der Bundeswehr
Universität München

Einsatz von Sprachmodellen zu Stance Detection und Sentiment Analysis

Lina Suttrup
1120810

Betreuung:
Prof. Dr. Oswald

Institut für Verteilte Intelligente Systeme
Universität der Bundeswehr München
Fakultät für Elektrotechnik und Technische Informatik

Inhaltsverzeichnis

Bestätigung

Hiermit versichere ich, dass die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden.

Ferner habe ich vom Merkblatt über die Verwendung von Abschlussarbeiten Kenntnis genommen und räume das einfache Nutzungsrecht an meiner Arbeit der Universität der Bundeswehr München ein.

Neubiberg, den 31.08.2020

Lina Suttrup

1. Einleitung

Was wäre, wenn Computer von ihren Erfahrungen lernen könnten, so wie Menschen es tun? Eine Frage, aus der viele Forschungsfelder entsprungen sind. **Künstliche Intelligenz** ist ein großes Themengebiet, aber ohne **Machine Learning** sind Computer nicht „schlauer“ als die Programme, die sie ausführen - und die von Menschen programmiert werden. Zwar kann ein Computer ein Problem mit vorgegebenen Lösungsweg meist weit schneller lösen als der Mensch selber, aber wie kommt er auf diesen Lösungsweg? **Machine Learning** und **Deep Learning** konzentrieren sich auf genau diese Problemstellung.
Seit der Entwicklung erster **Machine Learning** Algorithmen hat dieses Feld immense Fortschritte gemacht, besonders in den letzten Jahren, in denen auch **Deep Learning** durch die Entwicklung immer schnellerer Prozessoren sowie der Möglichkeiten der modernen GPUs zur Parallelisierung von Aufgaben immer weiter in den Vordergrund rücken konnte. Machine Learning und Deep Learning ist heute schon weit verbreitet und wird in den verschiedensten Feldern eingesetzt - von E-Mail Spam Klassifizierung bis hin zur Erkennung von Tumoren auf Röntgenbildern, es bleibt aber weiterhin ein großes Forschungsfeld.

1.1 Natural Language Processing

Ein wichtiger Teil der Künstlichen Intelligenz ist die **Computerlinguistik**, oder auch **Natural Language Processing (NLP)**. Menschliche Sprache ist für Maschinen nur schwer verständlich, da sie im Gegensatz zu Maschinensprachen keiner genauen Logik folgen: Sie ist im Grunde unvorhersehbar und führt selbst bei Menschen zu Missverständnissen. Trotzdem gibt es heutzutage viele Anwendungen von **NLP** im Alltag, und Beispiele wie die heutigen Sprachassistenten zeigen, wie weit dieses Feld in einer sehr kurzen Zeit voranschreiten konnte.
Aufgrund der Komplexität des Problems gibt es meherer Herangehensweisen, um natürliche

Sprache für Maschinen verständlich zu machen. Ihnen allen ist gemein, dass sie Sprache durch Zahlen repräsentieren.
Es gibt Ansätze, die Wörter als einzelne Einheiten nehmen, und diesen einfach eine Zahl zuordnen. Dies hat den Vorteil, dass es leicht umzusetzen ist und auch keine hohe Anzahl an Trainingsdaten braucht. Das Problem daran ist, dass so sehr viel Bedeutung verloren gehen kann - in natürlicher Sprache sind häufig die umgebenden Wörter genauso wichtig wie das betrachtete Wort. Kontext und Wortzusammenhänge gehen durch diesen Ansatz verloren.
Eine weitere bekannte Technik ist der Einsatz von Wortverktoren, die durch ihre hohe Dimensionalität viele Beziehungen zwischen Wörtern bewahren können. Diese Repräsentation ist vielversprechend, aber auch hier kann nicht immer der Kontext gewahrt werden, da einem Wort genau ein Vektor zugeordnet wird.
Eine Verbesserung, nicht nur in diesem Aspekt, bringen **Sprachmodelle**. Sie können vielfältig eingesetzt werden, haben einen großen Wortschatz in verschiedenen Sprachen und sind vortrainiert - sie müssen entsprechend nur der Aufgabe angepasst werden, was einer enormen Zeitersparnis entspricht. Durch Feinabstimmung der verschiedenen Parameter kann bei Sprachmodellen auch viel erreicht werden.

