

Forschungsarbeit

Im Fachgebiet Informatik und Mathematik

Einsatz effektiverer Transformer-Encoder im Bereich
der Open Information Extraction

Verfasser: Henning Beyer
Klassenstufe: 13
Schuljahr: 2022/23
Schulischer Betreuer: Romy Schachoff
Außerschulischer Betreuer: Prof. Andreas Maletti
Datum: Leipzig, 23.01.2023

Kurzfassung

In dieser Forschungsarbeit wird im Bereich der Open Information Extraction (OIE) ein Encoder-Austausch am DetIE-Modell mit den effektiver vortrainierten Varianten RoBERTa und ELECTRA vorgenommen. Hiermit können insgesamt sehr relevante Verbesserungen von durchschnittlich bis zu 1,1 F1- und 1,3 AUC-Punkten ohne Verminderung der Extraktionsgeschwindigkeit erzielt werden, während dieser Encoder-Austausch zudem simpel für alle Modelle mit BERT-Encoder reproduzierbar und somit auch für zukünftige OIE-Modelle von großer Relevanz ist.

Dabei legt diese Forschungsarbeit neben einer praktischen außerdem eine wichtige theoretische Grundlage zu Versuchen dieser Art, indem diese die Theorie der Encoder RoBERTa, ELECTRA, DeBERTa sowie DeBERTaV3 beschreibt und die nötigen Kriterien für eine Verbesserung identifiziert: Die Sprachaufgabe benötigt die nötige Schwierigkeit und je nach der Schwierigkeit auch die nötige Menge an Trainingsdaten, um das Sprachwissen der Encoder effektiv nutzen zu können.

Messwerte, Messtabellen und Trainings-Logs sind über <https://github.com/HenningBeyer/DetIE-with-ELECTRA> einsehbar. Eine Wiederholung der Experimente mit DeBERTa- und DeBERTaV3-Encoder erfolgt in Kürze unter Erwartung noch höherer Leistungen.

Inhaltsverzeichnis

Kurzfassung.....	5
1. Einleitung.....	4
1.1 Stand der Forschung im Natural Language Processing.....	4
1.2 Stand der Forschung in der Open Information Extraction.....	4
1.3 Motivation und Ziele der Forschungsarbeit.....	5
2. Theorie	6
2.1 Transformer-Modelle.....	6
2.1.1 Grundlegende Funktionsweise.....	7
2.1.2 Scaled Dot-Product Attention	7
2.1.3 Multi-Head Attention	8
2.2 BERT-Modelle	9
2.3 Optimierte BERT-Varianten	9
2.3.1 RoBERTa	9
2.3.2 ELECTRA	10
2.3.3 DeBERTa	10
2.3.4 DeBERTaV3.....	12
2.4 Das DetIE-Modell	13
2.4.1 Order-Agnostic Loss	14
3. Verbesserung des DetIE-Modells mittels Encoder-Austausch.....	14
3.1 Experimentbeschreibung	14
3.1.1 Server und Hardware.....	14
3.1.2 Code des DetIE-Modells.....	15
3.1.3 Datensätze und Bewertungsvorgang	15
3.2 Durchführung der Experimente.....	16
3.3 Auswertung der Ergebnisse.....	17
3.3.1 Niedrige Leistungen für DeBERTa-Encoder.....	18
4. Fazit.....	18
Literaturverzeichnis.....	19
Danksagung	23
Selbstständigkeitserklärung.....	23
Abkürzungsverzeichnis	24

1. Einleitung

In dieser Forschungsarbeit werden die zwei Bereiche der Open Information Extraction (OIE) und des Natural Language Processings (NLP) behandelt. Hierbei befasst sich die OIE mit dem strukturierten Extrahieren von Relationstupel und bedient sich an einigen Methoden des Natural Language Processing (NLP), welches sich als übergeordneter Bereich mit der algorithmischen Verarbeitung der menschlichen Sprache befasst. Bevor Thema und Ziele vorgestellt werden, wird zuerst mit diesen beiden Bereichen als Hinführung in die Forschungsarbeit begonnen und deren aktueller Forschungsstand erläutert.

1.1 Stand der Forschung im Natural Language Processing

Der Bereich des NLP wird bereits effektiv für Sprachaufgaben wie die Übersetzung oder dem Schreiben von zusammenhängenden Texten eingesetzt. Eine steigende Nachfrage sowie große Fortschritte sorgen hierbei für eine anhaltende Forschung an immer effektiveren und effizienteren Methoden. Seit dem Jahr 2017 wechselt man nun vollständig zu den Transformer-Modellen [1], die mittels Parallelisierbarkeit und einer sehr hohen Parameteranzahl die menschliche Sprache tiefgründiger verstehen, sodass einfache Sprachaufgaben teils vollständig gelöst werden können.

Die Transformer-Modelle legen den Grundstein der heutigen NLP-Forschung und werden zum Erzielen neuer Bestleistung stets vergrößert, während ebenfalls das Vortraining zum Erlernen der Sprache optimiert wie bei den Modellen BERT [2] und dessen Nachfolgern RoBERTa [2], ELECTRA [3], DeBERTa [4] und DeBERTaV3 [5] wird. So konnte im Januar 2021 die menschliche Sprachleistung zum ersten Mal überhaupt in der SuperGLUE-Benchmark [6] vom DeBERTa-Modell übertroffen werden [4], [7]: DeBERTa trainierte mit 1,5 Milliarden Parametern für 30 Tage auf 256 GPUs und konnte hiermit Googles T5-Modell [8] mit 11 Milliarden Parametern dank eines optimierten Vortrainings eindeutig übertreffen.

Trotz der Fortschritte von Sprachverständnis und dem Trainieren mit zunehmend mehr Parametern fehlt es den Modellen immer noch an Wissen, um die menschliche Sprache tiefgründiger mit deren Kontext zu verstehen und Informationen einordnen zu können. Hierfür wird zurzeit an Methoden geforscht, um den Modellen ein Allgemeinwissen zu verleihen, indem Modelle wie ERNIE 3.0 [9] zusätzlich mithilfe von faktenbasierten Wissen trainiert werden oder wie LaMDA [10] ihre Antworten auf Grundlage von Wissensdatenbanken generieren. Indem Modellen wie LaMDA oder ChatGPT [11] Zugriff auf Internetseiten gewährt wird, können diese Modelle zudem noch eigenständiger mit eigenen Information-Retrieval-Systemen ihre Antworten generieren.

1.2 Stand der Forschung in der Open Information Extraction

Die OIE beschäftigt sich als Teilgebiet des NLP mit der Extraktion von strukturierten Informationen aus Texten in Form von Relationstupel nach dem Schema (*Subjekt; Relation; Objekt*) [12]. Mit diesen geordneten Informationen unterstützt dieser Bereich viele untergeordnete Aufgaben des NLP wie das Question Answering oder die Textzusammenfassung sowie ebenfalls jegliche Geschäftsprozesse, die auf Knowledge Bases angewiesen sind. Hierbei wird die OIE zum automatischen Extrahieren aus Internet- sowie Text-Dokumenten angewendet, während besonders die Effektivität der OIE-Modelle für die Menge an qualitativen Relationstupel verantwortlich ist.

Erstmalig wurde die OIE um 2007 mit dem Modell TEXTRUNNER eingeführt [13], [14] und geht seit dem Jahr 2018 zur Nutzung neuronaler Modelle über, welche die Sprachstrukturen in Sätzen flexibler erkennen können und eine Akkumulation von Fehlern durch Nutzung mehrerer verschiedener Teilkomponenten verhindern [15]. Generell teilen sich die neuronalen OIE-Modelle in zwei Bereiche auf: der Sequence Generation und dem Sequence Labeling. Die Sequence Generation umfasst dabei Seq2seq-Strukturen, die Extraktionen langsam autoregressiv Wort für Wort generieren und dadurch niedrige Extraktionsgeschwindigkeiten erzielen. Da Modelle der Sequence Generation Token aus einem eigenen Vokabular generieren, können diese wichtige benötigte Token wie z.B. *is*, *of* oder *from* selbstständig generieren, aber hingegen Tokens außerhalb des Vokabelsatzes nicht.

Beide Bereiche haben das Problem, oft redundante Extraktionen zu generieren, wogegen das aktuell leistungsstärkste Modell der Sequence Generation – IMoJIE [16] – einen iterativen Mechanismus benutzt, der für das Generieren jeder Extraktion zusätzlich alle bisher generierten Extraktionen erfasst. Zudem löst IMoJIE für die Sequence-Generation-Modelle das Out-of-Vocabulary-Problem [17] mit Nutzung der Copy-Attention [17]. Jedoch ist dieser iterative Mechanismus in Kombination mit der nicht parallelisierten Copy-Attention während des Decodings die Ursache für eine sehr langsame Extraktionsgeschwindigkeit. Innerhalb der Sequence-Generation existieren aktuell noch keine Lösungsansätze, die die beiden benannten Probleme bei einer annehmbar hohen Extraktionsgeschwindigkeit lösen können, weswegen man sich in der jetzigen OIE-Forschung am Sequence-Labeling orientiert, die das Redundanzproblem bei hoher Extraktionsgeschwindigkeit mit Modellen wie OpenIE6 [12] oder DetIE [15] effizient und simpel lösen kann.

Zwar können Modelle des Sequence-Labeling keine eigenen Token in die Extraktionen einfügen, dafür müssen diese aber nur zum Extrahieren die gegebenen Tokens klassifizieren, anstatt diese aus einem großem Vokabular autoregressiv und konsekutiv zu generieren, was Folgefehler vermeidet. Das Modell OpenIE6 [12] nutzt hierbei einen Mechanismus des *iterative Grid Labelings*, der im Gegensatz zu IMoJIE mit einem parallelisierten Decoding und nur einmaligem Encoding der Extraktionen für OpenIE6 eine mehr als zehnfache Extraktionsgeschwindigkeit verleiht [12]. Das Modell DetIE [15] vermeidet hingegen jegliche iterative Struktur gänzlich, indem es 20 Klassifikationen für potentielle Extraktionen in einer einzelnen Iteration vornimmt und damit bis zu 3-mal schneller als sein Vorgänger OpenIE6 extrahiert [15]. Diese gegen Folgefehler geschützte Methode macht DetIE zum aktuell leistungsfähigsten sowie zugleich zu einem der schnellsten Modelle der OIE [15], wobei jedes der bisherigen OIE-Modelle von einem BERT-Encoder Gebrauch macht. Über eine Anwendung von den besser vortrainierten Encoder-Modellen RoBERTa [2], ELECTRA [3], DeBERTa [4] und DeBERTaV3 [5] wurde dabei in der OIE noch nicht diskutiert.

