令和 3 年度茨城大学大学院理工学研究科情報工学専攻

修士学位論文

BERT の領域適応における複合語の語彙拡張

所属　情報工学専攻

著者　田中裕隆 (20NM716Y)

指導教員　新納浩幸教授

令和 4 年 2 月 2 日 (水)

令和 3 年度茨城大学大学院理工学研究科情報工学専攻　修士学位論文

BERT の領域適応における複合語の語彙拡張

著者

田中裕隆 (20NM716Y)

指導教員

新納浩幸教授

論文要旨

近年，事前学習モデルの適用によって機械学習システムの性能が大きく向上している．BERT は，自然言語処理タスクを処理できる事前学習モデルの一つである．BERT のような事前学習モデルは，大規模コーパスで事前学習し応用タスクデータで finetuning することで，様々な自然言語処理タスクを高精度に処理できる．

BERT への入力文は，日本語の場合，形態素解析と BPE によって Token 単位に分割される．分割された Token は，BERT モデルの各々の対応する埋め込み表現に変換され，後段の Transformer 層に入力される．Transformer 層は，Multi-head Attention モデルが核となり，Token 間の関係を埋め込んだベクトルを出力する．このような BERT モデルを，Masked Language Model と Next Sentence Prediction の二つの手法によって事前学習する．この事前学習によって得られた事前学習済み BERT モデルは，文脈に応じた単語埋め込み表現を出力することができる．

しかし，BERT のような事前学習モデルは，Token レベルで処理するために，BPE によって分割される未知語や複数の語で構成される表現を学習することは困難である．このような表現を学習するには，一つの Token として埋め込み表現を学習し，BERT の語彙を拡張すればよい．本研究では，そのような複数 Token で表現される語を対象にした BERT の語彙拡張における，追加語彙の埋め込み表現の構築について考える．提案手法では，追加の事前学習による領域適応を前提に，類義語による近似的な埋め込み表現で語彙追加を行った．

実験では，Amazon レビューコーパスを用いて，事前学習コーパスである Wikipedia での出現頻度が低く，Amazon レビューコーパスに複数回出現する語の内 20 語を対象として語彙拡張を行った．提案手法に加えて，静的な単語埋め込み表現を利用した手法と，subword の平均ベクトルの手法，BERTRAM の手法についてそれぞれ語彙拡張の実験を行い，Masked LM による追加語彙の予測精度で評価した．考察では，実験結果と各語彙の予測精度から傾向を分析した．

Master's Thesis in Scholastic 2021,
Major in Computer and Information Sciences,
Granduate School of Science and Engineering, Ibaraki University

Vocabulary Expansion of Compound Words in BERT's Domain Adaptation

Author : Hirotaka Tanaka (20NM716Y)

Adviser : Prof. Hiroyuki Shinnou

Abstract

In recent years, the pre-training models has greatly improved the performance of machine learning systems. BERT, a pre-training model, can process a variety of natural language processing tasks with high performance by pre-training on a large corpus and finetuning with downstream task data.

Japanese sentences as input to BERT are split into Token units by morphological analysis and BPE. The Tokens are each converted to the corresponding token embeddings and input to the following Transformer layers. The core of the Transformer layer is the Multi-head Attention model. It outputs vectors that embed the relationships between tokens. The model is pre-trained by two unsupervised tasks: Masked Language Model and Next Sentence Prediction. The pre-trained model outputs contextualized word representations.

However, since BERT processes at the Token level, it is difficult to learn unknown words that are split by the BPE or representations that consist of multiple words. Training such a representation can be solved by expanding the BERT vocabulary with a token embedding that is trained as a single token. The purpose of this paper is to establish a method for constructing token embeddings of additional vocabulary in the vocabulary expansion of BERT for such multiple words. In the proposed method, the token embedding of synonyms is added to BERT as an approximate token embedding of the additional vocabulary, assuming domain adaptation with additional pre-training.

In the experiment, we expanded the vocabulary of 20 words that appear frequently in the Amazon review corpus and infrequently in the Wikipedia pre-training corpus. In addition to the proposed method, we conducted vocabulary expansion experiments on the static word representation method, the mean vector of subwords method, and the BERTRAM method, respectively, and evaluated them by the prediction accuracy of the added vocabulary by MLM.

目次

第 1 章

序論

事前学習モデルは，多くの自然言語処理システムの性能を向上させている [1] [2]．事前学習モデルの一つである BERT [3] は，Transformer [4] モデルの Multi-head Attention を多層に積み重ねたモデルであり，入力単語列に対する文脈を考慮した単語埋め込み表現を出力することができる．

事前学習モデルは，大規模コーパスで事前学習したモデルを応用タスクデータに適用することで，様々な応用タスクを高精度に処理する．その性質上，応用タスクのドメインが事前学習したコーパスのドメインと大きく異なると，応用タスクを解く精度が高くならない領域適応の問題が存在する．Gururangan ら [5] は，事前学習モデルを領域適応させる手法として，Domain-Adaptive Pretraining（DAPT）と Task-Adaptive Pretraining（TAPT）の二つを提案している．この二つの手法に共通した重要な点は，応用タスクドメインのコーパスで追加の事前学習を行うことである．DAPT は応用タスクドメインに広く関連した教師なしの大きなコーパスを用いて学習を行うのに対し，TAPT は DAPT と比較してタスクにより関連した，より小さなコーパスで学習を行う．Gururangan らは RoBERTa で実験と評価を行っているが，BERT であれば応用タスクドメインのコーパスに対して Masked Language Model による追加事前学習をすればよい．

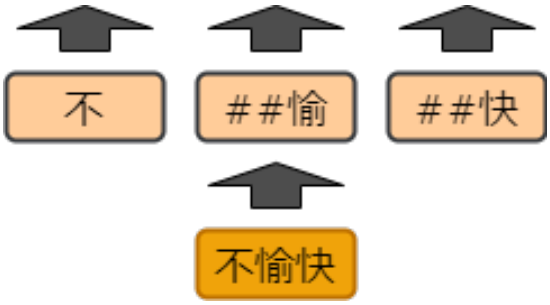
事前学習モデルの領域適応の問題は，語彙にも現れる．事前学習モデルの扱う語彙は，事前学習したコーパスに依存して決定される．標準的な BERT モデルの語彙は，事前学習コーパスに頻出する語彙から決定する．そのため，応用タスクデータに固有の語彙については，事前学習モデルが扱うことができず，適切な学習も行えない．応用タスクデータに現れる語彙に適応するためには，語彙を拡張する必要がある．事前学習モデルの領域適応における語彙の拡張手法については，Yao らの Adapt-and-Distill [6] や，Hong ら

