Universität Leipzig

Fakultät für Mathematik und Informatik

Forschungsseminar Deep Learning

Prof. Dr. Erhard Rahm

Wintersemester 2017/18

Rekurrente / rückgekoppelte neuronale Netze

Hausarbeit

Vorgelegt von: Andreas Haselhuhn

Matrikelnummer: 1473621

Studiengang: Master of Science Informatik

Betreuer: Ying-Chi Lin

Ort: Leipzig

Inhalt

1. Einleitung

Diese Arbeit möchte eine Einführung in rekurrente neuronale Netzwerke geben. Im ersten Abschnitt wird eine Einführung in den Aufbau eines Neurons sowie in den Aufbau von „feedforward-Netz“ gegeben. Der folgende Abschnitt möchte eine Einführung in rekurrenten neuronalen Netzwerken (RNN) vermittelt. Es wird auf die strukturellen Unterschiede zu „feedforward-Netzwerken“ eingegangen, sowie auf verschiedene Verbindungstypen, in RNN‘s. Im vorletzten Kapitel werden Beispiele für RNN’s gegeben und ihre Einsatzmöglichkeiten aufgezeigt. Zum Schluss wird eine kurze Zusammenfassung geliefert.

1.1. Neuron

Ein künstliches Neuron ist die kleinste Einheit in einem künstlichen neuronalen Netzwerk. Diese künstlichen Neuronen haben das Verhalten von biologischen Neuronen zum Vorbild. Ein künstliches Neuron besteht im Allgemeinen aus Eingängen, Summenfunktion, Aktivierungsfunktion und Ausgängen. Ein Beispiel für ein künstliches Neuron ist in Abbildung 1 zu sehen. Hier sind die Eingänge mit $x_1 \; bis \; x_n$ gekennzeichnet, sie sind meist Ausgänge anderer künstlicher Neuronen oder Eingaben in das künstliche neuronale Netzwerk. Es sind nummerische Werte die, mit Hilfe von Gewichtungen, in der Summenfunktion $\sum w_i \, x_i + b$ zusammengefasst werden. Aus dem Wert der Summenfunktion wird mit Hilfe der Aktivierungsfunktion$f(t)$, die Aktivierung des Neurons errechnet. Es werden meist Lineare, Binäre oder Sigmoide Aktivierungsfunktionen verwendet. Diese können auch mit einem Schwellwert versehen sein. Der Wert der Aktivierungsfunktion ist dann der Ausgang des Neurons. Dieser Wert wird entweder an die Eingänge andere Neuronen gesendet oder ist der Ausgabewert des künstlichen neuronalen Netzwerks.

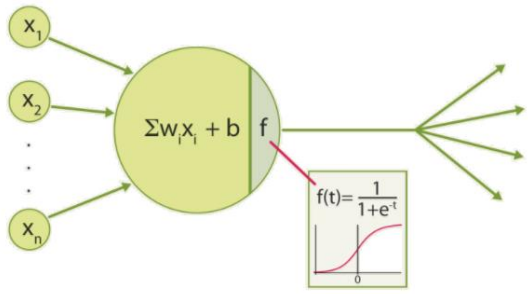

Abb. 1: Künstliches Neuron; w_i sind die Gewichtungen der einzelnen Eingaben x_i (Abgerufen 2018.03.01 von [07])

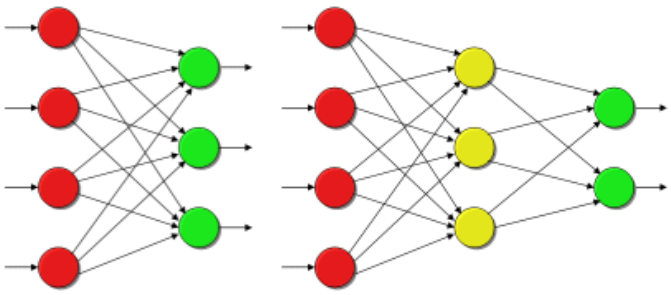

Abb. 2: Feedforward-Netzwerk, Links: nur Eingabe (rot) und Ausgabe (grün) -Layer, Rechts: Eingabe (rot), Hidden (gelb) und Ausgabe (grün)-Layer (abgerufen 2018.03.01 von [7])

1.2. Feedforward-Netzwerke (FFN)

Vorwärts gerichtete oder „feedforward“-Netzwerke (FFN) (Abbildung 2) sind häufig eingesetzte künstliche neuronalen Netze. Sie werden meist zur Mustererkennung oder zur Kategorisierung verwendet. Zum Beispiel werden in der Bilderkennung FFN’s eingesetzt. Tesla setzt, bei der Schilderkennung, auf FFN für ihre Autopiloten [1]. Diese Netzwerke können sich auf Veränderungen sehr schlecht einstellen, da Sie nach dem Training nicht mehr verändert werden. Dies kann zu einer Angreifbarkeit dieser Technik führen [2]. FFN’s können auch kein oder nur sehr schwer Vorhersagen treffen. Dies liegt an ihrer Struktur, der Unveränderlichkeit nach dem training und der Trainingsmethode.

