Ausarbeitung - Robust Named Entity Recognition in Idiosyncratic Domains

Christian Staudte

28. März 2018

Zusammenfassung: In folgender Ausarbeitung soll sich mit dem von Sebastian Arnold et al. entwickelten DATEXIS-NER befasst werden. Dieses System ermöglicht es dem Nutzer, automatisch Eigennamen aus Texten verschiedenster Domänen zu filtern. Verwendung soll das Verfahren insbesondere in der Informationsextraktion finden. Dabei dient DATEXIS-NER als Grundlage anderer Systeme, die gefundene Eigennamen verwenden, um ganze Wissensbasen aufzubauen.

Matrikelnummer: 3756415

Hiermit erkläre ich, die vorliegende wissenschaftliche Arbeit selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialen oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, den _____ _____

Ort, Datum Unterschrift

Inhaltsverzeichnis

1 Einleitung

Wer in seinen Projekten die Extraktion von Informationen anhand natürlichsprachiger Texte anstrebt, wird durch die Komplexität von Sprache, sowie deren gewaltige Wortmenge schnell an Grenzen stoßen. Aus diesem Grund haben sich im Laufe der Zeit zwei Probleme herauskristalisiert, die mit verschiedenen Verfahren gelöst werden sollen: Zum einen das Segmentierungsproblem, also das Aufspüren von relevanten Wörtern bzw. Wortgruppen in Fließtexten. Zum anderen müssen für gefundene Wörter passende Ontologien gewählt werden, um diesen konkrete Bedeutungen zu geben. [30]

In dieser Ausarbeitung soll lediglich das Segmentierungsproblem im Detail betrachtet werden. Das grundlegende Verfahren dazu lautet *Named Entity Recognition* (NER). Unter NER versteht man die automatische Extraktion von Eigennamen aus beliebigen Texten. Eigennamen wiederum sind "Nomen, die ein einzigartiges Objekt referenzieren, wie *London*, *Jupiter*, *Sahara* oder *Microsoft*" [35], dabei können Eigennamen auch zusammengesetzt sein, wie "*Golden Gate Bridge*". Die Relevanz von NER zeigt sich in vielen Anwendungen, wo es als Grundlage fungiert, denn werden Eigennamen ignoriert oder fehlerhaft gefiltert, können sie in weiteren Verarbeitungsschritten nicht genutzt werden. Informationen gehen unwiderruflich verloren. Da NER nur das Segmentierungsproblem löst, wird ein weiteres Verfahren namens *Entity Linking* (NEL) genutzt, bei dem nach Shen et al. [36] ein gefundener Eigenname mit einer Wissensbasis bzw. Ontologie verbunden wird. Eine Wissensbasis besitzt Informationen über *Named Entities*, deren semantische Klassen und alle Beziehungen untereinander. Bekannte Wissensbasen sind DBpedia und YAGO, die ihre Informationen automatisch von Wikipedia extrahieren. Konkrete Anwendungsdomänen für NER i.V.m. NEL wären Frage- und Antwortsysteme oder *Intent Based Searches* [19].

Bei Frage- und Antwortsystemen bekommt das System vom User eine Frage, z.B. "Was ist die Hauptstadt von Frankreich?" Als nächstes sucht eine NEL-Komponente *Frankreich* in der Wissensbasis, das in einer direkten Beziehung zur Hauptstadt *Paris* steht. Die Antwort auf die Frage kann sofort zurückgegeben werden. Eine weitere Verwendungsmöglichkeit von NER ist die *Intent Based Search*, welche von Suchmaschinen wie Google verwendet wird. Ziel von dieser ist es, für jede Person auf gegebene Suchanfragen passende Resultate zu liefern. Resultate können abhängig von Anfrageformulierungen stark individualisiert sein. Damit ist es nicht nur möglich, dem Nutzer ein effizienteres Suchen zu ermöglichen, es kann auch nutzerspezifische Werbung erzeugen werden. Dazu werden Browserverläufe von Nutzern analysiert und notwendige Informationen gefiltert, um Werbungen anzupassen.

Es existieren bereits mehrere Standard Annotatoren, die das NER-Problem zu lösen versuchen. Unter Annotator versteht man hier ein Programm, das *Named Entities* in einem Text automatisch markiert. Beispiele sind *Stanford NER* oder *Entityclassifer.eu*. Jedoch beherbergen aktuelle Annotatoren drei Kernprobleme: Erstens besitzen Textdomänen meist viele, fremdartige Wörter *(Idiosyncratic Language)*. Das heißt, Annotatoren können diese Domänen nur eingeschränkt verarbeiten, Eigennamen gehen verloren oder werden falsch zugeordnet. Bereits fehlerhafte Groß- und Kleinschreibungen wie es bei Überschriften der Fall ist, oder verschiedene Kontexte für das selbe Wort sind typische Ursachen. Zweitens sind Annotatoren für bestimmte Domänen spezialisiert, so dass z.B. mit Nachrichten/News anders als mit medizinischen Daten umgegangen wird. Eine Änderung der Anwendungsdomäne hätte fatale Folgen für die Ergebnisse der Segmentierung. Drittens erfordert ein effizientes Training der Annotatoren einen zu großen, gelabelten Textkorpus. Dieser steht nicht immer zur Verfügung, kann falsch gelabelt sein oder einen ungeeigneten Labelstandard verwenden. Zudem sind Kosten für neue, gelabelte Daten sehr hoch.

Für diese Probleme wollen Arnold et al., die Autoren des Papers *Robust Named Entity Recognition In Idiosyncratic Domains* [3], mit einem eigens entwickelten Annotator namens *DATEXIS-NER* Abhilfe schaffen.

Im Folgenden sollen aktuelle Verfahren für das NER-Problem - mit DATEX-NER als Schwerpunkt - behandelt werden. Als Hauptziel gilt zu ergründen, inwieweit DATEXIS-NER Vorteile gegenüber anderen Annotatoren bietet und ob dieses System sinnvolle Lösungen für alle genannten Probleme besitzt. Im Abschnitt 2 wird zunächst ein Überblick über verwandte Arbeiten anhand derer Herangehensweisen gegeben. Dazu gehören stochastische Modelle, wörterbuchbasierte Modelle und neuronale Netzwerke. Im Abschnitt 3 wird der DATEXIS-NER Annotator analysiert, indem zuerst sein Wortkodierungsverfahren mittels *Letter-Trigram Hashing* und anschließend sein Lernalgorithmus - *Bidirectional Long Short-Term Memory* (BLSTM) - herausgearbeitet und erläutert werden. Der Lernansatz wird wegen seiner Komplexität in einfachere Komponenten differenziert. Dazu gehören *Long Short-Term Memories* (LSTMs), die aus mehreren *Recurrent Neural Networks* (RNNs) bestehen, welche wiederum auf *Feed Fordward Networks* (FF-Netzwerke) basieren. In Abschnitt 4 werden vorerst die Kenngrößen zur Evaluierung: *Precision*, *Recall* und *F1-Score* betrachtet, um die Bedeutung der Ergebnisse aller Tests nachvollziehen zu können. Für diese Tests wird DATEX-NER anhand anderer NER-Systeme sowie eigener Konfigurationsmöglichkeiten verglichen. Somit lassen sich zum Schluss Vor- und Nachteile zusammenfassen.

2 Related Works

Mit *Named Entity Recognition* zur Informationsextraktion hat man sich vertieft seit 1995 bei den *Message Understanding Conferences* [28] MUC-6 und 7 auseinandergesetzt. Dabei handelt es sich um Forschungswettbewerbe, die neue Methoden zur Informationsextraktion fördern sollen. Als Eigennamen bzw. Phrasen gelten neben PERSONEN, ORGANISATIONEN und ORTEN auch spezifische ZEITLICHE und NUMERISCHE Ausdrücke (Maße).

Das Problem dieser Konferenzen war jedoch, dass ausschließlich Englisch als Korpussprache zum Einsatz kam [15]. Man hat sich auf sprachspezifische Faktoren verlassen ohne auf Eigenschaften anderer Sprachen zu achten. Nach 1995 hat man die Systeme auf weitere Sprachen wie Chinesisch, Japanisch, Spanisch oder Französisch angewandt und festgestellt, dass NER auf diesen verschieden operiert, aber ein großer Aufgabenbereich mittels einfacher Methoden abgedeckt werden kann. Daher hat die *Conference on Computational Natural Language Learning* (CoNLL) in den Jahren 2002 und 2003 die Aufgabe ins Leben gerufen, sprachunabhängiges NER so effizient wie möglich zu gestalten. Folglich haben mehr als ein dutzend Forschergruppen Arbeiten eingereicht, in denen sie über verschiedene Ansätze des maschinellen Lernens das Problem zu bewältigen versuchen.

2.1 Stochastische Modelle

2.1.1 Hidden Markov Modelle

Eine Herangehensweise an NER sind stochastische Modelle. Dazu gehört unter anderem das *Hidden Markov Model* (HMM), welches bereits 1999 von Bikel et al. [7] verwendet wurde.

Als Beispiel sei angenommen, man hat einen Satz $W_a = [w_1\ w_2\ w_3\ ...\ w_m]$ der Länge m gegeben. Dann sind die Wörter w_i mit $i \in [1, ..., m]$ beobachtbar und werden Emissionen genannt [20]. Zu dieser Wortfolge sei nun eine Zustandsfolge $S_a = [s_1\ s_2\ s_3\ ...\ s_m]$ der gleichen Länge gegeben, wobei s_i mit $i \in [1, ..., m]$ versteckte Zustände sind. Im Modell von Bikel et al. [7] bezeichnen diese Zustände entweder das Label NOT-A-NAME oder Klassenbezeichnungen wie PERSON oder ORGANISATION. Ziel ist es herauszufinden, wie wahrscheinlich eine Zustandsfolge S_a und Emissionskette W_a zusammengehören.

Zur Bestimmung dieser Wahrscheinlichkeit nutzt man die zwei Markow-Eigenschaften. Die 1. Eigenschaft besagt, dass ein versteckter Zustand s_i lediglich von seinem Vorgänger s_{i-1} abhängt: $P(s_i|s_{i-1})$. Die zweite Eigenschaft dagegen besagt, dass jede Emission w_i nur vom versteckten Zustand s_i abhängt: $P(w_i|s_i)$. Um diese Eigenschaften zu realisieren, greift man auf das *Naive Baïes* Modell zurück, nach welchem Emissionen vollständig unabhängig voneinander sind. Dieses Modell, kombiniert mit der 2. Markow-Eigenschaft, ergibt die Formel:

$$P(w_1, ..., w_m; s_1, ..., s_m) = \prod_{i=1}^{m} [P(s_i) \cdot P(w_i|s_i)] \tag{1}$$

Durch den eingeschränkten Kontext wird die Repräsentation der realen Welt zwar verfälscht, jedoch ist es weitaus recheneffizienter, als wenn alle Emissionen untereinander Abhängigkeiten besäßen [22].

Da HMMs mit sequentiellen Daten arbeitet, müssen nun noch Abhängigkeiten zwischen benachbarten Zuständen eingebunden werden. Dies wird erreicht, indem man die 1. Markow-Eigenschaft in die Gleichung einfließen lässt:

$$P(w_1, ..., w_m; s_1, ..., s_m) = P(s_1) \cdot P(v_1|s_1) \cdot \prod_{i=2}^{m} [P(s_i|s_{i-1}) \cdot P(v_i|s_i)] \tag{2}$$

Mit dieser Gleichung kann man berechnen, wie wahrscheinlich eine Emissionskette mit einer Zustandskette zusammenpasst. Indem man nun alle Kombinationen von Zustandsketten für eine Emissionskette betrachtet, lässt sich die wahrscheinlichste Zustandskette ermitteln. Zur Ausführung dieser Berechnung benötigt es jedoch fünf bekannter Mengen:

- $W = \{w_1, ..., w_M\}$ - Mögliche Emissionen
- $S = \{s_1, ..., s_N\}$ - Mögliche, versteckte Zustände
- $A \in \mathbb{R}^{NxN}$ - Übergangsmatrix für versteckte Zustände, s. $P(s_i|s_{i-1})$
- $B \in \mathbb{R}^{NxM}$ - Beobachtungsmatrix, s. $P(w_i|s_i)$
- $\pi \in \mathbb{R}^N$ - Wahrscheinlichkeit für Startzustände, s. $P(s_1)$.

