



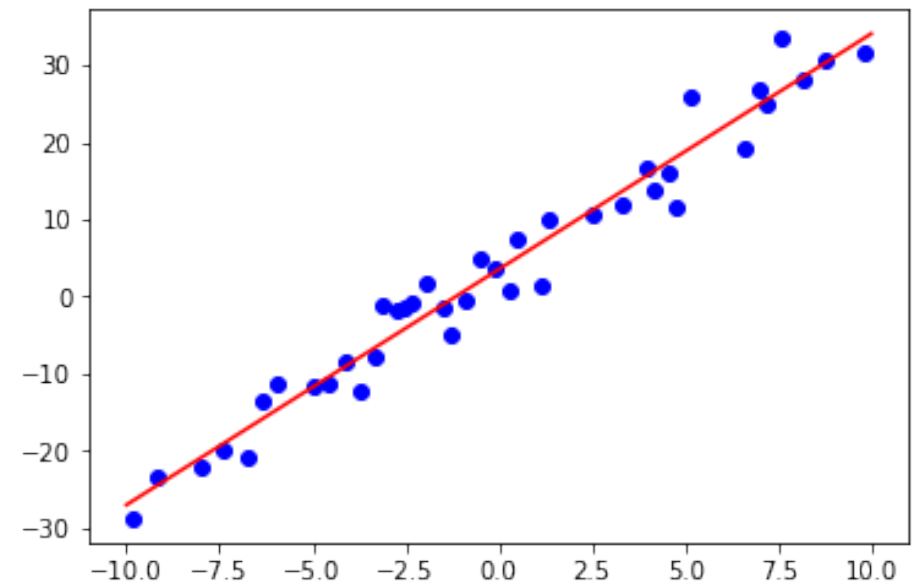
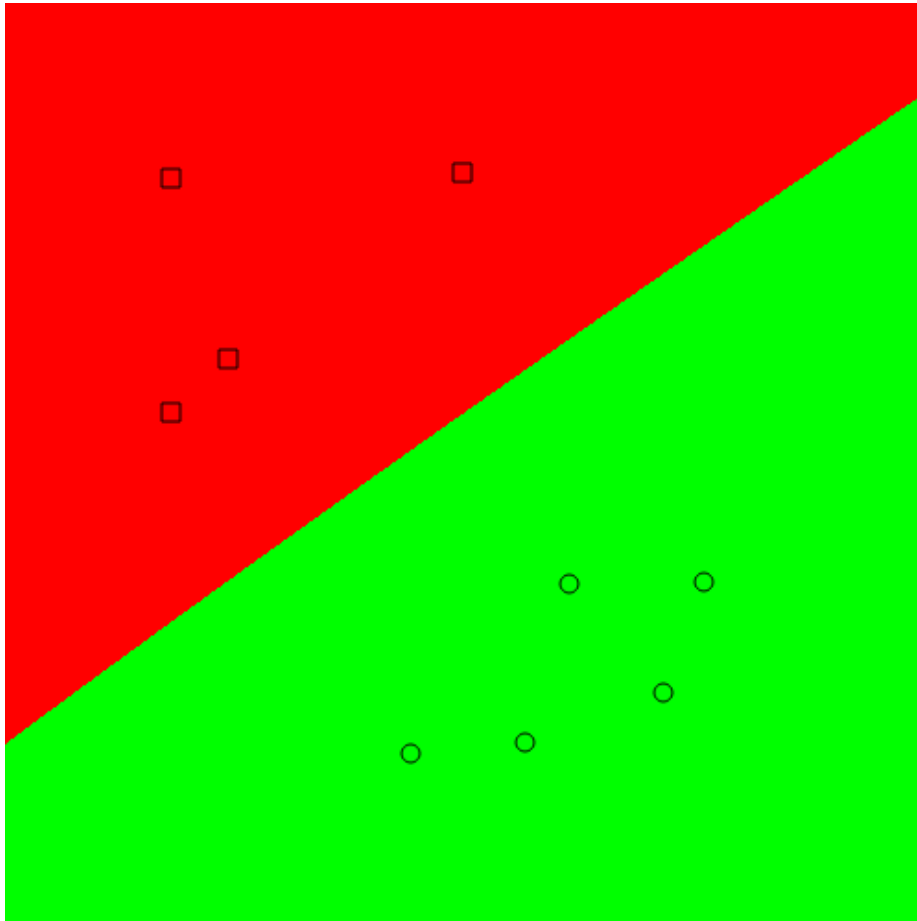
Supervised Learning

Für Binärklassifizierung

Gliederung

- 1. Die Problemstellung**
- 2. Adaline (Adaptive Linear Neuron) + Programmierter Teil**
- 3. (Multilayer Perceptron / Neuronales Netzwerk)**

