論文 / 著書情報
Article / Book Information

題目(和文)	タスクに応じた単語分割
Title(English)	Task-Oriented Word Segmentation
著者(和文)	平岡達也
Author(English)	Tatsuya Hiraoka
出典(和文)	学位:博士（工学）, 学位授与機関:東京工業大学, 報告番号:甲第11829号, 授与年月日:2022年3月26日, 学位の種別:課程博士, 審査員:岡崎 直観,徳永 健伸,篠田 浩一,宮崎 純,井上 中順
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第11829号, Conferred date:2022/3/26, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Doctoral Dissertation

Task-Oriented Word Segmentation

Tatsuya Hiraoka

February 24, 2022

Artificial Intelligence Course
Department of Computer Science
School of Computing
Tokyo Institute of Technology

A Doctoral Dissertation
submitted to School of Computing,
Tokyo Institute of Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Tatsuya Hiraoka

Thesis Committee:

Professor Naoaki Okazaki	(Supervisor)
Professor Takenobu Tokunaga	(Co-supervisor)
Professor Koichi Shinoda	(Co-supercisor)
Professor Jun Miyazaki	(Co-supervisor)
Associate Professor Nakamasa Inoue	(Co-supervisor)

Task-Oriented Word Segmentation *

Tatsuya Hiraoka

Abstract

Word segmentation or tokenization is a fundamental process in natural language processing (NLP). A sentence is split into small units such as words, subwords, or other tokens to process natural language on a computer for NLP tasks such as text classification. Because the downstream model is trained and evaluated with a tokenized sentence, the performance of the downstream model depends on the tokenization strategy. Therefore, exploring the proper tokenization method is a fundamental issue to improve NLP performance.

In general architectures of NLP, word segmentation or tokenization is considered a preprocessing task. In other words, sentences can be tokenized into tokens in advance of training the downstream model. This means that the tokenization strategy is not changed after preprocessing. However, recent studies have shown that the appropriate tokenization depends on the downstream task and model. This implies that a gap exists between tokenization as preprocessing and the training of the downstream model. In other words, the tokenization strategy must be determined in isolation from the downstream model, where the appropriate tokenization depends on the downstream task and the architecture of the downstream model. Determining the tokenization strategy without information about the downstream task and model is not recommended by this author, even though information can be accessed when choosing the tokenization strategy. To bridge this gap, a novel method is proposed herein to train both the tokenization module and downstream model simultaneously. In contrast to the conventional tokenization method in NLP, the proposed method improves the tokenization strategy during the training of the downstream model and enables the tokenization module to generate a more appropriate tokenization for the downstream model, thereby improving the performance of the model.

*Doctoral Dissertation, School of Computing
Tokyo Institute of Technology, February 24, 2022.

This study introduces two approaches to optimize tokenization. The first approach embeds the tokenization module into the architecture of the downstream model and exploits the sentence representation calculated in the downstream model to select better tokenization during the training of the model. This first approach is specialized to the downstream model using sentence vectors to solve a task such as text classification. The second approach exploits loss values of the downstream model calculated to optimize the tokenization module. This approach is applicable to various downstream models, as it uses only loss values for the update, and this implies that it can be used with various NLP tasks, including generation tasks such as machine translations. Both approaches employ neural networks for the tokenization module, known as a neural unigram language model, and the downstream model and tokenization module are trained simultaneously as combined neural networks.

This study evaluates the proposed method on two famous NLP tasks, namely, text classification and machine translation, on multiple languages. For text classification, sentiment analysis in Chinese, Japanese, and English is employed for a task using a single sentence for the input. The rating and genre prediction tasks are also exploited using reviews on E-commerce services in Chinese, Japanese, and English. In addition, natural language inference in English is employed for a task using multiple inputs. For machine translation, seven language pairs are used, where one side of the translation pair is English, and the other side uses German, Vietnamese, Chinese, Arabic, French, Hungarian, and Romanian. The experimental results demonstrate that the proposed method improves the performance of the downstream model by optimizing tokenization on both text classification and machine translation as compared with the conventional tokenization strategy. The experimental results also show that the proposed method improves the performance of the downstream task even when the already trained downstream model is used and its trainable parameters are frozen. These results demonstrate that the proposed method can improve the downstream performance only by finding more appropriate tokenization for the downstream model. The experimental results on text classification demonstrate that the proposed method can be applied to various downstream models such as classifiers with the self-attention mechanism, bi-directional long short-term memory (BiLSTM) encoders, and logistic regression. Finally, the results show that the proposed method are applicable to the downstream model, including BERT, which is a well-known large

pre-trained language model.

Analysis of the acquired tokenization by the proposed method shows that the optimized tokenization differs depending on the downstream task and model. For example, the proposed method acquires different tokenizations for different text classification tasks even when the input text is the same. This study also provides the observation that the number of tokens in the acquired tokenization differs depending on the downstream tasks and languages. For example, the number of tokens in the tokenization for text classification is much greater than that for a target side corpus of machine translation.

Keywords:

Word Segmentation, Tokenization, Language Model, Neural Network, Text Classification, Sentiment Analysis, Machine Translation

Acknowledgements

博士論文を執筆するにあたり，多くの方のお力添えをいただきました． この場を借りて感謝申し上げます．

主指導教員の岡崎直観教授には，博士課程への受け入れから博士論文の提出まで，研究のみならず私生活についても様々な相談をさせていただきました． 特にリサーチアシスタントとして雇用していただくことで経済的に安定した生活を送ることができ，無事三年間で卒業することが叶いました． 博士課程の入試面接で発表したスライドタイトルは，本論文と同じ“Task-Oriented Word Segmentation”でした．他大学からの進学にもかかわらず，こうして入学時点から一貫して同じテーマに取り組むことができたことは大変幸せで，岡崎先生には感謝してもしきれません．

徳永健伸教授には，おなじ自然言語処理分野のご専門として，私の研究テーマに深く関わる重要なコメントをいくつもいただきました． 特に，Soft-Tokenization（後段モデル側で複数の単語分割を考慮するような手法）に対して本研究をどのように位置づけたらよいかという点について，いち早くご指摘をくださりました． 本研究を進めるにあたり，このご指摘は大変参考になるものでした． また，本論文における6.5節は，予備審査で徳永先生にいただいたアイディアを元に行った実験です．

篠田浩一教授には，本研究の応用という観点から，多くのコメントを頂きました． 入試面接で本研究のアイディアを発表したときから，単語分割を今後どのように変えていくべきかという深い議論をさせていただき，研究の方向性を熟考するための材料をいくつもいただきました． 提案手法を様々なシーンで活用できるようにしたほうがよいという視点は，当初の提案手法を改善（4章）する強いモチベーションとなりました．

宮崎純教授には博士論文の審査に加えて，アカデミック・アドバイザーとして日頃の研究生活や私生活について気にかけていただきました． 博士課程から東京工業大学に異動して心細かった中，アカデミック・アドバイザーとの初めての面談でNAISTについてお話でき，非常にリラックスすることができたのを覚えています． 論文発表会では，単語分割を調整することで到達しうる最高性能との比較という重要な視点についてコメントをいただき，最終提出前に6.3節を追加することが出来ました．

金崎朝子准教授には，中間発表と予備審査で副査を担当していただき，本研究を位置づけるにあたって重要なアドバイスをいただきました． それまでUnsupervised

Word Segmentationの亜種として本研究を捉えていましたが，金崎先生に「Weak Surpervisionの一種なのではないか」とコメントを頂き，本研究を客観的に捉える一助となりました．

井上中順准教授には，論文発表会と最終審査で副査を担当していただきました．論文発表会直前の12月初旬に，突然副査の依頼をすることになってしまったにも関わらず，これを引き受けてくださり大変助かりました． 発表会では単語分割の語彙に関する鋭いご指摘をいただき，今後の研究の方向性を考える切っ掛けをくださいました．

高瀬翔助教には，博士課程でのすべての研究で深く関わっていただきました． 特に研究サイクルの回し方や論文執筆の作法，研究そのものの捉え方など，研究に対する様々な姿勢を学ばせていただきました． 高瀬さんの研究スタイルを模倣するには並外れた体力が必要ですが，今後も力をつけて今の高瀬さんと同じような勢いで研究を手がけられるように精進します． 研究室で最も雑談に付き合っていただいたのも高瀬さんでした． 研究に限らず生活や趣味，将来の話など様々な雑談に多くの時間を割いていただいたおかげで，孤独を感じずに博士課程生活を送れました． 今後も，たまに一緒に飲みに行っていただければ幸いです．

秘書・支援員の佐藤あゆみさん，中山都子さん，中川恵理子さん，小西由希子さん，雲財祐子さん，古谷奈緒子さんには，研究生活での事務処理面でとてもお世話になりました． 毎月のRAの勤怠申請から物品購入，出張申請など，自分ではできない煩雑な処理を請け負っていただき，その時間を研究に充てることができました． 特にD3の後半ではACT-Xの予算を使用するために様々なお手数をおかけいたしました． 私としては初めての研究費であり，わからないことが多くご迷惑をおかけすることばかりでした．

デンソーITラボラトリの内海慶さん，欅惇志さんには共同研究で貴重なアドバイスを頂きました． 博士課程での査読付きの業績はすべてデンソーITラボラトリのお二人との共同研究によるもので，お二人との出会いがなければ博士論文の執筆には至らなかったと感じております． 修士時代に内海さんの教師なし形態素解析の論文を拝見して以来，いつかお話できればと思っていましたが，まさか共同研究としてここまで密にお話できるとは思ってもいませんでした． また，欅さんには研究だけではなく，学振DCやACT-Xの申請についても相談させていただき，たくさんの助言をいただきました．

株式会社レトリバのみなさんには，2017年（修士1年生の夏）のインターン以来，長きにわたり大変お世話になりました． 博士課程ではパートタイムリサーチャーとして受け入れていただき，製品開発や企業での研究開発の一部に関わらせていただきました． この経験を通して，自分の自然言語処理に関する知識を社会でどのように活かしていきたいかを，じっくり考えることができました． 特に飯田大貴さんには，バイト先の上司・研究室の同僚という2つの立場から様々な相談に乗っていただ

きました．

岡崎研究室のみなさんには，定例のResearch Seminarでの発表やPaper Readingでの議論から日常生活での雑談に至るまで，様々な面でお世話になりました． 水木栄さんには，唯一の先輩として授業などの東工大の仕組みや社会人Dの苦労など，たくさんのお話を伺いました． I appreciate Mr. Sangwhan Moon for many conversations about research on NLP basics, such as tokenization. Also, he gave me many insightful stories of his life, including working and family. 丹羽彩奈さんは，最も年齢の近い同じフルタイム博士課程学生として，私生活や精神状態について多くの相談をさせてもらいました． 様々な挑戦機会に果敢に挑み，成果をもぎ取る丹羽さんの姿には関心しつつ，勇気をいただきました． Dの会の幹事をはじめ，様々なイベント運営にも積極的に参加しており，横の繋がりを広げることに注力するスタイルにはとても刺激を受けました． 金子正弘研究員には，学年の近い先輩として，博士論文の発表会や研究そのものへの心構えについて助言をいただきました． 舜夏海さん，馬尤咪さん，植木滉一郎さんとは一緒に研究をする機会を持たせていただき，共同で研究する楽しさや難しさを教えていただきました． メンターの一人として至らぬ点も多くあったかと思いますが，研究を嫌いになっていないでしょうか．研究という経験が皆さんの人生の糧となり，研究をしてよかったと思う瞬間が今後少しでもあれば幸いです． 馬尤咪さんには，投稿論文や本博士論文に含まれる中国語のネイティブチェックをしていただき，大変助かりました． また，サーバー係の皆さんには，研究室内の計算リソースの保守等で大変お世話になりました．COVID19の蔓延る世の中で，博士課程の後半ではほとんどがオンライン上でのやり取りになってしまいましたが，みなさんとの雑談や飲み会はとても楽しく，研究の種になる刺激を多くもらいました． 研究室環境で唯一気になっている事といえば，こたつスペースの治安が良すぎる点でしょうか． もうちょっとゲーム機とか楽器とか，寝っ転がってる人とか，謎の人形がおいてあるとか，治安が悪いほうが研究室としての愛着が湧いて良いかもしれません． またいつか岡崎研究室にお邪魔したときに，こたつスペースの進化が見られることを楽しみにしています．

徳永研究室の皆さんには，同じ自然言語処理を研究する仲間として多くの刺激をもらいました． 特に早稲田大学時代にも接点があった山田寛章研究員とは，東工大やACT-Xでも先輩として多くのお話をさせていただきました．

東京Dの会の皆様と研究の話で盛り上がることで，研究に対するモチベーションや研究シーズを多くいただきました． 私はD3で参加したのですが，もっと早く参加していればよかったと悔やまれます．

奈良先端科学技術大学院大学での修士課程時代に出会った方々には，博士課程でもお世話になりました． 博士論文のテーマであるタスクに応じた単語分割の最適化は，実は奈良で同期と勉強会をやった後にぐだぐだと話しながら生まれたテーマです． 本論文のコアアイディアとは異なりますが，本研究の最初期のアイディアにつ

いては和田崇史さんに実験を手伝ってもらいながら検証を行っていました． 残念ながら当時の手法では上手くいかなかったのですが，3年の時を経てなんとか本研究テーマを形にすることができました． その後もNAISTのみなさんには様々な雑談に付き合っていただき，研究テーマを育み本博士論文として結実するための土壌を提供していただきました． また学会などの機会に酒を飲みながら思い出話をしましょう． 野球とかBBQとかもしたいですね．

早稲田大学の学部生時代に出会った方々には，研究分野以外の外の世界を見せていただき，見識を広める一助となりました． また，久野正和教授のゼミで共に統語論を学んだ仲間である八木雄介さんには，博士課程においても言語学分野の知識を提供していただき，言語学分野の情報までを追いかけられない私の知識を助けていただきました． 自然言語処理と言語学で分野はすこし離れてしまいましたが，いつかお互いの知識を出し合い，共通する興味のあるテーマについて一緒に論文を書きましょう．

小学校，中学校，高校で出会った友人たちへ． 今でも変わらず話を聞いてくれてありがとうございます． 皆様とのつながりが私を地元に紐付け，前進する活力を与えてくれます． これからも変わらず仲良くしてください．

府中に住む父，母，妹，そしてお世話になっている親戚の皆様へ． 早稲田大学を卒業して就職せず，奈良に，それも異なる分野で進学するという選択を文句一つ言わず見守ってくれてありがとうございます． 早いもので五年が経ち，念願叶って博士号を取得するに至りました． この経験を糧に，今後も精進するのでぜひ見守っていてください．

Contents

List of Figures

List of Tables

1 Introduction

1.1 NLP and Tokenization

Natural language processing (NLP) is one of the core research fields that exploit massively collected text data, sometimes called big data, such as Wikipedia. Applications of NLP help people effectively use this type of large amounts of data and access required information efficiently. For example, text classification is exploited for fake news detection on social media [96, 87, 97]. Another popular example is machine translation that automatically converts a language into other languages [56, 95, 117, 27, 112]. Most NLP systems require a sequence of tokens as their input. For example, a word or subword that is a smaller unit than a word is used for the input of the systems.

Tokenization (or word segmentation[1]) is a fundamental problem in NLP. In the tokenization process, a given sequence of natural language is split into a sequence of tokens such as words. For example, the Japanese sentence "今日はいい天気ですね" can be tokenized into a sequence of words such as "今日/は/いい/天気/です/ね". This process is essential, particularly for languages that do not contain obvious boundaries (e.g., whitespaces) such as Japanese and Chinese. In addition, exploring appropriate tokenizations for languages containing obvious boundaries indicated by whitespaces, such as English [85, 86, 95, 36, 1, 5], also represents a better approach.

Tokenization is a critical process that affects the performance of NLP tasks because, in most cases, the raw natural language data must first be tokenized before being fed into an NLP system. The tokenized sequence of words varies

[1]Two similar terminologies describe the process that splits a raw sentence into a sequence of words or tokens. These are "word segmentation" and "tokenization." Although they are often confused, this dissertation uses "word segmentation" for the name of a task that splits a sequence into small pieces (and evaluates the tokenized sequence with human-annotated tokenization) and uses "tokenization" for the process itself and the tokenized sequence.

depending on the tokenization tool. For example, the aforementioned Japanese sentence can be tokenized in various ways, such as “今日は/いい天気/ですね”, “今日/は/いい/天気/で/す/ね”, and so on. Different tokenizations lead to different performances of downstream NLP tasks because the downstream model is trained with a different tokenization based on the tokenization tool.

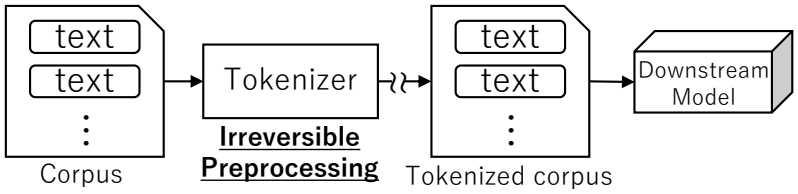
Interestingly, the tokenization varies even when a human annotates the word boundaries. For example, Sproat et al. [99] reported that the agreement of the tokenization annotation on the Chinese corpus was 76% among human annotators.

Existing studies have proposed various tokenization methods including rule-based word segmentation [82, 102, 57, 70, 24], dictionary-based word segmentation [58, 73, 109, 106], supervised word segmentation [84, 121, 75, 76, 68, 128, 129, 15, 122, 11, 123], and unsupervised word segmentation [107, 18, 29, 30, 72, 130, 113, 95, 60, 103, 53, 115]. Much of the prior research has reported that appropriate tokenization depends on each downstream task and model [120, 14, 78, 23, 39, 33]. In other words, the performance of a downstream model can be improved by determining the appropriate (task-oriented) tokenization for the downstream task and model.

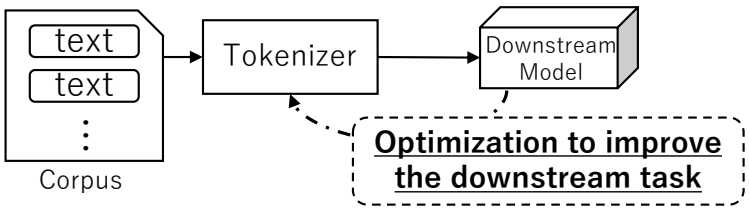
1.2 Problems of Conventional Tokenization

In traditional NLP, a given sentence is tokenized as part of a preprocessing task, as shown in Figure 1.1(a). Thus, an existing tokenizer is applied to the given sentence, and then the tokenized sentence is input into a model for a target downstream task (downstream model).

In the conventional pipelined approach, we obtain the most plausible tokenization deterministically based on the tokenizer (e.g., tokenization with the maximum scores predefined by a manually annotated dictionary). In other words, we can use only a single pattern of the tokenization for each sentence, whereas this would be inappropriate for the downstream task and model. If we want to find the appropriate tokenization for the downstream task and model, we must train a given downstream model with each possible tokenization and evaluate its performance to determine the appropriate tokenization. Performing this type of brute-force exploration whenever we construct a new downstream model is impractical.



(a) Conventional Irreversible Tokenization



(b) Optimizing Tokenization (proposed)

Figure 1.1: Overview of (a) conventional tokenization and (b) proposed optimizing tokenization. The tokenizer is directly optimized to improve the performance of the model for a downstream task using the loss of the target task.

Some studies have tackled this problem by varying the tokenization using a sampling strategy during the training phase to enable the downstream model to adapt to different tokenizations. This technique is called subword regularization [59], and previous studies have reported that it improves the performance of the downstream model on machine translation tasks [59, 90] and text classification tasks [39]. Although this type of strategy makes the downstream model robust against the gap between actually used tokenization and appropriate tokenization, little attention has been given to optimizing the tokenizers for a downstream task. Thus, if we acquire appropriate tokenization for a downstream task, we might improve the task performance.

By contrast, some studies have used multiple tokenized sentences to prevent the damage depending on the type of tokenization applied [16, 127, 123, 118, 65]. Their methods compute various tokenizations for a given sentence and then encode the tokenizations using an architecture based on long short-term memory [42] or Transformer [112]. Although their methods prevent error propagation from the tokenizer, they are intractable when handling all possible tokenizations due to computational costs.

Several studies have tackled the problem of identifying appropriate tokenization for downstream tasks, primarily machine translation tasks. Chang et al. [14] proposed a method to identify better tokenization for the corpus of a machine translation. Gowda and May [33] proposed a method to estimate the effectiveness of tokenization granularity on machine translation tasks. He et al. [37] introduced dynamic programming encoding (DPE) that attempts to identify the appropriate tokenization for a Transformer-based decoder on machine translation tasks. These methods identify the appropriate tokenization while considering the corpus of translation pairs, but they are isolated from a downstream model such as a neural encoder–decoder for machine translation.

1.3 Solutions

This study explores the problem of identifying an appropriate tokenization for downstream tasks and proposes a method to optimize a tokenizer based on the downstream task and model, as shown in Figure 1.1(b). In other words, the proposed method trains the tokenizer and downstream model simultaneously. This study introduces two approaches to optimize tokenization for downstream tasks depending on the loss value used in training the downstream model. Both approaches are based on the same idea: generate multiple tokenized sentences as candidates and input them into the downstream model; then, update the tokenizer to yield the appropriate tokenization for the downstream model. The two approaches are different in the manner in which multiple tokenized sentences are used.

The first approach weights the sentence vectors with the probabilities of their tokenization and inputs them into the downstream model. This approach then updates the parameters of the tokenizer to decrease the training loss such that the tokenizer outputs improved tokenization for the downstream task. However, this approach is limited by the architectures that use sentence vectors, such as in text classification tasks.

Unlike in the first approach, the second approach inputs multiple tokenizations into the downstream model separately. This approach weights the loss values calculated for each tokenization with the probabilities of each tokenization. This method then straightforwardly updates the tokenizer using the weighted sum of the loss values. Because this approach requires only loss values of the downstream

model, it is applicable to various tasks and models that use loss values in the training.

Both approaches can optimize the tokenizer to improve the performance by refining tokenization even if the given downstream model is already trained and the trainable parameters are frozen. This study calls this "refinement of tokenization" *postprocessing*. Thus, the proposed method can be easily applied to various situations, including cases in which the downstream model is sufficiently trained or it does not have trainable parameters.

This study conducted experiments on the two downstream tasks of text classification and machine translation. Experiments on text classification tasks exploit sentiment classification tasks on short-text social networking site (SNS) corpus, genre, and rating prediction tasks from review texts for products on E-commerce services and on the Stanford Natural Language Inference (SNLI). This study employs corpora in the three languages of Japanese, Chinese, and English. Both approaches of the proposed method can be used for text classification tasks. Experimental results show that the proposed method improves the performance of the downstream model by optimizing tokenization as compared to the baseline, which does not change the tokenization. Experimental results demonstrate that the proposed method can be applied to an architecture that uses a large pretrained language model. This study also shows that the proposed method can be combined with the state-of-the-art contextualized embeddings (i.e., BERT [22]) and improves the performance on text classification tasks in English.

Experiments on machine translation tasks exploit seven language pairs from the famous datasets of IWSLT and WMT. One side of the translation pairs is English and the other side includes German, Vietnamese, Chinese, Arabic, French, Hungarian, and Romanian. Unlike the first approach, the second approach of the proposed method is applicable to machine translation tasks. Experimental results show that the proposed method outperforms the baseline without optimizing tokenization in all datasets. In addition, the proposed method outperforms the existing method in terms of identifying appropriate tokenization for machine translation tasks [37].

This study also shows that the proposed method is applicable to a downstream model already trained as a postprocessing model. After the downstream models are trained on the downstream task, the proposed method is applied to refine the tokenization specialized for the model and task. Experimental results show that

the proposed method improves the performance even in postprocessing on both text classification and machine translation.

1.4 Contributions

This study provides the following contributions:

- Proposes two novel methods to optimize tokenization for both downstream tasks and downstream models.
- The first approach of the proposed method can optimize tokenization using sentence vectors corresponding to multiple tokenization candidates.
- The second approach extends the first approach and is applicable to various NLP tasks because of its simplicity.
- The proposed method is applicable to various downstream models even if it does not have any trainable parameters, such as in an already trained downstream model.
- Experimental results show that the proposed method improves the performances on both text classification and machine translation tasks.
- The obtained tokenizations by the proposed method are different depending on the characteristics of the downstream task and model.

1.5 Thesis Outline

The remaining sections of the thesis can be summarized as follows.

Chapter 2: Related Work and Preliminary

Chapter 2 overviews related studies on tokenization. The differences between the proposed method and those from relevant studies are described in detail. This chapter also provides basic knowledge about word segmentation and tokenization in NLP. Specifically, unsupervised word segmentation, which is most relevant to this study, is explained at length. In addition, this chapter introduces two

downstream tasks that are common in NLP and that are used in the experiments conducted in this study, namely, text classification and machine translation. This chapter also describes recent popular architectures used to solve these downstream tasks.

Chapter 3: OpTok: Optimizing Tokenization for Text Classification

This chapter introduces the first approach of the proposed method, which we call "optimizing tokenization" (*OpTok*). This approach is limited to tasks that use sentence vectors, such as text classification, but it includes the core idea (i.e., weighting elements calculated from tokenization candidates with the probabilities of each candidate) used in the second approach. This chapter also describes some techniques for stable training of the proposed method.

Chapter 4: OpTok4AT: Optimizing Tokenization for Various Tasks

This chapter introduces the second approach of the proposed method, which is based on the first approach and called OpTok for any task (*OpTok4AT*). The first approach is extended here to make it applicable to various NLP architectures. The core idea is the same as that of the first approach, and the extension is quite simple. This chapter explains the differences between the first and second approaches and describes the techniques for applying the proposed method to generation tasks such as machine translation.

Chapter 5: Experiments

Chapter 5 presents experiments conducted to confirm the performance of the proposed method. This study conducted experiments on text classification and machine translation tasks. Although both approaches can be used for text classification tasks, only the second approach is applicable to machine translation tasks.

Chapter 6: Discussion

Chapter 6 presents an analysis of the proposed method, including the effects of hyperparameters on the model's performance. Quantitative and qualitative analyses of actual tokenization obtained by the proposed methods are also discussed.

To confirm the behavior of the proposed method, this study conducted additional experiments using a simple downstream model that includes logistic regression. In addition, the chapter describes the technique for using the proposed method effectively on machine translation tasks. This chapter provides the observation on the experimetns under specific postprocessing settings on both tasks and under milti- and cross-task settings.

Chapter 7: Conclusion

This study on optimization of tokenization for downstream tasks is summarized in the final chapter. In addition, remaining problems and other issues, the study's limitations, and future directions of this research are all briefly discussed.

2 Related Work and Preliminary

This chapter reviews studies that have tackled the problem of tokenization. Specifically, the following Section 2.1 introduces two related studies that tackled unsupervised word segmentation and the problem of acquiring appropriate tokenizations.

The later part of this chapter introduces the basic knowledge to capture the core idea of the proposed method. This research is strongly related to existing studies on unsupervised word segmentation, where this study identifies appropriate tokenizations without any supervisory signal of tokenization itself. Inspired by subword regularization, the proposed method also uses a technique of stochastic tokenization for stable learning. This chapter explains the concepts of unsupervised tokenization and subword regularization and introduces two downstream tasks. This study is designed to improve downstream tasks by exploring appropriate types of tokenization based on the downstream task and model. Accordingly, this chapter presents sample text classification and machine translation tasks in NLP and explains the general architecture for these downstream tasks.

2.1 Related Work on Tokenization

2.1.1 Unsupervised Word Segmentation

This study is highly relevant to unsupervised word segmentation because, unlike in supervised word segmentation tasks, the proposed method does not use supervisory signals of tokenization. Instead, the proposed method trains the tokenization model with the supervisory signals of the downstream task. Thus, this setting more closely resembles an unsupervised rather than an supervised word segmentation task. The most well-known method of unsupervised word segmentation involves using a language model.

Deligne and Bimbot [21] introduced a language model for segmenting input sequence using a multigram language model. They trained the language model with an expectation–maximization (EM) algorithm using a forward-backward algorithm over the possible tokenization. Creutz and Lagus [18] introduced a method to find morphemes (subwords) in words using a language model-based algorithm. This method is known as Morfessor, and Virpioja et al. [113] extended this approach as Morfessor 2.0. Goldwater et al. [29, 30] proposed a Bayesian framework for unsupervised word segmentation using unigram and bigram language models. They estimated the parameters of the language model using Gibbs sampling only from the raw corpus without the supervisory signal of tokenization. Mochihashi et al. [72] followed this study and proposed a method that employs the nested Pitman–Yor process for the language model, which uses more long n-gram dependencies. They also exploited Gibbs sampling in estimating parameters. Uchiumi et al. [111] modified this architecture for joint unsupervised word segmentation and part-of-speech tagging. Zhikov et al. [130] proposed another approach using bidirectional character-level N-gram language models to detect word boundaries. Unsupervised word segmentation using a language model has received attention recently as a tokenization tool for neural networks. For example, Kudo and Richardson [60] developed SentencePiece composed of the unigram language model for tokenization to tokenize texts for the inputs of neural networks. SentencePiece estimates the language model using the EM algorithm.

Another means of unsupervised word segmentation is to use a data compression technique. Teahan et al. [107] proposed a method of unsupervised word segmentation for Chinese using prediction by partial matching (PPM). In recent NLP, a method using byte-pair encoding (BPE) [95] is now the most commonly used in this area.

WordPiece [32] is another well-known method of tokenization. This method tokenizes a sentence into words using a greedy longest-match-first strategy known as maximum matching [83]. Inspired by the Aho–Corasick algorithm [2], Song et al. [98] introduced a fast algorithm for WordPiece tokenization by organizing vocabulary using a trie [25] with cache.

Some researchers have recently tackled unsupervised word segmentation using a language model through neural networks. The most popular approach is a method using character-based language models [52, 53]. Sun and Deng [103] proposed a method using a long short-term memory (LSTM)-based character language model

known as a segmental language model. Wang et al. [115] extended this idea by using a bidirectional language model.

The current study is in this category of unsupervised word segmentation using a language model and includes neural networks in the design of the language model.

2.1.2 Studies Identifying Appropriate Tokenization

Numerous studies have improved NLP tasks in terms of tokenization. The main problem of conventional tokenization in NLP tasks is that only a single pattern of a tokenization is used based on the predefined tokenizer. Two well-known approaches to handle the tokenization problem on NLP tasks are introduced herein: 1) feeding the multiple tokenization patterns into the downstream model simultaneously, and (2) sampling tokenization candidates for each training epoch such that various tokenization patterns can be used for the training.

