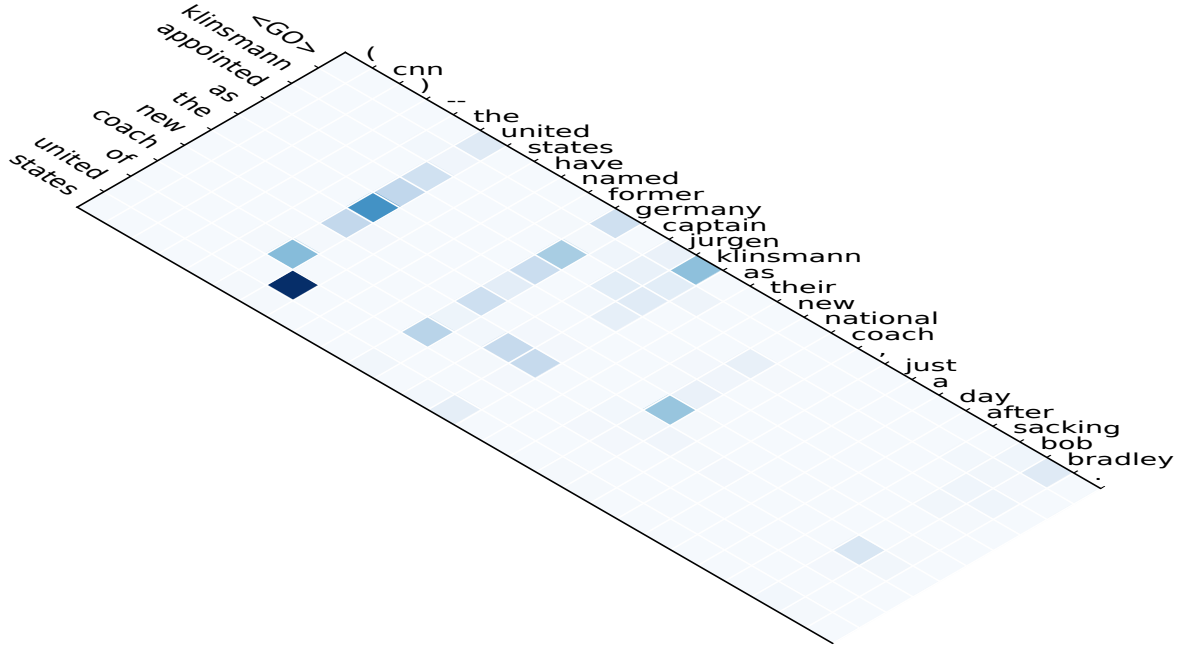
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Query-Based Abstractive Summarization Using Neural Networks

Master's thesis in Computer Science: Algorithms, Languages and Logic

JOHAN HASSELQVIST
NIKLAS HELMERTZ

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

Query-Based Abstractive Summarization Using Neural Networks

JOHAN HASSELQVIST
NIKLAS HELMERTZ

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Query-Based Abstractive Summarization Using Neural Networks
JOHAN HASSELQVIST
NIKLAS HELMERTZ

Supervisor: Mikael Kågebäck, Department of Computer Science and Engineering
Advisor: Henrik Alburg, Findwise AB
Examiner: Peter Damaschke, Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of the attention mechanism presented in this thesis, shown from an isometric perspective. It is described in more detail as Figure 7.1.

Gothenburg, Sweden 2017

Query-Based Abstractive Summarization Using Neural Networks
JOHAN HASSELQVIST
NIKLAS HELMERTZ
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Creating short summaries of documents with respect to a query has applications in for example search engines, where it may help inform users of the most relevant results. Constructing such a summary automatically, with the potential expressiveness of a human-written summary, is a difficult problem yet to be fully solved. In this thesis, a neural network model for this task is presented. We adapt an existing dataset of news article summaries for the task and train a pointer-generator model using this dataset to summarize such articles. The generated summaries are then evaluated by measuring similarity to reference summaries. We observe that the generated summaries exhibit abstractive properties, but also that they have issues, such as rarely being truthful. However, we show that a neural network summarization model, similar to existing neural network models for abstractive summarization, can be constructed to make use of queries for more targeted summaries.

Keywords: natural language processing, summarization, neural networks, sequence to sequence.

Acknowledgements

We would like to thank our supervisor Mikael Kågebäck, who has continuously supported us in this thesis work, even before we had decided on a topic. In addition, we would like to thank Henrik Alburg at Findwise, who has provided valuable feedback and ideas, as well as everyone at Findwise Gothenburg for a great work environment.

Johan Hasselqvist & Niklas Helmertz, Gothenburg, June 2017

Contents

1

Introduction

In this thesis, we present a model for generating summaries of text documents with respect to a certain query. This is known as *query-based summarization*. More specifically, the model is designed for brief, commonly single-sentence, summaries. A situation where this may be useful is when a user has performed a search in a search engine and a set of documents have been returned. Concise summaries could then be displayed along with the search results, giving a quick overview of how the document is related to the search query.

What is commonly done in search engines today is that text surrounding an occurrence of a search query in the document is displayed as a summary. This is an example of *extractive* summarization, which produces a summary that only contains parts of the original document. A significant difference in the model we present is that it generates an *abstractive* summary. This type of summary allows for rephrasing and using words not necessarily present in the original document, comparable to a human-written summary. This has the potential of summarizing documents in a more concise way than what is possible with an extractive summary, i.e. making it easier for a reader to understand the relationship between a document and a query.

In this chapter, we briefly present some research background, define the goals of this thesis work, and describe the thesis structure.

1.1 Context

Automatic text summarization has been a research topic for many years. In general, the goal is to concisely represent the most important information in documents. Much previous work in summarization has been using extractive methods (Nenkova and McKeown, 2012). Commonly, individual sentences are extracted and composed together to form a summary. This gives sentences that are as grammatically correct as the source document. They are however inherently limited, and cannot reproduce human-written summaries in general.

Abstractive summarization in particular is closely related to *natural language generation*, and it would be desirable to reach human-level performance in writing summaries. It may however require human-level understanding of the context of documents to produce results comparable to human-written ones.

An important progress in using neural network models for generating text is *sequence-to-sequence*, used by Sutskever et al. (2014) for machine translation. It is a way of mapping a varying-length input text to a varying-length output text, and it is applicable to machine translation as well as summarization.

In recent years, progress has been made on using neural network models for text summarization and similar problems. Some examples are sequence-to-sequence models for non-query-based abstractive summarization by Rush et al. (2015) and Nallapati et al. (2016). Neural network models have additionally been used for generating image captions (Karpathy and Fei-Fei, 2015), which is a form of summary, and for *question answering* problems, such as by Hermann et al. (2015) and Tan et al. (2015). Inspired by this progress, we design a model for query-based summarization using neural networks.

Chapter 2 goes into more detail on some of the mentioned works most closely related to the approach used in this thesis.

1.2 Goals

The goals of this thesis work are to create a neural network model for query-based abstractive summarization and to evaluate if it can make use of a query for a more targeted summary. For reaching these goals, we make the following contributions:

- A model for query-based abstractive summarization, presented in Chapter 4.
- A dataset for query-based abstractive summarization, created by adapting an existing dataset originally used for question answering, described further in Chapter 5.
- An evaluation of the performance of our model, and of whether the model makes use of the query for a more targeted summary, presented in Chapter 7.

1.3 Delimitations

To limit the scope of the thesis work, and partly due to the datasets available, we focus on queries consisting of one or a sequence of words making up a single *entity*[1], rather than a more general complete question. Grammatically, a single entity would not form a complete question, but it does still expresses a particular information request. Furthermore, the task is limited to *single-document* summarization. This is in contrast to the more general task of *multi-document* summarization, where information from multiple documents is combined to form a summary.

Yet another limitation of the thesis work is that we limit our experiments to the English language. This is mainly due to the availability of datasets. In addition, it

[1] A more detailed definition of entity can be found in Section 3.1.2.

would be ideal if the resulting summaries are coherent and grammatically correct. However, it is not a requirement of the final system.

1.4 Outline

The thesis is structured to first present relevant prerequisite concepts, in Chapter 3. This is followed by a description of the model and the dataset, independently in Chapters 4 and 5 respectively. Chapter 6 describes experiments in which the model is evaluated using the dataset. We then present and discuss the results in Chapter 7, as well as what could be the next step moving forward in Chapter 8.

2

Related Work

There are several tasks related to query-based summarization. In the following sections, we review some existing models for these tasks and their relation to our problem and approach.

2.1 Question Answering

The task of *question answering* is to produce an answer to a question posed in natural language. The task is very general and many other problems can be expressed as a question-answering problem. Summarizing with respect to a query may for instance be expressed as "What is a summary of the document with respect to the query X?", for the query X. If the answer to a question is a single complete sentence, then it is especially close to the types of query-based summaries considered in this thesis.

Otterbacher et al. (2009) present a model, *Biased LexRank*, which they use for a form of question answering as well as extractive query-based summarization. The answers they generate are full sentences, which makes it similar to our task of query-based summarization.

Hermann et al. (2015) present neural network models for question answering. For training these, they create a large dataset from CNN/Daily Mail news articles. We adapt this dataset for query-based summarization, as detailed in Chapter 5.

Kumar et al. (2016) introduce *Dynamic Memory Networks*, which they show reached state-of-the-art performance in a variety of NLP tasks. We draw inspiration from their use of a question module when we incorporate query information in our model.

2.2 General Summarization

General summarization differs from query-based summarization in that a document is summarized without respect to a query.

Nallapati et al. (2016) build upon a machine translation model by Bahdanau et al. (2015) and generate general abstractive summaries on multiple datasets, including the CNN/Daily Mail dataset by Hermann et al. (2015). Additions they make for their

model include a *pointer-generator mechanism* (Gülçehre et al., 2016) that allows the model to copy words from the source document.

See et al. (2017) propose a similar model, using a similar pointer-generator mechanism, that outperforms Nallapati et al. (2016) on a slightly different version of the CNN/Daily Mail dataset (making the result not "strictly comparable"). They also incorporate what they call *coverage* for avoiding repetitions in the output.

2.3 Query-Based Summarization

An early work evaluating several methods for extractive query-based summarization is presented by Goldstein et al. (1999). Besides "full queries", they use "short queries", which on average are 3.9 words. These are similar in length to the types of queries used in the experiments of this thesis work.

Besides the work by Otterbacher et al. (2009), recent work in query-based summarization has been done by Wang et al. (2013), using *parse trees* and *sentence compression*. It is described as not "pure extractive summarization".

During the later stages of this thesis work, Nema et al. (2017) propose a neural network model for query-based abstractive summarization, which has some similarities to the model we present. However, the dataset they use is smaller in both average document length and number of documents. Additionally, the types of queries used are different, in that they use complete questions as opposed to our single-entity queries.

3

Theory

In the following sections, various terms and concepts used throughout the thesis are explained.

3.1 Natural Language Processing

The field of *natural language processing* (NLP) contains various problems related to the processing of *natural languages*, such as English.

