Erklärung

Hiermit erkläre ich, dass ich diese schriftliche Abschlussarbeit selbstständig verfasst habe, keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe.

Tübingen, den 14.06.2017

(Steffen Schnürer)

Inhaltsverzeichnis

1 Einführung

Bishier noch bisschen leer...

2 Grundlagen

In diesem Kapitel möchte ich einige Grundlagen für die weiteren Kapitel dieser Arbeit legen. Zunächst erläutere ich den Aufbau eines neuronalen Netzes im allgemeinen, die Bedeutung von rekurrenten neuronalen Netzen (RNN), zeige am Bouncing Ball Szenario das Vanishing Gradient Problem auf und mache so die Notwendigkeit der LSTM-Technik deutlich.

2.1 Neuronale Netze

Als neuronales Netz bezeichnet man eine verwobene Struktur zwischen vielen einzelnen Zellen von meistens gleichem - aber keineswegs darauf beschränktem - Aufbau, den sogenannten Neuronen. Eine solche Zelle hat immer die Eigenschaft, dass sie Signale von anderen Zellen empfängt, diese gewichtet aufakkumuliert und abhängig von einer internen Aktivierungsfunktion ein entsprechendes Signal an andere Zellen weitergibt, die damit ihrerseits diesselbe Prozedur durchlaufen. Ein neuronales Netz im Gehirn einer Ameise hat ca. 250.000 Neuronen, ein menschliches 86 Milliarden [4] und wir haben lediglich eine wage Vorstellung, wozu diese imstande sind. In der Informatik werden solche Strukturen als künstliche neuronale Netze nachgebildet und je nach Problemstellung abgewandelt. Zur Vereinfachung meinen wir ab sofort, sofern nicht explizit anders angegeben, mit neuronalen Netzen künstliche neuronale Netze. Es gibt viele verschiedene Arten von neuronalen Netzen, die einfachste von Ihnen ist das Multilayer Perceptron (MLP). Die Zellen werden zu mehreren Ebenen(Layer) zusammengefügt. Erst das Inputlayer, in das die Daten hineingegeben werden, dann die versteckten Layer, versteckt weil hier wie in einer Blackbox die Berechnungen passieren und dann an das Outputlayer weiterleitet, aus welchem das Ergebnis des Netzes kommt.Es gibt eine geordnete Datenflussrichtung, eine Zelle erhält ihren Input von jeder Zelle der vorigenen Ebene und gibt ihren Output entsprechend an jede

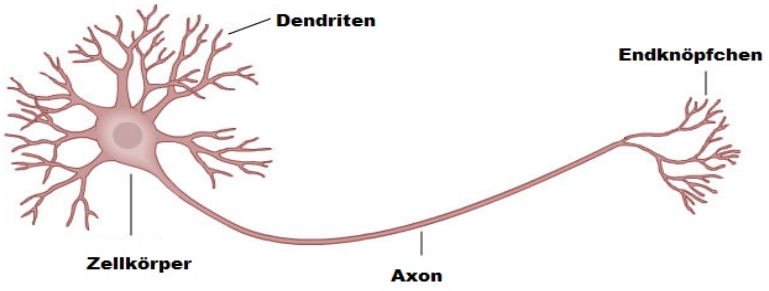

Abbildung 2.1: Schematische Darstellung eines biologischen Neuron [1]

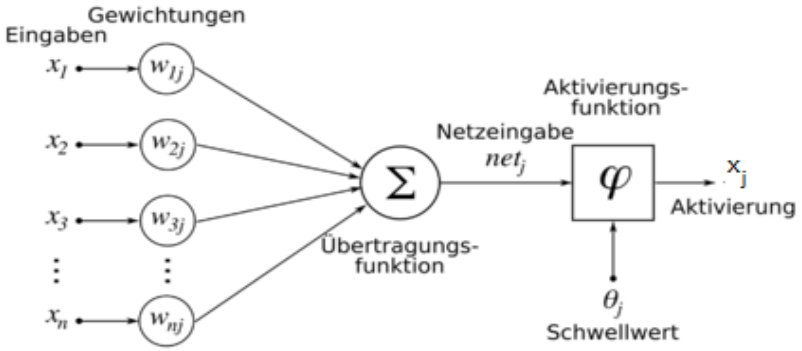

Abbildung 2.2: Schematische Darstellung einer Möglichkeit, ein künstliches Neuron zu simulieren [2]

