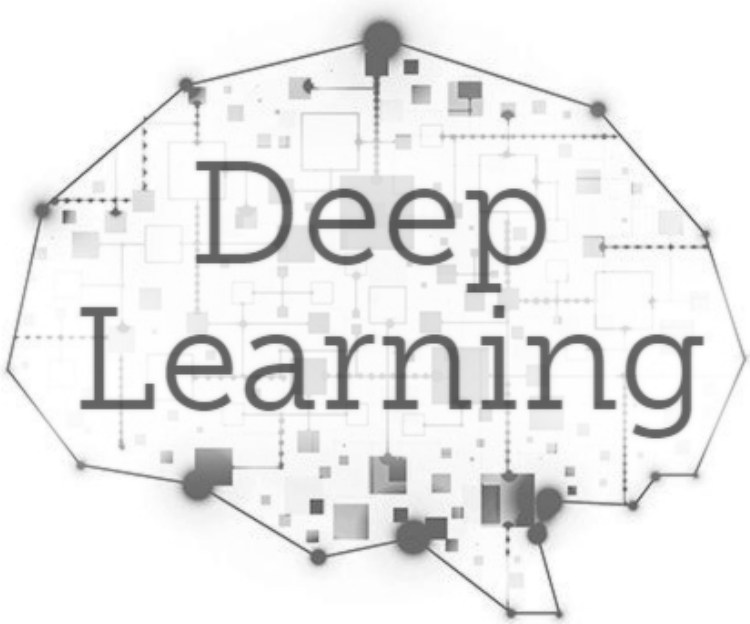


Silas Hoffmann, inf103088
5. Fachsemester
6. Verwaltungssemester

16. März 2020

Thema 2

Seminar

im Sommersemester 2020

Dozent: Prof. Dr. Dennis Säring
Fachbereich Informatik

Fachhochschule Wedel

Inhaltsverzeichnis

Abbildungsverzeichnis

Teil I.
Einführung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Teil II.
Geschichtliche Entwicklung

Im folgenden Abschnitt werde ich etwas auf die geschichtlichen Aspekte von neuronalen Netzen eingehen. Hierbei werden insbesondere die generellen Aspekte der generellen Funktionsweise von älteren Modellen bis hin zur aktuellen Entwicklung verfolgt. Ich werde versuchen die folgenden Leitfragen in diesem Abschnitt zu beantworten:

- Woher kommt Deep Learning und wie ist dieser Begriff im Kontext zur künstlichen Intelligenz einzuordnen?
- Welche Entwicklungen hat das Neuronale Netz von damals zu heute durchgemacht?

Um darauf näher eingehen zu können werden folgende Meilensteine behandelt:
todo Inhaltsverzeichnis (Sections)

1. McCulloch-Pitts-Neuron

1.1. Funktionsweise

Im Jahr 1943 entwickelten Warren McCulloch und Walter Pitts ein Modell welches die Funktionalität eines biologischen Neurons imitieren sollte. In der folgenden Abbildung 1 ist der grobe Aufbau eines Neurons zu sehen.

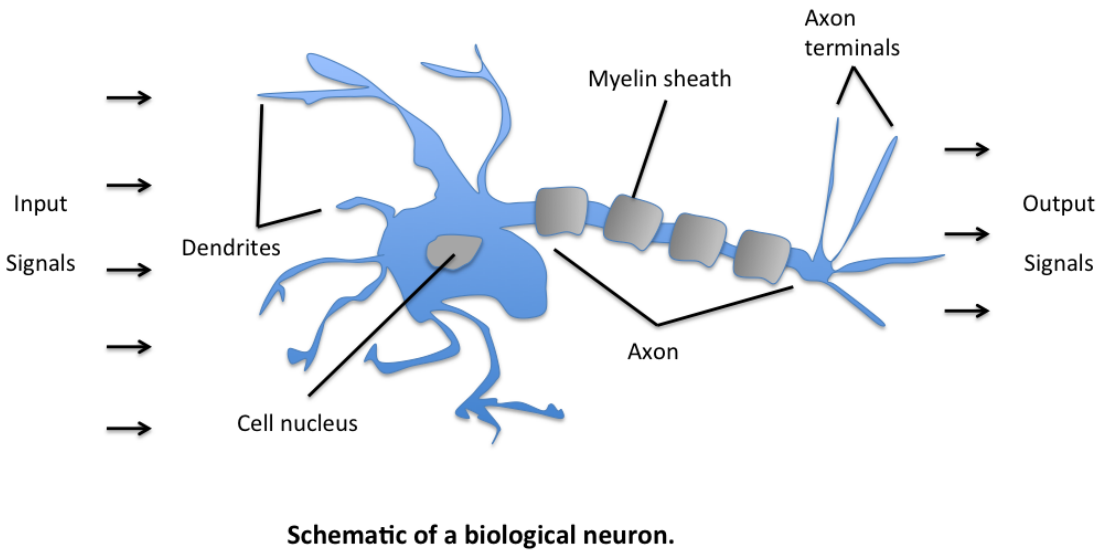
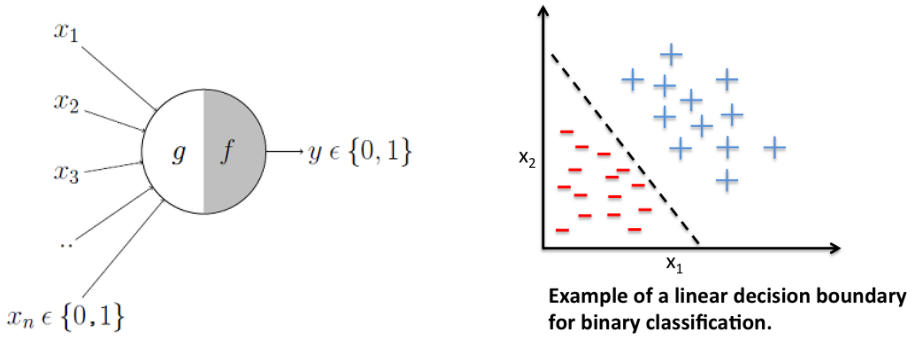


