Maschinelles Lernen

Als letzten Themenblock dieser Vorlesung wollen wir uns einer Klasse von Algorithmen zuwenden, deren Ziel es ist, künstliche Formen von Wissen zu generieren. Derartiges Wissen soll dabei anhand von Beispielen in einer "Lernphase" angeeignet werden und anschließend auf neue Situationen / Datensätze angewandt werden. Diese Idee des "maschinellen Lernens" gehört damit zu algorithmischen Ansätzen, große Datenmengen ("big data") dahingehend zu analysieren, daß charakteristische Merkmale extrahiert werden und damit eine Dimensionsreduktion durchgeführt wird.

Eine wichtige Rolle spielt die Form der Wissensrepräsentation. Hier unterscheiden wir zwei Typen – symbolische vs. subsymbolische Systeme. In symbolischen Systemen wird das Wissen, d.h. sowohl die Beispiele als auch die induzierten Regeln, explizit repräsentiert. In subsymbolischen Systemen wie sogenannten artifiziellen neuronalen Netzen wird Wissen implizit repräsentiert, d.h. es wird ein berechenbares Verhalten "antrainiert", ein direkter Einblick in die Lösungswege bleibt aber oft verwährt.

Wir wollen uns hier mit letzterem Zugang beschäftigen und künstliche neuronale Netze besprechen. Betrachten wir dazu das folgende Beispiel:

Ziffernerkennung + künstliche neuronale Netze

Wir wollen dem Computer beibringen, Ziffern zu erkennen und damit Postleitzahlen zu lesen

50937

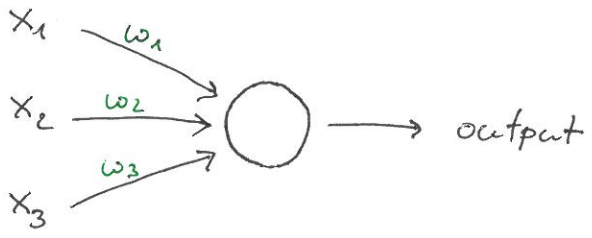
Im visuellen System des Menschen wird diese Aufgabe im primären visuellen Kortex bewältigt – einer Ansammlung von 140 Millionen Neuronen mit über 10 Milliarden Verbindungen. Eine derartige neuronale Struktur wird in künstlichen neuronalen Netzen modelliert und was die Zahl der Neuronen und Verbindungen betrifft erheblich vereinfachten Form implementiert. Tatsächlich reichen schon weniger als 100 Zeilen Code, um ein künstliches neuronales Netz zu implementieren, das Ziffern mit einer Genauigkeit von 96% erkennt. Kommerzielle Codes erreichen eine noch weit höhere Genauigkeit und werden etwa in der Postleitzahlenerkennung seit gut einem Jahrzehnt eingesetzt.

Im folgenden wollen wir drei Hauptaspekte von künstlichen neuronalen Netzen besprechen:

- künstliche Neuronen: Perzeptronen vs. Sigmoide Neuronen
- Netzarchitekturen
- Optimierung neuronale Netzes: überwachtes Lernen

Künstliche Neuronen – Perzeptronen (Frank Rosenblatt 1950-1960's)

Sogenannte Perzeptronen stellen die historisch erste und einfachste Form künstliches Neuronen dar. Ein Perzeptron akzeptiert multiple binäre Eingaben $x_1, x_2, \ldots, x_n$ und produziert einen einzigen binären Output, was wir schematisch darstellen können als



Der binäre Output eines solchen Perzeptrons berechnen wir wie folgt

$$\text{output} = \begin{cases} 0 & \sum_i \omega_i \cdot x_i \leq \text{threshold} \\ 1 & \sum_i \omega_i x_i > \text{threshold} \end{cases}$$

(ω_i: Gewichte)

