Arwed-Rossbach-Schule
Berufliches Gymnasium

Besondere Lernleistung

Im Fach Informatik- und Kommunikationssysteme

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, 23.02.2023

Kurzfassung

In dieser Besonderen Lernleistung wird im Bereich der Open Information Extraction (OIE) ein Encoder-Austausch am DetIE-Modell mit den effektiver vortrainierten Varianten RoBERTa, ELECTRA, DeBERTa und DeBERTaV3 vorgenommen. Hiermit können insgesamt 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 einfach für alle Modelle mit BERT-Encoder reproduzierbar und somit auch für zukünftige OIE-Modelle von großer Relevanz ist.

Dabei legt diese Besondere Lernleistung neben einer praktischen außerdem eine theoretische Grundlage zu Versuchen dieser Art, indem die Theorie aller behandelten Encoder zusammengefasst wird und zudem wichtige Verbesserungskriterien identifiziert werden: Die Sprachaufgabe benötigt genügend Komplexität und Trainingsdaten, um mit den verbesserten Sprachwissen der Encoder Verbesserungen zu erzielen. Ebenfalls benötigen OIE-Modelle ein absolutes Position-Embedding, weswegen die Encoder DeBERTa und DeBERTaV3 mit deren relativen Position-Embeddings als nicht anwendbar für die OIE bestimmt werden.

Messwerte und abgeänderte Skripte sind über https://github.com/HenningBeyer/DetIE-with-ELECTRA einsehbar und Änderungen im Code mit *hbeyer* markiert.

Inhaltsverzeichnis

1. Einleitung

In dieser Besonderen Lernleistung 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 Relationstupeln 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 Besondere Lernleistung 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 dank ihrer Parallelisierbarkeit und ihrer 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 Bestleistungen stetig vergrößert und im Vortraining wie bei den Modellen BERT [2], RoBERTa [3], ELECTRA [4], DeBERTa [5] und DeBERTaV3 [6] optimiert. So konnte im Januar 2021 die menschliche Sprachleistung zum ersten Mal überhaupt in der SuperGLUE-Benchmark [7] vom DeBERTa-Modell übertroffen werden [5], [8]: DeBERTa trainierte mit 1,5 Milliarden Parametern für 30 Tage auf 256 GPUs und konnte hiermit Googles T5-Modell [9] mit 11 Milliarden Parametern dank eines optimierten Vortrainings eindeutig übertreffen.

Trotz der Fortschritte im Sprachverständnis und dem Trainieren mit zunehmend mehr Parametern fehlt es den Modellen immer noch an Wissen, um die menschliche Sprache tiefgründiger zusammen mit dem Kontext zu verstehen und Informationen korrekt einordnen zu können. Hierfür wird zurzeit an Methoden geforscht, um den Modellen ein Allgemeinwissen zu verleihen, indem Modelle wie ERNIE 3.0 [10] zusätzlich mithilfe von faktenbasierten Wissen trainiert werden oder wie LaMDA [11] ihre Antworten auf Grundlage von Wissensdatenbanken generieren. Zudem können Modelle wie LaMDA oder ChatGPT [12] über ein eigenes Information-Retrieval-Systeme benötigte Informationen eigenständig aus Internetseiten entnehmen und damit ihre Antworten basierend auf Wissen 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 Relationstupeln nach dem Schema *(Subjekt; Relation; Objekt)* [13]. Mit diesen geordneten Informationen unterstützt dieser Bereich viele untergeordnete Aufgaben des NLP wie das Question Answering oder die Textzusammenfassung sowie ebenfalls jegliche Prozesse, die auf Wissensdatenbanken angewiesen sind. Hierbei wird die OIE zum automatischen Extrahieren aus Internet- sowie Text-Dokumenten angewendet, während besonders die Effektivität der verwendeten OIE-Modelle für die Menge und Qualität der Relationstupel verantwortlich ist.

Erstmalig wurde die OIE um 2007 mit dem Modell TEXTRUNNER eingeführt [14], [15] 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 vermeiden [16]. Generell werden die neuronalen OIE-Modelle in zwei Bereiche eingeordnet: der Sequence Generation und dem Sequence Labeling. Die Sequence Generation umfasst dabei Modell-Strukturen, die Extraktionen autoregressiv Wort für Wort generieren und dadurch niedrige Extraktionsgeschwindigkeiten erzielen. Dagegen umfassen Modelle des Sequence Labelings Modell-Strukturen, die jeden der Input-Tokens klassifizieren und anhand der Labels die Extraktionen nach der Klassifizierung zusammensetzen.

Beide dieser Bereiche haben das Problem, oft redundante Extraktionen zu generieren, wobei das aktuell leistungsstärkste Modell der Sequence Generation IMoJIE [17] einen iterativen Mechanismus benutzt, der für das Generieren jeder Extraktion wiederholt alle Informationen aller vorherigen Extraktionen erfasst, jedoch dadurch äußerst langsam extrahiert. Dagegen sind Lösungsansätze im Bereich des Sequence Labelings deutlich schneller, da alle Klassifikationen einer Extraktion in einer Modell-Ausführung anstatt Wort für Wort vorgenommen werden und zudem sind diese Ansatze einfacher, da keine zusätzlichen Maßnahmen gegen Folgefehler und keine Maßnahmen gegen das auftretende Out-of-Vocabulary-Problem [18] wie für Modelle der Sequence Generation getroffen werden müssen. Deswegen liegt der momentane Schwerpunkt der OIE-Forschung auf den Sequence-Labeling-Modellen wie OpenIE6 [13] oder DetIE [16].

Das Modell OpenIE6 [13] nutzt hierbei zur Redundanzvermeidung ein *iterative Grid Labeling* Mechanismus, der im Gegensatz zu IMoJIE mit nur einem einmaligen Encoding und einem nicht-autoregressiven Decoding bei höherer Leistung mehr als das Zehnfache der Extraktionsgeschwindigkeit von IMoJIE erreicht [13]. Das Modell DetIE [16] vermeidet hingegen jegliche iterative Struktur gänzlich, indem es gleich Klassifikationen für 20 potenzielle Extraktionen in einem Modell-Durchlauf vornimmt und damit bis zu 3-mal schneller als sein Vorgänger OpenIE6 bei erneuten Bestleistungen extrahiert [16].

Wie auch das DetIE-Modell machen alle aktuellen OIE-Modelle von einem BERT-Encoder gebrauch. Über eine Anwendung von den besser vortrainierten Encoder-Modellen RoBERTa [3], ELECTRA [4], DeBERTa [5] und DeBERTaV3 [6] wurde dabei in der OIE noch nicht diskutiert, obwohl durch einen solchen Encoder-Austausch Verbesserungen zu erwarten sind.

1.3 Motivation und Ziele der Forschungsarbeit

Diese Forschungsarbeit orientierte sich ursprünglich am IMoJIE-Modell [17] mit dem Ziel, dessen Encoder-Modell mit leistungsfähigeren Varianten auszutauschen. Hierfür wurden zuerst einfachere Experimente an einem leichteren Decoder-Austausch in einer Belegarbeit und stellte die Ergebnisse auch bei Jugend forscht vor. Hiermit konnte schließlich eine ineffizient programmierte Funktion als zentrale Ursache für die langsame Extraktionsgeschwindigkeit von IMoJIE ausfindig gemacht und erfolgreich optimiert werden, sodass die Laufzeit des IMoJIE-Modells von 247,4 auf 85,8 Sekunden für 641 Sätze ohne Leistungsbeeinträchtigung reduziert werden konnte.

Jedoch wird für diese Besondere Lernleistung ein Encoder-Austausch am aktuelleren DetIE-Modell [16] 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 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 auch anderen OIE-Modellen.

