Automatische Übersetzung von natürlicher Sprache nach SPARQL mit neuronalen Netzen

Bachelorarbeit

im Fachgebiet Informatik

vorgelegt von:	Nicolas Hoferer
	Sebastian Drozdz
Studienbereich:	Informatik
Betreuer:	Dr. Mark Cieliebak
	Prof. Dr. Kurt Stockinger

Erklärung betreffend des selbstständigen Verfassen einer Bachelorarbeit

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:	Unterschriften:
.....	.....
	.....

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Projektarbeiten zu Beginn der Dokumentation nach dem Abstract bzw. dem Management Summary mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zusammenfassung

Die vorliegende Arbeit setzt sich mit der Anwendung von neuronalen Netzen im Bereich der Sprachübersetzung auseinander. Das Ziel der Arbeit ist es, in natürlicher Sprache gestellte Fragen in entsprechende SPARQL-Abfragen zu übersetzten. SPARQL ist eine graphen-basierte Abfragesprache für RDF, welche eine wichtige Rolle beim Arbeiten mit dem Semantic Web spielt. Damit wäre eine Vielzahl von Datenquellen, wie zum Beispiel das DBPedia, einer breiten Öffentlichkeit zugänglich. Die computergestützte Sprachübersetzung bringt jedoch eine Vielzahl von Problemen mit sich. Die Mehrdeutigkeit der natürlichen Sprache bildet eines der Hauptprobleme, warum bis anhin solche Systeme es nicht zur Marktreife geschafft haben.

Wir verwenden einen RNN (Recurrent Neural Network)-basierenden Ansatz, um natürliche Sprache in SPARQL zu übersetzten, da in den letzten Jahren grosse Fortschritte im Bereich des maschinellen Lernens gemacht wurden, und viele Probleme aus der Sprachverarbeitung an das Modell ausgelagert werden. Ausserdem zeigt diese Art von Netzwerken bereits sehr gute Resultate bei klassischen Übersetzungsproblemen. Durch die Anwendung von verschiedenen Massnahmen und Preprocessing-Verfahren konnte ein bestehendes Modell signifikant verbessert werden. Uns standen Trainingsdaten aus dem SQA2018 Challenge und eine dazugehöriger DBPedia-Kopie zur Verfügung.

Das ursprüngliche System erzielte auf den Daten des SQA2018 Challenge anfänglich keine guten Resultate. Dank Anpassungen des Modells, wie zum Beispiel die Verwendung von Attention und Anpassung des Dropouts, konnte die Präzision gesteigert werden. Um die relativ kleine Anzahl von 5000 Trainingsfragen auszugleichen, wurden verschiedene Named Entity Recognition (NER) Methoden eingesetzt, welche dem Modell halfen, die Ressourcen besser zu erkennen, da diese durch generische Platzhalter in der ursprünglichen Frage ersetzt wurden. Diese Erweiterung ermöglicht es auch, Entitäten zu erkennen, welche nicht in den Trainingsdaten vorkommen. Im Verlauf der Arbeiten wurden diese Methoden auch auf die Ontologie (Ontology) und deren Eigenschaften (Properties) angewendet, wodurch die Präzision weiterhin gesteigert werden konnte.

Abstract

This thesis deals with the application of neural networks in the field of language translation. The aim of the work is to translate questions asked in natural language into corresponding SPARQL queries. SPARQL is a graph-based query language for RDF, which plays an important role in working with data from the semantic web. This approach would make a wide variety of data sources, such as DBPedia, accessible to a broader range of users. However, computer-assisted language translation brings several problems with it. The ambiguity of natural language is one of the major problems why such systems have so far not made it to maturity. We use an RNN (Recurrent Neural Network) based approach to translate natural language into SPQARL, as great progress has been made in the field of machine learning in the recent years and many of the language processing problems are tackled by the trained model. Moreover, this type of neural network shows already very good results with classic translation problems. Through the application of various measures and preprocessing procedures, an existing model could be significantly improved. We also had training data from the SQA2018 challenge and a DBPedia dump to our disposal.

Initially, the original system did achieve good results on the SQA2018 Challenge data. Thanks to adaptions of the model, such as the use of an attention mechanism and the adjustment of the dropout, the precision of the model could be increased. To compensate the relatively small number of 5000 training data sets, we used various Named Entity Recognition (NER) methods, which initially served to facilitate the model's translation of resources. In addition, we applied these methods to the ontology and properties of the data sets, which further increased the precision of the system.

Vorwort

An dieser Stelle möchten wir uns bei unseren Betreuern Prof. Dr. Kurt Stockinger und Dr. Mark Cieliebak für ihre Unterstützung, Rat und Führung bedanken. Des Weiteren möchten wir Kathrin Affoltern für ihre Unterstützung bezüglich DBPedia danken. Ausserdem danken wir dem Institut für Angewandte Informationstechnologie der Zürcher Hochschule für Angewandte Wissenschaften für die Bereitstellung der benötigten Rechenressourcen.

Inhaltsverzeichnis

1. Einleitung

Abfragesprachen wie zum Beispiel SQL haben eine lange Tradition in der Informationstechnologie und gehören zu den meistgenutzten Werkzeugen, wenn es um Datenabfrage geht. Zugleich sind sie nicht für den Endbenutzer ausgelegt, da sie ein gewisses Verständnis der zugrundeliegenden Speicherorganisation voraussetzen. Genau in diesem Spannungsfeld versucht diese Arbeit aufzuzeigen, wie mit strukturierten Daten benutzerfreundlicher gearbeitet werden kann.

Die Geschichte des maschinellen Übersetzens (MÜ), findet ihren Ursprung in den Anfängen der Informatik wieder. Bereits mit dem Einzug der ersten relationalen Datenbanksystemen wurden regel- und graphenbasierte Systeme erforscht, welche den Anwender unterstützen sollten Fragen in natürlicher Sprache zu formulieren. Regelbasierten Programme waren eine der ersten untersuchten Ansätze in diesem Bereich. Die Mehrdeutigkeit der Sprache hat jedoch bald gezeigt, dass solche rigiden Systeme keine Lösung für dieses Problem darstellten konnten. Beispielbasiertes MÜ bildet die nächste Entwicklung in der langen Geschichte vom MÜ. Sie bauen einen Übersetzungsspeicher mit häufig wiederkehrende Satzfolgen auf, welcher mithilfe von statistischen Methoden die Ähnlichkeit zu dem Quelltext berechnet und entsprechenden Kombinationen generiert. Statistische MÜ Verfahren sind wiederum eine Erweiterung des beispielbasierten Ansatzes. Hierzu werden grosse Textkorpusse, vorzugsweise bestehend aus Protokollen und amtlichen Dokumenten aus mehrsprachigen Ländern, einander zugeordnet. Die daraus resultierenden Übergangsregeln bilden die Basis solcher Systeme. Dank der Vorschritte im Bereich des maschinellen Lernens in den letzten Jahren, profitierte auch die automatisierte Sprachübersetzung von dieser Entwicklung. Eine Weiterentwicklung davon sind Übersetzungssystemen welche nicht eine andere natürliche Sprache als Ziel haben, sondern Abfragesprachen wie SQL, Prolog oder SPARQL. Das zugrundeliegende Modell dieser Arbeit ist ein solches System.

1.1. Motivation

Daten spielen heute in fast allen Bereichen des modernen Lebens eine wesentliche Rolle. Informationssysteme jeglicher Art helfen die Informationen zu strukturieren

und dessen Inhalt wieder auffindbar zu speichern. Durch die Menge der Daten sind Experten und Spezialisten gefragt, die wiederum das Wissen haben wie die Daten aus den jeweiligen Systemen ausgelesen werden können. An dieser Stelle setzt unserer Arbeit an welche mit Hilfe des maschinellen Lernens einen Agenten beschreibt, der im Stande ist eine Frage in natürlicher Sprache in die Abfragesprache SPARQL zu übersetzen.

Meist werden die gleichen Mechanismen wie bei der Sprachübersetzung verwendet. Die Freiheit und Möglichkeit, Anfragen an das jeweilige Informationssystem zu stellen, sollte wieder an dessen Anwender zurückgegeben werden. Diese sind oft auf die Hilfe der genannten Experten angewiesen. Da in naher Zukunft Daten mit einer sehr hohen Wahrscheinlich noch eine wichtigere Rolle spielen werden, ist es umso wichtiger das deren Anwendung und Erkundung möglichst auch von einem Endbenutzer ohne spezielle Kenntnisse durchgeführt werden kann. Die computergestützte Übersetzung von Anfragen in natürlicher Sprache und in die jeweilige Abfragesprache wäre ein grosser Schritt in diese Richtung. Unser Agent versucht die Lücke zwischen Endanwender und dem Informationssystem zu schliessen. An dieser Stelle soll erwähnt werden das ein solches System auch relativ einfach für eine andere Abfragesprache wie zum Beispiel SQL adaptiert werden kann. Wir haben uns für SPARQL entschieden, da zeitgleich die SQA2018 Challenge ausgeschrieben wurde, für welche wertvolle Trainingsdaten zur Verfügung stehen. Eine breite Abdeckung von verschiedenen Abfragesprachen ist essenziell für die Hilfe des Anwenders und darum sollten auch andere Abfragesysteme wie SQL oder Prolog untersucht werden. Das Modell nutzt den gleichen Aufbau wie Systeme, welche natürliche Sprachen übersetzen die bereits sehr gut erforscht sind. Ein grosser Vorteil von Abfragesprachen ist ein kompaktes Vokabular und eine weniger komplexe Syntax als bei natürlichen Sprachen. Auf der anderen Seite verzeiht eine Abfrage Sprache wenig Fehler ohne das die Aussage noch vom System verstanden werden kann oder das Resultat der Ausführung die korrekten Ergebnisse liefert. In diesem Spannungsfeld ist eine wichtige Aufgabe ein bestehendes System der Sprachübersetzung für Abfragesprachen zu erweitern zumachen.

1.2. Ziel der Arbeit

Wie bereits in der Einleitung erwähnt, versucht diese Arbeit die Lücke zwischen einer Abfragesprache wie SPARQL und dem Endanwender zu schliessen. Da die Anfrage in natürlicher Sprache formuliert werden kann, sind keine speziellen Kenntnisse nötig um die gewünschten Informationen an das System zu formulieren. Das Modell basiert auf bereits erprobten neuronalen Netzwerken aus der Sprachübersetzung.

1.3. Aufbau der Arbeit

Das Dokument ist wie folgt aufgebaut, Kapitel zwei fasst ähnliche und hilfreiche Arbeiten im Zusammenhang mit dem NL-to-SPARQL zusammen. Die grundlegenden Konzepte und Techniken des maschinellen Lernens, die sich ebenfalls auf die Problemstellung beziehen, speziell im Bereich von DBPedia, werden in Kapitel drei erläutert. Das vierte Kapitel widmet sich der Zusammenfassung des verwendeten Modells und der verwendeten Extrahierungsmechanismen. Kapitel fünf befasst sich mit den in dieser Arbeit verwendeten Daten. Das sechste Kapitel beschreibt die durchgeführten Experimente und die darin erhaltenen Resultate. Die Ergebnisse werden in Kapitel sieben beschrieben und diskutiert, anschliessend wird in Kapitel acht ein Ausblick bezüglich der Weiterentwicklung des Systems gegeben.

2. Verwandte Arbeiten

Im Rahmen der Vorbereitung dieser Arbeit wurden diverse bestehende Ansätze und relevante Literatur untersucht. Dabei haben wir nicht nur Ansätze berücksichtigt, die natürliche Sprache zu SPARQL umwandeln, sondern auch Ansätze, welche natürlich sprachige Fragen zu SQL umwandeln oder auch Literatur zur Problemstellung NL-to-SPARQL selbst. Im folgenden werden wir die verschiedenen Ansätze beziehungsweise Literatur vorstellen.

2.1. SPARQL as a Foreign Language

Beginnen möchten wir mit dem Paper, welches auch die Grundlage dieser Arbeit darstellt. Tommaso Soru et al. [29] stellten ein System vor, welches auf einem Seq2Seq-Modell für Neural Machine Translation basiert. Das Seq2Seq-Modell wird verwendet um die natürlichsprachigen Fragen in SPARQL umzuwandeln. In dieser Arbeit wurde die öffentliche Implementierung von tensorflow des Seq2Seq-Modells verwendet, am Modell wurden lediglich die gewählten Hyperparameter verändert, die Hauptarbeit liegt darin die Daten sinnvoll aufzubereiten, sodass das Modell auf diesen trainiert werden kann.

Zuerst werden manuell, sogenannte *Templates* erstellt, wie in Beispiel 2.1.1 gezeigt. Dabei besteht ein Template aus maximal drei Variablen. Der Aufbau des Templates ist wie folgt, zuerst kommen die Typen der drei Variablen, gefolgt von der Vorlage für die natürlichsprachige Frage, dem Querytemplate und dem *Generatorquery*, jeder dieser Teile wird jeweils mit einem Semikolon voneinander getrennt. Die Templates selbst werden immer auf einer neuen Zeile geschrieben.

Beispiel 2.1.1. *dbo:TelevisionShow;dbo:TelevisionShow;;does <A> have more episodes than <B>?;ask where <A> dbo:numberOfEpisodes ?a . <B> dbo:numberOfEpisodes ?b . filter (?a > ?b) ;select distinct ?a, ?b where ?a dbo:numberOfEpisodes ?x . ?b dbo:numberOfEpisodes ?y ;*

Anschliessend wird für jedes Template mithilfe des Generatorquery ein neuer Query generiert. Mit dem Generatorquery werden passende Entitäten für das entsprechende Template abgefragt, dabei werden Ressourcen abgefragt, welche den selben Typ haben, wie die im Template angegebenen Ressourcen. Ausserdem werden nur Templates verwendet, für welche mithilfe des Generatorquery mindestens 100 verschiedene Entitäten pro Variable gefunden werden, weiter werden maximal 300 Abfragen pro Template gesammelt.
Ein Beispiel des Generatorquery und einer daraus resultierenden Abfrage ist in Listing 2.1 respektive Listing 2.2 gezeigt.

Listing 2.1: Beispiel Generator Query

```
select distinct (?x) where {
    ?x dbo:abstract []
}
```

Listing 2.2: Beispiel neu generierter Query

```
select distinct (?x) , (str(?labx) as ?lx) where {
  ?x a dbo:Monument .
  ?x rdfs:label ?labx .
  FILTER(lang(?labx) = 'en') .
  ?x dbo:abstract []
}
```

Die gefundenen Entitäten werden in die natürlichsprachige Frage und den Query aus dem Template eingesetzt und somit wird der Datensatz generiert.

Nach der Generierung der Daten wird das Vokabular aus den Daten erstellt, dabei werden die SPARQL-Abfragen zuerst in Token umgewandelt und dann wird der Standard Vokabularbuilder aus Tensorflow verwendet. Eine genauere Beschreibung, wie die Abfragen zu Token umgewandelt werden folgt im Abschnitt 4.2, da dieser Vorgang auch in unserer Arbeit verwendet wurde.

Erst dann werden die generierten Daten in Trainings-, Validierungs- und Testdaten aufgeteilt (80%, 10%, 10%).
Danach folgt das Training und anschliessend kann das Modell für die Inferenz verwendet werden.

Grenzen dieses Ansatzes:
Dieser Ansatz baut das Vokabular basierend auf den Training- und Validierungsdaten auf, somit sind schlechtere Ergebnisse zu erwarten, wenn für das Vokabular nur

die Trainingsdaten verwendet werden. Des Weiteren wird durch den Generator das Modell nicht viel besser, da das Modell lediglich die gleichen Daten sieht, jedoch mit verschiedenen Entitäten. Dies führt wahrscheinlich zu einer besseren Umwandlung, wenn die selben Abfragen von den Trainingsdaten auch in den Validierungsdaten vorkommen und sich lediglich die Entitäten unterscheiden.

Ausserdem haben die Autoren zur Evaluierung die BLEU Metrik verwendet, diese eignet sich nur bedingt als Mass für die Güte einer generierten SPARQL-Abfrage. Geeigneter wäre das verwenden von einer auf SPARQL angepassten Genauigkeit, die hier erwähnten Metriken werden im Abschnitt 3.3 genauer erläutert.

In Abschnitt 4.1 werden die Hyperparameter dieses Modells erläutert, da dieses System die Grundlage dieser Arbeit bildet. Im Abschnitt 6.1 werden die Experimenten zu diesem Ansatz erläutert und die Auswirkungen der oben erwähnten Grenzen aufgezeigt.

2.2. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning

Die Arbeit von Zhong et al. [35] beinhaltet zwei Veröffentlichungen. Ein sogenanntes *Seq2SQL*-Modell sowie ein Datensatz für die Generierung von SQL Abfragen.
Das Modell Seq2SQL basiert auf einem Seq2Seq-Modell und einer Reinforcement Learning Strategie, welche für die Bewertung des Seq2Seq-Modells verwendet wird.

Abbildung 2.1 zeigt die Übersicht des Modells. Das Modell bekommt als Input eine natürlichsprachige Frage und die Spaltennamen der Tabelle. Dann wird die entsprechende SQL Abfrage generiert und anschliessend noch im Training auf der Datenbank abgefragt. Das Resultat der Ausführung wird zu einem Reward umgewandelt für den Reinforcement Learning Algorithmus.

Input: (Frage und Tabelle)
Der Input für das Modell ist die Konkatenation der Spaltennamen (wird benötigt für die Selektion und für die WHERE Klausel), die Frage (wird für die gesamte Abfrage benötigt) und das Vokabular der SQL Sprache (wie zum Beispiel SELECT, COUNT, etc.) Abbildung 2.2 zeigt Beispieldaten für den Input und Output. Der Input wären hierbei die Token von den Spaltennamen “Pick”, “#”, “CFL”, “Team” etc.; die Token für die Frage bestehend aus “How”, “many”, “CFL”, “teams”, etc.; die SQL Token bestehend aus “SELECT”, “WHERE”, “COUNT”, “MIN”, “MAX” etc.

Der Output des Modells ist eine SQL Abfrage, hierbei besteht das Output Vokabular aus dem Tabellen Schema, der natürlichsprachigen Frage und den SQL Schlüsselwörter.

Das Modell ist ähnlich wie ein *pointer network* mit *augmented inputs*, ein Pointer Network, ist ähnlich wie ein Seq2Seq-Modell mit Attention, aber anstatt eine Sequenz in eine andere zu übersetzen, wird eine Sequenz von Zeigern auf Elemente der Eingangssequenz produziert [33].

Modell
Der Encoder besteht aus zwei bidirektional LSTM Schichten und der Input wird mittels Embeddings der LSTM Schichten übergeben, dannach folgt das Pointer Network. Anschliessend folgt der Decoder mit zwei Schichten unidirektionalen LSTM's. Der Decoder generiert einen skalaren Attention Wert für jede Position der Input Sequenz und dann wird das Token, mit der höchsten Wahrscheinlichkeit als nächstes Token gewählt.

Normalierweise besteht eine SQL-Abfrage aus drei Teilen, Aggregation (*COUNT* oder *NONE*), *SELECT* und *WHERE*, deshalb klassifiziert das Netzwerk zuerst, ob eine Aggregation vorhanden ist, dann zeigt es mithilfe des Pointer Netzwerk auf die Spalten für den SELECT Teil der Abfrage. Ausserdem wird das Pointer Netzwerk für die Generierung der WHERE Klausel benutzt.

Bei der WHERE Klausel hat die Reihenfolge keinen Einfluss auf die Antwort, deshalb wird hier für die Berechnung der Kosten ein Reinforcement Learning Ansatz verwendet, indem die generierte Abfrage ausgeführt wird und das Resultat zur Bestimmung der Kosten verwendet wird.

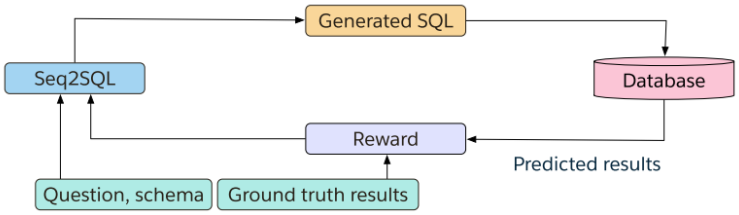


Abbildung 2.1.: Architekturübersicht vom Seq2SQL Modell [35]

.

Resultate: Die Autoren berichten von einer Verbesserung der *Execution Accuracy* von 35.9 % auf 59.4 % und eine Verbesserung der *Logical form accuracy* von 23.4 % auf 48.3 %, wobei die Execution Accuracy angibt, ob das Resultat der Abfrage korrekt ist und die Logical form accuracy gibt an, ob die Form der Abfrage korrekt ist.

WikiSQL: Bei WikiSQL handelt es sich um einen Datensatz und ein Softwaresystem, der Datensatz besteht aus Tabellen, Fragen, SQL-Abfragen und dem Resultat der Abfragen. Nutzt man es als System, erwartet es als Eingabe eine Tabelle und eine Frage, dann wird als Ausgabe die korrekte SQL Abfrage und das Resultat der generierten Abfrage zurückgegeben. Abbildung 2.2 zeigt ein Beispiel Ein- und Ausgang.

Abbildung 2.2.: Beispiel Input und Output für WikiSQL [35]

.

2.3. Question Answering Over Linked Data: What is Difficult to Answer? What Affects the F scores?

M. Saleem et al. [22] untersuchten die Auswirkungen von verschiedenen Fragen auf die Generierung der SPARQL-Abfrage. Ihre Analyse verwendet die Daten der QALD-6 Challenge, ihre Arbeit selbst, wurde im Rahmen von QALD-7 veröffentlicht.

Im folgenden ist eine kurze Auflistungen ihrer Resultate.

Resultate:
Die Komplexität der verschiedenen Fragekategorien zeigt, bei welcher Fragekategorie, welcher F-score erreicht wurde, die Resultate sind in Tabelle 2.3 gezeigt. Ausserdem sind die Systeme unterschiedlich gut, es gab kein teilnehmendes System, welches für jede Fragekategorie, die besten Ergebnisse erzielte.

Effekt der Anzahl von Triple Patterns und Keywords:
Je mehr Triple Patterns, desto schlechter die Performanz der Systeme.
Bezüglich der Keywords gilt generell, je mehr Keywords, desto schlechter ist die Performanz der Systeme, aber bei drei Keywords ist die Performanz besser wie bei zwei.

Effekt der Antwortlänge und des Antworttyp:
Je mehr antworten die Fragen haben, desto besser ist der F-score.
Die Antworttypen mit ihren F-scores werden in Tabelle 2.2 gezeigt.

Fragenkategorie	Makro F-score
When	0.57
In which	0.55
Who	0.48
What	0.44
Which	0.43
Where	0.38
Give me	0.34
How many	0.32

Tabelle 2.1.: Absteigend mit Makro F-score pro Fragenkategorie

Antworttyp	F-score
date	0.49
resource	0.46
boolean	0.38
string	0.36

Tabelle 2.2.: Absteigend mit Makro F-score pro Antworttyp

Der Effekt von Aggregation und der Form der SPARQL-Abfrage werden in Tabelle 2.3 respektive 2.4 gezeigt.

Aggregation	F-score
Mit	0.22
Ohne	47

Tabelle 2.3.: F-score Vergleich mit und ohne Aggregation

SPARQL Form	F-score
ASK	0.38
SELECT	0.43

Tabelle 2.4.: F-score Vergleich SPARQL Form

2.4. End-to-end Representation Learning for Question Answering with Weak Supervision

Dieser Ansatz von D. Sorokin et al. [28] ist in mehrere Phasen unterteilt. Er beinhaltet ein Neuronales Netzwerk für die Bewertung der Abfragen.

Entitiy linking Die erste Phase ist das *Entity linking*, hierbei wird der Input zu Token umgewandelt und part-of-speech tags werden hinzugefügt, mithilfe des StanfordCoreNLP Bibliothek [13]. Anschliessend werden *Token Fragmente* mithilfe von Regulären Ausdrücken extrahiert, welche alle Sequenzen von Substantiven und ihren modifieren auswählen. Dann werden die extrahierten Fragmente mittels eines Lookup in Wikidata (Lookup von Wikipedia Entitäten) gesucht, dies geschieht anhand der *Bezeichnung* der Wikidata Einträge. Die gesammelte Liste wird sortiert, dabei wird als Sortierungskriterium die Levenshtein Distanz zwischen dem Fragment und der Bezeichnung des Eintrags, zusammen mit dem Integer Teil der ID des Eintrags verwendet. Anschliessend werden die besten Kandidaten jedes Fragments für das finale Linking verwendet. Zum Beispiel, bei der Frage "What was the first Queen album?" werden die Entitäten "Queen" und "album" erkannt.