The first approach attempts to prevent segmentation errors by encoding multiple tokenizations jointly. Gong et al. [31] introduced multiple granularities of word segmentation tasks into Chinese corpora. Recent studies have investigated Lattice LSTM, which expands LSTM to enable multiple tokenizations to be taken as a lattice [16, 127, 123]. The calculation flow of LSTM cells was modified to consider directly the different granularities of the tokenization. Xiao et al. [118], Li et al. [65] followed this study by using a Transformer-based [112] architecture in which the attention mechanism was modified. Srinivasan et al. [100] modified the LSTM-based architecture for machine translation tasks to handle multiple granularities of tokenization.

The second approach makes the downstream model robust against tokenization errors by using sampled tokenization in each training step. This is a well-known technique called subword regularization [59]. The original study of subword regularization sampled tokenizations based on a unigram language model-based tokenizer called SentencePiece [60], which was trained on a training corpus using the EM algorithm. The aforementioned study demonstrated that training models with various tokenizations using subword regularization contributed to improved machine translation performance. Provilkov et al. [90] followed this approach by using subword regularization for BPE [95], which is widely used in NLP. Their method, known as BPE-Dropout, samples tokenization by randomly dropping

the merge operation in the BPE process. BPE-Dropout has a simple concept, but the experimental results demonstrated that it is competitive with the original subword regularization using SentencePiece. Hiraoka et al. [39] reported that subword regularization is useful for improving the performance of text classification tasks. They also proposed an additional technique of subword regularization that updates the language model for sampling tokenization during training. They scheduled a variety of sampled tokenizations by updating the language model so that a greater number of tokenizations could be used at the beginning of training and fewer at the end.

Optimization of tokenization has attracted attention mainly in the field of machine translation. For statistical machine translation, Nießen and Ney [80] and Goldwater and McClosky [28] attempted to obtain effective tokenization using handcrafted linguistic information. Some studies [120, 17, 78, 71] have attempted to optimize tokenization using certain criteria for machine translation, such as using a simple alignment model like the IBM model 1 [9]. Xiao et al. [119] directly tackled the problem of optimizing word segmentation for machine translation tasks by incorporating a tokenization module into the machine translation module using some handcrafted features.

Recent studies have also tackled this issue for generation tasks using neural networks. Gowda and May [33] analyzed the optimal granularity of tokenization on neural machine translation. Deguchi et al. [20] reported that using tokenizations composed of a similar number of tokens in the source and target sentences contributed to improved performance of machine translation, and Ho and Yvon [40] produced results supporting these findings.

Salesky et al. [93] developed incremental BPE, which automatically defines the number of BPE merge operations for neural machine translation. Their method iteratively trains and evaluates the downstream model for machine translation tasks with certain steps of the BPE merge operation; it then stops the operation based on the performance of the downstream model on the validation split. The aforementioned study inspired the proposed method in terms of using the loss values for the training of the tokenizer. He et al. [37] proposed a neural architecture to identify a better subword sequence of the target corpora in machine translation based on the tokenization of the source corpora by enhancing the study in Chan et al. [13]. They identified improved tokenization using a Transformer-based architecture that encodes characters to subwords conditioned

by the source tokenization. The current study differs from this research in that the proposed methods are applicable to various neural networks and optimizes the tokenization directly using only backpropagation from the training loss of the downstream tasks without any handcrafted criteria. In addition, the proposed method can optimize tokenization depending on the downstream model unlike previous research.

Some studies have shown that the multitask learning of word segmentation and other tasks help to improve NLP performance. In Japanese NLP, word segmentation is traditionally solved accompanied with part-of-speech tagging as the morphological analysis task [58, 76, 73, 109]. Recent work has reported that the joint learning of word segmentation, part-of-speech tagging, and lexical normalization improves normalization performance [38]. Peng and Dredze [86] used the example of a Chinese dataset to simultaneously train a model for named-entity recognition and word segmentation tasks. Two other studies demonstrated that the joint training of word segmentation and part-of-speech tagging tasks helped improve the performance of both tasks on a Vietnamese dataset [77] and Chinese dataset [108]. This study is different from studies on multitask learning that used supervisory signals for word segmentation. Specifically, this study optimizes tokenization only from information of the downstream task without using any supervisory signals about tokenization.

2.2 Preliminary of Unsupervised Word Segmentation

Because tokenization (or word segmentation) is a fundamental problem in NLP, several studies on tokenization have been conducted. Let $s = c_1...c_n...c_N$ be a sentence composed of N characters. This study defines the word "tokenization" or "word segmentation" as the process of converting s into a sequence of tokens $s' = w_1...w_m...w_M$, where M is the total number of tokens in s'. This study mainly uses "tokenization" to indicate this process and denotes the tokenized sequence s' by the word "tokenization." The term "word segmentation" is used to indicate the task of tokenization in which we train and evaluate a tokenization model.

One popular method of tokenization is the dictionary-based approach. Many

researchers have tackled the problem of controlling tokenization for the downstream task, as appropriate tokenization contributes to improved performance of the downstream task. In fact, as a feature selection process, the dictionary-based approach enables easy control of tokenization and is effective for use with NLP on formal text corpora such as newspapers. However, dictionary-based tokenization has the critical problems of portability and maintainability. To achieve appropriate tokenization for a downstream task, we must prepare a unique dictionary based on each task and language. Particularly for NLP with informal text such as short-text SNS, preparing the dictionary to include informal expressions and for low-resource languages is not easy. In addition, dictionary-based tokenization produces the problem of data sparseness in the training of the downstream model because the volume of the dictionary vocabulary becomes large and contains many low-frequency words.

In supervised word segmentation, a tokenizer with trainable parameters is trained using a manually annotated supervisory signal. Although this approach avoids the problem of maintaining the dictionary, creating the annotated data is still expensive.

To address these problems of dictionary-based and supervised tokenization, researchers have studied unsupervised tokenization methods that require neither the dictionary nor supervisory signal for tokenization. Although tokenization is not always appropriate for the downstream task[1], the unsupervised tokenization method can acquire vocabulary and tokenization automatically from a given raw corpora. In addition, because the popular methods of unsupervised tokenization are based on information theory and a language model, they can avoid using low-frequency words when performing tokenization. This characteristic contributes to addressing the problem of data sparseness. The following section introduces two well-known methods of unsupervised tokenization: one that uses BPE and another that employs a language model such as SentencePiece.

2.2.1 Byte Pair Encoding

Byte Pair Encoding (BPE) [95] is a well-known method used to perform unsupervised tokenization. BPE was initially developed as an algorithm for data

[1]Exploring the appropriate tokenization under unsupervised tokenization is the main focus of this research.

compression [26], but it can be used to tokenize natural language. Algorithm 1 outlines the process of BPE.

The tokenization process of BPE involves two iterative operations, namely, a counting operation and a merge operation. First, BPE splits the given corpus into a sequence of characters (Line 1) and then counts all token pairs (bigram) in the split corpus as the counting operation (Line 4). Second, BPE merges the most frequent token pair in the corpus as the merge operation (Line 5). Then, the counting operation of the next iteration runs for the corpus, including the merged pair. BPE continues this iteration until the specified maximum number of iterations or the total number of words in the vocabulary words reaches the specified value.

For example, given a raw sequence "abcabcdabb", BPE first splits the sequence into characters such as "a/b/c/a/b/c/d/a/b/b" and counts the number of pairs as "ab":3, "bc":2, and so on. BPE then merges the most frequent pair "ab" in the sequence as "ab/c/ab/c/d/ab/b" and counts the number of pairs again (e.g., "abc":2, "cab":1, and so on). BPE continues to count and merge the paired tokens iteratively.

The advantages of BPE are the simplicity of the algorithm, reproducibility, and controllability of the vocabulary volume. Because the merge operation is based on the frequency of the tokens, low-frequency tokens are represented as a sequence of short tokens called subwords, unlike dictionary-based tokenization that uses long and low-frequency tokens.

Algorithm 1 Algorithm for Byte Pair Encoding

1: Sequence of Characters: $s' = c_1...c_N$
2: $K \leftarrow$ Maximum Number of Marge Operation
3: **for** $k = 0$ to K **do**
4: $\quad A \leftarrow$ Frequency of Bigram in s'
5: $\quad w \leftarrow$ Most Frequent Bigram in A
6: $\quad s' \leftarrow$ Merge Bigram w in s'
7: **end for**
8: **return** s'

2.2.2 Tokenization with Language Model

Although BPE is a reasonable algorithm in terms of simplicity and portability, it has a problem in which the acquired tokenization often contains tokens that do not resemble a natural language. Because BPE merges the pairs of tokens deterministically, it cannot remove merged tokens from the vocabulary list even if it is unnatural for humans. For example, the word "preserve" includes two morphologies, "pre" and "serve", and the subword tokenization should be the same as the morphological boundary. However, once BPE merges the frequent subword "res" in this word, the word is tokenized as "p/res/e/r/v/e," and we cannot split "res" after this merge operation. In other words, we cannot use the tokenization "pre/serve" once merging of "res" has occurred.

One solution for this problem is an unsupervised tokenization method using a language model. The language model is a system for storing the probabilities of word occurrences in the corpus. For example, the probability of a word w is represented in a unigram language model by simply counting the frequency of the word as

$$p(w;\theta_D) = \frac{\mathrm{count}(w, D)}{\sum_{\hat{w}\in V} \mathrm{count}(\hat{w}, D)}, \tag{2.1}$$

where D and V are a corpus and vocabulary, respectively, $\mathrm{count}(w, D)$ indicates a function that returns the number of tokens w in the corpus D, θ denotes parameters of the unigram language model, and θ_D represents the parameters estimated over the corpus D. Using the unigram language model, we can calculate the probability of the tokenization s' by

$$p(s';\theta_D) = \prod_{w\in s'} p(w;\theta_D). \tag{2.2}$$

If we know the ground-truth probabilities of words $p(w;\theta_D)$ in the corpus, we can tokenize a sentence by finding the most plausible sequence: $\mathrm{argmax}_{s'} p(s';\theta_D)$. Although we can calculate the probabilities of all possible tokenizations s' to find the most plausible tokenization, researchers have employed a dynamic programming method called the Viterbi algorithm (see Algorithm 2) for the efficient calculation. In Algorithm 2, $\boldsymbol{a}$ represents an array that stores the best scores for each index and $\boldsymbol{b}$ stores the previous index for the backtrace. In addition,

$\boldsymbol{a}_i$ denotes an i-th element in the array a, and $s_{j:i}$ indicates a substring consisting of s that starts from a $j+1$-th character and ends with an i-th character, $s_{j:i} = c_{j+1}...c_i$.

However, in unsupervised tokenization, we cannot know the gold probabilities of words because the corpus D is not tokenized, and we cannot count the number of words in this type of untokenized corpus. Therefore, the probability of words $p(w; \theta_D)$ can be estimated using techniques such as Gibbs sampling and the EM algorithm.

Estimation using Gibbs Sampling

In Gibbs sampling, the parameters θ_D are estimated by iterative sampling with the tentative tokenization. Let $D'_k \ni s'_k$ be a set of sentences in $D \ni s$ tokenized with parameters $\theta_{D'_k}$ of the k-th iteration. In Gibbs sampling, the parameters $\theta_{D'_k}$ are defined as

$$\theta_{D'_k,w} = \text{count}(w, D'_k), \tag{2.3}$$

$$p(w; \theta_{D'_k}) = \frac{\theta_{D'_k,w}}{\sum_{\hat{w}} \theta_{D'_k,\hat{w}}}. \tag{2.4}$$

First, we initialize the vocabulary $V_{k=0}$ with a reasonable number of tokens (e.g., a vocabulary list containing all tokens that appear more than a specified number of times in the corpus). Second, we randomly tokenize the corpus $D'_{k=0}$ using the initialized vocabulary and set the initial parameters $\theta_{D'_{k=0}}$ by counting the number of tokens in the randomly tokenized corpus (e.g., $\theta_{D'_{k=0},w} = \text{count}(w, D'_{k=0})$). Following initialization, we iteratively select a sentence from the corpus and re-tokenize it by sampling the tokenization with a tentative parameters. The tentative language model $\theta^{(\backslash s'_{k-1})}_{D'_{k-1}}$ is thus created in which $\backslash s'_{k-1}$ indicates element exclusion of s'_{k-1} by removing several tokens consisting of previous tokenizations in the sentence $w \in s'_{k-1}$ from the parameter $\theta_{D'_{k-1}}$ as follows:

$$\theta^{(\backslash s'_{k-1})}_{D'_{k-1},w} = \theta_{D'_{k-1},w} - \text{count}(w, s'_{k-1}) \tag{2.5}$$

We then sample a new tokenization of the sentence s'_k from the tentative parameter $\theta^{(\backslash s'_{k-1})}_{D'_{k-1}}$ using the forward-filtering backward-sampling algorithm described in Section 2.3. Finally, we create the next parameter $\theta_{D'_k}$ by adding to the tentative

Algorithm 2 Viterbi Algorithm for Unigram Language Model

Sequence of Characters: $s = c_1 ... c_N$
Vocabulary: V
Maximum Length of Token: L
Parameters of Unigram Language Model: θ_D
$\boldsymbol{a} \leftarrow$ Zeros with Size of $N + 1$: $a_0, ..., a_N$
$\boldsymbol{b} \leftarrow$ Zeros with Size of $N + 1$: $b_0, ..., b_N$ (b_0 is not used.)
for $n = 1$ to N **do**
  for $l = 1$ to $\min(n, L)$ **do**
    $w \leftarrow s_{n-l:n}$
    if $w \in V$ **then**
      $t \leftarrow \boldsymbol{a}_{n-l} - \log p(w; \theta_D)$
      if $\boldsymbol{a}_n < t$ **then**
        $\boldsymbol{a}_n \leftarrow t$
        $\boldsymbol{b}_n \leftarrow n - l$
      end if
    end if
  end for
end for
$s' \leftarrow$ Empty List
$i \leftarrow N$
while $0 < i$ **do**
  $j \leftarrow \boldsymbol{b}_i$
  Add $s_{j:i}$ to s'
  $i \leftarrow j$
end while
$s' \leftarrow$ Reversed s'
return s'

parameter $\theta_{D'_{k-1}}^{(\setminus s'_{k-1})}$ the proper number of tokens that consist of the newly sampled tokenization of the sentence as follows:

$$\theta_{D'_k,w} = \theta_{D'_{k-1},w}^{(\setminus s'_{k-1})} + \text{count}(w, s'_k) \tag{2.6}$$

Through the process of sampling tokenization and recreating parameters iteratively, the parameters $\theta_{D'_k}$ become close to the ground truth, gradually reflecting the original corpus θ_D.

In addition to parameter estimation, we can construct a smaller vocabulary by truncating tokens during the iteration. Specifically, at each end of the iteration, we discard tokens whose probability is lower than a specified threshold (e.g., $p(w; \theta_{D'_k,w}) < \tau$). Reducing the size of the vocabulary using this technique is essential for unsupervised tokenization because the initial vocabulary often contains numerous tokens. Algorithm 3 describes this process using Gibbs sampling.

Estimation with EM Algorithm

Another means of estimating the parameters θ_D is to use the EM algorithm, which updates parameters with a likelihood over the current parameters [21]. For the EM algorithm, the parameters at the k-th iteration are denoted as $\theta_D^{(k)}$, and $\theta_D^{(k)}$ is defined as

$$\theta_{D,w}^{(k)} = p(w), \tag{2.7}$$

$$\tag{2.8}$$

where $\sum_{w \in D} \theta_{D,w}^{(k)} = 1$. Note that $p(w)$ itself is estimated as a parameter, unlike in the Gibbs sampling previously described. We then calculate the conditional likelihood of tokenization for a sentence in the corpus as

$$\mathscr{L}(s', s; \theta_D^{(k)}) = \prod_{w \in s'} \theta_{D,w}^{(k)}, \tag{2.9}$$

$$\mathscr{L}(s; \theta_D^{(k)}) = \sum_{s' \in S(s)} \mathscr{L}(s', s; \theta_D^{(k)}), \tag{2.10}$$

$$\mathscr{L}(s'|s; \theta_D^{(k)}) = \frac{\mathscr{L}(s', s; \theta_D^{(k)})}{\mathscr{L}(s; \theta_D^{(k)})}, \tag{2.11}$$

where $S(s)$ denotes a function returning possible tokenization candidates of s. The auxiliary function $Q(k, k+1)$ for the updated version of the EM algorithm

Algorithm 3 Estimation of Unigram Langauge Model with Gibbs Sampling

$\tau \leftarrow$ Threshold for Truncating Tokens
$D \leftarrow$ Raw Corpus
$k \leftarrow 0$
Initialize V_k with Reasonable Size
$D'_k \leftarrow$ Empty List for Tokenized Sentences
$\theta_{D'_k} \leftarrow$ Zero Parameters for k-th Iteration
for $s \in D$ **do**
    $s'_k \leftarrow$ Random Tokenization of s under V_k
    Add s'_k to D'_k
    for $w \in s'_k$ **do**
        $\theta_{D'_k,w} \leftarrow \theta_{D'_k,w} + \mathrm{count}(w, s'_k)$
    end for
end for
$K \leftarrow$ Maximum Iteration of Update
for $k = 1$ to K **do**
    $s'_{k-1} \leftarrow$ Select Tokenized Sentence from D'_{k-1}
    $\theta^{(\backslash s'_{k-1})}_{D'_{k-1}} \leftarrow \theta_{D'_{k-1}}$
    for $w \in s'_{k-1}$ **do**
        $\theta^{(\backslash s'_{k-1})}_{D'_{k-1}} \leftarrow \theta^{(\backslash s'_{k-1})}_{D'_{k-1},w} - \mathrm{count}(w, s'_{k-1})$
    end for
    $s'_k \leftarrow$ Sampling Tokenization of s using $\theta^{(\backslash s'_{k-1})}_{D'_{k-1}}$
    $\theta_{D'_k} \leftarrow \theta^{(\backslash s'_{k-1})}_{D'_{k-1}}$
    for $w \in s'_k$ **do**
        $\theta_{D'_k} \leftarrow \theta^{(\backslash s'_{k-1})}_{D'_{k-1},w} + \mathrm{count}(w, s'_k)$
    end for
    for $w \in V_{k-1}$ **do**
        if $\tau < p(w; \theta_{D'_k})$ **then**
            Add w to V_k
        end if
    end for
end for

over the entire corpus from the iteration k to $k+1$ is represented as

$$Q(k, k+1) = \sum_{s \in D} \sum_{s' \in S(s)} \mathscr{L}(s'|s, \theta_D^{(k)}) \log \mathscr{L}(s', s; \theta_D^{(k+1)}). \tag{2.12}$$

The parameter of the word $\theta_{D,w}^{(k+1)}$ can be re-estimated at the iteration $k+1$ by maximizing $Q(k, k+1)$ over $\theta_D^{(k+1)}$ as follows:

$$\theta_{D,w}^{(k+1)} = \frac{\sum_{s \in D} \sum_{s' \in S(s)} \text{count}(w, s') \mathscr{L}(s'|s; \theta_D^{(k)})}{\sum_{s \in D} \sum_{s' \in S(s)} \text{count}(s') \mathscr{L}(s'|s; \theta_D^{(k)})}, \tag{2.13}$$

where $\text{count}(s')$ returns a total number of tokens in s'.

Although we can re-estimate the parameters using (2.13), the explicit calculation of all possible tokenizations $s' \in S(s)$ requires considerable time and computational resources. Therefore, researchers have employed the forward-backward algorithm [101] to calculate the likelihood of words over possible tokenizations. Let $\alpha_{n,l}$ be a forward variable corresponding to a l character-length word ending at the n-th character in the sentence. We can calculate $\alpha_{n,l}$ recursively from the beginning of the sentence by

$$\alpha_n = \sum_{l=1}^{L} \alpha_{n,l}, \tag{2.14}$$

$$\alpha_{n,l} = \alpha_{n-l} p(s_{n-l+1:n}; \theta_D), \tag{2.15}$$

where L is the maximum length of tokens in the vocabulary and may be specified as a hyperparameter. Note that $\alpha_0 = 1$. Algorithm 4 summarizes the calculation of the forward variable for s.

The backward variable β_n corresponding to the likelihood of tokenizations on the last $N-n$ characters in the sentence can also be defined, where N denotes the number of characters that comprise the entire sentence. Similar to the forward variable, we can also calculate the backward variable recursively from the end of the sentence:

$$\beta_n = \mathscr{L}(s_{n+1:T}; \theta_D) \tag{2.16}$$

$$= \sum_{l=1}^{L} p(s_{n+1:n+l}; \theta_D) \beta_{n+l}, \tag{2.17}$$

where $\beta_N = 1$. Algorithm 5 illustrates the calculation of the backward variable for s.

Using the forward variable α, we can define the third variable γ_n corresponding to the average number of tokens in the tokenization of a part of the sentence $s_{1:n}$ as

$$\gamma_n = 1 + \sum_{l=1}^{L} \gamma_{n-l} \frac{\alpha_{n,l}}{\alpha_n}, \tag{2.18}$$

where $\gamma_0 = 0$. Algorithm 6 outlines this calculation.

Through dynamic programming of the forward-backward calculation, (2.13) can be rewritten with $\alpha^{(k)}$, $\beta^{(k)}$, and $\gamma^{(k)}$ can be calculated with the parameter at the iteration $\theta^{(k)}$, thereby avoiding explicit calculations of all possible tokenizations.

$$\theta_{D,w}^{(k+1)} = \frac{1}{|D|} \sum_{s \in D} \frac{\sum_{n=1}^{N} \sum_{l=1}^{L} \alpha_{n,l}^{(k)} \beta_n^{(k)} \mathbb{1}(s_{n-l+1:n} = w)}{\beta_0^{(k)} \gamma_N^{(k)}}, \tag{2.19}$$

$$\mathbb{1}(s_{n-l+1:n} = w) = \begin{cases} 1 & \text{if } s_{n-l+1:n} = w \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

Algorithm 7 describes this process. To train the parameters with the EM algorithm, we can also truncate tokens whose probability is lower than the threshold as well as the training using Gibbs sampling (Section 2.2.2).

SentencePiece [60] has recently proven to be a popular system that uses the unigram language model for tokenization. SentencePiece iteratively estimates the parameter of the language model using the EM algorithm and reduces the size of the vocabulary until it reaches the specified value.

2.2.3 Tokenization Differences by Methods

This subsection demonstrates differences in tokenization by the different methods. A tokenization model of BPE[2] and SentencePiece[3] are trained on a raw English corpus of "Alice's Adventures in Wonderland"[4]. The entire content of the novel were used to produce the tokenization model and tokenized the first two sentences using the trained models.

[2] `https://github.com/VKCOM/YouTokenToMe`
[3] `https://github.com/google/sentencepiece`
[4] `https://www.gutenberg.org/ebooks/11`

Algorithm 4 Forward Calculation with Unigram Language Model

Require: Sequence of Characters: $s = c_1...c_N$
Require: Parameters of Unigram Language Model: θ_D
$L \leftarrow$ Maximum Length of Tokens in Vocabulary
$\alpha \leftarrow (N+1) \times (L+1)$ Array Filled with Zeros ($\alpha_{0,1:}$ and $\alpha_{1:,0}$ are not used.)
$\alpha_{0,0} \leftarrow 1$
for $n = 1$ to N **do**
  for $l = 1$ to $\min(L, N - n + 1)$ **do**
    $w \leftarrow s_{n-l:n}$
    if $0 < p(w; \theta_D)$ **then**
      $\alpha_{n,l} \leftarrow \alpha_{n-l} p(w; \theta_D)$
    end if
  end for
end for
return α

Algorithm 5 Backward Calculation with Unigram Language Model

Require: Sequence of Characters: $s = c_1...c_N$
Require: Parameters of Unigram Language Model: θ_D
$\beta \leftarrow (N + 1)$ Array Filled with Zeros
$\beta_N \leftarrow 1$
for $n = N - 1$ to 0 **do**
  $b \leftarrow 0$
  for $l = 1$ to $\min(L, N - n)$ **do**
    $w \leftarrow s_{n:n+l}$
    if $0 < p(w; \theta_D)$ **then**
      $b \leftarrow b + p(w; \theta_D)\beta_{n+l}$
    end if
    $\beta_n \leftarrow b$
  end for
end for
return β

Algorithm 6 Calculation of Gamma with Forward Variable

Require: Forward Variable: α

$\gamma \leftarrow N + 1$-Sized Array with Zeros
for $n = 1$ to N **do**
    $g \leftarrow 0$
    for $l = 1$ to L **do**
        $g \leftarrow g + \gamma_{n-l} \frac{\alpha_{n,l}}{\alpha_n}$
    end for
    $\gamma_n \leftarrow 1 + g$
end for
return γ

Table 2.1 lists differences in tokenization used for the corpus. The tokenization by SentencePiece can correctly split punctuations because it relies on the language model, whereas BPE cannot split suffixes such as “ing,” in 6 lines. Another example is the tokenization of “making” in the second sentence. BPE splits this word into “ma” and “king”, whereas SentencePiece divides the word into “mak” and “ing”. The tokenization of SentencePiece is morphologically correct because it includes the suffix “-ing”. By contrast, BPE merges “king” in “making” because “king” frequently appears in the novel. BPE cannot change the merged tokens after it merges the two tokens depending on the frequency. These differences in tokenization affect the performances of the different NLP architectures. For example, Bostrom and Durrett [6] showed that a system using SentencePiece is superior to one using BPE for some English NLP tasks.

Algorithm 7 Estimation of Unigram Langauge Model with EM Algorithm

Training Data: $D \ni s$
Token Truncation Threshold: τ
$K \leftarrow$ Maximum Number of Update Iteration
$k \leftarrow 0$
$V_k \leftarrow$ Initialize Vocabulary with Reasonable Size of Tokens
$\theta_D^{(k)} \leftarrow$ Randomly Initialize Parameters
for $k = 1$ to K **do**
  $\hat{\theta} \leftarrow$ Zeros
  for $s \in D$ **do**
    $\alpha \leftarrow \text{Forward}(s, \theta_D^{(k)})$
    $\beta \leftarrow \text{Backward}(s, \theta_D^{(k)})$
    $\gamma \leftarrow \text{CalcGamma}(\alpha)$
    for $w \in V_k$ **do**
      $u \leftarrow 0$
      for $n = 1$ to N **do**
        for $l = 1$ to $\min(L, N - n)$ **do**
          if $s_{n-l+1,n} = w$ **then**
            $u \leftarrow u + \alpha_{n,l}\beta_n$
          end if
        end for
        $u \leftarrow \frac{u}{\beta_0 \gamma_N}$
      end for
      $\hat{\theta_w} \leftarrow \hat{\theta_w} + u$
    end for
  end for
  $\hat{\theta} \leftarrow \frac{\hat{\theta}}{|D|}$
  $\theta_D^{(k)} \leftarrow \hat{\theta}$
  for $w \in V_{k-1}$ **do**
    if $\tau < p(w; \theta_D^{(k)})$ **then**
      Add w to V_k
    end if
  end for
end for
return $\theta_D^{(K)}$

BPE	SentencePiece
Alice was beginning to get very ti red of sit ting by her **s is ter** on the **b an k** , and of **ha ving** nothing to do : once or **tw ice** she had **p eep** ed **into** the **b ook** her **s is ter** was read **ing,** but it had no **pi ct ures** or **con ver sation** s in **it, "and** what is the use of a **b ook ,"** thought Alice **"w it h out pi ct ures** or **con ver sation** s ?"	Alice was beginning to get very ti red of sit ting by her **sister** on the **bank** , and of **hav ing** nothing to do : once or **twice** she had **p e e p** ed **in to** the **book** her **sister** was read **ing ,** but it had no **picture s** or **conversation** s in **it , " and** what is the use of a **book , "** thought Alice **" w it ho u t picture s** or **conversation** s ?"
So she was **con s id er** ing in her **o w n m ind** (as well as she could , for the **h ot** day made her **fee l** very **s leep** y and **st up id**) , **w he** ther the **pleas ure** of **ma king** a d a is y **c ha** in would be **w ort h** the **tr oub le** of getting up and **pi ck** ing the d a is **i es,** when **sudden ly** a **White** Rabbit with p ink eyes ran close by **her.**	So she was **consider** ing in her **own mind** (as well as she could , for the **hot** day made her **feel** very **sleep** y and **st u p id**) , **whe ther** the **plea s ur e** of **mak ing** a d a is y **ch a** in would be **wor th** the **trouble** of getting up and **pick** ing the d a is **ies ,** when **suddenly** a **Whi te** Rabbit with p ink eyes ran close by **her .**

Table 2.1: The first two sentences of "*Alice's Adventures in Wonderland*" [12] tokenized by BPE and SentencePiece. I created the tokenization model whose vocabulary size was 1,000 for both methods. The bold font highlights the differences in tokenization between the methods.

2.3 Preliminary of Subword Regularization

When inputting natural language into an NLP architecture, we do not use the original surfaces of words such as those in the phrase "colorless green ideas" but use a sequence of vocabulary identifications (IDs) corresponding to each word in the phrase (e.g., "819 3498 1204"). Because vocabulary IDs are assigned systematically regardless of the surfaces of words, the IDs of words are entirely different even if some words are apparently similar. For example, although the word "ideas" and the substring "idea" have a similar surface, they have unrelated IDs (e.g., "1204" and "596") in most cases.

Because of these differences in NLP architectures, even a small difference in tokenization results in a completely different sequence of vocabulary IDs, as shown in Table 2.2. The table shows that we can express a sentence using various sequences of vocabulary IDs, but this is ambiguous for input into the NLP architectures. This ambiguity produces a gap between the training and inference of the NLP architecture, and this causes an inefficient inference. For instance, an architecture trained with the word "ideas" cannot handle an evaluation sequence that includes the word "idea" even though the system knows the similar word "ideas." This is because the two tokens have completely different vocabulary IDs.

Kudo [59] proposed subword regularization to address this problem, where the ambiguity is exploited for regularization to make the NLP architecture robust against differences in tokenization. Using subword regularization, we train the NLP architecture with various tokenizations sampled for each training epoch. In other words, we aim to maximize the marginal likelihood of the prediction by the architecture through training with subword regularization:

$$\mathscr{L}_{\text{marginal}}(\theta) = \sum_{s \in D} \mathbb{E}_{s' \sim p(s'|s)}[\log p(t|s';\theta)], \tag{2.21}$$

where θ denotes parameters of the NLP architecture and t is a supervisory signal corresponding to the input s.

