Karlsruhe Institute of Technology

Hashtag Vorhersage für Kurznachrichten von Twitter

Bachelorarbeit von

Sebastian Hennig

an der Fakultät für Informatik
Interactive Systems Labs

Erstgutachter:	Prof. Alexander Waibel
Zweitgutachter:	Prof. Tamim Asfour
Betreuender Mitarbeiter:	Dr. Jan Niehues

7. Dezember 2017 – 6. April 2017

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 5.4.2017

. . . . . .

(Sebastian Hennig)

Zusammenfassung

In sozialen Netzwerken besteht die Möglichkeit seine Inhalte mit Hashtags zu markieren. Somit kann der Inhalt eingeordnet werden und macht ihn für andere auffindbar. Jedoch muss der Markiervorgang noch vom Nutzer selbst ausgeführt werden. In dieser Arbeit werden wir verschiedene Neuronale Netze trainieren um Hashtags, die das Thema eines Inhalts darstellen, vorzuschlagen. Wir vergleichen hierzu verschiedene Netzwerkarchitekturen. Als erstes betrachten wir das Multi-Layer Perceptron mit verschiedenen Kostenfunktionen als grundlegenden Ansatz. Mit Convolutional Neural Networks versuchen wir Merkmale aus den Tweets zu filtern und somit die Ergebnisse zu verbessern. Zur weiteren Verbesserung der Ergebnisse, verwenden wir ein bi-dirketionales Long Short-Term Memory, das die zeitliche Abhängigkeit der einzelnen Wörter in einem Tweet besser erfassen soll. Außerdem stellen wir ein neues Modell zur Hashtag Vorhersage vor, das auf einem Sequence to Sequence Modell basiert und in der Lage ist, Hashtags Zeichen für Zeichen aufzubauen. Dieser Ansatz ermöglicht es der Netz Architektur neue Hashtags, die vorher noch nicht gesehen wurden, zu kreieren und gesehene Hashtags miteinander zu verknüpfen. Somit heben wir die, durch den Datensatz implizierte Beschränkung der Hashtagauswahl, auf den Raum der möglichen Hashtags, auf und ermöglichen es somit auch Tweets zu klassifizieren, deren Thema nicht durch die zur Verfügung stehenden Hashtags erfasst wird.

Inhaltsverzeichnis

1. Motivation

Twitter ist ein Mikroblogging-Dienst mit mehr als 300 Millionen aktiven Benutzern und 50 Millionen Tweets pro Tag[1]. Benutzer können hier Textnachrichten (Tweets) mit einer Beschränkung von 140 Zeichen veröffentlichen. Ein Tweet kann mit einem oder mehreren Hashtag(s) z.B. *#topic* ausgestattet werden und somit durch die Twitter eigene Suchfunktion gefunden werden. Hierbei wird der Hashtag nicht seperat gespeichert, sondern ist fest in einen Tweet eingebunden. Ein Großteil der Tweets, die Hashtags enthalten, haben diese am Ende angehängt. Dem Benutzer sind hier aber keine Grenzen gesetzt und Hashtags können auch in den Text des Tweets miteingearbeitet werden z.B. *"Smoked Pork #pizza THIS is heavenly!!!"*. Ein Tweet kann auch geteilt werden, wenn man den Inhalt eines anderen Benutzers verbreiten will. Hierdurch entstehen jeden Tag dynamisch neue Hashtags, die nicht immer nur die Themen der Tweets markieren, sondern auch Wertungen oder Ironie enthalten können. Durch die Zeichenbeschränkung ist der Inhalt der Tweets oft voller Abkürzungen oder grammatikalischen Fehlern, um so viel wie möglich aus den 140 Zeichen herauszuholen. Somit haben wir ein dynamisches Markierungssystem, das als Basis eine riesige Sammlung an verrauschten Daten hat. Jedoch werden insgesamt nur 10% der Tweets mit einem Hashtag versehen [KCK12] und somit wird immer nur ein kleiner Teil der gesamten Daten bei einer Suche angezeigt. Die Hashtag Vorhersage für Twitter ist also nicht nur eine sinnvolle, sondern auch eine äußerst interessante Natural Language Processing (NLP) Aufgabe. Es gibt viele verschieden Ansätze um NLP Probleme zu lösen. Einer davon, der durch den technischen Fortschritt und die damit einhergehende Steigerung der Rechenleistung wieder an Relevanz gewinnt, ist die Verwendung von Neuronalen Netzen.

1.1. Methoden

Dazu betrachten wir hier verschiedene Neuronale Netzwerk Ansätze.

Zuerst befassen wir uns mit einem einfachen Multi-Layer Perceptron, das aus einer Embedding und einer Hidden Layer besteht. Dabei wird die Auswirkung von zwei verschiedenen Kostenfunktionen auf das Ergebnis betrachtet. Außerdem benutzen wir verschiedene Regularisierungsmethoden sowie einen variierten Gradientenabstieg mit Moment, um die Ergebnisse zu verbessern. Weiterhin sehen wir, wie ADAM als optimiertes Gradientenabstiegsverfahren, noch schneller und stärker konvergiert.

Convolutional Neural Networks (CNN) werden normalerweise zur Lösung von Problemen, bei denen Bilder als Eingaben dienen, wie z.B. Gesichtserkennung verwendet. Ein CNN hat zwei Hauptbestandteile, eine oder mehrere Faltungsschichten und für jede dieser Schichten eine Pooling Operation. Dabei wird bei der Faltung mit einer Filtermatrix über

[1] `http://www.statisticbrain.com/twitter-statistics/`

das Bild, welches in Matrixform gegeben ist, iteriert und somit die Merkmale dieses Bildes herausgefiltert. Auf das Ergebnis der Faltung wird meist ein Pooling durchgeführt, um die Dimension der Merkmale stetig zu verkleinern, damit diese schließlich an eine lineare voll verbundene Schicht weitergeleitet werden können. Wir wollen dieses Verfahren nun so anpassen, dass auch unsere eindimensionalen Texteingaben, mithilfe einer 1D Faltung, für eine Merkmalsextraktion geeignet sind und somit eine gute Tweet Repräsentation gelernt werden kann. Um die Ergebnisse zu verbessern, benutzen wir die Optimierungsverfahren, die wir beim Multi-Layer Perceptron getestet haben.

Als letztes Verfahren der klassischen Klassifikation, betrachten wir eine spezielle Form des Recurrent Neural Networks, nämlich ein bi-direktionales Long Short-Term Memory. Im Allgemeinen wird diese Architektur für alle Arten von Anwendungen verwendet, die eine sequentielle Eingabe haben, oder zeitliche Abhängigkeiten in den Daten aufweisen. Bei einem NLP-Problem kann dadurch erkannt werden, ob es sinnvoll ist, dass ein bestimmtes Wort auf ein anderes in einem Satz folgt. Dieses Verfahren ist einfach auf unser Problem zu übertragen, indem wir die einzelnen Wörter des Tweets als Wortsequenz darstellen. Wir vergleichen die Ergebnisse mit denen des CNN und finden so die beste Möglichkeit, Hashtags für Tweets mit Neuronalen Netzen vorherzusagen.

Eine neue Herangehensweise um Hashtags vorherzusagen, bietet das hier vorgestellte Tweet2Seq Modell, das Hashtags mithilfe eines Sequenz Decoders Zeichenweise erzeugt. Auch wenn durch die Steigerung der Komplexität die Ergebnisse für die genaue Übereinstimmung der Hypothese mit der Referenz nicht verbessert werden konnten, so heben wir doch die Beschränkung des Hypothesenraums auf. Dadurch ermöglichen wir es unserem Modell eigene Hashtags zu kreieren oder bereits bekannte zu kombinieren. Somit soll auch neuen, bisher noch nicht gesehenen Tweets ein passender Hashtag, der dessen Thema erfasst, zugeordnet werden können.

2. Verwandte Arbeiten

Hashtag Vorhersage ist über die Jahre ein vieldiskutiertes Thema geworden und es gibt viele verschiedene Herangehensweisen, um dieses Problem zu lösen.

2.1. Ähnlichkeitsvergleiche

Viele Methoden basieren auf der Theorie, dass ähnlichen Tweets auch ähnliche Hashtags zugeordnet werden können. Es gibt jedoch verschiedene Dimensionen um die Ähnlichkeit eines Tweets zu bestimmen z.B. Länge des Tweets, Inhalt oder Autor.

Eine grundlegende Methode hierfür ist das Term Frequency - Inverse Document Frequency (TF-IDF) Maß, um die Ähnlichkeit von Dokumenten anhand der darin enthalten Wörter zu bestimmen [ZGS11]. Dabei beschreibt die Term Frequency wie oft ein Wort in allen Dokumenten auftaucht und die Inverse Document Frequency das Verhältnis von der gesamten Anzahl an Dokumenten zu der Anzahl Dokumente, in denen das Wort vorkommt. Dadurch wird die Gewichtung von Füllwörtern, die in jedem Dokument vorkommen, reduziert und für Wörter die nur selten in Dokumenten vorkommen erhöht. Beispielsweise betrachten wir die Eingabe *"Das ist unfassbar"*, wobei die Wörter *das* und *ist* eine höhere TF haben als *unfassbar*. Jedoch ist letzteres wichtiger, um den tatsächlichen Inhalt zu erfassen. Diese erhöhte Relevanz wird mithilfe der IDF realisiert, indem man die TF mit der IDF multipliziert und somit die TF-IDF enthält. Am Ende werden die TF-IDFs aller Wörter einer Eingabe aufaddiert und man erhält somit einen Bezugswert. Um diesem einen Kontext zu geben, berechnet man zunächst für jeden Tweet aus dem Datensatz D die TF-IDF über den gesamten Datensatz, und kann dann für eine neue Eingabe den ähnlichsten Tweet finden, indem man den minimalen Wert, beispielsweise durch $|tf\_idf(input) - tf\_idf(tweet)|$ für alle $tweet \in D$, bestimmt. Die Hashtags der ähnlichsten Tweets sind dann die Hashtag-Kandidaten, die jeweils noch einen Rang zugeordnet bekommen, um die geeignetsten Hashtags zu finden. In der Arbeit werden drei verschiedene Rangsysteme verglichen: Die Popularität bezüglich aller Hashtags, die Popularität der Hashtags innerhalb der Hashtag-Kandidaten und der Ähnlichkeitsrang, der einfach nach den berechneten TF-IDF Werten sortiert.

Eine abgeänderte Version von TF-IDF wird benutzt, um kollaboratives Filtern umzusetzen. Das heißt, dass nicht nur die Ähnlichkeit von Tweets, sondern auch Nutzerpräferenzen ausgewertet werden [Kyw+12]. Dafür wird für jeden Benutzer auf Basis der TF-IDF ein Gewicht für jeden Hashtag bestimmt, der die Verwendungshäufigkeit des Hashtags von diesem Benutzer widerspiegelt. Das Prinzip basiert darauf, dass ähnliche Nutzer die gleichen Hashtags benutzen. Nun kann die Cosinus-Ähnlichkeit zwischen Benutzern anhand der Gewichte berechnet werden und somit jedem Benutzer eine Liste der x-ähnlichsten Benutzer zugeordnet werden. Als nächstes wird ähnlich wie oben auch hier, die TF-IDF für

einen Tweet bestimmt und dann anhand dieser, die ähnlichsten Tweets gefunden. Aus den Hashtags der ähnlichen Tweets und den zuletzt verwendeten Hashtags der ähnlichen Benutzer wird dann eine Hashtag-Kandidaten-Liste erstellt, und diese anhand der Häufigkeit des Auftretens der einzelnen Kandidaten (Allgemeine Popularität s.o.) klassifiziert.

Andere glauben, dass das TF-IDF Maß ineffektiv für Tweets ist, da diese auf eine kurze Zeichenzahl beschränkt sind [DN15]. Außerdem ist TF-IDF für eine Echtzeitanwendung mit einem unendlichen Eingabestream ungeeignet, da der Vergleichsdatensatz fest ist und sich nicht an das dynamische Twitter, mit ständig wechselnden Hashtags, anpasst. Für diese Zwecke wird deswegen eine Klassifikation mit dem naiven Bayes und K-Nearest-Neighbours Algorithmus vorgeschlagen, wobei die Modelle anstatt auf einem von Anfang an festgelegten, auf einem FIFO-Datensatz trainiert werden und der letzte Tweet aus dem Modell entfernt wird, wenn ein neuer Tweet hinzukommt. Somit ist das System immer auf dem neusten Stand und kann sich dynamisch an die neuen Tweets und Hashtags anpassen.

Eine andere Herangehensweise, ist es die euklidische Distanz von Tweets zu bestimmen, um diese anhand der Distanz in Cluster zu unterteilen und somit zu klassifizieren [LWZ11].

2.2. Themen Modelle

Eine weitere Möglichkeit besteht darin, die Tweets mithilfe von *Latent Dirichlet Allocation* (LDA) einem bestimmten Thema zuzuordnen und dann anhand des Themas Hashtags vorzuschlagen [God+13]. Hierfür wird jedem Tweet tw eine multinomiale Verteilung θ_{tw}, über alle Themen T zugeordnet. Daraus können wir für jedes Wort im Tweet die Wahrscheinlichkeit, dass es zu einem Thema $T_x, x = 1 \ldots |T|$ gehört, bestimmen. Weiterhin bestimmen wir für jedes Thema $t \in T$ eine weitere multinomiale Verteilung φ_t, über alle Wörter W und können somit für jedes Thema, die Wahrscheinlichkeit, dass ein Wort $w \in W$ dieses repräsentiert, bestimmen. Die Parameter der Verteilungen werden mit dem Gibbs-Sampling Algorithmus gelernt, sodass die Tweets korrekt repräsentiert werden. Wenn wir nun für ein Tweet tw Hashtags vorschlagen, berechnen wir zuerst mit dem Gibbs-Sampling Algorithmus die Verteilungen θ_{tw} und $\varphi_{\theta_{tw}}$. Aus θ_{tw} kann das Thema des Tweets abgeleitet werden. Aus $\varphi_{\theta_{tw}}$ kann nun mit dem eben bestimmten Thema die Wörterverteilung bestimmt werden. Daraus können wir dann die Wörter ablesen, die das Thema am besten widerspiegeln und diese als Hashtags verwenden.

2.3. Neuronale Netze

Weiterhin bieten Neuronale Netze, die Hauptthema dieser Arbeit sind, eine gute Möglichkeit Hashtags vorherzusagen.

Tweet2Vec erstellt mit Hilfe einer bi-dirketionalen Gated Recurrent Unit eine Vektor-Repräsentation eines Tweets auf Zeichenbasis und nutzt diese, um erfolgreich Hashtags vorzuschlagen [Dhi+16]. Dazu wird zunächst für jedes einzelne Zeichen, ein One-Hot-Vektor der Länge $|C|$, wobei C die Menge aller Zeichen ist, angelegt. Dieser wird dann mit einer $|C| \times n$ Matrix, die als Lookup-Tabelle bezeichnet werden kann, multipliziert und somit auf einen Zeichen-Raum projeziert. Dabei ist n ein Hyperparameter der optimiert

werden kann und die Größe des Embeddings ändert. Diese eingebetteten Zeichen-Vektoren werden dann als Eingabe, für die beiden baugleichen Gated Recurrent Units verwendet, wobei die Reihenfolge der Zeichen-Vektoren für eine Einheit vertauscht wird. Es existiert also eine Vorwärts und eine Rückwärts Einheit. Die Ausgabe dieser zwei Einheiten wird addiert und bildet somit das finale Tweet Embedding. Dieses wird dann in eine Softmax-Schicht gegeben, deren Ausgabe die Größe H hat, wobei H die Anzahl aller möglichen Hashtags ist. Als Kostenfunktion wurde die kategorische Kreuz-Entropie verwendet und als Lernverfahren wurde der Mini-Batch Gradienten Abstieg mit Nesterov's Momentum benutzt. Diese Architektur erzielte in allen Einstellungen bessere Ergebnisse, als die gleiche Architektur auf Wortbasis, unabhängig von der Wörterbuchgröße.

Das Herausfiltern von Merkmalen, mit einem Convolutional Neural Network, erzielt ebenfalls gute Ergebnisse für die Hashtag Vorhersage [WCA14]. Das Netz hat wie oben eine Embedding Layer, jedoch auf Wortbasis, mit der Wort-Vektoren der Länge d erzeugt werden. Es folgt eine Faltungsschicht, für die k Embeddings aus der Schicht davor zu einem Vektor der Länge $k \cdot d$ konkateniert, als Eingabe dienen. Man nennt k auch die Fenstergröße. Dieses Fenster wird dann in 1er Schritten über den Tweet iteriert. Für jeden Iterationsschritt, wird der konkatenierte Vektor dann mit einer Filtermatrix der Größe $k \cdot d \times n$ multipliziert, um n Merkmale aus einem Fenster herauszufiltern. Aus allen Iterationsschritten wird nun der maximale Vektor bestimmt und es ergibt sich somit ein einheitlicher Ausgabevektor der Länge n, der an eine vollständig verbundene Schicht und anschließend eine lineare Schicht weitergeleitet wird, um die Hashtags zu bestimmen.

Eine andere Convolutional Neural Network Methode fixiert sich auf den Einfluss der Wortreihenfolge und vergleicht eine sequentielle und eine Bag-of-Word Repräsentation der Tweets, wobei die erstere besser für eine Sentiment Analyse und letztere besser für Hashtag Vorhersage geeignet ist [JZ15]. Der Unterschied zu [WCA14] besteht darin, dass es keine Embedding Layer bzw. Lookup-Tabelle gibt und die Faltung direkt auf der One-Hot-Vektor Repräsentation der Tweets durchgeführt wird. Auch hier wird eine Fenstergröße k festgelegt und dann k One-Hot-Vektoren konkateniert um einen Regionen-Vektor zu bestimmen. Durch das Schieben des Fensters über den Tweet, bekommen wir unsere Eingabematrix, welche aus den Regionen-Vektoren besteht und wenden darauf die klassische Faltung an, die auch bei Standard Convolutional Neural Networks, die Bilder als Eingabe haben, zum Einsatz kommt. Dadurch, dass die One-Hot-Vektoren davor nicht eingebettet werden, haben wir abhängig von der Vokabulargröße sehr große Matrizen mit vielen Nullen, die ohne Optimierung nur sehr ineffizient berechnet werden können. Doch mithilfe von Optimierungen zum berechnen dünn besetzter Matrizen und parallelen Faltungsoperationen lässt sich dieses Problem lösen.