Iterative representation generation Bei dem zweiten Schritt handelt es sich um eine Generierungsprozedur, welche die Art der Repräsentation bestimmt. Semantische Repräsentationen bestehen aus einer Fragenvariable Node (shaded circle), Entitäten mit einer fixen Wikidata ID (rectangles), constraints (rounded rectangles) und Wikidata Beziehungen (labeled arrows). Dann werden die semantischen Repräsentationen mithilfe von Wikidata evaluiert, indem alle Entitäten, welche die Fragenvariable Node ersetzen können verwendet werden, sodass alle Beziehungen und Bedingungen erfüllt sind, dies bedeutet, dass die Fragenvariable Node nun eine Antwort auf die Frage enthält. Das iterative Generieren beginnt mit einer leeren Repräsentation, welche nur eine Fragevariable enthält, dann werden verschiedene Aktionen auf die Repräsentation angewendet. Die möglichen Aktionen umfassen das Hinzufügen einer Beziehung oder einer Bedingung. Zuerst wird die *ADD_Relation* Aktion ausgeführt, da diese als einziges möglich ist. Die Generierung wird solange durchgeführt bis keine Aktion mehr möglich ist. Falls eine Repräsentation keine Antwort produziert, wird diese übersprungen. Zusätzlich wird nach jeder Iteration Beamsearch eingesetzt, dabei wird jede Kandidat Repräsentation mit dem *Neural Scorer* bewertet und die zehn besten werden für die nächste Iteration ausgewählt. Abbildung 2.3 zeigt die Repräsentation für verschiedene Schritte für die Frage "What was the first Queen album?".

Neural Scorer Der Neural Scorer basiert auf einem *Convolutional Neural Network*, er wird verwendet, um eine Kandidaten RepresentatRepräsentationion zu bewerten.

Mithilfe des Convolutional Neural Network wird die Frage und die Kandidaten Repräsentation in einen Vektor transformiert. Der Vektor der Frage wird dann mit dem der Vektor der Kandidatenrepräsentation verglichen mittels der *Kosinus-Ähnlichkeit*, um so einen Score zu erhalten.

Die Evaluierung dieses Modells wird zwar auf dem QALD7 Datensatz ausgeführt, aber das Modell wurde auf Daten des Autors trainiert. *upper bound*, entspricht einem Neural Scorer der jedes mal der korrekten Repräsentation als höchstes bewertet. Abbildung 2.4 zeigt die Tabelle der Resultate der ursprünglichen Arbeit. Es ist zu sehen, dass mehr als 50% der Fragen in SQL transformiert werden konnten, wobei das System einen F1-Score von 0.364 erreicht.

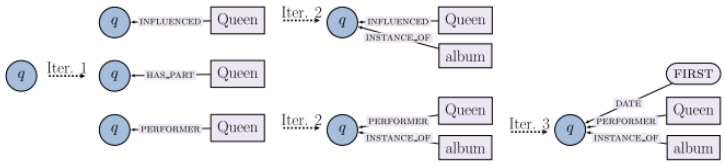


Abbildung 2.3.: Beispiel Iterationen für die Kandidatenrepräsentation. [28]

	Processed	Right	Partially right	Avg. Precision	Avg. Recall	F1	Global F1
Our system	80	25	36	0.3507	0.4318	0.3640	0.2912
Upper bound	80	47	30	0.7602	0.8980	0.7266	0.5812

Abbildung 2.4.: Resultate der ursprünglichen Arbeit. [28]

2.5. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning

Xu et al. veröffentlichten SQLNet basierend auf Seq2SQL. Dabei versuchen sie mithilfe von zwei neuen Techniken, das Probleme, dass die Reihenfolge bei Seq2Seq-Modellen eine Rolle spielt, lösen [34]. Dieses Problem ergibt sich in der WHERE Klausel, da die Reihenfolge hier nicht das Resultat der Abfrage beeinträchtigt, aber syntaktisch ergeben sich verschiedene Abfragen.

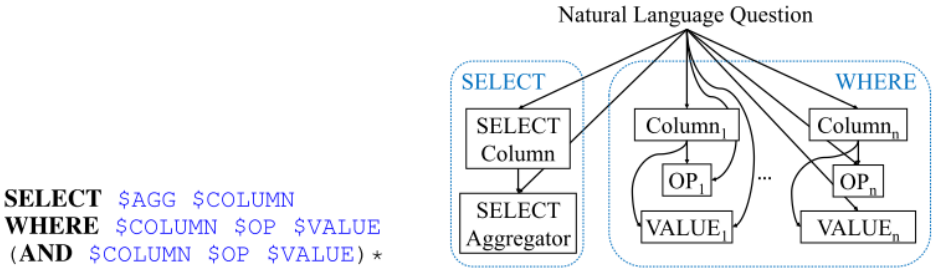
Beispiel 2.5.1. *SELECT result WHERE score="1-0" AND goal=16*
vs.
SELECT result WHERE goal=16 AND score="1-0"

Die Autoren behaupten, das bei einem Seq2Seq-Modell immer implizit die Reihenfolge eine Rolle spielt, um das zu vermeiden verwenden sie einen skizzenbasierten

Ansatz. Ihr Modell *SQLNet* generiert den Inhalt jedes Slots, als Slot wird hier jeder Teil der SQL Abfrage bezeichnet, welcher generiert werden muss.

Damit dies möglich ist führen sie zwei neuartige Techniken ein, *sequence-to-set* (zur Vorhersage einer ungeordneten Menge von Randbedingungen) und *column-attention* (zur Erfassung, der in der Skizze definierten Abhängigkeitsbeziehung bei der Vorhersage)

Abbildung 2.5 zeigt die Syntax einer Skizze und die Abhängigkeit in dieser.



(a) SQL Sketch

(b) Graphical illustration of the dependency in a sketch

Abbildung 2.5.: Syntax einer Skizze und die Abhängigkeiten der Slots. [34]

Sequence-to-set

Die Intuition hinter *sequence-to-set* ist, dass die Spaltennamen, welche in einer WHERE Klausel erscheinen, eine Untermenge des vollständigen Satzes aller Spaltennamen bilden, deshalb muss nur vorhergesagt werden, welche Spalten im WHERE Teil der Abfrage verwendet werden müssen.

Column-attention

Bei der Vorhersage einer Spalte wird ein Attention Mechanismus verwendet, welcher einen Score über die Token der Frage in natürlicher Sprache berechnet, um die Wahrscheinlichkeit der Auswahl der korrekten Spalte zu maximieren. Zum Beispiel ist das Token “number” relevanter für die Vorhersage der Spalte “No.”, als andere Token aus der Frage.

Für die Vorhersage des Wertes für die in der WHERE Klausel verwendeten Spalte wir ein Seq2Seq Ansatz verwendet, weil dort die Reihenfolge wiederum eine Rolle spielt.

Der grösste Nachteil dieser Implementierung ist, dass nur einfache Abfragen möglich sind. Es gibt zum Beispiel keine Möglichkeit wie “JOIN’s” beschrieben werden können. Somit wird die Mächtigkeit sehr stark eingeschränkt.

2.6. Natural Language Question Answering over RDF — A Graph Data Driven Approach

Der hier beschriebene Graphen-basierete Ansatz nutzt die Tatsache, dass RDF Entitäten sich als Graphen abbilden lassen. Dabei repräsentiert das Prädikat die Kante während das Objekt und Subjekt durch die anliegenden Knoten beschrieben wird. Der Ansatz von Lei Zou et al.[36] verfolgt das Ziel eine Anfrage in natürlicher Sprache N mit Hilfe eines Phrasen Wörterbuchs (para-phrase dictionary) in eine SPARQL-Abfrage umzuwandeln.

Die Umwandlung erfolgt in zwei Schritten. Zuerst wird die Frage analysiert (question understanding) anschliessend wird die generierte SPARQL-Abfrage ausgewertet (query evaluation). Dabei wird aus der in natürlicher Sprache vorliegenden Anfrage N mit Hilfe des Stanford Parser [13] ein Abhängigkeitsbaum abgebildet. Im nächsten Schritt werden jedem Knotenpunkt im Abhängigkeitsbaum eine Liste mit möglichen Kandidaten aus dem Phrasen Wörterbuch zugewiesen. Die Darstellung 2.5 soll das Verständnis für die Funktionen dieses Wörterbuchs näher bringen. Zu sehen sind die Begriffe und ihre entsprechenden Attribute aus DBPedia. Die letzte Spalte bildet die Wahrscheinlichkeit das diese Abbildung relevant sein könnte, diese dient dem Algorithmus den bestmöglichen Kandidaten auszuwählen.

Beziehungssätze	**Prädikate Pfade**	**Vertrauens-würdigkeit**
“be married to“	<spouse>	1.0
“play in“	<staring>	0.9
“play in“	<director>	0.5
“uncle of“	<has Child> -> <has Child> -> <has Child>	0.8

Tabelle 2.5.: Beispiel Auszug aus einem Phrasen Wörterbuch. [36]

Die Abbildung 2.6 zeigt eine Übersicht der Architektur. Diese veranschaulicht wie die uneindeutigen Zuweisungen mit Hilfe der Kandidatenliste während der Verarbeitung eingesetzt werden. Tatsächlich stellt jede Kante zusammen mit den beiden Endpunkten eine semantische Beziehung dar. Ein semantisches Abfragediagramm wird wie folgt aufgebaut. Zuerst werden alle semantischen Beziehungen in N extrahieren, die jeweils einer Kante in QS entsprechen, wenn die beiden semantischen Beziehungen ein gemeinsames Argument haben, teilen sie sich einen Endpunkt in QS.

Die semantische Relationen-Extraktion basiert auf dem bereits erwähnten Paraphrasen Wörterbuch. Die Beziehungsphrase ist eine Folge, die zwischen einem Paar von

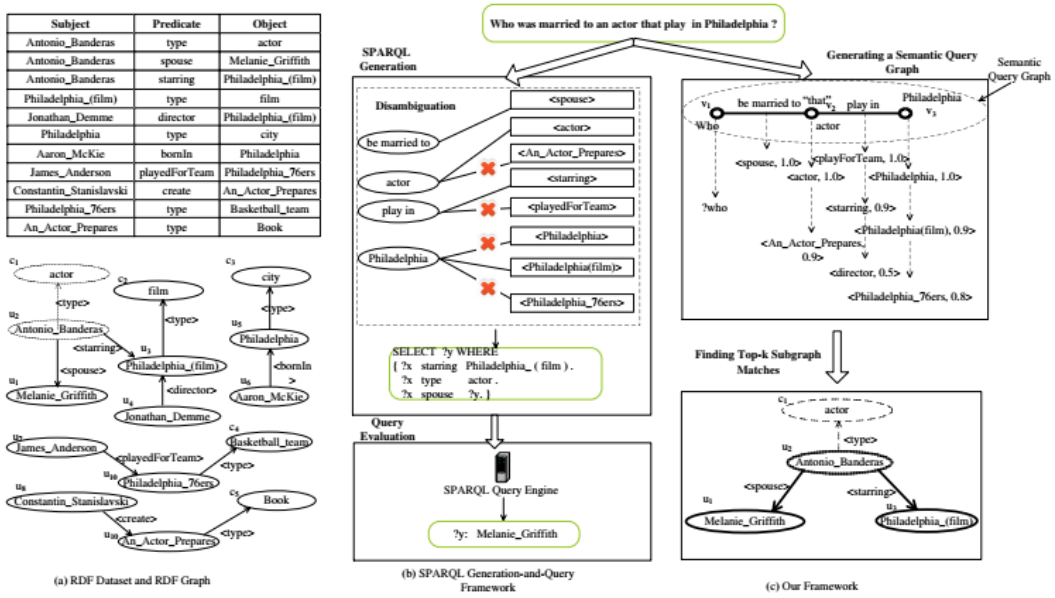


Abbildung 2.6.: Architektur des Graphen basierten Ansatzes. [36]

Entitäten in einem Satz auftritt, wie z.B. “verheiratet sein mit” und “einspielen”. Dazu wird ein Paraphrasen Wörterbuch D, wie in 2.5, erstellt, um Beziehungsphrasen auf einige Kandidatenprädikate oder Prädikats-Pfade abzubilden. In diesem Beitrag wird nicht diskutiert, wie man Beziehungsphrasen extrahiert. Diese Arbeit geht davon aus, dass die Beziehungsphrasen und ihre Unterstützungssätze gegeben sind. Die Aufgabe in der Offline-Verarbeitung ist es, die semantische Äquivalenz zu finden.

Die Autoren bestätigen, dass die graphenorientierte Verarbeitung einen grossen Vorteil beim Erkennen von Beziehungen zwischen den einzelnen Ressourcen bietet. Auf der anderen Seite ist das statische Phrasen Wörterbuch ein überholter Ansatz. Ein neuronales Netzwerke an dieser Stelle könnte Abhilfe schaffen.

3. Theoretische Grundlagen

Der erste Teil dieses Kapitels erklärt die grundlegenden Terminologien und theoretischen Konzepte, anschliessend folgen spezifischere Konzepte für die Problemstellung *NL-to-SPARQL*. Dabei werden die Grundlagen anhand der Problemstellung erläutert und gelten möglicherweise nicht exakt für andere Probleme.

3.1. Linked Data

In diesem Abschnitt geben wir eine kurze Zusammenfassung der verknüpften Daten und der damit verbundenen Abfragesprache *SPARQL*.

Resource Description Framework Das Resource Description Framework (RDF) beschreibt Ressourcen (verschiedene Dinge wie zum Beispiel Personen) und die Beziehungen zwischen ihnen. Es wurde entwickelt um die Probleme des HTML-basierten Internets zu lösen, welche nachfolgend erläutert werden. HTML ist eine Dokumentenbeschreibungssprache, dies führt dazu, dass es nur Daten enthält, welche erst beim Browser des menschlichen Benutzers erzeugt werden. Diese Daten lassen sich in ihrer Form nur sehr schlecht von Computern verarbeiten. Die enthaltenen Informationen werden nicht in einer für Computer geeigneten Struktur gespeichert, dies stellt ein Problemen für zum Beispiel Suchmaschinen, Einkaufsagenten und Datamining dar. [5]

Die erste Lösung dieses Problems wurde mithilfe von Extensible Markup Language (XML) erstellt, somit wurden die Informationen von der Darstellung getrennt und eine freie Definition der Begriffe und Zusammenhänge wurde gegeben. Die Folgen waren aber mehrdeutige Schreibweisen wie in Listing 3.1 gezeigt, ausserdem war die Bedeutung der verschiedenen Daten nicht definiert und nur hierarchische Strukturen konnten abgebildet werden. Die Syntax des XML wurde dann durch XML-Schema festgelegt, ein Beispiel eines XML-Schema ist in Listing 3.2 dargestellt. Aber auch XML-Schema haben keine Definition der Semantik, wie zum Beispiel, was bedeutet der <preis> eines Buches und wie ist dieser definiert. [5]

Listing 3.1: Beispiel für Mehrdeutigkeit in XML. [5]

```
<buch>
  <titel>Studentenkochbuch</title>
  <autor>Heinz Otto</autor>
  <preis>24 Euro</preis>
</buch>

<buch titel="Studentenkochbuch" autor="Heinz Otto" preis="24 Euro" />
```

Listing 3.2: Beispiel für ein XML Schema. [5]

```
<xs:element name="buch">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="titel" type="xs:string"/>
      <xs:element name="autor" type="xs:string"/>
      <xs:element name="preis" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Ein RDF Dokument besteht aus Statements, welche wiederum aus drei Teilen bestehen, ein Statement beschreibt dabei immer eine Eigenschaft einer Ressource oder eine Beziehung. Die verwendeten Bezeichner müssen eindeutig sein, dies wird mithilfe der URI erreicht. [24]

Subjekt: Die zu beschreibende Ressource, wird durch eine URI spezifiziert.
Prädikat: Die Eigenschaft der Ressource, welche durch dieses Statement beschrieben werden soll, wird ebenfalls durch eine URI spezifiziert.
Objekt: Der Wert der Eigenschaft, kann durch eine URI oder ein *Literal* spezifiziert werden, wobei ein Literal zum Beispiel ein Text, eine Zahl oder ein Datum ist.

Abbildung 3.1 zeigt ein Beispiel eines RDF-Triples, solch ein Triple besteht ebenfalls aus den selben drei Teilen und kann gut visualisiert werden. Mehrere solcher Triples können zusammen als Graph dargestellt werden, wie in Abbildung 3.2 gezeigt.

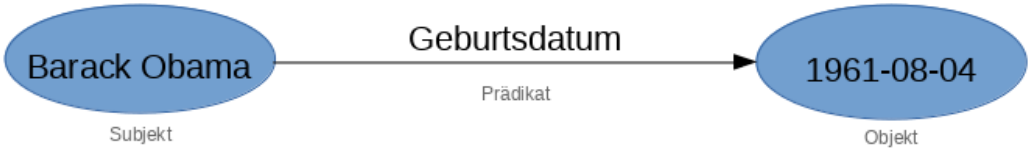


Abbildung 3.1.: Beispiel eines RDF Triples.

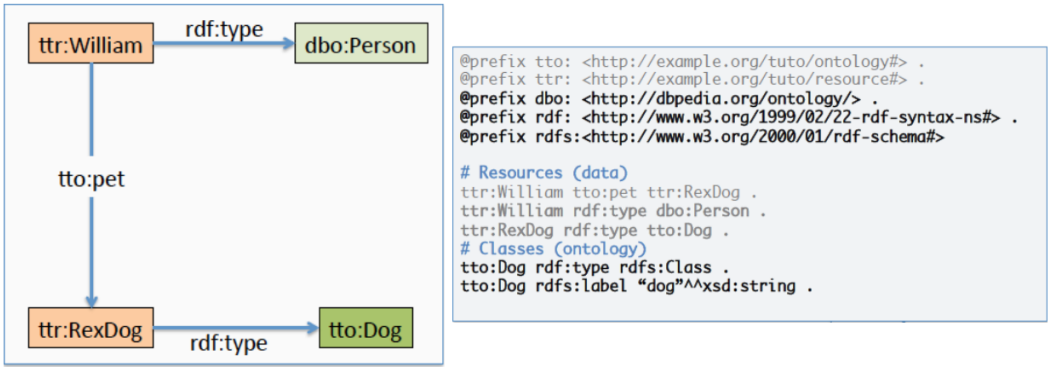


Abbildung 3.2.: Beispiel eines RDF Graphen und der dazugehörigen Statments. [24]

SPARQL

SPARQL ist eine RDF Abfragesprache und steht für *SPARQL Protocol and RDF Query Language*. Sie wird vom World Wide Web Consortium (W3C) definiert. Mithilfe von SPARQL kann man abfragen an einen RDF-Triplestore senden, ein Triplestore ist eine Datenbank die speziell für RDF-Daten konzipiert wurde, wie zum Beispiel GraphDB[1].

3.2. DBPedia

DBPedia ist ein Gemeinschaftsprojekt verschiedener Forschungseinrichtungen mit dem Ziel Informationen aus Wikipedia zu extrahieren und strukturiert zur Verfügung zu stellen.

Eine *Ontologie* kann als eine Konzeptualisierung oder ein Datenmodell einer Domäne angesehen werden. Sie beinhaltet *Klassen* und *Eigenschaften*. Eine Klasse ist eine Gruppierung von Ressourcen, jeder Klasse könnnen Eigenschaften zugeteilt werden, somit hat jede Ressource einer bestimmten Klasse die Eigenschaften dieser Klasse, ausserdem sind die Klassen hierarchisch aufgebaut, was dazu führt, dass die Eigenschaften vererbt werden. In der DBPedia Ontologie ist die Klasse "Ding" (englisch: "Thing"), die Basisklasse aller Klassen, somit erbt jede andere Klasse von dieser. Beispiel 3.2.1 zeigt die Klassenhierarchie für die Klasse "(Film-)Darsteller" (englisch: "Actor")

Beispiel 3.2.1. *Thing -> Agent -> Person -> Artist -> Actor*

Ausserdem können Ressourcen auch mehrere Klassen besitzen, so hat zum Beispiel die Ressource "Brad Pitt" unteranderem die Klasse "Actor" und "Producer". [24]

[1] http://graphdb.ontotext.com/

In unserer Arbeit verwenden wir die DBpedia Version 2016-04, diese umfasst circa 4'233'000 verschiedene Ressourcen. Sie setzt sich in der englischen Sprache aus 754 Klassen und fast 60'000 Eigenschaften zusammen.

Durch die Vielzahl von Eigenschaftsbeschreibungen gibt es oft mehrere Uniform Resource Identifiers, welche schlussendlich doch die gleiche Eigenschaft beschreiben. Wir haben unter anderem dieses Beispiel 3.2.2 bei der Auswertung der Eigenschaften gefunden.

Beispiel 3.2.2. *http://dbpedia.org/property/alternateName http://dbpedia.org/property/alternatetiveName http://dbpedia.org/property/alternatetiveNames http://dbpedia.org/property/altname*

Die Organisation der DBPedia Daten ist aus unserer Sicht teilweise sehr schlecht gelöst, da es im `http://dbpedia.org/ontology` Pfad Einträge gibt, die sowohl eine Ontologie wie auch die Eigenschaft von Ressourcen beschreibt. Sie unterscheiden sich lediglich durch ihre Schreibweise. Ontologien werden in der Regel gross geschrieben, die gleichnamige Eigenschaft klein. Folgendes Beispiel 3.2.3 zeigt die Problematik anhand der Rubrik Sport auf.

Beispiel 3.2.3. *http://dbpedia.org/ontology/Sport → beschreibt die Ontologie* ***Sport*** *http://dbpedia.org/ontology/sport → beschreibt die Eingenschaft* ***Sport***

3.3. Performanz-Metriken

Durch die Komplexität und Diversität zwischen verschiedenen Maschinellen Lernalgorithmen ist die Notwendigkeit einer universellen Leistungskennzahl entstanden. Ihre inhärente Komplexität macht den direkten Vergleich schwierig. Darüber hinaus muss die Performance-Metrik es den Entwicklern ermöglichen, zu erkennen, welche Änderungen zu einer Erhöhung der Qualität führen. Ausserdem muss die Leistungskennzahl bei grossen Datenmengen schnell und nicht zu rechenintensiv berechnet werden können. Aufgrund dieser Einschränkung sowie der Tatsache, dass unterschiedliche menschliche Auswertungen nicht nur länger Dauern und teurer sind, sondern auch nicht konsequent gleich sind, ist der folgende Abschnitt der Erläuterung verschiedener Metriken gewidmet.

3.3.1. Genauigkeit

Im Kontext von NL-to-SPARQL kann man die Genauigkeit auf verschiedene Weisen definieren, wir verwenden hier drei verschiedene Arten, welche nachfolgend definiert werden.

Zum einen kann Genauigkeit als die Proportion der korrekt generierten SPARQL-Abfragen eines Systems aufgefasst werden. Wenn zum Beispiel ein System für die Hälfte aller Beispiele die genau gleiche SPARQL-Abfrage generiert, hat es eine Genauigkeit von 50 %. Diese Art der Genauigkeit nennen wir *Genauigkeit*.

Die Verwendung von der gewöhnlichen Genauigkeit als Leistungsindikator wird typischerweise für Klassifikationsmodelle verwendet, im Kontext der Umwandlung von natürlicher Sprache zu SPARQL, gibt es jedoch in der Regel mehrere Abfragen für einen Quellsatz, welche zum korrekten Resultat führen. Das heisst, die Berechnung der Genauigkeit sollte speziell an die Eigenschaften von SPARQL angepasst werden.

Es können zum Beispiel unterschiedliche Variablen in der Ground Truth und der generierten Abfrage verwendet werden, solange dies konsequent für eine Abfrage gemacht wird, führt die Abfrage trotzdem zum selben Resultat. In Beispiel 3.3 wird dies anhand zweier Abfragen verdeutlicht. Ausserdem hat die Reihenfolge der Triples nicht immer eine Auswirkung auf das Resultat wie in Beispiel 3.4 gezeigt.

Listing 3.3: Beispiel für eine Abfrage, wo die Variabel konsequent anders.

```
SELECT DISTINCT ?uri WHERE {
  <http://dbpedia.org/resource/Londo_Mollari> {http://dbpedia.org/ontology/series}> ?uri
}
<=>
SELECT DISTINCT ?x WHERE {
  <http://dbpedia.org/resource/Londo_Mollari> <http://dbpedia.org/ontology/series> ?x
}
```

Listing 3.4: Beispiel für eine Abfragen, wo die Reihenfolge der Triples keine Rolle spielt.

