

“Machine Learning Language” Workshop
Bildungswerk Südhessen
27.6.2024

Roland Leißa



Outline

1. Über euch und uns
2. Maschinelles Lernen
3. Vorwärts/Rückwärts
4. Workshop

Outline

1. Über euch und uns
2. Maschinelles Lernen
3. Vorwärts/Rückwärts
4. Workshop

Ich bin...



Prof. Dr. Roland Leißa
Lehrstuhl „Programmiersprachen und Compilerdesign“



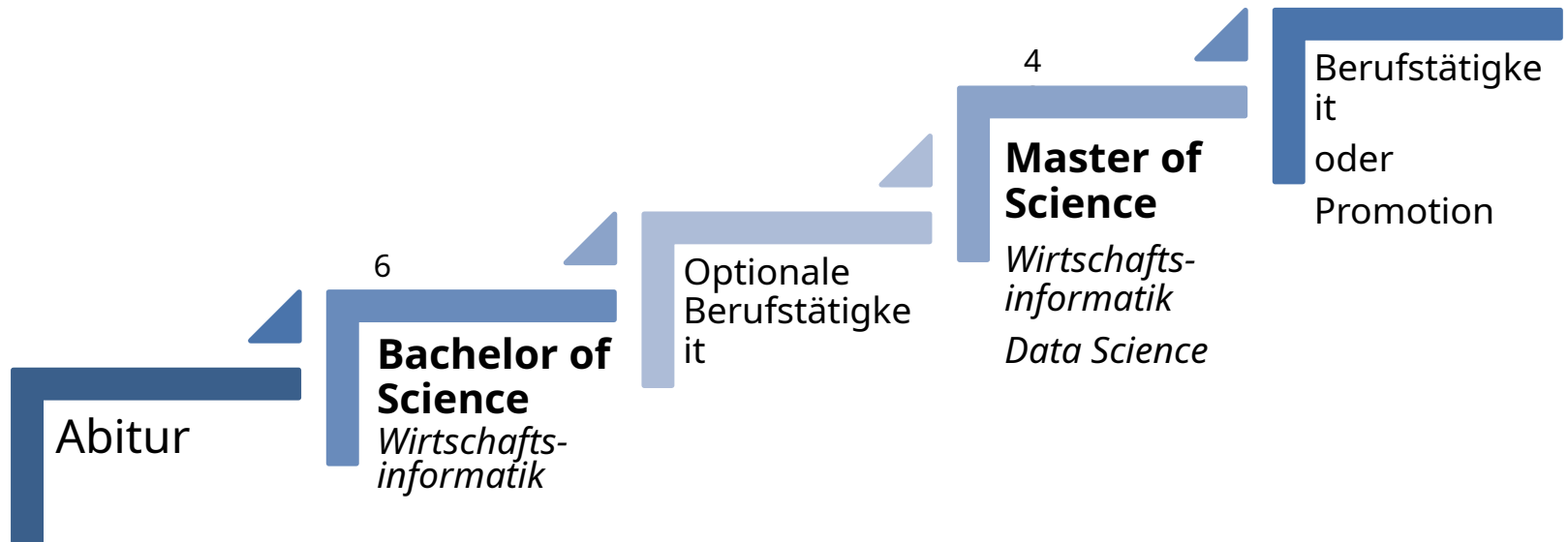
Universität Mannheim (im Schloss!)



Unsere Büros (nicht im Schloss!)

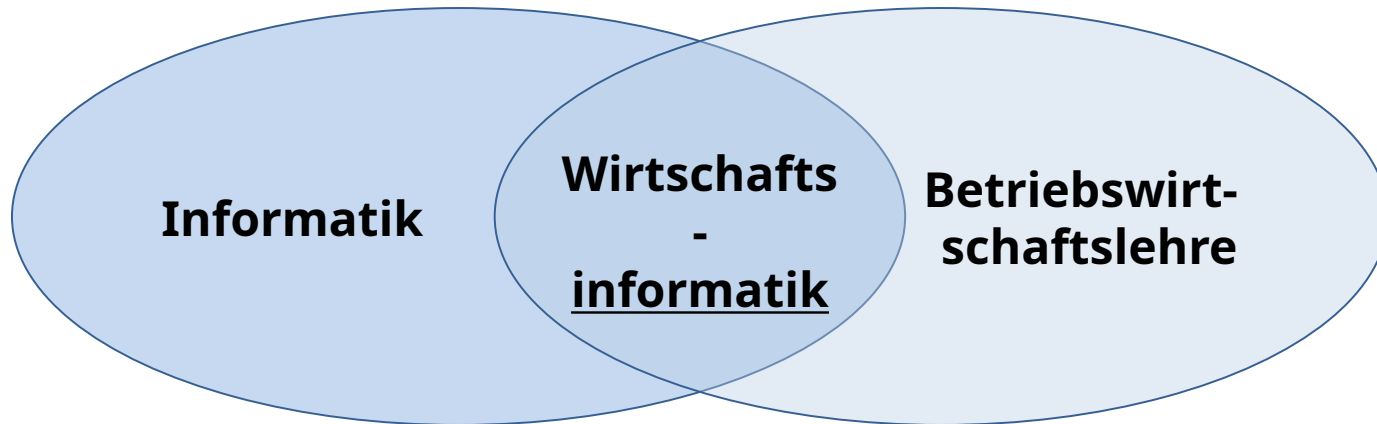


Studienangebot



Wirtschaftsinformatik in Mannheim

Mathematik



Wahlbereich

Wifo studiert – und dann?



McKinsey&Company



Startups

Mittel-
stand

Forschun
g

Absolventum Universität Mannheim  Netzwerk

Warum in Mannheim studieren?



Wir bieten:

- guter Ruf (national und international) der Professoren und Professorinnen der Uni Mannheim
- Kontakte in die Wirtschaft
- Vertiefung in BWL, **Informatik** oder Wifo
- Internationale Ausrichtung
- Große Auswahl an Partneruniversitäten
- Studieren im Schloss
- Nähe zu Metropolregion Rhein-Neckar
- Freizeitangebot

Und ihr seid...

Outline

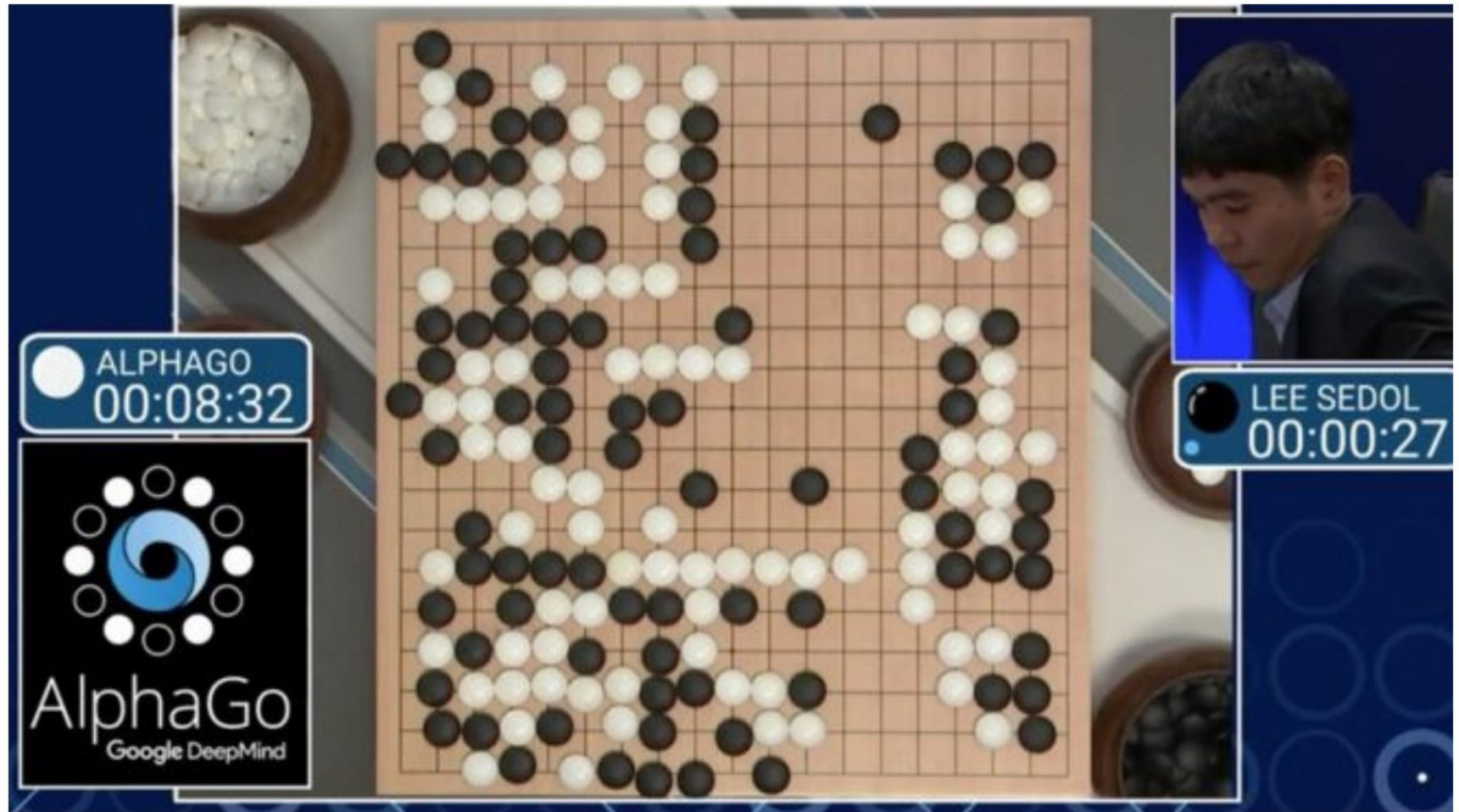
1. Über euch und uns
2. Maschinelles Lernen
3. Vorwärts/Rückwärts
4. Workshop

What is machine learning?

