Karlsruhe Institute of Technology

Neural Abstractive Summarization using Transformer Model

Master's Thesis of

Daniel Schumacher

at the Interactive Systems Lab
Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology (KIT)

Reviewer:	Prof. Dr. Alexander Waibel
Second reviewer:	Prof. Dr. Tamim Asfour
Advisor:	Dr. Jan Niehues

12. October 2018 – 11. April 2019

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, April 9, 2019

.....

(Daniel Schumacher)

Abstract

Automatic text summarization, a task of machine translation that automates the process of shortening a text while reserving the main ideas of the document. Previous approaches proposed in this field can be categorized into two basic types: extractive and abstractive summarization. While the former generates summarizations by deleting unimportant parts of the input sequence, the latter extracts the relevant information from the input sequence and generates a new, grammatically correct summarization.
This work deals with abstractive text summarization. The applicability of the *Transformer Model*, published by Vaswani et al. [2017], in the creation of abstract summaries, will be examined. For these comparisons, publically accessible datasets from existing research approaches will be used to provide good comparability. Moreover, we present novel datasets that investigate the ability of modern sequence-to-sequence systems to transfer their knowledge from one domain to a new one. Finally, we take a further step towards summarization systems suitable for everyday use by creating a new dataset showing the summaries of very long source inputs in the form of research papers.

Zusammenfassung

Automatische Textzusammenfassung, eine Aufgabe der maschinellen Übersetzung, die den Prozess der Kompression eines Textes automatisiert, während die Hauptideen des Eingabedokumentes erhalten bleiben. Bisherige Ansätze, die in diesem Bereich vorgeschlagen wurden, können in zwei grundlegende Arten unterteilt werden: extraktive und abstrakte Zusammenfassung. Während erstere Verkürzungen durch Löschen unwichtiger Teile der Eingabefolge erzeugt, extrahiert letztere die relevanten Informationen aus der Eingabefolge und erzeugt eine neue, grammatikalisch korrekte Zusammenfassung. Diese Arbeit befasst sich mit der abstraktiven textuellen Zusammenfassung. Dabei soll die Anwendbarkeit des *Transformer Models*, publiziert von Vaswani et al. [2017], in der Erstellung von abstraktiven Zusammenfassungen, geprüft werden. Für diese Experimente werden öffentlich zugängliche Datensätze aus bestehenden Forschungsarbeiten verwendet, um eine gute Vergleichbarkeit zu schaffen. Des Weiteren werden neuartige Datensätze vorgestellt, die die Fähigkeit moderner Sequenz-zu-Sequenz Systeme erforschen, ihr Wissen von einer Domäne auf eine neue zu übertragen. Einen weiteren Schritt zu alltagstauglichen Zusammenfassungssystemen wird durch die Erstellung eines neuen Datensatzes genommen, der die Zusammenfassung sehr langer Quelleingaben, in Form von Forschungsberichten, darstellt.

Contents

List of Figures

List of Tables

1 Introduction

In 2011, Hilbert and López [2011] published an astonishing study about the increasing digital memory capacity of machines created by humans. The world's technological capacity to store information grew from 2.6 exabytes (optimally compressed) in 1986 to 15.8 in 1993, over 54.5 in 2000, and to 295 exabytes in 2007. This is equivalent to less than one off-the-shelf 730-MB CD-ROM per person in 1986 (539 MB per person). In the following years this number increased dramatically, requiring roughly 4 discs per person in 1993, 12 discs in 2000, and almost 61 discs per person in 2007. Piling up the imagined summed up 404 billion CD-ROM would create a stack from the Earth to the Moon and a quarter of this distance beyond.
These numbers are backed by a recent[1] study that quantifies the amount of online text and multimedia information, human transfer online per year. It shows that we are transferring about 864 exabytes in the year 2015, raising it by nearly 40% to 1464 exabytes in 2017.
To make this amount of information processable for humans, extracting the most important facts is indispensable. We experienced this once more, while searching for this study on the internet. Understanding the content of the search results we faced, was only possible by presenting abstracts (summaries) to the reader, trying to process all the presented search results.
With the exploded quantity of online text and multimedia information in recent years, there has been a renewed interest in automatic summarization. In addition, modern users are increasingly interested in obtaining information from various sources, which requires filtering and translating information into an easily and quickly readable form. Therefore there is an increasing need for automatic systems capable of extracting the most relevant textual information and outputting it in the shortest, consistent and most informative way.
The effort to make computers understand and use information expressed in natural language is an open field of research known as natural language processing (NLP). NLP has been particularly difficult due to the often ambiguous and imprecise language utilized by humans. With achieving new state-of-the-art results in NLP, neural networks based approaches have replaced Statistical Machine Translation (SMT).

[1] Cisco Systems. n.d. Datenvolumen des globalen IP-Traffics in den Jahren 2014 bis 2017 sowie eine Prognose bis 2022 (in Exabyte pro Monat). Statista. Zugriff am 27. März 2019. Verfügbar unter https://de.statista.com/statistik/daten/studie/266869/umfrage/prognose-zum-datenvolumen-des-globalen-ip-traffics/

Considering this new perspective of new architectures, the previous field of extractive summarization, where the targets were generated by removing unimportant words from the source, receives less attention. Instead, the focus of research is now applied to abstractive summarization, where targets are novel sentences containing the important facts identified in the source.

1.1 Goals

In this thesis we explore the recent *Transformer Model* architecture presented by Vaswani et al. [2017] which sets new state-of-the-art performance in machine translation research. Therefore several experiments applying this approach in the fields of abstractive summarization were planed. The results are compared with previous approaches using prior architectures. To ensure the comparability the common datasets in summarization research should be used to evaluate the performance of the *Transformer Model*.
A problem which isn't examined by now in detail is the ability of neural models in summarization, precisely the *Transformer Model* in our work, to be able to generalize on completely different subjects compared to its original training data subject. In this thesis we present new corpora containing data collected from different research fields, to conduct novel experiments on the generalization ability.
Because of the fact that large-scale datasets for summarization of longer texts are very rare, we create novel corpus including very large source sequences to present new opportunities developing summarization models. This dataset should give researchers the possibility to explore difficulties handling large inputs.

1.2 Outline

In the next chapter 2, we present our background research on neural networks introducing the fundamentals to ensure the understanding of our conducted experiments and adoptions. After that we describe the background research in the summarization field in chapter 3 to give information to the reader on the most important characteristics in text summarization and evaluation. Furthermore we give details about related work in chapter 4 describing the recent approaches and results.on the main tasks in summarization: *Headline Generation* and *Abstract Generation*. Subsequently in chapter 5 we present the datasets used in this thesis to conduct experiments on. This includes introducing our new corpora and besides that an analytical comparison between the existing and new datasets. In chapter 6 we describe the details about our experiments to show and analyze the results in section 7 afterwards. Finally, chapter 8 concludes the thesis and explores future developments of our presented models.

2 Neural Networks Background Research

Achieving outstanding results in research topics for years, deep learning is used more and more in daily applications. New approaches in the field of machine translation or automatic speech recognition show great results compared to historically approaches. With the use of machine learning in the task of abstractive summarization, where the output contains entirely new generated output sentences, which do not necessarily exist in the source text, new state-of-the-art systems were recently published ([Rush et al., 2015]). In the following chapter background information about neural networks used in this thesis are presented.

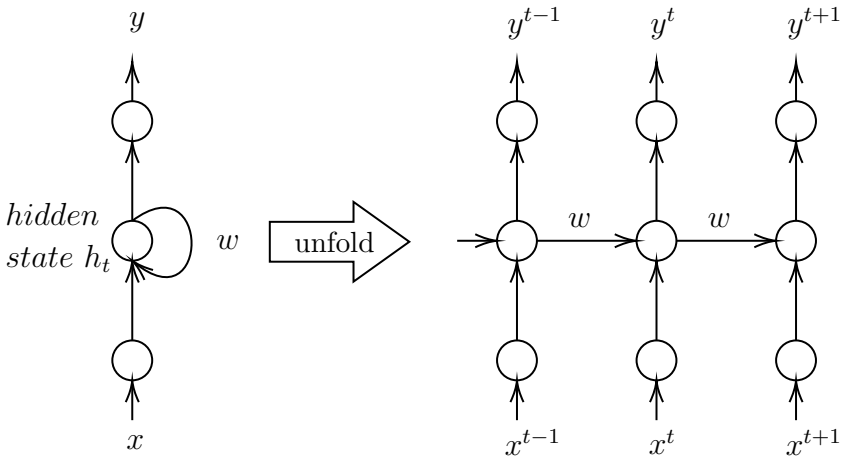
2.1 Recurrent Neural Networks

Recurrent neural networks (RNN) introduced a type of models, which are able to deal with sequential information and model their dependencies over time. Rumelhart et al. [1988a] mentioned recurrent nets first, describing them as layered feed forward networks (FFN), where each layer corresponds to one time step. Next, Elman [1990] published the first model architecture, where hidden unit patterns are fed back to themselves so the internal representation reflects the context of prior internal states. By introducing connections from one neuron to neurons of the previous layer, memory is formed over the past. Figure 2.1 *Recurrent neural networks over time* depicts the process of sharing context through weighted connections during continuous time steps. According to Elman [1990] this can be formulated as shown in following term:

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \\ y_t &= softmax(W_y h_t + b_y) \end{aligned} \tag{2.1}$$

Here, the authors proposed that the current hidden state h_t access the previous hidden state h_{t-1} to compute the output y_t. W_h, U_h, b_h and W_y are weights, which are trained by backpropagation introduced to neural networks by Rumelhart et al. [1988b]. Backpropagation uses the chain rule to compute the derivative of a loss function with respect to each trainable parameter in the network. While this approach has become standard, Jordan [1997] presented a novel architecture called *Jordan network*, where

Figure 2.1: Unfold recurrent neural networks



the following states depend on the previous outputs, not on hidden states.
The dynamics of this network across time steps can be visualized by unfolding the network (Figure 2.1). Given this figure, the model can be interpreted not as cyclic, but rather as a deep network with one layer per time step and shared weights across time steps. To train unfolded recurrent neural networks, the original backpropagation algorithm was adapted, to accumulate gradients through time steps. This algorithm is called backpropagation through time (BPTT), and was introduced by Werbos et al. [1990].
RNNs are ideal for tasks such as speech recognition and speech processing, where access to previous states plays an important role. For example, determining the semantic meaning of a word in a sentence often requires knowledge of the previous words. With RNNs, it is possible to create a model that processes phrases on a word level but includes information from preceding words in the classification decision.
A widely seen problem in using RNNs is the *vanishing gradient* problem, which describes the degeneration of the gradients, when the backpropagation algorithm runs in each time step. The gradients get smaller and smaller that by the time we get back to the beginning of the sentence, the gradient is so small that it has no significant effect on the parameters that need to be updated. One method to solve this is introduced by a new neural networks architecture, shown in the following section 2.1.1 *Long Short-Term Memory*.
Although recurrent based models became standard in handling temporal information, there are models without recurrence, such as time-delayed neural networks (TDNNs) introduced by Waibel et al. [1989].

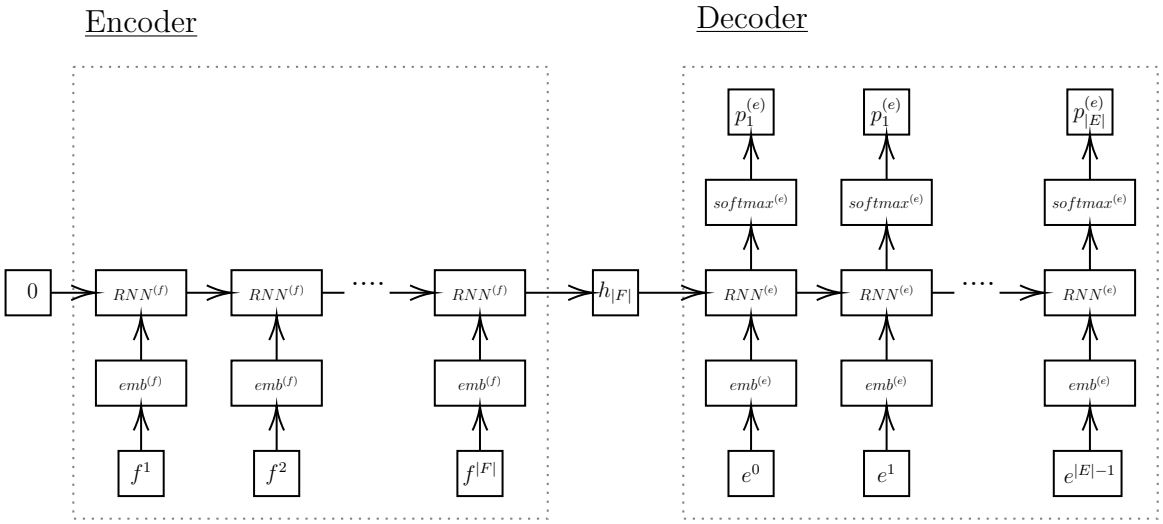
2.1.1 Long Short-Term Memory

Hochreiter and Schmidhuber [1997] introduced a novel architecture to overcome the problem of vanishing gradients and handle long-term dependencies in sequential data. The most fundamental idea behind the Long Short-Term Memory (LSTM) is that in addition to the standard hidden state h used by most neural networks, it also has a *memory cell* c, for which the gradient $\frac{dc_t}{dc_{t-1}}$ is exactly 1. Because of this special gradient, information stored in the memory cell does not significantly suffer from vanishing gradients, and thus LSTMs can capture long-distance dependencies more effectively than standard recurrent neural networks. Moreover LSTMs contains several *gates*. They control the removal or addition of information to the *memory cell* c. Gates make use of the sigmoid function, which outputs values between 0 and 1 and can be seen as a filter that restricts how much information the gate should let through.

2.2 Encoder-Decoder Models

Cho et al. [2014] present the first encoder-decoder neural model approach as novel architecture in neural machine translation, where the encoder "encodes" the source sentence F to a fixed context vector. The decoder uses this information in the context vector to "decode" the target sentence E.
Figure 2.2 *Encoder-decoder architecture* depicts this novel system architecture where the encoder is placed on the left side, containing a recurrent neural networks, expressed as $RNN^{(f)}$. Each time step calculates a new hidden state for phrase input f^t and previous hidden state h^{t-1}. Before we can use the words f^t from the input sentence E in our network, we have to create word embeddings $m_t^{(f)}$ (n-dimensional vector representations of word f^t). We visualize the generation of word embeddings $emb^{(f)}$ for each time step in figure 2.2.
Word embeddings are a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector, where words with similar meanings have similar representations. There are several different word embedding approaches, such as Gal and Ghahramani [2015] presenting *Embedding Layers* that use *one-hot* encoded words jointly learned with the neural network. In addition *Word2Vec* ([Mikolov et al., 2013]) and its extension *GloVe* presented by Pennington et al. [2014] introduce a statistical method to learn word embedding from a text corpus.

Figure 2.2: Encoder-decoder architecure



Mathematically, RNN-based encoder-decoder architectures can be formalized as follows.

$$
\begin{aligned}
m_t^{(f)} &= emb^{(f)}(f_t) \\
h_t^{(f)} &= \begin{cases} RNN^{(f)}(m_t^{(f)}, h_{t-1}^{(f)}) & t \geq 1, \\ 0 & \text{otherwise} \end{cases} \\
m_t^{(e)} &= emb^{(e)}(e_{t-1}) \\
h_t^{(e)} &= \begin{cases} RNN^{(e)}(m_t^{(e)}, h_{t-1}^{(e)}) & t \geq 1, \\ h_{|F|}^{(f)} & \text{otherwise} \end{cases} \\
p_t^{(e)} &= \text{softmax}(W_{hs} h_t^{(e)} + b_s)
\end{aligned}
\tag{2.2}
$$

Here in the first two lines, we create the word embedding for our phrase f_t and encoder hidden state $h_t^{(f)}$ for each word in time step t. In the first time step 0, the hidden state is set to an empty vector $h_0^{(f)} = 0$. This hidden state encodes the information from our source sentence E. Lines three to five show the decoder side, which creates word embeddings for the last output. Computing hidden states for decoder and calculating word probabilities over the target vocabulary using $p_t^{(e)} = \text{softmax}(W_{hs} h_t^{(e)} + b_s)$. $RNN^{(f)}$ and $RNN^{(e)}$ can be either *RNN* or *LSTM*. Until now, all the architectures are based on recurrent neural networks. In the upcoming section 2.4 *Transformer Model* describes new encoder-decoder architecture which doesn't use RNNs. In contrast, it is entirely based on attention, which we a describe in the next section 2.3 *Attention*.

