Entity Linking in der Praxis

Maschinelle Kontext-Abstraktion durch Kategorien beim Entity-Linking

2. Praxisarbeit

vorgelegt am 22.11.2021

Fakultät Wirtschaft

Wirtschaftsinformatik Data Science

WWI 2019F

von

Tobias Hölzer

Name des Betreuers in der Ausbildungsstätte:	DHBW Stuttgart:
IBM Andreas Schneider Senior Technical Specialist	Holzweißig Studiendekan Wirtschaftsinformatik
Unterschrift des Betreuers	

Um den Zugriff auf komplexe Wissensdatenbanken zu erleichtern soll eine Datenabfrage mit natürlicher Sprache entwickelt werden. Als Teil dieses Systems soll ein Entity-Linker textliche Erwähnungen von Entitäten in einer Frage zu den entsprechenden Entitäten in der Wissensdatenbank verlinken damit darauf basierend eine Antwort generiert werden kann. Diese Arbeit schlägt eine neue Architektur für ein Entity-Linking System vor, welche auf mit maschinellem Lernen trainierbaren Modellen basiert. Diese Modelle machen sich bestehende Methoden zur Abstraktion der texlichen Entitätserwähnung und ihres Kontexts zu nutze, in dem Kategorien anhand der Erwähnungen generiert und in einem zweiten Schritt über ein Transformer Modell mit Attributen aus der Wissensdatenbank abgeglichen werden. Es konnte gezeigt werden, dass diese neue Architektur zwar vielversprechend ist, jedoch zu komplex und somit zu Aufwendig bezüglich des Speicheraufwands und der Rechenzeit.

Inhaltsverzeichnis

Abkürzungsverzeichnis III

Abbildungsverzeichnis IV

List of Tables V

1 Einführung 1

2 Theoretischer Hintergrund des Entity-Linkings 3
- 2.1 Einführung Künstliche neuronale Netze . . . 3
  - 2.1.1 Feed Forward- und Convolutional Neural Networks . . . 6
  - 2.1.2 Fortgeschrittene neuronale Netze . . . 7
  - 2.1.3 Transformer Modelle . . . 7
- 2.2 Entity-Linking als Aufgabe des NLP . . . 8
  - 2.2.1 Aufbau von Wissensdatenbanken . . . 10
  - 2.2.2 Unteraufgabe Candidate-Generation . . . 10
  - 2.2.3 Unteraufgabe Candidate-Ranking . . . 12
- 2.3 Entity-Typing als Aufgabe des NLP . . . 12
- 2.4 Analyse existierender Entity-Linking Systeme . . . 13
  - 2.4.1 Abstraktion durch Kategorien . . . 13
  - 2.4.2 Attribut-Schemata für Verlinkungen zu unterschiedlichen Wissensdatenbanken . . . 15

3 Auswahl und Begründung der Forschungsmethodik 17

4 ACCEL Entwicklung und Architektur 18
- 4.1 Architektur des ACCEL Entity-Linkers . . . 18
  - 4.1.1 Aufbau des Entity-Typers . . . 18
  - 4.1.2 Architektur des Candidate-Rankers . . . 20
  - 4.1.3 Aufbau des Experiments zur Evaluierung des Entity-Typers und Candidate-Rankers . . . 22
- 4.2 Überarbeitung der ACCEL Architektur . . . 23
  - 4.2.1 Architektur des überarbeiteten Candidate-Ranker . . . 24
  - 4.2.2 Bestimmung der Hyperparameter des Candidate-Rankers . . . 25

5 Evaluierung und Test des ACCEL Entity-Linker 26

6 Diskussion und Vergleich mit weiteren Systemen 27
- 6.1 Vertrauenswürdigkeit der ACCEL KI . . . 27
- 6.2 Ausblick . . . 28

7 Fazit 29

Literaturverzeichnis 30

Appendix 34

Abkürzungsverzeichnis

ACCEL Abstract Context through Categories for Entity-Linking

BERT Bidirectional Encoder Representations from Transformers

CNN Convolutional Neural Network

DSR Design Science Research

ELMo Embeddings from Language Models

ET4EL Entity-Typing for Entity-Linking

LSTM Long short-term memory

KI Künstliche Intelligenz

KNN Künstliches neuronales Netz

MLF Multi-Layer Feed-Forward Neural Networks

NLP Natural Language Processing

RNN Recurrent Neural Network

SQL Structured Query Language

SPARQL SPARQL Protocol And RDF Query Language

Abbildungsverzeichnis

1 Beispiel Entity Linking 1

2 Neuron 4
3 MLF 6
4 Entity Linking Einordnung 9
5 Attribut Schema 15

6 Architektur Entity-Typer 19
7 Architektur Candidate Ranker 21
8 Lineares Aufwärmen der Lernrate 23
9 Architektur Candidate Ranker V2 24

Tabellenverzeichnis

1 Suche nach den besten Hyperparameter 25

2 Vergleich verschiedener Entity-Linker mit ACCEL 28

1 Einführung

Eines der fordernsten Probleme unseres Jahrhunderts ist der Klimawandel.[1] Um einer weiteren Erwärmung des Klimas entgegenzuwirken, müssen die Emissionen von Treibhausgasen in einem höchstmöglichen Maße reduziert werden und die Immissionen dieser Gase in einem erhöhten Maße erhöht werden. Hier spielt unter anderem die Materialforschung eine entscheidende Rolle, da sie klimaneutrale oder sogar klimapositive Baustoffe, Kunststoffe und weitere Stoffe entwickeln kann. Aufgrund des ständigen wachsenden Wissens dieses Forschungsfeldes helfen sogenannte Wissensdatenbanken dabei, dieses Wissen zu speichern, zu strukturieren und den Forschenden zugänglich zu machen. Jedoch müssen diese Datenbanken zurzeit mit komplizierten Abfragesprachen wie Structured Query Language (SQL) oder SPARQL Protocol And RDF Query Language (SPARQL) von den Forschenden bedient werden. Um dies zu erleichtern, soll eine Anwendung entwickelt werden, welche die Datenbanken über natürlicher Sprache zugreifbar werden lässt. Dazu soll Natural Language Processing (NLP)[2] genutzt werden, wobei die Anwendung in eine Aufgabe zur Beantwortung von Fragen und eine zum Bestandsaufbau der Wissensdatenbank unterteilt ist. Ein wichtiger Teil beider Aufgaben ist das sogenannte Entity-Linking [3], welches Erwähnungen von Entitäten in einem Text erkennt und zu einer entsprechenden Entität in einer Wissensdatenbank verlinkt.[4]

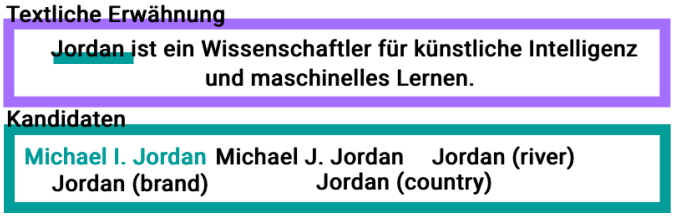

Abb. 1: Die Erwähnung *Jordan* soll in diesem Kontext zum englischen Forscher *Michael I. Jordan* verlinkt werden. Hierbei ist der Begriff *Jordan* eine Abkürzung für den vollständigen Namen des Wissenschaftlers. Der bloße Begriff *Jordan* kann jedoch auch mehrere Bedeutungen haben. (Beispiel Entity Linking)[5]

Bei der Bewältigung dieser Aufgabe ergeben sich mehrere Probleme aufgrund der Inkonsistenz natürlicher Sprache. So kann eine Entität mehrere unterschiedliche Synonyme besitzen, ein Beispiel hierzu wäre der Großkonzern *IBM* und sein Spitzname *Big Blue*. Genauso müssen Abkürzungen zu ihren ausgeschriebenen Langformen verlinkt werden, wie zum Beispiel der Begriff CO^2 ausgeschrieben *Kohlenstoff-Dioxid* bedeutet und somit beide Begriffe zur selben Entität verlinkt werden müssen. Ebenso kann es vorkommen, dass ein Begriff zu unterschiedlichen Entitäten verlinkt werden kann. Dies tritt vor allem bei Personen und Eigennamen auf wie in der Abbildung 1 beispielhaft an dem Begriff *Jordan* gezeigt werden kann. Des Weiteren kann

[1] Vgl. Masson-Delmotte u. a. 2021, S. 5 ff.
[2] z. Dt. Verarbeitung natürlicher Sprache
[3] z. Dt. Verlinken von Entitäten
[4] Vgl. Ruder 2021; Vgl. dazu ausführlich: Shen/Wang/Han 2014, S.443 f.
[5] Mit Änderungen entnommen aus: Shen/Wang/Han 2014, S. 444

es auch vorkommen, dass sich die Entität einer Erwähnung überhaupt nicht in der Wissensdatenbank befindet. Dieses Problem stellt eine eigene Unteraufgabe des Entity-Linkings dar, die Unlinkable-Mention-Prediction[6] oder auch NIL-Prediction[7] genannt.[8] Aufgrund der Komplexität des Problems werden heutzutage Modelle mit maschinellem Lernen darauf trainiert, Erwähnungen mit ihren Entitäten zu verlinken. Somit spielt auch die Datenqualität, mit der ein solches Modell trainiert wird, eine wichtige Rolle, was sich vor allem bei kleinen und spezialisierten Wissensdatenbanken aufgrund der geringen Anzahl an Entitäten und annotierten Textbeispielen als weiteres Problem herausstellt. Daher wird teilweise mit unterschiedlichen Wissensdatenbanken trainiert, wobei unterschiedliche, bzw. inkonsistente Daten-Schemata genutzt werden, was abermals ein Problem darstellen kann.[9]

Ziel dieser Arbeit ist es ein Artefakt zu entwickeln, welches Erwähnungen in einem Text mit Entitäten in einer kleinen aber spezialisierten Wissensdatenbank verlinkt. Dazu sollen verschiedene Modelle mit Methoden des maschinellen Lernens trainiert und gegen verschiedene Wissensdatenbanken getestet und evaluiert werden. Hierzu sollen existierende Modelle aus der Fachliteratur analysiert und ihre Komponenten zu einem neuen Modell zusammengeführt werden. **Die zentrale Fragestellung dieser Arbeit lautet daher, ob die einzelnen Komponenten beider Modelle zusammen harmonieren und sich gegenseitig positiv ergänzen.**

Hierzu wird die Design Science Research (DSR) Methodik verwendet, welche von Gregor/Hevner 2013 genauer beschrieben wurde. Bei der Durchführung dieser Methodik wird ein Artefakt über zwei Entwicklungs-Iterationen entwickelt und evaluiert, welches auf dem theoretischen Vorwissen bisheriger Arbeiten basiert. Bei der Entwicklung des Artefakts werden durch Laborexperimente Aufbau und Parameter des Artefakts bestimmt, woraufhin das Artefakt gegen im Forschungsfeld übliche Datensätze und Metriken getestet wird.

Der Aufbau dieser Arbeit hält sich an die vorgeschlagene Struktur von Gregor/Hevner 2013: Einführung, Literaturanalyse, Methodik, Artefakt Beschreibung, Evaluierung, Diskussion und Fazit.[10] Entsprechend stellt diese Arbeit im nächsten Kapitel (2) den theoretischen Hintergrund des Entity-Linkings anhand aktueller Forschungsliteratur dar, wobei insbesondere auf zwei Modelle eingegangen wird, welche Grundlage des Artefakts sein werden. In Kapitel 3 wird genauer auf den DSR Methodik eingegangen, sowie Forschungsfeld übliche Datensets und Metriken erläutert. Die Methodik wird in Kapitel 4 angewandt, indem die iterative Entwicklung des Artefakts beschrieben und evaluiert wird. Ausführliche Tests des Artefakts werden in Kapitel 5 genauer erläutert und die Ergebnisse in Kapitel 6 diskutiert. Ebenso wird im sechsten Kapitel ein Ausblick über weiteren Aspekte für zukünftige Forschung, welche sich aus dieser Arbeit ergeben, gewährt. Das Kapitel 7 fasst diese Arbeit abschließend zusammen. Im Anhang 1 befindet sich ein Glossar mit deutschen Übersetzungen für englische Fachwörter.

[6] z. Dt. Vorhersage von nicht verknüpfbaren Erwähnungen
[7] z. Dt. NULL-Vorhersage
[8] Vgl Shen/Wang/Han 2014, S. 445
[9] Vgl. Vyas/Ballesteros 2021, S. 1 f. Vgl. Sil u. a. 2012, S. 116 f.
[10] Gregor/Hevner 2013, S. 349 ff.

2 Theoretischer Hintergrund des Entity-Linkings

Die Aufgabe des Entity-Linkings gehört zur Aufgabengruppe des NLP. Zum Forschungsfeld des NLP gehört die „Entwicklung von computergestützten, linguistischen Modellen, sowie Prozesse um praktische Probleme des Verstehens natürlicher Sprache zu lösen.“[11] Eine weitere Aufgabe des NLP ist das sogenannte *Entity-Typing*[12], welche in dieser Arbeit für die Lösung der Entity-Linking Aufgabe genutzt wird und dessen Ziel es ist Typen oder auch Kategorien zu einer Entität, bzw. einer Erwähnung dieser zuzuordnen.[13] Aufgrund der Entwicklung immer schnellerer Computer und den Fortschritten bei der parallelen Verarbeitung in Computerprogrammen in den letzten Jahren werden in den meisten Bereichen des NLP heutzutage Modelle basierend auf Künstliches neuronales Netz (KNN) eingesetzt, welche aus mehreren Milliarden trainierbaren Parametern bestehen.[14] Somit wird heutzutage zum Großteil maschinelles Lernen sowie große Datensets eingesetzt, um NLP Aufgaben zu lösen. Da solche komplizierten und komplexe Modelle für Menschen, vor allem für fachfremden Nutzern dieser Modelle, immer schwerer zu verstehen und ihre Entscheidungen immer schwerer nachzuvollziehen sind, beschäftigt sich das Forschungsfeld der *Trustful AI*[15] mit Methoden, um Modelle erklärbar und vertrauenswürdig zu machen.[16] Im Folgenden werden Methoden des maschinellen Lernens, Arten von KNN sowie die Aufgabenbereiche des Entity-Linking sowie Entity-Typing näher beschrieben.

2.1 Einführung Künstliche neuronale Netze

Die Verwendung von KNN ist aus dem NLP Bereich nicht mehr wegzudenken, da sie komplizierte und abstrakte Sachverhalte gut darstellen können, wozu herkömmliche Algorithmen nicht in der Lage wären. KNN basieren auf dem Beispiel der neuronalen Netze in der Natur, sind jedoch deutlich weniger entwickelt. So finden sich 175 Milliarden künstliche Synapsen (trainierbare Parameter) in dem derzeit größten Sprachmodell *GPT-3* von Brown u. a. 2020.[17] Ein menschliches Gehirn hingegen kann deutlich mehr Synapsen aufweisen. Zwar gibt es viele unterschiedliche Schätzungen, wovon jedoch die meisten von einer Anzahl von mehr als einer Billion Synapsen ausgehen.[18] Ebenso kann davon ausgegangen werden, dass die Verknüpfungen zwischen den Nervenzellen beim Menschen deutlich weiter entwickelt ist, als es bei dem *GPT-3* Sprachmodell der Fall ist. Aber auch schon kleine KNN können einfache aber abstrakte Aufgaben lösen. So kann zum Beispiel ein Modell mit ungefähr 12.000 trainierbaren Parametern Ziffern auf Bildern bestimmen.[19]

[11] Otter/Medina/Kalita 2021, S. 604
[12] z. Dt. Typisierung von Entitäten
[13] Vgl. Lin/Ji 2019, S. 6197 f.
[14] Vgl. Otter/Medina/Kalita 2021, S.604
[15] z. Dt. vertrauenswürdige Künstliche Intelligenz (KI)
[16] z.B. Rossi u. a. 2019
[17] Vgl. Brown u. a. 2020, S. 5
[18] Vgl. von Bartheld/Bahney/Herculano-Houzel 2016, S. 3881
[19] Vgl. Nielsen 2015

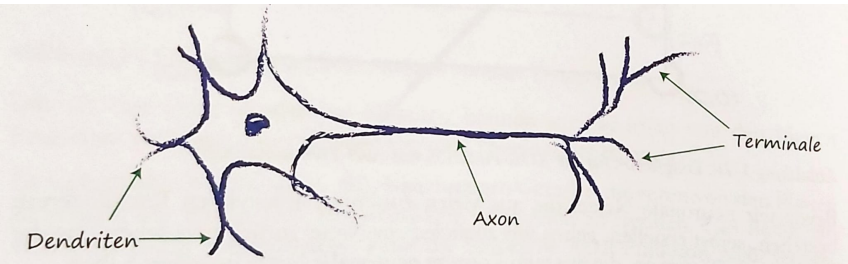

Abb. 2: Neuron/Nervenzelle als Grundeinheit eines biologischen Gehirns. (Neuron)[20]

Schaut man sich eine Nervenzelle wie in Abbildung 2 an, so kann diese auf einen Zellkern, ein Axon und Dendriten reduziert werden. Aufgabe der Nervenzelle ist es Signale von anderen Nervenzellen an andere Nervenzellen weiterzuleiten, aber nur wenn die Signale ausreichend stark sind.[21] Abstrahiert stellen die Dendriten Eingabeverbindungen von Signalen von anderen Knoten, den Zellkernen, dar. Über eine Aktivierungsfunktion wie zum Beispiel die Sigmoid-Funktion σ wird bestimmt, ob oder wie stark das Signal über die Ausgabeverbindung, das Axiom, weitergeleitet wird. Die Abstraktion der Nervenzelle nimmt also mehrere Eingangs-Signale auf und verarbeitet sie zu einem Ausgangs-Signal. Diese Abstraktion wird auch *Perzeptron* genannt und wurde von Rosenblatt 1962 vorgestellt.

Da in der Natur die Dendriten und Axiome die Signale der Nervenzelle unterschiedlich stark weiterleiten, können auch den Verbindungen zwischen den Knoten des Perzeptrons unterschiedliche Gewichte $W = \{w_1, w_2, w_3, ...w_i\}$ zugewiesen werden.[22] Angenommen x_1, x_2 und x_3 sind die Eingangswerte eines Perzeptrons bestehend aus einem Knoten mit drei Eingangsverbindungen und einer Ausgangsverbindung, so lässt sich der Ausgangswert y mit $y = \sigma(w_1 * x_1 + w_2 * x_2 + w_3 * x_3)$ berechnen. Bei entsprechender Wahl der Gewichte W kann diese Architektur kann als einfache Entscheidungsfindung genutzt werden.[23] Um die Gewichte anhand existierender Daten zu bestimmen wird maschinelles Lernen genutzt, welches die Gewichte iterativ über die Backpropagation [24] anpasst.[25] Durch den Zusammenschluss mehrerer trainierten Perzeptronen entstehen an bestimmte Aufgaben angepasste KNN, die Gewichte eines KNN werden dabei auch Parameter genannt. Da KNN häufig sehr viele trainierbare Parameter haben, wird hier auch von *Deep Learning*[26] gesprochen.[27] Dazu wird zuerst der Verlust über eine Verlustfunktion bestimmt, indem Beispieldaten in das KNN eingegeben werden und das resultierende Ergebnis mit dem Sollwert der Beispieldaten abgeglichen wird.[28] Daraufhin wird eine Backpropagation durchgeführt, es wird der berechnete Verlust auf die verschiedenen Gewichte verteilt und basierend darauf die Gewichte angepasst. Die Anpassung der Gewichte ähnelt dabei einem Gradientenabstieg, wes-

[20] Enthalten in: Rashid 2017, S. 30
[21] Vgl. auch im Folgenden Rashid 2017, S. 30f.
[22] Vgl. auch im Folgenden Rashid 2017, S.36 ff.
[23] Vgl. Nielsen 2015
[24] z. Dt. Fehlerrückführung
[25] Vgl. Rashid 2017, S. 59 ff.
[26] z. Dt. Tiefen-Lernen
[27] Vgl. Nielsen 2015
[28] Vgl. Rashid 2017, S. 14 ff.

halb die Probleme der zu langsamen Konvergenz zum Optimum und des Überschwingens, also dem Überspringen des Optimums, auftreten können. Daher wird die Stärke der Anpassung von einem Optimizer[29] Algorithmus bestimmt, welcher über eine Lernrate eingestellt werden kann und basierend darauf die Backpropagation durchführt. Somit bestimmt die Lernrate wie schnell das KNN lernt. Eine zu geringe Lernrate führt zu einem zu langsamen Lernprozess, wohingegen eine zu hohe Lernrate ältere Beispiele womöglich vergisst und das Optimum leichter überspringt. Ein häufig verwendeter Optimizer ist der Adam Optimizer, welcher „auf adaptiven Schätzungen von Momenten niedrigerer Ordnung basiert“[30].

