# Non-Autoregressive Unsupervised Summarization with Length-Control Algorithms

by

Puyuan Liu

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

# Abstract

Text summarization aims to generate a short summary for an input text and has extensive real-world applications such as headline generation. State-of-the-art summarization models are mainly supervised; they require large labeled training corpora and thus cannot be applied to less popular areas, e.g., less spoken languages, where paired data are rare.

In this thesis, I present a non-autoregressive unsupervised summarization model, which does not require parallel data for training. Our approach first performs edit-based search towards a heuristically defined score and generates a summary as pseudo-groundtruth. Then, we train an encoder-only non-autoregressive Transformer based on the search results. Further, we design two length-control algorithms for the model, which perform dynamic programming on the model output and are able to explicitly control the number of words and characters in the generated summary, respectively. Such length control is important for the summarization task, because the main evaluation metric for summarization systems, i.e., ROUGE score, is sensitive to the summary length, and because real-word applications generally involve length constraints.

Experiments on two benchmark datasets show that our approach achieves state-of-the-art performance for unsupervised summarization, yet largely improves inference efficiency. Further, our length-control algorithms are able to perform length-transfer generation, i.e., generating summaries of different lengths than the training target.

*"Half of the people who have embarked on a one hundred mile journey may fall by the way side"*

*—Strategies of the Warring States*

# Acknowledgements

I am deeply grateful to my supervisor Dr. Lili Mou, who provided crucial guidance and constant support to my research over the past two years. This short but fruitful journey not only helped me establish scientific thinking but largely broadened my horizon in Machine Learning research.

Most content of this thesis is drawn from papers completed during my stay in the NLP research group of Dr. Mou, and they would never be done without his incredibly patient supervision. Moreover, I really appreciate the co-authors of my papers: Chenyang Huang and Xiang Zhang, who brought ideas of non-autoregressive generation and assisted me to complete character-level length-control experiments. I would also like to thank other group members for their valuable suggestions to my research work and excellent weekly presentations: Anup Deshmukh, Mauajama Firdaus, Yongchang Hao, Dongheng Li, Yuxin Liu, Steven Lu, Guoqing Luo, Pourya Vakilipourtakalou, Yuqiao Wen, Zijun Wu, Qianqiu Zhang, and Zixuan Zhang.

Last but not least, I want to express special thanks to my families, who support me either emotionally or financially during my studies, including my admirable parents, gorgeous fiancée, and two cute black cats. Especially, my fiancée can always point me in the right direction and encourages me against trouble; no matter what I am confronted with, tranquility nestles in my heart with her accompany.

The research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant No. RGPIN2020-04465, the Amii Fellow Program, the Canada CIFAR AI Chair Program, a UAHJIC project, a donation from DeepMind, and Compute Canada (www.computecanada.ca).

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

Over the past few decades, the volume of textual data on the Internet has been growing rapidly. According to [87], there are over 4 billion web pages online as of 2016, many of which are based on text. Such amount of available text has led to the rise of many new fields, such as public opinion analysis [19, 31, 99], social spam detection [53, 105, 70], and sentiment analysis [16, 54, 16].

However, it is infeasible to tackle such tasks by manually analyzing the data, and automated tools are thus needed. Automatic text analysis is the research goal of Natural Language Processing (NLP), which is an interdisciplinary subject that involves linguistics, computer science, and artificial intelligence. Example tasks of NLP include question answering [8, 100, 75], information extraction [50, 52, 79], and text summarization [104, 1, 2].

In the early stage of NLP research, the typical method is based on hand-coded rules, such as grammars [92]. These rule-based approaches lack robustness against erroneous input (e.g., misspelled words), and become more difficult to manage when the number of rules grows. In the past decades, researchers have labeled a large number of corpora, and with the growth of computational resources, modern NLP methods are mainly data-driven and utilize machine learning techniques to automatically learn the desired tasks. These methods do not require human-specified rules and are thus less prone to

the above drawbacks.

In this thesis, we focus on text summarization, an important NLP task that aims at generating concise summaries for given texts while preserving the key information. Summarization has extensive real-world applications such as generating headlines for online documents, which can assist users to efficiently browse a large volume of textual data and alleviate the information overload problem [17].

Most of the early summarization systems [51, 13, 26, 21] output a summary by extracting salient content from the source text based on heuristically defined rules, e.g., extracting sentences with descriptive words [51]. On the contrary, modern summarization models are mainly based on deep learning [1, 2, 89] because of the effectiveness of neural networks on text generation [82, 46, 29]. These models are able to learn the task by training with human-written summaries, so that they yield outputs better than the previous rule-based approaches on various benchmark datasets. For example, Aghajanyan et al. [1] finetune a Bidirectional AutoRegressive Transformer (BART) [40], which gives state-of-the-art performance on Gigaword [22] and CNN/DailyMail [27] datasets.

However, these state-of-the-art summarization systems require massive parallel corpora (comprising long texts and their summaries) [104, 1, 2]. They are, unfortunately, expensive to obtain, preventing the applications of these systems to less popular domains and less spoken languages.

Therefore, unsupervised summarization is attracting increasing interest, as it does not require parallel data for training. One widely used approach is to compress a long text into a short one, and reconstruct it to the long text by a cycle-consistency loss [57, 90, 4]. However, the compressed sentence space is not differentiable, and such a method demands reinforcement learning or its alternatives for training, which is usually difficult [36].

Schumann et al. [76] present a word-extraction method for unsupervised summarization, which extracts a certain number of words from the source text as the summary.

Their approach defines a scoring function that evaluates the output summary based on heuristics and utilizes hill-climbing search to find certrain extracted words that can maximize the scorer. Although this method outperforms the cycle-consistency ones, it suffers from slow inference since it demands hundreds of search steps for each data sample. Moreover, the order of the extracted words is preserved, and thus the generated summary may be restricted and noisy.

## 1.2 Thesis Contributions

In this thesis, we propose a non-autoregressive approach to unsupervised summarization, which does not require parallel data for training and is able to predict all target tokens simultaneously. Moreover, we design two different length-control algorithms, which are able to explicitly constrain the number of words and characters, respectively, in a predicted summary.

Specifically, our approach utilizes the learning-from-search framework [41]; we adopt the search-based method [76] to generate pseudo-groundtruth summaries and then train a non-autoregressive model with the summaries. We propose an encoder-only Transformer [88] as the non-autoregressive architecture, composed of deep Transformer layers with residual connections, which are able to capture the strong correspondence between the input and output in the text summarization task. But then the output of such an architecture has the same length as its input and cannot be a summary. To address this issue, we further propose to train the model with the Connectionist Temporal Classification [CTC, 23] algorithm, which learns to insert blank tokens into the model output. These blank tokens will be removed later, yielding a shorter text than the model input and making summarization possible. Experiments show that our model outperforms not only its search-based teacher [76] but also all previous methods in terms of the ROUGE scores. Regarding inference efficiency, our method is over a thousand times faster than its search-based teacher [76] and roughly 10 times faster than an autoregressive Transformer [88].

Further, we propose two length-control algorithms, which perform dynamic programming (DP) on the outputs of our non-autoregressive model and are able to explicitly control the number of words and characters in a summary. Our DP divides the length-control problem into shared sub-problems. This is feasible because the non-autoregressive model predicts the probabilities independently at different steps. Specifically, our word-level length-control algorithm iteratively fills up a DP table that keeps the (approximately) most probable summaries of different lengths given the first several model outputs. Experimental results show that, when the number of words in a summary is constrained, our length-control algorithm not only achieves better performance than all other methods but also yields more complete summaries than naïvely truncating the over-lengthed outputs. Our length-control algorithm further enables length-transfer generation, i.e., generating summaries of different lengths from the training targets.

Our character-level algorithm, on the other hand, formulates length control as a knapsack-like problem, where the weight is the number of characters in a token and the value is the predicted probability of the token. Similar to the word-level length control, our character-level algorithm outperforms all other approaches when the number of characters in summaries is constrained, while retaining high inference efficiency. Also, the algorithm is capable of character-level length-transfer generation.

To sum up, the main contributions of this thesis include:

- We propose an effective and efficient non-autoregressive approach to unsupervised summarization.
- We design word-level and character-level length-control algorithms, which can explicitly control the number of words and characters, respectively, in the predicted summaries. To better understand the algorithms, we further conduct theoretical analysis of their exactness.
- We evaluated our approach on the Gigaword headline generation [72] and

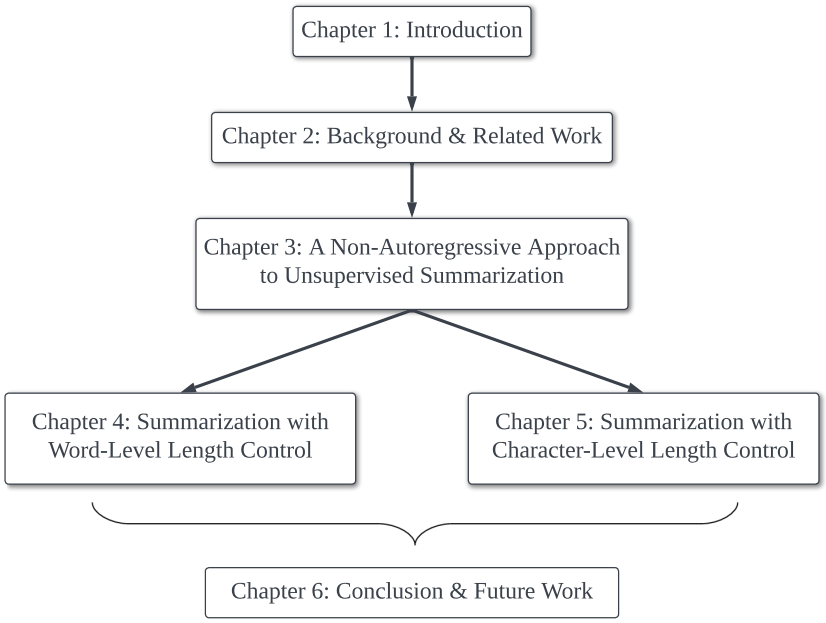


Figure 1.1: Roudmap of this thesis.

DUC2004 [22] datasets. Our approach achieves state-of-the-art unsupervised performance and is at least several times more efficient than the autoregressive Transformer [88].

## 1.3 Thesis Outline

This chapter introduces the background of natural language processing for the text summarization task, and gives an overview of our approach.[1]

Chapter 2 presents the related work, including summarization systems and text generation approaches.

Chapter 3 describes our non-autoregressive approach to unsupervised summarization, including the search-based method [76] that our model learns from, the architecture of our non-autoregressive summarization model, and our training strategy. Further, I will show the performance of our model in comparison with previous work on benchmark datasets when the summary length is not controlled.

[1]Part of the contents in Chapters 3 and 4 were published in Liu et al. [44]. Copyright©2022, Association for Computational Linguistics. Reused with permission.

Chapters 4 and 5 present the length-control algorithms and our theoretical analysis of the algorithms' exactness. Moreover, I will show their empirical performance on benchmark datasets and additional analyses of our approach.

Chapter 6 concludes the thesis, and discusses the limitation of our approach and future work.

# Chapter 2

# Background & Related Work

This chapter presents background and related work. Specifically, I will introduce the summarization task in §2.1 and text generation models in §2.2.

## 2.1 Summarization Task

Text summarization is an important task in Natural Language Processing (NLP), aiming at compressing a long text into a short one that keeps the main gist of the input. Since the 1950s, numerous methods have been proposed to solve the task [10, 58, 63, 67].

Summarization models can be generally categorized into two types: extractive and abstractive. Extractive methods output a summary by extracting important sentences or clauses from the source text [10, 58, 33], while abstractive methods are able to generate summaries with new expressions [63, 67, 20]. I will discuss these two approaches in §2.1.1 and §2.1.2, respectively.

Moreover, I will introduce unsupervised summarization (§2.1.3), which does not require any labeled training data and is applicable to less popular domains where paired data are rare. Besides, I will present recent work on summarization with length control in §2.1.4, due to the importance of controlling output length for real-world applications and fair comparison between different models [76, 74]. Last but not least, I will discuss non-autoregressive summarization (§2.1.5), which predicts output

tokens in parallel and is much more efficient than traditional autoregressive models at inference time.

### 2.1.1 Extractive Summarization

The extractive approach is widely adopted in early summarization systems [51, 13, 26, 21], which generate a summary by extracting source content (e.g., sentences) based on a heuristically defined scorer.

For example, Luhn [51] establishes a set of important words and then produces a summary by extracting source sentences with high coverage of the selected words. Fattah and Ren [15] weight source sentences based on more complicated features such as sentence relative length, sentence position, and whether numerical values are included. Erkan and Radev [14], by contrast, adopt a graph-based approach to measure the salience of sentences in the source text. Specifically, they model the source sentences as a graph, where edges are weighted by cosine similarity scores. A sentence is then scored by eigenvector centrality derived from the graph adjacency matrix.

Recent summarization methods are mainly data-driven and utilize machine learning approaches to evaluate sentences in the source text [62, 48, 106, 64]. Nallapati et al. [62], for instance, adopt a Recurrent Neural Network (RNN) as the backbone and treat the extractive summarization as a sequential classification problem. Specifically, each sentence in the source text is fed into the model sequentially, and a binary predictor is applied to determine whether the sentence should be included in the summary. Liu [48] finetunes a Bidirectional Encoder Representation Transformer [BERT, 34] such that the model can predict the binary extraction label in parallel for all sentences in the input text. Zhong et al. [106] formulate summarization as a text-matching problem in the semantic space, i.e., extracting source sentences with similar meaning to the reference summaries. Compared with the early approaches, these new methods are able to generate much better summaries.

However, the output summary of extractive methods is essentially a combination of extracted textual segments from the source text, lacking expression flexibility.

### 2.1.2 Abstractive Summarization

Abstractive methods can produce summaries with new expressions. They are mostly data-driven and vastly used in the neural era. Compared with the extractive ones, abstractive models possess much higher generation flexibility.

Modern abstractive summarization systems are mainly built with the encoder–decoder architecture, which employs two deep learning models as the encoder and decoder, respectively; the encoder converts the source text into continuous vector representations, from which the decoder generates the summary. The encoder–decoder model is also known as Seq2Seq, since it accepts a sequence as the input and produces another as the output. Commonly used networks for the Seq2Seq model include the Convolutional Neural Network [CNN, 96, 49, 60], Long-Short-Term-Memory [LSTM, 78, 77, 102], and the Transformer [1, 2, 84].

State-of-the-art Seq2Seq summarization systems are developed mainly by finetuning pretrained language models. For example, Aghajanyan et al. [1] refine a Bidirectional AutoRegressive Transformer (BART) [40] based on the trust-region theory. Specifically, they constrain movements in the representational density space for each optimization step to avoid the representational collapse, i.e., a massive decrease in the generalization ability of pretrained language models' internal representation. Experiments show that their approach yields a more generalizable representation than naïve finetuning, achieving better performance on various downstream tasks including summarization. Later, Aghajanyan et al. [2] introduce an additional step before tweaking the pretrained language models, named pre-finetuning. This intermediate step is essentially a massive multi-task finetuning, which is able to further improve the generalization ability of the representations and boost the summarization performance. Zhang et al. [104], by contrast, design a task-specific pre-training strategy for text summarization; they first

mask out salient sentences in the input text and then predict the masked content. Since the pretraining target is similar to the downstream summarization task, i.e., the essential contents in the source text, their approach can also boost the summarization performance.

### 2.1.3 Unsupervised Summarization

Current state-of-the-art summarization systems, whether extractive or abstractive, are typically trained in a supervised way with large training corpora, which contain source texts and their summaries [104, 1, 2]. However, such parallel data are expensive to obtain, preventing the applications to less popular domains and less spoken languages. Therefore, unsupervised text generation has been attracting increasing interest, as it does not require parallel data for training.

Previous work on unsupervised summarization can be roughly categorized into three groups.

The first group utilizes the structural information of the source text. For example, Yang et al. [101] propose to use the lead baseline, i.e., the first several sentences in the source text, as the pseudo-groundtruth. However, such approaches only work with well-structured texts such as news articles, where topic sentences exist and can be found in fixed locations.

The second group takes advantage of the information bottleneck: Wang and Lee [90] and Baziotis et al. [4] use cycle consistency for unsupervised summarization. Specifically, they compress the source text into a summary and then reconstruct the source from it, minimizing the difference between the source text and the reconstructed one. However, these methods usually require reinforcement learning or its variants for the model training because the compressed sentence space is indifferentiable; this makes the training difficult.

