卒 業 研 究 報 告 題 目

BERTを用いた事前学習の違いによる誤字訂正の比較

Comparison of Error Correction with Different Pre-Training Using BERT.

指 導 教 員　　出口 利憲 教授

岐 阜 工 業 高 等 専 門 学 校　電気情報工学科

19E13 菱田 隼人

令 和　　０ ６ 年（ ２ ０ ２ ４ 年）　　２ 月 １ ６ 日 提 出

Abstract

This study aims to evaluate the effectiveness of various pre-training methods in NLP for sentence generation, using BERT to correct errors. This paper examines the potential impact of limited computational resources on pre-training accuracy.

The dataset was used to evaluate the accuracy of responses in identifying misspellings within sentences. However, it was found that correct sentences were also identified as misspelled. Therefore, further improvements are necessary to enhance the accuracy of the identification process.

目 次

第1章　序論

2022年のChatGPTの公開以降、機械や人工知能による文章生成は世間の注目を集め、個人や企業で活用への関心が高まっている。自然言語処理においては特に要約や分類などのタスクでの活用が期待されており、実用的な活用方法なども考案されていくことであろう。人間と機械の共同作業はこれから日常的になっていくはずだ。

ここで、私たちが日常的に文章に対して煩わしさを大きく占めるエラーは、誤字であろう。誤字を判定するのは短文であれば簡単だが、長文では無視できない労力を要する。誤字を機械が判定できるようになれば、時間や労力を大きく軽減できるようになるはずだ。しかし、現状の誤字訂正は固有名詞を誤りと判定することがあり、誤字の状況によっては判定することが難しい。また、高機能な誤字訂正を利用するには高額な対価を支払い、巨大な計算資源を必要とすることになる。

そこで、本研究では家庭用の計算資源とオープンソースのデータベースで誤字訂正機能を有したプログラムを作成し、実際に多く利用されている処理モデルとどれほどの性能差があるのかを確認し、その性能差はどのようにして埋めることが出来るのかを考察することを目的とする。

本研究ではBERTという自然言語処理モデルの学習が二段階に行われるという特徴を利用することで、条件を揃えた実験かつ高性能な自然言語処理を以て誤字検出を実行する。用いるアーキテクチャは一般的に公開されているものを前提とし、かつ、その学習元のデータベースも一般的に公開されだれでも利用できるもののみで学習したものを前提とする。

第2章　機械学習とその背景

2.1　自然言語

2.1.1　自然言語

自然言語は自然発生的に誕生した、人間が日常的なコミュニケーションの手段として用いる言語のことである。

英語や日本語、アイヌ語や手話に至るまでコミュニティーに沿ったある程度の規則が存在し、なおかつ厳密な制約がない柔軟な表現を持つことが特徴である。

自然言語に対して人工言語も存在し、ある意味ではエスペラントやニュースピークのように現実または創作で人間が人間のコミュニケーションの手段として使用するよう想定して作成した言語、そしてある意味ではPythonやアセンブリのような機械的な処理のために作成された言語も意味する。

自然言語と人工言語は文脈による解釈という違いがある。自然言語では同一の文や単語でも前後の文や単語によって異なる意味を持つことがある。

例えば、“頭が赤い魚を食べる猫”という文がある[1)]。この文は頭が赤いのが猫なのか魚なのか、食べるのは頭か猫かがそのままでは不定である。これは自然言語の曖昧性を極端に写し取ったものだが、ここまで極端に不定な構造でも自然言語は許容することが出来るということである。

このような自然言語の曖昧性を機械的に処理するアルゴリズムは自然言語処理と呼ばれる。

2.2　機械学習

2.2.1　機械学習

機械学習は過去の経験を用いて現在または未来のタスクの精度を改善するように作られたアルゴリズムである。

経験とはアルゴリズムに与えられた過去の入力や出力、またはそれを導き出すために設定したパラメータ類であり、タスクとはアルゴリズムが与えられる課題である。精度はタスクの達成率を指すことが多い。

主な機械学習タスクは“教師あり学習”、“教師なし学習”、“強化学習”の3種類のカテゴリのどれかに属することが多い。

2.2.2　教師あり学習

教師あり学習は入力データとそれに対する正答をモデルに与える。そのデータはある確率分布に沿っていると仮定し、モデルは未知の確率分布を近似することで正答を予測する。

2.2.3　教師なし学習

教師なし学習は教師あり学習に対して、正答を与えない。データは未知の確率分布に沿っていると仮定するが、明確な正答が存在しないタスクに用いられた場合は主観的な判断でタスクの精度を判断する。

2.2.4　強化学習

強化学習はある環境に状態というパラメータ、エージェント自身に行動というパラメータを割当て、エージェントは状態から行動を決定し、環境はそれに応じてエージェントに報酬を与え状態を変化させる。というサイクルで最適な行動を学習する。

2.2.5　ニューラルネットワーク

ニューラルネットワークは教師あり学習や教師なし学習を問わず、機械学習の内部に用いられる数理モデルの 1 つである。人間の神経接続を模した構造をしており、ネットワークの名の通り、モデルは同列のノードをまとめた複数のレイヤにより構成され、ノードは何かしらの数値的な処理を行う。例えば、シグモイド関数 (Figure 2.1) やハイパボリックタンジェント (Figure 2.2)、Softmax 関数のような活性化関数を用いる

ニューラルネットワークの層を増やしたものはディープラーニングと呼称されている。

この論文に登場する再帰型ニューラルネットワークや Transformer もニューラルネットワークである。

2.2.6　シグモイド関数

シグモイド関数は $\varsigma_a(x) = \frac{1}{1+e^{-ax}}$ で表される関数である。入力 x を 0.0 から 1.0 の範囲に変換して出力する。 a はゲインを表し、値が大きいと勾配が大きくなる。 a が 1 のシグモイド関数は標準シグモイド関数と称される。ニューラルネットワークで使用されるときは、次のノードに伝達するための活性化関数として使用されることが多い。活性

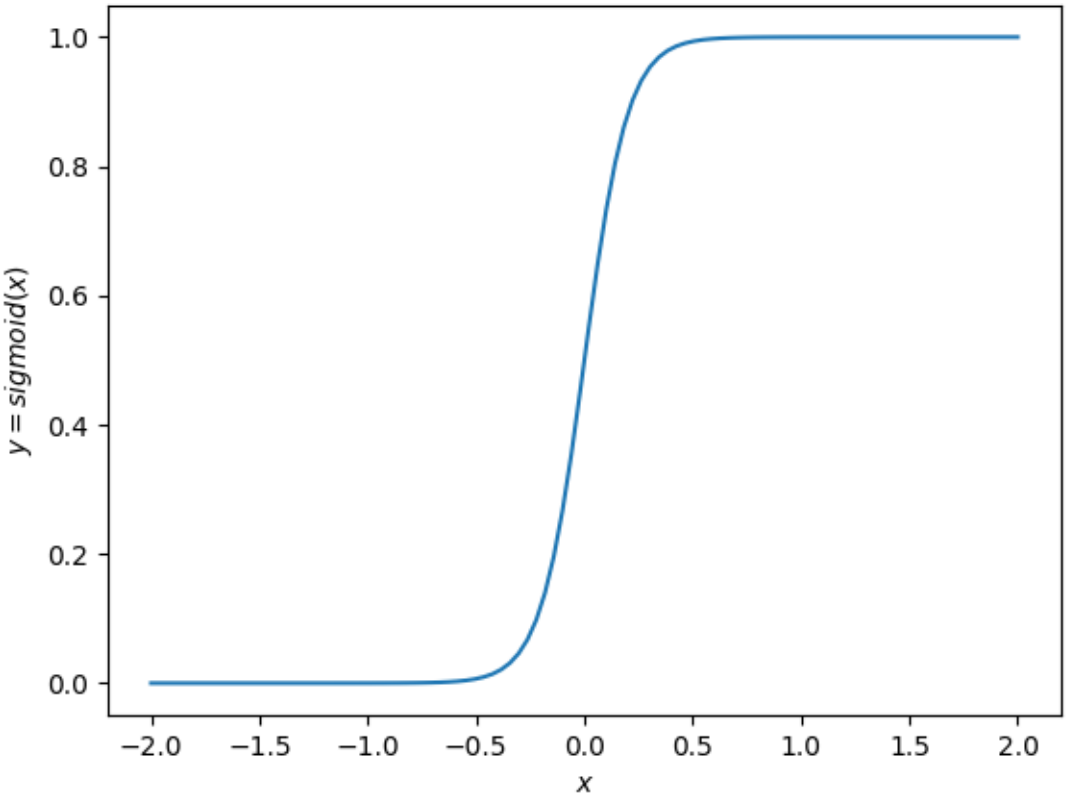

Figure 2.1: Example of sigmoid function.

