Arwed-Rossbach-Schule
Berufliches Gymnasium

Belegarbeit

Im Fachgebiet Informatik und Mathematik

Die Verbesserung der Modellgeschwindigkeit von IMoJIE mit einem GRU-Decoder

Verfasser: Henning Beyer
Klasse: A20/2
Schuljahr: 2021/22
Schulischer Betreuer: Romy Schachoff
Außerschulischer Betreuer: Prof. Andreas Maletti
Datum: Leipzig, 23.01.2022

Inhaltsverzeichnis

1. Kurzfassung

Das IMoJIE-Modell ist ein aktuelles Modell im Bereich der Open Information Extraction, welches strukturierte Fakten aus Texten extrahiert und das Copy-Attention-Modell mit einem iterativen Mechanismus erweitert. Inspiriert von der Verbesserung, die durch den Austausch des LSTM-Encoders mit einem BERT-Encoder für das Copy-Attention-Modell erzielt werden konnte, ersetze ich im IMoJIE-Modell den LSTM-Decoder mit einem GRU-Decoder, um die Anwendungsgeschwindigkeit von IMoJIE zu verbessern. Ich konnte die GRU als Decoder erfolgreich für IMoJIE implementieren und die Trainingsgeschwindigkeit um durchschnittlich 49 Minuten verbessern. Entgegen meiner Hypothese und der erwiesenen Geschwindigkeitsverbesserung der GRU gegenüber dem LSTM, konnte ich die Testgeschwindigkeit nicht verbessern, da die Berechnungszeit für einen Decoder-Schritt unerwartet zu durchschnittlich 97,63 % von der Berechnung der Copy- und generativen Wahrscheinlichkeiten abhängt. Dennoch ist genau diese Beobachtung sehr relevant, um eine zukünftige Geschwindigkeitsverbesserung für IMoJIE vorzunehmen.

2. Einleitung

Das IMoJIE-Modell als mein Hauptthema dieser Belegarbeit ist ein komplexes Modell und aktuell eines der leistungsfähigsten KI-Modelle im Bereich der Open Information Extraction (OIE), die sich mit dem strukturierten Extrahieren von Fakten befasst. Dabei bedient sich das IMoJIE-Modell auch an einigen sehr aktuellen Methoden des Natural Language Processing (NLP), welches sich mit der algorithmischen Verarbeitung der Sprache befasst und das übergeordnete Gebiet der OIE ist. Bevor ich das Thema und meine Ziele der Belegarbeit genauer vorstelle, beginne ich erst mit diesen beiden Bereichen als Hinführung in die Belegarbeit und erläutere mit deren aktuellen Forschungsstand.

2.1 Einführung in den Stand des Natural Language Processing

Der Bereich des NLPs hat im letzten Jahrzehnt einige Fortschritte gemacht und ist weiterhin ein sehr gefragtes Themengebiet der Informatik. Dieser Bereich verspricht nämlich allgemeine Aufgaben, wie die Analyse von Textdaten oder die Sprachgenerierung zukünftig zu automatisieren. Schon heute wird das NLP erfolgreich in Schreibprogrammen oder Suchmaschinen für z. B. die Übersetzung, die Textzusammenfassung oder dem Question-Answering eingesetzt. Um hierbei die komplexe Sprache in allen Ebenen zu verstehen, verwendet man spezielle Modelle wie die Recurrent Neural Networks oder Transformer-Modelle wie BERT. Vor wenigen Jahren orientierten sich die Forschungen im NLP an den Transformer-Modellen, die um 2017 in den sehr bekannten Forschungsbericht „Attention Is All You Need“ [1] vorgestellt wurden. Auf der Grundlage dieser Modelle präsentierte Google später das BERT-Modell [2], welches sich mit überragender Leistung auf viele NLP-Aufgaben flexibel anwenden lässt. Seitdem entstehen weiterhin neue Modelle nach dem Vorbild von BERT wie ALBERT [3], RoBERTa [4] oder DeBERTa [5], die mithilfe neuer Mechanismen und Methoden die Transformer-Struktur immer weiter ausreizen. So werden Transformer-Modelle zunehmend größer und benötigen für die Anwendung viele Ressourcen. Beispielsweise besteht das DeBERTa-Modell aus 1,5 Milliarden trainierbaren Parametern und musste für 30 Tage auf 256 GPUs trainiert werden [5].

Mittlerweile sind solche großen Modelle dafür aber in der Lage, das menschliche Sprachverständnis in einigen Aufgaben zu übertreffen: Im Januar 2021 übertraf DeBERTa die menschliche Leistung in

der SuperGLUE-Benchmark [6] als erstes Modell überhaupt [5], [7]. Jedoch sind solche Modelle noch nicht in der Lage eigenständig komplexere Texte zu generieren, da es ihnen an eigenem Allgemeinwissen fehlt. Vielmehr orientieren sich die Modelle dazu an anderen Texten und Wissensquellen. Deswegen wird im NLP auch danach gestrebt, die Modell-Leistung mithilfe von Allgemeinwissen meist in Form von Knowledge Graphs zu verbessern, da damit auch das Satzverständnis weiter verbessert wird. Einen sehr aktuellen Erfolg erzielt hierbei das wissensgestützte Modell ERNIE 3.0, welches neue Bestleistungen in 54 verschiedenen Benchmarks erzielt [8] und ebenfalls das DeBERTa-Modell in der SuperGLUE-Benchmark übertrifft [8], [9].

2.2 Einführung in die Open Information Extraction

Die OIE dagegen beschäftigt sich als Teilbereich des NLP mit der Extraktion von strukturierten Informationen aus Sätzen, meist in der Form von Relations-Tupel [10]. Hierbei spielt die OIE eine wichtige Rolle im Sprachverständnis und unterstützt im NLP viele untergeordnete Aufgaben, wie das Question Answering, die Textzusammenfassung oder die Erstellung von Wissensdatenbanken [10]. Ebenso wird die OIE extensiv für Suchanfragen und dem automatischen Auslesen von Finanzunterlagen angewendet. Seit erstmaliger Vorstellung des Bereiches mit dem Modell TEXTRUNNER um 2007 [10], [11], machte die OIE mit der Zeit einige Fortschritte. Erste Modelle der OIE waren entweder statistische oder regelbasierte Modelle mit Parsern, die aber später durch neuronale Modelle ersetzt wurden [10], [12]. Dabei basieren sehr aktuelle neuronale Modelle wie IMoJIE [12], OpenIE-6 [13] oder Multi^{2}OIE [14] auf einer Seq2seq-Struktur mit meist BERT-Encoder sowie LSTM-Decoder. Weiterhin besteht in der OIE das Problem des Mangels an großen annotierten Datensätzen, damit größere und leistungsfähigere Modelle trainiert werden können [15]. Deshalb besteht auch in der OIE das Ziel, neue große Datensätze zu entwickeln und ebenso Modelle zu trainieren, die mit wenigen Daten auskommen. In Zukunft könnte die OIE auch wissensbasierte Modelle im NLP maßgeblich unterstützen, indem große Wissensdatensätze für die wissensgestützten Modelle des NLP erstellt werden.

2.3 Ziele der Belegarbeit

Zu meinem Belegarbeitsthema wurde das IMoJIE-Modell besonders wegen der guten Möglichkeit aktuelle und zugleich relevante Versuche in der OIE und im NLP durchzuführen. Der öffentliche Code in GitHub [16] sowie die Seq2seq-Struktur von IMoJIE ermöglichen es mir nämlich sehr gut, einen Austausch der Teilmodelle vorzunehmen und an die aktuelle Forschung anzuknüpfen. Zudem hat mich besonders die große Verbesserung durch den Austausch des LSTM-Encoders mit einem BERT-Encoder am Copy-Attention-Modell [12] dazu inspiriert, ähnliche Versuche mit anderen Teilmodellen durchzuführen. So kann im IMoJIE-Modell auf einer Seite der BERT-Encoder und auf der anderen Seite der LSTM-Decoder ausgetauscht werden, um die Leistungsfähigkeit und Geschwindigkeit von IMoJIE zu erhöhen. Ich entscheide mich hierbei, den LSTM-Decoder in dieser Belegarbeit mit einem GRU-Decoder auszutauschen, da diese Idee für mich vor allem gut umsetzbar ist. Ursprünglich plante ich zusätzlich einen Austausch mit einer der aktuelleren BERT-Varianten oder mit einem bidirektionalen LSTM, was beides jedoch zu anspruchsvoll und zeitaufwendig wäre. Bei den BERT-Encodern bedarf es nämlich an viel zusätzlicher Theorie, wie auch bei der Implementation eines BiLSTM-Decoders, für den ich zusätzlich einen bidirektionalen Beam-Search [17] aufgrund der Copy-Attention implementieren müsste. Somit ersetzte ich in dieser Belegarbeit den LSTM-Decoder von IMoJIE mit einem GRU-Decoder und erhoffe mir hiermit eine schnellere Anwendungsgeschwindigkeit zu erzielen, da IMoJIE sehr viel langsamer im Vergleich zu anderen OIE-Modellen ist, wie im Forschungsbericht des Open-IE6-Modells bemängelt wurde [13]. Dabei ist meine Hypothese, dass ein GRU-Decoder

die Anwendungsgeschwindigkeit von IMoJIE leicht verbessert, da die GRU nach der Theorie weniger Berechnungsschritte wie das LSTM benötigt und dies auch Experimente in anderen Forschungsberichten belegen.