Bei dem Training eines KNN werden diese Schritte für jedes Trainingsbeispiel in einem Datenset durchgeführt. Da die Backpropagation sehr rechenaufwendig ist, werden häufig mehrere Trainingsbeispiele zu einem Batch [31] zusammengefügt und zusammen verarbeitet.[32] Damit das Netz von den Beispielen mehrmals lernen kann wird es manchmal über mehrere Epochen trainiert, eine Epoche beschreibt eine Iteration über alle Trainingsbeispiele. Da Batchgröße, Lernrate, Epochenanzahl, Verlustfunktion und der Optimierer vor dem Training eingestellt werden müssen, werden diese Variablen auch Hyperparameter bezeichnet und haben eine große Auswirkung auf den Lernerfolg des KNN. So fluktuiert zum Beispiel die Genauigkeit eines Klassifizierungsmodells „von 32,2% bis 92,6% aufrund unterschiedlich gewählten Hyperparamtern“[33].

Beim Einsatz von Modellen basierend auf maschinellem Lernen wird zwischen vier Phasen unterschieden: Training, Validierung, Test und Inferenz. Während dem Training werden die Parameter des Modells durch maschinelles Lernen automatisch angepasst und auf die Trainings-Beispiele angepasst.[34] Nach dem Training wird in der Evaluierungs-Phase, manchmal auch Validierungs-Phase genannt, das Modell anhand der Validierungs-Beispiele bewertet und basierend darauf Hyperparameter für ein erneutes Training zu bestimmen. So können optimale Hyperparameter iterativ ausgewählt werden. Da nach der Auswahl der optimalen Hyperparameter das Modell sowohl an die Trainings-, als auch an die Validierungs-Beispiele angepasst ist, wird in der Testphase der Erfolg und die Effektivität des Modells gemessen. Dies kann anhand der Auswertung weiterer Test-Beispiele durch verschiedene statistische Metriken erfolgen, es können je nach Forschungsfrage jedoch auch andere Bemessungsmethoden verwendet werden. Die Inferenz-Phase, auch Produktions-Phase genannt, beschreibt lediglich den Einsatz des Modells in seiner schlussendlichen Anwendungsumgebung.

Die Knoten eines KNN werden häufig in Schichten angeordnet, wobei jede Schicht als Abstraktion der vorherigen Schicht angesehen werden kann.[35] Durch unterschiedliche Anordnung der Knoten und Schichten können verschiedene Architekturen entstehen, welche unterschiedliche Aufgaben unterschiedlich gut lösen können. Einige davon werden im Folgenden vorgestellt.

[29] z. Dt. Optimierer
[30] Kingma/Ba 2015, S. 1
[31] z. Dt. Stapel
[32] Vgl. auch im Folgenden Nielsen 2015
[33] Zhang u. a. 2019, S. 287
[34] Vgl. auch im Folgenden Shah 2020
[35] Vgl. Nielsen 2015

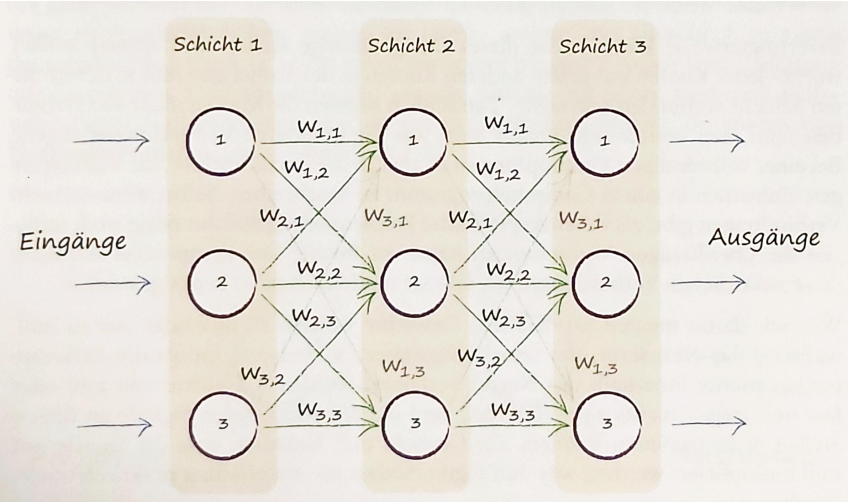

Abb. 3: Multi-Layer Feed-Forward Neural Networks (MLF)[36]

2.1.1 Feed Forward- und Convolutional Neural Networks

Multi-Layer Feed-Forward Neural Networks (MLF) [37] sind Zusammenschlüsse von künstlichen neuronalen Knoten angeordnet in mehreren Schichten, die Abbildung 3 zeigt ein solches Netz.[38] Dabei wird die erste Schicht *Eingangs-Schicht* und die letzte *Ausgangs-Schicht* genannt, die Schichten zwischen der Eingangs- und Ausgangs-Schicht heißen *Versteckte-Schichten*. Jede Schicht kann eine beliebige Anzahl an Knoten besitzen. Für jeden Eingabewert muss die Eingangs-Schicht jeweils einen Knoten aufweisen, für jeden gewünschten Ausgabe-Wert muss die Ausgabe-Schicht entsprechen jeweils einen Knoten besitzen. Grundlegend gilt, dass jeder Knoten einer Schicht mit allen Knoten der nächsten Schicht verbunden ist. Jede weitere Schicht nach der Eingangs-Schicht verarbeitet die Signale abstrakter als die vorherige.[39] Durch diesen Umstand kann eine größere Menge an Signalen, zum Beispiel die Pixel von Bildern, zu wenigen Signalen abstrahieren und zusammengefasst werden, um zum Beispiel Klassifizierungen wie *Hund oder Katze* durchzuführen. Daher werden MLF häufig dazu verwendet innerhalb einer größeren KNN-Architektur Signale eines Vektorraums in einen anderen Vektorraum zu konvertieren.

Um ein *Overfitting*[40] von MLF zu verhindern wurde die sogenannte *Dropout*[41]*-Methode* von Srivastava 2013 entwickelt. Dabei werden während des Trainings einzelne Knoten und ihre Gewichte versteckt, wodurch das Netz ohne sie rechnet und die Gewichte dieser Knoten nicht angepasst werden.[42] Die Dropout-Methode kann auch auf andere Arten von KNN angewandt werden.[43]

[36] Enthalten in: Rashid 2017, S. 37
[37] z. Dt. neuronale Multi-Schicht-Netze
[38] Vgl. auch im Folgenden Svozil/Kvasnicka/Pospichal 1997, S. 44 f.
[39] Vgl. Nielsen 2015
[40] z. Dt. Übermäßige Anpassung
[41] z. Dt. Signal-Löschung
[42] Vgl. Srivastava 2013, S. 3 ff.
[43] Vgl. Srivastava 2013, S. 2

Einen ähnlichen Aufbau wie MLF haben Convolutional Neural Network (CNN)[44], jedoch mit dem Unterschied, dass die Gewichte zwischen den Knoten geteilt werden.[45] Dies erspart nicht nur Speicheraufwand und Rechenzeit, sondern hilft bestimmte Merkmale der Eingangsdaten besser darzustellen. So können bei längeren Texten Wörter unabhängig von ihrer Position betrachtet werden und syntaktische Strukturen von Sätzen erlernt werden.[46] Daher werden CNN im NLP Bereich häufig zur Einbettung in Vektorräumen von Wörtern verwendet.[47]

2.1.2 Fortgeschrittene neuronale Netze

Im Gegensatz zu MLF und CNN bestehen Recurrent Neural Network (RNN)[48] nicht aus in verschiedenen Schichten angeordneten Knoten. Knoten eines RNN speichern ihre Signale und verarbeiten diese beim Eingang des nächsten Signals zusammen mit diesem weiter.[49] Eine wichtige Weiterentwicklung von RNN sind Long short-term memory (LSTM) [50] von Hochreiter/Schmidhuber 1997, welche sich mit Speicherzellen und Wächter-Einheiten den Kontext, also vergangene Eingangsdaten, eines Signals zu merken.[51] LSTM benutzen somit Zustände für die Verarbeitung von Signalen.

Embeddings from Language Models (ELMo)[52] ist ein im NLP Forschungsfeld erfolgreiches Sprachmodell von Peters u. a. 2018. ELMo konvertiert Token, also Wörter oder Wortteile, basierend auf ihrem Kontext durch zwei LSTM Schichten in eine Vektordarstellung.[53] So können nicht nur Buchstaben, sondern auch Wortteile und sogar ganze Wörter als Vektoren dargestellt und mit ihnen gerechnet werden.

2.1.3 Transformer Modelle

Transformer [54] sind komplexe Modelle basierend auf verschiedenen neuronalen Netzen und Methoden für maschinelles Lernen.[55] Sie basieren auf *Self-Attention* [56], einer Methodik, welche Werten einer beliebigen Sequenz eine Wichtigkeit zuweist.[57] Dies erwies sich vor allem bei der Übersetzung von Sätzen als sehr effektiv.[58]

[44] z. Dt. faltendes neuronales Netz
[45] Vgl. auch im Folgenden Yang/Li 2017, S. 229
[46] Vgl. Yu u. a. 2014, S. 4
[47] Vgl. Yang/Li 2017, S. 229
[48] z. Dt. Rekurrentes neuronales Netz
[49] Vgl. Hopfield 1982, S. 2555 f.
[50] überdauernder Kurz-Zeit Speicher
[51] Vgl. Hochreiter/Schmidhuber 1997, S. 1740 ff.
[52] z. Dt. Einbettungen von Sprachmodellen
[53] Vgl. Peters u. a. 2018, S. 2229
[54] z. Dt. Umwandler
[55] Vgl. Vaswani u. a. 2017, S. 6000
[56] z. Dt. Eigen-Aufmerksamkeit
[57] Vgl. Vaswani u. a. 2017, S. 6000 ff.
[58] Vgl. Vaswani u. a. 2017, S. 6005

Der Bidirectional Encoder Representations from Transformers (BERT)[59] Transformer von Devlin u. a. 2019 ist zurzeit das wohl beste Sprachmodell.[60] Im Gegensatz zu anderen Sprachmodellen nutzt BERT eine neue Trainingsmethode namens *Masked-Language-Model*[61], welche manche Token eines Satzes während dem Training maskiert und diese vom BERT anhand des Kontexts, also der anderen Token im Satz, erraten soll.[62] Dies erlaubt es dem Modell einen Satz als ganzes, also bidirektional, zu verarbeiten, was deutlich effektiver ist als es bei einer sequentiellen Verarbeitung wie bei zum Beispiel ELMo oder LSTM der Fall ist.

2.2 Entity-Linking als Aufgabe des NLP

Die Aufgabe des Entity-Linkings beschreibt die Herausforderung in einem Text Erwähnungen von Entitäten zu erkennen und diese genauer zu bestimmen, also mit Entitäten aus Wissensdatenbanken zu verlinken. Zwar ist in dieser Aufgabe auch die Erkennung dieser Erwähnungen, die sogenannte *Entity-Recognition*[63], enthalten, jedoch bezieht sich Entity-Linking meist nur auf die Verlinkung der Erwähnungen zu Entitäten, die sogenannte *Entity-Disambiguation*[64].[65]

Für eine Entity-Disambiguation muss die am besten geeignete Entität $e*$ aus der Wissensdatenbank KB von einer Erwähnung m basierend ihres Kontexts s verlinkt werden und kann mit $EL : (m, s, KB) \rightarrow e*$ als Funktion beschrieben werden. Da eine Wissensdatenbank eine endliche Menge an Entitäten besitzt, kann diese Aufgabe auch als ein Klassifizierungsproblem beschrieben werden, bei welchem Entitäten als Klassen angesehen werden und die Erwähnung mit einer dieser Klassen klassifiziert werden soll.[66] Wenn die Menge der Entitäten jedoch sehr groß ist, stellt sich dieser Ansatz als recht ineffizient heraus. „Der rationalisierte Ansatz für Entity-Linking besteht darin, es als ein Rangordnungsproblem zu behandeln."[67]

Um Entity-Linking Modelle zu bewerten werden die Ergebnisse mit einem Mikro F1-Score $F1$ gemessen:[68]

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{2.1}$$

$$Recall = \frac{TruePositive}{TruePositive + TrueNegative} \tag{2.2}$$

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.3}$$

[59] z. Dt. Bidirektionale Encoder-Repräsentation von Umwandlern
[60] Vgl. Horev 2018
[61] z. Dt. maskiertes Sprachmodell
[62] Vgl. auch im Folgenden Devlin u. a. 2019, S. 4174 f.
[63] z. Dt. Erkennung von Entitäten
[64] z. Dt. Entität Disambiguierung
[65] Vgl. Sevgili u. a. 2020, S. 3 f.
[66] Vgl. Sevgili u. a. 2020, S. 5 f.
[67] Sevgili u. a. 2020, S. 5
[68] Vgl. auch im Folgenden Sevgili u. a. 2020, S. 19

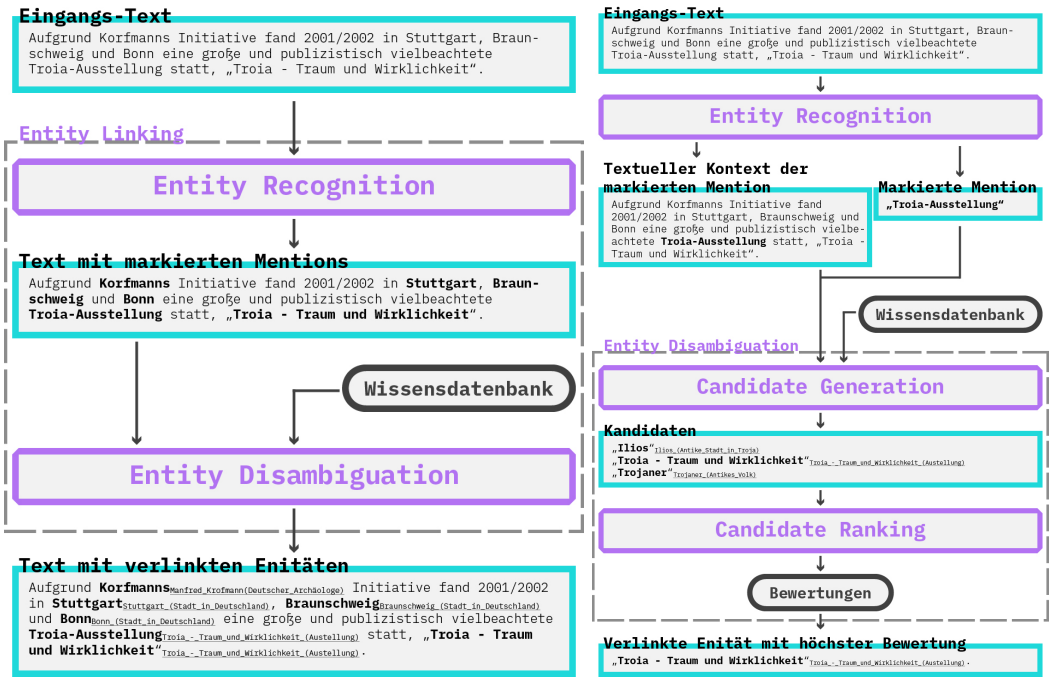

Abb. 4: Einordnung (links) und Ablauf anhand eines Beispiels (rechts) der Entity-Linking Aufgabe. (Entity Linking Einordnung)[70]

Wobei die Anzahl Richtig-Positiver $TruePositive$ gleich der Anzahl korrekt gefundener und verlinkter Erwähnungen ist, die Anzahl Falsch-Positiver $FalsePositive$ gleich der Anzahl fehlerhaft verlinkter Erwähnungen ist. Die Anzahl Positiver $Positive$ setzt sich aus $TruePositive + FalsePositive$ zusammen und repräsentiert die Anzahl vom Entity-Recognition Modell gefundenen Erwähnungen. Des Weiteren ist die Anzahl an Richtigen $True$ gleich der Anzahl an Erwähnungen der Grundwahrheit. Da bei der Entity-Disambiguation jedoch die Anzahl an Erwähnungen der Grundwahrheit und die Anzahl vom Entity-Recognition Modell gefundenen Erwähnungen gleich ist, also $True = Positive$, kann der Mikro F1-Score auch als Genauigkeit Acc angesehen werden:[69]

$$F1 = Acc = \frac{\#\text{ Korrekt verlinkte Erwähnungen}}{\#\text{ Erwähnungen der Grundwahrheit}} \tag{2.4}$$

Um bei dem Einsatz von trainierbaren Modellen in der Entity-Disambiguation Rechen- und Zeitaufwand zu optimieren, wird die Entity-Disambiguation häufig unterteilt in die zwei Schritte *Candidate-Generation*[71] und *Candidate-Ranking*[72], dargestellt in Figur 4.[73] Die Idee hinter der Aufteilung der Aufgabe ist die Reduzierung der möglichen Entitäten mithilfe eines schnellen Modells oder Algorithmus im ersten Schritt, um darauf im zweiten Schritt aus dieser kleineren Anzahl an Entitäten mithilfe eines aufwendigeren Modells die passendste Entität zu finden. Optional kann in einem dritten Schritt evaluiert werden, ob überhaupt eine passende Entität in

[69] Vgl. Shen/Wang/Han 2014, S. 456
[70] Mit Änderungen entnommen aus: Sevgili u. a. 2020, S. 3 f.
[71] z. Dt. Kandidatensuche
[72] z. Dt. Kandidatenbewertung
[73] Vgl. Sevgili u. a. 2020, S. 5 f.

der Wissensdatenbank existiert, die sogenannte Unlinkable-Mention-Prediction oder auch NIL-Prediction. Diese Zwei-Schritt-Architektur ist sehr üblich bei Entity-Disambiguation Systemen und aufgrund der größeren Komplexität des Candidate-Rankings fokussiert sich die Forschung primär auf diesen Schritt. [74]

2.2.1 Aufbau von Wissensdatenbanken

Die Aufgabe von Wissensdatenbanken ist Wissen strukturiert zu speichern. Um ein Entity-Linking Modell trainieren und evaluieren zu können benötigt es Beispiele welche aus Texten natürlicher Sprache und dazu verlinkten Entitäten aus einer solchen Wissensdatenbank bestehen. Eine Sammlung solcher Beispiele wird auch Datenset genannt, bei der Entity-Linking Aufgabe häufig genutzte Datensets sind unter anderem die AIDA-CoNLL-YAGO-2, Wikilinks, MSNBC und TAC-KBP-2010 Datensets. [75] Diese Datensets verlinken unterschiedlichste Textstücke zu unterschiedlichen Wissensdatenbanken. So basiert zum Beispiel das AIDA-CoNLL-YAGO-2, kurz AIDA Datenset, auf dem CoNLL 2003 Datenset und verlinkt Erwähnungen zur YAGO-2 Datenbank, welche wiederum ein Zusammenschluss der englischsprachigen Wikidata Datenbank und der Freebase Datenbank ist.[76] Somit verlinkt das AIDA Datenset auch zur Wikidata Datenbank. Das AIDA Datenset ist unterteilt in ein Trainingsset mit 17917 Beispielen und zwei Testtests TestA mit 4673 und TestB 4313, daher kann ein Testset für die Evaluierung eines Modells und das andere zum Testen eines Modells genutzt werden. Des Weiteren kommt es vor, dass bestehende Datensets nochmals mit weiteren Attributen aus der des Datensets entsprechenden Wissensdatenbank angereichert werden, um spezifische Daten für Ihre Lösung bereitzustellen.[77]

Ein Beispiel eines Domain-typisches Datenset für die Unteraufgabe der Candidate-Generation besitzt eine Erwähnung m, den textuellen Kontext c_{links} und c_{rechts}, sowie der Erwähnung entsprechenden Entität e. Dabei ergeben m, c_{links} und c_{rechts} die Textstelle s. Häufig ist der Kontext der Textstelle durch die Satzgrenzen abgegrenzt, sodass eine Textstelle häufig einen ganzen Satz darstellt. Dies ist jedoch nicht zwangsläufig der Fall. Existiert keine zu m gehörige Entität, so ist der Wert für e ein vordefiniertes Token wie zum Beispiel NIL. Beim Trainieren eines Candidate-Ranking Modells wird dem Beispiel eine Liste an Kandidaten C hinzugefügt, welche die Ergebnisse eines Candidate-Generators widerspiegeln sollten.

2.2.2 Unteraufgabe Candidate-Generation

Ziel der Candidate-Generation ist es eine kleine Anzahl an Entitäten aus einer Wissensdatenbank auszusuchen, worunter mit einer sehr hohen Wahrscheinlichkeit die passendste Entität zu einer

[74] Siehe exemplarisch Forschungsarbeiten von Onoe/Durrett 2020 S. 8579, Sevgili u. a. 2020 S. 4 ff. Shen/Wang/Han 2014 S. 445 f. und Vyas/Ballesteros 2021 S. 835 f.
[75] Siehe Arbeiten zu Datensets von Ellis/Getman/Strassel 2018, Hoffart 2013 S. 44, Hoffart u. a. 2011 S. 789 f., Singh u. a. 2012 und Cadez u. a. 2003
[76] Vgl. dazu ausführlich Hoffart u. a. 2011, S. 789
[77] Siehe Beispiel Onoe/Durrett 2020, S. 8579

Erwähnung ist. Formal ausgedrückt soll die Candidate-Generation für jede Erwähnung $m \in M$ eine Menge C Kandidaten-Entitäten generieren, bestehend aus k Entitäten $e_i \in KB$ mit $k \ll KB$.[78] Somit kann dieser Schritt als Funktion $CG : m \rightarrow C$ mit $C = \{e_1, e_2, ..., e_k\}$ beschrieben werden.

