Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik 6
Prof. Dr.-Ing. Hermann Ney

Seminar
Selected Topics in Human Language Technology and Pattern Recognition
WS 2019/20

State-of-the-Art Approaches in Extractive Text Summarization

Daniel Maximilian Swoboda
daniel.maximilian.swoboda@rwth-aachen.de

Immatriculation Number 378972

February 27th, 2020

Supervisor: Jan Rosendahl, M.Sc.

Contents

List of Tables

List of Figures

1 Introduction

Humans have developed the ability to produce vast amounts of text data far beyond any single person's capabilty of comprehension. For the longest time, one way of increasing the accessibility of texts has been the use of human-made summaries. These are the result of a process by which length and content are reduced, whilst retaining the key information and essential statements. Their purpose is to make the most important pieces of content readily accessible in a less time consuming manner. [23]

Creating these summaries can be a hard task for humans, often requiring domain knowledge to produce good results. Even then, by virtue of their nature, human-made summaries are subjective. Different parts of a text are viewed differently by different people. Thus, making it even harder to determine what a good summary is and to create one that is commonly accepted. With the ever growing amount of information produced, relying on humans to produce the summaries has proven to be less desirable for economic reasons and the above mentioned subjectivity problems. [23]

To accomodate for human shortcomings in these areas, automatic summary systems are used ever more often. With their research history spanning back to the 1950s, they have long been an actively researched topic in the field of natural language processing. During this time, the technology changed from simple heuristics to approaches based on deep lexical parsing and advanced, expert knowledge driven statistical models. [23] Recently, machine learning approaches and especially those based on deep language models are the main competitiors when it comes to creating the most concise automatic summaries [10]. One of the main fields of research is extractive summarization, in which a subset of the sentences of an input document is selected and presented as the summary [1].

This work's objective is to introduce two recent, comparatively performing, state-of-the-art neural network models for extractive text summarization: BanditSum and BertSumExt. While these models lead to similar results, trained on the same data-set (CNN/DailyMail), they follow different approaches, highlighting the variety of attempted solutions for the text summarization problem. While BanditSum uses a reinforcement learning approach on a bidirectional recurrent neural network, BertSumExt uses a pre-trained Transformer-based architecture (BertSum) to create its summaries. [27, 12]

In the first section, the text summarization problem is formally introduced as well as the sub categories "Extractive Text Summarization" and "Abstractive Text Summarization". Then, a common classification of summarizers and the steps in the summarization process are introduced. Additionally, the concept of summarization evaluation is introduced followed by challenges of text summarization in general and extractive text summarization in particular, including the problems of coherence and consistency. Lastly, in the main part, two models BertSum and BanditSum are introduced and their results and evaluation methods are critically discussed.

2 Related Work

A broad overview of the field of text summarization prior to the introduction of neural network based approaches is given by Torres-Moreno [23], with a more recent survey on the state-of-the-art in 2017 conducted by Allahyari et al. [1]

Reinforcement learning approaches for extractive text summarization were suggested by Narayan et al. [15] and continued by Dong et al. [27], while RNN based approaches have been the repeated subject of study since 2014 with work by Chung et al.[5], Cheng and Lapata [4], and Nallapati et al. [14]. Recently BERT [6] based approaches have been gaining popularity, with models introduced by Bae et al. [2], Liu et al. [12], Zhang et al. [28] and Ming et al. [29] delivering state-of-the-art results.

Finally, neural network based approaches, focusing on model-design, data-sets and evaluation metrics were recently critically evaluated by Kryciski et al. [10]

3 Automatic Text Summarization

Automatic Text Summarization (ATS) can be defined as a problem, based on the definition from Torres-Moreno 2014 [23] the following way:

DEFINITION 1
Automatic Text Summarization Problem
Input: *Given an input document* $c = (s_1, ..., s_{N_c})$ *of length* $|c| = N_c$ *(in sentences), with sentences* $s_i = (w_1^i, ..., w_{|s_i|}^i) \in c$ *of length* $|s_i|$ *(in words) and words* $w_j^i \in s_i$.

Question: *Create a new document* c' *of length* $N_{c'}$ *such that* $N_{c'} < N_c$, $\tau = \frac{N_{c'}}{N_c}$ *is as small as possible and* c' *contains as much key information from* c *as possible.* c' *should also be understandable by human readers.*

From this, it can be derived that ATS is concerned with reducing a text in length whilst retaining conciseness, fluency as well as the most important pieces of information from the original text [23]. Additionally, the overall meaning of the input text should be preserved as well [1].

τ represents the compression rate of the summary compared to the input document. Torres-Moreno introduces the following definition for automatic text summarization, hinting optimal compression rates [23]:

DEFINITION 2
An automatic summary is a text generated by a software, that is coherent and contains a significant amount of relevant information from the source text. Its compression rate τ *is less than 33%. [23]*

While the minimum compression rate of 33% suggested by Torres-Moreno does not necessarily need to be obeyed, we can conclude from this that the goal should be to decrease the compression rate as far as possible. Doing so, the target document is reduced to a size which measurably decreases the reading time and thus assists humans in handling vast amounts of text.

Multiple additonal arguments in favor of ATS systems exist. These include increased effectiveness in indexing tasks (if the indexing is performed on the smaller, more concise summaries), or lack of subjective bias compared to human-made summaries. [23]

3.1 Categories of Automatic Text Summarizers

Historically one kind of classification focusing on the level of supposed text understanding the algorithms and models have, was used. It distinguishes between surface-level algorithms, intermediate-level algorithms and deep parsing algorithms. With the first two

focusing on linguistic features and the last one focusing on linguistic and semantic features of the text. [23] With the shift of methods methods from expert-knowledge based approaches, to neural network based approaches [10], it is debatable whether it still is applicable. One of the main problems is that neural-approaches do not use specific predetermined features or linguistic qualities, but rather learn to look for specific patterns through the training process. While they might still use linguistic features, it is often unclear and not pre-determined which of them are of relevance.

A common classification, that is still applicable, focuses on how summaries are created. According to this approach, ATS algorithms can be categorized into three families [23]:

1. **Extractive Text Summarization**, which selects a subset of sentences (or other text units) from the input text.

2. **Abstractive Text Summarization**, which attempts to create new coherent and concise sentences containing the key information from the input text.

3. **Sentence Compression**, which reduces the length of the sentences in the input text.

Sentence compression can be considered an attempt to move towards abstractive summarization, therefore it will not be handled specifically [23]. Thus, only two classes of summarizers need to be viewed: extractive and abstractive summarizers, which is the typical classfication in literature [1]. Since the focus of this work is extractive summarization, abstractive summarization is only briefly introduced for the sake of completeness.

3.1.1 Abstractive Text Summarization

Abstractive Text Summarization describes techniques that create summaries using the key information of the input text and building sentences around them to form a coherent new document of reduced size [23]. Therefore, they must first detect and source the important pieces from the input text and then piece them together, working similarly to the way humans create summaries [1]. Historically, the information extraction is followed by some variant of text generation algorithm used to create a coherent and readable text document, containing the key information [1]. Neural network based approaches might be trained end-to-end to directly produce a summary from an input document [12].

3.1.2 Extractive Text Summarization

Extractive summarizers operate by finding and selecting a subset of text units (e.g. sentences, segments of senteces or paragraphs) from the input document, which contain its key information. [23]. A common mode of operation is sentence level extraction which is also used by both of the two introduced models. [1].

While extractive summaries, unlike human-created summaries, solely consist of the sentences that are present in the original document, they are still performing better than most abstractive summarizers [1]. Therefore, they are still actively researched and can be considered the standard approach [27, 23]. This is in part attributed to the natural language generation problems that abstractive summarizers face [1]. With more advanced neural network models, this might soon change.