化関数はパーセプトロンではステップ関数を用いていたが、バックプロパゲーションの登場に伴い微分可能な関数であることから採用された。また、シグモイド関数はハイパボリックタンジェントを用いた式、$\frac{\tanh(x/2)+1}{2}$ で近似でき、両者の形状は酷似している

2.2.7　ハイパボリックタンジェント

ハイパボリックタンジェント (双曲線正接関数) は $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ で表される関数である。入力を -1.0 から 1.0 の範囲に変換して出力する。シグモイド関数を拡張した関数として扱われることがある。シグモイド関数の微分係数は最大 0.25 になるが、ハイパボリックタンジェントの微分係数は 1.0 になり、微分係数が大きいことで勾配が大きくなり学習時間の短縮につながるため採用されていた。

しかし、シグモイド関数・ハイパボリックタンジェントは勾配が 0 に近い区間も存在し、一度勾配が 0 に近くなると学習が行われなくなってしまうという問題があった。これはニューラルネットワークの多層化に伴い大きな課題となったため、現在のディープラーニングの活性化関数は専ら正規化線形関数が用いられる。

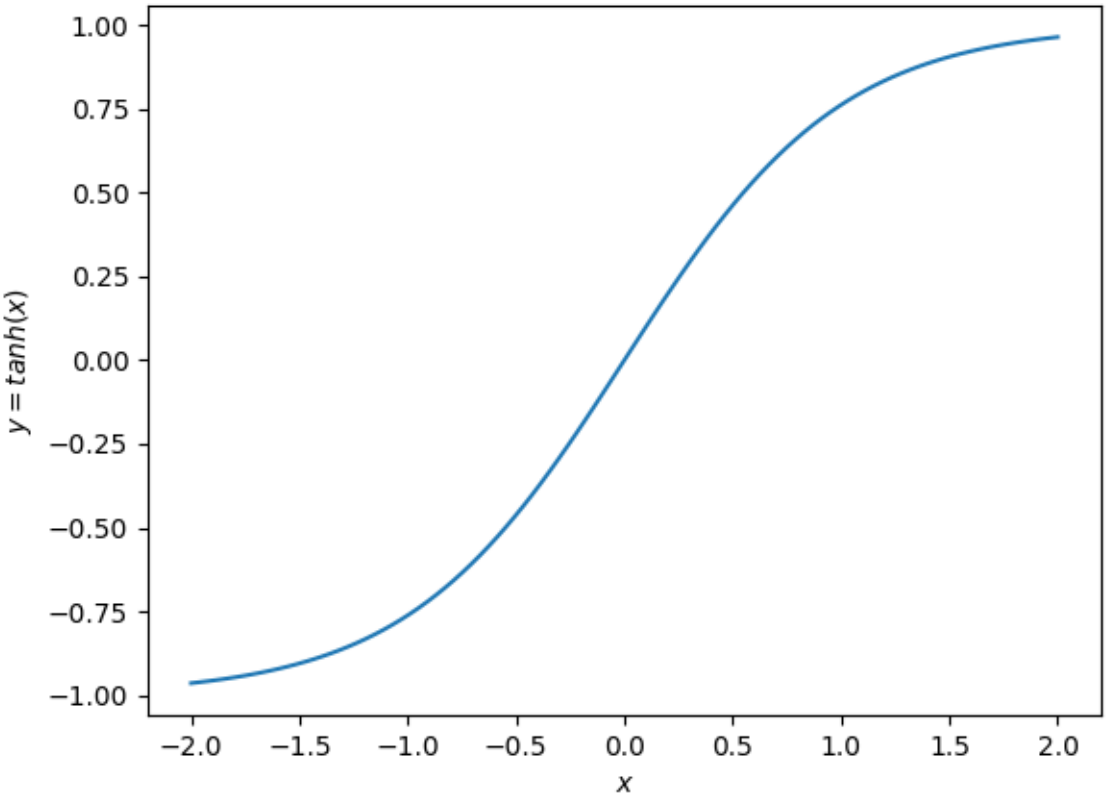

Figure 2.2: Example of hyperbolic tan function.

ハイパボリックタンジェントは RNN の最終層に活性化関数として位置し、次時刻の RNN への伝達に用いられる。

2.2.8 Softmax 関数

Softmax 関数は $f_i(x) = \frac{e^{x_i}}{\Sigma_{k=1}^{n} e^{x_k}}$ で表される関数である。n 個ある入力 x_n の総和を 1 にするように変換して出力する。n が 2 の場合はシグモイド関数となる。活性化関数としてはよくニューラルネットワークの最終層として用いられ、Transformer の最終層は softmax 関数である。

2.3 Python とそのライブラリ

2.3.1 Python

Python はオープンソースのインタプリタ型プログラミング言語である。

インタプリタとはソースコードを逐次機械語に翻訳・実行する形式であり、1 行ずつ実行して動作を確認するなどが可能である。

最たる特徴は豊富な種類のライブラリであり、本研究においては後述の Transformers をはじめとし、Pytorch、Matplotlib、Numpy を利用している。

2.3.2 Pytorch

Pytorch は Python 用のオープンソース機械学習ライブラリである。GPU のサポートやフレームワークの変換などの環境構築やテンソル演算や並列実行などの動作の改善を主とする。

本研究では一部を除き殆どの機械学習を Pytorch で行った。

2.3.3 TensorFlow

TensorFlow はオープンソース機械学習ライブラリであり、こちらは Python を初めとして C や C++や Java にも対応している。

Ubuntu での事前学習の改善の際には一度プログラムを TensorFlow で構築した。

2.3.4 Matplotlib

Matplotlib は Python のグラフ描画ライブラリである。様々なグラフ描画のサポート機能を有する。

2.3.5 Numpy

Numpy は Python の数値計算の強化を目的としたライブラリであり、ndarray や独自の関数により高速な数値計算を行うことが出来る。

2.4 形態素解析

2.4.1 形態素

形態素は言語の意味を持てる最小単位であり、所謂単語や接続詞がそれに該当する。

例えば“吾輩は猫である”を形態素解析ツール MeCab で形態素解析すると Table 2.1 のような出力が行われる。

Table 2.1: Examples of morphological analysis

Written form	Lexeme	Parts of speech	Inflected form	Standard form
吾輩	我が輩	代名詞	*	吾輩
は	は	助詞-係助詞	*	は
猫	猫	名詞-普通名詞-一般	*	猫
で	だ	助動詞	連用形	だ
ある	有る	動詞-非自立可能	終止形	ある

2.4.2 形態素解析

形態素解析は文を形態素に分割する処理のことで、自然言語処理の 1 つである。現在では検索エンジンや IME で活用されており、多様な形態素解析ツールが存在し、その仕組みは様々である。

2.4.3 MeCab

MeCab は京都大学情報学研究科と日本電信電話株式会社のコミュニケーション科学基礎研究所が共同開発した、日本語の代表的な形態素解析ツールである。OS や言語、コーパスに依存しない汎用的な設計により多くの採用例を有している。内部的にはマルコフ連鎖をベースとした Conditional Random Fields(CRF) を使用し、他の形態素解析ツールよりも高速であるとされる。

2.4.4 JUMAN++

JUMAN++は京都大学で開発された形態素解析ツールで、MeCab や前世代の JUMAN とは異なり Recurrent Neural Network Language Model (RNNLM) を用いた高性能な解析を特徴としている。

京都大学による RoBERTa の動作環境に含まれているが、Linux でしか動作しない。

2.4.5 サブワード

サブワードは形態素解析をさらに発展させたものであり、低頻度の形態素の要素の共通項を語彙として登録することで全体の語彙数を減らしつつ未知語の頻度を減少させることができる.