Einfache Methoden sind das sogenannte *Surface Form Matching*[79], *Alias Expansion*[80] sowie *Prior-Probability-Computation*[81].[82] Des Weiteren können Such-Maschinen wie Google-Search-Engine oder Wikipedia-Search genutzt werden, um Kandidaten von Wikipedia zu erhalten.[83] Dies kann jedoch nur genutzt werden, wenn die zu verlinkende Wissensdatenbank auf Wikipedia, bzw. Wikidata basiert, also ausschließlich Entitäten von Wikipedia besitzt. In den meisten Fällen erwies die Kombination mehrerer Methoden als besonders effektiv.[84]

Beim Surface Form Matching wird nach Ähnlichkeiten der benutzten Wörter von der Erwähnung und den Entitäten der Wissensdatenbank gesucht.[85] So wird ein Kandidatenset mit Entitäten erstellt, deren Orthografie ähnlich der Orthografie der Erwähnung ist. Dazu gibt es verschiedene Ansätze, welche entweder statistisch oder mithilfe von Machine Learning Modellen angewandt werden können. Ein Problem dieser Methode lässt sich anhand des Synonyms *Big Blue* für den Konzern *IBM* erklären. So könnte zum Beispiel bei der Erwähnung *Big Blue* ein Kandidatenset bestehend aus *Big_ Blue_ Sky* oder *Big_ Blue_ River* erstellt werden, jedoch ohne die korrekte Entität *IBM*. Dementsprechend ist diese Methode nicht für Synonyme oder Akronyme geeignet.

Dieses Problem wird durch die Alias Expansion gelöst. Bei dieser Methode wird für jedes mögliche Akro- und Synonym ein Eintrag in einem Wörterbuch angelegt, meist basierend auf den Metadaten der Wissensdatenbanken wie zum Beispiel die Weiterleitungsseiten von Wikipedia.[86] Solch ein Eintrag besteht aus einem Akro- bzw. Synonym-Wort Paar, wodurch auch zu Erwähnungen wie *Big Blue* die richtigen Kandidaten generiert werden können. Diese Methode kann erweitert werden zu einer *Name-Dictionary-Technique* [87] indem direkt die Kandidaten zu allen möglichen Erwähnungen in einem Wörterbuch gespeichert werden.[88] Dieses Wörterbuch lässt sich mithilfe von bestimmten Seiten und Attributen von Wikipedia generieren. Daher ist diese erweiterte Methode ähnlich wie die Suchmaschinen Methoden nur für auf Wikipedia basierender Wissensdatenbanken einsetzbar.

Auch die Prior-Probability-Computation basiert auf vorkalkulierten Annahmen und Metadaten aus der entsprechenden Wissensdatenbank.[89] Hierbei wird im Vorhinein für eine Vielzahl an

[78] Vgl. auch im Folgenden Sevgili u. a. 2020, S. 5 f. Vgl. Vyas/Ballesteros 2021, S. 836
[79] z. Dt. Oberflächenform Abgleich
[80] z. Dt. Erweiterung durch Aliase
[81] z. Dt. Wahrscheinlichkeitsberechnung
[82] Vgl. Sevgili u. a. 2020, S. 5 ff. Vgl. Shen/Wang/Han 2014, S. 446 ff.
[83] Siehe Beispiel Shen/Wang/Han 2014, S. 449
[84] Vgl. Sevgili u. a. 2020, S. 7
[85] Vgl. auch im Folgenden Sevgili u. a. 2020, S. 6; Vgl. dazu ausführlich Shen/Wang/Han 2014, S. 448
[86] Vgl. auch im Folgenden Sevgili u. a. 2020, S. 6
[87] z. Dt. Namenswörterbuch Technik
[88] Vgl. auch im Folgenden Shen/Wang/Han 2014, S. 447 f.
[89] Vgl. auch im Folgenden Sevgili u. a. 2020, S. 7

Erwähnung-Kandidaten Paaren gebildet, zum Beispiel basierend auf Wikipedia Hyperlinks oder der Häufigkeit auftretender Entitäten in einem Text.

2.2.3 Unteraufgabe Candidate-Ranking

Beim Candidate-Ranking ist das Ziel die Entitäten einer Kandidatenliste basierend auf einer Erwähnung zu bewerten und darauf basierend eine Entität aus dieser Kandidatenliste zur Erwähnung zu verlinken.[90] Formal soll aus einer k großen Kandidatenliste C bestehend aus Entitäten $e_i \in KB$ mit $k \ll KB$ eine Entität e^* zu einer Erwähnung m basierend auf dessen Kontexts s verlinkt werden. Dies lässt sich ebenfalls als Funktion $CR : (m, s, C) \rightarrow e^*$ mit $e* = \arg\max_{e_i \in C} score(e_i)$ darstellen.

Wichtig anzumerken ist, dass bei dieser Definition die beste Entität aus der gewerteten Kandidatenliste das Ergebnis der Funktion ist. In manchen Fällen wird das Candidate-Ranking als Funktion mit der gewerteten Kandidatenliste als Ergebnis definiert.[91]

Bei diesem Schritt können eine Vielfalt an Methoden und Architekturen gute Ergebnisse erzielen. Bei Methoden welche Machine Learning Modelle nutzen werden häufig Erwähnung und Entitäten separat encodiert, um daraufhin miteinander verglichen werden zu können.[92] Dazu werden entsprechend Erwähnungs- bzw. Entitätsencodierer verwendet, welche ebenfalls als eigene Zwischenschritte angesehen werden können.[93] Die Nutzung solcher Encodierer ist jedoch nicht zwangsweise nötig, so gibt es zum Beispiel Modelle basierend auf anderen Methoden.[94]

2.3 Entity-Typing als Aufgabe des NLP

Lösungsansätze des Entity-Typings Problems lassen sich auch auf die Entity-Linking Aufgabe übertragen, wie zum Beispiel bei dem Lösungsansatz von Onoe/Durrett 2020. Entity-Typing beschreibt die Aufgabe zu einer Erwähnung m, ähnlich wie beim Entity-Linking, innerhalb eines Kontexts s Typen oder Kategorien T aus einem sogenannten *Typeset*[95] D zuzuordnen. Entsprechend kann also das Entity-Typing als Funktion $ET : (m, s, D) \rightarrow T$ formuliert werden. Resultierende Typen können als abstrakte Darstellung der Erwähnung angesehen werden. Bewertet werden Entity-Typing Modelle meist mit einem Makro-F1 Score, welcher sich wie in Gleichung 2.3 beschrieben aus der Spezifität und der Sensitivität ergibt.[96] Als $TruePostive$ erweisen sich alle

[90] Vgl. auch im Folgenden Sevgili u. a. 2020, S. 10; Vgl. auch im Folgenden Shen/Wang/Han 2014, S. 445; Vgl. auch im Folgenden Vyas/Ballesteros 2021, S. 836
[91] Vgl. Sevgili u. a. 2020, S. 10
[92] Vgl. Sevgili u. a. 2020, S. 10
[93] Vgl. Sevgili u. a. 2020, S. 7 ff.
[94] Vgl. Onoe/Durrett 2020, S. 8578 f. Vgl. Vyas/Ballesteros 2021, S. 835 f.
[95] z. Dt. Kategorie-Wörterbuch
[96] Vgl. Onoe/Durrett 2020, S. 8580; Vgl. Lin/Ji 2019, S. 6199

richtig vorhergesagten Kategorien, als $FalsePositive$ alle fälschlicherweise vorhergesagten Kategorien, als $TrueNegative$ alle richtig nicht vorhergesagten Kategorien und als $FalseNegative$ alle fälschlicherweise nicht vorhergesagten Kategorien.

2.4 Analyse existierender Entity-Linking Systeme

Eine Übersichtsstudie von Sevgili u. a. 2020 für die Entity-Linking Aufgabe verglich verschiedene Systeme und hob besonders effektive Systeme hervor. Ein überdurchschnittlich performendes System ist ein auf kontinuierlicher Sequenz-Bearbeitung basierendes System von Cao u. a. 2021, welches eine Genauigkeit von 93,3 % auf das AIDA-TestB Datenset und sogar 94,4 % auf das MSNBC Datenset aufweist.[97] Das System mit der besten Genauigkeit auf das AIDA-TestB Datenset wurde von Yamada u. a. 2019 entwickelt und erzielt 95 %.[98] Die Modelle dieser Systeme sind auf die Wissensdatenbank des Testsets trainiert und erzielen somit bessere Resultate als Systeme, wessen Modelle auf andere Wissensdatenbanken trainiert sind. Solch ein Domänen-übergreifendes System ist zum Beispiel der von Onoe/Durrett 2020 entwickelte Entity-Linker welcher Kategorien mit einem Entity-Typer erzeugt und diese mit den Kategorien einer Entität vergleicht.[99] Dieser Entity Linker weist eine Genauigkeit von 62,2 % auf das Wikilinks Test-Datenset auf, wurde jedoch mit einem Datenset basierend auf Wikipedia trainiert.[100] Ein weiteres Beispiel ist ein auf Attributen basierter Entity-Linker von Vyas/Ballesteros 2021 welcher eine Genauigkeit von 61,6 % auf das TAC-KBP-2010 Test-Datenset erzielt, ohne speziell auf dieses trainiert zu sein.[101] Die letzten beiden Entity-Linker werden aufgrund ihrer Unabhängigkeit zu einer spezifischen Wissensdatenbank im folgenden genauer beschrieben und analysiert.

2.4.1 Abstraktion durch Kategorien

Ein entscheidender Vorteil des von Onoe/Durrett 2020 entwickelten Entity-Linkers, „ET4EL“ genannt, ist die große Generalisierung der Eigenschaften von Erwähnungen, welche durch das vom Typeset D abhängiges Training erzeugt wird.[102] Somit kann dieses Modell auf alle Wissensdatenbanken mit gleichem Typeset linken.

Der Candidate-Ranker des Entity-Typing for Entity-Linking (ET4EL) benutzt einen Entity-Typer, um Kandidaten zu bewerten.[103] Das Typeset des Entity-Typers wird vor dem Training erzeugt und gespeichert, da das Netzwerk mit diesem trainiert wird und sich daher darauf anpasst. Ebenso wird angenommen, dass jede Entität der Wissensdatenbank mindestens einer Kategorie aus dem Typeset zuzuordnen ist. Um dies zu erreichen, kann zum Beispiel der Entity-Typer im

[97] Vgl. Cao u. a. 2021, S. 6
[98] Vgl. Yamada u. a. 2019, S. 3
[99] Vgl. Onoe/Durrett 2020, S. 8577 ff.
[100] Vgl. Onoe/Durrett 2020, S. 8581
[101] Vgl. Vyas/Ballesteros 2021, S. 8
[102] Vgl. Onoe/Durrett 2020, S. 8576
[103] Vgl. auch im Folgenden Onoe/Durrett 2020, S. 8576

Vorhinein jeder Entität der Wissensdatenbank Kategorien zuordnen. Eine andere Möglichkeit wäre diese Kategorien manuell zuzuordnen.

Der Entity-Typer encodiert die Erwähnung und seinen Kontext mit einem ELMo Netzwerk, um die Wörter in einem kontextualisiertem Vektorraum abzubilden.[104] Die resultierenden Wort-Encodierungen werden darauf mit einem bidirektionalem LSTM Netzwerk verarbeitet und in ein MLF gegeben. Dabei verändert das MLF die Vektorform der Encodierungen von der Form des LSTM Netzwerks auf die Größe des Kategorie-Wörterbuchs. Eine Sigmoid Funktion dient hierbei als Aktivierungsfunktion. Der resultierende Vektor bildet den Score für jede Kategorie des Wörterbuchs, geschlüsselt nach den Indexen. In einem zweiten Schritt werden die Scores der Kategorien einer Kandidaten-Entität aufsummiert, woraus sich ein Score für jeden Kandidaten ergibt.[105] Der Kandidat mit dem höchsten Score wird mit der Erwähnung verlinkt.

Nachteil dieses Verfahrens ist der entstehende Bias von überrepräsentierten Kategorien, welcher durch die bloße Aufsummierung der Scores der Kategorien entsteht. Die Forscher schrieben hierzu: „Wir haben festgestellt, dass die einfache Summierung der Ergebnisse besser abschneidet als andere Optionen wie die Mittelwertbildung oder die Berechnung eines logarithmischen Quotienten. Intuitiv profitiert die Summierung von Kandidaten mit vielen Kategorien, was das Modell in vorteilhafter Weise auf häufigere Entitäten ausrichtet. Es belohnt auch Modelle mit vielen korrelierten Typen, was problematisch ist, aber die Ansätze, die wir ausprobiert haben, die die Typkorrelation auf eine prinzipiellere Art und Weise behandeln, haben nicht besser abgeschnitten.“[106] Des Weiteren werden keine Ansätze für das Lösen des NIL-Prediction spezifiziert.

Üblicherweise wird zum Training und zur Evaluierung eines Candidate-Rankers ein Datenset, wessen Beispiele aus Erwähnung, Kontext, Kandidatenliste sowie einer richtigen Entität bestehen, genutzt. Jedoch wird lediglich der Entity-Typer innerhalb des Candidate-Rankers trainiert, weshalb für das Training ein Entity-Typing Datenset benutzt wird. Beispiele aus solch einem Datenset bestehen aus Erwähnung, Kontext und einem Typeset: $D_{Typer} = \{(m, s, T)^{(1)}, ..., (m, s, T)^{(l)}\}$. Für das Training des ET4EL wurde ein von den Forschern selbst zusammengestelltes Datenset basierend auf Wikipedia genutzt. Dabei wurden Hyperlinks des englischsprachigen Wikipedias genutzt, die Kategorien einzelner Entitäten wurden ebenfalls aus Wikipedia entnommen. Zur Evaluierung wurde das AIDA TestB Datenset sowie das Wikilinks Datenset genutzt. Das AIDA Datenset verlinkt Erwähnungen mit Entitäten aus Wikipedia, im Gegensatz zum Wikilinks, welches Entitäten zur Wikia Wissensdatenbank verlinkt und somit eine *ungesehene* Wissensdatenbank darstellt. Das Modell kann eine Genauigkeit von 85,9 % auf das Testdatenset AIDA TestB aufweisen, sowie eine Genauigkeit von 62,2 % auf das Wikilinks Testdatenset, bei einem Kandidaten Recall von 88 %. Des Weiteren konnte gezeigt werden, dass eine erhöhte Anzahl an Kategorien im Typeset zu einer höheren Genauigkeit führen. Eine weitere Erhöhung der Genauigkeit kann durch die Zunahme weiteren Kontexts, welcher über einen Satz hinausgeht, erzielt werden. Im Allgemeinen konnte gezeigt werden, dass die Abstraktion durch Kategorien eine

[104] Vgl. auch im Folgenden Onoe/Durrett 2019, S. 2410 f.
[105] Vgl. auch im Folgenden Onoe/Durrett 2020, S. 8576
[106] Onoe/Durrett 2020, S. 8578

wirksame Methode zum Verlinken von textlichen Erwähnungen zu Entitäten in einer Datenbank ist.

2.4.2 Attribut-Schemata für Verlinkungen zu unterschiedlichen Wissensdatenbanken

Ein ganz entscheidender Nachteil des ET4EL Entity-Linkers ist die Abhängigkeit von Typen oder Kategorien einer Wissensdatenbank. Die zu verknüpfende Wissensdatenbank des ET4EL ein Typeset aufweisen und jeder Entität entsprechende Kategorien zuweisen. Anders ist dies bei einem Entity-Linker, welcher unabhängig von Wissensdatenbank Schemata genutzt werden kann, dieser hätte dadurch den Vorteil einer sehr großen Generalisierbarkeit auf unterschiedliche Wissensdatenbanken.

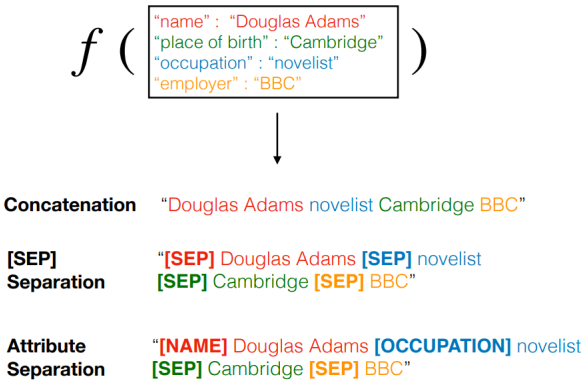

Abb. 5: Verschiedene Methoden der textlichen Darstellung von Attributen aus einer beliebigen Wissensdatenbank. (Attribut Schema)[107]

Ein Beispiel dafür ist ein auf dem BERT Transformer basierender Entity-Linker von Vyas/Ballesteros 2021, welcher Attribute aus der Wissensdatenbank nutzt. Das Forschungsteam benutzte dabei eine Methode namens *Attribute-Separation*[108], um die unterschiedlichsten Wissensdatenbank-Schemata textlich darzustellen, wie die Abbildung 5 zeigt. Dabei werden einzelne Attribute einer Entität zusammengefügt, teilweise mit speziellen Separierungen. Diese Separierungen werden von BERT in besonderer Form behandelt und dienen dem Transformer als wichtige Informationen zum Unterscheiden und Gewichten verschiedener Textattribute.

Der Candidate-Generator besitzt zwei unabhängige BERT Transformer wovon einer die Erwähnung zusammen mit ihrem Kontext und einer die Entität encodiert. Als Encodierung wird dabei jeweils die gebündelte Vektordarstellung des Textes aus der Vektordarstellung des $[CLS]$ Tokens im letzten Layer genutzt, die Encodierung des BERT Transformers wird somit reduziert. Der encodierten Erwähnung wird daraufhin noch ein weiterer Vektor hinzugefügt, welcher die Stelle

[107] Enthalten in: Vyas/Ballesteros 2021, S. 3
[108] z. Dt. Attribut Separierung

der Erwähnung im Kontext markiert. Beide Encodierungen werden daraufhin mittels eines Vektorvergleiches verglichen. Die N der Erwähnung ähnlichsten Entitäten werden als Kandidaten ausgewählt.

Der Fokus des Forschungsteams lag jedoch nicht auf dem Candidate-Generator, sondern auf dem Candidate-Ranker. Dieser folgt einem ähnlichen Aufbau wie der Candidate-Generator, jedoch mit einem anstatt zwei BERT Modellen. Hierzu wird die Erwähnung, der Kontext und die Entität mit ihren Attributen zusammen im BERT Transformer encodiert und die Kodierungen wie beim Candidate-Generator reduziert. Dabei werden die Attribute jeder Entität mit der Attribute-Separation Methode textlich dargestellt, wie in Abbildung 5 zu sehen ist. Die reduzierten Encodierungen werden darauf in mit einem künstlichem, linearem, neuronalem Netzwerk gewichtet und zusammengefasst. Das daraus resultierende Ergebnis stellt den Score des Kandidaten dar. Der Kandidat mit dem höchsten Score aus der Kandidatenliste wird zur Erwähnung verlinkt.

Da der Fokus der Forschung auf der Attribute-Separation und somit auf dem Candidate-Ranker lag, ist keine NIL-Prediction beschrieben. Der Ansatz der Attribute-Separation könnte noch verbessert werden, dazu schrieben die Forscher: „Eine Möglichkeit, diese Lücke zu schließen, ist der Einsatz automatischer Techniken zur Umwandlung von Tabellen in Text, um beliebige Entitäten in flüssigen und adäquaten Text umzuwandeln. Eine andere vielversprechende Richtung ist, über BERT hinaus zu anderen vortrainierten Repräsentationen überzugehen, von denen bekannt ist, dass sie Entitätsinformationen bedder encodieren.“[109]

Trainiert werden der Candidate-Generator und der Candidate-Ranker mit dem AIDA Datenset, das TAC-KBP-2010 Datenset wird zur Evaluierung beider Systeme genutzt. Nach dem Training erreichte der Candidate-Generator einen Recall@32 von 91,25 %. Die darauf aufbauende Genauigkeit des Candidate-Rankers betrug vor dem weiteren Training (Finetuning) auf das Testset 61,6 %, danach 84,9 %. Es stellte sich heraus, dass die Attribut-Separierungen eine deutliche Steigerung der Genauigkeit des Systems verursachen können.

