令和 5 年度 修士論文

機械学習を用いた
文体の自動変換についての検討

龍谷大学　大学院　理工学研究科　情報メディア学専攻

学籍番号　：　T22M065
氏名　　：　安野 俊也

指導教員　：　三好 力 教授

内容梗概

日本語は文法構造や表現の多様性、敬語など非常に複雑な言語である。適切な表現方法は社会的背景や話者間の関係に応じて変化するため、コミュニケーションにおいて文体の変換は重要である。しかし、これまでの研究において、二言語間の機械翻訳の研究は盛んに行われているが、日本語の文体の変換に関する研究は少ない。そこで、本研究では CycleGAN を応用した日本語の文体を自動変換する手法を提案する。事前学習を行ったモデルを用いて、変換および逆変換を連続的に行うことで、文体の変換を実現するモデルである。最後に、提案手法と既存技術である Transformer を用いた手法と比較する。

Abstract

Japanese is a very complex language in terms of grammatical structure, diversity of expressions, and honorifics. Since appropriate ways of expression vary according to social context and inter-speaker relationships, stylistic translation is important in communication. However, while there have been many studies on machine translation between two languages, there have been few studies on stylistic translation of Japanese. Therefore, this study proposes a method for automatic conversion of Japanese sentence structure based on CycleGAN. This model achieves stylistic transformation by continuously performing transformation and inverse transformation using a pre-trained model. Finally, we compare the proposed method with an existing technique using Transformer.

目次

1 序論

1.1 研究背景

日本語は文法構造や表現の多様性、敬語など非常に複雑な言語である。日本語には書き言葉と話し言葉の違いや漢字、ひらがな、カタカナの 3 種類の文字の使用、文節の構造が多様であるなどの言語的特性があり、体言止め文や助詞の省略などの特徴は日本語の表現の豊かさを生み出している。敬語は尊敬語や謙譲語、丁寧語などが存在し、話者と聞き手の関係性に基づいて使い分けられている。

敬語の使用が社会的背景や話者間の関係に応じて変化するため、特に様々な表現方法がある日本語において、適切な文体を選択することはコミュニケーションに不可欠である。例えば、ビジネス文書や公式文書、学術論文などでは丁寧な表現が求められる一方で、SNS や個人的な文章ではカジュアルな表現が求められる。

文体変換は文章のスタイルを変更するプロセスであり、言語の柔軟性や表現の豊かさを反映している。例えば、ビジネス文書からカジュアルな文書への変換、敬語から丁寧語への変換、体言止め文から述語を含む文への変換などがある。

異なる二言語間の機械翻訳の研究は盛んに行われているが、日本語の文体変換を人手ではなく自動的に行なおうとする研究は少ない。文体の自動変換に関するこれまでの研究において、ルールベースや機械学習に基づいたアプローチなどの方法が試みられてきた。林らの研究 [1] では、体言止め文における文末の名詞のタイプ・係り受け関係・時制等の情報を用いた統計的な手法により、省略表現の補完を実現する方法を提案された。この研究では書き言葉を話し言葉に変換することに焦点を当てており、特にニュースの音声出力の自然さを目標としている。文末が名詞や代名詞で終わる「体言止め文」を、話し言葉に適した形式に変換する技術を紹介し、書き言葉と話し言葉の言語的差異を強調している。ニューステキストの省略された表現を補う統計的方法を用いて、合成音声の自然さを向上させることを目指している。また、下地らの研究 [2] においては、LSTM-RNN を用いた文体の変換手法を提案された。この研究では、One-hot ベクトルによって表現された文章を学習し、その後、学習モデルに入力して文の終わりまで次の単語を予測する。このプロセスを通じて、文体の変換を行う。実験を通じて、この手法で文体を変換することが可能であることが示されている。

しかし、これらの研究にはいくつかの課題が存在する。林らの研究に用いられる統計的な

手法においては人手で変換ルールを作成しており、目的が変わるごとに変換ルールを人手で作成し直す必要がある。さらに、この手法ではルールでの対応が困難な場合が存在し、柔軟な変換が行えないという課題がある。また、下地らの研究に用いられる LSTM においては文章を順番に読み込むため、文章の長さに比例して計算量が増加するという課題がある。その他にも並列処理が困難であるため、学習に時間がかかる、大規模なデータセットを用いることができないという課題がある。さらに、これらの研究においては、変換前と変換後の日本語表現が対になっている学習データを大量に用意することが難しいという課題がある。

1.2 研究目的

本研究では機械学習を用いて日本語の文体を自動変換する手法を提案し、変換性能を実験で評価する。その中でも大量の対になっている学習データを収集することが難しいとされている中で、対データが少い場合でも一定以上の精度で変換することを目的とする。

1.3 本論文の構成

本論文の構成は次の通りである。第 2 章では関連技術として機械翻訳に関する基本事項、本研究でも参考にした CycleGAN およびこれまでの日本語における文体の変換の事例を紹介する。第 3 章では本研究の提案手法として CycleGAN を応用した文体の自動変換システムの具体的な構成を説明する。第 4 章では提案手法を用いて実際に変換を行い、評価スコアを用いて行った実験の結果と考察について述べる。最後に、第 5 章では全体を通した結論と今後の展望について述べる。

2　関連技術

2.1　機械翻訳

機械翻訳とは、コンピュータを利用して、ある自然言語を別の表現の自然言語に変換を行うものである [3]。以下に、機械翻訳の代表的な 3 つのアプローチを示す。

2.1.1　ルールベース機械翻訳

ルールベース機械翻訳は、人間が作成した言語の規則、文法、辞書などのルールに基づいて翻訳を行うものである。このアプローチでは翻訳のための規則を定義し、その規則をコンピュータに適用して翻訳を行う。

ルールベース機械翻訳の主な特徴は、その透明性と予測可能性である。システムは定義された規則に従って動作するため、翻訳のプロセスを理解しやすく、エラーの原因を特定しやすい。また、ルールに基づいて翻訳を行うため、学習データが少なくても翻訳を行うことができる。

しかし、ルールベース機械翻訳では翻訳のための規則を人間が作成する必要があるが、その言語におけるルールをすべて網羅した規則を作成することは複雑で時間がかかる。また、自然言語は表現が多様で時や場所における変動が大きいため、規則に基づいて完璧な翻訳を出力することは難しい。さらに、文学作品などのように、文法に従わない表現を含む文章に対しては、ルールベース機械翻訳では翻訳を行うことができない場合がある。

これらの理由からルールベース機械翻訳はニューラル機械翻訳などのより高度な手法に取って代わられつつある。ただし、技術的なドキュメント屋法律文書などの文法や語彙が限定された特定の専門分野においてはその正確性と一貫性が高く評価されており、依然として使用されている。

2.1.2　統計的機械翻訳

統計的機械翻訳とは、大量の対訳データから統計的なモデルを構築し、そのモデルを用いて翻訳を行うものである。このアプローチでは言語の出現頻度やパターンを分析し、その情報を基に翻訳を行う。

統計的な機械翻訳の主な特徴は、ルールベース機械翻訳と比べて柔軟性が高いことである。特定の言語規則に依存せず、大量の対訳データから自動的に翻訳のためのモデルを構築する

ため、翻訳のための規則を人間が作成する必要がない。また、新しい言語ペアに対応するためには適切なコーパスを追加するだけであり、拡張が容易である。

しかし、統計的機械翻訳においてもいくつかの課題が存在する。この手法は利用可能な対訳データに依存するため、対訳データが少ない言語ペアにおいては翻訳の精度が低下する。また、文の意味が複雑である場合や文化的なニュアンスが含まれる場合には、翻訳の精度が低下する場合がある。さらに、統計的機械翻訳は計算コストが高く、リアルタイムでの翻訳には向いていない場合がある。

2.1.3 ニューラル機械翻訳

ニューラル機械翻訳とは、深層学習やニューラルネットワークを用いて翻訳を生成する機械翻訳の一種である。このアプローチでは、複数のニューラルネットワークを組み合わせて大量のテキストデータから言語間の関係を学習する。

一般的なニューラル機械翻訳は、以下の 2 つから構成される。

エンコーダ エンコーダは、入力文をベクトル表現に変換するネットワークである。入力されたテキストが分析され、それを言語の意味を表す高次元のベクトル空間にマッピングする。このベクトル表現は、文の意味内容を包括的に捉えたものであり、単なる単語の並び以上の情報を含んでいる。

デコーダ デコーダは、エンコーダで得られたベクトル表現を元に、出力文を生成するネットワークである。このプロセスでは、内部表現からターゲット言語 (翻訳先の言語) における適切な文を生成するために、エンコーダと同様のネットワークが使用される。

ニューラル機械翻訳の主な特徴は、精度の高さや自然で流暢な翻訳を生成する能力である。また、もう一つの特徴は、文全体を考慮して翻訳を行う「シーケンス・ツー・シーケンス」(Sequence to Sequence) モデルの使用である。これにより、より長い文脈を考慮した翻訳が可能となり、文法的にも意味的にも正確な翻訳が生成される。また、Attention メカニズムを導入することで、翻訳時に特に重要な単語やフレーズに焦点を当てることができ、翻訳の精度がさらに向上する。

ニューラル機械翻訳は、統計的機械翻訳と比較して、特に長い文や複雑な文構造を持つ文の翻訳において優れた性能を発揮する。しかし、高品質な翻訳を生成するためには大量の学習データと計算リソースが必要であり、これが主な課題になっている。

2.2 GAN

GAN[4] とは、Generative Adversarial Network(敵対的生成ネットワーク) と呼ばれる生成モデルの一種である。GAN はノイズから画像を生成する Generator と Generator が生成した画像と実際の画像を判別する Discriminator の 2 つのネットワークから構成される。

Generator の役割は、実際に存在しないデータ (例えば、画像や音声など) を生成することである。この生成されたデータは実際のデータと区別できないようにする必要がある。一方、Discriminator の役割は、Generator が生成したデータと実際のデータを識別することである。

GAN の学習においては、Generator は、Discriminator をだますようなリアルなデータを生成しようと試る。一方、Discriminator は、より正確に本物と偽物を区別しようとする。この競争的なプロセスによって、Generator は徐々に本物に近いデータを生成する能力を向上させ、Discriminator は識別能力を高める。

しかし、動作の不具合が多く、学習が不安定であるという問題がある。例えば、ある段階を超えると学習が進まず学習不足になる「勾配消失問題」や、入力された画像と酷似した画像ばかり生成してしまう「モード崩壊問題」などがある。そのため、GAN の学習においては Generator と Discriminator のバランスを調整する必要がある。

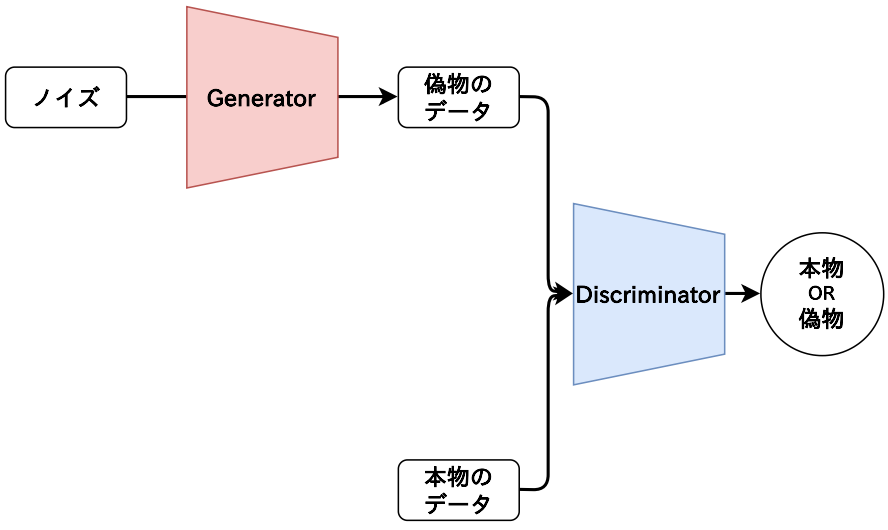

図 1　GAN の概要

2.3 CycleGAN

CycleGAN[5] は異なる 2 つのドメイン間で画像を変換するための敵対的生成ネットワーク (GAN) の一種である。この技術は、一方のドメインの画像をもう一方のドメインの画像に変換することができ、その過程でドメイン間のマッピングを学習することができる。例えば、夏の風景を冬の風景に、馬をシマウマに、画像を変換することができる。

CycleGAN の特徴はペアになっていない画像を使用して学習することができる点にある。従来の GAN モデルでは、変換前と変換後の画像がペアになっている必要があったが、CycleGAN はこの要件を必要としない。

また、CycleGAN は「サイクル一貫性損失」と呼ばれる独自の機能を使用している。これは、元のドメインに戻った際に入力画像が再構築されることを保証する機能で、変換の品質を向上させるのに役立つ。

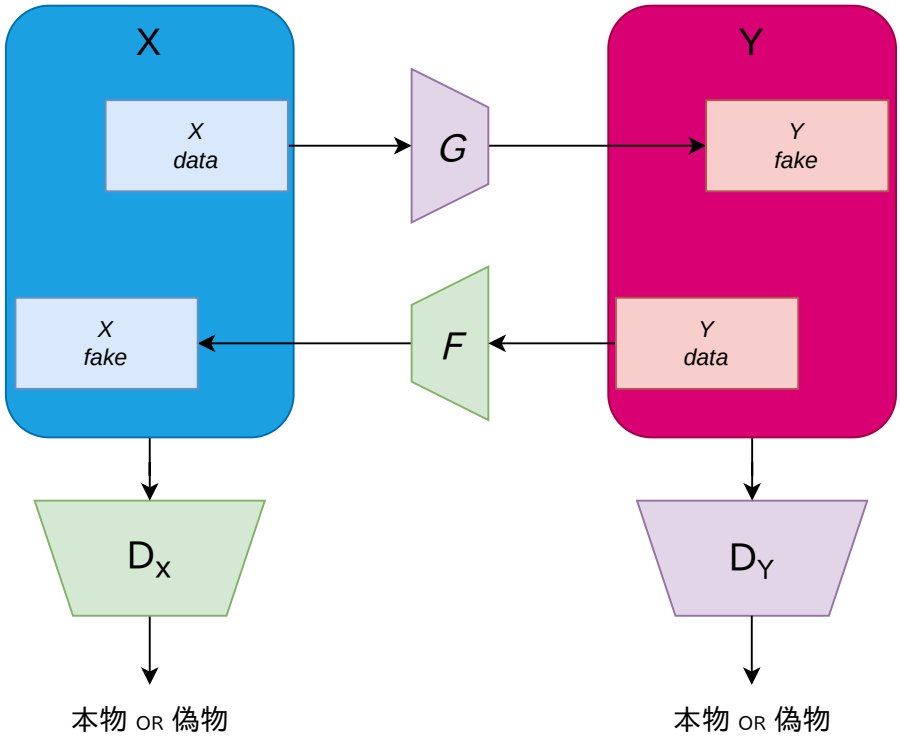

図 2　CycleGAN の概要

2.4 Seq2Seq

Seq2Seq[6] とは、特に自然言語処理（NLP）分野で用いられる、シーケンスデータ（例えば、文や単語の列）を別のシーケンスデータに変換するためのニューラルネットワークモデルである。このモデルは、機械翻訳、自動要約、質問応答システムなど多様なタスクに応用されている。

Seq2Seq モデルの基本的な構成は、エンコーダとデコーダという 2 つの主要な部分から成り立っている。エンコーダは入力シーケンス (例えば、翻訳元の文) を取り込み、それを固定長のベクトル表現に変換する。このベクトルは、入力シーケンスの意味的な内容を圧縮して表現している。次に、デコーダはこのベクトルを使用して、目的の出力シーケンス (例えば、翻訳先の言語の文) を生成する。

