Stuttgart Media University

Sequential transfer learning in NLP for text summarization

A thesis for the academic degree
'Master of Science' of

Pascal Fecht

Computer Science and Media
at the Faculty of Print and Media

Reviewer: Prof. Dr.-Ing. Johannes Maucher
External advisors: Dipl.-Ing. Christian Meder
Dipl.-Inform. Hans-Peter Zorn

November 2018 – May 2019

Stuttgart Media University
Faculty of Print and Media
Nobelstr. 10
70569 Stuttgart

Hiermit versichere ich, **Pascal Fecht**, ehrenwörtlich, dass ich die vorliegende Masterarbeit mit dem Titel: *"Sequential transfer learning in NLP for text summarization"* selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 23 Abs. 2 Master-SPO (3 Semester)) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

Stuttgart, den 09.05.2019

. . . . . .

(Pascal Fecht)

Abstract

This thesis investigates recent techniques for transfer learning and their influence on machine summarization systems. A current trend in Natural Language Processing (NLP) is to pre-train extensive language models in advance and adapt these to address problems in various task domains. Since these techniques have rarely been investigated in the context of text summarization, this thesis develops a workflow to integrate and evaluate pre-trained language models in neural text summarization. Based on news articles of the CNN / DailyMail dataset [35] and the CopyNet [32] summarization model, the conducted experiments show that transfer learning can have a positive impact on summarising texts. Further findings suggest that datasets with less historical data are more likely to benefit from transfer learning. On the other hand, however, this work demonstrates that the components of text summarization models limit the abilities of state-of-the-art transfer learning techniques. In the field of machine learning, this thesis is designed for readers interested in the state-of-the-art in transfer learning for NLP and its influence on the generation of summaries.

Keywords Natural Language Processing, Text Summarization, Transfer Learning, Language Model, Sequence-to-Sequence Model

Zusammenfassung

Diese Arbeit untersucht aktuelle Techniken für das Transferlernen und dessen Einfluss auf Systeme, die Texte maschinell und automatisiert zusammenfassen. Ein derzeitiger Trend für die maschinelle Verarbeitung natürlicher Sprache ist, extensive Sprachmodelle vor zu trainieren und diese anschließend zur Lösung verschiedener Problemstellungen zu adaptieren. Da kaum Untersuchungen dieser Techniken im Kontext des maschinellen Zusammenfassens existieren, entwickelt diese Arbeit einen Prozess, der vortrainierte Sprachmodelle in ein Zusammenfassungssystem integriert und deren Einfluss evaluiert. Die Resultate dieses Prozesses mit Nachrichtenartikeln des CNN / DailyMail Datensatzes [35] auf Basis des CopyNet [32] Zusammenfassungsmodells zeigen, dass Transferlernen das Zusammenfassen von Texten positiv beeinflussen kann. Weitere Erkenntnisse legen dar, dass Datensätze mit weniger historischen Daten in größerem Maße vom Transferlernen profitieren. Auf der anderen Seite dokumentiert diese Arbeit auch, dass die Komponenten von Zusammenfassungssystemen die Kapazitäten aktueller Techniken des Transferlernens limitieren. Im Kontext des maschinellen Lernens richtet sich diese Arbeit an Leser, die sich über den technologischen Stand des Transferlernens für natürliche Sprache allgemein, sowie dessen Einfluss auf die Erstellung von Zusammenfassungen informieren möchten.

Contents

List of Figures

List of Tables

List of Abbreviations

BERT	Bidirectional Encoder Representations from Transformers
BiLM	Bidirectional Language Model
BiLSTM	Bidirectional Long Short-Term Memory Network
BOW	Bag-of-Words
CBOW	Continuous Bag-of-Words
CLF	Classification
CoVe	Contextual Word Vectors
ELMo	Embeddings from Language Models
FFNN	Feed Forward Neural Network
GAN	Generative Adversarial Network
GloVe	Global Vectors
GRU	Gated recurrent unit
LM	Language model
LSTM	Long Short-Term Memory Network
NEG	Negative Sampling
NLP	Natural Language Processing
NSP	Next Sentence Prediction
OOV	Out-of-Vocabulary
PG	Pointer Generator
RL	Reinforcement Learning
RNN	Recurrent Neural Network
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
RR	Repetition Rate
Seq2seq	Sequence-to-sequence
SOTA	State-of-the-art
ULM-FiT	Universal Language Model Fine-tuning for Text Classification

1. Introduction

1.1. Motivation

Summarizing is the ability of writing a brief summary of the essential content given in a text. Humans use their literacy to understand the overall meaning of the text, identify crucial parts and write the summary in their own words. Driven by the rise of the world wide web, however, the amount of publicly available texts has been rising sharply. The overwhelming extent of resources reaches the limits of human abilities to process the available data. In this context, *automatic summarization systems* have a great potential to compress texts and to aid users to focus on the essentials.

Two types of approaches for automatic summarization systems can be distinguished. *Extractive methods* aim to identify the crucial information of a written text and solely copy these parts as summary [15, 80]. As a result, longer sequences of words in a summary are usually not fluid and easily readable text. To encounter this, *abstractive methods* aim to express the summaries in coherent and fluent text [75, 61]. However, teaching a computer to summarize with abstractive methods is a complicated task. This requires a brief introduction to the field of Natural Language Processing (NLP).

The understanding and processing of natural language is still a challenging task in artificial intelligence. Computational techniques are applied to understand single characters and words, the connection between words and the big picture of a text. On top of this, computers cannot interpret words as they are but have to transfer them to numerical representations. Fortunately, the developments in the field of machine learning show promising results in learning machine-readable word representations [58, 64].

In the field of text summarization, recent abstractive methods profit from the rise of deep neural networks [81]. First, neural approaches learn deep representations to understand the overall meaning of the text and to identify crucial parts. Besides this, the second challenge in abstractive text summarization is to write a summary with sequences of words in a fluent style. The neural approaches in abstractive text summarization face this challenge by writing new sequences word-by-word and are thus categorized as *sequence-to-sequence models*. This type of models has recently been addressed with neural networks by separating the reading (encoding) of words from the writing (decoding) of words [11].

From another point of view, a summarization system is optimized for the objective of a *single* task. In contrast to this, a question answering system, for instance, is only applicable to understand and answer questions. Regarding the humans' behavior, this workflow would be on par with learning the literacy and the task-specific skills from scratch. Consequently, the human ability to transfer learnings is essential to solving unseen and new tasks in natural language.

In order to be able to reuse previously learned knowledge, *transfer learning* methods share beneficial information across multiple tasks. In other fields of machine learning like computer vision, transfer learning for image classification with *ImageNet* [18] has become a widespread workflow [7, 70]. For natural language, *word embeddings* compress the sparse input data and capture the meaning of words in a dense representation [58, 64]. Even though these representations cannot overcome the obstacle of learning the task- and domain-specific knowledge from scratch, word embeddings provide a better starting point for the initial layer in neural networks.

Going one step further, recent approaches in NLP transfer deep neural networks for multiple text classification tasks [65, 68, 19]. The large-scale models are trained once and subsequently adapted to several tasks with varying objectives like question answering or natural language inference (NLI) [19]. This process is referred to as *sequential transfer learning* [73]. Since approaches in sequential transfer learning incorporate a deep language understanding across multiple tasks, they have great potential to address the challenges in text summarization.

1.2. Research objectives

This thesis studies the abilities of transfer learning for the task of text summarization. The main hypothesis of this thesis is the following:

> *Sequential transfer learning with deep neural networks in natural language processing promotes the task of abstractive text summarization.*

The following research questions are asked to address this hypothesis:

1. How do recent approaches address transfer learning for NLP?
2. How do deep neural networks achieve the state-of-the-art in text summarization?
3. Can the task of text summarization benefit from sequential transfer learning?

1.3. Course of investigation

This thesis is divided into seven chapters: After this introduction and the definition of the research objectives, Chapter 2 describes the theoretical fundamentals in machine learning, neural networks, and NLP. Subsequently, Chapter 3 addresses the first research question by presenting a taxonomy of transfer learning for NLP and recent approaches in the field. Afterwards, Chapter 4 focuses on the second research question by defining abstractive text summarization and introducing related work.

Chapter 5 describes the approach of the practical part of this thesis. In this context, the implementation of the neural summarization model is presented, and the experimental setup for the following chapter is provided. Based on the state-of-the-art in transfer learning and text summarization as well as the baseline summarization model, Chapter 6 addresses the third research question. For this reason, experiments analyze and discuss the influence of differently sized datasets, self-attention, pre-trained word embeddings and recent transfer learning approaches on the generation of summaries.

In conclusion, Chapter 7 reflects the presented findings and outlines directions for further research.

2. Theoretical foundations

This chapter provides the theoretical foundations for this thesis. The first section shortly introduces NLP and machine learning and defines the relevant fields for this thesis. The subsequent sections discuss approaches for the tasks of language modeling and word embeddings. These are particularly useful for sequential transfer learning in NLP (see Chapter 3). The following sections explain recurrent sequence-to-sequence (seq2seq) models, attention in general, and the improvements of attention for seq2seq models. Finally, Section 2.6 introduces the fully attention-based Transformer model.

2.1. Background

This section introduces machine learning and natural language processing (NLP). As these are only explained *very briefly*, this thesis presupposes knowledge of the basics in the respective field. Consequently, the aim of the following sections is not to introduce the theory but to define the notation and terms for the remainder of this thesis.

2.1.1. Machine learning

Machine Learning is the subfield of artificial intelligence which deals with teachning machines to learn from experience. Mitchell [60] defines, *"A computer is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E"*. In conventional programming, engineers craft rules and programs to compute answers for input data. Machine learning, however, leads to a paradigm shift such that models learn the rules to achieve a task based on experience given through data and answers.

In machine learning, a model receives n *input* instances of a dataset $\mathcal{D}$ in the form of vectors $x_i \in \mathbb{R}^d$ with dimensionality d. For *supervised learning*, the training target of x_i is given through its label y_i. In *unsupervised learning*, however, the dataset has no labels. The machine learning approaches in this thesis are *classification* tasks which interpret y_i as predefined *class* label. In contrast to this, y_i is a continuous value in *regression* which is out-of-scope for the remainder of this thesis.

Discriminative classifiers approximate the posterior probability $P(C_j \mid x_i; \Theta)$ for class C_j given the input x_i with a set of parameters Θ. During training, the model is optimized to predict the best probabilities given the gold target r_i. In this context, the objective function $J(\mathcal{D}, \Theta)$ measures the error between the predicted y_i and expected outcome r_i. The training goal is to minimize (or maximise) the objective function. In the scope of this thesis, the optimization method is *stochastic gradient descent* [41].

Neural networks A neural network is a framework for discriminative machine learning algorithms which inolves one or multiple layers with non-linear activation functions. The

concept was inspired by the function and connection of *neurons* in the human brain [73]. A neural network consists of an initial *input layer* and a final *output layer* which computes categorical distributions from the intermediate values with activation functions like the softmax or the sigmoid function. Deep neural networks are a subfield of neural networks consisting of muliple *hidden layers* between the input and output layer [3]. The idea of these stacked algorithms is that each successive layer contains a meaningful representation [12].

2.1.2. Natural Language Processing (NLP)

Natural Language Processing (NLP) is a set of techniques for teaching computers to understand and process natural language. Language consists of words which differ from the numerical values that computers process. Nonetheless, machine learning algorithms have led to promising results for a wide range of NLP tasks such as part-of-speech (POS) tagging, machine translation or text summarization [6, 87].

One challenge in NLP is the transfer of unstructured text to machine-readable representations. From a machine learning perspective, the input in NLP is a *text corpus* $\mathcal{D}$ consisting of *words* $(u_1, \cdots, u_N)$. These words are uniquely represented in the *vocabulary* $\mathcal{V}$ of length L with $(v_1, \cdots, v_L)$. The principal objective during encoding of words is to capture their meaning and the relation between words in rich *numerical* representations.

A *Bag-of-Words (BOW)* is a simple method to extract features from a text $\mathcal{D}$. In the first step, a vocabulary $\mathcal{V}$ is created which consists of unique n-grams of $\mathcal{D}$. In general, an n-gram is defined as a continuous sequence of n words in a text. Single words ($1-$grams) are called *unigram*. In the second step, the bag-of-words scores the occurrences of words. These scoring functions are often based on counts or frequencies. In a simple example, a bag-of-words might contain the counts of occurrences of unique words (*unigrams*). For information retrieval in language, however, the occurrence of multiple (n) words in a sequence may be important. Extending the approach above, a bag-of-words with n-grams may count the occurrences of a sequence of n words in $\mathcal{D}$.

2.2. Language modeling (LM)

Manning, Raghavan, and Schütze [54] define, *"A language model is a function that puts a probability measure over strings drawn from some vocabulary"*. From the probabilities of a sequences of words $P(u_1, \cdots, u_N)$, a language model derives the probability $P(u_i \mid u_1, \cdots, u_{i-1})$ of a word u_i given a sequence of previous words $(u_1, \cdots, u_{i-1})$. This is due to the fact that the conditional probability can be expressed as the probabilities of two subsequences [26].

N-gram models One fundamental kind of language models counts n-grams and outputs the most likely word based on the frequency of these n-grams [10]. Therefore, the probability $P(u_1, u_2, \cdots, u_N)$ of a sequence can be approximated by the product of the probability of each word u_i given $n-1$ previous words as

$$P(u_1, \cdots, u_N) = \prod_{i=1}^{N} P(u_i \mid u_{i-(n-1)}, \cdots, u_{i-1}). \tag{2.1}$$

Let $count_n(s)$ be a function that counts the n-grams of a sequence s. The conditional probability can be approximated from the observed frequencies of an n-gram model as

$$P(u_i \mid u_{i-(n-1)}, \cdots, u_{i-1}) = \frac{count_n(u_{i-(n-1)}, \cdots, u_{i-1}, u_i)}{count_n(u_{i-(n-1)}, \cdots, u_{i-1})}. \tag{2.2}$$

However, *n-gram models* are very sparse, and as k increases, the captured data is not sufficient to generate the next word. Furthermore, a specific n-gram which misses in the training data is unkown during inference. For this reason, *smoothing* techniques are a common practice to create more robust n-gram based models [10]. However, this thesis focuses on a different type of language models: *neural language models*.

Neural language models Generally speaking, neural networks are able to learn distributed representations. Neural language models use this ability and learn continuous-valued representations that fight the course of dimensionality [5]. The course of dimensionality in language modeling refers to the fact that the number of potential sequences of words grows with the size of the vocabulary. For instance, a sequence with 10 words from a vocabulary of size 100,000 results in 10^{50} possible sequences of words [4]. In this context, distributed representations provide a better generalization since unseen sequences of words may have similar features to known sequences from training.

Bengio et al. [5] propose a *probabilistic language model* as one layer feed-forward neural network (FFNN). Similar to the n-gram model beforehand, the neural language model approximates a probability for a word given its context of $n-1$ previous words, i. e. $P(u_i \mid u_{i-(n-1)}, \cdots, u_{i-1})$. However, in the approach by Bengio et al. [5], the FFNN requires a pre-defined and fixed-length context which typically includes only five to ten previous words [59]. *Recurrent Neural Language Models (RNN LM)* overcome this deficiency with recurrent states covering much longer contexts [59]. Recent approaches with recurrent and self-attention based language models are introduced later in this thesis.

Training objective and perplexity In contrast to other tasks such as sentence classification or translation, language models do not have a particular task like predicting a class label or translate a word. On top of this, language models can be very extensive with millions of input words. Thus, these models are evaluated with their *perplexity* during training and testing. Perplexity is a widely-known metric from information theory to evaluate probability distributions. In machine learning and NLP, the perplexity is a measure of how well the model predicts samples and is closely related to the cross-entropy loss.

Let an n-gram based language model have the vocabulary size N, an approximation of the *cross-entropy* as objective function $J(\Theta)$ is

$$J(\Theta) = -\frac{1}{N}\sum_{i=1}^{N} \log P(u_i \mid u_{i-(n-1)}, \cdots, u_{i-1}) \tag{2.3}$$

where $P(u_i \mid u_{i-(n-1)}, \cdots, u_{i-1})$ is the probability distribution of a neural network with the parameters Θ. The perplexity is $b^{J(\Theta)}$ where b is the base of the log. For the binary cross entropy above, $b = 2$ and the perplexity is simply $2^{J(\Theta)}$.

Regarding language modeling as NLP task in the scope of this thesis, the objective is to capture a sequence of words and predict the next word or character with the highest

probability. This objective makes language modeling one of the few *unsupervised learning* tasks in NLP. The training target, respectively the next word, is directly given from the text corpus. Since resources of plain text are overwhelming in the web, language models are trained on a large text corpus. Nowadays, common datasets have between millions [57] and a billion [9] training words. Large-scale language models on these datasets are the powerhouse of pre-trained models in NLP and discussed in Chapter 3.

2.3. Word embeddings

As explained in the previous sections, computers require the transformation of free text into numerical representations. The most straight-forward approach, *one-hot encoding*, represents words as atomic vectors in a high-dimensional space. One-hot encoded representations are unique vectors with only 0's and a single 1. For instance, a vocabulary with L distinct words has L representations such that $v_1 = [\mathbf{1}, 0, \cdots, 0]$, $v_2 = [0, \mathbf{1}, 0, \cdots, 0]$ and $v_L = [0, \cdots, 0, \mathbf{1}]$. However, the human vocabulary consists of a vast amount of distinct words resulting in extremely high-dimensional and sparse representations. Secondly, distances between representations are identical and do not provide information about the linguistic similarity of words in the vocabulary [58, 64].

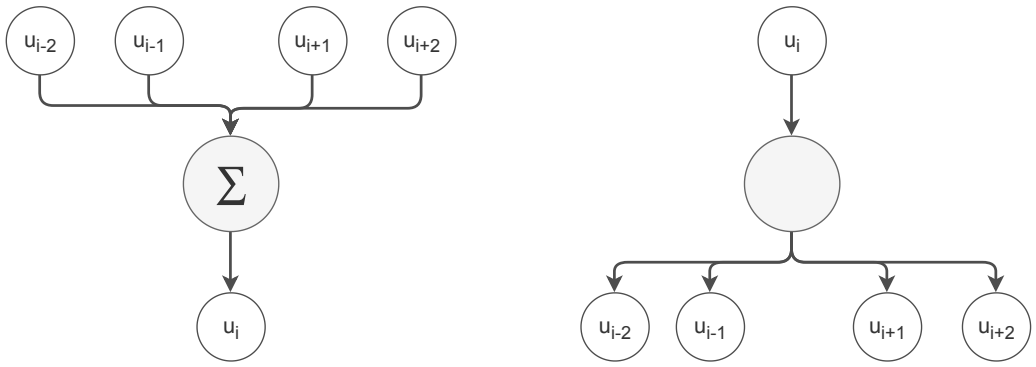
Background Embeddings overcome this issue by representing words as dense vectors in a *Continuous Vector Space Model*. There are two different methods of word embeddings which aim to reflect the meaning of a word. First, *count-based methods* use statistics such as how often words co-occur in a specific context and create embeddings by reducing the dimensionality of the collected data. Secondly, *context-predicting models* learn the word vectors directly by optimizing the abilites to predict a word vector in a given context [2].

Reference to language models The task of language modeling has the objective to predict a word given its context (see Section 2.2). Valuable word representations also require an understanding of the word in different contexts. Consequently, language models and word embeddings are deeply-connected and recent approaches [55, 65] extract the embeddings directly from the internals of trained language models (see Section 3.2). Due to computational reasons, traditional word embeddings and context-predictive models are not directly derived from language models [58]. The following section briefly introduces two common word embedding models: *word2vec* and *GloVe*.

2.3.1. Skip-gram and Continuous-Bag-of-Words (CBOW)

Mikolov et al. [58] introduce two different models: The *skip-gram model* and the *Continuous-Bag-of-Words (CBOW)*. Both models compute word embeddings by operating on a local context which is restricted to a fixed window size $2n + 1$. Further, both approaches slide this window over the text corpus and distinguish between $2n$ context words and a single center word u_i. However, the models follow different objectives which are illustrated in Figure 2.1 and described below.

(a) Continuous-Bag-of-Words (CBOW) The objective of the CBOW model is similar to neural language models. As illustrated on the left side of Figure 2.1, the model aims to predict the center word u_i based on the sum of context words. In contrast to unidirectional



(a) CBOW model (b) Skip-gram model

Figure 2.1.: Continuous-Bag-of-Words (CBOW) and Skip-Gram model [58].

language models, the model does also take the future context $(u_{i+1}, \cdots, u_{i+n})$ into account. Following the notation of Equation 2.3, the training objective $J(\Theta)$ can be defined as

$$J(\Theta) = -\frac{1}{N}\sum_{i=1}^{N} \log P(u_i \mid u_{i-n}, \cdots, u_{i-1}, u_{i+1}, \cdots, u_{i+n}). \tag{2.4}$$

(b) Skip-gram The right side of Figure 2.1 illustrates the *skip-gram* model. In contrast to the CBOW, the skip-gram model has an inverted objective to predict the context words given a single center word u_i. Hence, the objective function $J(\Theta)$ can be defined as

$$J(\Theta) = -\frac{1}{N}\sum_{i=1}^{N} \left(\sum_{-n \geq j \geq n, \neq 0} \log P(u_{i+j} \mid u_i) \right) \tag{2.5}$$

word2vec Skip-gram and CBOW are shallow neural networks that compute the probability distribution with a softmax function. However, this softmax operation requires the most computational time in the model and has constrained word embeddings for years [64]. Thus, a significant contribution of the above approach relies on the successful approximation of the softmax [58]. This thesis will, however, not cover the approximation of the softmax in detail. The skip-gram model with an approximated softmax using *negative sampling (NEG)* [33] on a large corpus resulted in the publicly available *word2vec* embeddings.