[109] Vyas/Ballesteros 2021, S. 8

3 Auswahl und Begründung der Forschungsmethodik

Diese Arbeit beschäftigt sich mit der Forschungsfrage, ob die Konzepte aus den Kapiteln 2.4.1 (Abstraktion durch Kategorien) und 2.4.2 (Attribut-Schemata für Verlinkungen zu unterschiedlichen Wissensdatenbanken) zusammen harmonieren und ein Modell ergeben, welches die Entity-Linking Aufgabe besser löst. Besser ausgedrückt: *Harmonieren die beiden Methoden (1) der Abstraktion von Erwähnungen durch Kategorien und (2) der Abstraktion von Entitäten aus Wissensdatenbanken durch Attribute?* Diese Arbeit nutzt dazu die DSR Methode, da diese Methodik der Entwicklung von auf KNN basierenden Modellen eine uniformierte und nachvollziehbare Struktur gibt. Bei der DSR Methode wird Wissen in Form eines Artefakts zu erzeugt, um verschiedene Arten und Methoden der Wissens-Erzeugung in einem Konzept zusammenzufassen.[110] Um trotz dieser Zusammenfassung der Methoden eine genaue Darstellung des erzeugten Wissens darbieten zu können, unterscheiden Gregor/Hevner 2013 zwischen verschiedenen Arten von Artefakten. Eine davon ist das Modell, welches eine funktionierende Abbildung oder auch Abstraktion einer Theorie, also des neu erzeugten Wissens, ist.[111] Somit soll ein Modell vergangene Zustände erklären oder Zustände in der Zukunft vorhersagen können. Ziel dieser Arbeit ist es somit ein Modell zu entwickeln, welches die Aufgabe des Entity-Linkings bewältigen kann, um somit Einblicke in die Zusammenhänge zwischen textlichen Darstellungen einer Entität und dieser verstehen kann. Es handelt sich nach Gregor/Hevner 2013 um eine Verbesserung, also das Ziel „bessere Lösungen in Form von effizienteren und effektiveren Produkten, Prozessen, Serviceleistungen, Technologien oder Ideen zu kreieren“[112].

Die Entwicklung des Artefakts geschieht in dieser Arbeit iterativ, es werden entsprechend strukturiert durch Laborexperimente Parameter des Artefakts in jeder Iteration angepasst, um das bestmögliche Artefakt zu entwickeln. Ziel des Laborexperiments ist *Ursache-Wirkungs-Beziehungen zwischen Variablen zu analysieren*[113]. Diese Analyse soll genutzt werden, um die Parameter des Artefakts anzupassen.

Um das Artefakt evaluieren zu können, also ein Laborexperiment durchführen zu können, benötigt es eine strukturierte Methode. Das in dieser Arbeit durchgeführte Laborexperiment basiert auf der Annahme, dass das Artefakt ein Modell basierend auf KNN ist und somit mithilfe von maschinellem Lernen trainiert werden kann. Entsprechend kann nicht bestimmt werden, ob das Modell die Aufgabe vollständig oder gar nicht lösen kann, sondern eher wie gut das Modell die Aufgabe lösen kann. Um dies zu bestimmen, soll das Modell die Entity-Linking Aufgabe für eine Reihe an Beispielen in einem Evaluierungsdatenset lösen, der Erfolg des Modells wird, wie in Gleichung 2.4 beschrieben, mit der F1-Genauigkeit bestimmt. Daher ist der Ablauf eines Experiments wie folgt: (1) Training des Modells, (2) Evaluierung des Modells und (3) Anpassung Parameter anhand der Evaluierungsergebnissen.

[110] Vgl. Gregor/Hevner 2013, S. 339
[111] Vgl. Gregor/Hevner 2013, S. 342
[112] Gregor/Hevner 2013, S. 346
[113] Eschweiler/Evanschitzky/Woisetschläger 2009, S. 363

4 ACCEL Entwicklung und Architektur

In einem neuen Konzept für einen Entity-Linker wurden die Konzepte des Entity-Typings und der Attribut-Separierung kombiniert, um die Vorteile beider Systeme nutzen zu können. Dieser neue Entity Linker wird im Folgenden Abstract Context through Categories for Entity-Linking (ACCEL)[114] genannt und ist eine Kombination der beiden Systeme aus den Kapiteln 2.4.1 (Abstraktion durch Kategorien) und 2.4.2 (Attribut-Schemata für Verlinkungen zu unterschiedlichen Wissensdatenbanken).

Im Folgenden wird lediglich die Aufgabe des Candidate-Rankings betrachtet. Es wird also davon ausgegangen, dass aus allen Entitäten einer Wissensdatenbank KB eine kleine Anzahl $k \ll KB$ an potenziellen Kandidaten $C \in KB$ für eine Erwähnung m durch einen Candidate-Generator ausgewählt wurde. Ebenso wird nicht näher auf die NIL-Prediction eingegangen.

Das Konzept hinter ACCEL ist die Abstrahierung des Kontexts durch Zuweisung von Kategorien aus einem fest definiertem Typeset. Hierzu wird mit einem Entity-Typer die Erwähnung basierend auf ihrem Kontext abstrahiert, indem der Entity-Typer zum Kontext passende Kategorien aus einem vordefiniertem Typeset auswählt. Diese Kategorien werden daraufhin in einem Modell genutzt, um die Erwähnung abstrakt darzustellen und sie leichter mit Entitäten aus einer beliebigen Wissensdatenbank vergleichen zu können. Alle Modelle sind mit der Python Bibliothek *PyTorch-Lightning* von Falcon u. a. 2019 implementiert, um maximale Reproduzierbarkeit zu gewährleisten. Die Implementierung ist im Anhang 2 und 3 zu finden.

4.1 Architektur des ACCEL Entity-Linkers

Der Entity-Typer von Onoe & Durrett aus Kapitel 2.4.1 wurde in veränderter Form in einem neuartigen Candidate-Ranker mit der Attribute-Separation Methode aus Kapitel 2.4.2 verbunden, um den passendsten Kandidaten für eine Erwähnung zu finden.

4.1.1 Aufbau des Entity-Typers

Dieser Entity-Typer basiert auf dem Entity-Typer des ET4EL aus Kapitel 2.4.1, funktioniert jedoch mit einem Encodierer basierend auf dem BERT Transformer anstelle eines ELMo Netzwerks. Diese Änderung an dem Entity-Typer ist notwendig, um eine plattformübergreifende Lösung zu bieten, da die benutzte ELMo Implementierung zum Zeitpunkt dieser Arbeit veraltet und teilweise nicht mehr nutzbar ist. Es handelt sich bei diesem Entity-Typer ein trainierbares Modell basierend auf KNN, welches entsprechend mit Methoden des maschinellen Lernens, bzw. Deep Learnings, trainiert wird.

[114] z. Dt. Kontext-Abstraktion durch Kategorien beim Entity-Linking

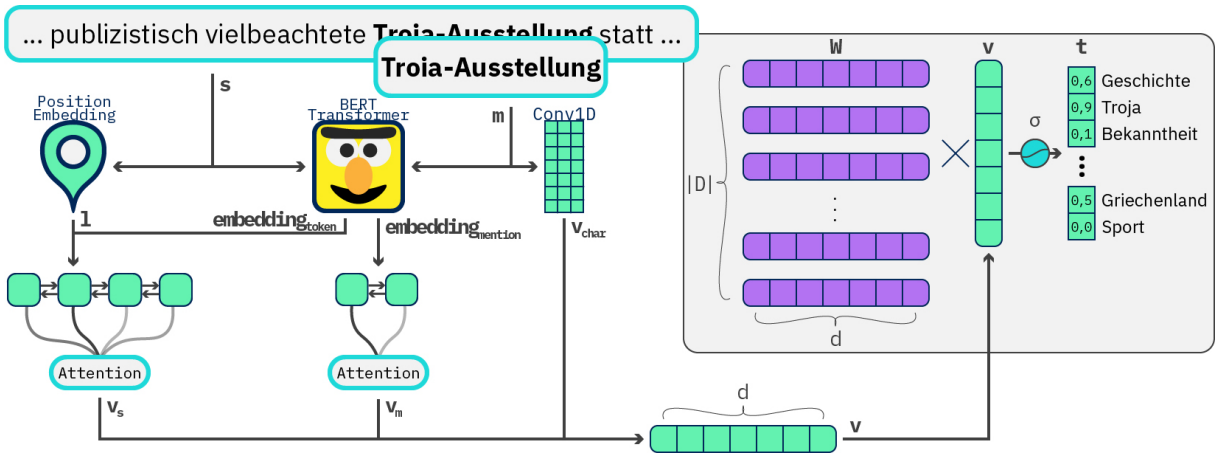

Abb. 6: Der Entity-Typer encodiert die Erwähnung zusammen mit ihrem Kontext und erzeugt durch ein Gewicht W für alle Kategorien des Typesets D einen Score t. In diesem Beispiel der Erwähnung *Troia-Ausstellung* wird unter anderem der Kategorie *Troja* ein Score von 0,9 zugewiesen. (Architektur Entity-Typer)[115]

Um die in Kapitel 2.3 (Entity-Typing als Aufgabe des NLP) beschriebene Problematik des Entity-Typings zu vereinfachen, wird von Unabhängigkeit zwischen den Kategorien ausgegangen. Eine Abhängigkeit zwischen den Kategorien wäre zum Beispiel, dass bei einer Zuweisung der Kategorie *Hafenstadt* ebenfalls der Oberbegriff *Stadt* mit zugewiesen werden muss. Durch die Unabhängigkeit der Kategorien lässt sich das Entity-Typing Problem auf ein binäres Klassifizierungsproblem reduzieren, wenn jeder Kategorie aus dem Typeset entweder *zur Erwähnung passend* oder *zur Erwähnung unpassend* zugewiesen wird. Bei einem Typeset D der Größe $|D|$ ist also das Ziel einen Vektor mit der Größe $|D|$, bestehend aus Einsen und Nullen, zu berechnen, wobei jeder Skalar des Vektors entweder den Zustand (1) *zur Erwähnung passend* oder (0) *zur Erwähnung unpassend* einer Kategorie darstellt.

Wie in Abbildung 6 zu erkennen ist, wird dazu die Erwähnung m zusammen mit ihrem Kontext s in einem BERT *Tokenizer*[116] in eine *Token*[117]*-Sequenz* $token_m = [CLS]\ c_{links}\ m\ c_{rechts}\ [SEP]$ verwandelt. Dabei sind $[CLS]$ und $[SEP]$ BERT-spezifische Token auf die der vortrainierte BERT Transformer trainiert wurde, wobei $[CLS]$ den Anfang und $[SEP]$ das Ende einer Sequenz angibt.[118] Die Token Sequenz $token_m$ wird daraufhin vom BERT Transformer encodiert und durch das sogenannte *Pooler-Output*[119] des Transformers zur Token-Einbettung $embedding_{token} = pooler(BERT(token_m))$ reduziert.[120] Diese Token-Einbettung ist eine Vektordarstellung der Token. Die Erwähnung-Einbettung $embedding_{mention} = embedding_{token}[i_{m-start} : i_{m-ende}]$ wird aus der Token-Einbettung entnommen. Die Token werden in einer Positions-Einbettung l Positionen zugeordnet: (1) Ob das Token vor der Erwähnung ist, (2) das erste Token der Erwähnung ist, (3) ein anderes Token der Erwähnung ist oder (4) hinter der Erwähnung ist.

[115] Mit Änderungen entnommen aus: Onoe/Durrett 2020, S. 8578
[116] z. Dt. Tokenisator
[117] Wortteil
[118] Vgl. Devlin u. a. 2019, S.4175 f.
[119] z. Dt. Pooler-Ausgabe
[120] Vgl. Devlin u. a. 2019, S.4174

Die Token-Einbettung wird zusammen mit der Positions-Einbettung in einem bidirektionalem LSTM Network gefolgt von einem Attention-Netzwerk gegeben und zur Satz-Darstellung $v_s = Attention(biLSTM(embedding_{token}; l))$ verarbeitet. Gleichzeitig wird die Vektordarstellung der Erwähnung $v_m = Attention(biLSTM(embedding_{mention}))$ auf gleiche Weise aus der Erwähnung-Einbettung berechnet. Ebenso werden die einzelnen Zeichen der Erwähnung in einem 1D CNN zu einer weiteren Vektordarstellung der Erwähnung $v_{char} = Conv(m_{char})$ verarbeitet. Die drei Vektordarstellungen v_s, v_m und v_{char} werden zusammengeführt und ergeben die finale Vektordarstellung $v = [v_s; v_m; v_{char}] \in \mathbb{R}^d$ der Erwähnung, bei der d die sich aus den Einzel-Komponenten ergebende Dimension ist.

In einem letzten Schritt wird Vektor-Repräsentation v der Erwähnung durch ein MLF mit dem Gewicht $W \in \mathbb{R}^{|D| \times d}$ transformiert, um einen decodierten Vektor der Größe $|D|$ zu erhalten. Da der resultierende decodierte Vektor nun die gleiche Größe wie das Typeset aufweist und es sich hierbei um ein binäres Klassifizierungsproblem handelt, entspricht jeder Skalar des Vektors genau einem Score einer Kategorie. Die Scores werden in einer Sigmoid Funktion angeglichen, wodurch die Scores als relativen Eignungsgrad angesehen werden können. Ist dieser Score größer als ein vordefinierter Grenzwert *threshold*, z. B. $threshold = 0.5$ (50 %), gilt diese Kategorie als (1) *zur Erwähnung passend*, sowie (0) *zur Erwähnung unpassend* wenn der Score kleiner als dieser vordefinierte Grenzwert ist. Ebenso wird die Kategorie mit dem höchsten Score als *zur Erwähnung passend* angesehen, sodass der Entity-Typer einer Erwähnung mindestens eine Kategorie zuweist, selbst wenn alle Scores kleiner als der Grenzwert sind. Somit ist das finale Resultat des Entity-Typers:

$$
\begin{aligned}
t &= \sigma(W \times v) \\
T &= D[(t > threshold) \cup argmax(t)] \\
\Phi &: (m, s) \rightarrow T
\end{aligned}
$$

Da in dem letzten Schritt die Anzahl und die Reihenfolge der Kategorien des Typesets D entscheidend für die schlussendliche Zuordnung der Kategorien zur Erwähnung ist, muss das Typeset vor dem Training des Modells definiert werden und auch während der Anwendung des Entity-Typers weiterhin genutzt werden. Bei einer Änderung des Typesets muss wiederholt trainiert werden.

4.1.2 Architektur des Candidate-Rankers

Bei diesem Candidate-Ranker wird wie in Abbildung 7 beschrieben einer Erwähnung m Kategorien zugewiesen, welche daraufhin mit der Attribute-Separation Methode aus Kapitel 2.4.2 zusammengefügt werden. Angenommen eine Entität e besitzt eine Anzahl j an Attributen A, so wird diese Entität nach Ausführung der Methode dementsprechend durch $token_e = [CLS]$ e $[ATT]$ a_1 $[ATT]$ a_2 $[ATT]$ ... a_j $[SEP]$ dargestellt. Da die Erwähnung m als nicht identifizierte

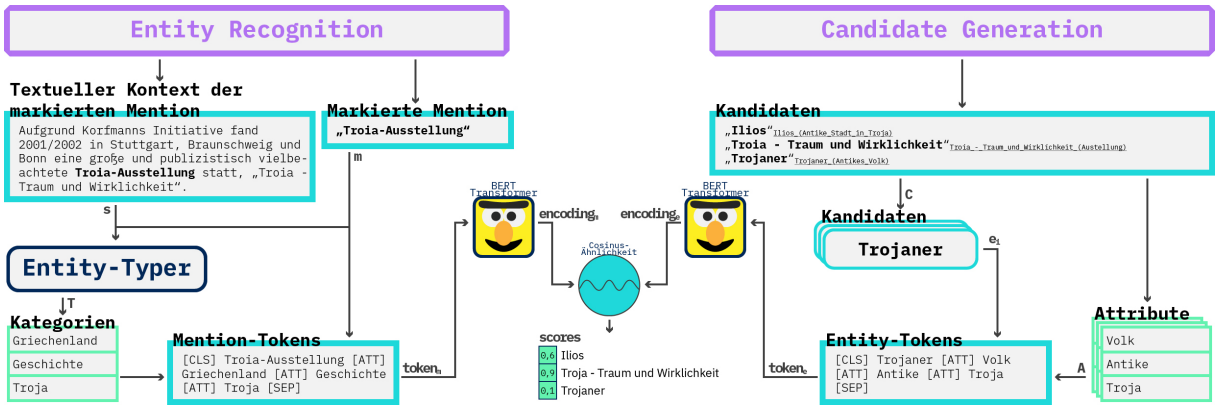

Abb. 7: Der Candidate-Ranker abstrahiert die Erwähnung und die Kandidaten, um sie mit zwei BERT Transformer zu encodieren und zu vergleichen. So wird in diesem Beispiel der Erwähnung *Troia-Ausstellung* die Entität *Troja* mit einem Score von 0,9 zugewiesen. (Architektur Candidate Ranker)

Entität gilt und die aus dem Entity-Typer zugewiesenen Kategorien $T = \{t_1, t_2, ..., t_i\}$ als Attribute dieser Erwähnung angesehen werden können, kann die Erwähnung dementsprechend durch diese Methode neu dargestellt werden. Die neue Darstellung der Erwähnung m ist somit:

$$token_m = [CLS]\ m\ [ATT]\ t_1\ [ATT]\ t_2\ [ATT]\ ...\ t_i\ [SEP] \tag{4.1}$$

Die entstandene Token-Sequenz $token_m$ wird daraufhin in einem vor-trainiertem BERT Transformer verarbeitet und encodiert. Ähnlich wie beim Modell aus Kapitel 2.4.2 wird die Vektorcodierung der Erwähnung $encoding_m = pooler(BERT(token_m))$ aus dem Pooler-Output des BERT Transformer gewonnen.

Das Vorgehen zum Encodieren eines Kandidaten ist ähnlich. Zuerst wird ein Kandidat e_i zusammen mit seinen Attributen A durch die Attribute-Separation Methode verarbeitet und als Token-Sequenz dargestellt:

$$token_e = [CLS]\ e\ [ATT]\ a_1\ [ATT]\ a_2\ [ATT]\ ...\ a_j\ [SEP] \tag{4.2}$$

Auch diese Kandidat-Token-Sequenz wird mit einem BERT Transformer verarbeitet und encodiert, wodurch die Vektor-Darstellung $encoding_e = red(BERT(token_e))$ des Kandidaten e_i entsteht.

Die beiden Vektor-Darstellungen werden daraufhin auf ihre Kosinus-Ähnlichkeit untersucht, der daraus entstehende Wert repräsentiert den Score $score_{e_i} = cos(encoding_m, encoding_e)$ des Kandidaten. Als Letztes wird die Erwähnung wird im letzten Schritt mit dem Kandidaten welcher den höchsten Score aufweist verlinkt.

4.1.3 Aufbau des Experiments zur Evaluierung des Entity-Typers und Candidate-Rankers

Da der Entity-Typer auf dem ET4EL aus Kapitel 2.4.1 basiert, basiert das Training des Entity-Typers ebenfalls auf dem Training des ET4EL, somit wurde auch das gleiche Datenset für das Training genutzt. Dieses Datenset wurde aus Wikipedia generiert und enthält mehr als 6 Millionen Beispiele.[121] Für das Training wurde das Datenset $D_{Wiki} = \{(m, s, T)^{(1)}, ..., (m, s, T)^{(l)}\}$ in ein Trainingsset $D_{ET-Train}$, welches 99,9 % der Beispiele enthält, und ein Evaluierungsset DET_Eval, welches die rechtlichen 0,1 % der Beispiele enthält, geteilt. Das genutzte Typeset D wurde aus diesem Datenset D_{Wiki} generiert, dazu wurde die Anzahl aller auftretenden Kategorien gezählt und die 60.000 häufigst-auftretenden Kategorien dem Typeset hinzugefügt. Dementsprechend ist die Anzahl an möglichen Kategorien auf $|D| = 60.000$ begrenzt. Der Grenzwert $threshold$ wurde auf $threshold = 0.5$ gesetzt, es wurden somit alle Kategorien welche einen höheren Score als 0,5 aufweisen als *passend* markiert.

Der Entity-Typer wurde mit einer Lernrate von 2e−3 in 32 Beispiel-großen Batches trainiert. Dabei wurden die Parameter des Modells durch einen Adam Algorithmus basierend auf einer *Binary Cross Entropy*[122] Verlustfunktion backpropagiert. Alle 100.000 Beispiele wurde das Modell mit dem Evaluierungsset evaluiert, nach ungefähr 10 Epochen veränderte sich die Genauigkeit des Entity-Typers nicht mehr signifikant. Die Trainings-Zeit belief sich auf knapp eine Woche mit einer modernen High End Grafikkarte. Aufgrund dieser hohen Trainings-Zeit wurde das Modell nur einmal trainiert, es wurden also keine Wiederholungen des Experiments durchgeführt.

