Aufbau und Evaluierung eines dynamischen Wide & Deep Learning Frameworkes für Text Classification

Masterarbeit zur Erlangung des

Master of Advanced Studies ZFH in

Data Science

vorgelegt von

Manuel Streiff

geboren am 17.03.1990

von Zürich, ZH

eingereicht

Dr. Mark Cieliebak

Winterthur, 31. Januar 2019

Management Summary

Bei Wide and Deep Learning handelt es sich um ein Konzept, welches 2016 von Cheng et al. bei Google vorgestellt wurde. Das Konzept beschreibt ein Machine Learning Modell, welches mittels zwei Komponenten, einer Wide- und einer Deep-Komponente, die Prozesse des Erinnerns (Memorization) und des Generalisierens (Generalization) nachbildet. Dafür kommt im Entwurf von Google ein lineares Modell sowie ein DNN zum Einsatz.
Das Konzept wurde von verschiedenen Parteien aufgenommen, weiterentwickelt, und die daraus gewonnenen Erkenntnisse in diversen Veröffentlichungen dokumentiert. So setzte beispielsweise Huawei eine Factorization Machine als Wide-Komponente ein und zog weitere Architekturen eines neuronalen Netzes in Betracht. Burel at al. wendeten Wide and Deep Learning auf Tweets an, welche im Kontext von Krisen entstanden. Zudem leiteten sie aus den Inhalten der Tweets semantische Informationen ab, welche sie als zusätzliche Features für die Wide-Komponente verwendeten.

Aufbauend auf der Literatur wurde ein kleines Framework erstellt, welches die Anwendung eines Wide and Deep Learning Modells für Text Classification vereinfachen soll. Als Basis des Frameworks kommt Keras zum Einsatz. Zentral wird ein Wide and Deep Modell zur Verfügung gestellt, welches aus je einer Wide- und Deep-Komponenten abgeleitet wird. Einige einfache Architekturen solcher Komponenten werden dabei direkt durch ds Framework bereitgestellt, ebenso wie einige Grundfunktionen zur Evaluation des Modelles.

Anschliessend wurden mit Hilfe des Frameworks drei Datensets evaluiert, eines in Englisch, sowie zwei in Deutsch. In der Evaluation kamen vier verschiedene Ansätze zum Einsatz, welche jeweils miteinander verglichen wurden: Ein Wide-Modell, ein Deep-Modell, ein Model Ensemble aus den beiden sowie ein Wide and Deep Modell. Diese Ansätze wurden jeweils mit verschiedenen Modellen und Parametern geprüft. Die Ergebnisse haben gezeigt, dass Wide and Deep Learning ähnlich gute Resultate erzielen kann wie ein Model Ensemble. In einigen Anwendungsfällen zeigte sich aber, dass das Wide and Deep Modell sensibler auf Einbrüche einzelner Komponenten reagiert, als dies beim Ensemble der Fall ist. Es darf jedoch angenommen werden, dass das Wide and Deep Modell gegenüber dem Ensemble mehr Potenzial hat, da es mehr Spezifikationsmöglichkeiten bietet, welche im Rahmen dieser Arbeit jedoch nicht ausgeschöpft wurden.

Inhaltsverzeichnis

1 Einleitung

Während der letzten drei Jahre wurde im MAS Data Science umfassendes Wissen rund um Data Science vermittelt. Dieses Wissen soll nun in dieser Arbeit praktisch angewendet werden. Mit den unterschiedlichen Anwendungsgebieten von Machine Learning standen zahlreiche Möglichkeiten zur Verfügung.

Bei Tamedia AG, als grösste private Mediengruppe der Schweiz, spielen Texte in verschiedenen Formaten eine zentrale Rolle. Und mit den heutigen Technologien, wie z.B. Social Media, wird es immer wichtiger, diese Texte rasch analysieren zu können. Sei es im Bereich des Datenjournalismus', bei der Verarbeitung von Kommentaren auf den Newsseiten oder der Kategorisierung von Kundenanfragen: Texte sind allgegenwärtig.
Auf der anderen Seite ist es im Bereich Data Science wichtig, mit der Entwicklung Schritt zu halten und sich immer wieder mit neuen Ansätzen und Methoden auseinanderzusetzen. Einen interessanten Schritt in diese Richtung hat 2016 Cheng et al. im Rahmen von Recommendation Systems bei Google gemacht: Sie entwickelten ein Konzept, welches sie Wide and Deep Learning nannten und Vorteile verschiedener Machine Learning Ansätze kombinieren sollte.

Mit diesen zwei Aspekten war der Rahmen dieser Arbeit geboren: Einerseits ist ein Problem vorhanden, welches es zu lösen gilt, auf der anderen Seite existiert ein möglicher, neuer Lösungsansatz, welchen es zu evaluieren gilt. Kombiniert man diese, erhält man das Ziel dieser Arbeit: Wide and Deep Learning soll angewendet werden, um verschiedene Text Classification Probleme anzugehen. Die Resultate sollen im Rahmen von Ablationsstudien verglichen werden, um das Potenzial von Wide and Deep Learning in diesem Anwendungsbereich abschätzen zu können.

1.1 Ziele der Arbeit

Die konkreten Ziele dieser Arbeit wurden in einer vorgelagerten Disposition festgelegt. Es wurden je drei qualitative und quantitative Ziele definiert:

Qualitative Ziele:

1. Die Literatur zum Thema Wide and Deep Learning soll umfassend und detailliert bearbeitet und zusammengefasst werden.
2. Anhand der erarbeiteten Literaturrecherche soll ein Überblick über den momentanen Stand im Bereich Wide and Deep Learning gegeben werden.
3. Basierend auf den gewonnenen Erkenntnissen soll ein Ausblick gewagt werden: Wo liegen Chancen und Möglichkeiten von Wide and Deep Learning, wo allfällige offene Probleme?

Quantitative Ziele:

1. Es sollen unterschiedliche Wide and Deep Learning Algorithmen entwickelt und mittels Ablationsstudie evaluiert werden.
2. Es sollen verschiedene Hyperparameter und Optimizer eingesetzt und deren Resultate verglichen werden.
3. Es soll ein Framework programmiert werden, welches den Einsatz und die Konfiguration von Wide and Deep Learning Modellen vereinfacht.

1.2 Aufbau der Arbeit

Die Arbeit gliedert sich in folgende Kapitel:

Grundlagen und Literatur
Gemäss den Zielen ist die Bearbeitung der vorhandenen Literatur eine zentrale Aufgabe dieser Arbeit. Daher wird diese in einem separaten Kapitel abgehandelt. Es sollen bekannte Einsatzgebiete sowie Stärken und Schwächen des Konzeptes analysiert und bewertet werden.

Framework
Das zweite Kapitel wird der Erstellung des Frameworks gewidmet. Dazu gehört die grobe Beschreibung des Funktionsumfangs wie auch Erklärungen, warum einzelne Designentscheidungen getroffen wurden. Die Verwendung des Frameworks wird anhand eines konkreten Beispieles aufgezeigt und dokumentiert.

Evaluation
In der Evaluationsphase wird das Framework auf die verfügbaren Daten angewandt. Hierfür stehen drei Datensammlungen zur Verfügung, welche kurz beschrieben werden, bevor die Klassifizierungsergebnisse vorgestellt werden.

Ausblick
Im Ausblick wird eine Prognose gewagt, wie es mit Wide and Deep Learning weitergehen könnte. Zusätzlich findet ein kritischer Rückblick des Autors auf die gesamte Arbeit sowie die gewonnenen Erkenntnisse und Kompetenzen statt.

Grundlagen und Literatur

2

Im ersten Abschnitt dieses Kapitels werden allgemeine Grundlagen zum Thema Machine Learning vorgestellt, wobei die wichtigsten Begriffe aufgezeigt werden, welche in der weiteren Arbeit zur Anwendung kommen. Anschliessend wird auf das Problem der Text Classification eingegangen und einige State-of-the-Art Methoden präsentiert. In einem letzten Schritt wird das Thema Wide and Deep Learning bearbeitet. Damit soll die Frage beantwortet werden, wie das Problem der Text Classification angegangen werden könnte. Auch hier werden konkrete Anwendungsbeispiele vorgestellt.

2.1 Grundlagen

In diesem Abschnitt wird ganz kurz ein Einstieg ins Thema Machine Learning gegeben. Es wird darauf eingegangen, was man sich unter Machine Learning vorstellen kann, bevor die drei Komponenten Repräsentation, Evaluation und Optimierung, wie sie von Domingos definiert wurden, etwas detaillierter erklärt werden [Dom12]. Dabei richtet sich der Fokus bereits auf die für das Problem der Text Classification relevanten Themen.

2.1.1 Machine Learning

> *Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions.*
>
> **— emerj.com**

Machine Learning, als Teilbereich der künstlichen Intelligenz, war in den letzten Jahren medial sehr präsent und kommt bereits in den verschiedensten Bereichen zum Einsatz, sei es zur Erkennung von Spam, von Sprache für Mensch-Maschinen-Kommunikation oder zur Unterstützung in verschiedenen Anwendungen im Gesundheitsbereich. Beim obigen Zitat handelt es sich um eine Definition, welche mehrere Experten gemeinsam festgelegt haben und die versucht, den verschiedenen Aspekten von Machine Learning gerecht zu werden. Während diese Definition die zentralen Faktoren abdeckt, kann die persönliche Meinung einzelner Experten in gewissen Bereichen durchaus etwas abweichen.
Es gibt verschiedene Ansätze, Machine Learning in weitere Unterbereiche aufzuteilen, wie z.B. nach Art des Lernens (Supervised, Unsupervised und Reinforcement

Learning) oder der angewandten Technik (Klassifikation, Regression, Entscheidungsbaum, Deep Learning, etc.) [Fag18].

Domingos wiederum hat ein Konzept aufgestellt, mit welchem sich sämtliche Machine Learning Ansätze in drei Komponenten aufteilen lassen: Repräsentation, Evaluation und Optimierung. Auf diese Komponenten wird in den folgenden Kapiteln genauer eingegangen [Dom12].

Representation	Evaluation	Optimization
Instances	Accuracy/Error rate	Combinatorial optimization
K-nearest neighbor	Precision and recall	Greedy search
Support vector machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naive Bayes	Posterior probability	Continuous optimization
Logistic regression	Information gain	Unconstrained
Decision trees	K-L divergence	Gradient descent
Sets of rules	Cost/Utility	Conjugate gradient
Propositional rules	Margin	Quasi-Newton methods
Logic programs		Constrained
Neural networks		Linear programming
Graphical models		Quadratic programming
Bayesian networks		
Conditional random fields		

Abb. 2.1: Die drei Komponenten lernender Algorithmen nach Domingos

2.1.2 Repräsentation

Der Bereich der Repräsentation lässt sich grob in zwei Bereiche unterteilen. Einerseits geht es darum, in welchem Format die bereitgestellten Daten verarbeitet werden sollen. Diesen Prozess der Auswahl und Aufbereitung der Daten in die gewünschten Formate nennt man auch Feature Engineering. Der zweite Bereich ist das eigentliche Modell, welches das Problem abbilden soll.
Um hier nicht zu tief in die Details zu versinken, werden die Themen jeweils nur sehr grob bearbeitet.

Texte als Features

Text in seiner eigentlichen Form wird normalerweise nicht direkt als Input für Modelle verwendet. Es werden zuerst sogenannte Features daraus abgeleitet, welche maschinell besser verarbeitet werden können. Dafür haben sich verschiedene Ansätze etabliert:

Bag of Words
Das Bag of Words-Verfahren ist weit verbreitet und eignet sich für verschiedenste Modelle. Für einen Bag of Words wird zuerst eine Liste von Wörter definiert. Darin können alle Wörter enthalten sein, die im Trainingsdatensatz vorkommen oder eine Teilmenge davon, beispielswiese die häufigsten 10'000. Anschliessend wird pro Text gezählt, welches Wort wie häufig vorkommt. Über alle Texte hinweg erhält man so eine meist relativ dünn besetzte Matrix, da pro Text nur ein Bruchteil der Wörter vorkommt. Allerdings lässt sich am Vorkommen gewisser Wörter bereits viel über den Inhalt des eigentlichen Textes aussagen, insbesondere z.B. im Zuge einer thematischen Kategorisierung [RM18].

tf-idf

Tf-idf steht für “term frequency-inverse document frequency“ und beurteilt die Wortrelevanz, resp. Aussagekraft einzelner Wörter. Da Wörter, welche häufig vorkommen, wie z.B. “der“, “die“ oder “das“, für das Modell häufig nicht relevant sind, wird die absolute Worthäufigkeit der inversen Dokumenthäufigkeit gegenübergestellt. Tf-idf wertet also Wörter stärker, welche in einem spezifischen Dokument häufig vorkommen, aber nur in wenigen Dokumenten. Diese sagen meist viel über die Charakteristik eines einzelnen Textes aus [RM18].

Word embeddings

Ein weiterer Ansatz ist die Repräsentation von Wörtern als sogenannte Embeddings. Einer der am weitesten verbreiteten solcher Ansätze ist beispielsweise word2vec, welcher 2013 von Mikolov et al. vorgestellt wurde. Die Berechnung von word2vec basiert selbst auf einem neuronalen Netz und versucht, Beziehungen zwischen Wörtern zu lernen und als Vektoren abzubilden. So können Relationen verschiedener Terme mathematisch wiedergegeben werden, z.B.: “Paris“ - “France“ + “Italy“ = “Rome“ [Mik+13].

Weitere Möglichkeiten

Natürlich können anhand des Textes auch weitere Features abgeleitet werden. So können z.B. die Anzahl verschiedener Worttypen wie Nomen oder Verben gezählt werden, die Länge des Textes oder Wörter, das Vorkommen gewisser Zeichen, etc. Hier sind den Möglichkeiten quasi keine Grenzen gesetzt, allerdings kann die Berechnung dieser Features rasch sehr aufwändig werden.

Modelle

Neben den Daten gehört auch das Modell zur Repräsentation. Das Modell bildet das eigentliche Problem ab: Es erhält Daten in dem zuvor festgelegten Format und leitet daraus ein Resultat ab. Mithilfe der Optimierung werden die Parameter innerhalb des Modells so angepasst, dass das Problem möglichst genau abgebildet wird. Häufig im Rahmen von Text Classification verwendete Modelle sind:

Logistische Regression

Die logistische Regression wird häufig als Modell zur Vorhersage eines binären Wertes (0 oder 1) verwendet. Anders als z.B. eine lineare Regression wird nicht ein direkter Zielwert geschätzt, sondern meist die Wahrscheinlichkeit, dass das Objekt z.B. einer Klasse A angehört (Maximum Likelihood Prinzip).

Support Vector Machine (SVM)

Ein SVM dient ebenfalls dazu, Daten in Klassen einzuteilen. Dafür werden die zur Verfügung stehenden Daten als Vektoren abgebildet. Anschliessend wird versucht, eine Hyperebene so in diesen Vektorraum zu legen, dass ein möglichst grosser Raum zwischen den verschiedenen Klassen entsteht. Neue Objekte werden dann anhand ihres Vektors entsprechend klassifiziert.

DNN

DNN’s, Dynamic Neural Networks, oder auch Feed Forward Neural Networks ge-

nannt, können als einfachste Vertreter der neuronalen Netze betrachtet werden. Sie bestehen hauptsächlich aus “fully-connected layers“. Das bedeutet, dass die Inputparameter von den Knoten solcher “fully-connected layers“ verarbeitet werden und danach auf alle Nodes des nachfolgenden Layers weitergegeben werden. Der Output dieses Layers wird wiederum an jeden Node weitergeleitet, usw.

Abb. 2.2: Grobe Architektur eines DNN [Vee16]

CNN

Im Gegensatz zu einem DNN verwendet ein CNN, ein Convolutional Neural Network, nicht nur die einzelnen Inputs, sondern bezieht auch “angrenzende“ Werte mit ein. Daher eignet sich diese Art von Netzen insbesondere für Daten mit Datenpunkten, welche eng mit ihren jeweiligen Nachbarn verbunden sind, wie z.B. Pixel eines Bildes oder Buchstaben/Wörter in einem Text. Dabei sind zwei Arten von Layers zentral: Convolutional Layer, welche die Werte inkl. Nachbarwerten verarbeiten, und Pooling Layer, welche mehrere Werte aggregieren.

