# ANALYSIS OF CUSTOMER REVIEWS IN THE MOVIE DOMAIN: IDENTIFICATION OF ASPECT AND SENTIMENT FEATURES WITH NEURAL NETWORKS

MASTER THESIS

Submitted to  
Prof. Dr. Bernd Heinrich  
Institute of Management Information Systems  
Faculty of Business, Economics, Management Information Systems  
University of Regensburg

By  
Ahmad Alismail  
Franz-Mayer-Straße 13 App-0.23, 93053 Regensburg  
Student ID Number: 2080053  
Management Information Systems

Regensburg, 19.07.2021

Table of Contents

[List of Figures iii](#_Toc77547180)

[List of Tables iv](#_Toc77547181)

[List of Abbreviations v](#_Toc77547182)

[1 Introduction 1](#_Toc77547183)

[1.1 Motivation 1](#_Toc77547184)

[1.2 Research Questions 3](#_Toc77547185)

[1.3 Thesis Outline 3](#_Toc77547186)

[2 Theoretical Background 4](#_Toc77547187)

[2.1 Aspect, Sentiment, and Modifier Features Extraction 4](#_Toc77547188)

[2.2 Feedforward Neural Networks 5](#_Toc77547189)

[2.3 Recurrent Neural Networks 7](#_Toc77547190)

[2.3.1 Bidirectional Recurrent Neural Networks 8](#_Toc77547191)

[2.3.2 Long Short-Term Memory 8](#_Toc77547192)

[2.4 Conditional Random Fields 10](#_Toc77547193)

[2.5 Text Representation Methods for Deep Learning 11](#_Toc77547194)

[2.5.1 Non-Contextual Word Embeddings 12](#_Toc77547195)

[2.5.2 Contextual Word Embeddings 13](#_Toc77547196)

[3 Model Architecture 15](#_Toc77547197)

[3.1 Embedding Layer 16](#_Toc77547198)

[3.2 Bi-LSTM Layer 16](#_Toc77547199)

[3.3 CRF Layer 17](#_Toc77547200)

[4 Experimental Evaluation 19](#_Toc77547201)

[4.1 Dataset 19](#_Toc77547202)

[4.1.1 General Description 19](#_Toc77547203)

[4.1.2 Annotation Process 20](#_Toc77547204)

[4.1.3 Data Preparation 21](#_Toc77547205)

[4.1.4 Dataset Statistics 22](#_Toc77547206)

[4.2 Evaluation Metrics 24](#_Toc77547207)

[4.3 Experimental Setup 26](#_Toc77547208)

[4.4 Model Training and Hyperparameters 28](#_Toc77547209)

[4.5 Experimental Results 29](#_Toc77547210)

[4.5.1 The Impact of the CRF Decoding Layer: 29](#_Toc77547211)

[4.5.2 Shallow vs. Deep Bi-LSTM Layer: 30](#_Toc77547212)

[4.5.3 The Effect of Word Embeddings: 30](#_Toc77547213)

[4.5.4 Quantity vs. Quality: 31](#_Toc77547214)

[5 Error Analysis 32](#_Toc77547215)

[5.1 Aspect Term Extraction: 32](#_Toc77547216)

[5.2 Sentiment Term Extraction 36](#_Toc77547217)

[5.3 Modifier Term Extraction 39](#_Toc77547218)

[6 Conclusions and Future Work 41](#_Toc77547219)

[7 Reference List viii](#_Toc77547220)

[A Statement of independent work vi](#_Toc77547221)

###### List of Figures

[Figure ‎2.1: Shallow Feedforward Neural Network. 6](#_Toc77547222)

[Figure ‎2.2: Simple Recurrent Neural Network. 7](#_Toc77547223)

[Figure ‎2.3: Typical structure of the Bidirectional Recurrent Neural Network. 8](#_Toc77547224)

[Figure ‎2.4: Long-Short Term Memory network unfolded over three time steps, with a zoomed cell. 9](#_Toc77547225)

[Figure ‎3.1: Architecture of the Bi-LSTM-CRF model. 15](#_Toc77547226)

[Figure ‎3.2: Predictions of the Bi-LSTM layer and potential CRF layer corrections. 17](#_Toc77547227)

[Figure ‎4.1: Annotated sentence that includes all ASM-features. 19](#_Toc77547228)

[Figure ‎4.2: An example of the disagreement among annotators with the elected shared annotation. 21](#_Toc77547229)

[Figure ‎4.3: Sentence length distribution in the dataset. 22](#_Toc77547230)

[Figure ‎4.4: Frequency of class labels in both training approaches. 24](#_Toc77547231)

[Figure ‎4.5: Visualization for Confusion Matrix for a multi-class classification problem. 25](#_Toc77547232)

[Figure ‎5.1. Frequency of correctly predicted labels by the ATE model compared to the actual labels in the test data. 32](#_Toc77547233)

[Figure ‎5.2. Frequency of incorrectly predicted labels by the ATE model. 33](#_Toc77547234)

[Figure ‎5.3. Most frequently misclassified tokens by the ATE model. 33](#_Toc77547235)

[Figure ‎5.4. Frequency of correctly classified tokens by the ATE model. 35](#_Toc77547236)

[Figure ‎5.5. Frequency of correctly predicted labels by the STE model compared to the actual number within the test data. 36](#_Toc77547237)

[Figure ‎5.6. Most frequently misclassified tokens by the STE model. 37](#_Toc77547238)

[Figure ‎5.7: Frequency of the most correctly classified sentiment terms. 37](#_Toc77547239)

[Figure ‎5.8. Frequency of correctly predicted labels by the MTE model compared to the actual number within the test data. 39](#_Toc77547240)

[Figure ‎5.9. Frequency of the most correctly classified modifier terms. 39](#_Toc77547241)

###### List of Tables

[Table ‎2‑1: The behavior of word co-occurrence probabilities with various probe words*.* 12](#_Toc77547345)

[Table ‎4‑1: Number of training, validation, and testing sentences. 23](#_Toc77547346)

[Table ‎4‑2: Summary of evaluated models and baseline in the experiments. 27](#_Toc77547347)

[Table ‎4‑3: Hyperparameters used in the experiments. 28](#_Toc77547348)

[Table ‎4‑4: Final F1 scores (in %) for all evaluated model configurations and baseline. 29](#_Toc77547349)

###### List of Abbreviations

ABSA Aspect-based sentiment analysis

ASM Aspect, sentiment, and modifier features

ATE Aspect term extraction

BERT Bidirectional encoder representation from transformers

BP Backpropagation

BPTT Backpropagation through time

BRNN Bidirectional recurrent neural network

CRF Conditional random field

FNN Feed-forward neural network

GD Gradient Descent

GloVe Global vector for word representation

HMM Hidden Markov model

LSTM Long short-term memory

MEMM Maximum entropy Markov model

ML Machine learning

MLM Masked language model

MTE Modifier term extraction

NER Named entity recognition

NLP Natural language processing

NN Neural network

NSP Next sentence prediction

OOV Out-of-the-vocabulary word

PTM Pre-trained model

RNN Recurrent neural network

SA Sentiment analysis

SGD Stochastic gradient descent

STE Sentiment term extraction

## Introduction

### Motivation

Sentiments and opinions drive actions and propel behavior. Whether ordering food from a new restaurant or picking the next movie to watch, people attempt to validate their choice before committing to the buying decision. With the advancement of communication technology and the rapid growth of review platforms and social media channels, online reviews play a significant role in digital commerce. According to Agarwal et al. (2020, p. 112), 90% of customers consider online reviews a fundamental element that directly impacts their buying decisions. This is because the opinions and sentiments provided in such reviews help educate the potential buyer as to whether or not they should make the purchase. On the other side, these reviews are crucial to companies as they always want to analyze consumer feedback about their products and services. However, manually analyzing the immense amounts of online reviews is not feasible. Therefore, these businesses usually resort to Natural Language Processing (NLP) techniques as an automated alternative. NLP is concerned with the use of computational techniques for processing and understanding human languages (Chowdhury, 2003).

Among the essential research domains of NLP is Sentiment Analysis (SA), which is concerned with analyzing opinions and sentiments expressed in textual data such as customer reviews or social media posts. SA can be conducted on document, sentence, or aspect-based levels. Aspect-Based Sentiment Analysis (ABSA) aims to discover aspects (i.e., characteristics of a product or service) and the sentiment expressed towards them (Liu, 2020, p. 1,9-11). For instance, in the sentence *“The film was not good, but the soundtracks were great”*, the user expressed a positive sentiment toward a particular aspect which is *“soundtracks”* and a negative sentiment toward *“film”*. ABSA provides an automated approach for inspecting an immense number of online reviews and generates a brief summary thereof. The result of such detailed analysis is beneficial for companies to gain an insight into their customers’ tendencies and behavior in order to maintain adequate quality of service and improve their services/products in such a way that maximizes customer satisfaction (Agarwal et al., 2020, p. 111,139-140).

ABSA involves two key subtasks: aspect term extraction (ATE) and sentiment term extraction (STE) (Feng et al., 2019). ATE examines online reviews and extracts the aspect terms referenced in these reviews. On the other hand, STE identifies sentiment words expressed toward the aspect terms in these reviews (Agarwal et al., 2020, pp. 140–142). According to Feng et al. (2019), ATE and STE constitute the basis of ABSA and govern the accuracy and success of the results. In the above example, ATE aims to extract the aspect terms “*film*” and “*soundtracks*,” while STE targets the sentiment terms *“good”* and *“great.”* Additionally, modifier terms extraction (MTE) is important to the reliability of the process. MTE identifies words that can change the orientation or modify the intensity of sentiment terms. In the same example, MTE detects the modifier term “*not*.”