A basic concept used throughout the thesis is the concept of a *vocabulary*, denoted V, which we define as a set of *words*. A *word* in the context of this thesis is more general than sequences of letters in a language, which is the more common use of *word*. Our definition of *word* allows for inclusion of other symbols, for example a word consisting only of the period symbol may be a member of the vocabulary.

Another commonly used concept in NLP is an n-gram. An n-gram is a subsequence of length n of connected units in a sequence. In NLP, the unit is commonly words. In the sequence "this is a sequence", the 2-grams are "this is", "is a" and "a sequence".

3.1.1 Tokenization

As a pre-processing step, text in natural languages is often split into a list of smaller meaningful units. Such units are called *tokens*, and the process of breaking down the text to tokens is called *tokenization*. A naive form of tokenization could be to separate the text at every whitespace or non-alphanumeric symbol. A less naive approach would preserve abbreviations and split text more semantically. A tokenized example sentence can be seen in Table 3.1.

Table 3.1: Comparison between a naive and a less naive tokenization method of a sentence. Tokens are separated by spaces for readability.

Original	In the U.S., Swedish furniture isn't uncommon.
Naive tokenization	In the U . S . , Swedish furniture isn ' t uncommon .
Less naive tokenization	In the U.S. , Swedish furniture is n't uncommon .

The resulting tokens of the tokenization are in this thesis later referred to as *words*, belonging to the vocabulary of the tokenized language used in this thesis.

3.1.2 Named Entity Recognition

Information extraction is a class of tasks that involve extracting structured information from documents. An example of such a task is *named entity recognition*, which is the classification of parts of text into different categories, such as *persons* or *locations*, or no category. An example from the sentence "The mathematician Jeff Paris visited the city of Paris." is that "Jeff Paris" should be annotated as a person, and the last "Paris" as a location.

3.1.3 Language Models

Language modeling is the task of assigning a probability to a sequence of symbols, e.g. words in a vocabulary. The probability of such a sequence of words in a vocabulary, V, i.e. $P(w_1, ..., w_N)$ where $w_i \in V$, can be defined as the probability

$$P(w_1, ..., w_N) = \prod_{n=1}^{N} P(w_n \mid w_1, ..., w_{n-1}).$$

In the case of translation and summarization tasks, finding an optimal output sequence can be defined as maximizing the conditional probability of the output sequence dependent on some input X,

$$P(w_1, ..., w_N \mid X) = \prod_{n=1}^{N} P(w_n \mid w_1, ..., w_{n-1}, X).$$

In the particular case of query-based summarization, X would correspond to the input document that is to be summarized as well as the query.

3.1.4 Beam Search

Finding the text sequence with the highest joint probability in the set of all possible sequences that fulfill some constraints is not trivial. For instance, when searching for the most probable sequence of N words, a naive way is to exhaustively search through all the sequences. This is, however, not feasible in general, since with a vocabulary V, the number of solutions scales exponentially as $|V|^N$. On the other hand, if one were to greedily construct such a sequence by iteratively choosing the next word as the word with the highest probability one might discard words that eventually lead to more probable results. One compromise between these two extremes is *beam search*, which is a generalized breadth-first search algorithm that allows for restriction of the search space.

Just like with breadth-first search, beam search starts off with a root node. All possible child nodes for this root node are then evaluated, but, differently from with breadth-first search, only the n best nodes are considered as candidates for a possible solution. These n candidates are then expanded in a similar way to the root node, giving a total of n^2 candidates. Out of these n^2 candidates, only the best n are kept and expanded in the same way iteratively, until either n solutions have been found or there are no more nodes to expand. The selection of n, the *beam width*, gives a trade-off between execution time and memory usage, and the potentiality of finding a better solution. An infinite beam width corresponds to normal breadth-first search.

3.2 Machine Learning

The field of machine learning is about algorithms for detecting patterns and making predictions based on data, i.e. *learning* from the data, rather than algorithms whose behaviors have been specified explicitly.

Some machine learning tasks are *supervised learning* tasks, where a model can be trained on pairs of input and expected output, to learn patterns in the input that makes it possible to predict the output correctly. Commonly, the goal is for this learning to generalize well, so that it could be used to produce useful output for new input it has not been trained on.

In the context of machine learning problems, you commonly have a collection of data known as a *dataset*. An example of a dataset is the MNIST dataset (LeCun et al., 1998), which contains images of hand-written digits, along with what digit the image is supposed to represent.

For evaluation of machine learning models, you can divide the dataset into a *training*, *validation*, and *test* set. The idea is that we can train the machine learning model on a part of the data, the training set, and then evaluate it by its predictive ability on another part of the dataset it has not been trained on. If the model performs well on the training set, but not the validation set, the model does not generalize well. This is known as *overfitting* and is undesirable in a machine learning model.

The reason for there being both a *validation* and *test* set is that the performance on the validation set can be used for making decisions about model design and for tuning *hyperparameters*, such as the size of different parts of the model. After the model has been trained and fully specified, its final generalization performance is determined using the test set. The reason for not using the validation set performance is that hyperparameters for the model may have been optimized in a way that improves the performance due to characteristics specific to the validation set.

Training is commonly performed iteratively over samples in the training set. A cycle of training over all dataset samples is known as an *epoch*. *Early stopping*, described in detail by Goodfellow et al. (2016, pp. 246-248), is a technique for determining when to stop training the model. In an iterative training process, we can estimate the performance on the validation set. If we observe that the performance on the

validation set is decreasing, it signals that the model is overfitting. If this continues for some time, such as over a number of epochs, the training is stopped.

3.3 Neural Networks

Neural networks, or *artificial neural networks*, are models based on solving computational problems in a way inspired by how the brain operates. The main similarity is the idea of neurons being interconnected and outputting signals depending on those they receive from other neurons. This can for instance be used for supervised learning, where we can adjust the network to produce the output by performing computations on the input.

In this section, we present some neural network concepts and examples. More detailed background on neural networks is described by Goodfellow et al. (2016).

3.3.1 Feedforward Neural Networks

Neural networks are commonly modeled as weighted directed graphs. A commonly used type of neural network is a *feedforward neural network*. An example of such a network can be seen in Figure 3.1.

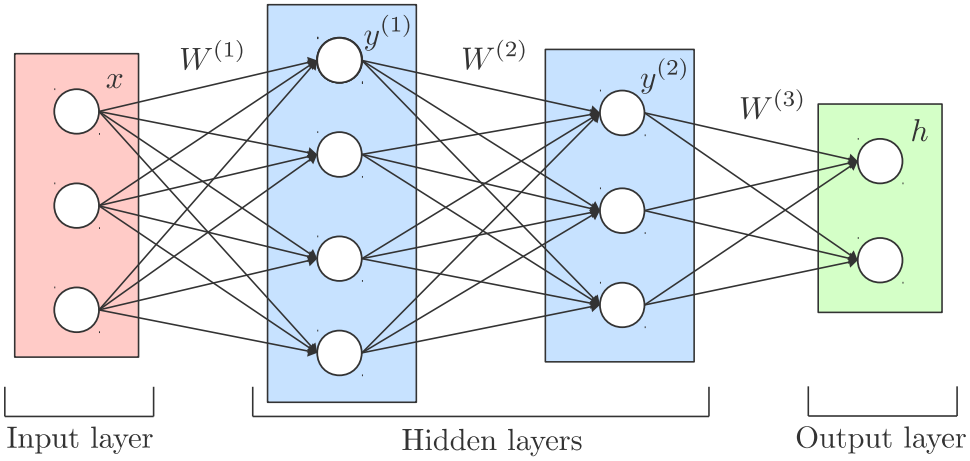


Figure 3.1: A fully connected feedforward neural network with two hidden layers.

A feedforward neural network consists of a sequence of layers, each consisting of a number of nodes. Nodes are connected to subsequent layers with edges. The weight of the edge from node j in layer l to node i in layer $l+1$ is a value in a matrix $W_{ij}^{(l)}$. It is an $n \times m$ matrix, where m and n are the number of nodes in layer l and $l+1$, respectively. The leftmost and rightmost layer are the input and output layer, respectively. The input data is the column vector x. After the computation, the input having been *fed* through the network, the final output is the vector h. Each individual layer's output vector is $y^{(l)}$. We additionally define $y^{(0)} = x$ and $y^{(3)} = h$.

For each layer after the input layer, $l \geq 1$, the output is computed as

$$y^{(l)} = f\left(W^{(l)} y^{(l-1)} + b^{(l)}\right).$$

The bias vector $b^{(l)}$ is not represented in the figure. Its role is similar to $W^{(l)}$, but its contribution does not depend on the previous layer's output. The function f is the *activation function*. It should be a non-linear function, such as element-wise application of the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. There are however several other options for such functions. That the function is differentiable is necessary for the next phase.

Generally, the goal is to assign the weights in the graph, so when data is set in the input layer, the resulting vector on the other end is the desired result. This can be done using *backpropagation* (Rumelhart et al., 1986). Given training data, consisting of input and the desired output, the technique adjusts the weights to accommodate the sample. It can be seen as *learning* to find a pattern. For backpropagation, we use a loss function $L(y, \hat{y})$, where y is the known desired output and $\hat{y}$ is the predicted output. It is designed to give a numerical indication of how wrong the network's prediction is. The objective is to minimize this error. This can be done using *stochastic gradient descent* by updating the weight according to

$$W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \eta \frac{\partial L}{\partial W_{ij}^{(l)}}$$

for all layers l, rows i and columns j. An equivalent update is done of the bias, with $b^{(l)}$ taking the place of $W^{(l)}$. The step size η determines how sensitive the update is to the gradient. Having differentiable functions throughout the computation makes it possible to compute $\frac{\partial L}{\partial W_{ij}^{(l)}}$ using the chain rule. Training of neural networks is described further in Section 3.3.6.

Feedforward networks are appropriate for certain kinds of problems, for example *classification*. Imagining a complex three-dimensional surface, we want to classify if a new point is likely below or above the surface, given a set of points for which we know which side of the surface each is on. Training on such points gives us a network that can be used to estimate if new points are above or below the surface.

3.3.2 Recurrent Neural Networks

There are numerous problems relating to text processing where a feedforward neural network is not well suited. When input data is text written in some language, the words are connected in an order that forms some pattern. For exploiting this structure, there are *recurrent neural networks* (RNN). What makes them *recurrent* is the repeated use of the same neural network layer for each element of an input sequence, such as words. The RNN can be designed to emulate how a human may read a text, processing each word sequentially, and forming an understanding of the text.

A simple approach to designing an RNN is to feed output from the previous item as additional input to a neural network in the current step. This can be viewed as adding a cycle to the previously shown feedforward network. We show this simple RNN compared the previous feedforward network, but with only one single layer, in Figure 3.2.

