Automatische Figurenerkennung in literarischen Texten

Seminararbeit im Rahmen der Veranstaltung
Digitale Objekte modellieren 2: Word Embeddings
im Wintersemester 2018/2019
an der
Julius-Maximilians-Universität Würzburg
Institut für Deutsche Philologie
Lehrstuhl für Computerphilologie und
Neuere Deutsche Literaturgeschichte

Severin Simmler
severin.simmler@stud-mail.uni-wuerzburg.de
Matrikelnummer: 2028090
Studienfach: Digital Humanities, M.A.
Fachsemester: 3

betreut von
Prof. Dr. Fotis Jannidis

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Würzburg,
den 14. April 2019
Ort, Datum

Severin Simmler

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

> The complete meaning of a word is always contextual, and no study of meaning apart from a complete context can be taken seriously.
>
> John Rupert Firth (1935, 37)

Literarische Texte sind als Forschungsgrundlage für die computergestützte Verarbeitung natürlicher Sprache in vielerlei Hinsicht problematisch; eingeschränkt oder überhaupt nur auf Papier verfügbar, fehlerhaft digitalisiert, sprachlich komplex konstruiert. Dagegen gibt es allein in der deutschen Wikipedia mehr als zwei Millionen Artikel (Wikimedia Foundation 2019); frei zugänglich, *born-digital*, sachlich formuliert—dazu kategorisiert und mit weiteren Metadaten ausgezeichnet.

Eine schiere Masse an Textdaten wie die Wikipedia ist in den meisten Fällen nötig, um etwas Komplexes wie die Bedeutung eines Wortes auch nur annähernd modellieren zu können. Hier wird ein Wort nicht mehr als Folge von Zeichen dargestellt, sondern zum Beispiel als hochdimensionaler, dünnbesetzter Vektor der Worthäufigkeiten in den Dokumenten eines Textkorpus (*bag-of-words*). Damit werden Verfahren aus der linearen Algebra anwendbar; die geometrische Beziehung zweier Vektoren ähnelt der semantischen ihrer zu repräsentierenden Worte. Nun hat sich aber gezeigt, dass dichtbesetzte Vektoren, im Kontext sogenannter *word embeddings*, die bessere Wahl für Probleme aus dem Bereich der maschinellen Sprachverarbeitung sind (Jurafsky und Martin 2009, 18). Modelle wie *skip-gram* (Mikolov et al. 2013) werden auf sehr großen Textkorpora trainiert, und können auf diese Weise einen Vektorraum produzieren, in dem jedem Wort im Vokabular des Korpus ein dichtbesetzter Vektor zugewiesen wird. Durch das Training werden so viele Informationen über Sprache im Vektorraum abgebildet, dass diese t r a n s f e r i e r t, und für ein verwandtes Problem genutzt werden können—man spricht von sogenanntem *transfer learning*. Die Vektoren in klassischen *word embeddings* (Mikolov et al. 2013; Bojanowski et al. 2016) sind dabei aber statisch und kontext-agnostisch. ‚Bank' als Kreditinstitut ist identisch mit ‚Bank' als Möbelstück. Diesem Problem der Mehrdeutigkeit wurde mit kontextabhängigen *word embeddings* entgegengewirkt (Devlin et al. 2018). Gegeben sei ein Wort in einer bestimmten Phrase, einem Satz, einem Paragraph, also einem Kontext, wird die Vektorrepräsentation auch entsprechend spezifisch sein.

In der vorliegenden Arbeit wird nun untersucht, inwiefern *word embeddings*, sowohl kontext-agnostisch, kontextabhängig, als auch kombiniert, für die automatische Erkennung von Figuren in literarischen Texten von Nutzen sind—im Vergleich zu einem klassischen Ansatz (Jannidis et al. 2015). Das heißt, die aus den Artikeln der Wikipedia gewonnenen generischen Informationen über Sprache werden genutzt, um ein Modell auf einem kleineren Korpus deutschsprachiger Romane, in dem mehrere tausend Figurenreferenzen verschiedener Art markiert sind, zu trainieren. Auf diese Weise soll das Modell

im Idealfall Sequenzmuster aus den Trainingsdaten präziser abstrahieren, und Figuren in unbekannten Erzähltexten besser erkennen.

Gegliedert in zwei großen und drei kleinen Kapiteln wird zunächst ein theoretisches Fundament gelegt (2), um im praktischen Teil (3) die Durchführung aller Experimente nachzuzeichnen (3.1–3.4), und die Ergebnisse zu diskutieren (3.5). Vor einem abschließenden Fazit mit Ausblick auf weitere Forschungsmöglichkeiten (6) wird ein Optimierungsansatz vorgeschlagen (4), sowie exemplarisch ein konkreter Anwendungsfall für die automatische Figurenerkennung aufgezeigt (5).

2 Theoretische Grundlagen

Im folgenden Teil werden theoretische Grundlagen geklärt, die zum Verständnis der in Kapitel 3 durchgeführten Experimente erforderlich sind. Nach einer kurzen Einführung in das Problem der automatischen Figurenerkennung (2.1) wird das Konzept kontextagnostischer und kontextabhängiger *word embeddings* erläutert (2.2). Anschließend wird ein Verfahren, sowie drei Metriken zur Evaluation vorgestellt (2.3).

2.1 Figurenerkennung

Automatische Figurenerkennung, als Teilaufgabe des generischen *named entity recognition* (NER) Problems, zielt darauf ab, alle Figuren in einem literarischen Text zu erkennen, entweder regelbasiert oder abgeleitet von einer Wahrscheinlichkeitsverteilung, und einem entsprechenden kategorialen Deskriptor zuzuweisen. Eine Kombination der beiden Ansätze, regelbasiert und probabilistisch, kann unter Umständen die Erkennungsrate des Systems erhöhen.

Regelbasiert heißt, dass beim Labeling der Instanzen einer Sequenz[1] ein zuvor definiertes Regelwerk berücksichtigt wird; dieses kann beispielsweise umfangreiche Listen mit konkreten Beispielen für bestimmte Entitäten enthalten. In Algorithmus 1 wird für jedes Wort eines Satzes geprüft, ob es in einer Liste mit Vornamen vorkommt. Trifft dies zu, wird dem Wort das Label $person$ zugewiesen, andererseits $word$. Ein Satz[2] würde demnach folgendermaßen gelabelt werden:

Charlotte	und	**Eduard**	und	**J.**	**W.**	**von**	**Goethe**
$person$	$word$	$person$	$word$	$word$	$word$	$word$	$word$

Hier wird zwei von insgesamt drei Entitäten das korrekte Label $person$ zugewiesen. **J. W. von Goethe** wurde nicht erkannt, da der Name in der Liste nicht erfasst ist. Durch strikte Regeln können regelbasierte Systeme zwar sehr genau sein, erfassen aber oft nur wenige Entitäten—bedingt durch beispielsweise die endlichen Listen.