Rekurrente Netze werden auch von [Li+16] verwendet, um Hashtags vorherzusagen. Hierzu wird der Attention Mechanismus, mit einer thematischen Verteilung durch LDA kombiniert und in einer Long Short-Term Memory (LSTM) Architektur untergebracht. Dazu wird jede LSTM Ausgabe von h_0 bis h_n mit einem Attention Gewicht a_t multipliziert. Dieses Gewicht wird aus der thematischen Verteilung und der Ausgabe des LSTMs berechnet. Die gewichteten Ausgaben werden dann an eine lineare Schicht mit anschließender Softmax Schicht weitergegeben, um einen Hashtag-Kandidat vorzuschlagen.

3. Hintergrundwissen

In diesem Kapitel behandeln wir die grundlegenden Konzepte, die später in dieser Arbeit verwendet werden. Im ersten Abschnitt beschäftigen wir uns mit künstlichen neuronalen Netzen und dem Backpropagation Algorithmus. Als Erweiterung des einfachen neuronalen Netzes, wird im zweiten Abschnitt das Multi-Layer Perceptron vorgestellt. In Abschnitt drei wird das Prinzip des Convolutional Neural Network behandelt. Dann beschreiben wir Recurrent Neural Networks und das Konzept des Long Short-Term Memory, welches eine spezielle Form des ersteren darstellt. Als letztes werden noch grundlegende Optimierungsverfahren vorgestellt, welche die Ergebnisse verbessern und für jede Netzarchitektur anwendbar sind.

3.1. Künstliche Neuronale Netze

Künstliche Neuronale Netze lernen anhand von Beispielen verschiedene Parameter, sodass sie eine gewünschte Zielfunktion möglichst genau approximieren. Dazu werden künstliche Neuronen miteinander vernetzt und modellieren somit grob die biologische Funktionsweise unseres Gehirns. Das besondere an dieser Herangehensweise ist, dass der Lösungsweg nicht statisch festgelegt, sondern durch den Backpropagation-Algorithmus gelernt wird und der Rechner somit Probleme wie "von selbst" lösen kann.

3.1.1. Perzeptron

Die einfachste Architektur eines solchen Netzes, stellt das Perzeptron wie in Abbildung 3.1 dar. Es besteht aus einem Eingabe Vektor $\hat{X} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ und einem Gewichtsvektor $W = (w_0, \ldots, w_n) \in \mathbb{R}^{n+1}$. Mithilfe der Matrixmultiplikation, kann der Zwischenwert *net* ganz einfach durch XW^T bestimmt werden, wobei $X = (x_0, \hat{X}) \in \mathbb{R}^{n+1}$ und $x_0 = 1$ gilt. Somit ist w_0 von der Eingabe $\hat{X}$ unabhängig und wird als Bias bezeichnet. Der Netzwert $net = \sum_{i=0}^{n} w_i x_i$ wird dann an die Aktivierungsfunktion σ weitergeleitet. Wir wollen hier die vier häufigsten Aktivierungsfunktionen darstellen, um ein Bild von den Eigenschaften einer solchen zu bekommen.

Sign

$$\sigma(net) = \begin{cases} 1 & net > 0 \\ 0 & net = 0 \\ -1 & net < 0 \end{cases}$$

Treshold t

$$\sigma(net) = \begin{cases} 1 & net > t \\ -1 & \text{sonst} \end{cases}$$

Sigmoid

$\sigma(net) = \frac{1}{1+e^{-net}}$

Tangens Hyperbolicus

$\sigma(net) = \tanh(net)$

Es sind also Funktionen, die einen Großteil aller Werte aus dem Definitionsbereich auf 1 oder -1 im Wertebereich abbilden bzw. die schnell gegen 1 und -1 konvergieren, wie z.B. die Sigmoid- und die Tanh-Funktion. Hier haben wir wieder die Parallele zur Biologie, da das Neuron für eine Eingabe (Reiz) entweder 1 (aktiviert) oder -1 (keine Reaktion) ausgeben kann. Sogleich beschränkt diese binäre Ausgabe aber auch die Komplexität der Probleme, die ein einzelnes Perzeptron lösen kann. Denn schon das XOR-Problem ist für ein einzelnes Perzeptron, dass nur lineare Probleme bewältigen kann, unlösbar. Deswegen erweitern wir unser Perzeptron zu einem Netz aus mehreren Schichten, dem Multi-Layer Perceptron, um beliebig komplexe Probleme lösen zu können. Zuvor schauen wir uns aber noch für das einfache Perzeptron an, wie dieses überhaupt lernt und wie die Gewichte beeinflusst werden.

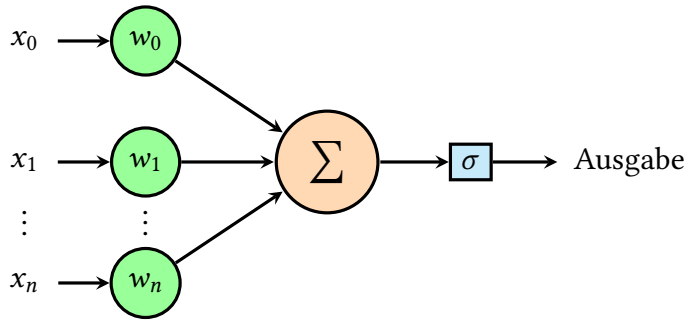

Abbildung 3.1.: Perzeptron mit Eingabe Vektor $x_1..x_n$, Gewichten $w_0..w_n$, Bias x_0 und einer Aktivierungsfunktion σ

3.1.2. Backpropagation

Um zu lernen, brauchen wir Beispiele und definieren einen Trainingsdatensatz T, wobei jedes Element $e \in T$ aus einer Eingabe *in* und einem Sollwert s besteht. Die Ausgabe des Netzes soll nun für jede Eingabe *in* eine Ausgabe *out* produzieren, die den zur Eingabe gehörigen Sollwert s möglichst genau approximiert. Um die Güte des momentanen Modells zu bewerten, benötigen wir eine Fehlerfunktion, auch Kostenfunktion genannt, welche die Abweichung von Ist- und Sollwert bestimmt. Eine weit verbreitete Fehlerfunktion ist die quadratische

$$E = \frac{1}{|T|} \sum_{e \in T} (out_e - s_e)^2, \tag{3.1}$$

die den Gesamtfehler über den Trainingsdatensatz T berechnet. Dieser Fehler soll nun durch Anpassung der Gewichte minimiert werden. Hierzu verwenden wir den Gradientenabstieg. Bei diesem Verfahren bestimmen wir die Steigung an der aktuellen Position und können uns anhand dieser, in Richtung Minimum orientieren. Nach einem Schritt in Richtung des Minimums, wird die Steigung erneut bestimmt und die Richtung gegebenenfalls korrigiert. Dabei spielt die Schrittweite eine entscheidende Rolle. Wird sie zu klein gewählt, dauert der Prozess sehr lange und es besteht die Gefahr, in lokalen Minima stecken zu bleiben. Wird sie zu groß gewählt, kann das Minimum übersprungen werden. In unserem Fall implementieren wir den Gradientenabstieg, indem wir für jedes Gewicht w_i dessen

Auswirkung auf den gesamten Fehler $\Delta w_i = \frac{\partial E}{\partial w_i}$ bestimmen. Damit können wir dann unser Gewicht w_i aktualisieren, indem wir ihm den neuen Wert $w_{i_{neu}} = w_i + (-\alpha \Delta w_i)$ zuweisen. Dabei ist α die Schrittweite, in unserem Kontext auch *Lernrate* genannt, Δw_i die Richtung in die wir hinabsteigen und w_i unser Ausgangspunkt. Als nächstes wollen wir noch zeigen, wie wir Δw_i konkret berechnen können. Dazu betrachten wir noch einmal den Aufbau in Abb. 3.1 mit Aktivierungsfunktion σ, die aufsummierte Netzausgabe net, sowie die Eingabe- und Gewichtsvektoren X und W. Dann kann Δw_i mithilfe der Kettenregel wie folgt berechnet werden:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial net} \cdot \frac{\partial net}{\partial w_i} \tag{3.2}$$

Es ist also wichtig, dass die Aktivierungsfunktion differenzierbar ist und daher werden in der Praxis häufig die Sigmoid- und die Tanh-Funktion gewählt. Nehmen wir nun an, wir schalten vor unser Perzeptron p noch $n-1$ weitere Perzeptrons $p_1 \dots p_{n-1}$. Dann bildet die Ausgabe der Perzeptrons $p_1 \dots p_{n-1}$ die Eingabe $\hat{X}$ für unser Perzeptron p. Wenn wir nun den Fehler weiter nach hinten propagieren und die Gewichte der Perzeptronen $p_1 \dots p_{n-1}$ berechnen wollen, werden auch die Ableitungen nach den x_i von p interessant, da diese die Ausgaben der $p_1 \dots p_{n-1}$ bilden.

$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial net} \cdot \frac{\partial net}{\partial x_i} \tag{3.3}$$

Nun können wir mit (3.2) die Gewichte von $p_1 \dots p_{n-1}$ aktualisieren indem wir jeweils $\frac{\partial E}{\partial \sigma}$ mit (3.3) ersetzen.

Als letztes wollen wir (3.3) noch etwas verallgemeinern indem wir annehmen, das x_i nicht nur als Eingabe für p sondern noch für weitere k Perzeptrons dient.

$$\frac{\partial E}{\partial x_i} = \sum_{j=0}^{k} \frac{\partial E}{\partial net_j} \cdot \frac{\partial net_j}{\partial x_i} = \sum_{j=0}^{k} \frac{\partial E}{\partial net_j} \cdot w_{ij}, \tag{3.4}$$

wobei die letzte Gleichheit schnell klar wird, wenn wir noch einmal die Definition des Netzwertes $net_j = \sum_{i=0}^{n} w_{ij} x_{ij}$ in Betracht ziehen. Der Backpropagation-Algorithmus besteht also insgesamt aus zwei Schritten. Einem Vorwärtschritt, bei dem der Eingabe Vektor ins Netz gegeben und der Fehler (3.1) berechnet wird. Und einem Rückwärtschritt, bei dem der Fehler im Netz zurück propagiert wird und die Gewichte aktualisiert werden [Wai16].

3.1.3. Stochastischer Gradienten Abstieg und ADAM

Eine gängige Methode um neuronale Netze zu trainieren, ist die Benutzung von Batches. Dazu werden immer eine vorher festgelegte Anzahl an Datenpunkten aus dem Trainingsdatensatz zu einem Batch zusammengefasst. Beim normalen Gradientenabstieg, wird für jeden Datenpunkt in einem Batch der Gradient bestimmt und dann der Durchschnittswert aller Gradienten, als Gewichtsanpassung Δw genutzt. Ein *Stochastischer Gradienten Abstieg* (SGD) beschleunigt diesen Prozess, indem für jeden Batch nur ein zufälliger Datenpunkt gewählt wird, für den der Gradient bestimmt wird und mit dem die Gewichtsaktualisierung ausgeführt wird. SGD ist aber nicht nur effizienter, sondern konvergiert schneller, bei

sich häufig wiederholenden Eingabedaten, kann aus lokalen Minima entkommen und ein besseres Minimum erreichen [Bot91]. Es gibt noch viele weitere Variationen des SGD, um noch bessere Ergebnisse zu erzielen. Eines dieser Verfahren, das wir in dieser Arbeit noch verwenden werden, ist ADAM [KB14]. ADAM kommt von 'adaptive moment estimation' und ist eine Kombination aus zwei anderen SGD-Optimierungsverfahren. Von *RMSProp* wird die Fähigkeit übernommen, auch mit nicht stationären Zielfunktionen umgehen zu können und von *AdaGrad* die Fähigkeit, auch bei dünn besetzten Gradienten gute Ergebnisse zu erzielen. Wie der Name schon sagt, schätzt ADAM zwei verschiedene Momente der Gradienten und nutzt eine adaptive Lernrate, um optimale Aktualisierungswerte zu bestimmen. Wobei das erste Moment den Durchschnitt der Gradienten und das zweite, die Varianz der Gradienten darstellt.

3.2. Multi-Layer Perceptron

Das *Multi-Layer Perceptron* (MLP) ist, wie schon in Abschnitt 3.1.2 angedeutet, eine Vernetzung von vielen Perzeptronen auf mehreren Schichten, sodass die Ausgabe eines Perzeptrons, die Eingabe eines oder mehrerer anderer ist. Dadurch ist man nicht mehr auf die Lösung von linearen Problemen beschränkt, sondern kann mit der richtigen Netzstruktur beliebig komplexe Probleme lösen. Das MLP ist ein Feed-Forward Netz. Das bedeutet, dass ein Neuron aus einer Schicht, nur mit der nächsten Schicht in Richtung Ausgabe verbunden werden kann, bzw. dass es keine Verbindungen innerhalb einer Schicht, oder zu einer vorherigen Schicht gibt. Wenn jedes Neuron aus einer Schicht i mit jedem Neuron aus der Schicht $i + 1$ verbunden ist, nennt man die Schicht i auch vollständig verbundene Schicht, wie in Abb. 3.2 zu sehen. Die Schichten zwischen Eingabe und Ausgabeschicht, werden *Hidden Layers* genannt. Es gibt aber auch noch andere nützliche Schichten, von denen wir hier noch die *Embedding Layer* und die *Softmax Layer* vorstellen wollen.

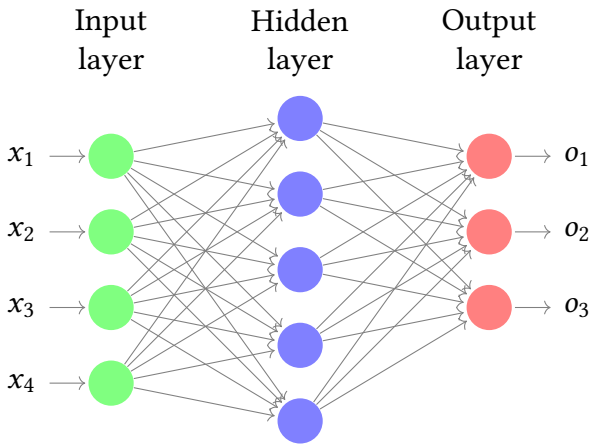

Abbildung 3.2.: MLP mit vollständig verbundenen Schichten, Eingabe Vektor x und Ausgabe Vektor o. Jeder Knoten stellt ein einzelnes Perzeptron dar.

Embedding Layer

Um die Funktionsweise der Embedding-Layer richtig beschreiben zu können, beschäftigen

wir uns zuerst mit der Bag-of-Words (BoW) Repräsentation eines Dokuments. Ein Wörterbuch ist eine beliebig große Menge an Wörtern, von denen möglichst viele im Dokument vorkommen. Diese Menge wird dann noch um ein *UNKNOWN* Token erweitert. Jetzt können wir jedem Wort aus dem Dokument, ein Element im Wörterbuch zuordnen. Der Index des zugehörigen Elements, wird dann in einer Liste L gespeichert. Für Wörter, die nicht im Wörterbuch enthalten sind, aber dennoch im Dokument vorkommen, wird der Index gespeichert, an dem das *UNKNOWN* Token steht. Die BoW Repräsentation des Dokumentes ist nun ein Vektor $v \in \{0,1\}^{|V|}$, mit dem Wörterbuch V und für die einzelnen Elemente v_i, die mit 0 initialisiert werden, gilt: $v_i = 1$ genau dann, wenn $i \in L$. Es ist zu beachten, dass das Dokument nicht vollständig wiederhergestellt werden kann, da sowohl die Wortreihenfolge, als auch das mehrfache Auftreten von gleichen Wörtern, bei dieser Darstellung nicht in Betracht gezogen werden. Außerdem können die *UNKNOWN* Tokens nicht rekonstruiert werden.

Die Embedding Layer verhält sich wie eine Lookup-Tabelle. Diese wird durch eine Gewichtsmatrix $W \in \mathbb{R}^{|V| \times dim}$ repräsentiert, wobei dim die Größe des Embeddings darstellt. Nun können wir einfach das Matrixprodukt $e = vW$ bilden, wobei v die BoW Repräsentation eines Dokumentes ist und erhalten ein Embedding $e \in \mathbb{R}^{dim}$, welches das Dokument repräsentiert. Veranschaulicht dargestellt, wird für jedes Wort des Dokuments das entsprechende Embedding herausgesucht und anschließend werden diese aufsummiert, um eine Vektorrepräsentation des Dokuments zu erhalten. Wir werden aber später auch noch eine Repräsentation der Tweets benötigen, die die zeitliche Reihenfolge beachtet. In diesen Fällen werden wir nicht die Embeddings der einzelnen Wörter aufsummieren, sondern den Tweet als eine $l \times dim$ Matrix darstellen, wobei l die Anzahl der Wörter des Tweets ist und jede Zeile dieser Matrix einem Wort entspricht [Col+11].

Softmax Layer

Diese Schicht wird häufig für Multi-Klassen Klassifikationsprobleme als Ausgabeschicht verwendet. Dabei entspricht die Größe des Ausgabevektors, der Anzahl der möglichen Klassen. Dieser Ausgabevektor O wird dann durch die Softmax Funktion so verändert, dass jeder Wert $o_x \in O$ in dem Bereich $(0,1)$ liegt und sich alle Werte zu 1 aufsummieren.

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{c=1}^{C} e^{x_c}} \quad \text{für } i = 1, \ldots, C \quad , \tag{3.5}$$

wobei C die Anzahl an Klassen ist. So erhalten wir eine Wahrscheinlichkeitsverteilung über alle möglichen Klassen für eine Eingabe, die sich gut mit der erwarteten Verteilung vergleichen lässt.

3.3. Convolutional Neural Network

Ein *Convolutional Neural Network* (CNN) ist eine Architektur, um Merkmale aus einer Eingabe herauszufiltern und anhand dieser zu klassifizieren. Dieses Prinzip wurde erstmals zur Spracherkennung, in einem *Time-Delay Neural Network* (TDNN) umgesetzt. Das TDNN ist in der Lage, bereits bekannte phonetische Merkmale, ohne diese implizit zu definieren, selbstständig zu lernen und dann anhand dieser Merkmale, Daten richtig zu klassifizieren

[Wai+89]. Die Weiterentwicklungen dieser Idee führten zu dem CNN, wie wir es heute kennen. Die grundlegenden Bestandteile eines solchen CNN, sind die Convolution Layer und die Pooling Layer. Von diesen zwei Schichten, werden oft mehrere in abwechselnder Reihenfolge hintereinander geschaltet, wobei eine Convolution Layer gewöhnlich den Anfang bildet. Die Ausgabe der letzten Pooling Layer wird dann in der Regel an eine oder mehrere voll verbundene Hidden Layers weitergeleitet und abschließend in eine Softmax-Layer gegeben, um eine Klassifikation für die ursprüngliche Eingabe zu errechnen. Um die Funktionsweise eines CNNs klar zu machen, stellen wir im folgenden die Convolution und Pooling Layer genauer vor.