2.3.2. GloVe (Global Vectors)

Another approach by Pennington, Socher, and Manning [64] implements a *count-based* model to capture the meaning of words regarding the entire corpus. Generally speaking, *Global Vectors (GloVe)*) are representations which rely on information given by ratios of co-occurrence probabilities. In the first step, the model iterates over a text corpus and counts the global co-occurrences of words. These are collected in a co-occurrence matrix X in which each cell X_{ij} captures how often a word i appears in the context of j. Further, i and j are part of a predefined vocabulary.

In the second step, the co-occurrences matrix X is compressed to vector values in continuous space. Pennington, Socher, and Manning [64] propose a weighted least squares objective which minimizes

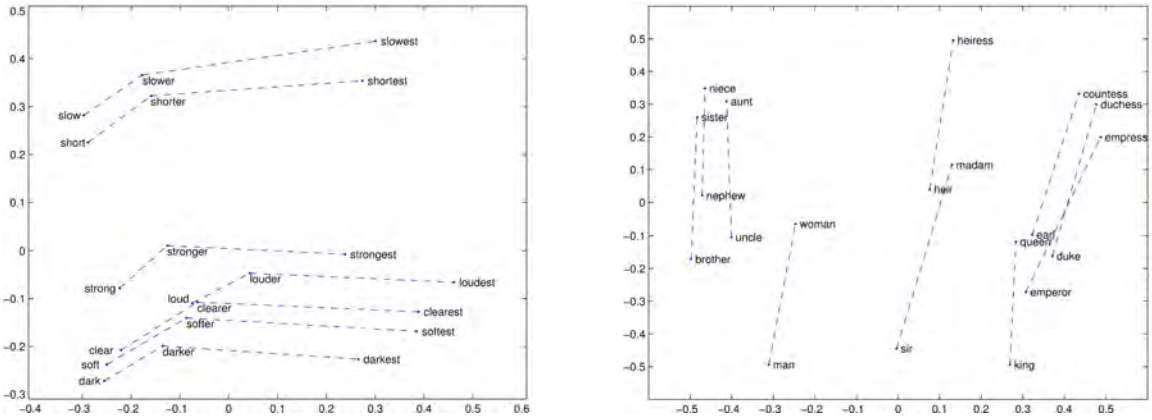
$$J(\Theta) = \sum_{i,j=1}^{N} f(X_{ij})(w_i^\top \widetilde{w}_j + b_i + \widetilde{w}_j - \log X_{ij})^2 \tag{2.6}$$

where w_i is the word vector with bias b_i for word i and $\widetilde{w}_j$ is the context-vector with bias $\widetilde{b}_j$ of word j. On top of this, the weighting function f assigns lower weights to uncommon and noisy word pairs. With W and $\tilde{W}$, the model generates two sets of word vectors which are supposed to perform equally if X is symmetric [64]. The GloVe model has been trained on varying sized datasets from one up to 42 billion (Common Crawl) tokens of data. Except for Common Crawls, the model builds a vocabulary of the 400,000 most frequent words [64].

2.3.3. Evaluation and interpretation

Another reason for the popularity of word2vec and GloVe is their interpretability. First, the cosine distance between two word vectors is a useful metric to measure the relatedness of words in the vector space [58]. The nearest neighbors of a target word refer to the word vectors with the highest similarity to the target word. These neighbors tend to be related in their meaning. For instance, the four nearest neighbors of the word frog in the pre-trained GloVe embeddings are frogs, toad, liteoria, leptodactylidae[1].

Secondly, Figure 2.2 shows linear patterns observed from a two-dimensional plot of GloVe embeddings. The left side of Figure 2.2 shows the relation of comparatives and superlatives like `slow`, `slower` and `the slowest`. Interestingly, the GloVe embeddings encapture valuable information about the word analogy since the distances are very similar for different compartives and superlatives. As illustrated on the right side of Figure 2.2, GloVe embeddings also capture content-specific values like gender-specific data. Following the word vector from



(a) Comparative-Superlative

(b) Gender-specific

Figure 2.2.: 2-dimensional plot of patterns in GloVe embeddings (https://nlp.stanford.edu/projects/glove/).

[1] https://nlp.stanford.edu/projects/glove/

`king` and adding the distance of the word `woman` ends up to be near the vector of the word queen, i. e. `king + woman = queen`. These patterns and other aspects of traditional word embeddings have been thoroughly investigated [45, 77].

Summary The advances of word embeddings are obvious. Instead of learning patterns over and over again, word embeddings are a meaningful starting point for machine learning algorithms in NLP. Furthermore, pre-trained word embeddings such as GloVe and word2vec have been made available for the public. This is essential for tasks with small datasets that lack sufficient data to extract complex language patterns. In the scope of this thesis, word embeddings are the first approach to transfer learnings from one task in NLP to another task. Thus, Section 6.4 shows the benefits of GloVe embeddings for the task of text summarization.

2.4. Sequence-to-sequence tasks

Sequence-to-sequence (Seq2seq) tasks, including language modeling, machine translation or text summarization, aim to generate a new output sequence for a given input sequence. For language models (see Section 2.2), the output is typically given as a single character or word. In contrast to this, the task of abstractive summarization, which is the subject of Chapter 4, aims to summarize an input text in multiple words or even entire sentences. These seq2seq tasks imply that inputs and outputs can differ in length, i. e. have a *variable dimensionality*. However, neural networks require inputs and outputs to have a fixed dimensionality by design. Encoder-decoder models overcome this limitation and are described in the following sections.

2.4.1. Encoder-decoder models

A fundamental concept is the separation of two processes: *encoding* and *decoding*. While the encoder processes an input sequence and generates a fixed-length representation, the decoder uses this intermediate representation to generate output. An early approach by Cho et al. [11] uses a *Recurrent neural network (RNN)* as the encoder and another RNN as the decoder to address the task of neural machine translation. Figure 2.3 illustrates this encoder-decoder model.

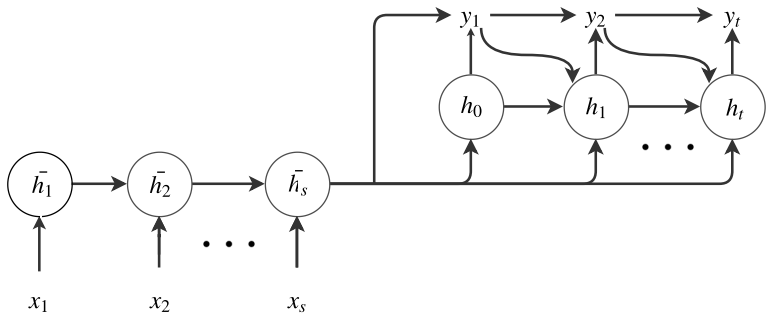


Figure 2.3.: Sequence-to-sequence processing with an encoder-decoder model [11].

The **encoder** receives an input sequence $(x_1, \cdots, x_s)$ and starts to compress the information of the first word in a fixed-length vector called *hidden state* $\bar{h}_1$. Subsequently, each x_i with

$i \in [2; s]$ is compressed in a respective hidden state $\bar{h}_i$ and takes the last hidden state $\bar{h}_{i-1}$ as well as the input x_i into account:

$$\bar{h}_i = f(\bar{h}_{i-1}, x_i) \tag{2.7}$$

where f is a non-linear activation function. By recurrently applying this encoding process until $i = s$, the last hidden state $\bar{h}_s$ captures the entire input sequence and is called *thought vector* $c = \bar{h}_s$.

The **decoder** has the responsibility to predict a new sequence step-by-step. For each decoding step i with $i \in [1; t]$ where t is the target length, the decoder creates a hidden state h_i and output y_i. Regarding the thought vector c, the last hidden state h_{i-1} and the decoded output y_{i-1}, the decoder's hidden state h_i is

$$h_i = f(h_{i-1}, y_{i-1}, \mathbf{c}). \tag{2.8}$$

Finally, the hidden state h_i is fed to another activation function $g(x) = \text{softmax}(x)$ to output the probability distribution

$$P(y_i \mid y_{i-1}, \cdots, y_1, c) = g(h_i, y_{i-1}, c). \tag{2.9}$$

The encoder and decoder are jointly trained by minimizing the conditional log likelihood

$$\min -\frac{1}{N} \sum_{i=1}^{N} \log P(y_i \mid x_i; \Theta) \tag{2.10}$$

where N the number of training samples and $P(x_i, y_i; \Theta)$ the probability of x_i given y_i approximated with a set of parameters Θ.

2.4.2. Bidirectional and deep LSTMs

Long Short-Term Memory Networks (LSTM) are a special kind of recurrent neural networks that address the problem of learning long-term dependencies in sequential data [36]. However, this thesis does not introduce the internals of LSTM cells but presupposes a fundamental understanding of LSTMs[2]. In the notation of recurrent cells from the Equations 2.7 and 2.8, LSTMs are treated as activation function f for a recurrent cell such that

$$h_i = \text{LSTM}(h_{i-1}, x_i). \tag{2.11}$$

Regarding the task of machine translation, Sutskever, Vinyals, and Le [81] use an encoder-decoder model with *multi-layer LSTM-based cells* to significantly outperform statistical translation methods. The core idea is similar to Cho et al. [11] (see Section 2.4.1) but improves these previous approaches for two reasons. First and as mentioned beforehand, LSTM cells have higher capabilities to capture long-dependencies than traditional recurrent networks [36]. Secondly, the approach builds on a deep-LSTM which stacks multiple LSTMs-layers on top of each other. Let L be the number of layers of a deep LSTM, each layer $l \in [1; L]$ contains hidden states in the form of

$$h_i^{(l)} = \text{LSTM}(h_{i-1}^{(l)}, h_i^{(l-1)}). \tag{2.12}$$

[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Each LSTM cell takes the output of the previous layer $h_i^{(l-1)}$ or respectively $h_i^{(0)} = x_i$ and the previous hidden state $h_{i-1}^{(l)}$ as input. Similar to Equation 2.9, a softmax computes the output probabilities with the top-most state of the last hidden layer $h_i^{(L)}$ [30].

Bidirectional RNN (BRNN) Generally speaking, a shortcoming of recurrence is their unidirectional context. A *Bidirectional RNN (BRNN)* overcomes this deficiency by taking the positive and negative time direction into account [78]. Given an input sequence $(x_1, \cdots, x_s)$, a BRNN processes the sequence forwards (x_1 to x_s) *and* backwards (x_s to x_1). In this context, the forward and backward direction are distinct processing layers and connected to the same output layer [28].

Applying bi-directionality to LSTMs, one LSTM processes x_1 to x_s in a forward hidden state $\overrightarrow{h}_i$ and another LSTM processes x_s to x_1 in a backward hidden state $\overleftarrow{h}_i$. The resulting hidden states $\overrightarrow{h}_i$ and $\overleftarrow{h}_i$ can be described as

$$\overrightarrow{h}_i = \text{LSTM}(\overrightarrow{h}_{i-1}, x_i) \tag{2.13}$$

$$\overleftarrow{h}_i = \text{LSTM}(\overleftarrow{h}_{i+1}, x_i) \tag{2.14}$$

and are often *shallowly concatenated* to a hidden state $h_i = [\overrightarrow{h}_i; \overleftarrow{h}_i]$ as output of the *Bidirectional Long Short-Term Memory Network (BiLSTM)* [48].

Deep BiLSTM In summary, Figure 2.4 illustrates a bidirectional and stacked LSTM model. At each layer $l \in [1; L]$, two distinct LSTM cells compute a forward hidden state $\overrightarrow{h}_i^{(l)}$ and backward hidden state $\overleftarrow{h}_i^{(l)}$ as input for the top-next layer $(l + 1)$. Finally, the forward and backward hidden states of the top-most layer ($\overrightarrow{h}_i^{L}$ and $\overleftarrow{h}_i^{L}$) are combined.

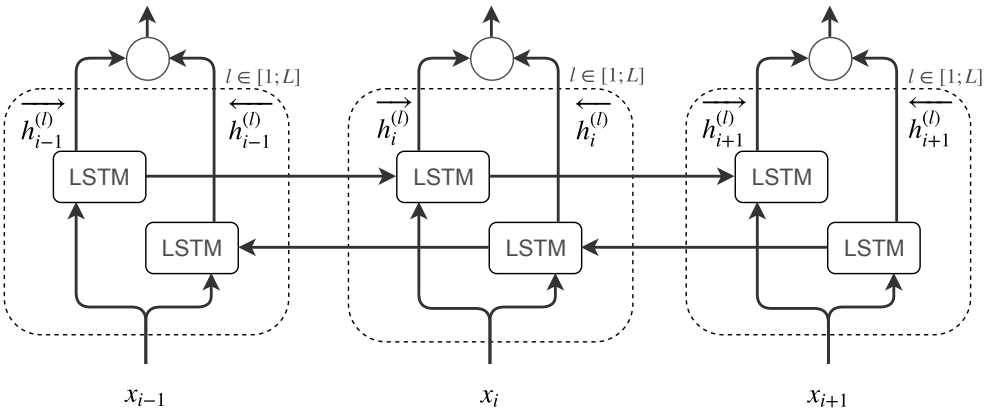


Figure 2.4.: Stacked (deep) bidirectional LSTM model.

Bidirectional and deep LSTMs are used in encoders of many seq2seq tasks in NLP. The stacked-bidirectional RNNs, especially with LSTM cells, are useful to capture the future context during encoding. In contrast to this, decoders generate outputs step-by-step where the future does not yet exist. For text summarization, several approaches use a single-layer biLSTM encoder and a single-layer LSTM decoder [63, 79, 44]. Chapter 3 describes these approaches in much more detail.

2.5. Attention

Encoder-decoder models as described in the previous section solve the problem of pre-defined and fixed sequence lengths. However, the strict separation between encoding and decoding leads to a single point of communication in the form of a *fixed-length thought vector*. From the perspective of how humans solve problems like summarizing a text, such a clear separation does not seem intuitive. Typically, humans incrementally mark words and sentences that contain important information instead of reading entire paragraphs without taking notes in one pass. Consequentially, a pure encoder-decoder model can't take advantage of the contextual information provided by the specific corresponding input tokens [1]. This is where *attention* comes into place.

2.5.1. Global attention

For the task of natural machine translation, Bahdanau, Cho, and Bengio [1] propose a model that benefits the decoder from learning to pay *attention* to parts of the encoded sequence (*memory*). More specifically, the decoder uses a mechanism to search for relevant parts in the input sequence to predict the next output. This kind of attention covers the entire memory of a model and is also called *global attention* [51].

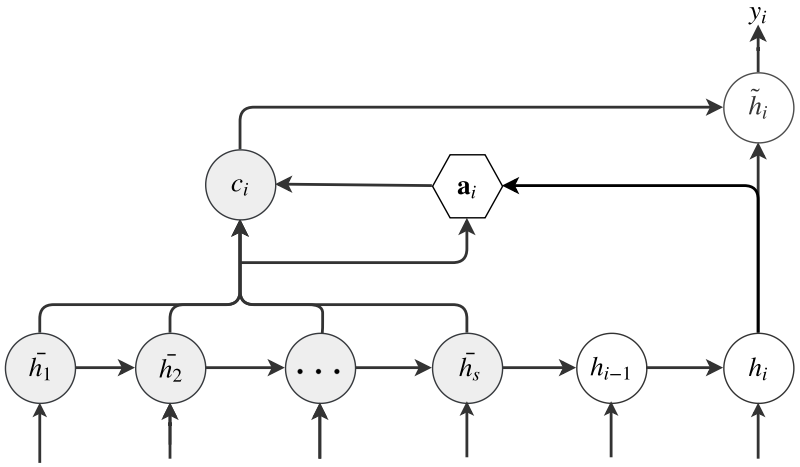


Figure 2.5.: Global attention illustrating the source (encoder) hidden states **(grey)** and target (decoder) hidden states **(white)** [51].

An attention-based encoder-decoder model for predicting the i-th output word y_i is illustrated in Figure 2.5. The model consists of source hidden states $(\bar{h}_1, \cdots, \bar{h}_s)$ and target (decoder) hidden states $(h_1, \cdots, h_t)$. This corresponds to the architecture of sequence-to-sequence models (see Section 2.4). In addition, an intermediate state $\tilde{h}_i$ is defined as

$$\tilde{h}_i = \tanh(W_c[c_i; h_i]) \tag{2.15}$$

where W_c is a learned parameter and c_i is the context vector. Expressed differently, the hidden state h_i and context vector c_i are jointly passed to a layer with a non-linear activation function (tanh).

The context vector c_i is the weighted sum of the input states and defined as

$$c_i = \sum_{j=1}^{s} a_{ij} \bar{h}_j \tag{2.16}$$

where an *attention weight* a_{ij} scores the j-th source state for the i-th decoding step. This attention weight a_{ij} is

$$a_{ij} = \text{softmax}(f_{attn}(h_i, \bar{h}_j)) \tag{2.17}$$

where an *attention function* f_{attn} calculates a normalized alignment score between the target hidden state h_i and a source hidden state $\bar{h}_j$. The two most common attention functions are explained below.

Additive attention Bahdanau, Cho, and Bengio [1] propose an attention function as a one layer feed forward network with

$$f_{attn}(h_i, \bar{h}_j) = v_a^\top \tanh(W_a[h_i; \bar{h}_j]) \tag{2.18}$$

where v_a and W_a are learned parameters. Beforehand, the function concatenates the last hidden state h_i with the corresponding source state $\bar{h}_j$.

Dot-product (multiplicative) attention Following the spirit of additive attention, Luong, Pham, and Manning [51] simplify the attention function as

$$f_{attn}(h_i, \bar{h}_j) = \begin{cases} h_i^\top W_a \bar{h}_j & \text{general} \\ h_i^\top \bar{h}_j & \text{dot-product} \end{cases} \tag{2.19}$$

where *general* has an additionally learned parameter W_a. While additive and multiplicative attention are similar in their theoretical complexity, the dot-product attention is more space-efficient and faster with optimized matrix multiplication operations [84].

2.5.2. Self-attention

Global attention benefits from looking back to the input sequence during the *decoding* in encoder-decoder models. In contrast to this, the *encoder* in models based on LSTMs either neglects the future context or uses separate forward and backward LSTMs (see Section 2.4.2). However, these two LSTMs do not share parameters and are shallowly combined in the end. *Self-attention* lets the sequence attend to its context and weights the relevance of the respective parts of the input. Hence, self-attention is not limited to encoder-decoder models.

Lin et al. [48] propose a self-attentive or intra-attentive model for sentence embeddings. Given a sequence of word embeddings $(x_1, \cdots, x_s)$, the objective is to create a meaningful representation of the entire sequence. First, Lin et al. [48] computes the hidden state $h_i = [\overrightarrow{h}_i; \overleftarrow{h}_i]$ at the i-th timestep with a forward and backward LSTM (see Section 2.4.2). Secondly, the approach incorporates self-attention by adapting the additive attention function (see. Equation 2.18). In contrast to applying attention to source and target states, let

$H = (h_1, h_2, \cdots, h_n)$ be a matrix obtaining n hidden states. The attention weights a_i from Equation 2.17 can be simplified to a single attention weight a as

$$a = \text{softmax}(v_a \tanh(W_a H^\top)) \tag{2.20}$$

and the single context vector c_i from Equation 2.16 to a single context vector c as

$$c = Ha^\top. \tag{2.21}$$

Further, Lin et al. [48] argue that multiple hops of attention are required to capture different components within a sequence. Thus, instead of learning a vector v_a, the approach introduces a learned matrix V_a. The corresponding attention matrix A is

$$A = \text{softmax}(V_a \tanh(W_a H^\top)) \tag{2.22}$$

and combined with the hidden states H such that

$$C = AH. \tag{2.23}$$

Finally, a penalization term P reduces the redundancy in A over multiple hops and ensures that the model focuses on different parts of the sequence [48]. P is defined as

$$P = ||(AA^\top - I)||_F^2 \tag{2.24}$$

where $|| \bullet ||_F^2$ is the squared Frobenius norm of a matrix [48].

Another implementation of self-attention, *multi-head attention*, is the powerhouse of the Transformer and thoroughly discussed in the following section.

2.6. The Transformer

At each step of generating a character or word, global attention encourages the decoder to give more weight to the important parts of the input sequence. This successful but computational-expensive layer [51] addresses the issue of fixed-length vectors between the encoder and decoder. On the other hand, however, the encoder learns to represent the input sequence without having benefits of global attention. For this reason, self-attention introduces a mechansim to aid the encoding of input sequences. Regarding these advances of attention mechanisms, the question raises whether seq2seq tasks are capable for both, global attention and self-attention. The answer is *The Transformer* [84].

2.6.1. Architectural overview

Going even further, the Transformer is an encoder-decoder architecture which is entirely based on attention without using recurrent neural networks. The model achieved state-of-the-art results in sequence-to-sequence tasks such as machine translation [84] and has applications in a variety of tasks in NLP (see Section 2.6.4). While the following section introduces the architecture and composition of the Transformer in general, the subsequent sections describe the model in more detail. Figure 2.6 illustrates the architecture of the encoder-decoder model.