In Abbildung 2 kann man erkennen, dass ihre Struktur einem Baum ähnelt. FFN’s sind gerichtete Graphen, mit Kreis beziehungsweise Zyklen Freiheit. Sie bestehen immer aus einer Eingabeschicht, mit beliebig vielen Knoten und einer Ausgabeschicht mit beliebig vielen Knoten, die aber nach dem Training fest sind. Zwischen diesen beiden Schichten können, müssen aber nicht, beliebig viele versteckte Schichten (Hidden-Layer) liegen. Verbindungen darf es immer nur zwischen einer Schicht n und der Schicht $n + 1$ geben. Es gibt vollständig verbundene FFN’s. In diesen hat jedes Neuron einer Schicht n eine gerichtete Verbindung zu jedem Neuron der Schicht $n + 1$.

Eine Eingabe durchläuft ein neuronales Netz synchron. Das heißt wenn im Zeitschritt T_1 eine Eingabe in das neuronale Netzwerk eingegeben wird, so wird sie im Zeitschritt T_2 in der nächst tieferen Schicht verarbeitet. Wobei ein Neuron nie mehr als einen Zeitschritt benötigt um ein Signal zu verarbeiten. Somit wird ein neuronales Netz der Tiefe n in n-Zeitschritten durchlaufen. Die Tiefe eines neuronalen Netzwerkes ist die Anzahl der Schichten. Das heißt immer eine Eingabe und eine Ausgabe-Schicht, also mindesten zwei und die Anzahl der Hidden-Layer. In der Abbildung 2 ist links ist die Tiefe zwei und recht ist die Tiefe drei.

FFN’s werden mit Backpropagation trainiert.

1.3. Backpropagation, Training von FFN

Das Backpropagation-Verfahren ist ein verbreitetes Verfahren zum Trainieren von künstlichen neuronalen Netzwerken. Es gehört zur Gruppe der überwachten Lernverfahren [3]. Es ist ein Fehler optimierendes Verfahren. Um es an zuwenden wird eine Menge von Eingabe- und Ausgabewerten benötigt.

Das Verfahren läuft in drei Phasen ab. In Phase eins werden die Eingaben vorwärts durch das Netz propagiert. Es werden also alle Berechnungen Ausgeführt und man erhält ein Ergebnis. Im zweiten Schritt wird diese mit dem erwarteten Ergebnis verglichen und ein Fehler errechnet. Dieser wird als Fehler des Netzwerks betrachtet. Im dritten Schritt wird dieser Netzwerkfehler rückwärts durch das Netzwerk propagiert. Wobei bei jeder Neuronenverbindung errechnet wird wie stark sie an diesem Fehler beteiligt war. An Hand dieses, Neuronenfehlers, kann das Gewicht w_i (Abbildung 1) neu berechnet werden. Durch dieses Verfahren wird bei jedem Anlegen einer Eingabe eine Annäherung an das erwartete Ergebnis erreicht.

Dieses Verfahren versucht nicht ein Biologisches Verfahren zu imitieren oder abzubilden. Es ist ein rein mathematisches Verfahren und kein experimentell gefundenes Verfahren. Daher wird es häufig von Neurowissenschaftlern kritisiert [3].

2. Rekurrente neuronale Netze (RNN)

Rekurrente neuronale Netzwerke (recurrent neural network) oder RNN sind so ähnlich Aufgebaut wie FFN's. In einem RNN sind aber Kreise und Zyklen zugelassen. Das heißt es sind auch Verbindungen zwischen einer Schicht und ihrem Vorgänger oder zwischen Neuronen in einer Schicht möglich.

Durch diese Kreisstrukturen entstehen Rückkopplungen aus vorangegangen Ergebnissen. Das kann heißen, wenn mir aus einem früheren Schritt bekannt ist das ich auf einer Autobahn fahre, so ist ein Stoppschild eher unwahrscheinlich. Somit können Rückkopplungen als Gedächtnis aufgefasst werden. Des Weiteren kann man ein RNN auch zur Verarbeitung von Sequenziellen oder Zeitlich nicht synchronen Daten verwenden. Ein FNN behandelt alle Eingaben unabhängig voneinander. Mit Hilfe eines RNN können Abhängigkeiten aufgebaut und Schlussfolgerungen gezogen werden. Diese Schlussfolgerungen oder Vorhersagen, die mit Hilfe eines RNN's getätigt werden können, sind der menschlichen Kognition sehr ähnlich. Daher können diese Netze aber auch falsche Schlüsse ziehen. Sie sind also nicht wie ein math. Algorithmus, der wenn er terminiert ein „richtiges" Ergebnis liefert.

Die Einsatzgebiete für RNN's sind zum Beispiel Übersetzungen von Sprachen, Spracherkennung, Vorhersage von skalar Werten aus einer Wertereihe (zum Beispiel Aktienkurse, Wettervorhersagen, Temperaturwerte), in Attraktornetzen oder zur früh Erkennung von Tumorzellen. FFN's können Tumorzellen sehr gut von normalen Zellen unterscheiden. RNN's können vorhersagen ob eine Zelle zu einer Tumorzelle werden könnte.

