Wenn die Gerade sich biegen muss (Polynomiale Regression)

Kapitel online

Wenn eine gerade Linie nicht mehr passt: wie neuronale Netze krumme, nicht-lineare Zusammenhänge lernen.

Praxis · Kapitel 18 von 50 · ca. 25 Min. · 200 XP

Einfach gesagt: Doppelt so viel Wasser heißt nicht doppelt so viel Wachstum — mehrere Neuronen in einer Zwischenschicht lernen die Kurve mit dem Optimum in der Mitte.

Gieß deine Pflanze doppelt so viel — wächst sie dann doppelt so schnell? Eben nicht: Zu wenig Wasser lässt sie verdursten, zu viel ertränkt sie. Die Wahrheit ist eine **Kurve** mit einem Optimum in der Mitte. Und eine Gerade — alles, was dein Netz bisher konnte — ist gegen so etwas chancenlos.

In „Multiple Regression“ hast du deinem Netz mehr *Eingänge* gegeben. Heute bekommt es zum ersten Mal *Tiefe*: einen **Hidden Layer** — eine Zwischenschicht aus 8 Neuronen zwischen Eingang und Ausgang. Mit ihr kann das Netz Geraden knicken, stapeln und zu Kurven zusammensetzen. Das ist der Moment, in dem neuronale Netze wirklich mächtig werden.

Roter Faden: Wir gehen wieder den ganzen Weg — von der Gießkanne runter zu den Tensoren und mit einer fertigen Vorhersage wieder hoch.

Was du hier lernst

- Was ein **Hidden Layer** ist — und warum er aus 1 Zahl 8 „Meinungen“ macht
- Warum es ohne **Aktivierungsfunktion** (ReLU) keine Kurven gäbe
- Warum 8 Neuronen mit derselben Formel trotzdem 8 verschiedene Dinge lernen
- Der Unterschied zwischen **Training** und **Inferenz**

Für den Unterricht

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

Erkunden am Gerät (15 Min): die interaktiven Module auf herrki.de/140_polynomiale_regression ausprobieren.

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

Die neue Architektur: eine Schicht dazwischen

Bisher sah dein Netz so aus: eine Zahl rein, ein Neuron rechnet, eine Zahl raus. Jetzt schieben wir eine Zwischenschicht ein — und plötzlich läuft jede Eingabe durch **8 Neuronen gleichzeitig**:

Input Layer [1] Wassermenge

→

⋮

Hidden Layer [8] 8 Neuronen (ReLU)

→

Output Layer [1] Wachstum (cm)

Was mit dem Tensor passiert:

1. **Input-Tensor:** [1] — eine einzige Zahl, z. B. 250 ml
2. **Hidden Layer:** [1] wird zu [8] — aus einer Zahl werden 8 verschiedene „Blickwinkel“
3. **Aktivierung (ReLU):** jedes der 8 Neuronen schneidet Negatives ab: `max(0, x)`
4. **Output Layer:** [8] wird zu [1] — die 8 Blickwinkel werden zu EINER Vorhersage gemischt

Schlüssel-Erkenntnis:

Der Hidden Layer ist wie ein Team aus 8 Gutachtern: Alle sehen dieselbe Wassermenge, aber jeder achtet auf etwas anderes — „ist es zu wenig?“, „ist es optimal?“, „ist es zu viel?“. Das Output-Neuron hört sich alle 8 Meinungen an und mischt daraus die endgültige Antwort.

Runter zum Tensor: Schauen wir einer Zahl bei der Reise zu: 250 ml

Genug Überblick — wir schicken jetzt EINEN konkreten Wert durchs Netz und rechnen jeden Schritt mit. Das ist unser gewohnter Weg „runter zum Tensor“:

Schritt 1: Input Layer

Tensor: [1]

```
[250]
```

Eine einzige Zahl: die Wassermenge in Milliliter.