1.2 Sentiment Analysis

Sentiment Analysis ist die Einordnung von Text in Kategorien von Emotionen, wobei es verschiedene Skalen gibt. Die einfachste ist hierbei die Einordnung in „Neutral“, „Positiv“ und „Negativ“, aber es gibt auch Zuordnungen zu Werten von -1 bis 1 oder anderen Zahlenwerten. Diese Werte werden nicht immer mit einheitlicher Bedeutung genutzt.

1.3 Stance detection

Bei **Stance Detection** geht es darum, herauszufinden, wie die Haltung des Autors eines Textes zu einer bestimmten These oder einem allgemeinen Begriff ist. Beste Ergebnisse wurden durch Unterteilung der Aufgabe in zwei Schritte erzielt; Zunächst wird **Topic Modeling** genutzt, um herauszufinden, ob der Text überhaupt mit der Stance in Verbindung gebracht werden kann. Daruafhin wird, falls dies der Fall ist, auf die Stance bezogene **Sentiment Analysis** genutzt, um die Haltung des Autors zu ermitteln.

1.4 Zielsetzung und Vorgehen

Ziel dieser Arbeit ist es, zu erforschen wie gut sich verschiedene **Sprachmodelle** zum Einsatz in Sentiment Analysis und Stance Detection eignen. Dazu werden die verschiedenen Modelle vorgestellt und analysiert, welche für den Einsatz in Frage kommen, da die Modelle durch unterschiedlichen Aufbau oder auch verfügbare Datensätze nicht alle gleich gute Herangehensweisen an die Problemstellung darstellen. Für die Versuche werden mehrere **Corpora** verwendet, von denen die meisten öffentlich zugänglich sind. Alle **Corpora** bestehen nur aus Tweets, die sich aufgrund der limitierten Zeichenlänge und der Popularität der Plattform *Twitter* gut für die Auswertung in beiden Aufgaben eignen. Die Ergebnisse werden ausgewertet und miteinander verglichen.

2. Sprachmodelle

Bis zur Einführung von Sprachmodellen gab es für NLP in Machine Learning ein großes Problem: Textdaten, wie auch Sprache, sind inherent sequentiell. Im Gegensatz zu Bildern, bei denen alle Daten auf einmal verarbeitet werden können, konnte die Textverarbeitung die Beschleunigung durch den Einsatz von GPUs nicht nutzen, da viele Aufgaben nicht parallelisiert werden konnten [2].
Weiterhin gab es keine Möglichkeit, ein schon vortrainiertes Modell für andere Aufgaben weiter zu verwenden, da das Training für die verschiedenen Aufgaben und auch Domänen sehr spezifisch angegangen werden muss. Diese Faktoren führten zu einem deutlichen Mehraufwand von NLP z.B. gegenüber Bildverarbeitung.
Das Ziel von Sprachmodellen ist es, parallele Verarbeitung der Daten möglich zu machen sowie, im Gegensatz zu vortrainierten **Wortvektor Embeddings** wie **GLoVe** den Kontext der einzelnen Wörter im Satz zu wahren. In den Embeddings wird jedem Wort ein Vektor zugewiesen, sodass ähnliche Wörter nah beieinander liegen und die Beziehung zwischen Wörtern erhalten bleibt. Dadurch können interessante Rechnungen aufgestellt werden:

```
London - England + Spanien = Madrid
```

Mit ganzen Textsequenzen können aber auch diese Vektor Embeddings aufgrund der fehlenden Kontextinformationen nur bedingt gut umgehen.