Key question: How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?

- Easy to write good programs for certain tasks (e.g., shortest path)
- Hard to write good programs for other tasks (e.g., spam detection)
- Machine learning generally comprises
 - ▶ A **task** T (e.g., playing Checkers)
 - ▶ A **performance metric** P (e.g., percent of games won)
 - ▶ Training **experience** E (e.g., playing games)
 - ▶ Goal: machine learns to reliably improve performance P at task T , following experience E

In 2016, DeepMind's AlphaGo beat Go master Lee Se-dol



Let's learn!

- Roland distinguishes good and bad triples
- Here is a good triple

2 4 8

- Ask me about triples being good or bad; then tell the rule I use to distinguish (you are allowed to guess only once!)
- Could you write a program that could guess such rules?

Klassifikation

- Diese Aufgabe war eine **Klassifikations**aufgabe
 - Gegeben: **Eingabe** („2 4 8“)
 - Gesucht: **Klasse** („gut“)
 - Endlich viele Klassen
- Andere Beispiele
 - E-Mail -> Spam? („ja“ / „nein“)
 - Go-Feldbelegung -> Nächster Zug?
 - Text -> Nächstes „Wort“?

Wo befindet sich Karben?

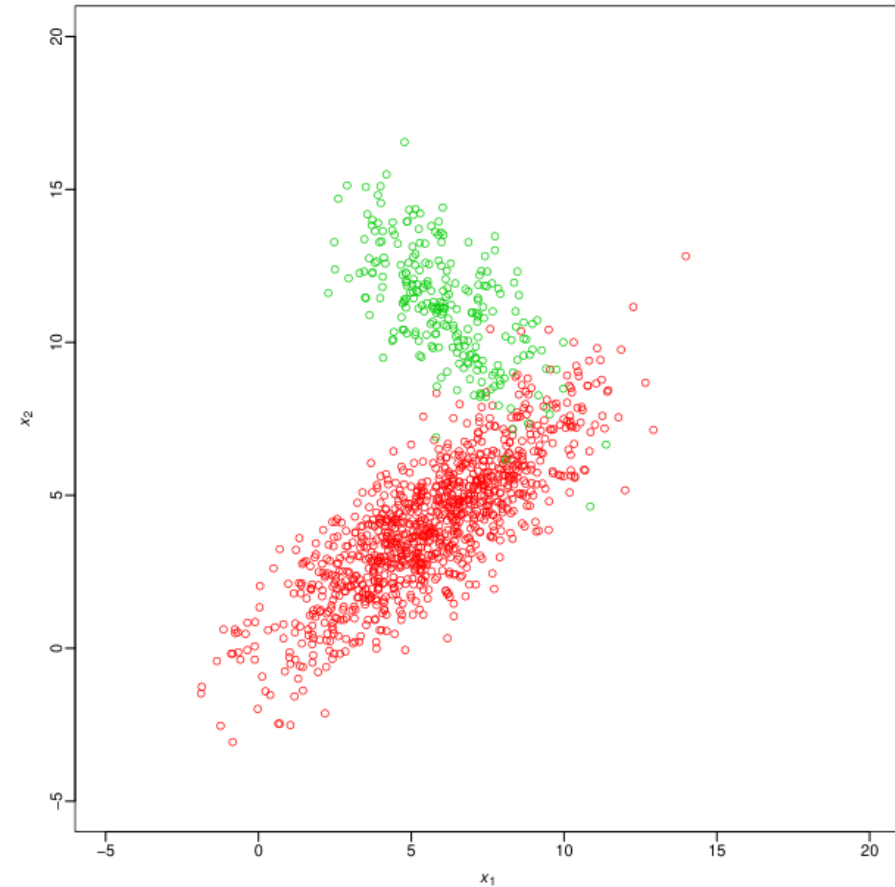
Karben ist eine Stadt in Hessen, Deutschland. Sie liegt im Wetteraukreis, etwa 15 Kilometer nordöstlich von Frankfurt am Main. Karben besteht aus mehreren Stadtteilen und ist in der Region Rhein-Main-Gebiet verortet, was ihr eine gute Anbindung an Frankfurt und andere umliegende Städte ermöglicht.

Wie lernt man?

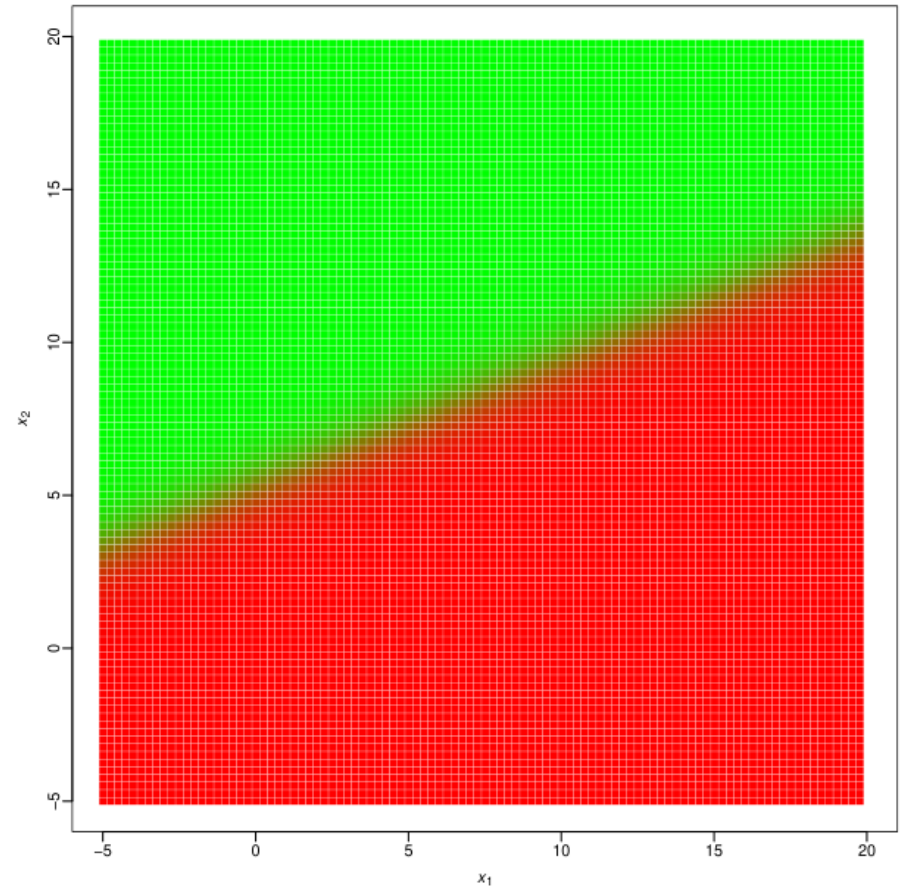
- Was denn überhaupt?
 - Eine Funktion $f(\text{Eingabe}) = \text{Klasse}$
 - **Machinelles Lernen lernt Funktionen**
- **Beispiel** = Eingabe-Ausgabe Paar
 - „\$\$\$MONEY_RICH_YOU!!!“ -> Spam
- **Überwachtes Lernen**
 - Lernen von (vielen) Beispielen = **Trainingsdaten**
- Und wie?
 - So dass f auf den Trainingsdaten gut funktioniert
 - Eigentlich: so dass f auf zukünftigen Eingaben gut funktioniert

