

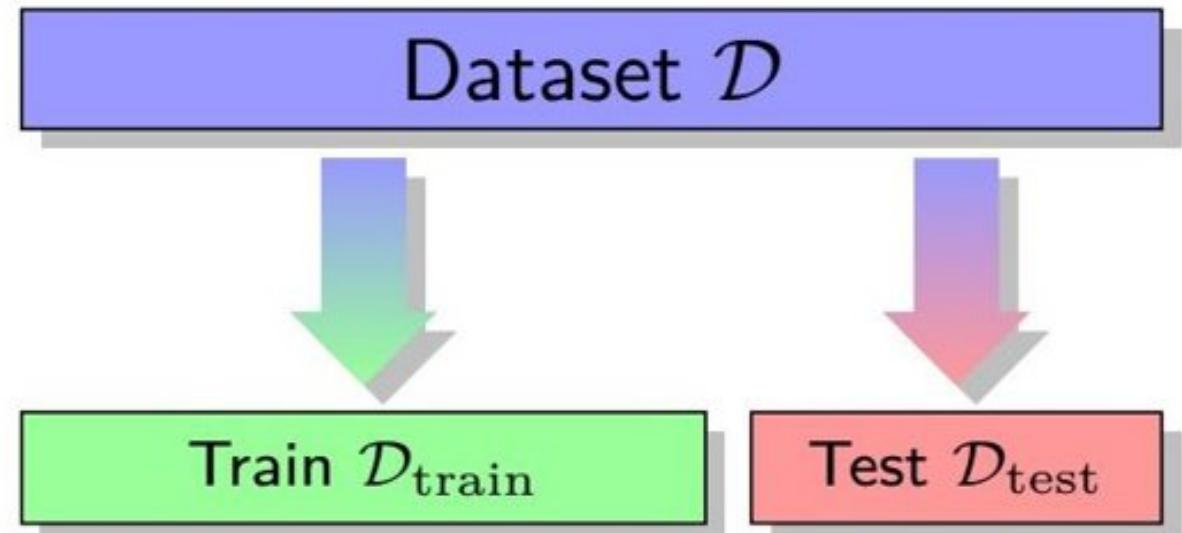
Einführung in neuronale Netze

Bausteine neuronaler Netzwerke

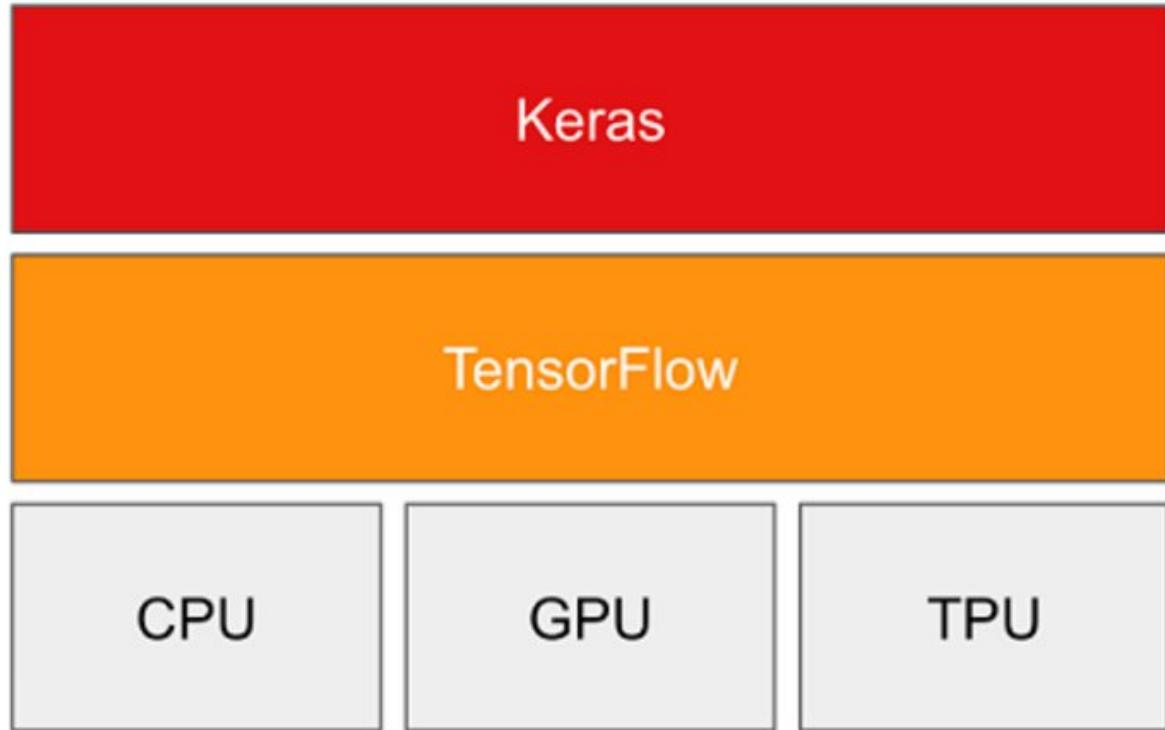
Training-Test Split

Der Trainingsdatensatz wird aufgeteilt in einen Trainingsdatensatz und einen Testdatensatz

- Auf dem Trainingsdatensatz wird trainiert
- Nach dem Training wird auf dem Testdatensatz getestet z.B. unter Verwendung der Performance Metrik
- Durch Verwendung des Testdatensatzes wird festgestellt ob das Modell auf neue Beispiele generalisiert