例えば“吾輩”はそのままでは未知語として扱われる場合がある。サブワードを定要した場合“吾”(我と同じとして扱われる)と“*輩”(アスタリスクはワイルドカード)に分解され、機械的には“先輩”や“後輩”の類義語として処理される。これにより未知語を削除することが出来る。

2.4.6 SentencePiece

SentencePiece はサブワード分解ツールであり、形態素を事前に設定した語彙数に達するまで分割しサブワードを生成する。

日本語を前提に作成されたツールだが、同一コーパスに複数言語が混入していても問題なく分割できるという特徴がある。形態素解析とは同列の目的・用途ではないが、形態素解析機能を同梱している。

2.5 従来の自然言語処理と活用

2.5.1 Word2vec

Word2vec は単語の分散表現の取得のために開発されたモデルであり、実際には Skip-gram と CBOW の 2 つの技術の総称である。

そもそも単語の分散表現とは単語を固定長ベクトルで表し、意味を定量的に表すための手法である。ベクトル化により単語同士の関連度や加算減算が出来るようになり、単語を数学的な計算により求めることができるようになった。

なお Word2Vec は文の並びの情報を利用することが出来ないため、言語モデルとしての活用はできない。

2.5.2 RNN[2)]

RNN は Recurrent Neural Network と呼ばれ、深層学習において過去の情報から現在・未来の入力に対する精度を向上させる構造が最たる特徴である。

この時系列に従って処理を循環させる機構は時間に従って展開することが可能である。例えば文を翻訳するときは、分かち書きされた入力順に展開された RNN が入力を処理する (ように見える) ことで 1 文分の入力を意味するある状態ベクトル (文脈ベクトルと呼ばれる) を出力することが出来る。この機構を Encoder と呼ぶ。

Encoder によって処理された状態ベクトルは機械にしか読むことが出来ないため、En-

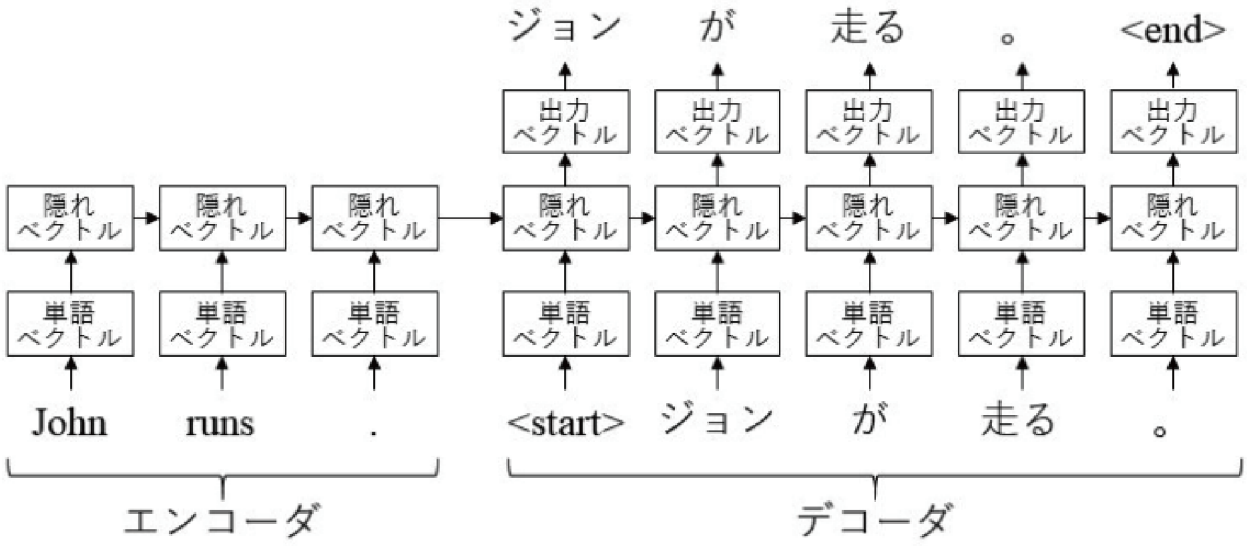

Figure 2.3: Outline of Seq2seq.

coder と同様にして人間の言語に再翻訳する構造が必要である。この機構を Decoder と呼ぶ。この Encoder と Decoder の 2 者を用いて言語処理を行うモデルを Encoder-Decoder モデルと呼び、RNN を用いた Encorder-Decoder モデルは Sequence-To-Sequence(以下 Seq2Seq) と名付けられている。概略は Figure 2.3 に示す。

RNN の詳しい構造は、各時間に対する入力に重みを掛けたベクトルと前機構の結果に重みを掛けたベクトル (隠れベクトルと呼ばれる) の和にバイアスを与えたものを、ハイパボリックタンジェント (tanh) で処理するという構造になっている。

この回帰的なモデルでは短い文を本質的に翻訳することが得意であるが、自然言語処理でニュアンスを重視する場面は少なくない。

この RNN の最終処理はハイパボリックタンジェントである。そのため、主に長文の翻訳において多数の問題が顕在することになった。例えば小数点以下の値の乗算を繰り返すと勾配が消失したり爆発したりして文の特徴が失われること、文を表すベクトルは固定ベクトルであるために文が長ければ長いほど文の先頭側の情報が欠落してしまうことがあること、分散表現を取得しようとするとコストが多くことかかるなどである。

RNN の文の勾配が消失するという問題を解決するため、Seq2Seq では LSTM(Long Short-Term Memory) が誕生した。これは全ての時系列の情報と、その情報が必要かどうかが記録される専用の記憶部を持つことで、RNN は必要な情報のみを取り出して次時刻に出力する事が出来るようになった。これにより、Seq2Seq は数十単語ほどの文でも学習が可能になった。しかし、これでは更に単語が多くなれば記憶すべきベクトルの次

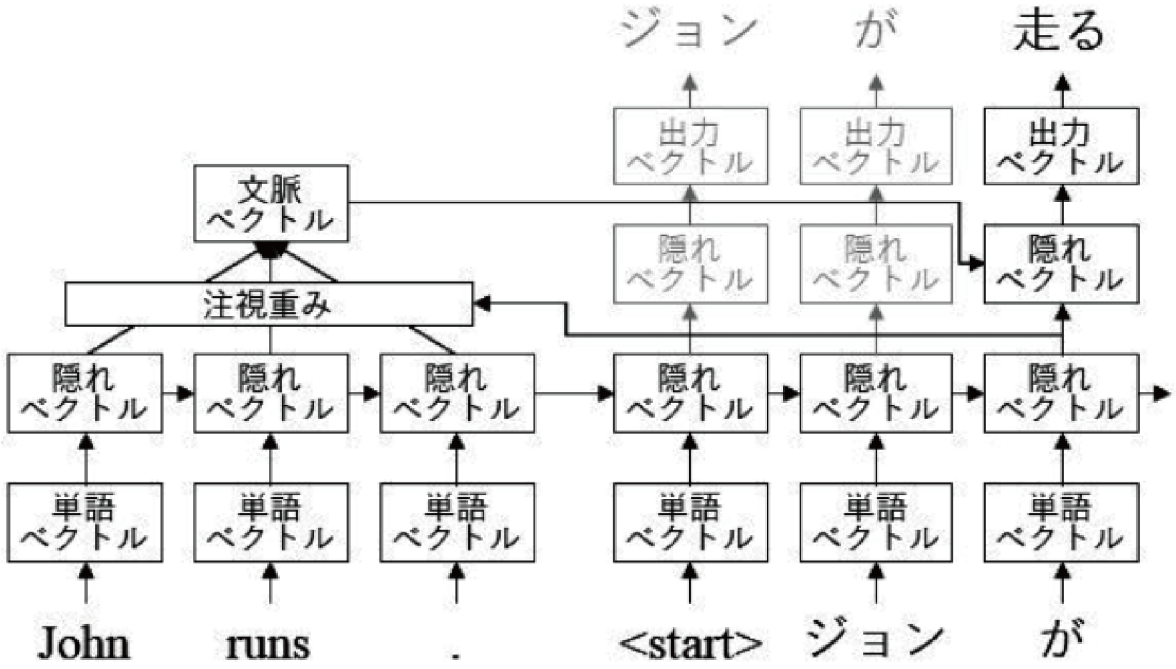

Figure 2.4: Outline of Seq2seq with attention.