Seq2Seq モデルの重要な特徴の一つは、変数長の入力と出力を扱う能力である。れにより、モデルは任意の長さの文を入力として受け取り、それに対応する任意の長さの文を出力することができる。

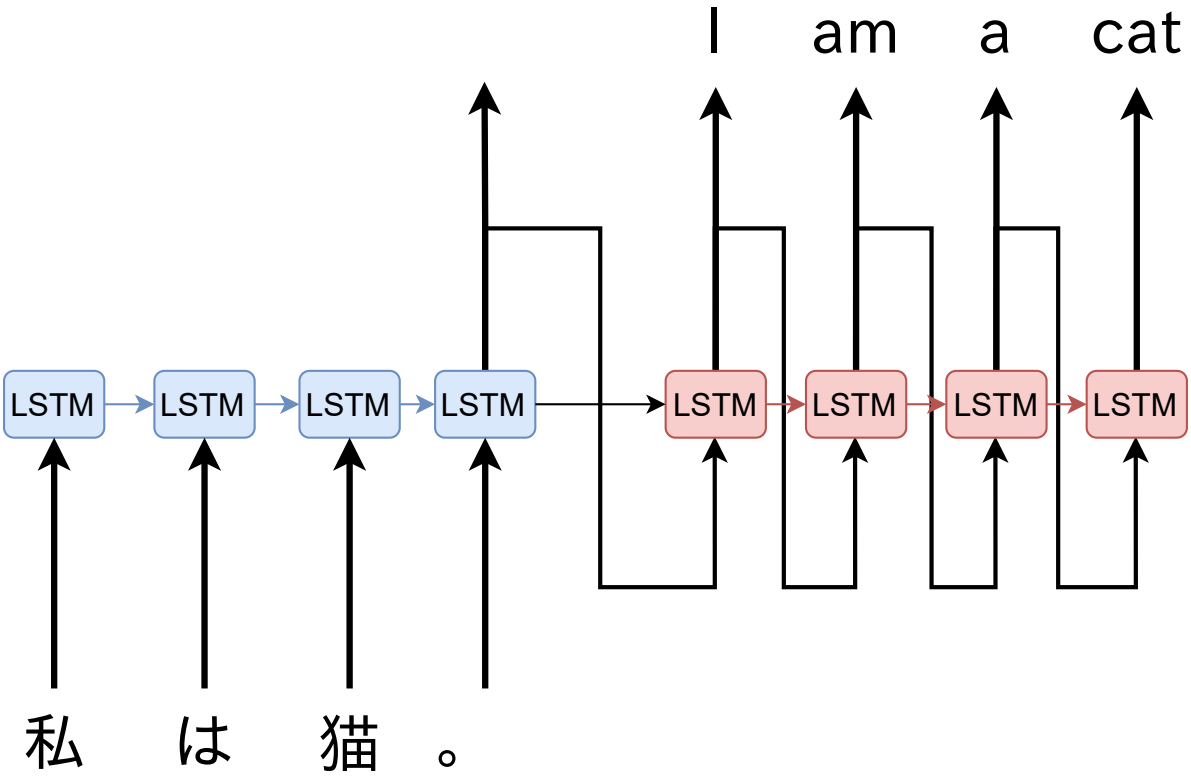

図 3　Seq2Seq の概要

また、Seq2Seq モデルはしばしば Attention メカニズムと組み合わされる。Attention は、デコーダが出力の各部分を生成する際に、エンコーダの出力のどの部分に「注意」を払うべきかを決定する機能である。これにより、モデルはより精度の高い翻訳や要約を生成することが可能になる。

Seq2Seq モデルは、その効率性と精度の高さから、現代の NLP タスクにおいて広く採用されている。特にニューラル機械翻訳の分野においては、従来の統計的翻訳モデルを大きく上回る性能を示しており、多くの商用翻訳システムの基盤技術として用いられている。

2.5 Transformer

Transformer[7] とは、2017 年に「Attention is All You Need」という論文で初めて紹介されたアーキテクチャであり、特に自然言語処理（NLP）分野で広く用いられている。

Transformer の主要な特徴は、アーキテクチャが完全に Attention に基づいて構築されていることである。従来の Seq2Seq モデルとは異なり、Transformer はリカレント層 (RNN) や畳み込み層 (CNN) を使用せず、代わりに「自己注意」(self-attention) と呼ばれるメカニズムを活用している。自己注意メカニズムにより、モデルは入力シーケンス内の各要素が他のすべての要素とどのように関連しているかを効率的に計算できる。

Transformer の利点は、並列処理の容易さと長い距離の依存関係を扱う能力にある。従来のリカレントや畳み込みモデルでは、シーケンスデータを逐次的に処理する必要があったが、Transformer はシーケンス内の全ての要素を同時に処理することができる。これにより、トレーニングの速度が大幅に向上する。

Transformer モデルは、特に Google の「BERT」(Bidirectional Encoder Representations from Transformers) や OpenAI の「GPT」(Generative Pre-trained Transformer) など、さまざまな大規模な事前学習済み言語モデルの基盤として用いられている。

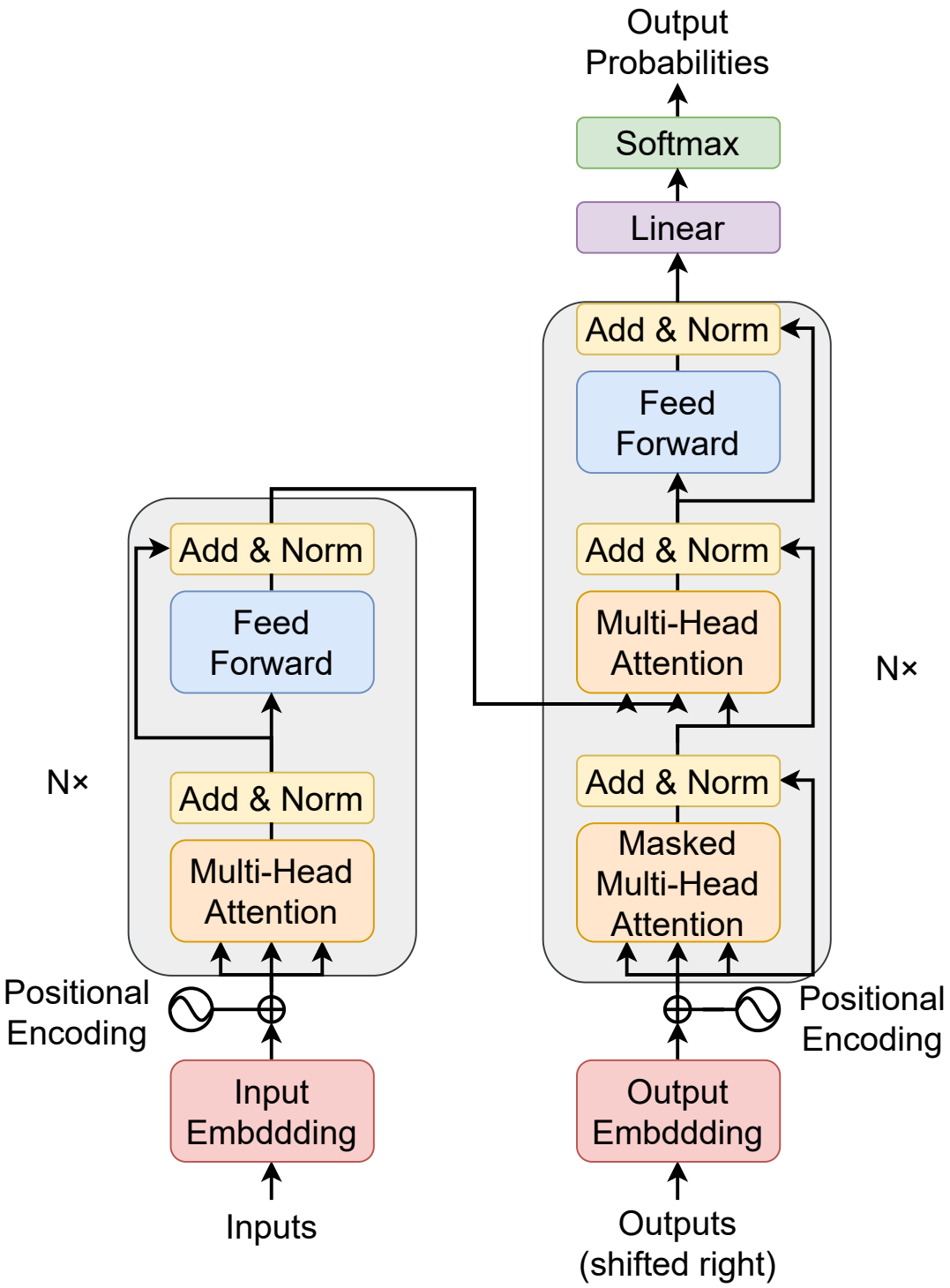

図 4　Transformer の概要

2.6 Attention

Attention とは、深層学習、特に自然言語処理分野で用いられる概念であり、ニューラルネットワークが特定の情報に注意を向けることを可能にする仕組みである。これはモデルが大量のデータの中から関連性の高い情報に焦点を合わせ、無関係または重要でない情報を無視するのに役立つ。

Attention メカニズムの基本的なアイデアは、入力データの各部分が出力の異なる部分にどの程度影響を与えるかをモデル化することである。例えば、機械翻訳のタスクにおいて、ある単語の翻訳時に、元の文のどの単語に注意を払うべきかを判断することができる。

Attention の利点の一つは、モデルが長いシーケンスデータ内の関連情報を効果的に捉える能力である。従来のシーケンスモデル (例えば、リカレントニューラルネットワーク) では、逐次的に処理を行うため、長い距離の依存関係を捉えることが難しい。モデルは、入力シーケンスの任意の部分に注意を向け、重要な情報を抽出することができるため、より複雑な文脈や長距離の関係を処理することができる。

この Attention は、Transformer モデルで中心的な役割を果たしている。また、Attention は可視化が可能であり、モデルがどの入力に焦点を当てているかを明らかにすることで、モデルの解釈性と透明性を高めることができる。これにより、モデルの決定プロセスを理解しやすくなり、さらなる改善や調整が容易になる。

2.7 交差エントロピー誤差

交差エントロピー誤差 (Cross Entropy Error)[8] とは、主に分類問題において用いられる誤差関数の一つである。この誤差関数は、モデルの出力が真の確率分布からどれだけ離れているかを評価するために使用される。特に、二値分類や他クラス分類の問題で広く使用されている。

以下に、交差エントロピー誤差の計算式を示す。

$$H(p, q) = -\Sigma_x p(x) \log q(x) \tag{1}$$

p は真の確率分布、q は出力の確率分布である。クラス分類においては、真の確率分布は one-hot ベクトルで表され、正解クラスの要素が 1 で、それ以外の要素が 0 である。そのため、

正解クラスが 1 のときにのみ誤差が計算される。

2.8 評価指標

2.8.1 BLEU

BLEU[9] とは、機械翻訳の出力品質を評価するために広く使用されるメトリックである。BLEU スコアは、機械翻訳の結果と人間が行った参照翻訳との間の類似性を数値で表す。この評価は、特に n-gram の一致に基づいている。BLEU スコアは 0%から 100%までの範囲で表され、高い値はより良い翻訳品質を示す。

BLEU スコアの計算では、まず機械翻訳の出力と一つ以上の参照翻訳の間で、n-gram の一致数をカウントする。次に、この一致数を機械翻訳の n-gram の総数で割ることにより、一致率を求める。さらに、翻訳の文が参照翻訳よりも短くなることを避けるために、翻訳の長さに対するペナルティが適用されることがある。最終的な BLEU スコアは、これらの一致率とペナルティを組み合わせて算出される。

以下に、BLEU スコアの計算式を示す。

$$BLEU = BP_{BLEU} \times \exp(\sum_{n=1}^{N} w_n \log p_n) \tag{2}$$

ただし、

$$P_n = \frac{\Sigma_i \text{翻訳文 } i \text{ と参照訳 } i \text{ で一致した } n-gram \text{ 数}}{\Sigma_i \text{翻訳文 } i \text{ 中の全 } n-gram \text{ 数}} \tag{3}$$

$$w_n = \frac{1}{N} \tag{4}$$

である。BP_{BLEU} とは翻訳文が参照訳よりも短くなることを避けるためのペナルティである。翻訳文の方が長い場合は 1 であり、ペナルティは適用されない。

しかし、BLEU スコアは n-gram の一致に基づいているため、活用形や語順の違いを考慮することができない。そのため、類義語や同義語の違いに対しては弱いという問題がある。その結果、類義語が使われた時には参照訳とは異なるとみなされ、スコアが低くなることがある。

その一方、翻訳文の内容が原文とは意味的には異なっているにもかかわらず、BLEU スコアが高くなることがある。例えば、文章に「not」という単語が入った誤訳が存在するとき、文章の意味は逆転してしまうが、その他の単語が完全に一致している場合、BLEU スコアは

限りなく 100 に近づくことがある。

2.8.2 METEOR

METEOR[10] とは、機械翻訳の出力の品質を評価するために使用されるメトリックの一つである。METEOR は、機械翻訳が生成した文と参照翻訳 (人間が行った翻訳) との間の一致を評価するが、BLEU とは異なり、単語レベルの一致だけでなく、同義語や文法的変形を考慮した柔軟な一致も含まれる。

METEOR スコアの計算では、まず機械翻訳出力と参照翻訳の間で単語の一致を識別する。この際、同義語や語幹の一致も考慮される。次に、これらの一致に基づいて精度 (precision) と再現率 (recall) を計算し、この二つのバランスをとるために調和平均 (F スコア) が求められる。さらに、文の順序の一致を評価するためにペナルティが加えられることがある。

METEOR スコアは、機械翻訳が生成した文が参照翻訳とどれだけ似ているかを示す。スコアは 0 から 1 の範囲であり、1 に近いほど高品質な翻訳を意味する。このメトリックは、BLEU に比べて言語のニュアンスや文脈に対する違いに強いため、翻訳の流暢さや自然さを重視する場合に有効である。

以下に、METEOR スコアの計算式を示す。まず、精度 P と再現率 R の超過平均を以下の式で求める。

$$F_{mean} = \frac{P \times R}{\alpha \times P + (1 - \alpha) \times R} \tag{5}$$

α は精度と再現率のバランスを調整するパラメータであり、どちらの要素を重視するかを決定する。

$$Penalty = \gamma \times (\frac{\#chunks}{\#unigrams\_matched})^{\beta} \tag{6}$$

ここで、$\#chunks$ は一致した単語の塊の数、$\#unigrams\_matched$ は一致した単語の数である。β および γ は、ペナルティの強さを調整するパラメータである。ほとんどの一致した単語が一致した塊の中にある場合、ペナルティは小さくなる。

最終的な METEOR スコアは、以下の式で求められる。

$$Score = F_{mean} \times (1 - Penalty) \tag{7}$$

最終的な METEOR スコアは、F スコアとペナルティの積で表される。なお、α、β、γ はハイ

パーパラメータである。

本研究では [10] を参考にしてそれぞれ 0.9、3、0.5 とした。

3　提案手法

本章では、2 章で述べた関連技術を踏まえて、本研究で提案する文体変換手法について述べる。

本研究では、CycleGAN を手がかりとして、対データの少ない場合においても高い精度で文体変換を行うことができる手法を提案する。CycleGAN の特徴として、学習をサイクル的に行うことで、より高い精度で変換を行うことができるようにすることが挙げられる。本研究では、この特徴を活かし、自然言語処理の分野における文体変換にも適用することで、対データの少ない場合においても高い精度で文体変換を行うことができるようになるのではないかと考えた。