Beispiel: Logistische Regression

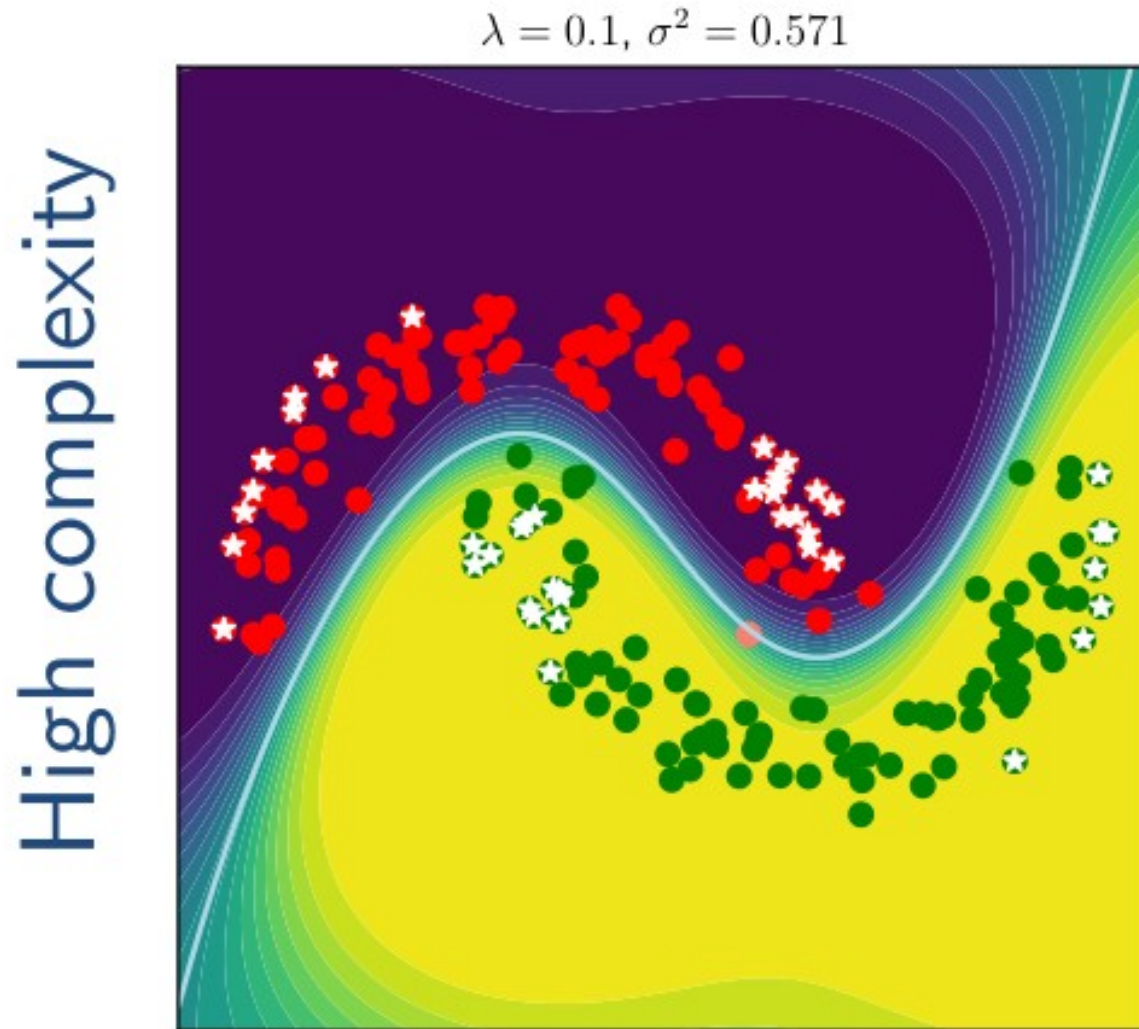
Data



Prediction



Beispiel: L1VM + Logistische Regression



Outline

1. Über euch und uns
2. Maschinelles Lernen
3. Vorwärts/Rückwärts
4. Workshop

Modelle

- Wir haben: Trainingsdaten
- Wir wollen Funktion $f(\text{Eingabe}) \rightarrow \text{Klasse}$
 - So eine Funktion heißt **Modell**
- **Modellklasse** = Menge der in Frage kommenden Funktionen
- Lernen (für heute): Wähle eine der Funktionen aus

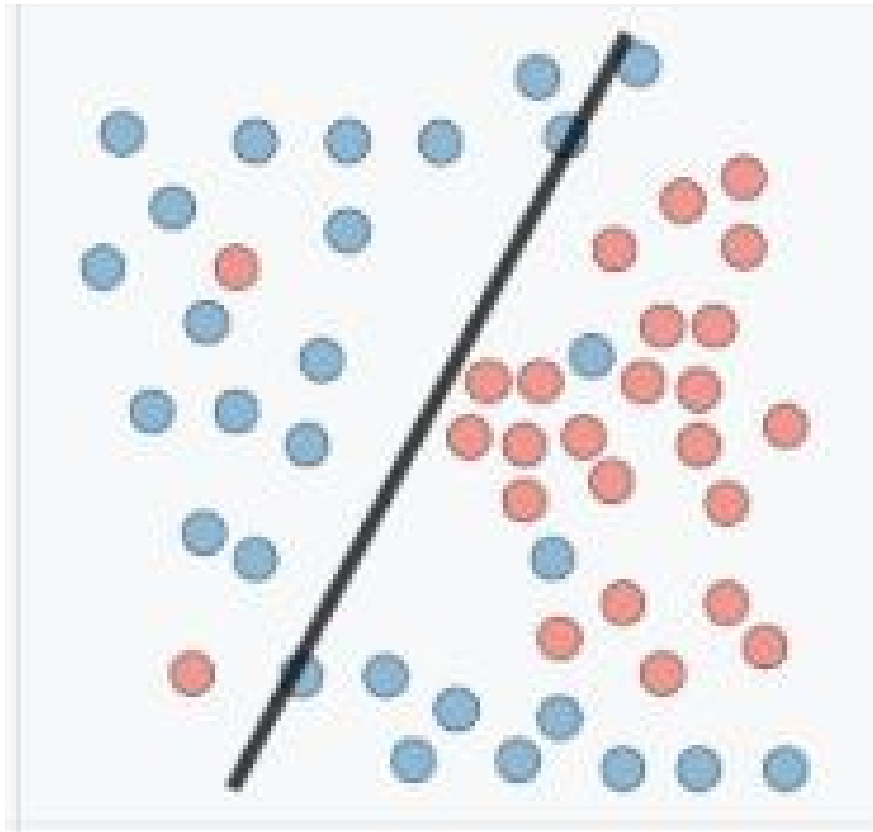
Parametrisches Modell

- Eine Modellklasse ist **parametrisch**, wenn jede Funktion der Modellklasse durch eine feste Anzahl **Parameter** beschrieben wird
- Beispiel: $f(\text{Einkommen}) \rightarrow \text{reich} / \text{arm?}$
 - Modellklasse mit Parameter $w \in \mathbb{R}$:
 $f(\text{Einkommen}) = \text{„reich“}$ wenn $\text{Einkommen} > w$ sonst „arm“
 - Wir schreiben: $f_w(x)$
- **Diese Modellklasse ist unendlich groß!**
 - Jede reelle Zahl entspricht einer anderen Funktion
 - Das ist bei den meisten Modellklassen so
 - Anzahl der Parameter: 10, 100, 1000, ..., ~2 Billionen
- Lernen = Wähle eine Parameterbelegung \hat{w} aus

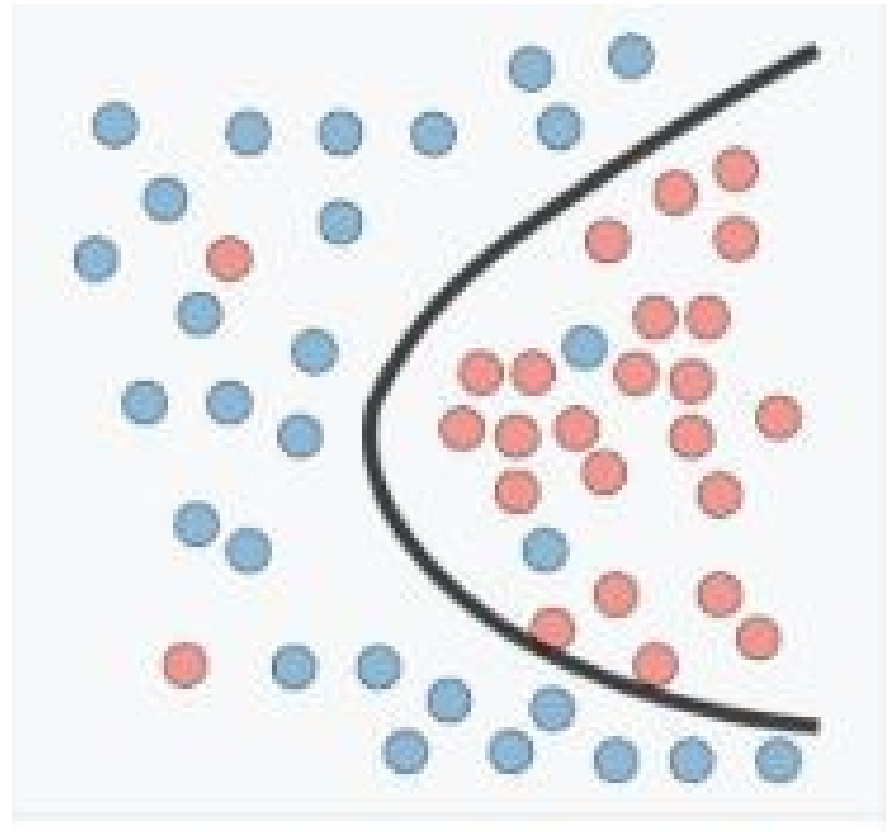
Kostenbasiertes Lernen

- Wie wählt man eine Parameterbelegung aus?
- Wir verwenden: **Kostenfunktion** misst Modellgüte
 - Klein -> Modell gut auf Trainingsdaten
 - Groß -> Modell schlecht auf Trainingsdaten
 - Für gegebenes w : Für jedes Trainingsbeispiel (x_i, y_i)
 1. Berechne $\hat{y} = f_w(x_i)$
 2. Bewerte den Unterschied (**loss**) zwischen y_i (richtig) und \hat{y} (momentane Vorhersage)
 - Kostenfunktion z.B. „durchschnittlicher Unterschied“
- **Kostenbasiertes Lernen**
 1. Erstelle eine Kostenfunktion $J(w)$
 2. Minimiere die Kostenfunktion $J(w)$ in Bezug auf w
 3. So gefundenes w = Ausgewählte Parameterbelegung

Beispiel (links blau, rechts rot)



Modell w1
 $J(w1) = \text{"14 Fehler"}$
(schlecht)



Modell w2
 $J(w2) = \text{"4 Fehler"}$
(gut)

Wie minimiert man die Kostenfunktion?