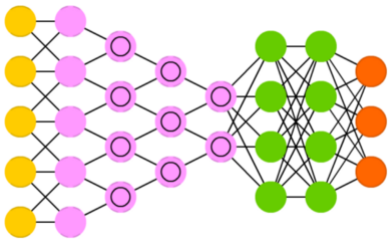

Abb. 2.3: Grobe Architektur eines CNN [Vee16]

RNN

Ein RNN, ein Recurrent Neural Network, ist wiederum in der Lage, sich innerhalb des Netzes an vorherige Datenpunkte zu erinnern. Somit eignet sich ein RNN insbesondere für sequentielle Datensätze, in welchen auch die Reihenfolge der Inputs eine Rolle spielt. Dies ist auch bei Text der Fall: Wörter machen häufig erst in einer gewissen Reihenfolge Sinn. Die wahrscheinlich meistverbreitete Architektur eines “Recurrent node“ ist das LSTM (Long Short-Term Memory) von Hochreiter und Schmidhuber [HS97].

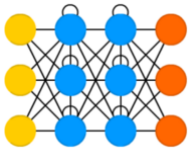

Abb. 2.4: Grobe Architektur eines RNN [Vee16]

2.1.3 Evaluation

Auch der Bereich Evaluation kann wiederum in zwei Bereiche eingeteilt werden: Einerseits sind sogenannte Loss-Funktionen Teil der Evaluation. Diese werden für das Training eines Modelles verwendet und helfen dem Optimierer, die erzielten Resultate einordnen zu können. Daneben gibt es weitere Evaluationsmethoden, welche hauptsächlich dem Entwickler, resp. Data Scientisten, helfen, Modelle untereinander vergleichen zu können.

Loss-Funktionen

Eine bekannte Loss-Funktion ist beispielsweise der Mean Squared Error (MSE), welcher häufig bei linearen Regressionen angewendet wird. Im Rahmen dieser Arbeit wird “binary crossentropy“, verwendet, welcher sich insbesondere für binäre Klassifizierungsprobleme eignet. Diese Loss-Funktionen sind sehr mathematisch, weshalb hier auf eine detailliertere Beschreibung verzichtet wird.

Konfusionsmatrix

Eine sogenannte Konfusionsmatrix wird eingesetzt, um die Resultate eines, meist binären, Klassifikationsproblemes zu evaluieren. Dabei wird der vom Modell vorhergesagte Wert dem tatsächlichen Wert gegenübergestellt. Ein optimales Modell würde lediglich “true positives“ und “true negatives“ haben. Die beiden anderen Boxen widerspiegeln Fehlklassifikationen [RM18]. In der Praxis ist es jedoch meist nicht möglich, Fehlklassifikationen zu verhindern.

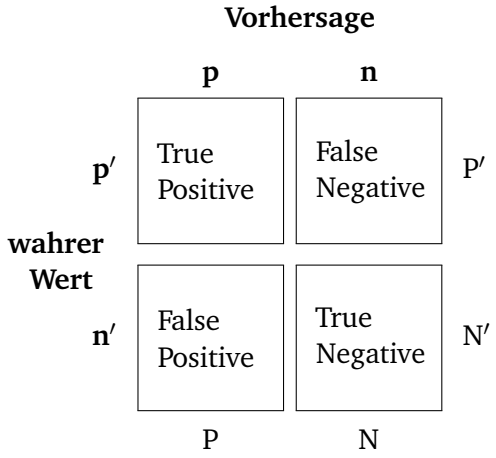

Abb. 2.5: Konfusionsmatrix

Präzision vs. Recall

Auf Basis der Konfusionsmatrix können Precision (deutsch: Genauigkeit) und Recall (deutsch: Trefferquote) berechnet werden [RM18]:

$$precision = \frac{truepositives}{truepositives + falsepositives}$$

$$recall = \frac{truepositives}{truepositives + falsenegatives}$$

Precision drückt aus, wieviele der als Kategorie A prognostizierten Objekte tatsächlich zu der Kategorie A gehören. Ist die Precision 100%, gehören alle als A kategorisierten Fälle tatsächlich auch zur Kategorie A. Ist die Precision 0%, sind alle als Kategorie A eingeteilten Fälle in Wahrheit nicht Kategorie A.
Recall wiederum beschreibt, wie viele der tatsächlich vorhandenen Objekte der Kategorie A auch als Kategorie A vorhergesagt werden. Ist diese Quote 100%, wurden alle Objekte der Kategorie A korrekt erkannt. Ist der Recall wiederum 0%, wurde kein Objekt der Kategorie A korrekt erkannt. Es ist also möglich, dass die Precision 100% beträgt, also alle vom Modell als A kategorisierten Objekte auch A sind, der Recall aber nur 50% erreicht, da die Hälfte der Objekte der Kategorie A nicht korrekt erkannt und als B kategorisiert wurde.
Da im Normalfall keine perfekten Quoten erreicht werden können, muss beim Erstellen des Modelles meist eine Frage über die Priorität geklärt werden. Was ist wichtiger: Sollen alle Objekte der Kategorie A erkannt werden, oder soll sichergestellt werden, dass in den prognostizierten Kategorie A Objekten keine Kategorie B Objekte vorhanden sind? Mit dem Festlegen eines Grenzwertes, dem sogenannten Threshold, kann der Klassifizierer entsprechend beeinflusst werden.

Eine Messgrösse, welche das Verhältnis von Precision und Recall berücksichtigt und daher ausgewogene Modelle belohnt, ist die sogenannte F1-Score:

$$F1 - Score = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

Da die F1-Score Precision sowie Recall miteinbezieht, kann diese Messgrösse nur bedingt durch das Anpassen des Grenzwertes optimiert werden.
Sowohl Precision und Recall, wie auch die F1-Score können als Macro Average-Wert berechnet werden. Dabei wird die Precision sowie Recall pro Klasse berechnet, bevor anschliessend die jeweiligen Durchschnittswerte über alle Klassen berechnet werden. Die Macro Average F1-Score bildet sich dann aus diesen beiden Durchschnittswerten.

ROC und AUC

Wie im vorherigen Abschnitt erwähnt, hat der Threshold, also der Grenzwert, ab welcher Wahrscheinlichkeit ein Objekt der Klasse A zugewiesen wird, eine zentrale Bedeutung. Ein wichtiges Werkzeug hierbei ist die ROC-Kurve, siehe Abbildung 2.6, Ein Beispiel einer ROC-Kurve. ROC steht für “Receiver Operating Characteristics“, auf

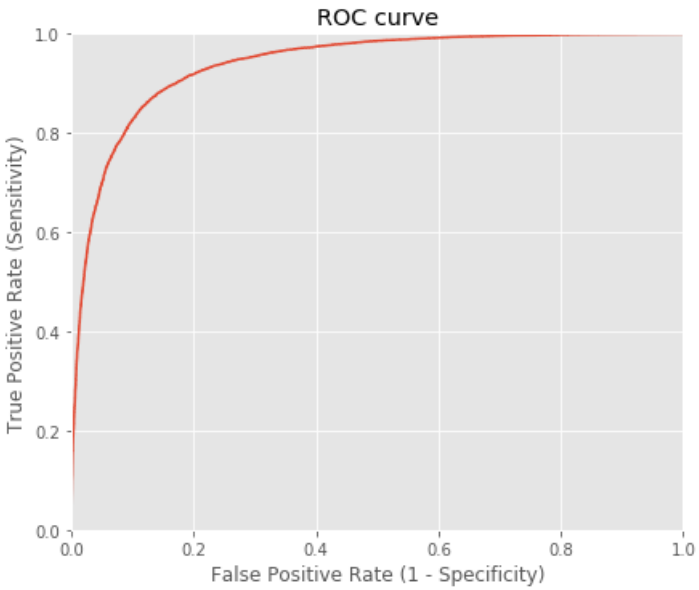

Abb. 2.6: Ein Beispiel einer ROC-Kurve

deutsch in etwa Grenzwertoptimierungskurve. Sie zeigt, wie sich die Falsch-Positiv, resp. Richtig-Positiv-Rate gegenüber veschiedenen Thresholds verhält [RM18].
In einem optimalen Modell würde die Kurve eine Linie über den oberen linken Ecken darstellen. Zeigt die Linie eine Diagonale, entspricht das Modell mehr oder weniger dem Zufall. Die AUC, Area under Curve, dient als Metrik, um die ROC-Kurve zu messen. Das Ziel ist, dass die Fläche unterhalb der Linie, also die AUC, möglichst gross ist.

2.1.4 Optimierung

Optimierer sind dafür verantwortlich, die Parameter eines Modelles zu optimieren. In einem linearen Modell bestimmen beispielsweise verschiedene Parameter, wie stark die verschiedenen Features miteinbezogen werden, und was für einen Einfluss sie auf das Resultat haben. Auf die verschiedenen Arten von Optimierungen, resp. Optimierern, soll hier nicht weiter eingegangen werden.
Im Bereich neuronaler Netze für Text Classification wird häufig der Adam-Optimizer verwendet. Daher wird auch im Rahmen dieser Arbeit auf diesen zurückgegriffen. Beim Adam Optimizer handelt es sich um einen Optimierungsalgorithmus, der 2015 von Kingma et al. vorgestellt wurde. Er ist eine Erweiterung des Stochastic Gradient Descent (SGD) und wird momentan vor allem im Bereich Computer Vision und NLP eingesetzt [KB14].

2.2 Text Classification

Text Classification ist eine Teildisziplin von NLP, Natural Language Processing. Das Ziel von NLP ist es, natürliche Sprache, normalerweise in Form von Text, mit Computern zu verarbeiten und Informationen daraus zu gewinnen [Con+16].
Text Classification wird häufig eingesetzt, wenn es darum geht, verschiedene Texte in

vordefinierte Kategorien einzuteilen. Dies kann beispielsweise bei Spam-Mails (Spam / kein Spam) oder bei Online-Kommentaren (veröffentlichen / nicht veröffentlichen) der Fall sein. Es ist natürlich auch möglich, Texte in mehr als zwei verschiedene Kategorien zu unterteilen (Zeitungsartikel in Rubriken) oder einem Text mehrere Kategorien zuzuweisen (Hashtags bei einem Instagramm-Post) [SZS18].
Dieses Thema hier umfassend zu bearbeiten, würde den Rahmen dieser Arbeit sprengen. Um dennoch eine gute Basis für das weitere Vorgehen aufzubauen, werden hier vier Verfahren vorgestellt, welche momentan zum State-of-the-Art gehören, resp. in letzter Zeit gute Resultate erzielten, und in der Literatur gut beschrieben sind. Diese Ansätze sollen insbesondere auch als Grundlage für die Wide-, resp. Deep-Komponente eines möglichen Wide and Deep Modelles dienen.

2.2.1 Support Vector Machine (SVM)

Ausgangslage

Joachmis veröffentlichte 1998 eine Arbeit über SVM's im Bereich von Text Classification. Er wies darauf hin, dass die Menge der verfügbaren Onlinedaten, welche häufig in Textform entstehen, rasch wächst und somit gute Lösungen zur Text Classification immer wichtiger werden. Zudem soll dies Klassifizierung einfach anhand von Trainingsdaten passieren, und damit nicht abhängig von aufwändig aufbereiteten Features sein. Dabei lehnte er sich stark an Information Retrieval an, wo Texte als Vektoren (z.B. Bag of Words) repräsentiert werden, sowie deren idf (inverse document frequency). Dies wäre ein Schritt in Richtung automatischer Text Classification, da kein spezifisches Feature Engineering mehr nötig wäre [Joa98].

Modell

Um dieses Problem anzugehen, bediente sich Joachims an dem von Vapnik et al. vorgestellten SVM [CV95]. SVM's basieren auf dem Prinzip der Structural Risk Minimization (SRM). Dabei wird versucht, eine Hypothese für die Klassifikation zu finden, welche eine möglichst grosse Distanz zwischen die beiden zu unterscheidenden Klassen bringt, also neue, unbekannte Texte möglichst genau, resp. eindeutig, einordnen kann. Ein wichtiger Punkt ist, dass SVM sehr universal und relativ unabhängig von der Anzahl Dimensionen des Feature Vektors funktionieren können. Joachims zeigt das Problem der Text Classification vier zentrale Eigenschaften, die für eine Anwendung von SVM's sprechen:
(1) Texte besitzen normalerweise einen hochdimensionalen Input, da jedes Wort als einzelner Vektor abgebildet wird. (2) Nur sehr wenige Wörter (=Features) sind irrelevant. Ein Versuch zeigte, dass auch ein mit "irrelevanten" Wörtern (solche mit sehr kleinem Informationsgehalt) trainiertes SVM eine Klassifikation erreicht, welche besser als der Zufall ist. (3) Jedes Dokument wird als dünn besetzter Vektor abgebildet: Von all den Wörtern, die in der Dokumentensammlung vorhanden sind, sind nur wenige im zu klassifizierenden Dokument vorhanden. (4) Die meisten Text Classification Probleme sind linear lösbar.

Ergebnisse

In einem Vergleich stellte Joachims ein SVM vier anderen Algorithmen gegenüber, darunter naive Bayer und K-nearest-neighbours. Als Basis dienten zwei Datensets: Einmal eine Textsammlung von Reuters mit 9'603 Trainings- und 3'299 Testtexten, welche sich in 90 Kategorien unterteilen. Aus den Texten wurden 9'962 Terme extrahiert. Zusätzlich stand ein weiteres Datenset zur Verfügung, aus welchem je 10'000 Texte für Training und Text verwendet wurden. Dabei handelte es sich um medizinische Dokumente, welche anhand definierter Terme in "disease categories" eingeteilt wuden. Anhand von 15'561 Termen wurden diese in 23 Kategorien unterteilt, wobei hier Mehrfachkategorisierungen möglich waren.
Von den traditionellen Modellen zeigte kNN die besten Resultate. SVM war jedoch in den meisten Fällen besser oder ebenbürtig mit den Ergebnissen, welche kNN erreichte. SVM zeigt sich daher als sehr gut geeignet zur Anwendung auf Text Classification Probleme.

2.2.2 CNN

Ausgangslage

Zhang et al. haben ein Convolutional Neural Network (CNN) entwickelt, welches zeichenbasiert Texte klassifiziert. Ihre Überlegung war, Text als rohes Signal zu behandeln. Eine ähnliche Betrachtung hat bereits im Bereich von Computer Vision und Speech Recognition zu grossen Fortschritten geführt. So ist z.B. im Bereich der Texte eine Klassifizierung möglich, ohne dass Wissen über Semantik oder Syntax der Sprache vorhanden ist. Dies bringt einige Vorteile mit sich, da so auch Schreibfehler und z.B. Emojis einfacher verarbeitet werden können. Im veröffentlichten Papier werden auch Texte in chinesischer Sprache klassifiziert [ZZL15].

Modell

Als Input wurden die einzelnen Zeichen eines Textes codiert und die einzelnen Texte damit vektorisiert. Dies ergibt ein Tensor der Grösse `Anzahl Zeichen des verwendeten Alphabetes x definierte Maximumlänge eines Textes`. In einigen Fällen wurden zudem einzelne Wörter mittels Thesaurus durch Synonyme ersetzt, um eventuell bessere Resultate zu erzielen. Verwendet wurde ein 1-dimensionales CNN, dessen Hauptbestandteile sogenannte 1-dimensionale Convolutional Module waren (Temporal Convolutional Layers mit Max Pooling). Nach 6 Layern Convolutional Modules werden die Features durch drei Fully-connected Layers (inkl. 2 Drop-out Layers) weiterverarbeitet.

Ergebnisse

Die Ergebnisse wurden mit diversen anderen Modellen verglichen, einigen traditionellen Logistic Regression Classifiers (Bag of N-Grams, Bag of Words, Bag of Means) wie auch Deep Learning Lösungen (Wort-basiertes CNN sowie LSTM). Das vorgeschlagene zeichenbasierte CNN stellte sich dabei als kompetitiv heraus, wobei seine

Stärken eher im Bereich der Verarbeitung grosser Datensammlungen liegen, wie bei CNN's üblich. Zudem wurde die Hypothese aufgestellt, dass das vorgeschlagene CNN bei unbearbeiteten, direkt von Benutzern erfassten Texten besser funktioniert, was mit der besseren Berücksichtigung von Emojis und Schreibfehlern erklärt werden könnte.

2.2.3 Very Deep CNN

Ausgangslage

Conneau et al. gingen im Bereich der zeichenbasierten CNNs noch einen Schritt weiter, resp. tiefer, und entwickelten ein VDCNN, ein Very Deep Convolutional Neural Network. Der Einsatz von neuronalen Netzen brachte Fortschritte im Bereich von Text Classification, diese sind jedoch noch lange nicht so gross wie in anderen Bereichen, wie z.B. Computer Vision und Speech Recognition. Die Autoren zeigten dabei eine Analogie auf: Texte bestehen aus einzelnen Zeichen, Wörtern, Sätzen und Abschnitten, ebenso wie auch Bilder aus verschiedenen Ebenen bestehen: Pixeln, Symbole, Objekte, etc.
Conneau et al. argumentieren, dass die meisten NLP-Ansätze weiterhin Wort-basiert sind und mit Embeddings arbeiten. Dies bringt allerdings Probleme mit sich, da mit einfachen Wort-Vektoren die Information über die Wort-Reihenfolge verloren geht. Die Reihenfolge kann jedoch grossen Einfluss auf die Aussage einen Textes haben. Mit RNN's kann auch die Reihenfolge berücksichtigt werden. Conneau et al. nehmen zudem Bezug auf Zhang et al., und heben inbesondere deren Erkenntnisse bzgl. der Anwendung auf grosse Datensammlungen hervor [Con+16].