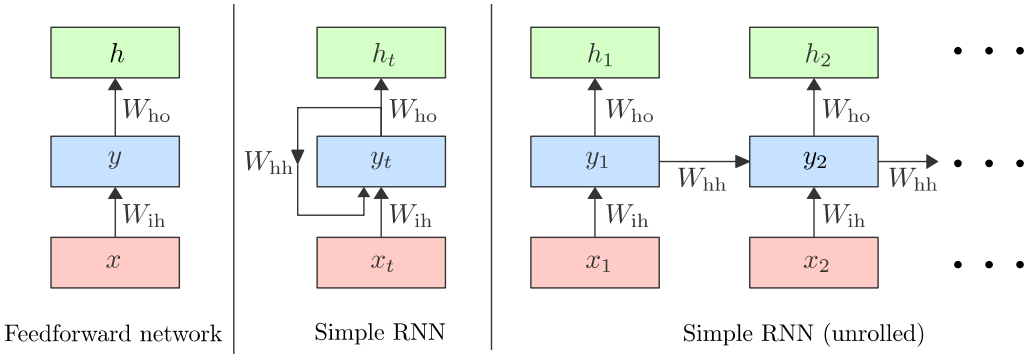


Figure 3.2: The simple RNN compared to a feedforward network.

The boxes represent the intermediate, hidden, vectors. The weight matrices are labeled differently, with W_{ih} connecting the input to the hidden layer, equivalent to $W^{(1)}$ in Figure 3.1. The matrix W_{ho} transforms the hidden vector to the output. An additional matrix W_{hh} transforms the vector in the hidden layer to input for the next iteration. Computations are done similarly to in the feedforward network, but in several iterations $t \in \{1, 2, ...\}$ for each input item x_t. The hidden state y_t, which depends on two inputs, is computed as

$$y_t = f\left(W_{\text{ih}} x_t + W_{\text{hh}} y_{t-1} + b_{\text{h}}\right),$$

where y_0 is initialized to the zero vector. The bias vectors, such as b_{h}, are again not included in the figure. The output vector h_t can be computed as for the feedforward network, by

$$h_t = f\left(W_{\text{ho}} y_t + b_{\text{o}}\right).$$

The weights can, as with the feedforward example, be trained using backpropagation to generate a desired sequence of h_t. The total loss L may be the sum of the loss for each h_t.

3.3.3 Gated Recurrent Units

The previously presented simple RNN is known to suffer from the *vanishing/exploding gradient problem* (Hochreiter, 1991; Y. Bengio et al., 1994), i.e. the backpropagated errors tend to either become too large, or too small, as they are propagated back through the network. This results in RNNs not performing well for long-term dependencies. An example of long-term dependence is that the use of "her" or "his" may depend on a name mentioned much earlier in the text. Hochreiter and Schmidhuber (1997) propose using *long short-term memory* (LSTM) units

that alleviate this issue, making it easier to train RNNs for finding long-term dependencies. Similar to LSTM, the *gated recurrent unit* (GRU) (Cho et al., 2014) does not suffer from the vanishing/exploding gradient problem, but is simpler and less computationally intensive, while still achieving comparable results on many tasks (Chung et al., 2014; Kumar et al., 2016).

Compared to the simple RNN, the GRU network is recurrent through a more complex procedure. However, they are similar in that we pass a previous hidden state h_{t-1}, which can be initialized to the zero vector the first iteration. The architecture is illustrated in Figure 3.3.

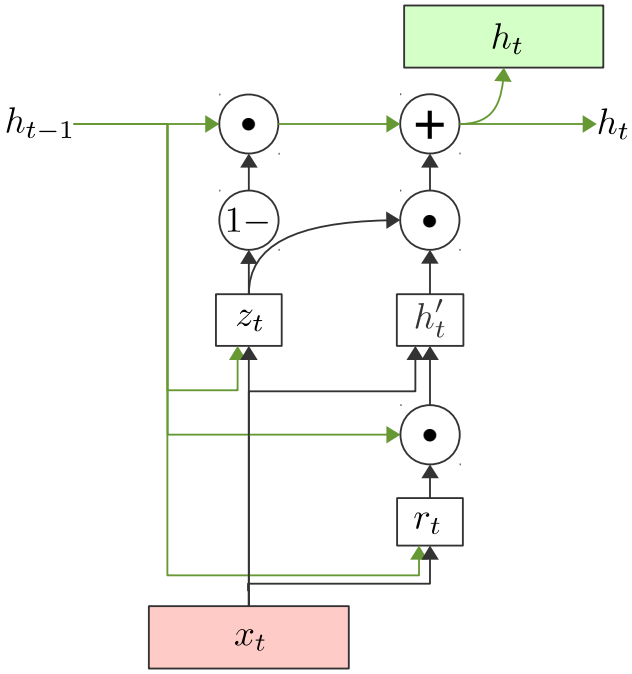


Figure 3.3: Diagram of the GRU architecture. The named boxes represent intermediate vectors. Each except x_t and h_t is the output vector of a neural network, without hidden layers, whose input is shown by the incoming arrows. Boxes with multiple input vectors have the input concatenated. A circle signifies a vector operation of the incoming elements, where $+$ is vector addition, $\cdot$ is element-wise multiplication, and "1−" computes the complement probability for the input elements in $[0, 1]$, by subtracting each from 1.

In Figure 3.3, the vectors r_t and z_t are scaling vectors, intended to regulate what information is let through. These can be described as *gates*. They have elements in $[0, 1]$, generated from a network with the logistic sigmoid, σ, previously defined in Section 3.3.1. The vector h'_t is rather intended to carry data. Its elements are in $[-1, 1]$, generated from a network with a tanh activation function. The entire GRU

architecture can be described by the formulas

$$
\begin{aligned}
r_t &= \sigma(W^r[x_t, h_{t-1}] + b^r) \\
z_t &= \sigma(W^z[x_t, h_{t-1}] + b^z) \\
\\
h'_t &= \tanh(W^h[x_t, r_t \odot h_{t-1}] + b^h) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot h'_t,
\end{aligned}
$$

where $\odot$ is element-wise multiplication, $[x_1, x_2]$ is the concatenation of the vectors x_1 and x_2, forming a vector whose dimension is the sum of the dimension of x_1 and x_2. The W with a superscript are weight matrices, and the b with a superscript are bias vectors. The exact dimension of the vectors are configurable parameters, which must be set so the vector operations are defined.

The vector z_t is intended to regulate how much information from the previous state should be forgotten. For example, in terms of language, if x_t corresponds to a token for a *period* character, it may produce a z_t that causes information from the previous sentence encoded in h_{t-1} to be forgotten. Similarly, i_t regulates what of the new input is added to the state c_t. The final o_t regulates how the state influences the output.

To shorten following formulas in this thesis, we denote an entire GRU update step as

$$h_t = \text{GRU}(h_{t-1}, x_t).$$

3.3.4 Word Embeddings

How to use words as input to a neural network is not obvious. Neural networks internally use numerical vectors. *Word embeddings* give a way to represent words of a vocabulary as fixed size vectors. Given a vocabulary V, we can encode each word uniquely using a *one-hot* encoding. This gives a vector of length $|V|$ where every word in the vocabulary is mapped uniquely to some dimension, which a value of 1, while the other dimensions are 0. This vector can be transformed to an embedding for the word by multiplying it by an embedding matrix W_{emb} of dimensionality $d_{\text{emb}} \times |V|$, where d_{emb} is the word embedding dimensionality, commonly a hyperparameter in neural network models.

The intention is that the embeddings capture some characteristics of words, giving useful vector representations. For instance, two related words such as football and soccer may be expected to be close to each other in the vector space. Two methods for generating word embeddings are word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014).

3.3.5 Sequence-to-Sequence Models

In translation and summarization tasks we want to find a mapping from an input sequence to an output sequence, where the output length is not necessarily dependent

on the input length. One way to do this is by using a *sequence-to-sequence* model (Sutskever et al., 2014).

A sequence-to-sequence model for sequences of words has at least two core components. Firstly, an *encoder*, which reads the input sequence word by word, producing an internal fixed-size representation of it. This internal representation is then passed to the second component, the *decoder*, which generates the output sequence. Both the encoder and the decoder are usually RNNs, and the input words are converted to vector input through a word embedding layer. Figure 3.4 shows an overview of how a sequence-to-sequence model produces an output sequence an example task.

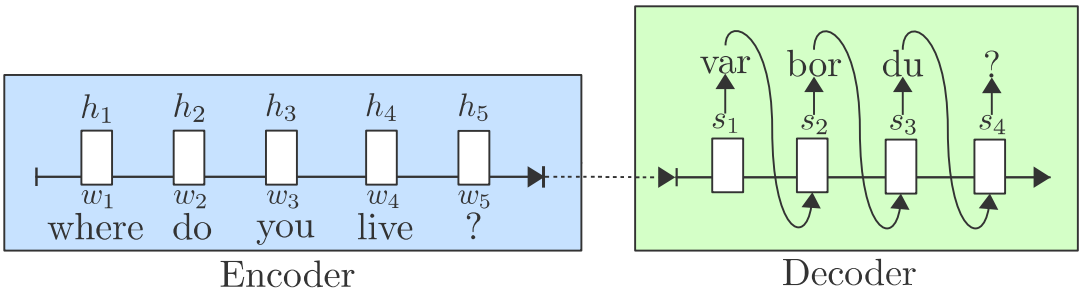


Figure 3.4: Visualization of a sequence-to-sequence model outputting a sentence translated from English to Swedish. The boxes in the encoder represent RNN time steps with the word shown below as input. The words above the the boxes in the decoder signifies the output words. Further, it contains some variables defined subsequently in this chapter.

In a basic sequence-to-sequence model, using GRUs, the encoder can be defined with a single GRU cell. At the ith input word w_i, it then produces a hidden state

$$h_i = \text{GRU}(h_{i-1}, E(w_i)),$$

where $E(w_i)$ is the word embedding of w_i. The initial state h_0 can be a zero vector.

The final encoder hidden state is a representation of all the input, and is passed on to the decoder as its initial state, defined as s_0. The decoder is defined in a similar way to the encoder, with separate internal weights and biases, and its states are computed as

$$s_t = \text{GRU}(s_{t-1}, E(y_{t-1})),$$

where y_{t-1} is the word computed from the previous decoder hidden state. At the first time step, $t = 1$, where there is no previously generated word, a special token signifying the start of the output can be used instead.

For selecting an output word, the decoder hidden state can then be projected on the vocabulary V as

$$z_t = W s_t + b,$$

where $W \in \mathbb{R}^{|V| \times d}$ and $b \in \mathbb{R}^{|V|}$ are trainable parameters, and d is the decoder hidden state size. The result, z_t, contains values that can be converted to the probabilities

p_{tj} of outputting each vocabulary word j at time step t by normalizing it using the *softmax* function, as

$$p_{tj} = \frac{\exp(z_{tj})}{\sum_k \exp(z_{tk})}.$$

If we define this as the probability $P_t(w)$ where w is the jth vocabulary word, the output word at time step t, y_t, can be decided by computing the argmax of the probabilities over the vocabulary as

$$y_t = \arg\max_{w \in V} P_t(w).$$

For many problems, it has been found to be beneficial to use more of the RNN states than the final fixed-size hidden state. *Attention* is a mechanism for allowing the model to access more information in the decoding process, by letting it identify relevant parts of the input and use the encoder hidden state at these locations. This technique has been used successfully for machine translation (Bahdanau et al., 2015) and image captioning (Xu et al., 2015).

