Bachelor's Thesis

Context-based Sentence Segmentation for Irregular Domains

Krisztina Agoston

Examiner: Prof. Dr.Hannah Bast
Adviser: Matthias Hertel

University of Freiburg

Faculty of Engineering

Department of Computer Science

Chair for Algorithms and Data Structures

Januar 17th, 2022

Writing Period

01. 10. 2021 – 17. 01. 2022

Examiner

Prof. Dr.Hannah Bast

Adviser

Matthias Hertel

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

Sentence segmentation is a fundamental task in Natural Language Processing (NLP). Its purpose is to find sentence boundaries within a text, divide a it into individual sentences, and provide them for follow-up processes in the NLP pipeline.
However, sentence segmentation is regarded as a solved problem, but text from an irregular domain still presents a challenge. For this reason, we focus on specific domains with special syntax and a variety of punctuation.
We propose for this task two context-based approaches: a Long Short-term Memory (LSTM) model trained on a large collection of sentences. We also present a variation of this model, the bidirectional LSTM, offering the ability to use not only the previous, but also the future information in a sequence. We compare these models to two state-of-the-art systems: NLTK and spaCy.
Our system returns competitive results to the solutions of these systems, with an F1-Score of 91.39% on hard cases and 96.94% on randomly selected paragraphs from the Wikipedia corpus.

Zusammenfassung

Sentence segmentation ist eine grundlegende Aufgabe in der Verarbeitung natürlicher Sprache (NLP). Sie dient dazu, Satzenden in einem Text zu finden, den Text in einzelne Sätze aufzuteilen und für Folgeprozesse in der NLP-Pipeline bereitzustellen. Dieses Problem wird jedoch als gelöstes Problem angesehen, aber Text aus einer unregelmäßigen Domäne stellt immer noch eine Herausforderung dar. Aus diesem Grund konzentrieren wir uns auf bestimmte Domänen mit spezieller Syntax und einer Vielzahl von Satzzeichen. Für diese Aufgabe schlagen wir ein Long Short Term Memory (LSTM)-Modell vor, das an große Anzahl von Sätzen trainiert wurde. Wir präsentieren auch eine Variation dieses Modells, das bidirektional LSTM, das die Möglichkeit bietet, nicht nur die vorherigen, sondern auch die zukünftigen Informationen in einer Sequenz zu verwenden. Wir vergleichen diese Modelle mit zwei State-of-the-Art-Systemen: NLTK und spaCy. Unser System übertrifft die Lösungen dieser Systeme mit einem F1-Score von 91.39% bei harten Fällen und 96.94% bei zufällig ausgewählten Absätzen.

Contents

1 Introduction

1.1 Task definition

Natural Language Processing (NLP) includes all the processes performed by computers working with natural language, like speech and written text. NLP manipulates textual data and the multiple steps of this process build a pipeline. Many tasks in this pipeline require correctly defined sentences as their input. The task that splits a text into sentences is called sentence segmentation. As in this essential step of the NLP pipeline our aim is to locate the correct sentence boundaries, this task is often called sentence boundary detection (SBD). In the next section, we will describe the difficulty of the problem in detail.

1.2 Complexity of the problem

The task seems to be as simple as splitting sentences after each period. However, period-usage is ambiguous, because it defines not only the end of a sentence but it is also used for other purposes within a sentence:

- Abbreviations:

  U.S. Special Operations Forces

  Mr. Brown
- Initials: *J. F. Kennedy*
- Ordinal numbers:

  Table 1.

  Claim 3.

- Ellipses:

  "I have a dream....I have a dream today."

  “Did he ... peacefully?” she asked.

Sadvilkar et al. [1] state as a further problem the domain specific use of punctuation. For example, drug names in medical documents, case citations in legal texts and references in academic articles. These irregular domains use punctuation in ways that are uncommon in newswire documents. The segmentation is even more complicated if a sentence ends with an abbreviation. In this case, the period in the abbreviation also serves as the full stop to end the sentence:

> *His helmet is at the National Museum of American History in the Smithsonian museum in Washington D.C.*

A structure containing one or more sub-sentences makes the problem more complex:

> *Ronald Reagan would evoke both the sentiment and the legacy of Kennedy’s speech 24 years later in his "Tear down this wall!" speech.*

The difference between British and American English in the position of quotation marks can also cause errors in the process of determining sentence boundaries:

> *British: "Joy means happiness”.*
> *American: "Joy means happiness."*

Because of these ambiguities, is is difficult to denote a punctuational mark as a sentence boundary. However, the scale of the difficulties highly depends on the domain. Wong et al. [2] state that in the Wall Street Journal (WSJ) about 42% of the periods denote abbreviations and decimal points, while in the Brown corpus it is only 11%. That means that just by treating every period as a sentence boundary, we can isolate sentences with 58% and 89% accuracy, respectively.

Besides the difficulties mentioned above, we also have to consider the different characteristics of the languages. However, in some languages the usage of the punctuation marks differs from the western tradition, where the sentence-ending punctuations are the period, question mark, and exclamation mark (www.translatemedia.com) [3]. These differencies are for example:

- Hindi or Tibetian uses a straight line instead of period;
- Armenian full stop looks similar to colon;
- the Greek question mark is like the semicolon ;
- Thai uses no symbols at all to mark sentence ends.

This demonstrates that SBD systems are genre- and domain-dependent, but also language-sensitive. This dependency also makes portability hard and the task furthermore to an unsolved problem.

1.3 Motivation

The importance of the SBD task lies in the fact that a lot of NLP tasks take a sentence as basic input (spam filter, machine translation, sentiment analysis, POS tagging, text correction). That means every error in SBD propagates into the further steps and affects the overall performance of the later tasks. Although there are several systems with different approaches from the last three decades, they can not deal with special domains and their special period usage, resulting in a limited accuracy of these systems. These systems are evaluated mostly on newswire texts that are well-formed and structured without special abbreviations, for example. Therefore these systems do not generalize to more difficult domains and their evaluation results are hardly held in practice.

1.4 Contribution

In this thesis our goals are the following:

- We present a context-based system for the sentence boundary detection task using a long short-term memory (LSTM) and bidirectional long short-term memory (Bi-LSTM) neural network.
- We study variants of these models at the character and token level.
- We compare the different versions of our model depending on various methods of creating a context window.
- We compare our models to the state-of-the-art system spaCy and the popular statistical system Punkt included in NLTK.
- We evaluate not only on newswire corpus, but also on special domains as Wikipedia or arXiv.
- With regard to the evaluation, we deal with the problem of unbalanced data.

This work is organized as follows.

In the next section, we review some previous approaches to the sentence segmentation problem. In Section 3, we describe the machine learning approach that we used for building our model. After describing the process of creating our datasets for training and evaluation in Section 4 we give a review of the benchmarks we used in the evaluation in Section 5. In Section 6, we describe the metrics for evaluating our segmenter and compare different approaches to the sentence segmentation problem in a series of experiments.

2 Related Work

In this section, we give an overview of the development of approaches to the sentence segmentation problem.

There have been several systems for the SBD problem in the last 30 years with competitive results. These systems can be categorized into three groups:

- rule-based SBD, using handcrafted grammars and lists of abbreviations;
- statistical approaches;
- machine learning approaches.

And these approaches can be further subdivided into supervised and unsupervised applications, depending on the training method if required: the model trained on labelled data is supervised, unlike unsupervised training on raw, unannotated text.

2.1 Rule-based approaches

The rule-based approach tries to grasp patterns in the language by defining rules and promises more control over ambiguity issues. The first notable rule-based system is from Mikheev et al. (2002) [4]. It dealt with the two most frequent sources of ambiguity: names that begin with uppercase letters and are not necessarily a sentence start, like *'Dr. Brown'*, and abbreviations that have a period but not necessarily end a sentence, like *'etc.'*. This method uses a small set of rules for determining sentence boundaries, relying on the information about abbreviations and proper names gained from the local context in the document. The reported low error rates depend on the correctness of this information and vary between 0.01%

and 2.0% on the Brown Corpus and 0.13% to 4.0% on the Wall Street Journal (WSJ) Corpus.

PySBD The most recent example of rule-based methods is PySBD by N. Sadvilkar et al (2020) [1]. This system adapts a set of hand-constructed rules for 22 languages and is designed to cover a variety of domains. It is compared to spaCy's dependency parser based on supervised machine learning and the unsupervised NLTK, which is relevant for our experiment. In English it gives 97.00% accuracy on the Genia corpus, while NLTK gives 87.95%. This method includes all the positive features of rule-based systems: no need for training data, can be extended or adjusted or set up for a new language, and errors are interpretable. On the other hand, rule-based systems require a large amount of manual effort for setting up rules and encoding language features, and it is also hard to maintain the system. As the set of abbreviations is not closed and we cannot have a list of all possible abbreviations, the system based on such a list is always fragile [5]. In a regular corpus, the rule-based systems can give a satisfactory result. However, these systems are mostly not robust and flexible enough and can perform badly on a special corpus with irregularities or ill-formed text unlike to newswire texts or literature. For these reasons, they are highly language and genre dependent.

2.2 Statistical models

Statistical models capture the features of the language by collecting statistics about different patterns. They use, for example, statistics about the occurrences of common sentence starter words or calculate conditional probability for tokens followed by a period to classify a punctuation as a sentence boundary. A shortcoming of the statistical approach is the high complexity of such systems. They also need a large amount of data to work properly.

Punkt A remarkable and popular unsupervised statistical SBD model is Punkt from Kiss and Strunk (2006) [6], which is also used by the leading Python library NLTK (2006) [7]. It does not need any additional annotations of sentence boundaries in the training data or lists of abbreviations. This token-based model relies on the observation that an abbreviation and its following period appear mostly together and build a collocation. Through filtering out abbreviations a large percentage of sentence boundaries can be found accurately. In order to detect the abbreviations, the system collects statistics about occurances of tokens, punctuations, token length, casing and also collocations of tokens. It calculates probabilities based on these statistics and uses them in testing heuristics. These heuristics are basic rules that are used to decide if a token is e.g. a frequent sentence starter, or build a collocation with a period. The results are used to classify a token as a sentence boundary. For the learning process, it requires only a larger amount of unlabelled text from the same domain as the target text. Their reported rates of errors on 'classic' test sets are 1.02% (Brown) and 1.65% (WSJ).

2.3 Machine learning approaches

The most recent approaches focus on machine learning. Applying algorithms using neural networks to this task, we can overcome the limitations of the previous systems. We need neither hand-crafted pattern definitions nor manually built statistical systems. With neural networks the resulted automated systems transfer the natural language into a subject of a mathematical function. This task becomes a binary classification problem that labels a punctuation mark as a sentence boundary or not.

As neural networks work only with numbers, the words must be represented by numbers. These numbers can describe different aspects of the word and build a feature vector, often with hundreds of dimensions.

$$\text{beautiful} \Rightarrow [0.2, 40.1, 3.2, -0.123]$$

One of the earliest machine learning approaches is from Riley (1989). He encodes the words using their characteristics like word length and case, probability of beginning or ending a sentence, etc. The words are represented by these feature vectors. The reported accuracy of this method is 99.8% on Brown corpus. However, Riley's approach treats the SBD problem simply as a disambiguation of a period, which is stated by Read et al. [8] as an extreme limitation of the task .

Satz The list of potential sentence boundaries has been widened in the system called Satz by Palmer & Hearst (1997) [9], to period, question mark and exclamation mark. The method combined two different learning approaches: neural network and decision tree. It used context around a punctuation mark: the tokens immediately preceding and following each punctuation mark were taken into account. Words are represented by the probability that a word occurs as each part of speech (POS). These probabilities are estimated using a lexicon containing part-of-speech frequency data from an external source. For unknown words, it uses a list of heuristics. On Wall Street Journal (WSJ) the original error rate of 1.1% using neural net increases only to 3.3% by lower-case-only format text, and to 3.5% by upper-case-only format. It still used hand-crafted rules for heuristics and a list of abbreviations and a lexicon with part-of-speech frequencies for word representation.