Abbildung 1: McCulloch-Pitts-Zellne - Genereller Aufbau [2]



(a) (b)

Abbildung 2: McCulloch-Pitts-Zelle: Aufbau und Klassifizierung [2]

Die sogenannten *Dendriten* (englisch *dendrites*) nehmen Informationen auf. Sie besitzen Rezeptoren welche in der Lage sind Signale anderer Neuronen aufzunehmen. Diese Signale bewirken elektrische Veränderungen in dem Neuron welche vom Zellkörper (*Soma*) interpretiert / verarbeitet werden. Dieser Zellkörper sammelt alle Informationen und speichert diese im sogenannten *Axonhügel* (engl. Axonhillock) welcher die Ursprungsstelle des *Axons* beziehungsweise *Neuriten* beschreibt. Wenn das gebündelte Signal stark genug sein sollte wird es an den nächsten Teil des Neurons, dem *Axon*, weitergeleitet. Ab diesem Zeitpunkt wird das Signal als *Aktionspotential* bezeichnet und wird über die *Axon* übertragen. Am Ende wird das Signal an diverse *Axonterminale* weitergeleitet welche per Neurotransmitter mit den jeweils nächsten Dendriten verbunden sind.

Dieser biologische Aufbau dient als Grundlage für die Entwicklung des Modells von McCulloch und Pitts. Das Augenmerk ihres Modells liegt in erster Linie darauf Klassifizierungsprobleme zu lösen. Bei einem Klassifizierungsproblem wird *Das zu klassifizierende Objekt X ist dabei durch einen Merkmalvektor $\vec{x}$ aus dem betrachteten Merkmalraum M mit der Dimension n charakterisiert. Das Problem besteht nun darin zu entscheiden, ob das Objekt X in der betrachteten Klasse K liegt.* [1] oder nicht. Der grobe Aufbau eines sogenannten *McCulloch-Pitts-Zelle* ist in **??** zu sehen. Erwähnt sei auch noch, dass man mit diesem Modell lediglich binäre Klassifizierungen mittels einer linearen Entscheidungsfunktion / Aktivierungsfunktion durchführen kann (siehe 2)

Das Modell kann beliebig viele Input-werte aufweisen. Wichtig hierbei: Sie dürfen nur boolescher Natur sein (nur falsch oder wahr). Bei gegebenen Werten führt das Neuron selbst zwei Arbeitsschritte durch:

1. Erst werden alle Werte aufaddiert (in der Abbildung dargestellt durch die Funktion g). Dies imitiert das Verhalten der *Dendriten* in einem biologischen Neuron.

2. Anschließend überprüft die Funktion f ob ein gegebener Schwellwert überschritten wurde oder nicht (gibt dies entsprechend in Form einer booleschen Ausgabe weiter). Das biologische Neuron tut dies mittels des *Axonhügels*.

Die übliche Notation dieses Modells gibt vor, dass der jeweilige Schwellwert jeweils in

die linke Seite des Kreises geschrieben wird während die rechte Seite ausgegraut wird. Im folgenden seien einmal beispielhaft das *AND* und das *OR* Gatter dargestellt. (Für weitere Beispiele siehe Unterabschnitt A.1)

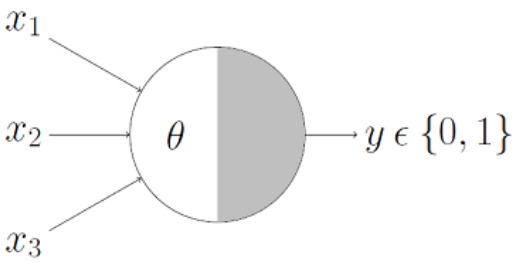


Abbildung 3: Biologische Neuronen - Notation [2]