元数が爆発してしまうし、次元数を固定とすると情報が欠落してしまうという制約を抱えていた。これらを解決するための手法として開発されたのが Attention である。

2.5.3 Attention

Attention(注視機構) とは、ある 2 文の単語ごとの関係度を表す機構である。Seq2Seq においては Decoder が Encoder の隠れベクトルの重み付き和である、文脈ベクトルも参照することが出来るという形で実装されている。概略は Figure 2.4 に示す。

ここで、隠れベクトルの重みを更新するのは Decoder 側であるため、Decoder は Encoder が持つ必要な情報を Decoder 自身の隠れベクトルを介さずに取得することが出来るようになる。隠れベクトルの重みは、最も単純なものでは Decoder と Encoder の隠れベクトル同士の内積による方法が用いられる。つまりは Decoder は Encoder の各隠れベクトルとの関係度を注視している。

これで隠れベクトルの次元数爆発や情報の欠落を大きく軽減しつつ、更に精度の高い文生成を可能となった。しかし、RNN は文の要素を時系列に従って逐次的に実行するため処理が長く、かつ並列化出来ないので高速に実行できないという問題が顕在した。もはや RNN を改善することでは対応できない根本的な要因に対し、新たなモデルの提案が行われた。

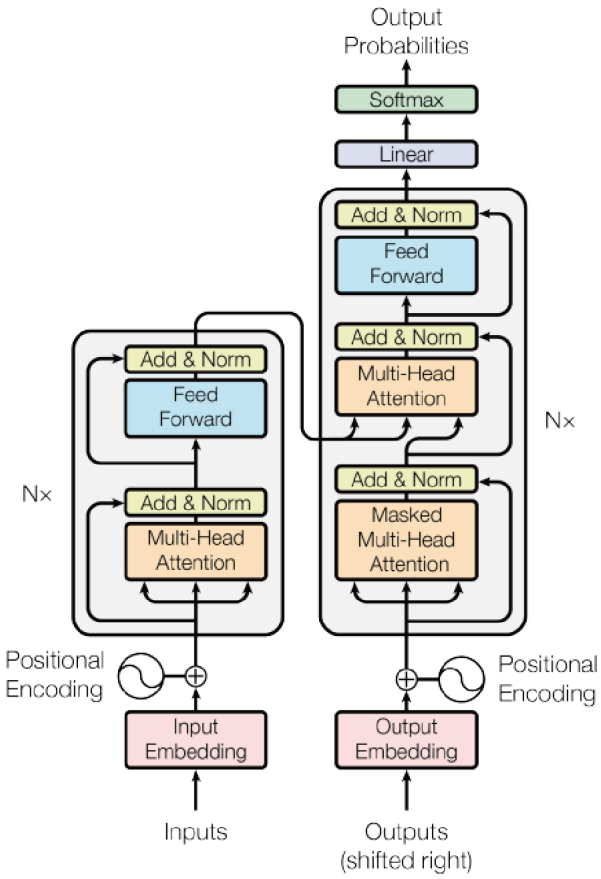

Figure 2.5: The transformer model architecure.

2.6 Transformers

2.6.1 Transformer

Transformer は 2017 年、Google によって発表されたアーキテクチャであり、以前より主流であったモデルの RNN に対し機械翻訳のタスクにおいて学習コストとスコアが優秀であるとしている [3)]。

それらのモデルとの最も大きな差異は Encoder-Decoder モデルでありながら内部には RNN を一切用いず、両者は Self-Attention(自己注視) という層で接続されていることである。概略は Figure 2.5 に示す。なお、図の左半分は Encoder で右半分が Decoder である。

まず両者を分離して RNN を用いないことで、Encoder は時系列に縛られずに単語毎に並列に処理を行えるようになり、Decoder は文生成を行わない時は並列処理を行えるようになった。しかし、これでは文中の単語の位置の情報を特定できなくなってしまう。そのため、ここで Self-Attention を用いることになる。先述した Attention は Decoder が Encoder の情報を参照する手法だったが、Self-Attention は Encoder は自分自身の入力の他の単語を、Decoder は自分自身の出力の他の単語を選択的に参照する手法を指す。

また、Attention の前には Positional Encoding という処理を行い、文中の単語の位置を単語にあらかじめ埋め込んでおく。これにより Self-Attention の層で位置情報が消失・変化しても元の入力の位置は参照できるようになっている。

Transformer を提案した論文”Attention is All You Need“[3] ではこの Self-Attention は Encoder と Decoder にそれぞれ 6 層ずつ実装することで単語同士の関係度を明らかにしている。さらには柔軟な観点での注視を実現するため、Multi-Head Attention(複数ヘッド注視) という多数の Self-Attention によって得られた情報を元にベクトルを生成する方式が使用されている。単純な Self-Attention は Figure 2.6 に、Multi-Head Attention の概略図は Figure 2.7 に示す。柔軟な視点とは単語の順番や種別による表現であり、それはつまり文章のニュアンスを重視することが出来るようになったということである。

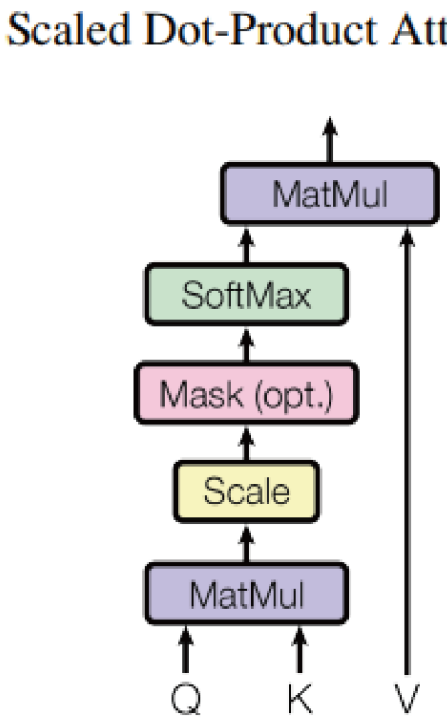

Figure 2.6: Scaled Dot-product Attention.

また、Decoder の最初層は Masked Multi-head Attention という特別な機能が与えられている。これは学習時に予測を行う単語をマスクしてから選択を行う手法であり、タスクの学習時にカンニングを行うことを防ぐ処理である。

Transformer の最終の処理は Softmax 関数である。Softmax を採用したことで、Trans-

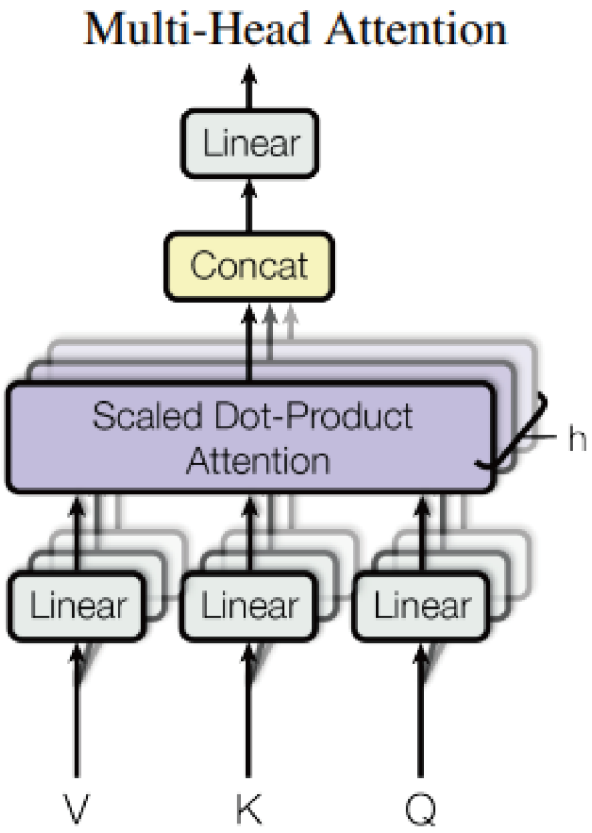

Figure 2.7: Multi-head Attention.

former は RNN のような勾配消失問題は発生しにくいとされている。

2.6.2 BERT[4)]

BERT は Bidirectional Encoder Representation from Transformers の略で、Transformer の Encoder を用いたモデルである。2018 年に Google によって発表されたこのモデルは、文を双方向、つまり前からも後ろからも学習することで文脈を織り込んだ出力を行えるという強力な特徴を有している。これは Transformer をモデルとする言語モデルの GPT とよく比較される点である。BERT の概略図を Figure 2.8 に示す。

BERT はそれだけではなく、タスクによってアーキテクチャを変更しないという特徴もある。サイズの異なる BASE と LARGE の 2 種類のみが区別され、例えば感情分析や質問応答も同じアーキテクチャ上で行える。しかし、それだけでは効率的な学習を行えないため、事前学習を行い、次にファインチューニングを行うことでタスクに特化したモデルを生成することが出来る。