3. Theorie

Das IMoJIE-Modell nutzt einige Teilmodelle und Mechanismen wie das LSTM-Modell oder die Copy-Attention, die ich in der Theorie beschreiben werde. Ich beginne hierbei mit neuronalen Netzen als Einstieg in das maschinelle Lernen und gehe danach detaillierter auf die verschiedenen RNN-Modelle ein. Die Seq2seq-Modelle und die Attention-Mechanismen sind hierbei wichtige Grundlagen für die die Modelle CopyNet [18] sowie IMoJIE. Im Praxis-Abschnitt hilft ein grobes Grundverständnis der Theorie beim Verstehen der Experimente und Auswertungen.

3.1 Die Grundlagen des maschinellen Lernens

3.1.1 Künstliche neuronale Netze

Von den klassischen künstlichen neuronalen Netzen (KNN) hat man schon viel gehört oder gelesen: Die KNNs haben die Fähigkeit, in kurzer Zeit große Mengen an Daten zu erlernen, womit diese schnell die menschliche Leistung in simplen Aufgaben übertreffen. Einer der wohl bekanntesten Erfolge der KNNs ist hierbei das Modell AlphaGo, welches nach wiederholten Trainingsspielen gegen sich selbst den Go-Weltmeister um 2016 in mehreren Spielen Go eindeutig besiegte [19]. Dabei können neuere Modell-Varianten dieses Niveau eines Weltmeisters schon in nur 24 Stunden erreichen [20].

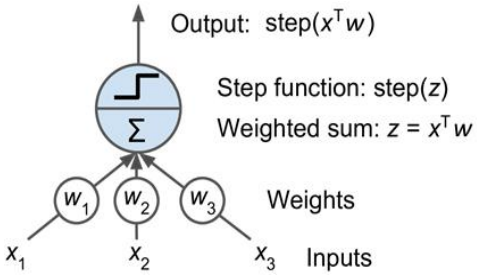

Abb. 1: Ein einzelnes Perzeptron mit einer Stufen-Aktivierungsfunktion. Bearbeitet. [21]

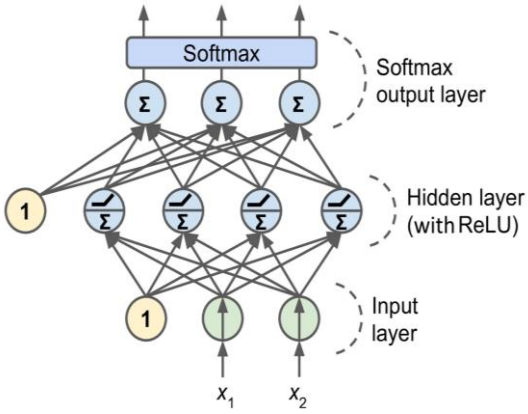

Abb. 2: Ein Multilayer Perceptron mit einer Softmax-Funktion für die Mehrfachklassifikation und zwei Bias-Neuronen. Bearbeitet. [21]

Die KNNs bestehen hierbei aus mehreren Bausteinen wie den Rosenblatt Perzeptronen, die in einer Netzstruktur angeordnet werden und durch die Optimierung ihrer einzelnen Gewichte lernen können. Einzelne Perzeptronen bilden aus einen Input- und einen Gewichtsvektor eine gewichtete Summe $z = x^T w$, woraufhin diese anschließend das Signal $\phi(z)$ ihrer Aktivierungsfunktion ϕ weiterleiten. Bei Aneinanderfügen mehrerer dieser Perzeptronen, erhält man ein Multilayer Perceptron bzw. schon ein KNN, wobei diese eine Grundstruktur mit Input-Layer, Hidden Layer und Output-Layer haben. Hierbei ist nur die Anzahl der Perzeptronen im Input- und Output-Layer von der Aufgabe abhängig; andere Hyperparameter wie die Anzahl der Perzeptronen und die Art der Aktivierungsfunktion sind im Hidden Layer beliebig veränderbar. Bei vielen eingebauten Perzeptronen wird z. B. die Leistungsfähigkeit auf Kosten der Geschwindigkeit erhöht. Bei solchen veränderbaren Parametern spricht man von Hyperparametern.

3.1.2 Aktivierungsfunktionen

Die Aktivierungsfunktionen sind nun essenzieller Bestandteil der einzelnen Neuronen und ermöglichen als nicht lineare Funktion die vielseitige Verarbeitung der Eingabewerte. Aktivierungsfunktionen haben dabei sowohl im Hidden- als auch im Output-Layer verschiedene Aufgaben und müssen sorgfältig ausgewählt werden, da dies großen Einfluss auf Leistung und Lernfähigkeit 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 $\sigma(x)$ zurückgibt. Eine dritte häufig verwendete Aktivierungsfunktion ist 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 [21]:

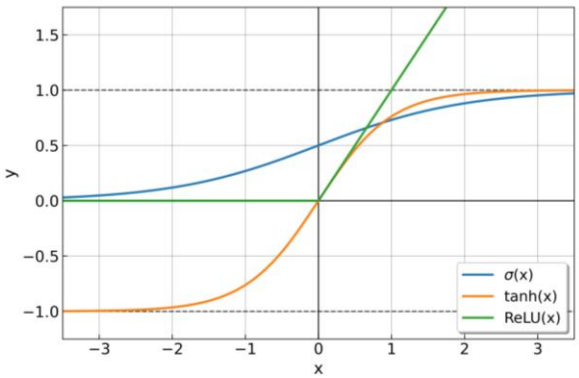

Abb. 3: Die Graphen der drei im typischen Aktivierungs-funktionen des maschinellen Lernens.

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

Dagegen wird für die Mehrfachklassifikation die Softmax-Funktion im Output-Layer eingesetzt, da diese für jede Klasse eine eigene Wahrscheinlichkeit zurückgibt. Die Softmax-Funktion wird hier wie folgt definiert [22]:

$$\widehat{y_k} = P(y = k|x) = \frac{\exp({w_k}^T x)}{\sum_{j=1}^{K} \exp({w_j}^T x)}$$

Wobei in dieser Formel die gewichtete Summe z der k-ten Klasse durch die alle anderen gewichteten Summen geteilt wird, um für die Klasse y die Vorhersage-Wahrscheinlichkeit $\widehat{y_k}$ zu bestimmen. K steht für die Anzahl der Klassen; w_k sind die Gewichte der k-ten Klasse und $\exp(x)$ ist hier eine übersichtlichere Schreibweise für e^x. Die Summe aller Wahrscheinlichkeiten beträgt 1 wegen der Teilung durch den Normalisierungsterm $\sum_{j=1}^{K} \exp({w_j}^T x)$. Insgesamt wandelt die Softmax-Funktion die Bewertung jeder Klasse in eine Wahrscheinlichkeitsverteilung um, in der die Klasse mit der höchsten Wahrscheinlichkeit als Output ausgegeben wird.