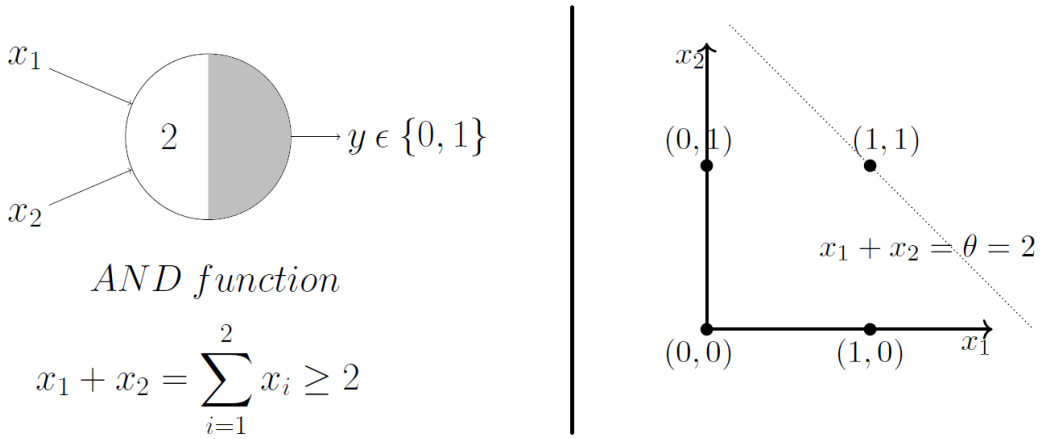


Abbildung 4: McCulloch-Pitts-Zelle - AND Gatter [2]

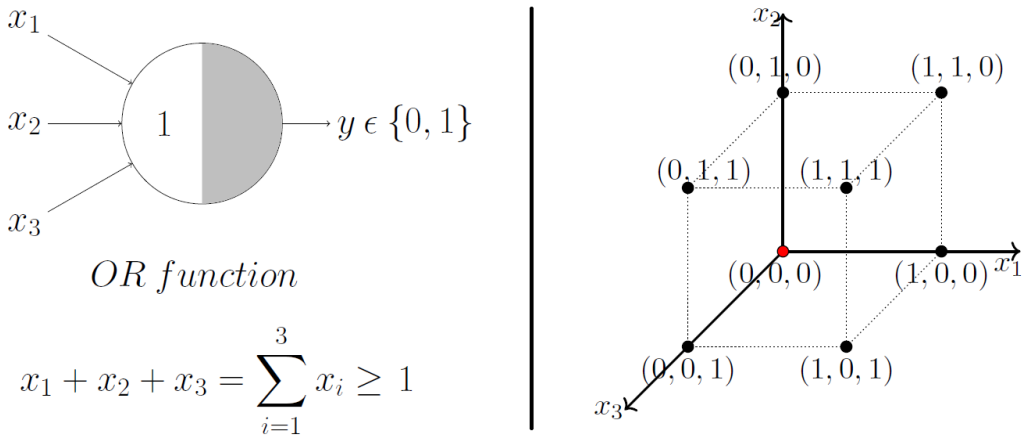


Abbildung 5: McCulloch-Pitts-Zelle - OR Gatter [2]

1.2. Nachteile bzw. Verbesserungspotenzial

- Dieses Modell erlaubt wie gesagt nur boolesche Eingabewerte, viele Modelle erfordern allerdings kontinuierliche Werte. Mit diesen wäre es zum Beispiel deutlich einfach ein Bild oder Ähnliches zu analysieren.
- Die Schwelle (Theta) muss stets manuell bestimmt werden. Einen Trainingsalgorithmus wie man ihn von heutigen Ansätzen her kennt gibt es in diesem Modell nicht.
- Es gibt keinerlei Priorisierungsmöglichkeit zwischen den einzelnen Eingabewerten. Jeder hat einen gleichgroßen Einfluss auf das Ergebnis, so etwas wie ein Auschlußkriterium gibt es hierbei also nicht.
- Es ist nicht möglich *gedeckelt* Gatter wie zum Beispiel ein *XOR* abzubilden. Bei einem Neuron mit zwei Input müsste zum Beispiel ein Schwellwert von 1 genau getroffen werden. Dieses Modell ist allerdings nur in der Lage zu entscheiden ob ein Schwellwert *überschritten* wurde oder nicht.

2. Perceptron

2.1. Aufbau und Notation

Im Jahr 1958 entwickelte der US-amerikanischer Psychologe und Informatiker Frank Rosenblatt das sogenannte *Perceptron*. Dieses stellt das älteste neurale Netz dar welches teilweise auch heutzutage noch genutzt wird. Inspiriert wurde Rosenblatt vom Auge einer Fliege wobei die Entscheidung der nächsten Flugrichtung in Teilen bereits im Auge stattfindet. Das Perceptron stellt in diesem Zusammenhang eine direkte Abbildung dieser Beobachtung dar.

