Zurich University
of Applied Sciences

School of Engineering

SPINNINGBYTES

Projektarbeit Informatik Herbstsemester 2016

TopicThunder: Themenspezifische Sentiment-Analyse mit Deep Learning

Autoren	Tobias Huonder Simon Müller
Hauptbetreuung	Mark Cieliebak
Nebenbetreuung	Dominic Egger
Industriepartner	SpinningBytes AG
Datum	22. Dezember 2016

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum: Unterschriften:

…… ……

……..

……

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Projektarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zusammenfassung

Sentiment-Analyse ist ein Untergebiet von Text-Mining und behandelt die Auswertung von Texten. Dabei werden Informationen bezüglich dem Sentiment aus den Texten gewonnen und bewertet. Das Ziel dieser Arbeit besteht darin, ein auf Twitterdaten ausgelegtes Sentiment-Analyse-System zu verbessern. Zudem soll dieses System auf themenspezifische Sentiment-Analyse ausgebaut werden. Ein weiteres Ziel ist die Teilnahme an SemEval-2017 vorzubereiten. Das bestehende System konnte den SemEval-2016-Wettbewerb im Bereich der Sentiment-Analyse auf Twitterdaten bereits für sich entscheiden. Bisher kann es einen kompletten Tweet nach seinem Sentiment klassifizieren. Um Auswertungen zu einzelnen Themen zu machen, wird die themenspezifische Sentiment-Analyse aber immer wichtiger.
In dieser Arbeit wird untersucht, wie das bestehende System verbessert werden kann. Dafür werden Experimente definiert und ausgewertet. Dazu gehören das Verändern der Batchgrösse, das Verwenden von Class-Weights und das Erweitern der Trainingsdaten um zusätzliche themenspezifische Tweets. Für das Erreichen dieser Ziele wird das bestehende System zuerst genau analysiert und die Architektur komplett überarbeitet. So wird sichergestellt, dass das System einfach zu erweitern ist. Im Anschluss wird eine Baseline gesetzt. Darauf wird mit den Experimenten aufgebaut.
Die Experimente zeigen einen guten Ansatz zur Optimierung. Das Endresultat ist ein neues System, welches das bisherige System im Subtask B um ca. drei Prozentpunkte und im Subtask C um ca. einen Prozentpunkt schlägt. Zudem ist die Architektur sauber strukturiert sowie dokumentiert. Das Preprocessing wird mit einem generischen und modularen Pipeline-Modul geregelt. Die neue Architektur setzt den Grundstein für die Weiterarbeit und den Ausbau des Systems.

Abstract

Sentiment analysis is a branch of text mining that deals with the evaluation of texts, where information concerning the sentiment is extracted and evaluated. Subject of this project report is the improvement of an already existing sentiment analysis system that processes data gathered from Twitter. Additionally the system should be expanded to handle sentiment analysis on a specific topic to prepare for an entry at the SemEval-2017 competition where a year before, at the SemEval-2016, the same system was able to win on the topic of sentiment analysis on data from Twitter. Up to now the system is capable of classifying a complete Tweet by it's sentiment. To go a step further and incorporate the topic, the topic-specific sentiment analysis gains more and more importance.
This project report takes a look at the current system and analyses, where possible improvements can be made. This includes variations in batch size or the usage of class-weights as well as the expansion of training data by including topic-specific Tweets. In order to meet the requirements, the current system's architecture is analysed in detail and fully rewritten to ensure that it can be easily expanded in the future. A baseline is then set where experiments can be set upon.
The experiments show that the chosen approach is promising. The end result is a new system that beats the current one by 3 percentage points in subtask B and 1 percentage point in subtask C. Additionally, the architecture of the new system is properly structured and documented. The preprocessing is done with a generic and modular pipeline-module and, together with the new architecture, forms a solid foundation that can be expanded upon.

Inhaltsverzeichnis

1 Einleitung

Bei der automatischen Sentiment-Analyse entscheidet ein Computer, ob ein Text positiv, negativ oder neutral ist. Das InIT hat ein System für die Sentiment-Anlayse entwickelt. Momentan kann das System einen ganzen Text nach dem Sentiment klassifizieren. Das Ziel dieser Arbeit ist es, die Analyse zu verfeinern und das Thema des Textes zu analysieren. Zusätzlich wurde die Teilnahme am SemEval-2017-Wettbewerb als Ziel definiert. SemEval bedeutet Semantic Evaluation und ist ein anerkannter Wettbewerb im Bereich der Sentiment-Analyse. Das SwissCheese-Team konnte den Subtask A vom Task 4 am SemEval-2016-Wettbewerb für sich entscheiden (vgl. [1]). Ziel ist es, im Jahr 2017, zusätzlich zu Subtask A in den Subtasks B und C die vorderen Ränge zu belegen (vgl. [2]).

1.1 Sentiment-Analyse

Die Sentiment-Analyse ist ein Untergebiet von Text-Mining. Das Ziel von Text-Mining ist die automatische Auswertung von Texten und das Extrahieren von Informationen (vgl. [3]). Dabei wird bei der Sentiment-Analyse der Text nach seinem Sentiment klassifiziert. In dieser Arbeit wird diese Analyse noch spezifischer. Es soll das Sentiment bezüglich eines spezifischen Themas analysiert werden. Bisher wurde nur der Text als Ganzes betrachtet.

1.2 SemEval

SemEval ist ein Wettbewerb für Computersysteme, bei dem die semantische Evaluation von Texten im Vordergrund steht. SemEval wurde 1998 das erste mal durchgeführt und findet nun jährlich statt. Die Aufgaben des Wettbewerbs werden laufend den aktuellen Problemen der Sprachanalyse angepasst. Momentan werden folgende drei Gebiete behandelt:

- „Semantic comparison for words and texts“[4]
- „Detecting sentiment, humor, and truth“[4]
- „Parsing semantic structures“[4]

Jedes dieser Gebiete ist in Tasks unterteilt, für welche man sich anmelden kann. In dieser Arbeit werden lediglich Teilgebiete des zweiten Gebiets behandelt. Das Ziel von SemEval ist, durch den Konkurrenzkampf zwischen den einzelnen Teilnehmern die Kreativität zu fördern. Nach den absolvierten Aufgaben werden alle Lösungsansätze veröffentlicht. Dies soll den technischen Fortschritt und die Innovation vorantreiben.

1.2.1 Task 4

Der Task 4 gehört zum Gebiet „Detecting sentiment, humor, and truth“[4] und analysiert Twitterdaten anhand ihres Sentiments. Dieser Task ist in weitere Subtasks aufgeteilt. Für diese Arbeit sind nur die Subtasks B und C relevant, welche sich auf die themenspezifische Sentiment-Analyse beziehen.

1.2.1.1 Subtask B

> „Given a tweet known to be about a given topic, predict whether it conveys a positive or a negative sentiment towards the topic.“ [1]

1.2.1.2 Subtask C

> „Given a tweet known to be about a given topic, estimate the sentiment it conveys towards the topic on a five-point scale ranging from HIGHLYNEGATIVE to HIGHLYPOSITIVE.“ [1]

1.3 Zielgruppe

Diese Arbeit spricht Studenten und Software-Entwickler an, welche eine Affinität zu Machine-Learning und Sentiment-Analyse besitzen. Zu Beginn der Arbeit werden alle Grundlagen kurz erklärt. Um ein tieferes Verständnis zu entwickeln, werden die Ressourcen in Form von Quellen zur Verfügung gestellt.

2 Grundlagen

Im folgenden Kapitel werden die für diese Arbeit relevanten Themen zusammengefasst.

2.1 Ressourcen

Nachfolgend werden hilfreiche Ressourcen aufgelistet, um sich in die Themen einzuarbeiten und einen erleichterten Einstieg zu ermöglichen.

Thema	**Ressource**
Machine-Learning	Coursera - Machine-Learning [5]
Machine-Learning	Medium Blog - Machine-Learning [6]
Convolutional Neural Network	University of Oxford Youtube - Convolutions [7]
Word-Embeddings	Tensorflow - Word-Embeddings [8]

Tabelle 1: Ressourcen für Grundlagen

2.2 Künstliche Intelligenz

Künstliche Intelligenz befasst sich mit der Automatisierung von intelligentem Verhalten. Das Ziel der Künstlichen Intelligenz ist es, die menschenähnliche Intelligenz mit einem Computer nachzubilden, sodass er eigenständig Probleme bearbeiten kann.

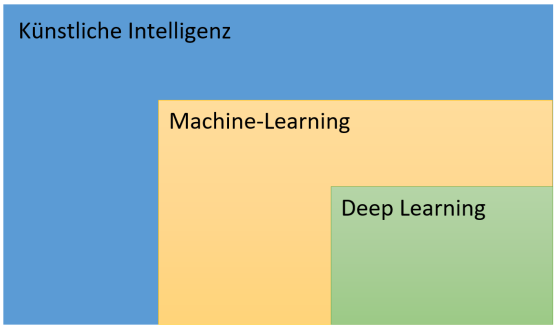

Abbildung 1: Gebiete Künstliche Intelligenz

2.3 Machine-Learning

Machine-Learning ist ein Ansatz der Künstlichen Intelligenz (Abbildung 1). Eine Maschine leitet aus Beispielen eine Logik oder ein Muster ab. Später können diese auf unbekannte Daten angewendet werden. Dabei sollte die Maschine auf unbekannten Daten möglichst gut generalisieren. Diese Fähigkeit zur Generalisierung ist auch das Mass für die Güte des Systems.

Machine-Learning ist der Gegensatz zu einem statischen, regelbasierten Verfahren. Bei einem regelbasierten Verfahren müssen die Regeln statisch definiert werden. Bei Machine-Learning hingegen ist die Maschine fähig, die Regeln dynamisch zu erkennen bzw. zu lernen. Es kann daher auf verschiedene Probleme angewendet werden. Ein sehr simple Art des Machine-Learnings ist zum Beispiel die Lineare Regression.

2.3.1 Supervised Learning

Beim Supervised Learning werden die Trainingsdaten in Form von Input und erwartetem Output zur Verfügung gestellt. Dabei wird nun versucht, die Parameter so zu verändern, dass das Resultat möglichst gut zum erwarteten Wert passt.

2.3.2 Unsupervised Learning

Unsupervised Learning versucht ohne den erwarteten Output, eine Logik oder ein Muster in den Inputdaten zu erkennen und eine Aussage darüber zu machen. Ein gutes Beispiel dafür ist die Kategorisierung von Mails. Als Input bekommt das System eine Menge Mails, von welchen alle ähnlichen Mails vom System in dieselbe Kategorie eingeteilt werden.

2.4 Deep Learning

Deep Learning ist ein weiterer Ansatz von Machine-Learning, welcher direkt vom menschlichen Gehirn inspiriert ist. Deep Learning basiert auf mehrschichtigen neuronalen Netzen und erlangt dadurch schrittweise ein tiefer gehendes Verständnis über eine Information. Mittels der verschiedenen Schichten künstlicher Neuronen kann das gesamte System trainiert werden und sich selbst verbessern. Weitere Informationen zu Neuronalen Netzen sind dem nächsten Abschnitt zu entnehmen.

2.4.1 Neuronale Netze

Die Grundidee der künstlichen, neuronalen Netze kommt von dem Neuronennetz des menschlichen Gehirns. Ziel ist es, mit künstlichen neuronalen Netzen das menschliche Gehirn nachzubauen, damit dieses lernen kann wie ein Mensch. Die künstlichen, neuronalen Netze werden heute eingesetzt, um konkrete Anwendungsprobleme in den verschiedensten Gebieten zu lösen, wie zum Beispiel Texterkennung, Bilderkennung und Gesichtserkennung.

2.4.1.1 Neuronen

Ein Neuronales Netz besteht aus mehreren Neuronen, welche Informationen speichern und mit anderen Neuronen über Kanten verbunden sind (vgl. Abbildung 2).

2.4.1.2 Layer

Neuronen werden in Layer organisiert. Dabei wird zwischen Input, Output und Hidden Layers unterschieden. Mit Hidden Layer wird jeder Layer zwischen Input und Output bezeichnet. Bei einem Fully-Connected-Layer sind alle Neuronen mit denjenigen des nächsten/vorherigen Layers verbunden. Die Kanten gewichten dabei die Information eines Neurons und geben sie weiter (vgl. Abbildung 2).