1.3 Motivation und Ziele der Forschungsarbeit

Diese Forschungsarbeit orientierte sich ursprünglich am IMoJIE-Modell [16] mit dem Ziel, dessen Encoder-Modell mit effektiveren Varianten auszutauschen. Hierfür unternahm der Teilnehmer zuerst Experimente an einem leichteren Decoder-Austausch in einer Belegarbeit und stellte die Ergebnisse auch bei Jugend forscht vor. Hiermit konnte schließlich eine zentrale Ursache für die langsame Extraktionsgeschwindigkeit von IMoJIE ausfindig gemacht werden, nämlich eine einzelne ineffiziente Funktion, die zudem erfolgreich optimiert wurde, wodurch die Laufzeit des IMoJIE-Modells von 247,4 auf 85,8 Sekunden für 641 Sätze ohne Leistungsbeeinträchtigung reduziert werden konnte.

Jedoch wird in dieser Forschungsarbeit, die parallel zu einer Besonderen Lernleistung angefertigt wird, ein Encoder-Austausch am aktuelleren DetIE-Modell [15] vorgenommen, da es für die Forschung nicht nur viel relevanter ist, sondern da dieses Modell ebenfalls mit effektiveren Encodern neue Bestleistungen in der OIE erzielen könnte.

Somit wird in dieser Forschungsarbeit der BERT-Encoder des DetIE-Modells mit effektiveren Varianten ausgetauscht, um dessen Leistung zu steigern. Hierzu wird die folgende Hypothese formuliert, an der sich während der gesamten Forschungsarbeit orientiert wird:

- Durch den Austausch des BERT-Encoders mit den effektiveren Encoder-Varianten RoBERTa, ELECTRA, DeBERTa sowie DeBERTaV3 verbessern sich die Leistungen von DetIE und anderen OIE-Modellen.

2. Theorie

Das behandelte DetIE-Modell benutzt als Encoder ein BERT-Modell und als Decoder neben neuronalen Netzwerken auch Transformer-Schichten. Mit der nachfolgenden Theorie werden neben diesen Modellen alle der neuartigen, zu testenden Encoder RoBERTa, ELECTRA, DeBERTa sowie DeBERTaV3 aufeinanderfolgend beschrieben, sodass die Intention und Durchführung der Experimente verständlich werden und Argumente der Auswertung nachvollzogen werden können. Auch wenn die Theorie mit den Transformer-Modellen beginnt und keine weiteren Grundlagen für NLP und OIE extra beschreibt, sollte man sich ohne ausführlichen Vorkenntnissen anhand der Einleitung in die Theorie einlesen können. Spätestens ab dem BERT-Modell bleiben die Grundkonzepte ähnlich, womit dann die folgenden Encoder-Modelle und DetIE verständlicher werden.

2.1 Transformer-Modelle

Nachdem die Sprachmodelle mit Seq2seq-Struktur und Attention-Mechanismen für die Anwendung immer weiter vergrößert wurden, stößt man mit den bisherigen rekurrenten neuronalen Netzen schnell an deren Leistungsgrenzen und benötigt eine zudem sehr hohe Rechenleistung, um annehmbare Leistungen für die meisten Sprachaufgaben zu erzielen. Grund hierfür ist vor allem die eingeschränkte Möglichkeit der Parallelisierung durch den autoregressiven Ablauf der Modelle [1], was auch davon abhält die gegebenen Modell-Strukturen für die Leistungssteigerung weiter zu vergrößern.

Gegen dieses Problem gehen Transformer-Modelle nun mit einer Modellstruktur ganz ohne rekurrente Mechanismen vor und stattdessen basieren diese nahezu vollständig auf der Self-Attention, die eine weitaus größere Parallelisierung der Berechnungen zulässt [1]. Somit sind Transformer simpler, skalierbarer und deutlich leistungsfähiger als die besten Vorgängermodelle. Bis heute bilden die Strukturen der Transformer den Grundstein für die Forschung an weiteren Sprachmodellen des NLP.

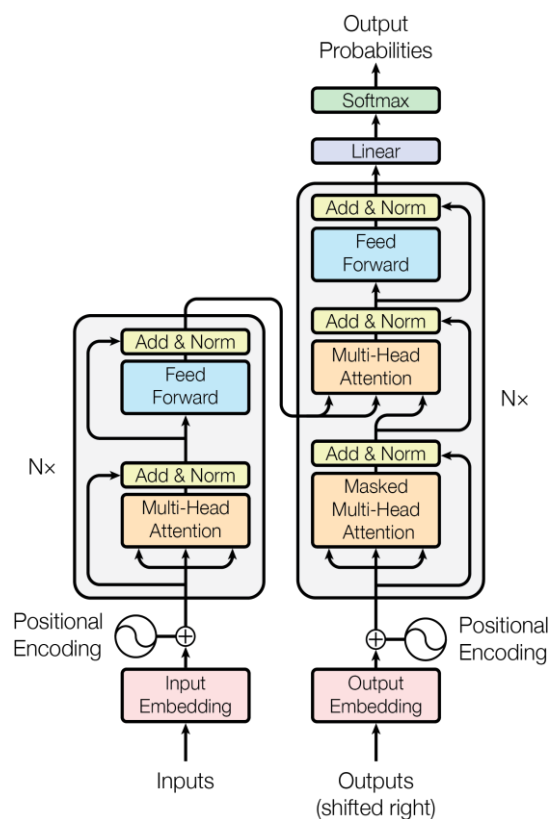


Abb. 1: Die Abbildung der Transformer-Modell Struktur [1].

2.1.1 Grundlegende Funktionsweise

Wie die vorherigen Seq2seq-Modelle verwenden die Transformer sogenannte Encoder und Decoder, um zuerst Informationen aus einem Input-Sequenz zu extrahieren und danach damit die Output-Sequenz gezielt zu generieren. Jedes Transformer-Modell besteht aus N Encoder- und Decoder-Blöcken, wobei der Encoder mit nur einem Durchlauf die Bedeutung der Tokens in der Input-Sequenz erfasst und dann die Informationen des letzten Encoder-Blocks an alle Decoder-Blöcke übergibt. Dabei ist nur der Decoder rechts in Abb. 1 autoregressiv und generiert die Output-Token weiterhin schrittweise basierend auf der letzten Ausgabe eines Tokens.

Zu Beginn der Berechnungen werden die Embedding-Vektoren berechnet und hiermit eine Embedding-Matrix gebildet, wobei das Embedding die Bedeutung der Input-Tokens repräsentiert. Diese Berechnung übernimmt ein einschichtiges neuronales Netzwerk, das die Tokens in Vektoren der Länge $d = 512$ konvertiert. Zu diesen Bedeutungen wird anschließend die Informationen des Positional Encodings hinzuaddiert, sodass die absolute Position aller Wörter wie folgt mit erfasst wird [1]:

$$PE_{(pos,2i)} = \sin(pos/10000^{\frac{2i}{d}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{\frac{2i}{d}})$$

Dabei steht i für einen einzelnen Embedding-Vektor-Index und pos für die absolute Position des Tokens in der Input-Sequenz. Diesen Input-Embedding-Vektoren wird nun mithilfe eines Multi-Head Attention Blocks der eigenen Kontext verliehen, indem jeder Vektor mit dem Ergebniss der Self-Attention gewichtet wird. Das Ergebnis wird über eine *Residual Connection* zu der vorherigen Embedding-Matrix addiert und das Ergebnis anschließend normalisiert. Diese *Layer Normalization* [18] sorgt für ein erleichtertes und schnelleres Training, indem es von den Matrizenwerten den Durchschnitt subtrahiert und die Ergebnisse durch die Standardabweichung teilt. Hiermit erhalten die Werte einen Durchschnitt von 0 sowie eine einheitliche Varianz. Schließlich erfolgt eine ähnliche Berechnung mit einem neuronalen Netzwerk anstelle des Multi-Head-Attention-Blocks.

Im Gegensatz zum Encoder verwendet der Decoder ein Masked Multi-Head Attention Block, der lediglich Positionen im Output ausblendet, die noch nicht generiert wurden. Ebenso erhält der mittige Multi-Head-Attention-Block jeder Decoder-Schicht die Hidden-State-Vektoren des letzten Encoder-Blocks, um Encoder-Informationen mit den Decoder-Informationen zu kombinieren.

2.1.2 Scaled Dot-Product Attention

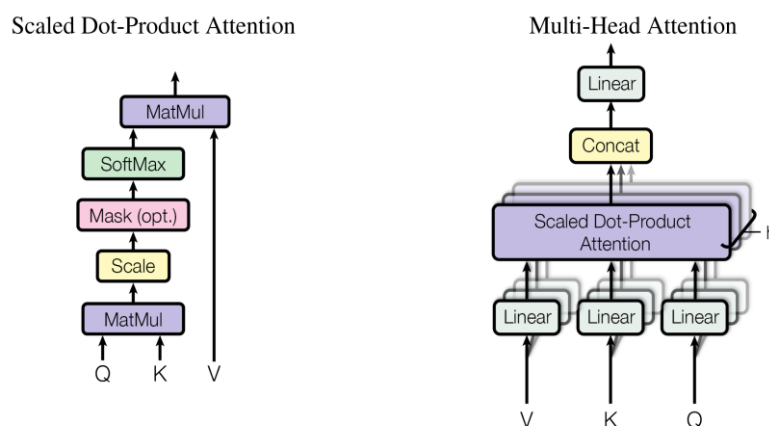


Abb. 2: Die detaillierte Abbildungen der im Transformer verwendeten Multi-Head Attention (rechts) und der hierin verwendeten Scaled Dot-Product Attention (links) [1].

