — dort rufst du zum ersten Mal `model.fit()` auf und weißt jetzt genau, was drinsteckt: Loss messen (115), Gefälle ertasten, Schritt machen (dieses Kapitel) — im Kreis, bis es passt (005). Und wenn dabei je „Loss: NaN“ auftaucht, grinst du nur und drehst die Lernrate runter.

Zum Weiterlesen

- **Der Überblick:** Sebastian Ruder, *„An overview of gradient descent optimization algorithms“* (2016) — arxiv.org/abs/1609.04747 — wie aus deinem einen Schritt SGD, Momentum, RMSProp und Adam werden.
- **Zum Spielen:** Distill, *„Why Momentum Really Works“* — distill.pub/2017/momentum — interaktiv wie unser Kugel-Modul: warum ein bisschen Schwung aus dem Zickzack heraushilft.
- **Sauber hergeleitet:** Michael Nielsen, *Neural Networks and Deep Learning*, Kapitel 1 — neuralnetworksanddeeplearning.com/chap1.html — dieselbe Tal-Metapher, nur mit der Mathematik dahinter.
- **Auf HerrKI:** 115 — Wie falsch ist falsch? (die Höhe der Landschaft) und 135 — Alles auf eine Skala (wie du den schmalen Graben wieder rund bekommst)

Kurzer Check – mit Lösung

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

Frage 1: Was „ertastet“ der Gradient?

- Die Anzahl der Datenpunkte.
- ✔ **Die Steigung der Loss-Landschaft an der aktuellen Stelle — also die Richtung, in der es bergab geht.**
- Die beste Lösung, direkt und ohne Umwege.

Der Gradient kennt nur die LOKALE Steigung — wie deine Füße im Nebel. Deshalb braucht es viele kleine Schritte statt eines großen Sprungs.

Frage 2: Dein Training meldet plötzlich „Loss: NaN“. Die wahrscheinlichste Ursache?

- ✔ **Die Lernrate ist zu groß — die Schritte springen über das Tal und schaukeln sich auf.**
- Der Computer ist zu langsam.
- Es gibt zu viele Daten.

Genau das hast du in der Schüssel gesehen: zu große Schritte → Aufschaukeln → die Werte explodieren ins Unendliche. Erste Hilfe: Lernrate verkleinern (z. B. ÷10).

Frage 3: Warum kann die Kugel in der Zwei-Täler-Landschaft „falsch“ landen?

- Weil der Gradient absichtlich lügt.
- ✔ **Weil Gradient Descent nur bergab läuft — aus einem lokalen Tal führt jeder Weg erst einmal bergauf.**
- Weil die Lernrate immer zu klein ist.

Ein lokales Minimum ist wie eine Mulde am Hang: ringsum geht es bergauf, also bleibt die Kugel liegen. Gegenmittel in der Praxis: mehrmals mit anderem Start würfeln — genau wie hier.

Das nimmst du mit

- **Gradient Descent ist bergab gehen im Nebel.** Gefälle ertasten, einen Schritt machen, wieder ertasten — mehr steckt nicht dahinter.
- **Der Gradient kennt nur deinen Standort.** Eine Steigung pro Gewicht: das Vorzeichen sagt die Richtung, die Größe die Dringlichkeit.
- **Die Lernrate ist DER Drehknopf.** Zu groß → die Kugel schaukelt sich auf und der Loss wird `NaN`; zu klein → gefühlter Stillstand. Erste Hilfe: durch 10 teilen.
- **Plateau und lokales Minimum sind normal.** Flach heißt langsam, nicht kaputt — und auch echte Trainings finden nie „das perfekte“, sondern ein gutes Tal.
- **Der komplette Lernschritt ist eine Zeile.** `gewichte.sub(gradient.mul(lernrate))` — bei zwei Gewichten genauso wie bei einer Million.

So zitierst du dieses Kapitel

HerrKI: Bergab zum kleinsten Fehler (Gradient Descent). herrki.de/118_gradient_descent, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.