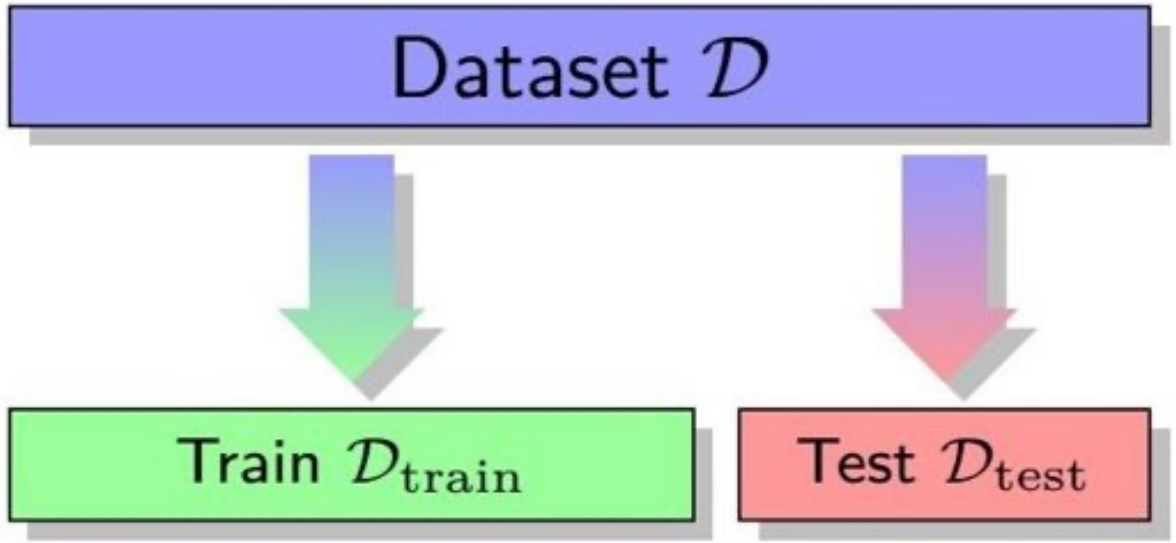
Einführung in neuronale Netze

Bausteine neuronaler Netzwerke

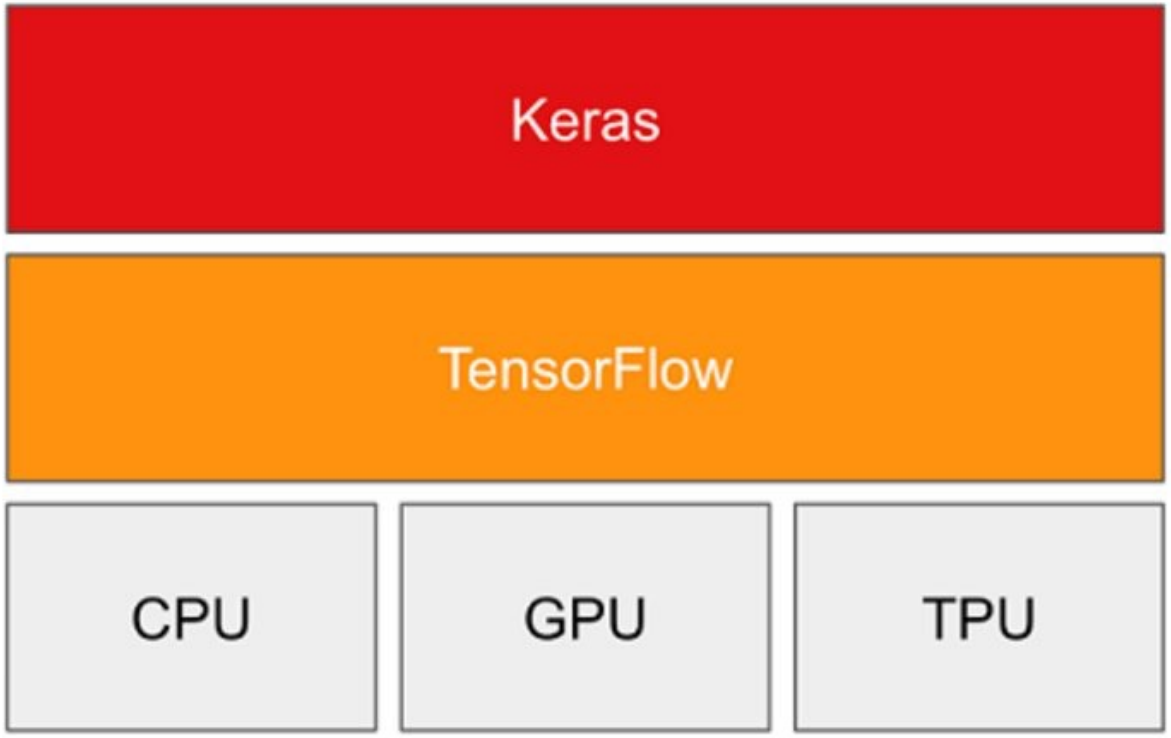
Training-Test Split

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



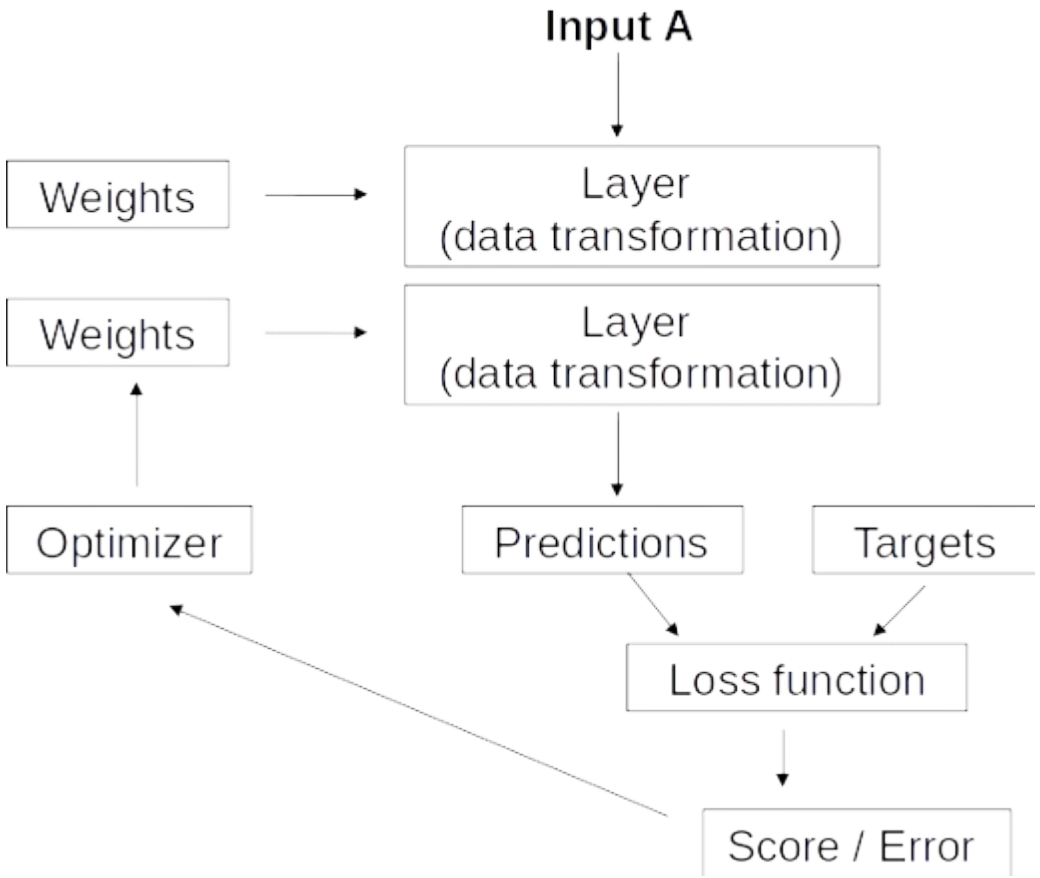
Die Keras API



High Level API: Deep Learning Entwicklung, Layers,

Backend: Autodiff, Optimierer, Loss-Funktionen, …

Bausteine eines neuronalen Netzwerks



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

- 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 unseren Modell einzuschätzen?