Hierin liegt nun die Stärke der probabilistischen Figurenerkennung. Ausgangspunkt ist ein Datensatz, der korrekt gelabelte Beispiele enthält. Jedem Element in einer Sequenz, also jedem Token in einem Satz, ist sein, in der Regel durch Menschenhand, passendes Label zugewiesen. Mithilfe von Verfahren aus dem Bereich des maschinellen Lernens leitet ein Algorithmus nun bestimmte Muster aus den Daten ab, die für jede Entitätsklasse besonders typisch sind. Durch dieses generalisierte Wissen kann für die Worte einer (unbekannten) Sequenz eine Wahrscheinlichkeitsverteilungen über alle Labels berechnet werden. Das Label, das für ein Wort am wahrscheinlichsten ist, wird ausgewählt (Algorithmus 2).

[1] Eine Sequenz ist im Kontext dieser Arbeit immer ein Satz. Instanzen der Sequenz sind Worte bzw. Tokens.
[2] Alle fett hervorgehobenen Worte zählen zur Klasse $person$ und müssten entsprechend erkannt werden.

Ein auf Algorithmus 2 basiertes Modell hat aus den Trainingsdaten abstrahiert, dass Personennamen unter Umständen aus mehreren Tokens bestehen können, bindet sich dabei aber nicht wie ein regelbasiertes System an konkrete Worte, und würde den Beispielsatz folgendermaßen labeln:

Charlotte	und	**Eduard**	und	**J.**	**W.**	**von**	**Goethe**
$person$	$word$	$person$	$person$	$person$	$person$	$person$	$person$

Hier wurden nun zwar den Komponenten aller drei Entitäten das passende $person$ Label zugewiesen, fälschlicherweise aber auch einmal dem Token ‚und'. Das Ergebnis ist demnach so zu interpretieren, dass **Charlotte**, sowie **Eduard und J. W. von Goethe** jeweils einer Entität entspricht, was für den ersten Fall korrekt, für den zweiten aber falsch ist. Das Modell bildet zwar ab, dass ein Name aus mehreren Einzelteilen bestehen kann, hat aber Schwierigkeiten, zwei kurz aufeinander folgende Namen voneinander zu trennen.

Algorithmus 1 Regelbasierter Ansatz für Figurenerkennung

```
1:  function RegelbasierteFigurenerkennung(sentence)
2:      names ← {Wolfgang, Charlotte, Ottilie, Eduard, Johann}
3:      labels ← [ ]
4:      for token in sentence do
5:          if token in names then
6:              add person to labels
7:          else
8:              add word to labels
9:          end if
10:     end for
11:     return labels
12: end function
```

Algorithmus 2 Probabilistischer Ansatz für Figurenerkennung

```
1: function ProbabilistischeFigurenerkennung(sentence)
2:     model ← f(x)
3:     labels ← [ ]
4:     for token in sentence do
5:         probabilities ← f(token)
6:         add label with max(probabilities) to labels
7:     end for
8:     return labels
9: end function
```

2.2 Word Embeddings

Ein *word embedding* wandelt ein Wort, das zunächst als Folge von Zeichen dargestellt wird, in eine Reihe von Zahlen um; die numerische Repräsentation desselben Wortes kann dabei unter Umständen variieren[3]. Diese Übersetzung ist in der Regel erforderlich, wenn natürliche Sprache von Algorithmen aus dem Bereich Machine- oder Deep Learning verarbeitet werden soll, da mathematische Operationen nicht auf einen String bzw. auf rohen Text angewendet werden können, viele Algorithmen dies aber erfordern. Somit bildet ein oder mehrere *embeddings* immer die erste Schicht eines Modells mit neuronaler Architektur, die alle eingehenden Tokens in eine entsprechende Repräsentation überführt.

Der sogenannte *skip-gram* Algorithmus, der in $word2vec$[4] (Mikolov et al. 2013) implementiert ist, versucht für ein Zielwort alle Kontextwörter in einem bestimmten Fenster vorherzusagen, und lernt dabei, wo einzelne Worte in einem Vektorraum positioniert sind:

> [The skip-gram algorithm] learns embeddings by starting with an initial set of embedding vectors and then iteratively shifting the embedding of each word w to be more like the embeddings of words that occur nearby in texts, and less like the embeddings of words that don't occur nearby (Jurafsky und Martin 2009, 21).

In $fastText$[5] wird der *skip-gram* Gedanke weitergeführt, wobei ein Wort aber nicht mehr als solches behandelt wird, sondern als Summe von n-grams. Es werden also „representations for character n-grams" (Bojanowski et al. 2016, 1) statt der „representations of words" (Mikolov et al. 2013, 10) gelernt; damit wird das Modell bei Worten, die nicht Teil des Vokabulars des Trainingskorpus waren, flexibler. In welchem konkreten Kontext ein Wort auftritt, wird von diesen klassischen *word embeddings* außer Acht gelassen, ist aber erforderlich, um zwischen mehrdeutigen Worten zu unterscheiden.

Dieser Schwäche wird von kontextabhängigen Sprachmodellen wie BERT (Devlin et al. 2018) entgegengewirkt „by jointly conditioning on both left and right context in all layers" (ebd., 1). BERT macht sich einen sogenannten Aufmerksamkeits-Mechanismus zunutze, der eine Beziehung zwischen bestimmten Worten in einer Sequenz herstellt, um eine Repräsentation der Sequenz als Ganze zu berechnen (Vaswani et al. 2017, 1); w i c h t i g e n Worten in der Sequenz wird also besondere Aufmerksamkeit gegeben. Kontextabhängige *word embeddings* zeichnet nun aus, dass der Vektor für ein Wort, je nachdem mit welchen anderen Worten es in einer Sequenz auftaucht, variiert, und somit dynamisch ist—Ambiguitäten können abgebildet werden.