Modell

In der Umsetzung verwendeten Conneau et al. ein CNN, welches grob in drei Bereiche aufgeteilt werden kann. In der ersten Phase wurde ein zweidimensionaler Vektor erstellt, welcher die Character Embeddings darstellte. Anschliessend wurde dieser Vektor einem ersten, separaten Convolutional Layer übergeben. In einer zweiten Phase gab es eine dynamische Anzahl sogenannter Convolutional Blocks mit unterschiedlichen Filtergrösse von 64, 128, 256 sowie 512. In verschiedenen Tests wurde eine variierende Anzahl an Blocks pro Filtergrösse eingesetzt, von 2 bis hin zu 16. Die Blocks bestanden wiederum aus zwei Convolutional Layern, sowie jeweils anschliessendem Temporal Batch Norm und ReLu Layer. In einer letzten Phase wurden auch hier die Features an drei full-connected Layers übergeben. Im Gegensatz zu Zhang wurde kein Drop-out angewendet.

Ergebnisse

Die Evaluation fand auf den gleichen Datensammlungen statt wie das beschriebene CNN von Zhang et al. Und obwohl Conneau et al. auf eine Variierung der Daten anhand von Synonymen verzichtete, konnten dessen Resultate übertroffen werden. Es hat sich zudem gezeigt, dass sich die Ergebnisse bis hin zu 29 Convolutional Layern (10/10/4/4 Blocks) verbesserte, bevor die Genauigkeit wieder abnahm,. Dieses

Verhalten konnte auch bereits in anderen sehr tiefen Netzen beobachtet werden und könnte damit erklärt werden, dass so die Einflüsse der Backpropagation verloren gehen und eine Optimierung des gesamten Netzes erschwert wird.
Conneau et al. kamen zu zwei Aussagen, welche hier kurz erwähnt werden sollten. Erstens ist mit einem tieferen Netz auch eine grössere Datensammlung notwendig, um gute Ergebnisse zu erreichen. Tiefe Netze haben auch bei kleinen Datensammlungen flächere übertroffen, kommen jedoch noch nicht an die Ergebnisse von linearen Modellen heran.
Zweitens scheint im Vergleich zu Fortschritten in anderen Anwendungsbereichen auch die Anzahl der zu bestimmenden Klassen eine grosse Rolle zu spielen. Während in den hier evaluierten Datensammlungen nur sehr wenige Klassen (2 bis 14) vorhanden waren, kommen z.B. in Computer Vision häufig tausende von Klassen vor. Im Bereich Text Classification ist daher der Beitrag eines Textes zur Definition der Klasse wesentlich kleiner.

2.2.4 Recurrent CNN

Ausgangslage

Lai et al. brachten den Einwand auf, dass viele Methoden zur Text Classification die Reihenfolge der Wörter innerhalb eines Textes nicht oder zu wenig berücksichtigen. Diese trägt in vielen Fällen jedoch stark zur Aussage eines Textes bei. Zudem können Wörter mit Merhfachbedeutungen ohne Kontext falsch interpretiert werden. N-Grams können dieses Problem teilweise abfangen, gerade Vektoren langer N-Grams, welche mehr Kontext bieten würden, sind aber dünn besetzt und benötigen sehr viel Trainingsdaten. Die Autoren zogen verschiedene Architekturen in Betracht: Recursive Neural Nets, Recurrent Neural Nets und Convolutional Neural Nets. Da sich das Recursive Neural Net eher für kurze Texte eigne und über eine hohe Time complexity verfügt, haben sie sich entschieden, die besten Teile aus Recurrent Neural Net und CNN zu kombinieren und ein Recurrent CNN zu entwerfen [Lai+15].

Modell

Die Hauptkomponente des angewendeten Modells bestand aus einer sogenannten “recurrent structure“. Es wurde eine bi-direktionale Architektur gewählt, womit die einzelnen Nodes jeweils Zugriff auf den gesamten Kontext haben: Einmal den Text vor, einmal den Text nach dem gerade bearbeiteten Wort. Die drei Komponenten (Vorher-Kontext, Word Embedding, Nachher-Kontext) wurden zusammengefasst und einer `tanh`-Aktivierungsfunktion übergeben. Anschliessend wurden die Werte in einem MaxPooling-Layer aggregiert. Damit wurden einerseits verschieden lange Sätze auf eine gleiche Länge gebracht, andererseits sollen so nur die wichtigsten Faktoren eines Textes weiter berücksichtigt werden. Zum Schluss wurde ein Output Layer mit `softmax` angewandt.
Als Hyperparameter wurde ein “stochastic gradient descent“ mit einer Lernrate von 0.1 gewählt, eine Hidden Layer-Grösse von 100 sowie eine Vektorgrösse von je 50 für Word Embeddings und Kontext. Die Word Embeddings wurden mit word2vec anhand von Wikipedia trainiert.

Ergebnisse

Das Modell wurde auf vier Datensets angewandt, drei englische und ein chinesisches. Bei allen Datensets handelte es sich um Merhfachklassifizierungen (englische Datensets: 4 oder 5 Klassen, chinesisches Datenset: 20 Klassen). Das Modell wurde mit diversen anderen Modellen mit teilweise unterschiedlichen Feature-Ansätzen verglichen. Einerseits wurden Baseline Modelle verwendet: eine logistische Regression oder ein SVM mit verschiedenen Feature Sets (Bag of Words, Bigrams, Average Embeddings). In einer zweiten Gruppe kamen die State-of-the-Art Modelle des jeweiligen Datensets zum Zuge: LDA-basierte Ansätze, Tree Kernels, ein Recursive Neural Net und ein CNN.
Erwartungsgemäss haben die neuronalen Netze besser abgeschnitten als die Baseline-Modelle. Gegenüber den State-of-the-Art Modellen erzielte das vorgeschlagene RCNN in zwei der vier Datensets die besten Resultate, in einem dritten Datenset war es ebenbürtig mit einem der verwendeten Tree Kernels, wobei dieses jedoch auf aufwändig aufbereiteten Features basiert. Im Vergleich unter den neuronalen Netzen schloss das RCNN am Besten ab. Dies wird damit begründet, dass den separaten Netzen jeweils etwas fehlt: beim CNN ist es die Information aus dem Kontext, dem rekursiven Netz mangelt es an der Fokusierung, welche mit dem Max Pooling erzwingt wird.

2.3 Wide and Deep Learning

Dieses Kapitel setzt sich mit dem zweiten Fokus dieser Arbeit auseinander, Wide and Deep Learning. Im ersten Abschnitt wird Wide and Deep Learning vorgestellt, wie es 2016 von Google konzeptioniert wurde. Dabei wird nicht nur auf die von Google veröffentlichte Literatur eingegangen, sondern auch auf die durch Tensorflow verfügbaren Python-Bibliotheken. In den nachfolgenden Abschnitten werden drei weitere Anwendungen vorgestellt, welche Wide and Deep Learning entweder weiterentwickeln oder einen ähnlichen Ansatz auf neue Themen anwenden.

2.3.1 Google's Wide and Deep Learning

Wide and Deep Learning ist ein Konzept, welches 2016 von Cheng et al. bei Google entwickelt und vorgestellt wurde [Che+16]. In diesem Abschnitt wird zuerst auf eine Analogie eingegangen, welche die Überlegungen hinter Wide and Deep Learning aufzeigen soll. Anschliessend wird die Ausgangslage beschrieben, welche als Basis für die Pilotarbeit von Cheng et al. diente, bevor die einzelnen Komponenten des Modelles sowie die erzielten Resultate genauer analysiert werden. Zum Schluss wird auf die Funktionen zu Wide and Deep Learning eingegangen, welche momentan in Tensorflow verfügbar sind.

Analogie

Die Herleitung der Idee und der dahinterliegenden Logik wird ausführlich in einem Blogeintrag im Google AI Blog ausgeführt [Che16]. Zentral dabei ist die Unterscheidung zwischen Generalisieren und (spezifischem) Erinnern (Englisch: Generalization

and Memorization).
Für ein einfacheres Verständnis wird ein Beispiel aus der Tierwelt beigezogen: Als Menschen sehen wir, dass Spatzen und Tauben fliegen können und erinneren uns daran, wenn wir wieder Spatzen oder Tauben sehen. Mit der Zeit lernen wir: Tiere mit Flügel können fliegen. Wenn wir also einen Raben sehen, gehen wir automatisch davon aus, dass auch dieses Tier fliegen kann. Sehen wir allerdings Pinguine, bemerken wir: Pinguine können nicht fliegen, obwohl sie Flügel haben.
Wir Menschen generalisieren mit der Zeit also, dass Tiere mit Flügel fliegen können. Sehen wir eine Ausnahme, erinnern wir uns später daran und wissen: dieses Tier kann nicht fliegen, obwohl es Flügel hat. Cheng et al. haben mit Wide and Deep Learning versucht, dieses Zusammenspiel von Generalisieren und Erinnern in einem Machine Learning Modell abzubilden. In ihrem Konzept übernimmt die Wide-Komponente die Aufgabe des Erinnerns (Memorization), während die Deep-Komponente für das Generalisieren (Generalization) zuständig ist.

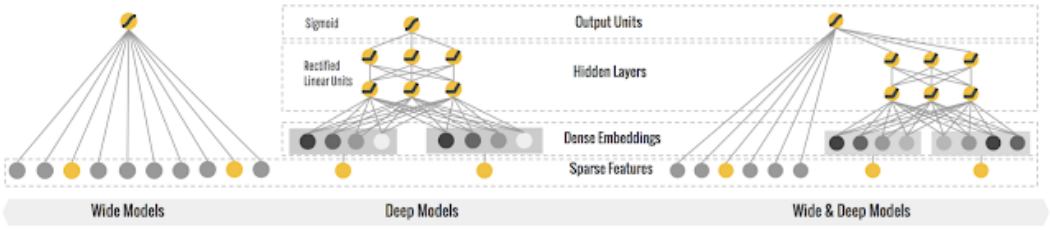

Abb. 2.7: Vergleich: Wide vs. Deep vs. Wide and Deep Modell

Ausgangslage

Cheng et al. beschreiben die Anwendung eines solchen Wide and Deep Learning Modelles auf ein Recommendation System für den Google App Store. In diesem Bereich wird es als grosse Herausforderung beschrieben, Memorization und Generalization gleichzeitig zu erreichen. Einerseits bestehen direkte Beziehungen zwischen verschiedenen Apps (wer App A herunterlädt, lädt auch App B herunter), gleichzeitig sollen aber auch Verknüpfungen zu neuen oder ähnlichen Apps generiert werden, welche bisher noch nicht existieren (wer App A und B herunterlädt, könnte auch App C herunterladen, auch wenn bisher noch nicht passiert) [Che+16]. Im Bereich der App Online-Recommendations wurden bisher meist logistische Regressionsmodelle eingesetzt. Diese haben einerseits gute Resultate geliefert, andererseits sind sie sehr performant, was bei in diesem Anwendungsbereich zentral ist.

Modell

Die Architektur eines Wide and Deep Learning Modelles besteht grob aus drei Teilen: einer Wide-Komponente, einer Deep-Komponente und einem gemeinsamen Layer am Ende des Modells.
Die Aufgabe der Wide-Komponente ist es, spezifische Dinge zu merken (Memorization). Dies geht am besten mit einer flachen, breiten Architektur. Die Idee ist es, auf diese Weise viele Abhängigkeiten möglichst direkt abzubilden. In der Implementation von Google übernimmt diese Aufgabe ein lineares Modell. Als Input für das lineare Modell dienen sowohl unbearbeitete Features wie auch transformierte. Die sogenannten “cross-product“ Transformationen, bei welchen mehrere Bedingungen

verknüpft werden (z.B. `"AND(gender=female, language=en)"`), spielen dabei eine zentrale Rolle [Che+16].
Die Deep-Komponente hat die Aufgabe des Generalisierens. Es sollen grobe Zusammenhänge in den Daten erkannt, gelernt, und auf neue Beispiele angewendet werden können. Dies ist eine Stärke von neuronalen Netzen. Cheng et al. haben hierfür ein normales "feed-forward neural net" eingesetzt. Die Basisdaten für die Deep-Komponente sind sehr ähnlich. Während allerdings in der Wide-Komponente bereits mit "cross-products" gearbeitet wird, Beziehungen zwischen Features also explizit definiert werden, arbeitet die Deep-Komponente mit den direkten Features (z.B. `"(gender=female)"`). Diese Komponente soll die Beziehungen dazwischen von Grund auf selber lernen.
In einem letzten Schritt werden die beiden Komponenten dann zusammengeführt. Es ist dabei wichtig zu berücksichtigen, dass die beiden Komponenten Teil eines Modelles sind und es sich nicht um separate Modelle handelt. Das heisst, dass gesamte Modell wird gemeinsam trainiert. Dadurch können die beiden Bereiche zusammenarbeiten und sich gegenseitig ergänzen. Damit unterscheidet sich dieser Ansatz zentral von einem Model Ensemble. In einem Model Ensemble würde man separate Modelle erstellen und anschliessend, beispielsweise über eine Art "Voting", eine Lösung bestimmen.

Ergebnisse

Das Modell wurde in einer produktiven Umgebung getestet, wobei zwei Aspekte zentral waren: Erfolg der Empfehlungen (App-Akquisitionen) und Performance. Es wurden drei Modelle angewendet: Ein Wide-Modell, ein Deep-Modell sowie das Wide and Deep-Modell. Im Bereich der App-Akquisitionen erzielte das Wide and Deep-Modell die besten Resultate. Es erreichte eine Steigerung von 3.9% gegenüber dem Wide-Modell. Das Deep-Modell erzielte ebenfalls eine Verbesserung von 2.9% zum Wide-Modell, schloss damit aber dennoch signifikant schlechter ab als der Wide and Deep Ansatz. Im Bereich der Performance konnte mit der Aufteilung der Batches auf 4 Threads die Latenz von 31 ms auf 14 ms gesenkt werden, bei über 10 Millionen Appbewertungen pro Sekunde.

Wide and Deep Learning in der Tensorflow-Bibliothek

Das Wide and Deep Learning Konzept wurde bereits von Google in Tensorflow integriert. Es gibt momentan zwei Klassen, welche die Anwendung von Wide and Deep vereinfachen sollen: Den `DNNLinearCombinedClassifier` und den `DNNLinearCombinedRegressor`. Beide Klassen sind gleich aufgebaut. Sie benötigen eine Liste von Features für den linearen Teil (Wide) sowie für den DNN Teil (Deep). Zusätzlich können separate Optimizer angegeben werden und die Anzahl Units sowie Layers der Deep-Komponente können definiert werden. Hier ein Beispiel für die Initialisierung eines `DNNLinearCombinedClassifier` [Goo18c]:

```
estimator = DNNLinearCombinedClassifier(
    # wide settings
    linear_feature_columns=[categorical_feature_a_x_categorical_feature_b
    ],
    linear_optimizer=tf.train.FtrlOptimizer(...),
    # deep settings
```

```
    dnn_feature_columns=[
        categorical_feature_a_emb, categorical_feature_b_emb,
        numeric_feature],
    dnn_hidden_units=[1000, 500, 100],
    dnn_optimizer=tf.train.ProximalAdagradOptimizer(...),
    # warm-start settings
    warm_start_from="/path/to/checkpoint/dir")
```

2.3.2 Semantic Wide and Deep Learning

Ausgangslage

Während Naturkatastrophen und anderen Krisensituationen werden über soziale Plattformen sehr viele Informationen ausgetauscht. Burel et al. stellten ein Wide and Deep Modell vor, welches solche Nachrichten in Kategorien wie Information, Ratschlag, Hilfsangebot, etc. einteilt.
Diese Nachrichten während einer Krise sind zahlreich, kommen sehr schnell und können in sehr unterschiedlichem Format verfügbar sein. Bisherige Ansätze, diese Informationen für Behörden und Organisationen sinnvoll zu nutzen, beschränkten sich auf die Identifizierung, ob eine Nachricht überhaupt krisenbezogen ist, sowie auf die darauf aufbauende Frage, auf welche Krise sich die Nachricht bezieht. Hierfür wurden bisher Classifier oder auch Clustering Ansätze verwendet, auf welche jedoch nicht näher eingegangen wurde. Deep Learning hatte aber in letzter Zeit signifikante Verbesserungen gebracht. Das hier vorgestellte Sem-CNN baut einerseits auf der von Google vorgeschlagenen Wide and Deep Architektur auf, andererseits liegt ein grosser Fokus auf der Verwendung von semantischen Informationen, mit welchen die Nachrichten angereichert werden [BSA17].