Das Modell ist eine Weiterentwicklung von der McCulloch-Pitts-Zelle (siehe Abschnitt 1). Allerdings ist das Perceptron in der Lage die unterschiedlichen Eingabewerte zu priorisieren. Dies geschieht mittels reeller Gewichte welche jeweils mit den Inputwerten verrechnet werden: $\sum_j w_j x_j$. Gleich bleibt jedoch die binäre Klassifizierung wie schon bei der McMulloch-Pitts-Zelle.

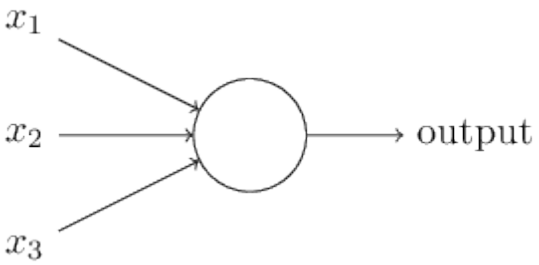


Abbildung 6: Perceptron - drei Eingabewerte [5]

Die Abbildung 6 beschreibt ein einfaches Perceptron mit drei Eingabewerten. Genau wie die MP-Zelle verwendet dieses Modell einen Schwellwert θ um den letztendlichen Ausgabewert zu bestimmen, jedoch möchte ich hier noch einmal etwas genauer auf die Notation des Ganzen eingehen, da diese auch in späteren Abschnitten benötigt wird.

Die Funktion welche berechnet ob ein Schwellwert überschritten wird oder nicht wird *Aktivierungsfunktion* genannt (hier mit g bezeichnet).

$$g(\mathbf{z}) = \begin{cases} 0 & \text{if } \mathbf{z} \leq \theta \\ 1 & \text{if } \mathbf{z} > \theta \end{cases} \tag{1}$$

wobei gilt

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \tag{2}$$

$$\mathbf{z} = w_1 x_1 + \cdots + w_m x_m = \sum_{j=1}^{m} x_j w_j = \mathbf{w}^T \mathbf{x} \tag{3}$$

Sämtliche Gewichte und Eingabewerte können als Vektoren betrachtet werden. Die Summe der Produkte kann dadurch wiederum schlicht als *Vektorpunktprodukt* verstanden werden (2).

Geplottet sieht die Aktivierungsfunktion g übrigens folgendermaßen aus (siehe **??**):

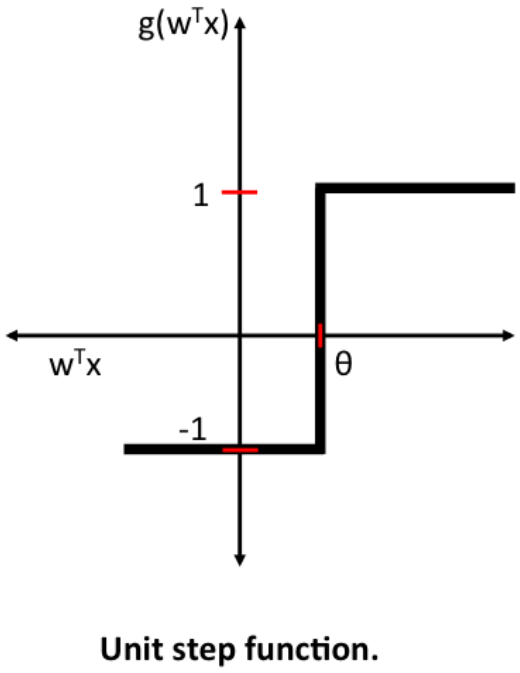


Abbildung 7: Perceptron - Einheits-Sprungfunktion [2]

Um die generelle Notation des gesamten Modells zu vereinfachen bietet es sich außerdem an den Schwellwert θ auf die linke Seite der Gleichung zu ziehen (siehe 1). Damit gelten folgende Rahmenbedingungen:

$$g(\mathbf{z}) = \begin{cases} 0 & \text{if } \mathbf{z} \leq 0 \\ 1 & \text{if } \mathbf{z} > 0 \end{cases} \tag{4}$$

wobei gilt

$$\mathbf{z} = \mathbf{w_0 x_0} + w_1 x_1 + \cdots + w_m x_m = \sum_{j=1}^{m} x_j w_j = \mathbf{w}^T \mathbf{x} \tag{5}$$