3.2 Rekurrente neuronale Netzwerke

Im Gegensatz zu den KNNs werden die rekurrenten neuronalen Netzwerke (RNNs) für sequenzielle Daten, wie z. B. für Sätze eingesetzt, bei denen Abhängigkeiten zwischen den einzelnen zeitlichen Input-Bestandteilen bestehen. Anders als bei den KNNs, bei welchen diese wichtigen Abhängigkeiten nicht berücksichtigt werden, erlaubt nämlich ein rekurrenter Mechanismus der RNNs, die Informationen vorheriger RNN-Zellen mit in die aktuelle Berechnung miteinzubeziehen. So hängt die Berechnung für eine RNN-Zelle von allen vorherigen Hidden States ab. Dadurch, dass in die Berechnung der vorherige Hidden State h_{t-1} eingeht, kann ein RNN den aktuellen Sequenz-Input x_t besser einordnen und somit auch besser eine zusammenhängende Sequenz oder auch einen einzelnen Output-Wert generieren. Ein Hidden State h_t für eine RNN-Zelle berechnet sich dabei wie folgt [23]:

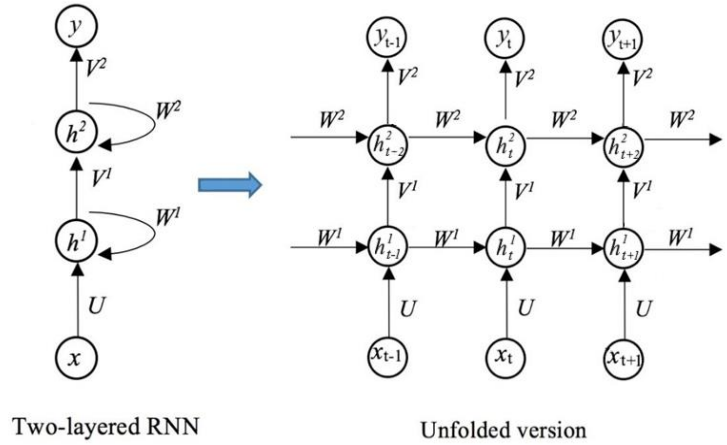

Abb. 4: Der Bau eines zweischichtigen RNNs: links zusammengefasst und rechts ausgebreitet. Hierbei sind U, V und W Gewichtsmatrizen, die mehrere Gewichtsvektoren enthalten. x_t, h_t und y_t stellen jeweils die Input-, Hidden-State- und Output-Werte dar. Bearbeitet. [22]

$$h_t = \phi(Wh_{t-1} + Ux_t)$$

Der aktuelle Hidden State h_t bezieht in dieser Formel die Informationen des vorherigen Hidden States h_{t-1}, wobei bei dieser rekurrenten Berechnung h_{t-1} sowie x_t durch die im Layer geteilten Gewichtsmatrizen W und U gewichtet werden. Schließlich kann für jede RNN-Zelle abhängig von der RNN-Struktur ein Output y_t berechnet werden [23]:

$$y_t = \phi(Vh_t)$$

Heutzutage werden klassische RNNs selten verwendet, da neuere Varianten, wie das LSTM und die GRU die Schwachpunkte des RNNs minimieren und sich einzelne Informationen der Sequenzteile länger und genauer merken können. Dabei ist die geringe Merkfähigkeit der RNNs ein Problem, bei der die Information eines Hidden States nur über wenige Schritte erhalten bleibt. Aber vor allem bei langen Sequenzen ist das RNN äußerst anfällig für das Problem der schwindenden und explodierenden Gradienten, die durch wiederholte Gradienten-Produkte während der Optimierung mit der Backpropagation-through-Time entstehen [23].

3.2.1 Long Short-Term Memory

Das um 1997 von Hochreiter und Schmidhuber entwickelte LSTM löst einige für das RNN unlösbare Aufgaben [24] und minimiert das Problem der schwindenden und explodierenden Gradienten [23]. Seit seiner Veröffentlichung wurde es weiter angepasst und es entstanden auf Grundlage dieses LSTMs neue Modelle wie das Peephole-LSTM [25] oder die GRU [26]. Das LSTM verwendet denselben rekurrenten Mechanismus der RNNs aber verarbeitet die Eingabewerte getrennt von den Hidden States. Hierbei ist das Ziel der LSTM-Zellen, das Zell-Gedächtnis gegen sowohl irrelevante und

schwankende Inputs als auch gegen schwankende Hidden States zu schützen [24], die das Zell-Gedächtnis maßgeblich stören.

Wie in Abb. 4 erhalten die LSTMs als Input zusätzlich einen Cell-State c_{t-1} von den vorherigen Zellen, wobei der Cell-State einzelne Informationen getrennt von dem vorherigen Hidden State h_{t-1} und Input x_t mitführt. Der Hidden State h_{t-1} dagegen, verändert den Cell-State c_{t-1} und reguliert gleichzeitig, welche Informationen des geänderten Cell-States c_t als neuer Hidden State h_t ausgegeben werden. Dieser ausgegebene Hidden State nimmt dann in der nächsten Zelle Einfluss auf den nächsten Cell-State, indem gezielt Informationen erst vergessen und danach überschrieben werden, ohne dass der Hidden State h_t nur durch einer Kombination mit dem Input x_t und dem vorherigen Hidden State h_{t-1} gebildet wird. Somit erhält das LSTM insgesamt ein „langes Kurzzeitgedächtnis“ und eine deutlich größere Merkfähigkeit als die RNNs.

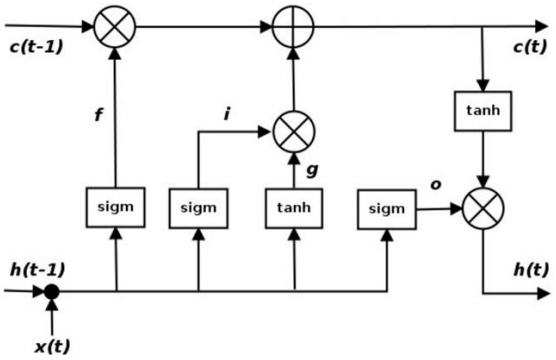

Abb. 5: Der Aufbau einer LSTM-Zelle mit den Gates f, i, g und o. Die Symbole $\oplus$ und $\otimes$ stellen hier jeweils die Matrizenaddition und das elementweise Produkt $\circ$ dar [23].

Der Informationsfluss im LSTM erfolgt mithilfe des Input-Gates i, des Forget-Gates f, des Output-Gates o sowie des Gates g mit folgenden Formeln [23]:

$$
\begin{aligned}
i &= \sigma(W_i h_{t-1} + U_i x_t) \\
f &= \sigma\big(W_f h_{t-1} + U_f x_t\big) \\
o &= \sigma(W_o h_{t-1} + U_o x_t) \\
g &= tanh\big(W_g h_{t-1} + U_g x_t\big) \\
c_t &= (f \circ c_{t-1}) + (g \circ i) \\
h_t &= tanh(c_t) \circ o
\end{aligned}
$$

Hierbei lässt sich die Funktionsweise der LSTM-Zelle und deren Gates leichter nachvollziehen. Diese haben hier folgende Funktionen nach den Formeln:

- Das Forget-Gate f steuert, wie viele der gemerkten Informationen im Cell-Stat c_{t-1} vergessen werden sollen. Hierzu wird der Cell-State c_{t-1} mit den Werten der σ-Funktion multipliziert, welche zwischen 0 und 1 liegen.
- Das Gate g entnimmt mithilfe der tanh-Funktion Informationen aus h_{t-1} und x_t
- Das Input-Gate i bestimmt anhand der Hidden States h_{t-1} und Inputs x_t, wie viele der entnommenen Informationen aus g zum Cell-State c_{t-1} addiert werden.
- Schließlich reguliert das Output-Gate o, welche Informationen des Cell-States c_t als neuer Hidden State h_t ausgegeben werden.

3.2.2 Gated Recurrent Units

Die GRU [26] ist eine alternative LSTM-Variante, die ähnliche Leistungen wie das LSTM erzielt, jedoch durch weniger Berechnungsschritte schneller zu trainieren ist [23]. Anders als das LSTM, besitzt die GRU nur zwei Gates und benötigt als Input keinen Cell-State. Ebenfalls hat die GRU zwei Gewichtsmatrizen weniger zu optimieren und umfasst hiermit diese vier Formeln zur Berechnung [23]:

$$z = \sigma(W_z h_{t-1} + U_z x_t)$$
$$r = \sigma(W_r h_{t-1} + U_r x_t)$$
$$c = tanh(W_c(h_{t-1} \circ r) + U_c x_t)$$
$$h_t = (z \circ c) + ((1 - z) \circ h_{t-1})$$

Hierbei bestimmt das Reset-Gate r, wie viele Informationen des Hidden States h_{t-1} in den Cell-State c übernommen werden. Danach bestimmt das Update-Gate z, in welchem Verhältnis der Hidden State h_{t-1} vom Cell State c überschrieben wird. Dabei löscht das Reset-Gate r im Voraus Teile der Informationen des Hidden States h_{t-1}, damit dieser Hidden State mit ähnlichem Verhältnis vom Input x_t in der Gleichung für h_t überschrieben werden kann. Wieder gehen kaum Informationen verloren, da gezielt entschieden wird, sowohl Informationen aus dem vorherigen Hidden State h_{t-1} als auch aus der Kombination des Inputs x_t mit dem Hidden State h_{t-1} zu übernehmen.