[3] Im Fall von kontextabhängigen *word embeddings*.

[4] https://github.com/tmikolov/word2vec

[5] https://fasttext.cc/

Um dies an einem Beispiel zu verdeutlichen, wurde das Wort ‚Bank' in den folgenden vier Sätzen im Sinne von Kreditinstitut (a und b), sowie Möbelstück (c und d) in den BERT Vektorraum eingebettet:

a) Eine Bank ist ein Kreditinstitut.

b) Ich bringe mein Geld gerne zur Bank.

c) Die Bank ist zum Sitzen da.

d) Eine Bank ist auch ein Möbelstück.

Die vier 3072-dimensionalen Vektorrepräsentationen desselben Wortes unterscheiden sich alle, da das Wort in unterschiedlichen Sätzen mit teilweise verschiedenen, aber für eine bestimmte Bedeutung typischen Wörtern verwendet wird. Mithilfe des Verfahrens t-SNE (Hinton und van der Maaten 2008) wurden die zu einer Matrix zusammengesetzten Vektoren auf zwei Dimensionen reduziert, und in Abbildung 2.1 visualisiert. Hier wird deutlich, wie gut BERT Mehrdeutigkeit abbildet. Das Kreditinstitut lässt sich deutlich vom Möbelstück unterscheiden; bei einem klassischen *word embedding* würde sich der Vektor für alle vier Sätze nicht unterscheiden.

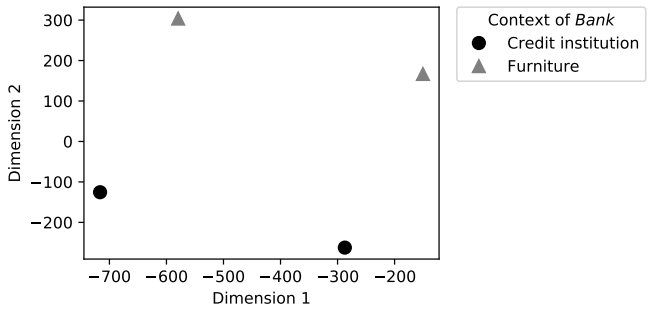

Abbildung 2.1: Positionen im BERT Vektorraum für das Wort ‚Bank'. Mithilfe des Verfahrens t-SNE von 3072 auf zwei Dimensionen reduziert.

2.3 Evaluationsverfahren

Bei der sogenannten k-fachen Kreuzvalidierung wird ein Datensatz zunächst in k etwa gleich große Teile aufgespalten, wobei jeder Teil einmalig als Validierungsset[6] verwendet wird, während die $k - 1$ verbleibenden Teile das Trainingsset bilden (Abbildung 2.2). Das heißt, insgesamt werden k Modelle trainiert, und anschließend zum Beispiel der F_1 Wert für das jeweilige Validierungsset berechnet. Es wird angenommen, dass der Datensatz eine repräsentative Stichprobe der Grundgesamtheit ist—im Fall dieser Arbeit also aller deutschsprachiger literarischer Texte. Somit nähert sich das arithmetische Mittel der aus der Kreuzvalidierung gewonnenen F_1 Werte dem der Grundgesamtheit an. F_1 ist dabei das harmonische Mittel von $Precision$ und $Recall$ (Formel 2.1).

[6] Validierungsset heißt, dass dieser Teil des Datensatzes nicht für das Training verwendet wird, sondern vom Modell klassifiziert wird und dieses anhand des F_1 Scores bewertet wird.

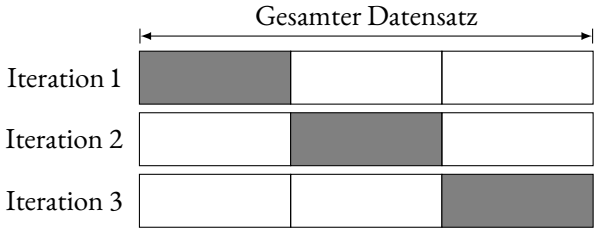

Abbildung 2.2: Aufteilung eines Datensatzes in Validierungsset (grau) und Trainingssets (weiß) bei einer 3-fachen Kreuzvalidierung. In insgesamt drei Durchläufen werden drei Modelle mit dem Trainingsset erstellt, und mit dem Validierungsset evaluiert.

$Precision$ und $Recall$ berechnen sich mit Zähldaten, wie das Modell Sequenzen im Testset gelabelt hat. Dabei ist TP (*true positive*) die Zahl der vollständig und korrekt erkannten Entitäten. FP (*false positive*) gibt die absolute Häufigkeit der Tokens bzw. Gruppe von Tokens an, die fälschlicherweise als Entität klassifiziert wurden. FN (*false negative*) ist die Zahl der Entitäten, die vom Modell nicht als solche erkannt wurden. Bei der $Micro$ Gewichtung werden alle Zahlen ungeachtet der Klassen ermittelt; dies ist unter Umständen dann sinnvoll, wenn diese ungefähr gleichverteilt sind, oder das Modell als Ganzes evaluiert werden soll (Formel 2.2).

Bei der $Macro$ Gewichtung werden $Precision$ bzw. $Recall$ für jede Klasse separat berechnet (Formel 2.2), und anschließend gemittelt, indem die aufsummierten Werte durch N, also die Anzahl der Klassen, geteilt wird; dies kann vor allem dann sinnvoll sein, wenn die Instanzen ungleich verteilt sind, und unterrepräsentierte Klassen gleich viel Einfluss auf den F_1 Wert haben sollen, wie überrepräsentierte (Formel 2.3).

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.1}$$

$$Precision_{Micro} = \frac{TP}{TP + FP} \qquad Recall_{Micro} = \frac{TP}{TP + FN} \tag{2.2}$$

$$Precision_{Macro} = \frac{\sum_{n=1}^{N} Precision_n}{N} \qquad Recall_{Macro} = \frac{\sum_{n=1}^{N} Recall_n}{N} \tag{2.3}$$

3 Experimente

In diesem Kapitel wird zunächst das verwendete Korpus (3.1), die Hyperparameteroptimierung (3.2), sowie die Modellarchitektur (3.3) vorgestellt. Die Qualität des Modells wurde bei einer 10-fachen Kreuzvalidierung der Trainingsdaten mit dem gemittelten $F1$ bewertet (3.4). In einer abschließenden Diskussion (3.5) werden die Schwächen und Stärken des Modells aufgezeigt. Sowohl das Modell (Simmler 2019a), als auch eine Python Bibliothek (Simmler 2019b), mit der das Modell in wenigen Zeilen Code verwendet werden kann, stehen öffentlich zur Verfügung[1].