3.3.1. Convolution Layer

Diese Schicht beruht, wie der Name schon sagt, auf dem Prinzip der mathematischen Faltung. Wir benutzen dazu eine Filtermatrix, die auch *Kernel* genannt wird. Dieser Kernel wird dann über die Eingabematrix iteriert und der jeweilige Ausschnitt der Eingabematrix, den der Kernel gerade überdeckt, wird mit dem Kernel elementweise multipliziert. Die Summe aller Elemente der Ergebnismatrix, ist ein Eintrag in der entstehenden *Feature-Map*. Abb. 3.3 veranschaulicht dieses Verfahren. Die Schrittweite, um die der Kernel in jedem Schritt verschoben wird, nennt man auch *stride*. Meistens wird dieser sehr klein gewählt, sodass sich Werte aus der Eingabematrix überlappen, bzw. mehrmals vom Kernel erfasst werden. Er kann aber auch der Größe des Kernels entsprechen, sodass jeder Wert der Eingabematrix nur einmal in Betracht gezogen wird.

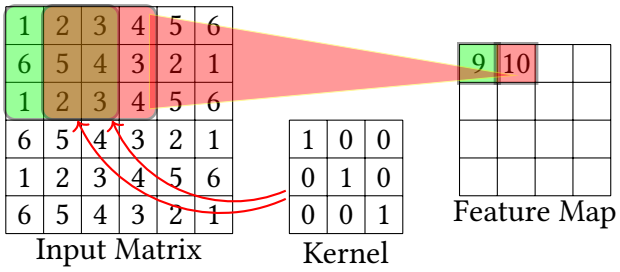

Abbildung 3.3.: Faltung einer 6×6 Matrix mit einem 3×3 Kernel und einer Schrittweite $stride = 1$. Es entsteht eine 4×4 Feature Map.

3.3.2. Pooling

Die *Pooling Layer* dient dazu, die Größe der Feature Maps zu reduzieren und somit die einzelnen Merkmale zu verallgemeinern. Dafür muss zunächst eine Pooling Größe festgelegt werden. Diese ist meist relativ klein, z.B. 2×2 oder 3×3, da bei zu großen Pooling Operationen, zu viel Information verloren geht. Auch diese Pooling Operation wird über die Feature Map iteriert, wobei hier der stride, im Gegensatz zur Convolution-Layer, meistens so groß ist wie die Größe des Pooling Operators. Die Pooling Operation fasst alle Werte die sie erfasst zu einem zusammen, und verkleinert die Feature Map somit um den Faktor

$n \cdot m$, bei Pooling Größe $n \times m$. Die meist verwendeten Pooling Operatoren sind Max- und Average-Pooling.

MaxPool Hierbei wird das Maximum aller Werte, die die Pooling Operation erfasst, ermittelt und ausgegeben.

AveragePool Hierbei wird der Durchschnitt aller Werte, die die Pooling Operatin erfasst, gebildet und ausgegeben.

3.3.3. Netzarchitektur

In einem CNN werden mehrere Convolution und Pooling Layer hintereinander gehängt, um immer allgemeinere Merkmale zu bilden, anhand derer die Eingabe korrekt klassifiziert werden kann. Eine beispielhafte Netzarchitektur ist in Abb. 3.4 abgebildet. Ein Kernel kann mehrere Kanäle (Channels) haben, was bedeutet, dass der Kernel nicht nur aus einer, sondern mehreren Filtermatrizen derselben Größe besteht, die über die Eingabematrix laufen. Somit können pro Schicht anstatt einem, mehrere Merkmale herausgefiltert werden. Die Anzahl der Kanäle kann für jede Convolution Layer verschieden sein. Ein anwendungsnahes Beispiel für Kanäle, ist die RGB Repräsentation eines Bildes. Dabei wird für jede der drei Farben eine eigene Matrix bzw. Kanal angelegt, um die Intensität der jeweiligen Farbe für jeden Pixel festzulegen. Dem Leser wird aufgefallen sein, dass

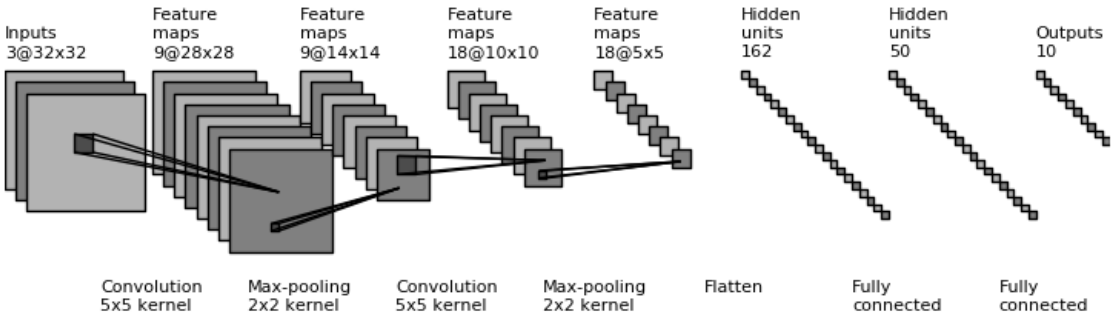

Abbildung 3.4.: Ein CNN mit einem Bild im RGB Format als Eingabe. Während der Kernel der ersten Convolution Layer 3 Kanäle besitzt, hat der Kernel der zweiten Convolution Layer nur 2 Kanäle.

diese Architektur sehr verschieden von einem MLP ist und wir bisher keine Gewichte bzw. Parameter definiert haben die gelernt werden können. Bei CNNs sind diese Gewichte die Kernels der Convolution-Layers. Die Werte dieser Matrizen werden auch hier mit dem Backpropagation Algorithmus angepasst. Das Netz lernt also wie eine Filtermatrix aussehen muss, um aussagekräftige Merkmale aus der Eingabe herauszufiltern. Die voll verbundenen Hidden Layers und die Softmax-Layer dienen dann dazu, Merkmalen Klassen zuzuordnen, um Eingaben anhand ihrer Merkmalsrepräsentation richtig zu klassifizieren [ZF13].

3.4. Recurrent Neural Network

Das *Long Short-Term Memory* (LSTM) ist eine Erweiterung des *Recurrent Neural Network* (RNN) um zeitliche Abhängigkeiten in der Eingabe zu lernen. Dabei ist das besondere bei einem LSTM, dass nicht nur Kurzzeit sondern auch Langzeit Abhängigkeiten gelernt werden können. Bevor wir dies genauer betrachten, wollen wir erst einmal das grundlegende Prinzip des RNN vorstellen und dann darauf aufbauend die Besonderheiten des LSTM.

3.4.1. RNN

Das Recurrent Neural Network basiert auf der Idee ein Gedächtnis anzulegen, das sich an die bisher gesehene Eingabe erinnert, um aus dieser Erinnerung eine sinnvolle Fortsetzung abzuleiten, oder um nach dem Sehen der kompletten Eingabe, diese anhand des Gedächtnisses zu bewerten. Für die Umsetzung eines solchen rekurrenten Netzes, benötigen wir demnach zuerst eine sequentielle Eingabe. Theoretisch können alle Eingaben zu einer Sequenz umgewandelt werden. Jedoch macht es mehr Sinn, diese Netzarchitektur für Anwendungen zu benutzen, deren Eingabe tatsächlich sequentiell ist, wie z.B. Sprache. Hier spielen die vorhergegangen Wörter eine wichtige Rolle, um Möglichkeiten für ein Folgewort vorzuschlagen, oder um sinnvolle Sätze zu bilden. Im Gegensatz zu den bisherigen Architekturen, bei denen die Eingabe als Ganzes in das Netz geleitet wurde, wird bei einem RNN die Eingabe Sequenz nach und nach, anhand der zeitlichen Reihenfolge, eingegeben. All dies, kann mit nur einer Einheit, welche in Abb. 3.5 veranschaulicht wird, umgesetzt werden. Ein RNN kann damit einfach durch einen Gedächtniszustand s_t und

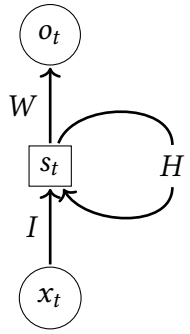

Abbildung 3.5.: Einfache RNN Einheit

die Gewichte I, H, W definiert werden. Sei nun X die Eingabesequenz und x_t die Eingabe zum Zeitpunkt t. Dann können wir die Ausgabe o_t zum Zeitpunkt t wie folgt berechnen.

$$s_t = \sigma(Ix_t + Hs_{t-1}) \tag{3.6}$$

$$o_t = \varphi(Ws_t), \tag{3.7}$$

mit einem Initialwert s_0, einer gut differenzierbaren Aktivierungsfunktion σ z.B. Tanh oder Sigmoid und einer Ausgabefunktion φ, z.B. die Indentität oder Softmax [Elm90]. Um die Parameter I, H, W und optional auch s_0 zu lernen verwenden wir wieder den Backpropagation Algorithmus. Dazu wird das RNN, wie in Abb. 3.6, ausgerollt. Das bedeutet, dass das

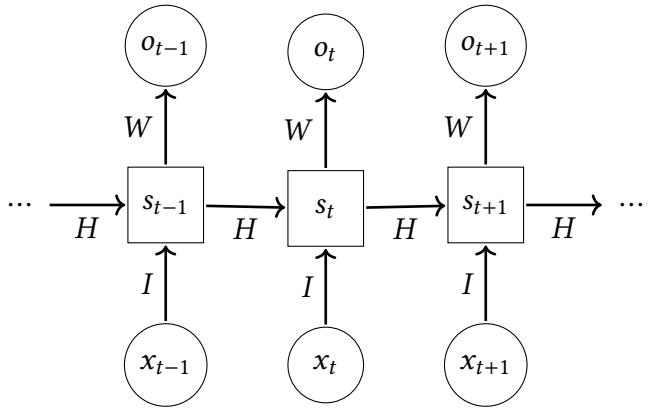

Abbildung 3.6.: Ausgerolltes RNN, wobei jeder Zeitschritt t einer Schicht entspricht.

RNN nicht auf sich selbst verweisend dargestellt wird, sondern als eine verbundene Reihe an RNN-Einheiten, die jeweils das entsprechende Sequenz-Element als Eingabe haben. Somit erhalten wir wieder unser Schichten Modell, durch das wir den Fehler durchpropagieren können. Dieses Verfahren nennt man auch *backpropagation through time* (BPTT) [WZ89].

3.4.2. LSTM

Jedoch führt BPTT bei einem einfachen RNN oft zu explodierenden oder verschwindenden Gradienten. Ersteres bedeutet, dass durch immer größer werdende Gradienten das Minimum übersprungen wird und es somit zu oszillierenden Gewichten kommt, was letztendlich zur Divergenz des Gradientenabstiegs führt. Bei verschwindenden Gradienten werden diese bei der Propagierung des Fehlers durch das Netz so klein, dass die Gewichte nur noch sehr langsam angepasst bzw. gar nicht mehr verändert werden und auch somit keine Konvergenz zustande kommt. Das LSTM löst diese Probleme mit einem konstanten Fehlerfluss und ermöglicht somit das Lernen von Langzeitabhängigkeiten. Dies wird mithilfe einer sogenannte *Memory Cell*, wie in Abb. 3.7, umgesetzt. Sie besteht aus einem Gedächtnis-Zustand mem_t, einem Input-Gate i_t, einem Output-Gate o_t und einem Forget-Gate f_t. Der Gedächtnis-Zustand ist eine rekurrent mit sich selbst verbundene Einheit, die für den konstanten Fehlerfluss verantwortlich ist. Auf diese selbst rekurrente Verbindung, wird das Forget-Gate aufgesetzt welches bestimmt, ob ein Gedächtnis-Zustand erinnert oder vergessen wird. Das Input-Gate ist dafür zuständig, den Gedächtnis-Zustand von unnötigen Eingaben und Rauschen in den Daten zu schützen. Analog dazu schützt das Output-Gate, die darauf folgenden Memory Cells vor Störungen durch unwichtige Gedächtnis-Zustände. Sei x_t die Eingabe und h_t die Ausgabe von einer Memory Cell zum Zeitpunkt t, mit Gewichten $W_i, W_f, W_c, W_o, H_i, H_f, H_c, H_o$ und Bias b_i, b_f, b_c, b_o, dann

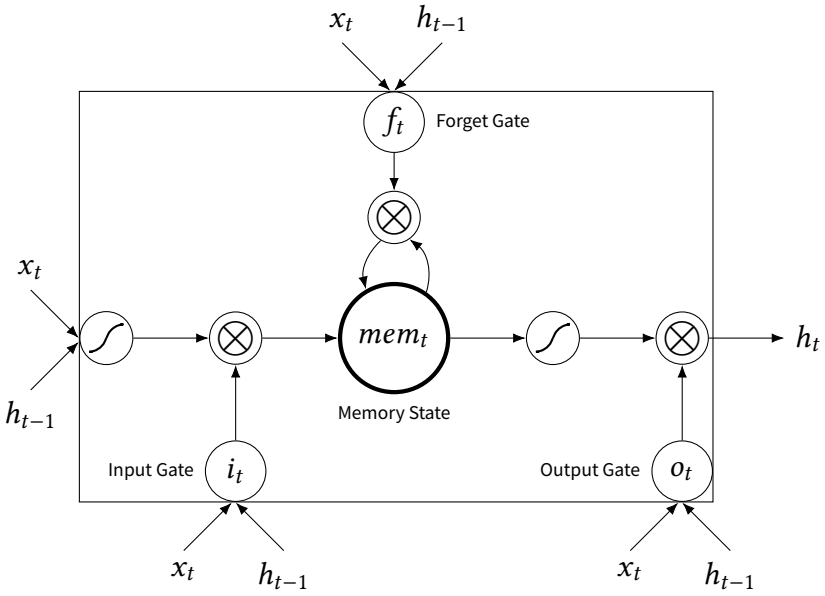

Abbildung 3.7.: Aufbau einer Memory Cell

können die Zustände dieser Memory-Cell wie folgt berechnet werden.

$$i_t = \sigma(W_i x_t + H_i h_{t-1} + b_i) \tag{3.8}$$
$$f_t = \sigma(W_f x_t + H_f h_{t-1} + b_f) \tag{3.9}$$
$$mem_t = f_t \otimes mem_{t-1} + i_t \otimes tanh(W_c x_t + H_c h_{t-1} + b_c) \tag{3.10}$$
$$o_t = \sigma(W_o x_t + H_o h_{t-1} + b_o) \tag{3.11}$$
$$h_t = o_t \otimes tanh(mem_t), \tag{3.12}$$

wobei $\otimes$ die Elementweise-Multiplikation darstellt, σ die Sigmoid Funktion ist und $h_0 = 0$ gilt [HS97; GSC00].

3.5. Trainings Optimierungen

Eines der häufigsten Probleme bei Neuronalen Netzen, ist das sogenannte *Overfitting*. Hierbei handelt es sich um das Auswendiglernen von Trainingsdaten, sodass diese fast fehlerfrei klassifiziert werden können, aber ungesehene Daten einen sehr großen Fehler aufweisen. Eine Möglichkeit dem entgegen zu wirken, ist die Wahl eines angemessen großen Trainingsdatensatzes. Weiterhin sollte darauf geachtet werden, dass möglichst wenig bis gar keine Duplikate in dem Trainingsdatensatz vorhanden sind. Aber selbst diese zwei Maßnahmen dämmen das Problem nicht vollständig ein. Deswegen lernen wir noch drei weitere Methoden kennen, um unseren Erfolg beim Klassifizieren von ungesehenen Daten zu verbessern.

Mischen der Trainingsdaten

Eine recht simple aber doch effiziente Methode ist das Durchmischen der Trainingsdaten. So wird verhindert, dass einem Datenpunkt jedes mal die gleichen Datenpunkte und deren Aktualisierungen vorausgehen. Die Datenpunkte sind dadurch also voneinander unabhängig.

Dropout

Beim Dropout werden in jeder Trainingsiteration ein bestimmter Prozentsatz an Neuronen aus der Berechnung ausgeschlossen. Dadurch wird der Einfluss von einzelnen Neuronen reguliert und das Netz lernt auch bei fehlenden Informationen noch gute Ergebnisse zu erzielen.

Regularisierung

Unter Regularisierung versteht man die Addition eines Terms auf die Kostenfunktion. Der Term dient dazu, große Gewichte zu bestrafen und diese nur bei wesentlichen Verbesserungen gegenüber kleinen Gewichten zu bevorzugen. Eine Möglichkeit für einen Regularisierungsterm, der auch für diese Arbeit von Relevanz ist, ist die L2-Regularisierung R.

$$R(\theta) = \sum_{i=0}^{|\theta|} \theta_i^2 \tag{3.13}$$

Dabei stellt θ die Parameter des Netzes da. Jedoch nur die Gewichte, die keine Bias sind. Dieser Term wird dann wie folgt in die Kostenfunktion eingearbeitet.

$$E = cost + \lambda R(\theta), \tag{3.14}$$

wobei $\lambda \in \mathbb{R}$ ein Parameter zur Anpassung der Regulierung ist. Wird λ klein gewählt, wird die Optimierung der Kostenfunktion priorisiert. Wählt man jedoch ein großes λ, so wird die Priorität von der Kostenfunktion, auf die Beschränkung der Gewichtsgrößen gelegt. Diese Methode trägt dazu bei, das Rauschen in den Trainingsdaten besser zu verarbeiten und somit eine allgemeinere Zielfunktion zu approximieren, anstatt die Trainingsdaten auswendig zu lernen.

4. Hashtag Vorhersage - Netzwerkarchitekturen

In diesem Kapitel werden die konkreten neuronale Netze Modelle, die zur Vorhersage verwendet werden, vorgestellt. Zu Beginn werden wir erst einmal unser Problem -Hashtag Vorhersage- definieren und dann Lösungsansätze vorschlagen. Als ersten Ansatz betrachten wir eine MLP Architektur, die als Grundlage dient, um verschiedene Optimierungsmethoden zu testen und als Performanz-Vergleich für komplexere Modelle dienen soll. Ein etwas von dem Standard abweichendes Modell des CNNs wird in Abschnitt 4.3 vorgestellt, um die Tweets anhand ihrer Merkmale einem geeigneten Hashtag zuzuordnen. In Abschnitt 4.4 stellen wir ein bi-direktionales LSTM vor, das die zeitliche Abhängigkeit der einzelnen Wörter in einem Tweet ausnutzt. Im letzten Abschnitt stellen wir eine abgeänderte Version des LSTM Modells vor, das nicht mehr der klassischen Klassifikation entspricht, sondern einen Sequence to Sequence Ansatz implementiert.