2.1. Typen von Verbindungen in RNN

In rekurrenten neuronalen Netzwerken werden im Allgemeinen vier verschiedene Rekurrente Verbindungen unterschieden. Die erste ist die direkte Rückkopplung (direct feedback). Hier ist der Ausgang eines Neurons auch direkt einer seiner Eingänge (Abbildung 3, blau w_d). Der zweite Fall ist in Abbildung 2 als grün w_i dargestellt, sie wird als indirekte Rückkopplung (indirect feedback) bezeichnet. Hier wird der Ausgang mit dem Eingang eines Neurons aus der nächst tiefer liegenden Schicht verbunden. Also wenn das Neuron in der Schicht n liegt, so wird es mit einem Neuron aus der Schicht n-1 Verbunden. Der dritte Fall ist die seitliche Rückkopplung (lateral feedback). Sie verbindest den Ausgang eines Neurons der Schicht n mit dem Eingang eines Neurons der Schicht n. Der letzte Verbindungstyp ist das vollständig verbunden Netzwerk. Es verbindet einen Ausgang jedes Neurons mit jedem anderen Neuron im Netzwerk (siehe Abbildung 4).

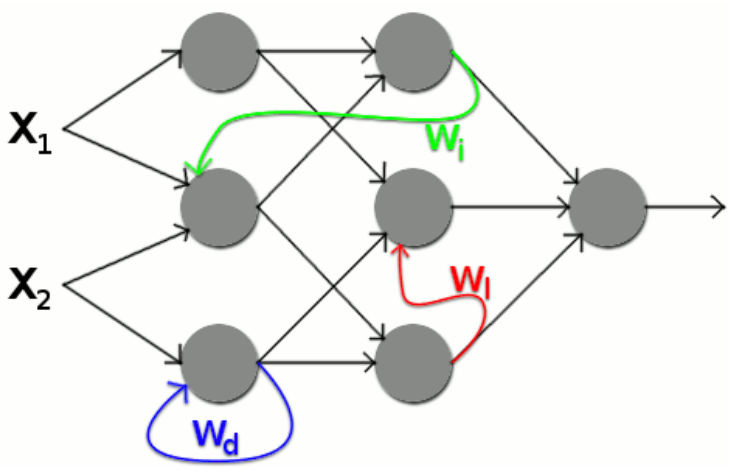

Abb. 3: mögliche Verbindungen in RNN'S (Abgerufen 2018.03.01 von [3]) ; grün : indirect feedback; rot : lateral feedback; blau : direct feedback

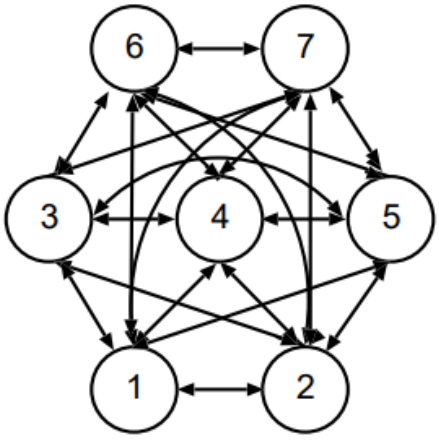

Abb. 4: vollständig verbundenes Netzwerk (Abgerufen 2018.03.01 von [11]

2.2. Backpropagation Through Time (BPTT), Training von RNN

Backpropagation Through Time ist ein an das Backpropagation angelehntes Verfahren. Es erweitert letzteres um einen zeitlichen Faktor. BPTT wird benötigt um rekurrente neuronale Netzwerk zu trainieren. Die Trainings Daten für BPTT sind eine geordnete Sequenz von k Eingabe- / Ausgabe-Paaren. Als nächstes muss das RNN aufgefaltet werden (Abbildung 5). Das aufgefaltete Netz enthält dann k Eingaben und k Ausgaben. Dies geschieht in dem das Netzwerk in der Zeit erweitert wird. Jedes Eingangssignal, Hidden-Neuron und jedes Ausgangssignal wird dupliziert und mit einem Zeitstempel versehen. Somit wird das Netzwerk für jeden benötigten Zeitschritt komplett dupliziert. Hat man ein Netzwerk mit 4 Hidden-Knoten und zwei Zeitschritten so erhält man ein Netzwerk mit 8 Hidden-Knoten. Im Beispiel Abbildung 5 wird ein Netzwerk mit einem Hidden-Knoten (links) beliebig oft erweitert (rechts). Auf das erweiterte Netzwerk wird der Backpropagation-Algorithmus angewandt.