Schritt 2: Hidden Layer — aus 1 mach 8

Transformation: [1] → [8]. Jedes der 8 Neuronen rechnet dieselbe Mini-Formel, aber mit *eigenem* Gewicht und Bias:

```
neuron[i] = input × gewicht[i] + bias[i]
```

Neuron 1 zum Beispiel:

```
neuron[0] = 250 × 0.05 + (−2.0) = 10.5
```

Alle 8 zusammen (vor ReLU):

```
[10.5, −5.2, 8.3, −1.0, 15.6, 3.2, −8.1, 12.4]
```

8 verschiedene „Blickwinkel“ auf dieselbe Wassermenge — manche positiv, manche negativ.

Schritt 3: ReLU — der Türsteher

Regel: `ReLU(x) = max(0, x)` — Positives darf durch, Negatives wird zu 0.

Vor ReLU:

```
[10.5, −5.2, 8.3, −1.0, 15.6, 3.2, −8.1, 12.4]
```

Nach ReLU:

```
[10.5, 0, 8.3, 0, 15.6, 3.2, 0, 12.4]
```

Warum dieser simple Schnitt alles ändert: Ohne ReLU wäre das ganze Netz — egal wie viele Schichten — nur eine kompliziert aufgeschriebene Gerade. Erst die Knicke machen Kurven möglich. In „Warum eine Gerade nicht reicht“ baust du solche Kurven aus einzelnen ReLU-Knicken selbst zusammen!

Schritt 4: Output Layer — aus 8 mach 1

Transformation: [8] → [1]. Das Ausgabe-Neuron gewichtet alle 8 Meinungen und addiert sie zusammen:

```
output = h[0]×w[0] + h[1]×w[1] + … + h[7]×w[7] + bias output = 10.5×0.8 + 0×0.3 + 8.3×0.6 + … = 20.1
```

Finale Vorhersage: `20,1 cm Wachstum`

Die ganze Reise in vier Zeilen

```
Input: [250] ← Wassermenge Hidden vor ReLU: [10.5, −5.2, 8.3, −1.0, 15.6, 3.2, −8.1, 12.4] Hidden nach ReLU: [10.5, 0, 8.3, 0, 15.6, 3.2, 0, 12.4] Output: [20.1] ← Wachstum in cm
```

Warum rechnen die 8 Neuronen nicht alle das Gleiche?

Die Frage ist absolut berechtigt: Alle 8 benutzen exakt dieselbe Formel `input × gewicht + bias`. Müssten sie dann nicht alle dasselbe Ergebnis liefern — und wir hätten 8-mal dasselbe Neuron bezahlt?

Die Antwort steckt in zwei Wörtern: Zufall + Feedback.

Schritt 1: Der Zufall verteilt die Startpositionen

Vor dem Training würfelt TensorFlow für jedes Neuron ein eigenes Startgewicht. Schon beim allerersten Forward Pass (Input: 250 ml) rechnet deshalb jedes Neuron etwas anderes:

Neuron	Start-Gewicht (gewürfelt)	250 × Gewicht	nach ReLU
1	+0.523	+130.8	130.8
2	−0.812	−203.0	**0** ✂
3	+0.156	+39.0	39.0
4	−0.391	−97.8	**0** ✂
5	+0.687	+171.8	171.8
6	+0.234	+58.5	58.5
7	−0.567	−141.8	**0** ✂
8	+0.445	+111.3	111.3

(Alle Biases starten bei 0, deshalb lassen wir sie in der Tabelle weg. ✂ = von ReLU abgeschnitten.)

Das nennt man „Symmetry Breaking“: Würden alle 8 mit demselben Gewicht starten, bekämen sie auch immer dasselbe Feedback — und blieben für alle Ewigkeit identische Klone. Der Zufall am Anfang bricht diese Symmetrie. Einmal verschieden, immer verschieden.