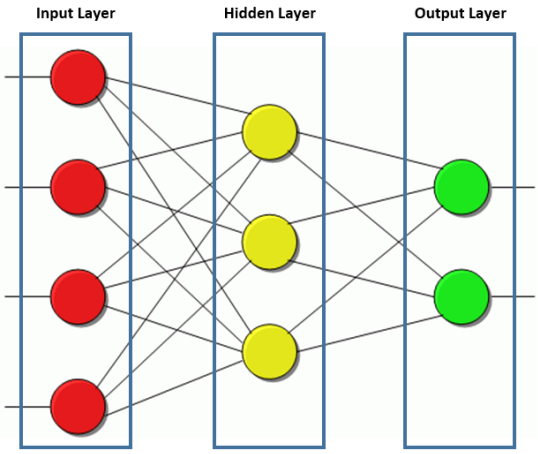

Abbildung 2: Einfaches Neuronales Netz [9]

Abbildung 2 zeigt ein einfaches Neuronales Netz mit einem Hidden Layer und zwei Ouputs. Damit können zwei Klassen unterschieden werden.

2.4.1.3 Training

Input/Output Der Input eines Neuronalen Netzes besteht aus Ausprägungen der Daten und wird am ersten Layer des Netzes angelegt. Dabei steht für jede Ausprägung ein Inputneuron. Bei einem Bild wäre dies zum Beispiel ein einzelnes Pixel.

Der Outputlayer eines Netzes besteht bei einem Klassifikationsproblem aus mehreren Neuronen. Jedes dieser Neuronen repräsentiert eine Klasse. Die Werte der einzelnen Klassen werden normalisiert, um eine Wahrscheinlichkeit für diese Klasse zu erhalten. Danach kann die wahrscheinlichste Klasse ausgewählt werden (binäre Klassen). Ein Datensatz enthält also Inputdaten, bestehend aus Ausprägungen und einer effektiven binären Klasseneinteilung.

Forward Propagation Bei der Forward Propagation wird der Input pro Layer anhand der Kanten gewichtet und weiter propagiert. Im letzten Layer ensteht der Output, die Hypothese. Die Hypothese soll möglichst nahe am effektiven Outputwert sein. Der Unterschied zwischen der Hypothese und dem effektiven Output wird absoluter Fehler genannt. Es gilt nun, diesen Fehler mit einem geeigneten Verfahren zu optimieren.

Backward Propagation Diese Minimierung des absoluten Fehlers geschieht während der Backward Propagation, bei welcher die Gewichte anhand der Gradienten schrittweise wieder zurückgerechnet werden und der absolute Fehler dadurch minimiert wird.

Evaluation Um die Güte des Trainings zu bestimmen, muss das Netz auf ungesehenen Daten evaluiert werden. Ungesehene Daten werden verwendet, da das Netz möglichst gut generalisieren und keine Overfittingprobleme oder hohe Varianz aufweisen soll.

Convolutions Die Möglichkeit der effizienten Berechnung von Deep Convolutional Neural Networks (CNN) ist der Grund für grosse Fortschritte in Deep Learning. Vor allem die Bild- und Texterkennung profitieren davon.

Convolution Layer sind nicht Fully-Connected und können dadurch auch lokale Aspekte der Trainingsdaten lernen. Dies wird mit Filtern erreicht, welche über den Input geschoben werden. Die einzelnen Schritte teilen sich die Gewichte und können so die gelernten Features wiederverwenden. Ein Beispiel dafür ist die Kantenerkennung bei Bildern, wo einmalig das Feature gelernt wird und auf das ganze Bild angewendet werden kann. Um die Information aus dem Output des Convolutional Layer weiter zu extrahieren, wird anschliessend noch ein Pooling-Layer angehängt, welches zum Beispiel nur die markantesten Merkmale behält. Dadurch wird auch die Dimensionalität verringert.

2.4.2 Word-Embeddings

Ein Word-Embedding ist eine Vektorrepräsentation eines Wortes. Dieser Vektor ist eine eindeutige numerische Repräsentation von diesem Wort und hat beliebig viele Dimensionen. Bei der Generierung dieser Word-Embeddings kommen die ursprünglich von Google entwickelten Modelle, welche mit Word2Vec benannt wurden, für das Training zum Einsatz. Dahinter stehen wiederum neuronale Netze (vgl. [10]).

Das Ziel der Word-Embeddings ist es, dass Wörter mit einer ähnlichen Bedeutung im Vektorraum eine kleine Distanz haben. So kann das Computersystem später anhand der Word-Embeddings mit trivialer Vektorarithmetik einfach ähnliche Wörter finden. Je besser diese Word-Embeddings sind, desto besser kann der Computer mit diesen Repräsentation arbeiten. Im Folgenden wird der sonst multidimensionale Vektorraum in zwei Dimensionen abgebildet.

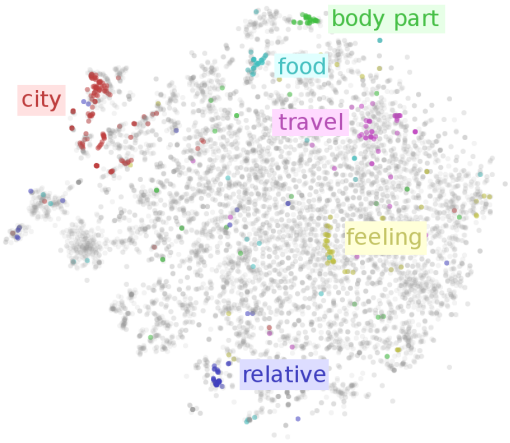

Abbildung 3: Zweidimensionale Repräsentation Word-Embeddings [11]

Jeder Punkt in Abbildung 3 zeigt ein einzelnes Wort. Durch das Training mit Word2Vec sind ähnliche Wörter nah beieinander.

2.5 Evaluation

Um zu messen, wie gut ein System ist, gibt es verschiedene mathematische Verfahren, die einen Score berechnen. Das einfachste Mass wäre die Genauigkeit, welches aber bei Klassifikationsproblemen mit unbalancierten Klassen ungeeignet ist. Um den Effekten des Genauigkeitsparadoxons entgegenzuwirken, wird Precision und Recall verwendet (vgl. [12]).

2.5.1 Precision und Recall

Im folgenden Abschnitt werden die Begriffe Precision und Recall erläutert.

2.5.1.1 Binäre Klassifizierung

Bei einer binären Klassifizierung können vier mögliche Fälle auftreten (vgl. [5]). In der Tabelle 2 werden alle vier Fälle aufgezeigt.

Hypothese	Korrekter Wert: True	Korrekter Wert: False
True	True Positive	False Positive
False	False Negative	True Negative

Tabelle 2: Beurteilung Binärer Klassifizierer

2.5.1.2 Precision

Die Precision ist ein Mass für die Qualität einer Messung.

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{2.1}$$

2.5.1.3 Recall

Der Recall ist ein Mass für die Vollständigkeit einer Messung.

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{2.2}$$

2.5.2 Average

Sobald ein Score über mehrere Klassen berechnet werden muss, wird ein Verfahren benötigt, welches ein Mittel über die jeweiligen Scores berechnet. Dafür gibt es die Folgenden zwei Verfahren.

- Macroaveraged
- Microaveraged

Bezüglich SemEval Subtask C ist aber nur das Macroaverage Verfahren relevant.

2.5.2.1 Micoraverage

Beim Microaverage wird der Score über alle Klassen hinweg berechnet und der Mittelwert gebildet (vgl. [13]). Dadurch haben unbalancierte Klassen einen grossen Einfluss auf das Resultat.

2.5.2.2 Macroaverage

Beim Macroaverage wird der Score pro Klasse berechnet und danach der Mittelwert gebildet (vgl. [13]).

Der Vorteil gegenüber dem Microaverage ist, dass er robuster gegenüber unbalacierter Klassen ist, da zuerst der Average pro Klasse berechnet wird. Der Microaverage bezieht sich also auf einzelne Dokumente, wobei Macroaveraging sich hingegen auf Klassen von Dokumenten bezieht (vgl. [1]).

2.5.3 SemEval

SemEval gibt für die einzelnen Subtasks vor, wie sie ausgewertet werden müssen.

2.5.3.1 Subtask B

Subtask B wird mit dem Macroaveraged Recall ausgewertet. Dabei wird der Macroaveraged Recall pro Thema berechnet und danach der Macroaverage für alle Themen gebildet. Dabei ist ρ^{PN} der Macroaveraged Recall pro Thema der einzelnen Klassen(vgl. 7.3.1) ρ^{Pos} bzw. ρ^{Neg} (vgl. [1]).

$$\rho^{PN} = \frac{\rho^P + \rho^N}{2} \tag{2.3}$$

2.5.3.2 Subtask C

Der Subtask C wird mit dem Macroaveraged Mean Absolute Error (MAE) ausgewertet. Hier ist $\mathcal{C}$ das Set der Klassen(vgl. 7.3.2), Te_j die Anzahl korrekter Dokumente dieser Klasse, $|h(x_i) - y_i|$ der absolute Fehler und das hochgestellte M steht für Macroaveraged (vgl. [1]).

$$MAE^M(h, Te) = \frac{1}{|\mathcal{C}|} \sum_{j=1}^{|\mathcal{C}|} \frac{1}{|Te_j|} \sum_{x_i \in Te_j} |h(x_i) - y_i| \tag{2.4}$$

3 Technologien

Im folgenden Kapitel werden die verwendeten Technologien erläutert.

3.1 Python[1]

Das bestehende System wurde mit Python 2.7 geschrieben. Es wurde aber vor allem aus Gründen der Unicode-Kompatibilität auf Python 3.5 umgestellt (vgl. 6.3).

3.2 Git

Um eine saubere Versionierung des Codes und den Resultaten sicherzustellen, wurde mit Git gearbeitet. Die Ordnerstruktur wurde von Anfang an genau definiert, um die Experimente strukturiert abspeichern zu können. So wird sichergestellt, dass alle Experimente wiederholt werden können und keine wichtigen Daten verloren gehen.

3.3 Gensim[2]

Die benutzte Word2Vec-Implementation stammt von Gensim, ist in Python geschrieben, gut dokumentiert und einfach zu verwenden (vgl. [14]).

3.3.1 Optimierung

Da das Rechnen von Word-Embeddings einen sehr grossen Aufwand bedeutet, wurde darauf geachtet, alle Optimierungsschritte einzuhalten. Die zwei Wichtigsten davon sind (vgl. [15]):

- Verwenden von Basic Linear Algebra Subprograms (BLAS) [16]
- Verwenden von Cython [17]

3.3.2 Parallelisierung

Zu den Performance Optimierungen gehört auch die Parallelisierung. Dazu sollte man Word2Vec mit möglichst vielen Workern starten, um eine hohe Parallelisierung und damit eine Zeitersparnis zu erreichen (vgl. [18]).

[1] 3.5
[2] 0.13.3

3.4 Deep Learning Bibliotheken

In dieser Arbeit wurden dieselben Deep Learning Bibliotheken verwendet wie im bestehenden System.

3.4.1 Theano[3]

Theano ist eine freie Softwarebibliothek für effiziente und multidimensionale numerische Berechnungen. Theano ist optimiert für schnelle Berechnungen auf CPU sowie GPU Architekturen (vgl. [19]).

3.4.2 Keras[4]

Keras ist eine high-level-Softwarebibliothek für Neuronale Netze. Keras ist in Python geschrieben und läuft sowohl mit TensorFlow[20] als auch mit Theano. Keras hat den Fokus darauf gesetzt, das schnelle Definieren und Durchführen von Experimenten zu ermöglichen, was mit TensorFlow und Theano komplizierter ist (vgl. [21]).

[3] 0.8.2
[4] 1.1.2

4 Bestehendes System

Das vom InIT bereitgestellte System wurde anhand des vom SwissCheese-Team entwickelten Systems aufgebaut (vgl. [22]). Das SwissCheese-Team konnte den Subtask A von Task 4 bei SemEval-2016 für sich entscheiden.

Das bereitgestellte System war aber nur teilweise für den Anwendungsfall dieser Arbeit geeignet und musste daher stark umgebaut werden.

