technische universität
dortmund

Bachelorarbeit

Einsatz einer End-to-End Lösung für die Relevanzbewertung von Fragen im Question-Community-Answering

Maurice Freund
April 2019

Gutachter:
Prof. Dr. Katharina Morik
M. Sc. Sebastian Buschjäger

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (LS 8)
www-ai.cs.uni-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

Der Wachstum des Internets und das Voranschreiten der Digitalisierung in den letzten Jahrzehnten hat zu einer beträchtlichen Ansammlung von Daten und Informationen geführt. Ein Großteil davon liegt in Form von Texten vor, die jeden Tag geschrieben, gesammelt und abgespeichert werden. Die Menge an generierten Daten ist dabei so groß, dass automatische Systeme zur Auswertung nötig sind. Bei der Verarbeitung von natürlicher Sprache ist dabei neben der syntaktischen Analyse auch ein semantisches Verständnis nötig. Da gerade letzteres Computer vor große Probleme stellt widmet sich ein ganzes Forschungsgebiet, das "Natural Language Processing" (NLP), dieser Aufgabe.
Ein bewährter Ansatz im NLP ist das sogenannte maschinelle Lernen (ML). Hierbei lernen Systeme, auch Modelle genannt, mittels einer gegebenen Menge an Beispieldaten verschiedene Aufgaben zu lösen. Beispielsweise können ML-Verfahren eingesetzt werden um Klassifikationsprobleme zu lösen. Hier müssen Objekte einer oder mehreren vorher festgelegten Kategorien zugewiesen werden. Beispielsweise können Nachrichtenartikel in verschiedene Themengebiete oder Kundenrezensionen in positive oder negative Bewertungen eingeteilt werden.
Ein weiterer Bereich in dem NLP und ML zum Einsatz kommen ist das Themengebiet "Information Retrieval" (IR). Hier werden basierend auf Anfragen durch einen User Informationen aus einer Datenquelle zurückgeliefert. Dabei liegen sowohl die Anfrage wie auch die Ergebnisse meist in komplexer Form vor. Ein Beispiel dafür sind Suchmaschinen die basierend auf von Benutzern formulierten Suchanfragen die dazu passenden Webseiten zurückliefern müssen.
Neben dem Auswählen der richtigen Ergebnisse gehört zum IR häufig aus das Sortieren derselben. So liefern Suchmaschinen unter Umständen tausende von Treffern zurück zeigen aber die mit der höchsten Relevanz zuerst an. Auch für dieses Ranking lassen sich ML-Modelle einsetzen mit denen sich das Gebiet "Learning to Rank" befasst.
Ein praktisches Problem bei dem die Analyse natürlicher Sprache und Sortierung von Objekten zusammen kommen ist das Bewerten von Antworten im "Question Communi-

ty Answering" (CQA). Beim CQA stellen Nutzer Fragen zu verschiedenen Themen in Online-Frage-Foren. Diese Fragen können von anderen Usern beantwortet werden. Der Vorteil solcher Foren ist, dass es kaum Beschränkungen dafür gibt wer Antworten verfassen darf. Dadurch werden verschiedene Lösungsansätze und Diskussionen ermöglicht. Allerdings kann auf Grund der freien Natur keine Garantie für die Qualität der Antworten geben werden. Ein Fragesteller findet also unter Umständen eine Fülle verschiedener Antworten vor, die nicht unbedingt alle relevante Informationen für die Beantwortung der Frage liefern. Deshalb ist eine automatische Bewertung der Antworten bezüglich ihrer Relevanz zur gestellten Frage sinnvoll.
Da es sich dabei um ein relevantes Thema handelt befasst sich auch der Task 3 des SemEval 2017[1] mit dem CQA. Speziell geht es in einem Subtask um die Bewertung von Kommentaren, die zur Beantwortung einer Frage formuliert wurden, bezüglich ihrer Relevanz. Ein System soll alle für eine Anfrage verfügbaren Antworten so sortieren, dass Kommentare mit stärkerem Bezug zu Frage weiter vorne stehen. Diese Aufgabe soll die Grundlage dieser Arbeit bilden.

1.1 Motivation und Zielsetzung

ML-Modelle lösen Klassifikationsprobleme indem sie einen Zusammenhang zwischen den Merkmalen von Objekten und den entsprechenden Klassen lernen. Dabei kann für die Generierung dieser Merkmale, auch Features genannt, eines von zwei Verfahren eingesetzt werden. Beim manuellen Feature-Engineering entscheiden menschliche Entwickler welche Merkmale genutzt werden. So kann bei der Betrachtung von Fragen und Antworten im CQA zum Beispiel festgehalten werden ob bestimmte Wörter in den Texten vorkommen oder ob der Autor einer Antwort auch der Fragesteller ist. Solche Features können von ML-Modellen genutzt werden um die Klasse eines Objekts vorherzusagen.
Ein Großteil der Lösungen, die für den SemEval Task 3 eingereicht wurden, setzen auf manuell erstellte Merkmale. Allerdings hat dieses Verfahren einige Nachteile. Das Auswählen und Erstellen manueller Features ist ein aufwendiger Prozess. Zudem ist es möglich, dass die Entwickler Merkmale auswählen, die nicht im Zusammenhang mit der Klasse stehen, also für die Bearbeitung der Aufgabe irrelevant sind. Andersherum können relevante Features übersehen werden. Außerdem wird die Lösung für ein Problem auf die gegebene Aufgabenstellung spezialisiert. Um ein bewährtes Modell für eine neue Aufgabe einzusetzen müssen unter Umstände ganz neue Features implementiert werden.
Um diese Probleme zu lösen können sogenannte "End-to-End"-Modelle eingesetzt werden. Diese erhalten die weitestgehend unverarbeiteten Daten als Eingabe und konstruieren selbstständig relevante Features, die dann für die Vorhersage der Klasse genutzt werden. Somit wird das Feature-Engineering automatisiert. Außerdem erlernen diese Modelle die

[1] http://alt.qcri.org/semeval2017/task3/

Features anhand von Trainingsdaten, für die die richtige Klasse bereits bekannt ist. Es werden also nur Merkmale genutzt die zumindest in den Beispieldaten relevant waren. Außerdem können "End-to-End"-Modelle mit weniger Aufwand für die Lösung anderer Probleme genutzt werden. Dafür müssen sie lediglich mit entsprechend anderen Trainingsdaten trainiert werden.
Aus diesen Vorteilen ergibt sich, dass es interessant ist dein Einsatz von "End-to-End"-Modellen im CQA zu untersuchen. Als Grundlage dient der SemEval 2017 Task 3 Subtask A, die Sortierung von Antworten bezüglich ihrer Relevanz zur Frage. Ein "End-to-End"-Modell soll zur Bearbeitung dieser Aufgabe genutzt werden. Ziel ist es den Einsatz des ML-Modells zu bewerten. Dafür sollen die Ergebnisse mit anderen Lösungen verglichen und Vor- und Nachteile des Modells zusammengestellt werden.

1.2 Aufbau der Arbeit

Zunächst sollen die theoretischen Grundlagen der in der Arbeit eingesetzten Verfahren erläutert werden. Das zweite Kapitel beschäftigt sich anschließend mit der Aufgabenstellung, die in der Arbeit betrachtet werden soll. Zudem soll eine formale Definition des Problems gegeben werden. Darauf folgt eine Beschreibung der Experimentumgebung, die für die Arbeit entwickelt und für die Durchführung genutzt wurde. Anschließend werden Aufbau, Hypothese und Ergebnisse aller durchgeführten Ergebnisse beschrieben.
Zum Schluss folgt eine Evaluation der erzielten Ergebnisse. Es soll vor allem darauf eingegangen werden wie die Ergebnisse im Vergleich mit anderen Verfahren zu interpretieren sind und welche Erkenntnisse über den Einsatz von End-to-End Verfahren getroffen werden können. Zudem sollen weitere mögliche Verbesserungsmöglichkeiten besprochen werden.

1.3 Verwandte Arbeiten

Ziel des Rankings im "Information Retrieval" ist es Dokumente die als Ergebnis einer Anfrage zurückgeliefert wurden in eine sortierte Reihenfolge zu bringen. Zum Lösen dieser Aufgabe können ML-Verfahren eingesetzt werden, die das Sortieren der Dokumente erlernen. Diese Verfahren werden unter dem Begriff "Learning to Rank" [15] zusammengefasst. Hierbei unterscheidet man zwischen verschiedenen Herangehensweisen basierend darauf welche Daten den Modellen zur Verfügung gestellt werden.
Die unter dem Begriff "pairwise" zusammengefassten Verfahren betrachten jeweils Paare von Texten und entscheiden welcher der beiden vor dem anderen einsortiert werden soll ([1], [2], [26]). Diese paarweise Präferenz wird anschließend genutzt um alle Dokumente, die zu einer Anfrage gehören, zu sortieren. Im Gegensatz dazu berücksichtigen "listwise"-Algorithmen alle auf eine Anfrage zurückgelieferten Dokumente und lernen diese zu sortieren ([3], [24]).

Die dritte Möglichkeit, die sogenannten “pointwise”-Verfahren, erhalten als Eingabe lediglich ein Dokument und bestimmen dessen Relevanz zur dazugehörigen Anfrage. Eine Herangehensweise dafür ist es die Dokumente in Relevanzklassen zu unterteilen und anschließend ML-Modelle darauf zu trainieren die Klasse eines Dokuments vorherzusagen [14]. Anschließend findet die Sortierung anhand der vorhergesagten Klassen statt. Es wird also effektiv eines Klassifikationsproblem statt eines Rankingproblems gelöst.
Für die Klassifikationen von Texten können verschiedene ML-Methoden eingesetzt werden. In [11] werden unter anderem Klassifizierer auf Basis von “naive Bayes”, “nearest Neighbour” und Entscheidungsbäumen für die Kategorisierung von Nachrichtentexten eingesetzt. Für die Erkennung von Spamnachrichten in E-Mails wird in [8] ebenfalls das “naive Bayes” Verfahren genutzt. Ein Ansatz unter Verwendung einer SVM wird von Joachims in [12] getestet um Texte in “Bag-of-Words”-Form zu klassifizieren.
Eine Repräsentationsmethode für Texte alternativ zum “Bag-of-Words” sind die sogenannten Word-Embeddings. Ein Beispiel hierfür ist das “Word2Vec”-Verfahren aus [16]. Hier werden einzelne Wörter in Vektoren einer festgelegten Länge überführt. Mit diesen Vektoren können Beziehungen zwischen Begriffen durch die Entfernung im Vektorraum und durch einfache algebraische Operationen modelliert werden.
Auch neuronale Netze eignen sich für die Klassifikation von Texten. Im Gegensatz zu Verfahren die manuelles Erstellen von Features erfordern ([21], [23]) besitzen neuronale Netze die Fähigkeit relevante Features im Input selber zu erlernen. So wird in [30] ein “Convolutional Neural Network” eingesetzt um Texte zu klassifizieren, die als Folge einzelner Zeichen anstatt von Wörtern repräsentiert werden. Das Modell erkennt selbständig Muster in den Zeichenfolgen, die Auskunft über die Klasse geben.
Die Besonderheit bei der Bewertung von Antworten bezüglich der Relevanz zur gestellten Frage ist, dass zwei Texte analysiert werden müssen statt nur einer. Deshalb muss ein Modell für diese Aufgabe in der Lage sein zwei Texte und ihre Beziehungen zueinander zu untersuchen. Um dies zu tun wurde in [28] eine siamesische Netzwerkarchitektur bestehend aus CNNs kombiniert mit einem Attention-Mechanismus eingesetzt um Textpaare und deren Einfluss aufeinander für verschiedene NLP-Aufgaben zu modellieren.
Der Einsatz von CNNs hat sich in den letzten Jahren immer öfter bewährt um NLP-Probleme zu lösen. So setzten He, Gimpel und Lin [7] ein siamesisches neuronales Netz auf Basis von zwei CNNs ein um Textpaare auf Ähnlichkeit zu überpüfen. Severyn und Moschitti [22] nutzten eine ähnliche Architektur um die Relevanz von Sätzen in Bezug auf eine Anfrage zu ermitteln.
Um Textpaare auf Ähnlichkeit zu untersuchen haben Mueller und Thyagarajan [17] Long Short-Term Memory Networks (LSTM) eingesetzt um Texte in Form von Word-Embeddings in eine Vektorrepräsentation zu überführen. Dabei wurden die LSTMs so trainiert, dass Texte mit hoher Ähnlichkeit in Vektoren transformiert werden, die im Vektorraum nahe beieinander liegen. Anschließend wurde lediglich mit Hilfe einer einfachen Vektordistanz-

metrik wie der Manhattan-Distanz die Ähnlichkeit der beiden Texte bestimmt.
Als Grundlage für die Aufgabenstellung wird der SemEval 2017 Task 3 genutzt. Da dieser Wettbewerb bereits abgeschlossen ist, existieren bereits eine Reihe von Lösungen, die sich mit der Aufgabe beschäftigt haben. Die meisten Methoden nutzten Verfahren bei denen es nötig war manuelles Feature-Engineering zu betreiben. So hat das Team Beihang-MSRA in [5] traditionelle NLP-Features wie zum Beispiel die Longest-Common-Subsequence in Zusammenhang mit Features auf Basis von neuronalen Netzen eingesetzt. Diese nutzte ein "Gradient Boosted Regression Tree" um eine Vorhersage zu treffen.
In [6] wurde eine SVM genutzt, die mittels einer Reihe manuell erstellter Features die Textpaare klassifiziert. Diese Features modellieren unter anderem die Ähnlichkeit zwischen den beiden Texten sowie Übereinstimmungen in der Satzstruktur. Mit dieser Lösung erreichte das Team den ersten Platz für die Aufgabe der Relevanzbewertung von Kommentaren.
Das Team EICA [27] modellierte die Ähnlichkeit von Frage und Kommentar in dem es zum Beispiel die Distanz zwischen den Vektorrepräsentationen berechnete. Außerdem nutzte das Team zusätzliche Features zu den Metadaten, zum Beispiel ob eine Kommentar vom Fragesteller selbst verfasst wurde. Als Klassifizierer kamen eine SVM, sowie ein logistisches Regressions-Modell zum Einsatz.
Im Gegensatz zu den anderen beiden Lösungen kam das Team FuRongWang [29] mit weniger Feature-Engineering aus in dem es ein siamesisches CNN-Modell einsetzte welches sich End-to-End trainieren lies. Da diese Lösung der geplanten Vorgehensweise dieser Arbeit sehr ähnlich ist wird dieses Modell später als Vergleichsmöglichkeit für die erzielten Ergebnisse genutzt.

Kapitel 2

Hintergrund und Theorie

Bei der Aufgabenstellung der Arbeit handelt es sich um ein maschinelles Lernproblem, welches mit Hilfe eines neuronalen Netzes gelöst werden soll. Aus diesem Grund soll im Folgenden die Theorie hinter diesen Begriffen zusammengestellt werden. Zunächst soll die Theorie und Funktionsweise von neuronalen Netzen erklärt werden. Der anschließende Abschnitt befasst sich mit den Metriken, mit denen die Modelle evaluiert werden. Zuletzt wird das Thema Word Embeddings aufgegriffen, die dafür genutzt werden Texte in eine für das Modell verwendbare Repräsentation zu überführen.

2.1 Neuronale Netze

Wie in der Einleitung beschrieben soll ein Hauptaspekt dieser Arbeit sein die gegebene Aufgabenstellung mit Hilfe eines End-to-End-Verfahrens zu lösen. Das heißt, dass das eingesetzte Modell in der Lage ist anhand der Originaldaten, in diesem Fall also anhand der Frage- und Antworttexte, die gesuchte Größe, hier die Relevanz, vorherzusagen. Hierfür eignen sich insbesondere neuronale Netze, welche in der Lage sind selbständig anhand der Inputdatenmenge Features zu erkennen, die Aufschluss über die Zielgröße geben. Im Folgenden sollen deshalb die einzelnen Bestandteile und Mechanismen von neuronalen Netzen beschrieben werden.