2. Theorie

Das behandelte DetIE-Modell benutzt als Encoder ein BERT-Modell und als Decoder neben neuronalen Netzwerken auch Transformer-Layer. Mit der nachfolgenden Theorie werden neben diesen Modellen alle der Encoder RoBERTa, ELECTRA, DeBERTa sowie DeBERTaV3 aufeinanderfolgend beschrieben, sodass die Intention und Durchführung der Experimente verständlich werden. Zuvor wird jedoch noch eine leicht verständliche Einführung in das maschinelle Lernen gegeben und ebenfalls der gesamte Sprachverarbeitungsvorgang in KI-Modellen vereinfacht dargestellt. Spätestens nach dem Verstehen der Transformer-Modelle bleiben die Grundkonzepte sehr ähnlich, womit die folgenden Encoder und das DetIE-Modell gut erschließbar sind.

2.1 Grundlagen des maschinellen Lernens

2.1.1 Künstliche neuronale Netze

Mit Sicherheit hat man schon einmal von den künstlichen neuronalen Netzen (KNN) etwas gehört: Sie sind die grundlegende Modell-Struktur des maschinellen Lernens und werden oft als Teil-Modelle in anderen Modellen wie den Transformern verwendet. Deren Fähigkeit, effizient einfache Daten zu erlernen, macht sie flexibel einsetzbar, wobei sie bei ausreichender Größe schnell die menschliche Leistung in eindeutig definierbaren Aufgaben übertreffen können. Einer der hierbei wohl bekanntesten Erfolge erzielt das Modell AlphaGo, welches den Go-Weltmeister eindeutig besiegen konnte, nachdem es aus mehreren Spielen gegen sich selbst das Wissen eines Weltmeisters erlangt hat [19].

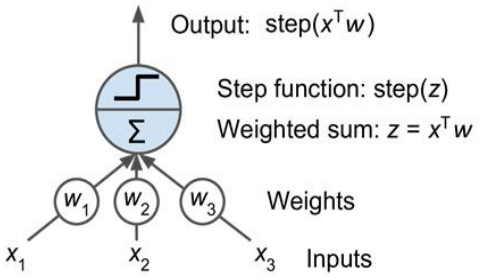

Abb. 1: Ein einzelnes Perzeptron mit einer Stufen-Aktivierungsfunktion. Bearbeitet. [39]

Dabei setzen sich die KNNs aus mehreren Bausteinen wie den Perzeptronen in Abb. 1 zusammen, die mehrere Eingabe Werte in einem Vektor x erhalten und aus diesen mit trainierbaren Gewichten w eine gewichtete Summe $z = x^T w$ bilden. Schließlich sorgt eine Aktivierungsfunktion ϕ als Komponente für die Umformung der gewichteten Summe in ein Ausgabe-Signal $\phi(z)$.

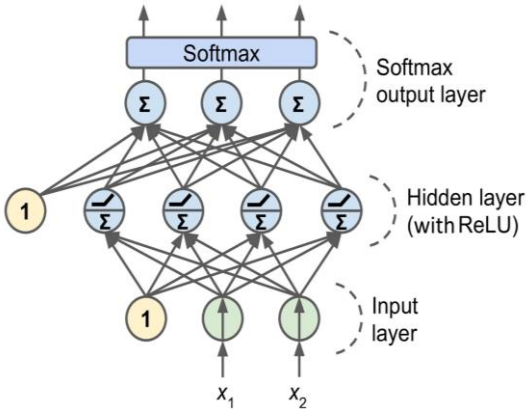

Abb. 2: Ein Multilayer Perceptron mit einer Softmax-Funktion für die Mehrfachklassifikation und zwei Bias-Neuronen. Bearbeitet. [39]

Sobald mehrere dieser Perzeptronen zusammengeschlossen werden, spricht man von einem künstlichen

neuronalen Netzwerk wie in Abb. 2. Jedes der Perzeptronen ist dabei im KNN mit jeder vorherigen und nachfolgenden Schicht verbunden, wobei der gezielte Einsatz von Aktivierungsfunktionen den internen Informationsfluss bestimmt. Je mehr Perzeptronen das KNN besitzt, desto tiefgründiger kann es die Input-Informationen verarbeiten, um einen genauen Output zu generieren. Jedoch benötigen größere neuronale Netzwerke auch mehr Trainingsdaten und Optimierungsdurchläufe im Modell-Training.

Die KNNs bzw. deren Gewichte werden im Modell-Training mit der sogenannten Backpropagation optimiert, die schrittweise den Fehler jeder Schicht zurückverfolgt und dabei zusammen mit dem Gradienten-Abstiegsverfahren die Gewichte optimiert. Hierbei produzieren KI-Modelle mit deren Vorhersagen durchgehend Fehler, die durch eine Fehlerfunktion E wie z. B. den mittleren absoluten Fehler (MAE) oder auch dem mittleren quadratischen Fehler (MSE) beschrieben wird:

$$\mathrm{MAE}(X,\theta) = \frac{1}{N}\sum_{i=1}^{N} |\,\hat{y}_i - y_i|, \qquad \mathrm{MSE}(X,\theta) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Diesen Fehler gilt es für einen Datensatz X zu minimieren, um die Modell-Vorhersagen zu verbessern. Dafür wird nun mittels der Backpropagation die partielle Ableitung der Fehler-Funktion berechnet und die Gradienten multipliziert mit der Lernrate α von den bisherigen Modell-Parametern θ subtrahiert [20]:

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial E(X,\theta_t)}{\partial \theta}$$

Hiermit nähert man sich mit jedem Gradienten-Schritt einem lokalen Minimum der Fehlerfunktion E, womit das Modell durch Optimierung seiner Gewichte lernt, präzise Vorhersagen aus den Eingabe-Werten zu berechnen.

2.1.2 Aktivierungsfunktionen

Wie erwähnt sind die Aktivierungsfunktionen ein zentraler Bestandteil von Perzeptronen, die die nicht-lineare Verarbeitung von Informationen ermöglichen und in neuronalen Netzen den Informationsfluss Steuern. Dabei haben Aktivierungsfunktionen sowohl im Hidden- als auch im Output-Layer verschiedene Aufgaben und müssen sorgfältig ausgewählt werden, da dies Einfluss auf Leistung und Lernfähigkeit des Modells hat. Etwa sollten die Outputs im Output-Layer in einem bestimmten Wertebereich ausgegeben werden: Dies können einzelne Zahlenwerte der ReLU-Funktion oder auch Wahrscheinlichkeitswerte im Intervall $y_i \in [0;1]$ sein, welche die Sigmoid-Funktion $\sigma(z)$ zurückgibt. Eine dritte häufig verwendete Aktivierungsfunktion ist dagegen der Tangens hyperbolicus, dessen Wertebereich auch negative Werte umfasst und der Sigmoid-Funktion im Hidden Layer vorzuziehen ist. Diese drei Aktivierungsfunktionen werden wie folgt definiert [39]:

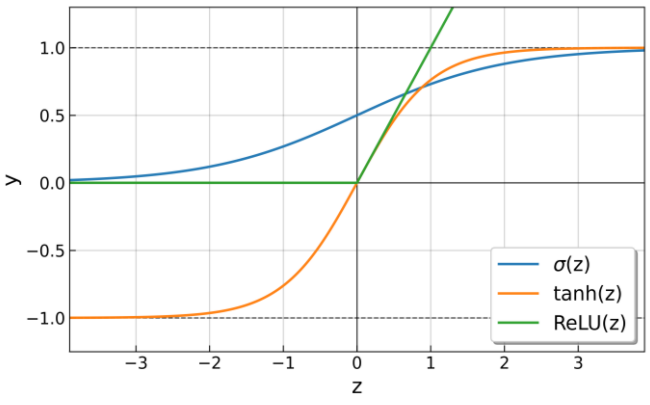

Abb. 3: Die Graphen der drei im typischen Aktivierungsfunktionen des maschinellen Lernens.