```
SELECT DISTINCT ?uri WHERE {
  <http://dbpedia.org/resource/The_Ultimate_Merger> <http://dbpedia.org/ontology/
      executiveProducer> ?uri.
  <http://dbpedia.org/resource/Trump_Productions> <http://dbpedia.org/ontology/
      keyPerson> ?uri
}
<=>
SELECT DISTINCT ?uri WHERE {
  <http://dbpedia.org/resource/Trump_Productions> <http://dbpedia.org/ontology/
      keyPerson> ?uri .
  <http://dbpedia.org/resource/The_Ultimate_Merger> <http://dbpedia.org/ontology/
      executiveProducer> ?uri
}
```

Deshalb berechnen wir zusätzlich zur *Genauigkeit* auch noch die *SPARQL-Genauigkeit*, welche speziell an SPARQL angepasst ist und die obengenannten Eigenschaften berücksichtigt.

Die letzte Berechnungsmöglichkeit, bezieht sich nicht auf die SPARQL-Abfrage selbst, sondern auf die Resultate, welche die Ausführung der Abfrage generiert, diese nennen wir hier *Ausführungsgenauigkeit*. Da das Resultat einer Abfrage aus mehreren Antworten bestehen kann, ist die Ausführungsgenauigkeit alleine eine Schlechte Metrik. Im folgenden werden wir anhand eines Beispiels zeigen, warum dies so ist und welche Metriken den Nachteil der Ausführungsgenauigkeit beseitigen.

Dafür benötigen wir die folgenden Begriffe:
True Positives (TP): Anzahl der Antworten des Systems, welche auch im Ground Truth Resultat sind.
False Positives (FP): Anzahl der Antworten des Systems, welche sich nicht im Ground Truth Resultat finden.
True Negatives (TN): Anzahl der negativen Beispiele, welche auch so klassifiziert wurden, diese Zahl ist hier nicht definiert, da es sich
bei NL-to-SPARQL nicht um eine Klassifizierungsproblem handelt. False Negatives

(FN): Anzahl der Antworten, welche im Ground Truth Resultat sind, aber nicht im Resultat des Systems.

Die Formel für die Ausführungsgenauigkeit ist in Gleichung 3.1 dargestellt.

$$Ausführungsgenauigkeit = (TP)/(TP + FP + FN) \tag{3.1}$$

Das Problem der Genauigkeit ist, dass sie grösser wird, solange es mehr TP als FP gibt, deshalb verwendet man *Präzision* und *Ausbeute*, welche nachfolgend erläutert werden.

3.3.2. Präzision

Präzision beantwortet die Frage, wie viele von den Resultaten durch die generierte Abfrage korrekt waren. Gleichung 3.2 zeigt die Formel zur Berechnung von Präzision. Meistens wird Präzision zusammen mit einem anderen Mass namens *Ausbeute* verwendet. Wenn alle Antworten der generierten Abfrage korrekt sind, auch wenn nicht alle gefunden wurden, ist die Präzision 1.0. Die Präzision kann erhöht werden, wenn nur die Antworten generiert werden, bei welchen sich das System sicher ist, dies verringert allerdings die Ausbeute.

$$Präzision = TP/(TP + FP) \tag{3.2}$$

3.3.3. Ausbeute

Die Ausbeute beantwortet die Frage, wie viele Antworten das System unter allen richtigen Antworten gefunden. Gleichung 3.3 zeigt die Formel zur Berechnung der Ausbeute. Einen Ausbeute von eins erhält man, wenn man alle Antworten generieren würde, auch Falsche Antworten, dies wiederum führt zu einer erheblichen Verringerung der Präzision. Deshalb muss eine Balance zwischen Ausbeute und Präzision gefunden werden, dabei entscheidet man sich für das System, bei welchem das für das Problem wichtigere Mass maximiert wird. Weiter gibt es das Mass *F-Score*, welches verwendet werden kann um ein Mass zu haben, welches Präzision und Ausbeute zu einem einzigen Wert kombiniert.

$$Ausbeute = TP/(TP + FN) \tag{3.3}$$

3.3.4. F-Score

F-Score (F1-score oder auch F-Measure) ist ein Mass für die Performanz eines Systems, definiert ist er als das gewichtete harmonische Mittel von Präzision und Ausbeute. Es wurde definiert um nur eine einzige Zahl für den Performanzvergleich zu brauchen. Der beste Wert liegt bei 1 und der schlechteste bei 0.

Die Formel vom F1-Score ist in Gleichung 3.4 gezeigt.

$$F_1 = 2 * \frac{Präzision * Ausbeute}{Präzision + Ausbeute} \tag{3.4}$$

Die generelle Formel für ein positives β ist in Gleichung 3.5 dargestellt, diese wird gebraucht wenn es mehr als zwei Klassen gibt.

$$F_\beta = (1 + \beta^2) * \frac{Präzision * Ausbeute}{(\beta^2 * Präzision) + Ausbeute} \tag{3.5}$$

In dieser Arbeit verwenden wir für die Evaluierung des Systems unteranderem den *Mikro* F-score und den *Makro* F-score, diese sind wie folgt definiert. Beim Mikro F-score wird zuerst die Summe aller True und False Positives und aller True- und False negatives berechnet und dann aufgrund dieser wird die Präzision und Ausbeute berechnet, um den F-score zu erhalten. Beim Makro F-score werden Präzision, Ausbeute und F-Measure pro Frage berechnet und dann der Durchschnitt verwendet. Wenn es Fragen mit keiner Antwort gibt, erhalten diese einen F-Score von 0.

3.3.5. BLEU

Der BLEU ist eine übliche Performanz-Metrik im Bereich von NMT, er wird verwendet, um zu messen, wie genau ein vorhergesagter Zielsatz im Vergleich zu seiner Referenz ist. Vereinfacht ausgedrückt bewertet der BLEU eine generierte Sequenz, indem diese mit einer vorgegebenen Referenz vergleicht wird und das Vorhandensein oder Fehlen von Wörtern in der Hypothese sowie die Reihenfolge und den Grad ihrer Trennung bewertet. Ein BLEU von Null kennzeichnet eine völlig ungenaue Übersetzung und ein Score von Eins eine fehlerfreie Übersetzung. Die Formel für BLEU wird in Gleichung 3.6 dargestellt, wobei P die Punktzahl, m_{max} die Anzahl der einzelnen Wörter aus der Hypothese in der Referenz und w_t die Gesamtzahl der Wörter in der Hypothese bezeichnet.

$$P = \frac{m_{max}}{w_t} \tag{3.6}$$

In der Praxis bedeutet dies, dass für die Hypothese “The bird is in the tree” das Wort “the” einen m_{max}-Wert von 2 hat, weil es zweimal in der Referenz “The bird is in the big tree” vorkommt und w_t einen Wert von 7 aufweist. In diesem Fall ist die Punktzahl 0.85, da in der Hypothese das Wort “big” fehlt. Dieser Prozess wertet jedoch noch nicht die Position einzelner Wörter in der Referenz aus. Dies geschieht durch die Einbeziehung einer *N-Gramm-Präzision*. Die *N-Gramm-Präzision* fasst Wörter zusammen, typischerweise in Gruppen von eins bis vier, und wiederholt den obigen Prozess. Für eine *Bigramm-Präzision* wird das erste und das zweite Wort (*"the bird"*) genommen und gemessen, wie oft diese Sequenz in der Referenz erscheint und dann werden schliesslich die einzelnen BLEU-Scores addiert. [7]

3.4. Neuronale Netzwerke

In diesem Abschnitt erklären wir Künstliche Neuronale Netzwerke (NN's), auch *Feed-Forward Neural Networks* gennant.

Neuronale Netzwerke
Neuronale Netzwerke sind ein mathematisches Modell, welches von der Struktur des menschlichen Gehirns (menschliches Neuronales Netzwerk) insperiert wurde [8]. NNs bestehen üblicherweise aus einer Eingangsschicht, mehreren versteckten Schichten und einer Ausgangsschicht [26]. Jede Schicht in einem *NN* kann mehrere *Neuronen* haben. Das *Neuron* berechnet aus der Eingabe und den entsprechenden Gewichten eine Ausgabe. Die Funktion, die für diese Berechnung innerhalb eines *Neuron* verantwortlich ist, wird als *Aktivierungsfunktion* bezeichnet. [7]

Abbildung 3.3 stellt eine Grundstruktur eines *NN* mit einer einzelnen versteckten Schicht dar. In der Abbildung werden die Eingänge mit x, die Gewichte mit w, die Funktion der Neuronen mit δ, die Aktivierungsfunktion mit a and der Ausgang des y gekennzeichnet. [7]

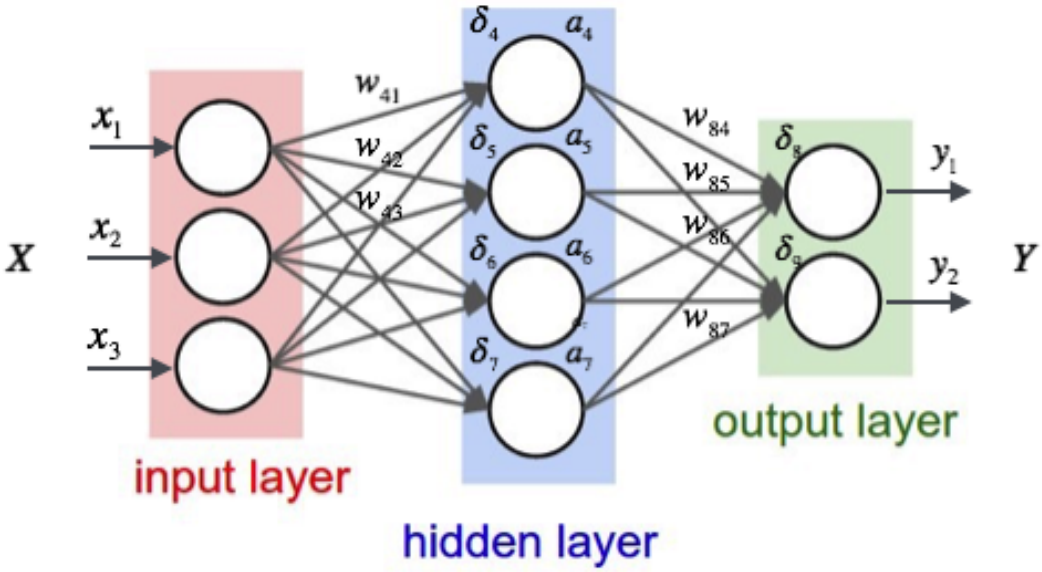


Abbildung 3.3.: Basic Feedforward Neural Network. [32]

Einfach gesagt werden *NNs* trainiert, indem sie Trainingsdaten erhalten, welche aus zwei Komponenten bestehen. Die eine Komponente stellt den Eingang dar, Daten mit deren Hilfe das *NN* die Vorhersagen generieren kann. Die andere Komponente wird für die Überprüfung der Genauigkeit der Vorhersagen und die Anpassung der Geweichte benutzt. Das Netzwerk berechnet eine Vorhersage basierend auf den

Funktionen der einzelnen Neuronen, den Eingangsgewichten der Neuronen und deren Aktivierungsfunktion. Zur Bewertung der Leistung wird in bestimmten Zeitabständen ein *loss* oder auch *Kosten* anhand einer *loss-* oder *Kosten-funktion* berechnet. Ein kleiner *loss* deutet darauf hin, dass die Vorhersagen des *NNs* genauer sind, also entsprechen sie den erwarteten Ergebnissen eher. Es existieren viele verschiedene *Lossfunktionen*, welche für verschiedene Problemstellungen mehr oder weniger geeignet sind. Im Grunde haben sie alle die Funktion, die Vorhersagen mit den korrekten Ergebnis zu Vergleichen und somit ein Mass anzugeben, wie gut die Vorhersage mit der korrekten Antwort übereinstimmt. Während des Trianingsprozesses werden die Gewichte fortlaufend optimiert, um das *loss* zu minimieren, damit die Genauigkeit der Vorhersagen steigt. [7]

Für diese Arbeit verwenden wir die Kostenfunktion *Cross-Entropy*, diese kann benutzt werden um zu messen, wie gut eine erwartete Wahrscheinlichkeitsverteilung (y in der Formel) durch ein trainiertes Modell vorhergesagt wird, wobei die vorhergesagte Wahrscheinlichkeitsverteilung als $\hat{y}$ bezeichnet wird. Gleichung 3.7 zeigt die Formel der *Cross-Entropy* Kostenfunktion. [7]

$$J^{(t)}(\Theta) = -\sum_{j=1}^{|V|} y_{t,j} * log\hat{y}_{t,j} \tag{3.7}$$

Während des Trainings besteht das Ziel darin, die Kosten möglichst zu verkleinern, was aber nicht direkt unser Hauptziel ist, da wir die Leistung des Netzwerks mittels der Genauigkeit bewerten. Die Genauigkeit gibt den Anteil der Vorhersagen an, welche genau mit der korrekten Antwort übereinstimmt. [7]

Backpropagation

Backpropagation ist eine Methode, um die *Kosten* eines Neuronalen Netzwerks zu verringern, indem die Gewichte des NN verändert werden. Dies wird mithilfe von Algorithmen umgesetzt, die als *Optimierer* bekannt sind. Backpropagation wird von diesen Optimierern verwendet. Unsere Arbeit nutzt einen der bekanntestent Optimierer, den sogenannten *Stochastic Gradient Descent* Optimierer. [7] Die Allgemeine Arbeitsweise von Backpropagation ist wie folgt:

1. Das *NN* wird mit Daten versorgt, berechnet Vorhersagen und die Kosten.
2. Die *Kosten* werden rückwärts durch das *NN* propagiert, wobei jedes Gewicht aufgrund seines Einflusses auf die Kosten aktualisiert wird.

Die Veränderung der Gewichte erfolgt durch die Subtraktion von dem Gewicht minus eines Wertes, welcher den Einfluss des Gewichts angibt. Der Einfluss des Gewichts

wird berechnet mithilfe der partiellen Ableitung der Kostenfunktion in Bezug zum jeweiligen Gewicht, anschliessend wird das Resultat mit der Lernrate multipliziert, die Lernrate gibt an wie stark sich ein Gewicht bei einer Aktualisierung verändert. Gleichung 3.8 zeigt die Aktualisierungsregel für ein Gewicht, wobei w_i das Gewicht ist, η die Lernrate und $\frac{\partial E}{\partial w_i}$ ist die partielle Ableitung der Kostenfunktion E in Bezug zum jeweiligen Gewicht w_i [21] [7].

$$w_i = w_i - \eta * \frac{\partial E}{\partial w_i} \tag{3.8}$$

3.5. Recurrent Neural Networks

Recurrent Neural Networks

Zusätzlich zum Informationsfluss eines NNs, besitzen *RNNs* eine zeitliche Dimension, das heisst sie sind in der Lage, die zeitliche Lage jeder Eingabe zu berücksichtigen. Dies wird erreicht durch hinzufügen einer Verbindung vom Ausgang des *Neurons* zum Eingang, wie in Abbildung 3.4 gezeigt wird. [27] [7]

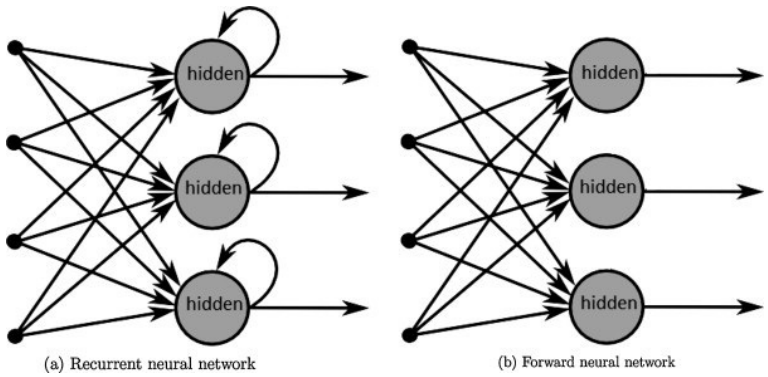


Abbildung 3.4.: Vergleich der Strukturen eines NN und eines RNN [19]

.

Abbildung 3.5 zeigt, wie in einem Basis-RNN mit drei Zellen die Eingänge verarbeitet werden. Zelle A erhält den ersten Input (bezeichnet als X_{t-1}) und erzeugt seinen Zustand (bezeichnet mit h_{t-1}). Bei der Verarbeitung des nächsten Einfangs X_t wird dieser mit dem Zustand h_{t-1} kombiniert, um den nächsten Schritt oder Zustand h_t zu berechnen. Bei einem RNN, wird der erste Eingang an die erste RNN-Zelle übergeben und der Ausgang, welcher von der Zelle produziert wurde, wird als Eingang für die nächste Zelle verwendet. Abbildung 3.5 stellt ein Basis-RNN mit drei Zellen dar. [7]

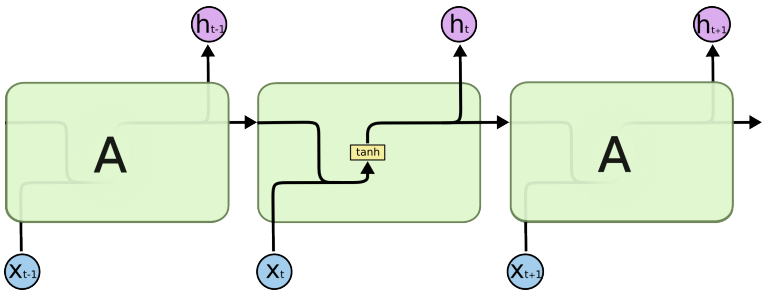


Abbildung 3.5.: Beispiel eines RNN mit drei Zellen [16]

.

Vanishing / Exploding Gradients:
Aufgrund der eingebauten Wiederholung in RNN's können Probleme wie Vanishing oder Exploding Gradients auftreten [2]. [7]

Exploding Gradients:
Das erste Problem, welches durch die rekursive Struktur des RNN's verursacht wird, sind *exploding gradients*. Gradienten, die sehr gross werden, werden als *exploding gradients* bezeichnet. [17]. Diese können bei der Ausführung von backpropagation entstehen, wo der Gradient ein Produkt aus mehreren Jacobimatrizen besteht. Wenn nun die Norm dieser Matrizen gross ist, neigt der Gradient dazu unendlich zu werden, wodurch die Fähigkeit des Netzwerks etwas zu lernen sinkt. Eine Lösung dieses Problem vorgeschlagen von Y. Bengio, ist es, den Gradienten zu kürzen, wenn dessen Norm einen definierten Schwellwert überschreitet. [2]. Diese Technik wird als *Gradient Clipping* bezeichnet. [7]

Vanishing Gradients
Das Problem der *vanishing gradients* kann während des Trianings des Netzwerks auftreten, beim verringern der Kosten, in Bezug zu den Gewichten. Um den Gradienten zu berechnen wird die Ableitung der Kostenfunktion bezüglich der Gewichte berechnet. Während dieses Prozesses, fliesst der Gradient rückwärts durch das RNN und die jeweiligen Aktivierungsfunktionen. Wenn nun die Aktivierungsfunktion wie zum Beispiel *tanh* zu einem Ergebnis in der Domäne [-1, 1] führt, wird durch die mehrfache Anwendung der Ableitung mittels der *Kettenregel* das Resultat immer kleiner. Letztendlich ist das Resultat ein sehr kleiner Gradient und das Problem der *vanishing gradients* ist das Ergebnis [17]. Ein Gradient von oder sehr nahe an Null führt zu winzige Aktualisierungen der Gewichte und damit ist das Netzwerk nicht mehr in der Lage sich zu verbessern und es lernt nichts mehr. [7]

Dies kann jedoch durch den Einsatz von *Long Short-Term Memory* (LSTM) Zellen vermieden werden, diese sind ein fortgeschrittener Typ von RNN Zellen. [7]

LSTM

Das Aufkommen von *LSTMs* durch Schmidhuber und Hochreiter geht das Problem von vanishing gradients an und führt die Fähigkeit ein Langzeitinformationen zu speichern [6]. Die Hauptidee hinter *LSTMs* ist der Zellzustand, der die Informationen repräsentiert, welche in der LSTM Zelle in einem hochdimensionalen Vektor gespeichert sind. Die *LSTM* Zelle enthält drei Arten von Gattern, die auf dem Zellzustand operieren. [7]

Forget gate: Dieses Gate ist für die Entscheidung verantwortlich, welche Informationen zu behalten und welche zu vergessen sind. Dies geschieht durch die Verwendung einer *sigmoid* Schicht, die für jede Zahl im Zellzustand eine Ausgabe im Bereich von [0, 1] erzeugt. Eine Ausgabe von eins führt dazu, dass alle Informationen erhalten bleiben, während eine Ausgabe von null zu einer vollständigen Entfernung von Informationen führt. [7] Gleichung 3.9 zeigt die Formel des Forget Gates.

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{3.9}$$

Input gate: Das Input Gate entscheidet, welche neuen Informationen im Zellzustand gespeichert werden sollen. Um neue Informationen zu speichern, entscheidet zunächst eine *sigmoid-Schicht*, welche Werte aus dem Zellzustand, aktualisiert werden sollen, in Gleichung 3.10 als i_t bezeichnet. Dies geschieht auf gleiche Weise wie im forget gate. [7]

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{3.10}$$

Nach der Identifizierung der zu aktualisierenden Werte, berechnet eine *tanh-Schicht*, weleche Eingabewerte im Zellzustand gespeichert werden sollen. Dieser Wert wird als C_t in der Gleichung 3.11 bezeichnet. [7]

$$\tilde{C}_t = \tanh(W_c * [h_{t-1}, x_t] + b_C) \tag{3.11}$$

Anschliessend wird der Zellzustand aktualisiert indem der alte Zustand mit f_t multipliziert wird, um die Dinge zu vergessen, welche vom forget gate ausgewählt wurden, woraufhin wir die Summe von $i_t * C_t$ und dem modifizierten Zustand berechnen. Dieser Schritt wird in Gleichung 3.12 gezeigt. [7]

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.12}$$

Output gate: Das output gate berechnet zunächst, welche Werte aus dem Zellzustand in die Ausgabe einfliessen sollen, mithilfe einer *sigmoid-Schicht*. Dann bringt es die Werte aus dem Zellzustand in die Domäne von [-1, 1], durch eine *tanh-Schicht* und, basierend auf der Ausgabe von der vorherigen *sigmoid-Schicht*, wird der Ausgang generiert. [7]

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.13}$$

$$h_t = o_t * \tanh(C_t) \tag{3.14}$$

Abbildung 3.6 stellt eine Grundstruktur eines RNN dar, ähnlich wie in Abbildung 3.5, jedoch wird in diesem Szenario die Verwendung von *LSTMs* als Zellen durch die innere Struktur der Zelle gezeigt. [7]

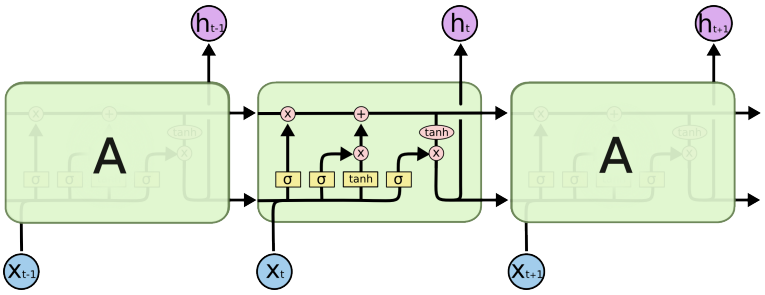


Abbildung 3.6.: Beispiel eines RNN mit drei LSTM Zellen [16]

.

3.6. Sequence-to-Sequence

Die *RNN* und *LSTM* Zellen sind Schlüsselkomponenten von *Sequence-to-Sequence* (Seq2Seq) Modellen. Obwohl sie in der Domäne von *Neural-Machine-Translation* ihre ersten grossen Erfolge hatten, können sie generell sehr gut lernen Eingangssequenzen zu Ausgangssequenzen abzubilden, wobei beide Sequenzen unterschiedliche Längen haben können. [7]

Diese Fähigkeite macht Seq2Seq-Modelle besonders vorteilhaft im Bereich der NL-to-SPARQL Domäne, da die Eingangssequenzen, nicht gleich viele Wörter beinhalten wie die Ausgangssequenzen. [7]

Eine Seq2Seq Architektur besteht aus zwei rekurrenten Modellen, einem *encoder* und einem *decoder*. [7]

Die beiden Modelle werden gemeinsam trainiert, so dass das ganze Modell die bedingte Wahrscheinlichkeit der Zielsequenz für einen bestimmten Eingangssatz gemeinsam maximieren kann. Zusammenfassend ist die Aufgabe des Encoders, den Eingangssatz auf einen Vektor fester Länge abzubilden, der allgemein als *hidden state* bezeichnet wird. Er erhält die Eingabe sequentiell in Form von *Token*, die je nach spezifischer Seq2Seq Implementierung entweder einzelne Wörter oder Zeichen sein können. Wenn das erste Token eines neuen Eingangsatzes an den Encoder übergeben wird, wird ein definiertes *Sequenzanfangssymbol* (SOS) an den Anfang der Sequenz gesetzt. Mit jedem neuen Token, welches dem Encoder zugeführt wird, wird sein *hidden state* aktualisiert, um die neuen Informationen aufzunehmen. Nach dem Empfang des letzten Tokens wird ein *Sequenzendsymbol* (EOS) an das Ende der Sequenz gesetzt. Die Token SOS und EOS dienen dazu, dem Decoder mitzuteilen, wo die Sequenzen beginnen und enden. Sobald eine Sequen kodiert wurde, wird der gesamte *hidden state* des Encoders, auch bekannt als *thought vector*, an den Decoder übergeben, der ihn als seinen Anfangs *hidden state* verwendet und mit der Dekodierung beginnt. Dies geschieht ebenfalls sequentiell pro Token, dabei wird die Ausgabe noch an eine *softmax Schicht* gegeben. Diese Schicht erzeugt eine Wahrscheinlichkeitsverteilung, die angibt, mit welcher Wahrscheinlichkeit welches nächste Token gewählt wird. Anschliessend erhält man so die Token, welche lokal die höchste Wahrscheinlichkeit besitzen. Diese Technik wird auch als *Greedy decoding*, zu deutsch *gieriges dekodieren* bezeichnet, da die Auswahl des Tokens auf der lokalen Wahrscheinlichkeit basiert. Ein Verfahren, welches die Gesamtwahrscheinlichkeit maximiert ist *Beam-search*, dies wird in Abschnitt 3.8 erläutert. [7] Abbildung 3.7 zeigt eine allgemeine Encoder-Decoder Architektur ohne SOS-Token. [7]

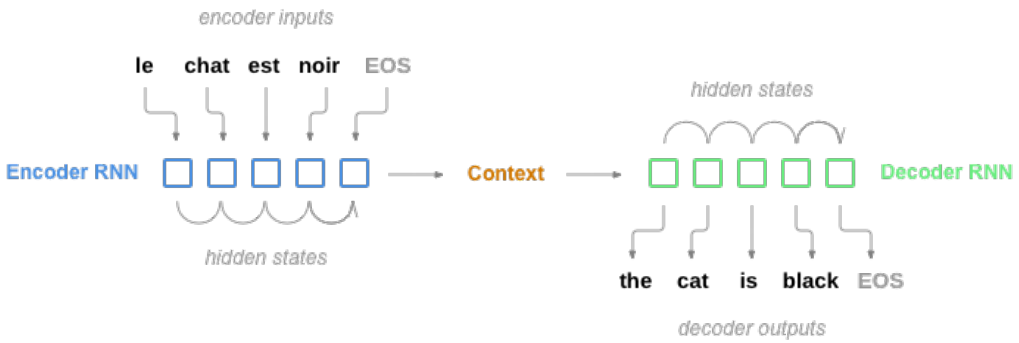


Abbildung 3.7.: Generelle Struktur einer Encoder-Decoder Architektur [20]

.

3.7. Attention

Der attention Mechanismus wurde von Dzmitry Bahdanau, et al. [1] eingeführt. Die Idee ist es, eine direkte Abkürzung zwischen Ziel- und Quellsätzen bereitzustellen, indem jedes Wort im Quellsatz nach seiner Bedeutung für die Vorhersage des aktuellen Tokens bewertet wird. Die Intuition dahinter ist es, die menschliche Übersetzung nachzuahmen. Während der menschlichen Übersetzung wird der Quellsatz kontinuierlich ausgewertet, um zu beurteilen, ob die aktuelle Übersetzung die Quelle exakt abbildet. Der Attentionmechanismus versucht, das Problem zu lösen, einen thought vector fester Länge als einzige Darstellung der Eingabefolge zu haben. Ohne diesen Mechanismus, sind lange Übersetzungen sehr schwer, weil die gesamte Sequenz in einem einzigen Vektor, der nach jedem *Kodierungsschritt* aktualisiert wird, darzustellen. Allerdings verbessert Attention nicht nur die Fähigkeit, Token aus einem Gedankenvektor zu dekodieren, die *Ausrichtung/alignment* wird auch verbessert. Die Ausrichtung bezieht sich hierbei auf den Prozess herauszufinden, welcher Teil einer Eingangssequenz für jedes Wort der Ausgangssequenz relevant ist. Es folgt eine kurze Erläuterung des Attentionmechanismus. [7]

Encoding: In einem *einfachen* Encoder-Decoder-Modell codiert der Encoder den Eingang in einen einzigen Vektor fester Länge. Mit dem Attention Modell, benötigt das Modell den Ausgang des Encoders nach jedem Eingangsschritt. In der Veröffentlichung sind dies Zustände *annotations* gennant. [1] [7]

Dekodierung: Bei jedem Zeitschritt der Dekodierung werden mehrere Schritte ausgeführt, um die den Attentionmechnismus anzuwenden. Diese Schritte werden im Folgenden näher erläutert. [7]

1. **Ausrichtung:** In diesem Schritt erhält jede *Annotation* eine Punktzahl, die den Grad ausdrückt, in dem die aktuelle *Annotation* auf die aktuelle Ausgabe abgebildet wird. [7]

2. **Normalisieren von Alignment Scores:** Um eine Wahrscheinlichkeitsverteilung zu berechnen, die die Wahrscheinlichkeit jeder Annotation in Bezug auf ihren aktuellen Zeitschritt anzeigt, werden die Alignment scores durch die Verwendung einer Softmax-Funktion normalisiert. [7]

3. **Erzeugen eines neuen thought vectors:** Ein neuer thought vector wird durch Multiplizieren jeder Annotation mit dem entsprechenden Gewicht erzeugt. [7]

4. **Dekodierung der nächsten Ausgabe:** Anstatt den ursprünglichen thought vector wie bei einem normalen RNN zu verwenden, wird der aktualisierte thought vector verwendet. [7]

Es gibt verschiedene andere Attentionmechanismen wie Luong's *Scaled-Luong* [11], welches von der normalisierten Form des Bahdanau attention mechanismus inspiert wurde, und Bahdanau's *Normed Bahdanau* [1], welches wiederum von der Saliman's weight normalization [23] inspiriert wurde. [7]
Ein potenzielles Problem, dass mit dem Attentionmechanismus verbudnen ist, ist der Anstieg der Berechnungsdauer. Wenn beispielsweise die Quell- und Zielsequenz eine Länge von 50 Token überschreiten, müssen insgesamt $50 * 50 = 2500$ attention values berechnet werden. Abbildung 3.8 zeigt ein Beispiel einer *Aurichtungsmatrix* (englisch: *alignment matrix*), diese zeigt, welche Token der Eingangssequenz für die Token der Zielsequenz, einen hohen beziehungsweise geringen Attention Score erhalten. [7]

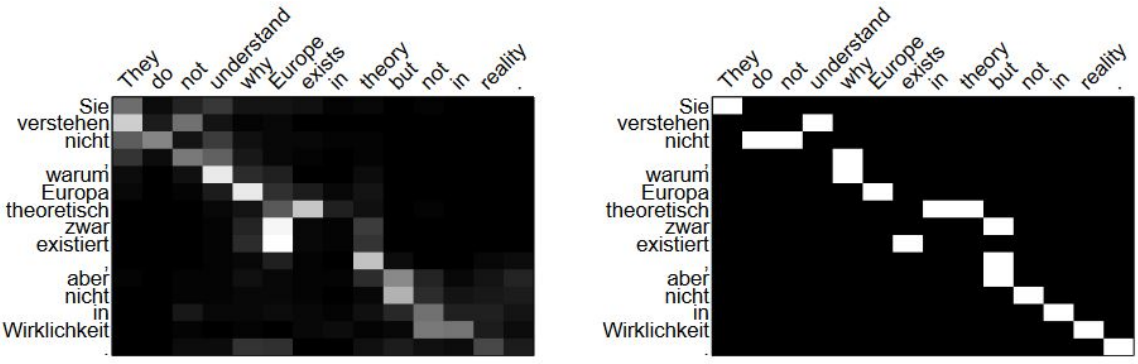


Abbildung 3.8.: Visualisierung der Ausrichtungsmatrix (Links: Ausrichtungsmatrix eines NMT-Systems; Rechts: Korrekte Ausrichtungsmatrix) [11]

3.8. Beam Search

Die Erklärung des Decoders verwendete das sogenannte *greedy* dekodieren, bei dem das nächste übersetzte Wort auf der Grundlage der höchsten lokalen Wahrscheinlichkeit ausgewählt wird. Aus diesem Grund werden Sätze wie *"Das Wettter ist heute wirklich sehr schön!* mit *"The weather today is nice!"* übersetzt, wobei die richtige Übersetzung näher am Satz *"The weather today is very nice!"* wäre. Abbildung 3.9 zeigt einen greedy Decoder, der das nächste Wort anhand seiner lokalen Wahrscheinlichkeit auswählt. Die lokalen Wahrscheinlichkeiten werden durch das Gesamte Seq2Seq-Modell berechnet, aber die Umwandlung in Wahrscheinlichkeiten findet mithilfe der letzten Softmax Schicht statt. [7]

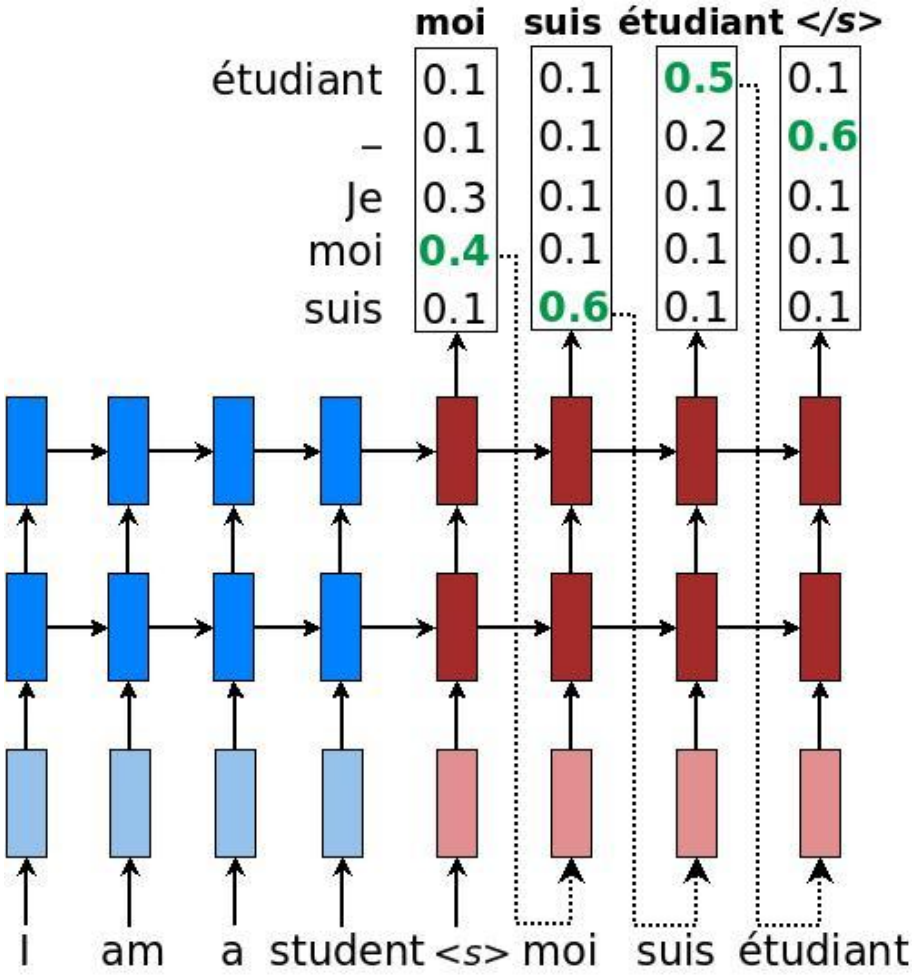


Abbildung 3.9.: Beispiel für einen greedy Decoder, bei dem bei jedem Zeitschritt das Wort mit der aktuell höchsten Wahrscheinlichkeit ausgewählt wird. [9]

Um dieses Problem zu minimieren, kann die Übersetzung basierend auf der gesamten Wahrscheinlichkeit optimiert werden. Dies geschieht durch die Anwendung einer

Technik, die als *beam search* bekannt ist. Beam search berücksichtigt bei jedem Zeitschritt die n besten Hypothesen, wobei n als *beam-width* bezeichnet wird. Eine grössere beam-width sollte die Qualität der Übersetzung verbessern, dabei wird aber die Zeit für das Dekodieren erhöht. [7]

3.9. Vokabular

Das Vokabular bezieht sich in dieser Arbeit auf eine bestimmte Sammlung von Wörtern, die vom Encoder und Decoder erkannt und somit korrekt codiert respektive decodiert werden können. Aufgrund der begrenzten Verfügbarkeit von Speicher- und Rechenressourcen muss das Vokabular jedoch auf eine feste Grösse (k) beschränkt werden. Das gesamte Vokabular einer Sprache im Speicher zu haben, ist kein praktikabler Ansatz. Die englische Sprache zum Beispiel enthält etwa 1 Millionen Wörter [15], was bereits eine sehr grosse Menge ist, die man im Speicher halten müsste. Einige Sprachen wie zum Beispiel Deutsch, erlauben es zudem einzelne Wörter zu neun Wörtern zusammenzufügen. Zum Beispiel kann man die deutschen Wörter *SSynapse"* und *SZerrung"* zusammenfassen zum Wort *SSynapsenzerrung"*. Um den Speicherverbrauch zu begrenzen, wird das Vokabular auf die n häufigsten Wörter in den Daten beschränkt. Dies würde jedoch den Wortschatz stark einschränken. Um das Problem zu verkleinern, gibt es eine Reihe von Techniken wie *Hybrid Word-Character Models* [11] oder *Subword Encoding* [25]. Dabei wird das Vokabular vergrössert, ohne dass mehr Speicherplatz benötigt wird. In unserer Arbeit haben wir uns gegen die Optimierung des Vokabulars entschieden, da die Vokabulare sowohl für die Englische Sprache wie auch für SPARQL basierend auf unserem Daten sehr klein sind. Ein festes Vokabular oder ein Vokabular mit BPE nicht geeignet, da SPARQL URI's verwendet und es schwer/nicht möglich ist für ein RNN die URI's mithilfe von BPE zu generieren. Alle URI's ins Vokabular aufzunehmen ist nicht möglich, da es alleine schon ca. 4 Millionen Ressourcen gibt und dies zum einen zu viel Speicher benötigen würde und zum anderen die Rechendauer sich erheblich erhöhen würde. [7]

3.10. Part-of-Speech Tagging

Unter Part-of-Speech Tagging (POS) versteht man das bestimmen der Wortarten von Wörtern in einem Satz (wird in englisch "Part-of-speech" gennant). Die Kenntniss der Wortarten ist sehr nützlich und gibt Auskunft über die wahrscheinlichen Nachbarwörter. Verwendet wird POS vorallem für Named-Entity Recognition, das finden von Entitäten in Texten, wie zum Beispiel Menschen oder Organisationen,

ausserdem wird es in anderen Datenextrahierungs Verfahren benutzt. [4] In Rahmen dieser Arbeit wurde der Part-of-Speech Tagger von der *Stanford Core NLP* Bibiliothek [13] für die Erkennung von Informationen in Texten verwendet, dabei ist zu beachten das nur ein sehr kleiner Teil der Arbeit dieses Verfahren verwendet, deshalb wird der Algorithmus hier nicht weiter erörtert. In dieser Arbeit wurde nur eine Teilmenge der POS-Tags verwendet, welche in der Tabelle 3.1 erläutert werden.

Tag	**Beschreibung**	**Beispiel**
NN	Substantiv (Singular) oder Masse	Präsident
NNS	Substantiv (Plural)	Präsidenten
DT	Bestimmungswort	Bei dem Wort \enquoteKaffeetasse ist das Bestimmungswort \enquoteKaffee

Tabelle 3.1.: Übersicht über die in dieser Arbeit verwendeten POS-Tags.

3.11. Stemming

Stemming oder auch Stammformreduktion bezeichnet die Technik Wörter zu ihrer Stammform umzuwandeln. Ein oft verwendeter Algorithmus für Stemming ist der sogenannte *PorterStemmer*, welcher von Porter [18] veröffentlicht wurde, dieser wird auch im Rahmen dieser Arbeit verwendet. Stemming wird vorallem in der Textanalyse verwendet, um zu vermeinden, dass ein Wort mit der selben Bedeutung als verschiedene Wörter erkannt wird, nur weil das Plural verwendet wurde oder es konjungiert wurde. [12] Beispiel 3.11.1 zeigt ein Wort in verschiedenen Wortformen, welche durch Stemming aber zur selben Stammform reduziert werden. In Beispiel 3.11.2 wird Stemming auf einen Satz angewendet

Beispiel 3.11.1. *car, cars, car's, cars*
=>
car citeManning2008

Folgendes Beispiel zeigt das Resultat von Stemming von einem ganzen Satz.

Beispiel 3.11.2. *the boy's cars are different colors*
=>
the boy car be differ color [12]

4. Modell

4.1. Beschreibung

Das Modell aus dem paper *SPARQL as a foreign language* stellt unsere Baseline dar. Dieses Modell wurde bereits im Abschnitt 2.1 im Rahmen der Verwandten Arbeiten erläutert. Als Seq2Seq-Modell wird die Implementierung von Google/Tensorflow verwendet [10]. Standardmässig werden alle Wörter als eigene Token codiert, auch die URI's von der SPARQL-Abfrage, dies führt dazu, dass neue Entitäten/URI's nicht erkannt werden. Um dieses Problem zu lösen, verwenden wir ein Pre- und Postprocessing. Unser Modell stellt somit eine Erweiterung der Baseline dar und wird in diesem Kapitel näher erläutert.

Tabelle 4.1 zeigt die Hyperparameter dieses Ansatzes. Anzumerken ist hier, dass die Werte bei Dropout, die *droprate* darstellen, also wie viel Prozent der Neuronen zufällig abgeschaltet werden.

Antworttyp	**Anzahl Vorkommen**
Zellentyp	LSTM
Anzahl Schichten	2
Zellentyp	Unidirektional
Optimizer	SGD
Lernrate	1.0
Anzahl Units	128
Dropout	0.2
Loss	Crossentropy
Src/Tgt seq. Länge	50
Gradient Max Clip Norm	5.0
Batch size	128
Anzahl Trainingsschritte	120'000

Tabelle 4.1.: Hyperparameter der Baseline.

Die vorliegenden Fragen reichen beim Weiten nicht aus um ein entsprechendes Seq2Seq Modell zu trainieren. Der Ansatz des zugrunde liegenden Generators verfolgt das Ziel aus den vorgegeben Fragen neue Fragestellungen zu generieren. Dazu

werden in einem ersten Schritt die korrespondierende SPARQL-Abfragen nach zu ersetzenden Entitäten durchsucht und durch eine Variable ersetzt

Die gesuchten Elemente aus allen Entitäten der Abfrage können leicht gefunden werden, da aus der Nomenklatur der Ressourcen URI Rückschlüsse auf dessen Zweck gezogen werden können was der Generator zu nutzen weist. Natürlich müssen die durch Variablen ersetzten Ressourcen noch in der Frage in natürlicher Sprache überführt werden. Dies geschieht in nächsten Schritt. Hier zeigt sich leider, dass nicht alle Fragen korrekt verarbeitet werden können, da in der Frage nicht immer die genau gleiche Beschreibung respektive Name wie in den DBPedia Einträgen verwendet werden. In unseren Testdaten hat es sich gezeigt, dass etwa 70% der Ressourcen erkannt werden.

Hyperparameter Zusätzlich zu den Standard Hyperparametern aus Tabelle 4.1 haben wir Experimente mit weiteren Hyperparametern durchgeführt. Dabei haben wir die Parameter und Werte aus Tabelle 4.2 verwendet. Eine Erläuterung der Gründe, wieso wir diese Parameter versucht haben findet sich im Kapitel 6 wieder, hier möchten wir nur eine Übersicht der Parameter geben.

Antworttyp	**Anzahl Vorkommen**
Dropout	0.2 und 0.5
Decay schema	keins, luong10 und eigenes ...
Optimierer	SGD und Adam
Anzahl Units	100 und 128
Beam Search	kein, 10

Tabelle 4.2.: Übersicht verschiedener getesteter Hyperparameter für das Baseline Modell.

4.2. Preprocessing

Das aufbereiten der Daten ist ein essenzieller Schritt um sinnvolle Resultate aus den Daten zu generieren. Das Ziel der Datenaufbereitung ist die Daten in ein passendes Format umzuwandeln und gegebene Unstimmigkeiten oder Probleme zu beheben. Im Kontext von NL-to-SPARQL ist es wichtig die Daten synchron zu halten, so dass die Paare von NL-Frage und SPARQL immer übereinstimmen. Ausserdem muss der Text, jedes Token in einen Vektoren umgewandelt werden, mit dem das NN umgehen kann. Dies wird mithilfe von Word Embeddings erreicht, Word Embeddings repräsentieren ein Wort in einem hochdimensionalen Vektor, wobei sie die Eigenschaft haben, das semantisch ähnliche Wörter in dem hochdimensionalen Raum nahe beieinander liegen.

4.3. Extrahierungsverfahren

Eine Erweiterung unseres Modells ist das extrahieren und ersetzen von DBPedia URI's. Die Extrahierung besteht dabei aus zwei Phasen, einer im Preprocessing, welche das extrahieren der URI's macht und einer zweiten, welche nach der Umwandlung zu SPARQL im Postprocessing, die speziellen Token wieder durch URI's ersetzt.

Entitäten Extrahierung Im Preprocessing ersetzen wir die Entitäten durch Variablen und das Modell soll besser generalisieren, ausserdem werden die Entitäten damit nicht mehr im Vokabular benötigt. Im Postprocessing ersetzen wir dann die Variable wieder mit der richtigen Entität. Dabei kommen verschiedene Verfahren zur Anwendung. Unter anderem werden die Daten durch den Online Service Spotlight[2] mit potenziellen Ressourcen angereichert. Wie genau dieser Mechanismus funktioniert wird in 6.2.1 beschrieben.

Ontologieklassen Extrahierung Hier werden im Preprocessing Wörter, welche eine Ontologieklasse darstellen, mit speziellen Token ersetzt. Somit können alle Ontologieklassen im Vorfeld dem Seq2Seq-Modell im Vokabular gegeben werden. Wie genau dieser Mechanismus funktioniert wird in 6.2.2 beschrieben.

Eigenschaften Extrahierung Bei der Extrahierung der Eigenschaften gehen wir gleich vor, wobei hier im Preprocessing Wörter, welche eine Eigenschaft darstellen, mit speziellen Token ersetzt. Somit kann dem Seq2Seq-Modell die DBPedia-Eigenschaften einer Frage übergeben werden. Dieser Mechanismus wird in 6.2.3 beschrieben.

[2] https://www.dbpedia-spotlight.org/

5. Daten

Dieses Kapitel erklärt und untersucht die Datensätze, welche in dieser Arbeit verwendet wurden. Dabei wird vor allem auf die Struktur der Daten eingegangen, mit dem Ziel das Verständnis der zugrundeliegenden Daten zu erlangen um so ein System zu bauen, welches die Konzepte und Struktur der Daten möglichst ausnutzt. Ausserdem werden die notwendigen Techniken des Preprocessings erklärt, um die Daten für das Training korrekt aufzubereiten.

In unserer Arbeit haben wir insgesamt vier verschiedene Datensätze verwendet. Der erste Datensatz ist von der SQA2018 Challenge, welche von der Project-Hobbit Organisation stammt. Der zweite Datensatz wurde von uns selbst erstellt auf Basis der MovieDatebank. Der dritte Datensatz basiert auf der QALD 7 Challenge und der vierte Datensatz wurde von den Autoren von [29] veröffentlicht.

5.1. Movie-Datensatz

Zu Beginn unserer Arbeit haben wir uns dafür entschieden einen kleinen Datensatz selbst zu erstellen. Dafür verwenden wir die Movie Datenbank, welche zu einem Triplestore umgebaut wurde im Rahmen einer Bachelorarbeit an der ZHAW [30] Die Movie Datenbank basiert ebenfalls zu grossen Teilen auf DBPedia, bietet aber den Vorteil einer besseren Übersichtlichkeit. Unsere Intention dahinter war uns zuerst auf ein kleines einfacheres Teilproblem zu konzentrieren, um die Konzepte von Verlinkten Daten besser zu verstehen. Ausserdem bietet diese Datenbank uns die Möglichkeit das SPARQL und auch das SQL Backend zu verwenden.

Für diesen Datensatz haben wir selbst insgesamt 20 Fragen und die jeweiligen SPARQL-Abfragen von Hand erstellt, sodass wir diese auf die Movie Datenbank anwenden können. Diese Daten werden hier nicht weiter beschrieben, da sie nur zu Lernzwecken verwendet wurden.

5.2. SQA2018

Der Scalable Question Answering 2018 (SQA2018) Datensatz wurde im Rahmen der SQA2018 Challenge [3] veröffentlicht von der "project-hobbit" Organisation. Er enthält Fragen in natürlicher Sprache, Keywords, SPARQL-Abfragen und die Resultate der jeweiligen Abfrage.

Die darin erhaltenen Abfragen sind einfach gehalten und basieren auf DBPedia (DBPedia Dump 2016-04). Ein Algorithums wurde verwendet um basierend auf wenigen Ursprungsfragen, neuer Fragen zu generieren indem Suchwunsch und die Form des natürlichen Sprachausdrucks verändert wurden. Um genau zu sein wurden die Enitäten von den QALD1 bis QALD5 zufällig mit Instanzen von der selben DBpedia Klasse ersetzt (In der Frage und in der Abfrage). Abfragen, welche kein ordentliches Ergebnis geliefert haben wurden entfernt. [4]

Um einen Eindruck der Daten zu erhalten werden in Listing 5.1 fünf zufällig gewählte Fragen, die entsprechnden SPARQL-Abfragen und das Resultat der jeweiligen Abfrage dargestellt.

Listing 5.1: Fünf Beispiel Fragen, die entsprechende SPARQL-Abfrage und die Resultate der Abfrage aus den SQA2018 Daten.