の AVocaDo [7]，BERTRAM [8] といった研究がある．

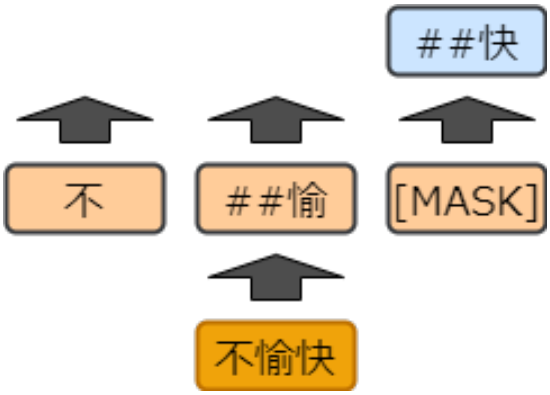
未知語については，上記のように基本的には高い精度で処理することができないが，標準的な BERT では可能な限り未知語として処理することはしない．BERT への入力文は，形態素解析と Byte-Pair Encoding によって分割された Token 単位で処理される．これにより，未知語が既知の複数の語に分割されて処理されることで，少ない語彙数で多くの単語を処理できる．しかし BERT のような事前学習モデルは，Token レベルの表現を学習するために，複数の語で構成される複合語や固有表現，句の表現を学習することは困難である．(図 1.1 参照) 例えば，「不愉快」という単語について処理するとき，「不」「##愉」「##快」の 3 つの Token に分割され，それぞれ Token レベルで処理が行われる．このとき，BERT モデルは Token 単位の予測については高精度に行うことができるため，3Token の内一つの Token が MASK されたとしても，予測が可能である．しかし，「不」「##愉」「##快」の 3Token 全てを MASK した場合，BERT モデルは「不愉快」としての表現を学習していないため，1 つの Token を予測する場合と比べて予測が困難になる．このような表現を一つの Token として扱えば，適切な予測が可能になる．

複数の語で構成されるエンティティの埋め込み表現を学習するための事前学習モデルとしては，LUKE [9] がある．LUKE は，通常の Token の埋め込み表現とは別にエンティティのための埋め込み表現を用意し，通常の Token とエンティティ間の関係を Entity-aware Self-attention によってモデル化・学習する．ただし，既存の事前学習モデルをエンティティに適応する手法ではなく，LUKE はエンティティを学習可能な新たな事前学習モデルであるため，大規模コーパスによる高コストな事前学習を要する．

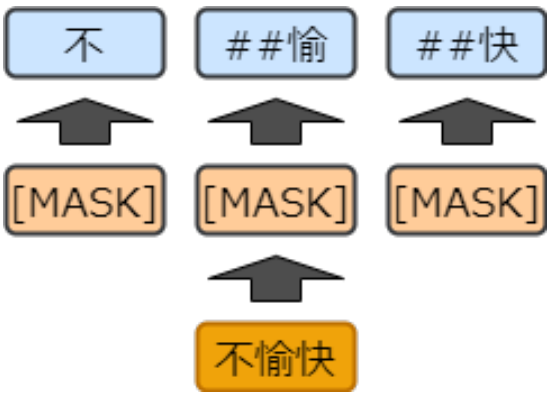
本研究では，事前学習済み BERT モデルにおいて複数の Token で表現される語を対象にした語彙拡張を行う際の，追加語彙の埋め込み表現の構築手法について提案する．語彙拡張では，追加語彙の埋め込み表現を得ることができればよい．応用タスクデータによって追加の事前学習を行い，事前学習済みモデルを領域適応させることを前提とすれば，その前段階で追加語彙の近似的な埋め込み表現を追加しても，適切な学習が可能であると考える．



(a) 複数 Token に分割され，Token レベルで処理



(b) Token 単位の予測は高精度に行える



(c) 分割された表現全体を予測することは困難

(d) 単体 Token の処理であれば，Token 単位の予測は容易

図 1.1: 語彙追加の対象となる語の例

第 2 章

関連研究

2.1 NLP における機械学習手法と単語埋め込み表現

2.1.1 SVM と RNN

自然言語処理（Natural Language Processing, NLP）においては，従来より機械学習を適用したタスク処理システムが構築されてきた．例えば，NLP タスクの一つである文書分類に適用される機械学習手法として，サポートベクターマシン（SVM）が広く知られている．SVM は，カーネルトリックによって非線形問題を処理できることが特徴の機械学習手法である．

また，文章を単語や文字の時系列情報として扱うことを考えれば，RNN を適用できる．RNN は再帰的なニューラルネットワークであり，時系列情報を処理できる機械学習モデルである．例えば，入力 x_0 に対して次の系列情報である x_1 を予測するように事後確率を学習することができる．自然言語処理においては，単語列 $w_t(t \in \{0, 1, 2, \ldots, n\})$ が与えられたとき，単語列の事後確率から言語を次のようにモデル化する．

$$P(w_0, w_1, w_2, \ldots, w_n) = P(w_0)P(w_1|w_0)\cdots P(w_n|w_0, w_1, \cdots, w_{n-1}) \tag{2.1}$$

$$logP(\mathrm{w}) = \sum_{t=0}^{n} logP(w_t|\mathrm{w}_{<t}) \tag{2.2}$$

RNN はこのような言語モデルを学習することができる．現実的な応用タスクに適用する RNN モデルとしては，その発展形である LSTM が広く知られている．LSTM は高い表現力を持つ代わりに処理速度が遅い特徴がある．これに対して，表現力よりも処理速度に優れた RNN モデルとして GRU がある．

2.1.2　BoW と TF-IDF

一方で，自然言語を機械学習モデルで扱う場合，モデルに対してどのような素性を入力とするかは常に課題となる．画像や音声などの観測値と違い記号列である自然言語は，これを計算可能なベクトルに変換する必要がある．最も基本的な方法として Bag of Words (BoW) がある．BoW は，文書中に出現する単語の頻度を素性とするベクトルである．文書中に頻出する単語がその文書を特徴として表現される．しかしこの方法では，多くの文書で頻出してしまい，文書を特徴づける単語としてふさわしくない語も重要な単語として表現されてしまう問題がある．これに対応した表現として TF-IDF がある．TF は Token frequency であり，BoW と同様，単語の頻度である．IDF は Inverse document frequency であり，日本語では逆文書頻度と言われる．これは，全文書中にある単語を含む文書の頻度の逆数を取ったものであり，多くの文書に出現するほど低い値をとる．これによって，多くの文書に出現し文書を特徴づける単語として重要ではない語に対しては，その重要度を下げるように表現される．文書 d における単語 t の出現数を $n_{t,d}$，文書 d における全ての単語 $t' \in d$ の出現数を $\sum_{t' \in d} n_{t',d}$，全文書数を $|D|$，単語 t を含む文書数を $|\{d \in D : d \in t\}|$ とするとき，TF-IDF は次式で表せる．