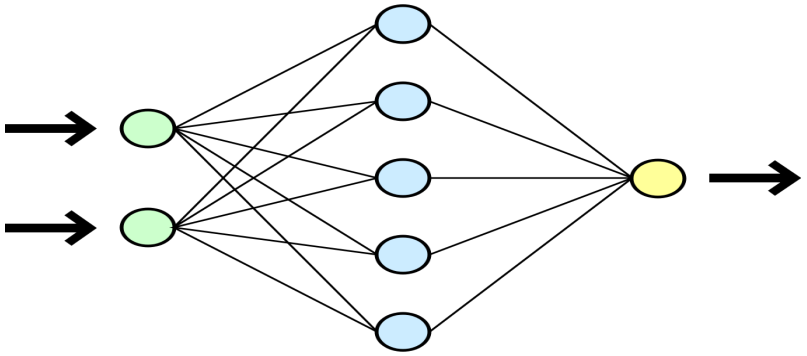

Abbildung 2.3: Ein Multilayer Perceptron mit 2 Inputneuronen(grün), einem Hidden Layer mit 5 Neuronen(blau) und einem einzelnen Outputneuron(gelb). [3]

Zelle der folgenden Ebene weiter, Feedforward genannt. Ein solches Netz sieht man in Abbildung 2.3.

$$x_j = \varphi_h(net_h) = \varphi_h(\sum_{i=1}^{n} x_i w_{ij}) \tag{2.1}$$

In jeder Zelle j werden die Inputsignale x_i, also die Aktivierungen der vorigen Zellen mit Index i unterschiedlich gewichtet w_{ij} aufsummiert net_h (Abbildung 2.2). Eben diese Gewichte sind der Kern des maschinellen Lernens, da bei einmal geschickt gefundenen Gewichten, komplexe Aufgabenstellungen und Probleme mit (vergleichsweise) wenig Rechen- sowie Programmieraufwand gelößt werden können. Im Wesentlichen versucht ein MLP immer eine Funktion zu berechnen welches einen Inputvektor der Größe n auf einen Outputvektor der Größe m abbildet.

$$f_{MLP} : R^n \to R^m$$

Ein Beispiel für eine solche Funktion wäre für ein gegebenes Bild zu entscheiden, ob darauf ein Gesicht zu erkennen ist. Die Länge des Inputs wäre hier z.B. mehrere Millionen, einfach die Pixelwerte, als Output wäre hier nur eine Zahl, nämlich ob ein Gesicht zu sehen ist oder nicht. Der Output y des Netzes ist also abhängig vom Input x, aber auch von den Gewichten w. Die Gewichte werden entweder mit Hilfe von Trainingsdaten, bestehend aus Inputdaten und entsprechenden Outputdaten, also den zugehören Lösungen, oder mit einer zu erlenenden Zielfunktion, die einem aus gegebenen Inputdaten die gewünschten Lösungen liefert, über mehrere Trainingsläufe (Epochen) justiert. Fügt man in die Inputebene entsprechende Inputdaten x ein, berechnet die Aktivierungen aller Zellen, vergleicht die Aktivierungen y des Outputlayers des Netzes mit den gegebenen Lösungen z und erhält eine Fehlerabweichung $E(z, y)$:

$$E(z, y) =_{def} \frac{1}{2} \sum_{i=1}^{m} (z_i - y_i)^2 \tag{2.2}$$

Die Herausforderung ist es nun, diejenigen Gewichte w zu finden, für die die Fehlerabweichung über alle Trainingsdaten minimal ist.

$$arg_w min(\sum_{(x,z) \in trainset} E(z, f_{MLP}(w, x))) \tag{2.3}$$

Bei der Anpassung der Gewichte der versteckten Zellen hat man aber das Problem, das man keinen direkten Fehler für sie kennt, da man nur für die Outputschicht den gewünschten Output hat und so einen Fehler ermitteln kann. Als Lösung ist hier der Backpropagation-Algorithmus geläufig. Dieser besteht aus 3 Schritten, die entweder solang durchlaufen werden bis der Fehler einen festgesetzten Schwellenwert(Threshold) unterschreitet, oder eine vorgegebene Anzahl an Epochen durchlaufen wurde.

Feedforward

Das Inputlayer wird mit der Eingabe des jeweiligen Testsatzes befüllt und die Aktivierungen vorwärts Layer für Layer, Zelle für Zelle berechnet.

