Neural Automatic Summarization

Emil Lynegaard
Natalie Schluter (Supervisor)
IT University of Copenhagen
`ecly@itu.dk`

June 3, 2019

Abstract

Most recent neural network techniques for abstractive summarization, operate on a word-by-word level for generating summaries. Models generally grow huge, to encapsulate a large output space, resulting in systems requiring several days of training. We propose an optimized version of the Pointer Generator semi-abstractive summarization approach, which uses a smaller model, while retaining a large output space, by enabling copying of words from the input. As a result, we drastically reduce the required training time of said approach, without reductions in quality of the resulting summaries. We further experiment with several novel expansions of the Pointer Generator. One such approach extends the architecture to operate on important phrases instead of words, where importance is learned as part of the training. By extending copying of words, to copying of phrases, and allowing both input and output to consist of such phrases, we effectively reduce the size of both input and output. This approach results in significant computational savings for both training and inference. We show that both our optimized Pointer Generator and our phrase level architecture produce competitive results.

Contents

List of Figures

List of Tables

1 Introduction

Automatic summarization uses computers to generate a summary of a document, preserving as much important information as possible. Given the vast amounts of unstructured data available today, the applications for automatic summarization systems are wide and many. Examples include, search engine previews, question answering and document analysis. In this paper we focus on neural summarization, in particular models remedying the so called *rare word problem*, by incorporating a copy mechanism. Most neural architectures for summarization require very large datasets, with models needing several days of training. Our work propose significant optimizations for a recent state-of-the-art model, drastically reducing model size and training time. We further present a new novel architecture, which segments both input and output into *chunks*, offering further computational benefits.

Concretely, summarization seeks to produce a summary $\mathbf{y}$ consisting of tokens $(y_1, \ldots, y_M)$ from a document $\mathbf{x}$ consisting of tokens $(x_1, \ldots, x_N)$ where $M < N$. To produce a summary $\mathbf{y}$ from $\mathbf{x}$, consider $\mathcal{Y}$ the set of all possible summaries of length M consisting of words from vocabulary V. Given this, we consider an automatic summarization system to be *abstractive* if it seeks to find the optimal sequence from this $\mathcal{Y}$,

$$\begin{aligned}&\arg\max s(\mathbf{x}, \mathbf{y})\\ &\mathbf{y} \in \mathcal{Y}\end{aligned} \tag{1}$$

under some scoring function $s : x \times y \mapsto \mathbb{R}$. In contrast to this, we consider a system to be *extractive* if it constructs the summary using only words present in the input document $\mathbf{x}$. To represent this, we consider our summary as consisting only of tokens from $\mathbf{x}$, represented with x_{i_1} corresponding to the first word in $\mathbf{y}$, referring to x_i from the input document. This gives us the following definition:

$$\begin{aligned}&\arg\max s(\mathbf{x}, (x_{i_1}, \ldots, x_{i_M}))\\ &0 < i < N\end{aligned} \tag{2}$$

Lastly, we consider the related task of document *compression*, where we limit ourselves to constructing $\mathbf{y}$ solely by removing words from $\mathbf{x}$. This can be formulated as extractive summarization, with an added constraint on the order of the words in the output:

$$\begin{aligned}&\arg\max s(\mathbf{x}, (x_{i_1}, \ldots, x_{i_M}))\\ &0 < i < N\\ &i_j < i_{j+1}\end{aligned} \tag{3}$$

In the following paper, we focus on a recent approach for neural summarization from See et al. (2017), that combines an abstractive approach, with an extractive approach, by allowing the abstractive system to copy words from input, effectively expanding V with all the words from $\mathbf{x}$. For this, we propose several different variations and optimizations of the system, as well as new ways to train said system, yielding improvements to both summary quality and training time.

Based on our optimizations, we further investigate the possibility of a neural summarization system, that operates on a level higher than words. Most neural summarization approaches rely on attention calculations (Cf. Section 3) with complexity $O(NM)$, leaving huge potential computational savings to be had, by reducing the size of N and M. As such, we introduce a novel approach to summarization, that allows the model to process the input as a sequence of chunks. Retaining the approach of enabling copying from the input, we allow the model to copy chunks from the input. The optimizations we present, as well as the novel chunking approach, have resulted in the production of a separate related paper, Lynegaard and Schluter (Submitted).

2 Preliminaries

In this work, we apply neural networks to the task of automatic summarization. Neural networks are used for learning approximations of complex models and consist of *neurons*. We can view each neuron as a linear function $y = wx + b$, where x is the input, w is the learnable weight, b is an optional learnable bias and y is the output. A simple instantiation of a small *feedforward* (FF) neural network is shown in Figure 1.

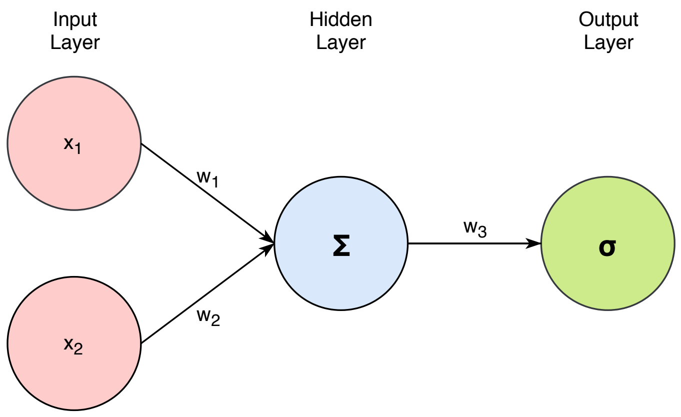

Figure 1: A simple feedforward neural network.

Here our neural network takes as input the vector $\mathbf{x}$ consisting of (x_1, x_2) and computes: $\sigma((x_1w_1 + x_2w_2)w_3)$, where σ is the logistic sigmoid function. In practise, we represent this concisely using matrices, where input layer weights are a 1×2 matrix W_i consisting of $\binom{w_1}{w_2}$ and the hidden layer w_h is the scalar w_3. We can now write our computation from Figure 1 as $\sigma(\mathbf{x}W_iw_h)$.

For our specific instantiation from Figure 1, each layer is fully-connected (or dense), meaning that every neuron in each layer is connected to every neuron in the subsequent layer. We refer to σ as the activation function, or non-linearity, for our single output neuron. The learnable parameters of our network are (w_1, w_2, w_3), for which we hope to find suitable values, making our network approximate our desired model.

In this paper, we will make extensive use of a class of neural networks called *recurrent neural network* (RNN). RNNs extend the FF neural network by handling variable-length input and by allowing it to represent context while processing sequential input, such as natural language. In essence, RNNs can be seen as a looping FF, taking the output of the previous iteration as input.

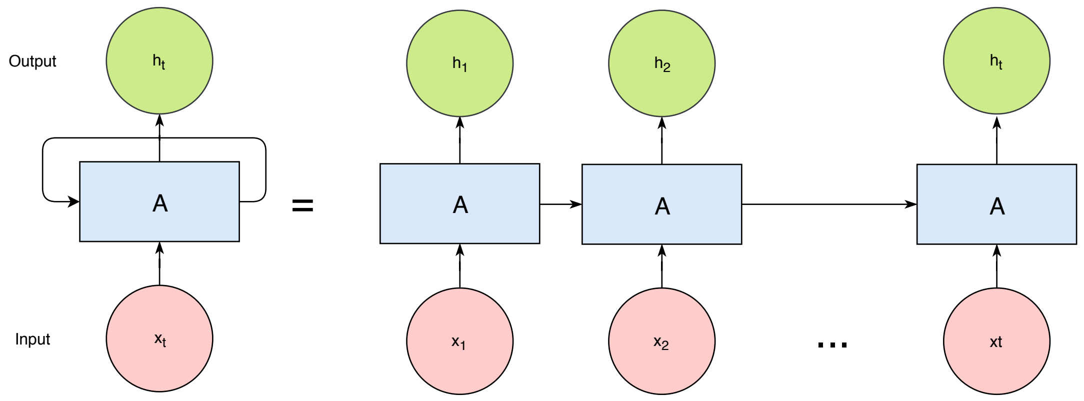

Figure 2: An unrolled recurrent neural network.

Formally, a *vanilla* RNN given the sequence $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ computes its recurrent state h_t by:

$$h_t = g(Wx_t + Uh_{t-1}) \tag{4}$$

where $h_0 = 0$, g is a non-linearity and both W and U are learnable parameters. As the vanilla RNN was found difficult to train for capturing long-term dependencies within a sequence[1], expansions remedying these limitations have been proposed.

Specifically, we focus on two most common RNN variations, namely *long short-term memory* (LSTM) from Hochreiter and Schmidhuber (1997) and *gated recurrent units* (GRU) from Cho et al. (2014). Both LSTM and GRUs fit into Figure 2, as implementations of $\mathbf{A}$, which we refer to as the RNN *cell*. LSTMs and GRUs mainly differ in the fact that LSTMs maintain an additional so called *memory cell*. Both LSTMs and GRUs are composed of *gates*, which modulate the flow of information into and out of the cell. In Figure 3, we show the internals of both LSTMs and GRUs, as well as each of their gates.

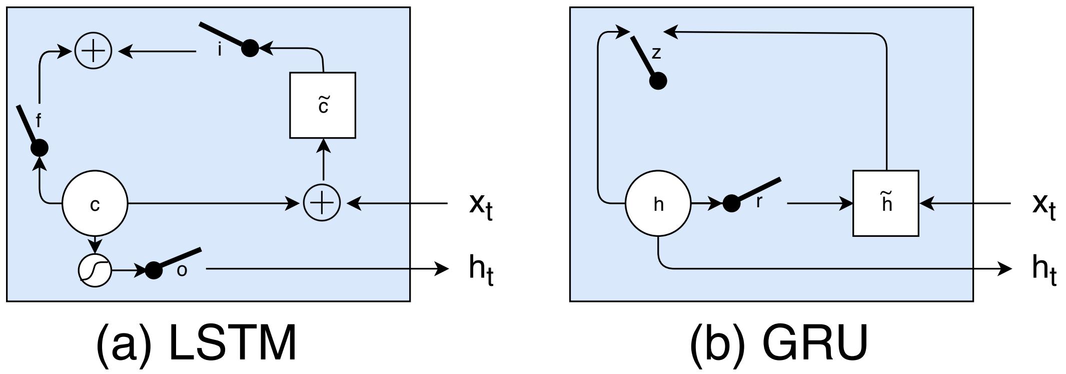

Figure 3: Illustration of (a) LSTM and (b) GRU from Chung et al. (2014). LSTM has three gates: input gate i, forget gate f and output gate o. GRU has two gates: update gate z and reset gate r.

LSTM For LSTMs, we compute h_t at time step t, using the memory c_t and the output o_t:

$$\begin{aligned} h_t &= o_t \tanh(c_t) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + V_0 c_t) \end{aligned} \tag{5}$$

where W_o, U_o and V_o are learnable parameters, with V_o being a diagonal matrix. Intuitively, the output gate o_t modulates the degree to which the memory's content is exposed. The memory c_t is computed by partially forgetting previous memory (c_{t-1}) and partially adding new memory ($\tilde{c}_t$):

$$\begin{aligned} c_t &= f_t c_{t-1} + i_t \tilde{c}_t \\ \tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1}) \end{aligned} \tag{6}$$

where W_c and U_c are learnable parameters. We modulate the amount of existing memory retained using the forget gate f_t, and the amount of new memory added using input gate i_t, computed as:

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + V_f c_{t-1}) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1}) \end{aligned} \tag{7}$$

[1]While we omit details on how the parameters of a neural network are learned (see backpropagation and gradient descent), we refer the reader to the 'vanishing gradient problem' for an explanation of the shortcomings of vanilla RNNs.

where W_f, W_i, U_f, U_i, V_f and V_i are learnable parameters, with V_f and V_i being diagonal matrices. We note that in contrast to vanilla RNNs, LSTM are able to modulate the extent to which memory is overwritten. Intuitively, this allows the LSTM to retain information about possible important features found early in the sequence processed.