The third group adopts heuristically defined scoring functions to generate summaries. For instance, Schumann et al. [76] propose an edit-based local search approach, which

outperforms the cycle-consistency models. Specifically, they randomly extract several words from the source sentence as the summary. Then they iteratively change the selection and non-selection of two words to maximize a heuristic function that evaluates the summary fluency and information preservation. One major drawback of this method is the slow inference, since it requires hundreds of local search to produce the summary for each sample.

Conversely, our approach possesses a much higher inference efficiency because we employ a machine learning model as the backbone and do not need to perform any search during inference.

### 2.1.4 Summarization with Length Control

Recently, Liu et al. [49] show that length control is the key to summarization, since it is typically a requirement by real-world applications, such as fitting the screen width. Moreover, the main evaluation metric for summarization systems, i.e., the ROUGE score [43], is found to be sensitive to the summary length [76, 74]; summarization systems may achieve higher scores by simply generating longer output.

In early extractive summarization research, truncating is adopted for fair comparison [58, 61, 33], but the resulting summary may not be complete sentences. As pointed out by [74, 76], the length-control problem is not adequately addressed for summarization models in the neural era, probably because researchers do not want to hurt the completeness.

Most of the previous summarization approaches control the output length by feeding length information together with the source text into a machine learning model, which is expected to automatically learn the length control from the input length. For example, Liu et al. [49] inject the length information by rescaling the input embedding based on the desired output length. Takase et al. [85] encode the remaining length budget into the positional embeddings of the decoder input (token) in each decoding step. Saito et al. [74] first build a summary prototype by extracting a set of key

sentences based on the desired summary length. Then they feed the prototype to a Transformer model [88] to finalize the summary. However, empirical results show that these approaches cannot explicitly control the summary length and may generate summaries longer than the given length budget.

There are also methods that can perform explicit length control but have their own limitations. Kikuchi et al. [35], for instance, propose two ad hoc modules to control the output length at inference time. The first module inhibits the decoder from generating the EOS tag before reaching the desired summary length and stops decoding after finishing up the length budget. By contrast, the second one only keeps sequences whose lengths are less than or equal to the desired length during the beam search. Despite the simplicity, these heuristic methods cannot guarantee the completeness and quality of the produced summaries. Schumann et al. [76] perform constrained discrete optimization by selecting a certain number of words from the source text as the output. However, their approach suffers from extremely slow inference because hundreds of local search steps are needed for each data sample. Moreover, their method can only control the number of words in the summary and is not able to constrain the summary length by characters.

### 2.1.5 Non-Autoregressive Summarization

Non-autoregressive summarization models predict all summary tokens in parallel. They have a much higher inference efficiency than the traditional autoregressive ones, which generate one token at a time. However, non-autoregressive summarization generally has a worse output than autoregressive methods, due to the lack of dependencies between the simultaneously predicted summary tokens.

Recently, researchers are attempting to alleviate this drawback: Yang et al. [98] employ an additional autoregressive model to predict the Part-of-Speech (POS) tags for the summary and feed the source text together with the POS tags into a non-autoregressive Transformer [25] to generate summaries. Su et al. [80] utilize a pretrained Bidirectional

Encoder Representation Transformer [BERT, 34] as the non-autoregressive backbone and develop an extra Conditional Random Field (CRF) to perform structure decoding based on the non-autoregressive outputs. Jia et al. [32] generate a summary by simultaneously extracting sentences from the source text based on their encoding given by the ALBERT model (an enhanced version of BERT) [38]. Qi et al. [68] first pretrain an encoder–decoder non-autoregressive model on a large text corpus and then finetune the pretrained model for the summarization task.

However, none of the current non-autoregressive summarization models can match the performance of even a standard autoregressive Transformer [88]. Our approach, by contrast, shrinks this performance gap and can even outperform the autoregressive Transformer in the unsupervised setting.

## 2.2 Text Generation Approaches

Text generation is the process of producing human readable texts such as dialogues [94, 42, 45] and summaries [1, 49, 10]; it is an important research area in natural language processing.

In this section, I will introduce three text generation approaches related to this thesis: autoregressive, non-autoregressive, and search-based.

### 2.2.1 Autoregressive Generation

Autoregressive (AR) models are widely adopted for modeling time-varying processes such as economics [59] and climate [5]; such models make predictions for each time step based on the previous predictions.

Most of the state-of-the-art text generation models are AR, and predict target tokens sequentially. The probability of a model output $\mathbf{y}$ is given by:

$$P(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{T} P\left(\mathrm{y}_i \mid \mathbf{y}_{<i}, \mathbf{x}\right) \tag{2.1}$$

where $\mathbf{x}$ is the model input, $T$ is the generation length, $\mathrm{y}_i$ is the prediction at $i$th time steps, and $\mathbf{y}_{<i}$ refers to the predictions prior to the $i$th step.

Common AR architectures for text generation include the recurrent neural network [RNN, 71] and Transformer [88]. These architectures have been adopted as the backbone for state-of-the-art models of various text generation tasks, such as dialogue generation [45, 86, 93], translation [95, 97, 83], and summarization [104, 1, 2].

However, AR models can only predict one target token at a time and are not able to fully utilize the parallel processing of modern hardware (e.g., GPU), which largely limits their inference efficiency.

### 2.2.2 Non-Autoregressive Generation

Non-autoregressive (NAR) models predict all target tokens in parallel, i.e.,

$$P(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{T} P\left(\mathrm{y}_i \mid \mathbf{x}\right) \tag{2.2}$$

Thus, they can fully utilize the parallel processing of GPU/CPU and enjoy a much higher inference efficiency than the AR approaches.

On the other hand, NAR generation is more difficult than AR due to the lack of dependencies among the simultaneously generated output tokens. Previous work addresses the dependency issue by iterative refinement [39, 56, 7] or structured decoding with the conditional random field (CRF) [81, 80, 9]. Another approach is training non-autoregressive models with the Connectionist Temporal Classification (CTC) algorithm [23], which is able to address a common problem in NAR generation, namely, token repetition, by merging consecutive identical tokens (unless separated by an empty token).

Recently, text generation approaches with high inference efficiency, such as NAR models, are drawing increasing attention. This is because inference efficiency has become an important evaluation metric for text generation models as a result of the deployment to low-resource devices (e.g., mobile phones).

### 2.2.3 Search-Based Generation

Search-based generation produces texts by iterative edits, and can be generally divided into three steps:

- Construct a scoring function for the output, such as cosine similarity between the embeddings of the source text and model output.
- Define the action space of search, such as deleting a word from the intermediate output; and
- Employ a search algorithm (e.g., hill-climbing search) to iteratively edit the output to maximize the pre-defined scoring function.

Previous search-based text generation methods are mainly applied to the unsupervised setting, where groundtruth labels are unknown and the training target needs to be heuristically defined. For example, Liu et al. [47] develop an unsupervised paraphrase model by simulated annealing. Specifically, they design a heuristic function that involves the semantic similarity between the output paraphrases and the source text, as well as the output expression diversity and fluency. In each iteration, an edit action for the paraphrase is chosen from insertion, deletion, and replacement; the search agent then determines whether the action should be accepted. Kumar et al. [37] propose an unsupervised approach to sentence simplification; they adopt a similar action space to Liu et al. [47] but change the "replacement" action to "keep". Moreover, their scoring function is defined to be a mixture of semantic preservation, keyword coverage, and output length. Schumann et al. [76] present an unsupervised summarization method based on hill-climbing search, extracting a specific number of words from the source text as the summary. Similar to Liu et al. [47] and Kumar et al. [37], Schumann et al. [76] evaluate the output summary based on semantic preservation and language fluency.

However, these methods all suffer from slow inference since enormous search steps

are needed to perform inference for each data sample. Moreover, the generalization ability of the search-based method is generally limited since the scoring functions of the model output are heuristically defined.

In summary, this chapter reviews the related work of the summarization task and text generation models. Based on the review, we would propose a learning-from-search approach to unsupervised summarization with additional length-control algorithms to address the drawbacks of previous work.

# Chapter 3

# A Non-Autoregressive Approach to Unsupervised Summarization

## 3.1 Overview

Unsupervised summarization, in contrast to supervised summarization that requires large training corpora, demands no labeled training data and is thus suitable for less popular domains, where paired data are difficult to collect. Previous unsupervised models are mostly trained with cycle consistency [57, 90, 4], compressing the source text into a short one and then reconstructing the source from the compressed text. Due to the indifferentiability of the compressed sentence space, such an approach requires reinforcement learning (or its variants), which makes the training difficult [36].

Recently, Schumann et al. [76] propose an edit-based approach to unsupervised summarization. Their model maximizes a heuristically defined scoring function that evaluates the quality (e.g., fluency and semantics) of the generated summary with length constraints, achieving higher performance than cycle-consistency methods. However, the search approach is slow in inference because hundreds of search steps are needed for each data sample. Moreover, their approach can only select words from the input sentence with the word order preserved. Thus, it is restricted and may generate noisy summaries due to the local optimality of search algorithms.

In this chapter, we propose a non-autoregressive approach to unsupervised summarization, shown in Figure 3.1. The idea is to perform search as in Schumann et al. [76]

and, inspired by Li et al. [41], to train a machine learning model to smooth out search noise and to speed up the inference process. Different from Li et al. [41], we propose to utilize *non-autoregressive* decoders, which generate all output tokens in parallel due to our following observations:

- Non-autoregressive models are several times faster than autoregressive generation, which is important when the system is deployed.
- The input and output of the summarization task have a strong correspondence. Non-autoregressive generation supports encoder-only architectures, which can better utilize such input–output correspondence and even outperform autoregressive models for summarization.

In this chapter, we first introduce the search-based summarization teacher [76] and our proposed non-autoregressive model and training strategy in §3.2. Then, we show the empirical performance of our approach in comparison with previous methods on benchmark datasets in §3.3.

## 3.2 Methodology

### 3.2.1 Search-Based Summarization

Consider a given source text $\mathbf{x} = (\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_n)$. The goal of summarization is to find a shorter text $\mathbf{y} = (\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_m)$ as the summary.

Previously, Schumann et al. [76] formulate summarization as word-level extraction (with order preserved), and apply edit-based discrete local search to maximize a heuristically designed objective.

Specifically, the objective function considers two aspects:

- A language fluency score $f_{\mathrm{LM}}(\mathbf{y})$, given by the reciprocal of language models'

perplexity:

$$f_{\text{LM}}(\mathbf{y}) = \frac{1}{\sqrt[2|\mathbf{y}|]{\prod_{i}^{|\mathbf{y}|} \frac{1}{p_{\overrightarrow{\text{LM}}}(\text{y}_i \mid \mathbf{y} < i)} \prod_{i}^{|\mathbf{y}|} \frac{1}{p_{\overleftarrow{\text{LM}}}(\text{y}_i \mid \mathbf{y} > i)}}} \tag{3.1}$$

where $\overrightarrow{\text{LM}}$ and $\overleftarrow{\text{LM}}$ are forward and backward LSTM [28] language models; and

- A semantic similarity score $f_{\text{SIM}}(\mathbf{y}; \mathbf{x})$, given by the cosine embeddings:

$$\cos(\boldsymbol{e}(\mathbf{x}), \boldsymbol{e}(\mathbf{y})) \tag{3.2}$$

where $\boldsymbol{e}(\cdot)$ calculates the sentence embedding by averaging the word embeddings learned by a sent2vec model [66].

The overall objective combines the two aspects as

$$f(\mathbf{y}; \mathbf{x}) = f_{\text{LM}}(\mathbf{y}) \cdot f_{\text{SIM}}(\mathbf{y}; \mathbf{x})^{\gamma} \tag{3.3}$$

where $\gamma$ is a weighting hyperparameter.

Further, the desired summary length can be specified as a hard constraint, achieved by searching only among sentences of the correct length. Suppose the desired summary length is $T$, the approach selects $T$ random words from the input, and maximizes the scoring function (3.3) by changing the selection and non-selection of two words.

A greedy hill-climbing algorithm determines whether the change is accepted or not. In other words, a change is accepted if the score improves, or rejected otherwise. Such a process continues until a (possibly local) optimum is found.

A pilot analysis in [74, 76] shows that words largely overlap between a source text and its reference summary. This explains the high performance of such a word-extraction approach, being a state-of-the-art unsupervised summarization system and outperforming strong competitors, e.g., cycle consistency [90, 4].

### 3.2.2 Non-Autoregressive Model for Summarization

Despite the high performance, such edit-based search has several drawbacks. First, the search process is slow because hundreds of local search steps are needed to obtain

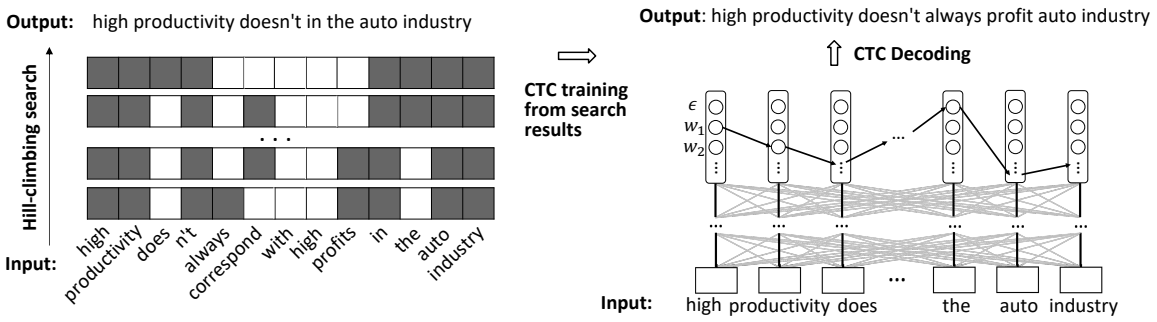


Figure 3.1: The overview of our approach. In each search step, input words corresponding to grey cells are selected.

a high-quality summary. Second, their approach only extracts the original words with order preserved. Therefore, the generated summary is restricted and may be noisy.

To this end, we train the non-autoregressive model with generated summaries of the search-based method [76]. In this way, the machine learning model can smooth out the search noise and is much faster, largely alleviating the drawbacks of search-based summarization. Compared with training an autoregressive model from search results [41], non-autoregressive generation predicts all the words in parallel, further improving inference efficiency by several times.

Moreover, a non-autoregressive model enables us to design an encoder-only architecture. It is more suited to the summarization task due to the strong correspondence between input and output, which cannot be fully utilized by encoder–decoder models, especially autoregressive ones.

Specifically, we propose to use the multi-layer Transformer [88] as the non-autoregressive architecture for summarization. Each Transformer layer is composed of a multi-head attention sublayer and a feed-forward sublayer. Additionally, there is a residual connection in each sublayer, followed by layer normalization.

Let $X^{(n)} \in \mathbb{R}^{T \times d}$ be the representation at the $n$th layer, where $T$ is the number of words and $d$ is the dimension. Specially, the input layer $X^{(0)}$ is the embeddings of words. Suppose we have $h$ attention heads. The output of the $i$th head in the $n$th attention sublayer is $A_i^{(n)} = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}}\right) V_i$, where $Q_i$, $K_i$, and $V_i$ are matrices

calculated by three distinct multi-layer perceptrons (MLPs) from $X^{(n-1)}$; $d_k$ is the attention dimension.

Multiple attention heads are then concatenated:

$$A^{(n)} = \text{Concat}\left(A_1^{(n)}, \ldots, A_h^{(n)}\right) W_O$$

where $W_O \in \mathbb{R}^{d \times d}$ is a weight matrix.

Then, we have a residual connection and layer normalization by

$$\bar{A}^{(n)} = \text{LayerNorm}\left(X^{(n-1)} + A^{(n)}\right) \tag{3.4}$$

Further, an MLP sublayer processes $\bar{A}^{(n)}$, followed by residual connection and layer normalization, yielding the $n$th layer's representation

$$X^{(n)} = \text{LayerNorm}\left(\bar{A}^{(n)} + \text{MLP}(\bar{A}^{(n)})\right) \tag{3.5}$$

The last Transformer layer $X^{(N)}$ is fed to softmax to predict the words of the summary in a non-autoregressive manner, that is, the probability at the $t$th step is given by $\text{softmax}(W\boldsymbol{x}_t^{(N)})$, where $\boldsymbol{x}_t^{(N)}$ is the $t$th row of the matrix $X^{(N)}$ and $W$ is the weight matrix.