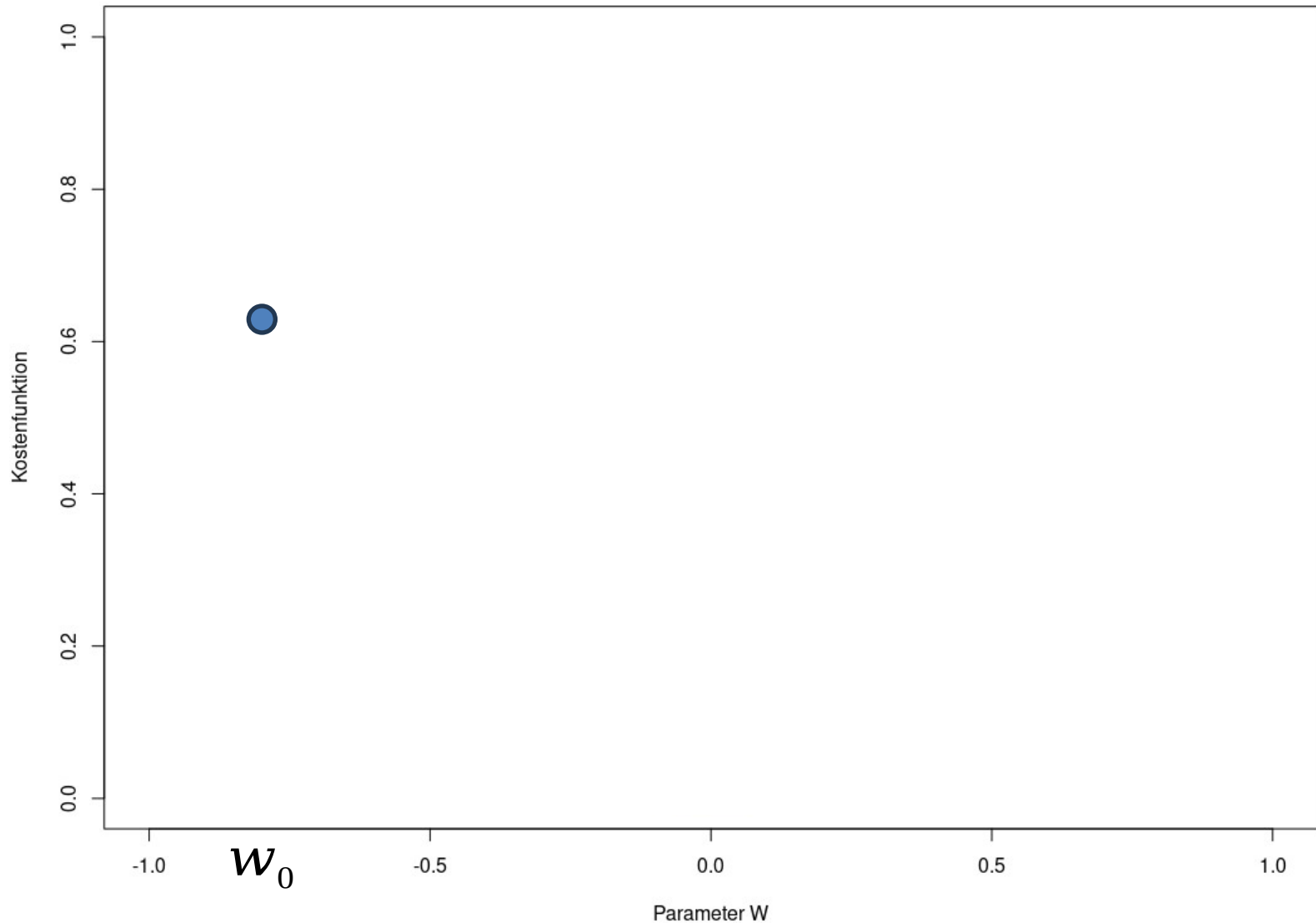
Ein **iteratives Lernverfahren**:

1. Starte mit einer „beliebigen“ Belegung

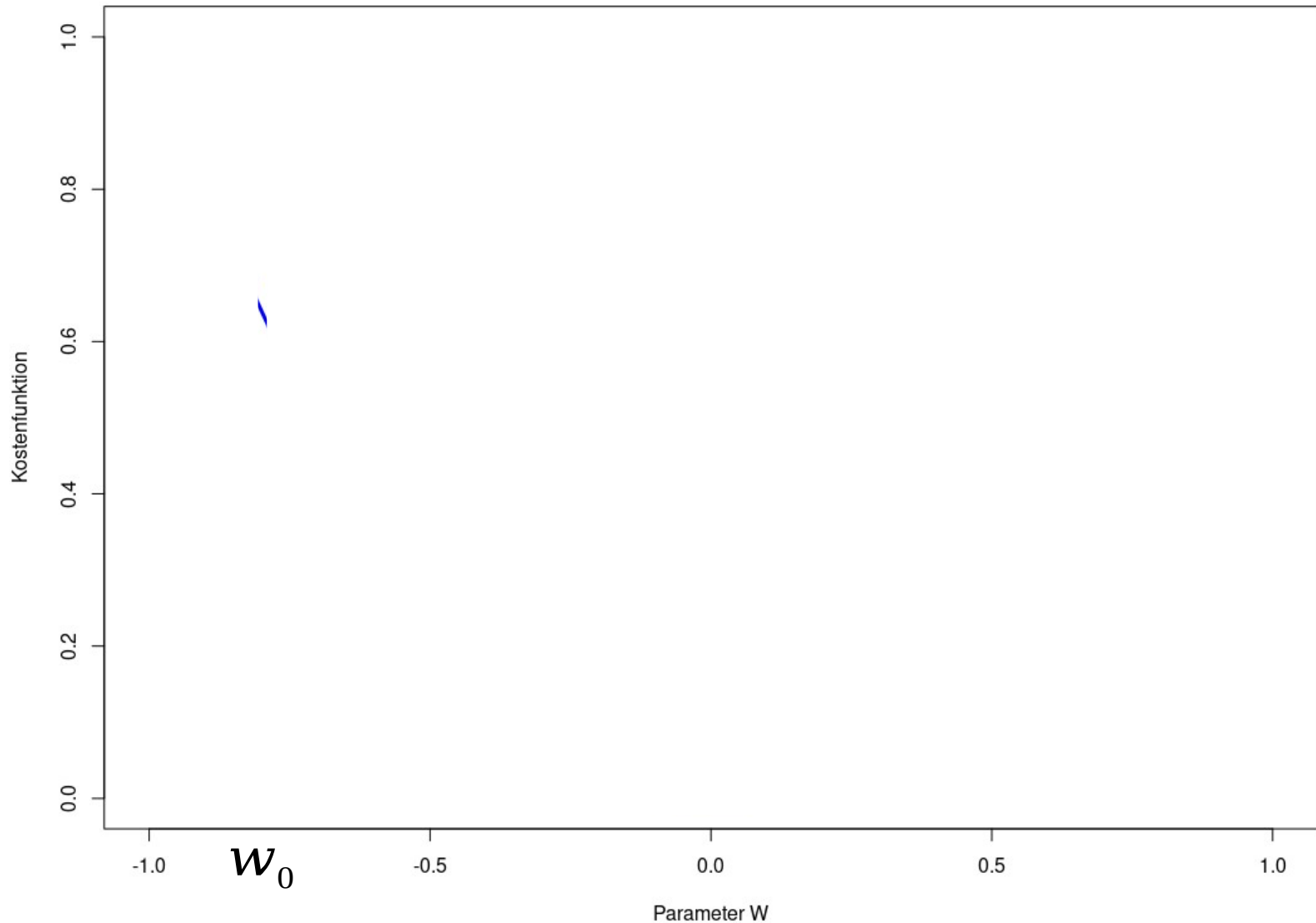
$$w = w_0$$

2. Bestimme die Kostenfunktion $J(w)$
3. Ändere w so dass die Kostenfunktion nach der Änderung etwas kleiner wird
4. Gehe zu Schritt 2