To sample the tokenization from $p(s'|s)$, Kudo [59] originally proposed a method employing the unigram language model for unsupervised tokenization of SentencePiece. Provilkov et al. [90] followed this research and proposed a method of subword regularization for BPE tokenization called BPE-Dropout. This section describes BPE-Dropout and the original method of SentencePiece for subword regularization.

Tokenization	Vocabulary ID Sequence
colorless / green / ideas	819 3498 1204
color / less / green / idea / s	934 14851 3498 596 43
col / or / less / gre / en / id / ea / s	772 124 14851 501 30 4194 3332 43

Table 2.2: Differences in vocabulary IDs by tokenization of "colorless green ideas".

2.3.1 BPE-Dropout

When sampling a tokenization from the distribution $p(s'|s)$ in the equation (2.21), we need a probability of tokens consisting of s' (i.e., $p(w; D)$ over the corpus D). However, the unsupervised tokenization process of BPE does not use any language model that has word probabilities because BPE tokenization consists of simple operations, namely, the counting and merging of tokens, as described in Section 2.2.1. Therefore, Provilkov et al. [90] proposed a method called BPE-Dropout to sample tokenization by modifying the merging process of BPE tokenization.

BPE-Dropout samples a candidate of tokenization by randomly skipping the merge operation in BPE tokenization using a dropout rate p_{dropout}. Algorithm 8 describes the tokenization process with BPE-Dropout. Compared to the original process of BPE as shown in Algorithm 1, the only difference is the dropout part of the merge operation (lines 6 to 11). We can control the granularity of tokenization by changing the dropout rate p_{dropout}. The higher dropout rate p_{dropout} leads to more fine-grained granularity with tiny tokens because most merging operations are skipped. In addition, BPE-Dropout with $p_{\text{dropout}} = 1.0$ yields a sequence of characters, and $p_{\text{dropout}} = 0.0$ is exactly the same tokenization as in the original BPE process.

2.3.2 Subword Regularization with Language Model

Kudo [59] originally proposed subword regularization using a unsupervised tokenization method called SentencePiece. In contrast to BPE-Dropout, $p(w; D)$ can be calculated using the language model θ_D for the tokenization process. With this language model, a single tokenization from the distribution can be sampled:

$$p(s'_i|s) \simeq \frac{p(s'_i; D)^a}{\sum_{k=1}^{K} p(s'_k; D)^a}, \tag{2.22}$$

Algorithm 8 Algorithm for BPE-Dropout

Sequence of Characters: $s' = c_1...c_N$
$p_{\text{dropout}} \leftarrow$ Dropout Rate
$K \leftarrow$ Maximum Number of Marge Operation
for $k = 0$ to K **do**
  $A \leftarrow$ Frequency of Bigram in s'
  for $a \in A$ **do**
    $p \leftarrow$ Sample Value $[0, 1]$ from Uniform Distribution
    if $p < p_{\text{dropout}}$ **then**
      Remove a from A
    end if
  end for
  $w \leftarrow$ Most Frequent Bigram in A
  $s' \leftarrow$ Merge Bigram w in s'
end for
return s'

where s'_i is the i-th most plausible tokenization of s ($1 \leq i \leq K$), and K is a hyperparameter to limit the number of tokenized candidates. In other words, we sample tokenization from K-best candidates of tokenization. a is also a hyperparameter that controls the smoothness of the distribution for the sampling. A smaller a makes the distribution more uniform such that we can sample a greater variety of tokenizations, whereas a larger a causes the distribution to peak more often, from which we can sample the most plausible tokenizations more frequently.

We can find the K-best candidates of tokenization using forward filtering-backward A* algorithm [74]. In this algorithm, we first calculate the forward variable α for the possible tokenization, as described in (2.15). We then apply A*-search for α from the end of the sentence. Algorithm 9 overviews the backward A*-search for the K-best tokenization.

We can theoretically specify $K = \infty$ for the sampling of tokenization. However, because the number of possible tokenizations of the sentence increases to $O(2^N)$, calculating all possible paths in the backward A*-search consumes an impractical amount of time. Therefore, Kudo [59] employs forward filtering-backward sampling [94] to sample one tokenization from all possible candidates directly. We sample a token from the end of the sentence using the forward variable α

Algorithm 9 Forward Filtering-Backward A* Algorithm

Sequence of Characters: $s = c_1 ... c_N$
Parameters of Unigram Language Model: θ_D
$\alpha \leftarrow \text{Forward}(s, \theta_D)$
$S \leftarrow$ Empty List to Store Tokenization
$n \leftarrow N$
$Q \leftarrow$ Empty List to Store Scores
$R \leftarrow$ Empty List for Backtrace
$i \leftarrow i$
$Q_i \leftarrow 1$
$R_i \leftarrow [N]$
while $|S| < K$ **do**
    $j \leftarrow \text{argmax}_j(Q_j)$
    for $l = 1$ to L **do**
        $i \leftarrow i + 1$
        $Q_i \leftarrow \alpha_{H_j - l, l} Q_j$
        $R_i \leftarrow H_j$
        Add $H_j - l$ to R_i
        if $R_i = 0$ **then**
            $s' \leftarrow$ Empty List
            for $k = |R_i|$ to 1 **do**
                Add $s_{R_i : R_{i-1}}$ to s'
            end for
            Add s' to S
            Remove Q_i and R_i
        end if
    end for
end while
return S

in forward filtering-backward sampling while we apply A*-search to the K-best tokenization for α. Specifically, we sample a token $s_{n-l:n}$, which starts from the $n-l+1$-th character and ends at the n-th character, depending on the forward probabilities of tokens ending with the n-th index based on α:

$$w \sim p(s_{n-l:n}|s_{0:n}), \tag{2.23}$$

$$p(s_{n-l:n}|s_{0:n}) = \frac{\alpha_{n,l}}{\alpha_n}. \tag{2.24}$$

We can sample tokens from the end of the sentence recursively using this equation. Algorithm 10 presents an overview of the sampling process of tokenization for s.

Algorithm 10 Forward Filtering-Backward Sampling

Sequence of Characters: $s = c_1...c_N$
Parameters of Unigram Language Model: θ_D
$\alpha \leftarrow \text{Forward}(s, \theta_D)$
$n \leftarrow N$
while $0 < n$ **do**
    Sample $s_{n-l:n}$ from $\frac{\alpha_{n,l}}{\alpha_n}$
    $w \leftarrow s_{n-l:n}$
    Add w to s'
    $n \leftarrow n - l$
end while
Reverse s'
return s'

2.4 Preliminary of Downstream Task

The goal of this research is to acquire appropriate tokenization depending on the downstream tasks. Accordingly, this study employs two major tasks in NLP: text classification and machine translation. This section introduces these tasks and the general architecture of the neural networks used.

2.4.1 Text Classification

Task Overview

Text classification is a popular task in NLP, which requires that the downstream model predict the correct label t assigned to a sentence s. For example, in sentiment analysis, the model must predict a sentiment label (e.g., positive or negative) corresponding to the input sentence.

Formally, we train the downstream model for text classification θ_{TC} using training data $D \ni (s, t)$ to maximize the probability of determining the correct label:

$$p(t|s; \theta_{\mathrm{TC}}) = f(t|s'; \theta_{\mathrm{TC}}), \tag{2.25}$$

where s' is a tokenization of s, and f is a classifier that uses the parameter θ_{TC}.

We train the parameters θ_{TC} to minimize the cross-entropy loss $\mathscr{L}$ against the correct label:

$$\mathscr{L}(\theta_{\mathrm{TC}}) = \sum_{(s,t)\in D} \log p(t|s; \theta_{\mathrm{TC}}). \tag{2.26}$$

Generally, the classifier with neural networks f is composed of three modules: a sentence encoder that converts the tokenized sentence into a sentence vector, a multi-layer perceptron (MLP) that converts the sentence vector to a label-sized vector, and a softmax function for the label-sized vector. Specifically, the classifier f is extended as follows:

$$f(t|s'; \theta_{\mathrm{TC}}) = \mathrm{softmax}(\mathrm{MLP}(g(s', \theta_{\mathrm{TC}}^{(\mathrm{enc})}), \theta^{(\mathrm{MLP})_{\mathrm{TC}}}))_t, \tag{2.27}$$

where $(\cdot)_t$ denotes an operation that extracts an element corresponding to the label t. $\theta_{\mathrm{TC}}^{(\mathrm{enc})}, \theta_{\mathrm{TC}}^{(\mathrm{MLP})}$ are parameters for the sentence encoder and MLP, respectively, and they compose θ_{TC}. This study employs three general architectures for the sentence encoder g: an attention-based encoder [46], a BiLSTM-based

encoder [131, 50], and the large pre-trained language model-based encoder using BERT [22]. For simplification, the parameters (i.e., $\theta_{\mathrm{TC}}, \theta_{\mathrm{TC}}^{(\mathrm{enc})}, \theta_{\mathrm{TC}}^{(\mathrm{MLP})}$) are omitted from the equations in the following description of sentence encoders.

Neural Classifier with Attention Mechanism

One simple means of obtaining the sentence vector from a tokenized sentence is the bag-of-words method [35]. Let $\boldsymbol{v}_w$ be a word embedding corresponding to a word w. We calculate a vector of the tokenized sentence $\boldsymbol{h}_{s'}$ with bag-of-words as follows:

$$\boldsymbol{h}_{s'} = \frac{1}{|s'|} \sum_{i=1}^{|s'|} \boldsymbol{v}_{w_i}, \tag{2.28}$$

where w_i is an i-th word in the tokenized sentence s'[5].

Although the sentence vector calculated with bag-of-words is very simple, a problem exists in that all tokens in the sentence have the same amount of information, as bag-of-words simply averages the word embeddings[6]. To avoid this problem, instead of simple averaging, Iyyer et al. [46] used a weighted sum of the word embeddings as follows:

$$a_{w_i} = \mathrm{softmax}(\mathrm{MLP}^{(\mathrm{attn})}([w_1 ... w_i ... w_{|s'|}]))_i, \tag{2.29}$$

$$\boldsymbol{h}_{s'} = \tanh(\mathrm{MLP}^{(\mathrm{enc})}(\sum_{i=1}^{|s'|} a_{w_i} \boldsymbol{v}_{w_i})), \tag{2.30}$$

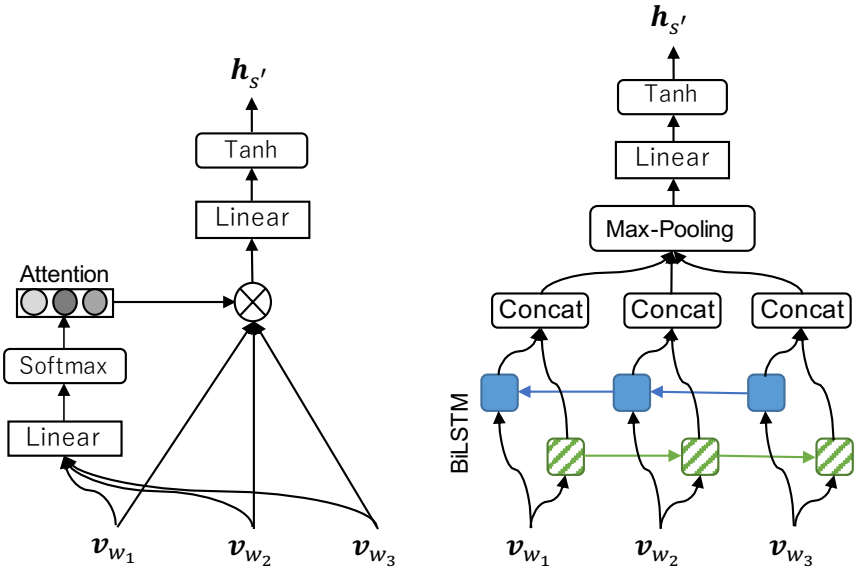
where $\mathrm{MLP}^{\mathrm{attn}}(\cdot)$ outputs scores for the attention. $\tanh(\cdot)$ and $\mathrm{MLP}^{(\mathrm{enc})}(\cdot)$ are a hyperbolic tangent activation function and MLP to encode the weighted sum of embeddings. Finally, we use the sentence vector as $g(s') = \boldsymbol{h}_{s'}$ in (2.27). Figure 2.1a presents an overview of the aforementioned calculation of the sentence vector.

Neural Classifier with BiLSTM

The classifier with the attention-based encoder is simple and effective because it exploits word embeddings to calculate the sentence vector. However, it cannot use

[5]Strictly speaking, we must use one-hot vectors as $\boldsymbol{v}_w$ such that the element corresponding to w is 1 and is 0 for the others when referring to this architecture as bag-of-words.

[6]There is a variation that we simply sum up the vectors without averaging.



(a) Attention Encoder (b) BiLSTM Encoder

Figure 2.1: Two types of encoders using neural networks (attention mechanism and BiLSTM). Each figure shows a calculation of a sentence representation $\boldsymbol{h}_{s'}$ using a sequence of word embeddings $\boldsymbol{v}_{w_1}, \boldsymbol{v}_{w_2}, \boldsymbol{v}_{w_3}$ corresponding to a sequence of words w_1, w_2, w_3 in a tokenized sentence s'.

word-order information because the sentence vector is calculated as a weighted sum of the word embeddings. In other words, this type of simple architecture with averaging or weighted sum of word embeddings cannot distinguish between sentences that share the same words, such as in the following sentences.

(a) Mary hits John

(b) John hits Mary

Here, the sentence representation of (a) and (b) is the same (e.g., $a_{\text{Mary}}\boldsymbol{v}_{\text{Mary}} + a_{\text{hits}}\boldsymbol{v}_{\text{hits}} + a_{\text{John}}\boldsymbol{v}_{\text{John}} = a_{\text{John}}\boldsymbol{v}_{\text{John}} + a_{\text{hits}}\boldsymbol{v}_{\text{hits}} + a_{\text{Mary}}\boldsymbol{v}_{\text{Mary}}$). This causes a problem in a text classification task that requires the model to capture the semantic meaning of sentences.

To address this problem, many researchers have used LSTM [42] for the encoder. LSTM recursively encodes a sequence of words with an architecture com-

posed of three gates, namely, forget, input, and output. The recursive calculation of LSTM has five hidden vectors corresponding to each time step n: vectors for the forget gate $\boldsymbol{f}_n$, an input gate $\boldsymbol{i}_n$, an output gate $\boldsymbol{o}_n$, an output vector $\boldsymbol{h}_n$, and a cell vector $\boldsymbol{c}_n$. These hidden vectors are calculated as follows:

$$\boldsymbol{f}_n = \text{sigmoid}(W_f \boldsymbol{v}_{w_n} + U_f \boldsymbol{h}_{n-1} + \boldsymbol{b}_f), \tag{2.31}$$

$$\boldsymbol{i}_n = \text{sigmoid}(W_i \boldsymbol{v}_{w_n} + U_i \boldsymbol{h}_{n-1} + \boldsymbol{b}_i), \tag{2.32}$$

$$\boldsymbol{o}_n = \text{sigmoid}(W_o \boldsymbol{v}_{w_n} + U_o \boldsymbol{h}_{n-1} + \boldsymbol{b}_o), \tag{2.33}$$

$$\boldsymbol{c}_n = \boldsymbol{f}_n \circ \boldsymbol{c}_{n-1} + \boldsymbol{i}_n \circ \tanh(W_c \boldsymbol{v}_{w_n} + U_c \boldsymbol{h}_{n-1} + \boldsymbol{b}_c), \tag{2.34}$$

$$\boldsymbol{h}_n = \boldsymbol{o}_n \circ \tanh(\boldsymbol{c}_n), \tag{2.35}$$

where W, U, and $\boldsymbol{b}$ are trainable parameters for each gate, and $\boldsymbol{v}_{w_n}$ is a word embedding corresponding to the n-th word in the tokenized sentence s'. $\circ$ denotes the Hadamard product. For the explanation, the aforementioned calculation is denoted with a function named LSTM($\cdot$):

$$\boldsymbol{h}_n = \text{LSTM}(\boldsymbol{v}_{w_1}, ..., \boldsymbol{v}_{w_n}). \tag{2.36}$$

Many researchers have reported that we can improve NLP performance by using two LSTM encoders for the forward direction $\overrightarrow{\text{LSTM}}$ and backward direction $\overleftarrow{\text{LSTM}}$ [131, 50]:

$$\overrightarrow{\boldsymbol{h}_n}, \overrightarrow{\boldsymbol{c}_n} = \overrightarrow{\text{LSTM}}(\boldsymbol{v}_{w_1}, ..., \boldsymbol{v}_{w_n}), \tag{2.37}$$

$$\overleftarrow{\boldsymbol{h}_n}, \overleftarrow{\boldsymbol{c}_n} = \overleftarrow{\text{LSTM}}(\boldsymbol{v}_{w_N}, ..., \boldsymbol{v}_{w_n}). \tag{2.38}$$

The architecture using two LSTMs for both directions is known as BiLSTM. Finally, we calculate the sentence vector $\boldsymbol{h}_{s'}$ by max-pooling over output vectors at all time steps:

$$\boldsymbol{h}_{s'} = \tanh(\text{MLP}^{(\text{enc})}(\text{maxpool}(\overrightarrow{\boldsymbol{h}_1} \oplus \overleftarrow{\boldsymbol{h}_1}, ..., \overrightarrow{\boldsymbol{h}_N} \oplus \overleftarrow{\boldsymbol{h}_N}))), \tag{2.39}$$

where $\oplus$ denotes vector concatenation. Figure 2.1b presents an overview of the calculation of the sentence vector with BiLSTM.

Neural Classifier with Large Pre-Trained Model

The performances of recent NLP approaches is brought by a model pre-trained on large text corpora. Specifically, we pretrain a sentence encoder on large text data

as a language model. We then train the language model in unsupervised learning without any annotations. In the training of the language model, we train the encoder to predict the next word from the previous context. More specifically, we maximize the probability of the word as conditioned by the context $p(w_n|s_{\backslash n})$, where $s_{\backslash n}$ is a sequence of words excluding the n-th word from s.

For example, using the BiLSTM-based encoder, we pretrain the encoder with a bi-directional language model:

$$\boldsymbol{h}_{\backslash n} = \overrightarrow{\text{LSTM}}(\boldsymbol{v}_{w_1}, ..., \boldsymbol{v}_{w_{n-1}}) \oplus \overleftarrow{\text{LSTM}}(\boldsymbol{v}_N, ..., \boldsymbol{v}_{n+1}), \tag{2.40}$$

$$p(w_n|s_{\backslash n}) = \text{softmax}(W\boldsymbol{h}_{\backslash n} + \boldsymbol{b})_{w_n}, \tag{2.41}$$

where W is a trainable parameter that converts $\boldsymbol{h}_{\backslash n}$ to a vocabulary-size array, and $\boldsymbol{b}$ is a trainable bias parameter. As described in (2.37) and (2.38), we encode the forward context $s_{1:n-1}$ and backward context $s_{N:n+1}$ with the forward and backward LSTMs, respectively. We then calculate the probability of the target word using the concatenation of bidirectional information. The encoder is pretrained to maximize the probabilities of target words over the large corpus.

This pretraining makes the sentence encoder informative on the large text corpus. Through these pretrained parameters, the sentence encoder can encode the sentence with rich information of natural language, contributing to improved performance on downstream tasks. ELMo [88] is a well-known pretrained encoder with some layers of BiLSTMs.

The primary pretrained model for most recent NLP is the bidirectional encoder representations from transformers (BERT) [22]. BERT is a sentence encoder that uses a Transformer-based architecture. Instead of the recursive encoding of words such as in BiLSTM, the Transformer-based encoder converts a sequence of words into a sentence vector in a single step:

$$\boldsymbol{h}_n = \text{BERT}(\boldsymbol{v}_{w_1}, ..., \boldsymbol{v}_{w_N})_n, \tag{2.42}$$

where $\text{BERT}(\cdot)$ is an encoder part of the Transformer known as a BERT encoder, which outputs N hidden vectors $(\boldsymbol{h}_1, ..., \boldsymbol{h}_n, ..., \boldsymbol{h}_N)$, corresponding to input word embeddings $(\boldsymbol{v}_1, ..., \boldsymbol{v}_n..., \boldsymbol{v}_N)$. The following Section 2.4.2 explains the architecture of the Transformer, as it was originally proposed for machine translation tasks.

For the pretraining of the BERT encoder, we calculated the probability of a

word conditioned by the context as follows:

$$\boldsymbol{h}_{\backslash n} = \mathrm{BERT}(\boldsymbol{v}_{w_1}, ..., \boldsymbol{v}_{w_{n-1}}, \boldsymbol{v}_{w_{\mathrm{MASK}}}, \boldsymbol{v}_{w_{n+1}}, ..., \boldsymbol{v}_{w_N})_n, \tag{2.43}$$

where $\boldsymbol{v}_{w_{\mathrm{MASK}}}$ is a word embedding corresponding to a special token w_{MASK} indicating a masked token. We can then obtain the probability of the word w_n over the context $s_{\backslash n}$ by inputting $\boldsymbol{h}_{\backslash n}$ to (2.41) and we can pretrain the encoder as the language model. The pretraining of the language model with the special token indicating mask is called a masked language model.

In addition to the masked language model, BERT is pre-trained on another task known as next sentence prediction. In this task, BERT is trained to predict whether the given two sentences are neighbors in the original corpus. Let $s_{(i)}$ be an i-th sentence in the original corpus D. Formally, we maximize the probability that a sentence $s_{(j)}$ follows a sentence $s_{(i)}$ in the original corpus:

$$\boldsymbol{h}_{\mathrm{CLS}} = \mathrm{BERT}(\boldsymbol{v}_{w_{\mathrm{CLS}}}, \boldsymbol{v}_{w_1^{(i)}}, ..., \boldsymbol{v}_{w_N^{(i)}}, \boldsymbol{v}_{w_{\mathrm{SEP}}}, \boldsymbol{v}_{w_1^{(j)}}, ..., \boldsymbol{v}_{w_M^{(j)}})_{\mathrm{CLS}}, \tag{2.44}$$

$$p(i+1=j|s_{(i)}, s_{(j)}) = \mathrm{sigmoid}(W\boldsymbol{h}_{\mathrm{CLS}} + b), \tag{2.45}$$

where $\boldsymbol{v}_{w_{\mathrm{CLS}}}$ and $\boldsymbol{v}_{w_{\mathrm{SEP}}}$ are word embeddings corresponding to special tokens for text classification and the indicator of a sentence boundary, respectively. W and b are trainable parameters to calculate a score for the next sentence prediction. $w_1^{(i)}$ and $w_1^{(j)}$ are the first words in the tokenized sentences $s'_{(i)}$ and $s'_{(j)}$ whose lengths are N and M, respectively. Note that $s'_{(i)}$ and $s'_{(j)}$ are different from the similar notation of k-th plausible tokenization of a sentence s'_k.

Once we pretrain the BERT encoder, we obtain the sentence vector $\boldsymbol{h}_{s'}$ as

$$\boldsymbol{v}_{s'} = \mathrm{maxpool}(\mathrm{BERT}(\boldsymbol{v}_{w_1}, ..., \boldsymbol{v}_{w_N})). \tag{2.46}$$

Alternative methods for obtaining the sentence vector are to use average pooling instead of max pooling or simply to employ the hidden vector corresponding to w_{CLS} and $\boldsymbol{h}_{\mathrm{CLS}}$ [92]:

$$\boldsymbol{v}_{s'} = \mathrm{BERT}(\boldsymbol{v}_{w_{\mathrm{CLS}}}, \boldsymbol{v}_{w_1}, ..., \boldsymbol{v}_{w_N})_{\mathrm{CLS}}. \tag{2.47}$$

Text Classification for Two Inputs

In the previous explanation of text classification, the focus was on a task that inputs a single sentence to the model. This was done for the sake of simplicity.

However, some tasks require the model to use more than two sentences. For example, natural language inference (NLI) [7] inputs two sentences to the model, and the model then predicts whether one sentence $s_{(1)}$ semantically entails the other sentence $s_{(2)}$ with three labels: entailment, contradiction, and neutral.

Two options can be used to input two sentences for the text classifier. One option is a method to input concatenation of two sentences into a single-sentence encoder:

$$s'_{(1\oplus 2)} = s'_{(1)} \oplus w_{\text{SEP}} \oplus s'_{(2)}, \tag{2.48}$$

where w_{SEP} is a special token indicating a boundary of sentences, and $s'_{(1)}$ and $s'_{(2)}$ are tokenized sequences of $s_{(1)}$ and $s_{(2)}$, respectively. Note that $s'_{(1)}$ and $s'_{(2)}$ are different from the similar notation of k-th plausible tokenization of a sentence s'_k. We input the concatenated sequence $s'_{(1\oplus 2)}$ to the sentence encoder $\boldsymbol{h}_{s'_{(1\oplus 2)}} = g(s'_{(1\oplus 2)})$ and use the sentence vector for the text classification. This option is simple but consumes more time for the recursive calculation of a long concatenated sequence. In addition, this type of long input causes unstable learning, which is known as the vanishing or exploding gradient problem [41].

The other option is to prepare two unique sentence encoders for each sentence.

$$\boldsymbol{h}_{s'_{(1)}} = g_{(1)}(s'_{(1)}), \tag{2.49}$$

$$\boldsymbol{h}_{s'_{(2)}} = g_{(2)}(s'_{(2)}), \tag{2.50}$$

$$\boldsymbol{h}_{s'_{(1\oplus 2)}} = \boldsymbol{h}_{s'_{(1)}} \oplus \boldsymbol{h}_{s'_{(2)}}, \tag{2.51}$$

where $g_{(1)}(\cdot)$ and $g_{(2)}(\cdot)$ are different sentence encoders corresponding to the two sentences. If the language used in two inputs, we can also use the same encoder for $g_{(1)}(\cdot)$ and $g_{(2)}(\cdot)$. This study used the latter option for the task of NLI following the literature [7].

2.4.2 Machine Translation

This study employs machine translation as a downstream task because it is a famous task of NLP as well as text classification. This section first describes the task definitions of machine translation tasks and then introduces Transformer, on which the most recent state-of-the-art models are based.

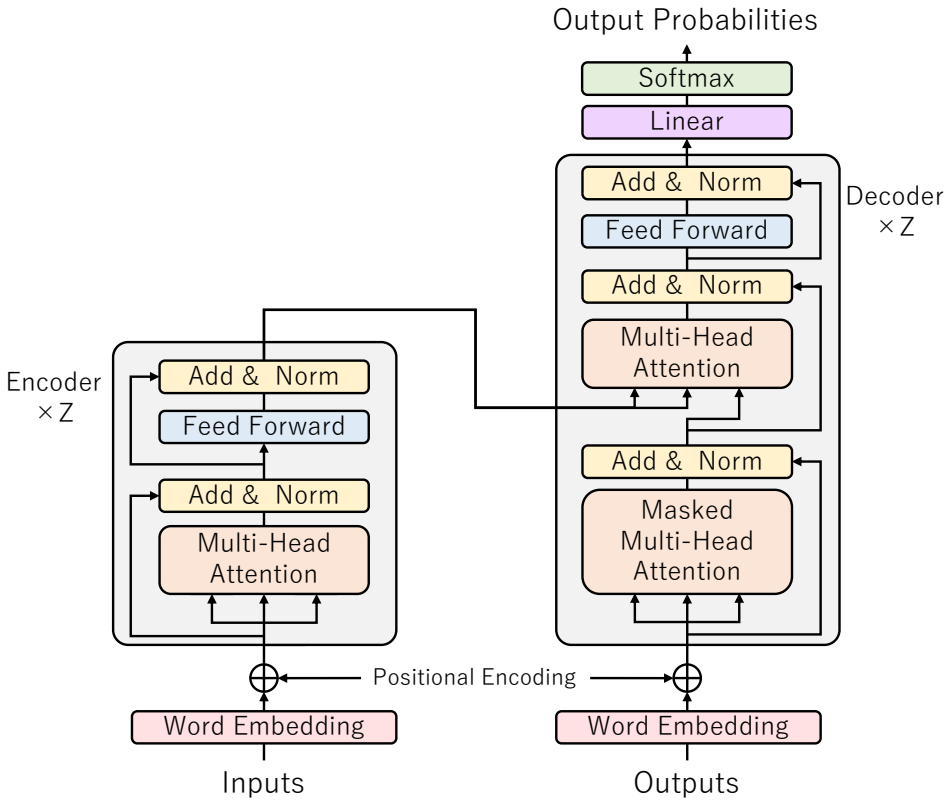


Figure 2.2: Outline of Transformer for machine translation.

Task Overview

Machine translation tasks require that the downstream model convert a source sentence written in one language to a target sentence in another language (i.e., English to German). Let s and t be the source and target sentences, respectively. The tokenized sentences are $s' = w_1^{(s')}...w_n^{(s')}...w_N^{(s')}$ and $t' = w_1^{(t')}...w_m^{(t')}...w_M^{(t')}$. Machine translation can be formally defined as a task that maximizes a probability of the translation from s to t:

$$p(t|s;\theta) = \prod_{m=1}^{M} p(t'_m|s'_{1:N}, t'_{1:m-1};\theta), \tag{2.52}$$

where θ denotes parameters of the translation model. As (2.52) shows, machine translation is considered a language model of the target language conditioned by the source sentence.

Transformer

This section introduces a state-of-the-art architecture for machine translation tasks called Transformer [112]. Figure 2.2 overviews the architecture of Transformer. Transformer is a sequence-to-sequence model referred to as encoder–decoder [104], as it is composed of an encoder f and decoder g. The encoder converts the input sentence into a vector representation, and the decoder generates an output sentence from the encoded sentence:

$$t' = g(f(s')). \tag{2.53}$$

As Figure 2.2 shows, the encoder part of Transformer consists of two modules: multi-head attention and feed-forward networks. The decoder part is composed of three modules: masked multi-head attention, multi-head attention, and feed-forward networks.