GRU Similarly to the LSTM, GRUs use gates to modulate the flow of information. For GRUs, we compute h_t using the update gate z_t as a linear interpolation of the previous recurrent state (h_{t-1}) and the candidate state ($\tilde{h}_t$):

$$\begin{aligned} h_t &= (1 - z_t)h_{t-1} + z_t\tilde{h}_t \\ z_t &= \sigma(W_z x_t + U_z h_{t-1}) \end{aligned} \tag{8}$$

where W_z and U_z are learnable parameters. We note that GRUs modulate the degree to which information is retained/added in a similar fashion to LSTMs, differing in the GRU being unable to module the degree to which its state is exposed (the LSTM's output gate). The candidate state $\tilde{h}_t$ is calculated using the reset gate r_t, which allows the GRU to partially forget its previous state:

$$\begin{aligned} \tilde{h}_t &= \tanh(W x_t + U(r_t \odot h_{t-1})) \\ r_t &= \sigma(W_r x_t + U_r h_{t-1}) \end{aligned} \tag{9}$$

where W, U, W_r and U_r are learnable parameters, and $\odot$ is element-wise multiplication. We note in particular that GRUs, with its two gates, have fewer parameters than LSTMs, making them faster to train.

For further reading, from which this section has drawn much inspiration, we refer to Christopher Olah's blog post[2] for a simple RNN and LSTM explanation, as well as Chung et al. (2014) for an in-depth empirical analysis and comparison of GRUs and LSTMs.

3 Related Work

Recurrent neural networks (RNN) have proven themselves well-suited for several natural language processing application, based on their ability to represent context, while processing sequential input. The nodes in an RNN are connected, forming a directed graph exhibiting temporal dynamic behavior.

In Cho et al. (2014) the RNN Encoder-Decoder neural network architecture was introduced for machine translation. Here an RNN first encodes the source sequence into a fixed size vector representation. This vector representation is then fed to another RNN that decodes the vector representation back into a target sequence of words, representing the resulting translation.

Following the introduction of RNN Encoder-Decoders for machine translation, Bahdanau et al. (2015) introduced an *attention* mechanism, that allowed the model to focus on parts of the input when generating an output word. This was done by learning an alignment α^t between the sequence of encoder outputs h consisting of $(h_1, \ldots, h_N)$ and a decoder state s_t at time step t:

$$\begin{aligned} e^t &= v \ \tanh(W_h h + W_s s_t + b_{attn}) \\ \alpha^t &= \text{softmax}(e^t) \end{aligned} \tag{10}$$

where v, W_h, W_s and b_{attn} are learned parameters. This alignment is then used to calculate a decoder context h_t^*, that conceptually focuses its attention towards parts of the input at the current

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

time step. This allows the decoder RNN to shift its attention around the input at each time step, as it produces its corresponding output. One way to calculate h_t^* given α^t is simply as a weighted sum:

$$h_t^* = \sum_{i=1}^{N} \alpha_i^t h_i \tag{11}$$

This way of calculating the context is used by Bahdanau et al. (2015) and See et al. (2017), as well as several others. Lastly, to produce the output at each time step t, we compute a probability distribution P_{vocab} over the vocabulary as follows:

$$P_{vocab} = \text{softmax}(V'(V[s_t, h_t^*] + b) + b') \tag{12}$$

where V', V, b and b' are learnable parameters.

3.1 Rare word problem

While Cho et al. (2014) and Bahdanau et al. (2015) both introduced their models for machine translation, a similar architecture was first pioneered for summarization by Rush et al. (2015). Here, the RNN Encoder-Decoder architecture was used for abstractive headline generation, given the first sentence of an article. Prior to Rush et al. (2015), most successful summarization systems had used extractive non-neural methods. Intuitively, given enough computational resources and a sufficiently sophisticated model, an abstractive model will always be able to emulate an extractive model, making it a more powerful abstraction. In practise, resources are limited, and abstractive models limit their vocabulary to a fixed size, rendering summaries with rare words impossible.

Pointer Generators were originally introduced to remedy this limitation in machine translation by Luong et al. (2015b). Here they introduce a novel approach to allow an Encoder-Decoder model to produce out-of-vocabulary (OOV) words, by *pointing* to the input. This approach was adapted for summarization in Gu et al. (2016) which See et al. (2017) further improved, by introducing a coverage mechanism, using attention history to reduce repeated attention. The pointing is done by by calculating a generation probability $p_{gen} \in [0, 1]$ for each time step t, using h_t^* and s_t from Bahdanau et al. (2015), as well as x_t, the current input for the decoder:

$$p_{gen} = \sigma(w_{h^*} h_t^* + w_s s_t + w_x x_t + b_{ptr}) \tag{13}$$

where the vectors w_{h^*}, w_s, w_x and bias term b_{ptr} are learnable parameters. The scalar p_{gen} then functions as a soft switch, dictating whether to generate a word from the vocabulary, by sampling from the probability distribution P_{vocab}, or to copy a word from the input by sampling from α^t. This results in the following final word-level prediction over a now extended vocabulary:

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} \alpha_i^t \tag{14}$$

The coverage mechanism extends the model by keeping track of a coverage vector c^t, as the sum of attention distributions for all previous time steps:

$$c^t = \sum_{t'=0}^{t-1} \alpha^{t'} \tag{15}$$

where c^t is initialized with zeros for $t = 0$. The attention mechanism from Equation 10 is then extended with the learned vector w_c, projecting c^t as follows:

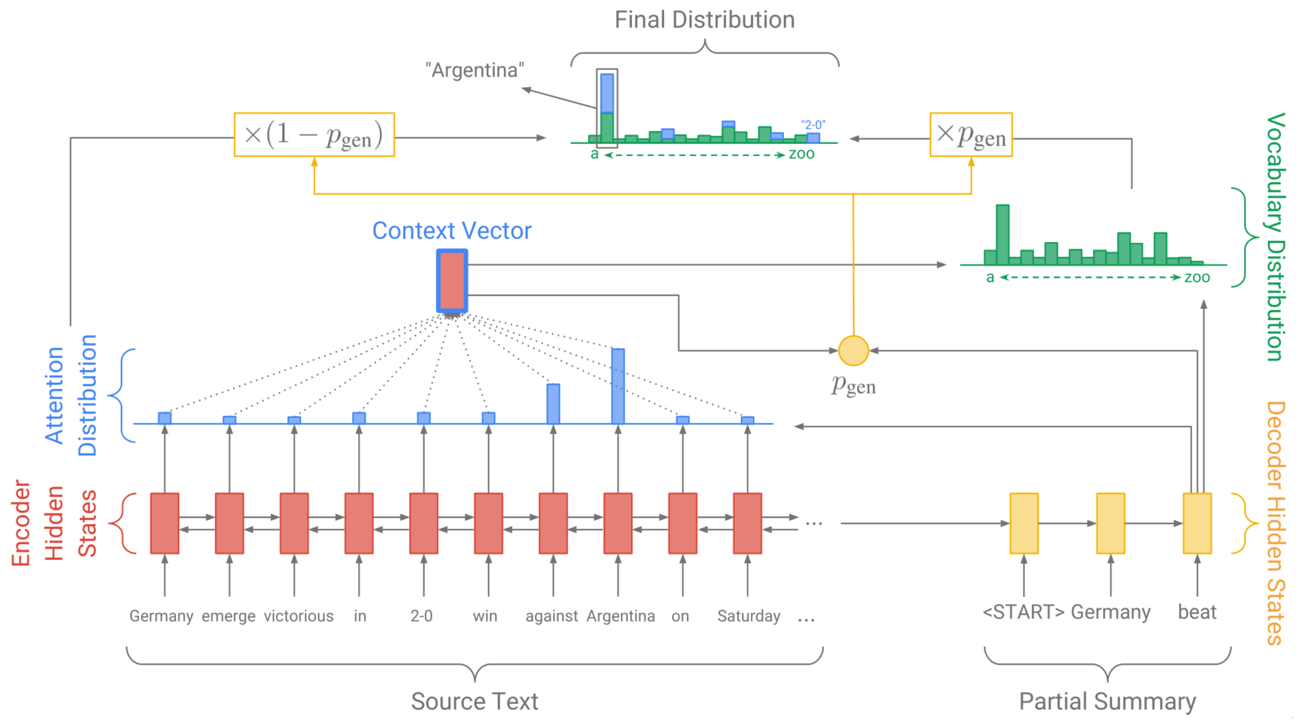

Figure 4: The architecture as seen in the Pointer Generator paper (See et al., 2017).

$$e^t = v \ \tanh(W_h h + W_s s_t + w_c c^t + b_{attn}) \tag{16}$$

Lastly, See et al. (2017) adds a coverage loss term to the model, penalizing repeated attention. This is calculated as shown in Equation 17, whereafter it is reweighed by some hyperparameter λ and added to the standard negative log-likelihood loss:

$$\begin{aligned} \text{covloss}_t &= \sum_i \min(\alpha_i^t c_i^t) \\ \text{loss}_t &= -\log P(w_t^*) + \lambda \ \text{covloss}_t \end{aligned} \tag{17}$$

where w_t^* is the target word at time step t.

The Pointer Generator model from See et al. (2017) has been used as an integral part of several other recent state-of-the-art models. An overview of of the Pointer Generator architecture is shown in Figure 4.

Gehrmann et al. (2018) extends the Pointer Generator with a content selector, that restricts the model to only point to the selected parts of the input. With the added content selector, they manage to significantly improve ROUGE [3] scores. Gehrmann et al. (2018) also experiment with transformer models from Vaswani et al. (2017), finding that this did not further improve their scores.

In Hsu et al. (2018), Pointer Generator is extended by normalizing the word-level attention based on a sentence-level attention, calculated as the sentence selection probability from Nallapati et al. (2017). Along with this, they introduce a novel inconsistency-loss, which penalizes inconsistencies between the word-level and sentence-level attention distributions.

Another extension of the Pointer Generator model was introduced in Krycinski et al. (2018), where they factor the decoder into an external language model responsible for generation and use

[3]ROUGE: **R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation. For a description of ROUGE, refer to Section 5.2.

the Pointer Generator model for extraction and compaction. They further extend the model with a mixed learning objective, that jointly optimizes for ROUGE while encouraging abstraction.

Finally, Liu et al. (2018) propose training a Pointer Generator as a *generative adversarial network* (GAN) (Goodfellow et al., 2014), where a separate adversarial discriminator is trained to distinguish the generated summaries from the ground truth. They combine their discriminative objective with cross entropy loss on the generator, finding that this training architecture improved the scores from See et al. (2017).

3.2 Reinforcement Learning

Several recent results have indicated that mixed learning objectives, using reinforcement learning to optimize for metrics other than maximum likelihood, allow additional improvement on the document summarization task. Paulus et al. (2017) use a combination of maximum likelihood and ROUGE to train a model adapted from See et al. (2017). Their learning objective rewards outputs resulting in high ROUGE scores, while they hypothesize that their maximum likelihood objective essentially operates as a language model conditioned on previous outputs, resulting in improved output coherence. Their model is augmented with an intra-attention mechanism, which allows the attention mechanism to alter its context based on previous output, effectively creating a different form of coverage. A similar mixed learning objective was used for an extractive system by Narayan et al. (2018), where it was applied to summarization using sentence extraction. Here a convolutional neural network was used to produce sentence level encodings, that are fed to a document-encoder RNN, which produces an encoded representation of the entire document. The document encoding, alongside the sentence encodings, are then given as input to a sentence extractor RNN, which produces sentence rankings from which selections can be made.

Similarly, Chen and Bansal (2018) produce sentence level encodings using a convolutional neural network. They then use a Pointer Network (Vinyals et al., 2015) to calculate sentence extraction probabilities, trained with reinforcement learning, optimizing for ROUGE-L (F1). Finally, the extracted sentences are then paraphrased using a Pointer Generator model. Another contribution of Chen and Bansal (2018) is parallel decoding, which was made possible by first operating on the sentence-level, then the word-level. They report 10–20x faster inference speed and 4x faster training convergence than previous long-paragraph Encoder-Decoder models.

4 Data

There exists a rather limited amount of datasets used for training neural summarization systems. The most frequently used summarization corpus is the CNN/Dailymail (CNNDM) dataset from Hermann et al. (2015). Recently a significantly larger dataset called Newsroom was curated (Grusky et al., 2018), which contains roughly 4x the training pairs of CNNDM. No papers have however been published with models tuned and trained for the Newsroom dataset, making it difficult to compare results. Another large dataset consisting of articles from New York Times (NYT) (Sandhaus, 2008) was recently adapted for summarization, and has been evaluated in some newer work. While we focus on CNNDM to facilitate comparison with existing work, we report scores for both Newsroom and NYT in Section 7, enabling future comparison to our work. In this section we will briefly introduce and compare CNNDM, Newsroom, NYT and the test datasets from the Document Understanding Conferences[4] (DUC).