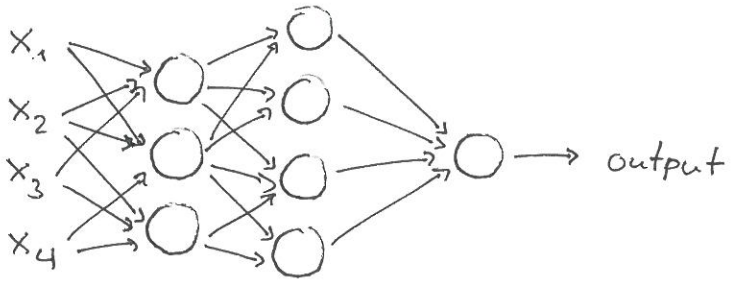
Beispiel: Sie wollen entscheiden, ob Sie am Wochenende zu einem Jazz-Konzert gehen wollen und betrachten dabei folgende Kriterien

1. Ist das Wetter gut? x_1
2. Möchte Ihre Freundin / Ihr Freund mitkommen? x_2
3. Gibt es eine gute ÖPNV-Verbindung? x_3

Option 1: $\omega_1 = 6$, $\omega_2 = 2$, $\omega_3 = 2$ threshold = 5
Sie mögen Jazz wirklich sehr und gehen, wenn das Wetter gut ist.

Option 2: $\omega_1 = 6$, $\omega_2 = 2$, $\omega_3 = 2$ threshold = 3
Sie gehen, wenn gutes Wetter ist oder wenn Ihr Freund mitkommt und ÖPNV gut ist.

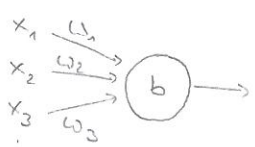
An dem Beispiel erkennen wir, wie ein einziges Perzeptron genutzt werden kann, um eine (relativ einfache) Entscheidung zu treffen. Entsprechend können wir uns jetzt schon vorstellen, daß ein Netzwerk von Perzeptronen komplexere Entscheidungen herbeiführen kann.



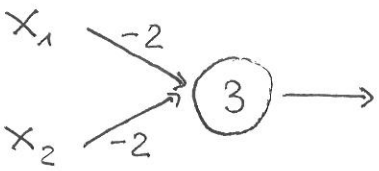
Zur weiteren Formalisierung wollen wir schließlich noch obige Notation für die Berechnung des Outputs vereinfachen

(b: Bias)

$$\text{output} = \begin{cases} 0 & \vec{\omega} \cdot \vec{x} + b \leq 0 \\ 1 & \vec{\omega} \cdot \vec{x} + b > 0 \end{cases}$$



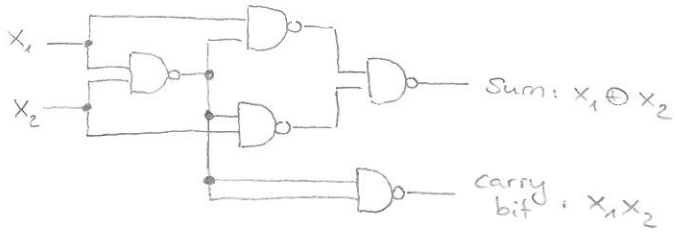
Um die algorithmische Mächtigkeit künstlicher neuronaler Netze abschätzen zu können, ist es wichtig festzustellen, daß Perzeptronen elementare logische Binärfunktionen berechnen können. Dies sei am sogenannten NAND Gate veranschaulicht:

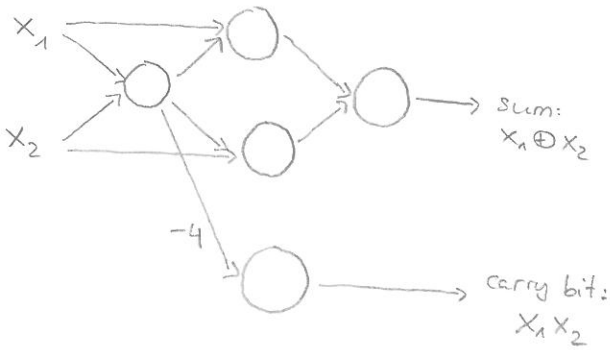


x_1	x_2	output
0	0	1
0	1	1
1	0	1
1	1	0

Das ist eine wichtige Einsicht, denn das NAND-Gate ist ein sogenanntes universelles Gate, d.h. eine beliebig komplexe logische Funktion von n binären Inputs läßt sich auf einen Schaltkreis solcher NAND-Gates reduzieren. Ähnliches können wir also auch auf Netzwerke von Perzeptoren schließen – sie erlauben universelles Rechnen und sind somit genau so mächtig wie alle anderen (universellen) Rechenmodelle.