2.3 Attention Mechanism

A general problem with the previously described encoder-decoder models is the aspect that it attempts to store information about sentences of any arbitrary length in a hidden vector of fixed size. In other words, even if our translation system takes input of lengths between 10 and 100 words, it always uses the same fixed context vector. This leads to the problem that if the network is too small it cannot handle long sentences. On the other hand, if we increase number of parameters in the network it consumes more resources.
Bahdanau et al. [2014] first introduced an attention mechanism in their encoder-decoder model. The basic idea of attention is to encode longer input sequences in larger representations, compared to smaller inputs. Instead of attempting to convert the entire input sequence into one vector representation (fixed context vector), they now produce a single vector representation for each input word. This leads the system to create more representations for longer input sequences and fewer for smaller ones, avoiding inefficient representations. By referring to these vectors in the decoding step, the neural network creates a mapping between input and output words through the attention matrix.
The original attention mechanism is applied to bidirectional RNNs ([Bahdanau et al., 2014]), where two different encoders are used. One is traveling forward and one is traveling backward over the input sequence. Introducing the additional reverse encoder ([Sutskever et al., 2014]) improves the problem of long-distance dependencies between words at the beginning and end of the sequence.

$$\begin{aligned}
\overrightarrow{h}_j^{(f)} &= \begin{cases} \overrightarrow{\mathrm{RNN}}(emb(f_j), \overrightarrow{h}_{j-1}^{(f)}) & j \geq 1, \\ 0 & \text{otherwise} \end{cases} \\
\overleftarrow{h}_j^{(f)} &= \begin{cases} \overleftarrow{\mathrm{RNN}}^{(f)}(emb(f_j), \overleftarrow{h}_{j+1}^{(f)}) & j \leq |F|, \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{2.3}$$

Equation 2.3 shows the computation of hidden states for every word running forward and backward over the source sentence using RNNs. Here, $\overrightarrow{h}_{j-1}^{(f)}$ is the hidden state of the word on the left side and $\overleftarrow{h}_{j+1}^{(f)}$ shows the hidden state of the word on the right side. In the following equation 2.4 both vectors $\overrightarrow{h}_j^{(f)}$ and $\overleftarrow{h}_j^{(f)}$ are concatenated to obtain a bidirectional representation of one word. Further, each representation of a word f_j is concatenated to a matrix $H^{(f)}$, representing the entire input sequence (Equation 2.5). Each column represents a word in our input sequence.

$$h_j^{(f)} = [\overleftarrow{h}_j^{(f)}; \overrightarrow{h}_j^{(f)}]. \tag{2.4}$$

$$H^{(f)} = \mathrm{concat\_col}(h_1^{(f)}, \ldots, h_{|F|}^{(f)}). \tag{2.5}$$

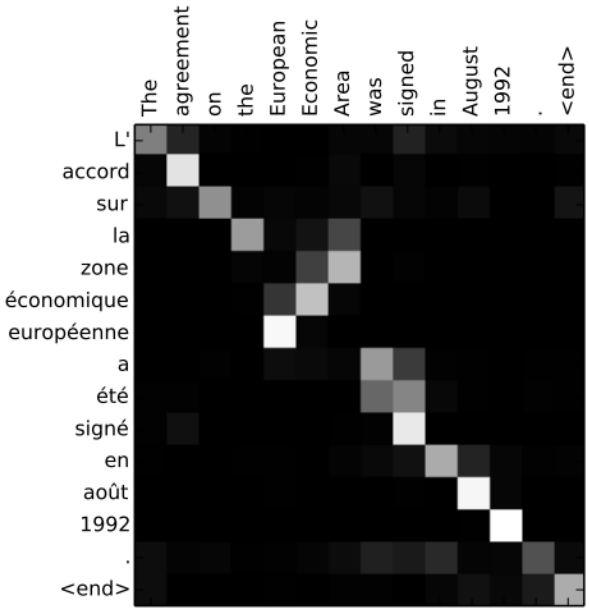


Figure 2.3: Attention weight for correspondence between source and target words ([Bahdanau et al., 2014])

Because we now have a matrix representing our source sequence, we cannot apply traditional $softmax(...)$, which takes a vector as input. One key innovation of the attention mechanism is introducing α_t as *attention vector* which is used to compute vector representation of $H^{(f)}$ given a time step t. This attention vector controls how much we focus on a specific input $h_j^{(f)}$ when predicting the next word in our output sequence. In other words it contains weights, controlling influence of inputs for our next output. Figure 7.4 depicts attention weights for correspondence between source and target word. For example, the word *août* is highly connected to the word *August* showing by higher attention weight (light pixels in matrix).

When applied to RNN encoder-decoder systems, attention mechanisms can be implemented by calculating an attention score $a_{t,j}$ as shown in Eq. 2.6. Mathematically, given the current hidden states of the encoder and the decoder at time step t, the attention score can be calculated originally by using Multilayer Perceptrons ([Bahdanau et al., 2014]).

$$a_{t,j} = \text{attn\_score}(h_j^{(f)}, h_t^{(e)}). \tag{2.6}$$

Moreover, the authors propose a simple dot product between the hidden states, as it does not add additional parameters to the network. After calculation and normalizing how the attention score $a_{t,j}$, a context vector c_t is computed, shown in equation 2.7.

Further, similar to previous RNNs without attention, *softmax* is used to calculate probability of each output word (Eq. 2.8).

$$c_t = H^{(f)}\alpha_t. \tag{2.7}$$

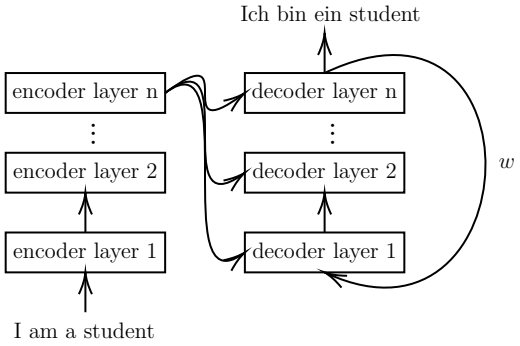
$$p_t^{(e)} = \text{softmax}(W_{hs}[h_t^{(e)}; c_t] + b_s). \tag{2.8}$$

We introduced the attention mechanism in detail, because in the upcoming section 2.4 a novel architecture using a completely new attention mechanism , which replaces the demand of RNNs by fully focusing on attention, is described.

2.4 Transformer Model

Vaswani et al. [2017] presented a new encoder-decoder architecture which introduces a completely new design of handling sequential data. By that time, all state-of-the-art approaches used recurrent or convolutional neural networks to model dependencies over time (cp. section 2.1). In contrast to previous approaches, they entirely use attention to handle the sequence-to-sequence task. Their main goal is to reduce sequential computations, which are needed to compute hidden representations for words or other phrases in our sequence. Because of dependencies of further states in regard to previous states, parallel computing is slowed down and the number of operations increases depending on the distance between input and output words. In addition *Transformer Model* allows the stacking of layers similar to modern convolutional layer approaches. Building deeper networks makes it possible to represent dependencies of close inputs in early layers while distant inputs represent their dependencies in deeper layers.
After introducing the attention mechanism for encoder-decoder systems it has became an integral part of sequence models to handle dependencies without regard to the distance between source and targets. In their work Vaswani et al. [2017] show how to replace recurrent approaches, like LSTMs, entirely with attention to improve parallelization significantly. Doing so, they present new attention spin-off called multi-head attention, which we will discuss in detail in section 2.4.1.2.

Figure 2.4: Abstract Transformer Model Architecture

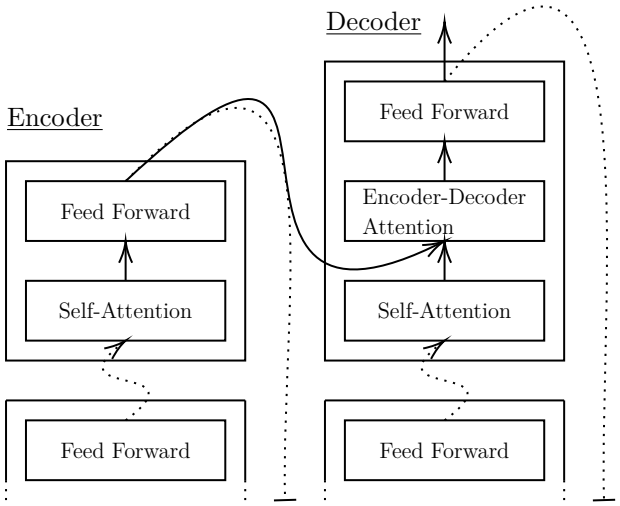


In figure 2.4 *Abstract Transformer Model Architecture* we present a conceptual overview of the *Transformer Model* architecture showing stacked encoder layers on the left side and decoder layers on the right side. We begin our introduction on each layers, combine them and present the whole structure at last. Each layer uses combined to multi-head attention which we describe in section 2.4.1.2.

2.4.1 Self-Attention

Before introducing *Transformer Model*, encoder-decoder models based on RNNs handled their context while decoding new words entirely different. By using previous hidden states weighted with attention they generated new words depending on their context from the source sequence. Now, self-attention is the method Transformer uses to connect relevant words into the new words, the model is currently decoding.
For introductory purposes, we first present self-attention (also called Scaled Dot-Product Attention) based on vectors before moving to the generalized approach using matrices that allow for increased parallelization. Having each stacked encoder and decoder layer based on an identical layout, figure 2.5 depicts self-attention as a key component of the *Transformer Model*.
Self-attention is computed for each input word (or character, if the model is character based), after it is converted into word embedding representation. The main components are three vectors named *query* ($q \in \mathbb{R}^{d_k}$), *key* ($k \in \mathbb{R}^{d_k}$) and *value* ($v \in \mathbb{R}^{d_v}$). They are based on the input represented as word embedding ($x \in \mathbb{R}^{d_{model}}$). Generally, dimensions d_k and d_v are smaller than embedding size d_{model} to make self-attention computationally efficient. The query can be seen as hidden state of decoder, key as the hidden state of encoder and value as the normalized weight determining the importancy that is put on the attention of the key. As figure 2.6 shows, *query* q_i is created from word embedding x_i by multiplying it with a trainable weight matrix W_q (Equation 2.9a). The same procedure is done to create *key* k_i and *value* v_i.

Figure 2.5: Transformer Layers in Detail



Weight matrices in self-attention learn features in sequences, comparable to kernels in convolutional layers learning features from multidimensional inputs.

$$\begin{aligned} q_i &= W_q x_i \\ k_i &= W_k x_i \\ v_i &= W_v x_i \end{aligned} \tag{2.9a}$$

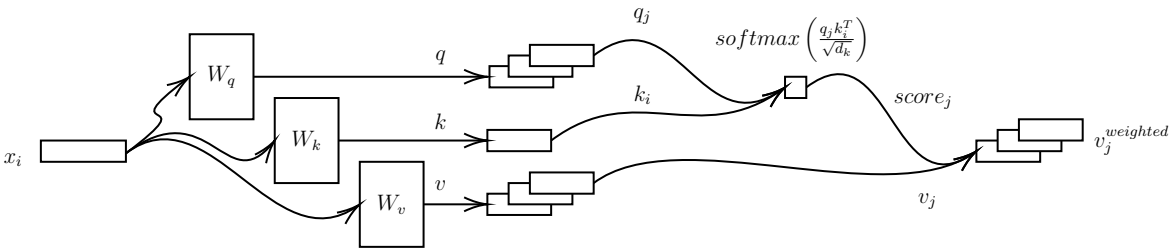
$$score_j = \frac{q_i k_j^T}{\sqrt{d_k}} \tag{2.9b}$$

$$v_j^{weighted} = softmax(score_j) v_j \tag{2.9c}$$

$$\text{Self-Attention}(Q, K, V) = \frac{softmax(Q * K^T)}{\sqrt{d_k}} V \tag{2.9d}$$

In equation 2.9b we calculate scores by dot product for each current query q_i with respect to all keys from the input sequence k_j. This score determines, how much focus should be placed on words k_j while encoding q_i. In addition we divide the score by $\sqrt{d_k}$ to have more stable gradients. Moreover equation 2.9c we use the score as weight

Figure 2.6: Computation of Self-Attention



for each value v_j to compute it's dependency for q_i (decoder state). Finally we provide the self-attention computation in matrix form in equation 2.9d, which is used for implementation to improve parallelization. Matrices Q, K and V represent batches of sequences which are processed in one computation step by applying computationally efficient matrix multiplication.

2.4.1.1 Position-wise FFN

As shown in figure 2.5 each layer in encoder and decoder also consists of fully connected layers including max-pooling and *ReLU* ([Glorot et al., 2011]) activation function. The inner layer is controlled by dimension d_{ff}. It projects the weighted values $v_j^{weighted}$ to the dimension d_{ff}. This is followed by applying the *RELU* activation function. Afterwards it's projected back to d_{model} dimension, described by equation 2.10. This output is used as input for the next layer on the stack.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.10}$$

2.4.1.2 Multi-Head Attention

One problem of self-attention in *Transformer Model* is the reduced resolution due to averaging attention-weighted positions ([Vaswani et al., 2017]). To increase resolution, the authors propose the multi-head attention which performs the self-attention h-times, representing the amount of heads in the model. This affects equation 2.9d as shown in equation 2.11.

$$\begin{aligned} head_i &= \text{Self-Attention}(QW_{Qi}, KW_{Ki}, VW_{Vi}) \\ \text{MultiHeadAttention}(Q, K, V) &= Concat(head_1, ..., head_h)W^O \end{aligned} \tag{2.11}$$

By calculating each head in parallel this further increases the performance. In addition heads can focus on different positions to expand the models ability to create dependencies between target and source words. By concatenation of all heads, we

receive a weighted values matrix, which is projected onto output dimension d_{model} by applying linear transformation with weight matrix W^O.

2.4.2 Encoder-Decoder Architecture

Finally we present the complete transformer model architecture in figure 2.7 *The Transformer - Model Architecture.* Either encoder and decoder side can be stacked Nx times using the layer module (figure 2.5). Vaswani et al. [2017] recommend to do this 6 times on both sides.
After embedding on encoder input is applied, authors introduce a novel approach to represent the position of token in sequence called *Positional Encoding.* Moreover we provide more information in section 2.4.2.1. In addition to self-attention modules and position-wise feed forward networks, the authors include residual connections, to allow gradients to flow through the network without applying them to non-linear activation functions. Vaswani et al. [2017] propose this to reduce the vanishing or exploding gradients in deep neural networks.

2.4.2.1 Positional Encoding

In RNNs the position of tokens is encoded by processing them one at a time. In the *Transformer Model* the authors purpose to encode time as a trigonometric function (e.g. *sine* wave). This vector is added to all inputs and outputs (shown in figure 2.7) to represent the position within the sequence. The advantage of using formulas 2.12 given by the authors is being able to scale to unseen lengths of sequences.

$$\begin{aligned} PE(pos, 2_i) &= sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \\ PE(pos, 2_i+) &= cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \end{aligned} \tag{2.12}$$

2.4.2.2 Generating Outputs

The final layers on decoder side turn the stack of float vectors into words. The models follow previous research and calculate the softmax to obtain a probability distribution over the output words. Furthermore the highest predictions are used in beam search ([Reddy et al., 1977]). Before applying softmax a linear layer adjusts the embedding size from internal dimension d_{model} to the target vocabulary size.