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{2.3}$$

$$tf(t, d) = \frac{n_{t,d}}{\sum^{t' \in d} n_{t',d}} \tag{2.4}$$

$$idf(t, D) = \log \frac{|D|}{|\{d \in D : d \in t\}|} \tag{2.5}$$

なお，実用的には 0 除算を避けた実装をする必要がある．上記の TF-IDF の式では文書中に存在しない単語 t について計算する場合，idf 値の分母は 0 となり 0 除算が発生する．また，全ての文書に出現する単語の場合，idf 値が 0 になり tf 値が考慮できなくなる．Python の機械学習ライブラリである scikit-learn では，これらの問題に対応して idf 値を次式のように定義している．

$$idf(t, D) = \log \frac{1 + |D|}{1 + |\{d \in D : d \in t\}|} + 1 \tag{2.6}$$

BoW や TF-IDF は文書を表現するベクトルである．これに対して単語をベクトルで表す手法としては，one-hot 表現がある．これは，扱う語彙をベクトルの各要素に割り当て，単語に対応する次元の要素のみ 1，それ以外の次元の要素を 0 とする表現である．

BoW や TF-IDF，one-hot 表現によるベクトルは，単語数の数だけ次元数が大きい疎ベクトルとなる特徴がある．高次元ベクトルを対象とした機械学習では，次元の呪いによって学習が困難になり，適切に処理することができない．

2.1.3　Word2Vec

高次元の疎ベクトルによる次元の呪いの問題に対処するためには，低次元な密ベクトルによって自然言語を表す必要がある．そのようなベクトル表現として Word2Vec のような分散表現がよく知られている．分散表現は単語をベクトル空間に埋め込んだ表現である観点から，単語埋め込み表現とも呼ばれる．分散表現は Word2Vec の他にも，fastText や GloVe などがある．

Word2Vec は，単語の意味はその単語の周囲に分布する単語によって決定されるという分布仮説に基づくモデルである．Word2Vec 以外にも，単語を低次元の密ベクトルで表す単語埋め込み表現の多くは，この分布仮説に基づいている．Word2Vec による分散表現は，ニューラルネットワークで定義されたモデルを学習することで得ることができる．Word2Vec は，skip-gram と CBoW（Continuous Bag-of-Words）の 2 種類のモデルが用いられる．skip-gram は，単語列のある単語を入力としたとき，その前後数単語を予測するように学習を行う．これに対して CBoW は，skip-gram と入出力関係が逆となる．前後数単語から，ターゲットとなる単語を予測するように学習する．いずれもニューラルネットワークの構成は入力から中間層へ出力する全結合層と，中間層から主力層への全結合層からなる．入力は単語の one-hot 表現で与えられる．出力は Softmax 関数を通して各単語の確率的なスコアとして出力される．学習時は，モデルの出力と one-hot 表現で与えられる正解ラベルとの交差エントロピー損失を最小化するように学習する．このようにして得られた学習済みモデルの出力する各単語に対応する中間表現が，単語の分散表現となる．CBoW と比較して skip-gram の方が頻出単語予測や類推問題において高い精度で予測をすることができる．しかし，skip-gram は計算コストの高い softmax 関数の処理が多いため，CBoW の方が高速に学習を行うことができる利点がある．

Word2Vec による分散表現の特徴は加法構成性にある．例えば，「日本」と「首都」の分散表現を加算して得られる分散表現は，「東京」の分散表現と高い類似性を持つ．したがって，Word2Vec による分散表現は単語間の関係を表現できる能力を持つ．

Word2Vec の学習データは，単語の一般的な意味表現を埋め込むために，大規模コーパ

スで学習を行う．そのようにして得られた学習済みの分散表現を，LSTM で構成されるような機械学習モデルの入力として用いて，ターゲットとなる応用タスクデータによってモデルを学習する．このような学習は転移学習の一つである．このように，応用タスクデータの学習の前段階で大規模コーパスによる事前学習を行い，応用タスクに適用するようなモデルを，事前学習モデルと呼ぶ．Word2Vec はこの観点で事前学習モデルと呼ぶことができる．

分散表現は，それぞれの単語に対して一対一で対応する単一の意味表現を学習する．しかし実際の単語は，同じ表現で複数の語義を表す．例えば「犬」という単語であれば，動物の犬の場合の他に，スパイの意味も存在する．このような複数の語義を持つ単語に対して，Word2Vec のようなモデルでは表現できない．複数の語彙を持つ単語を処理するタスクは，語義曖昧性解消と呼ばれる．

2.1.4　ELMo

語義曖昧性解消に対処して単語埋め込み表現を獲得する手法として ELMo [1] がある．ELMo は，単語に対して一対一で対応する意味表現を埋め込むのではなく，その単語が出現する文脈に依存して変化する埋め込み表現を出力する．このような埋め込み表現は文脈化単語埋め込み表現 (contextualized word representation) と呼ばれる．文脈化単語埋め込み表現に対して，従来の Word2Vec のような埋め込み表現は静的な単語埋め込み表現と呼ばれる．

ELMo は，双方向の LSTM モデルによって学習する．LSTM は，時系列情報を逐次処理する RNN モデルの特性から，系列データを単方向の処理しか行うことができない．自然言語は分布仮説の観点からも，ある単語に対してはその前後の単語分布情報が重要になる．単方向の処理のみ行う LSTM では，ある単語の予測に対してその前方の単語分布情報しか入力することができない．後方の単語分布情報を埋め込むには，単方向の LSTM に加えて，その逆方向の処理を行う LSTM を追加すればよい．このように，二つの LSTM を用いて双方向の処理を行うモデルは Bi-directional LSTM と呼ばれる．ELMo は，この Bi-directional LSTM を多層に積み重ねたモデルである．

ELMo による単語埋め込み表現は，多層に積み重ねられた Bi-directional LSTM の各層の出力の重みづけ和を取ったものを用いる．

ELMo の学習は大規模コーパスによって行う事前学習モデルの一つである．

ELMo による単語埋め込み表現を応用タスクに適用する際は，従来の静的な単語埋め込み表現に結合して用いる．このように，事前学習したベクトルをモデルへの入力ベクトルの特徴量として追加する手法は，feature-based approach と呼ぶ.

2.1.5　Transformer と Attention

これまで主流であった RNN ベースのモデルに代わり，Attention ベースのモデルが注目を集めたきっかけの一つは，Transformer モデルの提案である．Transformer は，Attention モデルベースの機械翻訳のためのモデルである．従来の機械翻訳や多くの自然言語処理タスクに適用されたディープラーニングモデルは，RNN もしくは CNN が用いられてきた．しかし，Transformer モデルはそのどちらも用いずに，Attention モデルを中心にモデルを構成している.