Wichtig hierbei, es wird ein zusätzliches Gewicht welches den negativen Schwellwert hält, sowie einen zusätzlicher Inputwert mit dem Wert 1, eingeführt ($w_0 = -\theta$ und $x_0 = 1$). Bei der Berechnung (siehe Gleichung 4) fließt dieser Faktor nun als negativer Summand mit ein wodurch man nur noch prüft ob die Gesamtsumme kleiner Null ist oder nicht. Von dieser Notation wird insbesondere bei der *Lernregel* (siehe Abschnitt 2.2) Ge-

brauch gemacht. Mittels dieser Vereinfachung lässt sich ein Modell auch folgendermaßen darstellen:

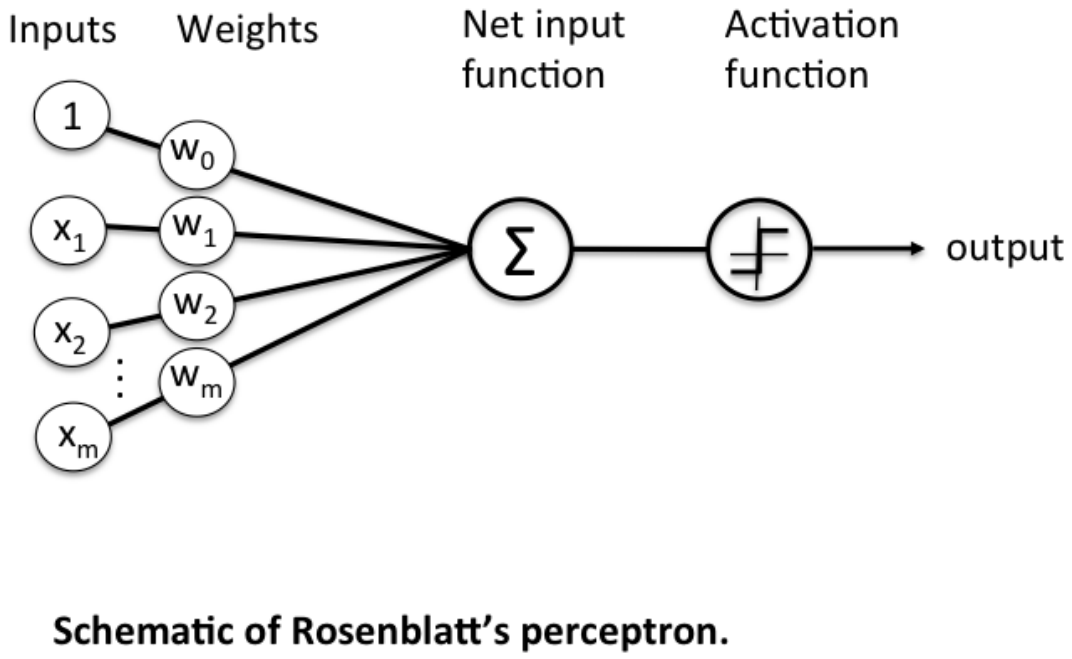


Abbildung 8: Perceptron - Modelansicht [2]

Letztendlich sei jedoch erwähnt, dass es in der Literatur auch öfters eine Darstellung mittels eines *Bias* gibt, siehe Gleichung Gleichung 6 [5]

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{6}$$

2.2. Lernregel

Erklärung Das bisher vorgestellte Modell beinhaltet bis jetzt wenig Eigenschaften für einen *lernenden* Algorithmus, dies ändert sich jedoch mit der Idee von Rosenblatt das Modell selbst die Angleichung der Gewichte übernehmen zu lassen. Hierzu wird auf ein eine Menge von Trainingsdatensätzen zurückgegriffen. Diese Datensätze bestehen aus Eingabewerten für das System und der korrespondierenden Ausgabe. Die Lernregel selbst sieht zusammengefasst folgendermaßen aus.

- Initialisiere die Gewichte mit einem sehr kleinen Wert oder dem Wert Null.
- Für jeden Datensatz der Menge:
  - Berechne den Ausgabe-Wert des Systems
  - Gleiche die Gewichte an

Der genannte Ausgabewert wird wie bereits angedeutet über die Aktivierungsfunktion des Modells bestimmt. Die einzelnen Komponenten des *Gewichtevektors* werden getrennt betrachtet und angeglichen. Ein Aktualisieren eines einzelnen Gewichts innerhalb des Gewichtsvektors wird formal mit $w_j := w_j + \Delta w_j$ beschrieben (das Dreieck kennzeichnet, dass es sich um eine *Veränderung* handelt). Da die beschriebene Lernregel inkrementell arbeitet muss man eine sogenannte *Lernrate* bestimmen. Dieser Wert bestimmt letztendlich wie stark ein Gewicht bei einer Iteration verändert wird. Formal wird diese Rate mit dem Zeichen η dargestellt. Die eigentliche Formel zum Angleichen der Gewichte sieht folgendermaßen aus:

$$\Delta w_j = \eta \, (\text{target}^{(i)} - \text{output}^{(i)}) \, x_j^{(i)} \tag{7}$$