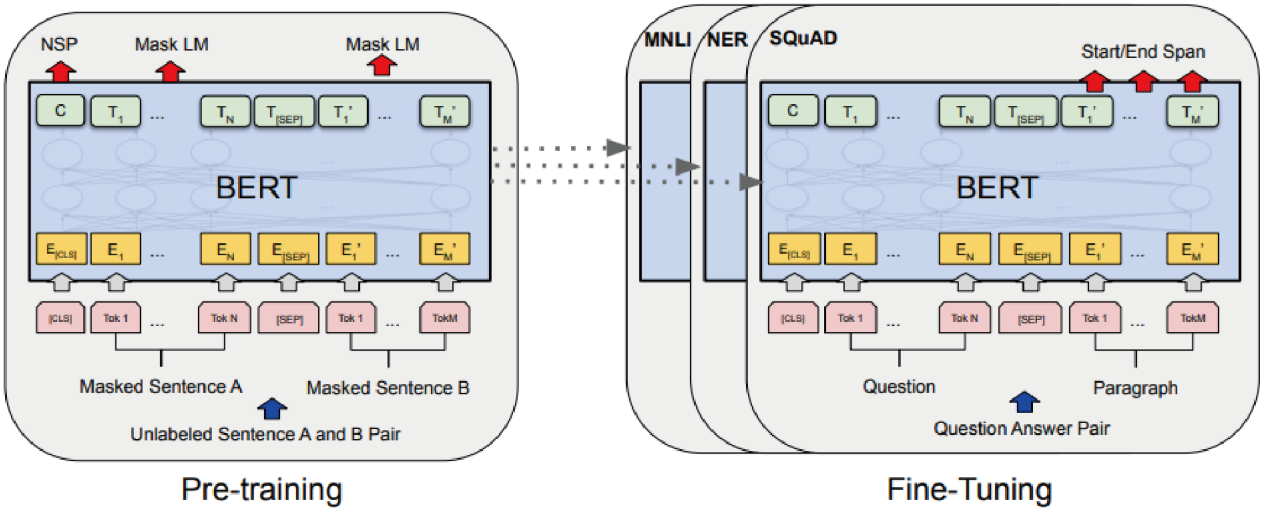

Figure 2.8: Overall pre-training and fine-tuning procedures for BERT.

2.6.3 トーカナイズ

入力された文はトーカナイザーにより形態素解析ののち意味的に分割しトークン化される。多言語対応のために行われる処理で、形態素解析ツールと大量のデータセットが存在すればコンピュータで表せるどんな言語でも BERT は自然言語処理を行える。さらにスペシャルトークンを明示的または自動的に追加し、文の構造や機能を理解させることが出来る。

2.6.4 CLS トークン

スペシャルトークンの 1 つで、文の始まりを意味する。BERT は CLS トークン以降の単語の羅列を文章として認識し、文の分散表現を求めるようになる。

2.6.5 SEP トークン

スペシャルトークンの 1 つで、文の接合部、または終了を意味する。SEP トークンを文で挟むと 2 文として認識し、SEP トークン以降のトークンが無いとそこを文の終端として認識する。

2.6.6 MASK トークン

スペシャルトークンの 1 つで、トークンが隠されていることを表す。事前学習やタスクで使用すると、BERT は MASK トークンで隠されたトークンを予測しようとする。

2.6.7　事前学習

事前学習は巨大なデータセットをラベル無しで入力することで行われる。入力する文が 1 文か 2 文かで、後述の 2 種類のタスクのどちらか、または両方を選択することが出来る。

事前学習されたモデルはファイルで保存されるため、事前学習のみ強力なマシンで行いファインチューニング以降他のマシンで動作させたり、Hugging Face や GitHub で公開することもできる。

2.6.8　Masked Language Modeling

Masked Language Modeling (以下、MLM) は BERT に実装された事前学習の 1 つで、ランダムなトークンをマスクトークンで隠して本来のトークンを当てさせるタスクである。なお、マスクされたトークンは 10 % の確率で他のトークンに置き換わったり、トークンを開示したりなどのイベントを発生させ、ファインチューニングとの差異を軽減している。

BERT の実行時はこれと同じ仕組みでトークン生成を行うことが出来るため、BERT の文章生成の基本的な仕組みである。

2.6.9　Next Sentence Prediction

Next Sentence Prediction(略語は NSP) は BERT に実装されたもう 1 つの事前学習であり、入力された 2 文が連続しているかを推測するタスクである。

これ自身は文章生成を行うわけではなく、あくまである文とある文が連続的であるかのみを判断する。

2.6.10　ファインチューニング

ファインチューニングは特定のタスクの精度を向上させるために少量のラベル有りデータを用いて学習を行う手法である。Figure 2.9 のように各種タスクに合わせたファインチューニングが提案されているため、ユーザーは適するラベルを選択する必要がある。

これは BERT だけでなくトーカナイザーにも学習させることが出来る。事前学習後のモデルを初期値として、初期値を変動させずにパラメータを変動させられることから、BERT のタスク毎にアーキテクチャを変化させなくても良いという特徴の一助となって

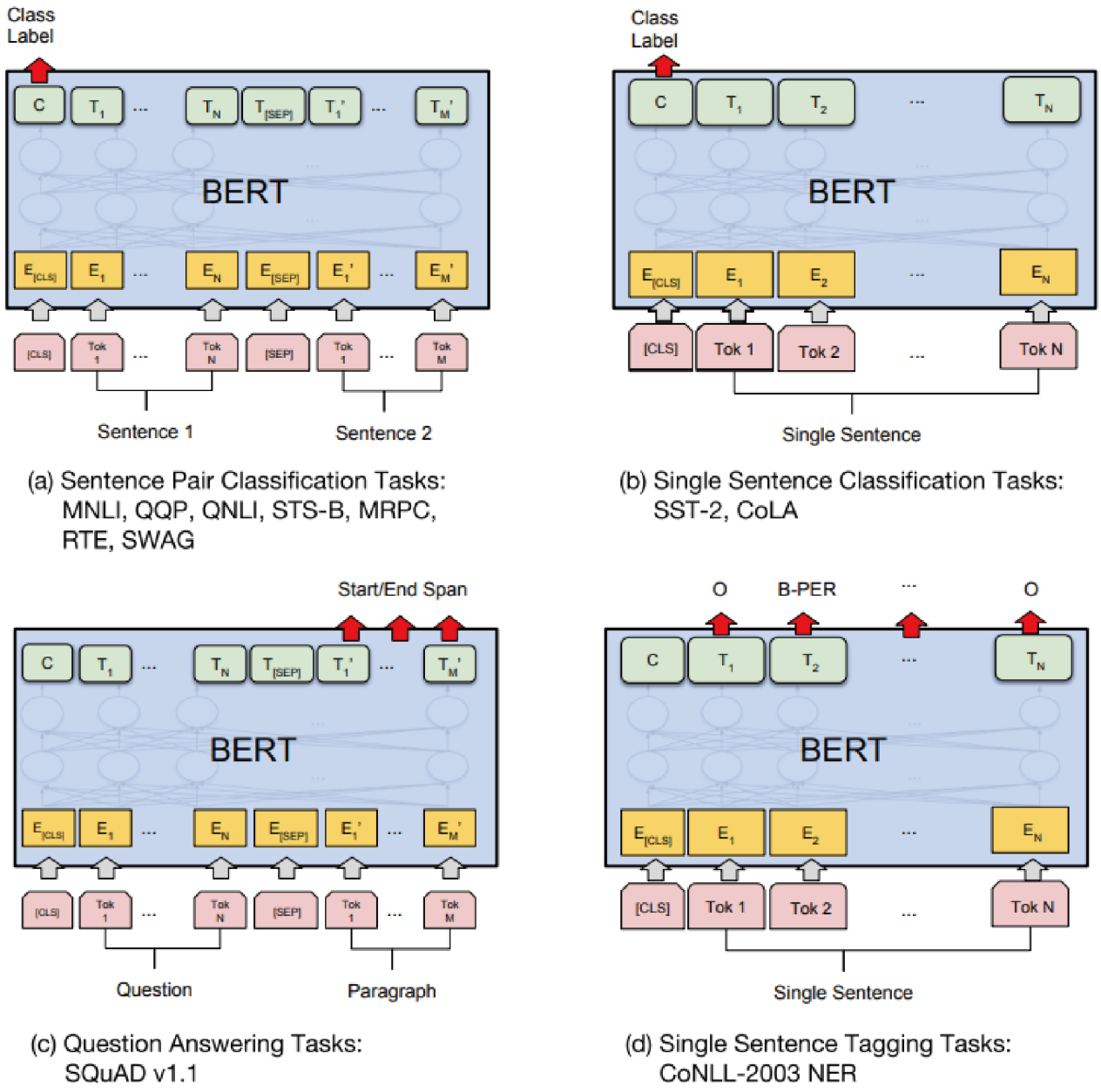

Figure 2.9: Illustrations of Fine-tuning BERT on different tasks.

