Karlsruhe Institute of Technology

Master Thesis

A Hierarchical Approach to Neural Context-Aware Modeling

Author:
Patrick Huber

1st Reviewer:
Prof. Dr. Alexander Waibel

2nd Reviewer:
Prof. Dr. Tamim Asfour

Supervisor:
Dr. Jan Niehues

This thesis is submitted in fulfillment of the requirements for the Master's degree of Science

at the

Institute for Anthropomatics and Robotics (IAR)

Interactive Systems Lab

May 28th 2017 to November 27th 2017

Declaration

I, Patrick Huber, hereby declare that I have developed and written the enclosed thesis titled, A Hierarchical Approach to Neural Context-Aware Modeling completely by myself, and have not used sources or means without declaration in the text. Furthermore, I declare, that this thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Date: _____ Signed: _____

Abstract

In the research field of natural language processing, many approaches for translation and transcription tasks are based exclusively on the local context of a sentence. This restriction is reasonable when working with techniques such as n-gram models. Modern machine-learning approaches, for example neural networks, have become far more powerful and thus lifted the restriction of the models to a local context. Therefore, the potential of employing an extended context to enhance natural language processing systems is examined by evaluating the systems on a newly introduced text-understanding task.

In the context of text understanding problems, a novel task to detect semantic errors in contextual text passages is established. The definition of the task does thereby not disclose any information about the position of the modifications, nor give insight about the number of out-of-context substitutions on the data.

To evaluate the computational models on the task, an existing large-scale data source is adopted through a newly introduced automated modification process. Altering the data on a semantic level enables the machine-learning algorithms to be tested against a challenging set of modified text passages that require comprising a broader narrative discourse. The automated modification process is applied to the 2016 TEDTalk corpus containing over 2,000 TEDTalks distributed over various fields and topics. Automating the process enables the cost-effective modification of the complete dataset.

To create a baseline setup, a standard language model and a supervised binary classification model are employed. Both approaches are only referring to one sentence at a time and thus are not capable to infer meaning out of the greater context.

In order to overcome this limitation, a new model topology is developed to assess the potentials of employing an extended context to improve existing systems. The new model-design hierarchically encodes the narrative history and employs the most salient features of the long-term context to support the intra-sentence out-of-context error detection.

The result of the evaluation shows the limitation of the baseline models, indicating that the contextual errors cannot be identified when limiting the computational models to a single sentence. Utilizing the context-aware topology with hierarchical context representations increases the performance, showing the efficiency of hierarchical context models for the semantic error detection task.

Zusammenfassung

Moderne Ansätze im Bereich der natürlichen Sprachverarbeitung verwenden einen sehr beschränkten Kontext für Übersetzungs- und Transkriptionsaufgaben. Diese Begrenzung ist bei der Verwendung von n-gram basierten Ansätzen sinnvoll. Moderne Maschine-Learning Algorithmen, wie Neuronale Netze, sind inzwischen jedoch deutlich fortgeschrittener und machen diese Beschränkung hinfällig. Im Zuge dieser Entwicklung wird das Potenzial eines erweiterten Kontextes für maschinelle Sprachverarbeitungssysteme untersucht und auf eine Textverständnis Aufgabe angewandt.

Im Bereich der Textverständnis Probleme wird dazu eine neue Aufgabe zur Erkennung von semantischen Fehlern in kontextuellen Textabschnitten eingeführt. Per Definition der Aufgabe werden weder Informationen über die Position der modifizierten Wörter im Text, noch die Anzahl an Ersetzungen preisgegeben.

Um die Berechnungsmodelle auf der Aufgabe zu evaluieren wird eine existierende Datenbasis mit einem neuentwickelten Modifikationsprozess angepasst. Die Daten werden dabei auf einem semantischen Level verändert, so dass verschiedene maschinelle Lernverfahren darauf getestet werden können. Die Modifikationen verlangen die Verwendung eines erweiterten Zusammenhangs. Der automatisierte Prozess wird auf die TEDTalk Datenbank aus dem Jahr 2016 mit über 2.000 TEDTalks aus unterschiedlichen Themengebieten angewandt, um den Datensatz für die Textverständnis Aufgabe anzupassen. Die Anpassung des Datensatzes erfolgt aufgrund der Automatisierung des Prozesses zu minimalen Kosten.

Als Ausgangspunkt für die Evaluation wird ein einfaches Sprachmodell sowie ein überwachtes, binäres Klassifikationsmodell trainiert und getestet. Beide Ansätze basieren dabei nur auf einen Satz und sind deshalb nicht in der Lage Informationen aus dem Zusammenhang zu erschließen.

Um diese Beschränkung zu überkommen wird ein neues Modell entwickelt, welches das Potenzial eines erweiterten Kontexts für existierende Systeme aufzeigt. Der neu entwickelte Ansatz fasst den Zusammenhang des Texts dabei hierarchisch zusammen um die Erkennungsrate von Kontext-Fehlern zu verbessern.

Das Ergebnis der Evaluation zeigt, dass die kontextuellen Fehler beim Verwenden eines lokalen Kontexts von nur einem Satz nicht gefunden werden können. Bei der Verwendung von Modellen mit hierarchischer Kontextrepräsentation wird die Erkennungsrate von Kontext-Fehlern verbessert. Das Ergebnis zeigt damit das Potenzial von hierarchischen, kontextsensitiven Modellen für die Aufgabe der kontextuellen Fehlererkennung.

Acknowledgements

I would like to thank my thesis advisor Dr. Jan Niehues of the Interactive Systems Lab at the Karlsruhe Institute of Technology (KIT). The door to Jan's office was always open whenever I ran into an issue with my research or writing. He consistently supported me to make this thesis my own work and steered me in the right direction whenever needed.
I would also like to acknowledge Lansi Chu and Tobias Huber for their support and continuous encouragement as the proof readers of this thesis, and I am greatly thankful for their very valuable comments.
Finally, I want to express my profound gratitude to my parents and to my whole family for providing me with help and advice throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you.

Patrick Huber

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of Artificial Intelligence (AI) has been skyrocketing for the last few years, becoming one of the most promising research topics in the context of computer science. Even though many of the basic concepts and methodologies for state-of-the-art algorithms have already been developed decades ago (McCulloch and Pitts, 1943; Kleene, 1951), the recent abundance of data and the increasing availability of computational resources enabled a revival of machine learning.

Machine Learning as a major research field within AI strives to achieve universal application in solving arbitrary tasks rather than special functionalities and well-defined functions. This trend is also reflected on the research topic of natural language processing (NLP). Closed systems that are only applicable for a limited group of people or under special preconditions are no longer sufficient and mostly replaced by universal approaches being able to adopt and learn self-reliant.

In the area of NLP, many approaches for translation and transcription systems have been solely based on a local context using methodologies like n-grams and sentence-based lattices for decades. With the introduction of neural networks, the base concept for the models changed, but the restriction to a very limited scope remained. The context of most modern translation and transcription systems is still limited to a single sentence, not considering valuable information of the greater context.

1.1 The Institute

The Interactive Systems Lab (ISL) formed in 1989 is part of the Institute of Anthropomatics and Robotics (IAR) at the Karlsruhe Institute of Technology (KIT).

The IAR focuses on human centered research with the goal to enhance human-machine collaboration (IAR).

In this context, the ISL is a research lab aiming to enhance multilingual, multimodal communication between humans and computers focused on NLP tasks. The main research topics are speech recognition and synthesis as well as translation systems and language understanding (ISL). The mission and vision of the ISL is not only

to develop new technologies on an academic level, but also to introduce and transition the results directly into the society (ISL). This approach was already successful in creating solutions for real-world problems by introducing Jibbigo in 2009, the first commercially available speech translator on a mobile phone and the first ever lecture translation service in 2012.

1.2 Motivation

Two highly complex tasks in the area of NLP are the transcription of natural speech and the translation of unstructured text. While both tasks use different approaches and methodologies, the most state-of-the-art systems in both areas only take one sentence at a time into account. This limitation is reasonable, as it addresses the most general case. Additionally, the systems complexity increases by adding a longer context through the introduction of new free parameters. Just taking a longer context into account is therefore not advisable. Nevertheless, considering a longer context can highly benefit the performance of the overall system.

1.3 Problem Statement

With the motivation for this thesis mentioned above, it should be determined whether an extended context can enhance the semantic error detection within contextual text passages compared to systems solely relying on a local context.

1.4 Contributions to the Field

We present a new approach to enhance existing models by incorporating a narrative context. The context-aware extension can benefit many computational models in various fields, especially when working with contextual text bases.

A prominent example where the context-aware extension can add direct value to the overall computation is automatic speech recognition (ASR). In addition to the acoustic model, the language model plays a crucial role in the fundamental equation of speech recognition. The context-aware models proposed within this work can highly benefit the language model by extending the employed context.

Another application is to augment translation systems. State-of-the-art systems are mostly utilizing the advantages of asynchronous encoder-decoder neural network designs. To extend these models to take a longer context into account, an additional computation step can be added after the translation itself. The history of prior translated sentence can then be employed to assess the currently translated sentence and enhance the reliability of the translation.

1.5 Structure of the Thesis

This thesis contains eight chapters describing the process to design and evaluate the newly proposed context-aware computational model. Following this chapter, the main concepts used within the thesis are introduced in chapter 2 and the previous work conducted in the wider area of context-aware models is discussed in chapter 3.

In chapter 4, the newly developed semantic error detection task is introduced and formally described. Subsequently, the developed modification process to extend the evaluation dataset with out-of-context tokens is presented. The chapter finishes with a short remark on the data partitioning process to separate the data into three mutually exclusive sets for training, development and testing.

Chapter 5 discusses the data preprocessing and makes the first decisions influencing the design of the models. It describes the transformation process to convert the data into adequate inputs for the models. Following this, multiple baseline model topologies are introduced and explained. Consecutively, the design of the newly proposed context-aware model is described in this chapter.

The evaluation of the computational approaches is shown in chapter 6. Multiple instances with different hyper-parameters are implemented and tested. The approaches are assessed against the development dataset in a first step and the best performing model is subsequently evaluated against the test set. The chapter closes with an overview of all model instantiations and their respective performance.

Finally, a conclusion is drawn in chapter 7 and further steps are shortly described in chapter 8.

Chapter 2

Background

In the following chapter the basic terminology, algorithms, computing systems and frameworks that will be used within this thesis, are described. This introduction should give a fundamental understanding of the basic concepts and provide an overview of the context.

2.1 Neural Networks

Artificial neural networks (in the following also referred to as neural networks) have been firstly introduced by McCulloch and Pitts (1943) in their work *A logical calculus of the ideas imminent in nervous activity*. At the time, the approach was purely conceptual, describing the idea of a connectionist network consisting of neurons inspired by the biological brain. Every element in the network is receiving inputs, processing them and generating outputs. Despite the purely conceptual nature of artificial neural networks, the basic research by McCulloch and Pitts already covers the most vital parts of current state-of-the-art neural networks. Unlike the classical tasks that computers were initially developed for, neural networks are trying to solve tasks that are inherently easy for humans to perform, but particularly difficult for machines. One of the most famous examples for this type of tasks is pattern recognition (Shiffman, 2012).

Since the first mentioning of artificial neural networks in 1943, many different types and adoptions for specific tasks have been developed. In image recognition and face detection, convolutional neural networks (CNNs) are widely used. In the field of NLP recurrent neural networks (RNNs) are common. Despite the differences in the architecture, all these neural networks share a set of commonalities including the basic computational units, so called perceptrons or artificial neurons.

2.1.1 Perceptron

The perceptron (or artificial neuron) is the basic computational unit in an artificial neural network; it resembles a mathematical model of a biological neuron. A simplified

model of a biological neuron is shown in figure 2.1.

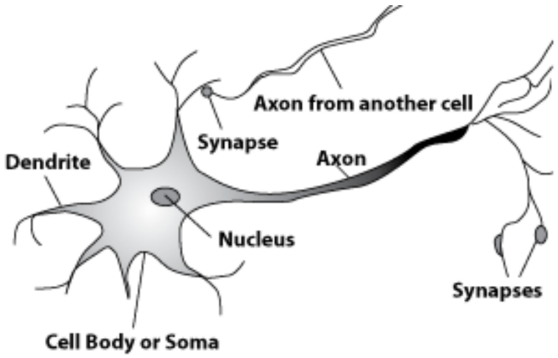

FIGURE 2.1: Simplified Model of a Biological Neuron (Aboukarima et al., 2015)

Within the model, the three major components of a biological neuron are displayed. The **dendrites** receive electrical signals from other cells, representing the inputs of the neuron. The **neuron body** processes these signals received from the dendrites and activates the **axon** to transmit an output signal to connected cells. At the synapses, located between the dendrites and axons, the electrical signals are modulated (Aboukarima et al., 2015).

In comparison to the biological neuron, figure 2.2 shows the architecture of an artificial neuron.

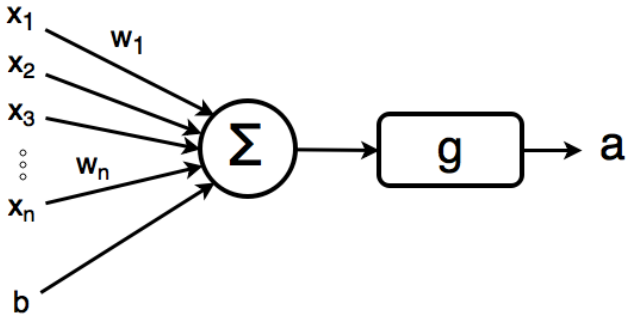

FIGURE 2.2: Model of a Perceptron derived from Jain et al. (1996)

All major parts of the biological neuron are also modeled in the perceptron. The inputs $X = \{x_1, x_2, x_3, ..., x_n\}$ represent the dendrites. Instead of electrical signals, the perceptron receives numerical values as its inputs. The characteristics of the synapses are also modeled within the artificial neuron by multiplying the inputs X with a weight matrix W. The behavior of the biological cell body, emitting an output signal when the inputs outreach a certain threshold, is calculated as a weighted sum of the inputs with the weight matrix $\sum X * W$ by the perceptron. On this summation of inputs and weights, a non-linear activation function is subsequently applied to determine the

output of the artificial neuron. Some of the common activation functions, including the sigmoid and hyperbolic tangent non-linearity, are shown in figure 2.3.

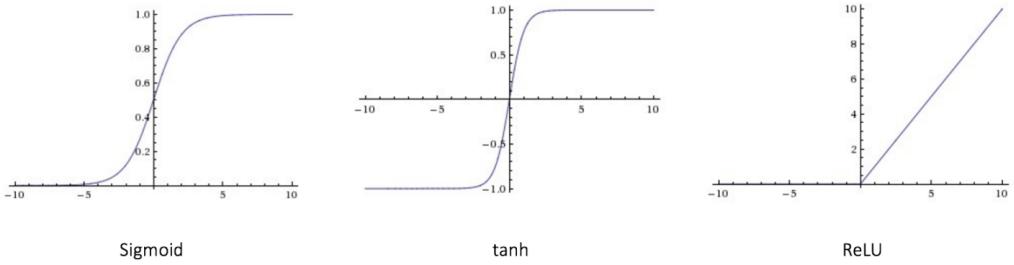

FIGURE 2.3: Common Neural Activation Functions (Karn, 2016)

The complete computation of the perceptron can be described by the equation

$$a = g(\sum_{n} W_{\mathrm{n}} * X_{\mathrm{n}}) \tag{2.1}$$

with g representing the activation function of the neuron and a being the output of the perceptron. While a single perceptron, fully described by equation 2.1, only executes a simple and very limited computation, the artificial neural network's complexity is introduced by the connection of these simple units.

2.1.2 Feed-Forward Neural Networks

Feed-forward neural networks are one of the most common and basic types of connectionist models. The network composes of a hierarchical arrangement of perceptrons separated in discrete layers building on top of each other. The information within the network flows strictly in one direction, from the input layer towards the output layer, without any cycles (Figure 2.4). Thus, feed-forward networks are a generalization of the simple perceptron, extending the functionality to be able to create universal approximations of arbitrary functions (Hornik et al., 1989).

2.1.3 Recurrent Neural Networks

In comparison to feed-forward neural networks, RNNs contain directed cycles to allow the system to embody a dynamic temporal behavior. This allows RNNs to overcome the restriction of feed-forward and convolutional networks, which only accept fixed-sized inputs and produce fixed-sized outputs. Furthermore, these models are also limited to a set amount of computational steps determined by the number of layers in the model. RNNs on the other hand enable an arbitrary number of operations within the network on both input and output sequences, thus making RNN computations

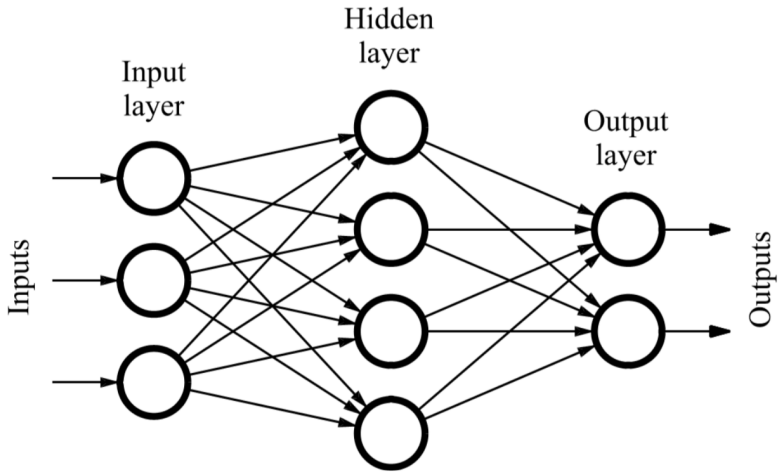

FIGURE 2.4: Feed-Forward Network Topology (Quiza and Davim, 2011)

turing-complete (Siegelmann, 1995). These characteristics are especially interesting for NLP tasks. Working with flexible length sentences as inputs and outputs, text-based tasks highly benefit from the flexible layout of RNNs.

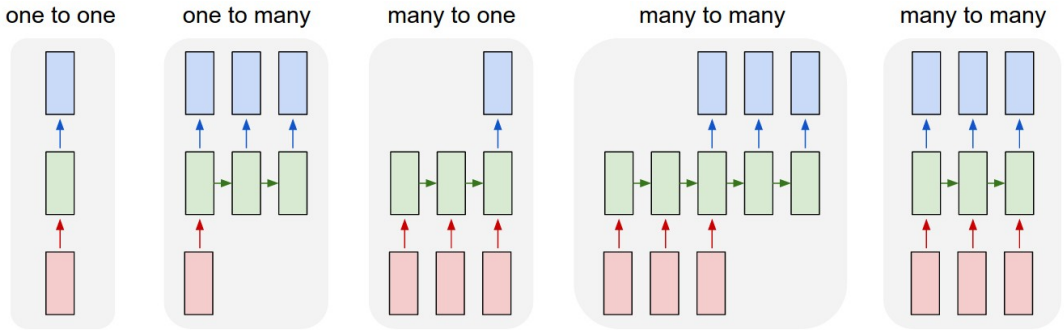

FIGURE 2.5: RNN Topologies (Karpathy, 2015)

The ability of RNNs to employ flexible length input and output sequences resulted in various architectural patterns to be developed. The most common designs are displayed in figure 2.5. Within the figure, every rectangle represents a tensor, while arrows indicate computations.

On the left side, a standard feed-forward network, as described in section 2.1.2, is shown for comparison (one-to-one). The feed-forward network takes a fixed-size tensor as an input and returns a fixed-size output tensor. Next to the feed-forward network, an unfolded RNN with fixed-size input and sequence output is displayed (one-to-many). This type of network is frequently used for tasks like image captioning, where an input picture with a set amount of pixels is entered, while the output is a sequence of words describing the picture. The many-to-one architecture in the

middle of the figure takes a sequence as the input of the model and outputs a fixed-size tensor at the end of the computation. This type is widely used within the area of sentiment analysis with a sentence fed into the network and the sentiment of the sentence returned. For tasks with input and output sequences, two different architectures can be applied. An asynchronous design as shown in the second model to the right, or a synchronous approach as shown on the right. The asynchronous approach first feeds in the complete input sequence. After all the inputs are entered, the system starts to output the resulting sequence. Through this asynchronous design, the whole information of the input sequence is stored within the network before any output is made. A common use-case for this kind of network topology is neural machine translation, where the whole input sentence in a source language L_1 is fed into the system before the sentence is translated into the target language L_2. In contrast to that, the synchronized many-to-many topology takes one input of the sequence and calculates one output simultaneously. This approach is used for many sequence classification tasks (Hochreiter and Schmidhuber, 1997).

2.2 Language Models

A language model can be described as an estimator on how likely a sequence of words W is a valid sentence in a language. A language model LM should therefore be true in the following cases:

$$pLM(\text{"}You\ have\ a\ nice\ dog\text{"}) > pLM(\text{"}A\ nice\ dog\ you\ have\text{"}) \tag{2.2}$$

$$pLM(\text{"}You\ have\ a\ nice\ dog\text{"}) > pLM(\text{"}You\ think\ a\ nice\ dog\text{"}) \tag{2.3}$$

This small example shows that the choice of words as well as the order of words in W is important for the performance of the language model LM. To target that, language models are applied on a sequence of words W rather than an unordered set. The traditional approach to solve this problem is to use an n-gram model. N-gram models are a statistical tool to compute the probability of a given word w_n to appear after a known sequence of words (w_1 , w_2 , ..., w_{n-1}). The equation of a general n-gram model using the chain rule is described by

$$p(w_1, w_2, ..., w_n) = p(w_1) * p(w_2|w_1) * ... * p(w_n|w_1, w_2, ..., w_{n-1}) \tag{2.4}$$

Since it is not possible to find sufficient examples for every word w_n after a sequence of predecessors w_1, w_2, ..., w_{n-1} in the text, the n-gram model is extended by the Markov assumption and narrowed down to a n-gram model with usually $n <= 4$. To normalize the n-gram probabilities, the count is divided by the number of appearances

of the word, the unigram probability.

$$p(w_2|w_1) = \frac{count(w_1, w_2)}{count(w_1)} \tag{2.5}$$

There are many extensions to the standard n-gram model to enhance performance, such as different smoothing approaches (Add-One, Good-Turing, Kneser-Ney), Back-Off methods and interpolation (Koehn, 2009). These n-gram models have been state of the art for many years.

Through the fast increase of computational resources within the last decades and the new possibilities to compute neural networks, neural language models are improving rapidly and already outperform the classical n-gram approach, as described by Bengio et al. (2003). Neural language models thereby use neural networks to calculate the probability function. One of the reasons the neural approach is superior to the n-gram approach is the way it targets the curse of dimensionality. While n-gram models fight the problem by limiting the history of the text, neural language models use a so-called embedding layer to reduce the complexity. This approach, first brought up by Bengio et al. in 2003, represents every word in the vocabulary as a real valued point in a vector space, which drastically reduces the amount of parameters of the network. Word embeddings are then used as inputs for the RNN. The use of RNNs allows the neural language model to learn significantly longer contexts than the respective n-gram approach.

2.3 Frameworks

Throughout the last years, machine learning frameworks have followed the trend of AI with an increasing amount of high-quality computational libraries. In the following sections, some of the most advanced neural network libraries at the time of writing this thesis will be analyzed and compared due to their suitability for the task.