Elephant The idea of composing feature vectors was further developed by Evang et al. (2013) [10]. The character-based system called Elephant uses a recurrent neural network (RNN). RNNs are commonly used for sequential data with the possibility of feeding the network with the prior output in the current time step, as depicted on Figure 1 [11].

Elephant uses a RNN to learn to predict the next character from a raw, unlabelled text. After the training, the output value of the hidden layer will be saved as a feature. This automatically learned vector representation will be combined with other

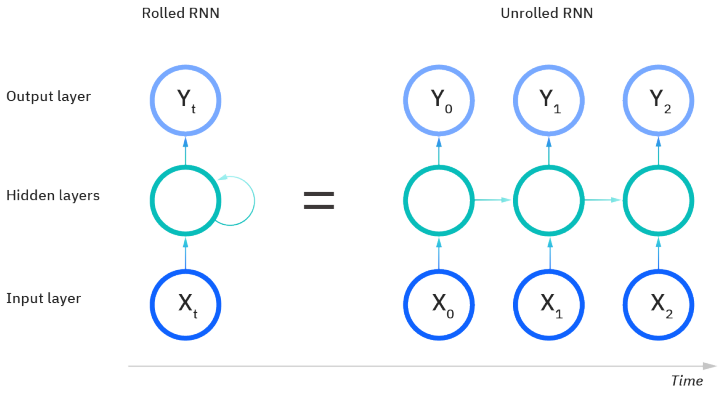


Figure 1: Feed back the output of the Recurrent Neural Network (adapted from ibm.com [11]

hand-crafted features (Unicode categories, Unicode character codes, and combination of the two). This is then used in the training of a conditional random field (CRF) which is commonly used for modelling sequential data, labelling a single sample considering the "neighboring" samples. Elephant uses the idea of IOB tags and predicts for each character one of the following tags:

- I for inside,
- O for outside,
- T for beginning of a token and
- S for sentence starter.

This system reports a high accuracy with an error rate of 0.27‰ and 100% F1 score on English newswire text taken from the Groningen Meaning Bank. It should be noted that this data set contains only 2,886 sentences, with 80 percent of them being used for training. For evaluation 10% of the sentences were used, resulting in about 280 sentences to evaluate on. The error rate is calculated combined for tokenization and sentence segmentation, making comparison to any other system hard.

spaCy The NLP library spaCy (www.spacy.io) uses another type of neural network for pattern recognition, a convolutional neural network (CNN). CNN is a neural

network designed to identify patterns in images. It is used in NLP to detect patterns: weight matrices called filters slide through the word vectors, searching for a pattern. These filters are also learned during the training process.

spaCy's sentence segmentation component, the Parser, uses CNN. The Parser is based on syntactic dependency labels. The dependency tree describes the relations between parts within a sentence. The sentence boundaries are derived from this parse tree. To train the Parser, one would need labeled data containing sentences flagged with the dependencies depicted in Figure 2 [12], making portability complex. The Parser works on common domains well, but an irregular text causes issues because the inserted formulas, references, etc. break these dependencies.

On spaCy's website, a full pipeline accuracy of 0.951 is stated on OntoNotes 5.0 corpus in English. A more detailed evaluation on a 13 MB small sample from this corpus reports an F1-score of 0.90, precision of 0.92, and recall of 0.89. The evaluation is available for different languages, for German for example an F1-score of 0.93 is stated. Ortmann et al. (2019) report an F1-score of 0.8940 for Parser on a mixed data from five selected text types in German containing a total of 559 sentences [13].

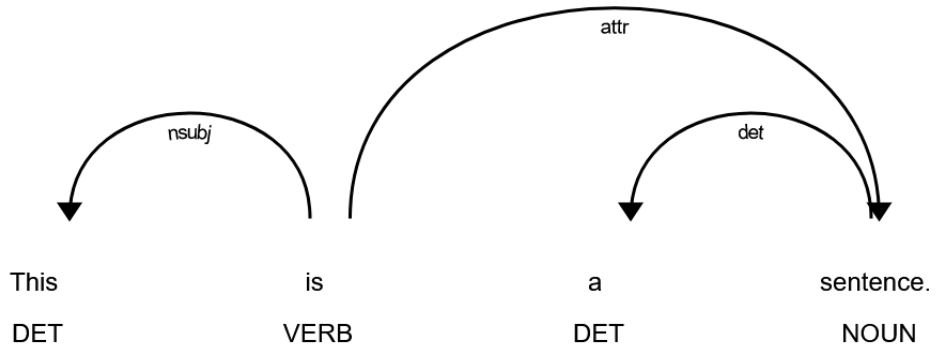


Figure 2: Dependencies visualized within a sentence (adapted from spacy.io) [12]

spaCy has another sentence segmenter component, the SentenceRecognizer (Senter). It uses supervised learning which means we need data samples with their expected outputs. This labelled data contains separate sentences where the sentence starter token is annotated. This is easier to provide than training data for the dependency

parser. There is no evaluation available on the Senter component on the website. Besides the above mentioned source there is no paper where spaCy's system has been described or evaluated as far as we know.

LSTM The last two approaches we present use a variation of the RNN, the Long short-term memory (LSTM) neural network. LSTM (S. Hochreiter, J. Schmidhuber, 1997) is an architecture that solves the problem of long-term dependencies. That is being unable to connect contextual information within a sequence if they are very far apart from each other. RNN cannot predict correctly if the previous state that influences the current output is not from the last several time steps. In a task to predict the next word, RNN would fail on the second example in Figure 3:

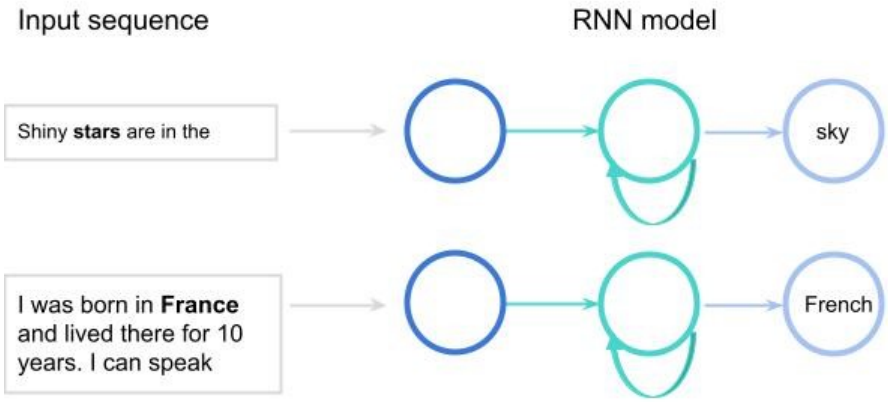


Figure 3: Long dependency problem by RNN. Using RNN for predicting the next word for a sequence. The relevant context is written bold. Examples are adapted from Manik Soni [14]

This feature is very useful for catching the connection between words or characters in a sentence. Focusing on this type of neural network, we give a detailed description of the LSTM architecture in Section 3.1.

The first system that presents both LSTM and its bidirectional variant bi-LSTM is from Toleu et al. (2017). Toleu et al., unlike Elephant, abandon task-specific feature engineering in favor of learned embeddings. These automatic embeddings are a matrix of automatically learnt weights. The weights encode many linguistic features of each word in the vocabulary [15]. Toleu et al. (2017) compare the LSTM

networks to a fully connected feedforward network. That is commonly referred to simply as neural networks, and we give a short description of it in Section 3.
The system using the feedforward network makes a prediction for a middle character in a context window depending on the left and right adjacent characters in the window, which is similar to the idea of the bi-LSTM model. On the development set, the LSTM model achieves a 99.34% precision, recall, and F-measure. The bi-LSTM has also 99.34% recall, but a higher precision (99.67%) resulting in a higher F-measure (99.5%) on an English newswire corpus. The fully connected neural network model outperforms the two LSTM models with 100% in all the metrics. Toleu's et al. (2017) [16] explanation is that the LSTM models are more sensitive to the size of the training data and need a larger training set to achieve higher scores. As the English newswire domain contains only 2886 sentences, we think that the networks could be overfitted, resulting in nonrealistic high scores on the training data. This assumption is supported by the results of the feedforward neural network evaluated on the Italian language with 42,674 sentences, where the F-measure drops to 97.78%.
On the Kazakh data set the performance drops further, which Tolue et. al. explain by the noisiness of the data. In the final evaluation on the test set, all the models, including Elephant, have a perfect F-measure, whereas Punkt has 98.51%. In Italian which has larger training data, all the models are outperformed by Elephant and Punkt, with a 99.51% and 98.34% F-measure respectively. The bi-LSTM model improves 11 times over Elephant in error rate. However, in terms of error rate, the models are not comparable with Punkt because they use a combined error rate for sentence and token segmentation.

The comparison of different models is difficult due to lack of common test benchmarks and metrics, as Wicks et al. (2021) also states [17]. Some approaches are evaluated on the Wall Street Journal (WSJ) and Brown corpora, but others on Europarl or other newswire corpus. Metrics differ in the evaluation too: some report only the error rate, the others only the F1-score or accuracy. The least interpretable is the

combined evaluation of tokenization and sentence segmentation of Elephant.

Deep-EOS The second approach, Deep-EOS from Schweter et al. (2019) [18] includes a CNN along with the two LSTMs network models. According to the evaluation on different languages, the LSTM and bi-LSTM models perform better as the CNN model, giving the lowest F-measure 97.59% on German SETimes corpus.

The set of potential end-of-sentence marks is restricted to (?, !, :, ;, .) by these last approaches. That is broader compared to Riley (1989). However, this restriction rules out the languages not using the western type of punctuation, or text with missing punctuation, but also in American English, where the closing quotation mark stands after the period.

Both of these machine learning approaches contain the positive features of a machine learning system: they require neither rule nor feature engineering. This way, the inflexibility of the rule-based systems is solved as manually constructed rules are avoided. They are character-based, so they do not depend on previous tokenization. That makes portability easier and the system can be adapted to different languages and corpora by training on the target domain and language.
We can observe that the models can perform outstandingly on a well-formed newswire text. But if the corpus contains more irregularities, the sentence segmentation models' performance declines.

3 Implementation

To solve the SBD problem, we chose a machine learning approach, as it was successfully applied to NLP tasks and significantly outperformed traditional approaches, becoming a state-of-the-art technology (Hang Li et al. 2018) [19]. Machine learning is based on neural networks - a system inspired by the way the human brain learns and works. Like neurons in the brain, nodes (depicted in Figure 4) are building up a network to transform a piece of information into a prediction.