Die benannten Multi-Head-Attention-Blöcke verwenden als grundlegenden Attention-Mechanismus die Scaled Dot-product Attention, die danach strebt die gegebenen Vektoren der Value-Matrix V mit den Ähnlichkeitswerten zwischen zwei Vektoren Query Q und Key K zu gewichten. Hierbei sind die Vektoren Q und K im Transformer immer exakt gleich, bevor sie innerhalb der Multi-Head Attention durch Parameter-Matrizen unterschiedlich gewichtet werden, um Bedeutungen der Tokens auf tieferer Ebene zu erfassen. Anders als die Additive Attention wird für die Berechnungen der Scaled Dot-Product Attention kein neuronales Netzwerk benutzt und kann mittels einem einzigen Skalarprodukt berechnet werden, was in der Praxis einen hoch optimierten Matrix-Multiplikations-Code zulässt [1]. Entsprechend der Abb. 2, berechnet sich die Scaled Dot-Product Attention wie folgt [1]:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Hierbei wird durch den Term $\sqrt{d_k}$ geteilt, um zu große Werte des Skalarprodukts zu vermeiden, welche mit zunehmender Sequenzlänge zunehmen können [1]. In der Anwendung werden ebenfalls anstatt der Vektoren Matrizen verwendet, deren Spalten mehrere Embedding-Vektoren der einzelnen Tokens bilden. Dabei gilt, dass im Skalarprodukt QK^T jeder Vektor der Matrix Q mit allen anderen unterschiedlich gewichteten Vektoren der Matrix K elementweise gewichtet werden, sodass hiermit alle möglichen Token-zu-Token-Ähnlichkeiten berechnet werden. Diese resultierende Ähnlichkeitsmatrix gewichtet wie in der oberen Formel die Matrix V und verleiht damit den Vektoren in dieser Matrix eine Bedeutung im Bezug auf die gesamte verarbeitete Sequenz.

2.1.3 Multi-Head Attention

Anstatt nur eine einzelne Berechnung mit der Scaled Dot-Product Attention vorzunehmen werden mit der Multi-Head Attention mehrere verschiedene, erlernte lineare Projektionen berechnet bzw. mehr Parameter in die Berechnungen eingebracht, womit die Self-Attention tiefgründiger und von verschiedenen Sichtweisen berechnet wird [1]. Die Berechnung findet jeweils pro Attention-Head $head_i$ mit den drei eigenen Parametermatrizen $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d_k}$ statt, die die Matrizen $Q, K, V \in \mathbb{R}^{N \times d}$ mehrmals unterschiedlich gewichten bzw. projizieren. Dabei beträgt $d_k = \frac{d}{h} = 64$ bei standardmäßig $h = 8$ Attention-Heads.

Alle der Attention-Heads $head_{1...h} \in \mathbb{R}^{N \times d_k}$ werden nach ihrer Berechnung horizontal aneinandergefügt, um sie als eine Matrix schließlich mit der weiteren Parameter-Matrix $W^O \in \mathbb{R}^{hd_k \times d}$ multiplizieren zu können. Somit berechnet sich die Multi-Head Attention wie folgt [1]:

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O,$$

$$\text{wobei } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

2.2 BERT-Modelle

Seit dem Jahr 2018 gibt es die für das heutige NLP unerlässlichen BERT-Modelle [19], die den Encoder des Transformer-Modells darstellen und erst in einer Feinabstimmung mit einem weiteren beliebigen Decoder zusammengesetzt werden. Hierbei erlangen BERT-Modelle mit zwei neuen Trainingsobjektiven sowohl ein antrainiertes bidirektionales Satzverständnis als auch ein Gesamtsatzverständnis, wodurch BERT-Modelle erneut neue Bestleistungen bei einer simpleren Struktur erzielen.

Das BERT-Modell erlangt dabei sein verbessertes Satzverständnis in einem sogenannten Vortraining, während es hierbei den Objektiven des *Masked Language Modelings* und der *Next Sentence Prediction* folgt, die ein wesentliches Verständnis aller Satzbestandteile vom Modell erfordern. Anschließend erhalten die vortrainierten BERT-Encoder in einer weiteren sogenannten Feinabstimmung einen neuen untrainierten Decoder für Anwendungsaufgaben wie z.B. dem Question Answering.

Innerhalb des *Masked Language Modelings* werden dabei 15% der gegebenen Tokens mit einem gleichbleibenden Masken-Schema statisch überdeckt, wobei das Modell diese Tokens wieder korrekt in einer Mehrfachklassifikation vorhersagen muss. Diese überdeckten Tokens bilden wiederum zu 80% die *[MASK]*-Tokens und jeweils zu 10% zufällige oder doch die korrekten Tokens [19]. Andererseits muss in einer *Next Sentence Prediction* vom Modell bestimmt werden, ob ein aus dem Datensatz entnommener Satz auf den gegebenen Input-Satz folgt oder nicht.

2.3 Optimierte BERT-Varianten

2.3.1 RoBERTa

Ein Jahr nach der Veröffentlichung von BERT erschien um 2019 das sogenannte **Robustly Optimized BERT Approach** Modell [2], das einige Mängel in der Parameter-Konfiguration und Trainingsweise des BERT-Modells feststellte, sorgfältig analysierte und diese am BERT-Modell verbesserte. Hieraus entsteht aus insgesamt 7 kleineren Anpassungen das leistungsfähigere RoBERTa-Modell, wobei zu den drei wichtigsten die folgenden Anpassungen gehören [2]:

- Das Entfernen der *Next Sentence Prediction* und des NSP-Loss, wobei aber weiterhin nachfolgenden Sätze mit als Modell-Input für das Vortraining verwendet werden, sodass das Satzverständnis erhalten bleibt.
- Das *Dynamic Masking*, das anstatt einer statischen Maske nun jede Trainingsepoche das Maskierungsschema verändert und damit die Trainingsdaten effizienter nutzt.
- Das Trainieren auf ausschließlich der vollständigen Sequenz-Länge von 512 Tokens, während das Vortraining bei BERT für 90% der Trainingsschritte für ein schnelleres Vortraining auf eine Sequenzlänge von 128 beschränkt wurde [19].

Schließlich beinhalten die letzten vier ebenfalls wichtigen Erneuerungen des RoBERTa-Modells ein umfangreicheres Training in Form von einem viel größeren Trainingsdatensatz, einer größeren Trainings-Batch-Size von 8.000, einer Anpassung der Hyperparameter sowie einer stark gesteigerten Anzahl an Trainingsdurchläufen [2]. BERT war nämlich, wie die Autoren des RoBERTa-Modells herausgefunden haben, signifikant untertrainiert [2] und konnte noch weitere Informationen der Sprache in seiner Transformer-Struktur erfassen.

2.3.2 ELECTRA

Um 2020 erscheint nun das ELECTRA-Modell [3], das mit einer Generative Adversarial Network (GAN) Struktur [20] anstelle des *Masked Language Modelings* eine jetzt viel stichprobeneffizientere *Replaced Token Detection* im Vortraining verwendet und neben einem Leistungszuwachs ebenso die Konvergenzzeit stark reduziert. Im Unterschied zu den bekannten GAN-Strukturen, trainiert ELECTRA mit einem nicht-konkurrierenden Trainingsobjektiv, während das Generator- und Diskriminator-Modell in Abb. 3 separat voneinander optimiert werden. Nach dem Vortraining wird schließlich der Diskriminator, der die Vorhersagen des Generators korrigieren muss, aus der GAN-Struktur gelöst und für die weitere Feinabstimmung als ELECTRA-Modell genutzt.

2.3.2.1 Replaced Token Detection

BERT-Modelle überdecken innerhalb des Masked Language Modelings nur 15 % der gegebenen Tokens statisch und davon 80% dieser Tokens mit einem *[MASK]*-Token, der dann in weiteren Prozessen keine Verwendung findet. Die *Replaced Token Detection* in Abb. 3 löst dieses Problem der geringen Stichprobeneffizienz, indem anders als auch in RoBERTa zu 15%, wirklich jeder Token in jeden Trainingsdurchlauf zufällig überdeckt werden kann, womit das Vortraining des ELECTRA-Modells mit deutlich weniger Trainingsdurchläufen die Satzbedeutungen schneller erlernen kann [3].

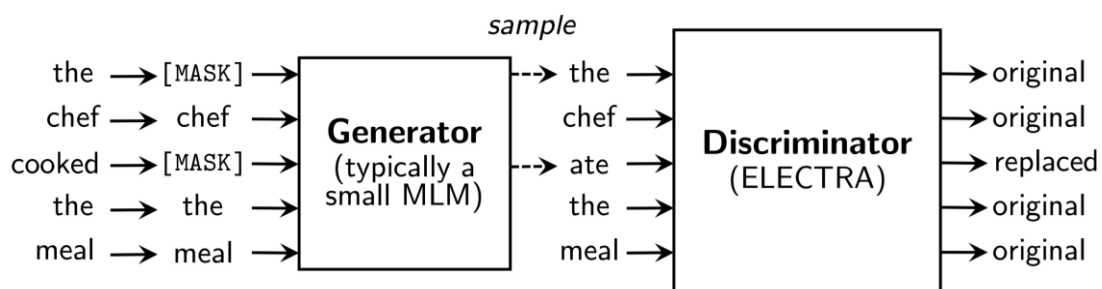


Abb. 3: Die Darstellung der abgewandelten GAN-Architektur mit Transformer-Encodern für jeweils den Generator und den Diskriminator. Die Aufgabe des Generators ist es, die zufällig überdeckten Tokens wieder mit den korrekten Original-Tokens zu ersetzen. Dagegen muss der Diskriminator für jeden der Tokens erkennen, ob dieser durch den Generator falsch ersetzt wurde oder nicht [3].

In der Praxis erzielen ELECTRA-Modelle, die mit etwa 25 % bis 50 % kleineren Generator-Modellen vortrainiert werden bessere Leistungen [3], weswegen diese beabsichtigt weniger leistungsstark gehalten werden. Ebenfalls wird in der Praxis ein Weight-Tying zwischen den Token-Embedding-Modellen des Generators und Diskriminators durchgeführt [3], damit beide Modelle den Input gleichermaßen repräsentieren.