Supervised learning



Die Problemstellung: Binärklassifizierung

Gegeben sind in Vektorform gefasste Eigenschaften eines Objekts oder Ereignis (features):

$$\vec{x}$$

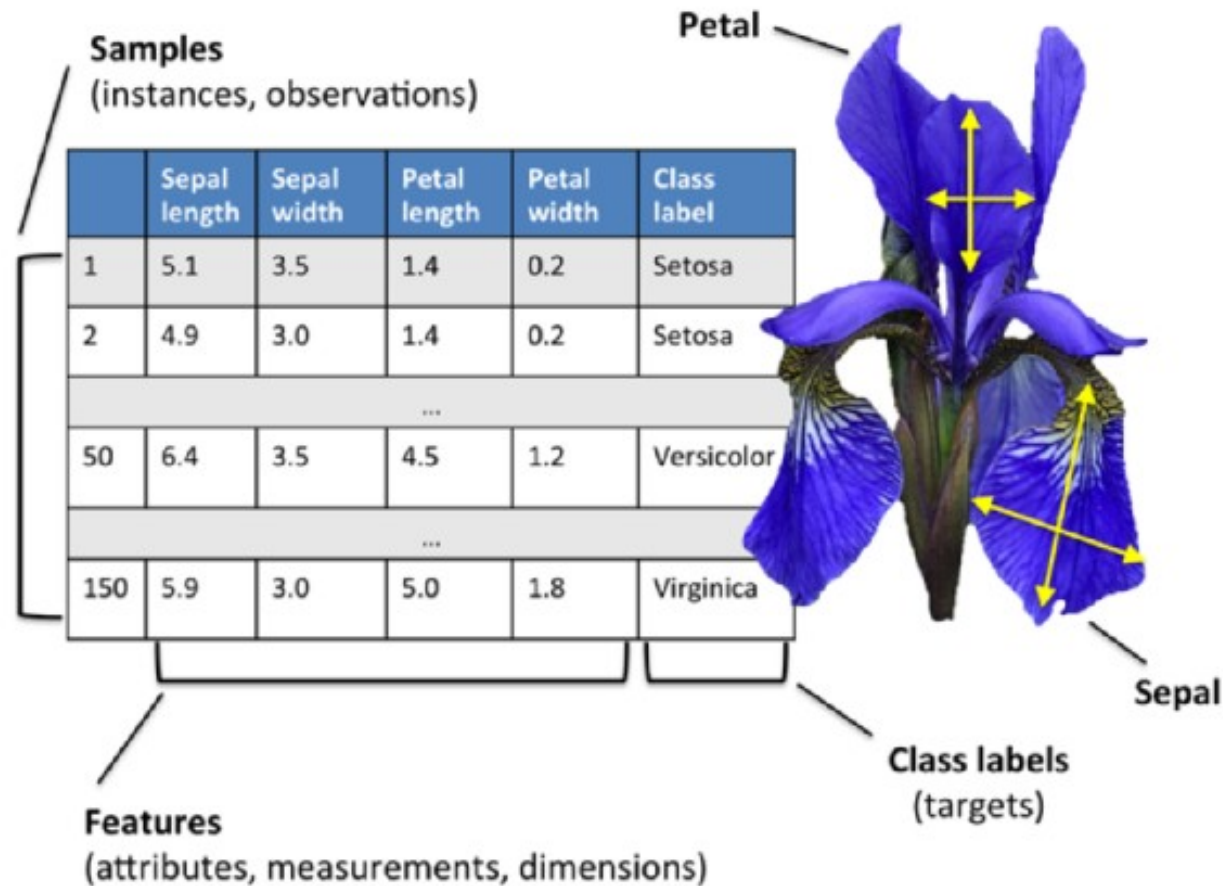
Zu jedem \vec{x} gibt es eine passende Klasse (target):

$$y(\vec{x})$$

Für Binärklassifizierung gilt:

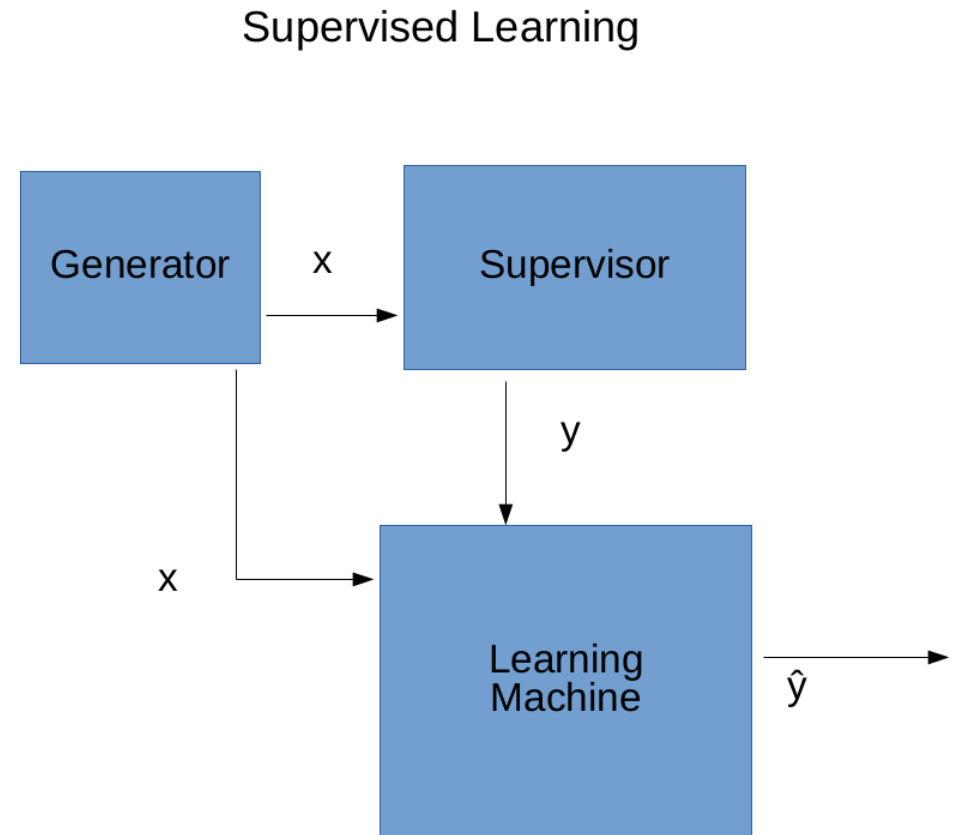
$$y \in \{0,1\}$$

Beispiel für Binärklassifizierung:

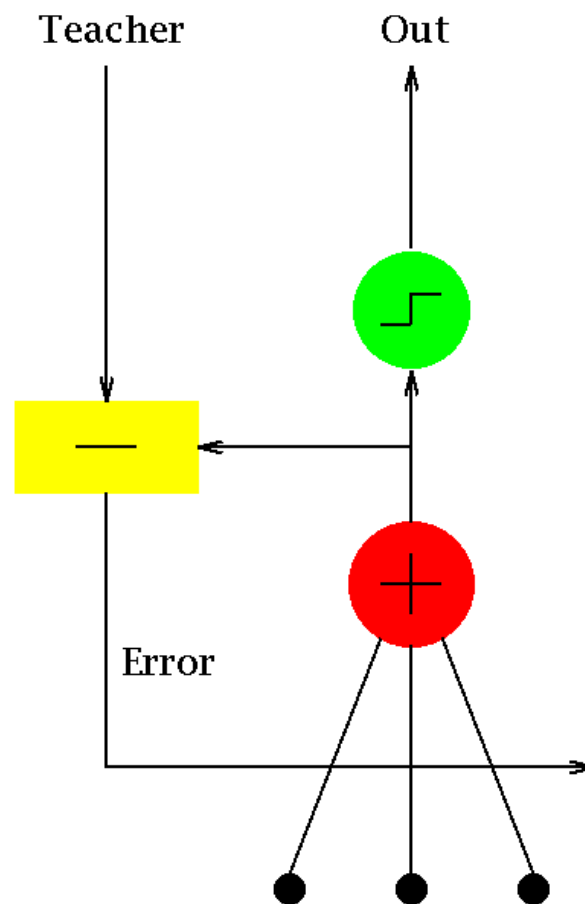


Die Problemstellung: Supervised Learning

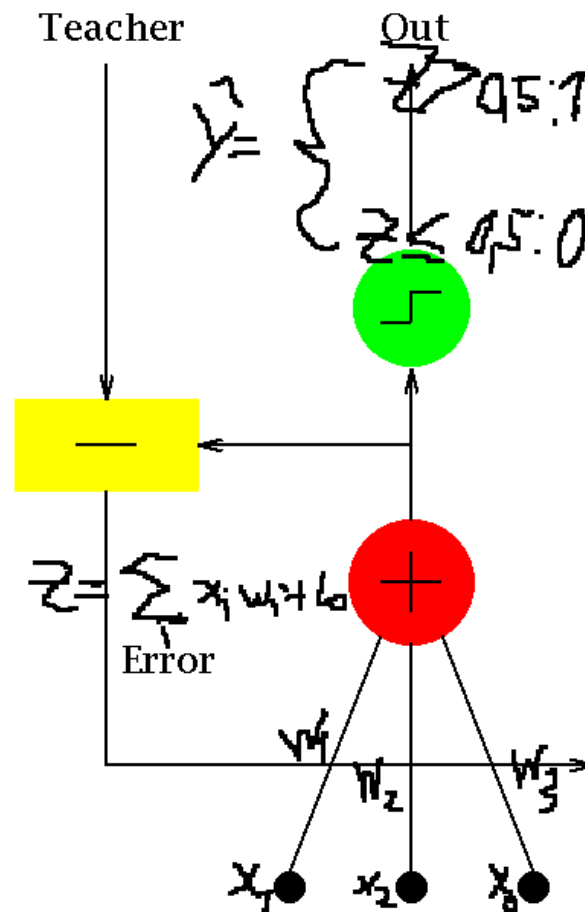
- Gegeben sind Daten x die vom Generator kommen
- Der Supervisor gibt die korrekte Klasse $y(x)$ an die Parameter der Funktion $y(x)$ sind nicht bekannt
- Die Lernmaschine erstellt eine Annäherung $\hat{y}(x)$ die das Verhalten des Supervisors wiedergeben soll.