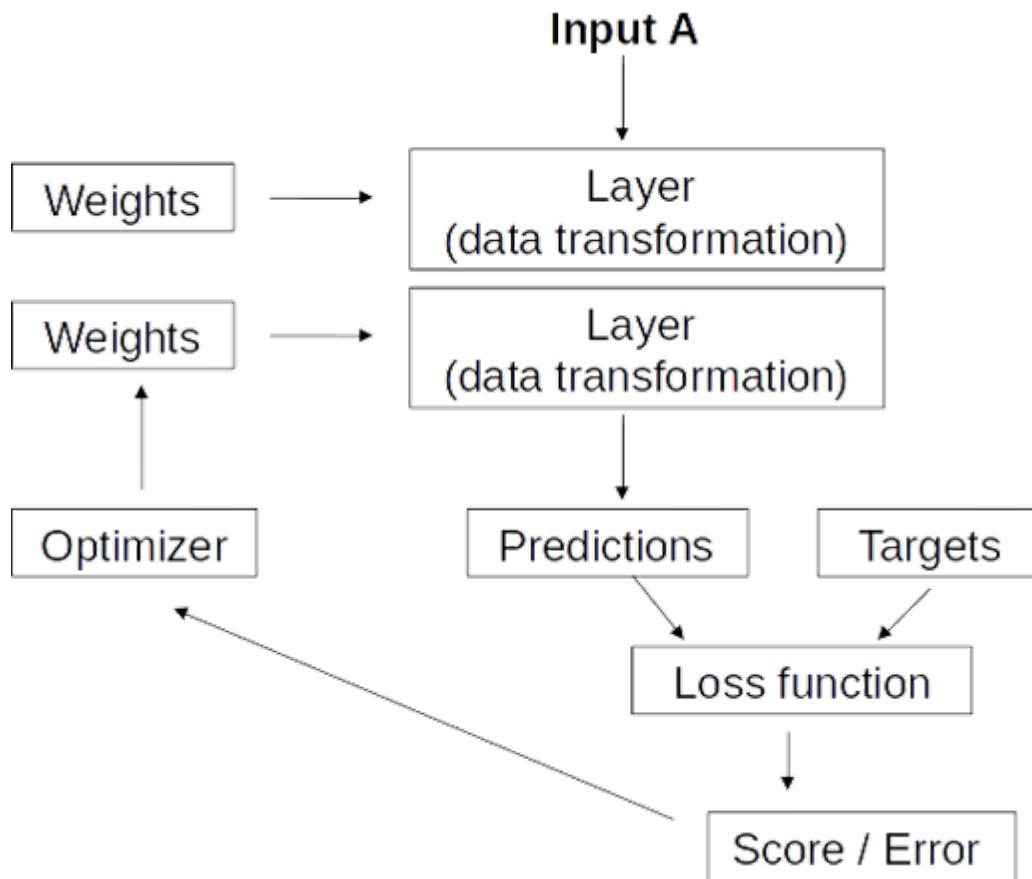
Die Keras API



High Level API: Deep Learning Entwicklung,
Layers,

Backend: Autodiff, Optimierer, Loss-
Funktionen, ...

Bausteine eines neuronalen Netzwerks



- Modell - Gegeben ein Input soll ein Output erzeugt werden
- Loss - Wie formulieren wir mathematisch, was wir erreichen wollen?
- Optimierer - Welcher Algorithmus minimiert unseren Loss?
- Metrik - Welches menschlich verständliche Maß verwenden wir, um die Performance unseres Modells einzuschätzen?

Adapted from 'Deep Learning with Python' by Francois Chollet.