また、提案手法では Transformer を用いた文体変換モデルを提案する。Seq2Seq モデルではなく、Transformer を用いる理由としては、Transformer は Seq2Seq モデルに比べて長い文脈を考慮する能力が高く、文体変換においてもより高い精度を期待できるためである。また、Transformer は Attention メカニズムを用いており、文の中で特に重要な単語やフレーズに焦点を当てることができるため、文体変換においてもより適切な変換が行えると考えられる。さらに、提案手法においては追加データを大量に用いることによって、既存技術であるモデルをより高い精度で文体変換を行うことができるように学習させることを目指しているため、処理時間などの問題がある Seq2Seq モデルよりも Transformer を用いることが適していると考えられる。

提案システムの全体像について図 5 に示す。

提案手法は学習フェーズと変換フェーズの 2 つのフェーズで構成される。学習フェーズでは、元の文体の文章を目的の文体の文章に変換するためのモデルを学習する。変換フェーズでは、学習フェーズで学習したモデルを用いて、実際に文体変換を行う。

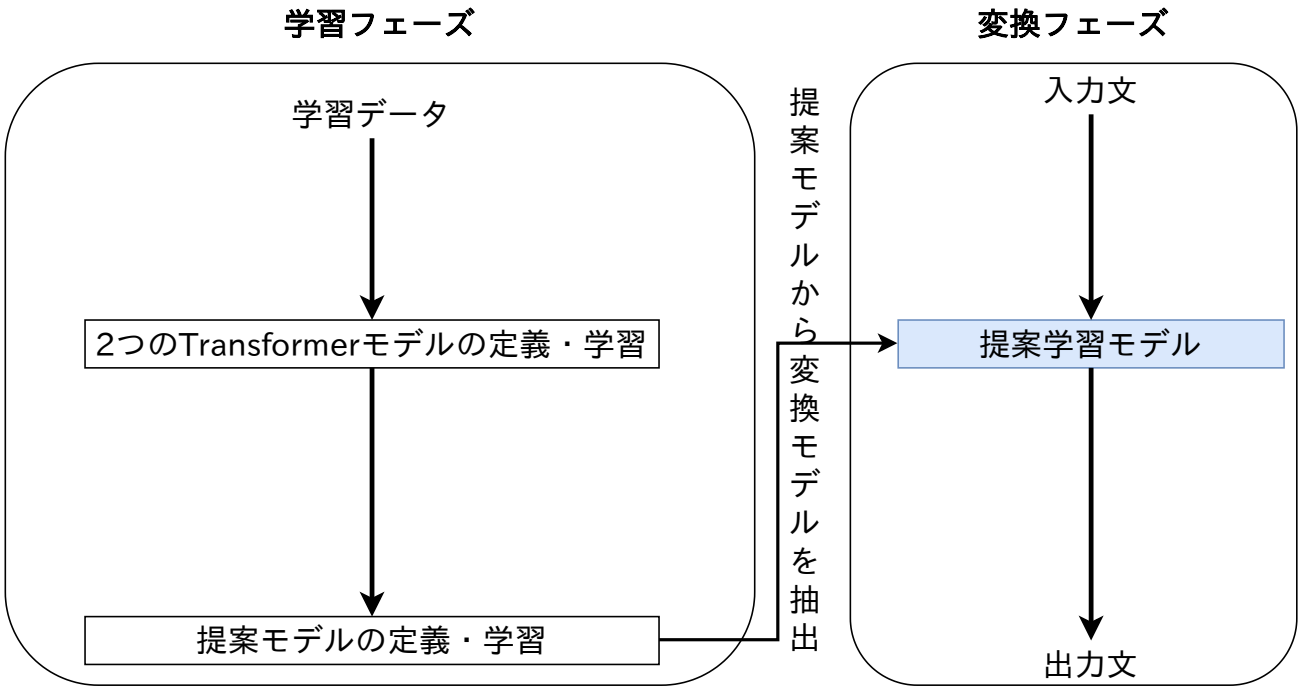

図 5　提案手法の全体構成図

3.1　学習フェーズ

本節では、文章を学習し、モデルを生成するフェーズについて述べる。提案手法の学習過程の全体図を図 6 に示す。

まず、学習データには、対応させた文体の異なる 2 種類のテキストデータを用意する。そして、学習に用いる文章に対して、Mecab を用いて分かち書きを行い、出現単語をまとめた単語辞書を作成し、単語を ID に変換する。その後、データを学習用、検証用、テスト用に分割する。

次に、提案モデルで用いるために、事前学習として Transformer を用いて文章を学習する。ここでは、元の文体の文章が入力となり、目的の文体の文章が出力となるように学習を行う。次に、先ほどとは反対に、目的の文体の文章が入力となり、元の文体の文章が出力となるように学習を行う。このようにして、変換モデル及び逆変換の 2 つのモデルを生成する。

提案モデルの学習方法を図 7 に示す。提案モデルでは先ほど事前学習した変換モデルと逆変換モデルを直列に接続して学習する。まず、変換モデルに元の文体の文章を入力し、目的

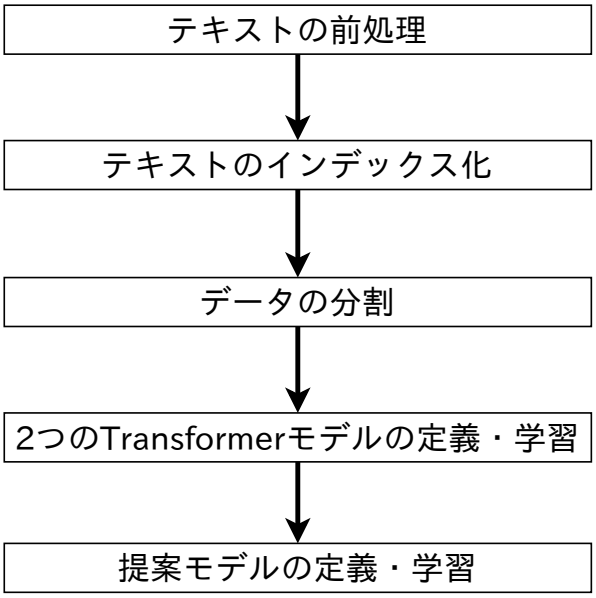

図 6　学習フェーズの概要

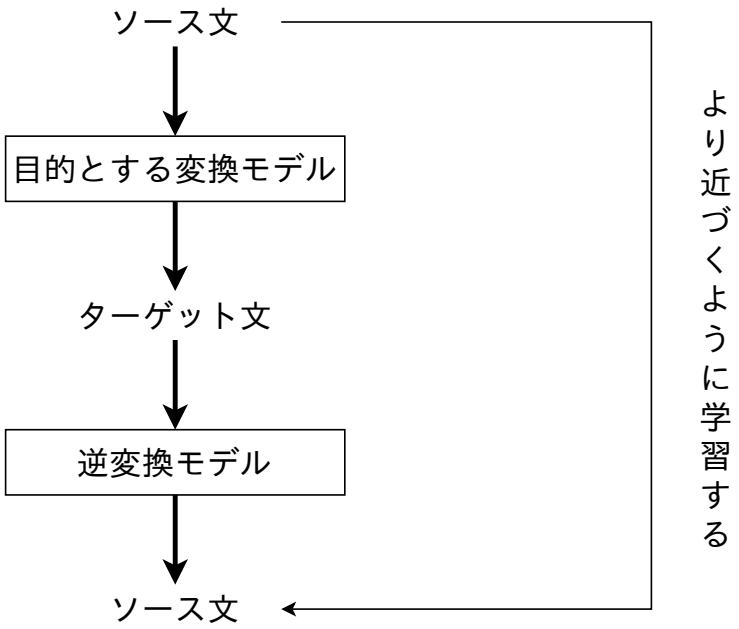

図 7　提案手法の学習の概要

の文体の文章を出力する。その後、逆変換モデルに先ほどの出力を入力し、元の文体の文章を出力する。さらに、変換モデルに入力した元の文体の文章と逆変換モデルから出力された元の文体の文章を交差エントロピー誤差を用いて損失を計算し、求められた損失を用いて変換モデルおよび逆変換モデルに逆伝播を行う。このようにして、変換と逆変換を一つのモデルとして連続して行うことで、より高い精度の変換を行うことを目的とする。また、この学

習方法では元の文体の文章のみを用いて学習を行うため、目的の文体の文章との対データが存在しない場合でも学習を行うことができ、多数の新規データが利用することが可能である。さらに、変換および逆変換を連続して行って学習することで、モデルが入力データの意味を保持しつつ、文脈に応じた適切な表現形式を選択する能力を獲得することが期待できるため、意味の同一性を保ちつつ文体変換を行うことができると考えられる。

提案手法のモデルの損失関数は、以下の式で表される。

$$Loss(p_{source}, q_{source'}) = -\Sigma_x p(x) \log q(x) \tag{8}$$

p_{source} は元の文体の文章、$q_{source'}$ は連続学習によって変換された文体の文章である。この損失関数を用いることで、目的の文体の文章が元の文体の文章に変換されることを目指す。

3.2 変換フェーズ

本節では、3.1 節の学習フェーズで生成したモデルを用いて、文体変換を行うフェーズについて述べる。

変換フェーズの全体図を図 8 に示す。

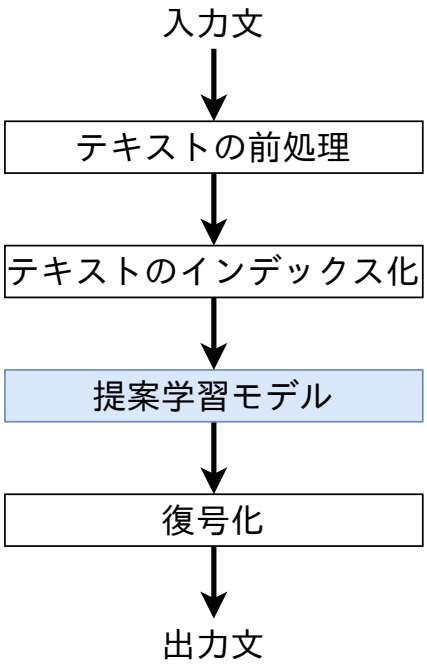

図 8　変換フェーズの概要

3.1 節と同様にして、入力文に対して分かち書きを行う。その後、単語辞書を用いて単語を ID に変換し、ベクトル化を行う。そして、3.1 節で生成したモデルにベクトルを入力し、変換

を行う。変換されたベクトルを単語辞書を用いて単語に変換し、出力文を得る。

4　評価実験

本章では、3 章で提案した手法を用いて実際に文体変換を行い、評価スコアを用いて行った実験の結果と考察について述べる。

4.1　概要

まず、本実験では文体変換の性能を評価するために、METEOR スコアを用いて評価を行う。実験は、対データを用いた事前学習ありの場合と、対ではないデータを用いた事前学習ありの場合と、事前学習なしの 3 つのケースで行う。ここでの対ではないデータとは、元々対であったデータをランダムにシャッフルすることで対を外したデータのことである。これを用いることで、CycleGAN の特徴である対ではないデータを用いた学習を行うことができるという点を文体変換に適用できるかを調べることができる。また、一項目に対して 3 回の評価を行い、その平均値を最終的なスコアとする。さらに、過学習を防ぐために評価損失が 5 回連続で減少しなかった場合に学習を終了する早期終了を行う。今回、METEOR スコアを用いる理由としては、BLEU スコアは n-gram の一致に基づいているため、活用形や語順の違いを考慮することができないため、文章の一部が変換される文体変換においては METEOR スコアの方が適していると考えられるためである。

4.2　実験環境

表 1　実験環境

OS	Windows 10
CPU	Intel(R) Core(TM) i5-11400 CPU @ 2.60GHz
GPU	GeForce RTX 3060 12GB
メモリ	64GB
Python	3.8.10
Pytorch	1.11.0+cu113
nltk	3.7
Mecab	0.996

本実験における実験環境を表 1 に示す。また、実験に使用したデータセットには丸山らのやさしい日本語コーパス [11] を用いた。さらに、損失関数には交差エントロピー誤差を用いた。

実装はすべて Python で行い、提案手法のモデルの実装には PyTorch を用いて行い、学習には NVIDIA の GPU の RTX 3060 12GB を用いた。また、分かち書き等の前処理には MeCab を用いた。

4.3 実験結果

4.3.1 対データを用いた事前学習ありの場合

学習データ 100 の場合

事前学習データ数が 100 の場合の実験結果を図 9、図 10 に、増減率を表 2 に示す。また、追加データの数が 100 の時のそれぞれの損失を図 11、図 12 に示す。Transformer および 100 個から 10000 個の追加データを用いた追加学習を行った提案モデルにおいても最小値はすべて 0.0 であった。また、最大値および平均値において、追加データの数に関わらず METEOR スコアは減少しており、追加データの数が 10000 の場合において最大値、平均値ともに最も減少した結果となった。

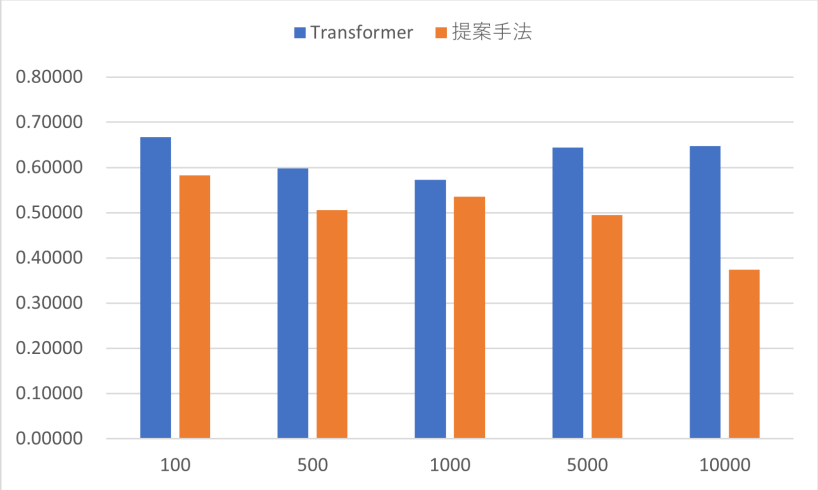

図 9 事前学習データ数 100 の最大値

表 2 事前学習データ数 100 の METEOR スコアの増減率

追加データ数	最大値 (%)	平均値 (%)
100	-12.59587	-25.86258
500	-15.42422	-31.12745
1000	-6.37666	-15.22415
5000	-23.23315	-15.72912
10000	-42.36099	-47.52262