Adaptive Linear Neuron



Adaptive Linear Neuron



Das Verlustfunktional

- Um die optimalen Parameter zu finden benötigt man eine Ziel-, oder Verlustfunktion (bzw. ein Verlustfunktional), anhand dessen die Parameter bewertet werden können:

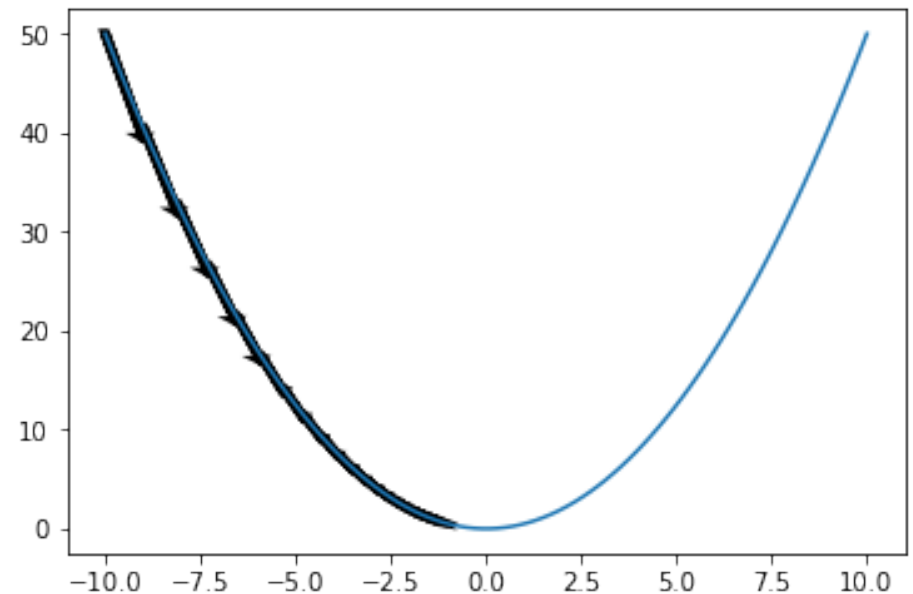
$$z_i = \vec{w} \cdot \vec{x}_i + b$$

$$L(\vec{w}, b) = \frac{1}{2} * \sum_1^l (y_i - z_i)^2$$

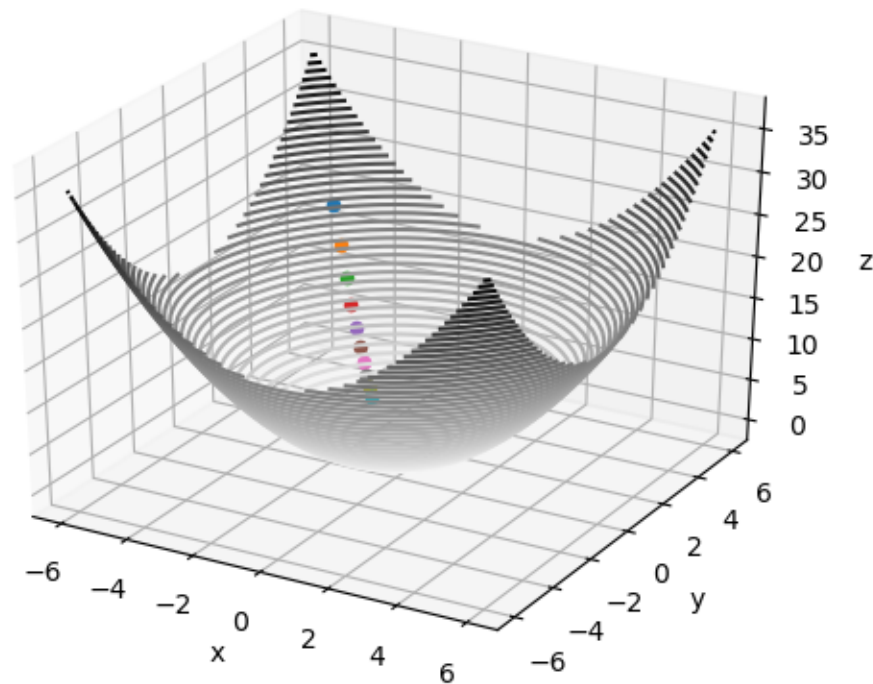
Das Gradientenverfahren: Idee

• **Ableitung = Steigung =
Wie Stark geht es
aufwärts**

**=> Läuft man entgegen
der Steigung so landet
man irgendwann in
einem Minimum**



Gradientenverfahren in 3d



AdalineSGD Pseudocode:

1. Initialisiere Parameter: $\vec{w} = \vec{0}, b = 0$

2. Wiederhole n mal:

3. Für alle x_j :

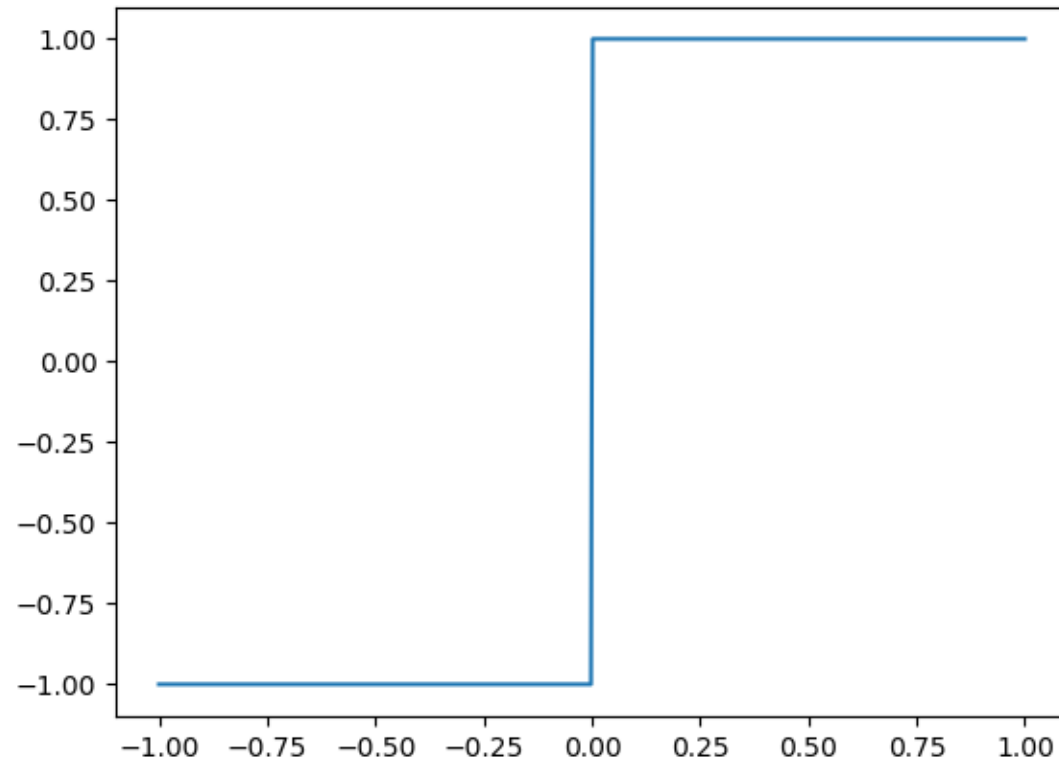
4. Für alle w_i : $w_i := w_i - \alpha * \frac{dL_j}{dw_i}$