Zur Bestimmung dieser Parameter wird das HMM anhand eines Trainingskorpus ausreichend trainiert, worauf hier jedoch nicht näher eingegangen werden soll. Es sei angenommen, dass alle relevanten Mengen ermittelt wurden, dann spricht man von einem Markow-Prozess bzw. einer Markow-Kette [25] und das HMM lässt sich in Form eines endlichen Automaten, s. Abb. 1, darstellen.

Aus mathematischer Sicht sind HMMs sehr simpel. - Diese Einfachheit bereitet aber Probleme: Zustände hängen lediglich vom vorherigen Zustand ab und Emissionen nur von einem Zustand. Dabei existieren auch Abhängigkeiten zwischen Emissionen sowie weitere

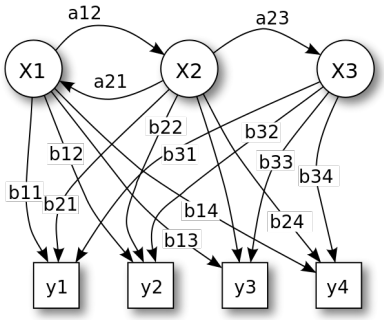

Abbildung 1: Beispielautomat für ein HMM. y_i = Emission; x_i = versteckter Zustand; a_{ij} = Übergangswahrscheinlichkeit, z.B. $a_{12} = P(x_2|x_1)$; b_{ij} = Emissionswahrscheinlichkeit, z.B. $b_{22} = P(y_2|x_2)$. Quelle: https://tinyurl.com/yacqwrzq

Features die betrachtet werden können. Ein weiteres Problem ist, dass "dieser Ansatz ein ungeeignetes, generatives Modell mit gemeinsamen Wahrscheinlichkeiten $[P(s, w)]$ berechnet, obwohl [NLP]-Probleme nur bedingte Wahrscheinlichkeiten $[P(s|w)]$ benötigen", so McCallum et al. [26]

2.1.2 Maximum Entropy Markov Modelle

Um die im Abschnitt 2.1.1 genannten Defizite zu beheben, haben sich HMMs unter anderem zu *Maximum Entropy Markov Models* (MEMMs) weiterentwickelt.

Im Kontrast zum generativen HMM, sind MEMMs konditionale Modelle, welche nicht wie HMMs die Wahrscheinlichkeit $P(S, W)$ berechnen, sondern lediglich Wahrscheinlichkeiten versteckter Zustände bezüglich einer gegebenen Emissionen [18], also $P(S|W)$. Nach McCallum et al. [26] wird die Übersetzungsfunktion $P(s_i|s_{i-1})$ und Beobachtungsfunktion $P(w_i|s_i)$ eines HMMs durch $P(s_i|s_{i-1}, w_i)$ ersetzt. Man betrachtet den neuen Zustand s_i in Abhängigkeit zur dazugehörigen Emission w_i und den vorherigen Zustand s_{i-1}. Emissionen können folglich mit einem Zustandsübergang assoziiert werden.

Um neben Wort- und Zustandsbeziehungen weitere Features im Modell zu nutzen, verwenden MEMMs nicht den *Naïve Bayes* Ansatz, sondern das *Maximum Entropy Framework*. (ME). Im Gegensatz zum *Naïve Bayes*, basieren ME-Modelle auf bedingten Wahrscheinlichkeitsverteilungen. Diese ermöglichen es, "unter gegebenen Einschränkungen (Features) des Trainingskorpus die A-priori-Wahrscheinlichkeit [Anfangswahrscheinlichkeit] zu maximieren, ohne mit beliebigen Annahmen die Allgemeingültigkeit zu verletzten", so Klinger et al. [22].

Jetzt stellt sich natürlich die Frage, was genau sind Features? Features können Eigenschaften wie "Groß- und Kleinschreibung, Wortendungen, POS *(Part-of-Speech)*, Formatierungen, Positionen im Text/Satz, [...]" [26] festhalten. Dadurch lassen sich neben einzelnen Worten auch komplexere Emissionen beschreiben, z.B. ganze Textzeilen. In solchen Fällen kann man die Wortmenge oder spezielle, im Satz enthaltene Zeichen mit Features Beschreiben.

Dargestellt werden Features als Funktionen. McCallum et al. [26] verwenden binäre Funktionen mit $f_k \to \{0, 1\}$, wobei auch beliebige Funktionen verwenden werden können. Angenommen man will ein Feature definieren, das Groß- und Kleinschreibung (GuK)

erkennt, so ließe es sich wie folgt definieren:

$$f_{\langle \mathbf{GuK},s\rangle}(w_i, s_i) = \begin{cases} 1 & w_i \text{ ist großgeschrieben und } s = s_i \\ 0 & sonst, \end{cases} \tag{3}$$

wobei s den Zielzustand beschreibt. Zusätzlich erhält jedes Feature ein Gewicht λ_k. Diese Gewichte müssen aber erst durch Training des Modells bestimmt werden. Je größer eine Gewichtung λ_k bezüglich f_k ist, desto relevanter ist das Feature.

Um jetzt das *Maximum Entropy Framework* mit den neuen Features zu verknüpfen, wurde folgende Gleichung entwickelt:

$$P(s_i|s_{i-1}, w_i) = P_{s_{i-1}}(s_i|w_i) = \frac{1}{Z(w_i, s_{i-1})} \cdot exp(\sum_a \lambda_a \cdot f_a(w_i, s_i)). \tag{4}$$

$Z(w_i, s_{i-1})$ bildet die Summe aller möglichen Wahrscheinlichkeiten und dient der Normalisierung. Innerhalb der Exponentialfunktion werden zu einer gegebenen Emission und Zustand alle Features mit jeweiliger Gewichtung betrachtet und aufsummiert. Diese Möglichkeit zur Verwendung mehrerer Features macht MEMMs weitaus dynamischer als HMMs.

Als Verwendungsbeispiel haben Finkel et al. [17] ein MEMM trainiert, um Namen von Zellkulturen, Zelltypen, Proteinen, RNA und DNA zu erkennen. Dazu haben sie mehrere Features definiert, z.B. das aktuelle Wort, sein Vorgänger und Nachfolger und weitere Worteigenschaften wie Groß- und Kleinschreibung oder ob Zahlen im Wort enthalten sind. In Tests erreicht ihr System beim Finden vollständiger Eigennamen - also Eigennamen, die links- und rechtsseitig korrekt beschränkt werden - eine F1-Score von $\approx 70\%$ (s. Abschnitt 4.1 zu F1-Score).

2.1.3 Conditional Random Fields

Trotz der neuen Möglichkeiten die MEMMs bieten, besitzen sie eine entscheidende Schwachstelle: das *Label Bias Problem*. Wie in Formel 4 zu sehen, werden alle Emissionen w_i mit dazugehörigem Zustand s_i lokal normalisiert:

$$\frac{1}{Z(w_i, s_{i-1})}. \tag{5}$$

Angenommen ein Zustand besitzt wenig Folgezustände, dann wird der Nenner (5) klein und der gesamte Faktor groß. Solche Zustände erreichen höhere Wahrscheinlichkeiten und werden bevorzugt.

Conditional Random Fields (CRFs) sind eine Erweiterung der MEMMs, die das *Label Bias Problem* lösen. Die Idee lautet, alle Wahrscheinlichkeiten nicht mehr lokal, sondern global zu normalisieren, so Lafferty et al. [24]. Auf CRFs soll hier jedoch nicht näher Eingegangen werden, denn obwohl sie heute öfters als MEMMs zum Einsatz kommen, sind sie vom grundlegenden Aufbau her gleich.

Um die stochastischen Modelle abzuschließen, sei auf Abbildung 2 verwiesen, die die genannten Modelle mit ihren jeweiligen Methoden nochmals grafisch zusammenfasst.

2.2 Dictionary Based Annotation

Wörterbuchbasierte Ansätze nutzen Lexikons/Wörterbücher, mit denen Eigennamen in Texten gesucht werden. Meistens kommt dafür Wikipedia zum Einsatz. Ein Vorteil dieser

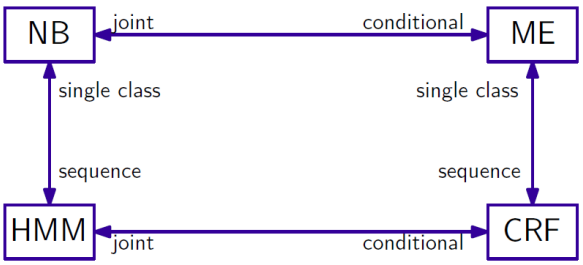

Abbildung 2: Zusammenhänge zwischen allen genannten Modellen (CRF und MEMM lassen sich hier synonym betrachten). NB: Naïve Bayes, ME: Maximum Entropy, HMM: Hidden Markov Model, CRF: Conditional Random Field [22]

Methode ist, dass Wörterbücher in sich eine Wissensbasis bilden. Kann man einer Eingabe einen konkreten Wörterbucheintrag zuordnen, so lässt sich nicht nur eine Kategorie wie PERSON, ORGANISATION, etc. ableiten, sondern auch Beziehungen zu anderen Eigennamen. Neben dem NER-Problem wird gleichzeitig das NEL-Problem gelöst.

Beispiel eines *Dictionary-Based* Ansatzes ist *DBpedia Spotlight* [10]. DBpedia ist ein Projekt, "mit dem strukturierte Informationen aus Wikipedia extrahiert und Webanwendungen zugänglich gemacht werden" [12]. Die Idee ist nun, die gesamte, strukturierte Ontologie von DBpedia mit allen vorgegebenen Klassen zu verwenden, um Eigennamen aus Texten zu filtern und zu kategorisieren. Im Gegensatz zu den Stochastischen Modellen (2.1), die nur zwischen einfachen Klassen wie PERSON, ORGANISATION, ORT, etc. differenzieren, besitzt DBpedia mehrere hundert Klassen die zur Eigennamenerkennung genutzt werden können, so Mendes et al. [10]. Trotz der einfachen Idee, benötigt dieser Ansatz drei wichtige Schritte, um passende DBpedia Ressourcen mit Eigennamen zu verlinken:

1. *Spotting*: Es müssen Ausdrücke in der Eingabe gesucht werden, die womöglich eine DBpedia Ressource darstellen. Für den Satz "Washington liegt nord-westlich der USA" wären **Washington** und **USA** solche Ausdrücke.

2. *Candidate Selection*: Hier werden mögliche DBpedia Ressourcen für die gefundenen Ausdrücke gesucht. Für **Washington** aus dem vorherigen Beispiel existieren folgende Ressourcen: **George_Washington, Washington,_D.C. und Washington_(U.S._state)**.

3. *Disambiguation*: Es wird für jeden Ausdruck bezüglich seines Kontext die passende DBpedia Ressource ausgewählt. Im Bespiel wird deutlich, dass es sich hier bei **Washington** um den Bundesstaat - **Washington_(U.S._state)** - handeln muss.

Von den drei Schritten dient insbesondere der erste zur *Entity Recognition*. Schritt zwei und drei dagegen dem *Entity Linking*. Der komplexeste Schritt dieses Verfahrens ist dabei die Disambiguierung, also Schritt 3, da zu jedem möglichen Ausdruck ein Kontextraum bestimmt werden muss. Unter Kontext versteht man hierbei benachbarte Wörter bis hin zu ganzen Paragraphen. Um diesen Raum zu bestimmen und zu vergleichen, kommt ein *Vector Space Model* (VSM) zum Einsatz. Trotz hiesiger Verwendung für *Entity Linking* wird im folgenden Abschnitt 2.3 näher auf VSMs eingegangen, da diese auch in modernen Ansätzen für NER große Relevanz aufweisen.