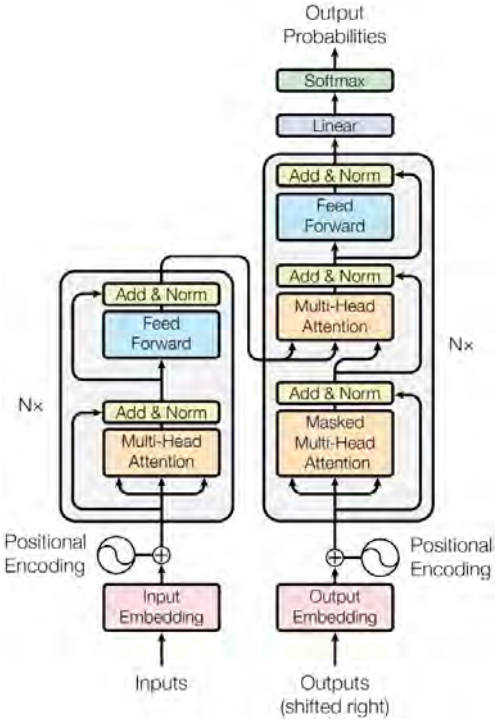


Figure 2.6.: The Transformer model with an encoder (left) and a decoder (right) [84].

Encoder An input sequence $\mathbf{x} = (x_1, \cdots, x_s)$ is embedded (red) as input for the Transformer encoder. Following a similar concept of deep LSTMs (see Section 2.4.2), the encoder consists of N stacked sublayers. Each of these N sublayers computes (1) the *multi-head attention* (orange), a form of self-attention, followed by (2) a *position-wise feed-forward* network (FFNN, blue). Further, a *residual connection* [34] surrounds each cell by combining the past results with new computations and normalizing the output (*Add & Norm*, yellow). The output of each layer is the input of the top-next layer, and the output of the final layer is the decoder's input. Listing 2.1 technically describes the Transformer encoder as pseudo-implementation.

```
layer_input = word_embedding(x) + positional_encoding(x)

for N:
  self_attn = multi_head_attention(layer_input)
  self_attn = norm(self_attn) + layer_input

  ffnn = feed_forward_nn(self_attn)
  layer_output = norm(ffnn) + self_attn
  layer_input = layer_output

encoder_output = layer_output
```

Listing 2.1: Pseudo-implementation of a Transformer encoder.

Decoder As illustrated on the right side of Figure 2.6, the shape of the Transformer decoder is similar to the encoder. Listing 2.2 shows a respective pseudo-implementation.

```
layer_input = word_embedding(y) + positional_encoding(y)

for N:
  masked_attn = multi_head_attention(mask(layer_input))
  masked_attn = norm(masked_attn) + layer_input

  global_attn = multi_head_attention(masked_attn + encoder_output)
  global_attn = norm(global_attn) + masked_attn

  layer_output = norm(feed_forward_nn(global_attn)) + global_attn
  layer_input = layer_output

decoder_output = layer_output
y = softmax(linear(decoder_output))
```

Listing 2.2: Pseudo-implementation of the Transformer decoder.

At the i-th decoding step, the model embeds the already generated words $(y_1, \cdots, y_{i-1})$ and feeds the embeddings to the Transformer decoder. Each layer applies (3) a *masked multi-head-attention* to encode these embeddings. The mask operation ensures that the self-attention ignores the future context $(y_i, \cdots, y_t)$, where t is the target length. Subsequently, each layer (4) looks back to relevant parts of the encoder which incorporates the concept of *global attention* in the Transformer. Finally, the results are fed to (5) a *feed-forward neural network* to compute the output of a layer. The top-most layer is the final output, and a linear layer with softmax classifier generates the output y_i.

2.6.2. Multi-head attention

The power of the Transformer is based on two types of attention. In order to separate the responsibilities in attention [17], each Transformer layer processes three vectors: keys $K \in d_k$, values $V \in d_v = d_k$ and queries $Q \in d_q$. The *scaled-dot-product attention* at the core of the Transformer modifies the dot-product attention (see Section 2.5) by scaling the result of the factor $\frac{1}{\sqrt{d_k}}$.

(a) Multi-head-attention

(b) Scaled Dot-product-attention

Figure 2.7.: Attention in the Transformer [84].

Hence, the attention function $Attention(Q, K, V)$ is defined as

$$Attention(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.25}$$

and Figure 2.7b illustrates the computations of the dot-product-attention.

Let $\mathcal{M}$ be the encoder's output and $\mathcal{I}$ denote the input of a Transformer layer (see `layer_input` in Listing 2.1 and Listing 2.2). On the one hand, the *global attention* between the encoder and decoder weights the relevance of the encoder's output ($V = \mathcal{M}$) for the current decoding step. The weight assigned to each value is calculated concering the encoder's output ($K = \mathcal{M}$) and the current input ($Q = \mathcal{I}$) [84]. On the other hand, many approaches [84, 19, 16] set $Q = K = V = \mathcal{I}$ to encode a sequence with *self-attention*.

Additionally, the model computes the attention operation with multiple heads. These attention heads are an implementation to include multiple hops of self-attention (see Section 2.5.2). Each of the N Transformer layers computes a $MultiHeadAttention(Q, K, V)$ consisting of $i \in [1; h]$ attention heads $head_i$

$$MultiHeadAttention(Q, K, V) = [head_1; \cdots; head_h]W^0 \tag{2.26}$$

where $W_0 \in \mathbb{R}^{d_{model} \times d_v}$ is a learned parameter. Each $head_i$ linearly projects the queries (QW_i^Q), keys (KW_i^K) and values (VW_i^V). These projections capture different parts of the sequence (see Section 2.5.2) and are the input of the actual attention function. Thus, a $head_i$ is defined as

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.27}$$

where the queries, keys and values are shards of dimensionalities $d_{k_i} = \frac{1}{h}d_k$, $d_{v_i} = \frac{1}{h}d_v$, $d_{q_i} = \frac{1}{h}d_q$. The Transformer employs $h = 8$ attention heads with a total dimensionality of $d_{model} = 512$ and $d_{k_i} = d_{v_i} = d_{q_i} = 64$ [84].

2.6.3. Positional encoding

The Transformer with pure self-attention comes with the drawback of losing the position of inputs in the sequence [84]. However, the position is important for several tasks in NLP, and a similar obstacle exists for neural networks with convolutional layers [24]. As illustrated in Figure 2.6, a *positional encoding* overcomes this limitation in the Transformer by including positional information to the word embeddings.

In general, positional embeddings can either track the relative or absolute position in the input sequence. Further, the embeddings can be fixed constraints or learned parameters during training. An earlier approach for convolutions in seq2seq models learns positional embeddings during training regarding their absolute position [24]. In contrast to this, the Transformer uses *sinusoidal functions* and relative positions. The functional approach aims to capture a more profound sense of the relative input positions with different frequencies of the *sin* and *cos* functions [84]. More precisely, the approach uses the two frequencies

$$\begin{aligned} PE_{(pos,2i)} &= \sin(pos/10000^{\frac{2i}{d}}) \\ PE_{(pos,2i+1)} &= \cos(pos/10000^{\frac{2i}{d}}) \end{aligned} \tag{2.28}$$

where *pos* is the absolute position of the input, $d = d_{model}$ the dimensionality of the model and i the dimension of the function. The hypothesis of relative positions is that these may scale better for large sequences [84]. Even though the proposal motivates for their sinusoidal approach, the results compared to learned embeddings are very similar [84].

2.6.4. Applications and developments of the Transformer

The Transformer by Vaswani et al. [84] has been applied and adapted to several problems. For language modeling, Al-Rfou et al. [72] show that a Transformer with $N = 64$ layers outperforms previous recurrent variants. For text summarization (see Chapter 4), Liu et al. [50] use a Transformer model to capture multiple input documents for the generation of text summaries. Furthermore, Chapter 3 introduces various approaches [68, 19, 69] of Transformer models for sequential transfer learning.

A further development of the Transformer addresses the obstacle of capturing large input texts. Even though self-attention has higher capabilities to capture long-term dependencies [72], the context-length remains fixed due to memory and computation limitations [16]. For this reason, Dai et al. [16] propose the *Transformer-XL*, a model sliding over the input, computing the self-attention for a fixed-length context and storing the results. Subsequent windows attend to previous contexts with a *recurrent layer* on top of the Transformer.

2.7. Conclusion

Machine learning in the field of NLP teaches machines to understand and process natural language. Approaches like the Bag-of-Words model count the occurrences of words and build a model based on their frequencies. Furthermore, neural approaches have become an important tool in a wide range of NLP tasks such as text classification, language modeling or machine translation.

Language modeling is the task of predicting the next word or character given the sequence of previous words or characters. Neural language models train on large datasets, oftenly extracted from the web [57]. Word embeddings have similarities to language models but the objective of creating meaningful representations of words. The pre-trained word embeddings [58, 64] are a starting point for NLP models.

Sequence-to-sequence tasks aim to understand and process a sequence and create new characters or words. These models commonly consist of an encoder which represents the input sequence and a decoder which generates new outputs. Various approaches use recurrent neural networks (RNNs) like LSTMs [36] and improve their capabilities by implying deep models with bidirectional contexts [30].

Furthermore, attention is a technique which follows the idea of learning to attend for specific parts of a sequence. Global attention improves the output generation because the decoder looks back to the relevant part of the encoder [51]. On top of this, self-attention learns to capture the meaning of a sentence [48]. Combining these two types of attention with seq2seq models, the Transformer solely bases on attention without recurrent states [84].

In conclusion, this chapter exerted the theoretical foundations of language modeling, word embeddings, seq2seq models, attention and the Transformer in the use-case of transfer learning for NLP (see Chapter 3) and the task of text summarization (see Chapter 4).

3. State-of-the-art in transfer learning for NLP

This chapter provides an overview of recent advances in pre-training, which show promising results across a wide range of tasks in NLP. Before introducing the relevant approaches, the first section classifies the approach of this work in the broad field of transfer learning. Based on this, Section 3.2 describes contextual embeddings which are an extension to context-free word embeddings (see Section 2.3). The subsequent sections explain the process of overcoming the obstacle that transfer learning is currently limited to word embeddings. These approaches like OpenAI GPT (see Section 3.3.2) and BERT (see Section 3.3.3) are essential for sequential transfer learning in NLP and the practical part of this thesis.

3.1. Introduction and demarcation

Generally speaking, transfer learning in NLP denotes the process of applying the model of a *source task* to a *target task*. In practice, different approaches exist, and Ruder [73] defines a taxonomy for transfer learning.

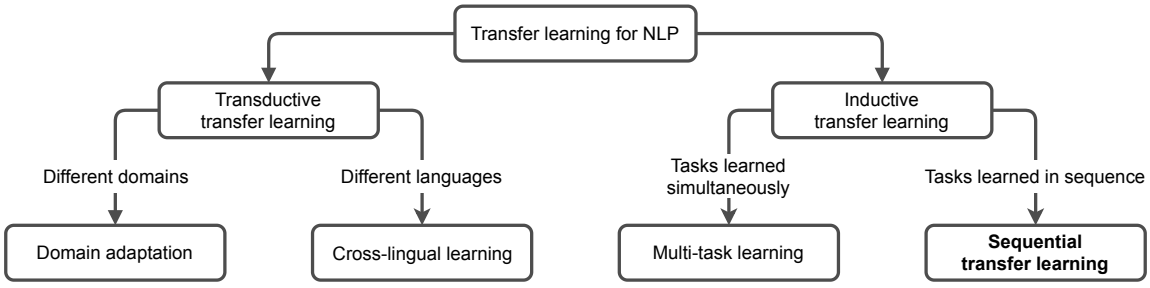


Figure 3.1.: Taxonomy for transfer learning in NLP [73].

As illustrated in Figure 3.1, transfer learning in NLP is addressed in two ways. *Transductive transfer learning* categorizes methods in which the source and target task have the same objective. While *cross-lingual learning* transfers the information to a different language [74], *domain adoption* applies the model to another domain [39]. For *inductive* transfer learning, however, the objective of the source task is different from the target task. The goal of *sequential transfer learning* is first to train a model on one task and then adopt the model to another task. The adaptation differs from the second type, *multi-task learning* [14], in which two or more tasks are jointly learned at the same time.

This chapter focuses on recent and promising approaches for *sequential transfer learning*. Furthermore, the practical part (see Chapter 5 and Chapter 6) includes these approaches in the task of text summarization. Consequently, other applications of transfer learning are out-of-scope for this thesis.

Stages of sequential transfer learning Sequential transfer learning consists of the following two stages [73]:

1. **Pre-Training** - A model is trained for the *source task* on a large dataset with high computational costs and long training times. A common objective of the source task is to be very generic and applicable to many target tasks.
2. **Fine-Tuning** - The pre- trained model is transferred to a *downstream (target) task*[1] with much fewer data. This phase is much shorter, and target tasks can benefit from an extensively pre-trained source model.

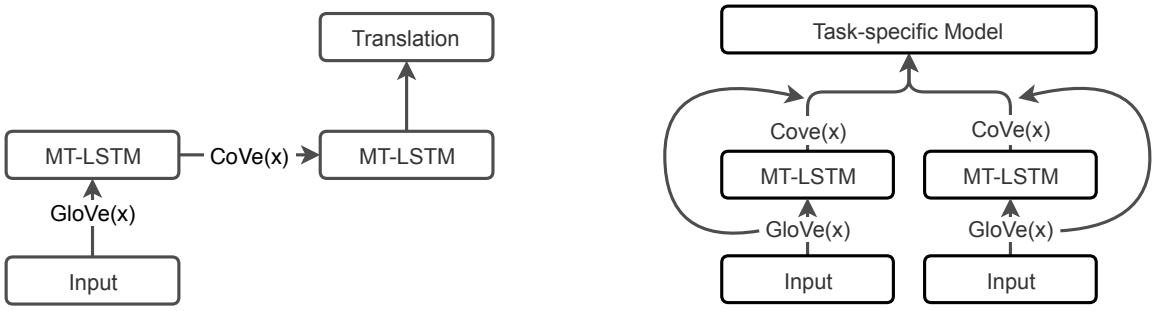
The remainder of this chapter provides a recent history of sequential transfer learning in NLP using deep neural networks.

3.2. Contextual embeddings

As introduced in Section 2.3, word embeddings such as GloVe or word2vec produce semantic representations of words. These models iterate over an entire text corpus to generate exactly one word representation for each unique lexical token. However, this neglects the inherent ambiguity. For instance, the word *rock* can either mean a stone or a genre of music, depending on its context. This word representation remains the same in both cases for *context-free* word embeddings such as GloVe. In contrast to this, the following section introduces recent approaches that aim to overcome this limitation by creating *context-dependent* word embeddings.

3.2.1. Contextual Word Vectors (CoVe)

The encoder in sequence-to-sequence-models processes input sequences to meaningful representations as foundation for the decoder's predictions (see Section 2.4). For this reason, encoders seem to be a sensible and feasible solution to capture the contextual information of a word. Furthermore, language models that aim to predict a word given its context are in the core comprehensive encoders (see Section 2.2). McCann et al. [55] follow this



(a) Pre-training: Encoder-decoder for machine translation.

(b) Fine-Tuning: CoVe with GloVe as input for a task-specific model.

Figure 3.2.: The two stages of contextual vectors (CoVe) [55].

[1] A downstream task denotes a supervised target task like text classification or text summarization.

idea by extracting the deep LSTM encoder of an attentional seq2seq-model trained for machine translation. The authors claim that extracting contextual information of the pre-trained encoder leads to various improvements for tasks like sentiment analysis and question answering.

As illustrated in Figure 3.2, the proposed model has two separate steps. At first, *pre-training* is the training of an encoder-decoder model with global attention for machine translation. The encoder consists of bidirectional LSTM cells whereas the decoder uses unidirectional LSTMs. The details are out-of-scope for this thesis but refer to a plain architecture of encoder-decoder models (see Section 2.4). During pre-training, the encoder and decoder solely train on a dataset for machine translation. After the model performs sufficiently well, the encoder's output is re-used to extract *Contextual word vectors (CoVe)*.

More specifically, let $\mathbf{x} = (x_1, \cdots, x_L)$ be a sequence of L words and GloVe($\mathbf{x}$) the corresponding sequence of pre-trained word embeddings. The model calculates a sequence of L context vectors CoVe($\mathbf{x}$) as

$$\text{CoVe}(\mathbf{x}) = \text{MT-LSTM}(\text{GloVe}(\mathbf{x})) \tag{3.1}$$

where MT-LSTM is the encoder of the machine translation. To further increase the influence of GloVe embeddings, McCann et al. [55] recommend to concatenate the context vectors with the GloVe embeddings during fine-tuning. Thus, a sequence of final embeddings $\mathbf{e} = (e_1, \cdots, e_L)$ is

$$\mathbf{e} = [\text{GloVe}(\mathbf{x}); \text{CoVe}(\mathbf{x})]. \tag{3.2}$$

In summary, the approach by McCann et al. [55] uses the *final* output of an LSTM encoder to extract contextual information of words. Even though the approach shows promising results, a primary obstacle for sequential transfer learning remains. During pre-training, the encoder learns to interpret datasets for the specific task of machine translation.

3.2.2. Embeddings from Language Models (ELMo)

Peters et al. [65] generalize the previously introduced approach to learn contextual embeddings. In contrast to McCann et al. [55], the approach is based on a *bidirectional language model (biLM)* to generate the *Embeddings from Language Models (ELMo)*. As a reminder to Section 2.2, language modeling is the fundamental task to approximate the probability for the next word given its preceding words. These language models are trained on much larger datasets compared to supervised models like machine translation. Further, the task of predicting the next word given its context is related to some approaches of context-predictive word embeddings (see Section 2.3).

Bidirectional language model (biLM) Respecting both, the history and future contexts is a common concept in sequence-to-sequence models (see Section 2.4.2). In Section 2.2, n-gram based language models have the objective to predict a word u_i given its preceding words $(u_1, \cdots, u_{i-1})$. This is also referred to as *forward language model*. In contrast to this, an n-gram based *backward language model* approximates the probability of a sequence

$P(u_1, \cdots, u_n)$ by the product of the probability of each word u_i given $n-1$ subsequent words:

$$P(u_1, u_2, \cdots, u_N) = \prod_{i=1}^{N} P(u_i \mid u_{i+1}, \cdots, u_{i+(n-1)}). \tag{3.3}$$

Altogether, a *bidirectional language model (biLM)* is based on the combined probabilities of a forward and backward model.

Peters et al. [65] implement such a biLM with multiple layers of LSTM cells (see Figure 3.3). Following the notation from Section 2.4.2, let u_i be a word at position i and x_i the word embedding of u_i. The bidirectional language model computes a forward hidden state $\overrightarrow{h}_i^{(l)}$ and a backward hidden state $\overleftarrow{h}_i^{(l)}$ for each layer $l \in [1, \cdots, L]$ as

$$\overrightarrow{h}_i^{(l)} = \text{LSTM}(\overrightarrow{h}_i^{(l-1)}, \overrightarrow{h}_{i-1}^{(l)}) \tag{3.4}$$

$$\overleftarrow{h}_i^{(l)} = \text{LSTM}(\overleftarrow{h}_i^{(l-1)}, \overleftarrow{h}_{i+1}^{(l)}). \tag{3.5}$$

For each word u_i, the LSTM cells takes the output of the previous layer $\overrightarrow{h}_i^{(l-1)}$ (or $\overleftarrow{h}_i^{(l-1)}$) or respectively the embedded input word $\overrightarrow{h}_i^{(0)} = \overleftarrow{h}_i^{(0)} = x_i$ as input. Furthermore, the LSTM takes the preceding $\overrightarrow{h}_{i-1}^{(l)}$ or respectively the subsequent hidden state $\overleftarrow{h}_{i+1}^{(l)}$ into account.

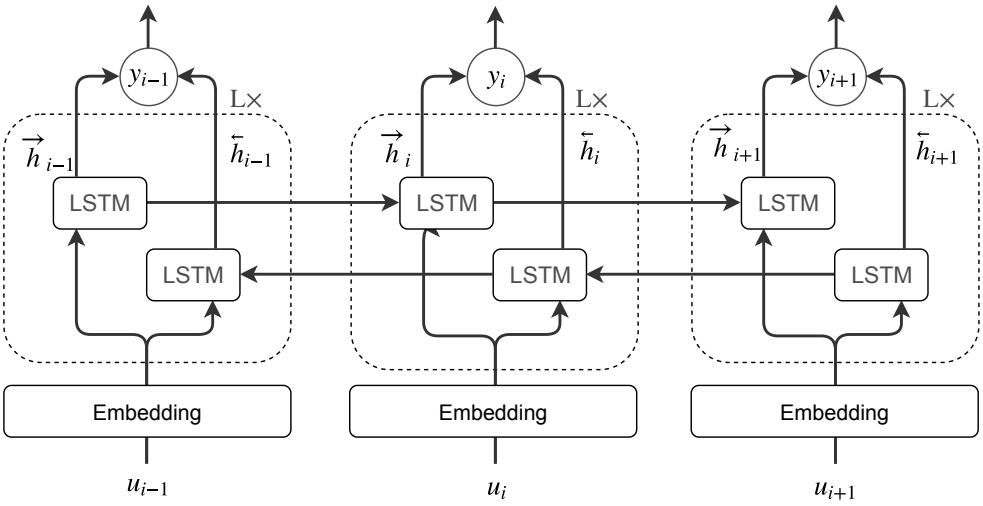


Figure 3.3.: ELMo model with stacked-bidirectional LSTM cells.

