Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim Geisenheim

Fachbereich Design Informatik Medien
Studiengang Master Informatik

Künstliche neuronale Netze

Das Perzeptron

Sebastian Otte

Dezember 2009

1 Grundlegendes

Als Perzeptron bezeichnet man eine Form von künstlichen neuronalen Netzen, die dem Modell von Frank Rosenblatt in [Ros58] folgen. In seiner ursprünglichen Variante besteht ein Perzeptron nur aus einem einzigen Neuron, dem *einfachen Perzeptron*[1]. Inzwischen haben sich aber auch komplexere Modelle von Perzeptronen etabliert, bei denen mehrere einfache Perzeptron-Neuronen zu einem Netz zusammen geschlossen werden. Klassischerweise werden die Neuronen dabei in sogenannten *Layern* (Schichten) angeordnet, wobei die Neuronen innerhalb eines Layer i.d.R. nicht untereinander verbunden sind, sondern meist ausschließlich mit den Neuronen des direkt folgenden Layers. Wir beschränken uns hier auf Perzeptronen, bei denen jedes Neuron eines Layers mit jedem Neuron des nachfolgenden Layers verbunden ist. Ein solches Perzeptron bezeichnet man auch als *FeedForward-Netzwerk* (vgl. [Hei94]).
Perzeptronen werden u.a. bei der Erkennung von Mustern eingesetzt. Dabei wird ein Perzeptron auf bestimmte Eingabemuster trainiert und ist danach in der Lage ähnliche Muster zu erkennen.

[1] Das Perzeptron basiert wiederum auf der McCulloch-Pitts-Zelle nach [MP43].

2 Perzeptron Modell

Bevor wir uns mit mehrzelligen Perzeptronen beschäftigen, werden wir das Grundmodell des einfaches Perzeptrons näher be-

trachten (vgl. hierzu [Hei94]). Das Ein-

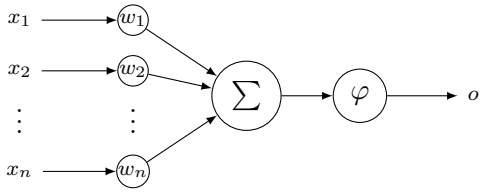


Abbildung 1: Einfaches Perzeptron

fache Perzeptron, wie es in Abbildung 1 dargestellt ist, verfügt über n Eingänge $x_1, \ldots, x_n \in \mathbb{R}$, mit jeweils einer Gewichtung $w_1, \ldots, w_n \in \mathbb{R}$. Zudem gibt es einen Ausgang $o \in \mathbb{R}$.

2.1 Eingangsfunktion

Für die Berechnung der Ausgabe wird zunächst die gewichtete Summe der angelegten Eingabewerte gebildet:

$$\sigma = \sum_{i=1}^{n} x_i w_i. \tag{2.1}$$

Diese erste Verarbeitung bezeichnet man auch als die *Eingangsfunktion* des Neurons.

2.2 Aktivierungsfunktion

Nachdem nun σ bekannt ist, wird die Ausgabe o mit Hilfe der *Aktivierungsfunktion* φ bestimmt:

$$o = \varphi(\sigma). \tag{2.2}$$

Für φ kommen eine ganze Reihe unterschiedlicher Funktionen in Frage. Eine Möglichkeit ist hier die Verwendung einer einfachen Schwellenwertfunktion f_θ mit einem kostanten Schwellenwert θ:

$$f_\theta(x) = \begin{cases} 1, & \text{falls } x \geq \theta, \\ 0, & \text{sonst.} \end{cases} \tag{2.3}$$

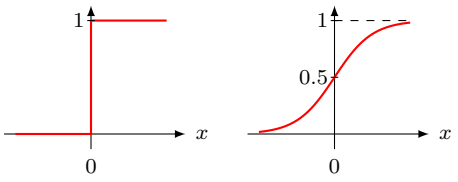
Eine solche Funktion, auch *Stufenfunktion* genannt, führt zu einer „scharfen“ Aktivierung des Neurons. Die Stufenfunktion ist jedoch nicht differenzierbar[2]. Daher verwendet man meist, insbesondere für mehrschichtige Perzeptronen, die diffenzierbare *sigmoid-Funktion*, welche wie folgt definiert ist:

$$sig(x) = \frac{1}{1 + e^{-x}}. \tag{2.4}$$

Die entsprechende Ableitung lautet:

$$sig'(x) = sig(x)(1 - sig(x)). \tag{2.5}$$

Wie in Abbildung 2 ersichtlich ist, führt die sigmoid-Funktion auch zu einer weicheren Aktivierung des Neurons.