Bisherige Versuche mit der GRU und dem LSTM legen nahe, dass die GRU bei schnellerer Berechenbarkeit gleichwertige Leistungen wie das LSTM erzielt: Bei der Emotionsbestimmung aus verrauschten Audio-Sequenzen konnte eine GRU die Laufzeit um 18,16 % verkürzen und dabei meist ähnliche Leistungen, wie das LSTM erzielen [27]. Hierbei kann die GRU auch die Leistung des LSTM übertreffen [28], [29], [30] oder dieser unterliegen, was dabei aber von der Aufgabe abhängig ist.

3.3 Seq2seq-Modelle

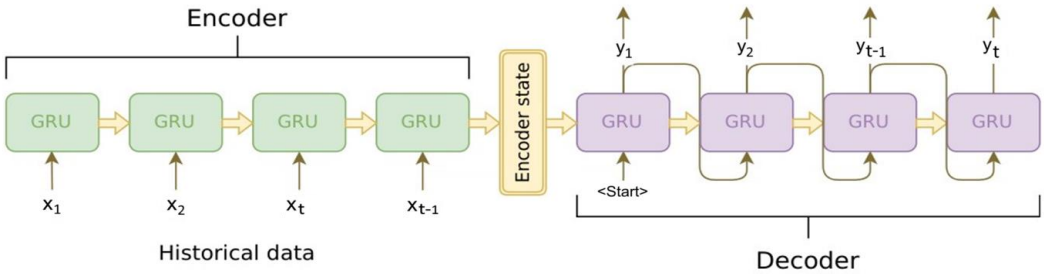

Abb. 6: Ein Seq2seq-Modell mit der GRU als Encoder- und Decoder-Modell. Der Encoder erhält eine Input-Sequenz, auf dessen Grundlage der Decoder eine Output-Sequenz generiert. Bearbeitet. [23].

Für Aufgaben im NLP, bei denen eine neue Sequenz generiert werden soll, erkennen gewöhnliche RNN-Modelle die Satzbedeutung erst schrittweise mit jedem neuen Sequenzinput, müssen aber trotzdem eine neue Sequenz generieren, bevor der Satzinhalt überhaupt ganz verstanden wurde. Die Lösung dagegen ist die Verwendung der Seq2seq-Struktur in Abb. 6. Diese Struktur löst nämlich dieses Problem, indem der Encoder erst alle Informationen des Satzes erfasst und dem Decoder liefert, bevor sich dieser Decoder voll auf die Generierung der Text-Sequenz fokussieren kann. Im Seq2seq-Modell werden als Encoder und Decoder meist RNN-Modelle wie LSTMs oder GRUs verwendet, wobei häufig auch bidirektionale RNN-Modelle sowie Transformer-Modelle mit größerer Leistung verwendet werden.

Dadurch, dass sich der Decoder nur am letzten Encoder-Hidden-State s_t, sowie seinen eigenen generierten Outputs y_{t-1} orientiert, geht die Erinnerung an den Input-Satz zunehmend mit Länge der Sequenz verloren. Hieraus ergibt sich ein Schwachpunkt der Seq2seq-Modelle. Ebenfalls besteht für jeden Decoder-Berechnungsschritt eine Abhängigkeit von bestimmten Textstellen, die ein einzelner Vektor nicht übermitteln kann [23]. Somit verwendet man in Seq2seq-Modellen die Attention-Mechanismen, die beide dieser Probleme beheben.

3.3.1 Attention Mechanismen in Seq2seq-Modellen

Attention-Mechanismen im Allgemeinen helfen die Aufmerksamkeit des Modells auf bestimmte Zell-Zustände zu richten, um wichtige Informationen von diesen zu erhalten. Hierbei wird in Seq2seq-Modellen während des Decodings die Aufmerksamkeit auf alle Encoder-Hidden-States gerichtet, um Informationen des Input-Satzes in Form eines Kontext-Vektors c_j zu erhalten. Der Attention-Mechanismus behebt dabei das Problem, den gesamten Satzkontext in einen einzelnen Vektor zusammenfassen zu müssen [18] und ermöglicht Informationen des Encoders auch über längere Sätze zu merken. Jeder Decoder-Hidden-State s_t berechnet sich jetzt mit einen neuen Kontext-Vektor c_j:

$$s_t = \phi(Ws_{t-1} + Uy_{j-1} + W_c c_j)$$

Wobei sich dieser Kontext-Vektor schrittweise mit diesen Formeln berechnet [10], [23]:

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

Hierbei bewertet zuerst ein Alignment-Modell, wie sehr sich die Inputs des Encoders und Decoder übereinstimmen, wobei der Vergleich mithilfe der Encoder-Hidden-States h_i und der Decoder-Hidden-States s_{j-1} erfolgt [10]. Danach werden alle Scores e_{ij} durch die Softmax-Funktion in eine Wahrscheinlichkeitsverteilung konvertiert und schließlich mit einer gewichteten Summe der Kontext-Vektor c_j berechnet [23]. Es gibt hierbei verschiedene Attention-Mechanismen, die durch unterschiedliche Definitionen des Alignment-Modells gekennzeichnet sind. Eine dieser Varianten ist die Content-Based Attention mit folgender Formel [23]:

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

Dieses Alignment-Modell berechnet den Kosinus vom Winkel zwischen den beiden Vektoren h_i sowie s_{j-1} und gibt dabei Werte von 1 bei großer Vektor-Ähnlichkeit bis -1 bei entgegengesetzten Vektoren zurück. Ein weiterer bekannter Attention-Mechanismus, wie die Additive Attention in Abb. 7a, nutzt ein neuronales Netzwerk mit den trainierbaren Gewichten W_h, W_s und einen trainierbaren Alignment-Vektor v_a, für die folgende Berechnung [23]:

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

Insgesamt können die Attention-Mechanismen in drei Kategorien eingeteilt werden [23]: die Self-Attention, die Soft Attention und die Hard Attention. Die Self-Attention ermittelt dabei für jedes Input-Wort die eigene Bedeutung im Satz und wird vor allem in den Transformer-Modellen wie BERT

eingesetzt, da die Self-Attention eine geringe Berechnungskomplexität besitzt und Berechnungen parallelisiert durchgeführt werden können [1]. Die häufig verwendete Soft Attention wie z. B. die Additive Attention, berechnet dagegen die Attention mit der gesamten Input-Sequenz, während die Hard Attention nur einen Teil der Input-Sequenz miteinbezieht [23]. Die Merkmale der Soft Attention sind hierbei die Differenzierbarkeit, aber ein großer Rechenaufwand, wogegen die Hard Attention schneller zu berechnen, aber nicht differenzierbar ist [23]. Zudem sind die Varianten der Hard Attention schwerer zu implementieren und benötigen für den Einsatz z. T. Reinforcement-Learning-Methoden.

3.4 Das Copy-Attention-Modell

Die Copy-Attention-Modelle sind Seq2seq-Modelle, die mit einem Copy-Mechanismus einzelne Wörter aus dem Input-Satz kopieren können, was besonders wichtig für Aufgaben in der OIE ist, da man in dieser viele Begriffe aus dem Input-Satz übernimmt. Ebenfalls lösen die Copy-Attention-Modelle das out-of-Vocabulary-Problem [18] und ermöglichen somit auch anders als die vorherigen und ähnlichen Pointer Networks [31] Sätze mit unbekannten Vokabeln zu generieren [18]. Ursprünglich wurde der Copy-Mechanismus um 2016 mit dem Modell CopyNet [18] eingeführt und erstmals um 2018 in der OIE verwendet [10], wobei dieser Copy-Mechanismus große Ähnlichkeiten mit dem vorherigen Pointer Mechanismus der Pointer Networks hat [18]. Ab 2020 nutzte auch das IMoJIE-Modell das Copy-Attention-Modell als sein Grundmodell, wobei der LSTM-Encoder des CopyNet-Modells mit einem BERT-Encoder ersetzt wurde, was zu einer großen Leistungsverbesserung führte [12].