Because Transformer is composed of piles of linear transformations, the architecture cannot distinguish the order of the input sequence as well as can the attention-based encoder referred to in (2.30). Therefore, Transformer introduced positional encodings corresponding to the position of the n-th word based on periodic functions. Specifically, when the size of the word embedding $\boldsymbol{v}_{w_n}$ located at the n-th position in the sentence is d, the positional encoding is a d-sized vector whose i-th element is calculated as

$$\mathrm{PE}(n, 2i) = \sin(\frac{n}{10000^{\frac{2i}{d}}}), \tag{2.54}$$

$$\mathrm{PE}(n, 2i+1) = \cos(\frac{n}{10000^{\frac{2i}{d}}}). \tag{2.55}$$

We then calculate the positional encoded word embedding $\tilde{\boldsymbol{v}_{w_n}}$:

$$\tilde{\boldsymbol{v}_{w_n}} = \boldsymbol{v}_{w_n} + [\mathrm{PE}(n, 1), ..., \mathrm{PE}(n, d)]. \tag{2.56}$$

Regarding the multi-head attention module, a scaled dot-product attention that can be considered a single head attention module is described. For a simple explanation, we consider the attention mechanism of the encoder part for a sentence s'. This module requires three inputs $Q = \boldsymbol{q}_1, ..., \boldsymbol{q}_n, ..., \boldsymbol{q}_N$, $K = \boldsymbol{k}_1, ..., \boldsymbol{k}_n, ..., \boldsymbol{k}_N$, $R = \boldsymbol{r}_1, ..., \boldsymbol{r}_n, ..., \boldsymbol{r}_N$[7] corresponding to each word $w_n^{(s')}$, where

[7] Q, K, and V are known as a query, key, and value for the input to the attention layer. R and $\boldsymbol{r}$ are originally denoted as V and $\boldsymbol{v}$, respectively, in Vaswani et al. [112]. This study uses R and $\boldsymbol{r}$ to avoid the conflict with the notation of the word embedding $\boldsymbol{v}$.

the sizes of each vector $\boldsymbol{q}_n$, $\boldsymbol{k}_n$, and $\boldsymbol{r}_n$ are d dimension vectors, and the sizes of Q, K, and R are $N \times d$. Note also that $N = |s'|$. Using these inputs, Transformer calculates the attended vector $H = \boldsymbol{h}_1, ..., \boldsymbol{h}_n, ..., \boldsymbol{h}_N$ using function Attention($\cdot$):

$$H = \text{Attention}(Q, R, K), \tag{2.57}$$

$$= \text{softmax}(\frac{QK^\top}{\sqrt{d}})R. \tag{2.58}$$

To capture the various aspects of the relation between Q and K, Transformer employs a multi-head attention mechanism composed of I single head attentions:

$$H = \text{MultiHead}(Q, R, K), \tag{2.59}$$

$$= [H^{(1)}\oplus, ..., \oplus H^{(i)}\oplus, ... \oplus H^{(I)}]W^O, \tag{2.60}$$

$$\text{where } H^{(i)} = \text{Attention}(QW_Q^{(i)}, KW_K^{(i)}, RW_R^{(i)}), \tag{2.61}$$

where $H^{(i)}$ is an output of the i-th single head attention whose inputs are converted with trainable parameters $W^{(i)}$ corresponding to each head, and W^O is a trainable parameter to convert the concatenated outputs of each attention module. The sizes of trainable parameters $W_Q^{(i)}$, $W_K^{(i)}$, and $W_R^{(i)}$ are $d \times \frac{d}{I}$, and the size of W^O is $d \times d$. The inputs for the first layer of the encoder part are the positional encoded word embeddings $Q = K = R = [\tilde{\boldsymbol{v}_{w_1}}, ..., \tilde{\boldsymbol{v}_{w_N}}]$ calculated using (2.56), and those for the second or later layers are $Q = K = R = O^{(\text{prev})}$, where $O^{(\text{prev})}$ is an output of the previous encoder layer.

The output of the multi-head attention H is converted using a parameterized feed-forward network accompanied by a layer normalization [3]:

$$C = \text{LayerNorm}(Q + H), \tag{2.62}$$

$$O = \text{LayerNorm}(C + \text{FFN}(C)), \tag{2.63}$$

where LayerNorm($\cdot$) and FFN($\cdot$) are modules for the layer normalization and feed-forward network, respectively. The FFN($\cdot$) is defined using two linear transformations and a ReLU activation:

$$\text{FFN}(C) = \max(0, CW_1 + \boldsymbol{b}_2)W_2 + \boldsymbol{b}_2, \tag{2.64}$$

where W and $\boldsymbol{b}$ are trainable parameters.

The encoder part of Transformer is composed of layers that include the aforementioned multi-head attention and feed-forward network. Figure 2.2 illustrates

the Transformer layers piled Z times for both the encoder and decoder. The final output of the encoder is vectors $O^{(\text{enc})} = \boldsymbol{o}_1^{(\text{enc})}, ..., \boldsymbol{o}_N^{(\text{enc})}$ that are generated by the last layer of the encoder layer.

The architecture of the decoder part is similar to that of the encoder. We input the positional encoded word embeddings to the decoder part and calculate the probability of the next word $w_{m+1}^{(t')}$ from the previous history (e.g., $\boldsymbol{o}_1^{(\text{dec})}, ..., \boldsymbol{o}_m^{(\text{dec})}$ corresponding to $w_1^{(t')}, w_m^{(t')}$. Unlike the encoder input, we feed only the forward context to the decoder because the decoder part should be trained as a language model to generate a sequence of words. Specifically, Transformer realizes the input mechanism, such as masked multi-head attention, which masks the attention weight for the elements corresponding to the words after the predicted word. For instance, the model to predict the $m+1$-th word masks the attention for elements at $m+1, ..., M$, and then the model is unable to access information from the later parts of the input sequence.

Another difference of the decoder part is the input of the multi-head attention module. As Figure 2.2 shows, the multi-head attention module located in the decoder part receives the output vector $O^{(\text{enc})}$ from the encoder as well as the input inside the decoder. In short, the multi-head attention module takes vectors $Q = O^{(\text{enc})}$, $K = O^{(\text{enc})}, R = H^{(\text{prev})}$ for its input. This mechanism is called the cross-attention (or source-target attention) mechanism [4, 69] between the encoder and decoder, and it enables the model to generate words in the target language by considering the encoder information.

3 OpTok: Optimizing Tokenization for Text Classification

3.1 Model Outline

This chapter describes the proposed architecture for **op**timizing **tok**enization (OpTok). OpTok identifies an appropriate tokenization for a downstream task, particularly for text classification tasks, using sentence representation. In other words, OpTok identifies a tokenization that yields a better score for a downstream task. OpTok converts a given sentence s into a sequence of tokens in vocabulary $w \in V$ (i.e., $s' = w_1...w_i...w_I$), where I is the number of tokens included in the sentence, and i is an index of the token in the sequence. The downstream model achieves the best score with s' among all possible tokenized sentences. Thus, let $q(\cdot)$ be a loss function, z be a ground truth of the downstream task, and $f(\cdot)$ be the downstream model (i.e., any neural architecture). OpTok searches the tokenization s' that minimizes the loss value of the downstream task: $\text{argmin}_{s'}(q(z, f(s')))$.

To search for s' that satisfies $\text{argmin}_{s'}(q(z, f(s')))$, OpTok is trained based on the loss value of the downstream task ($q(z, f(s'))$). Thus, both OpTok and the downstream model are optimized simultaneously. This is in contrast to a traditional pipeline approach, which tokenizes a given sentence as part of preprocessing. OpTok generates multiple tokenized sentences as candidates, and OpTok is trained to assign a high probability to a better tokenization based on the score of the downstream task. During the inference step, OpTok is configured to output only the most plausible tokenized sentence, as this reduces computational costs.

Figure 3.1 presents an overview of OpTok with the downstream model during

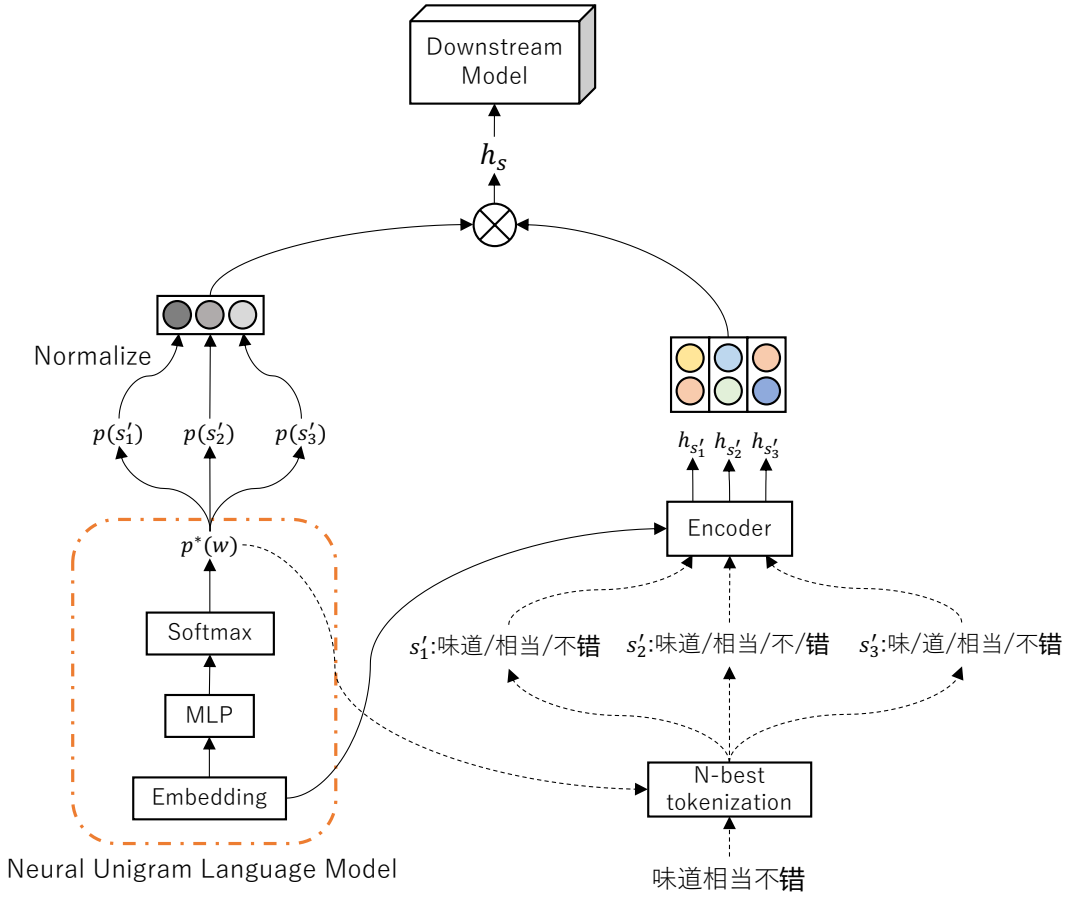


Figure 3.1: Outline of the proposed method for calculating a sentence vector $\boldsymbol{h}_s$ with the 3-best tokenizations during the training phase. At the inference, OpTok uses the 1-best tokenization as well as general neural architectures. The arrowed continuous lines indicate the differentiable paths for back-propagation. We can use various architectures as the *Encoder*, which converts a sequence of tokens into a single vector. The *downstream model* is the architecture used for downstream tasks (i.e., MLP for text classification).

training. OpTok constructs N tokenized sentences and converts them into vector representations with a neural encoder. Then, OpTok combines the probabilities of each tokenization with the vector representations. OpTok computes the sum of the vector representations weighted by the probabilities and then inputs it into the

downstream model. Thus, OpTok assigns a high probability to the tokenization to improve the performance of the downstream task. We can therefore obtain s' satisfying $\mathrm{argmin}_{s'}(q(z, f(s')))$ through the training. The details of each module are described in this section.

3.2 Neural Unigram Language Model

OpTok calculates the probability of a token $p(w)$ with a neural unigram language model as follows:

$$d_w = \mathrm{MLP}(\boldsymbol{v}_w), \tag{3.1}$$

$$p(w) = \frac{\exp(d_w)}{\sum_{\hat{w} \in V} \exp(d_{\hat{w}})}, \tag{3.2}$$

where MLP denotes a multilayer perceptron containing trainable parameters, and $\boldsymbol{v}_w$ is an embedding of the word w.

To stabilize the learning, as explained in Section 3.5, OpTok employs a smoothed distribution of unigram probability [59] using a hyperparameter α. The smoothed probability is obtained as

$$p^*(w) = \frac{p(w)^\alpha}{\sum_{\hat{w} \in V} p(\hat{w})^\alpha}. \tag{3.3}$$

This smoothing technique resembles the one for subword regularization described in (2.22). A sentence is converted into a sequence of tokens based on the probability of a tokenized sentence:

$$p(s') = \prod_{w \in s'} p^*(w). \tag{3.4}$$

Vocabulary V is initialized with a reasonable number of tokens. To choose the initial vocabulary, both supervised and unsupervised word segmentation methods are available (e.g., publicly available pretrained tokenizers [58, 122] and vocabulary acquired using unsupervised word segmentation [29, 72, 95]). This study uses SentencePiece [60] for initialization.

3.3 Module for Selecting Tokenization

OpTok generates multiple tokenized sentences as candidates and converts them into a single vector using their probabilities during the training phase. OpTok

first obtains the N-best tokenization of the sentence $s'_1, ..., s'_n, ..., s'_N$ using the forward-DP backward-A* algorithm [74] for the probabilities generated using the language model described in Section 3.2. OpTok then converts the tokenized sequences into the vectors $\boldsymbol{h}_{s'_n}$ severally as follows:

$$\boldsymbol{h}_{s'_n} = g(s'_n), \tag{3.5}$$

where $g(\cdot)$ is a neural encoder that encodes the sequence of tokens, such as a module using a convolutional neural network [125, 63, 47] or BiLSTM [131, 50]. It was determined that the learning is stabilized by sharing word embeddings between the encoder and neural unigram language model.

Finally, OpTok calculates the final vector of the sentence by weighting the vectors of the candidates using their probabilities calculated through (3.4) as follows:

$$a_n = \frac{p(s'_n)}{\sum_{m=1}^{N} p(s'_m)}, \tag{3.6}$$

$$\boldsymbol{h}_s = \sum_{n=1}^{N} a_n \boldsymbol{h}_{s'_n}. \tag{3.7}$$

As with the attention mechanism, we normalize the probability to meet a restriction $\sum_{n=1}^{N} a_n = 1$[1].

We can use this vector $\boldsymbol{h}_s$ in the same manner as the general encoded vectors. For example, we can construct a neural text classifier by converting $\boldsymbol{h}_s$ into a label-sized vector with an MLP. When the entire model is updated with a training loss, such as the cross-entropy loss against the gold label, the language model is also updated to assign the higher probability to the useful tokenization for the downstream task.

At the inference, we obtain the optimal tokenization using the Viterbi algorithm [114]. We can also use N-best tokenization for the downstream model in the evaluation phase. Although this option might improve the performance of the downstream model, this study did not use this option because it is uncommon in evaluation of NLP tasks.

During the tokenization process, OpTok does not consider a token that includes whitespaces as the candidate of words. In other words, we can input a sentence

[1] An alternative approach for sampling plausible tokenizations using Gumbel softmax [48] was attempted but was found to cause instability in the learning.

that includes whitespaces, such as in English, and treat the whitespace as word boundaries in the tokenization.

3.4 Restricting Vocabulary

When we identify the appropriate tokenization using the proposed method, the tokenizer often converges to the local optima, which the tokenizer uses a longer and more unique tokens for each downstream task. This problem is known as length bias [62, 61]. In calculating word probability using the unigram language model, the tokenizer with the proposed method frequently chooses a tokenization with fewer tokens (i.e., a tokenization containing many longer tokens).

To mitigate these local optima, a restriction is introduced for the size of the vocabulary during training. OpTok constructs the restricted vocabulary V' sampled from the original vocabulary V (where $|V'| \leq |V|$) at the beginning of each mini-batch and uses V' as the vocabulary in the mini-batch. The sampling is processed based on the smoothed probability of tokens $p^*(w)$ described in Section 3.2. We then calculate the new probability distribution of tokens in V' by normalizing their probabilities. In addition, OpTok prepares the embeddings for entire tokens in V. However, this study treats a token outside V' as an unknown token in each mini-batch. At the inference, OpTok constructs vocabulary by taking the top-$|V'|$ tokens from V based on the updated token probabilities obtained by (4.2). There is also an option to use the entire vocabulary for the inference.

Because the lengths of frequent words are shorter than those of infrequent words, the sampled V' contains many shorter words. Thus, we can use vocabulary that does not contain longer and unique words for the training of the proposed method in many cases. In other words, we can frequently use the tokenizations composed of shorter and general words for the training. This procedure is a type of vocabulary restriction that uses a continuous cache technique [34, 52].

This type of sampling of the vocabulary results in a diversity of tokenizations with the N-best candidates during training. When the lower α mentioned in Section 3.2 is selected , the distribution of the tokens becomes flatter, and the model can sample various tokens for V'. In addition, through the sampling process, we can reduce the importance of words that are not useful in V for the downstream task.

3.5 Maintaining the Characteristics of the Language Model

Because the optimization of OpTok depends only on the loss function for the downstream task, the language model of OpTok may be considerably different from the unigram language model (i.e., in terms of frequency of words) obtained from the training corpus. In some cases, we must keep the corpus-based language model. To address these cases, we can use the following loss for the sentence s to update the language model using the neural EM algorithm [21, 66, 110]:

$$\mathcal{L}_s^{\mathrm{lm}} = -\sum_{n=1}^{N} a_n \sum_{w \in s'_n} \log p^*(w). \tag{3.8}$$

We then optimize the weighted sum of the downstream task loss and $\mathcal{L}_s^{\mathrm{lm}}$. Consider text classification as an example. We use cross-entropy loss for the ground-truth label of the sentence $\mathcal{L}_s^{\mathrm{cl}}$. We can then optimize the following equation:

$$\mathcal{L}_s = \mathcal{L}_s^{\mathrm{cl}} + \mu \mathcal{L}_s^{\mathrm{lm}}, \tag{3.9}$$

where μ is a hyperparameter. When μ is sufficiently large to ignore $\mathcal{L}_s^{\mathrm{cl}}$, the training of the proposed method is nearly the same as the scheduled training with stochastic tokenization [39].

4 OpTok4AT: Optimizing Tokenization for Various Tasks

4.1 Model Outline

Because OpTok exploits sentence representations to obtain the gradients for the tokenizer, it cannot be used for NLP tasks that do not require sentence representations, such as machine translation tasks. To address this problem, this study extends OpTok to be applicable to various NLP tasks to establish OpTok for any task (OpTok4AT).

Like OpTok, OpTok4AT consists of a tokenizer and downstream model, and the two modules are optimized simultaneously. Section 4.2 of this chapter first presents the training outline of a case in which we use one sentence as input. Section 4.3 introduces the training of the tokenizer, and Section 4.4 presents the training of downstream model. Finally, Section 4.5 explains the training strategy for a task that requires multiple inputs, such as machine translation.

4.2 Optimizing Tokenization with Loss

The concepts of OpTok and OpTok4AT are the same. This section explains the notation and purpose of the proposed method. OpTok4AT tokenizes a sentence s into a sequence of words w in vocabulary V, $s' = w_1, ..., w_I$, where I is the sequence length. In this tokenization process, the purpose is to minimize the following loss value corresponding to the tokenized input s':

$$\mathcal{L}_{s'} = q(f(s'), z), \tag{4.1}$$

where $f(s')$ is a downstream model that outputs a prediction of the downstream task from a tokenized sentence s', and $q(f(\cdot), z)$ is a task-specific loss function between a model prediction and supervisory signal z.

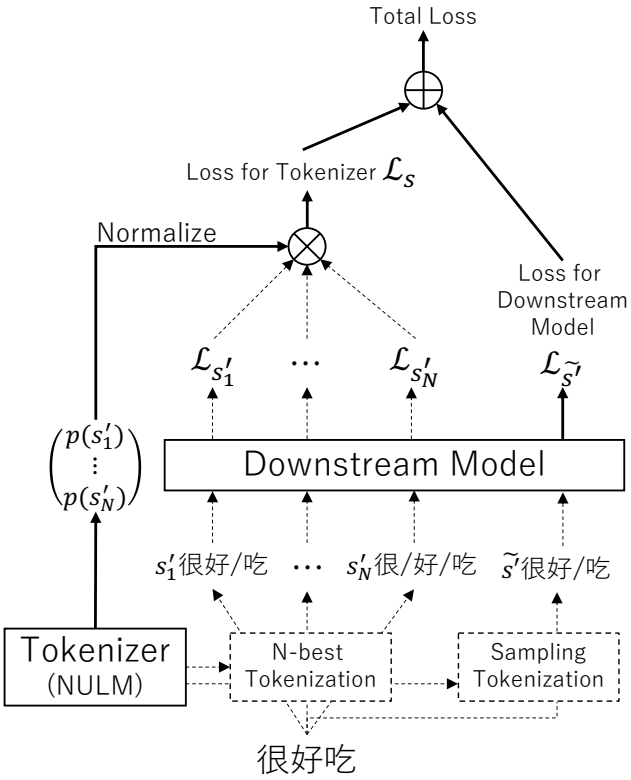


Figure 4.1: Overview of OpTok4AT in which losses for a tokenizer $\mathcal{L}_s$ and for a downstream model $\mathcal{L}_{\tilde{s}'}$. $\mathcal{L}_s$ and $\mathcal{L}_{\tilde{s}'}$ are calculated using N-best tokenizations (Section 4.3) and a sampled tokenization (Section 4.4), respectively. The arrowed continuous lines indicate differentiable paths for back-propagation.

Figure 4.1 presents an outline of OpTok4AT. To determine the tokenization that satisfies $\text{argmin}_{s'}(q(f(s'), z))$, the tokenizer is updated to assign a higher probability to a useful tokenization for the downstream model. Specifically, OpTok4AT constructs N tokenizations $s'_1, ..., s'_n, ...s'_N$ for a training instance and then compute loss values for each tokenization. Each loss is weighted with probability $p(s'_n)$ computed by the tokenizer and the weighted sum is used to train

the tokenizer:

$$a_n = \frac{p(s'_n)}{\sum_{m=1}^{N} p(s'_m)}, \tag{4.6}$$

$$\mathcal{L}_s = \sum_{n=1}^{N} a_n \mathcal{L}_{s'_n}. \tag{4.2}$$

This study used N-best tokenizations. In these equations, losses $\mathcal{L}_{s'_1}, ..., \mathcal{L}_{s'_N}$ corresponding to N-best tokenizations are weighted with their sentence probabilities normalized such that the sum is 1. By optimizing the tokenizer based on the weighted sum $\mathcal{L}_s$, the tokenizer assigns high probability to the appropriate tokenization for the downstream model.

We can use any function for $f(\cdot)$ and $q(f(\cdot), \cdot)$ in (4.1). Therefore, unlike OpTok, OpTok4AT has no restrictions on the downstream task and model. For instance, in a case in which text classification is the downstream task, $f(\cdot)$ is a neural network that predicts a label of a given tokenized sentence, and $q(f(\cdot), \cdot)$ is the cross-entropy loss between the model prediction and true label.

Unlike with OpTok, OpTok4AT updates the tokenizer directly using the loss values corresponding to each tokenized candidates. The differences between the two versions of the method can be seen by comparing (3.7) and (4.2). Unlike OpTok, whose architecture includes the weighted sum of sentence vectors with the tokenization probabilities in the downstream calculation, the tokenizer in OpTok4AT is connected to the outside of the downstream model. Therefore, OpTok4AT does not require sentence representation and can be easily managed with various NLP tasks.

4.3 Tokenizer using Neural Unigram Language Model

As in OpTok, OpTok4AT employs a neural unigram language model as the tokenizer. The architecture of the neural unigram language model has been explained in Section 3.2. Specifically, the model calculates the unigram probability

of a word $p(w)$ with a word embedding $\boldsymbol{v}_w$ as follows:

$$d_w = \mathrm{MLP}(\boldsymbol{v}_w), \tag{4.1}$$

$$p(w) = \frac{\exp(d_w)}{\sum_{\hat{w} \in V} \exp(d_{\hat{w}})}, \tag{4.2}$$

where $\mathrm{MLP}(\cdot)$ is a multilayer perceptron. Vocabulary V is initialized with a reasonable number of words. For example, for the initialization, we can use a tokenization through a dictionary-based tokenizer such as MeCab [58] or unsupervised word segmentation such as SentencePiece [60] or BPE [95]. The probability of a tokenization $p(s')$ is calculated as follows:

$$p(s') = \prod_{w \in s'} p(w). \tag{4.3}$$

Unlike OpTok, OpTok4AT does not calculate the probability of the tokenization using the smoothed language model described in (3.4). This is because OpTok4AT employs subword regularization for the training of the downstream model, as described in Section 4.4, and this indirectly varies the tokenized candidates by requiring the tokenizer to use a different type of tokenization in each training epoch.

For the training with (4.2), we obtain N-best tokenizations by applying the forward-DP backward-A* algorithm [74] for possible tokens against sentence s. In the inference phase, we can also obtain the 1-best appropriate tokenization for the downstream task using the Viterbi algorithm [114] with the trained neural unigram language model. Like OpTok, OpTok4AT treats whitespaces as word boundaries. We can also use the forward-DP backward-A* algorithm to input the N-best tokenization into the downstream model even during the evaluation phase. For example, OpTok4AT is applicable to the N-best decoding [59] for machine translation tasks.

4.4 Downstream Model Training

The downstream model can be trained directly using the loss $\mathcal{L}_s$ in (4.2), but this study does not choose this option for two reasons. The first reason is limited computer memory. When using the loss $\mathcal{L}_s$ for the update of the downstream

model in the OpTok4AT architecture, we must calculate the forward and backward calculations of the neural networks as holding N models with the computational graph. This is because OpTok4AT includes the entire architecture of the downstream model. This is unlike OpTok, which inputs the weighted sum of N sentence representations into a single downstream model. Although this is not a problem when using a small downstream model such as a BiLSTM-based text classifier, updating the downstream model is not easy when we use larger models such as a classification architectures with BERT [22] or a machine translation architectures with Transformer [112]). The second reason is the gap between the training and evaluation. As in OpTok, when the loss $\mathcal{L}_s$ is used for the training of the downstream model, a gap exists between the training phase that uses N-best tokenizations and the evaluation phase that uses only one tokenization. This gap might have an undesirable effect on the performance of the downstream task. In other words, both the downstream model and evaluation phase should be trained with a single tokenization, if possible.

To address these problems, this study uses subword regularization [59] to train the downstream model of OpTop4AT. Thus, $\mathcal{L}_{\tilde{s'}} = q(f(\tilde{s'}), z)$ is calculated for a single sampled tokenization $\tilde{s'}$ and $\mathcal{L}_{\tilde{s'}}$ is used to train the downstream model. More specifically, OpTok4AT updates the entire model to minimize the sum of two losses $\mathcal{L}$: one to train the neural unigram language model; the other to train the downstream model:

$$\mathcal{L} = \mathcal{L}_s + \mathcal{L}_{\tilde{s'}}. \tag{4.4}$$

Because only the single sampled tokenization is used for the training of the downstream model, OpTok4AT does not require that N models be stored for the update of the downstream model.

The unigram probabilities of tokens calculated with (4.2) are used to sample tokenization $\tilde{s'}$ as follows:

$$\tilde{s'} \sim \frac{p(\tilde{s'})^\alpha}{\sum_{k=1}^{K} p(s'_k)^\alpha}. \tag{4.5}$$

Here, $\alpha \in \mathbb{R}^+$ is a hyperparameter that controls the diversity of the sampled tokenization. If we set α to a lower value, the distribution is similar to the uniform distribution; otherwise, the distribution strongly depends on each tokenization probability $p(s'_k)$. K is a hyperparameter denoting the number of candidates for

sampling, and we use forward-filtering backward sampling [94, 72] if $K = \infty$. Subword regularization was described in Section 2.3.

Subword regularization not only enhances the downstream model but also provides various tokenizations to the downstream model during training. Therefore, subword regularization enables exploring the appropriate tokenization, and OpTok4AT does not require using techniques to vary the tokenization candidates used in OpTok (Section 3.4).

4.5 Training with Multiple Sentences as Inputs

The case in which we use a single sentence as input was previously discussed. However, we need to input multiple sentences to the downstream model in some tasks. This section describes a training strategy that can be used for these cases.

To compute the loss value for training the tokenizer, multiple tokenizations are considered for one sentence, and the sampled tokenization is used for the others. For example, in machine translation, two inputs (the source and target sentences) are used to train the downstream model. The source and target sentences are the input of the downstream model and supervisory signal, respectively. In this case, the tokenizations of both source and target sentences can be optimized by OpTok4AT. Let s and t be the source and target sentences, respectively, and s' and t' be the corresponding tokenizations. The neural unigram language model of the source side is updated using the loss value:

$$a_n = \frac{p(s'_n)}{\sum_{m=1}^{N} p(s'_m)}, \tag{4.6}$$

$$\mathcal{L}_{s'_n} = q(f(s'_n), \tilde{t'}), \tag{4.6}$$

$$\mathcal{L}_s = \sum_{n=1}^{N} a_n \mathcal{L}_{s'_n}, \tag{5.2}$$

where $\tilde{t'}$ is a sampled tokenization for the target sentence. The loss for the neural

unigram language model of the target side is also calculated using

$$b_n = \frac{p(t'_n)}{\sum_{m=1}^{N} p(t'_m)}, \tag{4.7}$$

$$\mathcal{L}_{t'_n} = q(f(\tilde{s'}), t'_n), \tag{4.8}$$

$$\mathcal{L}_t = \sum_{n=1}^{N} b_n \mathcal{L}_{t'_n}, \tag{4.9}$$

where $\tilde{s'}$ is a sampled tokenization.

To train the downstream model with the multiple sentences, sampled tokenizations are used for all input sentences. For example, in machine translation, OpTok4AT computes the loss value $\mathcal{L}_{\tilde{s'},\tilde{t'}} = q(f(\tilde{s'}), \tilde{t'})$ for the sampled and target sentences and use it for the training of the downstream model. When the downstream model is trained for a mini-batch $B \ni (s, t)$, the parameters are updated for the following loss value:

$$\mathcal{L} = \sum_{(s,t)\in B} \mathcal{L}_s + \mathcal{L}_t + \mathcal{L}_{\tilde{s'},\tilde{t'}}. \tag{4.10}$$

Figure 4.2 outlines this training process for the neural unigram language model of the source side, where the training for the target side can be similarly described.

