Text Summarization mit Hilfe von neuronalen Modellen auf Basis von Online-Artikeln der Süddeutschen Zeitung

Daniel Christ, Christian Wirtgen
{Daniel.Christ, Christian.Wirtgen}@student.hs-rm.de
Dozent: Prof. Adrian Ulges
Adrian.Ulges@hs-rm.de

Masterprojekt
Sommersemester 2020
Hochschule RheinMain

Zusammenfassung

In diesem Projekt wurde ein neuronales Modell zur extraktiven Generierung von Zusammenfassungen von deutschen Nachrichtenartikeln entwickelt. Der Datensatz besteht aus Online-Artikeln der Süddeutschen Zeitung, bei denen die Ground-Truth-Labels aus einleitenden Stichpunkten generiert wurden. Das Modell verwendet einen vortrainieren BERT-Encoder zur Berechnung von einer Satz- und Dokumenten-Repräsentation. Je nach Ansatz des Dataloaders werden entweder Satz-Tokens oder das gesamte Dokument mit BERT codiert. Durch die Mittelung dieser Codierung wird die Dokumenten-Repräsentation berechnet. Für die Satz-Repräsentationen werden die enstprechenden Sätze maskiert und gemittelt. Beide fließen in einen Classifier ein, der die Wahrscheinlichkeiten zur Zugehörigkeit zu der Zusammenfassung lernt. Das Modell erreichte auf Validierungsdaten einen F1-Score von 0.26 im Vergleich zu der Lead3Baseline von 0.13 des hier verwendeten Datensatzes. Eine manuelle Evaluation der Ergebnisse hat gezeigt, dass die vom Modell ausgewählten Sätze mit den generierten Ground-Truth Zusammenfassungen der Stichpunkte vergleichbar sind.

Inhaltsverzeichnis

1 Einleitung

Dieses Masterprojekt beschäftigt sich mit Text Summarization, einer Unterdomäne von NLP (Natural Language Processing). Text Summarization hat das Ziel, aus Textdokumenten eine kohärente und flüssige Zusammenfassung zu erstellen und dabei die wesentlichen Inhalte des Textes wiederzugeben. Bei diesem Thema wird zwischen abstraktiver und extraktiver Text Summarization unterschieden. Bei dem extraktiven Ansatz werden typischerweise Sätze direkt aus dem Originaldokument ausgewählt, die den Text möglichst gut beschreiben sollen. Die Zusammenfassung wird dann durch die Aneinanderreihung der ausgewählten Sätze gebildet. Bei der abstrakten Methode werden zunächst Inhalte aus dem Quelldokument extrahiert und anschließend werden komplett neue Sätze generiert, die nicht zwingend das Vokabular aus dem Dokument verwenden muss. Bei diesem Projekt kommt aufgrund des verwendeten Datensatzes (Kapitel 3) die extraktive Version zum Einsatz.
Das Projekt kann in drei unterschiedliche Phasen gegliedert werden: Zunächst wird nach verwandten Arbeiten (Kapitel 2) recherchiert und diese in der Arbeitsgruppe vorgestellt. Danach wird mit der Planung und Implementierung (Kapitel 4) des Modells begonnen. Zum Schluss wird das Modell trainiert und fine-getuned und die Ergebnisse evaluiert (Kapitel 5).

2 Verwandte Themen und Arbeiten

In der Konzeptionsphase des Projektes wurde zunächst nach verwandten Arbeiten im Bereich der Text Summarization recherchiert, um bereits erfolgreiche Konzepte aufgreifen zu können. Als Idee für das grobe Grundkonzept eines einfach strukturierten, extraktiven Modells wurde dabei die Arbeit *SummaRuNNer: A Recurrent Neural Network based Sequence Model for Extractive Summarization of Documents* [12] ins Auge gefasst. Eine Übersicht über das Modell von SummaRuNNer gibt Abbildung 1. SummaRuNNer verfügt über zwei Schichten zur Repräsentation von Wörtern und Sätzen. Diese wurden mit (bidirektionalen) rekurrenten neuronalen Netzwerken (RNNs) realisiert. Die erste Schicht dient zur Generierung von Wortrepräsentationen, die auf Satzebende gemittelt werden und der zweiten RNN-Schicht dann als Eingabe dienen. Die Ausgabe der zweiten Schicht repräsentiert dann die Sätze des Dokuments. Um das gesamte Dokument zu repräsentieren werden dann wiederum die Satzrepräsentationen gemittelt. In einer linearen Entscheidungsschicht fließen dann die Satz und Dokumentreräsentationen, zusammen mit positional Embeddings ein, um eine Wahrscheinlichkeit für die Zugehörigkeit zur Zusammenfassung zu lernen.

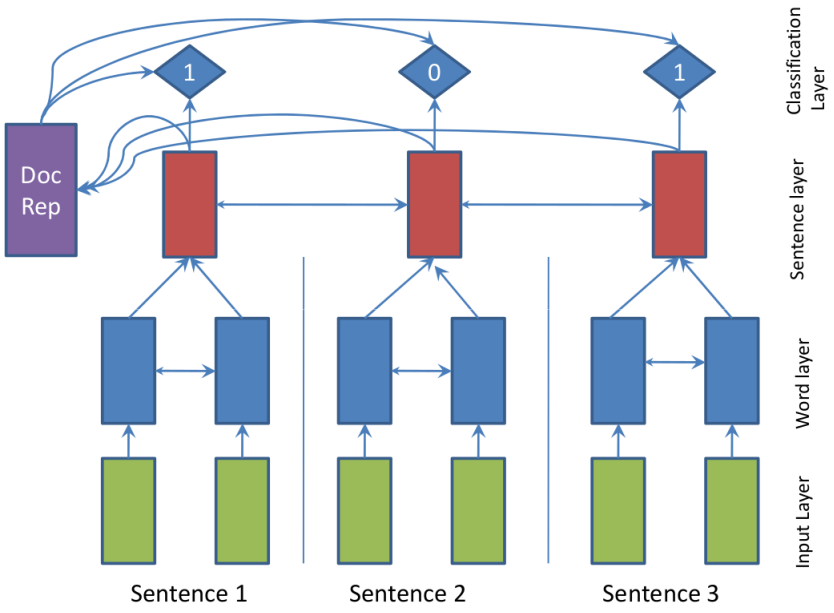

Abbildung 1: SummaRuNNer [12, S. 2, Figure 1]

Das hier verwendete Modell lehnt sich vom Konzept her an die Architektur von SummaRuNNer an, verwendet jedoch keine zweite Schicht, sondern bildet die Satzrepräsentationen direkt. Dafür soll aufgrund der großen Erfolge der Transformer Architektur [15] in den letzten Jahren, anstelle der RNNs ein Attention basiertes Modell zum Einsatz kommen. RNNs haben das Problem, weit voneinander entfernte Token in Kontext zu stellen. Im Gegensatz dazu lässt Attention den Kontext jedes Tokens mit jedem anderen Token in die Berechnung der Embeddings einfließen. Die Wahl des Encoders fiel dabei auf ein vortrainiertes BERT-Modell [3], das während des Trainings fine-getuned werden soll. Die Token der Wörter von Teilen des Dokuments, oder des gesamten Dokuments, sollen auf Satzebene maskiert und von BERT verarbeitet werden, um dann gemittelt und an eine lineare Schicht weiter gegeben zu werden.
Die Arbeit *Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting (2018)* [2] ergänzt den extraktiven Schritt mit einem abstraktiven Umschreiben der ausgewählten Sätze. Obwohl dies in diesem Projekt noch nicht umgesetzt werden konnte, ist dieser Ansatz aber als spätere Weiterführung interessant (siehe Kapitel 7).