(a) Schwellenwert-Funktion mit $\theta = 0$ (b) sigmoid-Funktion

Abbildung 2: Aktivierungsfunktionen

[2] Die Differenzierbarkeit der Aktivierungsfunktion ist wichtig für den im weiteren Verlauf erläuterten Lernalgorithmus

3 Layer

Interessant wird nun die Vernetzung vieler einzelner Neuronen. Wie bereits erwähnt, werden Neuronen in einem mehrzelligen Perzeptron in Layern angeordnet. Dabei unterscheidet man drei verschiedene Arten von Layern:

- Input-Layer: enthält keine Neuronen im eigentlichen Sinne, sondern trägt lediglich die *Netzeingabe*.
- Output-Layer: präsentiert die *Netzausgabe*.
- Hidden-Layer: verdeckte Schicht(en) zwischen Input- und Output-Layer.

Die verschiedenen Schichten eines Perzeptrons mit den entsprechend verbundenen Neuronen sind in Abbildung 3 dargestellt.

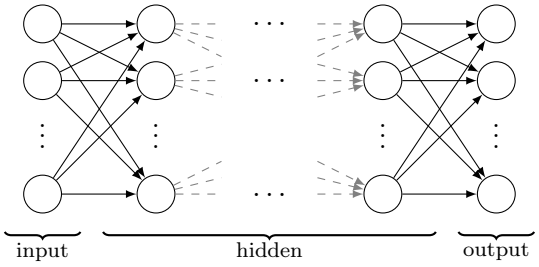


Abbildung 3: Die Layer eines Perzeptrons.

3.1 Neuronen und Verbindungen

Bisher haben wir rein informell von Verbindungen zwischen Neuronen gesprochen, jedoch brauchen wir für spätere Erläuterungen eine entsprechende formale Notation. Diese werden wir im Folgenden schrittweise einführen:
Zunächst bezeichnen wir die Menge der Neuronen (Units) eines Netzes als U mit:

$$U = \{u_1, u_2, ..., u_n\}. \tag{3.6}$$

Entsprechend der unterschiedlichen Arten von Layern definieren wir zusätzlich als kleine Hilfsmengen:

- U_I = die Menge der Eingabeneuronen,
- U_H = die Menge der versteckten Neuronen,
- U_O = die Menge der Ausgabeneuronen.

Die Verbindungen zwischen den Neuronen werden in der Menge der Verbindungen (Connections) C erfasst:

$$C = \{(i, j)|u_i \rightarrow u_j\}, \tag{3.7}$$

dabei sei $\rightarrow$ zu lesen als „vorwärts verbunden mit“. Desweiteren definieren wir C_j als die Menge der Neuronen (Indizes), die Eingang von u_j sind (die vorwärts verbunden sind mit u_j) mit:

$$C_j = \{i|(i, j) \in C\}. \tag{3.8}$$

Analog dazu sei C^j definiert als die Menge der Neuronen (Indizes), die u_j als Eingang haben (mit denen u_j vorwärts verbunden ist) mit:

$$C^j = \{k|(j, k) \in C\}. \tag{3.9}$$

Die Abbildung 4 stellt die Mengen C_j und C^j eines Neurons u_j anschaulich dar.

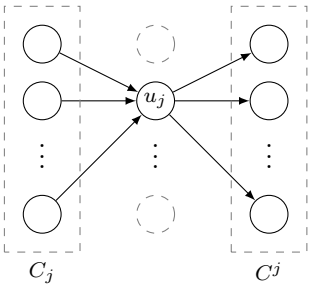


Abbildung 4: Die Mengen C_j und C^j.

3.2 Gewichtungen

Die eigentliche Information, die ein Perzeptron-Netz enthält, wird in Form von Gewichtungen (Faktoren) kodiert. Dabei gibt es für jede Verbindung zwischen zwei Neuronen genau einen solchen Faktor:

$$w_{ij} \in \mathbb{R} \text{ mit } (i,j) \in C. \tag{3.10}$$

Durch die vorangegangenen Definitionen können wir σ aus (2.1) für u_j nun wie folgt schreiben:

$$\sigma_j = \sum_{i \in C_j} o_i w_{ij}, \tag{3.11}$$

wobei wir anstelle von x_i direkt die Ausgabe o_i des *Vorgängerneurons* u_i einsetzen. Für o_j ergibt sich also:

$$o_j = \varphi(\sigma_j). \tag{3.12}$$

Die in (3.10) definierente Gewichtungen nennt man auch *Gewichtungenmatrix*. Hier sei noch mal deutlich hervorgehoben, dass Input-Neuronen keine Gewichtungen in Sinne von Abbildung 1 haben. Daher fällt der Input-Layer bei der Benennung der *Lagigkeit* eines Perzeptrons weg, bespielsweise nennt man ein Perzeptron mit einem Input- und einem Output-Layer *einlagig* oder *einschichtig*.