2.5 Byte Pair Encoding

A drawback of state-of-the-art NMT systems is that huge vocabularies of natural languages often exceed the memory of graphic processing units (GPU). Therefore the

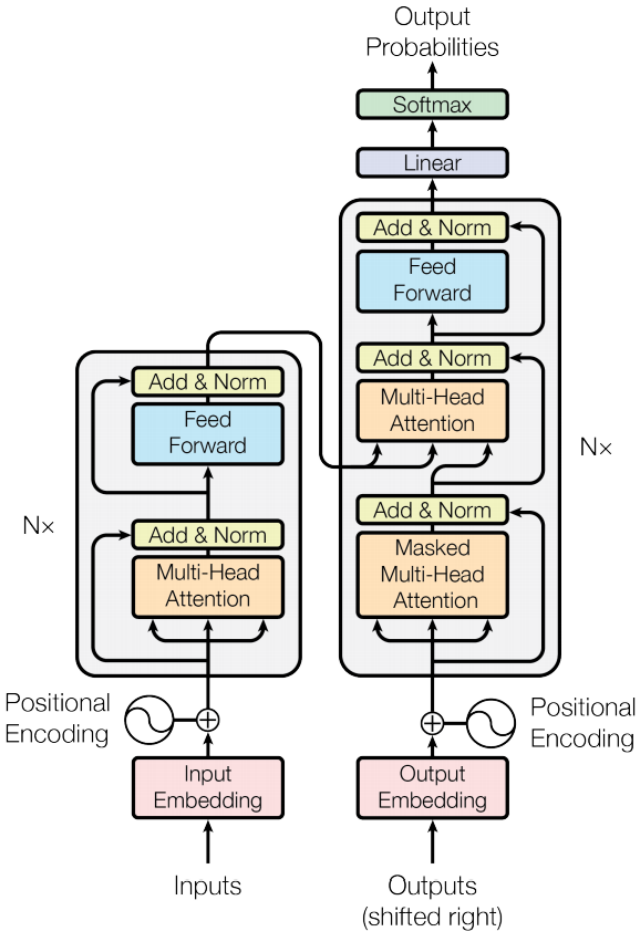


Figure 2.7: The Transformer - model architecture ([Vaswani et al., 2017])

vocabulary size must be reduced. Splitting words into sub-word units is a way to do so.

Byte Pair Encoding (BPE) was first introduced by Sennrich et al. [2015] presenting a new approach to segment text into subword units. This approach addresses the handling of out-of-vocabulary words by encoding rare and unknown words as sequences of subword units. One advantage is that for morphologically rich languages, morphological variants of words can be generated, that do not occur in the training data. They define models containing the merging operations specified by hyperparameter and learned from joined training corpora. This is important, because it ensures that names or rare words are separated in the same learned way for source and target corpora. In applying BPE on training data, based on learned merge models, the size of splits which is defined by n-grams is also set by hyperparameter.

We apply BPE in preprocessing for all used datasets listed in chapter 5. Hyperparameters for the amount of merge operations are specified in the Experiments section 7.1.

3 Summarization Background Research

In this chapter, we present a classification of different summarization systems along with our background research in the field of summarization research.

3.1 Characterization of Summarizations

According to Jones [1998] and Dong [2018], the task of text summarization can be classified by the following factors: input, purpose and output. In this thesis we especially concentrate on single-document including monolingual text on source and target side. Our systems generate generic abstract summarizations. In one of our experiments (6.2), we investigate if our system can be used in general purpose (summarizing different subjects) or only domain-specific scenarios.

3.1.1 Input Factors

3.1.1.1 Single-document vs. multi-document

These factors describe the document input of the summarization system, which distinguishes between generating a summarization based on a single document or multiple documents. The task of entering multiple documents creates the problem of concatenating the inputs in an appropriate way to the system.

3.1.1.2 Monolingual, multilingual vs cross-lingual

Separating the task of summarization according to the language pairs in the processing, the system can be classified into three groups. In a monolingual system, the input language of the source document and the output language of the summarization stay the same. Besides this, the system can only handle one language pair. This fact differentiates the first type from multilingual systems, where the system can handle multiple language pairs at the same time. Nevertheless the generated summary has the same language as its input text. In contrast a cross-lingual system distinguishes from the types by the fact that the generated summary can be generated in different languages. In other words it also adds the task of translating to the system.

3.1.2 Purpose Factors

3.1.2.1 Generic vs. user-oriented

The key difference between generic and user-oriented systems is the fact, that generic summarizations need to be suitable for all types of readers, not including their special objectives or interests. Its counterpart is called a user-oriented system, which produces personalized outputs including certain information from the source text, which is indicated as most interesting for the special readers. This fact makes it mandatory to know the interests of a user and add this information as additional input to the generation of the summarization.

3.1.2.2 General purpose vs domain-specific

General purpose systems are able to create summarizations across any domain of the input with no or just small modifications. In contrast, the domain-specific systems only cover a small range of inputs for which the system is specifically created for.

3.1.3 Output Factors

3.1.3.1 Extractive vs abstractive

The summarization of the input can be created using two different approaches, defining how input information is transformed to an output. The first and original approach is called extractive summarization, which selects parts of the input. This approach often reduces the readability of the summary. On the other hand is the abstractive summarization, which generates new outputs. This includes selecting the important source information and constructing a novel sentence. In current research extractive and abstractive summarization are becoming more and more mixed up See et al. [2017] including a copy mechanism to directly add important content from the source to the new output.

3.2 TF-IDF Sentence Ranking

In this section, we introduce the task of information retrieval for summarization, which has the purpose of finding sentences containing more important information compared to others. In other words, we discard entire sentences which do not include information to be transfered to our system output, the summarization of the source sequence. Salton and Michael [1983] were the first to present the numerical statistic *term frequency–inverse document frequency* (if-idf) reflecting the importance of words in a document.
The *TF-IDF* computes a product of term frequency and inverse document frequency.

Term frequency $TF(d,t)$ counts the number of times a specific term t occurs in a document d. Furthermore the inverse document frequency $IDF(t)$ measures how important a term is by computing its rareness.

$$\begin{aligned} \text{TF}(d,t) &= f_{t,d} \\ \text{IDF}(t) &= \log \frac{N}{DF(t)} \\ \text{TF-IDF}(d,t) &= TF(d,t) * IDF(t) \end{aligned} \tag{3.1}$$

Compared to the original approach, we do not measure the importance of a word by its rareness. In contrast, we assume that a sentence is more important for our target summarization, if the words of a sentence are used more often in the source sequence. In other words, if a topic (words) is more often part of the source or discussed more often, is should get more importance in our goal summarization.
In order to find the most important sentences, we sort all sentences descending by their TF-IDF score. After sorting, we sum up the amount of words of each sentence, stopping after exceeding a threshold defining the maximum number of words a source sequences is allowed to have.

3.3 Evaluation Metrics

We apply ROUGE Score evaluation to measure the quality of our output summarizations. Since Mani [2001] and Lin [2004] published their work, *ROUGE-1*, *ROUGE-L* and *ROUGE-2* (which were published as ROUGE-N) have become the standard quality metrics in field of summarization. We introduce and discuss the basic ROUGE metrics in 3.3.1 *Standard ROUGE*. Furthermore there are several additional metrics, which measure different qualities of summaries. For example whether the summary contains words from source sequence or percentage of novel words in summaries. We will present these metrics in section 3.3.2 *Further Metrics*.

3.3.1 Standard ROUGE Metrics

ROUGE, which stands for *Recall-Oriented Understudy for Gisting Evaluation*, defines metrics, to determine the quality of generated summaries comparing to (human) gold summaries. By calculating the overlapping units such as n-grams, word sequences and word pairs between source and target corpus.
Despite the usually used ROUGE-N and ROUGE-L measure, Lin [2004] also published ROUGE-W and ROUGE-S Scores. ROUGE-1 and ROUGE-2, which are special types of ROUGE-N, measure the n-gram co-occurrence between the candidate and reference summary defined in equation 3.2. Thereby, n stands for the length of the n-gram,

$gram_n$, and $Count_{match}(gram_n)$ is the maximum number of n-grams co-occurring in a candidate summary. $S(ReferenceSummaries)$ is a set of reference summaries.

$$\begin{aligned} \text{ROUGE-N}_{recall} &= \frac{\sum S(ReferenceSummaries) \sum gram_n Count_{match}(gram_n)}{\sum S(ReferenceSummaries) \sum gram_n Count(gram_n)} \\ \text{ROUGE-N}_{precision} &= \frac{\sum S(ReferenceSummaries) \sum gram_n Count_{match}(gram_n)}{\sum gram_n Count(gram_n)} \end{aligned} \tag{3.2}$$

Clearly, the ROUGE-N metric is based on recall (true positive rate/ hit rate) to compare the n-grams from candidate and reference summary.
In contrast, ROUGE-L compares the longest common sequences (LCS). For a given sequence $X = [x_1, x_2, ..., x_m]$ and subsequence $Z = [z_1, z_2, ..., z_n]$, ROUGE-L increases depending on a strictly increasing sequence of indices $[i_1, i_2, ..., i_k]$ when $x_i = z_i$ (sequence X equals subsequence Z).

$$\begin{aligned} \text{ROUGE-L}_{recall} &= \frac{LCS(S,G)}{length(S)} \\ \text{ROUGE-L}_{precision} &= \frac{LCS(S,G)}{length(G)} \end{aligned} \tag{3.3}$$

ROUGE-W defines an extension of ROUGE-L as *Weighted Longest Common Subsequences*. It enhances the previously introduced ROUGE-L score by weighting consecutive matches higher. The weight depends on the length of the current consecutive matches. Usually ROUGE-W is not used in the latest evaluations of summarization models.
ROUGE-S measures Skip-Bigram Co-Occurrence Statistics, which describes the matches of arbitrary bigrams in two sequences. Because of low significance in longer sequences, this metric is usually not used either.
As we already introduced in equation 3.2 and 3.3 each metric is defined by recall *recall* (R) and *precision* (P) value. The commonly used *ROUGE* evaluation metric is defined by its balanced F_1 score shown in equation 3.4. In the following results chapter 7 we evaluate our experiment outcomes using this score.

$$F_1 = 2\frac{PR}{P+R} \tag{3.4}$$

3.3.2 Further Metrics

Evaluating our models with the percentage of copied words from source sequence to target summaries gives information about the abstraction level our models reach. This is an important feature of abstractive models compared to extractive approaches.

Recently additional metrics have been published to measure more qualities in the complex task of comparing summarizations.
Kryscinski et al. [2018] define the metric *novel n-grams* measuring new words generated by the model, which were not part of the source sequence. This advanced metric also rates the abstraction level of our models.

3.3.3 Fact Aware Neural Abstractive Summarization

A main duty of generating summaries of documents is the preservation of facts. Changing the truth content, while processing sources, makes the summary worthless. By discarding a word, for example negations like *not* or *no*, the statement of facts and entire sentences changes. Cao et al. [2018] discuss this problem in their paper by running a study on summaries created on the *Gigaword* dataset. They compare several models (we present more details about these models in section 4.2), presented over the past years on their faithfulness. Unfortunately the authors do not propose a metric to automatically evaluate summarization systems to keep facts unchanged. They manually set *FAITHFUL*, *FAKE* or *UNCLEAR* flags on a small test set.
In the upcoming chapter we introduce the system of Cao et al. [2018] and other researchers which published novel architectures in the summarization research field in the past years.

4 Related Work

The active research topics in the field of abstractive summarization do concentrate on two main tasks, headline and abstract generation.
On the abstractive headline generation task Rush et al. [2015] created the first attempt using different encoder-decoder approaches with an attention mechanism. Achieving favorable results on this task during the past years, people where looking for more challenging competitions. Increasing the input size leads to the task of summarizing whole text snippets or articles to abstracts. We will discuss recent approaches published in both fields in this chapter.

4.1 Historical Methodology

The field of automatic summarization research started with an extraction-based summarization task. Its goal is to automatically extract objects from an entire source sequence and process these objects without modifying them. An Example of this task is *keyphrase extraction*, where the goal is to select individual words and tag a document with these key words. Furthermore the task of *document summarization* has received a large amount of attention, with the goal of selecting whole sentences without modifying them and create abstracts of documents. Pioneering work done by Luhn et al. [1959] and Edmundson [1969] showed novel approaches to extract sentences used to classify the entire document. For example, the sentence after section headings like *Introduction* or *Conclusion* was used to identify the document's topic. Furthermore they showed, that sentences including words like *significant* or *hardly* also signal topic sentences. Hovy et al. [1999] published their work about creating a novel text summarization system called *SUMMARIST*. They proposed a pipeline including the steps *Identification*, *Interpretation* and *Generation*. The *Identification* step is used to extract the source (sub-)topics. Then *Interpretation* tags the sentences with identified topics and decides which are the most important with regard to the desired abstract length. Finally, *Generation* is used to formulate sentences, using sentence generating systems shown in Bateman et al. [1991].
Despite concentrating on neural based approaches, several technologies from extraction-based summarization are used in preprocessing steps of current systems. In section 3.2 we already presented the historic approach *TF-IDF* (see section 3.2) which is still frequently used in modern systems. Moreover unsupervised concepts like *LexRank*

([Erkan and Radev, 2004]) and *TextRank* ([Mihalcea and Tarau, 2004]) may be frequently used again for preprocessing in the future.

4.2 Headline Generation

Presenting the first neural based abstractive summarization approach, Rush et al. [2015] concentrated their research on single sentence generation from news articles based on the *Gigaword* dataset. They use different encoder approaches (*Bag-of-Words*, *Convolutional* and *Attention-based*) to capture the meaning representation of the source. In other words it transforms a sequence of word embeddings $\mathbf{w}_1, \ldots, \mathbf{w}_T$ to a vector $\mathbf{d}$, which is used as the meaning representation of the input. Afterwards the decoder is used to generate a summary based on these representations. They achieved state-of-the-art performance on DUC-2004 using their attention-based encoder combined with attentional feed-forward neural network on the decoder side. We present their results in section 7.5.1 as baseline called **ABS** for publishing the first scores applying neural encoder-decoder model as we do for the *Transformer Model*.
Their approach has been improved with recurrent decoders ([Chopra et al., 2016]) and hierarchical networks ([Nallapati et al., 2016]) presenting the best results on the *DUC-2004* and the *Gigaword* datasets.
The CNN-based attentive encoder used in Chopra et al. [2016] is similar to the attentive encoder proposed by [Rush et al., 2015], except that the attention weights are computed, based on the aggregated vectors obtained by a CNN model. In addition, they replace the previous decoder with a recurrent neural network. The authors propose two decoder models based on the Elman RNN ([Elman, 1991]) and the LSTMs. The resulting models are called **RAS-LSTM** and **RAS-Elman** and will be included in our evaluation in 7.5.1.
Nallapati et al. [2016] published further improvements on the encoder side replacing it with a feature-rich hierarchical attentive encoder based on bidirectional-GRUs to create a meaning representation of the source. The additional features are concatenated with the word embedding input vector and contain linguistic features, such as term-frequency (TF) and inverse document frequency (IDF) of the word. On the decoder side, the authors propose uni-directional GRUs including a large vocabulary trick to reduce the softmax computation time. It limits the number of words the decoder can generate during training by defining a small vocabulary for each mini-batch. It contains a subset of words from the global target dictionary. In addition they use a pointer network, which directly copies words from the source. This improves the summary quality by including the rare words from the sources. Their approach is called **HA-RNNs** (Hierarchical Attentive RNNs) and is included in the evaluation of this thesis in chapter 7.5.1. Furthermore to the headline generation task, the authors published the first results for the abstract generation task using the *CNN/ DailyMail*

dataset. We will analyze their system in the following section 4.3.
The latest results were published by Cao et al. [2018] together with studiying the faithfulness of summaries. In their approach, the authors propose a dual-attention network to combine two separate encoders. Each encoder uses separate attention mechanisms on the input to produce independent context vectors. The first encoder is used to encode the entire source sequence comparable to Cho et al. [2014]. In addition they propose to use a second encoder to include the facts of the first source sentence. They extract the facts using *Open Information Extraction* (OpenIE) ([Banko et al., 2007]). The facts are concatenated and used as input to the second encoder to produce another context vector. Afterwards they combine both context vectors using feed-forward networks to define the final representation of the source. On the decoder side they use the popular GRU based architecture including an attention mechanism ([Bahdanau et al., 2014]). Their best model $\mathbf{FTSum_g}$ sets a new benchmark in evaluating on *Gigaword* dataset results.