For the task of language modeling during **pre-training**, the final hidden states of the forward ($\overrightarrow{h}_i^L$) and the backward model ($\overleftarrow{h}_i^L$) are softmax-normalized to output two probability distributions for y_i. In this context, the neural networks have distinct parameters for the forward ($\overrightarrow{\Theta}_{LSTM}$) and backward model ($\overleftarrow{\Theta}_{LSTM}$). Nevertheless, the approach shares parameters for the final softmax layer (Θ_s) and the embeddings (Θ_x) [65]. Let $\mathcal{D}$ be a text corpus during pre-training, the training objective $J(\mathcal{D}, \Theta)$ of the overall biLM is to jointly minimize the negative log-likelihood of the forward and backward model with

$$J(D,\Theta) = -\frac{1}{N}\sum_{i=1}^{N}((\log P(u_i \mid u_1,\cdots,u_{i-1};\Theta_x,\overrightarrow{\Theta}_{LSTM},\Theta_s) \tag{3.6}$$

$$+ (\log P(u_i \mid u_{k+1},\cdots,u_N;\Theta_x,\overleftarrow{\Theta}_{LSTM},\Theta_s)). \tag{3.7}$$

Fine-Tuning After pre-training the forward and backward language models, the final softmax layer is discarded and the hidden states of each layer are concatenated such that $h_i^{(l)} = [\overrightarrow{h}_i;\overleftarrow{h}_i]^{(l)}$ is the output of layer l for the input word u_i. Subsequently, these layers of the biLM *jointly* capture the contextual information of a word. More specifically, the representation R_i of the word u_i is a set of hidden states and defined as

$$R_i = \{h_i^{(l)} \mid l = 1,\cdots,L\}. \tag{3.8}$$

Further, a task-specific weighting function $E(R_i;\Theta_e)$ collapses R_i into a fixed-length vector resulting in the final ELMo_i representation of word u_i as

$$\text{ELMo}_i = E(R_i;\Theta_e) = \gamma^{task}\sum_{j=0}^{L} s_i^{\text{task}} h_i^{(l)}. \tag{3.9}$$

Since each layer of the deep LSTM contributes to the final ELMo embeddings individually, the softmax-normalized parameter s^{task} weights the relevance of each layer. Furthermore, γ^{task} is a task-specific scaling factor [65]. Similar to the Context Vectors (CoVe) [55], the ELMo embeddings are combined with word embeddings for the integration into downstream tasks [65].

In summary, ELMo embeddings are pre-trained word representation extracted from a deep bidirectional LSTM for language modeling. These embeddings have high capabilities for a wide range of NLP tasks and improve the state-of-the-art in six different tasks from the field of text classification [65]. A distinction to previous work is that the embeddings are *deep* and extracted from all internal LSTM layers. Nonetheless, there are still two shortcomings. First, the embeddings are no standalone solution but require the above task-specific parameters [65]. Secondly, the high-dimensional representations are still compressed to *fixed-length vectors* coming with the cost of losing information [37].

3.3. Fine-tuning language models

Previous approaches in Section 3.2 demonstrate that contextual embeddings can be beneficial and enhance context-free embeddings on various NLP tasks [55, 65]. Nevertheless, CoVe and ELMo are yet an auxiliary input for the embedding layer of the corresponding downstream model. Besides this, these models are still trained from scratch which limits the capabilities of transfer learning in NLP. In many cases, the majority of parameters is randomly initialized, and the model requires large datasets with long training times to converge. In summary, the contextual embeddings are limited in being beneficial for tasks with smaller datasets.

3.3.1. A framework for pre-training and fine-tuning

The next step towards transfer learning in NLP is to reuse the entire pre-trained model. For instance, let a binary classifier detect spam emails and be based on a pre-trained language model using biLSTM cells. Instead of predicting the next word, the spam classifier adds a linear layer with softmax to output the probability of a given sequence to be classified as spam. In this particular example, the number of randomly initialized parameters is limited to the final layer of a model.

However, approaches in this field have been unsuccessful in recent history, mainly because a randomly initialized classifier on top of an extensive language model tends to catastrophic forgetting [37]. Thus, Howard and Ruder [37] propose a toolbox with methods to increase the generalization and to replace task-specific models with a single language model. This set of tools is called *Universal Language Model Fine-tuning for Text Classification (or: ULM-FiT)* and defines the following three stages for transfer learning in NLP.

1. **General-domain LM pretraining** in which a large language model is pre-trained on a large corpus like Wikitext-103 [57]. This is the most time-consuming part but is done only once for multiple tasks.
2. **Target task LM fine-tuning** adapts the general language model to the domain-specific data. For instance, for the task of sentiment analysis on movie reviews like IMDb dataset [52], the model specifies in the understanding and building sentences of movie reviews.
3. **Target task classifier fine-tuning** is the final step which integrates the output of the language models in the task-specific environment. For text classification tasks, a linear layer processes the output and a final hidden layer outputs the class probabilities.

Another important observation of Howard and Ruder [37] is the criticality of the final fine-tuning stage. Since new parameters are not pre-trained but randomly initialized, the risk of losing information is high [86]. *Gradual unfreezing* encounters this by freezing pre-trained layers. The layers are unfrozen once at a time, starting from the top-most layer [37]. This goes hand in hand with the *slanted triangular learning rates (STLR)* which first increases and then decays the learning rate to get a quicker conversion [37].

In summary, ULM-FiT is a conceptual work that introduces a framework to apply sequential transfer learning in NLP. Based on this idea, the following sections introduce specific approaches in the field of model fine-tuning.

3.3.2. Language modeling using a Transformer

The *Transformer* consists of multiple self-attention layers instead of recurrent cells (see Section 2.6). A multi-layer Transformer is *deeply* bidirectional compared to biLSTMs which use a *shallow concatenation* of a distinct forward and backward model. Furthermore, the underlying self-attention leads to promising improvements for tasks with long-term dependencies [84]. For the task of language modeling, an assumption is that the prediction of a word may benefit from capturing more information in a wider context.

Consequentially, the Transformer seems to have a high potential for language modeling and transfer learning in NLP. Unfortunately, the encoder-decoder architecture of the original

Transformer does not fit to the requirements of language models. However, the decoder of a Transformer approximates the probability of a subsequent word given a sequence of preceding words. For this reason, Radford et al. [68] extract a *Transformer decoder* and train it on a large text corpus for language modeling.

Language model (LM) During decoding, outputs are predicted word-by-word using the *history-only* because the future context logically does not yet exist. For this reason, the original Transformer masks out any potential future information in the decoder (see Section 2.6). As a result, OpenAI GPT implements a *unidirectional* language model.

Let $\mathcal{D}$ be a text corpus, u_i the input word at the i-th step and k be the size of the context window, the training objective is to minimize the negative log-likelihood with

$$J_{LM}(\mathcal{D}, \Theta) = -\frac{1}{N} \sum_{i=1}^{N} \log P(u_i \mid u_{i-k}, \cdots, u_{i-1}) \tag{3.10}$$

where $P(u_i \mid u_{i-k}, \cdots, u_{i-1})$ is conditional probability approximated by a neural network with parameters Θ.

Transformer decoder The architecture of the proposed Transformer decoder is illustrated in Figure 3.4. Given an input sequence $U = (u_1, \cdots, u_s)$ with *word embeddings* W_e and *positional embeddings* W_p (see Section 2.6), the first layer of the multi-stacked Transformer h^0 is

$$h^0 = UW_e + W_p. \tag{3.11}$$

Furthermore, the model consists of $L = 12$ layers with Transformer blocks computing the hidden state $h^{(l)}$ of each layer $l \in [1, L]$ with

$$h^{(l)} = \text{transformer\_block}(h^{(l-1)}). \tag{3.12}$$

Each layer performs masked-multi-head attention with $h = 12$ attention heads (see Section 2.6.2) followed by a position-wise feed forward network. The output of the final hidden layer is fed to a classifier such that the probability $P(u_i \mid u_{i-k}, \cdots, u_{i-1})$ is approximated as

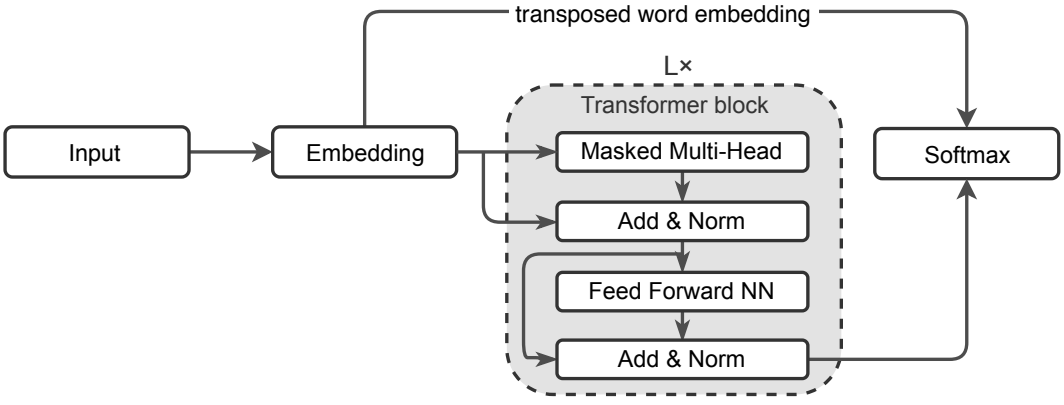


Figure 3.4.: Language modeling with a Transformer decoder model [68].

$$P(u_i \mid u_{i-k}, \cdots, u_{i-1}) = \text{softmax}(h^L W_e^\top). \tag{3.13}$$

Compared to the original Transformer in Section 2.6, the decoder does not require an encoder-decoder attention-layer. On top of this, the model is more comprehensive with $L = 12$ layers. Furthermore, OpenAI GPT handles longer sequences with 512 tokens and is trained for 100 epochs on the large *BooksCorpus* dataset [88, 68].

Fine-Tuning The pre-trained Transformer decoder model is adjusted for various downstream classification tasks. Let $\mathcal{C}$ be a labelled dataset for a text classification task. Each instance of $\mathcal{C}$ has an input sequence $U = (u_1, \cdots u_m)$ and a single label y. Hence, the input U is fed to the pre-trained Transformer, and output of the final hidden state in the last layer is h_m^L. The conditional probability $P(y \mid u_1, \cdots, u_m)$ of target y given the input $(u_1, \cdots, u_m)$ is

$$P(y \mid u_1, \cdots, u_m) = \text{softmax}(h_m^L W_y) \tag{3.14}$$

where W_y is the only learned parameter for the classification task. Furthermore, the fine-tuning model follows the same objective as the language model to minimize the negative log likelihood. Thus, $J_{\text{CLS}}(C, \Theta)$ is

$$J_{\text{CLS}}(\mathcal{C}, \Theta) = -\frac{1}{N} \sum_{u,y \in \mathcal{C}} \log P(y \mid u_1, \cdots, u_m). \tag{3.15}$$

In addition and similar to other work [66], the pre-trained language model objective $J_{LM}(D, \Theta)$ from Equation 3.10 is an auxiliary fine-tuning objective. In conclusion, $J(C, \Theta)$ jointly minimizes the negative log-likelihood of the pre-trained language model as well as the negative-log likelihood of the downstream task with

$$J(\mathcal{C}, \Theta) = J_{CLS}(\mathcal{C}, \Theta) + \gamma J_{LM}(\mathcal{C}, \Theta) \tag{3.16}$$

where γ is a task-specific scalar to scale the LM objective. Radford et al. [68] observe that the auxiliary loss and the scaling parameter accelerate the convergence during pre-training and increase the generalization during fine-tuning.

In summary, the approach (1) trains a Transformer decoder and (2) fine-tunes it for different tasks. Regarding these two stages abstractly, the approach is similar to CoVe [55] or ELMo [65]. However, instead of fine-tuning the approach for the extraction of word embeddings, the model replaces the entire encoding part of the downstream task. Further ablation studies for natural language inference point out that fine-tuning benefits up to 9% from *each* Transformer layer [68]. This study substantiates that transfer learning benefits from the entire model and not only the last layer.

3.3.3. Bidirectional Encoder Representations from Transformers (BERT)

The previously introduced OpenAI GPT uses a Transformer decoder with masked multi-head attention for language modeling. Compared to ELMo (see Section 3.2.2), the Transformer decoder is not able to capture the future context during pre-training. For this reason, Devlin et al. [19] propose the *Bidirectional Encoder Representations from Transformers (BERT)* to overcome the obstacle of unidirectionality. For better comparison, Figure 3.5 illustrates the distinctions between previous approaches and BERT.

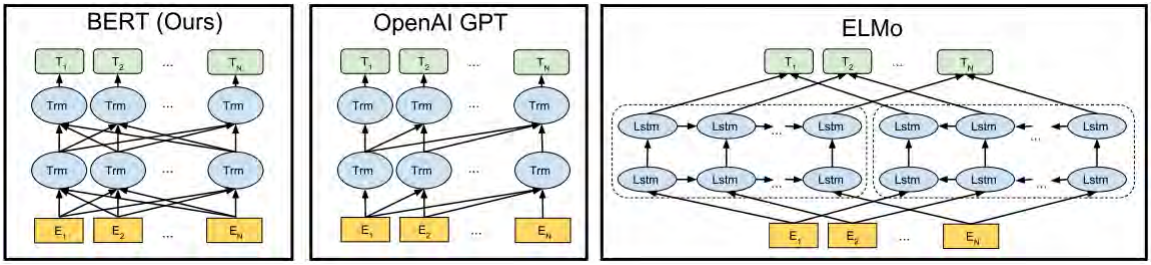


Figure 3.5.: Comparision of sequential transfer learning approaches [19].

ELMo, on the right side, *shallowly* concatenates a forward and backward LSTM to cover both contexts. In contrast to this, OpenAI GPT in the middle is based on a *deep* Transformer decoder architecture but neglects the future context. Finally, BERT on the right side learns both contexts with a Transformer in a *deeply-bidirectional* language model.

Cloze task Building a bidirectional language model on top of a Transformer entails challenges. First and foremost, the model has to ensure that an input token does not see itself in a multi-layered context. Otherwise, predictions become trivial since the target word is part of the input. Thus, Devlin et al. [19] propose a *Masked language model (masked LM)* that resembles the idea of a *cloze task* [82]. Instead of predicting the subsequent word given a sequence of preceding words, the masked LM masks certain words (targets) in the input sequence and predicts these with the remaining words. More specifically, 15% of the input tokens are dedicated as targets and replaced applying the following rules:

- 80%: Replace the target word with `[MASK]` token, e.g. for the target word `cat`:
  `the cat ate the mouse > the [MASK] ate the mouse`
- 10%: Replace the target with a *random* word
  `the cat ate the mouse > the cat ate chicken mouse`
- 10% Keep the sample unchanged.
  `the cat ate the mouse > the cat ate the mouse`

Next Sentence Prediction (NSP) The above language model has no knowledge of relations between two sequences in the input. However, NLP tasks like Question Answering (QA) or Natural Language Inference (NLI) benefit from the knowledge of the relatedness of sentences [19]. For this reason, an auxiliary binary classification task is introduced during pre-training. Given two sequences A and B, the objective is to approximate the probability whether a sentence B actually comes after a sentence A. BERT samples two sequences from the corpus with a maximum length of 512. 50% of the time B comes after A, 50% of the time it does not.

Besides the masking approach, the language model during pre-training of BERT is very similar to the one of OpenAI GPT. The model uses Transformer (self-attention) cells to learn the context and is trained to predict the missing (masked) words. Furthermore, the training objective of BERT is to jointly minimize the mean masked LM likelihood *and* the mean NSP likelihood.

Implementation BERT benefits from its tokenization and the collection of metadata. For instance, let A = ('I', 'like', 'cats') and B = ('My', 'dog', 'is', 'chasing', 'them') be two input sentences of sequence $U = \{A; B\}$. The input words are word-piece tokonized [85] as illustrated in the top row of Figure 3.6. As usual for Transformer-based models, the word embeddings ($E_{[CLS]}, \cdots, E_{[SEP]}$) are combined with positional encodings ($E_0, \cdots, E_s$). In addition to this, BERT stores metadata in the form of a binary *segment* which is 0 for tokens of A and 1 for tokens of B. Hence, BERT processes the combined embeddings and the metadata.

Figure 3.6.: Processing input sequences in BERT [84].

Furthermore, a unique token `[CLS]` prepends the sequence (u_0 = `[CLS]`) such that the first output state h_0 of the BERT model represents the entire sequence. The hidden states ($h_1, \cdots, h_s$) are pooled, fed to a linear layer and later used for text classification during fine-tuning. The intuition behind this can be illustrated using an exemplary binary spam classification model. The hidden states are pooled and fed to a linear layer and a softmax which classifies whether an input is spam. Instead of learning to pool during fine-tuning, `[CLS]` or respectively h_0 learns this behavior during pre-training. Additionally, the `[SEP]` token indicates the end of subsequences (A and B) for the NSP classification.

Fine-Tuning The application of the pre-trained BERT model to downstream tasks is straight forward. BERT does neither use multiple stages like ULM-FiT nor task-specific scaling parameters (see Section 3.3.1). The only adjustments are made to hyperparameters like the learning rate and the batch size because the fine-tuning infrastructure is typically much smaller then during pre-training. As described above, the first hidden state h_0 is fed to a task-specific linear layer for classification tasks. For other tasks like token-type classification tasks, each hidden state h_i at state i for token u_i is fed to a linear layer classifying the token. Last but not least, the hidden states can be directly treated as contextual word embeddings similar to ELMo in Section 3.2.2. These contextual BERT embeddings are used in the experiments in Chapter 6.

In summary, BERT introduces a novel masked language model which extends the OpenAI GPT by a bidirectional context. Furthermore, *practical* complements like the next sentence prediction and the classification token learn patterns for fine-tuning tasks. As studies show, the most significant impact of BERT builds on the masked language model [19]. Another important impact is based on the complexity of the pre-trained models. The first model, $BERT_{BASE}$, has L = 12 Transformer layers and h = 12 attention heads which is similar to the OpenAI GPT [68]. These models have around 110 million parameters [19]. The second model, $BERT_{LARGE}$, consists of L = 24 each of which has h = 12 attention heads with

around 310 million parameters[2]. The computational resources[3] to pre-train these language models is bypassed which implies great chances for sequential transfer learning.

3.4. Conclusion

Transfer learning in NLP can either be transductive or inductive. The focus of this thesis is on inductive approaches of sequential transfer learning. In the first stage, a large model is pre-trained ahead-of-time for a source task. The second step fine-tunes the model for a dedicated downstream task. Contextual vectors like (CoVe) [55] and ELMo [65] improve the obstacle of traditional word embeddings (see Section 2.3) that the representations are identical in any context. Even though contextual word embeddings show promising results [65], the approaches still limit themselves to the embedding layer.

In comparison to computer vision, NLP models are shallower and require different transfer learning techniques [37]. Approaches like OpenAI GPT [68] and BERT [19] build on Transformer cells and pre-train large-scale language models. The models are fine-tuned for downstream tasks and show promising advances towards sequential transfer learning in NLP. Nevertheless, the downstream tasks are mainly text classifiers which replace the last layer of the language model with a classifier. Neither OpenAI GPT nor BERT address a sequence-to-sequence task like text summarization in their approaches [68, 19].

Further work is available in the field of *multi-task learning* (see Section 3.1). McCann et al. [56] introduce the *decaNLP*[4], a set of ten popular NLP tasks with the respective datasets to evaluate the abilities of a trained model for multiple tasks. Furthermore, Radford et al. [69] propose the *OpenAI GPT-2* which shows that language models adapt to downstream task without any adjustments (*zero-shot learning*).

Another fact is that the achieved performance relies on extraordinary huge models. The OpenAI GPT model has around 110 million parameters [68] and the large version of BERT around 310 million parameters [19]. In comparison to this, the most extensive model of OpenAI GPT-2 is a Transformer [69] with 1.5 billion parameters which has over a magnitude more parameters than BERT.

The practical part of this thesis in Chapter 6 examines the influence of sequential transfer learning with approaches like ELMo, OpenAI GPT and BERT for the task of text summarization.

[2] https://github.com/google-research/bert
[3] $\text{BERT}_{\text{LARGE}}$ was trained on 16 Cloud TPUs (64 TPUS chips) for 4 days.
[4] https://decanlp.com/

4. Related work in neural text summarization

This chapter introduces the task of generating text summaries in the context of Deep Learning and gives an overview of related work in this field. This establishes the baseline for the upcoming experiments in Chapter 5. The first section demarcates the crucial aspects of text summarization for this thesis and declares a terminology for the remaining thesis. The subsequent section introduces recent Deep Learning approaches for text summarization and discusses their concepts and tasks. Finally, the last section explains common metrics to evaluate generated summaries.

4.1. Demarcation and terminology

The field of automatic summarization has a long history [62] and implies many different fields and problems. As a reminder, the objective of this thesis is to utilize recent advances in sequential transfer learning from Chapter 3 on neural text summarization. For this reason, the following section is not intended to be a thorough introduction about the process of generating summaries. Hence, neural approaches are only fundamentally compared to human behaviors, and many linguistic concepts are out of scope for this thesis. The following section introduces common terms and narrows the relevant topics of this thesis.

Extractive and abstractive summaries As a reminder, there are two established approaches to summarise a text. *Extractive summaries* are a concatenation of several words of the input document, whereas *abstractive summaries* aim to convey relevant information of the input and to create a fluent and concise summary [75]. Regarding humans' behavior, the abstractive approach seems to be more intuitive. Given an input text, humans would probably mark important passages (extract information), but instead of just concatenating them, humans would use their literacy to create fluent summaries. Consequentially, this thesis focuses on approaches of *abstractive summarization*. Nevertheless, the distinction between extractive and abstractive is an important aspect for the evaluation in Chapter 6.