4.1. Hashtag Vorhersage

Anhand des Titels dieser Arbeit ist das Problem, welches wir lösen wollen, einfach zu formulieren: Finde für einen Tweet einen Hashtag, der den Inhalt bzw. das Thema des Tweets möglichst gut erfasst. Dieses Problem werden wir nun so formulieren, dass die im folgenden vorgestellten Methoden darauf anwendbar sind. Dafür definieren wir zunächst einen Tweet als einen Text t und die Menge der darin enthaltenen Hashtags als Label L. Sofern der Benutzer seinen Tweet mit Hashtags versehen hat, können wir einen Tweet als Datenpunkt (t, L) darstellen. Viele dieser Datenpunkte zusammengefasst ergeben somit einen Datensatz D. Ziel ist es nun, anhand dieses Datensatzes Tweet Repräsentationen zu lernen und somit für Tweets, die keine Hashtags enthalten, für die also $L = \emptyset$ gilt, eine Menge an Hashtags vorzuschlagen. Anders ausgedrückt, wir wollen eine Texteingabe mindestens einer Klasse zuordnen. Da ein Tweet nicht nur mit einem Hashtag versehen werden kann und es auch öfters vorkommt, dass mehrere Tags für einen Tweet verwendet werden, handelt es sich nicht nur um ein Multi-Klassen Klassifikations Problem, sondern noch dazu um ein Multi-Label Klassifikations Problem. Um dieses zu lösen, teilen wir unsere Tweets in einzelne Wörter auf. Diese Wörter werden dann als Vektor in ein künstliches neuronales Netz gegeben, welches einen Hashtag vorschlägt. Dieser Vorschlag wird dann mit dem Label L verglichen und anhand der Differenz, die Parameter des Netzes aktualisiert. So wird durch mehrfaches iterieren über den Datensatz anhand von Beispielen gelernt, welche Hashtags für einen Tweet verwendet werden können. Mit dem trainierten Netz können dann für Tweets ohne Label Hashtags vorgeschlagen werden.

4.2. MLP

In diesem Abschnitt stellen wir neben unserem MLP Modell, auch noch die zwei Kostenfunktionen Kreuzentropie und BP-MLL vor. Wobei letztere für Multi-Label Klassifikationen entworfen wurde und somit auch für unser Problem relevant ist.

Das Multi-Layer Perceptron Modell, welches wir in dieser Arbeit verwenden, besteht aus drei Schichten und bildet das Referenzmodell. Die erste Schicht ist eine Embedding-Layer, wie in Abschnitt 3.2 beschrieben, die für die einzelnen Wörter des Tweets ein Word Embedding erzeugt. Wenn wir die Wörter als Punkte in einem Raum betrachten, dann führt das Embedding dazu, dass ähnliche Wörter eine kleine Distanz zueinander haben, also in eine gemeinsame Region abgebildet werden. Durch diese Abstraktion muss nicht mehr jedes Wort einzeln gewichtet werden, denn es genügt eine Gewichtung der Region, die das Wort beinhaltet, um dessen Relevanz für die Klassifikation zu bestimmen. Auf die Embedding Layer folgt eine voll verbundene Hidden Layer. Abgeschlossen wird das ganze mit einer Softmax Layer, die eine Wahrscheinlichkeitsverteilung für die möglichen Hashtagklassen errechnet. Diese wird dann mit der Verteilung des zugehörigen Hashtag Labels verglichen. Dabei wird die Verteilung für ein Hashtag-Label wie folgt gebildet.

$$\Gamma_t(h) = \begin{cases} 1 & \text{wenn } h \text{ in } t \text{ enthalten ist} \\ 0 & \text{sonst} \end{cases}, \tag{4.1}$$

wobei h ein Hashtag aus der Menge der möglichen Hashtagklassen $\mathcal{H}$ für einen Tweet t ist. Da die Hashtag Vorhersage, wie oben festgestellt, ein Multi-Label-Klassifikations Problem ist, hat ein Label mindestens eine bis möglicherweise $|\mathcal{H}|$ verschiedene Klassenzugehörigkeiten. Um die Ausgabe unseres Netzes zu bewerten, wird die Kreuzentropie als Kostenfunktion gewählt.

$$E(P, Y) = -\sum_{h \in \mathcal{H}} P(h) log(Y(h)), \tag{4.2}$$

mit der berechneten Wahrscheinlichkeitsverteilung des Netzes P und der Verteilung des Labels Y. Wir wollen jedoch den Fakt, dass es sich bei der Hashtag Vorhersage nicht nur um eine Multi-Klassen, sondern eben auch um eine Multi-Label Klassifikation handelt in Betracht ziehen und stellen deshalb noch alternativ die BP-MLL Kostenfunktion vor.

$$E = \frac{1}{|Y||\overline{Y}|} \sum_{(k,l) \in Y \times \overline{Y}} exp(-(c_k - c_l)), \tag{4.3}$$

wobei Y eine Untermenge von $\mathcal{H}$ und $\overline{Y}$ die komplementäre Menge zu Y darstellt. Eine Klasse ist in Y enthalten, wenn das Label des Tweets für diese Klasse einen positiven Eintrag hat. Die Wahrscheinlichkeit, die das Netz für eine Klasse $i \in \mathcal{H}$ berechnet, wird mit c_i angegeben. Diese Kostenfunktion ist gut für Multi-Label-Klassifikationen geeignet, da richtig klassifizierte Klassen höher gewertet werden, bzw. sie wirken sich stärker auf den Fehler aus, als falsch klassifizierte. Dem Vorteil der angepassten Gewichtung steht jedoch die Laufzeit, die mit der Anzahl an möglichen Klassen quadratisch steigt, gegenüber. Für eine genauere Beschreibung der Funktionsweise von BP-MLL verweisen wir auf [ZZ05].

4.3. CNN

Unser CNN Modell ist stark an das *#TagSpace* Modell von [WCA14] angelehnt. Hier wurde das klassische CNN Modell zur Bilderkennung so umgewandelt, dass es für Text-Eingaben geeignet ist. Am Anfang des Netzes steht wie auch beim MLP eine Embedding Layer. Die

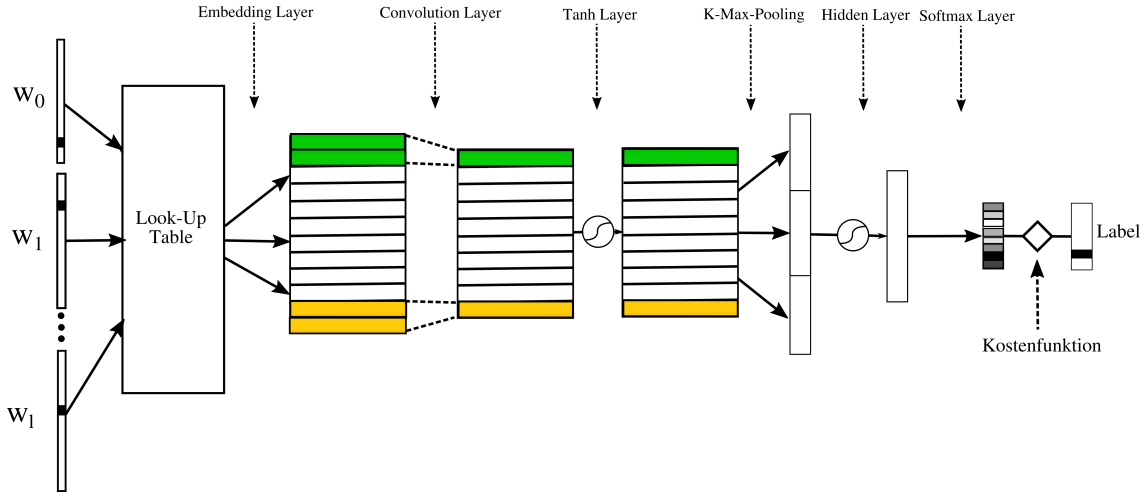

Abbildung 4.1.: CNN mit 3-Max-Pooling und Kontextfenstergröße = 2

Word Embeddings werden dann in eine Convolution Layer gegeben. Diese unterscheidet sich jedoch wesentlich von der Convolution Layer, die in Abschnitt 3.3 beschrieben ist und wird deswegen genauer betrachtet. Nachdem die einzelnen Wörter des Tweets embedded wurden, erhalten wir eine Matrix $I \in \mathbb{R}^{l \times e}$, wobei l die Anzahl der Wörter des Tweets und e die Größe des Embeddings ist. Diese Matrix unterteilen wir nun in sogenannte Kontextfenster K. Ein Kontextfenster der Größe n enthält dann genau n Zeilen aus I. Das Kontextfenster wird über die Zeilen von I iteriert und kann für eine Anfangszeile j durch einen Vektor $K_j = (i_{j,0}, \ldots, i_{j,e}, i_{j+1,0}, \ldots, i_{j+1,e}, \ldots, i_{j+n-1,0}, \ldots, i_{j+n-1,e})$ beschrieben werden, wobei $i_{a,b}$ ein Eintrag aus I ist. Durch ein Padding oberhalb und unterhalb von I erhalten wir so genau l Kontextfenster der Größe $|K_j| = n \cdot e$. Jedes Kontextfenster wird dann mit einem Kernel $F \in \mathbb{R}^{n \cdot e \times f}$ multipliziert, wobei f ein frei wählbarer Parameter ist und die Dimension des Kernels bestimmt. Die Ausgabe der Faltung, eine $l \times f$ Matrix, wird dann durch eine TanH Layer zu einer K-Max-Pool Layer geleitet. K-Max-Pooling funktioniert genauso wie das normale Max-Pooling, mit der Besonderheit, dass pro Pooling Operation nicht der größte Wert der erfassten Feature Map, sondern die k größten Werte bestimmt werden [KGB14]. Die Pooling Operation hat dabei die Größe $l \times 1$ oder anders ausgedrückt, die Größe einer Spalte der Matrix, die wir von der tanH Layer erhalten. Bildlich gesehen, wird diese Spalte dann von links nach rechts über die Ausgabe der tanH Layer mit $stride = 1$ geschoben und speichert für jeden Schritt die k größten Werte. Die ursprüngliche Reihenfolge der k größten Werte pro Spalte wird beibehalten. Es werden dann entsprechend dieser Reihenfolge, k Vektoren der Länge f gebildet. Durch anschließende Konkatenation erhalten wir als Endergebnis des K-Max-Poolings einen Vektor $m \in \mathbb{R}^{k \cdot f}$ mit konstanter Ausgabegröße, den wir an eine Hidden-Layer weiterleiten. Die Ausgabe der Hidden-Layer wird auch wie oben in eine Softmax-Layer gegeben, deren

Ausgabe mit dem Label verglichen wird. Eine Visualisierung des Aufbaus ist in Abb. 4.1 gegeben.

4.4. LSTM

Das LSTM Modell, das hier verwendet wird, besteht aus einer Vorwärts- und einer Rückwärts-Komponente. Dabei haben beide LSTM Einheiten für jeden Zeitschritt t die gleiche Eingabe x_t. Der Unterschied besteht darin, dass die Vorwärts Einheit mit x_0 beginnt und die Rückwärts Einheit mit x_n. Die jeweils letzte Ausgabe der zwei Einheiten, also h_n^f für die Vorwärts und h_0^b für die Rückwärts, werden konkateniert und an eine Hidden Layer weitergeleitet. Die Eingaben x_i sind dabei nicht die einzelnen Wörter, sondern die entsprechenden Embeddings für dieses Wort. Die Worte werden also wie in den vorherigen Modellen erst durch eine Embedding Layer geleitet, bevor sie an das LSTM weitergegeben werden. Dabei kann die Hidden Layer beliebig viele Neuronen besitzen. Nach der Hidden Layer folgt wie üblich die Softmax Layer die eine Wahrscheinlichkeitsverteilung über alle Hashtagkandidaten erstellt. Die Vorwärts und Rückwärts Komponente ist in Abb. 4.2 dargestellt.

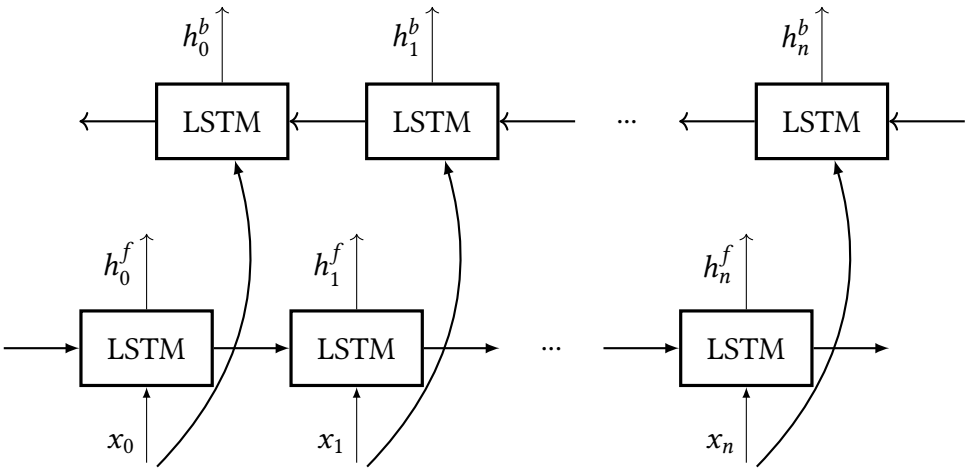

Abbildung 4.2.: Bi-direktionales LSTM, wobei die Ausgaben h_n^f und h_0^b konkateniert und an eine Hidden-Layer weitergeleitet werden.

Eine weitere Möglichkeit, die Ausgaben des LSTM zu nutzen, ist das Average-Pooling aller Ausgaben. Diese Methode wurde von [TQL15] verwendet und führte zu besseren Ergebnissen in der Sentiment Analyse. Wir implementieren dieses Verfahren auch in dieser Arbeit um zu sehen, ob Average-Pooling auch für unser Problem geeignet ist. Dabei werden beim Average-Pooling die Ausgaben zu jedem Zeitpunkt t gespeichert und der Durchschnitt über alle Ausgaben h_t gebildet. Dieser Durchschnitt wird jeweils für beide Richtungen h^f und h^b gebildet, sodass wir durch Konkatenation der beiden Durchschnitte wieder einen Vektor derselben Größe wie zuvor erhalten, der an die Hidden-Layer weitergeleitet wird.

4.5. Tweet to Sequence Modell

In diesem Abschnitt wird durch den Tweet to Sequence Decoder (Tweet2Seq) eine neue Methode eingeführt, um Hashtags vorherzusagen. Bisher wurde zur Vorhersage eines Hashtags eine Klasse aus einer Menge von möglichen Klassen gewählt. Dabei konnten nur bekannte Hashtags bzw. Klassen ausgewählt werden. Die Erweiterung unseres Modells durch einen Sequenz Decoder ermöglicht es neue Hashtags, die noch nie gesehen wurden zu erstellen, oder schon bekannte Hashtags miteinander zu kombinieren. Dies wird durch den sequentiellen Aufbau der Klassifikation verwirklicht. Dazu wird der Ergebnis Hashtag Zeichen für Zeichen sequentiell aufgebaut. Dieses Verfahren basiert auf einem Sequence to Sequence (Seq2Seq) Modell, welches häufig für Übersetzungen verwendet wird. Wir werden im folgenden das klassische Seq2Seq Modell vorstellen, welches auch als Referenzmodell dienen soll. Schließlich werden wir anhand des Konzepts des Seq2Seq Modells unser neues Modell, den Tweet to Sequence Decoder, herleiten und beschreiben.

4.5.1. Sequence to Sequence Modell

Viele Probleme sind zu komplex, um sie durch eine einfache Klassifikation mit verschiedenen Labeln zu modellieren. Die Übersetzung von Texten ist ein Beispiel dafür. Hier macht es mehr Sinn das Ergebnis Wort für Wort aufzubauen, anstatt einem beliebigen Text eine vorgefertigte Übersetzung fester Länge zuzuordnen. Durch den sequentiellen Aufbau erhalten wir also eine speziell auf die Eingabe zugeschnittene Übersetzung variabler Länge. Jedoch wollen wir in unserem Fall nichts übersetzen, sondern einen Tweet klassifizieren. Deswegen bauen wir, anstatt der Übersetzung Wort für Wort, den jeweiligen Hashtag Zeichen für Zeichen auf. In unserem Fall ermöglichen wir dem Modell somit neue Hashtags selbständig zu kreieren, oder bekannte miteinander zu kombinieren. Basierend auf dem Übersetzungsmodell von [SVL14] implementieren wir ein Seq2Seq Modell, das einer Wortsequenz eine Zeichensequenz zuordnet. Das Modell besteht aus zwei Teilen, einem Encoder und einem Decoder. Der Encoder bildet eine Repräsentation des Eingabe Tweets, welche dann an den Decoder weitergeleitet wird und von letzterem dann ein Hashtag vorgeschlagen wird. Der Encoder und der Decoder bestehen jeweils aus einer LSTM Einheit. Die letzte Ausgabe und der letzte Gedächtniszustand des Encoder LSTMs wird als Initialisierung für das Decoder LSTM verwendet. Bevor wir die Wort Sequenz des Tweets der Länge l in den Encoder geben, erzeugen wir wieder mithilfe der Embedding Layer für jedes Wort des Tweets ein Word Embedding. Diese werden dann sequentiell in den Encoder gegeben. Solange die einzelnen Word Embeddings des Tweets in das Netz gegeben werden, werden die Ausgaben zu jedem Zeitschritt ignoriert. Sobald das letzte Wort in den Encoder gegeben wurde, wird mit den finalen Werten h_l^{enc} und mem_l^{enc} des Encoders, der Decoder initialisiert. Als erstes wird jeweils ein # Zeichen, zum Zeitpunkt t, in den Decoder gegeben. Anhand dieser Eingabe und der Tweet Repräsentation, die sich aus der vorherigen Ausgabe h_{t-1} und dem vorherigen Gedächtniszustand mem_{t-1} zusammensetzt (Ausgabe Encoder), wird das erste Zeichen des Ausgabe Hashtags bestimmt. Dieses dient dann für den nächsten Zeitschritt $t + 1$, anstatt dem # Zeichen, als Eingabe des Decoders. Das Ausgabezeichen zum Zeitpunkt $t + 1$ dient dann wieder als Eingabe für

den nächsten Schritt zum Zeitpunkt $t + 2$ usw. Dieses Verfahren wird sukzessiv fortgesetzt bis das *EOF* Zeichen als Ausgabe erscheint, wie in Abb. 4.3 dargestellt.