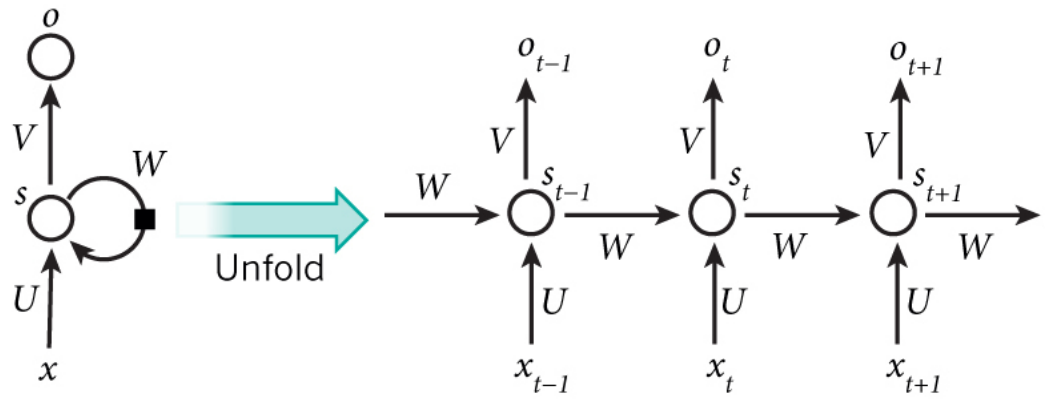

Abb. 5: Backpropagation Through Time: Auffaltung eines RNN über die Zeit. Man kann erkennen, dass sich das Netz in diskreten Zeitschritten dupliziert (Abgerufen am 2018.03.01 von [10])

2.3. „Long short-term memory“ neuronale Netzwerke

„Long short-term memory“ auf Deutsch ein langes Kurzzeitgedächtnis ist eine Angepasstes rekurrentes neuronales Netzwerk. Es zeigte sich das beim Anlernen künstlicher neuronaler Netzwerke (KNN) mit dem Backpropagation Verfahren ein Ungleichgewicht bei tiefen Netzwerken entsteht [4]. Tiefer Schichten werden entweder zu wenig oder zu viel trainiert. Dies geschieht da die Faktoren in der Gewichtungsmatrix eines KNN multipliziert werden. Da Werte zwischen Null und Eins beim Multiplizieren immer kleiner werden, verschwindet der

Fehler. Wenn aber die Fehler Werte größer als Eins sind so wächst die Fehlerzahl an und Knoten die tiefer im Netzwerk liegen werden übertrainiert. Um diese Problem zu lösen wurde ein LSTM-Modul entworfen. In einem KNN werden die einfachen Neuronen durch LSTM-Module ersetzt. Diese sind wieder rum ein RNN. Es gibt verschiedene Arten von LSTM-Modulen. Hier möchte ich das LSTM-Modul, für ein convolutional Neuronal Network vorstellen. Dieses, hier ausgewählte, LSTM-Modul besitzt einen Eingang (in Abbildung 6 x_t bezeichnet) und einen Ausgang (Abbildung 6 h_t). Somit wirkt es von außen wie ein normales Neuron und ist auch diskret. Ein LSTM-Modul erlaubt es, in einem RNN, einen relativ konstanten und anwendbaren Fehlerfluss zu gewähren. In einem LSTM werden verschiedene Funktionen vereint, nicht nur eine Summenfunktion, wie im Neuron. Diese werden als Gates bezeichnet. Die Gates können Zellzustände entfernen oder hinzufügen und bieten die Möglichkeit Informationen optional passieren zu lassen. In dem Vorgestellten Model gibt es 3 zusätzliche Gates und eine innere Zelle. Es gibt das Input-Gate, dieses bestimmt das Maß mit dem neue Werte in die Zelle fließen können. Das Forget-Gate, dieses bestimmt wie Informationen in dem Modul verbleiben oder vergessen werde. Das letzte Gate ist das Output-Gate. Es legt das Maß fest, in wie weit ein Wert, der in der Zelle, errechnet wurde ausgegeben wird. Der Datenfluss zwischen der Eingabe, den Gates, der Zelle und der Ausgabe wird durch Vektor- und Matrizenoperationen bestimmt.

Eine mathematische und tiefer führende Erläuterung, würde diese Arbeit übersteigen. Sie soll nur einen Einstieg in das Thema künstliche rekurrente neuronale Netzwerk liefern. Tiefer und weiterführende Informationen zu „Long short-term memory“ Netzwerken und dem verschieden Aufbau ihrer Module, können aber in den Arbeiten [4], [5], [6] [7] gefunden werden.

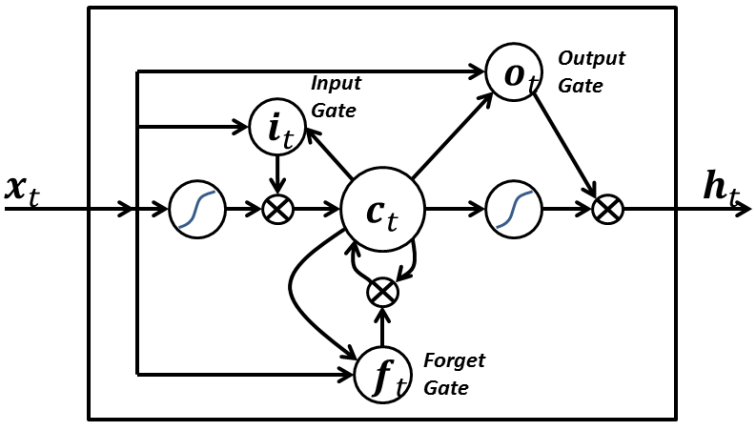

Abb. 6: LSTM-Modul : Die innere Zelle ist mit c_t dargestellt. Kreise mit X-Symbolen sind elementweise Multiplikationen und mit einer Kurve, dargestellte Kreise, sind Sigmoidfunktionen. Die Pfeile, die von der Zelle zu den Gates verlaufen, sind Gucklochinformationen vom letzten Durchlauf. (Abgerufen am 2018.03.01 von [4])