Attention モデル自体は，Transformer 以前から自然言語処理タスクに適用されている．Attention モデルの考え方は，ある検索 Query を与えるとき，Query に最も一致する Key を求め，この Key に対応する Value を出力する検索システムである.

Transformer 以前の自然言語処理分野では，Attention モデルを Seq2Seq モデルに組み込んだモデル [10] がよく知られている．Seq2Seq [11] は系列データを入力として系列データを出力するモデルであり，自然言語処理の機械翻訳タスクに適用できる．Seq2Seq は Encoder-Decoder モデルとして構成されており，モデルの Encoder 部分で入力となる系列データを埋め込み，Encoder で埋め込んだベクトルから Decoder 部分で出力系列を生成する．その Encoder-Decoder モデルの実体は LSTM で実装される．入力単語列を逐次 LSTM の入力とし，入力後は入力単語列の終端<EOS>を表すベクトルを入力する．<EOS>ベクトル入力後の LSTM は Decoder として機能し，LSTM の出力を次の LSTM の入力とすることで出力単語列を得ることができる．LSTM モデルで実装された Seq2Seq の問題点は，長い系列データを LSTM に入力する場合，LSTM が情報を保持できず，入力データに対応した適切な出力が困難になることである．このような問題に対処して，Attention モデルを組み込んだ Seq2Seq は，入力系列情報をより Decoder に入力するように構成される．従来の Seq2Seq の実体は純粋な LSTM モデルであり，Encoder から Decoder に渡されるデータは，Encoder の最終系列に対する出力のみである．これに対して，Attention モデルを組み込んだ Seq2Seq では系列の全ての入力に対する LSTM の出力を Encoder から Decoder に与える．Decoder は，通常の LSTM の

出力と Encoder の出力を Attention モデルに入力する．LSTM の出力は Attention の Query として与えられ，Encoder の出力は Key, Value として与えられる．これにより，Attention モデルは Decoder の LSTM の出力クエリとの関連度に応じた Encoder の出力を得る．Decoder の最終的な出力は，LSTM の出力と Attention モデルの出力とを結合したベクトルを入力とした全結合層の出力となる．

Transformer モデルの Attention モデルは, Seq2Seq で適用された Attention とは違った方法で利用されている．Transformer における Attention は，Self-Attention と呼ばれる．Attention モデルの入力は，Query，Key，Value の 3 つあるが，この 3 つの入力全てに同一のベクトルを与える．これによって，同じ単語列において単語間の関係を埋め込んだ出力を得ることができる．また Transformer では，この Self-Attention を並列に計算する Multi-head Attention で構成される．

ここで, Multi-head Attention について数式で示す．Attention の Query, Key, Value の 3 つの入力について，それぞれ Q, K, V と表す．今，単語埋め込み表現が m 次元であったとする．Multi-head attention では m 次元ベクトルを $d_k(= m/k)$ 次元に圧縮する線形変換器を Q, K, V それぞれに対して用意する．Q, K, V の実体は $d_k \times d_k$ の線形変換行列である．Multi-head attention の入力は n 個の m 次元ベクトルであるが，これが先の圧縮機で $n \times d_k$ の行列 X に変換され，Q, K, V に渡され $n \times d_k$ の行列 XQ, XK, XV ができる．これらを Q', K', V' とおき，Scaled Dot-Product Attention により self attention を行う．

$$\mathtt{s}softmax\left(\frac{Q'K'^T}{\sqrt{d_k}}\right)V' \tag{2.7}$$

これは $n \times d_k$ の行列である．上記の処理を k 個並行して行うと，$n \times d_k$ の行列が k 個作成され，これらを結合して $n \times m$ の行列が作成できる．これを更に同次元に線形変換することで Multi-head attention の出力が作られる．

RNN モデルでは系列データを逐次処理するために，時間コストが大きかった．これに対して Self-Attention による Transformer モデルでは系列データを並列処理可能なため，高速な処理が可能である．また，Transformer モデルで用いる単語埋め込み表現は，従来のモデルと同様に Word2Vec などで事前学習を行い学習された分散表現を用いる．

2.2　BERT

BERT の特徴の一つは，事前学習済みの BERT モデルを finetuning するだけで，様々な自然言語処理タスクを高精度に処理できることである．ELMo は，事前学習したモデルの出力を Word2Vec のような分散表現と同様にタスク処理のための後段モデルへの入力として用いる feature-Based approach で利用される．これに対して BERT は，モデルの最終層にタスク処理用の全結合層を 1 層加えて，モデル全体をタスク処理のために学習すればよい．このような手法を finetuning approach と呼ぶ.

従来の自然言語処理タスクを解くディープラーニングモデルは，Word2Vec のような分散表現と，主に LSTM で構成されるタスク処理用のモデルの両方を用いる必要があった．BERT は，単語埋め込み表現とタスク処理の適用を一つのモデルで，End-to-End で学習し処理することができる.

2.2.1　モデル

BERT の出力の特徴として，文脈化単語埋め込み表現を出力できる点がある．文脈化単語埋め込み表現を出力するモデルとしては ELMo がある．ELMo との違いの一つは，ELMo が Bidirectional LSTM を重ねたモデルであることに対して，BERT モデルが Transformer の Multi-head Attention を重ねたモデルであることである．ELMo では，双方向の文脈を獲得するために，単方向の LSTM を 2 つ処理する必要があった．しかし，Multi-head Attention では双方向の文脈を一度の処理で獲得することができる．双方向の文脈を 1 つのモデルで表現できることから，2 つのモデルの組み合わせで表現する場合よりも高い表現力が得られる.

また Seq2Seq で示したように，ELMo で用いられている LSTM は単語間が長距離であるほどその間の関係を学習することが困難になる．Multi-head Attention を採用した BERT では，より広範囲の依存関係を埋め込むことができる.

2.2.2　事前学習

Multi-head Attention で双方向の文脈を考慮するためには損失関数 (目的関数) も工夫する必要がある．同様に Multi-head Attention で構成される Transformer や GPT-2 で