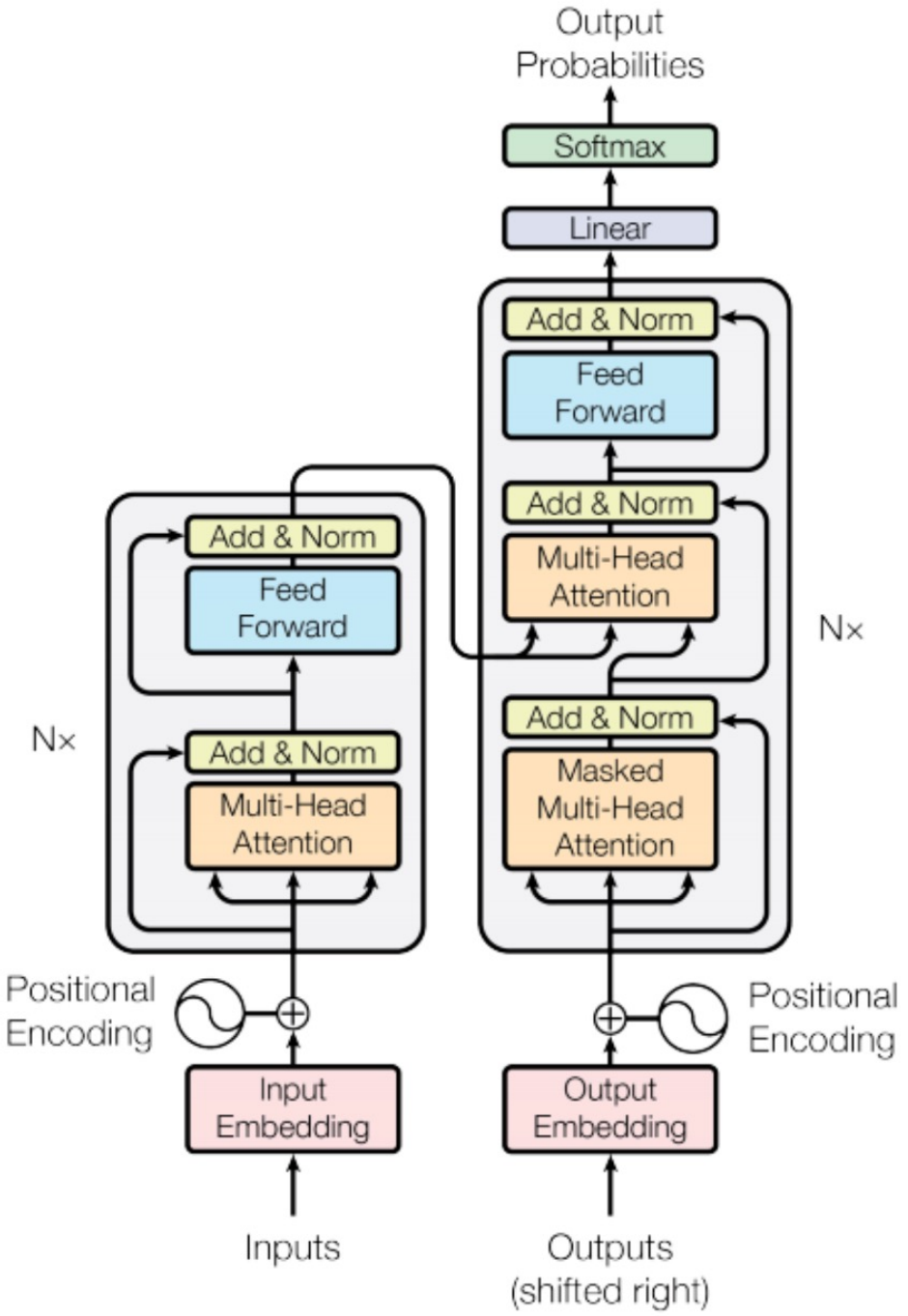
Unterschied - Loss und Metrik

Warum optimieren wir nicht nach der Metrik?

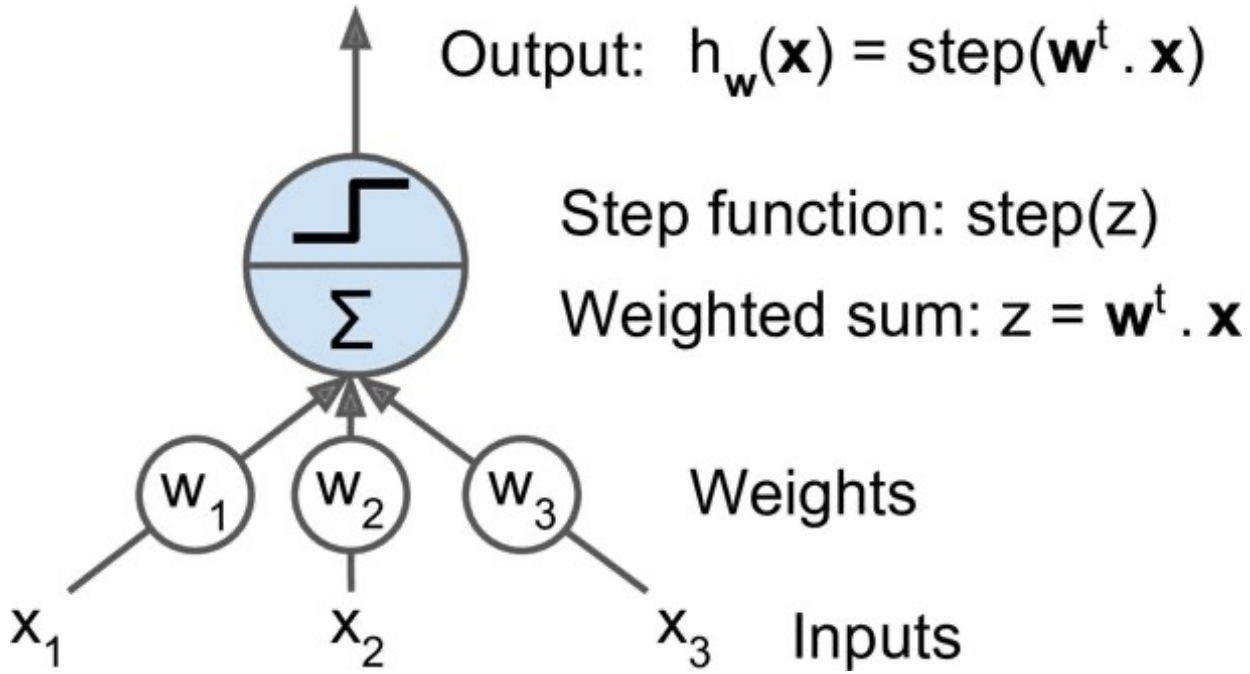
- Metrik nicht unbedingt ableitbar
- Metrik ändert sich oft initial wenig
- Metrik kann natürliche Plateaus erreichen

→ Gute Lossfunktionen weisen diese Eigenschaften nicht auf!