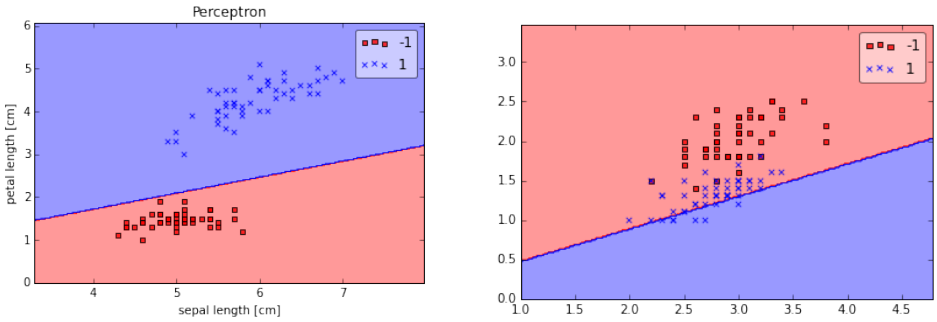
Zuerst wird ein Datensatz gewählt durch dessen Ausgabewerte alle Gewichte angeglichen werden sollen. Generell gilt, dass das i in den Klammern nicht als Exponent sondern als Index für den betrachten Trainingsdatensatz steht. Dann wird die Differenz des optimalen Ausgabewerts und des erreichten Werts berechnet. Diese Differenz wird anschließend mit der Lernrate und dem korrespondierenden Eingabewert des betrachteten Gewichts multipliziert. Dies wird vielleicht etwas klarer wenn man noch einmal einen Blick in das Diagramm 8 auf Seite 11 wirft. Der Eingabewert $x_j^{(i)}$ steht also für die Komponente mit dem Index j in dem Eingabevektor des Trainingsdatensatzes an der Stelle i innerhalb der Menge von Trainingsdaten.

Anwendung Für einen zweidimensionalen Trainingsdatensatz [1] würde die Lernregel folgendermaßen aussehen. Hervorzuheben sei jedoch, dass alle Gewichte gleichzeitig angeglichen werden.

$$\begin{aligned}
\Delta w_0 &= \eta(\text{target}^{(i)} - \text{output}^{(i)}) \\
\Delta w_1 &= \eta(\text{target}^{(i)} - \text{output}^{(i)}) \, x_1^{(i)} \\
\Delta w_2 &= \eta(\text{target}^{(i)} - \text{output}^{(i)}) \, x_2^{(i)}
\end{aligned} \tag{8}$$

In dieser Darstellung wurde die bereits in 2.1 besprochene Notation verwendet bei der ein zusätzliches Gewicht mit dem Schwellwert eingeführt wurde. Da der eigentliche Eingabewert hierbei lediglich den Faktor *1* besitzt fällt er hierbei einfach weg. Wir haben also neben diesem einbezogenen *Bias* lediglich die zwei weiteren Gewichte die angeglichen werden.

[1] Ein zweidimensionaler Datenpunkt besitzt einen zweidimensionalen Eingabevektor (also genau zwei Eingabewerte).



(a) (b)

Abbildung 9: Perceptor - Problematische Klassifizierung [2]

Für den Fall, dass das Modell einen Datensatz richtig klassifiziert gibt es genau zwei Möglichkeiten:

$$\begin{aligned} \Delta w_j &= \eta(1^{(i)} - 1^{(i)})\, x_j^{(i)} = 0 \\ \Delta w_j &= \eta(1^{(i)} - 1^{(i)})\, x_j^{(i)} = 0 \end{aligned} \tag{9}$$

Durch den Differenzfaktor von Null verändert sich das betrachtete Gewicht erwartungsgemäß nicht. Bei dem entsprechenden Gegenteil ist dies nicht der Fall, hier könnte es zum Beispiel so aussehen:

$$\begin{aligned} \Delta w_j &= \eta(1^{(i)} - -1^{(i)})\, x_j^{(i)} = \eta(2)\, x_j^{(i)} \\ \Delta w_j &= \eta(-1^{(i)} - 1^{(i)})\, x_j^{(i)} = \eta(-2)\, x_j^{(i)} \end{aligned} \tag{10}$$

Eine Implementierung des Beschriebenen Konzepts in der Programmiersprache Python ist auf Github [3] zu finden. Ich werde hier jedoch nicht weiter auf die Details der Implementierung eingehen.

Wie schon die McCulloch-Pitts Zelle (siehe Abbildung 2) ist das Perceptron nur in der Lage mittels einer linearen Klassifizierungs-Funktion die zwei unterschiedlichen Gruppen auseinander zu halten (siehe Abbildung 9).