3.1 Korpus

Modelle für NER sind oft auf Zeitungskorpora trainiert (Frauqui und Pado 2010; Lample et al. 2016; Hänig et al. 2014), und erkennen entsprechend nur wenige *named entities* in Erzähltexten[2]. Jannidis et al. (2015) konnten mit dem Stanford Parser (Finkel et al. 2005) und einer Sammlung mehrere Romane einen $F1$ Score von nur 24% erreichen, was auf einen sehr niedrigen $Recall$ zurückzuführen war (Jannidis et al. 2015, 3). Aus diesem Grund entstand DROC[3], ein Korpus literarischer Texte mit etwa 390.000 annotierten Tokens und mehr als 50.000 Figurenreferenzen. Tabelle 3.1 sind Statistiken zum Korpus selbst, und Tabelle 3.2 Zahlen zur Verteilung der Klassen zu entnehmen.

Das Korpus wurde von TEI XML in ein Tab-separiertes Format umgewandelt, in dem ein fortlaufender Satzindex, Tokens und entsprechende Annotationen in je einer Spalte enthalten sind. Diese Datei kann mit Flair (Akbik et al. 2018), einer *high-level* Python Bibliothek, die auf PyTorch (Paszke et al. 2017) aufsetzt, eingelesen und als Datensatz für die Modellerstellung verwendet werden. Hierfür wurde das Korpus im Verhältnis 80:15:5 in Trainings- Test- und Devset aufgeteilt. Für die Hyperparameteroptimierung wurde das Devset, und für die Evaluation des finalen Modells das Testset verwendet.

NER ist im Allgemeinen ein stark domänenabhängiges Problem, wobei für die Figurenerkennung in literarischen Texten zwischen verschiedenen syntaktischen Kategorien unterschieden werden muss:

> In a literary text, a [character] reference can appear in one of three broad syntactic categories: 1) as a proper noun; 2) as a nominal phrase; 3) as a pronoun. Detecting categories 2) and 3) is usually beyond the scope of named entity recognition (Krug et al. 2017, 4).

[1] https://github.com/severinsimmler/figur

[2] In Zeitungen werden in der Regel reale Personen, Orte, Organisationen erwähnt—literarische Texte bewegen sich oft in einer fiktionalen Welt. Desweiteren, und dies wird sich im weiteren Verlauf auch zeigen, gibt es bei der Figurenerkennung sehr spezifische Entitäten.

[3] https://gitlab2.informatik.uni-wuerzburg.de/kallimachos/DROC-Release

Quelle	TextGridRep[5]
Veröffentlichungszeitraum	1788–1984
Romane	90
Sätze	18.161
Tokens	393.164
Längster Satz, in Tokens	385
Kürzester Satz, in Tokens	1
Durchschnittliche Satzlänge, in Tokens	21
Standardabweichung Satzlängen	18,39

Tabelle 3.1: Details zum DROC Korpus.

	Annotierte Tokens	
	Absolut	Relativ
Pron	21.072	0,6157
AppTdfW	5.516	0,1611
Core	5.376	0,1571
AppA	2.257	0,0659

Tabelle 3.2: Details zur Verteilung der Klassen des DROC Korpus.

Nach Krug et al. (2017) wurden im Korpus alle Nominalphrasen annotiert, die eine Referenz auf eine Figur enthält. Dabei wurde zwischen folgenden Entitätsklassen unterschieden:

a) **Eigennamen**: Dies schließt Vor-, Nach- oder Familiennamen ein, die nicht zwangsläufig Teil der fiktionalen Welt sind (wie etwa eine historische Person). Klassenname: $Core$.

b) **Gattungsnamen**: Gemeint sind Berufsbezeichnungen, Geschlechtertermini, Titel etc. wie zum Beispiel ‚Bäcker', ‚Mann', oder ‚Graf', die Teil der fiktionalen Welt sind, oder sich auf generische Entitäten beziehen, die nicht Teil der fiktionalen Welt sind. Klassennamen: $AppTdfW$ bzw. $AppA$.

c) **Pronomen**: Jegliche Arten von Pronomen, etwa Personal-, Possessiv-, Reflexiv- oder Relativpronomen. Klassenname: $Pron$.

d) **Keine Entität**[4]: Entspricht ein Token keiner der vorausgegangenen Klassen, wird es der Klasse $Word$ zugeteilt.

3.2 Hyperparameter

Im Rahmen einer umfangreichen Hyperparameteroptimierung wurden insgesamt 50 Modelle für jeweils eine Epoche mit variierenden Hyperparametern trainiert; zum Einsatz kam hier die Python Bibliothek Hyperopt (Bergstra et al. 2015).

[4] Bei Krug et al. (2017) gab es diese Klasse nicht explizit und wurde erst in dieser Arbeit eingeführt.

Folgende Parameter wurden in den angegeben Suchräumen optimiert:

a) **RNN Schichten**: Anzahl der biLSTM Layer; $\{1, 2\}$. Optimal: 1.

b) **Hidden size**: Anzahl der Neuronen pro Layer; $\{32, 64, 128\}$. Optimal: 64.

c) **Dropout**: Zufällig ausgewählte Neuronen werden mit einer Wahrscheinlichkeit von $unif(0, 0.5)$ auf Null gesetzt. Optimal: 0.001.

d) **Learning rate**: Wie schnell (oder langsam) wird gelernt; $\{0.05, 0.1, 0.15, 0.2\}$. Optimal: 0.15.

e) **Mini batch size**: Anzahl der Instanzen, die auf einmal durch das Netzwerk getrieben werden; $\{8, 16, 32\}$. Optimal: 8.

Zudem wurden verschiedene Varianten von *word embeddings* als Features gegeneinander abgewogen (Abbildung 3.1). Zum Einsatz kamen die kontext-agnostischen fastText (Bojanowski et al. 2016) und Character Embeddings (Lample et al. 2016), sowie die kontextabhängigen Flair (Akbik et al. 2018) und BERT (Devlin et al. 2018).