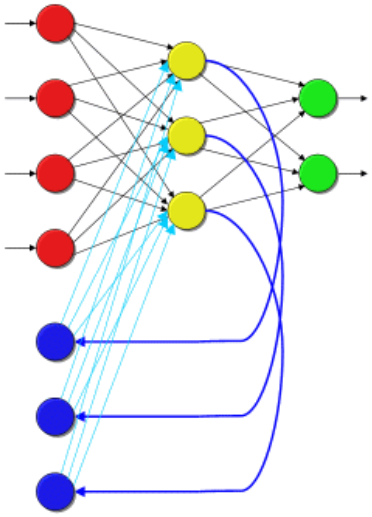

Abb. 7: Beispiel eines Simple recurrent networks (Abgerufen am 2018.03.01 von [8])

3. Beispiel für rekurrente neuronale Netzwerke und ihre Anwendungen

In diesem Kapitel möchte ich verschiedene rekurrente neuronale Netzwerke und ihre Anwendungen Vorstellen.

3.1. „Simple recurrent networks“ [8]

In einem „simple recurrent network“ gibt es für jedes Neuron einer Hidden-Schicht eine Kontexteinheit (Kontext-Unit, KU), in Abbildung 7 blau. Die KU befinden sich in der Schicht $n-1$. Eine KU erhält als einzige Eingabe den Ausgang seines Zugeordneten Neurons und ist selber ein Neuron. Der Ausgang, dieser KU, ist mit jedem Neuron in der Hidden-Schicht n verbunden. Somit erhält jedes Neuron in der Schicht n zum Zeitpunkt t das Ergebnis aller Neuronen aus der Schicht n vom Zeitpunkt $t-1$. Dieses Gedächtnis wird in Abbildung 8 näher erläutert. Ein Simple recurrent network kann willkürliche Sequenzen lernen. Es wird mit Backpropagation Through Time trainiert. Wobei die Gewichtungen der KU nicht fix sind sondern auch angepasst werden.

Diese Netze können bei sich wiederholenden Tätigkeiten eingesetzt werden, bei denen eine Vorhersage über den Zeitpunkt des Erscheinens eines Ereignisses möglich ist. Zum Beispiel das bestellen von Tee oder Kaffee. Da diese nach absehbarer Zeit verbraucht werden. Da eine Bestellung und Lieferung Zeit beansprucht muss die Bestellung also mit Vorlauf

ausgelöst werden. Durch das „Gedächtnis“ eines Simple RNN’s kann dies gelöst werden und anhand der Trainingsdaten eine Vorhersage getroffen werden wann eine Bestellung ausgelöst werden muss.

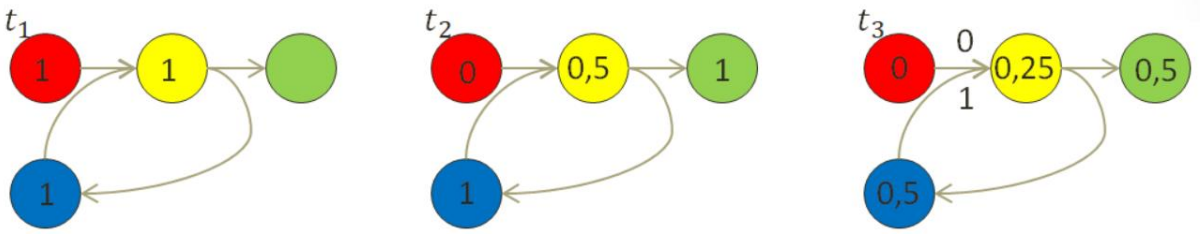

Abb. 8: Werteverlauf einer Kontext-Unit mit drei Zeitschritten: Im ersten Zeitschritt ist die Kontext-Unit mit 1 vorbelegt und es liegt am Eingang eine 1 an. Als Summenfunktion des Hidden-Neurons soll hier der Durchschnitt gebildet werden und für die Ausgabe soll $f(x) = x$ gewählt werden. Somit ergibt sich für das Hidden-Neuron die 1 und für die Kontext-Unit im Zeitschritt 2 auch die 1. Wenn im Zeitschritt 2 eine 0 am Eingang anliegt so ergibt sich für die Summer 0,5 genauso wie für die KU. Im Zeitschritt 3 liegt wieder eine 0 am Eingang. Daher ergibt sich der Wert 0,25. Somit hat die erste Eingabe, die eine 1 war, immer noch eine Auswirkung auf den Ausgabewert. Dieser nimmt aber mit der Zeit. Als Anwendung könnte eine Bestellung ausgelöst werden wenn der Ausgabewert unter einen bestimmten Schwellwert fällt, also eine bestimmte Zeit vergangen ist und somit neuer Kaffee bestellt werden muss.