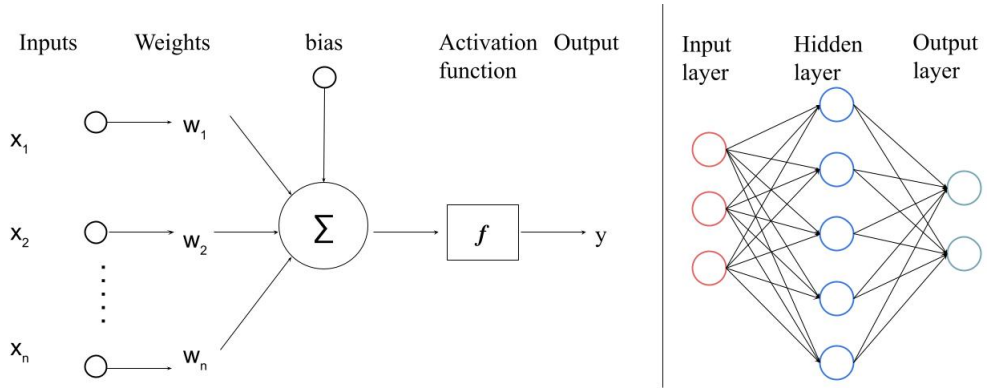


Figure 4: a) Operation within a neuron
b) Neurons building a neural network ([20])

During the training of a neural network, the goal is to find a nonlinear function that maps the input to the desired output to model their complex relationship. This function contains the inputs (x_i) multiplied with weights (w_i) and modified with bias (b). In the training phase, the weights are updated until the network gives an output close enough to the desired output.

$$\text{Input} \Rightarrow \sigma(\sum(Weights \cdot Input) + bias) \Rightarrow \text{Output}$$

A network can contain more than one hidden layer, resulting in a deep neural network.

There are several types of neural network architectures designed to deal with different problems: recurrent neural networks are used to solve problems related to time series data, while convolutional neural networks are designed especially for image and video processing.

For the SBD problem, we chose a special kind of recurrent neural network, because traditional (feedforward) neural networks cannot involve previous inputs in the current calculation. We create two different architectures of neural networks: LSTM and bi-LSTM, that we describe in the next section.

3.1 LSTM model architecture

LSTM (Hochreiter and Schmidhuber, 1997; Gers et al., 2000) is a type of neural network that is preferred to be used for sequential data because it is capable of learning long-term dependencies. If there is a large gap between the prediction and the relevant information, it can "remember". Therefore, LSTM with its special architecture can be used for the current Sentence Boundary Detection task: we want to predict, whether a token is a sentence boundary or not, based on the previous tokens.

The **input layer** provides data in the correct form for the next layers in the neural network. LSTM needs the input as a three dimensional matrix with the dimensions *batch size x sequence length x embedding size*. Our model can work on a token or a character level, they are called basic units in the following. In our model, the raw text is converted into context windows, containing indices of the most frequent tokens or characters. The units with a frequency of 1 we substitute with the UNK tag. The whole process of creating the context windows is described in section 4.1.

The **embedding layer** encodes the units' indices into vectors. Instead of hand-engineered features, we use embeddings that get updated in the learning process,

automating the feature extraction. The embeddings are vectors that capture meaningful information to represent the basic unit in the model (see Figure 5). The embedding vectors are stored in a look-up table dimensioned by the vocabulary and the predefined depth of the embedding vector.

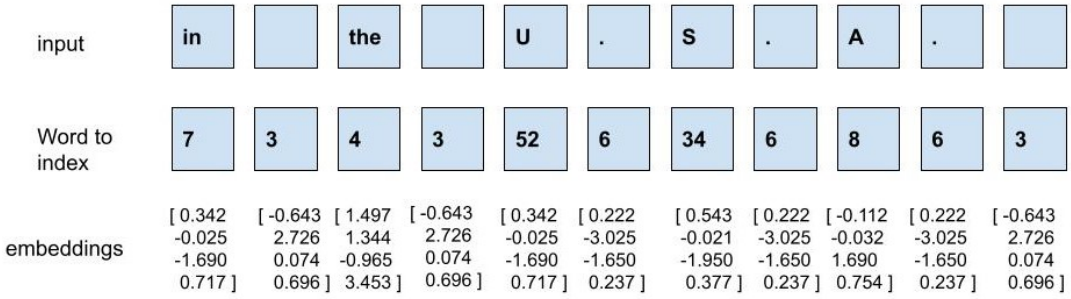


Figure 5: Embedding vectors for words

The **hidden layer** consists of LSTM units depicted in Figure 6. These units get a context window, iterate through the basic units, and perform for each of them a non-linear computation with randomly initialized weight matrices at the beginning.

Within a unit, there is a cell state (ct), which is a kind of memory chaining through all the time steps. An LSTM is distinguished by its gates: an input gate, an output gate, and a forget gate, all these are modifying the cell state. These gates are layers in the network that add or remove information from the cell state by using an activation function. The activation function acts like a filter and decides whether a unit should be activated or not. All three gates work with a sigmoid function, which is a nonlinear function that transforms its input to a number between 0 and 1:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

These gates are the essence of LSTM, being themself neural networks. Using them, we can update the cell state and calculate the new hidden state:

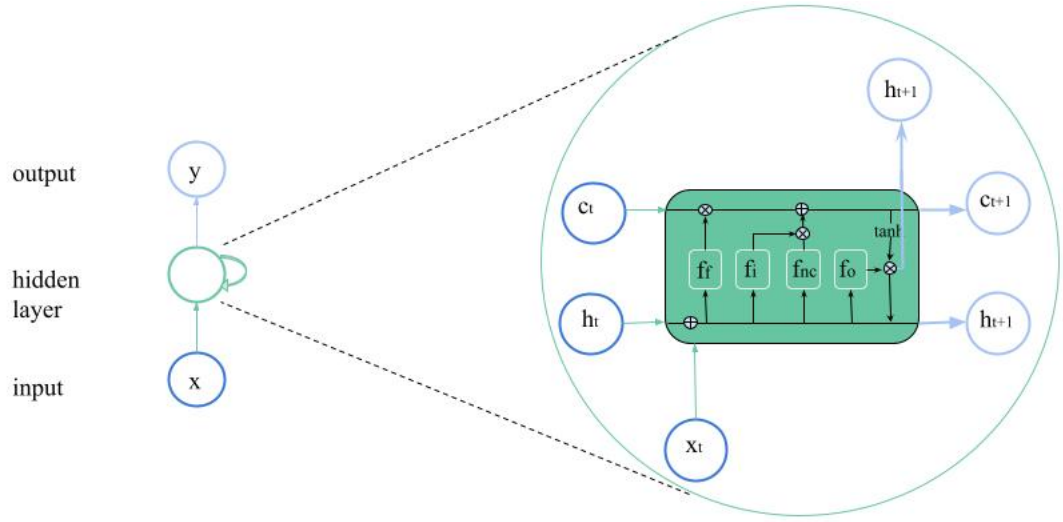


Figure 6: LSTM unit

The **forget gate** filters the old state and decides what information to discard.

$$f_f = \sigma(W_f \cdot (h_{t-1}, x_t) + bias_f)$$

That is done by multiplying the previous cell state by the output of the forget gate. This output is close to 0 for irrelevant information and closer to 1 for relevant information.

The **input gate** decides what information to let through from the current input x and should be added to the tong-term memory.

$$f_i = \sigma(W_i \cdot (h_{t-1}, x_t) + bias_i)$$

The new input will be regulated between -1 and 1 by the tanh layer, resulting in a **new candidate**:

$$f_{nc} = tanh(W_nc \cdot (h_{t-1}, x_t) + bias_nc)$$

$$\text{where } tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Through negative outputs of the tanh function, it is possible to decrease the impact of this component in the cell state. We multiply this new candidate with the result of the input gate and update the cell state with it.

The **output gate** calculates how much of the input is used in our output h_t,

$$f_o = \sigma(W_o \cdot (h_{t-1}, x_t) + bias_o)$$

The updated cell state will be sent through a tanh function again and multiplied by the output of the output gate.

The **linear layer** transforms the output of the hidden layer to the label space. That is a dimension of two, - as we have two kinds of labels: 1 for sentence boundary and 0 otherwise. The output of the linear layer - the logits - is scaled with a softmax layer. The softmax function takes a vector as input and transforms the real numbers within a vector into real numbers between 0 and 1 so that they sum up to 1. These normalized values are interpreted as probabilities for the given label.

$$softmax(\overrightarrow{v}) = \frac{e^{\overrightarrow{v}_i}}{\sum_{j=1}^{k} e^{\overrightarrow{V}_j}}$$

The **output layer** transforms the probabilities from the softmax layer into labels with the help of a threshold. All the context windows with a probability higher than this threshold get a label of 1, meaning that the last unit in the window is a sentence boundary. The complete model architecture is shown in Figure 7.

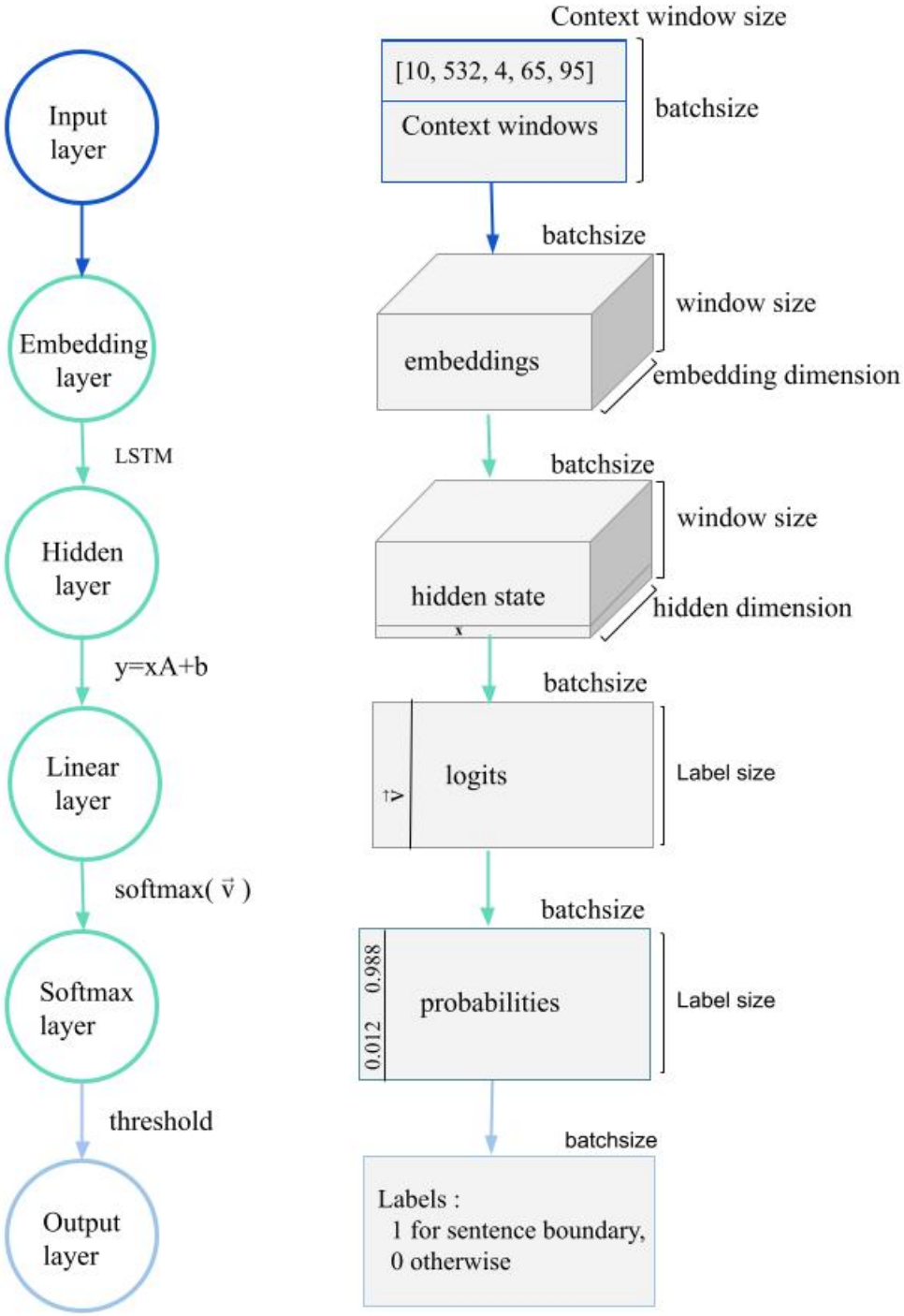


Figure 7: LSTM model architecture

3.2 Bidirectional LSTM

Bidirectional LSTM or Bi-LSTM has a similar architecture, but it has an additional layer that calculates on the reversed input as depicted in Figure 8. This is a useful feature, because in this way, we can also get information about the context after the basic unit we want to label. The Bi-LSTM has respectively two outputs: forwards and backwards. We can combine these by performing some arithmetic calculation, usually addition or multiplication. We use addition in our Bi-LSTM model.