It is emphasized that, in the vocabulary, we include a special blank token $\epsilon$, which is handled by dynamic programming during both training and inference. This enables us to generate a shorter summary than the input with such a multi-layer Transformer.

Our model can be thought of as an encoder-only architecture, differing from a typical encoder–decoder model with cross attention [88, 4, 107]. Previously, Su et al. [80] propose a seemingly similar model to us, but put multiple end-of-sequence (EOS) tokens at the end of the generation; thus, they are unable to maintain the correspondence between input and output. Instead, we allow blank tokens scattering over the entire sentence; the residual connections in Eqns (3.4) and (3.5) can better utilize such input–output correspondence for summarization.

### 3.2.3 Model Training

Our model is trained with the Connectionist Temporal Classification [CTC, 23] algorithm. CTC allows a special blank token $\epsilon$ in the vocabulary, and uses dynamic programming to marginalize out such blank tokens, known as *latent alignment* [73]. In addition, non-autoregressive generation suffers from a common problem that words may be repeated in consecutive steps [25, 39]; thus, CTC merges repeated words unless separated by $\epsilon$. For example, the sequence of tokens $a\epsilon\epsilon aabb\epsilon$ is reduced to the text $aab$, denoted by $\Gamma(a\epsilon\epsilon aabb\epsilon) = aab$.

Concretely, the predicted likelihood is marginalized over all possible fillings of $\epsilon$, i.e., all possible token sequences that are reduced to the groundtruth text:

$$P(\mathbf{y}|\mathbf{x}) = \sum\nolimits_{\mathbf{w}:\Gamma(\mathbf{w})=\mathbf{y}} P(\mathbf{w}|\mathbf{x}) \tag{3.6}$$

where $P(\mathbf{w}|\mathbf{x})$ is the probability of generating a sequence of tokens $\mathbf{w}$. Although enumerating every candidate in $\{\mathbf{w} : \Gamma(\mathbf{w}) = \mathbf{y}\}$ is intractable, such marginalization fortunately can be computed by dynamic programming in an efficient way.

Let $\alpha_{s,t} = \sum_{\mathbf{w}_{1:s}:\Gamma(\mathbf{w}_{1:s})=\mathbf{y}_{1:t}} P(\mathbf{w}_{1:s}|\mathbf{x})$ be the marginal probability of generating $\mathbf{y}_{1:t}$ up to the $s$th decoding slot. Moreover, $\alpha_{s,0}$ is defined to be the probability that $\mathbf{w}_{1:s}$ is all $\epsilon$, not match any word in $\mathbf{y}$. The $\alpha_{s,t}$ variable can be further decomposed into two terms $\alpha_{s,t} = \alpha^{\epsilon}_{s,t} + \alpha^{\neg\epsilon}_{s,t}$, where the first term is such probability with $\mathrm{w}_s = \epsilon$, and the second term $\mathrm{w}_s \neq \epsilon$. Apparently, the initialization of $\alpha$ variables is

$$\alpha^{\epsilon}_{1,0} = P(\mathrm{w}_1 = \epsilon|\mathbf{x}) \tag{3.7}$$

$$\alpha^{\neg\epsilon}_{1,1} = P(\mathrm{w}_1 = \mathrm{y}_1|\mathbf{x}) \tag{3.8}$$

$$\alpha^{\epsilon}_{1,t} = 0, \forall t \geq 1 \tag{3.9}$$

$$\alpha^{\neg\epsilon}_{1,t} = 0, \forall t > 1 \text{ or } t = 0 \tag{3.10}$$

Eqn. (3.9) is because, at the first prediction slot, the empty token $\epsilon$ does not match any target words; Eqn. (3.10) is because the predicted non-$\epsilon$ first token must match exactly the first target word.

The recursion formula for $\alpha^{\epsilon}_{s,t}$ is

$$\alpha^{\epsilon}_{s,t} = \alpha_{s-1,t} P(\mathrm{w}_t = \epsilon | \mathbf{x})$$

since the newly predicted token $\epsilon$ with probability $P(\mathrm{w}_t = \epsilon|\mathbf{x})$ does not match any target word, inheriting $\alpha_{s-1,t}$.

The recursion formula for $\alpha^{\neg\epsilon}_{s,t}$ is

$$\alpha^{\neg\epsilon}_{s,t} = \begin{cases} \left(\alpha^{\epsilon}_{s-1,t-1} + \alpha^{\neg\epsilon}_{s-1,t}\right) P(\mathrm{w}_s = \mathrm{y}_t|\mathbf{x}), & \text{if } \mathrm{y}_t = \mathrm{y}_{t-1} \\ \left(\alpha_{s-1,t-1} + \alpha^{\neg\epsilon}_{s-1,t}\right) P(\mathrm{w}_s = \mathrm{y}_t|\mathbf{x}), & \text{otherwise} \end{cases}$$

Here, $\mathrm{w}_s$ is not $\epsilon$, so we must have $\mathrm{w}_s = \mathrm{y}_t$, having the predicted probability $P(\mathrm{w}_s = \mathrm{y}_t|\mathbf{x})$.

If $\mathrm{y}_t = \mathrm{y}_{t-1}$, then we have two sub-cases: first, $\mathbf{w}_{1:s-1}$ is reduced to $\mathbf{y}_{1:t-1}$ with $\mathrm{w}_{s-1} = \epsilon$ separating two repeating words in $\mathbf{y}$, having probability $\alpha^{\epsilon}_{s-1,t-1}$; or second, $\mathbf{w}_{1:s-1}$ is reduced to $\mathbf{y}_{1:t}$ with $\mathrm{w}_{s-1} = \mathrm{y}_t \neq \epsilon$, having probability $\alpha^{\neg\epsilon}_{s-1}$, which implies we are merging $\mathrm{w}_{s-1}$ and $\mathrm{w}_s$.

If $\mathrm{y}_t \neq \mathrm{y}_{t-1}$, $\mathbf{w}_{1:s-1}$ is reduced to either $\mathbf{y}_{1:t-1}$ or $\mathbf{y}_{1:t}$. In the first case, $\mathrm{w}_{s-1}$ can be either $\epsilon$ or non-$\epsilon$, given by $\alpha_{s-1,t-1} = \alpha^{\epsilon}_{s-1,t-1} + \alpha^{\neg\epsilon}_{s-1,t-1}$. In the second case, we must have $\mathrm{w}_{s-1} \neq \epsilon$, which has a probability of $\alpha^{\neg\epsilon}_{s-1,t}$.

Finally, $\alpha_{|\mathbf{w}|,|\mathbf{y}|}$ is the marginal probability in Eqn. (3.6), as it is the probability that the entire generated sequence matches the entire target text.

The CTC maximum likelihood estimation is to maximize the marginal probability, which is equivalent to minimizing the loss $-\alpha_{|\mathbf{w}|,|\mathbf{y}|}$. Since the dynamic programming formulas are differentiable, the entire model can be trained by backpropagation in an end-to-end manner with auto-differentiation tools. We use PyTorch[1] in our implementation.

[1] https://www.pytorch.org/

## 3.3 Experiments

### 3.3.1 Setup

**Datasets.** We evaluated our non-autoregressive model on Gigaword headline generation and DUC2004, which are benchmark datasets for summarization.

The headline generation dataset [72] is constructed from the Gigaword news corpus [22], where the first sentence of a news article is considered as input text and the news title is considered as the summary. The dataset contains 3.8M, 198K, and 1951 samples for training, validation, and test, respectively. Based on the analysis of the training size in §3.3.2, we used 3M samples for model training.

It should be emphasized that, when our model learns from search, we only use the input of the training corpus: we perform search [76] for each input, and train our model from the search results. Therefore, we do not utilize any labeled parallel data, and our approach is unsupervised.

Moreover, we trained two variants of our model with search results of 8 and 10 words, respectively, which are the lengths that Schumann et al. [76] consider for the Gigaword test set.

The DUC2004 dataset [65] is designed for testing only and contains 500 samples, where we also took the first sentence of an article as the input text. Our model was transferred from the above headline generation corpus. Since Schumann et al. [76] evaluate 13-word summaries on DUC2004, we followed their setting and trained an additional model from search results of the same length.

**Evaluation Metrics.** We evaluated the quality of predicted summaries by ROUGE scores[2] [43], which are the most widely used metrics in previous work [90, 4, 107]. Specifically, ROUGE-$n$ evaluates $n$-gram overlap between a predicted summary and its reference summary; ROUGE-L, instead, measures the longest common sequence between the predicted and reference summaries.

[2]https://github.com/tagucci/pythonrouge

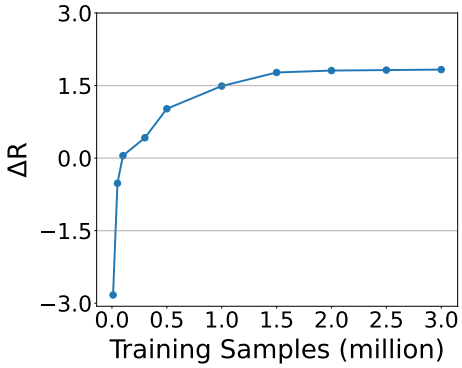


Figure 3.2: Performance versus the number of training samples when learning from 10-word search results. Notice that our model is trained by pseudo-groundtruth given by unsupervised edit-based search [76]. Thus, our approach is indeed unsupervised.

Different ROUGE variants are adopted in previous work, depending on the dataset. We followed the standard evaluation scripts and evaluated headline generation by ROUGE F1 [90, 4, 76] and DUC2004 by Truncate ROUGE Recall [11, 91].

In addition to summary quality, we also evaluated the inference efficiency of different methods, as it is important for the deployment of deep learning models in real-time applications. We report the average inference time in seconds for each data sample, and compare the speedup with Schumann et al. [76]'s search approach, which achieves (previous) state-of-the-art ROUGE scores in the unsupervised setting.

### 3.3.2 Implementation Details

Our non-autoregresive model has a Transformer encoder as the basic structure, generally following the settings in [88]: 6 encoder layers, each having 8 attention heads. The dimension was 512 for attention and 2048 for feed-forward modules.

Our training used a batch size of 4K tokens, with a maximum of 200K updates. We used Adam with $\beta = (0.9, 0.98)$. In general, the learning rate warmed up to 5e-4 in the first 10K steps, and then decayed to 1e-9 with the inverse square-root schedule, except that we find the maximum learning rate of 1e-4 worked better for headline generation with the summary length of 8. We set the $\ell_2$ weight decay to 0.01.

The training of our model is based on Schumann et al. [76]'s prediction on the input

| # | Approach | | | ROUGE F1 | | | | Inf.Time | Speedup |
|---|---|---|---|---|---|---|---|---|---|
| | | | | R-1 | R-2 | R-L | ΔR | | |
| 1 | Baseline | | Baziotis et al. [4]† | 25.39 | 8.21 | 22.68 | -6.19 | – | – |
| 2 | | | Wang and Lee [90]† | 27.29 | 10.01 | 24.59 | -0.58 | – | – |
| 3 | | | Zhou and Rush [107]† | 26.48 | 10.05 | 24.41 | -1.53 | – | – |
| 4 | 8 words | Search | Schumann et al. [76]† | 26.32 | 9.63 | 24.19 | -2.26 | – | – |
| 5 | | | Our replication | 26.17 | 9.69 | 24.10 | -2.42 | 6.846 | 1.346x |
| 6 | | Learn from search | Su et al. [80] | 26.95 | 9.56 | 24.85 | -1.11 | 0.017 | 542x |
| 7 | | | Ours | 28.72 | 10.02 | 25.80 | 2.07 | **0.005** | **1843x** |
| 8 | 10 words | Search | Schumann et al. [76]† | 27.52 | 10.27 | 24.91 | 0.23 | – | – |
| 9 | | | Our replication | 27.35 | **10.25** | 24.87 | 0 | 9.217 | 1x |
| 10 | | Learn from search | Su et al. [80] | 28.01 | 9.92 | 25.57 | 1.03 | 0.020 | 461x |
| 11 | | | Ours | **29.14** | 10.23 | **25.84** | **2.75** | **0.005** | **1843x** |

Table 3.1: Results on the Gigaword headline generation test set. **R-1, R-2, R-L:** ROUGE-1, ROUGE-2, ROUGE-L. **ΔR:** The difference of total ROUGE (sum of R-1, R-2, and R-L) in comparison with the (previous) state-of-the-art search method under replication. **Inf.Time:** Average inference time in seconds for one sample on an i9-9940X CPU and a RTX6000 GPU. **Speedup:** Relative to [76]. †Results quoted from previous papers; others are given by our experiments.[3]

of the Gigaword headline generation training set. We show performance against the number of training samples in Figure 3.2. As seen, our model outperforms its search teacher even with a small set of 0.1 million samples, and the performance saturates as the number of samples increases. Based on this analysis, we used 3 million samples from the 3.8 million Gigaword training set to train our models.

### 3.3.3 Results

**Main Results.** Table 3.1 presents the performance of our model and baselines on the Gigaword headline test set.

Both Wang and Lee [90] and Baziotis et al. [4] utilize cycle consistency [57] for unsupervised summarization; the performance is relatively low, because the cycle consistency loss cannot ensure the generated text is a valid summary. Zhou and

[3]Su et al. [80] involve a parameter to control the penalty on summary length, we set it to 1 for results in Table 3.1, i.e., no length penalty.

Rush [107] perform beam search towards a step-by-step decomposable score of fluency and contextual matching, achieving higher scores than the cycle-consistency methods. All of these methods have worse performance than the 10-word summaries generated by [76][4], which performs edit-based local search and yields the (previous) state-of-the-art performance.

Our approach follows [76], but trains a non-autoregressive model from search results. We consider training summaries of two different lengths, i.e., 8 and 10 words. In both settings, our approach outperforms its search-based teacher [76] by 2.07–2.75 points in terms of the total ROUGE score (Rows 7 & 11, Table 3.1). As mentioned, Schumann et al. [76] only extract original words with order preserved, yielding noisy sentences. Our model, as a student, learns from the search-based teacher model and is able to smooth out its noise. This is a compelling result, as our student model outperforms its teacher.

Regarding inference efficiency, our method does not need iterative search and is thus more than 1800 times faster than the 10-word search-based method [76]. This shows our approach is extremely efficient in inference, which is important for real-time applications.

Although the efficiency of [4], [90] and [107] is not available, we still expect our approach to be a few times faster (in addition to our higher ROUGE scores) because their models are autoregressive. By contrast, our approach is non-autoregressive, meaning that it predicts all words simultaneously. We will provide a controlled comparison between autoregressive and non-autoregressive models in §4.3.

Table 3.2 shows the results on the DUC2004 dataset. The cycle-consistency approach [4, 91] does not perform well on this dataset, outperformed by an early rule-based syntax tree trimming approach [103] and the state-of-the-art edit-based search [76].

[4]Schumann et al. [76] present a few variants that use additional datasets for training language models (in an unsupervised way). In our study, we focus on the setting without data augmentation, i.e., the language model is trained on non-parallel the Gigawords corpus.

| Model | ROUGE Recall | | | | Time | Speedup |
|---|---|---|---|---|---|---|
| | R-1 | R-2 | R-L | ΔR | | |
| Zajic et al. [103]† | 25.12 | 6.46 | 20.12 | -5.35 | – | – |
| Baziotis et al. [4]† | 22.13 | 6.18 | 19.30 | -9.44 | – | – |
| West et al. [91]† | 22.85 | 5.71 | 19.87 | -8.62 | – | – |
| Schumann et al. [76]† | 26.04 | **8.06** | 22.90 | -0.05 | – | – |
| Our replication | 26.14 | 8.03 | 22.88 | 0 | 12.314 | 1x |
| Su et al. [80] | 26.34 | 7.71 | 22.85 | -0.10 | 0.022 | 559x |
| Ours | **26.68** | 7.75 | **23.04** | **0.47** | 0.005 | 2463x |

Table 3.2: Results on the DUC2004 dataset. †Quoted from previous papers.

The performance of our model is consistent with Table 4.2, outperforming all previous methods in terms of the total ROUGE score, and being over 2000 times faster than the search approach [76].

In general, the proposed model not only achieves state-of-the-art ROUGE scores for unsupervised summarization, but also is more efficient when deployed. Results are consistent on both datasets, demonstrating the generality of our approach.

# Chapter 4

# Summarization with Word-Level Length Control

## 4.1 Overview

Controlling output length is the nature of the summarization task, for example, displaying a short news headline on a mobile device. Moreover, Schumann et al. [76] show that the main evaluation metric ROUGE [43] is sensitive to the summary length, and longer summaries tend to achieve higher ROUGE scores. Thus, it is crucial to control the summary length for fair comparison.