2.1.1 Neuronen

Wie der Name suggeriert sind künstliche neuronale Netze Zusammenschlüsse von Neuronen, auch Einheiten oder Units genannt. Diese Neuronen erhalten einen Input, den sie mittels einfacher Rechenoperationen in eine Ausgabe umwandeln . Auf Grund des biologischen Vorbilds der Neuronen wird diese Ausgabe auch Aktivierung genannt. Die Arbeitsweise der Neuronen und ihre Verknüpfung untereinander unterscheidet die verschiedenen Architekturen für neuronale Netze.
In der einfachste Form berechnen Neuronen eine Linearkombination ihrer Eingabewerte

[25]. Sei x die Eingabe eines Neurons in Form eines Vektors mit i Elementen. Die Ausgabe $\hat{y}$ lässt sich dann wie folgt berechnen:

$$z = w^\top x + b \tag{2.1}$$

$$\hat{y} = f(z) = f(w^\top x + b) \tag{2.2}$$

Hierbei wird der Vektor $w = (w_1, ..., w_i)$ als Gewichtsvektor und b als Bias bezeichnet. Diese Elemente werden während des Trainings gelernt. Die Funktion $f(z)$ wird als Aktivierungsfunktion bezeichnet. Es existieren eine Reihe von häufig eingesetzten Aktivierungsfunktionen. In dieser Arbeit werden zwei verschiedene Arten genutzt. Zum einen die Sigmoid-Funktion, die wie folgt definiert ist [25]

$$f(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

Zum anderen die rektifizierte lineare Funktion [10]:

$$f(z) = max\{0, z\} \tag{2.4}$$

Nichtlineare Aktivierungsfunktionen erlauben es neuronalen Netzen nichtlineare Zusammenhänge zwischen dem Input und der Zielgröße zu erlernen. Zudem erlauben sie es den Wertebereich der Ausgaben von Neuronen zu beschränken. So kann der Output einer Sigmoid-Funktion als Wahrscheinlichkeit interpretiert werden, da die Ausgabewerte immer zwischen 0 und 1 liegen.
In künstlichen neuronalen Netzen werden die Neuronen in Schichten, auch Layer genannt, angeordnet. Dabei besteht ein Netz immer aus einem Input- und einem Output-Layer, zwischen denen beliebig viele Schichten, die so genannten Hidden-Layer, eingefügt werden können. Diese können je nach Netzart unterschiedlich konstruiert sein. Die genaue Arbeitsweise und die Unterschiede zwischen den einzelnen Schichtarten soll im folgenden besprochen werden.

2.1.2 Fully Connected Layer

Der "Fully Connected Layer" (FC-Layer) erhält seinen Namen aus dem Umstand, dass seine Neuronen mit jedem Neuron aus der vorhergehenden Schicht über Gewichtskanten verbunden sind. Der Input x jedes Neurons wird damit von dem Vektor aller Aktivierungen der Vorgänger gebildet.
Die Funktionsweise eines FC-Layers lässt sich aus der Kombination von mehreren Neuronen unter Verwendung der zuvor aufgestellten Gleichungen herleiten. Seien zunächst l_i und l_j der i-te und j-te Layer eines neuronalen Netzes. Dabei gelte, dass l_j der Nachfolger von l_i sei, also gelte $i = j - 1$. Des weiteren bezeichnen n_i und n_j die Anzahl der Neuronen in

den Layern l_i und l_j. Da n_j Neuronen aus l_j mit n_i Einheiten aus l_i verbunden sind lassen sich die Gewichte zwischen l_i und l_j als eine Matrix definieren [25]:

$$W^{[j]} = \begin{pmatrix} w_{11}^{[j]} & \cdots & w_{1n_i}^{[j]} \\ \vdots & \ddots & \vdots \\ w_{n_j1}^{[j]} & \cdots & w_{n_jn_i}^{[j]} \end{pmatrix} \tag{2.5}$$

Dabei bezeichne $w_{kl}^{[j]}$ das Gewicht, welches das k-te Neuron aus l_j mit dem l-ten Neuron aus l_i verbindet. Mit $Y^{[j-1]}$ als Ausgabe von l_i (Input von l_j) lässt sich nun die Ausgabe $Y^{[j]}$ von l_j berechnen als:

$$Z^{[j]} = W^{[j]}Y^{[j-1]} + b^{[j]} \tag{2.6}$$

$$Y^{[j]} = g(Z^{[j]}) \tag{2.7}$$

Dabei ist $b^{[j]}$ der Vektor aller Biaswerte der Neuronen in l_i. Die Aktivierungsfunktion g wird in diesem Fall elementweise auf jeden Wert in $Z^{[j]}$ angewendet.

2.1.3 Convolution Layer

Eine besondere Art der neuronalen Netze sind die "Convolutional Neural Networks" (CNN). Diese Netze eignen sich für die Verarbeitung von Daten, die in rasterähnlicher Form auftreten [10]. Dies können zum Beispiel die Pixelraster von Bilddaten sein oder wie in dieser Arbeit der Zusammenschluss von Wortvektoren zu Matrizen.
Der Vorteil von CNNs ist, dass Neuronen in Convolution Layern nur mit einem Teil der Vorgänger-Neuronen verbunden sind. Außerdem teilen sich einige Neuronen ihre Gewichte untereinander. Diese Eigenschaften führen zu einer besseren Effizienz bezüglich Laufzeit und Speicherbedarf bei der Verarbeitung von großen Eingaben, wie zum Beispiel Bilddaten [10]. In dieser Arbeit werden CNNs für die Verarbeitung von Texten verwendet. Aus diesem Grund soll im folgenden die Funktionsweise für speziell diesen Anwendungsfall erklärt werden.
Im Folgenden sei angenommen, dass das i-te Wort in einem Text durch einen k-dimensionalen Vektor $x_i \in \mathbb{R}^k$ repräsentiert wird (siehe 2.4). Jeder Text $x_{1:n}$ wird dann als Konkatenation seiner Wortvektoren beschrieben, also

$$x_{1:n} = x_1 \oplus x_2 \oplus ... \oplus x_n = \begin{bmatrix} x_{1,1} & x_{2,1} & \dots & x_{i,1} \\ \vdots & \dots & \ddots & \vdots \\ x_{1,k} & x_{2,k} & \dots & x_{i,k} \end{bmatrix}. \tag{2.8}$$

Der Convolution-Layer nutzt einen Filter $w \in \mathbb{R}^{h \times k}$ welcher auf jeweils h aufeinander folgenden Wörtern eingesetzt wird. Für jede Position i in $x_{1:n}$ berechnet sich das Ergebnis der Faltung des Inputs mit dem Filter w wie folgt [13]:

$$c_i = f(w \cdot x_{i:i+h} + b). \tag{2.9}$$

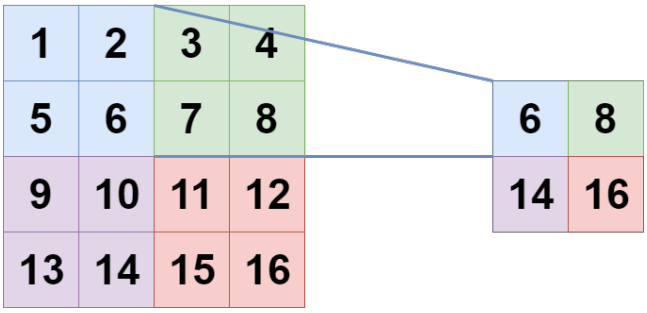

Abbildung 2.1: Beispiel einer Max-Pooling-Operation auf einem 4×4-Input mit einer Kernelgröße und Schrittweite von zwei. Der Input wird mit diesen Parametern halbiert.

Dabei bezeichne $x_{i:i+h}$ die Konkatenation der Wörter $x_i, ..., x_{i+h}$. Alle Werte, die aus der der Anwendung des Filters an jeder Textposition entstehen, zusammen ergeben die sogenannte "Feature-Map" des Filters

$$c = [c_1, c_2, ..., c_n]. \tag{2.10}$$

Dabei wird der Textinput an den Rändern mit Nullvektoren aufgefüllt, damit die Ausgabe c die selbe Länge besitzt wie $x_{1:n}$. Dieser Vorgang wird auch als Padding bezeichnet [25]. Anstelle eines einzelnen Filters werden in CNNs typischerweise eine Reihe von verschiedenen Filtern $W = (w_1...w_m)$ eingesetzt. Die Anzahl und Größe dieser Filter kann als Hyperparameter des Netzes festgelegt werden. Jedes Neuron in einem Convolution-Layer übernimmt dabei die Berechnung der Faltung eines Filters an einer bestimmten Position. Die Ausgabe des Layers ist also eine $n \times m$ Matrix mit einer Reihe für jeden Filter und einer Spalte für jede Filterposition.

2.1.4 Pooling

Das Pooling wird genutzt um die Ausgabe eines Layers zu verkleinern. Dabei werden naheliegende Werte in der Ausgabe durch eine repräsentante Größe ersetzt [10]. So wird zum Beispiel beim "Max-Pooling" von nebeneinander liegenden Ausgaben nur die Größte unter ihnen behalten. Die Pooling-Operation besitzt zwei Parameter. Zum einen die Größe des Kernels der über den Input bewegt wird um alle überdeckten Werte zusammenzufassen. Zum anderen die Weite mit der der Kernel bei jedem Schritt bewegt wird. Ein Beispiel für eine Anwendung des Poolings wird in Abbildung 2.1 gegeben. Zu den Vorteilen des Poolings gehört zum einen die Reduktion der Datenmenge während diese das Netz durchläuft und zum anderen die Invarianz gegenüber Verschiebung der Muster im Input [31].

2.1.5 Long Short Term Memory Networks

"Long Short Term Memory Networks" (LSTM) sind eine Art der rekurrenten neuronalen Netze (RNN). Ihr Haupteinsatzgebiet ist die Analyse von Daten, die in sequenzieller Form

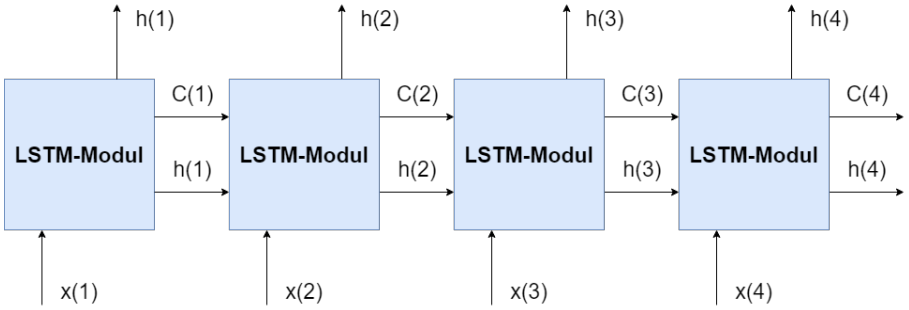

Abbildung 2.2: Schematischer Aufbau einer Sequenz von vier LSTM-Modulen.

vorkommen. Ein Beispiel hierfür sind Texte, die als Sequenz von Wörtern behandelt werden können. Das Besondere an der Verarbeitung dieser Daten ist, dass Informationen über vorherige Zeitschritte relevant für die Bearbeitung des aktuellen Sequenzsegments sind. Aus diesem Grund werden LSTMs durch eine Kette von Modulen gebildet, wie in Abbildung 2.2 gezeigt. Das t-te Module eines LSTMs erhält das t-te Element der Inputsequenz x, im Folgenden $x(t)$ genannt, als Eingabe. Weitere Eingaben des Moduls sind die Ausgabe $h(t-1)$ sowie der sogenannte Cell-State $C(t-1)$ des Vorgängers. Der Cell-State ist ein Vektor, der von jedem Modul angepasst werden kann indem Informationen hinzugefügt oder entfernt werden. Somit dient der Cell-State als Gedächtnis mit denen Informationen über frühere Elemente der Eingabesequenz abgerufen werden können. Zudem berechnet jedes Modul einen Ausgabewert h der an die Nachfolger weiter gegeben wird.

Um den Cell-State anzupassen und die Ausgabe zu berechnen nutzt jedes Modul eine Kombination verschiedener Elemente, die sogenannten Gates. Jedes dieser Gates besitzt einen eigenen Gewichtsvektor, der das Verhalten bestimmt und dessen Werte während des Trainings optimiert werden. Um die Funktionsweise eines LSTMs zu beschreiben soll im Folgenden stellvertretend das t-te Modul eines LSTMs betrachtet werden. Wie zuvor beschrieben erhält das Modul $x(t)$, $h(t-1)$ und $C(t-1)$ als Eingabe. Folgende Formeln beschreiben wie die Ausgaben $C(t)$ und $h(t)$ eines Moduls aus den Eingaben berechnet werden [20]:

$$f(t) := \sigma(w_f(x(t) + h(t-1))), \tag{2.11}$$

$$ff(t) := \sigma(w_{ff}(x(t) + h(t-1))), \tag{2.12}$$

$$C^*(t) := \tau(w_C(x(t) + h(t-1))), \tag{2.13}$$

$$i(t) := ff(t) \cdot C^*(t), \tag{2.14}$$

$$C(t) := f(t) \cdot C(t-1) + i(t), \tag{2.15}$$

$$fff(t) := \sigma(w_{fff}(x(t) + h(t-1))), \tag{2.16}$$

$$h(t) := fff(t) \cdot \tau(C(t)), \tag{2.17}$$

Hier bezeichne σ die Sigmoid-Funktion aus 2.1.1 und τ den Tangens Hyperbolicus (Tanh) definiert als [20]:

$$\tau(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.18}$$

Diese beiden Funktionen werden genutzt um Werte zwischen 0 und 1 (Sigmoid) beziehungsweise -1 und 1 (Tanh) zu erzeugen. Jedes LSTM-Modul besitzt drei Gates. Das Forget-Gate (Gleichung 2.11) bestimmt welche Informationen aus dem Cell-State entfernt werden. Das Input-Gate erstellt einen Kandidaten C^*, der zum Cell-State hinzugefügt werden soll (Gleichungen 2.12, 2.13 und 2.14). Der neue Cell-State $C(t)$ entsteht indem der vorherige mit dem Ergebnis des Forget-Gates multipliziert wird und die Ausgabe des Input-Gates anschließend addiert wird (Gleichung 2.15). Zum Schluss wandelt das Output-Gate den neuen Cell-State unter Berücksichtigung der anderen Eingaben in die Ausgabe des Moduls um (Gleichungen 2.16 und 2.17). Die Ausgabe h hat die Form (m, p), wobei m die Länge der Eingabesequenz und p ein wählbarer Hyperparameter ist.
Die Gates erlauben es LSTMs Informationen über längere Zeit zu behalten. Dadurch können Verbindungen von weiter auseinander liegenden Zeitschritten erkannt werden [10]. Dies führt zu einer größeren Performance im Vergleich zu traditionellen RNNs.

2.1.6 Dropout

Beim Training von neuronalen Netzen kann es dazu kommen, dass sich das Modell zu stark an die Trainingsdaten anpasst. Das bedeutet, dass das Netz zwar gute Ergebnisse auf den bekannten Daten erzielt, aber die Performance auf neuen Daten nicht halten kann. Dieses Verhalten wird als Overfitting bezeichnet. Die Technik Dropout [9] bietet eine Möglichkeit Overfitting zu vermeiden. Der Hintergrundgedanke ist, dass komplexe Co-Adaptionen von Neuronen in den Hidden-Layern zu einer starken Anpassung an die Trainingsdaten führt, während sich die Ergebnisse auf ungesehenen Daten verschlechtern. Aus diesem Grund wird beim Dropout für jedes Trainingsbeispiel jedes einzelne Neuron der Hidden-Layer mit einer Wahrscheinlichkeit p ignoriert. Der Wert von p kann als Hyperparameter des Netzes gewählt werden. Es ist auch möglich unterschiedliche Wahrscheinlichkeit für verschiedene Hidden-Layer zu nutzen.
Mit diesem Verfahren wird die Co-Adaption von Neuronen erschwert, da nicht garantiert ist, dass jedes Neuron bei jedem Trainingsschritt zur Verfügung stehen. Ein alternative Sichtweise ist, dass während des Trainings eine Reihe verschiedener Netze trainiert wird, die unterschiedliche Teilmengen der Hidden-Neuronen nutzen, sich aber die Gewichte für die jeweils verfügbaren Neuronen teilen. Nachdem ein Netz mit Dropout trainiert wurde werden für Vorhersagen auf ungesehenen Daten immer alle Neuronen benutzt.

2.2 Training von neuronalen Netzen

Damit ein neuronales Netz lernt eine bestimmte Aufgabe zu lösen müssen seine Parameter, also Gewichte und Biaswerte der Neuronen, so angepasst werden, dass für eine entsprechende Eingabe die korrekte Ausgabe erzeugt wird. Dazu wird beim Training eine Funktion genutzt, die den Fehler des Netzes beschreibt. Diese wird auch als Lossfunktion L bezeichnet. Andere Namen für die Lossfunktion sind zum Beispiel Kosten- oder Zielfunktion. Das Netz erhält Daten x als Eingabe, für die die korrekt Ausgabe y bekannt ist und trifft eine eigene Vorhersage $f(x;\theta)$. Dabei bezeichne θ den Vektor der alle Gewichte und Biaswerte des neuronalen Netzes umfasst. Mit der Lossfunktion kann anschließend der Fehler $L(f(x;\theta),y)$ bestimmt werden, also wie stark sich $f(x;\theta)$ von y unterscheidet. Ziel des Trainings ist diesen Fehler zu minimieren und damit das Modell zu verbessern.
Je nach Aufgabenstellung können verschiedene Funktionen als Lossfunktionen eingesetzt werden. Eine für Klassifikationsprobleme geeignete Funktion ist die Kreuzentropie. Für ein binäres Klassifikationsproblem mit $\hat{y} = f(x;\theta)$ als geschätzte und y als tatsächliche Klasse lautet die Definition der Kreuzentropie [25]:

$$L(\hat{y}, y) = -(y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})). \tag{2.19}$$

Bei einem Klassifikationsproblem mit zwei Klassen gilt $y \in \{0, 1\}$. Die Ausgabe des Modells $\hat{y}$ liegt zwischen 0 und 1. Je näher der Wert von $\hat{y}$ liegt, desto kleiner ist der Wert der Kreuzentropie.