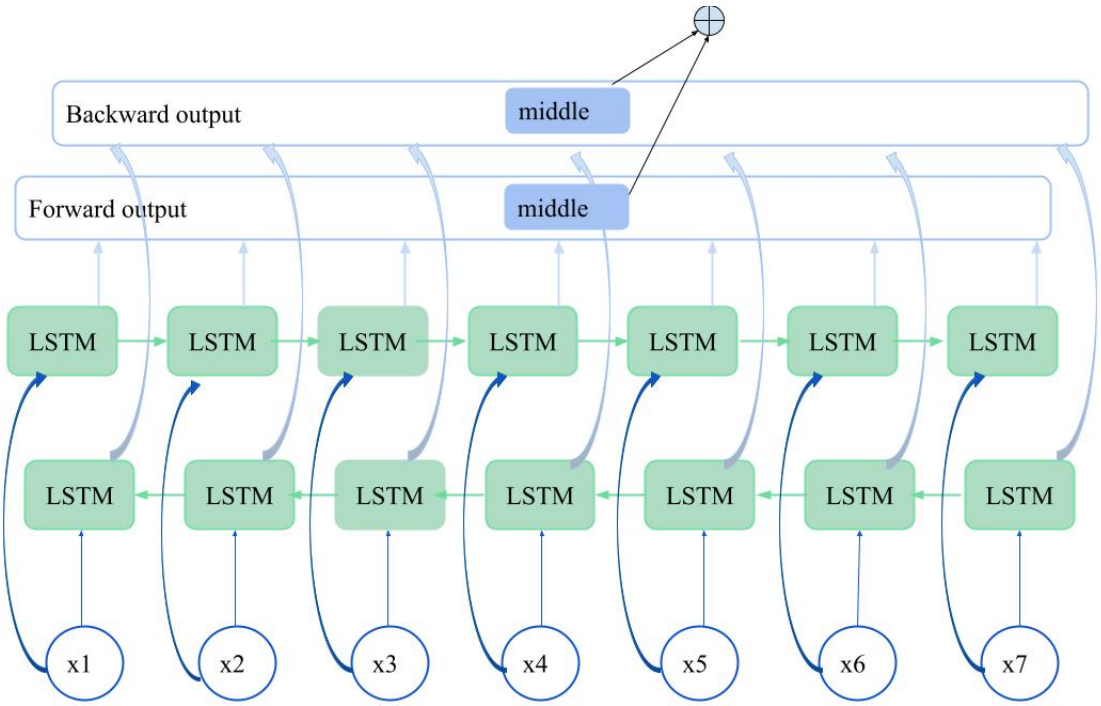


Figure 8: Bi-LSTM hidden layer

The models were implemented in Python with the Pytorch machine learning framework.

4 Approach

To be able to use the advantage of LSTM on long-term dependencies, we create sequences as the input for our LSTM and bi-LSTM models. These sequences are context windows containing basic units, including also the one we predict for. The process of data preparation is described in the next section.

4.1 Data preparation

Our training data is an arbitrary text split into sentences. In order to obtain context windows that we can provide as input for the models, we have to preprocess this data. We use two different approaches to compare token-based and character-based models. For the character-based approach, we create training data from an input text by splitting it into characters with list comprehension, also keeping the whitespaces.

[char for char in sentence]

In the token-based approach, we split the input text into tokens with a simple regular expression pattern, also keeping the whitespaces. The pattern $\backslash W$ matches every non-word character, and splitting on that gives us every word, whitespace, and punctuation in a string. We also keep every whitespace between the units, because the whitespace carries important information: '*U.S.A*' should not be separated, however '*she left the U.S. A year later*' should be split.

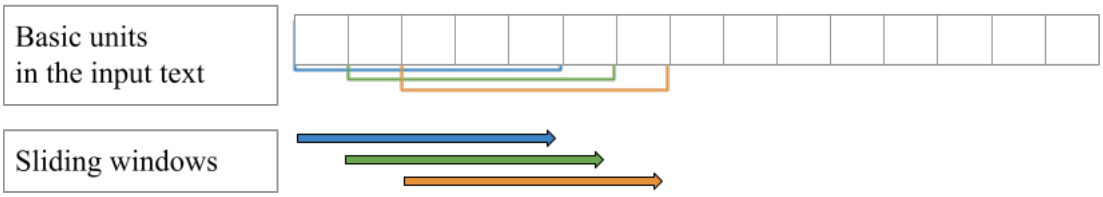
Labels Each unit at the end of a sentence is defined as a sentence boundary, and every other unit where the text should not be split as not sentence boundary. Each unit will be labelled with a 0 or 1 accordingly.

Context windows The input text will be transformed into equally sized context windows containing consecutive units. We train models both character- and token-based. There are three methods to compare: a context window

- for every basic unit - token or character
- only for any punctuation in the input text
- only for sentence end marks.

In the first approach, we create a context window with the sliding window method (see Figure 9) for every basic unit in the training data.

Figure 9: The sliding window method



In models focusing on every punctuations, the context windows are created only for punctuations in the text. We consider here any kind of punctuation, including comma, appostrophe or brackets.
In the third approach, we focus on possible sentence-ending marks and create a context window on the following punctuations: ", ',), !, ?, .,]. We create context windows of size k, containing k-1 previous units for unifirectional LSTM models, where we label the last unit. For the bidirectional version, we take $\frac{k}{2}$ (rounded down to the next integer) units before and after the middle unit for a context window. The different methods are demonstrated in Figure 10.

For bi-LSTM, we make a prediction for the unit in the middle of the context window using the previous and following basic units. To be able to predict label for units with less context as our context window we use a padding unit *PAD* for the first basic units. We also use this padding for the last units in the bi-LSTM model. The

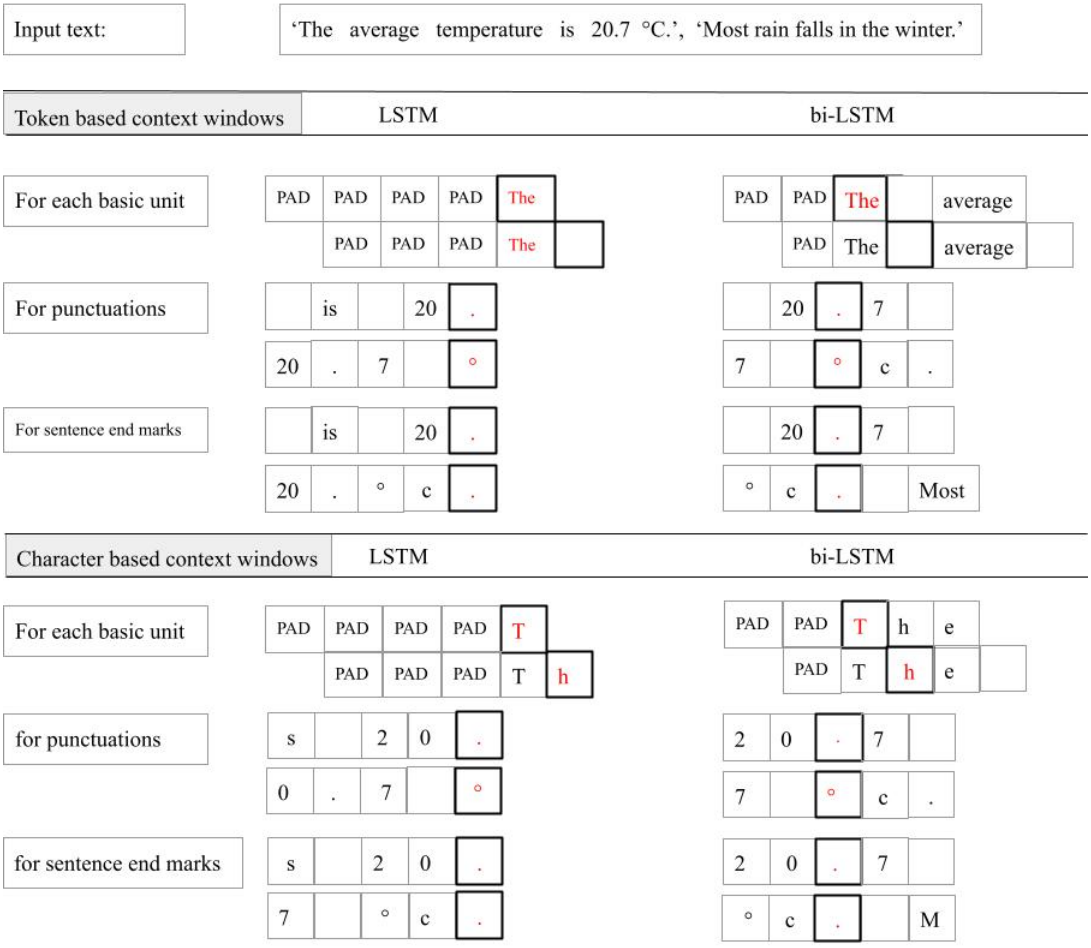


Figure 10: Different versions of context windows. Context window size of 5 created on the two example sentences. The unit we predict for is marked with bold rectangle.

different methods are shown in Figure 10 with the first two context windows for each version. We also use a special unit *DIG* for substituting numbers.

We have found that the size of the context plays a key role. As the size of the context grows, so does the accuracy of the model, but also the training time. To optimize the gain of the larger context, we have chosen the window size where the development loss is low enough with an acceptable training time. So for uni-LSTM, a context window size of 9 was chosen. In order to be consequent we chose for bi-LSTM the same size of context after the current basic unit resulting in a context window of 17.

During the preparation, we create a unit-index mapping from the words/characters of the input text by mapping every unit to an integer. This enables us to encode the units as numbers that will be the input for the embedding layer of our models. The entire data preparation process is demonstrated in Figure 11.