Modell

Das beschriebene Sem-CNN ist relativ komplex und konzentriert sich neben dem eigentlichen Modell auch sehr stark auf das Data Preprocessing. Als Input für das Modell werden zwei Embeddings aufbereitet. Einerseits wird mit word2vec ein gängiger Word Embedding Ansatz angewendet. Die daraus gewonnenen Embeddings laufen in die Deep-Komponente des Modelles. Andererseits werden anhand der Texte semantische Informationen abgeleitet und ebenfalls als Vektoren dargestellt. Hierfür wird TextRazor in Verbindung mit DBpedia eingesetzt. Diese Vektoren repräsentieren, welche semantischen Informationen in der Nachricht vorkommen und bereiten diesen als Input für die Wide-Komponente von Sem-CNN auf.
Als Ansatz der Extraktion der semantischen Informationen wurden zwei Varianten geprüft: Semantische Konzepte und semantische Abstrakte. Bei einem semantischen Konzept handelt es sich um ein Abstrahierung (Beispiel: Barack Obama = Politiker), während semantische Abtrakte den Begriff weiter beschreiben (Beispiel: Barack Obama = Politiker, Präsident, USA).
Das eigentliche Sem-CNN basierte auf 300 Word Embeddings und arbeitete mit einem 128 mal 3/4/5 grossen Convolutional Filter. Zudem wurde eine Dropout Rate von 0.5 sowie ein Adam Optimierer eingesetzt. Es wurden 2'000 Iterationen mit einer Batch Grösse von 256 durchgeführt. Im Gegensatz zu Google's Ansatz verwendet diese Lösung also ein CNN anstelle eines DNN. Zudem verwendet Sem-CNN gänzlich

unterschiedliche Inputparameter für Wide und Deep, während beim Ansatz von Google grundsätzlich die gleichen Informationen verarbeitet wurden, lediglich in unterschiedlichem Format.

Ergebnisse

Als Datenset stand eine Sammlung von knapp 28'000 Tweets zur Verfügung, welche in verschiedene Informationskategorien unterteilt sind. Diese wurden in zwei kleineren Datensets weiter verfeinert: Ein ausgeglichenes Datenset (gleiche Anzahl Tweets pro Kategorie) sowie ein ausgeglichenes Datenset mit Tweets, welche min. zwei semantische Informationen enthalten.
Sem-CNN wurde mit zwei bisherigen Ansätzen verglichen: eine, SVM auf Basis von td-idf sowie einem SVM auf Basis von Word Embeddings. Um auch den Einfluss der semantischen Informationen abzuschätzen, wurden die gleichen SVM's auch in einer weiteren Variante mit den semantischen Konzepten, resp. Abstrakten trainiert. Von den traditionellen Modellen erzielte das auf td-idf basierte SVM die besten Resultate. Die Verwendung von word2vec wie auch von semantischen Informationen brachte eine Verschlechterung der Resultate mit sich. Zudem wurden für das grösste Datenset mit allen Tweets die besten Ergebnisse erreicht.
Das Sem-CNN konnte bereits nach sehr wenigen Schritten seine Resultate optimieren. Es erreichte in etwa die gleichen Werte wie das beste SVM, konnte auf den kleineren Datensets jedoch bessere Resultate ausweisen, während diejenigen des SVM dort etwas schlechter ausfielen.

2.3.3 DeepFM

Ausgangslage

Guo et al. bezogen sich in ihrer Veröffentlichung sehr stark auf das Konzept von Google's Wide and Deep Learning. Auch ihre Arbeit fand im Bereich der Recommendation Systems für Smartphone Apps statt, allerdings bei Huawei. Die Autoren gingen ebenfallls genauer auf die Stärken des Konzeptes ein. So steht ein Recommendation System häufig im Dilemma zwischen der Abbildung von Low-Order und High-Order Interaktionen. Hier konnte wiederum die Parallele zu Memorization und Generalization aufgezeigt werden. Bei Low-Order Interaktionen handelt es sich um direkte Zusammenhänge. Eine Order-2 Interaktion ist beispielsweise, dass zur Mittagszeit vermehrt Food-Apps heruntergeladen werden (Interaktion zweier Faktoren). Eine Order-3 Interaktion ist, dass Benutzer, die männlich und jung sind, sich vermehrt für Spiele-Apps interessieren (Interaktion zwischen drei Faktoren).
Lineare Modelle zeigten bereits gute Ergebnisse, haben aber Mühe, Feature Interaktionen abzubilden und sind daher häufig auf manuelles Feature Engineering angewiesen. Factorizations Machines können solche Feature Interaktionen automatisch modellieren. Da im Bereich der App Recommendations jedoch sehr viele Features existieren, sind auch diese häufig auf Low-Order Interaktionen beschränkt. Neuronale Netze wiederum eignen sich für die Repräsentation von High-Order Interaktionen, können aber Gefahr laufen, Low-Order Interaktionen zu wenig zu gewichten, welche für App Recommendations ebenfalls sehr wichtig sind. Eine simultane Betrachtung solcher Features wird durch Wide and Deep Learning vereinfacht.

Die Autoren argumentieren, dass es eine der Hauptherausforderungen ist, diese Interaktionen zu definieren. Insbesondere bei High-Order Interaktionen ist dies eine schwierige Aufgabe. In Google's Konzept sehen sie dies daher als zentrale Verbesserungsmöglichkeit, da dort diese Interaktionen bereits als Input dienen und daher weiterhin manuell aufbereitet werden müssten. Mit DeepFM wollen die Autoren ermöglichen, dass das Modell mit nur einem Input zurecht kommt, der weitgehend auf Feature Engineering verzichten kann. Das heisst, dass diese Interaktionen nicht manuell abgebildet werden müssen, sondern als Teil des Algorithmus' erstellt werden [Guo+17].

Modell

DeepFM basiert, wie bereits der Google-Ansatz, auf zwei Bereichen: Einer Wide-Komponente, für welche hier eine Factorization Machine eingesetzt wird, sowie eine Deep-Komponente, für welche verschiedene Architekturen neuronaler Netze geprüft wurden. Die Idee dahinter war, mit der Factorization Machine die Low-Order Interactions abzubilden, während die High-Order Interactions im neuronalen Netz abgebildet werden sollten. Im Bereich des neuronlen Netzes kamen verschieden Architekturen in Frage. CNN und RNN wurden verworfen, da sich diese hauptsächlich für ´´nebeneinanderliegende´´ oder sequentielle Daten eignen, was im Bereich der App Recommendations nicht der Fall ist. Für die weitere Evaluation wurde einerseits ein normales DNN in Betracht gezogen, sowie verschiedene Varianten eines PNN. Ein PNN ist ein produktbasiertes neuronales Netz, welches auf verschiedene Weisen modelliert werden kann: Mit Outer-Join, mit Inner-Join, oder beidem. Für die Evaluation wurden alle Ausführungen getestet.

Ergebnisse

In einem ersten Schritt wurde das Modell intern evaluiert. Insgesamt wurden 12 verschiedene Modelle evaluiert, darunter das ursprüngliche Konzept von Google, wie auch neuronale Netze separat und in Verbindung mit einer Factorization Machine. Es wurden vier Punkte festgestellt: (1) Die Kombination von Factorization Machine mit DNN zeigte die besten Resultate. (2) PNN's mit Inner- oder beiden Joins benötigten eine sehr lange Rechenzeit. (3) Modelle mit Wide und Deep Teil zeigten bessere Resultate als nur der Deep Teil. (4) DeepFM, mit dem selben Input für Wide und Deep, zeigte bessere Resultate als Wide and Deep von Google, mit unterschiedlichen Inputs. Zudem wurden verschiedene Hyperparameter geprüft. Am Besten zeigte sich eine Architektur mit ReLu, Dropout-Raten zwischen 0.6 und 0.9 und 200 bis 400 Nodes pro Layer bei 3 Layern.
In einem zweiten Schritt wurde DeepFM im Rahmen eines A/B-Testing im App Markt von Huawei getestet. Es wurde dabei einem gängigen Logistic Regression Modell gegenübergestellt. Die Ergebnisse zeigten, dass DeepFM dem logistischen Modell klar überlegen ist. So konnte die CTR (Click-through-rate) um bis zu 25% verbessert werden.
Zudem stellten die Autoren fest, dass DeepFM auch persönlichere und heterogenere Empfehlungen liefert. Da ein Logistic Regression Modell viel stärker mit Low-Order Interaktionen arbeitet, werden häufig die gleichen, bereits stark verbreiteten Apps empfohlen. Da DeepFM auch High-Order Interaktionen miteinbezieht, werden viel spezifischere und auch neue Empfehlungen generiert.

2.3.4 Wide and Deep CNN

Ausgangslage

Zheng et al. beschrieben ein Wide and Deep Convolutional Neural Network, welches sie zur Erkennung von Stromdiebstahl entworfen haben. Stromdiebstahl ist in einigen Ländern ein grosses Problem, so wird beispielsweise in Kanada pro Jahr Strom im Wert von ca. 100 Mio. Dollar gestohlen. Grundsätzlich gibt es zwei Möglichkeiten, Stromdiebstahl zu erkennen: durch Hardware (z.B. Smart Meter) oder durch Software, in Form von Datenanalysen. Bisher wurden häufig Hardware-basierte Lösungen eingesetzt, diese bringen jedoch diverse Problem mit sich. Der Einsatz ist teuer, die Geräte sind anfällig gegenüber Wettereinflüssen und müssen regelmässig gewartet werden. Mit dem Aufkommen von Smart Grids besteht nun erstmals die Möglichkeit, das Problem relativ einfach durch Datenanalysen anzugehen. Allerdings gibt es für solche Modelle zahlreiche Herausforderungen. Es wird meist Themenwissen für das Feature Engineering benötigt und die Erkennungsraten bleiben relativ niedrig. Zudem werden spezielle Geräte benötigt, um den Stromverbrauch in verschiedenen Bereichen zu messen. Trotzdem werden häufig falsche oder auch gar keine Werte übertragen. Daher wird häufig auf Anomalie-Erkennung gesetzt: Während der Verbrauch von normalen Kunden häufig periodisch verläuft, widerspricht der Verbrauch von Stromdieben diesen Mustern [Zhe+18].

Modell

Vor dem eigentlichen Modell mussten die Daten speziell aufbereitet werden. Die fehlenden oder falschen Messwerte mussten erkannt und durch Interpolation vervollständigt oder korrigiert werden. Anschliessend setzten die Autoren auf eine Wide and Deep Kombination, ähnlich wie die von Google vorgestellte Variante. Für die Deep-Komponente wurde jedoch ein CNN eingesetzt, während für die Wide Komponente ein sehr flaches neuronales Netz genutzt wurde, welches lediglich aus einem Fully Connected Layer bestand. Das CNN betrachtete die Datenpunkte als "Bild", wobei die Daten mit einem Format von `{7 (Wochentage) x Anzahl Wochen` verarbeitet wird, um die Periodizität des Verbrauchs bestmöglich abzubilden. Es werden bis zu 5 Concolutional Layer gestapelt, bevor zum Schluss noch ein Pooling Layer und ein Fully connected Layer eingesetzt werden.

Ergebnisse

Als Datenset standen die Verbrauchszahlen von über 42'000 Kunden über 3 Jahre zur Verfügung, wobei es sich bei 3'600 Verbrauchern um Stromdiebe handelte. Das Modell zeigte eine Verbesserung gegenüber bisherigen Modellen, allerdings erzielte die Kombination von Wide und Deep nur eine kleine Verbesserung gegenüber dem alleinstehenden CNN.

2.4 Zusammenfassung

In der Literatur wurden sehr unterschiedliche, aber durchwegs sehr interessante Anwendungsfälle bearbeitet und beschrieben. Um hier bereits kurz auf die Ziele der Arbeit zurückzugreifen, wird hier eine grobe Übersicht zu Wide and Deep Learning gegeben, sowie kurz auf mögliche Chancen und Probleme eingegangen.

2.4.1 Was ist Wide and Deep Learning?

Wide and Deep Learning ist ein spannender Ansatz, dessen Grundidee einleuchtend wirkt. Er verbindet flache, Modelle, die “nahe“ an den Daten sind und direkte Zusammenhänge lernen, mit tiefen Modellen, welche für die Abstraktion und höhere Relationen verantwortlich sind.
Rückblickend auf die Literatur kann aber nicht genau abgegrenzt werden, was Wide and Deep Learning tatsächlich ist. Es wurde in verschiedenen Veröffentlichungen auf Wide and Deep Learning verweiesen, die Architekturen unterscheiden sich jedoch stark. Während der ursprüngliche Entwurf von Google mit einem linearen Modell und einem DNN relativ einfach gehalten ist, gehen viele der nachfolgenden Modelle einen Schritt weiter und implementierten ausgefallenere Methoden.

2.4.2 Mögliche Chancen und Probleme

Wide and Deep Learning eröffnet viele neue Möglichkeiten, dies bringt natürlich Chancen, wie auch mögliche Problem mit sich.
Einerseits können komplexere Modelle erstellt werden, welche Probleme noch genauer nachbilden könnten. Es gilt, viele verschiedene Kombinationen auszuprobieren, die Vorteile kennenzulernen und so zu verstehen, welche Kombinationen in welcher Form gut funktionieren können. Durch die verschiedenen Komponenten könnte es noch wichtiger werden, die richtigen Daten im richtigen Format durch die richtigen Algorithmen zu verarbeiten.
Andererseits könnte dieser Vorteil auch gleichzeitig das grösste Problem darstellen: Mit noch komplexeren Modellen und noch ausgefalleneren Variationen kann versucht werden, Modelle zu entwickeln, die nur wenig bessere Resultate erzielen. Daher ist hier wichtig, auch immer einfache Modelle beizuziehen. So können die Ergebnisse entsprechend abgewägt werden und zu komplizierte Modelle, welche viel Trainingsdaten und Rechenzeit benötigen, vermieden werden.

3 Framework

In den folgenden Kapiteln wird das erstellte Framework genauer beschrieben. Hierfür werden zuerst die Ansätze und Überlegungen ausgeführt, welche zu den Entscheidungen geführt haben, das Framework in der vorliegenden Form zu implementieren. Dabei werden auch verwendete Bibliotheken und Online-Quellen vorgestellt. Anschliessend wird das entwickelte Framework beschrieben und an einem Praxisbeispiel vorgestellt. Als letzter Abschnitt erfolgt ein kurzes Fazit zur Implementierung.

3.1 Ansätze und Überlegungen

Als Basis für die anschliessende praktische Umsetzung des Frameworkes mussten zuerst ein paar Entscheidungen getroffen. Diese basieren einerseits auf der behandelten Literatur, andererseits auf eigenen Überlegungen. Auf diese Entscheidungen wird in den ersten beiden Abschnitten eingegangen. In einem dritten Abschnitt wird auf verwendete Online-Quellen verwiesen, welche für die Umsetzung des Frameworks sehr hilfreich waren.

3.1.1 Ansätze aus der Theorie

Beim Studium der im Kapitel 2, Grundlagen und Literatur, aufgeführten Unterlagen kamen viele Ansätze auf, welche im Kontext dieser Arbeit durchaus Sinn machen. Folgend werden einige Ansätze beschrieben, welche gedanklich in den Entwicklungsprozess einflossen, aber leider nicht alle umgesetzt werden konnten.

Convolutional Neural Nets für Text Classification

Ein zentraler Ansatz, welcher aus zahlreichen Veröffentlichungen im Bereich der Text Classification erwähnt wurde, ist die Anwendung von CNN's. Im Vergleich zum Wide and Deep Ansatz von Cheng et al., welcher mit einem einfachen DNN arbeitete, ist eine Umsetzung mit einer CNN-Variante daher sicher miteinzubeziehen [Che+16]. Zudem lässt sich die von Conneau et al. ausgeführte Analogie sehr gut nachvollziehen: Was bei Bildern mit Pixeln gut funktioniert, sollte theoretisch auch bei Texten entsprechend mit Zeichen oder Wörtern gut funktionieren [Con+16].