3.2. RNN mit variabler Eingabe

Alle bisher vorgestellten Ansätze haben eine feste Anzahl von Eingabeknoten. Diese werden vor dem Training festgelegt und können nach dem Training nicht mehr verändert werden ohne das komplette RNN neu zu trainieren. Will man einen Text übersetzten so hat dieser meist eine variable Länge. Abbildung 9 zeigt, dass man zur Vorhersage eines Wortes für zwei Sätze zwei unterschiedliche FFN benötigen würde und diese auch trainieren müsste. Will man also für beliebige Sätze oder Wortgruppen, dass nächste Wort vorhersagen. So sind FFN oder Simple RNN eine eher schlechte Wahl.

Die beiden folgenden Ansätze liefern hier für eine praktikable Lösung. Sie binden eine Zeitabhängigkeit, in die Eingabe eines künstlichen neuronalen Netzwerks ein. Das heißt das Eingaben Zeitlich auf einander abgestimmt geschehen müssen um eine vailde Ausgabe zu erhalten. Diese Netzwerke lassen sich nur mit Backpropagation through time trainieren. In Abschnitt 3.2.1. soll eine Möglichkeit Aufgezeigt werden um aus belibigen Folgen einen skalaren Wert zu erhalten und der Abschnitt 3.2.2. zeigt eine Möglichkeit auf aus Sequnzen neue Sequenzen zu erzeugen. Dies wird bei Übersetzungen von Texten benötigt, da hier für jedes Wort meist ein übersetztes Wort gefunden werden muss.

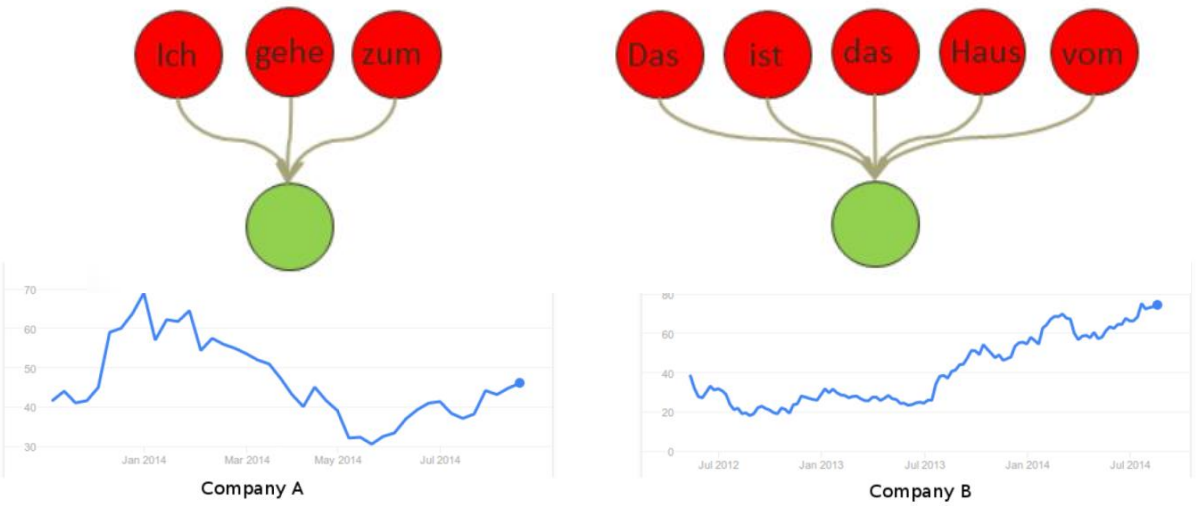

Abb. 9: Neuronale Netze zur Verarbeitung mit verschiedenen Eingaben. Anhand des Beispiels oben, erkennt man das Problem der Wortvorhersage mit FFN's. Für jede Satzlänge oder Wortgruppe würde ein eigenes neuronales Netzwerk erstellt und trainiert werden. Ein ähnliches Problem ergibt sich beider Vorhersage von Aktienkursen (untere Grafik; Grafik aus [5] Seite 9 entnommen). Da nicht alles Aktien immer am gleichen Tang an die Börse gegangen sind. Somit müsste für jede Aktie auch ein neues neuronales Netzwerk trainiert werde.