SwissCheese wurde für die Sentiment-Analyse für Tweets konzipiert. In SwissCheese werden die Themen der einzelnen Tweets nicht beachtet, sondern lediglich das Sentiment des gesamten Tweets ausgewertet. Das Ziel war es, dieses System mittels Reverse-Engineering zu verstehen und auf themenbasierte Sentiment-Analyse zu optimieren.

4.1 Neuronales Netz

Das verwendete Convolutional Neural Network (CNN) wurde aus dem bestehenden System entnommen und ist in Abbildung 4 vereinfacht dargestellt.

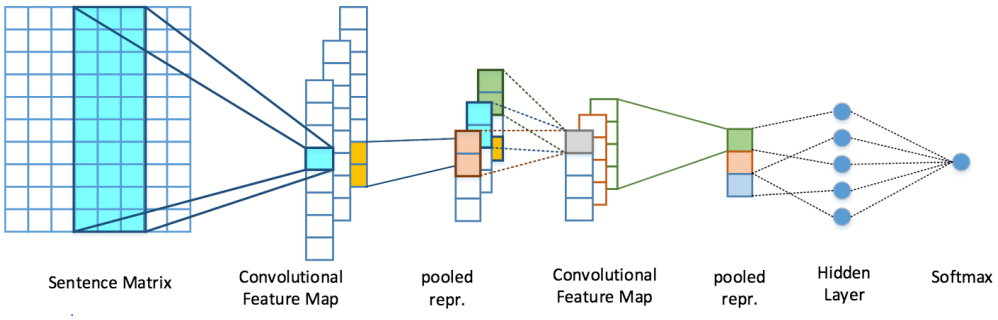

Abbildung 4: Architektur CNN [22]

4.2 Parameter

Die Parameter wurden aus dem bestehenden System übernommen und werden daher nur kurz beschrieben. Falls diese verändert werden, ist dies explizit angegeben.

Das verwendete Convolutional Neural Net (CNN) wird ab hier mit „Modell“ abstrahiert.

4.3 Trainingsphasen

Das bestehende System war grob in drei Phasen aufgeteilt.

4.3.1 Embedding

In dieser Phase werden die Word-Embeddings unsupervised mittels Word2Vec erstellt und trainiert. Dies wird mit einer grossen Datenmenge von ca. 200 Millionen Datensätzen (vgl. 7.1) gemacht, um möglichst viele Wörter im Vokabular aufzunehmen und ein gutes Training der Word-Embeddings zu erreichen. Das Vokabular und die Matrix, welche die Vektoren beinhaltet, werden anschliessend extrahiert und separat abgespeichert. Das Vokabular dient als Lookup-Tabelle für den Index der Wörter.

4.3.1.1 Parameter

Nachfolgend werden die Parameter für das Training mit Gensim Word2Vec aufgelistet.

Minimum Word-Count Ein Wort muss mindestens 15 mal vorkommen, um in das Vokabular aufgenommen zu werden.

Size Die Dimensionalität des Wortvektors wurde auf 52 gesetzt.

4.3.2 Distant Supervised

Ein grosse Problem des Trainings ist, dass nur eine begrenzte Menge gut annotierter Daten für das Training der Supvervised-Phase verfübar ist.

> „A more promising approach, often called „weak“ or „distant“ supervision, creates its own training data by heuristically matching the contents of a database to corresponding text.“ [23]

Aufgrund des Datenmangels, wird zwischen der Unsupervised- und Supervised-Phase noch eine Distant-Supervised-Phase eingeschoben. Diese Phase soll mit ca. 100 Millionen zusätzlichen Datensätzen(vgl. 7.2) eine weitere Verbesserung bringen. Dabei wird während einer Epoche, mit den anhand der Smileys in zwei Klassen (Positiv und Negativ) klassifizierten Daten, trainiert.

4.3.2.1 Parameter

Nachfolgend werden die Parameter für das Training in der Distant-Supervised-Phase mit Keras und dem Backend Theano beschrieben.

Optimizer Als Optimizer für Gradient Descent (vgl. [5]) wurde Adadelta [24] verwendet, welcher eine dynamische Anpassung der Learning-Rate ermöglicht.

Shuffle In jeder Epoche werden die gesamten Trainingsdaten zufällig gemischt. Diese Option ist standardmässig eingeschaltet.

Loss Die Loss-Funktion wurde auf Categorical Crossentropy gesetzt.

Batchsize Die Batchsize ist auf 1500 gesetzt.

Epochs In der Distant-Supervised-Phase wird nur genau eine Epoche gemacht und dadurch ist kein EarlyStopping nötig.

4.3.3 Supervised

In der dritten und letzten Phase wird das System mit den SemEval-Trainingsdaten trainiert und anhand der SemEval-Testdaten validiert. Hier ist die Anzahl Epochen variabel und richtet sich nach dem EarlyStopping-Monitor.

4.3.3.1 Parameter

Nachfolgend werden die Parameter für das Training in der Supervised-Phase mit Keras und dem Backend Theano beschrieben.

Optimizer Als Optimizer für Gradient Descent (vgl. [5]) wurde Adadelta [24] verwendet, welcher eine dynamische Anpassung der Learning-Rate ermöglicht.

Shuffle In jeder Epoche werden die gesamten Trainingsdaten zufällig gemischt. Diese Option ist standardmässig eingeschaltet.

Loss Die Loss-Funktion wurde auf Categorical Crossentropy gesetzt.

EarlyStopping Das Training wird gestoppt, falls der Monitor für die EarlyStopping-Callback-Funktion nach 50 Epochen keine Verbesserung mehr feststellt.

Batchsize Die Batchsize ist auf 1000 gesetzt.

Epochs Die maximale Anzahl Epochen beträgt 1000. Diese wird aber fast nie erreicht, ausser bei Fehlern an der Optimierung im Training (Oszillation des Trainingsverlaufs).

4.4 Baseline

Um die Resultate der neuen Experimente zu vergleichen, musste eine Baseline gelegt werden, auf der aufgebaut werden konnte. Da die Daten für die Embedding- und die Distant-Phase, auf dem das bestehende System aufbaute, nicht mehr zur Verfügung

standen, wurde mit dem bereits trainierten Modell weitergearbeitet. Darauf konnte die Supervised-Phase mit den SemEval-Daten (7.3) aufgesetzt werden.

Eine klare Voraussetzung nach dem Umbau war die Erhaltung des aktuellen Resultats. Bei Erreichung dieser Voraussetzung mit dem neuen System, wird dieses Resultat als ´"Baseline“ bezeichnet. Die Resultate des Gewinners von Subtask B „Tweester“ und Subtask C „TwiSE“ von SemEval-2016 und dem bestehendem System werden zum Vergleich aufgelistet (vgl. [1]).

System	**Subtask**	**Metrik**	**Score**
Bestehendes System	B	Macroaveraged Recall	0.8044
Tweester	B	Macroaveraged Recall	0.7970
Bestehendes System	C	MAE	0.6938
TwiSE	C	MAE	0.7190

Tabelle 3: Resultate Bestehendes System und Gewinner SemEval-2016.

5 Optimierungen

Im Verlauf dieser Arbeit wurden die Scores innerhalb der folgenden zwei Phasen optimiert.

5.1 Embedding-Phase

In der Embedding-Phase konnte die Laufzeit verbessert werden, indem die Embeddings trainiert und das Vokabular direkt extrahiert wurde. Die Optimierung gelang durch eine Funktion der word2vec-Implementation, welche bisher nicht verwendet wurde. Diese lässt es zu, die Embeddings zu trainieren und danach direkt das Vokabular daraus zu extrahieren oder zu durchlaufen. Bisher wurden, für das Vokabular und das Training Daten komplett durchlaufen. Die Datenmenge beläuft sich auf ca. 200 Millionen Tweets(7.1). Leider sind keine genauen Zahlen für einen Vergleich vorhanden. Es kann aber aus der Anzahl Loops und der grossen Datenmenge geschlossen werden, dass die optimierte Variante beträchtlich schneller läuft.

5.2 Supervised-Phase

Im bestehenden System war der Monitor für das EarlyStopping des Subtasks B auf den F1-Score gesetzt. Dieser ist aber die Evaluationsmetrik für Subtask A. Dadurch wurde der ModelCheckpoint, welcher den besten Stand des Modells abspeichert, zum falschen Zeitpunkt aufgerufen. Mit der Änderung dieses Monitors auf den Macroaveraged Recall konnte der Score im besten Fall um drei Punkte verbessert werden.

6 Architektur

Die bestehende Architektur des Systems war kompliziert und schwerfällig gestaltet. Eine Dokumentation war nicht vorhanden und die Struktur konnte nur schwer erkannt werden.

Aufgrund dieser Umstände wurde das komplette System bis auf die Evaluationsmetriken umgeschrieben und neu aufgebaut. Dies brachte den Vorteil, eines klaren, kommentierten und effizienten Codes, welcher die Vorgänge besser nachvollziehbar und somit verständlicher machte.

6.1 Grundlagen

Im folgenden Abschnitt werden die grundlegenden Veränderungen des Architektur erläutert.

6.1.1 Naming

Die Variablen, Funktionsnamen und Imports wurden in einem einheitlichen Stil umbenannt. Dabei wurde speziell auf die Einhaltung der „Pythonic“ Idiomen und Best-Practices geachtet (vgl. [25]).

6.1.2 Verzeichnisstruktur

Die Verzeichnisstruktur wurde so aufgebaut, dass pro Experiment Unterordner zur jeweiligen Kategorie (Model, Resultate, Log, Embeddings) erstellt und die Dateien mit einem einheitlichen Namen über alle Experimente hinweg benannt wurden (vgl. A.2).

6.1.3 Dokumentation

Für jede Funktion, Methode und Klasse wurden Python-Docstrings mit Beschreibung, Parametern und Returnwerten erstellt. Zudem wurden vereinzelt Blockkommentare für die Klarheit und Strukturierung in den Skripts verwendet.

6.2 Konfiguration

Die Konfiguration wurde so aufgebaut, dass möglichst einfach neue Experimente definiert werden können. Bereits durchgeführte Experimente können nochmals gestartet werden. Der Ordnername kann frei gewählt werden, sollte aber der Einfachheit halber dem Experimentnamen entsprechen.

6.2.1 Aufteilung

Die Konfigurationen wurden nach Embedding-, Distant- und Supervised-Phase aufgeteilt. Dadurch kann jede Phase einzeln gestartet und die Parameter jeweils definiert werden.

6.2.2 Definition

Die Konfigurationsdateien werden in JSON-Dateien spezifiziert. Es besteht eine Standardkonfiguration, welche überschrieben wird. Dadurch müssen über mehrere Experimente hinweg beständige Attribute nicht immer definiert werden und die Redundanz wird minimiert.

6.2.3 Abhängigkeiten

Jede Phase ist von den Resultaten der vorherigen Phase abhängig. Dies führt zu einem gewissen Aufwand beim Definieren und es muss bei der Erstellung von Experimenten auf die korrekte Angabe der abhängigen Dateien geachtet werden. Dieser Ansatz wurde aber bewusst gewählt, da dadurch für verschiedene Experimente eine Abhängigkeit definiert werden kann. So können zum Beispiel die Embeddings in Experiment "A" erstellt und aus Experiment "B" referenziert werden, da es unnötig ist, die Embeddings mit der gleichen Konfiguration nochmals zu berechnen. Diese Abhängigkeiten funktionieren für die folgenden Konfigurationselemente:

- Model-Weights
- Model-Definition
- Embedding-Matrix
- Embedding-Vokabular

6.3 Unicode

Die entwickelte Applikation wurde mit der Umstellung auf Python 3.5 zusätzlich noch auf korrektes Unicode-Handling umgestellt, da im bestehenden System alles ausser ASCII ignoriert wurde.

Für diesen Ansatz wurde das Konzept "Unicode sandwich" verwendet.

> „Keep all text in your program as Unicode, and convert as close to the edges as possible." [26]

Zusätzlich zum Umsetzen dieser Best-Practice werden durch das Verwenden von Unicode Emojis und allen zusätzlichen Zeichen zu ASCII erkannt und das System erlaubt es somit auch Sprachen mit einem erweiterten Zeichensatz wie Deutsch oder Chinesisch zu analysieren.