Schritt 2: Das Feedback verstärkt die Unterschiede

Bei der Backpropagation (kennst du aus „Gradient Descent“) fragt das Netz: „Wie viel hat JEDES Neuron zum Fehler beigetragen?“ Weil alle 8 unterschiedlich gerechnet haben, bekommt auch jedes seinen eigenen Korrektur-Befehl:

Neuron	Gradient	Korrektur
1	−0.42	Gewicht wird kleiner
2	+1.83	Gewicht wird größer (weniger negativ)
3	−0.87	Gewicht wird kleiner
4	+0.56	Gewicht wird größer
5	−1.21	Gewicht wird kleiner
6	−0.34	Gewicht wird kleiner
7	+0.91	Gewicht wird größer
8	−0.67	Gewicht wird kleiner

✔ **Der entscheidende Punkt:** Jedes Neuron hat einen anderen Output produziert, deshalb bekommt jedes einen anderen Gradienten. Die Unterschiede werden mit jedem Trainingsschritt **größer**, nicht kleiner!

Schritt 3: Nach 300 Epochen — jeder hat seinen Job

Aus 8 zufälligen Startpunkten sind 8 Spezialisten geworden:

Neuron	Gelerntes Gewicht	Sein Spezialgebiet
1	`+0.382`	Mittlere Wassermengen (linearer Anteil)
2	`−0.623`	„Zu wenig Wasser“-Detektor
3	`+0.891`	„Optimaler Bereich“-Detektor
4	`−0.287`	Unterstützt die „Zu wenig“-Erkennung
5	`+0.534`	Allgemeine Wachstumsrate
6	`+0.112`	Kleine Korrekturen
7	`−0.478`	„Zu viel Wasser“-Warner
8	`+0.723`	Verstärkt den optimalen Bereich

Darum lohnt sich das 8er-Team

1. **Zufälliger Start:** jedes Neuron beginnt woanders
2. **Andere Rechnung:** andere Gewichte → andere Outputs
3. **Anderes Feedback:** jedes bekommt seinen eigenen Gradienten
4. **Spezialisierung:** über die Epochen entstehen 8 verschiedene „Rollen“

Analogie: 8 Leute lösen gemeinsam ein Puzzle. Jeder fängt zufällig woanders an — und ganz von selbst wird einer zum Rand-Spezialisten und eine zur Himmel-Expertin. Niemand hat das eingeteilt! Genau so entstehen die „Zu-wenig-Wasser-Detektoren“ und „Übergieß-Warner“ in deinem Netz.

Training vs. Inferenz — zwei Betriebsarten, ein Netz

Dein Netz kennt zwei Modi — genau wie du beim Lernen: **Üben mit Lösungsheft** (Training) und **die Prüfung** (Inferenz). Der ganze Unterschied steckt darin, was mit den Gewichts-Tensoren passiert:

Training (Üben mit Lösungsheft)

Ziel: Die Gewichte so lange anpassen, bis die Vorhersagen zu den bekannten Antworten passen.

1

Forward Pass

Input-Tensor → Hidden Layer → Output-Tensor

```
[250] → [8 Neuronen] → [20.5] (Vorhersage)
```

2

Loss berechnen

Wie falsch war die Vorhersage? (Kennst du aus „Fehler messen“)

```
Loss = (20 − 20.5)² = 0.25
```

3

Backward Pass (Backpropagation)

Der Fehler wandert rückwärts durchs Netz und berechnet für **jedes einzelne Gewicht** einen Gradienten — erst im Output Layer, dann im Hidden Layer.

4

Gewichte aktualisieren

Der Optimizer macht bei jedem Gewicht einen kleinen Schritt bergab:

```
W = W − Lernrate × Gradient
```

Inferenz (die Prüfung)

Ziel: Mit den fertig gelernten Gewichten neue Fragen beantworten.

1

Forward Pass

Input-Tensor → Hidden Layer → Output-Tensor

```
[300] → [8 Neuronen] → [19.2] (Vorhersage)
```

✔ Das war’s. Wirklich.

Kein Backward Pass, keine Gewichts-Updates. Die Gewichte sind eingefroren — das Netz wendet nur noch an, was es gelernt hat.

⚡ Deshalb ist Inferenz so schnell:

1× vorwärts rechnen statt vorwärts + rückwärts + Update. Genau darum kann dir ChatGPT in Sekunden antworten, obwohl sein Training Monate gedauert hat — du erlebst dort immer nur die Inferenz.

Jetzt trainierst du das Netz selbst!