Das beste kontext-agnostische sollte mit dem besten kontextabhängigen *embedding* kombiniert, das heißt übereinander gestapelt werden (vgl. Abbildung 3.3), in der Hoffnung damit bessere Ergebnisse erzielen zu können. Tatsächlich wurde bei fastText mit BERT eine Verbesserung des $Macro\ F1$ Werts[6] von 0,60 auf 0,66 beobachtet (Abbildung 3.1).

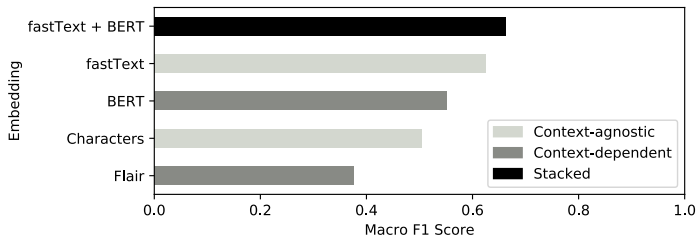

Abbildung 3.1: Jeweils bestes Modell mit verschiedenen Features. Für jede Variante wurden insgesamt 50 Modelle á eine Epoche mit variierenden Hyperparametern trainiert und mit dem Devset evaluiert. Die fünfte Klasse, die keine Entität repräsentiert, wurde noch berücksichtigt. $Macro\ F1$ als Evaluationsmaß soll bei der Optimierung die unterrepräsentierten Klassen nicht vernachlässigen.

Um die ideale Anzahl der Epochen zu ermitteln, wurde mit den optimierten Hyperparametern ein Modell über 100 Epochen trainiert. Der höchste $Micro\ F1$ Wert[7], 0.9639 inklusive der Klasse $Word$, wurde bei Epoche 55 erreicht (Abbildung 3.2).

[6] Hier wurde die $Macro$ Gewichtung gewählt, da die unterrepräsentierten Klassen bei der Optimierung nicht vernachlässigt werden sollten.

[7] Hier wurde die $Micro$ Gewichtung gewählt, da bei dem letzten Hyperparameter das Modell als Ganzes optimiert werden sollte.

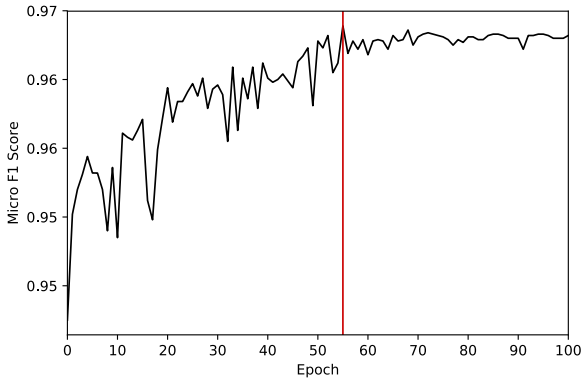

Abbildung 3.2: $Micro\ F1$ im Verlauf der Epochen. Das Maximum wird bei 55 erreicht (senkrechter Strich), was der idealen Anzahl Epochen entspricht (*early stopping*). Das Modell wurde mit den optimalen Hyperparametern (vgl. Abbildung 3.1) trainiert und mit dem Dev Set evaluiert.

3.3 Modellarchitektur

Das Modell (Abbildung 3.3) besteht aus insgesamt zwei Schichten. In der ersten Schicht sind zwei *word embeddings* aufeinander gestapelt; das erste bettet die eingehenden Tokens nacheinander in einen 300-dimensionalen kontext-agnostischen, das zweite in einen 3072-dimensionalen kontextabhängigen Raum ein. Diese werden an einen bidirektionalen Long Short-Term Memory Layer (biLSTM)[8] weitergereicht. Diese *many-to-many* Architektur nimmt eine Sequenz von Tokens als Eingabe, und produziert eine gleichlange Sequenz von Labels.

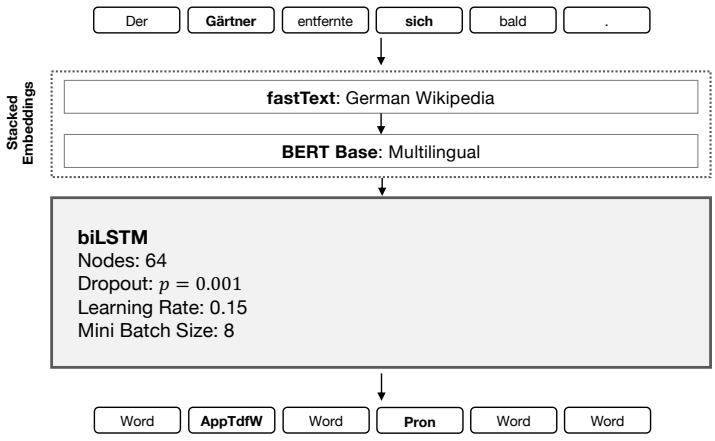

Abbildung 3.3: Architektur des Modells.

[8] Ein biLSTM ist eine Variante eines bidirektionalen rekurrenten neuronalen Netzwerkes, worauf aus formalen Gründen nicht weiter eingegangen werden kann. Ein Long Short-Term Memory Netzwerk „can learn to bridge time intervals [...] without loss of short time lag capabilites“ (Hochreiter und Schmidhuber 1997, 1), das heißt Informationen sind in LSTMs für eine gewisse Zeit beständig. Eine Eingabe wird über mehrere Zeitschritte hinweg verarbeitet, springt zurück an den Anfang des Intervalls, und korrigiert ggf. Fehler, die sich im Laufe der Zeit herausgestellt haben (ebd., 4f).

	Micro Precision	Micro Recall	Micro F1
Jannidis et al. 2015	0,9486	0,8560	0,8998
Simmler 2019b	0,8057	0,6682	0,7287

Tabelle 3.3: Vergleich von $Precision$, $Recall$ und $F1$ des Modells zur Baseline.

3.4 Evaluation

Um das Modell mit der Baseline von Jannidis et al. (2015), einem *conditional random field classifier*, zu vergleichen, wurde das Trainingsset (217.608 Tokens, 9.992 Instanzen) 10-fach kreuzvalidiert (vgl. ebd.). Die Klasse $Word$ wurde dabei außen vor gelassen, da diese keine Entität repräsentiert. Ein TP wurde dann gezählt, wenn eine Entität korrekt und vollständig erkannt wurde.