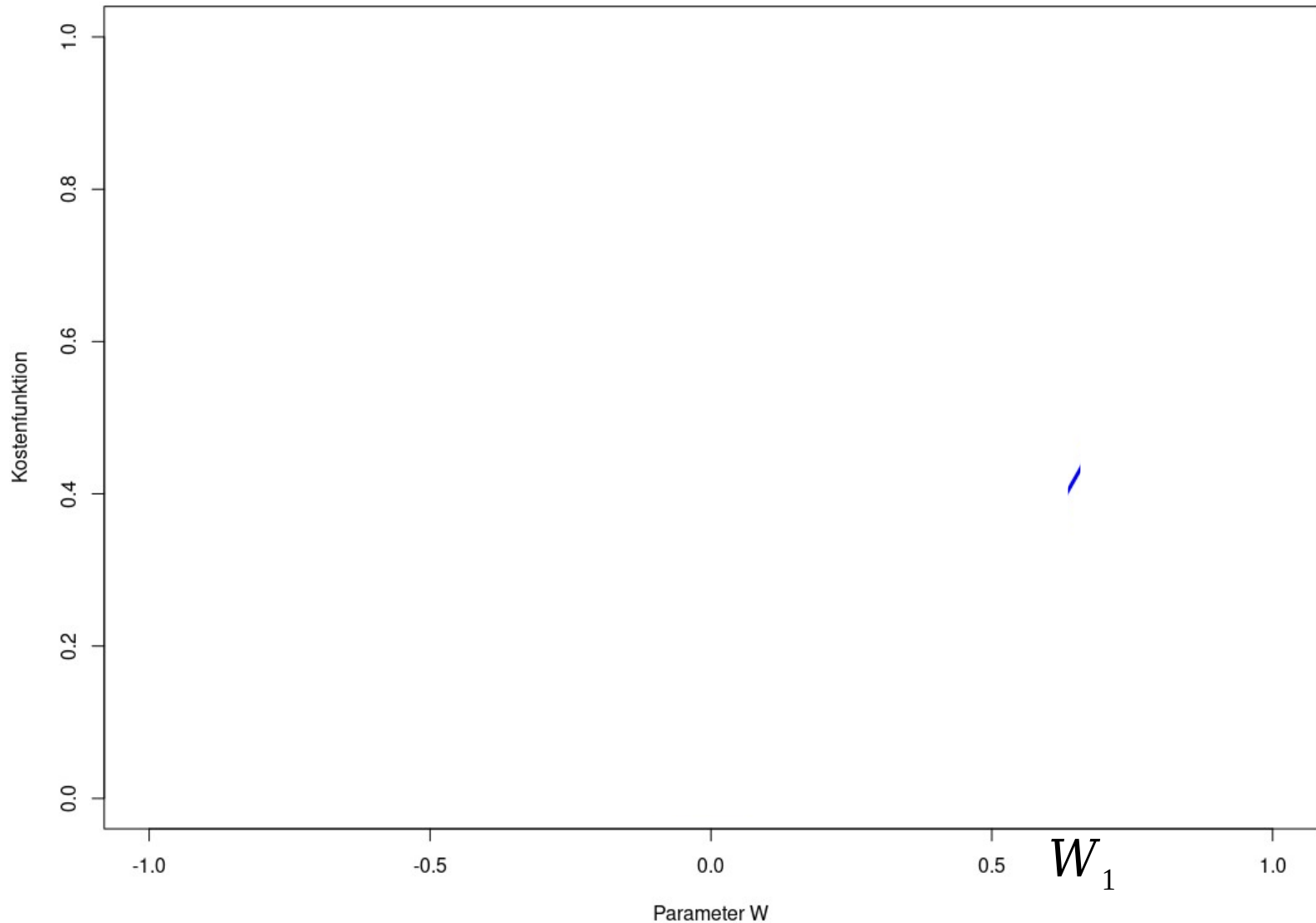
Beispiel (w_0) – Wo liegt w_1 ? UNKLAR!



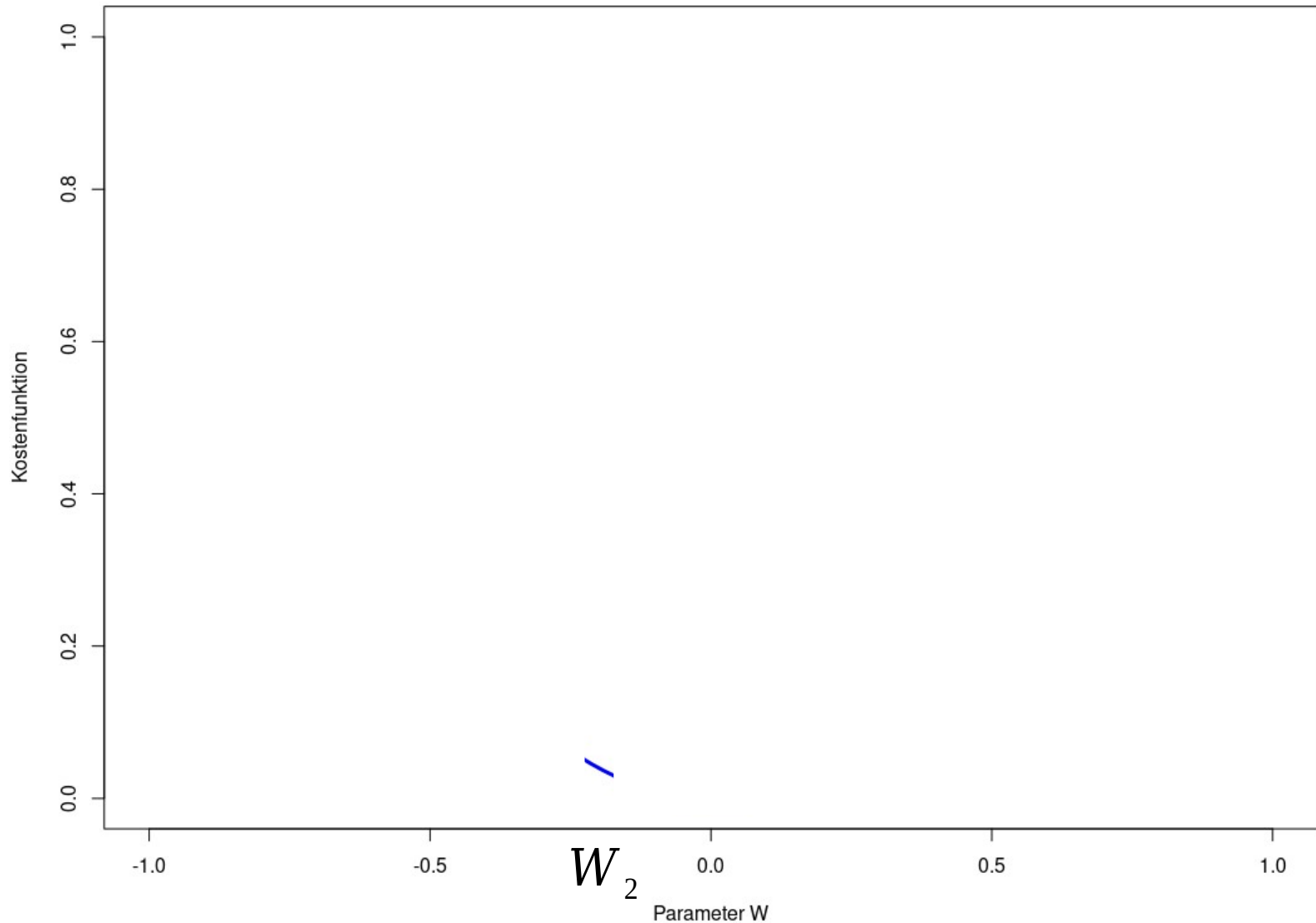
Beispiel (w_0) – Wo liegt w_1 ? RECHTS!



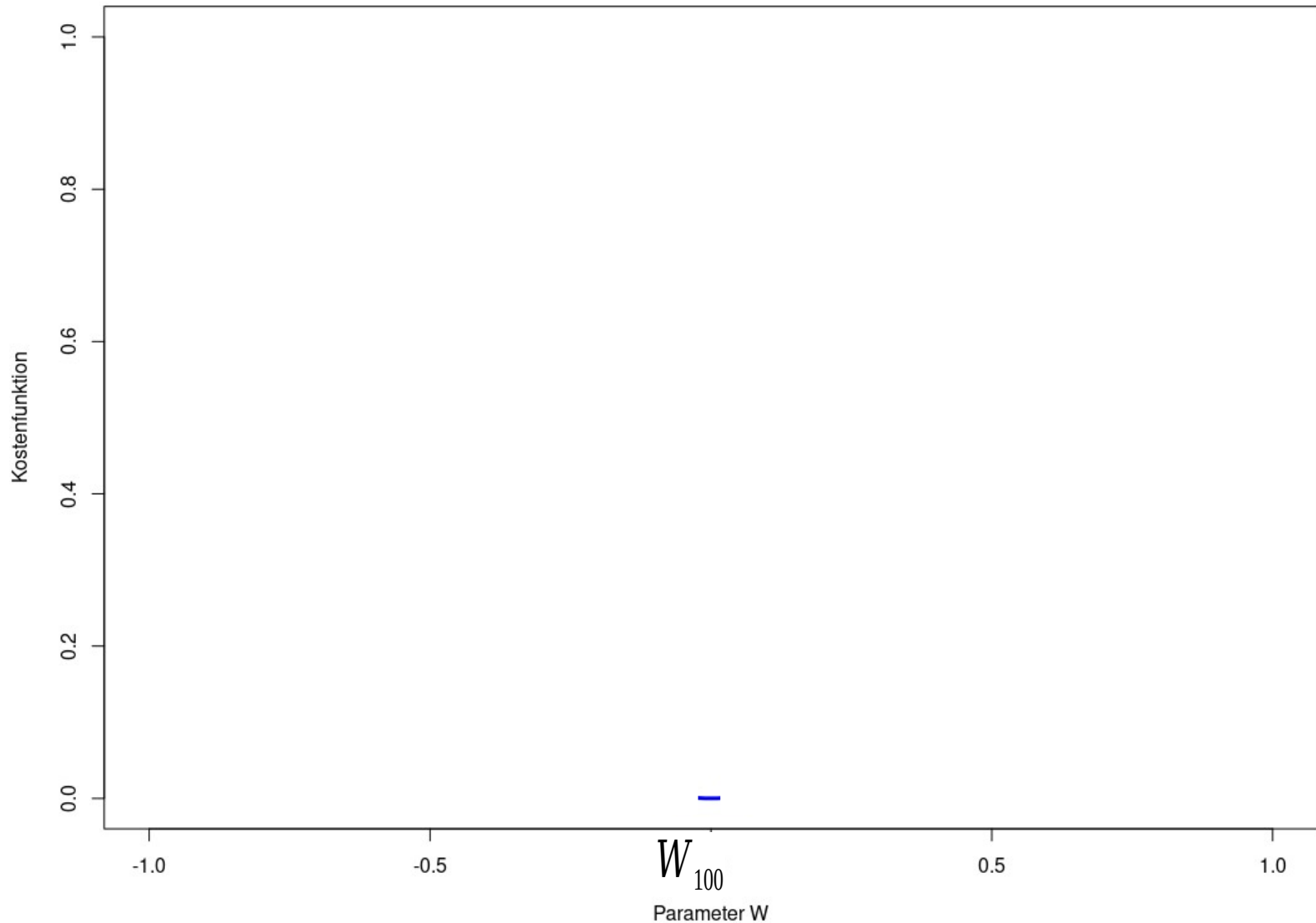
Beispiel (w_0) – Wo liegt w_1 ? LINKS!