6.4 Reader

Eine zentrale Komponente für die Datenverarbeitung sind die Readerklassen, welche von einer Quelle lesen und in die gewünschte, einheitliche Form bringen. In diesem Fall wird das mit der Klasse Tweet erreicht. Ausserhalb der Reader-Klassen soll die Art der Quelle keine Rolle spielen, dies wird mit der von den Reader-Klassen erzeugten Instanzen der Klasse Tweet erreicht werden.

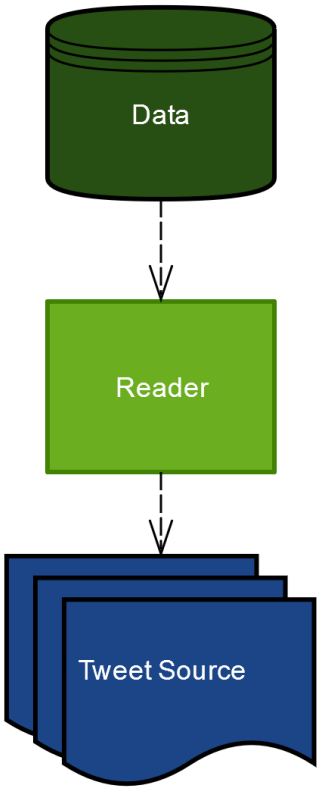

Abbildung 5: Ablauf Reader

6.4.1 Klasse Tweet

Die Klasse Tweet modelliert und beinhaltet alle Informationen eines Tweets plus die Daten in der Form, welche für das Training gebraucht werden:

- Tokens
- Sample

Durch das Verwenden einer Klasse kann im ganzen System konsistent die gleiche Struktur verwendet werden. Auf dem Tweet müssen lediglich die ID und der Text gesetzt werden.

6.4.2 Abstrakte Basisklasse

Da es für jedes Format oder Quelle einen eigenen Reader braucht, wurde die Readerklasse selbst als abstrakte Basisklasse (ABCMeta-Klasse) definiert und alle spezifischen Reader davon abgeleitet. Dadurch besteht eine Vorgabe, wie ein Reader auszusehen hat und es erleichtert das Hinzufügen von neuen Implementierungen. Ausserdem muss man sich ausserhalb des Readers nicht um dessen Implementierung kümmern. Dies ist bei der Verwendung der Pipeline ein grosser Vorteil.

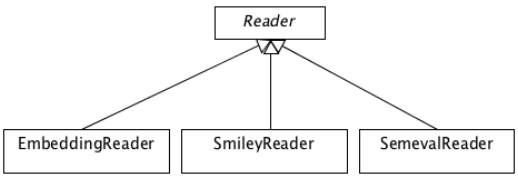

Abbildung 6: Reader Klassendiagramm

6.4.3 Implementation

Die abstrakte Klasse spezifiziert drei Attribute und eine Methode, welche implementiert werden müssen.

- Methode: *read()*
- Attribut: *source*
- Attribut: *config*
- Attribut: *meta_ info*

Nachfolgend werden nur *read* und *meta_ info* genauer betrachtet, da *source* und *config* einfache Attribute sind.

6.4.3.1 Methode: *read*

Die Read-Funktion öffnet und liest von der Quelle und liefert mithilfe der yield-Anweisung ein Tweet-Objekt. Dadurch wird die Read-Methode zu einem Generator von Tweet-Objekten und ermöglicht so effizientes Lesen der Quelle. Zu erwähnen ist hier nochmals, dass der Reader nicht zwingend von einer Datei lesen muss, sondern zum Beispiel auch von einem Netzwerkstream Daten lesen und Tweets erstellen kann. Ausserhalb des Readers wird lediglich ein Generator von Tweets erwartet.

6.4.3.2 Methode: *meta_ info*

Das Attribut *meta_ info* beinhaltet Informationen über den Reader selbst. Im Falle von unserem System wären das die Anzahl der Zeilen oder Tweets und zusätzlich für den Klassenausgleich wichtige Daten wie die Anzahl Tweets pro Labelklasse. Da bei einer Quelle mit grossen Datenmengen wie zum Beispiel einer Datei das Zählen der Anzahl

Datensätze einen grossen Aufwand bedeutet, wurde der Ansatz des Lazy Loadings für dieses Attribut gewählt.

Diese Klassenverteilung wird in einem sogenannten Counter gespeichert. Dadurch können genaue Analysen und arithmetische Operationen von Metainformationen zwischen mehreren Readern ohne Aufwand ausgeführt werden.

```
86 test_n_samples = sum(source.meta_info['n_samples'] for source in
   ↪ pipeline_test.sources)
```

Listing 1: Beispiel für das Zusammenführen der Anzahl Samples von mehreren Readern

6.4.4 Factory

Zusätzlich zur abstrakten Klasse wurde eine Factory für das Erstellen der Reader verwendet. Dadurch kann in der Konfiguration zu jeder Quelle ein Readertyp spezifiziert und anschliessend über die Factory der passende Reader erstellt werden.

6.5 Pipeline

Eine Zentrale Komponente des Systems ist die Pipeline, welche das ganze Datenhandling und Preprocessing übernimmt und modular gestaltet. Der komplette Ablauf der Pipeline ist in Abbildung 7 aufgezeigt.

6.5.1 Idee

Mithilfe einer Pipeline können grosse Datenmengen einfach verarbeitet werden und mit dem Speicher wird zudem sparsam umgegangen. Ausserdem macht es den Code sehr verständlich, da immer mit einem vergleichsweise kleinen Subset der Daten und mit einer dadurch definierten Funktion gearbeitet werden kann.

Für die Grundimplementation und die Grundfunktionalität wird auf einen Blogeintrag verwiesen (vgl. [27]).

Die Idee war es nun, diese Pipeline auf die Bedürfnisse anzupassen und möglichst erweiterbar zu machen. Die Anforderungen waren wie folgt:

- Es müssen statische Referenzen mitgegeben werden können.
- Die Pipeline muss sich auch mit mehreren Datenquellen gleich verhalten.
- Die Pipeline soll zusätzlich iterierbar sein.
- Es müssen Batches von Daten erstellt werden können.
- Die Pipeline soll möglichst generell gehalten werden das heisst nicht nur für diese spezifische Anwendung eingesetzt werden können.
- Die Pipeline soll einfach verwendet sowie erweitert werden können.

6.5.2 Datenquellen

Die Pipeline kann die Daten von mehreren Quellen beziehen, welche normalerweise von einem Reader geliefert werden. Sie ist so aufgebaut, dass die Anzahl Quellen keine Rolle spielt. Die Quellen werden beim Aufbau der Pipeline aneinander gehängt und die Prozessorfunktionen entsprechend angewendet.

6.5.3 Prozessoren

Ein Prozessor ist eine Funktion, welche den Input prozessiert und ihn in einer definierten Form weiterreicht. Dabei kann ein Prozessor auf die statischen Referenzen der Pipeline zugreifen. Zu beachten gilt es, dass der Output eines Prozessors mit dem Input des nächsten übereinstimmen muss. Dies wurde im aktuellen System über das Tweet-Objekt einfach gelöst. Der erste Prozessor in der Kette ist per Definition die Datenquelle. Der letzte Prozessor der Pipeline gibt das Outputformat der Pipeline an. Dabei ist es empfehlenswert, einen zusätzlichen Prozessor zu definieren, welcher das Outputformat definiert.

```
5  def printer(tweets, statics):
6      '''
7      Print a tweet object and yield
8
9      Args:
10         tweets: tweet generator
11         statics: pipeline statics
12     Yields:
13         Tweets
14     '''
15     for tweet in tweets:
16         print(tweet)
17         yield tweet
```

Listing 2: Beispiel einer Prozessorfunktion, welche jeweils den Tweet ausgibt

Wie in Listing 2 dargestellt, kann z.B. eine Printer-Prozessorfunktion in die Pipeline eingefügt werden.

6.5.4 Statics

Die statischen Referenzen der Pipeline sind in einem Dictionary organisiert und werden jeder Prozessorfunktion als Parameter zur Verfügung gestellt. Da die Referenz auf die jeweiligen Objekte übergeben wird, spart die Pipeline wiederum an Speicher und gewinnt an Geschwindigkeit da zum Beispiel aufwendige Objekte nur zu Beginn erstellt werden müssen. Im Falle des aktuellen Systems wären das Vokabular und der Tokenizer statische Referenzen.

Die Statics können zwar in den Prozessoren verändert werden, es wird aber davon abgeraten.

```
65 def labeler(tweets, statics):
66     '''
67     Convert Tweet label and yield
68     Generates the value for keras Embedding Layer Input y
69 
70     Args:
71         tweets: tweet generator
72         statics: pipeline statics(config)
73     Yields:
74         Tweets
75     '''
76     config = statics['config']
77     n_labels = config['n_labels']
78 
79     for tweet in tweets:
80         tweet.y = get_label(tweet.label, n_labels)
81         yield tweet
```

Listing 3: Beispiel einer Prozessorfunktion, welche auf die Statische Referenz der Konfiguration zugreift

6.5.5 Implementation

Die Pipeline ist wiederum als Klasse modelliert und liegt in einem eigenen Modul.

Nachfolgend werden die zur Verfügung gestellten Methoden aufgelistet. Die komplette Implementation der Pipeline ist dem Anhang zu entnehmen.

6.5.5.1 Methode: *add_static*

Diese Methode fügt dem Dictionary von statischen Referenzen eine weitere hinzu. Der Name der Referenz ist dabei der Schlüssel.

6.5.5.2 Methode: *add_processor*

Diese Methode fügt der Liste von Prozessoren eine Prozessorfunktion an.

6.5.5.3 Methode: *generate*

Listing 4 zeigt, wie die Prozessoren in der richtigen Reihenfolge über jede Quelle verschachtelt und aneinadergehängt werden. Der Rückgabewert ist wiederum ein Generator, welcher alle Prozessoren über alle Quellen beinhaltet.

```
48      def generate(self):
49          '''
50          Generate from source and apply all processors
51
52          Yields:
53              Generated samples from sources
54          '''
55          pipelines = []
56          for source in self.sources:
57              pipeline = source.read()
58              for processor in self.processors:
59                  pipeline = processor(pipeline, self.statics)
60
61              pipelines.append(pipeline)
62          return chain.from_iterable(pipelines)
```

Listing 4: Pipeline Methode: *generate*

6.5.5.4 Methode: *generate_batch*

Bei dieser Methode werden die von der generate-Methode verarbeiteten Tweets anhand der Batchgrösse gesammelt und dann als Batch weitergereicht. Zu erwähnen ist, dass der gesamte Batch mit einer Permutationsfunktion noch verändert werden kann. Die Idee hinter der Permutationsfunktion ist, dass der Batch, nachdem er die ganze Pipeline durchlaufen hat, noch weiter verändert werden kann. Dies kann zum Beispiel für das Sortieren oder Entzippen nützlich sein (entzippen im Falle von Python Tupels). Das Entzippen war speziell wichtig, da Keras die Daten für die Fit-Funktion in einem speziellen Format erwartet.

6.5.5.5 Methode: *generate_batch_inf*

Da Keras einen unendlichen Generator erwartet, wurde die Pipeline zusätzlich um diese Funktion erweitert. Das ist aber mit Vorsicht zu geniessen, da der Generator manuell abgebrochen werden muss. Vorzugsweise sollte die Iteratorfunktionalität verwendet werden.

6.5.5.6 Methode: *__iter__*

Da ein Generator nicht neu gestartet werden kann, musste die generate Methode in der *__iter__* Methode verpackt werden.

Nun ist die Pipeline durch diesen einfachen Trick auch iterierbar, was viele Vorteile bei der Verwendung mit zum Beispiel Word2Vec schafft.