4.3 Abstract Generation

Combined in one paper, Nallapati et al. [2016] published new state-of-the-art results on the *Gigaword* dataset together with presenting a very rare large-scale new corpus *CNN/ Daily Mail* (5.3) for multiple sentence summarization with the goal to create abstracts for entire news articles. Thereby, the authors opened a new research area by presenting a new model **HA-RNNs**, which is specialized to process longer inputs (see section 4.2 for a description of this approach).
See et al. [2017] presented a new model for the abstract generation task called **Pointer-Generator Network**. Their encoder completely resembles the architecture of Chopra et al. [2016]. On the decoder side, they used a single-layer uni-directional LSTM. In addition, a decoder/pointer switch that is similar to [Nallapati et al., 2016] is used for copying source tokens directly to the target summarization. Moreover the authors investigate the problem Nallapati et al. [2016] already described. They observed that the models produce repetitions when generating multi sentences outputs. Because of this, they propose a coverage mechanism penalizing repeated attentions on already attended words. They maintain a coverage vector c^t, which contains the sum of attention over all previous decoder timesteps.

$$\begin{aligned} c^t &= \sum_{t'=0}^{t-1} a^{t'} \\ e_i^t &= v^T \tanh(W_h h_i + W_s s_t + w_c c_i^t + b_{\text{attn}}) \end{aligned} \tag{4.1}$$

In equation 4.1 c^t represents the scope, which words in the source documents have received by the attention mechanism. The coverage vector is included as extra input to the attention mechanism calculating e_i^t. It is combined with a learnable parameter

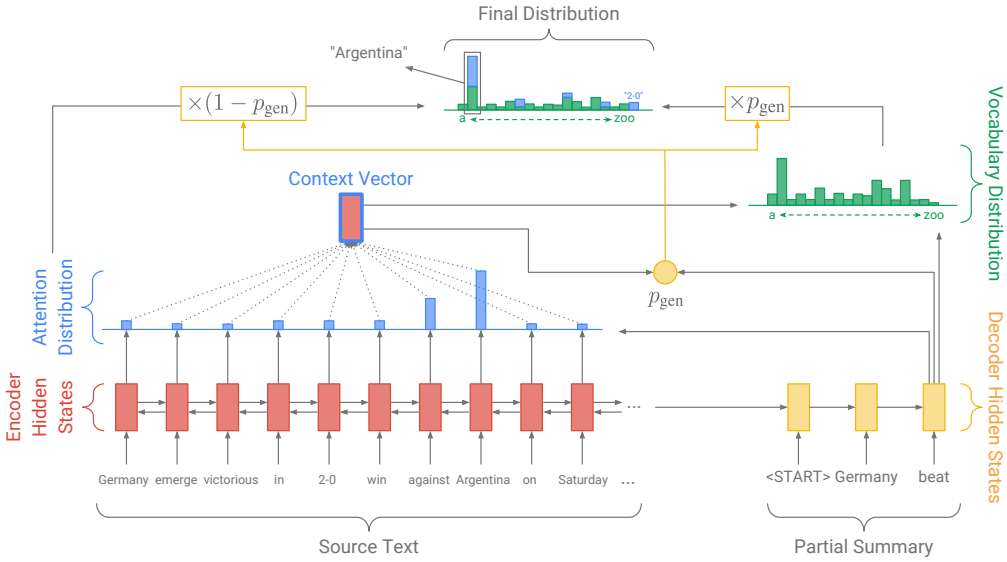


Figure 4.1: Pointer-Generator Network ([See et al., 2017])

w_c to automatically adjust its influence. Including the history of decisions made by attention mechanism it helps to reduce repetitions in current decision (choosing where to attend next).

Based on previous approach Paulus et al. [2017] present a new attention score calculation called *intra-attention mechanism* on the encoder side. In their **Neural Intra-attention model**, they combine hidden states h_t^e from the source embeddings $x = \{x_1, x_2, \ldots, x_n\}$ together with already generated outputs $y = \{y_1, y_2, \ldots, y_{n'}\}$ (summary) to produce h_t^d using bi-directional LSTMs. Both, source and target hidden states are joined using a weighted dot-product function $f(h_t^d, h_i^e) = h_t^{dT} W_{\text{attn}}^e h_i^e$ producing $e_{ti} = f(h_t^d, h_i^e)$ which is then used to compute the attention score (see chapter 2.3, Equation 2.6). The basic building block used on the decoder side is similar to See et al. [2017], where intra-attentions used an extra input to prevent the system generating from already produced outputs in the target summary.

4.4 Using Transformer Model in Summarization

Presenting an entirely new architecture for sequence-to-sequence translation tasks, Vaswani et al. [2017] released new state-of-the-art results for machine translation on various language pairs. Presenting the possibility to stack layers in the encoder and decoder to build deep models is a novel way to work with languages. Therefore it becomes also very interesting in the task of summarization.

Liu et al. [2018] published the first work in summarization research using the *Trans-*

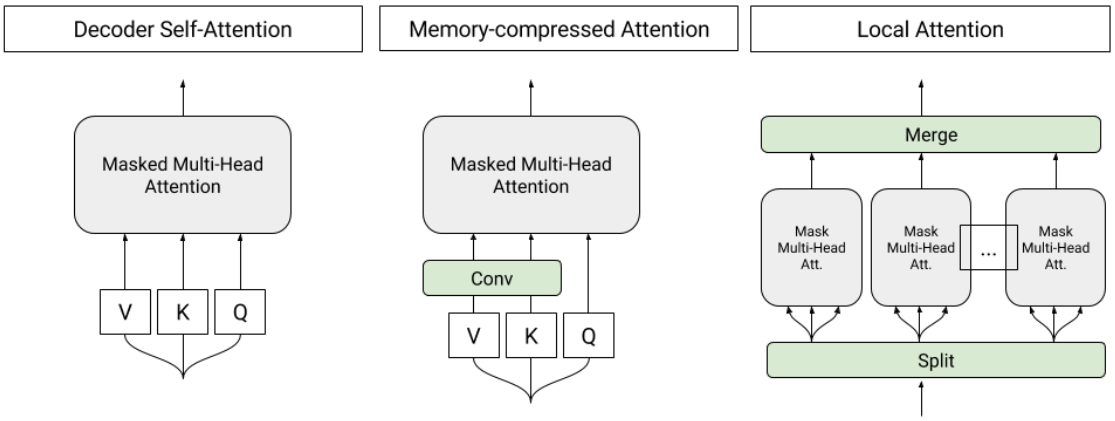


Figure 4.2: Local-attention and Memory-compressed-attention ([Liu et al., 2018])

former Model. They investigate the feasibility of generating long sequences (summaries) of various input sources. In more detail, they show how to generate Wikipedia[1] articles based on combined corpora from the articles citations and web search results. Publishing a new dataset containing more than 2.3M examples up to 10^5 source words. As a result of using long sequence inputs, they propose a novel way to compute Self-attention in the *Transformer Model*, saving as much memory as possible called *Local-attention* (L-Layer) in combination with *Memory-compressed attention* (M-Layer). Figure 4.1 depicts the fundamental principle of the novel attention implementations. The main idea behind the new calculation of attention is to split the input into blocks of fixed size, reducing the memory allocation. Furthermore they reduce the number of keys and values (See equation 2.9a in section 2.4.1) by using strided convolutions in *Memory-compressed attention*. They are stacking layers alternating between the two attention types resulting in a 5 layer architecture called *LMLML*.

[1] wikipedia.org/

5 Datasets

In the following sections we present the datasets used in our experiments. The first two corpora *Gigaword*[1] (5.1, [Graff, 2003]) and *CNN/ DailyMail*[2] (5.3,[Hermann et al., 2015]) are publicly accessible. They are used to compare the experiments results with existing approaches for headline- and abstract-generation tasks. Furthermore we provide new datasets based on research papers for both tasks. In the following we present statistics generated on the different corpora, showing difference and similarities between the publically available datasets and our own.

5.1 Gigaword

The *Gigaword* dataset was originally published by the Linguistic Data Consortium (LDC), based on several news paper corpora. It provides source and target files for training and validation. It contains around 3.8 million news articles sourced from various united states and international news services over the last two decades. As targets, the titles of news articles are used. For evaluation Rush et al. [2015] extracted a test set of 2000 sentences including headlines from the source corpus, which we will also use to evaluate our models.
A general preprocessing is already performed on the original corpus (e.g tokenization, substitution of numbers, etc.). We also apply our own preprocessing to the corpus, removing special characters.
Information about data distribution is shown in section 5.2.4 *Corpora Comparison*. In addition to the *Gigaword* corpus, DUC-200x[3] provides standardized test data without connections to our training data.

5.1.1 DUC-2003 and DUC-2004 Evaluation Datasets

The standard sentence summarization evaluation set is associated with the *DUC-2003* and *DUC-2004* tasks published by Over et al. [2007]. The data for each task consists of 624 and 500 news articles respectively from the New York Times and Associated Press Wire services. Each is paired with 4 different human-generated reference summaries, capped at 75 bytes. This dataset is for evaluation only.

[1] github.com/harvardnlp/sent-summary
[2] github.com/JafferWilson/Process-Data-of-CNN-DailyMail
[3] duc.nist.gov/data.html

5.2 ArXiv Headline Generation

Directly comparable to the previous *Gigaword* dataset, we provide a new novel *ArXiv Headline Generation* datasets, which offers new opportunities to evaluate or train models. By comparison to the existing *Gigaword* dataset, the task stays the same, creating headlines based on multiple source sentences. In contrast to most previous published corpora, our dataset does not consist of news article related examples. The data for our new corpus is extracted from *arXiv*[4]. The source data consists of the abstracts of research papers and the targets are articles titles.

5.2.1 Data Structure

Compared to the *Gigaword* corpus the basic structure stays the same, containing source and target files for training, validating and testing. Moreover we publish two different datasets containing data from different research fields. The first corpus contains all papers from Computer Science research, which are available on arXiv until December 2018, which results in more than $180k$ examples. In addition we present a corpus including abstracts and titles from the field of Physics research, storing more than $970k$ examples. Both models can be used for separate training, giving new opportunities to compare the performance of models trained on different type of domain. In Section 7.2.3 we present our results that show the ability to generalize over different subjects used for training and testing.

5.2.2 Corpus Creation

We use the fact that *arXiv* participates in the Open Archives Initiative[5] (OAI) as OAI-PMH data-provider. Lagoze and Van de Sompel [2001] presented the initiative in 2001, enhancing the access to e-prints in scholarly communication. Data-providers like *arXiv* submit nightly meta data updates. For our purpose, the meta data includes all information we need to build our corpus, containing abstracts and titles from all newly added papers. Furthermore, classification of papers in research areas is also available to distinguish between physics and computer science papers. Using a freely available OAI-PMH implementation[6], accessing the *arXiv* metadata is also free of charge.

5.2.3 Corpus Preprocessing

We apply additional preprocessing on the data harvested from *arXiv*. This includes transforming to lower case, normalization of punctation and tokenization. Furthermore

[4] arxiv.org/
[5] openarchives.org/OAI/2.0/openarchivesprotocol.htm
[6] github.com/infrae/pyoai

we replaced numbers, removed formulas from abstracts and deleted special characters. In table 5.1 we show two examples from our new corpora.

5.2.4 Corpora Comparison

Finally we provide statistical comparison between *Gigaword* and *ArXiv Headline Generation* corpora to present differences and similarities on the data level. In table 5.2 the first row shows the number of examples in each corpora. In this case, significant differences between the three datasets exist, to compare how much the corpus size affects the performance. Additionally we present the data distribution of words for source and target side. As we can clearly see, the number of words is higher on our own datasets, compared to *Gigaword*. In contrast to this fact, the number of words on target size stays nearly constant between all corpora. In conclusion, newly created models need to summarize from longer sources to constant headline sizes.

Table 5.2: Comparison of Headline Generation Datasets

		arXiv Headline Generation	
Corpus	**Gigaword**	**Computer Science**	**Physics**
#Examples	3803957	186578	972048
Source Comparison			
Lowest #Words	11	2	1
Highest #Words	99	512	846
Average #Words	30.8	139.7	122.6
Target Comparison			
Lowest #Words	2	2	1
Highest #Words	45	46	66
Average #Words	7.7	9.9	9.9

In the following we present analysis of data distribution over all datasets for the headline generation task. The histograms 5.1 *Data Distribution* Gigaword *Source* and 5.2 *Data Distribution* Gigaword *Target* visualize the distribution of source and target lengths in the corpus. We show the same statistics for *ArXiv Headline Generation* copora including Computer Science papers (see figure 5.3 *Data distribution Computer Science Papers sources* and 5.4 *Data distribution Computer Science Papers targets*) as well as Physics papers (see 5.5 *Data distribution Physics Papers sources* and 5.8 *Data distribution Physics Papers targets*). On one hand, the histograms show the source side (in *Gigaword* corpus it's visualizing an abstract of news article and in *ArXiv Headline generation* corpora it's the abstract of papers), the data distribution

Table 5.1: Examples of novel ArXiv Headline Generation corpora

Field of Research	**Abstract**	**Title**
Computer Science	in this paper we study the phase transition behavior emerging from the interactions among multiple agents in the presence of noise . we propose a simple discrete time model in which a group of non mobile agents form either a fixed connected graph or a random graph process and each agent taking bipolar value either # or # updates its value according to its previous value and the noisy measurements of the values of the agents connected to it . we present proofs for the occurrence of the following phase transition behavior at a noise level higher than some threshold the system generates symmetric behavior or disagreement whereas at a noise level lower than the threshold the system exhibits spontaneous symmetry breaking or consensus . the threshold is found analytically . the phase transition occurs for any dimension . finally we demonstrate the phase transition behavior and all analytic results using simulations . this result maybe found useful in the study of the collective behavior of complex systems under communication constraints .	phase transitions on fixed connected graphs and random graphs in the presence of noise
Physics	we consider the interpretation of tetrad fields as reference frames inspacetime . reference frames may be characterized by an antisymmetric acceleration tensor whose components are identified as the inertial accelerations of the frame . this tensor is closely related tog ravitoelectromagnetic field quantities . we construct the set of tetrad fields adapted to observers that are in free fall in the schwarzschild spacetime and show that the gravitational energy momentum constructed out of this set of tetrad fields in the framework of the teleparallel equivalent of general relatrivity vanishes . this result is in agreement with the principle of equivalence and may be taken as a condition for a viable definition of gravitational energy .	on reference frames in spacetime and gravitational energy in freely falling frames

Figure 5.1: Data Distribution *Gigaword* Source

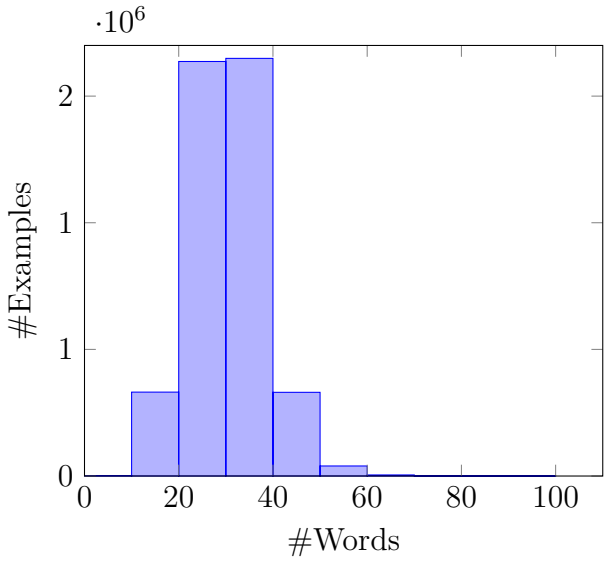
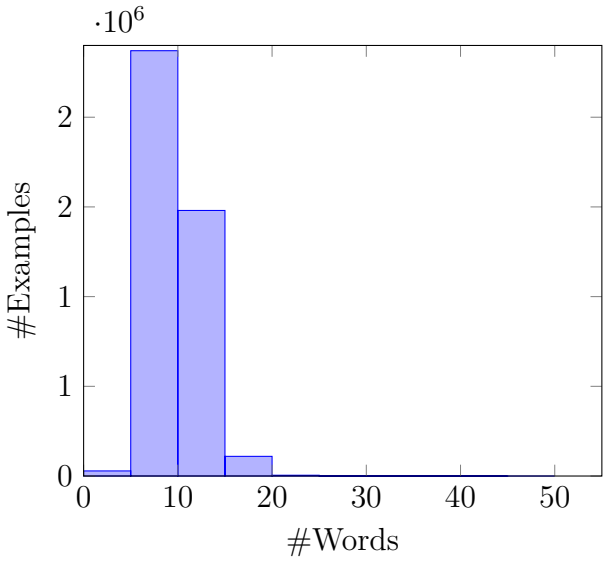


Figure 5.2: Data Distribution *Gigaword* Target



is similar, but having different scale of X-Axis. Where the most examples of *Gigaword* Corpus containing 20 to 40 words, the other corpora contain at least 50 up to 250 words. On the other hand, the target side shows similar data distribution over all corpora, where most of the headlines contain between 5 and 25 words. Finally this analysis can be used to remove outliers from the corpora, to reduce the range of words. As we can see in all histograms, the right side is often deserted, which means the amount of examples in containing that number of words is very low. The number compared to the mean value is much smaller.