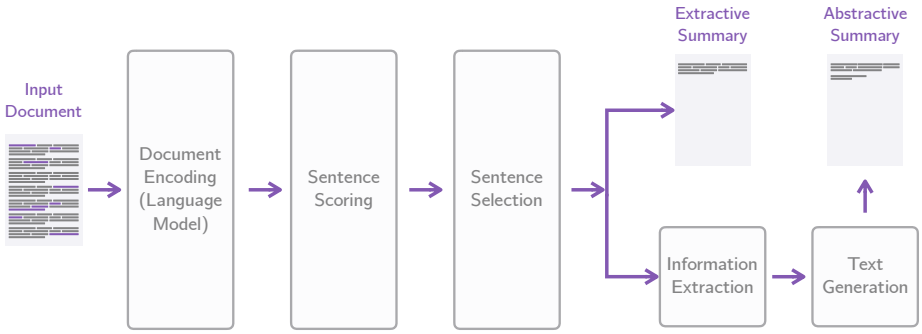

Figure 1: Overview of a common automatic text summarization architecture. The input document is passed through a preprocessing step, followed by a sentence scoring step assigning a score to each sentence of the document. An extractive summarizer selects a subset of sentences and presents this as the summary. By using information extraction on selected sentences, an abstractive summary can also be created from this.

However, extractive summarizers face several limitations of their own like coherence, readability and cohesion problems. To reslove these issues several techniques like anaphora[1] resolution algorithms have been developed in the past [23], with most of them not being used anymore in neural network based approaches.

3.2 Extractive Summarization Steps

In order to create summaries, a summarizer usually follows three sequential step from input document to final summary, as visualized in Figure 1 [1]:

- **Intermediate Representation:** In order to process the input documents, the text is first converted into an intermediate representation. Such a representation can be a word-level embedding in the form of a feature vector, which is common in ANN based approaches or a list of feature indicators. [1, 27, 12, 19]
- **Sentence Scoring:** Based on the intermediate represenation, the sentences are weighted. The weight of the sentence should be indicative of its importance in the context of the document. [1]
- **Summary Sentences Selection:** To finalize the summary, the sentences weights, which might be combined from different scores, are used to order sentences by their assumed importance. The algorithm then chooses from this selection. [1]

The text as a whole is processed by the algorithm, and encoded in a processable format (e.g. vector embeddings). These sentences are then weighted with a heuristic, that is either a human-made or learned by a machine learning algorithm on a large corpus of training data. Usually, the sentences with the highest scores are used to assemble the resulting summary. [23]

[1] Anaphoras are expressions that refer to preceeding expressions (e.g. the anaphora "they" would refer to a previously established subject).

This sequential procedure is also common amongst ANN based approaches, of which most follow an encoder-decoder structure [12]. The intermediate representation is the result of the encoder, the scoring the result of the decoder and the summary sentence selection is usually not performed directly by the network.

While sentence scoring is a leading approach, additional measures might being considered by different algoirhtms, since the most important sentences, might not be the most concise or coherent ones when combined. [1]

Figure 1 also highlights, how an extractive summarizer might be used in the preprocessing steps of an abstractive summarizer, by providing the sentences with the key information in a more digestable form for an information extraction algorithm. [23]

3.3 Evaluation

In order to measure the quality of an automatic summary it must be evaluated in the context of the input text, i.e. comparing it to through one or multiple measures to the input text or a pre-existing summary for that input [23]. This can be, similarly to summarization itself, a subjective task, especially when manually performed by humans. One of the reasons for this, is that no perfect summary exists [23] and that it is hard to determine which information must be included in a summary for it to be good, meaning that any and all evaluations can never be absolute measures of quality. In fact, evaluation of summaries can be considered an open problem with only partial solutions [23].

Two kinds of evaluations exist: intrinsic evaluations and extrinisic evaluations. While intrinisic evaluations consider the quality of the summary itself (in regards to readability, coherence, information content, etc.), extrinsic summaries consider the quality in the context of another system, like a Q&A system. [23] Since intrinsic evluations are more common and were used to evaluate both BertSum and BanditSum, the focus of this introduction lies on intrinisic approaches. A common test objective used in intrinsic evaluations is to compare the system summary, created by the summarizer, to one or more reference summaries, refered to as 'gold-standard summaries'. Intrinsic evaluations can be performed using humans or (semi-)automatic systems. [23]

3.3.1 Human Evaluation

Human evaluation is a common part of measuring the performance of summarization algorithms in conference, competitive and research environments [1]. Usually multiple people compare the input text or a gold-standard summary to an automatically generated summary and score them on several critera like strucutre, non-redundancy, correctness, coherence, etc. [23] For BanditSum and BertSum limited human evaluations have been performed [12, 27]

3.3.2 Automatic Evaluation

A variety of different approaches for automatic measurement of summary quality exist. The most common one of these metrics is ROUGE: the Recall Oriented Understudy for Gisting Evaluation [1]. ROUGE requires a gold-standard summary and compares the lexical similarity of the algorithmically created summary and the gold-standard one [1]. Two varitions of ROUGE are important to the evaluation of the methods introduced here:

- ROUGE-n: Measures the ratio of common n-grams[2], which appear both in the gold-standard summary and the summary to be evaluated, and the total number of n-grams in the gold-standard summary. [1]
- ROUGE-L: Is a measure of the longest common subsequence between the gold-standard summary and the summary that is subject to evaluation. In other words, ROUGE-L is a measure of biggest common n-gram. [1]

4 Challenges in Extractive Text Summarization

Although extractive text summarization is often considered to be easier to implement, robust and well performing, it faces several challenges that are proving difficult to resolve [23]. Among those challenges are coherence and cohesion which are strongly connected to the quality of a summary. [23] When it comes to machine learning algorithms there is the additional problem of suitable data-sets and exposure bias [10].

4.1 Cohesion

A summary, or any text, can be considered to have a good level of cohesion when it lacks both unresolved anaphoras and temporal references. Natural text, usually makes use of both of these features heavily, which are both hard to automatically identify and resolve. Since cohesion is located at a linguistical level, it could theoritically be resolved by using language-specific linguistical rules for identification of such references. Once identified they could be more easily resolved. However, natural language often suffers from wrong usage of anaphoras and other references, making it impossible to be resolved based on linguistic rules alone. Additionally, some references might not be backtrackable because of their distance or errornous use of language in the original text. [23]

4.2 Coherence

Coherence on the other side is semantical problem. It refers to the absence of logical contradictions and redundancy of sentences. Both aspects are important in different regards: The absence of logical contradictions is among the key criteria for a summary to be correct and understandable, as it would otherwise fail to convey the relevant information of the source document. The presence of redundant sentences, on the other hand, means that the summary is longer than it should be and contains bulk information that is not needed for comprehension. [23] In the worst case, a redundant sentence is included instead of one that would remove a logical contradiction.

4.3 Training Data-Sets

Modern, machine learning approaches (i.e. artificial neural network-based approaches) require a large amount of training data. To accommodate for this, several text data-sets for summarization tasks have been created. Most of these are based on news articles or other journalistic work like the CNN/DailyMail data-set [10]. Other sources include

[2] An n-gram is a part of a text, its unit can be sentences, words, multiple words combined, or even parts of words. If $n = 2$ and the unit is words, then an n-gram is any selection of 2 words that appear together in the document. [11]

forums with so called "TL;DR" (too long, didn't read) sections or the tutorial platform WikiHow. [10]

There are several problems with regards to the use of these data-sets. One is the lack of constraintness in the example summaries, leading to unconstrained summaries that are too ambigous. Additionally, news articles infuse a layout bias into the training, as news articles place most relevant information at the beginning. This information is then also included in the summary, which can lead the extractor to prefer sentences at the beginning, thus potentially delivering bad results on texts that are not in the respective layout. Finally, since the original data creators often are not organizations concerned with the creation of good training material for neural networks, but rather news papers and other publishing companies, their quality might not be perfect for the training process. In the case of extractive summarization this problem is even worse, since summaries in training sets like CNN/DailyMail are typically abstractive in nature. This makes it harder to train extractive models directly on it, often requiring some sort of heuristic to create an extractive summary from the original text and abstractive gold-standard summary. All of these factors combined degrade the quality of extractive summarizers trained on these data-sets. [10]

5 BanditSum - A Reinforcement Learning Approach

BanditSum is a reinforcement learning (RL) based approach to directly train an extractive BiRNN based model on abstractive summaries, introduced by Yue Dong et al. in November 2018.