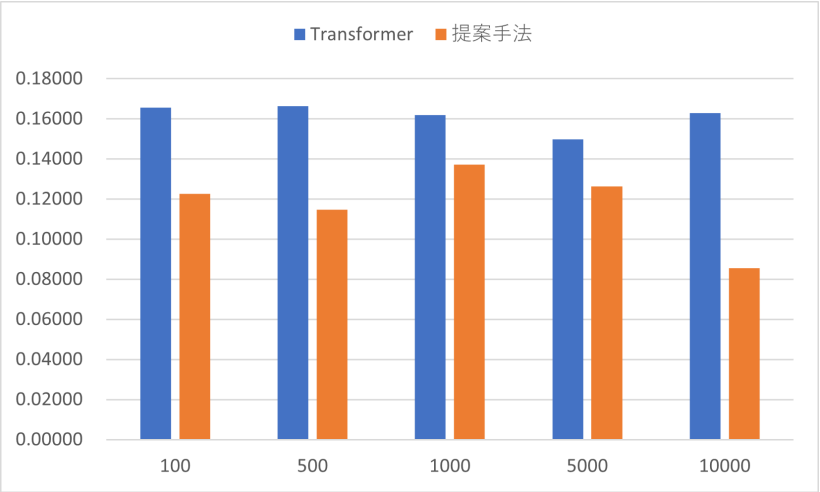

図 10　事前学習データ数 100 の平均値

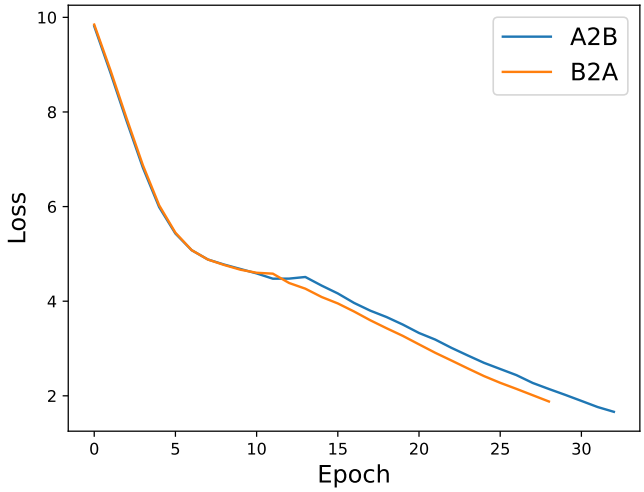

図 11　事前学習データ数 100 における事前学習時の損失

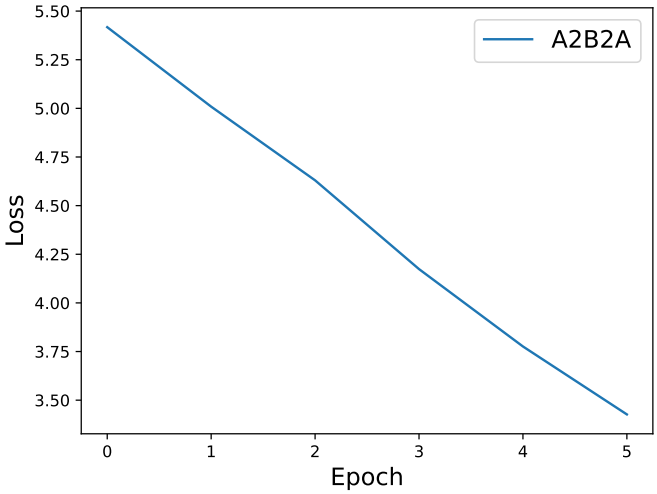

図 12　事前学習データ数 100 における提案手法学習時の損失

学習データ 1000 の場合

事前学習データ数が 1000 の場合の実験結果を図 13、図 14、図 15 に、増減率を表 3 に示す。また、追加データの数が 100 の時のそれぞれの損失を図 16、図 17 に示す。最小値においては、追加データの数が 100、1000、10000 の場合において増加し、その他の追加データの数の場合において減少した。最大値および平均値においては、追加データの数に関わらず METEOR スコアは減少しており、追加データの数が 10000 の場合において最大値、追加データの数が 500 の場合において平均値ともに最も減少した結果となった。

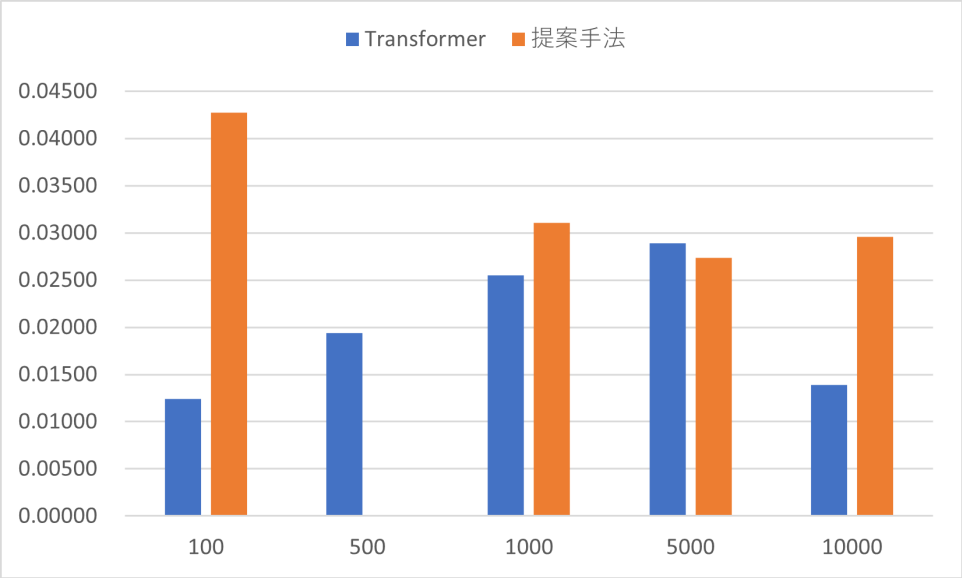

図 13　事前学習データ数 1000 の最小値

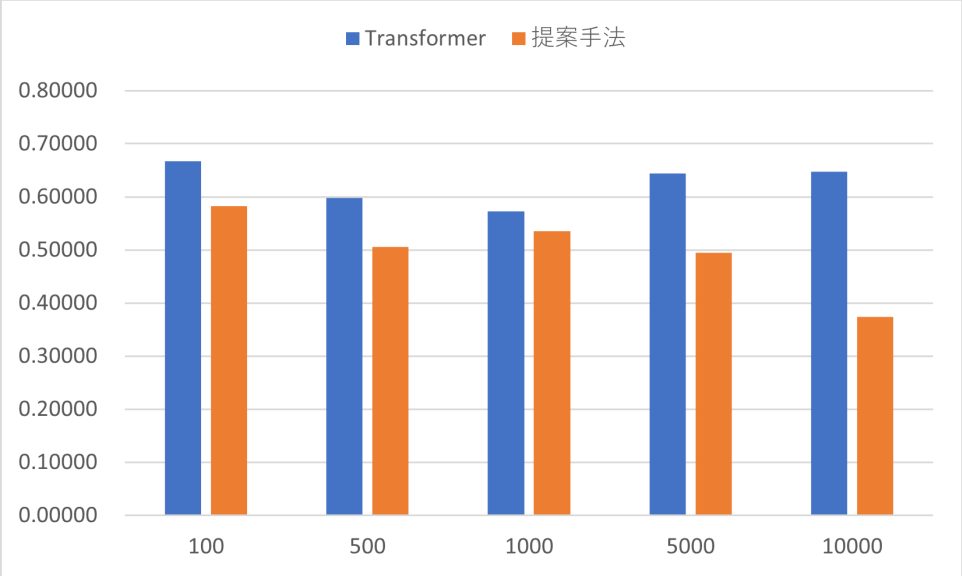

図 14　事前学習データ数 1000 の最大値

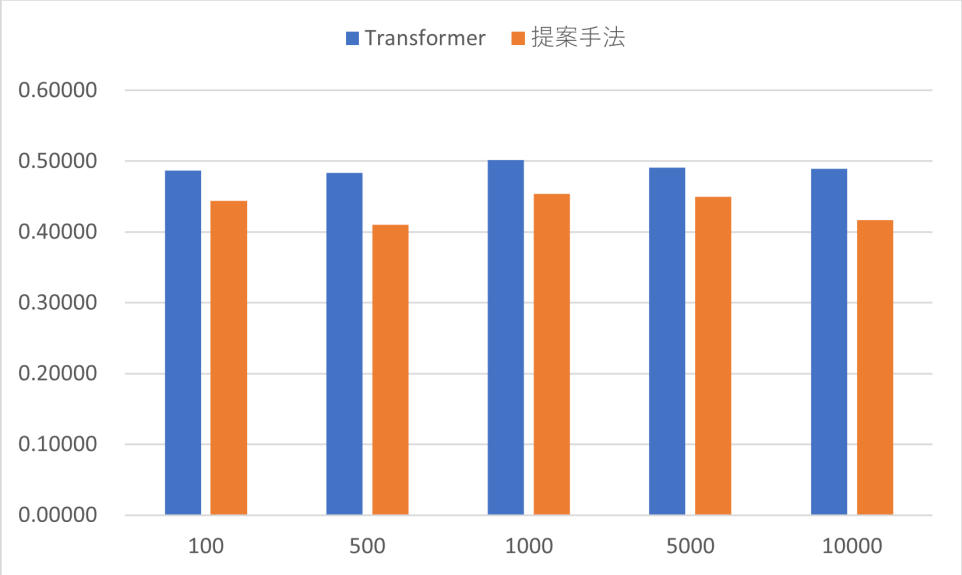

図 15　事前学習データ数 1000 の平均値

表 3　事前学習データ数 1000 の METEOR スコアの増減率

追加データ数	最小値 (%)	最大値 (%)	平均値 (%)
100	243.65596	-0.00288	-8.80901
500	-100.00000	0.00206	-15.30066
1000	21.75484	-0.00645	-9.56074
5000	-5.44608	-0.00964	-8.32398
10000	112.91901	-0.00987	-14.85210

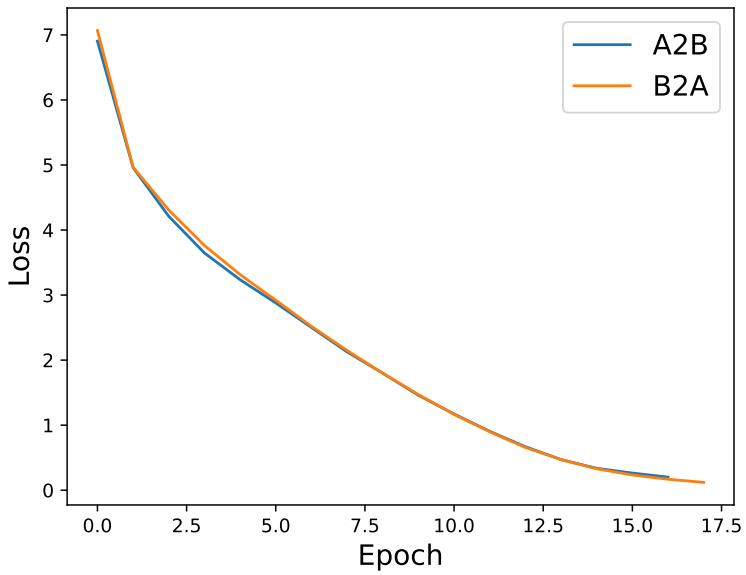

図 16　事前学習データ数 1000 における事前学習時の損失

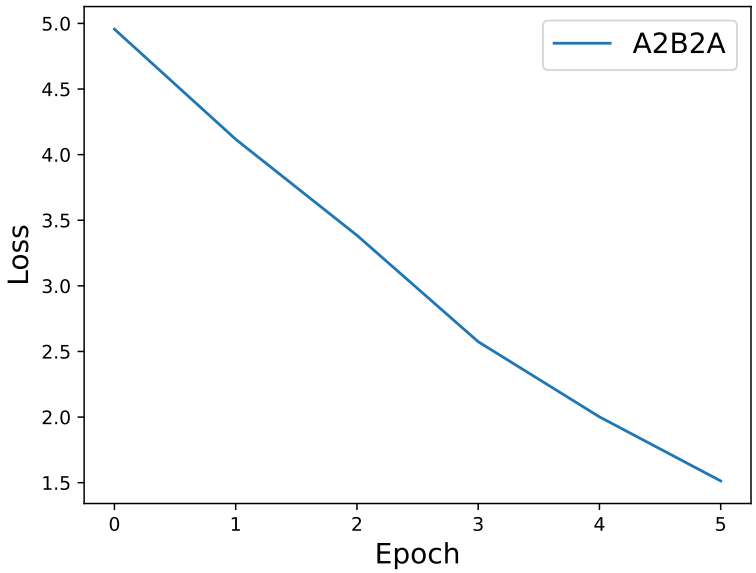

図 17　事前学習データ数 1000 における提案手法学習時の損失

学習データ 10000 の場合

事前学習データ数が 10000 の場合の実験結果を図 18、図 19、図 20 に、増減率を表 4 に示す。また、追加データの数が 100 の時のそれぞれの損失を図 21、図 22 に示す。最小値においては、追加データの数が 100、5000、10000 の場合において増加し、その他の追加データの数の場合において減少した。最大値においては、追加データの数が 10000 の場合においてのみ増減がなく、その他の追加データの数の場合において減少した。平均値においては、追加データの数に関わらず METEOR スコアは減少しており、追加データの数が 500 の場合において最も減少した結果となった。

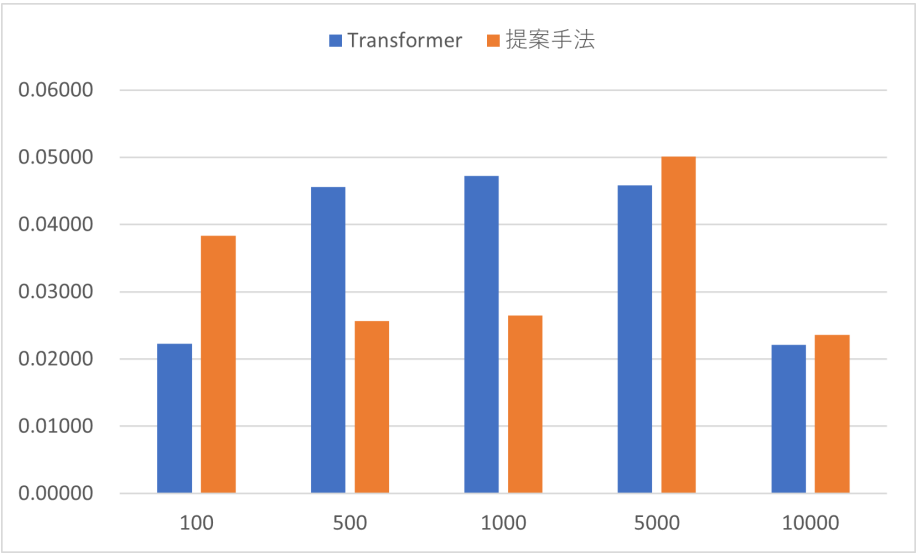

図 18　事前学習データ数 10000 の最小値

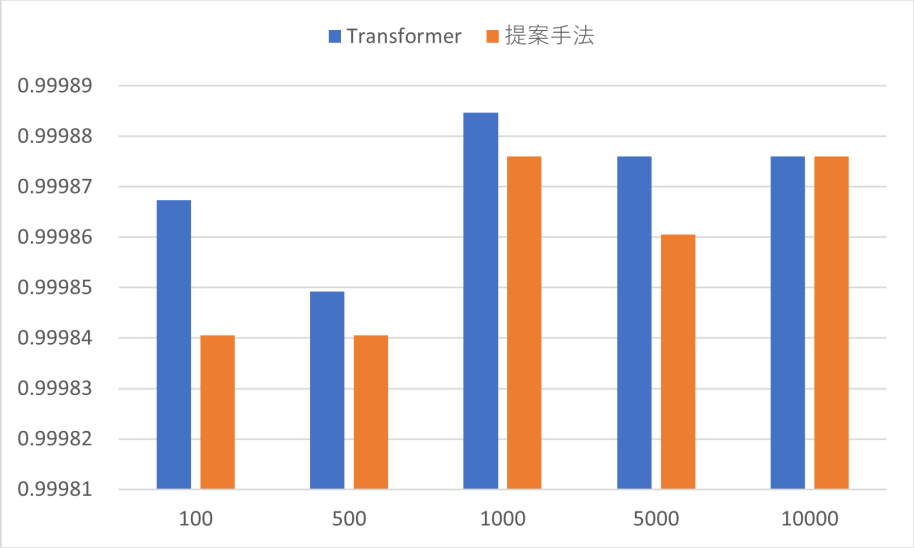

図 19　事前学習データ数 10000 の最大値

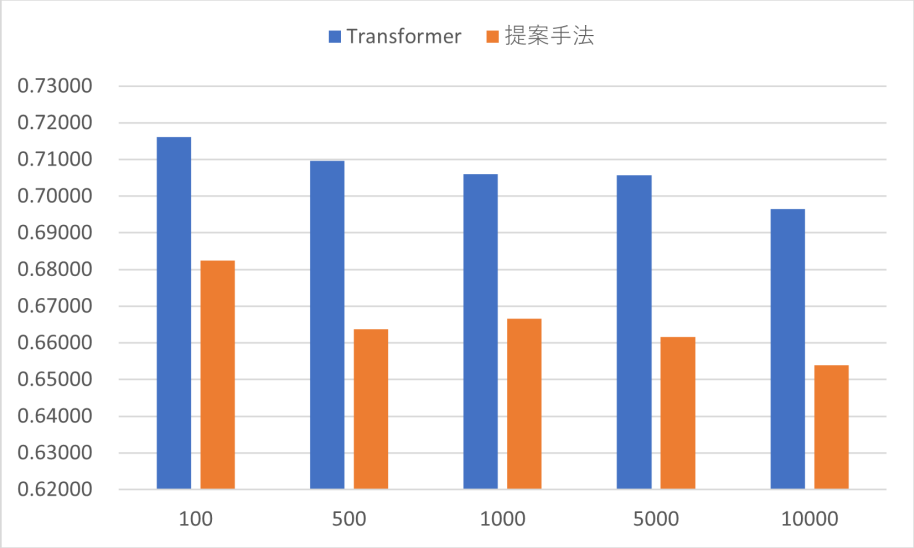

図 20　事前学習データ数 10000 の平均値

表 4　事前学習データ数 10000 の METEOR スコアの増減率

追加データ数	最小値 (%)	最大値 (%)	平均値 (%)
100	71.87666	-0.00268	-4.71322
500	-43.75110	-0.00087	-6.46778
1000	-43.94635	-0.00087	-5.58848
5000	9.31607	-0.00155	-6.25968
10000	6.49788	0.00000	-6.11082