5.3 CNN/ Daily Mail

Hermann et al. [2015] introduced the *CNN/ Daily Mail* dataset in 2015 for their task on teaching machines to answer questions. Insufficient data lead them to create a new corpus, collecting data from CNN[7] since 2007 and Daily Mail[8] since 2010. As the first, Nallapati et al. [2016] first used the corpus in 2016 for abstractive summarization.

5.3.0.1 Data Structure and Preprocessing

Comparing the *CNN/ Daily Mail* dataset to the previous datasets from headline generation the size of sources and targets increased. The enhanced task of abstract generation uses sources containing multiple sentences to produce abstracts also consist-

[7] www.cnn.com
[8] www.dailymail.co.uk

Figure 5.3: Data distribution *Computer Science Papers* sources

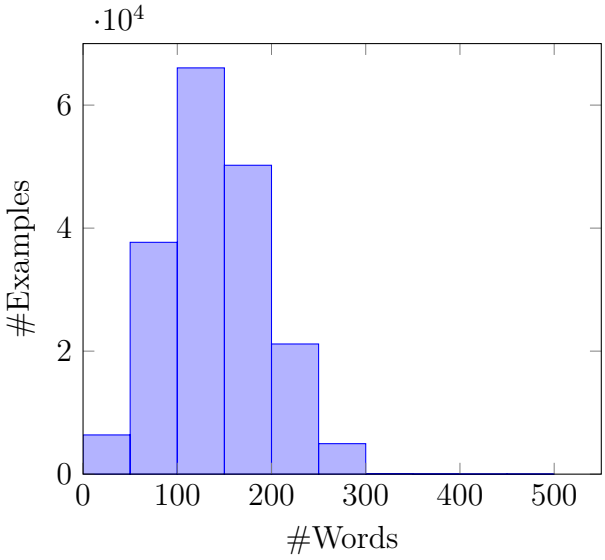


Figure 5.4: Data distribution *Computer Science Papers* targets

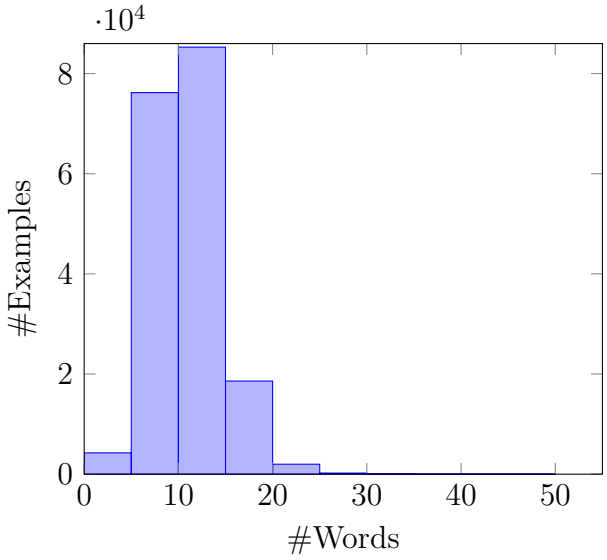


Figure 5.5: Data distribution *Physics Papers* sources

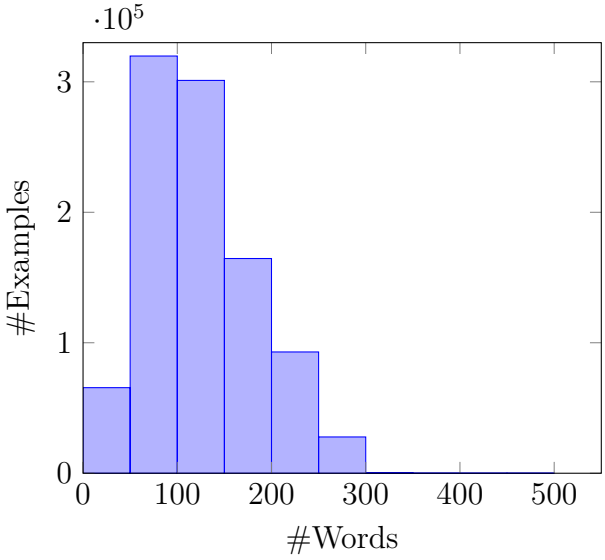
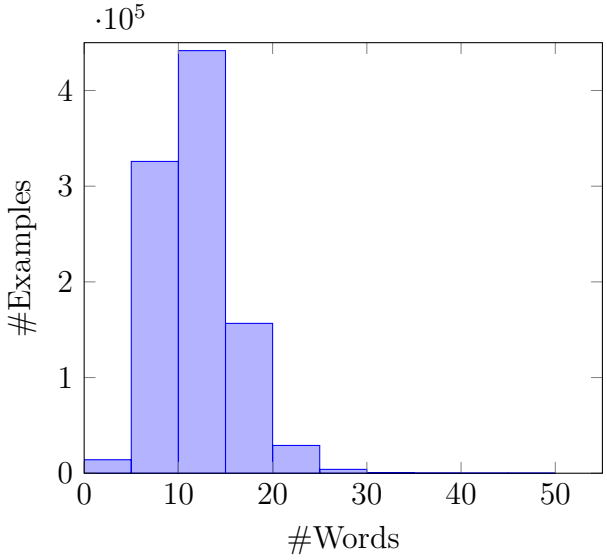


Figure 5.6: Data distribution *Physics Papers* targets



ing of multiple sentences. In section 5.4.1.1 *Corpora Comparison* we present once more a statistical comparison between the freely available *CNN/ Daily Mail* dataset and our own *ArXiv Abstract Generation* dataset. Furthermore to prepare the original corpus for the summarization task, several preprocessing steps are needed. The original dataset contains the story, which is a news article. It also contains highlights, which are concatenated to create the articles abstract. Next, we remove outliers from the dataset. As we can see in table 5.3 *Comparison of abstract generation datasets*, the example containing the highest amount of words is far above the average. Removing all examples over 380 words removes only 35 out of 311971 entries from the dataset. Finally we replace unnecessary characters and numbers in sources and targets with placeholders.

5.4 ArXiv Abstract Generation

Until now, *CNN/ Daily Mail* corpus is the only available dataset for abstract generation task on summarization, which is big enough to train modern neural systems. Because of this, we introduce a new dataset for abstract generation task. Our corpus is based on research papers published on *arXiv*, similar to our corpora for the headline generation task.

5.4.1 Data Structure

The main idea behind our new corpus is to create research paper abstracts from the papers contents. Doing so, we download the whole papers LaTeX[9] sources, apply preprocessing which we describe more in detail in section 5.4.2 to produce plain text. One of the problems we face during the preparation of our dataset is the wide range of content length of the papers. This problem is resolved by removing outliers and applying further selection of sentences by their statistical word importance using *tf-idf*. Our current corpus contains papers published in the range from july 2017 to december 2018 resulting in 143882 successfully processed papers. After removing outliers which includes huge papers, we present our novel corpus containing 91847 examples. Additionally our scripts can be applied to generate more examples from the past years to extend the dataset. We discuss further possibilities about this dataset in our future work in section 8.2.

5.4.1.1 Corpora Comparison

In this section we compare the corpus *CNN/ Daily Mail* with our own, presenting results in table 5.3 *Comparison of abstract generation datasets*. The following analysis

[9] www.latex-project.org

Table 5.3: Comparison of abstract generation datasets

Corpus	CNN/ Daily Mail	arXiv Abstract Generation
#Documents	311971	91847
Source Comparison		
Lowest #Words	9	3
Highest #Words	2190	7417
Average #Words	688.4	3363.9
Target Comparison		
Lowest #Words	2	2
Highest #Words	1953	346
Average #Words	49.5	141.5

of *CNN/ Daily Mail* dataset is done after removing outliers. As we can see on the source side, our dataset contains longer sequences with 3364.1 words in average compared to 688.4 words. Also the target side reflects the same, surpassing the *CNN/ Daily Mail* datasets with respect to the sequence length.

5.4.2 Corpus Creation

Similar to the datasets in section 5.2 our novel *ArXiv Headline Generation* corpus is based on papers from *arXiv*. In contrast, the abstracts of papers, which were the source of the summarization in the last corpora, become now the target. These targets should be generated by the entire content of each paper. This raises several new challenges, we faced during creation of our novel corpus. First, the usage of OAI-PMH delivers only abstracts and titles of papers. Consequently a new way to access the LaTeX Sources of papers is needed. This is realized by using a bulk data access[10] provided by Amazon S3[11] where *arXiv* publishes monthly all new papers sources in archives. These archives are downloaded using s3cmd[12] implementation. Amazon S3 applies the *requester pays buckets* concept which is not free of charge. In the following we conduct several steps to process the resulting corpus from LaTeX sources. We will describe these steps shortly in the following sections.

[10] arxiv.org/help/bulk_dat_s3
[11] aws.amazon.com/s3/
[12] github.com/s3tools/s3cmd

Figure 5.7: Data distribution *CNN/ Daily Mail* sources

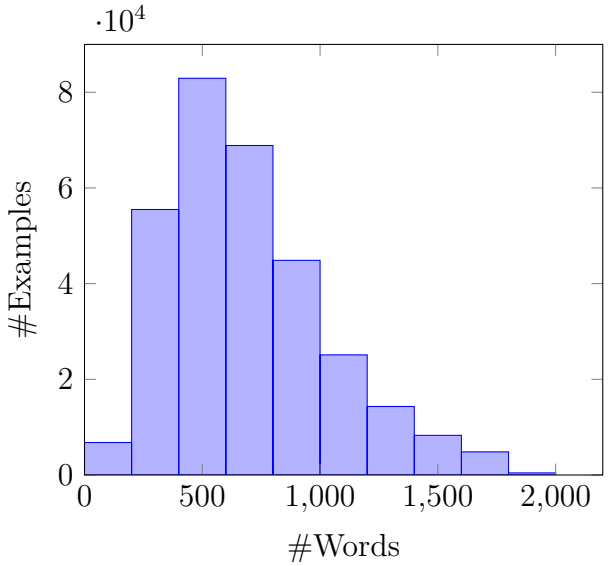


Figure 5.8: Data distribution *CNN/ Daily Mail* targets

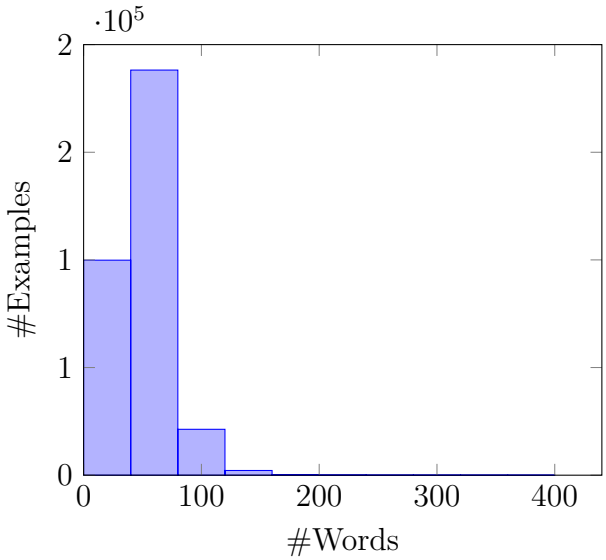


Figure 5.9: Data distribution *ArXiv Abstract Generation* sources

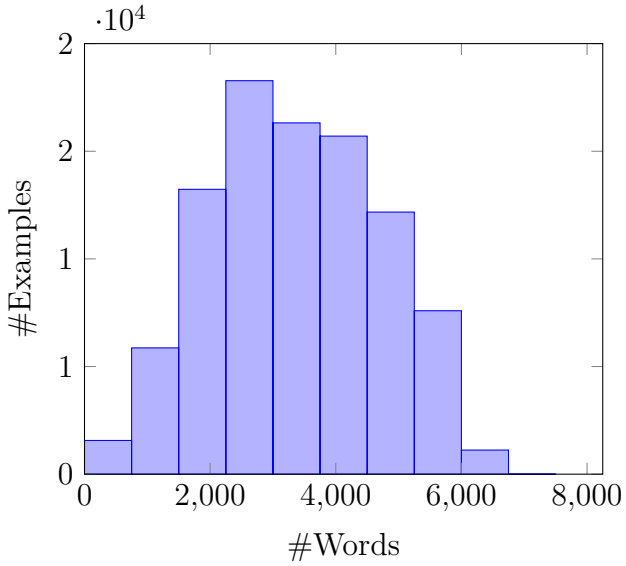
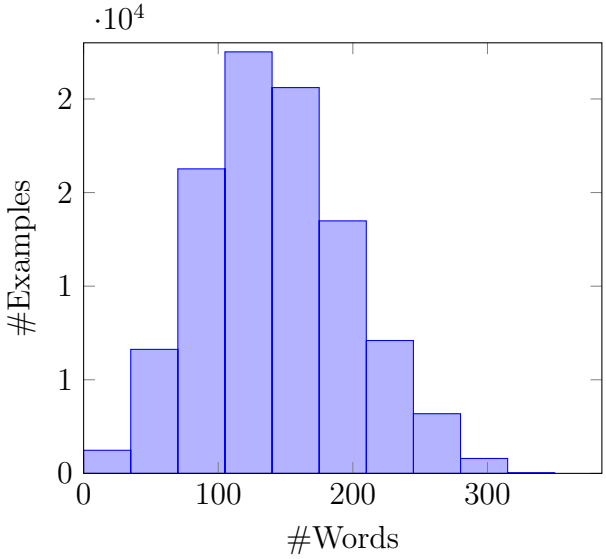


Figure 5.10: Data distribution *ArXiv Abstract Generation* targets



5.4.2.1 LaTeX sources to plain text

To use papers as training data in the summarization task, it is necessary to remove all LaTeX commands. Before this happens, the archives are unpacked and preprocessed. In this step, we remove unnecessary captions of figures, tables and formulas directly in the LaTeX sources. This is already done in LaTeX sources, because filtering for special LaTeX commands is far easier than removing it in plain text. Moreover we search for the main LaTeX file, which can be different for each paper. After the main file is indicated, we combine all sources. Next, we conduct a very important step, which removes the abstracts of the papers from our source files. Without this step, the system would be trained on inputs containing the desired outputs. After this we use adjusted implementation of *opendetex*[13] to remove LaTeX commands. Moreover preprocessing is applied to the resulting plain text files to remove unnecessary characters, which is described in the next paragraph.

5.4.2.2 Plain text preprocessing

In addition we process the plain text files, to remove unnecessary content. For example we apply our preprocessing on formulas, numbers, comments and special characters to reduce the complexity. A lot of special characters from the LaTeX source are copied to plain text by *opendetex*, which are redundant or needless. Another problem appears in papers from mathematics or physics, which include large sections of formulas and equations. Removing them results often in words without connections. To manage this, we exclude the whole sentence. This should not affect the summarization performance, because detailed information about equations are usually not part of the abstract. Last but not least we apply lowercasing, punctation normalization and tokenization, as we already presented in our previously created datasets for the headline generation task.

5.4.2.3 Matching papers and abstracts

Once more we use our OAI-PMH implementation from the *ArXiv Headline Generation* corpora to download abstracts for our new *ArXiv Abstract Generation* dataset. Doing so, the unique *arXiv* ID is used to combine the plain text of the source papers with the abstracts. We apply the same preprocessing to abstracts as described in section 5.2.3.
In the following chapter we describe our novel experiments conducted on the new created datasets in this thesis. Furthermore we use the publically accessible datasets and compare the performance of the *Transformer Model* to previous approaches.