2.2.1 Stochastic Gradient Descent

Ein häufig eingesetzter Algorithmus für die Optimierung der Parameter eines Netzes ist der sogenannte "Stochastic Gradient Descent" (SGD). Die Idee hinter dem Gradientenabstieg ist, dass der Gradient angibt wie sich eine kleine Änderung der Funktionsparameter auf den Wert einer Funktion auswirkt. Somit kann die Funktion minimiert werden indem man die Parameter schrittweise so anpasst, dass der Funktionswert kleiner wird.
Die Kostenfunktion $L(f(x;\theta),y)$ beim Trainieren von neuronalen Netzen ist eine Funktion in Abhängigkeit der Netzparameter θ. Somit kann mittels Gradientenabstieg L durch anpassen der Werte von θ minimiert werden. Dies geschieht mit Hilfe der folgenden Regel für das Update der Gewichte [20] [1]

$$\theta_{t+1} = \theta_t - \epsilon \nabla L. \tag{2.20}$$

[1] Die hier gewählte Notation unterscheidet sich teilweise zu der in [20] um die Einheitlichkeit zu vorherigen Kapiteln zu bewahren.

Dabei wird ϵ als Lernrate bezeichnet. Diese ist ein wählbarer Parameter, der bestimmt wie stark die Anpassung der Gewichte ausfällt. Für ein einzelnes Gewicht $w_t \in \theta_t$ ergibt sich das Gewichtsupdate aus:

$$w_{t+1} = w_t - \epsilon \frac{\partial L}{\partial w_t}. \tag{2.21}$$

Um die Laufzeit des Trainings zu beschleunigen wird beim SGD der Gradient der Zielfunktion geschätzt, statt ihn mit allen Trainingsbeispielen zu berechnen. Zu Beginn des Trainings werden die Gewichte θ_0 zunächst mit zufälligen Werten initialisiert. Außerdem steht eine Menge von Trainingsbeispielen X mit dazugehörigen Zielgrößen Y zur Verfügung. Bei jedem Durchlauf t des Algorithmus wird ein sogenannter Batch von m Beispielen $x^{(1)}, ..., x^{(m)}$ gezogen und für jedes Beispiel $x^{(i)}$ die geschätzte Ausgabe des Modells $f(x^{(i)}; \theta)$ berechnet. Die Größe der einzelnen Batches ist als Hyperparameter wählbar. Mit dieser wird der Gradient der Lossfunktion bezüglich der Gewichte wie folgt geschätzt [10]:

$$\hat{g} = \frac{1}{m} \sum_i \nabla_\theta L(f(x^{(i)}; \theta_t), y^{(i)}). \tag{2.22}$$

Die Schätzung des Gradienten unter Verwendung von Batches beschleunigt den Optimierungsprozess, da Gewichtsanpassungen nach jeweils m Trainingsbeispielen durchgeführt werden, statt nach dem gesamten Trainingsdatensatz. Die Gewichte des Models werden mit Hilfe des geschätzten Gradienten angepasst:

$$\theta_{t+1} = \theta_t - \epsilon \hat{g}. \tag{2.23}$$

2.2.2 L2-Regularisierung

Neben Dropout bietet die L2-Regularisierung eine weitere Möglichkeit das Overfitting von neuronalen Netzen abzuschwächen. Dies geschieht in dem die Modellparameter möglichst klein gehalten werden um die Komplexität des Netzes zu reduzieren. Dadurch fällt es dem Netz schwerer zu spezifische Zusammenhänge in den Trainingsdaten zu erlernen.
Bei der Regularisierung wird ein zusätzlicher Term zur Zielfunktion hinzugefügt. Im Fall der L2-Regularisierung wird folgender Term gewählt [20]:

$$L^{improved} = L^{original} + \frac{\lambda}{m} ||w||_2^2. \tag{2.24}$$

Dabei ist λ der Regularisierungsparameter und passt an wie stark sich die Regularisierung auf das Training auswirkt. Die Updateregel aus Gleichung 2.21 ändert sich dadurch folgendermaßen:

$$w_{t+1} = w_t - \epsilon (\frac{\partial L^{original}}{\partial w_t} + \frac{\lambda}{n} \cdot w_t). \tag{2.25}$$

Der konstante Faktor $\frac{\lambda}{n}$ führt dazu, dass die Gewichte bei den Updates tendenziell kleiner werden, solange ein Wachstum des Gewichts nicht ein wesentlich besseres Ergebnis liefert.

2.3 Evaluierungsmetriken

Um die Qualität eines neuronalen Netzes zu messen und es mit anderen Modellen zu vergleichen werden eine Reihe verschiedener Metriken eingesetzt. Diese untersuchen verschiedene Aspekte und beschreiben die Güte der Vorhersagen von Modellen aus verschiedenen Perspektiven.

2.3.1 Klassifikationsmetriken

Für Klassifikationsprobleme bei denen entschieden werden muss ob ein Objekt zu einer bestimmten Klasse gehört oder nicht, können die Ausgaben eines Netzes in eine von vier Kategorien eingeteilt werden [25]. Gehört ein Objekt zu der betrachteten Klasse und wird vom Modell auch entsprechend klassifiziert gehört es zur Menge der "True Positives". Andersherum sind "True Negatives" die Objekte, für die richtigerweise entschieden wurde, dass sie nicht zu der Klasse gehören. Die vom Modell falsch klassifizierten Objekte werden entweder als "False Positives" oder "False Negatives" bezeichnet. Erstere wurden vom Modell der Klasse zugewiesen obwohl sie ihr nicht angehören. Letztere sind eigentlich Teil der Klasse wurden aber nicht entsprechend klassifiziert. Folgende Tabelle verdeutlicht den Zusammenhang noch einmal:

	Vorhersage NEIN	Vorhersage JA
Wahrer Wert NEIN (tno)	*TRUE NEGATIVES (tn)*	*FALSE POSITIVES (fp)*
Wahrer Wert JA (ty)	*FALSE NEGATIVES (fn)*	*TRUE POSITIVES (tp)*

Basierend auf diesen Kategorien können nun verschiedene Metriken gebildet werden. Die Genauigkeit, oder Accuracy, ist eine häufig eingesetzte Metrik für Klassifikationsprobleme. Ihre Definition lautet wie folgt:

$$acc = \frac{tp + tn}{N}. \tag{2.26}$$

Dabei ist N die Anzahl aller verfügbaren Objekte. Die Genauigkeit beschreibt also den Anteil der korrekt klassifizierten Objekte relativ zu der Menge aller Objekte.
Der Einsatz der Genauigkeit ist problematisch auf unbalancierte Datensätze, also solche bei denen ein Ergebnis wesentlich häufiger vorkommt als andere. Gehören zum Beispiel 90% der Objekte nicht der betrachteten Klasse an würde ein Modell welches ausschließlich negativ klassifiziert eine Genauigkeit von 90% erreichen. In solchen Fällen können die Metriken "Precision" und "Recall" eingesetzt werden, die wie folgt definiert sind:

$$Precision = \frac{tp}{tp + fp} \tag{2.27}$$

$$Recall = \frac{tp}{ty}. \tag{2.28}$$

Die "Precision" beschreibt das Verhältnis der korrekt positiv klassifizierten Objekte in Bezug zu allen positiv klassifizierten Objekte. Der "Recall"-Wert bestimmt wie viele von den tatsächlich positiven Objekten korrekt klassifiziert wurden. Diese Werte setzen also ein größeres Gewicht darauf, dass möglichst viele positive Testfälle korrekt klassifiziert werden.
Um die beiden Metriken in eine zu kombinieren bildet der "F1-Score" das harmonische Mittel aus beiden:

$$F1 = 2\frac{Precision \cdot Recall}{Precision + Recall} \tag{2.29}$$

2.3.2 Ranking Metrics

Die zuvor beschriebenen Metriken lassen sich einsetzen um Klassifikationsmodelle zu bewerten. Da die Aufgabenstellung dieser Arbeit als Klassifikationsproblem interpretiert und gelöst werden können diese Werte die Qualität der eigenen Modelle beschreiben. Allerdings ist die Hauptaufgabe der Problemstellung das Sortieren von Antworten nach ihrer Relevanz zur gestellten Frage. Aus diesem Grund müssen noch weitere Metriken eingesetzt werden um die Sortierungen der Antworten zu bewerten.
Zu diesem Zweck kommen zwei weitere Metriken zum Einsatz, die alle von einem Modell produzierten Sortierungen evaluieren. Dabei wird davon ausgegangen, dass Elemente dieser Sortierungen, hier Dokumente genannt, entweder relevant oder nicht relevant bezüglich einer Anfrage sein können.
Die erste eingesetzte Metrik ist der "Mean Reciprocal Rank" (MMR). Für jede Anfrage q sei $r(q)$ der Rang des ersten relevanten Dokuments in der sortierten Liste. Der MMR-Wert ist dann definiert als [15]:

$$MRR = \frac{1}{r(q)}. \tag{2.30}$$

Dieser Wert ist also nur abhängig vom ersten relevanten Dokument auf das man in der Liste stößt. Die Idee ist hier, dass es zum Beispiel bei Suchanfragen wichtig ist, dass ein User möglichst schnell ein relevantes Suchergebnis findet.
Die zweite Metrik, die die Listen für alle Anfragen miteinbezieht lautet "Mean Average Precision" (MAP) [15]. Zunächst wird für jede Position k der sortierten Liste für eine Anfrage q die "Precision" P an der Stelle k berechnet:

$$P@k(q) = \frac{rel(k)}{k}. \tag{2.31}$$

Dabei ist $rel(k)$ die Anzahl der relevanten Dokumente, die sich unter den ersten k Dokumenten in der sortierten Liste befinden. In einem weiteren Schritt wird die "Average Precision" (AP) definiert als Durchschnitt aller "Precision"-Werte an den Rängen an denen sich relevante Dokumente befinden:

$$AP(q) = \frac{\sum_{k_1}^{m} P@k(q) \cdot l_k}{no\_rel(q)}. \tag{2.32}$$

Hier ist $no\_rel(q)$ die Anzahl aller relevanten Dokumente für die Anfrage q und l_k ist gleich 1 wenn das Dokument an der Stelle k in der sortierten Liste relevant ist und ansonsten gleich 0. Der Durchschnitt der AP-Werte für alle Anfragen bildet dann die MAP.

2.4 Word2Vec

Damit ein neuronales Netz Daten verarbeiten kann müssen diese in numerischer Form vorliegen. Das bedeutet, dass natürlich-sprachliche Texte zunächst in eine geeignete Repräsentation überführt werden müssen. Für diesen Schritt gibt es einige verschiedene Ansätze. So wird beim "One-Hot-Encoding" zum Beispiel jedes Wort in einem Text als Vektor der Länge V dargestellt, wobei V die Größe des Vokabulars ist. Jeder Begriff in dem Vokabular erhält einen Index. Der "One-Hot"-Vektor eines Wortes besitzt an dem entsprechenden Index den Wert 1 an allen anderen Stellen den Wert 0. Ein ganzer Text wird dann durch die Konkatenation der einzelnen Wortvektoren zu einer Matrix repräsentiert.
"One-Hot"-Vektoren bieten eine einfache Möglichkeit Texte zu codieren, besitzen allerdings einige Nachteile. So wächst jeder einzelne Wortvektor mit dem Vokabular, was zu einer großen Datenmenge führen kann. Außerdem ist es anhand von One-Hot-Vektoren nicht möglich semantische oder syntaktische Beziehungen zwischen einzelnen Wörtern zu erkennen. Eine andere Möglichkeit Wörter zu repräsentieren, welche in diesen Punkten Verbesserung bringt, sind sogenannte "Word-Embeddings" oder hier speziell das "Word2Vec"-Verfahren [16].
"Word2Vec"-Modelle lernen Vektorrepräsentationen anhand von Textdatensätzen. Diese Vektoren besitzen dabei eine Länge N, die zuvor als Parameter gewählt werden kann und werden so gelernt, dass Wörter mit ähnlicher Bedeutung im Vektorraum näher beieinander liegen [19]. Außerdem lassen sich auch Beziehungen zwischen Wörtern durch einfache algebraische Berechnungen ermitteln.
Die Idee hinter "Word2Vec" kommt aus dem Gedanken, dass ähnliche Wörter in Texten in vergleichbaren Kontexten vorkommen, also von den selben Wörtern umgeben sind. Deshalb gibt es zwei verschiedene Methoden für das Erlernen der Vektoren. Beim "Continuous Bag-Of-Words" (CBOW) wird versucht das zentrale Wort anhand der Kontextwörter vorherzusagen. Andersherum versuchen "skip-gram"-Modelle anhand eines gegebenen Wortes die umliegenden Wörter zu bestimmen.
Beide Verfahren benutzen neuronale Netze mit einem Hidden-Layer um ihre Aufgabe zu erfüllen. Die Neuronenzahl dieser Schicht entspricht der gewünschten Länge der resultierenden Wortvektoren. Die Gewichte des Hidden-Layers werden nach dem Training genutzt um die Wortvektoren zu bestimmen. Wörter, die in ähnlichen Kontexten auftreten, erzeugen bei der Eingabe in ein "skip-gram"-Modell einen vergleichbaren Output und sind das Ergebnis von ähnlichen Inputs eines CBOW-Modells. Aus diesem Grund ähneln sich auch die trainierten Gewichte aus dem Hidden-Layer und damit die resultierenden Wortvekto-

ren.

Im Gegensatz zu dem “One-Hot-Encoding”-Verfahren lässt sich die Größe von Vektoren die aus “Word2Vec”-Modellen entstehen unabhängig vom Vokabular festlegen. Außerdem stehen den Modellen, die “Word2Vec”-Vektoren als Input nutzen, weitere Informationen über Wortbeziehungen zur Verfügung. Allerdings erhöht das Trainieren der CBOW- oder “skip-gram”-Modelle den Rechenbedarf.

Kapitel 3

Aufgabenstellung

In diesem Kapitel soll die Aufgabenstellung beschrieben werden. Zunächst wird der SemEval Task vorgestellt, sowie dessen Vorgaben besprochen. Im Anschluss soll das Lernproblem für die neuronalen Netze formal definiert werden.

3.1 SemEval 2017 Task 3

Bei SemEval handelt es sich um eine Reihe von Workshops, die sich mit der Evaluierung von Semantik-Analyse-Systemen beschäftigt. Dazu werden jährlich eine Reihe von Tasks zu verschiedensten Problemstellungen in der Semantik-Analyse ausgeschrieben. Forscher in diesem Gebiet können diese Tasks nutzen um ihre eigenen Methoden zu evaluieren und mit anderen Vorgehensweisen zu vergleichen. Im Jahr 2017 befasste sich der Task 3 des SemEval mit dem Thema "Community Question Answering"[1]. Der Task unterteilte sich in verschiedene Subtasks, die unterschiedliche Aspekte des Themas betrachteten. Hier soll es speziell um Subtask A gehen, das Sortieren von Kommentaren bezüglich der Relevanz zu der gestellten Frage. Daten die zur Lösung der Aufgabe genutzt werden können, sowie Evaluationsmetrik und Aufgabenstellung wurden vom Task vorgegeben.
Die Daten die für den Task zur Verfügung gestellt wurden enthielten Fragen und Kommentare aus dem Forum "QatarLiving"[2]. Der Datensatz enthält für jede Frage die bis zu zehn zuerst verfassten Antworten. Ein Teil der Daten beinhaltet zudem für jedes Frage-Kommentar-Paar eine Bewertung der Relevanz. Diese Bewertung findet in drei Stufen statt. Kommentare können entweder als "Good", "Potentially Useful" oder "Bad" eingestuft werden. Diese Label sind zuvor von menschlichen Bewertern vergeben worden. Obwohl der Trainingsdatensatz die Stufen "Potentially Useful" und "Bad" enthält werden diese im Testdatensatz und bei der Evaluierung als eine Klasse behandelt. Aus diesem Grund soll auch in dieser Arbeit nur zwischen relevanten ("Good") und irrelevanten Kommentaren

[1] http://alt.qcri.org/semeval2017/task3/
[2] https://www.qatarliving.com/forum

Datensatz	Kommentare insgesamt	"Good"	"Potentially Useful"	"Bad"
dev	2440	818	413	1209
train	40288	17361	5519	17408
test	2930	1523	0	1407

Tabelle 3.1: Größe und Klassenverhältnis der Datensätze. Für die Experimente werden Kommentare mit den Labeln "Potentially Useful" und "Bad" zu einer Klasse zusammengefasst.