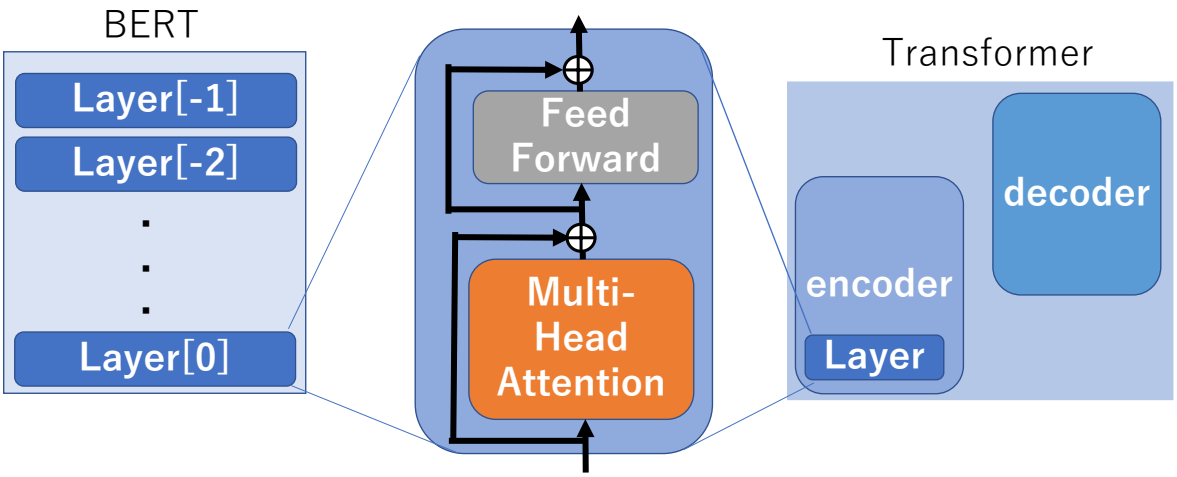


図 2.1: BERT のモデル図

は，一般的な言語モデルのように系列データである単語列を順に事後確率で求めていく観点から，ターゲットとなる単語の後方単語列はマスキングされ，双方向の文脈を埋め込むことはしていなかった．これに対して BERT は，Masked Language Model によって事前学習を行う．Masked Language Model の基本的な考え方は，入力単語列の単語をマスクしたり他の単語に置き換えたりすることで得られるノイズを含んだ単語列から，元の単語列を予測することである．一般的な Masked Language Model におけるノイズを含む単語列は，入力単語列の 15% に対して，次のような条件で生成する．

- 80% は，特殊トークンである `[MASK]` に置き換える．
- 10% は，ランダムな別のトークンに置き換える．
- 残り 10% は，そのまま残す．

入力単語列を $\bar{\mathrm{x}}$，マスクなどによってノイズを含んだ単語列を $\hat{\mathrm{x}}$ 単語 t に対するマスキングの有無を m_t で表すとき，Masked Language Model は次式のように表せる．

$$logP(\bar{\mathrm{x}}|\hat{\mathrm{x}}) \approx \sum_{t=1}^{T} m_t logP(x_t|\hat{\mathrm{x}}) \tag{2.8}$$

Masked Language Model が単語表現を事前学習することに加えて，BERT は 2 文を入力とした学習をするために，Next Sentence Prediction で事前学習を行う．Next Sentence Prediction は，2 つの入力文が連続しているか否かを予測する問題である．このタスクを処理するため，入力トークン列は図 2.2 のようになる．

このとき，特殊トークン `[CLS]` に対する埋め込み表現を後段の全結合層に入力し，入

図 2.2: BERT に 2 文入力する場合のトークン列

力の 2 文が連続した文であるかを予測する．

2.2.3　BERT の Input Embeddings

BERT への入力文は，形態素解析によって形態素に分割した後，Byte-Pair Encoding によってさらに分割された Token 列に変換される．これによって，できる限り既知の語に分割され未知語が発生しないように処理している．

BERT では，モデルの入力 Token 列の長さは固定される．これは，Self-Attention の際に入力長に合わせてパラメータを用意するためである．入力単語列の BERT の固定入力長への対応例を図 2.3 に示す．入力する単語列の長さがモデルで設定した最大入力長より短いとき，単語列の後ろを特殊トークンである `[PAD]` で埋める．入力する単語列の長さがモデルの最大入力長より長いとき，最大長より長い単語列の後ろの部分を切り捨てる．

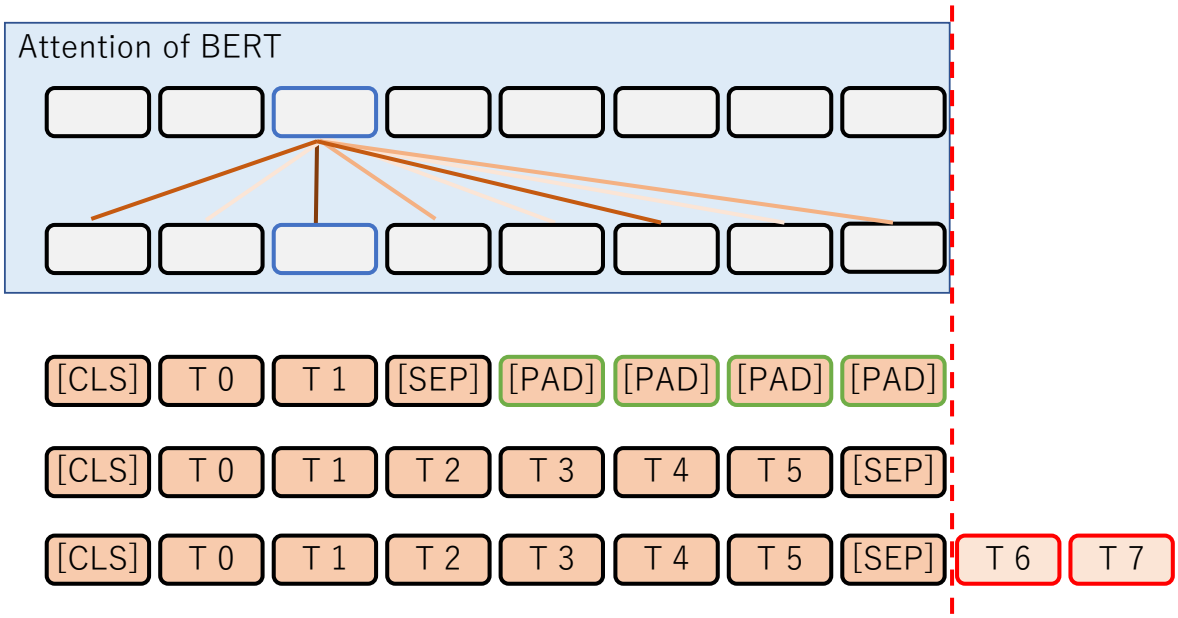


図 2.3: BERT の固定入力長への対応

入力文の Tokenize 処理によって得られた Token 列は，それぞれ BERT のモデルへ

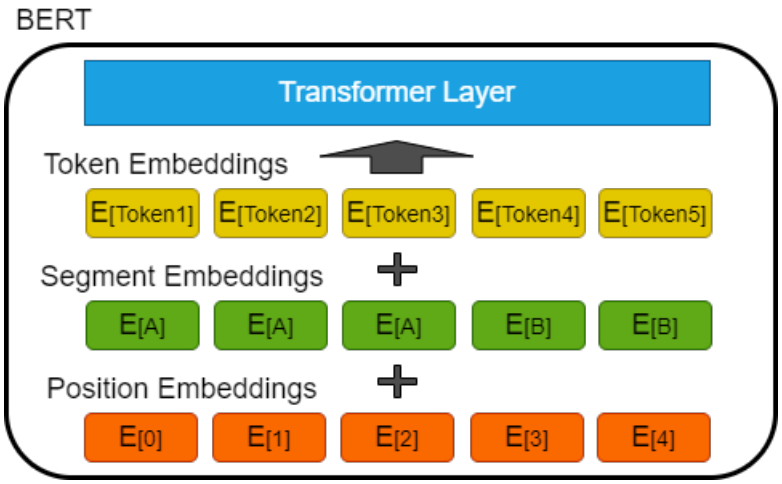


図 2.4: BERT の入力ベクトル

の入力のための埋め込み表現に変換される．BERT の入力となる埋め込み表現は Token Embeddings と Segment Embeddings，Position Embeddings の 3 種類ある．(図 2.4 参照)