Es konnte keine Verbesserung nachgewiesen werden (Tabelle 3.3); $Micro\ Precision$ streut zwischen 0,7611 und 0,8725, $Micro\ Recall$ zwischen 0,5814 und 0,7458, $Micro\ F1$ entsprechend zwischen 0,6903 und 0,7542 (Abbildung 3.4).

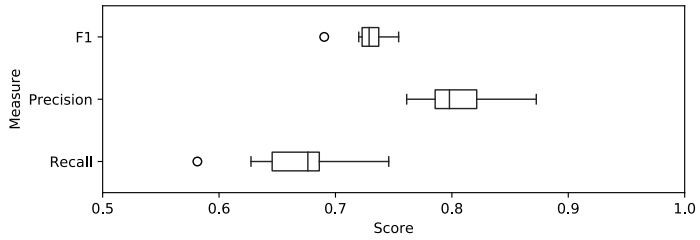

Abbildung 3.4: Verteilung von $Micro\ Precision$, $Micro\ Recall$ und $Micro\ F1$ bei der 10-fachen Kreuzvalidierung des Trainingsset.

3.5 Diskussion

Die Konfusionsmatrix in Abbilung 3.5 zeigt, wie das mit dem gesamten Trainingsset erstellte finale Modell das Testset (58.265 Tokens, 2.726 Instanzen) gelabelt hat. Nur etwa 1% der Tokens der Klasse $Word$ wurden fälschlicherweise als $Pron$ klassifiziert. Hier handelt es sich zwar tatsächlich um insgesamt 598 Pronomen, die sich aber zum Beispiel nicht auf Figuren beziehen (*Erbäte **sich's** zur Gnade*), Tippfehler (***Er** Verfolgungswahn brach aus*), oder auch Annotationsfehler sind.

Die Klasse $Pron$ hat mit 73% am besten abgeschnitten, trotzdem wurden viele der Pronomen nicht als solche erkannt. Inwiefern die Erkennungsrate für diese Klasse erhöht werden kann, ist den nachfolgenden Optimierungsvorschlägen (4) zu entnehmen. Mit insgesamt 72% erkannten Entiäten schließt sich die $Core$ den Pronomen an. Auch hier wurde mehr als ein Viertel nicht erkannt (zum Beispiel Namen wie *Eulenspiegel*, *Heinrich* oder *Häsli*). Teilweise wurden Fragmente als $AppTdfW$ gelabelt, wie *König* in *König Chosroes*. Mit 58% wurde nur knapp über die Hälfte der Tokens von $AppTdfW$ korrekt erkannt. Einige wurden hier der Klasse $Core$ zugeordnet (*Frau Professorin* etc.). Die Klasse $AppA$, zudem auch die Klasse, mit den wenigsten Trainingsbeispielen, hat eine Erkennungsrate von nur 2%. Der größte Teil wurde hier überhaupt nicht erkannt, der

zweitgrößte Teil als *AppTdfW* klassifiziert (wie zum Beispiel *Provinzmenschen* oder *Untertanen*).

Mit Eigennamen, die aus mehr als einem Wort bestehen, kommt das Modell hingegen sehr gut zurecht. Von insgesamt 102 Instanzen im Testset, die aus zwei oder mehr Tokens bestehen, wurden 101 vom Modell korrekt und komplett erkannt.

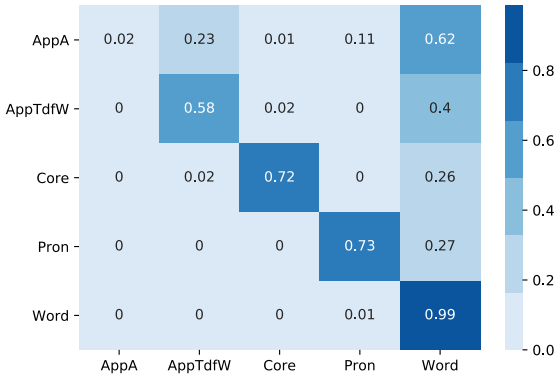

Abbildung 3.5: Konfusionsmatrix des finalen Modells evaluiert mit dem Testset. Angegeben sind die relativen Häufigkeiten der klassifizierten Entitäten. Je dunkler die Zellen in der Diagonale, desto besser ist das Modell.

4 Optimierungsvorschlag

Um die Erkennungsrate der Figurenerkennung zu erhöhen, wurden Experimente mit einem regelbasierten Ansatz durchgeführt. Da es, im Gegensatz zu Eigennamen der Figuren, nur eine endliche Anzahl an Pronomen gibt, wurden zunächst alle als $Pron$ ausgezeichneten Tokens im Trainingsset ausgelesen, um eine Liste anzufertigen; von insgesamt 187 Pronomen wurden 42 ausgewählt. Demonstrativpronomen wie *der*, *die* oder *das*, die nicht eindeutig als solche zu erkennen sind bzw. die Wortart aus dem Kontext erschlossen werden muss, wurden aus der Liste entfernt, um *false positives* vorzubeugen. Die Auswahl ist Abbildung 4.1 zu entnehmen; diese wurden, vergleichbar mit Algorithmus 1, beim Labeling berücksichtigt.

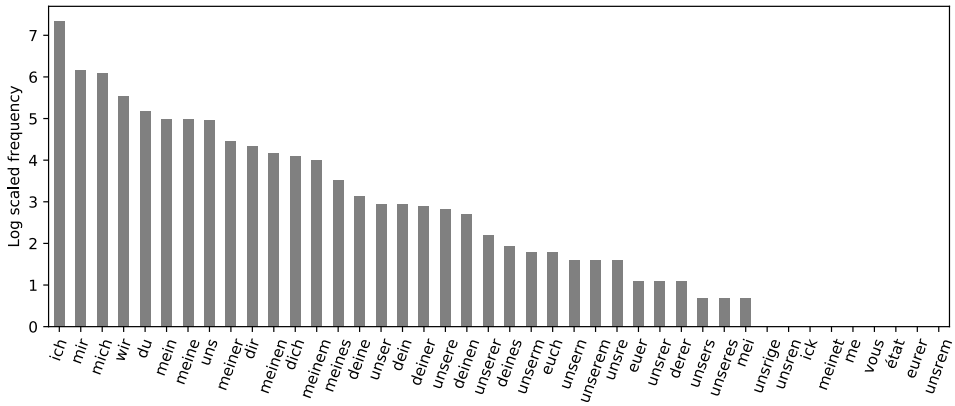

Abbildung 4.1: Die 42 ausgewählten Pronomen mit ihrer logarithmisch skalierten Häufigkeit im Trainingsset.