$$\sigma(z) = \frac{1}{1+e^{-z}}, \qquad tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}, \qquad ReLU(z) = max(0,z)$$

Dabei ist in der Sprachverarbeitung besonders die Softmax-Funktion als Output-Aktivierungsfunktion von Bedeutung, da alle Mehrfachklassifikationen der Modelle mit dieser Softmax-Funktion vorgenommen werden. Beispielsweise werden, um genau ein Wort aus einem Vokabelsatz von meist über 30.000 Wörtern zu generieren, alle 30.000 Wort-Output-Werte des Modells mittels der Softmax-Funktion normalisiert, sodass jeder Output-Wert des KNNs einer prozentualen Wahrscheinlichkeit entspricht, mit der das Modell besser trainiert werden kann. Diese Softmax-Funktion ist wie folgt definiert [21] und verwendet als Vektor-Funktion alle gegebenen Input-Werte des Vektors z, um eine einzelne Wort-Wahrscheinlichkeit der Wort-Klasse k zu bestimmen [22]:

$$\hat{y}_k = f_k(z) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Die gewichtete Summe z der k-ten Klasse wird hierbei durch die gewichteten Summen aller Klassen geteilt, wodurch die Summe aller Wahrscheinlichkeiten immer genau 1 beträgt.

2.2 Grundlagen der natürlichen Sprachverarbeitung

2.2.1 Allgemeiner Sprachverarbeitungsvorgang

Anhand der Abb. 4 der nächsten Seite soll der gesamte Sprachverarbeitungs- sowie Extraktionsvorgang von OIE-Modellen exemplarisch dargestellt und die einzelnen Teilvorgänge erklärt werden.

Typische Sprachmodelle wie das *Long Short-Term Memory* (LSTM) [23] oder die effektiveren Transformer-Modelle [1] benötigen als Input Zahlenwerte und können den reinen Input-Satz nicht verarbeiten. Hierfür findet eine Vorverarbeitung des Input-Satzes statt, die zuerst ein *Tokenizing* vorsieht, bei dem der Input-Satz in sogenannte Token zerlegt wird, welche z. B. einzelne Buchstaben, Teilsilben oder auch ganze Wörter umfassen. Diese Token-Strings werden schließlich mittels des *Embeddings* in Zahlen umgewandelt, wobei für Transformer-Modelle zwei Arten des Embeddings existieren: das Position-Embedding und das Token-Embedding. Beide werden in der jetzigen Praxis mittels einschichtiger neuronaler Netzwerke berechnet, die Token-Positions-Indizes und Token-Vokabelsatz-Indizes in mehrdimensionale Vektoren mit 512 Elementen umwandeln und den Indizes hiermit einen größeren Informationsgehalt verleihen.

Die Embedding-Parameter werden zusammen mit dem Sprachmodell trainiert und erlernen so die Sprache detailliert in Form von Zahlen-Vektoren darzustellen, womit der Token-Vokabel-Satz aller Tokens einen Vektor-Raum darstellt, in dem Tokens mit ähnlicher Bedeutung Cluster bilden. Ähnliche Cluster liegen näher beieinander und Differenzen zwischen den Bedeutungsvektoren sorgen für eine Vektorverschiebung durch Entfernen einzelner Bedeutungsebenen: Z. B. landet man bei der Bedeutung des Tokens 'König', wenn die Bedeutung des Tokens 'Frau' von dem Bedeutungsvektor 'Königin' subtrahiert wird.

Schließlich ist die Summe aller Embeddings der Input für das Encoder-Decoder-Modell, in welchem der Encoder durch Verarbeitung aller Input-Token-Embeddings eines Satzes dessen Gesamtsatzbedeutung entnimmt und entsprechend der jeweils aller anderen Input-Tokens jeden einzelnen Token seine korrekte Bedeutung im Satz verleiht. Diese Zwischen-Informationen des Encoders werden in Form von einer *Hidden-State-Matrix H* an den Decoder weitergegeben, der diese Informationen zur Output-Bestimmung weiterverarbeitet. Ein letztes einschichtiges KNN mit Softmax-Funktion bestimmt mithilfe des Decoder-Output-Vektors die Output-Wahrscheinlichkeiten aus denen über einen beliebigen Konvertierungsalgorithmus der Output gebildet wird.

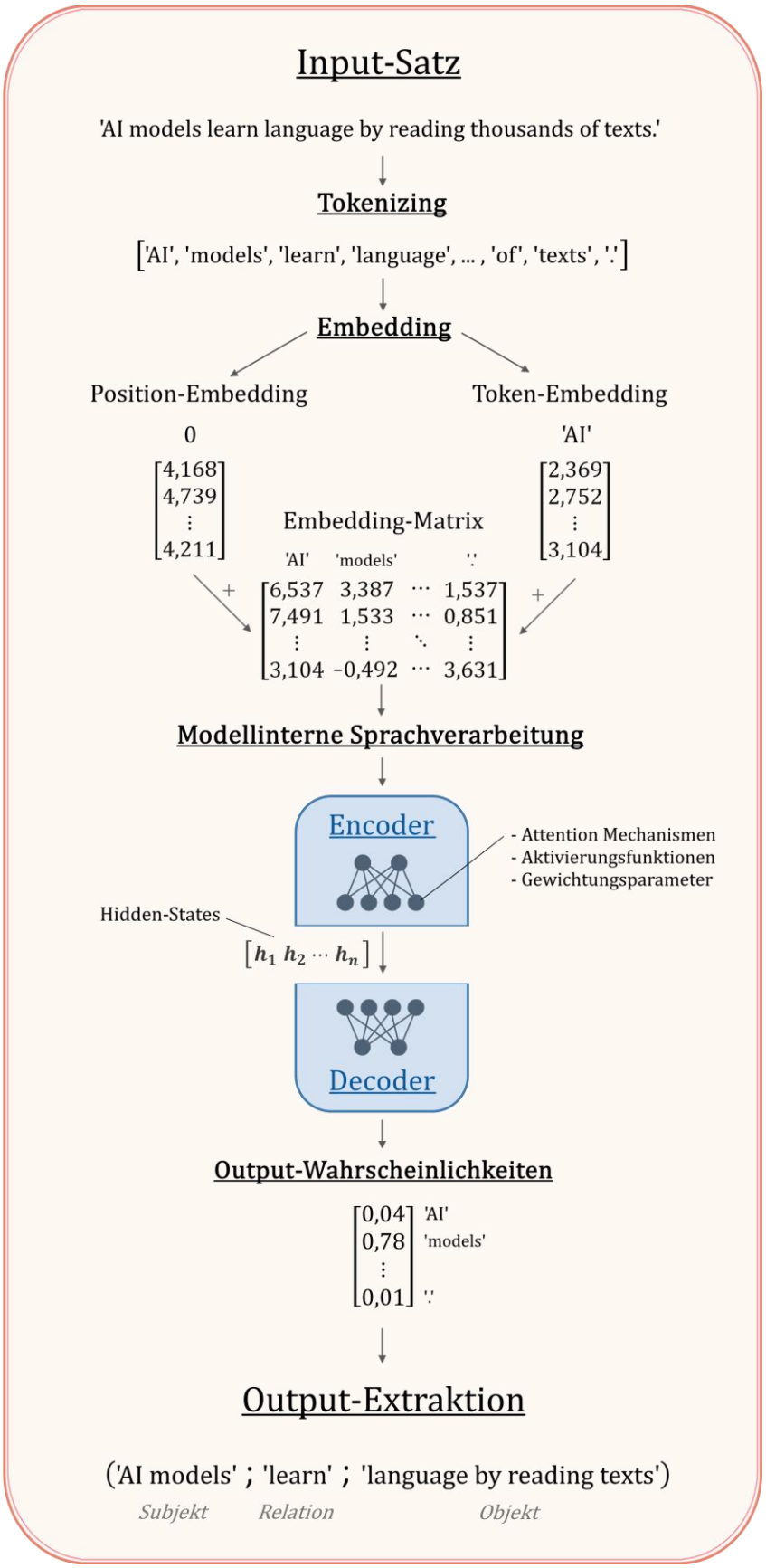

Abb. 4: Der vollständige Sprachverarbeitungsvorgang am Beispiel der Open Information Extraction.