There are two main attention types: *soft attention* and *hard attention*. Soft attention generates a distribution over all the encoder hidden states, giving weights to how important different parts of the input is. This is done by first scoring all the encoder outputs, given some input that can differ depending on the application. This input often includes at least the previous decoder hidden state. The scoring can be done in different ways, for instance as a dot product or using additional neural network layers. We denote the ith encoder hidden state as h_i, the previous decoder hidden state as s_{t-1}, and the scoring of h_i as e_i. The scoring can be expressed using a score function as

$$e_i = \text{score}(h_i, s_{t-1}).$$

The attention distribution α is then computed taking the softmax of all the scores, as

$$\alpha_i = \frac{\exp(e_i)}{\sum_j \exp(e_j)}$$

The output of the attention is often called a *context vector*, c, and is an average of the encoder hidden states, weighted by the attention distribution, as

$$c = \sum_i \alpha_i h_i.$$

This combines information from the encoder states that were given higher scores, and therefore hopefully can benefit the decoding process. In machine translation, a word in the target language is often connected to a single word in the source language, which can be attended to using this mechanism.

Hard attention focuses on a single state h_i with probability α_i as a context vector instead of taking a weighted average. However, this step is not differentiable, so one cannot use standard backpropagation.

3.3.6 Training

The training of a neural network is an optimization process where we attempt to optimize the parameters, such as weight matrices, according to some error measure. This is often done using some form of *backpropagation*, as described in the feedforward example in Section 3.3.1.

More specifically, backpropagation can be done using *stochastic gradient descent* (SGD). SGD differs from the standard *gradient descent* in that it updates parameters of the network after every observation in the training set, rather than doing one parameter update after having computed the loss for the whole training set. In particular with a large dataset, SGD generally leads to faster convergence, i.e. in a lower number of iterations. Compared to computing the gradient over the entire training set, gradients based on individual samples may vary greatly from sample to sample. This commonly leads to a more chaotic path in the parameter space throughout the training procedure.

To stabilize the gradient somewhat, and to decrease training time per sample, *mini-batch gradient descent* can be used. Instead of using a single sample or the whole training set, a subset of the training set is used for each parameter update.

Further, the order in which samples, or mini-batches, are trained on affects the training outcome. When training over multiple epochs, it is common to shuffle the samples, or mini-batches, in-between. This has been observed to result in faster convergence (Y. Bengio, 2012).

4

Model

For the task, we have designed a sequence-to-sequence model with attention and a pointer mechanism, making it a *pointer-generator* model. The input for the problem is a document and a query. These are sequences of words passed to a document encoder and a query encoder respectively. The encoders' outputs are then passed to the attentive decoder, which generates a summary. Both encoders, as well as the decoder, use RNNs with GRUs. Each occurrence of GRU, with a subscript, in the formulas in the following sections has separate weights and biases. The entire model is depicted in Figure 4.1. The different components and variables in the figure will be explained in detail throughout the chapter.

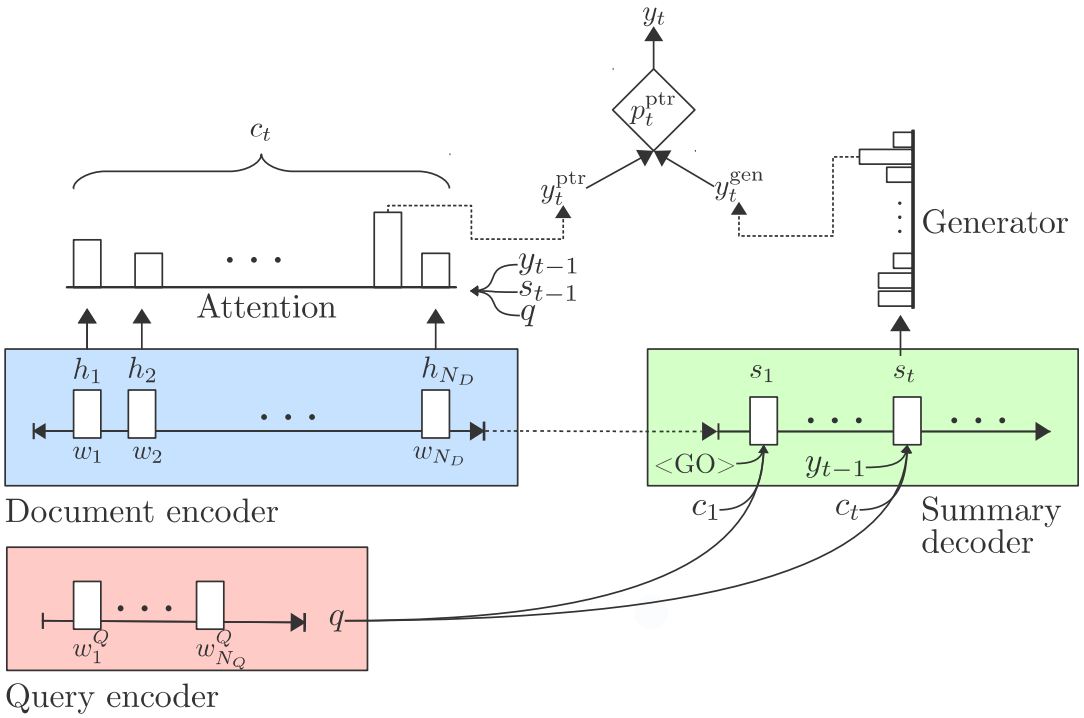


Figure 4.1: Overview of our model. It illustrates connections between parts of the model at a fixed decoder time step t, using the variables defined in this chapter. The bottom part, containing labeled boxes, correspond to the different RNNs. The top part is intended to visualize the two ways the output word y_t can be selected, through the pointer and generator mechanism, to the left and right respectively.

4.1 Document Encoder

The document encoder processes an input document, generating a state for each input word. To get a representation of the context around a word, we use a bidirectional RNN (Schuster and Paliwal, 1997) encoder, so both the context before and after contribute to the representation. This is used by Bahdanau et al. (2015) amongst others, achieving good results on a similar task related to text comprehension.

The combined RNN hidden state at time step i, h_i, and the intermediate states, $\overrightarrow{h}_i$ and $\overleftarrow{h}_i$, from the forward reader and backward reader respectively, are computed as

$$\begin{aligned}
\overrightarrow{h}_i &= \mathrm{GRU}_{\overrightarrow{\mathrm{doc}}}(\overrightarrow{h}_{i-1}, E(w_i)) \\
\overleftarrow{h}_i &= \mathrm{GRU}_{\overleftarrow{\mathrm{doc}}}(\overleftarrow{h}_{i-1}, E(\overleftarrow{w}_i)) \\
h_i &= [\overrightarrow{h}_i, \overleftarrow{h}_i],
\end{aligned}$$

where $w_i \in V$, for the vocabulary V, is word i in the input document; $\overleftarrow{w}_i$ is word i in the reversed input; and $E(w_i)$ is the word embedding of w_i. The initial states $\overrightarrow{h}_0$ and $\overleftarrow{h}_0$ are zero vectors. Due to the concatenation, the combined state h_i has twice the dimensionality of the state of each unidirectional encoder. The document encoder state dimensionality is denoted d_{doc} and the word embedding dimensionality d_{emb}.

4.2 Query Encoder

The query encoder is responsible for creating a fixed-size internal representation of the input query. Unlike the document encoder, the query encoder is a unidirectional RNN encoder since queries are relatively short compared to documents and we only use the final state to represent the whole query. The RNN state h_i^Q at query word i, is updated according to

$$\begin{aligned}
h_i^Q &= \mathrm{GRU}_{\mathrm{que}}\left(h_{i-1}^Q, E\left(w_i^Q\right)\right) \\
q &= h_{N_Q}^Q,
\end{aligned}$$

where w^Q is the input query and N_Q is the length of the query. The initial state h_0^Q is the zero vector. The query encoder state dimensionality is denoted d_{que}.

4.3 Decoder

The decoder is a unidirectional RNN for constructing a summary of the input document by depending on the final state of the input encoder, the query. It utilizes soft attention, in combination with a pointer mechanism, as well as a *generator* part similar to Bahdanau et al. (2015). The query embedding q is fed as input

at each decoder time step. This is similar to the *answering module* in a question answering model presented by Kumar et al. (2016), who use an RNN-encoded *question* representation as input at each decoder time step. In our model, the RNN state is updated according to

$$s_t = \mathrm{GRU}_{\mathrm{dec}}\left(s_{t-1}, [c_t, q, E\left(y_{t-1}\right)]\right),$$

where $s_0 = h_{N_D}$, the final document encoder state, N_D being the number of input words; y_0 corresponds to a special `<GO>` token, used at the initial time step when no previous word has been predicted; c_t is the *context vector* at time step t from the attention mechanism, defined subsequently; and $y_{t-1} \in V$ is the predicted output word at time step $t-1$. This is either from the generator mechanism, or the pointer mechanism, also defined subsequently. The word embeddings are the same as are used in the encoder.

The intention of the inclusion of q to the input of $\mathrm{GRU}_{\mathrm{dec}}$ is to give the decoder the ability to tune the structure of the output sequence to eventually output something concerning the query. For example, if the query is a location, the decoder can output words leading up to an appropriate inclusion of the location.

The generator outputs a word from a subset of the vocabulary $V_{\mathrm{gen}} \subseteq V$ at each time step. The selection of the output words is done through a distribution of words in V_{gen}, computed through a softmax as

$$p_{tj}^{\mathrm{gen}} = \frac{\exp(z_{tj})}{\sum_k \exp(z_{tk})},$$

for $j \leq |V_{\mathrm{gen}}|$, an index uniquely mapped to a word $w \in V_{\mathrm{gen}}$, and z_{tj} as defined subsequently. Defining this as the probability $P_t^{\mathrm{gen}}(w)$, we then select output word y_t^{gen} with the highest probability by

$$y_t^{\mathrm{gen}} = \arg\max_{w \in V_{\mathrm{gen}}} P_t^{\mathrm{gen}}(w).$$

The softmax probability depends on z_{tj}, the output from two linear transformations on the decoder state and context vector, defined as

$$z_t = W_{\mathrm{gen}}^{(2)}\left(W_{\mathrm{gen}}^{(1)}[s_t, c_t] + b_{\mathrm{gen}}^{(1)}\right) + b_{\mathrm{gen}}^{(2)},$$

where $W_{\mathrm{gen}}^{(1)} \in \mathbb{R}^{d_{\mathrm{gen}} \times (d_{\mathrm{dec}} + d_{\mathrm{doc}})}$, $b_{\mathrm{gen}}^{(1)} \in \mathbb{R}^{d_{\mathrm{gen}}}$, $W_{\mathrm{gen}}^{(2)} \in \mathbb{R}^{|V_{\mathrm{gen}}| \times d_{\mathrm{gen}}}$ and $b_{\mathrm{gen}}^{(2)} \in \mathbb{R}^{|V_{\mathrm{gen}}|}$ are trainable hyperparameters, in which d_{gen} is the dimensionality of the hidden layer. The main function of this layer is to reduce the dimensionality of the input, for reducing computation time for the final layer with size $|V_{\mathrm{gen}}|$.