2.1 Vergleich zu anderen Ansätzen

Wie schon erwähnt muss Sprache sequentiell versarbeitet werden. Dazu werden meist **Gated RNNs** und **Long Short Term Memory (LSTM)** - Netzwerke verwendet [2]. Durch einige

Techniken kann man mit diesen Netzen auch gute Laufzeiten erreichen, aber es besteht ein weiteres Problem: Der Kontext der Wörter geht verloren. Zwar können **LSTMs** gut mit kurzen Sätzen umgehen, aber je länger die Wortsequenz, desto schlechter kann sie verarbeitet werden: Es gibt dabei Probleme mit **vanishing** oder **exploding gradients**, je nach Einstellung der Parameter.
Durch die sequentiellen Verarbeitung steigt auch die Verarbeitungszeit mit der Textlänge an, wobei **LSTMs** generell langsamer sind als **RNNs**, da die Neuronen komplexer sind.
Der Umstand, dass diese Netze für jede Anwendung spezifisch trainiert werden müssen, macht ihren Einsatz sehr umständlich. Sprachmodelle hingegen sind vortrainiert, sie brauchen nur eine geringe Anzahl Daten und damit auch wenig Zeit, um für verschiedene Anwendungen einsatzbereit zu sein.

2.2 Transformer

Transformer sind die Basis vieler der heutigen Sprachmodelle [4][5]. Sie wurden 2017 entwickelt, um das Problem der sequentiellen Verarbeitung zu lösen: Als Eingabe kann ein **Transformer** eine ganze Textsequenz verarbeiten.
Wie in Abbildung 2.1 zu sehen, wird die Eingabe zunächst in Wortvektoren für jedes Eingabewort umgewandelt, die sich aus der Addition des Vektors im **Embedding Space** mit einem **Positionsvektor (Positional Encoding)**, der die Position des Wortes im Satz enthält, ergibt. Der Positionsvektor wird durch Sinus- und Kosinusfunktionen bestimmt. Dieser Eingabevektor wird an einen **Encoder** weitergegeben, der eine **Attention-Matrix** für die Eingabesequenz erstellt. Sie besteht aus einem Vektor der Länge der Textsequenz für jedes Wort. Diese Matrix enthält Informationen darüber, wie relevant die einzelnen Wörter der Sequenz füreinander sind: In dem Satz

```
Der braune Hund
```

bezieht sich z.B. das Wort „Der“ sowie das Adjektiv „braune“ auf das Wort „Hund“. In dem Attention-Vektor wären entsprechend die Werte für „Hund“ in den Vektoren der beiden anderen Wörter höher. **Attention** ist der Schlüsselpunkt für die Funktion der Transformer: Dadurch erhalten die Wortvektoren ihre eigentliche Kontextinformation. Um die Werte zu optimieren werden hier die Mittelwerte von acht Vektoren pro Wort ermittelt.

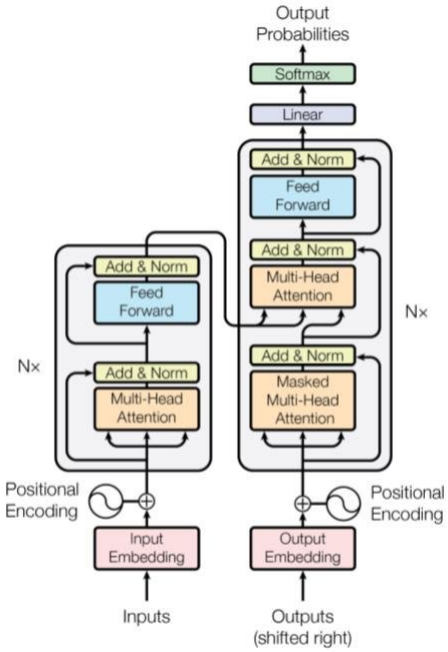

Abbildung 2.1: Aufbau eines Transformers [2]

Alle bisher errechneten Vektoren werden danach aufaddiert, normalisiert und jeder Vektor wird in ein Feed-Forward-Netz gegeben. Hierbei kann parallelisiert werden, da die einzelnen Vektoren voneinander unabhängig sind. Die Ausgabe des **Encoders** sind die mit Kontext und Attention encodierten Wortvektoren.
Der **Decoder** ist ähnlich aufgebaut. Er nimmt einen zweiten Input an - der eigentliche Output bei Supervised Learning - der, wie Abbildung 2.1 zeigt, zunächst auch in einen Eingabevektor mit Positionsinformation und Attention-Matrix umgewandelt wird. Im nächsten Schritt werden Input und Output gemeinsam in ein weiteres Netz zur Attention-Bestimmung gegeben, in dem die Attention Matrix der beiden Vektoren zueinander bestimmt wird: Die Frage die hierbei beantwortet wird ist, wie welche Wörter in den beiden Sequenzen zueinander stehen. Dies ist gut an dem Beispiel der Sprachübersetzung zu sehen, bei der der zu lernende Output der Input in einer anderen Sprache ist.

```
The brown dog
```