2.3.1 Keras

Keras is a high-level open source machine-learning library especially designed to facilitate neural networks. The library is fully written in python building on top of Tensorflow, Microsoft Cognitive Toolkit (CNTK) and Theano. The focus of the framework is on fast prototyping, reducing the time from the idea to the finished result and accelerating research. The framework supports CNNs as well as RNNs and can be executed on CPUs and GPUs (Chollet).

With these properties, the library is suited for standard neural network topologies, which are inheritably supported through high-level wrappers. The framework can also be used for individual network designs through the custom extension concept.

Nevertheless, new and individualized network topologies are often easier to generate from low-level libraries.

2.3.2 Tensorflow

Tensorflow is an open source low-level machine-learning interface originally developed by Google to implement and execute machine-learning algorithms. The library can be used for a wide variety of computational tasks including neural network models. Tensorflow is supported by a steadily growing community and delivers production-ready implementations within a wide range of areas such as speech recognition, computer vision, NLP and many more (Tensorflow Documentation). The Tensorflow library comes in a separate version for CPU and GPU enabled computations. The stateful dataflow-like model within Tensorflow allows the computations to be flexible and comprehensive. To enhance debugging procedures within large graphs, the integrated Tensorboard functionalities enable visual representations of the topologies.

The characteristics described above make the Tensorflow framework particularly interesting for customized neural networks with special topologies. Compared to the Keras framework, Tensorflow offers a more flexible approach to create computational graphs from low-level modules (Abadi et al., 2016).

Chapter 3

Related Work

There are multiple related approaches to the newly introduced semantic out-of-context error detection task. The LAMBADA dataset introduced by Paperno et al. (2016) is one of the first and most comprehensive data sources addressing the task of text understanding by the means of word prediction. One of the main differences between our newly introduced task and the LAMBADA dataset is the possibility to modify large data sources at a low cost. Through our fully automated modification process to enable datasets on the task, large corpora can be modified with out-of-context tokens. This greatly benefits deep learning algorithms, as they need an extensive amount of data to be trained on. The modification process introduced in this thesis, thus, represents a cost efficient alternative to hand crafted databases, which are not only tedious to create, but with $1.24 per data point (Paperno et al., 2016) are also very expensive. Other related datasets are the Children's Book Test (CBT) dataset by Hill et al. (2015) and the CNN/Daily Mail benchmark by Hermann et al. (2015), which both focus on summarization and question answering tasks. In the domain of NMT, Sennrich (2016) introduced a dataset with automatically inserted errors focusing on advanced computational models. The paper by Burlot and Yvon (2017) proposes an evaluation process for NMT models, assessing the morphological properties of a system. The process substitutes nouns, as well as other part-of-speech tokens with filtered and randomly chosen replacement words.

Within the domain of context-aware models, most approaches focus on question answering tasks rather than semantic out-of-context error detection. Related work within the area of question answering is conducted by Iyyer et al. (2014a) showing that recursive neural networks can be combined with a parse tree (De Marneffe et al., 2006) and a multinomial logistic regression classifier to achieve good results on trivia-like tasks by extracting the essential information from the context. The task introduced by Iyyer et al. varies from this work, as it targets a question-answering problem based on a short paragraph rather than an out-of-context prediction on long narratives. Additionally, it utilizes a heterogeneous methodology instead of an end-to-end connectionist learning approach. Other related work is conducted by Iyyer et al. (2014b).

Chapter 4

Task Definition and Preparation

The following sections describe the newly introduced semantic out-of-context error detection task. The first section (4.1) defines the task itself and shows the significance and complexity of the semantic out-of-context error detection. In the second section, the process to create the ground-truth is described, followed by a section on suitable large-scale datasets. The following section evaluates the data transformation process and the last section describes the data partitioning.

4.1 The Task

Within the area of text understanding, our new evaluation task is defined as the detection of out-of-context errors in contextual text passages. Therefore, artificially inserted out-of-context error tokens are uniformly distributed over the dataset. In comparison to approaches with well-known target words $\mathrm{w_n}$, the out-of-context word replacements within this task are at random positions $\mathrm{w_r}$ in the series of word-tokens described by the ordered sequence of words W =($\mathrm{w_1}$,$\mathrm{w_2}$,...,$\mathrm{w_m}$) and $w_\mathrm{n}, w_\mathrm{r} \in W$. Compared to the task of finding the correct word $\mathrm{w_n}$ at a known position, which can be described as a classification task, the presented task can be interpreted as a binary sequence labeling problem defined by the input sequence W = ($\mathrm{w_1}$,$\mathrm{w_2}$,...,$\mathrm{w_m}$) of length M representing the text passage and the output sequence $\mathrm{L_{ooc}}$ = ($\mathrm{l_1}$, $\mathrm{l_2}$, ..., $\mathrm{l_m}$), also of length M, with $l_\mathrm{i} \in L_\mathrm{ooc}$ representing the label of the input element $w_\mathrm{i} \in W$. The labels $\mathrm{L_{ooc}}$ thereby separate the two classes $\{0, 1\}$, representing valid-context tokens (0) and out-of-context tokens (1). Through the definition of the task, every word $w_\mathrm{r} \in W$ needs to be assessed against every other word $w_\mathrm{q} \in W$ in the sequence with $r \neq q$. The task definition does not provide any information about the position of the modifications, nor give any insight about the total number of out-of-context substitutions on the data.

4.2 Transformation Process

The goal of the dataset transformation process is to modify an existing database by artificially inserting out-of-context errors in text passages with strong contextual features. The six-staged substitution procedure is fully automated to enable the modification of entire large-scale datasets without the costly validation of the data by human subjects. A crucial task emerging through the automated processing is the reasonable replacement of context related words. The substitutions fulfill two requirements. First of all, the modified dataset serves as the ground truth to test the train context-aware computational models against. Secondly, through the out-of-context modifications on the training set, supervised models can be trained on the data. The modifications are achieved through the following computational steps.

4.2.1 Tokenization

To be able to make predictions, it is crucial to carefully split the continuous data into word tokens. Simply splitting the data at white spaces results in poor performance of the model, as the simple tokenization does not cover special cases like contractions (*don't* instead of *do not*). The tokenizer needs to separate contractions and punctuations, which both do not contain white spaces. To also take these special cases into account, the Natural Language Toolkit (NLTK) tokenizer framework is used to split the data. The careful tokenization of the data consumes a noticeable amount of computation time. To save computational resources, the tokenized result is saved on the hard drive to be reusable. After the tokenization of the data, the tokens are sorted by their appearance in the data. The count of every word in the data source will be used later to improve the insertion of out-of-context samples.

4.2.2 Dataset Filtering

To enhance the dataset quality, non-contextual parts of the data are removed, especially excluding self-contained short text passages with less than 200 words.

4.2.3 Part-of-Speech Tagging

For the semantic out-of-context substitutions, only certain part-of-speech (POS) classes are taken into account. Thus, every token in the dataset is assigned a POS category. As described by Paperno et al. (2016), context is especially critical to nouns, whereas other POS classes can often be inferred directly out of the local context of a sentence. Our substitution process, therefore, focuses on the replacement of nouns.

4.2.4 Candidate Selection

To ensure that a text passage contains a sufficient context, the nouns determined by the POS-tagger are filtered for contextual coherence. A context is assumed if the same noun appears multiple times within the same text passage. The last appearance of the noun within a context of up to ten sentences qualifies as a suitable replacement and is saved as a potential out-of-context substitution candidate. Out of the list of potential replacement candidates, a predetermined number of tokens are randomly selected according to a uniform distribution.

4.2.5 Contextual Replacement

For every selected token in the original dataset, a syntactically suited replacement token is determined. This task is especially challenging, as the substitute word should not be semantically close to the original one. Replacing with such words leads to substitutions with synonyms and is difficult to find. If the substitute word on the other hand is too syntactically different, the task can be solved without the use of a narrative model. To solve this problem, two different approached are analyzed:

- Replacement by word2vec similarity
- Replacement by appearance window

The *replacement by word2vec similarity* is based on Google's pre-trained word embedding model *word2vec*, which compares words in a high-dimensional real-valued space. The substitution algorithm takes the original word and computes 50 related words from the *word2vec* model based on their cosine similarity (Word2Vec). This approach has two major drawbacks.
The first drawback is the nature of an external data source. The Google *word2vec* model contains over 1,000,000 common English words, but does not cover all the potential words to substitute. This leads to the problem, that the original words that have not been found within the *word2vec* model cannot be encoded.
Secondly, the computational approach to find related words with the *word2vec* vectors is not consistent in terms of difficulty, as with decreasing cosine similarity the retrieved words are quickly turning from synonyms to purely random words. By automatically choosing a word, the chance of either taking a synonym or a syntactically non-fitting word is high. Therefore, the *word2vec* approach does not fulfill the requirements for this task.

The *replacement by appearance window approach* is based on the assumption that words with a similar word count on the original dataset are suitable substitutes. There are multiple reasons that support this hypothesis.
(1) Words with a similar word count appear about equally often within the language. This avoids very common words being substituted by rare words.

(2) The replaced words are typically not related to the original word, as the overall word count generally does not infer semantic affiliation. The chance to exchange a word with its synonym is therefore relatively low.
(3) Empirical tests with the *replacement by appearance window approach* have shown to be a reasonable trade-off. The semantic differences between the words can be found within the context while it is difficult to spot the replacements within one sentence. Within the appearance window further filtering regarding the tense and grammatical number are executed to determine the most suitable substitution.

As a result of this, the *replacement by appearance window approach* is chosen to find suitable word substitutes within a similarity window. The window is initially set to ± 10 words around the appearance of the original noun and can be extended up to ± 30 words. Within the appearance window the substitute noun is chosen randomly.

4.2.6 Output

After the process is finished, the result is saved on the hard drive. To enhance the flexibility of the data for different tasks, multiple outputs are generated. The outputs are:

- The change-log file with all the changes in JSON format (changes_text)
- The original text (original_text)
- The modified text (modified_text)
- The text with indicated modifications (indicated_text)
- The text with synchronous original and modified words (sync_text)

Figure 4.1 shows the high-level program flow of the manipulation process.

4.3 Comparison of Data Sources

Criteria	OpenSubtitles	Reuthers	TED
Dataset size	106k movies	1.8m articles	2.6k talks
Availability	Free	On request	Free
Contextual features	Low	High	High
Format	XML	Text	XML
Miscellaneous	Different genres	Short news articles	Long talks

Table 4.1: Comparison of Different Data Sources

To determine a suitable dataset to modify with the described substitution process and test the computational models on, three different types of data sources that

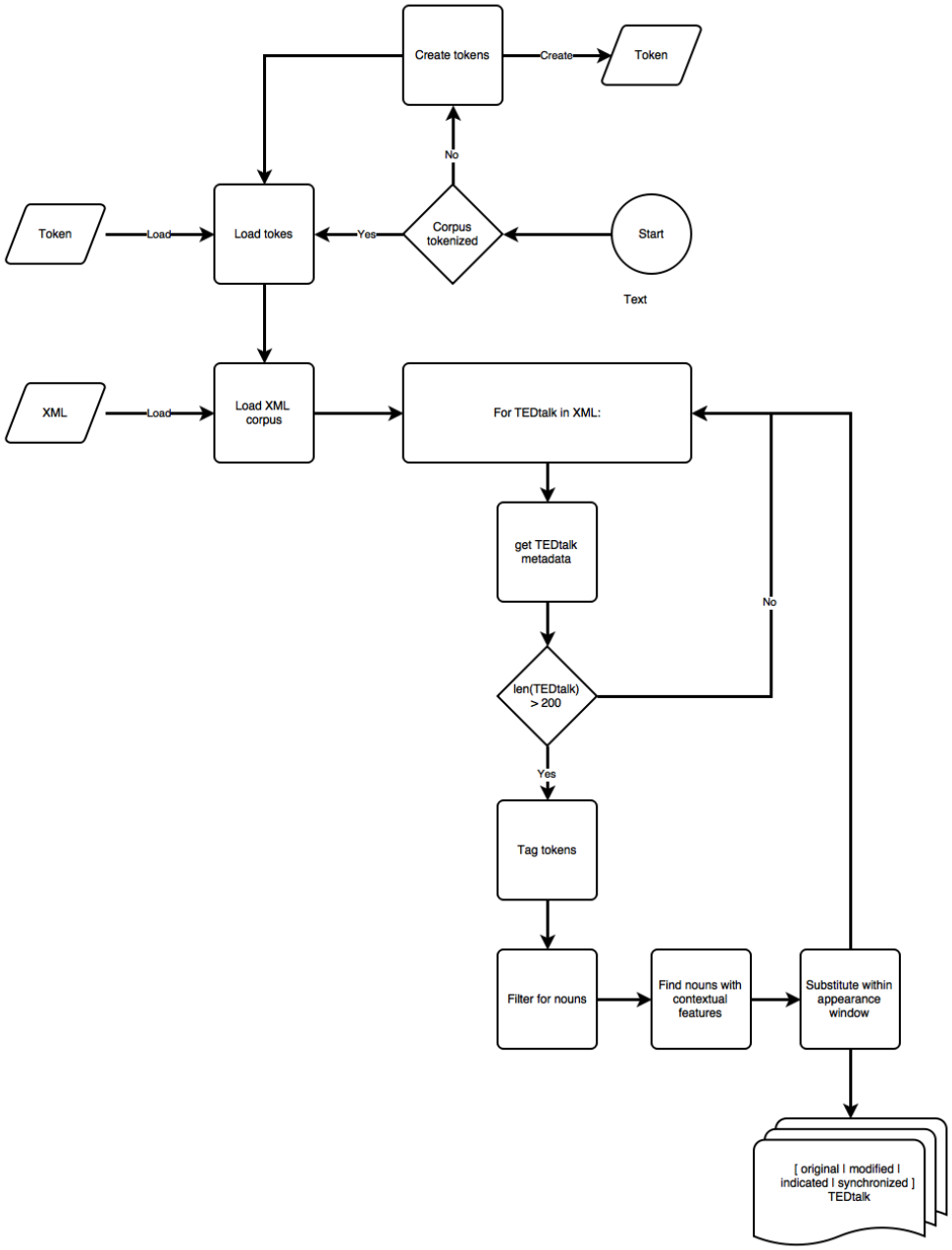

FIGURE 4.1: Dataset Generation Process

can provide narrative information are compared. The two main factors taken into account are the dataset size and the contextual features of the data. To complete the comparison, further criteria like the availability, the format and miscellaneous features are analyzed.

One type of data that is often taken into account is *movie subtitles*. The main advantages of movie subtitles are the wide variety of genres and the large amount of available data. With over 106,000 different movies, the OpenSubtitles database

is a huge and free corpus to train on (Tiedemann, 2016). However, as movies are multimodal with a heavy focus on visual information, the subtitles alone are often not enough to recover contextual features; especially fast changes in settings and multiple storylines make the retrieval of a continuous narrative inaccessible. Therefore, the OpenSubtitles movie database is not suitable for the task.

The second frequently used dataset type is *news articles*. News transcripts cover a broad field of topics and have high internal cohesion, covering one narrative story per article. The Reuthers news corpus contains 1.8 million news stories with strong narrative features (Lewis et al., 2004). The corpus is not free to download, thus, a special request needs to be made to the owners. Even though the data source fulfills most of the requirements, the main problem with the news articles dataset is the average length per news story. To be able to resolve long-term contextual features, the dataset needs to have an extended narrative story. As the news stories provided are not of sufficient length, the corpus is not suitable for the task.

The third type of data source in this comparison is *presentations*. Speeches are normally coherent in their topic as well as multiple thousand words long. One famous corpus containing a wide variety of presentations is the TEDTalk dataset, containing over 2,600 talks. Every talk covers one specific topic and is scheduled to be between 30 to 45 minutes in time. With these two factors, coherence and length, the dataset is a great fit for the task of context comprehension.

Out of this comparison, the TEDTalk database is chosen as the suitable database for the evaluation of the computational models. The data contains transcripts of presentations held between 2007 and 2016 and can be downloaded in a structured XML format from the WIT^3 website, which simplifies the processing of the corpus (Cettolo et al., 2012). Apart from the data, the transcripts of the talks contain additional metadata. The TEDTalk corpus has strong contextual features and long, coherent contexts, which makes the dataset especially suitable for the task of context comprehension by error detection. Additionally, it covers a wide variety of topics, which keeps the data unbiased and supports generalization.

Figures 4.2 and 4.3 show two outputs of the modified TEDTalk data. Figure 4.2 displays the change-log JSON-file containing the collection of changes applied to the TEDTalk. Every dataset in the collection contains the index of the changed token, the original word and the replacement. With this information the changes are fully defined.

$$\begin{bmatrix} 165 & performance & engineering \\ 257 & oceans & businesses \\ 563 & system & place \end{bmatrix}$$

FIGURE 4.2: TEDTalk Change-Log

Figure 4.3 contains a short piece of the modified TEDTalk by Marina Abramović *An art made of trust, vulnerability and connection* from 2015. One word within the extract has been replaced according to the modification process explained above.

> *Welcome to the performance world. First of all, let's explain what the performance is. So many artists, so many different explanations, but my explanation for* ***engineering*** *is very simple.*

FIGURE 4.3: Modified TEDTalk File

4.4 Evaluation of the Transformation Process

With the process described above, all TEDTalk transcripts between 2007 and 2016 are modified with out-of-context errors (Cettolo et al., 2012). The design of the modification process assures that the changes are semantical rather than syntactical. To prove the quality of the data modifications, the distribution of the original TEDTalk tokens is compared with the replaced words (Figure 4.4). Therefore, every word in the vocabulary is assessed at the replaced positions. The probabilities of the words are sorted and divided into four equal quarters representing the four classes on the x-axis. The distribution of the original TEDTalk tokens, with 93.35% of the words within the top 25% of the vocabulary, shows the effectiveness of the baseline language model on the original data. The distribution of the modified words, with 84.6% of the artificially replaced words within the highest quarter (75%-100%), shows that the replacements also fit well in the local context.

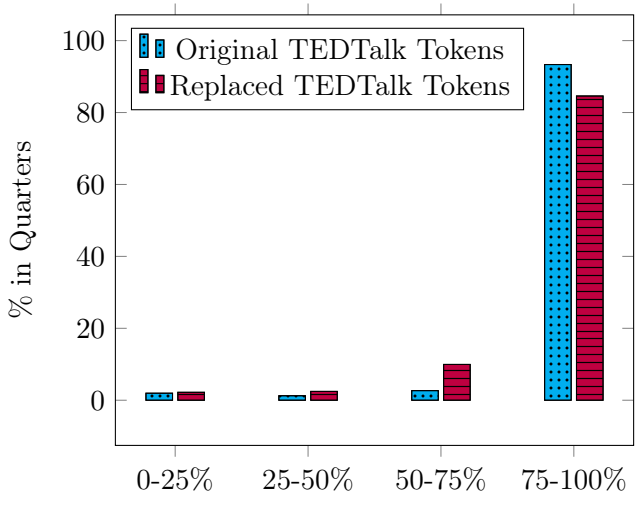

FIGURE 4.4: Distribution of Original and Modified Word Predictions

As the analysis is executed on the unseen development set, the substitutions in general do not seem to be obvious when only analyzing the sentence itself. The comparison of the original words with the modified words at the same position shown in figure 4.4 proves that the modifications on the TEDTalks keep the syntactical

correctness within the sentence as the replacements cannot be found with the standard language model. Please note that this comparison does not give any insight in how good the language model can predict semantically wrong words within the text, as only the positions of the modified tokens are taken into account for the comparison. The language model itself and the result on the modified TEDTalk corpus will be discussed later.

4.5 Data Partitioning

The 2,600 modified TEDTalks are split into disjoined training, development and test sets. For the ratio of the sets, the commonly used $60 - 20 - 20$ split is utilized to randomly distribute the data into mutually exclusive partitions with 60% training, 20% development and 20% test data.

Chapter 5

Model Topology

To assess the influence of the context for the semantic error detection task, as described in section 4.1, a novel context-aware network topology is presented. The first section describes the necessary preprocessing to enable the data to be inserted into the networks (5.1). To compare the newly developed network designs against a baseline, standard neural network models employing a local context are described in section 5.2. The extended topology integrating a broader context into the computation is subsequently described in section 5.3.

5.1 Preprocessing

To be able to feed data into a neural network, multiple preprocessing steps need to be performed. In the first step, an atomic element is defined. As the data in this case is in text format, there are two established ways to encode the input for the neural network: character-based or word-based. In the next step, the data is transformed into a corresponding numerical representation to run the computational model.

5.1.1 Atomic Elements

The first task of the preprocessing is to decide on the atomic element to use within the models. This decision influences the preprocessing of the data as well as the design and architecture of the computational model. Common approaches are to encode the input word-by-word or character-by-character. This initial decision has great impact on the performance of the model.

Character based systems have a single symbol as input and output at each time step, leading to predictions on character level. During training, the network needs to learn not only the structure of sentences, but also the word structure to output valid words, which makes the tasks inheritably more difficult. The approach is most commonly used for languages with rich morphology such as Finish, Turkish and Russian. The sub-word information introduced through the character level improves the

system for these languages. In addition, character-based systems on average need less parameters than word-based approaches (Kim et al., 2015; Xie and Rastogi, 2017).

An advantage of word-based models over character-based models is the limitation to a known set of words. While character-based models can create all kinds of fictional words, word-based models are limited to a vocabulary of a known size and a well-defined <UNK>-Token, which represents all other words. This makes many tasks easier to perform, as words can be easily compared. Furthermore, word-based models on average show higher accuracy and lower computational cost than character-based LMs (Kim et al., 2015).

The large vocabulary size in word-based systems introduces large input and output layers, while the hidden layer is relatively small. As the character-based model additionally learns the word structure, the hidden layer is larger for character-based approaches (Kim et al., 2015).

The task to find out-of-context words in a contextual text is, therefore, very suitable for the word-based approach. The use of words as atomic elements additionally simplifies the word-to-word evaluation.

5.1.2 Data Representation

To transform the data into suitable inputs for the computational model, the dataset is loaded from the hard drive into the working memory. The data is subsequently tokenized on sentence and word level to preserve the hierarchical structure of the text. Depending on the chosen vocabulary size n_{vocab}, only the n_{vocab} most frequent words are kept in the vocabulary. All other words are substituted by the <UNK> token. The choice of a proper vocabulary size is an important part of the preprocessing and should not be chosen too small, as the vocabulary size highly influences the performance of the system later on.

To indicate the beginning and end of a sentence, a leading <START> token and a trailing <END> token are added to the data.

As neural networks are purely numeric, words cannot be used as the inputs for the computational model. Therefore, the text is transformed into a numeric representation. To create this representation, there are $(n_{\text{vocab}} + 4)$ classes generated, one class for every word in the vocabulary plus four additional classes for the <UNK>, <START> and <END> token as well as the <PAD> symbol, which will be discussed later. The affiliation of a word to one of the groups is represented through a one-hot-encoding.

In the one-hot-encoding every class is represented as an integer value within an array of length $(n_{\text{vocab}}+4)$. As every class contains exactly one word of the vocabulary and every word is only present in one class, the array consists of $(n_{\text{vocab}} + 3)$ *zero*

values and a single *one* value. This representation is highly redundant and inefficient, especially as the number of classes grows. To represent a word within the vocabulary, $(n_{\text{vocab}} + 4) * 4Bytes$ of memory are used (assuming 32bit integer values). For a vocabulary size of 30,000 words, that results in a memory usage of $30,004 * 4 = 120,016Bytes = 120kBytes$ to encode one input. To overcome this problem the words need to be encoded in a more efficient way.