The model has a soft attention mechanism, based on one used by Bahdanau et al. (2015) for machine translation. The result of the attention mechanism is a context vector c_t produced at each time step t, computed as

$$\begin{aligned} c_t &= \sum_i \alpha_{ti} h_i \\ \alpha_{ti} &= \frac{\exp(e_{ti})}{\sum_k \exp(e_{tk})} \\ e_{ti} &= \mathrm{score}(h_i, s_{t-1}, E(y_{t-1}), q), \end{aligned}$$

where h_i is the document encoder hidden state at index i. The score function is defined as

$$\text{score}(h, s, x, q) = v_{\text{att}}^{\intercal} \tanh(W_{\text{att}}[h, s, x, q] + b_{\text{att}}),$$

where $W_{\text{att}} \in \mathbb{R}^{d_{\text{att}} \times (d_{\text{doc}} + d_{\text{dec}} + d_{\text{emb}} + d_{\text{que}})}$ is a weight matrix, $v_{\text{att}} \in \mathbb{R}^{d_{\text{att}}}$ is a vector, and b_{att} is a bias vector, all of which are trained together with the rest of the network. The query q is included for the model to focus attention around query words when appropriate.

4.3.1 Pointer Mechanism

A general issue is that with a generator mechanism limited to frequent words, infrequent words cannot be generated. In our model, perhaps most problematic, the query words may not be possible to generate. Further, if the model needs to learn to output names, and there are many different ones and few occurrences of each in the training data, training a model to generate them correctly is problematic. A way to solve these issues is to allow the model to directly copy a word in the input document to the output summary, or *point* to it. This may additionally be viewed as using the input text as a secondary output vocabulary, in addition to V_{gen}.

Combining a pointer mechanism with a generator mechanism gives a *pointer-generator model*. This has been done for summarization by Nallapati et al. (2016), and is presented separately by some of the authors (Gülçehre et al., 2016). Additionally, Merity et al. (2017) and See et al. (2017) present similar models with the same capability.

The pointer mechanism adds a *switch*, $p^{\text{ptr}} \in (0, 1)$, at each decoder time step t, to the model. It is computed as the output of a linear transformation fed through a sigmoid activation function, as

$$p_t^{\text{ptr}} = \sigma\left(v_{\text{ptr}}^{\intercal}[s_t, E(y_{t-1}), c_t] + b_{\text{ptr}}\right),$$

where $v_{\text{ptr}} \in \mathbb{R}^{d_{\text{dec}} + d_{\text{emb}} + d_{\text{doc}}}$ and b_{ptr} are vectors, all of which are trained together with the rest of the network.

If $p_t^{\text{ptr}} > 0.5$, a word is copied from the input, otherwise the generator output is used. What is copied from the input for the tth decoder word is determined by the attention distribution. Specifically, at time step t, we select the word at index

$$i'_t = \arg\max_i \alpha_{ti}$$

in the document, where the attention is highest, as

$$y_t^{\text{ptr}} = w_{(i'_t)}.$$

The final output word can then be defined as

$$y_t = \begin{cases} y_t^{\text{ptr}} & \text{if } p_t^{\text{ptr}} > 0.5 \\ y_t^{\text{gen}} & \text{otherwise} \end{cases}$$

4.4 Training Loss

The model is trained in when to use the pointer mechanism in a supervised manner. We define an additional training input x_t^{ptr} that is either 1 if the pointer mechanism is set to be used for the tth word in the summary, or 0 otherwise. For training this, we define a loss function

$$L_{\text{ptr}} = \sum_{t=1}^{N_S} \left(x_t^{\text{ptr}} \left(-\log p_t^{\text{ptr}} \right) + \left(1 - x_t^{\text{ptr}} \right) \left(-\log \left(1 - p_t^{\text{ptr}} \right) \right) \right).$$

This is to train p_t^{ptr} to predict x_t^{ptr}.

For training the generator mechanism, we define a loss over the generator softmax layer as

$$L_{\text{gen}} = \sum_{t=1}^{N_S} \left(1 - x_t^{\text{ptr}} \right) \left(-\log P_t^{\text{gen}}(w^*) \right),$$

where N_S is the length of the target summary, $w^* \in V_{\text{gen}}$ is the the tth word in the target summary. Multiplying by $\left(1 - x_t^{\text{ptr}} \right)$ excludes any addition to the loss when the pointer mechanism is set to be used.

We introduce a form of supervised attention for when the pointer mechanism is set to be used for an output word by introducing a loss function

$$L_{\text{att}} = \sum_{t=1}^{N_S} x_t^{\text{ptr}} \left(-\log \alpha_{ti^*} \right),$$

where i^* is the index in the input document to point to.

The final loss function is the sum of the different losses, normalized by the length, computed as

$$L = \frac{1}{N_S} \left(L_{\text{gen}} + L_{\text{att}} + L_{\text{ptr}} \right).$$

5

Dataset

In this chapter, we describe the dataset that has been constructed for the experiments performed in this thesis work. Additionally, properties and problems of the dataset are discussed.

To train a deep learning neural network model for question answering, Hermann et al. (2015) constructed a dataset of document–query–answer triples from CNN and Daily Mail news articles. Included with each published news article, there are a number of human-written *highlights*, which summarize different aspects of the article. Table 5.1 shows some example highlights for a single article. They construct a document–query–answer by considering a named entity in a highlight to be unknown, making the highlight into a Cloze-style question (Taylor, 1953), whose answer is the entity made unknown. An example document and a Cloze-style question and its answer can be seen in Table 5.2.

Table 5.1: Highlights of a CNN article titled "Airline quality report sorts out the duds from the dynamos in 2012".

1. Hawaiian Airlines again lands at No. 1 in on-time performance
2. The Airline Quality Rankings Report looks at the 14 largest U.S. airlines
3. ExpressJet and American Airlines had the worst on-time performance
4. Virgin America had the best baggage handling; Southwest had lowest complaint rate

The article data comes from web scraping of the corresponding news organization's web site. For the question-answering model, they have then been tokenized and *anonymized* by Hermann et al. (2015). The anonymization step replaces named entities with entity markers, such as "@entity12", which refers to the same entity in multiple locations in a document, found through "simple entity detection". The same marker does not refer to the same entity across documents however.

We propose using the CNN/Daily Mail dataset for query-based abstractive summarization by regarding each highlight as a summary of its document, and entities in the highlight as queries. For every occurrence of an entity in a highlight, we construct a document-query-summary triple for query-based summarization. Table 5.2 shows for a sample document a Cloze-style question compared and the corresponding query-summary pair constructed by us. If an entity is mentioned in multiple highlights, we consider there being multiple target references for the document-query pair.

Table 5.2: Example of dataset samples generated from a document-query pair using our method compared to Hermann et al. (2015). In the Cloze-style questions, the entity corresponding to the answer has been replaced by X.

Document (cnn) former vice president walter mondale was released from the mayo clinic on saturday after being admitted with influenza , hospital spokeswoman kelley luckstein said . " he 's doing well . we treated him for flu and cold symptoms and he was released today , " she said . mondale , 87 , was diagnosed after he went to the hospital for a routine checkup following a fever , former president jimmy carter said friday . " he is in the bed right this moment , but looking forward to come back home , " carter said during a speech at a nobel peace prize forum in minneapolis . " he said tell everybody he is doing well . " mondale underwent treatment at the mayo clinic in rochester , minnesota . the 42nd vice president served under carter between 1977 and 1981 , and later ran for president , but lost to ronald reagan . but not before he made history by naming a woman , u.s. rep. geraldine a. ferraro of new york , as his running mate . before that , the former lawyer was a u.s. senator from minnesota . his wife , joan mondale , died last year .
Highlight walter mondale was released from the mayo clinic on saturday , hospital spokeswoman said
Cloze-style question walter mondale was released from the X on saturday , hospital spokeswoman said **Cloze-style answer** mayo clinic
Our query mayo clinic **Our target summary** walter mondale was released from the mayo clinic on saturday , hospital spokeswoman said

The dataset is not clean in the sense that the highlights cannot always be considered summaries of information in the article. For example, there are highlights such as "At the bottom of the page, comment for a chance to be mentioned on CNN Student News. You must be a teacher or a student age 13 or older to request a mention on the CNN Student News Roll Call", i.e. there are other notices unrelated to the particular article shown as highlights.

An additional problem with the dataset is that highlights sometimes refer to previous highlights. For example, if the first highlight refers to a person with their name, subsequent highlights sometimes refer to the same person with only a pronoun. This means that when we regard a highlight as a reference summary to a news article, it may be the case that who or what the pronouns refer to is ambiguous.

5.1 Processing

As opposed to Hermann et al. (2015), we do not use an anonymized version of the dataset. Instead we use a less processed version of the dataset and process it through a different tokenization algorithm, followed by lowercasing the text. It is processed through the Python library NLTK[1] (Bird et al., 2009), which uses the tokenizers *Punkt Sentence Tokenizer*[2] and *Penn Treebank Tokenizer*[3], for English. We noticed that this processing commonly resulted in words beginning with apostrophe characters improperly. We applied another small tokenization step that splits the word at those apostrophes, except for some common cases, such as the genitive "'s" and "'til".

Furthermore, we have performed a different split for creating our training, validation, and test set than Hermann et al. (2015). They train and evaluate on articles from CNN and Daily Mail separately, for example evaluating only on CNN articles after training on only CNN articles. Validation and test sets have been created by reserving articles from a single month of a single year to each.

We decided to train our model on a mix of CNN and Daily Mail articles, with a proportion of them being reserved for validation and test sets. This proportion is selected so that it is similar to the proportion of the CNN data split done by Hermann et al. (2015). Which articles are included for the validation and test set is determined randomly with equal probability for every article. If we instead were to use the same split as Hermann et al. (2015) when we mix the CNN and Daily Mail articles, Daily Mail would be greatly overrepresented. This is because the CNN articles span over several more years than the Daily Mail articles, so a single month of Daily Mail articles contains a larger proportion of all Daily Mail articles than what a single month of CNN articles does for all CNN articles.

Some statistics of the resulting dataset can be seen in Table 5.3.

Table 5.3: Statistics of the dataset. The # symbol is used as an abbreviation of "number of".