wäre entsprechend hier der Output (Decoder Input). Es wird in diesem Schritt analysiert, wie das Wort „The“ zu „Der“, sowie „braune“ zu „brown“ und „Hund“ zu „dog“ steht. Diese Vektoren werden wieder aufaddiert, normiert und in ein Feed-Forward-Netz gegeben, um noch einmal

addiert und normiert zu werden.
Danach wird noch eine lineare und eine Softmax-Schicht angwandt, um am Ende in diesem Beispiel das Wort zu erhalten, dass mit der höchsten Wahrscheinlichkeit als nächstes in dem Satz steht. In der Lernphase wird hierzu **Masking** genutzt, bei dem je ein Wort in dem Satz „maskiert“, also als leerer Platzhalter markiert wird.

2.3 GPT

2.4 ELMo

2.5 BERT

2.6 ULMFit

3. Einsatz von Sprachmodellen

Im Zuge dieser Arbeit wurden Versuche mit verschiedenen Sprachmodellen durchgeführt, die im Folgenden beschrieben werden.

3.1 Versuchsaufbau

Der Source Code für die Versuche ist in Python verfasst, da die Sprachmodelle alle eine übersichtliche Schnittstelle zur Verwendung in Python bieten. Die erstellten Netze sind durch die Bibliotheken *keras* [9] und *Tensorflow* [10] implementiert worden.
Der Code wurde zum Testen auf Googles Plattform *Google Colab* [11] ausgeführt, da mit dieser Plattform eine Laufzeitumgebung mit Möglichkeit zur Nutzung von einer GPU sowie - falls nötig - einer TPU zur Verfügung steht. Die genutzten Datensätze werden über Links zu der Datenquelle (*kaggle.com, github.com*) eingebunden.

3.2 Die Daten

Für **Sentiment Analysis** werden ausschließlich Daten genutzt, die von der Plattform *Twitter* genommen wurden. Diese Daten eignen sich sehr gut für diese Aufgabe, da die Länge der Textstücke auf 280 Zeichen begrenzt ist [7] und es eine große Nutzerbasis und damit eine große Vielfalt an formulierten Texten gibt. Hierbei werden die Tweets genommen, die nur Text und entsprechende Emojis enthalten und in Englisch verfasst sind. Einige der genutzten Datensätze sind gelabelt, um damit **Supervised Learning** zu betreiben, andere sind zum Test der entstehenden Netze und nicht gelabelt.

Die Daten werden alle vorverarbeitet, um die Texte einheitlich und gut verarbeitbar zu machen. Hierzu werden für die Tweets folgende Schritte durchgangen:

- alle Wörter in Kleinbuchstaben umwandeln
- doppelte Buchstaben entfernen
- Whitespaces an Anfang und Ende entfernen

Weiterhin werden die Tweets einer **Tokenization** unterzogen, durch die die Wörter, Emojis und Links in **Tokens** umgewandelt werden. Hierbei wird auch **Stemming** angewendet: Dadurch werden die Wörter in ihre Grundform umgewandelt mit vorausgehenden oder nachfolgenden Silben als separate Tokens. Aus dem Wort „playing“ werden so z.B. die zwei Tokens <play> und <ing>. Diese Art der **Tokenization** ist für den Einsatz mit Sprachmodellen am besten geeignet, da auch in diesen nicht alle Vokabeln einer Sprache enthalten sein können - das wäre schlicht ein zu großes Vokabular. Durch **Stemming** wird gewährleistet, dass die meisten Wörter, sowie die jeweilige Präfix und Suffix je einem Vektor zugeordnet werden können (sollten keine Rechtschreibfehler enthalten sein).
Die Daten für den Teil **Stance detection** haben unterschiedliche Ursprünge. Datensätze bestehen hauptsächlich aus Artikeln, deren Überschriften und den jeweiligen angesprochenen Stances. Auch diese Daten werden einer **Tokenization** unterzogen. Diese unterscheidet sich nur leicht von der für die **Sentiment Analysis** durchgeführte, da die Artikel im Allgemeinen keine sogenannten *handles* oder Emojis enthalten. Diese Datensätze sind gelabelt, um einen späteren Test möglich zu machen.
Wie bei sehr vielen Machine Learning Aufgaben hängen die Ergebnisse stark von der Qualität der genutzten Daten ab. Es wurde demnach versucht, alle Datensätze so zu reinigen, dass die Performanz optimal wurde. Auf etwaige zusätzliche Schritte wird in der jeweiligen Beschreibung eingegangen.