```
1  Frage: "which party has come in power in mumbai north?"
2  SPARQL: SELECT DISTINCT ?uri WHERE {
3          ?x <http://dbpedia.org/property/constituency> <http://dbpedia.org/resource/
               Mumbai_North_(Lok_Sabha_constituency)> .
4          ?x <http://dbpedia.org/ontology/party> ?uri .
5        }
6  Antworten: ["http://dbpedia.org/resource/Indian_National_Congress", "http://dbpedia.org/
       resource/Bharatiya_Janata_Party"]
7
8
9  Frage: "what is the largest city in the country where the san marcos river originates?"
10 SPARQL: SELECT DISTINCT ?uri WHERE {
11         <http://dbpedia.org/resource/San_Marcos_River> <http://dbpedia.org/ontology/
               sourceMountain> ?x .
12         ?x <http://dbpedia.org/ontology/largestCity> ?uri .
13         ?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/
               ontology/Country>
14   }
15 Antworten: ["http://dbpedia.org/resource/New_York_City"]
16
17
18 Frage: "count the wars in which people awarded with the croix de guerre fought."
```

[3] https://project-hobbit.eu/challenges/sqa-challenge-eswc-2018/
[4] Offiziele Beschreibung der Daten: https://hobbitdata.informatik.uni-leipzig.de/SQAOC/Documentation.txt