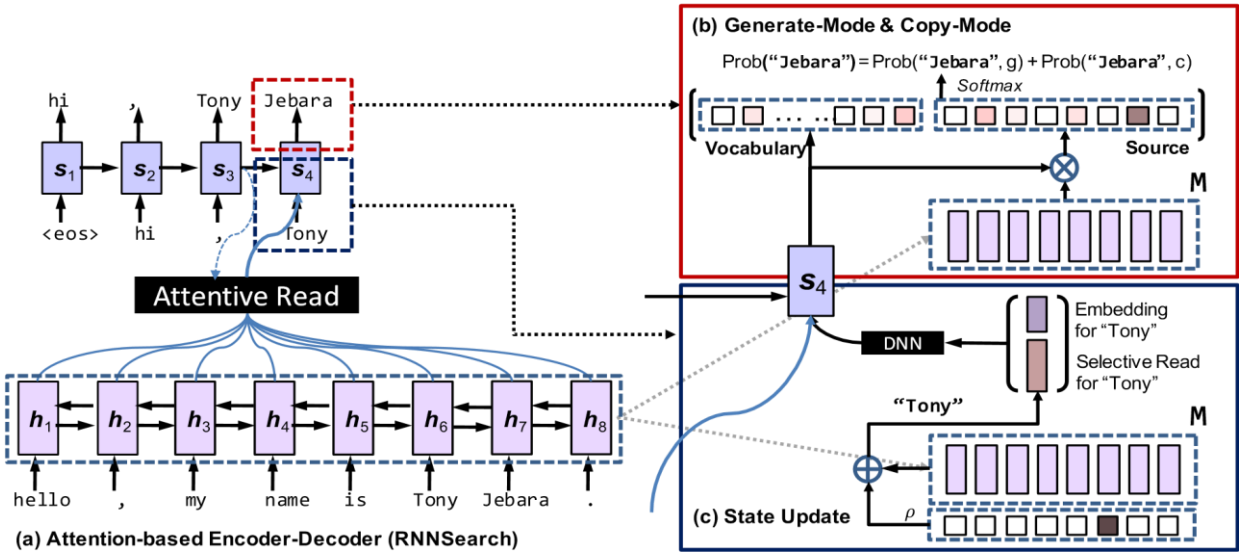

Abb. 7: Die Darstellung des CopyNet-Modells mit bidirektionalen RNN-Encoder und RNN-Decoder [18].
(a): Stellt die gewöhnliche Encoder-Decoder-Struktur mit der Additive Attention dar.
(c): Stellt den Input der RNN-Zelle und die Berechnung des Selective Reads dar. M bezeichnet die Encoder-Hidden-States und ρ die Selective Weigths. Ein Deep Neural Network (DNN) projiziert das Embedding und den Selective Read in die passende Inputgröße.
(b): Zeigt die Berechnungsschritte für die einzelnen Wortwahrscheinlichkeiten. Der Decoder-Hidden-State s_4 gibt einen Vektor zurück, aus dem mit der Softmax-Funktion die Wahrscheinlichkeiten für jedes im Vokabel- und Inputsatz vorhandene Wort ermittelt wird.

Wie in der detaillierten Abbildung 7 besteht ein Copy-Attention-Modell aus einem Seq2seq-Modell mit der Additive Attention und jetzt auch einer zusätzlichen Copy-Attention, mit welcher der Selective Read ζ berechnet wird. Ebenfalls wird die Wahrscheinlichkeit für jedes Wort mit zwei Modi bestimmt: einen Generative Mode und einen Copy-Mode, wobei der Generative Mode für jedes Wort y_t im Vokabelsatz V die Wortwahrscheinlichkeiten wie ein RNN berechnet. Dagegen berechnet der

Copy-Mode speziell für die Wörter der Input-Sequenz eine eigene Copy-Wahrscheinlichkeit. Für jedes Wort ergibt sich schließlich die Gesamtwahrscheinlichkeit aus den Wahrscheinlichkeiten der beiden Modi. Diese Formeln hierfür lauten nun wie folgt [18]:

$$p(y_t, \mathrm{g}|\cdot) = \begin{cases} \frac{1}{Z} e^{\psi_g(y_t)}, & y_t \in V \\ 0, & y_t \in \chi \cap \bar{V} \\ \frac{1}{Z} e^{\psi_g(UNK)} & y_t \notin V \cup \chi \end{cases}, \qquad p(y_t, \mathrm{c}|\cdot) = \begin{cases} \frac{1}{Z} \sum_{j:x_j = y_t} e^{\psi_c(x_j)}, & y_t \in \chi \\ 0 & sonst \end{cases}$$

Hierbei steht das Symbol $\cdot$ für alle involvierten Variablen s_t, y_{t-1}, c_t und M, welche Einfluss auf die Score-Funktionen $\psi_g(\cdot)$ und $\psi_c(\cdot)$ nehmen. χ steht hierbei für die Menge aus den Wörtern des Input-Satzes und Z steht hierbei für einen Normalisierungsterm, der die Summe aller Scores beider Modi umfasst: $Z = \sum_{v \in V \cup \{UNK\}} e^{\psi_g(v)} + \sum_{x \in X} e^{\psi_c(x)}$ [18]. Somit berechnen sich die beiden Wahrscheinlichkeiten der Modi mit einer geteilten Softmax-Funktion, was zu einer flexibleren Kombination beider Modi, als bei den vorherigen Pointer Networks führt [18]. Die Score-Funktion ψ_g des Generative Modes berechnet sich dabei, wie bei einem gewöhnlichen RNN und gibt einen Vektor mit den Scores für jede Vokabel v_i zurück [18]:

$$\psi_g(y_t = v_i) = V s_t, \qquad v_i \in V \cup UNK$$

Dagegen wird die Copy-Score-Funktion ψ_c mithilfe eines eigenen neuronalen Netzwerks berechnet, das die Encoder-Hidden-States h_j dabei in die passende Größe zur Multiplikation mit dem Decoder-Hidden-State s_t umformt. Die Bewertung findet somit nach der folgenden Formel statt [18]:

$$\psi_c(y_t = x_j) = \sigma(h_j^T W_c)\, s_t, \qquad x_j \in \chi$$

Für jeden Decoder-Hidden-State s_t wird hiermit ein Vergleich mit allen Encoder-Hidden-States h_j gemacht. Dabei ist der Score je größer, desto ähnlicher sich die Hidden States h_j und s_t sind, da sich bei der Multiplikation gleiche Werte vergrößern bzw. groß bleiben und schließlich einen höheren Score ergeben. D. h. man ermittelt mit der größten Ähnlichkeit zwischen beiden Hidden States das zu kopierende Wort und somit führt Encoder-Hidden-State bei viel Ähnlichkeit auf sein Input-Wort zurück, welches kopiert werden soll. Mit den Copy-Scores wird anschließend der Selective Read ζ für den nächsten Decoder-Schritt berechnet, mit denen die Informationen vorheriger Kopiervorgänge Einfluss auf die nächsten Copy- und Generative Scores nehmen.

3.5 Das IMoJIE-Modell

Das IMoJIE-Model – kurz für **I**terative **M**em**o**ry-Based **J**oint Open **I**nformation **E**xtraction – ist ein aktuelles Modell im Bereich der Open Information Extraction, das strukturierte Fakten in Form von Tripeln extrahiert. Dabei besteht das IMoJIE-Modell aus einem Copy-Attention-Modell mit BERT-Encoder und LSTM-Decoder, wobei IMoJIE diese Modell-Struktur mit einem neuen iterativen Mechanismus erweitert, mit welchem jede neue generierte Extraktion an alle bisherigen Extraktionen einschließlich den Input-Satz angefügt wird wie in Abb. 8. Wenn IMoJIE so mehrere Fakten extrahiert, kann es sich hiermit an seinen letzten Extraktionen orientieren, womit es ein iteratives Gedächtnis (Iterative Memory) erhält, Redundanzen vermeidet und eine variable Anzahl an Extraktionen generieren kann [12].

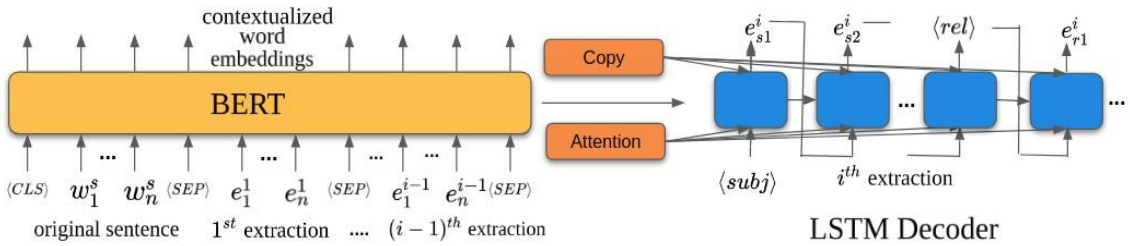

Abb. 8: Die Darstellung des IMoJIE-Modells mit BERT-Encoder und LSTM-Decoder. Der Input besteht aus dem originalen Input-Satz sowie allen bisherigen Extraktionen [12].

