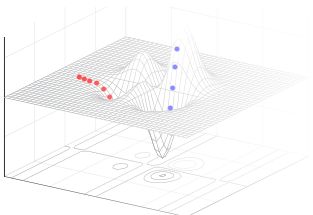
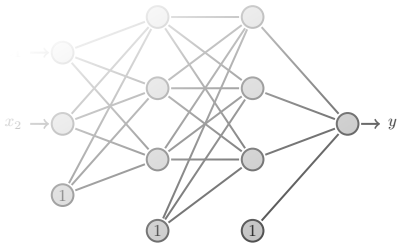
Multilayer Perceptron

Prof. Dr. Jörg Frochte

Maschinelles Lernen

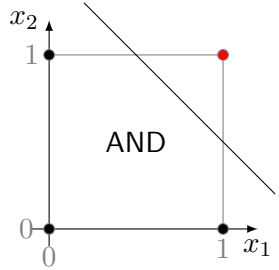




Grenzen eines einlagigen Netzes

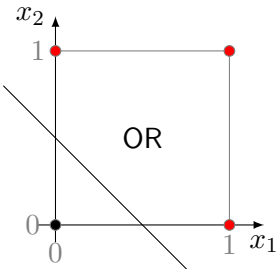
- Was ist es also, was uns zwingt, komplexere Strukturen einzusetzen?
- Dazu muss man sich klar machen, was unser einfaches Perzeptron wirklich kann.
- Wir wissen, dass es Klassen fehlerfrei trennen kann, die linear separierbar sind.
- Das sind die Art von Problemen, die es lernen kann.
- Um komplexere Zusammenhänge deutlich machen zu können, wäre es sinnvoll, wenn unser Netzwerk die grundlegenden logischen Verknüpfungen lernen könnte.
  - OR
  - AND
  - XOR

Elementare logische Operationen



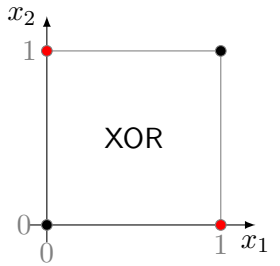
AND ist linear separierbar

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



OR ist linear separierbar

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

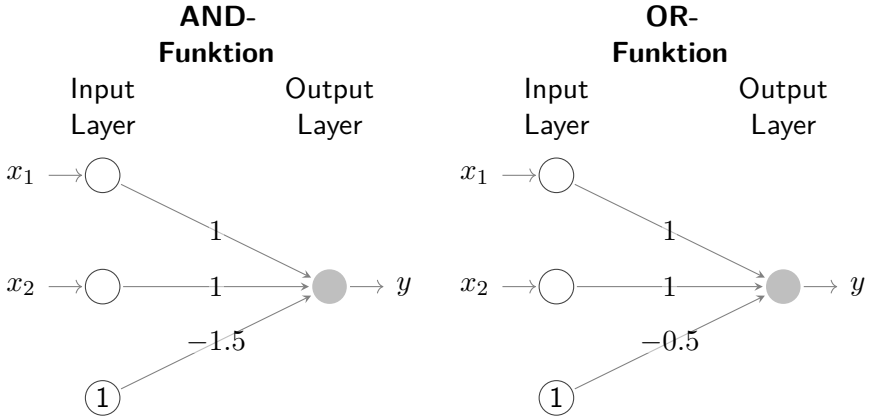


XOR ist *nicht* linear separierbar

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Elementare logische Operationen als Netzwerk

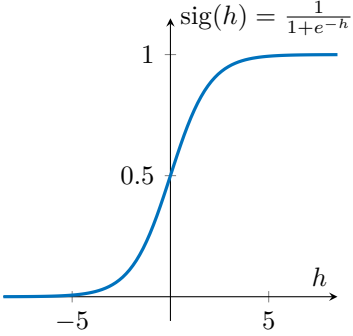
- Die zugehörigen Perzeptrons für AND und OR können analytisch ermittelt werden.
- Im Output-Neuron wird dabei jeweils die Heaviside-Funktion verwendet.