4 Propagation

Als *Propagation* oder auch *Testen des Netzes* bezeichnet man den Vorgang, bei dem eine Eingabe an den Input-Layer angelegt und durch das gesamte Netz, also durch alle Layer bis zum Output-Layer, „hindurch progagiert“ wird (vgl. [Kri07]). Dieser Prozess verläuft dabei in folgenden Schritten:

1. Eingabe wird in den Input-Layer kopiert.
2. Für jeden Hidden-Layer wird nun, beginnend mit dem ersten, für jedes enthaltene Neuron mit der Vorschrift aus (3.12) die Ausgabe bestimmt. Dabei dient die Ausgabe der gerade berechneten Schicht als Eingabe der nächsten Schicht.
3. Es wird für jedes Neuron im Output-Layer mit (3.12) die Ausgabe bestimmt. Die Eingaben für die Berechnung sind entweder die berechnete Ausgabe des letzten Hidden-Layers, oder in einem einschichtigen Perzeptron die Werte des Input-Layers.

4. Zuletzt kann die *Netzausgabe* am Output-Layer abgelesen werden.

Die Schritte der Propagation sind beispielhaft in Abbildung 5 skizziert. Das Beispielnetz ist zweischichtig mit drei Eingabe-Neuronen, einem Hidden-Layer mit zwei Neuronen und einem Output-Layer mit einem Neuron.

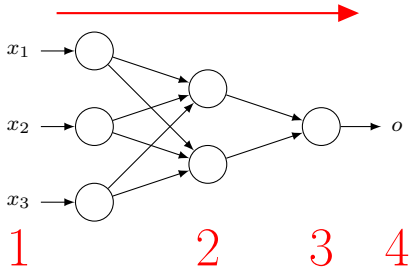


Abbildung 5: Propogation Beispiel.

Anmerkung: Ein Perzepton kann auch als *Assoziativ-Speicher* oder als *BlackBox-Funktion* betrachtet werden, da es einen Eingabevektor auf einen Ausgabevektor abbildet (vgl. [Hei94]).

5 Lernen

Das Lernen in einem Perzeptron-Netz geschieht über die Manipulation der Gewichtungenmatrix. Da eine willkürliche Belegung der einzelnen Gewichte mit Sicherheit kein befriedigendes Ergebnis verspricht, muss mit System vorgegangen werden. Dazu führen wir den Begriff *Fehler des Netzes* ein.

5.1 Fehler

Der Fehler des Netzes quantifitiert die Abweichung der tatsächlichen Netzausgabe zu einer gewünschten oder erwarteten Netzausgabe, nachdem die zugehörige Eingabe durch das Netz propagiert wurde. Als klassische *Abschweichungs-* oder *Fehlerfunktion* wird der *quadratische Fehler* SSE (engl. *Sum of Square Error*) eingesetzt (vgl. [RR95], [Hei94]). Für ein Perzeptron könnte die SSE-Funktion wie folgt aussehen:

$$SSE = \sum_{i|u_i \in U_O} (p_i - o_i)^2, \qquad (5.13)$$

wobei p_i den erwarteten Wert für das Output-Neuron u_i und o_i dessen tatsächlichen Wert angibt.
Da ein Perzeptron i.d.R. nicht nur eine, sondern mehrere Eingaben kennen soll, werden wir die Fehlerfunktion etwas erweitern. Dafür sei T definiert als die Menge der *Trainingsdatensätze*. Als neue Notation schreiben wir für ein $t \in T$:

- $o_i(t)$ meint den Wert des Ausgabe-Neurons u_i, nachdem der Datensatz t durch das Netz propagiert wurde und
- $p_i(t)$ meint den erwarteten Wert des Ausgabe-Neurons u_i nach Propagation von t.

Somit können wir die SSE-Funktion über alle Trainingsdatensätze formulieren als:

$$SSE_T = \sum_{t \in T} \sum_{i|u_i \in U_O} (p_i(t) - o_i(t))^2. \qquad (5.14)$$

Grundsätzlich ist das Ziel des Lernens nun, diesen Fehler zu minimieren, denn je geringer der Fehlerwert ist, desto ähnlicher ist die tatsächliche Netzausgabe der erwarteten Netzausgabe.
Bei Perzeptronen verwendet man üblicherweise einen angestrebten *minimalen Fehlerwert*, den wir hier als ξ bezeichnen. Das Netz wird dann solange mit der Trainingsmenge T trainiert bis gilt $SSE_T \leq \xi$. Die Größe von ξ ist dabei abhängig von der Art des Problems und der gewünschten Genauigkeit. In der Literatur hat sich für ξ eine Größenordnung von $\xi \in [0.01; 0.1]$ bewährt. Das einmalige Trainieren der Trainigsmenge T nennen wir *Trainingszyklus*.
Diese Art des Lernens bezeichnet man übrigens als *überwachtes Lernen* (vgl. [Hei94]).