Neurale Netze auf Zeichen- und Wortbasis

Bei CNN's werden im Bereich der Text Classification verschiedene Ansätze gefahren. So gibt es, neben der geschichtlichen Entwicklung, verschiedene Gründe, die eine Anwendung auf einzelne Wörter rechtfertigen. So können diese einfach mit bestehenden Dictionaries abgeglichen werden, es können Stopwörter definiert werden

und die Wörter können mit semantischen Informationen angereichert werden.
Je nach Text bringt aber auch eine Betrachtung auf Zeichenebene durchaus Vorteile. So wäre ein solches Netz wahrscheinlich weniger anfällig gegenüber Schreibfehlern. Zudem werden vor allem in online erfassten Texten immer häufiger Sonderzeichen wie z.B. Emojis verwendet, welche zeichenbasiert besser verarbeitet werden könnten. Zudem ist hier auch noch auf eine Schweizer Gegebenheit hinzuweisen: Mit dem Schweizerdeutsch, für welches keine fixen Regeln existieren, könnte ein zeichenbasierter Ansatz allenfalls besser zurechtkommen, da Wörter in verschiedenen Regionen unterschiedlich geschrieben werden. Als letzter Punkt soll auch hier nochmals auf die Analogie mit den Bilddaten hingewiesen werden: Schlussendlich müsste hier das Zeichen mit dem Pixel als unterstes Element gleichgestellt werden, was im Bereich der Bildverarbeitung den grossen Fortschritt gebracht hat.

Anreicherung der Daten mit semantischen Informationen

Der von Burel et al. vorgestellte Ansatz, dass Texte mit semantischen Informationen angereichert werden, erschien, insbesondere im Rahmen von Wide and Deep Learning, sehr naheliegend [BSA17]. Bei Datensets, welche beispielsweise aus dem journalistischen Umfeld stammen, könnte eine Anreicherung mit semantischen Informationen durchaus Sinn machen. So werden z.B. häufig Namen bekannter Personen oder Orte genannt, welche ohne semantische Informationen schwierig einzuordnen sind.
Auch wenn man nochmals die Analogie des Erinnerns und Generalisierens beizieht, können die semantischen Informationen durchaus auch als Schritt Richtung Generalisierung betrachtet werden, da unterschiedliche Wörter mit gleicher oder ähnlicher Semantik dadurch entsprechend bearbeitet werden können.

3.1.2 Weitere Ansätze und Ideen

Neben der Theorie gab es auch weitere Ansätze, resp. Entscheidungen, welche es zu treffen galt. Diese werden hier kurz beschrieben.

Tensorflow vs. Keras

In einer ersten Version wurde das Ziel verfolgt, das Framework mit Tensorflow aufzubauen. Die Überlegung war, die bereits von Google zur Verfügung gestellten Funktionen zu Wide and Deep Learning zu erweitern. Es stellte sich jedoch rasch heraus, dass dieser Weg sehr kompliziert werden würde. Tensorflow stellt spezifische Ansprüche an z.B. Datenformate, welche schwierig vollständig abzudecken sind. Es wurde daher entschieden, Keras genauer zu analysieren, welches dem Anwender viele Aufgaben abnimmt und damit flexibler eingesetzt werden kann.
Rasch zeigte sich, dass die Anforderungen auch durch Keras abgedeckt werden können. Keras besteht grob aus zwei Bereichen: den Sequential Modellen und der Functional API. Die Sequential Modelle sind sehr einfach in der Anwendung und daher auch weiter verbreitet. Mit der Functional API können aber auch komplexere Modelle erstellt werden. Daher wurde entschieden, bei der Umsetzung dieses Wide and Deep Learning Frameworks auf die Functional API von Keras zu setzen.

Funktionsorientiert vs. Objektorientiert

Nachdem mit der Wahl der zugrunde liegenden Technologie bereits ein Grundsatzentscheid getroffen wurde, galt es, auch bzgl. der Architektur des Frameworks eine Entscheidung zu treffen: Sollte das Framework funktions- oder objektorientiert aufgebaut werden? Für beide Ansätze gab es gute Gründe dafür, wie auch dagegen. Mit einem funktionsorientierten Ansatz kann man eine grosse Anzahl an bereits vordefinierten Modellen direkt zur Verfügung stellen. Zudem können Funktionen definiert werden, welche auch mit anderen Keras-Modellen verwendet werden können. Andererseits wäre die Anwendung des Frameworks weniger aufgeräumt: Für jede Funktion müssten immer alle benötigten Objekte übergeben werden, was eine Anwendung rasch sehr unübersichtlich machen könnte.
Der objektorientierte Ansatz wiederum würde es erlauben, ein tatsächliches Wide and Deep Model als eigene Klasse, mit eigenen spezifischen Funktionen, zur Verfügung zu stellen, was die Anwendung um einiges vereinfachen könnte. Allerdings würde es schwieriger werden, dem Anwender die gleichen Freiheiten wie bei einem funktionsorientierten Ansatz zu bieten.
Um das Beste aus beiden Welten übernehmen zu können, wurde ein Mischansatz gewählt: Es soll eine Klasse für das Wide and Deep Modell verfügbar gemacht werden und dazu eine Art Helper, welcher verschiedene Funktionen, sowohl für das eigene Modell, wie auch für übrige Keras Modelle, beinhaltet.

Kombination von Modellen vs. Kombination von Layern

Die Hauptherausforderung des Framework war es, die beiden Bereiche Wide und Deep geeignet miteinander zu kombinieren. Auch dafür kamen zwei Ansätze in Frage: Einerseits können in Keras zwei unabhängige Modelle erstellt werden, ein Deep Modell und ein Wide Modell. In einem weiteren Schritt können die beiden Modelle in einem dritten Modell kombiniert werden [Rob18]. Dies hat Vor- und Nachteile: Einerseits stehen die beiden separaten Modelle zur Verfügung und können beispielsweise einzeln vortraininert werden oder direkt für eine Ablationsstudie verwendet werden. Andererseits ist man bei der Zusammenführung der beiden Komponenten stark eingeschränkt: Die Anzahl Outputs der beiden Modelle muss beispielsweise bereits mit der Anzahl vorherzusagenden Klassen übereinstimmen. Im Falle einer binären Klassifikation kann also nur jeweils ein Wert in den gemeinsamen Layer einfliessen, was sich nachteilig auswirken könnte.
Die andere Lösung wäre, ein einziges gemeinsames Modell aufzubauen, welches mit zwei Inputs für Wide und Deep arbeitet, aber deren restliche Architektur unabhängig ist. So können verschiedene Ansätze evaluiert werden, z.B. in dem man unterschiedliche Mengen an Werten aus den beiden Komponenten in den gemeinsamen Layer einfliessen lässt.
Auch hier wurde entschieden, beide Möglichkeiten bereit zu stellen. Zuerst wurde aufgrund der Einfachheit der Ansatz unterschiedlicher Modelle implementiert. Aufgrund der erwähnten Nachteile wurde diese Funktion aber ausgelagert und in der eigentlichen Wide and Deep Modell Klasse wird der Ansatz des einzelnen Modelles gefahren, welcher mehr Freiheit zulässt. Das Wide and Deep Modell soll jedoch die Möglichkeit bieten, daraus direkt ein Modell der einzelnen Komponente abzuleiten.

3.1.3 Nützliche Tutorials und Online-Quellen

Während der Entwicklung des Frameworks kamen immer wieder Fragen bzgl. Umsetzung und Lösungsmöglichkeiten auf. Im Zuge der Programmierarbeiten wurden daher auch diverse Online-Quellen konsultiert. Die Wichtigsten sind hier kurz beschrieben.

Predicting the price of wine

Das Tutorial "Predicting the price of wine with the Keras Functional API and TensorFlow" von Robinson hat sich als Basis für die ersten Implementierungsschritte sehr gut geeignet. Einerseits was er zentral für die Entscheidung, von einer sehr technischen Tensorflow Umsetzung abzusehen und auf die stark vereinfachte Keras Functional API zu setzen. Gleichzeitig wählte auch Robinson ein Wide and Deep Learning Ansatz, womit bereits ein gutes Beispiel vorhanden war, wie eine solche Anwendung ohne Standardklasse von Tensorflow aussehen könnte [Rob18].
Interessant am Tutorial war zudem, dass zuerst zwei gänzlich unabhängige Modelle erstellt wurden: ein sehr flaches Netz, welches als Inhalt die Wörter als Bag of Words erhält und als Wide Komponente genutzt wird, sowie ein etwas tieferes Netz, welches die Embeddings verarbeitet und als Deep Komponente eingesetzt wird.
Erst wenn die beiden Modelle vorhanden sind, werden sie in einem dritten Schritt zusammengeführt:

```
1 merged_out = layers.concatenate([wide_model.output, deep_model.output])
2 merged_out = layers.Dense(1)(merged_out)
3 combined_model = Model(wide_model.input + [deep_model.input], merged_out)
4 combined_model.compile(loss='mse',optimizer='adam', metrics=['accuracy'])
```

Machine Learning Mastery

Brownlee zeigt auf seinem Blog "Machine Learning Mastery" detaillierte Lösungen für verschiedene, meist sehr spezifische, Herausforderungen. Viele dieser Herausforderungen galt es auch während dieser Arbeit zu überwinden, wobei die meisten mit Hilfe dieses Blogs gelöst werden konnten [Bro18].

Practical Text Classification With Python and Keras

Der Artikel "Practical Text Classification With Python and Keras" von Janakiev half insbesondere zu Beginn der Implementierung, in das Thema Text Classification mit Keras einzusteigen. Dank der klaren Schritt für Schritt Anleitung, inklusive CNN, konnte damit rasch ein erstes, funktionierendes Model erstellt werden. Insbesondere die einfache Beschreibung von den verschiedenen Embeddings sowie Evaluationen waren sehr hilfreich [Jan18].

3.2 Aufbau

Aufgrund der getroffenen Entscheide wurde ein aus zwei Bereichen bestehendes Framework implementiert. Einerseits gibt es die Klasse `WideDeepModel`, welche

im ersten Teil beschrieben wird. Sie verhält sich wie ein Keras-Model und bietet die gleichen Grundfunktionalitäten an. Anschliessend wird der `WideDeepHelper` vorgestellt. Dabei handelt es sich um eine Sammlung an Funktionen, welche im Rahmen der Entwicklung implementiert wurden und den Einsatz von Wide and Deep Modellen vereinfachen soll. Danach werden die Funktionalitäten im Rahmen eines Beispiels demonstriert. Zum Schluss wird noch auf mögliche Verbesserungen des Frameworks eingegangen.

3.2.1 WideDeepModel

Die Klasse `WideDeepModel` orientiert sich an den Modellen von Keras und bietet daher ebenfalls die gängigen Funktionen `fit`, `evaluate` und `predict` an. Damit soll eine gewisse Grundkompatibilität mit anderen Keras-basierten Funktionalitäten sichergestellt werden. Dazu werden vier weitere Funktionen zur Verfügung gestellt: `getWideModel`, `getDeepModel`, `full_binary_evaluation` und `log_evaluation`. Die Funktionen werden in den folgenden Abschnitten detaillierter beschrieben.

`WideDeepModel`

Die Klasse `WideDeepModel` ist die Basis des Framework und steht für das eigentliche Model. Sie basiert auf den Input- und Output-Layers der beiden Komponenten. Diese können selber definiert werden oder anhand der im `WideDeepHelper` verfügbaren Funktionen generiert werden. Weiter kann die Anzahl der zu bestimmenden Klassen (Standard: binäre Klassifikation) sowie Optimizer, Loss-Funktion und die zu messenden Kennzahlen festgelegt werden. Dabei stehen einem die Möglichkeiten von Keras offen, es kann also mit der normalen Bezeichnung `'adam'` gearbeitet werden oder es kann selber ein Optimierer definiert und übergeben werden. In der Funktion `_build_model` werden dann die definierten Layer zu einem Modell zusammengeführt.

```
1  def __init__(self, wide_in_layer, deep_in_layer, wide_out_layer,
       deep_out_layer, num_of_classes = 1,
2                  optimizer = 'adam', loss = 'binary_crossentropy',
3                  metrics = ['accuracy']):
4          """
5          Initialization for the Character Level CNN model.
6
7          Args:
8              wide_in_layer (keras.layers): Input layer of wide part
9              deep_in_layer (keras.layers): Input layer of deep part
10             wide_out_layer (keras.layers): Output layer of wide part
11             deep_out_layer (keras.layers): Output layer of deep part
12             num_of_classes (int): Number of classes in data. default: 1
13             optimizer (str): Training optimizer. default: 'adam'
14             loss (str): Loss function. default: 'binary_crossentropy'
15             metrics (list[str]): metrics to observe. default: ['accuracy
       ']
16         """
17         self.wide_in_layer = wide_in_layer
18         self.deep_in_layer = deep_in_layer
19         self.wide_out_layer = wide_out_layer
20         self.deep_out_layer = deep_out_layer
21         self.num_of_classes = num_of_classes
22         self.optimizer = optimizer
23         self.loss = loss
24         self.metrics = metrics
```

```
25          self._build_model()  # builds self.model variable
```

`fit`

Die `fit`-Funktion ist für das Training des Modells verantwortlich. Hier werden die Testdaten festgelegt sowie weitere Trainingsparameter, wie z.B. die Anzahl Epochen und Batch-Grösse. Für das `WideDeepModel` wird ein Array in der Form `[wide_train + deep_train]` als Trainings-Datensatz erwartet.

`evaluate` & `predict`

Diese beiden Funktionen, welche auf das trainierte Modell angewendet werden können, basieren auf den entsprechenden Keras-Funktionen und übernehmen einen Grossteil der entsprechenden Funktionalitäten.

`getWideModel` & `getDeepModel`

Um einfach ein Modell der Wide-, resp. Deep-Komponente ableiten zu können, stehen entsprechende Funktionen zur Verfügung. Diese initiieren ein Modell anhand der definierten Layers und verwenden die im `WideDeepModel` definierten Parameter.

`full_binary_evaluation`

Um rasch eine grobe Aussage zum Modell treffen zu können, steht eine Funktion zur Verfügung, welche automatisch verschiedene Messgrössen berechnet und Diagramme generiert. Neben dem Trainingsdaten und -labels kann der Threshold festgelegt werden, der für die Kategorisierung verwendet werden soll.

`log_evaluation`

Neben der Möglichkeit, mit `full_binary_evaluation` ein einzelnes Modell genauer untersuchen zu können, bietet diese Funktion die Möglichkeit, die berechneten Messgrössen in einer Datei abzuspeichern. Damit können rasch mehrere Modelle evaluiert werden, und dann anhand der gespeicherten Informationen verglichen werden.

3.2.2 WideDeepHelper

Im Kontext von Text Classification werden viele Funktionalitäten immer wieder benötigt, sei es, um gewisse Features zu generieren, rasch ein einfaches Modell zu erstellen oder ein solches zu evaluieren. Die Idee des `WideDeepHelpers` ist es, diese Funktionen so einfach wie möglich zur Verfügung zu stellen, so dass sie einfach in verschiedenen Situationen eingesetzt werden können. Die Funktionen werden folgend in drei Bereichen kurz vorgestellt.

Feature Engineering

Das Feature Engineering kann jeweils viel Zeit in Anspruch nehmen. Die Idee dieser Funktionen ist es, diese Arbeit etwas zu erleichtern und beispielsweise automatisch einen Bag of Words zu erzeugen. Damit soll einerseits die Entwicklungszeit reduziert werden, andererseits soll so auch eine breitere Sammlung an Features zur Verfügung

stehen, was den Einsatz von etwas innovativeren Modellen vereinfachen und damit auch zu besseren Ergebnissen führen soll. Einige der Funktionen werden hier vorgestellt.

`getfeatures_bow`
Insbesondere die Wide-Komponente arbeitet häufig mit der Repräsentation der Wörter als Bag of Words. Daher wird eine Funktion zur Verfügung gestellt, die anhand von Trainings- und Testdatensatz automatisch diese Features aufbereitet und zurück gibt.

`getfeatures_embeddings`
Word Embeddings werden eingesetzt, um Wörter und deren Relationen so darzustellen, dass sie mathematisch verarbeitet werden können. Keras bietet hier bereits sehr gute Funktionalitäten an, die in dieser Funktion noch weiter vereinfacht wurden.

`getembeddings`
Gute Embeddings zu trainieren erfordert viel Zeit und gute Trainingsdaten. Daher wird häufig auf bereits vortrainierte Word Embeddings zurückgegriffen, beispielsweise word2vec oder GloVe. Mit `getembeddings` können diese einfach ins Modell integriert werden.