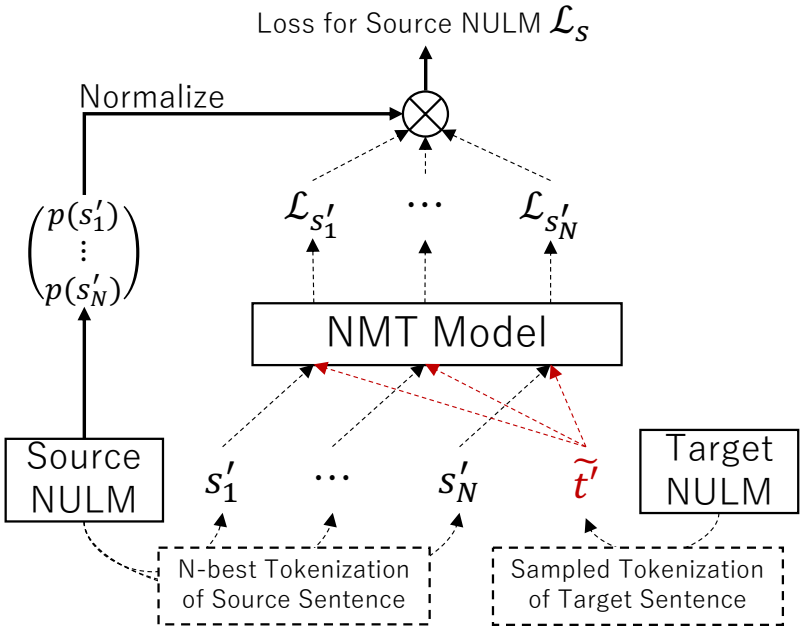


Figure 4.2: Overview of the calculation of a tokenization loss $\mathcal{L}_s$ for the source-side neural unigram language model in NMT requiring the two inputs of source and target sentences s and t. The arrowed continuous lines indicate the differentiable paths for back-propagation.

5 Experiments

To validate the applicability of the proposed methods to various downstream tasks, experiments on text classification and machine translation tasks were conducted based on previous studies. The two tasks are popular benchmarks in NLP and are sufficiently different to evaluate the proposed methods, as one is a classification problem and the other is a generation problem. This study exploited datasets in three languages for text classification tasks and seven language pairs for machine translation tasks to evaluate the effectiveness of the proposed methods over certain languages.

This study employed SentencePiece [60] as a tokenizer of the baseline method for both tasks. Subword regularization [59] was also used for the SentencePiece tokenization as a strong baseline. For the machine translation tasks, DPE [37], an existing method based on neural networks, can be used to find better tokenization of the target sentences depending on the translation corpora. Therefore, we employed DPE as another baseline.

This chapter describes experiments on text classification (Section 5.1) and machine translation (Section 5.2). Each section describes the datasets used in the experiment, the settings for the training configuration, and results.

5.1 Text Classification

To evaluate the proposed methods, experiments on text classification tasks were first conducted, in which a model predicted the label from a text as its input. Both OpTok and OpTok4AT are applicable to text classification tasks. The task definitions and commonly used architectures were previously described in Section 2.4.1.

	Positive	Negative	Neutral	Total
Weibo(Zh)	407,057	263,995	-	671,052
Twitter(Ja)	10,319	16,035	135,830	162,184
Twitter(En)	56,462	43,538	-	100,000

Table 5.1: Dataset components on sentiment analysis.

	Entailment	Contradiction	Neutral	Total
Train	183,416	183,187	182,764	549,367
Validation	3,329	3,278	3,235	9,842
Test	3,368	3,237	3,219	9,824
Total	190,113	189,702	189,218	569,033

Table 5.2: Overview of the dataset splitting of SNLI.

5.1.1 Dataset

To confirm the effectiveness of the proposed methods on various languages, this study used datasets in sentiment analyses of Chinese, Japanese, and English. The corpora on the SNS domain were used because they have many informal expressions, and thus the effects of differences in tokenization on the performance of text classification could be assessed. This study also conducted experiments on a dataset of E-commerce reviews in which sentences contained two types of labels (genre of a product and rating of a review) to investigate whether OpTok and OpTok4AT found different tokenizations for each label[1]. In addition, this study used a textual entailment dataset in English to determine whether the proposed methods could be applied to tasks that used two sentences as input.

Table 5.1 presents overviews of the datasets of sentiment analysis. For sentiment analysis, each dataset is randomly split into a ratio of 8:1:1 for training, validation, and testing. Each dataset of genre and rating prediction is split into a ratio of 8:1:1 to achieve well-balanced genres, in which both tasks shared the same split. Details on these datasets are as follows.

[1]The actual tokenization acquired by the proposed methods are discussed in the foolowing Section 6.4.

Weibo(Zh)

Weibo(Zh)[2] includes short Chinese texts on the Weibo SNS using two sentiment labels: *positive* and *negative*. Because the available data are already tokenized with a preprocesser[3], the whitespaces are removed to detokenize them.

Twitter(Ja)

Twitter(Ja) [105][4] is a dataset of short Japanese texts from a short-text Twitter SNS on products such as electric appliances. The samples of this dataset initially used more than one of five sentiment labels for a target topic: *positive*, *negative*, *neutral*, *both positive and negative*, and *unrelated*. Tweets totaling 352,554 were available by the summer of 2018, and tweets that used only a single sentiment label of *positive*, *negative*, or *neutral*, are extracted for the experiment. In other words, this study removed *both positive and negative* and *unrelated* to prevent confusion.

Twitter(En)

Twitter(En)[5] is a dataset of short English texts from the Twitter SNS that uses two sentiment labels: *positive* and *negative*. This study exploited this corpus without any preprocessing such as casing and pre-tokenization.

SNLI

SNLI [7] is a widely used dataset for recognizing textual entailment, which is a type of text classification that requires two input sentences of a (*premise* and *hypothesis*) in English. This study employed this dataset to validate the performance of OpTok and OpTok4AT when using multiple sentences. More specifically, this study used the default split of this corpus and applied only the labeled samples following existing studies [7, 88, 22, 91, 43]. In other words, this study used only samples with labels of *entailment*, *contradiction*, or *neutral*. Table 5.2 presents an overview of data splitting.

[2] `https://github.com/wansho/senti-weibo`
[3] `https://github.com/wansho/weibo-preprocess-toolkit`
[4] `http://www.db.info.gifu-u.ac.jp/data/Data_5d832973308d57446583ed9f`
[5] `https://www.kaggle.com/c/twitter-sentiment-analysis2`

Amazon: Genre&Rating(En)

Genre(En) and Rating(En) are datasets in English that were created from Amazon product data [79][6]. This data has reviews from 24 product genres with attached ratings from users of 1 to 5. 5,000 reviews were sampled from product genres containing a sufficient number of reviews. In this process, the number of tokens in each review was counted based on whitespaces and reviews containing more than 200 tokens were removed. The sampled reviews were used for rating and genre prediction tasks from the same review texts. Table 5.3 lists the contents of the dataset.

JD.com: Genre&Rating(Zh)

Genre(Zh) and Rating(Zh) are datasets in Chinese that were created from a review corpus of the Chinese E-commerce service, JD.com [126][7]. Each review contains a label of the product genre with an attached rating from a user of 1 to 5. Datasets for genre and rating prediction tasks were created as with the English Amazon datasets. Reviews with 13 product genres that had a sufficient number of reviews were extracted and 6,000 reviews were sampled from each genre and type of rating. This study used only reviews that contained more than three and fewer than 100 characters. Table 5.4 lists the contents of the dataset.

Rakuten: Genre&Rating(Ja)

Genre(Ja) and Rating(Ja) are datasets in Japanese that were created from a review corpus of the Japanese E-commerce service, Rakuten [45]. Datasets for genre and rating prediction tasks were created as with the English Amazon datasets. Reviews with 21 product genres that had a sufficient number of reviews were extracted and 5,000 reviews were sampled from each genre and type of rating. This study used only reviews that contained fewer than 100 characters. Table 5.5 lists the contents of the dataset.

[6] http://jmcauley.ucsd.edu/data/amazon/
[7] http://yongfeng.me/dataset/

	Rating					
Genre	1	2	3	4	5	Total
Musical Instruments	109	122	352	991	3,426	5,000
Pet Supplies	254	280	502	848	3,116	5,000
Video Games	359	234	526	1,044	2,837	5,000
CDs and Vinyl	241	221	411	1,011	3,116	5,000
Toys and Games	166	156	474	1,063	3,141	5,000
Sports and Outdoors	159	181	347	1,053	3,260	5,000
Health and Personal Care	241	233	479	956	3,091	5,000
Office Products	97	151	456	1,353	2,943	5,000
Books	191	238	519	1,159	2,893	5,000
Beauty	272	289	523	996	2,920	5,000
Baby	238	296	513	995	2,958	5,000
Electronics	333	216	411	943	3,097	5,000
Patio Lawn and Garden	198	234	588	1,190	2,790	5,000
Automotive	124	146	354	943	3,433	5,000
Cell Phones and Accessories	331	286	564	1,000	2,819	5,000
Grocery and Gourmet Food	195	252	567	1,000	2,986	5,000
Clothing Shoes and Jewelry	206	296	483	1,068	2,947	5,000
Tools and Home Improvement	194	151	387	986	3,282	5,000
Kindle Store	133	143	427	1,241	3,056	5,000
Apps for Android	548	277	582	1,033	2,560	5,000
Home and Kitchen	224	218	338	936	3,284	5,000
Digital Music	251	266	498	1,181	2,804	5,000
Amazon Instant Video	239	232	530	1,121	2,878	5,000
Movies and TV	299	280	526	998	2,897	5,000
Total	5,602	5,398	11,357	25,109	72,534	120,000

Table 5.3: Dataset components of Genre&Rating created from Amazon product data.

	Rating					
Genre	1	2	3	4	5	Total
图书音像	6K	6K	6K	6K	6K	30K
电脑/办公	6K	6K	6K	6K	6K	30K
美妆个护	6K	6K	6K	6K	6K	30K
家用电器	6K	6K	6K	6K	6K	30K
家居生活	6K	6K	6K	6K	6K	30K
其他	6K	6K	6K	6K	6K	30K
母婴/玩具	6K	6K	6K	6K	6K	30K
鞋类箱包	6K	6K	6K	6K	6K	30K
手机/数码	6K	6K	6K	6K	6K	30K
服饰服装	6K	6K	6K	6K	6K	30K
家具/家装/建材	6K	6K	6K	6K	6K	30K
运动户外	6K	6K	6K	6K	6K	30K
钟表/首饰/眼镜/礼品	6K	6K	6K	6K	6K	30K
Total	78K	78K	78K	78K	78K	390K

Table 5.4: Dataset components of Genre&Rating created from JD.com.

	Rating					
Genre	1	2	3	4	5	Total
車用品・バイク用品	5K	5K	5K	5K	5K	25K
ダイエット・健康	5K	5K	5K	5K	5K	25K
ペット・ペットグッズ	5K	5K	5K	5K	5K	25K
靴	5K	5K	5K	5K	5K	25K
スイーツ・お菓子	5K	5K	5K	5K	5K	25K
バッグ・小物・ブランド雑貨	5K	5K	5K	5K	5K	25K
美容・コスメ・香水	5K	5K	5K	5K	5K	25K
インテリア・寝具・収納	5K	5K	5K	5K	5K	25K
日用品雑貨・文房具・手芸	5K	5K	5K	5K	5K	25K
ジュエリー・アクセサリー	5K	5K	5K	5K	5K	25K
食品	5K	5K	5K	5K	5K	25K
ホビー	5K	5K	5K	5K	5K	25K
スマートフォン・タブレット	5K	5K	5K	5K	5K	25K
インナー・下着・ナイトウエア	5K	5K	5K	5K	5K	25K
家電	5K	5K	5K	5K	5K	25K
スポーツ・アウトドア	5K	5K	5K	5K	5K	25K
キッズ・ベビー・マタニティ	5K	5K	5K	5K	5K	25K
キッチン用品・食器・調理器具	5K	5K	5K	5K	5K	25K
メンズファッション	5K	5K	5K	5K	5K	25K
花・ガーデン・DIY	5K	5K	5K	5K	5K	25K
レディースファッション	5K	5K	5K	5K	5K	25K
Total	105K	105K	105K	105K	105K	525K

Table 5.5: Dataset components of Genre&Rating created from Rakuten data.

5.1.2 Settings

Neural Unigram Language Model

For the neural unigram language model in OpTok and OpTok4AT, this study used a two-layered perceptron as MLP in (4.1). The size of word embeddings was 64, and the language model shared word embeddings with the encoder for text classification. The hidden size of the MLP was 96.

Encoders

Two types of neural encoders were used to confirm that the proposed methods were applicable regardless of encoder type: one encoder that uses an attention mechanism and one that uses BiLSTM to compute $\boldsymbol{h}_{s'}$ in (3.5). As Figure 2.1 in Section 2.4.1 shows, this study applied the attention encoder or BiLSTM encoder to the tokenized sentences based on the unigram language model and then fed the outputs to the linear layer. In these procedures, the activation function (hyperbolic tangent) was applied after the linear layer. The output of the BiLSTM encoder was max-pooled before fed into the linear layer. For the attention encoder, this study used a linear layer whose hidden size was 256 to calculate attention weights. For the BiLSTM encoder, the hidden size for each LSTM layer was 256. The size of the sentence vector calculated by each encoder was 256. A dropout was applied to the sentence representations with a rate of 0.5. For SNLI, parameters were shared between two encoders corresponding to the *premise* and *hypothesis* and input the concatenation of each encoded representation into the MLP classifier. The parameters of the tokenizer in the proposed methods were also shared for both the *premise* and *hypothesis*. For the classifier of the downstream model, this study used a three-layered perceptron, which outputs a label-sized vector with a hidden size of 256.

Baselines

This study compared the proposed methods (OpTok and OpTok4AT) with SentencePiece [60], which is a widely used tokenizer. A tokenized sentence was obtained based on SentencePiece and then the tokenized sentence was treated as input to the encoder. In other words, the unigram language model in the proposed methods was replaced with the SentencePiece tokenizer and one tokenized

sentence was used as input to the same architecture. Many studies have reported that training models with stochastic tokenization or subword regularization leads to better performance of the downstream tasks than when training a model using deterministic tokenization [59, 39, 90]. Therefore, this study also trained the encoder and downstream model using subword regularization provided by SentencePiece.

Initialization

The tokenization model of SentencePiece was trained on the training split of each dataset. The sizes of the vocabularies were determined using validation splits among 8,000, 16,000, 24,000, and 32,000 words, and this study selected 16,000 words each for Twitter(Ja), Twitter(En), and Genre&Rating(Zh), and 32,000 words each for Weibo(Zh), SNLI, and Genre&Rating(Ja, En). The coverage of vocabulary was 1.0 to include all characters in vocabulary.

This study also used a vocabulary obtained by SentencePiece as the initial vocabulary of the OpTok and OpTok4AT for each task. The neural unigram language models of OpTok and OpTok4AT were initialized by training them to minimize the KL divergence loss between their distributions of token probabilities and those of token probabilities obtained by SentencePiece. The initialization of the neural unigram language model of the proposed methods was terminated when the KL divergence loss became less than 1e-7 or when the number of the training steps reached 100,000.

The word embeddings were pretrained with a bidirectional language model task on the training split of each dataset, and they were fixed during text classification training. The bidirectional language model was composed of forward and backward LSTMs whose hidden size was 128. The language model was trained on each training corpus of the downstream tasks with cross-entropy loss functions for both directions in the same manner as the general training of neural language models. The number of training epochs was 50. Because the optimal tokenization was unclear during pretraining, the bidirectional language model was trained with sampling tokenization (subword regularization) under each training epoch using SentencePiece. For Genre&Rating tasks, this study used the same word embeddings pretrained on the training split for both genre and rating prediction tasks. This study did not use any outside resource for pretraining other than the

training split.

Training on Downstream Task

The neural unigram language model in OpTok and OpTok4AT and the downstream model were trained using a cross-entropy loss for the gold labels. This study employed Adam [54] to update the parameters with the default settings of PyTorch.

The smoothing hyperparameter of subword regularization α was set to 0.2 for both SentencePiece and OpTok, as recommended in Kudo [59]. For the training of the proposed methods, the size of the N-best tokenization of the proposed methods was $N = 3$, and the size of the restricted vocabulary $|V'|$ was half the size of the initial vocabulary. At the inference, this study used the 1-best tokenization and top-$|V'|$ of the vocabulary based on the language model. Both OpTok and OpTok4AT used the loss function to maintain the characteristics of the language model described in Section 3.5 and the hyperparameter for the loss was set as $\mu = 0.01$. Experiments were conducted five times from a random initialization (excluding the pretrained parameters) and the average F1 scores were included in the results. The maximum number of training epochs was 20. The model exhibiting the highest performance on the validation split was selected and evaluated on the test split for each trial.

5.1.3 Results

Results with the Attention Encoder

Table 5.6 presents the experimental results with the attention encoder on text classification tasks. The results demonstrate that OpTok surpassed the baseline methods using SentencePiece (SP and SP+R), excluding those with the dataset Genre(En). OpTok4AT outperformed OpTok in eight of 10 datasets. The results showed that OpTok4AT was superior to OpTok in text classification with neural networks including the attention encoder.

The table also shows an interesting result. Based on a comparison of the scores of SP and SP+R, subword regularization did not have a positive effect on the performance with the attention encoder. We considered that the structure of the attention encoder was too simple for subword regularization to increase its

robustness. The attention encoder may have hampered the performance of the proposed methods because the methods also included stochastic tokenization in the training of the tokenizer.

Results with the BiLSTM Encoder

Table 5.7 presents the experimental results using the BiLSTM encoder on text classification. As compared to Table 5.6, the scores of BiLSTM encoders were higher than those of the attention encoders. This suggests that the downstream model with BiLSTM can more effectively perform text classification tasks because of its richer structure.

Table 5.7 shows that OpTok outperformed the baseline methods with SentencePiece under all datasets, and OpTok4AT surpassed OpTok under eight datasets. With the other two datasets, the performance of OpTok4AT was comparable to that of OpTok. The difference in the performances of the proposed methods was due to the different strategies employed between the training and downstream models. OpTok trained the downstream model with a weighted sum of sentence vectors corresponding to the N-best tokenization based on their tokenization probabilities, but it used the 1-best tokenization in the inference. This gap may have hampered the performance of the downstream model. By contrast, because OpTok4AT trained the downstream model with only a single sampled tokenization, the downstream model received one tokenization in both training and inference consistently. OpTok4AT improved the performance due to this consistency.

Tables 5.6 and 5.7 show that the proposed methods were capable of improving the performances of two neural text classifiers. These results suggest that the proposed methods are applicable to various settings of text classification using neural networks. This study also confirmed the improved performance of text classification using the more advanced neural network BERT [22], as described in the following Section 6.20.

	SP	SP+R	OpTok	OpTok4AT
Weibo(Zh)	89.44	89.28	89.88*	**90.10***
Twitter(Ja)	80.99	81.16	82.60*	**82.96***
Twitter(En)	70.59	69.50	**71.98**	71.31*
Genre(Zh)	35.43	35.54	37.59*	**38.18***
Rating(Zh)	45.60	45.35	45.67*	**45.97***
Genre(Ja)	33.35	33.81	**36.25***	35.71*
Rating(Ja)	46.48	46.44	46.89	**47.41***
Genre(En)	60.46	60.04	60.23	**61.17***
Rating(En)	57.29	58.17	**59.69***	59.57*
SNLI	66.71	66.03	68.27*	**68.73***

Table 5.6: Experimental results on text classification tasks (F1-score) with the attention encoder. SP and R denote SentencePiece and subword regularization, respectively. The highest scores are highlighted in bold. * indicates that the score was significantly higher than that of the baseline system (SP+R) with McNemar's test ($p < 0.05$).

	SP	SP+R	OpTok	OpTok4AT
Weibo(Zh)	92.70	92.79	92.93	**93.06***
Twitter(Ja)	85.89	86.51	**87.39***	87.27*
Twitter(En)	75.98	77.31	**79.04***	78.63*
Genre(Zh)	44.19	47.95	48.22*	**48.41***
Rating(Zh)	48.96	49.41	49.63*	**49.76***
Genre(Ja)	46.82	47.86	50.21*	**50.79***
Rating(Ja)	51.95	52.30	53.19*	**53.37***
Genre(En)	70.17	71.19	71.88	**71.83**
Rating(En)	66.42	67.53	67.68	**67.90**
SNLI	75.62	76.75	77.04	**77.05**

Table 5.7: Experimental results on text classification tasks (F1-score) with the BiLSTM encoder. SP and R denote SentencePiece and subword regularization, respectively. The highest scores are highlighted in bold. * indicates that the score was significantly higher than that of the baseline system (SP+R) with McNemar's test ($p < 0.05$).

5.2 Machine Translation

This study also conducted experiments on machine translation tasks, which are generation problems. OpTok4AT can be used for machine translation tasks. However, OpTok cannot be used for generation tasks because it requires sentence-level representation and cannot calculate the cross-attention between source and target sentences. Task definitions and commonly used architectures were previously described in Section 2.4.2.

5.2.1 Settings

For machine translation experiments, this study employed IWSLT and WMT corpora on eight language pairs, where one side was English (En) and the other sides were German (De), Vietnamese (Vi), Chinese (Zh), Arabic (Ar), French (Fr), Hungarian (Hu), and Romanian (Ro). Table 5.8 provides overviews of the datasets on the machine translation tasks. As the table shows, this study employed two types of English–German pairs from IWSLT14 and WMT14 for the comparison between the lower resource setting (IWSLT14) and richer resource setting (WMT14).

Following existing studies [19, 44, 116], all the datasets except for the Chinese corpus were pre-tokenized with the Moses tokenizer[8], and the Chinese corpus was pre-tokenized with jieba[9]. The performance of each method was evaluated using detokenized BLEU with SacreBLEU [89].

Regarding recent tokenizers for machine translation, this study compared the proposed method both with SentencePiece and DPE [37], which tokenizes a target sentence while considering the source tokenization. The official implementation of DPE[10] was employed and the tokenization model of DPE was trained using SentencePiece tokenization. As with text classification, subword regularization was used as a strong baseline.

For the downstream model, this study used Transformer [112] implemented in Fairseq [81]. For the IWSLT dataset, Transformer (small) was used and the initial vocabulary was created using SentencePiece with a 16,000 vocabulary size for each language. For the WMT dataset, Transformer (base) was employed, and

[8] `https://github.com/moses-smt/mosesdecoder`
[9] `https://github.com/fxsjy/jieba`
[10] `https://github.com/xlhex/dpe`

Dataset	Language Pair	Train	Validation	Test	Total
IWSLT14	De-En	160,239	7,283	6,750	174,272
IWSLT15	Vi-En	130,933	768	1,268	132,969
	Zh-En	209,941	887	1,261	212,089
IWSLT17	Ar-En	235,527	888	1,205	237,620
	Fr-En	236,653	890	1,210	23,8753
WMT09	Hu-En	1,517,584	2,051	3,027	1,525,187
WMT14	De-En	4,520,620	3,000	3,003	4,526,623
WMT16	Ro-En	612,422	1,999	1,999	616,420

Table 5.8: Overviews of datasets on machine translation tasks. The table shows the number of sentences in each split of a dataset.

the vocabulary size was 32,000. The coverage of vocabulary was 1.0 to include all characters in vocabulary. Similar to text classification tasks, the proposed neural unigram language model was initialized with the results of SentencePiece. The hyperparameters for subword regularization were $\alpha = 0.2$ for IWSLT, $\alpha = 0.5$ for WMT, and $k = \infty$ for both datasets. The number of tokenizations for the training of the proposed method was $N = 8$ for ISWLT and $N = 3$ for WMT. The translation model was trained with 100 epochs and the model that scored the highest on the validation split was selected for the evaluation of the test split. A beam-search was employed for the decoding with a beam width of 5. The other hyperparameters were selected based on the existing implementation for machine translation using Fairseq[11]. Table 5.9 summarize the hyperparameters.

In the training of the neural machine translation model with DPE, subword regularization was applied for the source-side language, similar to He et al. [37]. For the proposed method, this study prepared three configurations with the proposed method: for only a source-side language, for only a target-side language, and for both side languages. When the proposed method was applied for a single side of the translation pair, subword regularization was used for the other side as well as the training of DPE.

[11] `https://github.com/pytorch/fairseq/tree/master/examples/translation`

Parameter	$\text{Transformer}_{\text{small}}$	$\text{Transformer}_{\text{base}}$
Encoder Embedding Size	512	512
Encoder FFN Embedding Size	1,024	2,048
Number of Encoder Attention Heads	4	8
Number of Encoder Layers	6	6
Decoder Embedding Size	512	512
Decoder FFN Embedding Size	1,024	2,048
Number of Decoder Attention Heads	4	8
Number of Decoder Layers	6	6
Clipping Norm		0.0
Dropout Rate		0.3
Weight Decay		0.0001
Max Tokens for Mini-Batch		1,000
Optimizer		Adam
β_1 and β_2 for Adam		0.9, 0.98
Learning Rate		0.0005
Learning Rate Scheduler		Inverse Square Root
Warming-Up Updates		4,000

Table 5.9: Overviews of datasets on machine translation tasks. The table shows the number of sentences in each split of a dataset.

5.2.2 Results

Table 5.10 presents the detailed performances of the three configurations, showing that the system employing the proposed approach (OPT/SP+R, SP+R/OPT, or OPT/OPT) achieved the best performances with most datasets. The proposed method when used only with the decoder side (SP+R/OPT) succeeded on many datasets. By contrast, when the proposed method with both sides (OPT/OPT) was used, the performance degraded. These results suggest that optimizing the tokenization of source and target languages simultaneously is challenging and can degrade performance. The simultaneous optimization of source and target languages on machine translation is later described in Section 6.2.

The scores revealed that the proposed method outperformed DPE in most datasets. Although the performance of the proposed method was poorer than

that of DPE in the Hu-En pair, the performances of the two models were mostly comparable. These results demonstrated that the proposed method improved the performance on machine translation to a greater extent than did the existing method. Moreover, unlike DPE, which is limited to the encoder side, the proposed method was shown to be applicable to both the encoder and decoder sides.

	Encoder Decoder	SP SP	SP+R SP+R	SP+R DPE	OPT SP+R	SP+R OPT	OPT OPT
IWSLT14	De-En	33.79	35.03	35.02	34.90	**35.78***	35.13*
	En-De	28.09	29.13	29.39	29.56	**29.57***	29.30
IWSLT15	Vi-En	28.70	28.78	28.85	29.34*	**29.69***	29.44*
	En-Vi	30.87	31.60	31.63	31.41	**31.74***	31.70*
	Zh-En	20.44	21.17	21.38	21.63*	21.65*	**21.89**
	En-Zh	14.40	15.25	15.21	15.45	**15.59**	15.31
IWSLT17	Ar-En	29.23	29.39	29.37	29.48	**30.04**	29.78
	En-Ar	15.45	17.75	17.83	**18.49***	18.18	18.21*
	Fr-En	37.87	38.43	38.52	**38.82**	38.68	38.58
	En-Fr	37.95	39.83	39.90	40.01*	**40.08***	39.68
WMT09	Hu-En	14.84	15.51	**15.75**	15.73	15.74	15.60
	En-Hu	11.02	12.14	12.30	12.30	**12.37**	12.33
WMT14	De-En	31.46	31.89	31.97	**32.19***	31.98*	31.90
	En-De	27.10	27.41	27.49	**27.62**	27.52	27.44
WMT16	Ro-En	29.10	31.79	31.80	31.80	**31.83**	31.72
	En-Ro	21.78	24.05	24.29	24.36	**24.53***	24.03

Table 5.10: Results of experiments on machine translation task using IWSLT and WMT corpus (BLEU). We show the tokenization methods for the encoder and decoder. SP and R denote SentencePiece and subword regularization, respectively. OPT refers to the proposed method of OpTok4AT. The highest scores are highlighted in bold. * indicates that the score was significantly higher than that of the baseline system (SP+R/SP+R) with a statistical significance estimation using bootstrap resampling [55] ($p < 0.05$).

6 Discussion

6.1 Performance Improvement by Tokenization

6.1.1 Performance Improvement for a Randomly Initialized Classifier

Whether the optimized tokenization led to the improvement described in Chapter 5 remains unclear, as all components were trained simultaneously. Therefore, this study next investigated whether the optimized tokenization contributed to improved performance of the downstream task. To validate the effect of only tokenization, only the neural unigram language model in OpTok and OpTok4AT was trained on the text classification task. In other words, the parameters of the BiLSTM encoder in the downstream model were fixed using random initialization. This study then checked the improvements in training loss and F1 score on the validation split by updating only the parameters of the neural unigram language model for tokenization.

Experiments on Twitter(Ja) were conducted under the same settings as described in Section 5.1, where the results are presented in Figure 6.1. Figure 6.1 shows the differences in training loss and validation F1 score from the values at the beginning of the training. This figure shows that the training loss decreased with the number of epochs, whereas the validation F1 score increased for both OpTok and OpTok4AT. The results indicate that the proposed methods identified the appropriate tokenization, which improved task performance, and suggest that the optimized tokenization contributed to improved performance overall.

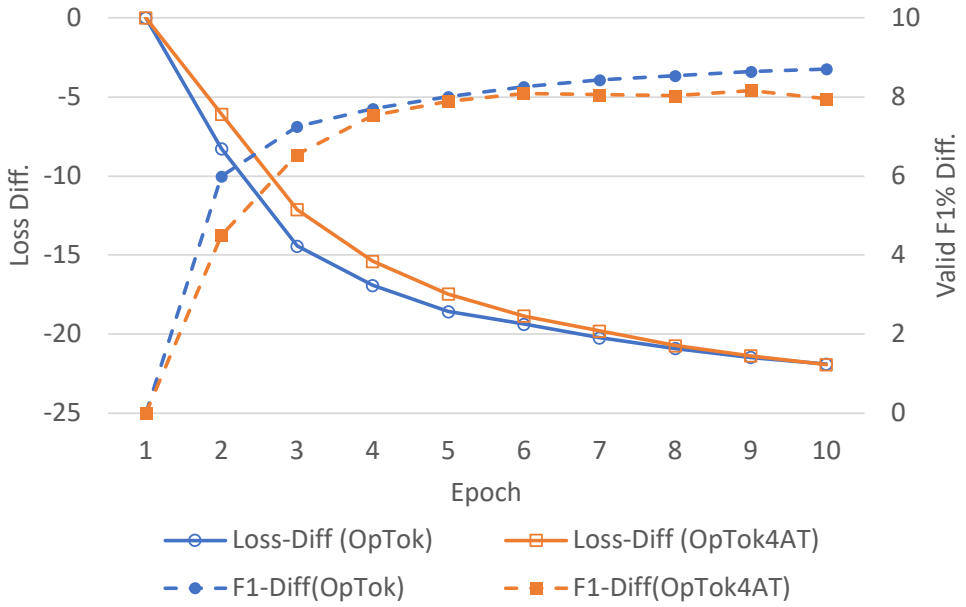


Figure 6.1: Average improvement (difference from the values at the beginning of the training) of validation F1 score and training loss on Twitter(Ja) over five trials when only tokenization with OpTok and OpTok4AT was updated.