Wie sich gezeigt hat (Tabelle 4.1), haben sich alle Werte durch die Liste nochmals verschlechtert. Dies war etwa darauf zurückzuführen, dass Pronomen nur in bestimmten Fällen für die Figurenerkennung relevant sind. Vom Regelsatz wird zum Beispiel nicht berücksichtigt, ob ein Pronomen in direkter Rede vorkommt, oder sich nicht auf eine Figur, sondern auf eine Sache bezieht; diese Fälle sind nach Jannidis et al. (2017) für die Figurenerkennung allerdings nicht relevant sind.

	Micro Precision	**Micro Recall**	**Micro F1**
Jannidis *et al.* 2015	0, 9486	0, 8560	0, 8998
Simmler 2019b	0, 8057	0, 6682	0, 7287
Simmler 2019b + Liste	0, 5884	0, 5433	0, 5651

Tabelle 4.1: Vergleich von $Precision$, $Recall$ und $F1$ des Modells in Kombination eines Regelsatzes zum Modell alleine und zur Baseline.

5 Anwendungsbeispiel

Um entgegen der enttäuschenden Ergebnisse aufzuzeigen, wie wertvoll ein solches Modell dennoch ist, wurden Goethes Wahlverwandtschaften[1] mithilfe der im Rahmen dieser Arbeit entwickelten Python Bibliothek (Simmler 2019a) gelabelt—der Zeitaufwand auf vier CPUs betrug für den gesamten Roman etwa 70 Sekunden. Der rohe Text wurde zunächst eingelesen, in Sätze[2] segmentiert, und vom Modell gelabelt. Es wurde der naive Ansatz[3] verfolgt, dass eine Beziehung zwischen zwei Entitäten besteht, wenn beide im selben Satz erwähnt werden. So wurde exemplarisch untersucht, wie die Hauptpersonen miteinander (Abbildung 5.1), sowie die Hauptpersonen mit Appelativa (Abbildung 5.2) in Verbindung stehen. Die Netzwerke wurden mit dem Python Paket NetworkX[4] erstellt und visualisiert. Entitäten entsprechen Knoten, und für jedes gemeinsame Auftreten in einem Satz wurde das Gewicht der entsprechenden Kante inkrementiert. Das Labeling des Modells wurde mit geringem Aufwand manuell überarbeitet. Problematisch war hier, dass ‚Hauptmann' entweder als $AppA$ oder $AppTdfW$ erkannt wurde, in diesem konkreten Fall aber eine Hauptperson ist, nämlich Hauptmann Otto.

Wie Abbildung 5.1 zu entnehmen ist, besteht eine ausgeprägte Dreiecksbeziehung zwischen Eduard, Charlotte und Ottilie. Zwischen Luciane und Ottilie besteht beispielsweise nur eine indirekte Verbindung durch Charlotte und Eduard. In Abbildung 5.2 hingegen wird deutlich, mit welchen Figuren Eduard und Charlotte in Beziehung stehen, so werden beide häufig unter anderem mit einem Gatten, einer Gattin und einem Vater im selben Satz erwähnt[5], was bereits Schlüsse auf die Beziehung der beiden zulässt—ein Ehepaar mit Kinder.

[1] Mit dem Modell von Jannidis et al. 2015 wurde eine ähnliche Analyse der Wahlverwandtschaften durchgeführt (`http://kallimachos.de/kallimachos/index.php/Tutorial_Figurennetzwerke`), die zu ähnlichen Erkenntnissen führte, zum Beispiel, dass Charlotte und Ottilie die am stärksten vernetzte Knoten sind.

[2] Mithilfe der Python Bibliothek spaCy (`https://spacy.io/`).

[3] Auch hier wurde auch die Tatsache außer Acht gelassen, dass zum Beispiel Eduard und der Hauptmann mehrfach benannt sind.

[4] `https://networkx.github.io/`

[5] Womit sie selbst gemeint sind. Untersuchungsgegestand der sogenannten Koreferenzauflösung ist es, verschiedene

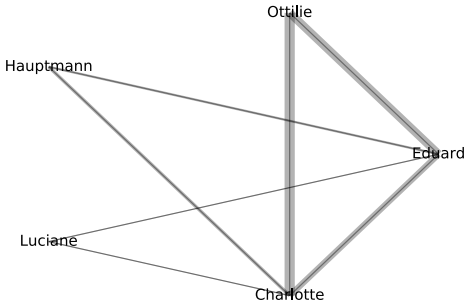

Abbildung 5.1: Soziales Netzwerk von fünf Figuren in den Wahlverwandtschaften (Goethe 1809). Je dicker die Kante zwischen zwei Knoten, desto öfter werden die Figuren gemeinsam in einem Satz erwähnt.

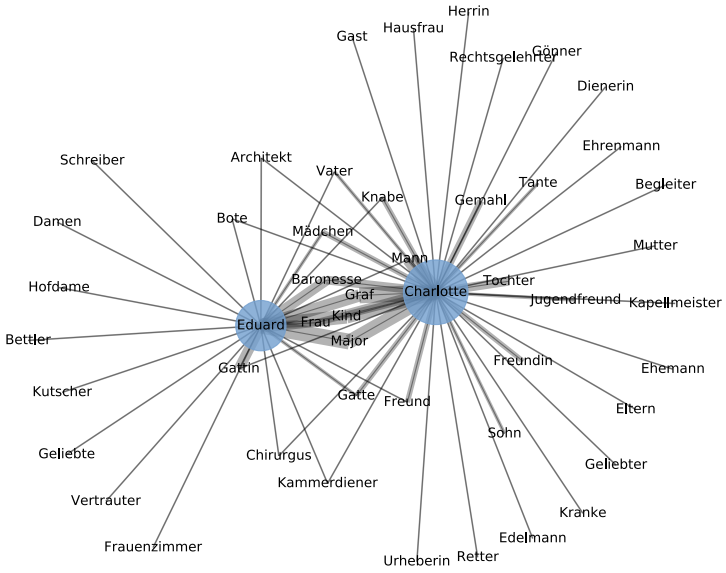

Abbildung 5.2: Beziehung von Eduard und Charlotte zu den Figuren der Klasse Appelativ (Abstraktum und Teil der fiktionalen Welt) in den Wahlverwandtschaften (Goethe 1809). Je dicker die Kante zwischen zwei Knoten, desto öfter werden die Figuren gemeinsam in einem Satz erwähnt. Die blauen Kreise dienen allein der besseren Darstellung.