[4]https://duc.nist.gov/

4.1 CNN/DailyMail

Almost all recent neural summarization systems are trained and evaluated using the CNNDM dataset. The CNNDM dataset consists of 287,226 training pairs, 13,368 validation pairs and 11,490 test pairs. Each article in the dataset is accompanied by 3–4 highlights, that are used to form the corresponding summary. The dataset is generally used in two distinct forms: (1) anonymized, preprocessed with entities such as *Washington* replaced with a unique identifier such as `@example1` and (2) a non-anonymized version of the data, that operates directly on the original text. All our work, as well as the work we compare to, use the non-anonymized version.

While the CNNDM's summaries are abstractive, they may closely resemble excerpts, due to the inherent nature of newspaper article highlights. This may have the effect of slightly favoring extractive systems. Because of this, evaluation of different, possibly more diverse datasets, may be valuable.

4.2 New York Times

The annotated NYT corpus (Sandhaus, 2008), is a large scale dataset that has been adapted for summarization in Paulus et al. (2017). The resulting summarization dataset consists of 654,762 articles with corresponding summaries Sandhaus (2008), divided into 589,294 training pairs, 32,734 validation pairs and 32,734 testing pairs. Following Paulus et al. (2017), several other papers, such as Gehrmann et al. (2018) and Celikyilmaz et al. (2018) have used the NYT dataset as an addition point of comparison, alongside CNNDM.

4.3 Newsroom

A new summarization dataset was recently introduced by Grusky et al. (2018). The Newsroom dataset is substantially bigger than CNNDM, with a total of 1.3million articles, each of which is accompanied by a single summary written by authors and editors in the newsroom of various major publications. The dataset is divided into 995,041 training pairs, 108,837 validation pairs, 108,863 test pairs. In Grusky et al. (2018), the dataset is compared to several others, including CNNDM, and is found to contain summaries exhibiting more diverse summarization strategies. This is depicted in Figure 5, where larger coverage of the two-dimensional space indicates a more varied summarization strategy.

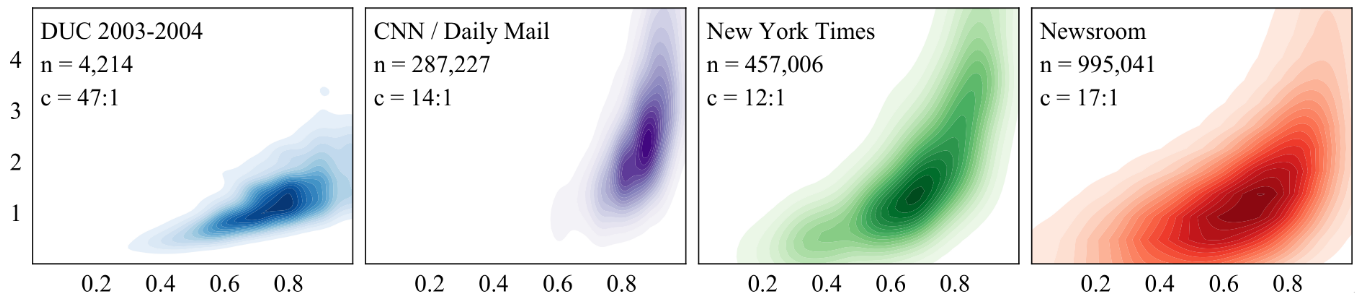

Figure 5: The dataset comparison from Grusky et al. (2018). X-axis represents extractive fragment coverage (higher values represents less novelty). Y-axis represents density, a measure of the average length of extracted segments (higher values represent longer extracted sentences). At the top left corner of of each dataset's plot is the total number of articles n, and the word-level compression ratio c.

Based on the diverse data found in Newsroom, it may provide a better basis for fairly evaluating summarizations models at the news summarization task. The Newsroom dataset has currently only been evaluated on a limited set of summarization systems in Grusky et al. (2018), making it difficult to compare results. Ideally, recent state of the art systems evaluated on CNNDM are to be trained and evaluated on the Newsroom dataset, giving an idea of the quality correlation between Newsroom and CNNDM. It may additionally be of interest to see how a model's ROUGE score correlation between CNNDM and Newsroom, translates to test sets such as DUC, or how it correlates with human evaluation.

4.4 DUC

The DUC dataset is a small test dataset of 200 articles with accompanying handwritten summaries. It was originally introduced for a multi-document summarization task, meaning that each summary has multiple corresponding articles. Additionally, each set of articles have 4 summaries written by 4 different human summarizers. The DUC dataset exists in several different versions, from which we conduct analysis on the 2004 set. While we do not evaluate any of our work on DUC, we include it here for completion.

4.5 Analysis

Extractive systems rely on reference summaries having a small amount of novel words, as their vocabulary is limited to that of the corresponding article. Other systems, such as the content selector from Gehrmann et al. (2018), rely on the summary being reasonably aligned with its corresponding article for training. To get an overview of the structure of the previously discussed datasets' attributes, including novelty and summary/article alignment, we compute various metrics for each dataset. These attributes are presented in Table 1.

	Article Length	Summary Length	Edit Distance	Novel %	LCS
CNNDM	694 (352σ)	56 (25σ)	20.19	17.50%	36.66
Newsroom	672 (818σ)	28 (28σ)	6.29	11.66%	22.71
New York Times	693 (602σ)	42 (33σ)	13.20	17.09%	29.40
DUC (Individual Articles)	605 (450σ)	-	-	-	-
DUC (Concat. Articles)	6,051 (2612σ)	105 (5σ)	37.39	6.24%	68.75

Table 1: Number of words in article and summaries, as well as corresponding standard deviation (σ), edit distances, novel word percentages and longest common subsequence (LCS). All values are shown as the mean across the entire development sets. For DUC, which only consists of 200 test instances, we use the entire dataset. As DUC is a multi-document summarization dataset, we show the results for concatenated articles, as well as a separate row for individual article stats. Edit distance was calculated using Levenshtein with 0-cost deletes. All input was lowercased with punctuation removed.

While we find that CNNDM has a slightly higher percentage of novel words in its summaries than the other datasets, we still note that its summaries share longer common subsequences with their corresponding articles than both Newsroom and NYT. Although average article lengths align well across different datasets, we observe larger relative differences between summary lengths.

5 Experiments

In the following section, we introduce our experimental setup, our evaluation metric and our baseline models. We first reimplement the best model from See et al. (2017), which we compare to existing implementations thereof. Based on our reimplementation, we propose an optimized model, yielding similar results using both less parameters and training time. We then investigate several tunes of our optimized model and propose various novel expansions, both in terms of architecture and training configuration. The code used for both training and evaluating all our work is made freely available on GitHub[5].

5.1 Dataset

For all our experiments, we use the CNNDM dataset described in Section 4.1, to allow comparison with previous work. The preprocessing of the data is similar to See et al. (2017), which uses the non-anonymized version of CNNDM, lowercased and tokenized using CoreNLP (Manning et al., 2014).

5.2 Evaluation

For evaluating summarization systems, the most commonly used metric is ROUGE (Lin, 2004). ROUGE is divided into several variants, out of which we normally use ROUGE-N (where n $\leq$ 2) and ROUGE-L. ROUGE is generally thought to correlate well with human judgement for evaluation of a summary, although ROUGE-2 specifically shows the highest correlation according to Graham (2015). We show the ROUGE percentile calculations, albeit when ROUGE scores are presented, we multiply them with 100, resulting in scores between 0–100. For each of our metrics, we calculate both precision and recall, to finally calculate the corresponding F1-score. Note that ROUGE supports evaluation using multiple reference summaries, although we only work with single reference summaries for all our work.

ROUGE-N For ROUGE-N, we calculate the score based on the multiset of all grams of size n in our predicted summary (S), and the multiset all of grams of size n from a set of reference summaries (R).

$$\begin{aligned}
\text{ROUGE-N}_P &= \frac{|S \cap R|}{|S|} \\
\text{ROUGE-N}_R &= \frac{|S \cap R|}{|R|} \\
\text{ROUGE-N}_{F1} &= 2 \cdot \frac{\text{ROUGE-N}_P \cdot \text{ROUGE-N}_R}{\text{ROUGE-N}_P + \text{ROUGE-N}_R}
\end{aligned} \tag{18}$$

Intuitively, we are measuring the n-gram overlap between S and R.

[5] https://github.com/ecly/pointer_summarization

ROUGE-L We calculate ROUGE-L using the longest common subsequence between our predicted summary y_{pred} and any of the reference summaries y_{refs}.

$$\begin{aligned}
\text{ROUGE-L}_P &= \frac{LCS(y_{pred}, y_{refs})}{|y_{pred}|} \\
\text{ROUGE-L}_R &= \frac{LCS(y_{pred}, y_{refs})}{|y_{refs}|} \\
\text{ROUGE-L}_{F1} &= 2 \cdot \frac{\text{ROUGE-L}_P \cdot \text{ROUGE-L}_R}{\text{ROUGE-L}_P + \text{ROUGE-L}_R}
\end{aligned} \tag{19}$$

From this, we derive that ROUGE-L represents how large a percentile of y_{pred} corresponds to a subsequence in y_{refs}.

5.3 Pointer Generator

To improve on the neural summarization task, we initially reimplement the Pointer Generator from See et al. (2017) in PyTorch[6]. We do this following Section 3, where we find that the Pointer Generator architecture has served as the core of several of the current state-of-the-art systems. To evaluate our reimplementation of the Pointer Generator, we compare our results to scores reported in See et al. (2017); Gehrmann et al. (2018), as well as a retrained model using the original code[7]. We describe the systems and any possible differences in the following paragraphs and present an overview of the results in Table 2.

Pointer Generator (See et al. 2017) Inspecting the source code of the original Pointer Generator implementation, we observe a few differences that were not covered in its corresponding paper. In the description of the public code, the authors vaguely describe how they obtained the results from the paper. This was done by starting with small values for *maximum article length* and *maximum summary length*, which are used for truncating the instances. Throughout training, they then manually intervened, increased these values and resumed training. No exact values were given for this procedure, making it impossible to fully replicate.

Secondly, for every time step t, See et al. (2017) computes a joint representation of the decoder input and context h^*_{t-1} (zero vector for $t = 0$). During training, the decoder input is the embedding of the ground truth token for $t - 1$, which we denote $\bar{w}^*_{t-1}$.

$$\hat{x}_t = W_{context}[\bar{w}^*_{t-1}, h^*_{t-1}] \tag{20}$$

They then feed $\hat{x}_t$ and last hidden state s_{t-1} to the decoder LSTM, instead of $\bar{w}^*_{t-1}$ and s_{t-1}, which may otherwise have been assumed. Lastly, since they use the final hidden state from the bidirectional encoder as their unidirectional decoder's initial state, they project this down using two dense layers, followed by a `ReLU`[8] activation function.

In the description of the official repository, it is stated that the authors struggled to fully replicate their own results following some code changes. They attribute this to the model from the paper having been slightly overfitted to random hyperparameter tuning, from which the best model was selected.

[6]https://pytorch.org/

[7]https://github.com/abisee/pointer-generator

[8]For an overview of activation functions, refer to Wikipedia: https://en.wikipedia.org/wiki/Activation_function

As the official Pointer Generator repository contains the test outputs for the models from See et al. (2017), we attempt to recompute their ROUGE scores. In doing so, we find that the official ROUGE script, which they reportedly used, fail to recreate their scores[9]. Instead we found that we could match their reported scores using our previously mentioned Python ROUGE reimplementation (py-rouge[10]), which uses an improved stemmer[11] resulting in better scores. We suspect that the scripts may have been mixed up during experimentation, or that it was not clear to the authors that py-rouge yielded different scores, as it does not fully recreate the official script. We report the corrected values using the official ROUGE script in Table 2.

Pointer Generator (See et al. 2017) Rerun To investigate reproducability of the original Pointer Generator results, we rerun the published code[12]. We retrain their system according to instructions provided in the description of the official repository, using TensorFlow[13] 1.0 which was stated as compatible. Our retrained version uses the exact hyperparameters described in See et al. (2017), but refrain from stopping and resume the training as described on the repository, since exact values were omitted.