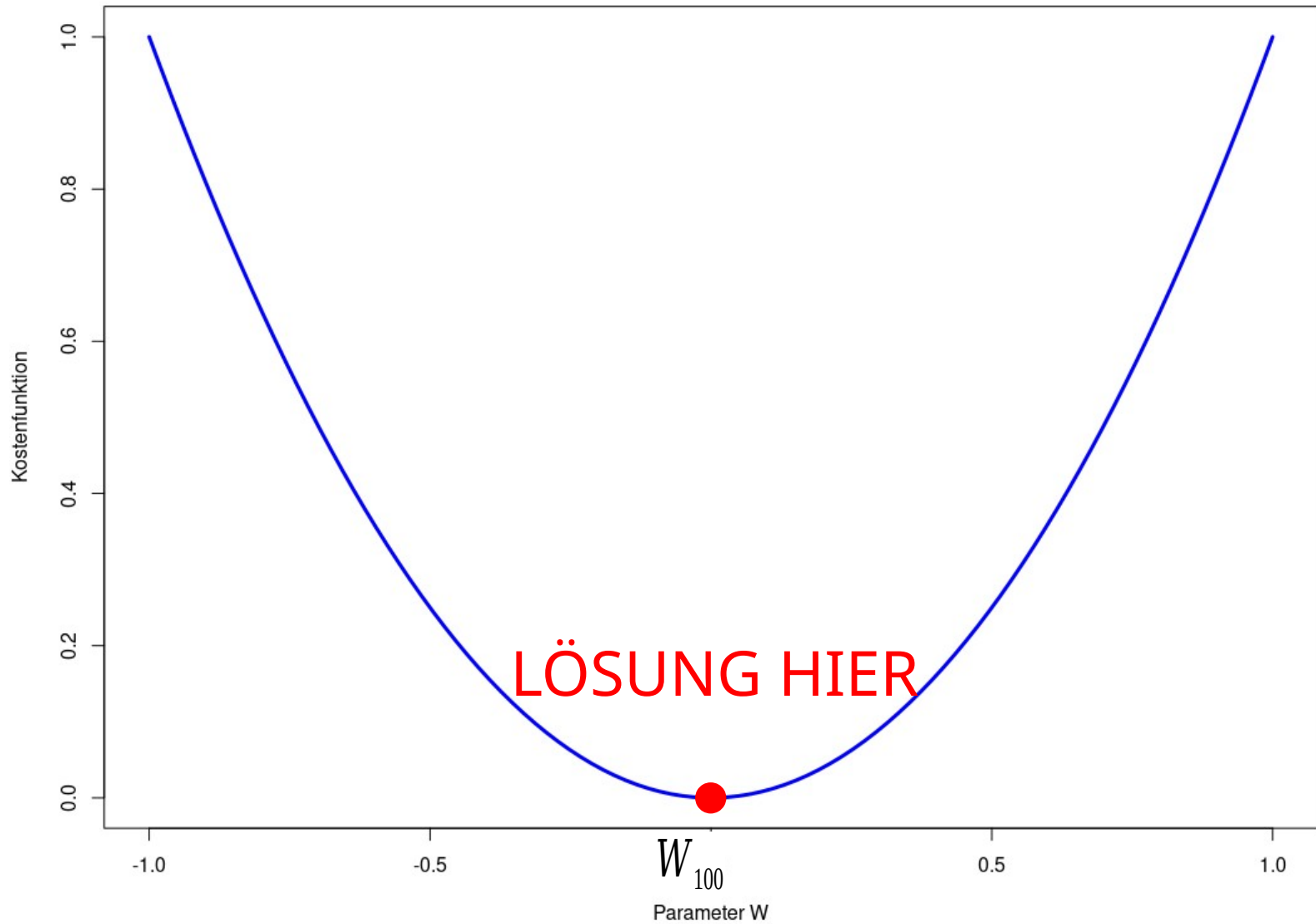
Beispiel (w_2) – Wo liegt w_3 ? RECHTS!



Beispiel (w_{100}) – Wo liegt w_{101} ? BLEIBEN!



Beispiel (w_{100})



Wie minimiert man die Kostenfunktion?

Gradientenbasierter Abstieg:

1. Starte mit einer „beliebigen“ Belegung

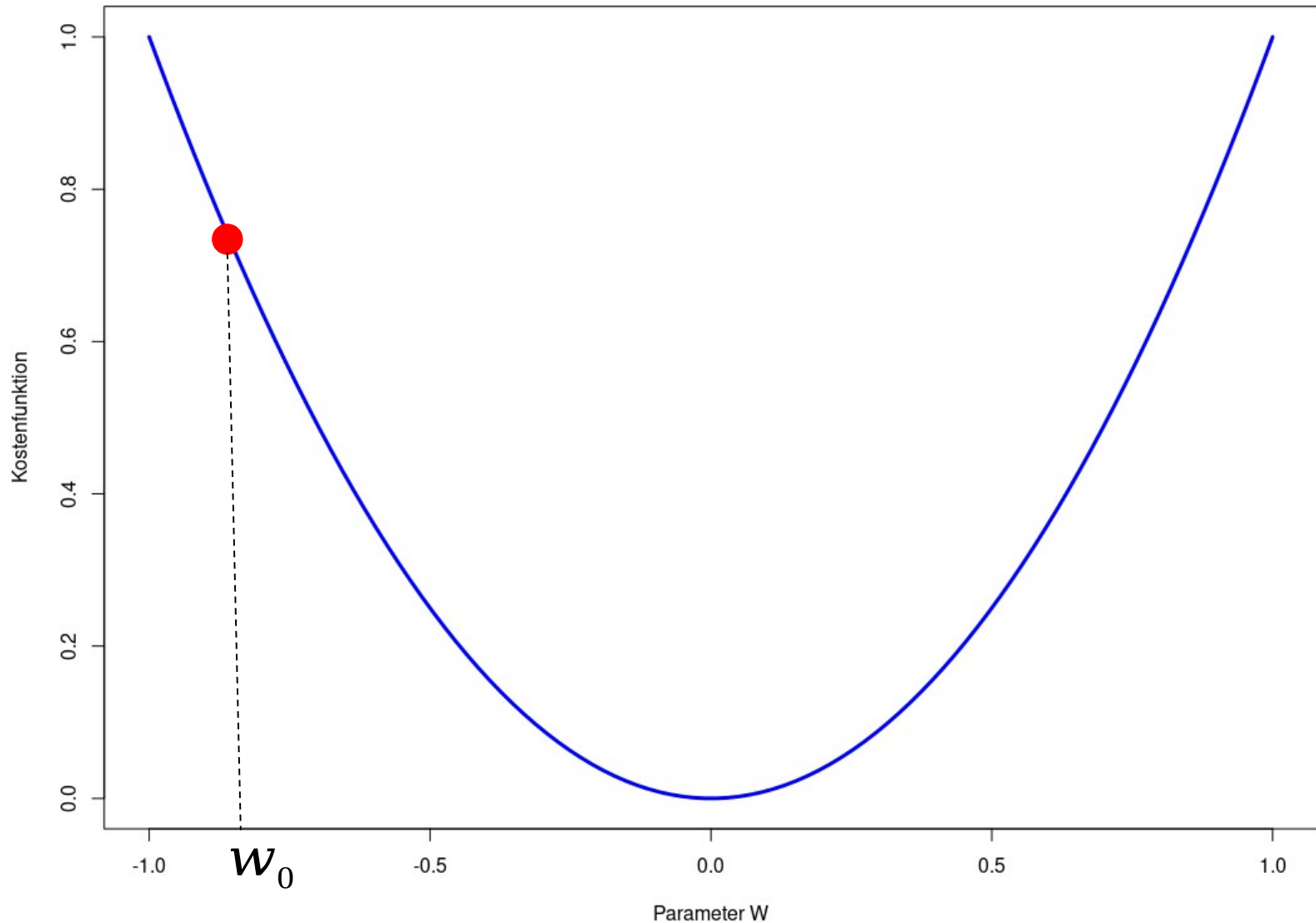
$$w = w_0$$

2. Bestimme die Kostenfunktion $J(w)$
3. Bestimme die Richtung des stärksten Anstiegs $\delta = J'(w)$ der Kostenfunktion (**Gradient**) und gehe ein bisschen in die andere Richtung