```
SPARQL: SELECT DISTINCT (COUNT(?uri) as ?callret) WHERE {
        ?x <http://dbpedia.org/ontology/award> <http://dbpedia.org/resource/
            Croix_de_guerre_1939\u20131945_(France)> .
        ?x <http://dbpedia.org/property/battles> ?uri .
        ?uri <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/
            ontology/MilitaryConflict>
    }
Antworten: ["335"]


Frage: "does the us route 281 have a junction at san antonio?"
SPARQL: ASK WHERE {
        <http://dbpedia.org/resource/U.S._Route_281> <http://dbpedia.org/property/
            junction> <http://dbpedia.org/resource/San_Antonio>
    }
Antworten: [true]


Frage: "in how many places have the companies started in newcastle worked?"
SPARQL: SELECT DISTINCT (COUNT(?uri) as ?callret) WHERE {
        ?x <http://dbpedia.org/ontology/foundationPlace> <http://dbpedia.org/resource/
            City_of_Newcastle> .
        ?x <http://dbpedia.org/ontology/regionServed> ?uri .
        ?uri <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/
            ontology/Place>
    }
Antworten: ["8"]
```

Tabelle 5.1 zeigt die verschiedenen Antworttypen aller SQA2018 Trainingsdaten.

Antworttyp	**Anzahl Vorkommen**
Keine Antwort	2
Ressource	3972
Wahrheitswert	368
Zahl	658

Tabelle 5.1.: Grobe Antworttypen und ihre Anzahl Vorkommen im SQA2018 Trainingsset.

Ein Indikator für die Komplexität der Fragen, stellt die Anzahl an Sprüngen in der SPARQL-Abfrage dar, konkret gibt es 4533 Fragen, bei denen kein Sprung nötig ist, 221 die einen Sprung enthalten und 246 welche zwei Sprünge enthalten.

Tabelle 5.2 zeigt die URI Präfixe und wie oft diese im SQA2018 Datensatz vorkommen. Weiter fällt auf das es in einer Abfrage maximal zwei Variabeln gibt und diese immer *?uri* und *?x* heissen. *?uri* kommt 11668 mal vor und *?x* insgesamt 4740 mal.

URI Prefix	Anzahl Gesamt	Anzahl Distinct
http://dbpedia.org/resource/	6621	3968
http://dbpedia.org/property/	3906	312
http://dbpedia.org/ontology/	6255	471
http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	1925	1

Tabelle 5.2.: URI Präfixe und ihre Anzahl im SQA2018 Datensatz)

Abbildung 5.1 zeigt das erste Wort der Natürlichsprachigen Fragen gelistet.

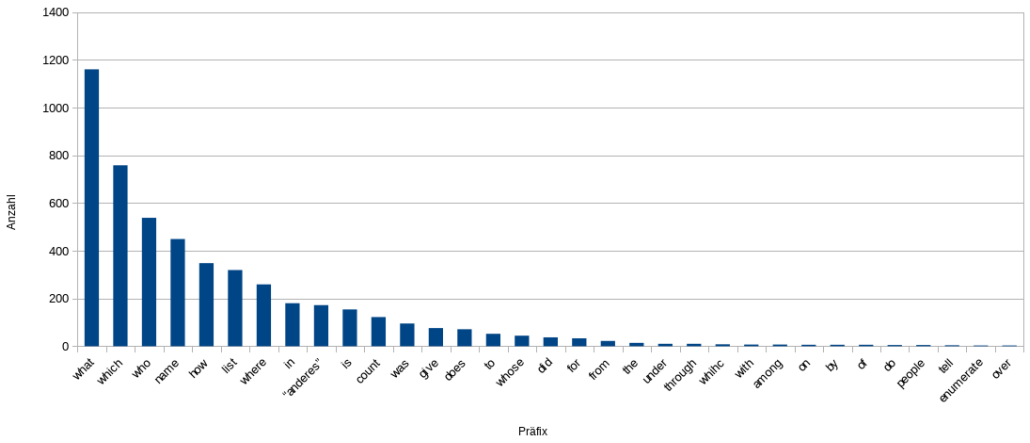


Abbildung 5.1.: Fragenpräfixverteilung im SQA2018 Datensatz.

Verschiedene Abfrage Templates:
Hier zeigen wir grobe Templates für die SPARQL-Abfragen, basierend auf den Daten. In Tabelle 5.3 zeigen wir die verschiedenen SPARQL-Templates, welche wir mittels Regulären Ausdrücken, aus den Daten generiert haben. Dabei wurde erkannt, dass die Abfragen keine *GROUP BY* und *Filter* Operationen verwendet werden, also keine Aggregation verwendet wird. Dies zeigt ebenfalls, dass es sich um tendenziell einfachere Abfragen handelt.

Abbildung 5.2 zeigt welche Arten von Named Entities in den Trainingsdaten vorhanden sind. Die Named Entities wurden mithilfe der *Stanford CoreNLP* Bibliothek [13] erkannt.

Prefix	Anzahl Vorkommen
ASK WHERE s p o	368
SELECT DISTINCT	4632
SELECT DISTINCT ?uri WHERE ...	3974
SELECT DISTINCT COUNT(?uri) WHERE ...	658
SELECT DISTINCT ?uri WHERE s p o	907
SELECT DISTINCT ?uri WHERE s p o . s p o	1715
SELECT DISTINCT ?uri WHERE s p o . s p o . s p o	1352
SELECT DISTINCT COUNT(?uri) WHERE s p o	93
SELECT DISTINCT COUNT(?uri) WHERE s p o . s p o	388
SELECT DISTINCT COUNT(?uri) WHERE s p o . s p o . s p o	177

Tabelle 5.3.: Templates und die Anzahl wie oft diese im SQA2018 Datensatz.

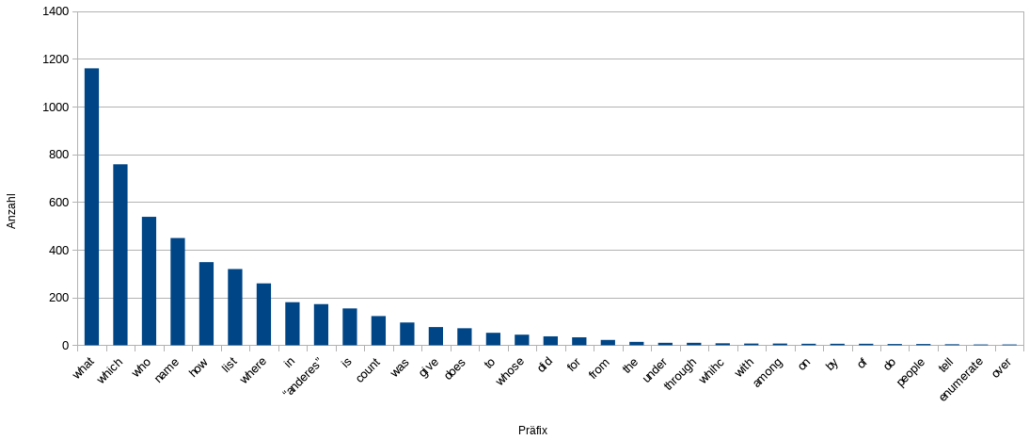


Abbildung 5.2.: Entitätstypenverteilung im SQA2018 Datensatz.

5.3. Monument

Der Monument Datensatz wurde von den Autoren des Baseline Modells verwendet und die Resultate der Autoren basieren ebenfalls auf diesem Datensatz, daher haben wir diesen für die Verifizierung der Baseline verwendet, um sicherzustellen das sich keine Fehler im Code befinden und die Implementierung richtig verwendet wird.

Der Datensatz basiert auch auf der DBPedia Ontologie, aber hierbei werden nur Fragen in Bezug zu Denkmälern verwendet. Dieser Datensatz kann einerseits selbst erstellt werden mithilfe der veröffentlichten Templates, die Templates wurden bereits im Abschnitt 2.1 beschrieben, andererseits veröffentlichten die Autoren auch noch den bereits aufbereiteten Datensatz [5].

[5] https://github.com/AKSW/NSpM

Der schon Aufbereitete Datensatz basiert auf insgesamt 39 Templates und umfasst insgesamt 8544 Daten, dabei wurden diese in 8344 Trainingsdaten, 100 Validierungsdaten und 100 Testdaten unterteilt.

Der von uns aufbereitete Datensatz basiert zwar auch auf 39 Templates, beim instanzieren dieser werden aber nur 29 Templates verwendet, da die anderen zu wenige Resultate lieferen. Insgesamt erhalten wir 8366 Daten, wobei wir 6693 als Trainingsdaten, 836 als Validierungsdaten und 837 als Testdaten verwenden.

6. Experimente und Resultate

In diesem Kapitel wird die Methodik der durchgeführten Experimente beschrieben, sowie die entsprechenden Ergebnisse präsentiert. Im laufe dieser Arbeit wurden acht Schritte vorgenommen um das bestehende System erst zu analysieren und basierend auf diesen Resultaten zu erweitern um so möglichst gut das Problem NL-to-SPARQL zu lösen.

	Beschreibung	**SPARQL-Genauigkeit**	**Genauigkeit**	**Ausführungs-genauigkeit**	**BLEU**	**Mikro F1**	**Makro F1**
1	Baseline auf SQA2018 Daten	0.40	0.20	4.80	41.91	0.043	0.058
2	Attention implementiert	3.00	2.60	7.60	52.96	0.544	0.121
3	Dropout von 20% auf 50% erhöht	5.40	4.40	9.60	55.80	0.548	0.131
4	NER mit Spotlight	16.72	16.39	16.39	65.94	0.909	0.258
5	Resourcen Methode	16.90	21.71	21.71	66.11	0.671	0.221
6	Ontology Extraction	5.30	4.21	15.40	60.00	0.402	0.168
7	Property Extraction	20.60	11.40	11.40	56.06	0.037	0.142
8	Alle Methoden kombiniert	27.60	26.80	30.69	73.81	0.660	0.287

Tabelle 6.1.: Experiment Übersicht

6.1. Aufstellen der Baseline

In diesem Abschnitt folgen die Experimente, die nötig sind um unser Baseline Modell zu verifizieren und anschliessend die Metriken auf dem richtigen Datensatz zu berechnen.

6.1.1. Verifizieren der Baseline

Die erste Art von Experiment zielt auf die Verifizierung der veröffentlichten Resultate und dem untersuchen dieser ab. In unserem ersten Experiment haben wir das Standardmodell von *SPARQL as a Foreign Language* [29] basierend auf deren Code [6] ausgeführt. Dabei haben wir die zur Verfügung gestellten Templates *Monument* verwendet, welche vom Herausgeber des Papers veröffentlicht wurden und auf der

[6] https://github.com/AKSW/NSpM

QALD 7 Challenge [31] basieren, respektive selbst vom Autor erstellt wurden. Details zu den hier verwendeten Datensätzen sind im Abschnitt 5.3 beschrieben.

Bei der Ausführung der bereitgestellten Implementierung auf dem schon aufbereitetem Monument Datensatz aus Abschnitt 5.3 erhalten wir sehr ähnliche Resultate wie vom Autor publiziert. Wenn wir nun aber die Daten selbst aufbereiten mit den zur Verfügung gestellten Skripts, erhalten wir einen schlechteren BLEU Score, dies liegt an einem Fehler in der Umwandlung der Wörter zu Token, hierbei werden zum einen mehrere Leerzeichen hintereinander eingefügt und zum anderen, wird ein zusätzlicher Punkt vor der schliessenden Klammer in der SPARQL-Abfrage eingefügt, beide Fehler haben zwar keine Auswirkung auf das Resultat der Abfrage, dennoch beeinträchtigen sie den BLEU score. Ein Beispiel dieser zwei Probleme ist in Listing 6.1 gezeigt. Nach dem beheben dieses Fehlers, erhalten wir auch sehr ähnliche Scores.

Listing 6.1: Beispiel einer SPARQL-Abfrage mit doppelten Leerzeichen und einem zusätzlichen separations Punkt.