Das Szenario

Wir simulieren 30 Messungen an unserer roten Pflanze mit Wassermengen von 0 bis 500 ml. Die Natur folgt dabei heimlich dieser Regel:

```
Wachstum = −0.0003 × (Wasser − 250)² + 20
```

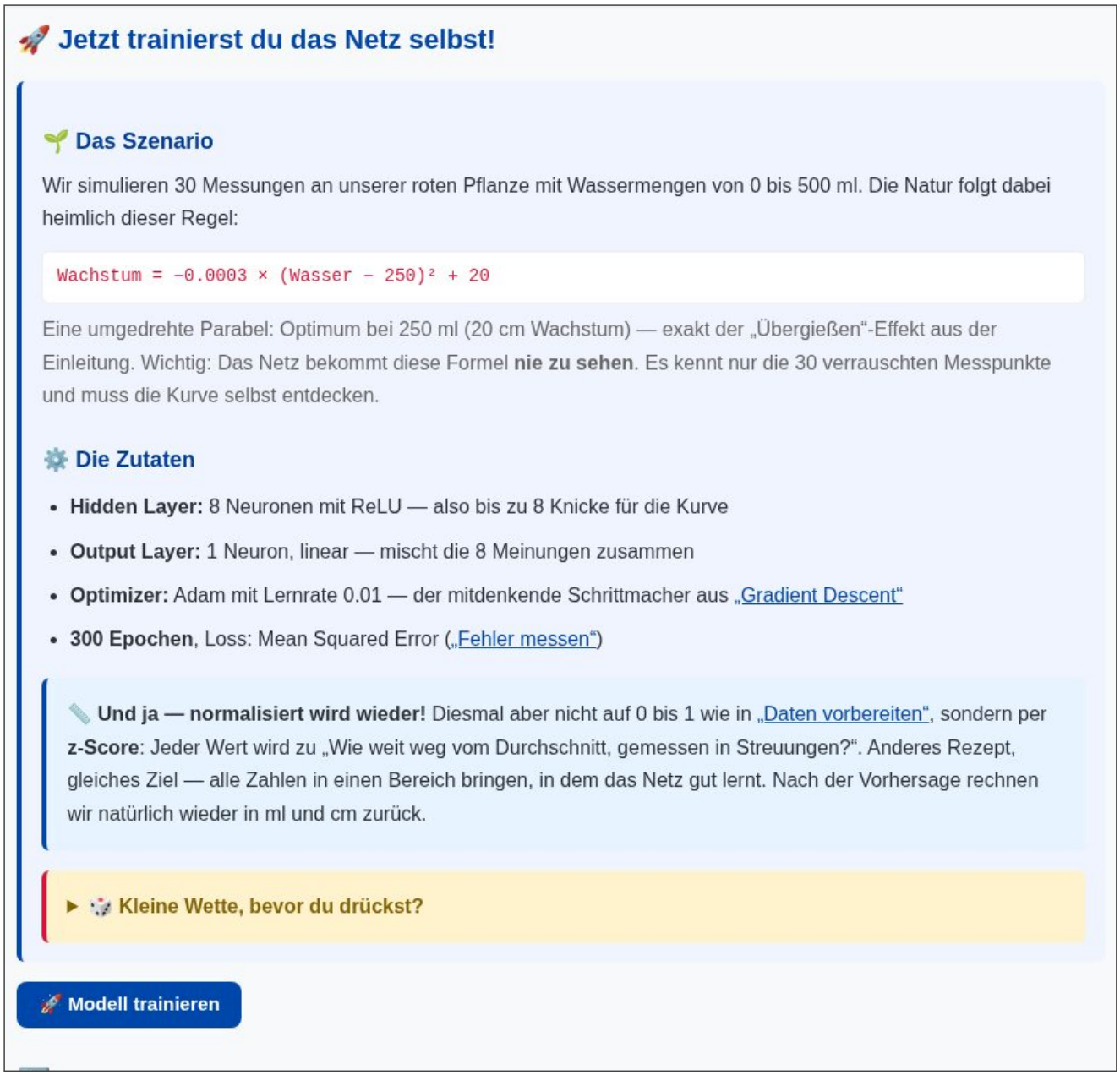
Eine umgedrehte Parabel: Optimum bei 250 ml (20 cm Wachstum) — exakt der „Übergießen“-Effekt aus der Einleitung. Wichtig: Das Netz bekommt diese Formel **nie zu sehen**. Es kennt nur die 30 verrauschten Messpunkte und muss die Kurve selbst entdecken.