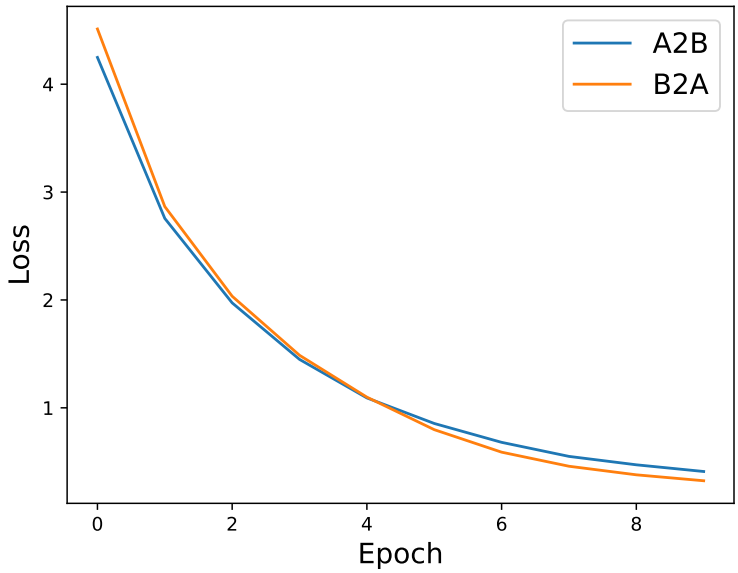

図 21　事前学習データ数 10000 における事前学習時の損失

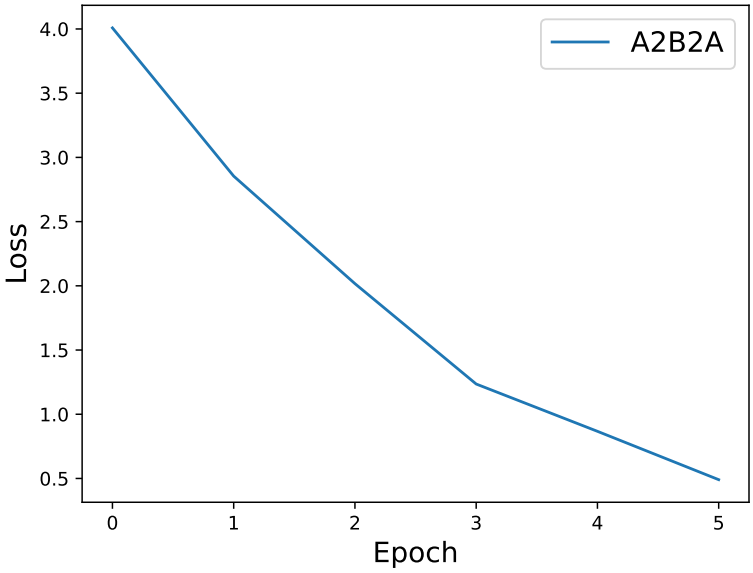

図 22　事前学習データ数 10000 における提案手法学習時の損失

4.3.2　対ではないデータを用いた事前学習ありの場合

学習データ 100 の場合

事前学習データ数が 100 の場合の実験結果を図 23、図 24 に、増減率を表 5 に示す。また、追加データの数が 100 の時のそれぞれの損失を図 25、図 26 に示す。Transformer および 100 個から 10000 個の追加データを用いた追加学習を行った提案モデルにおいても最小値はすべて 0.0 であった。最大値においては、追加データの数が 500 の場合においてのみ増加し、その他の追加データの数の場合において減少した。平均値においては、追加データの数に関わらず METEOR スコアは減少しており、追加データの数が 100 の場合において最も減少した結果となった。

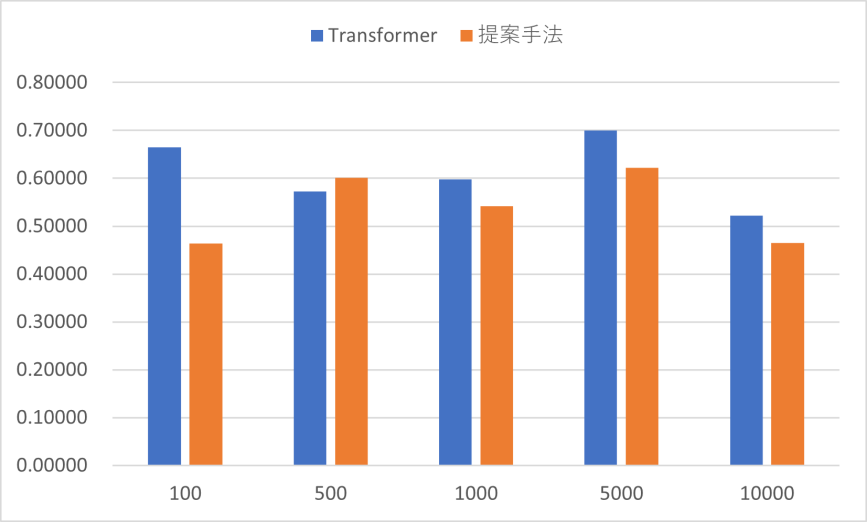

図 23　対ではない事前学習データ数 100 の最大値

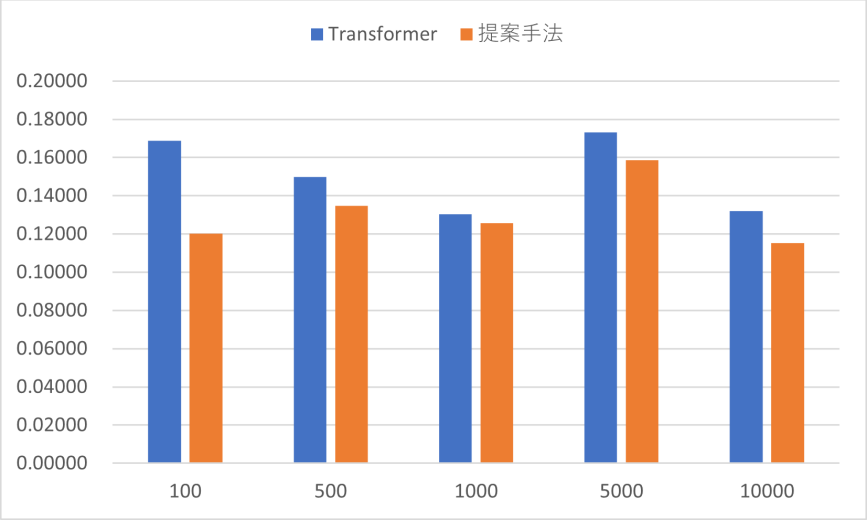

図 24　対ではない事前学習データ数 100 の平均値

表 5　対ではない事前学習データ数 100 の METEOR スコアの増減率

追加データ数	最大値 (%)	平均値 (%)
100	-30.27361	-28.74178
500	4.96404	-10.11115
1000	-9.38705	-3.71799
5000	-11.12020	-8.41694
10000	-10.99442	-12.54966

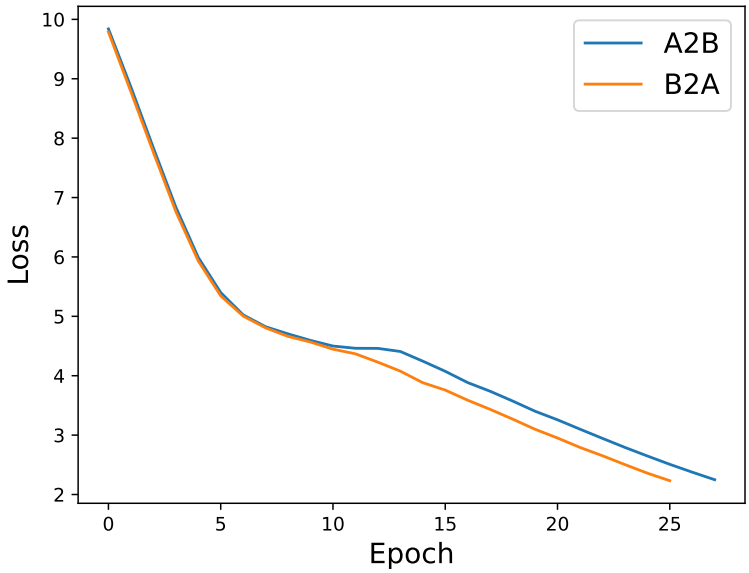

図 25　対ではない事前学習データ数 100 における事前学習時の損失

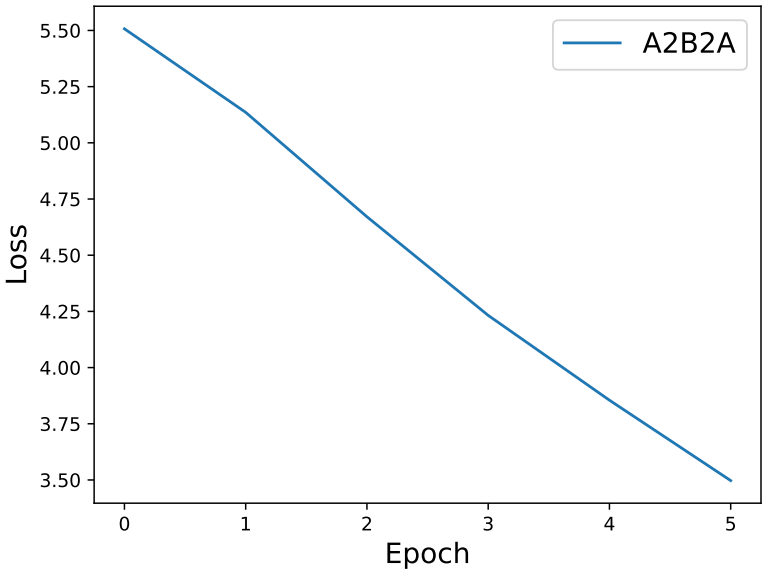

図 26　対ではない事前学習データ数 100 における提案手法学習時の損失

学習データ 1000 の場合

事前学習データ数が 100 の場合の実験結果を図 27、図 28 に、増減率を表 6 に示す。また、追加データの数が 100 の時のそれぞれの損失を図 29、図 30 に示す。Transformer および 100 個から 10000 個の追加データを用いた追加学習を行った提案モデルにおいても最小値はすべて 0.0 であった。最大値においては、追加データの数が 1000 および 10000 の場合において増加し、その他の追加データの数の場合において減少した。平均値においては、追加データの数に関わらず METEOR スコアは減少しており、追加データの数が 500 の場合において最も減少した結果となった。

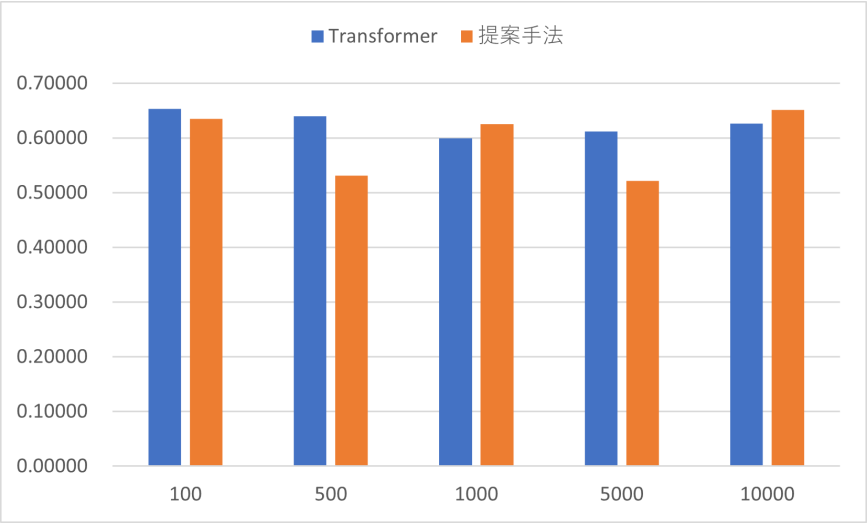

図 27　対ではない事前学習データ数 1000 の最大値

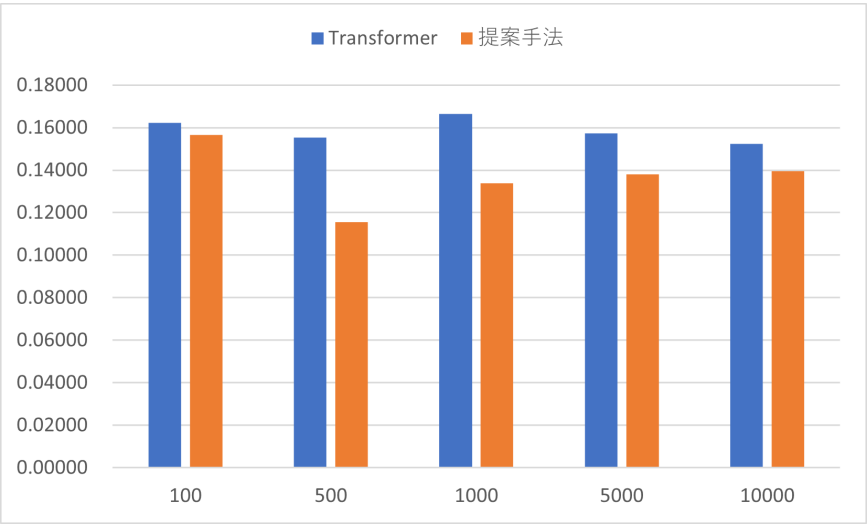

図 28　対ではない事前学習データ数 1000 の平均値

表 6　対ではない事前学習データ数 1000 の METEOR スコアの増減率

追加データ数	最大値 (%)	平均値 (%)
100	-2.75424	-3.48770
500	-16.97892	-25.53887
1000	4.45302	-19.54105
5000	-14.71569	-12.21760
10000	3.98438	-8.50013