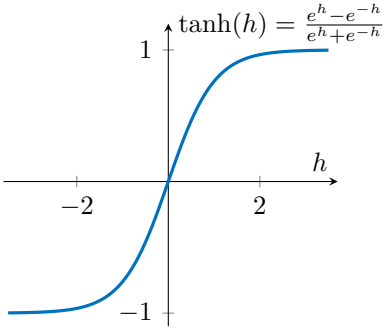


- Für XOR lässt sich, wie schon absehbar, keine entsprechende Darstellung bzw. Gewichte w finden.
- Um diese Limitierung aufzubrechen, werden wir zum einen komplexere Strukturen einfügen, in denen mehrere Neuronen neben-, aber vor allem hintereinander verschaltet werden.
- Diese sind jedoch weit komplexer zu trainieren als unser einfaches Neuronen-Modell zuvor.
- Um dieses Training durchzuführen, kommen Verfahren aus der mathematischen Optimierung zum Einsatz.
- Diese Verfahren basieren jedoch im Regelfall auf Ableitungen und die von uns bisher verwendete Heaviside-Funktion ist nicht differenzierbar.

Aktivierungsfunktionen

- Heaviside-Funktion ist als Aktivierungsfunktion nur von historischem Interesse.
- Im Training ist sie schlecht, weil die Ableitung fast überall 0 ist.
- Im Output-Layer für eine Regression wird keine Aktivierungsfunktion verwendet.
- Typische Aktivierungsfunktionen für Hidden-Layer:

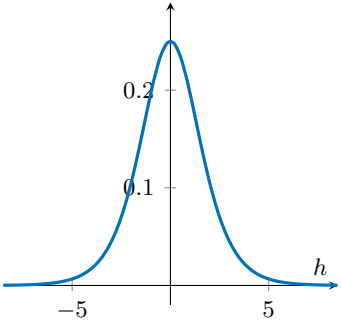




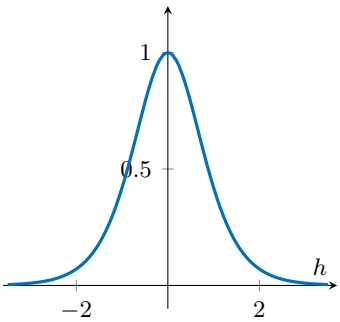
Ableitungen der Aktivierungsfunktionen

- Die Ableitungen der Aktivierungsfunktionen haben eine einfache Struktur.
- Das ist wichtig, weil die Ableitungen automatisch gebildet werden und zwar über viele Schichten mittels Kettenregel.

$$\mathrm{sig}'(h) = \mathrm{sig}(h)\,(1 - \mathrm{sig}(h))$$

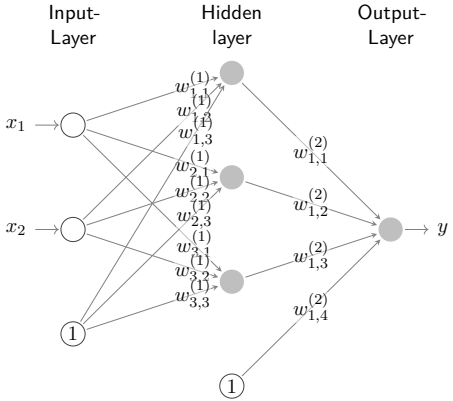


$$\tanh'(h) = (1 + \tanh(h))\,(1 - \tanh(h))$$



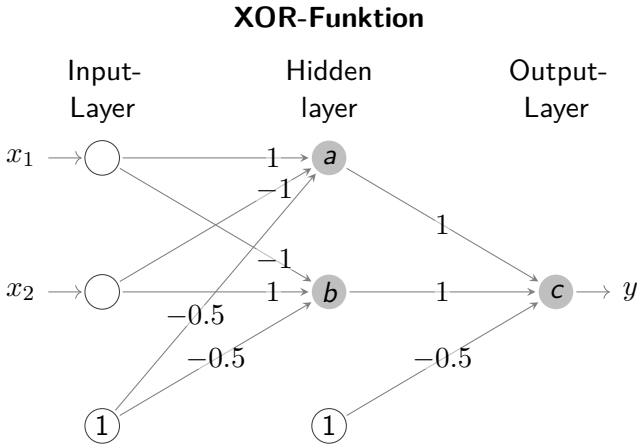
Perzeptron mit einem Hidden-Layer

- Das letzte Netz bestand nur aus einem **Input-Layer**, in dem die Merkmale dem Netzwerk zugefügt werden, und dem **Outputlayer**, welcher die fertige Berechnung bereitstellt.
- Im Gegensatz zu dem Netzwerk zuvor, gibt es im Netz rechts eine weitere Schicht zwischen diesen beiden.
- Man spricht von einem **Hiddenlayer**. Das Netzwerk wird also als etwas angesehen, das in Schichten organisiert ist.
- Die Komplexität des Modells hängt von der Anzahl der Neuronen pro Schicht und der Anzahl der Schichten ab.
- Rechts ist ein Beispiel mit einem Hiddenlayer.