Token Embeddings は，各々の単語 Token に対応した埋め込み表現である．BERT が最終的に出力する単語埋め込み表現は，文脈に応じて変化する文脈化単語埋め込み表現である．しかし，BERT への入力ベクトルとなる最初の埋め込み表現は，Token と一対一で対応した固有の埋め込み表現となる．Token Embeddings の語彙数は，形態素解析と BPE による Tokenize を行った Token を処理する場合，経験的に 32000 語程度に設定し学習する．

Segment Embeddings は，1 文目と 2 文目それぞれに対応した埋め込み表現である．BERT は 2 文の入力に対応しており，これら 2 文をそれぞれ識別するための埋め込み表現である．

Position Embeddings は，入力 Token 毎の位置情報を与える埋め込み表現である．BERT で文脈情報を処理する Attention 機構では，RNN と違い時系列に関する情報を持たない．そのため，位置に関する情報を入力の埋め込み表現で与える．

これら三つの埋め込み表現が，BERT 後段の Transformer 層への入力となる．

2.3　BERT への語彙追加手法

学習済み BERT モデルに語彙を追加する場合，BERT の Token Embeddings で用いられる埋め込み表現に追加語彙に対応する埋め込み表現を追加すればよい．したがって語彙追加手法は，BERT の埋め込み表現空間における追加語彙の埋め込み表現をどのように得るかが課題となる．

2.3.1　静的な単語埋め込み表現を利用した手法

追加語彙の埋め込み表現を得る方法の一つは，word2vec や fastText [12] [13] のような分散表現モデルによる静的な単語埋め込み表現を利用する方法である．追加する対象の単語を学習した分散表現モデルを得ることができれば，それを BERT の埋め込み表現に導入すればよい．追加する語彙を学習済みの分散表現モデルが存在しない場合でも，BERT モデルを一から事前学習するよりも，追加語彙を含んだ語彙集合で分散表現モデルを一から学習する方が少ない計算コストで作成することができる．

具体的には，まず追加語彙を学習した分散表現モデルを用意する．次に，BERT と分散表現モデルで共通して学習している語彙集合を基に，分散表現モデルの埋め込み表現を BERT の埋め込み表現に写像するための変換行列を学習する．このような学習は，Mikolov ら [14] の提案した手法を用いることができる．Mikolov らの手法では，変換のソース単語ベクトルとターゲット単語ベクトルの平均二乗誤差を小さくするように確率的勾配降下法で学習を行う．そのようにして得られた変換モデルを利用して，静的な単語埋め込み表現から BERT の埋め込み表現空間に写像した追加語彙の埋め込み表現を得る．

静的な単語埋め込み表現を利用する場合，変換モデルの設計が得られる埋め込み表現の品質を左右する．より高品質な変換モデルを作成するためには，分散表現モデルの埋め込み表現と BERT の埋め込み表現をそれぞれ単位ベクトルに正規化するなどの処理を要する．平子ら [15] は，静的な単語埋め込み表現を BERT の単語埋め込み空間へマッピングする際のベクトルの補正や正規化手法について提案している．

2.3.2　subword の平均ベクトル

事前学習済み BERT モデルのみを利用して追加語彙の埋め込み表現を得る手法としては，subword の平均ベクトルを用いる手法がある．この手法は Yao らの Adapt-and-Distill [6] においても用いられている．

追加の対象となる語彙は，事前学習済みモデルにおいて複数 Token で処理される．例えば「不愉快」という語であれば「不」「##愉」「##快」の 3Token である．事前学習済みモデルにはこれら 3Token の Token Embeddings が学習されている．事前学習済み BERT モデルからこれら 3Token の Token Embeddings の平均ベクトルを求め，それを追加語彙の埋め込み表現として追加する．

この手法では BERT の事前学習モデルのみを用いて追加語彙の埋め込み表現を得ることができる．また，subword のベクトルから構成するのみであり，追加語彙の埋め込み表現のための学習は必要ない．

2.4　BERTRAM

BERTRAM [8] は，学習済み BERT モデルの出力を利用して追加語彙の埋め込み表現を得る手法である．BERTRAM の研究では，事前学習済みの BERT が事前学習したコーパスにおける出現率が低頻度な語の追加を対象にしている．

BERTRAM では，追加語彙の埋め込み表現を Form-Context Model で学習する．Form-Context Model は，埋め込み表現を Form Model と Context Model から構成する．Form Model では文字ベースで単語の埋め込み表現を得る．入力単語は各々の文字に分割され，全ての文字 n-gram の平均ベクトルが，Form Model による埋め込み表現となる．文字 n-gram の埋め込み表現は Form-Context Model の学習によって獲得される．

Context Model では文脈から単語の埋め込み表現を得る．Context Model の構成は，BERTRAM の研究では SHALLOW と REPLACE，ADD の 3 つの手法が試みられている．SHALLOW(図 2.5 参照) では，文中の追加語彙を `[MASK]` トークンに置き換えて事前学習済み BERT に入力し，BERT が出力した `[MASK]` トークンに対応する文脈化単語埋め込み表現を Context Model の埋め込み表現とする．