```
1 SELECT DISTINCT var_uri WHERE brack_open  dbr_Gestapo dbo_parentOrganisation
      var_x sep_dot var_x dbo_leader var_uri sep_dot brack_close
```

Tabelle 6.2 zeigt die Scores für die Modelle mit den bereits aufbereiteten Daten, den selbst aufbereiteten Daten und die publizierten Scores aus dem Paper [29]. Zusätzlich zu der BLEU Metrik zeigen wir hier auch andere Metriken, welche wir ebenfalls in den weiteren Experimenten verwenden werden.

	Veröffentlichte Resultate	**Selbst Aufbereitet**	**Bereits Aufbereitet**
BLEU	0.20	33.46	**77.37**
Genauigkeit	-	3.000	**10.048**
SPARQL-Genauigkeit	-	3.000	**10.048**
Ausführungsgenauigkeit	-	4.000	**7.143**
Mikro Präzision	-	0.114	**0.243**
Mikro Ausbeute	-	0.067	**0.124**
Mikro F1-Score	-	0.084	**0.164**
Makro Präzision	-	0.040	**0.138**
Makro Ausbeute	-	0.040	**0.136**
Makro F1-Score	-	0.040	**0.136**

Tabelle 6.2.: Vergleich der Modelle basierend auf dem Monumentdatensatz.

Ein Problem bei diesem Modell ist, dass der verwendete Generator die Trainingsdaten künstlich vergrössert und somit gleiche oder sehr ähnliche Sätze mehrmals im Trainingssatz vorkommen, ausserdem ist es auch möglich, dass Sätze sowohl in den Validierungsdaten als auch in den Trainingsdaten vorkommen, da die Aufteilung der Daten dies nicht berücksichtigt. Weiter haben wir festgestellt, dass die Fragen in den Validierungsdaten auf 29 von insgesamt 39 Templates basieren, wobei alle 29 Templates auch im Trainingssatz enthalten sind, dies bedeutet alle Validierungsdaten sind auch in den Trainingsdaten vorhanden, natürlich unterscheiden sich die Entitäten der Trainings- und Validierungsdaten, trotzdem beinträchtigt dies die Performanz auf neuen, ungesehenen Daten.

Man kann nur teils erkennen, dass die Fragen korrekt übersetzen werden, welche relativ gesehen häufig in den Trainingsdaten vorhanden sind und diese auch häufig in den Validierungsdaten vorkommen. Wenn Fragen mit dem gleichen Template falsch übersetzt werden, liegt das daran, dass andere Ressourcen verwendet wurden. Tabelle 6.3 zeigt die Templates und wie oft diese wo vorkommen.

6.1.2. Baseline auf dem SQA2018 Datensatz

In diesem Experiment haben wir das genau gleiche Modell, jedoch ohne den Generator, aus dem vorherigen Experiment auf dem SQA2018 Datensatz 5.2 trainiert und evaluiert. Dieses Modell gilt als unsere Baseline, da es erstens auf dem gleichen Datensatz basiert auf den sich auch die restlichen Experimente dieser Arbeit beziehen, ausserdem wurden bei diesem System noch keine Änderungen von uns vorgenommen, bis auf das Beheben des Fehlers in der Datenaufbereitung. Tabelle 6.4 zeigt die Metriken auf dem SQA2018 Datensatz, wobei das Modell *Baseline (no fix)* den Fehler bei der Datenaufbereitung noch beinhaltet, im Modell *Baseline + sepdot* wurde das Problem mit dem zusätzlichen Separierungspunkt behoben und im Modell *Baseline + fixes* wurde zusätzlich das Problem mit mehrfachen Leerzeichen behoben. Es ist zu erkennen, dass das Entfernen des zusätzlichen Separierungspunkt und auch das entfernen der mehrfachen Leerzeichen die Performanz des Modelles nur geingfügig verändert.

6.1.3. Baseline mit Attention Mechanismus und erhöhter Droprate

Das Baseline Modell hat eine schlechte Performanz, deshalb versuchen wir hier unterschiedliche Techniken um dies zu beheben. Als erstes evaluieren wir die Verwendung von einem Attention Mechanismus, da dieser vermutlich den höchsten Performanz

Template	Korrekte SPARQL Abfrage	Korrektes Resultat	Gesamtanzahl	Anzahl in Trainingsdaten	Anzahl in Validierungsdaten	Anzahl in Testdaten
give me the location of <A>	10	11	300	252	26	22
where can one find <A>	5	5	300	251	28	21
how long is <A>			300	248	22	30
is <A> in <B>		7	300	247	32	21
latitude of <A>			300	246	24	30
is <A> older than <B>		17	300	245	27	28
what is <A>	15	15	300	245	30	25
building date of <A>	3	3	289	244	25	20
is <A> a <B>	1	5	300	241	27	32
what is <A> all about	1	1	300	241	28	31
which is taller between <A> and <B>			300	241	27	32
what do <A> and <B> have in common		1	300	240	33	27
what is <A> related to	4	4	300	240	31	29
where is <A>	6	6	300	240	33	27
location of <A>	4	4	300	238	32	30
what are the coordinates of <A>	2	2	300	238	26	37
<A>	2	3	300	237	33	30
longitude of <A>	6	6	300	235	32	33
what is <A> about	1	1	300	235	31	34
when was <A> completed	3	3	289	235	26	28
where is <A> located in	5	5	300	235	33	32
give me the <A> tallest <B>			300	232	33	35
how north is <A>	6	6	300	232	28	40
is <A> more recent than <B>	2	24	300	232	39	29
when was <A> built	3	4	289	225	35	29
was <A> finished by <B>			289	223	34	32
what's <A> native name			264	220	18	26
who designed <A>	3	3	217	161	33	23
which is longer <A> or <B>			129	95	10	24
how many <A> are there in <B>			0	0	0	0
how many <A> does <B> have			0	0	0	0
the longest <A>			0	0	0	0
what are the <A> northernmost <B>			0	0	0	0
what are the <A> southernmost <B>			0	0	0	0
what is the <A> with the most <B>			0	0	0	0
what's the most recent <A> of <B>			0	0	0	0
what's the oldest <A> of <B>			0	0	0	0
what's the tallest <A>			0	0	0	0
which <A> has the most <B>			0	0	0	0
Summe	**82**	**136**	**8366**	**6694**	**836**	**837**

Tabelle 6.3.: Templates des Monument Datensatzes und die Häufigkeit in den verschiedenen Teildatensätzen.

	Baseline no fix	**Baseline + sep-dot**	**Baseline + fixes**
BLEU	27.69	41.53	**41.91**
Genauigkeit	**0.200**	**0.200**	**0.200**
SPARQL-Genauigkeit	0.200	0.200	**0.400**
Ausführungsgenauigkeit	4.200	4.500	**4.800**
Mikro Präzision	0.065	0.061	**0.084**
Mikro Ausbeute	0.020	0.021	**0.029**
Mikro F1-Score	0.031	0.034	**0.043**
Makro Präzision	0.059	0.061	**0.062**
Makro Ausbeute	0.054	0.056	**0.057**
Makro F1-Score	0.054	0.057	**0.058**

Tabelle 6.4.: Metriken der Baseline, mit verschiedenen Fixes

Gewinn bringen wird, ausserdem wird er in den meisten Seq2Seq-Modellen verwendet wird. Des Weiteren leidet das Modell an Overfitting, da nur 4500 Trainingsdaten zur Verfügung stehen, um das Overfitting zu verringern erhöhen wir die Droprate von 20 % auf 50 %, dies sollte zu einer besseren Performanz führen, da das Modell resistenter gegen Overfitting wird und nun besser Generalisieren sollte.

Wie erwartet führt die Verwendung eines Attention Mechanismus zur grössten Steigerung der Performanz von insgesamt 2.4 % Accuracy und generell verbessern sich alle Scores, vorallem Präzision, Ausbeute und der damit verbundene F1-Score. Die Erhöhung der Droprate erhöht die Scores ebenfalls, zum Beispiel steigt die Accuracy nochmals um 1.8 %, aber hier erhöhen sich Präzision, Ausbeute und der F1-Score nur gering. In Tabelle 6.5 sind die vollständigen Scores der hier vorgestellten Modelle.

6.2. Extrahierung von URI's

Ein grosser Teil der SPARQL-Abfrage besteht aus URI's bestehend aus Ressourcen (Resource), Ontologien (Ontology) und deren Eigenschaften (Property). Entitäten Extrahierung nutzt diese Eigenschaft aus und versucht die Fragen für den Agenten zu generalisieren. Die nächsten Experimente befassen sich mit der Evaluierung der unterschiedlichen Extrahierungsmechanismen.

	Baseline	Attention	Attention + Dropout 50 %
BLEU	41.91	52.96	**55.80**
Genauigkeit	0.200	2.600	**4.400**
SPARQL-Genauigkeit	0.400	3.000	**5.400**
Ausführungsgenauigkeit	4.800	7.600	**9.600**
Mikro Präzision	0.084	0.680	**0.690**
Mikro Ausbeute	0.029	0.453	**0.455**
Mikro F1-Score	0.043	0.544	**0.548**
Makro Präzision	0.062	0.127	**0.132**
Makro Ausbeute	0.057	0.124	**0.130**
Makro F1-Score	0.058	0.121	**0.131**

Tabelle 6.5.: Evaluierung der verschiedenen Modellanpassungen.

6.2.1. Entitäten Extrahierung

In diesem Schritt versuchen wir dem Modell die Arbeit, die Ressourcen aus der natürlichen Sprache selber abzuleiten, abzunehmen in dem wir Teile durch generische Variablen ersetzten. Kann ein Teil in der Frage durch eine Ressource ersetzt werden, wird diese der Inferenz in die erzeugte Anfrage eingesetzt. Dadurch werden Fragen aus den Trainingsdaten generalisiert und zu Frage Vorlagen vereint. Dieses Verfahren ermöglicht es, dass ähnliche Fragen, die sich nur durch ein unterschiedliches Objekt unterscheiden zusammen geführt werden können. Fragen mit ähnlichem Aufbau werden so zu Frage-Templates zusammengeführt. Das neuronale Netz muss keine Transformation von Ressourcen vornehmen und diese auch nicht in sein Vokabular aufnehmen. Dies hat auch zur Folge dass Anfragen mit unbekannten (nicht in den Trainingsdaten vorkommenden) Ressourcen möglich ist. Folgendes Beispiel soll die Idee dieses Verfahrens näher bringen.

Beispiel 6.2.1. *Which software was developed by Mikrosoft? → Which <var0> was developed by <var1>*

6.2.1.1. DBPedia Spotlight

In der ursprünglichen Version unseres NER Verfahrens wurde DBPedia Lookup eingesetzt, welches mit Hilfe von Schlüsselwörtern nach entsprechenden Ressourcen im DBPedia Einträgen sucht. Jedoch stellte sich bald heraus, dass dieses einfache Suchen nach entsprechenden einzelnen Wörtern in den meisten Fällen nicht zum Erfolg

führt, da es viele Ressourcen gibt die nicht nur mit einem Wort umschrieben werden (z.B Information Technologie). Da die Komplexität dieses Problems den Rahmen unserer Arbeit sprengen würde, greifen wir auf eine bereits existierende und erprobte Lösung zurück.

Die genaue Funktionsweise von Spotlight wird in den Arbeit von Daiber et al. [3] und Mendes et al. [14] beschrieben. Sie basiert auf einem Modell für NER. DBpedia Spotlight verwendet einen **Confidence-Score** als Eingabe und wendet ihn intern auf unterschiedlichen Filter an und bestimmt damit die möglichen Kandidaten für die zu ersetzende Entität an. Eine hoher Score stellt sicher, dass nur Satzteile ersetzt werden welche auch mit einer hohen Wahrscheinlichkeit der entsprechenden Ressource zugewiesen werden können. Jedoch hat sich gezeigt, dass dabei viele Entitäten nicht erkannt werden. Ein niedriger Wert hingegen liefert viel mehr Kandidaten, führt aber dazu das die ursprüngliche Frage möglicherweise an Bedeutung verliert. Es war deshalb wichtig einen optimalen Wert für die das Verfahren zu eruieren. Die folgenden zwei Beispiele zeigen die beiden die Anwendung mit unterschiedlichen Werten für den Confidence-Score. Folgende Beispiele 6.2.3 und 6.2.2 zeigen die Auswirkungen unterschiedlicher Confidence-Score auf eine Frage.

Beispiel 6.2.2. *Was Winston Churchill the prime minister of Selwyn Lloyds?* → *Was <http://dbpedia.org/resource/Winston_Churchill> the prime minister of Selwyn Lloyds?*

Beispiel 6.2.3. *Was Winston Churchill the prime minister of Selwyn Lloyds?* → *<http://dbpedia.org/resource/Was-sceptre> <http://dbpedia.org/resource/Winston_Churchill> the <http://dbpedia.org/resource/Prime_Minister_of_the_United_Kingdom> of <http://dbpedia.org/resource/Selwyn_Lloyd>?*

Um den optimalen Wert für den Confidence-Score zu ermittlen haben wir alle Daten mit unterschiedlichen Werten verarbeitet und die dazugehörige Confusion-Matrix, Ausbeute, Präzision und F-Score berechnet. Die folgende zusammengestellte Tabellen 6.6, 6.7, 6.8 und 6.9 stellen die unterschiedlichen Confidence-Score's gegenüber.

Um die Resultat der unterschiedlichen Scores noch besser miteinander vergleichen zu können, steht die unten aufgeführte Tabelle 6.10 zur Verfügung.

Aus den Messungen geht hervor das ein Confidence-Score von 0.1 den besten F-Score erreicht. Trotzdem werden auch mit der niedrigsten Confidence-Score nur etwa die Hälfte der möglichen Entitäten richtig erkannt. Beim Vergleichen der Fragen mit der dazugehörigen SPARQL-Abfrage (Ground Truth) haben wir festgestellt, dass viele Orte, Personen oder Objekte in der Frage mit drei oder vier Worten beschrieben

		Predicated Value	
		NO	YES
Actual Values	NO	0	1252
	YES	2983	3638

Precision	0.744
Recall	0.549
F-Score	0.632

Tabelle 6.6.: Resultate für Confidence-Score 0.1

		Predicated Value	
		NO	YES
Actual Values	NO	0	899
	YES	3311	3310

Precision	0.786
Recall	0.500
F-Score	0.611

Tabelle 6.7.: Resultate für Confidence-Score 0.2

		Predicated Value	
		NO	YES
Actual Values	NO	0	899
	YES	3311	3310

Precision	0.786
Recall	0.500
F-Score	0.611

Tabelle 6.8.: Resultate für Confidence-Score 0.5

werden und Spotlight mit diesen nicht gut zurecht kommt. Trotz verschiedenster Bemühungen konnten wir keine Möglichkeit finden Spotlight für diese Art von NER Problemen zu optimieren. Um ein optimal verarbeitendes NER System zu simulieren, haben wir auf die Referenzfragen aus den Trainingsdaten zurückgegriffen und damit die restlichen Entitäten in DBPedia Ressourcen aufgelöst. Dieses Verfahren nehnen wir *Resource Backreference* welches im folgenden Abschnitt genauer beschrieben wird.

		Predicated Value	
		NO	YES
Actual Values	NO	0	681
	YES	3498	3123

Precision	0.821
Recall	0.472
F-Score	0.599

Tabelle 6.9.: Resultate für Confidence-Score 0.7

Confidence Score	Ausbeute	Präzision	F-Score
0.1	0.549	0.744	0.632
0.2	0.500	0.786	0.611
0.5	0.500	0.786	0.611
0.7	0.472	0.821	0.599

Tabelle 6.10.: Übersicht der Resultate der einzelnen Confidence-Matrix

6.2.1.2. Resource Backreference

Im Resource Backreference Verfahren möchten wir einen Zustand erzeugen wo möglichst alle Entitäten aus den Fragen durch die entsprechende DBPedia Ressource substituiert werden. Das Verfahren nutzt den Umstand, dass uns zur jeder Frage die entsprechende SPARQL-Abfrage (Ground Truth) zur Verfügung steht. Zuerst wird in jeder Frage alle Ressourcen aus der referenzierenden SPARQL-Abfrage ausgelesen. Anschliessend wird ein Schlüssel aus dem letzten Teil der jeweiligen Ressource generiert. Auf dem Schlüssel werden unterschiedliche Operationen angewendet, welche die Chance erhöhen, dass dieser in den Fragen vorkommt. Folgendes Beispiel 6.2.4 zeigt wie der Schlüssel erzeugt wird.

Beispiel 6.2.4. *<http://dbpedia.org/resource/Winston_Churchill> → Winston_Churchill → Winston Churchill*

Mit dem extrahieren Schlüsselwort und der ursprünglichen Ressource wird ein Wörterbuch aufgebaut das beim Durchsuchen der Frage in natürlicher Sprache zum Einsatz kommt. Mit Hilfe eines einfachen Regelwerks wird die Fragen nach den entsprechenden Schlüsselwort durchsucht. Kann ein Teil der Frage dem Schlüssel zugewiesen werden, wird dieser durch die dahinter liegende Ressource ersetzt. Das Beispiel 6.2.5 zeigt wie das Nachschlagen im erzeugten Wörterbuch funktioniert. Es entspricht formal der NER Methode von Spotlight.

Beispiel 6.2.5. *Who was Winston Churchill* → *Who was* <*http://dbpedia.org/resource/Winston_Churchill*>

Mit dem beschriebenem Backreference Resource Verfahren konnten bei über 90 % der Fragen die richtige Ressource gefunden werden. Dank dieses Verfahrens konnten wir die Auswirkung eines nahezu perfekt funktionierendem NER Prozesses auf das Modell beobachten. In der Übersichtstabelle 6.1 sieht man, dass durch diese Anpassung die Accuracy auf Rund 21 % verbessert werden konnte.

Tabelle 6.11 zeigt die Resultate mit Spotlight, wenn alle gefundenen Ressourcen ersetzt werden. Bei Tabelle 6.12 handelt es sich um die Scores, wobei nur die Entitäten ersetzt wurden, welche auch in der korrekten SPARQL-Abfrage vorhanden sind, dies wurde für die Spotlight und die Ressourcen Methode dargestellt.

Man sieht in den Tabellen, das die Spotlight Version wo nur die korrekten Entitäten ersetzt wurden eine.

	Attention + Dropout 50 %	**Enitäten Extrahierung**	**Enitäten Extrahierung + Dropout 50 %**
BLEU	55.80	57.71	**59.45**
Genauigkeit	4.400	7.000	**8.900**
SPARQL-Genauigkeit	5.400	7.200	**9.600**
Ausführungsgenauigkeit	9.600	11.420	**14.400**
Mikro Präzision	0.690	0.719	**0.755**
Mikro Ausbeute	0.455	0.482	**0.488**
Mikro F1-Score	0.548	0.577	**0.592**
Makro Präzision	0.132	0.124	**0.161**
Makro Ausbeute	0.130	0.122	**0.151**
Makro F1-Score	0.131	0.121	**0.158**

Tabelle 6.11.: Metriken der Spotlight Version, wobei alle Entitäten ersetzt wurden, egal ob diese korrekt sind oder nicht.

	Attention + Dropout 50 %	Enitäten Extrahierung	Enitäten Extrahierung + Dropout 50 %	Ressource	Ressource + Dropout 50 %
BLEU	55.80	61.84	65.94	**66.11**	65.83
Genauigkeit	4.400	11.710	16.390	**21.710**	16.900
SPARQL-Genauigkeit	5.400	11.710	16.720	16.900	**17.600**
Ausführungsgenauigkeit	9.600	15.600	16.390	21.710	**25.290**
Mikro Präzision	0.690	0.979	**0.987**	0.959	0.969
Mikro Ausbeute	0.455	0.824	**0.843**	0.516	0.501
Mikro F1-Score	0.548	0.894	**0.909**	0.671	0.661
Makro Präzision	0.132	0.234	**0.258**	0.223	**0.258**
Makro Ausbeute	0.130	0.233	**0.261**	0.221	0.260
Makro F1-Score	0.131	0.233	**0.258**	0.221	**0.258**

Tabelle 6.12.: Metriken der Spotlight und Ressourcen Version, wobei nur Entitäten ersetzt wurden, welche korrekt sind.

6.2.2. Ontologieklassen Extrahierung

Als nächsten Schritt haben wir einen Mechanismus implementiert, welcher die Ontologieklassen aus der NL Frage extrahiert.

Dafür haben wir zuerst ein einfaches extrahieren mittels einem Invertierten Index für alle Ontologieklassen aus unserem DBPedia Dump erstellt. Dieser Mechanismus geht jedes Wort in der NL Frage durch, transformiert es in Kleinschreibung und testet, ob eine passende URI im Invertierten Index existiert. Wenn eine URI gefunden wurde, wird das passende Wort in der NL Frage mit dem Token der gefundenen URI ersetzt, Beispiel 6.2.6 zeigt ein Beispiel für ein Wort und das passende URI Token.

Beispiel 6.2.6. *president -> dbo_President dbo steht für* `http://dbpedia.org/ontology/`

Für die Messung der Qualität dieser Technik, berechnen wir den F1-Score. Die Berechnung dieser Metrik benötigt die Kennzahlen *TruePositives FalsePositives* und *FalseNegatives*, wie in Abschnitt 3.3.4 beschrieben. Diese Kennzahlen erhalten wir aus den echten URI's der Ontologieklassen in der SPARQL-Abfrage und der Ausführung des Mechanismus. Abbildung 6.13 zeigt die Confusion Matrix von diesem Mechanismus. Die Confusion Matrix zeigt, dass der Mechanismus mehr Wörter als Ontologieklassen klassifiziert und viele Ontologieklassen gar nicht erkannt werden können. Anzumerken ist, das wir die Metriken hierbei für alle Daten zusammen berechenen und nicht separat für die Trainings- und Validierungsdaten. Unsere Analyse ergab, das oftmals eine Ontologie nicht direkt erkannt wurde, da das entsprechende Wort in der NL Frage im Plural geschrieben war, wie in Beispiel 6.2.7.

Beispiel 6.2.7. *Bei der folgenden Frage sollte für das Wort band's die Ontologieklasse dbo_Band erkannt werden.*
Frage: which band's former member are kevin jonas and joe jonas?

		Predicated Value	
		NO	YES
Actual Values	NO	0	2599
	YES	1409	509

Tabelle 6.13.: Extrahierung der Ontologieklassen nur mithilfe von Monogrammen.

Um das Problem der verschiedenen Wortformen zu beheben, haben wir den Invertierten Index mittels Stemming aufgebaut, respektive setzen wir auch beim Lookup Stemming ein. Wie in Tabelle 6.29 gezeigt, hilft Stemming und erhöht den F1-Score um ca. 11 % auf 32.47 % Die Confusion-Matrix für den Ansatz mit Stemming ist in Tabelle 6.14 gezeigt.

		Predicated Value	
		NO	YES
Actual Values	NO	0	4336
	YES	744	1174

Tabelle 6.14.: Extrahierung der Ontologieklassen mittels Monogrammen und Stemming.

Ein Problem welches durch das Stemming entsteht ist, dass es nun Klassen gibt, welche das selbe Schlüsselwort verwenden, Beispiel 6.2.8 zeigt dies.

Beispiel 6.2.8. `http://dbpedia.org/ontology/Organ` → *organ*
`http://dbpedia.org/ontology/Organisation` → *organ*

Das Problem lässt sich wiederum lösen, indem wir die Schlüsselwörter, die gleich sind separat speichern und bei einem Matching mit Stemming, die URI wählen, welche näher an dem gematchten ungestemmten Wort liegt, anhand der Levenshtein Distanz.

Auch durch Stemming erhalten wir einen geringen F1-Score, dies liegt daran, dass viele Ontologieklassen aus mehreren Wörtern bestehen wie in Beispiel 6.2.9 gezeigt.

Beispiel 6.2.9. *Ontologieklasse: dbo_AmericanFootballTeam*
which american football team has stadium as o.co coliseum?
→

which `http://dbpedia.org/ontology/AmericanFootballTeam` *has stadium as o.co coliseum?*

Ein Möglichkeit dieses Problem zu lösen ist, die Verwendung von *N-Gramme* für den Aufbau des Indexes. Auch hier haben wir eine Version mit Stemming und eine ohne Stemming evaluiert. Wir haben 1-Gramme (Monogramme), 2-Gramme (Bigramme), 3-Gramme (Trigramme) und 4-Gramme (Tetragramme) verwendet, wobei wir alle kleineren N-Gramme für jeden Ansatz verwenden, dies bedeutet bei der Verwendung von Bigramme, versuchen wir die Wörter der Frage erst mit dem Bigramm zu matchen und anschliessend mit dem Monogramm. Die Confusion-Matrix für jedes dieser Modelle sind in den Tabellen 6.15, 6.16, 6.17, 6.18, 6.19 und 6.20 gezeigt.

Das Hauptproblem dieser Ansätze ist eine hohe Rate von False Positives, dies bedeutet es werden zu viele Wörter zu einer Ontologieklasse gemappt. Es werden also Wörter gemappt welche nicht in der Abfrage vorkommen. Ein Exemplar dafür ist in Beispiel 6.2.10 gezeigt.

Beispiel 6.2.10. *Ausgangsfrage: who is the office holder for constituencies castro district and haight-ashbury ?*
Erkannt: who is the `http://dbpedia.org/ontology/OfficeHolder` *for constituencies castro* `http://dbpedia.org/ontology/District` *and haight-ashbury ?*
Korrekt: who is the `http://dbpedia.org/ontology/OfficeHolder` *for constituencies castro district and haight-ashbury ?*

		Predicated Value	
		NO	YES
Actual Values	NO	0	2702
	YES	1236	682

Tabelle 6.15.: Extrahierung der Ontologieklassen mittels Bigrammen und ohne Stemming

		Predicated Value	
		NO	YES
Actual Values	NO	0	4440
	YES	417	1501

Tabelle 6.16.: Extrahierung der Ontologieklassen mittels Bigrammen und der Verwendung von Stemming.

Um die hohe Anzahl an False Positives zu verringern haben wir versucht Muster in den Fragen mit dem Einsatz von POS-Tags zu finden. Dafür haben wir das StanfordCoreNLP Packet [13] verwendet. Zuerst haben wir für jedes Wort in den Fragen

		Predicated Value	
		NO	YES
Actual Values	NO	0	2703
	YES	1220	698

Tabelle 6.17.: Extrahierung der Ontologieklassen mittels Trigrammen und ohne Stemming.

		Predicated Value	
		NO	YES
Actual Values	NO	0	4441
	YES	385	1533

Tabelle 6.18.: Extrahierung der Ontologieklassen mittels Trigrammen und der Verwendung von Stemming.

		Predicated Value	
		NO	YES
Actual Values	NO	0	2703
	YES	1220	698

Tabelle 6.19.: Extrahierung der Ontologieklassen mittels Tetragrammen und ohne Stemming.

		Predicated Value	
		NO	YES
Actual Values	NO	0	4441
	YES	385	1533

Tabelle 6.20.: Extrahierung der Ontologieklassen mittels Tetragrammen und der Verwendung von Stemming.

die POS-Tags bestimmt, anschliessend haben wir geschaut welche Reihenfolge von POS-Tags als Ontologieklassen erkannt wurden, obwohl es keine waren. Unsere Hypothese ist, dass die Präzision zwar erhöht wird, im gegensatz aber der Ausbeute sinken wird, da nun weniger Ontologieklassen klassifiziert werden.

Wir konnten bei der Durchführung drei Muster erkennen, welche darauf hin deuten, dass es sich bei dem aktuellen Wort um keine Ontologieklasse handelt, auch wenn dies nicht immer gilt, bei dem Modell mit Trigramme ohne Stemming konnten 1032 False Positives vermieden werden, dafür wurden auch 60 True Positives nicht gefun-

den. Bei dem Modell mit Trigramme und Stemming konnten 1751 False Positives vermieden werden, dabei wurden 98 True Positives weniger erkannt.

Das erste Muster ist in Beispiel 6.2.11 gezeigt, das zweite und dritte Muster in den Beispielen 6.2.12 respektive 6.2.13. Eine Erklärung der verwendeten POS-Tags wurde bereits in Abschnitt 3.10 gegeben.

Beispiel 6.2.11. *Muster: NN/NNS + DT + NNS/NN*
Frage: "name the municipality of roberto clemente bridge ?"
Laut Muster ist "municipality" keine Ontologieklasse, laut SPARQL ist es aber eine.

Beispiel 6.2.12. *Muster: NN/NNS NN/NNS*
Frage: "list some musicians associated with famous guitar players?"
Laut Muster ist "guitar" keine Ontologieklasse und dies ist auch so laut der SPARQL-Abfrage.
Frage 2: "what university campuses are situated in indiana?"
Laut Muster ist "university" keine Ontologieklasse, aber laut der SPARQL-Abfrage schon.

Beispiel 6.2.13. *Muster: DT NN/NNS DT*
Frage: "list the producer of the movies in which rosemarie dewitt in the star cast?"
Laut Muster ist "producer" keine Ontologieklasse und dies ist auch so laut der SPARQL-Abfrage.

Diese Technik haben wir ebenfalls mit dem Einsatz der N-Gramme evaluiert, die Confusion-Matrizen sind in den Tabellen 6.21, 6.22, 6.23, 6.24, 6.25, 6.26, 6.27 und 6.28 gezeigt.

		Predicated Value	
		NO	YES
Actual Values	NO	0	1477
	YES	1486	432

Tabelle 6.21.: Extrahierung der Ontologieklassen mittels Monogrammen mit POS Mustern aber ohne Stemming

In Tabelle 6.29 sind die Präzision, Ausbeute und F1-Score für alle Modelle gezeigt.

Es ist zu erkennen, das die Modelle mit den POS-Tags eine höhere Präzision haben und dafür einen geringeren Ausbeute, da nun viele False Positives vermieden werden, dabei aber auch einige True Positives nicht erkannt werden. Bei dem Modell mit Tetragramme ist die Präzision und auch der F1-Score niedriger, wenn die POS Muster verwendet werden.

		Predicated Value	
		NO	YES
Actual Values	NO	0	2436
	YES	914	1004

Tabelle 6.22.: Extrahierung der Ontologieklassen mittels Monogrammen mit POS Mustern und Stemming

		Predicated Value	
		NO	YES
Actual Values	NO	0	1670
	YES	1296	622

Tabelle 6.23.: Extrahierung der Ontologieklassen mittels Bigrammen mit POS Mustern und ohne Stemming

		Predicated Value	
		NO	YES
Actual Values	NO	0	2688
	YES	515	1403

Tabelle 6.24.: Extrahierung der Ontologieklassen mittels Bigrammen mit POS Mustern und Stemming

		Predicated Value	
		NO	YES
Actual Values	NO	0	1671
	YES	1280	638

Tabelle 6.25.: Extrahierung der Ontologieklassen mittels Trigrammen mit POS Mustern aber ohne Stemming

		Predicated Value	
		NO	YES
Actual Values	NO	0	2690
	YES	483	1435

Tabelle 6.26.: Extrahierung der Ontologieklassen mittels Trigrammen mit POS Mustern und Stemming

Im Gegensatz zum letzen besten Modell ohne Extrahierung der Entität, dem Modell mit Attention Mechanismus und Dropout von 50 %, hat sich die Performanz des

		Predicated Value	
		NO	YES
Actual Values	NO	0	2599
	YES	1409	509

Tabelle 6.27.: Extrahierung der Ontologieklassen mittels Tetragrammen mit POS Mustern aber ohne Stemming

		Predicated Value	
		NO	YES
Actual Values	NO	0	4336
	YES	744	1174

Tabelle 6.28.: Extrahierung der Ontologieklassen mittels Tetragrammen mit POS Mustern und Stemming

Modell	**Präzision**	**Ausbeute**	**F1-Score**
Monogramm kein POS kein Stemming	16.74	28.26	21.01
Monogramm kein POS Stemming	21.81	63.55	32.47
Monogramm POS kein Stemming	22.95	23.97	23.43