Innerhalb der Open Information Extraction werden mit dem Encoder-Decoder-Modell die Output-Sequenzen im Fall der Sequence Generation autoregressiv Wort-für-Wort generiert oder im Fall des Sequence Labelings in einem Durchgang klassifiziert und zu einer Extraktion der Form (*Subjekt*; *Relation*; Objekt) zusammengesetzt.

2.2.2 Attention-Mechanismen

Innerhalb der modell-internen Sprachverarbeitung gibt es neben der Encoder-Decoder-Struktur ebenfalls noch das wichtige Konzept der Attention-Mechanismen. Diese helfen dem Decoder Informationen aus dem Encoder-Hidden-States zu beziehen und damit auch diese nach mehreren Verarbeitungsschritten nicht langsam zu vergessen.

Allgemein besitzt jede Art der Attention diese grundlegenden Formeln zur Berechnung eines Kontext-Vektors c_j, der die Attention-Informationen enthält [14] [22]:

$$e_{ij} = align(h_i, s_{j-1}),$$
$$b_{ij} = softmax(e_{ij}),$$
$$c_j = \sum_{i=0}^{N} h_i b_{ij}$$

Zunächst bewertet ein Alignment-Modell, wie sehr sich die Informationen des Encoders und Decoders übereinstimmen, wobei der Vergleich mithilfe der Encoder-Hidden-States h_j und der Decoder-Hidden-States s_{j-1} erfolgt [14]. Jeder dieser Hidden-State-Vektoren enthält Informationen für jeweils einen Token. Danach werden alle Scores e_{ij} durch die Softmax-Funktion in eine Wahrscheinlichkeitsverteilung konvertiert, die schließlich die Hidden-States gewichtet.

Es gibt hierbei verschiedene Attention-Mechanismen, die durch unterschiedliche Definitionen des Alignment-Modells gekennzeichnet sind. Eine dieser Varianten ist die Content-Based Attention mit der folgenden bekannten Formel für die Vektor-Ähnlichkeit [22]:

$$e_{ij} = \cos(h_i, s_{j-1}) = \frac{h_i \cdot s_{j-1}}{\|h_i\| \cdot \|s_{j-1}\|}$$

Je kleiner der Winkel zwischen den beiden Hidden-State-Vektoren ist, desto ähnlicher sind diese hierbei zueinander. Die sogenannte Additive-Attention benutzt dagegen zur Entnahme von Encoder-Informationen neuronale Netzwerke mit den trainierbaren Gewichten W_h, W_s und v_a, die noch spezifischere Sprach-Informationen im Gegensatz zur Content-Based Attention entnehmen können [25]:

$$e_{ij} = v_a^T \tanh(W_h h_i + W_s s_{j-1})$$

Neben diesen Arten der Attention-Mechanismen gibt es noch die anschließend mit den Transformer-Modellen [1] beschriebene Multi-Head Attention als Form der Self-Attention, die neben einem Vergleich der Encoder- und Decoder-Hidden-States auch einen Vergleich nur zwischen den Encoder-Hidden-States vornehmen kann, um damit schon während des Encodings mehr Informationen aus den Input-Tokens zu entnehmen.

2.3 Transformer-Modelle

Nachdem die Sprachmodelle mit Encoder-Decoder-Struktur und Attention-Mechanismen für die Anwendung immer weiter vergrößert wurden, stößt man mit den bisherigen rekurrenten neuronalen Netzen mit der Additive-Attention 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, die stattdessen auf der Self-Attention basiert und damit 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.

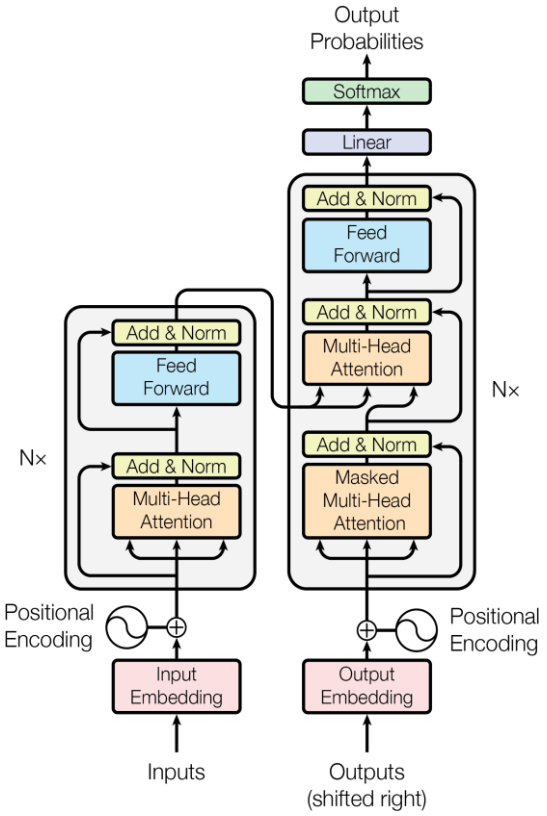

Abb. 5: Die Abbildung der Transformer-Modell Struktur [1].

2.3.1 Grundlegende Funktionsweise

Wie die vorherigen Encoder-Decoder-Modelle verwenden die Transformer sogenannte Encoder und Decoder, um zuerst Informationen aus einer Eingabe-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 bzw. der Wortbestandteile 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. 5 autoregressiv und generiert die Output-Token weiterhin schrittweise basierend auf der letzten Ausgabe eines Tokens.

Zu Beginn 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, das die absolute Position aller Token wie folgt erfasst. Hierfür nutzen die originalen Transformer Sinus- und Kosinus-Funktionen, die jedoch in der Praxis durch neuronale Netze ersetzt werden.

Den Input-Embedding-Vektoren aus Positions- und Token-Embedding wird nun mithilfe eines Multi-Head-Attention-Blocks ein eigener Kontext verliehen, indem jeder Vektor mit dem Ergebnis 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* [20] 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 einen 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.

2.3.2 Scaled Dot-Product Attention

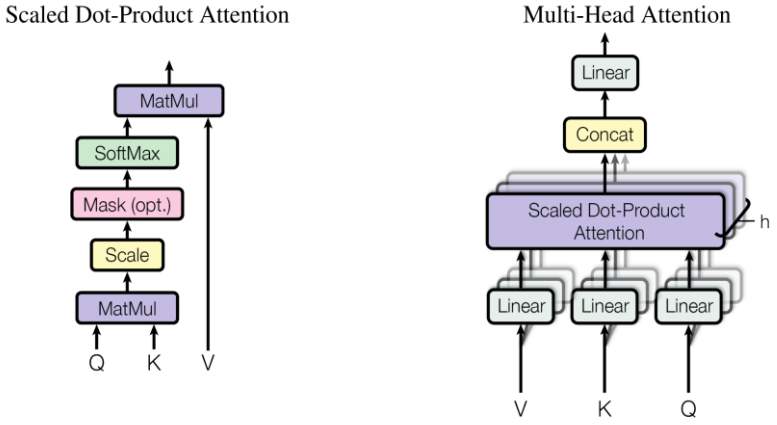

Abb. 6: Die detaillierte Abbildung der im Transformer verwendeten Multi-Head Attention (rechts) und der hierin verwendeten Scaled Dot-Product Attention (links) [1].