As the classes in the one-hot-encoding are mutually exclusive, an index-representation can be used, where classes are represented by the index of the one-hot tensor. Instead of having a tensor with $(n_{\text{vocab}} + 3)$ *zero* values and a single *one* value for example at position 22, only the index of the one value, which is index 22, is saved. This way, the sparse $(n_{\text{vocab}} + 4)$ dimensional tensor can be encoded in a single integer value. To transform the words in the vocabulary into the numeric representation, a bidirectional mapping between the words in the vocabulary and unique indexes are created. The transformation is described by the function f and the unordered list $[u_1, u_2, ..., u_{\text{n}}]$ as inputs and returns $[f(u_1), f(u_2), ..., f(u_{\text{n}})]$, with $n > 0$.

Through the definition of the vocabulary, the number of words is limited to a fixed amount. Apart from the limitation of words in the vocabulary, the length of a sentence is limited as well, to be able to input the data into the computational model. Just using the length of the longest sentence as an upper bound, however, leads to excessive unnecessary computation, as an average sentence is normally significantly shorter. It is therefore advisable to choose a smaller sentence length that covers most of the cases. Sentences that exceed the limitation are shortened to the chosen length and sentences that are shorter than the defined sentence length are padded and masked for the network to properly handle multiple length inputs. The padding adds the specified <PAD> token to the end of the sentence to extend the input to the required length. With this unique token defined, the masking can dynamically cut this token out through a simple binary comparison. Figure 5.1 illustrates the complete preprocessing pipeline.

5.2 Baseline Model

Neural language models are the state-of-the-art approach for unsupervised prediction of words based on the sentence history. In this case, the language model is used as the baseline to improve the context comprehension task upon. As one of the first, Bengio et al. (2003) proposed a computational model to target the task of language modeling using neural networks. The model shown in figure 5.2 will be used as a starting point for the creation of the model. The detailed design of the system is illustrated in the following section. After discussing the architecture of the neural language model, a supervised alternative of the model using binary classification will be introduced.

I don't just see a person. I see an opportunity.

$$\downarrow$$

$$\begin{bmatrix} I & don & 't & just & see & a & person & . \\ I & see & an & opportunity & . & & & \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} I & don & 't & just & see & a & person & . \\ I & see & an & < U > & . & & & \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} < S > & I & don & 't & just & see & a & person & . & < E > \\ < S > & I & see & an & < U > & . & < E > & & & \end{bmatrix}$$

$$\downarrow$$

word	<P>	a	an	don	I	just	person	see	't	.	<S>	<E>	<U>
index	0	1	2	3	4	5	6	7	8	9	10	11	12

$$\downarrow$$

$$\begin{bmatrix} 10 & 4 & 3 & 8 & 5 & 7 & 1 & 6 & 9 & 11 \\ 10 & 4 & 7 & 2 & 12 & 9 & 11 & & & \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 10 & 4 & 3 & 8 & 5 & 7 & 1 & 6 \\ 10 & 4 & 7 & 2 & 12 & 9 & 11 & \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 10 & 4 & 3 & 8 & 5 & 7 & 1 & 6 \\ 10 & 4 & 7 & 2 & 12 & 9 & 11 & 0 \end{bmatrix}$$

FIGURE 5.1: Preprocessing Applied to the TEDTalk by Matt Stone

5.2.1 Language Model

The architecture of the baseline neural language model is derived from the original model proposed by Bengio et al. (2003), which employs the most fundamental architecture for a neural language model. The model in figure 5.2 takes an index encoded one-hot representation as the input, which is transformed into a real valued tensor in the embedding layer. The embedded values are fed into the hidden layer using the tanh activation function. The output layer consists of one neuron per class. As activation for the output layer the softmax function is used to create probabilities.

The basic architecture by Bengio et al. (2003) is extended by state-of-the-art algorithms and elements to enhance the models' performance on the task. The recurrent

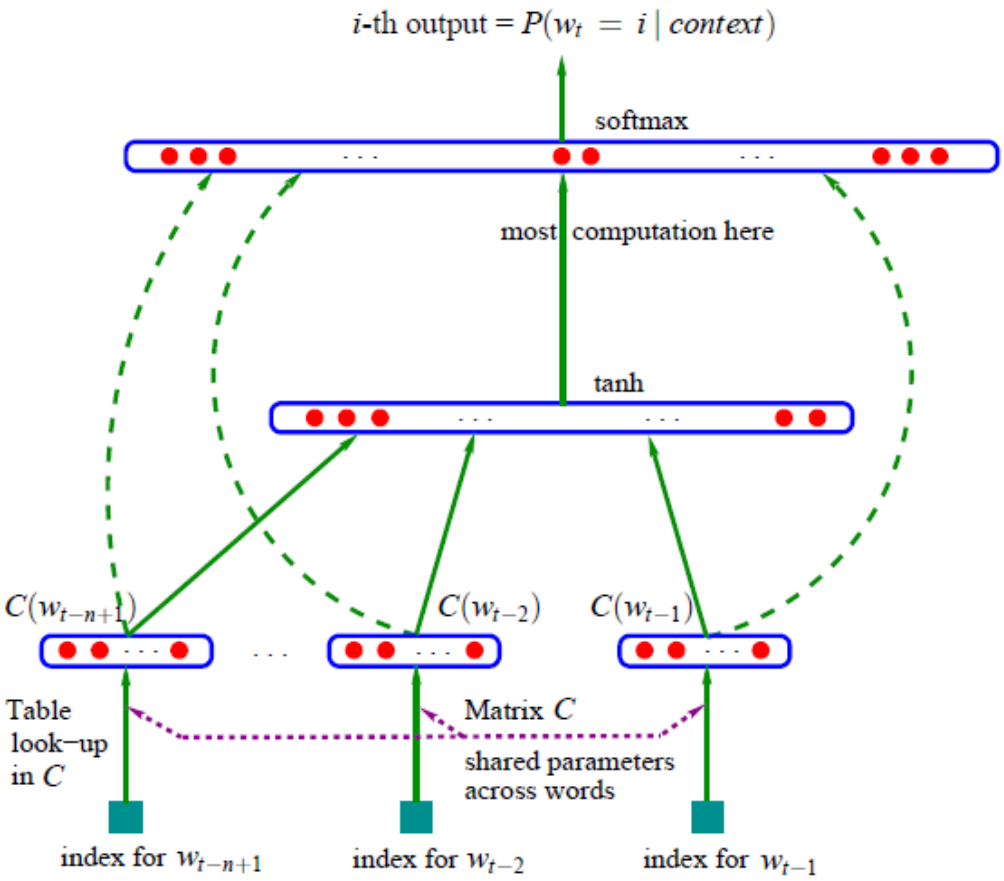

FIGURE 5.2: Basic Neural Language Model developed by Bengio et al. (2003)

neural network described in the paper is realized with standard RNN cells. This type of cells is problematic as it cannot keep long-term memory due to the design of the cell. For the baseline network, the state-of-the-art LSTM units are used within the hidden layer of the network as proven more effectively by Xie and Rastogi (2017). Within the paper *Deep Poetry: Word-Level and Character-Level Language Models for Shakespearean Sonnet Generation* Xie and Rastogi (2017) showed the superior performance of LSTM cells compared to GRU and standard RNN cells for word-RNNs (Sundermeyer et al., 2012). The architecture that is used for the baseline system is shown in figure 5.3.

Input Layer

The input layer is the gateway for the preprocessed data to be entered into the computational model. As the index-encoded representations of the words in the dataset are not related (there is no order to the indexes, as there is no order to the words themselves), the network provides one neuron for every class in the one-hot-encoded data. The outputs of the preprocessing are index-encoded and therefore every word is represented by one integer value. Nevertheless, every integer value in the preprocessing

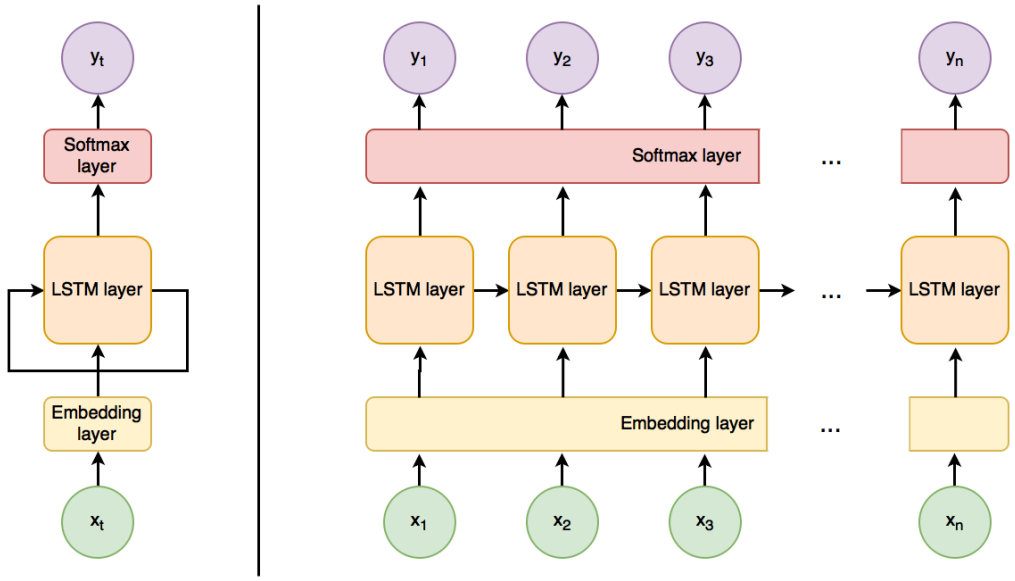

FIGURE 5.3: Folded- and Unfolded LSTM-RNN for the Baseline Language Model, based on Olah (2015)

output does represent a tensor of the vocabulary size n_{vocab} in the one-hot-encoding. For the data to be entered into the neural network, the one-hot-representation is needed. Therefore, the number of neurons in the input layer is fixed to the number of unrelated classes n_{vocab}. The main purpose of the input layer is to provide an entry point into the computational model, as the neurons within the input layer do not perform any non-linear computation. The layer can therefore be seen as a placeholder to enter the data into the network. To simplify the input of data and to enhance the systems' performance, all state-of-the-art deep learning frameworks are supporting the index-representation as input for the one-hot-encoded input layer to save space during training and testing.

Embedding Layer

The next layer within the computational model is the embedding layer, which was first introduced by Bengio et al. (2003). Without the embedding layer, the high dimensional input is propagated through the network and the densely connected layers increase the amount of connections in a polynomial manner. This leads to an immensely high number of weights. The problem, referred to as the curse of dimensionality, is tackled by using an embedding layer.

Instead of propagating the sparse one-hot-encoded input layer, the embedding layer utilizes a combination of static word vectors and flexible weights to densely represent words as real valued points in a high dimensional vector space. This transformation drastically reduces the amount of parameters of the network and learns semantic relations between words jointly during the training process (Bengio et al., 2003; Xie

and Rastogi, 2017). The essential role of the embedding layer is to transform the one-hot-encoded input from the previous layer into an n-dimensional real valued vector by applying the parameterized function W:

$$W : words \to \mathbb{R}^{\mathrm{n}} \tag{5.1}$$

W is typically implemented as a lookup matrix where every row represents one class of the input. The result is a matrix of dimension $[n_{\mathrm{vocab}} * size_{\mathrm{embedding}}]$ initialized with a random uniform distribution within the range $[-1, 1)$. The randomly initialized matrix is static throughout the training process and shared between all inputs. Parameterized by the matrix θ, the matrix-lookup is defined by:

$$W_\theta(w_{\mathrm{n}}) = w_{\mathrm{n}} \times \theta = \theta_{\mathrm{n}} \tag{5.2}$$

Figure 5.4 shows a small example of the transformation within the embedding layer for a matrix θ with vocabulary size $n_{\mathrm{vocab}} = 4$ and an embedding size $size_{\mathrm{embbeding}} = 5$.

$$W_\theta(w_2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}^T \times \begin{bmatrix} 0.2 & -0.5 & -0.8 & 0.1 & 0.9 \\ -0.8 & -0.4 & 0.1 & -0.6 & 0.0 \\ -0.1 & -0.3 & 0.8 & 0.9 & -0.4 \\ 0.9 & -1.0 & -0.3 & 0.7 & 0.5 \end{bmatrix} = \begin{bmatrix} -0.8 \\ -0.4 \\ 0.1 \\ -0.6 \\ 0.0 \end{bmatrix}^T$$

FIGURE 5.4: Embedding Transformation

With this computation, words are transformed into another domain, but as the matrix is immutable, no semantic relations are learned. To be able to learn word embeddings, the outputs of the embedding layer are weighted by the layer connections. The flexible weights, combined with the static word embeddings, add semantic information about the words during the training process.

Figure 5.5 shows an example of the learned word embeddings projected into a two-dimensional value space using the t-distributed stochastic neighbour embedding (t-SNE) technique. The extract shown contains two clusters of words. In the top left corner the words *police* and *military* are close together, in the right lower area, a cluster of names has formed.

Recurrent Layer

The recurrent layer (also called context layer or state) is the essential building block of the computational model to encode the sequential nature of continuous text, which cannot be modeled without a temporal component. Using a standard feed-forward neural network for the language model would therefore exceed the architectural limits very fast. Due to the topology of the RNN model, every time step is not only depended on the direct input at the time but also on the history. Figure 5.6 shows the overall structure of a folded and unfolded recurrent neural network respectively.

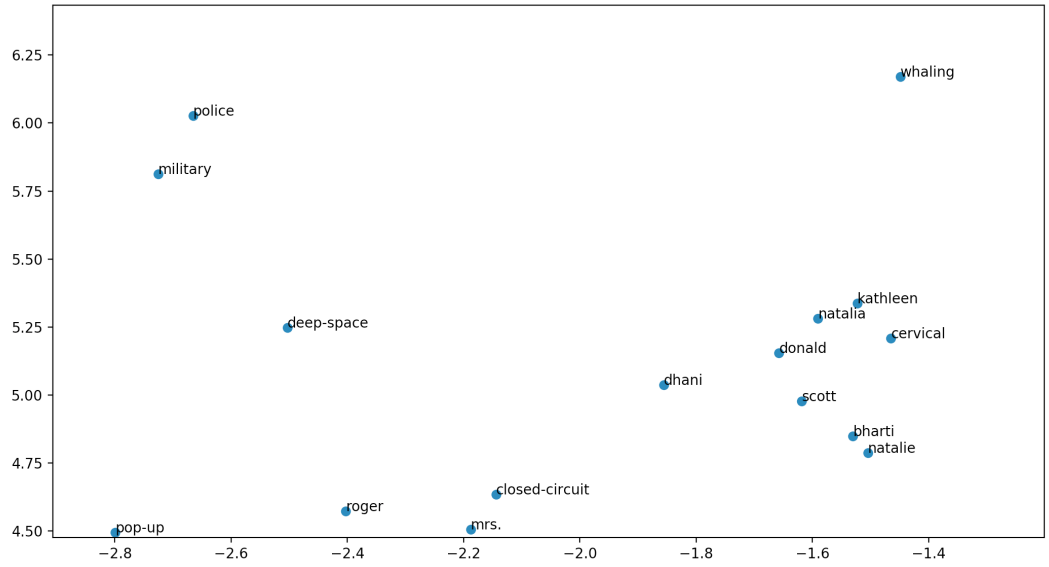

FIGURE 5.5: t-SNE Visualizations of Word Embeddings learned during the Training of the Language Model

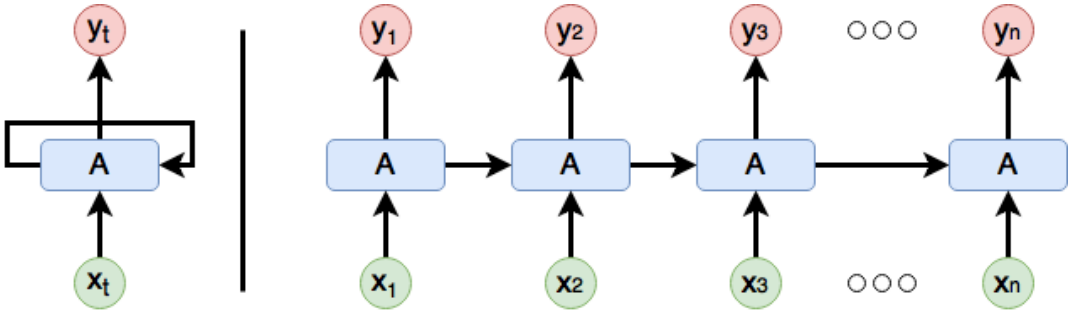

FIGURE 5.6: Recurrent Neural Network Design derived from Olah (2015)

The recurrent layer A represents the main building block of the network. It combines two input sources to compute its internal state $\mathrm{h_t}$: the value from the dense representation of the words, transformed by the embedding layer x_t at time step t and the former hidden state of the network $h_{\mathrm{t\text{-}1}}$ from the last time step $t-1$. Therefore, the output y_t is not only influenced by the input x_t, but by the entire history of past inputs. The RNN displayed in figure 5.6 on the left can hence be understood as multiple copies at different time steps t, each passing the internal state to the successor (right side of the figure). The internal state h of the RNN gets updated at every time step representing the history of the network. The initial state to calculate the first hidden state h_0 is initialized with zeroes.
During the learning process of the network, three matrices are trained to adjust the networks output according to its input:

- The weight matrix W_{xh} between the input x_t at time t and the hidden layer $\mathrm{h_t}$

- W_{hh} between the hidden layer at time step $t-1$ and t
- W_{hy} between the hidden layer h_t and the output y_t

Figure 5.6 shows the matrices as arrows. The update of the hidden state and the output layer can be described by equations 5.3 and 5.4, with g and h being element wise applied activation functions.

$$h_\text{t} = g(W_{\text{hh}} * h_{\text{t-1}} + W_{\text{xh}} * x_\text{t}) \tag{5.3}$$

$$y_\text{t} = h(W_{\text{hy}} * h_\text{t}) \tag{5.4}$$

Please note that the network shares the weight parameters W_{xh}, W_{hh} and W_{hy} across all time steps to reduce the total number of parameters. The equations further describe the standard version of RNN units. This type of artificial neurons has some fundamental restrictions when applied on real-world problems, as it is not able to learn long-term dependencies (Bengio et al., 1994). In practice, the slightly different long short-term memory (LSTM) and gated recurrent units (GRU) units are mostly used (Hochreiter and Schmidhuber, 1997; Cho et al., 2014), as these types of neurons outperform the classical RNN cells through advanced update equations and additional dynamics (Chung et al., 2014). Nevertheless, the basic concept remains the same for the different RNN cells (Hochreiter and Schmidhuber, 1997).

The neural language model implements a synchronous recursive network with a many-to-many relation, as the network processes one input word at a time and predicts the most likely next word synchronously. To achieve a higher abstraction within the model, multiple recursive layers can be stacked together. The output of the recursive layer is fed into the final layer of the computational model, the output layer.

Output Layer

The final layer of the neural network encodes the result of the computation. Depending on the definition of the network parameters, the output layer varies in shape. For the language modeling task the output is the prediction of the next word within the given sequence, therefore, the output layer encodes every possible word class. Just like the input layer, the number of neurons in the output layer is defined by the vocabulary size n_{vocab}. As the output of the language model is defined as probabilities, the layer performs a softmax operation on the inputs computing the equation

$$\sigma(z_\text{t}) = \frac{e^{z_\text{t}}}{\sum_{k=1}^{K} e^{z_\text{k}}} \tag{5.5}$$

with σ representing the softmax function and $z_\text{t} = W_{\text{hy}} * h_\text{t}$ encoding the layer input as the product of output of the recursive layer and the weight matrix. The softmax

non-linearity ensures that the sum of all outputs at each time step sums up to one

$$\sum_{k=1}^{K} \sigma(z_{\mathrm{k}}) = 1 \tag{5.6}$$

The output of one class at one time step can be interpreted as the probability of the encoded word following at the next time step.

5.2.2 Binary Classification Model

The second baseline model used within this thesis is a binary supervised network. Through the automatic data source augmentation with out-of-context tokens (chapter 4), supervised models can be trained on the data. The unsupervised language model described in the previous section learns the structure of a sentence and predicts the next word solely based on the history of the sentence. In contrast to the language model learning on unlabeled data, the supervised approach uses labeled data to discriminate between the classes. Through the additional information to which class the word belongs to, the supervised model performs a binary classification with two output classes

$$\begin{aligned} original &\rightarrow 0 \\ out-of-context &\rightarrow 1 \end{aligned}$$

instead of one output class per word in the vocabulary. This change influences the architecture of the computational model as the number of neurons in the output layer of the network is reduced from 30,000 neurons down to 2. The inputs of the network and the input layer, as well as the embedding layer and the hidden layer stay unmodified. The unsupervised model definition of the language model described in the previous section is adopted accordingly to be able to perform supervised training on. Additionally, the preprocessing procedure is adopted, as the target values changed from the next word within the sequence to a binary decision on which class the current word within the sequence belongs to. With these changes performed, a new supervised model is trained.

5.3 Context Aware Model

To improve the performance on the semantic out-of-context error detection task, the narrative is taken into account. Only using information out of the current sentence, as done by the baseline systems in section 5.2, is per definition of the task not effective. A broader discourse is therefore employed to enhance the performance. The key element to efficiently retrieve the narrative out of a longer context is to encode and compress the context information effectively.

To be able to design a context aware model, the context needs to be present within the network before the first output is made. This requirement is also crucial for other tasks based on neural networks, for example neural machine translation (NMT), sentiment analysis and chatbots. All these tasks require the whole input to be encoded within the system before any output can be made. For this type of task, the so called sequence-to-sequence (also sometimes written sequence-2-sequence) or encoder-decoder models are employed as described in the paper by Sutskever et al. (2014). The model consists of an encoder part that encodes the inputs and maps them into a fixed size vector. This vector, often referred to as the thought vector, is then used to initialize the hidden state of the second part of the model, the decoder. The specific functionality executed by the decoder varies significantly depending on the given task. Nevertheless, the decoder always generates the output sequence of the network. Figure 5.7 shows the high-level structure of a sequence-to-sequence model reading the input sentence "ABC" and producing the output sequence "WXYZ".

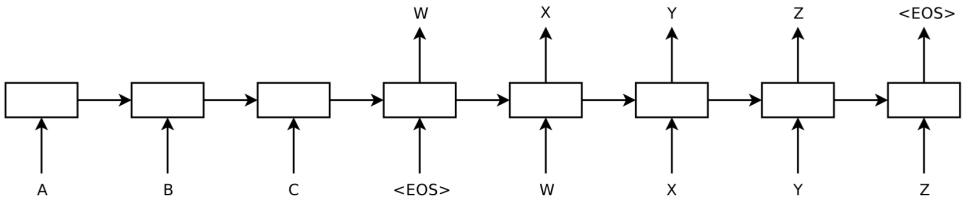

FIGURE 5.7: Sequence-to-Sequence Model Overview

In comparison to the standard model defined in figure 5.7, the architecture is changed to fit the task of context aware modeling. Mainly developed for translation systems, the standard sequence-to-sequence model takes words as inputs for both the encoder and the decoder. To input the narrative into the encoder, the model design is adopted. While the decoder stays mostly identical to the one proposed by Sutskever et al. (2014), the encoder is augmented with a hierarchical component to compress the context information. To create a proper context representation within the encoder, multiple processing steps need to be applied for the thought vector to contain all the significant information of the context. The various parts of the computational model are further discussed in the following sections.