Die Architektur – Layers



Bausteine eines neuronalen Netzwerks



Source: Hands-on Machine Learning, Aurelion Geron

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

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

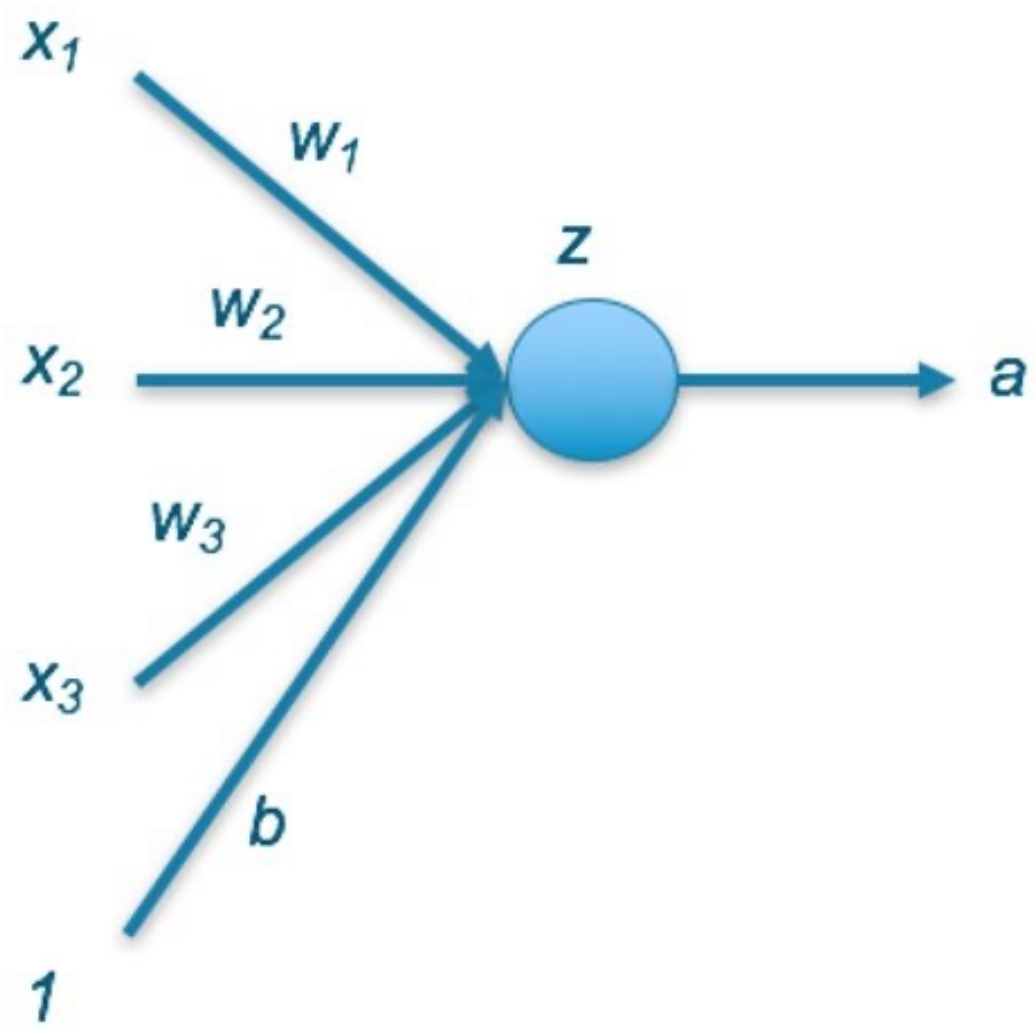
Neurone:

- Linear-affine Abbildung:

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

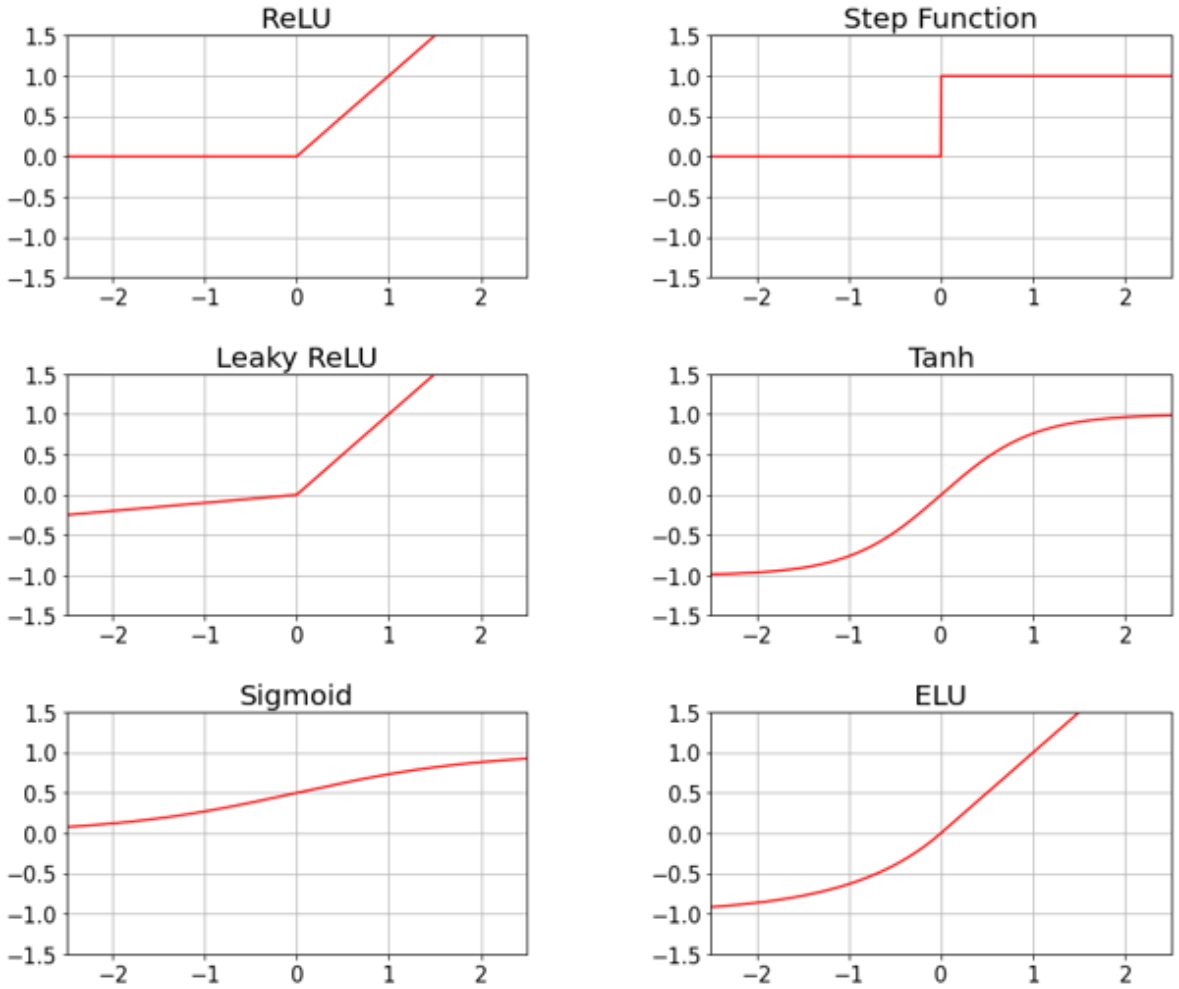
- Nicht-lineare Aktivierungsfunktion:

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



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: $\frac{1}{1 + e^{-x}}$

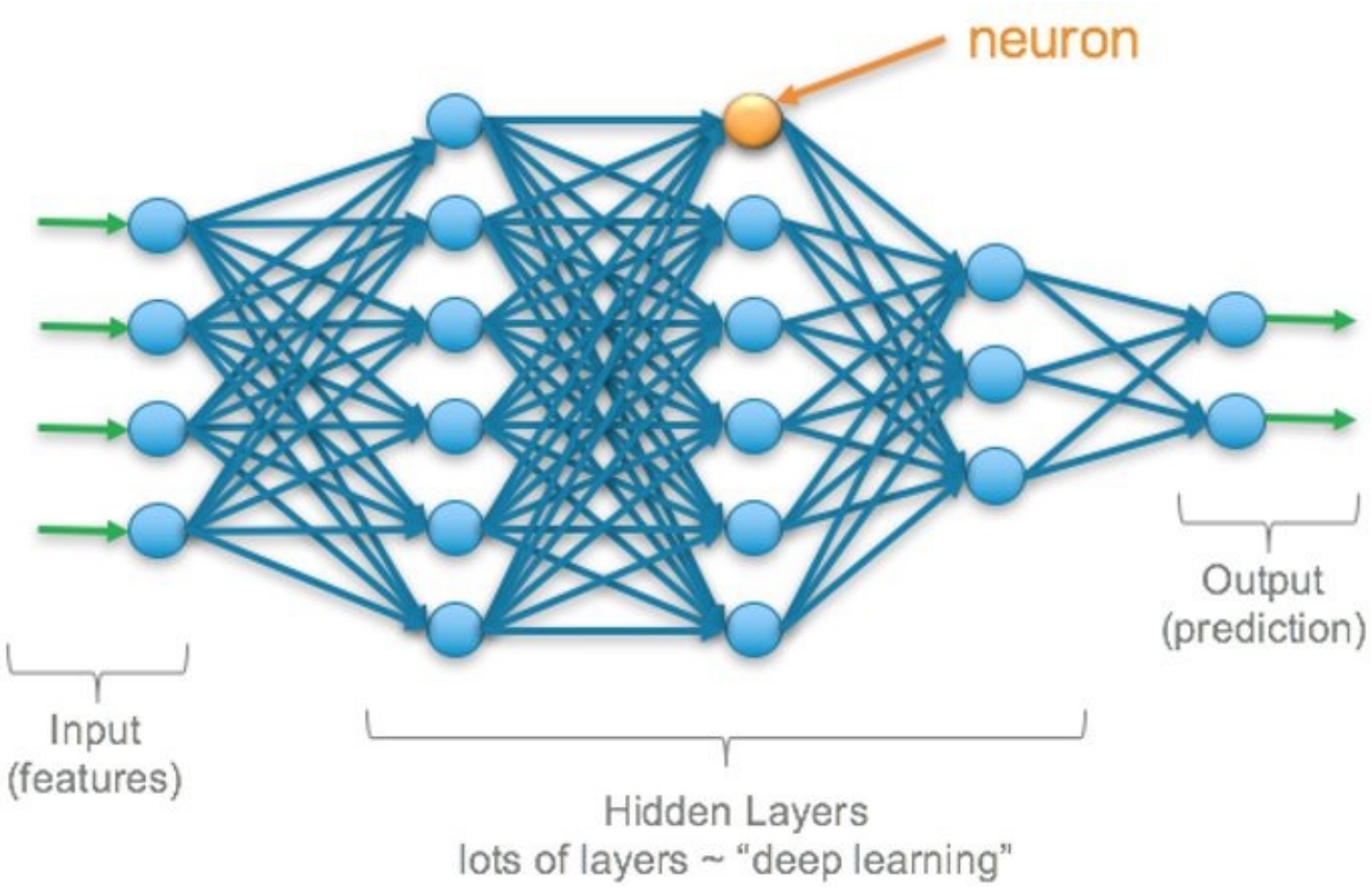
  - Als Output layer für binäre Klassifikationsprobleme