5. Für b : $b := b - \alpha * \frac{dL_j}{db}$

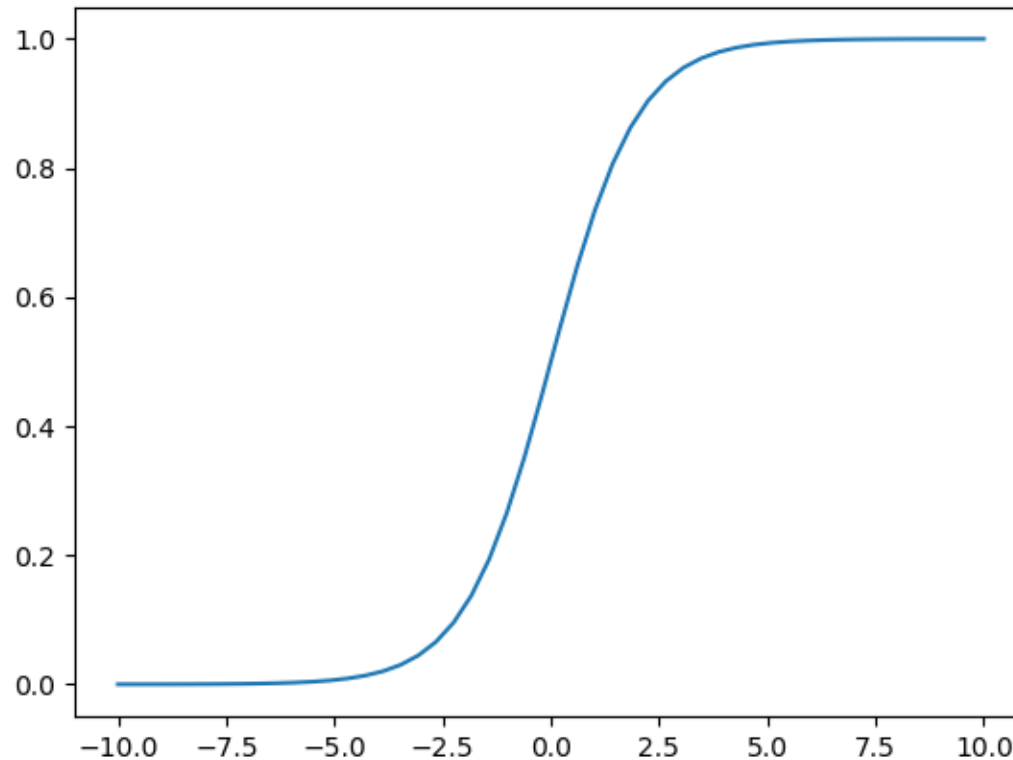
Semantische analyse mit Tfidf

- **Dokumente können dargestellt werden, indem man die Wörter zählt und diese dann als vektor darstellt.**
- **Mit Tfidf wird die Zählung umgewichtet, indem Wörter, die in allen dokumenten häufig vorkommen abgewertet werden.**

Diskrete Aktivierungsfunktion (Schrittfunktion)