5.3.1 Context Representation

The context representation is a crucial part of the context-aware model definition. To be able to represent the whole context in a fixed size vector requires a strong compression of the data. Throughout the compression, the vital parts of the context need to be preserved to augment the decoder of the model with the information of the complete history. To efficiently encode the information of the narrative, a three-layered approach is implemented.

An established approach to encode and compress the salient information on a word level is word embeddings, mapping sparse one-hot-encoded input tensors into dense representations. This representation is learned jointly with the model itself and learns to encode semantical meaning in the output representation. The transformation of the one-hot-encoded input representations into dense real-valued vectors applies the first layer of abstraction to the context representation. To keep track of a narrative story within a sentence, a similar encoding approach within greater context is developed. A simple lookup table, as they are used to calculate word embeddings, is not sufficient to represent a whole sentence. For the sentence representation a more complex approach is employed. Two distinct methodologies to retrieve the context representation on sentence level are used within this thesis:

- The final hidden state of a language model
- The final hidden state out of a neural machine translation system

The first method to encode a sentence is to utilize the final state of a language model hidden layer as a dense representation of the sentence. During the training of the language model, it learns which word w_i is most likely to follow a given sequence of previous tokens $\{w_1 ... w_{i-1}\}$ by keeping track of the sentence history. The history is thereby kept within the hidden state of the recursive layer, which is directly propagated into the output layer. Cutting off the output layer of the existing language model described in section 5.2.1 returns the 512 dimensional hidden state of the network instead of the 30,000 output classes. The hidden state at the last time step encodes the information of the complete sentence. Figure 5.8 shows the adopted architecture of the neural language model to retrieve the final hidden state.

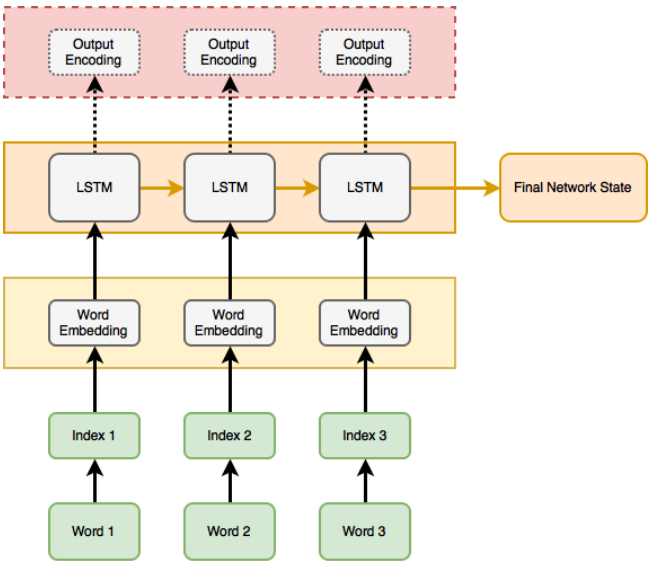

FIGURE 5.8: Language Model Context Vector Extraction

The second approach to encode the sentences is through a neural machine translation system. Based on the sequence-to-sequence network proposed by Sutskever et al.

(2014), a translation model from English to German is implemented. The model architecture encodes the English input sentence into a fixed size thought-vector before the network starts to decode the sentence in German. Through this architectural restriction, the complete information used by the decoder to output the German sentence is solely based on the thought-vector initializing the hidden state of the decoder. The entire information of the English source-sentence is therefore stored within the fixed sized vector. That makes the thought-vector of the neural machine translation system an efficient encoding of the context on a sentence level. Figure 5.9 shows the setup of the neural machine translation system.

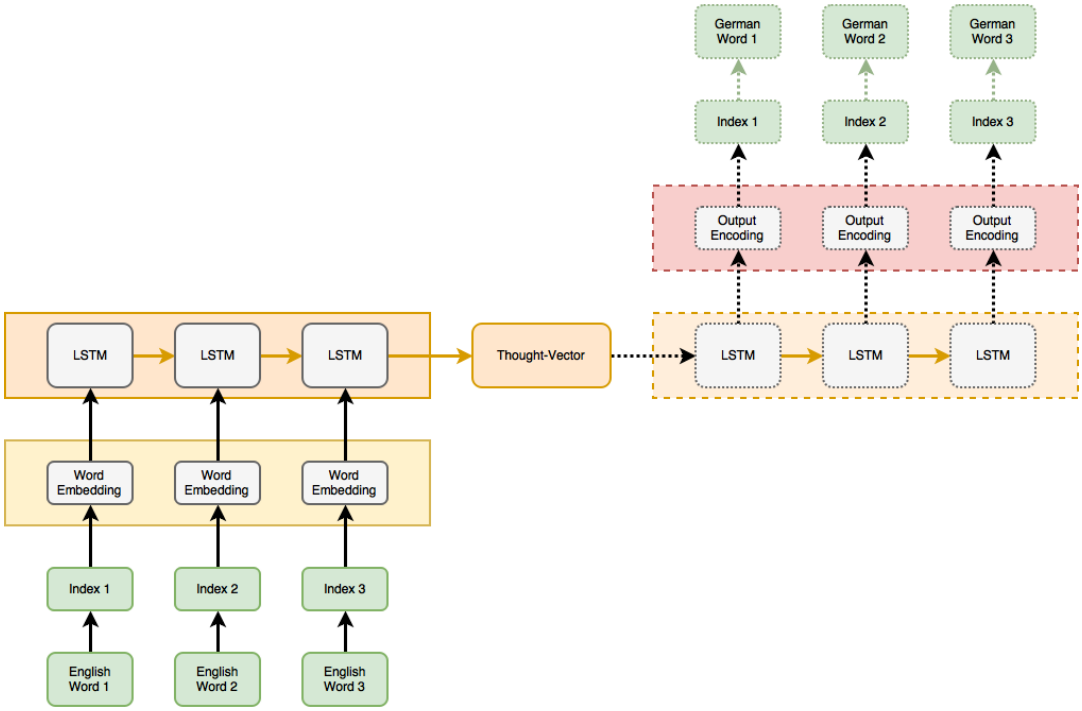

FIGURE 5.9: NMT Model Thought-Vector Extraction

During the joint training of the encoder and decoder the English source sentence is given to the encoder while the German target sentence is used as an input to the decoder. The outputs of the decoder are then trained to output the next word of the German sentence such as the neural language model in section 5.2.1.

To retrieve the context representation from the jointly trained neural machine translation encoder-decoder model, the decoder part is detached and the thought-vector is used as the output of the system as indicated in figure 5.9.

5.3.2 Context Aware Encoder

With the compressed sentence representations encoded in a 512 dimensional real valued tensor within the range $[-1.0, 1.0)$, the sequence-to-sequence encoder can be fed

with the most salient features of the narrative. In comparison to standard sequence-to-sequence models used for neural machine translation and chatbots, the architecture of the model is modified. As the standard encoder-decoder models take text as the inputs for both the encoder and the decoder, the architecture is adopted to use the hierarchical context representations. Hence, the model is able to handle multidimensional inputs. The complete three-layered hierarchy to encode the context is shown in figure 5.10.

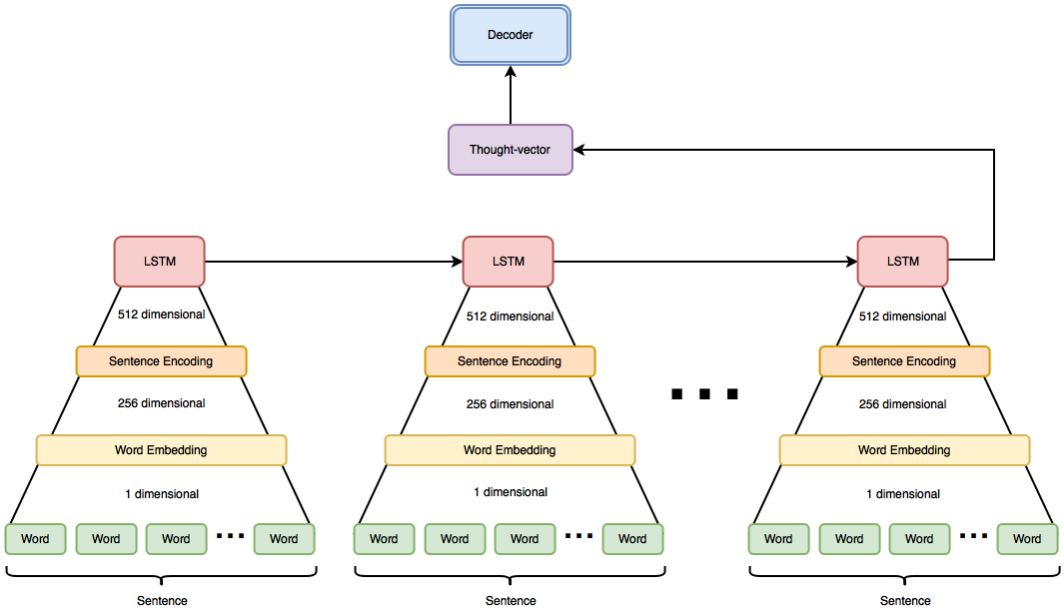

FIGURE 5.10: Hierarchical Context Representation

With this approach, the narrative story is highly compressed and efficiently encoded into a 512 dimensional thought-vector. The hierarchical methodology of the model ensures that the most salient features on every level of abstraction are merged together into one fixed-size representation. Initializing the decoder with the thought-vector representing the narrative story enables the computation to adapt to the given context.

5.3.3 Context Aware Decoder

The decoder within a sequence-to-sequence model is initialized by the hidden state of the final encoder time step. The computation is thereby triggered through the completion of the encoder calculation. This asynchronous design ensures that all the information from the encoder, in this case the whole narrative context representation, is available within the system before the decoder starts. Through this design choice, the architecture is superior for tasks that require a set of initial information to perform computations.

While some tasks (e.g. translation) need to define different computational phases during training and testing (mostly referred to as *training*- and *infer-phase*), the computation for the task of context comprehension by semantic error detection remains unmodified for both phases. The structure of the decoder is highly dependent on the specific task. Nevertheless, the overall encoder-decoder model has a well-defined overall topology: The encoder calculates the context vector from the multidimensional narrative input of the former narrative and the decoder computes a prediction for the current sentence.

With this approach, to first encode the narrative and then analyze the current sentence including that information, the model can extract semantic meaning out of the context. Furthermore, the sequence-to-sequence approach possesses an additional helpful characteristic for the specific task of narrative-based word prediction through the independence of the encoder and decoder inputs. This allows the encoder to take complex sentence encoding as an input while the decoder inputs the sentence. An high-level overview of the complete architecture is shown in figure 5.11.

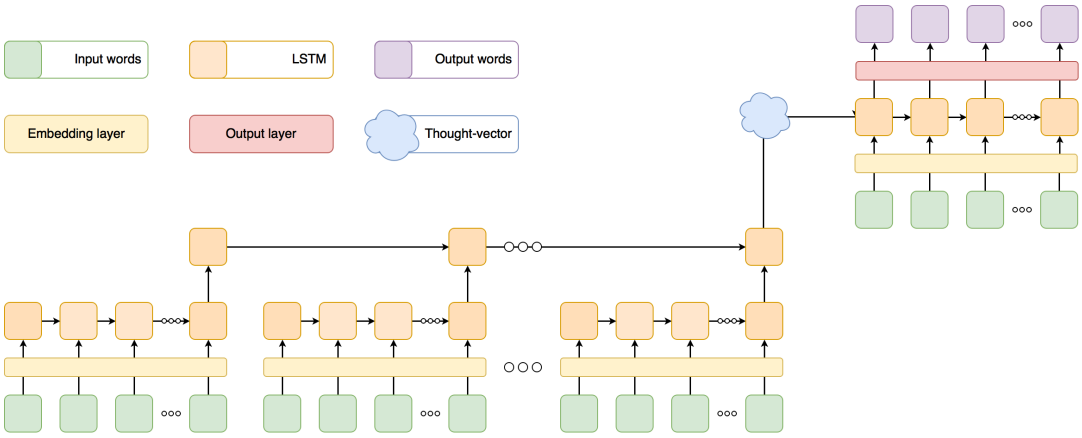

FIGURE 5.11: Full Encoder-Decoder Model Design

Chapter 6

Evaluation

In the following chapter, the computational models are evaluated. At first, every model is individually assessed on the development set. The best performance on the measures is subsequently evaluated on the test set. Whenever possible, the results are normalized by the word unigram probability. The methodologies to evaluate the models are illustrated in the following section. Afterwards, the results of the individual models are shown. An overall comparison is drawn at the end of the chapter.

6.1 Measures

To assess the performance of the models on the text understanding task, every model is evaluated on the newly introduced task defined in chapter 4. For the unsupervised models, the non-task-specific perplexity measure is additionally assessed.

6.1.1 F-Score

The F-score is a common way to evaluate computational models on the binary sequence labeling problem. The score measures the systems' performances by considering the precision p and the recall r defined by

$$p = \frac{tp}{tp + fp} \tag{6.1}$$

$$r = \frac{tp}{tp + fn} \tag{6.2}$$

where tp is the true positive labeled results, fp is the false positive labeled results and fn is the false negative labeled results. The F-score is calculated by

$$F = 2 * \frac{p * r}{p + r} \tag{6.3}$$

and represents the weighted average of its two factors, precision and recall (F-Score Definition).

The newly introduced instance of a binary sequence labeling problem is defined as the task to measure the ability of a computational model to discriminate words according to the context relevance. Therefore, every token W in the data sequence is assigned a binary label $L_{\text{ooc}} \in \{0, 1\}$, with 0 representing valid-context tokens and 1 indicating out-of-context tokens.

To evaluate the performance of the models, the probability of every word $w_{\text{r}} \in W$ is compared to the likelihood of every other word $w_{\text{q}} \in W$, with $r \neq q$. The words in W are subsequently ordered according to the probability within the sequence. To normalize the data, the predicted probabilities are additionally divided by the unigram probability.

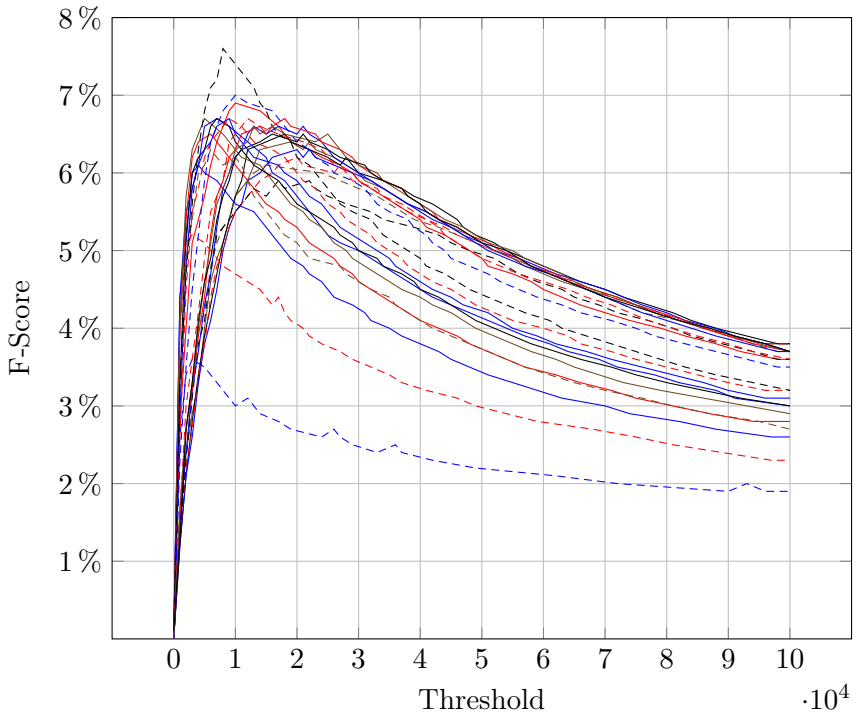

FIGURE 6.1: Unfiltered F-Score Example

A perfect system would assign the lowest probabilities to the semantic out-of-context tokens. As the task definition does not give the number of replacements, the amount of out-of-context tokens is unknown. However, to compare the models, a dynamic threshold $thres_{\text{range}}$ for the separation between the original words and the replaced tokens is utilized. The best separation threshold is evaluated by analyzing the F-score for every possible threshold on the development set. For the TEDTalk development set with 1 million word tokens, the threshold parameter is set to $range = [1..1,000,000]$ with a step size of 100. For the purpose of readability, the displayed F-scores are pre-filtered for the best generation of every model instance and limited to the least likely 100,000 words. Figure 6.1 shows an example with unfiltered data.

6.1.2 Perplexity

To measure the performance of the model, the perplexity is calculated for various epochs on the development set. The metric measures how efficiently the language model predicts unseen data and is defined as

$$PP(W) = P(w_1 w_2 ... w_{\mathrm{N}})^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_{\mathrm{N}})}} \tag{6.4}$$

The base equation for the perplexity $PP(W)$ of the word W within a sequence of length N is defined in equation 6.4. For a coherent sequence of words, the probability of every word is dependent on the previous sequence. The inverse probability $\frac{1}{P(w_1, w_2 ... w_{\mathrm{N}})}$ can therefore be transformed into $\prod_{i=1}^{N} \frac{1}{P(w_{\mathrm{i}}|w_1 ... w_{\mathrm{i\text{-}1}})}$ as illustrated in equation 6.5.

$$\sqrt[N]{\Pi_{i=1}^{N} \frac{1}{P(w_{\mathrm{i}}|w_1 ... w_{\mathrm{i\text{-}1}})}} \tag{6.5}$$

The probability of the word w_{i} defined by $P(w_{\mathrm{i}}|w_1 ... w_{\mathrm{i\text{-}1}})$ is calculated by applying the neural language model on the sentence $\{w_1, w_2, ..., w_{\mathrm{i\text{-}1}}, w_{\mathrm{i}}\}$. The output of the calculation is the probability distribution at time step i. The product $\prod$ in equation 6.5 multiplies the inverse probabilities of all words within the sequence. This leads to very large values for the product $\prod$ when dealing with longer sequences. To enhance the numeric stability of the computation, the calculation is transferred into the log-space by extending the function with $2^{log_2(PP(W))}$, as shown in equation 6.6.

$$2^{log_2 \sqrt[N]{\Pi_{i=1}^{N} \frac{1}{P(w_{\mathrm{i}}|w_1 ... w_{\mathrm{i\text{-}1}})}})} \tag{6.6}$$

Utilizing the logarithm rules, the product $\prod$ is converted into a sum $\sum$, by moving the log into the product (Equation 6.7).

$$2^{\sum_{i=1}^{N} log_2(P(w_{\mathrm{i}}|w_1 ... w_{\mathrm{i\text{-}1}})^{-\frac{1}{N}})} \tag{6.7}$$

Applying the power rule to the exponent adds $-\frac{1}{N}$ as a factor in the sum as shown in equation 6.8.

$$2^{\sum_{i=1}^{N} -\frac{1}{N} * log_2(P(w_{\mathrm{i}}|w_1 ... w_{\mathrm{i\text{-}1}}))} \tag{6.8}$$

The factor $-\frac{1}{N}$ is constant within the sequence modeled by the sum $\sum$ and can thus be pulled out of the summation (Equation 6.9).

$$2^{-\frac{1}{N} * \sum_{i=1}^{N} log_2(P(w_{\mathrm{i}}|w_1 ... w_{\mathrm{i\text{-}1}}))} \tag{6.9}$$

With the perplexity measure defined in equation 6.9, the calculation is numerically stable and can be applied on the output of the computational models (Perplexity Definition).

6.2 Baseline Model

The two baseline models, described in section 5.2, are evaluated on the preprocessed evaluation dataset shown in chapter 4. First, the language model instance is trained and tested. Subsequently, the supervised binary classification network is set up and evaluated.

6.2.1 Language Model

To train and test an instance of the baseline neural language model, the hyper-parameters of the network need to be defined. Hyper-parameters are referred to as the parameters of the computational model and the training process itself. For many of the free parameters, extensive studies have been conducted on general tasks, to achieve the optimal performance. Nevertheless, through differences in the network topology and special requirements for certain tasks, the best set of hyper-parameters is not necessarily the combination of the best individual parameters. Thus, many of the parameter combinations require careful adoptions to achieve the best results.

One of the most important hyper-parameters during the training of the neural language model is the loss function. With a given label y_i for a data point representing the ground truth and the output of the neural network $f(x_\mathrm{i}|\theta)$ for the input $\mathrm{x_i}$ at time step t and the neural network θ, the loss function measures the error of the label-output combination. The loss used for the training of the network is defined by

$$loss(f(x_\mathrm{i}|\theta), y_\mathrm{i}) = ln(f(x_\mathrm{i}|\theta)) * y_\mathrm{i} \tag{6.10}$$

calculating the product of the natural logarithm of the network output and the ground-truth. As the loss is defined per data point, it is used within a more general cost function, summarizing multiple loss functions over a greater part of the training set. The number of data points that are combined together is defined by the batch size. The cost function used for the model is the cross entropy cost function defined by

$$\varepsilon = -\frac{1}{n}\sum_{i=1}^{N} loss(f(x_\mathrm{i}|\theta), y_\mathrm{i}) = -\frac{1}{n}\sum_{i=1}^{N} ln(f(x_\mathrm{i}|\theta)) * y_\mathrm{i} \tag{6.11}$$

The cost function summarizes N data points by calculating the sum of the respective loss functions. To normalize the cost of the network, the result of the summation is divided by the negative amount of data points $-\frac{1}{n}$. This error value ε is the input for the optimizer of the model that is used to minimize the cost function through back propagation of the error into the network.

Various functions have been developed to solve the optimization task. Most of the modern optimizers are momentum based including the Adagrad, AdaDelta, and Adam algorithms. The Adam optimizer has multiple advantages compared to the

other learning methods and is proven to work very well in practice. It outperforms the other algorithms through its fast learning speed, leading to a rapid converge. It also avoids common optimizer problems such as high variance and vanishing learning rate (Kingma and Ba, 2014). The nature of the Adam optimizer decrements the learning rate throughout the training process to improve the performance of the network and avoid local minima. Even though the learning rate is dynamically adopted over time, the initial value of the learning rate parameter still has crucial impact on the optimizer's performance and needs to be carefully chosen. For the language model, the learning rate has been set between $1e-3$ and $1e-4$ where a rate of $1e-3$ is recommended by the original authors (Kingma and Ba, 2014).

Another important hyper-parameter of the network is the number of generations. Depending on the depth and width of the network, the necessary epochs until the network converges varies greatly. The number of iterations is a trade-off between network size and computation time. As most of the models evaluated within this thesis have been empirically proven to converge within 16 iteration and the computational effort during the training remains reasonable, the models are trained for 16 iterations.

Through computational constraints of the models, further assumptions need to be made. One important parameter to establish is the vocabulary size n_{vocab}. While the size of the vocabulary has been kept flexible during the model definition, it needs to be defined for the instance of the network. As described in the previous chapter, the size of the vocabulary corresponds to the number of classes and defines the dimension of the input and output layer of the computational model. This direct dependency between the vocabulary size and the dimensions of the network makes the vocabulary size an important factor. If n_{vocab} is chosen very small, the performance of the system will decrease as many words are substituted by the <UNK> token. Setting the number of classes disproportionately high will significantly slow down the training process. The size of a reasonable vocabulary is also highly dependent on the data source and can range from a few thousand words (Bengio et al., 2003) up to a million (Mikolov et al., 2013) unique tokens in the vocabulary. For the task of language modeling on the TEDTalk dataset the vocabulary size n_{vocab} is set to 30,000 unique words resulting in 30,000 classes.