Layers und Modelle

Während der Implementation wurden immer wieder verschiedene Modelle ausprobiert. Viele Architekturen haben sich zudem in mehreren Beispielen im Internet etabliert. Damit diese Modelle in Zukunft nicht selber entwickelt werden müssen, wurden diese direkt in das Framework integriert.
Hier werden die wichtigsten Architekturen vorgestellt, welche jeweils als `getlayer`- wie auch als `getmodel`-Funktion zur Verfügung stehen. Die Layer-Funktionen geben den Start- und End-Layer zurück, wobei die Grösse des Outputs festgelegt werden kann. Die Modell-Funktionen wiederum sind von der Anzahl der Klassen abhängig. Zum Schluss wird noch auf die Funktionen `combine_widedeep_model` und `getmodel_ensemble` eingegangen, welche zur Zusammenführung zweier Modelle in ein Wide and Deep Modell oder ein Ensemble verwendet werden können.

`getlayers_wide` / `getmodel_wide`
Mit dieser Funktion wird eine einfache Wide-Komponente generiert. Diese besteht aus einem Input Layer, einem Dense Layer als Hidden Layer sowie einem Dense Layer als Output Layer. Inputgrösse, Outputgrösse und die Aktivierungsfunktion können über Parameter festgelegt werden.

`getlayers_deep_simplednn` / `getmodel_deep_simplednn`
Im Keras-Tutorial für Text Classification kommt ein kleines DNN zum Einsatz, welches auf dem IMDB-Datensatz bereits gute Resultate erzielt. Da dieses einerseits aufgrund der Einfachheit rasch auf neue Daten angewendet werden kann und sich andererseits auch gut als Baseline-Modell für den IMDB-Datensatz eignet, wurde es direkt in das Framework integriert [Goo18b].

`getlayers_deep_simplecnn` / `getmodel_deep_simplecnn`
Neben dem “Simple DNN“ wurde auch ein sehr einfaches CNN implementiert. Auch hier wurde auf eine Architektur zurückgegriffen, welche sehr einfach ist, aber in einem Anwendungsfall schon gute Resultate liefern konnte [Jan18].

`getlayers_deep_simplernn` / `getmodel_deep_simplernn`
Das “Simple RNN“ vervollständigt die Gruppe der einfachen Modelle. Es handelt sich wiederum um ein sehr einfaches Modell, das bereits gute Ergebnisse erzielen konnte [Goo18a].

`getlayers_deep_dynamicdnn` / `getmodel_deep_dynamicdnn`
Neben den fix bereitgestellten “Simple“-Modellen steht auch die Möglichkeit bereit, anhand eines Übergabeparameters ein dynamisches DNN zu erzeugen. Dabei können die Anzahl Layers sowie Anzahl Nodes festgelegt werden. Zusätzlich kann definiert werden, ob nach einem Dense-Layer jeweils ein Dropout Layer hinzugefügt werden soll.

`combine_widedeep_model`
Die Idee hinter dieser Funktion ist es, zwei beliebige Modelle, normalerweise ein Wide und ein Deep Modell, zu kombinieren, wobei die jeweils benötigten Inputs als Übergabeparameter festgelegt werden. Als optionaler Übergabeparameter wird die Ausgabegrösse festgelegt, mit welcher die Anzahl zu bestimmenden Kategorien definiert wird. Dieser Wert ist normalerweise auf 1 gesetzt, was einer binären Kategorisierung entspricht.

`getmodel_ensemble`
Diese Funktion erstellt ein Ensemble aus den ihr übergebenen Wide- und Deep-Komponente. Deren Outputs werden in einem Average-Layer zusammengefasst. Ein Model Ensemble kann beispielsweise dazu verwendet werden, das Resultat eines Wide and Deep Modelles besser einordnen zu können.

Evaluation

Im Bereich der Evaluation stehen vier verschiedene Funktionen zur Verfügung, welche verschiedene Diagramme generieren. Diese werden normalerweise durch `full_binary_evaluation` aufgerufen, können so aber auch separat genutzt werden, z.B. zum Erzeugen einer Konfusionsmatrix oder ROC-Kurve.
Zusätzlich werden auch hier die Funktionen `full_binary_evaluation` und `log_evaluation` bereitgestellt, welche ein Modell als Parameter entgegen nehmen. So soll auch für andere Modelle, beispielsweise für das Modell der Wide- oder Deep-Komponente, im Rahmen einer Ablationsstudie dieselbe Evaluationsübersicht erzeugt werden können.

3.2.3 Anwendungsbeispiel

Das Zusammenspiel der verschiedenen Komponenten ist ein zentraler Faktor eines guten Frameworks. Es ist wichtig, dass sie gut ineinander greifen und sich ein flüssiges Bild ergibt. Dies soll am Beispiel des Keras-Datensatzes demonstriert werden.

Dabei wird nur auf die technische Umsetzung eingegangen. Details zum Datensatz, Ergebnisse sowie Evaluation sind im Kapitel 4.2, Anwendungsfall 1: IMDB, zu finden.

In einem ersten Schritt werden die bereitgestellten Daten in Python geladen. Dieser Schritt wird noch nicht gross vom Framework abgedeckt. Es wird davon ausgegangen, dass mit vier Objekten gearbeitet wird: Je ein Trainings- und Textset für Texte sowie Labels. Diese können normalerweise gut anhand verfügbarer Texte abgeleitet werden.

```
from mas.final.WideDeepModel import WideDeepModel
import mas.final.WideDeepHelper as WideDeepHelper
import pandas as pd

train_file = "C:/BAS/python/mas/IMDB/imdb_train.csv"
test_file = "C:/BAS/python/mas/IMDB/imdb_test.csv"

CSV_COLUMNS = ['sentence', 'label']

train_df = pd.read_csv(train_file, delimiter = '|', header = None, names
    = CSV_COLUMNS)
test_df = pd.read_csv(test_file, delimiter = '|', header = None, names =
    CSV_COLUMNS)

# Train and test features
text_train = train_df['sentence']
text_test = test_df['sentence']

# Train and test labels
labels_train = train_df['label']
labels_test = test_df['label']
```

In einem nächsten Schritt werden die Features erstellt. Dafür werden zuerst deren Eckpunkte festgelegt. In diesem Beispiel wird ein Bag of Words für die Wide-Komponente verwendet, während die Deep-Komponente mit Word Embeddings arbeitet.

```
# Set Parameters
wide_input_size = 30000 # vocab size
wide_output_size = 1 # number of values from deep part to combined layer
deep_input_size = 256 #words per text
deep_embedding_input = 30000 # vocab size
deep_embedding_output = 100 # vocab size
deep_output_size = 1 # number of values from deep part to combined layer

# Define features for our wide part with the WideDeepHelper
bow_train, bow_test = WideDeepHelper.getfeatures_bow(text_train,
    text_test, vocab_size=wide_input_size)

# Define features for our deep part with the WideDeepHelper
embed_train, embed_test = WideDeepHelper.getfeatures_embeddings(
    text_train, text_test, vocab_size=deep_embedding_input,
    max_seq_length=deep_input_size)
```

Anschliessend wird die Architektur der verschiedenen Komponenten festgelegt. Hier werden beide Möglichkeiten aufgezeigt: Für die Wide-Komponente werden die Layers mit Hilfe des `WideDeepHelpers` generiert, für die Deep-Komponente werden sie direkt in der Anwendung definiert.

```
# Define layers for our wide part with the WideDeepHelper
wide_input, wide_output = WideDeepHelper.getlayers_wide(wide_input_size,
    wide_output_size)

# Define layers for our deep part with the functional API
deep_input = keras.layers.Input(shape=(deep_input_size,))
embedding = keras.layers.Embedding(deep_embedding_input,
    deep_embedding_output, input_length=deep_input_size)(deep_input)
pool =  keras.layers.GlobalAveragePooling1D()(embedding)
dense1 = keras.layers.Dense(16, activation='relu')(pool)
deep_output = keras.layers.Dense(deep_output_size, activation='sigmoid')(
    dense1)
```

Mit den erzeugten Layers kann nun das `WideDeepModell` aufgebaut und trainiert werden.

```
# Use Framework to generate the wide and deep model
combined_model = WideDeepModel(wide_input, deep_input, wide_output,
    deep_output, num_of_classes = 1, optimizer = 'adam', loss = '
    binary_crossentropy', metrics = ['accuracy'])
combined_fit = combined_model.fit(bow_train, embed_train, labels_train,
    epochs=2, batch_size=512, validation_split=0.1)
```

In einem letzten Schritt wird das trainierte Modell evaluiert. Da es sich bei diesem Modell um einen binären Classifier handelt, können wir die bereitgestellte Funktion verwenden, welche automatisch verschiedene Messgrössen berechnet sowie die wichtigsten Diagramme darstellt.

```
combined_model.full_binary_evaluation([bow_test, embed_test], labels_test
    , class_labels=["toxic", "non-toxic"], class_threshold = 0.5,
    normalize=True)
```

Weiter könnten nun anhand des `WideDeepModel`'s die Wide- oder Deep-Komponente als eigene Modelle erzeugt werden. Diese könnten anschliessend separat trainiert und mit der gleichen Funktion, welche auch im `WideDeepHelper` zur Verfügung steht, evaluiert werden.

3.2.4 Verbesserungsmöglichkeiten

Ein Framework, insbesondere in einem so umfassenden Anwendungsbereich wie Machine Learning, hat natürlich immer unzählige Möglichkeiten, noch erweitert und verbessert zu werden. Daher werden hier nur die Wichtigsten beschrieben, welche sich auch gut in den Kontext der Arbeit einordnen lassen.

Weitere Funktionen zur Generierung von Features

Mit guten Features kann ein Modell rasch verbessert werden. Jedoch benötigt gutes Feature Engineering auch immer sehr viel Zeit. Daher wäre es sicher interessant, hier noch mehr Möglichkeiten anzubieten.
Erste Möglichkeiten zur automatischen Generierung von semantischen Informationen, wie sie im Semantic Wide and Deep zur Anwendung kommen, konnten bereits implementiert werden, befinden sich aber noch nicht auf einem wirklich nutzbaren

Level. Des Weiteren wäre eine Erweiterung interessant, dass auch mehr mit zeichenbasierten Features und Modellen gearbeitet werden kann. Die bisher verfügbaren Funktionen beziehen sich jeweils auf Wörter als Basis, was im Kontext der Arbeit auch mehr Möglichkeiten (Embeddings, semantische Informationen) erlaubte.

Besseres Angebot bei Layern/Modellen

Die mit den bereitgestellten Funktionen verfügbaren Layer, resp. Modelle sind relativ starr und bieten wenig Spielraum. Hier wäre es sicherlich interessant, beispielsweise die Anzahl Layers, Nodes, etc. via Parameter stärker beeinflussen zu können. Es ist jedoch wichtig, eine gute Balance zwischen rascher Anwendung und Konfigurationsfreiraum zu finden.
Zudem wäre es interessant, auch einige komplexere Modelle einzubinden. Die momentan verfügbaren Modelle sind relativ einfach, dadurch könnten die Möglichkeiten rasch ausgeschöpft sein.

Funktion für den direkten Vergleich von Modellen

Momentan bietet das Framework lediglich die Möglichkeit, ein spezifisches Modell detaillierter zu analysieren. Gerade im Bereich Wide and Deep Learning ist es aber durchaus interessant, verschiedene Modelle untereinander zu vergleichen, sei dies Wide vs. Deep vs. Wide and Deep oder unterschiedliche Konfigurationen verschiedener Komponenten. Daher wäre es sicher eine interessante Verbesserungsmöglichkeit, diesen Vergleich direkt anzubieten, wenn möglich mit einer dynamischen Anzahl an Modellen. Ein erster Schritt ist mit dem Schreiben der Resultate in eine Datei bereits gemacht.

3.3 Fazit

Die Entwicklung des Frameworks war komplizierter als erwartet und nahm sehr viel Zeit in Anspruch. Verschiedene Versionen wurden immer wieder hinterfragt, verbessert, neu ausgerichtet und erneut hinterfragt. Diverse Ideen kamen auf, wurden eingearbeitet und teilweise wieder verworfen. Dennoch oder gerade deshalb bietet das Framework mit dem zur Verfügung gestellten `WideDeepModel` Möglichkeiten an, rasch einfache Wide and Deep Modelle zu erstellen. Damit wird jedoch nur ein kleiner Teil der Aufgabe übernommen. Richtig gute Modelle erfordern häufig komplexere Modelle, welche sehr flexibel in das WideDeepModel eingearbeitet werden können, momentan aber noch nicht durch den `WideDeepHelper` bereitgestellt werden.

4 Evaluation

Um das Konzept von Wide and Deep Learning im Bereich von Text Classification zu prüfen, wurden verschiedene Modelle auf drei unterschiedliche Datensets evaluiert. Im ersten Abschnitt wird auf das Vorgehen der Evaluation eingegangen: Welche Modelle wurden angewandt, welche Messgrössen eingesetzt, etc. Anschliessend werden die verschiedenen Datensets sowie die erzielten Ergebnisse dokumentiert. Im letzten Abschnitt werden die gesammelten Erkenntnisse zusammengeführt, um übergreifende Aussagen zu treffen und allfällige Schlüsse zu ziehen.

4.1 Vorgehen der Evaluation

Dieser Abschnitt beschreibt das Vorgehen während der Evaluation. Zuerst werden die verschiedenen Modelle vorgestellt, sowie aufgezeigt, in welchem Anwendungsfall sie zum Einsatz kommen. Anschliessend wird auf die verschiedenen Hyperparameter eingegangen, welche neben den Modellen ebenfalls evaluiert wurden. Zum Schluss wird definiert, anhand welcher Messgrössen die unterschiedlichen Modelle bewertet wurden.

4.1.1 Modelle

Bevor mit der Evaluation begonnen werden konnte, galt es, eine Entscheidung zu treffen: Soll der Fokus auf einem detaillierten Modell mit ausgereiften Features liegen, auf welchem unterschiedliche Parameter angewendet werden, um ein möglichst gutes Resultat zu erhalten, oder sollen eher viele, dafür einfachere, Modelle verglichen werden. Da es das Ziel der Arbeit ist, Wide and Deep Learning als Konzept zu untersuchen, wurde entschieden, dass der Fokus auf mehrere einfache Modellen gelegt werden soll. Damit sollen einerseits mehr Aspekte des Themas abgedeckt werden, andererseits können auch die verschiedenen Komponenten des entwickelten Frameworks besser miteinbezogen werden.

Zudem werden in den Evaluationen jeweils die Ergebnisse einer Wide-Komponente (Wide), einer Deep-Komponente (Deep), eines Ensembles der beiden (WD-Ens) sowie eines Wide and Deep Modelles (WD-Mod) verglichen. Die eingesetzten Komponenten werden folgend kurz vorgestellt. In der Tabelle 4.1, Übersicht Modelle, wird zusätzlich aufgelistet, in welchen Anwendungfällen die unterschiedlichen Deep-Komponenten zum Einsatz kommen.

Wide-Komponente: Single Dense-Layer

Für die Wide-Komponente wurde ein einzelner Dense-Layer mit einem Bag of Words verwendet, wie er durch das entwickelte Framework bereitgestellt wird. Als Aktivierungsfunktion kam der Standard des Frameworks zum Einsatz, “ReLu“. Für den Bag

Komponente	IMDB	GermEval	Kommentare
Simple DNN	X	X	X
Simple CNN		X	X
Simple RNN			X

Tab. 4.1: Übersicht Modelle

Parameter	Standartwert	Alternativen
Optimizer	Adam(lr=0.005, beta1=0.9, beta2=0.999, epsilon=1e-08, decay=0.0)	
Epochen	5	10, 15
Batch-Grösse	64	256
Embeddings	Initial	GloVe [PSM14]
Grösse des Vokabulars	10'000	5'000, 20'000, 30'000

Tab. 4.2: Übersicht Parameter

of Words wurden verschiedene Vokabular-Grössen getestet: 5'000, 10'00, 20'000 und 30'000 Wörter.

Deep-Komponente: Simple DNN

Das eingesetzte Simple DNN stimmt mit der im Keras-Tutorial vorgestellten Variante überein. In diesem Test kamen vier verschiedene Grössen für das Word Embedding zum Einsatz: 5'000, 10'000. 20'000 und 30'000. Zudem wurde eine Variante mit einem vortrainierten Word Embedding (GloVe) angewendet, bei welchem 10'000 und 30'000 Wörter berücksichtigt wurden.

Deep-Komponente: Simple CNN

Auch im Simple CNN kam die direkt durch das Framework bereitgestellte Architektur zum Einsatz. Damit sollte abgeschätzt werden können, wie sich ein CNN im Kontext von Wide and Deep Learning verhalten könnte.

Deep-Komponente: Simple RNN

Wie die beidenanderen Deep-Komponenten wurde auch dieses einfache RNN direkt aus dem Framework übernommen.