Backward-Pass

Das Ergebnis am Outputlayer wird mit der gewünschten Lösung des Testsatzes verglichen und der Fehler berechnet. Diese werden entsprechend gewichtet die unteren Layer rückwärts Zelle für Zelle rückpropagiert. Oft werden hier statt dem tatsächlichen Fehler direkt der Gradient bzw. ein Zwischenwert für dessen Berechnung propagiert??.

Anpassen der Gewichte

Dies ist der wichtige Schritt, hier werden nun Zelle für Zelle die Gewichte entsprechend des Gradienten angepasst. Mit den neuen Gewichten ist in der Regel der Output des Netzes genauer, der Fehler also geringer und der Algorithmus lässt die Fehlerfunktion im Idealfall gegen 0 konvergieren. [5]

Die Gewichte werden anhand eines absteigenden Gradienten der Fehlerfunktion, abgeleitet nach den Gewichten w_{ij}, aktuallisiert. Hierzu werden die δ, die aufsummierten Fehler für jede Zelle wie folgt berechnet:

$$\delta_k = \varphi'_k(net_k)(x_k - z_k) \tag{2.4}$$

$$\delta_h = \varphi'_h(net_h) \sum_{k \in K} w_{hk}\delta_k \mid KNeuronenausdemoberenLayer \tag{2.5}$$

Berechnung des Delta Fehlers für jedes Neuron des Outputlayers(oben) und der versteckten Layer (unten).
Wobei erst für die Outputlayer die δ_k gebildet und in die jeweils unteren Layer beim bilden der δ_h gewichtet aufsummiert (Backpropagation). Die Formel für das Gewichtsupdate mit einer Lernrate η lautet dann:

$$\Delta w_{ij} =_{def} -\eta \frac{\partial E}{\partial w_{ij}} = -\eta x_i \delta_j \tag{2.6}$$

Formel zur Berechnung der Gewichtupdates aus dem Gradienten der Fehlerfunktion [6]
Es gibt nun noch verschiedene Techniken um das Ergebnis weiter zu verbessern und das lernen noch schneller und stabiler zu machen, an dieser Stelle wollen wir unser Augenmerk aber in eine andere Richtung lenken.

2.2 Rekurrente Neuronale Netze

Neuronale Netzwerke haben einen für einen 1-Dimensionalen Inputvektor einen eindeutig zu berechnenden 1-Dimensionalen Outputvektor und für die meisten Problemstellungen ist auch genau diese Kompetenz gefordert. Es gibt aber auch Probleme, bei denen der Output nicht nur von diesem, sondern auch von allen beziehungsweise einigen vorigen Inputs und Zuständen abhängt. Ein anschauliches Beispiel wäre hier die Klassifizierung von Videoausschnitten, wo die Bedeutung eines einzelnen Frames vom Kontext der vorigen Frames abhängt, hingegen ein einfacheres Beispiel wäre das in dieser Arbeit betrachtete Bouncing Ball Szenario. Befindet sich ein Ball im Punkt $p_1 = (0/0)$ und der nächste Ort p_2 soll vorhergesagt werden, ist es natürlich entscheidend ob der Ball sich zuvor beispielsweise im Punkt $p_0 = (0.1/0, 1)$ oder

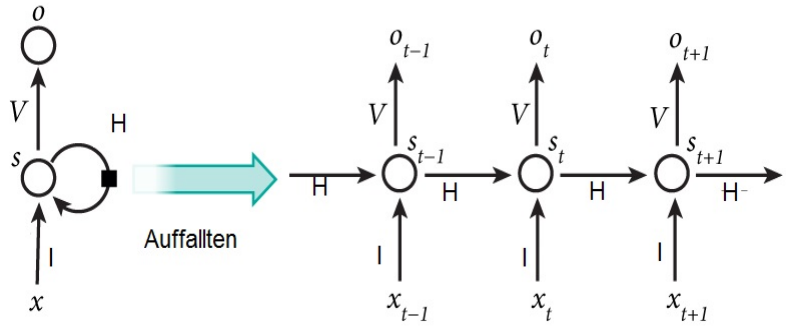

Abbildung 2.4: Um Backpropagation für ein rekurrentes Netz zu benutzen, muss man es durch die Zeit auffallten. [?]