Given an input document c of size N_c, BanditSum produces a summary c' of the fixed-size M, with M being a hyperparameter. Its RL-based training allows to omit the heuristical pre-processing step of extractive summary creation, which would not be possible in a supervised learning setup. To achieve this, extractive summarization is formulated as a contextual bandit problem and approached with policy gradient learning, allowing the use of abstractive summaries from the data-set directly as gold-standard summaries during training. [27]

5.1 Reinforcement Learning

Reinforcement learning (RL) refers to a category of machine learning methods that make use of evaluative feedback to train an agent. Unlike supervised learning, the deviation from the desired output is not used directly to update the training parameters. Instead, during the training process, the results are evaluated using a score R called reward. The objective for the agent is to maximize R. This is achieved by exploring the action space A and updating the parameters based on the received reward compared to previous rewards. The agent is controlled by a policy π which is usually modelled as a probability distribution over the action space.[22]

5.1.1 Multi-armed Bandit Problem

One formalization that can be used to describe RL problems is the multi-armed bandit problem. Also known as k-armed bandit problem, it can be formalized the following way: at each time-step t, one action a from an action space A with $|A| = k$ is selected by the agent. The selected action at time step t, denoted as A_t, yields a reward R_t based on

a specific probability distribution for each a. This probability distribution is not known to the agent, in order to maximize the reward over a large number of time steps, it is necessary to gather information about the expected reward $E[R_t|A_t = a]$ for all a. [22]

Choosing the action a with the highest known expected reward at t is referred to as knowledge exploitation, whereas taking a different action to gain more information on its expected reward is called exploration. While exploration can decrease the short term reward, it is most likely to increase the long term reward, therefore some level of exploration is required for an agent to perform well. [22]

The name *multi-armed bandit* is derived from the following analogy: Imagine a row of slot machines (one-armed bandits). Each has one arm to pull, representing one of the actions. For each machine the probability to hit the checkpot is different. The goal would be to make the most amount of money, which means exploiting the slot machine with the highest expected value as much as possible. Finding this machine requires exploration of the different slot machines available.

A strategy to always uses the action with the biggest reward is known as a greedy strategy. ϵ-Greedy, is a simple strategy that mainly focuses on on exploitation, exploring only with a small, predetermined probability ϵ. [22]

5.1.2 Contextual Bandits

One of the major limitations of the k-armed bandit formalization is that the set of actions is fixed. This can either lead to large action spaces (i.e. if one summary is one action, all potential summaries of the language would have to be included) or to the inability to perform good in all potential contexts if the action space is limited. The formalization is nonassociative, as it does not allow the agent to learn how to associate a scenario or state with an action. One extension, which tries to introduce context, thus improving action space size and performance, is called the contextual bandit problem [22].

In this new setting, a set of multiple action sets $A^* = \{A_1, A_2, ...\}$ exists. For all $a \in A_i$ a fixed expected reward that is unkown to the agent is set. However, at each t a different one of the A_i is presented to the agent. The setting is equivalent to a setting in which the agent is presented a different k-armed bandit task at each t. [22]

Under this reformulation it is now the agent's objective to maximize the total reward, which can only be achieved if the agent knows which action has the highest expected reward depending on the set of actions it is presented with. In a contextual bandit setting, an input is available to the agent which is called a context. The context c identifies each action set A_i. [22] This means, the objective can be formulated the following way: map each context to an action that maximizes the total reward. In other words, it needs to find a policy π that assigns the highest probabilities to the actions with the highest preference or expected value in each context c.

5.1.3 Policy Gradient Reinforcement Learning

Policy gradient methods (PG) are class of reinforcement learning approaches focusing on training the policy of the agent directly. In a PG setting, the policy π_Θ has a set of parameters Θ, which are trained using a scalar performance measure $V(\Theta)$. This performance measure, or value function, judges the current parameter settings based on the training objective. If the policy is a neural network, then Θ is equivalent to a vector of the connection weights of the network. [22]

π_Θ is accordingly defined as [22]:

$$\pi_\Theta(a|c) = P[A_t = a|C_t = c], \tag{1}$$

the probability of action a at time-step t given the context c. Since the policy is determined by its parameters, the probability of an action depends on the values of Θ. For simplicity here only the case with one context $\pi_\Theta(a)$ is considered, since this is also used by BanditSum during training.

The training process of PG methods uses a value function $V(\Theta)$ and a learning rate α to update the paramaters of the policy in the following way:

$$\Theta_{\text{new}} = \Theta + \alpha \nabla V(\Theta), \tag{2}$$

with the performance measure $V(\Theta)$ being arbitary but differentiable with respect to Θ. This expression means that each parameter is updated correspondingly to its ascent in the value function. In other words the contribution of a parameter to the current value determines how much its weight is increased. [22] Optimally, good values lead to bigger weight increases than bad values.

Maximizing the reward of the most probable actions for the agent to take is one common objective for RL tasks like contextual bandits and k-armed-bandit problems. In such cases $V(\Theta)$ is usually defined as the expected value of the reward function:

$$V(\Theta) = E[R(a)] = \sum_{a \in A} R(a)\pi(a), \tag{3}$$

as this is a measure of the current performance of the agent in such a setting.

The expected reward is equal to the sum of the reward of each action times its probability. When using this value function, the following gradient results if (2) is applied:

$$\nabla_\Theta V(\Theta) = \nabla_\Theta \sum_{a \in A} R(a)\pi(a) = \sum_{a \in A} R(a)\nabla_\Theta \pi(a). \tag{4}$$

Since R does not depend on Θ, R is constant in the gradient and therefore only the gradient of π in respect to Θ is relevant. For each a each parameter in Θ is updated by the product of its ascent and the reward it yielded. [22] This way, the parameters are changed depending on the reward and how much they contributed to it. High reward actions increase the weight more than low reward actions. If the gradient of a value function like this is used to update the parameters, it will change them in such a way that the reward is maximized along the ascent.

Usually during training, the value function is approximated using a set of samples. The resulting sampled gradient does not need to match the actual gradient but rather just needs to be proportional to it.

5.2 Modeling Extractive Summarization as a Contextual Bandit

BanditSum uses a contextual bandit to model the extractive summarization task. Because of this formalization, a policy gradient approach can be used to train a policy for summarization. In the case of of BanditSum the policy is a recurrent neural network (RNN) that is trained using a ROUGE-based reward. This particular setup is meant to create sentence-level single-document summaries of a small, fixed size M. [27]

5.2.1 Action-Space Modeling

The contextual bandit model requires an action space A^*, which depends on the context space C since the context dictates which actions are available to the agent in each $c \in C$. In the case of text summarization, one of the possible context spaces is the infinite set of possible sentence-based text documents. This approach is used by BANDITSUM [27]. Intuitively, this makes sense because for each document, different sentences are more important than others. Additionally it can be argued that if the context is a document, then similar documents which might require similar actions to create concise summaries, have a similar context and should be treated similarly by the agent.

Based on this, there are multiple ways of defining A^*. One option would be to make each action equal to appending a corresponding sentence to the resulting summary. That way, running the agent for M timesteps yields a M-sized summary. However, BANDITSUM uses a different design where each action corresponds to a M-sized summary, so that the agent only performs one timestep per document and summary. This can be formalized the following way: Each context/document $c \in C$ is a tuple $c = (s_1, s_2, ..., s_{N_c})$ of length N_c such that each element of the tuple corresponds to one sentence of c in the original order. An action then is a M-tuple of indices $a = (a_1, a_2, ..., a_M)$ referencing to sentences of c. Based on the action chosen by the agent, a summary can be recreated by extracting the sentences that were selected through a's indices. [27]

Suppose $M = 3$ and $c = (s_1, s_2, s_3, s_4, s_5, s_6)$, then the action $a_1 = (2, 5, 6)$ yielding a summary s_2, s_5 and s_6 is one of the available actions $a_i \in A$ to the BANDITSUM agent in the context c. For each document c the objective, in accordance to the contextual bandit formalism, is then to select the summary a_i such that the reward $R(a)$ is maximized.