4.1.2 Parameter

Neben den beschriebenen Architekturen wurden auch unterschiedliche Parametrisierungen ausprobiert. Die verschiedenen Parameter, deren gewählter Standardwert sowie teilweise eingesetzte Variationen sind in der Tabelle 4.2, Übersicht Parameter, aufgelistet. Wurden Variationen geprüft, wird dies in der Evaluation entsprechend erwähnt.

4.1.3 Messgrössen

Resultate können, wie im Kapitel 2, Grundlagen und Literatur, aufgezeigt wurde, auf verschiedene Arten gemessen werden. Mit der Anpassung des Thresholds können die Ergebnisse rasch zugunsten der Precision, des Recalls oder auch der Accuracy beeinflusst werden. Um hier einen möglichst fairen Vergleich vorzunehmen, wurde die Macro Average F1-Score als Kern-Messgrösse gewählt. Einerseits wurde diese z.B. bei GermEval bereits eingesetzt und erlaubt so auch einen Vergleich mit anderen Ansätzen, andererseits belohnt sie ein ausgeglichenes Modell, in welchem nicht eine einzelne Klasse bevorzugt werden kann [WSR18].
Die Macro Average F1-Score wurde auch zur Wahl des jeweiligen Threshold-Wertes verwendet: Für den Vergleich verschiedener Modelle wurde pro Modell jeweils der Threshold gewählt, welcher den besten Macro Average F1-Score ergab (Als mögliche Threshold-Werte wurden jeweils Werte in 0.1 Schritten gewählt, also 0.1, 0.2, 0.3, ..., 0.9).

4.2 Anwendungsfall 1: IMDB

Aufgrund der einfachen Verfügbarkeit wurde zuerst eine Evaluation anhand des IMDB-Datensets durchgeführt, welches direkt durch Keras zur Verfügung steht. Dabei handelt es sich um User-Reviews in Englisch, welche auf IMDB veröffentlicht wurden [Ker].

4.2.1 Datenset & Aufbereitung

Die Datensammlung besteht aus 25'000 Reviews, welche in zwei Kategorien unterteilt sind: "positive" und "negative". Die einzelnen Texte sind in der verfügbaren Form bereits für Machine Learning aufbereitet: Die einzelnen Wörter wurden bereits durch Indexes ersetzt. So wurde das häufigste Wort durch 1 ersetzt, das zweithäufigste durch 2, usw. Unbekannte Wörter werden durch eine 0 repräsentiert [Ker]. Für eine, im Rahmen des Frameworks, einfachere Verarbeitung und realistischere Ausgangslage wurde die Textrepräsentation wieder dekodiert und die Texte inkl. Labels in zwei CSV-Dateien abgelegt, je eines für Training und für Test.

4.2.2 Modelle & Parameter

Da das von Keras vorgestellte Simple-DNN bereits auf dieses Datenset angewandt wurde und auch durch das Framework bereitgestellt wird, eignete es sich sehr gut für eine erste grobe Evaluierung [Goo18b]. Das Augenmerk bei diesem Anwendungsfall lag vorerst auf unterschiedlichen Parametern: Neben unterschiedlichen Grössen des Vokabulars für den Bag of Words und die Word Embeddings wurden die Modelle jeweils unterschiedlich lange trainiert. Damit sollte einerseits der Einfluss der Datenmenge beobachtet werden, andererseits auch das Zusammenspiel der Wide- und Deep-Komponenten bei unterschiedlich langem Training. Da es sich bei diesem Datenset um das einzige in englischer Sprache handelt, eignete es sich auch gut, um einmal vortrainierte Word Embeddings zu testen.

4.2.3 Ergebnisse

Das einfache DNN aus dem Keras Text Classification Tutorial erreichte bereits eine Accuracy von 87% [Goo18b]. In Tabelle 4.3, “Anwendungsfall 1, IMDB: Macro Average F1-Scores der eingesetzten Modelle“, sind die erreichten Macro Average F1-Scores ersichtlich. Die Accuracy liegt jeweils ca. 0.5 bis 1% darüber, womit diese Modelle etwas besser abschnitten als das DNN aus dem Tutorial. Lediglich das Deep-Modell erreicht bei wenigen Epochen etwas schlechtere Werte. Mit mehr Trainingsepochen steigt dieser Wert jedoch an, während die Wide-Komponente dann aufgrund von Over-Fitting etwas schlechter abschneidet.
Zudem können weitere Muster erkannt werden: (1) Mit grösserem Vokabular werden auch bessere Resultate erreicht. (2) Mit der Verwendung von GloVe scheint sich der Einfluss der Deep-Komponente zu verstärken, was sich in den Tests negativ auf das Ensemble und das Wide and Deep Modell auswirkt. (3) Das Ensemble und das Wide and Deep Modell liefern die besten Resultate.

Da mehr Epochen bessere Resultate zeigen und auch den Einfluss der Deep-Komponente verstärken, werden in den folgenden Evaluationen vermehrt mehr Epochen eingesetzt. Die Grösse der Batch-Size wird hingegen bei 64 belassen.

4.3 Anwendungsfall 2: GermEval

Als weitere Textsammlung standen die Texte der GermEval 2018 zur Verfügung. Dabei handelt es sich um eine Aufgabe, welche 2018 von verschiedenen Gruppen, grösstenteils Hochschulen und Universitäten, in Angriff genommen wurde. Die Teilnehmer haben ihre Resultate veröffentlicht sowie ihre Ansätze dokumentiert, weshalb sich dieses Datenset ebenfalls gut für einen Vergleich eignete [WSR18].

4.3.1 Datenset & Aufbereitung

Die Textsammlung besteht aus zwei Teilen, einem Trainingsset mit 5’009 Texten und einem Testset mit 3’398 Texten. Die Texte, welche in deutscher Sprache verfasst sind, wurden auf zwei Arten kategorisiert. Es gibt eine binäre Kategorie “OFFENSE“ und “OTHER“, sowie eine Kategorie mit mehreren Ausprägungen, in welcher die Kategorie “OFFENSE“ noch weiter unterschieden wird. In diesem Anwendungsfall wurde lediglich die binäre Klassifikation geprüft.
Die Daten standen als zwei CSV-Dateien zur Verfügung, je eines für Training und für Test.

4.3.2 Modelle & Parameter

Neben dem Simple-DNN kam in diesem Anwendungsfall auch ein Simple-CNN zum Einsatz. Zudem wurden die Modelle etwas länger trainiert, bis hin zu 15 Epochen. Die Idee war, damit die kleinere Datenmenge etwas zu kompensieren. Da dieses Datenset in deutsch ist, wurden keine vortrainieren Word Embeddings eingesetzt.

Modell & F1-Score	Wide	Deep	WD-Ens	WD-Mod
Simple DNN@5'000 vocab, 5 Ep.	88.65%	86.92%	**88.93%**	88.65%
Simple DNN@5'000 vocab, 10 Ep.	88.63%	88.05%	88.54%	**88.65%**
Simple DNN@5'000 vocab, 10 Ep., 256 batch	88.30%	85.685%	88.59%	**88.59%**
Simple DNN@10'000 vocab, 5 Ep.	88.98%	87.22%	88.99%	**89.01%**
Simple DNN@10'000 vocab, 5 Ep. mit GloVe	88.4%	88.14%	88.57%	**88.87%**
Simple DNN@10'000 vocab, 10 Ep.	88.54%	88.36%	88.60%	**88.75%**
Simple DNN@10'000 vocab, 10 Ep., 256 batch	88.57%	86.09%	**88.89%**	88.67%
Simple DNN@20'000 vocab, 5 Ep.	89.04%	87.11%	**89.08%**	88.93%
Simple DNN@20'000 vocab, 10 Ep.	**88.71%**	88.33%	88.64%	88.61%
Simple DNN@20'000 vocab, 10 Ep., 256 batch	88.63%	86.16%	**88.89%**	88.42%
Simple DNN@30'000 vocab, 5 Ep.	88.74%	87.24%	**89.13%**	89.00%
Simple DNN@30'000 vocab, 5 Ep. mit GloVe	**88.98%**	88.07%	88.46%	88.63%
Simple DNN@30'000 vocab, 10 Ep.	88.62%	88.35%	88.59%	**88.71%**
Simple DNN@30'000 vocab, 10 Ep. mit GloVe	**88.74%**	87.55%	87.66%	88.07%

Tab. 4.3: Anwendungsfall 1, IMDB: Macro Average F1-Scores der eingesetzten Modelle

Modell & F1-Score	Wide	Deep	WD-Ens	WD-Mod
Simple DNN@5'000 vocab, 10 Ep.	62.7.0%	0.00%	**64.69%**	60.96%
Simple DNN@10'000 vocab, 10 Ep.	63.52%	0.00%	**65.42%**	61.63%
Simple DNN@20'000 vocab, 10 Ep.	64.05%	0.00%	**65.93%**	62.44%
Simple DNN@30'000 vocab, 15 Ep.	65.86%	0.00%	**66.92%**	64.47%
Simple CNN@5'000 vocab, 5 Ep.	57.97%	0.00%	**59.06%**	57.10%
Simple CNN@30'000 vocab, 15 Ep.	65.76%	0.00%	**66.98%**	62.89%

Tab. 4.4: Anwendungsfall 2, GermEval: Macro Average F1-Scores der eingesetzten Modelle

4.3.3 Ergebnisse

Im Vergleich zu IMDB wurden klar schlechtere Resultate erreicht. Für die Deep-Komponente konnte kein brauchbares Modell gefunden werden, was Aussagen zum Model Ensemble und zum Wide and Deep Modell stark erschweren. Dies könnte auf zwei Faktoren zurückzuführen sein: (1) Es stehen zu wenige Daten zur Verfügung, um die Deep-Komponente zu trainieren. (2) Da das Modell die Wörter direkt, ohne Stemming, verwendet, könnte das Modell in der deutschen Sprache mehr Mühe haben, da Wörter in mehr unterschiedlichen Formen geschrieben werden wie beispielsweise im Englischen (Fälle, Konjunktionen, etc.).
Trotz den verzerrten Resultaten können einige Beobachtungen gemacht werden: (1) Das Wide and Deep Modell scheint stärker unter der schlechten Deep-Komponente zu leiden. Dies könnte auf die Tatsache zurückzuführen sein, dass die beiden Komponenten gemeinsam trainiert werden und damit auch die Wide-Komponente ausgebremst wurde. (2) Während das Model Ensemble sowie das Wide and Deep Modell mit dem Simple-CNN bei einem 5'000-Wort-Vokabular und 5 Epochen noch klar schlechter abschneidet als mit dem Simple-DNN, werden bei 30'000 Wörtern und 15 Epochen nahezu identische Resultate erzielt. Das Model Ensemble scheint davon mehr zu profitieren als das Wide and Deep Modell.
Die beste erreichte Macro Average F1-Score (Ensemble von ReLu-Layer mit Bag of Words plus Simple-CNN mit 30'000 Wort-Vokabular und 15 Epochen) beträgt 66.98%, was im Rahmen der GermEval Rang 36 von 51 ergeben würde [WSR18].

4.4 Anwendungsfall 3: Online-Kommentare

Der dritte und letzte Anwendungsfall basiert auf Kommentartexten von Webseiten der Tamedia AG. Es handelt sich um eine sehr grosse Textsammlung in Deutsch, welche intern zur Verfügung steht und bereits kategorisiert wurde. Die Texte wurden in zwei Kategorien aufgeteilt: "non-toxic", also Kommentare, welche veröffentlicht

werden können, und “toxic“, solche Kommentare, welche gegen gewisse Regeln verstossen und nicht veröffentlicht werden sollten.

4.4.1 Datenset & Aufbereitung

Für die Kategorisierung der Kommentare sind klare Regeln definiert, welche online nachgeschlagen werden können [Tag18]. Die Daten selber standen als Pickle-Dateien zur Verfügung: Eine Datei mit 509’675 positiven Kommentaren, eine zweite Datei mit 207’661 negativen Kommentaren. Für die Evaluation wurden daraus zwei unterschiedlich grosse, ausbalancierte Trainings-/Testsets erstellt: einmal mit je 50’000, einmal mit je 100’000 Kommentaren.

4.4.2 Modelle & Parameter

Anhand dieses Datensets wurde eine weitere Architektur für die Deep-Komponente eingesetzt. Neben den bereits verwendeten Simple-DNN und Simple-CNN wurde nun zusätzlich ein Simple-RNN evaluiert. Während die bisherigen Modelle jeweils auf einem Notebook ohne dedizierte Grafikkarte gerechnet wurden, wurden die auf dem grösseren Datenset basierenden Modelle nun auf einer GPU (nVidia GTX 1080 Ti) gerechnet.

4.4.3 Ergebnisse

Dank der grösseren Datenmenge konnte in dieser Evaluation auch wieder die Deep-Komponente miteinbezogen werden. Die Resultate des Ensembles und des Wide and Deep Modelles lagen wieder sehr nahe beieinander.
Weiter konnten folgende Punkte beobachtet werden: (1) Ab 15 Epochen erreichte das Simple-DNN sehr gute Resultate. Sowohl das Ensemble wie auch das Wide and Deep Modell scheinen aber durch die Wide-Komponente gebremst zu werden. (2) Mit dem Simple-RNN konnte bei kleinem Vokabular trotz sehr langer Laufzeit keine Kategorisierung erreicht werden. Daher wurde das RNN nicht weiter verfolgt.

Auf einige detailliertere Auswertungen soll hier am Beispiel der Simple-DNN Variante mit einem Vokabular von 30’000 Wörtern, trainiert mit 100’000 Texten (je 50’000) während 15 Epochen, noch eingegangen werden.

Vergleicht man die Entwicklung der Accuracy über die Epochen des Wide- und des Deep-Modelles (Abbildung 4.1, Vergleich: Entwicklung Accuracy, Wide-Komponente vs. Deep-Komponente) sieht man klar, dass das Wide-Modell die optimale Anzahl Epochen rascher erreicht (bei ca. 8 Epochen), als dies beim Deep-Modell der Fall ist (bei 13 Epochen, evtl. sogar mehr).
Betrachtet man die Accuracy-Kurve des Wide and Deep-Modelles, fällt auf, dass diese sehr ähnlich aussieht wie diejenige des Wide-Modelles (Abbildung 4.2, Vergleich: Entwicklung Accuracy & Loss, Wide and Deep Modell). Der Verlust durch Over-Fitting kann in diesem Beispiel also nicht durch den zusätzlichen Einfluss des Deep-Modelles Wett gemacht werden.