Monogramm POS Stemming	30.16	55.19	39.00
Bigramm kein POS kein Stemming	20.04	36.48	25.86
Bigramm kein POS Stemming	25.56	79.90	38.73
Bigramm POS kein Stemming	26.99	33.36	29.83
Bigramm POS Stemming	34.60	74.77	47.30
Trigramm kein POS kein Stemming	20.44	37.40	26.42
Trigramm kein POS Stemming	25.87	**81.29**	39.25
Trigramm POS kein Stemming	27.52	34.29	30.52
Trigramm POS Stemming	**34.97**	76.16	**47.93**
Tetragramm kein POS kein Stemming	20.44	37.40	26.42
Tetragramm kein POS Stemming	25.87	**81.29**	39.25
Tetragramm POS kein Stemming	16.74	28.26	21.01
Tetragramm POS Stemming	21.81	63.55	32.47

Tabelle 6.29.: Übersicht der Resultate für die Extrahierung der Ontologieklassen.

Modells bezüglich der Genauigkeit und der SPARQL-Genauigkeit verschlechtert, dafür sind die Ausführungsgenauigkeit und die F-Score bezogenen Metriken verbessert worden, dies liegt daran, das nun Fragen, welche mehr Antworten erhalten korrekt übersetzt worden sind.

	Attention + Dropout 50 %	**Ontologieklassen Extrahierung**	**Ontologieklassen Extrahierung (korrekt)**
BLEU	55.80	54.23	**60.00**
Genauigkeit	4.400	3.600	**4.210**
SPARQL-Genauigkeit	5.400	4.900	**5.300**
Ausführungsgenauigkeit	9.600	11.340	**15.400**
Mikro Präzision	**0.690**	0.270	0.340
Mikro Ausbeute	0.455	0.432	**0.492**
Mikro F1-Score	**0.548**	0.332	0.402
Makro Präzision	0.132	0.163	**0.175**
Makro Ausbeute	0.130	0.169	**0.177**
Makro F1-Score	0.131	0.166	**0.168**

Tabelle 6.30.: Vergleich der Ontologieklassen Extrahierung.

6.2.3. Eigenschaften Extrahierung

Bis hier haben wir die Ressourcen und die Ontologieklassen aus den SPARQL-Abfragen extrahiert, es ist naheliegend nun zu versuchen auch die Eigenschaften zu extrahieren, der Abschnitt wird dieser Idee gewidmet.

Für die Extrahierung der Eigenschaften verwenden wir den selben Mechanismus wie bei der Extrahierung der Ontologieklassen. Diesmal evaluieren wir die Extrahierung der Eigenschaften mit Trigramme mit und ohne Stemming.

Tabelle 6.31 zeigt die Confusion-Matrix ohne Stemming und Tabelle 6.32 mit Stemming. Ausserdem sind die Scores in der Tabelle 6.33. Es ist zu sehen, dass hier das Modell mit Stemming eine geringere Präzision hat, da nun mehr Eigenschaften erkannt werden, ein weiteres Problem von Stemming ist, das nun viele Eigenschaften das selbe Schlüsselwort haben und dies zu grösseren Problemen führt, da es sehr viele unterschiedliche Eigenschaften gibt, welche den selben Wortstamm haben und das Mapping nun nicht eindeutig geschen kann.

Wie in Tabelle 6.31 und 6.32 zu erkennen ist, werden ca. 30'000 Wörter als Eigenschaften klassifiziert, obwohl es nur ca. 4'500 Eigenschaften sind. In Beispiel 6.2.14

		Predicated Value	
		NO	YES
Actual Values	NO	0	27440
	YES	2189	2141

Tabelle 6.31.: Extrahierung der Eigenschaften mittles Trigramme.

		Predicated Value	
		NO	YES
Actual Values	NO	0	29278
	YES	2886	1444

Tabelle 6.32.: Extrahierung der Eigenschaften mittles Trigramme und Stemming

Modell	**Präzision**	**Ausbeute**	**F1-Score**
Trigramme	**6.22**	**44.90**	**10.92**
Trigramme und Stemming	4.57	32.56	8.02

Tabelle 6.33.: Übersicht der Resultate der Eigenschaftextrahierung.

sieht man das fast alle Wörter als Eigenschaft erkannt werden. Da während der Inferenz die entsprechende SPARQL-Abfrage nicht zur Verfügung steht.

Beispiel 6.2.14. *who dbp_is dbp_the dbp_player dbp_who dbp_playedFor dbp_the dbp_team that dbp_ha dbp_the michigan stadium?*

Um zusehen ob das Modell trotzdem besser wird oder erst eine Verringerung der False Positives nötig ist haben wir ein Modell auf den Daten wo die Wörter als Eigenschaften codiert werden, wie im Preprocessing erkannt, trainiert und ein Modell auf den Daten wo nur die ca. 2000 korrekt erkannten Eigenschaften als solche codiert werden. Tabelle 6.34 zeigt die beiden Modelle im Vergleich, wobei das Modell "Eigenschaften Extrahierung (korrekt)", die Variante zeigt, welche nur die 2000 korrekt erkannten Eigenschaften als solche codiert.

Die Evaluierung ergab, dass sich die aktuelle Extrahierung der Eigenschaften nicht rentiert, da zuviele Wörter irrtümlich als Eigenschaften erkannt werden, die Performanz des Modells hat sich daher verschlechtert. Bei dem Modell "Eigenschaften Extrahierung (korrekt)" hat sich die Performanz aller Metriken, ausser den Mikro Metriken, verbessert.

	Attention + Dropout 50 %	**Eigenschaften Extrahierung**	**Eigenschaften Extrahierung (korrekt)**
BLEU	55.80	53.51	**56.06**
Genauigkeit	4.400	2.600	**11.400**
SPARQL-Genauigkeit	5.400	3.100	**20.600**
Ausführungsgenauigkeit	9.600	9.210	**11.400**
Mikro Präzision	**0.690**	0.045	0.056
Mikro Ausbeute	**0.455**	0.028	0.029
Mikro F1-Score	**0.548**	0.035	0.037
Makro Präzision	0.132	0.139	**0.142**
Makro Ausbeute	0.130	0.141	**0.142**
Makro F1-Score	0.131	0.140	**0.142**

Tabelle 6.34.: Scores der Modelle mit der Extrahierung der Eigenschaften im Vergleich zur Baseline

6.2.4. Kombination von Extrahierungsmechanismen

In diesem Abschnitt werden die Experimente mit Kombinationen von den Extrahierungsmechanismen erläutert.

6.2.4.1. Entity und Ontologieklassen Extrahierung

Zuerst haben wir die Extrahierung der Entitäten mit der Extrahierung der Ontologieklassen vereiningt. Einzeln betrachtet haben beide Extrahierungsverfahren die Scores verbessert und auch die Vereinigung beider Techniken hat die Scores nochmals erhöht.

6.2.4.2. Entity und Eigenschaften Extrahierung

Anschliessend haben wir die Extrahierung der Entitäten mit der Extrahierung der Eigenschaften vereiningt. Auch hier erhoffen wir uns eine Verbesserung der Performanz, wobei zu beachten ist, das nur etwa 50 % der Eigenschaften gefunden werden konnten. Tabelle 6.36 zeigt die Ergebnisse dieses Experiments. Hier ist ersichtlicht das alle Scores bis auf die Mikro Scores steigen.

	Attention + Dropout 50 %	**Ressource + Dropout 50 %**	**Ontologieklassen Extrahierung (korrekt)**	**Entity + Ontologieklassen Extrahierung**
BLEU	55.80	65.83	60.00	**71.19**
Genauigkeit	4.400	16.900	4.210	**25.900**
SPARQL-Genauigkeit	5.400	17.600	5.300	**20.600**
Ausführungsgenauigkeit	9.600	25.290	15.400	**25.900**
Mikro Präzision	0.690	**0.969**	0.340	0.944
Mikro Ausbeute	0.455	0.501	0.492	**0.550**
Mikro F1-Score	0.548	0.661	0.402	**0.695**
Makro Präzision	0.132	0.258	0.175	**0.268**
Makro Ausbeute	0.130	0.260	0.177	**0.268**
Makro F1-Score	0.131	0.258	0.168	**0.267**

Tabelle 6.35.: Scores der Entitäten und Ontologieklassen Extrahierung zusammen angewendet im Vergleich zu den jeweils einzeln angewendeten Verfahren.

	Attention + Dropout 50 %	**Ressource + Dropout 50 %**	**Eigenschaften Extrahierung (korrekt)**	**Entity + Eigenschaften Extrahierung**
BLEU	55.80	65.83	56.06	**70.35**
Genauigkeit	4.400	16.900	11.400	**23.700**
SPARQL-Genauigkeit	5.400	17.600	20.600	**24.400**
Ausführungsgenauigkeit	9.600	25.290	11.400	**28.090**
Mikro Präzision	0.690	**0.969**	0.056	0.288
Mikro Ausbeute	0.455	**0.501**	0.029	0.288
Mikro F1-Score	0.548	**0.661**	0.037	0.286
Makro Präzision	0.132	0.258	0.142	**0.956**
Makro Ausbeute	0.130	0.260	0.142	**0.520**
Makro F1-Score	0.131	0.258	0.142	**0.673**

Tabelle 6.36.: Ergebnisse der Entitäten und Eigenschaften Extrahierung.

6.2.4.3. Entity, Property und Ontologieklassen Extrahierung

Als abschliessende Experiment haben wir alle Extrahierungsmechanismen zusammengefügt und das Modell darauf trainiert. Bei diesem Experiment erwarten wir ein noch bessers Modell. Ausserdem stellt dieses Experiment eine obere Schranke für das aktuelle Modell dar, da bei allen Extrahierungsmechanismen die Variante verwendet wurde, wo die SPARQL-Abfrage bekannt sein muss, somit lässt sich zeigen, ob das Modell verbessert werden muss, da somit die Voraussetzung gilt, das mehr als 90 % aller möglichen Entitäten, ca. 50 % aller Eigenschaften und etwa 100 %

aller Ontologieklassen korrekt ersetzt wurden. Ohne die korrekte SPARQL-Abfrage würden weniger korrekte URI's ersetzt werden, aber mehr falsche URI's ersetzt.

Tabelle 6.37 zeigt die Resultate dieses Experiments.

Problem wenn alle drei zusammen und nicht purged, ontologieklassen und eigenschaftsklassen auf selbe wörter gemappt, nun wird einfach die ontologieklasse verwendet. Es war auch schon nur bei Eigenschaften zu erkennen, dass das Modell ohne purged schlechter war und dieser Fehler pflanzt sich auch hier fort.

	Attention + Dropout 50 %	**Ressource + Dropout 50 %**	**Eigenschaften Extrahierung (korrekt)**	**Ontologieklassen Extrahierung (korrekt)**	**Entity + Property + Ontologieklassen Extrahierung**	**Entity + Property + Ontologieklassen Extrahierung (korrekt)**
BLEU	55.80	65.83	56.06	60.00	51.41	**73.81**
Genauigkeit	4.400	16.900	11.400	4.210	3.400	**26.800**
SPARQL-Genauigkeit	5.400	17.600	20.600	5.300	3.600	**27.600**
Ausführungsgenauigkeit	9.600	25.290	11.400	15.400	5.800	**30.690**
Mikro Präzision	0.690	**0.969**	0.056	0.340	0.152	0.953
Mikro Ausbeute	0.455	0.501	0.029	0.492	0.006	**0.504**
Mikro F1-Score	0.548	**0.661**	0.037	0.402	0.012	0.660
Makro Präzision	0.132	0.258	0.142	0.175	0.061	**0.290**
Makro Ausbeute	0.130	0.260	0.142	0.177	0.067	**0.290**
Makro F1-Score	0.131	0.258	0.142	0.168	0.064	**0.287**

Tabelle 6.37.: Ergebnisse des Modells mit Extrahierung der Entitäten, Eigenschaften und Ontologieklassen.

6.2.4.4. Übersicht der Modelle

Nun folgt eine Übersich der Modelle um die Frage zu beantworten "Was schaffen wir und was wäre wenn die Extrahierungsmechanismen fast perfekte Resultate liefern?". Um diese Frage zu beantworten zeigen die folgenden Abbildungen die wichtigen Scores im Vergleich zwischen den Modellen, wobei alle drei Extrahierungsmechanismen einzeln und einmal vereint aufgeführt werden, zusätzlich wird noch das Baseline Modell aufgeführt. Die Varianten mit der Bezeichung *Limit* stellen hierbei die obere Grenze des jeweiligen Modells dar, da dort die meisten URI's korrekt gematcht werden konnten.

Bei der Extrahierung der Entitäten werden ca. 90 % der korrekten URI's gematcht, bei den Eigenschaften ca. 50 % und bei den Ontologieklassen ca. 100 %.

In Abbildung 6.1 wird der BLEU Score der Modelle verglichen, hier ist ersichtlich, dass der BLEU nicht viel abweicht, da die SPARQL-Abfragen sehr ähnlich sind.

Abbildung 6.2 vergleicht die Genauigkeit der Modelle, die grösste Verbesserung folgt aus der Extrahierung der Entitäten, wobei hier zu beachten ist, dass fast alle Enitäten extrahiert wurden und dies nicht der Fall ist bei den Eigenschaft.

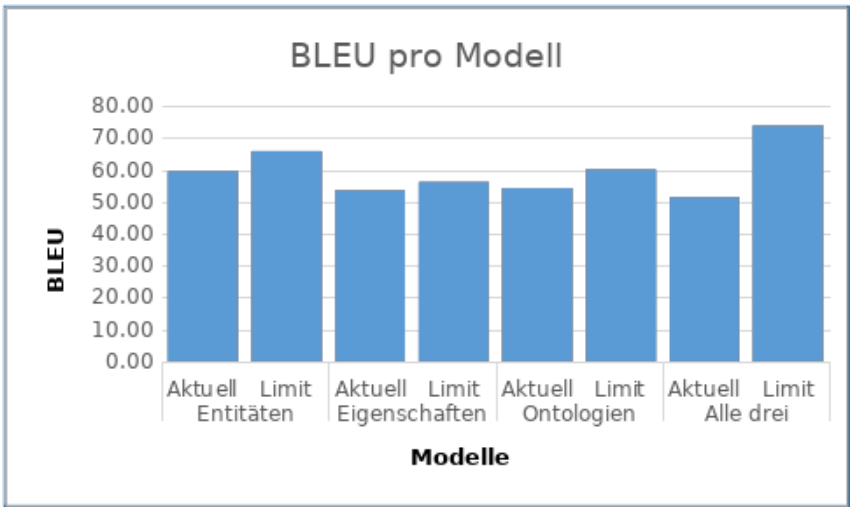


Abbildung 6.1.: Vergleich des BLEU Score für die verschiedenen Extrahierungsmechanismen.

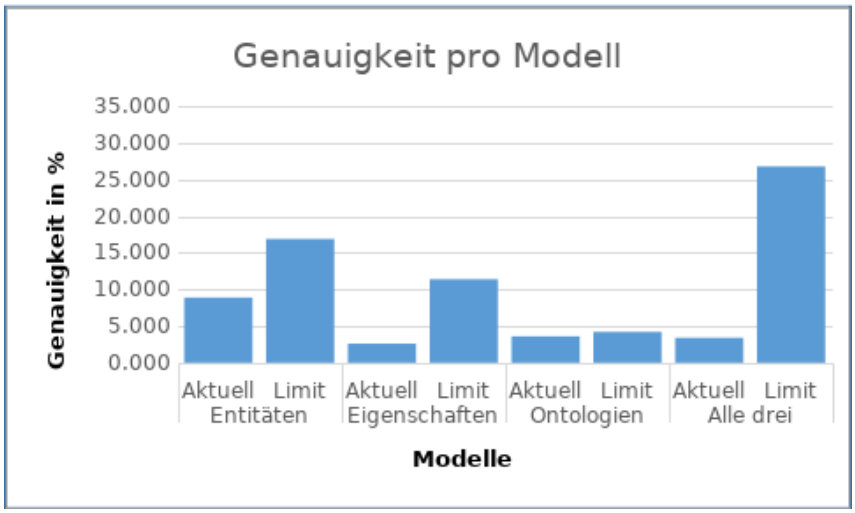


Abbildung 6.2.: Vergleich der Genauigkeit für die verschiedenen Extrahierungsmechanismen.

Der Vergleich der SPARQL-Genauigkeit ist in Abbildung 6.3 hier ist zu erkennen, dass dieser Score in etwa proportional zur Genauigkeit ist, deshalb folgen hier die gleichen Erkenntnisse.

Beim betrachten der Ausführungsgenauigkeit in Abbildung 6.4 fällt auf, dass bei der Extrahierung der Entitäten ein grösser Performanzgewinn durch die korrekte Erkennung resultiert, wie bei den anderen.

Aus Abbildung 6.5, der Vergleich des Mikro F1-Scores, folgt, dass die Extrahierung der Eigenschaften dem Modell nicht hilft.

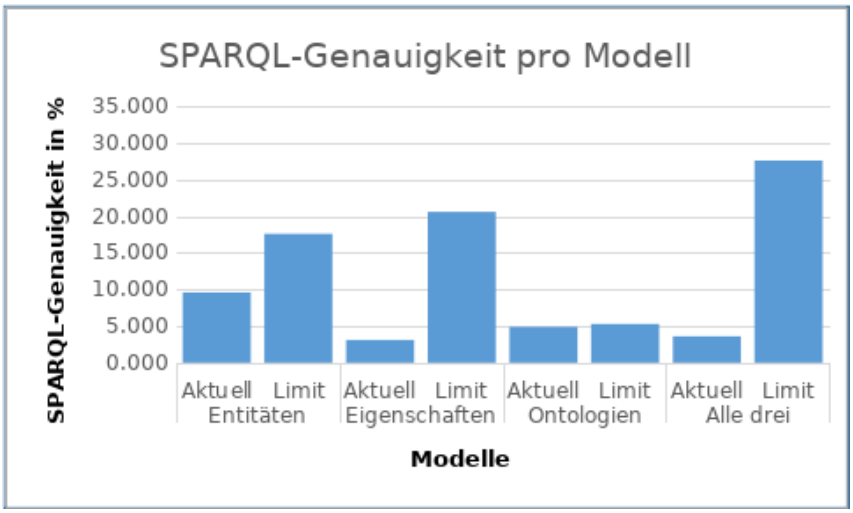


Abbildung 6.3.: Vergleich der SPARQL Genauigkeit für die verschiedenen Extrahierungsmechanismen.

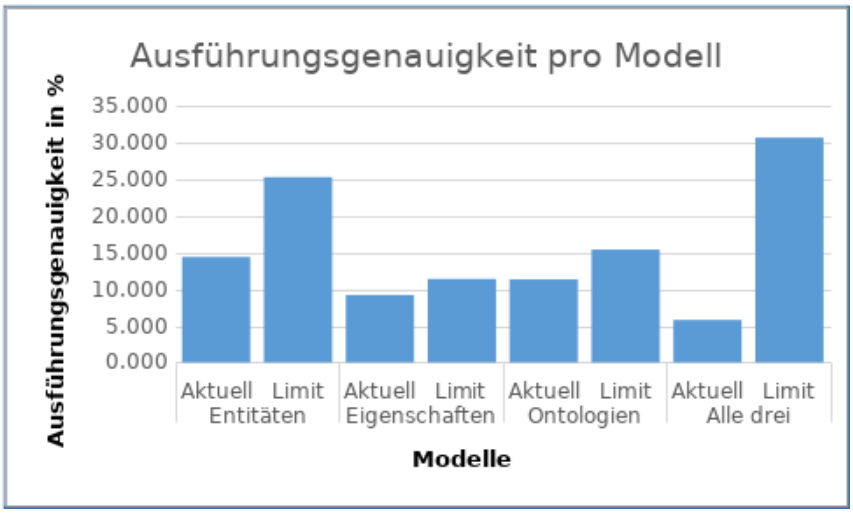


Abbildung 6.4.: Vergleich der Ausführungsgenauigkeit für die verschiedenen Extrahierungsmechanismen.

Wenn man den Makro F1-Score in der Übersicht in Abbildung 6.6 betrachtet, ist zu erkennen, dass diese Kennziffer für alle Modelle sehr ähnlich ist.

6.2.5. Auswertung der Experimente

Um besser zu verstehen wie unseres Modell das Übersetzten von natürlicher Sprache zu SPARQL gelernt hat, haben wir die Resultate der Inferenz Ausgabe ausgewertet und zusammengestellt. Die übersetzten Fragen (Output) des trainierten Models werden wie folgt klassifiziert

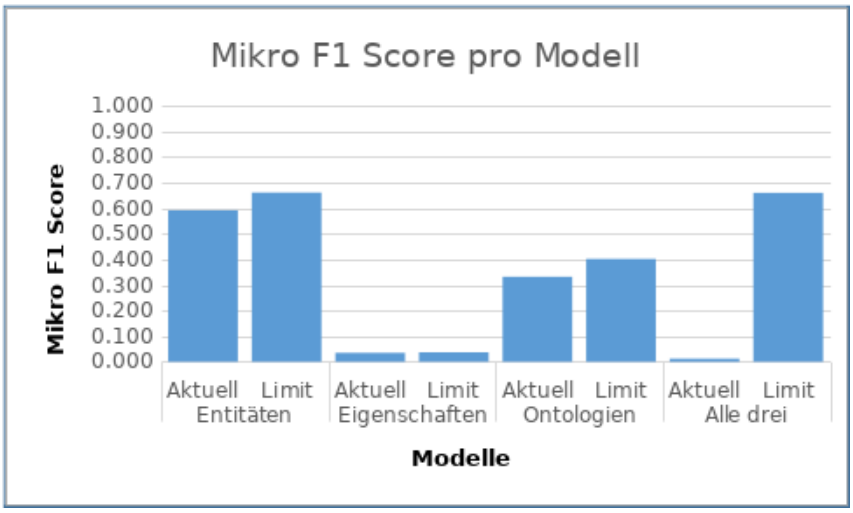


Abbildung 6.5.: Vergleich des Mikro F1-Score für die verschiedenen Extrahierungsmechanismen.

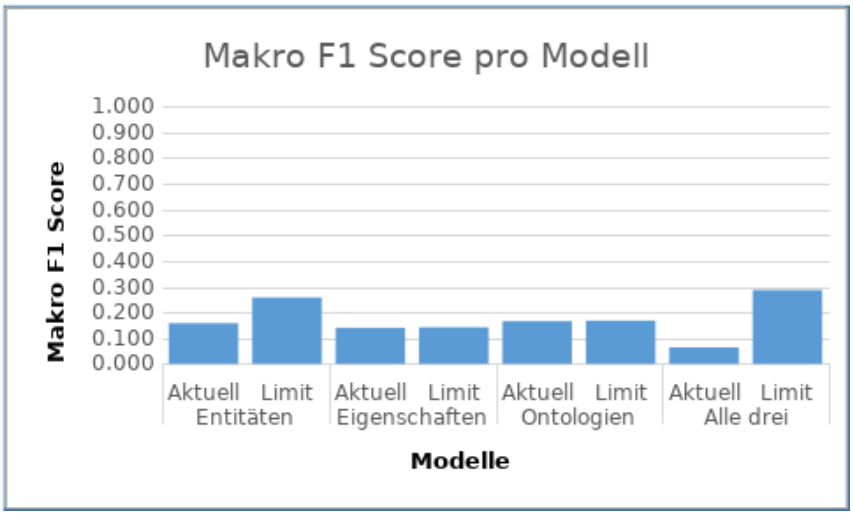


Abbildung 6.6.: Vergleich des Makro F1-Score für die verschiedenen Extrahierungsmechanismen.

Nicht korrekt übersetzte Anfragen → 0
Anfragen mit korrektem Resultat → 1
korrekt übersetzte Anfragen → 2

Die Übersicht 6.38 zeigt Anzahl und den Uniform Resource Identifier TFIDF Median für die jeweilige Klasse von Fragen. Der TFIDF wird auf allen Uniform Resource Identifier dem gesamten Korpus der Trainingsdaten angewendet. Für die Berechnung des TFIDF's werden nur Wörter welche durch *<http://.*>* beschrieben werden können herangezogen.

Beschreibung	Lable	Total	URI TFIDF Median
Nicht korrekt	0	365	6.579
Resultat der Anfrage korrekt	1	27	14.217
Resultat und Anfrage korrekt	2	90	6.526

Tabelle 6.38.: Übersicht der Experiment Auswertung

Um die Verteilung der validierten Fragen besser zu verstehen und möglicherweise eine Idee zu bekommen welche Fragen das Modell vorzugsweise verarbeiten konnte, haben wir noch einen Box Plot vom TFIDF 6.8 und den absoluten Zähler 6.7 der gefunden Ressourcen zusammengestellt. Auch diese Auswertung klärt die nicht wie der Agent vorzugsweise lernt. Auch die Untersuchung anderer Metriken wie z.B Anzahl Wörter in der Frage oder der SPARQL-Abfrage konnten nicht endgültig klären welche Arten respektive Aufbau von Fragen das System bereits gut kann. Die Grafiken zeigen nur das es in allen Klassen Fragen gibt, welche Ressourcen nutzen die sowohl kaum wie auch oft in den Trainingsdaten vorkommen. Wir konnten diese Frage nicht endgültig beantworten, gehen aber davon aus, dass das die Inferenz des Modells auf allen Arten von Fragen verteilt funktioniert oder uns die Metrik nicht bekannt ist.

Die Box Plot Auswertung der Anzahl URI's 6.7 zeigt, dass die Verteilung sehr unregelmässig ist, da es einzelne Ressourcen gibt, wie zum Beispiel `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`, welche oft vorkommen und somit die zu einer extrem linksstelligen Verteilung fuehren. Aus diesem Grund wurde noch zusätzlich die TFIDF Metrik eingeführt, welcher die Daten besser streut.

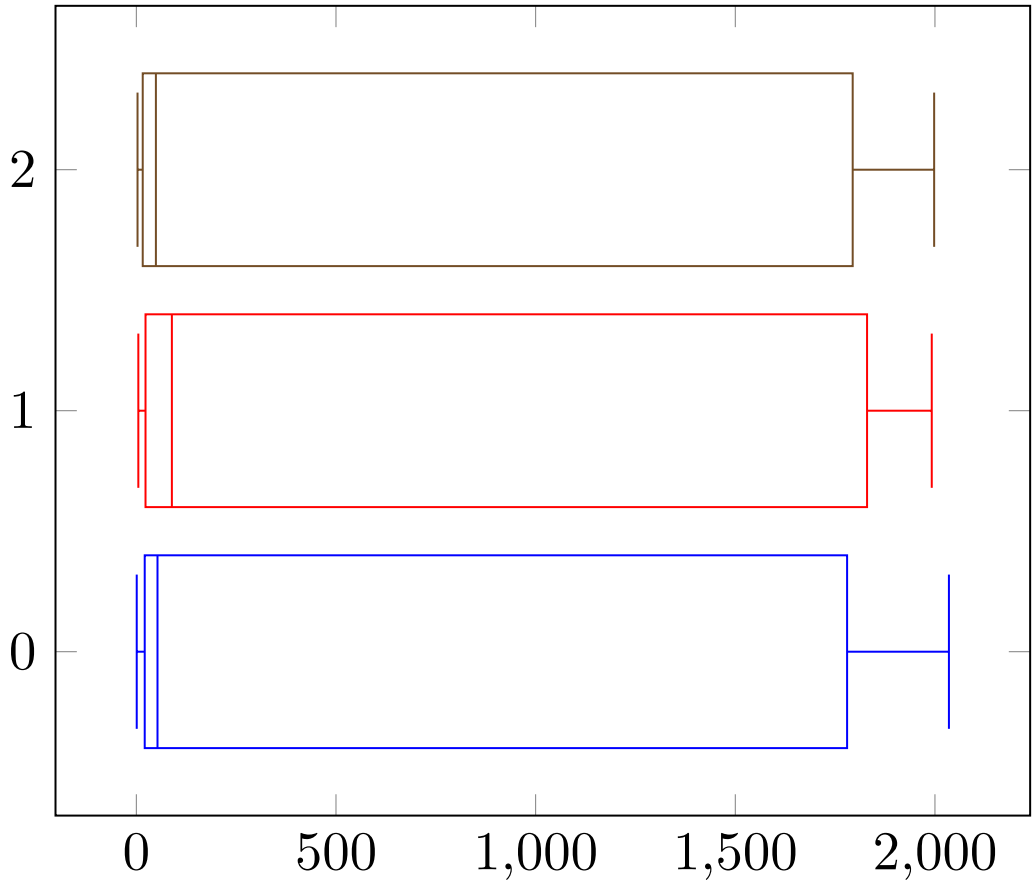


Abbildung 6.7.: Box Plot der Anzahl URI's in Referenzfragen

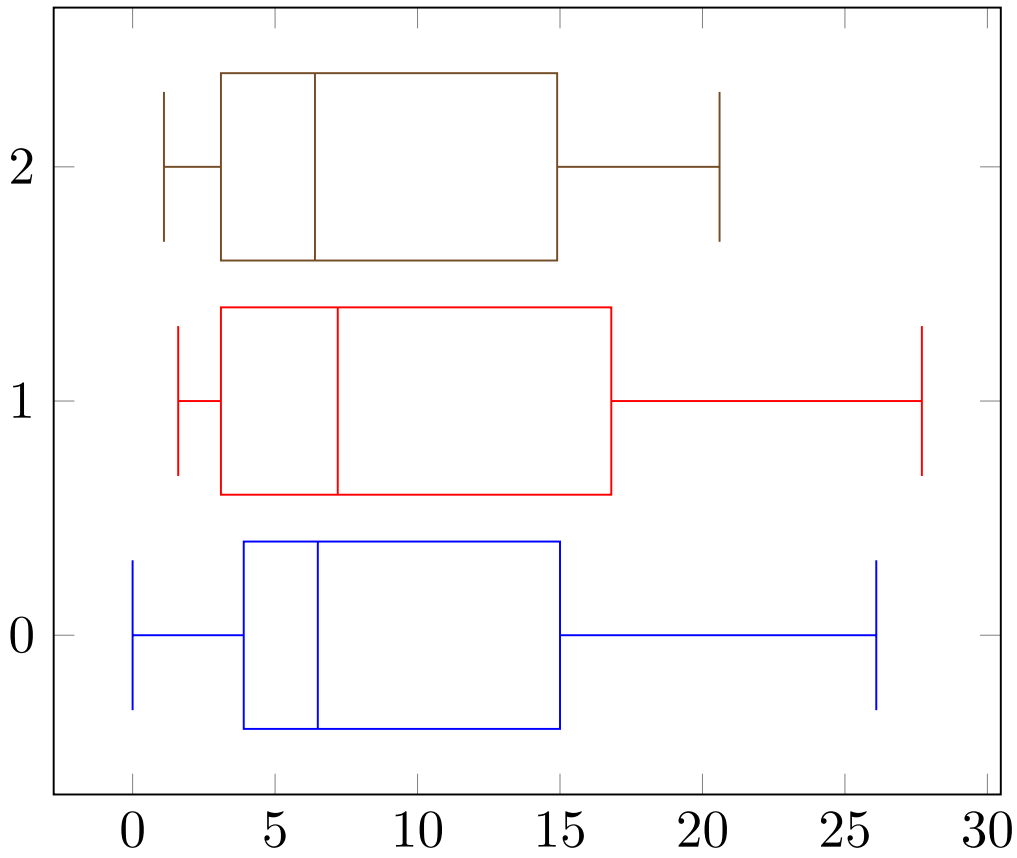


Abbildung 6.8.: Box Plot of URI TFIDF

7. Diskussion und Fazit

7.1. Allgemein

Das Ziel dieser Arbeit war die Verbesserung eines bestehenden Systems. Dabei wurden mehrere Schritte unternommen und weitere Möglichkeiten werden im Kapitel 8 in Aussicht gestellt. Auch wenn das bestehende System in dieser Arbeit relativ betrachtet signifikant verbessert wurde, kann es absolut betrachtet noch nicht in der Praxis eingesetzt werden.

Nichtsdestotrotz haben wir es geschafft ein bestehendes System um mehrere Faktoren zu verbessern. Was während der einzelnen Experimente lange nicht ersichtlich war. Erst durch die Zusammenstellung der Experimenten 6.1, konnten wir besser verstehen was wir erreicht hatten.

7.2. Modell

Die von uns gemachten Anpassungen an dem Modell beschränken sich auf die Einführung der Attention und der Anpassung des Dropouts. Es ist sehr gut möglich, dass weitere Modifikationen am Modell eine Verbesserung der Leistung mit sich gebracht hätten. Verschiedene Ansätze werden im 8 diskutiert. Jedoch sind diese im Fall von RNN's mit sehr langen Laufzeiten verbunden. Die Anpassung der entsprechenden Parameter ist durch die Arbeiten am Preprocessing ein wenig in den Hintergrund geraten. Zum Beispiel stand die Einführung eines Pointer Networks zur Diskussion. Wurde jedoch bei der Priorisierung immer wieder verworfen.

7.3. Preprocessing

Wenn die verwendeten Extrahierungsmechanismen genauer betrachtet werden, ist die Extrahierung der Entitäten das einfachste Problem und kann mit einem F-Score von 0.632 gelöst werden, dagegen wird beim extrahieren der Ontologieklassen im besten Fall ein F-Score von 0.479 erzielt. Noch schwieriger ist die Erkennung der Eigenschaften, hier wird nur ein F-Score von 0.109 erreicht. Interessant ist hierbei, dass das Modell mit den ca. 50 % korrekt erkannten Eigenschaften meistens einen höheren Score erhält als das Modell, bei welchem die Ontologieklassen extrahiert wurden, obwohl dort fast 100 % der Ontologieklassen korrekt erkannt wurden. Enttäuschend ist die Performanz des Modells mit allen drei Extrahierungen, wobei die Varianten verwendet wurden ohne die Beachtung der SPARQL-Abfragen, dort sind alle Metriken relativ schlecht, was aber daran liegt dass die Extrahierung der Eigenschaften ohne berüksichtigen der SPARQL-Abfragen das Modell verschlechtert, wie es auch schon der Fall war, wenn nur die Extrahierung der Eigenschaften betrachtet wurde.

7.4. Trainningsdaten

Ein wichtiger Faktor für weitere Optimierungsmassnahmen ist die Ableitung von Metriken aus den Experimenten, welche aufzeigen wo das Modell bereits gut arbeitet. Ein besseres Verständnis für die Stärken und Schwächen des Agenten sind essentiell für zukünftige Anpassungen. Nebst diesem müsste auch die Menge an Trainingsdaten in den Fokus von zukünftigen Anstrengungen gestellt werden. Da die maschinelle Generierung von Fragen oft zu Fragen mit ähnlichem Aufbau führt, ist es fast unmöglich ohne Hilfe von Benutzern weitere Trainingsdaten zu erzeugen. Eine Herausforderung in diesem Bereich ist es Anreize für Anwender mit entsprechenden Kenntnissen zu finden. Möglicherweise könnte die Einbindung eines spielerischen Elementes die Motivation Anfragen zu formulieren steigern.

Erstaunlicherweise konnten aus den Validierungsdaten keine eindeutigen Metriken für das Lernverhalten des Agenten abgeleitet werden. Weder das Klassifizieren der Fragen, noch die Auswertung verschiedener Kennzahlen wie zum Beispiel 6.7 oder 6.8 konnten dabei helfen besser zu verstehen auf welchen Art der Fragen das Modell gut anspricht. Ein wichtiger Hinweis auf das Verhalten des Modells konnte nicht eruiert werden.

7.5. Zielerreichung

Trotz einer gewissen Ernüchterung wurde ein Fortschritt erreicht und die Grundlage geschaffen um ein erfolgreiches System zu entwickeln. Natürlich hätten wir gerne ein Modell gehabt, welches mehr Fragen übersetzten kann. Der zeitliche Rahmen und die bereits erwähnte Anzahl Trainingsdaten waren von Anfang an unsere Haupt-Herausforderungen. Umso mehr würde es uns freuen, wenn die von uns erarbeitet Verfahren anderen helfen könnten sich mit dieser Thematik weiter zu beschäftigen und eventuell weiter zu entwickeln. Ausserdem können die verschiedenen vorgestellten Techniken auch auf andere Abfragesprachen wie zum Beispiel SQL angewendet werden.

8. Ausblick

Wie im Kapitel Diskussion und Fazit beschrieben, erhält die Extrahierung der Eigenschaften den niedrigsten F-Score, da die Präzision sehr schlecht ist, deshalb empfehlen wir zuerst eine Verringerung der False Positive Rate, indem analog zu der Verwendung der POS-Tags bei der Extrahierung der Ontologieklassen, auch hier die POS-Tags verwendet werden um Muster zu finden, welche zwar als Eigenschaften erkannt werden, aber keine sind. Ausserdem kann man versuchen nur Eigenschaften zu erkennen, für Wörter welche länger als ein bestimmter Schwellwert sind. Anschliessend kann man die Ausbeute erhöhen mithilfe einer Synonymdatenbank. Diese kann genutzt werden, um nicht nur die Wörter selbst mithilfe des Invertierten Index zu matchen, sondern auch Synonyme der Wörter.

Generell können die POS-Tags auch für das genauere extrahieren der Entitäten verwendet werden und gegebenenfalls können noch bessere Muster für die Extrahierung der Ontologien erkannt werden.

Ein weitere Idee wäre die Anpassung der Kostenfunktion speziell für die Struktur von SPARQL, indem die Kostenfunktion aufgrund mehrere Grund Truths berechnet werden könnte, da in dieser Arbeit zwar Metriken verwendet wurden, welche die verschiedenen möglichen Abfragen beachten, aber bei der Minimierung der Kosten beziehungsweise dem Trainieren des Modells dies nicht beachtet wird.

Sollte sich herausstellen, dass das bestehende Modell nicht in der Lage ist bessere übersetzungen zu liefern, wäre eine ähnliche Architektur wie die in der Arbeit *SQLNet* eine Optionm auch bei SPARQL spielt die Reihenfolge der Abfrage nicht immer eine Rolle. In dieser Arbeit wurden zwar die Ausführungsgenauigkeit und die SPARQL-Genauigkeit verwendet, aber diese wurden nur für die Evaluierung benutzt und nicht für das Optimieren des Modells selbst beziehungsweise minimieren der Kosten. Deshalb könnte die Verwendung eines Pointer Netzwerk, wo dieses genutzt wird um pro Slot die Eigenschaft, Ontologieklasse oder Ressource zu wählen verwendet werden. Dabei können beispielsweise dem Modell nur die Eigenschaften zur Verfügung gestellen werden, welche für die gegebene Ontologieklasse oder Ressource in Frage kommt.

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

A. Anhang

A.1. Glossar

Aktivierungsfunktion A *activation function* represents the mathematical function which belongs to a single *neuron*. 24

anaconda *Anaconda* is a software which makes it easier to manage python packages. iv

Ground Truth Als *Ground Truth* wird die korrekte Antwort bezeichnet. 19

Inferenz Als *Inferenz* wird der Vorgang bezeichnet ein bereits trainiertes Modell für die Vorhersage zu verwenden. 5

Levenshtein Distanz Die *Levenshtein Distanz* ist ein Mass für den Vergleich von zwei Texten. 10

Neural Machine Translation Als *NMT* wird die Übersetzung von Texten mithilfe von Neuronalen Netzen bezeichnet. 4

Neuron Ein *Neuron* im Kontext von *NNs* stellt eine Funktion dar. Seine Aufgabe ist es, eine bestimmte Anzahl von Eingangsvariablen zu empfangen und eine bestimmte Anzahl von Ausgangsvariablen zu erzugen. Zusätzlich haben die Eingangsvariablen ein Gewicht. Das *Neuron* erhält mindestens eine Variable und gibt mindestens eine Variable aus. 24

Overfitting *Overfitting* bedeutet das ein Modell, welches lernen kann, zu komplex ist und eine gute bis sehr gute Leistung auf den Trainingsdaten hat, aber auf neuen ungesehenen Daten schlechte Resultate erzielt. 49

pip *pip* is the package manager from python, to download and manage different python packages. iv

Uniform Resource Identifier Englisch für einheitlicher Bezeichner für Ressourcen 18, 69

A.2. Akronym Glossar

MÜ maschinellen Übersetzens 1

NER Named Entity Recognition 50–54

NN Neuronales Netzwerk 24

NN's Künstliche Neuronale Netzwerke 24

POS Part-of-Speech Tagging 34

RDF Resource Description Framework 13, 15

SPARQL SPARQL Protocol And RDF Query Language 1, 68

SQA2018 Scalable Question Answering 2018 40

TFIDF Term Frequency–Inverse Document Frequency 70

W3C World Wide Web Consortium 17

XML Extensible Markup Language 15

B. Technische Dokumentation

In dem folgenden Kapitel wird das bereitstellen der Umbegung und das Verwenden des Systems erklärt.

B.1. Vorbereitungen

Um die Software zu verwenden müssen zuerst mehrere Softwarevoraussetzungen erfüllt werden, ausserdem empfehlen wir ein genügen gutes Hardwaresystem mit einer Grafikkarte vom Hersteller Nvidia, um das Training zu verschnellern.
Wir empfehlen die Verwendung von Python 3.5.2 unter Ubuntu 16.04.

Wenn eine Grafikkarte von Nvidia zur Verfügung steht und diese benutzt werden soll muss die folgende Software installiert werden:

- Nvidia GPU-Treiber für die entsprechende Grafikkarte
- Nvidia Cuda 9.0
- Nvidia Cudnn 7.8

Zusätzlich müssen die folgenden Python-Pakete installiert werden. Um Versionskompatibilitätsprobleme und eventuell abweichende Ergebnisse zu vermeiden, empfehlen wir dringend die genau gleiche Version jedes Pakets zu verwenden. Alle Pakete können mit dem Paket-Manager von python *pip* installiert werden. Wenn anaconda verwendet wird, müssen eventuell trotzdem einige Pakete mit pip installiert werden, da nicht alle Pakete im Paketmanager von anaconda verfügbar sind. Nachfolgend ist die Auflistung aller notwendigen Python-Pakete:

- keras 2.1.5
- tensorflow-gpu 1.6.0 wenn eine Grafikkarte verwendet wird, ansonsten tensorflow 1.6.0
- numpy 1.14.2
- pandas 0.22.0
- sklearn 0.19.1
- scipy 1.0.1
- pyparsing 2.2.0
- rdflib 4.2.2
- SPARQLWrapper 1.8.1
- pyspotlight 0.7.2
- nltk 3.2.5

B.2. Repository Struktur

In den folgenden Tabellen wird die Stuktur des Repositories beschrieben. Tabelle B.1 zeigt die oberste Ebene des Repository und Tabelle B.2 den Inhalt des *<repo>/src* Verzeichnisses, in diesem befindet sich der Quellcode.

Ordner	**Inhaltsbeschreibung**
Data	Hier befinden sich alle Daten, ausserdem werden hier auch die trainierten Modelle gespeichert.
docker	Hier befinden sich die Dockerdateien, welche schon alle benötigten Python-Pakete enthalten.
src	Hier befindet sich der Quellcode des Softwaresystems, dieser Ordner wird in Tabelle B.2 näher erläutert.

Tabelle B.1.: Struktur des Repositories.

Ordner	**Inhaltsbeschreibung**
.	Direkt im Verzeichnis selbst befinden sich die Hauptskripte, welche verwendet werden für das Aufbereiten der Daten, das Training, die Inferenz und die Evaluierung der Modelle und Daten.
Preprocessing	In diesem Ordner befinden sich die Skripts für das aufbereiten der Daten, diese werden über ein Hauptskript aus dem Oberenverzeichnis aufgerufen.
nmt	Dieses Verzeichnis ist ein git-Submodul, in welchem sich der Code des Seq2Seq-Modell selbst befindet. Das Modell wird über ein Skript aus dem Oberenverzeichnis aufgerufen.
utilities	Dieses Verzeichnis beinhaltet Python-Skripte, welche Hilfsfunktionen zur Verfügung stellen.

Tabelle B.2.: Struktur des <repo>/src Verzeichnisses.

B.2.1. Download

Das Softwaresystem selbst kann mit git aus dem Github der ZHAW heruntergeladen werden, ausserdem befindet es sich ebenfalls auf der beigelegten CD. Im Folgenden beziehen wir uns auf den Ort, an dem Sie das Repository befindet als <repo>.

B.2.2. Preprocessing

Das Preprocessing, sollte nur einmal ausgeführt werden damit für alle Experimente die selben Datensätze für das Training und die Validierung verwendet werden.

Listing B.2 zeigt, mit welchem Kommando das gesamte Preprocessing durchgeführt werden kann, dabei muss man sich im <repo>/src Ordner befinden. Bevor das eigentliche Preprocessing gestartet wird, muss der Server für die StanfordCoreNLP Biliothek [13] ausgeführt werden, der entsprechnde Befehl ist in Listing B.1 gezeigt, hier muss der Befehl in dem Ordner ausgeführt werden, indem sich auch die Bibliothek befindet.

Listing B.1: Kommando um das Starten des Stanford CoreNLP Servers.