ATE and STE tasks were handled using a variety of techniques. Hu and Liu (2004) proposed a frequency-based model for extracting aspect terms that appear more frequently across a large number of reviews and then extracting sentiment terms related to them. Qiu et al. (2011) introduced a double propagation model, which exploits the syntactic relation between aspect and sentiment terms to extract them simultaneously. ATE can be treated as a sequence labeling problem, where each word in an input sequence is assigned a corresponding label (Liu, 2020, p. 182; Deng & Liu, 2018, p. 84). Hence, ATE was addressed using conventional supervised sequence labeling approaches such as Hidden Markov Models (HMM) and Conditional Random Fields (CRF) (Jin et al., 2009; Jakob & Gurevych, 2010). However, most of these techniques suffer from several limitations. They are heavily dependent on manually crafted features (e.g., semantic and syntactic relations). Hence, accomplishing high-quality performance requires a concentrated feature engineering effort (Minaee et al., 2021). Moreover, frequency-based and double propagation models show a low degree of transparency, making it difficult to replicate the experimental results reported by the authors (Marrese-Taylor & Matsuo, 2017). Last but not the least, most methods focused on ATE and STE and ignored MTE (Feng et al., 2019).

The limitations above can be overcome using Neural Networks (NNs), a branch of machine learning that typically learns in a supervised fashion to identify relations in a large set of input-output data pairs. Deploying multiple layers in the neural networks led to the name “deep learning” (Goldberg, 2017, p. 2). Deep learning models demonstrated a practical ability to deal with sequence labeling tasks without intensive feature engineering due to their powerful representation learning capability (He et al., 2020). This capability can also be exploited to learn good language representations, called “word embeddings”, from massive unlabeled text data. Word embeddings are text representation methods that capture the meaning of words using real-valued vectors in low dimensional space. Deep learning models can be initialized with word embeddings to avoid training from scratch and enhance the performance for downstream NLP tasks (Qiu et al., 2020). All things considered, deep learning models can provide a suitable approach to address ATE, STE, and MTE as sequence labeling problems. Throughout this work, the term “ASM-features” will refer to aspect, sentiment, and modifier features.

### Research Questions

These theoretical explanations illustrate that deep learning-based models can offer several advantages. In a practical application context, however, such potentials often have to be proven. This work analyzes how ASM-features can be extracted with a deep learning-based model from movie review data. In particular, it aims to answer the following research question:

* How effective is a deep learning-based model, namely Bi-directional Long Short-Term Memory Network with an additional CRF decoding layer, for ASM-features extraction tasks?

In addition, two specific questions will also be clarified. The first question is related to the extent to which word embeddings types affect ASM-features extraction tasks. The second question is more concerned with the structure of the dataset provided for training. Each review in the dataset was manually annotated on word level by three different human annotators. In other words, each ASM-feature has three distinct annotations that may conflict due to different annotators' decisions. Therefore, this work compares two approaches to learn from several annotators. In particular, this work will answer the following questions:

* What effect do different word embeddings have on the quality of ASM-features extraction tasks?
* How can textual data with (possibly conflicting) annotations of different annotators be used to improve the quality of ASM-features extraction tasks?

### Thesis Outline

This thesis proceeds as follows: Chapter 2 begins by formulating the ASM-features extraction problem and explaining key terminology and basic principles for different NN architectures and CRFs. Furthermore, it outlines text representation methods for deep learning models. Chapter 3 covers the model architecture of the deep NN used for ASM-features extraction, focusing on each of its layers. Chapter 4 illustrates the evaluation process and its outcomes. Chapter 5 analyzes the errors of the best model for ASM-features extraction. Finally, chapter 6 summarizes the main findings of this thesis and identifies areas for further research.

## Theoretical Background

This chapter illustrates the theoretical and conceptual foundations related to this research work. This chapter is divided into five sections. The first section depicts the ASM-feature extraction problem. The second section covers the fundamental theoretical concepts of simple NNs architecture. The third section outlines sophisticated NN architectures commonly employed in various NLP applications. The fourth section briefly reviews CRFs often applied in sequence labeling tasks. Finally, the fifth section describes common text representation methods for deep learning models.

### Aspect, Sentiment, and Modifier Features Extraction

In recent years, SA has received considerable scholarly attention in the NLP community. Early studies on SA focused on document-level and sentence-level classifications, which evaluate the sentiment of the whole document (e.g., movie review) or sentence as a positive, negative, or neutral. However, classifying opinionated text as atomic unit is insufficient for many applications. The reason for this is that people usually tend to share their opinions about some particular aspects of a product or service. ABSA analysis adopts a more comprehensive approach than the conventional SA levels and provides more useful knowledge about an author’s opinion on different aspects of products or services (Liu, 2020, p. 115). According to Feng et al. (2019), ABSA may be divided into different subtasks, among them:

1. Aspect term extraction (ATE)
2. Sentiment term extraction (STE)
3. Sentiment shifters extraction
4. Sentiment intensity extraction
5. Aspect sentiment classification

Tasks 3 and 4 aim to identify sentiment shifters, sentiment intensifiers, and sentiment diminishers. Sentiment shifters refers to a group of words that can reverse the orientation of sentiment words, e.g., *no*, *cannot*. Sentiment intensifiers and diminishers are words or phrases that can increase or decrease the intensity of sentiment words, e.g., *very* and *little* (Liu, 2020, pp. 59, 23–24). The term modifiers will be used in this work to refer to sentiment shifters, sentiment intensifiers, and sentiment diminishers. On the other hand, task 5 aims to analyze whether the sentiment expressed about a specific aspect is positive, negative, or neutral (Liu, 2020, p. 116). This task, however, is beyond the scope of this work. As previously stated, ATE task is a kind of sequence labeling problem. Likewise, STE and MTE tasks can also be treated as sequence labeling problems. More specifically, considering and are the input and output label sequences, respectively, where and , for every individual input , there is a unique output label (Deng & Liu, 2018, p. 84). In this work, ASM-features are extracted using three separate neural-based sequence labeling models with identical architecture. Each model will be trained on a set of sentences and corresponding labels. Following the training process, the model can examine a new, previously unobserved sentence and then assign a label to each word in that sentence. The labels indicate whether the word is part of a distinct ASM-feature or not. Words are labeled according to the BIO labeling scheme (Ramshaw & Marcus, 1995). For example, the input sentence *“I found this action movie very interesting.”* will be labeled as follows:

* *I<O> found<O> this<O>* ***action<B-A>******movie <I-A>*** *very<O> interesting<O>*
* *I<O> found<O> this<O> action<O> movie <O> very<O>* ***interesting<B-S>***
* *I<O> found<O> this<O> action<O> movie <O>* ***very<B-M>*** *interesting<O>*

Each input word is classified as either the beginning *“B-X”* or the inside *“I-X”* of a particular ASM-feature or otherwise *“O”* if not deemed a part of an ASM-feature. In this sentence, *“B-A”* denotes the beginning of an aspect term *“action movie”*, while *“I-A”* marks the inside part of the same aspect term.

### Feedforward Neural Networks

Among the different types of NNs, the Feedforward Neural Networks (FNNs) are a stereotypical example. These networks consist of a set of neurons arranged in layers. Figure ‎2.1 provides an example of a shallow three-layer FNN. The data proceeds through these networks starting from the input layer, and forward through the hidden layers, towards the output layer. A neuron is the primary computation unit in the NN. It collects the input values from the previous layer and produces a single output value. The weight represents the connection strength between neuron in layer and neuron in the subsequent layer, and its value is adjusted as the network learns. As shown in Figure ‎2.1, the third neuron in the hidden layer receives the sum of all of its input values ,, and multiplied by their respective weights , and . The result is then added to the bias value, denoted as , to produce the logit, denoted as . Next, in order to generate the output value, known as the activation , the neuron uses an activation function to apply a nonlinear transformation on the logit (Skansi, 2018, pp. 79–81).

FNNs are usually trained in a supervised fashion. The training process implies a gradual modification of the NN weights to minimize a predefined loss function. This process is carried out over several epochs for a given training set (Chollet, 2018, pp. 46–60). During the training, the gradient (i.e., the vector of partial derivatives) of the loss function with respect to the NN weights is computed via the backpropagation (BP) algorithm. The core idea behind BP is the chain rule from calculus. Then, the learning algorithm updates the weights of the NN using this gradient (Goodfellow et al., 2016, pp. 82, 200, 204). The most popular learning algorithms to optimize NNs are Gradient Descent (GD) based algorithms, such as mini-batch GD or Stochastic Gradient Descent (SGD). These algorithms change the weights in the opposite direction of the gradient. The magnitude of change is controlled by the learning rate (Ruder, 2017a).

|  |
| --- |
|  |
| Figure ‎2.1: Shallow Feedforward Neural Network. Figure source: (Adapted from Skansi, 2018, p. 80). |

Prior to NNs construction, several architecture-level parameters, known as hyperparameters, must be chosen. According to Chollet (2018, p. 263), the hyperparameters may include the number of hidden layers, learning rate, and the number of epochs. Hyperparameters are non-trainable. More specifically, they are not adjusted during the training process. Nevertheless, one approach to compute their values is splitting the data into training, validation, and testing sets. The NN is first trained using the training data with an initial set of hyperparameters. Throughout the training, the hyperparameters are updated according to the performance on validation data. This procedure is known as hyperparameter optimization. Eventually, the network performance will be evaluated based on the testing data (Goodfellow et al., 2016, pp. 118–119).

FNNs are primarily used for certain data types in which the order of elements is not important. Therefore, it is necessary to use more advanced types of NNs to process sequential data, such as text or time-series data (Aggarwal, 2018b, p. 271).

### Recurrent Neural Networks

|  |
| --- |
| Diagram  Description automatically generated |
| Figure ‎2.2: Simple Recurrent Neural Network. Figure source: (Adapted from LeCun et al., 2015). |

Recurrent Neural Networks (RNNs) are a specific variant of NNs used to deal with sequential data (Goodfellow et al., 2016, p. 367). They process the elements of an input sequence successively so that the output of each neuron is dependent on its current input value and the hidden state of the previous neurons. The hidden states can be viewed as a *“memory”* that enables RNNs to use prior knowledge regarding what has been learned so far during current processing. RNNs are suitable for modeling the sequential nature of human language in which the meaning of a word within a sentence is derived from the preceding words. For example, the different meaning of the word *“race”* in *“human race”* and *“arms race”*. Figure ‎2.2‎ illustrates simple RNN architecture, which is time-unfolded to accommodate the entire input sequence. In this figure, represents an input element to the network at time . In NLP applications, usually holds a numerical representation of textual input. denotes the memory at the same time step. , are the shared weights of the RNN over all time steps. represents the output at step (Young et al., 2018). In order to train RNNs, the gradients are calculated using a generalized type of the BP algorithm called backpropagation through time (BPTT) (Goodfellow et al., 2016, pp. 376–379). However, simple RNNs cannot capture long-term dependencies between distant elements in a sequence. This problem manifests as the time steps progress forward, the gradients obtained by BPTT become prone to either vanishing or exploding. In the literature, this complication is often referred to as the exploding and/or vanishing gradient problem (Goodfellow et al., 2016, p. 396, 398; Aggarwal, 2018b, p. 28). This issue was overcome with more advanced variants of RNNs, e.g., long short-term memory (LSTM) networks. Another architectural drawback of simple RNNs is their lack of consideration for future elements in the input sequence as the current output depends only on the previous elements. To resolve this, bidirectional recurrent neural networks were proposed (Agarwal et al., 2020, pp. 39–42).