5.2.2 Policy Network

BANDITSUM uses an encoder-decoder structure to determine, based on the input document c, affinity values q_i^c for each sentence s_i [27]. Through training, the network can learn to create embeddings that make it possible to extract certain features from the input text that are helpful in calculating representative affinity values.

The network's encoding consists of several steps, as shown in Figure 2. In a first step, for each sentence $s_i = (w_1, w_2, ..., w_{|s_i|})$ of c a word-level initial embedding is created using a pre-trained GloVe encoder. The GloVe layer is pretrained and maps input words w_j^i to a feature vector $\text{gl}_j^i = \text{GloVe}(w_j^i)$. This transformation is trained using unsupervised learning on a large corpus and has the intention to put words with similar meanings close to each other in their vector representation. A representation of the meaning of the word is encoded in its feature vector [20].

In the next step, the resulting sequence of GloVe vectors $\text{gl}_i = (\text{gl}_2^i, \text{gl}_2^i, ..., \text{gl}_{|s_i|}^i)$ is passed through a word-level bidirectional RNN (BiRNN) $f_{\text{enc\_w}}$ with long short-term memory (LSTM) [27]. Applying $f_{\text{enc\_w}}$ leads to a new sequence of vector word representations. This way features of the neighboring words from both sides within the sentence are added to the word representations, creating a contextual representation of the words within the sentence. For all $f_{\text{enc\_w}}(\text{gl}_j^i)$ of sentence s_i the results are subsequently averaged:

$$s_i^{\text{enc}} = \frac{1}{|s_i|} \sum_{j=1}^{|s_i|} f_{\text{enc\_w}}(\text{gl}_j^i),$$

leading to a representation of the sentence s_{i_enc} based on its word features.

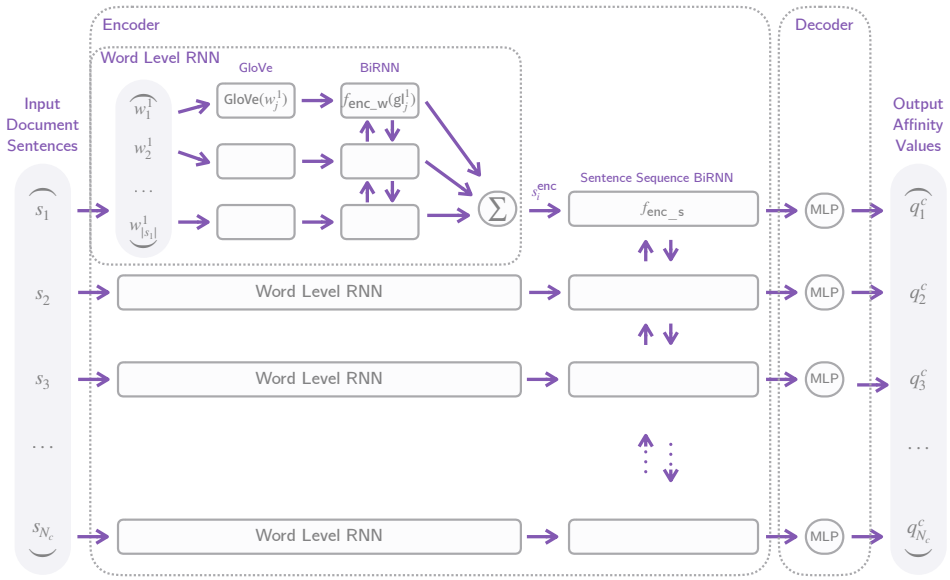

Figure 2: The neural network used to determine the sentence affinity in BanditSum. Sentences are first passed through a word embedding step and averaged to create sentence representations. These are then passed through a multi-layer sentence level BiRNN, before finally being passed through an MLP for evaluation.

In a final encoding step, the sentence feature vectors $(s_1^{\text{enc}}, s_2^{\text{enc}}, ..., s_{N_c}^{\text{enc}})$ of all sentences s_i of a document are passed in sequence through another BiRNN $f_{\text{enc\_s}}$ [27]. This step adds features of the neighboring sentences from both sides, creating a contextual represenation of the sentences within the document.

The resulting sentence feature vectors $f_{\text{enc\_s}}(s_1^{\text{enc}}, s_2^{\text{enc}}, ..., s_{N_c}^{\text{enc}})$ are now supposed to include contextual information of their words and the context of the sentence within the document. This final representation is now passed through the decoder d which yields the affinity scores $q_i^c = d(f_{\text{enc\_s}}(s_1^{\text{enc}}, s_2^{\text{enc}}, ..., s_{N_c}^{\text{enc}})_i)$. The decoder consists of a multi-layered perceptron (MLP) which, based on the final embeddings, calculates the affinity values from the range $[0, 1]$. [27]

5.2.3 Policy Model

The policy of BanditSum assigns a real-valued score, called 'probability' to each of the possible M sized summaries a of the document c. This score of a represents how likely it is for the agent to choose a. [27] However, since it does not reflect a real probability distribution, with the sum being not equal to 1 in most cases, it will be called score in this work.

Ultimately the goal is for the policy to assign the highest score to the summary that leads to the highest reward, and is therefore considered the best by the chosen metric. To accomplish this, the policy needs to be able to learn based on the input document and the sentences of the summary, which of the sentences are to be included in a summary. Thus, each of the sentences need to be assigned some weight that influences the score.

In BanditSum this is implemented by using a multi-layered RNN that is fed with the document c and outputs a vector q^c of dimension N_c of so called sentence affinities for the

document. These are a metric that for each sentence, based on its feature representation, stands for its likeliness to be included in the summary. The design does not normalize the affinities, however each affinity [27]

Sentence affinities are then further processed to yield a score on a summary level. Since BANDITSUM summaries are of equal size M, this can be achieved by sampling the affinities of each sentence in the summary [27]. Instead of a simple normalizing and sampling procedure for each a where the normalized affinities are multiplied with each other, such that for a high score all affinities need to be high:

$$\prod_{j=1}^{M} \frac{q_{a_j}^c}{\sum_i q_i^c},$$

a more complex approach is used by BANDITSUM. This is because the simple approach would not take into account exploration which is required for successful reinforcement learning. Thus, a small proability ϵ is introduced that, similarly to the ϵ-greedy strategy, leads to an adequate amount of exploration during training. [27] This results in the following sampling method:

$$\pi_\Theta(a|c) = \prod_{j=1}^{M} \frac{\epsilon}{N_c - j + 1} + \frac{(1-\epsilon) q_{a_j}^c}{\sum_{i=1}^{N_c} q_i^c - \sum_{k=1}^{j-1} q_k^c}. \tag{5}$$

This way each summary a is guaranteed a small score. Additionally this sampling procedure tries to mitigate the effects of the layout-bias problem of machine learning based extractive text summarizers. To achieve this, the weight of the affinity of sentences that occur later in the document is increased proportionally to their position in the expression, which is achieved by decreasing the normalizing denominator by the affinities of the previous sentences [27]. The resulting score distribution however is not normalized, this means that the sum of the scores over all summaries is not equal to 1.

5.2.4 Reward Modeling

The reward is used to tune the parameters of the policy such that the actions yielding a big reward are selected with a high score using policy gradient learning. Therefore, the reward should be measured by the quality of the resulting summary in a text summarization task. Since actions in BANDITSUM are equal to summaries and the reward function evaluates a single action a, the resulting reward is a measure of the summary. The reward needs to be determined automatically, thus only variants of automatic evaluation can be used.