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, sondern diese werden mit einem einzigen Skalarprodukt berechnet, was in der Praxis einen hoch optimierten Matrix-Multiplikations-Code zulässt [1]. Entsprechend der Abb. 6, berechnet sich die Scaled Dot-Product Attention wie folgt [1]:

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

Hierbei wird durch den Term $\sqrt{d_k}$ mit $d_k = 512$ 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 wird, 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.

2.3.3 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 $head_{1 \dots h} \in \mathbb{R}^{N \times d_k}$ 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]:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_\text{h}) W^O,$$

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

2.4 BERT-Modelle

Seit dem Jahr 2018 gibt es die für das heutige NLP unerlässlichen BERT-Modelle [2], die als Modell-Struktur lediglich 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 Trainingszielen sowohl ein antrainiertes bidirektionales Satzverständnis als auch ein Gesamtsatzverständnis, wodurch BERT-Modelle erneut neue Bestleistungen bei einer simplen Struktur erzielen.

Das BERT-Modell erlangt dabei sein verbessertes Satzverständnis in einem sogenannten Vortraining, während es hierbei den Trainingszielen 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 zu 10 % doch die korrekten Tokens [2]. Andererseits muss in einer *Next Sentence Prediction* vom Modell in einer binären Klassifikation bestimmt werden, ob ein aus dem Datensatz entnommener Satz auf den gegebenen Input-Satz folgt oder nicht.

2.5 Optimierte BERT-Varianten

2.5.1 RoBERTa

Ein Jahr nach der Veröffentlichung von BERT erschien um 2019 das sogenannte **R**obustly **O**ptimized **BERT A**pproach Modell [3], das einige Makel 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 [3]:

- Das Entfernen der *Next Sentence Prediction* und des NSP-Loss, wobei aber weiterhin nachfolgende Sätze mit als Modell-Input für das Vortraining verwendet werden, sodass das Satzverständnis erhalten bleibt.
- Das *Dynamic Masking*, das anstatt einer statischen Maske nun für jede Input-Sequenz 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 [2].

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 Sequenzen, einer Anpassung der Hyperparameter sowie einer stark gesteigerten Anzahl an Trainingsdurchläufen [3]. BERT war dabei, wie die Autoren des RoBERTa-Modells herausgefunden haben, signifikant untertrainiert [3] und kann noch weitere Informationen der Sprache in seiner Transformer-Struktur erfassen.

2.5.2 ELECTRA

Um 2020 erscheint nun das ELECTRA-Modell [4], das mit einer Generative Adversarial Network (GAN) Struktur [21] 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. 7 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.

2.5.2.1 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. 7 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 [4].

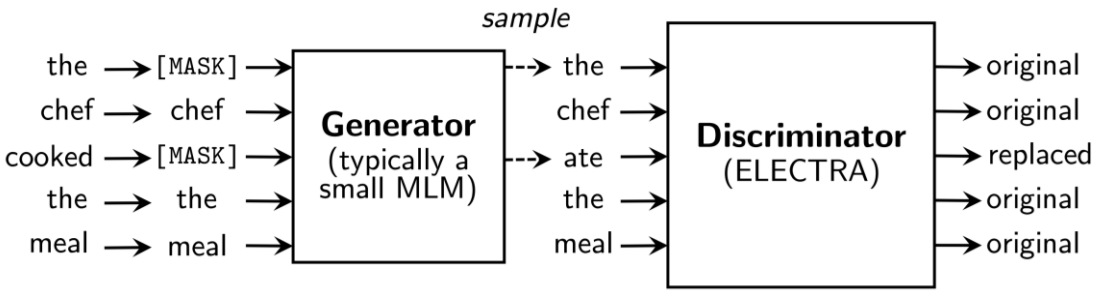

Abb. 7: Die Darstellung der abgewandelten GAN-Architektur mit Transformer-Encodern für jeweils den Generator und den Diskriminator. Die Aufgabe des Generators ist es, die zufällig überdeckten Tokens wieder mit den korrekten Original-Tokens zu ersetzen. Dagegen muss der Diskriminator für jeden der Token erkennen, ob dieser durch den Generator falsch ersetzt wurde oder nicht [4].

In der Praxis erzielen ELECTRA-Modelle, die mit etwa 25 % bis 50 % kleineren Generator-Modellen vortrainiert werden, bessere Leistungen [4], 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 [4], damit beide Modelle den Input gleichermaßen repräsentieren.

2.5.3 DeBERTa

Nur einen Monat später folgte auf das ELECTRA-Modell das DeBERTa-Modell [5], 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 [5]. Um hierbei weiterhin das *Masked Language Modelin*g durchzuführen, erhält dieses Vortraining eine kleine Anpassung mit dem *Enhanced Mask Decoding*.

2.5.3.1 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 [5], [22], [23], 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 [22], [23] 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 [23]:

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

wobei die Matrix A^K für ein berechnetes relatives Positional-Encoding steht. Hierbei fehlt aber eine ebenfalls noch wichtige Positions-zu-Inhalts-Beziehung, die notwendig ist, um die Token-Bedeutung

in Bezug zu dessen Position vollständig darzustellen [5]. Deswegen wird mit dem DeBERTa-Modell versucht, unter einem vektorweisen Berechnungsvorgang diese fehlende *Position-to-Content* Beziehung wie folgt zu ergänzen [5]:

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

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

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

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 Token; 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 $\tilde{A} = A_{c \to c} + A_{c \to p} + A_{p \to c}$ 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 [5].

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 $A_{c \to p}$ werden mit der Index-Funktion δ die 512 relativen Positionen des Bezugstokens zu allen anderen Tokens betrachtet und umgekehrt werden für $A_{p \to c}$ 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 $A_{c \to c}$ keine Matrix-Matrix-Multiplikationen 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 Gewichtmatrizen 15 Millionen Modell-Parameter mehr als das BERT-Modell.

2.5.3.2 Enhanced Mask Decoding

DeBERTa wird immer noch mit dem *Masked Language Modeling* Trainingsziel 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 Decoding*s verbesserte Leistungen [5].

2.5.4 DeBERTaV3

Anknüpfend an das bisherige DeBERTa-Modell [5] verbessert DeBERTaV3 [6] wieder merklich die Modell-Leistung, indem nun das wirksame Vortraining des ELECTRA-Modells [4] und das ebenso effektive Konzept der *Disentangled Attention* vereint werden. Hierbei erhält DeBERTaV3 anstelle des üblichen *Masked Language Modelings* nun das Vortrainingsziel der *Replaced Token Detection*, um die

Stärken beider Modelle zu kombinieren [6]: 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“ [6], weshalb für DeBERTaV3 zusätzlich der neue Mechanismus der *Gradient-Disentangled Embedding Sharing* entwickelt wird, wie in der Abb. 4:

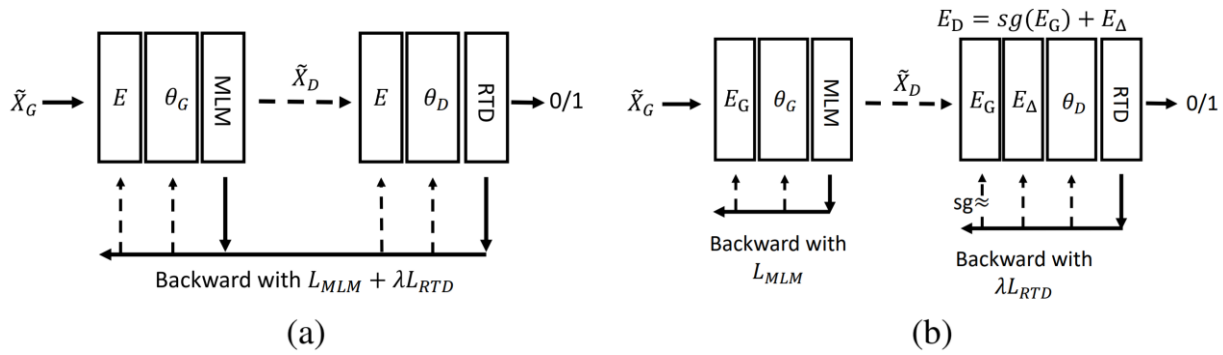

Abb. 8: Die Darstellung des Optimierungsprozesses bei geteilten Token-Embedding-Parametern (a) sowie mit der Methode des *Gradient-Disentangled Embedding Sharings* (b). Hierbei steht sg für den *Stop Gradient*, der die Optimierung von E_G verhindert. Bearbeitet. [6].