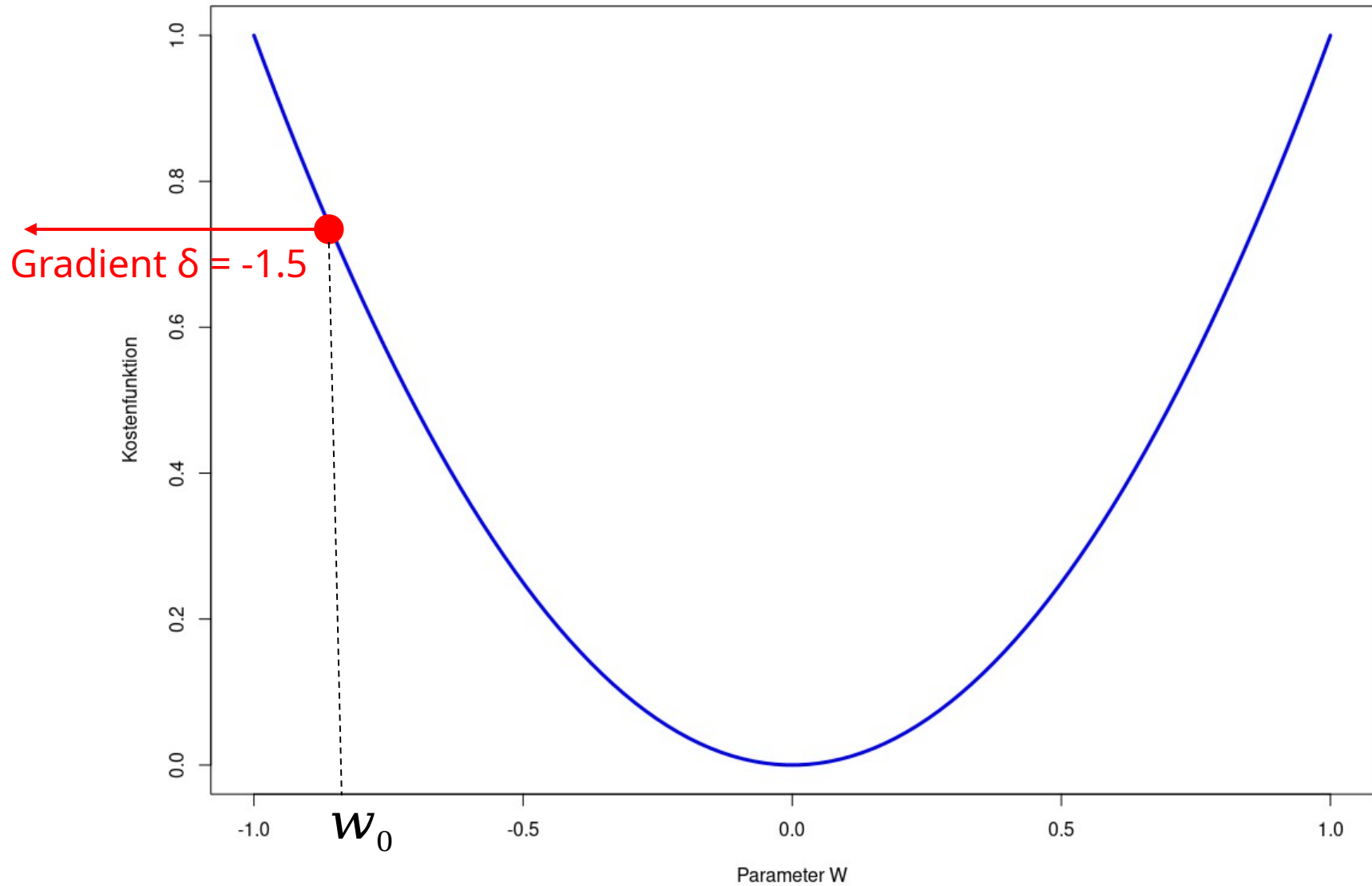
$$w \leftarrow w - \epsilon \delta$$

4. Gehe zu Schritt 2

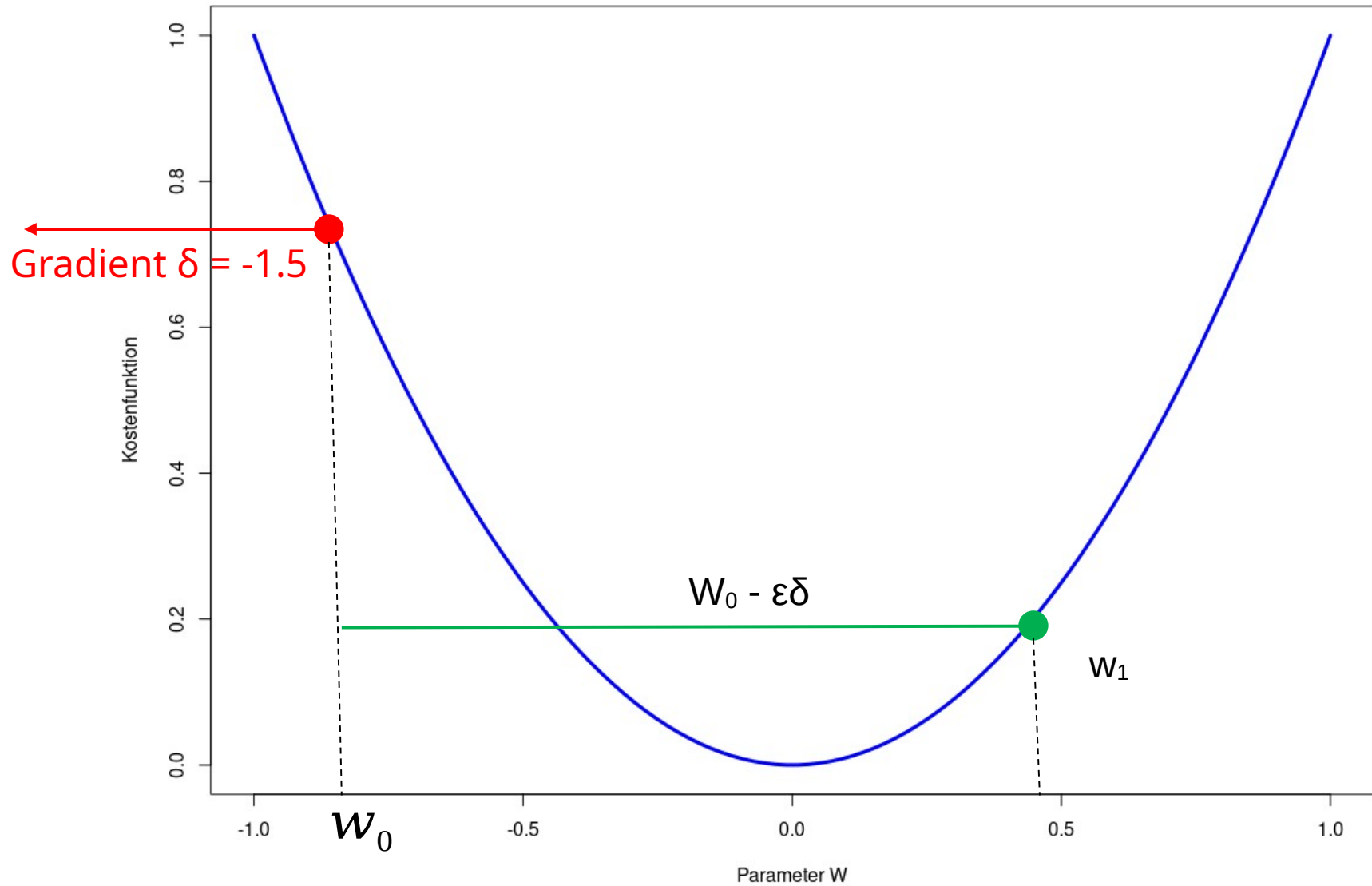
Beispiel: Kostenfunktion und W



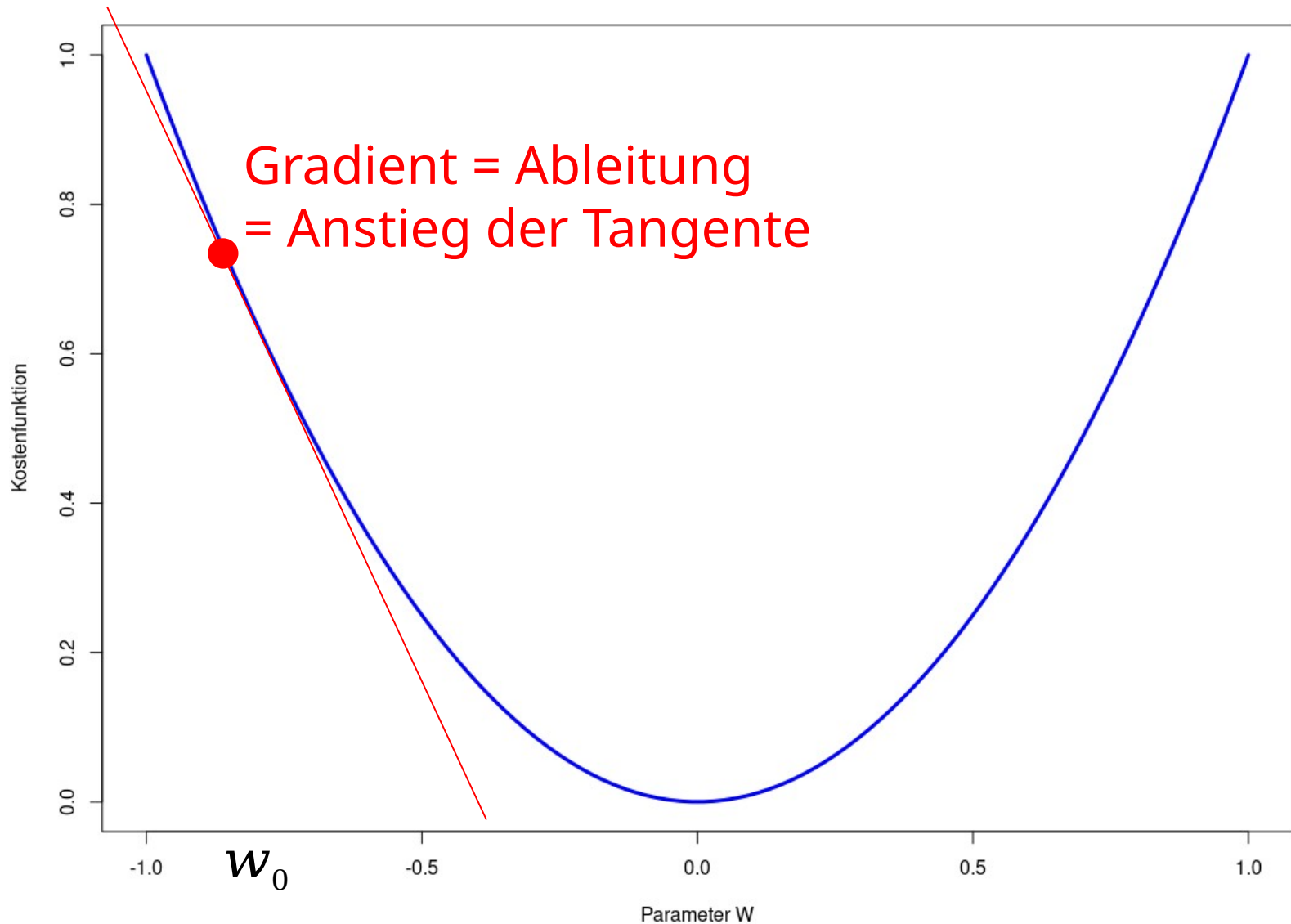
Beispiel: Gradient



Beispiel: Schritt



*Interpretation Gradient (ein Parameter)



*Gradient (viele Parameter)

- 1 Parameter
 - Gradient = Ableitung
- Viele Parameter
 - Eine partielle Ableitung pro Parameter
 - Gradient = Vektor aller partiellen Ableitungen
 - Beispiel:

$$J(w_1, w_2) = 5w_1 + w_2^2$$

$$\frac{\partial}{\partial w_1} J = 5$$

$$\frac{\partial}{\partial w_2} J = 2w_2$$

$$\partial = \begin{pmatrix} 5 \\ 2w_2 \end{pmatrix}$$

Outline

1. Über euch und uns
2. Maschinelles Lernen
3. Vorwärts/Rückwärts
4. Workshop

Von der Theorie zur Praxis

- In der Praxis
 - Modelle haben viele Parameter (groß)
 - Modelle sind komplex (teuer)
 - Und wie bestimmt man den Gradienten?
- Schön wäre:
 1. Implementiere ein Programm, dass $J(w)$ berechnet
 2. Gradientenberechnung passiert automatisch
 3. Effiziente Ausführung passiert automatisch
- Geht das?
 - **Ja, mit Informatik!**

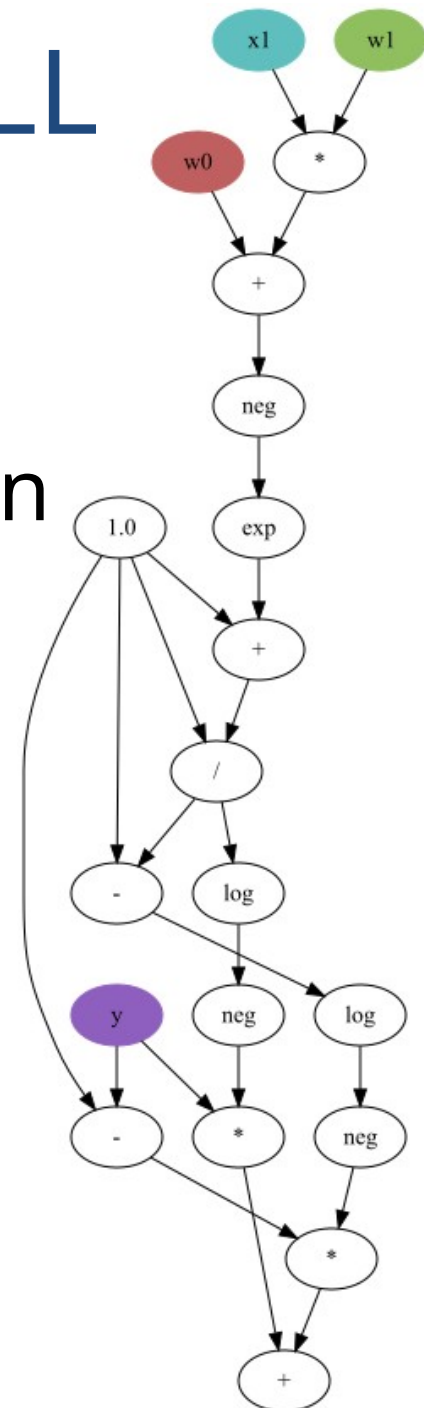
Workshop: MLL

2. Vorwärts: Implementiere $J(w)$
 - Dazu entwickeln wir eine **domänenspezifische Sprache (DSL)** namens MLL
 - Programme in MLL erstellen **Rechengraphen**
 - Ausführung via Interpretation
3. Rückwärts: Gradientenberechnung
 - Mittels “Backpropagation”