Bewertet wurde das Modell mit einem Makro F1-Score, wie in Kapitel 2.3 beschrieben. Da es sich bei dem Entity-Typer lediglich um eine Komponente des Entity-Linkers handelt, wurde der Entity-Typer nicht wie sonst üblich mit ungesehenen Daten getestet, sondern lediglich mit den Daten aus dem Evaluierungsset DET_{EVAL} evaluiert. Dabei wies der Entity-Typer einen Makro F1-Score von 39,1 % auf. Dieser Wert stellte sich bei ET4EL von Onoe und Durrett als für die Entity-Linking Aufgabe genügend heraus.[123]

Der Candidate-Ranker wurde mit dem AIDA Datenset trainiert, welches vor dem Training mit Daten von Wikidata angereichert wurde.[124] Da das AIDA Datenset ebenfalls eine Kandidatenliste besitzt, welche vordefinierte Kandidaten zu jedem Testbeispiel vorgibt, mit einem Recall@10 von 100 % für das Trainings- und TestA-Datenset aufweist, konnte so der Candidate-Ranker über durch diese Liste ersetzt, bzw. simuliert werden. Um Wikidata-Attribute der Kandidaten und der Entitäten während des Trainings nutzen zu können, wurden das AIDA Datenset im Vorhinein mit den 100 am häufigsten in der gesamten Wissensdatenbank aufkommenden Attribute angereichert.[125] Das resultierende Trainingsdatenset $D_{CR-Train} = \{(m, s, e, C)^{(1)}, (m, s, e, C)^{(2)}, ...,$

[121] Vgl. Onoe/Durrett 2020, S. 8579
[122] z. Dt. Binäre-Kreuzentropie
[123] Vgl. Onoe/Durrett 2020, S. 8578 ff.
[124] Sieh Arbeiten über das AIDA Datenset von Hoffart 2013 S. 44 und Hoffart u. a. 2011 S. 789 f.
[125] Siehe: Wikidata 2021

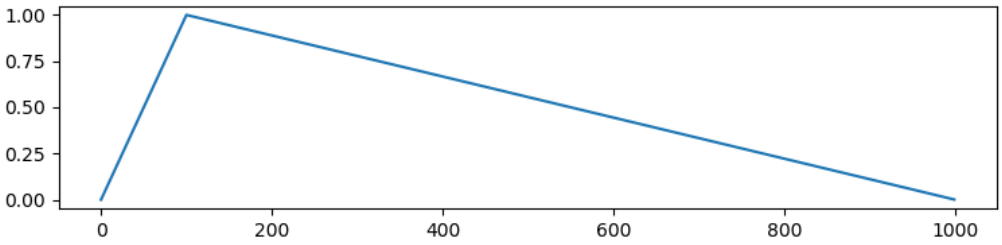

Abb. 8: Die Lernrate nimmt bis zu einer vordefinierten Trainingsiteration, in der Regel das Ende des ersten Batches, bis zu einem eingestelltem Höchststand linear zu, um daraufhin bis zum Ende des Trainings linear abzunehmen. (Lineares Aufwärmen der Lernrate)[126]

$(m, s, e, C)^{(n)}\}$ basiert auf dem AIDA Trainingsset mit 17917 Beispielen und das Evaluierungsdatenset $D_{CR-Eval} = \{(m, s, e, C)^{(1)}, (m, s, e, C)^{(2)}, ..., (m, s, e, C)^{(m)}\}$ basiert auf dem mit AIDA TestA Datenset mit 4673 Beispielen.

Da der Candidate-Ranker sehr viel Arbeitsspeicher, sowie viel Zeit zum Verarbeiten von Beispielen benötigt, wurde über lediglich drei Epochen mit einer Batchgröße von zwei Beispielen trainiert. Um ein Overfitting von Beispielen zu verhindern, wurde die Lernrate über die erste Epoche linear aufgebaut bis sie einen Höchstwert von 2e−5 erreicht hat. Daraufhin hat sie im weiteren Verlauf des Trainings bis zur letzten Epoche linear abgenommen, wie in Abbildung 8 zu erkennen ist. Da bei der Benutzung von BERT Transformern häufig der AdamW Optimizer benutzt wird, wurde auch in diesem Training der AdamW Algorithmus zur Backpropagation über eine Cross Entropy [127] Verlustfunktion genutzt.

Nach 20 Epochen konnte das Modell eine F1-Genauigkeit von lediglich 67,9 % auf das Evaluierungsdatenset erreichen. Mögliche Ursachen für dieses im Vergleich zu anderen Modellen schlechtem Ergebnis sind die geringe Anzahl an Epochen, sowie die geringe Größe der Batches. Die geringe Anzahl an Epochen ist durch die lange Trainingszeit einer Epoche notwendig, um das Modell innerhalb weniger Tage trainieren zu können. Ebenso war es aufgrund der Limitierungen des Arbeitsspeichers (ca. 8 GB) nicht möglich größere Batches zu nutzen. Es kann also allgemein gesagt werden, dass dieses Modell zu viel Arbeitsspeicher benötigt und gleichzeitig zu lange für ein Beispiel rechnet.

4.2 Überarbeitung der ACCEL Architektur

Die in Kapitel 4.1.3 beschriebenen Probleme sind auf Probleme in der Architektur des Modells zurückzuführen. So werden in dem ersten Candidate-Ranker des ACCEL zwei separate BERT Transformer zusätzlich zum Deep Learning Modell des Entity-Typers genutzt. Aufgrund ihres komplexen Aufbaus benötigen BERT Transformer viel Arbeitsspeicher und ebenfalls viel Zeit, um

[126] Enthalten in: Huggingface 2021
[127] z. Dt. Kreuzentropie

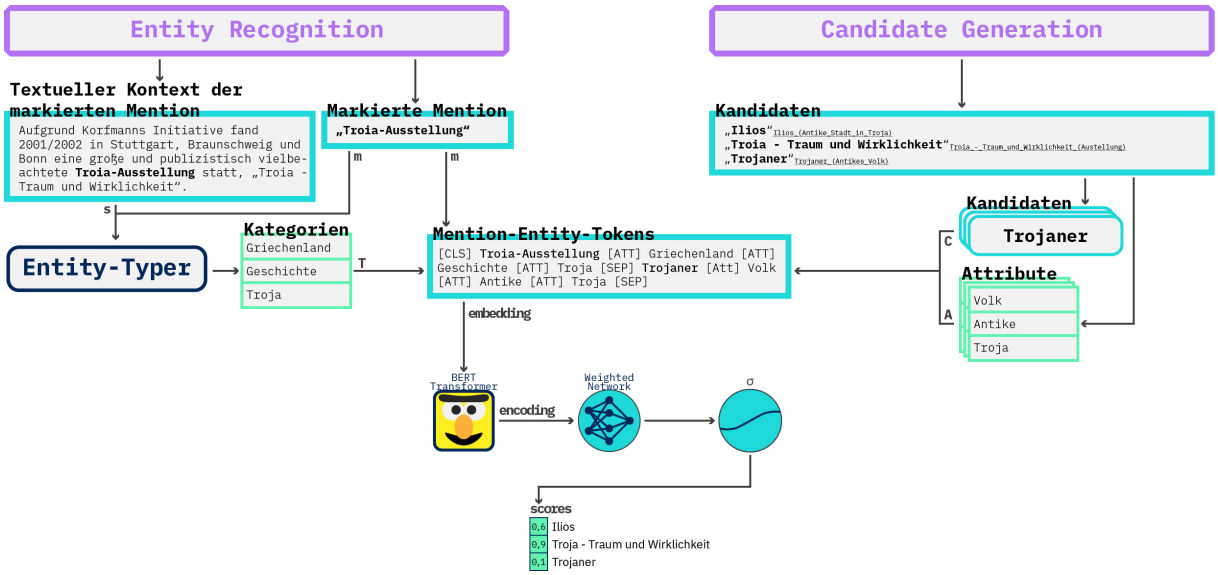

Abb. 9: Der Candidate-Ranker abstrahiert die Erwähnung und die Kandidaten, um sie mit einem, anstelle von zwei, BERT Transformer zu encodieren und zu vergleichen. So wird in diesem Beispiel der Erwähnung *Troia-Ausstellung* die Entität *Troja* mit ein Score von 0,9 zugewiesen. (Architektur Candidate Ranker V2)

die Beispiele zu verarbeiten, ebenso verhält es sich mit dem Entity-Typer. Eine neue Architektur des Candidate-Rankers des ACCEL soll diese Zustände verbessern.

4.2.1 Architektur des überarbeiteten Candidate-Ranker

Die neue Architektur nutzt ebenso wie in Kapitel 4.1.2 (Architektur des Candidate-Rankers) einen Entity-Typer um einer Erwähnung m Kategorien T zuzuweisen, um sie so abstrahieren zu können. Jedoch werden nun die Erwähnung und der Kandidat zusammen in eine Token-Sequenz eingebettet und anschließend encodiert, anstelle einer separaten Encodierung. So wird zunächst eine gemeinsame Token-Sequenz aus der Erwähnung, ihren Kategorien, des Kandidaten und seiner Attribute erzeugt.

$$token_{m,e} = [CLS]\ m\ [ATT]\ t_1\ [ATT]\ t_2\ [ATT]\ ...\ t_i\ [SEP] \\ e\ [ATT]\ a_1\ [ATT]\ a_2\ [ATT]\ ...\ a_j\ [SEP] \quad (4.3)$$

Diese Token-Sequenz ist bis auf die Ausnahme des *[CLS]* Tokens der Kandidat-Token-Sequenz identisch mit der Zusammensetzung der beiden Token-Sequenzen $token_m$ und $token_e$ der Gleichungen (4.1) und (4.2) aus dem Kapitel 4.1.2.

Die Token-Sequenz wird daraufhin mit einem BERT Transformer encodiert, auch hier ergibt sich die resultierende Encodierung $encoding_{m,e} = pooler(BERT(token_{m,e})) \in \mathbb{R}^d$ aus dem Pooler-Output des Transformers. Diese Encodierung wird daraufhin in einem MLF mit einem Gewicht $W \in \mathbb{R}^{|C| \times d}$ multipliziert und mit einer Sigmoid Funktion normalisiert, wodurch sich ein Vektor

mit derselben Größe $|C|$ der Kandidaten-Liste ergibt. Das normalisierte Ergebnis dieses Vorgangs kann dementsprechend als Scores $score = \sigma(W \times encoding_{m,e})$ der einzelnen Kandidaten angesehen werden. Der Kandidat mit dem höchsten Score wird zur Erwähnung verlinkt.

4.2.2 Bestimmung der Hyperparameter des Candidate-Rankers

Versuch Nr.	1	2	3
Lernrate	2e-6	1e-5	1e-6
Batchgröße	2	8	8
Evaluierungs-Genauigkeit (%)	90,7	94	93,1

Tab. 1: Suche nach den besten Hyperparameter

Aufgrund des geringeren Arbeitsspeicher-Verbrauchs sowie der geringeren Rechenzeit war es möglich für dieses Training die Batchgröße, sowie die Anzahl an Epochen zu vergrößern. Somit konnte das neue Modell mit einer Batchgröße von bis zu acht Beispielen trainiert werden. Um eine Stagnation der Evaluierungs-Genauigkeit während dem Training zu gewährleisten wurde das Modell über 20 Epochen trainiert, dabei dauert das Training sieben Stunden. Nach ungefähr 10 Epochen verbesserte sich die Genauigkeit des Candidate-Rankers nicht mehr signifikant. Die Tabelle 1 zeigt die Resultate von drei Experimenten, bei welchen jeweils die Hyperparameter Batchgröße und Lernrate verändert wurden. Aufgrund der weiterhin langen Dauer des Trainings war es nicht möglich weitere Experimente bezüglich der Hyperparameter durchzuführen, wodurch die Ergebnisse entsprechend ungenau sein können. Besagte Tabelle macht die Wichtigkeit einer ausreichend großen Batchgröße deutlich, mehr als zwei Prozentpunkte konnte eine Erhöhung der Batchgröße von zwei Beispielen in Experiment (1) auf acht Beispielen in Experiment (2) und (3) pro Batch erzielen. Ebenso spielt die Lernrate eine wichtige Rolle, welche in den Experimenten (2) und (3) knapp weniger als einen Prozentpunkt Unterschied ausmachen konnte. Über diese Suche nach Hyperparameter wurde somit eine Lernrate von $1e{-}5$ bestimmt, welche sich wie in Kapitel 4.1.3 beschrieben über die erste Epoche aufbaut und sich daraufhin über die verbleibenden Epochen linear abbaut.

5 Evaluierung und Test des ACCEL Entity-Linker

Getestet wurde der ACCEL Entity-Linker gegen das AIDA Test-B Datenset, welches zur Wikidata Wissensdatenbank verlinkt.[128] Das AIDA Datenset weist ebenfalls eine Kandidatenliste auf, welche vordefinierte Kandidaten zu jedem Testbeispiel vorgibt. Somit konnte bei dem Testen des ACCEL auf einen eigenen Candidate-Generator verzichtet werden, bzw. wurde dieser durch die bereits verfügbaren Daten simuliert. Diese Kandidatenliste weist für das AIDA TestB Datenset eine Genauigkeit von 92,37 % auf, bei jedem etwa 13. Beispiel ist die zu verlinkende Entität *nicht* in der Kandidatenliste der Erwähnung und kann somit auch nicht von einem Candidate-Ranker gefunden werden. Das Datenset selbst besitzt 4313 Beispiele und wurde noch mit weiteren Daten von Wikidata angereichert. Hierbei handelt es sich um die 100 am häufigsten in der gesamten Wissensdatenbank aufkommenden Attribute von jeder Entität und jedem Kandidaten des Datensets.[129]

Der ACCEL Entity-Linker konnte bei einem Test gegen das AIDA Test-B Datenset eine Genauigkeit von 76,61 % aufweisen. Unter Rücksichtnahme der Genauigkeit des simulierten Candidate-Generators kann die für ACCEL verantwortliche Genauigkeit bestimmt werden:

$$\frac{0,7661}{0,9237} = 0,8294$$

Somit könnte der ACCEL Entity-Linker in etwa eine Genauigkeit von 82,94 % mit einem unfehlbaren, also 100 % genauen, Candidate-Generator erreichen. Entsprechend kann gesagt werden, dass der Candidate-Ranker des ACCEL Entity-Linkers eine Genauigkeit von 82,94 % auf das AIDA TestB Datenset aufweist.

Für das Testen des Entity-Linkers wurde das Candidate-Ranking Modell in den Inferenzmodus geschaltet. Dies sorgt dafür, dass bei der Verarbeitung eines Testbeispiels keine Gradienten berechnet werden, wodurch weniger Arbeitsspeicher und weniger Rechenzeit für die Verarbeitung benötigt wird. So belief sich der Test auf ca. 2 Minuten und verbrauchte dabei maximal 6,61 GB Grafikkarten-Arbeitsspeicher.

[128] Sieh Arbeiten über das AIDA Datenset von Hoffart 2013 S. 44 und Hoffart u. a. 2011 S. 789 f.
[129] Siehe: Wikidata 2021

6 Diskussion und Vergleich mit weiteren Systemen

Die Tabelle 2 vergleicht unter anderem die Genauigkeiten der in Kapitel 2.4 beschriebenen Entity-Linker mit dem ACCEL Entity-Linker. Der ACCEL Entity-Linker schneidet im Vergleich mit 76,61 % deutlich schlechter als die anderen Linker mit >85 % ab. Dieser Umstand kann unterschiedlichste Ursachen haben, so kann beispielsweise ein hocheffektiver Candidate-Generator ausschlaggebend für ein besseres Ergebnis sein: würde dieser zu in jeder Kandidatenliste auch den richtigen Kandidaten liefern, so würde sich wie in Kapitel 5 beschrieben die Genauigkeit des ACCEL auf ca. 82,94 % belaufen. Als weitere Ursache kann die Wahl unpassender Hyperparameter gesehen werden. Unter Umständen könnte durch eine vollständige Suche nach Hyperparameter, sowie eine Analyse dieser, zu einer stark höheren Genauigkeit des Systems führen. Dies geschah jedoch nur unvollständig: Es wurden lediglich drei verschiedene Hyperparameter-Kombinationen ausprobiert, woraus sich keine optimale Konfiguration ableiten lässt.

Nichtsdestotrotz ist der Unterschied zwischen den besten Systemen und des ACCEL ausreichend groß, sodass die Forschungsfrage, ob *die beiden Methoden (1) der Abstraktion von Erwähnungen durch Kategorien und (2) der Abstraktion von Entitäten aus Wissensdatenbanken durch Attribute harmonieren*, zu beantworten. Es war möglich die beiden Methoden in einem Modell erfolgreich zu vereinigen, jedoch erweist sich die Abstraktion einer Erwähnung durch Kategorien als unzureichend effektiv aus, um eine Verbesserung gegenüber anderen Systemen bei der Lösung der Entity-Linking Aufgabe darzustellen. Ebenso ist diese Methode weniger effizient als die Methode von Onoe & Durrett: Sie benötigt deutlich mehr trainierbare Parameter und aufgrund der komplexen Architektur bestehend aus zwei trainierbaren Modellen. Daraus ergibt sich ebenfalls eine leicht längere Trainingszeit, sowie deutlich mehr benötigter Arbeitsspeicher der Grafikkarte zum Trainieren der Modelle. Somit kann bezüglich der Forschungsfrage abschließend gesagt werden, dass die Vereinigung der beiden Methoden sich als zu komplex und somit als nicht empfehlenswert herausstellt.

6.1 Vertrauenswürdigkeit der ACCEL KI

Ein weiterer wichtiger Punkt, welcher bei der Entwicklung von mit maschinellem Lernen trainierten Modellen erwähnt werden sollte, ist die Vertrauenswürdigkeit des Systems, insbesondere die Erklärbarkeit. Das ACCEL Modell weist eine in kleinteilige Schritte unterteilte Architektur auf, welche hilft die Entscheidungen des Modells zu erklären. So kann nach der Candidate-Generation das Modell konkret zeigen, welche Kandidaten es für eine Erwähnung ausgewählt hat. Zu jedem Kandidaten kann es zusätzlich einen Score angeben, was ebenfalls zur Erklärbarkeit der Entscheidungsfindung des Modells beiträgt. Der Candidate-Ranker ist in die zwei Schritte (1) Abstraktion der Erwähnungen und (2) Ranking basierend auf den Abstraktionen unterteilt, beide können ihre Entscheidungsfindung über Scores erklären. Der Entity-Typer kann darüber hinaus auch die zur

[130] Werte teilweise entommen aus Onoe/Durrett 2020 S. 8581, Cao u. a. 2021 S. 6 und Yamada u. a. 2019 S. 3

Entity-Linker	Acc.	Trainingszeit	Mem.	Parameteranzahl
ACCEL	76,61 %	6 Tage + 7 Stunden	min. 10 GB	77.3 M + 109 M
Onoe & Durrett (2020)	85,9 %	6 Tage	ca. 4 GB	77.3 M
Cao et al. (2021)	93,3 %	unbekannt	max. 12	unbekannt
Yamada et al. (2020)	95 %	10 Tage	max. 16	unbekannt

Tab. 2: Vergleich verschiedener Entity-Linker durch Testgenauigkeit (Acc.) auf das AIDA-TestB Datenset, Trainingszeit, Arbeitsspeicher (Mem.) und Parameteranzahl. (Vergleich verschiedener Entity-Linker mit ACCEL)[130]

Erwähnung zugewiesenen Kategorien aufzeigen, es kann also die Abstraktion der Erwähnung in für Menschen verständliche Worte gefasst werden.

Ein Problem des ACCEL Modells ist jedoch das Fehlen der NIL-Prediction. Dadurch ist es dem Nutzer des Modells nicht möglich fehlende Entitäten in der Wissensdatenbank zu erkennen, ebenso wenig wie vom Entity-Recognition Modell falsch erkannte Erwähnungen. Des Weiteren wurde das ACCEL Modell nicht gegen verschiedenste Biases getestet, genauso wenig fand eine Überprüfung der Trainingsdaten auf ethische Korrektheit statt.

6.2 Ausblick

Ein potenzieller Schritt die Genauigkeit des ACCEL zu erhöhen wäre die Durchführung von weiteren Experimenten mit dem Entity-Typer. Dieser ist maßgeblich für die Abstraktion der Erwähnung zuständig und somit äußerst relevant für die letztendliche Entscheidung des Entity-Linkers. Es könnte zum Beispiel eine genauere Analyse über die Auswirkung des Grenzwerts *threshold* des Entity-Typers durchgeführt werden, da dieser sehr direkte Auswirkung auf die Auswahl der Kategorien, also der Abstraktionen, der Erwähnung hat. Ebenso könnte versucht werden einen anderen Entity-Typer für die Abstraktion der Erwähnung zu nutzen, um die Wirkung dieses speziellen Modells zu evaluieren. Eine weitere Möglichkeit den Entity Linker zu verbessern wäre den Candidate-Generator zu untersuchen und zu verbessern, da in dieser Arbeit der Candidate-Generator nicht näher behandelt wurde.