#### Bidirectional Recurrent Neural Networks

|  |
| --- |
| Diagram  Description automatically generated |
| Figure ‎2.3: Typical structure of the Bidirectional Recurrent Neural Network. Figure source: (Adapted from Schuster & Paliwal, 1997). |

Bidirectional Recurrent Neural Networks (BRNNs) represent an extension to the basic unidirectional RNNs. BRNNs are designed to simultaneously process the elements of an input sequence in both time ways. As shown in Figure ‎2.3, this network involves two independent RNNs stacked side-by-side so that the output at each time step is calculated using the merged hidden states of both forward and backward networks (Schuster & Paliwal, 1997). BRNNs are frequently used in NLP. Therefore, they are called the “swiss army knife” of deep learning for NLP applications (Chollet, 2018, p. 219).

#### Long Short-Term Memory

Long Short-Term Memory networks (LSTMs) are an advanced type of RNNs proposed by Hochreiter and Schmidhuber (1997) to deal with a long sequence of inputs by integrating the concept of gates into their architecture. The structure of LSTMs is similar to a chain consisting of recurrently connected blocks called cells. LSTMs can cope with both the exploding and vanishing gradient problems occurred when performing BPTT (Agarwal et al., 2020, pp. 40–41).

|  |
| --- |
| Diagram  Description automatically generated |
| Figure ‎2.4: Long-Short Term Memory network unfolded over three time steps, with a zoomed cell. Figure source: (Adapted from Le et al., 2019) |

Figure ‎2.4 illustrates the standard LSTM network unrolled over time with basic cell architecture. LSTMs have two types of links that pass information from one cell to another. The first is the hidden state which represents the short-term memory at time step , whereas, the second is the cell state that serves as the long-term memory of the model (Skansi, 2018, p. 142). According to Olah (2015), LSTMs maintain the information in the cell state over a long sequence of inputs using a gating mechanism. An LSTM cell has three different gates that interact with each other to control the long-term memory, namely the forget gate, the input gate, and the output gate (see Figure ‎2.4). The forget gate selects which elements from the old cell state must be discarded or sustained. It generates a forget vector with values ranging between 0 and 1 via a sigmoid function, which takes the information from the current input and previous hidden state :

Here, is the sigmoid function, and are the weights and biases for the forget gate, respectively.

The next step is to select the relevant elements that should enter the new cell state . This decision is made by the input gate, which is composed of two parts. The first part creates an input vector with values ranging from 0 to 1 using a sigmoid function. The second part with a *tanh* function generates a vector of candidate values that can enter the cell state.

Here, , , and denote recurrent weights and biases for the input gate, respectively. In the following step, the old cell state gets multiplied by the forget vector . The result is added to to obtain the new cell state .

Finally, the output gate decides the elements that should continue as the short-term memory. It has two parts as well. First, the input and the previous hidden state are fed together into a sigmoid function to create an output vector . Second, the new cell state is fed into the *tanh* function and multiplied by the output vector to give the new hidden state .

Herein, and are the recurrent weights and biases for the output gate, respectively (Olah, 2015). This gating mechanism enables LSTMs to deal with long-term dependencies and thus achieve successful learning (Lu & Salem, 2017).

### Conditional Random Fields

CRFs is probabilistic sequence labeling model proposed by Lafferty et al. (2001). The underlying idea is that incorporating the context information of the immediate neighboring labels and when predicting the label for the current input token . The predictive process in CRF is modeled by maximizing the conditional probability for the label sequence given the corresponding sequence of input tokens (Aggarwal, 2018a, p. 398). CRFs offers several advantages compared to other statistical modeling methods for sequence labeling, such as the Hidden Markov Model (HMM) and the Maximum Entropy Markov Model (MEMM). CRF can relax the strong independence assumptions required by HMM (Lafferty et al., 2001). Furthermore, it removes the restriction of MEMM, according to which the label assignment for the current token depends only on its previous labels and not its following labels (Aggarwal, 2018a, pp. 397–398).

Before the advent of deep learning models, CRF was considered a standard approach to solving sequence labeling problems (Deng & Liu, 2018, p. 86). Over the past few years, CRF has been widely used as a final stage (i.e., a label decoding layer) in several deep learning models for sequence labeling (Huang et al., 2015; Lample et al., 2016; Augustyniak et al., 2019; Li et al., 2020). According to Lample et al. (2016), the CRF layer is used when the output labels are highly interdependent. Instead of modeling labeling decisions independently of one another, they are jointly modeled with a CRF layer, which aims to produce the optimal sequence of labels given a sequence of input tokens.

### Text Representation Methods for Deep Learning

Since machines cannot understand the human language, textual input must be encoded as real numbers. Words can be represented with low dimensional dense vectors (i.e., most elements are non-zero) called word embeddings. The distance and direction of vectors represent the semantic relationships among words. The closer the words are in meaning, the closer the distance between them. Synonyms are observed near each other, whereas antonyms are noticeably far from each other (Agarwal et al., 2020, pp. 63–64; Deng & Liu, 2018, pp. 259–260).

A massive amount of unlabeled text data is required for the learning process of word embeddings which is achieved using pre-trained models (PTMs) (Qiu et al., 2020). According to Qiu et al. (2020), two different generations of PTMs can be distinguished. In the first generation, the PTMs generate pre-trained non-contextual word embeddings, which can be used for other NLP tasks. The models themselves are then not required for downstream tasks. The second generation of PTMs aims to generate contextual word embedding by training deep neural architectures, also called neural contextual encoders. The pre-trained neural encoders are used later in the embedding layer of a deep learning model to find contextual embeddings for input words. The second-generation PTMs represent an effective realization of the transfer learning approach because they provide the starting point for deep learning models on NLP tasks (Young et al., 2018).

The following sections highlight prominent embedding methods currently employed with deep learning models.

#### Non-Contextual Word Embeddings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Probability and Ratio |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table ‎2‑1: The behavior of word co-occurrence probabilities with various probe word*.* (Pennington et al., 2014).

Pre-trained non-contextual word embeddings are the traditional methods for representing textual data. Each word is represented using a distinct multi-dimensional vector. For example, the word *“movie”* can be represented with the following 5-dimensional vector (1.3, 2.5, 2, -1, 0.3) (Skansi, 2018, p. 165).

Several techniques were introduced to learn word embeddings. For example, Mikolov et al. (2013) proposed Word2Vec, one of the most popular word embeddings generation methods based on the local usage context of the words. Furthermore, at Stanford, Pennington et al. (2014) developed Global Vector for Word Representation (GloVe) which is an unsupervised learning model capable of producing word representations. GloVe is trained on the aggregated statistics of a word-word co-occurrence matrix. This matrix is constructed after scanning the whole text corpus. It consists of a series of elements that represent the frequency with which a word is detected within the context of another word in a given text corpus. Based on this matrix, word-word co-occurrence probabilities are computed in order to calculate the ratio of these probabilities. As illustrated in Table ‎2‑1, the relationship of two target words, i.e., *“ice”* and *“stream”*, can be examined given other words , also called probe words. If the probe word is related to one of the target words, the ratio will be either large or small. On the other hand, if the probe word is either equally related to both target words or related to neither words, the ratio will be close to one. The relationships between words can be deduced using the global statistics of the corpus. Hence, it is useful to incorporate these statistics as the starting point for word embeddings learning (Pennington et al., 2014).

Pre-trained word embeddings are considered an efficient approach to improve various NLP applications. However, the same word is mapped to the same embedding vector regardless of its surrounding context (Qiu et al., 2020). For example, the word “*apple*” would have the same embedding vector in the following sentence: *“The apple juice was spilled on my apple MacBook!”*. Another limitation with these techniques is their inability to generate embedding vectors for unseen and rare words in the training corpus. In the literature, this problem is referred to as the Out-Of-Vocabulary (OOV) word problem (Zhao et al., 2018).

#### Contextual Word Embeddings

The second generation of PTMs produces contextual word embeddings. In other words, different embedding vectors are generated for the same word according to its surrounding context in a particular sentence (Qiu et al., 2020).

The revolution in the contextual embeddings methods started with the rise of Transformer neural architecture presented by Vaswani et al. (2017) in their research paper “Attention is All You Need”. The Transformer consists of two basic building blocks: encoder and decoder. Each block can be individually exploited as a part of a transformer-based model to create contextual word embedding (Alammar, 2018). One of the important advantages of the Transformer is its ability to model the relation between each pair of words in a sentence and thus capture the long-range dependencies of natural language (Qiu et al., 2020). Devlin et al. (2018) proposed a powerful PTM called BERT, an acronym for Bidirectional Encoder Representation from Transformers. As the name suggests, BERT utilizes the encoder part of the Transformer as a contextual neural encoder to generate contextualized embeddings. BERT is pre-trained on a vast text corpus (English Wikipedia and BookCorpus) through two unsupervised tasks to develop a strong understanding of language. These two tasks are masked language model (MLM) and next sentence prediction (NSP). In MLM, 15% of input words are randomly masked, then BERT is supposed to predict those words. In NSP, BERT learns to predict whether two sentences occurred consecutively in a given text or not (Devlin et al., 2018). BERT’s vocabulary consists of 30,000 tokens (words, subwords, and characters)[[1]](#footnote-1). BERT can represent OOV words by splitting them into subwords or even individual characters. For example, the word *“nonplussed”* is split into the following tokens: *“non”*, *“##p”*, *“##lus”*, and *“##sed”*. There are two approaches to use BERT for different NLP tasks: the fine-tuning approach and the feature-based approach. In the former, all weights of the pre-trained model and an additional task-specific layer are jointly trained to solve a particular classification task (Devlin et al., 2018). Whereas, in the latter, the pre-trained BERT model is used to generate contextual word embeddings, which are later used to seed an existing model (Alammar, 2018).