Modell & F1-Score	Wide	Deep	WD-Ens	WD-Mod
Simple DNN@5'000 vocab, 5 Ep.	68.33%	68.03%	**68.54%**	68.43%
Simple DNN@5'000 vocab, 5 Ep.	68.90%	65.18%	**69.44%**	68.80%
Simple DNN@30'000 vocab, 5 Ep.	69.80%	63.30%	**70.12%**	69.85%
Simple DNN@30'000 vocab, 10 Ep.	69.96%	68.96%	70.00%	**70.14%**
Simple DNN@30'000 vocab, 15 Ep.	68.84%	**69.81%**	69.12%	69.06%
Simple DNN@30'000 vocab, 15 Ep.	69.21%	**70.51%**	69.07%	69.48%
Simple CNN@30'000 vocab, 10 Ep.	**70.22%**	69.13%	68.85%	68.61%
Simple CNN@30'000 vocab, 15 Ep.	**69.06%**	68.78%	68.40%	68.23%
Simple RNN@5'000 vocab, 5 Ep.	68.83%	0.00%	68.83%	68.83%

Tab. 4.5: Anwendungsfall 3, Online-Kommentare: Macro Average F1-Scores der eingesetzten Modelle

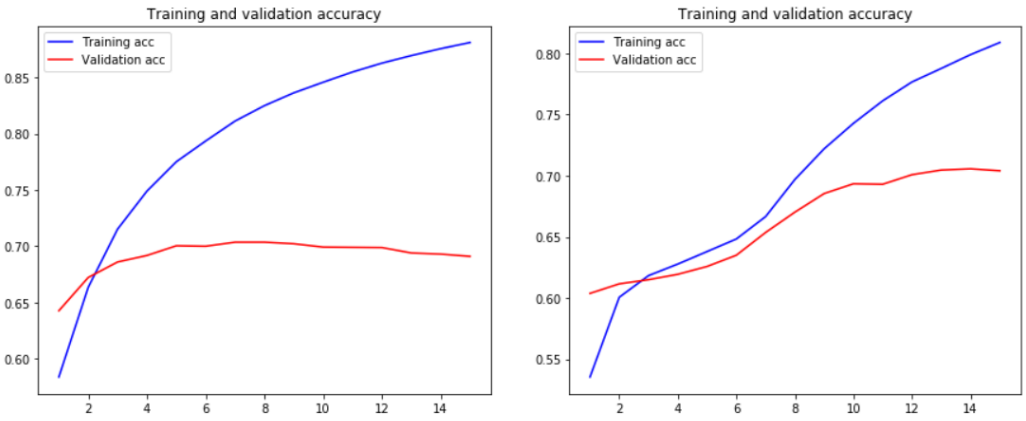

Abb. 4.1: Vergleich: Entwicklung Accuracy, Wide-Komponente vs. Deep-Komponente

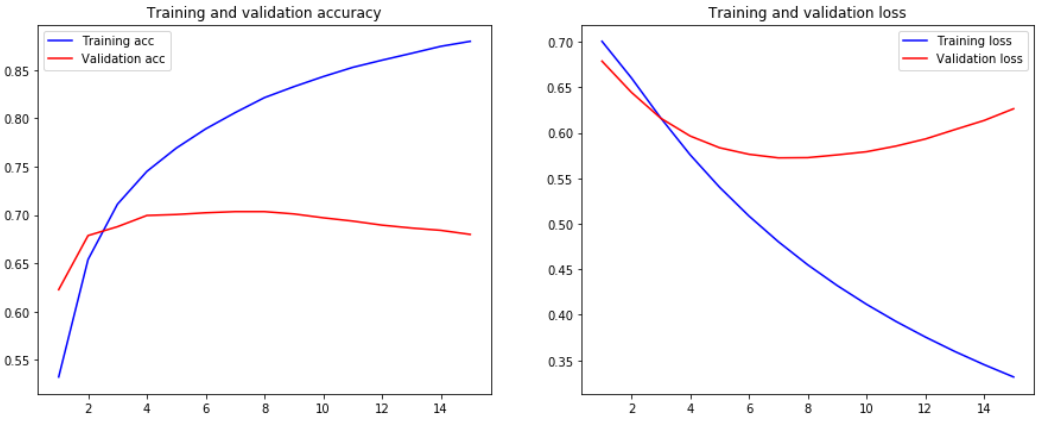

Abb. 4.2: Vergleich: Entwicklung Accuracy & Loss, Wide and Deep Modell

4.5 Fazit

Die Erkenntnisse, welche während der Evaluation der verschiedenen Datensets gemacht wurden, werden in einem ersten Abschnitt anhand von möglichen Problemen und Chancen zusammengefasst. Anschliessend findet eine Reflexion zum Vorgehen der Evaluation statt.

4.5.1 Mögliche Probleme und Chancen

Aus den Ergebnissen lassen sich sowohl mögliche Probleme, wie auch Chancen ableiten:

Probleme

- Ein Problem, welches bei der Wahl der Parametrisierung rasch auftrat: Im bereitgestellten Wide and Deep Modell kann die Anzahl Epochen nicht für die verschiedenen Komponenten separat festgelegt werden. Während beispielsweise ab etwa 15 Epochen die Deep-Komponente begann, gute bis sehr gute Ergebnisse zu liefern, sanken die Ergebnisse für die Wide-Komponente bereits

aufgrund von Over-Fitting. Bei einem Ensemble hingegen wäre es einfacher, die beiden Modelle über eine unterschiedliche Anzahl an Epochen zu trainieren.
Allenfalls müsste das Wide and Deep Modell entsprechend angepasst werden, dass die jeweiligen Layers ebenfalls separet vortrainiert werden könnten.

- In den evaluierten Setups zeigte sich jeweils ein sehr starker Einfluss der Wide-Komponente, während derjenige der Deep-Komponente weniger wahrnehmbar war. Mit vortrainierten Layers oder komplexeren Deep-Komponenten lässt sich dies allenfalls ausbalancieren.
- Der rasche Einfluss einer schlechteren Wide- oder Deep-Komponente auf das Resultat des Wide and Deep-Modelles lässt annehmen, dass diese Architektur stärker auf einzelne Variationen reagiert, als dies beispielsweise beim Ensemble der Fall ist. Dadurch ist bei der Spezifikation eines Wide and Deep-Modelles wohl mehr “Fingerspitzen-Gefühl“ notwendig.

Chancen

- Ein Wide and Deep Modell erreicht durch die Zusammenarbeit der beiden Komponenten teilweise sehr gute Ergebnisse. Für noch bessere Ergebnisse wäre es allenfalls hilfreich, für die beiden Komponenten noch weitere Features einzubringen, wie z.B. im Anwendungsfall 3 (Kommentare) zusätzliche Informationen zum betreffenden Artikel oder semantische Informationen zu Schlüsselwörtern des Kommentars.
- Im kleinen Rahmen des Anwendungsfalles 1 (IMDB) erreichte das Wide and Deep-Modell sehr rasch gute Resultate. Auch im Anwendungsfall 3 (Kmmentare) ist sichtbar, dass das Wide and Deep-Modell sehr rasch zu lernen scheint (Abbildung 4.2, Vergleich: Entwicklung Accuracy & Loss, Wide and Deep Modell). Allenfalls könnte davon ausgegangen werden, dass die Deep-Komponente des gemeinsamen Modelles durch das gemeinsame Training von der Wide-Komponente profitieren kann und sich der Lernprozess dadurch beschleunigt.

4.5.2 Rückblick Vorgehen

Zu Beginn der Evaluation waren für die Deep-Komponente diverse Modelle geplant. Aufgrund der zahlreichen verschiedenen Modelle und Parameter, welche es zu testen gab, reduzierte sich diese Zahl jedoch rasch auf zwei zentrale Architekturen: Simple-DNN und Simple-CNN.
So flossen schlussendlich 117 Modelle in die finale Evaluation ein. Inklusive Entwicklung wurden wohl weit über 1000 Modelle trainiert. Durch diese grosse Anzahl wurde die Evaluation sehr umfangreich und nahm viel Zeit in Anspruch, so dass keine grösseren Variationen bei den Netzarchitekturen evaluiert werden konnten.
Zudem wurden die Modelle jeweils nur anhand einer Messgrösse bewertet. Viele weitere Faktoren, wie z.B. Laufzeit, Konfusionsmatrix, ROC-Kurve, etc. flossen während der Entwicklung teilweise ein, hätten aber den Rahmen gesprengt, wären sie in die gesamte Evaluation miteinbezogen worden.

5 Ausblick und persönliches Fazit

In diesem Kapitel wird zuerst darauf eingegangen, wie sich Wide and Deep Learning in der Zukunft entwickeln könnte. Zum Schluss der Arbeit findet noch ein kurzer, persönlicher, aber durchaus auch kritischer Rückblick auf die Arbeit statt.

5.1 Wide and Deep Learning

Wide and Deep Learning bringt einige Vorteile mit sich, was bereits anhand der "Memorization and Generalization" Analogie anzunehmen war. Wahrscheinlich wird Wide and Deep Learning bereits viel häufiger angewendet als angenommen. Es stellt sich jedoch die Frage, wo Wide and Deep Learning beginnt, und wann es sich einfach um ein etwas komplexeres neuronales Netz handelt.
Wide and Deep Learning bringt den Vorteil, Daten verschiedener Natur und aus verschiedenen Kontexten in die Kategorisierung einfliessen zu lassen, wie dies z.B. mit den semantischen Informationen geschieht. Dies ermöglicht nicht nur neue Möglichkeiten mit bestehenden Datensammlungen. Es motiviert auch, sich Gedanken zu machen, welche Daten verarbeitet werden sollen: Welche Daten repräsentieren die Memorization, und welche die Generalization? Dadurch könnte man auch den Schluss ziehen, dass für die Anwendung von Wide and Deep Learning noch mehr Themenwissen und noch grössere Datenmengen benötigt werden, als dies mit anderen Architekturen der Fall ist.

5.2 Persönliches Schlusswort

Persönlich habe ich die zu Beginn gesteckten Ziele leider nicht ganz erreicht: Ich hätte mich gerne auch vertiefter mit einem Modell auseinandergesetzt und kompetitive Resultate erreicht. Im Kontext der Arbeit habe ich aber entschieden, mich auf eine oberflächlichere Evaluation zu konzentrieren. Dabei herrschte immer etwas eine Unsicherheit, resp. "Angst vor der Komplexität": Ich wählte meist den einfacheren Weg und testete bewährte Modelle mit verschiedene Parameter aus, anstatt dass ich mich mit neuen Modellen auseinandergesetzt hätte.
Dennoch kann ich rückwirkend von einer erfolgreichen Arbeit sprechen. Ich habe viel gelernt und kann einiges aus der Arbeit mitnehmen: Ich habe einerseits theoretische Einblicke in viele spannende, und vor allem auch aktuelle, Ansätze und Ideen erhalten. Andererseits konnte ich viele praktische Dinge anwenden, die wir in den letzten drei Jahren im Unterricht besprochen hatten.
In vielen Fällen hatte ich das Gefühl, dass ich ein oder zwei Schritte weitergegangen bin: Sei es bei der Entwicklung des Frameworks, was einiges an Python-Basiswissen erforderte. Oder sei es bei der Anwendung von vortrainierten Embeddings mit Hilfe

von GloVe. Oder sei es in der letzten Phase, als es darum ging, viele verschiedene Modelle und Ausprägungen auf verschiedenen Datensets zu evaluieren.

Literatur

[BSA17] Grégoire Burel, Hassan Saif und Harith Alani. „Semantic Wide and Deep Learning for Detecting Crisis-Information Categories on Social Media“. In: *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*. Hrsg. von Claudia d'Amato, Miriam Fernández, Valentina A. M. Tamma et al. Bd. 10587. Lecture Notes in Computer Science. Springer, 2017, S. 138–155 (zitiert auf den Seiten 17, 23).

[Che+16] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen et al. „Wide & Deep Learning for Recommender Systems“. In: *CoRR* abs/1606.07792 (2016). arXiv: 1606.07792 (zitiert auf den Seiten 14–16, 22).

[Con+16] Alexis Conneau, Holger Schwenk, Loıc Barrault und Yann LeCun. „Very Deep Convolutional Networks for Natural Language Processing“. In: *CoRR* abs/1606.01781 (2016). arXiv: 1606.01781 (zitiert auf den Seiten 9, 12, 22).

[CV95] Corinna Cortes und Vladimir Vapnik. „Support-Vector Networks“. In: *Machine Learning* 20.3 (1995), S. 273–297 (zitiert auf Seite 10).

[Dom12] Pedro M. Domingos. „A few useful things to know about machine learning“. In: *Commun. ACM* 55.10 (2012), S. 78–87 (zitiert auf den Seiten 3, 4).

[Guo+17] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li und Xiuqiang He. „DeepFM: A Factorization-Machine based Neural Network for CTR Prediction“. In: *CoRR* abs/1703.04247 (2017). arXiv: 1703.04247 (zitiert auf Seite 19).

[HS97] Sepp Hochreiter und Jürgen Schmidhuber. „Long Short-Term Memory“. In: *Neural Computation* 9.8 (1997), S. 1735–1780 (zitiert auf Seite 6).

[Joa98] Thorsten Joachims. „Text Categorization with Support Vector Machines: Learning with Many Relevant Features“. In: *Machine Learning: ECML-98, 10th European Conference on Machine Learning, Chemnitz, Germany, April 21-23, 1998, Proceedings*. 1998, S. 137–142 (zitiert auf Seite 10).

[KB14] Diederik P. Kingma und Jimmy Ba. „Adam: A Method for Stochastic Optimization“. In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980 (zitiert auf Seite 9).

[Lai+15] Siwei Lai, Liheng Xu, Kang Liu und Jun Zhao. „Recurrent Convolutional Neural Networks for Text Classification“. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.* 2015, S. 2267–2273 (zitiert auf Seite 13).

[Mik+13] Tomas Mikolov, Kai Chen, Greg Corrado und Jeffrey Dean. „Efficient Estimation of Word Representations in Vector Space“. In: *CoRR* abs/1301.3781 (2013). arXiv: 1301.3781 (zitiert auf Seite 5).

[PSM14] Jeffrey Pennington, Richard Socher und Christopher D. Manning. „GloVe: Global Vectors for Word Representation“. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, S. 1532–1543 (zitiert auf Seite 34).

[RM18] Sebastian Raschka und Vahid Mirjalili. *Machine Learning mit Python und Scikit-learn und Tensorflow. Das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics*. mitp, 2018 (zitiert auf den Seiten 4, 5, 7, 9).

[SZS18] Devendra Singh Sachan, Manzil Zaheer und Ruslan Salakhutdinov. „Investigating the Working of Text Classifiers“. In: *CoRR* abs/1801.06261 (2018). arXiv: 1801.06261 (zitiert auf Seite 10).

[Zhe+18] Zibin Zheng, Yatao Yang, Xiangdong Niu, Hong-Ning Dai und Yuren Zhou. „Wide and Deep Convolutional Neural Networks for Electricity-Theft Detection to Secure Smart Grids“. In: *IEEE Trans. Industrial Informatics* 14.4 (2018), S. 1606–1615 (zitiert auf Seite 20).

[ZZL15] Xiang Zhang, Junbo Jake Zhao und Yann LeCun. „Character-level Convolutional Networks for Text Classification“. In: *CoRR* abs/1509.01626 (2015). arXiv: 1509.01626 (zitiert auf Seite 11).

Webseiten

[Bro18] Jason Brownlee. *Machine Leearning Mastery*. 2018. URL: https://machinelearningmastery.com/ (besucht am 4. Jan. 2019) (zitiert auf Seite 25).

[Che16] Heng-Tze Cheng. *Wide & Deep Learning: Better Together with TensorFlow*. 2016. URL: https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html (besucht am 4. Nov. 2018) (zitiert auf Seite 14).

[Fag18] Daniel Faggella. *What is Machine Learning?* 2018. URL: https://emerj.com/ai-glossary-terms/what-is-machine-learning/ (besucht am 29. Dez. 2018) (zitiert auf Seite 4).

[Goo18a] Google. *imdb$_lstm.py$*. 2018. URL: https://github.com/keras-team/keras/blob/master/examples/imdb_lstm.py (zitiert auf Seite 29).

[Goo18b] Google. *Text classification with movie reviews*. 2018. URL: https://www.tensorflow.org/tutorials/keras/basic_text_classification (besucht am 14. Jan. 2019) (zitiert auf den Seiten 28, 35, 36).

[Goo18c] Google. *tf.estimator.DNNLinearCombinedClassifier*. 2018. URL: https://www.tensorflow.org/api_docs/python/tf/estimator/DNNLinearCombinedClassifier (besucht am 14. Nov. 2018) (zitiert auf Seite 16).

[Jan18] Nikolai Janakiev. *Practical Text Classification With Python and Keras*. 2018. URL: https://realpython.com/python-keras-text-classification/ (besucht am 4. Jan. 2019) (zitiert auf den Seiten 25, 29).

[Ker] Keras. *Datasets*. ? URL: https://keras.io/datasets/ (besucht am 4. Jan. 2019) (zitiert auf Seite 35).

[Rob18] Sara Robinson. *Predicting the price of wine with the Keras Functional API and TensorFlow*. 2018. URL: https://medium.com/tensorflow/predicting-the-price-of-wine-with-the-keras-functional-api-and-tensorflow-a95d1c2c1b03 (besucht am 23. Apr. 2018) (zitiert auf den Seiten 24, 25).

[Tag18] Tagesanzeiger. *Wie Sie bei uns online kommentieren*. 2018. URL: https://www.tagesanzeiger.ch/service/unsere-dienste/wie-sie-bei-uns-online-kommentieren/story/19367311 (besucht am 4. Jan. 2019) (zitiert auf Seite 39).

[Vee16] Fjodor Van Veen. *The Neural Network Zoo*. 2016. URL: https://www.asimovinstitute.org/neural-network-zoo/ (besucht am 9. Jan. 2019) (zitiert auf Seite 6).

[WSR18] Michael Wiegand, Melanie Siegel und Josef Ruppenhofer. *Overview of the GermEval 2018 Shared Task on the Identification of Offensive Language*. 2018. URL: https://github.com/uds-lsv/GermEval-2018-Data (besucht am 4. Jan. 2019) (zitiert auf den Seiten 35, 36, 38).

Abbildungsverzeichnis

Tabellenverzeichnis

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Selbständigkeitserklärung

Mit der Abgabe dieser Abschlussarbeit versichert der Studierende, dass er die Arbeit selbständig und ohne fremde Hilfe verfasst hat (Bei Teamarbeiten gelten die Leistungen der übrigen Teammitglieder nicht als fremde Hilfe).

Der unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Abschlussarbeit 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.

Winterthur, 31. Januar 2019

Manuel Streiff