Promises in neural approaches The process of generating summaries requires two capabilities. First, the extraction of important parts requires an understanding of language and semantics. Second, the creation of concise and fluent summaries build on capabilities to reorganize, modify and merge information differently [62]. Concerning these requirements, recent approaches based on *Deep Neural Networks* show promising results [61, 79]. Further approaches [42, 20] have been proposed but are not in the scope of this thesis.

Objective As for now, the vast majority of existing approaches generates a single summary of a single document (see Table 4.1). Thus, this thesis focuses on this particularly problem. Regarding the terminology, the input is named *document* respectively $\mathcal{D}$, and the generated summary by a neural network is called *candidate summary* respectively $\mathcal{S}$. As explained in

more detail in Section 6.1, approaches commonly evaluate the generated summaries against a human-written summary. This gold target is called the *reference summary* or $\mathcal{R}$.

4.2. Related work

For a first impression of recent work in the field of abstractive summarization, Table 4.1 illustrates seven current approaches. In detail, the table explains their underlying concepts, the applied evaluation metrics and whether the approach uses reinforcement learning (**RL**), global attention (**A**) or a pointer generator (**PG**).

	Dataset	**Model**	**Metrics**	**RL**	**A**	**PG**
Kryściński et al. [44]	CNN/DM	Encoder: BiLSTM Decoder: LSTM + **auxiliary language and contextual model**	ROUGE Novel n-grams	✓	✓	✓
Liu et al. [50]	**WikiSum**	**Transformer** Two-stage extractive-abstractive framework	ROUGE Test-perplexity	✗	✓	✗
Liu et al. [49]	CNN/DM	**GAN** **G**: like [79] **D**: CNN text classifier, max-over-time-pooling, softmax output)	ROUGE	✓	✓	✓
See, Liu, and Manning [79]	CNN/DM	Encoder: BiLSTM Decoder: Attention with **Coverage vector**	ROUGE **METEOR**	✗	✓	✓
Paulus, Xiong, and Socher [63]	CNN/DM New York Times	Encoder: BiLSTM Decoder: Single LSTM **Policy gradient learning**	ROUGE	✓	✓	✓
Nallapati et al. [61]	**CNN/DM** Gigaword DUC	Encoder: Bi-GRU Decoder: Uni-directional GRU	ROUGE	✗	✓	✗
Gu et al. [32]	LCSTS [38] DUC	Encoder: Bi-GRU Decoder: Uni-directional GRU **Copy mechansim**	ROUGE	✗	✓	✓
Rush, Chopra, and Weston [75]	**Gigaword** DUC	Multiple encoders tested (Bag-of-words, Convolutional, **Attention-Based**)	ROUGE	✗	✓	✗

Table 4.1.: Comparison of recent approaches addressing abstractive summarization sorted by the most recent proposals ascending. Highlights and novelities of papers are marked in **bold**. This table is not intended to be exhaustive but compares novel and relevant papers for this thesis.

Due to the importance for this thesis, the datasets or respectively tasks are explained separately in Section 4.2.2. Regarding the metrics, every approach evaluates the results

using ROUGE, and several approaches define custom scores. Section 4.3 describes and compares these more thoroughly.

4.2.1. Model and task-specific components

Many approaches in Table 4.1 use a recurrent encoder-decoder model. While earlier approaches mainly use a *Gated recurrent unit (GRU)* [13] as recurrent cells, later approaches use BiLSTM cells for the encoder and unidirectional LSTM cells for the decoder (see Section 2.4.2). However, an interesting approach by Liu et al. [50] is based on a Transformer decoder with pure self-attention instead of recurrent cells. This follows a similar architecture to the OpenAI Transformer (see Section 3.3.2) but for text summarization. Another approach by Liu et al. [49] utilises a *Generative Adversarial Network (GAN)* [27] with an encoder-decoder based generator and a CNN as discriminator.

Pointer-generator and coverage In addition to the architecture as described beforehand, the approaches introduce several task-specific components. An additional *pointer-generator network* [32, 79] copies tokens from the source document to the summary via pointing. This faces the problem that summarization systems tend to have many *Out-of-Vocabulary (OOV)* words during inference [79]. Another obstacle of these summarization systems is the repetition in summaries. A *coverage vector* [83] addreses this issue by tracking and controlling the covered and uncovered parts of the source document [79].

Reinforcement Learning (RL) Another concept is the addition of an auxiliary learning objective which dedicatedly measures the quality of generated summaries. For text summarization, however, the common ROUGE metric is content-based and therefore not differentiable [63]. Non-differentiability is a typical scenario for reinforcement learning techniques. Thus, various approaches from Table 4.1 [63, 49, 44] derive *policy-gradient learning* [71] from the task of image captioning. Paulus, Xiong, and Socher [63] define this as learning *"a policy that maximizes a specific discrete metric instead of minimizing the maximum-likelihood loss"*.

The practical part of this thesis is based on an implementation of the CopyNet model [32] which uses a copy mechanism similar to a pointer-generator (see Chapter 5). However, policy gradient learning and coverage are not investigated further in the remainder of this thesis. This is due to the fact that their improvements are not essential for the third research objective (see Section 1.2) and their addition would interfere the experiments in Chapter 6.

4.2.2. Tasks and datasets

The tasks in neural abstractive summarization can be distinguished by their objective as well as the given documents and summaries. Therefore, Figure 4.1 illustrates the three categories of summarization objectives and common tasks in recent approaches. While *sentence-level* summarization deals with documents of a few sentences and summaries of a couple of words, *document-level* contains documents and multiple sentence summaries. The third category is *multi-document* summarization and aims to summarize multiple documents to a single summary. The following sections explain the datasets in more detail.

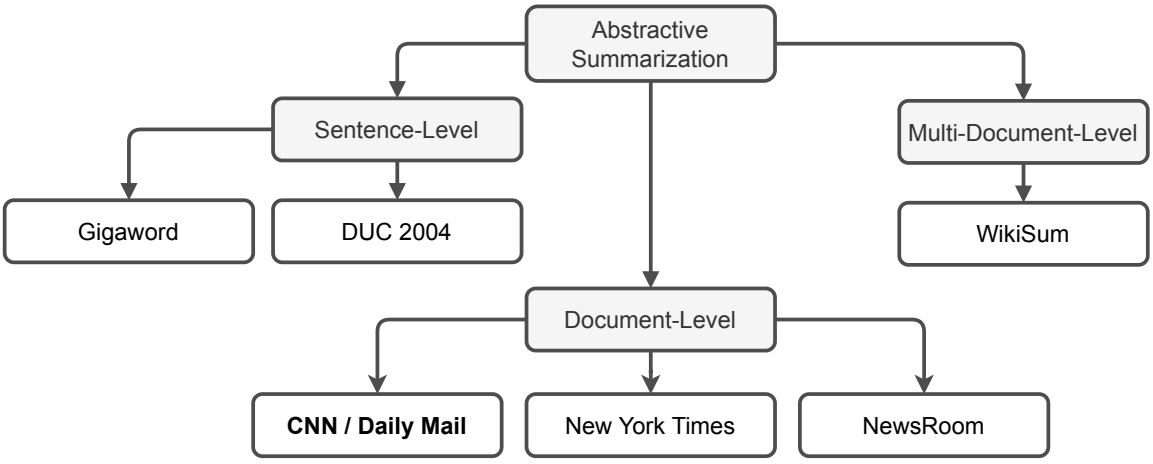


Figure 4.1.: Taxonomy of tasks and datasets in abstractive text summarization.

Sentence-level (Gigaword) Rush, Chopra, and Weston [75] propose the *Gigaword* dataset to address a task with documents of a few sentences and summaries of multiple words. This is also called a headline generation task. The dataset is very extensive and consists of 3,800,000 training pairs. Input documents contain around 30 words and reference headlines around 8 words on average [75]. Gigaword has been investigated heavily in neural summarization [75, 61, 47] but simplifies the objective of generating summaries because neural models have to track a relatively small context.

Document-level (CNN / Daily Mail) Hermann et al. [35] introduce a dataset consisting of online news articles extracted from CNN and DailyMail websites. The dataset contains around 300,000 training pairs, and each instance has a document with 781 tokens and a summary of 56 tokens, on average. Some approaches in Table 4.1 [50, 63, 61] are evaluated on an anonymized version by Nallapati et al. [61] which replaces named entities. Others [79, 44], however, use a non-anonymized version [79]. The different versions come with the cost of worse comparability as Table 4.2 shows.

Multi-Document level (WikiSum) Liu et al. [50] propose the *WikiSum* dataset for multi-document summarization. The dataset consists of English Wikipedia[1] articles as summaries and their citations as the source document. The citations are enriched with Google search results [50] and trained to cover the content of the Wikipedia article, i. e., the reference summary. The WikiSum dataset is promising for future approaches since current models predominantly support document-level only. Furthermore, the extent of the task requires skills with two steps of extracting and summarizing (see Section 4.1).

Further datasets Besides the datasets above, Table 4.1 references further datasets. First, *DUC2004*[2] is a high-quality sentence-level dataset created by the Document Understanding Conference (DUC). The small dataset contains 500 documents and is mainly used to test models, which were trained on larger datasets. Following a similar idea to CNN/DailyMail,

[1] https://www.wikipedia.org/
[2] https://duc.nist.gov/duc2004/

New York Times Corpus is another large text corpus with news articles [76]. Last but not least, a promising dataset for future research is *NewsRoom* by Grusky, Naaman, and Artzi [31] which contains over one million summaries extracted from over a hundred million web pages. The dataset promises a higher diversity than CNN/DailyMail [31].

Addressing the objectives in this thesis, the practical part in Chapter 5 implements a baseline model to perform document-level summarization with the CNN/DailyMail dataset.

4.3. Evaluation of summaries

The previous sections have addressed recent approaches in neural summarization while skipping evaluation techniques and metrics. In general, the results of neural summarization systems are evaluated with a human-written (reference) summary by comparing their content. The following sections introduce two metrics for a content-based evaluation, which are essential for the understanding of the evaluation in the practical part (see Section 6.1).

4.3.1. Content-based metrics (ROUGE)

A common metric for text summarization is the *Recall-Oriented Understudy for Gisting Evaluation (ROUGE)* [46] which expresses how well the content of a generated summary covers an ideal human-written summary[3]. The ROUGE-n score is based on n-grams (see Section 2.1.2) and measures the similarity of the generated and reference summaries. Furthermore, ROUGE-L scores the longest-common-subsequence (LCS) [46].

Let $f_{ng}(x, n)$ be a function that computes a set of n-grams occurring in text x. Furthermore, let $||\bullet||$ be the count of words in a set. Hence, the ROUGE-$\mathrm{n}_{(P)}$ score can be defined as

$$\text{ROUGE-}n_{(P)}(s,t) = \frac{||f_{ng}(s,n)|| - ||f_{ng}(s,n) \setminus f_{ng}(t,n)||}{||f_{ng}(s,n)||}. \tag{4.1}$$

where the subscript (P) denotes the *precision* of the ROUGE-n. The precision counts the matching n-grams in the candidate summary s and reference summary t and normalizes them by the total number of n-grams in the candidate summary s. Consequentially, a recall of ROUGE-n can be defined as

$$\text{ROUGE-n}_{(R)}(s,t) = \frac{||f_{ng}(s,n)|| - ||f_{ng}(s,n) \setminus f_{ng}(t,n)||}{||f_{ng}(\mathbf{t},n)||} \tag{4.2}$$

that normalizes the matching n-grams by the total number of n-grams in the reference summary t. The third n-grams based ROUGE score is ROUGE-$n_{(F)}$ which is the harmonic mean between precision and recall and defined as

$$\text{ROUGE-n}_{(F)} = 2 * \frac{\text{ROUGE-n}_{(P)} * \text{ROUGE-n}_{(R)}}{\text{ROUGE-n}_{(P)} + \text{ROUGE-n}_{(R)}}. \tag{4.3}$$

The ROUGE in text summarization is usually evaluated for $n = 1$ (unigrams) and $n = 2$ (bigrams). Further, the Rouge-L score measures the longest common subsequence between the candidate and the target summary. As shown in Table 4.1, many recent approaches

[3] In the context of thesis, the ROUGE score is defined for the particular task of text summarization under consideration of a single generated summary and a single reference summary

perform document-level summarization on the CNN / DailyMail dataset. Since the practical part of this thesis further uses this particular dataset, Table 4.2 shows the ROUGE evaluation results of recent approaches.

Reference	ROUGE-1	ROUGE-2	ROUGE-L	Entity-anony.
Kryściński et al. [44]	40.19	17.38	37.52	✗
Liu et al. [49]	39.92	17.65	36.71	✓
See, Liu, and Manning [79]	39.53	17.28	36.38	✗
Paulus, Xiong, and Socher [63]	39.87	15.82	36.90	✓
Nallapati et al. [61]	35.46	13.30	32.65	✓

Table 4.2.: Rouge results for CNN / DailyMail of recent approaches from Table 4.1.

The approaches in Table 4.2 are ordered by their publication date. As a reminder to Table 4.1, Nallapati et al. [61] use a pure encoder-decoder model with GRUs. Paulus, Xiong, and Socher [63] use reinforcement learning and a pointer generator network. See, Liu, and Manning [79] add the coverage mechanism. For this reason, the approaches outperform the baseline model on CNN / DailyMail by Nallapati et al. [61] with 4-5 ROUGE points. The last column in Table 4.2 indicates whether the approaches use the entity-anonymized (✓) or the non-anonymized (✗) version. Thus, the approaches are strictly spoken not comparable to each other. The practical part of this thesis uses the non-anonymized version of the dataset and baseline results are only compared with these approaches (see Section 5.1).

4.3.2. Measuring the abstractive ability

One primary objective of abstractive summarization is the generation of new words instead of merely copying the entire source (see Section 4.1). However, the ROUGE score measures the lexical similarity between reference and candidate summaries, which does not indicate the level of abstraction. Hence, a generated summary may contain very different words than the target summary with low ROUGE scores but still expresses a similar meaning.

For this reason, Nallapati et al. [61] report the percentage of matching words that occur in both, the candidate summary and the input document. In a slightly different notation to Equation 4.1, let $f_{ng}(x, 1)$ be a function that creates a set of words (1-grams) in text x, $||\bullet||$ be the number of words in *set*, s be the candidate summary, and d be the source document. Hence, the *copy rate* is defined as

$$\text{NOVEL-1}(s, d) = \frac{||f_{ng}(s, 1) \setminus f_{ng}(d, 1)||}{||f_{ng}(s, 1)||}. \tag{4.4}$$

For a first impression, Nallapati et al. [61] report the copy rate of their approach with 78.70%. This means that only 21.30% of the words in a summary are distinct from tokens that appear in the vocabulary generated from the source document.

Novel n-gram The copy rate or NOVEL-1 score compares the unigrams in the candidate summary and the source document. Kryściński et al. [44] extend this idea and define a *novel n-gram* metric to measure not only the unigrams but also overlaps of multiple words. Generalizing the notation from Equation 4.4, the novelty score NOVEL-n is defined as

$$\text{NOVEL-}n(s,d) = \frac{||f_{ng}(s,n) \setminus f_{ng}(d,n)||}{||f_{ng}(s,n)||} \tag{4.5}$$

Another approach by Liu et al. [50] evaluates the summarization model with the score of an additional language model and its perplexity (see Section 2.2). Furthermore, the approach adds the log-perplexity as auxiliary learning objective during training which tends to improve the generated summaries [50]. Another approach by Paulus, Xiong, and Socher [63] follows a similar idea.

4.4. Conclusion

This thesis focuses on abstractive text summarization which has recently shown promising developments with approaches of deep neural networks. Commonly based on encoder-decoder models with attention [75], related work includes components like coverage mechanisms [79], pointer generators [32, 63], and policy learning [71, 63]. Furthermore, the research in abstractive text summarization mainly addresses the generation of headlines (sentence-level) or short summaries (document-level). Further, the content-based ROUGE score [46] evaluates the generated summaries by comparing their content to a given reference summary. Consequently, models with better ROUGE scores have a higher lexical similarity to a single reference summary. However, this leads to a trade-off between expressing the meaning in different words and lexical overlaps to the target summary [44].

5. Approach and implementation

This chapter unveils the practical work of this thesis. In a nutshell, the objective of the practical part is to conduct experiments that show transfer learning abilities for the task of text summarization. This combines the previous two theoretical chapters and integrates several approaches such as self-attention (see Section 2.5.2) or contextual embeddings (see Section 3.2) in a state-of-the-art summarization model (see Section 4.2). However, before actually conducting the experiments in Chapter 6, this chapter reveals the general procedure of implementing and evaluating summarization models.

The first section introduces the CopyNet summarization model which is the baseline for the remainder of this thesis. The subsequent section deals with the research code framework AllenNLP which provides the core components of the practical part. Thus, Section 5.2 explains AllenNLP and the concrete implementation of the baseline model. The final section accounts for the experiment setup of creating and evaluating different runs in Chapter 6.

5.1. CopyNet model as a baseline

The baseline is an implementation of the *CopyNet* model by Gu et al. [32]. This approach has been mentioned in Chapter 4 and refers to the sequence-to-sequence approaches of Table 4.1. Similar to the pointer generator (see Section 4.2.1) in other approaches, a *copy mechanism* locates essential segments of the input sequence and puts these to the proper places in the candidate summary. These segments can either be entire sub-sequences or entities like names or organizations. The copy mechanism adds an *extractive* component and is particularly useful to reduce the number of words that have not seen during training (*OOV words*).

5.1.1. Overview

The CopyNet model relies on an encoder-decoder architecture (see Section 2.4 and Figure 2.4). Given s words $(u_1, \cdots, u_s)$ in an input document, the words are embedded as $(x_1, \cdots, x_s)$ in the first layer. Subsequently, an encoder processes each word u_i to an hidden state h_i. In the baseline approach, this hidden state is composed of a forward and a backward hidden state $h_i = [\overrightarrow{h}_i; \overleftarrow{h}_i]$ that encode the word with recurrent LSTM cells (see Section 2.4.2). The set of all encoder hidden states is referred to as the memory $\mathcal{M} = \{h_1, \cdots, h_s\}$.

Under consideration of the memory $\mathcal{M}$, the decoder generates the summary word-by-word by reading $\mathcal{M}$ in two distinct manners:

1. **Attentive read** - CopyNet is an attention-based encoder-decoder model that approximates the probabilities of an output y_i (see Section 2.5). Thus, the probability $P_{gen}(\bullet)$ of generating an output y_i can be expressed as the conditional probability of y_i given

the hidden state h_i, the last decoded output y_{i-1}, the context vector c_i, and the memory $\mathcal{M}$, i. e.

$$P_{gen}(y_i, \mathbf{g} \mid h_i, y_{i-1}, c_i, \mathcal{M}). \tag{5.1}$$

where $\mathbf{g}$ refers to the generation mode. This probability $P_{gen}(\bullet)$ is denoted as the *generation score* since the model performs an attentive read via global attention to access $\mathcal{M}$.

2. **Selective read** - In addition, the decoder reads the memory $\mathcal{M}$ to compute a probability of copying a word y_i from the source document. The probability $P_{copy}(\bullet)$ is referred to as *copy score* and defined similarly to P_{gen} as

$$P_{copy}(y_i, \mathbf{c} \mid h_i, y_{i-1}, c_i, \mathcal{M}). \tag{5.2}$$

where $\mathbf{c}$ refers to the copy mode.

In summary, the generation score defines the probability of generating a new token, and the copy score defines the probability of taking a token from the source document. The overall probability can be defined as

$$P(y_i \mid h_i, y_{i-1}, c_i, \mathcal{M}) = P_{gen}(y_i, \mathbf{g} \mid \bullet) + P_{copy}(y_i, \mathbf{c} \mid \bullet) \tag{5.3}$$

Let $\mathcal{V} = \{v_1, \cdots, v_N\}$ be a predefined vocabulary for the CopyNet model. In the approach of this thesis, the vocabulary consists of the $|\mathcal{V}|$ most frequent words extracted from the training, validation and testing dataset. However, the input sequence $(u_1, \cdots, u_s)$ of an instance during training or inference may contain words like entity names that are important for the summary but not part of $\mathcal{V}$. Furthermore, the CopyNet model requires these words in order to approximate the copy score. For this reason, a second *dynamic dictionary* $\mathcal{W} = \{w_1, \cdots, w_T\}$ consists of the unique words from the input sequence $(u_1, \cdots, u_s)$. Besides this, the `UNK` token is a placeholder for out-of-vocabulary words. Altogether, the model approximates probabilities for each word $y_i \in \mathcal{V} \cup \texttt{UNK} \cup \mathcal{W}$ under consideration of an individual dictionary $\mathcal{W}$ for each input sequence $(u_1, \cdots, u_s)$.

More specifically, the CopyNet applies the following heuristic to approximate the copy score $P_{copy}(\bullet)$ and generation score $P_{gen}(\bullet)$:

$$P_{gen}(y_i, \mathbf{g} \mid s_i, y_{i-1}, c_i, \mathcal{M}) = \begin{cases} \sigma(\exp(f_{attn}(y_i))), & y_i \in \mathcal{V} \\ \sigma(\exp(f_{attn}(\texttt{UNK}))), & y_i \notin \mathcal{V} \cup \mathcal{W} \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

$$P_{copy}(y_i, \mathbf{c} \mid s_i, y_{i-1}, c_i, \mathcal{M}) = \begin{cases} \sigma(\sum_{j:x_j=y_i} \exp(f_{copy}(x_j))), & y_i \in \mathcal{W} \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

where f_{attn} is the additive attention by Bahdanau, Cho, and Bengio [1] (see Section 2.5) [32]. Furthermore, $f_{copy}(x_j)$ is the scoring function for copying the j-th input word $x_j \in \mathcal{W}$ with the hidden state $\bar{h}_j$ for output y_i which is defined as

$$f_{copy}(x_j) = \tanh(\bar{h}_j^\top W_c) h_i \tag{5.6}$$

where W_c is a learned parameter.