In this chapter, we present a word-level length-control algorithm, which is able to generate a summary of the desired number of words based on output probabilities of the non-autoregressive model. Moreover, we show the experimental performance of our approach when there is a length budget. We further conduct ablation studies and examine the effectiveness of different model components.

## 4.2 Methodology

### 4.2.1 The Proposed Algorithm

Our word-level length-control algorithm follows the nature of CTC training and is based on dynamic programming (DP), dividing the length control into shared sub-problems; this is feasible because our non-autoregressive model predicts probabilities

independently.

However, our DP is an approximate algorithm because of the dependencies introduced by removing consecutive repeated tokens (§3.2.3). Thus, we equip our DP with a beam search mechanism.

We define $\mathscr{B}$ to be a DP table, where $\mathscr{B}_{s,t}$ is a set of top-$B$ sequences with $s$ predicted tokens that are reduced to $t$ words.

The initialization of $\mathscr{B}_{s,t}$ fills in the DP table for $t = 0$ and $s = 1$.

- For $t = 0$, we must have $\mathscr{B}_{s,0} = \{\underbrace{\epsilon \cdots \epsilon}_{s\text{-many}}\}$, because $t = 0$ means no non-$\epsilon$ word has been generated.
- For $s = 1$, we have

$$\mathscr{B}_{1,t} = \begin{cases} \{\epsilon\}, & \text{if } \ t = 0 \\ \text{top}_B\{\text{w} : \text{w}_1 \neq \epsilon\}, & \text{if } \ t = 1 \end{cases} \tag{4.1}$$

  where $\text{top}_B$ selects the best $B$ elements by the probability $P(\text{w}_s|\mathbf{x})$. Here, $t = 0$ is the same as the previous bullet item. If $t = 1$, the summary has a length of 1 and thus the only token in the summary, i.e., the selected token at the first slot, must be non-$\epsilon$. In this case, we select the top-$B$ probable words according to the value $P(\text{w}_1)$, i.e., the predicted probability of the first generation slot.

Then, the recursion of $\mathscr{B}_{s,t}$ can be categorized into three scenarios.

- First, the blank token $\epsilon$ is predicted for the $s$th generation slot, and thus the summary length $t$ remains the same, shown by the blue arrow in Figure 4.1. This yields a set of candidates

$$\mathscr{B}^{(1)}_{s,t} = \left\{\mathbf{b} \oplus \epsilon \ : \ \mathbf{b} \in \mathscr{B}_{s-1,t}\right\} \tag{4.2}$$

  where $\oplus$ refers to string/token concatenation.

- Second, a repeated word is predicted for the $s$th generation slot, i.e., $\text{b}_{s-1}$ for a subsequence $\mathbf{b}$ of length $s - 1$. In this case, the summary length $t$ also remains the same, also shown by the blue arrow in Figure 4.1. This gives a candidate set

$$\mathscr{B}^{(2)}_{s,t} = \left\{\mathbf{b} \oplus \text{b}_{s-1} \ : \ \mathbf{b} \in \mathscr{B}_{s-1,t}\right\} \tag{4.3}$$

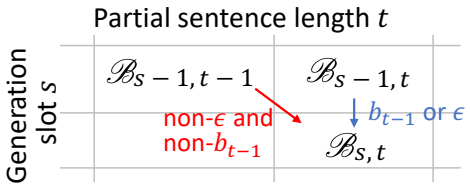


Figure 4.1: Illustration of our length-control algorithm.

- Third, a non-$\epsilon$, non-repeating word $\mathrm{w}_s$ is generated, increasing the summary length from $t-1$ to $t$, shown by the red arrow in Figure 4.1. This gives

$$\mathscr{B}^{(3)}_{s,t} = \mathrm{top}_B \left\{ \mathbf{b} \oplus \mathrm{w} \ : \mathbf{b} \in \mathscr{B}_{s-1,t-1}, \mathrm{w}_s \neq \epsilon, \mathrm{w}_s \neq \mathrm{b}_{s-1} \right\} \tag{4.4}$$

Based on the three candidates sets, we select top-$B$ sequences to keep the beam size fixed:

$$\mathscr{B}_{s,t} = \mathrm{top}_B(\mathscr{B}^{(1)}_{s,t} \cup \mathscr{B}^{(2)}_{s,t} \cup \mathscr{B}^{(3)}_{s,t}) \tag{4.5}$$

where the sequences are ranked by their predicted joint probabilities.

### 4.2.2 Theoretical Analysis

We perform a theoretical analysis on the exactness of our DP algorithm.

**Theorem 1.** *(1) If repeating tokens are not merged, then the proposed length-control algorithm with beam size $B = 1$ finds the exact optimum $\mathscr{B}_{S,T}$ being the most probable length-$T$ sentence given by $S$ prediction slots. (2) If we merge repeating tokens predicted by CTC-trained models, the above algorithm may not be exact.*

*Proof.* [Part (1)] This part concerns a variant of our decoding algorithm, which only removes the blank token $\epsilon$ but does not merge consecutive repeated tokens to a single word, i.e., Eqn. (5.5) is removed. We denote this by $\Gamma'$, for example, $\Gamma'(a\epsilon\epsilon aabb\epsilon) = aaabb$, as opposed to $\Gamma(a\epsilon\epsilon aabb\epsilon) = aab$ in our algorithm. We now show that, based on $\Gamma'$, our dynamic programming algorithm in §4.2 with beam size $B = 1$ is an exact inference algorithm.

We define $\beta_{s,t} = \max_{\mathbf{b}:|\mathbf{b}|=s,|\Gamma'(\mathbf{b})|=t} P(\mathbf{b}|\mathbf{x})$, where $|\cdot|$ denotes the length of a sequence. In other words, $\beta_{s,t}$ is the maximum probability of $s$ tokens that are reduced to $t$ words.

According to the definition, we have

$$\beta_{1,0} = P\left(\mathrm{w}_1 = \epsilon|\mathbf{x}\right) \tag{4.6}$$

$$\beta_{1,1} = \max_{\mathrm{w}_1 \neq \epsilon} P\left(\mathrm{w}_1|\mathbf{x}\right) \tag{4.7}$$

$$\beta_{s,t} = 0 \quad \text{for } s > t \tag{4.8}$$

In Eqn. (4.6), $\beta_{1,0}$ refers to the probability of one token that is reduced to zero words. In this case, the first predicted token can only be the blank token $\epsilon$, which corresponds to Eqn. (4.2) with $s = 1$ and $t = 0$. Likewise, $\beta_{1,1}$ is the maximum probability of one token that is reduced to one word. Thus, it is the probability of the most probable non-$\epsilon$ token, corresponding to Eqn. (4.4) with $s = 1$ and $t = 0$. Eqn. (4.8) asserts that fewer tokens cannot be reduced to more words; it is used for mathematical derivations, but need not to be explicitly implemented in our algorithm in §4.2.

The recursion variable $\beta_{s,t}$ is computed by

$$\beta_{s,t} = \max\left\{\beta_{s-1,t} \cdot P(\mathrm{w}_s = \epsilon|\mathbf{x}),\ \beta_{s-1,t-1} \cdot \max_{\mathrm{w}_s \neq \epsilon} P(\mathrm{w}_s|\mathbf{x})\right\} \tag{4.9}$$

In other words, the variable $\beta_{s,t}$ can inherit $\beta_{s-1,t}$ with a predicted blank token $\epsilon$, corresponding to Eqn. (4.2); or it can inherit $\beta_{s-1,t-1}$ with a predicted non-$\epsilon$ token, corresponding to Eqn. (4.4). Specially, if $t = 0$, then the second term has $\beta_{s-1,-1}$ undefined, and thus is ignored in the max operation.

We need the max operator to take the higher probability in the two cases, since $\beta_{s,t}$ is the maximum probability of $s$ tokens being reduced to $t$ words. This corresponds to Eqn. (4.5) with beam size $B = 1$.

To sum up, our inductive calculation guarantees that $\beta_{S,T}$ is the exact maximum probability of $\max_{\mathbf{b}:|\mathbf{b}|=S,|\Gamma'(\mathbf{b})|=T} P(\mathbf{b}|\mathbf{x})$ for the desired length $T$ with $S$ generation slots; our algorithm (if not merging repeating tokens) gives the corresponding $\mathscr{B}_{S,T}$ as $\operatorname{argmax} P(\mathbf{b}|\mathbf{x})$ under the same constraints, concluding the proof of Part (1).

| Word | $P(\mathrm{w}_1\|\mathbf{x})$ | $P(\mathrm{w}_2\|\mathbf{x})$ |
|---|---|---|
| I | 0.39 | 0.1 |
| like | 0.4 | 0.9 |
| coding | 0.1 | 0 |
| $\epsilon$ | 0.11 | 0 |

Table 4.1: An example of predicted probabilities of two generation slots, where we have a vocabulary of three words and a blank token $\epsilon$.

[Part (2)] CTC training [23] merges consecutive repeated tokens to a single word, unless separated by the blank token $\epsilon$. Since our model is trained by CTC, we should adopt this rule in inference as well. We show in this part that our algorithm, with beam size $B = 1$, may not yield the exact optimum with an example in Table 4.1.

We consider generating a sentence of two words from the two prediction slots, i.e., $S = T = 2$. Apparently, the optimal sequence is "I like" with probability $0.39 \cdot 0.9 = 0.351$. However, the algorithm would predict $\mathscr{B}_{1,1} = \{\text{"like"}\}$ because "like" is the most probably token in the first slot. Then, our algorithm will give $\mathscr{B}_{2,2} = \{\text{"like I"}\}$, because it has to select a non-repeating token based on $\Gamma$, yielding a non-optimal solution.

□

It is noted that, if we do not merge repeating tokens as in $\Gamma'$, our algorithm will give the exact optimum "like like" in the above example. This shows that merging consecutive repeated tokens requires the decoding algorithm to correct early predictions, and thus, our dynamic programming becomes an approximate inference. Nevertheless, our algorithm is able to generate a sequence of the desired length properly; its approximation happens only when the algorithm compares more repetitions with fewer $\epsilon$s versus more $\epsilon$s with fewer repetitions. Such approximation is further alleviated by beam search in our dynamic programming. Therefore, the proposed length-control algorithm is better than truncating a longer sentence; especially, our approach generates more fluent and complete sentences.

## 4.3 Experiments

### 4.3.1 Setup

Following §3.3, we adopted ROUGE scores as the evaluation metric and evaluated our word-level length-control algorithm on the Gigaword headline generation and DUC2004 datasets.

Moreover, our training parameters were the same as §3.3.2, and we additionally set the beam size to be 6 based on the analysis in §4.3.4.

### 4.3.2 Results and Analyses

**Main Results.** Table 4.2 presents the performance of our model and baselines on the Gigaword headline test set. For a fair comparison, we categorize all approaches by average summary lengths of ∼8 and ∼10 into Groups A and B, respectively, following [76].

The Lead baseline extracts the first several words of the input sentence and is thus a suitable baseline for our settings where the number of words in summaries is controlled. Despite its simplicity, the Lead approach is a strong summarization baseline adopted in most previous work [18, 4].

Both Wang and Lee [90] and Zhou and Rush [107] are unable to explicitly control the summary length: in a fair comparison of length 10 (Group B, Table 4.2), their performance is worse than the search-based approach [76].

We control the summary length of our model with two different methods: truncating longer summaries and decoding with our proposed length-control algorithm. Notice that this setting is different from §3.3, where the summary length is not constrained. Both of the two variants outperform [76] by 1.21–2.73 points in terms of the total ROUGE score (Rows 5–6 & 13–14, Table 4.2). This is consistent with the result in §3.3, confirming the effectiveness of our learning-from-search approach.

Moreover, our approach, even with dynamic programming and beam search for

| Group | # | Approach | | Len | ROUGE F1 | | | | Inf.Time | Speedup |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | R-1 | R-2 | R-L | ΔR | | |
| A (desired length 8) | 1 | Baseline | Lead (8 words)† | 7.9 | 21.39 | 7.42 | 20.03 | -11.12 | – | – |
| | 2 | Search | Schumann et al. [76]† | 7.9 | 26.32 | 9.63 | 24.19 | 0.18 | – | – |
| | 3 | | Our replication | 7.9 | 26.17 | **9.69** | 24.10 | 0 | 6.846 | 1x |
| | 4 | Learn from search | Su et al. [80] | 7.7 | 26.88 | 9.37 | 24.54 | 0.83 | 0.017 | 403x |
| | 5 | | Ours (truncate) | 7.8 | 27.27 | 9.49 | 24.96 | 1.76 | **0.005** | **1369x** |
| | 6 | | Ours (length control) | 7.8 | **27.94** | 9.24 | **25.51** | **2.73** | 0.041 | 167x |
| B (desired length 10) | 7 | Baseline | Lead (10 words)† | 9.8 | 23.03 | 7.95 | 21.29 | -10.2 | – | – |
| | 8 | | Wang and Lee [90]† | 10.8 | 27.29 | 10.01 | 24.59 | -0.58 | – | – |
| | 9 | | Zhou and Rush [107]† | 9.3 | 26.48 | 10.05 | 24.41 | -1.53 | – | – |
| | 10 | Search | Schumann et al. [76]† | 9.8 | 27.52 | **10.27** | 24.91 | 0.23 | – | – |
| | 11 | | Our replication | 9.8 | 27.35 | 10.25 | 24.87 | 0 | 9.217 | 1x |
| | 12 | Learn from search | Su et al. [80] | 9.4 | 27.86 | 9.88 | 25.51 | 0.78 | 0.020 | 461x |
| | 13 | | Ours (truncate) | 9.8 | 28.24 | 10.04 | 25.40 | 1.21 | **0.005** | **1843x** |
| | 14 | | Ours (length control) | 9.8 | **28.55** | 9.97 | **25.78** | **1.83** | 0.044 | 210x |

Table 4.2: Results on the Gigaword headline generation test set. **Len:** Average length of predicted summaries. **R-1, R-2, R-L:** ROUGE-1, ROUGE-2, ROUGE-L. **ΔR:** The difference of total ROUGE (sum of R-1, R-2, and R-L) in comparison with the (previous) state-of-the-art search method under replication. **Inf.Time:** Average inference time in seconds for one sample on an i9-9940X CPU and a RTX6000 GPU. **Speedup:** Relative to [76]. †Results quoted from previous papers; others are given by our experiments.

| Model | ROUGE Recall | | | | Time | Speedup |
|---|---|---|---|---|---|---|
| | R-1 | R-2 | R-L | ΔR | | |
| Lead (75 characters)† | 22.50 | 6.49 | 19.72 | -8.34 | – | – |
| Zajic et al. [103]† | 25.12 | 6.46 | 20.12 | -5.35 | – | – |
| Baziotis et al. [4]† | 22.13 | 6.18 | 19.30 | -9.44 | – | – |
| West et al. [91]† | 22.85 | 5.71 | 19.87 | -8.62 | – | – |
| Schumann et al. [76]† | 26.04 | **8.06** | 22.90 | -0.05 | – | – |
| Our replication | 26.14 | 8.03 | 22.88 | 0 | 12.314 | 1x |
| Su et al. [80] | 26.25 | 7.66 | 22.83 | -0.31 | 0.022 | 559x |
| Ours (truncate) | 26.52 | 7.88 | 22.91 | 0.26 | **0.005** | **2463x** |
| Ours (length control) | **26.71** | 7.68 | **23.06** | **0.40** | 0.048 | 257x |

Table 4.3: Results on the DUC2004 dataset. †Quoted from previous papers.

length control, is still over 100 times faster than its search-based teacher. This high efficiency of our approach is crucial for the deployment to real-world applications.

We show the results on the DUC2004 dataset in Table 4.3. As seen, our model

outperforms all previous methods in terms of the total ROUGE score, which is again consistent with previous results. Besides, our method is 100–2000 times faster than the search approach [76].

### 4.3.3 Comparison with Autoregressive and Encoder–Decoder Models

We conduct in-depth analyses on the proposed model in Table 4.4. Due to the limit of time and computational resources, we chose the Gigaword headline generation as our testbed. All the autoregressive (AR) and non-autoregressive (NAR) variants learn from the search output of our replication (Rows 2 & 11), where we achieve very close results to those reported in [76].