("Potentially Useful" und "Bad") unterschieden werden. Für die Experimente die Datensätze genutzt, die in Tabelle 3.1 aufgelistet sind.
Dabei ist der "train"-Datensatz eine Zusammensetzung verschiedener überschneidungsfreier Datensätze die in den Jahren 2015 und 2016 für die Bearbeitung des SemEval Task bereit gestellt wurden. Der "dev"-Datensatz enthält nur einen kleinen Teil der "train"-Daten und wird für Experimente genutzt bei denen es darum geht verschiedene Parameterkonfigurationen zu testen. Hierfür müssen Modelle mit verschiedenen Konfigurationen trainiert werden. Ein kleiner Datensatz verringert die Trainingszeit und damit die Zeit, die für das Experiment nötig ist und reicht gleichzeitig aus um die verschiedenen Konfigurationen miteinander zu vergleichen. Die Daten aus dem "test"-Datensatz werden genutzt um das Modell zu evaluieren. Die Aufteilung des "dev"- und "test"-Datensatzes sind von den SemEval Veranstaltern vorgegeben. Die "train"-Daten sind der Zusammenschluss aller zur Verfügung gestellten Daten, die eine Relevanzbewertung erhalten haben und nicht im "test"-Datensatz vorkommen.

3.2 Problemdefinition

Die Aufgabe, die vom SemEval Task gestellt wird ist das Sortieren aller verfügbaren Antworten bezüglich ihrer Relevanz zu der gestellten Frage. Ein Beispiel für Frage und Kommentare aus dem Datensatz lautet wie folgt:

Frage:
"Is there any place i can find scented massage oils in qatar?"
Kommentar 1:
Text: "Yes. It is right behind Kahrama in the National area."
Relevanz: "Good"
Kommentar 2:
Text: "Whats the name of the shop?" (Antwort des Fragestellers)
Relevanz: "Bad"

Kommentar 1 enthält relevante Informationen für die Beantwortung der Frage. Kommentar 2, in dem der Fragesteller weitere Informationen nachfragt, ist zwar sinnvoll bietet aber selber keine neuen relevanten Erkenntnisse. Aus diesem Grund sollte Kommentar 1 vor Kommentar 2 einsortiert werden.
Für jede Frage Q mit Antwortkommentaren $(c_1, ..., c_n)$ mit $n \leq 10$ soll eine Reihenfolge für die Kommentare gefunden werden, sodass die relevanten Antworten vorne stehen. Die Aufgabe ist also als Ranking-Problem gestellt. In der Praxis existieren nur zwei Klassen für die Kommentare und aus den Trainingsdaten ist nicht ersichtlich welcher von zwei Kommentaren aus der selben Klasse relevanter ist. Aus diesem Grund kann keine eindeutig optimale Reihenfolge erstellt werden, die das Modell während des Trainings als Lernvorgabe nutzt. Deshalb wird die Problemstellung in dieser Arbeit als Klassifikationsproblem gelöst. Das Problem wird damit wie folgt definiert: Für jede Frage $q_i \in Q$ und jedem Kommentar $c_j \in C_i$ wird ein Frage-Kommentar-Paar (q_i, c_j) erstellt. Dabei ist Q die Menge aller Fragen und C_i die Menge aller Kommentare, die als Antwort auf die Frage q_i verfasst wurden. Zu jedem Paar (q_i, c_j) existiert ein Label $y_{ij} \in \{0, 1\}$. Dieses Label ist gleich 1, wenn der Kommentar c_j als relevant für die Beantwortung der Frage q_i eingestuft wurde, also der Klasse "Good" angehört. Für Kommentare mit Einstufung "Potentially Useful" oder "Bad" ist das Label gleich 0. Die Anzahl der vorhandenen Beispiele und die Klassenaufteilung ist der Tabelle 3.1 zu entnehmen. Ziel der Experimente ist die Entwicklung eines Modells, welches für ein gegebenes Frage-Kommentar-Paar das korrekte Label vorhersagen kann.
Für die Bewertung des Modells und dem Vergleich mit anderen Lösungen soll die Evaluationsmetrik des SemEval Tasks, also der MAP-Score, genutzt werden um die Ergebnisse zu evaluieren. Da die MAP aber nur die Reihenfolge der Kommentare und nicht die Relevanzklassifikation selber bewerten kann müssen die Antworten auf ein Frage zunächst sortiert werden. Dazu wird das Modell so konstruiert, dass es für jedes Frage-Kommentar-Paar einen einzelnen Wert zwischen 0 und 1 ausgibt. Dieser wird als Wahrscheinlichkeit dafür interpretiert, dass der Kommentar relevant für die Beantwortung der Frage ist. Bei der Bewertung werden alle Kommentare zu einer Frage nach eben dieser Wahrscheinlichkeit absteigend sortiert und der MAP-Score als Metrik auf diesen Sortierungen berechnet.

Kapitel 4

Implementierung

Nachdem die Aufgabenstellung formal definiert wurde soll im Folgenden das eigene Vorgehen bei der Lösung des Problems vorgestellt werden. Vor allem soll beschrieben werden welche Einschränkungen bei der Wahl möglicher Lösungsmethoden vorliegen.
Die Arbeit soll sich mit der Einsetzbarkeit eines End-to-End-Modells für die Relevanzbewertung auseinander setzen. Dementsprechend beschränkt sich die Modellauswahl auf neuronale Netze welche sich besonders für ein solches Vorgehen eignen. Ziel ist es ein Modell zu entwickeln welches ohne manuelles Feature-Engineering auskommt und somit auch auf andere NLP-Probleme übertragbar ist. Deshalb gilt als weitere Beschränkung, dass das Modell für die Bestimmung des Relevanzlabels y_{ij} nur die Texte des Paares (q_i, c_j) zur Verfügung gestellt bekommt. Metadaten oder Features, die in anderen NLP-Kontexten unter Umständen nicht zur Verfügung stehen, werden nicht betrachtet.

4.1 Versuchsumgebung

Die Entwicklung des End-to-End-Modells soll in einer Reihe von Experimenten geschehen. Ausgehend von einem Basismodell, welches grundlegende Ideen aus verschiedenen vorherigen Arbeiten zusammenfasst, sollen nach und nach die Auswirkungen einzelner Anpassungen untersucht werden,
Zu diesem Zweck wurde eine Versuchspipeline entwickelt, die die Konstruktion, das Training und die Evaluation der einzelnen Modelle übernimmt. Als Programmiersprache für diese Pipeline wird Python genutzt. Die neuronalen Netze werden in Keras mit Tensorflow-Backend implementiert. Im folgenden sollen die einzelnen Bestandteile der Pipeline beschrieben werden.

4.1.1 Konstruktion der Modelle

Um verschiedene Ansätze zu testen werden zunächst die verschiedenen Modellkonzepte implementiert. Jedes dieser Konzepte unterscheidet sich von den anderen in der grundle-

Parametername	Wert
model_name	Modelltyp
cnn_no_filter	Anzahl der Filter für die CNNs
cnn_kernel_size	Filtergröße für die CNNs
lstm_units	Anzahl der Ausgabe-Units in den LSTM-Layern
sl_no_hidden_layers	Anzahl der Hidden-Layer im zweiten Level
sl_no_units:	Anzahl der Neuronen in den Hidden-Layern im zweiten Level
l2_regularization	Wert des Lambda-Parameters für die L2-Regularisierung
fl_dropout_rate	Dropout-Wahrscheinlichkeit für Level 1
sl_dropout_rate	Dropout-Wahrscheinlichkeit für Level 2

Tabelle 4.1: Hyperparameter der Experimentmodelle

genden Funktionsweise. So nutzt das Basismodell zum Beispiel CNNs während ein anderes Modell LSTMs einsetzt. Allen Modellen gemein ist, dass sie in zwei Teile, im folgenden Level genannt, zerteilt werden können. Das erste Level besteht aus zwei neuronalen Netzen, die jeweils den Frage- oder Kommentartext als Input erhalten und in einen einzelnen Feature-Vektor überführen. Das zweite Level besteht aus einem Fully-Connected-Network, welches den Feature-Vektor des ersten Levels nutzt um die Relevanz des Kommentars zu bestimmen. Beide Level werden zusammen End-to-End trainiert.
Die Hyperparameter, über die die einzelnen Modelle verfügen, sind in Tabelle 4.1 aufgelistet. Zu jedem Experiment gehören eine oder mehr Parameterkonfigurationen. Für jede dieser Konfigurationen wird das entsprechende Modell konstruiert, trainiert und getestet. Parameter die von einem bestimmten Modell nicht genutzt werden, werden dabei ignoriert. So braucht ein Modell ohne CNNs zum Beispiel keine CNN-Parameter.

4.1.2 Modelltraining

Für das Training der Netze werden je nach Experiment der “dev”-Datensatz oder die “train”-Daten genutzt. Der “dev”-Datensatz kommt dann zu Einsatz wenn viele Modelle in kurzer Zeit trainiert werden sollen, da die kleinere Datenmenge das Training beschleunigt. In allen anderen Fällen werden die “train”-Daten genutzt. In beiden Fällen werden 80% aller Beispiele des ausgewählten Datensatzes als so genannte Trainingsdaten eingesetzt. Die restlichen 20% dienen als Validierungsdaten. Die Trainingsdaten werden genutzt um das Modell bezüglich der Lossfunktion zu optimieren.
Am Ende jeder Epoche und nach dem das Training abgeschlossen wird die Qualität des Modells getestet indem alle Kommentare in den Validierungsdaten durch das Modell klassifiziert werden und die Genauigkeit und der durchschnittliche Loss auf den Vorhersagen berechnet wird. Als Lossfunktion wird die Kreuzentropie genutzt. Dabei werden die Para-

meter des Netzes nicht verändert weshalb sich das Modell nicht an die Validierungsdaten anpasst.
Um die Ergebnisse eines Experiments nachvollziehen zu können werden folgende Informationen gespeichert:

- Parameterkonfiguration des Modells
- Graph der Modellstruktur
- Erzielte Werte für Genauigkeit und Loss am Ende jeder Epoche für die Trainings- und Validierungsdaten
- Genauigkeit und Loss des vollständig trainierten Modells auf dem Validierungsdaten

Zudem wird auch das trainierte Modell mit den optimierten Gewichten gespeichert.

4.1.3 Modellevaluation

Nachdem alle Modelle trainiert wurden findet die Evaluation eines Experiments statt. Jedes Modell wird genutzt um die Relevanz von Frage-Kommentar-Paaren aus dem ungesehenen "test"-Datensatz vorherzusagen. Dabei erhält jeder Kommentar einen Konfidenzscore, der als Wahrscheinlichkeit interpretiert wird, dass der Kommentar relevant bezüglich der Frage ist. Anschließend werden die Kommentare bezüglich ihrer Relevanzscores sortiert. Auf diesen Sortierungen werden mittels eines Skripts, welches von den SemEval Veranstaltern zur Verfügung gestellt wurde, die Werte der Metriken MAP und MRR berechnet. Diese beiden Scores dienen zum Evaluieren von Rankings und geben Auskunft darüber wie gut sich die Relevevanzklassenvorhersagen der Modelle für das Sortieren der Kommentare nutzen lassen.
Da die Modelle aber in der Praxis ein Klassifikationsproblem lösen wie in 3.2 besprochen, werden weitere Metriken berechnet. Dabei handelt es sich um die Genauigkeit, die Precision, den Recall und den F1-Score. Mit diesen Werten wird die Fähigkeit der Modelle evaluiert, einen Kommentar korrekt als relevant oder irrelevant zu klassifizieren. Obwohl zwar, wie im SemEval Task auch, lediglich der MAP-Score ausschlaggebend ist um zu bestimmen welches Verfahren die besten Ergebnisse erzielt, ist es trotzdem sinnvoll die klassischen Klassifikationsmetriken zu berechnen. Mit ihnen kann das Verhalten einzelner Netze genauer untersucht werden. Zum Beispiel lässt sich mit Precision und Recall untersuchen wie gut ein Modell relevante Kommentare korrekt klassifiziert.

Kapitel 5

Experimente

Um den Einsatz eines End-to-End-Modells für die Relevanzbewertung von Frage und Kommentar Paaren zu untersuchen soll in einer Reihe von Experimenten ein solches Modell eingesetzt und schrittweise angepasst werden. Ziel ist es die Auswirkung der einzelnen Anpassungen getrennt zu betrachten.
Zunächst soll der allgemeine Experimentablauf beschrieben werden. Dafür werden die Vorverarbeitung der Texte, Metriken für die Bewertung der Ergebnisse und Parameter die für alle Modelle gelten vorgestellt. Anschließend wird jedes Modell und dessen Eigenheiten aufgezählt. Zudem werden für alle Modelle auch die Ergebnisse des Trainings und der Evaluation genannt.

5.1 Versuchsaufbau und Vorverarbeitung

Um die Modelle miteinander vergleichen zu können sollen sie unter möglichst gleichen Bedingungen trainiert und getestet werden. Solange nicht anders beschrieben nutzen die Experimente den “train”-Datensatz für das Training, der in Abschnitt 3.1 bereits beschrieben wurde. Einige Experimente bei denen eine kurze Trainingszeit wichtiger ist als die absolute Performance nutzen hingegen den “dev”-Datensatz. In beiden Fällen werden 80% des ursprünglichen Datensatzes genutzt um die Modelle zu optimieren. Die restlichen 20% dienen als Validierungsdaten, werden also genutzt um die Performance der Netze auf ungesehenen Daten zu bewerten. Alle Modelle nutzen den “test”-Datensatz für die Berechnung der Metriken mit denen die finale Bewertung stattfindet. Ein Aufbau aller Datensätze ist in Tabelle 5.1 zu finden. Alle Datensätze sind paarweise disjunkt.

Die Modelle werden über 20 Epochen mit einer Batch-Größe von 32 trainiert. Da es sich um ein binäres Klassifikationsproblem handelt gibt geben die Modelle für jedes Frage-Kommentar-Paar einen einzelnen Wert im Bereich zwischen 0 und 1 aus. Dieser wird als Wahrscheinlichkeit interpretiert, dass der Kommentar relevant für die Beantwortung der Frage ist. Bei einem Wert größer als 0,5 wird der Kommentar als relevant klassifiziert. Für

Datensatz	Kommentare insgesamt	Relevant	Irrelevant
train (Training)	32261	14464	17767
train (Validierung)	8057	2897	5160
dev (Training)	1952	660	1292
dev (Validierung)	488	158	330
test	2930	1523	1407

Tabelle 5.1: Aufbau der Experimentdaten

die Optimierung wird der Adagrad-Algorithmus mit der Kreuzentropie als Lossfunktion genutzt.
Während des Trainings werden die Modelle mit Hilfe der Genauigkeit und der Lossfunktion evaluiert. Für die Evaluation werden die Metriken Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), Precision (P), Recall (R), F1-Score (F1) und Genauigkeit (Acc) eingesetzt. Da der SemEval Task die verschiedenen Lösungen ausschließlich nach dem MAP-Score sortiert wird auch bei den Experimentergebnissen dieser Wert als Hauptkriterium genutzt. Die anderen Metriken liefern zusätzliche Informationen mit denen die Modelle tiefergehend analysiert werden können.

5.1.1 Datenvorverarbeitung

Da das Ziel der Arbeit ist ein Modell zu entwickeln, welches sich auch auf andere NLP-Probleme einsetzen lässt beschränkt sich die Vorverarbeitung der Daten auf wenige Schritte, die bei der Verarbeitung von Texten üblich sind. Zunächst werden alle Zeichen entfernt, bei denen es sich nicht um Buchstaben oder Zahlen handelt. Anschließend wird jeder Text in seine einzelnen Wörter zerlegt und alle Großbuchstaben durch Kleinbuchstaben ersetzt. Anschließend werden die Wörter in Word2Vec-Vektoren mit einer Länge von 300 überführt. Dazu wird ein vortrainiertes Modell genutzt[1]. Zwar ist dies ein allgemeines Modell und nicht speziell an die hier verwendeten Daten angepasst, dafür führt die größere Datenmenge von etwa 100 Milliarden Wörtern mit denen das Modell trainiert wurde zu einer Verbesserung der einzelnen Vektoren.
Die Eingaben eines neuronalen Netzes müssen alle die selbe Form haben. Das heißt, da der Input hier in Form von zu Matrizen konkatenierten Vektoren gegeben ist, müssen diese Matrizen alle die selben Dimension haben. Eine Dimension ist durch die Länge der Wortvektoren gegeben und für alle Texte gleich. Die andere wird durch die Textlänge definiert,

[1] https://code.google.com/archive/p/word2vec/

welche unterschiedlich ausfallen kann. Aus diesem Grund werden alle Texte auf eine fest gewählte Länge von 200 umgerechnet. Dazu werden an Texte mit weniger als 200 Wörtern Nullvektoren der Länge 300 angehängt. Bei längeren Texten wird der überstehende Anteil abgeschnitten. So ist garantiert, dass alle Inputs die selbe Form besitzen.

5.2 Experimentdurchführungen

Die folgenden Abschnitte werden nacheinander alle Modelle vorstellen. Begonnen wird bei dem sogenannten Basismodell. Dieses ist durch verschiedene Arbeiten inspiriert und versucht die Gemeinsamkeiten dieser Arbeiten zu implementieren. Das Basismodell wird genutzt um mit einer Gridsearch die beste Parameterkonfiguration für die folgenden Experimente zu erarbeiten. In diesen Experimenten wird das Basismodell schrittweise angepasst und erweitert und die daraus entstandenen Ergebnisse mit dem Grundmodell verglichen.
Zum Schluss soll noch ein Modell, welches als Lösung für den SemEval Task eingereicht wurde, nachimplementiert werden. So sollen zum einen die Ergebnisse der Lösung nachvollzogen werden und ein Vergleich für das aus den Experimenten hervorgegangene Modell geschaffen werden.