2.3.3 DeBERTa

Nur einen Monat später folgte auf das ELECTRA-Modell das DeBERTa-Modell [4], das wieder die Leistung seines Vorgängers übertrifft. DeBERTa nutzt dazu im Vortraining den vorteilhaften Mechanismus der *Disentangled Attention*, welcher die Hidden-State-Vektoren der Tokens nun mit getrennten Vektoren für Inhalt sowie Position darstellt und die Transformer-Berechnungen mittels entwirrter

Matrizen vornimmt [4]. Um hierbei weiterhin das *Masked Language Modeling* durchzuführen, erhält dieses Vortraining eine kleine Anpassung mit dem *Enhanced Mask Decoding*.

2.3.3.1 Disentangled Attention

Die *Disentangled Attention* ist die erste Erneuerung, die die Berechnungen auch nach dem Vortraining verändert, indem lediglich in der Scaled Dot-Product Attention des Transformers weitere Berechnungsschritte hinzugefügt werden.

Bisher verwendeten Transformer-Modelle Sinus- bzw. Kosinus-Funktionen zum berechnen eines absoluten Positional-Encodings, um die Tokens bezogen auf deren absoluten Position einzuordnen. Jedoch führt das neuere Konzept des relativen Positional-Encodings in der Anwendung bewiesenermaßen zu einem verbesserten Sprachverständnis [4], [21], [22], da es nämlich für jeden Token die eigene relative Position zu allen anderen Tokens in der Sequenz darstellt, was sehr wichtig zum genauen Interpretieren der Wortbedeutungen in längeren Sätzen ist. In bisherigen Implementationen [21], [22] berechnete sich die Output-Hidden-State-Matrix H_o mit einem relativen Position-Encoding, indem nur zwei Terme für die Inhalts-zu-Inhalts- sowie Inhalts-zu-Positions-Beziehungen wie folgt einbezogen wurden [22]:

$$H_o = \text{softmax} \left(\frac{QK^T + Q(A^K)^T}{\sqrt{d_k}} \right) V,$$

wobei die Matrix A^K für ein berechnetes relatives Positional-Encoding steht. Hierbei fehlt aber eine ebenfalls noch wichtige Positions-zu-Inhalts-Beziehungen, die notwendig ist, um die Token-Bedeutung in Bezug zu dessen Position vollständig darzustellen [4]. Deswegen wird mit dem DeBERTa-Modell versucht unter einem vektorweisen Berechnungsvorgang diese fehlende *Position-to-Content* Beziehung wie folgt zu ergänzen [4]:

$$\tilde{A}_{ij} = \underbrace{Q_i^c (K_j^c)^T}_{\text{Content-to-Content}} + \underbrace{Q_i^c (K_{\delta(i,j)}^r)^T}_{\text{Content-to-Position}} + \underbrace{K_i^c (Q_{\delta(j,i)}^r)^T}_{\text{Position-to-Content}},$$

$$H_o = \text{softmax} \left(\frac{\tilde{A}}{\sqrt{3d}} \right) V_c,$$

$$\text{wobei} \quad Q_c = HW_c^Q, \quad Q_r = PW_r^Q, \quad K_c = HW_c^K, \quad K_r = PW_r^K \quad \text{und} \quad V_c = HW_c^V$$

Hierbei sind neben den Matrizen $W_c^Q, W_r^Q, W_c^K, W_r^K, W_c^V \in \mathbb{R}^{d \times d}$ die noch nicht bekannten Berechnungen der Matrizen P und \tilde{A} zu erwähnen: $P \in \mathbb{R}^{2k \times d}$ ist eine Matrix mit den Vektoren des Relative-Distance-Embeddings der einzelnen Tokens; diese ist hierbei eine gewöhnliche, lernfähige, zufällig initialisierte Gewichtsmatrix, die von allen Transformer-Schichten geteilt wird. Dagegen beinhaltet die Matrix $\tilde{A} \in \mathbb{R}^{N \times N}$ mit $\tilde{A} = A_{c \rightarrow c} + A_{c \rightarrow p} + A_{p \rightarrow c}$ für jeden der $N = 512$ Tokens einen Attention-Score in Vektoren der Länge N . Auf die *Position-to-Position*-Beziehung wird dabei wegen deren geringen Informationsgehalt verzichtet [4].

Die Berechnung von \tilde{A} ist jedoch wegen der insgesamt $2k = 1024$ möglichen relativen Positionen in der Code-Implementation etwas aufwendiger und muss mit einer Index-Funktion $\delta(i, j) \in [0, 2k)$ umgesetzt werden: Für $A_{c \rightarrow p}$ werden mit der Index-Funktion δ die 512 relativen Positionen des Bezugstokens zu allen anderen Tokens betrachtet und umgekehrt werden für $A_{p \rightarrow c}$ mittels eines Index-Tauschs über δ die 512 relativen Positionen der anderen Tokens zum Bezugstoken betrachtet. Diese

$2k = 1024$ Relative-Distance-Embedding-Vektoren finden sich in den Matrizen $Q_r, K_r \in \mathbb{R}^{2k \times d}$ wieder und müssen mit der Index-Funktion δ für eine korrekte Kombination mit den Matrizen $Q_c, K_c \in \mathbb{R}^{N \times d}$ entnommen werden. Somit werden anders als für $A_{c \rightarrow c}$ keine Matrix-Matrix-Multiplikation mehr durchgeführt, sondern nur noch iterativ mehrere Vektor-Matrix-Multiplikationen, was die Ausführungsgeschwindigkeit in den Experimenten merklich mindern könnte und berücksichtigt werden muss. Zudem besitzt DeBERTa mit seinen zusätzlichen Gewichtsmatrizen 15 Millionen Modell-Parameter mehr als das BERT-Modell.

2.3.3.2 Enhanced Mask Decoding

DeBERTa wird immer noch mit dem *Masked Language Modeling Objektiv* vortrainiert und benötigt dabei zur Vorhersage der *[MASK]*-Tokens nach wie vor die absolute Position, damit die *[MASK]*-Token innerhalb des Satzes richtig eingeordnet werden können. Da jedoch nur noch relative Token-Positionen vorhanden sind, werden in DeBERTa ausschließlich den Transformer-Decodern im Vortraining die Informationen über die absoluten Token-Positionen zusätzlich übergeben, sodass die Sprache im Vortraining besser erlernt werden kann. Ergebnisse im Forschungsbericht zeigen mithilfe des *Enhanced Mask Decodings* verbesserte Leistungen [4].

2.3.4 DeBERTaV3

Anknüpfend an das bisherige DeBERTa-Modell [4] verbessert DeBERTaV3 [5] wieder merklich die Modell-Leistung, indem nun das wirksame Vortraining des ELECTRA-Modells [3] und das ebenso effektive Konzept der *Disentangled Attention* vereint werden. Hierbei erhält DeBERTaV3 anstelle des üblichen *Masked Language Modelings* nun das Vortrainingsobjektiv der *Replaced Token Detection*, um die Stärken beider Modelle zu kombinieren [5]: Nämlich der *Disentangled Attention* mit einer informativeren Repräsentation der Token-Positionen sowie der Stichprobeneffizienten und performanten *Replaced Token Detection*.

Innerhalb von ELECTRA führt das gegenseitige Teilen der Embedding-Gewichte zwischen den Generator und Diskriminator im Optimierungsprozess zu einem „Tauziehen“ [5], weshalb für DeBERTaV3 zusätzlich der neue Mechanismus der *Gradient-Disentangled Embedding Sharing* entwickelt wird, wie in der Abb. 4:

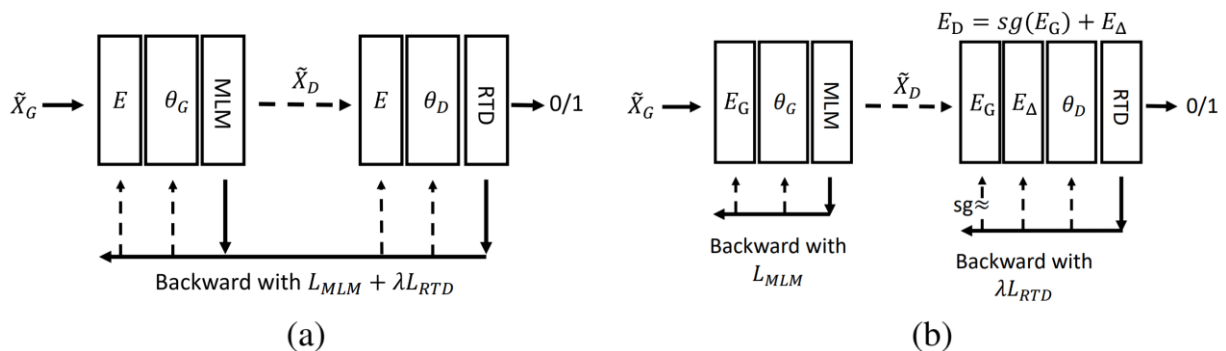


Abb. 4: Die Darstellung des Optimierungsprozesses bei geteilten Token-Embedding-Parametern (a) sowie mit der Methode des *Gradient-Disentangled Embedding Sharings* (b). Hierbei steht *sg* für den *Stop Gradient*, der die Optimierung von E_G verhindert. Bearbeitet. [5].

2.3.4.1 Gradient-Disentangled Embedding Sharing

Während des ELECTRA-Vortrainings teilen die Token-Embedder des Generators E_G und des Diskriminators E_D ihre Parameter miteinander in einem Embedding E , womit deren Optimierung zweimal anhand jeweils des Fehlers des Generators sowie anhand des Fehlers des Diskriminators stattfindet. Hierbei werden über Backpropagation, wie in Abb. 4 a, alle Berechnungsschritte vom Ende des Diskriminators bis hin zum Token-Embedding des Generators über Gradienten Produkte zurückverfolgt und mit einem AdamW-Optimizer [23] optimiert.