- Tanh: $\tanh(x)$

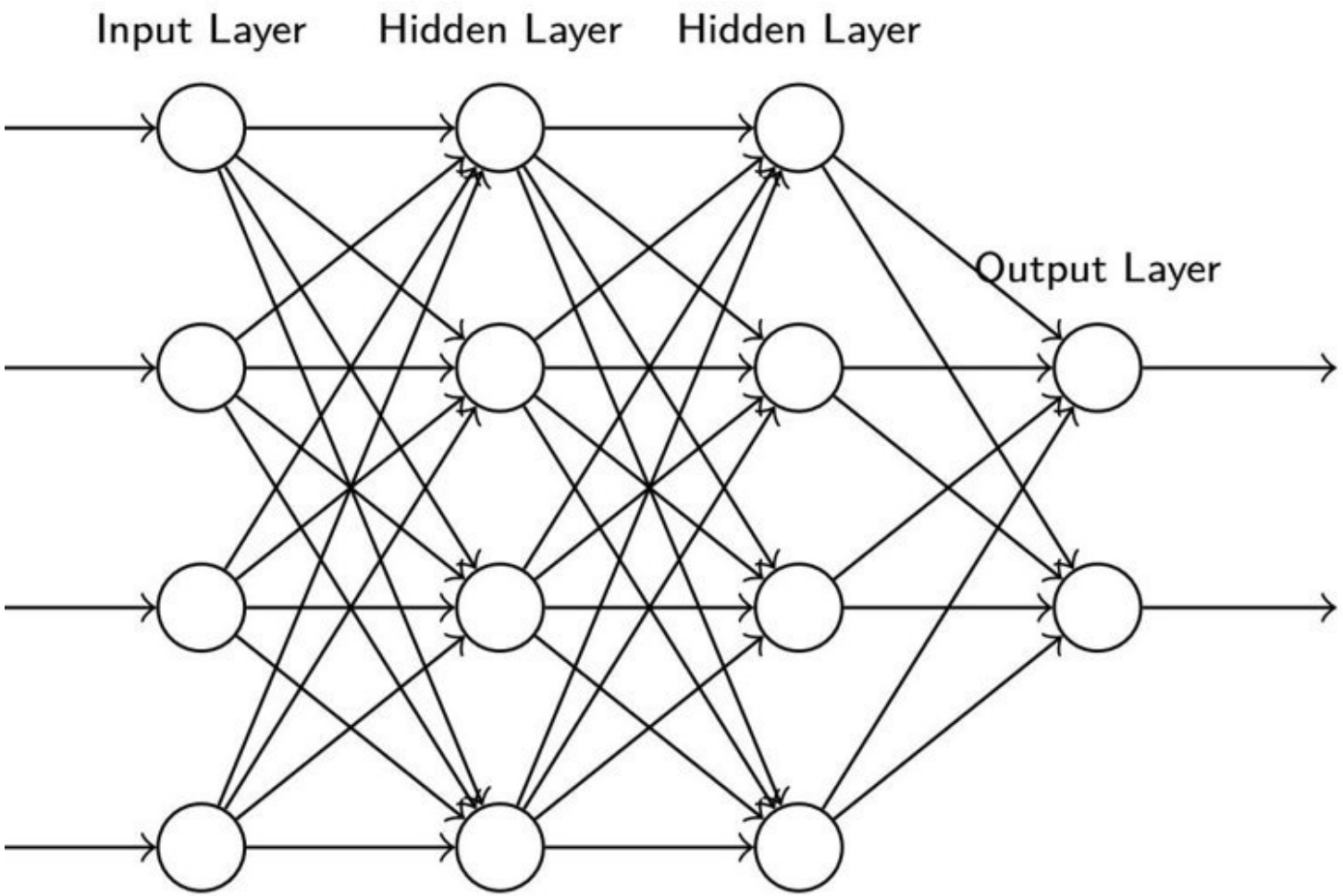
  - Findet Anwendung in rekurrenten Netzen z.B. LSTMs

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

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

  - Klassifikationsprobleme Cross Entropy:
  - Aktivierungsfunktion der Output Layer: Softmax

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