Aside from the limitation of possible input words though the vocabulary size, the maximal length of a sequence is defined. For language models one sentence traditionally represents a sequence. Thus, the length of the average sentences determines the maximal sequence size of 50 tokens.

The sparse input of the computational model defined by the vocabulary size is internally transferred into a dense representation through the embedding layer as shown in figure 5.4. Therefore, the second dimension of the embedding layer defined by the matrix of dimension $[n_{\text{vocab}} * size_{\text{embedding}}]$ has a fundamental impact on the network. The input of the embedding layer defined by n_{vocab} is transferred into representations

in the real valued space. This representation must yield enough entropy to sufficiently discriminate the various input classes. A suitable size for the embedding has been determined at around 300 dimensions (Xie and Rastogi, 2017).

The recursive layer plays a crucial role for the memory capacity of the network. Hence, the neurons within the layer need to be able to store enough information of the previous time steps. This makes the number of neurons within the layer a key feature of the network, which significantly determines its performance. 512 neurons per hidden layer have been established an efficient compromise amongst memorization and performance.

An overview of the hyper-parameters is shown in table 6.1.

Parameter	Model value
Optimizer	Adam
Cost Function	Cross Entropy
Learning Rates	[1e-3, 5e-4, 1e-4]
Iterations	16
Vocabulary Size	30,000
Sequence Length	50
Embedding Size	256
Hidden Layer Size	512
Hidden Layers	1
Batch Size	100

Table 6.1: Language Model Instance Hyper-Parameters

Further selected hyper-parameters of the computational model include the number of hidden layers and the batch size. With a larger number of hidden layers in the model, a higher degree of abstraction can be achieved at the cost of additional network parameters, which need to be learned during training. The amplification of the model slows down the learning process and requires a larger set of training data. Therefore, adopting the number of layers significantly changes the networks' performance during training and testing. By changing the number of hidden layers in the network and the resulting increase of free parameters, the batch size is adopted accordingly. Adding one additional layer of computation with the hyper-parameters described in table 6.1 introduces an extra $(512^2 + 512)$ weights and 512 further neuron states that need to be kept in memory.

All parameters within one batch need to be accessible in memory as the update of the model is performed on batch-level by applying the cross entropy cost function and propagating the error back into the network through the optimizer. These computational steps can easily exceed the systems' memory limitations. Adopting the batch size is a common way to avoid this limitation without reducing the amount of computational layers.

The model with the hyper-parameters described in table 6.1 is trained on the modified and preprocessed TEDTalk data.

Figures 6.2 and 6.3 show the results of the evaluation for the baseline language model on the development set. The first graph (6.2) displays the F-score of the neural networks on an increasing threshold range. Figure 6.3 shows the measured perplexity on the development and training set.

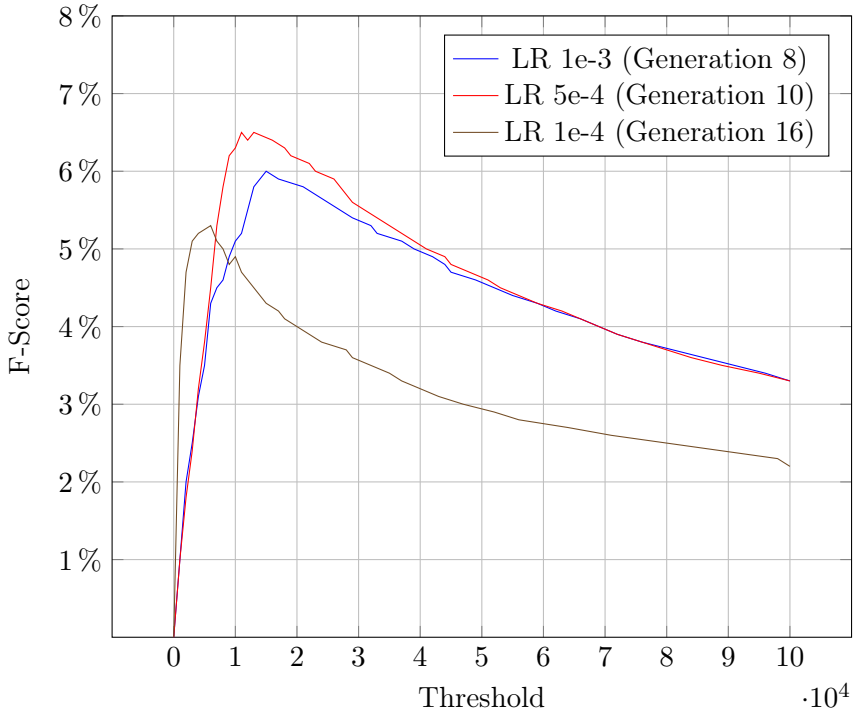

FIGURE 6.2: Baseline Language Model F-Score

Based on the analysis of the two measures, iteration 10 of the model with learning rate $5e-4$ reaches the best performance with a F-score of 6.49% on the development set. Hence, this model is selected for the final evaluation of the unsupervised baseline model. The final result on the test set is shown in table 6.2.

Measure	Model Performance
Precision	4.12%
Recall	15.42%
F-score	6.51%
Perplexity	115

TABLE 6.2: Baseline Language Model Final Result

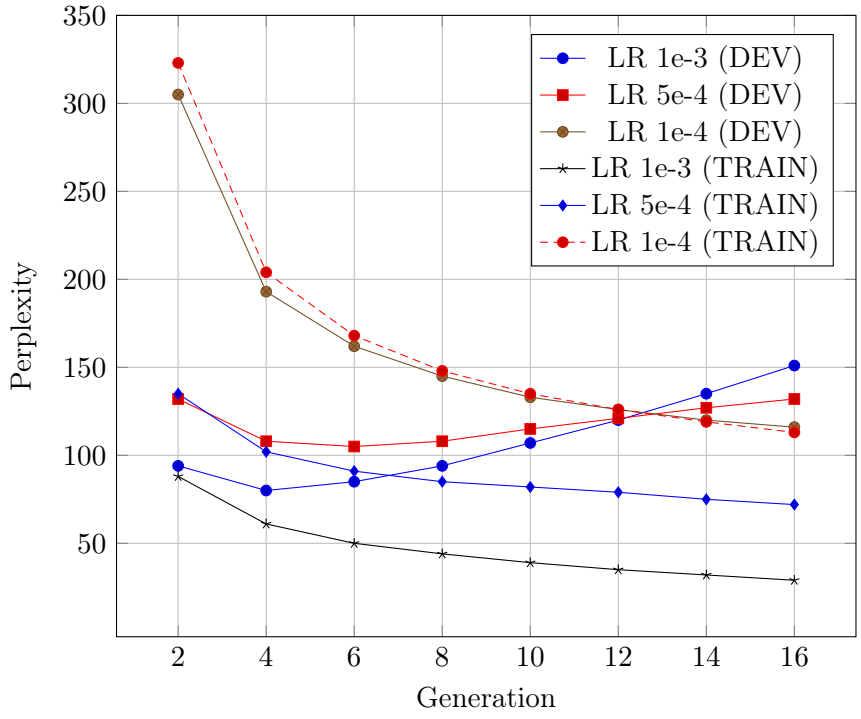

FIGURE 6.3: Baseline Language Model Perplexity

6.2.2 Binary Classification Model

The F-score of the supervised computational model is shown in figure 6.4 for the three distinct learning rates $1e-3$, $5e-4$ and $1e-4$.

The best performance is achieved in generation 8 utilizing the model with a learning rate of $1e-3$ by separating the binary classes at the least likely 5,000 tokens. The resulting F-score on the development set is 10.16%. The respective model with the best performing 8^{th} generation is therefore evaluated on the test set, specifying the maximal performance of the supervised computational model on the task. The results are shown in table 6.3.

Measure	Model Performance
Precision	8.94%
Recall	11.76%
F-score	10.16%

TABLE 6.3: Baseline Binary Classification Model Final Result

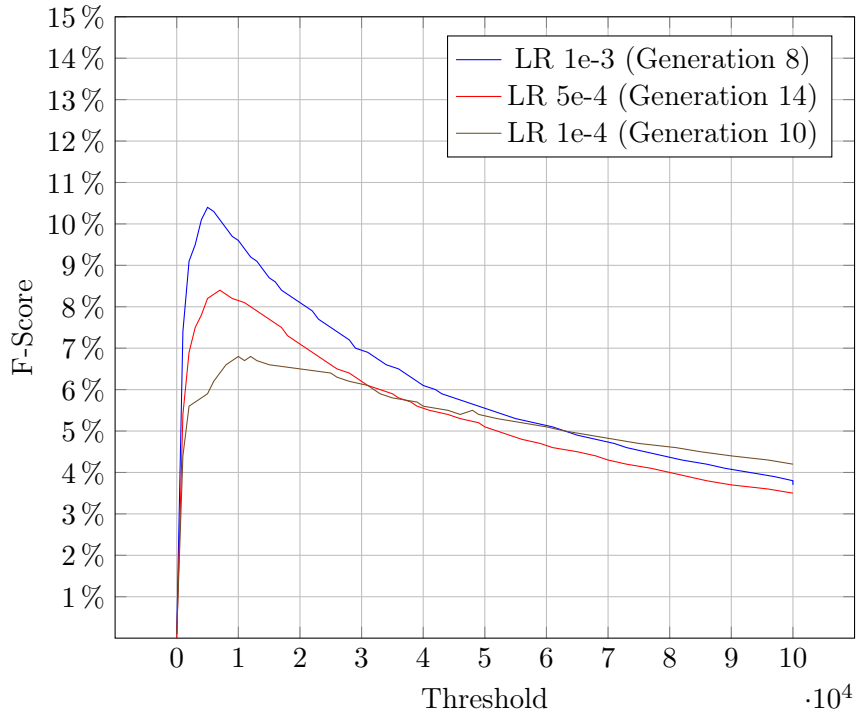

FIGURE 6.4: Baseline Binary Classification Model F-Score

6.3 Context Aware Model

To evaluate the influence of the broader context on the task of out-of-context error detection, three different context-aware model topologies are instantiated. The computational models employed are:
(1) A context-aware language model,
(2) an attention-based context-aware language model and
(3) a supervised context-aware binary model

Each model architecture employs two different context representations as described in section 5.3.1.
(a) The final state of a language model (LM-CR) and
(b) the thought-vector of a translation model (NMT-CR).
The first context representation is based on the best performing language model described above. The second approach, utilizing the NMT model, is shortly analyzed in the following section.

6.3.1 NMT Context Representation

With the NMT model architecture described in section 5.3.1, the translation model is trained for 16 generations with three different learning rates of $1e-3$, $5e-4$ and $1e-4$. Table 6.4 summarizes the free hyper-parameters used for the training. The parameters are chosen as close as possible to the parameters used within the language model, to keep the two context representations comparable.

Parameter	Model value
Optimizer	Adam
Cost Function	Cross Entropy
Learning Rates	[1e-3, 5e-4, 1e-4]
Iterations	16
Vocabulary Size (English)	30,000
Vocabulary Size (German)	30,000
Sequence Length	100
Embedding Size	256
Hidden Layer Size	512
Hidden Layers	1
Batch Size	100

Table 6.4: NMT Model Hyper-Parameters

The result of the bilingual training is shown in figure 6.5, displaying the average perplexity of the learned translation model tested on the mutually exclusive development set. The best perplexity is achieved by the model with a learning rate of $5e-4$, reaching a minimal perplexity of 46 in generation 8. Due to overfitting, the

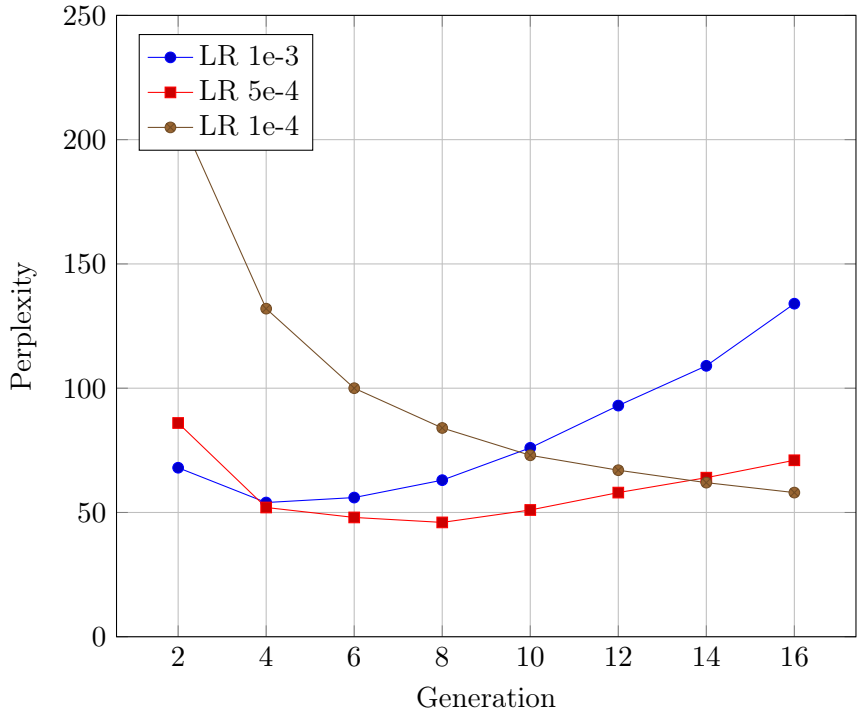

FIGURE 6.5: NMT Model Perplexity

perplexity increases in later iteration again. Using the described model represents a good trade-off between training efforts and performance. The final perplexity of the translation model evaluated on the test set is 49.

6.3.2 Context Aware Language Model

The context-aware language model implements the standard encoder-decoder model defined in chapter 5. To be able to compare the network with the baseline language model and assess the influence of the context on the system performance, the hyper-parameters of the context-aware system are chosen similar to the baseline model's parameters. The utilized hyper-parameters are described in table 6.5.

To create a reasonable trade-off between training time and model performance, the additional context parameter is restricted to 10 sentences. Based on these parameters, the context-aware language model is instantiated and trained. The exact topology of the neural network is shown in figure 6.6 created with the Tensorflow visualization tool Tensorboard.

The Tensorboard visualization of the network shows the data flow through the computational graph. The topology includes four input placeholders for the data to be fed into the network during training and testing.

The two inputs for the encoder are the *sentence representations* and the *context length*, which are entered into the network at every time step. The dimension of the

Parameter	Model value
Optimizer	Adam
Cost Function	Cross Entropy
Learning Rates	[1e-3, 5e-4, 1e-4]
Iterations	16
Vocabulary Size	30,000
Sequence Length	50
Context Length	10
Embedding Size	256
Hidden Layer Size	512
Hidden Layers	1
Batch Size	100

TABLE 6.5: Context Aware Language Model Hyper-Parameters

sentence representation data is

$$[batch\ size \times context\ length \times sentence\ representation] \tag{6.12}$$

To keep the batch size flexible between training and testing, the *batch size* dimension stays undefined and is determined during the execution of the network. This is possible, as the two other dimensions are defined and the *batch size* dimension can be dynamically inferred from these. Applying the hyper-parameters of the network, described in table 6.5, the input for the *sentence representations* is of dimension $[?, 10, 512]$. The *context length* is represented by a one-dimensional tensor of size [*batch size*] and contains the context length of every input (*sentence representation*) into the network. This information is crucial for the computational graph, as the flexible length of the context needs to be available to the network, to stop the computation after the context is fully fed into the network. This procedure is necessary rather for the correctness of the system than for the performance (Tensorflow Documentation). With the *sentence representation* and the *context length* defined, the encoder can dynamically compute the overall context of the system and pass the resulting thought-vector into the decoder part of the network.

The decoder inputs are the *target sentence* and the *sentence length* with the *target sentence* input having a dimensionality of

$$[batch\ size \times max\ sentence\ length] \tag{6.13}$$

representing the current sentence evaluated within the context. The *batchsize* dimension is undefined to allow different batch sizes during training and testing and the *max sentence length* dimension is of size 50 resulting in the input dimensions of $[?, 50]$. Every value in the tensor is an integer representing the position of the word in the one-hot encoded vector. This index-encoded input representation is subsequently

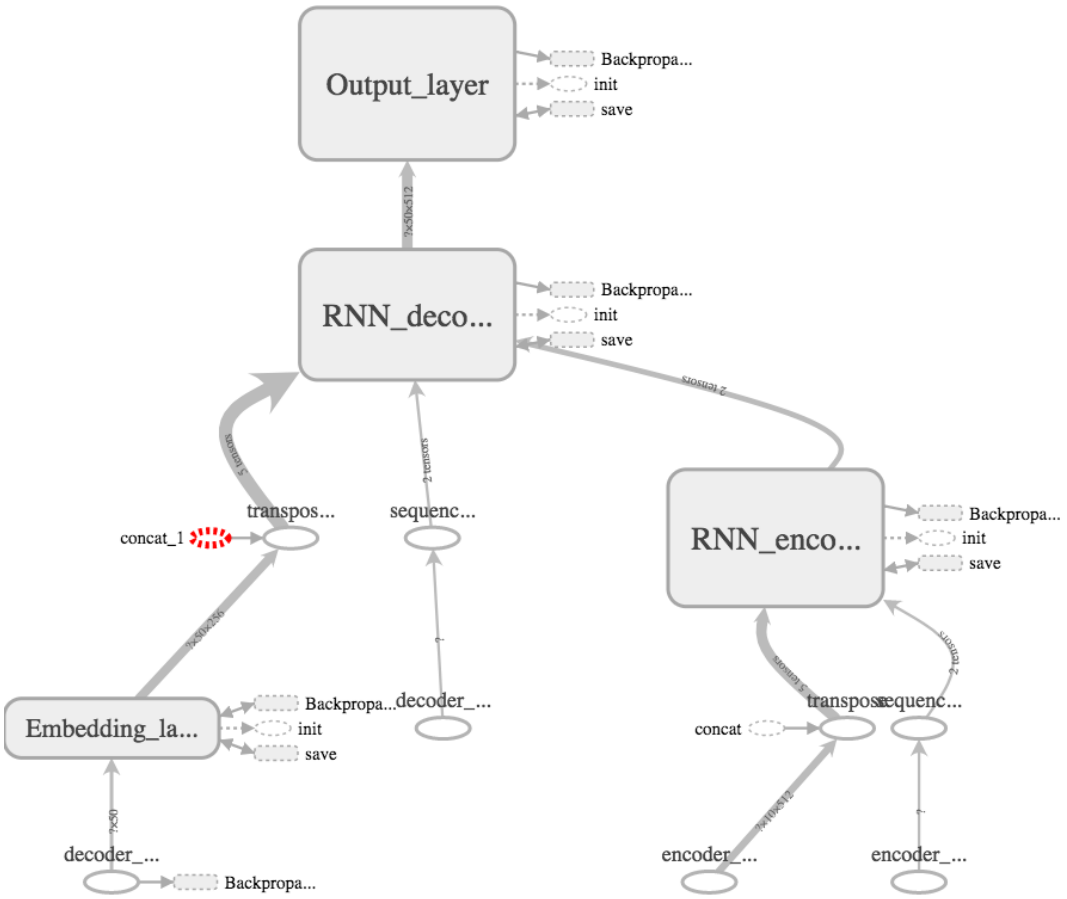

FIGURE 6.6: Context Aware Language Model Topology in Tensorboard

entered into the embedding layer. The embedding layer maps the $[?, 50]$ dimensional integer-valued input into a $[?, 50, 256]$ dimensional space of real values. This transformation does not affect the first two dimensions of the tensor. The second input of the decoder is the *sentence length*, which defines the length of every sentence input, with the dimensions [*batch size*]. This does not contradict with the intermediate embedding computation, as the embedding layer does not influence the first two dimensions. Together with the thought-vector from the encoder, the decoder computation is executed for every word in the *target sentence* and forwarded to the output layer through a tensor of dimension $[?, 50, 512]$ containing the output of the hidden units.

The models are trained on the TEDTalk training data for 16 iterations and subsequently evaluated on the development set to determine the best performance of the computation. The included measures for the context-aware language model on the development dataset are the perplexity and the normalized out-of-context detection

rate.

The results of the network utilizing the final state of the language model (LM-CR) to calculate the sentence representations are shown in figures 6.7 and 6.8.

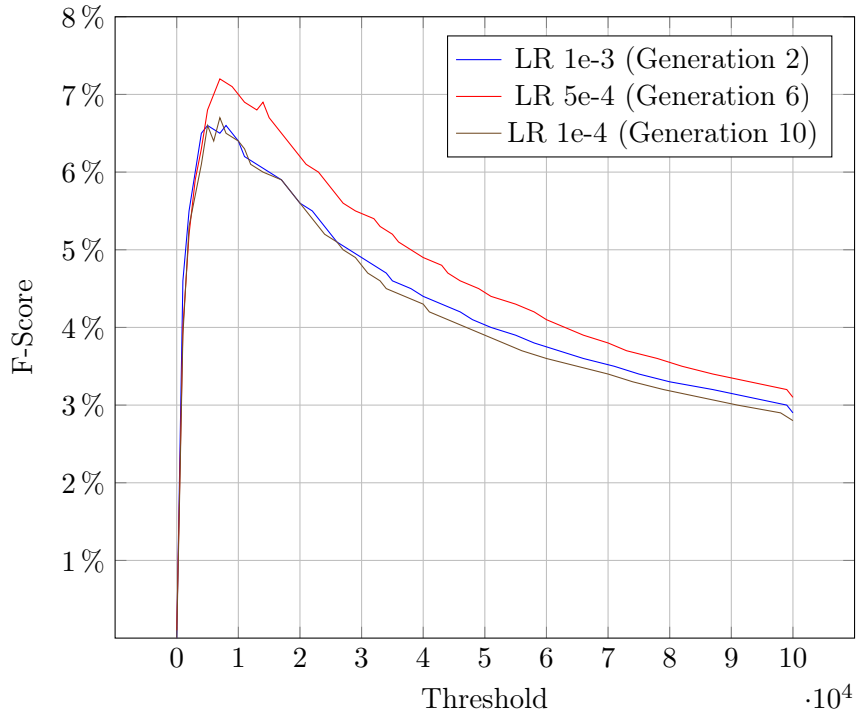

FIGURE 6.7: Context Aware Language Model F-Score (with Neural Machine Translation Context Representation)

The F-score evaluation in figure 6.7 reaches the best performance of 7.21% separating the least likely 7,000 words with a learning rate of $5e-4$. The perplexity graph (6.8), comparing the three learning rates, confirms the suitability of the model with a learning rate of $5e-4$ with a minimal perplexity of 74. With these two individual measures, the best performing network is determined.to be the network with a learning rate of $5e-4$, as it shows good performance on both tasks. The model is therefore tested on the test set for the final score (Table 6.6).