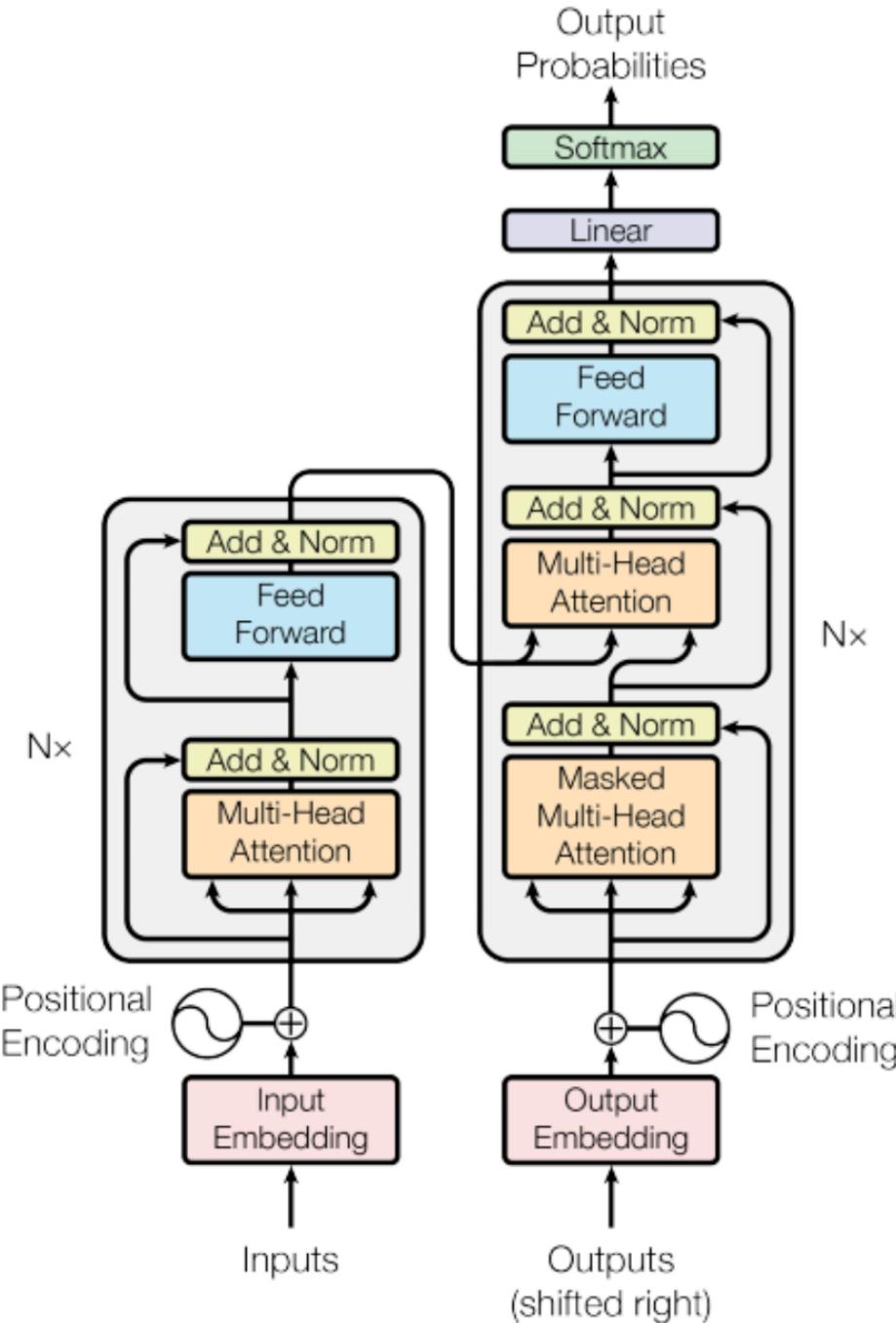
Unterschied - Loss und Metrik

Warum optimieren wir nicht nach der Metrik?

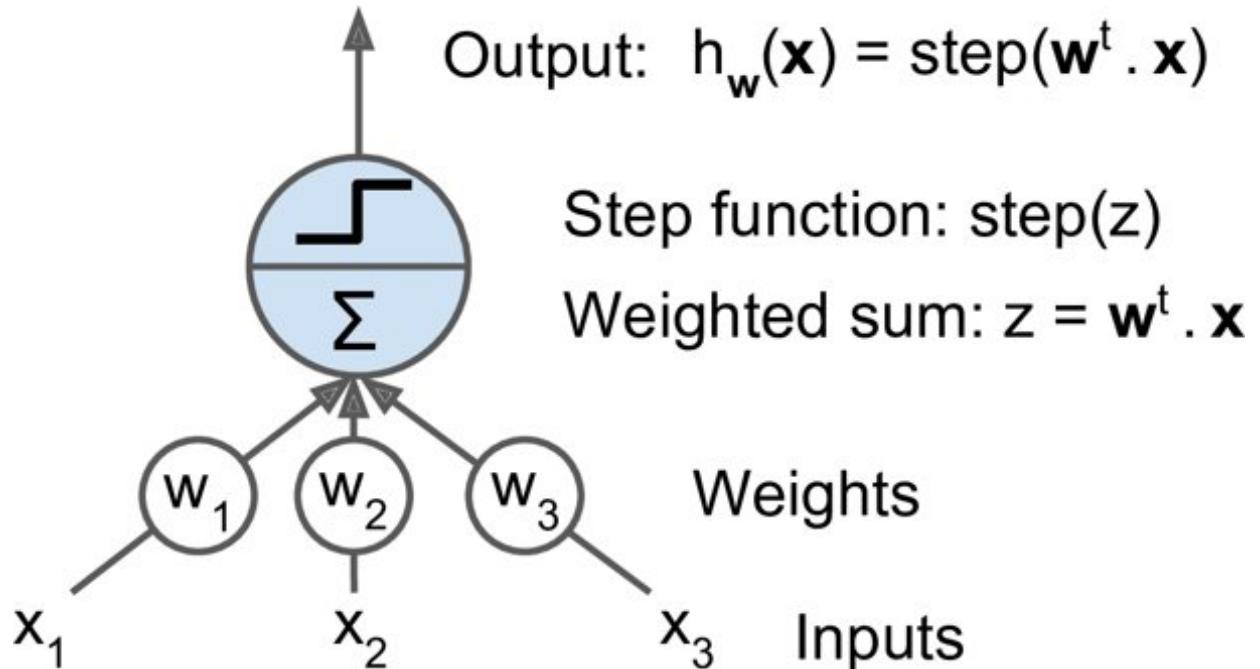
- Metrik nicht unbedingt ableitbar
 - Metrik ändert sich oft initial wenig
 - Metrik kann natürliche Plateaus erreichen
- Gute Lossfunktionen weisen diese Eigenschaften nicht auf!



Die Architektur – Layers



Bausteine eines neuronalen Netzwerks



- Deep Learning Modelle gibt es viele
- Grundbaustein ist immer das Neuron
- Äquivalent zu einer linearen Regression plus nicht-lineare Transformation

Source: Hands-on Machine Learning, Aurelion Geron

Lineare Regression:

$$f_{\mathbf{w}, b}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = \sum_{i=1}^n w_i x_i + b$$