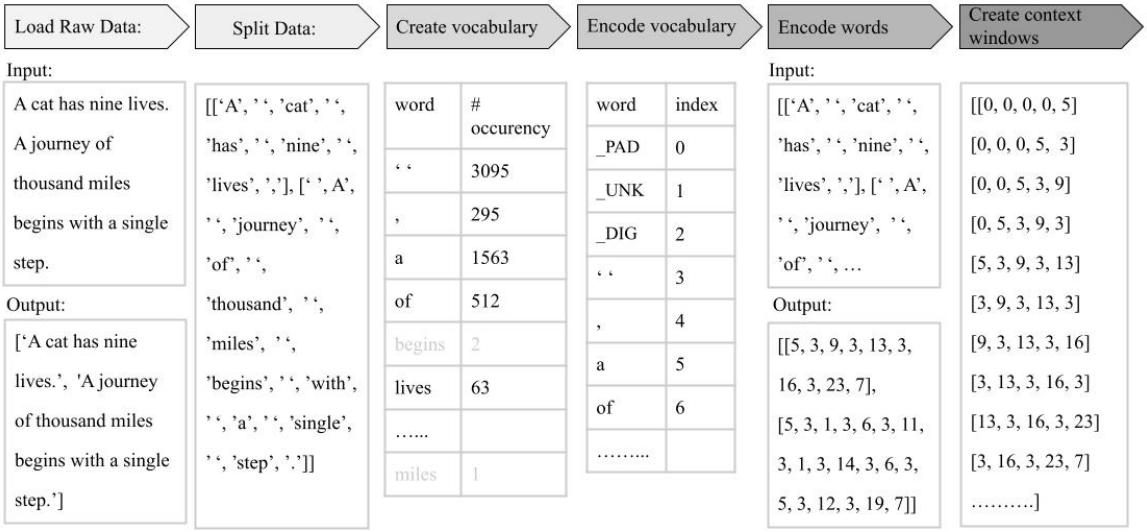


Figure 11: Data preparation: Create context windows from the input text

4.2 Experimental Setup

For our models, we have chosen a context size of 17 for the bidirectional and 9 for the unidirectional LSTM. That means, the unidirectional LSTM calculates on the 8 previous basic units and the bidirectional LSTM calculates on the 8 basic units before and after the current unit. Based on our experiments with hyperparameters, we have used the following setup:

- batch size of 256, which gives the number of context windows in one iteration of the training,
- embeddings size of 256, which defines the length of the feature vector encoding a basic unit,

- number of units 128, which is the dimension of the hidden layer, meaning the LSTM layer includes this number of neurons,
- number of layers 1, that means we do not use additional layer in our model,
- our learning rate was 0.01, which defines the size of the steps in adjusting the weights during the training in order to minimize loss,
- we used the stochastic gradient descent (SGD) optimizer algorithmus,
- we used the cross entropy loss function to measure the correctness of our prediction during the training.

4.3 Early stopping

We trained the models for 10 epochs on larger datasets, 100 epochs on small datasets. The best models were trained with the early stopping method for the final evaluation.

Early stopping is a method to finish the training procedure if there is no more improvement achieved on the development set. After every validation, we also check if the loss on the development set is smaller than the last stored value. We also have a patience parameter that gives the number of epochs we keep going with the training without improvement. We can set a maximum number of epochs, so that the training finishes at the latest after that number of epochs.

4.4 Threshold tuning

After the training, we calculate the F1-score (detailed description in section 6.1) for every threshold between 0.1 and 0.99 inclusive with a step size of 0.1, and we take the threshold with the best F1 score. With this method we can optimize the model on evaluation over the development set.

5 Benchmarks

SBD systems have historically been benchmarked on the Wall Street Journal/Penn Treebank corpora (Read et al., 2012). We have a small set of 3,914 sentences from the NLTK library and used as ground truth for evaluating models trained on other datasets. Besides WSJ we also evaluated on the Brown corpus which contains texts from 500 sources with varied genres. It was created at Brown University from works published in the United States in 1961. The Brown corpus and the WSJ corpus can be extracted from NLTK as a list of words and have to be re-tokenized.

We broaden our evaluation with the Europarl corpus which was also chosen by Wong et al. (2014) and Schweter et al. (2019). Europarl is extracted from the proceedings of the European Parliament. The English version contains more than 1.9 million sentences. As Schweter et. al. (2019) also mentions, the Europarl corpus has errors: some lines contain more than one sentence. For this reason, the corpus has to be cleaned before the training process:

- remove empty lines
- put missing linebreak after each period followed by 'I'
- remove a period from the beginning of a line

To test the generality of our tool except these usual corpora two additional special corpora were used: ArXiv and Wikipedia. The Wikipedia dataset has more than 45 million paragraphs from Wikipedia entries in one paragraph per line data format. The arXiv dataset contains above 1 million paragraphs in the field of scholarly articles from several research domains in the same format as Wikipedia. These two corpora are not cleaned, some lines contain only a part of a sentence. Before we used these corpora we applied some cleaning on them, removing lines containing only one word and also substituting references and citations by a number in brackets.

5.1 Datasets

Europarl, Wikipedia, Brown and arXiv were split into three sets: training, development, and test set. Training set is 80% of the whole dataset, it was used to create training data to train the models. We used subsets of the same datasets to see how different training set sizes affect model performance. The model is trained through supervised learning that uses training example-label pairs, where the label is the desired output. That means that the data containing paragraphs had to be split into sentences to be able to create their labels for the supervised learning. The remaining 20% was split to create a development and test set. The model was evaluated on the development set as we were building it. The test set was used for the final evaluation. Table 1 shows the size of the different corpora we used.

	total number of		sentences in
corpus	paragraphs	sentences	training set
Wall Street Journal	-	3.914	-
Europarl	-	1.906.966	500.000
Brown	-	56.323	36.321
arXiv	1.006.228	-	108.608
Wikipedia	45.676.715	-	3,787,371

Table 1: Size of sentence- and paragraph-based corpora and their training sets

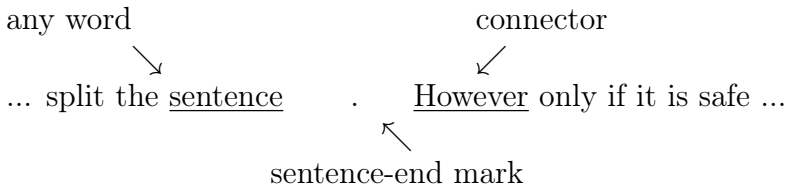
5.1.1 Training data

The training data for the LSTM models is a text containing separate sentences. The last unit of these sentences will be annotated as a sentence boundary during the preparation before the training.

Europarl contains sentences, but Wikipedia and arXiv have only paragraphs built up from one to several sentences. For creating large numbers of sentences, we used the basic assumption that an average English sentence has about 100 characters.

We take all the paragraphs with a maximum length of 100 characters and filter out most of the paragraphs with several sentences. In some cases, the paragraph stll contains more than one sentence. To split such paragraphs, we used a rule described below.

We defined a set of possible sentence end marks: . ? ! “ .) .” and filtered out all the paragraphs that do not have an end mark from this set. We also restricted the possible sentence starters in order to be able to concatenate the sentences in our training. This restriction is that the line should start with an uppercase letter, a number, a starting parenthese (, or a quotation mark “. If there is any punctuation within the line, we want to split it only if there is no doubt about its correctness. For this reason, we use a pattern and a set of common connecting words (however, therefore, because) and stop words (the, this, there, so):



As in Wikipedia the closing quotation mark often comes after the period - common for American English - we also let these combinations in. This method lets some errors through: that is, all the sentences starting with a word not in the common connecting words set or the exceptions where a word from this set follows an abbreviation. The number of errors will be balanced out by the significantly larger number of correct sentences.

In order to use all the sentence end marks in a context, after labelling them we concatenate all the sentences. In this way we can use sentence ends as the previous context for the first units in the next sentence. And the beginning of a sentence can be used as a context that follows for the end units in a previous sentence.

With this method using a maximum length of 100 characters we get from the Wikipedia corpus 3,787,371 sentences. For arXiv we use a maximum length of 200 characters to get a sufficient amount of sentences. In this way we get about 108,608

sentences, as it is shown in Table 1 in the previous section.

5.1.2 Development set

The goal of using separate data during the development is that we can check the output of the model during the training. A set of sentences extracted from the development set is used to test the model after a certain number of steps on this validation data. We calculate how much the model's output differs from our target and if it is better than the previously tested model, we save it as the best model.

5.1.3 Test set

To be able to measure and compare the performance of the models, a ground truth was created for a small subset of the test set for all the datasets. This ground truth contains the individual sentences from each of the example paragraphs. These sentences are the targets for our model.

	hard cases		random	
corpus	paragraphs	sentences	paragraphs	sentences
Wall Street Journal			571	3.914
Europarl			1043	6.876
Brown			1.409	9995
arXiv	1.000	4.091		
Wikipedia	775	2.928	929	3.267

Table 2: Size of the test subset with random sentences and hard cases

In the Wikipedia and arXiv datasets, we selected the paragraphs where the models of the two python libraries, NLTK and spaCy struggled, producing errors. Such hard cases were selected by segmenting the test set both with the default model of NLTK and spaCy, and taking the paragraphs where they gave different results. We

had to correct this segmentation manually. The evaluation results are influenced by the difficult cases in the ground truth, possibly giving lower values than on randomly selected text. That is why we also created a ground truth for random paragraphs. In the case of the Europarl and Brown datasets, we created the paragraphs using their sentences. A random integer between 4 and 15 gave the number of sentences for each created paragraph.

Table 2 shows the size of the created test subsets for evaluation. The evaluation performed on these datasets is presented in the next section.

6 Experiments

In this chapter, we first present the metrics we used in our analysis. Then follow all the experiments that we did in order to find the best solution to the sentence segmentation problem. We compared the models trained on training data with sentences of different maximum lengths. The results are listed in Table 4. Table 5 shows the comparison of the models trained on differently sized training data. Table 6 gives an overview of the models with various context window types. In Table 7 we compare the memory usage of the different context window approaches. That is followed by a comparison of an LSTM and bi-LSTM model using differently sized context windows in Table 8. The evaluation on ground truth with different levels of difficulty is in Table 9. The comparison of our best LSTM models with NLTK and spaCy is shown in Table 10. In Table 11 we test our models on different corpora.

6.1 Metrics

Several metrics are used to determine how well the model performs, focusing on different aspects. In the case of two-class classification, a confusion matrix is used to evaluate the classification models, comparing the values predicted by the model with the actual values. It combines true and false negatives and positives. In our model, that means:

- true positive (TP) is a correctly predicted sentence boundary
- false positive (FP) is a false predicted sentence boundary
- false negative (FN) is a missed sentence boundary
- and true negative (TN) are all the not sentence boundary cases that were also not predicted as sentence boundaries.

		Predicted	
		Positive	Negative
Actual	Positive	True positive	False Negative
	Negative	False Positive	True Negative

Table 3: Confusion matrix for two-class classification

These values in the confusion matrix are the basis for a set of metrics. These metrics are sensitive to the number of samples in a segment. Our training data set is imbalanced. That means we have much more occurrences in the non-sentence-boundary class than in the other. That causes distortion in the measures, where true negative cases are used in the calculation. Besides, the different methods used for creating context windows return different number of negative cases, which further sharpens this imbalance. For this reason we calculate these metrics on our ground truth in the following way. We segmentize the paragraphs in the ground truth. The spans returned by our model are then the basis of our evaluation. We compare the predicted spans with the expected ones. We compare for each span the beginning and the end.

- we compare the start position in the predicted and the expected spans:
  - if the start position in the prediction span is smaller than the expected one, that counts as a False Positive
  - if the predicted start position is greater than the expected one, that counts as a False Negative
  - if the start positions in the prediction and the expected span match, that counts as a True Positive
- after that, we compare the end position of the predicted and expected span:
  - if the end value of the predicted span is smaller than the expected end position, that counts as a False Positive
  - if the end value in the predicted span is greater than the expected end position, that counts as a False Negative

– if the positions in the prediction and the expected span are equal, that counts as True Positive.

After comparing the end positions in the span, we move on with the span containing a smaller end position. We keep doing that as long as we achieve the last span in both the prediction and the expected segmentation. See the next example:

	paragraph	spans
predicted	He was the president Mr.	[0, 10]
	Kennedy.	[11, 13]
expected	He was the president Mr. Kennedy.	[0, 13]

predicted	expected	resulted metrics
[0, 10]	[0, 13]	TP for 0 = 0, FP for 10 <13
[11, 13]	[0, 13]	FP for 11 >0, TP for 13 = 13

In this way, we count every error twice. We count the beginning and the end of the sentence as well. That gives us two counts for a sentence boundary. That means for a correct sentence where both the beginning and the end match, we get two true positive cases. As that holds for each span, it does not change the proportions.