Die Zutaten

- **Hidden Layer:** 8 Neuronen mit ReLU — also bis zu 8 Knicke für die Kurve
- **Output Layer:** 1 Neuron, linear — mischt die 8 Meinungen zusammen
- **Optimizer:** Adam mit Lernrate 0.01 — der mitdenkende Schrittmacher aus „Gradient Descent“
- **300 Epochen**, Loss: Mean Squared Error („Fehler messen“)

Am Gerät ausprobieren: herrki.de/140_polynomiale_regression#abschnitt-6

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Und ja — normalisiert wird wieder! Diesmal aber nicht auf 0 bis 1 wie in „Daten vorbereiten", sondern per **z-Score**: Jeder Wert wird zu „Wie weit weg vom Durchschnitt, gemessen in Streuungen?". Anderes Rezept, gleiches Ziel — alle Zahlen in einen Bereich bringen, in dem das Netz gut lernt. Nach der Vorhersage rechnen wir natürlich wieder in ml und cm zurück.

Kleine Wette, bevor du drückst?

Nach dem Training zeigt dir eine Tabelle die **echten gelernten Gewichte** deiner 8 Neuronen. Meine Wette: **Mehrere davon sind negativ.** Warum sollte ein Netz, das Wachstum vorhersagt, „Bremser“-Neuronen brauchen? Weil die Parabel eine rechte Hälfte hat! Ab 250 ml muss irgendetwas im Netz rufen: „Stopp, jetzt wird’s schlechter!“ — und genau das erledigen die Neuronen mit negativem Gewicht. Schau nach dem Training nach, ob ich recht behalte.

0%

Trainingsdaten (erste 10 von 30)
Modell-Struktur
Trainings-Ergebnis
Eigene Vorhersage machen (Inferenz) Gib eine Wassermenge ein — dein frisch trainiertes Netz sagt dir das Pflanzenwachstum voraus. Probier auch mal Extremwerte wie 0 oder 500 ml! **Das ist jetzt Inferenz:** nur Forward Pass, keine Gewichts-Updates!

Glückwunsch! Du verstehst jetzt Hidden Layer und Tensor-Transformationen!

Zurück zur Anwendung

Zurück zur Gießkanne vom Kapitelanfang: „Doppelt so viel Wasser = doppelt so viel Wachstum“ war von Anfang an falsch — dein Netz konnte es bis heute nur nicht anders sagen. Mit den 8 Neuronen der Zwischenschicht und ihren Knicken (ReLU) hat es die Kurve mit dem Optimum in der Mitte selbst gefunden: Mehr hilft, bis es kippt — und danach schadet es. Diese Form steckt in erstaunlich vielen Alltagsfragen: Dünger aufs Feld, Trainingsumfang vor dem Wettkampf, Dosis eines Medikaments, Preis und Absatz, Backtemperatur und Kruste. Wer nur Geraden zeichnen kann, sagt bei all dem „mehr ist besser“ und liegt jenseits des Optimums zuverlässig daneben.

Nimm das als Faustregel fürs Lesen von Diagrammen mit: Ein linearer Trend ist immer auch eine Behauptung über den Rand der Achse — und die stimmt fast nie. Warum eine Zwischenschicht *ohne* Aktivierungsfunktion trotzdem nur wieder eine einzige Gerade ergibt und warum genau dieser eine Knick alles ändert, nimmt sich das nächste Kapitel vor: „Warum eine Gerade nicht reicht (Aktivierungsfunktionen)“.