6.1.2 Tokenization as Post-processing

Settings

As described in Section 3 and 4 concerning the proposed methods and in Section 6.1.1 on the experiment conducted with the frozen downstream model, we can use the proposed methods for various architectures even if their parameters are frozen. Thus, the proposed methods can be applied as postprocessing to an already trained model. This study next evaluated the effectiveness of optimizing the proposed tokenizer for the trained model. The neural unigram language model of OpTok and OpTok4AT was trained with the final loss value without updating the parameters of the downstream model.

Experiments were conducted on text classification (sentiment analysis) and machine translation (IWSLT15) tasks. The downstream models described in Chapter 5 were trained with subword regularization [59]. The models were trained with 30 and 100 epochs for text classification and machine translation, respec-

tively. Only the tokenizer were then trained with five epochs using the loss values computed by the trained models.

In the experiment on Twitter(Ja), this study additionally used the downstream model trained with tokenization by MeCab [58], which is the well-known Japanese tokenizer that uses a dictionary. With this model, this study attempted to confirm that the proposed methods could be applied to a model trained with a tokenizer other than SentencePiece. The downstream model was trained for 30 epochs using the tokenization by MeCab, and the proposed method was applied for five epochs with the settings previously mentioned. Subword regularization was not used because MeCab does not have a language model for sampling the tokenization. To initialize the neural unigram language model of the proposed methods, the unigram language model was built by counting the number of words in the corpus tokenized by MeCab. To reduce the number of low-frequency words, words that appeared only once in the corpus were replaced with the special token indicating the unknown token.

In the experiment on Twitter(En), this study also used BERT for the downstream model to confirm that the proposed method was applicable to the large pretrained language model[1]. The downstream model for text classification was replaced with $\text{BERT}_{\text{base}}$ [2]. After fine-tuning the downstream model on each task for 30 epochs, only the tokenization was optimized for five epochs with the proposed methods. In addition to the experiment using MeCab, this study did not use subword regularization because the BERT published by HuggingFace employs the WordPiece tokenizer and does not contain any language model. The neural unigram language model of the proposed methods was initialized using the unigram language model built by counting the words in the corpus tokenized by WordPiece.

Results

Table 6.1 details the performances of the methods. The table shows that the proposed methods (OpTok and OpTok4AT) improved the performance from the base model trained with subword regularization (Base Model). OpTok4AT out-

[1]The experimental settings in which BERT [22] was used for end-to-end training rather than postprocessing are described in Section 6.9.

[2]HuggingFace: `https://github.com/huggingface/transformers`

	Base Model	OpTok	OpTok4AT
Sentiment Analysis (F1)			
Weibo(Zh, SentencePiece)	92.69	**93.08**	92.99
Twitter(Ja, SentencePiece)	85.88	86.23	**86.28**
Twitter(Ja, MeCab)	85.09	85.68	**85.96**
Twitter(En, SentencePiece)	77.21	77.41	**77.77**
Twitter(En, BERT)	81.88	81.89	**81.99**
IWSLT15 (BLEU)			
Vi-En	28.82	-	**28.91**
En-Vi	30.48	-	**30.60**
Zh-En	21.55	-	**21.82**
En-Zh	14.57	-	**14.83**

Table 6.1: Performance improvements when tokenization was optimized as post-processing by OpTok and our method. *Base Model* denotes a model trained with SentencePiece, Mecab, or BERT without optimizing tokenization. The highest scores are highlighted in bold.

performed OpTok on two datasets of text classification. In addition, on machine translation tasks, OpTok4AT consistently improved the BLEU scores. These results show that the proposed methods are useful in improving the performance of the downstream model even when a sufficiently trained model is used as the downstream model.

In the experimental setting with MeCab, the proposed methods improved the performance, which suggests that the proposed methods can be used in situations in which the downstream model is trained with a dictionary-based tokenizer. In addition, the proposed methods slightly improved the performance even under the setting with BERT. The proposed method can thus be used in various situations and improves the postprocessing performance regardless of the tokenizer used in the training of the downstream model.

6.2 Learning Both Encoder and Decoder

6.2.1 Settings

The results of the machine translation task (Section 5.2) revealed that the performance decreased when the proposed method was incorporated into both the encoder and decoder sides. The cause of this decrease was considered as the gap in the tokenization strategy between the source and target languages. As described in the following Section 6.4.2, the proposed method tokenizes the source- and target-side sentences into fine- and coarse-grained granularities, respectively. This study next attempted to train the proposed method on both the encoder and decoder sides simultaneously and in a stable manner using three possible strategies. In each strategy, the neural unigram language and downstream models were trained on machine translation simultaneously, and the total number of training epochs was 100.

Enc→Dec

Only the encoder-side neural unigram language model was trained for the first 50 epochs, with the decoder-side neural unigram language model being frozen; the decoder-side neural unigram language model was then trained for the last 50 epochs, with the encoder neural unigram language model being frozen.

Dec→Enc

The proposed method was trained using the reversed version of the Enc→Dec strategy. The decoder and encoder sides of the neural unigram language model were trained for the first and last 50 epochs, respectively.

Random

In each mini-batch training, the neural unigram language model was randomly trained on either the encoder or decoder side at a 0.5 ratio.

	Both	Enc→Dec	Dec→Enc	Random
Vi-En	29.44	**30.22***	29.47	29.37
En-Vi	31.70	**31.78**	31.33	31.70
Zh-En	21.89	**21.99**	21.82	21.66
En-Zh	15.31	**15.54**	14.88	15.14

Table 6.2: Performances of machine translation on the IWSLT15 datasets using three strategies for the simultaneous training of our method. The scores for "Both" are taken from the "OPT/OPT" column in Table 5.10. The highest scores are highlighted in bold. * indicates that the score was significantly higher than that of Both with a statistical significance estimation using bootstrap resampling [55] ($p < 0.05$).

6.2.2 Results

Table 6.2 presents the results of the experiments. These results indicate that the Enc→Dec strategy contributed to an improved performance of the simultaneous learning of tokenization on both sides. In particular, the scores of Vi-En, En-Vi, and Zh-En surpassed the best scores reported in Table 5.10, indicating that the Enc→Dec strategy was effective in the training of the proposed method. By contrast, the Dec→Enc strategy decreased the performance on many language pairs. The performance obtained using the random strategy was slightly poorer than that when using the original method (*Both* in the table). These results suggest that instead of optimizing both sides simultaneously, learning the tokenization from each side step-by-step and, more specifically, from the encoder to the decoder side, was more effective for machine translation tasks.

6.3 Comparison with Ideal Tokenization

This study focuses on the method to optimize tokenization for downstream tasks and models. Chapter 5 and Section 6.1.2 demonstrate that the performance of the downstream model improved by optimizing tokenization from scratch and already trained models, respectively. This raises a question: what is the upper bound of the performance improvement that can be obtained by optimizing tokenization?

This section analyzes the gap between the performance obtained by the pro-

posed methods and the one obtained by the ideal tokenization that is completely adapted to the evaluation data. This analysis used the validation split of sentiment analysis datasets: Weibo(Zh), Twitter(Ja), and Twitter(En).

The ideal tokenization was obtained by the brute-force search for the trained downstream model. In concrete, the downstream model was trained on the training split with subword regularization, and a tokenization that predicts the higher probability for the correct label was selected based on the trained model and the validation split. The N-best tokenization mode of SentencePiece was used to identify the ideal tokenization. The size of N-best tokenization was $N = 1,024$. Note that the N herein is different from the hyperparameter of the proposed methods. Because the ideal tokenization was selected based on the correct label, the performance with this tokenization is the upper bound that can be achieved by optimizing tokenization. This section compares the performance of OpTok and OpTok4AT with this upper bound.

Table 6.3 shows the maximum performance[3] on the validation split by each tokenization method. *SP+R* is the baseline model trained with subword regularization. *Post-Processing* in the table indicates that the tokenizations were optimized with the proposed method as post-processing (Section 6.1.2) based on the trained model (SP+R). *Scratch* in the table indicates that the tokenizations were obtained with the default learning with the proposed method used in Chapter 5. *Brute-force* shows the performance with the ideal tokenizations obtained with the aforementioned strategy.

Table 6.3 indicates that the proposed methods improve the validation performance in the post-processing settings, and more improvement was obtained with the scratch settings. However, there is a large gap between the performances of the proposed methods and the ideal tokenization. Although we cannot achieve exactly the same score as the brute-force because it uses the correct label, this result implies that there is still room for further performance improvement by optimizing tokenization. The possible improvement can be to use the more complex architecture for the tokenizer, such as bi- or tri-gram language models or sequential labeling architectures with neural networks.

[3]The other experiments show the test performance of models selected based on the validation split.

		Post-Processing		Scratch		
	SP+R	OpTok	OpTok4AT	OpTok	OpTok4AT	Brute-force
Weibo(Zh)	92.89	92.97	92.97	93.10	93.22	98.46*
Twitter(Ja)	86.25	86.57	86.29	87.31	87.22	95.47*
Twitter(En)	75.82	75.88	77.97	79.60	79.52	95.22*

Table 6.3: F1 scores on the validation split of Weibo(Zh), Twitter(Ja), and Twitter(En) with the BiLSTM-based classifier. * indicates that the score was significantly higher than that of the other methods with McNemar's test ($p < 0.05$).

6.4 Analysis of Tokenization

6.4.1 Optimized Tokenization on Text Classification

Task Oriented Tokenization

This study was also interested in whether the optimized tokenization was different when different downstream tasks were considered. Accordingly, this section analyzed the results of the Genre&Rating prediction in three languages (Japanese, English, and Chinese; see Section 5.1). The dataset contained two tasks linked to the same review corpus.

Tables 6.4 and 6.5 and 6.6 show the rankings of tokens whose probability significantly increased from the initial values on the genre and rating prediction tasks with OpTok and OpTok4AT. The optimized neural unigram language model assigned higher scores to tokens, which seemed useful for all tasks in all of the languages. For example, in English (Table 6.6), the OpTok trained on the genre prediction task assigned higher probabilities for words such as *gun* and *zombie* that may be useful in predicting the genre label. By contrast, when OpTok was trained on the rating prediction task, the trained language model yielded higher probabilities for words such as *bad* and *awesome*, which seemed important for rating prediction. A similar tendency was observed with the Japanese and Chinese datasets. In Chinese (Table 6.4), words such as 足 (foot) and 车 (car) yielded a higher probability for the genre prediction task, whereas the language model trained on rating prediction assigned a higher probability to words such as 清 (clean) and 差的 (poor). OpTok also yielded a higher probability in Japanese

OpTok				OpTok4AT			
Genre		Rating		Genre		Rating	
Token	Diff%	Token	Diff%	Token	Diff%	Token	Diff%
的	7.58	清	4.99	的	9.62	的	8.88
了	3.96	时	2.44	比	3.22	感觉	6.45
足	3.32	原	2.01	套	2.05	真的	4.47
买	2.32	实在	1.81	车	1.97	还是	2.74
气	2.18	也	1.73	在	1.45	非常	2.57
车	2.14	大概	1.79	厚	1.55	快	1.63
衣	1.98	出来的	1.77	买	1.27	质量	1.49
的那	1.84	差的	1.56	张	1.30	了	0.57
在	1.38	货	1.53	刮	1.20	要	1.33
不出	1.41	比较	1.51	给我	1.17	很好	1.25
细	1.39	橡	1.48	起来	1.17	后	1.12
饰	1.32	象	1.39	扣	1.15	我	0.97
子	1.25	洁	1.38	衣服	1.06	确实	1.10
钟	1.17	从	1.16	手机	1.02	能	1.02
电	1.09	傅	1.16	了	0.00	有些	0.91

Table 6.4: Token rankings based on the positive differences in probabilities between the initial and learned language models of the proposed methods on Genre&Rating(Zh). The downstream model is the classifier with the BiLSTM encoder.

(Table 6.5) for useful words according to task (e.g., 眠 (sleep) and 足 (foot) for the genre prediction and とても (very) and 星 (rating) for the rating prediction). The language model of OpTok increased the probability of functional words such as 的 (of) and 了 (done) in Chinese and の (of) in Japanese. Chinese and Japanese texts do not have word boundaries (as typically indicated by whitespaces) and do not have many subwords composed of a noun and functional word such as 车的 (of car) and 足の (of foot). The results suggest that the proposed methods assigned the higher probabilities to these functional words to cut them from the noun that is useful in predicting the correct label. Similarly, even on the tasks in English that indicated word boundaries with whitespaces, we observed the language model of the proposed methods increased the word probabilities of the functional suffixes such as “s” for the plural form and “ly” for adverbs.

OpTok				OpTok4AT			
Genre		Rating		Genre		Rating	
Token	Diff%	Token	Diff%	Token	Diff%	Token	Diff%
が	6.80	とても	2.57	る	10.09	して	7.16
の	5.99	星	2.19	水	8.03	少し	2.89
て	3.27	があるもの	1.45	の	5.33	のに	2.70
への	2.22	感じ	1.42	が	3.96	とても	2.36
汚れ	2.11	全然	1.39	に	4.02	目	1.88
眠	1.94	今も	1.38	っ	2.04	まだ	1.64
足	1.84	初めて	1.32	を	1.49	でした	1.21
食べ	1.76	もなく	1.22	て	1.26	なく	1.34
もの	1.75	広く	1.20	味	1.32	し	1.10
履き	1.53	気になる点	1.19	いる	1.03	商品が	1.29
使用	1.49	殆ど	1.13	箱	0.96	を	0.83
生地	1.47	留まり	1.07	盛	0.91	いる	1.19
です	1.28	まったく	1.02	感	0.82	いた	1.20
綺麗に	1.37	のが	0.98	丁	0.72	いい	1.17
たりする	1.27	全く	0.95	上	0.68	ので	0.89

Table 6.5: Token rankings based on the positive differences in probabilities between the initial and learned language models of the proposed methods on Genre&Rating(Ja). The downstream model is the classifier with the BiLSTM encoder.

This increase in the probability of functional words may have contributed to a performance improvement by dividing the useful nouns and stem words for the downstream tasks.

The rankings of OpTok4AT were similar to those of OpTok, where the language model assigned higher probabilities to words that were useful for solving the downstream model. These results demonstrate that OpTok and OpTok4AT optimized the tokenization to enable helpful tokens to be used frequently. Section 6.7 further discusses the relationship between the word rankings of the language model and word importance to solve the downstream task when using the simple classifier.

An example of optimized tokenization was extracted from the training split in the Chinese datasets, which included the difference in tokenization derived from

OpTok				OpTok4AT			
Genre		Rating		Genre		Rating	
Token	Diff%	Token	Diff%	Token	Diff%	Token	Diff%
gun	3.47	However	14.10	s	10.95	ll	17.75
grip	2.61	BUT	11.69	scent	4.24	well	7.41
zombie	2.26	bad	5.32	movie	2.74	best	6.05
professional	1.90	paced	3.66	relationship	1.45	bad	5.29
treat	1.69	Funk	2.99	soundtrack	0.99	great	3.36
gray	1.48	awesome	2.84	chair	0.98	consecutive	3.42
soap	1.48	Ok	2.60	cord	0.93	ly	2.86
dry	1.33	watch	2.08	mouth	0.93	remorse	2.13
collection	0.97	game	2.05	product	0.80	good	1.54
sleeper	0.94	Build	1.89	threaded	0.67	guess	1.25
instant	0.77	daughter	1.85	knife	0.66	worthwhile	1.18
phone	0.73	great	1.67	book	0.56	often	0.95
tea	0.68	There	1.59	quality	0.59	smart	0.95
scary	0.65	brand	1.38	jacket	0.65	Geffen	0.94
riddled	0.63	what	1.22	game	0.51	beautifully	0.78

Table 6.6: Token ranking based on positive differences in probabilities between the initial and learned language models of the proposed methods on Genre&Rating(En). The downstream model is the classifier with the BiLSTM encoder.

tasks listed in Table 6.8. In the tokenization optimized for the genre prediction task, both proposed methods extracted 防滑 (anti-slip) by cutting off the word 不 (not) from the initial tokenization (SentencePiece). The word 防滑 is often used in reviews of carpets, which is one of 家居生活 (household), and the example suggests that the model selected the tokenization that included this word to predict the correct label. By contrast, the tokenization optimized for rating prediction included sentiment words in both proposed methods, such as 不好 (bad) and 完全 (extremely), which correspond to lower ratings.

The Japanese examples listed in Table 6.9 showed similar tendencies as those the Chinese examples. The tokenization for the genre prediction extracted task-specific words such as 香り (fragrance) and 髪 (hair), which correspond to the correct label “美容・コスメ・香水” (Beauty, Cosmetic, Perfume). The tok-

Method	Tokenization
SentencePiece	I like to listen to **CDs** when **traveling** and this is a one of my **favorites** .
Genre (Gold: CDs and Vinyl)	
OpTok	I like to listen to **CD s** when **travel ing** and this is a one of my **favorites** .
OpTok4AT	I like to listen to **CD s** when **traveling** and this is a one of my **favorites** .
Rating (Gold: 5)	
OpTok	I like to listen to **CDs** when **traveling** and this is a one of my **favorite s** .
OpTok4AT	I like to listen to **CDs** when **traveling** and this is a one of my **favorites** .

Table 6.7: Differences in tokenization depending on the downstream task (Genre or Rating prediction) in English.

enization for the rating prediction extracted 全然 (at all), which is a negative polarity expression that co-occurs with a negative expression (e.g., なし (not) in this case).

The English examples listed in Table 6.7 also exhibited similar behaviors as those of the others. In the tokenization optimized for the genre prediction task, the model cut off an inflection of *CD-s* to generalize the token *CD* for predicting the proper genre. Furthermore, the tokenization of OpTok for the rating task cut off the inflection of *favorite-s* to generalize *favorite* to predict the higher ratings. The tokenization of OpTok for the genre prediction also extracted *travel* by cutting off *ing*, which corresponds to the improper label *Sports and Outdoors*. The proposed methods cannot change the tokenization depending on the context of the sentence because the unigram language model is used for the tokenization. Although this is not a major problem when using a rich downstream model, which can consider the context, these characteristics of the proposed methods may cause failed predictions such as when wrongly predicting the *Sports and Outdoors* label instead of *CDs and Vinyl* using the extracted word *travel* for a sentence.

These examples in the three languages demonstrate that both versions of the proposed method can tokenize sentences in different ways depending on the downstream tasks. The tokenizations of OpTok and OpTok4AT were shown to be

Method	Tokenization
SentencePiece	东西 非常不好！完全 不防滑！
Genre (Gold: 家居生活*)*	
OpTok	东西 非常不好！完全不 防滑！
OpTok4AT	东西 非常不好！完全不 防滑！
Rating (Gold: 1)	
OpTok	东西 非常 不好！完全 不防滑！
OpTok4AT	东西 非常 不好！完全 不防滑！

Table 6.8: Differences in tokenization depending on the downstream task (genre or rating prediction) in Chinese. The text translates as "This is extremely bad! (This is) not anti-slip at all!", where words in parenthesis are omitted in the original text.

different in some cases because of the differences in architectures and the random initialization of the downstream models.

Tokenization Granularity

This study next confirmed the tokenization tendencies by the proposed method in terms of granularity. Specifically, this subsection compared the number of tokens in the tokenized sentences between the initial tokenization by SentencePiece and that by the proposed methods OpTok and OpTok4AT.

Table 6.10 shows the ratio of the number of tokens between the initial and optimized tokenizations on the E-commerce review datasets. In the table, a value greater than 1.0 indicates that the number of tokens in the optimized tokenization increased over that of the initial tokenization. For example, the tokenization optimized to Genre(Zh) with OpTok was 1.5405 times longer than the initial tokenization by SentencePiece.

The values listed in Table 6.10 demonstrate that tokenizations optimized by the proposed methods became longer than the initial tokenizations with all datasets and languages. The results also demonstrate that both OpTok and OpTok4AT had the same tendency of tokenization in terms of granularity. This means that the proposed methods split words into tiny units to capture a greater amount of small information from sentences for text classification tasks. Interestingly, for all datasets, the tokenizations optimized to genre prediction tasks were longer

Method	Tokenization
SentencePiece	**香りは** すき だけど 、 痛 んだ **髪に は全然 効果なし** 。
Genre (Gold: **美容・コスメ・香水***)*	
OpTok	**香り は** すき だけど 、 痛 んだ **髪 に は全然 効果 なし** 。
OpTok4AT	**香り は** すき だけど 、 痛 んだ **髪 に は全然 効果 なし** 。
Rating (Gold: 2)	
OpTok	**香りは** すき だけど 、 痛 んだ **髪に は 全然 効果なし** 。
OpTok4AT	**香りは** すき だけど 、 痛 んだ **髪に は 全然 効果なし** 。

Table 6.9: Differences in tokenization depending on the downstream task (genre or rating prediction) in Japanese. The text translates as "(I) like the fragrance, but (it does) not do anything for (my) damaged hair at all.", where words in parenthesis are omitted in the original text.

than those for rating prediction tasks in all languages. These results suggest that information of small pieces such as small subwords was important in solving genre prediction tasks effectively for the downstream model, and the proposed methods caused the granularity of tokenization to become much smaller for the effective training. More specifically, the proposed methods split suffixes such as "-s" in the aforementioned extracted nouns that strongly corresponded to each genre label, and the length of tokenization increased.

6.4.2 Optimized Tokenization on Machine Translation

This study next analyzed the tokenization obtained using the proposed method on a machine translation task. Table 6.11 presents a comparison of tokenizations with SentencePiece, DPE, and the proposed method. The IWSLT15 Zh-En corpus was utilized for this comparison and English-side sentences were tokenized using each method. In this comparison, only the English-side tokenization was optimized with the proposed method.

Table 6.11a presents a comparison of the tokenization on the source side between SentencePiece and the proposed method. The proposed method splits words into smaller segments as compared to SentencePiece (where the latter performs the initial tokenization in the proposed method). Specifically, the proposed method cuts off the suffix from a stem word. For example, it splits "don" into "do-n", "have" into "hav-e", and "hours" into "hour-s".

	OpTok	OpTok4AT
Genre(Zh)	1.5405	1.5137
Rating(Zh)	1.4249	1.3807
Genre(Ja)	1.5250	1.5834
Rating(Ja)	1.3224	1.2742
Genre(En)	1.0620	1.0845
Rating(En)	1.0415	1.0305

Table 6.10: Ratio of the number of tokens between the initial tokenization (SentencePiece) and the optimized tokenization (OpTok and OpTok4AT) on the corpora of E-commerce reviews. The downstream model is the classifier with the BiLSTM encoder.

Table 6.11b presents a comparison of the tokenization on the target side between SentencePiece, DPE, and the proposed method. Compared to the tokenization on the source side, the proposed method does not split words into tiny units on the target side. The proposed method exhibits the same tendency of tokenization as DPE, such as splitting the past-verb suffix "-ed". However, DPE tokenization contains smaller units than the tokenization of the proposed method. An example of this is the difference in the tokenization for the word "away".

Tokenization Granularity

To compare the granularities of each tokenizer, this subsection presents the number of tokens in the corpus tokenized by each method. Table 6.12 lists the ratio of the number of tokens in the training corpus between the initial tokenization (SentencePiece) and the optimized tokenization (DPE and the proposed method). In the table, a value greater than 1.0 indicate an increase in the number of tokens as compared to SentencePiece.

The results revealed that the number of tokens in the proposed method increased for the source-side tokenization, which means that the proposed method tokenizes a source corpus into small units by splitting morphemes, as shown in Table 6.11a. This suggests that tokenizations containing small units do not hurt the performance of NMT because the neural encoder can handle fine-grained inputs. This tendency supports the existing studies that showed that using character-level tokenization for the encoder contributes to an improvement in machine transla-

tion performance [51, 64].

For the tokenization of the target side, the ratio of the number of tokens for the proposed method was slightly smaller than the initial tokenization, except for the En-Zh pair. This result suggests that the proposed method seeks an appropriate tokenization to assist in the decoding process while maintaining the granularity of the initial tokenization (e.g., separate-d and separat-ed in Table 6.11b). Regarding the translation of the En-Zh pair, the proposed method split a Chinese sentence in the target side into smaller tokens. Chinese characters contain much more information than English characters, and fewer Chinese tokens are used in a sentence as compared to English. This difference possibly caused the increased number of tokens on the target side to use the same granularity as the source English corpus.

Compared with the tokenization by the proposed method, the number of tokens for the DPE varied for each language pair. DPE tokenization proved to be more flexible than the proposed method because DPE employs Transformer and a special decoding algorithm for tokenization, whereas OpTok4AT simply uses a unigram language model and the Viterbi algorithm. In addition, DPE tokenizes the target sentence by directly considering the source tokenization in which a source sentence is input into Transformer. By contrast, OpTok4AT uses the target-side neural unigram language model trained with information from both sides to identify the target-side tokenization. Although the flexibility of tokenization with OpTok4AT is limited, the proposed method improves the performance on NMT tasks, as the experimental results showed. The proposed method cannot change the tokenization depending on the context because of the unigram language model, which does not consider the context to calculate the probabilities of tokenizations. However, this characteristic of the proposed method also represents an advantage that we can use for consistent tokenization in the entire corpus. The performance improvement in machine translation tasks (see Table 5.10) suggests that this characteristic of the proposed method using a unigram-based language model contributes to improved performance.

SentencePiece	Student s **don ’ t have** long **hours** of learning .
OpTok4AT	Student s **do n ’ t hav e** long **hour s** of learning .
Target	学生 在 校 学习 时间 不 长 。

(a) Tokenization difference for the source language (En-Zh)

Source	引力 与 其它 力 分 隔 开来
SentencePiece	Gra vity **separate d away** from the other force s .
DPE	Gra vity **separat ed a way** from the other force s .
OpTok4AT	Gra vity **separat ed away** from the other force s .

(b) Tokenization difference for the target language (Zh-En)

Table 6.11: Comparison of English tokenizations on Zh-En pairs using SentencePiece (SP), DPE, and our method. Different results of tokenizations are highlighted in bold.

Encoder Decoder	OPT SP+R	SP+R OPT	SP+R DPE
IWSLT14			
De-En	2.5353	0.9992	1.0439
En-De	1.3809	0.9996	0.9923
IWSLT15			
Vi-En	1.5320	0.9993	1.0428
En-Vi	1.4650	0.9999	0.9923
Zh-En	1.5175	0.9994	0.9907
En-Zh	1.3516	1.4713	1.0346
IWSLT17			
Ar-En	2.5350	0.9997	0.9952
En-Ar	1.4765	0.9994	0.9945
Fr-En	1.7194	0.9996	1.0001
En-Fr	1.5996	0.9997	0.9935

Table 6.12: Ratio of the number of tokens between initial tokenization (SentencePiece) and optimized tokenization (DPE and our method) on the IWSLT corpora. SP+R denotes SentencePiece with subword regularization.

6.5 Cross-domain Evaluation

The experimental results described in Section 6.1.2 demonstrated that the proposed methods can improve the performance of the downstream model simply by optimizing the tokenization. This fact indicates that the proposed methods are types of domain adaptation, as OpTok and OpTok4AT refine the tokenization based on the domain of the downstream model and task. This section next conducted a thorough analysis on whether tokenizations optimized by OpTok and OpTok4AT are specialized to the given downstream model and task. This subsection compares the performances of the downstream model when 1) the tokenization is optimized to the same task used to train the downstream model, and 2) the tokenization is optimized to a different task.

This comparison exploits the dataset of genre and rating predictions used in the experiment on text classification tasks. As mentioned in Section 5.1, the genre and rating prediction tasks are created from the same single review corpus. Therefore, this analysis can compare the effects of tokenization on the downstream task.

The downstream model with the BiLSTM encoder was trained for each downstream task using subword regularization of SentencePiece as well as experiments in 6.1.2. After the 30-epoch training, the parameters of the downstream model were frozen, and the tokenization was optimized to the same downstream task using the proposed method with the trained downstream model. This study then compared the performances of the downstream model with tokenizations for the same downstream task and for a different task. For example, the downstream model was trained and the tokenization was optimized for the genre prediction task.

Similarly, the downstream model and tokenization were trained for the rating prediction task. This study then compared the performance of the downstream model for the genre prediction task with the tokenization derived from genre and rating predictions. This analysis hypothesized that if the tokenization by the proposed method is specialized to the genre prediction, the genre prediction model with tokenization for the genre prediction should perform better than with tokenization for rating prediction. OpTok4AT was used for the analysis because OpTok4AT proved to be superior to OpTok in most of the previous experiments (Section 5.1).

Table 6.13 presents the experimental results on the review datasets in Chinese, Japanese, and English. The first row of the table shows the performances of the downstream model trained on the genre prediction task of JD.com with 30 epochs (SP+R), the model optimized with an additional five epochs with only tokenization on genre prediction (SP+R $\rightarrow$ Genre), and the model optimized with an additional five epochs with only tokenization on rating prediction (SP+R $\rightarrow$ Rating).

The experimental results demonstrated that the performance of the downstream model with tokenization optimized to the same task was higher than that with tokenization for the different task in all datasets and in all languages. These results verified that the proposed method can identify tokenization specialized to the downstream model and task through its optimization process.

Interestingly, the results also showed that tokenization for the different task could contribute to improving the performance of the other downstream task. For example, tokenization for Rating(Zh) improved the performance of the downstream model for Genre(Zh) as compared to the original performance. This result suggests that the tokenization for rating prediction includes the useful tokenization for genre prediction. Specifically, the performance of the downstream model for the genre prediction was improved with tokenization for rating prediction tasks in all languages as compared to the original performance. This indicates that the tokenization specialized to the rating prediction task was partially effective in the genre prediction task.

By contrast, the results also demonstrated that tokenization for genre prediction did not have a good effect on the downstream model for the rating prediction task. This result suggests that the optimized vocabulary for genre preiction does not include the useful vocabulary for rating prediction because the tokenization for genre prediction was split into tiny granularities by the proposed method (Section 6.4.1).

Note that the experiment in which the same task was used for the downstream model and the tokenizer was identical to the experiment described in Section 6.1.2. Thus, these experimental results indicated that OpTok4AT also contributed to improved performance of postprocessing not only for sentiment classification (Section 6.1.2), but also for text classification on the review corpus in terms of rating and genre predictions.