	Training	**Validation**	**Test**
#documents	300,805	4,652	4,652
#document-query pairs	1,066,377	16,308	16,593
#document-query-summary triples	1,294,730	19,827	20,046
Average #words per document	773.02	778.78	775.70
Average #words per query	1.52	1.53	1.52
Average #words per summary	14.44	14.52	14.40

[1]Version 3.0, URL: `http://www.nltk.org/`
[2]Described at: `http://www.nltk.org/_modules/nltk/tokenize/punkt.html`
[3]Described at: `http://www.nltk.org/_modules/nltk/tokenize/treebank.html`

The dataset can be reproduced using a script made available on GitHub[4].

5.2 Dataset Structure

We organize the dataset triples hierarchically, first by document, then query, then reference. The documents and queries are numbered numerically starting with *1*, while the references are numbered alphabetically starting with *A*. Document 1 may have queries 1.1 and 1.2, and reference summaries A.1.1, B.1.1 and A.1.2[5]. The order is shuffled amongst document, query and reference IDs.

5.3 Other Datasets

There exist other datasets from *Document Understanding Conferences* (DUC)[6], especially from DUC 2005, DUC 2006 and DUC 2007 used for query-based abstractive multi-document summarization, such as by ShafieiBavani et al. (2016). These datasets, however, are much smaller than the CNN/Daily Mail corpus, and the summaries are rather long (around 250 words), unlike the shorter summaries that are the focus for this thesis work.

Nema et al. (2017) create a dataset with which they train their own query-based abstractive summarization model. Their dataset is however smaller, consisting of only 12,695 document-query-summary triples, and their documents are only 66 words long on average, compared the CNN/Daily Mail average of over 750 words. While we believe that the tasks we attempt to solve are very similar, we think that summarizing longer documents as with the CNN/Daily Mail dataset is of even greater interest.

[4]URL: `https://github.com/helmertz/querysum-data`
[5]Selected for matching the format expected by pyrouge
[6]URL: `http://duc.nist.gov/data.html`

6

Experiments

In this chapter, we detail how we use the constructed CNN/Daily Mail dataset with our model. We first describe some more practical details about how we trained the model and how we generate summaries, followed by how we evaluate the generated summaries. Additionally, we present an evaluation method for measuring how well it incorporates the query when generating summaries.

6.1 Vocabulary

The subset V_{gen} of the entire vocabulary V used for the model in these experiments contains a number of special tokens, which we differentiate from the other words, and frequent words from the training set. There are four special tokens: `<UNK>`, `<GO>`, `<EOS>`, and `<PAD>`. The `<UNK>` token, from *unknown*, is for representing words not in the vocabulary. `<GO>` is the word fed as previous output at the first decoder time step and signifies the start of the summary generation. `<EOS>`, *end of sequence*, is appended to the query fed through the query encoder and the target summaries. When the decoder generates the `<EOS>` token, the generated summary is regarded as complete. `<PAD>` is included for technical implementation reasons, and the network is not trained to output these. In addition to the special tokens, V_{gen} contains many as possible of the most frequent words from the training set summaries until V_{gen} contains 20,000 words. This correspond to the words the generator mechanism can output.

The complete vocabulary V used for the input is substantially larger than what the decoder generator can output. It additionally contains the 150,000 most frequent words in the training set documents not in V_{gen}.

Word embeddings for the vocabulary words are initialized with 100-dimensional GloVe embeddings[1], trained on "Wikipedia 2014 + Gigaword 5". If the word does not have a GloVe embedding, we initialize the word embedding by sampling the per-dimension univariate normal distributions with means and standard deviations of the entire collection of GloVe embeddings. The vocabulary is expanded by adding those of the 100,000 most frequent words in the pre-trained GloVe embeddings that are not yet in the vocabulary. This gives a total vocabulary size of $|V| = 173,256$.

[1]Downloadable as "glove.6B.zip" at: `https://nlp.stanford.edu/projects/glove/`

The word embeddings for these are trained along with the network.

6.2 Pointer Training

We train the model to use the pointer mechanism for occurrences of entities in the summaries, specifying the training target used for the loss contributions L_{ptr} and L_{att}. The entities are the same that are used for anonymization by Hermann et al. (2015). For each entity word in the summaries, we point to the corresponding index of the word in the first occurrence of the entity in the document. Specifically, we process the non-anonymized summaries from beginning to end, trying to greedily find literal occurrences of entities, prioritizing entities consisting of more words. We then point to the first complete occurrence of the entity in the corresponding document. That we only train the pointer for the first occurrence is potentially problematic. It would seem more reasonable to train it to attend to any instance of the entity in the text, but this would make the loss more complex.

The pointer mechanism is additionally used for words not included in the smaller vocabulary V_{gen} used for the generator mechanism, but which are not part of an entity. We train the model to point to the first occurrence in the document, if there are any. If it is neither in V_{gen} nor the input document, to avoid training the model to output `<UNK>`, we make the loss zero at these time steps. Due to the nature of our evaluation, presented in detail in Section 6.6, we want the model to produce a *best guess*, rather than `<UNK>`.

6.3 Training Details

All weight matrices are initialized using Tensorflow's implementation of Xavier uniform initialization (Glorot and Y. Bengio, 2010) and all bias vectors are initialized as zero vectors.

During training, when the previous output word is input to the next decoder step ($E(y_{i-1})$), rather than feeding the actual predicted output word, we feed it the previous word directly from the reference summary, as if it had generated the correct output. This technique is used to speed up training. The trade-off is discussed in more detail by S. Bengio et al. (2015), who additionally propose an approach based on gradually decreasing the probability of using the technique.

We use a sampling-based loss computation for L_{gen} introduced by Jean et al. (2015), with a sample size of 512. Essentially, this means that the probability of every word w, $P_t^{\text{gen}}(w)$, is not computed when maximizing the probability of the target word. Beside the target word, only a randomly sampled subset of other words is used when computing the softmax probabilities. This is done for reducing computation time during training.

Both during training and test time, we limit the document length to the first 800

words, to reduce computation time.

The loss L is minimized using the SGD-based Adam optimizer (Kingma and Ba, 2015). We used mini-batches of 30 samples, with an averaged loss over all the samples in the batch. The mini-batches remained the same over epochs, but the order in which they were trained on was randomized between every epoch.

Experiments have been run on a single Nvidia Tesla K80, with 12 GB of memory. The model has been implemented using the machine learning library *TensorFlow* (Abadi et al., 2015), version 1.1. The code for the model is available on GitHub[2].

Training the model took approximately 18 hours per epoch. We used early stopping based on the validation set and stopped training when we did not gain an improvement for a whole epoch. This gave our best-performing model parameters after approximately three epochs, a total of 54 hours, which are used for the subsequent evaluation. In contrast to Nallapati et al. (2016) and See et al. (2017), this number of epochs is low. However, in a single epoch, we train on the same documents and target summaries multiple times, due to how the dataset is constructed.

6.4 Generating Summaries

When generating output, beam search is used. This affects the word selected from the generator mechanism. When extending a partial summary, instead of extending it with only the most probable word

$$y_t^{\text{gen}} = \arg\max_{w \in V} P_t^{\text{gen}}(w),$$

the top-k words are selected, giving k different partial summaries for each of the previous k partial summaries, totaling k^2 partial summaries at that time step. These partial summaries are considered one by one, starting with the most probable. If a considered summary is complete, i.e. ends with the `<EOS>` token or reaches the maximum output length, it is added to a pool of k completed summaries. If it is not a complete summary, it is instead added to a pool of k best partial summaries to be extended at the next step. Once k complete summaries have been found, the most probable of these is chosen as the output summary.

For time steps where the pointer mechanism is used, the partial summaries are prioritized by the same probabilities as if the generator had been used instead, so k partial summaries with different probabilities are created for the word chosen by the pointer mechanism. This is difficult to justify, but we hope that this should give a reasonable probability at time steps when the pointer mechanism is used, preventing summaries using the pointer mechanism more to be prioritized.

In our experiments, we use a beam width of $k = 5$ and a maximum output length of 32.

[2]URL: `https://github.com/helmertz/querysum`

A slight deviation from what is presented in Section 4.3.1 is that when the pointer mechanism is used and the attended word was not in V, we do not output `<UNK>`, which it is otherwise interpreted as in the model, but rather the actual input word before it being converted to an index in the vocabulary. This may be viewed as a post-processing step.

6.5 Model Parameters

The hyperparameter used for the experiments can be seen in Table 6.1. We have not performed extensive hyperparameter tuning, but instead examined hyperparameters used for similar models, such as Nallapati et al. (2016) and See et al. (2017).

Table 6.1: Hyperparameters of the model used for the experiments.

Hyperparameter		**Value**
Word embedding size	d_{emb}	100
Document encoder size	d_{doc}	512
Query encoder size	d_{que}	256
Decoder size	d_{dec}	512
Attention hidden size	d_{att}	256
Generator hidden size	d_{gen}	256

6.6 Evaluation

The standard evaluation method for automatic summarization is *ROUGE* (Recall-Oriented Understudy for Gisting Evaluation) (Lin, 2004). It is based on comparing n-grams between a generated summary and one or multiple reference summaries. A possible flaw when evaluating summaries is that synonyms, and variations of words, are counted as different words, resulting in lower scores even if the summaries are semantically similar.

ROUGE scores texts as a ratio from 0 to 1, where 1 is a perfect score. This is often presented as 0 to 100 percent, as is done in this thesis. We will be presenting results with *F-scores*, which is the harmonic mean of the two measurements *precision* and *recall*. Precision measures the relevance of the evaluated summaries' contents and recall measures how much of the reference summary is represented in an evaluated summary. For n-gram based ROUGE metrics, if we let X and Y be the set of all n-grams in an evaluated summary and reference summary respectively, we can compute the precision as $\frac{|X \cap Y|}{|X|}$. Recall is computed in a similar way as $\frac{|X \cap Y|}{|Y|}$.

Using the Python package *pyrouge*[3], our results are evaluated with F-scores from

[3]Version 0.1.3, `https://pypi.python.org/pypi/pyrouge`. The arguments used for the `ROUGE-1.5.5.pl` script are: `-c 95 -2 4 -U -r 1000 -n 4 -w 1.2 -a -m`

four different metrics provided by ROUGE: *ROUGE-1*, *ROUGE-2*, *ROUGE-L*, and *ROUGE-SU4*. ROUGE-1 and ROUGE-2 are the scores for 1-grams and 2-grams respectively. ROUGE-L and ROUGE-SU4 are more complex metrics, detailed by Lin (2004). When evaluating a generated summary against multiple reference summaries, it is evaluated against all the targets individually, and only the highest score is used.

6.6.1 Query Dependence

To determine whether incorporating a query benefits our model, we use an evaluation metric based on ROUGE. Instead of evaluating the generated summary for a document and a query with ID n against the reference summaries for that query, we evaluate it against the reference summaries for query $n + 1$, i.e. the query ID has been offset. For the query with the highest ID, the reference summaries for the first query are used. The idea is that if the score is lower than for the normal evaluation, then the model has made use of the additional information in the query. Table 6.2 shows for an example document, 1, what the generated summaries are evaluated against during the query-dependence evaluation.