We first tried vanilla encoder–decoder NAR Transformer (Rows 4 & 13) [25], where we set the number of decoding slots to be the desired summary length; thus, the blank token and the length-control algorithm are not needed. As seen, a vanilla NAR model does not perform well, and CTC largely outperforms vanilla NAR in both groups (Rows 5–6 & 14–15). Such results are highly consistent with the translation literature [73, 6, 24, 69, 29].

Our proposed encoder-only model outperforms encoder–decoder ones in both groups in terms of the total ROUGE score, when the summary length is controlled by either truncating or length-control decoding (Rows 8–9 & 17–18). Profoundly, our non-autoregressive model is even better than the autoregressive Transformer (Rows 3 & 12). We also experimented with the previous supervised non-autoregressive summarization model [80] in our unsupervised learning-from-search setting. Although such an approach appears to be encoder-only, it adds end-of-sequence (EOS) tokens at the end of the generation, and thus is unable to utilize the input–output correspondence. Their performance is higher than vanilla NAR models, but lower than ours. By contrast, our model is able to capture such correspondence with the residual connections, i.e., Eqns. (3.4) and (3.5), in its encoder-only architecture.

| # | Approach | | ROUGE Recall | | | | Speedup |
|---|---|---|---|---|---|---|---|
| | | | R-1 | R-2 | R-L | ΔR | |
| Group A (desired length 8) | | | | | | | |
| 1 | Search | Schumann et al. [76] | 26.32 | 9.63 | 24.19 | 0.18 | – |
| 2 | | Our replication | 26.17 | **9.69** | 24.10 | 0 | 1x |
| 3 | AR | Transformer (truncate) | 26.65 | 9.51 | 24.67 | 0.87 | 58x |
| 4 | NAR encoder–decoder | Vanilla | 24.87 | 8.33 | 22.74 | -4.02 | 571x |
| 5 | | CTC (truncate) | 27.30 | 9.20 | 24.96 | 1.5 | 571x |
| 6 | | CTC (length control) | 27.76 | 9.13 | 25.33 | 2.26 | 149x |
| 7 | NAR encoder–only | Su et al. [80] | 26.88 | 9.37 | 24.54 | 0.83 | 403x |
| 8 | | Ours (truncate) | 27.27 | 9.49 | 24.96 | 1.76 | **1396x** |
| 9 | | Ours (length control) | **27.94** | 9.24 | **25.51** | 2.73 | 167x |
| Group B (desired length 10) | | | | | | | |
| 10 | Search | Schumann et al. [76] | 27.52 | **10.27** | 24.91 | 0.23 | – |
| 11 | | Our replication | 27.35 | 10.25 | 24.87 | 0 | 1x |
| 12 | AR | Transformer (truncate) | 27.06 | 9.63 | 24.55 | -1.23 | 66x |
| 13 | NAR encoder–decoder | Vanilla | 25.77 | 8.69 | 23.52 | -4.49 | 709x |
| 14 | | CTC (truncate) | 28.14 | 10.07 | 25.37 | 1.11 | 709x |
| 15 | | CTC (length control) | 28.45 | 9.81 | 25.63 | 1.42 | 192x |
| 16 | NAR encoder–only | Su et al. [80] | 27.86 | 9.88 | 25.51 | 0.78 | 461x |
| 17 | | Ours (truncate) | 28.24 | 10.04 | 25.40 | 1.21 | **1843x** |
| 18 | | Ours (length control) | **28.55** | 9.97 | **25.78** | **1.83** | 210x |

Table 4.4: Model analysis on the headline generation test set. All autoregressive (AR) and non-autoregressive (NAR) models use the Transformer architecture.

Generally, the efficiency of encoder-only NAR[1] (without length-control decoding) is ∼2 times faster than encoder–decoder NAR and ∼20 times faster than the AR Transformer.

Further, our length-control decoding improves the total ROUGE score, compared with truncating, for both encoder–decoder CTC and encoder-only models (Rows 6, 9, 15, & 18), although its dynamic programming is slower. Nevertheless, our non-autoregressive model with length control is ∼200 times faster than search and ∼3 times faster than the AR Transformer.

[1]The standard minimal encoder–decoder NAR model has 6 layers for the encoder and another 6 layers the decoder [88]. Our model only has a 6-layer encoder. Our pilot study shows that more layers do not further improve performance in our encoder-only architecture.

### 4.3.4 Analysis of Beam Search

As mentioned, our length-control decoding algorithm involves beam search within its dynamic programming, because the algorithm does not find the exact optimum when it merges repeating words. We analyze the effect of the beam size in our length-control algorithm.

In addition, we compare our approach with CTC beam search [23].[2] Typically, a CTC-trained non-autoregressive model can be decoded either greedily or by beam search. The greedy decoding finds the most probable token at each step, i.e., $\mathrm{w}_i^* = \mathrm{argmax}_{\mathrm{w}_i} P(\mathrm{w}_i|\mathbf{x})$, and reduces the tokens to a sentence by $\Gamma(\mathrm{w}_1, \cdots, \mathrm{w}_T)$, where $T$ is the number of decoding steps. The CTC beam search algorithm searches for the most likely sentence by marginalizing all token sequences that are reduced to $\mathbf{y}$, i.e., $\mathrm{argmax}_{\mathbf{y}} \sum_{\mathbf{w}:\Gamma(\mathbf{w})=\mathbf{y}} P(\mathbf{w}|\mathbf{x})$.

We show results in Figure 4.2, where we chose 10-word Gigaword headline generation as the testbed with our model (Group B, Table 4.2). Notice that CTC beam search does not control the output length, and for fair comparison, we truncated its generated summaries. This also shows that our novel decoding approach and CTC beam search are distinct algorithms.

As seen in Figure 4.2a, the beam search does play a role in our length-control algorithm. When the beam enlarges from 1 to 6, the performance (orange solid line) increases by 1.2 points in $\Delta$R, the difference of total ROUGE in comparison with [76] under our replication (Row 10, Table 4.2). However, further increasing the beam size does not yield an additional performance gain. This is consistent with previous literature in autoregressive generation [55], which also suggests a beam size of 5–7 is the best in their applications. In terms of the efficiency (Figure 4.2b), a larger beam size monotonically increases the inference time. However, the overhead of beam search is relatively small in our dynamic programming, and thus we chose a beam size of 6

[2] Our implementation of CTC beam search is based on https://github.com/parlance/ctcdecode

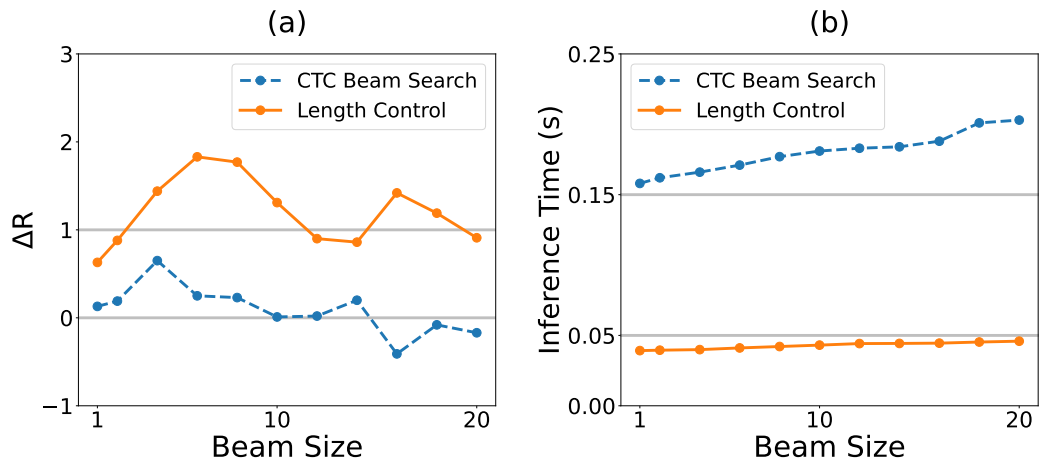


Figure 4.2: Comparing our length-control method and the truncated CTC beam search on the Gigaward headline generation test set.

in our experiments.

Our length-control algorithm significantly outperforms CTC beam search (dashed blue lines) in terms of both ΔR and efficiency. Especially, CTC beam search is three times slower, and degrades more significantly than our length-control decoding when the beam size increases.

### 4.3.5 Case Study

We show in Table 4.5 example summaries generated by our approach with truncating and length-control decoding, as well as the previous state-of-the-art method [76]. We observe that our method without length control generates slightly longer summaries, and if truncated, the output may be incomplete; by contrast, our length-control algorithm generates a fluent and complete sentence of the desired length by dynamic programming. Compared with [76], our method (length control) generates a more informative summary that includes the main clause (*united nations condemned*), which also appears in the reference summary.

| |
|---|
| **Input:** the united nations condemned saturday an attack on russian embassy employees in baghdad that claimed the life of one russian and resulted in the kidnapping of four others |
| **Reference:** un condemns murder of russians in iraq with annan comment |
| **Schumann et al. [76]:** attack on russian embassy in baghdad claimed one in four |
| **ours (truncate):** an attack on russian embassy employees in baghdad claimed in kidnapping of four others |
| **Ours (length control):** united nations condemned attack on russian embassy employees in baghdad |

Table 4.5: Example summaries for Gigaword headline generation. The gray words are truncated for fair comparison.

### 4.3.6 Length-Transfer Generation

In previous sections, we present results where our model is trained on search outputs [76] that have the same length as the inference target. This follows the common assumption in machine learning that training and test samples are independently identically distributed.

In this section, we show the performance of length-transfer summary generation, where the prediction has a different length from that of training. We denote such a model by $\text{Ours}_{i\to j}$, referring to training with $i$ words and testing for $j$ words.

As seen in Groups A & B in Table 4.6, our approach with length transfer is slightly worse than our model trained on the correct length, which is understandable. Nevertheless, length-transfer decoding still outperforms the search teacher and other baselines.

Moreover, we consider the third setting in [76], where the target length is 50% of the input. Since it takes time to obtain pseudo-groundtruths given by the edit-based search, we would directly transfer already trained models to this setting by our length-control decoding. Results are shown in Group C, Table 4.6. We observe $\text{Ours}_{10\to 50\%}$ is better than $\text{Ours}_{8\to 50\%}$, which makes much sense because the latter has a larger gap during transfer. Remarkably, both $\text{Ours}_{8\to 50\%}$ and $\text{Ours}_{10\to 50\%}$ outperform [76] and other baselines, achieving new state-of-the-art unsupervised performance on this setting as well.

| Group | # | Approach | | Len | ROUGE F1 | | | | Inf.Time | Speedup |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | R-1 | R-2 | R-L | ΔR | | |
| Group A (desired length 8) | 1 | Baseline | Lead (8 words)† | 7.9 | 21.39 | 7.42 | 20.03 | -11.12 | – | – |
| | 2 | Search | Schumann et al. [76]† | 7.9 | 26.32 | 9.63 | 24.19 | 0.18 | – | – |
| | 3 | | Our replication | 7.9 | 26.17 | **9.69** | 24.10 | 0 | 6.846 | 1x |
| | 4 | Learn from search | Su et al. $[80]_{8\to8}$ | 7.7 | 26.88 | 9.37 | 24.54 | 0.83 | 0.017 | 403x |
| | 5 | | Su et al. $[80]_{10\to8}$ | 8.4 | 25.71 | 8.94 | 23.65 | -1.84 | 0.018 | 380x |
| | 6 | | Ours (truncate) | 7.8 | 27.27 | 9.49 | 24.96 | 1.76 | **0.005** | **1369x** |
| | 7 | | $\text{Ours}_{8\to8}$ | 7.8 | **27.94** | 9.24 | **25.50** | **2.73** | 0.041 | 167x |
| | 8 | | $\text{Ours}_{10\to8}$ | 7.9 | 27.12 | 9.08 | 24.86 | 1.10 | | |
| Group B (desired length 10) | 9 | Baseline | Lead (10 words)† | 9.8 | 23.03 | 7.95 | 21.29 | -10.2 | – | – |
| | 10 | | Wang and Lee [90]† | 10.8 | 27.29 | 10.01 | 24.59 | -0.58 | – | – |
| | 11 | | Zhou and Rush [107]† | 9.3 | 26.48 | 10.05 | 24.41 | -1.53 | – | – |
| | 12 | Search | Schumann et al. [76]† | 9.8 | 27.52 | **10.27** | 24.91 | 0.23 | – | – |
| | 13 | | Our replication | 9.8 | 27.35 | 10.25 | 24.87 | 0 | 9.217 | 1x |
| | 14 | Learn from search | Su et al. $[80]_{8\to10}$ | – | – | – | – | – | – | – |
| | 15 | | Su et al. $[80]_{10\to10}$ | 9.4 | 27.86 | 9.88 | 25.51 | 0.78 | 0.020 | 461x |
| | 16 | | Ours (truncate) | 9.8 | 28.24 | 10.04 | 25.40 | 1.21 | **0.005** | **1843x** |
| | 17 | | $\text{Ours}_{8\to10}$ | 9.9 | 28.32 | 9.58 | 25.46 | 0.89 | 0.044 | 210x |
| | 18 | | $\text{Ours}_{10\to10}$ | 9.8 | **28.55** | 9.97 | **25.78** | **1.83** | | |
| Group C (desired length 50% of the input) | 19 | Baseline | Lead (50% words)† | 14.6 | 24.97 | 8.65 | 22.43 | -4.58 | – | – |
| | 20 | | Fevry and Phang [18]† | 14.8 | 23.16 | 5.93 | 20.11 | -11.43 | – | – |
| | 21 | | Baziotis et al. [4]† | 15.1 | 24.70 | 7.97 | 22.41 | -5.55 | – | – |
| | 22 | Search | Schumann et al. [76]† | 14.9 | 27.05 | 9.75 | 23.89 | 0.06 | – | – |
| | 23 | | Our replication | 14.9 | 27.03 | 9.81 | 23.79 | 0 | 17.462 | 1x |
| | 24 | Learn from search | Su et al. $[80]_{8\to50\%}$ | – | – | – | – | – | – | – |
| | 25 | | Su et al. $[80]_{10\to50\%}$ | – | – | – | – | – | – | – |
| | 26 | | $\text{Ours}_{8\to50\%}$ | 14.9 | 28.39 | 9.78 | 24.94 | 2.48 | **0.052** | **336x** |
| | 27 | | $\text{Ours}_{10\to50\%}$ | 14.9 | **28.53** | **9.88** | **25.10** | **2.88** | | |

Table 4.6: Analysis of length-transfer summary generation. A subscript $i \to j$ (or $j\%$) refers to a model trained with $i$ words and tested for $j$ (or $j\%$) words. **Len:** Average length of predicted summaries. **R-1, R-2, R-L:** ROUGE-1, ROUGE-2, ROUGE-L. **ΔR:** The difference of total ROUGE (sum of R-1, R-2, and R-L) in comparison with the (previous) state-of-the-art model [76] under replication. **Inf.Time:** Average inference time in seconds for one sample on an i9-9940X CPU and a RTX6000 GPU. **Speedup:** Relative to [76]. †Results quoted from previous papers; others are given by our experiments. Se et al. [80]'s approach has a soft length penalty to encourage short output, but cannot generate longer summaries than trained.

We further compare with Su et al. [80], who use a soft length penalty to encourage short summaries. However, their length control works in the statistical sense but may fail for individual samples. Moreover, such a length penalty cannot generate longer

summaries than trained. Even in the setting of $10 \rightarrow 8$, their generates summaries are slightly longer than required, while the performance degrades much more considerably than our approach.

These results show that our novel length-control decoding algorithm is not only effective when generating summaries of similar length to the training targets, but also generalizes well to different desired summary lengths without re-training. In general, our method is an effective and efficient unsupervised summarization system with the ability to explicitly control the number of words.

# Chapter 5

# Summarization with Character-Level Length Control

## 5.1 Overview

Despite the effectiveness of the word-level length-control algorithm, constraining the summary length by the number of characters is a more realistic setting in real-world applications. For example, the headline shown in a mobile app or web page is constrained by the screen width (roughly speaking, the number of characters), rather than the number of words.

In this chapter, we introduce a character-level length-control algorithm, which can explicitly control the number of characters in a summary. Similar to previous chapters, we also show its experimental performance on benchmark datasets, i.e., Gigaword and DUC2004. Besides, we analyze the effect of different parameters of our algorithm.