いる。

2.6.11 Hugging Face

Hugging Face は BERT や GPT などの自然言語処理を初めとする様々なアーキテクチャを公開・共有できるプラットフォームである。

BERT の事前学習には巨大なリソースと時間を必要とする。例えば BERT を発表した論文[4] では BASE モデルの学習には 4 つの TPU を 4 日間稼働することで学習が完了したとしている。これを個人で行うには非常に労力がかかる、そのため比較用に Hugging

Face 上で日本語対応の事前学習済みのモデルを入手した。

2.6.12 Transformers

Transformers は Huggin Face が提供する Transformer を円滑に利用するためのライブラリである。主に PyTorch や TensorFlow と Hugging Face 間との連携やモデルのアップデートの提供機能を主とするが、今回は事前学習を行える機能を利用する。

2.6.13 RoBERTa と AlBERT

RoBERTa は BERT の事前学習を改良したモデルであり、主に MLM に於いてマスクパターンを変更する動的マスキングを実装することで過学習が軽減され性能が向上している。

AlBERT は A Lite BERT の略称で、BERT の軽量モデルである。パラメータが膨大でモデルの拡張が難しいという BERT の欠点を改善したことで、パラメータ数を削減しかつ精度を向上を実現している。これにより一般的な GPU でも比較的軽量に事前学習を行えるようになった。

第3章　実験準備

3.1　目的

この実験の目的は、誤字判別において自作の少リソースで構築したBERTと、公開されているBERTを比較し可用性を確認することである。

3.2　構成

当実験に使用したPCの構成は以下の通りである.

- CPU:Intel i5 10400F
- GPU:Nvidia GeForce　RTX3060ti 8GB
- RAM:32GB
- OS:Windows 10
- Python 3.9.4

3.3　環境の構築

3.3.1　環境の変遷

この実験の環境は最終的にはWindowsで構築されているが、最初期はVirtualBox上に構築したUbuntuで実行し、トーカナイザの準備の一部まではUbuntu上で行った実験準備のデータをWindowsに移行して使用している。そのため、Linuxを前提とした一部の機構は移行と同時に代替できる機構に変更している。

また、事前学習において、BERTのConfigはMacを利用して試行した設定も含まれている。

3.3.2　実行環境の構築

実行環境は通してPython3.9.4で実行し、PyTorchのバージョンは22.0、なお、Pytorchは実行するプラットフォームを指定する必要があるが、Geforce RTX 3060tiを使用する場合のプラットフォームはCUDA 11.8である。

エディターはUbuntuはnano、WindowsはVisual Studio Codeを使用している。PythonはPyランチャーを介して実行した。

3.3.3　データセットの取得

データセットは誤字判別に京都大学の“日本語 wikipedia 入力誤りデータセット”[5] と、事前学習には併せて Wikipedia をスクレイピングした自作データセットを使用した。

日本語 wikipedia 入力誤りデータセットはトレーニング用とテスト用のデータセットが収録されており、誤字の含まれる文と訂正文がセットになっている。事前学習ではそのなかからトレーニング用の訂正後の文のみを与え、ファインチューニングにはトレーニング用のデータセットをどちらも与えることとした。

自作データセットは 2022 年の 5 月ごろにビックデータ解析で使用したものを流用した。Wikipedia において“秀逸な記事”の評価を獲得している記事とランダムに取得した記事をスクレイピングしたものである。容量の制約から全てのデータは用いず、ランダムに文を抜き出したものを使用した。

3.4　BERT モデルの自作

3.4.1　コーパスの用意

BERT の自作モデルを制作するに於いて、まずは事前学習用のデータセットを 1 行ごとに取り出せるようにコーパスに変換する。

コーパスはルビ・タグ・特殊記号・改行を削除し、句点までを 1 行としてテキストファイルに保存する。本来は億単位の文データで学習するのが望ましいとされているが、少リソースでの構築を目的としていることやメモリサイズを考慮し約 700 万行程度に抑えた。

3.4.2　トーカナイザの準備

まずは先行事例[6] と同様に MeCab と WordPiece で形態素解析とサブワードへの変換を行いトーカナイザの学習を行った。しかし、実行環境の変化に伴い動作しなくなったため、急遽 SentencePiece のみを用いた学習へと変更した。

SentencePiece に移行したが、使用していた BERT のトーカナイザが SentencePiece の形式に未対応であったため、トーカナイザのみ AlBERT のものを使用することにした。

また、JUMAN++を用いて形態素解析ツールの違いによる学習結果の比較を行う予定であったが、JUMAN++は Linux のみでしか動作しなかったため採用を見送った。

3.4.3 事前学習

事前学習以降はWindowsを用いて実行することとした。

BERTにはConfigを設定する。語彙数はデフォルトの32003とし、アーキテクチャーは隠れ層とヘッドが12層のBERT-baseとした。また、1文のみを読みだすことを前提としているため事前学習はMLMのみを行うこととした。コーパスから読みだした文は自動的に1文ずつ読みだしてトークンに変換するように設定。エポック数は10で、バッチサイズは32で設定したが、試行の結果バッチサイズは4で実行した。

いざ事前学習を行うと、GPUのメモリ不足により度々停止してしまうようになった。調査の結果、GPUのメモリは通常のPythonのキャッシュ処理では解放されないことが判明したため、バッチサイズの縮小やRAMへのモデルのマウントなどを行い28時間ほどで事前学習が完了した。

3.5 BERTモデルの取得とファインチューニング

3.5.1 BERTモデルの取得

自作モデルとの比較として、Transformersを介して2種類のモデルを取得することとした。

1つは東北大学 自然言語処理研究グループによる “cl-tohoku/bert-base-japanese-v3”[6] である。

事前学習はCC-100データセット[7]と2023年1月2日時点の日本語版wikipediaのダンプファイルで行われており、サブワード分割はMeCabとWordPieceを用いている。アーキテクチャーは隠れ層とヘッドが12層のBERT-baseで、学習時間はTPUを用いて約19日とされている。

もう1つは安岡孝一氏による、“roberta-base-japanese-aozora-char” である。

こちらはBERTではなくRoBERTaであり、事前学習は青空文庫で行われており、サブワード分割を行わず1文字ずつトークン化している。アーキテクチャは隠れ層とヘッドが12層のRoBERTa-BASEで、学習時間はGPU(Nvidia A-100 40GBであることに注意)で19時間11分とされている。

3.5.2　ファインチューニング

ファインチューニング用にデータセットから文を抽出し、正解か不正解かをラベル付けした。BERT を提案した論文[4)]に従い、バッチサイズは 32、エポック数は 4 ですべてのモデルに対し行った。

第4章　実験

4.1　実験内容

4.1.1　概要

本実験は3種類のBERTに1127行の誤字の含まれた文とその訂正文、合わせて2254行を入力し、BERTが誤字が含まれていると判断したらFalseを、訂正文と判断したならTrueを返す。

正解率を以て可用性とし、自作のBERTでどれほどの性能を有するかを確認する。また、学習元のデータベースの違いで可用性にどれほどの違いがあるかを確認する。3種類のBERTはBERT-baseで制作した自作モデル(以下、自作モデルとする)、 東北大学がBERT-baseで制作した”cl-tohoku/bert-base-japanese-v3”(以下、東北大学モデルとする)、安岡孝一氏がRoBERTa-baseで制作した”roberta-base-japanese-aozora-char”(以下、青空文庫モデルとする)を用いる。

4.1.2　判断基準

誤字が含まれているかは、BERTのMasked Language Modelを利用する。まずはトークンをマスクし、マスクされたトークンと思われる単語を100単語生成する。100単語中に元の単語が含まれていれば正しいとし、含まれていなければ誤字であるとする。これを文のすべてのトークンで行う。

なお、100単語という基準は東北大学モデルに対し10文の誤字を一部含んだ平文を入力し、最も正解率が高かったものである。

4.2　実験結果

4.2.1　正解数

まずはTable 4.1に各モデルの誤字検出の正答数を示し、グラフをFigure 4.1に示す。

誤字検出において最も高い正答数を示したのは自作モデルであった、続いて青空文庫モデル、東北大学モデルとなった。自作モデルに関しては正答数が高すぎる印象を受け、単語の生成が正しく行えていないのではないかという疑念が生じる。青空文庫モデルは想定よりも高く、東北大学モデルとの差異を調べる必要がある。

続いて正しい文を正しく検出できた正答数をTable 4.2に示し、グラフをFigure 4.2に

示す。自作モデルは正答数が極めて少ないことが分かった。東北大学モデルは正しい文を検出する精度が他モデルよりも良いことが判明したが、誤字検出に比べると半分程度の数である。青空文庫モデルは 3 分の 1 程度の数に精度を落としており、想定よりも低くなった。

最後に 2 つの正答数を合わせたグラフを Figure 4.3 に示す。全体で最も正答数が高かったのは東北大学モデルであった。これより、平文の状態で誤字を検出するならば東北大学モデルが最も優れているということが分かった。誤字検出で正答数の高かった 2 者は正しい文章の検出で大きく精度を落としていることが分かる。

Table 4.1: The table of correctly answered as erroneous.