Table 6.2: Example of reference summaries used during normal evaluation compared to with offset queries.

	Reference summaries	
Query ID	**Normal**	**Offset queries**
1.1	A.1.1, B.1.1	A.1.2
1.2	A.1.2	A.1.3, B.1.3
1.3	A.1.3, B.1.3	A.1.1, B.1.1

It is worth to mention that two reference summaries for different queries may be the same, as the same original highlight may be used as a reference summary for multiple queries. In these cases, the query will be appropriate for the summary and the model may have benefited from the query even in the query-offset evaluation. It may be possible to avoid this in some cases. However, for simplicity, we use this method.

6.7 Baseline

As a baseline, we compare the results to a simple extractive summary, designed specifically for the dataset used in this thesis work. The baseline summary is constructed by selecting the first sentence in the document containing the query, without restricting the length of the document. If no such sentence is found, i.e. the document does not contain the query, the first sentence of the document is used instead. This does occur in the dataset, but not frequently.

We additionally observe that the average length of baseline sentences using the CNN/Daily Mail dataset is commonly greater than for the reference summaries. The average number of words is 30.56 for the baseline summaries, while it is 14.44 for the reference summaries. It may be possible to gain a higher ROUGE score if a fewer number of words around the query occurrence is selected, but it might not form a complete sentence.

7

Results and Discussion

After running the experiments described in the previous section, we evaluate the performance of our model in comparison to the baseline. The ROUGE scores on the test set can be seen in Table 7.1. We observe that they are lower than the baseline model, which we later will comment on in more detail, followed by a discussion on possible improvements. We denote the baseline model, described in Section 6.7, *first query sentence* and the model developed for this thesis *our model*.

Table 7.1: ROUGE F-scores of the different models on the test set. Numbers in bold signifies the best score among the models.

	ROUGE			
Model	1	2	L	SU4
First query sentence	**33.81**	**18.19**	**29.22**	**17.49**
Our model	18.25	5.04	16.17	6.13

We observe that the attention at a time step appears to often be highly focused on only a few words in the document. An example of an output summary can be seen in Table 7.2, and Figure 7.1 shows the attention distribution over time for the same generated summary.

Table 7.2: Model output and reference summary for a document-query pair in the test set. Only the beginning of the document is shown; "[...]" signifies that the document continues.

Document (cnn) – the united states have named former germany captain jurgen klinsmann as their new national coach , just a day after sacking bob bradley . bradley , who took over as coach in january 2007 , was relieved of his duties on thursday , and u.s. soccer federation president sunil gulati confirmed in a statement on friday that his replacement has already been appointed . " jurgen is a highly accomplished player and coach with the experience and knowledge to advance the program , " said gulati . [...]
Query united states
Reference jurgen klinsmann is named as coach of the united states national side
Output klinsmann appointed as the new coach of united states

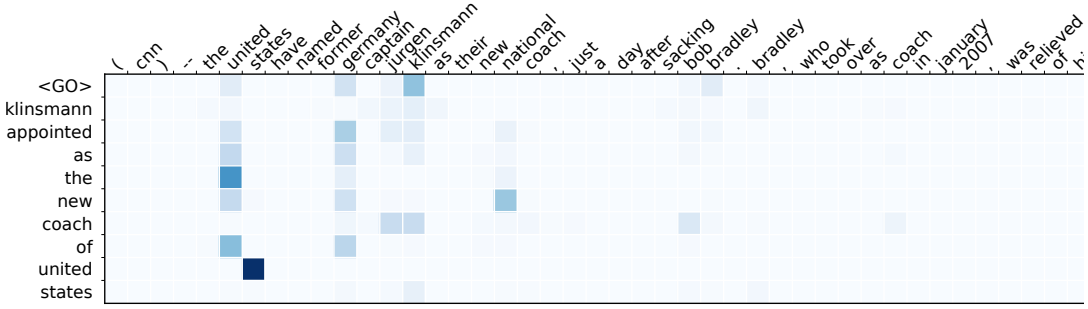


Figure 7.1: Visualization of the attention distribution, α_{ij}, as the summary in Table 7.2 is generated. The words of the document are shown on the horizontal axis, from left to right. Only a limited number of document words are shown. The vertical axis shows the output words, from top to bottom, after the `<GO>` token. The darker a cell is, the higher the attention on that position, influencing the word selected on the row below. The final row shows the attention distribution before the `<EOS>` token was generated.

Another observation we make is that the attention often is focused at the beginning of the documents. However, there are certainly instances when entities are selected from far back in documents. This bias may partly be due to our decision to point to the first occurrences of entities during training. Although, it has been noted by Goldstein et al. (1999) that the beginning of news articles often summarizes the article quite well.

From examining some of the output summaries from our model, we see that they often strongly match the topic of the input documents, but they rarely succeed in generating summaries rephrasing something actually stated in the article. Table 7.3

shows an example output that is fairly grammatically correct, but not truthful with respect to the article.

Table 7.3: Model output and reference summary for a document-query pair in the test set.

Document president barack obama sided with open-internet activists on monday , urging the federal communications commission to draft new rules that would reclassify the broadband net to regulate it more like a public utility . the end result would tie the hands of internet service providers that want to cut special deals with services like netflix , youtube , hulu and amazon to push their streaming content along a ' fast lane ' that ordinary americans ca n't access . [...]
Query netflix
Reference obama 's vision would bar providers like verizon and comcast from cutting deals with hulu , netflix and amazon so their streaming content could be delivered along online ' fast lanes '
Output obama 's chief executive of netflix has refused to allow users to access the service

We observe that the model manages to learn some of the dataset samples which are not actual summaries, described in Chapter 5, such as notices repeated over several articles. The generated summary shown in Table 7.4 is an example of this. Interestingly, the model manages to literally repeat the reference summary, up to the maximum output length limit.

Table 7.4: Model output and reference summary for a document-query pair in the test set.

Document february 13 , 2015 a breakthrough in belarus , a verdict in italy , and an expected veto in the u.s. all headline cnn student news this friday . [...]
Query cnn student news roll call
Reference at the bottom of the page , comment for a chance to be mentioned on cnn student news . you must be a teacher or a student age 13 or older to request a mention on the cnn student news roll call .
Output at the bottom of the page , comment for a chance to be mentioned on cnn student news . you must be a teacher or a student age 13 or older to

We can frequently see repetitions of the same phrases; an extreme example can be

seen in Figure 7.2. The model appears to get stuck trying to begin a summary. Additionally, we observe that the repetition can be observed in the attention distribution as well. The same problem has been seen by Nallapati et al. (2016), who make an addition, *temporal attention* (Sankaran et al., 2016), to their model for alleviating the issue of repetitions. See et al. (2017) propose using *coverage* to solve the same issue.

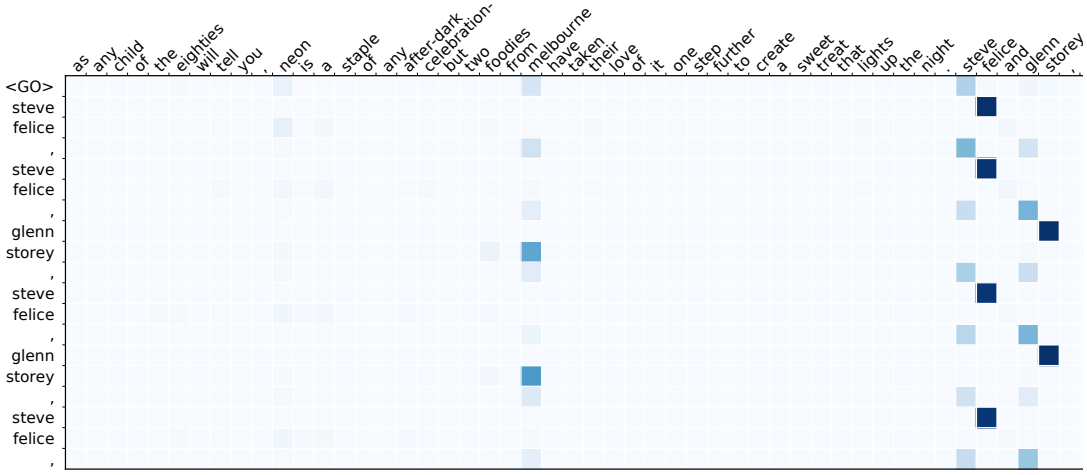


Figure 7.2: Visualization of the attention distribution, α_{ti}, as an output summary for a document-query pair is generated. The query is "australia". The format is the same as in Figure 7.1.

Before running experiments, we suspected that it may be difficult for the pointer mechanism to sequentially point out words that make up longer entities. However, we see that this is done successfully quite often. For an example summary, the certainty of selecting a sequence of entity words can be seen in Figure 7.3.

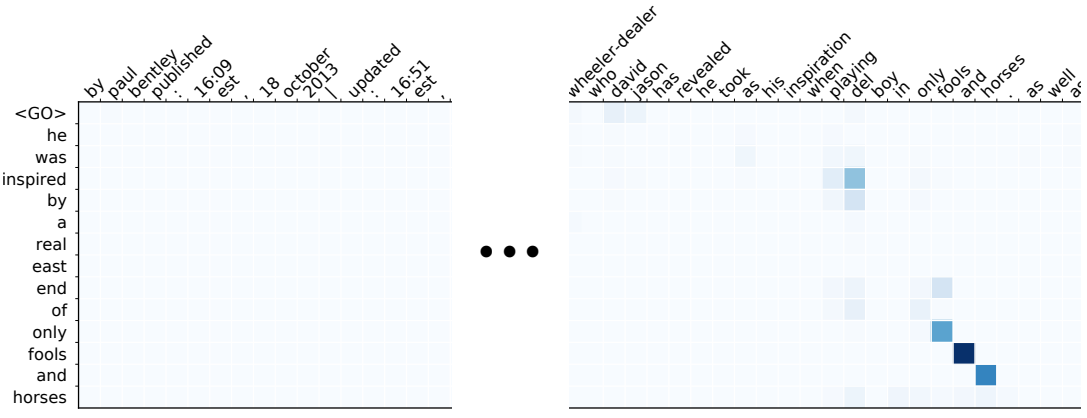


Figure 7.3: Visualization of the attention distribution, α_{ti}, as an output summary for a document-query pair in the test set is generated. The query is "only fools and horses". The format is the same as in Figure 7.1. The ellipsis signifies that parts of the attention distribution has been skipped.

Compared to the reference summaries, the output is generally shorter. The average

number of words in output summaries is 11.27, while the dataset average is 14.44. As is noted by Wu et al. (2016), beam search commonly favors shorter summaries. They propose an addition of *length normalization*, for reducing this tendency. Implementing such a measure may improve the results of our model as well.