Beispiel: Binäre Addition

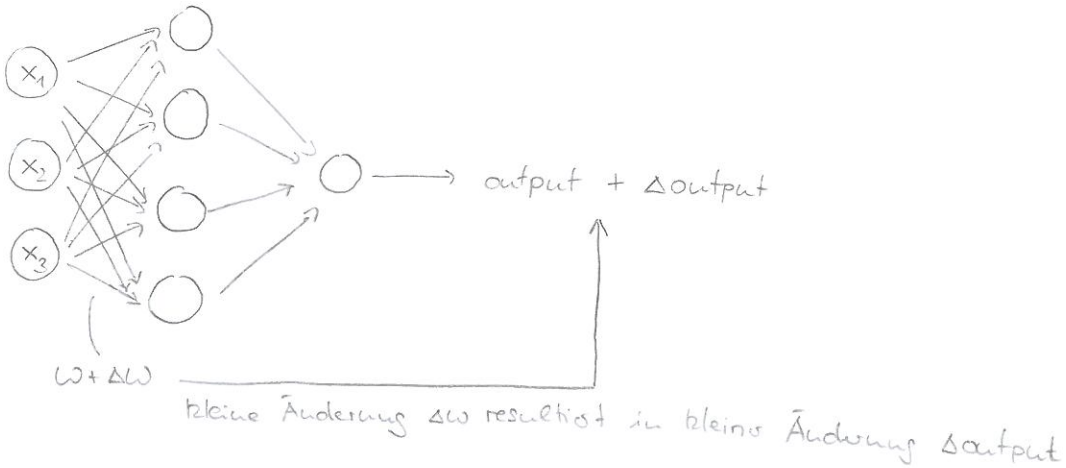




Jenseits ihrer prinzipiellen Universalität liegt die algorithmische Mächtigkeit von neuronalen Netzen allerdings darin, daß ihre Gewichte so optimiert werden können, daß das Netz bestimmte Funktionalität bekommt. Dieser Optimierungsprozess kann automatisiert werden, etwa indem eine große Vielzahl an Datensätzen von Input/Output Parametern betrachtet werden und das Netz, bzw. dessen Gewichte, so optimiert werden, daß es die vorgegebenen Input/Output-Paare reproduzieren kann. Man spricht hierbei von sogenanntem "überwachten Lernen".

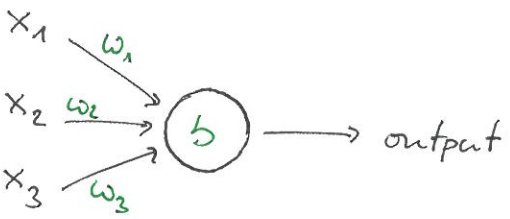
Sigmoide Neuronen

Ein solches Lernprozess erfolgt durch eine inkrementelle Variation der Gewichte. Hiefür ist es essentiell, daß kleine Änderungen eines beliebigen Gewichts auch in eine kleinen Änderung des Outputs resultieren.



Wir werden sehen, daß dies eine notwendige Bedingung sein wird, um automatisiertes Lernen zu ermöglichen.
Bedauerlicherweise ist dies aber für ein Netzwerk von Perzeptoren nicht der Fall! Eine kleine Änderung in einem Gewicht oder Bias kann dazu führen, daß der binäre Output von 0 nach 1 oder umgekehrt flippt – eine erhebliche Änderung! Dies macht es unmöglich, die Gewichte graduell so zu verändern, daß der gewünschte Output für eine große Input-Menge möglich ist.