Jede Extraktion wird während des iterativen Mechanismus wie in Abb. 8 nacheinander an den Input-Satz angefügt, wobei der gesamte Input für jede der maximal 10 Iterationen durch den BERT-Encoder erneut encodiert wird. Nach dem Encoding generiert der LSTM-Decoder mithilfe des Beam-Search-Algorithmus den Output-Satz, wobei der Beam-Search 5 separate Sequenzen mit dem Decoder-Modell generieren lässt, von denen die Beste für den Output ausgewählt wird. Dabei kann das IMoJIE-Modell in der aktuellen Extraktion ein *EndofExtractions*-Tag generieren und somit entscheiden diesen iterativen Mechanismus abzubrechen [12]. Zudem wird das IMoJIE-Modell mit einem eigenen aggregierten Datensatz trainiert, der mit den Extraktionen aus Wikipedia-Artikeln mithilfe mehrerer OIE-Modelle erstellt wurde; daher der Begriff „Joint" im IMoJIE-Modellname. Hierfür wurde im Forschungsbericht ein neuer Score-and-Filter-Mechanismus eingeführt, welcher die optimalste Kombination der Modell-Extraktionen für den Datensatz herausfiltert [12].

4. Praxis

In der Praxis versuche ich die Trainings- sowie Extraktionsgeschwindigkeit des IMoJIE-Modells zu verbessern und meine Hypothese nachzuweisen, indem ich den LSTM-Decoder mit einem GRU-Decoder ersetze. Hierbei untersuche ich sowohl die Leistung als auch die Testgeschwindigkeit und nehme hierzu einige veröffentlichte[1] Python-Skripts und IPYNB-Dateien zur Analyse.

4.1 Experimentelles Setup

4.1.1 Server und Hardware

Das Trainieren und Verwenden von großen Modellen wie IMoJIE erfordert eine große Rechenkapazität, sowie eine Parallelisierung der Berechnungen mithilfe von GPUs. Hierfür stellt mir die Universität Leipzig einen Server mit der CUDA-Version 11.4 und den zwei folgenden Grafikkarten zur Verfügung: Eine NVIDIA TITAN RTX mit 24 GB RAM und eine NVIDIA TITAN V mit 12 GB RAM. Mit diesen Grafikkarten werden große Matrizen- und Tensor-Berechnungen effizient und parallelisiert durchgeführt, sodass sich die Trainingszeit des IMoJIE-Modells von einigen Tagen auf ein paar Stunden reduziert. Damit während der langen Berechnungszeit keine dauerhafte Verbindung zwischen dem Heimcomputer und dem Server bestehen muss, verwende ich einen GNU Screen, wobei ich ebenfalls für die Dateiarbeit im Serververzeichnis das Tool WinSCP [32] verwende. Im Ganzen habe ich hiermit alles, was ich für die Experimente benötige und kann im Wesentlichen keine weiteren Änderungen zur Vermeidung von Abweichungen vornehmen, da auch CUDA-Version und die GPUs auf dem Server festgelegt sind.

[1] https://github.com/HenningBeyer/IMoJIE_with_GRU_decoder

4.1.2 Code des IMoJIE-Modells

Der gesamte benutzte Code für IMoJIE ist in GitHub öffentlich verfügbar [16] und in Python geschrieben. Wichtige Bibliotheken sind hierbei PyTorch und AllenNLP [33]. Dabei ist PyTorch eine grundlegende Bibliothek für das maschinelle Lernen und ermöglicht die Anwendung einer GPU, wogegen AllenNLP als neuere NLP-Bibliothek das Experimentieren mit NLP-Modellen vereinfacht, indem diese hierzu mehrere Klassen zur Verfügung stellt. Das Training kann z.B. mithilfe von Konfigurationsdateien vorgenommen werden, wobei aus den angegebenen Parametern das Gesamtmodell instanziiert wird. Auch kann die Testung mit dem CaRB-Datensatz [34] über eine Reihe von Bash-Befehlen ausgeführt werden. Hierbei ist das IMoJIE-Modell in dem Skript *copy_seq2seq_bahdanu.py* implementiert, dessen Code etwa 1200 Zeilen umfasst. Natürlich implementiere ich dieses Modell aufgrund des Umfangs und der Komplexität nicht erneut und führe lediglich Analysen der Extraktionen und der Screen-Logs, nach dem Trainieren der Modelle durch. Hierbei starte ich das Modell-Training über das Skript *allennlp_script.py*, dass sowohl den Trainings-, als auch den Testablauf initialisiert.

4.2 Tests mit LSTM-Decodern

Zuerst trainiere und teste ich das originale IMoJIE-Modell bei den ursprünglichen Einstellungen und ermittle, ob ich mit der gegebenen Hardware ein leistungsfähiges IMoJIE-Modell trainieren kann. Für das Modell-Training verändere ich dabei die Trainings-Batch-Size, die im Zusammenwirken mit der Hardware zu unterschiedlichen Modell-Leistungen führt.

LSTM-Decoder			
Batch-Size	**F1-Opt**	**AUC**	**CUDA-Device**
128	51,20	31,10	TITAN RTX, Cuda=11.4
64	58,50	31,10	TITAN RTX
32*	**53,50**	**33,30**	TeslaV100, Cuda=10.0
32	46,20	26,90	TITAN RTX
16	51,90	33,10	TITAN RTX
8	50,70	31,60	TITAN RTX
8	**53,10**	**33,70**	TITAN V, Cuda=11.4

Tab. 1: Die Modell-Leistung bei unterschiedlicher Trainings-Batch-Size und Hardware. *Aus IMoJIE-Forschungsbericht [12].

Die Messergebnisse in Tab. 1 zeigen die Abweichungen der Leistungen nach dem Trainieren mit verschiedenen Batch-Sizes und GPUs. Da ich durchgängig hohe Modell-Leistungen und ähnliche Messergebnisse wie im Forschungsbericht erhalte, schließe daraus, dass ich weitere Experimente mit verlässlichen Messwerten durchführen kann. Weitere Tests mit einem zweischichtigen LSTM als Decoder wie in Abb. 4, lieferten dagegen niedrige Modell-Leistungen. Diese Beobachtung bedeutet, dass nicht einfach ein größerer Decoder mit mehr Schichten verwendet zur Steigerung der Modell-Leistung verwendet werden kann.

4.3 Tests mit GRU-Decodern

Die GRUs erzielen ähnliche Leistungen wie LSTM-Modelle, sind aber nach der Theorie schneller in der Anwendung. Weil IMoJIE mit seinem iterativen Mechanismus sehr langsam gegenüber anderen OIE-Modellen ist [13], erhoffe ich mir, die Extraktionsgeschwindigkeit von IMoJIE mit einem GRU-

Decoder zu verbessern. Den GRU-Decoder konnte ich dabei ähnlich wie einen LSTM-Decoder implementieren, nur erhält die GRU keinen Cell-State als Input, wogegen alle restlichen Berechnungen mit den Attention-Mechanismen gleichbleiben.

GRU-Decoder			
Batch-Size	**F1-Opt**	**AUC**	**CUDA-Device**
128	52,00	**32,90**	TITAN RTX
64	00,50	00,10	TITAN RTX
32	51,30	31,10	TITAN RTX
16	**53,30**	**32,90**	TITAN RTX
8	00,40	00,00	TITAN RTX
8	51,30	31,10	TITAN V

Tab. 2: Die Modell-Leistung von IMoJIE mit GRU-Decoder bei unterschiedlichen Trainings-Batch-Sizes.

Batch-Size	**Trainingszeit in h**		**Testzeit in s**	
	GRU	**LSTM**	**GRU**	**LSTM**
128	7,7	8,6	395,9	448,1
64	7,1	8,1	547,8	509,6
32	8,8	9,5	235,2	533,5
16	11,3	11,4	359,4	433,2
8	18,3	19,1	537,4	442,1
8	16,9	18,4	247,4	292,7
Ø	**11,7**	**12,5**	**387,2**	**443,2**

Tab. 3: Die Anwendungszeiten von IMoJIE mit GRU- und LSTM-Decoder bei unterschiedlichen Trainings-Batch-Sizes. Die Testzeiten sind mit gleicher Batch-Size und GPU gemessen.