[13] github.com/pkubowicz/opendetex

6 Experiments

Performing abstract summarization is about finding the most important facts from a given source sequence and generating new output, which puts the facts into readable and grammatically correct sentences.
Our main contributions are presenting the first evaluations on the commonly used summarization corpora *Gigaword* and *CNN/ DailyMail*. In addition we present novel datasets harvested from *arXiv* providing new possibilities in comparison of models performance and generalization. Furthermore we provide a very challenging dataset including long source and target sequences, which contains research topics compared to existing news articles datasets.
In the following we introduce our experiments, in the beginning mainly focusing on searching for ideal hyperparameters in the training phase of the *Transformer Model* in the field of summarization, presented in section 6.1. Moreover we conduct several tests with different preprocessing steps shown in section 6.1.1.
Furthermore we present an entirely new experiment investigating the generalization performance of abstractive models, if they were trained on different research topics. We show novel results in evaluating models in unseen science fields. Afterwards we present how fast a model can catch up after finetuning its parameters on the other research topic.
Working with datasets containing large inputs, memory costs are a problem using the *Transformer Model*. Different approaches and their results are shown in 6.3.1.
After the training phase, we further investigate the possibilities to improve the performance of our models generating summaries of higher quality in the inference process.
In the end we sum up our results and present our best models on each of the five datasets that are used in this work.

6.1 Models and Experimental Setup

In the following we describe the experiments showing the usage of different hyperparameters in training phase. For the tests we use the implementation[1] of the *Transformer Model*, which was forked from main repository to track all changes of this thesis in a different fork. This implementation is oriented at the original model published by

[1] github.com/quanpn90/NMTGMinor

Vaswani et al. [2017] and is intended for use in machine translation research.
Finding the suitable hyperparameters is key in the training phase of the *Transformer Model*. Because of missing comparable research work, we first looked to the machine translation task, where the *Transformer Model* already replaced previous encoder-decoder models based on LSTMs. Popel and Bojar [2018] propose several suggestions how the *Transformer Model* can be trained to achieve the best results on the machine translation task. We mainly investigate the influence of different number of *layers* and *learning rates*. Moreover the influence of novel update method *noam* ([Vaswani et al., 2017]) is investigated and compared to previous used approaches.
Our implementation splits up the *batch size* in two components. The first component is called *batch size words*, which defines the maximum number of words used in each iteration. The second component is called *batch size sentences* and controls the number of lines used from the parallel training corpus for single iteration. One iteration contains the forward pass using encoder and decoder layers. Using a generator, the decoder outputs are converted to tokens using a generator, described in section 2.4.2.2. We apply *negative likelihood loss* (*NLLLoss*).

6.1.1 Preprocessing Setup

We apply different preprocessing steps on our datasets, such as investigating the influence of removing unnecessary tokens, applying BPE using different merge sizes or splitting tokens in smaller n-grams to reduce the vocabulary size. Moreover we use scripts published by Moses[2] including tokenization (inserting spaces between words and punctuation) and cleaning (Removing mis-aligned sentences).
These steps were also very helpful in the time we created our new datasets presented earlier. Because of very noisy data returned by the crawler we used to get data from *arXiv* (see 5.4.2).

6.2 Generalization Experiments

In this section we describe the experiments about the generalization ability of the *Transformer Model*. At first we train two separate models on each research topic, computer science and physics, described in section 5.2. Based on these models we conduct evaluations, generating headlines (summaries) for each model on a random test set collected from the dataset, excluded from the dataset before it was trained on it. These results serve as our baseline in this experiment. We apply the standard evaluation metric (3.3) ROUGE to measure the summaries quality. Additionally we determine the number of copied words and average length of outputs.
After showing the initial performance of our models on similar data compared to

[2] statmt.org/moses/

the training data, we investigate the scores our models can reach on test sets from a different research subject. We apply the same measurements on the generated outputs. This is followed by our last experiment, to determine how fast a model can catch up on a different topic by finetuning it on the provided training data. We measure the number of epochs needed for our best model to produce summaries of similar quality than its counterpart baseline model. On top we perform the initial tests for our baseline system once again, to see if the performance of the finetuned models has changed on their originally learned data.

6.3 Model adaptation

In this section we describe our model adaptations performed on the *Transformer Model* approach to overcome problems we faced during this work and furthermore improve the performance on well-known problems in summarization, as we showed in our chapter about related work 4.

6.3.1 Reducing Memory costs

Working on long sequence inputs, memory costs are always a problem in machine translation. To address this problem we investigated three approaches to reduce the memory allocation during training of our *Transformer Model*.

6.3.1.1 Local-Attention Experiments

Liu et al. [2018] showed in their work the possibility of the *Transformer Model* to be able to work with very long input sequences by adapting the computation of *self-attention*. The author didn't publish either their implementations of *local-attention* nor *memory-compressed attention*. To overcome our memory problems while training on large inputs from our *ArXiv Abstract Generation* dataset (5.4) we included their ideas in our implementation.
Reusing their idea to split inputs in fixed sized blocks to reduce memory costs in computing the *self-attention*. We observe, that calculating k_i (key) and v_i (value) from equation 2.9a on smaller blocks reduces memory costs because smaller linear transformation is needed. To keep performance of *self-attention* constant we decide to keep q_i (queries) on its original size. To apply the computation of scores in equation 2.9b we then use a masked dot product, only calculating the partial score for a reduced size of k_i. These values are then merged into one full score to compute the full softmax afterwards. While saving around 20 percent of memory, the speed decreased about 50 percent compared to our original implementation. Because of this loss, we prefer using *gradient checkpointing* instead of *Local-attention*, which we will introduce in the next sections.

6.3.1.2 Gradient Checkpointing Experiments

Our implementation provides the possibility to use *gradient checkpointing* ([Griewank and Walther, 2000]) for individual layers of the *Transformer Model* on the encoder as well as on the decoder side. *Gradient checkpointing* describes a method to save memory by recalculating the forward pass while propagating the gradients backwards through the model. More in detail, in the common implementation, the activation values of single learnable parameters in the model are saved after each forward pass, to calculate the gradients and therefore the weight updates in the subsequent backward pass. By applying *gradient checkpointing* we do not store the activations after the forward pass. In contrast only the current loss is saved. As a result the activations are once more calculated during the backward pass to use them to compute gradients using backprogagation. To sum it up, we again sacrifice speed to save memory, but compared to *Local-Attention* the reduction of speed is smaller. Therefore we prefer this method to reduce our memory cost when working on long input sequences.

6.3.1.3 TF-IDF Sentence Ranking Experiments

Using *TF-IDF Sentence Ranking* (3.2) to reduce memory does not count as model adaptation. We include it in this section, because we used this method, described in 3.2, to exclude statistically less important sentences from our source dataset. This reduction of length saved memory while processing the sequences in the training phase. One advantage of this method is the flexible configuration of the length threshold we can apply to the source sequences. This threshold sets the desired amount of words after the process. Our implementation excludes less important sentences (by *TF-IDF* scores), until the threshold is reached. Therefore we are able to reduce the length of sources to a length we choose, for example to use the memory of GPU in the most efficient way.

6.4 Inference Experiments

Presenting the last section of experiments, we apply different tests in the inference phase on trained models depending on different characteristics of our datasets. A common problem is finding suitable parameters to generate the best summaries. We evaluate different hyperparameters including coverage penalty and length normalization presented by Wu et al. [2016].

6.4.1 Parameter Tuning

Achieving best possible results on a trained models in summarization depends on the accurate parameters in inference. The hyperparameters describing the coverage

penalty and length normalization presented by Wu et al. [2016] are used to regulate the outputs. These hyperparameters in inference are known as *alpha* and *beta* values. Coverage penalty *beta* is used to encourage the model to translate all of the provided input. Length normalization *alpha* deals efficiently with the problem of comparing hypotheses of different lengths during decoding. We conduct experiments on generating different length summaries to explore the effects of resulting *ROUGE* scores. Moreover we test if the influence of different coverage penalties to models improves the outputs.

6.4.2 Removing Duplicated Trigrams

Paulus et al. [2017] observe in their evaluation on the *CNN / DailyMail* dataset, that the reference summaries almost never contain the same trigrams twice. Replacing already generated trigrams with different tokens at testing time improved their results. We apply this experiment on the *CNN/ DailyMail* and the *ArXiv Abstract Generation* datasets to verify if this assumption can improve the quality of the output summaries in general. We use this datasets, because our baseline outputs suffer from repeated tokens at test time, which reduce the *ROUGE* scores constantly.

7 Results

In this chapter we present the results of our experiments. We explain our results of the experiments in the order we proposed them in section 6. Starting with *Experimental Setup*, followed by our *Generalizaton Experiment*. In the end we present experiments we conduct during inference, ending with a full overview about the evaluation for all used datasets.

7.1 Experimental Setup

This section gives an overview on our configurations used on the *Transformer Model* for summarization tasks. If no other information on the specific experiment is given, we use the following settings as default. We apply word embedding size of 512 dimensions and a internal feed forward layer of 2048 which is part of each encoder and decoder layer (Figure 2.7). Furthermore we apply *Adam* ([Bengio and LeCun, 2015]) as optimization algorithm. In addition we use BPE with $30k$ merge operations as preprocessing to our corpora.

7.1.1 Parameter Tuning

We begin presenting our results of finding ideal hyperparameters for the summarization task with the *Transformer Model*.

7.1.1.1 Experiments with different Number of Layers

In the following we present the results of our experiments finding the ideal number of layers in the *Transformer Model* for summarization task. We conduct these experiments on several of our datasets shown in section 5 to confirm our results. We conduct early experiments on insufficient preprocessed training data, which leads to worse results. Nevertheless we present the results of these experiments in the sections 7.1.1.1 and 7.1.1.2, because they show different model settings, which leads us to the best configuration we use for later experiments and corrected preprocessing, returning our best results.

In table 7.1 we describe different configurations of numbers of layers for the encoder and decoder in our *Transformer Model*. We enhanced our implementation to allow imbalanced models. Overall we observe the best results using balanced encoder and

Table 7.1: Experiments with different number of layers on *CNN/ DailyMail* Corpus

Encoder Layers	Decoder Layers	ROUGE-1	ROUGE-2	ROUGE-L
6	6	22.45	6.43	16.75
4	4	21.92	5.79	16.02
3	3	20.53	5.31	15.67
6	4	21.80	5.73	16.23
4	6	21.73	5.71	16.10

Table 7.2: Experiments with different number of layers on *Gigaword* Corpus

Encoder Layers	Decoder Layers	ROUGE-1	ROUGE-2	ROUGE-L
6	6	31.95	13.43	27.68
4	4	31.77	13.02	26.94
3	3	30.55	13.22	25.34

decoder layers sizes with amount of 4 and 6. Furthermore, we see similar results using the *Gigaword* dataset, strengthening our results. Table 7.2 depicts the results of evaluation which matches with our previous experiments. In the following we use balanced models containing 4 and 6 layers as baseline models for our other datasets. The choice of the number of layers naturally changes the number of training parameters of our *Transformer Model*, making deeper networks both contain more parameters and increase training time substantially. Testing deeper models containing 8 or more layers on encoder and decoder side raise the amount of parameters in the model significant. This leads to resource problems on the available GPUs and time.

7.1.1.2 Experiments with different Learning Rates and Batch Sizes

In this section we show the influence of different learning rates and update methods. As baseline we use our *Transformer Model* containing 6 layers. The experiment is performed on the *CNN/ DailyMail* dataset. Table 7.3 compares *Mini-batch Gradient Decent* with sizes of 2048 (baseline) and 1024. We apply a constant learning rate of 0.001.

Table 7.3: Experiments using different Batch Size Update on *CNN/ DailyMail* Corpus

Batch Size Update	Learning Rate	ROUGE-1	ROUGE-2	ROUGE-L
2048 (baseline)	0.001	22.45	6.43	16.75
1024	0.001	21.32	5.43	15.12

Table 7.4: Experiments using different Update Methods on *CNN/ DailyMail* Corpus

Update Method	Warm-up steps	ROUGE-1	ROUGE-2	ROUGE-L
regular		22.45	6.43	16.75
noam	**4000**	**24.25**	**6.82**	**19.79**

In our last experiment of parameter tuning we compare regular learning rates, where the learning rate is constant in the training process, with the learning rate decay scheme *noam*, proposed by the authors Vaswani et al. [2017]. *Noam* describes a scheme how to bring the learning rate warm-up and decay together. It combines linear warm-up for a given number of training steps, followed by a inverse square root decay. Using *noam*, we could outperform our baseline on each *ROUGE* score. However we still perform a lot worse in *ROUGE* scores compared to previous encoder-decoder models for summarization based on LSTMs. We discuss this problem and provide our solution in the next section 7.1.2.

7.1.2 Impact of Preprocessing

Table 7.5: Comparison after improving preprocessing

Model	ROUGE-1	ROUGE-2	ROUGE-L
seq-to-seq + attn baseline ([See et al., 2017])	30.49	11.17	28.08
Transformer + insufficient preprocessing (6 Layers)	24.25	6.82	19.79
Transformer + accurate preprocessing (6 Layers)	32.21	11.08	27.08

This section investigates the influence of our preprocessing steps on the summarization performance. Using BPE (2.5) in preprocessing is a standard procedure in neural machine translation since it was published. Sennrich et al. [2015] propose widely used implementation to segment text into subword units. It improves handling of Out-Of-Vocabulary (OOV) words. In addition the authors offer implementations to further segment rare words into character n-grams, intended to separate words in individual parts which are known by the model. In our earlier experiments this script is used to segment rare words in smaller n-grams. Besides drastically reducing the vocabulary size it also reduces the performance of our models. In table 7.5 we compare our models including insufficient and accurate preprocessing, returning similar results as the baseline of See et al. [2017] in ROUGE scores. In our further experiments we build the models with sufficient preprocessing only applying BPE, without further word splits. We obtain overall good *ROUGE* score performance in the following

experiments. Detailed comparison between state-of-the-art systems and models in this work is presented in section 7.5

7.2 Generalization Experiment

In this section we present our novel experiment, studying the ability of neural models, especially the *Transformer Model* in our work, to generalize on different subjects. Publishing the corpora *ArXiv Headline Generation* we present two datasets with similar source and target structures but offering topics on different research subjects.

7.2.1 Initial Experiments on ArXiv Headline Generation Dataset

Table 7.6: Experiments using *ArXiv Headline Generation* Physics Model

Evaluation Set	**Model**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
PY	Transformer (4 Layers)	42.61	23.14	36.79
PY	Transformer (6 Layers)	43.54	23.60	37.72
CS	Transformer (4 Layers)	27.31	9.49	23.03
CS	Transformer (6 Layers)	28.02	9.65	23.78

Table 7.7: Experiments using *ArXiv Headline Generation* Computer Science Model

Evaluation Set	**Model**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
CS	Transformer (4 Layers)	32.63	12.39	26.93
CS	Transformer (6 Layers)	33.25	12.55	27.09
PY	Transformer (4 Layers)	19.71	3.68	15.84
PY	Transformer (6 Layers)	19.44	3.4	15.76

Presenting our initial experiments on both datasets we show *ROUGE* score evaluations of processing our test set using the Physics model (PY) in table 7.6 and the Computer Science model (CS) in table 7.7. It is clearly visible that both models perform well on their test set randomly excluded of the dataset before they are trained on it. However we can discover, that the Physics model performs nearly 10 *ROUGE* points better on each score. We think this because of the larger amount of training data the Physics corpus contains, compared to Computer Science.
Furthermore we provide evaluations conducted on the respective other test set of

subjects. The results show that the Physics model performs much better and nearly reaches performance of the computer science model itself. Reasons for this might be also the larger number of training examples in Physics corpus and moreover the increased size of vocabulary ($34k$ on source and $33k$ on target side in Physics dataset compared to $31k$ and $29k$ in Computer Science dataset). For our tests we apply BPE (2.5) using the trained models from our initial preprocessing. In BPE preprocessing of finetuning the Physics model the BPE model of the initial Physics dataset is used on the Computer Science corpus. The same procedure is conducted in finetuning of the Computer Science model.
In addition we see very similar results for our balanced models containing 4 and 6 layers in encoder and decoder, which confirms our initial experiments in section 7.1.1.1.