Abschließend lässt sich zum *Dictionary Based* Ansatz sagen, dass dieser zwar das NER- und NEL-Problem gleichzeitig löst, jedoch starke Probleme beim Auffinden relevanter Ausdrücke aufzeigt. Seltene, nicht klassifizierte Wörter, Rechtschreibfehler in der Eingabe oder nicht-wörtliche Ausdrücke wie Jahreszahlen bereiten starke Probleme, so Arnold et al. [3]. Dies wird beim Vektormodell von Mirkov et al. [29] deutlich, welches 27% der Eigennamen im GENIA Datensatz - ein Datensatz für medizinische Daten - nicht findet, schlichtweg weil sie nicht im Wörterbuch enthalten sind.

2.3 Vector Space Modelle

Ein moderneres Konzept zur Realisierung von *Entity Recognition* sind neuronale Netzwerke. Dabei handelt es sich um Funktionen der Art $\mathbb{R}^n \to \mathbb{R}^m$, wobei $\mathbb{R}^n$ der Input in Form eines Wortes ist und $\mathbb{R}^m$ ein dazu passender Output (Label). Wie genau diese Netzwerke funktionieren, soll im Abschnitt 3.1 behandelt werden. In diesem Kapitel sollen lediglich Möglichkeiten erläutert werden, um Wörter als Vektor reeller Zahlen darzustellen.

2.3.1 One-Hot Encoding und Bag-of-Words Modelle

Beim *One-Hot Encoding* wird zur Kodierung einer Objektmenge $\mathcal{O}$, mit $|\mathcal{O}| = N$ der Vektorraum $\mathbb{R}^N$ verwendet. Ein erlaubter Vektor zur Kodierung eines Objekts $o \in \mathcal{O}$ besteht dabei aus einer 1, die restlichen Werte sind 0 [32]. Zur Kodierung von Wörtern kann man $|\mathcal{O}|$ als Menge von Einträgen in einem Wörterbuch verstehen. Jeder *One-hot* Vektor kodiert ein Wort dieses Wörterbuches.

Im Gegensatz dazu, kodieren *Bag-of-Words* Modelle mehrere Objekte mit einem Vektor gleichzeitig. Angenommen man möchte einen Satz kodieren, so wird für jedes Wort der *One-hot* Vektor gebildet und diese aufsummiert [4].

2.3.2 Das Continuous Skip-Gram Modell

Da beide Varianten vom vorherigen Abschnitt 2.3.1 kaum aussagekräftige Kontextinformationen liefern, gilt es diese zu verbessern. *Skip-Gram* Modelle nutzen ein Wort als Input, um andere Wörter in einem bestimmten Bereich um den Input vorherzusagen. Dies hat zur Folge, dass ähnliche Wörter wie “klug” und “intelligent”, oder Wörter der gleichen Domäne ähnliche Vektoren erzeugen, so McCormick [27].

Passende Vektorrepräsentationen der Wörter müssen jedoch erst durch ein eigenes, neuronales Netzwerk trainiert werden, bevor man dieses Methode in anderen Anwendungen verwenden kann. Dieser Overhead hat zur Folge, dass auch heute oft *One-hot Encoding* oder einfache Abwandlungen davon zum Einsatz kommen.

3 Ein Neuer Ansatz: DATEXIS-NER

3.1 Die Verwendung neuronaler Netzwerke

Im Abschnitt 2 wurden verschiedene Ansätze zur Lösung des NER-Problems vorgestellt. Hier soll eine neue Möglichkeit aufgezeigt werden: Künstliche neuronale Netzwerke. Diese, “kurz: KNN [...], sind Netze aus künstlichen Neuronen. Sie sind Forschungsgegenstand der Neuroinformatik und stellen einen Zweig der künstlichen Intelligenz dar. [...]” [23]. KNNs haben das Ziel, ein abstrahiertes Modell der menschlichen Informationsverarbeitung zu erstellen, um insbesondere Aufgaben zu lösen, welche Menschen einfach erscheinen, von

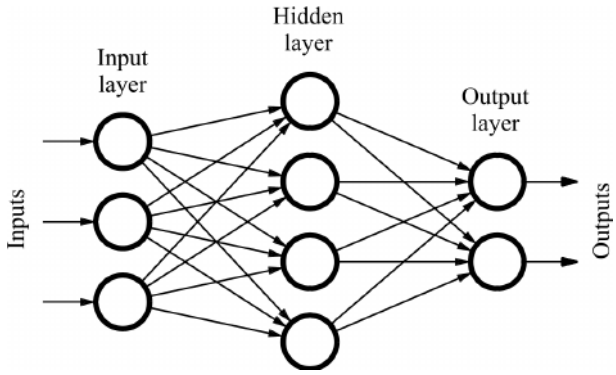

Abbildung 3: Aufbau eines FF-Netzwerks aus Neuronen und Kanten mit Input-, einem Hidden- und Output-Layer. Quelle: https://tinyurl.com/ybxu8dpg

Computern aber nur schwer umsetzbar sind. Zu solchen Aufgaben zählt neben NER auch Objekterkennung in Bildern.

3.1.1 Grundlage - Feed Forward Networks

Zu den einfachsten Arten Künstlicher neuronaler Netzwerke zählen *Feed Forward Networks* (FF-Netzwerke). Diese bestehen aus drei Teilen: Einem Input-Vektor $\mathbb{R}^m$, einer bestimmten Anzahl *Hidden Layers*, und einem Output-Vektor $\mathbb{R}^n$, s. Abbildung 3. Einzelne Werte dieser Komponenten werden als Neuronen (Knoten) bezeichnet. Zur Berechnung des Outputs werden Input-Neuronen mit Hilfe trainierter Gewichte mit einem *Hidden Layer* verknüpft. Genauso werden aufeinanderfolgende *Hidden Layers* miteinander verbunden, und der letzte *Hidden Layer* mit den Output-Neuronen. In einem Knoten eingehende Kantenwerte, werden aufsummiert und mit einer Aktivierungsfunktion eingeschränkt.

Es sei vereinfacht angenommen, man hat ein FF-Netzwerk mit nur einem *Hidden Layer* gegeben. Dann lässt sich das Netzwerk in Matrix-Form aufschreiben [9]:

$$\begin{aligned} h &= \phi(W \cdot x) \\ y &= V \cdot h \end{aligned} \tag{6}$$

- x ist der Eingabevektor $\mathbb{R}^m$
- W ist eine Wichtungsmatrix der Form $\mathbb{R}^{k \times m}$
- ϕ ist die Aktivierungsfunktion für die Werte des *Hidden Layers* h.
- V ist die zweite Wichtungsmatrix $\mathbb{R}^{n \times k}$ um den *Hidden Layer* mit dem Output y zu verbinden.

Als Aktivierungsfunktionen finden insbesondere drei Stück in vielen Fällen Anwendung. Zum einen die Sigmoid Funktion:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{7}$$

die Eingaben auf den Bereich $(0, 1)$ abbildet. Als nächstes die Hyperbelfunktion:

$$tanh(x), \tag{8}$$

die den Wertebereich $(-1, 1)$ besitzt. Und zuletzt die *Rectified linear Unit*:

$$ReLU(x) = max(0, x) \tag{9}$$

Nachdem die grundlegenden Definitionen zum FF-Netzwerk klar sind, gilt es dieses zu trainieren. Das heißt, man will optimale Werte für alle Matrizen - im vorherigen Beispiel W und V - bezüglich eines Trainingskorpus berechnen. Dazu verwendet man *Supervised Learning*. Diese "Methode richtet sich [...] nach einer im Vorhinein festgelegten zu lernenden Ausgabe, deren Ergebnisse bekannt sind. Die Ergebnisse des Lernprozesses können mit den bekannten, richtigen Ergebnissen verglichen, also „überwacht", werden" [5]. Zum Beispiel kann eine Tabelle mit Paaren der Form $P = (Wort \times Label)$ vorhanden sein, wobei gelten könnte: $(USA, Named\_Entity) \in P$ oder $(hallo, No\_Named\_Entity) \in P$.

Ein häufig verwendetes Berechnungsverfahren, das *Supervised Learning* nutzt, lautet *Gradient Descent* und läuft folgendermaßen ab:

1. Die zu trainierenden Parameter bekommen einen randomisierten Startwert, typischerweise zwischen 0 und 1.

2. Es wird über alle richtigen Paare (x, y) iteriert und die zu trainierenden Parameter stückweise angepasst.

Zur Bestimmung, inwieweit Parameter angepasst werden müssen, gilt es eine Kostenfunktion zu minimieren:

$$J = \frac{1}{n} \sum_i (y_i - f(x_i))^2 \tag{10}$$

n ist die Menge aller richtigen Wertpaare, y_i ist das korrekte Ergebnis und $f(x_i)$ das Ergebnis, das vom neuronalen Netzwerk berechnet wird. Diese Funktion summiert alle quadratischen Fehler auf. Berechnet man dazu nun die Partiellen Ableitung bezüglich aller Parameter und subtrahiert das jeweilige Ergebnis vom aktuellen Parameter, dann wird der Parameter so angepasst, dass der Fehler aus Formel 10 kleiner wird:

$$m := m - \alpha \cdot \frac{\delta J}{\delta m} \tag{11}$$

Dieses Verfahren wird auch als *Backpropagation* bezeichnet. α ist die Lernrate, also wie schnell der Algorithmus zu einem Minimum konvergieren soll. Ein zu großes α kann dafür sorgen, dass der Algorithmus vom Minimum divergiert, deswegen sollte α nie zu groß sein.

Es gibt noch zwei weitere, relevante Verfahren neben dem Standard *Gradient Descent*: *Stochastic Gradient Descent* und *Mini Batch Gradient Descent*. Beide Verfahren haben das Ziel, die Trainingszeit für sehr große Trainingsdatensätze zu optimieren. Der Unterschied zum Standardverfahren ist der, dass Kosten und Gradienten nicht mehr über den kompletten Trainingsdatensatz, sondern nur noch über ein *(Stochastic)* bzw. wenig *(Mini Batch)* Trainingsbeispiele berechnet werden. In beiden Verfahren kann mehrmals über alle Trainingspaare iteriert werden, bis die Gesamtkosten einen ausreichend kleinen Wert erreichen. Zwar sind beide Verfahren sehr schnell, jedoch nähert sich die Lösung nur ungleichmäßig einem Optimum an [16].

Die hier vorgestellten FF-Netzwerke sind äußerst simpel, jedoch können sie nicht mit zeitabhängigen bzw. sequentiellen Daten arbeiten. "FF-Netzwerke leiden unter Amnesie bezüglich vergangener Zustände. Sie erinnern sich nur an den prägenden Moment des Trainings [ohne auf Kontexte zu achten]." [13]

3.1.2 Recurrent Neural Networks

Um das Problem der fehlenden Erinnerung eines FF-Netzwerkes anzugehen, haben sich *Recurrent Neural Networks* (RNNs) entwickelt, die einen internen Speicher verwalten, der der Klassifikation von Input-Daten hilft.

RNNs nutzen dazu, wie beim FF-Netzwerk, Informationen des aktuellen Zeitpunktes t, aber auch Informationen vom vorherigen Zeitpunkt $t-1$. Somit existieren pro zu berechnendem Label zwei Eingaben: Der Standard-Input x_t und Informationen des vorherigen *Hidden Layers* h_{t-1}.