7 Fazit

Aufgrund des immer wachsenden Wissens der Menschheit ist die Nutzung sogenannter Wissensdatenbanken u. a. in der Materialforschung unumgänglich. Um die Nutzung dieser zu vereinfachen, untersuchte diese Arbeit eine neuartige Lösung der Entity-Linking Aufgabe, um später als Teil einer NLP Anwendung dazu beizutragen, leichter auf Wissensdatenbanken zugreifen zu können. Diese neuartige Lösung beinhaltet die Vereinigung zwei in der Forschung benutzten Methoden, woraus sich die Fragestellung, ob *die beiden Methoden (1) der Abstraktion von textlichen Erwähnungen durch Kategorien und (2) der Abstraktion von Entitäten aus Wissensdatenbanken durch Attribute harmonieren* ergab.

Um die Forschungsfrage zu beantworten wurde ein neues Entity-Linking System entwickelt: ACCEL. Bei der Entwicklung des Systems entstanden drei Artefakte: ein neuartiger Entity-Typer, sowie zwei Candidate-Ranker, wobei alle Artefakte auf KNN basierte und somit mit maschinellem Lernen trainierbare Modelle sind. Nach einer genaueren Evaluierung dieser Artefakte konnte der neu entwickelte ACCEL Entity-Linker als ineffektiv zur Lösung der Entity-Linking Aufgabe bewertet werden. Daraus ergab sich bezüglich der Forschungsfrage, dass Vereinigung der beiden Abstraktionsmethoden für textliche Erwähnungen und Entitäten zwar harmonieren, jedoch zu komplex und daher nicht effektiv genug ist.

Jedoch stellte sich heraus, dass der ACCEL Entity-Linker seine Ergebnisse besonders gut erklärbar darstellen kann. Somit könnte in Zukunft der Entscheidungsfindungsprozess des Systems genauer analysiert werden, wodurch eine weitere Entwicklung und Anpassung der Komponenten des Systems zu erleichtern und somit die Effektivität des Systems weiter zu erhöhen. Außerdem kann das Vertrauen von Menschen in solche Systeme erhöht werden, wodurch sich Benutzer besser auf das Ergebnis verlassen können.

Als weiteres Resultat dieser Arbeit kann die Python Implementierung des Artefakts mit der Python Bibliothek PyTorch-Lightning angesehen werden. Diese Implementierung fokussiert sich auf Reproduzierbarkeit der Ergebnisse, desweiteren können anderere Forschende Teile der Implementierung nutzen und weiter verbessern.

Abschließend kann als Fazit dieser Arbeit gezogen werden, dass Methoden zur Abstraktion durch Kategorien und Attribute für Entity-Linking Modelle zwar effektiv sein können, jedoch häufig zu einer zu komplexen Architektur des Modells führen. Trotzdem können solche Methoden zur Verbesserung der Erklärbarkeit der Ergebnisse beitragen.

Literaturverzeichnis

Brown, T. B./Mann, B./Ryder, N./Subbiah, M./Kaplan, J./Dhariwal, P./Neelakantan, A./Shyam, P./Sastry, G./Askell, A./Agarwal, S./Herbert-Voss, A./Krueger, G./Henighan, T./Child, R./Ramesh, A./Ziegler, D. M./Wu, J./Winter, C./Hesse, C./Chen, M./Sigler, E./Litwin, M./Gray, S./Chess, B./Clark, J./Berner, C./McCandlish, S./Radford, A./Sutskever, I./Amodei, D. (2020): Language Models are Few-Shot Learners. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Hrsg. von Hugo Larochelle/Marc'Aurelio Ranzato/Raia Hadsell/Maria-Florina Balcan/Hsuan-Tien Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Cadez, I./Heckerman, D./Meek, C./Smyth, P./White, S. (2003): Model-Based Clustering and Visualization of Navigation Patterns on a Web Site. In: *Data Mining and Knowledge Discovery* 7, S. 399–424. DOI: 10.1023/A:1024992613384.

Cao, N. D./Izacard, G./Riedel, S./Petroni, F. (2021): Autoregressive Entity Retrieval. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. URL: https://openreview.net/forum?id=5k8F6UU39V.

Devlin, J./Chang, M.-W./Lee, K./Toutanova, K. (2019): BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Hrsg. von Jill Burstein/Christy Doran/Thamar Solorio. Association for Computational Linguistics, S. 4171–4186. DOI: 10.18653/v1/n19-1423. URL: https://doi.org/10.18653/v1/n19-1423.

Ellis, J./Getman, J./Strassel, S. (2018): TAC KBP English Entity Linking - Comprehensive Training and Evaluation Data 2009-2013 LDC2018T16. Philadelphia. DOI: https://doi.org/10.35111/13g2-th80. URL: https://catalog.ldc.upenn.edu/LDC2018T16.

Eschweiler, M./Evanschitzky, H./Woisetschläger, D. (2009): Laborexperiment. In: *Empirische Mastertechniken: Eine anwendungsorientierte Einführung für die Marketing- und Managementforschung*. Hrsg. von Carsten Baumgarth/Martin Eisend/Heiner Evanschitzky. Wiesbaden: Gabler Verlag, S. 361–388. ISBN: 978-3-8349-8278-0. DOI: 10.1007/978-3-8349-8278-0_12. URL: https://doi.org/10.1007/978-3-8349-8278-0_12.

Falcon, W. u. a. (2019): PyTorch Lightning. Python Bibliothek. URL: https://pytorch-lightning.readthedocs.io/en/latest/.

Gregor, S./Hevner, A. R. (2013): Positioning and Presenting Design Science Research for Maximum Impact. In: *MIS Q.* 37.2, S. 337–355. URL: http://misq.org/positioning-and-presenting-design-science-research-for-maximum-impact.html.

Hochreiter, S./Schmidhuber, J. (1997): Long Short-Term Memory. In: *Neural Comput.* 9.8, S. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

Hoffart, J. (2013): Discovering and disambiguating named entities in text. In: *Proceedings of the 2013 SIGMOD/PODS Ph.D. Symposium, New York, NY, USA, June 23, 2013*. Hrsg. von Lei Chen/Xin Luna Dong. ACM, S. 43–48. DOI: 10.1145/2483574.2483582. URL: https://doi.org/10.1145/2483574.2483582.

Hoffart, J./Yosef, M. A./Bordino, I./Fürstenau, H./Pinkal, M./Spaniol, M./Taneva, B./Thater, S./Weikum, G. (2011): Robust Disambiguation of Named Entities in Text. In: S. 782–792. URL: https://aclanthology.org/D11-1072/.

Hopfield, J. J. (1982): Neural networks and physical systems with emergent collective computational abilities. eng. In: *Proceedings of the National Academy of Sciences of the United States of America* 79.8. PMC346238[pmcid], S. 2554–2558. ISSN: 0027-8424. DOI: 10.1073/pnas.79.8.2554. URL: https://pubmed.ncbi.nlm.nih.gov/6953413.

Horev, R. (2018): BERT Explained: State of the art language model for NLP. URL: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270 (besucht am 15. 11. 2021).

Huggingface (2021): Optimization — transformers 4.11.3 documentation. Huggingface. URL: https://huggingface.co/transformers/main_classes/optimizer_schedules.html (besucht am 15. 11. 2021).

Kingma, D. P./Ba, J. (2015): Adam: A Method for Stochastic Optimization. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Hrsg. von Yoshua Bengio/Yann LeCun. URL: http://arxiv.org/abs/1412.6980.

Lin, Y./Ji, H. (2019): An Attentive Fine-Grained Entity Typing Model with Latent Type Representation. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, S. 6197–6202. DOI: 10.18653/v1/D19-1641. URL: https://aclanthology.org/D19-1641.

Masson-Delmotte, V./P. Zhai, A. P./Connors, S. L./Péan, C./Berger, S./Caud, N./Chen, Y./Goldfarb, L./Gomis, M. I./Huang, M./Leitzell, K./Lonnoy, E./Matthews, J./Maycock, T. K./Waterfield, T./Yelekçi, O./Yu, R./Zhou, B. (2021): IPCC, 2021: Summary for Policymakers. In: *The Physical Science Basis. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*.

Nielsen, M. A. (2015): Neural Networks and Deep Learning. Determination Press. URL: http://neuralnetworksanddeeplearning.com/.

Onoe, Y./Durrett, G. (2019): Learning to Denoise Distantly-Labeled Data for Entity Typing. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Hrsg. von Jill Burstein/Christy Doran/Thamar Solorio. Association for Computational Linguistics, S. 2407–2417. DOI: 10.18653/v1/n19-1250. URL: https://doi.org/10.18653/v1/n19-1250.

– **(2020)**: Fine-Grained Entity Typing for Domain Independent Entity Linking. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative*

Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020. AAAI Press, S. 8576–8583. URL: https://aaai.org/ojs/index.php/AAAI/article/view/6380.

Otter, D. W./Medina, J. R./Kalita, J. K. (2021): A Survey of the Usages of Deep Learning for Natural Language Processing. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.2, S. 604–624. DOI: 10.1109/TNNLS.2020.2979670.

Peters, M. E./Neumann, M./Iyyer, M./Gardner, M./Clark, C./Lee, K./Zettlemoyer, L. (2018): Deep Contextualized Word Representations. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. Hrsg. von Marilyn A. Walker/Heng Ji/Amanda Stent. Association for Computational Linguistics, S. 2227–2237. DOI: 10.18653/v1/n18-1202. URL: https://doi.org/10.18653/v1/n18-1202.

Rashid, T. (2017): Neuronale Netze selbst programmieren - Ein verständlicher Einstieg mit Python. Heidelberg: Dpunkt.Verlag GmbH. ISBN: 978-3-960-09043-4.

Rosenblatt, F. (1962): Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Cornell Aeronautical Laboratory. Report no. VG-1196-G-8. Spartan Books. URL: https://books.google.ca/books?id=7FhRAAAAMAAJ.

Rossi, F./Sekaran, A./Spohrer, J./Caruthers, R./Cutler, A./Pribić, M./Humphrey, L. (2019): Everyday Ethics for Artificial Intelligence. IBM Corp. URL: https://www.ibm.com/watson/assets/duo/pdf/everydayethics.pdf (besucht am 15. 11. 2021).

Ruder, S. (2021): Entity Linking. Englisch. URL: http://nlpprogress.com/english/entity_linking.html (besucht am 15. 11. 2021).

Sevgili, Ö./Shelmanov, A./Arkhipov, M. Y./Panchenko, A./Biemann, C. (2020): Neural Entity Linking: A Survey of Models based on Deep Learning. In: *CoRR* abs/2006.00575. URL: https://arxiv.org/abs/2006.00575.

Shah, T. (2020): About Train, Validation and Test Sets in Machine Learning. URL: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7 (besucht am 15. 11. 2021).

Shen, W./Wang, J./Han, J. (2014): Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. In: *IEEE Transactions on Knowledge and Data Engineering* 27.2, S. 443–460. ISSN: 1041-4347. URL: http://dx.doi.org/10.1109/tkde.2014.2327028.

Sil, A./Cronin, E./Nie, P./Yang, Y./Popescu, A.-M./Yates, A. (2012): Linking Named Entities to Any Database. Englisch. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. EMNLP-CoNLL '12, S. 116–127. URL: https://aclanthology.org/D12-1011.

Singh, S./Subramanya, A./Pereira, F./McCallum, A. (2012): Wikilinks: A Large-scale Cross-Document Coreference Corpus Labeled via Links to Wikipedia. technischer Bericht UM-CS-2012-015. University of Massachusetts, Amherst. URL: https://web.cs.umass.edu/publication/docs/2012/UM-CS-2012-015.pdf (besucht am 15. 11. 2021).

Srivastava, N. (2013): Improving Neural Networks with Dropout. Masterarbeit. University of Toronto.

Svozil, D./Kvasnicka, V./Pospichal, J. (1997): Introduction to multi-layer feed-forward neural networks. In: *Chemometrics and Intelligent Laboratory Systems* 39.1, S. 43–62. ISSN: 0169-7439. DOI: https://doi.org/10.1016/S0169-7439(97)00061-0. URL: https://www.sciencedirect.com/science/article/pii/S0169743997000610.

Vaswani, A./Shazeer, N./Parmar, N./Uszkoreit, J./Jones, L./Gomez, A. N./Kaiser, L./Polosukhin, I. (2017): Attention is All you Need. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Hrsg. von Isabelle Guyon/Ulrike von Luxburg/Samy Bengio/Hanna M. Wallach/Rob Fergus/S. V. N. Vishwanathan/Roman Garnett, S. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Von Bartheld, C. S./Bahney, J./Herculano-Houzel, S. (2016): The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting. In: *Journal of Comparative Neurology* 524.18, S. 3865–3895. DOI: https://doi.org/10.1002/cne.24040. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cne.24040. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cne.24040.

Vyas, Y./Ballesteros, M. (2021): Linking Entities to Unseen Knowledge Bases with Arbitrary Schemas. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. URL: http://dx.doi.org/10.18653/v1/2021.naacl-main.65.

Wikidata (2021): Wikidata:Database reports/List of properties/Top100. Wikidata. URL: https://www.wikidata.org/wiki/Wikidata:Database_reports/List_of_properties/Top100 (besucht am 15. 11. 2021).

Yamada, I./Washio, K./Shindo, H./Matsumoto, Y. (2019): Global Entity Disambiguation with Pretrained Contextualized Embeddings of Words and Entities. In: *arXiv: Computation and Language*.

Yang, J./Li, J. (2017): Application of deep convolution neural network. In: *2017 14th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, S. 229–232. DOI: 10.1109/ICCWAMTIP.2017.8301485.

Yu, L./Hermann, K. M./Blunsom, P./Pulman, S. (2014): Deep Learning for Answer Sentence Selection. In: *CoRR* abs/1412.1632. arXiv: 1412.1632. URL: http://arxiv.org/abs/1412.1632.

Zhang, X./Chen, X./Yao, L./Ge, C./Dong, M. (2019): Deep Neural Network Hyperparameter Optimization with Orthogonal Array Tuning. In: *Neural Information Processing - 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12-15, 2019, Proceedings, Part IV*. Hrsg. von Tom Gedeon/Kok Wai Wong/Minho Lee. Bd. 1142. Communications in Computer and Information Science. Springer, S. 287–295. DOI: 10.1007/978-3-030-36808-1_31. URL: https://doi.org/10.1007/978-3-030-36808-1%5C_31.

Appendix

Appendix Directory

Anhang 1 Glossar 35
Anhang 2 PyTorch Lightning Implementierung des Entity-Typers 37
Anhang 3 PyTorch Lightning Implementierung des Candidate-Rankers 58

Anhang 1: Glossar

Englisches (Fach-)Wort	**Deutsche Übersetzung**
Entity-Linking	Verlinken von Entitäten
(Fine-Grained) Entity-Typing	Detailreiche Typisierung von Entitäten
Entity-Disambiguation	Entität Disambiguierung
Entity-Recognition	Erkennung von Entitäten
Mention	Erwähnung
Sentence	Satz
Knowledge Base	Wissensdatenbank
Encoding	Encodierung
Typeset	Kategorie-Wörterbuch
NLP	Verarbeitung natürlicher Sprache
NIL-Precition	NULL-Vorhersage
Unlinkable-Mention-Prediction	Vorhersage von nicht verknüpfbaren Erwähnungen
Candidate-Generation	Kandidatensuche
Candidate-Ranking	Kandidatenbewertung
Machine Learning	maschinelles Lernen
Deep Learning	Tiefen-Lernen
Multi-layer feed-forward neural networks	neuronale Multi-Schicht-Netze
Surface Form Matching	Oberflächenform Abgleich
Alias Expansion	Alias Erweiterung
Prior-Probability-Computation	Wahrscheinlichkeitsberechnung
Name-Dictionary-Technique	Namenswörterbuch Technik
Attribute-Seperation	Attribut Separierung
Layer	Schicht
Transformer	Umwandler
Score	Score (steht im Duden)
Binary Cross Entropy	Binäre-Kreuzentropie
Loss function	Verlustfunktion
Optimizer	Optimierer
Backpropagation	Fehlerrückführung
Batch	Batch
Overfitting	Übermäßige Anpassung
Dropout	Signal-Löschung
F1-score	F1-Score/Maß

Englisches (Fach-)Wort	**Deutsche Übersetzung**
Convolution Neural Network	faltendes neuronales Netz
Recurrent Neural Network	Rekurrentes neuronales Netz
ELMo	Einbettungen von Sprachmodellen
BERT	Bidirektionale Encoder-Repräsentation von Umwandlern
Explainability	Erklärbarkeit
Biases	Biases
Trustful AI	Vertrauenswürde KI
Self-Attention	Eigene Aufmerksamkeit
Masked-Language-Model	maskiertes Sprachmodell
Tokenizer	Tokenisator
Pooler-Output	Pooler-Ausgabe