3 Datensatz

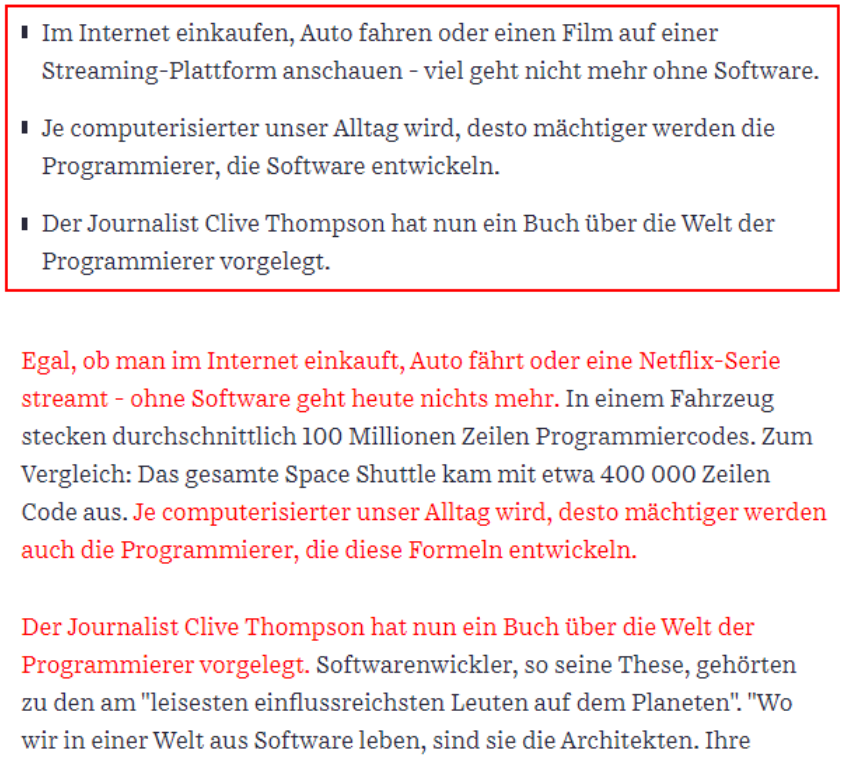

Abbildung 2: Auszug aus einem Trainingsartikel: Rote Box beinhaltet die Stichpunkte, rote Sätze markieren die Sätze, welche den Stichpunkten am ähnlichsten sind.

Das Modell soll auf einen bestehenden Datensatz[1] trainiert werden. Dieser Datensatz besteht aus Online-Artikeln der Süddeutschen Zeitung[2] (SZ) aus dem Zeitraum Januar 2013 bis April 2019 aus verschiedensten Kategorien (u.a. Politik, Wirtschaft, Sport, ...). Die gesammelten Artikel verfügen über dieselben Merkmale: Am Anfang des Textes müssen zusammenfassende Stichpunkte vorkommen. Dabei wurden nur Artikel aufgenommen, die zwischen zwei und vier Stichpunkte verfügen und mindestens doppelt so viele Paragraphen haben wie Stichpunkte. Das Modell soll später anhand von Ground-Truth-Labels trainiert werden, die mögliche Zusammenfassungssätze des Artikels markieren. Um diese Labels zu generieren, werden mit Hilfe von BERT die ähnlichsten Sätze des Textes zu jedem Stichpunkt herausgesucht. Diese erhalten dann das Label 1 und die

[1] `https://gitlab.cs.hs-rm.de/adrian.ulges/sz-crawler`
[2] `https://www.sueddeutsche.de/`

restlichen Sätze das Label 0. Abbildung 2 zeigt die Stichpunkte des Artikels in der roten Box, sowie den jeweils ähnlichsten Satz zu jedem Stichpunkt in rot. Da sich der Stichpunkt jedoch sehr stark von dem ausgewählten Satz unterscheiden kann (besonders inhaltlich), wurden Schwellwerte eingeführt, die die Qualität des Ground Truths sicher stellen sollen. Die unterschiedlichen Kategorien der Artikel haben jeweils andere Schwellenwerte. Erst wenn die Ähnlichkeit zwischen dem Stichpunkt und dem Artikelsatz den Schwellwert übersteigt, ist dieser qualitativ gut genug um als Trainingsobjekt in Frage zu kommen. Aus den Artikeln konnten so 31.389 Stichpunkte extrahiert werden, wovon 8000 Sätze als Trainingsobjekte geeignet sind. Jedes Trainingsobjekt entspricht nun einem Sample, welches zum Training des Modells genutzt werden kann.


```
"0": {
    "article": [
        "Bei einem Selbstmordanschlag in der afghanischen Hauptstadt Kabul kamen mindestens 63 Menschen ums Leben.",
        "Ein Selbstmordattentaeter sprengte sich am Samstagabend in einer Hochzeitshalle in die Luft.",
        "Mindestens 182 Menschen wurden verletzte, wie ein Sprecher des Innenministeriums mitteilt."
    ],
    "ground_truth": [
        0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
    ],
    "bullets": [
        {
            "bullet": "Der sogenannte Islamische Staat hat sich zu dem Anschlag bekannt.",
            "hit": "Die Terrormiliz Islamischer Staat (IS) hat sich zu dem Anschlag bekannt.",
            "index_hit": 6
        },
        {
            "bullet": "Laut eines Ueberlebenden befanden sich 1200 Gaeste auf der Feier.",
            "hit": "Ein anderer Ueberlebender sagte, es haetten sich 1200 Gaeste auf der Feier befunden.",
            "index_hit": 5
        }
    ],
    "key": "Mehr_als_60_Tote_nach_Anschlag_in_Kabul.json"
},
```


Abbildung 3: Ein Artikel (gekürzt) besteht aus den einzelnen Sätzen, Array mit dem Ground Truth, den Stichpunkten, sowie dem Namen der Quelldatei

Während dieses Projekts wurden zusätzlich zu den schon vorhandenen Samples alle in Frage kommenden Artikel zwischen dem Zeitraum April 2019 bis Mai 2020 gesammelt und dem Datensatz hinzugefügt. Dafür musste der Crawler umgeschrieben werden, da sich das Web-Layout der SZ in der Zwischenzeit maßgeblich verändert hatte. Die Anzahl wurde auf 38.564 Stichpunkte und 9100 Trainingssamples erhöht. Die Schwellenwerte der einzelnen Kategorien wurden nach der Hinzunahme nochmals überprüft, da diese sich durch die Änderung der Daten verschieben können. Eine händische Analyse ergab jedoch, dass eine Änderung der Schwellenwerte nicht nötig ist. Die Trainingssamples wurden in Validierungs- und Trainingsdaten aufgeteilt. Die Validierungsdaten verfügen über 300 Samples, die Trainingsdaten hingegen über die restlichen 8800 Samples. Die Daten liegen als JSON-Dateien vor (Abbilding 3). Diese beinhalten

für jeden Artikel die einzelnen Sätze des Textes (“article“), die Stichpunkte (“bullet“) mit ihren jeweils ähnlichsten Treffern (“hit“), sowie den Artikelnamen bzw. den Name der Quelldatei (“key“). Des Weiteren verfügt jeder Artikel über einen Ground Truth Vektor, welcher angibt, ob ein Satz zur Zusammenfassung gehören soll oder nicht. Der Ground Truth Vektor hat die gleiche Größe wie das Feld “article“, welches aus Übersichtlichkeitsgründen in Abbildung 3 gekürzt wurde.
Um einen Überblick über den Datensatz zu bekommen, wurden verschiedene Aspekte untersucht. So hat ein Artikel im Durchschnitt 32 Sätze und ein einzelner Satz hat im Mittelwert 15 Wörter. Ein Satz wird im Durchschnitt von 27 Tokens (N-Gramme + Special Tokens) repräsentiert.

4 Modell

Bei der Planung und Implementierung des Modells kristallisierten sich vier verschiedene Hauptkomponenten heraus: Dataset, Tokenizer, Extractor und Abstractor. Im ersten Schritt wird das Dokument, also der eigentliche Artikel, codiert. Der Tokenizer teilt die einzelnen Sätze hierbei in N-Gramme auf. Mit Hilfe dieser Tokens kann der Satz dann codiert werden. Dieser Schritt soll durch ein vortrainiertes Modell durchgeführt werden. Wenn ein Satz codiert wurde, kann dann daraus die Satz-Repräsentation berechnet werden. Zusätzlich soll es auch eine Dokumenten-Repräsentation geben, welche zusammen mit der Satz-Repräsentation in einen Classifier einfließt. Der Classifier hat die Aufgabe zu bestimmen, ob der Satz zu der Zusammenfassung gehören soll oder nicht. An dieser Stelle wäre der extraktive Ansatz beendet. Für die weitere abstraktive Bearbeitung der ausgewählten Sätze wird noch ein zusätzlicher Abstractor benötigt, welche die ausgewählten Sätze des Extrators umschreibt und anpasst. Die Abstraktor Komponente konnte aus zeitlichen Gründen nicht mehr implementiert werden. Die geplante Architektur ist in Figur 4 abgebildet.
Für die Codierung der Sätze kommen verschiedene Modelle in Frage, die bereits in Kapitel 2 vorgestellt wurden. Bei der Auswahl war es wichtig, dass das Modell die deutsche Sprache unterstützt, weshalb einige Modelle nicht in Frage kamen. Die Wahl viel zunächst auf BERT [3]. In Form eines Webservers (bert-as-service [17]) konnten hier aus Sätzen fertige BERT-Embeddings berechnet werden. Dies hatte jedoch Nachteile: Die Berechnung dauerte sehr lange und man ist immer auf einen erreichbaren Webserver angewiesen. Es lag daher der Entschluss nahe, das BERT-Modell für das Encoding direkt in das Gesamtmodell zu integrieren. Dieser Ansatz hat auch den Vorteil, dass man während des Trainings BERT direkt auf den Daten fine-tunen kann.