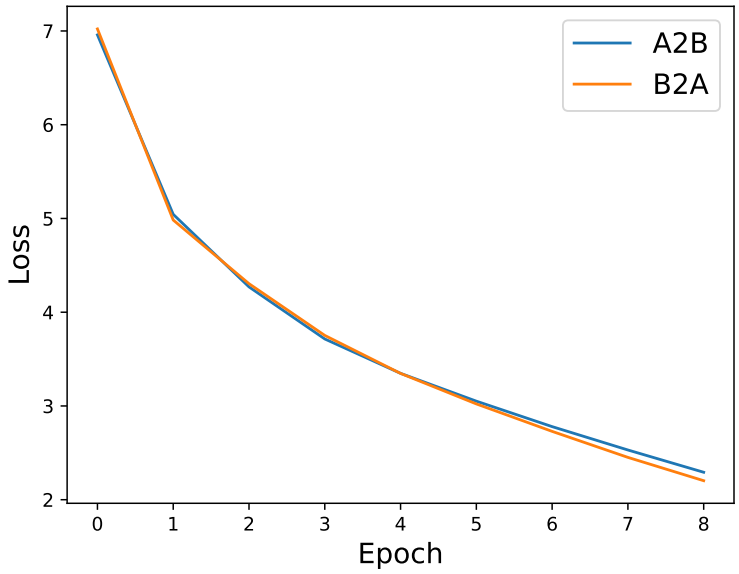

図 29　対ではない事前学習データ数 1000 における事前学習時の損失

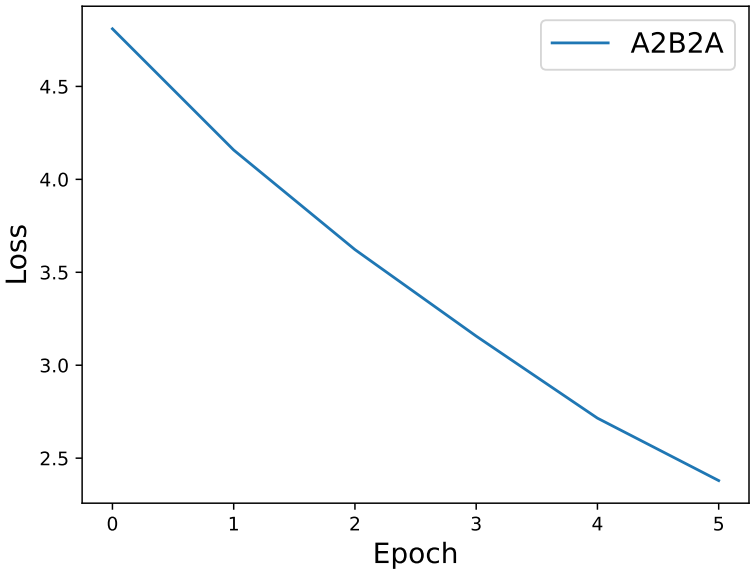

図 30　対ではない事前学習データ数 1000 における提案手法学習時の損失

学習データ 10000 の場合

事前学習データ数が 100 の場合の実験結果を図 31、図 32 に、増減率を表 7 に示す。また、追加データの数が 100 の時のそれぞれの損失を図 33、図 34 に示す。Transformer および 100 個から 10000 個の追加データを用いた追加学習を行った提案モデルにおいて、提案手法において 10000 個の追加データを用いた場合のみ最小値が 0.00575 であり、それ以外の場合はすべて 0.0 であった。最大値においては、追加データの数が 10000 の場合を除いて増加し、追加データの数が 10000 の場合において減少した。平均値においては、追加データの数に関わらず METEOR スコアは減少しており、追加データの数が 10000 の場合において最も減少した結果となった。

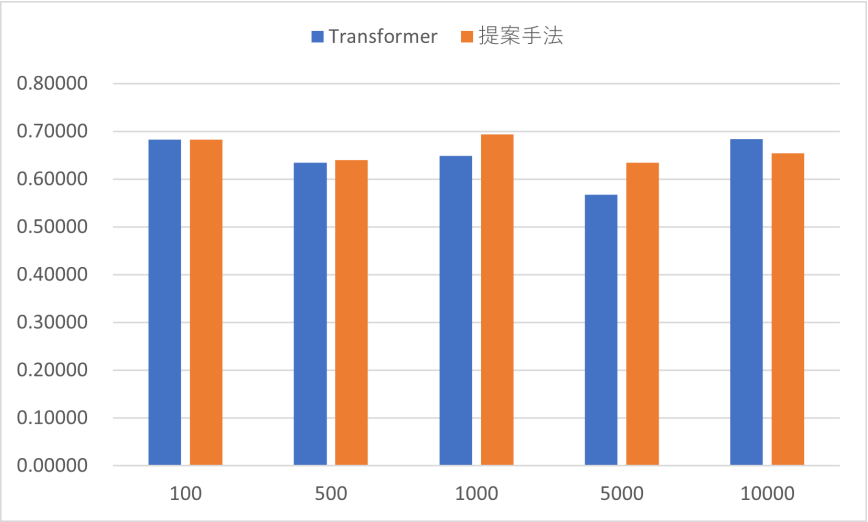

図 31　対ではない事前学習データ数 10000 の最大値

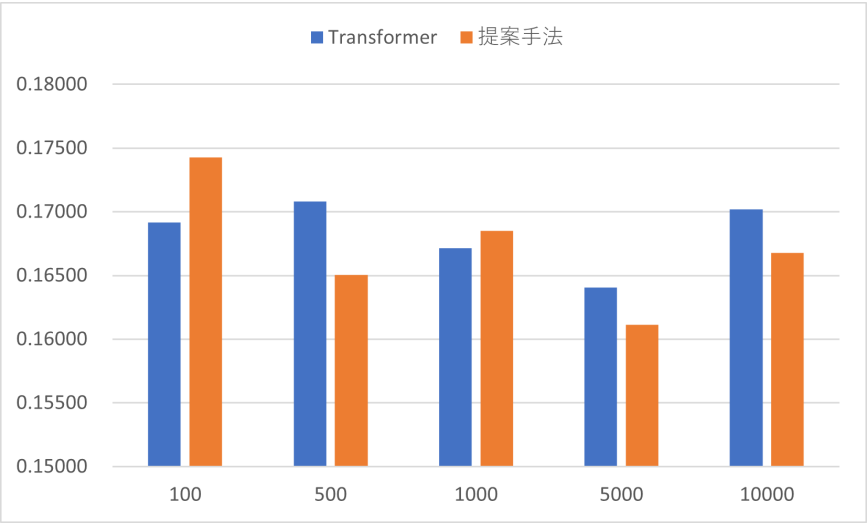

図 32　対ではない事前学習データ数 10000 の平均値

表 7　対ではない事前学習データ数 10000 の METEOR スコアの増減率

追加データ数	最大値 (%)	平均値 (%)
100	0.04886	-25.86258
500	0.92656	-31.12745
1000	6.88971	-15.22415
5000	11.89805	-15.72912
10000	-4.42880	-47.52262

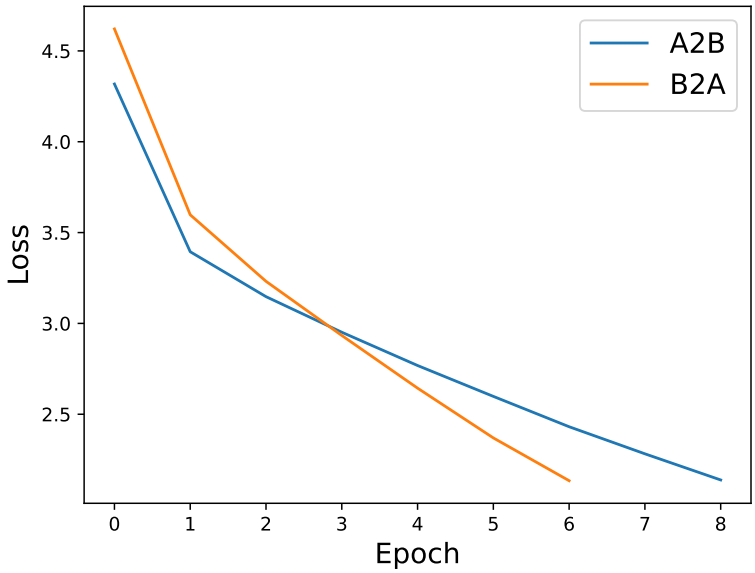

図 33　対ではない事前学習データ数 10000 における事前学習時の損失

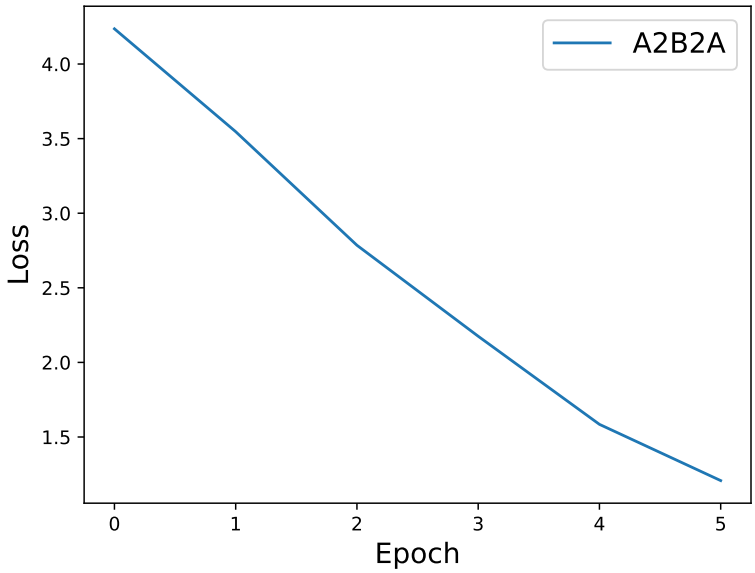

図 34　対ではない事前学習データ数 10000 における提案手法学習時の損失

4.3.3　事前学習なしの場合

事前学習なしの場合の実験結果を表 8 に示す。また、追加データの数が 100 の時の損失を図 35 に示す。最小値においては、学習データの数に関わらずすべて 0.0 であった。最大値および平均値においては、学習データの数が 1000 の場合において最大値、平均値ともに最も高い結果となり、学習データの数が 500 の場合において最大値、平均値ともに最も低い結果となった

表 8　事前学習なしの METEOR スコア

追加データ数	最小値	最大値	平均値
100	0.00000	0.04048	0.00015
500	0.00000	0.02067	0.00004
1000	0.00000	0.05396	0.00158

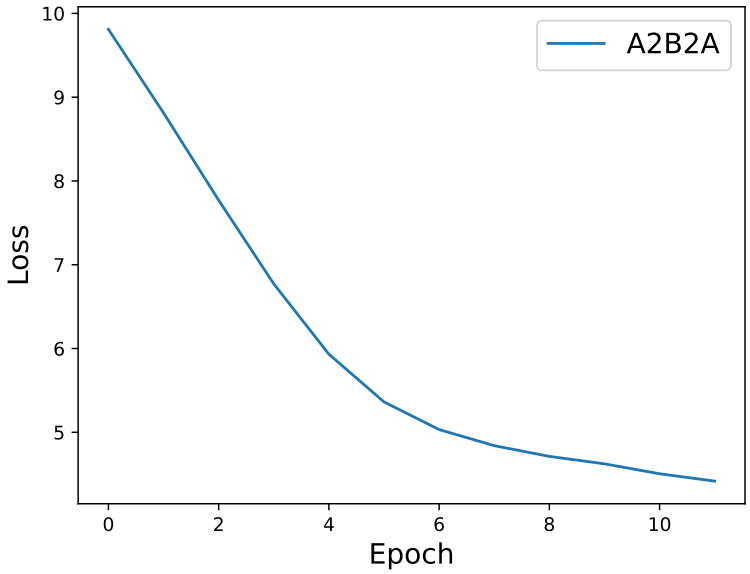

図 35　事前学習なしの場合の提案手法学習時の損失

4.4 考察

4.4.1 対データを用いた事前学習ありの場合

まず、対データを用いた事前学習ありの場合について考察する。基本的には、表 2、表 3、表 4 より追加データの数に関わらず最大値、平均値ともに METEOR スコアは減少することが分かった。しかし、追加データの数に関わらず、最小値のスコアが上昇した場合もあり、Transofomer の 3 倍以上のスコアに上昇した場合が存在した。そのため、対データを用いた事前学習ありの場合においては、追加データを用いて追加学習を行うことによって、Transformer が学習をしきれない部分の学習を行うことができると考えられる一方、その精度は微細なものであると考えられる。また、最小値のスコアが改善した場面において追加データのどの要素がスコアの改善に寄与したのかを調査することが今後の課題である。

4.4.2 対ではないデータを用いた事前学習ありの場合

次に、対ではないデータを用いた事前学習ありの場合について考察する。基本的には、表 5、表 6、表 7 より追加データの数に関わらず最大値、平均値ともに METEOR スコアは減少することが分かった。しかし、追加データの数に関わらず、最大値や平均値が上昇した場合も存在した。一方、上昇率は対データを用いた時と比較して低い結果となった。そのため、対データを用いて事前学習を行ったモデルに比べて、対ではないデータを用いて事前学習を行ったモデルは提案手法のモデルの恩恵が少ないと考えられる。また、最小値のスコアは事前学習データ数が 10000 の時の追加データ数が 10000 の場合においてのみ改善したため、対ではないデータを用いた事前学習ありの場合においてはより多くのデータが必要であると考えられる。その他にも、対データを用いた事前学習ありの場合と同様に、事前学習データ数や追加データ数が METEOR スコアと相関性があまりないことが分かったため、今後の課題として、その相関性を調査することが挙げられる。

4.4.3 事前学習なしの場合

最後に、事前学習なしの場合について考察する。表 8 より最大値、平均値ともにデータ数を増やすことでスコアが上昇する傾向が見られた。一方、今回の実験においてはデータ数が 500 の場合において最大値、平均値ともに最も低い結果となった。また、最小値においては、学習データの数に関わらずすべて 0.0 であった。データ数が 1000 の場合においての平均値が

最も高い結果となったが、このスコアは事前学習ありのスコアデータと比較すると低い結果となった。そのため、事前学習なしで学習を行う際は、Transformer や事前学習ありの場合と比較して、より多くのデータが必要であると考えられる。

5　結論

本研究では機械学習によって画像変換を行う手法である CycleGAN を応用し、文体変換を行う手法を検討した。また、文体変換の性能を評価するために、METEOR スコアを用いて評価を行った。その結果、対データを用いた事前学習ありの場合において、基本的に追加データの数に関わらず最大値、平均値ともに METEOR スコアは減少した。しかし、追加データの数に関わらず、スコアが上昇した場合もあった。また、対ではないデータを用いた事前学習ありの場合においても、基本的に追加データの数に関わらず最大値、平均値ともに METEOR スコアは減少した。しかし、追加データの数に関わらず、スコアが上昇した場合もあった。さらに、事前学習なしの場合においては追加データの数が増えるにつれて、最大値、平均値ともに METEOR スコアが上昇する傾向が見られた。

今後の展望としては、提案手法のモデルの損失関数の改善や、他の評価指標を用いた評価を行うことが挙げられる。また、提案手法のモデルの構造の改善や、他のデータセットを用いた実験を行うことも考えられる。

謝辞

本論文の作成にあたり、多くの助言、指導をしてくださった三好 力 教授に心からお礼申し上げます。また、議論に協力してくださった三好研究室の皆様や学友の皆様に心から感謝いたします。

参考文献

[1] 林由紀子, 松原茂樹. ニュース記事の自然な音声出力のためのテキスト変換. 言語処理学会第 14 回年次大会発表論文集, pp. 790–793, 2008.

[2] 下地健太, 森田和宏, 泓田正雄. Lstm-rnn を用いた文体変換手法. 人工知能学会全国大会論文集 第 33 回 (2019), pp. 2L4J901–2L4J901. 一般社団法人 人工知能学会, 2019.

[3] 機械翻訳とは？ — phrase. https://phrase.com/ja/blog/posts/machine-translation/. (Accessed on 02/07/2024).