3.2.1. RNN zur skalar Wertevorhersage

Die RNN-Struktur die ich hier vorstellen möchte, wurde in [5] vorgestellt (siehe Abbildung 11). Sie kann beliebig viele Eingabewerte zu einem, als Vorhersage nutzbaren, skalaren Ausgabewert berechnen. Sie kann also für Probleme wie in Abbildung 9 gezeigt eingesetzt werden. Die Werte $x_0, x_1, \dots, x_T$ bezeichnen skalare Eingabewerte. Zum Beispiel könnten dies ein Temperatur- oder Aktienkurs-Verlauf sein (z.B. x_0 die Temperatur vor T-Tagen bis x_T heute). Die Werte $h_0, h_1, \dots, h_T$ stellen die Hidden-States des RNN dar. Jeder der Kreise in Abbildung 9 stellt ein Layer des Netzwerkes dar.

Es gibt 3 Set von Parametern: die Input-Hidden-Gewichte (W), die Hidden-Hidden-Gewichte (U) und die Hidden-Label-Gewichte (V). Alle W's, U's und V's werden geteilt. Dadurch wächst die Anzahl der Parameter nicht wenn T wächst. Es bleiben immer nur die Parameter W,U und V. Mit dieser Notation werden die Hidden-States rekursiv berechnet, siehe Abbildung 10.

Um die Gradienten des RNN zu berechnen wird dann Backpropagation through time eingesetzt.

$$f(x) = Vh_T$$
$$h_t = \sigma(Uh_{t-1} + Wx_t), \text{ for } t = T, .., 1$$
$$\dots$$

Abbildung 10 : Berechnung der Hidden-States (Abgerufen am 2018.03.01 von [5] entnommen)

Da es aber im Netzwerk gebundene Gewichte gibt, kann man die gleiche Idee wie im konvolutionellen neuronalen Netzwerken anwenden. Im Vorwärtsdurchlauf wird vorgetäuscht das die Gewichte nicht geteilt werden. Es gibt also Gewichte $W_0, W_1, \dots, W_T$ und $U_0, U_1, \dots, U_T$. Bei der Backpropagation wird dann ein Gradient im Bezug auf die W's und U's gebildet. Der Endgradient von W ist dabei die Summe aller Gradienten der W's und der Endgradient von U ist dabei die Summe der Gradienten der U's.

Diese Gradienten können wie schon in 2.3. beschrieben stark ansteigen oder gegen Null tendieren. Dies macht das Lernen sehr langsam oder kann zum Overfitting führen. Daher wird in [5] vorgeschlagen ein Gradienten-Clipping ein zu führen. Diese soll zum Beispiel Werte über eins immer auf eins setzten. Somit wird ein Ansteigen der Gradienten Verhindert. Genauso könnte man untere Schwellwerte einfügen. Die genauen Werte müssten aber anhand der Daten und für jedes Problem jeweils neu definiert werden.

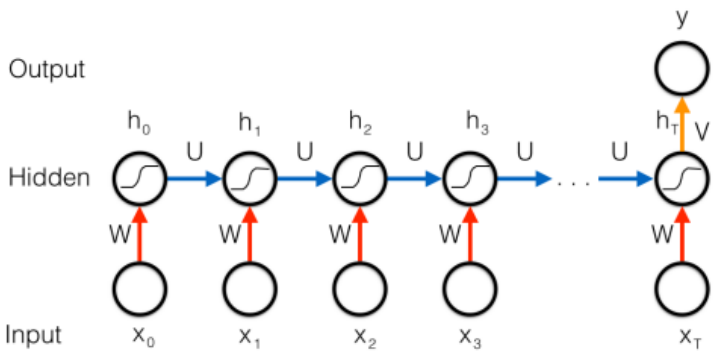

Abb. 11: RNN für Skalar Werte (Abgerufen am 2018.03.01 von [5] entnommen)

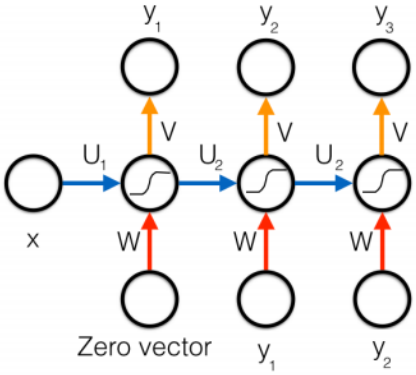

Abb. 12: RNN für Sequenzielle Werte (Abgerufen am 2018.03.10 von [5] entnommen)

3.2.2. RNN für Sequenzen vorhersagen

Die bis jetzt vorgestellten Verfahren hatten immer eine feste Ausgabelänge. Für viele Anwendungen ist aber eine variable Ausgaben-Sequenzlänge sinnvoller. Das hier gezeigte Verfahren wurde in [9] und [5] vorgestellt.

Es bedient sich eines Netzwerks der Struktur in Abbildung 12. Hier wird angenommen das es einen Eingabevektor X gib. Sowie drei Ausgabe-Sequenzen y_1, y_2, y_3. Aus 3.2.1. wissen wir das wir für die Vorhersage von y_1,aus X, ein rekurrentes Netzwerk verwendet werden kann, da dies genau der aus 3.2.1. bekannten Struktur besteht. Der „Zero vector“ wurde, in den ersten Schritt, eingefügt um das Netzwerk simple und konsistent mit den folgenden Schritten zu halt. Für die Berechnung von $y_2$wird nun $y_1$als Eingabe verwendet. Da dieses Netzwerk wieder mit einem Netz aus 3.2.1. dargestellt werden kann. Kann wieder wie in vorherigen Schritt vorgegangen werden. Dies lässt sich beliebig vortführen und ermöglicht somit eine variable Ausgabe von beliebig vielen, oder beliebig benötigten, Werten.