BANDITSUM uses a ROUGE-based reward, combining multiple ROUGE measures together. The summary that corresponds to a is compared to a gold standard summary g from the training data-set that belongs to the document c [27]:

$$R(a, g) = \frac{1}{3}(\text{ROUGE-1}_f(a, g) + \text{ROUGE-2}_f(a, g) + \text{ROUGE-L}_f(a, g)), \tag{6}$$

which is the arithmetic average of the measures ROUGE-1_f, ROUGE-2_f and ROUGE-L_f. ROUGE-1 and ROUGE-2, which work on 1-grams and 2-grams respectively, as well as ROUGE-L have been shown to be a somewhat conclusive measure for the recall between an automatic and a gold standard summary. ROUGE-1 was shown to work comparatively well on short summaries, while ROUGE-2 performs better evaluating single-document

summaries and ROUGE-L performs acceptable in both tasks. [11] Since BanditSum is a single document summarizer, that is meant to create short summaries, the combination of ROUGE-1 and ROUGE-2 promises a reasonable approach within the limits of automatic evaluation.

Precision, which measures the amount of n-grams that are not included in the gold standard summary, is also determined for each of the used ROUGE variants. Recall and precision are then combined to the f-measure using the harmonic mean [21].

The reward design used by BanditSum means that its policy is trained to directly improve the ROUGE-1, ROUGE-2 and ROUGE-L measures of its summaries which are also used in the performance evaluation of extractive ATS.

5.2.5 Search

To perform search – that is to find the best summary according to the policy π – BanditSum first samples all potential summaries a_i for the given c and M, i.e. every M sized subset of c. It then uses the trained policy to determine the probability $\pi_\Theta(a_i|c)$ for each summary. Based on this, the summary a_j with the highest probability is then greedily chosen and reconstructed using the sentence indices of a_j.

5.2.6 Value Function Approximation and Variance Reduction

With the reward and policy defined the value function $V(\Theta)$ can now be determined and used for training:

$$V(\Theta) = \sum_{a \in A} R(a, g)\pi_\Theta(a|c), \tag{7}$$

with the implementation used by BanditSum only determining the value for a given c. This leads to the following gradient according to (4):

$$\begin{aligned} \nabla_\Theta V(\Theta) &= \sum_{a \in A} R(a, g)\nabla_\Theta \pi_\Theta(a|c) && (8)\\ &= \sum_{a \in A} R(a, g)\frac{\nabla_\Theta \pi_\Theta(a|c)}{\pi_\Theta(a|c)}\pi_\Theta(a|c) && (9)\\ &= \sum_{a \in A} R(a, g)\nabla_\Theta(\log \pi_\Theta(a|c))\pi_\Theta(a|c), && (10) \end{aligned}$$

where the expression $\nabla_\Theta(\log \pi_\Theta(a|c))$ is normalizing the gradient based on its probability. $\nabla_\Theta V(\Theta)$ is also differentiable since the π_Θ is differentiable in regards to Θ. In fact, just the network itself needs to be differentiable in this case, which it is by design. This is similar to the formula used by the REINFORCE algorithm. [22, 27, 25]

Since the entire probability distribution of π_Θ is usually not known, it is sampled B times leading to the following:

$$\nabla_\Theta V(\Theta) \approx \sum_{i=1}^{B}(R(a_i, g) - b)\nabla_\Theta(\log \pi_\Theta(a_i|c)), \tag{11}$$

which is a uniform sample of the value gradient of B samples $(a_1, a_2, ..., a_B)$ in respect to Θ for a given document c. Additionally a baseline b is introduced that is meant to reduce the variance, which is high in the REINFORCE formulation. The baseline for a given c is equal to the reward of the summary with the highest reward following the current policy. This, eventually, is the value function gradient used by BanditSum. [27]

6 BERTSum - Pre-Trained Encoder for Text Summarization

BertSum is a pre-training based encoder for text summarization based on a modified Bert architecture introduced by Yang Liu et al. in November 2019 together with the extractive summarization model BertSumExt.

Given document c of size N_c and summary size m, it produces a summary c' of size m. Using a specialized variant of the general purpose Bert encoder, it creates vector representations of sentences with document-level context. The resulting encodings are then passed through a simple decoding layer to create a sentence scoring. c' is then constructed by picking the top m sentences.

While this approach does not specifically address any of the problems tackled by BanditSum, it performs on a similar state-of-the-art level [12]. A possible contribution to its performance is the use of Bert as a basis for the encoder, which recently surged in popularity amongst the research community, delivering good results.

6.1 BERT

The representation of input text and its context is an important part of neural network-based approaches to text summarization. One method used to achieve representation of context in natural language processing (NLP) is representation learning. [3] This is a method by which a representation of the input data is created by a neural network, called encoder, which is pre-trained or part of the end-to-end training of the entire model.

Encoder based architectures are used by most neural NLP approaches, which creates the possibility for using a shared encoder model in multiple tasks. Such an encoder can then be pre-trained, in an attempt to reduce the amount of training required for the rest of the model. Bert (Bidirectional Encoder Representations from Transformers) is a pre-trained encoder, designed to be used in a variety of so called downstream tasks, all of which require learned input representations. [6] Since its release, Bert proved its capabilites as a state-of-the-art encoder providing a better representation of the semantical relation of words and sentence-pairs. Bert is used as an integral part of Google's search engine. [17]

6.1.1 Word Embeddings and Neural Language Models

The prospect of representation learning is to transform the input data in a way that makes it easier to extract the information that is required to perform a processing task on it. In natural language processing these representations are called word embeddings. [3] Mathematically speaking, the goal is to create a sequence of continous valued representations $(\hat{w_1}, \hat{w_2}, ..., \hat{w_{|s_i|}})$ for the symbol representations (e.g. words of a sentence) $s_i = (w_1, w_2, ..., w_{|s_i|})$ by applying an encoder function f_{enc}. The resulting vector $f_{\text{enc}}(s_i)$ is of a lower dimensionality, creating the need for abstraction and feature extraction in order to preserve the meaningful information of the input. This leads to more useful representations that contain the relevant information in a reduced and processable form. [3]

A neural network can be trained to create word embeddings. While there are multiple architectures that can be used for such a task, the same basic principle is shared amongst these. In a first step the input is compressed to a vector of the target embedding size by passing it through multiple layers with hidden dimensionalities different from the input.

This setup leads to a transformation of the input symbol representation. The encoder can either create representations of just the word by itself, or it can include contextual information, like surrounding words, leading to more complex and rich representations.

In order to get meaningful representations a training task is required that ensures that the important information can be reconstructed from the embedding. To accomplish this, in simple cases a decoder f_{dec} is used that is trained together with the encoder on an auto-encoder task. The encoder-decoder setup is meant to first compress the input and then recreate it from the compressed form. In the optimal case $f_{\text{dec}}(f_{\text{enc}}(s_i)) = s_i$. In practice the networks are trained until a certain, minimal, level of loss is reached. [9]

If the training on the encoder is not based on the final task directly, it is called pre-training, since the network needs to be fine-tuned for the task specific use.[9] The encoder can be called a language model if it is trained on a context rich, large corpus, leading to an understanding of words in different contexts. [12]

There are multiple widely used word embeddings like GloVe (which is used by BanditSum) and word2vec. These are available in a pre-trained form that can be used for new applications that need an encoder. Both are trained in a way that during training, the context of a word is considered, e.g. by providing whole sentences in which the words occur during training. That way the semantic relations between words are attempted to be contained in the mathematical representation of the words. [20, 13]

6.1.2 Pre-training of Language Models

Pre-training of language models contributed to a variety of different natural language processing tasks. These include sentence level tasks like inference and paraphrasing as well as token level tasks like named unit recognition and question answering. [6] Commonly, pre-trained models are used as encoders in networks designed for these tasks [12].