Diese Werte können eine große Spannweite von Informationen der Sequenz zur Verfügung stellen, da die jeweilige Information h_{t-1} pro Eingabe im RNN weitergetragen wird. Sie ermöglichen es, " Korrelationen zwischen Events zu finden, die weit auseinander liegen, wobei diese Korrelationen *long-term dependencies* genannt werden" [13]. Angenommen ein RNN besitzt, wie unter Abschnitt 3.1.1, einen Input, einen Output sowie einen *Hidden Layer*, dann ließe sich der Output wie folgt berechnen:

$$\begin{aligned} h_t &= \phi(W \cdot x_t + U \cdot h_{t-1}) \\ y_t &= V \cdot h_t \end{aligned} \tag{12}$$

Der einzige Unterschied zum FF-Netzwerk ist der, dass $U \cdot h_{t-1}$ zum neuen *Hidden Layer* aufaddiert wird. Lediglich beim ersten Wert einer Eingabesequenz wird auf diesen Summanden verzichtet. h_{t-1} bezeichnet Werte des vorherigen *Hidden Layers* bezüglich t und U ist eine neue Wichtungsmatrix zwischen *Hidden Layern*, die genau wie W trainiert werden muss. U gibt an, wie stark der Einfluss vergangener Ereignisse auf eine neue Eingabe ist.

Natürlich lassen sich RNNs nicht mit einfacher *Backpropagation* trainieren, da vorherige Ergebnisse Einfluss auf neue haben. Daher wurde ein neuer Ansatz entwickelt: *Backpropagation Through Time* (BPTT). Dieser unterscheidet sich zur normaler *Backpropagation* nur in der Hinsicht, dass die verschiedenen Zeitschritte miteinander verbunden werden. Die geschachtelte Funktion eines FF-Netzwerks erweitert sich nur um eine Zeitkomponente. Wie bei einfacher *Backpropagation* müssen nun die Ableitung bezüglich der Parameter berechnet und diese Parameter angepasst werden. Da BPTT für lange Trainingssequenzen sehr komplex wird, gibt es verschiedene Möglichkeiten der Approximation, z.B. *Tunctated* BPTT [13].

RNNs können somit zwar sequentielle Daten mit Kontextbezug labeln, jedoch besitzen sie ein großes Problem, das *Vanishing Gradient Problem*. Informationen, die sequentiell durch das Netzwerk geleitet werden, durchlaufen viele Multiplikationen, typischerweise mit Werten im Intervall (-1, 1). Damit gehen Informationen, die in der Vergangenheit liegen, immer weiter verloren. Dies geht weiter bis zum Punkt, an dem womöglich wichtige, in der Vergangenheit liegende Informationen, gar nicht mehr verwendet werden können. Nun stellt sich die Frage: Wie wichtig ist Kontext, der so weit weg liegt? Ist dieser überhaupt relevant? In vielen Fällen nicht, in Sachen NLP aber schon. Zur Verdeutlichung sei folgender Beispielsatz gegeben [31]:

> "Ich komme aus Frankreich und spreche fließend _".

Ein RNN, das ausreichend auf einem passenden Textkorpus trainiert wurde, kann vorhersagen, dass das letzte Wort mit großer Wahrscheinlichkeit "französisch" lauten muss. Der Kontext liegt nah. - Dies sieht im nächsten Beispiel anders aus:

> "Ich komme aus Frankreich [...]. Ich spreche fließend _".

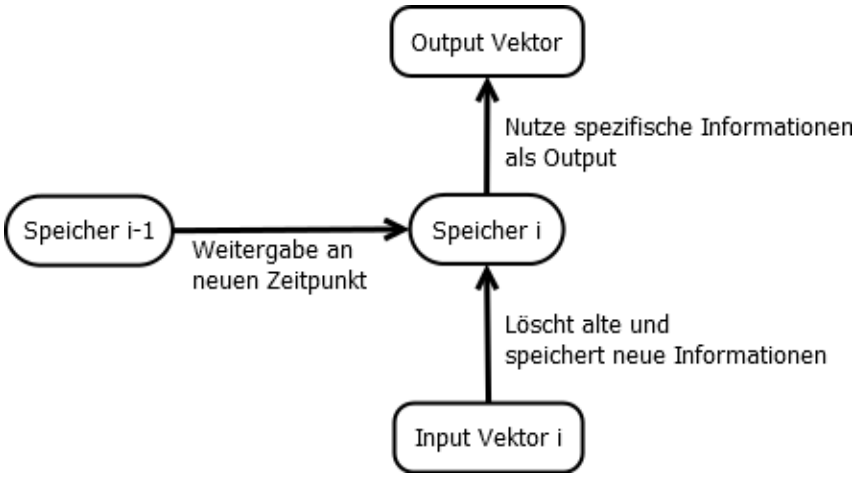

Abbildung 4: Idee des LSTMs. Informationen gelangen zum Speicher und bearbeiten diesen. Betimmte Informationen des Speichers dienen als Output.

In diesem Fall liegt zwischen beiden Informationsteilen eine Lücke. Es kann vorkommen, dass diese Lücke sehr groß ist (> 1.000 Wörter). In solchen Fällen vergisst ein RNN, dass man aus Frankreich kommt und könnte keine Sprache mehr zuordnen, sofern in der Lücke keine weiteren, relevanten Informationen vorhanden sind.

3.1.3 Long short-term Memories

Long short-term Memories, kurz: LSTMs, sind eine Weiterentwicklung von RNNs, die das *Vanishing Gradient Problem* zu lösen versuchen. Die Idee ist Folgende: Vergangene Informationen sollen nicht gespeichert werden, indem sich das Netzwerk daran erinnert, sondern so, dass man von jedem Zeitpunkt aus auf beliebige Informationen der Vergangenheit zugreifen kann. Dazu ist ein echter Speicher notwendig. Inputs müssen auf den Speicher zugreifen und diesen aktualisieren können. Danach sollen relevante Informationen des aktualisierten Speichers gefiltert und als Output verwendet werden. Vereinfacht wird das Konzept in Abbildung 4 gezeigt. Damit ist es möglich, "Zeitintervalle mit mehr als 1.000 Schritten zu überbrücken, selbst bei rauschenden, nicht komprimierbaren Input-Sequenzen, ohne nah aufeinanderfolgende Zusammenhänge zu benachteiligen", so Hochreiter und Schmidhubert [21]. Wie genau der Speicher und der Rest des LSTMs realisiert wird, hat Christopher Olah auf seinem Blog [31] kurz und präzise zusammengetragen. Auf Abbildung 5 sieht man den Aufbau eines LSTMs. Im Gegensatz zum RNN durchläuft jeder Input vier unabhängige, neuronale Netzwerkschichten. Ein weiterer Unterschied ist, dass jeder Zustand mit zwei Informationen des vorherigen Zustandes plus Input arbeitet und auch zwei neue Informationen weiterleitet.

Die wichtigste Komponente dieses Modells ist C_{t-1} bzw. C_t. Dabei handelt es sich um den bereits genannten Speicher, auf dem die Inputs arbeiten müssen. Inputs sollen jedoch nicht beliebig viele Informationen löschen oder speichern, die Menge wird mit sog. *Gates* reguliert. *Gates* schränken den Informationsfluss mit der *Squashing*-Funktion Sigmoid (7).

Bevor das LSTM mit den Berechnungen beginnt, wird der Input angepasst, denn dieser setzt sich aus der normalen Eingabe x_t und dem Output h_{t-1} des vorherigen Layers

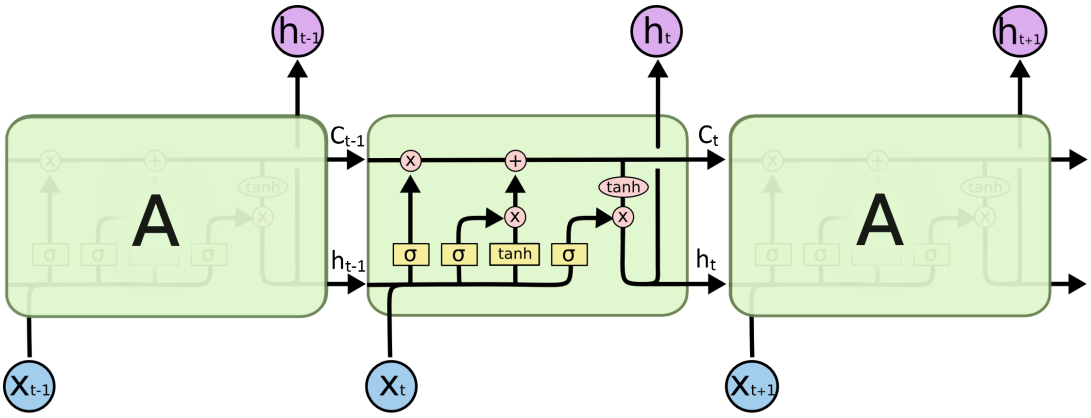

Abbildung 5: Darstellung eines LSTMs zum Zeitzustand t. Quelle: [31]

zusammen. Beides sind Vektoren die konkateniert werden. Beispiel:

$$\begin{pmatrix} a \\ b \end{pmatrix} \circ \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a \\ b \\ x \\ y \end{pmatrix} \tag{13}$$

Als Kurzschreibweise für $a \circ b$ soll $[a, b]$ dienen.

Die erste Komponente eines LSTMs ist das *Forget Gate*, um zu bestimmen, welche Informationen vom vorherigen Speicherzustand C_{t-1} gelöscht werden sollen. Die Gleichung dazu sieht wie folgt aus:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{14}$$

W_f ist die zu trainierende Matrix, welche mit dem konkatenierten Input $[h_{t-1}, x_t]$ multipliziert wird. Standardmäßig wird in neuronalen Netzen ein trainierter *bias*, hier b_f, dazu addiert. Da es sich bei f_t um ein Gate handelt, wird das Ergebnis des Layers mit einer Siqmoid-Funktion 7 gesquasht. Um mit diesem Ergebnis den Speicher von C_{t-1} partiell zu löschen, werden beide Werte miteinander, punktweise multipliziert ($\otimes$). Interpretieren lässt sich dieser Vorgang so: Die Sigmoid-Funktion bildet alle Werte eines Vektors auf den Wertebereich (0, 1) ab. Diese Werte kann man als prozentuale Anteile verstehen. Das Multiplizieren von C_{t-1} und f_t hat lediglich zur Folge, dass gesagt wird, wie viel Prozent an Informationen pro Speichereintrag behalten werden. Beispiel:

$$\begin{aligned} f_t = \begin{pmatrix} 0.8 \\ 0.1 \end{pmatrix}, C_{t-1} = \begin{pmatrix} 2.5 \\ -1.7 \end{pmatrix} \\ f_t \otimes C_{t-1} = \begin{pmatrix} 80\% \\ 10\% \end{pmatrix} \otimes \begin{pmatrix} 2.5 \\ -1.7 \end{pmatrix} = \begin{pmatrix} 2 \\ -0.17 \end{pmatrix} \end{aligned} \tag{15}$$

Eine Auffälligkeit hierbei ist, dass gelten muss: $C_{t-1} \in \mathbb{R}^c$ und $f_t \in \mathbb{R}^c$. Weiterhin muss nach Gleichung 14 gelten: $W_f \in \mathbb{R}^{c \times n}$ wobei $[h_{t-1}, x_t] \in R^n$. c annotiert hierbei die *Number of Cell States* innerhalb eines LSTMs, also wie groß der Speicher ist. Je größer der Speicher ist, desto mehr Informationen können über weite Zeiträume beibehalten werden. Es gilt darauf zu achten die Speichergröße in Abhängigkeit der Menge gegebener Trainingsdaten zu wählen, denn kleine Datenmengen brauchen nur einen kleinen Speicher.