Pointer Generator (Gehrmann et al. 2018) In Gehrmann et al. (2018) they try to reimplement the Pointer Generator model and achieve results close to the those of the original. This is however done using several tuning steps, that were not done in the original implementation. The steps taken to reproduce the results for Gehrmann et al. (2018) are described in the paper and in the corresponding OpenNMT summarization document[14]. The main differences from their description are:

- Learning rate halves after an epoch with no validation improvement
- Sentences in targets are surrounded with <t> and </t> tags
- Decoder uses $hidden\ size = 512$ instead of $hidden\ size = 256$
- The loss of a sequence is divided by its length, supposedly favoring longer sequences
- Beam search[15] uses $k = 5$ instead of $k = 4$
- Uses coverage penalty from Wu et al. (2016) during beam search
- Uses a reworked version of length normalization from Wu et al. (2016) during beam search
- They block the model from repeating trigrams during beam search

While analyzing the published test output[16] of Gehrmann et al. (2018), we find that they do not account for <t> and </t> tags when restricting output length during inference. As a result, their shortest and longest outputs are of length 28 and 94 respectively, where these are 35 and 120 for See et al. (2017).

[9]Repository with our evaluation of their outputs: https://github.com/ecly/see_et_al_2017_rouge
[10]https://pypi.org/project/py-rouge/
[11]The process of reducing a word to its stem/root. As an example, both *fishing* and *fisher* would stem to *fish.*
[12]We run their code with the following script: https://github.com/ecly/pointer-generator/blob/master/rerun.sh
[13]https://tensorflow.org
[14]http://opennmt.net/OpenNMT-py/Summarization.html
[15]Beam search is a heuristic search algorithm used to find the most likely summary during inference.
[16]https://github.com/sebastianGehrmann/bottom-up-summary

Pointer Generator (Our implementation) For our own implementation, we try to match the exact implementation and hyperparameters from See et al. (2017). As was also done for our rerun of the original implementation, we refrain from stopping and resuming the training to alter article and summary lengths. We train a Pointer Generator Model without coverage for 230,000, reporting ROUGE scores. We then train for another 3,000 iterations with both coverage and coverage loss enabled and evaluate again. When we add coverage to our model, we observe the similar initial 0.5 coverage loss, as reported by See et al. (2017). Additionally, our coverage loss reaches the reported 0.2 after the 3,000 coverage enabled iterations.

Moreover, we report the results of our implementation with added trigram repetition blocking, as used in Gehrmann et al. (2018), since we curiously found this to have comparable impacts on ROUGE scores to coverage.

	ROUGE		
Model	1	2	L
POINT-GEN (SEE ET AL., 2017)	36.44	15.66	33.42
POINT-GEN (SEE ET AL., 2017) (CORRECTED)	36.16	15.61	33.21
POINT-GEN + COV (SEE ET AL., 2017)	**39.53**	17.28	**36.38**
POINT-GEN + COV (SEE ET AL., 2017) (CORRECTED)	39.24	17.22	36.15
POINT-GEN (SEE ET AL., 2017) RERUN	36.59	15.80	33.51
POINT-GEN + COV (SEE ET AL., 2017) RERUN	39.11	17.13	36.02
POINT-GEN (GEHRMANN ET AL., 2018)	36.25	16.17	33.41
POINT-GEN + COV (GEHRMANN ET AL., 2018)	39.12	**17.35**	36.12
POINT-GEN (OUR IMPL.)	34.48	14.44	31.61
POINT-GEN (OUR IMPL. + TRIGRAM BLOCK)	38.17	16.39	35.38
POINT-GEN + COV (OUR IMPL.)	38.16	16.23	35.14
POINT-GEN + COV (OUR IMPL. + TRIGRAM BLOCK)	38.81	16.59	35.78

Table 2: Overview of full-length ROUGE F1-scores for Pointer Generator implementations. All models reported both with and without coverage.

From the results in Table 2, we find that our implementation performs slightly worse than the original implementation. While it is unclear where our reimplementation differs, and thereby where the ROUGE score discrepancy originates from, we deem our implementation sufficient to build upon. We also note, that the implementation from Gehrmann et al. (2018) relied on several different optimizations to achieve similar scores to the original, increasing our confidence in using our implementation as the basis for optimization and tuning.

5.4 Baselines

To investigate the quality of our improved models, we compare them to the following state-of-the-art baselines:

Lead-3 is one of the commonly used baselines for news summarization. The Lead-3 baseline curates the summary by taking the first 3 sentences of the document. For full-length document summarization, this provides a very strong baseline, which has only recently been beaten by neural abstractive approaches.

Pointer Generator + Coverage from See et al. (2017). This model extended the Pointer Network from Gu et al. (2016) with a coverage mechanism, which keeps track of attention coverage and penalizes repeated word-level attention. While it failed to beat Lead-3, it has inspired or even been at the core of many recent summarization model, such as Gehrmann et al. (2018), Hsu et al. (2018), Krycinski et al. (2018) and Celikyilmaz et al. (2018).

Bottom-Up Summarization from Gehrmann et al. (2018) extends the Pointer Generator model with a content selector. The content selector is trained separately, and used to modify the Pointer Generator's copy attention distribution. It does so by creating a mask, restricting the network to only point to the masked source.

Sentence Extract + Abstract from Chen and Bansal (2018) uses reinforcement learning to train a sentence extractor, which they combine with a Pointer Generator abstractor model, paraphrasing each sentence. The abstractor is trained with usual cross entropy, resulting in a multi-task learning objective. We compare to their best model, which they referred to as 'rnn-ext + abs + RL + rerank'.

5.5 Validation

To avoid overfitting ours models during training, we implement running validation using the CN-NDM validation set. Traditionally, validation is implemented by calculating average loss on the validation set. We instead find validation using our test-time scoring metric ROUGE, better for gauging a model's progress. As ROUGE is slower to compute than loss, since it requires running beam search, we ideally want to validate on as few samples as possible, while retaining accurate validation scores. We attempt to implement validation by randomly sampling instances from the validation set, running beam search and calculating ROUGE scores. As we need a single number to represent the quality of our models at runtime, we calculate the validation score as the harmonic mean of the F1-scores for ROUGE-1, ROUGE-2 and ROUGE-L. For choosing the smallest viable sample size, we conducted experiments for various various sizes, reporting the variance of the resulting ROUGE and validation scores. The results are shown in Table 3.

	ROUGE σ^2			Validation
Samples	1	2	L	Score σ^2
100	1.29	1.26	1.32	1.62
250	0.68	0.68	0.69	0.89
500	0.26	0.27	0.26	0.34
750	0.24	0.23	0.23	0.30
1000	0.14	0.11	0.13	0.15
2000	0.07	0.06	0.06	0.08

Table 3: Validation set variance (σ^2) on CNNDM's validation set (13,368 instances) of various sample sizes. Each sample size was tested 100 times, and we report the variance across the different metrics. The experiment was conducted using an arbitrary model, that achieved ROUGE-1: 38, ROUGE-2: 16 and ROUGE-L:35 on the CNNDM test set. Note that validation ROUGE scores are calculated using a Python reimplementation (py-rouge). This results in slightly different ROUGE scores than the official Perl implementation, but runs roughly 4x faster.

For a reasonably accurate indication of a model's quality, sample sizes of around 1,000–2,000 instances theoretically seem sufficient. In practise, we found that for scheduling purposes, such as reducing learning rates at plateaus, sampling was too noisy. As a result, our final models use ROUGE validation as described above, but run on all CNNDM validation instances after each epoch.

5.6 Optimization

To facilitate improvement at the neural summarization task, we seek to test a wide span of both expansions and variations of the Pointer Generator model. We introduce a base model, which we will describe in detail in the following section, producing similar ROUGE scores to See et al. (2017), using both less time and iterations for training. The reduced training time proves incredibly helpful, as we want to test several different hypotheses with limited resources and time.

Base model Our base model derives heavily from Pointer-Generator + Coverage, with a couple of additions and alterations. We hypothesize that by using pointing, smaller vocabularies suffice, as we merely need enough words to properly paraphrase most content. For this reason, and to speed up training, we use a vocabulary size of 10,000, in contrast to the 50,000 used originally. To further speed up training, we use 100 dimensional word embeddings (instead of 128), that we initialize with *Global Vector for Word Representation* (GLoVe) embeddings (Pennington et al., 2014) for words in our vocabulary that are also in the GLoVe data. For a vocabulary of size 10,000, we find GLoVe embeddings for all but 21 words, out of which 4 are special tokens such as UNK. Loading pretrained embeddings was found to both improve ROUGE scores and speed up progress in the early stages of training. For the Encoder and Decoder RNNs, we use GRU cells instead of LSTM cells, which were faster to train with complimentary increases in ROUGE scores. To prevent overfitting, we add dropout (Srivastava et al., 2014) with $p = 0.1$ to our embeddings and before our final prediction layer from Equation 12. In See et al. (2017), they refrained from training with coverage and coverage loss for the entire training duration, as they found it kept the model from focusing on its main objective. We however found it feasible to keep coverage enabled throughout training, using a coverage loss weight (λ) of 0.25. We also found that models trained significantly quicker using Adam (Kingma and Ba, 2014) (original uses AdaGrad) with a learning rate of 0.001. Similarly to Gehrmann et al. (2018), we reduce the learning rate whenever validation scores did not improve over one epoch. For validation, we follow Section 5.5, and validate on the full CNNDM validation dataset using the harmonic mean of ROUGE F1-scores. We reduce the learning rate by a factor of 10 when scores seize to improve. We similarly use the validation scores to implement early stopping with *patience* = 1. Lastly, we follow Gehrmann et al. (2018) and Paulus et al. (2017) during inference, preventing the model from repeating trigrams.

As our model was found too conservative with its output length, we modify the beam search from See et al. (2017) to increase minimum output length from 35 up to 45. We additionally follow Gehrmann et al. (2018), using the length penalty from Wu et al. (2016) with $\alpha = 2.2$, resulting in the following scoring of a sequence:

$$\begin{aligned} s(\mathbf{x}, \mathbf{y}) &= \log P(\mathbf{y}|\mathbf{x})/lp(\mathbf{y}) \\ lp(\mathbf{y}) &= \frac{(5 + |\mathbf{y}|)^\alpha}{(5+1)^\alpha} \end{aligned} \tag{21}$$

This beam search configuration was found to slightly improve ROUGE scores, and produces outputs of length closer to those of See et al. (2017).

By using a smaller vocabulary and embeddings, and with the use GRU cells instead of LSTM cells, our resulting base model has a total of 5,179,434 parameters, compared to the 21,501,265 from the original Pointer Generator. This allowed much quicker experimentation, as we found our base model trained substantially faster than our reimplementation of See et al. (2017), requiring 100,000 less iterations, with each iteration being roughly 4x faster.

Despite out commitment to smaller vocabularies, we do however test larger vocabularies, finding that it can slightly improve performance.

5.7 Tuning

The following subsections will briefly cover some of the hyperparameters that we tune and test with our base model. We also showcase some models, such as a model with LSTM cells, that motivated the specific configuration used for our base.

Activation From Equation 12, we observe that no non-linearity is used between the two projections used to form the final vocabulary distribution. Non-linear activation functions allow neural networks to learn more complex models, leading to our hypothesis that adding an activation to Equation 12 could yield performance improvements. We experiment with `TanH`, `ReLU` and `Sigmoid` activations, from which we find `TanH` the most prominent. We train a base model altering the vocabulary prediction as follows: $P_{vocab} = \text{softmax}(V' \tanh(V[s_t, h_t^*] + b) + b')$. We refer to this as BASE + TANH ACTIVATION.

Scheduled λ During training, we found that higher values for λ sped up learning for the first few epochs, while low values enabled more fine tuning. Based on this, we propose a scheduled coverage weight. We train a base model, where we scale λ by 0.75 each time validation scores decrease. We refer to this model as BASE + SCHEDULED λ.

Vocabulary size Following our hypothesis of smaller being sufficient when using pointing, we train base models with vocabulary sizes 5,000, 20,000 and 50,000, to properly find the most suitable size. We refer to these models as BASE + $|V| = \{5000, 20000, 50000\}$

Dropout For our base model we use dropout with $p = 0.1$ to prevent overfitting. To motivate this value for p, we also train models with $p = 0.25$ and $p = 0.5$, referred to as BASE + DROPOUT $p = \{0.25, 0.5\}$

GLoVe The base model has its word embedding weights initialized using GLoVe word vectors. To motivate this decision, we train a base model without this initialization, referring to it as BASE + W/O GLOVE.