Additionally, σ normalizes the terms with both, copy and generation score, using the shared softmax such that

$$\sigma = \frac{1}{\sum_{v \in \mathcal{V} \cup \{\texttt{UNK}\}} \exp(f_{attn}(v)) + \sum_{x \in \mathcal{W}} \exp(f_{copy}(x))}. \tag{5.7}$$

5.1.2. Further features

The CopyNet distinguishes to traditional encoder-decoder models in the input during decoding. At the i-th timestamp, the decoder typically uses the embedding of the last decoded word (y_{i-1}) as input (see Section 2.4.1). In CopyNet, however, the input is referred to as *state* and contains additional information of the source memory $\mathcal{M}$. Given the decoder copied the last word from the source document, another attention mechanism decides which hidden states of the encoder are particularly useful for copying the next word. This is similar to the attentive read as introduced above and dedicated to improving the copy mechanism. In case the last hidden word is not part of the source document, the attention is not computed [32].

In summary, CopyNet is based on an encoder-decoder model with global attention. Additionally, the decoder learns to copy words or entire sequences from the source document. At each decoding step, the model computes probabilities for copying the word from source and generating a word from the vocabulary. These probabilities are jointly optimized with backpropagation during training by minimizing the negative log-likelihood [32]. In this thesis, the model is implemented with the framework AllenNLP [23] as introduced in the following section. For a better understanding, Section 5.2.3 provides a concrete implementation of the CopyNet model based on AllenNLP and Figure 5.3 illustrates the particular architecture.

5.2. AllenNLP: A Natural Language Processing Platform

Practical approaches in the field of Deep Learning and NLP often target a research objective. The goal of this thesis is to show the abilities of transfer learning for text summarization. Research implementations necessitate *prototyping* with fast development times and immediate feedback instead of well-tested and stable productive code. Nonetheless, the complexity of Deep Learning comes along with the data science pipeline of data preprocessing, training, optimization of the model and evaluation of results. For this reason, even simple prototypes for the task of text summarization depend on many components to produce stable results. For instance, the CopyNet model requires encoding, decoding, attending and copying steps.

Consequently, the experiments in Chapter 6 do not implement these components from scratch but are based on a research framework named AllenNLP[1][23]. It consists of re-usable components and pipelines to train, maintain and serve models. AllenNLP itself is based on pytorch[2], an open source deep learning platform for research prototyping. Before introducing the architecture in the subsequent section, motivating features for AllenNLP are:

[1] https://allennlp.org/
[2] https://pytorch.org/

- **Training loop** - AllenNLP defines a training loop with a pipeline of loading data, batching, iterating, training and evaluating the results. Mature core components essential to conduct research-oriented experiments.
- **Logging and Monitoring** - AllenNLP stores checkpoints of the model and handles logging. A Tensorboard[3] visualizes the training history in real time. During prototyping, this is particularly useful to get immediate feedback about the performance of a model.
- **Loosely-coupled and Reusability** - AllenNLP enables a fast development process of research code with hacks during prototyping and consists of reusable components for different tasks.
- **SOTA models** - AllenNLP provides the implementation for various tasks which helps to produce less error-prone code and reproduce the performance.

5.2.1. Architecture overview

Figure 5.1 illustrates the core architecture of the AllenNLP framework with components involved in training. For inference, Section 5.3 discusses a separate prediction component.

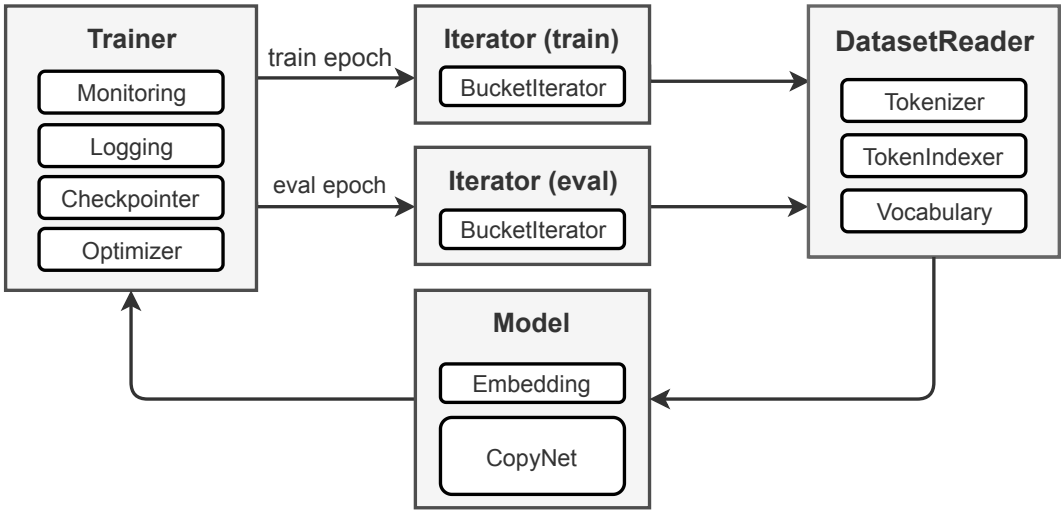


Figure 5.1.: Architecture of the components in AllenNLP with the CopyNet model.

The *Trainer* is a rich component which is responsible for the actual training procedure. In the beginning, the trainer sets up the training parameters and starts looping over N epochs. An *Iterator*, actually one for training and another one for validation, loads data in memory and handles smart batching. The *DatasetReader* is another rich component reading from disk and pre-processing the raw text to an instance. Subsequently, the *Model* receives the batches with instances, generates output probabilities and computes the loss during training. In the final step, the trainer triggers the optimization, logs the epoch results and backs up the model's checkpoint. In the next round, the trainer evaluates the model on the validation dataset, followed by another training run. This training loop repeats this process for N

[3] https://github.com/tensorflow/tensorboard

epochs. Due to their importance of this thesis, the DatasetReader and Model component are explained in more details below.

5.2.2. Dataset Reader for text summarization

For text summarization in this thesis, a dataset reader retrieves (source) documents and reference (target) summaries from disk and creates raw input and output sequences. These input pairs are processed by exchangeable components as illustrated in Figure 5.2.

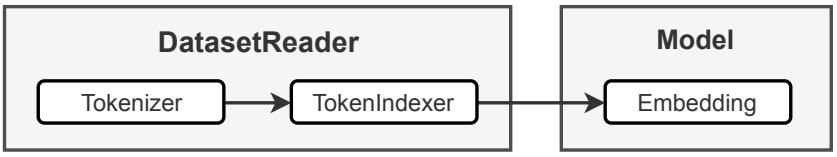


Figure 5.2.: Components and interfaces of data readers in AllenNLP.

First, a *Tokenizer* splits the raw text into lists of single tokens, which may be single words, word-pieces, characters or any other desired behavior. The *TokenIndexer* creates a lookup table of tokens by indexing the tokens in the vocabulary. The third data-specific component, which is part of the model in AllenNLP, is the *Embedding*. This component receives the token ids and computes the word embeddings for the model.

This loosely-coupled architecture is the baseline for different experiments in Chapter 6. The benefits are more understandable using the example of integrating contextual embeddings like ELMo in AllenNLP (see Section 3.2.2). The goal is to feed input sequences to an auxiliary ELMo model computing contextual embeddings. In this scenario, the tokenizer splits the input sequences based on whitespaces and punctuation. The token indexer creates the numerical representations of these tokens. The ELMo-specific embedder internally uses the pre-trained ELMo-model that generates the ELMo-embeddings. These are concatenated with the word embeddings and passed to the encoder. Other approaches like BERT or OpenAI-GPT replace the ELMo embeddings but keep the rest of the model unchanged.

5.2.3. CopyNet model

The model component in AllenNLP performs the actual computations and is also exchangeable due to the requirements of the task. For this thesis and as previously described, the model is an implementation CopyNet as introduced in theory in Section 5.1. Figure 5.3 illustrates the respective model.

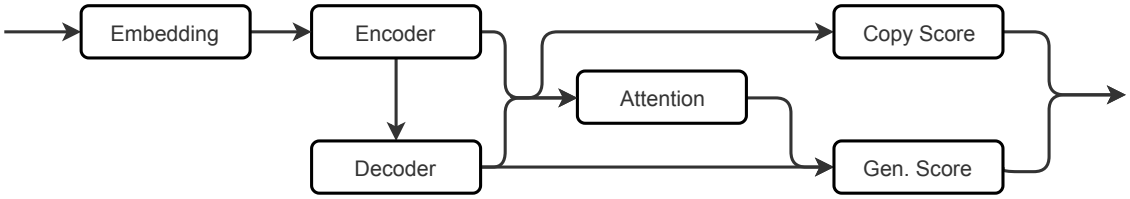


Figure 5.3.: Implementation of the CopyNet model based on AllenNLP.

As described in Section 5.1, the *encoder* receives input embeddings and computes a fixed-length representation using a BiLSTM. The *decoder* generates two output probabilities for copying a source token to the target or generating a new token from the vocabulary. For the generation of new tokens, an additional global *attention* component computes the generation score (*gen. score*) based on the output of the encoder and decoder. For copying tokens from the source, a *copy score* weights the output of the encoder and decoder. Finally, the generation score and the copy score are combined and represent the final output probabilities.

However, the model does not predict a single token with the best score (*greedy decoding*) but implements a *beam search* [29] algorithm. The model generates outputs word by word and keeps track of k hypotheses of active candidates at each timestep. In the following step, the model computes the next word for all hypotheses and saves the k new hypotheses with the highest sequence probability [22]. However, increasing the beam size can significantly reduce the decoder's speed. Thus, the number of hypotheses is $k = 4$ for all experiments in this thesis, and the model creates 4 candidate summaries for each input document. However, if not stated differently, the evaluation refers to the most likely candidate summary.

Another task of models in AllenNLP is the computation of the training loss during training. For the CopyNet model above, the negative log-likelihood is computed and reported to the trainer, which triggers the optimization of parameters. Similar to other components of AllenNLP, the CopyNet model is highly configurable. The experiments in this thesis are limited to varying two components. First, Section 6.3 compares recurrent LSTM encoder cells to a self-attention encoder. Secondly, the sections 6.4 and 6.5 use different pre-trained and contextual embeddings.

5.3. Experimental setup

In this thesis, properties like the dataset and task, the encoder type, additional pre-trained embeddings, the type of model and the evaluation metrics define the setup of experiments. This section introduces the properties and the workflow to conduct experiments with the AllenNLP framework.

Workflow overview The primary objective of the workflow is to provide the same environment for all experiments. The same conditions are an essential baseline for evaluation of different approaches in Chapter 6. In general, the workflow involves three stages:

1. **Creating the vocabulary** - The vocabulary is created in a separate step ahead of time and used for experiments with the same dataset. This removes one complexity from the training run and accelerates the training for large datasets.

2. **Training** - The second stage is the training and optimization of the model of an experiment run. This can take multiple hours or days for each experiment.

3. **Testing & Evaluation** - In the last stage, the predictor serves the best model from training and evaluates with a dedicated test dataset. Each experiment uses the same test dataset.

After conducting several experiments with different properties, the evaluation metrics and sample predictions are compared and discussed. Essentially, this is the content of Chapter

6. Before that, however, the following sections take a more in-depth look in the training and testing stage. Additionally, Figure 5.4 illustrates the two stages and the associated components.

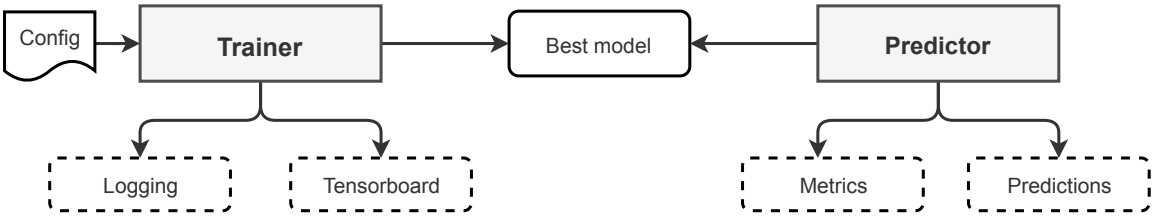


Figure 5.4.: Workflow of experiments during training and testing.

Training The left side of Figure 5.4 shows the training. The *Trainer* loads a *configuration* with the properties of an experiment. The configuration is a declarative file in JSON format and sets up components such as the data reader, the model and the trainer itself. In a more technical description, the configuration defines the initialization of the components of AllenNLP. For a better understanding, Listing A.1 illustrates the configuration of the baseline summarization task on the CNN/DailyMail dataset as referenced in Section 6.2.

Besides the technical details, the declarative configuration comes with a significant benefit: the separation of experiment definition and implementation. Regarding the prototyping research programming, the process starts with many constraints in the source code for a first running baseline. Subsequently, the constraints move to configurable parameters which enable reproducible and comparable experiment runs.

Furthermore, the trainer has abundant features to track metrics during training and evaluation. In periodic intervals, the component tracks metrics such as the training and validation loss or the ROUGE scores (see Section 6.1). For visualization, a Tensorboard illustrates the course of values and shows the learning pattern of models with different configurations. Finally, the best-performing model is selected by the lowest negative validation log likelihood and stored on disk for testing and evaluation.

Testing and evaluation After training a model over hours or days, the best model is served and evaluated with a test dataset. This test dataset is identical for each experiment type. Similar to the training process, the testing component reports a *test loss* and the ROUGE scores for each document-summary-pair (see Section 6.1). Even though these metrics give a quantitative measure of the generated summaries, the quality of summaries is still very subjective (see Section 4.1 and Section 6.6). Thus, the testing component stores the best predictions of testing pairs for each experiment.

6. Experiments and discussion

This chapter provides experiments analyzing the transfer learning abilities of pre-trained models for the task of text summarization. This includes approaches such as self-attention (see Section 2.5.2), pre-trained word embeddings (see Section 2.3.2) and contextual embeddings (see Section 3.2) in the summarization model and workflow of the previous chapter (see Chapter 5). Before conducting the actual experiments, the first section introduces the metrics and features for the evaluation. Thereupon, a baseline analysis compares the summarization model of this thesis to recent approaches. The remainder of this chapter utilizes three subsets of the CNN / DailyMail dataset to conduct various experiments. The final section discusses the results and future work.

6.1. Features for comparison

This section explains how the experiments are evaluated and motivates for particular features. Content-based metrics such as ROUGE have been extensively explained in Section 4.3. Thus, their theoretical background is not covered in-depth in this section. However, less common and problem-specific evaluation metrics are explained in more detail.

Content-based (ROUGE) The primary metric to measure the performance of a generated summary is the evaluation of its content with a reference summary. The respective metric for text summarization is the ROUGE score [46]. In this thesis, the experiments are evaluated on *unigrams* (ROUGE-1), *bigrams* (ROUGE-2) and the longest common subsequence (ROUGE-L). Each of these reports the precision (ROUGE-n P), recall (ROUGE-n R) and F1 score (ROUGE-n F). While the precision suggests how well the generated summary is reflected by the reference summary, the recall expresses how well the content of the generated summary covers the reference. On top of this, the F1 Score (ROUGE-n F) measures the balance between Precision and Recall. Evaluation results that report the ROUGE-n score without a suffix (F, P or R) refer to the F1 score.

Average length (Length) The length of a summary denotes the total number of words in a generated summary. In the following sections, the average length of all generated summaries is referred to as *average length* (or short: *Length*). Even though the length is a fairly trivial measure, this metric can show learning patterns in combination with content-based metrics like ROUGE. For instance, a high ROUGE score is much easier to achieve on shorter summaries.

Copy rate The *copy rate* is another content-based metric to analyze the matching words between the document and the candidate summary. The copy rate extends the NOVEL-n from Section 4.3.2 by averaging the result for $n \in [1, 2, 3, 4]$. The respective mean μ_n is

$$\mu_n = \frac{1}{n} \sum_{k=1}^{n} \text{NOVEL-}k(s, d). \tag{6.1}$$

In this thesis, μ_4 for $n = 4$ is denoted as *copy rate*. This score is interpretable as a percentage value of copied n-grams in comparison to the total number of n-grams in the generated summary. Thus, the copy rate is an indicator of the abstractive abilities and the level of paraphrasing of the trained model.

Repetition rate (RR) Another metric is the *repetition rate* which scores a candidate summary by the number of repetitive words. This is a useful metric for text summarization since models tend to repeat words and entire phrases [79, 61]. The repetition rate becomes even more interesting for smaller datasets as used in this thesis. In general, less repetition indicates a better summary, and the aim is to overcome repetition entirely. The used metric follows an approach by Cettolo, Bertoldi, and Federico [8] which calculates the repetition rate for the n-grams of a given text. More specifically, the approach computes the geometric mean to address the exponential rate decay that exists when n increases [8]. In a similar notation to the ROUGE score in Section 6.1, the repetition rate RR-$n(s)$ of a candidate summary s is defined as

$$\text{RR-}n(s) = \left(\prod_{k=1}^{n} \frac{||f_{ng}(s,k) - f_{ng}(s,k,1)||}{||f_{ng}(s,k)||} \right)^{(1/n)} \tag{6.2}$$

where n is the maximum number of n-grams, $f_{ng}(s,k)$ is a function creating a list of k-grams of s and $f_{ng}(s,k,1)$ consists of *unique* k-grams of s. Furthermore, $||\bullet||$ is the number of words in a set. Similar to the copy rate beforehand, RR-4 with $n = 4$ is referred to as *repetition rate* (short: *RR*) for the remainder of this chapter.

Exemplary summaries Section 4.3 already points out the difficulties of measuring the quality of generated summaries with content-based n-gram metrics. Therefore, the following sections frequently show exemplary predictions from the test dataset to give an insight into the quality of the summaries. These are subjectively selected and not evaluated using qualitative methods.

6.2. Baseline analysis

The baseline of the experiments in this chapter is an implementation of the CopyNet model [32] (see Sections 5.1 and 5.2.3). More specifically, the experiments focus on text summarization of multi-sentence documents with the non-anonymized version of the *CNN / DailyMail (CNN/DM)* dataset [79] (see Section 4.2.2). An instance represents a pair of a document with hundreds of words and a target (reference) summary with tens of words. Similar to recent work [79, 25], the input documents are clipped to a length of 400 words and the target summaries to a length of 100.

The embeddings of dimensionality $d_{emb} = 128$ are randomly initialized and learned during training. The encoder is a single-layer bidirectional LSTM of dimensionality $d_{enc} = 512$. The decoder is a single-layer LSTM of dimensionality $d_{dec} = d_{enc}$. For inference, the model uses *beam search* [43] with a beam size of 4. Following recent work [79, 25], the model is trained with *Adagrad* [21], a learning rate $\eta = 0.15$ an initial accumulator value of 0.1. The same optimizer is reused throughout all experiments.

Approach	ROUGE-1	ROUGE-2	ROUGE-L
CopyNet (this thesis)	35.89	15.90	33.10
See, Liu, and Manning [79] (PG)	36.44	15.66	33.42
See, Liu, and Manning [79] (best)	39.53	17.28	36.38
Kryściński et al. [44]	40.19	17.38	37.52
lead-3 baseline [79]	40.34	17.70	36.57

Table 6.1.: ROUGE scores of the baseline model and recent work on CNN / DailyMail. (PG) indicates a seq2seq model with pointer generator, (best) indicates the best model of the correspsonding work. The lead-3 baseline extracts the first three sentences of the document as a summary.

For the plausibility of the subsequent experiments, Table 6.1 illustrates the achieved ROUGE scores of the model and recent work from Table 4.1. The CopyNet model implements a similar mechanism to the pointer-generator by See, Liu, and Manning [79] and achieves comparable results. Nonetheless, approaches with components such as reinforcement learning and coverage perform better (see Section 4.2.1). However, the aim of the model is not to achieve state-of-the-art results but to be a reasonable baseline for the following experiments. Essentially, the primary outcome of this analysis is that the model performs sufficiently well to meet the requirements of this thesis.

For a better understanding of the output generation, Table 6.2 shows an exemplary summary as generated by the CopyNet model. Even though the candidate summary contains relatively fluent text, there are two shortcomings. First, the model copies not only words but entire sentences from the source document (see the bold text passages). Regarding the copy rate, the model copies 92.12% of the n-grams from the source document (see Table 6.4). Secondly, the candidate summary contains repetitive words and sequences of words (*“ young lioness , ” and “ young lioness , ” and “ young lioness ”*). The remainder of this chapter deals with these shortcomings in more detail.

6.2.1. Downsized datasets for this thesis

The CNN / DailyMail dataset is too extensive for a plausible evaluation of the transfer learning capabilities with recent approaches like OpenAI GPT, BERT, and ELMo. As illustrated in Table 6.3, the full, respectively LARGE dataset contains almost 300,000 training pairs and 13,500 validation samples. Each training run is performed on a single GPU with 12 GB of RAM (NVIDIA TITAN X[1]). Under consideration of this hardware configuration and the given implementation, the full dataset requires more than three days to converge.

Besides these computational limitations, Ruder [73] hypothesizes that pre-trained models have a more significant impact on smaller datasets. For this reason, a valuable contribution is the analysis of sequential transfer learning for smaller datasets with fewer data.