5.2.1 Basismodell und Gridsearch

Das Basismodell kombiniert die gemeinsamen Ideen mehrerer Arbeiten ([22], [7], [28], [32]). Den Arbeiten gemein ist, dass sie ein Paar von CNNs einsetzen um die Frage-Kommentar-Paare in Feature-Vektoren umwandeln, die dann von Klassifizierern genutzt werden um eine Vorhersage über die Relevanz des Kommentares zu treffen. Nach diesem Prinzip wird auch das Basismodell konstruiert.
Dieses Netz soll die Grundlage für die Experimente schaffen. Dafür wird das Modell im ersten Experiment genutzt um mit Hilfe einer Gridsearch die besten Parameter zu bestimmen. Dabei wird für jeden anpassbaren Parameter eine Liste von Werten aufgestellt. Für jede mögliche Kombination von Werten aus diesen Listen wird ein Modell konstruiert, trainiert und evaluiert. Die Parameter, die die besten Ergebnisse erzielen werden für die anderen Experimente genutzt. Da für dieses Verfahren viele Modelle trainiert werden müssen wird jedes Modell nur mit dem “dev”-Datensatz trainiert. Dies reduziert die Zeit, die für das Training nötig ist. Für die Evaluation wird der selbe Evaluationsdatensatz genutzt, der auch bei den anderen Experimenten zum Einsatz kommt.
Die Hypothese für dieses erste Experiment ist, dass das Modell zwar in der Lage ist den Zusammenhang zwischen Frage-Kommentar-Paaren zu lernen aber auf Grund des simplen Aufbaus und der geringeren Trainingsdatenmenge schlechtere Ergebnisse erzielt als die anderen Lösungen des SemEval Tasks.

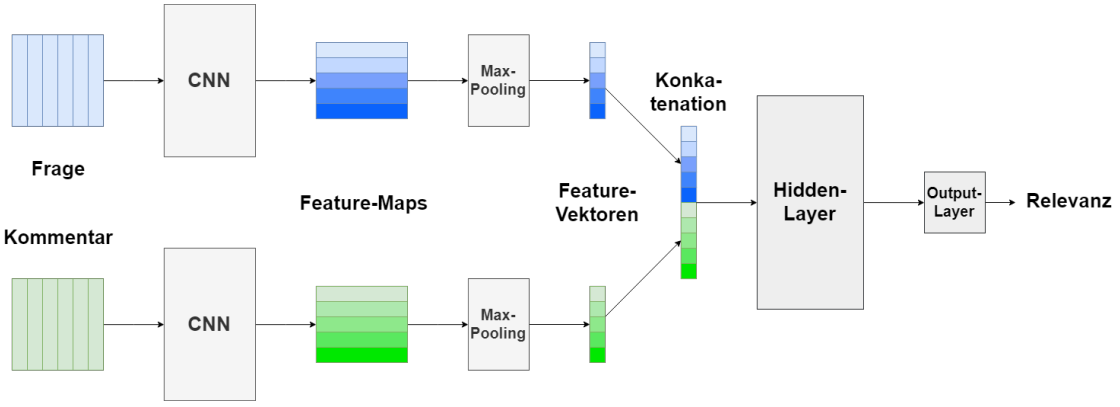

Abbildung 5.1: Schematischer Aufbau des Basismodells.

Netzaufbau

Als Eingabe nimmt das Netz Frage-Kommentar-Paare entgegen. Das Modell setzt zwei CNNs ein, wobei das eine die Frage und das andere den Kommentar als Input erhält. Beide Netze besitzen die selbe Filteranzahl und Filtergrößen benutzen aber getrennte Gewichte. Dies ermöglicht das Erkennen von unterschiedlichen Mustern in den Frage- und Kommentartexten. Die Intuition hinter dem Einsatz von CNNs für die Verarbeitung von Texten ist, dass diese Muster in Form von aufeinanderfolgenden Wörtern, auch N-Grams genannt, erkennen. Dabei erlernen die Netze anhand der Trainingsdaten die Muster, die für die Relevanzklassifizierung entscheidend sind.
Jeder Filter der CNNs wird auf eine Position in den Eingabetexten angewandt, wofür der linke und rechte Rand der Textmatrizen mit Nullvektoren aufgefüllt wird. Da alle Texte auf 200 Wörter gestreckt oder gestutzt werden, entstehen für jeden Filter 200 Ausgabewerte. Nach den CNNs werden diese Werte mit Hilfe von Max-Pooling auf jeweils einen Wert pro Filter reduziert. Das bedeutet für jeden Filter wird nur die höchste Aktivierung behalten und die Position dieser Aktivierung im ursprünglichen Text fließt nicht in die Entscheidung mit ein.
Angenommen die CNNs setzen jeweils F Filter ein, dann ist die Ausgabe jedes Netzes nach Pooling ein Vektor der Länge F. Diese Vektoren werden als Feature-Vektoren bezeichnet. Die CNNs übernehmen also die Aufgabe des “Feature-Engineerings”, welche in anderen Verfahren von menschlichen Entwicklern übernommen wird. Die Feature-Vektoren beider Netze werden zu einem einzelnen Vektor der Länge $2F$ konkateniert. Dieser dient als Input für einen Fully-Connected-Layer auf den der Output-Layer des Modells folgt. Diese Ausgabeschicht besteht aus einem einzelnen Neuron mit Sigmoid-Aktivierung. Die Neuronen in allen anderen Schichten nutzen die ReLU-Aktivierung.

Parametername	Wert
cnn_no_filter	[128, 1024, 2048]
cnn_kernel_size	[3, 5, 7]
lstm_units	Nicht benötigt
sl_no_hidden_layers	1
sl_no_units	[128, 1024, 2048]

Tabelle 5.2: Werte der Hyperparameter für die Gridsearch. Für Parameter mit mehreren Werten wird jeweils ein Modell pro Wert konstruiert.

Modellparameter

Um die Gridsearch durchzuführen müssen für die Parameter aus der Liste in 4.1.1, die angepasst werden sollen eine Reihe von Parameterwerten festgelegt werden. Die variablen Parameter und die gewählten Werte sind in der Tabelle 5.2 aufgeführt. Aus diesen Parametern lassen sich 27 verschiedene Kombinationen erstellen. Für jede davon wird ein Modell trainiert. Die restlichen Parameter werden wie in Abschnitt 5.1 gewählt.

Ergebnisse

Die Werte der Evaluationsmetriken für alle trainierten Modelle sind in Tabelle A.1 im Anhang aufgeführt. Die Ergebnisse der besten fünf Modelle sind zusätzlich in Tabelle 5.3 aufgelistet. Die Parameterkonfigurationen sind in Tabelle A.2 zu finden.
Die Werte für den MAP-Score liegen zwischen 0.7891 für das schlechteste und 0.8122 für das beste Modell. Ein ausschlaggebender Parameter scheint die Filteranzahl in den CNNs zu sein. Von den Top 10 Modellen besitzen sieben jeweils 2048 Filter pro CNN. Unter den letzten zehn Modellen finden sich hauptsächlich solche mit lediglich 128 Filtern. Die Erhöhung der Anzahl der Filter gibt dem Modell die Möglichkeit eine größere Anzahl an Mustern im Input zu erkennen, was offensichtlich zu einer Verbesserung führt.
Die Graphen in den Abbildungen 5.2 und 5.3 zeigen die Entwicklung der Genauigkeit und dem Wert der Lossfunktion während des Trainings. Dabei werden sowohl die Werte für den Trainingsdatensatz (Blau) als auch für die Validierungsdaten (Orange) am Ende jeder Epoche gezeigt. Zu erkennen ist, dass das Modell schnell lernt die Trainingsdaten richtig zu klassifizieren. Innerhalb der ersten zehn Epochen erreicht das Modell eine Genauigkeit von annähernd 100% und einen entsprechend kleinen Loss. Allerdings verschlechtern sich ab diesem Punkt die Werte für die ungesehenen Validierungsdaten. Dies ist ein Hinweis auf Overfitting, das Modell lernt die Muster in den Trainingsdaten also so genau, dass sich die Vorhersagen auf neuen Daten verschlechtern. Dieses Problem soll in den Experimenten 5.2.2 und 5.2.3 weiter untersucht werden.
Wie zuvor vermutet erreicht aber auch das Top-Modell noch keine Ergebnisse, die sich

Modell	MAP	Genauigkeit	MRR	F1-Score	Precision	Recall
1	**0.8122**	0.6532	88.3783	0.5662	0.8095	0.4353
2	0.8108	0.6573	**88.6508**	**0.5767**	0.8057	**0.4491**
3	0.8098	**0.6584**	88.037	0.5749	**0.8137**	0.4445
4	0.8097	0.6468	88.1682	0.5514	0.8112	0.4176
5	0.809	0.6495	88.0156	0.5628	0.8002	0.434

Tabelle 5.3: Metriken der fünf besten Modelle aus der Gridsearch.

mit den anderen Lösungen messen lassen. Dort würde das aktuelle Modell den 11. Platz von 14 belegen. Eine solche Performance ist aber wie bereits in der Experimenthypothese beschrieben auf Grund der wenigen Trainingsdaten und der Modellstruktur zu erwarten. Ziel der Gridsearch war es, eine solide Parameterkonfiguration zu finden. Dementsprechend soll das Top-Modell mit 2048 Filtern in beiden Netzen, einer Filtergröße von 5 und 1024 Neuronen im Hidden-Layer als Basis für die Modelle der anderen Experimente dienen.

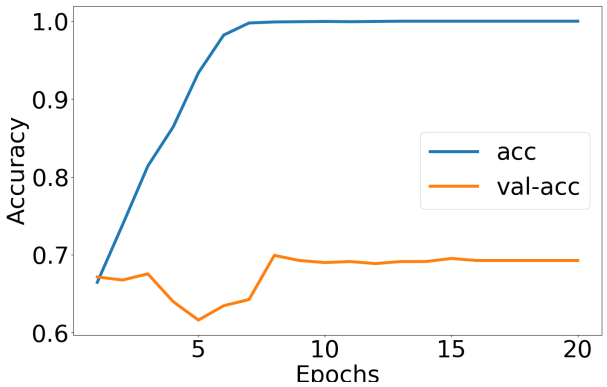

Abbildung 5.2: Entwicklung der Trainingsgenauigkeit.

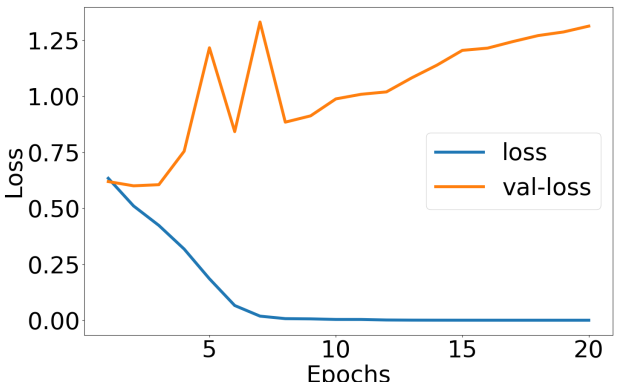

Abbildung 5.3: Entwicklung der Lossfunktion während des Trainings.

Parametername	Wert
cnn_no_filter	2048
cnn_kernel_size	5
lstm_units	Nicht benötigt
sl_no_hidden_layers	1
sl_no_units	1024

Tabelle 5.4: Werte der Hyperparameter des besten Modells aus der Gridsearch.

5.2.2 Erhöhung der Datenmenge

Die Modelle aus dem letzten Experiment wurden nur auf einer kleinen Datenmenge trainiert um Trainingszeit einzusparen. Aus diesem Grund soll in einem weiteren Experiment das Modell mit den laut der Gridsearch besten Parametern erneut trainiert werden. Diesmal wird aber der vollständige Trainingsdatensatz genutzt.
Die Intention hinter dieser Änderung ist, dass dem Modell weitere verschiedene Beispiele zur Verfügung stehen. Dadurch fällt der Einfluss von Mustern, die zuvor nur in dem kleinen Datensatz vorkamen und das Modell ist in der Lage einen allgemeineren Zusammenhang zwischen Eingabetexten und Relevanzklasse zu erlernen. Die Hypothese für das Experiment ist also, dass das in 5.2.1 aufgetretene Overfitting-Problem abschwächt und eine Verbesserung der Metriken zu beobachten ist.

Modellparameter

Neben den Standardparametern werden für das Experiment die Parameter übernommen, die in der Gridsearch die besten Ergebnisse erzielt haben. Diese sind in Tabelle 5.4 aufgeführt.

Ergebnisse

Die Ergebnisse in Tabelle 5.5 zeigen, dass die Erhöhung der Datenmenge durchaus zu einer Verbesserung des Modells geführt hat. Der Wert für den MAP-Score ist im Vergleich zu dem selben Modell aus der Gridsearch gestiegen. Hervorzuheben sind auch die deutliche Verbesserung der Genauigkeit, des Recalls und daraus resultierend auch des F1-Scores. Da der Recall angibt wie viele von den relevanten Kommentaren korrekt klassifiziert wurden lässt sich aussagen, dass das neue Modell relevante Kommentare öfter erkennt als sein Vorgänger. Dies ist von Vorteil, da es beim Information Retrieval und Ranking insbesondere wichtig ist, relevante Ergebnisse zu erkennen und möglichst weit vorne einzusortieren.
Das in 5.2.1 angesprochene Overfitting konnte allerdings nicht beseitigt werden. Wie die Abbildungen 5.4 und 5.5 zeigen ist zwar die Genauigkeit auf den Validierungsdaten ge-

Modell	MAP	Genauigkeit	MRR	F1-Score	Precision	Recall
Full-Data	0.833	0.7502	88.0469	0.7408	0.804	0.6868

Tabelle 5.5: Metriken des mit dem vollem Datensatz trainierten Modells.

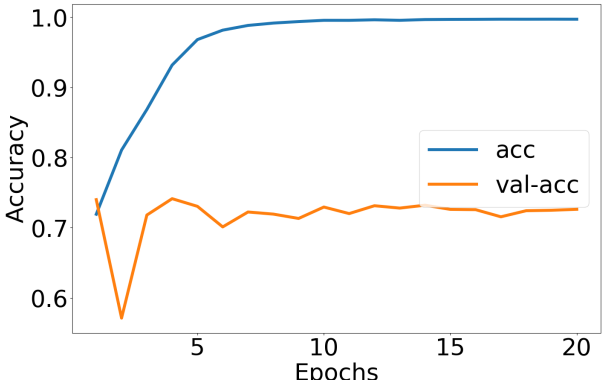

Abbildung 5.4: Entwicklung der Trainingsgenauigkeit beim Training mit allen Daten

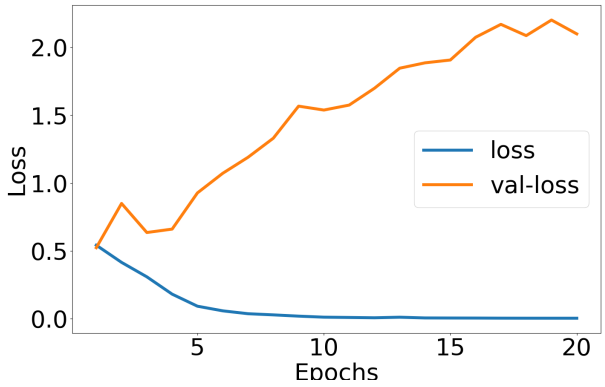

Abbildung 5.5: Entwicklung der Lossfunktion während des Trainings mit allen Daten.

stiegen, der Wert der Lossfunktion weicht aber noch stärker als zuvor von den Werten der Trainingsdaten ab.

Festzuhalten ist also, dass eine Erhöhung der Datenmenge zu einer Verbesserung führt aber das Overfitting-Problem nicht löst. Somit konnte die Hypothese nur zum Teil bestätigt werden.

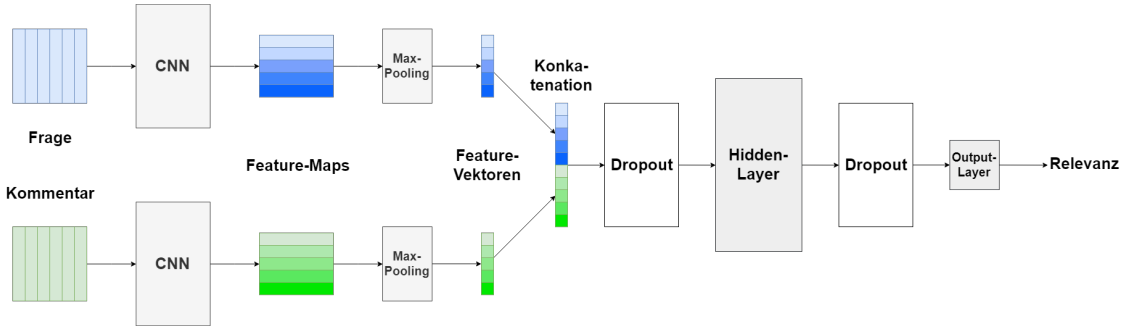

Abbildung 5.6: Potentielle Positionen für Dropout-Layer.