Model	Correct Answered	Total
Self-made	1103	1127
Tohoku University	816	1127
Aozora Bunko	925	1127

Table 4.2: The table of correctly answered as inerrant.

Model	Correct Answered	Total
Self-made	56	1127
Tohoku University	491	1127
Aozora Bunko	275	1127

4.2.2 正解率

上記の結果で正解率を計算したものを Table 4.3 に示す。最も高い東北大学モデルの正解率でも全体では 58.0 % であり、想定よりも精度は高くない。全体的な正解率の開きは 6.6 % ほどでありそこまで顕著な性能差がないことが分かった。

4.3 考察

4.3.1 自作モデルについて

今回の実験では自作モデルは誤字検出に関しては高い精度を示したが、正しい文に対しては極端に精度を落とすという結果になった。大きな要因は事前学習のデータセット

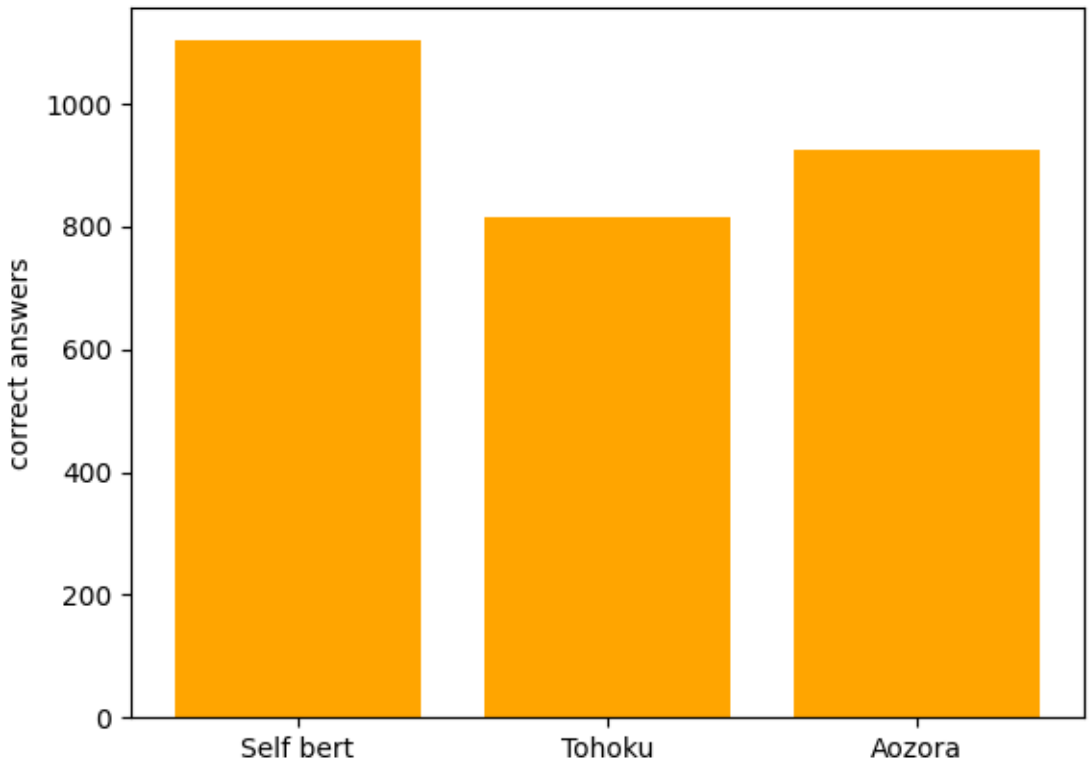

Figure 4.1: The graph of correctly answered as erroneous.

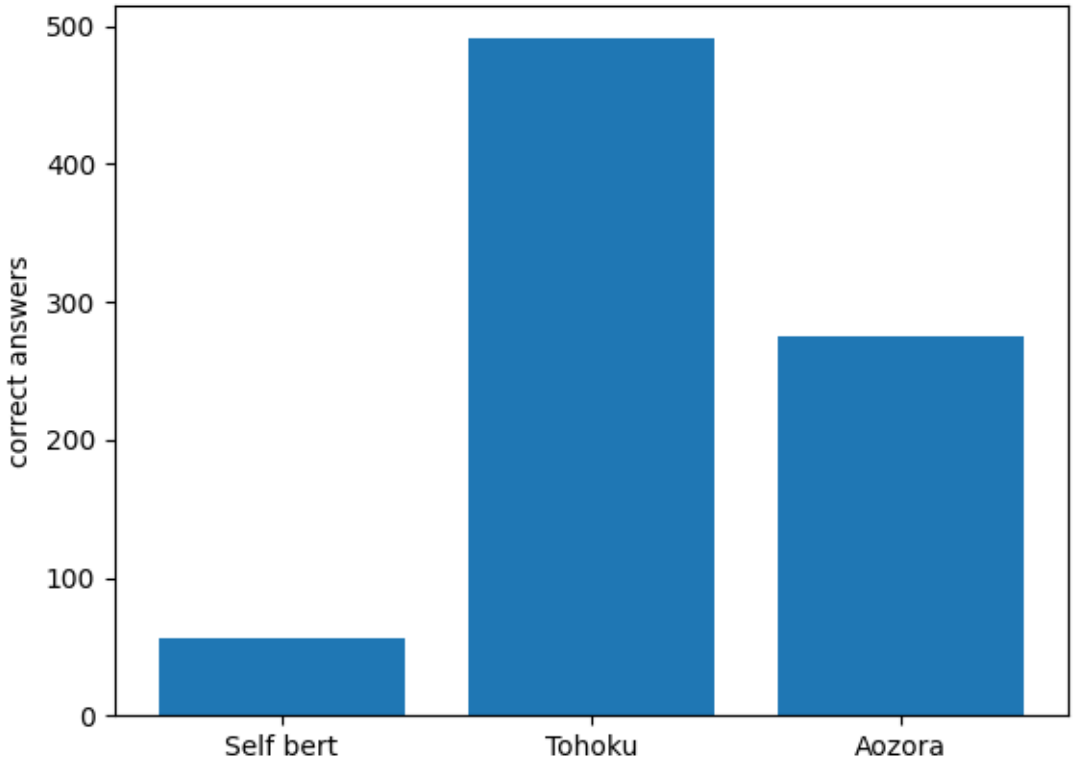

Figure 4.2: The graph of correctly answered as inerrant.