Als konkretes BERT Modell wurde DistilBERT [13] ausgewählt, da dieses Modell 40% kleiner als das herkömmliche BERT ist, aber noch 97% der Kapazität besitzt. Für die konkrete PyTorch Implementierung wurde das DistilBERT von Hugginface [16] genommen. Als Tokenizer wird der dazugehörige DistilBERT-Tokenizer genutzt, welcher ebenfalls vortrainiert ist. Da das vortrainierte DistilBERT nur maximal 512 Tokens gleichzeitig verarbeiten kann, wurden die BERT-Position-Embeddings auf 2048 erweitert. Dabei wurde wie bei der Arbeit *Text Summarization with Pretrained Encoders* [10] vorgegangen. Mit dieser Position-Embedding-Größe können jetzt nicht nur Sätze, sondern auch ganze Artikel codiert werden.
Mit fortlaufender Entwicklung wurde auch PyTorch Lightning [4] in das Projekt integriert. Dies ermöglicht die Strukturierung des Quellcodes und vereinfacht die Implementierung von Features wie z.B. das Abspeichern von Checkpoints des Modells. In diesem Schritt wurde auch die DataSet Klasse implementiert, welche die Artikel-Daten lädt und für den weiteren Gebrauch aufbereitet. Es wurde klar, dass ein Sample im Dataset auf verschiedene Arten gehandhabt werden kann und damit drei verschiedene Grundarten entstehen:

- Sliding-Window
- Dokumenten-weise
- Sample-weise

Bei dem *Slidings-Window* Ansatz besteht ein Sample nicht nur aus einem einzigen Satz, sondern aus mehreren, welche direkt vor oder nach dem Hauptsatz stehen. Dadurch wird zusätzlicher Kontext in einem Sample berücksichtigt. Wie viele Sätze als Kontext hinzugenommen werden, kann durch einen freien Parameter bestimmt werden. Die Dokumentenrepräsentationen werden hier im Vorfeld berechnet und zur Trainingslaufzeit aktualisiert. Der Vorteil dieser Methode ist, dass mehr Samples gleichzeitig in eine Batch in den Grafikspeicher passen.
Bei dem *Dokumenten-weisen* Ansatz hingegen besteht ein Sample aus dem gesamten Dokument. Hier werden alle Sätze dieses Dokuments auf einmal verarbeitet. Dies hat den Vorteil, dass die Satz- und Dokumentenrepräsentation sehr einfach zur Trainingslaufzeit berechnet werden können. Dokumente die zu lang sind (>2048 Tokens), müssen entsprechend gekürzt oder weggelassen werden. Auch können in einem Batch nicht unterschiedliche Sätze aus verschiedenen Dokumenten vorkommen, obwohl dies wünschenswert wäre.
Genau diese Nachteile versucht der *Sample-weise*-Ansatz zu ändern. Hier gibt es für jeden Satz im Dokument ein Sample, dieser besteht jedoch nicht nur aus den Satz-Tokens, sondern aus dem gesamten Dokument. Dadurch können in einem

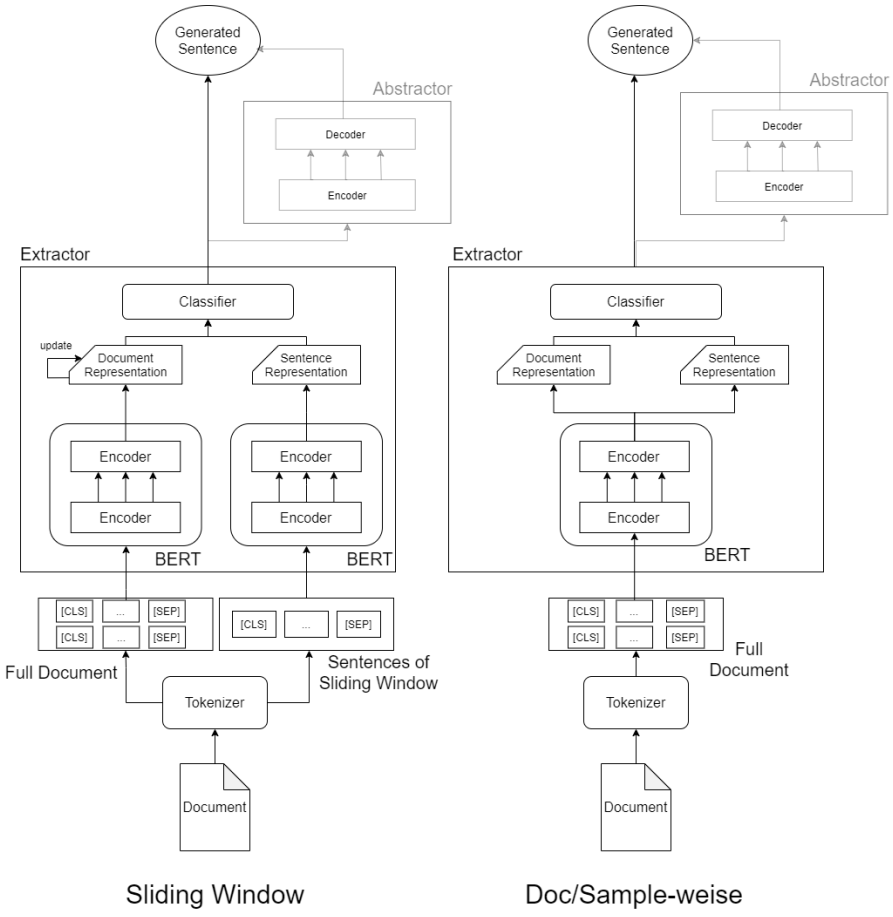

Abbildung 4: Architekturelle Unterschiede der verschiedenen Ansätze. Der Abstraktor stellt eine nicht implementierte, aber geplante Komponente dar.

Batch unterschiedliche Sätze aus verschiedenen Dokumenten vorkommen und trotzdem können die Satz- und Dokumentenrepräsentationen vollständig zur Trainingslaufzeit ausgerechnet werden. Durch den hohen Speicherbedarf passen allerdings nur wenige Samples auf eine Grafikkarte.
Abbildung 17 zeigt den Graph mit dem Sample-weise bzw. Dokumenten-weise Ansatz. Der Input des Models besteht aus den einzelnen Samples mit den Tokens, den Masken (zur Maskierung der relevanten Sätze des Gesamtdokuments) und den jeweiligen Dokumentenlängen (#Gesamt Tokens). Der input_ids Tensor wird mit dem DistilBERT Modell codiert. Zusammen mit den Dokumentenlängen kann nun die Dokumentenrepräsentation errechnet werden (*doc_reps*). Mit Hilfe der Masken und dem Output des BERT-Modells wird die Satzrepräsentation berechnet (x Tensor). Anschließend wird ein Positional-Embedding auf den Tensor hinzugerechnet und man erhält die fertige Satzrepräsentation (*sentence_reps*). Die Satzrepräsentation wird mit der Dokumentenrepräsentation konkateniert (*input_59*) und wird dann an den Classifier weitergereicht. Nach einer ersten Linear-Schicht wird der Batch normalisiert und durch die Dropout-Schicht zu einer zweiten Linearschicht weitergegeben. Mit einen abschließenden (log-)Softmax werden die Wahrscheinlichkeiten für jeden Satz ermittelt.

Anders hingegen ist der Ablauf des Sliding-Window Ansatz. Hier werden die Dokumenten-Repräsentationen im Vorfeld durch BERT-berechnet und in ein Dictionary abgespeichert. Ein Sample besteht hier aus 7 Sätzen: Hauptsatz plus drei Sätze, welche vor/nach dem Hauptsatz stehen. Der Input des Modells besteht aus den Samples, den Masken (zur Selektierung des Hauptsatzes), den vorberechneten Dokumenten-Repräsentationen und der Anzahl der Sätze pro Artikel. Nach der Codierung der Samples durch BERT wird mit den Masken die einzelnen Satz-Repräsentationen ausgerechnet. Danach werden die Dokumenten-Repräsentationen aktualisiert.

$$NewDocRep = (1 - \frac{1}{K}) \cdot CurrentDocRep + \frac{1}{K} \cdot SentenceRep$$

Dafür wird ein bestimmer Anteil der aktuellen Dokumenten-Repräsentation (*CurrentDocRep*) mit einem Anteil der Satz-Repräsentation (*SentenceRep*) aufaddiert. K ist hierbei die Satzlänge des Artikels. Danach wird das Positional Embedding auf die Satz-Repräsentation hinzugefügt und mit der Dokumentenrepräsentation konkateniert. Nun wird konkatenierte Repräsentation an den Classifier weitergegeben. Genau wie bei den beiden anderen Ansätzen geht der Input durch zwei Linear-Schichten inklusive Layer Normalization und Dropout. Die einzelnen Wahrscheinlichkeiten werden dann durch ein (log-)Softmax ermittelt.