5.2.3 Dropout und Regularisierung

Im Experiment 5.2.1 wurde beobachtet, dass sich die Performance der Modelle auf den Trainingsdaten von der auf dem Validierungsdatensatz unterscheidet. Auch das beste Netz passte sich zu stark an die Daten des Trainings an, wodurch sich die Qualität der Vorhersagen auf ungesehenen Daten senkt.
Um dem Overfitting entgegen zu wirken wird in diesem Experiment noch einmal das beste Modell aus der Gridsearch trainiert. Neu ist der Einsatz von Dropout-Layern beziehungsweise L2-Regularisierung. Mit diesen Techniken soll die Komplexität des Netzes reduziert und ein zu starkes Anpassen an die Trainingsdaten verhindert werden. Dabei sollen die jeweiligen Methoden getrennt voneinander getestet werden. Für die Dropout-Layer werden drei Netze trainiert. Das erste besitzt einen Dropout-Layer nach der Konkatenation der Feature-Vektoren (Dropout-Concat), während das zweite Dropout nach dem Hidden-Layer einsetzt (Dropout-Hidden). Das dritte Modell fügt einen Dropout-Layer an beiden Stellen ein (Dropout-Both). Die genauen Positionen sind in Abbildung 5.6 dargestellt.

Für die L2-Regularisierung kommen ebenfalls zwei Modelle zum Einsatz. Diese nutzen unterschiedliche Werte für den Lambda-Parameter. Auf Grund der größeren Menge zu trainierender Modelle wird für dieses Experiment ebenfalls der “dev”-Datensatz eingesetzt. Es wird erwartet, dass die hier eingesetzten Techniken das Overfitting der Netze reduzieren, dass also die Ergebnisse für die Trainings- und Validierungsdaten näher beieinander liegen.

Modellparameter

Als Vorlage für die Parameter wird das beste Modell aus 5.2.1 gewählt. Diese sind in Tabelle 5.6 aufgeführt. Neu hinzukommen der Parameter für die Wahrscheinlichkeit, dass ein Neuron beim Dropout ausfällt und die Lambda-Parameter für die L2-Regularisierung. Die Dropout-Wahrscheinlichkeit wird in allen Modellen als 0.5 gewählt. Für den Lambda-Parameter werden die Werte 0.01 (L2-0.01) und 0.001 (L2-0.001) getestet.

Parametername	Wert
cnn_no_filter	2048
cnn_kernel_size	5
lstm_units	Nicht benötigt
sl_no_hidden_layers	1
sl_no_units	1024
dropout_rate	0.5
l2_parameter	[0.01, 0.001]

Tabelle 5.6: Werte der Hyperparameter für die Dropout- und Regularisierungsmodelle.

Modell	MAP	Genauigkeit	MRR	F1-Score	Precision	Recall
L2-0.01	**0.8149**	**0.6894**	**87.9868**	**0.6574**	0.7705	**0.5732**
Dropout-Hidden	0.8046	0.6283	87.5264	0.4984	0.8349	0.3552
Dropout-Both	0.804	0.6358	87.4644	0.5192	0.8276	0.3782
Dropout-Concat	0.8028	0.6266	86.9893	0.4893	**0.8465**	0.3441
L2-0.001	0.8001	0.6392	87.1779	0.5406	0.7995	0.4084

Tabelle 5.7: Ergebnisse für die Experimente mit Dropout und L2-Regularisierung.

Ergebnisse

An den Ergebnissen in Tabelle 5.7 ist zu erkennen, dass die meisten Modelle schlechtere Werte erzielen, als das Basismodell aus 5.2.1. Nur das Netz L2-0.01 kann einen höheren MAP-Score erreichen. Der Vorsprung fällt aber nicht signifikant aus.
Die Metrikverläufe in den Abbildungen 5.7 und 5.8 zeigen, dass sich der Einsatz von L2-Regularisierung auf das Trainingsverhalten auswirkt. Die Genauigkeit auf den Trainingsdaten steigt wesentlich langsamer an und auch der Loss fällt nicht so stark wie zuvor (vergleiche Abbildungen 5.2 und 5.3). Das Modell passt sich also weniger an die Trainingsdaten an als zuvor.
Allerdings verbessern sich hierdurch die Vorhersagen auf ungesehenen Daten nicht wodurch es zu keine signifikante Performance-Steigerung kommt. Nichtsdestotrotz lässt sich aussagen, dass die L2-Regularisierung mit dem gewählten Parameter das Overfitting abschwächt. Damit bestätigt sich die Experimenthypothese zumindest zum Teil. Die Abbildungen 5.9 und 5.10 zeigen jedoch, dass Dropout zumindest in diesem Fall kein Abhilfe schafft.

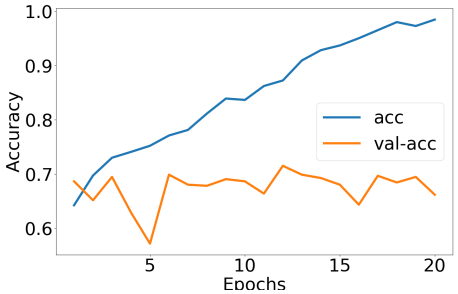

Abbildung 5.7: Entwicklung der Trainingsgenauigkeit des Modells L2-0.01.

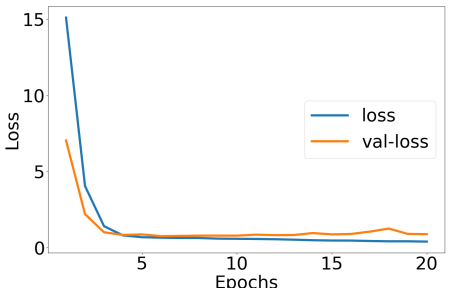

Abbildung 5.8: Entwicklung der Lossfunktion während des Trainings des Modells L2-0.01.

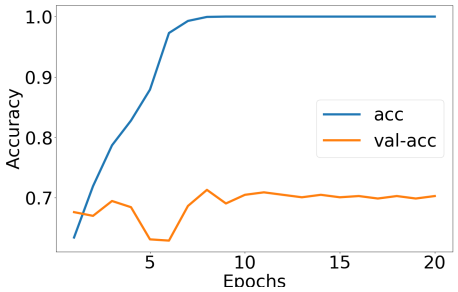

Abbildung 5.9: Entwicklung der Trainingsgenauigkeit des Modells Dropout-Hidden.

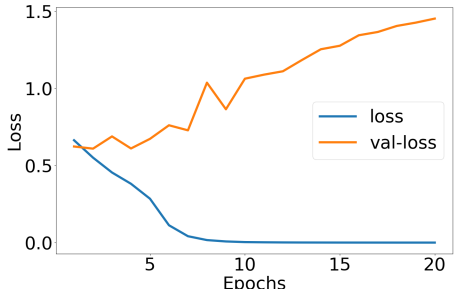

Abbildung 5.10: Entwicklung der Lossfunktion während des Trainings des Modells Dropout-Hidden.

5.2.4 MultiCNN

In den vorherigen Experimenten nutzten alle CNNs immer genau einen Wert für die Größe der Filter. Diese Filtergröße bestimmt intuitiv wie lang die Wortgruppen oder N-Grams sind, die von den Netzen betrachtet werden. Die CNNs erlernen die Wortmuster der entsprechenden Länge und deren Zusammenhang zur Relevanz des Kommentars. Allerdings können aussagekräftige Wortmuster in verschiedenen Größen vorkommen. Die Gridsearch aus dem ersten Experiment hat ergeben, dass sich mit einer Filtergröße von 5 die besten Ergebnisse erzielen lassen. Dennoch kann es sein, dass kürzere und längere Wortmuster aufschlussreiche Indikatoren für die Relevanzbewertung existieren.
Aus diesem Grund soll in diesem Experiment eine Art des CNNs eingesetzt werden, welches mehrere Filtergrößen gleichzeitig einsetzen kann. Ein ähnliches Modell kommt auch in [7] zum Einsatz. Dieses Modell wird im folgenden MultiCNN genannt. Auch mit unterschiedlichen Werten für die Filtergröße ändert sich die Ausgabe des Netzes nicht. Nach wie vor wird für jeden Filter eine Aktivierung für jede Position im Text ausgegeben und durch das Pooling auf den größten Wert reduziert.
Für das Experiment wird erwartet, dass die gleichzeitige Betrachtung unterschiedlich großer Wortmuster eine allgemeinere Analyse der Daten ermöglicht. Es wird also untersucht ob sich die Evaluationswerte verbessern.

Modellparameter

Alle Parameter bis auf die Filtergröße und -anzahl werden aus dem vorherigen Experiment übernommen. Die CNNs benutzen gleichzeitig die Filtergrößen 1, 3, 5 und 7. Es werden insgesamt zwei Modelle getestet, die unterschiedliche Werte für die Anzahl der Filter einsetzen. Das erste Modell (MultiCNN-2048) benutzt für jede Filtergröße jeweils 2048 Filtermaps, also insgesamt 8192 Filter. Somit stehen für jede Filtergröße so viele Filter zur Verfügung wie in den vorherigen Modellen für die jeweils einzige Größe. Dadurch werden zwei Parameter gleichzeitig angepasst was zu einem zusätzlichen Einfluss auf die Ergebnisse führen kann. Aus diesem Grund wird ein zweites Modell (MultiCNN-512) trainiert welches für jede Filtergröße 512 also insgesamt 2048 Filter einsetzt. Dadurch besitzt dieses Modell insgesamt genauso viele Filter wie das ursprüngliche Netz.

Ergebnisse

Wie die Ergebnisse in Tabelle 5.9 zeigen, verbessert sich das MultiCNN gegenüber dem vorherigen Modell. Somit erfüllen sich die Erwartungen an das Modell. Dabei erreichen sowohl das MultiCNN-512 als auch das MultiCNN-2048 ähnliche Werte. Die Verbesserung der Ergebnisse lässt sich also hauptsächlich durch die Verwendung mehrerer Filtergrößen erklären und ist weitestgehend unabhängig von der Anzahl der verwendeten Filter.
Auch die anderen Metriken, mit Ausnahme der Precision, verbessern sich gegenüber dem

Parametername	Wert
cnn_no_filter	512 oder 2048
cnn_kernel_size	[1, 3, 5, 7]
lstm_units	Nicht benötigt
sl_no_hidden_layers	1
sl_no_units	1024

Tabelle 5.8: Hyperparameter für die MultiCNNs. Es werden zwei Modelle mit verschiedenen Parametern für die Filteranzahl konstruiert die alle Werte für die Filtergröße gleichzeitig nutzen.

Modell	MAP	Genauigkeit	MRR	F1-Score	Precision	Recall
MultiCNN-512	**0.8431**	0.7638	**89.99**	**0.7633**	0.7966	**0.7328**
MultiCNN-2048	0.8409	**0.7669**	89.2244	0.7614	**0.8134**	0.7157

Tabelle 5.9: Metriken für die trainierten MultiCNNs.

Modell aus 5.2.2. Ein Anstieg der Recall-Wertes bei gleichzeitigem sinken der Precision lässt darauf schließen, dass das MultiCNN-512 einen größeren Anteil der relevanten Kommentare erkennt, dabei aber auch mehr irrelevante Antworten falsch klassifiziert. Da der MAP-Score sich allerdings erhöht scheint das Modell im Allgemeinen besser in der Lage zu sein relevanten Kommentaren höhere Ausgabewerte zuzuweisen, wodurch sie im anschließenden Ranking höher einsortiert werden.

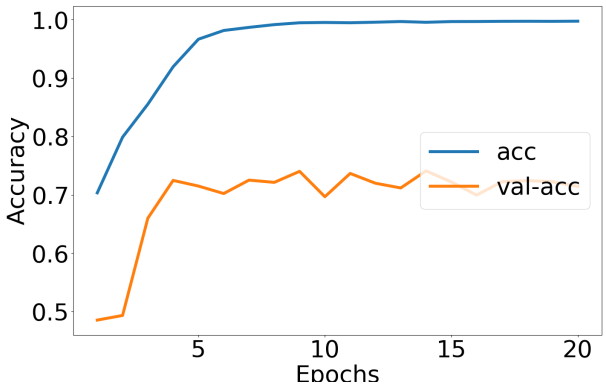

Abbildung 5.11: Entwicklung der Trainingsgenauigkeit für das MultiCNN-512.

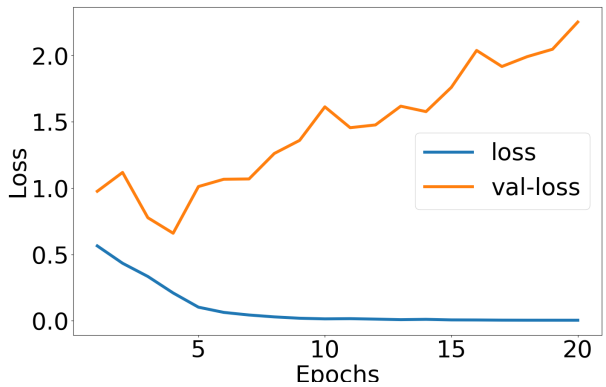

Abbildung 5.12: Entwicklung der Lossfunktion während des Trainings des MultiCNN-512.

Parametername	Wert
cnn_no_filter	Nicht benötigt
cnn_kernel_size	Nicht benötigt
lstm_units	64 oder 256
sl_no_hidden_layers	1
sl_no_units	1024

Tabelle 5.10: Parameter für die LSTMs.

5.2.5 LSTM

Allen vorherigen Netzen ist gemein, dass sie für das Feature-Engineering CNNs einsetzen. Es ist aber auch möglich, diese Aufgabe mit LSTMs zu lösen [17]. Für dieses Experiment werden daher die beiden CNNs durch LSTMs ersetzt. Da die Inputtexte alle die Länge 200 haben besitzen die Netze jeweils 200 Zellen, eine für jede Stelle in den Texten.
Als Output der LSTMs wird der Output jeder einzelnen Zelle verwendet. Dabei kann die Größe der Ausgabe durch einen Hyperparameter bestimmt werden. Angenommen der Wert dieses Parameters sei N, dann ist der Output jeder Zelle ein Vektor der Länge N. Die Ausgabe des gesamten LSTMs ist damit eine Matrix der Form (200, N) also der Zusammenschluss der einzelnen Vektoren. Diese Matrix wird nach dem LSTM in einen flachen Vektor der Länge $200 \cdot N$ umgewandelt indem die einzelnen Vektoren konkateniert werden. Dieser Vektor wird anschließend wie die Ausgabe der CNNs in den anderen Netzen behandelt. Das bedeutet, dass die Ausgabevektoren beider LSTMs konkateniert werden und dann als Eingabe in einen Hidden-Layer gegeben werden.
Da LSTMs die Ergebnisse der vorherigen Zeitschritte bei jeder Entscheidung nutzen, können sie Zusammenhänge von Wörtern verarbeiten, die unter Umständen weit auseinander liegen. CNNs betrachten hingegen immer nur Gruppen von Wörtern die direkt aufeinander folgen. Es ist also interessant den Einsatz von LSTMs zu untersuchen. Dieses Experiment soll feststellen wie sich die Änderung der Architektur auf die erzielten Metrikwerte auswirkt.

Modellparameter

Für alle Parameter die dieses Modell mit den vorherigen gemein hat werden die Werte des besten Modells aus der Gridsearch übernommen. Durch die Nutzung von LSTMs kommt ein neuer Parameter hinzu, die Anzahl der Neuronen die in den einzelnen Bestandteilen der LSTM-Zellen zum Einsatz kommen. Für dieses Experiment werden zwei Werte für diesen Parameter getestet und entsprechend zwei verschiedene Modelle trainiert. Die Liste der Hyperparameter ist in Tabelle 5.10 zu finden.

Modell	MAP	Genauigkeit	MRR	F1-Score	Precision	Recall
LSTM-64	**0.8325**	0.7201	89.8075	**0.708**	0.7735	**0.6527**
LSTM-256	0.8304	**0.7229**	**90.1365**	0.7071	**0.7846**	0.6435

Tabelle 5.11: Metriken für die trainierten LSTM Modelle.