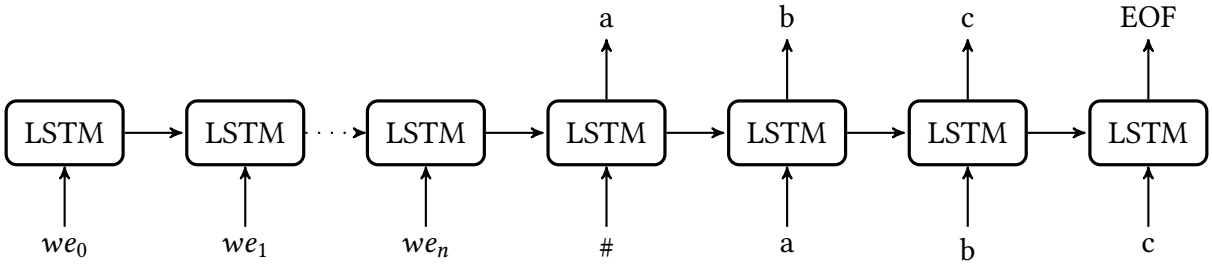

Abbildung 4.3.: Seq2Seq Modell mit den Word Embeddings des Eingabe Tweets $we_0, we_1, \ldots, we_n$ und dem vorhergesagten Hashtag *#abc*

Wie das jeweilige Ausgabezeichen zustande kommt, wollen wir anhand einer näheren Betrachtung eines Decoder Zeitschritts erläutern. Wir betrachten dazu den Zeitpunkt t, mit dem Eingabezeichen $x_t \in \mathcal{A}$. Dabei ist $\mathcal{A}$ das Alphabet aller Zeichen, die in den Hashtags des Datensatzes vorkommen. Bevor x_t in die LSTM Einheit gegeben wird, durchläuft das Eingabezeichen eine Zeichen Embedding-Layer um eine homogene Eingabegröße zu bewahren. Von der LSTM Einheit erhalten wir eine *Ausgabe* h_t und ein Gedächtniszustand mem_t, die beide für den nächsten Zeitschritt benötigt werden. Die Ausgabe h_t wird dann an eine Softmax-Layer weitergeleitet, die eine Verteilung über den Zeichenraum erzeugt. Von dieser Verteilung suchen wir das Element mit der größten Wahrscheinlichkeit. Dieses ist dann unser *Ausgabezeichen*, das auch die Eingabe x_{t+1} für den nächsten Zeitschritt bildet.

4.5.2. Tweet to Sequence Decoder

Der Tweet to Sequence Decoder ist eine Kombination des Bi-LSTM Modells, das in Abschnitt 4.4 vorgestellt wurde und dem Decoder des Seq2Seq Modells. Mit dem Bi-LSTM erzeugen wir zuerst eine Tweet Repräsentation, die wir dann an einen Sequenz Decoder weitergeben. Dabei wird die Ausgabe der Hidden-Layer des Bi-LSTM in zwei gleich große Vektoren geteilt. Diese Vektoren dienen als Initialisierung der Decoder LSTM Einheit. Dabei wird der erste Vektor als h_0 verwendet und der zweite als mem_0. Auch hier wird als erstes wieder das # Zeichen als initiale Eingabe in den Decoder gegeben. Das komplette Modell ist in Abb. 4.4 dargestellt.

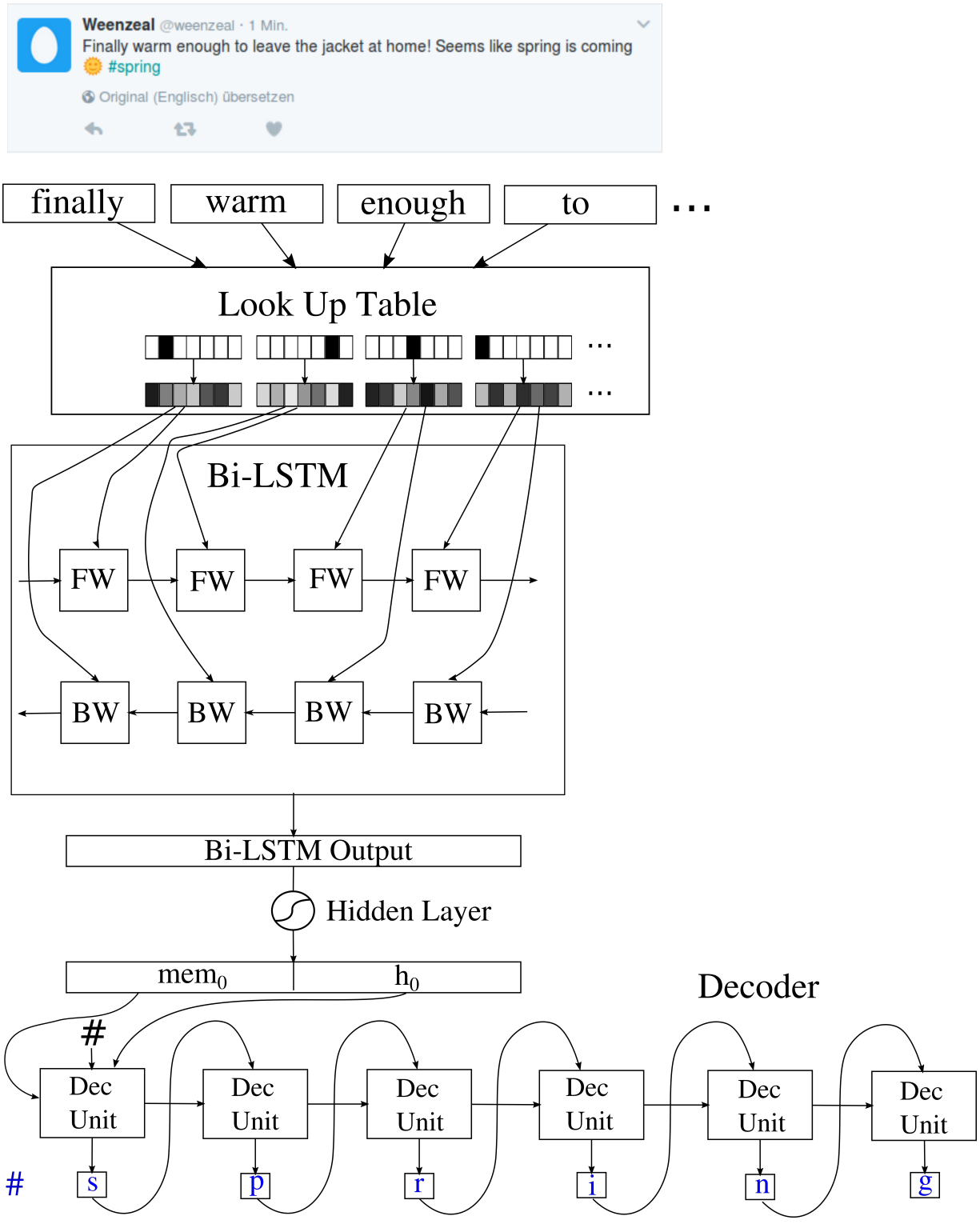

Abbildung 4.4.: Komplettes Tweet2Seq Modell von der Eingabe bis zur Ausgabe

Da wir uns bisher nur mit der Funktionsweise der Hashtag Generierung des Tweet2Seq Modells beschäftigt haben, werfen wir noch einen Blick auf den Lernprozess. Sei $a = (a_1, a_2, \ldots, a_l)$ ein Tweet mit l Wörtern und $b = (b_1, b_2, \ldots, b_{\bar{l}})$ der dazugehörige Hashtag, der aus $\bar{l}$ Zeichen besteht, dann ist es das Ziel des Modells, die Wahrscheinlichkeit $p(b_1, \ldots, b_{\bar{l}} | a_1, \ldots, a_l)$ zu maximieren. Wenn r die Tweet Repräsentation des Bi-LSTMs darstellt, kann die Wahrscheinlichkeit wie folgt berechnet werden.

$$p(b_1, \ldots, b_{\bar{l}} | a_1, \ldots, a_l) = \prod_{i=1}^{\bar{l}} p(b_i | r, b_1, \ldots, b_{i-1}) \tag{4.4}$$

Sei nun v_i die Wahrscheinlichkeitsverteilung, die sich aus der Softmax Layer des Decoders zum Zeitpunkt i ergibt. Dann ist $p(b_i | r, b_1, \ldots, b_{i-1})$ der Wert der Verteilung an der Stelle des entsprechenden Zeichens $v_i(b_i)$. Um diese Wahrscheinlichkeit nun zu maximieren, trainieren wir das Netz mit einer der Kreuzentropie ähnlichen Kostenfunktion. Dazu berechnen wir für jede Ausgabewahrscheinlichkeitsverteilung des Decoders die Kreuzentropie und summieren dann über alle Ausgaben auf.

$$E = -\sum_{i=1}^{\bar{l}} \sum_{c \in \mathcal{A}} \mu_i(c) \log(v_i(c)), \tag{4.5}$$

wobei μ_i die Verteilung für das i-te Zeichen des Hashtag Labels über dem Alphabet $\mathcal{A}$ darstellt. Durch Minimierung dieser Kostenfunktion wird die Wahrscheinlichkeit (4.4) maximiert. Die Anpassung der Parameter wird mit BPTT realisiert.

4.5.3. Bi-LSTM mit CNN - Decoder

Bisher haben wir bei komplexeren Modellen nur immanente Variationen vorgenommen, sodass die klassischen Merkmale, die dieses Modell ausmachen, erhalten blieben. In diesem Abschnitt stellen wir eine Netzarchitektur vor, die zwei verschiedene Neuronale Netze Modelle, nämlich das LSTM und das CNN vereint und somit über die Grenzen der einzelnen Modelle hinausgeht.

Dazu passen wir das Modell aus Abb. 4.4 an und schalten parallel zum Bi-LSTM ein CNN. Dieses CNN besteht, ähnlich wie unser Modell in Abschnitt 4.3, aus einer Convolution Layer und einer K-Max-Pooling Layer. Damit erhalten wir eine Netzstruktur, die in Abb. 4.5 dargestellt ist.

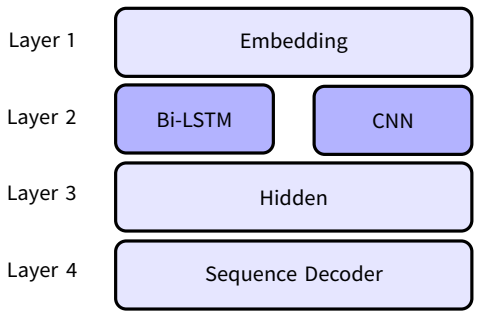

Abbildung 4.5.: Vereinfachte Darstellung des Bi-LSTM + CNN - Sequenz Decoder

Die Ausgabe des K-Max-Pooling wird an die Hidden Layer weitergeben, an die auch die Ausgabe des Bi-LSTMs gegeben wird. Die Word Embeddings, die dem Bi-LSTM als Eingabe dienen, werden auch als Eingabe für die Convolutional Layer genutzt. Für jeden Tweet wird also die Ausgabe des CNNs mit der Ausgabe des Bi-LSTMs konkateniert und an die Hidden-Layer weitergegeben. Diese bildet dann, wie vorher auch, die Basis für den Decoder.

5. Ergebnisse

In diesem Kapitel vergleichen wir die im vorherigen Kapitel vorgestellten neuronalen Netze miteinander, testen Optimierungsmethoden auf ihre Effizienz und analysieren die vorgeschlagenen Hashtags. Bevor wir jedoch die Performanz der oben vorgestellten Systeme vergleichen, werfen wir noch einen Blick auf die verwendeten Trainingsdatensätze.

Mithilfe der Twitter API haben wir zwei Datensätze gesammelt. Den ersten haben wir durch filtern nach den Top 10 Trending Hashtags zusammengestellt. Der zweite entstand durch das Sammeln aller Tweets, die mit mindestens einem Hashtag versehen sind. Danach haben wir die Tweets, die einen der Top 100 Hashtags enthalten, herausgefiltert. Alle Tweets wurden in chronologischer Reihenfolge gespeichert. Hashtags, die zum größten Teil oder ausschließlich von Bots verwendet werden und die dazu gehörigen Tweets wurden gelöscht. Um jedoch auf 100 verschiedene Hashtags zu kommen, wurde für jeden gelöschten Hashtag der nächst häufigste in den Datensatz aufgenommen. Beide Datensätze wurden sowohl von Spam Tweets befreit, die mehrmals hintereinander getweetet wurden, als auch von Retweets. Nach der Filterung bestehen die Datensätze jeweils aus mehr als 300.000 Tweets. Jedem Tweet ist dabei mindestens einer bis maximal 10 bzw. 100 Klassen (Hashtags) zugeordnet. Wir haben Hashtags die in einem Tweet enthalten sind gelöscht, da diese ja vorhergesagt werden sollen. URLs wurden durch ein URL-Token und *@user* Notationen entsprechend durch ein Benutzer-Token ersetzt. Die Tweets wurden in Tokens umgewandelt und die Häufigkeit des Auftretens jeder dieser Tokens, wurde festgestellt. Die 10.000 am häufigsten vorkommenden Tokens wurden behalten und alle anderen Tokens mit einem *UNKNOWN*-Token ersetzt. Ein weitere Datensatz, der nur für das Tweet2Seq Modell verwendet wird, wurde analog zum Top 100 Hashtag Datensatz erzeugt, jedoch ohne das Filtern der häufigsten Hashtags. Dieser *große Datensatz* besteht also quasi aus den Rohdaten der Top 100 Hashtags, auf die jedoch die Tokenisierung angewandt wurde. Das Ausgabealphabet für die sequentielle Hashtag Vorhersage wurde auf die Zeichen *a-z*, *1-9* und dem _ Zeichen beschränkt. Dadurch wurden Tweets, die mit Hashtags, die chinesischen Zeichen oder anderen Spezialzeichen enthalten, markiert wurden, aus dem Datensatz gelöscht. Der große Datensatz enthält nach der Anwendung der genannten Operationen knapp über 1 Millionen Tweets und 320.737 verschiedene Hashtags. Jeder Datensatz besteht aus zwei Textdateien, wobei in der ersten die tokenisierten Tweets und in der zweiten die dazugehörigen Hashtags enthalten sind. Genau genommen haben wir sogar fünf Datensätze. Während für den großen Datensatz nur sequentielle Label verwendet werden, gibt es für die Top 10 und Top 100 Hashtags jeweils ein Datensatz mit klassischen Labeln, und einen mit sequentiellen Labeln. Diese unterscheiden sich jedoch nicht inhaltlich, sondern nur syntaktisch voneinander. Die Datensätze wurden jeweils in Trainingsdaten, Validierungsdaten und Testdaten unterteilt. Dabei ist sowohl für die Top 10 als auch für die Top 100 Hashtags eine Verteilung von 80% Trainingsdaten, 10% Validierungsdaten und 10% Testdaten angesetzt. Somit ist der Korpus der Trainingsdaten für diese

Datensätze ca. 240.000 Tweets groß. Dementsprechend enthalten die Validierungs- und Testdaten jeweils ca. 30.000 Tweets. Für den großen Datensatz haben wir eine Verteilung von 90% Trainingsdaten, 5% Validierungsdaten und 5% Testdaten gewählt. Somit haben Validierungs- und Testdaten einen Umfang von ca. 50.000 Tweets, während der dazugehörige Trainingsdatensatz ca. 900.000 Tweets umfasst. Die Validierungs- und Testdaten wurden jeweils an einer zufälligen Stelle am Stück aus dem gesamten Datensatzes ausgeschnitten und zwar so, dass es keine Überlappung zwischen den beiden Datensätzen gibt. Der Rest, der nach Ausschneiden der beiden Blöcke übrig blieb, wurde dann als Trainingsdatensatz definiert. Trotz Filterung von Spam Tweets und Hashtags, die fast ausschließlich von Bots verwendet werden, enthalten die Datensätze noch verstreut Tweets von Bots. Da diese jedoch oft gut geeignet sind, um Tweet Repräsentationen für bestimmte Themen zu lernen und auch in einer realen Anwendung vorkommen würden, behalten wir diese in den Datensätzen. Des Weiteren gibt es eine nicht geringe Anzahl an Tweets, die sehr kurz sind und teilweise aus ein bis zwei Wörtern, oder sogar nur aus einem Hashtag ohne eigentlichen Tweet, bestehen. Damit steigt die Wahrscheinlichkeit, dass identische Tweets öfters vorkommen. Aus den eben genannten Gründen haben wir daher eine partielle Überschneidung von Trainings- und Testdaten bei allen drei Datensätzen, die in Tabelle dargestellt 5.1 ist.

Datensatz	Überschneidung in %
Top 10 #	25.72
Top 100 #	28.59
Großer	19.19

Tabelle 5.1.: Überschneidung von den Testdaten bezüglich der Trainingsdaten

5.1. Fehlermaße

Um die Güte unsere Netze zu bestimmen, benutzen wir durchgehend den *One-Error*. Diese Fehlerfunktion misst, wie oft die höchst bewertete Klasse der Ausgabe nicht in der Menge der korrekten Klassen des Labels enthalten ist.

$$err_D = \frac{1}{|D|} \sum_{d \in D} \begin{cases} 1, & Y_d(\arg\max O_d) \neq 1 \\ 0, & sonst \end{cases} , \tag{5.1}$$

mit Datensatz D, der Ausgabe des Netzes O_d als Wahrscheinlichkeitsverteilung und dem Label Y_d, wobei $Y_d(i)$ die Zugehörigkeit zu der i-ten Klasse für jeweils einen Datenpunkt $d \in D$ widerspiegelt. In unserem konkreten Fall bedeutet dies, dass der Hashtag der mit der höchsten Wahrscheinlichkeit bewertet wird, mit einem der tatsächlichen Hashtags, die im ursprüngliche Tweet verwendet wurden, übereinstimmen muss. Der optimale Fehler wäre also $err_d = 0$ und unser Modell ist umso besser, je kleiner der One-Error ausfällt.

Eine Fehlerfunktion die wir *Five-Error* nennen, funktioniert ähnlich wie der One-Error. Jedoch genügt es hier, wenn anstatt der wahrscheinlichsten, eine der 5 wahrscheinlichsten

prädizierten Hashtag Klassen mit einer Klasse aus dem Label übereinstimmt. Diese Fehlerfunktion wird für die Top 100 Hashtag Klassifikationen verwendet, bei denen die Auswahl an verschiedenen Klassen deutlich höher ist und es somit mehrere Hashtags geben kann die zu einem Tweet passen. Eine Ausgabe des Modells könnten beispielsweise die zwei Hashtags *#food* und *#dinner* mit den zwei höchsten Wahrscheinlichkeiten $P(\#food) = 0.6$ und $P(\#dinner) = 0.4$ sein. Das Label enthält jedoch nur die Klasse *#dinner*. Dann wäre *#food* immer noch eine gute Vorhersage, auch wenn sie nach dem One-Error falsch wäre. Wir wollen also dem Modell mithilfe des Five-Error mehr Chancen geben, bei ähnlichen Hashtags denjenigen zu treffen, der tatsächlich im Label enthalten ist.