5 Experimente & Ergebnisse

Dieses Kapitel beschäftigt sich mit verschiedenen Experimenten und deren Ergebnissen. Ziel der Experimente war es, dass Modell so zu optimieren, dass ein möglichst gutes Endergebnis in Form der F1-Kategorien (Kapitel 5.2) vorliegt. Dazu wurden nicht nur Hyperparameter, wie Lernrate (Kaptiel 5.4), optimiert, sondern auch architekturelle Veränderungen, wie Layer Normalization und Dropout (Kapitel 5.8), vorgenommen.

5.1 Basismodell & Hyperparameter

Für alle nachfolgenden Experimente wurde ein Basismodell ausgewählt, damit die einzelnen Training-Runs miteinander vergleichbar sind. Von den drei verschiedenen Dataset-Ansätzen wurde der Sample-wise genommen, da dieser am vielversprechendsten in den ersten Trainingsversuchen war. In Tabelle 1 sind die einzelnen Hyperparameter und deren Default-Werte aufgelistet. In den Experimenten wurden immer die Default-Werte genommen, falls nicht, sind diese Änderungen vermerkt.

Hyperparameter	Default Values
Lernrate	1e-6
Batchsize	4
Accumulation Grad	4
Dropout Rate	0.3
Label Smoothing Grad	0.2
Class Weights	1/15
Positional Embedding	Sinusoidal

Tabelle 1: Hyperparameter und deren Default-Werte

Die Lernrate bestimmt wie stark das Modell beim Training lernt, sprich wie stark die Updates der Backpropagation ausfallen. Da es sich um eine Komponente des Modells um das vortrainierte BERT handelt, muss diese Komponente nur fine getuned werden. Es ist wird daher geraten, die Lernrate nicht zu hoch auszuwählen (siehe auch Kapitel 5.4). Die Batchsize gibt an, wie viele Samples bei einem Trainingsschritt genommen werden. Der limitierende Faktor ist hierbei die Speicherkapazität der Grafikkarte. Zum Training stand eine Nvidia Quadro P6000 mit 24 GB Speicher zur Verfügung. Die größtmögliche Batchsize für diese Karte lag bei 4. Mit Gradient Accumulation wird die Globale Batch in mehrere Mini-Batches aufgeteilt um den limitierenden Faktor der Speichergröße zu umgehen (siehe Kapitel 5.7). Mit dem Default Accumulation Grad von 4 entspricht die globale Batchsize 16, was im Durchschnitt genau einem halben Artikel entspricht. Die Dropout Rate bestimmt die Wahrscheinlichkeit, ob ein

Input der Schicht auf Null gesetzt wird (siehe Kapitel 5.8). Der Label Smoothing Grad bestimmt, wie stark der Label Tensor geglättet wird (siehe Kapitel 5.9). Die Class Weights geben an, wie stark eine Klasse in der Loss-Funktion berücksichtigt werden soll. Da es zwei Klassen gibt, wird hier das Verhältnis (1:15) zwischen den zwei Klassen angegeben (siehe auch Kapitel 5.5). Dem Modell stehen drei unterschiedliche Möglichkeiten für das Positional Embedding zur Verfügung. Als Default Wert wurde hier das Sinusoidal Embedding ausgewählt (siehe Kapitel 5.3).

5.2 F1 Metrik

Zur automatischen Evaluierung der Trainingsergebnisse wurde die *F1 Metrik* verwendet, die als Kombination von Genauigkeit und Sensitivität beide Werte gleich gewichtet. Es wurden zwei verschiedene Berechnungsmethoden angewendet, die beide jeweils in den Trainings- und Validierungsepochen durchgeführt und verfolgt wurden.
Zum einen wurde der *Mikro-F1* Wert verwendet, indem alle *Richtig Positiven (TP)*, *Falsch Postivien (FP)* und *Falsch Negativen (FN)* Ergebnisse der kompletten Epoche gezählt werden, um anschließend einen einzelnen Wert

$$F_1 = \frac{TP}{TP + 0.5 \cdot (FP + FN)}$$

zu berechnen.
Im Gegensatz dazu wurde der *Makro-F1* Wert berechnet, der die jeweiligen F1-Scores der einzelnen Dokumente der momentanen Epoche berechnet und anschließend mittelt. Da das Modell die einzelnen Sätze der Dokumente zufällig in unterschiedlichen Batches abarbeitet, wurden die Zählungen in Python Dictionaries des jeweiligen Dokumentes abgespeichert und am Epochenende berechnet, gemittelt und zurückgesetzt.
Zum Vergleich der Ergebnisse kann die *Lead3Baseline* (die ersten 3 Sätze des Dokuments als Zusammenfassung) herangezogen werden, die auf dem vorliegenden Datensatz, mit den beschriebenen F1-Berechnungen, Werte von 0.13 (Makro und Mikro) erreichte. In den folgenden Kapiteln wird in den Tabellen die Lead3-Baseline zu Übersicht mit angegeben und unter dem Namen L3-BL aufgeführt.

5.3 Positional Embeddings

Die Position und die Reihenfolge der Wörter ist für jede Sprache sehr wichtig. Sie definieren zum Beispiel die Grammatik oder die eigentliche Semantik des Satzes.

Die Position der Wörter ist für das Modell deshalb eine wichtige Information. Positional Embeddings beschreiben genau diese Position von Wörtern bzw. der Tokens. Transformer Modelle wie BERT verfügen über Sinusoidal Positional Embeddings. Hier werden Sinus- und Cosinus Funktionen über verschiedenen Frequenzen benutzt, um Informationen über die absolute und relative Position der Wörter zu injizieren [15].
Es wurde deshalb in einem Experiment ausprobiert, ob die zusätzliche Injektion von Positions-Informationen einen Einfluss auf das Ergebnis hat. Dazu wurden zwei verschiedene Positional Embedding Typen ausprobiert. Der erste Typ ist das oben beschriebene Sinusoidal Positional Embedding. Als Alternative wurden zwei lernbare Embedding Layers in das Modell integriert, welche ebenfalls zusätzliche Informationen über die relative und absolute Position der Sätze liefern. Die Berechnung dieser absoluten und relativen Positional Embeddings wurde aus dem SummaRuNNer Paper [12] übernommen. Die absoluten Embeddings codieren die Nummer des Satzes im Dokument, wohingegen die relativen den Abschnitt des Dokumentes codieren, in dem sich der Satz befindet (hier: in welchem Zehntel des Textes befindet sich der Satz?). Die Embedding Layer werden beim Trainingprozess automatisch mit gelernt. Um beide Typen miteinander vergleichen zu können, wurde ein Run ohne zusätzliche Positional Embeddings durchgeführt. Wie man in Tabelle 5 sehen kann, ist das Sinusoidal Embedding bei allen F1-Scores am besten. Dies ist ein Zuwachs von 2-3% gegenüber dem Modell ohne zusätzliches Positional Embedding. Auch die lernbaren Embeddings (Abs.+ Rel.) bringen ein Gewinn von circa 1%.

Embedding Type	Train Micro F1	Train Macro F1	Val Micro F1	Val Macro F1
Sinusoidal	**25.53**	**19.99**	**25.57**	**20.18**
Abs. + Rel.	23.68	19.11	23.64	19.20
None	22.69	0.1815	22.60	18.22
(L3-BL)	-	-	12.89	12.58

Abbildung 5: Vergleich der Positional Embeddings (Settings: Epoch=1)

5.4 Lernrate

Das PyTorch Lightning Modul bietet einen Lernrate Finder[3], welcher eine gute initiale Lernrate zum Trainieren sucht. Abbildung 6 zeigt zwei Vorschläge

[3] `https://pytorch-lightning.readthedocs.io/en/latest/lr_finder.html`

des Finders, einmal mit deaktivierten und einmal mit aktivierten Gradient Accumulation (siehe Kapitel 5.7). Wie man an dem linken Graphen sehen kann, schlägt der Finder eine Lernrate von circa 0.00007 vor. Diese liegt jedoch auf einen Plateau, welches zwischen 10^{-5} und 10^{-4} stark angestiegen ist und bei 10^{-3} wieder fällt. Die vorgeschlagene Lernrate wird deshalb nicht als optimal eingestuft. Ähnlich verhält es sich mit den Graphen mit aktivieren Gradient Accumulation. Hier liegt die empfohlene Lernrate bei circa 0.008, obwohl der Loss in dieser Region stetig steigt. Es wurde daher eine Lernrate von 10^{-6} ausgewählt, da diese in beiden Versionen in einem langen abfallenden Abschnitt liegt und in der Nähe keine großen Loss-Sprünge vorzufinden sind.