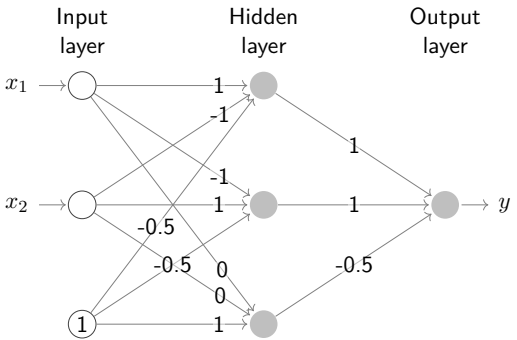
Perzeptron mit einem Hidden-Layer für XOR

- Welchen Sprung an Möglichkeiten man mit einem Hidden-Layer macht, erkennt man in der Abbildung rechts.
- Wir verwenden hier noch ein letztes Mal, anstatt einer Sigmoid-Funktion in den Hidden- und Output-Neuronen, jeweils die Heaviside-Funktion.
- In den Übungen rechnen wir dieses noch einmal detaillierter nach.

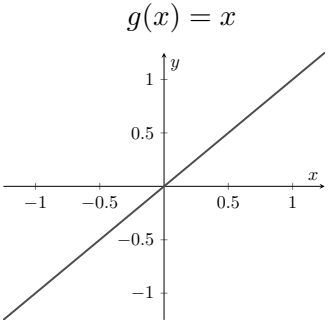


Bias-Neuron

- An dem Beispiel-Perzeptron für die XOR-Funktion erkennt man auch, dass die Bias-Neuronen in dem Hidden-Layer im Gegensatz zu dem im Input-Layer optional sind.
- Die Hidden-Layer können mit Bias-Neuronen versehen werden, jedoch kann das Netz bei Bedarf selbst Bias-Neuronen in diesen Schichten ausbilden.
- Hierfür muss mindestens in der ersten Schicht ein Bias-Neuron vorhanden sein.
- Ein Nachteil daran, diesen Effekt im Inneren ausbilden zu müssen, ist u. a. dass mehr Gewichte bestimmt werden müssen.



- Eine weitere Aktivierungsfunktionen, die oft in den Output-Layern Verwendung findet ist die **lineare Aktivierungsfunktion**.
- Wird im Output-Layer nur auf Sigmoid oder Tangens Hyperbolicus zurückgegriffen, ist der Wertebereich des Netzwerkes auf $(0, 1)$ bzw. $(-1, 1)$ für den Output $\hat{y}$ begrenzt.
- Technisch betrachtet können in einem Netzwerk immer weitere Hidden-Layer hinzugefügt werden.

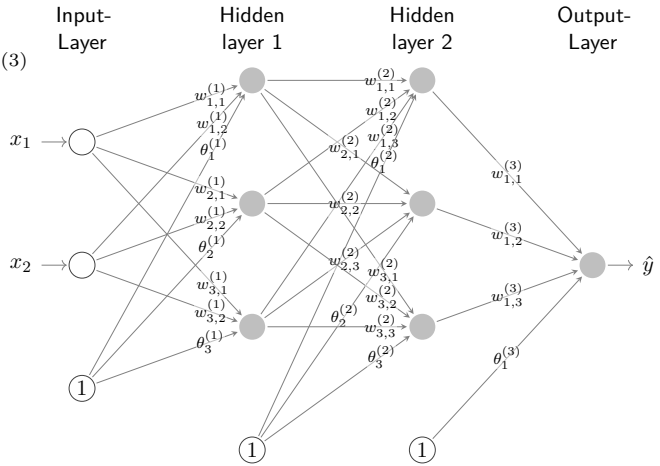


- Bei Netzwerken mit mehr als zwei Hidden-Layern spricht man schon von tiefen bzw. **Deep Networks**.
- Jede weitere Schicht über zwei Hidden-Layern erweitert jedoch nicht mehr die Klasse der darstellbaren Funktionen.
- Man kann alle stetigen Funktionen sogar schon mit einem Hidden-Layer approximieren, wenn man in dem Hidden-Layer Sigmoid-Funktionen und im Outputlayer eine lineare Aktivierungsfunktion vorsieht.