Dabei aber getrennte Embeddings für Generator und Diskriminator zu nutzen, verringert die Modell-Leistung [3], wogegen jedoch auch das Teilen der Embedder-Gewichte die Effektivität des Vortrainings hemmt und für ein „Tauziehen“ sorgt [5], indem eine Optimierung der Embedder-Parameter in die Richtung zweier auseinanderliegender Optima wie folgt stattfindet [5]:

$$g_E = \frac{\partial L_{MLM}}{\partial E} + \lambda \frac{\partial L_{RTD}}{\partial E}$$

Dagegen löst das *Gradient-Disentangled Embedding Sharing* dieses Problem, indem immer noch ein Teilen des Embeddings E_G für Generator und Diskriminator erfolgt, jedoch nun das Embedding E_D mit einer zusätzlichen Gewichtsmatrix E_Δ dargestellt wird und damit mit einem *Stop-Gradient-operator* sg das Embedding E_D separat von E_G optimiert wird [5]:

$$E_D = sg(E_G) + E_\Delta$$

Hierbei ist lediglich zu bemerken, dass E_Δ als Nullmatrix initialisiert und im Diskriminator anstelle von E_G und dass E_Δ nach dem Vortraining zu E_G addiert wird. Ebenso ist zu beachten, dass DeBERTaV3 insgesamt etwa 185 Millionen Parameter besitzt, von denen rund 98 Millionen Parameter für das Embedding benötigt werden. Dies kommt daher, dass der Vokabeldatensatz von DeBERTaV3 mit 128.000 Tokens bei Token-Embedding-Vektoren der Größe 768 deutlich mehr Parameter als BERT für einen Vokabelsatz von etwa 30.000 Token beansprucht. Jedoch besitzt DeBERTaV3 im Gegensatz zu DeBERTa genau dieselbe Anzahl an Modell-Parametern wie das BERT-Modell, da ab dem Modell DeBERTaV2 in der Implementation [24] nun die Gewichte der Matrizen W_c^K, W_r^K und W_c^Q, W_r^Q in den Matrizen W^K und W^Q geteilt werden. Zudem nutzt DeBERTaV3 eine neue Methode des relativen Embeddings, das nun die Token-Distanz relativ zur Sequenz-Mitte ermittelt, womit $P \in \mathbb{R}^{2k \times d}$ zu $P \in \mathbb{R}^{k \times d}$ reduziert wird [24].

2.4 Das DetIE-Modell

Das zurzeit mit leistungsfähigste und zugleich schnelle OIE-Modell DetIE [15] gehört zu den Sequence-Labeling-Modellen, womit es die gegebenen Tokens klassifiziert, um daraus Extraktionen des Schemas (*Subjekt; Relation; Objekt*) zu extrahieren. Im Gegensatz zu den iterativen Methoden von IMojIE [16] und OpenIE6 [12] vermeidet DetIE ein iteratives sowie autoregressives Extrahieren gänzlich, indem der DetIE-Decoder alle Tokens innerhalb einer Tensorrechnung $N = 20$ potentielle Extraktionen klassifiziert und daraus daraus die Extraktionen zusammenbaut.

DetIE besteht aus einem BERT-Encoder, von welchen die unteren 8 der 12 Schichten nicht trainiert werden und als *frozen Layers* bezeichnet werden. Zudem besitzt DetIE einen untrainierten zweischichtigen Transformer-Encoder als Decoder [25], dessen Input und Output jeweils durch ein neuronales Netzwerk umgeformt wird.

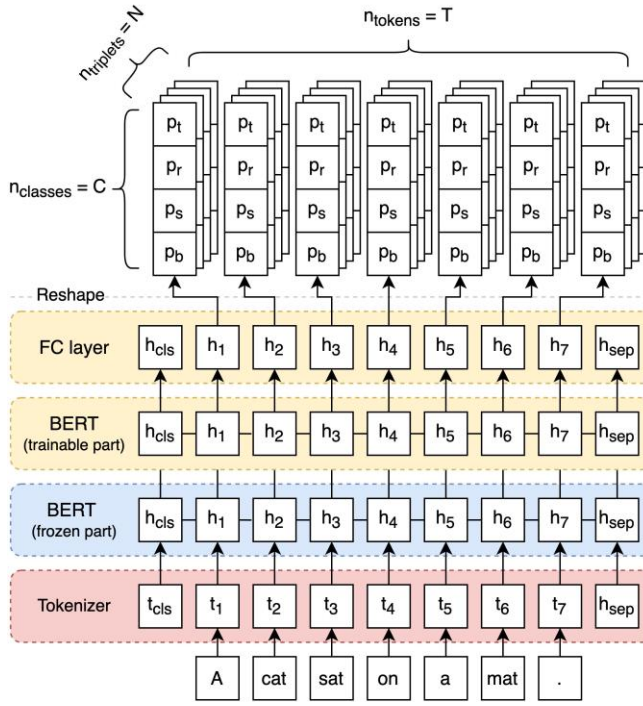


Abb. 5: Die Darstellung der DetIE-Modellstruktur einschließlich des Output-Tensors [15].

Schließlich wird aus dem Output-Tensor $P_{out} \in \mathbb{R}^{T \times N \times C}$ mit den Output-Wahrscheinlichkeiten der 4 Klassen *Background*, *Subject*, *Relation* und *Object* über Nutzung die vorhergesagte Klasse bestimmt und aus einer verbleibenden Matrix $C_{out} \in \mathbb{R}^{T \times N}$ der Klassen die Extraktionen wie folgt zusammengebaut: Aus den 20 potentiellen Sequenzen werden alle Sequenzen entfernt, bei denen mindestens eine der Token-Klassen *Subject*, *Relation* oder *Object* gänzlich fehlt [25], ebenfalls werden alle Background-Tokens entfernt und danach die Extraktionen nach dem Schema (Subject; Relation; Object) zusammengesetzt. Hierbei wird angenommen, dass alle Extraktionsbestandteile ohne Lücken zueinander existieren [25], sodass für jedes der Tags eine Spanne mit Start- und End-Index aus den Klassifikationen entnommen wird und daraus die Extraktionen zusammengebaut werden.

2.4.1 Order-Agnostic Loss

Das trainieren der Sequence Labeling Modelle erweist sich oftmals als schwierig, da Ähnlichkeits Metriken zwischen Extraktionen und den gegebenen Lösungsextraktionen nur schwer definierbar sind [15]. Hierfür nutzt DetIE die eingeführte Intersection-over-Union-Metrik, wobei mithilfe der Ungarischen Methode [26], die optimale bijektive Zuordnung zwischen den M gegebenen Lösungsextraktionen und den N generierten Extraktionen basierend mit der Gewichtung dieser IoU-Metrik ermittelt wird. Die IoU-Metrik berechnet sich dabei wie folgt [15]:

$$IoU_{nm} = \frac{I_{nm}}{U_{nm}} = \frac{\sum_{t,c} p_{tnc} l_{tmc}}{\sum_{t,c} p_{tnc} + \sum_{t,c} l_{tmc} - \sum_{t,c} p_{tnc} l_{tmc}}$$

Hierbei steht p_{tnc} für eine einzelne Vorhergesagte-Wahrscheinlichkeit der Klasse c für einen Token t einer potentiellen Extraktion n ; und l steht für einen *One-Hot-Tensor* der 4 Klassen einer vorgegebenen Extraktion m . Bei vollständiger Übereinstimmung beträgt $I_{nm} = 1$, womit dann auch gilt, dass $IoU_{nm} = 1$, denn für einen Token t betragen die konstanten Terme $\sum_c p_{tnc}$ und $\sum_c l_{tmc}$ immer jeweils 1. Wenn schließlich die geeigneten n - m Paare zugeordnet wurden, wird das DetIE-Modell mit dem Cross-Entropy-Fehler der optimal zugeordneten Extraktions-Lösungs-Paare optimiert.

3. Verbesserung des DetIE-Modells mittels Encoder-Austausch

3.1 Experimentbeschreibung

3.1.1 Server und Hardware

Das Trainieren und Verwenden von Transformer-Modellen erfordert eine große Menge an Arbeitsspeicher und Rechenleistung. Hierfür stellt die Universität Leipzig einen Server mit diesen zwei Grafikkarten zur Verfügung: Eine NVIDIA TITAN RTX mit 24 GB RAM und eine NVIDIA TITAN V mit 12 GB RAM. Damit während der Berechnungszeit keine dauerhafte Verbindung zwischen dem Heimcomputer und dem Server bestehen muss, wird ebenfalls ein GNU-Screen verwendet und das Tool WinSCP [27] wird für eine erleichterte Dateiarbeit im Serververzeichnis verwendet.

3.1.2 Code des DetIE-Modells

Der gesamte benutzte öffentlich zugängliche Code für DetIE ist in GitHub verfügbar [28] und basiert auf den Bibliotheken PyTorch, Pytorch-Lightning [29], Hydra [30] sowie der Bibliothek PyTorch-Transformers [31]. Dabei ist PyTorch eine grundlegende Bibliothek für das maschinelle Lernen, wogegen Hydra den Aufbau von sehr komplexen Anwendungen mit der Enbindung von Config-Dateien sehr verallgemeinert. PyTorch-Lightning vereinfacht hierbei den PyTorch-Code durch ein flexibles Framework aus vorimplementierten PyTorch-Modell-Funktionen. Die Bibliothek PyTorch-Transformers beinhaltet hingegen alle benötigten Implementationen der Transformer BERT, RoBERTa, ELECTRA, DeBERTa und DeBERTaV3.

Am Modell-Code mussten keine Änderungen vorgenommen werden, sodass das GitHub-Repository <https://github.com/HenningBeyer/DetIE-with-ELECTRA> ausschließlich zum Einsehen der Messdaten, Extraktionen und Trainings-Screen-Logs angelegt wird. Zusätzlich können die Scripts und einige Messwerte der vorherigen Arbeit für IMoJIE unter <https://github.com/HenningBeyer/IMoJIE-Faster-Copy-Attention> eingesehen werden.

3.1.3 Datensätze und Bewertungsvorgang

Für alle Details zu den verwendeten Datensätzen und der Bewertungsschemas wird mit auf die ausführliche Beschreibung im DetIE-Forschungsbericht verwiesen [25] und die wichtigsten Merkmale der Datensätze in Tab. 1 zusammengefasst.

Aufteilung	Datensatz	Sätze	Tupel
Training	IMoJIE	91.725	190.611
	LSOIE	34.780	100.862
Testung	CaRB	641	2.715
	OIE2016	3.200	10.359

Tab. 1: Die Auflistung der Trainings- und Testdatensatzgrößen der verwendeten Datensätze IMoJIE [16], LSOIE [32], CaRB [33] sowie OIE2016 [34].