In comparison to Nallapati et al. (2016) and See et al. (2017), our ROUGE scores are low. They use a different version of the dataset where all highlights are combined to form a single, often multi-sentence, summary. With similar models, they get ROUGE-1 results of around 35 on the general summarization task. Speculatively, it might be that our training procedure may not be as effective. While they always train the model to output the same summary for the same document, we often have completely different target summaries for different queries, where the queries make up a much smaller part of the input. Additionally, differences in the training time of the models are discussed in Section 6.3.

7.1 Query Dependence

The result of the query dependence evaluation, described in Section 6.6.1, can be seen in Table 7.5. We can see that the ROUGE scores for the normal evaluation is higher for all metrics, with statistical significance, according to the ROUGE-reported 95% confidence intervals. This indicates that the model benefits from the information provided by queries.

Table 7.5: ROUGE F-scores of our model, from the normal evaluation and with offset queries, as presented in section 6.6.1.

	ROUGE			
Evaluation of	1	2	L	SU4
Our model	**18.25**	**5.04**	**16.17**	**6.13**
Our model + offset queries	16.06	3.89	14.25	5.18

The difference in scores might not appear very large, but there are some possible factors explaining why relatively high scores may be reached with offset queries. Firstly, a part of the ROUGE score is a result of simply generating common words and punctuation marks, such as *the*, commas, and quotation marks. Furthermore, when constructing the dataset, we often choose multiple queries from the same highlight. This means that identical reference summaries for different queries are rather common, and so, when we intend to evaluate against a different query for the same document, we might actually evaluate against a summary appropriate for the query. For these reasons, it is difficult to tell exactly how much the resulting output depends on the query, but we can tell that it is statistically significant.

From examining output summaries, we see that the query does not consistently influence what aspect of the article is summarized. Neither is the query consistently

repeated in output summaries, which they are in summaries the model has been trained on.

The model does not always seem successful in constructing a summary where inserting the query word would be appropriate. In some instances, we see that the query is attended to, but not at the time steps that would be appropriate. An example of this is in an article named "'It was absolutely enormous': Meteor explodes over Arizona on the eve of the year's biggest cosmic shower". In the article, they mention the meteor shower Geminid, the query in this example, a number of times, but it never appears in the generated summary. From inspecting the attention distributions, we see that the word "geminid" has strong attention at several locations in the input and over several time steps, but not at a time step where it would make grammatical sense, which may explain why it is not copied with the pointer mechanism. The following sentence shows words in bold that would have been "geminid" if the decoder had decided to use the pointer mechanism instead of generating a word: "footage **captured** by facebook **shows** a light light on the show".

Additionally, we have observed that output for the same document with different queries, X and Y, can give summaries like "X scored two goals" and "Y scored two goals", i.e. the query did not influence what is to be stated, but only the entity selected through the pointer mechanism. Even if the statement is truthful for one of them, it likely is not for both.

8

Conclusion and Future Work

We have designed a model for query-based abstractive summarization and evaluated it on a new dataset designed for the task. While the overall performance of the model is not enough to outperform our extractive baseline, we have shown that we can incorporate a query in a neural network summarization model and utilize the information to create more focused summaries.

We find several recurring issues with the generated summaries, some similar to what has been noticed for general abstractive summarization (Nallapati et al., 2016; See et al., 2017): (1) the model sometimes gets stuck repeating the same phrase over and over, (2) the summaries are not consistently relevant to the query, and (3) the summaries are rarely truthful.

The first of the mentioned problems has been explored by Nallapati et al. (2016) and See et al. (2017). They use *temporal attention* and *coverage* respectively, and it should be possible to extend our model with one of these techniques.

The second problem could be caused by the model having difficulties in locating what is relevant to the queries. A property of this dataset is that the exact query sequence always appears in the query-based summary. This is however not necessary in query-based summaries in general, and we have for this reason designed the query usage in the model in a general fashion. For this limited dataset, we believe one can improve the performance by making an explicit shortcut for repeating the entire query sequence in the output. For a more general solution, one might try to help the model by marking literal occurrences of the query words in the document. In the current model, such matches would instead have to be found through word embeddings, propagated through the RNNs. It could also be possible to identify query occurrences through the use of *coreference resolution* or similar, to not only capture literal occurrences, but also similar occurrences of the same entity, such as also marking "Obama" if the query was "Barack Obama".

The third problem is lessened to some degree when using pointer-generator models. However, our model still struggles with, for instance, choosing the correct names at the correct places. More broadly, truly solving the problem of abstractively summarizing a text seems to require a level of understanding of texts that is quite far from what we observe in our experiments.

A different approach to the task of generating natural language that we find interesting would be to, instead of generating a sentence from beginning to end, allow generation

of words on either side of previously generated words. This could for example lead to a model learning to first output a word very central to the query, and then build a sentence around this word. With this dataset, we could explicitly output the query as the initial word, and we would avoid the issue of not reaching a point in the summary generation where the query word would fit.

There are also ethical considerations related to trying to summarize text automatically. In some sense, an abstractive summarization system takes on the same role as a person attempting to rephrase information in a document, and similar ethical difficulties are present. A lack of control of the text generated by the system is problematic. Summaries might be accidentally offensive in some way, or even completely contradict the original text. For instance, it might be the case that a negation is not considered, and therefore a summary states the opposite of the original text. Someone unaware of the possibility of flaws in the generated summary might become misinformed. This problem is however, to a limited extent, not unique to abstractive summarization, as even extractive summarization systems can present information taken out of context, where surrounding text may be important for interpreting it correctly.

For summarizing results in a search engine, the resulting summaries from our model would likely not be very useful. However, we believe that models with high expressiveness, using natural language generation, can be made reliable enough to serve a role beyond what is possible with an extractive summary.

Bibliography

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from `tensorflow.org`.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *International Conference on Learning Representations (ICLR 2015)*. arXiv: `1409.0473 [cs.CL]`.

Bengio, Samy, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer (2015). "Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., pp. 1171–1179.

Bengio, Yoshua (2012). "Practical Recommendations for Gradient-Based Training of Deep Architectures". In: *Neural Networks: Tricks of the Trade: Second Edition*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 437–478.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2, pp. 157–166.

Bird, Steven, Ewan Klein, and Edward Loper (2009). *Natural Language Processing with Python*. 1st. O'Reilly Media, Inc.

Cho, Kyunghyun, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734.

Chung, Junyoung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio (2014). "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *ArXiv e-prints*. arXiv: 1412.3555 [cs.NE].

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

Goldstein, Jade, Mark Kantrowitz, Vibhu Mittal, and Jaime Carbonell (1999). "Summarizing Text Documents: Sentence Selection and Evaluation Metrics". In: *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '99. Berkeley, California, USA: ACM, pp. 121–128.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.

Gülçehre, Çaglar, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio (2016). "Pointing the Unknown Words". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 140–149.

Hermann, Karl Moritz, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom (2015). "Teaching machines to read and comprehend". In: *Advances in Neural Information Processing Systems*, pp. 1693–1701.

Hochreiter, Sepp (1991). "Untersuchungen zu dynamischen neuronalen Netzen". PhD thesis. diploma thesis, institut für informatik, lehrstuhl prof. brauer, technische universität münchen.

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

Jean, Sébastien, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio (2015). "On Using Very Large Target Vocabulary for Neural Machine Translation". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1–10.

Karpathy, Andrej and Li Fei-Fei (2015). "Deep Visual-Semantic Alignments for Generating Image Descriptions". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR 2015)*. arXiv: 1412.6980 [cs.CL].

Kumar, Ankit, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher (2016). "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, pp. 1378–1387.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lin, Chin-Yew (2004). "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*. Ed. by Stan Szpakowicz Marie-Francine Moens. Barcelona, Spain: Association for Computational Linguistics, pp. 74–81.

Merity, Stephen, Caiming Xiong, James Bradbury, and Richard Socher (2017). "Pointer Sentinel Mixture Models". In: *International Conference on Learning Representations (ICLR 2017)*. arXiv: `1609.07843 [cs.CL]`.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). "Distributed Representations of Words and Phrases and Their Compositionality". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., pp. 3111–3119.

Nallapati, Ramesh, Bowen Zhou, Cicero Nogueira dos Santos, Çaglar Gülçehre, and Bing Xiang (2016). "Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond". In: *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, pp. 280–290.

Nema, Preksha, Mitesh Khapra, Anirban Laha, and Balaraman Ravindran (2017). "Diversity driven Attention Model for Query-based Abstractive Summarization". In: *ArXiv e-prints*. arXiv: `1704.08300 [cs.CL]`.

Nenkova, Ani and Kathleen McKeown (2012). "A Survey of Text Summarization Techniques". In: *Mining Text Data*. Ed. by Charu C. Aggarwal and ChengXiang Zhai. Boston, MA: Springer US, pp. 43–76.

Otterbacher, Jahna, Gunes Erkan, and Dragomir R. Radev (2009). "Biased LexRank: Passage Retrieval Using Random Walks with Question-based Priors". In: *Information Processing and Management* 45.1, pp. 42–54.

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

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–538.

Rush, Alexander M., Sumit Chopra, and Jason Weston (2015). "A Neural Attention Model for Abstractive Sentence Summarization". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 379–389.

Sankaran, Baskaran, Haitao Mi, Yaser Al-Onaizan, and Abe Ittycheriah (2016). "Temporal Attention Model for Neural Machine Translation". In: *ArXiv e-prints*. arXiv: 1608.02927 [cs.CL].

Schuster, Mike and Kuldip K Paliwal (1997). "Bidirectional Recurrent Neural Networks". In: *IEEE Transactions on Signal Processing* 45.11, pp. 2673–2681.

See, Abigail, Peter J. Liu, and Christopher D. Manning (2017). "Get To The Point: Summarization with Pointer-Generator Networks". In: *ArXiv e-prints*. arXiv: 1704.04368 [cs.CL].

ShafieiBavani, Elaheh, Mohammad Ebrahimi, Raymond Wong, and Fang Chen (2016). "A Query-Based Summarization Service from Multiple News Sources". In: *2016 IEEE International Conference on Services Computing (SCC)*, pp. 42–49.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). "Sequence to Sequence Learning with Neural Networks". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems*. NIPS'14. Montreal, Canada: MIT Press, pp. 3104–3112.

Tan, Ming, Bing Xiang, and Bowen Zhou (2015). "LSTM-based Deep Learning Models for Non-factoid Answer Selection". In: *ArXiv e-prints*. arXiv: 1511.04108 [cs.CL].

Taylor, Wilson L (1953). "'Cloze procedure': a new tool for measuring readability". In: *Journalism Bulletin* 30.4, pp. 415–433.

Wang, Lu, Hema Raghavan, Vittorio Castelli, Radu Florian, and Claire Cardie (2013). "A Sentence Compression Based Framework to Query-Focused Multi-Document Summarization". In: *ACL*.

Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean (2016). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *ArXiv e-prints*. arXiv: 1609.08144 [cs.CL].

Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio (2015). "Show, attend and tell: Neural image caption generation with visual attention". In: *International Conference on Machine Learning*, pp. 2048–2057.