3.2.1 Sentiment140

Dieser Datensatz [8] enthält 16 Millionen Tweets, die mit einem Sentiment-Wert von 0 bis 4 annotiert sind, wobei 4 positiv und 0 negativ ist. Da **sentiment140** von seinen Urhebern schon zum Training für Sentiment Analysis genutzt wurde, ist dieser Datensatz schon weitestgehend gesäubert: Er enthält nur noch wenige doppelte Buchstaben und keine Emoticons, was für die

reine Textauswertung hilfreich ist. Somit müssen in diesem Datensatz noch Nutzernamen und Links entfernt oder in Tokens umgewandelt werden.
Es sind weiterhin nur Tweets enthalten, in denen nicht positive und negative Sentiments gleichzeitig ausgedrückt werden. Der Datensatz wurde durch Auswertung von Emoticons erstellt.

3.3 Sentiment Analysis

3.3.1 Ausgangsbasis

Um einen Vergleich zu haben, wie performant und genau **Sentiment Analysis** mit Sprachmodellen ist, wurde zunächst ein Ansatz ausgewertet, der keinen Gebrauch von Deep Learning macht. Die Veränderung der Genauigkeit im Vergleich zu diesem Ansatz gibt einen Anhaltspunkt, wie viel Sprachmodelle in Sentiment Analysis leisten können.
Zur Auswerung wird der Datensatz 3.2.1 genommen, da dieser genügend groß ist, um eine fundierte Aussage zu treffen. Dieser Datensatz wird mit pythons *pandas* Bibliothek eingelesen, gesäubert und im Anschluss mit der Bibliothek *TextBlob* ausgwertet. Diese Auswertung basiert auf manuell erstellten Dokumenten, in denen allen Wörtern Sentiment-Werte zugeordnet wurden. *TextBlob* werten diese Werte für einen gegebenen Text aus und gibt einen Wert zwischen -1 und 1 zurück (negativ zu positiv).

```
filteredData.tweet.map(lambda x: TextBlob(x).sentiment.polarity)
```

Listing 3.1: Auswertung mit TextBlob

3.3.2 ELMo

3.3.3 BERT

3.4 Stance Detection

3.4.1 BERT

3.4.2 Verwendung anderer Fine Tuning Methoden

4. Auswertung

4.1 Ergebnisse

4.2 Ausblick

Abbildungsverzeichnis

Literaturverzeichnis

[1] Howard, J., Ruder, S., Universal Language Model Fine-tuning for Text Classification. 2018, arXiv:1801.06146

[2] Vaswani, A. et al., Attention Is All You Need. 2017, arXiv:1706.03762v5

[3] Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C. , Lee, K., Zettlemoyer, L., Deep contextualized word rep-resentations. 2018, arXiv:1802.05365v2

[4] Devlin, J., Chang, M., Lee K. , Toutanova, K., BERT: Pre-training of Deep Bidirectional Transformers forLanguage Understanding. 2019, arXiv:1810.04805v2

[5] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I., Improving language understanding by generative pre-training. 2018

[6] Go, A., Bhayani, R., Huang, L., Twitter Sentiment Classification using Distant Supervision. 2009

[7] So twitterst du. https://help.twitter.com/de/using-twitter/how-to-tweet

[8] http://help.sentiment140.com/for-students

[9] https://keras.io/

[10] https://www.tensorflow.org/

[11] https://colab.research.google.com/notebooks/intro.ipynb