Für das Training werden wie im DetIE-Forschungsbericht jeweils die zwei Trainingsdatensätze LSOIE [32] und der IMoJIE-Trainingsdatensatz [16] verwendet, wobei der LSOIE-Datensatz aus einem annotierten QA-SRL-Datensatz erstellt und der IMoJIE-Datensatz aus mehreren von OIE-Modellen extrahierten Wikipedia-Extraktionen gewonnen wurde. Auf der Seite der Testdatensätze wird betont, dass ausschließlich der CaRB-Datensatz mit exakt 641 Testsätzen zur Bestimmung der Modell-Leistungen verwendet wird und nur zur Messung der Modell-Geschwindigkeits der OIE2016-Test-Datensatz. Für die Testung der Modell-Leistung erfolgt eine Bewertung nach den insgesamt 4 Bewertungsschemas CaRB [33], CaRB(1-1) [12], OIE2016 [34] sowie WiRe57 [35], die alle nach der Zuordnung von Vorhersage- und Lösungs-Extraktionen die Bewertungsmetriken wie folgt berechnen und sich nur im Zuordnungsschema unterscheiden:

$$Prec = \frac{\sum_t |G_t \cap P_t|}{\sum_t |P_t|}, \quad Rec = \frac{\sum_t |G_t \cap P_t|}{\sum_t |G_t|} \quad \text{und} \quad F1 = \frac{2 * Prec * Rec}{Prec + Rec}$$

Hierbei stellen G und P die einander zugeordneten Extraktions-Multimengen der Gold- und Prediction-Tokens dar, wobei sich der F1-Score aus dem harmonischen Mittel der Precision- und Recall-Scores berechnet. Die Werte der AUC-Metrik werden für DetIE approximiert, da es keine für die Berechnung nötigen Konfidenz-Werte bei Vorhersagen liefert [25]. Unter dem Bewertungsschema WiRe57 existiert dabei keine originale Implementation zur Berechnung der AUC-Scores.

3.2 Durchführung der Experimente

Alle DetIE-Modelle werden jeweils mit unterschiedlichen Encoder-Modellen auf der Batch-Size 32 für 30 Epochen trainiert, wobei das Modell mit den besten Validierungsergebnissen für die weitere Testung verwendet wird. Durch die gleichartige Transformer-Struktur aller Encoder-Modelle wird keine Änderung am Modell-Code benötigt. Bei der Durchführung werden zudem alle Modelle mit BERT-, RoBERTa- und ELECTRA-Encoder in insgesamt fünfmal trainiert und das Modell mit der höchsten Leistung für die jeweilige Tabelle ausgewählt. Denn die Leistungs-Intervalle der Modelle überschneiden sich, sodass bei der geringere Leistungen als im DetIE-Forschungsbericht auftreten. Schließlich erhält man so die folgenden Messergebnisse:

Modell	Encoder	CaRB-Bewertungsschemas							
		CaRB		CaRB(1-1)		OIE16-C		WiRe57-C	
		F1	AUC	F1	AUC	F1	AUC	F1	
DetIE _{IMoJIE}	BERT	52,1	37,0	40,7	24,7	56,3	39,0	36,4	
DetIE _{IMoJIE+IGL-CA}	BERT	47,3	35,7	<u>43,2</u>	<u>29,8</u>	<u>67,4</u>	<u>54,2</u>	<u>37,7</u>	
DetIE _{IMoJIE}	RoBERTa	<u>52,6</u>	<u>37,2</u>	41,0	24,8	56,9	39,5	37,1	
DetIE _{IMoJIE+IGL-CA}	RoBERTa	48,1	35,9	43,8	30,0	67,7	54,0	39,0	
DetIE _{IMoJIE}	ELECTRA	52,7	37,9	41,5	25,5	58,3	41,1	37,0	
DetIE _{IMoJIE+IGL-CA}	ELECTRA	46,8	35,7	43,0	30,0	67,7	55,1	37,5	

Tab. 2: Die Auflistung der Maximal-Leistungen des DetIE-Modells bei Trainieren mit den Encodern BERT, RoBERTa und ELECTRA. Das Training erfolgt mit dem IMoJIE-Datensatz und die Testung mit dem CaRB-Datensatz von 641 Sätzen bei den vier verschiedenen, oben aufgelisteten Bewertungsschemas. Die besten und zweitbesten Ergebnisse werden jeweils fett gedruckt bzw. unterstrichen dargestellt.

Modell	Encoder	CaRB-Bewertungsschemas							
		CaRB		CaRB(1-1)		OIE16-C		WiRe57-C	
		F1	AUC	F1	AUC	F1	AUC	F1	
DetIE _{LSOIE}	BERT	35,1	24,6	32,2	20,7	65,9	<u>51,4</u>	28,4	
DetIE _{LSOIE+IGL-CA}	BERT	36,9	<u>26,9</u>	34,2	23,0	63,4	50,4	30,1	
DetIE _{LSOIE}	RoBERTa	35,4	24,0	32,8	20,8	63,0	47,9	29,6	
DetIE _{LSOIE+IGL-CA}	RoBERTa	38,1	26,6	35,4	<u>23,1</u>	63,7	50,0	31,5	
DetIE _{LSOIE}	ELECTRA	35,0	24,6	32,3	20,9	<u>65,7</u>	51,2	29,1	
DetIE _{LSOIE+IGL-CA}	ELECTRA	<u>37,1</u>	27,2	<u>34,7</u>	23,6	65,9	53,5	<u>31,1</u>	

Tab. 3: Die Auflistung aller Messergebnisse bei Training mit dem LSOIE-Datensatz [32] und den gleichen in Tab. 2 genannten Bedingungen.

Die IGL-CA-Komponente [12] stellt dabei lediglich eine Zusatzkomponente dar, die mit einem deterministischen Algorithmus für eine Auftrennung aller Konjunktionsstrukturen innerhalb der Input-Sätze sorgt, sodass das DetIE-Modell insgesamt aus einer höheren Anzahl an einfacheren Sätzen extrahiert. Alle Modellen werden erst nach dem Training mit einer IGL-CA-Komponente kombiniert und benötigen kein erneutes Training.

Modell	Encoder	CaRB-Bewertungsschemas								Geschwind.
		CaRB		CaRB(1-1)		OIE16-C		WiRe57-C		
		F1	AUC	F1	AUC	F1	AUC	F1		(Sätze je Sek.)
IMoJIE *	BERT	<u>53,5</u>	33,3	41,4	22,2	56,8	39,6	36,0		2,6
IGL-OIE *	BERT	52,4	33,7	41,1	22,9	55,0	36,0	34,9		142,0
CIGL-OIE *	BERT	54,0	35,7	42,8	24,6	59,2	40,0	36,8		142,0
OpenIE6 *	BERT	52,7	33,7	46,4	26,8	65,6	48,4	40,0		31,7
DetIE _{IMoJIE} *	BERT	52,1	36,7	40,1	24,0	56,0	38,7	36,0		708,6
DetIE _{IMoJIE+IGL-CA} *	BERT	47,3	35,1	43,1	29,3	67,7	54,0	37,8		112,2
DetIE _{IMoJIE}	BERT	52,1	37,0	40,7	24,7	56,3	39,0	36,4		500,0
DetIE _{IMoJIE+IGL-CA}	BERT	47,3	35,7	43,2	<u>29,8</u>	<u>67,4</u>	<u>54,2</u>	37,7		203,2
DetIE _{IMoJIE}	RoBERTa	52,6	<u>37,2</u>	41,0	24,8	56,9	39,5	37,1		522,1
DetIE _{IMoJIE+IGL-CA}	RoBERTa	48,1	35,9	<u>43,8</u>	30,0	67,7	54,0	<u>39,0</u>		206,4
DetIE _{IMoJIE}	ELECTRA	52,7	37,9	41,5	25,5	58,3	41,1	37,0		511,9
DetIE _{IMoJIE+IGL-CA}	ELECTRA	46,8	35,7	43,0	30,0	67,7	55,1	37,5		205,9

Tab. 4: Die Auflistung aller Messergebnisse der Tab. 2 im Vergleich zu den bisherig leistungsfähigsten Modellen der OIE. Alle mit einem * markierten Modelle sind aus dem DetIE-Forschungsbericht [15] übernommen.

Der Grund für die höhere Modell-Geschwindigkeit der mit Nutzung der IGL-CA-Komponente liegt in der Nutzung einer leistungsfähigeren CPU begründet: In den Experimenten wird eine Intel® Xeon® Gold 6140 mit 18 Prozessorkernen benutzt, wodurch die Laufzeit der langsameren nicht-parallelierten Prozesse stark verringert wird.

3.3 Auswertung der Ergebnisse

Aus den Tabellen kann entnommen werden, wie die Modell-Leistungen mit der Nutzung von RoBERTa- und ELECTRA-Encodern für das DetIE-Modell bei nahezu gleicher Extraktionsgeschwindigkeit ansteigt und sich mit beiden Encodern verbessert. Dabei erzielen alle DetIE-Modelle in Tab. 2 ohne IGL-CA-Komponente stetig von BERT zu RoBERTa zu ELECTRA eindeutige Leistungsverbesserungen, wobei Modelle der Tabelle 2. mit IGL-CA-Komponente auch Verbesserungen erzielen, aber in einem geringeren Ausmaß. In Tab. 3 bei Training mit dem LSOIE-Datensatz ist dies umgekehrt: Die Modelle mit RoBERTa- und ELECTRA-Encoder ohne IGL-CA-Komponente erzielen insgesamt nur geringe Verbesserungen, während die Modelle mit IGL-CA-Komponente die Leistung eindeutig verbessern.

Dieser Unterschied entsteht durch die unterschiedliche Trainingsdatensatzgröße der Datensätze und der Vereinfachung der Input-Sätze durch die Satzauftrennung der IGL-CA-Komponente: Der Decoder und die 4 oberen trainierten Encoder-Layer von DetIE können mit den kleineren LSOIE-Datensatz nicht vollständig an die Extraktionsaufgabe angepasst werden und demnach scheitert das Modell nicht am Sprachverständnis, sondern an der Fähigkeit, das Sprachwissen in die nötigen Extraktionen zu konvertieren und das Sprachwissen zu nutzen. Dagegen erzielen die DetIE-Modelle des LSOIE-Datensatzes mit einer IGL-CA-Komponente merkbare Verbesserungen, da die Extraktionsaufgabe durch die Auftrennung der Konjunktionsstrukturen so vereinfacht wurde, dass der Decoder das Sprachwissen eines kürzeren Satzes vollständig versteht und bei der Extraktion aus kürzeren Satzbestandteilen durch ein besseres Sprachwissen profitiert.