## 5.2 Methodology

### 5.2.1 The Proposed Algorithm

Our character-level length-control algorithm, similar to the word-level one proposed in Chapter 4, is based on dynamic programming (DP). Specifically, we formulate character-level length control as a knapsack-like problem; we treat the number of

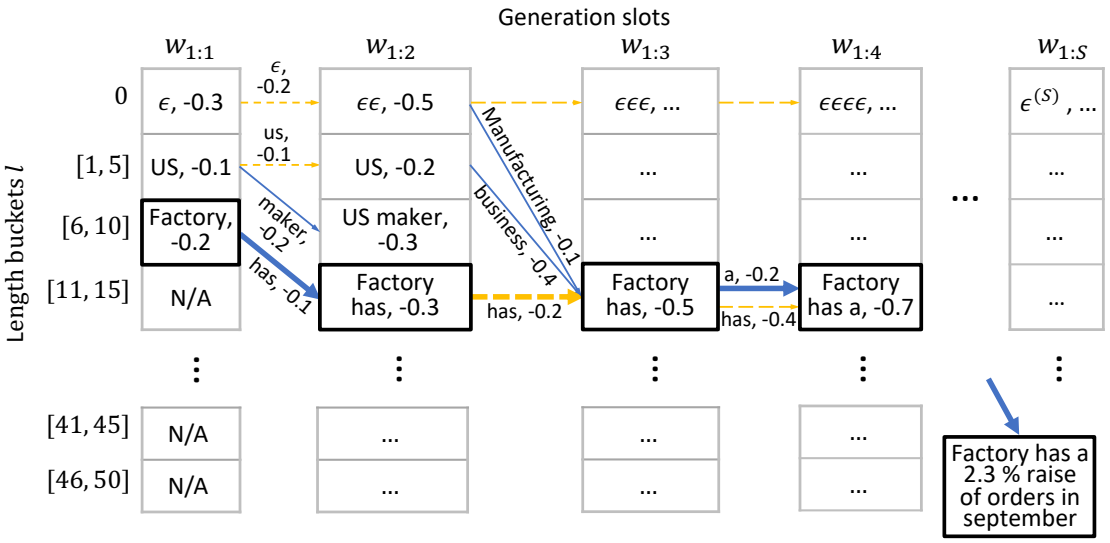


Figure 5.1: Illustration of our character-level length-control algorithm. Dashed yellow arrows refer to transitions that do not increase the summary length, while solid blue arrows refer to the increase of length. Thick arrows and blocks refer to the selected path by CTC. Due to the space limit, $\epsilon$ is omitted in the predicted sentence, and we use $\epsilon^{(S)}$ to denote a sequence of $S$-many $\epsilon$s. The number demonstrates the value (i.e., log-probability) of a word.

characters in a word (plus one) as the weight,[1] denoted by $u(\mathrm{w})$ for the word w, and the predicted log-probability as the value $v_s(\mathrm{w}) = \log P_s(\mathrm{w}|\mathbf{x})$ for the prediction slot $s$. Our goal of character-level length-control summarization can be formulated as

$$\underset{\mathrm{w}_1,\cdots,\mathrm{w}_S}{\text{maximize}} \quad \sum_{s=1}^{S} v_s(\mathrm{w}_s) \tag{5.1}$$

$$\text{subject to} \sum_{\substack{\mathrm{y}\in\mathbf{y} \\ \mathbf{y}=\Gamma(\mathrm{w}_1,\cdots,\mathrm{w}_S)}} u(\mathrm{y}) < U \tag{5.2}$$

where $U$ is the total length budget. Here, the value is the sum of the log-probability of every generation slot including $\epsilon$, whereas the length is said in terms of the words of the CTC-reduced sequence $\mathbf{y} = \Gamma(\mathrm{w}_1, \cdots, \mathrm{w}_S)$.

We observe that handling every possible integer weight (i.e., length) as in a standard knapsack algorithm may slow down the inference. Thus, we divide the lengths into buckets for efficient inference. Formally, let the $l$th bucket cover the length ranging

[1]For the purposes of research, we assume every word is appended with another character, namely, a white space. In real applications, our algorithm can be applied with any measure of length, such as the display width of a word in some font.

from $\alpha \cdot (l-1)+1$ to $\alpha \cdot l$ characters, where $\alpha$ is a hyperparameter controlling the bucket size. We denote by $\mathbf{d}^{s,l} = \mathrm{d}_1^{s,l} \cdots \mathrm{d}_s^{s,l}$ the most probable[2] $s$-token sequence that is reduced to a summary in the $l$th length bucket. Specially, we let $\mathbf{d}^{s,0}$ mean that the reduced summary has zero words.

The initialization of $\mathbf{d}^{s,l}$ fills in the DP table for $l = 0$ and $s = 1$.

- For $l = 0$, we must have $\mathbf{d}^{s,0} = \underbrace{\epsilon \cdots \epsilon}_{s\text{-many}}$, because $l = 0$ means no non-$\epsilon$ word has been generated. (First row of Figure 5.1)
- For $s = 1$, we have

$$\mathbf{d}^{1,l} = \begin{cases} \epsilon, & \text{if } \ l = 0 \\ \underset{\mathrm{w}:u(\mathrm{w})\in[\alpha\cdot(l-1)+1,\alpha\cdot l]}{\operatorname{argmax}} v_1(\mathrm{w}), & \text{if } \ l > 0 \end{cases} \tag{5.3}$$

  Here, $l = 0$ is the same as the previous bullet item. For $l > 0$, we select the most probable word for each length bucket according to the value $v_1(\cdot)$, i.e., the predicted log-probability of the first generation slot. (First column of Figure 5.1.)

The DP recursion is to compute $\mathbf{d}^{s,l}$ based on a newly predicted token $\mathrm{w}_s$, assuming its top-left sub-table is filled. This involves three scenarios:

- Case 1: $\mathrm{w}_s = \epsilon$. In this case, the new word is $\epsilon$. Thus, the index for generation slots increases from $s-1$ to $s$, but the summary length does not change. We denote $\mathscr{D}_1^{s,l}$ as the set containing the candidate sequence, given by

$$\mathscr{D}_1^{s,l} = \left\{ \mathbf{d}^{s-1,l} \oplus \epsilon \right\} \tag{5.4}$$

  where $\oplus$ denotes string concatenation. (See yellow dash arrows in Figure 5.1.)
- Case 2: $\mathrm{w}_s \neq \epsilon$, but $\mathrm{w}_s = \mathrm{d}_{s-1}^{s-1,l}$. In other words, the candidate non-$\epsilon$ word $\mathrm{w}_s$ for the $s$th slot is identical to the last token of $\mathbf{d}^{s-1,l}$. Since repeated tokens are merged during CTC decoding, the output length index $l$ is unchanged. We include this sequence in a set:

$$\mathscr{D}_2^{s,l} = \left\{ \mathbf{d}^{s-1,l} \oplus \mathrm{d}_{s-1}^{s-1,l} \right\} \tag{5.5}$$

[2]In theory, beam search may also be adopted here as in §4.2. However, our pilot study shows unnoticeable improvement and thus we do not equip this algorithm with beam search.

(Also see yellow dash arrows in Figure 5.1.)

- Case 3: $\mathrm{w}_s \neq \epsilon$ and $\mathrm{w}_s \neq \mathrm{d}_{s-1}^{s-1,l'}$ for some $l' \leq l$. That is, $\mathrm{w}_s$ is neither $\epsilon$ nor repetition, and thus the summary length will be increased from bucket $l'$ to $l$. We denote this candidate set by

$$\mathscr{D}_3^{s,l} = \Bigg\{ \mathbf{d}^{s-1,l'} \oplus \mathrm{w}_s \ : \ \Big( u(\mathrm{w}_s) + \sum_{\mathrm{d} \in \mathbf{d}^{s-1,l'}} u(\mathrm{d}) \Big) \in [\alpha \cdot (l-1) + 1, \alpha \cdot l], \quad (5.6)$$
$$\mathrm{w}_s \neq \epsilon, \mathrm{w}_s \neq \mathrm{d}_{s-1}^{s-1,l'}, \text{ and } l' \leq l \Bigg\}$$

(See blue arrows in Figure 5.1.)

Then, our DP finds the most probable sequence at each recursion step:

$$\mathbf{d}^{s,l} = \underset{\mathbf{d} \in \mathscr{D}_1^{s,l} \cup \mathscr{D}_2^{s,l} \cup \mathscr{D}_3^{s,l}}{\operatorname{argmax}} \sum_{\mathrm{s}=1}^{S} v_s(\mathrm{d}_s) \quad (5.7)$$

where $\mathrm{d}_s$ is the $s$th token of a sequence $\mathbf{d}$ from the three candidate sets above.

### 5.2.2 Theoretical Analysis

Similar to §4.2.2, we present a theorem regarding the exactness of our character-level length-control algorithm.

**Theorem 2.** *(1) If the bucket size $\alpha = 1$ and consecutive repetitions are not merged, then $\mathbf{d}^{S,T}$ is the most probable sentence of $T$ characters given by the $S$ prediction slots. (2) If $\alpha \neq 1$ or repeating tokens are merged, our algorithm may not be exact.*

*Proof.* [Part (1)] Our model is trained by the Connectionist Temporal Classification (CTC) algorithm [23], which merges repeated consecutive tokens and removes $\epsilon$s in the output sequence. Since the merging operation establishes dependencies between tokens in the output sequence, our length-control algorithm is inexact.

In this part, we consider the non-merging reduction operation $\Gamma'$ defined in Theorem 1, for example, $\Gamma'(aa\epsilon abb\epsilon) = aaabb$. Our thus revised algorithm works as follows.

We denote $\widetilde{\mathbf{d}}^{s,l} = \widetilde{\mathrm{d}}_1^{s,l} \cdots \widetilde{\mathrm{d}}_s^{s,l}$ as the recursion variable, being the most probable $s$-token sequence that is reduced to a summary of length $l$.

The initialization of $\widetilde{\mathbf{d}}^{s,l}$ is the same as the original length-control algorithm (§5.2.1), since the merging operation is not involved here. However, the recursion involves only two cases:

- Case 1: $\mathrm{w}_s = \epsilon$. The recursion of this case is also the same (see Eqn. 5.4):

$$\widetilde{\mathscr{D}}_1^{s,l} = \left\{ \widetilde{\mathbf{d}}^{s-1,l} \oplus \epsilon \right\} \tag{5.8}$$

- Case 2: $\mathrm{w}_s \neq \epsilon$. We have a set of candidate sequences:

$$\widetilde{\mathscr{D}}_2^{s,l} = \left\{ \widetilde{\mathbf{d}}^{s-1,l'} \oplus \mathrm{w}_s \ : \ \left( u(\mathrm{w}_s) + \sum_{\mathrm{d} \in \widetilde{\mathbf{d}}^{s-1,l'}} u(\mathrm{d}) \right) = l, \mathrm{w}_s \neq \epsilon, \text{ and } l' < l \right\} \tag{5.9}$$

  This is analogous to Eqn. (5.6), where $\alpha = 1$ (due to our theorem assumption). Also, the condition $\mathrm{w}_s \neq \widetilde{\mathrm{d}}_{s-1}^{s-1,l'}$ in Eqn. (5.6) is dropped here because this algorithm variant does not merge repeated tokens.

Then, the algorithm chooses the most probable candidate sequence as $\widetilde{\mathbf{d}}^{s,l}$, given by

$$\widetilde{\mathbf{d}}^{s,l} = \operatorname*{argmax}_{\mathbf{d} \in \widetilde{\mathscr{D}}_1^{s,l} \cup \widetilde{\mathscr{D}}_2^{s,l}} \sum_{s=1}^{S} v_s(\mathrm{d}_s) \tag{5.10}$$

Now we will prove that the algorithm is exact: suppose $P_{s,l} := \sum_{i=1}^{s} v_i(\widetilde{\mathrm{d}}_i^{s,l})$ is the log probability of $\widetilde{\mathbf{d}}^{s,l}$, we have

$$P_{s,l} = \max_{\mathrm{d}_1 \cdots \mathrm{d}_s : |\Gamma'(\mathrm{d}_1 \cdots \mathrm{d}_s)| = l} \sum_{i=1}^{s} v_i(\mathrm{d}_i) \tag{5.11}$$

In other words, $\widetilde{\mathbf{d}}^{s,l}$ is the most probable $s$-token sequence that is reduced to length $l$. This is proved by mathematical induction as follows.

**Base Cases.** For $l = 0$, the variable $\widetilde{\mathbf{d}}^{s,0}$ can only be $s$-many $\epsilon$s. The optimality in Eqn. (5.11) holds trivially.

For $s = 1$ but $l > 0$, the algorithm chooses $\widetilde{\mathrm{d}}^{1,l} = \operatorname*{argmax}_{\mathrm{d}_1:u(\mathrm{d}_1)=l} v_1(\mathrm{d}_1)$. Therefore, $P_{1,l} = \max_{\mathrm{d}_1:|\Gamma'(\mathrm{d}_1)|=l} v_1(\mathrm{d}_1)$, showing that Eqn. (5.11) is also satisfied with only one term in the summation.

**Induction Step.** The induction hypothesis assumes

$$P_{s-1,l'} = \max_{\mathrm{d}_1\cdots\mathrm{d}_{s-1}:|\Gamma'(\mathrm{d}_1\cdots\mathrm{d}_{s-1})|=l'} \sum_{i=1}^{s-1} v_i(\mathrm{d}_i)$$

for every $l' < l$. We will show that the algorithm finds the sequence $\widetilde{\mathbf{d}}^{s,l}$ with

$$P_{s,l} = \max_{\mathrm{d}_1\cdots\mathrm{d}_s:|\Gamma'(\mathrm{d}_1\cdots\mathrm{d}_s)|=l} \sum_{i=1}^{s} v_i(\mathrm{d}_i)$$

.

According to Eqn. (5.10), the variable $\widetilde{\mathbf{d}}^{s,l}$ is the most probable sequence in $\widetilde{\mathscr{D}}_1^{s,l} \cup \widetilde{\mathscr{D}}_2^{s,l}$. Thus, we have

$$\begin{aligned}
P_{s,l} &= \max_{l',\mathrm{d}_s:l'+u(\mathrm{d}_s)=l} \{P_{s-1,l'} + v_s(\mathrm{d}_s)\} && (5.12)\\
&= \max_{l'} \left\{ P_{s-1,l'} + \max_{\mathrm{d}_s:l'+u(\mathrm{d}_s)=l} v_s(\mathrm{d}_s) \right\} && (5.13)\\
&= \max_{l'} \left\{ \max_{\mathrm{d}_1\cdots\mathrm{d}_{s-1}:|\Gamma'(\mathrm{d}_1\cdots\mathrm{d}_{s-1})|=l'} \sum_{i=1}^{s-1} v_i(\mathrm{d}_i) + \max_{\mathrm{d}_s:l'+u(\mathrm{d}_s)=l} v_s(\mathrm{d}_s) \right\} && (5.14)\\
&= \max_{l'} \left\{ \max_{\substack{\mathrm{d}_1\cdots\mathrm{d}_s:\\ |\Gamma'(\mathrm{d}_1\cdots\mathrm{d}_{s-1})|=l'\\ |\Gamma'(\mathrm{d}_1\cdots\mathrm{d}_s)|=l}} \sum_{i=1}^{s} v_i(\mathrm{d}_i) \right\} && (5.15)\\
&= \max_{\mathrm{d}_1\cdots\mathrm{d}_s:|\Gamma'(\mathrm{d}_1\cdots\mathrm{d}_s)|=l} \sum_{i=1}^{s} v_i(\mathrm{d}_i) && (5.16)
\end{aligned}$$

Here, (5.13) separates the max operation over $l'$ and $\mathrm{d}_s$; (5.14) is due to the induction hypothesis; (5.15) holds because the two max terms in (5.14) are independent given $l'$, and thus the summations can be grouped; and (5.16) further groups the two max operations with $l'$ eliminated. The last two lines are originally proved in [30] and also used in [12].

[Part (2)] We now prove our algorithm may be inexact if $\alpha \neq 1$ or repeated tokens are merged. We show these by counterexamples.[3]

[3]To make our counterexample intuitive, we work with probabilities, rather than log probabilities.

| Word | $P_1(\cdot\|\mathbf{x})$ | $P_2(\cdot\|\mathbf{x})$ |
|---|---|---|
| I | 0.3 | 0.1 |
| am | 0.4 | 0.6 |
| a | 0.2 | 0.05 |
| $\epsilon$ | 0.1 | 0.25 |

Table 5.1: A counterexample showing that our algorithm may be inexact if $\alpha \neq 1$ or repeated tokens are merged. Here, we set the vocabulary to be three words plus a blank token $\epsilon$.