Another powerful PTM is Flair, proposed by Akbik et al. (2018). Flair utilizes a character-level neural language model for extracting word representations. According to the authors, Flair has a number of attractive features:

1. It is pre-trained on a huge unlabeled text.
2. It generates contextual word embeddings for input words.
3. Flair can find representations for misspellings and OOV words because it deals with input words and their surrounding context as sequences of characters.

In this approach, a bidirectional LSTM is used as language modeling architecture, which receives sentences as sequences of characters. At each position in the sequence, the model learns to predict the next character given all preceding characters. This enables the hidden states of the model to develop a strong language understanding automatically. The underlying idea is exploiting the hidden states of the PTM to derive contextual word embeddings for each word in the input sequence. Flair embeddings are extremely useful in sequence labeling tasks. Furthermore, these embeddings can be extended with different types of embeddings such as GloVe or BERT to form the final embedding vector, also referred to as “stacked embedding” (Akbik et al., 2018).

## Model Architecture

This chapter represents a technical description of the deep learning-based model for the ASM-features extraction from movie reviews, namely a bi-directional LSTM network with an additional CRF decoding layer (Bi-LSTM-CRF).

|  |
| --- |
|  |
| Figure ‎3.1: Architecture of the Bi-LSTM-CRF model. Figure source: (Adapted from Huang et al., 2015) |

In sequence labeling tasks, predicting a label for a given word is influenced by not only by its precedents but also its subsequent words in the text (Deng & Liu, 2018, p. 93). Taking the sentence *“Vin Diesel is a great actor”* as an example. The context information of the word *“Diesel”* is important to label it as a part of the actor's name. Therefore, the Bi-LSTM-CRF is a suitable choice because it exploits the advantages of both Bi-LSTM and CRF to incorporate the context information of words. This architecture is proposed by Huang et al. (2015) for sequence labeling. Akbik et al. (2018) achieved excellent results in sequence labeling tasks (e.g., NER) by extending the Bi-LSTM-CRF architecture with a powerful and flexible embeddings layer. Hence, the architecture employed in the experiments is based on the work by Akbik et al. (2018).

As shown in Figure ‎3.1, the architecture of the Bi-LSTM-CRF model comprises three primary layers: an embedding layer, a Bi-LSTM layer, and a CRF decoding layer. In the first step, the embedding layer generates embedding vectors for each word in the input sequence. Different PTMs can be integrated in the embedding layer to generate powerful word representations. The resultant embedding vectors are then passed through the Bi-LSTM layer, which encodes the contextual information for a specific input word. Finally, the CRF decoding layer makes use of directly neighboring labels when predicting the ultimate label (Huang et al., 2015). The layers of the Bi-LSTM-CRF architecture are described in the following subsections.

### Embedding Layer

The embeddings layer receives a sequence of N words (, ,..., ) and generates an embedding vector for each word (, ,..., ). As shown in Figure ‎3.2, the sentence *“I love action movies”* is split on the word level and passed through the embedding layer. The words are mapped into embeddings vectors using the pre-trained word embeddings and/or the second generation of PTMs discussed in ‎2.5. In other words, the final embedding vector can be any type of word embeddings or stacked embeddings. For example, Flair embeddings can be concatenated with BERT embeddings to form the final embedding vector for a word in position , and it is given by:

Here , represent the contextual Flair and BERT embedding vectors, respectively.

Using stacked embeddings is beneficial to enrich word representations with additional semantic and syntactic information. The performance of the Bi-LSTM-CRF architecture in sequence labeling tasks, e.g., named entity recognition (NER), is enhanced by using a combination of GloVe and Flair embeddings (Akbik et al., 2018). This leads to the assumption that the quality of ASM-features extraction tasks might be improved as well using a combination of multiple embedding types.

### Bi-LSTM Layer

The Bi-LSTM network introduced by Graves and Schmidhuber (2005) is used as a layer in this architecture to capture both forward and backward dependencies between sequence words. These networks can also be stacked on top of each other to build a deep Bi-LSTM layer (Graves et al., 2013). The (deep) Bi-LSTM layer receives the resultant embedding vectors from the embedding layer and returns the scores for every attainable label. The label with the highest score is selected as the final prediction of a particular token. As shown in Figure ‎3.2, the output of the Bi-LSTM layer for the word *“action”* is the following scores: 1.2 (*B-A*), 0.8 (*I-A*), 0.04 (*O*). The word *“action”* is labeled as “*B-A”* because it has the highest score. Similarly, all other words in the input sequence are labeled. The Bi-LSTM layer can produce correct predictions corresponding to the input sequence. In some cases, the predicted labels may be invalid. For example, both words *“action”* and *“movie”* are labeled as *“B-A.”* Therefore, the output of the (deep) Bi-LSTM layer is fed into the CRF layer that can correct some invalid predictions (Li, 2017).

|  |
| --- |
| Diagram  Description automatically generated |
| Figure ‎3.2: Predictions of the Bi-LSTM layer and potential CRF layer corrections. Figure source: (Adapted from Li, 2017). |

### CRF Layer

As the final layer in the sequence labeling model, the CRF layer receives a sequence of vectors from the (deep) Bi-LSTM layer and produces the final sequence of output labels. This layer has the ability to learn some restrictions related to BIO labeling scheme to ensure the validity of the predicted sequence labels. These restrictions are learned automatically during the learning process (Augustyniak et al., 2020). The following are examples in the case of ATE:

* The first predicted label in the label sequence can be *“B-A”* or *“O,”* but not *“I-A”*.
* Some patterns in the label sequence are not valid, such as *“O I-A”*, because the first predicted label for an aspect-term must be *“B-A.”* Therefore, this invalid prediction must be replaced with *“O B-A”*.

Therefore, using the CRF layer on top of the (deep) Bi-LSTM layer enables the model to produce more valid predictions. As shown in Figure ‎3.2, the word *“movie”* is now correctly classified as *“I-A”* rather than *“B-A”*. For this reason, the CRF layer is highly powerful for predicting the correct labels of multi-word aspect terms (Augustyniak et al., 2020, 2019). For instance, *“delivery time”, “sound quality”,* or names of the actors in the movie domain.

## Experimental Evaluation

Several experiments were performed systematically to evaluate how far the Bi-LSTM-CRF model is effective for the ASM-features extraction task, and what impact do different word embeddings types have on the quality of the extraction tasks. In addition, the experiments assess the model's performance based on two training approaches that combine the knowledge provided by multiple annotators.

This chapter is organized as follows: the first section describes the dataset and data preparation methods to learn from multiple annotators. The second section introduces the metrics used to assess the performance of the ASM-feature extraction models. The third section presents the Bi-LSTM-CRF model variants as well as one baseline model. The fourth section presents the hyperparameters used in the experiments. Finally, the fifth section compares the results of the different models on the test data.

### Dataset

#### General Description

|  |
| --- |
| A picture containing text, clock  Description automatically generated |
| Figure ‎4.1: Annotated sentence that includes all ASM-features. |

The dataset includes user-generated movie reviews extracted from the Amazon review dataset (Ni et al., 2019). The reviews are separated into individual sentences, each of which is broken on word level to form a list of words (or tokens). The dataset consists of 3,324 sentences written in English. Each sentence is associated with three sets of annotations provided by three different human annotators. Given one set of annotations, the individual sentence has three primary types of labels, which represent the ASM-features. As shown in Figure ‎4.1, the sentence is comprised of seven tokens with corresponding aspect, sentiment, and modifier labels. This sentence contains a multi-word aspect term that extends across two tokens, i.e., *“action movie”*. In addition, there are two unique sentiment terms (i.e., *“good”* and *“great”*) and one modifier term *“very”*. In the dataset, aspect terms fall under one of the following pre-defined aspect categories:

* Movie: refers to the movie in general, e.g., cinematography, staging, and soundtrack.
* Actor: refers to the actors in the movie.
* Director/Producer: refers to the director or producer of the movie.
* Shipping: refers to the delivery service upon purchasing the movie.
* Story: refers to the movie’s plot, events, and scenario.
* Price: refers to the movie’s price.

Sentiment terms, as mentioned before, are words and phrases expressed by users to describe their personal perspective towards one of the above aspect categories. Modifier terms, in turn, refer to a group of words that modify the intensity or change the orientation of the sentiment term.

It is worth mentioning that each set of annotations includes two other minor types of labels: uncertainty and difficulty labels. These labels indicate that the annotator during the annotation process had difficulty or was uncertain when assigning an appropriate ASM-feature label to a token/group of tokens. This issue can occur when the sentence contains complex structures or slang expressions. However, uncertainty and difficulty labels were not taken into consideration during the training process. This decision was made based on the adopted approaches for data preparation to deal with (possibly conflicting) annotations explained in Section ‎‎4.1.3.

#### Annotation Process

The reviews in the dataset were essentially raw, i.e., lacking any ASM-feature labels. Therefore, it was necessary to annotate these reviews in order to train and test a supervised deep learning model. The entire annotation process was conducted manually by 27 human annotators from the University of Regensburg using an online annotation platform called INCEpTION, which allows several persons to work on the same annotation project simultaneously (Klie et al., 2018). Participating annotators received specific annotation guidelines, according to which they had to check each word in every review and provide an appropriate annotation. Since the dataset includes multi-word terms representing the same entity, e.g., *“action movie”*, the data was annotated using the BIO labeling scheme proposed by Ramshaw and Marcus (1995). The BIO format denotes the beginning (B), inside (I), and outside (O) of a particular ASM-feature.