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, Vol. 27, , 2014.

[5] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

[6] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, Vol. 27, , 2014.

[7] 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*, Vol. 30, , 2017.

[8] 交差エントロピー誤差をわかりやすく説明してみる #機械学習 - qiita. https://qiita.com/kenta1984/items/59a9ef1788e6934fd962. (Accessed on 02/08/2024).

[9] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

[10] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pp. 65–72, 2005.

[11] Takumi Maruyama and Kazuhide Yamamoto. Simplified corpus with core vocabulary. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

A 付録

```
  # -*- coding: utf-8 -*-

import math
import time
import os
import pandas as pd
import numpy as np
from nltk.translate.meteor_score import meteor_score
import MeCab
from sklearn.model_selection import train_test_split
from collections import Counter
from tqdm import tqdm
import torch
import torch.nn as nn
from torch import Tensor
from torch.nn import (
    TransformerEncoder, TransformerDecoder,
    TransformerEncoderLayer, TransformerDecoderLayer
)
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import DataLoader
from torchtext.vocab import vocab
import os
import matplotlib.pyplot as plt定数の設定

"""""""
# の設定 Mecab
t = MeCab.Tagger("-Owakati")

# データ数の設定
# 事前学習用データ数
PRE_TRAIN_DATA_NUM = 10000 # 事前学習用データ数

# 用データ数 A2B2A
A2B2A_TRAIN_DATA_NUM = 100 # 用学習データ数 A2B2A

# 評価・テストデータ数
VALID_DATA_NUM = 1000 # 評価データ数
```

```
TEST_DATA_NUM = 1000 # テストデータ数

PATIENCE = 5 # 早期終了のパラメータ

# 損失リストの初期化
loss_A2B = []
loss_B2A = []
loss_A2B2A = []


"""の設定 CUDA"""
# デバイスの設定
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f実行環境："{device}")データの読み込み


"""""""
# 学習に使うデータの取得
path = os.path.join("data", "2020.1.7.csv")
df = pd.read_csv(path, header=0, encoding="utf-8")

# 単語辞書を作成する関数を修正
def build_vocab(texts):
    counter = Counter()
    for text in texts:
        counter.update(t.parse(text).split())
    return vocab(counter, specials=['<unk>', '<pad>', '<start>', '<end>'])

# 全テキストデータを用いて単語辞書を作成
all_texts = pd.concat([df日本語原文 ["#()"], dfやさしい日本語 ["#"]])
vocab_all = build_vocab(all_texts)データの前処理


"""""""
# 学習データを分割する関数
def split_data(data_src, data_tgt, train_size=0.7, valid_size=0.15, test_size
    =0.15):
    # 訓練データと残りのデータに最初に分割
    texts_src_train, remaining_src_data, texts_tgt_train, remaining_tgt_data =
        train_test_split(data_src, data_tgt, train_size=train_size)
```

```
77
78      # 残りのデータを訓練用と検証用に分割
79      texts_src_valid, texts_src_test, texts_tgt_valid, texts_tgt_test =
            train_test_split(remaining_src_data, remaining_tgt_data, train_size=
            valid_size, test_size=test_size)
80
81      return texts_src_train, texts_src_valid, texts_src_test, texts_tgt_train,
            texts_tgt_valid, texts_tgt_test
82
83  # データを分割
84  texts_src_train, texts_src_valid, texts_src_test, texts_tgt_train,
        texts_tgt_valid, texts_tgt_test = split_data(df日本語原文["#()"], dfやさし
        い日本語["#"], train_size=sum([PRE_TRAIN_DATA_NUM, A2B2A_TRAIN_DATA_NUM]),
        valid_size=VALID_DATA_NUM, test_size=TEST_DATA_NUM)
85
86  # 用に学習データをさらに分割A2B2A
87  texts_src_train, texts_src_A2B2A, texts_tgt_train, texts_tgt_A2B2A =
        train_test_split(texts_src_train, texts_tgt_train, train_size=
        PRE_TRAIN_DATA_NUM)
88  # 用に学習データと評価データに分割A2B2A
89  #texts_src_train_A2B2A, texts_src_valid_A2B2A, texts_tgt_train_A2B2A,
        texts_tgt_valid_A2B2A = train_test_split(texts_src_A2B2A, texts_tgt_A2B2A,
        train_size=A2B2A_TRAIN_DATA_NUM, test_size=VALID_DATA_NUM)
90
91  # データ数の確認
92  print(f事前学習用学習データ数："{len(texts_src_train)}")
93  print(f追加学習用学習データ数："{len(texts_src_A2B2A)}")
94  print(f評価データ数："{len(texts_src_valid)}")
95  print(fテストデータ数："{len(texts_src_test)}")
96
97  def convert_text_to_indexes(text, vocab):
98      return [vocab['<start>']] + [vocab[token] if token in vocab else vocab['<
            unk>'] for token in t.parse(text.strip("/n")).split()] + [vocab['<end
            >']]
99
100 ## テキストをインデックス化し、バッチサイズごとにベクトル化
101 def data_process(texts_src, texts_tgt, vocab):
102
103     data = []
104     for (src, tgt) in zip(texts_src, texts_tgt):
```

```
105         src_tensor = torch.tensor(
106             convert_text_to_indexes(text=src, vocab=vocab),
107             dtype=torch.long
108         )
109         tgt_tensor = torch.tensor(
110             convert_text_to_indexes(text=tgt, vocab=vocab),
111             dtype=torch.long
112         )
113         data.append((src_tensor, tgt_tensor))
114
115     return data
116
117 # データをインデックス化
118 # 学習用
119 ## ⇒用 AB
120 train_data_A2B = data_process(texts_src=texts_src_train, texts_tgt=
        texts_tgt_train, vocab=vocab_all)
121 # ⇒用 BA
122 train_data_B2A = data_process(texts_src=texts_tgt_train, texts_tgt=
        texts_src_train, vocab=vocab_all)
123 # ⇒⇒用 ABA
124 train_data = data_process(texts_src=texts_src_A2B2A, texts_tgt=texts_tgt_A2B2A,
        vocab=vocab_all)
125
126 # 評価用
127 ## ⇒・⇒⇒用 ABABA
128 valid_data_A2B = data_process(texts_src=texts_src_valid, texts_tgt=
        texts_tgt_valid, vocab=vocab_all)
129 # ⇒用 BA
130 valid_data_B2A = data_process(texts_src=texts_tgt_valid, texts_tgt=
        texts_src_valid, vocab=vocab_all)
131
132
133 #
134 def translate(model, text, vocab_src, vocab_tgt, seq_len_tgt, START_IDX, END_IDX
        ):
135     model.eval()
136     tokens = convert_text_to_indexes(text=text, vocab=vocab_src)
137     num_tokens = len(tokens)
138     src = torch.LongTensor(tokens).reshape(num_tokens, 1)
```

```
139     mask_src = (torch.zeros(num_tokens, num_tokens)).type(torch.bool)
140
141     predicts = greedy_decode(
142         model=model, src=src,
143         mask_src=mask_src, seq_len_tgt=seq_len_tgt,
144         START_IDX=START_IDX, END_IDX=END_IDX
145     ).flatten()
146
147     return ' '.join([vocab_tgt.get_itos()[token] for token in predicts]).replace
            ("<start>", "").replace("<end>", "")
148
149
150 def greedy_decode(model, src, mask_src, seq_len_tgt, START_IDX, END_IDX):
151
152     src = src.to(device)
153     mask_src = mask_src.to(device)
154
155     memory = model.encode(src, mask_src)
156     memory = model.transformer_encoder(model.positional_encoding(model.
            token_embedding_src(src)), mask_src)
157     ys = torch.ones(1, 1).fill_(START_IDX).type(torch.long).to(device)
158
159     for i in range(seq_len_tgt - 1):
160
161         memory = memory.to(device)
162         memory_mask = torch.zeros(ys.shape[0], memory.shape[0]).to(device).type(
                torch.bool)
163         mask_tgt = (generate_square_subsequent_mask(ys.size(0), PAD_IDX).type(
                torch.bool)).to(device)
164
165         output = model.decode(ys, memory, mask_tgt)
166         output = output.transpose(0, 1)
167         output = model.output(output[:, -1])
168         _, next_word = torch.max(output, dim = 1)
169         next_word = next_word.item()
170
171         ys = torch.cat([ys, torch.ones(1, 1).type_as(src.data).fill_(next_word)
                ], dim=0)
172         if next_word == END_IDX:
173             break
```

```
174
175     return ys
176
177 seq_len_tgt = max([len(x[1]) for x in train_data])
178
179 # を使用して形態素解析を行う関数 MeCab
180 def tokenize_japanese(text):
181     return t.parse(text).strip().split()
182
183 def calculate_meteor_scores(model, texts_src, texts_tgt, vocab_src, vocab_tgt,
        seq_len_tgt, START_IDX, END_IDX):
184     scores = []
185     for src, tgt in zip(texts_src, texts_tgt):
186         translated = translate(
187             model=model, text=src, vocab_src=vocab_src, vocab_tgt=vocab_tgt,
188             seq_len_tgt=seq_len_tgt, START_IDX=START_IDX, END_IDX=END_IDX
189         )
190         hypothesis = tokenize_japanese(translated)
191         reference = tokenize_japanese(tgt)
192         score = meteor_score([reference], hypothesis)
193         scores.append(score)
194     return np.array(scores)
195
196 batch_size = 32
197 PAD_IDX = vocab_all['<pad>']
198 START_IDX = vocab_all['<start>']
199 END_IDX = vocab_all['<end>']
200
201 def generate_batch(data_batch):
202
203     batch_src, batch_tgt = [], []
204     for src, tgt in data_batch:
205         batch_src.append(src)
206         batch_tgt.append(tgt)
207
208     batch_src = pad_sequence(batch_src, padding_value=PAD_IDX)
209     batch_tgt = pad_sequence(batch_tgt, padding_value=PAD_IDX)
210
211     return batch_src, batch_tgt
212
```

```
# データローダーの作成
## 学習用
train_iter_A2B = DataLoader(train_data_A2B, batch_size=batch_size, shuffle=True,
     collate_fn=generate_batch, drop_last=True, pin_memory=True)
train_iter_B2A = DataLoader(train_data_B2A, batch_size=batch_size, shuffle=True,
     collate_fn=generate_batch, drop_last=True, pin_memory=True)
train_iter = DataLoader(train_data, batch_size=batch_size, shuffle=True,
    collate_fn=generate_batch, drop_last=True, pin_memory=True)

## 評価用
valid_iter_A2B = DataLoader(valid_data_A2B, batch_size=batch_size, shuffle=True,
     collate_fn=generate_batch, drop_last=True, pin_memory=True)
valid_iter_B2A = DataLoader(valid_data_B2A, batch_size=batch_size, shuffle=True,
     collate_fn=generate_batch, drop_last=True, pin_memory=True)

"Transformer を使った翻訳モデルの設計"
# Embedding とPositionalEncoding のクラス
class TokenEmbedding(nn.Module):

    def __init__(self, vocab_size, embedding_size):

        super(TokenEmbedding, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.embedding_size = embedding_size

    def forward(self, tokens: Tensor):
        return self.embedding(tokens.long()) * math.sqrt(self.embedding_size)

class PositionalEncoding(nn.Module):

    def __init__(self, embedding_size: int, dropout: float, maxlen: int = 5000):
        super(PositionalEncoding, self).__init__()

        den = torch.exp(-torch.arange(0, embedding_size, 2) * math.log(10000) /
             embedding_size)
        pos = torch.arange(0, maxlen).reshape(maxlen, 1)
        embedding_pos = torch.zeros((maxlen, embedding_size))
        embedding_pos[:, 0::2] = torch.sin(pos * den)
        embedding_pos[:, 1::2] = torch.cos(pos * den)
        embedding_pos = embedding_pos.unsqueeze(-2)
```

```
247
248         self.dropout = nn.Dropout(dropout)
249         self.register_buffer('embedding_pos', embedding_pos)
250
251     def forward(self, token_embedding: Tensor):
252         return self.dropout(token_embedding + self.embedding_pos[:
                token_embedding.size(0), :])
253
254 # 翻訳モデル全体のクラス
255 class Seq2SeqTransformer(nn.Module):
256
257     def __init__(
258         self, num_encoder_layers: int, num_decoder_layers: int,
259         embedding_size: int, vocab_size_src: int, vocab_size_tgt: int,
260         dim_feedforward:int = 512, dropout:float = 0.1, nhead:int = 8
261     ):
262
263         super(Seq2SeqTransformer, self).__init__()
264
265         self.token_embedding_src = TokenEmbedding(vocab_size_src, embedding_size)
266         self.positional_encoding = PositionalEncoding(embedding_size, dropout=
                dropout)
267         encoder_layer = TransformerEncoderLayer(
268             d_model=embedding_size, nhead=nhead, dim_feedforward=dim_feedforward
269         )
270         self.transformer_encoder = TransformerEncoder(encoder_layer, num_layers=
                num_encoder_layers)
271
272         self.token_embedding_tgt = TokenEmbedding(vocab_size_tgt, embedding_size)
273         decoder_layer = TransformerDecoderLayer(
274             d_model=embedding_size, nhead=nhead, dim_feedforward=dim_feedforward
275         )
276         self.transformer_decoder = TransformerDecoder(decoder_layer, num_layers=
                num_decoder_layers)
277
278         self.output = nn.Linear(embedding_size, vocab_size_tgt)
279
280     def forward(
281         self, src: Tensor, tgt: Tensor,
282         mask_src: Tensor, mask_tgt: Tensor,
```

```
283         padding_mask_src: Tensor, padding_mask_tgt: Tensor,
284         memory_key_padding_mask: Tensor
285     ):
286 
287         embedding_src = self.positional_encoding(self.token_embedding_src(src))
288         memory = self.transformer_encoder(embedding_src, mask_src,
                padding_mask_src)
289         embedding_tgt = self.positional_encoding(self.token_embedding_tgt(tgt))
290         outs = self.transformer_decoder(
291             embedding_tgt, memory, mask_tgt, None,
292             padding_mask_tgt, memory_key_padding_mask
293         )
294         return self.output(outs)
295 
296     def encode(self, src: Tensor, mask_src: Tensor):
297         return self.transformer_encoder(self.positional_encoding(self.
                token_embedding_src(src)), mask_src)
298 
299     def decode(self, tgt: Tensor, memory: Tensor, mask_tgt: Tensor):
300         return self.transformer_decoder(self.positional_encoding(self.
                token_embedding_tgt(tgt)), memory, mask_tgt)
301 
302 # マスキング
303 def create_mask(src, tgt, PAD_IDX):
304 
305     seq_len_src = src.shape[0]
306     seq_len_tgt = tgt.shape[0]
307 
308     mask_tgt = generate_square_subsequent_mask(seq_len_tgt, PAD_IDX)
309     mask_src = torch.zeros((seq_len_src, seq_len_src), device=device).type(torch
            .bool)
310 
311     padding_mask_src = (src == PAD_IDX).transpose(0, 1)
312     padding_mask_tgt = (tgt == PAD_IDX).transpose(0, 1)
313 
314     return mask_src, mask_tgt, padding_mask_src, padding_mask_tgt
315 
316 
317 def generate_square_subsequent_mask(seq_len, PAD_IDX):
```

```
318     mask = (torch.triu(torch.ones((seq_len, seq_len), device=device)) == 1).
            transpose(0, 1)
319     mask = mask.float().masked_fill(mask == 0, float('-inf')).masked_fill(mask
            == PAD_IDX, float(0.0))
320     return mask
321
322 "Transformer を使った翻訳モデルの学習"