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. 8 a alle Berechnungsschritte vom Ende des Diskriminators bis hin zum Token-Embedding des Generators über Gradienten-Produkte zurückverfolgt und mit einem AdamW-Optimizer [24] optimiert.

Dabei aber getrennte Embeddings für Generator und Diskriminator zu nutzen, verringert die Modell-Leistung [4], wogegen jedoch auch das Teilen der Embedding-Gewichte die Effektivität des Vortrainings hemmt und für ein „Tauziehen“ sorgt [6], indem eine Optimierung der Embedder-Parameter in die Richtung zweier auseinanderliegender Optima wie folgt stattfindet [6]:

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

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 [6]:

$$E_\mathrm{D} = sg(E_\mathrm{G}) + E_\Delta$$

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 [25] 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 [25].

2.6 Das DetIE-Modell

Das zurzeit mit leistungsfähigste und zugleich schnelle OIE-Modell DetIE [16] 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 [17] und OpenIE6 [13] vermeidet DetIE ein iteratives sowie autoregressives Extrahieren gänzlich, indem der DetIE-Decoder alle Tokens innerhalb einer Tensorrechnung $N = 20$ potenzielle Extraktionen klassifiziert und aus den klassifizierten Tokens die Extraktionen zusammenbaut.

DetIE besteht aus einem BERT-Encoder, von welchem 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 [26], dessen Input und Output jeweils durch ein neuronales Netzwerk umgeformt wird.

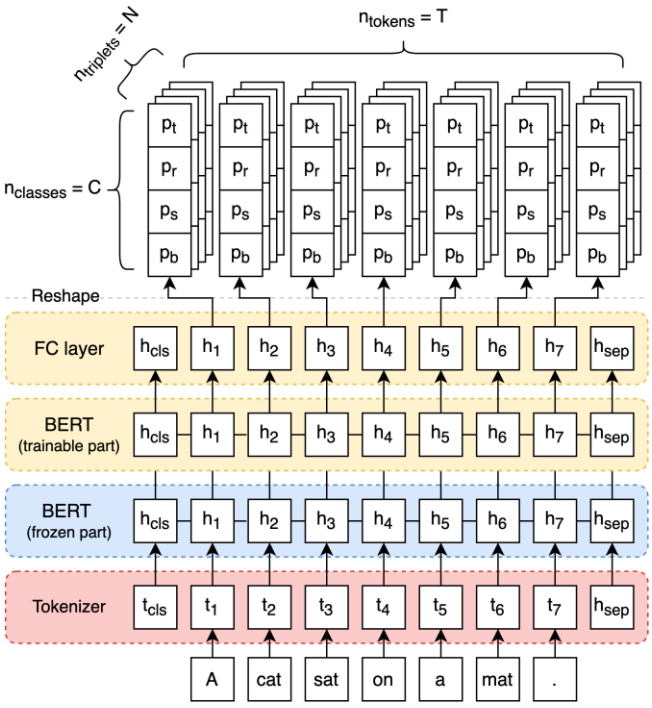

Abb. 9: Die Darstellung der DetIE-Modellstruktur einschließlich des Output-Tensors [16].

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* die vorhergesagte Klasse ausgewält und aus der verbleibenden Matrix $C_{out} \in \mathbb{R}^{T \times N}$ die Extraktionen zusammengebaut: Aus den 20 potenziellen Sequenzen werden alle Sequenzen entfernt, bei denen mindestens eine der Token-Klassen *Subject, Relation* oder *Object* gänzlich fehlt [26], 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 [26], sodass für jede dieser drei Klassen eine Spanne von Start- bis End-Index als Extraktionsbestandteil entnommen wird.

2.6.1 Order-Agnostic Loss

Das Trainieren der Sequence Labeling Modelle erweist sich oftmals als schwierig, da Ähnlichkeitsmetriken zwischen Extraktionen und den gegebenen Lösungsextraktionen nur schwer definierbar sind [16]. Hierfür nutzt DetIE die eingeführte Intersection-over-Union-Metrik, wobei mithilfe der Ungarischen Methode [27], 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 [16]:

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

Hierbei steht p_{tnc} für eine einzelne Vorhersage-Wahrscheinlichkeit der Klasse c für einen Token t einer potenziellen 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 auch gilt, dass $IoU_{nm} = 1$. Wenn schließlich die geeigneten 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 [28] wird hier 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 [29] und basiert auf den Bibliotheken PyTorch, Pytorch-Lightning [32], Hydra [31] sowie der Bibliothek PyTorch-Transformers [32]. 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.

3.1.3 Datensätze und Bewertungsvorgang

Für alle Details zu den verwendeten Datensätzen und der Bewertungsschemas wird auf die ausführliche Beschreibung im DetIE-Forschungsbericht verwiesen [26] 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 [17], LSOIE [33], CaRB [34] sowie OIE2016 [35].

Für das Training werden wie im DetIE-Forschungsbericht jeweils die zwei Trainingsdatensätze LSOIE [33] und der IMoJIE-Trainingsdatensatz [17] verwendet, wobei der LSOIE-Datensatz aus einem annotierten QA-SRL-Datensatz erstellt wurde 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-Geschwindigkeit der OIE2016-Test-Datensatz. Für die Ermittlung der Modell-Leistung erfolgt eine Bewertung nach den insgesamt 4 Bewertungsschemas CaRB [34], CaRB(1-1) [13], OIE2016 [35] sowie WiRe57 [36], die alle nach der Zuordnung von Vorhersage- und Lösungs-Extraktionen die Bewertungsmetriken wie folgt berechnen und sich nur im Zuordnungsschema unterscheiden:

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

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 zurückgibt [26]. 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 durch die gleichartige Transformer-Struktur aller Encoder-Modelle keine Änderung am Modell-Code benötigt wird. Bei der Durchführung werden zudem alle Modelle mit BERT-, RoBERTa- und ELECTRA-Encoder insgesamt fünfmal trainiert und davon die Messungen mit der jeweiligen Höchst-Leistung in die Tabellen übernommen. Die Leistungs-Intervalle der Modelle überschneiden sich nämlich, sodass geringere Leistungen als im DetIE-Forschungsbericht auftreten können. 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**	51,4	28,4
$DetIE_{LSOIE+IGL-CA}$	BERT	36,9	26,9	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**	23,1	63,7	50,0	**31,5**
$DetIE_{LSOIE}$	ELECTRA	35,0	24,6	32,3	20,9	65,7	51,2	29,1
$DetIE_{LSOIE+IGL-CA}$	ELECTRA	37,1	**27,2**	34,7	**23,6**	**65,9**	**53,5**	31,1

Tab. 3: Die Auflistung aller Messergebnisse bei Training mit dem LSOIE-Datensatz [33] und den gleichen in Tab. 2 genannten Bedingungen.