Für das Tweet2Seq Modell wollen wir noch zwei andere Fehlermaße einführen, die für die sequentielle Ausgabe aussagekräftiger sind als der One-Error. Das erste Fehlermaß ist der sogenannte *BLEU Score*. Dieses Fehlermaß wird primär genutzt um die Güte von maschinellen Übersetzungen zu bestimmen. Dazu wird ein Übersetzungskandidat auf Übereinstimmungen bezüglich n-Gramme, mit einer oder mehreren Referenzen getestet. Für eine ausführliche Erklärung von BLEU verweisen wir auf [Pap+02]. In unserem Fall verwenden wir Zeichen anstatt Wort n-Gramme und berechnen somit den C-BLEU (Character-BLEU) mit $n = 4$ für unsere Hashtagkandidaten. Obwohl BLEU eine sehr gute Güte für Übersetzungen bietet, können wir ohne weitere Untersuchungen nicht sagen, wie wertvoll die Ergebnisse unseres angepassten C-BLEU Scores sind. Deswegen führen wir noch die *Character Error Rate* (CER) ein, die ein anschaulicheres Gütemaß bietet. Diese basiert auf der Word Error Rate [Wai16] und berechnet sich aus der Anzahl Ersetzungen, Löschungen und Einfügungen von einzelnen Zeichen, die benötigt werden, um aus der Hypothese die Referenz zu konstruieren.

$$CER = \frac{\#Sub + \#Ins + \#Del}{Z}, \tag{5.2}$$

wobei $\#Sub$ die Anzahl an Ersetzungen, $\#Ins$ die Anzahl an Einfügungen, $\#Del$ die Anzahl Löschungen und Z die Anzahl der Zeichen der Referenz sind.

5.1.1. Komplexität bezüglich Fehlermaß

Um später eine Vorstellung davon zu haben, wie gut die hier vorgestellten Methoden funktionieren, legen wir hier die Komplexität unseres Problems für die beiden Datensätze bezüglich des One-Errors dar. Dazu berechnen wir den One-Error über den gesamten Datensatz, wobei für jeden Tweet der Hashtag als Klassifizierung vorgeschlagen wird, der am häufigsten im gesamten Datensatz auftaucht. Für den Top 10 Hashtags Datensatz war der häufigste Hashtag *#trump*. Wenn alle Tweets mit *#trump* klassifiziert werden, beträgt der One-Error 79.16%. Für den Top 100 Hashtags Datensatz wurde analog ein One-Error von 90.35% für den am häufigsten verwendeten Hashtag *#thanksgiving* berechnet. Dass sich der meist verwendete Hashtag für die Top 10 und Top 100 unterscheidet, ist auf den zeitlichen Abstand zwischen den Aufnahmen der einzelnen Datensätze zurückzuführen. Außerdem ist zu beachten, dass manche Tweets selbst für Menschen unmöglich vorherzusagen sind, da erst der dazugehörige Hashtag dem Tweet einen Sinn gibt. Ein Beispiel dafür ist der Tweet *"I do love"*. Hier würde ein Mensch und auch mit hoher Wahrscheinlichkeit ein neuronales Netz den Hashtag *#love* vorschlagen. Dieser ist zwar nicht völlig unpassend,

jedoch hat dieser keinen Bezug auf die Intention des Nutzers, da das Original *"I do love #watchdog"* lautet. Dies ist leider kein Einzelfall sondern, wie weitere Beispiele im Anhang A.1 zeigen, öfters der Fall und sollte daher bei der Komplexitätsfrage beachtet werden.

5.2. Multi-Layer Perceptron

Bevor wir uns mit der eigentlichen Hashtag Vorhersage beschäftigen, verwenden wir das Multi-Layer Perceptron als grundlegende Herangehensweise um die Optimierungsmethoden, die in Kapitel 3.5 beschrieben wurden, zu testen. Die Anwendung dieser Optimierungen soll dem Phänomen, das in Abb. 5.1 gezeigt ist, entgegenwirken. Man sieht dort, wie ab einem bestimmten Zeitpunkt der Validierungsfehler nicht mehr sinkt sondern wieder leicht ansteigt, während der Fehler auf den Trainingsdaten weiter sinkt. Dies ist der sogenannte Overfitting Effekt. Um dieses Auswendiglernen der Trainingsdaten ein-

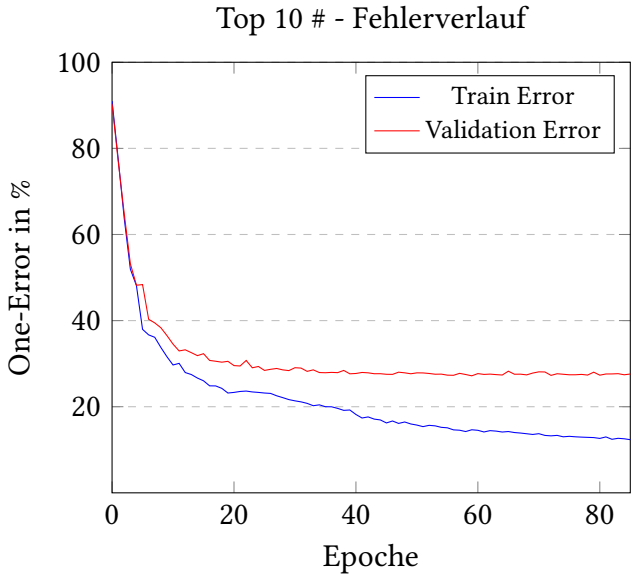

Abbildung 5.1.: Visualisierung des Overfitting Effekts bei einem MLP.

zudämmen, wenden wir die Regularisierungsmethoden auf das MLP an. Die Verfahren, die wir nach und nach unserem Modell hinzufügen, sind die L2-Regularisierung, Dropout, Mischen der Trainingsdaten und Momentum. Momentum baut auf dem Stochastischen Gradienten Abstieg auf und benutzt eine dynamische Lernrate, sowie ein dynamisches Moment, um diesen zu optimieren. Man kann dieses Verfahren als eine Vereinfachung von ADAM sehen. Wir sind zu dem Ergebnis gekommen, dass durch Hinzunahme von Regularisierungsmethoden die Fehlerrate verringert werden kann und durch das Verbessern des SGD eine schnellere Konvergenz erzielt wird. Diese Verbesserung der Ergebnisse ist in Tabelle 5.2 zu sehen.

Aufgrund der positiven Resultate, die durch die Optimierungen erzielt werden konnten, werden für alle folgenden Untersuchungen jeweils alle Methoden, die in Tabelle 5.2 aufgeführt sind, verwendet. Weiterhin haben wir die Auswirkung einer Erweiterung des

MLP - Top 10 Hashtags

Optimierungen	One-Error in %
-	34.02
L2 Reg & Momentum	30.88
Dropout	27.33
Mischen	26.54

Tabelle 5.2.: Eine neue Optimierung schließt immer alle darüber liegenden mit ein.

MLP, von einer Hidden Layer auf zwei Hidden Layers, untersucht. Jedoch konnten die Ergebnisse durch zwei Schichten nicht verbessert werden, wie in Tabelle 5.3 zu sehen ist.

One Error - MLP

Anzahl Hidden-Layers	Top 10 #	Top 100 #
1	26.54	55.76
2	26.95	55.75

Tabelle 5.3.: Auswirkung von verschiedener Anzahl an Hidden Layers

5.2.1. BP-MLL

Obwohl es sich bei unserem Problem um eine Multi-Label Klassifikation handelt, brachte die Änderung der Kostenfunktion zu BP-MLL keine Verbesserung zur Kreuzentropie. Dies könnte daran liegen, dass wir die Gradienten nicht wie bei [ZZ05] beschrieben von Hand berechnet und in unseren Code eingefügt, sondern die automatische Gradientenberechnung von Theano benutzt haben, die die Gradienten anhand des entstehenden Graphen bestimmt und deswegen von den manuell berechneten abweichen könnte [The16]. Die Ergebnisse der extra für Multi-Label Klassifikationen entwickelte Kostenfunktion BP-MLL, sind in Tabelle 5.4 zu finden. Da nicht nur die Ergebnisse nicht verbessert wurden, sondern

BP-MLL - One Error

Kostenfunktion	Top 10 #	Top 100 #
Kreuzentropie	26.54	55.76
BP-MLL	32.82	58.53

Tabelle 5.4.: MLP mit verschiedenen Kostenfunktionen

auch der Rechenaufwand quadratisch mit der Anzahl der Label wächst, wird für alle weiteren Untersuchungen die Kreuzentropie Kostenfunktion verwendet.

5.3. CNN

Durch die Verwendung von CNNs konnten wir, nach dem justieren der Hyperparameter, die Fehlerrate für die Top 10 als auch für die Top 100 Hashtags verbessern. Die optimalen

Parameter sind: eine Embedding Größe von 256, eine Filtergröße von 500, eine Fenstergröße von 5 und eine Hidden Layer Größe von 512, die wir für alle CNNs verwenden. Wir haben außerdem den Einfluss von verschiedenen K-Max-Pooling Variationen getestet. Bei den Top 10 Hashtags haben die verschiedenen Pooling Variationen nicht mehr als ein Rauschen verursacht und keine große Auswirkung gehabt. Bei den Top 100 Hashtags konnte jedoch eine leichte Steigerung der Performance bei erhöhtem k festgestellt werden.

CNN K-Max-Pooling - One Error

k	Top 10 #	Top 100 #
1	23.59	51.71
2	**23.50**	51.04
3	23.79	**50.64**

Tabelle 5.5.: Verschiedene Max-Pooling Größen bei einem CNN. Das Beste Ergebnis wurde jeweils hervorgehoben

Weiterhin haben wir für CNN und MLP den Unterschied zwischen Sum-Pooling und Max-Pooling getestet und haben herausgefunden, dass in beiden Fällen das Max-Pooling besser geeignet ist, wie man in Tabelle 5.6 sehen kann. Sum-Pooling ist das Standard Verfahren des MLPs, das auch für die bisherigen MLP Modelle verwendet wurde, um eine einheitliche Tweet Repräsentation zu erreichen. Dabei wird jedes Word Embedding, das aus der BoW Repräsentation entspringt aufsummiert, um einen Vektor einheitlicher Größe zu erhalten, der an die Hidden Layer weitergegeben werden kann. Das Max-Pooling des MLPs funktioniert genauso, wie das K-Max-Pooling des CNNs mit $k = 1$. Es wird also jeweils der größte Wert jeder Spalte der Repräsentationsmatrix ermittelt, um diese maximalen Werte dann zu einem einheitlichen Vektor zusammenzufügen. Dabei entspricht eine Zeile der Repräsentationsmatrix einem Word Embedding. Der wesentliche Unterschied zu dem CNN Pooling besteht darin, dass die Pooling Operation direkt nach der Embedding Layer durchgeführt wird, um einen einheitlichen Vektor als Netzeingabe zu erhalten, während bei unserem CNN das Pooling erst kurz vor der Softmax Layer am Ende des Netzes angewendet wird.

CNN/MLP Vergleich - Top 10 Hashtags

One-Error in %	Sum-Pool	Max-Pool
MLP	26.54	24.60
CNN	24.16	23.59

Tabelle 5.6.: Vergleich zwischen Sum- & Max-Pooling für CNN und MLP

5.4. LSTM

Mithilfe von einem bi-direktionalen LSTM konnten wir unser Ergebnis, für die Klassifizierung der Top 100 Hashtags, nochmals verbessern. Wir haben bei diesem Modell zwei Variationen der Ausgabe gewählt, die wir an die Hidden Layer weiterleiten. Zuerst eine

Version, bei der wir den Durchschnitt der Ausgaben des LSTMs für jeden Zeitschritt verwendet haben und einmal die bekanntere Version, bei der wir jeweils nur die Ausgabe des letzten LSTM-Schritts an die Hidden Layer weiterleiten. Wir haben hier um eine möglichst gute Performanz Referenz zum CNN zu erhalten, die Embedding Größe auf 256 gesetzt und die Hidden Layer Größe an die des CNNs angepasst. Wir sind zum Ergebnis gekommen,

LSTM - One Error

Datensatz	Average-Pooling	Regular
Top 10 #	25.62	25.52
Top 100 #	44.42	44.17

Tabelle 5.7.: Unterschied zwischen Average-Pooling und der Regulären Verwendung der letzten Ausgabe des LSTMs

dass sich die beiden Varianten bezüglich der Fehlerrate nur minimal unterscheiden und die Differenz nicht ausreicht, um ein Rauschen auszuschließen. Obwohl die Verwendung von Average-Pooling zu Verbesserungen bei der Sentiment Analyse von [TQL15] geführt hat, führte das gleiche Verfahren bei unserem Problem zu keinem Vorteil gegenüber der bekannten Methode. Um nun unabhängig vom Referenzmodell das bestmögliche Ergebnis zu erzielen, haben wir das LSTM noch mit verschiedenen Änderungen getestet. Eine simplere Version des LSTM, bei der wir die Hidden-Layer nach den LSTM-Einheiten entfernt haben, nennen wir Min-LSTM. Dort gehen die Ausgaben des LSTM also direkt in eine Softmax-Layer. Für beide LSTMs, also mit und ohne Hidden-Layer, haben wir den Unterschied von Average-Pooling zur Standard Ausgabe und den Einfluss der Embedding Größe getestet. Die Ergebnisse sind in Tabelle 5.8 zu finden. Im Durchschnitt ist das LSTM mit

One-Error in %	*Min-LSTM*		*LSTM*	
Embedding	Average-Pooling	Regular	Average-Pooling	Regular
128	46.17	45.78	45.21	46.04
256	44.92	44.83	44.42	44.17
512	44.04	44.01	43.96	43.76

Tabelle 5.8.: Vergleich verschiedener LSTM-Modelle - Top 100 #

Hidden-Layer effektiver als das ohne, auch wenn die Differenz eher gering ausfällt. Analog verhält sich wie oben schon festgestellt, die reguläre Ausgabe zum Average-Pooling. Einen deutlichen Einfluss auf das Ergebnis hatte jedoch die Embedding Größe. Hier wurden die Ergebnisse umso besser, je größer das Embedding.

5.5. Tweet to Sequence Modelle

Die bisher verwendeten Modelle haben alle aus einer Menge von möglichen Hashtags den, bezüglich der zuvor berechneten Tweet Repräsentation, geeignetsten ausgewählt. Durch das Tweet2Seq Modell wird der Ausgaberaum vergrößert, sodass auch neue Hashtags, die nicht in den Trainingsdaten enthalten sind, erzeugt werden können. Im folgenden wird

die Performanz dieser sequentiellen Decoder Modelle getestet. Wir haben 4 verschiedene Modelle auf den Top 10 und Top 100 Hashtags getestet. Das erste und einfachste Modell ist ein Sequence to Sequence Modell wie in Abschnitt 4.5.1 beschrieben, wobei kein eigenes LSTM für Encoder und Decoder verwendet wurde, sondern nur ein einziges LSTM, das beides - Encoder und Decoder- zugleich ist, und die Ausgaben erst ab dem Zeitpunkt, an dem das # Zeichen als Eingabe anliegt, relevant sind. Damit dies funktioniert, muss die Embedding Größe der Zeichen, der Embedding Größe der Tweets angepasst werden. Diese Modell nennen wir *Basic Seq2Seq*. Das zweite Modell, das dem in Abschnitt 4.5.1 beschriebenen Modell exakt entspricht, nennen wir *Seq2Seq*. Der Unterschied zwischen dem Basic Seq2Seq Modell und dem Seq2Seq Modell besteht darin, dass das Basic Seq2Seq Modell nur aus einem LSTM besteht, welches gleichzeitig als Encoder und Decoder fungiert, während dem Seq2Seq Modell ein LSTM für den Encoder und ein zweites für den Decoder zur Verfügung steht. Für diese und alle folgenden Sequenz Modelle haben wir eine Zeichen Embedding Größe von 50 gewählt. Das dritte Modell ist das erste in Abschnitt 4.5.2 beschriebene Netz ohne das CNN, welches wir *Tweet2Seq* nennen. Das letzte Modell ist das darauf folgende Modell mit dem zusätzlichen CNN neben dem Bi-LSTM, welches wir *Bi-LSTM+CNN* nennen. Jedes dieser Modelle benutzt eine Word Embedding Größe von 256. Die Parameter der LSTM Einheiten, sowie die Filtergröße des CNNs haben ebenfalls eine Größe von 256. Die Hidden Layer hat dementsprechend jeweils eine Ausgabe Größe von 512. Die Eingabe Größe der Hidden Layer ist je nach Modell angepasst, da die Ausgabe des Bi-LSTM+CNN Modells größer ist als die der anderen Modelle. Für das CNN wird aufgrund der bisherigen Ergebnisse eine Fenstergröße von 5 und $k = 3$ für das K-Max-Pooling gewählt. Da das erstellen der Hashtags Zeichen für Zeichen offensichtlich ein komplexeres Problem als die einfache Klassifikation ist, verwundert es nicht, dass die Ergebnisse der Sequenz Decoder Modelle bezüglich der Top 10 und Top 100 Hashtags nicht mit den vorherigen Modellen mithalten können. Den Mehrwert, den die Sequenz Decoder Modelle trotz schlechterer Klassifikation bieten, wollen wir an konkreten Beispielen zeigen. Zuvor werfen wir aber noch einen Blick auf die Performanz der vier Modelle. Während,

Modell	C-BLEU	CER	One-Error
Basic Seq2Seq	85.71	1.96	28.91
Seq2Seq	**85.90**	**1.81**	27.69
Tweet2Seq	85.58	1.88	**27.53**
Bi-LSTM+CNN	85.50	1.89	28.03

Tabelle 5.9.: Performanz der Sequenz Decoder Modelle auf den Top 10 Hashtags. Das Beste Ergebnis ist jeweils hervorgehoben

wie man in Tabelle 5.9 sieht, für die Top 10 Hashtags das Seq2Seq und das Tweet2Seq ungefähr gleich gute Ergebnisse liefern, kann das Bi-LSTM+CNN Modell nur den dritten Platz belegen. Eine Erklärung hierfür wäre das Occam's Razor Prinzip. Es besagt, dass das einfachste System, das ein Problem beschreibt, auch das beste ist. Das Bi-LSTM+CNN könnte also zu komplex sein. Eine andere Erklärung könnte sein, dass die Ausgaben der anderen Modelle nur auf LSTMs basieren. Da diese Ausgaben dann wieder an einen

LSTM Decoder weitergeleitet werden, könnte es für das Modell leichter sein die richtigen Parameter zu lernen, da die Teilsysteme ähnlich sind. Bei der Bi-LSTM+CNN Ausgabe muss das Netz nämlich lernen, aus einer Konkatenation von einer Bi-LSTM Ausgabe und einer CNN Ausgabe eine neue LSTM Initialisierung zu schaffen. Dies ist schwieriger, als eine LSTM Ausgabe in eine LSTM Eingabe zu verwandeln.