Usually pre-training is accomplished through unsupervised training on large corpora of text that prepares the model for later fine-tuning [7, 12]. It establishes an initialization point that leads to a reduction in the amount of training required and a restriction of values to particular regions. This adds robustness and decreases the time needed to train the complete model on the downstream tasks. [7, 6]

Two different approaches for using pre-trained models in downstream tasks can be identified:

- **Feature-based:** Word embeddings resulting from the pre-trained model are directly included as (additional) features in the downstream task. The model is not modified anymore in this case. [6]

- **Fine-tuning:** The pre-trained model is included into the network for the downstream task and is used as part of the encoder. Its parameters are trained together with the other parameters. [6]

Such models have been mainly used for language understanding tasks, but have recently also found their way into text generation problems. This can be attributed to their contribution to the recent performance gains in language understanding. [12]

A problem with current models is that they are completely unidirectional or concatenate two unidirectional embeddings. For such models, only one sided context of the input words is relevant to the representations but not an actual combination of both. This severly limits the possible performance of pre-trained models. [6]

6.1.3 Masked Language Model and Next Sentence Prediction

One possibility to further increase the performance of pre-trained language models for fine-tuning is to replace unidirectional models with bidirectional models. This leads to both left-hand and right-hand context having relevance on the word represenation. Bidirectionality can be achieved by using training objectives that require both sides of the context of a word or by training on sentence context tasks. It is not trivial to perform the training since normal left-to-right model trainings would allow each word to indirectly see itself in a multi-layered neural network. [6]

Two of the objectives that can be used to perform this kind of training are the masked language model objective (MLM) and the next sentence prediction objective (NSP):

- **Masked Language Model:** For this training objective, the encoder network to be trained is extended with a softmax decoder over the training vocabulary. In a first step, a certain percentage of input words is masked. The masked input sequence is then fed into the encoder, resulting in embeddings for the mask words, which contain the left-and-right context information. These are then fed into the decoder in order to determine the original word based on the features contained in the mask embedding. Both encoder and decoder are trained simultaneously. [6]

- **Next Sentence Prediction:** This training objective consists of a more simple decoder with two possible outputs. The encoder is fed two sentences in sequence. In 50% of the cases, the second sentence is the actual sentence that follows the first sentence in the document. The other 50% a random sentence taken from the same document is used. Based on this input, the decoder decides, using the sentence encodings, whether the first sentence is followed by the second one or not. This leads to a representation of sentence relations within the embedding. [6]

In order to implement these tasks, the encoder network architecture needs to support the concurrent processing of left and right contexts of an input sequence. One potential way of implementing this is the use of a BiRNN like in BanditSum. Another, more sophisticated, option is a Transformer.

6.1.4 Transformers and Self Attention

Transformers [24] are a common attention-based model architecture that is widely used in NLP tasks [6]. Similarly to BanditSum's architecture, the Transformer also uses an encoder-decoder structure, with a design based on the needs of language translation models. [24] Thus, its encoder is a suitable design for use in language modeling tasks.

Based on the original transformer architecture, the encoder can be seperated from the decoder and its learned representations can be used for different downstream tasks other than machine translation [6]. The encoder operates as depicted in Figure 3: In a first step each token w_j (e.g. word) of a sequence s_i (e.g. sentence) is passed through an initial embedding network [24]. This can either be end-to-end trained with the rest of the network, or be a pre-trained embedding like GloVe. Transformer encoders in their original implementation use an embedding of the dimensionality 512, leading to a $|s_i| \times 512$ matrix V_{s_i} (with 'V' standing for value) that represents the input sentence, where $|s_i|$ is the number of words in s_i. A learned positional embedding is then added to each of the word representations:

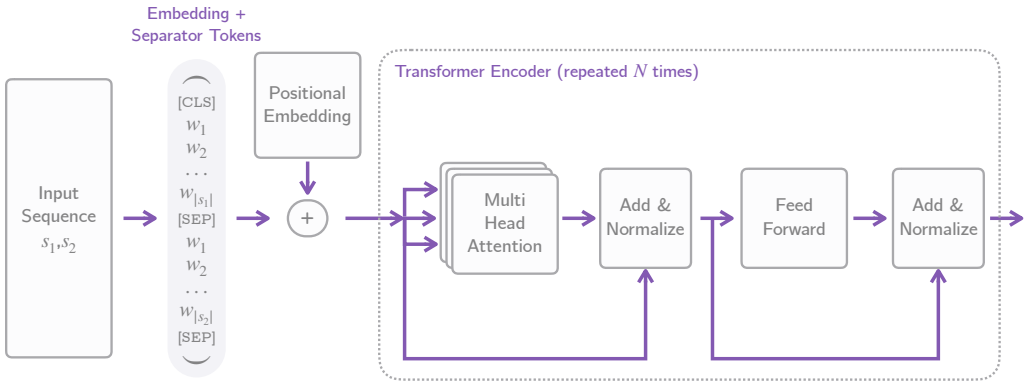

Figure 3: Encoder of the Transformer architecture. Input embeddings are passed through several identicial layers consisting of a multi-head attention and a feed-forward layer.

$$V = V_{s_i} + V_{\text{pos}_{|s_i|}}.$$

V is then fed through several identical layers. Each of the layers consists of a so called multi-head self-attention operation and a simple feed forward network. The multi-head attention performs a row of several self-attention operations (each is referred to as a head) with different input weights that are subsequently concatenated:

$$f_{\text{att}}(V) = \text{softmax}\left(\frac{VV^T}{\sqrt{512}}\right)V. \tag{12}$$

VV^T determines the attention of the different words in regards to the other words in the sentence, divided by the scaling factor $\sqrt{512}$ which is based on the dimensionality of the token representations. This is necessary since the dot-product attention leads to large growths in magnitude. The resulting attention is then applied on the original representations, adding context information on the word in its sentence. [24]

Different heads, with different, learned weights are concatenated and averaged:

$$\sum_{i \in |\text{heads}|} \frac{\text{weight}_i}{|\text{heads}|} f_{\text{att}}(V), \tag{13}$$

in order to combine them. The resulting matrix is passed through a feed-forward network f_{FFN} that acts as a position based normalizer. After both steps, the result is applied to the input of the operation in the form of a residual connection. The process of using self-attention and a feed-forward network is repeated several times, eventually leading to the target word representations of a sentence. [24]

One of the major advantages of this design is the high potential for parallelization, since the final embeddings for each word of a sentence are determined simultaneously. Furthermore, this process can be applied to a batch of sentences in parallel as well. [24]

6.1.5 BERT Architecture

BERT is an encoder architecture for a pre-training-fine-tuning approach, i.e. the BERT encoder is first pre-trained to be then fine-tuned with additional encoder and decoder

layers for a downstream task. The architecture uses several stacked Transformer encoder layers to create both word and sentence embeddings that are meant to convey left-and-right context as well as sentence level context. It uses the MLM (masked language model) and NSP (next sentence prediction) training tasks during pre-training to achieve this level of contextual embedding. [6, 12]

BERT uses a special formatting for its input sequences, that can carry up to two sentences:

`[CLS]` $\hat{s_1}$ `[SEP]` $\hat{s_2}$ `[SEP]`

$\hat{s_i}$ is the sequence of WordPiece word embeddings from the input sentence s_i, while `[CLS]` and `[SEP]` are special tokens. This modular two sentence input sequence design allows for Q&A tasks or single sentence applications. The initial WordPiece embeddings are either whole words or parts of words marked with special characters that indicates their connection to the previous word token [26, 6]

The input sequence starts with a special token `[CLS]` that indicates the beginning of the sentence, the final embedding of which can be interpreted as being the representation of the entire sequence. This is followed by the WordPiece tokens of the words of the input sentences s_1 and s_2. Each input sentence includes another special seperator token `[SEP]` at the end. This inital embedding is concatenated with segment embeddings E_a or E_b indicating which token belongs to which sentence of the input. [6]

Position embeddings E_i mark the position of each token in the input sequence, which is also used in the standard transformer model [24, 6]. This input sequence is the input matrix V of the Transformer.