Zum Weiterlesen

- **Jenseits der Geraden:** James, Witten, Hastie, Tibshirani, *An Introduction to Statistical Learning*, Kapitel 7 „Moving Beyond Linearity“ (PDF gratis) — statlearning.com — Polynome, Splines und Co.: wie die Statistik Kurven baut, bevor ein Netz es tut.
- **Die Kehrseite von „mehr Kurve“:** Google Machine Learning Crash Course, *Overfitting* — developers.google.com/machine-learning/crash-course (developers.google.com/machine-learning/crash-course/overfitting) — was passiert, wenn dein Netz mehr Knicke bekommt, als die Daten hergeben.
- **Das Standardwerk dahinter:** Christopher M. Bishop, *Pattern Recognition and Machine Learning*, 2006, Abschnitt 1.1 — PDF gratis bei Microsoft Research (18 MB) (microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf) — das Buch beginnt mit genau diesem Beispiel: eine Kurve durch Punkte legen und dabei zu weit gehen.
- **Auf HerrKI:** 145 — Warum eine Gerade nicht reicht (der ReLU-Knick im Einzelteil) und 240 — Overfitting & Regularisierung (wenn das Netz die Punkte auswendig lernt)

Kurzer Check – mit Lösung

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

Frage 1: Warum kann dein Netz jetzt Kurven lernen, während es vorher nur Geraden konnte?

- Weil es länger trainiert wurde.
- ✔ **Hidden Layer + ReLU: viele geknickte Geradenstücke setzen sich zu einer Kurve zusammen.**
- Weil der Computer schneller geworden ist.

Der ganze Zauber ist „Kurven-Lego": Jedes ReLU-Neuron liefert einen Knick, die Ausgabeschicht mischt sie zusammen. Keine Magie — clever kombinierte Knicke.

Frage 2: Dein Hidden Layer hat 8 ReLU-Neuronen. Wie viele Knickstellen kann die gelernte Kurve höchstens haben?

- Genau eine.
- ✔ **Bis zu 8 — mehr Neuronen bedeuten eine feiner geformte Kurve.**
- Unendlich viele, egal wie klein das Netz ist.

Pro ReLU-Neuron ein möglicher Knick. Deshalb: komplexere Formen → mehr Neuronen oder mehr Schichten. Die umgedrehte Gieß-Parabel bekommt ihr Dach aus genau diesen Knicken.

Frage 3: Was passiert bei der Inferenz (deiner Vorhersage unten) NICHT mehr?

- Das Rechnen mit Tensoren.
- Die ReLU-Aktivierung.
- ✔ **Gewichts-Updates — es wird nur noch vorwärts durchgerechnet.**

Training = raten + messen + nachjustieren. Inferenz = nur noch einsetzen (Forward Pass). Die gelernten Gewichts-Tensoren bleiben dabei unangetastet.

Das nimmst du mit

- **Der Hidden Layer macht aus 1 Zahl 8 Meinungen.** Eine Zwischenschicht zwischen Eingang und Ausgang — das ist der ganze Architektur-Unterschied zur Geraden.
- **Ohne ReLU gäbe es keine Kurve.** Erst der Knick lässt aus gestapelten Geraden etwas anderes als eine Gerade werden.
- **Pro ReLU-Neuron ein möglicher Knick.** 8 Neuronen → bis zu 8 Knickstellen; eine feinere Form braucht mehr Neuronen oder mehr Schichten.
- **Acht gleiche Formeln, acht verschiedene Rollen.** Zufälliger Start und ein eigener Gradient pro Neuron führen ganz von selbst zur Spezialisierung — niemand teilt das ein.
- **Training und Inferenz sind zwei Betriebsarten.** Beim Vorhersagen wird nur noch vorwärts gerechnet; die gelernten Gewichts-Tensoren bleiben unangetastet.

Weiter geht's: Du hast eben erlebt, wie 8 ReLU-Knicke gemeinsam eine Parabel formen. In „Warum eine Gerade nicht reicht“ zerlegst du diesen Trick Stück für Stück — und danach wartet in „Exponentielle Regression“ dein erstes richtig tiefes Netz auf dich.

So zitierst du dieses Kapitel

HerrKI: Wenn die Gerade sich biegen muss (Polynomiale Regression). herrki.de/140_polynomiale_regression, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.