The most frequently used measure is the accuracy, which can tell us how many predictions the classification model got right. Accuracy shows the overall performance, predicting all the classes. [21].

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

However, if we have imbalanced data, accuracy can be misleading by giving a high rate despite numerous errors. Therefore, we define the accuracy on our ground truth by calculating the rate of correctly returned sentences and we use this form in our evaluation.

$$Accuracy = \frac{Correct\ Sentences}{All\ Sentences}$$

We can also calculate the proportion of false negative and false positive predictions within all the predictions. It gives us the other most commonly used metric, the error rate. However, it is calculated with the true negative predictions, which can cause a very low value in the case of imbalanced data. We modify the error rate and calculate it without the true negative predictions:

$$\text{Error rate} = \frac{FP + FN}{TP + FP + FN}$$

The other way we can remedy the imbalance problem is to get the error rate as the complement of the accuracy:

$$Error\ rate = 1 - Accuracy$$

It will give us only the rate of the false sentences and no detailed information about the model. For this reason we use the first version for calculating the error rates.
To have a better understanding of error types, we calculate further metrics based on the values in the confusion matrix: precision and recall. These focus on positive predictions. Precision concentrates on the accuracy of the positive predictions within all the positive predictions of the model, That means how correct are the predicted sentence boundaries.

$$Precision = \frac{TP}{TP + FP}$$

Recall shows the model's ability to return all the positive values from the expected values, meaning how many sentence boundaries the model is able to find. High recall shows that most of the sentence boundaries are labelled correctly.

$$Recall = \frac{TP}{TP + FN}$$

We can combine these two metrics into one, that is the F-score, the harmonic mean of precision and recall. Within the F-score we can give an additional weight β to give less importance to precision. We used the F1-score with 1, as we consider both metrics equally important.

$$F_\beta\text{-score} = \frac{(1+\beta^2)TP}{(1+\beta^2)TP + \beta^2 FN + FP}$$

$$\text{F1-score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

6.2 Baseline systems

To measure the performance of our system we compared our models to the following baseline algorithms.

6.2.1 Our Baseline algorithm

We compare our models with a simple baseline algorithm that splits a string on every match on the following pattern:
punctuation such as closing paranthese, period, quotation mark, exclamation mark and question mark, followed by whitespace and an uppercase letter as the following example shows.

. Here...

By evaluating the result of this algorithm, we define a lower bound, a minimum score that every algorithm should be able to match or outperform.

6.2.2 Baseline tools

We compared our model to two NLP toolkits, NLTK and spaCy. From NLTK, we used the Punkt sentence tokenizer, the default segmentizer, which we also trained on each corpus and customized with a list of abbreviations. The spaCy's Parser requires training data that is annotated with dependencies. As this was not available and to create it would be effort and time consuming, we used the default Parser for sentence segmentation. We trained the Senter from spaCy also on each corpus and customized it with the abbreviation list gained from training the NLTK segmenter.

6.3 Results

In this section, we give an overview of the experiments we conducted on all the aspects that have an impact on our models.

Sentence length We applied a simple algorithm described in Section 5.1.1 to gain our training sentences from the Wikipedia and arXiv corpora by setting a maximum length of a paragraph.

Different maximum lengths gave us different training sets. By setting this length to a larger value, we got a larger number of sentences, however with more possible errors. We tested how the various sentence lengths influenced the model's accuracy. We took a subset of 500,000 sentences from the Wikipedia training set with a maximum length of 100, 150, and 200 characters per sentence. The results of a token based model in Table 4 show that the longer sentences resulted in a higher recall but a lower precision. The maximum length of 150 characters resulted in a larger drop in precision that could not be balanced out by the higher recall. That was the case especially with the uni-LSTM model, that returned 4.36% higher error rate on 150 character maximum length. The bi-LSTM model with a maximum of 200 characters, on the other hand, had a higher accuracy due to its higher recall.

maximum characters in a sentence	Error rate	Precision	Recall	F1-score	Accuracy
bi-LSTM model(17)					
100	0.0796	0.9507	0.9666	0.9586	89.79%
150	0.1058	0.9111	0.9797	0.9442	89.50%
200	0.0882	0.9327	0.9760	0.9539	90.77%
uni-LSTM model(9)					
100	0.1782	0.8591	0.9499	0.9022	80.02%
150	0.2246	0.7933	0.9717	0.8735	79.44%
200	0.2153	0.8031	0.9717	0.8794	79.84%

Table 4: Model evaluation on different sentence length
evaluated on the hard cases of the Wikipedia development set.
Model description: token based models trained on 500,000 sentences from Wikipedia corpus, With context window created for the whole training set. Size of the context window is in brackets next to the model type.

Size of the training data We trained our models on differently sized training data from the Wikipedia corpus, to test the influence of the training data size on the accuracy of the models. The results in Table 5 confirm that a larger amount of training data can improve the models' accuracy. The scale of the improvement is notable as we largen our data from 7,000 to 500,000 sentences. Having even more sentences does not necessarily grant further improvement, at least not on a large scale. The same effect was observed in NLTK, where the training on the whole Wikipedia data with 44,230,320 paragraphs instead of 1,000,000 did not improve the accuracy of the segmentizer: it slightly dropped from 87.55% to 86.99%. However, it also must be noted, that our method for generating the training data leaves errors in the training data. That can be a hold back for the further improvement by training on a larger dataset.

Training sentences	token based full text	token based punct.	token based end marks	character based full text	character based punct.	character based end marks
			LSTM(9)			
7,000	75.41	75.52	75.48	78.59	77.52	78.50
500,000	80.02	78.28	79.30	81.11	81.04	81.18
3,780,371	**80.28**	**82.67**	**79.99**	**81.40**	**82.49**	**82.02**
			bi-LSTM(17)			
7,000	88.59	87.29	87.47	85.47	87.90	87.40
500,000	89.83	90.45	91.25	**91.90**	**91.97**	91.68
3,780,371	**89.90**	**91.32**	**91.54**	91.54	90.74	**91.97**

Table 5: Accuracy of different models trained on different size training data Rate of correctly returned sentences of unidirectional and bidirectional LSTM models, context window created with various methods. Trained on sentences from the Wikipedia corpus, evaluated on the development subset of 2753 sentences with hard cases.

Type of the context window There are two main approaches for creating the input for our models: the token- and character-based approaches. For these approaches we can create a context window in 3 different ways:

- full text method: context window created for every basic unit in the whole input text
- method on punctuations: a context window created for every punctuation in the input text
- method only on possible sentence end marks: context window created for the punctuations in the predefined set of possible sentence ending punctuations: . ? !)] " ’

Table 6 gives a comparison of the models trained on 500,000 Wikipedia sentences for 10 epochs with different methods for creating the context window.

We find that the lowest error rates are returned by the following two bi-LSTM models: the character-based context window for each unit and for every punctuation mark. These two models return all metric values with a small difference. Their

bi-LSTM model(17)	Error rate	Precision	Recall	F1-score	Accuracy
Token based whole	0.0787	0.9511	0.9671	0.9590	89.83%
Token based on punctuation	0.0754	**0.9528**	0.9689	0.9608	90.45%
Token based on end marks	0.0740	0.9481	0.9755	0.9616	91.25%
Char. based whole	**0.0720**	0.9501	0.9755	0.9526	91.90%
Character based on punctuation	0.0733	0.9478	0.9766	**0.9620**	**91.97%**
Character based on end marks	0.0787	0.9388	**0.9802**	0.9590	91.68%
uni-LSTM model(9)	Error rate	Precision	Recall	F1-score	Accuracy
Token based whole	**0.1780**	**0.8591**	0.9501	**0.9023**	80.02%
Token based on punctuation	0.1906	0.8509	0.9432	0.8947	78.28%
Token based on end marks	0.2192	0.7994	**0.9711**	0.8769	79.30%
Character based	0.1957	0.8283	0.9653	0.8916	81.11%
Character based on punctuation	0.1899	0.8333	0.9628	0.8951	81.04%
Character based on end marks	0.1904	0.8333	0.9660	0.8948	**81.18%**

Table 6: Model evaluation on context window types
Evaluated on the hard cases from the Wikipedia development set. *Model description*: Models trained for 10 epochs on 500,000 sentences from the Wikipedia corpus, context window size 9 for LSTM and 17 for bi-LSTM.

error rates are the lowest, with 0.0720 and 0.0733 respectively. They also have the highest accuracy, with 91.90% and 91.97%, and the highest F1-score, with 0.9526 and 0.9620. An improvement from 89.83% to 91.90% in accuracy between the token- and character-based models means another 28 correctly segmentized sentences out of 2752 sentences in our ground truth.

The third model with high accuracy is the model with the context window on sentence end marks. It has the highest recall, but at the same time, the lowest precision. That means that we can identify the highest proportion of actual sentence boundaries correctly with this method. Although the precision is lower than by the other methods, which means a higher number of false positive predictions. That results in the second lowest F1-score (0.9590) under the bi-LSTM models. It can be caused by the reduced set of context windows, which results in smaller training data. Training with more epochs or more training data could improve this measure.

We find the largest difference in precision between character- and token-based versions of the uni-LSTM models with context window for each unit. Here the precision drops by 0.031 from 0.8591 for the token-based model to 0.8283 by the character-based model. Just the opposite can be observed by the uni-LSTM models with the context window on sentence end marks. Here, the character-based model improves the precision by 0.034 in comparison to the token-based version.

We observe the best results in recall and accuracy by the models concerning only the possible sentence end marks. The explanation for this can be that training data is more dense and balanced. That results in a faster learning process and eliminates unnecessary negative cases. That also reduces the training time: for token-based models about 4-8 times, for character-based models about 15-30 times.
Overall, the bi-LSTM character-based models stand out as the best models, having only a small difference in accuracy.

Memory usage Each method produces a differently sized input for our models. The size of this input data is the largest if we create a context window for every character, although most of them have a negative label. We improve the proportion of positive-labelled data if we create a context window only for punctuation. In this way, we optimize the data preparation time and the training time, but also the memory usage, because the training data sample will be smaller, as shown in Table 7.

Number of sentences	Context window created for every character		token		punctuation	
	Memory	Window	Memory	Window	Memory	Window
7,000	28 MB	790k	10 MB	282k	1 MB	25k
500,000	1,25 GB	34,952k	472 MB	13,133k	48 MB	1,334k
3,780,371	9 GB	254,460k	3 GB	95,308k	336 MB	9,338k

Table 7: Memory usage and training data size by different methods on creating a context window. Training data is from the Wikipedia corpus.

Context size We ran an experiment to evaluate how the size of the context impacts the performance of the model. Table 8 shows the F1-score and the sentence accuracy of the character-based models with a context window created only for punctuation. The models were trained on 500,000 Wikipedia sentences.

Context window size	LSTM			bi-LSTM		
	F1-score	Acc %	Time diff.	F1-score	Acc %	Time diff.
5	0.892	80.4	-	0.949	90.2	-
7	0.887	80.6	+4.3%	0.956	91.1	+9.1%
9	**0.892**	**80.9**	+11.9%	0.957	91.5	+10.4%
11	0.882	79.6	+13.2%	0.959	91.9	+17.4%
13	0.883	79.3	+12.3%	0.959	91.8	+23.9%
15	0.884	79.4	+13.8%	**0.962**	**92.1**	+38.9%
17	0.883	79.2	+14.7%	0.960	91.8	+45.7%
19	0.884	79.2	+28.2%	0.961	91.9	+52.0%
21	0.885	79.2	+22.3%	0.960	91.9	+60.0%
23	0.886	79.4	+29.9%	0.960	91.8	+70.4%
25	0.895	80.1	+24.1%	0.960	91.8	+81.0%

Table 8: Experiment with the context window size
Growth of training time compared to the model with the shortest context window. Evaluated on a ground truth with hard cases from the development set. The best result is marked bold.
Model description: character-based models with a context window created only for punctuation. Trained for 10 epochs on 500,000 Wikipedia sentences.