A larger embedding dimension intuitively gives the model more space to represent the vocabulary. We train a model using 200-dimensional GLoVe embeddings, exploring whether the existing 100-dimensional embeddings suffice. We refer to this model as BASE + GLOVE200.

LSTM To motivate our choice of GRU cells for our RNNs, we also train a base model with LSTM cells, referred to as BASE + LSTM.

Optimizer In See et al. (2017); Gehrmann et al. (2018) they use AdaGrad for optimization of their parameters. To motivate our deviation from this, we train a base model using the AdaGrad (Duchi et al., 2011) configuration from See et al. (2017), with a learning rate of 0.15 and an initial accumulator value of 0.1. We further train a model using RMSProp (Tieleman and Hinton, 2012), to showcase the results of a different popular algorithm for gradient descent. We refer to the models as BASE + {ADAGRAD,RMSPROP}.

Multi-layer RNN Increasing model size has the potential of allowing more complex problems to be represented. Because of this, we investigate whether a multi-layered decoder RNN improves performance. We train a base model with a 2-layered decoder RNN and a dropout with $p = 0.1$ between the 2 stacked RNNs, referring to it as base + 2l-decoder.

Hidden Dimension While our base configuration already drastically reduces our model's parameter count, we investigate whether even less parameters still allow competitive performance. We train a model with half of the hidden size of See et al. (2017), namely 128. The resulting model has 2,772,138 parameters, 7 times less than the original Pointer Generator, and we refer to it as BASE + HIDDEN = 128.

5.8 Novel approaches

We propose several adaptations of the Pointer Generator, both in terms of model architecture and in terms of training, that we present in the following paragraphs.

Filtered vocabulary We hypothesize that a model utilizing pointing, has little need for proper nouns in its vocabulary. Intuitively, proper nouns are redundant for paraphrasing, as the exact noun generally should be used. As a result, we believe that proper nouns should always be copied, rather than generated. Thus, we experiment with vocabularies where proper nouns have been filtered. In theory, this allows smaller vocabularies to contain more useful words for paraphrasing, while keeping a consistent representation of proper nouns in the form of UNK .

To implement this, we modify the data preprocessing pipeline[17] from See et al. (2017) to include part-of-speech tagging using CoreNLP (Manning et al., 2014). We then use the tags to exclude proper nouns, hereby building a vocabulary based on word frequencies without words labeled as proper nouns. In Figure 6, we show the words that are removed/introduced because of this vocabulary building procedure, for a vocabulary of size 1,000. This comes out to a total of 109 words being different in the vocabulary, which increases to 1,693 words for a vocabulary of size 10,000. We train a base model for which we use the proper noun filtered vocabulary, which we refer to as BASE + FILTERED V.

Input filtering We suspect that some of the instances in the CNNDM dataset are of such low quality, that they provide no meaningful value during training. Because of this, we seek to filter out bad training instances, testing whether this improves performance or at least reduces training iterations. Examples of outliers, in the form of very short summaries, are shown in Figure 7. An example of an instance with a very large summary is shown in Appendix A.1, Figure 13. We also find that there exist instances with compression ratios < 1, making them very ill-suited for a

[17]https://github.com/abisee/cnn-dailymail

Removed union, university, james, queen, bbc, defense, facebook, christmas, july, michael, district, york, russia, arsenal, monday, labour, mail, obama, november, daily, january, miss, december, david, april, de, league, coast, france, ms, sunday, washington, u.s., county, central, martin, australia, justice, saturday, premier, committee, park, island, london, iraq, council, liverpool, smith, america, east, brown, october, airport, california, king, cameron, #, cup, mark, africa, middle, friday, june, china, paul, john, van, church, army, dr, germany, florida, mrs, george, wednesday, secretary, chelsea, prince, beach, uk, august, thursday, march, mr, bank, syria, college, green, apple, tuesday, september, tour, administration, prime, department, center, royal, europe, twitter, cnn, west, england, energy, bill, britain, february, manchester, united, english

Introduced convicted, birth, goes, offered, words, range, pm, land, alone, wo, allow, hour, vehicle, growing, built, denied, soldiers, interest, low, either, figures, beat, immediately, shown, 2007, faces, spoke, accident, opened, appears, believes, contributed, weather, companies, appear, walk, ended, loss, followed, probably, joined, reason, works, share, results, passed, protect, chinese, troops, main, changed, 23, fall, brain, customers, period, £, happen, herself, train, title, lived, especially, attention, refused, cars, simply, contact, coach, parts, impact, global, create, industry, offer, conditions, damage, stopped, ten, economic, else, boys, raised, events, towards, winning, hair, safe, scored, buy, opening, increase, 8, reach, allegations, investigators, 26, success, wedding, spending, signed, step, understand, seems, 9, door, eventually, questions, challenge

Figure 6: Words introduced/removed from V, when filtering proper nouns for $|V| = 1000$.

summarization task, per our definition of summarization. One such instance, out of a total of 26, is shown in Appendix A.1, Figure 14.

We first propose a filter, which solely removes extreme outliers:

$$\begin{gathered} 5 \leq M \leq 200 \\ 200 \leq N \leq 4000 \\ 2 \leq N/M \end{gathered} \tag{22}$$

where N is the number of tokens in the article, and M is the number of tokens in the summary. This results in a total of 380 instances filtered from the training data, and we refer to this configuration as BASE + FILTER (OUTLIERS).

We further propose a configuration that filters instances, where the summary's length is outside the inference bounds used in See et al. (2017).

$$35 \leq \ M \leq 120 \tag{23}$$

This filter configuration removes 37,983 instances and we refer to it as BASE + FILTER (SOFT).

Lastly, we propose a configuration that focuses our training on instances, with summaries of length close to the average output length from See et al. (2017) (˜61 tokens):

$$45 \leq \ M \leq 75 \tag{24}$$

This results in drastically fewer instances, removing 137,603 pairs, leaving us with a total of 149,510 instances. We refer to this configuration as BASE + FILTER (STRICT).

Attention mechanism To investigate whether the attention mechanism described in See et al. (2017) is ideal for the pointing architecture, we propose and evaluate two alternatives. In particular, we further examine the *general* and *dot* attention mechanisms described in Luong et al. (2015a):

$$\begin{aligned} dot: e_i^t &= s_t^\top h \\ general: e_i^t &= s_t^\top W_a h \end{aligned} \tag{25}$$

Article (777 tokens) here ' s to health and wellbeing (and gorgeous skin !) with the new year in mind . liz earle lifts a glass to the benefits of juicing the beauty benefits liz earle i have been singing the praises of juicing for 30 years and i am more excited than ever to share in my new book (see page 66) the many varied health , beauty and wellbeing benefits (. . .)

Summary (4 tokens) by liz earle .

Article (999 tokens) by david wilkes updated : 01:42 est , 26 october 2011 they look so cute it is almost impossible to believe these dogs have had their day . but the sealyham terrier , once beloved of hollywood stars and royalty , is now ' rarer than a tiger ' and on the verge of extinction. the staggering decline in the popularity of the little white dogs is highlighted in the latest edition of country life magazine as it sends an ' sos ' – that is ' save our sealyhams ' – message to its well–heeled readers (. . .)

Summary (6 tokens) sos : save our sealyhams .

Article (570 tokens) by leah simpson published : 16:46 est , 19 july 2012 — updated : 02:31 est , 20 july 2012 with the season finale airing on sunday night , the bachelorette star emily maynard is already making arrangements to extend her 15 minutes of fame - with a move to hollywood on the cards . but in the meantime , emily maynard is pinned down to north carolina (. . .)

Summary (4 tokens) bachelorette spoiler alert .

Figure 7: Articles with corresponding very short summaries from CNNDM's training data.

Both of these attention mechanisms were found to perform well for the tasks in Luong et al. (2015a). We train a base for each of these mechanisms, replacing GRU cells with LSTM cells for *dot*, as we would otherwise have to alter other parameters for $s_t^\top$ and h to be compatible for matrix multiplication. As none of these attention mechanism incorporate coverage, we propose novel coverage variants of both. Here we bias them on a projection of the coverage at the current time step:

$$\begin{aligned} \textit{dot} + \textit{coverage} &: e_i^t = s_t^\top h + w_c c_t \\ \textit{general} + \textit{coverage} &: e_i^t = s_t^\top W_a h + w_c c_t \end{aligned} \tag{26}$$

We found that our coverage variants converged 2–3 epochs faster than their base variants. We also found that coverage improved scores when evaluated without trigram repeat blocking, indicating that the coverage bias helps. Similar to our Pointer Generator reimplementation, trigram repetition blocking was however found to be as efficient as coverage. Because of this, resulting scores were similar both with and without coverage, hence we only report the scores for our coverage variants. We refer to these models as BASE + {DOT,GENERAL} + COV.

Managing unk We notice that some models, especially with small vocabularies, occasionally output UNK tokens. With pointing enabled, we rarely expect target summaries to contain words that are neither in the known vocabulary or in the corresponding article. This can however occur, as we observe some training instances where summaries reference entities that are not mentioned in the article. This may stem from the nature of the highlights from CNNDM, where one such highlight could be a quote from some entity separate from the article. As a result, our models occasionally learn to predict UNK, which however never improves scores at test time. We therefore propose three methods to explicitly manage UNK.

- BLOCK UNK. Here we simply modify inference, to prevent our model from predicting UNK.
- PEN. UNK. We propose to penalize UNK predictions during training, by adding the sum of UNK probabilities to the total loss. Since our targets can contain UNK, this puts our model in a situation where it is penalized no matter what it predicts. Because we additionally ignore the loss for any UNK tokens in the target.
- SAMPLE UNK. Since we use strict teacher forcing during training and targets can contain UNK, we occasionally use UNK as input. At test time we never want the model to predict UNK, making it redundant to teach the decoder to handle UNK inputs. As such, we propose to use our model's prediction from $t-1$ rather than $\bar{w}^*_{t-1}$ as input for the decoder, whenever w^*_{t-1} is UNK. Note that this is only useful when our prediction from $t-1$ is in V, as the alternative otherwise again becomes UNK. We consider this an extension of PEN. UNK, applied on top of it.

5.9 Reproducability

To ensure reproducability of our results, our published repository include configuration files that can be used to rerun each experiment discussed throughout the paper. While we strive for full reproducability, by seeding all sources of randomness, we do rely on some non-deterministic PyTorch code[18]. Although PyTorch supports disabling these non-deterministic features, this slows down the execution of some otherwise fully parallelizable tasks. We therefore choose not disable these features,. As a result, the scores we present cannot be deterministically replicated using our configuration files, but based on a few experiments, variance seem insignificant.

5.10 Experimental Results

We present an overview of all ROUGE scores, as well as training iterations used for both tuned models and novel approaches in Table 4.

From our experiments we find that several systems improve on our base model. Defining improvement as having beaten the base across every ROUGE metric, we improve for 4 tuning models and 2 novel models.

Tuning For the tuned models, we improve on the base for BASE + TANH ACTIVATION, BASE + $|V| = \{20000, 50000\}$, and BASE + 2L-DECODER. We generally find that our base configuration is well-tuned as is, but note that larger vocabularies, as well as a stacked decoder, can lead to improved scores. We find that the only tune that improves the base without altering parameter count, is the addition of a non-linearity prior to our final prediction layer.