7.2.2 Ability to Generalize

To illustrate the ability of generalization we compare the outputs of both models in table 7.8. It contains an example of the Computer Science test set, including the source and reference title of the paper and predictions of both models before and after the finetuning. In section 7.2.3 we will discuss the results of our finetuned models in more detail.
It is clearly visible that the original title of our shown example is focused on the method used in their paper *non parametric belief propagation* and their goal *fault identification*, which can be seen in table 7.8. Both is successfully identified and used to generate a new summary by the Computer Science Model. In contrast, the Physics Model has problems identifying the method and summarizes the goal inaccurately. Besides this it already produces acceptable outputs on different research subjects.
Anticipating the results from our following experiments on finetuned models, we show the improved output of the Physics model after it was trained for 3 epochs on the Computer Science corpus. Section 7.2.3 shows further evaluation scores. Reviewing the new generated output of Physics model (shown in table 7.8) illustrates the improvement of our model in determining the method of their paper *non parametric belief propagation*, which is now the same as the output of the Computer Science model. Moreover, the model improved its output of their thesis goal from *linear measurements of fault location* to *fault identification*, which also matches with the goal of the original title. In contrast, the results of the finetuned Computer Science model decreased in generating the accurate title. The new output shows the correctly identified goal *identification of fault patterns* but performed worse on finding the correct approach used in this research paper.

Table 7.8: Output Comparison in *ArXiv Headline Generation Task* on Computer Science test set

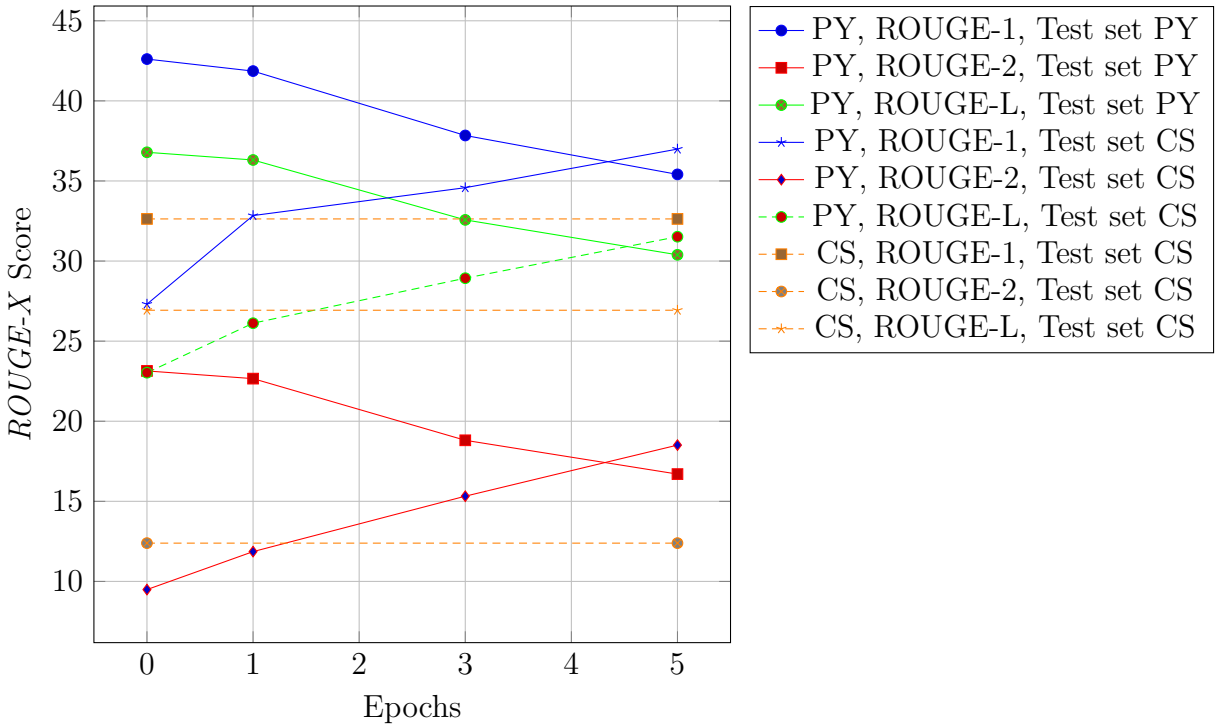
Source	we consider the problem of identifying a pattern of faults from a set of noisy linear measurements . unfortunately maximum a posteriori probability estimation of the fault pattern is computationally intractable . to solve the fault identification problem we propose a non parametric belief propagation approach . we show empirically that our belief propagation solver is more accurate than recent state of the art algorithms including interior point methods and semidefinite programming . our superior performance is explained by the fact that we take into account both the binary nature of the individual faults and the sparsity of the fault pattern arising from their rarity .
Gold Target	fault identification via non parametric belief propagation

	Model Computer Science	**Model Physics**
Prediction	a non parametric belief propagation approach to fault identification	a non parametric message passing algorithm for linear measurements of fault location
Prediction after Finetuning	identification of fault patterns from noisy linear measurements	non parametric belief propagation for fault identification

7.2.3 Finetuning Experiments

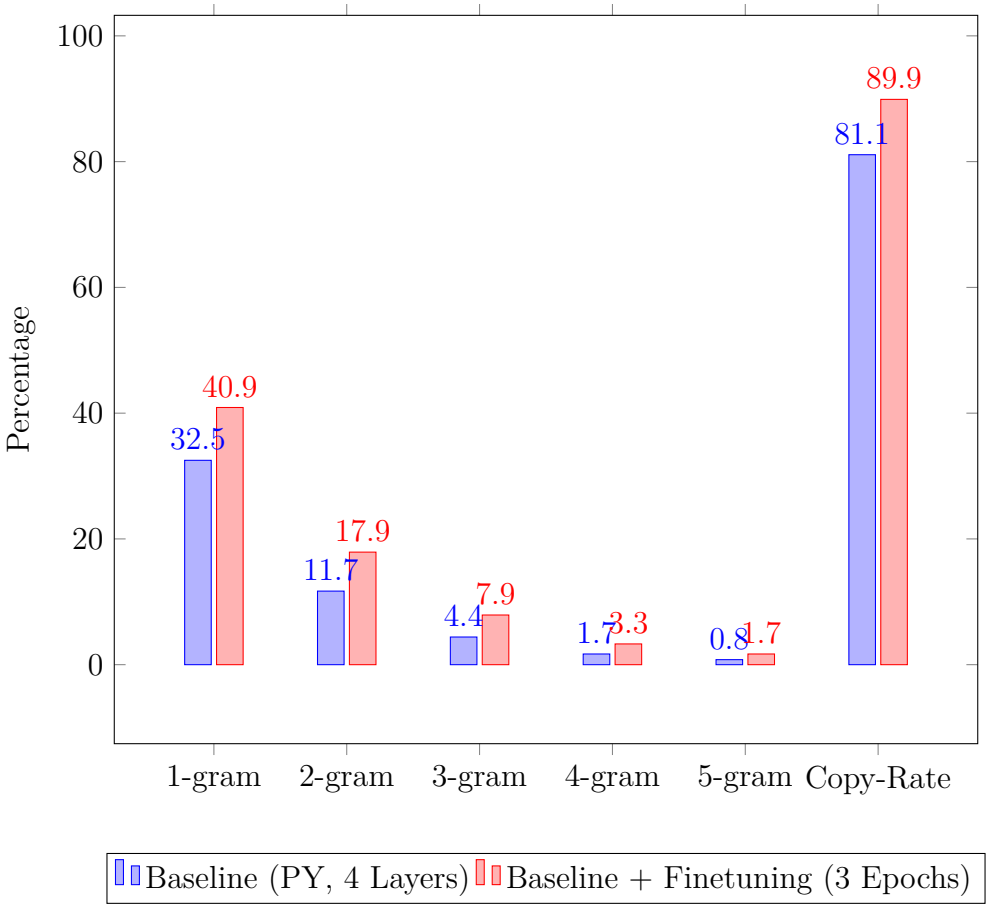
As last test in our Generalization Experiment we investigate how fast a model is able to catch up in terms of summarization performance compared to the model which is initially trained on this research subject. To explore this, we use our best performing Physics Model, trained it on the Computer Science dataset and vice versa. To make this possible we reused our vocabulary created on the initial training.
Evaluating the performance of summarization systems is often comparing the sources with the output of each model separately. A novel method to rate the results for our entire test set is an adaption of Kryscinski et al. [2018], which evaluates new n-grams produced by the model according to the source input and target summary. In our experiment we do not measure novel n-grams, in contrast we are interested in the overlapping n-grams between the outputs of our separately trained models. Measuring the overlapping 1-grams and 2-grams is equal to calculating *ROUGE* scores between target summaries. This evaluation describes the similarity of outputs given by different models. Now we compare our initial outputs and the outputs of the finetuned models, evaluating if the summaries get more similar. Additionally we measure the copy rate from source to summary and also the common *ROUGE* scores.
For our experiments, we use the *Transformer Model* containing 4 layers with reduced

Figure 7.1: *ROUGE-X* progress in Finetuning Experiment



amount of parameters as our baseline, because we think the model is likely to behave better in finetuning because of fewer parameters. In figure 7.1, we show the progress of *ROUGE-X* scores over 5 epochs. Values in Epoch 0 show the baseline scores of PY model on both test sets. The tuning of parameters on CS corpus increases the summarization performance on the corresponding dataset constantly for 5 epochs. In contrast we can detect a falling performance on the initial trained subject Physics. Between epoch 3 and 5 we can see the performance of CS is beating PY. As a result we want to set the focus on the model after first epoch, where the performance on Computer Science data increased a lot and on the other hand the decreasing performance for Physics is still acceptable. Furthermore we include the performance of CS baseline model (orange dotted lines) as constants to compare it with the finetuned model. Surprisingly the finetuned model outperforms the CS baseline after the first epoch. The reason may be the extensive training on Physics corpus, which contains considerably more examples compared to our Computer Science corpus (see corpus comparison in section 5.2.4).
In the end we compare the summaries on word level before and after the finetuning. Figure 7.2 depicts the overlapping n-grams, where we see a significant increasing overlap in n-grams between the initial PY model compared with the initial CS model and the finetuned PY model. In addition we present the Copy-Rate of both models in regarding to the sources. That value increased by 8.8% after 3 epoch parameter

Figure 7.2: Overlapping n-grams and Copy-Rate on baseline and finetuned PY model evaluated on CS



tuning. This indicates that copying the correct words from source increases the *ROUGE* scores.

7.3 Reducing Memory Costs

In this section we present our experiments to reduce memory costs of the *Transformer Model*. We mainly conduct these tests on the *CNN / DailyMail* corpus, to see the influence on datasets containing large inputs. The goal of these experiments is to compare existing methods to save memory and measure the influence on our results in the summarization task.

Our approaches to save memory, presented in section 6.3.1 differentiate between influencing the performance and only reducing the training speed. *Gradient checkpointing* has no influence on the results of forward- and backward-pass, it only changes its way of calculation. However, using *TF-IDF Sentence Ranking* (**??**) to rank sentences by their statistical importance allows us to exclude supposedly unimportant sentences. In

Table 7.9: Experiments to reduce memory on *CNN/ DailyMail*

Model	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
Transformer (4 Layers)	33.88	11.01	27.98
Transformer + TF-IDF (4 Layers)	34.05	11.67	28.25

the following we conduct experiments to measure the influence of removing sentences based on *TF-IDF Sentence Ranking*.
Table 7.9 shows our experiments using *TF-IDF Sentence Ranking* to reduce the source length of examples from the *CNN /DailyMail* dataset from average of 688.4 to a maximum of 400 words. To reduce the source words about around 40% on average, we apply our sentence ranking. Afterwards we exclude the most unimportant sentences according to this metric until we reach the threshold. We applied the reduction of source also on the test set. The results show that there is no major difference on *ROUGE* score evaluation between the full sources and the reduced. This indicates that the sentences, which were ranked as unimportant are classified as unimportant by the *Transformer Model* as well.
We use these results for our last experiment in section 7.5.4, to investigate the possibility of summarizing entire research papers from *ArXiv* using our novel corpus.

7.4 Inference Experiments

Achieving best possible results on a trained model in summarization depends on the accurate parameters in inference. In the introduction 6.4 of our conducted experiments we explained the coverage penalty *beta* and length normalization *alpha* presented by Wu et al. [2016]. Coverage penalty *beta* is used to encourage the model to translate all of the provided input. Length normalization *alpha* deals efficiently with the problem of comparing hypotheses of different lengths during decoding. In other words *alpha* is used to control the number of words the output summary should contain.
In table 7.10 we present our experiments using various settings. As baseline model we use our *Transformer + TF-IDF (4 Layers)*. Although these results give a good overview how to set the configuration in inference, the configuration depends on the desired target length, which depends on the reference summaries.
After conducting our tests we discovered changing *beta* value had no effect on generating different summaries. All experiments shown in table 7.10, *beta* is set to 1. Empirically we search for best *alpha* settings producing the best summaries on the *CNN/DailyMail* dataset. We observe our best results using *alpha* equals 6 producing summaries containing 48.93 words in average. Compared to the reference summaries our model produces outputs which contain 0.65 words less achieving our best results. Further increasing the *alpha* value did not result in better evaluation.

Table 7.10: Evaluation of Inference Experiment on *CNN/ DailyMail* dataset

Alpha	**Average Summary Length**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
0.3	30.96	30.12	10.43	24.33
0.6	36.49	31.74	11.04	26.46
0.9	43.81	33.12	11.40	28.33
1.2	47.24	33.58	11.40	28.33
2.0	48.50	33.77	11.67	28.76
5.0	48.91	33.95	11.67	28.25
6.0	**48.93**	**34.05**	**11.68**	**28.25**
8.0	48.92	33.86	11.62	29.32
10.0	48.93	33.78	11.47	29.33
	49.58 (Reference Summary)			

7.4.1 Removing Duplicated Trigrams

Removing duplicated trigrams at inference time, as purposed by Paulus et al. [2017], decreased the amount of repetitions in the generated summaries. In table 7.11 we compare the outputs of our baseline *CNN/ DailyMail* model from the previous inference experiment and replaced already generated trigrams with different predictions.

Table 7.11: Experiment removing duplicated trigrams

Model	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
Baseline	34.05	11.68	28.25
Baseline + Removing Duplicated Trigrams	**34.93**	**11.69**	**30.81**

Our experiment confirms the results of Paulus et al. [2017] that removing duplicated trigrams increases *ROUGE* scores. Moreover we conduct the same experiment for our *ArXiv Abstract Generation* dataset also suffering from generated repetitions in summaries. Similar to the previous experiment, we observe increasing *ROUGE* scores as well. We present the evaluation compared to our baseline in section 7.5.4.

7.5 Full Results Overview

Our last section presents an overview over the best performing models created in this thesis in comparison to the state-of-the art research papers.

7.5.1 Gigaword Corpus Evaluation

We begin to compare our results with state-of-the-art apporaches on the *Gigaword* corpus. Usually results are compared using the *DUC-2004* evaluation set and the *Gigaword test set* published by Rush et al. [2015]. In table 7.12 we present our evaluation on *DUC-2004* setting new state-of-the-art performance for *ROUGE-X* scores. Outperforming all previous encoder-decoder approaches using LSTMs and GRUs including special attention architectures shows the potential of the *Transformer Model* in neural machine translation tasks ([Vaswani et al., 2017]).
Furthermore we present our evaluation on the *Gigaword test set* in table 7.13 showing good summarization performance as well. Only the approach of Cao et al. [2018]) adding facts from the first sentence as additional input to the encoder performs better in terms of single overlapping n-grams and longer sequences. In contrast, our *Transformer Model* sets new state-of-the-art score for *ROUGE-2*.
Finally 7.14 shows further evaluations of our *Gigaword* model on *DUC-2003* and our own random test set excluded from entire *Gigaword* corpus. We can observe that our model exceeded a score of 50 in *ROUGE-1*.
Additionally we compare the Copy-Rate of our *Gigaword* models for the previous presented test sets of *Gigaword* dataset and *DUC-2004*, because this factor shows a significant influence of *ROUGE* scores in Headline generation task. Our model copied on average 75.8% of the source tokens in the new generated summary. Moreover our model copied nearly the same amount of words on *DUC-2004* dataset with about 75.1% in average. Douma [2018] published the same evaluation for their models. The best model on *DUC-2004* is *sent2vec+wemb*, which copied 84.3% of the words in their target summary. However comparing the *ROUGE* scores in table 7.12 show, that our *Transformer Model* outperforms their approach with lower copy rate. This shows that the *Transformer Model* model has a great ability to select important words from the source sequence, which also were selected by the humans, to achieve better *ROUGE* scores.