Modell	C-BLEU	CER	One-Error
Basic Seq2Seq	64.11	5.32	60.32
Seq2Seq	68.74	5.28	**57.92**
Tweet2Seq	**70.92**	**5.08**	58.31
Bi-LSTM+CNN	66.72	5.95	61.59

Tabelle 5.10.: Performanz der Sequenz Decoder Modelle auf den Top 100 Hashtags. Das Beste Ergebnis ist jeweils hervorgehoben

Auch bei den Top 100 Hashtags tritt dasselbe Phänomen wie bei den Top 10 Hashtags auf, wie in Tabelle 5.10 zu sehen ist. Auch hier kann das Bi-LSTM+CNN trotz der höheren Parameter Anzahl nicht mit dem Seq2Seq und Tweet2Seq Modell mithalten. Obwohl die beiden Modelle Seq2Seq und Tweet2Seq fast identische Ergebnisse liefern, haben wir uns für den großen Datensatz, der über 330.000 verschiedene Hashtags enthält, für das Tweet2Seq Modell entschieden, da durch die Rückwärtskomponente im LSTM unabhängig von der Vorwärtskomponente noch Merkmale gespeichert werden können, was bei einem so großen Suchraum von Vorteil sein kann. Das mit dem großen Datensatz trainierte Modell vergleichen wir mit dem auf den Top 100 Hashtags trainierten Modell. Dabei nutzen wir jeweils drei Testdatensätze. Der erste Testdatensatz ist der normale Testkorpus des großen Datensatzes. Der zweite Testdatensatz wird gefiltert, sodass nur noch die Top 100 Hashtags in diesem enthalten sind. Der dritte Testdatensatz wird mit den Top 10 Hashtags gefiltert. Es werden also alle Tweets aus dem original Testdatenkorpus des großen Datensatzes, die nicht mit einem der Top 10 bzw. Top 100 Hashtags markiert wurden, gelöscht. Wir nennen das Modell, das auf dem großen Datensatz trainiert wurde *Modell Groß* und das Modell, das auf den Top 100 Hashtags trainiert wurde *Modell Top 100*.

	Modell Groß			Modell Top 100		
One Error	66.02	70.40	79.46	33.14	58.31	95.31
C-BLEU	64.79	58.23	52.63	82.66	70.92	50.49
CER	4.49	5.64	7.01	2.84	5.08	9.77
Filter	Top 10	Top 100	-	Top 10	Top 100	-

Tabelle 5.11.: Vergleich der zwei Tweet2Seq Modelle, Modell Groß und Modell Top 100. Beide Modelle wurden jeweils für die drei Testdatensätze ausgewertet und sind durch die verschiedenen Filter gekennzeichnet.

Tabelle 5.11 zeigt, dass das Modell Top 100 für die Top 10 und Top 100 gefilterten Testdaten sehr gut abschneidet, jedoch für die gesamten Testdaten des großen Datensatzes

ein sehr schlechtes Ergebnis erzielt. Im Gegensatz zu dem Modell Groß, welches für den gesamten Testdatensatz ein akzeptables Ergebnis liefert, aber für die Top 10 und Top 100 gefilterten Testdaten deutlich schlechter als das Top 100 Modell ist. Die schlechteren Ergebnisse bezüglich der Top 10 und Top 100 Hashtags, lassen sich durch die erheblich größere Auswahl an möglichen Hashtags erklären. Während ein Modell, welches auf den Top 10 oder Top 100 Hashtags trainiert wurde, schon nach dem ersten bzw. den ersten zwei bis drei Zeichen erahnen kann, welcher Hashtag in Frage kommt, gibt es bei dem großen Datensatz mehr als 300.000 verschiedene Hashtags, wodurch auch nach dem dritten oder vierten Zeichen noch nicht feststeht, auf welchen Hashtag die bisherige Ausgabe hinauslaufen soll. Jedoch kann, wie man an dem schlechten Ergebnis des Modells Top 100 auf den gesamten Testdaten sieht, durch eine zu kleine Auswahl an Hashtags, nicht die gesamte Bandbreite von möglichen Tweets und deren Themen abgedeckt werden. Hier die Balance zwischen einer großen Themenabdeckung und einer moderaten Anzahl an Hashtags zu finden, um die Vorhersage zu optimieren, könnte Thema zukünftiger Forschungsarbeiten sein.

5.5.1. Beispielhafte Analyse einzelner Ausgaben

In diesem Abschnitt wurde das Tweet2Seq Modell mit den Top 10 Hashtags, Top 100 Hashtags sowie dem großen Datensatz trainiert. Wir betrachten also nun beispielhaft die Vorhersagen der einzelnen Modelle. Alle Beispiele sind im Anhang A.2 - A.4 aufzufinden.

Bei den Top 10 Hashtags blieb aufgrund der geringen Anzahl an Labels kein großer Spielraum für neue Kreationen von Hashtags und es wurde meistens ein Hashtag erzeugt, der einem Label entsprach. Jedoch kam es unter anderem zu Kombination von zwei Hashtags aus der Labelmenge. In den Labels sind die Hashtags *#trump* und *#notmypresident* enthalten. Das Netz lieferte mehrere Male für Tweets, die von dem Benutzer mit *#trump* markiert wurden, den selbst erzeugten Hashtag *#trumppresident* zurück. Oft wurden auch unsinnige Kombinationen erzeugt, wie aus den Hashtags *#travel* und *#thanksgiving* der Hashtag *#travelgiving*. Außerdem war das Netz in der Lage mehrere Hashtags für einen Tweet richtig vorherzusagen. Dies wurde durch Aneinanderreihung aller Hashtags, mit denen ein Tweet markiert wurde, ermöglicht. Die Hashtags eines Labels wurden mit einem # Symbol getrennt. Damit war das Netz in der Lage zu lernen, dass durch Ausgabe des # Symbols ein neuer Hashtag hinzugefügt werden kann.

Diese Ergebnisse konnten auch bei der Auswertung der Top 100 Hashtags vermehrt und in besserer Qualität gefunden werden. Weiterhin war es dem Netz auf den Top 100 Hashtag Daten möglich, neue Hashtags zu generieren und Hashtags für Tweets vorzuschlagen, die teilweise passender sind, als die vom Benutzer verwendeten Hashtags. Der Tweet *"black friday can't pass this. pripaso dash cam for car safety and easy to install."* beispielsweise, wurde vom Nutzer mit dem Hashtag *#ad* markiert. Dieser ist zwar durchaus sinnvoll, da es sich jedoch um ein konkretes Angebot bezüglich des Black Fridays handelt, kann der Hashtag *#blackfriday*, welcher vom Neuronalen Netz vorgeschlagen wurde, durchaus als die besserer Wahl gesehen werden. Ein neuer Hashtag *#neat*, der nicht in den Top 100 Trainingsdaten vorkam, wurde von dem Netz entworfen und mehrmals verwendet. Jedoch war der Hashtag nicht geeignet, um die Themen der Tweets zu erfassen. Außerdem ist bei den Vorhersagen des Netzes zu sehen, dass dieses die Struktur von englischen Wörtern

gelernt hat. Auch wenn die Hashtags oft keine konkreten Wörter sind, so können sie doch gelesen werden und sind keine zufällige Konkatenation von Buchstaben. Beispiele hierfür sind die Hashtags *#fosiness* und *#bananan*, die vom Netz kreiert wurden.

Bei dem Auswerten des großen Datensatzes ist als größte Differenz zu den bisherigen Vorhersagen, das häufige Auftreten von Zeichenketten, die weit von einem zur Vorhersage geeigneten Begriff entfernt sind, aufgefallen. Während bei den Top 10 und Top 100 Hashtags fast jeder Vorschlag ein existierender Begriff oder einem solchen sehr ähnlich war, wurde bei der Auswertung des großen Datensatzes vermehrt Zeichenketten wie *#taaees* oder *#saaeea* vorgefunden. Außerdem entschloss sich das Modell auch ab und zu keinen Hashtag, also als erstes Zeichen *EOF* auszugeben. Dies kann auf die deutlich höhere Anzahl an Hashtags, mit denen das Netz trainiert wurde, zurückgeführt werden. Im Gegensatz zu den bisher gezeigten Beispielen, repräsentieren die folgenden nur die besten Ergebnisse. Oft wurden längere Hashtags abgekürzt. Teilweise wurde nur ein Buchstabe z.B. *#photograph* anstatt *#photography* weggelassen, aber auch Abkürzungen wie *#cfa* anstatt *#cfapeachbowl* kamen vor. Ein Tweet bezüglich eines Sportereignisses, bei dem Detroit (DET) gegen Minnesota (MIN) antrat, wurde vom Nutzer mit *#DETvsMIN* markiert. Unser Modell hat hier die Mannschaften vertauscht und für denselben Tweet den Hashtag *#MINvsDET* vorgeschlagen, was auf eine gute Tweet Repräsentation hinweist. Dies zeigen auch Beispiele wie *"I just voted for The Walking Dead to win favorite TV Show at People's Choice Awards 2017! Cast your vote: !url "*, welche ein sehr eindeutiges Thema haben und es somit dem Netz durch eine gute Repräsentation erleichtern, den richtigen Hashtags *#23pcas* vorzuschlagen. Ein weiteres Beispiel bezüglich der Tweet Repräsentation bietet der Tweet *"Noora I'm just as perplexed as you are."*, der von unserem Modell mit *#thenicebot* markiert wurde, obwohl der ursprüngliche Hashtag *#skam* lautet. Denn Tweets die normalerweise mit *#thenicebot* markiert werden, beginnen mit einem Namen, wie hier Noora, und enden mit einem Kompliment an die genannte Person. Obwohl bei diesem Beispiel nicht direkt ein Kompliment gemacht wurde, so ist doch die genannte Person direkt angesprochen worden. Dieses Muster hat das Netz erkannt und für seine Vorhersage ausgenutzt. Bei Tweets die mit dem Hashtag *#illuminateworldtour* markiert wurden, wurden die Namen der Städte, auf die sich der Tweet bezog, an das Ende des jeweiligen Hashtags gehängt. Das Netz versuchte nicht für jeden Tweet die richtige Stadt zu erraten, sondern vereinheitlichte alle Hashtags zu diesem Thema und bildete den Hashtag *#illuminateworldtourmanten*. Es gibt nun zwei Möglichkeiten dies zu interpretieren. Eine davon ist, dass die durchschnittlichen Zeichenfolge, die sich aus allen Städten zusammensetzt, das Wort *manten* ergibt. Da unser Modell jedoch genau für den Zweck Wortstrukturen zu lernen gebaut wurde, könnte dies auch ein Versuch sein, das Wort *tournament* zu bilden, um somit eine Verallgemeinerung des ursprünglichen Hashtags zu erreichen. Unser Modell war außerdem in der Lage, zwischen verschiedenen Verkehrsmeldungen zu unterscheiden. Verkehrsmeldungen werden je nach Inhalt mit einem bestimmten Hashtag markiert. Dieses System hat das Modell versucht zu emulieren, indem es oft die richtigen Buchstaben, jedoch meistens die falschen Zahlen vorschlug. Beispiele hierfür sind die richtigen Hashtags *#m7bus* und *#i195*, für die das Netz die Hashtags *#m24bus* und *#i82* entsprechend vorschlug.

5.5.2. Allgemeine Analyse aller Vorhersagen

In diesem Abschnitt betrachten wir allgemeine Analysen bezüglich der Vorhersagen des Tweet2Seq Modell.

Als erstes vergleichen wir, wie viele neue Hashtags das Modell kreiert hat. Dabei heißt neu, dass der Hashtag nicht als Label in den Trainingsdaten vorhanden ist. Neu heißt aber auch, dass Ausgaben wie *#taaaaeeen* als neuer Hashtag gelten, da diese offensichtlich nicht in der Menge der Label vorkommen. Von der Neuheit des Hashtags darf also nicht auf die Güte der Vorhersage geschlossen werden.

Datensatz	Ref. in Test	% Test nicht in Train	Hyp. nicht in Train	% insgesamt neu
Großer	19197	16.81	34520	91.84
Top 100 #	96	0	5553	43.68
Top 10 #	10	0	1727	13.63

Tabelle 5.12.: Vergleich von drei Tweet2Seq Modellen, die auf jeweils unterschiedlichen Datensätzen trainiert wurden. Erläuterungen zu den einzelnen Spalten sind im folgenden aufgeführt.

In Tabelle 5.12 sehen wir vier verschiedene Spalten mit Abkürzungen, deren Bedeutungen wir kurz erläutern werden. Die erste Spalte steht für die Anzahl an Referenzen im Testdatensatz. Es wird also gezählt, aus wie viel verschiedenen Hashtags sich die Labelmenge des Testdatensatzes zusammensetzt, bzw. wie viele Hashtags aus der Menge aller möglichen Hashtags in den Testdaten vorkommen. Die zweite Spalte gibt prozentual an, wie viele der Referenzen des Testdatensatzes nicht als Referenz im Trainingsdatensatz auftauchen. Mathematisch ausgedrückt, stellt die zweite Spalte die Anzahl an Tweets im Testdatensatz, deren Referenz $r \in REF_{test}$ in der Menge $REF_{test} \backslash REF_{train}$ enthalten ist, im Verhältnis zu der Gesamt Anzahl an Tweets im Testdatensatz dar. Dabei ist REF_{test} die Menge aller verschiedenen Hashtags, die in den Testdaten vorkommen und REF_{train} die Menge aller verschiedenen Hashtags, die in den Trainingsdaten vorkommen. In der dritten Spalte sehen wir die Anzahl an verschiedenen Hypothesen, sprich Vorhersagen für Tweets aus dem Testdatensatz, die nicht in den Referenzen des Trainingsdatensatzes vorkommen und somit neu von dem Modell kreiert wurden, bzw. $|HYP_{test} \backslash REF_{train}|$, wobei HYP_{test} die Menge aller verschiedener Vorhersagen des Testdatensatzes ist. Da das Modell seine neuen Hashtag Entwürfe durchaus auch öfter verwenden kann, wird in der letzten Spalte dargestellt, welcher Anteil aller Vorhersagen des Testdatensatzes, aus neuen Hashtags besteht. Durch Vergrößerung der Label Menge wird es dem Netz also nicht nur ermöglicht, einen größeren Anteil an neuen Hashtags zu kreieren, sondern auch eine größere Variation in denselben hervorzubringen. Jedoch ist zu beachten, dass durch die erhöhte Freiheit des Netzes, Hashtags selbständig zu bilden, auch die Wahrscheinlichkeit für unsinnige Ausgaben ansteigt. Während für die Top 10 und Top 100 ein Großteil aller neuen Hashtags, Wörter der englischen Sprache waren, oder jedenfalls eine erkennbare Wortstruktur vorweisen konnten, wurden bei dem großen Datensatz vermehrt zufällige Zeichenkombinationen als Ausgabe vorgefunden. Hierbei ist aber auch zu berücksichtigen, dass weder die Top 10, noch die Top 100 Hashtags unbekannte Referenzen in ihren Testdatensätzen

enthalten, während bei dem Großen Datensatz mehr als 15% der Testdaten mit einem Hashtag markiert sind, der während dem Training noch nie gesehen wurde.

Weiterhin haben wir ausgewertet, wie oft die verschiedenen Modelle, bereits durch die Label bekannte Hashtags, reproduziert haben. Dazu wurde für das jeweilige Modell die Anzahl an bekannten Hashtags, zu der Gesamtanzahl an Vorhersagen, sprich der Größe des jeweiligen Testdatensatzes, ins Verhältnis gesetzt. Man kann in Tabelle 5.13 sehen,

Datensatz	# aus Top 100	# aus Top 10
Großer	5.37	2.57
Top 100 #	56.32	15.74
Top 10 #	-	86.37

Tabelle 5.13.: Vorhersagen die einem bekannten Label aus den Top 100 bzw. Top 10 Hashtags entsprechen im prozentualen Verhältnis zu der Gesamtanzahl an Vorhersagen.

dass die Vorhersagen des Modells, welches auf den Top 10 Hashtags trainiert wurde, fast einer klassischen Klassifikation gleichkommen, bei der bekannterweise nur Hashtags aus der Label Menge verwendet werden können. Bei dem Modell, dass auf den Top 100 Hashtags trainiert wurde, ist nur noch knapp über die Hälfte der Vorhersagen bereits bekannt und der Anteil an Eigenkreationen des Netzes schon deutlich höher. Eine fast völlig von den bekannten Hashtags unabhängige Vorhersage bringt das Modell, das auf dem großen Datensatz trainiert wurde. Da jedoch sicherlich eine große Menge an brauchbaren Hashtags in dem großen Datensatz enthalten ist und es nicht von Nöten ist mehr als 90% aller Label neu zu erfinden, wenn geeignete Hashtags schon existieren, sollte eine Lösung angestrebt werden, die einen Ausgleich zwischen Verwendung von bereits bekannten Hashtags und der Erfindung neuer Hashtags schafft.

5.6. Gesamtvergleich

In diesem Abschnitt wollen wir alle vorgestellten Architekturen untereinander vergleichen.

Wir haben hier von jedem Modell das beste Ergebnis genommen und kommen zu dem Schluss, dass das LSTM am besten für die Hashtag Vorhersage geeignet ist. Jedoch ist es für einfache Probleme wie die Vorhersage der Top 10 Hashtags schon zu komplex und muss dort dem CNN den Vortritt überlassen. Wie zu erwarten, schneidet dass Tweet2Seq Modell schlechter ab als die anderen Modelle, wenn auf exakte Übereinstimmung mit dem Label überprüft wird. Dafür, dass das Erzeugen von Hashtags Zeichen für Zeichen, ein deutlich schwereres Problem darstellt, ist der Unterschied zwischen den Modellen für die Top 10 Hashtags jedoch nicht besonders signifikant. Für die Top 100 Hashtags beträgt die Differenz zum besten klassischen Klassifikationsmodell 14,14%, wobei sich der Unterschied von 2,56% zum schlechtesten Modell auch hier in Grenzen hält, wie in Tabelle 5.14 zu sehen ist.