Novel For our novel approaches, we confirm the hypothesis that filtering proper nouns from V has limited impact on a model utilizing pointing. We also find that filtering outlier instances can lead to a non-trivial decrease in training iterations needed for convergence, be it at a small cost in ROUGE scores. In the future, it would be interesting to try removal of random instances, instead of targeting instances with certain length attributes, to see how that impacts the resulting scores. For our novel attention mechanisms with coverage adaptations, we find that they show promise, but fail to improve on the attention mechanism from See et al. (2017). Lastly, we find that both

[18] https://pytorch.org/docs/stable/notes/randomness.html

	ROUGE			Training
Model	1	2	L	Iterations
BASE	39.03	17.01	36.25	126K
BASE + TANH ACTIVATION	39.17	17.04	36.33	126K
BASE + SCHEDULED λ	39.07	17.01	36.34	179K
BASE + $\|V\| = 5000$	38.93	16.95	36.12	126K
BASE + $\|V\| = 20000$	**39.22**	**17.14**	36.38	90K
BASE + $\|V\| = 50000$	39.19	17.09	**36.42**	90K
BASE + DROPOUT $p = 0.25$	39.09	16.99	36.22	126K
BASE + DROPOUT $p = 0.5$	39.12	16.98	36.07	90K
BASE + W/O GLOVE	38.83	16.78	36.04	90K
BASE + GLOVE200	39.06	16.98	36.24	126K
BASE + LSTM	38.94	16.99	36.14	90K
BASE + ADAGRAD	38.63	16.60	35.73	90K
BASE + RMSPROP	39.00	16.92	36.20	144K
BASE + 2L-DECODER	39.17	17.12	36.28	90K
BASE + HIDDEN = 128	38.96	16.96	36.15	90K
BASE + FILTERED V	39.01	16.96	36.17	90K
BASE + FILTER (OUTLIERS)	39.04	16.98	36.23	108K
BASE + FILTER (SOFT)	38.86	16.87	36.00	78K
BASE + FILTER (STRICT)	38.77	16.80	35.89	65K
BASE + DOT + COV	38.67	16.71	35.84	72K
BASE + GENERAL + COV	38.93	16.91	36.15	89K
BASE + BLOCK UNK	39.12	17.03	36.33	126K
BASE + PEN. UNK	39.11	16.97	36.30	179K
BASE + SAMPLE UNK	39.20	17.06	**36.42**	144K

Table 4: Overview of experimental results. All models are evaluated using official Rouge 155 Perl script[19]. We report full length F1-scores and total training iterations. Base model shown at the top, followed by tuning models, lastly followed by models with a novel element. Iterations are rounded to nearest thousand.

BASE + BLOCK UNK and BASE + SAMPLE UNK improve on the base model. Simply blocking the prediction of UNK is useful, as it can be retroactively applied to an already trained model, while using our model's prediction instead of teacher forcing UNK, shows the most promise. We consider all of our approaches for explicitly managing UNK meaningful, as the prediction of UNK has no value in any actual application of summarization.

In Section 7, we attempt to combine some of the most promising configurations and compare the results to our baselines.

6 Summarization Beyond Words

Neural summarization research has generally been limited to producing relatively short summaries of either truncated or short text, due to the inherent computational complexity of both training and inference. Because of this, we propose a new way of summarizing, that first coarsely filters the input document and then divides the remainder up into *chunks*, that we operate on instead of words. For this, we adapt the Pointer Generator architecture to allow copying of chunks instead of words, which has the advantage of ensuring grammatical consistency within the chunk. This approach implicitly

[19]https://pypi.org/project/pyrouge/

enables predictions of multiple words, effectively reducing the amount of predictions needed to create some summary $\mathbf{y}$.

To enable summarization on chunked input, we use a multi-task approach for (1) detection of chunks and (2) generation of summaries. We also introduce and experiment with a third task (3) chunk reconstruction, where we attempt to retain as much information about the sequence of words within a chunk as possible. An overview of the approach is presented in Figure 8.

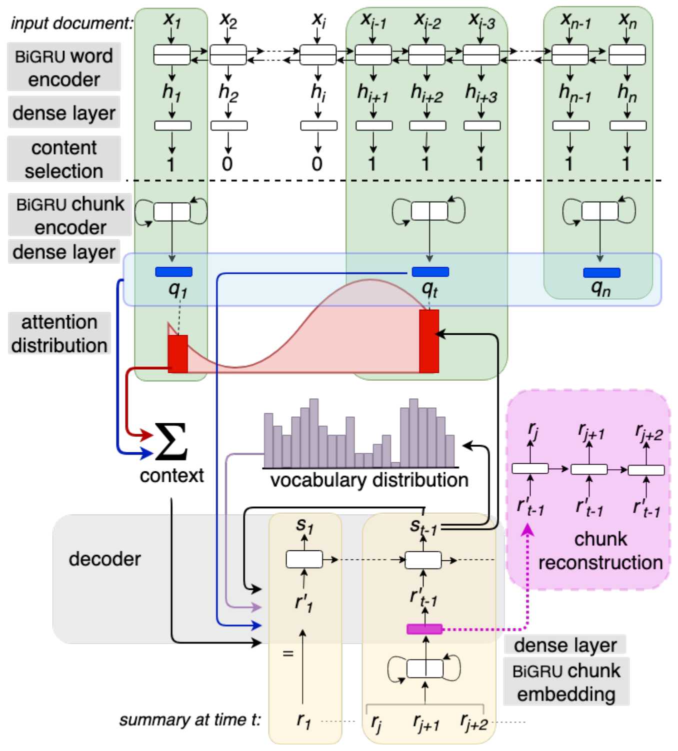

Figure 8: The chunking architecture, as presented in Lynegaard and Schluter (Submitted).

6.1 Chunk detection

For detecting chunks, we reuse the encoded sequence h from our original architecture, formulating the detection as a word-level binary tagging task, as done for the content selector in Gehrmann et al. (2018). For each encoded word h_i, we consider it either discarded (0) or selected (1), by its satisfaction of the following predicate:

$$\epsilon \leq \sigma(h_i W_p W_s + b_s) \tag{27}$$

where W_p, W_s and b_s are learnable parameters. Based on the resulting binary sequence, we define a chunk as any consecutive sequence of 1s.

As there exists no supervised data for the task, we use the algorithm and implementation[20] from Gehrmann et al. (2018), for creating supervised alignment data between the document and its summary. Here we define a word x_i as selected if it is (1) part of the longest possible subsequence of words $\mathbf{s} = x_{j:k}$ where $(j \leq i \leq k \leq N)$ if $\mathbf{s} \in \mathbf{x}$ and $\mathbf{s} \in \mathbf{y}$ and (2) there exists no earlier $\mathbf{u}$ with $\mathbf{s} = \mathbf{u}$. During training, we use binary cross entropy to compute an *alignment loss* term, which we add to our existing loss term described from Equation 17.

[20]https://github.com/sebastianGehrmann/bottom-up-summary

6.2 Chunk representation

To compute a contextual representation of the ith chunk c_i, corresponding to $c_i = (h_j, h_{j+1}, \ldots, h_{j+|c_i|})$, we feed each of the encoded words within c_i through a bidirectional GRU *chunk encoder*. We then define the chunk's contextual representation q_i as a projection of the concatenation of the last hidden state for each direction of the chunk encoder, given c_i. To make our chunk encoder compatible with the rest of the model, we use *hidden size* = 256, which is used original Pointer Generator Encoder.

As the Pointer Generator model uses the previous time step's ground truth embedding (w^*_{t-1}) as decoder input at time step t, we compute a compatible *chunk embedding* for each chunk. Intuitively, we aim to capture the exact contents of the chunk, ignoring the context of the chunk in $\mathbf{x}$. To do this, we use a new bidirectional GRU. We use the same approach as the chunk encoder, but operate on the embeddings of the words within the chunk instead of their contextually encoded representation h. The chunk embedding r'_i is thereby computed with a separate *chunk embedder*, based on the embeddings for the words corresponding to c_i. From this, at time step t, our decoder now receives $r'_t - 1$, which corresponds to either a word or a chunk. If we generated a word at $t-1$, we do as for the original Pointer Generator, and lookup r'_{t-1} in our embedding matrix. If we instead copied a chunk at $t-1$, we lookup the corresponding chunk embedding instead.

6.3 Chunk reconstruction

To encourage our model to retain as much information as possible about the words within our chunk embeddings, we propose chunk reconstruction. Here we use a unidirectional GRU, to predict the sequence of words within the chunk, for all chunks made from our input $\mathbf{x}$. For a chunk r'_i, we predict the sequence of words within the chunk, by feeding our GRU a projection of the concatenation of r'_i and the prediction from t', where $t' < |r'_i|$, using a special SOS token for $t = 0$. The loss for a single chunk, is then computed as the average cross entropy loss across the words within the chunk sequence. We define the complete chunk reconstruction loss term *reconloss*, as the average cross entropy loss across all the chunks made from $\mathbf{x}$. To avoid reconstruction loss being the dominant figure in our total loss term, we scale it by some κ before adding it to our existing loss term. This gives us our final loss term:

$$\text{loss} = \frac{1}{M}(\sum_{t=0}^{M} \text{loss}_t + \lambda \ \text{covloss}_t) + \text{alignloss} + \kappa \ \text{reconloss} \tag{28}$$

where M is the length of the target summary, alignloss is the binary cross entropy loss from Section 6.1 and reconloss is our just defined, averaged cross entropy reconstruction loss term.

6.4 Chunked Pointer Generator

To apply Pointer Generator to our chunked architecture, we first alter the attention energy calculation from Equation 16 to operate on our sequence of encoded chunks q:

$$e^t = v \ \tanh(W_q q + W_s s_t + w_c c^t + b_{attn}) \tag{29}$$

We also change our context vector (h^*_t) calculation from Equation 11, representing it as a weighted sum of chunk encodings:

$$q^*_t = \sum\nolimits_i \alpha^t_i q_i \tag{30}$$

Our vocabulary predictions from Equation 12 are now made using the chunked context vector q_t^*:

$$P_{vocab} = \text{softmax}(V'(V[s_t, q_t^*] + b) + b') \tag{31}$$

Lastly, we alter our soft switch calculation of p_{gen} from Equation 13, to use q_t^*:

$$p_{gen} = \sigma(w_{q^*} q_t^* + w_s s_t + w_x x_t + b_{ptr}) \tag{32}$$

leaving the rest of the equations as is.

As our target summaries originally contain no chunks during training, we modify them at runtime, replacing occurrences of subsequences corresponding to a chunk, with the corresponding chunk's unique identifier. We prioritize longer chunks while applying this replacement, by doing it for all chunks in descending order of chunk size. The significant length reductions achieved by chunking both inputs and targets is shown in Figure 9.

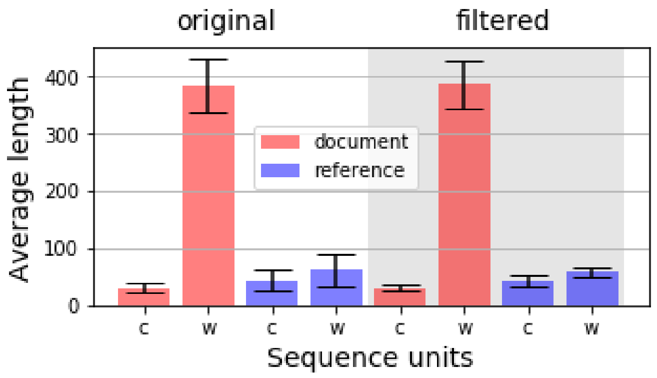

Figure 9: Reduction in sequence length of articles/summaries following chunking. The *filtered* section shows reduction in sequence lengths, using the *(strict)* filter defined in Section 5.8. Figure as seen in Lynegaard and Schluter (Submitted).

6.5 Chunk experiments

We implement our chunking architecture on top of our existing base model implementation. Similarly, we derive hyperparameters from the base model, wherever otherwise specified.

As we hypothesize that the introduction of chunk embeddings into our existing embedding space requires additional space, we train models with both embedding dimensions 100 and 200, with and without reconstruction loss. We compare our chunk models to the original Pointer Generator model with coverage from See et al. (2017), as well as our base model from Section 5.6, with 100 and 200 dimensional embeddings. We additionally showcase the results for the 200 dimensional chunk model with reconstruction loss, using the reference content selection, which we denote the oracle model. As chunk models were found to achieve their best validation scores very quickly, we limit ourselves to 100,000 training instances. We also found that chunk models were sensitive to the length of their input, resulting in the selection of the 100,000 instances using the *(strict)* filter described in Section 5.8. For each model, we tune both α and ϵ on a subset of 2,000 instances from the CNNDM validation set. During training, we use $\kappa = 0.1$ for all chunk models and increase dropout to $p = 0.5$. We also reduce λ to 0.1 during training, as we intuitively want to allow some repeated attention to each chunk, in case we need to generate words for stitching together copied chunks. The overview of scores is presented in Table 5.