Anhang 2: PyTorch Lightning Implementierung des Entity-Typers

```
from collections import defaultdict
from dataclasses import dataclass
from functools import lru_cache
from os import path
from typing import List, Tuple

import pytorch_lightning as pl
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchmetrics
from allennlp.data.tokenizers.spacy_tokenizer import SpacyTokenizer
from allennlp.modules.elmo import Elmo, batch_to_ids
from torch.autograd import Variable
from torch.nn import functional as F
from torch.nn.utils.rnn import (pack_padded_sequence,
                                pad_packed_sequence)

from accel.utils import Mention, Mentions

tokenizer = SpacyTokenizer()

MAX_TOKEN_LENGTH = 50


def load_vocab_dict(vocab_file_name, vocab_max_size=None):
    """Loads vocabulary from file ("conll_categories.txt")
        and maps the first X entries to a dict of ids and
        categories
    """
    with open(vocab_file_name, encoding="utf-8") as f:
        text = [x.strip() for x in f.readlines()]
        text = text[:vocab_max_size]
        file_content = dict(zip(text, range(len(text))))
    return file_content


@dataclass(frozen=True)
class EmbeddedBatch():
```

```
40      mention_embeddings: torch.Tensor
41      sentence_embeddings: torch.Tensor
42
43
44  class TyperMention(Mention):
45      """Mention type which wraps a mention and its context into
46          one single class
47
48      Properties:
49      `mention`: `str` The mention in the text
50      `left_context`: `str` The context before the mention
51      `right_context`: `str` The context after the mention
52      `sequence`: `str` The complete sentence: left_context
53                  + mention + right_context
54      `tokens`: `List[str]` Splitted sequence
55      `tokens_length`: `int` Number of tokens in sequence
56      `borders`: `Tuple[int, int]` Border indicies of the mention
57      """
58
59      mention: str
60      left_context: str
61      right_context: str
62
63      @staticmethod
64      def find_index_in_sequence(a: list, b: list):
65          return [(i, i + len(b)) for i in range(len(a))
66                  if a[i:i + len(b)] == b]
67
68      @staticmethod
69      def replace_numbers(tokens):
70          return [
71              "<number>" if t.ensure_text().isnumeric() else
72              t.ensure_text() for t in tokens
73          ]
74
75      @property
76      @lru_cache()
77      def tokens(self) -> List[str]:
78          left_tokens = self.replace_numbers(
79              tokenizer.tokenize(
80                  self.left_context))[-MAX_TOKEN_LENGTH:]
```

```
81          mention_tokens = self.replace_numbers(
82              tokenizer.tokenize(self.mention))
83          right_tokens = self.replace_numbers(
84              tokenizer.tokenize(
85                  self.right_context))[:MAX_TOKEN_LENGTH]
86          return left_tokens + mention_tokens + right_tokens
87
88      @property
89      @lru_cache()
90      def tokens_length(self) -> int:
91          return len(self.tokens)
92
93      @property
94      @lru_cache()
95      def borders(self) -> Tuple[int, int]:
96          mention_tokens = [
97              t.ensure_text()
98              for t in tokenizer.tokenize(self.mention)
99          ]
100         return self.find_index_in_sequence(
101             self.tokens, mention_tokens)[0]
102
103     @property
104     @lru_cache()
105     def mention_length(self) -> int:
106         start, end = self.borders
107         return end - start
108
109
110 class TyperMentions(Mentions, List[TyperMention]):
111     """List of mentions, inherits from list
112     """
113     @classmethod
114     def charge(cls, mentions: Mentions):
115         return cls([
116             TyperMention(m.mention, m.left_context,
117                          m.right_context) for m in mentions
118         ])
119
120     @property
121     def tokens(self) -> List[List[str]]:
```

```
122         return [c.tokens for c in self]
123
124     @property
125     def tokens_lengths(self) -> List[int]:
126         return [c.tokens_length for c in self]
127
128
129 # ELMo
130 ELMO_OPTIONS_FILE = """https://s3-us-west-2.amazonaws.com
131 /allennlp/models/elmo/2x4096_512_2048cnn_2xhighway/
132 elmo_2x4096_512_2048cnn_2xhighway_options.json"""
133 ELMO_WEIGHT_FILE = """https://s3-us-west-2.amazonaws.com/
134 allennlp/models/elmo/2x4096_512_2048cnn_2xhighway/
135 elmo_2x4096_512_2048cnn_2xhighway_weights.hdf5"""
136 ELMO_EMBEDDINGS_DIM = 1024
137
138
139 class ELMoPretrainedEmbedder():
140     """ Get ELMo Embeddings
141     """
142     def __init__(self):
143         self.embedder = Elmo(ELMO_OPTIONS_FILE,
144                              ELMO_WEIGHT_FILE, 3)
145
146     def embed(self, mentions: TyperMentions):
147         """ Embed mentions to sentence- and mention
148             embeddings.
149
150         Args:
151             mentions (Mentions): List of mentions
152
153         Returns:
154             EmbeddedBatch: Mention and sentence
155             embeddings (grouped in own datatype wrapper)
156
157         Dims:
158             - EmbeddedBatch.mention_embeddings.shape =
159                 `(batch_size, 3, max_mention_tokens_length,
160                 ELMO_EMBEDDINGS_DIM)`
161             - EmbeddedBatch.sentence_embeddings.shape =
162                 `(batch_size, 3, max_tokens_length,
```

```
163                 ELMO_EMBEDDINGS_DIM)`
164
165             with:
166             - `batch_size = Number of passed mentions`,
167             - `max_tokens_length = Length of the longest
168                 mention sentence`,
169             - `max_mention_tokens_length = Length of the
170                 longest mention` and
171             - `ELMO_EMBEDDINGS_DIM = Output dim from
172                 ELMo embedder, defined by loaded options
173                 and weights`
174         """
175         bsz = len(mentions)
176         token_ids = batch_to_ids(mentions.tokens)
177         embs = self.embedder(
178             token_ids)['elmo_representations']
179
180         # Sentence Embeddings
181         sentence_embeddings = torch.stack(embs).permute(
182             1, 0, 2, 3)
183
184         # Mention Embeddings
185         max_mention_tokens_length = max([
186             mention.mention_length for mention in mentions
187         ])
188         mention_embeddings = torch.zeros([
189             bsz, 3, max_mention_tokens_length,
190             ELMO_EMBEDDINGS_DIM
191         ])
192         for i, mention in enumerate(mentions):
193             start_ind, end_ind = mention.borders
194             mention_length = end_ind - start_ind
195             mention_embeddings[
196                 i, :, :
197                 mention_length, :] = sentence_embeddings[
198                     i, :, start_ind:end_ind, :]
199
200         return EmbeddedBatch(mention_embeddings,
201                              sentence_embeddings)
202
203
```

```
204 PATH_TO_CHARDICT = path.normpath(
205     path.join(path.dirname(__file__),
206               "ontology/char_vocab.english.txt"))
207
208
209 class SentenceEncoder(nn.Module):
210     """
211         Encode Sentence
212           - Get word lokation tokens
213           - concat sentence embeddings with lokation tokens
214           - fed into bi lstm (with dim_hid)
215           - span attention
216           - s = Attention(bi-LSTM([s'; l]))
217     """
218     def __init__(self, dropout_rate, rnn_dim,
219                  embeddings_dim, mask_dim, attention_dim):
220         super().__init__()
221
222         # Define dims
223         self.rnn_dim = rnn_dim
224         # Must be same as ELMo output dim (1024)
225         self.embeddings_dim = embeddings_dim
226         self.mask_dim = mask_dim
227         self.combined_dim = self.embeddings_dim + self.mask_dim
228         self.attention_dim = attention_dim
229
230         # Define networks
231         self.weighted_sum = ELMoWeightedSum()
232         self.location_LNN = nn.Linear(4, self.mask_dim)
233         self.input_dropout = nn.Dropout(dropout_rate)
234         self.bi_lstm = BiLSTM(self.combined_dim,
235                               self.rnn_dim)
236         self.attentive_sum = SelfAttentiveSum(
237             self.rnn_dim * 2, self.attention_dim)
238
239     def get_location_tokens(self, mentions: TyperMentions,
240                             device) -> torch.Tensor:
241         """Get location tokens:
242         Each word is assigned one of four location
243             tokens, based on whether
244         - (1) the word is in the left context,
```

```
245         - (2) the word is the first word of the
246             mention span,
247         - (3) the word is in the mention span
248             (but not first), and
249         - (4) the word is in the right context.
250 
251         Returns:
252             Tensor: Location tokens.
253                 Dim - (batch_size, max_seq_length, 4)
254         """
255         max_seq_length = max(mentions.tokens_lengths)
256         bsz = len(mentions)
257         location_tokens = torch.zeros(
258             [bsz, max_seq_length, 4], device=device)
259         for i, mention in enumerate(mentions):
260             start_ind, end_ind = mention.borders
261 
262             location_tokens[i, :start_ind, 0] = 1.0
263             location_tokens[i, start_ind, 1] = 1.0
264             location_tokens[i, start_ind + 1:end_ind,
265                             2] = 1.0
266             location_tokens[i,
267                             end_ind:mention.tokens_length,
268                             3] = 1.0
269         return location_tokens
270 
271     def forward(self, mentions: TyperMentions,
272                 embeddings: EmbeddedBatch):
273         """Get sentence representations.
274 
275         Args:
276             mentions (TyperMentions): List of mentions
277             embeddings (EmbeddedBatch): Embeddings
278                 from ELMo embedder
279 
280         Returns:
281             Tensor: Sequence representation.
282                 Dim - (batch_size, 2*rnn_dim)
283         """
284         # embeddings.sentence_embeddings:
285         # torch.Size([bsz, 3, maximum_sentence_length,
```

```
286             #                   embeddings_dim])
287             weighted_embeddings = self.weighted_sum(
288                 embeddings.sentence_embeddings)
289             # weighted_embeddings:
290             # torch.Size([bsz, maximum_sentence_length,
291             #                   embeddings_dim])
292
293             location_tokens = self.get_location_tokens(
294                 mentions,
295                 embeddings.sentence_embeddings.device).view(
296                     -1, 4)
297             # location_tokens:
298             # torch.Size([bsz*maximum_sentence_length, 4])
299             location_mask = self.location_LNN(location_tokens)
300             # location_mask:
301             # torch.Size([bsz*maximum_sentence_length,
302             #                   mask_dim])
303             location_mask = location_mask.view(
304                 weighted_embeddings.size()[0], -1,
305                 self.mask_dim)
306             # location_mask:
307             # torch.Size([bsz, maximum_sentence_length,
308             #                   mask_dim])
309
310             weighted_embeddings = torch.cat(
311                 (weighted_embeddings, location_mask), 2)
312             # weighted_embeddings:
313             # torch.Size([bsz, maximum_sentence_length,
314             #                   combined_dim])
315             weighted_embeddings = self.input_dropout(
316                 weighted_embeddings)
317
318             sequence_lengths = torch.tensor(
319                 mentions.tokens_lengths,
320                 device=embeddings.sentence_embeddings.device)
321
322             sequence_rep = self.bi_lstm(weighted_embeddings,
323                                         sequence_lengths)
324             # sequence_rep:
325             # torch.Size([bsz, maximum_sentence_length,
326             #                   2*rnn_dim])
```

```
327         sequence_rep = self.attentive_sum(sequence_rep)
328         # sequence_rep: torch.Size([bsz, 2*rnn_dim])
329         return sequence_rep
330
331
332 class MentionEncoder(nn.Module):
333     """
334         Encode Mention (word level)
335             - m' fed into bi lstm (with dim_hid)
336             - concat hidden states of both directions
337             - summed by span attention
338             - m_word = Attention(bi-LSTM([m'; l]))
339         Encode Mention (character level)
340             - characters get embedded and
341                 fed into 1-D convolution
342             - m_char = CNN(mention_chars)
343     """
344     def __init__(self, dropout_rate, cnn_dim,
345                  embeddings_dim, attention_dim):
346         super().__init__()
347
348         # Define dimensions
349         self.cnn_dim = cnn_dim
350         # Must be same as ELMo output dim (1024)
351         self.embeddings_dim = embeddings_dim
352         self.attention_dim = attention_dim
353
354         # Define networks
355         self.weighted_sum = ELMoWeightedSum()
356         self.input_dropout = nn.Dropout(dropout_rate)
357         self.bi_lstm = BiLSTM(self.embeddings_dim,
358                               self.embeddings_dim // 2)
359         self.attentive_sum = SelfAttentiveSum(
360             self.embeddings_dim, self.attention_dim)
361         self.cnn = CNN(self.cnn_dim)
362
363         # Load char dictionary
364         self.char_dict = defaultdict(int)
365         char_vocab = [u"<unk>"]
366         with open(PATH_TO_CHARDICT, encoding="utf-8") as f:
367             char_vocab.extend(c.strip()
```

```
                                for c in f.readlines())
            self.char_dict.update(
                {c: i
                 for i, c in enumerate(char_vocab)})

    @staticmethod
    def pad_slice(seq, seq_length, pad_token="<none>"):
        """Fills a sequence with a pad_token until
            it reached the desire length
        """
        return seq + ([pad_token] * (seq_length - len(seq)))

    def get_mention_characters(self,
                               mentions: TyperMentions,
                               device):
        """Gets characters from mention, padded to
            longest mention length
        """
        mentions_characters = [[
            self.char_dict[x] for x in list(mention.mention)
        ] for mention in mentions]
        # max(..., 5): 5 because CNN uses 5 as kernel
        # size in Conv1d
        max_span_chars = max(
            max(
                len(characters)
                for characters in mentions_characters), 5)
        mentions_characters = [
            self.pad_slice(characters,
                           max_span_chars,
                           pad_token=0)
            for characters in mentions_characters
        ]
        span_chars = torch.tensor(mentions_characters,
                                  dtype=torch.int64,
                                  device=device)
        # span_chars: torch.Size([bsz, max_span_chars])
        return span_chars

    def forward(self, mentions: TyperMentions,
                embeddings: EmbeddedBatch):
```

```
409         # embeddings.mention_embeddings:
410         # torch.Size([bsz, 3, maximum_mention_length,
411         #                 embeddings_dim])
412         weighted_embeddings = self.weighted_sum(
413             embeddings.mention_embeddings)
414         # weighted_embeddings:
415         # torch.Size([bsz, maximum_mention_length,
416         #                 embeddings_dim])
417         weighted_embeddings = self.input_dropout(
418             weighted_embeddings)
419
420         mention_lengths = torch.tensor(
421             [
422                 mention.mention.count(" ") + 1
423                 for mention in mentions
424             ],
425             device=embeddings.mention_embeddings.device)
426
427         mention_word = self.bi_lstm(weighted_embeddings,
428                                     mention_lengths)
429         # mention_word:
430         # torch.Size([bsz, maximum_mention_length,
431         #                 embeddings_dim])
432         mention_word = self.attentive_sum(mention_word)
433         # mention_word: torch.Size([bsz, embeddings_dim])
434
435         mention_chars = self.get_mention_characters(
436             mentions, mention_word.device)
437         # mention_chars:
438         # torch.Size([bsz, maximum_mention_length])
439         mention_chars = self.cnn(mention_chars)
440         # mention_chars: torch.Size([bsz, cnn_dim])
441         mention_rep = torch.cat(
442             (mention_word, mention_chars), 1)
443         # mention_rep:
444         # torch.Size([bsz, embeddings_dim + cnn_dim])
445         return mention_rep
446
447
448 class SimpleDecoder(nn.Module):
449     def __init__(self, output_dim, answer_num):
```

```
450         super().__init__()
451         self.linear = nn.Linear(output_dim,
452                                 answer_num,
453                                 bias=False)
454
455     def forward(self, inputs):
456         output_embed = self.linear(inputs)
457         return output_embed
458
459
460 # Borrowed from AllenNLP
461 def sort_batch_by_length(
462         tensor: torch.autograd.Variable,
463         sequence_lengths: torch.autograd.Variable):
464     """
465     @ from allennlp
466     Sort a batch first tensor by some specified lengths.
467
468     Parameters
469     ----------
470     tensor : Variable(torch.FloatTensor), required.
471         A batch first Pytorch tensor.
472     sequence_lengths :
473         Variable(torch.LongTensor), required.
474         A tensor representing the lengths of
475         some dimension of the tensor which
476         we want to sort by.
477
478     Returns
479     -------
480     sorted_tensor : Variable(torch.FloatTensor)
481         The original tensor sorted along the batch
482         dimension with respect to sequence_lengths.
483     sorted_sequence_lengths : Variable(torch.LongTensor)
484         The original sequence_lengths sorted by
485         decreasing size.
486     restoration_indices : Variable(torch.LongTensor)
487         Indices into the sorted_tensor such that
488         ``sorted_tensor.index_select(0,
489             restoration_indices) == original_tensor``
490     """
```

```
491
492     if not isinstance(tensor, Variable) or not isinstance(
493             sequence_lengths, Variable):
494         raise ValueError(
495             """Both the tensor and sequence lengths
496                 must be torch.autograd.Variables."""
497         )
498
499     (sorted_sequence_lengths,
500     permutation_index) = sequence_lengths.sort(
501         0, descending=True)
502     sorted_tensor = tensor.index_select(0,
503                                         permutation_index)
504     # This is ugly, but required - we are creating a
505     # new variable at runtime, so we
506     # must ensure it has the correct CUDA vs non-CUDA
507     # type. We do this by cloning and
508     # refilling one of the inputs to the function.
509     index_range = sequence_lengths.data.clone().copy_(
510         torch.arange(0, len(sequence_lengths)))
511     # This is the equivalent of zipping with index,
512     # sorting by the original
513     # sequence lengths and returning the now sorted indices.
514     index_range = Variable(index_range.long())
515     _, reverse_mapping = permutation_index.sort(
516         0, descending=False)
517     restoration_indices = index_range.index_select(
518         0, reverse_mapping)
519     return (sorted_tensor, sorted_sequence_lengths,
520             restoration_indices)
521
522
523 class ELMoWeightedSum(nn.Module):
524     def __init__(self):
525         super(ELMoWeightedSum, self).__init__()
526         self.gamma = nn.Parameter(torch.randn(1))
527         self.S = nn.Parameter(torch.randn(1, 3))
528         self.softmax = nn.Softmax(dim=1)
529
530     def forward(self, x):
531         """
```

```
532         x: ELMo vectors of (batch size, 3, 1024) or
533         (batch size, 3, seq len, 1024).
534         """
535         S = self.softmax(self.S)  # normalize
536         if x.dim() == 3:
537             batch_size, n_layers, emb_dim = x.shape
538             x = x.permute(0, 2, 1).contiguous().view(
539                 -1, 3)  # (batch_size*1024, 3)
540             x = (x * S).sum(
541                 1) * self.gamma  # (batch_size*1024, 1)
542             x = x.view(batch_size,
543                        emb_dim)  # (batch_size, 1024)
544         elif x.dim() == 4:
545             batch_size, n_layers, seq_len, emb_dim = x.shape
546             x = x.permute(0, 2, 3, 1).contiguous().view(
547                 -1, 3)  # (batch_size*seq_len*1024, 3)
548             x = (x * S).sum(
549                 1
550             ) * self.gamma  # (batch_size*seq_len*1024, 1)
551             x = x.view(
552                 batch_size, seq_len,
553                 emb_dim)  # (batch_size, seq_len, 1024)
554         else:
555             print('Wrong input dimension: x.dim() = ' +
556                   repr(x.dim()))
557             raise ValueError
558         return x
559
560
561 class BiLSTM(nn.Module):
562     def __init__(self, embeddings_dim, rnn_dim):
563         super().__init__()
564         self.lstm = nn.LSTM(embeddings_dim,
565                             rnn_dim,
566                             bidirectional=True,
567                             batch_first=True)
568
569     def forward(self, weighted_embeddings,
570                 sequence_lengths):
571         (sorted_inputs,
572         sorted_sequence_lengths,
```

```
573         restoration_indices) = sort_batch_by_length(
574             weighted_embeddings, sequence_lengths)
575         packed_sequence_input = pack_padded_sequence(
576             sorted_inputs,
577             sorted_sequence_lengths.data.long().tolist(),
578             batch_first=True)
579         packed_sequence_output, _ = self.lstm(
580             packed_sequence_input, None)
581         unpacked_sequence_tensor, _ = pad_packed_sequence(
582             packed_sequence_output, batch_first=True)
583         context_rep = unpacked_sequence_tensor.index_select(
584             0, restoration_indices)
585         return context_rep
586 
587 
588 class SelfAttentiveSum(nn.Module):
589     """
590     Attention mechanism to get a weighted sum of
591     RNN output sequence to a single RNN output dimension.
592     """
593     def __init__(self, output_dim, hidden_dim):
594         super(SelfAttentiveSum, self).__init__()
595         self.key_maker = nn.Linear(output_dim,
596                                    hidden_dim,
597                                    bias=False)
598         self.key_rel = nn.ReLU()
599         self.hidden_dim = hidden_dim
600         self.key_output = nn.Linear(hidden_dim,
601                                     1,
602                                     bias=False)
603         self.key_softmax = nn.Softmax()
604 
605     def _masked_softmax(self, X, mask=None, alpha=1e-20):
606         # X, (batch_size, seq_length)
607         X_max = torch.max(X, dim=1, keepdim=True)[0]
608         X_exp = torch.exp(X - X_max)
609         if mask is None:
610             mask = (X != 0).float()
611         X_exp = X_exp * mask
612         X_softmax = X_exp / (
613             torch.sum(X_exp, dim=1, keepdim=True) + alpha)
```

```
614         return X_softmax
615
616     def forward(self, input_embed):
617         mask = (input_embed[:, :, 0] != 0).float()
618         input_embed_squeezed = input_embed.view(
619             -1,
620             input_embed.size()[2])
621         k_d = self.key_maker(input_embed_squeezed)
622         k_d = self.key_rel(k_d)  # this leads all zeros
623         if self.hidden_dim == 1:
624             k = k_d.view(input_embed.size()[0], -1)
625         else:
626             k = self.key_output(k_d).view(
627                 input_embed.size()[0],
628                 -1)  # (batch_size, seq_length)
629         weighted_keys = self._masked_softmax(
630             k, mask=mask).view(input_embed.size()[0], -1, 1)
631         weighted_values = torch.sum(
632             weighted_keys * input_embed,
633             1)  # batch_size, seq_length, embed_dim
634         return weighted_values
635
636
637 class CNN(nn.Module):
638     def __init__(self, output_dim):
639         super(CNN, self).__init__()
640         self.conv1d = nn.Conv1d(
641             100, output_dim,
642             5)  # input, output, filter_number
643         self.char_W = nn.Embedding(115, 100)
644
645     def forward(self, span_chars):
646         # [batch_size, char_embedding, max_char_seq]
647         char_embed = self.char_W(span_chars).transpose(
648             1,
649             2)
650         # list of [batch_size, filter_dim, max_char_seq,
651         #             filter_number]
652         conv_output = [
653             self.conv1d(char_embed)
654         ]
```

```
655         # batch_size, filter_dim, max_char_seq, filter_num
656         conv_output = [
657             F.relu(c) for c in conv_output
658         ]
659         # batch_size, filter_dim, 1, filter_num
660         cnn_rep = [
661             F.max_pool1d(i, i.size(2)) for i in conv_output
662         ]
663         cnn_output = torch.squeeze(
664             torch.cat(cnn_rep, 1),
665             2)  # batch_size, filter_num * filter_dim, 1
666         return cnn_output
667
668
669 PATH_TO_VOCAB = path.normpath(
670     path.join(path.dirname(__file__),
671               "ontology/conll_categories.txt"))
672
673
674 class FineGrainedEntityTyper(pl.LightningModule):
675     def __init__(self,
676                  learning_rate=2e-3,
677                  mention_dropout=0.5,
678                  input_dropout=0.5,
679                  rnn_dim=50,
680                  cnn_dim=50,
681                  mask_dim=50,
682                  attention_dim=100,
683                  answer_num=60000,
684                  threshold=0.5,
685                  **kwargs):
686         super().__init__(**kwargs)
687         self.save_hyperparameters()
688
689         embeddings_dim = ELMO_EMBEDDINGS_DIM
690         output_dim = 2 * rnn_dim + embeddings_dim + cnn_dim
691
692         self.embedder = ELMoPretrainedEmbedder()
693         self.sentence_encoder = SentenceEncoder(
694             input_dropout, rnn_dim, embeddings_dim,
695             mask_dim, attention_dim)
```

```
696         self.mention_encoder = MentionEncoder(
697             mention_dropout, cnn_dim, embeddings_dim,
698             attention_dim)
699         self.decoder = SimpleDecoder(output_dim, answer_num)
700 
701         self.answer2id = load_vocab_dict(
702             PATH_TO_VOCAB, vocab_max_size=answer_num)
703         self.id2answer = {
704             v: k
705             for k, v in self.answer2id.items()
706         }
707 
708         self.f1 = torchmetrics.F1(num_classes=answer_num,
709                                   threshold=threshold,
710                                   average="macro")
711 
712     @staticmethod
713     def add_model_specific_args(parent_parser):
714         parser = parent_parser.add_argument_group(
715             "FineGrainedEntityTyper")
716         parser.add_argument("--mention-dropout",
717                             type=float,
718                             default=0.5)
719         parser.add_argument("--input-dropout",
720                             type=float,
721                             default=0.5)
722         parser.add_argument("--rnn-dim",
723                             type=int,
724                             default=50)
725         parser.add_argument("--cnn-dim",
726                             type=int,
727                             default=50)
728         parser.add_argument("--mask-dim",
729                             type=int,
730                             default=50)
731         parser.add_argument("--attention-dim",
732                             type=int,
733                             default=100)
734         parser.add_argument("--answer-num",
735                             type=int,
736                             default=60000)
```

```
737         parser.add_argument("--threshold",
738                             type=float,
739                             default=0.5)
740         return parent_parser
741
742     def _get_logits(self, mentions: Mentions):
743         """
744         1. Get ELMo Embeddings
745         2. Encode Sentence
746         3. Encode Mention (word level and char level)
747         4. Concat and decode all three vectors
748             v = [s, m_word, m_char]
749         """
750         mentions = TyperMentions.charge(mentions)
751         embeddings = self.embedder.embed(mentions)
752         sequence_rep = self.sentence_encoder(
753             mentions, embeddings)
754         mention_rep = self.mention_encoder(
755             mentions, embeddings)
756
757         representation = torch.cat(
758             (sequence_rep, mention_rep), 1)
759         logits = self.decoder(representation)
760         return logits
761
762     def forward(self, mentions: Mentions):
763         mentions = TyperMentions.charge(mentions)
764         logits = self._get_logits(mentions)
765         outputs = torch.sigmoid(logits)
766
767         # Decode predictions to categories
768         predictions: List[List[Tuple[str, float]]] = []
769         for output in outputs:
770             output_indices = (
771                 output >
772                 self.hparams.threshold).nonzero().squeeze(1)
773             if len(output_indices) == 0:
774                 output_indices = torch.argmax(output,
775                                               dim=0,
776                                               keepdim=True)
777             predicted_categories = [
```

```
778                 (self.id2answer[i.item()], output[i].item())
779                 for i in output_indices
780             ]
781             predictions.append(predicted_categories)
782         return predictions
783 
784     def on_train_start(self):
785         self.logger.log_hyperparams(self.hparams)
786 
787     def training_step(self, batch, batch_idx):
788         mentions, categories_batch = batch
789 
790         # Encode target categories to target tensor
791         targets = torch.zeros([
792             len(categories_batch), self.hparams.answer_num
793         ],
794                               device=self.device)
795         for i, categories in enumerate(categories_batch):
796             answer_ids = [
797                 self.answer2id[c] for c in categories
798                 if c in self.answer2id
799             ]
800             for answer_idx in answer_ids:
801                 targets[i, answer_idx] = 1
802 
803         logits = self._get_logits(mentions)
804         loss = F.binary_cross_entropy_with_logits(
805             logits, targets)
806 
807         self.log("train_loss", loss)
808         return loss
809 
810     def _eval_step(self, batch):
811         mentions, categories_batch = batch
812 
813         # Encode target categories to target tensor
814         targets = torch.zeros([
815             len(categories_batch), self.hparams.answer_num
816         ],
817                               dtype=torch.int,
818                               device=self.device)
```

```
819         for i, categories in enumerate(categories_batch):
820             answer_ids = [
821                 self.answer2id[c] for c in categories
822                 if c in self.answer2id
823             ]
824             for answer_idx in answer_ids:
825                 targets[i, answer_idx] = 1
826 
827         logits = self._get_logits(mentions)
828         outputs = torch.sigmoid(logits)
829         self.f1(outputs, targets)
830 
831     def validation_step(self, batch, batch_idx):
832         self._eval_step(batch)
833 
834     def validation_epoch_end(self, results):
835         accuracy = self.f1.compute()
836         self.log("validation_accuracy",
837                  accuracy,
838                  prog_bar=True)
839         self.f1.reset()
840 
841     def test_step(self, batch, batch_idx):
842         self._eval_step(batch)
843 
844     def test_epoch_end(self, results):
845         accuracy = self.f1.compute()
846         self.log("test_accuracy", accuracy, prog_bar=True)
847         self.f1.reset()
848 
849     def configure_optimizers(self):
850         optimizer = torch.optim.Adam(
851             self.parameters(),
852             lr=self.hparams.learning_rate)
853         return optimizer
```