Dataset	Task for Downstream Model	Task for Tokenization		
		SP+R	SP+R ↪ Genre	SP+R ↪ Rating
JD.com (Chinese)	Genre	48.85	**49.29**†‡	49.14
	Rating	53.39	53.37	**53.66**†
Rakuten (Japanese)	Genre	45.48	**46.12**†‡	45.64†
	Rating	48.94	49.07	**49.18**†‡
Amazon (English)	Genre	71.64	**71.78**	71.66
	Rating	67.56	67.56	**67.72**†‡

Table 6.13: Performances of the downstream models with tokenizers trained on different tasks. The downstream model was trained with SentencePiece and subword regularization, and the tokenizer was trained with the trained downstream model whose parameters were frozen. Bold highlights indicate the highest performances among the evaluation tasks. †indicates that the score was significantly higher than that of the baseline system (SP+R) with the McNemar's test ($p < 0.05$). ‡also indicates that the score of the model with the matched tokenizer significantly overcomes that of the model with the mismatched tokenizer with the McNemar's test ($p < 0.05$).

6.6 Multitask Learning

The previous sections 6.4 and 6.5 and the following section 6.7 demonstrate that the proposed methods acquire task-specific tokenization through training. For example, the proposed methods trained on the genre prediction task use more tokens that seem helpful in solving genre prediction tasks rather than rating prediction tasks. Based on these results, this study is interested that the proposed method was trained on multiple tasks such as genre and rating prediction tasks. This section reports the experimental results of the proposed method on multitask learning.

The experiment exploited the E-commerce datasets Genre(Zh, Ja, En) and Rating(Zh, Ja, En). For multitask learning, the downstream model for text

classification described in 2.27 was modified as follows:

$$f(t_{\text{Genre}}|s';\theta_{\text{TC(Genre)}}) = \text{softmax}(\text{MLP}(g(s', \theta_{\text{TC}}^{(\text{enc})}), \theta^{(\text{MLP})_{\text{TC(Genre)}}}))_{t_{\text{Genre}}}, \quad (6.1)$$

$$f(t_{\text{Rating}}|s';\theta_{\text{TC(Rating)}}) = \text{softmax}(\text{MLP}(g(s', \theta_{\text{TC}}^{(\text{enc})}), \theta^{(\text{MLP})_{\text{TC(Rating)}}}))_{t_{\text{Rating}}}, \quad (6.2)$$

where $\theta^{(\text{MLP})_{\text{TC(Genre)}}}$ and $\theta^{(\text{MLP})_{\text{TC(Rating)}}}$ are parameters of MLP for the genre and rating prediction tasks, respectively. In this experiment, the BiLSTM-based encoder was used as g. Note that the single BiLSTM-based encoder was used for both genre and rating prediction tasks for multitask learning. For each mini-batch B, the parameters were updated to minimize the sum of cross-entropy losses for each task:

$$\mathcal{L}_{\text{Genre}}(s, t_{\text{Genre}}) = -\log f(t_{\text{Genre}}|s';\theta_{\text{TC(Genre)}}), \quad (6.3)$$

$$\mathcal{L}_{\text{Rating}}(s, t_{\text{Rating}}) = -\log f(t_{\text{Rating}}|s';\theta_{\text{TC(Rating)}}), \quad (6.4)$$

$$\mathcal{L}_{\text{Multitask}}(B) = \sum_{(s,t_{\text{Genre}},t_{\text{Rating}})\in B} \mathcal{L}_{\text{Genre}}(s, t_{\text{Genre}}) + \mathcal{L}_{\text{Rating}}(s, t_{\text{Rating}}). \quad (6.5)$$

Figure 6.2 presents an overview of the architecture for multitask learning.

Table 6.14 summarizes the experimental results on multitask learning. The results showed that the proposed methods outperformed the baseline method (SP+R). The scores for each method were lower than those of the single task settings reported in Table 5.7. This implies that multitask learning of the genre and rating prediction tasks did not help to improve the performance of each task.

The main focus of this experiment was on the acquired tokenization by the proposed methods. Table 6.15 shows the tokenization examples optimized to the multiple tasks. This comparison used the same sentences as those in the tokenization examples for the single tasks listed in Tables 6.8, 6.9, and 6.7.

Compared to the Chinese examples listed in Table 6.8, the tokenization for the multiple tasks listed at the top of Table 6.15 had the same tendencies of tokenization optimized for the rating and genre predictions. The tokenization by OpTok included 完全 (extremely) and 不 (not) corresponding to the proper rating label and 防滑 (anti-slip) corresponding to the proper genre label as analyzed in Section 6.4.1. The tokenization by OpTok4AT also contained 不好 (bad) and 防滑 corresponding to the rating and genre labels, respectively.

The tokenizations in Japanese (middle of 6.15) showed the same tendencies as the Chinese examples. Both tokenizations of OpTok and OpTok4AT contained 香

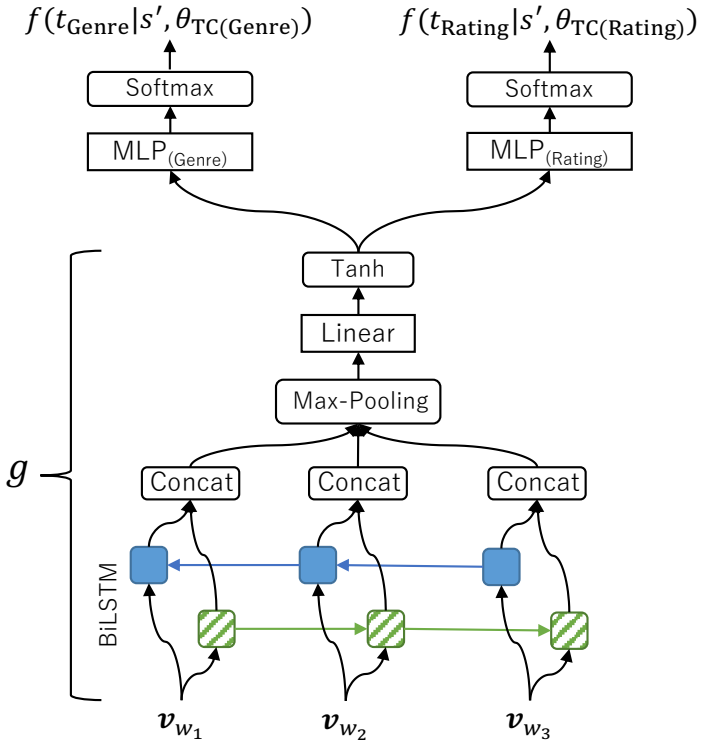


Figure 6.2: Overview of the downstream model for multitask learning using the BiLSTM-based encoder.

ゥ (perfume) and 髪 (hair) for the appropriate genre label and 全然 (at all) for the appropriate rating label. The English tokenizations showed similar tendencies. Both tokenizations of OpTok and OpTok4AT (bottom of 6.15) included *CD* and *favorite* corresponding to the appropriate genre and rating labels, respectively. In addition to the word corresponding to the proper label, both tokenizations had a word *travel* that corresponded to the improper genre label. As mentioned in Section 6.4.1, the proposed methods cannot change the tokenization depending on context because of the characteristic of the unigram language model. The examples show that this problem occurs with both multiple- and single-task settings.

The tokenization examples demonstrate that the proposed methods trained on multiple-task settings include the characteristics of tokenization optimized for both the rating and genre prediction tasks. The results suggest that the proposed method can be applied to multi- and single-task learning and can improve their

	SP	SP+R	OpTok	OpTok4AT
Genre(Zh)	44.95	46.32	46.64*	**47.60***
Rating(Zh)	46.60	46.77	47.19*	**48.15***
Genre(Ja)	43.95	47.11	48.00*	**49.87***
Rating(JA)	49.41	51.23	51.08	**52.21***
Genre(En)	67.23	70.81	70.61	**71.46***
Rating(En)	63.22	65.67	**65.79***	65.67

Table 6.14: Experimental results of multitask learning on E-commerce datasets. SP and +R denote SentencePiece and subword regularization, respectively. The highest scores are highlighted in bold. * indicates that the score was significantly higher than that of the baseline system (SP+R) with the McNemar's test ($p < 0.05$).

performances by acquiring the appropriate tokenizations.

6.7 Analysis with Simple Downstream Model

The previous experiments used the downstream model with a complicated architecture such as the BiLSTM-based encoder. This type of architecture achieves higher performance due to the rich representation of word embeddings. However, determining the token that is effective in solving the downstream task is somewhat difficult. By contrast, the proposed methods focus on the task-specific token for the appropriate tokenization.

To confirm the relationship between the acquired tokenization by the proposed methods and the effective tokens for the downstream task, this analysis exploited a simple classifier using multinomial logistic regression for multi-class classification. This simple method calculates the probability of label t for the tokenized sentence s' as follows:

$$\boldsymbol{h}_{s'} = \sum_{w \in s'} \boldsymbol{v}_w^{(\mathrm{OneHot})}, \tag{6.6}$$

$$p(t|s') = \mathrm{softmax}(W\boldsymbol{h}_{s'} + b)_t, \tag{6.7}$$

where $\boldsymbol{v}_w^{(\mathrm{OneHot})}$ is a one-hot vector whose element corresponding to the word w is 1 and is 0 for the others. The calculated sentence vector $\boldsymbol{h}_{s'}$ is known as a

Method	Tokenization
Chinese (Genre: 家居生活, Rating: 1)	
SentencePiece	东西 **非常不好** ! **完全 不防滑** !
OpTok	东西 **非常不好** ! **完全 不 防滑** !
OpTok4AT	东西 **非常 不好** ! **完全不 防滑** !
Japanese (Genre: 美容・コスメ・香水, Rating: 2)	
SentencePiece	**香りは** すき だけど 、 痛 んだ **髪に は全然 効果なし** 。
OpTok	**香り は** すき だけど 、 痛 んだ **髪 には 全然 効果 なし** 。
OpTok4AT	**香り は** すき だけど 、 痛 んだ **髪 には 全然 効果 なし** 。
English (Genre: CDs and Vinyl, Rating: 5)	
SentencePiece	I like to listen to **CDs** when **traveling** and this is a one of my **favorites** .
OpTok	I like to listen to **CD s** when **travel ing** and this is a one of my **favorite s** .
OpTok4AT	I like to listen to **CD s** when **travel ing** and this is a one of my **favorite s** .

Table 6.15: Differences in tokenization in Chinese (top), Japanese (middle), and English (bottom), from experiments with multitask settings.

bag-of-words vector. W and b are the trainable parameters and they are updated using the cross-entropy loss function as well as the other experiments with neural networks as described in Section 5.1. Inspecting the trained weight of W helps to analyze the importance of the token for each downstream task. For example, the weight $W_{a,b}$ in the a-th row and b-th column of W indicates how the a-th token in the vocabulary contributes to classifying the sentence into the b-th label.

Table 6.16 shows the performance of the downstream tasks by each method. The results demonstrate that the proposed method contributes to improved performance in most settings even with the simple text classifier.

Analyzing the trained weight of the simple downstream model helps to understand the behaviors of the proposed methods. This subsection presents an analysis of the differences among SentencePiece, OpTok, and OpTok4AT in terms of the top ten important words with the highest weight in each label. The top ten important words are listed in Tables 6.17, 6.18, and 6.19, corresponding to Chinese, Japanese, and English E-commerce datasets of rating prediction tasks, respectively. For OpTok and OpTok4AT, the tables also list the top ten words

	SP	SP+R	OpTok	OpTok4AT
Weibo(Zh)	93.18	92.86	**93.36**†‡	93.20‡
Twitter(Ja)	85.42	85.24	85.56	**85.66**‡
Twitter(En)	75.77	75.54	75.68	**75.91**
Genre(Zh)	47.77	47.62	47.43	**47.83**‡
Rating(Zh)	47.97	47.52	47.40	**48.10**
Genre(Ja)	50.02	50.03	50.87†‡	**51.00**†‡
Rating(Ja)	49.62	49.16	49.96‡	**50.07**‡
Genre(En)	73.01	73.33	72.85	**73.61**†
Rating(En)	61.44	60.67	**61.74**‡	61.62‡

Table 6.16: Experimental results of text classification with the simple encoder. SP and +R denote SentencePiece and subword regularization, respectively. The highest scores are highlighted in bold. †and ‡indicate that the score significantly overcomes that of the baseline systems, SP and SP+R, respectively, with the McNemar's test ($p < 0.05$).

whose unigram probabilities increased from the initial probabilities in the same manner as shown in Tables 6.4, 6.5, and 6.6. This analysis chose three of five ratings (Ratings 1, 3, and 5) for the table analyses.

The important word rankings of the initial tokenization (SentencePiece) listed in Tables 6.17a, 6.18a, and 6.19a demonstrate that words that include positive or negative sentiments are regarded as the most important features with the highest weights. This is in addition to the direct expression indicating the rating itself, such as "☆1", "☆3", and "☆5". in the Japanese examples.

In the Chinese examples shown in Table 6.17a, the downstream model trained with SentencePiece assigns greater weight to long and redundant words in each label. For example, some words in the rankings of Ratings 1, 3, and 5 contain the same substring "垃圾 (rubbish)", "一般 (ordinary)", and "非常 (extremely)", respectively. Assigning greater importance to these long and redundant words may cause overfitting of the training corpus and lead to poorer performance on the evaluation data because the samples in the evaluation data may not contain these long words. It is helpful to assign greater weights to short and general words to avoid overfitting in terms of generalization.

The rankings of OpTok and OpTok4AT (Tables 6.17b and 6.17c) demonstrate

that the proposed methods alleviate the problem of long and redundant words. The proposed methods split these long words by assigning higher probabilities to shorter tokens, as shown in the "*Rank*" column in the tables. For instance, the ranking of OpTok4AT (Table 6.17c) contains "垃圾", and the ranking for Rating 1 includes fewer redundant tokens, including "垃圾". This tendency relaxes the overfitting problem on the evaluation dataset and contributes to improved performance.

The Japanese examples (6.18) show similar tendencies as the Chinese examples. The rankings of OpTok and OpTok4AT are composed of shorter words compared to those of SentencePiece and the Chinese examples. OpTok and OpTok4AT also relax the problem of redundancy of words in the ranking such as "二度と (never)". in the ranking of Rating 1. For the results of OpTok4AT, although the number of words including "満足 (satisfied)". in the column of Rating 5 is the same as that of SentencePiece, the unigram language model of OpTok4AT increases the probability of the word "満足", as shown in the left column. This implies that the proposed method tries to split the substring from the long and redundant words, but it does not occur, possibly because of the number of training epochs.

The tendencies of English examples (Table 6.19) are different from those of the Asian languages. The contents of the rankings are similar to each other with the three tokenization methods. This result suggests that the variety of tokenization in English is less than that of Japanese and Chinese because English sentences are already split using whitespaces. Therefore, the tokenization hardly changes from the initial tokenization by the proposed methods. As shown in the example of the acquired tokenization in English by the proposed methods with the BiLSTM classifier (Table 6.7), the proposed methods mainly change the tokenization of the suffixes or prefixes of words, such as "s" for the plural form. Accordingly, the changes like ones in Chinese and Japanese could not be identified in the analysis of the weight of words in English.

Rating: 1	Rating: 3	Rating: 5
_假货	给个中评	五星
_太垃圾了	中评	_非常满意
_垃圾	感觉一般	_非常满意非常满意
_质量非常差	_感觉一般般	一如既往的好
_非常差	_一般吧	_非常好用
垃圾产品	_一般般	_很好很强大
_质量太差	但是物流	_非常好
垃圾中的垃圾	_效果一般	_非常满意非常满意非常满意
垃圾中的战斗机	_一般般吧	_很满意
_很差	不是特别好	_好好好好好好

(a) SentencePiece (Rating)

Rank	Rating: 1	Rating: 3	Rating: 5
差	假货	给个中评	五星
太	垃圾	一般	质量好
不错	很不	不是很	好东西
能	差	不是特别	一如既往的
!	优点	不是太	正品
可以	垃圾中的战斗机	马马虎虎	不错
也	骗人	一分钱一分货	物美价廉
不	假的	不太	第二次买了
喜欢	大骗子	还可以	货真价实
一般	坏	还不错	棒

(b) OpTok (Rating)

Rank	Rating: 1	Rating: 3	Rating: 5
!	给差评	中评	五星
太	差评	给个中评	非常好
不错	垃圾中的战斗机	一般吧	非常满意
喜欢	大骗子	逍	质量好
一般	千万别买	感觉一般	很满意
差	串色	般般	货真价实
垃圾	垃圾	韵达快递	很好
很	假货	一般般	一如既往的好
很好	上当	马马虎虎	很棒
咋	就坏了	不是很	质量很好

(c) OpTok4AT (Rating)

Table 6.17: Top ten important words in the Rating(Zh) dataset.

Rating: 1	Rating: 3	Rating: 5
_最悪	☆3	大満足です
☆1	星三つ	☆5
_二度と買いません	★3	これがないと
無駄な買い物でした	-2	とてもきれいな
_二度と買わない	三つ	大変満足しております
二度と買いません	普通に美味しかったです	大変満足です
_最悪です	可もなく不可もなく	大満足
度と買いません	良しとします	色違いも欲しくなりました
二度と利用しません	値段相応だと思います	で大満足です
最悪です	のがちょっと	大好評でした

(a) SentencePiece (Rating)

Rank	Rating: 1	Rating: 3	Rating: 5
ません	最悪です	☆3	大満足
まだ	最悪	★3	最高です
少し	☆1	星三つ	大満足です
もう少し	捨てました	-2	素晴らしい
ない	壊れました	可もなく不可もなく	これがないと
て	二度と	良しとします	☆5
とても	返品したい	まだ使ってませんが	満足です
は	_二度と	三つ	もっと早く
使用して	詐欺	まだ届いていませんが	リピートです
買って	使い物にならない	_届くのが楽しみです	リピです

(b) OpTok (Rating)

Rank	Rating: 1	Rating: 3	Rating: 5
とても	最悪	☆3	大満足です
お安く	☆1	★3	大満足
満足	最悪です	三つ	☆5
ません	二度と買いません	星三つ	で大満足です
安く	最悪でした	可もなく不可もなく	素晴らしい
ですが	度と買いません	まずまず	最高です
ませんでした	詐欺	まだ使ってませんが	大変満足しています
てしまいました	返品したい	良しとします	最高
してます	二度と	-2	手放せません
もう少し	捨てました	ふつうに	満足です

(c) OpTok4AT (Rating)

Table 6.18: Top ten important words in the Rating(Ja) dataset.

Rating: 1	Rating: 3	Rating: 5
_worst	_okay	_Excellent
_terrible	_OK	_Highly
_waste	_ok	Highly
_garbage	However	_Perfect
_useless	_However	_wonderful
_horrible	_alright	_masterpiece
_crap	BUT	_Great
_refund	_however	_perfect
_threw	_average	_amazing
_NOT	_Ok	_awesome

(a) SentencePiece (Rating)

Rank	Rating: 1	Rating: 3	Rating: 5
.	_worst	_okay	_stocked
_	_terrible	_ok	_Highly
_It	_garbage	_OK	Love
_love	_waste	However	_GLAD
This	_useless	_However	_awesome
I	_horrible	BUT	Excellent
_this	_refund	_alright	_masterpiece
’	_worthless	_average	Great
_it	_crap	_Unfortunately	Perfect
_of	_junk	_jarring	_cardinals

(b) OpTok (Rating)

Rank	Rating: 1	Rating: 3	Rating: 5
_	_worst	_okay	_Highly
,	_terrible	_ok	Excellent
_this	_waste	_OK	Highly
t	_horrible	okay	_perfectly
_I	_garbage	However	_awesome
_to	_useless	_alright	_amazing
_great	_refund	_However	_fantastic
_at	_crap	OK	Love
_it	_trash	_average	Perfect
_and	_threw	_Unfortunately	Great

(c) OpTok4AT (Rating)

Table 6.19: Top ten important words in the Rating(En) dataset.

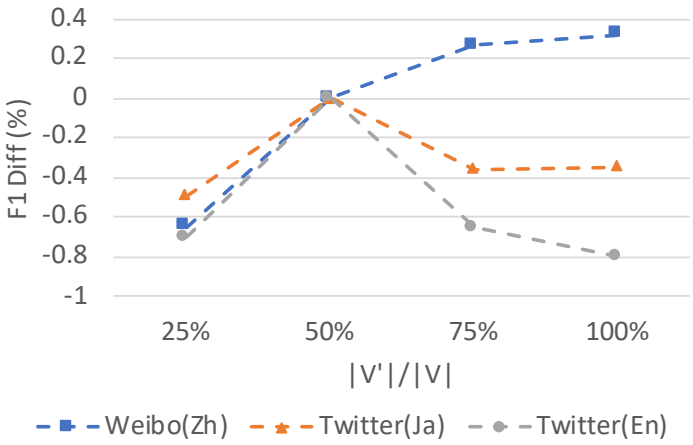


Figure 6.3: Differences in scores using 50% of the entire vocabulary reported in Table 5.7 against the different $|V'|$ on a sentiment analysis. The sizes of the vocabularies are 16,000 for Twitter(Ja) and Twitter(En), and 32,000 for Weibo(Zh). The size of N-best is $N = 3$.

6.8 Effects of Hyperparameters

This study introduces two hyperparameters to control OpTok and OpTok4AT: the size of the N-best tokenization used for the training of the neural unigram language model, and the weight of the loss to maintain the characteristics of the language model μ, as described in Section 3.5. Another hyperparameter is introduced for the training of OpTok, namely, the number of words in the restricted vocabulary $|V'|$. This section analyzes the effects of each hyperparameter on the performance of downstream tasks.

6.8.1 Number of Words in the Restricted Vocabulary

The training of OpTok exploits the restricted vocabularies to vary the candidates in N-best tokenization (Section 3.4). Using larger restricted vocabularies for the training enables the downstream model to exploit more word embeddings as the variation in N-best tokenization decreases. By contrast, a smaller vocabulary restricts the downstream model to use more limited word embeddings while using a large variety of tokenization. This subsection reports the effects of hyperparameters on the performance of sentiment analysis.

Figure 6.3 reports the effects derived from the sizes of the restricted vocabu-

laries in the different languages. The figure shows the performances achieved by the proposed method, where the sizes of the vocabularies were reduced to 25%, 50% (the default settings used in Table 5.7), 75%, and 100% from their initial sizes. The figure shows differences in the average F1 scores over five trials from scores reported in Table 5.7. As the figure shows, restricting the vocabulary size to 50% contributed to improved performances with the Japanese and English datasets. These results verify that the vocabulary restrictions work well for the proposed method. In addition, decreasing the vocabulary size negatively affected the performance proportionately to the Chinese dataset. In fact, the average best performance achieved by the full size of the vocabulary (i.e., 100% for 32,000) was 93.14, which was higher by 0.21 over the score of OpTok (Table 5.7). This result suggests that decreasing the size of the vocabulary is unnecessary for languages holding vast types of characters because this type of restriction causes a leaking of useful tokens and the production of many unknown tokens in both the training and evaluation, as reported by Hiraoka et al. [39].

6.8.2 Number of N-best Tokenization

The proposed methods update the neural unigram language model using N-best tokenized candidates. More specifically, OpTok inputs the weighted sum of sentence representations of N-best tokenizations into the downstream model, as indicated in (3.7), whereas OpTok4AT minimizes the weighted sum of loss values calculated from the downstream model whose inputs are N-best tokenizations, as indicated in (4.2). For both proposed methods, a larger N enables the model to update the neural unigram language model with more candidates of tokenization. In other words, OpTok and OpTok4AT can explore the appropriate tokenization from a larger search space with a large N. Ideally, the proposed methods can find more appropriate tokenization with the larger N, and the performance improves together with N. This section provides an analysis of the effects of the hyperparameter N on the performances of the downstream tasks.

Experiments were conducted on text classification and machine translation with different N, as described in Section 5. Figures 6.4a and 6.4b show the results of text classification and machine translation, respectively. These figures illustrate the differences from the model performance with the settings described in Section 5 (i.e., $N = 3$ for text classification and $N = 8$ for machine translation).

Text Classification

For the text classification task, this section analysed the effects of N on the sentiment analysis datasets for Chinese, Japanese, and English. Figure 6.4a shows that $N = 3$ achieved the best performance for OpTok in all languages, whereas an increase in N decreased the performance. The decline may have been due to the differences in the encoding strategies between the training and evaluation, as described in Section 4.4. With a larger N, a task-specific module such as MLP for text classification could be trained using the weighted sum of the various tokenizations, whereas the module requires a sentence representation encoded with the best tokenization in the inference.

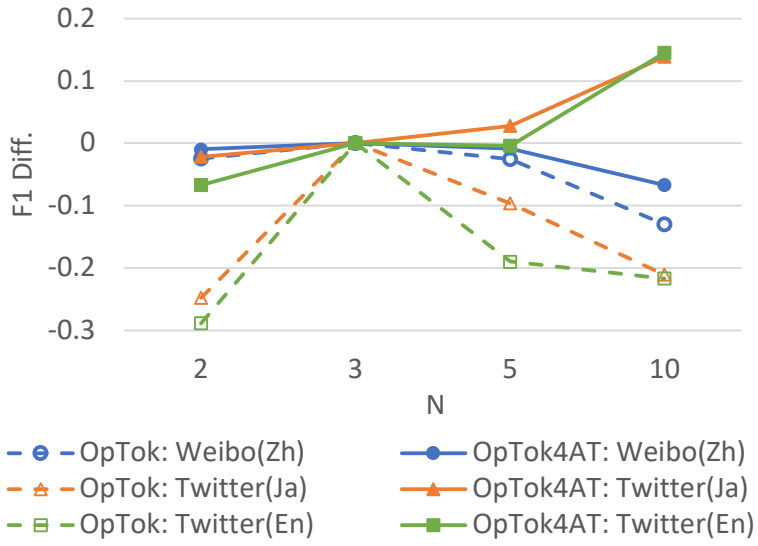
Figure 6.4a shows that the number of N did not have a strong effect on the performance of OpTok4AT. In addition, unlike the result with OpTok, the larger N led to slightly better performance for the Japanese and English datasets. By contrast, the performance for the Chinese dataset decreased with a large N. This occurred possibly because a Chinese sentence has more tokenization candidates than in the other languages, and the optimization of tokenization becomes unstable with a larger N.

These results demonstrate that OpTok4AT is more robust to large N as compared to OpTok with text classification tasks. As described in Sections 4.4 and 5.1, OpTok4AT avoids the gap between training and inference in terms of the weighting strategy. Because OpTok4AT uses only a single sampled tokenization to train the downstream model, the number of N does not damage the text classification performance. In other words, OpTok4AT uses the N tokenizations only to update the neural unigram language model isolated from the downstream model.

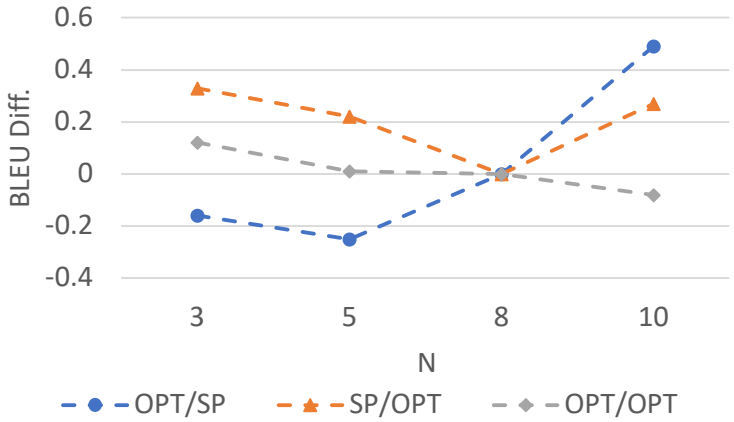
Machine Translation

For the machine translation task, experiments were conducted to confirm the effect of N on the performance of OpTok4AT using the Vi-En pair of IWSLT15. Figure 6.4b illustrates that the number of N did not have a strong effect on the performance when using OpTok4AT solely for the target side (SP/OPT). When incorporating the proposed method to the source side (OPT/SP), the performance increased with a large N. The result suggests that the proposed method could find an appropriate tokenization from the large search space when a large

N was used because the neural encoder of NMT allows various tokenizations for its input. When using the proposed method for both the encoder and decoder (OPT/OPT), the performance decreased slightly with a larger N. This result suggests that optimization of the tokenization of both sides with a large N became unstable because tokenization on the source side varied considerably during training, as mentioned Section 6.2 and 6.4.2.



(a) Text classification



(b) Machine Translation

Figure 6.4: Differences in performance against N on text classification (6.4a) and machine translation (Vi-En, 6.4b).

6.8.3 Hyperparameter that Maintains the Characteristics of Language Model μ

This study introduced the additional loss value ($\mathcal{L}_s^{\mathrm{lm}}$) to maintain the characteristics of the neural unigram language model, as explained in Section 3.5, and the effect of this loss value could be controlled using the weight μ as a hyperparameter. The larger μ has a significant effect on the training of the proposed methods in maintaining the characteristics of the language model, as described in (3.9). The word probabilities of the neural unigram language model trained with the large μ reflects the word frequency in the training corpus. In the experiments described in Chapter 5, this hyperparameter was set to $\mu = 0.01$. This subsection discusses the effect of various μs on the performance of the downstream models and nature of language models using Chinese, Japanese, and English datasets of sentiment analysis.

Figure 6.5a shows the perplexity [49, 8] calculated with the trained neural unigram language model of OpTok and OpTok4AT with different μs. The sentences in the training data were tokenized with the trained neural unigram language model into 1-best tokenizations and the perplexity PPL was calculated as follows:

$$H = \frac{1}{\sum_{s \in D} |s'|} \sum_{s \in D} \sum_{w \in s'} -\log_2 p(w), \tag{6.8}$$

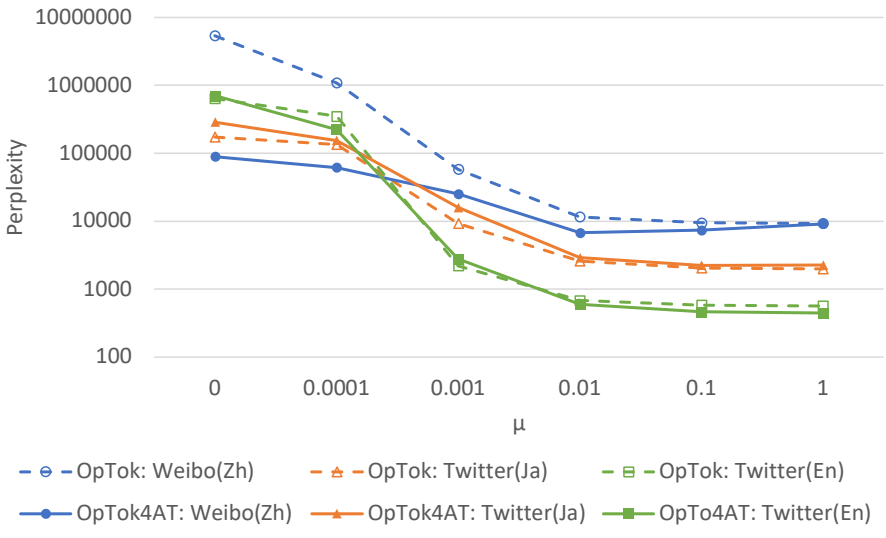
$$\mathrm{PPL} = 2^H, \tag{6.9}$$

where D is the training data, s' is the 1-best tokenization of s with the neural unigram language model, and $p(w)$ is derived from the trained neural unigram language model.