5.2 Delta-Regel

Die *Delta-Regel* oder auch *Perzeptron-Lernregel* ist nur für einschichtige[3] Perzeptronen definiert. Informell ausgedrückt, berechnet die Delta-Regel eine Veränderung von Gewichtungen anhand des Einflusses der an der Verbindung beteiligten Neuronen auf den Fehler. Die Gewichtungsänderungen ergeben sich wie folgt:

1. Eine bestimmte Eingabe, dessen korrekte Ausgabe bekannt ist, wird durch das Netz propagiert.

2. Für jedes Ausgabe-Neuron u_j, wird zunächst δ_j berechnet mit:

$$\delta_j = (p_j - o_j). \tag{5.15}$$

3. Die Gewichtungsänderung Δw_{ij} für jede Gewichtung ergibt sich nun aus:

$$\Delta w_{ij} = \delta_j o_i \varepsilon, \tag{5.16}$$

dabei ist $\varepsilon \in \mathbb{R}$ die sogenannte *Lernrate*, welche die Intensität des Trainings (Stärke der Gewichtungsänderung) angibt. Die Gewichtungsanpassung selbst geschieht letztendlich durch:

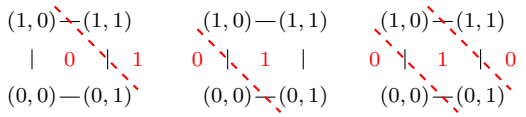
$$w_{ij}^{neu} = w_{ij} + \Delta w_{ij}. \tag{5.17}$$

5.2.1 XOR-Problem

Es wurde von Marvin Minsky und Seymour Papert bewiesen, dass einschichtige Perzeptronen die XOR-Funktion nicht lösen können (vgl. [Hei94]). Der Grund: Einschichtige Perzeptronen können gerade die *linear separierbaren* Probleme lösen bzw. lernen. Die XOR-Funktion hingegen ist nicht linear separierbar, während die AND- und die OR-Funktion linear separierbar sind. Linear separierbar heißt, dass die Ergebnismenge durch die *Hyperebene* (vorzustellen als Linie) in zwei[4] Klassen separiert werden kann. Dies soll Abbildung 6 veranschaulichen. Für AND und OR ist die Separierung leicht, während sie für XOR nicht möglich ist (hier angedeutet durch zwei Linien). Gelöst werden kann das XOR-Problem jedoch durch Hinzufügen von Hidden-Layern. Sie-

[3] Perzeptron ohne Hidden-Layer (nur Input- und Output-Layer).

[4] Am Beispiel zweidimensionaler Funktionen.



(a) AND (b) OR (c) XOR

Abbildung 6: Lineare Separierbarkeit.

he [Hei94],[Kri07] für detailierte Informationen.

5.3 Back-Propagation

Bei der *Back-Propagation* wird der Fehler beginnend beim Output-Layer rückwärts durch das Netz propagiert. Da die Delta-Regel nur für einschichtige Perzeptronen definiert ist, muss diese für mehrschichtige Perzeptronen erweitert werden. Es ergibt sich δj durch:

$$\delta_j = \begin{cases} \varphi'(\sigma_j)(p_j - o_j), & \text{falls } u_j \in U_O \\ \varphi'(\sigma_j) \sum_{k \in C^j} \delta_k w_{jk}, & \text{sonst.} \end{cases} \tag{5.18}$$

Bei der Bestimmung von δ_j wird explizit zwischen Output- und Hidden-Neuronen unterschieden. Während der Fehler bei Output-Neuronen unmittelbar berechnet werden kann, muss dieser bei Hidden-Neuronen rekursiv „zurück berechnet“ werden. Im Detail arbeitet die Back-Propagation wie folgt:

1. Analog zu einschichtigen Perzeptronen wird zuerst eine Eingabe durch das Netz propagiert.

2. Für jedes Nicht-Input-Neuron u_j, wird nun δ_j berechnet.

3. Die Gewichtungsänderung Δw_{ij} für jede Gewichtung wird bestimmt.

4. Die neuen Gewichtungen w_{ij}^{neu} werden berechnet. Dabei sei zu beachten, dass die neuen Gewichtungen erst dann übernommen werden, wenn die Back-Propagation komplett durchgelaufen ist. Anders ausgedrückt: Bei sämtlichen Berechnungen während der Back-Propagation werden stets die alten Gewichtungen herangezogen (vgl. [RR95]). Dies war bei der einfachen Delta-Regel unerheblich, da es unter Verwendung der Delta-Regel nur eine Gewichtungen-tragende Schicht geben kann.