im Punkt $p_0 = (-0.1/-0.1)$ befand. Hierzu werden den Zellen aus dem verstecktem Layer des neuronalen Netzes rekurrente (lat. recurrere:zurücklaufen) Verbindungen hinzugefügt. Eine Zelle h bekommt nun im Zeitpunkt t seinen Input immernoch von allen Zellen des unteren Layers I, zusätzlich nimmt sie aber auch noch als Input den Output aller Zellen desselben Layers, aber aus dem vorigen Zeitschritt H', die Formel für die Aktivierung lautet also:

$$x_h^t = \varphi_h(net_h^t) = \varphi_h(\sum_{i \in I} w_{ih} x_i^t + \sum_{i \in H'} w_{h'h} x_{h'}^{t-1}) \tag{2.7}$$

Da diese neuen rekurrenten Inputs auch wieder gewichtet verarbeitet werden, müssen diese erst noch geschickt gefunden werden. Dies macht man mit Backpropagation durch die Zeit (BPTT). Dies passiert analog zur Backpropagation im MLP, jedoch werden die einzelnen Zeitschritte aufgeklappt (Abbildung 2.4). Die Backpropagation berechnet sich also wiefolgt:

$$\delta_k^t = \varphi_k'(net_k^t)(x_k^t - z_k^t) \tag{2.8}$$

$$\delta_h^t = \varphi_h'(net_h^t)(\sum_{k \in K} w_{hk} \delta_k^t + \sum_{h' \in H'} w_{hh'} \delta_{h'}^{t+1}) \tag{2.9}$$

Berechnung des Delta Fehlers für jedes Neuron des Outputlayers(oben) und der versteckten Layer (unten).
[?] Den Backpropagation Algorithmus wie bei einem Multilayer Perzeptron anwenden um das Netz zu trainieren und passende Gewichte, auch für die rekurrenten Verbindungen zu erhalten. Nun hat man ein Neuronales Netz mit einer Art Kurzzeitgedächtnis, welches beim Auswerten des Inputs die Nahe Vergangenheit mit in Betracht zieht. Hat es in der Anwendung aber tiefere Abhängigkeiten, wo ein vergangener Input einen Einfluss auf die fernere Zukunft hat, hat auch das RNN noch Schwierigkeiten:

2.3 Das Vanishing Gradient Problem

Wir haben gesehen wie beim Training eines rekurrenten Netzes über die Bildung des Gradienten geschickt Gewichte gefunden werden, die Probleme lösen können bei denen Abhängigkeiten über die Zeit auftreten. Erstrecken sich diese Abhängig

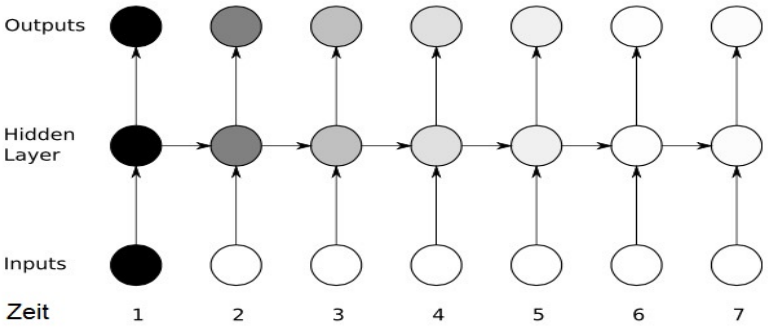

Abbildung 2.5: Das Problem vom verschwindendem Gradienten in einem rekurrenten Netz. Ein Fehler im Zeitschritt 1 erzeugt einen Gradienten, der aber auf die folgenden Zeitschritte immer weniger Einfluss hat. [?]

Literaturverzeichnis

[1] online.science.psu.edu/bisc004_activewd001/node/1907 (abgerufen am 10.05.2017)

[2] de.wikipedia.org/wiki/K%C3%BCnstliches_Neuron (abgerufen am 10.05.2017)

[3] de.wikipedia.org/wiki/Neuronales_Netz#/media/File:Neural_network.svg (abgerufen am 10.05.2017)

[4] en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons (abgerufen am 20.05.2017)

[5] www.neuronalesnetz.de/ (abgerufen am 02.06.2017)

[6] www.informatikseite.de/neuro/node24.php (abgerufen am 03.06.2017)

[7] www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introducti (abgerufen am 04.06.2017)

[8] http://eric-yuan.me/rnn2-lstm/ (abgerufen am 14.05.2017)

[9] aa (abgerufen am 10.05.2017)

Abbildungsverzeichnis