4. Zusammenfassung

Diese Arbeit hat einen Einstieg in rekurrente künstliche neuronale Netzwerke gegeben. Als erster Einstieg wurde allgemein ein kurzer Einblick in die Grundstruktur von neuronalen Netzwerken gegeben. Es zeigte sich das so genannte „Feedforward-Netzwerke“ (FFN) gut zur Mustererkennung geeignet sind. Sie werden mit Backpropagation trainiert. Dieses Verfahren ist ein Fehlerminimierungsverfahren und benötigt eine Eingabemenge und die dazu passenden Ausgaben. Diese werden verglichen und aus dem Ergebnis wird ein Fehlerberechnet der dem Netzwerk mitgeteilt wird. Dieser wird zur Anpassung von Gewichtungen eingesetzt.

Im zweiten Teil wurden rekurrente neuronale Netzwerke (RNN) eingeführt. Da diese einem FFN sehr ähneln, wurden die Unterschiede zu diesen aufgezeigt. Ein Trainingsverfahren für ein RNN wurde mit der Backpropagation Through Time (BPTT) vorgestellt und die Unterschiede zum normalen Backpropagation aufgezeigt. Dann wurde eine spezielle Form eines RNN vorgestellt die „Long short-term memory“ RNN. Diese zeichnen sich durch einen komplexeren Aufbau der einzelnen Knoten bzw. Neuronen aus. Dieser Aufbau lässt aber eine gleichmäßiges Training des Netzwerkes zu.

Im dritten Teil wurden drei Beispiele und ihre Anwendungen für ein RNN aufgezeigt. Es wurden ihre Eigenschaften erörtert. Des Weiteren wurde ihre grundlege Funktion dargelegt. Es konnte gezeigt werden das RNN zur Vorhersage von Zeitpunkten (Bestellungen) oder einzelnen Werten (morgiger Aktienkurs / morgige Temperatur) oder zum übersetzten von Texten eingesetzt werden können.

5. Literaturverzeichnis

[1] Tesla, „Hardware für autonomes Fahren," 2018. [Online]. Available: https://www.tesla.com/de_DE/autopilot. [Zugriff am 01 03 2018].

[2] B. Weddeling, „Wenn Hacker Auto-Algorithmen knacken," http://www.handelsblatt.com, 08 08 2017. [Online]. Available: http://www.handelsblatt.com/unternehmen/it-medien/valley-voice/valley-voice-wenn-hacker-auto-algorithmen-knacken/20159330.html. [Zugriff am 01 03 2018].

[3] „Rekurrentes neuronales Netz," [Online]. Available: https://de.wikipedia.org/wiki/Rekurrentes_neuronales_Netz. [Zugriff am 01 03 2018].

[4] „Long short-term memory," wikipedia, [Online]. Available: https://de.wikipedia.org/wiki/Long_short-term_memory. [Zugriff am 01 03 2018].

[5] Q. V. Le, „A Tutorial on Deep Learning Part 2: Autoencoders, Convolutional Neural Networks and Recurrent Neural Networks," *Google Brain, Google Inc. 1600 Amphitheatre Pkwy, Mountain View, CA 94043,* 20 10 2015.

[6] D. Kriesel, „Ein kleiner Überblick über Neuronale Netze," 01 03 2007. [Online]. Available: http://www.dkriesel.com/science/neural_networks. [Zugriff am 01 03 2018].

[7] Thorten, „Rekurrente Neuronale Netze," 27 12 2016. [Online]. Available: http://www.riemerundco.de/bl/?p=191. [Zugriff am 01 03 2018].

[8] K. F. W. Günter Daniel Rey, „Neuronale Netze- Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung," [Online]. Available: http://www.neuronalesnetz.de/. [Zugriff am 01 03 2018].

[9] O. V. a. Q. V. L. I. Sutskever, „Sequence to sequence learning with neural networks. InAdvances in neural information processing systems," in *pages 3104–3112*, 2014.

[10] B. Britz, „WILDML - Artificial Intelligence, Deep Learning, and NLP," 17 09 2015. [Online].

Available: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/. [Zugriff am 01 03 2018].

[11] P. D. P. Ueberholz, „Neuronale Netze,“ 2014/2015. [Online]. Available: https://lionel.kr.hs-niederrhein.de/~ueberholz/Lehre/NumInf2_WS1415/numinf_11_nn.pdf. [Zugriff am 01 03 2018].

[12] A. Trinkwald, „Netzgespinste,“ 06 2016. [Online]. Available: https://www.heise.de/ct/ausgabe/2016-6-Die-Mathematik-neuronaler-Netze-einfache-Mechanismen-komplexe-Konstruktion-3120565.html. [Zugriff am 01 03 2018].