REPLACE(図 2.6 参照) では，`[MASK]` トークンを使わずに，Form Model による埋

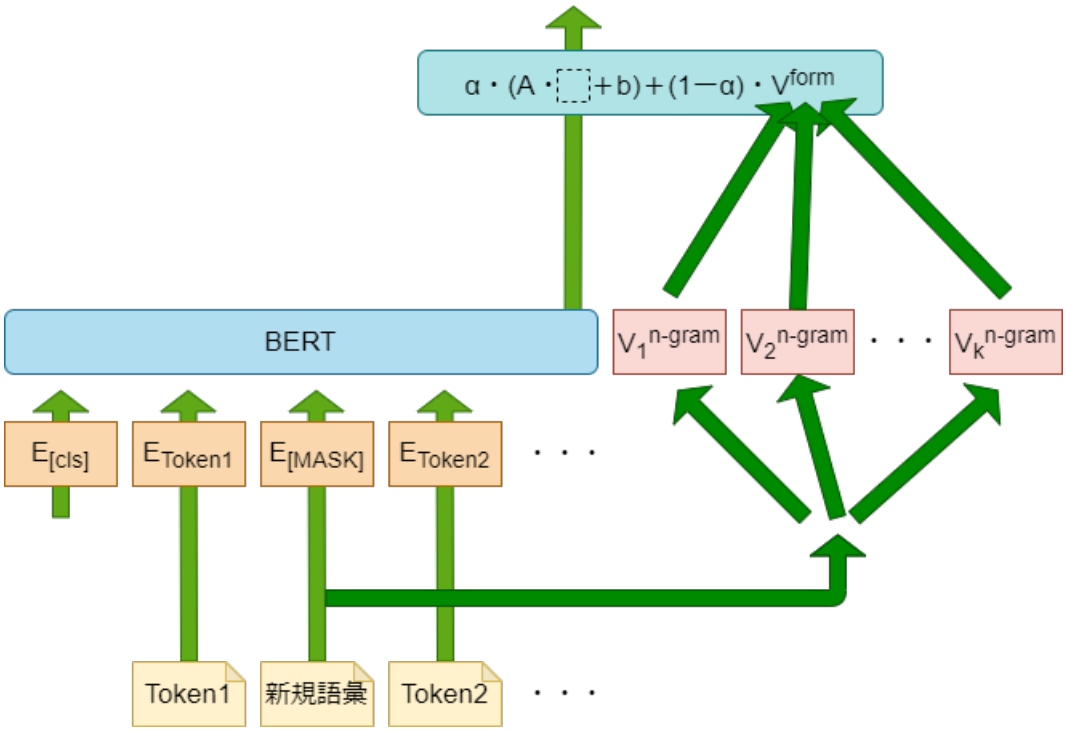


図 2.5: BERTRAM の Context Model 手法が SHALLOW の場合の処理

め込み表現で文中の追加語彙を置き換えて事前学習済み BERT に入力する．そして，BERT が出力した Form Model による埋め込み表現に対応する文脈化単語埋め込み表現を Context Model の埋め込み表現とする．

ADD(図 2.7 参照) の手法は，SHALLOW と REPLACE の手法を組み合わせた手法である．まず，文中の追加語彙を `[MASK]` トークンに置き換える．その後，文頭に Form Model による埋め込み表現と「」の単語列を追加する．このようにして得られた文を事前学習済み BERT に入力し，BERT が出力した `[MASK]` トークンに対応する文脈化単語埋め込み表現を Context Model の埋め込み表現とする．

Form Model の埋め込み表現と Context Model の埋め込み表現を組み合わせる手法は，Context Model の手法が SHALLOW である場合，式 (2.9) によって得られる．

$$v_{(w,C)} = \alpha \cdot (A \cdot v_{(w,C)}^{context} + b) + (1 - \alpha) \cdot v_{(w,C)}^{form} \tag{2.9}$$

Context Model の手法が REPLACE か ADD である場合は，Context Model の入力に Form Model による埋め込み表現が用いられているため，最終的な Form-Context Model の出力は Context Model の埋め込み表現を線形変換するのみである．(式 (2.10))

$$v_{(w,C)}^{new} = (A \cdot v_{(w,C)}^{context} + b) \tag{2.10}$$

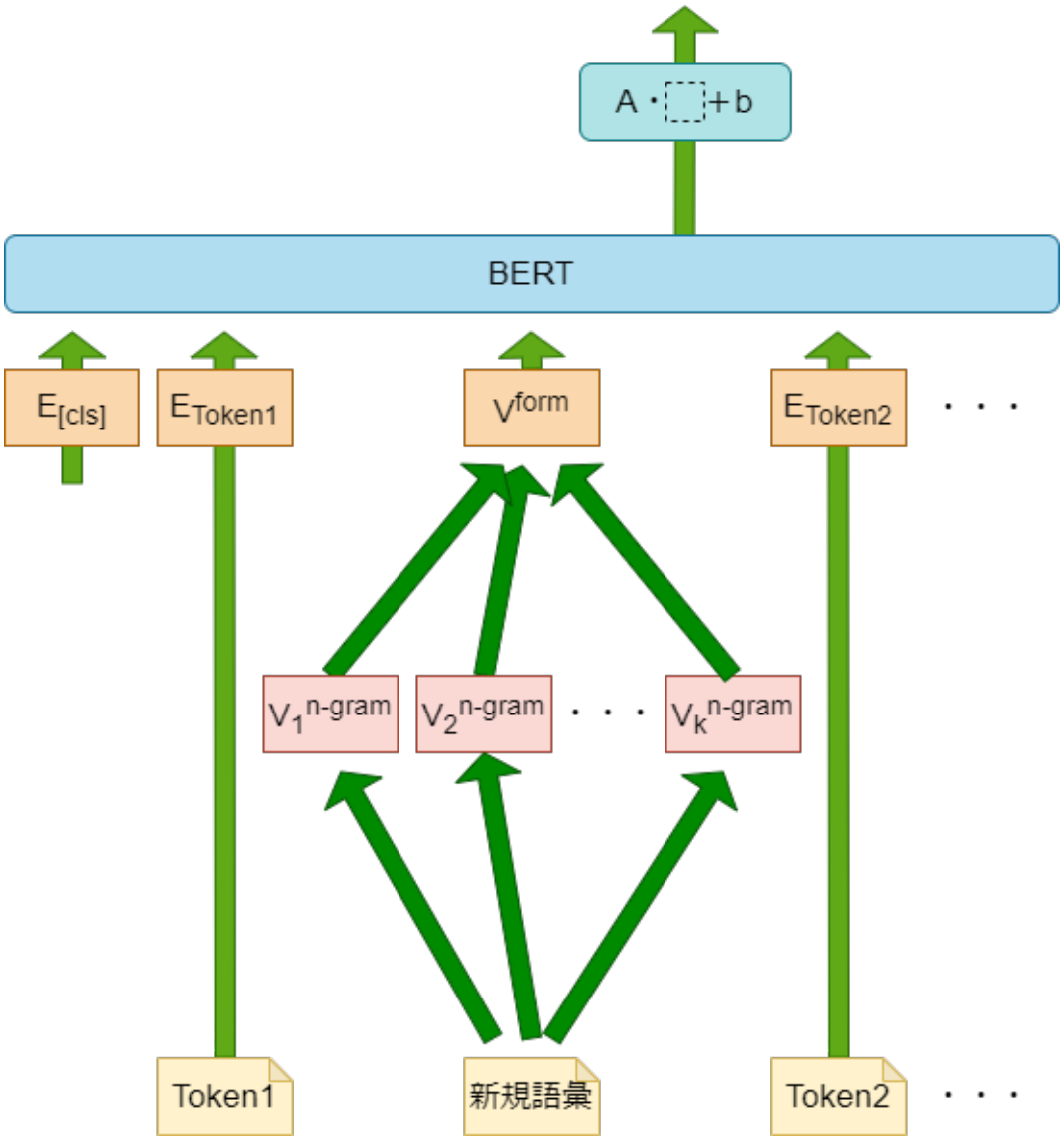


図 2.6: BERTRAM の Context Model 手法が REPLACE の場合の処理

学習時は，式 (2.11) に示す損失関数で損失が最小化するように学習する．e_w は単語 w に対応する BERT による既知の埋め込み表現であり，$v_{(w,C)}$ は，単語 w と文 C を入力とする Form-Context Model で構成された BERTRAM の出力である．