[1] https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/

Document (article)	
-lrb- cnn -rrb- **the fbi charged a philadelphia woman on thursday with trying to travel overseas to fight for isis .** she 's one of three women arrested this week on terror charges . two new york women were also taken into custody . an fbi complaint cites numerous social media messages dating back to august 2013 that were sent by **keonna thomas , 30 , also known as " young lioness " and " fatayat al khilafah . "** one twitter message said , " if we truly knew the realities ... we all would be rushing to join our brothers in the front lines pray allah accept us as shuhada -lsb- martyrs -rsb- . " another said , " when you 're a mujahid -lsb- violent jihadi fighter -rsb- your death becomes a wedding . " **the fbi said thomas purchased an electronic visa to turkey on march 23 .** turkey is known as the easiest place from which to enter syria and join isis . an isis manual advises recruits to buy round-trip tickets to vacation spots such as spain and then purchase tickets for their real destination once they arrive overseas , the fbi said . on march 26 , thomas purchased a ticket to barcelona , with a march 29 departure and an april 15 return to the united states , the complaint said . it 's not clear when or where she was arrested . she was charged with knowingly attempting to provide material support and resources to a designated foreign terrorist organization . she could be sentenced to 15 years in prison . on thursday , noelle velentzas , 28 , and her former roommate , asia siddiqui , 31 , were arrested in new york and accused of planning to build an explosive device for attacks in the united states , federal prosecutors said . in the past 18 months , the justice department 's national security division has prosecuted or is prosecuting more than 30 cases of people attempting to travel abroad to join or provide support to terrorist groups . of those cases , 18 allegedly involve support to isis . " the terrorist threat is more decentralized , more diffuse , more complicated , " homeland security secretary jeh johnson told reporters thursday . " it involves the potential lone wolf actor , it involves the effective	
Candidate summary	**Reference Summary**
the fbi charged a philadelphia woman on thursday with trying to travel overseas to fight for isis . keonna thomas , 30 , also known as " young lioness , " and " young lioness , " and " young lioness " and " fatayat al khilafah " the fbi said thomas purchased an electronic visa to turkey on march 23 .	the fbi cites social media messages sent by keonna thomas , 30 . she 's accused of trying to travel overseas to join isis . thomas is one of three women facing federal terror charges this week .

Table 6.2.: Exemplary output instance of the CopyNet model.

Consequently, the experiments in the remainder of this chapter use three subsets of the CNN / DailyMail dataset (see Table 6.3). The subsets are sampled from the k starting training and validation instances of CNN / DailyMail. The intuition of Base is to represent a task with much experience. This primarily reduces the computational power of Large during

Name	Dataset	Training	Validation	Testing	Tr. duration
Large	CNN / DailyMail	287,227	13,368	11,490	$>$ 3 days
Base	$\subseteq$ Large	100,000	4,000	11,490	$\sim$ 18 hours
Small	$\subseteq$ Base	20,000	1,000	11,490	$\sim$ 4 hours
Mini	$\subseteq$ Small	3,000	500	11,490	1h20min

Table 6.3.: Differently sized subsets of CNN / DailyMail for following experiments.

training. In contrast to this, SMALL and MINI refer to tasks with a small and respectively a very small number of training and validation instances. These two subsets are used to show learning patterns of the summarization model on fewer data and to further analyse the abilities of pre-trained models. Nevertheless, each subset utilizes the full testing set of 11,490 instances for evaluation.

6.2.2. Evaluation of the datasets

In order to gain a first impression of the quality of summaries, the CopyNet model is trained with the three subsets in a similar configuration to the full dataset beforehand. Each dataset uses the same vocabulary consisting of the 50,000 most frequent words of the full CNN/DM dataset. In order to reduce the computational power, the experiments on smaller datasets are trained for ten instead of 25 epochs. Furthermore, the embeddings are reduced to a dimension d_{emb} = 100 which ensures the comparability to the transfer learning approaches in subsequent sections. Table 6.4 lists the ROUGE scores, length, copy rate and repetition rate (RR) for a single training run on the corresponding datasets.

Dataset	ROUGE-1	ROUGE-2	ROUGE-L	Length	Copy rate	RR
LARGE	35.89	15.90	33.10	54.02	92.12%	0.10
BASE	26.91	9.93	23.72	22.43	92.80%	0.00
SMALL	26.20	9.90	23.90	41.53	87.66%	0.36
MINI	14.93	2.12	13.77	20.80	31.89%	0.25
BASE – LARGE	-8.98	-5.97	-9.38	-31.59	0.68%	-0.10
SMALL – BASE	-0.71	-0.03	0.17	19.10	-5.14%	0.36
MINI – SMALL	-11.27	-7.77	-10.13	-20.73	-55.77%	-0.11

Table 6.4.: Results of the full CNN / DailyMail dataset and the base, small and mini subsets.

Large summary	Base Summary
the fbi charged a philadelphia woman on thursday with trying to travel overseas to fight for isis . keonna thomas , 30 , also known as “ young lioness , ” and “ young lioness , ” and “ young lioness ” and “ fatayat al khilafah ” the fbi said thomas purchased an electronic visa to turkey on march 23 .	keonna thomas , 30 , also known as “ young lioness ” and “ fatayat al khilafah ” turkey is known as the easiest place from which to enter syria and join isis .
Small summary	**Mini Summary**
thomas purchased an electronic visa to turkey on march 23 . turkey is known as the easiest place from which to enter syria and join isis . she could be sentenced to 15 years in prison .	new : fbi : fbi to turkey to turkey

Table 6.5.: Exemplary output instances of the large, base, small and mini dataset.

The first finding of Table 6.4 is that BASE generates much shorter summaries than LARGE while copying many words and sequences from the source document. In contrast to this, the average of SMALL increases by over 18 tokens per summary compared to BASE and still reaches similar ROUGE results. However, the repetition rates of BASE and SMALL indicate that models with fewer data and shorter training times achieve higher ROUGE scores with much more repetition. Finally, MINI scores significantly worse than larger datasets and creates short, fragmented and repetitive summaries.

The findings of Table 6.4 are further underlined with the exemplary output instance (see Table 6.2) of the different datasets in Table 6.5. While MINI creates a fragmented and nonsense summary, SMALL achieves a higher fluency but still misses essential parts compared to the reference summary. In summary, the results show that the text summarization system requires much data and long training times to produce satisfactory results.

6.3. Self-attention

The baseline model uses recurrent states in the form of LSTM networks that encode and decode sequences. Regarding the Sections 2.5.2 and 2.6, self-attention and the Transformer in sequence-to-sequence tasks have shown promising results. Thus, this experiment analyses the benefits of self-attention over recurrent states during encoding. More specifically, *stacked-multi-head attention* (see Section 2.6.2) replaces the single-layer bidirectional LSTM of hidden size $N = 512$. The self-attention has a hidden dimension $H = 256$ and uses $N = 4$ layers. For the multi-head attention, the model uses $h = 8$ heads each of which computes the attention of dimensionality $d_k = d_v = d_q = 1024$ (see Section 2.6.2). Table 6.6 compares the results of the biLSTM and self-attention on the different datasets for a single training run.

	BASE		SMALL		MINI	
	biLSTM	**SA**	**biLSTM**	**SA**	**biLSTM**	**SA**
ROUGE-1	26.91	27.76	26.20	32.85	14.93	2.14
ROUGE-2	9.93	10.34	9.90	12.83	2.12	0.04
ROUGE-L	23.72	24.50	23.90	29.76	13.77	2.04
Length	22.43	29.21	41.53	58.30	20.80	10.91
Copy rate	92.80%	99.56%	87.66%	91.92%	31.89%	19.49%
RR	0.00	0.00	0.36	0.11	0.25	0.21

Table 6.6.: Results of the large, base, small and mini dataset.

For BASE with 100,000 training pairs, the self-attention achieves similar ROUGE scores but generates longer summaries than the biLSTM. However, the self-attention copies almost any (copy rate = 99,56%) words and sequences from the source document. Another valuable finding from Table 6.6 is that self-attention produces significantly better results with less repetition and longer sequences on SMALL. The comparison of two exemplary output summaries in Table 6.7 further substantiates these observations. Regarding the results of MINI, the dataset and training time is not sufficient to create meaningful and fluent output. However, self-attention has a higher confusion in the beginning and does not generate any useful content.

Small biLSTM	Small Self-Attention	Reference
thomas purchased an electronic visa to turkey on march 23 . turkey is known as the easiest place from which to enter syria and join isis . she could be sentenced to 15 years in prison	fbi charged a philadelphia woman on thursday with trying to travel overseas to fight . she 's one of three women arrested this week on terror charges . new york women were also taken into custody .	the fbi cites social media messages sent by keonna thomas , 30 . she 's accused of trying to travel overseas to join isis . thomas is one of three women facing federal terror charges this week .
yahya rashid , a uk national from northwest london , was detained at luton airport . he is due to appear in westminster magistrates ' court on wednesday . he is due to appear in westminster magistrates ' court on wednesday	a uk national from northwest london , was detained at luton airport on tuesday after he arrived on a flight from istanbul . he been charged with engaging in conduct in preparation of acts of terrorism .	london 's metropolitan police say the man was arrested at luton airport after landing on a flight from istanbul . he 's been charged with terror offenses allegedly committed since the start of november

Table 6.7.: Exemplary output instances using a biLSTM and a self-attention encoder.

6.4. Pre-Trained word embeddings

The previous experiments randomly initialize and train the word embeddings during training. This section uses a first approach of transfer learning by initializing the embedding layer with pre-trained GloVe embeddings of dimensionality 100 (see Section 2.3.2). Table 6.8 illustrates the results of GloVe embeddings in comparison to the baseline results of the three datasets from table 6.4.

	Base		Small		Mini	
	Base	**GloVe**	**Base**	**GloVe**	**Base**	**GloVe**
ROUGE-1	26.91	31.93	26.20	27.49	14.93	17.28
ROUGE-2	9.93	12.49	9.90	10.56	2.12	6.58
ROUGE-L	23.72	28.69	23.90	25.26	13.77	16.13
Length	22.43	50.17	41.53	43.81	20.80	15.45
Copy rate	92.80%	94.96%	87.66%	84.31%	31.89%	81.00%
RR	0.00	0.13	0.36	0.33	0.25	0.03

Table 6.8.: GloVe results of the base, small and mini dataset.

In general, the model with GloVe embeddings achieves improvements across all datasets. Interestingly, the pre-trained word embeddings lead to much longer sequences with more repetition for Base whereas the repetition decreases for Small. Another finding from Table 6.8 is that the embeddings have a great impact on Mini. Even though the summaries are not comparable to larger datasets and training times, the copy rate of 81,00% in GloVe comparison to 31.89% in Base suggest that the summaries express the content better.

In summary, the results demonstrate that pre-trained GloVe embeddings can be a valuable starting point for summarization models with datasets of different sizes. Similar to other tasks in NLP, recent work for text summarization incoorperates GloVe embeddings [63, 40, 25] instead of learning the embeddings from scratch. However, these approaches do not explicitly report the influence of GloVe embeddings.

6.5. Contextual embeddings

The previous word embeddings are a first step towards sequential transfer learning in NLP. Going one step further, this section extends the CopyNet model with the three approaches from Sections 3.2 and 3.3: *ELMo* [65], *BERT* [19] and *OpenAI GPT* [68]. These approaches are denoted as *contextual embeddings* and added in addition to the word embeddings.

Let w denote the word embedding and c the contextual embedding of an input x. The overall embedding is the concatenation of the word as well as the contextual embedding and denoted as $e = [w; c]$. The embeddings of dimensionality $d_e = d_w + d_c$ are the input of the encoder. Due to their large number of parameters, contextual embeddings are not optimized during training. However, Section 6.5.2 presents the results of fine-tuning and optimizing ELMo. The word embeddings are not pre-trained in order to be comparable to the baseline results. Altogether, Table 6.9 shows the results of contextual embeddings with ELMo, BERT and OpenAI GPT on the Base dataset for a single training run.

	Baseline	**GloVe**	**ELMo**	**GPT**	**BERT**
Dim. (d_e)	100	100	1124	868	868
ROUGE-1	26.91	31.93	32.16	31.33	32.17
ROUGE-2	9.93	12.49	13.30	12.56	14.02
ROUGE-L	23.72	28.69	29.26	27.94	29.75
Length	22.43	50.17	44.75	40.71	35.23
Copy rate	92.80%	94.96%	90.95%	96.69%	88.07%
RR	0.00	0.13	0.13	0.06	0.08

Table 6.9.: Results of contextual embeddings on the base dataset.

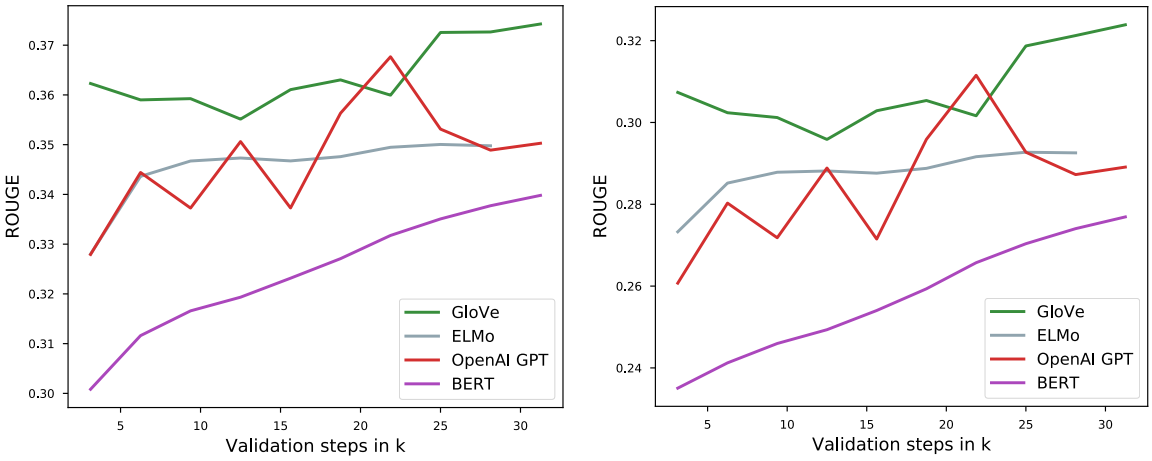
The dimensionality of the embeddings d_e indicates the higher computational costs of models with contextual embeddings. Comparing the three contextual approaches, ELMo uses a single layer bidirectional LSTM (see Section 3.2.2) of dimensionality $d_c = 512 + 512$ and OpenAI GPT utilizes a Transformer decoder of dimensionality $d_c = 512$. The BERT model in Table 6.9 refers to the pre-trained BERT_{BASE} model (see Section 3.3.3) since the hardware configuration does not meet the requirements of the large model.

Regarding the results of the two Transformer-based approaches, BERT and OpenAI GPT outperform the baseline by around four to five ROUGE points. Since both achieve very similar results, the bidirectional context of BERT (see Section 3.3.3) only leads to marginal changes in this setup, especially with the implications as described in the following paragraph.

Another finding of the experiment is that ELMo achieves similar results to GloVe. Hence, ELMo embeddings encourage the model to create longer sequences with more repetition than the Transformer-based approaches. In summary, contextual embeddings do have slight improvements compared to pre-trained GloVe embeddings.

BERT implications Under the configuration and implementation of the experiments, BERT embeddings come with a major drawback. By design, the pre-trained BERT model can process sequences with up to 512 word-piece tokens [19] (see Section 2.6). In the experiments, documents contain at most 400 words which may exceed the length of 512 after word-piece tokenization. As a result, the overflowing word pieces are ignored and it became apparent that the model is not robust enough. In the above experiment, the model creates empty summaries for around 86.07% of the test instances. Thus, the results in Table 6.9 reflect only 1,600 of the full 11,490 test instances. Furthermore, Section 6.5.1 confirms this observation on smaller datasets and Section 6.6 discusses an improvement for future approaches.

The course of training process Figure 6.1 illustrates the course of validation ROUGE scores for GloVe, ELMo, OpenAI GPT and BERT during the training run from Table 6.9. The results are an indicator for the learning process. Since the ROUGE scores are calculated on the corresponding validation set with 4,000 instances, the scores in Table 6.9 during testing are lower for all approaches. Regarding the course for ROUGE-1 and ROUGE-L, the contextual embeddings start with significantly lower scores than GloVe in the beginning but increase during training. The values of OpenAI GPT are erratic but generally growing. An important note is that the BERT results neglect the blank summaries (see above).



(a) ROUGE-1 (b) ROUGE-L

Figure 6.1.: Course of validation ROUGE scores of BASE with contextual embeddings.

6.5.1. Smaller datasets

In the next step, the downsized datasets are used to evaluate the hypothesis by Ruder [73] that pre-trained models have a greater impact on smaller datasets. As described beforehand, the BERT integration in the concrete scenarios of this thesis is fragile because 86.07% of the generated summaries are blank (see Section 6.5). Experiments with BERT on Small confirm similar results with 83.54% of blank summaries. On Mini, however, the model creates summaries for each training instance but these consist predominantly of `@@UNKNOWN` tokens. Hence, this section focuses on ELMo and OpenAI GPT having a more robust integration in the given workflow. Table 6.10 compares the results of Small and Mini with GloVe, ELMo and OpenAI GPT (GPT) to the baseline results of Table 6.4.

	Small				Mini			
	Base	**GloVe**	**ELMo**	**GPT**	**Base**	**GloVe**	**ELMo**	**GPT**
ROUGE-1	26.20	27.49	27.53	30.01	14.93	17.28	23.29	19.34
ROUGE-2	9.90	10.56	10.51	11.93	2.12	6.58	9.12	7.90
ROUGE-L	23.90	25.26	25.49	27.82	13.77	16.13	21.94	18.24
Length	41.53	43.81	38.83	43.09	20.80	15.37	52.25	15.84
Copy rate	87.66%	84.31%	78.29%	81.73%	31.89%	80.85%	83.48%	73.44%
RR	0.36	0.33	0.29	0.24	0.25	0.03	0.36	0.03

Table 6.10.: Results of GloVe and OpenAI GPT on the small and mini dataset.

In general, the results in Table 6.10 show a positive influence of contextual embeddings on smaller datasets. For Small, the Transformer-based OpenAI GPT outperforms the GloVe embeddings with around 2-3 ROUGE-$*$ points while creating summaries of the same length and copying less. In contrast to this, ELMo shows very similar results to the pre-trained GLoVe embeddings. Besides this, Table 6.11 exemplifies the qualitative results. For the particular output generations, the contextual approach learns to recognize more essential parts of the input document (see Table 6.2) than the baseline model.

The experiments on Mini in Table 6.10 further support the results of Small. The GloVe embeddings lead to significant improvements over the baseline and OpenAI GPT achieves between three and five ROUGE-$*$ points more than GloVe. On top of this, the generated summaries are around fifteen to twenty words on average long and thus much shorter than Small. However, ELMo is an exception since the model achieves higher ROUGE-$*$ scores by generating much longer summaries with more repetition (RR = 0.36). Regarding the exemplary output generations in Table 6.11, the summaries do not contain meaningful content regarding the input text (see Table 6.2). Furthermore, the summary for ELMo repeats the same sequence of words five times in a row.

In comparison to the results of Base in Table 6.9, the findings support the hypothesis that sequential transfer learning has a higher influence on smaller datasets [73].

Small GloVe	Small ELMo	Small OpenAI GPT
two new york women were arrested this week on terror charges . turkey is known as the easiest place from syria and join isis . turkey is known as the easiest place from which enter syria and join isis .	fbi : “ when you ’re a mujahid -lsb- violent jihadi fighter -rsb- your death becomes a wedding ” turkey is known as the easiest place from which to enter syria and join isis . turkey is known as the potential lone wolf actor .	new : an fbi complaint cites numerous social media messages back back to august 2013 . she could be sentenced to 15 years in prison . she could be sentenced to 15 years in prison .
Mini GloVe	**Mini ELMo**	**Mini OpenAI GPT**
homeland security secretary jeh johnson says it ’s not clear when or where she was arrested .	charleston is suspected of involvement in 32 commercial robberies dating to november 2013 . charleston is suspected of involvement in 32 commercial robberies . **charleston is suspected of involvement in 32 commercial robberies . (×5)**	keonna thomas , 30 , also known as “ young lioness ”

Table 6.11.: Exemplary output instances of the model with pre-trained GloVe embeddings and contextual OpenAI GPT embeddings on the small and mini dataset.

6.5.2. ELMo fine-tuning

In the previous experiments, contextual embeddings are fixed parameters and not optimized during training. This is mainly due to the large number of parameters and the corresponding training complexity. However, the authors of AllenNLP refer to the fact that ELMo can negatively impact the first iterations until the bidirectional language model resets its internal states[2]. For this reason, Table 6.12 compares result of fixed and learned ELMo embeddings on Base and Small with a single training run.

	ROUGE-1	ROUGE-2	ROUGE-L	Length	Copy rate	RR
Base (fixed)	32.16	13.30	29.26	44.75	90.95%	0.13
Base (learned)	28.43	12.51	26.24	26.94	88.20%	0.06
Small (fixed)	27.53	10.51	25.49	38.83	78.29%	0.29
Small (learned)	28.14	10.85	26.10	39.96	76.52%	0.30

Table 6.12.: Results of fixed parameters and learning ELMo embeddings on Base and Small.

The experiments show different findings for the two datasets. For Small with 20,000 training instances, the model achieves very similar results with fixed and learned ELMo embeddings. In contrast to this, the model on Base with 100,000 training instances achieves higher ROUGE-∗ scores with fixed parameters than fine-tuning the ELMo embeddings. However, learning ELMo embeddings leads to shorter summaries with less repetition.