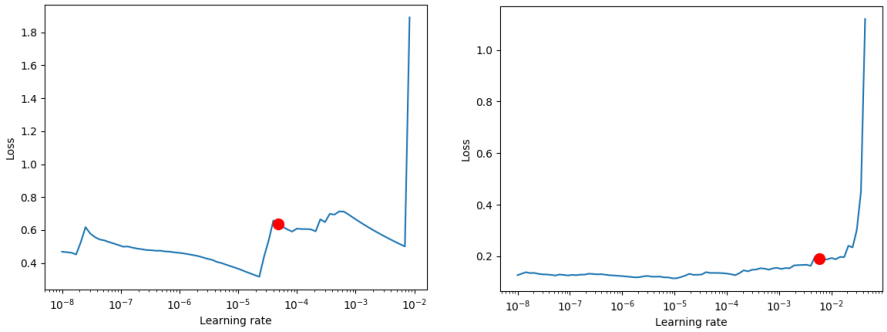

Abbildung 6: Lightning Learning Rate Finder: Rechter Graph ohne Gradient Accumulation, linker Graph mit Gradient Accumulation

5.5 Class Weights

Bei der Bestimmung, ob ein Satz zu der Zusammenfassung gehören soll oder nicht, gibt es zwei Möglichkeiten: Der Satz gehört zur Zusammenfassung (True) oder gehört nicht zur Zusammenfassung (False). Ein Artikel besteht im Durchschnitt aus 32 Sätzen, aber es gibt nur 1-2 Sätze, die zur Zusammenfassung (Ground Truth) gehören sollen. Damit ist das Dataset sehr stark im Ungleichgewicht, wodurch das Modell beim Traininieren viel mehr Sätze sieht, die nicht zur Zusammenfassung gehören als umgekehrt. Um diesem Ungleichgewicht entgegenzuwirken, wurden Klassengewichte eingeführt. Durch eine Analyse des Datasatzes ergab es sich, dass es 15 mal mehr Sätze gibt, die nicht zur Zusammenfassung gehören. Die Gewichtung wurde deshalb mit 1 für die “False“-Klasse und mit 15 für die “True“-Klasse gewählt. Wenn also ein Satz, der zur True-Klasse gehört, falsch prognostiziert wird, wird dies entsprechend dem Klassengewicht berücksichtigt. Der Loss erhöht sich dadurch und Backpropagation fällt dement-

sprechend stärker aus. Wie in Abbildung 7 ersichtlich wird, steigt zwar wie im Vorfeld angenommen der Validierungs-Loss, aber die F1-Scores steigen in allen Kategorien leicht an (circa 0.3-0.5%).

Class Weights	Val. Loss	Train Micro F1	Train Macro F1	Val. Micro F1	Val. Macro F1
On	0.3687	**25.53**	**19.99**	**25.57**	**20.18**
Off	**0.1585**	25.00	19.65	25.00	19.80
(L3-BL)	-	-	-	12.89	12.58

Abbildung 7: Vergleich der Klassen-Gewichte (Settings: GradientAcc=Off, Epoch=1)

5.6 Loss-Funktion

Mit der Einführung der Klassengewichte, ergab sich auch die Fragestellung, welche Loss-Funktion benutzt werden sollte. Da es sich bei der Klassifizierung um ein binäres Problem handelt (entweder True oder False) kann die letzte Linear-Schicht des Modells entweder nur eine oder zwei Wahrscheinlichkeiten ausgeben. Je nach Art des Outputs werden dann zwei verschiedene Loss-Funktionen benutzt. Bei einem Output-Neuron kommt die Binary-Cross-Entropy-Funktion (BCEwithLogits[4]) zum Einsatz. Die Cross-Entropy ist ein Maß für die Differenz zwischen zwei Wahrscheinlichkeitsverteilungen für eine gegebene Zufallsvariable oder eine Menge von Ereignissen. Sie kann daher als eine negative log-Wahrscheinlichkeit für die (Input-)Daten angesehen werden [5]. Da es bei der Problemstellung nur zwei verschiedene Klassen gibt, kann daher die Binary-Cross-Entropy angewandt werden. Die BCEwithLogits-Funktion verwendet, anders als die normale Cross-Entropy, ein Sigmoid Aktivierung und berechnet unter der Berücksichtigung der Klassen-Gewichte den Loss. Der unreduzierte BCELogits-Loss kann daher als folgende Formel beschrieben werden (w = Weight, x=Input, y=Target):

$$BCELogitsLoss_n = -w_n[y_n \cdot log\ \sigma(x_n + (1 - y_n) \cdot log(1 - \sigma(x_n))]$$

Bei zwei Output-Neuronen-Variante wird die Cross-Entropy-Funtion[5] benutzt, welche statt eines Sigmoids eine Log-Softmax Aktivierung benutzt. Anders als der BCELoss kann der Cross-Entropy-Loss für Multi-Class Classification beutzt

[4] `https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html`
[5] `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`

werden. Der unreduzierte Cross-Entropy-Loss kann als folgende Formel mit der Berücksichtigung der Klassen-Gewichte beschrieben werden:

$$CELoss(x, class) = weight[class][-x[class] + log(\Sigma_j exp(x[j]))]$$

Die zwei Output-Neuronen Variante mit der Cross-Entropy-Funktion erzielte im Test bessere Ergebnisse (Abbildung 8). Besonders die Micro-F1 Werte stiegen um circa 1.9% an. Bei den Macro-F1 ließ sich ein Zuwachs von knapp 1% verzeichnen.

Loss-Function	Train Micro F1	Train Macro F1	Val. Micro F1	Val. Macro F1
CrossEntropy	**25.53**	**19.99**	**25.57**	**20.18**
BCEwithLogits	23.68	19.11	23.64	19.20
(L3-BL)	-	-	12.89	12.58

Abbildung 8: Vergleich der Loss-Funktionen (Settings: GradientAcc=Off, Epoch=1)

5.7 Gradient Accumulation

Oftmals hat man das Problem, dass die Batch-Size wegen eines begrenzten GPU-Speichers limitiert wird; dies ist auch hier der Fall. Abhilfe kann hier Gradient Accumulation schaffen. Die Idee ist, dass die große Globale-Batch in mehrere kleine Mini-Batches aufgeteilt wird. Die Mini-Batches werden dann sequentiell über ein bestimmtes Intervall verarbeitet. Dabei werden die einzelnen Gradienten der Mini-Batches aufsummiert und erst dann erfolgt die Backpropagation [6]. Es wurden gute Erebnisse mit einem Kumulierungs-Grad von 4 erzielt. Dies bedeutet, dass vier Mini-Batches berechnet werden, welche aus, in diesem Fall, vier Samples bestehen. Dies entspricht einer globalen Batch-Size von 16. Durch die Anwendung steigt zwar der Trainings- und Validtion Loss, aber dafür steigt der F1-Score um circa 1% (siehe Abbildung 9). Ein schöner Nebeneffekt ist auch, dass sich die gesamte Trainingszeit um 13 Stunden reduziert (von 8.3h auf 7.0h pro Epoche).

5.8 Layer Normalization & Dropout

Der Trainingsaufwand und die Trainingsdauer für große neuronale Netze ist teuer und dauert lange. Um die Trainingszeit zu reduzieren, kann eine Layer Nor-

Gradient Accumulation	Train Loss	Val. Loss	Epoch Loss	Micro F1	Total Train. Time
On	0.1634	0.3463	0.3453	**25.60**	**2d 22h**
Off	**0.0969**	**0.3455**	**0.3332**	24.60	3d 11h
(L3-BL)	-	-	-	12.89	-

Abbildung 9: Vergleich Gradient Accumulation (GradientAcc=Off, Epoch=10)

malization angewendet werden. Eine Mini-Batch besteht aus Tensoren, welche alle die gleichen Anzahl von Features haben. Eine Achse des Mini-Batch-Tensors enspricht der Batch, wobei die anderen Achsen die gewünschten Merkmale beschreiben. Bei einer Layer Normalization wird der Input einer Mini-Batch über die Features normalisiert. Der Mittelwert und die Varianz, welche für die für die Normalisierung verwendet werden, werden aus den summierten Inputs einer Schicht berechnet [1]. Dies ist auch der Unterschied zu anderen Normalisierungsformen wie der Batch-Normalization. In dem ausgeführten Experiment wurde die Layernormalization im Classifier zwischen zwei Linear-Schichten angewandt. Obwohl der Classifier über nur sehr wenige Schichten verfügt, konnte eine Zeitersparnis von 7 Minuten pro Epoche beobachtet werden.