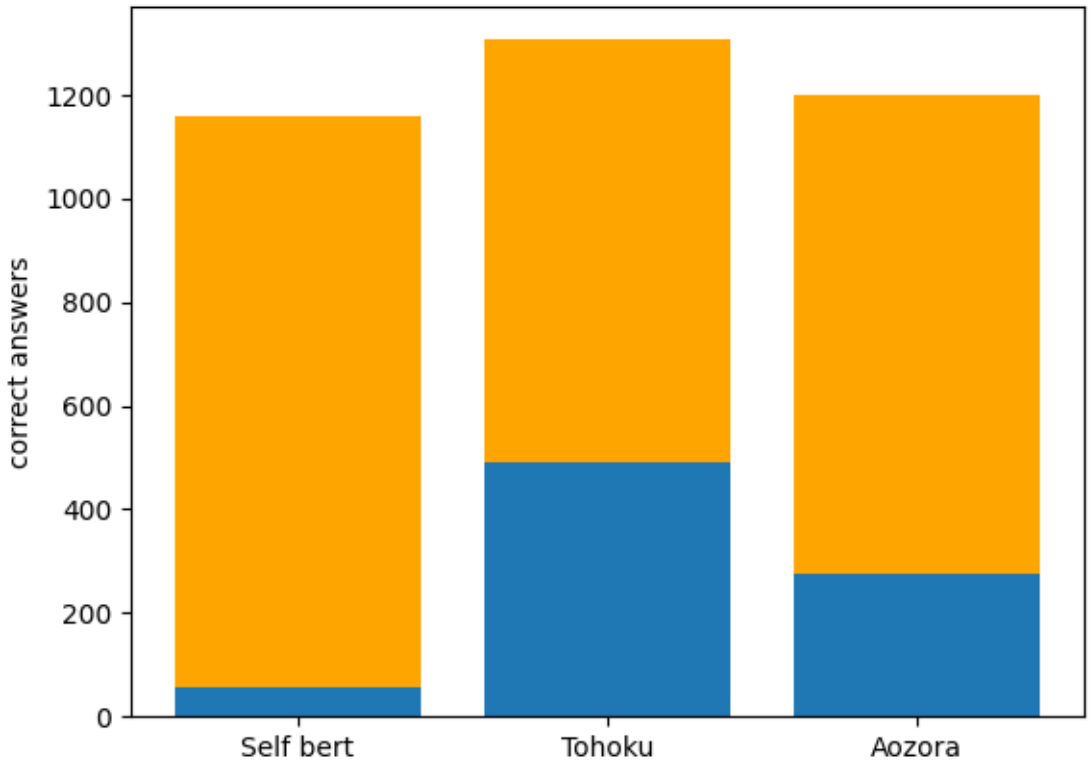

Figure 4.3: The graph of correctly answered.

Table 4.3: The table of accuracy as correctly answered.

Model	Incorrect (%)	Correct (%)	Total (%)
Self-made	97.9	5.0	51.4
Tohoku University	72.4	43.6	58.0
Aozora Bunko	82.0	24.4	53.2

の量であろう。正しい文の正解率が低いのはおそらく名詞のほとんどを誤りであると判断しているためであろうと思われる。誤字データセットの元は Wikipedia であり、様々な固有名詞が局所的に現れることから事前学習のデータセットでカバーできない語彙は誤りでありとしてしまっている可能性があり、事前学習においてのデータセットによる差異となっている。

これに対して精度を改善するには、まずは事前学習のデータセットの数を増やすという解決策が思い浮かぶ。しかし、家庭用の計算資源では限界がある。そのため、例えば固有名詞をあらかじめ代替トークンで置き換えてから誤字検出を行ったり、誤字検出後に特定の語彙は誤字判定を強制的に打ち消すなどの処理を行うことが考えられる。しか

し、これらは固有名詞と誤字が明確に分けられる場合にしか用いることが出来ず、固有名詞そのものに誤りが含まれていた時は対応することはできなくなる。

4.3.2 青空文庫モデルについて

青空文庫モデルの事前学習のデータセットは青空文庫のみであることから、実験前は誤字検出ではそこまで精度が高くないと考えていたが実際は東北大学モデルよりも良いという結果になった。これは、青空文庫モデルはサブワード分割ではなく単文字分割であるからではないかと考えた。トーカナイザが誤字を含めた単語をトークンとして生成すると、BERT はその前後の単語で判断する。これが品詞の誤りのとき前後の単語の接続のみが自然だとしても正答として処理されることがある。

単文字分割だと単純に判断できる数が増えるため精度が高いと思われる。反対に、文字毎であるため固有名詞や頻出する語群では正しい判断が行えないデメリットもある。もちろんデータセットの語彙の偏りによる差異も含まれると思われるがトーカナイザの違いによる影響は大きい可能性がある。

また、RoBERTa と BERT による違いは実験結果ではよくわからなかった。しかし、東北大学モデルや自作モデルに比べ青空文庫モデルは GPU のメモリ枯渇による強制停止の回数は少なかった。最適化によりメモリ使用量が減少している可能性がある。

4.3.3 東北大学モデルについて

東北大学モデルの精度は想定よりも高くなかった。誤字検出においてはデータセット内に含まれる誤字を参照してしまっていることや、100 単語という閾値が低かった可能性がある。ファインチューニングの入力データや量次第で向上する可能性もある。

正しい文の正解率も低いが、他のモデルと同様に固有名詞やトーカナイザによる影響が大きいと考えられる。

4.3.4 判断基準について

今回の実験により、100 単語を生成するという誤字の判断方式は誤字検出にとっては基準が低く、正しい文の検出にとっては基準が高かった場面が見られた。これは誤字の種類にも品詞の違いにも影響されるものであるため、今回の誤字認識に適合した基準が存在すると考えられる。特に助詞としてトークン化された場合は誤字としてのパターン

が少ないため、単語数による判断は難しい場合もあり、例えば品詞ごとに閾値を割り当てたり、BERT による文すべての再生成を行うなどの処理が考えられる。

また、専門的な単語の仕様が予想されるときには特定の単語は機械的に誤字訂正のスルーを行ったり、代替トークンに置き換えたりする機構をつけるなどの工夫で、ケースによるが精度の改善を行える可能性もある。

4.3.5 曖昧さの検出

今回はデータセットを用いたため、誤字は明確に間違っているというラベル付けを行えるが、実際には正しい文は曖昧さや多義的な解釈によって明確に正しいという判断を行えない場合がある。同義語や類義語との関連度を以て曖昧さを数値化するという方法で誤字を検出するなどの精度を向上させることとは異なるアプローチでの手法も検討の余地がある。

第5章　結論

本研究では、誤字検出を少ないリソースで事前学習したBERTと一般に配布されているBERTを実行し、それぞれの正解率を比較した。

データセットはWikipediaをスクレイピングしたデータと入力誤りデータセットを事前学習で、入力誤りデータセットのみを実験で行いることとした。BERTの事前学習のために取得したデータセットからコーパスを作成し、まずはトーカナイザを学習、続いてBERT本体を事前学習した。一般公開されている2種類のモデルを取得し、全てのBERTに同じ条件でファインチューニングを行った。

実験を行うため3つのモデルに誤字の含まれる文と正しい文を入力し、誤字の含まれる文か正しい文かを判断する実験を行った。

結果は自作モデルは多くの正しい文章を誤字の含まれる文であると判断してしまっていた。全体の正解率は半分程度だったが、誤字検出は高い正解率を示した。誤字とする基準や品詞毎の処理は検討の余地があり、正しい文章に対する精度はさらに上げることが出来る可能性がある。

また公開されている2つのモデルも全体の正解率は半分程度で、正しい文章の正解率は高くても4割程度であるため、正しい文章を正しいと判断するには更なる工夫が必要である。

謝辞

最後に、本研究を進めるにあたり、ご多忙中にも関わらず多大なご指導をしていただきました出口利憲先生、また、共に勉学に励んだ同研究室のメンバーに厚く御礼申し上げます。

参考文献

1) 中村明裕, “ 頭が赤い魚を食べる猫 ”, Twitter, 2020, 2024/2/21.
https://x.com/nakamurakihiro/status/1230798247989366784

2) Katsuhiro Sudoh, “Advances of Neural Machine Translation”, 2019, 2024/2/21.
https://www.jstage.jst.go.jp/article/jjsai/34/4/34_437/_pdf/-char/ja

3) Vaswani, A. et al. “Attention Is All You Need” , 2017, 2024/2/21.
https://arxiv.org/abs/1706.03762

4) Devlin, J. et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, 2018, 2024/2/21.
https://arxiv.org/abs/1810.04805

5) 京都大学 言語メディア研究室, “ 日本語 Wikipedia 入力誤りデータセット ”, 京都大学 言語メディア研究室 ホームページ, 2024, 2024/2/21.
https://nlp.ist.i.kyoto-u.ac.jp/?日本語 Wikipedia 入力誤りデータセット

6) 東北大学 自然言語処理研究グループ, “Pretrained Japanese BERT models”, Github, 2024, 2024/2/21.
https://github.com/cl-tohoku/bert-japanese

7) Facebook, “CC-100: Monolingual Datasets from Web Crawl Data”, data.statmt.org, 2024, 2024/2/21.
https://data.statmt.org/cc-100/

8) 安岡孝一, “KoichiYasuoka/roberta-base-japanese-aozora-char”, Hugging face, 2024, 2024/2/21.
https://huggingface.co/KoichiYasuoka/roberta-base-japanese-aozora-char