#### Data Preparation

|  |
| --- |
|  |
| Figure ‎4.2: An example of the disagreement among annotators with the elected shared annotation. |

The annotation accuracy relies on the annotator’s subjective perception and personal experience. Since three different individuals annotated each review, they might assign different class labels for the same token. As shown in Figure ‎4.2, Annotator 1 and Annotator 2 agree on two aspect terms: *“drama”* and *“music”*. However, Annotator 2 mislabeled the aspect term *“movie”* as *“O”*. In addition, Annotator 3 mislabeled all aspect terms except the last one, i.e., *“movie”*. Unfortunately, the dataset does not involve any information about the annotators’ expertise. Such information can be used to give more weight to the annotations provided by reliable annotators (Raykar et al., 2010). Disagreement among annotators can be rectified through an extensive reassessment supervised by domain experts which is both time- and effort-consuming. As an alternative, the training data can be prepared in several forms to combine the information provided by all annotators and train the sequence labeling models. In this work, all models were trained according to the following approaches:

* **Training on shared annotations***:* different annotations for the same sentence are aggregated to produce a single annotation that only contains labels that were agreed upon, or shared, among the majority of the annotators. Then, each sentence with its corresponding shared annotation is passed into the sequence labeling model during the training process. As shown in Figure ‎4.2, shared annotation involves labels that received the most votes from three annotators. This approach also tries to avoid uncertain situations by the labels, which reflect a high degree of agreement among all annotators.
* **Training on all annotations***:* each sentence with corresponding annotations provided by the three annotators is passed into the sequence labeling model during the training. This approach aims to retain as many ASM-feature labels as possible and not decrease the available training data per ASM-feature.

#### Dataset Statistics

|  |
| --- |
| Chart, histogram  Description automatically generated |
| Figure ‎4.3: Sentence length distribution in the dataset. |

This section provides a statistical overview of the dataset. Analyzing the dataset provides important facts for the error analysis after training ASM-features extraction models. The dataset contains 3224 reviews. The longest sentence comprises of 165 tokens, whereas the shortest sentencecontains 1 token only. The average token count across the sentences in the dataset is 17.04. Figure ‎4.3 shows the sentence length distribution. The distribution shows that 97.42% of sentences have 50 tokens or less. Only 83 sentences include more than 50 tokens. Further analysis shows that the most frequent tokens within the dataset are stopwords and punctuations. Stopwords refer to a group of words that occur frequently in natural languages and add little or no meaning to the overall sentence. (Hapke et al., 2019, p. 51). They make up almost 40% of the tokens in the dataset. Examples of the most common stopwords in the dataset are *“the”*, *“to”*, and *“of”*. All tokens were converted to lowercase so that the model treats them equally. No additional text preprocessing techniques were performed, such as the removal of punctuations/stopwords, stemming, spelling corrections, etc. The dataset was split into three subsets as follows:

* **Training data**: Comprising 75% of the **original** annotated data. This set of annotated sentences was used to train the model.
* **Validation data**: It amounted to 8% of the **training** data. This subset is dedicated to tuning the learning rate. The limited small size of the overall dataset restricted the size of the validation set because it was necessary to preserve a major part of the dataset to train the model.
* **Testing data**: this set consists of 25% of the **original** annotated data and serves as the unseen examples that were later used to evaluate the performance of the trained model.

|  |  |  |
| --- | --- | --- |
|  | Training with shared annotations | Training with all annotations |
| # of training sentences |  |  |
| # of validation sentences |  |  |
| # of testing sentences |  |  |
| Table ‎4‑1: Number of training, validation, and testing sentences. | | |

Table ‎4.1 shows the number of sentences used for training and testing the sequence labeling models for the ASM-features extraction. In the training phase, the actual number of both training and validation sentences is dependent on the training approach, i.e., whether the model is trained on the shared or on all annotations. On the other hand, in the testing phase, the model performance is evaluated against all annotations, i.e., the same predicted annotation for a test sentence is compared with annotations provided by three different annotators.

For each ASM-feature, most of the tokens in both training approaches are annotated with *“O”*. Taking the training set for ATE with the shared annotations as an example, 94.8% of tokens are annotated with *“O”*. That is, the dataset is imbalanced. This occurs when the number of tokens corresponding with a specific class label, i.e., *“O”*, is considerably larger than other class labels, which occur infrequently. In the literature, this problem is referred to as an imbalanced or skewed class distribution problem (Sun et al., 2009). The effect of this problem on the model predictions will be discussed in more detail in Chapter ‎5.

Figure ‎4.4 displays the frequency of class labels in both training approaches after excluding *“O”* labels. An obvious distinction can be made for inter-class and intra-class observations. The first notable observation is the substantial difference in number between intra-class labels when considering both training approaches. The frequency of labels declines significantly only when the shared annotations are used to train the model. Moreover, the “*B\_S”* label is obviously the most frequent label. Ranking last in terms of frequency are the modifier terms, with *“IM”* significantly less than *“BM”*.

There are also other notable trends regarding token frequency within a specific class label in the entire dataset. For instance, the top three tokens assigned with the *“B\_A”* class label are nouns like *“movie”*, *“story”*, and *“film”*. In the case of the *“B\_S”* class label, adjectives like *“great”* and *“good”* are more frequent than verbs such as *“love”* and *“like”*. Furthermore, the most frequent tokens among the class labels *“I\_A”*, *“I\_S”*, and *“IM”* are stopwords and the exclamation mark.

|  |
| --- |
|  |
| Figure ‎4.4: Frequency of class labels in both training approaches. |

### Evaluation Metrics

The performance of the text classification models can be evaluated using a set of metrics. The most widely used performance metrics are Precision, Recall, and F1-Score. They provide a comprehensive overview of the classification quality when the test data is imbalanced (Minaee et al., 2021). These metrics are calculated based on the prediction results provided by the confusion matrix. This matrix quantifies the number of (in)correct predictions made by the model for each class (Sokolova & Lapalme, 2009). Figure ‎4.5 shows a confusion matrix for a multi-class classification problem with classes. According to Krüger (2018, pp. 71–72), there are four types of prediction results in the confusion matrix when considering an individual class label :

* True Positive (TP): denotes the number of observations that belong to class and were correctly classified as .
* True Negative (TN): denotes the number of observations that do not belong to class and were not classified as .
* False Positive (FP): denotes the number of observations that do not belong to class but were incorrectly classified as .
* False Negative (FN): denotes the number of observations that belong to class but were incorrectly classified as another class.

|  |
| --- |
| Chart  Description automatically generated |
| Figure ‎4.5: Visualization for Confusion Matrix for a multi-class classification problem. Figure source: (Adapted from Krüger, 2018, p. 71). |

An observation represents a token in the case of the ASM-features extraction task. With respect to the precision per class, it expresses how certain the model is when making predictions for this class. Mathematically, it is the number of correctly predicted tokens from class divided by the total number of tokens predicted by the model as being :

On the other hand, the recall per class represents how well the model is able to detect this class correctly. Mathematically, it is the number of correctly predicted tokens from class divided by the total number of tokens from the same class in the test data (Krüger, 2018):

The precision or recall for a multi-class classification model is the average of the same metric of all classes (Sokolova & Lapalme, 2009). For example, precision and recall of the ATE model are calculated as follows:

Where and are precision and recall for the classes *“B\_A”*, *“I\_A”*, and *“O”*, respectively.

Finally, F1-score is a metric that considers the precision and recall taking their harmonic mean. The best value of this score is 1 (best precision and recall), and the worst value is 0 (Minaee et al., 2021, p. 25). In the case of ATE, the F1-score is computed as follows:

Since ASM-features are extracted separately, the final F1-score for all extraction tasks is calculated as follows:

Where , , denote the F1-score for ATE, STE, and MTE respectively. In this work, is the primary metric used to evaluate the model’s performance. Precision and recall are also important because they provide useful insights to understand the model’s behavior.

### Experimental Setup

The performance of the Bi-LSTM-CRF model architecture was evaluated with different types of word embeddings in the embedding layer, namely GloVe (Pennington et al., 2014), Flair (Akbik et al., 2018), and BERT (Devlin et al., 2018). In order to validate the performance of this architecture, it was compared with a vanilla Bi-LSTM network. Table ‎4‑2 presents a list of all models tested during the evaluation. Each model was trained on both the shared and all annotations. All models were implemented using Flair framework[[2]](#footnote-2). Flair is suitable for the evaluation because it allows the application of powerful deep learning models to sequence labeling tasks with flexible configuration options. Furthermore, it facilitates concatenating several types of word embeddings using a simple interface (Akbik et al., 2019).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model Abbreviation** | **Model Description** | **GloVe** | **BERT** | **Flair** | **CRF** |
| **G-Bi-LSTM** | A Bi-LSTM network without CRF decoding layer using pre-trained non-contextual GloVe embeddings. It is thus the baseline. | yes | no | no | no |
| **G-Bi-LSTM-CRF** | A standard Bi-LSTM-CRF model that relies solely on non-contextual GloVe embeddings. | yes | no | no | yes |
| **GF-Bi-LSTM-CRF** | An extension of the Bi-LSTM-CRF model in which non-contextual GloVe embeddings are combined with contextual Flair embeddings to form the final embedding vector for each token in the input sequence. This combination is recommended by Akbik et al. (2018) for sequence labeling tasks. | yes | no | yes | yes |
| **BF-Bi-LSTM-CRF** | An extension of the Bi-LSTM-CRF model in which contextual BERT and Flair embeddings are concatenated to form the final embedding vector. | no | yes | yes | yes |
| **GBF-Bi-LSTM-CRF** | A similar extension in which GloVe, BERT, and Flair are concatenated to form the final embedding vector. | yes | yes | yes | yes |
| Table ‎4‑2: Summary of evaluated models and baseline in the experiments. The prefixed letters refer to the word embedding type. | | | | | |

### Model Training and Hyperparameters

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Optimizer | SGD |
| Mini-Batch size | 32 |
| Max epochs | 150 |
| Learning rate | 0.1 |
| Anneal factor | 0.5 |
| Patience | 3 |
| Recurrent units | 256 |
| Bi-LSTM layers | 1, 2 |
| Fine Tune | no |

Table ‎4‑3: Hyperparameters used in the experiments.