LayerNorm + Droput	Val. Loss	Val Micro F1	Val. Macro F1	Total Train Time
On	**0.3615**	25.48	**20.19**	11h 40h
Off	0.3687	**25.58**	20.18	11h 47h
(L3-BL)	-	12.89	12.58	-

Abbildung 10: Vergleich der Loss-Funktionen (GradientAcc=Off, Epoch=1)

Bei einem Dropout werden während des Trainings einige Elemente des Eingegabetensors zufällig auf Null gesetzt. Durch einen Parameter kann angegeben werden, wie Wahrscheinlich ist es, dass ein Element genullt wird. Mit einem Dropout soll ein Neuron Features erkennen, die im Allgemeinen nützlich sind, was zu besseren Ergebnissen führen kann. [7] Wie in Abbildung 10 ersichtlich ist, führte das Dropout mit einer Wahrscheinlichkeit von 0.3 zu keinem besseren Ergebnis.

5.9 Label-Smoothing

Label Smoothing ist eine Technik, die eine Overconfidence eines Modells vermeidet. Hierbei werden die Target-Tensoren in Soft-Targets umgewandelt, indem

die gewichteten Durchschnittswerte der Hard-Targets uniform über die Labels verteilt werden [11].
Da die Cross-Entropy-Loss Funktion in Pytorch keine Smoothed Labels unterstützt, musste auf die KLDivLoss-Funktion[6] ausgewichen werden. Die Kullback-Leibler Divergence ist der Cross-Entropy ähnlich (siehe Kapitel 5.6), beschreibt jedoch die Differenz zwischen zwei Wahrscheinlichkeitsverteilungen über dieselbe Variable [5]. Die unreduzierte Kullback-Leibler Divergence Loss kann als folgende Formel beschrieben werden (y = Target, x = Input):

$$KLDivLoss_n = y_n \cdot (log\ y_n - x_n)$$

Durch einen Smoothing-Parameter kann die Stärke des Smoothings bestimmt werden. Bei dem ausgeführten Experiment wurde eine Smoothing-Wert von 0.2 benutzt. So wird aus dem Target-Tensor [0, 1, 0] der Soft-Target-Tensor [0.2, 0.8, 0.2]. Das Label Smoothing führte zu einem positven Testergebniss. Der Validation-Loss sank um 12%, während die F1-Scores um circa 1-1,5% in allen Kategorien stiegen (siehe Abbildung 11).

Label Smoothing	Val. Loss	Train Macro F1	Val. Micro F1	Val. Macro F1
On	**0.0304**	**20.98**	**26.08**	**21.17**
Off	0.1585	19.65	25.00	19.80
(L3-BL)	-	-	12.89	12.58

Abbildung 11: Vergleich Label Smoothing (ClassWeights=Off, Loss=KLDivLoss/CrossEntropy, Epoch=1)

In einem weiteren Vergleich wurden zusätzlich noch die Klassen-Gewichte (Kapitel 5.5) hinzugenommen. Dies erhöhte den Validation-Loss deutlich, jedoch wurden noch bessere Werte in den F1-Kategorien erreicht (Abbildung 12). So stieg der Macro F1 Score im Training um circa 1%, während bei der Validierung einen Anstieg um 1.3% zu verzeichnen ist.

[6] `https://pytorch.org/docs/stable/generated/torch.nn.KLDivLoss.html`

Label Smooting	Val. Loss	Train Macro F1	Val. Micro F1	Val. Macro F1
+ Class Weights	6.128	**22.09**	**27.34**	**22.24**
- Class Weights	**0.0304**	20.98	26.08	21.17
(L3-BL)	-	-	12.89	12.58

Abbildung 12: Vergleich Label Smoothing + Class Weights (Loss=KLDivLoss, Epoch=1)

5.10 Vergleich: Sliding Window, Dokumenten-weise und Sample-weise

In diesem Kapitel werden die verschiedene Ansätze der Dataloader aus Kapitel 4 miteinander verglichen. Abbildung 13 (Tabelle 1) zeigt die verschiedenen Ergebnisse und Settings der einzelnen Runs. Alle Runs wurden mit denselben Settings ausgeführt. Wie man erkennen kann, ist der Sample-Wise-Ansatz bezogen auf den Epoch Loss (durchschnitts Loss am Ende einer Trainingsepoche) der niedrigste mit einem Wert von 0.3559. Der Sliding-Window Ansatz liegt in der Mitte (0.5654), dicht gefolgt von dem Doc-Wise-Ansatz mit dem höchsten Wert von 0.6517. Auch gab es deutliche Unterschiede bei der Anzahl an benötigen Steps und der benötigten Trainingszeit für zwei vollständige Epochen. Die Total Steps beinhaltet auch die Schritte für die Validierung. Der schnellste Ansatz mit einer Trainingszeit von knapp 2 Stunden ist der Doc-Weise-Ansatz (1h7m/Epoche), gefolgt von mit Sliding-Window mit circa 4 Stunden (2h2m/Epoche) und zum Schluss der Sample-Wise Ansatz mit einer Trainingszeit von fast zwei Tagen (11h33m/Epoche).

Wie Abbildung 13 (Tabelle 2) zeigt, lohnt sich der Mehraufwand des Sample-Wise Ansatzes, da dieser in allen F1-Kategorien mit Abstand die besten Ergebnisse erzielt. Als schlechtester Ansatz stellt sich der Doc-Wise Ansatz heraus mit F1-Validierungs Werten von 0.0441 (Micro) und 0.0377 (Macro). Im Schnitt ist Sliding-Window fast doppelt so gut als der Doc-Wise Ansatz, erreicht trotzdem nur entäuschende F1 Validierungs Werte von 0.0988 (Micro) und 0.0766 (Macro). Der Sample-Wise ist knapp 2,5 fach besser mit F1 Validierungswerten von 0.2551 (Micro) und 0.2013 (Macro) und liegt damit als einziger deutlich über der Lead3-Baseline mit 0.13 (Micro und Macro).
Da alle Ansätze mit denselben Settings und der gleichen Architektur ausgeführt wurden, kann die unterschiedliche Performance nur durch die grundlegende Unterscheidung eines Samples erklärt werden. Bei dem Doc-Wise Ansatz ist in

Type	Epoch Loss	Val. Loss	Total Steps	Total Train Time
Sample-Wise	**0.3559**	**0.3536**	35.85k	23h 6m
Doc-Wise	0.6517	0.6479	**4.38k**	**2h 13m**
Sliding Window	0.5654	0.5776	9.424k	4h 3m

(a) Tabelle 1: Epoch- und Validation Loss, sowie Trainingszeitraum

Type	Train Micro F1	Train Macro F1	Val. Micro F1	Val. Macro F1
Sample-Wise	**25.61**	**20.08**	**25.51**	**20.13**
Doc-Wise	4.49	3.80	4.41	3.77
Sliding Window	9.89	7.60	9.88	7.66
(L3-BL)	-	-	12.89	12.58

(b) Tabelle 2: Trainings- und Validierungs-F1-Scores

Abbildung 13: Vergleich der verschiedenen Ansätze. (Settings: LabelSmoothing=Off, Loss=CrossEntropy, Epoch=2)

einem Sample das gesamte Dokument enthalten, statt nur eines einzelnen Satzes. Die beschleunigt zwar erheblich die Trainingszeit, hat aber keine positive Einwirkung auf das Endergebnis. Der Kontext für das gesamte Dokument ist zwar vorhanden, aber das Modell kann anscheinend keine Parallelen zu anderen Dokumenten oder Sätzen finden. Es sieht immer nur ein Dokument aus einer bestimmten Kategorie und kann daraus anscheinend keine übergreifende Rückschlüsse ziehen. Bei dem Sliding-Window Ansatz besteht ein Sample immer aus einem Ziel-Satz plus eine Anzahl von Sätzen, die vor oder nach dem Ziel-Satz stehen. Damit erhält man zusätzlichen Kontext, welcher die Satz-Repräsentation ergänzt. Auch kann ein Batch aus Sätzen von verschiedenen Dokumenten bestehen, was beim Doc-Wise Ansatz nicht der Fall war. Dies bringt auch einen deutlichen Gewinn in den F1 Ergebnissen im Vergleich zu dem Doc-Wise Ansatz.