Um zu berechnen, welche neuen Informationen dem Speicher hinzugefügt werden, kommt ein ähnliches Prozedere zum Einsatz. Zuerst soll bestimmt werden, welche Informationen bezüglich des konkatenierten Inputs gespeichert werden. Hierfür kommt eine

neuronale Netzwerkschicht zum Einsatz, wobei das Ergebnis mit *tanh* gesquasht wird. Der Wertebereich aller Vektoreinträge beträgt somit (-1, 1). Man will keine prozentualen Anteile, sondern konkrete Informationen berechnen, die auch negativ sein können. Ein weiterer Vorteil ist die kurze Berechnungszeit der Ableitungen gegenüber der Sigmoid-Funktion [2]. Die Netzwerkschicht hat folgenden Aufbau:

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_c) \tag{16}$$

Diese ist analog zur Gleichung 14, aber mit eigener Matrix W_C und *tanh*-Aktivierung. Die mit dieser Funktion gefundenen Informationen $\tilde{C}_t$ werden noch nicht gespeichert. Zuerst muss bestimmt werden, welche Informationen dieses Vektors wie relevant sind, um zu verhindern, dass der Speicher mit irrelevanten Daten vollgeschrieben wird. Dazu wendet man, ähnlich zum *Forget Gate*, eine eigene Netzwerkschicht mit Sigmoid-Aktivierung an. Das *Input Gate*:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{17}$$

Multipliziert man nun beide Informationen $\tilde{C}_t$ und i_t, geschieht das gleiche wie im Beispiel 15: Nur bestimmte, prozentuale Anteile der neuen Informationen bleiben erhalten. Um diese Informationen zu speichern, werden sie zum Speicher aufaddiert. Die Vollständige Gleichung zur Aktualisierung des Speichers lautet:

$$C_t = (f_t \otimes C_{t-1}) + (i_t \otimes \tilde{C}_t) \tag{18}$$

Somit erhält man den neuen Speicherzustand C_t. Der erste Summand löscht alte Informationen und der zweite fügt neue hinzu.

Mit diesem Speicherzustand C_t gilt es herauszufinden, was als Output fungieren soll. Da nicht alle Informationen des Speichers relevant sind, wird auch hier ein *Gate* verwendet, das *Output Gate*:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{19}$$

Zuerst wird der Speicher mit der tanh-Funktion präpariert, da die Addition neuer Informationen Werte erzeugen kann, für die gilt $x \notin (-1, 1)$. Dann wird das Ergebnis mit dem *Output Gate* punktweise multipliziert, man erhält:

$$h_t = o_t \otimes tanh(C_t) \tag{20}$$

h_t ist der Output und wird an den nächsten Zeitzustand weitergegeben. Da $C_t \in \mathbb{R}^c$ sein muss fällt auf, dass auch $h_t \in \mathbb{R}^c$ gilt (aufgrund der punktweisen Vektormultiplikation). Man kann weiterhin darauf schließen, dass $[h_{t-1}, x_t] \in \mathbb{R}^{c+n}$ gilt, wobei n die Dimension des Inputs x_t ist. Mit diesem Wissen kann man das LSTM auch als Funktion $R^n \times R^c \times R^c \rightarrow R^c \times R^c$ darstellen [14].

Damit ist das LSTM bezüglich des Aufbaus vollständig. Lediglich das Training der vier, einzelnen Netzwerkschichten fehlt. Doch genau wie beim RNN geschieht dies mit Hilfe von BPTT. Zusammenfassend erhält man somit ein Netzwerk zur Verarbeitung sequentieller Daten, das das *Vanishing Gradient Problem* der RNNs mit Hilfe eines eigenen Speichers C_t umgeht.

3.1.4 Bidirectional LSTMs

Einfache LSTMs mögen zwar einen weiten Kontextraum abdecken, jedoch besitzen sie auch Makel. Ein wichtiger Punkt ist der, dass Kontext nur linksseitig gelernt wird. Zeitpunkte

können nur auf ihre Vergangenheit blicken, jedoch nicht in die Zukunft. Um dieses Problem zu umgehen haben sich *Bidirectional* LSTMs entwickelt. Bei diesem Verfahren werden zwei LSTMs benötigt: $LSTM_1$, $LSTM_2$. Das $LSTM_1$ wird normal auf den Trainingsdaten trainiert. Beim $LSTM_2$ wird der gesamte Trainingskorpus reversiert, also von hinten nach vorne betrachtet. Somit lernt das $LSTM_2$ den rechtsseitigen Kontext und $LSTM_1$ den linksseitigen. Gibt man diesen LSTMs nun einen Satz der Länge n als Eingabe, dann wird dieser in das $LSTM_1$, und reversiert in das $LSTM_2$ gegeben. Die Ergebnisse eines Inputs werden konkateniert und dienen als Ausgabe: $h_t = h1_t \circ h2_{n-t}$.

Typischerweise erreichen BLSTMs bei NLP-Problemen geringfügig bessere Ergebnisse als LSTMs. Schließlich ist in natürlichsprachlichen Eingaben nie klar, wo genau sich Kontext zu einem Wort befindet. [33]

3.2 Aufbau von DATEXIS-NER

3.2.1 Vorverarbeitung des Inputs

Da DATEXIS-NER Eigennamen aus Texten filtert, liegen Eingaben in Form einzelner Sätze, Paragraphen oder ganzer Texte vor. Ein Satz eines Dokuments lässt sich als Wortsequenz $w = (w_0, w_1, ..., w_n)$ darstellen. Ein gefundener Eigenname in w ist die längste Sequenz aufeinanderfolgender Wörter, die sich auf einen, konkreten Eigennamen beziehen. Somit werden Eigennamen links- und rechtsseitig eingegrenzt. Weitere Annahmen sind, dass Eigennamen weder rekursiv sind, noch sich gegenseitig überlappen. Rekursion würde bedeuten, dass Eigennamen vollständig in anderen Eigennamen liegen können. Bei Überlappung dahingegen bedeutet es, dass zwei verschiedene, benachbarte Eigennamen sich gleiche Wörter teilen.

Gegebene Wortsequenzen werden an ein neuronales Netzwerk weitergegeben. Dazu könnte man, wie in Abschnitt 2.3, Wörter mit *One-Hot Encoding* einlesen oder *Skip-Gram* Modelle nutzen. Solche Verfahren, insbesondere das *One-Hot Encoding*, können jedoch fatale Probleme aufweisen [3]:

- Rechtschreibfehler machen Eingaben unbrauchbar
- POS Fehler
- Groß- und Kleinschreibung werden ignoriert
- Unbekannte Wörter und irregulärer Kontext können nicht gehandhabt werden

Aus diesem Grund findet ein Verfahren Anwendung, das Huang et al. [1] entwickelt haben. In ihrer Arbeit haben sie sich mit Web-Suchen auseinandergesetzt, in welchen auch Term/Wort-Vektoren vorkommen. Nach ihnen würde einfaches *One-Hot Encoding* wegen des gigantischen Vokabulars einen viel zu großen *Input Layer* erzeugen, der die Trainingsgeschwindigkeit stark einschränkt. Als Lösung haben sie ein *Word Hashing* entwickelt.

Word Hashing soll die Dimension der Wort-Vektoren stark einschränken. Das Verfahren basiert auf n-Grammen (hier konkret *Letter*-Trigrammen) und funktioniert wie folgt: Angenommen man hat das Wort "Test" gegeben. Dann werden zuerst Hashes am Anfang und Ende des Wortes gesetzt → "#Test#". Daraufhin werden alle Möglichkeiten, drei (Tri) aufeinanderfolgende Zeichen (*Letters*) aus der Eingabe zu extrahieren, in einer Liste gespeichert: $Tri_{Test} = [\#Te, Tes, est, st\#]$. Daher die Bezeichnung ''*Letter*-Trigramm". Alle gefundenen Trigramme eines Trainingskorpus werden in ein eigenes Wörterbuch $\mathcal{W}_{Tri}$

zusammengefasst und indiziert. Für die Größe des Wörterbuches gilt: $|\mathcal{W}_{Tri}| = N$, und für einen Vektor, der ein beliebiges Wort w kodieren soll: $v_w \in \mathbb{R}^N$. Um einen spezifischen Vektor v zu einem Wort w zu erstellen, kommt das *Bag-of-Words* Modell (2.3.1) zum Einsatz. Das Wort w wird in seine Trigramme gesplittet und für jedes Trigramm ein *One-Hot* Vektor erstellt, die zum Schluss alle aufaddiert werden. Für $w =$"$Test$" könnte dieses Prozedere wie folgt aussehen:

$$Vec_{Tri,Test} = Vec([\#Te, Tes, est, st\#]) = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \tag{21}$$

mit einem Wörterbuch der Größe 5. Angenommen Trigramme sind nicht im Wörterbuch enthalten, dann tragen diese nicht zum Wortvektor bei.

Der erste Vorteil von Trigrammen ist der, dass Sprachen nur eine eingeschränkte Menge Trigramme besitzen. Jedes Jahr entstehen neue Wörter, was Wortmengen hochdynamisch macht. Dahingegen kann ein Wörterbuch $\mathcal{W}_{Tri}$ nicht beliebig wachsen, da es nur 3^{xs} Kombinationen von Trigrammen gibt, wobei x die Menge verschiedener Symbole in einer Sprache S angibt. In einem Beispiel nach Huang et al. [1] ist ein Trainingskorpus mit 500.000 verschiedene Wörter gegeben. Diese können mit Hilfe von Trigrammen und einem Vektor der Dimension 30.621 dargestellt werden. Die eigentliche Vektorgröße von 500.000 wird um das 16-fache reduziert. Ein weiterer Vorteil ist der, dass unbekannte bzw. falsch geschriebene Wörter keinen 0-Vektor erzeugen. Rechtschreibfehler wirken sich meist nur auf wenige Trigramme aus und auch unbekannte Wörter können Trigramme beinhalten, die bereits gesehen wurden. Durch Betrachtung des Morphologischen Aufbaus von Wörtern mittels Trigrammen ist es also möglich, einem Wort weitaus mehr Informationen zu entziehen, als bei anderen Verfahren, wie *One-Hot Encoding*.

Der einzige Nachteil vom *Letter-Trigram Hashing* sind Kollisionen. Das heißt, zwei verschiedene Wörter können einen gleichen Vektor erzeugen. Jedoch haben Tests von Huang et al. [1] gezeigt, dass Kollisionen für Trigramme so selten vorkommen, dass man sie ignorieren kann, s. Tabelle 1

In DATEXIS-NER reicht jedoch die Morphologie der Wörter nicht aus. Es sollen weitere Features kodiert werden: Wortlänge, Großschreibung am Anfang, innerhalb oder des gesamten Wortes, verwendete Zahlen und Interpunktionen [3]. Für den morphologischen Wortaufbau werden ausschließlich kleingeschriebene Trigramme verwendet. Weitere Features werden als *flag-bits* dem Trigramm-Vektor angehangen. Somit gilt für einen Wortvektor: $v_w \in \mathbb{R}^{N+f}$, wobei f die Menge der verwendeten *Feature-Flags* angibt.

	Letter-Bigram		**Letter-Trigram**	
Word Size	Toke Size	Collision	Token Size	**Collision**
40k	1.107	18	10.306	**2**
500k	1.607	1.192	30.621	**22**

Tabelle 1: Vergleich Kollisionen von Bi- und Trigrammen in Korpi der Größe 40.000 und 500.000. Bigramme erzeugen bei 500k Wörtern 1.192 Kollisionen, für Trigramme jedoch nur 22. [1]

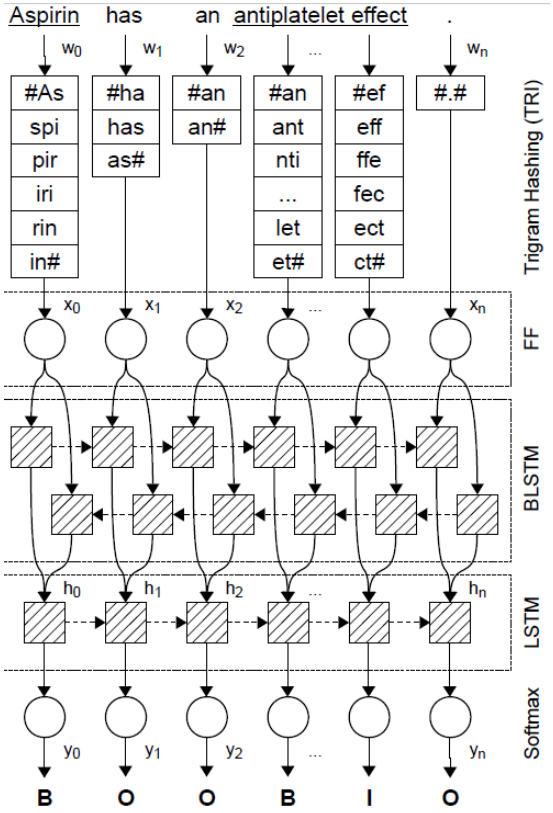

Abbildung 6: Aufbau des Netzwerkes in DATEXIS-NER Quelle: [3]

3.2.2 Aufbau des Netzwerkes

Nach Berechnung der Wortvektoren, werden diese einem zusammengesetzten, neuronalen Netzwerk übergeben, s. Abbildung 6. Die erste Schicht ist ein FF-Netzwerk (3.1.1) mit 150 Neuronen. Trotz *Letter-Trigram Hashing* können Wortvektoren sehr groß sein. Arnold et al. [3] haben für die Evaluation ihres Systems unter anderem mit dem CoNLL2003 Datensatz gearbeitet, in dem 10.831 verschiedene Trigramme gefunden wurden. Die FF-Schicht in DATEXIS-NER soll Eingabevektoren verkleinern und optimieren, für das Beispiel also: $\mathbb{R}^{10.831} \rightarrow \mathbb{R}^{150}$.