Table ‎4‑3 presents a list of the hyperparameters used in the experiments. All models were trained using the SGD algorithm. The batch size was set to 32, i.e., model weights are updated for every mini-batch of 32 sentences (Ruder, 2017a). The maximum number of epochs to train each model was set to 150. The learning rate is not a fixed value but rather changes dynamically during the training according to a learning rate annealing technique. The initial learning rate was set to 0.1, with an annealing factor equaling 0.5, and a patience value of 3. The annealing technique implemented works as follows: the learning rate starts with 0.1, then it is halved if training loss on the validation data does not improve for 3 successive epochs (Akbik et al., 2018). The training process halts in two events, whichever occurs first; either the learning rate becomes too small, after which the SGD algorithm will barely update the weights, which in turn causes the training to progress extremely slowly, or the maximum number of training epochs is surpassed. The number of recurrent units per LSTM layer was set to 256. In addition, all models were trained first with one and then with two Bi-LSTM layers to assess whether they increase or decrease the quality of the overall classification. The pre-trained BERT and Flair models were utilized as feature extractors in order to generate contextual embeddings, i.e., their weights were frozen and not fine-tuned during training. The generic pre-trained BERT model (BERT-base) was utilized rather than the large version (BERT-large) due to limited computational resources.

### Experimental Results

The final F1 scores were calculated as per Equation (4.6) and presented in Table ‎4.4. The results show that the Bi-LSTM-CRF model performs remarkably well on ASM-features extraction tasks with small amount of training data. In addition, using a combination of contextual word embeddings are helpful to improve the quality of the ASM-features extraction tasks. Moreover, the quality of the classification was improved with training on all annotations, i.e., it is a useful strategy to combine the knowledge provided by several annotators in this dataset. The best performance was obtained with the two-layer BF-Bi-LSTM-CRF configuration trained on all annotations. The following subsections describe some observations in more detail.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | | **Training on shared annotations** | **Training on all annotations** |
| G-Bi-LSTM *(baseline)* | 1 Layer | 48.71 | 49.93 |
| 2 Layers | 48.28 | 53.39 |
| G-Bi-LSTM-CRF | 1 Layer | 57.89 | 58.39 |
| 2 Layers | 57.33 | 60.82 |
| GF-Bi-LSTM-CRF | 1 Layer | 62.53 | 62.12 |
| 2 Layers | 62.45 | 63.43 |
| BF-Bi-LSTM-CRF | 1 Layer | 62.89 | 63.20 |
| 2 Layers | **64.27** | **64.45** |
| GBF-Bi-LSTM-CRF | 1 Layer | 63.64 | 63.82 |
| 2 Layers | 63.83 | 63.30 |
| Table ‎4‑4: Final F1 scores (in %) for all evaluated model configurations and baseline. | | | |

#### The Impact of the CRF Decoding Layer:

The first two experiments aimed to isolate the impact of the CRF decoding layer on the model performance and thus on the output quality. The results confirm the positive impact of using a CRF layer in generating the optimal sequence of labels for the input sequence. As shown in Table ‎4‑4, the G-Bi-LSTM-CRF model is always superior to the G-Bi-LSTM. The improvement by utilizing the CRF layer lies between 7.43 and 9.18 percentage points for all evaluated configurations. A more detailed analysis of predictions and the impact of the CRF layer shows that the G-Bi-LSTM model could not predict most of the *“I\_X”* labels. For example, the two-layer G-Bi-LSTM trained on all annotations predicted only 0.72% of *“I\_A”* labels correctly; on the other hand, the same model with an additional CRF layer predicted 7% of *“I\_A”* labels correctly.

#### Shallow vs. Deep Bi-LSTM Layer:

In the experiments, the effect of increasing the depth of the Bi-LSTM layers on the model’s performance was analyzed. All models were trained with shallow/one Bi-LSTM layer and with deep/two Bi-LSTM layers. The number of hidden states per Bi-LSTM layer was kept fixed (256 hidden states for each direction). As can be seen from Table ‎4‑4, the results vary according to the following factors: (1) whether the model was trained on shared or all annotations and (2) the selected type of word embeddings:

* **Training on all annotations:** Increasing the number of the Bi-LSTM layers improved the model performance in most cases. It is also apparent that the improvement in F1-scores was gradually decreasing by incorporating more semantic and syntactic information into the final word embedding vector. For example, increasing the depth of the Bi-LSTM layer gave a 2.48 percentage points higher F1-score by using non-contextual GloVe embeddings; on the other hand, it gave a 1.31 percentage points higher F1-score by using a combination of GloVe and contextual Flair embeddings.
* **Training on the shared annotations:** Increasing the number of layers has no apparent effect on the performance. Although it improved the performance of the BF-Bi-LSTM-CRF model, it led to similar results in most cases.

#### The Effect of Word Embeddings:

As shown in Table ‎4‑4, the performance of the Bi-LSTM-CRF model with a combination of non-contextual GloVe embeddings and contextual embeddings derived from Flair outperformed the same model with only GloVe embeddings. This improvement can be attributed to the ability of Flair embeddings to cope with the OOV words problem. Moreover, they can capture different meanings of the word depending on its contextual use (Akbik et al., 2018). On the other hand, the second combination of contextual Flair and BERT embeddings produced richer word representations and thus resulted in the best performance. The two-layer BF-Bi-LSTM-CRF configuration gave a 1.02 percentage points higher F1 score than the two-layer GF-Bi-LSTM-CRF by training on all annotations. Contrary to expectations, the two-layer GBF-Bi-LSTM-CRF configuration gave a 1.15 percentage points lower F1-score than the two-layer BF-Bi-LSTM-CRF by training on all annotations. That is, combining different word embedding types does not always guarantee better results.

#### Quantity vs. Quality:

As mentioned in Section ‎‎4.1.3, two training approaches were used to learn from several annotators and thus to deal with (possible conflicting) annotations. As shown in Table ‎4‑4, training on all annotations improved the performance in most cases. The observed increase in F1-score lies between 0.18 and 3.49 percentage points for all evaluated model configurations. The impact of both training approaches on the performance can be illustrated by calculating both average precision and average recall over all ASM-features extraction tasks.

Considering the two-layer BF-Bi-LSTM-CRF configuration, the average precision was 71.64% by training on all annotations, while it increased to 72.75% by training on the shared annotations. That is, this configuration generates more trustworthy predictions by training on shared annotations. On the other hand, the average recall was 58.58% by training on all annotations, while that value decreased to 57.58% by training on shared annotations. More specifically, this configuration is able to detect more ASM-features labels correctly by training on all annotations.

In practical applications, choosing an appropriate model may depend on several factors, including the requirements for recall (quantity) and precision (quality). With respect to the ASM-features extraction tasks, it could be argued that quantity might be preferable over quality. A possible reason for this is that the ATE and STE tasks represent the first phase in the ABSA (Ganganwar & Rajalakshmi, 2019). Thus, failure to discover these features harms the entire ABSA tool significantly. For this reason, the two-layer BF-Bi-LSTM-CRF configuration trained on all annotations would be preferred over its counterpart trained on the shared annotations. The next chapter aims to analyze the predictions and errors of the best configuration on test data.

## Error Analysis

The two-layer BF-Bi-LSTM-CRF model configuration trained on all annotations is the best available choice for ASM-features extraction. This chapter analyzes prediction errors of this configuration on the test data for each ASM-features extraction task. That is, error analysis is performed on the predictions of ATE, STE, and MTE tasks. Such analysis is beneficial to detect the weaknesses for each task and find the most common reasons for performance drop. Therefore, it represents the starting point to improve the quality of extraction tasks.

### Aspect Term Extraction:

|  |
| --- |
|  |
| Figure ‎5.1. Frequency of correctly predicted labels by the ATE model compared to the actual labels in the test data. |

The precision and recall values of the ATE model are 70.85% and 58.17%, respectively. Consequently, the F1-score equals 63.88%. The number of correctly predicted labels is 38528 labels, whereas 2320 labels were incorrectly predicted. Figure ‎5.1 shows the number of correctly predicted labels compared to the total number of labels within the test data for each class label. What stands out in this figure is the difference between the numbers of correctly predicted class labels. The model successfully predicted 37317 *“O”* labels, which corresponds to 98.32% of the actual *“O”* labels within the test set. On the other hand, the number of correctly predicted *“B\_A”* labels equals 964, which constitutes 53.79% of the actual *“B\_A”* labels. In addition, the model was able to predict 22.39% only of the actual *“I\_A”* labels, and this translates to 247 labels. The performance drop when predicting *“B\_A”* and *“I\_A”* class labels can be attributed to the imbalanced data problem. The modest number of these class labels within the training data impairs the model’s ability to discover their classification rules. In other words, test tokens that belong to the infrequent classes are more likely to be misclassified. For this reason, this issue is also known as the rare class classification problem (Sun et al., 2009). This is further demonstrated in Figure ‎5.2, which illustrates the frequency of actual labels that were incorrectly predicted by the ATE model. From this figure, it can be seen that both class labels *“B\_A”* and *“I\_A”* were frequently misclassified into the majority class label, i.e., *“O”*.

|  |
| --- |
| Chart  Description automatically generated |
| Figure ‎5.2. Frequency of incorrectly predicted labels by the ATE model. |
| Chart, bar chart  Description automatically generated |
| Figure ‎5.3. Most frequently misclassified tokens by the ATE model. |

The second step of the error analysis is to identify the most common misclassified tokens. As can be seen from Figure ‎5.3, the model frequently failed to classify both *“of”* and *“the”* tokens correctly as a part of multi-word aspect terms. Both tokens were frequently misclassified as *“O”* although they are annotated with *“I\_A”*. This is associated with their number of occurrences within the training data. For instance, *“the”* appears 5163 times as *“O”* in the training data, whereas this number declines to 112 in the case of *“I\_A”*. Moreover, the word *“movie”* was misclassified as *“B\_A”* 29 times, although it is actually annotated with *“O”.* This could be an indicator of inconsistent annotations in the test data. For instance, the word *“movie”* in the following sentence was classified as *“B\_A”* by the ATE model:

* *“This* ***movie*** *is so well crafted”*

In the test data, however, *“movie”* in this sentence was incorrectly annotated with *“I\_A”* by two annotators. As previously stated, ASM-features were separately extracted using three distinct models with identical architecture and configurations. However, extracting aspect and sentiment terms separately has a disadvantage. The ATE model may classify some tokens as aspects despite the lack of related sentiment terms. For example, the word *“movie”* in the following sentence was misclassified as *“B\_A”* although the sentence does not include any sentiment term:

* *“This* ***movie*** *was focused around a sword that is stolen, retrieved and stolen again”*

Although the model could predict many proper nouns correctly (e.g., actor names), it sometimes failed to classify multi-word aspect terms that contain proper nouns with an apostrophe end. For instance, the following tokens were misclassified as *“O”*, although they are *“B\_A”* and *“I\_A”*:

* *“Leslie’s videos”*
* *“Maggie’s anger”*

Furthermore, the model sometimes failed to classify tokens that represent movie titles. For instance, the following tokens were misclassified as *“O”*:

* *“The Lost Skeleton of Cadavra”*
* *“Rum Diary”*
* *“Seven Psychopaths”*

The reason for the last two issues can be explained by the insufficient examples in the training data. Prediction analysis also shows that the most frequently extracted aspect terms are explicit aspects (see Figure ‎5.4). In other words, the aspect/feature is expressed directly in the sentence (Liu, 2020, p. 29). For example, the ATE model identified all explicit aspects (in bold) in the following sentence:

* *“Cheap, bad* ***acting****, bad* ***script****, bad* ***costumes****, bad* ***image****, bad* ***music****”*

However, aspects are not always mentioned explicitly in the sentence. The author of the review may use adjectives, adverbs, verb phrases, or more complex expressions to indicate aspects implicitly. In the previous example, *“cheap”* indicates the movie in general. Such aspect expressions are referred to as implicit aspects (Liu, 2020, p. 188). Prediction analysis shows that the model was not able to detect implicit aspects in some cases. For example, the following tokens (in bold), which indicate the movie in general, story, and shipping, were misclassified as *“O”*:

* *“****Engaging*** *to the end”*
* *“Really* ***worth seeing****”*
* *“****On time*** *and* ***as promised****”*

|  |
| --- |
| *Chart, histogram  Description automatically generated* |
| Figure ‎5.4. Frequency of correctly classified tokens by the ATE model. |

In some other cases, it is not easy to interpret the model’s behavior, i.e., why the model made a particular classification decision for a particular token. For example, the model misclassified the following tokens (in bold) as *“B\_A”* and *“I\_A”*, although they are *“O”*:

* *“One of the strangest movies I've seen in a very long time, it is about the* ***final days of World War II and the life of the Japanese emperor Hirohito****”*

In addition, some minor changes among sentences may radically alter the token’s classification. The following sentences provide an example:

* *“When I first watched this* ***movie*** *<O> I enjoyed the beginning”*
* *“When I second watched this* ***movie*** *<B\_A> I enjoyed the beginning”*

The word *“movie”* was misclassified in the first sentence, whereas it was correctly classified following a minor change in the second sentence. Such observations demonstrate how tricky it is to interpret the behavior of a deep learning model. Therefore, these models are often described as black boxes. In other words, it is not easy to understand their inner working and predictions by humans (Wallace et al., 2019; Horel et al., 2018).

### Sentiment Term Extraction

|  |
| --- |
|  |
| Figure ‎5.5. Frequency of correctly predicted labels by the STE model compared to the actual number within the test data. |

The precision and recall values of the STE model are 72.59% and 58.74%, respectively; thus, the F1-score equals 64.94%. One reason why the model has achieved better performance on this task is the increasing number of sentiment labels (see Figure ‎4.4). Figure ‎5.5 shows the number of correctly predicted labels compared to the total number of labels in the test data after excluding the class label *“O”*. The model correctly predicted 63.79% and 14.24% of actual *“B\_S”* and *“I\_S”* class labels, respectively. Similar to the ATE model, the imbalanced data problem impairs the STE model’s ability to identify labels of the minority classes, i.e., “*B\_S”* and *“I\_S”.* This problem was particularly pronounced when classifying tokens associated with the class label *“I\_S”*. As shown in Figure ‎5.6, in the list of the most misclassified tokens appear those annotated with *“I\_S”* in the test data. For example, *“the”* was incorrectly predicted into the class label *“O”* more than 50 times.

Prediction analysis of the STE model also shows that the low-quality annotations hurt the model’s performance. Evaluating the performance on test data that contain incorrect annotations results in a lower F1-score value. For example, all the following tokens were classified as *“O”*, but they are actually *“B\_S”* and *“I\_S”*:

* *“The video offers circulation stretching, nutritional, tips, a nutritional smoothie recipe, talk with Dr.”*

In this case, the model classification was considered incorrect, although the sentence does not contain any sentiment terms.

|  |
| --- |
| Chart, bar chart  Description automatically generated |
| Figure ‎5.6. Most frequently misclassified tokens by the STE model. |
| Chart, bar chart  Description automatically generated |
| Figure ‎5.7: Frequency of the most correctly classified sentiment terms. |

The most correctly classified sentiment terms are adjectives (see Figure ‎5.7), for example, *“great”* was correctly classified more than 120 times. In addition, verbs such as *“love”* and *“like”* appear (with less frequency) in the list of top 10 correctly classified sentiment terms. According to Liu (2020, pp. 58–59), adjectives, adverbs, and verbs are the most common types of sentiment terms. However, extracting sentiment terms can be much more complex than identifying some individual words. In many types of sentences, authors may use idioms or expressions to convey their feeling/opinion without using explicit sentiment words (Liu, 2020, pp. 58–59). The following are some examples from the test dataset:

* *“a roller coaster ride”*
* *“big box office draw”*
* *“complete full moon”*
* *“icing on the cake”*

All of these examples represent multi-word sentiment terms that are annotated with *“B\_S”* and *“I\_S”* in the test set, but the model misclassified them as *“O”*.

Besides idioms, sentiment terms can also include conditional sentences. Dealing with such sentences was difficult for both human annotators and the STE model. On the one hand, annotators provided inconsistent annotations for conditional sentences. On the other hand, the STE model could detect only some individual words within the conditional sentence. For example, the model identified only *“fan”* and “essential” as sentiment terms, although the entire sentence is annotated with *“B\_S”* and *“I\_S”*:

* *“If you are a Lizzy* ***fan*** *it is* ***essential*** *for your collection”*

Evaluating the model performance using additional test sentences shows that it may not differentiate between the opinionated and not-opinionated sentences, i.e., whether the sentence conveys a positive/negative feeling or not (Liu, 2020, p. 90). For example:

* *“Is Game of Thrones* ***worth*** *watching?”*

In this sentence, the model identified *“worth”* as a sentiment term, but the sentence carries no feeling toward the show *“Game of Thrones”*.

The model also failed to detect several multi-word sentiment terms that include sarcasm. According to a definition provided by Agarwal et al. (2020, p. 238), sarcasm is a *“ form of verbal irony that is intended to express contempt or ridicule, i.e., sarcasm has an implied negative sentiment but may not have a negative surface sentiment. ”* For example:

* *“I dunno if that’s artistry”*
* *“feel like a stage play”*
* *“like watching the sound of someone rake their nails across a chalkboard”*
* *“comes off as something equal to a Saturday night live parody”*

In these example sentences, the model classified all tokens as *“O”*, although they are *“B\_S”* and *“I\_S”*.

In addition, the model could not detect several of the multi-word sentiment terms which involve recommendations or suggestions. For example:

* *“everyone should watch”*
* *“real reason to purchase”*
* *“be aware!”*
* *“don’t loose another hour and a half of your short life watching”*

### Modifier Term Extraction

|  |
| --- |
|  |
| Figure ‎5.8. Frequency of correctly predicted labels by the MTE model compared to the actual number within the test data. |
|  |
| Figure ‎5.9. Frequency of the most correctly classified modifier terms. |

Surprisingly, the MTE model achieved a relatively good performance with a small number of modifier labels. The precision and recall values of the MTE model are 71.48% and 58.80%, respectively. Therefore, the F1-score equals 64.52%. A possible explanation for these results may be the high similarity of the modifier terms. The number of incorrectly predicted labels is 808 while the number of correctly predicted labels 40040. As previously mentioned, this is attributed to the high frequency of the *“O”* labels in the test data. Figure ‎5.8 presents the number of correctly predicted labels compared to the total number of labels in the test data for both class labels *“BM”* and *“IM”*. The model correctly predicted 54.40% and 22.74% of actual *“BM”* and *“IM”* class labels, respectively.

Punctuations and stopwords are the most frequently correctly classified tokens since they belong to the majority class *“O”*. On the other hand, intensifiers appear on the list of the most frequently correctly classified modifiers. As shown in Figure ‎5.9, the intensifier word *“very”* was correctly classified more than 80 times.

Prediction analysis of the MTE model shows that the low-quality annotations increased the number of the misclassified tokens. Furthermore, the model failed to detect many modifier terms that include *“all”* and a *“lot”*. For example, the following tokens were misclassified as *“O”*:

* *“all of them”*
* *“at all”*
* *“all around”*
* *“hell of alot of”*
* *“a lot of”*

It is also noted that the model sometimes misclassifies modifiers when they appear in the same context with some uncommon words in the training data. For example, the model misclassified the modifier *“of all times”* in the following sentence:

* *“a full hour of history legends & the best country music* ***of all times****.”*

However, the model correctly classified this modifier when replacing *“country music”* with more common words in the training data such as *“story”* and *“characters”*. This indicates that increasing the amount of training data may allow the model to learn additional relationships between tokens to deal with different situations. Consequently, increasing training data would improve the model’s performance.

## Conclusions and Future Work

ASM-features identification and extraction are important ABSA tasks because they represent the basis for any ABSA tool. In this work, ASM-features extraction tasks were handled as a set of independent sequence labeling tasks. The main goal was to analyze how ASM-features can be extracted using neural-based architecture. The Bi-LSTM-CRF model architecture was chosen to perform these tasks because it combines the advantages of both the Bi-LSTM network and CRF model. In all evaluated configurations, the Bi-LSTM-CRF model shows a good performance on all tasks without resorting to manually hand-crafted features or intensive text preprocessing techniques, outperforming the Bi-LSTM network on the same test data. The trained model could deal with sentences that involve several explicit ASM-features. Unfortunately, it cannot always detect complex types of ASM-features, such as implicit aspects, idioms, or sarcasm.