Im Gegensatz dazu besteht ein Sample im Sample-Wise Ansatz aus dem gesamten Artikel. Die Satz- und Dokument-Repräsentationen werden im Forward-Schritt zusammen aus denselben BERT-Encoding berechnet, was bei dem Sliding-Window Ansatz nicht der Fall war. Eine Batch kann hier ebenfalls aus Samples von verschiedenen Dokumenten bestehen. Es wird vermutet, dass bei diesem Ansatz der Kontext mit den besten Informationen vorhanden ist.

6 Visualisierung & Evaluierung

Um einschätzen zu können, wie gut das trainierte Modell, unabhängig von der F1-Metrik, wirklich ist, wurde das Ergebniss visualisiert, sowie eine manuelle Evaluation durchgeführt. Dafür wurde das Modell mit den besten F1-Scores genommen. Dies war der Sample-Weise Ansatz mit aktiviertem Label Smoothing und Class Weights.

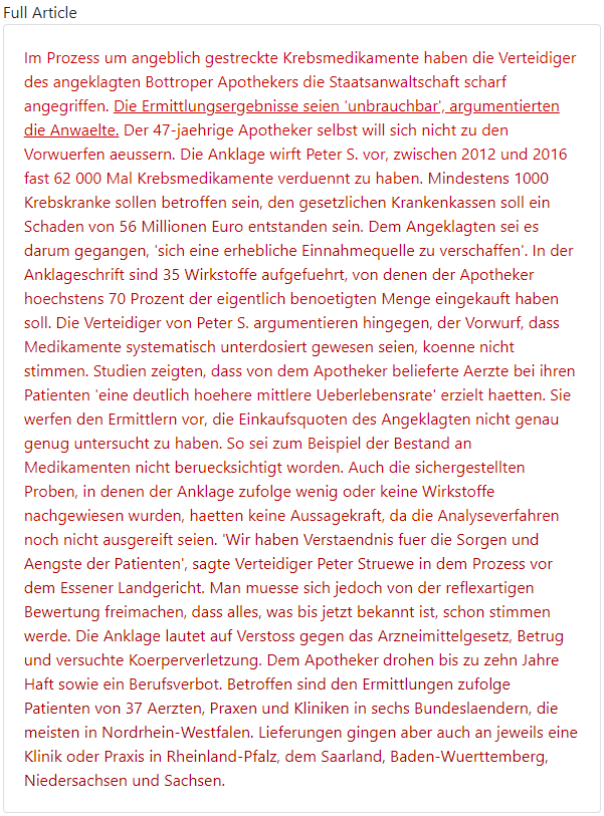

Abbildung 14: Visualisierung eines Artikels anhand der Satz-Wahrscheinlichkeiten

Abbildung 14 zeigt die Visualisierung eines Artikels anhand der Satz - Wahrscheinlichkeiten. Jeder Satz wurde anhand seiner Wahrscheinlichkeit eingefärbt. Leuchtend Rot (RGB 255, 0, 0) steht dabei für eine maximale Wahrscheinlichkeit (p=1.0). Ein dunkles Rot (RGB 127, 0, 0) steht für eine mittlere Sicherheit (p=0.5), während Schwarz (RGB 0, 0, 0) für eine minimale Wahrscheinlichkeit steht (p=0.0). Jeder Satz kann auch die Farbräume dazwischen einnehmen. Wie man anhand der Abbildung erkennen kann, sind die verschiedenen Rot-

Abstufungen nur schwer zu erkennen. Dunkle oder gar Schwarz-Töne sind gar nicht vorhanden. Dies bedeutet, dass sich das Modell im Allgemeinen sicher ist, dass ein Satz zur der Zusammenfassung gehören soll. Die einzelnen Sätze unterscheiden sich bei ihrer Wahrscheinlichkeit um circa 1-5% je nach Dokument.

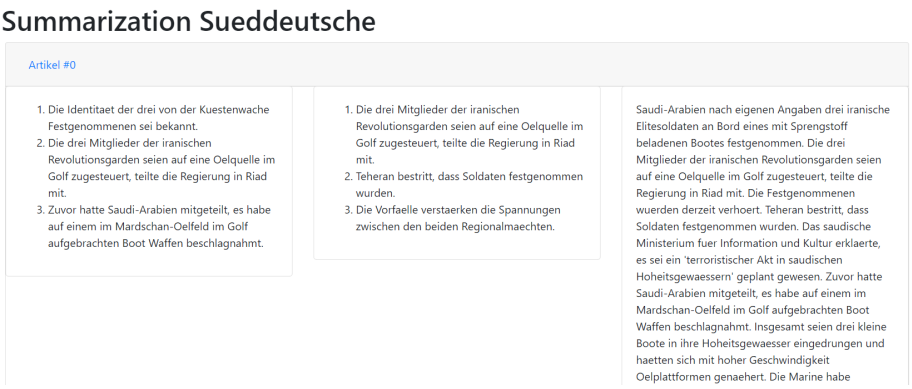

Abbildung 15: Ansicht der Evaluierung: Spalte 1 & 2 enthalten Ground Truth und generierte Zusammenfassung (zufällige Positionierung), Spalte 3 enthält den vollständigen Artikel

Für die Evaluierung generierte das Modell für 15 zufällig ausgewählte Artikel aus dem Validierungsdatensatz Zusammenfassungen. Die Ergebnisse lassen sich durch eine Web-Ansicht betrachten. Wie Abbildung 15 zeigt, wurde für jeden Artikel die generierte und die originale Zusammenfassung, sowie der ganze Artikel in Text-Form, gezeigt. Die Positionierung der Zusammenfassungen waren ebenfalls zufällig. Weitere Informationen wie die Artikel-Überschrift oder den F1-Score standen nicht zur Verfügung. Jeder Satz der Zusammenfassung wurde anhand einer Metrik, bestehend aus gut, mittel und schlecht, von zwei Experten bewertet. Zusätzlich sollte man entscheiden, welcher der beiden Zusammenfassungen in Hinblick auf den Gesamtartikel besser war. Diese Bewertungen wurden anschließend mit der Auflösung verglichen (siehe Abbildung 16).

Tabelle 2 zeigt die zusammengezählten Ergebnisse der Evaluierung beider Teilnehmer. Die ausgewählten Sätze des Modells wurden sehr ausgeglichen bewertet. Es gab fast so viele gute (#16) wie schlechte Sätze (#14). Als mittel-gut (+-) wurden 8 Sätze bewertet. Der Ground Truth wurde im Vergleich etwas besser bewertet. Hier waren 20 Sätze gut, 8 mittel-gut und 10 Sätze schlecht. Interessant ist hierbei die prozentuale Übereinstimmung der Teilnehmer. Bei den guten und mittel-guten Sätzen besteht eine 60% Übereinstimmigkeit. Dies beu-

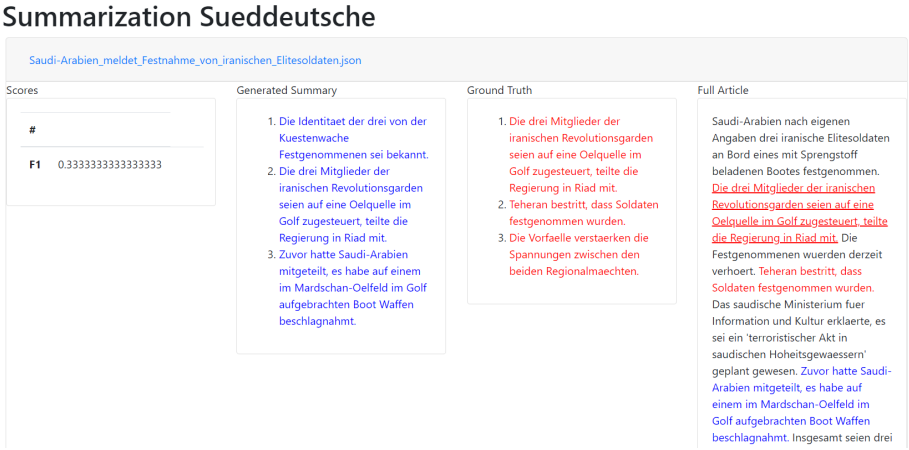

Abbildung 16: Ansicht der Ground Truth: Generierte Sätze (Blau), Ground Truth (Rot), übereinstimmende Sätze (Rot+Unterstrichen)

detet, dass nur knapp über die Hälfte der Sätze gleich bewertet wurden. Noch niedriger ist die Übereinstimmung bei den schlechten Sätzen mit 40 bzw. 42%.