Ergebnisse

Die Ergebnisse für beide trainierten Modelle sind aus der Tabelle 5.11 zu entnehmen. Zu erkennen ist, dass sich keine signifikanten Unterschiede zwischen dem LSTM-64 und dem LSTM-256 feststellen lassen. Auch im Vergleich zu dem Modell aus 5.2.2 werden ähnliche Werte für den MAP-Score erzielt. Die restlichen Metriken mit Ausnahme des MRR-Scores fallen allerdings schlechter aus als bei dem Modell, welches auf CNNs basiert.
Betrachtet man zum Vergleich das MultiCNN aus 5.2.4 so fällt der Unterschied noch deutlicher aus. Das LSTM-64 liegt mit einem MAP-Score von 0.8325 hinter dem MultiCNN-512 mit 0.8431 zurück. Auch in den anderen Metriken erreicht es zum Teil deutlich schlechtere Werte.
Betrachtet man die Graphen in den Abbildungen 5.13 und 5.14 so erkennt man, dass der Verlauf der Genauigkeitskurve für die Trainingsdaten zunächst etwas flacher ausfällt als bei den vorherigen Modellen. Allerdings ist das selbe Overfitting-Verhalten wie auch zuvor zu erkennen.
Allgemein lässt sich aussagen, dass das LSTM Modell in den Tests eine schlechtere Performance zeigt, als die Modell auf CNN-Basis. Dies könnte sich aber durch einen größeren Datensatz ändern, da LSTMs in der Regel auf Grund der erhöhten Anzahl interner Parameter von einer Erhöhung der Beispiele profitieren.

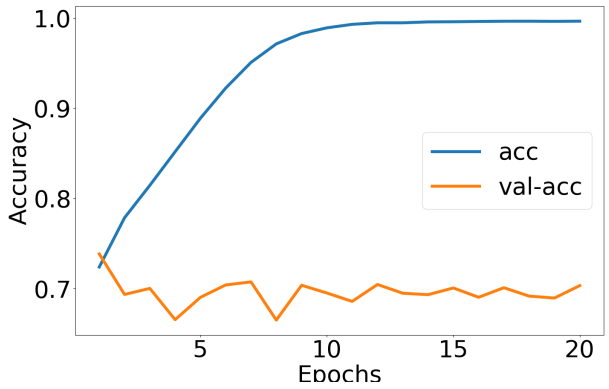

Abbildung 5.13: Entwicklung der Trainingsgenauigkeit für das LSTM-64.

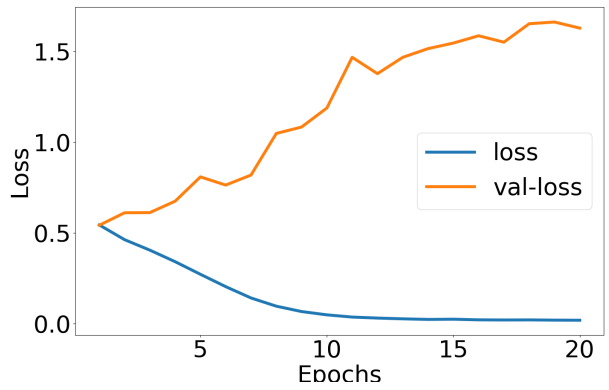

Abbildung 5.14: Entwicklung der Lossfunktion während des Trainings des LSTM-64.

Parametername	Wert
cnn_no_filter	2048
cnn_kernel_size	5
lstm_units	64 oder 256
sl_no_hidden_layers	1
sl_no_units	1024

Tabelle 5.12: Werte der Hyperparameter für die C-LSTMs.

5.2.6 C-LSTM

Bisher wurden für die Netze, die Inputtexte in Feature-Vektoren umwandeln entweder CNNs oder LSTMs eingesetzt. Es ist aber auch möglich, beide Architekturen zu nutzen. Dieses Modell trägt den Namen C-LSTM [32]. Wie in 5.2.1 beschrieben enthält die Ausgabe eines CNNs einen Wert für jeden der F Filter für jede mögliche Position im Text. Mit Texten die 200 Wörter enthalten ist die Ausgabe also eine Matrix der Form (200, F). Diese kann nun wiederum als Input für ein LSTM genutzt werden.
Das LSTMs aus 5.2.5 erhielten als Input jeweils einen Text der Form (200, 300), also 200 Wortvektoren der Länge 300. Jeder einzelne Wortvektor stellte dabei einen Zeitschritt da, der als Input für eine Zelle des LSTMs genutzt wurde. Für dieses Experiment wird die Eingabe für die t-te Zelle des LSTMs durch die Aktivierungen der Filter aus dem CNN an der t-ten Stelle im Text gebildet. Dies ist also ein Vektor der Länge F.
Bei diesem Modell kommt kein Pooling zum Einsatz, da der Output des CNNs den Input des LSTMs bildet. Dadurch werden anders als zuvor keine Werte mehr verworfen und die Position an der die CNN-Filter eine bestimmte Aktivierung erzeugt haben kann sich auf die Entscheidung des Netzes auswirken.
Die Hypothese für das Experiment ist, dass sich sich die Ergebnisse des Modells durch den gleichzeitigen Einsatz von CNNs und LSTMs verbessern. Dazu sollen die erzielten Metrikwerte mit den vorherigen Modellen verglichen werden.

Modellparameter

Das Modell nutzt die Parameter die sich in der Gridsearch bewährt haben. Da sich für die Anzahl der Neuronen in den LSTM-Layern keine klar besseren Ergebnisse für eine der beiden getesteten Werte ergeben haben werden auch für dieses Experiment wieder zwei Modelle trainiert, die jeweils einen der Werte übernehmen. Die Hyperparameter sind in 5.12 aufgeführt.

Modell	MAP	Genauigkeit	MRR	F1-Score	Precision	Recall
C-LSTM-64	**0.8283**	0.7191	88.4707	0.7078	0.7705	0.6546
C-LSTM-256	0.8262	**0.7256**	**89.2455**	**0.7139**	**0.7793**	**0.6586**

Tabelle 5.13: Metriken für die trainierten C-LSTMs.

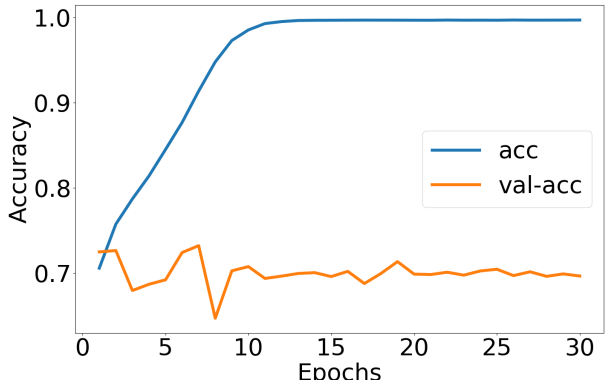

Abbildung 5.15: Entwicklung der Trainingsgenauigkeit für das C-LSTM-64.

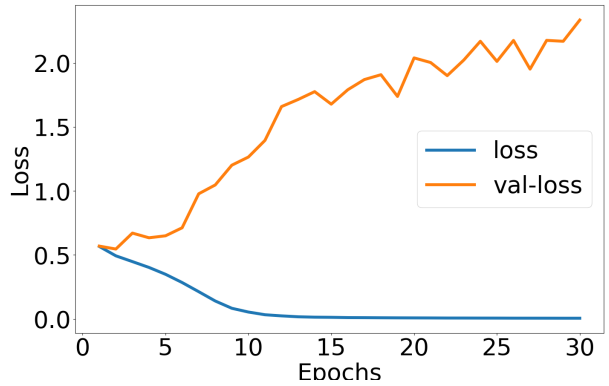

Abbildung 5.16: Entwicklung der Lossfunktion während des Trainings des C-LSTM-64.

Ergebnisse

Die Ergebniss aus Tabelle 5.13 zeigen, dass sich die beiden trainierten Netze im Vergleich zu ihren Vorgängern verschlechtern. Beide C-LSTMs erreichen ähnliche MAP-Scores. Das C-LSTM-64 erreicht dabei einen leicht besseren Wert von 0.8283. Damit kann es weder das LSTM-64 mit 0.8325 noch das MultiCNN-512 mit 0.8431 erreichen. Auch bei den anderen Metriken lassen sich keine Vorteile für die C-LSTMs erkennen. Die Hypothese, dass die C-LSTMs die Ergebnisse verbessern, kann also nicht bestätigt werden.

5.2.7 Vergleichsmodell

Obwohl die meisten Lösungen des SemEval Tasks auf manuelles Feature-Engineering setzen stehen auch Arbeiten zur Verfügung, die die Aufgabe mit Hilfe von End-to-End-Modellen bearbeitet haben. Eines dieser Modelle soll im folgenden implementiert werden. Dies dient dazu die Ergebnisse der Arbeit zu bestätigen und einen Vergleich zum eigenen Modell herzustellen.
Als Vorlage wird die Lösung des Teams FuRongWang [29] genutzt. Das Modell dieser Arbeit ähnelt dem MultiCNN-Modell aus 5.2.4. Auch hier werden zwei CNNs genutzt die gleichzeitig unterschiedliche Filtergrößen einsetzen können. Auch der Einsatz des Poolings und des Hidden-Layers ist vergleichbar.
Die Autoren implementieren allerdings zwei Neuerungen. Zum einen wird die Wortüberlappung der beiden Texte als zusätzliches Feature vom Netz genutzt. Dafür wird sowohl für die Frage $q_i = (x_1^q, x_2^q, ..., x_n^q)$ als auch für den Kommentar $c_i = (x_1^c, x_2^c, ..., x_m^c)$ eines Paares ein Vektor nach den folgenden Mustern konstruiert:

$$q_{feat}^{(i)} = \begin{cases} 1, & x_i^q \in c_i \\ 0, & x_i^q \notin c_i \end{cases} \tag{5.1}$$

$$c_{feat}^{(i)} = \begin{cases} 1, & x_i^c \in q_i \\ 0, & x_i^c \notin q_i \end{cases} . \tag{5.2}$$

Hierbei sind $q_{feat}^{(i)}$ und $c_{feat}^{(i)}$ jeweils das i-te Element des Frageüberlappungsvektors q_{feat} und des Kommentarüberlappungsvektors c_{feat}. Diese Vektoren kommen an zwei Stellen zum Einsatz. Zum Einen werden sie an die Wortvektorinpunts angehängt und zum anderen zusammen mit den Ausgaben des Poolings konkateniert. Für ersteres werden die Vektoren durch das Auffüllen mit Nullen oder durch Verkürzen auf die selbe Länge gebracht wie die Texte.
Die andere Neuerung nimmt die Form einer zusätzlichen Schicht im Netzwerk an. Die Feature-Vektoren, die von den CNNs in Kombination mit dem Pooling erzeugt werden bilden den Input für einen sogenannten “Interactive Layer”. Sei der Feature-Vektor für die Frage als q und für den Kommentar als c bezeichnet, dann berechnet sich die Ausgabe des “Interactive Layers” als

$$z_{int} = f(q^\top W c). \tag{5.3}$$

W ist dabei eine $K \times K$-Matrix, mit K als Länge von q und c, deren Werte während des Trainings gelernt werden. Die Ausgabe des Layers ist also ein einzelner Wert, dieser wird zusammen mit den Feature- und Überlappungsvektoren konkateniert.
Das Experiment soll dazu dienen die Ergebnisse des Teams FuRongWang zu bestätigen. Es wird also erwartet, dass ähnliche Werte für die Metriken erreicht werden. Außerdem können die Ergebnisse genutzt werden um dieses Modell mit denen aus den eigenen Experimenten zu vergleichen.

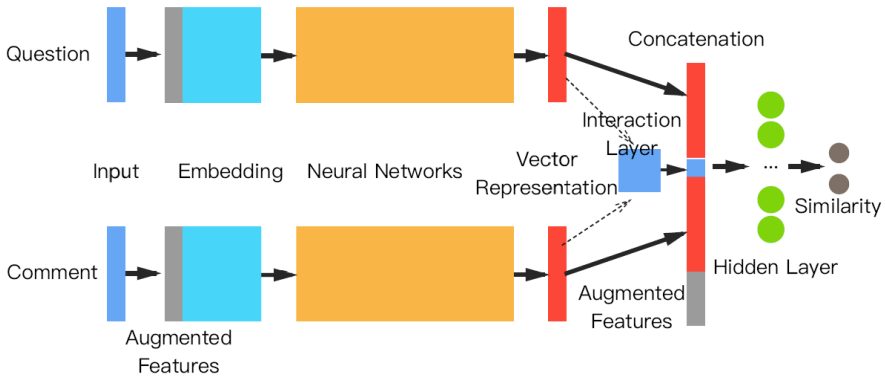

Abbildung 5.17: Aufbau des Vergleichsmodells [29].

Parametername	Wert
cnn_no_filter	800
cnn_kernel_size	[1, 2, 3, 5]
lstm_units	Nicht benötigt
sl_no_hidden_layers	1
sl_no_units	256

Tabelle 5.14: Hyperparameter des Vergleichsmodells. Das Netz nutzt gleichzeitig alle angegebenen Werte für Filtergrößen.

Modellparameter

Für die Parameter werden die Werte genutzt die auch vom Team FuRongWang eingesetzt wurden. Diese sind der Tabelle 5.14 zu entnehmen. Die einzigen Abweichungen sind, dass das selbe Word2Vec-Modell und der selbe Optimizer wie in den anderen Experimenten genutzt wird um einen besseren Vergleich zu erhalten. Da nicht klar ist, ob das Modell genauso schnell lernt wie die anderen Netze wird es über 30 statt 20 Epochen trainiert.

Ergebnisse

Im Test erreicht das hier trainierte Modell einen MAP-Score von 0.831, wie der Tabelle 5.15 zu entnehmen ist. Dieser Wert liegt unter dem von 0.8426 der im Rahmen des SemEval Tasks von dem Modell erreicht wurde [18]. Die Abweichung könnten zum einen durch die Änderung die in diesem Test durchgeführt wurden, also zum Beispiel durch die Verwendung eines anderen Word2Vec-Modells, erklärt werden. Zudem basiert die Implementierung dieses Modells lediglich auf der Beschreibung in dem zu der Lösung veröffentlichten Paper. So ist es also auch möglich, dass sich die hier verwendete Implementierung von der vom

Modell	MAP	Genauigkeit	MRR	F1-Score	Precision	Recall
Vergleichsmodell	0.831	0.7676	89.376	0.7644	0.8077	0.7255

Tabelle 5.15: Metriken für das Vergleichsmodell.

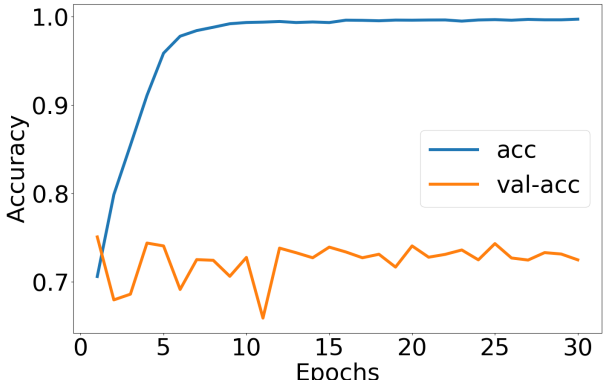

Abbildung 5.18: Entwicklung der Trainingsgenauigkeit für das Vergleichsmodell.

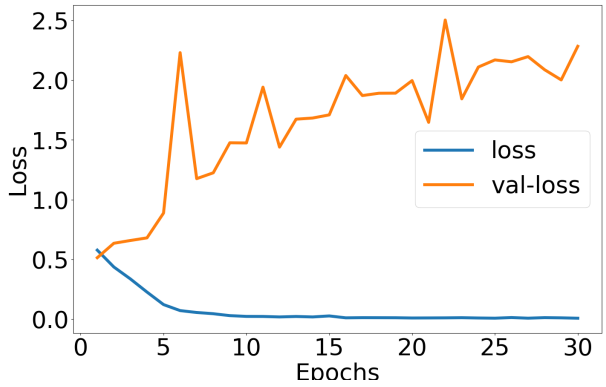

Abbildung 5.19: Entwicklung der Lossfunktion während des Trainings des Vergleichsmodells.

Team FuRongWang eingesetzten in noch weiteren Punkten unterscheidet.
Jedoch befinden sich die Ergebnisse zumindest in einem ähnlichen Bereich. Große Abweichungen in eine Richtung lassen sich nicht feststellen. Im Vergleich zu dem MultiCNN aus 5.2.4 welches dem Vergleichsmodell am ähnlichsten ist erreicht letzteres ein schlechteres Ergebnis. Es lässt sich aber nicht mit Sicherheit aussagen, dass dies an der Verwendung der Überlappungsvektoren oder des Interactive-Layers liegt, da sich die Modelle auch in einer Reihe anderer Parameter wie zum Beispiel der Anzahl der Neuronen im Hidden-Layer unterscheidet.