Suppose $\alpha \neq 1$ and in particular we assume $\alpha = 2$. We further assume repeated tokens are not merged. Consider the example shown in Table 5.1. The length-control algorithm finds $\widetilde{\mathbf{d}}^{1,1} = \{\text{“am”}\}$, and then $\widetilde{\mathbf{d}}^{2,2} = \{\text{“am I”}\}$ with the probability of $0.4 \cdot 0.1 = 0.04$, as the first bucket covers the length range $[1, 2]$ and second $[3, 4]$. Here, we notice that two words are separated by a white space, which also counts as a character. However, the optimum should be {“I am”}, which has a probability of $0.3 \cdot 0.6 = 0.18$.

Now suppose repeated tokens are merged, and we further assume the length bucket $\alpha = 1$ in this counterexample. Again, this can be shown by Table 5.1: the algorithm finds $\mathbf{d}^{1,1} = \{\text{“I”}\}$ and $\mathbf{d}^{1,2} = \{\text{“am”}\}$, based on which we have $\mathbf{d}^{2,3} = \{\text{“I a”}\}$ with probability $0.3 \cdot 0.05 = 0.015$. However, the optimum should be {“a I”} with probability $0.2 \cdot 0.1 = 0.02$.

□

The above theoretical analysis helps us understand when our algorithm is exact (or inexact). Empirically, our approach works well as an approximate inference algorithm.

**Discussion.** Our DP algorithm is inspired by the standard 0-1 knapsack problem [3], but also differs in several significant ways. First, we merge consecutive tokens during CTC decoding; this establishes some dependencies among different generation slots, and thus exact inference with DP is not possible. Second, our value function is non-stationary, as it changes over time. We require that every slot should select a

token, either $\epsilon$ or a word. In both cases, the token's value is added to the total value. Therefore, our algorithm is compatible with negative values, namely, log probabilities in our application, because only the relative difference matters for the value function.

## 5.3 Experiments

### 5.3.1 Setup

Similarly to §4.3, we use ROUGE scores to evaluate the performance; we experimented with our character-level length-control algorithm on the Gigaword headline generation and DUC2004 datasets.

Our training parameters, still, were the same as §3.3.2. For the character-level length-control algorithm, we adopted a bucket size of 4 and only considered the most probable 20 words for every generation slot (cf. $w_s$ in Eqn. 5.6) due to efficiency concerns.

### 5.3.2 Results

In this section, we compare our method with previous non-autoregressive summarization models in the character-level length-control setting.

**Results on Gigaword Headline Generation.** Table 5.2 presents the performance on the Gigaword test set under two different length settings, i.e., 50 and 60 characters, where machine learning models are trained by 8 and 10-word pseudo-summaries [76], respectively; this is because 8-word summaries have an average length of $48.75 \approx 50$ characters, while 10-word ones have $60.17 \approx 60$. Additionally, we observe that our proposed algorithm is the only machine learning-based approach that can perform explicit character-level length control. For fair comparison, we truncate the output of other methods to satisfy the length constraints.

As seen, our model even with truncating outperforms all other non-autoregressive (NAR) models [80, 68, 98] in both settings. Specifically, Su et al. [80] emit multiple end-of-sequence tokens at the end of the output sequence to generate a shorter summary

| Group | # | Approach | | Len | ROUGE F1 | | | | Time |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | R-1 | R-2 | R-L | ΔR | |
| A (desired length 50) | 1 | Baseline | Lead-50 chars | 49.03 | 20.66 | 7.08 | 19.30 | -9.23 | – |
| | 2 | Search | Schumann et al. [76] (truncate) | 45.45 | 24.98 | 9.08 | 23.18 | 0.97 | 9.573 |
| | 3 | | Char constrained search | 44.05 | 25.30 | **9.25** | 23.43 | 1.71 | 17.324 |
| | 4 | NAR | Su et al. [80] (truncate) | 45.24 | 24.65 | 8.64 | 22.98 | 0 | 0.017 |
| | 5 | | Qi et al. [68] (truncate) | 44.54 | 24.31 | 7.66 | 22.48 | -1.82 | 0.019 |
| | 6 | | Yang et al. [98] (truncate) | 49.37 | 21.70 | 4.60 | 20.13 | -9.84 | – |
| | 7 | | Ours (truncate) | 47.77 | 25.79 | 8.94 | 23.75 | 2,21 | **0.012** |
| | 8 | | Ours (length control) | 47.03 | **27.45** | 8.87 | **25.14** | **5.19** | 0.025 |
| B (desired length 60) | 9 | Baseline | Lead-60 chars | 49.03 | 22.06 | 7.56 | 20.40 | -6.23 | – |
| | 10 | Search | Schumann et al. [76] (truncate) | 55.47 | 26.00 | 9.57 | 23.83 | 3.15 | 21.951 |
| | 11 | | Char constrained search | 56.09 | 26.15 | 9.58 | 23.84 | 3.32 | 42.459 |
| | 12 | NAR | Su et al. [80] (truncate) | 55.58 | 24.92 | 8.54 | 22.79 | 0 | 0.018 |
| | 13 | | Qi et al. [68] (truncate) | 54.14 | 25.17 | 8.44 | 23.02 | 0.38 | 0.020 |
| | 14 | | Yang et al. [98] (truncate) | 58.74 | 23.21 | 5.21 | 21.79 | -6.04 | – |
| | 15 | | Ours (truncate) | 57.23 | 26.80 | **9.73** | 24.44 | 4.72 | **0.012** |
| | 16 | | Ours (length control) | 57.13 | **28.00** | 9.63 | **25.35** | **6.73** | 0.032 |

Table 5.2: Performance on the Gigaword headline generation test set, where NAR stands for non-autoregressive. **Len:** Average number of characters in the predicted summaries. **R-1, R-2, R-L:** ROUGE-1, ROUGE-2, ROUGE-L. **ΔR:** The difference of total ROUGE (sum of R-1, R-2, and R-L) in comparison with the (previous) state-of-the-art NAR summarization system [80]. **Time:** Average inference time in seconds for one sample on an i9-9940X CPU and an RTX6000 GPU.[5]

than the source text (Rows 4 & 12); Qi et al. [68] propose to pretrain a summarization system in an autoregressive manner, and gradually adapt it to the NAR setting (Rows 5 & 13); Yang et al. [98][4] propose a two-step strategy of autoregressive part-of-speech (POS) prediction and non-autoregressive summary generation (Rows 6 & 14). Our model trained by CTC, even with character-level truncating, is able to surpass all these methods and the search-based teacher [76], which again demonstrates the effectiveness of our approach.

[4]Yang et al. [98] only provided execution commands in their GitHub repo, but no training code. We emailed the authors, but have not obtained the code either. The reported results are based on our best replication.

[5]Some performance is not consistent with Table 4.2 due to different implementations.

| # | Approach | | ROUGE Recall | | | | Time |
|---|---|---|---|---|---|---|---|
| | | | R-1 | R-2 | R-L | ΔR | |
| 1 | Baseline | Lead-75 chars | 22.52 | 6.50 | 19.74 | -4.97 | – |
| 2 | Search | Schumann et al. [76] (truncate) | 26.09 | **8.03** | 22.86 | 3.25 | 30.362 |
| 3 | | Char-constrained search | 26.30 | 7.95 | 22.78 | 3.30 | 31.540 |
| 4 | NAR | Su et al. [80] (truncate) | 24.67 | 7.25 | 21.81 | 0 | 0.017 |
| 5 | | Qi et al. [68] (truncate) | 22.79 | 5.91 | 20.05 | -4.98 | 0.018 |
| 6 | | Ours (truncate) | 26.43 | 7.86 | 22.66 | 3.22 | 0.012 |
| 7 | | Ours (length control) | **28.37** | 7.74 | **24.30** | **6.68** | 0.030 |

Table 5.3: Results on DUC2004 dataset.

Equipped with the length-control algorithm, our model has a significant boost in terms of ROUGE scores. In a fair comparison with the same base architecture, the length-control algorithm alone improves truncating by 2–3 points. Our full model achieves an improvement of more than 5 total ROUGE points compared with previous state-of-the-art NAR methods.

Regarding the inference efficiency, our method with truncating is faster than previous NAR models. Even with the length-control algorithm, our model is still in the same ballpark, being ∼500x faster than the search-based method.

Additionally, we design a variant of the unsupervised summarization method based on [76], where we directly impose the character-level length constraint during each search step (Rows 3 & 11). We find this approach outperforms truncating word-constrained search (Rows 2 & 10), but is much worse than our machine learning-based model with the length-control algorithm.

**Results on DUC2004.** Table 5.3 shows the performance of our model on DUC2004, where we constrain the summary length to be at most 75 characters, following previous work [91, 4, 76]. Following §3.3 and §4.3, we adopted 13-word summaries (∼80 characters) from [76] as our training targets.

Experiments show that our method with length control again largely outperforms previous NAR models, while retaining high inference efficiency. Results are consistent

| # | Approach | | Len | ROUGE F1 | | | Time |
|---|---|---|---|---|---|---|---|
| | | | | R-1 | R-2 | R-L | |
| 1 | AR | Transformer (truncate) | 46.62 | 26.31 | **9.33** | 24.29 | 0.092 |
| 2 | | Transformer (length control) | 45.23 | 25.33 | 9.03 | 23.44 | 0.095 |
| 3 | NAR | Ours (truncate) | 47.77 | 25.79 | 8.94 | 23.75 | **0.012** |
| 4 | | Ours (length control) | 47.03 | **27.45** | 8.87 | **25.14** | 0.025 |

Table 5.4: Comparing autoregressive (AR) and non-autoregressive (NAR) models on the Gigaword headline generation test set. Our length-control algorithm requires the predicted probabilities to be independent, and thus is not compatible with AR models.

with the Gigaword experiment.

### 5.3.3 Comparison with Autoregressive Models

We are curious about how our non-autoregressive model with character-level length control is compared with autoregressive (AR) methods. Thus, we consider the 50-character setting in Table 5.2, and train a standard AR Transformer with truncating and length-control decodings, and show results in Table 5.4.

As seen, our length-control algorithm is not compatible with the AR Transformer and hurts the ROUGE scores (Row 2). This is because our algorithm is based on dynamic programming and requires model outputs to be local, so that the length-control problem can be divided into shared sub-problems; however, the predicted probabilities of the AR Transformer depend on the partial generation at previous time steps. Note that this is not a disadvantage of our approach, but shows that NAR generation provides unique opportunities for length control.

Moreover, our approach achieves higher performance than the AR Transformer, which is a very strong result. This is because we learn from a search-based method that extracts source words as the summary [76], and our CTC training—with blank tokens scattering over the whole output sentence as appropriate—can capture such strong correspondence between input and output. Moreover, the proposed length-control algorithm is able to maintain the summary completeness given the length constraint,

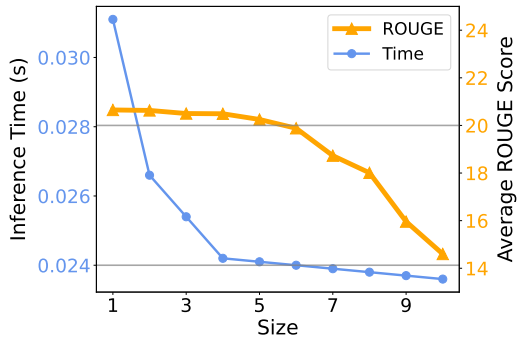


Figure 5.2: Performance of our method with different bucket sizes.

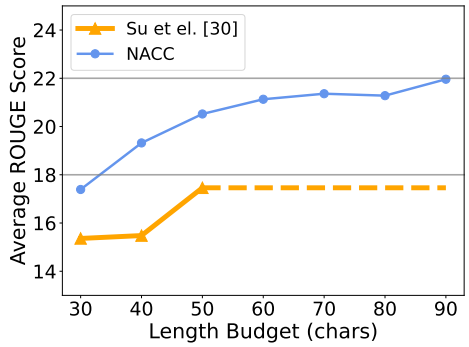


Figure 5.3: Length-transfer performance of our model and Su et al. [80].

achieving better ROUGE scores than our model with truncating.

### 5.3.4 Analysis of the Length Bucket

Our dynamic programming is an approximate algorithm with an $\alpha$-sized length bucket (see Figure 5.1 and §5.2.1). Here, we investigate the effect of the bucket size in terms of ROUGE scores (the arithmetic mean of R-1, R-2, and R-L) and inference efficiency when training targets are 8-word summaries from [76], following Table 5.2.

As seen in Figure 5.2, the ROUGE score continuously decreases with a larger bucket size (thick orange curve). This not only confirms the inexactness of our algorithm, but also shows that a small bucket size does not hurt the performance much. On the other hand, the inference time decreases drastically at the beginning (thin blue curve) because we have fewer dynamic programming steps; as the bucket size increases, the inference time convergences to our approach without length control. Based on this analysis, we set the bucket size to be 4 in our experiments.

### 5.3.5 Case Study

Table 5.5 shows example summaries generated by our method and the AR Transformer on the Gigaword test set.

As seen, a model without length control may generate a summary that happens to have the desired length (AR Transformer), or a longer summary (both AR Transformer

| | |
|---|---|
| **Input:** singapore airline and delta air lines announced two differing strategies to upgrade their long-haul in-flight service for business travelers . | |
| **Reference:** business travel : competing strategies ; crowded skies | |
| **50-character Setting:** | |
| **AR Transformer (no control):** | delta air lines differing strategies to upgrade their in-flight service for business travelers |
| **Ours (no control):** | delta air lines differing strategies to upgrade long-haul in-flight service for business travelers |
| **Ours (length control):** | delta air lines upgrade service business travelers |

Table 5.5: Example summaries for Gigaword headline generation, where gray words are truncated for fair comparison.

and our method). A longer summary requires truncating for explicit length control, which is undesired.

By contrast, our proposed algorithm is able to generate a summary whose length is close to but less than the length budget. The resulting summary is more complete than truncating, and better keeps the key information.

### 5.3.6 Length-Transfer Generation

Our model is capable of length-transfer generation, that is, generating summaries of different lengths from the training targets. Such generation is important to real-world applications where summaries of various lengths are needed for the same input, e.g., fitting different screen widths. Although generating a short enough summary may satisfy all possible length constraints, a longer summary that better utilizes the length budget can preserve more information; this is also reflected by ROUGE scores, which prefer longer summaries, as shown in [76].

Figure 5.3 compares the performance of our method with Su et al. [80] when learning from 8-word summaries. When the inference length budget is less than training ($x$-axis $< 50$), the ROUGE score of our approach decreases almost linearly with a decreasing length budget, but Su et al. [80]'s approach degrades faster than ours. For $x$-axis $> 50$, we find the soft penalty in [80] is unable to generate longer summaries than trained (shown by the dashed orange line), whereas our approach is able to utilize the increased length budget and achieve higher ROUGE scores.

# Chapter 6

# Conclusion & Future Work

## 6.1 Conclusion

In this thesis, we propose a non-autoregressive unsupervised summarization model, which does not require any labeled training data and thus is applicable to domains where paired samples are difficult to collect. Further, we develop two length-control algorithms, which are able to explicitly control the summary length at the word and character levels, respectively.

Experiments show that our approach not only achieves unsupervised state-of-the-art performance on Gigaword and DUC2004 datasets under length constraints, but also is at least several times faster than an autoregressive Transformer [88] at inference time. Moreover, our approach is able to perform length-transfer generation, that is, generating summaries of different lengths from the training target.

## 6.2 Limitation & Future Work

This thesis focuses on unsupervised summarization due to the importance of low-data applications. One limitation is that we have not obtained rigorous empirical results for supervised summarization, where our length-control algorithms may also work. This is because previous supervised summarization studies lack explicit categorization of summary lengths [101, 68], making comparisons unfair and problematic [76]. Such an observation is also evidenced by Su et al. [80], where the same model may differ by

a few ROUGE points when generating summaries of different lengths. Nevertheless, we have compared with Su et al. [80] in our setting and show the superiority of our approach under fair comparison. We plan to explore supervised summarization in future work after we establish a rigorous experimental setup.

# References


[1] Armen Aghajanyan et al. “Better fine-tuning by reducing representational collapse”. In: *ICLR*. 2020.

[2] Armen Aghajanyan et al. “Muppet: Massive multi-task representations with pre-finetuning”. In: *EMNLP*. 2021, pp. 5799–5811.

[3] Rumen Andonov, Vincent Poirriez, and Sanjay Rajopadhye. “Unbounded knapsack problem: Dynamic programming revisited”. In: *European Journal of Operational Research* 123.2 (2000), pp. 394–407.