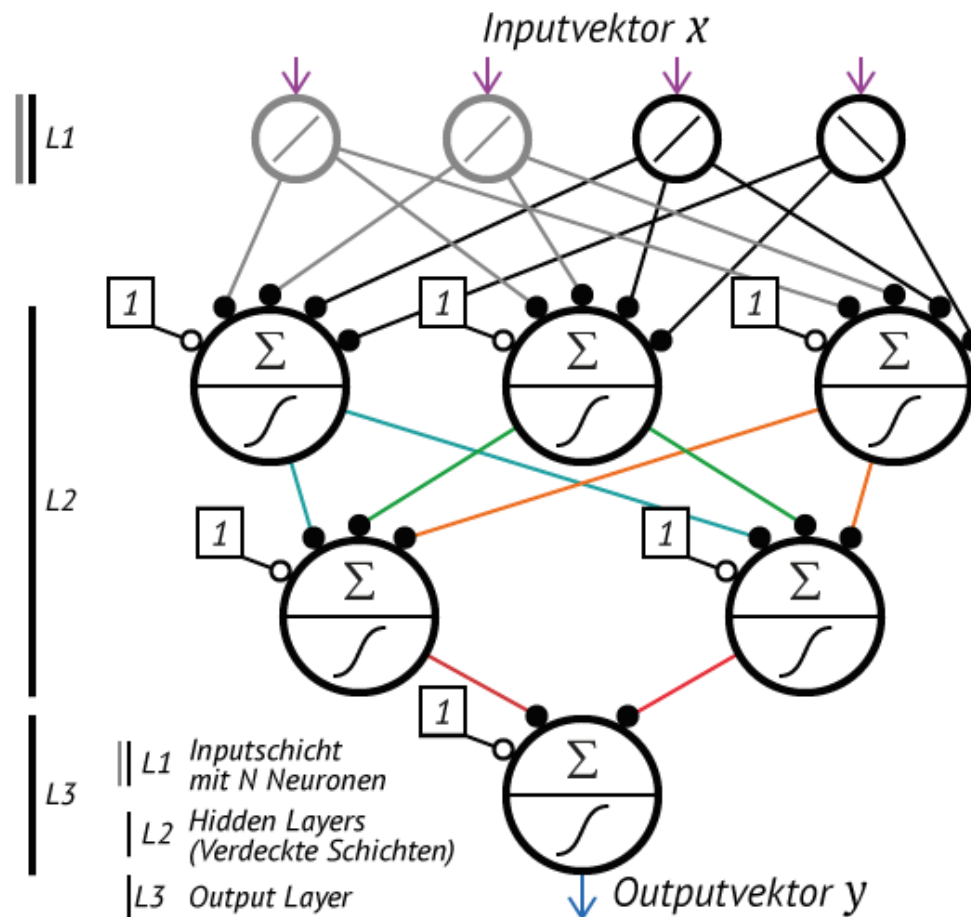


Kontinuierliche Aktivierungsfunktion / Sigmoid Funktion

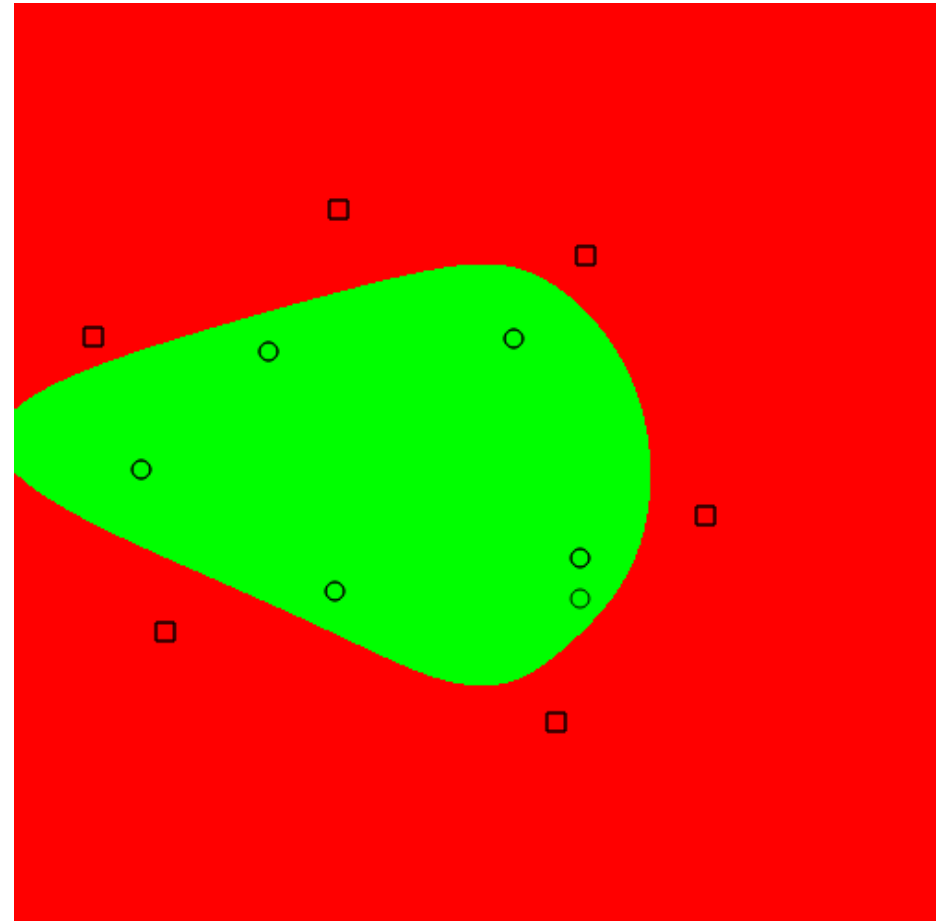
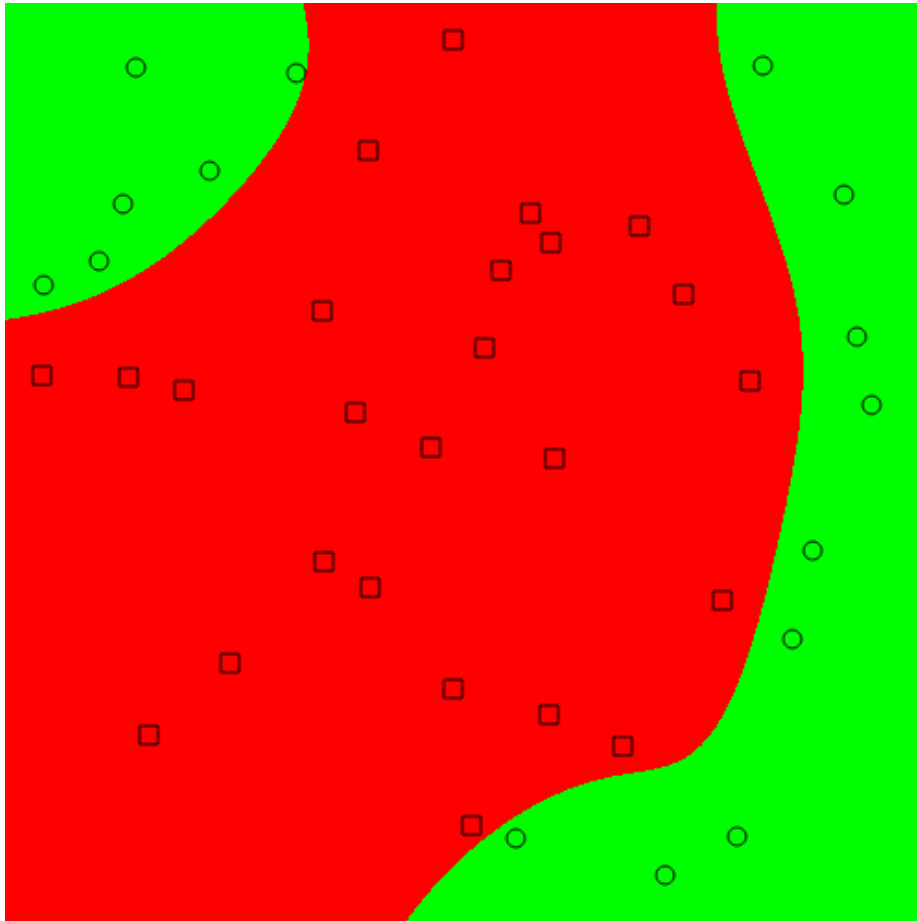