Die nächste Schicht ist ein BLSTM (3.1.4) mit 20 *Cell States*. Für den Speicher gilt also $C \in \mathbb{R}^{20}$. Das BLSTM sorgt dafür, links- und rechtsseitigen Kontext in der Eingabe zu filtern. Nach Arnold et al. [3] erzeugt das erste LSTM die Ausgabe $h_t = \overrightarrow{s_t} \otimes \overrightarrow{o_t}$, und das zweite $z_t = \overleftarrow{s_t} \otimes \overleftarrow{o_t}$. Beide Ergebnisse werden konkateniert: $[h_t, z_t]$ und der nächsten Schicht weitergegeben. Trainiert wird dieser und folgender Netzwerkabschnitt mit BPTT und dem *Stochastic Gradient Descent* Ansatz (3.1.1) mit einer Lernrate $\alpha = 0.005$.

Im dritten Abschnitt des Netzwerkes kommt ein weiteres LSTM vor, auch mit 20 *Cell States*. Dieses kann sich den gefundenen links- und rechtsseitigen Kontext zu Nutzen machen, um die finalen Eigennamen zu berechnen. Es ergibt sich ein Vektor $k_t = s_t \otimes o_t$ mit $k_t \in \mathbb{R}^{20}$.

Im letzten Teil der Berechnung wird das Ergebnis k_t an einen *3-Class Softmax Classifier* weitergegeben, um konkrete Ergebnisse zu liefern, ob es sich beim aktuellen Wort um ein Eigenname handelt oder nicht. Wie genau diese Ausgabe funktioniert, wird im nächsten Abschnitt beschrieben.

3.2.3 Output

Als Output soll jedem Wort einer Eingabesequenz $w = (w_0, w_1, ..., w_n)$ ein Label zugeordnet werden: $y = (y_0, y_1, ..., y_n)$. Als Labels wird der BIO2 Standard verwendet. Es gilt: $y_i \in \{B, I, O\}$ für $i \in [0, ..., n]$. B *(Begin)* kennzeichnet den Anfang und I *(Inside)* das Innere eines Eigennamens. O *(Outside)* bedeutet, dass das Wort kein Teil eines Eigennamens ist. Dieser Standard ermöglicht es, Grenzen gefundener Eigennamen einfach abzulesen.

Um nun rechnerisch zu bestimmen, welches Label einem Wort angehört, wird die letzte Ausgabe k_t des Abschnitts 3.2.2 mit einem *3-Class Softmax Classifier* wie folgt verarbeitet:

$$y_t = softmax(W_y \cdot k_t + b_y), \tag{22}$$

mit $W_y \in \mathbb{R}^{3\times 20}$ und $b_y \in \mathbb{R}^3$. Innerhalb der Funktion wird das Ergebnis k_t in einen Vektor $k'_t = W_y \cdot k_t + b_y, k'_t \in \mathbb{R}^3$ umgewandelt, daher auch die Bezeichnung *3-Class*. Ziel ist es, dass jeder Vektoreintrag von k'_t eine der Klassen aus {B, I, O} repräsentiert:

$$\begin{pmatrix} P(B) \\ P(I) \\ P(O) \end{pmatrix},$$

wobei P die jeweilige Wahrscheinlichkeit einer Klasse angibt. Die *Softmax* Funktion hat die Form $\mathbb{R}^K \to (0,1)^K$ [37] und lautet:

$$softmax(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, j = 1, ..., K. \tag{23}$$

Als Beispiel sei folgender Vektor gegeben, der mit einem *3-Class Softmax Classifier* verarbeitet wird:

$$v_w = \begin{pmatrix} 1.5 \\ -3 \\ 5.2 \end{pmatrix}, sum_{\langle e, v_w \rangle} = e^{1.5} + e^{-3} + e^{5.2}$$

$$softmax(v_w) = \begin{pmatrix} e^{1.5}/sum_{\langle e, v_w \rangle} \\ e^{-3}/sum_{\langle e, v_w \rangle} \\ e^{5.2}/sum_{\langle e, v_w \rangle} \end{pmatrix} = \begin{pmatrix} 0.0003 \\ 0.0241 \\ 0.9756 \end{pmatrix}.$$

Der unterste Wert hat mit 97,56% die höchste Wahrscheinlichkeit. Um nun ein Label festzulegen, nutzt man den *Hard Max* Ansatz, das heißt, der wahrscheinlichste Wert wird 1, der Rest 0. Besitzen mehrere Wörter die gleiche Wahrscheinlichkeit, wird nur einer dieser ausgewählt. Damit gibt es für jedes Wort lediglich eine Lösung. Im obigen Beispiel bekäme die Eingabe w das Label O. Kurz gesagt dient der *Softmax* dazu, "die größten Werte einer Menge hervorzuheben und die Werte zu unterdrücken, die weit unter dem maximalen Wert liegen" [37] Durch das Verwenden der Exponentialfunktion, wächst der Funktionswert mit größer werdenden Eingaben immer schneller an, was diese Hervorhebung ermöglicht.

4 Evaluation

4.1 Precision, Recall und F1

Um NER-Systeme nach ihrer Effizienz vergleichen und ordnen zu können, existieren drei Kenngrößen: *Precision*, *Recall* und der *F1-Score*. "Precision [...] gibt den Anteil der korrekt

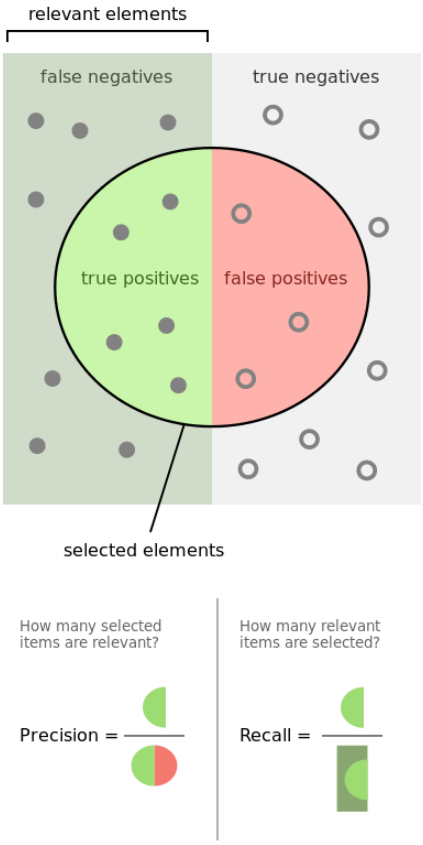

Abbildung 7: Berechnungen von Precision und Recall. Quelle: https://tinyurl.com/jkjvcke

als positiv klassifizierten Ergebnisse an der Gesamtheit der als positiv klassifizierten Ergebnisse an. [...] Recall [...] gibt den Anteil der korrekt als positiv klassifizierten Objekte an der Gesamtheit der tatsächlich positiven Objekte an" [6], s. Abbildung 7. Im einfachsten Fall könnte man alle gefundenen Labels einzeln an Hand eines Evaluationskorpus betrachten und schauen, ob sie richtig zugeordnet wurden oder nicht. Somit kann man die *BIO2 Label Performance* [3] bestimmen.

$$\begin{aligned} Prec_{BIO} &= \frac{1}{3} \cdot \sum_{c \in \{B,I,O\}} \frac{t_{p_c}}{t_{p_c} + f_{p_c}} \\ Rec_{BIO} &= \frac{1}{3} \cdot \sum_{c \in \{B,I,O\}} \frac{t_{p_c}}{t_{p_c} + f_{n_c}} \end{aligned} \tag{24}$$

Es werden sowohl für *Recall* als auch *Precision* die durchschnittlichen Werte über allen Labels $\{B, I, O\}$ bestimmt. Dieser Prozess heißt *Macro-Averaging* und soll Verfälschungen der Resultate wegen einer zu großen O-Klasse verhindern. Für die restlichen Parameter in 24 gilt:

t_{p_c}: Menge von Objekten der Klasse c; Label c wurde zugeordnet. *(True Positives)*

f_{p_c}: Menge von Objekten der Klasse $\bar{c}$; Label c wurde zugeordnet *(False Positives)*

f_{n_c}: Menge von Objekten der Klasse c; Label c wurde nicht zugeordnet *(False Negatives)*

t_{n_c}: Menge von Objekten der Klasse $\bar{c}$; Label c wurde nicht zugeordnet *(True Negatives)*

Um nun *Precision* und *Recall* in einem Ergebnis zusammenzufassen, verwendet man einen *F1-Score*. Dieser berechnet sich über das Harmonische Mittel:

$$F1_{BIO} = \frac{2 \cdot Prec_{BIO} \cdot Rec_{BIO}}{Prec_{BIO} + Rec_{BIO}} \tag{25}$$

Man verwendet das Harmonische Mittel, da hier schlechtere Werte einen größeren Einfluss auf das Ergebnis haben. Man will vermeiden, dass zum Beispiel ein *Recall* von 100% und eine Precision von 0.5% einen großen *F1-Score* erreichen. Sonst könnte man jedes Wort als Eigenname festlegen und hätte trotzdem ein "gutes" System, denn das Label *O* kommt mit Abstand am häufigsten vor.

Da es für NER-Systeme schwierig ist, exakte Grenzen von Eigennamen zu filtern, reicht es oftmals schon Überlappungen zu finden *(Weak Annotation Matches)*. Aus diesem Grund existiert ein *NER-Style F1*. *Precision* und *Recall* werden nicht über die einzelnen Labels bestimmt, sondern darüber, ob das System einen korrekten Eigennamen zumindest überlappt. Dies kann wie folgt aussehen:

Korpus:	B	I	O	O	O	B	I	O	B	O	O
System:	B	I	I	O	O	O	B	I	O	O	O

Im Korpus existieren 3 Eigennamen, die das System finden muss. Jedoch überlappt das System nur 2 der 3 Eigennamen. Es gilt: $Rec = (2/3)$. Des weiteren überlappen alle vom System gefundenen Eigennamen auch die des Korpus, es gilt somit auch: $Prec = (2/2) = 1$. Der *NER-Style F1* für dieses Beispiel würde 80% ergeben.

4.2 Konfigurationen von DATEXIS-NER

Um zu zeigen, wie gut ein BLSTM in Verbindung mit *Letter-Trigram Hashing* ist, wurden verschiedene Konfigurationen für DATEXIS-NER aufgebaut und miteinander verglichen. Insgesamt existieren neun verschiedene Konfigurationen, die sich durch jeweils zwei Merkmale auszeichnen: Die Verarbeitung des Inputs und die Art des Netzwerkes.