The larger context results in higher F1 and accuracy values by both LSTM models. But we can also state that only until a certain level, above which the larger context does not improve the model's performance. This optimal context size for the uni-LSTM model is 9, meaning that we have 8 basic units before the actual inspected unit. By bi-LSTM this optimum is at 15, meaning 7 basic units before and after the actual inspected unit. It is interesting to note, that the uni-LSTM model's accuracy decreases with larger context than the optimum. In contrary to that, the bi-LSTM model's accuracy stays approximately at the same level.

Ground truth Our ground truth contains the hard cases based on the output of NLTK and spaCy segmentizers. To be able to fairly evaluate the models, we created another ground truth from randomly selected paragraphs. The results for the three best bi-LSTM models and one LSTM model are presented in Table 9.

Because their accuracy values are similar, we chose all three character-based models from the bi-LSTM models as the best models. For the best uni-LSTM model, we chose a character-based model with a context window on sentence end marks, trained on the largest Wikipedia training set with 3,787,371 sentences. There is a large improvement in all metrics when evaluating them on a set of randomly sampled paragraphs compared to the results on the hard cases. The most outstanding difference is shown by the LSTM model, which improves from 81.90% to 93.88% in accuracy. That proves our assumption that the performance of the sentence boundary detection systems highly depends on the uncommon punctuation usage within the corpus.

Context window	error rate	precision	recall	F1	Acc
Test set with hard cases (2928 sentences)					
bi-LSTM(17)					
for each basic unit	0.0737	0.9534	0.9703	0.9617	90.95%
for punctuations	0.0880	0.9349	0.9738	0.9540	90.37%
for sentence end marks	**0.0699**	**0.9537**	**0.9741**	**0.9638**	**91.36%**
LSTM(9)					
for sentence end marks	0.1707	0.8497	0.9719	0.9067	81.90%
Test set with random cases (3267 sentences)					
bi-LSTM(17)					
for each basic unit	0.0182	0.9867	0.9950	0.9908	98.03%
for punctuations	0.0222	0.9830	0.9947	0.9888	97.68%
for sentence end marks	**0.0145**	**0.9882**	**0.9972**	**0.9927**	**98.28%**
LSTM(9)					
for sentence end marks	0.0511	0.9561	0.9878	0.9717	93.88%

Table 9: The best models evaluated on random and hard cases. Best character-based models evaluated on two types of test set: one with random paragraphs and one with hard cases.
Model description: The bi-LSTM models are trained on 500,000 Wikipedia sentences for 10 epochs, the LSTM model on 3,787,371 sentences for 10 epochs.

Comparison to other systems Table 10 shows the results of the evaluation of the best unidirectional and bidirectional model, the baseline algorithm described in Section 6.2 and the segmentizers of the two Python libraries NLTK and spaCy. We observe that the information about the units following the actual basic unit improved mostly the precision of the bi-LSTM model, as the recall of the two models are nearly the same. The LSTM's low precision shows the large number of false positive predictions.

This bi-LSTM model outperforms not only the LSTM model, but also all the other approaches. The second best accuracy comes from the statistical approach of the NLTK. Our bi-LSTM model's accuracy is higher because of its higher precision,

Type	Error rate	Precision	Recall	F1-score	Accuracy
Baseline	0.16	0.88	0.93	0.91	80.12%
LSTM(9)	0.19	0.82	0.97	0.89	80.00%
Bi-LSTM(15)	**0.07**	**0.95**	**0.97**	**0.96**	**91.39%**
NLTK default	0.15	0.87	0.98	0.92	84.87%
NLTK trained, customized	0.09	0.94	0.97	0.95	89.07%
spaCy parser default	0.23	0.81	0.94	0.87	73.53%
spaCy senter default	0.16	0.90	0.92	0.91	80.84%
spaCy senter trained	0.18	0.88	0.92	0.90	79.44%

Table 10: Comparison of different sentence segmentizer systems with our best uni- and bidirectional LSTM models. Evaluated on the Wikipedia test ground truth with hard cases.
Model description: Character-based models, context for sentence end punctuations, trained on 500,000 Wikipedia sentences with early stopping method. Size of the context window is in parentheses next to the name of the model.

which means that our model returns less false positives. The parser from spaCy that uses dependencies in a sentence has the lowest values, and performs only a bit better than the baseline algorithm. Its high error rate can be explained by the assumption that the abbreviations and quotes in Wikipedia break the relations within a dependency tree of a sentence. That explains the better values of the spaCy senter, which uses Convolutional Neural Network without the dependencies.

Domain types To test our best model's robustness against different genres, we train and evaluate it on corpora from various domains. The complexity and irregularity of the corpus clearly affect the accuracy of the model. Table 11 shows the results of two versions of our best bi-LSTM character based model. We find that both of our systems have the best accuracy on Europarl with only a small difference. On the other hand, on the Brown paragraphs, the bi-LSTM model with the context window for sentence end marks has a lot lower accuracy. Its recall value is lower too, which implies missed sentence boundaries. As this model has a context

window only for the predefined set of sentence end marks, which does not contain the semicolon, it never splits on this punctuation. However, in the ground truth of the Brown corpus, sentences often end with a semicolon. That causes a lot of false negatives, which decreases the model's accuracy.

Corpus	Error rate	Precision	Recall	F1-score	Accuracy
Context window on punctuations					
arXiv hard	0.05	0.98	0.97	0.97	92.91%
Europarl	0.02	0.99	0.98	0.99	96.80%
Wikipedia hard	0.08	0.94	0.97	0.96	90.64%
Brown	0.03	0.99	0.97	0.98	93.94%
Context window on sentence end marks					
arXiv hard	0.04	0.98	0.97	0.97	93.94%
Europarl	0.02	0.99	0.98	0.99	96.57%
Wikipedia hard	0.07	0.95	0.97	0.96	91.39%
Brown	0.07	0.99	0.93	0.96	86.31%

Table 11: Character based bi-LSTM(15) model evaluated on different corpora. In case of Wikipedia and arXiv the model was evaluated on the ground truth with hard cases from the test set.
Model description :Character-based bi-LSTM models trained with the early stopping method on the corresponding corpus.

According to Mikheev [4] in the Brown Corpus 92% of the periods come after a regular word (nonabbreviation). In the Wall Street Journal corpus 83% of sentences end with a regular word followed by a period. These are the simple cases for any sentence boundary detection system. The baseline algorithm can find these simple cases. However, as the results in Table 12 show, on these corpora had our baseline algorithm high error rates.

That is because these corpora contain titles without punctuation which our algorithm would not split. That resulted in many false negatives. Besides, the abbreviations of names like *Senator J. W. Fulbright* caused numerous false positives.

Corpus	Error rate	Precision	Recall	F1-score	Accuracy
arXiv	0.08	0.94	0.97	0.96	92.30%
Europarl	0.07	0.97	0.95	0.96	90.98%
Wikipedia	0.16	0.88	0.93	0.91	80.12%
Brown	0.21	0.94	0.83	0.88	67.43%
WSJ	0.24	0.83	0.89	0.86	69.78%

Table 12: Baseline algorithm evaluated on different corpora

Our baseline algorithm had the highest precision on the Europarl corpus. This can be explained by the small number of abbreviations in the corpus, so the simple pattern of *'arbitrary character - punctuation - whitespace - capitalized common start word'* found only the real sentence boundaries. The accuracy is however only the second highest, because of the cases with the before mentioned titles without punctuation. The highest recall was returned on the arXiv corpus. The large number of abbreviations and formels with variable names caused the low precision.

We compare the different systems' accuracy on these corpora in Table 13. On Europarl and on Wikipedia with random paragraphs all the systems return a good accuracy. However, on the other corpora the results are more diverse. NLTK has the lowest accuracy on Brown ground truth, due to the missing punctuation in titles. It is interesting to observe that no other system than bi-LSTM could outperform the baseline algorithm on arXiv. The trained NLTK made a lot of false positives on periods following a variable name, because the single letters are included in its abbreviation list. The default NLTK and the Senter from spaCy had many false positives because of rare abbreviations (*comp., publ.)*, besides the Senter missed to split after some words (e.g. furthermore).

On Wikipedia hard cases, spaCy's Senter has an accuracy around 10 % lower than the accuracy of NLTK or the bi-LSTM model. It is because of the quotation mark after the period that Senter joins to the next sentence.

On the random test cases of Wikipedia, all the systems have high accuracy, but

Corpus	Baseline	LSTM(9)	bi-LSTM(15)	default NLTK	trained NLTK	senter spaCy
Europarl	90.98	95.72	**96.80**	95.39	95.68	95.10
arXiv hard	92.30	90.37	**92.91**	74.31	87.29	91.93
Brown	67.43	92.17	**93.94**	85.01	83.91	93.84
Wikipedia hard	80.12	82.04	**90.64**	84.19	89.07	79.44
Wikipedia rand.	91.86	93.44	96.94	**99.42**	97.86	95.56
WSJ	69.79	72.75	89.29	**92.79**	91.74	80.32

Table 13: Accuracy of the different systems.
The systems were trained and evaluated on the corresponding corpus, except in the evaluation on WSJ, where we used the models trained on the Wikipedia training data. NLTK's PunktTokenizer and the spaCy's Senter are trained on the corresponding corpus, and customized with abbreviation list.
Model description: our best uni- and bidirectional LSTM models are character-based, the context window were created on punctuation, and the models were trained with early stopping.

the NLTK's default segmentizer had the highest value. For the evaluation on WSJ the systems were trained on the Wikipedia training data. On WSJ, our bi-LSTM model was the second best after the NLTK segmentizers. Most of the errors by the bi-LSTM model were caused by the unknown abbreviations.

The evaluation on WSJ showed that for all the systems, it is necessary to train on a training data similar to the target data. If we compare the results on the random Wikipedia paragraphs with those on the WSJ paragraphs, we find that all systems' accuracy dropped. The uni-LSTM model's accuracy reduced from 93.44% to 72.75%. Under all the systems, the default NLTK was not trained on our Wikipedia training data. It had the best performance on WSJ with 92.79% accuracy. This is because the default NLTK Punkt sentence tokenizer is pretrained for general use, that works for the WSJ ground truth and also for the random Wikipedia. It is important to note that our trained NLTK segmentizer only with customization returns 91.74% of the sentences correctly on WSJ, without our adjustment its performance is only 75.77%

It is also interesting to compare the two LSTM models. However the bi-LSTM model performed better on all the corpora, we find the largest difference on the Wikipedia paragraphs with hard cases and on the WSJ paragraphs.

6.3.1 Error types

There are two different types of errors: false negative and false positive predictions. False negative predictions are the missed sentence boundaries. The false positive predictions are the incorrect splits made by the model. We study these errors on our character-based bi-LSTM model with a context window size of 15, trained with the early stopping method.

We have found incorrect splits after some abbreviations. After the abbreviation *Dr.* the model split the sentence wrongly in 3 out of 13 cases. The same was with the abbreviation *Rev.*. Here the model failed 5 out of 8 times.

The model also failed on the rare sentence end marks like the closing parantheses. It was missed by the model, as in the examples below (we mars the split made by the model with || in the following):

|| *... translated as "Qīng Chéng" () The name...* ||
|| *... has also been Paramount 4000) Gone was the black...* ||

The reason is that it is more frequently used within a sentence in the training data. The model learns this and gives a low probability for a parantheses being a sentence boundary.