Neuron als linearer Regressor

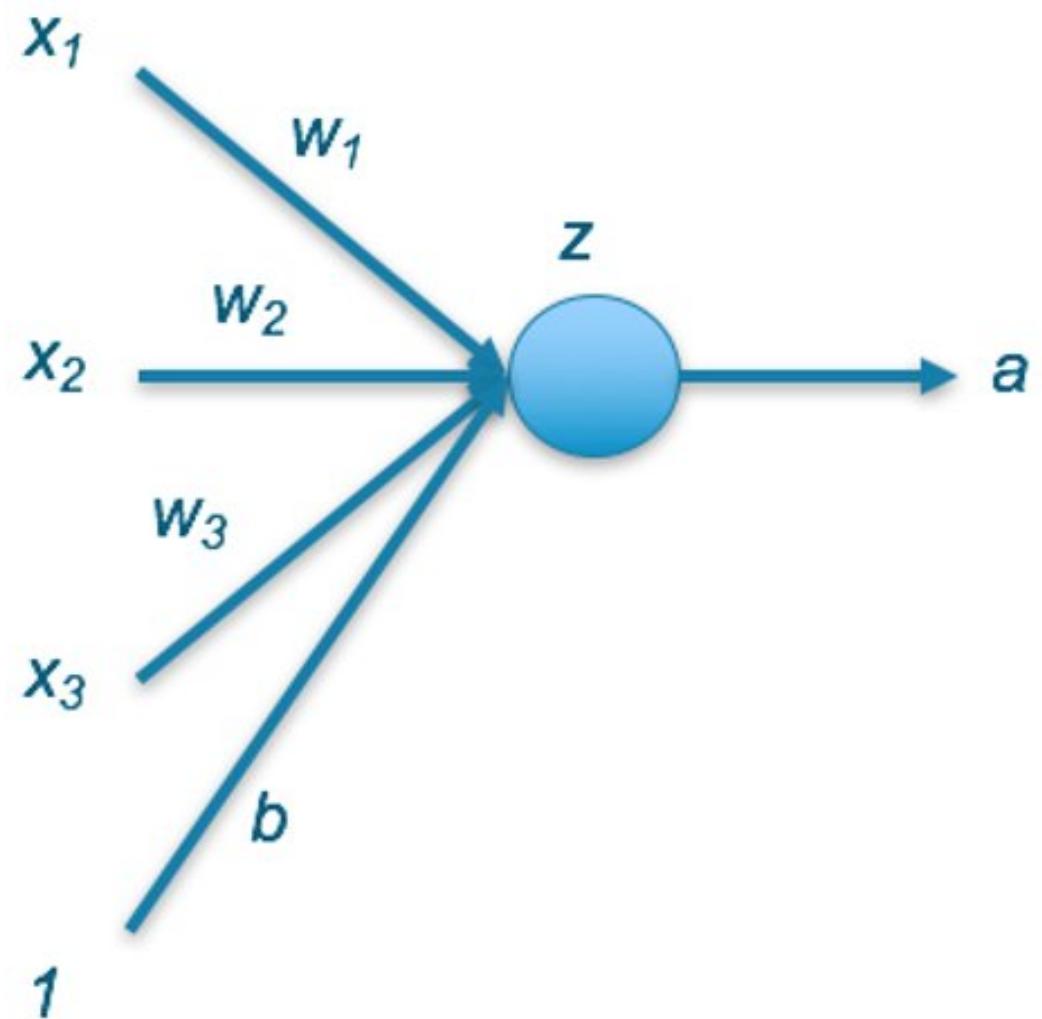
Neurone:

- Linear-affine Abbildung:

$$z = \sum_{i=1}^3 w_i x_i + b$$

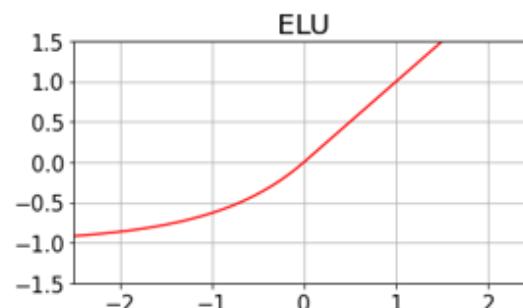
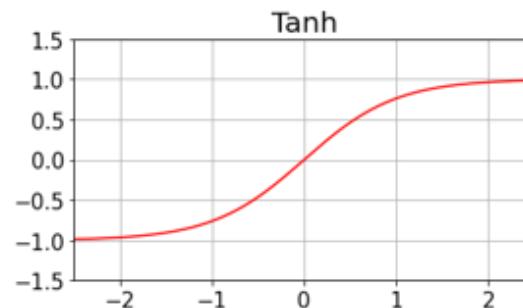
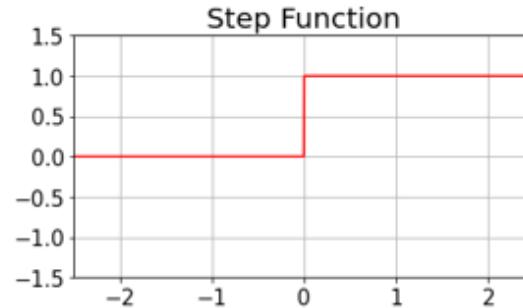
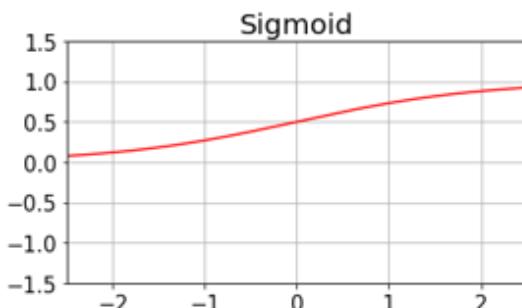
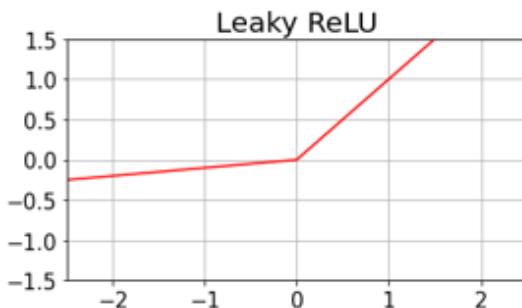
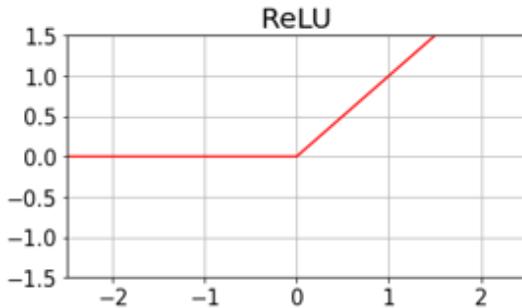
- Nicht-lineare Aktivierungsfunktion:

$$a = \varphi(z) = \varphi \left(\sum_{i=1}^3 w_i x_i + b \right)$$



Source: <https://medium.com/@srnghn/deep-learning-overview-of-neurons-and-activation-functions-1d98286cf1e4>