323 # モデルの学習と評価の関数の定義
324 def train(model, data, optimizer, criterion, PAD_IDX):
325
326     model.train()
327     losses = 0
328     for src, tgt in tqdm(data):
329
330         src = src.to(device)
331         tgt = tgt.to(device)
332
333         input_tgt = tgt[:-1, :]
334
335         mask_src, mask_tgt, padding_mask_src, padding_mask_tgt = create_mask(src,
                input_tgt, PAD_IDX)
336
337         logits = model(
338             src=src, tgt=input_tgt,
339             mask_src=mask_src, mask_tgt=mask_tgt,
340             padding_mask_src=padding_mask_src, padding_mask_tgt=padding_mask_tgt,
341             memory_key_padding_mask=padding_mask_src
342         )
343
344         optimizer.zero_grad()
345
346         output_tgt = tgt[1:, :]
347         loss = criterion(logits.reshape(-1, logits.shape[-1]), output_tgt.
               reshape(-1))
348         loss.backward()
349
350         optimizer.step()
351         losses += loss.item()
352
353     return losses / len(data)
```

```
354
355 def evaluate(model, data, criterion, PAD_IDX):
356
357     model.eval()
358     losses = 0
359     for src, tgt in data:
360
361         src = src.to(device)
362         tgt = tgt.to(device)
363
364         input_tgt = tgt[:-1, :]
365
366         mask_src, mask_tgt, padding_mask_src, padding_mask_tgt = create_mask(src,
                input_tgt, PAD_IDX)
367
368         logits = model(
369             src=src, tgt=input_tgt,
370             mask_src=mask_src, mask_tgt=mask_tgt,
371             padding_mask_src=padding_mask_src, padding_mask_tgt=padding_mask_tgt,
372             memory_key_padding_mask=padding_mask_src
373         )
374
375         output_tgt = tgt[1:, :]
376         loss = criterion(logits.reshape(-1, logits.shape[-1]), output_tgt.
               reshape(-1))
377         losses += loss.item()
378
379     return losses / len(data)
380
381 # モデルのインスタンスの作成
382 vocab_size = len(vocab_all)
383 embedding_size = 240
384 nhead = 8
385 dim_feedforward = 100
386 num_encoder_layers = 2
387 num_decoder_layers = 2
388 dropout = 0.1
389
390 # モデルの定義
391 model_A2B = Seq2SeqTransformer(
```

```
    num_encoder_layers=num_encoder_layers,
    num_decoder_layers=num_decoder_layers,
    embedding_size=embedding_size,
    vocab_size_src=vocab_size, vocab_size_tgt=vocab_size,
    dim_feedforward=dim_feedforward,
    dropout=dropout, nhead=nhead
)
model_B2A = Seq2SeqTransformer(
    num_encoder_layers=num_encoder_layers,
    num_decoder_layers=num_decoder_layers,
    embedding_size=embedding_size,
    vocab_size_src=vocab_size, vocab_size_tgt=vocab_size,
    dim_feedforward=dim_feedforward,
    dropout=dropout, nhead=nhead
)

# 重みの初期化
for p in model_A2B.parameters():
    if p.dim() > 1:
        nn.init.xavier_uniform_(p)

for p in model_B2A.parameters():
    if p.dim() > 1:
        nn.init.xavier_uniform_(p)

# 損失関数の定義
criterion = torch.nn.CrossEntropyLoss(ignore_index=PAD_IDX)

# オプティマイザの定義
optimizer_A2B = torch.optim.Adam(model_A2B.parameters())
optimizer_B2A = torch.optim.Adam(model_B2A.parameters())

# 初期学習：A →B
num_epochs_A2B = 100 ## エポック数
best_loss = float('Inf')
best_model_A2B = None
patience = PATIENCE
counter = 0

model_A2B.to(device=device)
```

```
432
433 for loop in range(1, num_epochs_A2B+1):
434
435     star_time = time.time() ## 計測開始
436
437     loss_train = train(model=model_A2B, data=train_iter_A2B, optimizer=
        optimizer_A2B, criterion=criterion, PAD_IDX=PAD_IDX)
438
439     loss_A2B.append(loss_train)
440
441     elapsed_time = time.time() - star_time ## 計測終了
442
443     loss_valid = evaluate(model=model_A2B, data=valid_iter_A2B, criterion=
        criterion, PAD_IDX=PAD_IDX)
444
445     print('[{}/{}] train loss: {:.2f}, valid loss: {:.2f} [{}{:.0f}s] count:
        {}, {}'.format(
446         loop, num_epochs_A2B,
447         loss_train, loss_valid,
448         str(int(math.floor(elapsed_time / 60))) + 'm' if math.floor(elapsed_time
            / 60) > 0 else '',
449         elapsed_time % 60,
450         counter,
451         '**' if best_loss > loss_valid else ''
452     ))
453
454     if best_loss > loss_valid:
455         best_loss = loss_valid
456         best_model_A2B = model_A2B
457         counter = 0
458
459     if counter >= patience:
460         break
461
462     counter += 1
463
464 # 翻訳の実行
465 # テストデータセットでスコアを計算 METEOR
466 meteor_scores = calculate_meteor_scores(
467     model=best_model_A2B, texts_src=texts_src_test, texts_tgt=texts_tgt_test,
```

```
468     vocab_src=vocab_all, vocab_tgt=vocab_all, seq_len_tgt=seq_len_tgt,
469     START_IDX=START_IDX, END_IDX=END_IDX
470 )
471
472 # スコアの平均、最大値、最小値を計算
473 average_meteor_score = np.mean(meteor_scores)
474 max_meteor_score = np.max(meteor_scores)
475 min_meteor_score = np.min(meteor_scores)
476 std_meteor_score = np.std(meteor_scores)
477
478 # 結果を出力
479 print(f"平均 METEOR Score: {average_meteor_score}")
480 print(f"最大 METEOR Score: {max_meteor_score}")
481 print(f"最小 METEOR Score: {min_meteor_score}")
482 print(f"標準偏差 METEOR Score: {std_meteor_score}")
483
484 # 初期学習：B →A
485 num_epochs_B2A = 100 ## エポック数
486 best_loss = float('Inf')
487 best_model_B2A = None
488 patience = PATIENCE
489 counter = 0
490
491 model_B2A.to(device=device)
492
493 for loop in range(1, num_epochs_B2A+1):
494
495     star_time = time.time() ## 計測開始
496
497     loss_train = train(model=model_B2A, data=train_iter_B2A, optimizer=
        optimizer_B2A, criterion=criterion, PAD_IDX=PAD_IDX)
498
499     loss_B2A.append(loss_train)
500
501     elapsed_time = time.time() - star_time ## 計測終了
502
503     loss_valid = evaluate(model=model_B2A, data=valid_iter_B2A, criterion=
        criterion, PAD_IDX=PAD_IDX)
504
```

```
505     print('[{}/{}] train loss: {:.2f}, valid loss: {:.2f} [{}{:.0f}s] count:
            {}, {}'.format(
506         loop, num_epochs_B2A,
507         loss_train, loss_valid,
508         str(int(math.floor(elapsed_time / 60))) + 'm' if math.floor(elapsed_time
                / 60) > 0 else '',
509         elapsed_time % 60,
510         counter,
511         '**' if best_loss > loss_valid else ''
512     ))
513
514     if best_loss > loss_valid:
515         best_loss = loss_valid
516         best_model_B2A = model_B2A
517         counter = 0
518
519     if counter >= patience:
520         break
521
522     counter += 1
523
524
525 # 提案学習：⇒⇒ ABA
526 class A2B2ATransformer(nn.Module):
527     def __init__(self, model_A2B, model_B2A, seq_len_tgt, START_IDX, END_IDX):
528         super(A2B2ATransformer, self).__init__()
529         self.model_A2B = model_A2B
530         self.model_B2A = model_B2A
531         self.seq_len_tgt = seq_len_tgt
532         self.START_IDX = START_IDX
533         self.END_IDX = END_IDX
534
535     def translate(self, text_batch, model, vocab_src, vocab_tgt, seq_len_tgt,
            START_IDX, END_IDX):
536         text_batch = text_batch.unsqueeze(1).to(device)
537         # ソースマスクを生成
538         mask_src = (torch.zeros(text_batch.size(0), text_batch.size(0))).type(
                torch.bool).to(device)
539         # デコード関数をバッチに適用
```

```
540         predicts_batch = self.greedy_decode(model=model, src=text_batch, mask_src
                =mask_src, seq_len_tgt=seq_len_tgt, START_IDX=START_IDX, END_IDX=
                END_IDX).flatten()
541         #predicts = self.greedy_decode(model=model, src=src, mask_src=mask_src,
                seq_len_tgt=seq_len_tgt, START_IDX=START_IDX, END_IDX=END_IDX).
                flatten()
542         return predicts_batch
543
544
545     def greedy_decode(self, model, src, mask_src, seq_len_tgt, START_IDX,
            END_IDX):
546
547         src = src.to(device)
548         mask_src = mask_src.to(device)
549
550         memory = model.encode(src, mask_src)
551         memory = model.transformer_encoder(model.positional_encoding(model.
                token_embedding_src(src)), mask_src)
552         ys = torch.ones(1, 1).fill_(START_IDX).type(torch.long).to(device)
553
554         for i in range(seq_len_tgt - 1):
555             memory = memory.to(device)
556             memory_mask = torch.zeros(ys.shape[0], memory.shape[0]).to(device).
                    type(torch.bool)
557             mask_tgt = (generate_square_subsequent_mask(ys.size(0), PAD_IDX).type
                    (torch.bool)).to(device)
558
559             output = model.decode(ys, memory, mask_tgt)
560             output = output.transpose(0, 1)
561             output = model.output(output[:, -1])
562             _, next_word = torch.max(output, dim = 1)
563             next_word = next_word.item()
564
565             ys = torch.cat([ys, torch.ones(1, 1).type_as(src.data).fill_(
                    next_word)], dim=0)
566             if next_word == END_IDX:
567                 break
568
569         return ys
570
```

```
571     def forward(self, srcs):
572         input_tgt = srcs[:-1, :].to(device)
573 
574         # →AB
575         predicted_indices = []
576         srcs = torch.transpose(srcs, 0, 1)
577 
578         predicted_indices = [self.translate(model=self.model_A2B, text_batch=src,
                vocab_src=vocab_all, vocab_tgt=vocab_all, seq_len_tgt=self.
               seq_len_tgt, START_IDX=self.START_IDX, END_IDX=self.END_IDX) for src
               in srcs]
579 
580         # →BA
581         predicted_indices = pad_sequence(predicted_indices, batch_first=True,
               padding_value=PAD_IDX).transpose(0, 1)
582         mask_src, mask_tgt, padding_mask_src, padding_mask_tgt = create_mask(
               predicted_indices, input_tgt, PAD_IDX)
583         output = self.model_B2A(predicted_indices, input_tgt, mask_src, mask_tgt,
                padding_mask_src, padding_mask_tgt, padding_mask_src)
584 
585         return output
586 
587 # モデルのインスタンス作成
588 model_A2B2A = A2B2ATransformer(best_model_A2B, best_model_B2A, 100, START_IDX,
        END_IDX).to(device)
589 
590 # オプティマイザーとロス関数の定義
591 #optimizer_A2B2A = torch.optim.Adam(model_A2B2A.parameters())
592 criterion_A2B2A = torch.nn.CrossEntropyLoss(ignore_index=PAD_IDX)
593 
594 # 学習ループ
595 num_epochs_A2B2A = 100 # エポック数を設定
596 best_loss = float("Inf")
597 best_model = None
598 patience = PATIENCE
599 counter = 0
600 
601 printここから学習("")
602 
603 for epoch in range(1, num_epochs_A2B2A+1):
```

```
604
605     # 学習モード
606     model_A2B2A.train()
607     total_loss_train = 0
608
609     for src, tgt in tqdm(train_iter):
610 # optimizer_A2B2A.zero_grad()
611         optimizer_A2B.zero_grad()
612         optimizer_B2A.zero_grad()
613
614         src, tgt = src.to(device), tgt.to(device)
615
616         # →→変換 ABA
617         output = model_A2B2A(src)
618
619         # 損失関数の計算
620         loss = criterion_A2B2A(output.reshape(-1, output.shape[-1]), src[1:,
                :].reshape(-1))
621         total_loss_train += loss.item()
622
623         # 誤差逆伝播法
624         loss.backward()
625
626         # パラメータの更新
627 # optimizer_A2B2A.step()
628         optimizer_A2B.step()
629         optimizer_B2A.step()
630
631     # 評価モード
632     model_A2B2A.eval()
633     total_loss_valid = 0
634
635     for src, tgt in tqdm(valid_iter_A2B):
636         src, tgt = src.to(device), tgt.to(device)
637
638         # →変換 AB
639         input_tgt = tgt[:-1, :]
640         mask_src, mask_tgt, padding_mask_src, padding_mask_tgt = create_mask(src,
                input_tgt, PAD_IDX)
```

```
641         output = model_A2B2A.model_A2B(src, input_tgt, mask_src, mask_tgt,
                padding_mask_src, padding_mask_tgt, padding_mask_src)
642         output_tgt = tgt[1:, :]
643         loss = criterion_A2B2A(output.reshape(-1, output.shape[-1]), output_tgt.
                reshape(-1))
644         total_loss_valid += loss.item()
645
646     loss_train = total_loss_train / len(train_iter)
647     loss_valid = total_loss_valid / len(valid_iter_A2B)
648
649     loss_A2B2A.append(loss_train)
650
651     # 損失関数の表示
652     print(f"Epoch {epoch}: train loss {loss_train:.2f}, valid loss {loss_valid
            :.2f}, counter {counter}")
653
654     if best_loss > total_loss_valid:
655         best_loss = total_loss_valid
656         best_model = model_A2B2A
657         counter = 0
658
659     if counter >= patience:
660         break
661
662     counter += 1
663
664 # 翻訳の実行
665 # テストデータセットでスコアを計算 METEOR
666 meteor_scores = calculate_meteor_scores(
667     model=best_model.model_A2B, texts_src=texts_src_test, texts_tgt=
            texts_tgt_test,
668     vocab_src=vocab_all, vocab_tgt=vocab_all, seq_len_tgt=seq_len_tgt,
669     START_IDX=START_IDX, END_IDX=END_IDX
670 )
671
672 # スコアの平均、最大値、最小値を計算
673 average_meteor_score = np.mean(meteor_scores)
674 max_meteor_score = np.max(meteor_scores)
675 min_meteor_score = np.min(meteor_scores)
676 std_meteor_score = np.std(meteor_scores)
```

```
677
678 # 結果を出力
679 print(f"平均 METEOR Score: {average_meteor_score}")
680 print(f"最大 METEOR Score: {max_meteor_score}")
681 print(f"最小 METEOR Score: {min_meteor_score}")
682 print(f"標準偏差 METEOR Score: {std_meteor_score}")
683
684 # 損失関数のグラフを描画
685 plt.figure()
686 plt.plot(loss_A2B, label="A2B")
687 plt.plot(loss_B2A, label="B2A")
688 plt.legend(fontsize=15)
689 plt.xlabel("Epoch", fontsize=15)
690 plt.ylabel("Loss", fontsize=15)
691 plt.savefig(f"D:/SynologyDrive/SynologyDrive/2022/SemiReporot/shuron/
      template2023/tex/fig/pdf/loss/loss_Transformer_{PRE_TRAIN_DATA_NUM}.pdf",
      format='pdf')
692 plt.close()
693
694 plt.figure()
695 plt.plot(loss_A2B2A, label="A2B2A")
696 plt.legend(fontsize=15)
697 plt.xlabel("Epoch", fontsize=15)
698 plt.ylabel("Loss", fontsize=15)
699 plt.savefig(f"D:/SynologyDrive/SynologyDrive/2022/SemiReporot/shuron/
      template2023/tex/fig/pdf/loss/loss_A2B2A_{PRE_TRAIN_DATA_NUM}.pdf", format='
      pdf')
700 plt.close()
```