3. Adeline

3.1. Aufbau

Im 1959 entwickelten der Stanford Professor Bernard Widrow und der Elektroingeneur Marcian Edward Hoff das sogenannte *Adilne-Modell*. Der Name ist ein Akronym für *ADAptive LINear Element*. Dieses Modell basiert auf dem Perceptron mit dem Unterschied, dass dieses Modell auf die Einheits-Sprungfunktion, wie sie das Peceptron verwendet, bei der Angleichung der Gewichte verzichtet. Es wird stattdessen eine lineare Aktivierungsfunktion $g(\mathbf{z})$ verwendet welche in diesem Fall erstmal mit der Identitätsfunktion besetzt wird (es gilt also $g(\mathbf{w}^T\mathbf{x}) = \mathbf{w}^T\mathbf{x}$). Außerdem wird eine Entscheidungsfunktion an das Ende des ganzen Modells gehängt um weiterhin die Werte quantifizieren zu können. Diese hat jedoch keinen Einfluss auf den Trainingsalgorithmus.

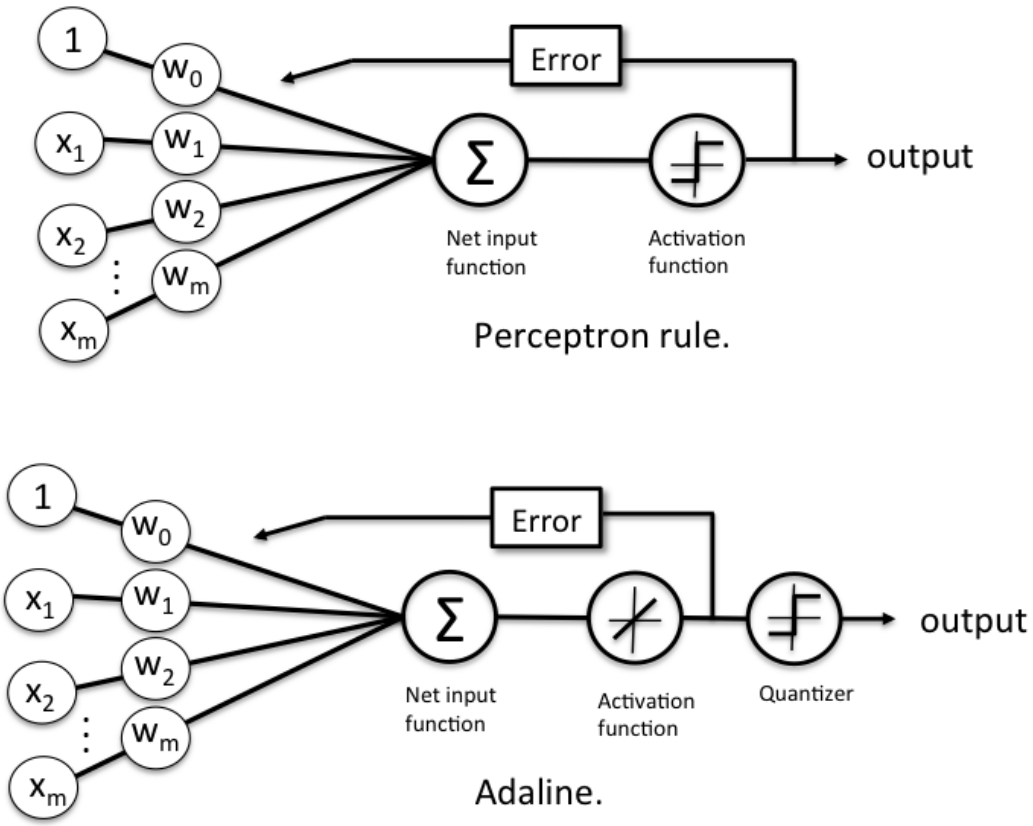


Abbildung 10: Adeline - Aufbau [2]

3.2. Lernregel

Widrow und Hoff definierten die Delta-Regel für den Lernalgorithmus ihres Modells. Dieser ist auch unter dem Namen *Least-Mean-Square-Algorithmus* bekannt und ist auch heute noch von Relevanz). Im Kern móchte man hierbei das Minimum einer Kostenfunktion über dem Modell bestimmen. Ich werde im folgenden Abschnitt darauf eingehen wie dieser funktioniert und wie genau er für dieses Modell eingesetzt werden kann.

todo Intro nochmal überarbeiten...

Gradient Descent Der wesentliche Nachteil der Einheits-Sprungfunktion ist der, dass sie nicht stetig und und damit nicht differenzierbar ist. Deswegen hat man sich beim Lernalgorithmus des Adeline-Modells dazu entschieden stattdessen die Identitätsfunktion zu verwenden. Es wird zuerst eine Kostenfunktion $J(\mathbf{w})$ definiert die minimiert werden soll. Die Kostenfunktion wird durch die *Regressionsquadratsumme* [2] definiert. Die Formel sieh folgendermaßen aus:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i}(\text{target}^{(i)} - \text{output}^{(i)})^2 \qquad \text{output}^{(i)} \in \mathbb{R} \tag{11}$$

Wichtig hierbei, der Vorfaktor $\frac{1}{2}$ gehört nicht zur herkömmlichen Regressionsquadratsumme, wurde hier jedoch hinzugefügt um später einfach ableiten zu können.