The evaluation results have also shown that the ASM-features extraction quality was improved with a combination of embeddings derived from BERT and Flair PTMs. This combination provides a contextualized word representation compared with non-contextual GloVe embeddings and captures richer semantic and syntactic information than the GloVe-Flair combination. Moreover, it enables the model to deal with OOV words as well as spelling and grammatical errors in many cases.

This work also compared two training approaches to learn from a dataset annotated by three different annotators. The quantitative analysis of the model's performance has shown that the model could extract more ASM-features correctly by training on all annotations. However, this result is valid for the dataset used in this work and may not be applicable to all datasets. For example, when the agreement among annotators in a dataset is very low, training a deep learning model on all annotations does not guarantee the best performance because it can mislead the model during training.

Finally, it is worth mentioning that learning to perform ABSA subtasks independently may not be very productive. Future work should focus on the learning methods that allow exploiting the interrelations between these subtasks to perform ABSA, such as multi-task learning (Ruder, 2017b; Yang et al., 2021). In addition, it would be interesting to explore the potential use of active learning methods. In an active learning setting, the model is iteratively trained by a human annotator. The main idea behind this technique is that the model chooses which examples need to be labeled by the annotator. The model can achieve the same performance with less amount of annotated data and thus less annotation effort (Schröder & Niekler, 2020). Another possible area of research would be to compare the interpretation methods for NLP models. Interpreting predictions of a deep learning model is important to increase the model's transparency and ensure it performs well on new data (Horel et al., 2018).

## Reference List

Agarwal, B., Nayak, R., Mittal, N., & Patnaik, S. (Eds.). (2020). *Deep Learning-Based Approaches for Sentiment Analysis*. Springer Singapore.

Aggarwal, C. C. (2018a). *Machine Learning for Text*. Springer International Publishing.

Aggarwal, C. C. (2018b). *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing.

Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., & Vollgraf, R. (2019). FLAIR: An easy-to-use framework for state-of-the-art NLP. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, 54–59.

Akbik, A., Blythe, D., & Vollgraf, R. (2018). Contextual String Embeddings for Sequence Labeling. *Proceedings of the 27th International Conference on Computational Linguistics*, 1638–1649.

Alammar, J. (2018, March 12). *The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)*. Retrieved March 08, 2021, from http://jalammar.github.io/illustrated-bert/

Augustyniak, Ł., Kajdanowicz, T., & Kazienko, P. (2020). Comprehensive Analysis of Aspect Term Extraction Methods using Various Text Embeddings. *ArXiv:1909.04917 [Cs]*.

Augustyniak, Ł., Kajdanowicz, T., & Kazienko, P. (2019). Aspect detection using word and char embeddings with (Bi) LSTM and CRF. *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, 43–50.

Chollet, F. (2018). *Deep learning with Python*. Manning Publications Co.

Chowdhury, G. G. (2003). Natural language processing. *Annual Review of Information Science and Technology*, *37*(1), 51–89.

Deng, L., & Liu, Y. (Eds.). (2018). *Deep Learning in Natural Language Processing*. Springer Singapore.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv:1810.04805 [Cs]*.

Feng, J., Cai, S., & Ma, X. (2019). Enhanced sentiment labeling and implicit aspect identification by integration of deep convolution neural network and sequential algorithm. *Cluster Computing*, *22*(3), 5839–5857.

Ganganwar, V., & Rajalakshmi, R. (2019). Implicit Aspect Extraction for Sentiment Analysis: A Survey of Recent Approaches. *Procedia Computer Science*, *165*, 485–491.

Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, *10*(1), 1–309.

Goodfellow, I., Yoshua, B., & Aron, C. (2016). *Deep Learning*. MIT Press.

Graves, A., Mohamed, A., & Hinton, G. (2013). Speech Recognition with Deep Recurrent Neural Networks. *ArXiv:1303.5778 [Cs]*.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*(5–6), 602–610.

Hapke, H. M., Lane, H., & Howard, C. (2019). *Natural language processing in action, Understanding, analyzing, and generating text with Python*. Manning.

He, Z., Wang, Z., Wei, W., Feng, S., Mao, X., & Jiang, S. (2020). A Survey on Recent Advances in Sequence Labeling from Deep Learning Models. *ArXiv:2011.06727 [Cs]*.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780.

Horel, E., Mison, V., Xiong, T., Giesecke, K., & Mangu, L. (2018). Sensitivity based neural networks explanations. *ArXiv:1812.01029 [Stat.ML]*.

Hu, M., & Liu, B. (2004). Mining opinion features in customer reviews. *AAAI*, *4*(4), 755–760.

Huang, Z., Xu, W., & Yu, K. (2015). *Bidirectional LSTM-CRF Models for Sequence Tagging*.

Jakob, N., & Gurevych, I. (2010). Extracting opinion targets in a single and cross-domain setting with conditional random fields. *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 1035–1045.

Jin, W., Ho, H. H., & Srihari, R. K. (2009). A novel lexicalized HMM-based learning framework for web opinion mining. *Proceedings of the 26th Annual International Conference on Machine Learning*, *10*(1553374.1553435).

Klie, J.-C., Bugert, M., Boullosa, B., de Castilho, R. E., & Gurevych, I. (2018). The inception platform: Machine-assisted and knowledge-oriented interactive annotation. *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, 5–9.

Krüger, F. (2018). *Activity, context, and plan recognition with computational causal behavior models*. Universiät Rostock. Fakultät für Informatik und Elektrotechnik.

Lafferty, J., McCallum, A., & Pereira, F. C. (2001). *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural architectures for named entity recognition. *ArXiv Preprint ArXiv:1603.01360*.

Le, X. H., Ho, H., Lee, G., & Jung, S. (2019). Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting. *Water*, *11*, 1387.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Li, J., Sun, A., Han, J., & Li, C. (2020). A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering*.

Li, M. (2017, December 9). *CRF Layer on the Top of BiLSTM - 1*. CreateMoMo. Retrieved March 15, 2021, from http://createmomo.github.io/2017/09/12/CRF\_Layer\_on\_the\_Top\_of\_BiLSTM\_1/index.html

Liu, B. (2020). *Sentiment Analysis Mining Opinions, Sentiments, and Emotions* (2nd Edition). Cambridge University Press.

Lu, Y., & Salem, F. M. (2017). Simplified Gating in Long Short-term Memory (LSTM) Recurrent Neural Networks. *ArXiv:1701.03441 [Cs, Stat]*.

Marrese-Taylor, E., & Matsuo, Y. (2017). Replication issues in syntax-based aspect extraction for opinion mining. *ArXiv Preprint ArXiv:1701.01565*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *ArXiv:1310.4546 [Cs, Stat]*.

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., & Gao, J. (2021). Deep Learning Based Text Classification: A Comprehensive Review. *ArXiv:2004.03705 [Cs, Stat]*.

Ni, J., Li, J., & McAuley, J. (2019). Justifying recommendations using distantly-labeled reviews and fine-grained aspects. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 188–197.

Olah, C. (2015, August 27). *Understanding LSTM Networks*. Retrieved February 28, 2021, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543.

Qiu, G., Liu, B., Bu, J., & Chen, C. (2011). Opinion word expansion and target extraction through double propagation. *Computational Linguistics*, *37*(1), 9–27.

Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., & Huang, X. (2020). Pre-trained Models for Natural Language Processing: A Survey. *ArXiv:2003.08271 [Cs]*.

Ramshaw, L., & Marcus, M. (1995). Text Chunking using Transformation-Based Learning. *Third Workshop on Very Large Corpora*.

Raykar, V. C., Yu, S., Zhao, L. H., Valadez, G. H., Florin, C., Bogoni, L., & Moy, L. (2010). Learning from crowds. *Journal of Machine Learning Research*, *11*(4).

Ruder, S. (2017a). An overview of gradient descent optimization algorithms. *ArXiv:1609.04747 [Cs]*.

Ruder, S. (2017b). An Overview of Multi-Task Learning in Deep Neural Networks. *ArXiv:1706.05098 [Cs, Stat]*.

Schröder, C., & Niekler, A. (2020). A Survey of Active Learning for Text Classification using Deep Neural Networks. *ArXiv:2008.07267 [Cs]*.

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681.

Skansi, S. (2018). *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer International Publishing. https://doi.org/10.1007/978-3-319-73004-2

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, *45*(4), 427–437.

Sun, Y., Wong, A. K., & Kamel, M. S. (2009). Classification of imbalanced data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, *23*(04), 687–719.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. *ArXiv:1706.03762 [Cs]*.

Wallace, E., Tuyls, J., Wang, J., Subramanian, S., Gardner, M., & Singh, S. (2019). Allennlp interpret: A framework for explaining predictions of nlp models. *ArXiv Preprint ArXiv:1909.09251*.

Yang, H., Zeng, B., Yang, J., Song, Y., & Xu, R. (2021). A multi-task learning model for chinese-oriented aspect polarity classification and aspect term extraction. *Neurocomputing*, *419*, 344–356.

Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing. *Ieee Computational IntelligenCe Magazine*, *13*(3), 55–75.

Zhao, J., Mudgal, S., & Liang, Y. (2018). Generalizing Word Embeddings using Bag of Subwords. *ArXiv:1809.04259 [Cs]*.

Statement of independent work

I hereby confirm that this thesis was written independently by myself without the use of any sources beyond those cited, and all passages and ideas taken from other sources are cited accordingly.

Ich habe die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt. Die elektronische Ausfertigung der Arbeit habe ich bereits beim Prüfer eingereicht.

|  |  |  |
| --- | --- | --- |
| Regensburg, 19.07.2021 |  | A picture containing icon  Description automatically generated |
| Ort, Datum |  | Unterschrift |

1. https://s3.amazonaws.com/models.huggingface.co/bert/bert-base-uncased-vocab.txt [↑](#footnote-ref-1)
2. https://github.com/flairNLP/flair [↑](#footnote-ref-2)