Bisher wurden nur die Ergebnisse des "besten"Vorschlags, mit der höchsten Eintrittswahrscheinlichkeit betrachtet. Da es bei den Top 100 Hashtags aber durchaus möglich ist, dass mehr als ein Hashtag aus der Menge von Labeln eine sinnvolle Klassifikation für den

One-Error in %	MLP	CNN	LSTM	Tweet2Seq
Top 10 #	24.60	23.59	25.52	27.53
Top 100 #	55.75	50.61	44.17	58.31

Tabelle 5.14.: Beste Ergebnisse der verschiedenen Modelle

Eingabe Tweet darstellt, vergleichen wir noch das CNN mit dem LSTM bezüglich des Five-Error Fehlermaßes. Dieser Vergleich wurde nur auf den Top 100 Hashtags durchgeführt, da bei den Top 10 Hashtags schon eine 50% Chance für eine richtige Klassifikation gegeben ist, wenn die Ausgabe eine zufällige Wahrscheinlichkeitsverteilung über die Labelmenge ist. Außerdem ist es durch die Verschiedenheit der Top 10 Hashtags sehr unwahrscheinlich, dass mehrere Hashtags für ein Thema eines Tweets in Betracht kommen. Das Embedding hat für alle Modelle eine Größe von 256 und die Anzahl an Parametern ist für alle Modelle gleich. Neben dem CNN haben wir bei dem LSTM nochmals zwischen Average-Pooling und dem klassischen LSTM unterschieden.

Five-Error in %		
CNN	LSTM Avg	LSTM
34.40	27.23	27.68

Tabelle 5.15.: CNN und LSTM im Vergleich für die Top 100 Hashtags

Tabelle 5.15 zeigt, dass auch hier die LSTM Modelle deutlich bessere Ergebnisse liefern konnten als das CNN. Die Differenz zwischen Average-Pooling und klassischer Ausgabe bei der LSTM Architektur fiel jedoch, wie auch schon beim One-Error, sehr gering aus. Für die Hashtag Vorhersage können wir also keinen Nutzen aus der Variation des Standard LSTMs ziehen.

Zum Abschluss vergleichen wir in Tabelle 5.16 die verschiedenen Modelle auf Testdatensätzen, bei denen keine Überschneidung mit den Trainingsdaten vorliegt. Wir verwenden dieselben Modelle, die für die Auswertung in Tabelle 5.14 verwendet wurden. Für die

One-Error in %	MLP	CNN	LSTM	Tweet2Seq
Top 10 #	22.03	29.87	31.11	26.14
Top 100 #	65.07	62.29	54.77	68.79

Tabelle 5.16.: Beste Ergebnisse der verschiedenen Modelle mit 0% Überschneidung der Test- mit den Trainingsdaten

Top 100 Hashtags haben sich die Ergebnisse an die zusätzliche Schwierigkeit der fehlende Überschneidung angepasst. Für diese gab es eine Überschneidung von 28,59% in den Testdaten. Das Herausfiltern dieser Überschneidungen führte zu einer Erhöhung der Fehlerrate von durchschnittlich ungefähr 18%. Dahingegen erzielte das MLP auf den Top 10 Hashtags, bei denen eine Überschneidung von 25,72% vorliegt, sogar bessere Ergebnisse, obwohl keine bekannten Beispiele in den Testdaten vorhanden waren. Dies lässt sich auf dessen geringe Komplexität zurückführen, durch die dieses nicht in der Lage ist, die

Trainingsdaten auswendig zu lernen. Damit geht auch der Vorteil einer Überschneidung der Trainings- und Testdaten, der durch den Overfitting Effekt entsteht, verloren. Diese Unfähigkeit des MLPs, die Trainingsdaten auswendig zu lernen, führt in diesem speziellen Fall sogar zu einer besseren Verallgemeinerung. Eine leichte Verbesserung kann auch bei dem Tweet2Seq Modell vorgefunden werden. Dies könnte daran liegen, dass die Tweets, die bei dem Datensatz ohne Überschneidung herausgefiltert wurden, zufälligerweise im Durchschnitt mit längeren Hashtags markiert wurden. Denn die Wahrscheinlichkeit, dass bei der Zeichenweise Vorhersage ein Fehler gemacht wird, steigt mit jedem Zeichen, das korrekt prädiziert werden muss.

6. Zusammenfassung

In dieser Arbeit haben wir verschiedene neuronale Netze Ansätze verwendet um Hashtags für Tweets vorherzusagen. Dabei haben wir an einem MLP Regularisierungs- und Optimierungsmethoden getestet und sind zu dem Ergebnis gekommen, dass durch die Kombination von bewährten Methoden signifikante Verbesserungen erzielt werden können. Die größte positive Auswirkung hatten jedoch diejenigen Verfahren, die den Gradientenabstieg optimieren. Für die Klassifikation der Top 10 Hashtags erzielte das CNN Modell die besten Ergebnisse. Bei der Klassifikation der Top 100 Hashtags konnte das CNN jedoch nicht mit der Bi-LSTM Architektur mithalten. Dieses konnte durch die sequentielle Eingabe der Tweets eine bessere Tweet Repräsentation generieren, als das CNN durch Merkmalsextraktion. Die neue Methode, Hashtags mit dem Tweet2Seq Modell Zeichen für Zeichen zu erzeugen, konnte wie erwartet, nicht so gute Ergebnisse für die exakte Übereinstimmung der Hypothese mit der Referenz liefern, wie die klassischen Klassifikationsmodelle. Jedoch war die Differenz zwischen den Modellen eher marginal. Das Tweet2Seq Modell war in der Lage, neue Hashtags zu kreieren und bekannte Hashtags miteinander zu kombinieren und somit gelegentlich bessere Vorschläge zu liefern, als der Verfasser des Tweets selbst. Die Steigerung der Komplexität des Tweet2Seq Modells, durch Hinzunahme einer zusätzlichen CNN Schicht, konnte die Ergebnisse nicht verbessern. Jedoch konnten wir durch Erhöhung der Parameterzahl, bei dem klassischen Seq2Seq Modell und dem Tweet2Seq Modell verbesserte Ergebnisse feststellen.

6.1. Ausblick

Wegen der Anzahl an verschiedenen Modellen, die getestet wurden, und aus zeitlichen Gründen war es nicht möglich, alle Hyperparameter zu optimieren. Für die besten Modelle, könnte in Zukunft die Auswirkung einer anderen BoW Größe auf die Repräsentation der Tweets betrachtet werden. Es ist auch denkbar, die Tweets auf einer Zeichenlevel Basis, anstatt auf einer Wortlevel Basis zu repräsentieren und mit den Ergebnissen von [WCA14] zu vergleichen. Die wegen der limitierten Hardware begrenzte Parameteranzahl, könnte erhöht werden und vor allem der Performanz des Tweet2Seq Modell zu gute kommen. In dieser Arbeit haben wir uns hauptsächlich mit der Optimierung der Tweet Repräsentation beschäftigt. Daher wäre es interessant, die Auswirkung von Optimierungen bezüglich des Decoders, wie z.B. die Verwendung von Beamsearch [WR16], zu betrachten. Schließlich kann noch die Auswahl der Trainingsdaten bzw. die Anzahl an unterschiedlichen Hashtag Labels optimiert werden, sodass die Anzahl an Hashtags groß genug ist, um alle Themen abzudecken und klein genug, um feste Strukturen im Sequenz Decoder zu lernen, anstatt diesen mit zu viel Varianz in den Hashtags zu überfordern.

Literatur

[Bot91] Léon Bottou. *Stochastic Gradient Learning in Neural Networks*. Nimes, France, 1991.

[Col+11] Ronan Collobert u. a. *Natural Language Processing (almost) from Scratch*. 2011.

[Dhi+16] Bhuwan Dhingra u. a. "Tweet2Vec: Character-Based Distributed Representations for Social Media". In: *ACL* (2016).

[DN15] Roman Dovgopol und Matt Nohelty. "Twitter Hash Tag Recommendation". In: *CoRR* abs/1502.00094 (2015). URL: http://arxiv.org/abs/1502.00094.

[Elm90] Jeffrey L. Elman. "Finding Structure in Time". In: *Cognitive Science* 14.2 (1990), S. 179–211.

[God+13] Fréderic Godin u. a. "Using Topic Models for Twitter Hashtag Recommendation". In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW '13 Companion. Rio de Janeiro, Brazil, 2013, S. 593–596.

[GSC00] Felix A. Gers, Jürgen Schmidhuber und Fred A. Cummins. "Learning to Forget: Continual Prediction with LSTM". In: *Neural Computation* 12 (2000), S. 2451–2471.

[HS97] Sepp Hochreiter und Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (1997), S. 1735–1780.

[JZ15] Rie Johnson und Tong Zhang. "Effective Use of Word Order for Text Categorization with Convolutional Neural Networks". In: *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL* (2015), S. 103–112.

[KB14] Diederik P. Kingma und Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). URL: http://arxiv.org/abs/1412.6980.

[KCK12] Elham Khabiri, James Caverlee und Krishna Y. Kamath. "Predicting Semantic Annotations on the Real-time Web". In: *Proceedings of the 23rd ACM Conference on Hypertext and Social Media*. HT '12. Milwaukee, Wisconsin, USA, 2012, S. 219–228.

[KGB14] Nal Kalchbrenner, Edward Grefenstette und Phil Blunsom. "A Convolutional Neural Network for Modelling Sentences". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, Volume 1: Long Papers*. Baltimore, MD, USA, 2014, S. 655–665. URL: http://aclweb.org/anthology/P/P14/P14-1062.pdf.

[Kyw+12] Su Mon Kywe u. a. "On Recommending Hashtags in Twitter Networks". In: *Social Informatics: 4th International Conference, SocInfo 2012. Proceedings* (2012).

[Li+16] Yang Li u. a. “Hashtag Recommendation with Topical Attention-Based LSTM”. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, 2016, S. 3019–3029.

[LWZ11] Tianxi Li, Yu Wu und Yu Zhang. *Twitter Hash Tag Prediction Algorithm*. 2011.

[Pap+02] Kishore Papineni u. a. “BLEU: a Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2002, S. 311–318.

[SVL14] Ilya Sutskever, Oriol Vinyals und Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *CoRR* abs/1409.3215 (2014). URL: http://arxiv.org/abs/1409.3215.

[The16] Theano Development Team. “Theano: A Python framework for fast computation of mathematical expressions”. In: *arXiv e-prints* abs/1605.02688 (2016). URL: http://arxiv.org/abs/1605.02688.

[TQL15] Duyu Tang, Bing Qin und Ting Liu. “Document Modeling with Gated Recurrent Neural Network for Sentiment Classification”. In: *EMNLP* (2015).

[Wai+89] Alexander Waibel u. a. “Phoneme Recognition Using Time-Delay Neural Networks”. In: *IEEE Transactions on Acoustics, Speech and Signal Processing* 37, No. 3 (1989), S. 328–339.

[Wai16] Alexander Waibel. “Maschinelles Lernen 1 & 2”. Kognitive Systeme Vorlesung, KIT. 2016.

[WCA14] Jason Weston, Sumit Chopra und Keith Adams. “#TAGSPACE: Semantic Embeddings from Hashtags”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), S. 1822–1827.

[WR16] Sam Wiseman und Alexander M. Rush. “Sequence-to-Sequence Learning as Beam-Search Optimization”. In: *CoRR* abs/1606.02960 (2016). URL: http://arxiv.org/abs/1606.02960.

[WZ89] Ronald J. Williams und David Zipser. “A Learning Algorithm for Continually Running Fully Recurrent Neural Networks”. In: *Neural Comput.* 1.2 (1989), S. 270–280.

[ZF13] Matthew D. Zeiler und Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *CoRR* abs/1311.2901 (2013). URL: http://arxiv.org/abs/1311.2901.

[ZGS11] Eva Zangerle, Wolfgang Gassler und Günther Specht. “Recommending #-Tags in Twitter”. In: *Proceedings of the 2nd International Workshop on Semantic Adaptive Social Web (SASWeb 2011)* (2011).

[ZZ05] Min-ling Zhang und Zhi-hua Zhou. *Multi-Label Neural Networks with Applications to Functional Genomics and Text Categorization*. 2005.

A. Anhang

A.1. Schwer klassifizierbare Tweets

Einige Beispiele für Tweets, deren Hashtags auch für Menschen nicht vorhersagbar sind, da die Hashtags den Kontext erst erzeugen oder dem vermuteten Kontext nicht gerecht werden.

Tweets	Hashtags
Going in...	#dinnertime
I find your lack of faith disturbing	#StarWars #DarthVader #grotesque #NationalCathedral
I do love	#Watchdog
Because I never actually left	#WhyIDontGoToTheBar
You know when you stare at word long enough, it starts to look really ridiculous? Today that word is opinion.	#thoughtswhilegradingpapers
eating is so much easier	#DatingIsHardBecause
Ever notice that we all look like a bunch of marbles?	#HowPlanetsTalkAboutEachOther
They CAN do it! Learning to navigate and map across an archaeological site!	#autism #Archaeology #autismawareness
It's that time!	#wineoclock

A.2. Tweet2Seq Vorhersagen: Top 10 Hashtags

Auswertungen der Vorhersagen des Tweet2Seq Modells für die Top 10 Hashtags. In der ersten Spalte befindet sich der Tweet, in der zweiten der vom Benutzer verwendete Hashtag und in der dritten Spalte die Vorhersage des Modells. Der Top 10 Hashtags Datensatz

enthält folgende Labels: *#cute, #iphone, #love, #nba, #notmypresident, #selfie, #thanksgiving, #thxbirthcontrol, #travel, #trump*

Tweet	Benutzer	Tweet2Seq
@user when will you apologize??	#trump	#trumppresident
A WWE hair match champion who took up politics as a hobby last year, beat a professional of 40 years and is now president.	#trump	#trumppresident
@user u didn't support now you do? And you support trump & bannon sessions?? really? How are they possible? Did u sell ur soul?	#trump	#trumppresident
The latest travel. The treasury of arts! Thanks to @user	#travel	#travelgiving
Donald Trump iPhone case 4!	#iphone #love	#iphone #love
Enjoy another sexy coed! Like & RT this hot stuff!	#cute #love	#cute #love
When the first Montreal meal also happens to be a cheflebrity sighting. Rachel ray why u photobomb my #selfie 😊	#selfie	#note

A.3. Tweet2Seq Vorhersagen: Top 100 Hashtags

Auswertungen der Vorhersagen des Tweet2Seq Modells für die Top 100 Hashtags. In der ersten Spalte befindet sich der Tweet, in der zweiten der vom Benutzer verwendete Hashtag und in der dritten Spalte die Vorhersage des Modells.

Tweet	Benutzer	Tweet2Seq
I'm team Aubie, WarEagle! Which mascot did you vote for in the Rivalry Week showdown?	#wareagle #cfapeachbowl	#wareagle #cfapeachbowl
Listen to PARTYNEXTDOOR CANDY Feat. NIPSEY. by partyomo np on SoundCloud	#np #soundcloud	#np #soundcloud
I entered for my chance to win a $250 Macy's gift card from @user!	#win #giveaway	#win #giveaway
Black Friday can 't pass this. Pripaso dash cam for car safety and easy to install.	#ad	#blackfriday
Even though I feel [name feeling], I am deeply grateful for the animals in my life.	#thanksgiving	#thankful
3 Social Media Marketing Mistakes You Might Be Making	#marketing	#socialmedia
Not only can he make motivation Music, this boy can drive in #SanDiego uber …	#sandiego	#neat
#DatingIsHardBecause nobody actually wants to date anymore.	#datingishardbecause	#neat
TV journos sill bruised after #Trump meet...	#trump	#neat
We're hiring! Click to apply: Desktop Support Analyst - !url	#sacramento #hiring #it	#bananana
As we looking this new transition!	#newyork #humanrights #together	#fosiness

A.4. Tweet2Seq Vorhersagen: Großer Datensatz

Auswertungen der Vorhersagen des Tweet2Seq Modells für alle zu Verfügung stehenden Hashtags. In der ersten Spalte befindet sich der Tweet, in der zweiten der vom Benutzer verwendete Hashtag und in der dritten Spalte die Vorhersage des Modells. In den folgenden

Beispielen wurden teilweise Ausgaben des Modells sowie auch die Hashtag Label der Benutzer groß geschrieben. Dies dient nur zum einfacheren Verständnis, und stellt nicht die exakte Ausgabe der Netzarchitektur dar.

Tweet	Benutzer	Tweet2Seq
In the colors of the Caribbean by Fruteo !url	#photography	#photograph
I voted for NC State's unis, Go Pack! Which uniform did you vote for in the @user rivals poll?	#cfapeachbowl	#cfa
ABCFsecretlife You have got me walking on air, without even using my hover mode.	#thenicebot	#thenices
Aretha Franklin sounds like a dying dog right now. Way to ruin the national anthem.	#DETvsMIN	#MINvsDET
My family home happy	#Whatiwantfor christmasin4words	#Thatiwantfor christmasin4words
#IAmThankfulFor the friends and family i have. they are all so so amazing and my life would be nothing without them. you know who you are.	#iamthankfulfor	#iamthankfulfor
I just voted for The Walking Dead to win favorite TV Show at People's Choice Awards 2017! Cast your vote: !url	#23pcas	#23pcas
Noora I'm just as perplexed as you are.	#skam	#thenicebot
miguelitomusic9 You are sweeter than honey.	#thenicebot	#thenicebot
kendallove You brighten my day.	#thenicebot	#thenicebot
I want to see @user at an #IlluminateWorldTourBoston ! Have your say at !url	#illuminateworldtourboston	#illuminateworldtourmanten

I want to see @user at an #IlluminateWorldTourPorto ! Have your say at !url	#illuminateworldtourporto	#illuminateworldtourmanten
My mom drank two glasses of wine and now I have to make the special k bars.	#canthang	#iamthankfulfor
Don't do it	#ItsATrap	#dubchallenge
I just checked in @user with #mPLACES to donate to @user With mPOINTS, you can too: !url	#mplaces	#mplusplaces
Im Michonne. Discover which #TheWalkingDead character you are! START QUIZ: !url	#thewalkingdead	#poddoftherings
Im Hulk. Discover which superhero you are! SIGN IN: !url	#superhero	#disney
Im Bowser. Discover which Supermario character You Are! !url	#supermario	#supermario
Win Enter to win a NEW #AppleWatchSeries2 !url	#applewatchseries2	#giveaway
Update: Incident on both directions at Broadway / Columbus Av between 57st and 70st	#m7bus	#m24bus
Incident on EB from exit 2-CR524 to exit 7-CR526!	#i195	#i82