```
1 java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer -port 9000 -timeout
      15000
```

Listing B.2: Kommando um das Preprocessing durchzuführen.

```
1 python OneTimePreprocessing.py
```

B.2.3. Scripts

In diesem Unterabschnitt erklären wir die Verwendung der bereitgestellten Skripte, dabei erklären wir nur die Skripte, welche für die Durchführung der Experimente notwendig sind. Alle nachfolgend gennanten Skripte finden Sie unter *<repo>/src/*.

B.2.4. Nachvolziehen der Experimente

Das Training kann mit dem Kommando aus Listing B.3 gestartet werden, dabei sind die Variablen “model_identifier”, “pfad_zu_dev_data” und “log_file”, mit den entsprechenden Strings zu ersetzen und man muss sich im *<repo>/src* Verzeichniss befinden.

Listing B.3: Kommando um das Training zu starten

```
1 sh pipeline_train.sh "model_identifier" "pfad_zu_dev_data" > "log_file"
```

Listing B.4 zeigt das Kommando, um das *baseline* Modell zu trainieren, mit *baseline/dev.tok* zu evaluieren und den log in der Datei *sqa2018_baseline.log* zu speichern.

Tabelle B.3.: NMT/src directory structure

Name	Description
OneTimePreprocessing.py	Dieses Skript führt das Preprocessing aus.
analyzer.py	Mit diesem Skript kann ein Modell evaluiert werden.
ask.sh	Dieses Skript kann für die Inferenz verwendet werden.
build_inverted_indices.py	Mit diesem Skript können die Invertierten Indexe erstellt werden.
eval_each.sh	Mit diesem Skript wird für ein Modell jeder Zwischenschritt berechnet.
eval_one_ckpt.sh	Mit diesem Skript wird für ein Modell ein Zwischenschritt berechnet.
exam_dict.py	Dieses Skript kann für das Auslesen der Invertierten Indexe verwendet werden.
find_best_models.py	Dieses Skript findet die besten Zwischenschritte aus einem Evaluierungslog.
interpreter.py	Dieses Skript wird verwendet um das Resultat der Inferenz von Token in normales SPARQL umzuwandeln.
pipeline_train.sh.py	Dieses Skript wird verwendet für das Training der Modelle verwendet.

Listing B.4: Beispiel um das Training des Baseline Modells zu starten

```
1 sh pipeline_train.sh baseline baseline/dev.tok > sqa2018_baseline.log
```

Nach dem Training kann der Befehl aus Listing B.5 verwendet werden um alle gespeicherten Zwischenschritte zu evaluieren.

Listing B.5: Kommando für das Evaluieren aller Zwischenschritte.

```
1 bash eval_each.sh "model_identifier" > ../Data/"model_identifier"_model/log_eval_each___"
      model_identifier".log
```

Anschliessend kann der Befehl aus Listing B.6 verwendet werden um den besten Zwischenschritt bezüglich der Metriken zu finden.

Listing B.6: Befehl für das Finden des besten Zwischenschritt.

```
1 python find_best_models.py "model_identifier"
```

B.2.5. Ausführen der Inferenz

Die Inferenz erfolgt mithilfe eines Shell-Skripts. Wie in der Tabelle B.2 beschrieben, wird das Skript *ask.sh* verwendet, um die Inferenz auszuführen. Beim ausführen des

Skripts, wird mithilfe eines Parameters das Modell ausgewählt und die Inferenz wird auf eine Datei angewendet. Beispiel B.2.1 zeigt das Kommando für die Inferenz von einer Datei und Beispiel.

Beispiel B.2.1. *sh ask.sh "model_identifierpath_to_ckpt_without_filestep_identifier input_file"*

Literatur

[1] Dzmitry Bahdanau, Kyunghyun Cho und Yoshua Bengio. "NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE". In: *ICLR* (2015). URL: https://arxiv.org/pdf/1409.0473.pdf.

[2] Yoshua Bengio, Patrice Simmard und Fransconi Paolo. "Learning Long-Term Dependencies with Gradient Descent is Difficult". In: *IEEE Transactions On Neural Networks* 5 (1994), S. 157. URL: http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf.

[3] Joachim Daiber u. a. "Improving Efficiency and Accuracy in Multilingual Entity Extraction". In: *ACM* (2012). URL: http://jodaiber.de/doc/entity.pdf.

[4] Daniel Jurafsky und James H. Marti. "Speech and Language Processing (3rd ed. draft)". In: (2017). URL: https://web.stanford.edu/{~}jurafsky/slp3/10.pdf.

[5] Uni Hamburg. *RDF und RDF Schema*. URL: https://nats-www.informatik.uni-hamburg.de/pub/SemWeb/TeilnehmerBeitraege/rdf.pdf (besucht am 12. 03. 2018).

[6] Sepp Hochreiter und Schmidhuber Jurgen. "LONG SHORT-TERM MEMORY". In: *Neural Computation* 9.8 (1997), S. 1735–1780. URL: http://www.bioinf.jku.at/publications/older/2604.pdf.

[7] Nicolas Hoferer u. a. "Neural Machine Translation". In: *Project Thesis Autumn 2017 ZHAW* (2017). URL: https://ml-nic.com/wp-content/uploads/2018/02/PA16{_}Neural-Machine-Translation-by-Einardan{_}and{_}Hofernic.pdf.

[8] Matthew Hutson. "Computer chip mimics human brain, with light beams for neurons". In: *Science* (2017). DOI: 10.1126/science.aan6998. URL: http://www.sciencemag.org/news/2017/06/computer-chip-mimics-human-brain-light-beams-neurons.

[9] Minh-Thang Luong, Eugene Brevdo und Rui Zhao. "Neural Machine Translation (seq2seq) Tutorial". In: *https://github.com/tensorflow/nmt* (2017). URL: https://github.com/tensorflow/nmt.

[10] Minh-Thang Luong, Eugene Brevdo und Rui Zhao. "Neural Machine Translation (seq2seq) Tutorial". In: *https://github.com/tensorflow/nmt* (2017). URL: https://github.com/tensorflow/nmt.

[11] Minh-Thang Luong, Hieu Pham und Christopher D Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: (2015). arXiv: arXiv:1508.04025v5. URL: https://arxiv.org/pdf/1508.04025.pdf.

[12] Christopher D. Manning, Prabhakar Raghavan und Hinrich Schütze. *Introduction to Information Retrieval*. Online Edition. Cambridge University Press, 2008, S. 32. ISBN: 0521865719. URL: https://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf.

[13] Christopher D Manning u. a. "The Stanford CoreNLP Natural Language Processing Toolkit". In: *52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations* (2014), S. 55–56. URL: https://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf.

[14] Pablo N Mendes u. a. "DBpedia Spotlight: Shedding Light on the Web of Documents". In: *I-Semantics* (2011). URL: https://www.dbpedia-spotlight.org/docs/spotlight.pdf.

[15] Merriam-Webster. *How many words are there in English?* URL: https://www.merriam-webster.com/help/faq-how-many-english-words (besucht am 31. 12. 2017).

[16] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (besucht am 11. 12. 2017).

[17] Razvan Pascanu, Tomas Mikolov und Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: (2012). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.421.8930{\&}rep=rep1{\&}type=pdf.

[18] M. F. Porter. "Readings in Information Retrieval". In: (1997). Hrsg. von Karen Sparck Jones und Peter Willett, S. 313–316. URL: http://dl.acm.org/citation.cfm?id=275537.275705.

[19] Pranoy Radhakrishnan. *Introduction to Recurrent Neural Network*. 2017. URL: https://towardsdatascience.com/introduction-to-recurrent-neural-network-27202c3945f3 (besucht am 11. 12. 2017).

[20] Sean Robertson. *Practical PyTorch: Translation with a Sequence to Sequence Network and Attention*. URL: https://github.com/spro/practical-pytorch/blob/master/seq2seq-translation/seq2seq-translation.ipynb (besucht am 11. 12. 2017).

[21] R Rojas. "The Backpropagation Algorithm". In: *Neural Networks* (1996). URL: https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf.

[22] Muhammad Saleem u. a. "Question Answering Over Linked Data: What is Difficult to Answer? What Affects the F scores?" In: *ESWC* (2017). URL: http://ceur-ws.org/Vol-1932/paper-02.pdf.

[23] Tim Salimans und Diederik P Kingma. "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks". In: (). URL: https://arxiv.org/pdf/1602.07868.pdf.

[24] Applied Sciences und Applied Information Technology. "Artificial Intelligence Introduction to Semantic Technologies Prof . Dr . Kurt Stockinger , Ana Sima Fall Semester 2017". In: *ZHAW* (2017).

[25] Rico Sennrich, Barry Haddow und Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". In: (2016). arXiv: arXiv:1508.07909v5. URL: https://arxiv.org/pdf/1508.07909.pdf.

[26] Steven W. Smith. *Neural Network Architecture*. URL: http://www.dspguide.com/ch26/2.htm (besucht am 12. 12. 2017).

[27] Richard Socher und Stanford University. "CS224n: Natural Language Processing with Deep Learning". In: *Stanford University* (2017). URL: http://web.stanford.edu/class/cs224n/.

[28] Daniil Sorokin und Iryna Gurevych. "End-to-end Representation Learning for Question Answering with Weak Supervision". In: *ESWC* (2017). URL: https://www.ukp.tu-darmstadt.de/fileadmin/user{_}upload/Group{_}UKP/publikationen/2017/Sorokin{_}Gurevych{_}QALD7{_}SharedTask{_}CameraReady.pdf.pdf.

[29] Tommaso Soru u. a. "SPARQL as a Foreign Language". In: (2017). arXiv: 1708.07624. URL: http://arxiv.org/abs/1708.07624.

[30] M Togni und S Panighetti. "Bachelor thesis: Information retrieval with context-recall in human-computer conversations". In: *ZHAW* (2017).

[31] Ricardo Usbeck u. a. "7th Open Challenge on Question Answering over Linked Data (QALD-7)". In: (). URL: https://svn.aksw.org/papers/2017/ESWC{_}2017{_}QALD/public.pdf.

[32] Venelin Valkov. *Creating a Neural Network from Scratch - TensorFlow for Hackers (Part IV)*. 2017. URL: https://medium.com/@curiousily/tensorflow-for-hackers-part-iv-neural-network-from-scratch-1a4f504dfa8 (besucht am 11. 12. 2017).

[33] Oriol Vinyals u. a. "Pointer Networks". In: *NIPS* (2015). URL: https://papers.nips.cc/paper/5866-pointer-networks.pdf.

[34] Xiaojun Xu, Chang Liu und Dawn Song. "SQLNet: GENERATING STRUCTURED QUERIES FROM NATURAL LANGUAGE WITHOUT REINFORCEMENT LEARNING". In: (2017). URL: https://arxiv.org/pdf/1711.04436.pdf.

[35] Victor Zhong, Caiming Xiong und Richard Socher. "SEQ2SQL: GENERATING STRUCTURED QUERIES FROM NATURAL LANGUAGE USING REINFORCEMENT LEARNING". In: *CoRR* abs/1709.0 (2017). arXiv: 1709.00103. URL: https://arxiv.org/pdf/1709.00103.pdf.

[36] Lei Zou u. a. "Natural Language Question Answering over RDF — A Graph Data Driven Approach". In: *SIGMOD14* (2014). DOI: 10.1145/2588555.2610525. URL: http://59.108.48.38:8080/ICSTWIPWeb/press/paper/13-hewenqiang-SIGMOD14.pdf.