Aktivierungsfunktionen

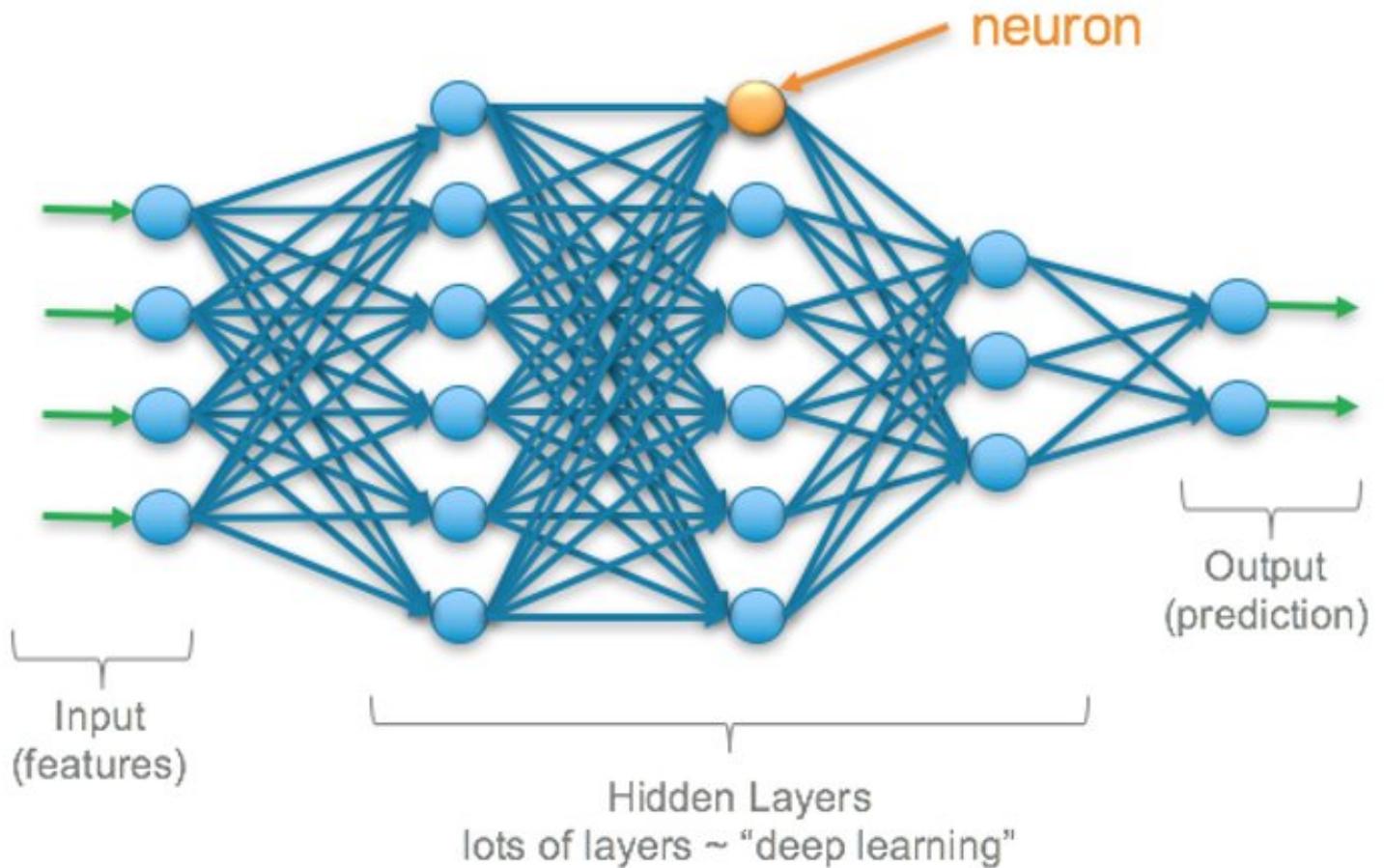


Wichtige Aktivierungsfunktionen:

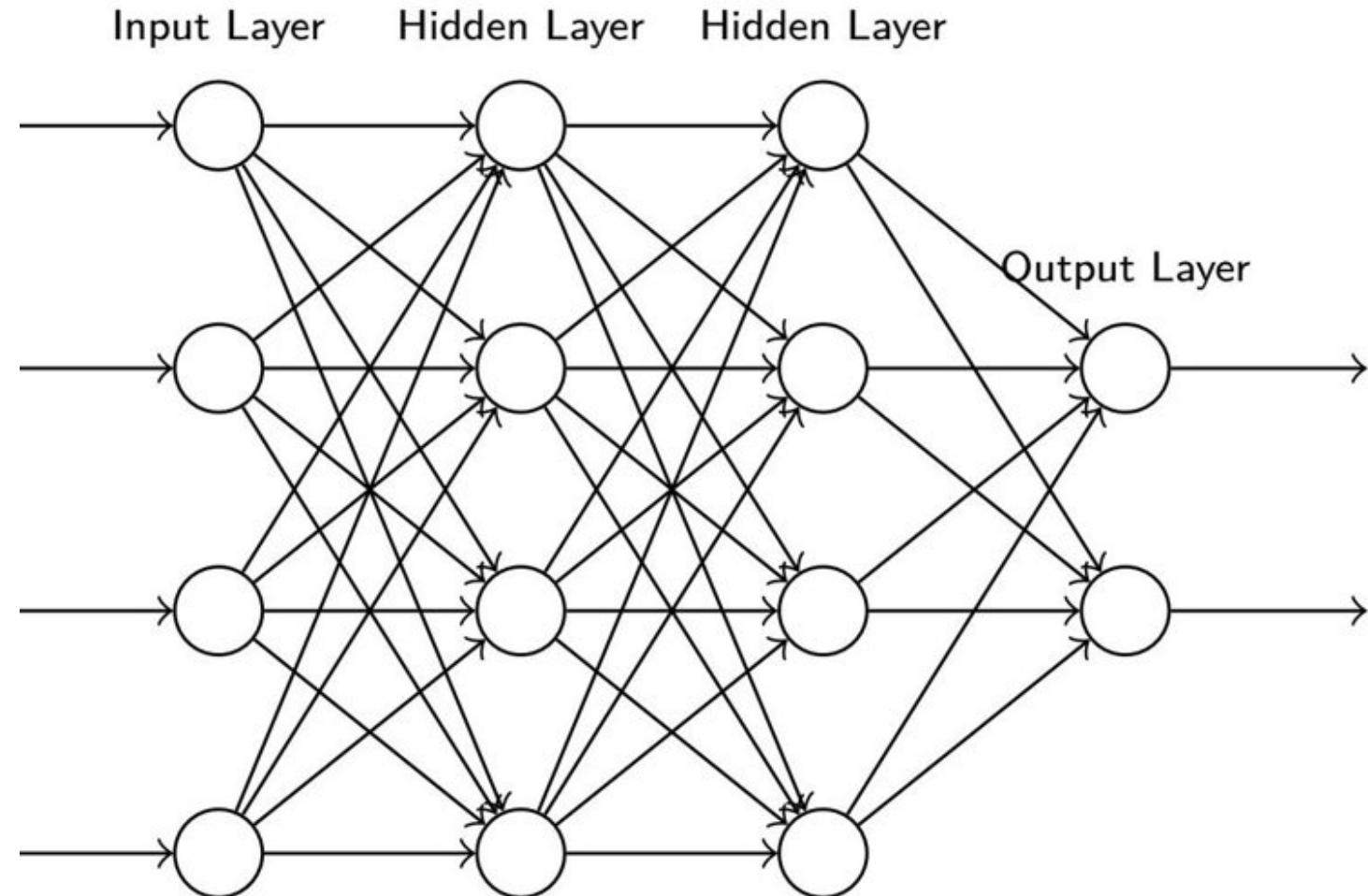
- ReLU (Rectified Linear Unit):
$$\max(0, x)$$
 - Meistverwendete Aktivierungsfunktion für hidden layer
- Sigmoid:
$$\frac{1}{1 + e^{-x}}$$
 - Als Output layer für binäre Klassifikationsprobleme
- Tanh:
$$\tanh(x)$$
 - Findet Anwendung in rekurrenten Netzen z.B. LSTMs

Neuronale Netzwerke

- Sind aus schichtenweise gestapelten Neuronen aufgebaut
- Jedes Neuron erhält als Input den Output aller Neuronen der vorherigen Schicht



Einführendes zum Training



Loss Funktionen

- Input der Loss Funktion:
 - target (Zielgröße)
 - prediction (Output)
- Output der Loss Funktion: Skalarer Wert
- Je kleiner der Loss Output desto besser entspricht der Output des Modells der Zielgröße

Training (Optimierungsproblem)

Minimiere Loss Funktion:

$$\min_{\theta} \mathcal{L}(\theta)$$

- Beispiele für Loss-Funktionen:
 - Regressionsprobleme Mean-Squared-Error:
 - Aktivierungsfunktion der Output Layer: Keine
- Klassifikationsprobleme Cross Entropy:
- Aktivierungsfunktion der Output Layer: Softmax

$$\mathcal{L} \propto \frac{1}{n} \sum_{i=1}^n (f(\mathbf{x}_i; \theta) - y_i)^2$$

$$\mathcal{L} \propto \sum_{i=1}^n p_i \log(q_i), \quad q_i = f(\mathbf{x}_i; \theta)$$



Die Ketten-Regel

- Neuronale Netze sind ineinander geschachtelte Funktionen
 - Affine Transformationen werden in Activation-Funktionen gesteckt
 - Ein Layer wird in das nächste gesteckt
- Solche Ketten von Funktionen werden nach der Kettenregel abgeleitet
 - „Äußere Ableitung mal innere Ableitung“.
- In Code mit mehr Funktionen: $(u \circ v)'(x_0) = u'(v(x_0)) \cdot v'(x_0)$

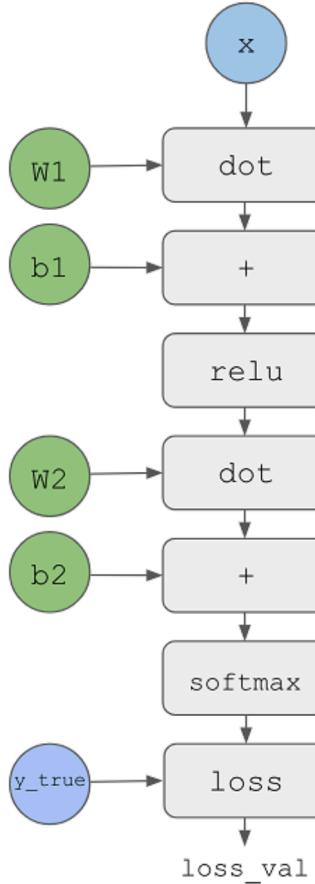
```
def fghj(x):  
    x1 = j(x)  
    x2 = h(x1)  
    x3 = g(x2)  
    y = f(x3)  
    return y  
  
grad(y, x) == grad(y, x3) * grad(x3, x2) * grad(x2, x1) * grad(x1, x)
```



Computation Graphs

Ein Computation Graph ist ein DAG aus Operationen also Tensoroperationen oder Anwendungen von Funktionen.