6.5.6 Verwendung

Die Verwendung besteht aus dem Erstellen der Pipeline mit der Liste aus Quellen/Readern und einer optionalen Konfiguration direkt zu den Statics angehängt wird. Danach können die Statischen Referenzen und die Prozessoren hinzugefügt werden. Dies muss zwingend vor dem ersten Generieren von Outputs gemacht werden. Danach kann die Pipeline durch Ansprechen der Generatoren oder des Iterators verwendet werden.

```
42 # Training Pipeline
43 pipeline_train = Pipeline(readers_train, config)
44 pipeline_train.add_static('vocab', vocabulary)
45 pipeline_train.add_static('tokenizer', tweet_tokenizer)
46
47 pipeline_train.add_processor(escaper)
48 pipeline_train.add_processor(tokenizer)
49 pipeline_train.add_processor(labeler)
50 pipeline_train.add_processor(embedder)
51 pipeline_train.add_processor(fit_sampler)
```

Listing 5: Konfiguration der Training-Pipeline für das Preprocessing der Supervised-Phase

6.5.7 Ausblick

Mit der Pipeline wurde ein sehr dynamisches und generisches Modul geschaffen, welches unabhängig von der Anwendung verwendet werden kann. Die Implementation ist sehr einfach und die Logik wurde auf die Prozessoren ausgelagert. Die Prozessoren könnten aber im Falle des aktuellen Systems noch so gestaltet werden, dass sie auch auf einem Set von Tweets funktionieren und dadurch schon auf der Ebene der Reader ein Batch weitergereicht werden kann. Dies könnte die Geschwindigkeit aufgrund der Lokalität des Zugriffs verbessern.

6.6 Logging

Das Logging geschieht pro Phase und die Logdateien werden von der Versionskontrolle erfasst, da sie ein wichtiger Bestandteil der Analyse sind. Zusätzlich besteht die Möglichkeit, der Pipeline einen Prozessor zur Verfügung zu stellen, welcher den Inhalt zur Konsole umleitet. Dadurch wird das Debugging extrem vereinfacht.

6.7 Runscript

Um Experimente durchzuführen, wird pro Experiment ein Bash-Script erstellt, welches die Zeitmessung der Phase mit dem Unix-Befehl *time* übernimmt, den Ordner für die

Logdateien erstellt und die gewünschten Scripts mit der jeweiligen Konfiguration ausführt. Durch die Modularität des Skripts und der Konfigurationen kann eine abgebrochene Phase neu gestartet werden und es muss nicht die gesamte Kette nochmals ausgeführt werden.

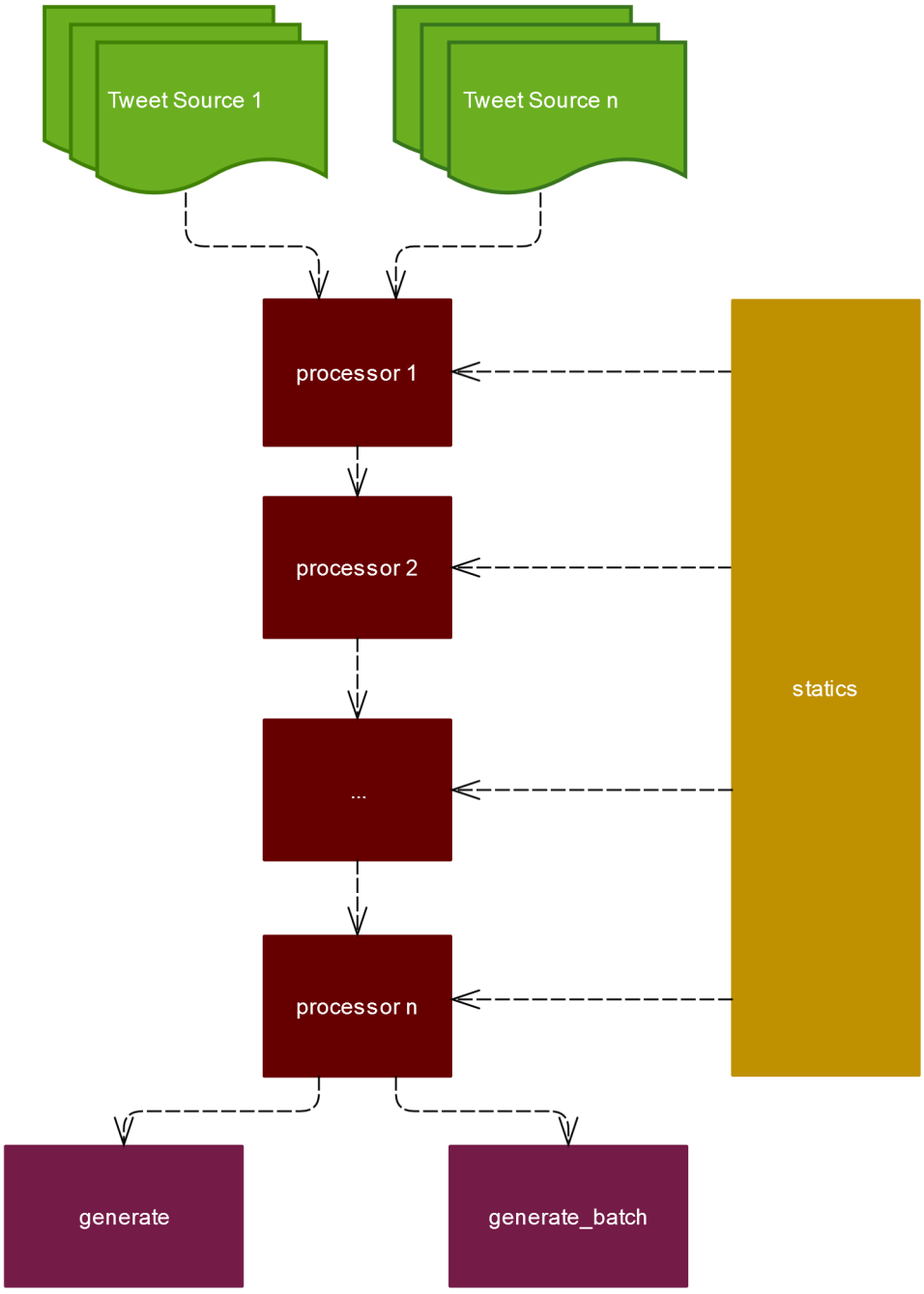

Abbildung 7: Ablauf Pipeline

7 Daten

Das ganze System basiert auf Tweets. Die Länge der Texte ist dadurch auf 140 Zeichen begrenzt. Alle Daten werden UTF-8 kodiert ausgeliefert, was dem Twitter Standard entspricht (vgl. [28]).

7.1 Embeddings

Die Embeddings und das Vokabular wurden aus ca. 200 Millionen Tweets generiert. Diese Tweets stammen aus einer internen Sammlung des Instituts und beinhalten unter anderem Emojis und diverse Sonderzeichen. Bezüglich den Themen der Tweets gibt es keine Angaben.

7.1.1 Format

Nachfolgend wird das TSV-Dateiformat für die Embedding-Tweets beschrieben.

Index	Wert
0	Language-Code
1	Tweet-ID
2	Timestamp
3	Tweet-Text
4	Twitter-Username
5	Tweet-Text

Tabelle 4: Embedding-Tweets TSV-Format

7.2 Smiley-Tweets

Für die Distant-Phase wurden ca. 100 Millionen anhand einer definierten Liste von Smiley annotierten Tweets verwendet. Diese Tweets stammen wiederum aus einer internen Sammlung des Instituts. Die Smileys, die für die Annotation verwendet wurden, wurden anschliessend aus den Tweets gelöscht, um die Resultate nicht zu verfälschen. Die Tweets beinhalten unter anderem Emojis und diverse Sonderzeichen. Bei der Annotierung wurde nur zwischen Positiv und Negativ unterschieden. Bezüglich den Themen der Tweets gibt es hier wiederum keine Angaben. Die Verteilung dieser beiden Klassen ist wie folgt:

Klasse	Anzahl
Positiv	79'113'088
Negativ	20'896'082
Total	**100'009'170**

Tabelle 5: Klassenverteilung Smiley-Tweets

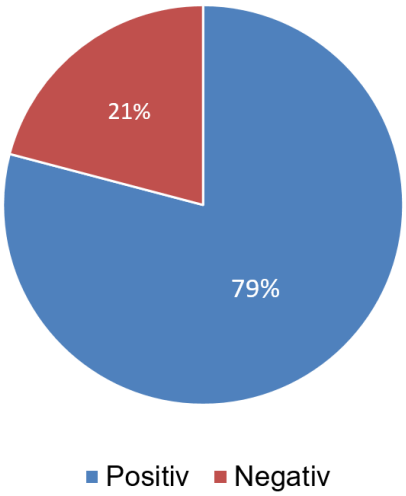

Abbildung 8: Klassenverteilung Smiley-Tweets

7.2.1 Format

Nachfolgend wird das TSV-Dateiformat für die Smiley-Tweets beschrieben.

Index	Wert
0	Tweet-ID
1	Language-Code
2	Label
3	Tweet-Text

Tabelle 6: Smiley-Tweets TSV-Format

7.3 SemEval

Die Supervised-Phase wurde mit den von SemEval bereitgestellten und für 2017 neu aufbereiteten Daten von SemEval-2016 durchgeführt (vgl. [29]). Für das Training wurden die jeweiligen Trainingssets für Subtask B und C verwendet. Die Tweets sind zwar in UTF-8 kodiert, beinhalten aber nur ASCII Zeichen. Die Themen der Tweets sind alphabetisch sortiert.

Nachfolgend werden nur die Trainingsdaten beschrieben, da über die nicht verfügbaren Testdaten des aktuellen SemEval-2017 keine Aussage gemacht werden kann.

7.3.1 Subtask B

Subtask B ist in zwei Klassen aufgeteilt(positiv, negativ).

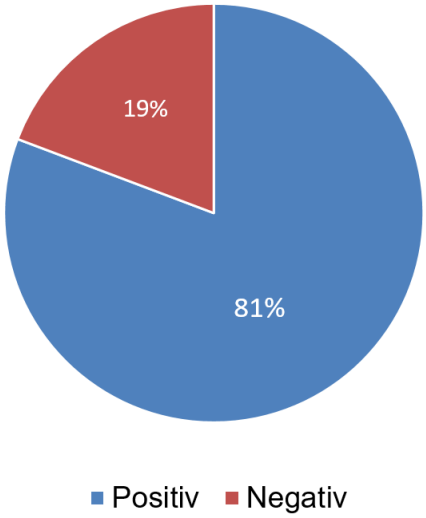

Abbildung 9: Klassenverteilung Subtask B

Label	Wert
positive	Positive
negative	Negativ

Tabelle 7: Klassen Subtask B

Klasse	Anzahl
Positiv	5088
Negativ	1213
Total	**6301**

Tabelle 8: Klassenverteilung Subtask B

7.3.2 Subtask C

Subtask C ist in fünf Klassen aufgeteilt(-2, -1, 0, 1, 2).

Label	Wert
-2	VeryPositive
-1	Positive
0	OK
1	Negative
2	VeryNegative

Tabelle 9: Klassen Subtask C

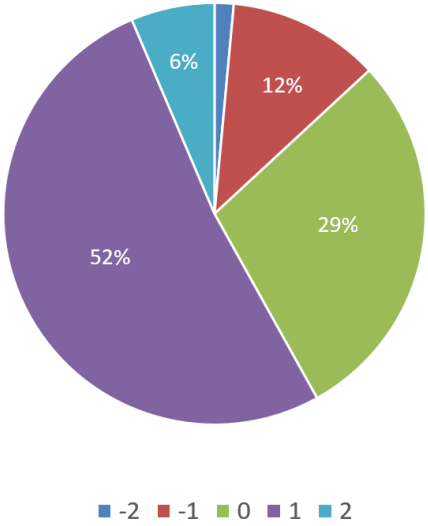

Abbildung 10: Klassenverteilung Subtask C

Klasse	Anzahl
-2	123
-1	978
0	2422
1	4342
2	538
Total	**8403**

Tabelle 10: Klassenverteilung Subtask C

7.3.3 Format

Nachfolgend wird das SemEval-2016 TSV-Dateiformat beschrieben.

Index	Wert
0	Tweet-ID
1	Thema
2	Label
3	Tweet-Text

Tabelle 11: SemEval-2016-Tweets TSV-Format

7.4 Themenspezifische Tweets

Da die vom Institut bereitgestellten Daten aus dem Jahr 2013 stammen, sind aktuelle Themen nicht in den Daten zu finden. Es ist aber damit zu rechnen, dass die Testdaten für SemEval-2017 aktuelle Themen beinhalten. Um sicherzustellen, dass diese in den

Word-Embeddings vorhanden sind, wurde im Verlauf der Arbeit ein Twitter-Crawler geschrieben.