[2] engl. *sum of squared errors* (SSE)

Teil III.
Aktuelle Entwicklung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Literatur

[1] Definition Klassifizierungssproblem. http://ekpwww.physik.uni-karlsruhe.de/~tkuhr/HauptseminarWS1112/Keck_handout.pdf. Aufgerufen am: 15-03-2020.

[2] McCulloch-Pitts Neuron. https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1. Aufgerufen am: 14-03-2020.

[3] Perceptor - Python Implementierung. https://github.com/rasbt/mlxtend/blob/master/mlxtend/classifier/perceptron.py. Aufgerufen am: 16-03-2020.

[4] Single-Layer Neural Networks and Gradient Descent. https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html. Aufgerufen am: 14-03-2020.

[5] M.A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

A. Anhang

A.1. McCulloch-Pitts-Zelle

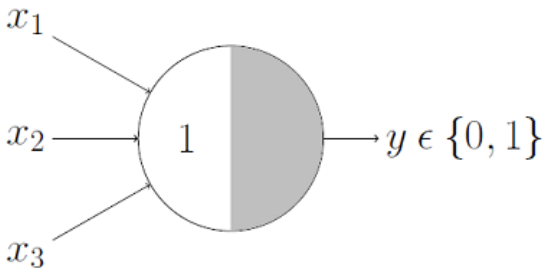


Abbildung 11: McCulloch-Pitts-Zelle - OR Gatter 2 [2]

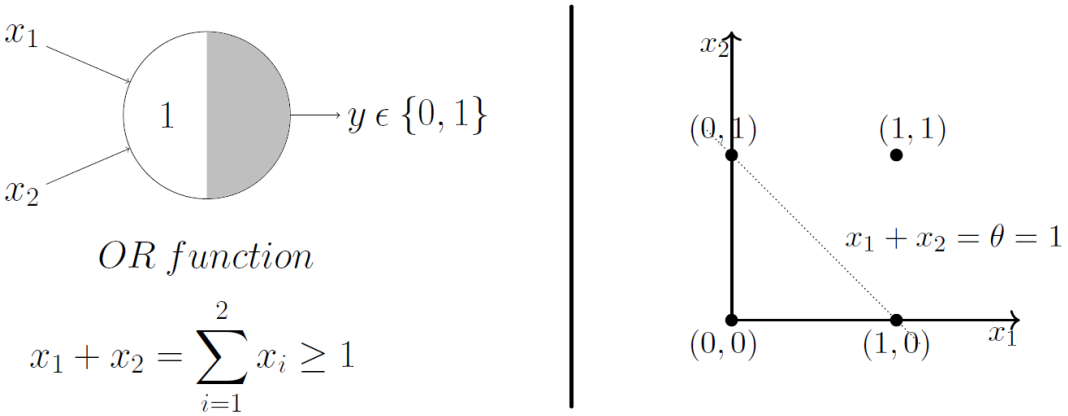


Abbildung 12: McCulloch-Pitts-Zelle - OR Gatter 3 [2]

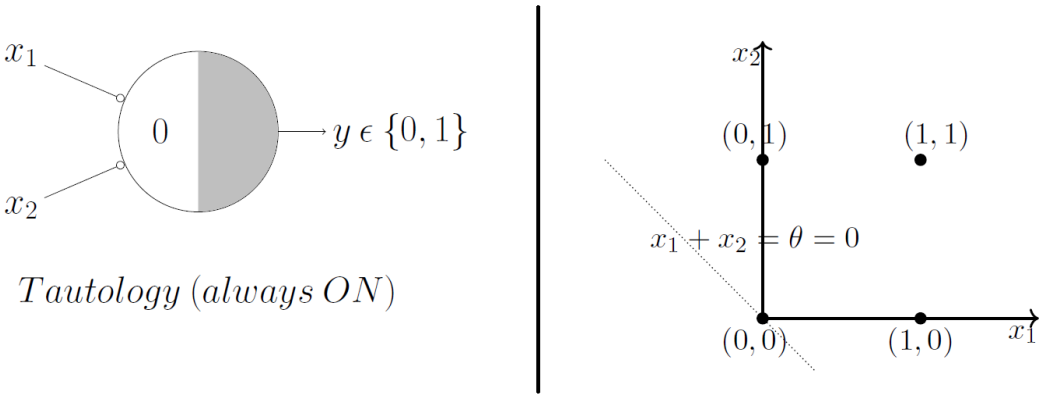


Abbildung 13: McCulloch-Pitts-Zelle - Tautologie [2]