Model	ROUGE 1	ROUGE 2	ROUGE L	Training Iterations
POINT-GEN (See et al., 2017) (corrected)	39.24	**17.22**	36.15	233K
BASE	39.03	17.01	**36.25**	126K
BASE + GLOVE200	39.06	16.98	36.24	126K
CHUNK100	**39.72**	16.34	36.22	6K
CHUNK100 + RECONLOSS	39.13	16.30	35.77	6K
CHUNK200	38.75	16.13	35.37	6K
CHUNK200 + RECONLOSS	39.59	16.38	36.00	6K
ORACLE-CHUNK200 + RECONLOSS	57.22	37.05	53.89	6K

Table 5: Chunking results on CNNDM. Iterations rounded to nearest thousand. Oracle model slightly grayed, to emphasize that we cannot compare directly to its scores.

We observe that our chunk based approaches are able to outperform both the best model from See et al. (2017) and our base model in the ROUGE-1 metric, using far less iterations (1 epoch). The oracle model showcases the value of proper chunk selection, illustrating a significant room for improvement by solely enhancing the chunk selection component. It is however important to note, that the chunk based approaches suffer significant reductions in ROUGE-2 scores, which we study in Section 8.

7 Results

We compare our results to the baselines described in Section 5.4. Here we focus on our base model, as well as the two best chunk models CHUNK100 and CHUNK200+RECONLOSS. We additionally present the model BASE + $|V| = 20000$ + BLOCK UNK, which we found was the best combination we could make, using the improvements presented in Section 5.10. Several different combinations of base experiments were tried, mostly finding that improvements did not accumulate or that scores even worsened. The baseline scores, as well as our best scores are presented in Table 6.

Model	ROUGE 1	ROUGE 2	ROUGE L
LEAD-3	40.10	17.50	36.30
POINT-GEN + COV* (See et al., 2017)	39.24	17.22	36.15
BOTTOM-UP (Gehrmann et al., 2018)	**41.22**	**18.68**	38.34
SENT-EXT + ABS (Chen and Bansal, 2018)	40.88	17.80	**38.54**
BASE	39.02	17.01	36.25
BASE + $\|V\| = 20000$ + BLOCK UNK	39.26	17.16	36.42
CHUNK100	39.72	16.34	36.22
CHUNK200 + RECONLOSS	39.59	16.38	36.00

Table 6: Results on CNNDM. We compare our base, our best combination of experiments, and our best chunking systems to our baselines. *Note that we use the corrected scores from See et al. (2017) as described in Section 5.3.

While we only successfully compete with one of our baselines, namely See et al. (2017), we consider both our base, the accompanying experiments and our chunking architecture as notable contributions to the summarization task. We significantly reduce required training time, first for our optimized base model and further for our chunking systems. In Figure 10, we show summaries generated by our base model vocabulary size 20,000 and unknown blocking, as well as those generated by our chunk model using 200 dimensional embeddings and reconstruction loss. We display these summaries alongside both the reference summary and the summary generated with the original Pointer Generator.

Article (truncated):
new delhi (cnn) an indian software pioneer and nine others have been sentenced to seven years in jail for their role in what has been dubbed india 's biggest corporate scandal in memory , police said . ramalinga raju , the former chairman of software services exporter satyam computers services , was also fined $ 804,000 , r.k. gaur , a spokesman for india 's central bureau of investigation , told cnn . in 2009 , satyam computers services was at the center of a massive $ 1.6 billion fraud case after its then-chairman raju admitted inflating profits with fictitious assets and nonexistent cash (. . .)

Reference Summary satyam computers services was at the center of a massive $ 1.6 billion fraud case in 2009 . the software services exporter 's chairman , ramalinga raju , admitted inflating profits . satyam had been india 's fourth-largest software services provider .

POINT-GEN + COV ramalinga raju was also fined $ 804,000 , a spokesman for india 's central bureau of investigation . investigators say losses to investors resulting from the company 's book manipulation were much higher . the case has been compared to the 2001 enron corp. scandal .

BASE + $|V| = 20000$ + BLOCK UNK ramalinga raju , the former chairman of software services exporter satyam computers services , was also fined $ 804,000 . in 2009 , satyam computers was at the center of a massive $ 1.6 billion fraud case . the case has been compared to the 2001 enron corp. scandal , in which a houston energy company 's earnings had been overstated by several hundred million .

CHUNK200 + RECONLOSS has been dubbed india 's biggest corporate scandal in memory , police say . ramalinga raju , the former chairman of software services exporter satyam computers services , was also fined $ 804,000 , r.k. gaur , a spokesman for india 's satyam computers services was at the center of a massive $ 1.6 billion fraud case after

Figure 10: Example of summaries generated by various models on a CNNDM document from the test set.

7.1 NYT

We report scores on the NYT dataset in Table 7 to facilitate comparison with future work. For preprocessing, we follow Paulus et al. (2017), using scripts supplied by the authors. At inference time, we follow the beam search configuration from Gehrmann et al. (2018). We observe that our base model consistently beats the Pointer Generator reimplementation from Gehrmann et al. (2018), across all ROUGE metrics.

	ROUGE		
Model	1	2	L
RL + INTRA-ATTENTION (Paulus et al., 2017)	47.03	30.72	**43.10**
DCA (Celikyilmaz et al., 2018)	**48.08**	**31.19**	42.33
BOTTOM-UP (Gehrmann et al., 2018)	47.38	31.23	41.81
POINT-GEN (Gehrmann et al., 2018)	45.13	30.13	39.67
BASE	45.23	30.41	39.85

Table 7: Results on NYT dataset. We use the models compared to in Gehrmann et al. (2018) as baselines, only reporting our score for the base model, due to both time and resource limitations.

7.2 Newsroom

In Table 8 we present scores for a base model trained and evaluated on the Newsroom dataset. As we experienced numerical instability during training, we found it necessary to use an initial learning rate of 0.0001 instead of the 0.001 used for the base. For inference, we alter the minimum summary length for inference to 25 instead of the 45 used for the base, to enable shorter summaries, as these are present in the Newsroom corpus. We observe that we confidently beat the best scores reported on Newsroom in Grusky et al. (2018), which makes us question the tuning/training procedure used for obtaining the reported Pointer Generator scores. We leave it to future work, to properly train and evaluate models on the Newsroom corpus, proposing our base model as a benchmark.

	ROUGE		
Model	1	2	L
LEAD-3 (Grusky et al., 2018)	31.31	21.59	29.03
POINT-GEN (See et al., 2017)	26.99	13.44	23.02
TEXTRANK (Barrios et al., 2016)	23.82	9.98	19.57
SEQ2SEQ + ATTN. (Rush et al., 2015)	6.10	0.44	5.51
BASE	**35.81**	**23.83**	**32.85**

Table 8: Results on the Newsroom dataset. We compare to the stemmed scores reported at the Newsroom website[21]. We provide results only for our base model, due to a lack of time and resources. Preprocessing consisted of lowercasing and tokenization with NLTK[22].

8 Discussion

In both our base experiments and our chunk experiments, we encounter several peculiarities that we deem worthwhile discussing. We present several of our thoughts and concerns in the following section, as well as discuss the current state of supervised data and automatic evaluation metrics for summarization.

[21]https://summari.es

[22]https://www.nltk.org/

8.1 Pointer Generator

We first note, that our reimplementation of Pointer Generator achieves slightly lower scores than the original. While we are unable to replicate the original training procedure exactly, we still observe lower scores than our reproducable rerun of the original implementation. Despite our best efforts, we did not manage to identify any differences between our implementation and the original (other than the use of PyTorch instead of TensorFlow), that could explain this discrepancy.

Based on Luong et al. (2015a), we propose two new attention mechanisms for the Pointer Generator task, providing coverage adaptations for both. While both of these yield lower scores than the base, we still consider future work into attention mechanisms for the Pointer Generator worthwhile.

From our experiments on the base model, we find that a vocabulary of size 20,000 works well on the CNNDM corpus. We note that this may possible bade made smaller, using a filtered version, where proper nouns are excluded from the vocabulary, as these were found to be unnecessary for our pointer based model.

Given our analysis of the CNNDM dataset, we find several instances that we consider to be ill-suited for learning summarization. Results when filtering these instances, indicate a small reduction in required iterations for convergence, but fail to improve scores.

Our experiments for explicit management of the UNK token indicate, that proper treatment of words that remain OOV, despite using the extended vocabulary of pointer systems, provides improvement at test time. We consider the prevention of generation of UNK tokens at inference time an obvious addition to any pointer based system.

Lastly, we uncovered several smaller tuning improvements to our base configuration. While we hoped that our improvements would accumulate once combined, we found that this was mostly not the case. We do however note that the expansion found most suitable for combination with other features, was the explicit management of UNK tokens.

8.2 Chunking

As the chunk based approach is the first of its kind, to our best knowledge, several uncertainties remain regarding the architecture and training thereof. Firstly, we find that reconstruction loss greatly improves the 200 dimensional embedding model, while it reduces performance for the 100 dimensional embedding model. Further experimentation is needed to properly determine whether reconstruction loss is a productive addition to the training procedure. While chunk based models yield impressive ROUGE-1 scores using few training iterations, we note that they get lower ROUGE-2 scores, which results in reduced coherence and grammatical correctness of the summaries. An example of an incoherent summary is presented in Figure 11, with more showcased in Appendix A.2.

crash site , " he says capt. yves naffrechoux of the high mountain gendarmerie . the plane crashed march 24 in rugged terrain of the alps about 6 miles . . investigators are not expected to return to the crash site all the main evidence from the site .

Figure 11: Example of an incoherent summary generated by a chunk model on CNNDM test document.

From the examples we observe that much work is still needed to effectively teach chunk based systems, to naturally stitch together copied chunks.

Although the reduction in training time may be explained by the task of stitching together chunks being easier than the original task, we suspect that further training may enable better coherence, despite our decreasing ROUGE validation scores. The decreasing ROUGE validation scores may result from ROUGE validation requiring a preset ϵ for chunk systems. An alternative approach, may compute validation scores using the golden chunks, which would not require a preset ϵ. We believe that future experiments, given proper configuration, will improve through several extra epochs.

Another interesting chunk model variation, could factor the content selector into a separately trained system, as in Gehrmann et al. (2018), retaining the rest of the chunking architecture. We also consider a mixed learning objective a possibility for increasing coherence and grammatical correctness, where the model is jointly optimized for cross entropy and ROUGE-2. Similar mixed learning objectives were found to work well for other models in recent work (Cf. Section 3).

Lastly, we suspect that robustness to varying target lengths could be improved, given further tuning of the training procedure. This would allow the use of the full, or a less strictly filtered, CNNDM dataset, which may prove useful for future configurations.

In Figure 12 we show the spread of chunk sizes during inference for all chunk models, given tuning of ϵ for ROUGE scores on the CNNDM validation set.

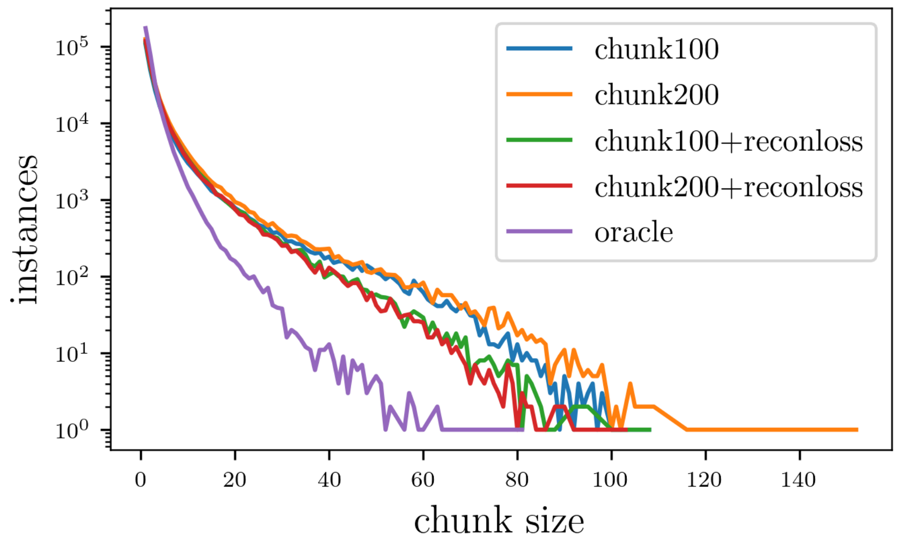

Figure 12: Histogram of chunk sizes during inference for all our chunk models as well as the oracle. Our models average chunk size lie between 4–4.5 words with a standard deviation between 6–7.5. In contrast, the oracle has an average chunk size of 2.28 with standard deviation of 2.59. We see that the oracle produced 173K chunks of size 1, where our models created 110K-126K such chunks. Instead our models produce substantially more chunks of size ≥ 4 than the oracle.