Für die Inputverarbeitung wurde neben Trigrammen (TRI) zum einen *One-Hot Encoding* (DICT) verwendet (2.3). Dies ist das einfachste Konzept, jedoch kann der Inputvektor extrem groß werden. Als weiteres Verfahren hat man die *GoogleNews Word2Vec* Einbettung (EMB) verwendet. Solche Einbettungen kodieren Wörter mit Hinblick auf ihren Kontext, weswegen häufig die Architektur von *Skip-Gram* Modellen (2.3.2) zum Einsatz kommt. Des weiteren wird die Vektorgröße im Gegensatz zum *One-Hot Encoding* stark verringert, in diesem Fall auf 300.

Neben dem BLSTM als Netzwerk, wurden auch ein FF-Netzwerk (3.1.1) und ein einfaches LSTM (3.1.3) getestet. Das FF-Netzwerk hat vollständige Verbindungen und insgesamt 3 *Hidden Layers* zu je 150 Neuronen mit ReLU-Aktivierung (9). Das Ergebnis geht an einen *3-Class Softmax Classifier* um das Label zu bestimmen. Die LSTM-Konfiguration ist dahingegen identisch zur BLSTM-Konfiguration, jedoch ohne die zweite (BLSTM-)Schicht.

4.3 Verwendete Datensätze

Insgesamt wurden zwei Modelle mit DATEXIS trainiert. Dazu hat man die beiden Standard Datensätze CoNLL2003 und GENIA verwendet, von denen jeweils zufällig 2.000 Sätze

CoNLL2003	GENIA
Germany NNP I-NP I-LOC 's	Inhibition Inhibition NN B-NP O
representative NN I-NP O	of of IN B-PP O
to TO I-PP O	NF-kappaB NF-kappaB NN B-NP B-protein
the DT I-NP O	activation activation NN I-NP O
European NNP I-NP I-ORG	reversed reverse VBD B-VP O
Union NNP I-NP I-ORG	the the DT B-NP O
's POS B-NP O	anti-apoptotic anti-apoptotic JJ I-NP O
veterinary JJ I-NP O	effect effect NN I-NP O
committee NN I-NP O	of of IN B-PP O
Werner NNP I-NP I-PER	isochamaejasmin isochamaejasmin NN B-NP O
...	...

Tabelle 2: Links ein Ausschnitt des CoNLL2003-Korpus [8] und rechts vom GENIA Korpus [38]

selektiert wurden. GENIA beinhaltet medizinische Daten mit Proteinen, Genen und Zellen. CoNLL2003 dahingegen besitzt allgemeine Eigennamen, also Personen, Orte und Organisationen. Um sich den Aufbau dieser Datensätze vorstellen zu können, sind in Tabelle 2 kurze Ausschnitte abgebildet.

Zum Testen der Allgemeinen Performance, wird DATEXIS-NER zusätzlich mit folgenden Datensätzen evaluiert:

- KORE50: Teilkorpus von AIDA, der wiederum auf dem CoNLL2003 Datensatz basiert. Er beinhaltet nicht eindeutige Eigennamen und Vornamen.
- ACE2004: Datensatz mit chinesischen und arabischen Dokumenten sowie Nachrichten und Broadcasts. Er dient als multilingualer Trainingskorpus.
- MSNBC: Eine Sammlung von Internet *Information Server Logs* für msnbc.com am 28.09.1999. Jeder Eintrag entspricht einem Seitenaufruf eines Users.

4.4 Evaulierung der Konfigurationen von DATEXIS

Zur Evaluierung der neun Konfigurationen (4.2) wird die *BIO Labeling Performance* (4.1) bezüglich CoNLL2003 und GENIA berechnet. Die Ergebnisse sind in Tabelle 8 aufgelistet. Für die Inputkodierung ergibt sich, dass der wörterbuchbasierte Ansatz(2.3.1) (DICT) mit *One-Hot Encoding* die schlechtesten Werte erzielt. Zwar lässt sich das Verfahren gut auf den GENIA-Korpus trainieren und anwenden, da medizinische Begriffe eindeutig sind, jedoch sind in einfachen Texten (News, Broadcasts, ...) oft Wortambiguitäten (Mehrdeutigkeiten) vorhanden, s. CoNLL2003. Dem Netzwerk wird es mit der DICT-Kodierung erschwert zu entscheiden, welche exakte Bedeutung das Eingabewort besitzt. Des weiteren erzeugt der DICT-Ansatz viele *False Negatives*, er labelt also Eigennamen als Kein-Eigenname. Dies liegt hauptsächlich daran, dass das trainierte Wörterbuch nicht vollständig ist oder im Evaluationskorpus Rechtschreibfehler enthalten sind. Im Gegensatz dazu ist das *GoogleNews Word2Vec Embedding* (EMB) bezüglich CoNLL2003 weitaus effizienter. Wortbedeutungen werden erkannt und und mit Hilfe eines kleinen Vektors ($\mathbb{R}^{300}$) dargestellt. Da dieses *Word2Vec* System jedoch vortrainiert ist und viele Wörter, insbesondere im GENIA Korpus unbekannt sind, schneidet das Verfahren für die medizinische Domäne schlechter ab.

Wie auch für DICT leidet diese Variante an einem unvollständigen Wörterbuch und Rechtschreibfehler im Evaluationskorpus. Zum direkten Vergleich: DICT erreicht für CoNLL2003 im Durchschnitt einen *F1-Score* von 67.8% und für GENIA 80.6%. EMB erreicht konträr dazu für CoNLL2003 durchschnittlich 87.5% und für GENIA 75.5%.

Ziel ist es nun, Eingaben so zu kodieren, dass sowohl die Vorteile von DICT, als auch die von EMB beibehalten werden. Hierzu kommen Trigramme (TRI) ins Spiel. Trigramme erreichen sowohl für CoNLL2003 als auch GENIA gute *F1-Scores*. Für CoNLL2003 erreichen sie im Durchschnitt 86.4% und für GENIA 83.3%. Durch die Eigenschaft von Trigrammen, morphologische Wortaspekte zu nutzen, bieten sie eine gute Generalisierung auf unterschiedlichen Domänen und können dem Netzwerk die Differenzierung von Wortambiguitäten ermöglichen.

Als nächstes gilt es die Netzwerktypen zu vergleichen. FF-Netzwerke erreichen bezüglich der Inputkodierungen und Korpora im besten Fall eine *F1-Score* von 78.8%. Dies liegt daran, dass Kontexte zwischen Wörtern vollständig ignoriert werden. Jedoch sind Kontexte wichtig um zu entscheiden, ob ein Wort ein Eigenname ist oder nicht. Ein LSTM erreicht Werte zwischen 80% und 92%, also weitaus bessere als FF-Netzwerke. BLSTMs erreichen sogar noch geringfügig bessere Werte, zwischen 80% und 93%. Es kann also auch rechtsseitiger Kontext zur *Entity Recognition* beitragen.

4.5 Evaluierung bezüglich anderer Systeme

Um DATEXIS-NER mit anderen System und Domänen zu vergleichen, werden Nachrichtenkorpora sowie ein medizinischer Datensatz betrachtet. Zur Evaluierung wird im Gegensatz zum vorherigen Abschnitt 4.4 der *NER-Style F1-Score* verwendet. (4.1).

In Abbildung 9 ist eine Tabelle gegeben, in der DATEXIS-NER bezüglich acht aktueller Systeme mit eigenen Herangehensweisen, welche bereits im Abschnitt 2 präsentiert wurden, verglichen wird. Als Datensätze dienen die vier Nachrichtenkorpora, die in 4.3 genannt wurden. Es zeigt sich, dass DATEXIS-NER die besten Ergebnisse für den *Recall* mit $\approx 96\%$ liefert. Die *Precision* ist dahingegen mit $\approx 75\%$ (Ohne den ACE2004-Datensatz zu beachten) weitaus schlechter. Man kann sagen, dass DATEXIS-NER mehr Wert auf die Quantität der Ergebnisse legt als auf die Qualität. Dies ist nicht verwunderlich, da ein Kritikpunkt gegenüber aller aktuellen Systeme der war, dass sie "im wesentlichen durch einen geringen *Recall* eingeschränkt sind", so Pink et al. [34]; und Ergebnisse, die im ersten Schritt vom *Entity Recognition* nicht gefunden werden, können auch später nicht genutzt

Conf guration	**CoNLL2003**			**GENIA**		
Layers	Prec	Rec	F1	Prec	Rec	F1
DICT+FF	55.8	51.1	53.3	75.6	65.8	70.4
EMB+FF	80.9	76.8	78.8	71.0	64.5	67.6
TRI+FF	74.7	74.3	74.5	75.3	74.2	74.8
DICT+LSTM	59.8	84.8	70.2	82.8	87.4	85.1
EMB+LSTM	94.2	89.8	91.9	81.2	78.0	79.6
TRI+LSTM	91.0	92.3	91.6	87.8	84.7	86.2
DICT+BLSTM	81.9	77.9	79.9	84.0	88.9	86.4
EMB+BLSTM	**95.4**	88.5	91.8	76.4	84.1	80.1
TRI+BLSTM	93.7	**92.4**	**93.0**	**88.6**	**89.3**	**89.0**

Abbildung 8: Verschiedene Konfigurationen von DATEXIS-NER [3]

Common Test Sets		**CoNLL2003**			**KORE50**			**ACE2004**			**MSNBC**		
corpus		RCV-1			RCV-1			newswire			MSNBC news		
topic		news (en)			news (en)			news (en)			news (en)		
annotation guideline		named entities			named entities			all mentions			wikif cation		
Annotator	Method	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
Babelfy	POS+DICT	53.8	70.4	61.0	72.1	73.6	72.9	12.1	42.2	18.8	43.3	77.8	55.6
DBpedia Spotlight	DICT	74.7	66.4	70.3	n/a	n/a	n/a	13.0	74.8	22.2	56.2	49.0	52.4
Entityclassif er.eu	Noun phrase	81.2	83.0	82.1	93.8	94.4	**94.1**	13.3	90.5	23.2	76.2	93.9	84.1
FOX	Ensemble	99.1	75.2	85.5	94.6	73.6	82.8	12.4	59.5	20.6	38.3	31.3	34.4
LingPipe MUC-7	LM+HMM	91.5	66.6	77.1	93.9	86.1	89.9	16.3	88.9	27.5	73.3	78.7	75.9
NERD-ML	Ensemble	59.9	72.0	65.4	70.4	82.6	76.0	19.6	47.4	27.7	69.7	57.2	62.8
Stanford NER	CRF+Dist	**99.5**	76.1	86.2	**94.8**	76.4	84.6	18.2	90.9	30.3	**95.2**	84.1	**89.3**
TagMe 2	DICT	68.3	47.7	56.2	66.5	88.2	75.8	**23.2**	71.6	**35.0**	55.6	38.0	45.2
DATEXIS-NER	BLSTM	87.8	**94.6**	**91.0**	92.6	**95.1**	93.8	13.3	**98.0**	23.4	73.3	**98.2**	83.9

Abbildung 9: Vergleich von DATEXIS-NER mit 8 anderen Systemen bezüglich vier Nachrichtenkorpora [3]

Methode	Noun Chunk Spotter	Tagger	Ensemble Learner
Erläuterung	Eigennamen mit Wörterbuch gesucht	HMMs, MEMMs oder CRFs	Kombinationen verschiedener Ansätze
Beispiele	Babelfy Entityclassifier.eu DBpedia Spotlight	Stanford NER LingPipe	FOX NERD-ML

Tabelle 3: Kategorisierung einzelner Systeme zur Lösung des NER-Problems nach ihrer jeweiligen, verwendeten Methode

werden. Vergleicht man nun die *NER-Style F1-Scores* von DATEXIS-NER mit denen der anderen Systeme fällt weiterhin auf, dass DATEXIS-NER meist am besten abschneidet. Dies liegt nicht nur am sehr guten *Recall*, sondern auch der *Precision*, die in jedem Fall mindestens dem Median entspricht.