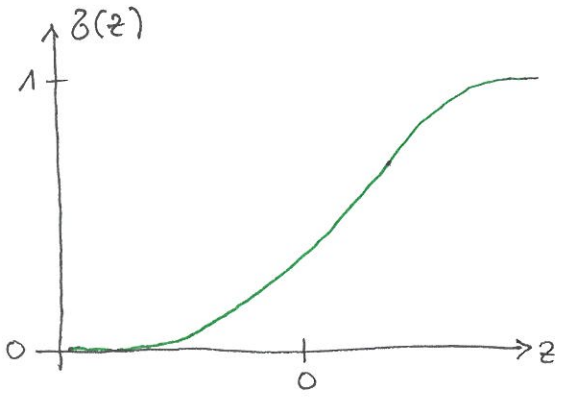
Um dieses Problem zu beheben, verändern wir die Charakteristik eines elementaren Neurons und betrachten sogenannte Sigmoide Neuronen:

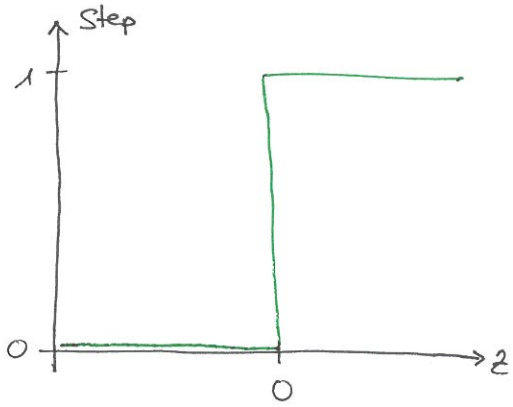


Wiederum betrachten wir n Input Variablen $\vec{x} = (x_1, x_2, \dots, x_n)$, erlauben aber nunmehr daß die Komponenten $x_i \in \mathbb{R}$ in $[0,1]$ sind. Ebenfalls gewichten wir die Eingaben mit Gewichten $\vec{w} = (w_1, w_2, \dots, w_n)$ und schreiben dem Neuron einen Bias-Wert b zu.

Der Output allerdings sei nicht mehr binär, sondern ebenfalls eine reelle Zahl in $[0,1]$. Diesen Output berechnen wir über die sigmoide Funktion $\sigma(z) = \frac{1}{1+e^{-z}}$ als

$$\sigma(\vec{x}, \vec{\omega}, b) = \frac{1}{1 + \exp\left(-\sum_i \omega_i x_i - b\right)}$$





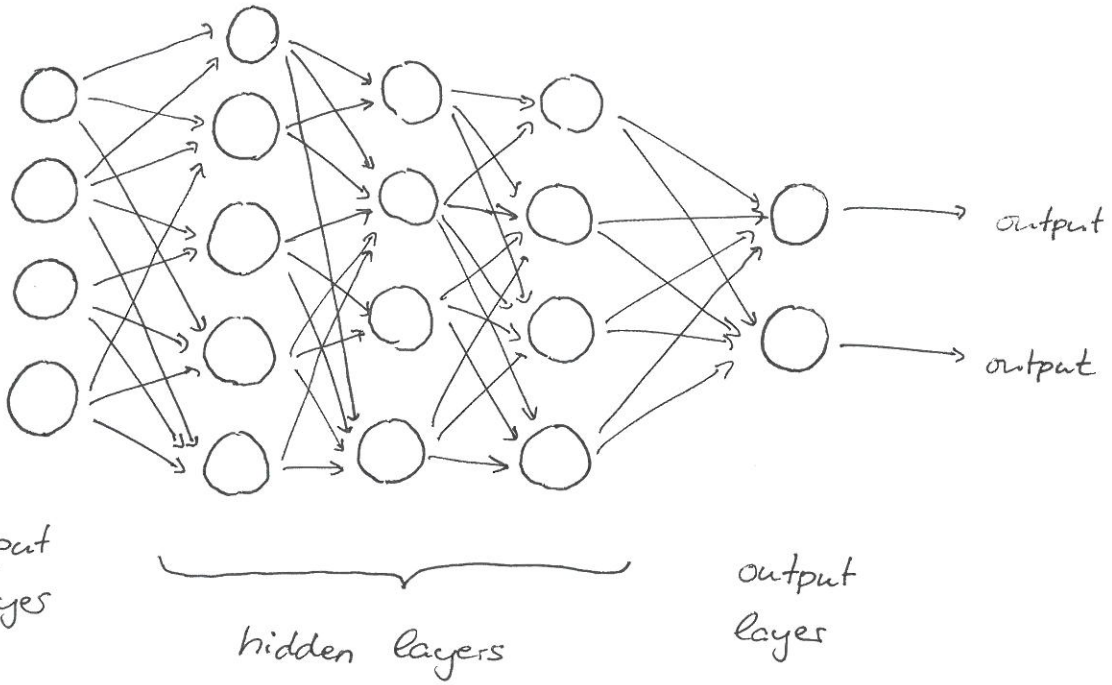
Die sigmoide Funktion $\sigma(z)$ ist eine verbreiterte Stufenfunktion. Im Falle eines solchen Stufenfunktion reduziert sich das sigmoide Neuron auf das Perzeptron. Die genaue Form von $\sigma(z)$ ist tatsächlich nicht wichtig, relevant ist vor allem ihre Stetigkeit, die nunmehr garantiert, daß