As our tuning did not lead to larger values of ϵ, which leads to fewer and smaller chunks, we suspect that our content selection ends up filtering too much important content, if ϵ gets too large. This is however feasible for the oracle, as it knows exactly what parts of the input to select. We imagine that future improvements in content selection, or extractive summarization in general, can translate almost directly into corresponding improvements for the semi-abstractive chunking architecture.

8.3 Summarization

While several advances are being made within the summarization field, we consider a main hindrance to be the lack of better abstractive supervised data, as well as better automatic evaluation metrics. From Grusky et al. (2018) and Figure 5, we note that both the NYT and Newsroom datasets contain summaries exhibiting much more diverse summarization strategies than CNNDM. Future work in summarization should consider including results across both of these corpora, possibly investigating which corpus lead to best correlation between ROUGE scores and human evaluation.

While we have not presented scores for other evaluation metrics than ROUGE, we note that especially METEOR[23] (Denkowski and Lavie, 2014) is often used alongside ROUGE in recent work. Although METEOR was introduced for machine translation, it integrates robustness to both paraphrasing and synonyms, possibly resulting in improved correlation with human evaluation at the summarization task.

9 Conclusion

We have presented and reimplemented the Pointer Generator architecture from See et al. (2017). In doing so, we have found and shown parts of the network, that were not made clear or pointed out originally. From our reimplementation, we have built an optimized version of the Pointer Generator, that maintains its high ROUGE scores, using far less training time. We further propose and evaluate several tunes and extensions of the architecture, some yielding notable results. While several of the experiments discover useful properties, we emphasize the benefits of initializing word embeddings with GLoVe embeddings, explicitly handling UNK tokens, and the ability to maintain high ROUGE scores while using a vocabulary partially devoid of proper nouns.

We further propose the novel multi-task Pointer Generator architecture, using a both filtered and chunked representation of the input. Adopting and repurposing the content selection from Gehrmann et al. (2018), we illustrate the significant reductions in size of both input and output, that chunking enables. Furthermore, we introduce a novel reconstruction loss term, guiding the network into producing chunk embeddings, that retain as much linguistic integrity as possible. Despite producing impressive ROUGE scores, we find that further work is needed, to consistently produce coherent summaries using chunk based approaches. We consider the incorporation of chunking into the Pointer Generator, a computationally efficient basis for future work on summarization.

[23]METEOR: **M**etric for **E**valuation of **T**ranslation with **E**xplicit **Or**dering

References

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*, 2015.

F. Barrios, F. López, L. Argerich, and R. Wachenchauzer. Variations of the similarity function of textrank for automated summarization. *CoRR*, abs/1602.03606, 2016. URL http://arxiv.org/abs/1602.03606.

A. Celikyilmaz, A. Bosselut, X. He, and Y. Choi. Deep communicating agents for abstractive summarization. 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 1662–1675, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N18-1150.

Y.-C. Chen and M. Bansal. Fast abstractive summarization with reinforce-selected sentence rewriting. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 675–686, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P18-1063.

K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In A. Moschitti, B. Pang, and W. Daelemans, editors, *EMNLP*, pages 1724–1734. ACL, 2014. ISBN 978-1-937284-96-1. URL http://aclweb.org/anthology/D/D14/D14-1179.pdf.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS - Neural Information Processing Systems*, 2014. URL http://arxiv.org/abs/1412.3555. Presented in NIPS 2014 Deep Learning and Representation Learning Workshop.

M. Denkowski and A. Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*, 2014.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2021068.

S. Gehrmann, Y. Deng, and A. Rush. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109, Brussels, Belgium, Oct.-Nov. 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D18-1443.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Y. Graham. Re-evaluating automatic summarization with BLEU and 192 shades of ROUGE. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 128–137, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1013. URL https://www.aclweb.org/anthology/D15-1013.

M. Grusky, M. Naaman, and Y. Artzi. Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies. *CoRR*, abs/1804.11283, 2018. URL http://arxiv.org/abs/1804.11283.

J. Gu, Z. Lu, H. Li, and V. O. Li. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1154. URL https://www.aclweb.org/anthology/P16-1154.

K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. 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.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

W.-T. Hsu, C.-K. Lin, M.-Y. Lee, K. Min, J. Tang, and M. Sun. A unified model for extractive and abstractive summarization using inconsistency loss. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 132–141, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P18-1013.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

W. Krycinski, R. Paulus, C. Xiong, and R. Socher. Improving abstraction in text summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1808–1817, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D18-1207.

C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In S. S. Marie-Francine Moens, editor, *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

L. Liu, Y. Lu, M. Yang, Q. Qu, J. Zhu, and H. Li. Generative adversarial network for abstractive text summarization. *AAAI 2018*, 2018. URL https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16238.

T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, Sept. 2015a. Association for Computational Linguistics. doi: 10.18653/v1/D15-1166. URL https://www.aclweb.org/anthology/D15-1166.

T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba. Addressing the rare word problem in neural machine translation. In *ACL*, 2015b.

E. Lynegaard and N. Schluter. Sequence to sequence abstractive summarisation beyond words. In *Empirical Methods in Natural Language Processing (EMNLP)*, Submitted. URL http://www.aclweb.org/anthology/D14-1162.

C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014. URL http://www.aclweb.org/anthology/P/P14/P14-5010.

R. Nallapati, F. Zhai, and B. Zhou. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. *CoRR*, abs/1611.04230, 2017. URL http://arxiv.org/abs/1611.04230.

S. Narayan, S. B. Cohen, and M. 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, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1158. URL https://www.aclweb.org/anthology/N18-1158.

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

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

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

E. Sandhaus. The new york times annotated corpus, 2008. URL https://catalog.ldc.upenn.edu/LDC2008T19.

A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics. URL http://aclweb.org/anthology/P17-1099.

N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1): 1929–1958, 2014. URL http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. 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.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5866-pointer-networks.pdf.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. 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.

A Appendix

A.1 Long summaries (CNNDM)

Article (390 tokens) by . david kent . as the 2014 world cup continues , sportsmail will be providing you with all you need to know about every fixture in brazil from team news and key battles to betting odds and opta stats ... here is all the information you need for the game in group b between spain and chile . click here to follow the spain vs chile world cup 2014 action live . venue : estadio jornalista mario filho , rio de janeiro . kick-off : 8pm -lrb- 4pm , brazil time -rrb- . tv coverage : bbc1 , from 7.30 pm . odds : spain 4/7 , draw 10/3 , chile 9/2 . referee : mark geiger -lrb- usa -rrb- . managers : vicente del bosque -lrb- spain -rrb- jorge sampaoli -lrb- chile -rrb- . video team profile : spain . humiliated : spain 's players react after they were beaten 5-1 by holland on friday night . one to watch : jorge valdivia . alexis sanchez and arturo vidal tend to hog the limelight for chile , while the good work of players like 30-year-old valdivia goes unnoticed . watch the playmaker in action and look at the way he moves with the ball and switches possession and you would think he was spanish . he 'll need to be at his best to match xavi , xabi alonso and sergio busquets though . key clash : javi martinez vs alexis sanchez . if , as expected , bayern munich 's javi martinez comes in for the out-of-sorts gerard pique alongside sergio ramos at the heart of spain 's defence then he will have his work cut out keeping the lively alexis sanchez quiet . martinez , a central midfielder by trade , will have to cope with the lightning pace of barcelona forward if spain are to stay in the tournament . best bet : . chile 's juventus midfielder arturo vidal to score at any time – 4/1 . head-to-head record : . played 10 , spain wins 8 , draws 2 . considering changes : spain head coach vicente del bosque saw his side lose 5-1 to holland . video all star xi : andres iniesta . opta stats : . main man : barcelona forward alexis sanchez leads the line for the chileans .

Summary (382 tokens) spain take on chile in both sides ' second group b match . holders spain could go out if they lose after being thrashed 5-1 by holland . chile beat australia in their first game and could go through with a win . chile have never beaten spain in their 10 previous encounters -lrb- d2 l8 -rrb- . spain have won the two previous meetings with chile at the world cup : in 1950 -lrb- 2-0 -rrb- and in 2010 -lrb- 2-1 -rrb- . the game in 1950 was played at the maracanã ; which is where this encounter is taking place . chile have lost their two previous world cup games against the defending champion , both against brazil -lrb- in 1962 and 1998 -rrb- . in their group opener against the netherlands , spain conceded 5 + goals in an international for the first time since losing 6-2 to scotland in june 1963 . only once before had spain conceded 5 + goals in a world cup game -lrb- 1-6 vs brazil in brazil in 1950 -rrb- . spain lost their opening game of the 2010 world cup finals -lrb- 0-1 v switzerland -rrb- but became the first team to ever lift the world cup trophy after having lost their opening game of the tournament . spain 's 1-5 defeat to the netherlands was the heaviest margin of defeat by a reigning world cup champion in the history of the tournament -lrb- four goals -rrb- . the previous heaviest were brazil losing 0-3 v france in 1998 and west germany losing 3-6 v france in 1958 . alexis sanchez has been involved in seven of the last eight goals that chile have scored -lrb- six assists and one goal -rrb- . spain have not lost back to back internationals since october 2006 -lrb- 2-3 v northern ireland and 0-2 v sweden -rrb- . the spaniards conceded more goals in 90 minutes against netherlands -lrb- 5 -rrb- than they had in their previous nine international matches combined -lrb- 4 -rrb- . spain have won just one of their last five internationals played at a neutral venue -lrb- l2 d2 -rrb- and that victory was a 2-0 win over el salvador -lrb- fifa ranked 68th -rrb- in washington earlier this month .

Figure 13: Example of an article with a very long summary, as well as a very low compression ratio from the CNNDM training set.

Article (10 tokens) downton abbey 's famously grim butler , mr bates .

Summary (168 tokens) and now for the good news ... following a particularly grim week , here 's a compendium of some of the world 's most comforting headlines : . supermodel smiles on catwalk . jack russell dog welcomes stranger . child at funfair ‘ delighted ’ by goldfish . katie price breasts ‘ roughly same size as last week ’ say experts . teenager looks up from phone , greets parent . political pundits agree to stop discussing hung parliament for next three months . diner finishes his curly kale . pensioner looks great in party hat . celebrity fails to compare life to roller-coaster . pet hamster repays child 's affection . ‘ cheer up , it may never happen ’ – downton 's mr bates enjoys belly-laugh . style journalist fails to employ the word ‘ iconic ’ sally bercow goes out on town , retains dignity . entire windfarm operates according to plan . miley cyrus feels a bit chilly , opts for extra layer .

Figure 14: Example of a nonsencial article from the CNNDM training set, with a corresponding summary with compression ratio much smaller than 1.

A.2 Incoherent chunking summaries (CNNDM)

Reference love dislocated his left shoulder on sunday during a tussle with boston 's kelly olynyk . olynyk 's right arm became entangled with love 's left arm while his shoulder suddenly popped out . he grabbed his arm and kept running toward the cleveland bench before going to the locker room , where he iced his shoulder . ’ i have no doubt in my mind that he did it on purpose , ’ love said .

CHUNK100 his left shoulder on sunday , has nba star kevin love , he called a ‘ bush-league ’ play . cleveland 's power forward was injured in the first quarter of the cavaliers ’ 101-93 victory jae crowder of the celtics . . olynyk was he and boston 's kelly olynyk .

CHUNK100 + RECONLOSS he called a ‘ bush-league ’ play . cleveland 's power forward was injured in the first quarter of the cavaliers ’ 101-93 victory . shoulder on sunday , has nba star kevin love , his opponent for first half in game . crowder of the celtics , . olynyk 's on sunday .

CHUNK200 he called a ‘ bush-league ’ play . cleveland 's power forward was injured in the first quarter of the cavaliers ’ 101-93 victory the first half in game four of the first round of the nba playoffs . arm became entangled with love 's left arm .

CHUNK200 + RECONLOSS he and boston 's kelly olynyk his opponent for he called a ‘ bush-league ’ play . cleveland 's power forward was injured in the first quarter of the cavaliers ’ 101-93 victory nba star kevin love , who dislocated his left shoulder on sunday , has a four-game sweep .

Figure 15: Example of summaries from all our chunk models on the CNNDM test set, all lacking coherence and grammatical correctness.