Wie Tab. 2 zeigt, konnte ich mit dem GRU-Decoder eine leicht niedrigere Modell-Leistung wie mit dem LSTM-Decoder erzielen, wobei ich mit der GRU zweimal AUC-Scores von etwa 0,00 erhalte, da diese Modelle ausschließlich Extraktionen aus Wortwiederholungen generierten. Ebenfalls konnte ich die Trainingszeit mit der GRU um durchschnittlich 49 Minuten verbessern, wie Tab. 3 zeigt. Aus den Messergebnissen kann ich daher schließen, dass die GRU als Decoder die Trainingsgeschwindigkeit verbessert, wogegen die Modell-Leistung der GRU etwas geringer als beim LSTM ausfällt aber dennoch nahe an dessen Maximalleistung liegt. Tendenziell sollte sich also die GRU bei anderen Hyperparametern geringfügig schlechter an die Testdaten anpassen können wie das LSTM.

Mit den Messwerten kann ich hingegen keine Erhöhung der Extraktionsgeschwindigkeit nachweisen, da jedes Modell eine andere Anzahl an Tokens und Extraktionen generiert und dementsprechend unterschiedlich viel Zeit beansprucht. Zwar erzielen Modelle mit GRU-Decoder durchschnittlich eine höhere Extraktionsgeschwindigkeit, jedoch hängt diese mit der geringeren Anzahl generierter Tupel und Tokens zusammen. Nach weiterer Untersuchung der Extraktionen mit einem Python-Skript stelle ich fest, dass die Modell-Leistung sowie die Testzeit sehr mit der Anzahl der Wortwiederholungen und der Anzahl generierter Extraktionen korrelieren. Hierbei generieren die GRU-Decoder durchschnittlich zwar weniger Extraktionen und Wortwiederholungen pro CaRB-Testsatz aber dafür sind diese Wortwiederholungen der GRU schwerwiegender und durchgängiger, während die Wortwiederholungen beim LSTM kurz und selten in betroffenen Extraktionen auftreten. Beide Decoder können trotzdem bei hoher Leistung qualitativ hochwertige Extraktionen generieren und hohe Leistungen erzielen, weswegen die GRU auch als Decoder für die Nutzung im IMoJIE-Modell geeignet ist.

Batch-Size	**Ø Extraktionen**		**Ø Tokens**		**Wiederholungen**	
	GRU	**LSTM**	**GRU**	**LSTM**	**GRU**	**LSTM**
128	3,0	5,8	399,7	487,0	6	322
64	10,0	7,7	500,0	474,5	638	331
32	2,3	9,7	297,7	493,5	8	612
16	2,4	3,8	356,2	408,2	1	10
8	10,0	4,9	500,0	426,5	640	222
8	2,3	2,6	297,7	330,3	8	2
Ø	**5,0**	**5,8**	**391,9**	**436,7**	**216,8**	**249,8**

Tab. 4: Tabelle mit der Analyse der Extraktionen. IMoJIE extrahiert für jeden der 641 Testsätze maximal 10 Tupel mit jeweils maximal 50 Tokens. Die Spalte „Wiederholungen“ gibt die Anzahl der Testsätze an, bei denen mindestens eine Token-Wiederholung vorkam.

4.4 Analyse der Encoder- und Decoder-Zeiten

Schließlich gab mir die umfangreiche Analyse der Extraktionen die Idee, die Modelle mit gleicher Anzahl an generierten Tokens zu vergleichen, um so eine Verbesserung der Extraktionsgeschwindigkeit nachzuweisen. Dabei erhalte ich für GRU- sowie LSTM-Decoder durchschnittlich fast identische Testzeiten bei etwa gleicher Anzahl generierter Tokens und Relations-Tupel. Somit beobachte ich keine schnellere Testgeschwindigkeit durch den GRU-Decoder, was meine Hypothese widerlegt.

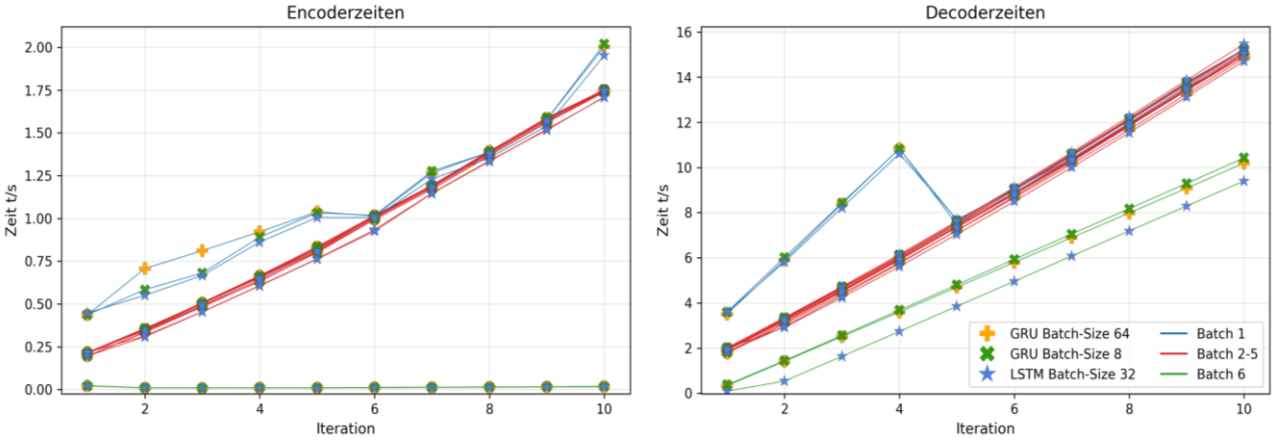

Abb. 9: Die Darstellung der Encoding- und Decoding-Zeiten während der Testphase. Die Daten stammen aus den Durchschnittswerten von 6 Messreihen, bei denen alle Modelle stets 10 Extraktionen mit der maximalen Anzahl 50 Tokens generierten*.

In Abb. 9 lässt sich beobachten, dass die Testzeiten für die Batches 2 bis 5 bei Encoding und Decoding linear mit der Anzahl der Tokens in der Input-Sequenz ansteigen. Dabei ist die Decoder-Zeit deutlich größer als die Encoder-Zeit und steigt ebenfalls steiler an. Weitere Laufzeitmessungen belegen hierbei, dass die Funktion *_gather_final_log_probs* durchschnittlich 97,63 % der gesamten Zeit eines Decoder-Schrittes beansprucht, also dass diese Funktion die hohen Decoder-Zeiten bestimmt. Hierbei werden in der Funktion *_gather_final_log_probs* die berechneten Copy- und generativen Wahrscheinlichkeiten für jeden der bis zu 50 Decoder-Schritte und für jeden einzelnen der maximal 500 Input-Tokens in jeder der $128 * 5$ Beam-Search-Sequenzen weiter angepasst, was den hohen Zeitanteil von 97,63 % erklärt. Dieser sehr zeitaufwendige, mit der CPU ausgeführte Vorgang übertrifft hiermit die Zeit zur Ausführung des Decoder-Modells und widerlegt meine Hypothese, dass sich die Extraktionsgeschwindigkeit mit der Verwendung eines GRU-Decoders signifikant verbessern lässt. Dagegen hätte die durchschnittliche gemessene Verbesserung von $1{,}609 * 10^{-4}$ Sekunden auf $1{,}502 * 10^{-4}$ Sekunden pro Ausführung von einer der 50 Decoder-Zellen kaum Einfluss auf die Modell-Geschwindigkeit, da die Berechnung der Additive Attention und der Copy-Attention zusammen mit dem Encoding im Verhältnis mehr Zeit beanspruchen.

Eine weitere Beobachtung lässt sich mithilfe der Encoder-Testzeiten machen. Dass nämlich die Encoder-Zeiten linear mit der Anzahl der Input-Tokens ansteigen, sollte nicht an der Ausführung des BERT-Transformers liegen, da dieser immer mit gleichgroßen Tensoren rechnet. Vielmehr sollte das mit der CPU berechnete Embedding für den linearen Anstieg sorgen. Messwerte bestätigten diese Annahme sowie auch die konstante und niedrige Ausführungszeit des Transformer-Encoders und dagegen eine hundert- bis tausendfach so große Ausführungszeit des BERT-Embeddings, welches linear mit der Anzahl der Tokens ansteigt.