[4] Christos Baziotis et al. “SEQ3: Differentiable sequence-to-sequence-to-sequence autoencoder for unsupervised abstractive sentence compression”. In: *NAACL-HLT*. 2019, pp. 673–681.

[5] Rami Benbenishty et al. “Testing the causal links between school climate, school violence, and school academic performance: A cross-lagged panel autoregressive model”. In: *Educational Researcher* 45.3 (2016), pp. 197–206.

[6] William Chan et al. “Imputer: Sequence modelling via imputation and dynamic programming”. In: *ICML*. 2020, pp. 1403–1413.

[7] Ethan A. Chi, Julian Salazar, and Katrin Kirchhoff. “Align-refine: Non-autoregressive speech recognition via iterative realignment”. In: *NAACL-HLT*. 2021, pp. 1920–1927.

[8] Eunsol Choi et al. “QuAC: Question answering in context”. In: *EMNLP*. 2018, pp. 2174–2184.

[9] Yuntian Deng and Alexander M Rush. “Cascaded text generation with Markov Transformers”. In: *NeurIPS*. 2020, pp. 170–181.

[10] Yue Dong et al. “BanditSum: Extractive summarization as a contextual bandit”. In: *EMNLP*. 2018, pp. 3739–3748.

[11] Bonnie Dorr, David Zajic, and Richard Schwartz. “Hedge trimmer: A parse-and-trim approach to headline generation”. In: *HLT-NAACL Text Summarization Workshop*. 2003, pp. 1–8.

[12] J.-B. Durand, P. Goncalves, and Y. Guedon. “Computational methods for hidden Markov tree models: An application to wavelet trees”. In: *IEEE Transactions on Signal Processing* 52.9 (2004), pp. 2551–2560.

[13] Harold P Edmundson. “New methods in automatic extracting”. In: *Journal of the Association for Computing Machinery* 16.2 (1969), pp. 264–285.

[14] Günes Erkan and Dragomir R Radev. “LexRank: Graph-based lexical centrality as salience in text summarization”. In: *Journal of Artificial Intelligence Research* 22.1 (2004), pp. 457–479.

[15] Mohamed Abdel Fattah and Fuji Ren. “GA, MR, FFNN, PNN and GMM based models for automatic text summarization”. In: *Computer Speech & Language* 23.1 (2009), pp. 126–144.

[16] Ronen Feldman. "Techniques and applications for sentiment analysis". In: *Communications of the ACM* 56.4 (2013), pp. 82–89.

[17] Ronen Feldman, James Sanger, et al. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, 2007.

[18] Thibault Févry and Jason Phang. "Unsupervised sentence compression using denoising auto-encoders". In: *CoNLL*. 2018, pp. 413–422.

[19] Kiran Garimella et al. "Quantifying controversy in social media". In: *ACM International Conference on Web Search and Data Mining*. 2016, pp. 33–42.

[20] Sebastian Gehrmann, Yuntian Deng, and Alexander M Rush. "Bottom-up abstractive summarization". In: *EMNLP*. 2018, pp. 4098–4109.

[21] Jade Goldstein et al. "Summarizing text documents: Sentence selection and evaluation metrics". In: *ACM SIGIR*. 1999, pp. 121–128.

[22] David Graff et al. "English Gigaword". In: *Linguistic Data Consortium, Philadelphia* (2003).

[23] Alex Graves et al. "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks". In: *ICML*. 2006, pp. 369–376.

[24] Jiatao Gu and Xiang Kong. "Fully non-autoregressive neural machine translation: Tricks of the trade". In: *ACL-IJCNLP*. 2021, pp. 120–133.

[25] Jiatao Gu et al. "Non-autoregressive neural machine translation". In: *ICLR*. 2018.

[26] Marti A. Hearst and Christian Plaunt. "Subtopic structuring for full-length document access". In: *ACM SIGIR*. 1993, pp. 59–68.

[27] Karl Moritz Hermann et al. "Teaching machines to read and comprehend". In: *NeurIPS*. 2015, pp. 1693–1701.

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

[29] Chenyang Huang et al. "Non-autoregressive translation with layer-wise prediction and deep supervision". In: *AAAI*. 2022, pp. 10776–10784.

[30] Frederick Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1998.

[31] Lianrui Jia. "What public and whose opinion? A study of Chinese online public opinion analysis". In: *Communication and the Public* 4.1 (2019), pp. 21–34.

[32] Ruipeng Jia et al. "Flexible non-autoregressive extractive summarization with threshold: How to extract a non-fixed number of summary sentences". In: *AAAI*. 2021, pp. 13134–13142.

[33] Mikael Kågebäck et al. "Extractive summarization using continuous vector space models". In: *Workshop on Continuous Vector Space Models and Their Compositionality*. 2014, pp. 31–39.

[34] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. "BERT: Pre-training of deep bidirectional Transformers for language understanding". In: *NAACL-HLT*. 2019, pp. 4171–4186.

[35] Yuta Kikuchi et al. "Controlling output length in neural encoder-decoders". In: *EMNLP*. 2016, pp. 1328–1338.

[36] Julia Kreutzer, Stefan Riezler, and Carolin Lawrence. "Offline reinforcement learning from human feedback in real-world sequence-to-sequence tasks". In: *Workshop on Structured Prediction for NLP*. 2021, pp. 37–43.

[37] Dhruv Kumar et al. "Iterative edit-based unsupervised sentence simplification". In: *ACL*. 2020, pp. 7918–7928.

[38] Zhenzhong Lan et al. "ALBERT: A lite BERT for self-supervised learning of language representations". In: *ICML*. 2019.

[39] Jason Lee, Elman Mansimov, and Kyunghyun Cho. "Deterministic non-autoregressive neural sequence modeling by iterative refinement". In: *EMNLP*. 2018, pp. 1173–1182.

[40] Mike Lewis et al. "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension". In: *ACL*. 2020, pp. 7871–7880.

[41] Jingjing Li et al. "Unsupervised text generation by learning from search". In: *NeurIPS*. 2020, pp. 10820–10831.

[42] Jiwei Li et al. "Deep Reinforcement Learning for Dialogue Generation". In: *EMNLP*. 2016, pp. 1192–1202.

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

[44] Puyuan Liu, Chenyang Huang, and Lili Mou. "Learning Non-Autoregressive Models from Search
for Unsupervised Sentence Summarization". In: *ACL*. 2022.

[45] Qian Liu et al. "You impress me: Dialogue generation via mutual persona perception". In: *ACL*. 2020, pp. 1417–1427.

[46] Shujie Liu et al. "A recursive recurrent neural network for statistical machine translation". In: *ACL*. 2014, pp. 1491–1500.

[47] Xianggen Liu et al. "Unsupervised paraphrasing by simulated annealing". In: *ACL*. 2020, pp. 302–312.

[48] Yang Liu. "Fine-tune BERT for extractive summarization". In: *arXiv preprint arXiv:1903.10318* (2019).

[49] Yizhu Liu, Zhiyi Luo, and Kenny Zhu. "Controlling length in abstractive summarization using a convolutional neural network". In: *EMNLP*. 2018, pp. 4110–4119.

[50] Yi Luan et al. "A general framework for information extraction using dynamic span graphs". In: *NAACL-HLT*. 2019, pp. 3036–3046.

[51] Hans Peter Luhn. "The automatic creation of literature abstracts". In: *IBM Journal of Research and Development* 2.2 (1958), pp. 159–165.

[52] Bodhisattwa Prasad Majumder et al. "Representation learning for information extraction from form-like documents". In: *ACL*. 2020, pp. 6495–6504.

[53] Benjamin Markines, Ciro Cattuto, and Filippo Menczer. "Social spam detection". In: *International Workshop on Adversarial Information Retrieval on the Web*. 2009, pp. 41–48.

[54] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. "Sentiment analysis algorithms and applications: A survey". In: *Ain Shams Engineering Journal* 5.4 (2014), pp. 1093–1113.

[55] Clara Meister, Ryan Cotterell, and Tim Vieira. "If beam search is the answer, what was the question?" In: *EMNLP*. 2020, pp. 2173–2185.

[56] Ning Miao et al. "CGMH: Constrained sentence generation by Metropolis-Hastings sampling". In: *AAAI*. 2019, pp. 6834–6842.

[57] Yishu Miao and Phil Blunsom. "Language as a latent variable: Discrete generative models for sentence compression". In: *EMNLP*. 2016, pp. 319–328.

[58] Rada Mihalcea and Paul Tarau. "TextRank: Bringing order into text". In: *EMNLP*. 2004, pp. 404–411.

[59] Niels Framroze Møller. "Bridging economic theory models and the cointegrated vector autoregressive model". In: *Economics* 2.1 (2008).

[60] Jessica Moore, Ben Gelman, and David Slater. "A convolutional neural network for language-agnostic source code summarization". In: *International Conference on Evaluation of Novel Approaches to Software Engineering*. 2019, pp. 15–26.

[61] Gabriel Murray, Steve Renals, and Jean Carletta. "Extractive summarization of meeting recordings". In: *European Conference on Speech Communication and Technology*. 2005, pp. 593–596.

[62] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. "SummaRuNNer: A recurrent neural network based sequence model for extractive summarization of documents". In: *AAAI*. 2017, pp. 3075–3081.

[63] Ramesh Nallapati et al. "Abstractive text summarization using sequence-to-sequence RNNs and beyond". In: *CoNLL*. 2016, pp. 280–290.

[64] Shashi Narayan, Shay B Cohen, and Mirella Lapata. "Ranking sentences for extractive summarization with reinforcement learning". In: *NAACL-HLT*. 2018, pp. 1747–1759.

[65] Paul Over and James Yen. "An introduction to DUC-2004: Intrinsic evaluation of generic news text summarization systems". In: *Document Understanding Conference*. 2004.

[66] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. "Unsupervised Learning of Sentence Embeddings Using Compositional n-Gram Features". In: *NAACL-HLT*. 2018, pp. 528–540.

[67] Romain Paulus, Caiming Xiong, and Richard Socher. "A deep reinforced model for abstractive summarization". In: *ICLR*. 2018.

[68] Weizhen Qi et al. "BANG: Bridging autoregressive and non-autoregressive generation with large scale pretraining". In: *ICML*. 2021, pp. 8630–8639.

[69] Lihua Qian et al. "Glancing Transformer for non-autoregressive neural machine translation". In: *ACL-IJCNLP*. 2021, pp. 1993–2003.

[70] Sanjeev Rao, Anil Kumar Verma, and Tarunpreet Bhatia. "A review on social spam detection: Challenges, open issues, and future directions". In: *Expert Systems with Applications* 186 (2021), p. 115742.

[71] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[72] Alexander M. Rush, Sumit Chopra, and Jason Weston. "A neural attention model for abstractive sentence summarization". In: *EMNLP*. 2015, pp. 379–389.

[73] Chitwan Saharia et al. "Non-autoregressive machine translation with latent alignments". In: *EMNLP*. 2020, pp. 1098–1108.

[74] Itsumi Saito et al. "Length-controllable abstractive summarization by guiding with summary prototype". In: *arXiv preprint arXiv:2001.07331* (2020).

[75] Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. "Improving multi-hop question answering over knowledge graphs using knowledge base embeddings". In: *ACL*. 2020, pp. 4498–4507.

[76] Raphael Schumann et al. "Discrete optimization for unsupervised sentence summarization with word-level extraction". In: *ACL*. 2020, pp. 5032–5042.

[77] Yusuke Shido et al. "Automatic source code summarization with extended tree-LSTM". In: *IJCNN*. 2019, pp. 1–8.

[78] Shengli Song, Haitao Huang, and Tongxiao Ruan. "Abstractive text summarization using LSTM-CNN based deep learning". In: *Multimedia Tools and Applications* 78.1 (2019), pp. 857–875.

[79] Gabriel Stanovsky et al. "Supervised open information extraction". In: *NAACL-HLT*. 2018, pp. 885–895.

[80] Yixuan Su et al. "Non-autoregressive text generation with pre-trained language models". In: *EACL*. 2021, pp. 234–243.

[81] Zhiqing Sun et al. "Fast structured decoding for sequence models". In: *NeurIPS*. 2019, pp. 3016–3026.

[82] Ilya Sutskever, James Martens, and Geoffrey Hinton. "Generating text with recurrent neural networks". In: *ICML*. 2011, pp. 1017–1024.

[83] Sho Takase and Shun Kiyono. "Lessons on parameter sharing across layers in Transformers". In: *arXiv preprint arXiv:2104.06022* (2021).

[84] Sho Takase and Shun Kiyono. "Rethinking perturbations in encoder-decoders for fast training". In: *NAACL-HLT*. 2021, pp. 5767–5780.

[85] Sho Takase and Naoaki Okazaki. "Positional encoding to control output sequence length". In: *NAACL-HLT*. 2019, pp. 3999–4004.

[86] Yi Tay et al. "Synthesizer: Rethinking self-attention for Transformer models". In: *ICML*. 2021, pp. 10183–10192.

[87] Antal Van den Bosch, Toine Bogers, and Maurice De Kunder. "Estimating search engine index size variability: A 9-year longitudinal study". In: *Scientometrics* 107.2 (2016), pp. 839–856.

[88] Ashish Vaswani et al. "Attention is all you Need". In: *NeurIPS*. 2017, pp. 5998–6008.

[89] Peng Wang et al. "Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework". In: *arXiv preprint arXiv:2202.03052* (2022).

[90] Yaushian Wang and Hung-Yi Lee. "Learning to encode text as human-readable summaries using generative adversarial networks". In: *EMNLP*. 2018, pp. 4187–4195.

[91] Peter West et al. "BottleSum: Unsupervised and self-supervised sentence summarization using the information bottleneck principle". In: *EMNLP-IJCNLP*. 2019, pp. 3752–3761.

[92] T. Winograd. *Procedures as A Representation for Data in A Computer Program for Understanding Natural Language*. M.I.T. Project MAC, 1971.

[93] Thomas Wolf et al. "TransferTransfo: A transfer learning approach for neural network based conversational agents". In: *arXiv preprint arXiv:1901.08149* (2019).

[94] Chien-Sheng Wu et al. "Controllable abstractive dialogue summarization with sketch supervision". In: *ACL-IJCNLP*. 2021, pp. 5108–5122.

[95] Lijun Wu et al. "R-drop: Regularized dropout for neural networks". In: *NeurIPS*. 2021, pp. 10890–10905.

[96] Peng Wu et al. "Social media opinion summarization using emotion cognition and convolutional neural networks". In: *International Journal of Information Management* 51 (2020), p. 101978.

[97] Haoran Xu, Benjamin Van Durme, and Kenton Murray. "BERT, MBERT, or BIBERT? A study on contextualized embeddings for neural machine translation". In: *arXiv preprint arXiv:2109.04588* (2021).

[98] Kexin Yang et al. "POS-constrained parallel decoding for non-autoregressive generation". In: *ACL-IJCNLP*. 2021, pp. 5990–6000.

[99] Yanxia Yang. "Research and realization of Internet public opinion analysis based on improved TF-IDF algorithm". In: *International Symposium on Distributed Computing and Applications to Business, Engineering and Science*. 2017, pp. 80–83.

[100] Zhilin Yang et al. "HotpotQA: A dataset for diverse, explainable multi-hop question answering". In: *EMNLP*. 2018, pp. 2369–2380.

[101] Ziyi Yang et al. "TED: A pretrained unsupervised summarization model with theme modeling and denoising". In: *EMNLP*. 2020, pp. 1865–1874.

[102] Li Yuan et al. "Cycle-SUM: Cycle-consistent adversarial LSTM networks for unsupervised video summarization". In: *AAAI*. 2019, pp. 9143–9150.

[103] David Zajic, Bonnie Dorr, and Richard Schwartz. "BBN/UMD at DUC-2004: Topiary". In: *HLT-NAACL Document Understanding Workshop*. 2004, pp. 112–119.

[104] Jingqing Zhang et al. "PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization". In: *ICML*. 2020, pp. 11328–11339.

[105] Xianchao Zhang, Shaoping Zhu, and Wenxin Liang. "Detecting spam and promoting campaigns in the Twitter social network". In: *IEEE International Conference on Data Mining*. 2012, pp. 1194–1199.

[106] Ming Zhong et al. "Extractive summarization as text matching". In: *ACL*. 2020, pp. 6197–6208.

[107] Jiawei Zhou and Alexander Rush. "Simple unsupervised summarization by contextual matching". In: *ACL*. 2019, pp. 5101–5106.