Mit dem IMoJIE-Datensatz sind dagegen genug Trainingsdaten gegeben, damit selbst die hypotaktischen Sätze gut verstanden werden und Verbesserungen durch Ausnutzung des besseren Sprachwis-

sens gegeben sind. Dabei ist hier nur eine kleine Verbesserung mit IGL-CA-Komponente zu beobachten, da die Teilsätze mit den gegebenen Trainingsdaten so gut verstanden werden, dass hierfür das Sprachwissen des BERT-Modells ausreichend ist und kein tieferes Sprachverständnis benötigt wird.

Bei Berücksichtigung dieser Faktoren erzielen die Modelle mit ELECTRA-Encoder ohne IGL-CA merkliche Verbesserungen von durchschnittlich 1,0 F1- und 1,3 AUC-Punkten für den IMoJIE-Datensatz sowie durchschnittlich 1,1 F1- und 1,3 AUC-Punkte für den LSOIE Datensatz. Diese Ergebnisse sind äußerst relevant, wenn man bedenkt, dass keine Verminderung der Extraktionsgeschwindigkeit vorliegt und dieser Encoder-Austausch sehr simpel für jedes andere OIE-Modell vorgenommen werden kann. Andernfalls, wenn das Modell nicht maßgeblich durch Sprachwissen profitieren kann, so sind die beobachteten Leistungsverbesserungen nahezu identisch oder minimal wie in den anderen Fällen. Gleichmaßen ist gegenüber dem momentan besten Modell des DetIE-Forschungsberichtes [15] – dem DetIE-Modell mit IGL-CA und Training mit dem IMoJIE-Datensatz – keine oder kaum eine Verbesserung zu beobachten.

3.3.1 Niedrige Leistungen für DeBERTa-Encoder

Die Modell-Leistungen für Modelle mit DeBERTa- und DeBERTaV3-Encodern wurden wegen ihrer geringen Leistung nicht aufgeführt, wobei diese niedrigen Leistungen durch die geteilte Relative-Embedding-Matrix zustande kommen, was bei den Versuchen zwar berücksichtigt wurde, aber unter einem Fehlschluss. Nämlich wurde bis zum Schreiben des Fazits vermutet, dass das relative Embedding die Leistungen des DetIE-Modells mindern würden. Denn ein zusätzliches Experiment, das als Beleg für diese Hypothese dienen soll, berücksichtigt nicht, dass die oberen 4 trainierten Transformer-Layer ein separat optimiertes relatives Embedding benötigen, um die Sprachinformationen der unteren 8 Layer in Extraktionswissen für den Decoder zu konvertieren.

Indem bei Trainieren aller DetIE-Modelle hiermit keine Leistungssteigerung für DeBERTa-Encoder beobachtet werden kann, kam es zu dem Fehlschluss, der erst am Ende bei Schreiben eines Lösungsvorschlags innerhalb des Fazits und der Unterteilung der Encoder-Informationen in Sprach- und Extraktionswissen ersichtlich wurde. Direkt nach den Vorprüfungen sollen die korrekten Messwerte von DetIE mit DeBERTa- und DeBERTaV3-Encoder ergänzt werden.

4. Fazit

Insgesamt werden mit den Experimenten dieser Forschungsarbeit sehr relevante Leistungsverbesserungen von bis zu 1,0 F1- und 1,3 AUC-Punkten ohne Verminderung der Extraktionsgeschwindigkeit erzielt, wobei Verbesserungen dieser Art ebenfalls für jegliche OIE-Modelle simpel reproduzierbar sind, sodass die Verbesserungen auch für zukünftige OIE-Modelle gelten. Damit wurde in dieser Forschungsarbeit eine wichtige, sowohl praktische als auch theoretische Grundlage erarbeitet, an der sich die zukünftige Forschung sowie zukünftige Projekte orientieren können.

Zudem werden anhand der Auswertung wichtige Faktoren ausfindig gemacht, die die Höhe der Verbesserungen bestimmen und auch auf andere Sprachaufgaben und Modell-Strukturen angewandt werden können. Nämlich bestimmt die Einfachheit der Aufgabe, ob ein höheres Sprachverständnis zu Verbesserungen führt und die Menge an Trainingsdaten bestimmt in Relation zur Einfachheit, ob der Decoder das Sprachwissen verstehen und damit ausnutzen kann. Diese Faktoren schränken die bislang noch beibehaltene Hypothese ein, dass besser vortrainierte Encoder die Leistungen von DetIE und anderen OIE-Modellen verbessern. Jedoch soll diese Hypothese noch vollständig mit DeBERTa- und DeBERTaV3-Encoder bewiesen werden.

Literaturverzeichnis

- [1] A. Vaswani, N. Shazeer, P. Niki, U. Jakob, J. Llion, G. N. Aidan, K. Lukasz und P. Illia, „Attention Is All You Need”, arXiv:1706.03762v5, 2017.
- [2] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer und V. Stoyanov, „RoBERTa: A Robustly Optimized BERT Pretraining Approach”, arXiv:1907.11692v1, 2019.
- [3] K. Clark, M.-T. Luong, Q. V. Le und C. D. Manning, „ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators”, arXiv:2003.10555, 2020.
- [4] P. He, X. Liu, J. Gao und W. Chen, „DeBERTa: Decoding-enhanced BERT with Disentangled Attention”, arXiv:2006.03654v6, 2021.
- [5] P. He, J. Gao und W. Chen, „DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing”, arXiv:2111.09543, 2021.
- [6] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy und S. R. Bowman, „SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems”, arXiv:1905.00537v3, 2019.
- [7] P. He, X. Liu, J. Gao und W. Chen, „Microsoft DeBERTa surpasses human performance on the SuperGLUE benchmark”, Microsoft, 6 Januar 2021. [Online]. <https://www.microsoft.com/en-us/research/blog/microsoft-deberta-surpasses-human-performance-on-the-superglue-benchmark/>. [11. Dezember 2021].
- [8] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li und P. J. Liu, „Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”, arXiv:1910.10683, 2020.
- [9] Y. Sun, S. Wang, S. Feng, S. Ding, C. Pang, J. Shang, J. Lui, X. Chen, Y. Zhao, Y. Lu, W. Liu, Z. Wu, W. Gong, J. Liang, Z. Shang, P. Sun, W. Liu, X. Ouyang, D. Yu, H. Tian, H. Wu und H. Wang, „ERNIE 3.0: Large-scale Knowledge Enhanced Pre-training for Language Understanding and Generation”, arXiv:2107.02137v1, 2021.
- [10] R. Thoppilan, D. D. Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, Y. Li, H. Lee, H. S. Zheng, A. Ghafouri, M. Menegali, Y. Huang, M. Krikun, D. Lepikhin, J. Qin, D. Chen, Y. Xu, Z. Chen, A. Roberts, M. Bosma, V. Zhao, Y. Zhou, C. C. Chang, I. Krivokon, W. Rusch, M. Pickett, P. Srinivasan, L. Man, K. Meier-Hellstern, M. R. Morris, T. Doshi, R. D. Santos, T. Duke, J. Soraker, B. Zevenbergen, V. Prabhakaran, M. Diaz, B. Hutchinson, K. Olson, A. Molina, E. Hoffman-John, J. Lee, L. Aroyo, R. Rajakumar, A. Butryna, M. Lamm, V. Kuzmina, J. Fenton, a. Cohen, R. Bernstein, R. Kurzweil, B. Aguera-Arcas, C. Cui, M. Croak, E. Chi und Q. Le, „LaMDA: Language Models for Dialog Applications”, arXiv:2201.08239v3, 2022.
- [11] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandish, A. Radford, I. Sutskever und D. Amodei, „Language Models are Few-Shot Learners”, OpenAI, arXiv:2005.14165, 2020.
- [12] K. Kolluru, V. Adlakha, S. Aggarwal, Mausam und S. Chakrabarti, „OpenIE6: Iterative Grid Labeling and Coordination Analysis for Open Information Extraction”, arXiv:2010.03147v1, 2020.
- [13] L. Cui, F. Wei und M. Zhou, „Neural Open Information Extraction”, arXiv:1805.04270v1, 2018.
- [14] D.-T. Vo und E. Bagheri, „Open Information Extraction”, arXiv:1607.02784v1, 2016.