Mehrlagiges Perzeptron

$$\hat{y} = \mathbf{W}^{(3)}\mathbf{g}(\mathbf{W}^{(2)}\mathbf{g}(\mathbf{W}^{(1)}\mathbf{x}+\boldsymbol{\theta}^{(1)})+\boldsymbol{\theta}^{(2)})+\boldsymbol{\theta}^{(3)}$$

- Es ist bekannt[1], dass **mehrlagige Perzeptrons** universelle Methoden zur Funktionsapproximation sind.
- Genügend Neuronen vorausgesetzt, kann jede stetige Funktion angenähert werden.
- g ist die elementweise Aktivierungsfunktion.



[1] George Cybenko, *Approximation by superpositions of a sigmoidal function*, in Mathematics of Control, Signals, and Systems (MCSS), 1989

- Die Gestalt der zu approximierenden Funktionen kann theoretisch sowohl über mehr als auch längere Hidden-Layer – also mehr Neuronen in jedem Layer – adressiert werden.
- Was besser ist, hängt sehr vom Anwendungsfall ab. Ein Problem, das jedoch lange Zeit mehr Hidden-Layer als zwei zu einem seltenen Phänomen gemacht hat, ist die größere Herausforderung beim Training.
- Bevor wir jedoch zum Training des Netzes übergehen, ist es hilfreich, sich klar zu machen, wie das Netz Werte berechnen würden, wenn es bereits vollständig trainiert wäre.
- Nehmen wir dazu an, wir würden
  - in allen Hidden-Layern auf explizite Bias-Neuronen verzichten, sowie
  - Sigmoid-Funktionen für die Aktivierung verwenden und
  - im Outputlayer eine lineare Aktivierungsfunktion nutzen.

- Wir betrachten dies am Beispielnetzwerk von oben. Die erste Schicht berechnet:

$$\mathbf{O}^{(1)} = \mathrm{sig}(\underbrace{\mathbf{W}^{(1)}\mathbf{x}}_{\mathbf{b}}) \tag{1}$$

- Dabei haben wir Gewichte der ersten Schicht zu einer Matrix zusammengefasst,

$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \end{pmatrix} \in \mathbb{R}^{3\times 3}$$

  welche mit den Merkmalen aus der Inputschicht multipliziert wird.

- Das Ergebnis ist ein Vektor $\mathbf{b}$, auf den die Sigmoid-Funktionen – wie in Gleichung (1) notiert – jeweils pro Eintrag von $\mathbf{b}$ angewendet wird, also:

$$\begin{pmatrix} \mathrm{sig}(b_1) \\ \vdots \\ \mathrm{sig}(b_n) \end{pmatrix} = \mathrm{sig}(\mathbf{b})$$

- Dieser Output $\mathbf{O}^{(1)}$ der ersten Schicht ist nun der Input der nächsten Schicht.
- Hierbei wird er seinerseits mit einer Matrix

$$\mathbf{W}^{(2)} = \begin{pmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} & w_{1,3}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} & w_{2,3}^{(2)} \\ w_{3,1}^{(2)} & w_{3,2}^{(2)} & w_{3,3}^{(2)} \\ w_{4,1}^{(2)} & w_{4,2}^{(2)} & w_{4,3}^{(2)} \end{pmatrix} \in \mathbb{R}^{4\times 3}$$

multipliziert und wir erhalten für die nächste Schicht:

$$\mathbf{O}^{(2)} = \mathrm{sig}(\mathbf{W}^{(2)}\mathbf{O}^{(1)})$$

- Das Ergebnis $\mathbf{O}^{(2)}$ dieser Schicht ist nun die Grundlage für die letzte Schicht – den Output Layer – in dem wir wie erwähnt von einer linearen Aktivierungsfunktion ausgehen wollten:

$$\hat{y} = \mathbf{W}^{(3)}\mathbf{O}^{(2)} = \mathbf{W}^{(3)}\,\mathrm{sig}(\mathbf{W}^{(2)}\mathbf{O}^{(1)}) = \mathbf{W}^{(3)}\,\mathrm{sig}(\mathbf{W}^{(2)}\,\mathrm{sig}(\mathbf{W}^{(1)}\,\mathbf{x}))$$

- Man sieht also, dass unsere Vorhersage $\hat{y}$ eine Nacheinanderausführung der Aktivierungsfunktionen im Zusammenspiel mit Matrix-Vektor-Multiplikationen ist.
- Dabei hatten wir gerade die Bias-Neuronen ignoriert.
- Beim Training des Netzwerkes geht es nun darum, die Gewichte optimal zu bestimmen.
- Was heißt nun hierbei *optimal* und wie bekommen wir das hin?
- Darum geht es dann beim nächsten Mal...