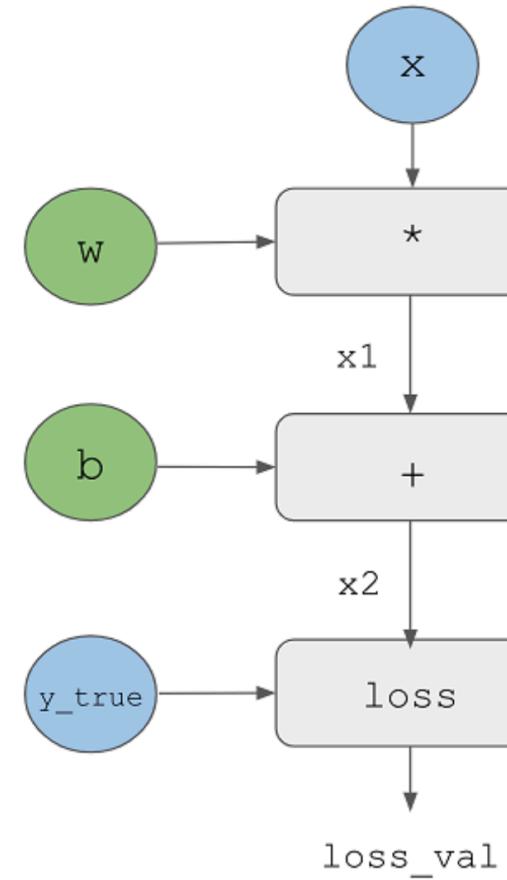
Unser erstes Model →



Backprop im Computation Graph

- Vorwärtsdurchgang: loss_val wird berechnet

```
loss_val = abs(y_true - y)
```



Backprop im Computation Graph

- Vorwärtsdurchgang: loss_val wird berechnet

$\text{loss_val} = \text{abs}(\text{y_true} - \text{y})$

- Rückwärtsdurchgang:
Gradienten für W und b
werden berechnet

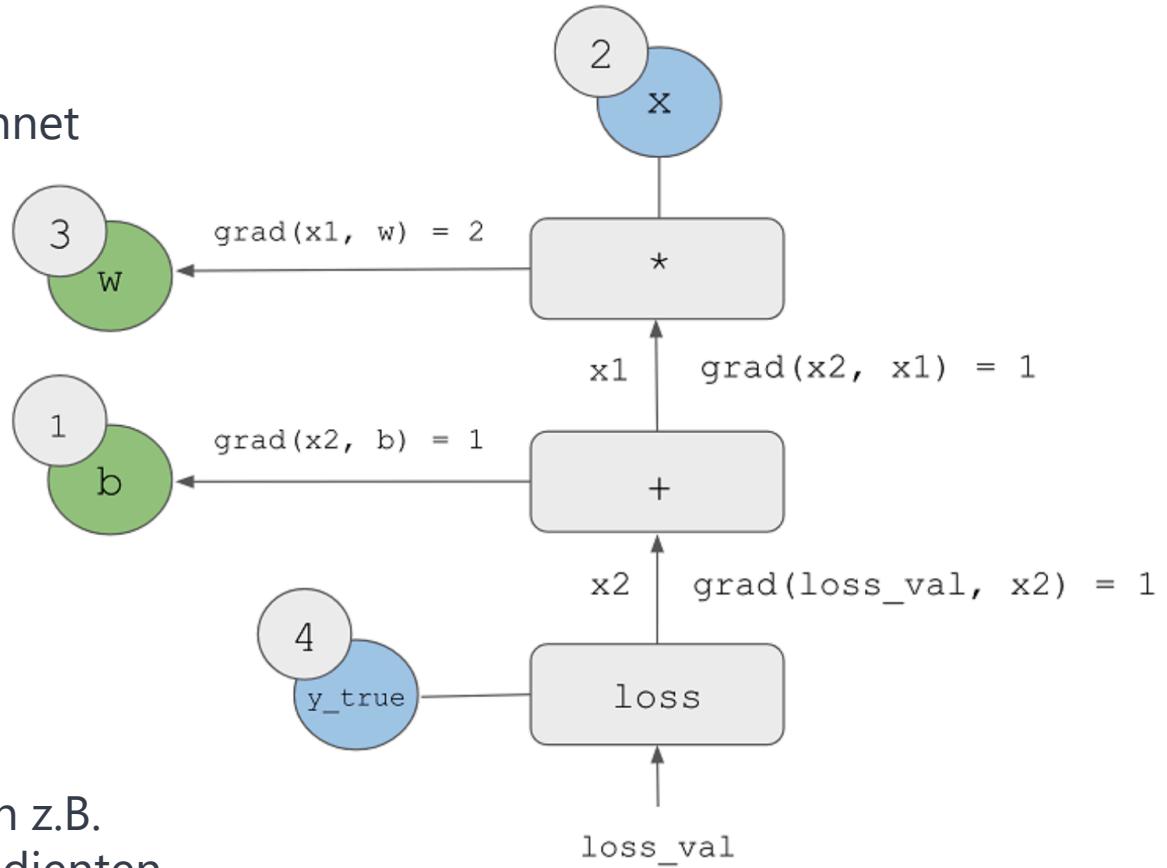
$$\text{grad}(\text{loss_val}, \text{x2}) = 1$$

$$\text{grad}(\text{x2}, \text{x1}) = 1$$

$$\text{grad}(\text{x2}, b) = 1$$

$$\text{grad}(\text{x1}, w) = 2$$

→ Kettenregel: Wir erhalten den Gradienten z.B.
von loss_val bezüglich W indem wir die Gradienten
der Kanten auf dem Weg von loss_val zu W multiplizieren.