- [15] M. Vasilkovsky, A. Alekseev, V. Malykh, I. Shenbin, E. Tutubalina, D. Salikhov, M. Stepnov, A. Chertok und S. Nikolenko, „DetIE: Multilingual Open Information Extraction Inspired by Object Detection”, arXiv:2206.12514, 2022.
- [16] K. Kolluru, S. Aggarwal, V. Rathore, Mausam und S. Chakrabarti, „IMoJIE: Iterative Memory-Based Joint Open Information Extraction”, arXiv:2005.08178v1, 2020.
- [17] J. Gu, Z. Lu, H. Li und V. O. Li, „Incorporating Copying Mechanism in Sequence-to-Sequence Learning”, arXiv:1603.06393v3, 2016.
- [18] J. B. Ba, R. J. Kiros und G. E. Hinton, „Layer Normalization”, arXiv:1607.06450v1, 2016.
- [19] J. Devlin, M.-W. Chang, K. Lee und K. Toutanova, „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, arXiv:1810.04805v2, 2019.
- [20] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville und Y. Bengio, „Generative Adversarial Networks”, arXiv:1406.2661v1, 2014.
- [21] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffmann, M. Dinculescu und D. Eck, „Music Transformer”, arXiv:1809.04281v3, 2018.
- [22] P. Shaw, J. Uszkoreit und A. Vaswani, „Self-Attention with Relative Position Representations”, arXiv:1803.02155v2, 2018.
- [23] I. Loshchilov und F. Hutter, „Fixing Weight Decay Regularization in Adam”, or arXiv:1711.05101v3, 2019.
- [24] P. He, X. Liu, J. Gao und W. Chen, „DeBERTa: Decoding-enhanced BERT with Disentangled Attention”, [Online]. <https://github.com/microsoft/DeBERTa>. [06.12.2022].
- [25] M. Vasilkovsky, A. Alekseev, V. Malykh, I. Shenbin, E. Tutubalina, D. Salikhov, M. Stepnov, A. Chertok und S. Nikolenko, „DetIE: Multilingual Open Information Extraction Inspired by Object Detection”, [Online]. <https://github.com/sberbank-ai/DetIE>. [20.11.2022].
- [26] H. W. Kuhn, „The Hungarian method for the assignment problem”, Naval research logistics quarterly, 1955.
- [27] M. Prikryl, „WinSCP”, [Online]. <https://winscp.net/eng/download.php>. [13.12.2021].
- [28] K. Kolluru, „IMoJIE: Iterative Memory-Based Joint Open Information Extraction”, [Online]. <https://github.com/dair-iitd/imojie/tree/master/imojie/models>. [12.12.2021].
- [29] Falcon, W., & The PyTorch Lightning team, „PyTorch Lightning”.
- [30] K. Nagrecha und A. Kumar, „Hydra: A System for Large Multi-Model Deep Learning”, arXiv:2110.08633, 2022.
- [31] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz und J. Brew, „HuggingFace's Transformers: State-of-the-art Natural Language Processing”, arXiv:1910.03771, 2020.
- [32] J. Solawetz und S. Larson, „LSOIE: A Large-Scale Dataset for Supervised Open Information Extraction”, arXiv:2101.11177, 2021.
- [33] S. Bhardwaj, S. Aggarwal und M. , „CaRB: A Crowdsourced Benchmark for Open IE”, 2019. [Online]. <https://aclanthology.org/D19-1651.pdf>. [13. Dezember 2021].
- [34] G. Stanovsky und I. Dagan, „Creating a Large Benchmark for Open Information Extraction”, 2016.

- [35] W. L  chelle, F. Gotti und P. Langlais, „WiRe57 : A Fine-Grained Benchmark for Open Information Extraction“, arXiv:1809.08962, 2019.
- [36] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma und R. Soricut, „ALBERT: A Lite BERT for Self-supervised Learning of Language Representations“, arXiv:1909.11942v6, 2020.
- [37] J. Steinwendner und R. Schwaiger, Neuronale Netze Programmieren mit Python, 2. Auflage Hrsg., Bonn: Rheinwerk Verlag, 2020.
- [38] Y. Ro, Y. Lee und P. Kang, „Multi2OIE: Multilingual Open Information Extraction Based on Multi-Head Attention with BERT“, arXiv:2009.08128v2, 2020.
- [39] A. G  ron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, 2. Auflage der Fr  hver  ffentlichung Hrsg., Sebastopol: O'Reilly, 2019.
- [40] D. Silver, T. Hubert und J. Schrittwieser, „Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm“, arXiv:1712.01815v1, 2017.
- [41] A. Gulli, A. Kapoor und S. Pal, Deep Learning with Tensorflow 2 and Keras, 2. Auflage Hrsg., Birmingham: Packt Publishing Ltd., 2019.
- [42] Y. H. Liu, Python Machine Learning By Example, 3. Auflage Hrsg., Birmingham: Packt Publishing Ltd., 2020.
- [43] F. Gers, „Recurrent nets that time and count“, ResearchGate: https://www.researchgate.net/publication/3857862_Recurrent_nets_that_time_and_count, 2000.
- [44] R. Rana, „Gated Recurrent Unit (GRU) for Emotion Classification from Noisy Speech“, in *arXiv:1612.07778v1*, 2016.
- [45] S. Hochreiter, „Long Short-term Memory“, 1997. [Online]. ResearchGate: https://www.researchgate.net/publication/13853244_Long_Short-term_Memory. [12. Dezember 2021].
- [46] D. Z. Liu und G. Singh, „A Recurrent Neural Network Based Recommendation System“, [Online]. <https://cs224d.stanford.edu/reports/LiuSingh.pdf>. [13. Dezember 2021].
- [47] K. Al-Sabahi, Z. Zuping und Y. Kang, „Bidirectional Attentional Encoder-Decoder Model and Bidirectional Beam Search for Abstractive Summarization“, arXiv:1809.06662v1, 2018.
- [48] O. Vinyals, M. Fortunato und N. Jaitly, „Pointer Networks“, arXiv:1506.03134v1.
- [49] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. Liu, M. Peters, M. Schmitz und L. Zettlemoyer, „AllenNLP: A Deep Semantic Natural Language Processing Platform“, arXiv:1803.07640v2, 2018.
- [50] J. Tang, Y. Lu, X. Lin, L. Sun, X. Xiao und H. Wu, „Syntactic and Semantic-driven Learning for Open Information Extraction“, arXiv:2103.03448v1, 2021.
- [51] J. Chung, C. Gulcehre, K. Cho und Y. Bengio, „Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling“, arXiv:1412.3555v1, 2014.
- [52] S. Ravichandiran, Deep Reinforcement Learning with Python, 2. Auflage Hrsg., Birmingham: Packt Publishing Ltd., 2022.
- [53] „Efficiently and effectively scaling up language model pretraining for best language representation model on GLUE and SuperGLUE“, [Online]. [27.11.2021].

- [54] R. Jozefowicz, W. Zaremba und I. Sutskever, „An Empirical Exploration of Recurrent Network Architectures”, 2015. [Online]. <http://proceedings.mlr.press/v37/jozefowicz15.pdf>. [13.12.2021].
- [55] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy und S. R. Bowman, „SuperGLUE Leaderboard”, [Online]. <https://super.gluebenchmark.com/leaderboard/>. [10.04.2022].
- [56] B. Zoph, I. Bello, S. Kumar, N. Du, Y. Huang, J. Dean, N. Shazeer und W. Fedus, „Designing Effective Sparse Expert Models”, arXiv:2202.08906, 2022.
- [57] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, 1958, p. 386–408.
- [58] K. Cho, B. v. Merriënboer, C. Gulcehre, D. Bahdanau, F. Bourgares, H. Schwenk und Y. Bengio, „Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”, arXiv:1406.1078v3, 2014.
- [59] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba und S. Fidler, „Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books”, arXiv:1506.06724, 2015.
- [60] D. K, „The Indicator Function”, [Online]. https://dk81.github.io/dkmathstats_site/prob-indicator.html. [24.08.2022].
- [61] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy und S. R. Bowman, „GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”, arXiv:1804.07461v3, 2019.
- [62] M. Schuster und K. Nakajima, „Japanese and Korean voice search”, <https://ieeexplore.ieee.org/document/6289079>, 2012.
- [63] D. P. Kingma und J. Ba, „Adam: A Method for Stochastic Optimization”, <https://arxiv.org/abs/1412.6980v9>, 2017.

Danksagung

Ich möchte mich allerherzlichst bei Elke Katz dafür bedanken, dass sie mir dieses Projekt mit diesem schwierigeren Thema ermöglicht hat. Sie half mir, den geeigneten Projektbetreuer Prof. Andreas Maletti zu finden und schaffte außerdem vielen anderen Schülern jedes Jahr gute Möglichkeiten, an Wettbewerben wie Jugend forscht teilzunehmen. Ich bedanke mich ebenfalls herzlichst bei meinem außerschulischen Projektbetreuer Prof. Andreas Maletti, mit welchem ich mich während des Projektes immer sehr gut verständigen konnte und der mich auch bei Schwierigkeiten immer gerne unterstützte. Ebenfalls bedanke ich mich sehr über die viele Unterstützung meiner schulischen Betreuerin Romy Schachoff, die mich während der Besonderen maßgeblich Lernleistung unterstützt hat.

Der Universität Leipzig möchte ich außerdem einen großen Dank dafür aussprechen, dass sie mir einen Server mit zwei leistungsfähigen Grafikkarten kostenlos für meine Versuche zur Verfügung gestellt haben. Ohne diesen leistungsfähigen Servern wäre diese Forschungsarbeit nicht umsetzbar gewesen.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Forschungsarbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe. Dazu zählt insbesondere, dass kein einziger Teil dieser Forschungsarbeit mit dem Tool ChatGPT geschrieben wurde und dieses Tool nicht im Zusammenhang mit dieser Arbeit benutzt wurde.

Ebenfalls versichere ich, dass alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht wurden.

Leipzig, 23.01.2023 (Ort, Datum)

Hemming Beyer (Unterschrift)

Abkürzungsverzeichnis

Abkürzung	Bedeutung
ALBERT	A Lite BERT
AUC	Area Under the Curve
BERT	Bidirectional Encoder Representations from Transformers
CaRB	A Crowdsourced Benchmark
CUDA	Compute Unified Device Architecture
DeBERTa	Decoding-Enhanced BERT with Disentangled Attention
DetIE	Information Extraction inspired by Object Detection
ELECTRA	Efficiently Learning an Encoder that Classifies Token Replacements Accurately
ERNIE	Enhanced Representation through Knowledge Integration
F1	Metrik, welche aus dem harmonischen Mittel von Precision und Recall gebildet wird.
GAN	Generative Adversarial Network
GLUE	General Language Understanding
GPU	Graphical Processing Unit; Grafikkarte
GRU	Gated Recurrent Unit
IGL-CA	Iterative Grid Labeling Coordination Analyzer
IMoJIE	Iterative Memory-Based Joint Open Information Extraction
IoU	Intersection over Union
LSOIE	Large-Scale Dataset for Supervised Open Information Extraction
LSTM	Long Short-Term Memory
LaMBDA	Language Models for Dialog Applications
NLP	Natural Language Processing
NN	Neurales Netzwerk
OIE	Open Information Extraction
QA-SRL	Question Answer Semantic Role Labeling
RNN	Rekurrentes neuronales Netzwerk
RoBERTa	Robustly Optimized BERT Approach
Seq2seq-Modell	Sequence-to-Sequence-Modell bzw. Encoder-Decoder-Modell
WiRe57	57 annotierte Sätze aus Wikipedia-Artikeln und Newswire-Artikeln von Reuters