* Nur das LSTM der Batch-Size 32 generierte immer 11 Tokens in der ersten Iteration des 6. Test-Batches.

Schließlich muss noch die Ursache für die Abweichungen im 1. Test-Batch besprochen werden, die in den ersten 4 Iterationen im ersten Test-Batch auftreten. Hierbei werden zwei Prozesse parallel nebeneinander ausgeführt, wobei die zweifache Ausführung der Funktion *_gather_final_log_probs* die Testzeit in den betroffenen Iterationen nahezu verdoppelt. In der 5. Iteration wird der zweite dieser Prozesse durch einen Out-of-Memory-Error terminiert, womit die Testung normal weiterläuft. Insgesamt sind diese und weitere Abweichungen insignifikant und beeinflussen die Beobachtungen minimal, weshalb ich mein Vorgehen präzise und verlässlich halte. Die Ursache für die doppelte Ausführung ist mir dabei am Code nicht ersichtlich.

5. Fazit

In dieser Belegarbeit konnte ich erfolgreich den GRU-Decoder für IMoJIE implementieren und nachweisen, dass dieser im IMoJIE-Modell ebenfalls wie der LSTM-Decoder qualitative Extraktionen bei ebenbürtiger Leistung generieren kann. Dabei ist der LSTM-Decoder nach den Messergebnissen jedoch etwas leistungsfähiger und deshalb dem GRU-Decoder vorzuziehen. Die Modell-Leistung ist für OIE-Modelle wichtig, um in kürzerer Zeit mehr brauchbare und nicht-redundante Informationen extrahieren zu können, wobei ein GRU-Decoder als weiterer signifikanter Hyperparameter eingesetzt werden kann, um hiermit ein leistungsfähiges IMoJIE-Modell beim Trainieren zu finden. Vorteilhaft ist das bei wenigen verfügbaren Hyperparametern, wie z. B. bei der TITAN V mit 12 GB RAM oder auch bei Einsatz von größeren BERT-Encodern zur Leistungssteigerung.

Des Weiteren konnte ich mithilfe der GRU die Trainingszeit zum Trainieren des IMoJIE-Modells um durchschnittlich 49 Minuten reduzieren, wobei dies eine kleine Verbesserung darstellt und meiner Hypothese entspricht. Hingegen konnte ich durch meine Beobachtung widerlegen, dass die Extraktionsgeschwindigkeit durch den Austausch mit einer GRU signifikant verbessert wird, da die Testzeit zu stark von der Berechnung der Additive Attention und Copy-Attention abhängt. Dennoch ist besonders die unerwartete Beobachtung von der sehr großen Laufzeitbeanspruchung der Funktion *_gather_final_log_probs* mit 97,63 % äußerst relevant für zukünftige Verbesserungsansätze für die Extraktionsgeschwindigkeit. Voraussichtlich kann diese Funktion wesentlich optimiert und die Extraktionsgeschwindigkeit von IMoJIE verbessert werden. Die weitere Beobachtung, dass das Embedding die Encoder-Zeit maßgeblich bestimmt, zeigt hierbei, dass ein größerer und leistungsfähiger BERT-Encoder in der Testphase wenig zusätzliche Zeit bei Ausführung benötigt. Somit bedeutet diese Beobachtung, dass Leistung von IMoJIE mithilfe eines größeren BERT-Encoders verbessert werden kann, ohne die Extraktionsgeschwindigkeit stark zu erhöhen.

Insgesamt bin ich mit den Ergebnissen meiner Belegarbeit zufrieden und konnte trotz des anspruchsvolleren Themas meine Hypothese dank der schrittweisen Analyse vollständig untersuchen und dabei auch relevante Beobachtungen für die weitere Forschung am IMoJIE-Modell machen. Ich plane diese Beobachtungen in einer besonderen Lernleistung genauer zu untersuchen und meine Forschung weiter über den begrenzten Umfang einer Belegarbeit auszuarbeiten. Zwar habe ich meine Hypothese vollständig ausgearbeitet, jedoch bieten meine Beobachtungen die seltene Möglichkeit, große Verbesserungen vergleichsweise leicht zu erzielen und dabei auch relevante theoretische Erkenntnisse zu gewinnen. Demnach plane ich die Laufzeitoptimierung der Funktion *_gather_final_log_probs* und das Training mit größeren BERT-Encodern durchzuführen. Hierbei erhoffe ich mit dem Einsatz von größeren BERT-Encodern eine nahezu unveränderte Extraktionsgeschwindigkeit bei einer signifikanten Leistungssteigerung nachweisen zu können.

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] 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.

[4] 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.

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

[6] 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.

[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] 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.

[9] J. Gao und S. Tiwary, „Efficiently and effectively scaling up language model pretraining for best language representation model on GLUE and SuperGLUE", [Online]. https://www.microsoft.com/en-us/research/blog/efficiently-and-effectively-scaling-up-language-model-pretraining-for-best-language-representation-model-on-glue-and-superglue/. [27. November 2021].

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

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

[12] K. Kolluru, S. Aggarwal, V. Rathore, Mausam und S. Chakrabarti, „IMoJIE: Iterative Memory-Based Joint Open Information Extraction", arXiv:2005.08178v1, 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] Y. Ro, Y. Lee und P. Kang, „Multi2OIE: Multilingual Open Information Extraction Based on Multi-Head Attention with BERT", arXiv:2009.08128v2, 2020.

[15] 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.

[16] K. Kolluru. [Online]. https://github.com/dair-iitd/imojie/tree/master/imojie/models. [12. Dezember 2021].

[17] K. Al-Sabahi, Z. Zuping und Y. Kang, „Bidirectional Attentional Encoder-Decoder Model and Bidirectional Beam Search for Abstractive Summarization", arXiv:1809.06662v1, 2018.

[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. Bonn: Rheinwerk Verlag, 2020.

[20] D. Silver, T. Hubert und J. Schrittwieser, „Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm", arXiv:1712.01815v1, 2017.

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

[22] Y. H. Liu, Python Machine Learning By Example, 3. Auflage. Birmingham: Packt Publishing Ltd., 2020.

[23] A. Gulli, A. Kapoor und S. Pal, Deep Learning with Tensorflow 2 and Keras, 2. Auflage. Birmingham: Packt Publishing Ltd., 2019.

[24] S. Hochreiter, „Long Short-term Memory", 1997. [Online]. ResearchGate: https://www.researchgate.net/publication/13853244_Long_Short-term_Memory. [12. Dezember 2021].

[25] F. Gers, „Recurrent nets that time and count", ResearchGate: https://www.researchgate.net/publication/3857862_Recurrent_nets_that_time_and_count, 2000.

[26] 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.

[27] R. Rana, „Gated Recurrent Unit (GRU) for Emotion Classification from Noisy Speech", in *arXiv:1612.07778v1*, 2016.

[28] R. Jozefowicz, W. Zaremba und I. Sutskever, „An Empirical Exploration of Recurrent Network Architectures", 2015. [Online]. http://proceedings.mlr.press/v37/jozefowicz15.pdf. [13. Dezember 2021].

[29] D. Z. Liu und G. Singh, „A Recurrent Neural Network Based Recommendation System", [Online]. https://cs224d.stanford.edu/reports/LiuSingh.pdf. [13. Dezember 2021].

[30] J. Chung, C. Gulcehre, K. Cho und Y. Bengio, „Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling", arXiv:1412.3555v1, 2014.

[31] O. Vinyals, M. Fortunato und N. Jaitly, „Pointer Networks", arXiv:1506.03134v1.

[32] M. Prikryl, „WinSCP", [Online]. https://winscp.net/eng/download.php. [13. Dezember 2021].

[33] 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.

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

Danksagung

Ich möchte mich allerherzlichst bei Elke Katz dafür bedanken, dass sie mir dieses Projekt mit dem schwierigeren Thema ermöglicht hat. Sie half mir dabei den geeigneten Projektbetreuer Prof. Andreas Maletti zu finden und schafft außerdem mir und 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 ständig dazu motivieren konnte, weiter an meinem Projekt zu arbeiten. Der Universität Leipzig möchte ich auch großen Dank aussprechen, dass sie mir einen ihrer Server mit zwei leistungsfähigen Grafikkarten kostenlos für meine Versuche zur Verfügung stellten.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Belegarbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe.

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

Leipzig, 23.01.2022 (Ort, Datum) Henning Beyer (Unterschrift)