6 Ausblick und Fazit

Thema der vorliegenden Arbeit war die automatische Figurenerkennung in literarischen Texten mit *word embeddings*. Gegliedert in mehreren Kapiteln wurden im ersten Teil theoretische Grundlagen geklärt, der Unterschied zwischen der regelbasierte und probabilistischen Figurenerkennung, sowie das Vorgehen bei der Evaluation, alos die k-fache Kreuzvalidierung und die Evaluationsmaße $Precision$, $Recall$ und $F1$ mit verschiedenen Gewichtungen.

Im nachfolgenden Teil wurde das verwendete Korpus, eine Sammlung annotierter deutschsprachiger Romane aus dem 19. und 20. Jahrhundert, näher beschrieben, die Hyperparameteroptimierung nachgezeichnet, sowie die Architektur des Modells vorgestellt. Bei der Evaluation hat sich herausgestellt, dass das Modell deutlich schlechter ist als die Baseline von Jannidis et al. (2015). Der Versuch die Erkennungsrate mit einer Liste von Pronomen zu erhöhen zog eine zusätzliche Verschlechterung der Ergebnisse nach sich. Nichtsdestotrotz konnten mit geringem Aufwand sinnvolle Figurennetzwerke in Goethes Wahlverwandtschaften erstellt und visualisiert werden.

Es gibt eine Reihe weiterer Möglichkeiten die Erkennungsrate der Figurenerkennung zu erhöhen. Statt nur die *embeddings* zu verwenden, könnte BERT als Ganzes *fine-tuned* werden, oder von Grund auf ein neues *language model* mit beispielsweise einem Zeitungskorpus, das auch Sprache aus dem 19. Jahrhundert abdeckt, trainiert werden, das die Grundlage für die Figurenerkennung bildet. Zudem könnten weitere Experimente mit regelbasierten Ansätzen gemacht werden, und große Listen mit Vornamen und Berufsbezeichnungen angelegt werden (vgl. Jannidis et al. 2015). Das regelbasierte und probabilistische Labeling kann hier s a n f t e r kombiniert werden[1], indem die Wahrscheinlichkeiten gewichtet statt überschrieben werden.

[1] In Kapitel 4 wurde die Vorhersage für $Pron$ vom Modell gänzlich ignoriert, und ausschließlich die regelbasierte Klassifizierung berücksichtigt.

Akronyme

AppA	Appelativ Abstraktum
AppTdfW	Appelativ Teil der fiktionalen Welt
biLSTM	Bidirektionales Long Short-Term Memory Netzwerk
DROC	Deutsches Roman Korpus
FN	False Negative
FP	False Positive
NER	Named Entity Recognition
RNN	Rekurrentes neuronales Netzwerk
TP	True Positive

Literatur

Akbik, A., D. Blythe, und R. Vollgraf (2018). "Contextual String Embeddings for Sequence Labeling". In: *COLING 2018, 27th International Conference on Computational Linguistics*, S. 1638–1649.

Bergstra, J., B. Komer, C. Eliasmith, D. Yamins, und D. D. Cox (2015). "Hyperopt: A Python Library for Model Selection and Hyperparameter Optimization". *Computational Science & Discovery* 8:1, S. 014008.

Bojanowski, P., E. Grave, A. Joulin, und T. Mikolov (2016). "Enriching Word Vectors with Subword Information". *arXiv preprint arXiv:1607.04606*.

Devlin, J., M.-W. Chang, K. Lee, und K. Toutanova (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *arXiv preprint arXiv:1810.04805*.

Faruqui, M. und S. Pado (2010). "Training and Evaluating a German Named Entity Recognizer with Semantic Generalization". In: *Proceedings of the Konvens 2010, Saarbrücken, Germany*.

Finkel, J. R., T. Grenager, und C. Manning (2005). "Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling". In: *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics*, S. 363–370.

Firth, J. R. (1935). "The Technique of Semantics". *Transactions of the Philological Society* 34:1, S. 36–72.

Goethe, J. W. von (1809). *Die Wahlverwandtschaften*. J. G. Cottaische Buchhandlung, Berlin.

Hänig, C., S. Bordag, und S. Thomas (2014). "Modular Classifier Ensemble Architecture for Named Entity Recognition on Low Resource Systems". In: *Proceedings of the KONVENS GermEval Shared Task on Named Entity Recognition*, S. 113–116.

Hochreiter, S. und J. Schmidhuber (1997). "Long Short-Term Memory". *Neural Computation* 9:8, S. 1735–1780.

Jannidis, F., I. Reger, L. Weimer, M. Krug, M. Toepfer, und F. Puppe (2015). "Automatische Erkennung von Figuren in deutschsprachigen Romanen". *Proceedings of the DHd Conference*.

Jurafsky, D. und J. H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson Prentice Hall, Upper Saddle River, N.J.

Krug, M., L. Weimer, I. Reger, L. Macharowsky, S. Feldhaus, F. Puppe, und F. Jannidis (2017). "Description of a Corpus of Character References in German Novels - DROC [Deutsches ROman Corpus]". *DARIAH-DE Working Papers* 27.

Lample, G., M. Ballesteros, S. Subramanian, K. Kawakami, und C. Dyer (2016). "Neural Architectures for Named Entity Recognition". *Proceedings of NAACL*.

Mikolov, T., K. Chen, G. Corrado, und J. Dean (2013). "Efficient Estimation of Word Representations in Vector Space". In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.

Simmler, S. (2019a). *figur: Figurenerkennung for German literary texts*. URL: https://doi.org/10.5281/zenodo.2592358.

– (2019b). *Neural Network Model for Figurenerkennung in German literary texts*. URL: https://doi.org/10.5281/zenodo.2592324.

van der Maaten, L. und G. Hinton (2008). "Visualizing High-Dimensional Data Using t-SNE". *Journal of Machine Learning Research* 9, S. 2579–2605.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, und I. Polosukhin (2017). "Attention Is All You Need". *CoRR*.

Wikimedia-Foundation (2019). *Sprachen*. URL: https://de.wikipedia.org/wiki/Wikipedia:Sprachen (besucht am 04. 04. 2019).