Modell	Encoder	CaRB-Bewertungsschemas							Geschiwind.
		CaRB		CaRB(1-1)		OIE16-C		WiRe57-C	
		F1	AUC	F1	AUC	F1	AUC	F1	(Sätze je Sek.)
IMoJIE*	BERT	53,5	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	29,8	67,4	54,2	37,7	203,2
$DetIE_{IMoJIE}$	RoBERTa	52,6	37,2	41,0	24,8	56,9	39,5	37,1	522,1
$DetIE_{IMoJIE+IGL-CA}$	RoBERTa	48,1	35,9	43,8	**30,0**	**67,7**	54,0	39,0	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 [16] übernommen.

Die IGL-CA-Komponente [13] 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 Modelle werden erst nach dem Training mit einer IGL-CA-Komponente kombiniert und benötigen kein erneutes Training.

Dabei liegt der Grund für die höhere Geschwindigkeit der Modelle mit IGL-CA-Komponente 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-parallelisierten Prozesse stark verringert wird.

3.3 Auswertung der Ergebnisse

3.3.1 Auswertung der Experimente mit RoBERTa- und ELECTRA-Encodern

Aus den Tabellen kann entnommen werden, wie die Modell-Leistungen sich mit der Nutzung von RoBERTa- und ELECTRA-Encodern für das DetIE-Modell bei nahezu gleicher Extraktionsgeschwindigkeit verbessern oder für bestimmte Modelle gleich bleiben. Im Unterschied zu Tab. 3 erzielen alle DetIE-Modelle in Tab. 2 ohne IGL-CA-Komponente stetig von BERT zu RoBERTa zu ELECTRA eindeutige Leistungsverbesserungen, wogegen Modelle mit IGL-CA-Komponente nur geringfügige Verbesserungen erzielen. In Tab. 3 bei Training mit dem LSOIE-Datensatz ist dies umgekehrt: Die Modelle mit RoBERTa- und ELECTRA-Encoder mit IGL-CA-Komponente verbessern eindeutig die Leistungen, während bei Modellen ohne IGL-CA-Komponente keine Verbesserungen zu beobachten sind.

Dieser Unterschied wird mit der unterschiedlichen Trainingsdatensatzgröße der beiden Trainingsdatensätze begründet, die den Encodern bei großer Trainingsdatenmenge erlauben, Verbesserungen für komplexe Input-Sätze zu erzielen und bei geringeren Mengen an Trainingsdaten nur erlauben, Verbesserungen für simple Input-Sätze mit der IGL-CA-Komponente zu erzielen. Zudem werden bei genügend Trainingsdaten mit dem IMoJIE-Datensatz bei zusätzlicher Vereinfachung der Sätze durch die IGL-CA-Komponente keine Verbesserungen erzielt, da sich sowohl BERT- als auch ELECTRA-Encoder an die Extraktionsaufgabe anpassen können und kein komplexeres Sprachwissen für die aufgetrennten Sätze notwendig ist.

Aus diesen Beobachtungen werden zwei allgemeine Verbesserungskriterien abgeleitet, die erfüllt werden müssen, um Verbesserungen mit einem Encoder-Austausch zu erzielen: Es müssen genügend Trainingsdaten vorhanden sein, um die Encoder möglichst vollständig an die Extraktionsaufgabe anzupassen und die Extraktionsaufgabe muss sprachlich komplex genug sein, damit ein verbessertes Sprachwissen zu verbesserten Leistungen führt.

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 relevant, wenn man bedenkt, dass keine Verminderung der Extraktionsgeschwindigkeit vorliegt und dieser Encoder-Austausch simpel für jedes andere OIE-Modell vorgenommen werden kann. Dabei werden keine oder nur minimale Verbesserungen gegenüber dem selbsttrainierten DetIE-Modell beobachtet.

3.3.2 Auswertung der Experimente mit DeBERTa- und DeBERTaV3-Encodern

Die Modell-Leistungen für die Modelle mit DeBERTa- und DeBERTaV3-Encoder fallen sehr gering aus, da diese Modelle kaum Extraktionen generieren. Bei genauerer Untersuchung wurde zuerst der Code der Experiment-Pipeline auf Fehler untersucht und keine Fehler in der Vorverarbeitung der Input-Sätze oder in der nachgängigen Zusammensetzung der Extraktionen gefunden. Stattdessen wird herausgefunden, dass, obwohl der Input für beide Encoder korrekt ist, mit den Output-Wahrscheinlicheiten für die DetIE-Klassen *Background, Subject, Relation* und *Object* fast ausschließlich nur die Klasse *Background* klassifizieren, sodass diese Extraktionen gefiltert und damit nicht generiert werden. Die Ursache liegt folglich in der encoder-internen Sprachverarbeitung.

Deswegen wird mithilfe der Theorie nach möglichen Fehlern gesucht und zwei mögliche Ursachen gefunden: Die DeBERTa-Encoder besitzen für die trainierten und nicht-trainierten Encoder-Schichten dasselbe relative Position-Embedding *P*, obwohl wie in Abb. 10 beide Encoder-Teile für die Verarbeitung unterschiedlicher Wissensbestandteile verantwortlich sind. Ebenfalls besitzen die DeBERTa-Encoder keine Art von absoluten Position-Embeddings, die für die Extraktion von Tupeln sowie der Grid-Labeling-Aufgabe von DetIE essentiell sind, um die Tokens innerhalb einer Extraktion einzuordnen.

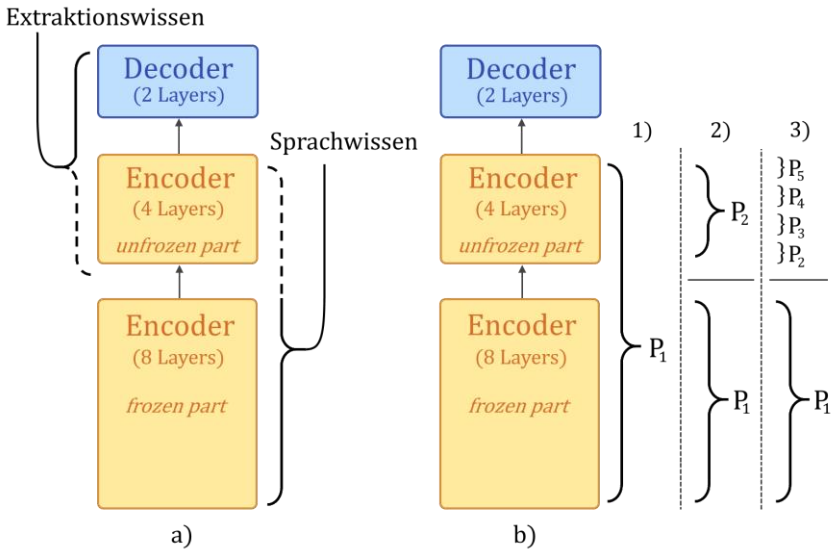

Abb. 10: Die Darstellung der Modell-Wissensbestandteile (a) und die Visualisierung der Experiment-Konfigurationen mit getrennt optimierten relativen Position-Embedding für die weiteren Experimente mit den DeBERTa-Encodern (b).

Somit werden die Experimente mit den DeBERTa-Encodern wiederholt, um eine korrekte Implementation der DeBERTa-Encoder zu testen. Hierfür werden die zwei neuen Konfigurationen der Abb. 10 b mit getrennt-optimierten relative Position-Embeddings angewendet. Dennoch verbleibt nach der Behebung immer noch das erwähnte Problem, sodass unter Ausschluss anderer Einflussfaktoren das fehlende absolute Position-Embedding der DeBERTa-Encoder als endgültige Ursache für die geringen Leistungen der DeBERTa-Encoder angesehen wird.