Nun zu einer genaueren Betrachtung der anderen Systeme. Diese lassen sich, wie in Tabelle 3 gezeigt, kategorisieren. Zum einen existieren *Noun Chunk Spotter*. Sowohl *Precision* als auch *Recall* liegen bei diesen Systemen im unteren Bereich. Für den CoNLL2003-Datensatz erreicht *DBpedia Spotlight* den größten F1-Score mit 70.2%, was im Vergleich zu den anderen Kategorien immer noch am schlechtesten ist. Jedoch besitzten Wörterbuch-basierte Ansätze einen entscheidenden Vorteil: Sie können gefundene Eigennamen direkt zu einer Wissensbasis verlinken. Wie bereits in Abschnitt 2.2 gezeigt, verwendet *DBpedia Spotlight* als Basis *DBpedia*. Für dieses System existiert sogar eine Demo-Webseite. Texte dienen als Input, werden analysiert und gefundene Eigennamen bekommen eine direkte Verlinkung zur Resource [11].

Als zweite Kategorie existieren *Tagger*, also Systeme, die hier stochastische Modelle verwenden. Diese zeigen im CoNLL2003 und KORE50 Datensatz eine hohe *Precision* auf. Insbesondere die *Precision* von *Stanford NER* im CoNLL2003 Datensatz sticht mit 99.5% heraus. Im Kontrast zu DATEXIS-NER legen diese Systeme einen größeren Wert aus Qualität anstatt Quantität. Man kann in den meisten Fällen annehmen, dass gefundene Eigennamen auch solche sind. Jedoch gehen viele Eigennamen verloren (s. geringer Recall),

Medical Test Set		**GENIA**		
corpus		GENIA 3.02		
topic		biomedical (en)		
annotation guideline		medical terms		
Annotator	Method	Prec	Rec	F1
Zhou and Su (2004)	HMM+SVM	76.0	69.4	72.6
Finkel et al. (2004)	MEMM	71.6	68.6	70.1
Settles et al. (2004)	CRF	70.3	69.3	69.8
GENIA tagger	CDN	75.8	67.5	71.4
LingPipe GENIA	DICT	**95.5**	**97.5**	**96.5**
DATEXIS-NER	BLSTM	83.5	85.4	84.4

Abbildung 10: Vergleich von DATEXIS-NER mit anderen Systemen, die auf medizinischen Daten spezialisiert sind bezüglich des GENIA Datensatzes. [3]

welche somit in *Higher Level Tasks* nicht mehr verwendet werden können.

Die letzte Kategorie bezeichnet *Ensembles*. Damit sind Systeme gemeint, die verschiedene Ansätze kombinieren. Eines dieser *Ensembles* ist NERD-ML, das jedoch nur mäßig gute Ergebnisse liefert (Ähnliche Ergebnisse wie die DICT-Ansätze). Dahingegen erreicht FOX, wie STANFORD-NER, sehr gute *Precision*-Werte für CoNLL2003 und KORE50. Besondere Eigenschaften, wie hohe *Precision* oder *Recall*, stechen bei *Ensembles* jedoch nicht heraus.

Eine Auffälligkeit bei der Evaluation aller Systeme sind die besonders schlechten Precision-Werte im ACE-Datensatz sowie eine hohe Varianz im MSNBC-Datensatz. Dies wird jedoch durch die unterschiedlichen Annotationsstandards verursacht und liegt nicht am Inhalt der Korpora.

Da DATEXIS-NER auf verschiedenen Domänen arbeiten muss, wurde zusätzlich der GENIA-Datensatz mit *NER-Style F1-Scores* getestet. Die Ergebnisse sind in Abbildung 10 gezeigt. Mit einer *F1-Score* von 84,4% hält DATEXIS-NER mit den anderen System ohne Probleme mit. Das System beweist seine Fähigkeit zur Generalisierung des NER-Problems auf sehr unterschiedlichen Domänen. Wieso das wichtig ist zeigt LingPipe GENIA. Zwar erreicht das System einen F1-Score von 96.5%, jedoch *overfittet* es das Problem. Das heißt, LingPipe GENIA ist so stark auf GENIA ausgerichtet, dass es Schwierigkeiten hat, auf anderen Domänen zu arbeiten. Es generalisiert das Problem nicht.

5 Zusammenfassung

Zum Schluss lässt sich sagen, dass DATEXIS-NER im Kontrast zu anderen Systemen eine sinnvolle Weiterentwicklung darstellt. Dies ist zu einem der Wortkodierung mit Trigrammen, sowie dem Netzwerktyp (BLSTM) zuzuschreiben. Erstens stellen viele, fremdartige Wörter *(Idiosyncratic Language)* kein großes Problem mehr da. Zwar existieren immer Wörter, die das System nicht kennt oder die schlicht weg falsch geschrieben sind. Doch auch für unbekannte Wörter existieren oft Trigramme, die bereits gesehen wurden. Nullvektoren kommen nur noch extrem selten vor. Als nächstes ist DATEXIS-NER nicht auf einer Domäne spezialisiert. Wie bereits mit den Nachrichtenkorpora und medizinischen Texten

gezeigt, schneidet das System bei beiden ähnlich gut ab. Weiterhin werden nur noch wenig Trainingsdaten benötigt. Lediglich 2.000 Sätze wurden pro Korpus verwendet. Dies liegt zum einen am kontextsensitiven BLSTM, sowie den Trigrammen. Wörter bestehen nicht mehr nur aus einer Informationen, sondern aus mehreren, morphologischen Informationsteilen, welche alle genutzt werden.

Damit werden die am Anfang genannten Probleme erfolgreich gehandhabt. Jedoch ist auch dieses System nicht perfekt. Ein großes Problem ist, dass nur Grenzen von Eigennamen gesucht werden. Das System kann weder Wissensbasen aufbauen, noch grobe Klassifikationen wie PERSON, ORGANISATION, ORT, etc. vornehmen. Dazu müssen andere Systeme verwendet werden.

Letztendlich kann man sagen, dass DATEXIS-NER gezeigt hat, wozu BLSTMs und Trigramme im Bezug zu NLP-Tasks in der Lage sind. Nicht nur werden Ergebnisse verbessert, auch läuft das gesamte Training weitaus effizienter ab. In Verbindung mit einem System, das gefundene Eigennamen einer Ontologie zuordnet, kann DATEXIS-NER einen großen Schritt in Richtung moderner Informationsextraktion darstellen.

Literatur

[1] *Learning Deep Structured Semantic Models for Web Search using Clickthrough Data.* ACM International Conference on Information and Knowledge Management (CIKM), October 2013.

[2] Wasi Ahmad. Understanding lstm networks. https://stackoverflow.com/questions/40761185/what-is-the-intuition-of-using-tanh-in-lstm#, 11 2016. [Online; accessed 17-March-2018].

[3] Sebastian Arnold, Felix Gers, Torsten Kilias, and Alexander Löser. Robust named entity recognition in idiosyncratic domains. 08 2016.

[4] Bag-of-words model. Bag-of-words model — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Bag-of-words_model&oldid=826312636. [Online; accessed 13-March-2018].

[5] Überwachtes Lernen. Überwachtes lernen — Wikipedia, the free encyclopedia. https://de.wikipedia.org/w/index.php?title=Überwachtes_Lernen&oldid=174090305. [Online; accessed 14-March-2018].

[6] Beurteilung eines binären Klassifikators. Beurteilung eines binären klassifikatorsn — Wikipedia, the free encyclopedia. https://de.wikipedia.org/w/index.php?title=Beurteilung_eines_binären_Klassifikators&oldid=174910686. [Online; accessed 20-March-2018].

[7] Daniel M. Bikel, Richard Schwartz, and Ralph M. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34(1):211–231, Feb 1999.

[8] Christophe Cerisara. Ner/corpus/conll-2003/eng.testa. https://github.com/synalp/NER/blob/master/corpus/CoNLL-2003/eng.testa, 11 2014. [Online; accessed 21-March-2018].

[9] Edwin Chen. Exploring lstms. http://blog.echen.me/2017/05/30/exploring-lstms/. [Online; accessed 14-March-2018].

[10] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems (I-Semantics)*, 2013.

[11] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N. Mendes. Sbpedia spotlight demo. http://demo.dbpedia-spotlight.org, 2013. [Online; accessed 21-March-2018].

[12] DBpedia. Dbpedia — Wikipedia, the free encyclopedia. https://de.wikipedia.org/w/index.php?title=DBpedia&oldid=171479200. [Online; accessed 12-March-2018].

[13] DL4J. A beginner's guide to recurrent networks and lstms. https://deeplearning4j.org/lstm.html#feedforward. [Online; accessed 14-March-2018].

[14] Dan Elton. What is the meaning of "the number of units in the lstm cell? https://www.quora.com/What-is-the-meaning-of-\T1\textquotedblleftThe-number-of-units-in-the-LSTM-cell#, 08 2017. [Online; accessed 17-March-2018].

[15] erikt@uia.ua.ac.be. Language-independent named entity recognition. https://www.clips.uantwerpen.be/conll2002/ner/. [Online; accessed 10-March-2018].

[16] Andrew Ng et al. Optimization: Stochastic gradient descent. http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/, 08 2017. [Online; accessed 19-March-2018].

[17] Jenny Finkel, Shipra Dingare, Huy Nguyen, Malvina Nissim, Christopher Manning, and Gail Sinclair. Exploiting context for biomedical entity recognition: From syntax to the web. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and Its Applications*, JNLPBA '04, pages 88–91, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[18] Generative model. Generative model — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Generative_model&oldid=827107391. [Online; accessed 11-March-2018].

[19] Chris Giarratana. User intent: It's the future of seo. https://www.searchenginejournal.com/user-intent-future-seo/184217/. [Online; accessed 10-March-2018].

[20] Hidden Markov Model. Hidden markov model — Wikipedia, the free encyclopedia. https://de.wikipedia.org/w/index.php?title=Hidden_Markov_Model&oldid=174320218. [Online; accessed 10-March-2018].

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[22] Roman Klinger and Katrin Tomanek. Classical Probabilistic Models and Conditional Random Fields, 2007.

[23] Künstliches neuronales Netz. Künstliches neuronales netz — Wikipedia, the free encyclopedia. https://de.wikipedia.org/w/index.php?title=Künstliches_neuronales_Netz&oldid=174192994. [Online; accessed 14-March-2018].

[24] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[25] Markow-Kette. Markow-kette — Wikipedia, the free encyclopedia. https://de.wikipedia.org/w/index.php?title=Markow-Kette&oldid=172334605. [Online; accessed 12-March-2018].

[26] Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 591–598, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[27] Chris McCormick. Word2vec tutorial - the skip-gram model. http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/, 04 2016. [Online; accessed 13-March-2018].

[28] Message Understanding Conference. Message understanding conference — Wikipedia, the free encyclopedia. https://de.wikipedia.org/w/index.php?title=Message_Understanding_Conference&oldid=168193301. [Online; accessed 10-March-2018].

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[30] Named-entity recognition. Named-entity recognition — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Named-entity_recognition&oldid=825190213. [Online; accessed 10-March-2018].

[31] Understanding LSTM Networks. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Online; accessed 15-March-2018].

[32] One-hot. One-hot — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=One-hot&oldid=808377328. [Online; accessed 13-March-2018].

[33] Sujit Pal. Why do we use bi-lstm for named entity recognition instead of normal lstm? https://www.quora.com/Why-do-we-use-Bi-LSTM-for-Named-Entity-Recognition-instead-of-normal-LSTM, 08 2017. [Online; accessed 17-March-2018].

[34] Glen Pink, Joel Nothman, and James R. Curran. Analysing recall loss in named entity slot filling. pages 820–830, 01 2014.

[35] Proper Noun. Prober noun — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Proper_noun&oldid=826164221. [Online; accessed 21-February-2018].

[36] W. Shen, J. Wang, and J. Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, Feb 2015.

[37] Softmax function. Softmax function — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Softmax_function&oldid=829752166. [Online; accessed 19-March-2018].

[38] Yoshimasa Tsuruoka. Genia tagger. http://www.nactem.ac.uk/GENIA/tagger/, 10 2006. [Online; accessed 21-March-2018].