In order to pre-train BERT two different decoders are used for the MLM and NSP tasks. The MLM task uses a decoder similar to the decoder used in the original transformer implementation, that includes a softmax over the vocabulary as its final output. 15% of input tokens are masked using a special mask token. To avoid a mismatch between the pre-training and fine-tuning where the mask token is not used anymore, 10% of the time the mask is replaced with a random word from the vocabulary and 10% of the time the original word is used. [6]

The NSP task is pre-trained by using a simple decoder that acts as a binary decision maker. The design of the input sequence, consisting of up to two sentences, can now be used here, with the second sentence being either the correct successor or not. The decoder is then trained together with the encoder to decide which is the correct one. [6]

6.1.6 Fine Tuning

BERT is fine-tuned for a downstream task by combining the pre-trained encoder and additional layers suitable for the downstream task into one network and training the complete network on the downstream task end-to-end. In this training all the parameters are changeable, including the pre-trained ones. Because the encoder is designed for the BERT-specific input sequence, the input must be formatted accordingly, thus being limited either one single sentence or a two sentence pair.

Fine-tuning of a BERT based network was shown to be comparatively inexpensive for a variety of NLP tasks by the authors. This leads to shorter training times and faster convergence of the downstream task network. [6]

6.2 Applying a BERT-like Architecture to Summarization Tasks

BertSum extends the architecture of Bert to create a pre-trained encoder that is suitable for both abstractive and extractive summarization tasks. Bert is not a wide-coverage model and only supports a maximum of two sentences in input. Therefore, several modifications are required to make Bert suitable for text summarization tasks, which requires wide-coverage, i.e. document level, context representations and multiple sentence inputs.

6.2.1 Architectural Modifications

The modifications of BertSum's architecture compared to the original one of Bert fall into two categories: changes to the input sequence representation and encoder structure modifications. Input sequences for BertSum consist of documents $c = (s_1, s_2, ..., s_{N_c})$, with each s_i being a sequence of words $(w_1, w_2, ..., w_{|s_i|})$. Since the original Bert architecture is limited to a maximum of two sentences in a sentence pair format, the input format is modified the following way, to accomodate for multiple sentences [12]:

`[CLS]` $\hat{s_1}$ `[SEP]` `[CLS]` $\hat{s_2}$ `[SEP]` ... `[CLS]` $\hat{s_{N_c}}$ `[SEP]`

Similar to Bert the words of the sentences s_i are first encoded using WordPiece resulting in $\hat{s_i}$. The `[CLS]` tokens are now used to signalize the start of a new sentence. While previously the encoding of the `[CLS]` token was a representation of the entire input sequence, each `[CLS]` token now is a representation of the sentence it is followed by in the context of the document. With these modifications it is now possible to perform document level tasks on the sentence representations. Initial embeddings of each of the sentences' words, together with the `[SEP]` and `[CLS]` tokens, represent the input sequence matrix V. [12]

The position embeddings E_a and E_b used by Bert to mark whether a token belongs to the first or second sentence of the input document are also used differently. In BertSum E_a is used to mark odd-indexed sentences in the input sequence, while E_b marks even-indexed sentences. Thus, every second sentence gets an E_b position embedding. This helps with differentiating tokens on a sentence level. [12]

6.2.2 BERTSumExt - Extractive Decoder

In order to perform the extractive summarization task, the BertSum encoder is extended by additional Transformer layers, which are responsible for performing a final set of sentence level feature extractions on each embedding $\texttt{[CLS]}_i^e$ of the `[CLS]`-tokens:

$(\texttt{[CLS]}_1^e, \texttt{[CLS]}_2^e, ..., \texttt{[CLS]}_{N_c}^e)$

This step adds additional document-level context between the sentence representations, with the resulting, final, embeddings being the output of the encoder of BertSumExt, as can be seen in Figure 4. The output is then fed through a sigmoid classification layer, that assigns a real valued score $y_i \in [0, 1]$ to each sentence, which is meant to be a score of the representativeness of a sentence in regards to its source document based on their cross document attention. To assemble the summary the m sentences with the highest values are chosen ordered descending by their values. [12]

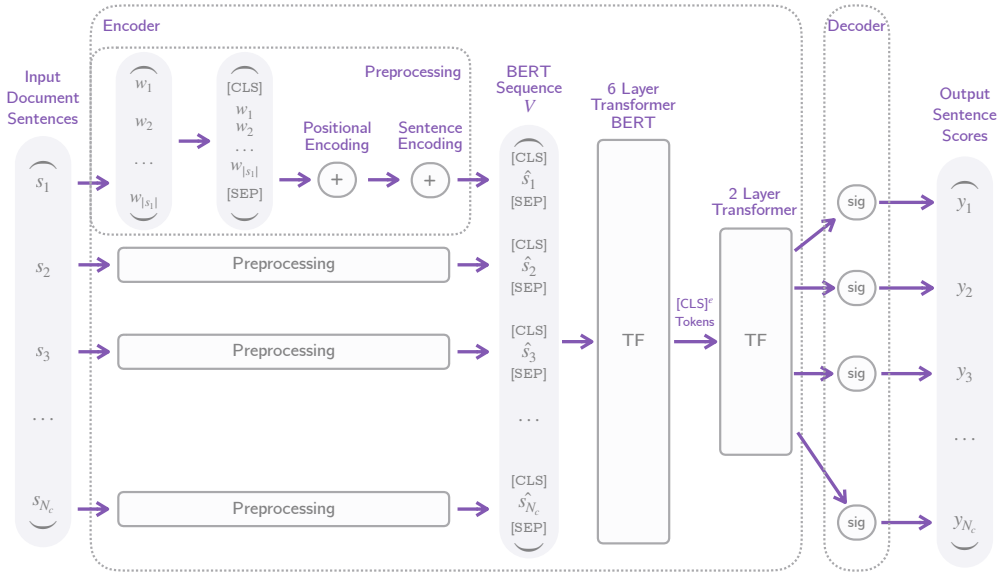

Figure 4: BertSumExt architecture. The input sentences are preprocessed and tokens for sentence discrimination are inserted. The input sequence is then fed through the Bert-style Transformer-based network. For extractive text summarization the sentence representations are fed through additional Transformer layers and a simple sigmoid decoder that yields a score for each sentence.

One problem with this approach, however, is that redundancy might be high, since the sentences that are most indicative of the document, might contain similar information. Therefore a process of "Trigram Blocking" is employed. This means, that based on the already selected sentences, selection candidates are skipped if they contain the same tri-gram as the already selected ones. [18, 12] Suppose for a given document c, BertSumExt yields scores $(0.3, 0.5, 0.1, 0.15, 0.63, 0.23)$, then the sentences s_5, s_2 and s_1 are chosen. However, if s_1 has an overlapping tri-gram with s_2, then instead of s_1, s_6 is chosen, if s_6 does not contain an overlapping tri-gram itself.

7 Results

Both BanditSum and BertSumExt have been trained and evaluated by the authors extensively on several data-sets. Human and automatic evaluation based on the ROUGE-F1 measure were used to test the performance of the models. The results of the automatic evaluation indicate state-of-the-art performance of both extractors with BertSumExt outperforming BanditSum. [27, 12]

7.1 Training

Both summarizers were trained on a variety of different data-sets, one of which being the CNN/DailyMail data-set (1) by Herman et al. [8], which is used commonly in benchmarks of text summarization tasks [27, 12]. The parameters $M = m$ were set to 3, producing sentence level summaries of size 3.

Table 1: Properties of the CNN/Daily Mail Data-Set

	CNN	DailyMail	Combined
No. Training Documents	90,266	196,961	287,227
No. Validation Documents	1,220	12,148	13,368
No. Test Documents	1,093	10,397	11,490
Average Document Length (sentences)	33.98	29.33	30.70
Average Summary Length (sentences)	3.59	3.86	3.77

BertSumExt was fine-tuned for 50,000 steps on three GTX1080Ti GPUs. This extends the pre-training of Bert on 3.3 million sentences from BookCorpus and the English Wikipedia. The authors did not mention how long the fine-tuning took in their setup. BanditSum on the other hand was trained on a single Titan Xp GPU for three epochs on the same data-set, with one epoch taking roughly 25.5 hours to complete. [27] [12]

BanditSum's training objective was to maximize its reward, and therefore ROUGE scores, according to the policy gradient scheme employed [27]. This leads to a model that is inherently biased on creating summaries that are lexically close to the gold standard summaries used during training. While this led to success, especially on the automatic evaluations where exactly this metric is used, it does not necessarily constitute to what humans would consider a good summary as it is limited to the style of summary it was exposed to [23]. Additionally its training setup could lead to a combinatorical explosion in the amount of potential summaries it considers, which is equal to the amount of 3-sized subsets of the input text.