7.4.1 Twitter-Crawler

Um aktuelle Themen in den Word-Embeddings zu berücksichtigen, werden pro Thema jeweils 200'000 zusätzliche Tweets heruntergeladen. So wird das Vokabular mit aktuellen Wörtern ergänzt. Dadurch sollen die Word-Embeddings noch spezifischer trainiert werden.

7.4.1.1 Technischer Ablauf

Twitter API Twitter bietet eine RESTful-Schnittstelle, welche erlaubt, Daten von Twitter zu lesen und abzuspeichern. An diese Schnittstelle können Suchanfragen gesendet werden und die API liefert das Resultat im JSON-Format zurück. Um die Schnittstelle zu benutzen, wird ein Twitter-Developer-Account benötigt. Leider ist die Anzahl der Anfragen pro Twitter-Account limitiert, um die Schnittstelle nicht zu überlasten. Pro Twitter-Account können 180 Anfragen pro 30 Minuten abgesetzt werden. Pro Anfrage werden maximal 100 Tweets geliefert. Im besten Fall können demnach pro Tag maximal 864'000 Tweets heruntergeladen werden. In dem Datenset von SemEval sind 100 Themen. Dies Bedeutet es werden 20'000'000 Tweets benötigt. Da SemEval die relevanten Daten erst kurz vor den Wettbewerb freigibt, werden mehrere Twitter-Accounts benötigt um diese Anzahl Tweets kurzfristig herunterzuladen.

Ablauf Twitter-Crawler

1. Themen aus dem Test-Datenset von SemEval werden extrahiert.
2. Themen werden auf die Twitter-Crawler aufgeteilt.
3. Crawlen der 200'000 Tweets für das jeweilige Thema und exportieren in TSV-Datei

Suchanfrage Parameter In der folgenden Tabelle werden die einzelnen Parameter der Anfrage aufgelistet.

Parameter	Beschreibung	Wert
q	Suchbegriff	Thema
lang	Sprache	en
result_type	Resultattyp	recent
count	Anzahl Tweets (Maximal 100)	100

Tabelle 12: Parameter Twitter-Crawler

8 Experimente

Im Verlauf der Arbeit werden diverse Ideen gesammelt, wie das bestehende System verbessert werden kann. Die Ideen werden diskutiert und ausgewertet. Die relevanten Ideen sind als Experimente formuliert und wurden durchgeführt.

Nummer	Name
1	Baseline
2	Random-Seed
3	Batchgrössen
4	Shuffle
5	Class-Weights
6	Unknown-Words
7	Themenspezifische Tweets

Tabelle 13: Experimente

Um die Experimente auszuwerten, wird jeweils das Test-Datenset von SemEval-2016 verwendet.

8.1 Experiment 1: Baseline

Es wird überprüft, ob das überarbeitete System(TopicThunder) die Genauigkeit des bisherigen Systems erreicht. Die Annahme besteht darin, dass mit dem überarbeiteten System der Score für beide Subtasks um mehrere Punkte verbessert wird.

Für das bestehende System stehen die ursprünglichen Trainingsdaten für die Embedding- und Distant-Phase nicht mehr zur Verfügung. Es konnte aber mit dem bereits erstellten Modell die Supervised-Phase auf den SemEval-2016 Trainingsdaten für den jeweiligen Task ausgeführt werden.

Für das überarbeitete System sind die Embedding- und Distant-Phase anhand den neuen und vom Institut bereitgestellten Daten ausgeführt worden. Danach wurde analog zum bestehenden System die Supervised-Phase mit denselben Daten ausgeführt.

8.1.1 Beobachtung

8.1.1.1 Subtask B

System	Recall
Bestehendes System	0.8044
ThopicThunder	0.8319

Tabelle 14: Experiment Baseline Subtask B

8.1.1.2 Subtask C

System	MAE
Bestehendes System	0.6938
ThopicThunder	0.7056

Tabelle 15: Experiment Baseline Subtask C

8.1.2 Auswertung

Die Genauigkeit des bisherigen Systems konnte erreicht werden. Im Subtask B wurde der Score um 3 Prozentpunkte gesteigert. Diese Verbesserung konnte durch die Optimierung des EarlyStopping-Monitors erzielt werden (vgl. 5.2). In Subtask C zeigte sich keine Veränderung, es konnte lediglich das bisherige Resultat erreicht werden.

8.2 Experiment 2: Random-Seed

Dieses Experiment überprüft, ob alle Berechnungen bei Mehrfachausführung identische Resultate erzielen. Dies kann durch die Fixierung des Random-Seeds erreicht werden. Durch diese Fixierung muss die Abweichung im Score beim mehrmaligem Durchlaufen des gleichen Experiments eine Differenz von Null vorweisen.

Bisher wurde der Random-Seed automatisch für jeden Lauf neu initialisiert. So konnten die gleichen Experimente mehrmals ausgeführt werden. Die Scores wiesen jedoch immer eine Differenz von ein bis zwei Prozentpunkten auf. Damit die Zahlen reproduzierbar sind, wird der Random-Seed nun im Konfigurationsfile definiert und im Code festgesetzt. Es wird viermal dasselbe Experiment durchgeführt. Zweimal mit einem zufälligen Random-Seed (Random-Seed False) und zweimal mit einem fixierten Random-Seed (Random-Seed True).

8.2.1 Beobachtung

8.2.1.1 No Random-Seed

Lauf	Random-Seed	Recall
1	False	0.8213
2	False	0.8328

Tabelle 16: Experiment No Random-Seed

8.2.1.2 Random-Seed

Lauf	Random-Seed	Recall
1	True	0.8027
2	True	0.8031

Tabelle 17: Experiment Random-Seed

8.2.2 Auswertung

Die Tabelle 16 zeigt einen klaren Unterschied zwischen den beiden Experimenten. Der erste Score weist eine Differenz von über einem Prozentpunkt auf. Leider weisen auch die weiteren Experimente mit dem fixierten Random-Seed eine Differenz von 0.1 Prozentpunkt im Score auf. Die Annahme, dass der identische Score erreicht werden kann, konnte nicht bestätigt werden. Nach weiteren Recherchen wurde ein GitHub-Issue von Theano gefunden, welches beschreibt (vgl. [30]), dass die Convolution-Operation auf der GPU nicht deterministisch implementiert ist. Dadurch bringt auch die Fixierung des Random-Seeds von NumPy, wie im Issue-Kommentar von Keras beschrieben, nichts (vgl. [31]). Die Resultate sind dadurch ohne zu Mitteln nicht vergleichbar, da die Varianz zu gross ist.

8.3 Experiment 3: Batchgrössen

Dieses Experiment evaluiert, mit welcher Batchgrösse der optimale Score erzielt wird. Wenn die Batchgrösse zu klein oder zu gross gewählt wird, werden die Resultate schlechter ausfallen, da keine Generalisierung stattfinden kann. Bisher wurde mit der Batchsize 1000 gearbeitet. Mit dieser Grösse wurden laut Information des Instituts bisher die besten Resultate erzielt. Um diese Information zu belegen oder ein neues Optimum zu finden, wurde das System mit zehn verschiedenen Batchgrössen ausgeführt. Die Batchgrössen hatten jeweils einen Abstand von 200 und startete bei eins. Es wurden also Batchgrössen von eins bis 2000 ausgewertet. Diese wurden verglichen und die beste Batchgrösse ermittelt.

8.3.1 Beobachtung

8.3.1.1 Subtask B

Batchgrösse	Recall
1	0.7318
200	0.8129
400	0.8111
600	0.8264
800	0.8097
1000	0.8287
1200	0.8354
1400	0.8179
1600	0.8195
1800	0.8303
2000	0.8258

Tabelle 18: Experiment Batchgrösse Subtask B

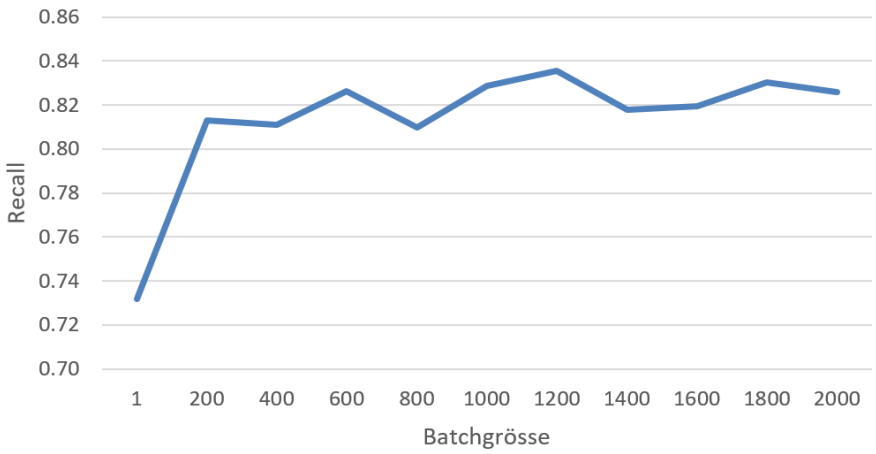

Abbildung 11: Verlauf Batchgrössen Subtask B

8.3.1.2 Subtask C

Batchgrösse	MAE
1	1.0922
200	0.7416
400	0.7216
600	0.7196
800	0.6844
1000	0.6964
1200	0.6972
1400	0.7073
1600	0.7076
1800	0.7074
2000	0.7110

Tabelle 19: Experiment Batchgrösse Subtask C

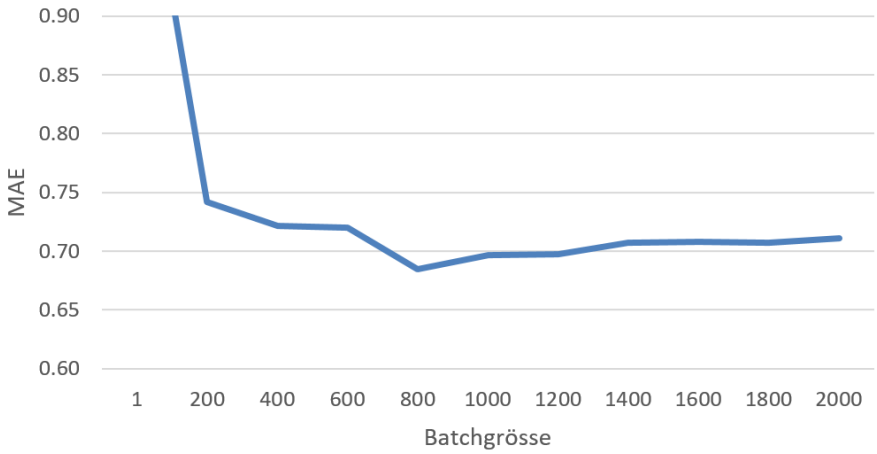

Abbildung 12: Verlauf Batchgrössen Subtask C

8.3.2 Auswertung

Subtask B In Abbildung 11 ist deutlich erkennbar, dass bei einer Batchsize zwischen 1000 und 1400 die besten Scores erreicht werden.

Subtask C In Abbildung 12 ist deutlich erkennbar, dass bei einer Batchsize zwischen 600 und 1000 die besten Scores erreicht werden.

8.4 Experiment 4: Shuffle

Dieses Experiment überprüft die Auswirkung der internen Reihenfolge der Daten in den Batches oder Epochen. Die Reihenfolge der Daten spielt eine grosse Rolle, da sonst

die Reihenfolge über die Epochen hinweg mittrainiert und dadurch die Fähigkeit der Generalisierung auf neue Daten vermindert wird.

Es werden drei verschiedene Ordnungen getestet:

- Belassen der Reihenfolge
- Randomisierte Reihenfolge auf Epochenebene
- Randomisierte Reihenfolge auf Batchebene

8.4.1 Beobachtung

8.4.1.1 Subtask B

Shuffle	Recall
None	0.7682
Epoch	0.8171
Batch	0.8257

Tabelle 20: Experiment Shuffle Subtask B

8.4.1.2 Subtask C

Shuffle	MAE
None	0.7937
Epoch	0.7030
Batch	0.7696

Tabelle 21: Experiment Shuffle Subtask C

8.4.2 Auswertung

Die Beobachtungen bestätigen, dass ohne Veränderung der Reihenfolge ein schlechteres Resultat erzielt wird. Signifikant ist der Unterschied zwischen Batch- und Epoch-Shuffle bei Subtask C. Das Resultat zeigt klar, dass die randomsierte Reihenfolge auf Epochenebene die besten Scores erzielt. Bei Subtask B ist es möglich, dass bezüglich des Problems mit dem Nichtdeterminismus (vgl. 8.2) ein Spezialfall aufgetreten ist. Dies konnte aber nicht bestätigt werden.