Anhang 3: PyTorch Lightning Implementierung des Candidate-Rankers

```
from dataclasses import dataclass
from functools import lru_cache
from typing import List, Tuple, Union

import pytorch_lightning as pl
import torch
import torch.nn as nn
import torchmetrics
from dataclasses_json import dataclass_json
from torch import nn
from torch.nn import functional as F
from transformers import (AdamW, BertConfig, BertModel,
                          BertTokenizerFast)
from transformers.optimization import (
    get_linear_schedule_with_warmup)

PROPERTIES = [
    ('cites work', 'P2860'), ('series ordinal', 'P1545'),
    ('author name string', 'P2093'), ('instance of', 'P31'),
    ('stated in', 'P248'), ('retrieved', 'P813'),
    ('reference URL', 'P854'), ('PubMed ID', 'P698'),
    ('title', 'P1476'), ('publication date', 'P577'),
    ('published in', 'P1433'), ('page(s)', 'P304'),
    ('volume', 'P478'), ('apparent magnitude', 'P1215'),
    ('astronomical filter', 'P1227'), ('issue', 'P433'),
    ('catalog code', 'P528'), ('DOI', 'P356'),
    ('catalog', 'P972'), ('author', 'P50'),
    ('main subject', 'P921'),
    ('language of work or name', 'P407'),
    ('country', 'P17'), ('PMCID', 'P932'), ('of', 'P642'),
    ('located in the administrative territorial entity',
     'P131'), ('proper motion', 'P2215'),
    ('point in time', 'P585'), ('stated as', 'P1932'),
    ('determination method', 'P459'),
    ('occupation', 'P106'), ('coordinate location', 'P625'),
    ('SIMBAD ID', 'P3083'), ('right ascension', 'P6257'),
    ('declination', 'P6258'), ('epoch', 'P6259'),
    ('sex or gender', 'P21'),
```

```
39      ('Google Knowledge Graph ID', 'P2671'),
40      ('constellation', 'P59'), ('start time', 'P580'),
41      ('found in taxon', 'P703'),
42      ('based on heuristic', 'P887'), ('given name', 'P735'),
43      ('VIAF ID', 'P214'), ('parallax', 'P2214'),
44      ('date of birth', 'P569'),
45      ('Wikimedia import URL', 'P4656'),
46      ('imported from Wikimedia project', 'P143'),
47      ('named as', 'P1810'), ('radial velocity', 'P2216'),
48      ('ResearchGate publication ID', 'P5875'),
49      ('Freebase ID', 'P646'), ('ortholog', 'P684'),
50      ('part of', 'P361'), ('country of citizenship', 'P27'),
51      ('image', 'P18'), ('end time', 'P582'),
52      ('GeoNames ID', 'P1566'),
53      ('distance from Earth', 'P2583'),
54      ('family name', 'P734'), ('parent taxon', 'P171'),
55      ('taxon name', 'P225'), ('taxon rank', 'P105'),
56      ('chromosome', 'P1057'), ('Commons category', 'P373'),
57      ('exact match', 'P2888'), ('subclass of', 'P279'),
58      ('Entrez Gene ID', 'P351'), ('collection', 'P195'),
59      ('place of birth', 'P19'),
60      ('GNS Unique Feature ID', 'P2326'),
61      ('date of death', 'P570'),
62      ('described by source', 'P1343'), ('Elo rating',
63                                         'P1087'),
64      ('GBIF taxon ID', 'P846'), ('location', 'P276'),
65      ('UniProt protein ID', 'P352'), ('inception', 'P571'),
66      ('ORCID iD', 'P496'), ('educated at', 'P69'),
67      ('category combines topics', 'P971'),
68      ('languages spoken, written or signed', 'P1412'),
69      ('applies to jurisdiction', 'P1001'), ('GND ID',
70                                           'P227'),
71      ('heritage designation',
72       'P1435'), ('located in time zone', 'P421'),
73      ('postal code', 'P281'), ('population', 'P1082'),
74      ('sport', 'P641'), ('has part', 'P527'),
75      ('WorldCat Identities ID', 'P7859'), ('follows',
76                                           'P155'),
77      ('followed by', 'P156'), ('copyright status', 'P6216'),
78      ('pronunciation', 'P7243'),
79      ('full work available at URL', 'P953'),
```

```
80      ('official name', 'P1448'),
81      ('member of sports team', 'P54'),
82      ('official website', 'P856'), ('employer', 'P108')
83  ]
84
85
86  @dataclass_json
87  @dataclass(frozen=True)
88  class Mention():
89      """Mention type which wraps a mention and its context
90          into one single class
91
92      Properties:
93      `mention`: `str` The mention in the text
94      `left_context`: `str` The context before the mention
95      `right_context`: `str` The context after the mention
96      `sequence`: `str` The complete sentence: left_context
97                  + mention + right_context
98      """
99
100     mention: str
101     left_context: str
102     right_context: str
103
104     def __repr__(self) -> str:
105         return f"Mention({self.mention})"
106
107     @property
108     @lru_cache()
109     def sequence(self) -> str:
110         s = (self.left_context + " " + self.mention +
111              " " + self.right_context)
112         return s
113
114
115 class Mentions(List[Mention]):
116     """List of mentions, inherits from list
117
118     """
119     def __repr__(self) -> str:
120         return f"Mentions({', '.join([m.mention for m in self])})"
```

```
121
122     @property
123     def sequences(self) -> List[str]:
124         return [c.sequence for c in self]
125
126     def dumps(self):
127         return Mention.schema().dumps(self, many=True)
128
129     def to_dict(self):
130         return Mention.schema().dump(self, many=True)
131
132     @classmethod
133     def loads(cls, arr: list):
134         return cls(Mention.schema().loads(arr, many=True))
135
136     @classmethod
137     def from_dict(cls, arr: list):
138         return cls(Mention.schema().load(arr, many=True))
139
140
141 @dataclass_json
142 @dataclass
143 class Entity():
144     label: str
145     attributes: dict
146     score: int = None
147
148     def __repr__(self) -> str:
149         if self.score:
150             return f"Entity({self.label})"
151         else:
152             return f"Entity({self.label}: {self.score})"
153
154
155 class Entities(List[Entity]):
156     """List of entities, inherits from list
157
158     """
159     def __repr__(self) -> str:
160         return f"Entities({', '.join(self.labels)})"
161
```

```
162     @property
163     def labels(self):
164         return [e.label for e in self]
165 
166     def dumps(self):
167         return Entity.schema().dumps(self, many=True)
168 
169     def to_dict(self):
170         return Entity.schema().dump(self, many=True)
171 
172     @classmethod
173     def loads(cls, arr: list):
174         return cls(Entity.schema().loads(arr, many=True))
175 
176     @classmethod
177     def from_dict(cls, arr: list):
178         return cls(Entity.schema().load(arr, many=True))
179 
180 
181 class Attribute2Text:
182     @classmethod
183     def concatination(cls, attributes: Union[dict,
184                                              list]) -> str:
185         """Concat attributes with just spaces
186         """
187         if type(attributes) == list:
188             return " ".join(attributes)
189 
190         c = ""
191         for value in attributes.values():
192             if type(value) in [str, int, float]:
193                 c += value
194             if type(value) == list:
195                 c += " ".join(value)
196             c += " "
197         return c.strip()
198 
199     @classmethod
200     def sep_separation(cls,
201                        attributes: Union[dict, list],
202                        special_token="[ATT]") -> str:
```

```
203         """Concat attributes with a `special_token`
204             (default "[ATT]")
205         """
206         if type(attributes) == list:
207             return "[ATT] " + " [ATT] ".join(attributes)
208 
209         c = ""
210         for value in attributes.values():
211             if type(value) in [str, int, float]:
212                 c += f"{special_token} {value}"
213             if type(value) == list:
214                 c += " ".join(f"{special_token} {v}"
215                               for v in value)
216             c += " "
217         return c.strip()
218 
219     @classmethod
220     def attribute_separation(cls, attributes: dict) -> str:
221         """Concat attributes based on attribtue key
222         """
223         c = ""
224         for key, value in attributes.items():
225             if type(value) in [str, int, float]:
226                 c += f"[{key.upper()}] {value}"
227             if type(value) == list:
228                 c += " ".join(f"[{key.upper()}] {v}"
229                               for v in value)
230             c += " "
231         return c.strip()
232 
233 
234 def init_bert() -> Tuple[BertTokenizerFast, BertModel]:
235     special_tokens = ["[MS]", "[ME]", "[ATT]"] + [
236         f"[{prop[0].upper()}]" for prop in PROPERTIES
237     ]
238     tokenizer = BertTokenizerFast.from_pretrained(
239         "bert-base-uncased",
240         additional_special_tokens=special_tokens)
241     config = BertConfig.from_pretrained("bert-base-uncased")
242     model = BertModel.from_pretrained("bert-base-uncased",
243                                       config=config)
```

```
244     model.resize_token_embeddings(len(tokenizer))
245     return tokenizer, model
246
247
248 def replace_numbers(s):
249     return " ".join([
250         "<number>" if w.lower().isnumeric() else w
251         for w in s.split()
252     ])
253
254
255 def transform_mention_categories(mention_categories):
256     res = []
257     for categories in mention_categories:
258         categories.sort(key=lambda x: x[1])
259         labels = [cat[0] for cat in categories][:10]
260         res.append(labels)
261     return res
262
263
264 def find_target_index(target: Entity, candidates: Entities):
265     return next(
266         (i for i, candidate in enumerate(candidates)
267          if candidate.label == target.label),
268         -1,
269     )
270
271
272 class AccelEncoder(nn.Module):
273     """Creates embeddings of a combination of mention and entity,
274         uses an entity-typer for that
275     """
276     def __init__(self) -> None:
277         super().__init__()
278         self.tokenizer, self.bert = init_bert()
279         self.device = "cpu"
280
281     def to(self, device):
282         super().to(device)
283         self.device = device
284
```

```
285     def encode_targets(self, candidates: List[Entities],
286                        targets: Entities):
287
288         bsz = len(targets)
289         csz = max(len(cand) for cand in candidates)
290         t = torch.zeros([bsz, csz],
291                         dtype=torch.float,
292                         device=self.device)
293         for i, (target,
294                 cands) in enumerate(zip(targets,
295                                         candidates)):
296             target_index = find_target_index(target, cands)
297             if target_index == -1:
298                 continue
299             t[i, target_index] = 1.0
300         return t
301
302     def embed(
303             self, mentions: Mentions,
304             category_labels: List[List[str]],
305             candidates: List[Entities]) -> List[List[str]]:
306         embeddings = []
307         for (mention, categories,
308              cands) in zip(mentions, category_labels,
309                            candidates):
310             embs = []
311             for candidate in cands:
312                 m_seps = Attribute2Text.sep_separation(
313                     categories)
314                 m_embedding = f"{mention.mention} {m_seps}"
315                 c_seps = Attribute2Text.attribute_separation(
316                     candidate.attributes)
317                 c_embedding = f"{candidate.label} {c_seps}"
318                 embedding = ("[CLS]" + m_embedding + "[SEP]"
319                              + c_embedding + "[SEP]")
320                 #embedding = replace_numbers(embedding)
321                 embs.append(embedding)
322             embeddings.append(embs)
323         return embeddings
324
325     def encode(self,
```

```
326                 batch: List[List[str]]) -> torch.Tensor:
327         bsz = max(len(e) for e in batch)
328         encodings = torch.zeros(len(batch),
329                                 bsz,
330                                 768,
331                                 device=self.device)
332         for i, embeds in enumerate(batch):
333             token_ids = self.tokenizer(
334                 embeds,
335                 return_tensors="pt",
336                 add_special_tokens=False,
337                 padding=True,
338                 truncation=True,
339                 max_length=128,
340             )
341             token_ids.to(self.device)
342             output = self.bert(**token_ids).pooler_output
343             encodings[i, :len(embeds), :] = output
344         return encodings
345 
346     def forward(self, mention: Mentions,
347                 mention_categories: List[List[str]],
348                 candidates: List[Entities]):
349         category_labels = transform_mention_categories(
350             mention_categories)
351         embedded = self.embed(mention, category_labels,
352                               candidates)
353         encodings = self.encode(embedded)
354         return encodings
355 
356 
357 class AccelDecoder(nn.Module):
358     def __init__(self) -> None:
359         super().__init__()
360         self.weight = nn.Linear(768, 1)
361 
362     def to(self, device):
363         super().to(device)
364         self.weight.to(device)
365 
366     def forward(self, encodings: torch.Tensor):
```

```
367         logits = self.weight(encodings).squeeze(dim=2)
368         return logits
369
370
371 class AccelRanker(pl.LightningDataModule):
372     def __init__(self, typer, learning_rate=2e-5, **kwargs):
373         super().__init__()
374         self.save_hyperparameters("learning_rate")
375
376         self.typer = typer
377
378         # Binary Problem
379         self.accuracy = torchmetrics.Accuracy()
380         self.train_accuracy = torchmetrics.Accuracy()
381
382         self.encoder = AccelEncoder()
383         self.decoder = AccelDecoder()
384         self.m = nn.Sigmoid()
385
386     def to(self, *args, **kwargs):
387         out = torch._C._nn._parse_to(*args, **kwargs)
388         self.encoder.to(device=out[0])
389         self.decoder.to(device=out[0])
390         return super().to(*args, **kwargs)
391
392     def _get_logits(self, mentions: Mentions,
393                     candidates: List[Entities]):
394         """
395         1. Get mention types
396         2. Encode Mention + Entities
397         3. Get score
398         """
399         with torch.no_grad():
400             mention_categories: List[List[Tuple[
401                 str, float]]] = self.typer(mentions)
402         encodings = self.encoder(mentions,
403                                  mention_categories,
404                                  candidates)
405         logits = self.decoder(encodings)
406         return logits
407
```

```
408     def forward(
409         self, mentions: Mentions, candidates: List[Entities]
410     ) -> List[Tuple[Entity, float]]:
411         logits = self._get_logits(mentions, candidates)
412         scores = self.m(logits)
413         entities = [
414             (cands[i], score[i].item())
415             for i, cands, score in zip(scores.argmax(
416                 dim=1), candidates, scores)
417         ]
418         return entities
419 
420     def on_train_start(self):
421         self.logger.log_hyperparams(self.hparams)
422 
423     def training_step(self,
424                       batch: Tuple[Mentions, List[Entities],
425                                    Entities], batch_idx):
426         mentions, candidates, targets = batch
427         logits = self._get_logits(mentions, candidates)
428         y = self.encoder.encode_targets(candidates, targets)
429         loss = F.binary_cross_entropy_with_logits(logits, y)
430         self.log("train_loss", loss)
431         self.train_accuracy(logits.argmax(dim=1),
432                             y.argmax(dim=1))
433         accuracy = self.train_accuracy.compute()
434         self.log("train_accuracy", accuracy)
435         return loss
436 
437     def training_epoch_end(self, outputs):
438         self.train_accuracy.reset()
439 
440     def _eval_step(self,
441                    batch: Tuple[Mentions, List[Entities],
442                                 Entities], batch_idx):
443         mentions, candidates, targets = batch
444         logits = self._get_logits(mentions, candidates)
445         y = self.encoder.encode_targets(candidates, targets)
446         scores = self.m(logits)
447         self.accuracy(scores.argmax(dim=1), y.argmax(dim=1))
448 
```

```
449     def validation_step(self, batch, batch_idx):
450         return self._eval_step(batch)
451 
452     def validation_epoch_end(self, results):
453         accuracy = self.accuracy.compute()
454         self.log("validation_accuracy",
455                  accuracy,
456                  prog_bar=True)
457         self.f1.reset()
458 
459     def test_step(self, batch, batch_idx):
460         return self._eval_step(batch)
461 
462     def test_epoch_end(self, results):
463         accuracy = self.accuracy.compute()
464         self.log("test_accuracy", accuracy, prog_bar=True)
465         self.f1.reset()
466 
467     def configure_optimizers(self):
468         optimizer = AdamW(self.parameters(),
469                           lr=self.hparams.learning_rate)
470         aida_size = int(946 / 8) + 1
471         lr_scheduler = get_linear_schedule_with_warmup(
472             optimizer, aida_size, 20 * aida_size)
473         lr_scheduler_config = {
474             "scheduler": lr_scheduler,
475             "interval": "step",
476             "frequency": 1
477         }
478         return {
479             "optimizer": optimizer,
480             "lr_scheduler": lr_scheduler_config
481         }
```

Erklärung

Ich versichere hiermit, dass ich meine 2. Praxisarbeit mit dem Thema: *Entity Linking in der Praxis* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

(Ort, Datum) (Unterschrift)