Abbildung 7 skizziert das allgemeine Vorgehen bei der Back-Propagation. Der Fehler wird rückwärts durch das Netz propagiert und die Gewichtungsänderung „unterwegs“ bestimmt.

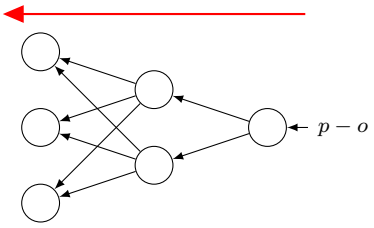


Abbildung 7: Back-Propogation Beispiel.

Anmerkungen:

- Man bezeichnet die Back-Propagation auch als *Gradientenabstiegsverfahren.*

- Die richtige Wahl von ε spielt sowohl bei der Back-Progagation als auch bei der Delta-Regel eine entscheidene Rolle. Es macht daher Sinn, eher mit einem kleinen ε und einer hohen Anzahl an Trainigszyklen zu beginnen und ε langsam (kontrolliert) zu erhöhen. In der Literatur wurde initial ein $\varepsilon = 0.05$ vorgeschlagen (vgl. [Hei94]). Die nötige Anzahl an Trainingszyklen kann schon bei kleineren Problemen mehrere tausend betragen.

- Das Lernen, also das Anpassen der Gewichte bezüglich eines bestimmten Problems, bezeichnet man auch als klassisches *Optimierungsproblem* für das es durchaus auch andere Lösungsstrategien gibt, als ein Gradientenabstiegsverfahren. Ein sehr effektives und eindrucksvolles Verfahren zeigt [Hei94] durch die Verwendung *genetischer Algorithmen*.

6 Ergänzungen

6.1 Bias-Neuronen

Manchmal ist es bei Perzeptronen hilfreich, sogenannte Bias-Neuronen zu aktivieren. Ein Bias-Neuron ist kein Neuron im eigentlichen Sinne, sondern hat immer einen konstanten Wert und kann jedem Nicht-Output-Layer beigefügt werden. Dadurch, dass ein Bias-Neuron immer konstant ist, aber trotzdem variable Gewichtungen hat, fungiert es als dynamischer Schwellenwert und kann die Ausgabe dadurch positiv beeinflussen (vgl. [Hei94]). Bias-Neuroren haben keine Eingänge, können also nicht das Ziel einer Verbindung sein. Abbildung 8 zeigt beispielhaft ein Netz

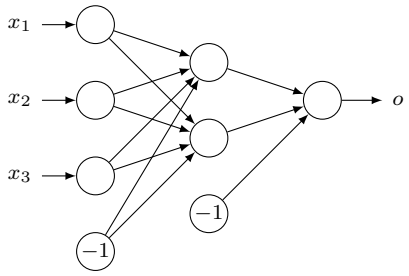


Abbildung 8: Bias-Neuronen Beispiel

mit Bias-Neuronen. Die Neuronen haben dabei den in der Literatur verwendeten Wert von -1.

Literatur

[Hei94] Heistermann, Jochen: *Genetische Algorithmen - Theorie und Praxis evolutionärer Optimierung*. Band 9. Stuttgart/Leibzig : B.G. Teubner Verlagsgesellschaft, 1994

[Kri07] Kriesel, David: *Ein kleiner Überblick über Neuronale Netze*. http://www.dkriesel.com, 2007. – gesichtet 10.11.2009

[MP43] Mcculloch, Warren ; Pitts, Walter: A logical calculus of the ideas immanent in nervous activity. In: *Bulletin of Mathematical Biology* 5 (1943), December, Nr. 4, 115–133. http://dx.doi.

org/10.1007/BF02478259. – DOI 10.1007/BF02478259

[Ros58] Rosenblatt, Frank: The perceptron: a probabilistic model for information storage and organization in the brain. In: *Psychological Review* 65 (1958), November, Nr. 6, S. 386–408

[RR95] Rao, Valluru B. ; Rao, Hayagriva: *C++, neural networks and fuzzy logic*. 2nd ed. New York, NY, USA : MIS:Press, 1995. – ISBN 1–55851–552–6