Eine Erklärung für den klaren Unterschied bei Subtask C im Gegensatz zu Subtask B könnte jedoch der Unterschied in der Anzahl Trainingsdaten sein (vgl. 7.3), welcher ca. 2000 beträgt. Dadurch ist bei Subtask C der Unterschied klarer erkennbar, da zwei Batches mehr, mit der Grösse 1000 benötigt werden. Auf Epochenebene kann bei einer grösseren Anzahl zudem auch besser gemischt werden.

Diese Annahmen müssen aber mit weiteren Experimenten überprüft werden.

8.5 Experiment 5: Class-Weights

Dieses Experiment überprüft, ob durch Class-Weights bei beiden Subtasks ein höherer Score erreicht wird. Da die Daten eine sehr ungleiche Verteilung der einzelnen Klassen aufweisen, sollte mit den Class-Weights eine Verbesserung im Score erreicht werden können.

Um die Klassengewichte zu berechnen, wird für alle Klassen die folgende Formel verwendet (vgl. [32]), wobei *samples* die Anzahl Datensätze, *classes* die Anzahl Klassen und $freq$ die Klassenfrequenz ist.

$$\frac{samples}{classes * freq} \tag{8.1}$$

Formel 8.1 gewichtet die Klassen invers proportional zu den Klassenhäufigkeiten.

8.5.1 Beobachtung

8.5.1.1 Subtask B

Experiment	Class-Weights	Recall
1	False	0.8225
2	True	0.8403

Tabelle 22: Experiment Class-Weights Subtask B

8.5.1.2 Subtask C

Experiment	Class-Weights	MAE
1	False	0.7088
2	True	0.6331

Tabelle 23: Experiment Class-Weights Subtask C

8.5.2 Auswertung

Die Verwendung von Class-Weights zeigt in beiden Subtasks eine deutliche Verbesserung der Scores. Da im Subtask C die Klassen noch unausgeglichener sind als in Subtask B, ist das Resultat noch klarer ausgefallen. Dieses Experiment zeigt auf, dass bei unbalancierten Klassen durch Class-Weigts einiges herausgeholt werden kann. Jedoch ist noch unklar, wie sich das System mit Class-Weights auf ungesehen Daten verhält, wenn die Klassen verschiedene Grade von Ausgeglichenheit aufzeigen. Aus diesem Grund wird anhand der Class-Weights-Messung kein Vergleich mit dem bisherigen System erstellt.

8.6 Experiment 6: Unknown-Words

Dieses Experiment überprüft, wie viele Wörter während der Supervised Phase nicht im Vokabular gefunden werden. Wird für ein Wort kein Word-Embedding gefunden, kann das System mit diesem Wort nichts anfangen und es wird ignoriert. Dies ist natürlich möglichst zu vermeiden. Dazu kommt, dass Usernamen und Hashtags sich sehr schnell ändern und dauernd neue hinzukommen.

Um herauszufinden, wie viele Wörter nicht gefunden werden, werden alle ungleichen Wörter des Test-Datensets von SemEval-2016 aus dem Vokabuar abgefragt und die Nicht-Gefundenen gezählt. Zusätzlich werden diese für eine spätere Anaylse aufgelistet. Der Anteil der Unknown-Words, welcher aus Usernamen und Hashtags besteht wird auch aufgezeigt.

8.6.1 Beobachtung

8.6.1.1 Subtask B

Beschreibung	**Anzahl Wörter**
Total	19'610
Unknown	2'910
davon Usernamen	1'495
davon Hashtags	617

Tabelle 24: Experiment Unknown-Words Subtask B

8.6.1.2 Subtask C

Beschreibung	**Anzahl Wörter**
Total	32'426
Unknown	6'058
davon Usernamen	3'120
davon Hashtags	1'135

Tabelle 25: Experiment Unknown-Words Subtask C

8.6.2 Auswertung

Dieses Experiment zeigt, dass eine grosse Anzahl an Wörtern im Vokabular fehlen. Viele der fehlenden Wörter sind aber Usernamen, welche vernachlässigt werden können. Zudem finden sich viele falsch geschriebene oder umgangssprachliche Worte im Vokabular. An den unbekannten Hashtags kann man erkennen, dass die Tweets nicht immer aktuell sind und auch sehr exotische oder unbeliebte Hashtags vorkommen.

Interessant ist nun, ob diese Zahl durch das Hinzufügen von aktuelleren Tweets verringert werden kann und ob dadurch eine Verbesserung im Score erreicht wird.

8.7 Experiment 7: Themenspezifische Tweets

Dieses Experiment überprüft, ob zusätzliche, themenspezifische Tweets beim Lernen der Word-Embeddings einen positiven Einfluss auf den Score der Supervised-Phase haben.

Wie die zusätzlichen Daten gesammelt wurden, ist im Kapitel Daten unter themenspezifische Tweets (vgl. 7.4) beschrieben.

Die bereits vorhandenen Daten werden mit den zusätzlichen Daten ergänzt und nach den 200 Millionen Tweets (vgl. 7.1) eingelesen. Anschliessend werden die Distant- und Supervised-Phase basierend auf diesen Daten ausgeführt und ausgewertet.

8.7.1 Beobachtung

8.7.1.1 Subtask B

System	Score
Baseline	0.8319
Topic Tweets	0.8136

Tabelle 26: Experiment Themenspezifische Tweets Subtask B Scores

Beschreibung	Anzahl Wörter
Total	19'610
Unknown	2'707
davon Usernamen	565
davon Hashtags	1'379

Tabelle 27: Experiment Themenspezifische Tweets Subtask B Unknown-Words

8.7.1.2 Subtask C

System	Score
Baseline	0.7056
Topic Tweets	0.7573

Tabelle 28: Experiment Themenspezifische Tweets Subtask C Scores

Beschreibung	Anzahl Wörter
Total	32'426
Unknown	5'654
davon Usernamen	1'054
davon Hashtags	2'887

Tabelle 29: Experiment Themenspezifische Tweets Subtask C Unknown-Words

8.7.2 Auswertung

Im Subtask B wurden die unbekannten Wörter von 2'910 auf 2'707 und im Subtask C von 6'058 auf 5'654 verringert. Obwohl die Anzahl der unbekannten Wörter verringert werden konnte, weist der Score keine Verbesserungen auf. Der Score verschlechterte sich im Subtask B um zwei und im Subtask C sogar um fünf Prozentpunkte.

Eine Erklärung für den schlechteren Score könnte die Qualität der zusätzlichen Tweets sein.

9 Diskussion

Aus dieser Projektarbeit geht hervor, dass es wichtig ist, eine gute Architektur des Systems zu haben und auf Optimierungen im Bereich der Laufzeit zu achten. Die Experimente konnten planmässig ausgeführt werden. Zusätzliche Experimente und weiterführende Analysen werden aber noch benötigt. Der Teilnahme an SemEval-2017 steht nichts mehr im Weg.

9.1 Architektur

In dieser Arbeit wurde die komplette Architektur des bestehenden Systems überarbeitet und erweitert. Durch die genaue Analyse der bestehenden Codebasis konnten viele Vorgänge verstanden und sogar optimiert werden. Zudem wurde ein sehr generisches Pipeline-Modul geschaffen, welches einen modularen Aufbau der Phasen erlaubt und nicht nur für diese Arbeit nützlich ist.

9.2 Optimierungen

Die grössten Optimierungen konnten durch die Experimente und das Überarbeiten der Architektur erreicht werden. Durch das so gewonnene tiefere Verständnis für das System, konnten klarere Abläufe definiert werden, welche eine weitere Verbesserung des Systems bewirkten. Der grosse Fokus auf die Architektur hat sich gelohnt.

Die Laufzeit der Systeme konnte aufgrund der fehlenden Daten des bestehenden System nicht verglichen werden. Aufgrund der in Abschnitt 5.1 beschriebenen Optimierungen, wird angenommen, dass die Laufzeit beträchtlich reduziert werden konnte.

9.3 Experimente

Es wurden alle Experimente ausgeführt und ausgewertet. Viele Experimente zeigten eine klare Verbesserung der Resultate. Bisher verwendete Parameter können jetzt mit genauen Daten hinterlegt werden. Leider war die Auswertung der Experimente nicht ganz einfach, da die nicht-deterministischen Berechnungen das Vergleichen der einzelnen Läufe sehr erschwerte. Zusätzliche Läufe, um die Resultate zu mitteln oder das Issue genauer zu untersuchen, konnten aus Zeitgründen nicht ausgeführt werden.

Erstaunlicherweise verschlechterte das Experiment mit den themenspezifischen Tweets (vgl. 8.7) den Score in den jeweiligen Tasks um mehrere Prozentpunkte. Diese Erkenntnis konnte in dieser Arbeit nicht genauer untersucht werden.

9.4 SemEval-2017

Mit dem neuen System konnten die Sieger in beiden Subtasks von SemEval-2016 überboten werden. Die effektive Teilnahme am Wettbewerb und allfällige Anpassungen sind nicht mehr Teil dieser Projektarbeit, werden aber im Anschluss in Angriff genommen.

10 Glossar

BLAS Softwarepaket, welches elementare Operationen der linearen Algebra in Low-Level Ebene implementiert.

Convolution Mathematischer Operator, der als Produkt von Funktionen verstandenen werden kann.

Counter Subklasse eines Python Dictionaries, welcher als Value die Anzahl des Keys hat. Ermöglicht arithmetische Operationen mit Dictionaries.

Cython Python zu C Compiler.

EarlyStopping Callback, um das Training des Modells zu beenden, falls keine Verbesserung nach einer gewissen Bedingung mehr auftritt.

F1-Score Statistische Messmethode, um die Genauigkeit eines Analysetools zu evaluieren. Harmonischer Mean aus Precision und Recall.

Feature Messbare Ausprägung.

Gensim Softwarebibliothek, die eine Implementation von Word2Vec in Python enthält [14].

Gradient Descent Numerisches Verfahren zum Lösen von allgemeinen Optimierungsproblemen.

Keras High-Level-Python-Bibliothek für Theano und TensorFlow.

Lazy Loading Entwurfsmuster der Softwareentwicklung. Laden der Variable erst nach dem ersten Zugriff und nicht schon bei Initialisierung.

ModelCheckpoint Speichert den Stand des Modells zu einem definierten Zeitpunkt ab.

NumPy Softwarebibliothek für multidimensionale numerische Berechnungen in Python.

Overfitting Übertraining des Modells. Das Modell generalisiert nicht und ist zu fest an Trainingsdaten angepasst.

Pooling-Layer Im Pooling-Layer werden Informationen von Neuronen anhand einer Funktion(z.B. maximaler Wert) zusammengefügt. Verringert die Dimensionalität.

Sample Einzelner Datensatz.

SemEval SemEval ist ein Wettbewerb für Computersysteme, bei dem die semantische Evaluation von Texten im Vordergrund steht.

TensorFlow Softwarebibliothek für effiziente und multidimensionale numerische Berechnungen in Python.

Text-Mining Analyseverfahren zur Entdeckung von Bedeutungsstrukturen in Textdaten.

Theano Softwarebibliothek für effiziente und multidimensionale numerische Berechnungen in Python.

Token Element einer Sprache, z.B. ein Wort.

Tokenizer Modul für das Aufteilen eines Satzes in Tokens.

Word-Embedding Vektorrepräsentation eines Wortes.

Word2Vec Modelle zur Generierung von Word-Embeddings.

Worker Thread für die Bearbeitung eines Programmteils.

11 Literatur

[1] P. Nakov, A. Ritter, S. Rosenthal, V. Stoyanov und F. Sebastiani, „SemEval-2016 Task 4: Sentiment Analysis in Twitter“, in *Proceedings of the 10th International Workshop on Semantic Evaluation*, Ser. SemEval '16, San Diego, California: Association for Computational Linguistics, Juni 2016.