BertSumExt was trained on the loss compared to a heuristically created extractive summary, using a greedy oracle, that picks the top 3 sentences, such that their ROUGE-2 score compared to the gold standard summary of the input text is maximized. [12] Such a heuristically created extractive summary is required since the included summaries are not extractive and this approach needs an extractive summary as a training target.

7.2 Evaluation

Evaluation on the CNN/DailyMail data-set was conducted automatically, based on the ROUGE-1, ROUGE-2 and ROUGE-L (ROUGE-F1) measures. Additionally tests with human judges were conducted on summaries created from this data-set. [12] [27] While human-based evaluations are more indicative of the performance than ROUGE based tests [23], the methods are too different for a direct comparison of their respective results. The focus of the evaluation is accordingly put on the comparable automatic evaluation based on the ROUGE-F1 measure. However, it is noted, that in their test settings, both algorithms outperformed other algorithms they were compared with.

7.2.1 ROUGE-F1

ROUGE-F1 consists of the previously introduced ROUGE-1, ROUGE-2 and ROUGE-L scores, meant to measure precision and recall of an automatic summary compared to a gold standard summary. Both approaches were evaluated on the CNN/DailyMail data-set and compared to different, state-of-the-art algorithms. Additionally they were compared to the LEAD-3 benchmark, which uses the summaries created when the first three sentences of a text are selected [16]. The comparatively good performance of LEAD-3 on this data-

Table 2: Results of the ROUGE-F1 Evaluations on the CNN/DailMail Data-Set

CNN/DailyMail Data-Set	**R1**	**R2**	**RL**
LEAD-3 (Narayan et al. [15])	39.6	17.7	36.2
LEAD-3 (BanditSum) [27]	40.0	17.7	36.2
LEAD-3 (BERTSUM) [12]	40.4	17.6	36.6
SUMMARUNNER (2017) [14]	39.6	16.2	35.3
SUMO (2019) [12]	41.0	18.4	37.2
BanditSum [27]	41.5	18.7	37.2
BERTSUMEXT [12]	43.2	20.2	39.6
BERTSUMEXT (large) [12]	43.8	20.3	39.9
BERTSUMABS [12]	41.7	19.3	38.7

set, however, is a good indicator for the layout bias that is inherent to news article based text. [12] [27]

Table 2 depicts the ROUGE-F1 performance of BertSum and BanditSum compared to three different implementations of the LEAD-3 benchmark and two other neural network-based summarizers. The test data shows that both summarizers outperform LEAD-3 and their competitors. Furthermore, BertSumExt clearly outperforms BanditSum in this setting. It is suggested by these results, that both approaches represent effective improvements over the LEAD-3 benchmark and that their resulting summaries are of state-of-the-art quality.

However, it must be noted that good ROUGE-F1 values are by itself not necessarily indicative of a good summary. They are solely focused on lexical overlap between reference and candidate summary. Additionally, good ROUGE scores do not imply factual correctness or relevance. This is attributable to both constraints of the environment under which ROUGE was first introduced as well as its limited ability to measure anything more than lexical attributes. [10]

8 Conclusion

State-of-the-art performance in extractive text summarization can be achieved using decisively different approaches in training and network architecture. While both of the introduced models include some sort of context based language understanding, one trains the model by formulating the problem as a contextual bandit reinforcement learning problem and teaches an agent to maximize the ROUGE value of the summaries compared to abstractive gold-standards. The other uses a state-of-the-art pre-trained encoder to achieve contextual document-level representations of the content, which is then used to pick the most relevant sentences of the text.

While both approaches lead to state-of-the-art scores beating historic and recent methods clearly, they still barely outperform LEAD-3 on news article-based data-sets. This can only in part be attributed to the layout bias in the training and evaluation data. Additionally ROUGE, while being the most common metric for automatic evaluation is not necessarily an adequate measure of summary quality. It is solely focused on lexical methods and relies on gold-standard summaries that are not always available. Training on ROUGE therefore is by itself a limited approach, as it is highly reliant on the quality of the gold-standards and does not accurately reflect what humans would define as a good

summary. Additionally it does not include any other important aspects like conciseness, coherence, etc. At most, a network trained on ROUGE can only imitate a certain style of summaries, or work for a certain type of text, without the possibility to adapt to different languages or other styles. Furthermore, both introduced approaches are incapable of creating variable size summaries, requiring decisions on the summary length during training (in the case of BanditSum) or during execution (in the case of BertSumExt). This does not reflect how humans would create summaries well.

Still, the state-of-the-art is improving rapidly with abstractive summarizers capable of outperforming extractive summarizers on the horizon. Current Bert-based approaches, outperforming the reinforcement learning based approaches, are a promising movement that could lead to better extractive and even abstractive summarizers, by achieving a high level of contextual understanding of entire documents. Further work in this research intensive area is required and could lead to even better contextual understanding, which is important for summarization tasks. Another area that needs additional research would be a replacement for the ROUGE metric. Any such replacement would, in the best case, consider other quality aspects of a summary like coherency and lack of logical contradictions. While these are hard to implement in automatic evaluations, it could help the field of text summarization a lot, providing a better basis for benchmarking and training.

References

[1] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saeid Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and Krys Kochut. Text Summarization Techniques: A Brief Survey. *International Journal of Advanced Computer Science and Applications*, 8(10):397–405, 2017.

[2] Sanghwan Bae, Taeuk Kim, Jihoon Kim, and Sang goo Lee. Summary level training of sentence rewriting for abstractive summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 10–20, Hong Kong, China, November 2019. Association for Computational Linguistics.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), August 2013.

[4] Jianpeng Cheng and Mirella Lapata. Neural summarization by extracting sentences and words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 484–494, Berlin, Germany, August 2016. Association for Computational Linguistics.

[5] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, June 2019.

[7] Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pre-training help deep learning? In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 201–208, May 2010.

[8] Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1693–1701, 2015.

[9] Djoerd Hiemstra. *Language Models*, pages 2061–2065. Springer New York, New York, NY, 2018.

[10] Wojciech Kryscinski, Nitish Shirish Keskar, Bryan McCann, Caiming Xiong, and Richard Socher. Neural Text Summarization: A Critical Evaluation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 540–551, November 2019.

[11] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, July 2004.

[12] Yang Liu and Mirella Lapata. Text Summarization with Pretrained Encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, page 37213731, November 2019.

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[14] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents, 2017.

[15] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Ranking sentences for extractive summarization with reinforcement learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1747–1759, June 2018.

[16] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Ranking sentences for extractive summarization with reinforcement learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1747–1759, June 2018.

[17] Pandu Nayak. Understanding searches better than ever before. Technical report, October 2019. https://blog.google/products/search/search-language-understanding-bert.

[18] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

[19] 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, October 2014.

[20] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[21] Yutaka Sasaki. The truth of the F-measure. *Teach Tutor Mater*, pages 1–5, January 2007.

[22] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[23] Juan-Manuel Torres-Moreno. *Automatic Text Summarization*. ISTE, 2014.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz 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.

[25] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.

[26] 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, ukasz 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. arXiv, 2016.

[27] Dong Yue, Yikang Shen, Eric Crawford, Herke van Hoof, and Jackie Chi Kit Cheung. BanditSum: Extractive Summarization as a Contextual Bandit. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3739–3748, November 2018.

[28] Xingxing Zhang, Furu Wei, and Ming Zhou. HIBERT: Document level pre-training of hierarchical bidirectional transformers for document summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5059–5069, Florence, Italy, July 2019. Association for Computational Linguistics.

[29] Ming Zhong, Pengfei Liu, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. Searching for effective neural extractive summarization: What works and whats nex. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, page 10491058, Florence, Italy, July 2019. Association for Computational Linguistics.