[2] https://github.com/allenai/allennlp/blob/master/tutorials/how_to/elmo.md

Another interesting observation on BASE during training is that the model converges after five epochs with learned parameters whereas the run with fixed parameters requires nine epochs to converge. Consequently, a simple fine-tuning without methods like *gradual unfreezing* [37] is not beneficial in this scenario. In summary, the results show that fine-tuning ELMo embeddings requires further investigations and specific techniques in order to be beneficial in the scenario of this thesis.

6.6. Summary and discussion

The implementation of the CopyNet model achieves comparable results to pointer-generator approaches of recent work [79]. In general, the findings motivate for sequential transfer learning to improve the generation of summaries. First, pre-trained GloVe, ELMo and BERT embeddings have found to be useful across the majority of datasets in the experiments. Furthermore, contextual embeddings have a stronger influence on tasks with fewer data. The following section discusses the essential findings of the experiments in more detail.

Level of abstraction *Abstractive methods* aim to paraphrase the content of the source document in a fluent and concise summary. As the observed copy rates across all experiments suggest, the CopyNet model incorporates fewer novel words for more extensive datasets and longer training times. Instead of paraphrasing sentences, the model learns to identify and extract the essential sequences of the document. This behavior of identifying and copying sets of words from the source goes in the direction of *extractive methods*. Thus, one finding of the experiments is that the model prefers to copy instead of generating novel words from the vocabulary.

Interpretability of metrics The challenges in evaluating text summaries are discussed in several parts of this thesis (see Section 4.3 and Section 6.1) and supported by the practical part in this chapter. First, n-gram based metrics classify words either as true or as false. In practice, *smoothing* is a common practice to address this issue [10]. As one finding of this thesis, the combination of ROUGE, the copy and repetition rate as well as the length of summaries has found to be a reliable indicator for the quality of a summary. Nonetheless, human-based observations of the summaries are indispensable, especially in the early stages.

Model fine-tuning One objective of sequential transfer learning is to replace the majority of a downstream model with the extensively pre-trained language model. Many researched downstream tasks like text classification or question answering create a thin layer for classification on top of the pre-trained models [19, 68]. Regarding text summarization and the implementation in this thesis, the model requires task-specific components such as a pointer-generator and a decoding component to generate summaries. In the connection, the large-scale language models with additional summarization-specific components require enormous computational power. Furthermore, the Transformer-based approaches [19, 68] are very sensitive to hyperparameters [67]. Hence, efforts in replacing the encoder of a summarization model with a pre-trained language model were not successful in the preliminary stages of this work.

As a result, the experiments in this thesis treat pre-trained models as contextual embeddings. Therefore, the deep neural language models are compressed to fixed dimensional vectors and the encoder, as well as the decoder, have to be trained from scratch for each experiment. Thus, the sequential transfer learning for the investigated scenario in text summarization in this thesis is limited to a better understanding of input words and sequences.

Implementation obstacles In the conducted experiments, the word-piece tokenization of BERT [19] has found to be problematic. While the CopyNet model needs to keep track of source words in order to copy them during decoding, the BERT model requires word-pieces to meet the fixed vocabulary of the pre-trained language model. On top of this, BERT has a maximum limit of 512 word-pieces whereas the CNN/DM dataset has 781 words on average. One feasible solution might be a *sliding window* similar to the approach by Dai et al. [16]. Nevertheless, the specific problems such as the word-piece tokenization remain.

Further work in multi-task learning Sequential transfer learning and approaches like ELMo, OpenAI GPT, and BERT are recent developments that have not been investigated extensively for text summarization. Gehrmann, Deng, and Rush [25] add auxiliary ELMo embeddings to their summarization model but do not incorporate the Transformer-based approaches (OpenAI GPT, BERT). Coming from the other direction, a promising work is the OpenAI GPT 2 [69]. The further development of OpenAI GPT has over a billion parameters and addresses several tasks with multi-task learning (see Section 4.1). For text summarization, the approach evaluates the language model with the CNN / DailyMail dataset. Table 6.13 illustrates the results.

	ROUGE-1	**ROUGE-2**	**ROUGE-L**
Small (no hint)	21.58	4.03	19.47
Small (`TL;DR:`)	29.34	8.27	26.58

Table 6.13.: Results of OpenAI GPT 2 on CNN / DailyMail [69].

The model summarizes without any modifications (*zero-shot transfer learning*). Thus, the document is the input and the model generates the hundred following words for this document [69]. Even though the results are not similar to models with task-specific components, the approach shows that a pre-trained language model learns to summarize better if the article ends with an appended `TL;DR:` token.

7. Conclusion and outlook

Recent developments show promising results in sharing knowledge across different tasks in NLP. Techniques for transfer learning are evolving and focus on the specific requirements of neural networks models in NLP [37]. In this context, many recent approaches are inductive methods where the source task is a large-scale language model and the target task is arbitrary. Their powerhouses are exceptionally extensive models that are trained with immense computing power and resources [19, 69]. These hold enormous potential for tasks and problems in natural language with fewer experience.

In abstractive text summarization, deep neural networks obtain enhancing results in conjunction with task-specific components. The experiments of this thesis have analyzed the capabilities of a deep neural network to summarize news articles from the CNN / DailyMail dataset [61]. The observations show that generated summaries often provide the crucial parts of the corresponding text. However, many words and sequences of words are extracted and copied from the text without any modifications. Furthermore, the model requires tens of thousands of training pairs in order to learn to summarize news articles.

Building the bridge to transfer learning for NLP, widespread approaches [19, 37, 68] omit investigations for the task of text summarization or any further sequence-to-sequence task. Hence, one primary achievement of this thesis is a workflow to apply transfer learning for text summarization. The practical foundation includes an implementation of the CopyNet summarization model [32] with the core components of the research framework AllenNLP [23]. This setup promotes configurable experiments with exchangeable components and thus enables the integration and comparison of transfer learning approaches in neural summarization models.

The conducted experiments suggest that sequential transfer learning is beneficial for summarizing. Pre-trained GloVe embeddings [64] and contextual embeddings extracted from ELMo [65], BERT [19] and OpenAI GPT [68] have found to be useful across three differently sized subsets of CNN / DailyMail. Furthermore, it became apparent that contextual embeddings have a greater impact on smaller subsets with less training records. This observation supports the hypothesis [73] that sequential transfer learning is more significant on datasets with fewer instances. In conclusion, the findings and observations in this thesis motivate for further investigations in sequential transfer learning for text summarization.

For the comparison of different approaches, the experiments in this thesis include pre-trained models as fixed-length embeddings. The influence of these embeddings is bordered to the first layer of a neural network. Hence, future work might replace entire parts of summarization models with pre-trained language models. Another finding of the conducted experiments is the high number of extracted words and sequences of words from the document. These results are at odds with the hypothesis that transfer learning may assist the model to summarise in a more abstractive and fluent style. For this reason, the influence of sequential

transfer learning given the CopyNet model has found to be constrained by the task-specific copy mechanism. Consequently, future research might focus on different summarization models and approaches. Finally, the results of this work exclusively focus on the CNN / DailyMail dataset and further investigations might verify these results on different datasets.

In conclusion, transfer learning paves the way for enhancements in many fields of natural language processing. Nonetheless, transfer learning techniques that are more suitable for sequence-to-sequence tasks have yet to be developed. Especially for text summarization, recent neural networks require individual components that exclusively address the challenges in text summarization. In order to share gained knowledge across multiple tasks in NLP, one objective is yet the incremental reduction of these task-specific components. One future direction might include improvements in sequential transfer learning with more powerful models and enhanced transfer learning techniques. Another promising direction is the field of multi-task learning which teaches a model to achieve multiple tasks at the same time [69]. In any case, natural language processing with deep learning remains one of the suspenseful and fast-moving fields in artificial intelligence.

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. Sept. 2014. arXiv: 1409.0473.

[2] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. *Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors*. Tech. rep. 2014. URL: http://ronan.collobert.com/senna/.

[3] Yoshua Bengio. "Learning Deep Architectures for AI". In: *Foundations and trends in Machine Learning* 2.1 (2009), pp. 1–127.

[4] Yoshua Bengio. "Neural net language models". In: *Scholarpedia* 3.1 (2008), p. 3881. DOI: 10.4249/scholarpedia.3881.

[5] Yoshua Bengio et al. "A Neural Probabilistic Language Model". In: *Journal of Machine Learning Research* 3 (2003), pp. 1137–1155. URL: http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf.

[6] Erik Cambria and Bebo White. "Jumping NLP Curves: A Review of Natural Language Processing Research". In: *IEEE Computational Intelligence Magazine* 9.2 (May 2014), pp. 48–57.

[7] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. *An Analysis of Deep Neural Network Models for Practical Applications*. May 2016. arXiv: 1605.07678.

[8] Mauro Cettolo, Nicola Bertoldi, and Marcello Federico. "The Repetition Rate of Text as a Predictor of the Effectiveness of Machine Translation Adaptation". In: *Proceedings of the 11th Biennial Conference of the Association for Machine Translation in the Americas (AMTA 2014)*. 2014, pp. 166–179.

[9] Ciprian Chelba et al. *One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling*. 2014. arXiv: 1312.3005v3.

[10] Stanley F. Chen and Joshua Goodman. "An empirical study of smoothing techniques for language modeling". In: *Computer Speech & Language* 13.4 (1999), pp. 359–394.

[11] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078.

[12] François Chollet. *Deep Learning with Python*. Manning, 2018.

[13] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555.

[14] Ronan Collobert and Jason Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning". In: *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[15] John M Conroy and Dianne P O'leary. "Text Summarization via Hidden Markov Models and Pivoted QR Matrix Decomposition". In: *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2001, pp. 406–407.

[16] Zihang Dai et al. “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context”. In: (Jan. 2019). arXiv: 1901.02860.

[17] Michał Daniluk et al. *Frustratingly Short Attention Spans in Neural Language Modeling*. Feb. 2017. arXiv: 1702.04521.

[18] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2009, pp. 248–255.

[19] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. arXiv: 1810.04805.

[20] Bonnie Dorr, David Zajic, and Richard Schwartz. “Hedge trimmer: A parse-and-trim approach to headline generation”. In: *Proceedings of the HLT-NAACL 03 on Text summarization workshop-Volume 5*. Association for Computational Linguistics, 2003, pp. 1–8.

[21] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization * Elad Hazan”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2121–2159.

[22] Markus Freitag and Yaser Al-Onaizan. *Beam Search Strategies for Neural Machine Translation*. 2017. arXiv: 1702.01806v2.

[23] Matt Gardner et al. “AllenNLP: A Deep Semantic Natural Language Processing Platform”. In: *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*. 2018, pp. 1–6.

[24] Jonas Gehring et al. “Convolutional sequence to sequence learning”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1243–1252. arXiv: 1705.03122v3.

[25] Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. “Bottom-Up Abstractive Summarization”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, pp. 4098–4109.

[26] Yoav Goldberg. “Neural network methods for natural language processing”. In: *Synthesis Lectures on Human Language Technologies* 10.1 (2017), pp. 1–309.

[27] Ian J. Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems* (June 2014), pp. 2672–2680. arXiv: 1406.2661.

[28] A. Graves and J. Schmidhuber. “Framewise phoneme classification with bidirectional LSTM networks”. In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 4. IEEE, pp. 2047–2052.

[29] Alex Graves. *Sequence Transduction with Recurrent Neural Networks*. 2012. arXiv: 1211.3711.

[30] Alex Graves, Navdeep Jaitly, and Abdel Rahman Mohamed. “Hybrid speech recognition with Deep Bidirectional LSTM”. In: *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2013 - Proceedings*. 2013, pp. 273–278.

[31] Max Grusky, Mor Naaman, and Yoav Artzi. *Newsroom: A Dataset of 1.3 Million Summaries with Diverse Extractive Strategies*. Apr. 2018. arXiv: 1804.11283.

[32] Jiatao Gu et al. *Incorporating Copying Mechanism in Sequence-to-Sequence Learning*. 2016. arXiv: 1603.06393v3.

[33] Michael U Gutmann and Aapo Hyvärinen. "Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics Aapo Hyvärinen". In: *Journal of Machine Learning Research* 13 (2012), pp. 307–361.

[34] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Dec. 2015, pp. 770–778. arXiv: 1512.03385.

[35] Karl Moritz Hermann et al. "Teaching machines to read and comprehend". In: *Advances in neural information processing systems*. 2015.

[36] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780.

[37] Jeremy Howard and Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification*. Jan. 2018. arXiv: 1801.06146.

[38] Baotian Hu, Qingcai Chen, and Fangze Zhu. *LCSTS: A Large Scale Chinese Short Text Summarization Dataset*. June 2015. arXiv: 1506.05865.

[39] Jing Jiang. *A Literature Survey on Domain Adaptation of Statistical Classifiers*. 2008. URL: http://www.mysmu.edu/faculty/jingjiang/papers/da%7B%5C_%7Dsurvey.pdf.

[40] Yaser Keneshloo, Naren Ramakrishnan, and Chandan K. Reddy. *Deep Transfer Reinforcement Learning for Text Summarization*. 2018. arXiv: 1810.06667.

[41] J. Kiefer and J. Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function". In: *The Annals of Mathematical Statistics* 23.3 (Sept. 1952), pp. 462–466.

[42] Kevin Knight and Daniel Marcu. "Summarization beyond sentence extraction: A probabilistic approach to sentence compression". In: *Artificial Intelligence* 139.1 (2002), pp. 91–107.

[43] Philipp Koehn. "Koehn, Philipp. "Pharaoh: a beam search decoder for phrase-based statistical machine translation models". In: *Conference of the Association for Machine Translation in the Americas*. Springer, Berlin, Heidelberg, 2004, pp. 115–124.

[44] Wojciech Kryściński et al. *Improving Abstraction in Text Summarization*. Aug. 2018. arXiv: 1808.07913.

[45] Omer Levy, Yoav Goldberg, and Ido Dagan. "Improving distributional similarity with lessons learned from word embeddings". In: *Transactions of the Association for Computational Linguistics 3* (2015), pp. 211–225.

[46] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of summaries". In: *Text Summarization Branches Out* (2004).

[47] Junyang Lin et al. *Global Encoding for Abstractive Summarization*. 2018. arXiv: 1805.03989.

[48] Zhouhan Lin et al. *A Structured Self-attentive Sentence Embedding*. 2017. arXiv: 1703.03130.

[49] Linqing Liu et al. *Generative Adversarial Network for Abstractive Text Summarization*. Nov. 2017. arXiv: 1711.09357.

[50] Peter J Liu et al. *Generating Wikipedia by Summarizing Long Sequences*. 2018. arXiv: 1801.10198.

[51] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. Aug. 2015. arXiv: 1508.04025.

[52] Andrew L. Maas et al. "Learning word vectors for sentiment analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. Portland, Oregon: Association for Computational Linguistics, 2011, pp. 142–150.

[53] Christopher D Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. London: MIT Press, 1999.

[54] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[55] Bryan McCann et al. *Learned in Translation: Contextualized Word Vectors*. 2017. arXiv: 1708.00107.

[56] Bryan McCann et al. *The Natural Language Decathlon: Multitask Learning as Question Answering*. June 2018. arXiv: 1806.08730.

[57] Stephen Merity et al. *Pointer Sentinel Mixture Models*. 2016. arXiv: 1609.07843.

[58] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781v3.

[59] Tomáš Mikolov et al. "Recurrent neural network based language model". In: *Eleventh annual conference of the international speech communication association*. 2010.

[60] Tom Mitchell. *Machine learning*. 1997. ISBN: 0070428077.

[61] Ramesh Nallapati et al. *Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond*. 2016. arXiv: 1602.06023.

[62] A Nenkova and K Mckeown. "Automatic Summarization". In: *Foundations and Trends R in Information Retrieval* 5.3 (2011), pp. 103–233. DOI: 10.1561/1500000015.

[63] Romain Paulus, Caiming Xiong, and Richard Socher. *A Deep Reinforced Model for Abstractive Summarization*. May 2017. arXiv: 1705.04304.

[64] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[65] Matthew E. Peters et al. *Deep contextualized word representations*. Feb. 2018. arXiv: 1802.05365.

[66] Matthew E. Peters et al. *Semi-supervised sequence tagging with bidirectional language models*. Apr. 2017. arXiv: 1705.00108.

[67] Martin Popel and Ondřej Bojar. "Training Tips for the Transformer Model". In: *The Prague Bulletin of Mathematical Linguistics* 110.1 (2018), pp. 43–70.

[68] Alec Radford et al. *Improving Language Understanding by Generative Pre-Training*. Tech. rep. 2018.

[69] Alec Radford et al. *Language Models are Unsupervised Multitask Learners*. 2019.

[70] Waseem Rawat and Zenghui Wang. "Deep convolutional neural networks for image classification: A comprehensive review". In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[71] Steven J. Rennie et al. *Self-critical Sequence Training for Image Captioning*. Dec. 2016. arXiv: 1612.00563.

[72] Rami Al-Rfou et al. *Character-Level Language Modeling with Deeper Self-Attention*. Aug. 2018. arXiv: 1808.04444.

[73] Sebastian Ruder. "Neural Transfer Learning for Natural Language Processing". PhD thesis. National University of Ireland, Galway, 2019.

[74] Sebastian Ruder, Ivan Vulić, and Anders Søgaard. *A Survey Of Cross-lingual Word Embedding Models*. June 2017. arXiv: 1706.04902.

[75] Alexander M. Rush, Sumit Chopra, and Jason Weston. *A Neural Attention Model for Abstractive Sentence Summarization*. Sept. 2015. arXiv: 1509.00685.

[76] Evan Sandhaus. "The New York Times Annotated Corpus". In: *Linguistic Data Consortium* 6(12):e26752, 2008 (2008).

[77] Tobias Schnabel et al. "Evaluation methods for unsupervised word embeddings". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2015, pp. 298–307.

[78] M. Schuster and K.K. Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[79] Abigail See, Peter J. Liu, and Christopher D. Manning. *Get To The Point: Summarization with Pointer-Generator Networks*. Apr. 2017. arXiv: 1704.04368.

[80] Dou Shen et al. "Document summarization using conditional random fields". In: *IJCAI*. 2007, pp. 2862–2867.

[81] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. Sept. 2014, pp. 3104–3112. arXiv: 1409.3215.

[82] Wilson L. Taylor. ""Cloze Procedure": A New Tool for Measuring Readability". In: *Journalism Bulletin* 30.4 (Sept. 1953), pp. 415–433.

[83] Zhaopeng Tu et al. *Modeling Coverage for Neural Machine Translation*. Jan. 2016. arXiv: 1601.04811.

[84] Ashish Vaswani et al. *Attention Is All You Need*. June 2017. arXiv: 1706.03762.

[85] Yonghui Wu et al. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. Sept. 2016. arXiv: 1609.08144.

[86] Jason Yosinski et al. *How transferable are features in deep neural networks?* Nov. 2014. arXiv: 1411.1792.

[87] Tom Young et al. *Recent Trends in Deep Learning Based Natural Language Processing*. 2018. arXiv: 1708.02709v8.

[88] Yukun Zhu et al. *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*. June 2015. arXiv: 1506.06724.

A. Appendix

A.1. AllenNLP configuration for text summarization on CNN / DailyMail

```
{
  "dataset_reader": {
      "type": "summarization",
      "lazy": true,
      "source_token_indexers": {
          "tokens": {
              "type": "single_id",
              "namespace": "source_tokens"
          }
      },
      "source_tokenizer": {
          "type": "word",
          "word_splitter": { "type": "just_spaces" }
      },
      "target_namespace": "target_tokens"
  },
  "iterator": {
      "type": "bucket",
      "batch_size": 32,
      "biggest_batch_first": true,
      "max_instances_in_memory": 1024,
      // ...
  },
  "model": {
      "type": "copynet_seq2seq",
      "attention": {
          "type": "linear",
          "activation": "tanh",
          "tensor_1_dim": 512,
          "tensor_2_dim": 512
      },
      "beam_size": 4,
      "encoder": {
          "type": "lstm",
          "bidirectional": true,
          "hidden_size": 256,
          "input_size": 128,
          "num_layers": 1
      },
      "max_decoding_steps": 100,
      "source_embedder": {
          "token_embedders": {
```

```
                "tokens": {
                    "type": "embedding",
                    "embedding_dim": 128,
                    "trainable": true,
                    "vocab_namespace": "source_tokens"
                }
            }
        },
        "token_based_metric": "rouge"
    },
    "train_data_path": {
        "type": "tuple",
        "is_training": true,
        "source_max_seq_leng": 400,
        "source_path": "data/sum/cnndm/train.txt.src",
        "target_max_seq_leng": 100,
        "target_path": "data/sum/cnndm/train.txt.tgt.tagged"
    },
    "validation_data_path": {
        "type": "tuple",
        "source_max_seq_leng": 400,
        "source_path": "data/sum/cnndm/val.txt.src",
        "target_max_seq_leng": 100,
        "target_path": "data/sum/cnndm/val.txt.tgt.tagged"
    },
    "trainer": {
        "cuda_device": 0,
        "grad_clipping": 5,
        "grad_norm": 2,
        "num_epochs": 20,
        "num_serialized_models_to_keep": 3,
        "optimizer": {
            "type": "adagrad",
            "initial_accumulator_value": 0.1,
            "lr": 0.15
        },
        "patience": 20
    },
    "vocabulary": { "directory_path": "work/allennlp/cnndm_base/vocabulary/50k" },
    "validation_iterator": {
        "type": "bucket",
        "batch_size": 32,
        "max_instances_in_memory": 1024,
        "padding_noise": 0.1,
        "sorting_keys": [[
          "source_tokens",
          "num_tokens"
        ]]
    }
}
```

Listing A.1: Experiment configuration in AllenNLP for the CopyNet model on the CNN/Daily Mail dataset.