[2] ZHAW. (2016). Online Administration Praktische Arbeiten. [Online], Adresse: `https://tat.zhaw.ch/tpada/arbeit_vorschau.jsp?arbeitID=14631` (besucht am 17. 12. 2016).

[3] D. R. Witte und J. Mülle, „Text Mining: Wissensgewinnung aus natürlichsprachigen Dokumenten“, März 2006.

[4] SemEval. (2016). Tasks < SemEval-2017. [Online], Adresse: `http://alt.qcri.org/semeval2017/index.php?id=tasks` (besucht am 20. 12. 2016).

[5] A. Ng. (2016). Machine Learning. [Online], Adresse: `https://www.coursera.org/learn/machine-learning` (besucht am 15. 12. 2016).

[6] A. Geitgey. (Mai 2014). Machine Learning is Fun. [Online], Adresse: `https://medium.com/@ageitgey/machine-learning-is-fun-80ea3ec3c471` (besucht am 8. 12. 2016).

[7] N. de Freitas. (2015). Machine Learning. [Online], Adresse: `https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/lecture9.pdf` (besucht am 21. 12. 2016).

[8] TensorFlow TM. (Dez. 2016). Vector Representations of Words. [Online], Adresse: `https://www.tensorflow.org/versions/r0.11/tutorials/word2vec/index.html` (besucht am 20. 12. 2016).

[9] G. Rey und K. Wender, *Neuronale Netze: eine Einführung in die Grundlagen, Anwendungen und Datenauswertung*, Ser. Aus dem Programm Huber: Psychologie-Lehrbuch. Huber, 2011.

[10] T. Mikolov, K. Chen, G. Corrado und J. Dean, „Efficient Estimation of Word Representations in Vector Space“, *CoRR*, Bd. abs/1301.3781, 2013.

[11] C. Olah. (Jan. 2015). Visualizing Representations: Deep Learning and Human Beingss. [Online], Adresse: `https://colah.github.io/posts/2015-01-Visualizing-Representations/` (besucht am 14. 12. 2016).

[12] A. Descoins. (März 2013). Why accuracy alone is a bad measure for classification tasks, and what we can do about it. [Online], Adresse: `https://tryolabs.com/blog/2013/03/25/why-accuracy-alone-bad-measure-classification-tasks-and-what-we-can-do-about-it/` (besucht am 8. 12. 2016).

[13] V. V. Asch, „Macro- and micro-averaged evaluation measures“, Sep. 2013, Basic Draft.

[14] R. Řehůřek und P. Sojka, „Software Framework for Topic Modelling with Large Corpora“, in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta: ELRA, Mai 2010, S. 45–50.

[15] R. Řehůřek. (Sep. 2013). Word2vec in Python, Part Two: Optimizing. [Online], Adresse: https://rare-technologies.com/word2vec-in-python-part-two-optimizing/ (besucht am 17. 12. 2016).

[16] J. J. Dongarra, J. Du Croz, S. Hammarling und R. J. Hanson, „Algorithm 656: An Extended Set of Basic Linear Algebra Subprograms: Model Implementation and Test Programs“, *ACM Trans. Math. Softw.*, Bd. 14, Nr. 1, S. 18–32, März 1988.

[17] R. Bradshaw, S. Behnel, D. Seljebotn, G. Ewing und et al. (2016). The Cython compiler. [Online], Adresse: http://cython.org (besucht am 19. 12. 2016).

[18] R. Řehůřek. (Okt. 2013). Parallelizing word2vec in Python. [Online], Adresse: https://rare-technologies.com/parallelizing-word2vec-in-python/ (besucht am 17. 12. 2016).

[19] Theano Development Team, „Theano: A Python framework for fast computation of mathematical expressions“, *arXiv e-prints*, Bd. abs/1605.02688, Mai 2016.

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu und X. Zheng, „TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems“, *CoRR*, Bd. abs/1603.04467, 2016.

[21] F. Chollet. (2015). Keras. [Online; Username: fchollet], Adresse: https://github.com/fchollet/keras (besucht am 8. 12. 2016).

[22] J. Deriu, M. Gonzenbach, F. Uzdilli, A. Lucchi, V. De Luca und M. Jaggi, „SwissCheese at SemEval-2016 Task 4: Sentiment Classification Using an Ensemble of Convolutional Neural Networks with Distant Supervision“, in *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, San Diego, California: Association for Computational Linguistics, Juni 2016, S. 126–130.

[23] R. Hoffmann, C. Zhang, X. Ling, L. Zettlemoyer und D. S. Weld, „Knowledge-based Weak Supervision for Information Extraction of Overlapping Relations“, in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, Ser. HLT '11, Portland, Oregon: Association for Computational Linguistics, 2011, S. 541–550.

[24] M. D. Zeiler, „ADADELTA: An Adaptive Learning Rate Method“, *CoRR*, Bd. abs/1212.5701, 2012.

[25] K. Reitz und T. Schlusser, *The Hitchhiker's Guide to Python*. O'Reilly Media, Sep. 2016.

[26] N. Batchelder. (März 2012). Pragmatic Unicode. [Online], Adresse: http://nedbatchelder.com/text/unipain.html (besucht am 19. 12. 2016).

[27] B. Langdon. (Dez. 2012). Generator Pipelines in Python. [Online], Adresse: https://brett.is/writing/about/generator-pipelines-in-python/ (besucht am 17. 12. 2016).

[28] Twitter, Inc. (2016). Character Counting - Twitter Developers. [Online], Adresse: https://dev.twitter.com/basics/counting-characters (besucht am 8. 12. 2016).

[29] SemEval. (2016). Data and Tools < SemEval-2017. [Online], Adresse: http://alt.qcri.org/semeval2017/task4/index.php?id=data-and-tools (besucht am 20. 12. 2016).

[30] P. L. (Username: lamblin). (Juni 2015). Flag for deterministic GPU operations. [Online], Adresse: https://github.com/Theano/Theano/issues/3029 (besucht am 14. 12. 2016).

[31] F. Chollet. (Juli 2015). is there any way to note down or control the random seeds? [Online; Username: fchollet], Adresse: https://github.com/fchollet/keras/issues/439 (besucht am 17. 12. 2016).

[32] scikit-learn. (2016). sklearn.ensemble.RandomForestClassifier. [Online], Adresse: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (besucht am 15. 12. 2016).

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

A Architektur

A.1 Pipeline

Gesamte Implementation der in Kapitel 6.5 gelisteten Pipeline.

```
from itertools import islice, chain
import random


class Pipeline(object):
    '''
    Generic Pipeline for handling processors and their corresponding statics
    Important: The Input/Output of processing functions need to correspond
    Statics are assets that can be referenced in the processing functions

    Args:
        sources: the list of Readers
        config: default config static
    '''
    def __init__(self, sources, config=None):
        self.sources = sources
        self.processors = []
        self.statics = {}
        self.config = config

        if self.config:
            self.add_static('config', self.config)

    def __iter__(self):
        for sample in self.generate():
            yield sample

    def add_processor(self, processor):
        '''
        Add a processing function to the pipeline

        Args:
            processor: the processing function
        '''
        if callable(processor):
            self.processors.append(processor)

    def add_static(self, name, static):
```

```
39          '''
40          Add a static Asset to the pipeline
41
42          Args:
43              name: the name of the asset
44              static: the static object
45          '''
46          self.statics[name] = static
47
48      def generate(self):
49          '''
50          Generate from source and apply all processors
51
52          Yields:
53              Generated samples from sources
54          '''
55          pipelines = []
56          for source in self.sources:
57              pipeline = source.read()
58              for processor in self.processors:
59                  pipeline = processor(pipeline, self.statics)
60
61              pipelines.append(pipeline)
62          return chain.from_iterable(pipelines)
63
64      def generate_batch_inf(self, permute=None, batch_size=1,
    ↪ shuffle=True):
65          '''
66          Infinitely generate batches (wrap if end is reached)
67
68          Args:
69              permute: the function to permute the batch with before
    ↪ generation
70                  (e.g. unzipping, ordering...)
71              batch_size: batch size
72              shuffle: whether to randomly shuffle the batch
73          Yields:
74              Infinite Batches of samples
75          '''
76          while True:
77              batches = self.generate_batch(batch_size=batch_size,
    ↪ shuffle=shuffle,
78                                            permute=permute)
79              for batch in batches:
80                  yield batch
81
82      def generate_batch(self, permute=None, batch_size=1, shuffle=True):
83          '''
```

```
84          Generates a batch of data
85
86          Args:
87              permute: the function to permute the batch with before
    ↪  generation
88                  (e.g. unzipping, ordering...)
89              batch_size: batch size
90              shuffle: whether to randomly shuffle the batch
91          Yields:
92              Batches of samples
93          '''
94          samples = self.generate()
95
96          it = iter(samples)
97          batch = list(islice(it, batch_size))
98
99          while batch:
100             # Call before permute
101             if shuffle:
102                 random.shuffle(batch)
103
104             if callable(permute):
105                 batch = permute(batch)
106
107             yield batch
108
109             batch = list(islice(it, batch_size))
```

Listing 6: Modul: *pipeline.py*

A.2 Verzeichnisstruktur

Für jedes Experiment wurde in den nachfolgenden Ordnern (ausser *data*) je ein Unterordner, welcher nach dem Experiment benannt ist, erstellt. Die Dateinamen werden normalerweise immer nach der dazugehörigen Phase oder ihrer Beschreibung (z.B. *vocabulary.pickle*) benannt.

A.2.1 Verzeichnis: *data*

Alle Semeval Trainings- und Testdaten für die Supervised-Phase befinden sich im Ordner *data*. Die Daten für die Embedding- und Distanz Phase wurden extern gespeichert, da die Dateigrösse für die Versionskontrolle zu gross und unpraktikabel ist. Aus diesem Verzeichnis wird nur gelesen.

A.2.2 Verzeichnis: *embeddings*

Zusätzlich zum Vokabular und der Embedding-Matrix wurde das trainierte Word2Vec-Modell abgespeichert, um einen reproduzierbaren Stand zu erhalten.

A.2.3 Verzeichnis: *log*

Die Logfiles wurden pro Phase erstellt und der gesamte STDOUT und sowie der STDERR wurden der Einfachheit halber jeweils in ein Logfile, benannt nach der Phase(z.B. *supervised.log*), umgeleitet. Zusätzlich besteht ein Logfile, in welchem die Zeitmessung jeder Phase festgehalten(*main.log*) ist.

A.2.4 Verzeichnis: *model*

Für die Modelle wurden jeweils die vom ModelCheckpoint gelieferten Gewichte und die dazugehörige Modelldefinition abgespeichert.

A.2.5 Verzeichnis: *results*

Die Resultate oder auch der Verlauf eines Experiments (Stand nach jeder Epoche) wird in einer JSON-Datei gespeichert. Diese beinhaltet den Verlauf aller Trainings- und Validationsmetriken.

B Getting started

B.0.1 Dependencies

TopicThunder uses the following dependencies: (not downloaded automatically)
- Anaconda3 4.2.0
- Theano 0.8.2
- Keras 1.1.2
- Gensim 0.13.3

Optional tools primary to speed-up the calculations:
- NVIDIA CUDA 8.0 (Optional)
- NVIDIA cuDNN (Optional)

B.0.2 Installation

```
git clone https://github.com/tobiashuonder/TopicThunder.git
```

B.0.3 Configuration

B.0.3.1 Theano configuration

http://deeplearning.net/software/theano/library/config.html

B.0.3.2 Keras configuration

https://keras.io/backend/

B.0.3.3 Experiment Configuration

Open the Configuration- or Runfiles e.g. *supervised.json* or *distant.json* or *embeddings.json* or *run_ experiments.sh* and set all properties to the prefered value.
The most important ones you need to set are the absolute paths to the data sets and the corresponding types of readers because they are different on every setup. Only the SemEval Datasets for the Supervised-Phase are included in the folder *data*.

B.0.4 Run experiments

To start the experiments, you have to be in the root folder of the project. Remember to start them in the background for convenience.
- Start a specific experiment:
  ```
  . /conf/$ExperimentName/run_experiments.sh &
  ```
- Start all experiments:
  ```
  . /conf/run_experiments.sh &
  ```

B.0.5 Analyze Experiment

The Log- and Result-Files can be found in the folder named after the corresponding experiment. Model- and Embedding-Files are also saved and can be reused.