$$\Delta\text{output} \approx \sum_i \frac{\partial\text{output}}{\partial \omega_i} \Delta\omega_i + \frac{\partial\text{output}}{\partial b} \Delta b$$

Das bedeutet, daß Δoutput eine lineare Funktion der (kleinen) Änderungen $\Delta\omega_i$ und Δb ist, was eben eine notwendige Voraussetzung für das Lernen, die Optimierung des neuronalen Netzwerks ist.

Architektur von neuronalen Netzwerken

Die Architektur von künstlichen neuronalen Netzen unterteilt sich typischerweise in input layer, sogenannte hidden layers und den output layer:



Die hidden layers bezeichnen dabei all jene Layer zwischen input und output layer. Die einzelnen Layer sind hierarchisch angeordnet - der output eines Layers ist der input des nächsten Layers. Solche Netzwerke ohne geschlossene Loops / Schleifen nennt man feedforward Netzwerke. Netzwerke mit nur einem hidden layer nennt man "flach" und bezeichnet das maschinelle Lernen solcher Netzwerke als "shallow learning". Netzwerke mit multiplen hidden layers hingegen nennt man "tief" und es hat sich hierfür die Bezeichnung "deep learning" eingebürgert. Die genaue Struktur des Netzwerks, etwa in Bezug auf die Zahl der Neuronen pro Layer oder die Art der Verbindungen, wird im allgemeinen heuristisch bestimmt.

Optimierung des Netzwerks – überwachtes Lernen

Abschließend wollen wir uns dem eigentlichen Lernprozess zuwenden. Darunter verstehen wir eine Optimierung der Gewichte auf den Kanten des Netzwerks und den Bias-Werten einzelner Neuronen für eine gegebene Netzwerkarchitektur.

Beim sogenannten überwachten Lernen geschieht dies anhand eines sogenannten Trainingsset von Daten. Für jeden Datensatz x aus diesem Set sei der gewünschte Output $y(x)$ des Netzwerks bekannt. Die Optimierung der Gewichte w und Bias-Werte b kann dann anhand einer Kostenfunktion der Form

$$C(w,b) = \frac{1}{2n} \sum_x \| y(x) - a \|^2$$

($y(x)$: gewünschtes Output; a: tatsächliches Output)

erfolgen, d.h. wir suchen das globale Minimum dieser quadratischen Kostenfunktion $C(w_{opt}, b_{opt}) \approx 0$. Diese Optimierung erfolgt gewöhnlich über einen sogenannten gradient descent Algorithmus, d.h. wir folgen jeweils der größten Ableitung der Kostenfunktion. Dazu ist es unabdingbar effizient die Gradienten der Kostenfunktion für kleine Abweichungen Δw und Δb zu berechnen. Dies ist für die komplexe Netzwerkarchitektur kein offensichtlich triviales Unterfangen, wurde aber durch den sogenannten back propagation Algorithmus in idealer Weise gelöst, auf den wir hier leider nicht mehr eingehen können.

Mit den obigen Erläuterungen haben wir aber nun alle Elemente in der Hand, um erste eigene Schritte zum maschinellen Lernen selbst zu gehen ⟶ siehe Programmier-Tutorial.