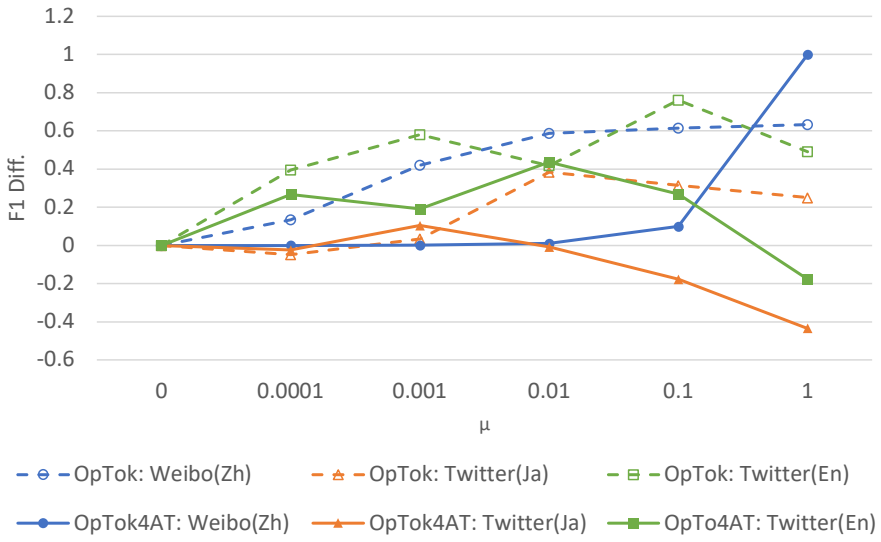
Figure 6.5a demonstrates that the perplexity tended to decrease with a small μ in all settings. This result indicates that the neural unigram language model of OpTok and OpTok4AT trained with the large μ properly reflects the frequency of words in the training data. On the Weibo(Zh) dataset, the perplexities of OpTok with the smaller μ were higher than those of OpTok4AT. This result suggests that the training of OpTok hardly changes the parameters of the neural unigram language model, and the tokenization becomes considerably different from the initial one. This difference derived from the differences in architectures in handling N-best tokenization. Unlike OpTok4AT, OpTok inputs the weighted

sum of the sentence representations of N-best tokenization, and the gradient for the neural unigram language model is calculated through the downstream model. Therefore, the gradient for the neural unigram language model of OpTok includes considerably more information than does OpTok4AT, and the parameters of the neural unigram language model hardly change from the initial parameters, particularly with the Chinese dataset (which, in our experiments, included a greater variety of tokenization).

Figure 6.5b shows the downstream task performances of downstream models trained with different μs. The values shown indicate the differences in performances of the model trained with $\mu = 0$. The results of OpTok showed that the performances tended to increase in a stable manner with the larger μ. By contrast, this stable tendency was not observed in the performance of OpTok4AT with μ, and the peak of the performance by OpTok4AT differed depending on languages. These differences in the proposed methods were due to the different architectures for handling the N-best tokenizations, and OpTok4AT was more sensitive against μ in terms of performance. This indicates that the hyperparameter μ should be carefully selected for OpTok4AT.



(a) Perplexity against μ



(b) Performance against μ

Figure 6.5: Differences in perplexity on the tokenization of the training (6.5a) split and the performance of the downstream tasks against the weight for maintaining the characteristics of the language model μ (6.5b).

6.9 Application for BERT

Numerous studies have recently focused on exploiting pretrained language models to enhance NLP tasks. These include ELMo [88], XLNet [124], GPT-3 [10], and BERT [22]. This section describes how OpTok and OpTok4AT are applicable to recent NLP modules based on BERT through an experiment on the English datasets of Twitter(En), Genre(En), and Rating(En).

In this experiment, the BiLSTM-based encoder in the downstream model was replaced with BERT, and this analysis conducted the same experiments as those described in Section 5.1. This experiment employed $\text{BERT}_{\text{base}}$ from HuggingFace[4] and fine-tuned its parameters, except for those of the word embeddings as well as the aforementioned experiments. Because the published tokenizer for $\text{BERT}_{\text{base}}$ is based on WordPiece, which does not include the probabilities for each piece, the probabilities were estimated on the training split using the EM algorithm [21, 66, 60] and initialized the language model of the proposed methods with these probabilities. The restricted vocabulary was not used in this experiment because the vocabulary of BERT contains many tokens not applicable to this experiment. Compared to the vocabulary initialized using SentencePiece on only the training split, restricting the vocabulary results in too little diversity of the N-best tokenization to cause overfitting of tokenization. Therefore, restricting the vocabulary was not necessary, as with the Chinese dataset described in Section 6.8.1. The trainable parameters of $\text{BERT}_{\text{base}}$ were fine-tuned using AdamW [67], and the neural unigram language model was updated in OpTok and OpTok4AT with Adam.

Table 6.20 shows the results of this experiment. For the experiment using the original BERT, sentences in the training split were tokenized using the longest-match-first algorithm of WordPiece implemented by HuggingFace. The model of *BERT+R* was trained with a stochastic tokenization such as SentencePiece based on the language model initialized using the EM algorithm.

The results show that the pretrained BERT improved the performance, as compared with the scores presented in Table 5.7. In addition, the model trained with subword regularization (BERT+R) surpassed the performance of the original BERT. These results revealed that subword regularization was effective in the experiment using the large pretrained language model. The model incor-

[4] `https://github.com/huggingface/transformers`

	Best in Table 5.7	BERT	BERT+R	OpTok	OpTok4AT
Twitter(En)	79.04	80.98	81.22	81.98*	81.67
Genre(En)	71.83	76.22	77.21	76.71	77.61*
Rating(En)	67.90	70.36	70.88	70.84	71.05*

Table 6.20: F1 scores on Twitter(En), Genre(En), and Rating(En) with $\text{BERT}_{\text{base}}$. The highest scores are highlighted in bold. * indicates that the score was significantly higher than that of the baseline system (BERT+R) with the McNemar's test ($p < 0.05$).

porating OpTok achieved the highest score on the Twitter(En) dataset. The performances of OpTok in Genre(En) and Rating(En) were only comparable to that of BERT+R. By contrast, the model with OpTok4AT scored higher than BERT and BERT+R with all datasets.

OpTok scored higher than OpTok4AT for Twitter(En) in the previous experiments when using a complicated neural architecture (Tables 5.6 and 5.7). The results are shown in Tables 5.6 and 5.7, where the same tendency was observed even in the experiments with BERT, which has a much more complicated neural architecture. These results indicate that OpTok4AT contributed to an improvement in the popular NLP architecture using BERT in terms of optimizing the tokenization. By contrast, OpTok could only be expected to improve performance with informal text such as in the Twitter corpora.

7 Conclusion

This study introduced new methods to address the NLP problem of identifying appropriate tokenization that otherwise cannot be found in preprocessing isolated from the downstream task. Two approaches based on the same idea were proposed. The proposed methods jointly train the tokenizer and downstream model to obtain the appropriate tokenization for the downstream model. The proposed methods exploit some tokenized candidates for the input sentence to update the tokenizer composed of a neural unigram language model.

The first approach, OpTok, updates the tokenizer by weighting the sentence vectors of each tokenization candidate with their probabilities as calculated by the language model. OpTok inputs the weighted sum of the sentence vectors into the downstream model like the general architecture using the neural networks. Thus, the parameters of the tokenizer are updated by applying back-propagation for the final loss value calculated with the weighted sum against the task-specific supervisory signal.

The second approach, OpTok4AT, calculates the loss values corresponding to each tokenization candidate by inputting it into the downstream model separately. OpTok4AT then updates the parameters of the tokenizer to minimize the final loss value calculated by weighting the loss values with the corresponding probabilities of tokenization candidates.

The advantage of OpTok is that the parameters of both the tokenizer and downstream model can be updated using the single loss value calculated by the weighted sum of the sentence vector. However, this is limited to architectures using sentence vectors during the calculation, such as text classification tasks. OpTok4AT relaxes this problem by extending OpTok and can be applied to various architectures if the loss value when using it can be calculated. Because OpTok4AT exploits the loss values corresponding to each tokenization candidates, the downstream runs multiple times depending on the number of candidates, and this results in time and memory inefficiencies. To avoid this problem, this

study proposed a training strategy using subword regularization and updated the tokenizer and downstream model with corresponding unique loss values.

Experimental results showed that both proposed methods contributed to performance improvements in text classification tasks with Chinese, Japanese, and English. The proposed methods are applicable to various neural architectures, such as models that use attention mechanisms, BiLSTM, multinomial logistic regression, and large pretrained language models such as BERT. OpTok4AT can also be applied to machine translation tasks with Transformer and contributes to performance improvements as compared with exiting works that attempt to optimize tokenization.

Analysis on the acquired tokenization by the proposed methods illustrated that tokenizations were different depending on the downstream tasks and languages. Tokenizations for text classification and encoder side of machine translation included many tokens by splitting sentences into tiny units, whereas tokenization for the decoder side of machine translation contained long tokens and includes the original tokenization by SentencePiece. Quantitative analysis on tokenization revealed that the proposed methods extracted stem words by cutting off suffixes to use short and general tokens for the training.

The technical problem with the proposed methods is their processing speeds when used in real-world applications. The proposed methods can yield an appropriate 1-best tokenization efficiently using the Viterbi algorithm once the tokenizer is trained. However, training the tokenizer with the proposed strategy is time-consuming because the parameters of the tokenizer are updated using N tokenized candidates. As demonstrated in Section 6.8.2, N should be at least 3 to achieve stable performance. The training speed depends on the number of N and considerable time is required to train the proposed method when using a larger N. For example, the average numbers of sentence per second processed during the training of the Twitter(En) dataset were 181 and 225 for OpTok and OpTok4AT, respectively. By contrast, the numbers were 1,181 for SentencePiece and 995 for SentencePiece with subword regularization. The proposed method requires a more effective training strategy to shorten the training time for an application.

Tokenization or word segmentation is a fundamental problem that affects the performance of NLP. Although many researchers have reported that an appropriate tokenization improves the performance of NLP, identifying the proper to-

kenization was not easy in the conventional NLP This study showed that we can achieve improved performance by refining the tokenization and that appropriate tokenization could be determined based on the downstream task and model. This study is expected to stimulate further research on task-oriented word segmentation.

References

[1] Pranav A and Isabelle Augenstein. 2kenize: Tying subword sequences for chinese script conversion. *arXiv preprint arXiv:2005.03375*, 2020.

[2] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] Danushka Bollegala, Ryuichi Kiryo, Kosuke Tsujino, and Haruki Yukawa. Language-independent tokenisation rivals language-specific tokenisation for word similarity prediction. *arXiv preprint arXiv:2002.11004*, 2020.

[6] Kaj Bostrom and Greg Durrett. Byte pair encoding is suboptimal for language model pretraining. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 4617–4624, 2020.

[7] Samuel Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, 2015.

[8] Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, Jennifer C Lai, and Robert L Mercer. An estimate of an upper bound for the entropy of english. *Computational Linguistics*, 18(1):31–40, 1992.

[9] Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[10] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[11] Deng Cai, Hai Zhao, Zhisong Zhang, Yuan Xin, Yongjian Wu, and Feiyue Huang. Fast and accurate neural word segmentation for chinese. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 608–615, 2017.

[12] Lewis Carroll. *Alice's Adventures in Wonderland*. 1865.

[13] William Chan, Yu Zhang, Quoc Le, and Navdeep Jaitly. Latent sequence decompositions. *arXiv preprint arXiv:1610.03035*, 2016.

[14] Pi-Chuan Chang, Michel Galley, and Christopher D Manning. Optimizing chinese word segmentation for machine translation performance. In *Proceedings of the third workshop on statistical machine translation*, pages 224–232, 2008.

[15] Xinchi Chen, Xipeng Qiu, Chenxi Zhu, and Xuan-Jing Huang. Gated recursive neural network for chinese word segmentation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1744–1753, 2015.

[16] Xinchi Chen, Zhan Shi, Xipeng Qiu, and Xuanjing Huang. Dag-based long short-term memory for neural word segmentation. *arXiv preprint arXiv:1707.00248*, 2017.

[17] Tagyoung Chung and Daniel Gildea. Unsupervised tokenization for machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 718–726. Association for Computational Linguistics, 2009.

[18] Mathias Creutz and Krista Lagus. Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*. Association for Computational Linguistics, July 2002.

[19] Hongyi Cui, Yizhen Wei, Shohei Iida, Takehito Utsuro, and Masaaki Nagata. University of tsukuba's machine translation system for iwslt20 open domain translation task. In *Proceedings of the 17th International Conference on Spoken Language Translation*, pages 145–148, 2020.

[20] Hiroyuki Deguchi, Masao Utiyama, Akihiro Tamura, Takashi Ninomiya, and Eiichiro Sumita. Bilingual subword segmentation for neural machine translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4287–4297, 2020.

[21] Sabine Deligne and Frederic Bimbot. Language modeling by variable length sequences: Theoretical formulation and evaluation of multigrams. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 169–172. IEEE, 1995.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[23] Miguel Domingo, Mercedes Garcıa-Martınez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. How much does tokenization affect neural machine translation? *arXiv preprint arXiv:1812.08621*, 2018.

[24] Nadir Durrani and Sarmad Hussain. Urdu word segmentation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 528–536, 2010.

[25] Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[26] Philip Gage. A new algorithm for data compression. *C Users J.*, 12(2): 23–38, feb 1994. ISSN 0898-9788.

[27] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252, 2017.

[28] Sharon Goldwater and David McClosky. Improving statistical mt through morphological analysis. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 676–683, 2005.

[29] Sharon Goldwater, Thomas L Griffiths, and Mark Johnson. Contextual dependencies in unsupervised word segmentation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 673–680. Association for Computational Linguistics, 2006.

[30] Sharon Goldwater, Thomas L Griffiths, and Mark Johnson. A bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112(1):21–54, 2009.

[31] Chen Gong, Zhenghua Li, Min Zhang, and Xinzhou Jiang. Multi-grained chinese word segmentation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 692–703, 2017.

[32] Google. The wordpiece algorithm in open source bert, 2018. URL `https://github.com/google-research/bert/blob/eedf5716ce1268e56f0a50264a88cafad334ac61/tokenization.py#L300-L399`.

[33] Thamme Gowda and Jonathan May. Finding the optimal vocabulary size for neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, Online, November 2020. Association for Computational Linguistics.

[34] Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016.

[35] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[36] Hangfeng He and Xu Sun. F-score driven max margin neural network for named entity recognition in chinese social media. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 713–718, 2017.

[37] Xuanli He, Gholamreza Haffari, and Mohammad Norouzi. Dynamic programming encoding for subword segmentation in neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online, July 2020. Association for Computational Linguistics.

[38] Shohei Higashiyama, Masao Utiyama, Taro Watanabe, and Eiichiro Sumita. A text editing approach to joint Japanese word segmentation, POS tagging, and lexical normalization. In *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*, Online, November 2021. Association for Computational Linguistics.

[39] Tatsuya Hiraoka, Hiroyuki Shindo, and Yuji Matsumoto. Stochastic tokenization with a language model for neural text classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1620–1629, 2019.

[40] Anh Khoa Ngo Ho and François Yvon. Optimizing word alignments with better subword tokenization. In *Proceedings of the 18th Biennial Machine Translation Summit (Volume 1: Research Track)*, pages 256–269, 2021.

[41] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

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

[43] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, 2018.

[44] Jingjing Huo, Christian Herold, Yingbo Gao, Leonard Dahlmann, Shahram Khadivi, and Hermann Ney. Diving deep into context-aware neural machine translation. *arXiv preprint arXiv:2010.09482*, 2020.

[45] Rakuten Inc. Rakuten dataset. Informatics Research Data Repository, National Institute of informatics. (dataset)., 2014.

[46] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1681–1691, 2015.

[47] Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding convolutional neural networks for text classification. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 56–65, 2018.

[48] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[49] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 1977.

[50] Mikael Kågebäck and Hans Salomonsson. Word sense disambiguation using a bidirectional lstm. In *Proceedings of the 5th Workshop on Cognitive Aspects of the Lexicon (CogALex-V)*, pages 51–56, 2016.

[51] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

[52] Kazuya Kawakami, Chris Dyer, and Phil Blunsom. Learning to create and reuse words in open-vocabulary neural language modeling. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1492–1502, 2017.

[53] Kazuya Kawakami, Chris Dyer, and Phil Blunsom. Learning to discover, ground and use words with segmental neural language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6429–6441, 2019.

[54] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[55] Philipp Koehn. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 388–395, 2004.

[56] Philipp Koehn. *Statistical machine translation*. Cambridge University Press, 2009.

[57] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[58] Taku Kudo. Mecab: Yet another part-of-speech and morphological analyzer, 2006. URL `http://taku910.github.io/mecab/`.

[59] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, 2018.

[60] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, 2018.

[61] Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. Applying conditional random fields to Japanese morphological analysis. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[62] John D Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[63] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[64] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 5:365–378, 2017.

[65] Xiaonan Li, Hang Yan, Xipeng Qiu, and Xuanjing Huang. Flat: Chinese ner using flat-lattice transformer. *arXiv preprint arXiv:2004.11795*, 2020.

[66] Percy Liang and Dan Klein. Online em for unsupervised models. In *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*, pages 611–619, 2009.

[67] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[68] Jin Kiat Low, Hwee Tou Ng, and Wenyuan Guo. A maximum entropy approach to chinese word segmentation. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, 2005.

[69] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

[70] Zin Maung Maung and Yoshiki Mikami. A rule-based syllable segmentation of myanmar text. In *proceedings of the IJCNLP-08 workshop on NLP for less privileged languages*, 2008.

[71] Coskun Mermer, Murat Saraclar, and Ruhi Sarikaya. Improving statistical machine translation using bayesian word alignment and gibbs sampling.

[72] Daichi Mochihashi, Takeshi Yamada, and Naonori Ueda. Bayesian unsupervised word segmentation with nested pitman-yor language modeling. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 100–108. Association for Computational Linguistics, 2009.

[73] Hajime Morita, Daisuke Kawahara, and Sadao Kurohashi. Morphological analysis for unsegmented languages using recurrent neural network language model. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[74] Masaaki Nagata. A stochastic japanese morphological analyzer using a forward-dp backward-a* n-best search algorithm. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 201–207. Association for Computational Linguistics, 1994.

[75] Tetsuji Nakagawa. Chinese and japanese word segmentation using word-level and character-level information. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 466–472, 2004.

[76] Tetsuji Nakagawa and Kiyotaka Uchimoto. A hybrid approach to word segmentation and pos tagging. In *Proceedings of the 45th annual meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 217–220, 2007.

[77] Dat Quoc Nguyen. A neural joint model for vietnamese word segmentation, pos tagging and dependency parsing. In *Proceedings of the The 17th Annual Workshop of the Australasian Language Technology Association*, pages 28–34, 2019.

[78] ThuyLinh Nguyen, Stephan Vogel, and Noah A Smith. Nonparametric word segmentation for machine translation. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 815–823. Association for Computational Linguistics, 2010.

[79] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, 2019.

[80] Sonja Nießen and Hermann Ney. Statistical machine translation with scarce resources using morpho-syntactic information. *Computational linguistics*, 30(2):181–204, 2004.

[81] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[82] David D Palmer. A trainable rule-based algorithm for word segmentation. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 321–328, 1997.

[83] David D Palmer. Tokenisation and sentence segmentation. *Handbook of natural language processing*, pages 11–35, 2000.

[84] Constantine Papageorgiou. Japanese word segmentation by hidden markov model. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.

[85] Nanyun Peng and Mark Dredze. Named entity recognition for chinese social media with jointly trained embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 548–554, 2015.

[86] Nanyun Peng and Mark Dredze. Improving named entity recognition for chinese social media with word segmentation representation learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 149–155, 2016.

[87] Verónica Pérez-Rosas, Bennett Kleinberg, Alexandra Lefevre, and Rada Mihalcea. Automatic detection of fake news. In *Proceedings of the 27th*

International Conference on Computational Linguistics, pages 3391–3401, 2018.

[88] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, 2018.

[89] Matt Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation (WMT)*, pages 186–191, 2018.

[90] Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. BPE-dropout: Simple and effective subword regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online, July 2020. Association for Computational Linguistics.

[91] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training.

[92] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, 2019.

[93] Elizabeth Salesky, Andrew Runge, Alex Coda, Jan Niehues, and Graham Neubig. Optimizing segmentation granularity for neural machine translation. *Machine Translation*, pages 1–19, 2020.

[94] Steven L Scott. Bayesian methods for hidden markov models: Recursive computing in the 21st century. *Journal of the American Statistical Association*, 97(457):337–351, 2002.

[95] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages P1715–1725, 2016.

[96] Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD explorations newsletter*, 19(1):22–36, 2017.

[97] Kai Shu, Suhang Wang, and Huan Liu. Beyond news contents: The role of social context for fake news detection. In *Proceedings of the twelfth ACM international conference on web search and data mining*, pages 312–320, 2019.

[98] Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou. Fast wordpiece tokenization. *arXiv preprint arXiv:2012.15524*, 2020.

[99] Richard Sproat, Chilin Shih, William A Gale, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for chinese. In *32nd Annual Meeting of the Association for Computational Linguistics*, pages 66–73, 1994.

[100] Tejas Srinivasan, Ramon Sanabria, and Florian Metze. Multitask learning for different subword segmentations in neural machine translation. *arXiv preprint arXiv:1910.12368*, 2019.

[101] Ruslan Leont'evich Stratonovich. Conditional markov processes. In *Nonlinear transformations of stochastic processes*, pages 427–453. Elsevier, 1965.

[102] Maosong Sun, Dayang Shen, and Benjamin K Tsou. Chinese word segmentation without using lexicon and hand-crafted training data. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 2*, pages 1265–1271, 1998.

[103] Zhiqing Sun and Zhi-Hong Deng. Unsupervised neural word segmentation for chinese via segmental language modeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4915–4920, 2018.

[104] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[105] Yu Suzuki. Filtering method for twitter streaming data using human-in-the-loop machine learning. *Journal of Information Processing*, 27:404–410, 2019.

[106] Kazuma Takaoka, Sorami Hisamoto, Noriko Kawahara, Miho Sakamoto, Yoshitaka Uchida, and Yuji Matsumoto. Sudachi: a japanese tokenizer for business. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, may 2018.

[107] W. J. Teahan, Yingying Wen, Rodger McNab, and Ian H. Witten. A compression based algorithm for Chinese word segmentation. *Computational Linguistics*, 26(3), 2000.

[108] Yuanhe Tian, Yan Song, Xiang Ao, Fei Xia, Xiaojun Quan, Tong Zhang, and Yonggang Wang. Joint chinese word segmentation and part-of-speech tagging via two-way attentions of auto-analyzed knowledge. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8286–8296, 2020.

[109] Arseny Tolmachev, Daisuke Kawahara, and Sadao Kurohashi. Juman++: A morphological analysis toolkit for scriptio continua. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[110] Ke M Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. Unsupervised neural hidden markov models. In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 63–71, 2016.

[111] Kei Uchiumi, Hiroshi Tsukahara, and Daichi Mochihashi. Inducing word and part-of-speech with pitman-yor hidden semi-markov models. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1774–1782, 2015.

[112] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you

need. *Advances in neural information processing systems*, 30:5998–6008, 2017.

[113] Sami Virpioja, Peter Smit, Stig-Arne Grönroos, and Mikko Kurimo. Morfessor 2.0: Python implementation and extensions for morfessor baseline. 2013.

[114] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[115] Lihao Wang, Zongyi Li, and Xiaoqing Zheng. Unsupervised word segmentation with bi-directional neural language model. *arXiv preprint arXiv:2103.01421*, 2021.

[116] Shuangzhi Wu, Xing Wang, Longyue Wang, Fangxu Liu, Jun Xie, Zhaopeng Tu, Shuming Shi, and Mu Li. Tencent neural machine translation systems for the wmt20 news translation task. In *Proceedings of the Fifth Conference on Machine Translation*, pages 313–319, 2020.

[117] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[118] Fengshun Xiao, Jiangtong Li, Hai Zhao, Rui Wang, and Kehai Chen. Lattice-based transformer encoder for neural machine translation. *arXiv preprint arXiv:1906.01282*, 2019.

[119] Xinyan Xiao, Yang Liu, Young-Sook Hwang, Qun Liu, and Shouxun Lin. Joint tokenization and translation. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1200–1208, 2010.

[120] Jia Xu, Jianfeng Gao, Kristina Toutanova, and Hermann Ney. Bayesian semi-supervised chinese word segmentation for statistical machine translation. In *Proceedings of the 22nd International Conference on Computational*

Linguistics-Volume 1, pages 1017–1024. Association for Computational Linguistics, 2008.

[121] Nianwen Xue. Chinese word segmentation as character tagging. In *International Journal of Computational Linguistics & Chinese Language Processing, Volume 8, Number 1, February 2003: Special Issue on Word Formation and Chinese Language Processing*, pages 29–48, 2003.

[122] Jie Yang, Yue Zhang, and Fei Dong. Neural word segmentation with rich pretraining. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 839–849, 2017.

[123] Jie Yang, Yue Zhang, and Shuailong Liang. Subword encoding in lattice lstm for chinese word segmentation. *arXiv preprint arXiv:1810.12594*, 2018.

[124] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[125] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

[126] Yongfeng Zhang, Min Zhang, Yi Zhang, Guokun Lai, Yiqun Liu, Honghui Zhang, and Shaoping Ma. Daily-aware personalized recommendation based on feature-level time series analysis. In *Proceedings of the 24th international conference on world wide web*, pages 1373–1383, 2015.

[127] Yue Zhang and Jie Yang. Chinese ner using lattice lstm. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1554–1564, 2018.

[128] Hai Zhao, Changning Huang, and Mu Li. An improved chinese word segmentation system with conditional random field. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, pages 162–165, 2006.

[129] Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. Deep learning for chinese word segmentation and pos tagging. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 647–657, 2013.

[130] Valentin Zhikov, Hiroya Takamura, and Manabu Okumura. An efficient algorithm for unsupervised word segmentation with branching entropy and mdl. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 832–842, 2010.

[131] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3485–3495, 2016.

Publication List

Refereed Journals

1. 平岡 達也, 高瀬 翔, 内海 慶, 欅 惇志, 岡崎 直観. 単語分割と後段モデルの損失値を用いた同時最適化. 自然言語処理, 29(1): 33 pages, 2022年3月. （採択済み）

2. Tatsuya Hiraoka, Sho Takase, Kei Uchiumi, Atsushi Keyaki, and Naoaki Okazaki. Recurrent Neural Hidden Markov Model for High-Order Transition. ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP). Volume 21, Issue 2, Article No.: 36, pp: 1-15. March 2022.

3. 平岡 達也, 高瀬 翔, 内海 慶, 欅 惇志, 岡崎 直観. テキストベクトルの重みづけを用いたタスクに対する単語分割の最適化. 自然言語処理, 28(2):479–507, 2021年6月.

Refereed International Conference Papers

4. Tatsuya Hiraoka, Sho Takase, Kei Uchiumi, Atsushi Keyaki, and Naoaki Okazaki. Joint Optimization of Tokenization and Downstream Model. In Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 244-255, Online, August 2021.

5. Tatsuya Hiraoka, Sho Takase, Kei Uchiumi, Atsushi Keyaki, Naoaki Okazaki. Optimizing Word Segmentation for Downstream Task. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, pages 1341-1351, Association for Computational Linguistics, November 2020.

6. Tatsuya Hiraoka, Hiroyuki Shindo, Yuji Matsumoto. Stochastic Tokenization with a Language Model for Neural Text Classification. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 1620-1629, July 2019.

Unrefereed Papers

7. 平岡 達也, 高瀬 翔, 内海 慶, 欅 惇志, 岡崎 直観. 単語の長さと構成要素を考慮した単語レベルの摂動. 言語処理学会第28回年次大会 (NLP2022), 6 pages, 2022年3月.

8. 平岡 達也, 高瀬 翔, 内海 慶, 欅 惇志, 岡崎 直観. 後段モデルの損失値を用いた単語分割のタスクへの最適化. 言語処理学会第27回年次大会 (NLP2021), pp. 486–491, 2021年3月. （若手奨励賞）

9. 平岡 達也, 高瀬 翔, 内海 慶, 欅 惇志, 岡崎 直観. RNNにより高次の依存を考慮したニューラル隠れマルコフモデル. 言語処理学会第26回年次大会 (NLP2020), pp. A4–2 (4 pages), 茨城大学（茨城県）, 2020年3月.

10. 平岡 達也, 高瀬 翔, 内海 慶, 欅 惇志, 岡崎 直観. RNNによる遷移確率計算を用いた隠れマルコフモデル. 第242回自然言語処理研究会, 2019-NL-242(2), pp. 1–6, 奈良先端科学技術大学院大学（奈良県）, 2019年10月. （若手奨励賞）

Co-authored Publications

11. Youmi Ma, Tatsuya Hiraoka, and Naoaki Okazaki. Named Entity Recognition and Relation Extraction Using Enhanced Table Filling by Contextualized Representations. Journal of Natural Language Processing, vol. 29, No. 1, 38 pages , March 2022. (Accepted)

12. 植木 滉一郎, 平岡 達也, 岡崎 直観. 記事に忠実ではない訓練事例も活用した見出し生成モデルの忠実性の改善法. 言語処理学会第28回年次大会 (NLP2022), 6 pages, 2022年3月.

13. Youmi Ma, 平岡 達也, 岡崎 直観. 畳み込みニューラルネットワークを用いた表ラベリングによる固有表現認識と関係抽出. 言語処理学会第28回年次大会 (NLP2022), 6 pages, 2022年3月.

14. 昇 夏海, 平岡 達也, 丹羽 彩奈, 西口 佳佑, 岡崎 直観. 企業情報を考慮したキャッチコピーの自動生成. 言語処理学会第27回年次大会 (NLP2021), pp. 450–454, 2021年3月.

15. Youmi Ma, 平岡 達也, 岡崎 直観. BERTを用いたTable-Fillingによる固有表現抽出と関係抽出. 言語処理学会第27回年次大会 (NLP2021), pp. 1274–1279, 2021年3月.