As the ellipsis is often used within a sentence but also as a sentence end, the model failed in some of these cases and did not split:

|| *breathe air ... People only want* ||
|| *at all hazards. ... Give the enemy no rest ...* ||

The aposthrope caused the large number of errors, both false positive and false negative. The reason is that in Wikipedia paragraphs the apostrophe stands often after the sentence ending period.

- false negative:

  || *future is remote.” In recent years* ||

  || "*the sexiest man on TV.*" *As Eddie films* ||

- false positive

  "chenel" || *"canal".*

  "tucking" || *the ball*

The spaCy's senter usually joined this last apostrophe to the next sentence.

bi-LSTM:

"It was as if he didn't want the guests to be there." || *Cleese and Connie Booth stayed on at the hotel after filming*

spaCy:

"It was as if he didn't want the guests to be there. || *" Cleese and Connie Booth stayed on at the hotel after filming*

It is common in Wikipedia that there is a proper name with punctuation within the sentence, like *Wham!*. The character based model did split on these cases because of the sentence end mark:

"Do You Miss Me?" || *became a Top 40 hit*
Soul group The Trammps recorded "Zing! || *Went the Strings of My Heart" in 1972.*

However for the more common cases it correctly ignored the sentence end mark. In the following example the sentence was split on *Splash!* but not on *Yahoo!*:

including Yahoo! Wireless in London, Splash! || *in Germany, OpenAir Frauenfeld in Switzerland,*

It is interesting to note that these errors were correctly classified by the token based model.

The model split on some rare initials, however for some other more common pattern the prediction was correct:

correct: *Karl H. Bergey*
incorrect split: *G. I.* || *in the Vietnam*

The same happened with the abbreviation *St.* for *Saint*. For most of the cases it was correctly predicted: *St Louis*, *St. Mary* was not split, however on the period within *St. John* the sentence was split.

The model correctly learned that after *U.S.* or *Washington D.C.* the sentence goes on. But that lead to errors within some other cases. Here the *U.S.* and *D.C.* stands at the end of the sentence:

popular in the U.S. Daddy Yankee then
in Washington D.C. Before and

These errors were also observed with the segmenters from spaCy's and NLTK.

The model with context window on punctuations split on the semicolon that we did not consider as a sentence end in our ground truth. This error was eliminated in the model which had a context window only for the predefined set of possible sentence end marks. This set did not include the semicolon so the model did not have to predict for it.

Unidirectional vs. bidirectional model When we compare the output of unidirectional and bidirectional LSTM, we notice the following difference. The unidirectional LSTM model split a sentence wrongly if there were several punctuation marks next to each other. As it is very common in the Wikipedia corpus that a quotation mark stands after the period at the sentence end, this pattern caused

the most errors, similarly to spaCy:

> *"It was as if he didn't want the guests to be there.* || *"*
>
> *It gave children financial advice for budgeting their allowances and saving for a big purchase, reviewed kid-oriented consumer products (e.g., toys, clothes, electronics, food, videogames, etc.)* ||
> *, and generally promoted smart consumerism in kids and teens;*

This mistake can also be observed by the ellipsis:

> *Give the enemy no rest .* ||
> *.. Do all the damage to railroads and crops you can.*

The uni-LSTM model missed also a sentence boundary if it stand after a number and did not split.

> *Although standard PC video cards had a flag to indicate that the vertical retrace was in progress, the PCjr added for the first time the ability to generate raster or VBLANK interrupts on IRQ* ***5. This*** *allows the use of mixed video modes.*

In these examples listed above, the context after the unit was useful in the prediction.

In contrast to the bidirectional model, the unidirectional model did not split on semicolons in lists.

> *Alfonso Letelier; Gustavo Becerra-Schmidt; Sergio Ortega; Leon Schidlowsky; Leni Alexander; Fernando García;*

6.4 Runtime

Table 14 displays the computational times required to run each system on a 64-bit machine with 16 GB of memory, 4-core AMD Ryzen 5 2100 Mhz Processor, and a GeForce GTX 1660 Ti GPU. We ran the systems on the Wikipedia corpus' ground truth with 2928 sentences.

model	runtime (in sec)
baseline	19.56
default NLTK	21.25
trained NLTK	20.41
customized and trained NLTK	19.50
spaCy's default Parser	32.50
spaCy's default Senter	28.20
spaCy's trained Senter	22.76
uni-LSTM token full text	190.56
uni-LSTM token on punctuation	41.62
uni-LSTM token on endpunctuation	36.33
uni-LSTM character full text	496.73
uni-LSTM character on punctuation	46.76
uni-LSTM character on endpunctuation	36.31
bi-LSTM token full text	350.94
bi-LSTM token on punctuation	56.61
bi-LSTM token on endpunctuation	42.17
bi-LSTM character full text	891.66
bi-LSTM character on punctuation	56.34
bi-LSTM character on endpunctuation	42.47

Table 14: Evaluation runtime of different systems on Wikipedia ground truth with 2928 sentences

The results show that the NLTK tokenizers are the fastest of all systems. SpaCy's trained Senter has only a slightly longer run time than NLTK.

Regarding our models, those using a context window on every basic unit have

an extremely long running time compared to other versions, but also to other systems. Our best model, the character-based bi-LSTM with the context window on punctuations, has a running time of 56.34s, which is two times larger than the spaCy's default Senter segmentizer. Our models use the GPU for calculations, which lowers the running time. Using a context window only on possible sentence end marks reduces the running time. Further optimization is possible through calculating only till the middle basic unit and not for the whole context window.

7 Conclusion

Our aim in this thesis was to present a solution for the non-trivial task of Sentence Boundary Detection for uncommon corpora like Wikipedia and arXiv. We proposed a model based on machine learning. The applied LSTM networks with the capability of solving the long term dependency problem proved their effectiveness in our experiments. Our best model beats the NLTK ans spaCy segmentizers on four out of five benchmarks.

Size of the data To demonstrate the importance of the training data size, we required a large number of labelled sentences. Correctly segmented data is mostly not provided or the size of the data is not sufficient. We solved this common problem by keeping only the paragraphs shorter than 100 characters and applying on them a splitting method with regular expression to produce a large amount of labelled training data. Despite the small rate of incorrect negative labels within our data, we observed a significant improvement by all models by training on larger datasets.

Context We have shown that the bidirectional LSTM models perform better, compared to the unidirectional version. The usage of the context after the actual token or character improves the model's precision, resulting in higher values both in F-score and accuracy.

We also experimented with different context types. Our results show that using the entire document to create context windows results in high accuracy, but at the expense of extremely long preparation and training time. It is sufficient to create them solely on punctuation or possible sentence ends. It is important to note that this effects the size of the training data, which must be considered in the training.

We tested the models at the character and token levels. Through the experiments we find that the character-based context improve the recall of the models while

reducing the error rate.

We dealt with the problem of imbalanced data. In our method for creating the context window, by restricting the basic units to a set of punctuation, we managed to decrease the number of true negative cases. In our evaluation, we used such metrics that we avoided an excessive number of true negative cases.

Complexity The evaluation of our model and other systems on different corpora confirms that sentence segmentation is not equally difficult in all genres and documents. Our results showed a large variance in the model's performance depending on the domain. Using random and hard ground truth showed results with a large difference.

This indicates a need to develop a generalized evaluation for these systems to be able to compare them.

8 Future work

The possible direction of future work on the sentence segmentation models can be the following:

The usage of pre-trained embeddings like word2vec can be considered by future development. It could further improve the models' performance.

Our model is trained on certain domains and does not generalize to domains different from the domain of the training data. That means our model has to be retrained for a new domain or a new language. Interesting topic for future work is the extension of our model to train it incrementally and merge the new information from the new training data into the existing model. This could provide a general purpose model.

Our model's results highly depend on a threshold. With higher threshold we can eliminate one type of error, but let other type of error through. The method for finetuning of the threshold therefore is important and need further research.

For future work it is desirable to improve the runtime of the system. By the bidirectional LSTM model we can optimize the workflow by calculating forward for the units before the candidate and calculate backward for the units afterwards instead of calculating for the whole context window forwards and backwards in the whole length.

In addition, it could be interesting to investigate how the errors of the sentence segmentation influence the results of the other tasks in the NLP pipeline.

9 Acknowledgments

First and foremost, I would like to thank...

- Matthias Hertel for his support, suggestions, and corrections during the whole work
- my family for their support
- my husband for his support as proofreader

List of Figures

List of Tables

Bibliography

[1] N. Sadvilkar and M. Neumann, "PySBD: Pragmatic sentence boundary disambiguation," in *Proceedings of Second Workshop for NLP Open Source Software (NLP-OSS)*, (Online), pp. 110–114, Association for Computational Linguistics, Nov. 2020.

[2] D. F. Wong, L. S. Chao, and X. Zeng, "isentenizer-μ: Multilingual sentence boundary detection model," *The Scientific World Journal*, vol. 2014, 2014.

[3] "Punctuation in different languages different punctuation." `https://www.translatemedia.com/translation-blog/punctuation-in-different-languages/`. Accessed: 2021-09-30.

[4] A. Mikheev, "Periods, capitalized words, etc.," *Computational Linguistics*, vol. 28, pp. 289–318, 2002.

[5] H. M. Robert Dale, H. L. Somers, *Handbook of Natural Language Processing*. Taylor & Francis, 2000.

[6] T. Kiss and J. Strunk, "Unsupervised multilingual sentence boundary detection," *Computational Linguistics*, vol. 32, no. 4, pp. 485–525, 2006.

[7] S. Bird, "NLTK: The Natural Language Toolkit," in *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, (Sydney, Australia), pp. 69–72, Association for Computational Linguistics, July 2006.

[8] J. Read, R. Dridan, S. Oepen, and L. J. Solberg, "Sentence boundary detection: A long solved problem?," in *Proceedings of COLING 2012: Posters*, pp. 985–994, 2012.

[9] D. D. Palmer, "Satz - an adaptive sentence segmentation system," *ArXiv*, vol. cmp-lg/9503019, 1995.

[10] K. Evang, V. Basile, G. Chrupała, and J. Bos, "Elephant: Sequence labeling for word and sentence segmentation," in *Proceedings of the EMNLP 2013*, 2013.

[11] IBM, "IBM Cloud Learn Hub," 2021.

[12] spaCy, "spacy industrial-strength natural language processing in python," 2021.

[13] K. Ortmann, A. Roussel, and S. Dipper, "Evaluating off-the-shelf nlp tools for german," in *KONVENS*, 2019.

[14] "Lstm cell: Understanding architecture from scratch with code," 2021.

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013.

[16] A. Toleu, G. Tolegen, and A. Makazhanov, "Character-based deep learning models for token and sentence segmentation," in *Proceedings of the 5th International Conference on Turkic Languages Processing (TurkLang 2017)*, 2017.

[17] R. Wicks and M. Post, "A unified approach to sentence segmentation of punctuated text in many languages," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, (Online), pp. 3995–4007, Association for Computational Linguistics, Aug. 2021.

[18] S. Schweter and S. Ahmed, "Deep-eos: General-purpose neural networks for sentence boundary detection," in *KONVENS*, 2019.

[19] H. Li, "Deep learning for natural language processing: advantages and challenges," *National Science Review*, vol. 5, no. 1, pp. 24–26, 2017.

[20] Q. Zhang, H. Yu, M. Barbiero, B. Wang, and M. Gu, "Artificial neural networks enabled by nanophotonics," *Light: Science & Applications*, vol. 8, p. 42, 05 2019.

[21] P. Branco, L. Torgo, and R. P. Ribeiro, "A survey of predictive modelling under imbalanced distributions," *ArXiv*, vol. abs/1505.01658, 2015.