Modell	MAP	Genauigkeit	MRR	F1-Score	Precision	Recall
Top-Modell-Gridsearch	0.8122	0.6532	88.3783	0.5662	0.8095	0.4353
Full-Data	0.833	0.7502	88.0469	0.7408	0.804	0.6868
L2-0.01	0.8149	0.6894	87.9868	0.6574	0.7705	0.5732
Dropout-Hidden	0.8046	0.6283	87.5264	0.4984	0.8349	0.3552
Dropout-Both	0.804	0.6358	87.4644	0.5192	0.8276	0.3782
Dropout-Concat	0.8028	0.6266	86.9893	0.4893	**0.8465**	0.3441
L2-0.001	0.8001	0.6392	87.1779	0.5406	0.7995	0.4084
MultiCNN-512	**0.8431**	0.7638	89.99	0.7633	0.7966	**0.7328**
MultiCNN-2048	0.8409	0.7669	89.2244	0.7614	0.8134	0.7157
LSTM-64	0.8325	0.7201	89.8075	0.708	0.7735	0.6527
LSTM-256	0.8304	0.7229	**90.1365**	0.7071	0.7846	0.6435
C-LSTM-64	0.8283	0.7191	88.4707	0.7078	0.7705	0.6546
C-LSTM-256	0.8262	0.7256	89.2455	0.7139	0.7793	0.6586
Vergleichsmodell	0.831	**0.7676**	89.376	**0.7644**	0.8077	0.7255

Tabelle 5.16: Ergebnisse aller Experimentmodelle.

5.3 Evaluation der Ergebnisse

Nachdem die Experimente abgeschlossen wurden sollen die Ergebnisse im Folgenden zusammengefasst und bewertet werden. Eine Auflistung der Ergebnisse der Relevanten Modelle findet sich in Tabelle 5.16. Zusammenfassend lässt sich sagen, dass durch den Einsatz des MultiCNN in Kombination mit einer Vergrößerung der Trainingsdatenmenge eine Steigerung der Performance im Vergleich zu den Modellen aus der Gridsearch erreicht werden konnte.

Auch die LSTM- beziehungsweise C-LSTM-Modelle bewiesen die Fähigkeit einen Zusammenhang zwischen den Texten eines Frage-Kommentar-Paares und der Relevanz des Kommentars zu erlernen. Dabei erreichten sie jedoch nicht die Qualität des Multi-CNNs. Dennoch wäre es interessant zu testen wie sich diese Netze verhalten wenn ihnen eine größere Menge an Trainingsdaten zur Verfügung stehen würde.

Das erfolgreichste Modell, das MultiCNN-512 aus 5.2.4, erreichte einen MAP-Score von 0.8431 und liegt damit unter den Lösungen die für den SemEval Task eingereicht wurden unterhalb des Modells des Teams SwissAlps [4] (Platz 7) und des Teams FuRongWang [29] (Platz 8). Die Ergebnisse aller Teilnehmer sind in [18] aufgelistet. Diese beiden Teams setzen für ihre Lösung ebenfalls neuronale Netze ein. Somit fällt das Ergebnis dieser Arbeit in den selben Rahmen wie die anderen End-to-End-Lösungen.

Kapitel 6

Fazit

Der Hauptaspekt dieser Arbeit ist es den Einsatz von End-to-End-Modellen zu testen und zu bewerten. Aus diesem Grund ist es sinnvoll die aus den Experimenten erkennbaren Argumente für und gegen den Einsatz solcher Lösungen zusammenzustellen. Dabei soll nicht nur auf die hier betrachtete Aufgabenstellung eingegangen werden sondern auf auch allgemein auf den Einsatz in NLP-Problemstellungen.
Ein Argument gegen den Einsatz des hier entwickelten Modells ist die reine Performance. Das beste Ergebnis im SemEval Task erreichte das Team KeLP mit einem MAP-Score von 0.8843 [6]. Diese Lösung bietet also signifikant bessere Ergebnisse. Die neuronalen Netze haben hier unter anderem den Nachteil, dass sie für eine gute Performance entsprechend viele Trainingsdaten benötigen und, dass sie zum Overfitting neigen. Wenn für eine Aufgabenstellung das Erzielen der bestmöglichen Ergebnisse ein kritischer Aspekt ist, ist dies ein starkes Argument gegen jedes Modell, welches nicht die Vorhersagen mit der besten Qualität liefert. Wenn also wie hier ein End-to-End-Modell hinter anderen Lösungen zurückliegt ist der Einsatz wenig sinnvoll. Jedoch gibt es auch andere Aspekte neben den erzielten Ergebnissen nach denen ein Modell bewertet werden kann. Diese können bei Aufgabenstellungen bei denen ein gewisser Rückgang der Performance akzeptabel ist zur Wahl eines auf dem Papier schlechteren Modells führen.
So spricht das Automatisieren des Feature-Engineerings für ein neuronales Netz. Die meisten der Lösungen, die im SemEval Task besser abgeschnitten haben benötigen das manuelle Entwickeln und Implementieren von Features. Dazu wird in den meisten Fällen Hintergrundwissen über den zugrundeliegenden Prozess benötigt. Außerdem besteht die Gefahr, dass Features entwickelt werden, die sich für die Lösung der Aufgabe als irrelevant herausstellen. Zugleich können wiederum Merkmale, die tatsächlich relevante Informationen zur Bearbeitung des Problems liefern, von den Entwicklern übersehen werden. Diese Features können dann in den meisten Fällen nicht mehr von den Modellen selbst erlernt werden und so gehen relevante Informationen verloren.
End-to-End-Modell erlernen hingegen selbständig welche Merkmale in den Trainingsdaten

im Zusammenhang mit der vorherzusagenden Zielgröße stehen. Dies spart Entwicklungszeit und verhindert Fehler durch die falsche Feature-Wahl. Dazu sei allerdings gesagt, dass manuelles Feature-Engineering durchaus seine Vorteile hat, wenn es gelingt qualitativ hochwertige Merkmale zu finden. End-to-End-Modelle sind also vor allem dann zu empfehlen wenn es für eine bestimmte Aufgabenstellung nur schwer möglich ist adäquate Features zu entwickeln.
Ein weiterer Vorteil für End-to-End-Modelle ist die Anpassungsfähigkeit an neue Aufgabenstellungen. So kann das hier entwickelte Modell mit neuen Daten auch für andere Probleme trainiert werden, da abgesehen von den Textdaten auf keine zusätzlichen Merkmale gesetzt wird. Wenn also Zusatzinformationen in anderen Datensätzen nicht vorkommen, stellt dies kein Problem dar.
Zuletzt spricht noch speziell für die hier getesteten Modelle, dass sie die Möglichkeit bieten durch manuelle Features erweitert zu werden. So können weitere Merkmale an den Feature-Vektor, der von den beiden neuronalen Netzen im ersten Level erzeugt wird, angehängt werden. Die Modelle sind also zunächst für verschiedene Problemstellungen einsetzbar und können nachträglich auf die einzelne Aufgabe spezialisiert werden um die Performance zu erhöhen.
Abschließend ist die Einsetzbarkeit der hier entwickelten Modelle im "Question-Community-Answering" und für die Relevanzbewertung von Antworten zu bewerten. Das MultiCNN-512, welches in den Experimenten die besten Ergebnisse erzielte, bietet eine im Vergleich zu den anderen Lösungen des SemEval Tasks durchschnittliche Performance. Für das Modell spricht das Erweiterungspotential durch ungetestete Verbesserung und die Möglichkeit es auch in anderen Kontexten einzusetzen. Ist nur die Bearbeitung einer einzelnen Aufgabe interessant und die Performance der einzige Aspekt mit denen mögliche Modelle bewertet werden eignen sich andere Verfahren allerdings besser. Für die Relevanzbewertung von Kommentaren lässt sich abschließend sagen, dass End-to-End-Modelle eine akzeptable Herangehensweise sind, aber andere Verfahren zu bevorzugen sind wenn die Performance das Hauptkriterium ist.

Anhang A

Weitere Informationen

A.1 Ergebnisse der Gridsearch

Modell	MAP	Genauigkeit	MRR	F1-Score	Precision	Recall
1	**0.8122**	0.6532	88.3783	0.5662	0.8095	0.4353
2	0.8108	0.6573	**88.6508**	0.5767	0.8057	0.4491
3	0.8098	0.6584	88.037	0.5749	0.8137	0.4445
4	0.8097	0.6468	88.1682	0.5514	0.8112	0.4176
5	0.809	0.6495	88.0156	0.5628	0.8002	0.434
6	0.8081	0.6539	87.2293	0.5667	0.8115	0.4353
7	0.8057	0.6539	87.9037	0.5637	**0.8177**	0.4301
8	0.8054	0.6536	87.8905	0.5631	0.8175	0.4294
9	0.8048	**0.6601**	87.9834	0.5797	0.8111	0.4511
10	0.8042	0.6427	87.46	0.5477	0.8005	0.4163
11	0.8038	0.6539	86.9589	0.5707	0.8033	0.4425
12	0.8038	0.6594	87.1618	0.5807	0.8063	0.4537
13	0.8038	0.6457	87.6274	0.553	0.8035	0.4215
14	0.803	0.656	87.7071	0.5758	0.8019	0.4491
15	0.803	0.6587	87.1762	0.5795	0.8058	0.4524
16	0.8026	0.6536	87.7079	0.5675	0.8083	0.4373
17	0.8022	0.6488	88.5831	0.5627	0.7976	0.4347
18	0.8019	0.6563	87.3122	0.5767	0.8014	0.4504
19	0.8014	0.657	86.8903	0.5811	0.7957	0.4576
20	0.8009	0.6509	88.0891	0.5736	0.7854	0.4517
21	0.7996	0.6526	86.6447	0.565	0.8091	0.434
22	0.799	0.6512	87.5131	0.5632	0.8066	0.4327
23	0.7987	0.6481	87.1003	0.5523	0.8154	0.4176
24	0.7979	0.6515	87.2149	0.5726	0.7898	0.4491
25	0.7958	0.6481	87.4782	0.5615	0.7971	0.4334
26	0.7943	0.6375	86.2812	0.5418	0.7899	0.4123
27	0.7891	0.6573	86.1641	**0.5922**	0.7764	**0.4787**

Tabelle A.1: Evaluationsergebnisse aller Modelle der Gridsearch aus Abschnitt 5.2.1. Die Parameterkonfigurationen sind in Tabelle A.2 aufgelistet.

Modell	Anzahl CNN-Filter	Filtergröße	Neuronen im Hidden-Layer
1	2048	5	1024
2	2048	3	2048
3	2048	5	2048
4	2048	7	128
5	1024	7	128
6	2048	5	128
7	2048	7	2048
8	1024	5	128
9	2048	3	1024
10	128	5	128
11	1024	3	128
12	2048	7	1024
13	2048	7	1024
14	1024	3	1024
15	2048	3	128
16	1024	5	2048
17	128	5	102
18	1024	3	2048
19	128	3	128
20	128	3	2048
21	1024	7	1024
22	1024	7	128
23	128	7	1024
24	128	5	2048
25	128	7	2048
26	128	7	128
27	128	3	1024

Tabelle A.2: Werte der variablen Hyperparameter der Gridsearch-Modelle.

Abbildungsverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[1] Burges, Chris, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton und Greg Hullender: *Learning to Rank Using Gradient Descent.* In: *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, Seiten 89–96, New York, NY, USA, 2005. ACM.

[2] Burges, Christopher J., Robert Ragno und Quoc V. Le: *Learning to Rank with Nonsmooth Cost Functions.* In: Schölkopf, B., J. C. Platt und T. Hoffman (Herausgeber): *Advances in Neural Information Processing Systems 19*, Seiten 193–200. MIT Press, 2007.

[3] Cao, Zhe, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai und Hang Li: *Learning to Rank: From Pairwise Approach to Listwise Approach.* In: *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, Seiten 129–136, New York, NY, USA, 2007. ACM.

[4] Deriu, Jan Milan und Mark Cieliebak: *SwissAlps at SemEval-2017 Task 3: Attention-based Convolutional Neural Network for Community Question Answering.* In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, Seiten 334–338. Association for Computational Linguistics, 2017.

[5] Feng, Wenzheng, Yu Wu, Wei Wu, Zhoujun Li und Ming Zhou: *Beihang-MSRA at SemEval-2017 Task 3: A Ranking System with Neural Matching Features for Community Question Answering.* Seiten 280–286, 01 2017.

[6] Filice, Simone, Giovanni Da San Martino und Alessandro Moschitti: *KeLP at SemEval-2017 Task 3: Learning Pairwise Patterns in Community Question Answering.* In: *SemEval@ACL*, 2017.

[7] He, Hua, Kevin Gimpel und Jimmy Lin: *Multi-Perspective Sentence Similarity Modeling with Convolutional Neural Networks.* In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Seiten 1576–1586. Association for Computational Linguistics, 2015.

[8] Heckerman, David, Eric Horvitz, Mehran Sahami und Susan Dumais: *A Bayesian Approach to Filtering Junk E-Mail.* July 1998.

[9] Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever und Ruslan Salakhutdinov: *Improving neural networks by preventing co-adaptation of feature detectors.* CoRR, abs/1207.0580, 2012.

[10] I., Goodfellow, Bengio Y und Courville A.: *Deep Learning. Das umfassende Handbuch : Grundlagen, aktuelle Verfahren und Algorithmen, neue Forschungsansätze.* MITP, Frechen, 2018.

[11] Jain, A. K. und Y. H. Li: *Classification of Text Documents.* The Computer Journal, 41(8):537–546, 01 1998.

[12] Joachims, Thorsten: *Text categorization with Support Vector Machines: Learning with many relevant features.* In: Nédellec, Claire und Céline Rouveirol (Herausgeber): *Machine Learning: ECML-98*, Seiten 137–142, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[13] Kim, Yoon: *Convolutional Neural Networks for Sentence Classification.* CoRR, abs/1408.5882, 2014.

[14] Li, Ping, Christopher Burges und Qiang Wu: *McRank: Learning to Rank Using Multiple Classification and Gradient Boosting.* 01 2007.

[15] Liu, Tie-Yan: *Learning to Rank for Information Retrieval.* Foundations and Trends® in Information Retrieval, 3(3):225–331, 2009.

[16] Mikolov, Tomas, Kai Chen, Greg Corrado und Jeffrey Dean: *Efficient Estimation of Word Representations in Vector Space.* CoRR, abs/1301.3781, 2013.

[17] Mueller, Jonas und Aditya Thyagarajan: *Siamese Recurrent Architectures for Learning Sentence Similarity.* In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, Seiten 2786–2792. AAAI Press, 2016.

[18] Nakov, Preslav, Doris Hoogeveen, Lluís Màrquez, Alessandro Moschitti, Hamdy Mubarak, Timothy Baldwin und Karin Verspoor: *SemEval-2017 Task 3: Community Question Answering.* In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, Seiten 27–48. Association for Computational Linguistics, 2017.

[19] P., Goyal, Pandey S. und Jain K.: *Deep Learning for Natural Language Processing.* Apress, Berkeley, CA,, 2018.

[20] S., Skansi: *Introduction to Deep Learning.* Springer, Cham, 2018.

[21] Scott, Sam und Stan Matwin: *Feature Engineering for Text Classification.* In: *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, Seiten 379–388, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[22] Severyn, Aliaksei und Alessandro Moschitti: *Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks.* In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, Seiten 373–382, New York, NY, USA, 2015. ACM.

[23] Sriram, Bharath, Dave Fuhry, Engin Demir, Hakan Ferhatosmanoglu und Murat Demirbas: *Short Text Classification in Twitter to Improve Information Filtering.* In: *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, Seiten 841–842, New York, NY, USA, 2010. ACM.

[24] Taylor, Michael, John Guiver, Stephen Robertson und Tom Minka: *SoftRank: Optimizing Non-smooth Rank Metrics.* In: *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, Seiten 77–86, New York, NY, USA, 2008. ACM.

[25] U., Michelucci: *Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks.* Apress, Berkeley, CA,, 2018.

[26] Wu, Qiang, Christopher J. Burges, Krysta M. Svore und Jianfeng Gao: *Adapting Boosting for Information Retrieval Measures.* Inf. Retr., 13(3):254–270, Juni 2010.

[27] Xie, Yufei, Maoquan Wang, Jing Ma, Jian Jiang und Zhao Lu: *EICA Team at SemEval-2017 Task 3: Semantic and Metadata-based Features for Community Question Answering.* Seiten 292–298, 01 2017.

[28] Yin, Wenpeng, Hinrich Schütze, Bing Xiang und Bowen Zhou: *ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs.* CoRR, abs/1512.05193, 2015.

[29] Zhang, Sheng, Jiajun Cheng, Hui Wang, Xin Zhang, Pei Li und Zhaoyun Ding: *FuRongWang at SemEval-2017 Task 3: Deep Neural Networks for Selecting Relevant Answers in Community Question Answering.* In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, Seiten 320–325. Association for Computational Linguistics, 2017.

[30] Zhang, Xiang, Junbo Zhao und Yann LeCun: *Character-level Convolutional Networks for Text Classification*. In: Cortes, C., N. D. Lawrence, D. D. Lee, M. Sugiyama und R. Garnett (Herausgeber): *Advances in Neural Information Processing Systems 28*, Seiten 649–657. Curran Associates, Inc., 2015.

[31] Zheng, Liang, Yali Zhao, Shengjin Wang, Jingdong Wang und Qi Tian: *Good Practice in CNN Feature Transfer*. CoRR, abs/1604.00133, 2016.

[32] Zhou, Chunting, Chonglin Sun, Zhiyuan Liu und Francis C. M. Lau: *A C-LSTM Neural Network for Text Classification*. CoRR, abs/1511.08630, 2015.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 4. April 2019

Maurice Freund