Table 7.12: Evaluation of Gigaword model on DUC2004 using F_1 Score

Model	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
ABS ([Rush et al., 2015]	26.55	7.06	20.12
ABS+ ([Rush et al., 2015])	28.18	8.49	23.81
Sent2vec+wemb ([Douma, 2018])	27.18	9.25	24.40
HA-RNNs ([Nallapati et al., 2016])	28.35	9.46	24.59
RAS-Elman ([Chopra et al., 2016])	28.97	8.26	24.06
Transformer (6 Layers) (ours)	30.79	**11.89**	**26.69**
Transformer (4 Layers) (ours)	**30.85**	11.57	25.86

Table 7.13: Evaluation of Gigaword model on test set published by [Rush et al., 2015]

Model	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
ABS+ ([Rush et al., 2015])	29.78	11.89	26.97
HA-RNNs([Nallapati et al., 2016])	33.78	15.97	31.15
RAS-Elman ([Chopra et al., 2016])	35.30	16.64	32.62
FTSum$_g$ ([Cao et al., 2018])	**37.27**	17.65	**34.24**
Transformer (6 Layers) (ours)	35.84	**18.09**	32.27
Transformer (4 Layers) (ours)	35.37	17.22	31.90

Table 7.14: Further Evaluation of Gigaword model

Evaluation Set	**Model**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
DUC2003	Transformer (6 Layers)	29.02	11.37	25.56
Random Gigaword test set	Transformer (6 Layers)	50.39	27.66	46.40

7.5.2 ArXiv Headline Generation Evaluation

Reviewing the scores presented on our novel *ArXiv Headline Generation* Corpus, we presented baseline systems for Physics and Computer Science subject in table 7.6 and 7.7 respectively. The analysis of presented results is done in section 7.2.1. Moreover we showed our finetuned experiments in figure 7.1. Our detailed description can be found in section 7.2.3.

7.5.3 CNN/ DailyMail Evaluation

Table 7.15: Evaluation of *CNN/ DailyMail* model

Model	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
seq-to-seq + attn baseline (150k vocab) ([See et al., 2017])	30.49	11.17	28.08
seq-to-seq + attn baseline (50k vocab) ([See et al., 2017])	31.33	11.81	28.83
HA-RNNs ([Nallapati et al., 2016])	35.46	13.30	32.65
Pointer-Generator Networks ([See et al., 2017])	39.53	17.28	36.38
Neural Intra-attention Model ([Paulus et al., 2017])	39.87	15.82	36.90
Transformer + TF-IDF (4 Layers)	34.93	11.69	30.81

Representing the dataset containing the longest source and target sequences in summarization task we present the final evaluation of our models trained on *CNN/ DailyMail* subsequently. The *Transformer Model* outperformed the baseline sequence-to-sequence models based on LSTMs on two out of three scores. Compared to modern approaches including novel coverage and copy mechanism included in the attention calculation the performance surpasses our models. *Transformer Models* are affected by a major problem of repeating words in the target summarization as well, which reduces the performance in each *ROUGE* score.
Moreover we analyze the abstractivness of our models, compared to the approaches presented by See et al. [2017] to measure the novel n-grams produced by the system which are not part of the source sequences. Figure 7.3 depicts our results in direct comparison to the Pointer-Generator Networks ([See et al., 2017]) shown in Figure 7.4. Comparing the abstractivness of all summaries, our model is located between the *sequence-to-sequence + attention* baseline model and the reference summaries. Because our model outperforms the baseline of See et al. [2017] the *Transformer*

Figure 7.3: Novel n-grams of *Transformer Model*

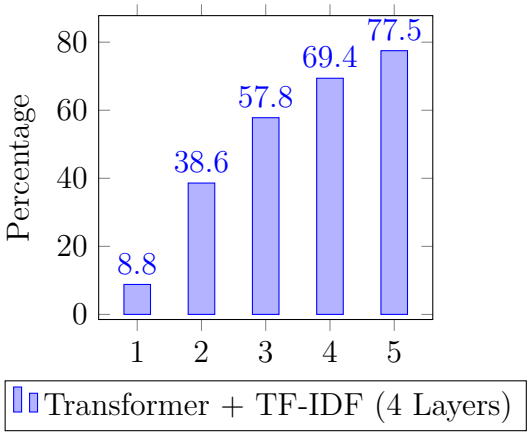
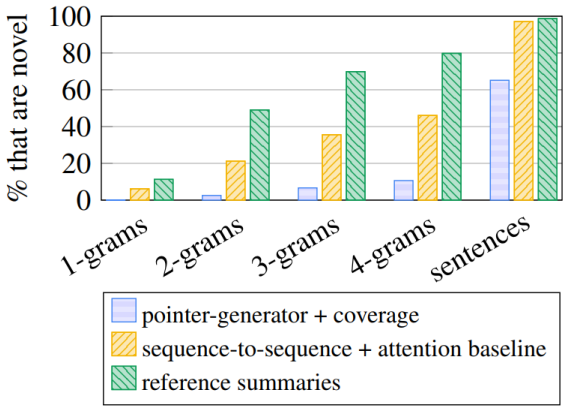


Figure 7.4: Novel n-grams published by See et al. [2017]



Model builds very abstractive summaries, but it is able to identify and include the important facts of summaries correctly as well.

7.5.4 ArXiv Abstract Generation Dataset Evaluation

Finally we present the evaluation of our novel corpus *ArXiv Abstract Generation* dataset (5.4), which is intended to be the next step to general purpose end-to-end trained summarization systems. Table 7.16 shows our baseline model reaching a good summarization performance on our randomly selected test set. Reviewing the outputs of our basline model we can clearly see the problem of repetitions in our sequence-to-sequence system. Compared to our model trained on *CNN/ DailyMail*, the model suffers from increased input length. While having a average input length of 688.4 words on the *CNN/ DailyMail* corpus it raises by nearly 400% to 3363.9 in our *ArXiv Abstract Generation* dataset.

To train our model, we used *TF-IDF Sentence Ranking* (3.2) to reduce the source inputs by to a maximum threshold of $3k$ words. Based on the results of our experiment shown in section 7.9, we could not see any decrease in performance using our *TF-IDF Sentence Ranking* approach, to shorten source sequences. Furthermore we applied *gradient checkpointing* on 4 of 6 layers on encoder and decoder side to reduce memory costs. For our training we used a *Tesla K80* GPU with $11.4GB$ of memory, which was barely able to handle our source inputs using *batch size words* of $3k$ and *batch size sentences* of 1 to process a single input sequence in one batch. This shows the large memory demand of the *Transformer Model* and proves the need for approaches to save memory during the training process in the summarization tasks. We use the *Transformer (6 Layers) + TF-IDF + gradient checkpointing (4 Layers)* model as our baseline. Moreover we conduct our inference experiment removing duplicated trigrams from generated summaries. In section 7.4.1 our evaluation showed the impact on the

CNN/ DailyMail dataset, increasing all *ROUGE* scores. The impact compared to our baseline on *ArXiv Abstract Generation* is significant. *ROUGE-2* increases more than 19%, *ROUGE-1* raises about 21% and grows more than 37%.

Table 7.16: Evaluation of *ArXiv Abstract Generation* model

Model	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
Baseline	24.81	7.28	16.62
Baseline + Removing Duplicated Trigrams	**30.03**	**8.67**	**22.84**

Presenting this baseline shows the technical possibilities of modern sequence-to-sequence systems based on the *Transformer Model*. Providing systems which are able to work on inputs containing more than $3k$ words is enough to summarize the text of most papers, which are currently published in research. Moreover in section 8.2 we show the technical possibilities to enhance this dataset to increase the amount of data or create various corpora of the *ArXiv Abstract Generation* dataset including different research subjects. This allows the reproduction our *Generalization Experiments* (6.2) on corpora containing large scale source sequences.

8 Conclusion

8.1 Review

To achieve the first goal of this thesis, to evaluate the performance of the *Transformer Model* on common summarization task datasets, we conducted experiments on the *Gigaword* and the *CNN / DailyMail* corpora. Comparing the results to state-of-the-art sequence-to-sequence approaches based on LSTMs showed that the *Transformer Model* is able to outperform existing systems on the shorter sequences contained in *Gigaword* corpus. Furthermore we showed with our experiments that the model provides a good baseline on longer sequence summarization including the *CNN / DailyMail* dataset. It was also able to outperform the baseline system of previous recurrent models. HHowever, most recent approaches in the form of novel coverage and copy mechanisms to avoid repetitions in the output summaries along with copying important facts directly from the source to the output sequence have a great impact on the results. These enhanced models performed better on longer sequences.
Presenting our novel results on experiments to explore the generalization ability of neural models on the summarization task showed surprising outcomes. Our baselines for both subjects in the *ArXiv Headline Generation* task demonstrate good results in producing titles of research papers. Comparing them gives first indications of how big the influence of large corpus is with respect to the model's performance. Furthermore presenting the results of our finetuning experiments showed how fast a trained model learns and improves its performance on different subjects. We did not expect a finetuned model to be able to outperform the model, which was initially trained on this subject's data from scratch. These results show once more the importance of large-scale corpora. Analyzing the learning process of finetuned models additionally showed the importance of finding the suitable point in time to stop the training, which results in models that are able to perform well on both corpora, the original and data used for the finetuning. The increase in performance after the first epoch already shows how fast modern neural networks, especially *Transformer Models* are able to learn and generalize on different topics.
Reviewing our third thesis goal, generating of a new corpus for very long input sequence summarization resulted in the dataset *ArXiv Abstract Generation* containing highly interesting data based on research papers, which allows summarization on different areas than the usual news article corpora. Besides the interesting contents

of our novel datasets, they provide new challenges to researchers because of longer sequences on source and target side. Furthermore we present a first baseline model generating summaries on *ArXiv Abstract Generation* to compare to further research.

8.2 Future Work

By improving the performance of translation system relentlessly in machine translation using the *Transformer Model*, we also think our models also have huge potential to outperform all previous sequence-to-sequence approaches in the summarization task. In this work we showed the first experiments using this new architecture giving researchers a good baseline including our configurations and experiences collected while performing this thesis.
Reviewing our experiments conducted on the *CNN/ DailyMail* dataset, we see good possibilities to improve the results by including a new copy and coverage mechanism in the current system. The approach presented by See et al. [2017] sets a possible direction to improve the performance. Moreover enhancements for the calculation of *self-attention* such as including information about the important facts in the source sequence possibly will improve future *Transformer Models*.
After presenting our new datasets for the *ArXiv Headline Generation* for Physics and Computer Science we see more perspectives in generalization experiments we conducted. For example arXiv offers the possibility of further subjects like *Economics* or *Statistics*, which could be used to create additional datasets, extending our experiments. Beyond that the aspects of corpus size can be explored more in detail by comparing very small datasets such as *Economics* which only contains around $42k$ examples to determine if this is enough to train summarization systems reaching similar scores.
Raising the size of sources and targets in summarization systems makes it more complicated to build systems reaching the same quality of outputs compared to systems working on shorter sequences. However reaching the goal of presenting a system generating business-reliable outputs, the feasibility to handle large inputs needs to be proved. One way to enhance the summarization quality is certainly the adoption of current models. On the other hand training on larger datasets will also ensure better performance. Using our methods, the *ArXiv Abstract Generation* dataset can be increased from currently around $91k$ examples to the complete size of arXiv database containing more than $1.5M$ research papers in march 2019. Furthermore the repository is constantly growing as new results in research are made every day.

Bibliography

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the web. In *IJCAI*, volume 7, pages 2670–2676, 2007.

John Bateman, Christian Matthiessen, Keizo Nanri, and Licheng Zeng. The re-use of linguistic resources across languages in multilingual generation components. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'91, pages 966–971, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN 1-55860-160-0. URL http://dl.acm.org/citation.cfm?id=1631552.1631605.

Yoshua Bengio and Yann LeCun, editors. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL https://iclr.cc/archive/www/doku.php%3Fid=iclr2015:accepted-main.html.

Ziqiang Cao, Furu Wei, Wenjie Li, and Sujian Li. Faithful to the original: Fact aware neural abstractive summarization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

Sumit Chopra, Michael Auli, and Alexander M Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, 2016.

Yue Dong. A survey on neural network-based summarization methods. *CoRR*, abs/1804.04589, 2018. URL http://arxiv.org/abs/1804.04589.

Nejmeddine Douma. Attention neural network-based abstractive summarization and headline generation. Master's thesis, KIT, 2018.

Harold P Edmundson. New methods in automatic extracting. *Journal of the ACM (JACM)*, 16(2):264–285, 1969.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Jeffrey L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2):195–225, Sep 1991. ISSN 1573-0565. doi: 10.1007/BF00114844. URL https://doi.org/10.1007/BF00114844.

Günes Erkan and Dragomir R Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of artificial intelligence research*, 22:457–479, 2004.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks, 2015.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

David Graff. English gigaword. 2003.

Andreas Griewank and Andrea Walther. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software (TOMS)*, 26(1):19–45, 2000.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1693–1701. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5945-teaching-machines-to-read-and-comprehend.pdf.

Martin Hilbert and Priscila López. The world's technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, 2011. ISSN 0036-8075. doi: 10.1126/science.1200970. URL http://science.sciencemag.org/content/332/6025/60.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Eduard Hovy, Chin-Yew Lin, et al. Automated text summarization in summarist. *Advances in automatic text summarization*, 14, 1999.

Karen Sparck Jones. Automatic summarising: factors and directions. *CoRR*, cmp-lg/9805011, 1998. URL http://arxiv.org/abs/cmp-lg/9805011.

Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997.

Wojciech Kryscinski, Romain Paulus, Caiming Xiong, and Richard Socher. Improving abstraction in text summarization. *CoRR*, abs/1808.07913, 2018. URL http://arxiv.org/abs/1808.07913.

Carl Lagoze and Herbert Van de Sompel. The open archives initiative: Building a low-barrier interoperability framework. In *Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '01, pages 54–62, New York, NY, USA, 2001. ACM. ISBN 1-58113-345-6. doi: 10.1145/379437.379449. URL http://doi.acm.org/10.1145/379437.379449.

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

Hans Peter Luhn et al. automatic derivation of information retrieval encodements from machine-readable texts. In *International Conference for Standards on a Common Language for Machine Searching and Translation (1959: Western Reserve University)*. International Business Machines Corp., Advanced Systems Development Division, 1959.

Inderjeet Mani. Summarization evaluation: An overview. 2001.

Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.

Ramesh Nallapati, Bing Xiang, and Bowen Zhou. Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023, 2016. URL http://arxiv.org/abs/1602.06023.

Paul Over, Hoa Dang, and Donna Harman. Duc in context. *Information Processing & Management*, 43(6):1506–1520, 2007.

Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *CoRR*, abs/1705.04304, 2017. URL http://arxiv.org/abs/1705.04304.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Martin Popel and Ondrej Bojar. Training tips for the transformer model. *CoRR*, abs/1804.00247, 2018. URL http://arxiv.org/abs/1804.00247.

D Raj Reddy et al. Speech understanding systems: A summary of results of the five-year research effort. department of computer science, 1977.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988a. ISBN 0-262-01097-6. URL http://dl.acm.org/citation.cfm?id=65669.104451.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988b.

Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *CoRR*, abs/1509.00685, 2015. URL http://arxiv.org/abs/1509.00685.

Gerard Salton and J Michael. Mcgill. 1983. *Introduction to modern information retrieval*, 1983.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368, 2017. URL http://arxiv.org/abs/1704.04368.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909, 2015. URL http://arxiv.org/abs/1508.07909.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon,

U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`.

A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339, March 1989. ISSN 0096-3518. doi: 10.1109/29.21701.

Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL `http://arxiv.org/abs/1609.08144`.