Measure	Model Performance
Precision	5.18%
Recall	11.05%
F-score	7.06%
Perplexity	79

TABLE 6.6: Context Aware Language Model (with Language Model Context Representation) Final Result

The results of the context-aware language model using the final state of the baseline language model to calculate the sentence representations, as described above, is

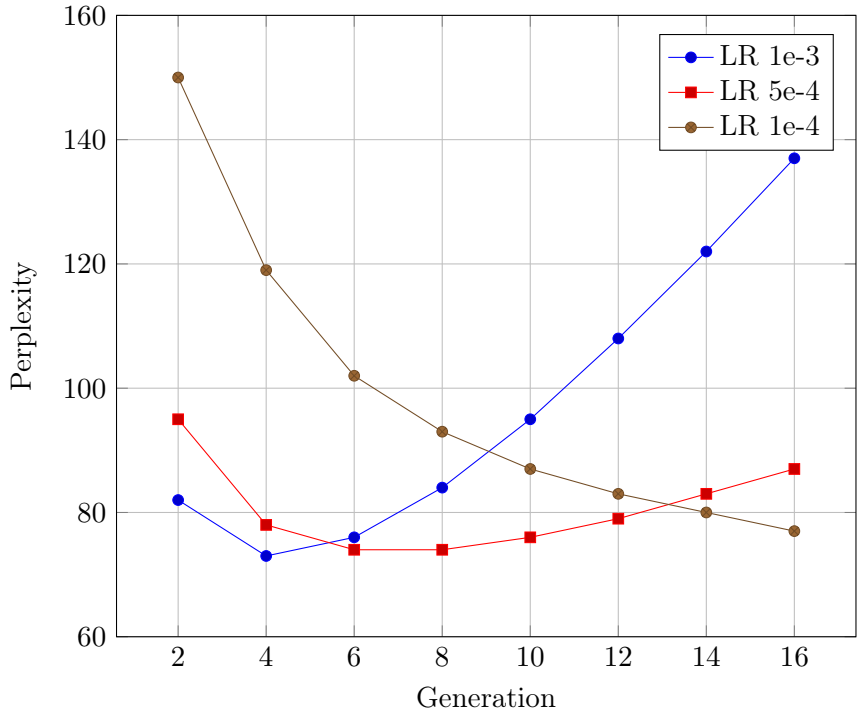

FIGURE 6.8: Context Aware Language Model Perplexity (with Language Model Context Representation)

compared to the result utilizing the translation model to encode the sentence representations. Therefore, the same evaluation is executed on the development set to find the best learning rate and training generation.

In the following, figure 6.9 describes the F-scores of the three networks, analyzing the abilities on the out-of-context error detection task. The second graph (6.10) subsequently described the perplexities.

With a separation value of 8,000 tokens on the development dataset, the best F-score is achieved by the model with a learning rate of $5e-4$ in generation 6 (7.55%). The shape of the perplexity graph in figure 6.10 also shows a good perplexity of 73 for the model with a learning rate of $5e-4$. Building on these results, the best combination of learning rate and training generation is determined to be generation 6 of model $5e-4$. The results of the model tested on the final test set are shown in table 6.7.

Comparing the two final outcomes on the test set described in tables 6.6 and 6.7, the approach utilizing the underlying translation model outperforms the baseline language model. While the performance on the development set is on the same level for both models, the result on the test set indicates an advantage of the translation based approach. It needs to be additionally noted, that the translation approach had less data to be trained on, as the approach requires multilingual training data.

The best result on the task utilizing the context-aware translation model is a

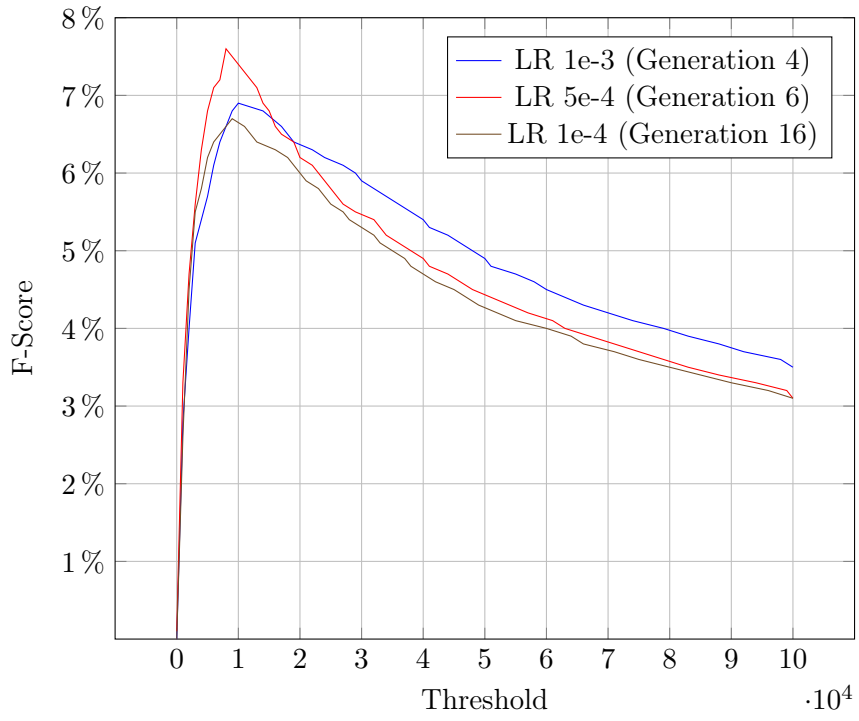

FIGURE 6.9: Context Aware Language Model F-Score (with Language Model Context Representation)

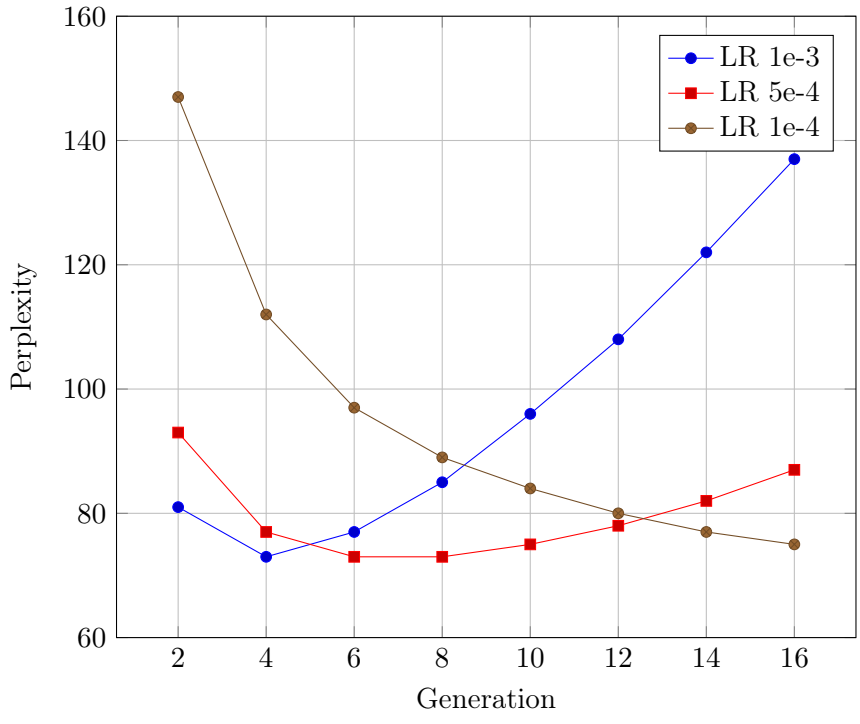

FIGURE 6.10: Context Aware Language Model Perplexity (with Neural Machine Translation Context Representation)

Measure	Model Performance
Precision	5.43%
Recall	11.31%
F-score	7.34%
Perplexity	76

TABLE 6.7: Context Aware Language Model (with Neural Machine Translation Context Representation) Final Result

detection rate of 7.34%. Compared to the performance of the baseline model with a test result of 6.51%, the detection rate has been increased by 0.83% absolute and 12.75% relative. This shows, that the context-aware extension of the model enhanced the performance on the semantic error detection task.

6.3.3 Attention-based Context Aware Language Model

The second context-aware language model employs an additional computational concept to extend the encoder-decoder model by introducing an attention mechanism. Attention has been developed to free the encoder-decoder design pattern from the fixed-length thought-vector by training the model to directly pay attention to the input sequence of the encoder instead of only relying on the thought-vector to encode all the relevant information. With this new concept, the output sequence can conditionally select relevant information from the input sequence. This approach has proven to enhance many systems within the domain of NMT and became state-of-the-art for translation systems (Bahdanau et al., 2014; Luong et al., 2015). Improving the performance of encoder-decoder models for translation purposes makes this approach also interesting for our task of context-aware error detection. The extended computational graph employing an attention module is displayed in figure 6.11.

Instead of only passing the thought-vector from the encoder to the decoder, the attention module is interconnected. The additional attention computation takes the thought-vector from the encoder as well as all the inputs at every time step into account. During the training, the attention module jointly learns which input to focus on during the execution. Due to the conditional attention on the encoder outputs, the network is supposed to be able to further discriminate important content of the source context from unstructured clutter. The trade-off for these additional computations are further parameters, which need to be trained during the learning process. To remain comparable, the hyper-parameters of the system are derived from the baseline models. The used hyper-parameters are illustrated in table 6.8.

The additional attention parameters are chosen to fit in the existing network topology with an attention size equal to the context length and a single layer of attention. The attention algorithm is chosen to be the Bahdanau attention.

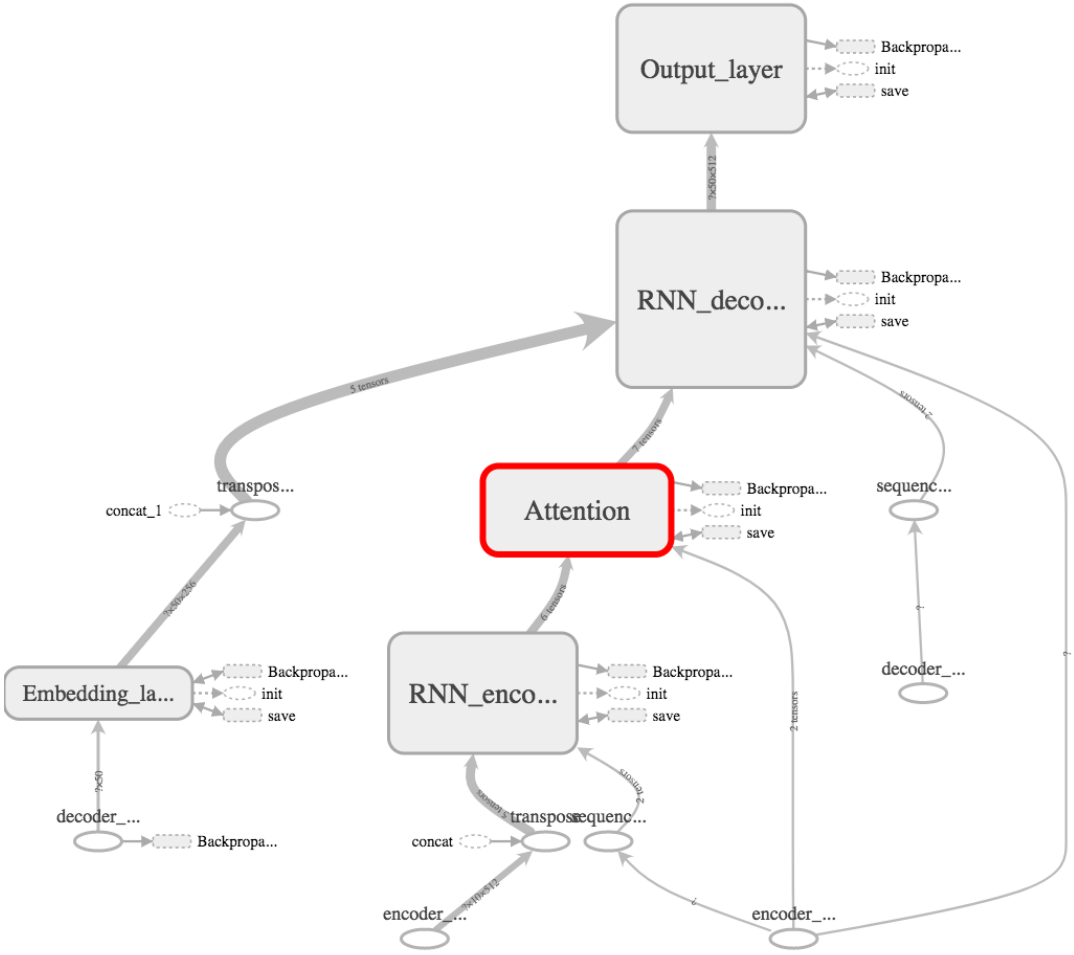

FIGURE 6.11: Attention Based Context Aware Language Model Topology in Tensorboard

With this extended architecture, the context-aware language model is trained and tested. The results on the development set are presented in figures 6.12 and 6.13 for the sentence representations with the baseline language model. Figures 6.14 and 6.15 show the results based on the translation model.

With the two measures utilizing the baseline language model for the sentence representations, the best combination of learning rate and training epoch are determined. The system with learning rate $5e-4$ reaches the best F-score on the out-of-context detection task in generation 10 with a threshold of 9,000 words gaining a score of 4.57%. The same model/generation combination also achieves a good perplexity for the model with 124. Iteration 10 of the $5e-4$ model is, hence, tested on the test dataset. The result of the measurement is shown in table 6.9.

Parameter	Model value
Optimizer	Adam
Cost Function	Cross Entropy
Learning Rates	[1e-3, 5e-4, 1e-4]
Iterations	16
Vocabulary Size	30,000
Sequence Length	50
Context Length	10
Attention	Bahdanau
Attention Memory Size	10
Attention Memory Depth	1
Embedding Size	256
Hidden Layer Size	512
Hidden Layers	1
Batch Size	100

TABLE 6.8: Attention Based Context Aware Language Model Hyper-Parameters

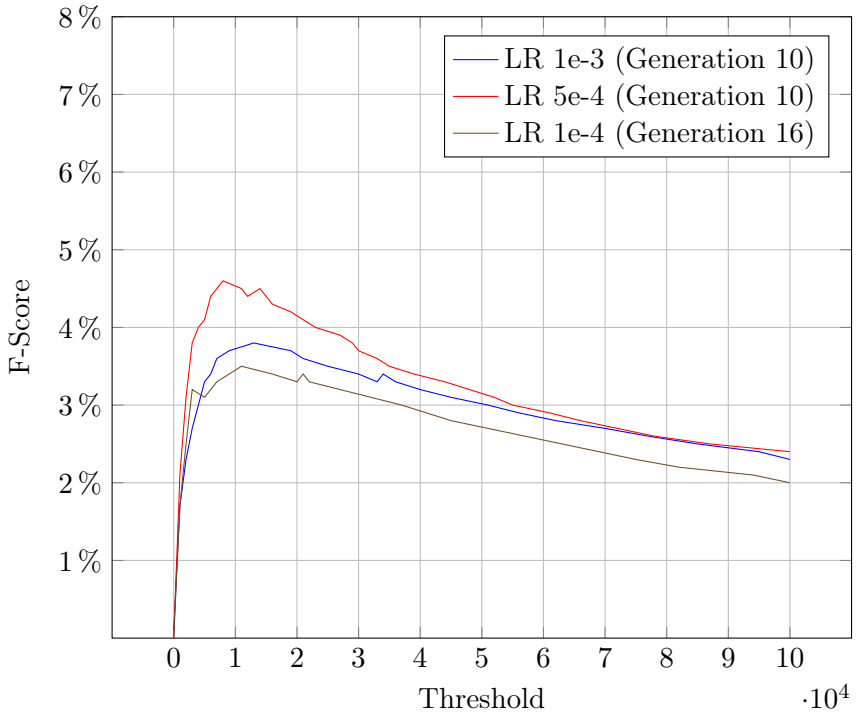

FIGURE 6.12: Attention Based Context Aware Language Model F-Score (with Language Model Context Representation)

The second approach for the attention based model again follows the context representations utilizing the translation model to encode the sentence representation. The results on the development set are described in the graphics below (Figures 6.14 and 6.15)

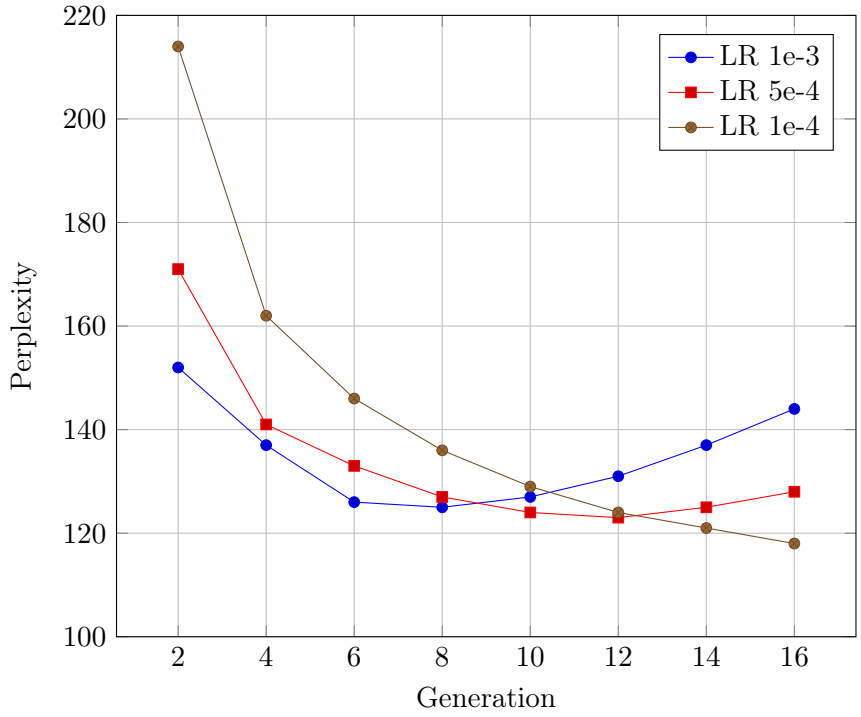

FIGURE 6.13: Attention Based Context Aware Language Model Perplexity (with Language Model Context Representation)

Measure	Model Performance
Precision	3.05%
Recall	7.32%
F-score	4.31%
Perplexity	125

TABLE 6.9: Attention Based Context Aware Language Model (with Language Model Context Representation) Final Result

Compared to the attention based model utilizing the baseline language model context representations (6.13), the F-score of the model based on the NMT representation shows significantly higher F-scores in graph 6.14. The best score is reached by the model with learning rate $5e-4$ in generation 6 with a threshold value of 10,000 (6.98%). The perplexity aligns well with the F-score, reaching a minimal value for the model with a learning rate of $5e-4$ in generation 6. The result on the test set is shown in table 6.10.

In case of the attention based context-aware language model, the sentence encoding with the translation model achieves significantly better results than the approach utilizing the baseline language model. The best performance on the test set is an F-score of 6.8%, outperforming the baseline network by 0.29% absolute and 4.46% relative. With this result, the attention based context-aware language model reaches a better performance than the baseline, but cannot enhance the performance of the standard

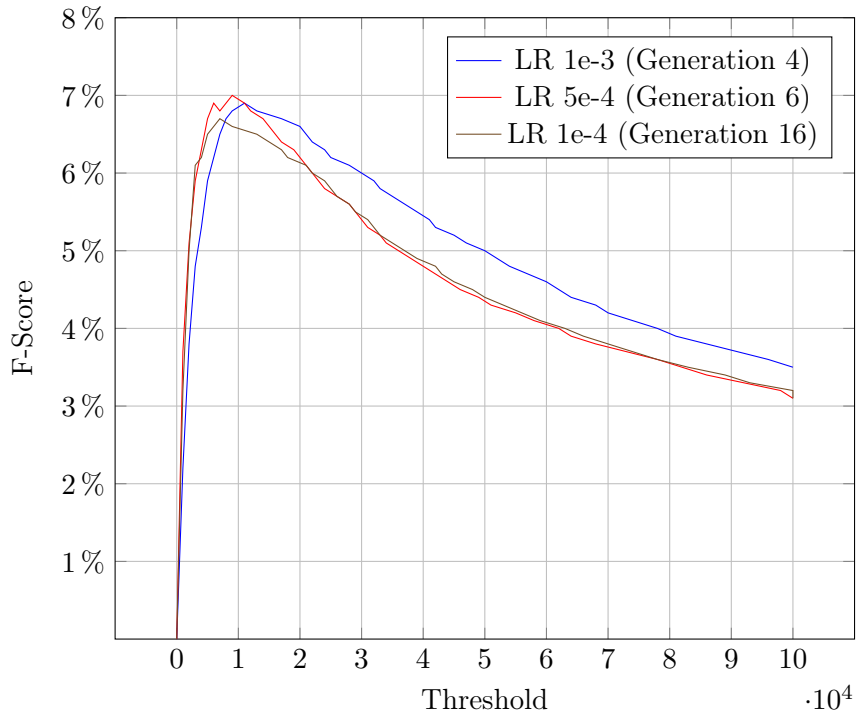

FIGURE 6.14: Attention Based Context Aware Language Model F-Score (with Neural Machine Translation Context Representation)

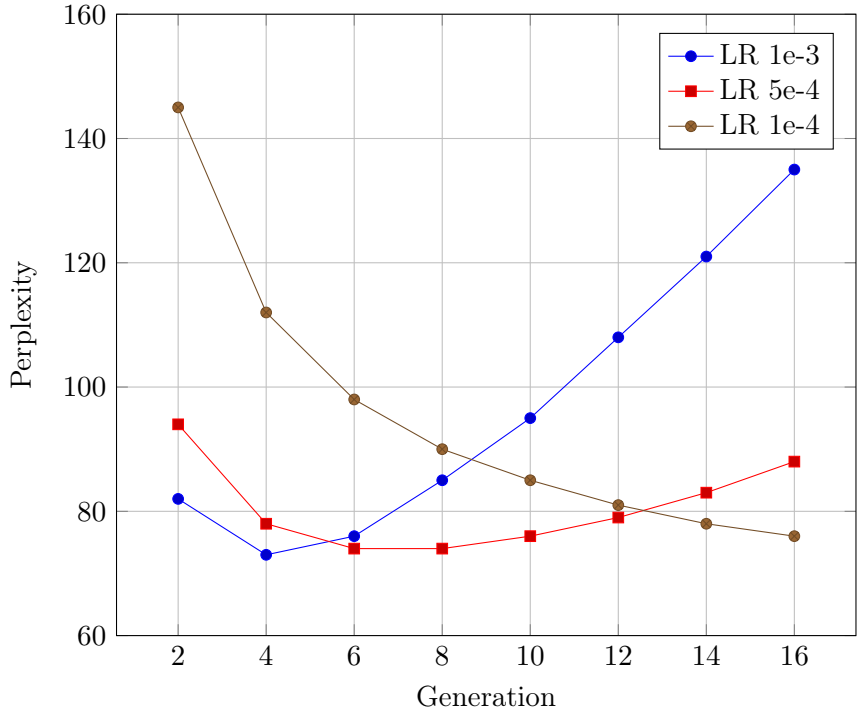

FIGURE 6.15: Attention based Context Aware Language Model Perplexity (with Neural Machine Translation Context Representation)

Measure	Model Performance
Precision	4.68%
Recall	12.43%
F-score	6.8%
Perplexity	82

TABLE 6.10: Attention based Context Aware Language Model (with Neural Machine Translation Context Representation) Final Results

context-aware language model described in chapter 6.3.2. A possible explanation for the performance of the attention based model is that the context relevant information are at arbitrary positions within the paragraph. This allows no prediction on which input sentence might contain the most salient information for the context, as the information is randomly distributed. The network can therefore not learn any relation between the target sentence and the position of the information in the context.