Backprop im Computation Graph

- Vorwärtsdurchgang: loss_val wird berechnet

$\text{loss_val} = \text{abs}(\text{y_true} - \text{y})$

- Rückwärtsdurchgang:
Gradienten für W und b
werden berechnet

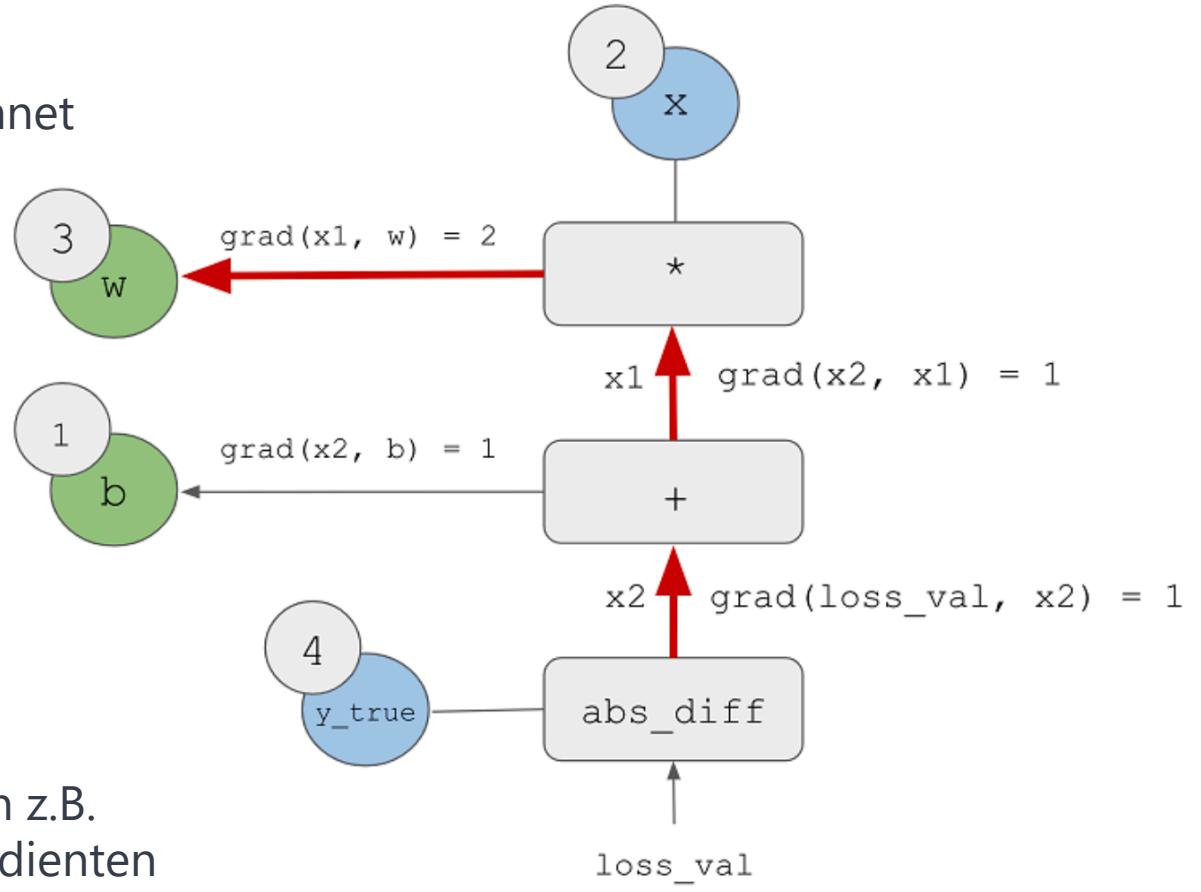
$$\text{grad}(\text{loss_val}, \text{x2}) = 1$$

$$\text{grad}(\text{x2}, \text{x1}) = 1$$

$$\text{grad}(\text{x2}, b) = 1$$

$$\text{grad}(\text{x1}, w) = 2$$

→ Kettenregel: Wir erhalten den Gradienten z.B.
von loss_val bezüglich W indem wir die Gradienten
der Kanten auf dem Weg von loss_val zu W multiplizieren.



Backprop im Computation Graph

Warum ist das toll?

- Automatic Differentiation
= die automatische Berechnung von Gradienten
- Backpropagation bedeutet, die Komplexität der Gradientenberechnung ist gleich der Komplexität des Vorwärtsdurchgangs.



Backprop im Computation Graph

Mathematische Details für eine Implementierung per Hand:
www.neuralnetworksanddeeplearning.com

Neural Networks and Deep Learning

Neural Networks and Deep Learning is a free online book. The book will teach you about:

- Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data
- Deep learning, a powerful set of techniques for learning in neural networks

Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. This book will teach you many of the core concepts behind neural networks and deep learning.

[Neural Networks and Deep Learning](#)

[What this book is about](#)

[On the exercises and problems](#)

► [Using neural nets to recognize handwritten digits](#)

► [How the backpropagation algorithm works](#)

► [Improving the way neural networks learn](#)

► [A visual proof that neural nets can compute any function](#)

► [Why are deep neural networks hard to train?](#)

► [Deep learning](#)
[Appendix: Is there a *simple* algorithm for intelligence?](#)

[Acknowledgements](#)

[Frequently Asked Questions](#)



Optimierer

- SGD
- RMSprop
- Adam
- Adagrad
- Nesterov
- Ftrl
- Etc.



Optimierer - Recherche

- SGD
- RMSprop
- Adam
- Adagrad
- Nesterov
- Ftrl
- Etc.