Die absoluten Positionsinformationen den DeBERTa-Encodern nachträglich als Input mit zu übergeben führt dabei zu keinen verbesserten Leistungen, da die vortrainierten DeBERTa-Encoder strikt auf Informationen des relativen Position-Embeddings angewiesen sind und durch weitere Informationen die Informationsverarbeitung im Encoder gestört wird. Auch der Versuch, die absoluten Positionsinformationen der Tokens dem Decoder zu übergeben, führt zu keiner signifikanten Änderung der Leistung, da Token-Bedeutungen erst tiefgründig zusammen mit den Token-Embeddings verarbeitet werden müssen, um benötigte Beziehungen von Inhalt und Position einzufangen.

Schließlich können die DeBERTa-Encoder nicht erneut vortrainiert werden, womit diese für die Open Information Extraction als nicht anwendbar eingestuft werden.

4. Fazit

Insgesamt werden mit den Experimenten dieser Forschungsarbeit Leistungsverbesserungen von durchschnittlich bis zu 1,1 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 von Relevanz sind. Damit wurde in dieser Forschungsarbeit eine wichtige, sowohl praktische als auch theoretische Grundlage für die zukünftige Forschung erarbeitet, mit der signifikante Verbesserungen möglich sind.

Zudem werden wichtige Verbesserungskriterien ausfindig gemacht, die die Höhe der Verbesserungen bestimmen: Die Sprachaufgabe benötigt genügend Komplexität und Trainingsdaten in Relation zu dieser Komplexität, damit Verbesserungen erzielbar sind. Zusätzlich wurde beobachtet, dass unbedingt Informationen des absoluten Position-Embeddings für Aufgaben der OIE benötigt werden und damit die Encoder DeEBRTa und DeBERTaV3 nicht in diesem Bereich anwendbar sind. Somit wird außerdem aufgestellte Hypothese, dass alle der vier versuchten Encoder zu Leistungsverbesserungen führt, auf die Encoder RoBERTa und ELECTRA bei den benannten Verbesserungskriterien eingeschränkt.

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] J. Devlin, M.-W. Chang, K. Lee und K. Toutanova, „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding“, arXiv:1810.04805v2, 2019.

[3] 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.

[4] 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.

[5] P. He, X. Liu, J. Gao und W. Chen, „DeBERTa: Decoding-enhanced BERT with Disentangled Attention“, arXiv:2006.03654v6, 2021.

[6] P. He, J. Gao und W. Chen, „DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing“, arXiv:2111.09543, 2021.

[7] A. Wang, Y. Pruksachatukun, 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.

[8] P. He, X. Liu, J. Gao und W. Chen, „Microsoft DeBERTa surpasses human performance on the SuperGLUE benchmark“, Microsoft, 6 Januar 2021. https://www.microsoft.com/en-us/research/blog/microsoft-deberta-surpasses-human-performance-on-the-superglue-benchmark/. [11. Dezember 2021].

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] L. Cui, F. Wei und M. Zhou, „Neural Open Information Extraction“, arXiv:1805.04270v1, 2018.

[15] D.-T. Vo und E. Bagheri, „Open Information Extraction“, arXiv:1607.02784v1, 2016.

[16] 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.

[17] K. Kolluru, S. Aggarwal, V. Rathore, Mausam und S. Chakrabarti, „IMoJIE: Iterative Memory-Based Joint Open Information Extraction“, arXiv:2005.08178v1, 2020.

[18] J. Gu, Z. Lu, H. Li und V. O. Li, „Incorporating Copying Mechanism in Sequence-to-Sequence Learning“, arXiv:1603.06393v3, 2016.

[19] J. Steinwendner und R. Schwaiger, Neuronale Netze Programmieren mit Python, 2. Auflage Hrsg., Bonn: Rheinwerk Verlag, 2020.

[20] J. B. Ba, R. J. Kiros und G. E. Hinton, „Layer Normalization“, arXiv:1607.06450v1, 2016.

[21] 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.

[22] 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.

[23] P. Shaw, J. Uszkoreit und A. Vaswani, „Self-Attention with Relative Position Representations“, arXiv:1803.02155v2, 2018.

[24] I. Loshchilov und F. Hutter, „Fixing Weight Decay Regularization in Adam“, or arXiv:1711.05101v3, 2019.

[25] P. He, X. Liu, J. Gao und W. Chen, „DeBERTa: Decoding-enhanced BERT with Disentangled Attention“, https://github.com/microsoft/DeBERTa. [06.12.2022].

[26] 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“, https://github.com/sberbank-ai/DetIE. [20.11.2022].

[27] H. W. Kuhn, „The Hungarian method for the assignment problem“, Naval research logistics quarterly, 1955.

[28] M. Prikryl, „WinSCP“, https://winscp.net/eng/download.php. [13.12.2021].

[29] K. Kolluru, „IMOJIE: Iterative Memory-Based Joint Open Information Extraction“, https://github.com/dair-iitd/imojie/tree/master/imojie/models. [12.12.2021].

[30] Falcon, W., & The PyTorch Lightning team, „PyTorch Lightning“.

[31] K. Nagrecha und A. Kumar, „Hydra: A System for Large Multi-Model Deep Learning“, arXiv:2110.08633, 2022.

[32] 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.

[33] J. Solawetz und S. Larson, „LSOIE: A Large-Scale Dataset for Supervised Open Information Extraction“, arXiv:2101.11177, 2021.

[34] S. Bhardwaj, S. Aggarwal und M. , „CaRB: A Crowdsourced Benchmark for Open IE“, 2019. https://aclanthology.org/D19-1651.pdf. [13. Dezember 2021].

[35] G. Stanovsky und I. Dagan, „Creating a Large Benchmark for Open Information Extraction“, 2016.

[36] W. Léchelle, F. Gotti und P. Langlais, „WiRe57 : A Fine-Grained Benchmark for Open Information Extraction“, arXiv:1809.08962, 2019.

[37] 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.

[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. Géron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, 2. Auflage der Frühveröffentlichung 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. 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“, 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. 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“, 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. Merrienboer, 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“, 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.

Abbildungsverzeichnis

[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.

[4] 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.

[5] P. He, X. Liu, J. Gao und W. Chen, „DeBERTa: Decoding-enhanced BERT with Disentangled Attention“, arXiv:2006.03654v6, 2021.

[6] P. He, J. Gao und W. Chen, „DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing“, arXiv:2111.09543, 2021.

[16] 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.

[39] A. Géron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, 2. Auflage der Frühveröffentlichung Hrsg., Sebastopol: O'Reilly, 2019.

Tabellenverzeichnis

[16] 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.

Abkürzungsverzeichnis

Abkürzung	Bedeutung
AUC	Area Under the Curve
BERT	Bidirectional Encoder Representations from Transformers
CaRB	A Crowdsourced Benchmark
DeBERTa	Decoding-Enhanced BERT approach
DetIE	Information Extraction inspired by Object Detection
ELECTRA	Efficiently Learning an Encoder that Classifies Token Replacements Accurately
ERNIE	Enhanced Representation through Knowledge Integration
F1-Score	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
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	Neuronales Netzwerk
OIE	Open Information Extraction
QA-SRL	Question Answer Semantic Role Labeling
RoBERTa	Robustly Optimized BERT Approach
WiRe57	57 annotierte Sätze aus Wikipedia-Artikeln und Newswire-Artikeln von Reuters

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 unterstütze. Ebenfalls bedanke ich mich sehr über die viele Unterstützung meiner schulischen Betreuerin Romy Schachoff, die mich während der Besonderen Lernleistung 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, 23.02.2023 (Ort, Datum) Henning Beyer (Unterschrift)