6.3.4 Context Aware Binary Model

The third model type to test the influence of a context to improve the system performance is the context-aware binary model. In contrast to the other context-aware models evaluated on the task, the binary classification model is trained on supervised training data, separating the two classes

$$original \rightarrow 0$$
$$out - of - context \rightarrow 1$$

The architecture of the model is similar to the model presented in figure 6.6. The only architectural difference to the standard context-aware language model is the output layer, which only contains two neurons performing a softmax activation function on the binary classes. Nevertheless, this slight design change within the network requires major changes in the preprocessing pipeline to adapt to the new network. The exact hyper-parameters used are shown in table 6.11.

All the parameters of the network are kept constant with only the output layer amount of neurons changed down to the two output classes. This largely decreases the amount of neurons and therefore the amount of trainable weights in the network. The former weight matrix between the hidden layer and the output layer with a size of $[512 \times 30,000]$ containing $512 * 30,000 = 15,360,000$ weights is reduced to the weight matrix of size $[512]$ containing $512 * 2 = 1,024$ individual weight values. This significantly decreases the computational efforts during the forward calculation and the backpropagation. The supervised binary classification networks described in table 6.11 are trained on the supervised TEDTalk training data. The performance

Parameter	Model value
Optimizer	Adam
Cost Function	Cross Entropy
Learning Rates	[1e-3, 5e-4, 1e-4]
Iterations	16
Vocabulary Size	30,000
Sequence Length	50
Context Length	10
Output Layer Size	2
Embedding Size	256
Hidden Layer Size	512
Hidden Layers	1
Batch Size	100

TABLE 6.11: Context Aware Binary Classification Model Hyper-Parameters

evaluation to find the superior combination of learning rate and training epoch uses the F-score of the network to compare the two context representations in figures 6.16 and 6.17.

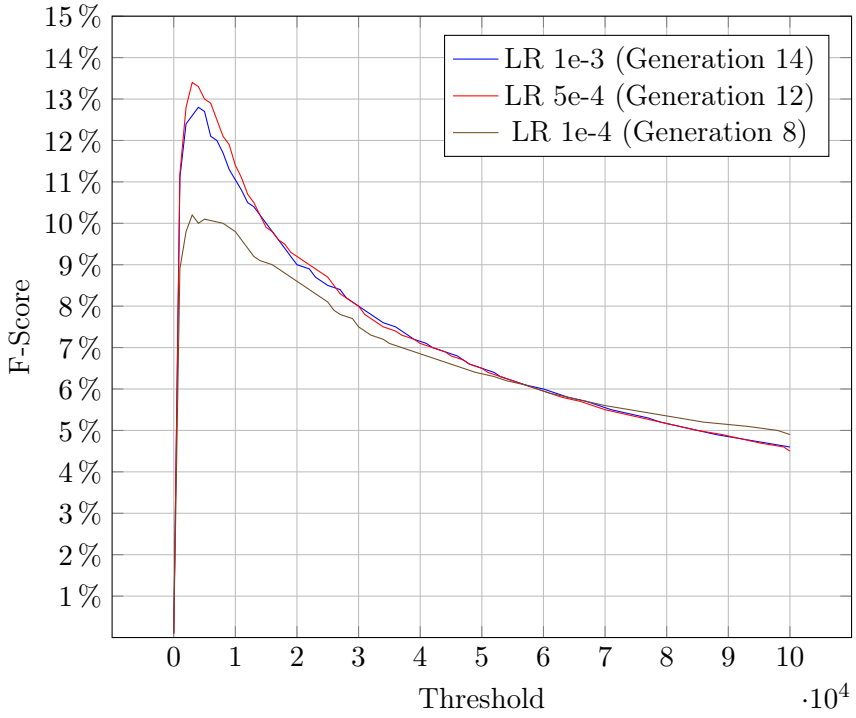

FIGURE 6.16: Context Aware Binary Classification Model F-Score (with Language Model Context Representation)

The best F-score with the baseline language model context representation in figure 6.16 is achieved by the model with $5e-4$ as the learning rate in iteration 12. It reaches

a F-score of 13.37% with a threshold value of 3,000 tokens. The model is therefore further evaluated on the test set, summarized in table 6.12.

Measure	Model Performance
Precision	13.4%
Recall	10.72%
F-score	11.92%

TABLE 6.12: Context Aware Binary Classification Model (with Language Model Context Representation) Final Result

The alternative approach utilizing the translation model to encode the sentence representations is analyzed in figure 6.17.

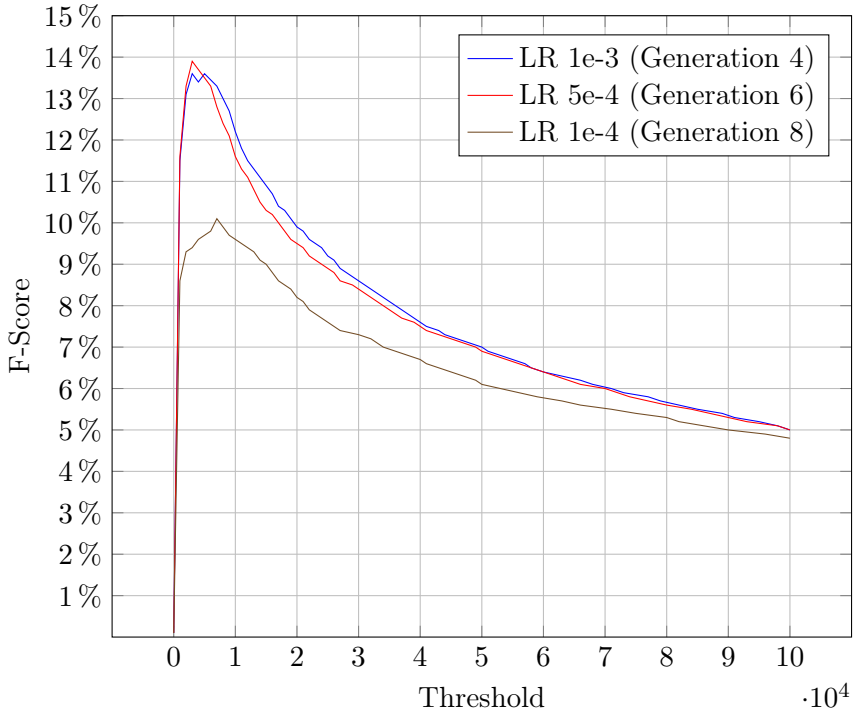

FIGURE 6.17: Context Aware Binary Classification Model F-Score (with Neural Machine Translation Context Representation)

Both learning rates of $1e-3$ and $5e-4$ show a F-score of nearly 14% with the score of the model $5e-4$ performing slightly better, reaching a F-score of 13.9% in generation 6 with a threshold of 3,000 tokens. The final result for this setup is calculated for generation 6 of the model with learning rate $5e-4$. The result is shown in table 6.13.

The performance of both binary classification models is comparable. Nevertheless, the network topology based on the translation sentence representations achieves better results on the development and the test set. With a detection rate of 12.23% of all inserted errors, the result of this model achieves the best results. Even though the

Measure	Model Performance
Precision	13.77%
Recall	11.01%
F-score	12.23%

TABLE 6.13: Context Aware Binary Classification Model (with Neural Machine Translation Context Representation) Final Result

supervised baseline model reached the best results on the baseline task with a F-score of 10.16%, the context-aware model enhances this high baseline by 2.07% absolute and 20.37% relative, showing that the context extension benefits all, unsupervised and supervised, models.

6.4 Evaluation Overview

The individual scores presented in the previous sections are consolidated in table 6.14. The direct comparison shows the superiority of context-aware models over sentence-based approaches. The best result in each category is highlighted for the unsupervised and supervised topologies.

Model	Perplexity	Precision	Recall	F-score
Baseline Lang Model	115	4.12%	15.42%	6.51%
Context Lang Model LM-CR	79	5.18%	11.05%	7.06%
Context Lang Model NMT-CR	**76**	**5.43%**	**11.31%**	**7.34%**
Context Attn Lang Model LM-CR	125	3.05%	7.32%	4.31%
Context Attn Lang Model NMT-CR	82	4.68%	12.43%	6.8%
Baseline Bin Class Model	-	8.94%	11.76%	10.16%
Context Bin Class Model LM-CR	-	13.4%	10.72%	11.92%
Context Bin Class Model NMT-CR	**-**	**13.77%**	**11.01%**	**12.23%**

TABLE 6.14: Final Comparison of the Model Performance

Chapter 7

Conclusion

In this work, we presented a novel architecture for context-aware computational models. The designed task reproduces common errors of state-of-the-art transcription and translation systems that can be prevented by taking a broader discourse into account. Our newly introduced substitution approach to modify existing datasets with out-of-context tokens represents the ground-truth for the assessment of the context-aware computational models. We show that the substituted tokens are difficult to infer by models that solely employ the local context. Thus, a context-aware approach is developed to enhance the performance on the task. Through our automated modification process, the necessary ground-truth generation is accessible and fast. With the 2016 TEDTalk database, modified through the automated modification pipeline, we enabled the training and testing of computational models against real-world out-of-context errors.

The application of two baseline models shows the difficulty to detect the out-of-context errors. The unsupervised language model, limited to the local context of the text, is trained on the unsupervised training data and tested against the ground-truth on the development and test sets. The second model, utilizing the supervised information during the training and testing phase, solves a binary classification task. To assess the significance of a contextual component, multiple hierarchical context-aware models are implemented to encode the broader discourse of the text into the computational processing. The context sensitive approaches employing the context of the text passage are compared with the baseline models to assess the importance of the context.

The context-sensitive translation model improves the baseline model by a factor of 12.75% relative and increases the detection rate from 6.51% to 7.34%. The second approach, to enhance the context sensitive language model by adding an attention mechanism, was not able to further improve the detection rate of out-of-context errors. A possible explanation for this behavior is given in the respective chapter.

The supervised model, which already reached an out-of-context detection rate of 10.16% in the baseline version is extended by a context component and further enhances the detection rate up to 12.23%.

This consistent improvement of the performance for both, unsupervised and supervised models, shows the significance of the context. This result proves, that the employment of a context within a contextual text passage enhances the overall performance of the system. Through the shallow design of the networks and the limited amount of neurons per layer, a basic example for the efficiency of the context is given. Expanding the network and introducing more sophisticated computational models can enhance this approach. Nevertheless, the basic assumption that a context-aware system outperforms models with only local context has been confirmed. Possible extensions that should be taken into account are described in the next chapter.

Chapter 8

Further Work

This work has been conducted to show the superiority of context-aware system over sentence-based approaches. To keep the efforts within this thesis manageable, some constraints have been introduced, including limited size topologies and basic network designs. To further investigate on the matter, an increased number of computational layers should be employed within the network to learn more abstract relations on the data. To employ more information from the context, more sophisticated approaches for the encoder component can be utilized, e.g. using bidirectional encoder architectures. Another open task based on this work is the assessment of different computational units within the network. We also applied a restriction for the vocabulary in this work. To save computational efforts during the training process, the vocabulary has been limited for all models. An extended vocabulary and open vocabulary approaches can further enhance the quality of the models by reducing the amount of unknown tokens. With a larger vocabulary and deeper models, the embedding layer is another source for improvements. A larger vector of real-valued word-embeddings can enhance the abstraction of words in the model.

Appendix A

Code

The following chapter contains code snippets of the instantiated model topologies for the baseline networks and the context-aware models.

A.1 Baseline Model

The baseline models are build on top of the Keras neural network library, which creates computational models through the interconnection of layers. The specific architecture is build on a high level structure.

A.1.1 Language Model

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Embedding, LSTM
from keras.layers.wrappers import TimeDistributed
from keras import optimizers

model = Sequential()
model.add(Embedding(input_dim=len(vocab), output_dim=embedding_size,
    mask_zero=True))
# Dynamic Number of LSTM Layers
for layer in range(0, nb_hidden_layers):
   model.add(LSTM(units=hidden_dimensions, return_sequences=True))
# Add a Dense Layer for every Timestep
model.add(TimeDistributed(Dense(len(vocab), activation='softmax')))
optimizer = optimizers.Adam(lr=learning_rate)
model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer)
```

A.1.2 Binary Classification Model

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Embedding, LSTM
from keras.layers.wrappers import TimeDistributed
from keras import optimizers

model = Sequential()
model.add(Embedding(input_dim=len(vocab), output_dim=embedding_size,
    mask_zero=True))
# Dynamic Number of LSTM Layers
for layer in range(0, nb_hidden_layers):
   model.add(LSTM(units=hidden_dimensions, return_sequences=True))
# Add a Dense Layer for every Timestep
model.add(TimeDistributed(Dense(1, activation='sigmoid')))
optimizer = optimizers.Adam(lr=learning_rate)
model.compile(loss='binary_crossentropy', optimizer=optimizer)
```

A.2 Context-Aware Model

The NMT context representation and the context-aware models use the Tensorflow deep-learning framework to build the topologies. Tensorflow offers low-level functionalities to build highly specific models, which cannot be plugged together with the predefined Keras layers.

A.2.1 NMT Context Representation

```
import tensorflow as tf
from tensorflow.python.framework import dtypes

# Inputs / Outputs
encoder_inputs = tf.placeholder(dtypes.int64, shape=[None,
    enc_timesteps_max])
encoder_lengths = tf.placeholder(dtypes.int32, shape=[None])
decoder_inputs = tf.placeholder(dtypes.int64, shape=[None,
    dec_timesteps_max])
decoder_lengths = tf.placeholder(dtypes.int32, shape=[None])
decoder_outputs = tf.placeholder(dtypes.int64, shape=[None,
    dec_timesteps_max])
masking = tf.placeholder(dtypes.float32, shape=[None, dec_timesteps_max])
start_token_infer = tf.placeholder(dtypes.int32, shape=[None])

# Cell Definition
encoder_cell = tf.contrib.rnn.LSTMCell(hidden_units)
```

```
if hidden_layers > 1:
   encoder_cell =
       tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.LSTMCell(hidden_units)
       for _ in range(hidden_layers)])

decoder_cell = tf.contrib.rnn.LSTMCell(hidden_units)
if hidden_layers > 1:
   decoder_cell =
       tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.LSTMCell(hidden_units)
       for _ in range(hidden_layers)])

# Encoder
embeddings_enc = tf.Variable(tf.random_uniform([enc_input_dimension,
    embedding_size], -1.0, 1.0), dtype=tf.float32)
encoder_inputs_embedded = tf.nn.embedding_lookup(embeddings_enc,
    encoder_inputs)
_, initial_state = tf.nn.dynamic_rnn(encoder_cell, encoder_inputs_embedded,
    sequence_length=encoder_lengths, dtype=tf.float32)

# Decoder
embeddings_dec = tf.Variable(tf.random_uniform([dec_input_dimension,
    embedding_size], -1.0, 1.0), dtype=tf.float32)
decoder_inputs_embedded = tf.nn.embedding_lookup(embeddings_dec,
    decoder_inputs)

final_layer = layers_core.Dense(units=dec_input_dimension)
helper = tf.contrib.seq2seq.TrainingHelper(decoder_inputs_embedded,
    decoder_lengths)
decoder = tf.contrib.seq2seq.BasicDecoder(decoder_cell, helper,
    initial_state, output_layer=final_layer)
outputs, _, _ = tf.contrib.seq2seq.dynamic_decode(decoder,
    maximum_iterations=dec_timesteps_max)
training_output = outputs.rnn_output

helper_infer = tf.contrib.seq2seq.GreedyEmbeddingHelper(embeddings_dec,
    start_token_infer, end_of_sequence_id)
decoder_infer = tf.contrib.seq2seq.BasicDecoder(decoder_cell, helper_infer,
    initial_state, output_layer=final_layer)
outputs_infer, _, _ = tf.contrib.seq2seq.dynamic_decode(decoder_infer,
    maximum_iterations=dec_timesteps_max)
infer_output = outputs_infer.sample_id

# Training
loss = tf.contrib.seq2seq.sequence_loss(targets=decoder_outputs,
    logits=training_output, weights=masking)
updates = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```

A.2.2 Context Aware Language Model

```
import tensorflow as tf
from tensorflow.python.framework import dtypes

# Inputs / Outputs
encoder_inputs = tf.placeholder(dtypes.float32, shape=[None,
    enc_timesteps_max, enc_input_dimension])
encoder_lengths = tf.placeholder(dtypes.int32, shape=[None])
decoder_inputs = tf.placeholder(dtypes.int64, shape=[None,
    dec_timesteps_max])
decoder_lengths = tf.placeholder(dtypes.int32, shape=[None])
decoder_outputs = tf.placeholder(dtypes.int64, shape=[None,
    dec_timesteps_max])
masking = tf.placeholder(dtypes.float32, shape=[None, dec_timesteps_max])

# Cell Definition
encoder_cell = tf.contrib.rnn.LSTMCell(hidden_units)
if hidden_layers > 1:
   encoder_cell =
       tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.LSTMCell(hidden_units)
       for _ in range(hidden_layers)])

decoder_cell = tf.contrib.rnn.LSTMCell(hidden_units)
if hidden_layers > 1:
   decoder_cell =
       tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.LSTMCell(hidden_units)
       for _ in range(hidden_layers)])

# Encoder
_, initial_state = tf.nn.dynamic_rnn(encoder_cell, encoder_inputs,
    sequence_length=encoder_lengths, dtype=tf.float32)

# Decoder
embeddings = tf.Variable(tf.random_uniform([vocab_size,
    input_embedding_size], -1.0, 1.0), dtype=tf.float32)
decoder_inputs_embedded = tf.nn.embedding_lookup(embeddings, decoder_inputs)
lstm_output,state = tf.nn.dynamic_rnn(decoder_cell,
    decoder_inputs_embedded, sequence_length=decoder_lengths,
    initial_state=initial_state)
transp = tf.transpose(lstm_output, [1, 0, 2])
lstm_output_unpacked = tf.unstack(transp)
outputs = []
for index, item in enumerate(lstm_output_unpacked):
   if index == 0:
      logits = tf.layers.dense(inputs=item, units=vocab_size)
   if index > 0:
      logits = tf.layers.dense(inputs=item, units=vocab_size, reuse=True)
```

```
   outputs.append(logits)

tensor_output = tf.stack(values=outputs, axis=0)
forward = tf.transpose(tensor_output, [1, 0, 2])

# Training
loss = tf.contrib.seq2seq.sequence_loss(targets=decoder_outputs,
    logits=forward, weights=masking)
updates = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```

A.2.3 Attention-based Context Aware Language Model

```
# Inputs / Outputs
encoder_inputs = tf.placeholder(dtypes.float32, shape=[None,
    enc_timesteps_max, enc_input_dimension])
encoder_lengths = tf.placeholder(dtypes.int32, shape=[None])
decoder_inputs = tf.placeholder(dtypes.int64, shape=[None,
    dec_timesteps_max])
decoder_lengths = tf.placeholder(dtypes.int32, shape=[None])
decoder_outputs = tf.placeholder(dtypes.int64, shape=[None,
    dec_timesteps_max])
masking = tf.placeholder(dtypes.float32, shape=[None, dec_timesteps_max])

# Cell Definition
encoder_cell = tf.contrib.rnn.LSTMCell(hidden_units)
if hidden_layers > 1:
   encoder_cell =
       tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.LSTMCell(hidden_units)
       for _ in range(hidden_layers)])

# Encoder
encoder_outputs, initial_state = tf.nn.dynamic_rnn(encoder_cell,
    encoder_inputs, sequence_length=encoder_lengths, dtype=tf.float32)

# Attention
single_cell_dec = tf.contrib.rnn.LSTMCell(hidden_units)
attention_mechanism = tf.contrib.seq2seq.BahdanauAttention(hidden_units,
    encoder_outputs, memory_sequence_length=encoder_lengths)
attn_cell = tf.contrib.seq2seq.AttentionWrapper(single_cell_dec,
    attention_mechanism, initial_cell_state=initial_state)
decoder_cell = attn_cell
if hidden_layers > 1:
   decoder_cell = tf.contrib.rnn.MultiRNNCell([attn_cell for _ in
       range(hidden_layers)])

# Decoder
```

```
embeddings = tf.Variable(tf.random_uniform([vocab_size,
    input_embedding_size], -1.0, 1.0), dtype=tf.float32)
decoder_inputs_embedded = tf.nn.embedding_lookup(embeddings, decoder_inputs)
lstm_output,state = tf.nn.dynamic_rnn(cell=decoder_cell,
    inputs=decoder_inputs_embedded, sequence_length=decoder_lengths,
    dtype=tf.float32)
transp = tf.transpose(lstm_output, [1, 0, 2])
lstm_output_unpacked = tf.unstack(transp)
outputs = []
for index, item in enumerate(lstm_output_unpacked):
   if index == 0:
      logits = tf.layers.dense(inputs=item, units=vocab_size)
   if index > 0:
      logits = tf.layers.dense(inputs=item, units=vocab_size, reuse=True)
   outputs.append(logits)
tensor_output = tf.stack(values=outputs, axis=0)
forward = tf.transpose(tensor_output, [1, 0, 2])

# Training
loss = tf.contrib.seq2seq.sequence_loss(targets=decoder_outputs,
    logits=forward, weights=masking)
updates = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```

A.2.4 Context Aware Binary Model

```
# Inputs / Outputs
encoder_inputs = tf.placeholder(dtypes.float32, shape=[None,
    enc_timesteps_max, enc_input_dimension])
encoder_lengths = tf.placeholder(dtypes.int32, shape=[None])
decoder_inputs = tf.placeholder(dtypes.int64, shape=[None,
    dec_timesteps_max])
decoder_lengths = tf.placeholder(dtypes.int32, shape=[None])
decoder_outputs = tf.placeholder(dtypes.int64, shape=[None,
    dec_timesteps_max])
masking = tf.placeholder(dtypes.float32, shape=[None, dec_timesteps_max])

# Cell Definition
encoder_cell = tf.contrib.rnn.LSTMCell(hidden_units)
if hidden_layers > 1:
   encoder_cell =
       tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.LSTMCell(hidden_units)
       for _ in range(hidden_layers)])

decoder_cell = tf.contrib.rnn.LSTMCell(hidden_units)
if hidden_layers > 1:
```

```
   decoder_cell =
       tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.LSTMCell(hidden_units)
       for _ in range(hidden_layers)])

# Encoder
_, initial_state = tf.nn.dynamic_rnn(encoder_cell, encoder_inputs,
    sequence_length=encoder_lengths, dtype=tf.float32)

# Decoder
embeddings = tf.Variable(tf.random_uniform([vocab_size,
    input_embedding_size], -1.0, 1.0), dtype=tf.float32)
decoder_inputs_embedded = tf.nn.embedding_lookup(embeddings, decoder_inputs)
lstm_output,state = tf.nn.dynamic_rnn(decoder_cell,
    decoder_inputs_embedded, sequence_length=decoder_lengths,
    initial_state=initial_state)
transp = tf.transpose(lstm_output, [1, 0, 2])
lstm_output_unpacked = tf.unstack(transp)
outputs = []
for index, item in enumerate(lstm_output_unpacked):
   if index == 0:
      logits = tf.layers.dense(inputs=item, units=2)
   if index > 0:
      logits = tf.layers.dense(inputs=item, units=2, reuse=True)
   outputs.append(logits)
tensor_output = tf.stack(values=outputs, axis=0)
forward = tf.transpose(tensor_output, [1, 0, 2])
forward_output = tf.nn.softmax(forward)

# Training
loss = tf.contrib.seq2seq.sequence_loss(targets=decoder_outputs,
    logits=forward, weights=masking)
updates = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```

Appendix B

Raw Data

This chapter contains the raw data of the presented graphs in the evaluation.