$$\sigma(x) = \frac{1}{(1 + e^{-x})}$$

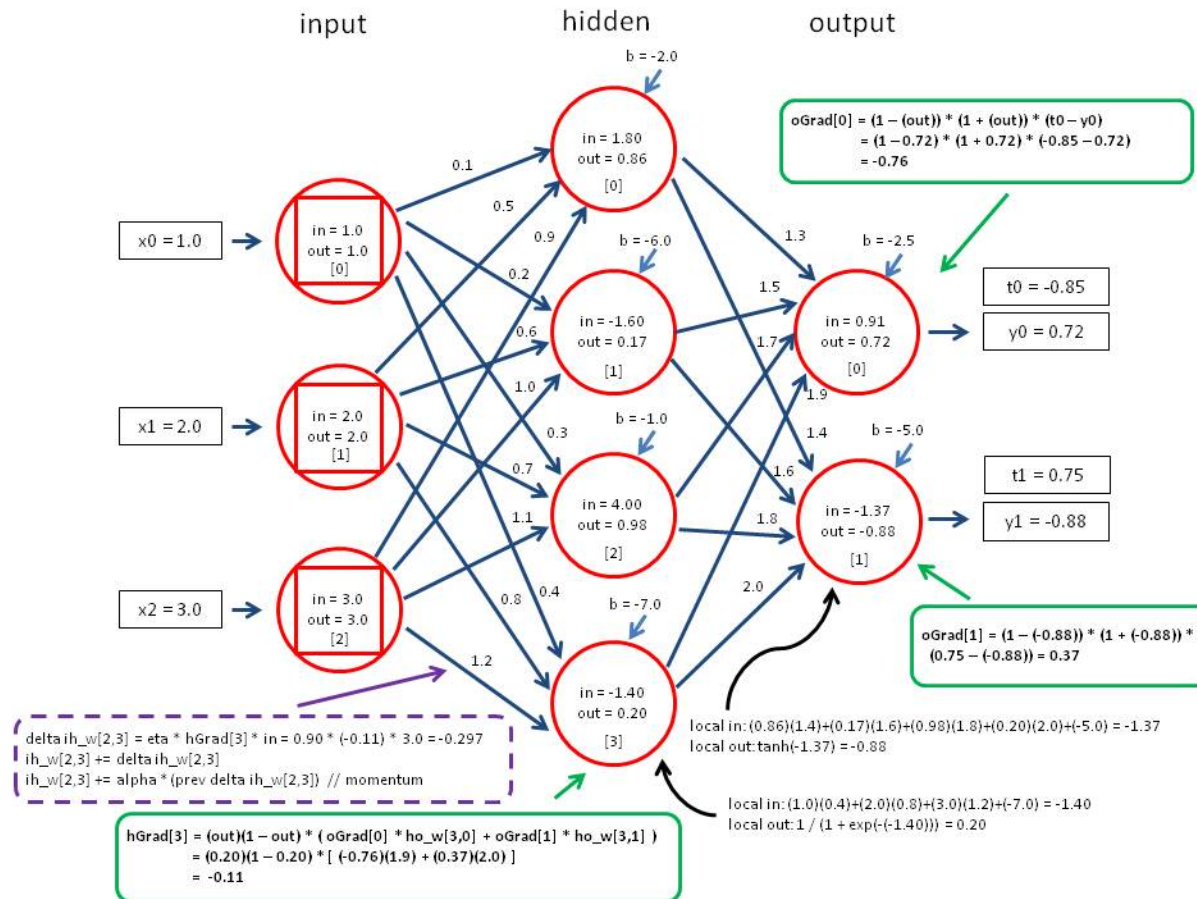
Multilayer-Perzeptron / Neuronales Netzwerk



Nichtlineare Klassifizierung mit Neuronalen Netzwerken



Trainingsverfahren Backpropagation



Anwendungen Neuronale Netze

•Medizin:

- Bilderkennung Hautkrebs

•Autoindustrie:

- Bilderkennung und Lokalisation von Objekten für Selbstfahrende Autos

•Management / Operations Research:

- Vorhersage und Analyse von Kundenverhalten
- Gewinn- / Risikovorhersagen

•Chemie:

- Vorhersage von Moleküleigenschaften

Quellcode:

Quellen

•Bilder:

- <http://www.informatikseite.de/neuro/img42.png>
- <https://jamesmccaffrey.files.wordpress.com/2012/11/backpropagationcalculations.jpg>
- <http://www.bogotobogo.com/python/scikit-learn/images/features/iris-data-set.png>
- https://upload.wikimedia.org/wikipedia/commons/b/be/Adaline_flow_chart.gif

•Bücher:

- Statistical Learning Theory (Vladimir N. Vapnik 1998)
- Python Machine Learning (Sebastian Raschka 2015)
- Deep Learning (Ian Goodfellow, Youshua Bengio, Aaron Courville 2016)

Graphiken wurden mit python matplotlib und html5 canvas erstellt.