	Generated			GroundTruth		
Rating	+	+-	-	+	+-	-
Total Votes	16	8	14	20	8	10
Votes Match	60%	60%	40%	66%	100%	42%
#Picked	11			17		
Picked Match	60%			83%		

Tabelle 2: Vergleich zwischen generierten und originalen Zusammenfassung. *Total Votes* gibt die Gesamtanzahl aller Stimmen für eine Kategorie an. *Votes Match* gibt die prozentuale Übereinstimmung der Abstimmenden an. *# Picked* gibt an, welche Zusammenfassung bevorzugt wurde. *Picked Match* gibt die prozentuale Übereinstimmung der ausgesuchten Zusammenfassung der beiden Evaluationsteilnehmer an. Rating: + (gut), +- (mittel), - (schlecht)

Etwas deutlicher wurde die Evaluation bei der Frage, welche Zusammenfassung generell besser ist. Hier lagen die originalen Zusammenfassungen mit 17 zu 11 Stimmen klar vorne. Ein Artikel wurde aus dieser Abstimmung herausgenommen, da die Ground Truth Zusammenfassung nur aus einem einzigen Wort bestand, welches jedoch vom Modell ausgewählt wurde. Insgesamt wurden 83% der originalen Zusammenfassung gleich gewählt. Bei der den genierten Zusammenfassungen des Modells sinkt der Wert auf nur noch 60%. Es lässt

sich festhalten, dass das trainierte Modell durchaus einen passablen Eindruck hinterlässt. Zwar sind die originalen Zusammenfassungen im Durchschnitt besser und wurden auch öfter als besser evaluiert, jedoch konnte das Modell bei einigen Artikeln bessere Zusammenfassungen liefern. Dieser Eindruck und die Evaluierungsergebnisse korrelieren durchaus mit den Ergebnissen der F1-Werten aus den Experimenten (Kapitel 5). Zuletzt lässt sich ebenfalls festhalten, dass die persönlichen Präferenzen bei Zusammenfassungen ebenfalls eine Rolle spielen. Sätze, welche als schlecht bewertet wurden, können für anderen gute oder mittel-gute Sätze für Zusammenfassungen eines Artikels sein, dies hat die prozentuale Übereinstimmung gezeigt. Sicherlich spielt der Inhalt und das Thema eines Artikels eine weitere Rolle, da die semantische Interpretation eines Artikels bei jedem anders ausfallen kann.

7 Diskussion & Ausblick

Bei den Tests der verschiedenen Ansätze in Kapitel 5.10 wurden beim besten Modell Validierungs-F1-Werte von 0.2013 (Makro) 0.2551 und (Mikro) gemessen. Die Lead3-Baseline des Datensatzes liegt bei ca. 0.13 (Makro und Mikro), welche das Modell deutlich übersteigen konnte. Dies zeigt, dass der Sample-Wise Ansatz der vielversprechendste ist. In der manuellen Evaluation in Kapitel 6 wurde gezeigt, dass die ausgewählten Zusammenfassungssätze durchaus mit den aus Stichpunkt generierten Ground-Truth Sätzen vergleichbar sind, auch wenn sie qualitativ nicht ganz an diese heran reichen.
Weiterführend kann zum einen der angedachte Abstractor implementiert werden, der mit dem Umschreiben der ausgewählten Sätze die Qualität der generierten Zusammenfassung verbessern soll. Diese reiht die Sätze momentan nur aneinander und beachtet weder Redundanz noch den Textfluss der Zusammenfassung. Dafür könnten die Paper *Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting (2018)* [2] und *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension (2019)* [9] Aufschluss zu der konkreten Umsetzung geben.
Den bereits implementieren Extractor betreffend könnten ohne größeren Aufwand andere vortrainierte BERT Modelle getestet werden, um deren Effekt auf die Trainingsergebnisse zu vergleichen. Der bestehende Extractor sollte außerdem mit anderen Datensätzen trainiert werden, um festzustellen, ob eine größere Anzahl an Samples die Qualität der Zusammenfassung erhöht. Dazu könnten zum Beispiel Datensätze wie *MLSUM* [14] zum Einsatz kommen, welche auch abstraktive Zusammenfassungen unterstützen. Auch offenbarte der verwendete Datensatz der SZ einige Schwächen. Die Artikelsätze wurden im Vorfeld mit

SpaCy [8] getrennt, was nicht immer zu optimalen Ergebnissen führte (siehe Beispiel aus Kapitel 6). Des Weiteren zeigt die Evaluierung, dass die Stichpunkte nicht immer gute Zusammenfassungen des Artikels sein müssen.
Außerdem können technische Verbesserungen implementiert werden, die die teils lange Epochendauer von 11-13 Stunden (siehe Kaptitel 5.10) verkürzen könnten. Dazu gehört der Support von Multi-GPU oder die 16-Bit Half Precision, welche beide von PyTorch Lighting unterstützt werden.

8 Appendix

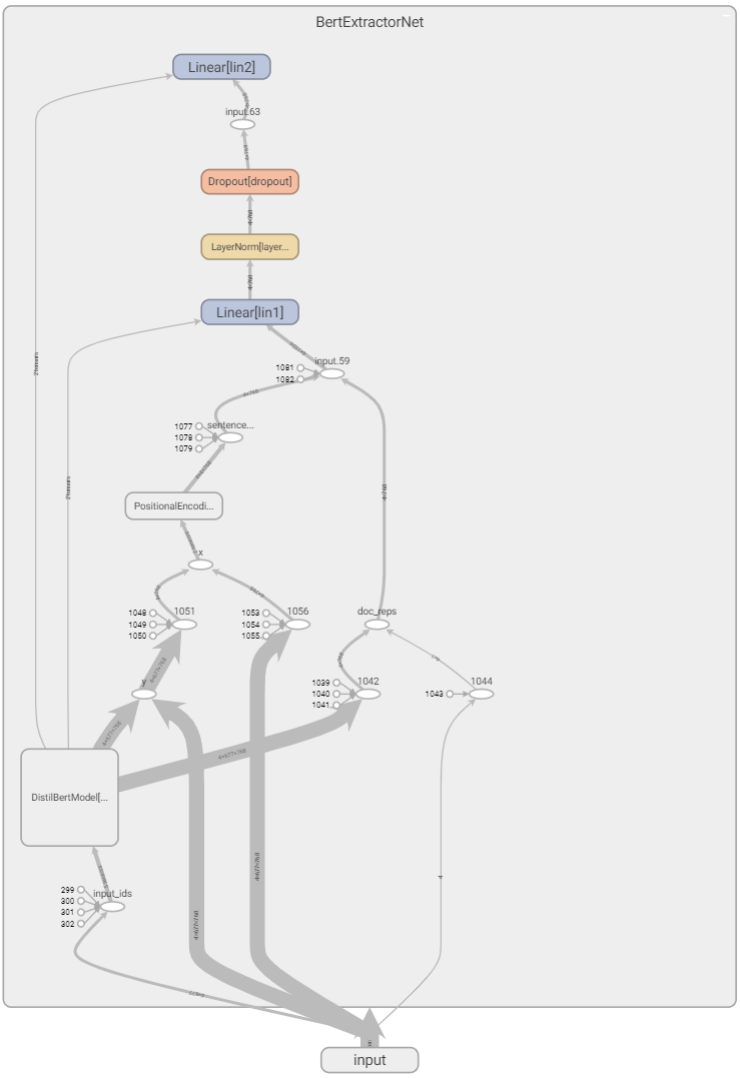

Abbildung 17: Graph des Extractors als Sample-weis Ansatz

Literatur

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[2] Yen-Chun Chen and Mohit Bansal. Fast abstractive summarization with reinforce-selected sentence rewriting, 2018.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[4] WA Falcon. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[5] Daniel Godoy. Understanding binary cross-entropy / log loss: a visual explanation. *towards data sience. Note: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a*, November 2018.

[6] Raz Haleva. What is gradient accumulation in deep learning? *towards data sience. Note: https://towardsdatascience.com/what-is-gradient-accumulation-in-deep-learning-ec034122cfa*, Januar 2020.

[7] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[8] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[9] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019.

[10] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. In *EMNLP/IJCNLP*, 2019.

[11] Rafael Müller, Simon Kornblith, and Geoffrey Hinton. When does label smoothing help?, 2019.

[12] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents, 2016.

[13] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019.

[14] Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, and Jacopo Staiano. Mlsum: The multilingual summarization corpus, 2020.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[16] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

[17] Han Xiao. bert-as-service. `https://github.com/hanxiao/bert-as-service`, 2018.

	Daniel	Christian
Datensatz & Crawling	10%	90%
F1-Metric	100%	0%
Planung Architektur	50%	50%
Implementierung Modell	50%	50%
Experimente	50%	50%
Visualisierung	0%	100%
Evaluierung	50%	50%

	Daniel	Christian
Einleitung	x	x
Verwandte Themen	x	x
Datensatz	x	x
Architektur	x	x
Experimente	x	x
Visualisierung & Eval	x	x
Disskusion und Ausblick	x	x