B.1 Baseline Model

Gen	Development			Training		
	LR 1e-3	LR 1e-4	LR 5e-4	LR 1e-3	LR 1e-4	LR 5e-4
2	94	305	132	88	323	135
4	80	193	108	61	204	102
6	85	162	105	50	168	91
8	94	145	108	44	148	85
10	107	133	115	39	135	82
12	120	126	121	35	126	79
14	135	120	127	32	119	75
16	151	116	132	29	113	72

Table B.1: Language Model Perplexity

Gen	LR 1e-3	LR 1e-4	LR 5e-4	Gen	LR 1e-3	LR 1e-4	LR 5e-4
1	0%	0%	0%	21000	5.8%	4%	6.2%
1000	1%	3.5%	1%	22000	5.8%	3.9%	6.1%
2000	2%	4.7%	1.8%	23000	5.7%	3.9%	6%
3000	2.5%	5.1%	2.4%	24000	5.7%	3.8%	6%
4000	3.1%	5.2%	3.2%	25000	5.6%	3.8%	6%
5000	3.5%	5.2%	3.8%	26000	5.6%	3.8%	5.9%
6000	4.3%	5.3%	4.5%	27000	5.5%	3.8%	5.8%
7000	4.5%	5.1%	5.3%	28000	5.5%	3.7%	5.7%
8000	4.6%	5%	5.8%	29000	5.4%	3.6%	5.6%
9000	4.9%	4.8%	6.2%	30000	5.4%	3.6%	5.6%
10000	5.1%	4.9%	6.3%	31000	5.4%	3.6%	5.5%
11000	5.2%	4.7%	6.5%	32000	5.3%	3.5%	5.5%
12000	5.5%	4.7%	6.4%	33000	5.2%	3.5%	5.4%
13000	5.8%	4.5%	6.5%	34000	5.2%	3.5%	5.4%
14000	5.8%	4.4%	6.5%	35000	5.2%	3.4%	5.3%
15000	6%	4.3%	6.5%	36000	5.2%	3.4%	5.3%
16000	6%	4.3%	6.4%	37000	5.1%	3.3%	5.2%
17000	5.9%	4.2%	6.4%	38000	5.1%	3.3%	5.2%
18000	5.9%	4.1%	6.3%	39000	5%	3.3%	5.1%
19000	5.9%	4.1%	6.2%	40000	5%	3.2%	5.1%
20000	5.9%	4%	6.2%	41000	5%	3.2%	5%

Table B.2: Language Model F-Score

Gen	LR 1e-3	LR 1e-4	LR 5e-4	Gen	LR 1e-3	LR 1e-4	LR 5e-4
1	0.1%	0.1%	0%	21001	8%	6.5%	7%
1001	7.4%	4.4%	5.4%	22001	7.9%	6.5%	6.9%
2001	9.1%	5.6%	6.9%	23001	7.7%	6.5%	6.8%
3001	9.5%	5.7%	7.5%	24001	7.6%	6.5%	6.7%
4001	10.1%	5.8%	7.8%	25001	7.5%	6.4%	6.6%
5001	10.4%	5.9%	8.2%	26001	7.4%	6.3%	6.5%
6001	10.3%	6.2%	8.3%	27001	7.3%	6.3%	6.5%
7001	10.1%	6.4%	8.4%	28001	7.2%	6.2%	6.4%
8001	9.9%	6.6%	8.3%	29001	7%	6.2%	6.3%
9001	9.7%	6.7%	8.2%	30001	7%	6.2%	6.2%
10001	9.6%	6.8%	8.2%	31001	6.9%	6.1%	6.1%
11001	9.4%	6.7%	8.1%	32001	6.8%	6%	6.1%
12001	9.2%	6.8%	8%	33001	6.7%	5.9%	6%
13001	9.1%	6.7%	7.9%	34001	6.6%	5.9%	6%
14001	8.9%	6.7%	7.9%	35001	6.6%	5.8%	5.9%
15001	8.7%	6.6%	7.7%	36001	6.5%	5.8%	5.8%
16001	8.6%	6.6%	7.6%	37001	6.4%	5.8%	5.8%
17001	8.4%	6.6%	7.5%	38001	6.3%	5.8%	5.7%
18001	8.3%	6.6%	7.3%	39001	6.2%	5.7%	5.6%
19001	8.2%	6.6%	7.2%	40001	6.1%	5.6%	5.6%
20001	8.1%	6.5%	7.1%	41001	6.1%	5.6%	5.5%

Table B.3: Binary Classification Model F-Score

B.2 Context-Aware Model

Generation	LR 1e-3	LR 1e-4	LR 5e-4
2	68	211	86
4	54	132	59
6	56	100	52
8	63	84	51
10	76	73	55
12	93	67	58
14	109	62	64
16	134	58	71
18	156	56	80

Table B.4: NMT Context Representation Perplexity

Generation	LR 1e-3	LR 1e-4	LR 5e-4
2	82	150	95
4	73	119	78
6	76	102	74
8	84	93	74
10	95	87	76
12	108	83	79
14	122	80	83
16	137	77	87

Table B.5: Context Aware Language Model Perplexity (with Language Model Context Representation)

Gen	LR 1e-3	LR 1e-4	LR 5e-4	Gen	LR 1e-3	LR 1e-4	LR 5e-4
1	0%	0%	0.1%	21001	6.4%	5.9%	6.2%
1001	2.8%	2.6%	3.3%	22001	6.3%	5.9%	6.1%
2001	4%	4.5%	4.7%	23001	6.3%	5.8%	6%
3001	5.1%	5.5%	5.6%	24001	6.2%	5.7%	5.9%
4001	5.4%	5.8%	6.3%	25001	6.2%	5.6%	5.9%
5001	5.7%	6.2%	6.8%	26001	6.2%	5.6%	5.7%
6001	6.1%	6.4%	7.1%	27001	6.1%	5.5%	5.6%
7001	6.4%	6.5%	7.2%	28001	6.1%	5.4%	5.6%
8001	6.6%	6.6%	7.6%	29001	6%	5.4%	5.5%
9001	6.8%	6.7%	7.5%	30001	5.9%	5.3%	5.5%
10001	6.9%	6.7%	7.4%	31001	5.9%	5.3%	5.5%
11001	6.9%	6.6%	7.3%	32001	5.8%	5.2%	5.4%
12001	6.9%	6.5%	7.2%	33001	5.8%	5.1%	5.3%
13001	6.9%	6.4%	7.1%	34001	5.7%	5.1%	5.2%
14001	6.8%	6.4%	6.9%	35001	5.7%	5%	5.2%
15001	6.8%	6.4%	6.8%	36001	5.6%	5%	5.1%
16001	6.8%	6.3%	6.6%	37001	5.6%	4.9%	5.1%
17001	6.6%	6.3%	6.5%	38001	5.5%	4.8%	5%
18001	6.6%	6.2%	6.5%	39001	5.5%	4.8%	5%
19001	6.4%	6.1%	6.4%	40001	5.4%	4.7%	4.9%
20001	6.4%	6.1%	6.2%	41001	5.3%	4.7%	4.8%

Table B.6: Context Aware Language Model F-Score (with Language Model Context Representation)

Generation	LR 1e-3	LR 1e-4	LR 5e-4
2	81	147	93
4	73	112	77
6	77	97	73
8	85	89	73
10	96	84	75
12	108	80	78
14	122	77	82
16	137	75	87

Table B.7: Context Aware Language Model Perplexity (with Neural Machine Translation Context Representation)

Gen	LR 1e-3	LR 1e-4	LR 5e-4	Gen	LR 1e-3	LR 1e-4	LR 5e-4
1	0%	0%	0.1%	21001	5.6%	5.6%	6.1%
1001	4.6%	3.9%	4%	22001	5.5%	5.4%	6.1%
2001	5.5%	5.3%	5.2%	23001	5.4%	5.4%	6%
3001	6%	5.7%	5.9%	24001	5.3%	5.2%	5.9%
4001	6.5%	6.1%	6.3%	25001	5.2%	5.2%	5.8%
5001	6.6%	6.6%	6.8%	26001	5.1%	5.1%	5.7%
6001	6.6%	6.4%	7%	27001	5.1%	5%	5.6%
7001	6.5%	6.7%	7.2%	28001	5%	5%	5.6%
8001	6.6%	6.5%	7.2%	29001	5%	4.9%	5.5%
9001	6.5%	6.5%	7.1%	30001	4.9%	4.8%	5.5%
10001	6.4%	6.4%	7.1%	31001	4.9%	4.7%	5.5%
11001	6.2%	6.3%	6.9%	32001	4.8%	4.7%	5.4%
12001	6.2%	6.1%	6.9%	33001	4.8%	4.6%	5.3%
13001	6.1%	6.1%	6.8%	34001	4.7%	4.5%	5.3%
14001	6.1%	6%	6.9%	35001	4.6%	4.5%	5.2%
15001	6%	6%	6.7%	36001	4.6%	4.5%	5.1%
16001	6%	6%	6.6%	37001	4.6%	4.4%	5.1%
17001	5.9%	5.9%	6.5%	38001	4.5%	4.4%	5%
18001	5.8%	5.9%	6.4%	39001	4.5%	4.4%	5%
19001	5.7%	5.7%	6.3%	40001	4.4%	4.3%	4.9%
20001	5.6%	5.6%	6.2%	41001	4.4%	4.2%	4.9%

Table B.8: Context Aware Language Model F-Score (with Neural Machine Translation Context Representation)

Generation	LR 1e-3	LR 1e-4	LR 5e-4
2	152	214	171
4	137	162	141
6	126	146	133
8	125	136	127
10	127	129	124
12	131	124	123
14	137	121	125
16	144	118	128

Table B.9: Attention-based Context Aware Language Model Perplexity (with Language Model Context Representation)

Gen	LR 1e-3	LR 1e-4	LR 5e-4	Gen	LR 1e-3	LR 1e-4	LR 5e-4
1	0.1%	0%	0%	21001	3.6%	3.4%	4.1%
1001	1.7%	1.7%	2.1%	22001	3.6%	3.3%	4.1%
2001	2.3%	2.5%	3.1%	23001	3.6%	3.3%	4%
3001	2.7%	3.2%	3.8%	24001	3.6%	3.3%	4%
4001	3%	3.2%	4%	25001	3.5%	3.3%	4%
5001	3.3%	3.1%	4.1%	26001	3.5%	3.3%	4%
6001	3.4%	3.2%	4.4%	27001	3.5%	3.2%	3.9%
7001	3.6%	3.3%	4.5%	28001	3.5%	3.2%	3.9%
8001	3.6%	3.3%	4.6%	29001	3.5%	3.2%	3.8%
9001	3.7%	3.4%	4.6%	30001	3.4%	3.2%	3.7%
10001	3.7%	3.4%	4.6%	31001	3.4%	3.2%	3.7%
11001	3.7%	3.5%	4.5%	32001	3.4%	3.1%	3.7%
12001	3.7%	3.5%	4.4%	33001	3.3%	3.1%	3.6%
13001	3.8%	3.5%	4.4%	34001	3.4%	3.1%	3.6%
14001	3.8%	3.5%	4.5%	35001	3.4%	3.1%	3.5%
15001	3.8%	3.5%	4.4%	36001	3.3%	3.1%	3.5%
16001	3.8%	3.4%	4.3%	37001	3.3%	3%	3.5%
17001	3.8%	3.4%	4.3%	38001	3.3%	3%	3.5%
18001	3.8%	3.4%	4.3%	39001	3.3%	3%	3.4%
19001	3.7%	3.4%	4.2%	40001	3.2%	3%	3.4%
20001	3.7%	3.3%	4.2%	41001	3.2%	2.9%	3.4%

Table B.10: Attention-based Context Aware Language Model F-Score (with Language Model Context Representation)

Generation	LR 1e-3	LR 1e-4	LR 5e-4
2	82	145	94
4	73	112	78
6	76	98	74
8	85	90	74
10	95	85	76
12	108	81	79
14	121	78	83
16	135	76	88

Table B.11: Attention-based Context Aware Language Model Perplexity (with Neural Machine Translation Context Representation)

Gen	LR 1e-3	LR 1e-4	LR 5e-4	Gen	LR 1e-3	LR 1e-4	LR 5e-4
1	0%	0%	0%	21001	6.5%	6.1%	6.1%
1001	2.2%	3.2%	3.7%	22001	6.4%	6%	6%
2001	3.8%	5%	5.1%	23001	6.4%	6%	5.9%
3001	4.8%	6.1%	5.9%	24001	6.3%	5.9%	5.8%
4001	5.3%	6.2%	6.3%	25001	6.2%	5.8%	5.8%
5001	5.9%	6.5%	6.7%	26001	6.2%	5.7%	5.7%
6001	6.2%	6.6%	6.9%	27001	6.2%	5.7%	5.7%
7001	6.5%	6.7%	6.8%	28001	6.1%	5.6%	5.6%
8001	6.7%	6.7%	6.9%	29001	6.1%	5.5%	5.5%
9001	6.8%	6.6%	7%	30001	6%	5.5%	5.4%
10001	6.8%	6.6%	7%	31001	6%	5.4%	5.3%
11001	6.9%	6.6%	6.9%	32001	5.9%	5.3%	5.3%
12001	6.9%	6.6%	6.8%	33001	5.8%	5.2%	5.2%
13001	6.8%	6.5%	6.8%	34001	5.8%	5.2%	5.1%
14001	6.8%	6.5%	6.7%	35001	5.7%	5.1%	5.1%
15001	6.8%	6.4%	6.6%	36001	5.7%	5.1%	5%
16001	6.8%	6.4%	6.5%	37001	5.6%	5%	5%
17001	6.7%	6.3%	6.4%	38001	5.6%	5%	4.9%
18001	6.7%	6.2%	6.4%	39001	5.5%	4.9%	4.9%
19001	6.7%	6.2%	6.3%	40001	5.5%	4.9%	4.8%
20001	6.6%	6.2%	6.2%	41001	5.4%	4.9%	4.8%

TABLE B.12: Attention-based Context Aware Language Model F-Score (with Neural Machine Translation Context Representation)

Gen	LR 1e-3	LR 1e-4	LR 5e-4	Gen	LR 1e-3	LR 1e-4	LR 5e-4
1	0.1%	0.1%	0.1%	21001	9%	8.5%	9.1%
1001	11.1%	8.9%	11.2%	22001	8.9%	8.4%	9%
2001	12.4%	9.8%	12.8%	23001	8.7%	8.3%	8.9%
3001	12.6%	10.2%	13.4%	24001	8.6%	8.2%	8.8%
4001	12.8%	10%	13.3%	25001	8.5%	8.1%	8.7%
5001	12.7%	10.1%	13%	26001	8.5%	7.9%	8.5%
6001	12.1%	10.1%	12.9%	27001	8.4%	7.8%	8.3%
7001	12%	10.1%	12.5%	28001	8.2%	7.8%	8.2%
8001	11.7%	10%	12.1%	29001	8.1%	7.7%	8.1%
9001	11.3%	9.9%	11.9%	30001	8%	7.5%	8%
10001	11.3%	9.8%	11.4%	31001	7.9%	7.4%	7.8%
11001	10.8%	9.6%	11.1%	32001	7.8%	7.3%	7.7%
12001	10.5%	9.4%	10.7%	33001	7.7%	7.3%	7.6%
13001	10.4%	9.2%	10.5%	34001	7.6%	7.2%	7.5%
14001	10.2%	9.1%	10.2%	35001	7.6%	7.1%	7.5%
15001	10%	9.1%	9.9%	36001	7.5%	7.1%	7.4%
16001	9.8%	9%	9.8%	37001	7.4%	7%	7.3%
17001	9.6%	8.9%	9.6%	38001	7.4%	7%	7.3%
18001	9.4%	8.8%	9.5%	39001	7.2%	6.9%	7.2%
19001	9.2%	8.7%	9.3%	40001	7.2%	6.9%	7.1%
20001	9%	8.6%	9.2%	41001	7.1%	6.8%	7.1%

Table B.13: Context Aware Binary Model F-Score (with Language Model Context Representation)

Gen	LR 1e-3	LR 1e-4	LR 5e-4	Gen	LR 1e-3	LR 1e-4	LR 5e-4
1	0.1%	0.1%	0.1%	21001	9.8%	8.1%	9.4%
1001	11.5%	8.6%	11.6%	22001	9.6%	7.9%	9.2%
2001	13.1%	9.3%	13.3%	23001	9.5%	7.8%	9.1%
3001	13.6%	9.4%	13.9%	24001	9.4%	7.7%	9%
4001	13.4%	9.6%	13.7%	25001	9.2%	7.6%	8.9%
5001	13.6%	9.7%	13.5%	26001	9.1%	7.5%	8.8%
6001	13.6%	9.8%	13.3%	27001	8.9%	7.4%	8.6%
7001	13.3%	10.1%	12.8%	28001	8.8%	7.4%	8.6%
8001	13%	9.9%	12.4%	29001	8.7%	7.4%	8.5%
9001	12.7%	9.7%	12.1%	30001	8.6%	7.3%	8.4%
10001	12.2%	9.6%	11.6%	31001	8.5%	7.3%	8.3%
11001	11.8%	9.5%	11.3%	32001	8.4%	7.2%	8.2%
12001	11.5%	9.4%	11.1%	33001	8.3%	7.1%	8.1%
13001	11.3%	9.3%	10.8%	34001	8.2%	7%	8%
14001	11.1%	9.1%	10.5%	35001	8.1%	7%	7.9%
15001	10.9%	9%	10.3%	36001	8%	6.9%	7.9%
16001	10.7%	8.8%	10.2%	37001	7.9%	6.9%	7.7%
17001	10.4%	8.6%	10%	38001	7.8%	6.8%	7.7%
18001	10.3%	8.5%	9.8%	39001	7.7%	6.8%	7.6%
19001	10.1%	8.4%	9.6%	40001	7.6%	6.7%	7.5%
20001	9.9%	8.2%	9.5%	41001	7.5%	6.6%	7.4%

Table B.14: Context Aware Binary Model F-Score (with Neural Machine Translation Context Representation)

Bibliography

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016. URL `http://arxiv.org/abs/1603.04467`.

A. Aboukarima, H. Elsoury, and M. Menyawi. Artificial neural network model for the prediction of the cotton crop leaf area. *Int. J. Plant Soil Sci*, 8:1–13, 2015.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL `http://arxiv.org/abs/1409.0473`.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Y. Bengio, R. D. andPascal Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137?1155, 02 2003. URL `http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf`.

F. Burlot and F. Yvon. Evaluating the morphological competence of machine translation systems. In *Proceedings of the Second Conference on Machine Translation*, pages 43–55, 2017.

M. Cettolo, C. Girardi, and M. Federico. Wit3: Web inventory of transcribed and translated talks. *Proceedings of the EAMT Conference*, 16:261–268, 05 2012. URL `http://www.mt-archive.info/EAMT-2012-Cettolo.pdf`.

K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL `http://arxiv.org/abs/1409.1259`.

F. Chollet. Keras documentation. URL `https://keras.io`.

J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL `http://arxiv.org/abs/1412.3555`.

M.-C. De Marneffe, B. MacCartney, C. D. Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, number 2006, pages 449–454. Genoa Italy, 2006.

F-Score Definition. F-score definition. URL `https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/`.

K. M. Hermann, T. Kociský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015. URL `http://arxiv.org/abs/1506.03340`.

F. Hill, A. Bordes, S. Chopra, and J. Weston. The goldilocks principle: Reading children's books with explicit memory representations. *CoRR*, abs/1511.02301, 2015. URL `http://arxiv.org/abs/1511.02301`.

S. Hochreiter and J. Schmidhuber. Long short-term memory. 9:1735–80, 12 1997.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. 1989.

IAR. IAR institut fuer anthropomatik und robotik. URL `https://www.informatik.kit.edu/1323.php`.

ISL. Interactive systems lab (isl). URL `http://isl.anthropomatik.kit.edu/english/index.php`.

M. Iyyer, J. Boyd-Graber, L. Claudino, R. Socher, and H. D. III. A neural network for factoid question answering over paragraphs. In *Empirical Methods in Natural Language Processing*, 2014a. URL `https://pdfs.semanticscholar.org/2872/52a5fb2f2a9e311eebf06e5ac49eb52eaadc.pdf`.

M. Iyyer, J. Boyd-Graber, and H. Daumé III. Generating sentences from semantic vector space representations. In *NIPS Workshop on Learning Semantics*, 2014b.

A. K. Jain, J. Mao, and K. Mohiuddin. Artificial neural networks: A tutorial. 1996.

U. Karn. A quick introduction to neural networks, 2016. URL `https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/`.

A. Karpathy. The unreasonable effectiveness of recurrent neural networks, 2015. URL `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-aware neural language models. *CoRR*, abs/1508.06615, 2015. URL `http://arxiv.org/abs/1508.06615`.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

S. C. Kleene. Representation of events in nerve nets and finite automata. Technical report, RAND PROJECT AIR FORCE SANTA MONICA CA, 1951. URL `https://www.rand.org/content/dam/rand/pubs/research_memoranda/2008/RM704.pdf`.

P. Koehn. Statistical machine translation. In *Statistical Machine Translation*, chapter Chapter 7: Language Models. Cambridge University Press, `http://www.statmt.org/book/slides/07-language-models.pdf`, 2009.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004.

M. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015. URL `http://arxiv.org/abs/1508.04025`.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943. URL `http://www.cse.chalmers.se/~coquand/AUTOMATA/mcp.pdf`.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL `http://arxiv.org/abs/1301.3781`.

C. Olah. Understanding lstm networks, 2015. URL `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. *CoRR*, abs/1606.06031, 2016. URL `http://arxiv.org/abs/1606.06031`.

Perplexity Definition. Perplexity definition. URL `https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf`.

R. Quiza and J. Davim. Computational methods and optimization. In *Machining of Hard Materials*, pages 177–208. 01 2011.

R. Sennrich. How grammatical is character-level neural machine translation? assessing MT quality with contrastive translation pairs. *CoRR*, abs/1612.04629, 2016. URL `http://arxiv.org/abs/1612.04629`.

D. Shiffman. The nature of code. In *THE NATURE OF CODE*, chapter Chapter 10. Neural Networks. Creative Commons Attribution-NonCommercial 3.0 Unported License, `http://natureofcode.com/book/chapter-10-neural-networks/`, 12 2012.

H. Siegelmann. Computation beyond the Turing limit. *Science*, 268:545–548, 1995. URL `http://binds.cs.umass.edu/papers/1995_Siegelmann_Science.pdf`.

M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. *Annual Conference of the International Speech Communication Association*, 13, 2012. URL `https://pdfs.semanticscholar.org/f9a1/b3850dfd837793743565a8af95973d395a4e.pdf`.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL `http://arxiv.org/abs/1409.3215`.

Tensorflow Documentation. Tensorflow documentation. URL `https://www.tensorflow.org`.

J. Tiedemann. Finding alternative translations in a large corpus of movie subtitle. In N. C. C. Chair), K. Choukri, T. Declerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, and S. Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may 2016. European Language Resources Association (ELRA). ISBN 978-2-9517408-9-1.

Word2Vec. Word2vec, 2017. URL `https://code.google.com/archive/p/word2vec/`.

S. Xie and R. Rastogi. Deep poetry: Word-level and character-level language models for shakespearean sonnet generation. 2017.