 - Transformiert den Rechengraph (fügt weitere Operationen dazu)
4. Schneller: effiziente Ausführung
 - Wir kompilieren das MLL-Programm nach LLVM
 - Und dann in nativen Maschinencode
5. Finale: Maschinelles Lernen
 - Wir lernen ein Modell (logistische Regression) mit MLL

Ein Rechengraph in MLL

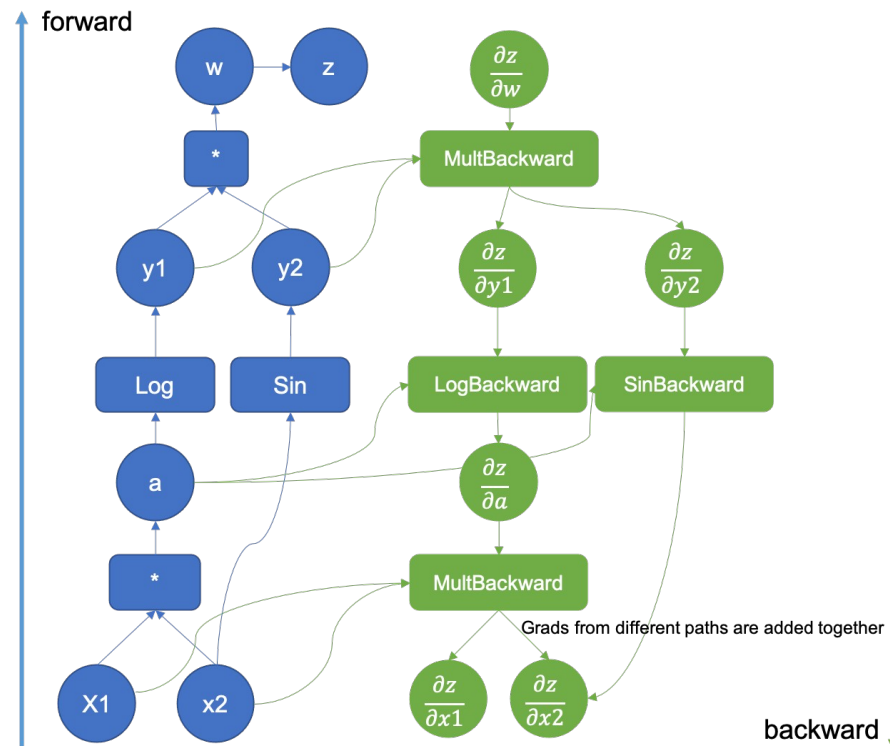
Rechengraph

- Blatt = Eingabe
- Innerer Knoten = Operation
- Ausgabe ganz unten
- Beispiel: logistische Regression, 1D, vorwärts



Systemunterstützung für ML

- Genauso funktioniert das auch in realen ML-Systemen
 - Implementiere nur “vorwärts”
 - Automatische Gradientenberechnung, Optimierung, Kompilierung (z.B. auf die GPU) uvm.
- Beispiel: Rechengraph in [Pytorch](#)



Ein Pytorch-Programm

```
# define model with one hidden layer
model = torch.nn.Sequential(
    torch.nn.Linear(dim_in, dim_hidden),
    torch.nn.ReLU(),
    torch.nn.Linear(dim_hidden, dim_out),
)

# define loss function (mean squared error)
loss_fn = torch.nn.MSELoss()

# pick optimizer (Adam)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

# run for 500 epochs
for t in range(500):
    y_pred = model(X)           # forward: model output (X = examples)
    loss = loss_fn(y_pred, y)   # forward: loss (y = labels)
    model.zero_grad()          # clear old gradients
    loss.backward()             # backward
    optimizer.step()            # update parameters
```

Viel Spaß!