$$\left\| e_w - v_{(w,C)} \right\| \tag{2.11}$$

BERTRAM の学習は 3 ステップで行われる．Context Model の手法は ADD とする

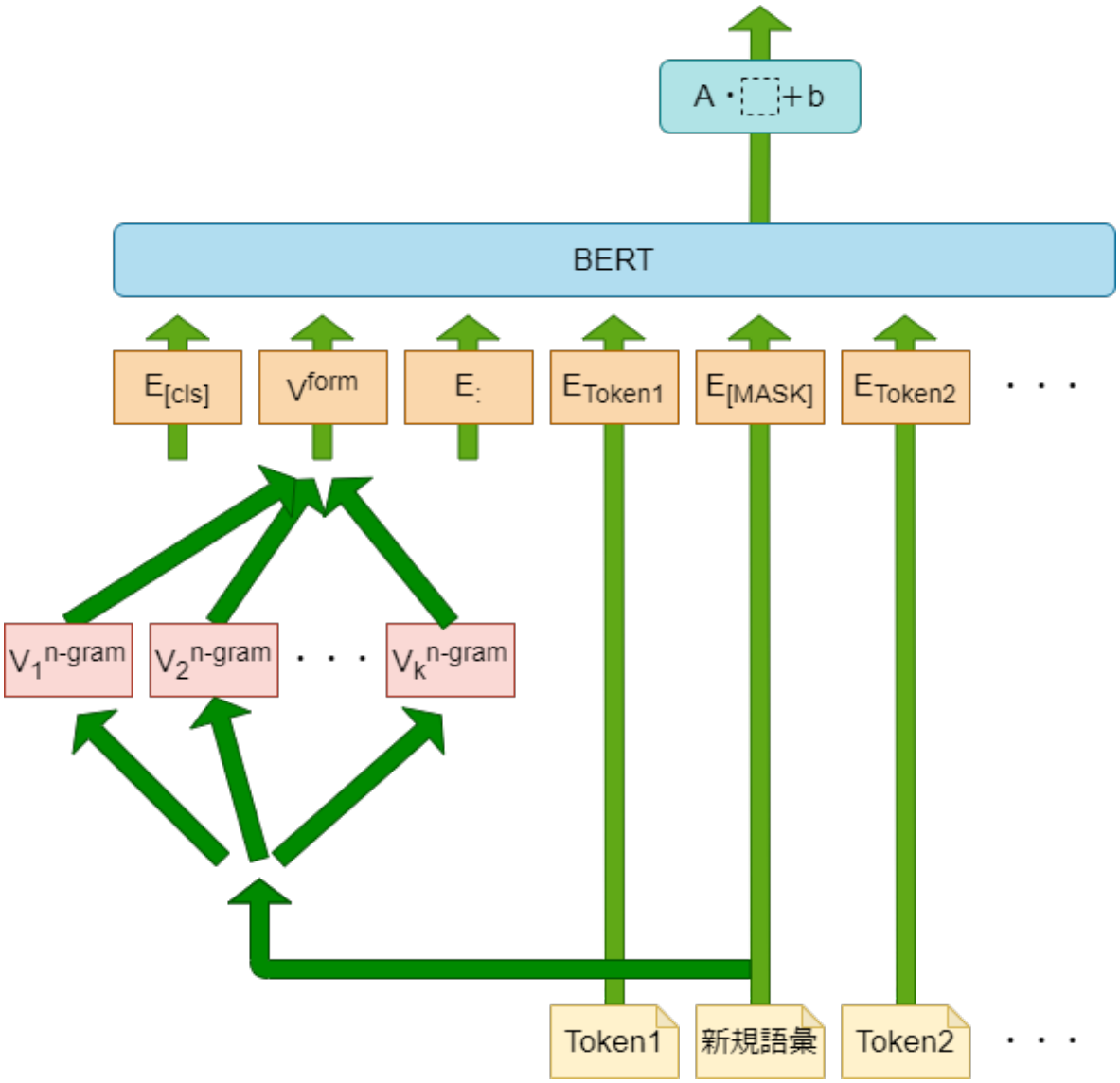


図 2.7: BERTRAM の Context Model 手法が ADD の場合の処理

とき，次の手順で学習する．

1. SHALLOW の手法によって Context Model による文脈ベクトル $v_{(w,C)}^{context}$ のみ学習する．
2. Form Model による語形ベクトルのみ学習する．
3. ADD の手法による文脈ベクトルの導出によって BERTRAM モデル全体のパラメータを学習する．

ADD および REPLACE の手法で BERTRAM の学習を行う際も，最初の手順で文脈ベクトルを学習する時は SHALLOW の手法で学習する．なお，学習の全ステップで学習

済み BERT モデルの学習は凍結される．

BERTRAM では，BERT に語彙を追加するとき，追加語彙「○○」を「<BERTRAM: ○○>」と表記した特殊トークンとして追加する．BERTRAM によって追加された埋め込み表現を利用する場合，データ中の追加語彙を「<BERTRAM: ○○>」の表記に置き換えて処理する．

第 3 章

提案手法

事前学習モデルは，応用タスクドメインに領域適応することによって応用タスクを高い精度で解くことができることが知られている．Gururangan らの手法によって，モデルが応用タスクドメインのデータで追加事前学習するとき，事前学習モデルの追加語彙の埋め込み表現は近似的な表現であっても追加事前学習によって適切な埋め込み表現を学習できると考えられる．

追加語彙の近似的な埋め込み表現は，追加語彙の類義語から得ることができる．また，追加語彙の類義語が事前学習済み BERT の語彙に含む語であれば，類義語の埋め込み表現は BERT が既に学習している．したがって本提案手法では，事前学習済み BERT の語彙に含む類義語の埋め込み表現を，追加語彙の埋め込み表現として BERT に追加する．そして，語彙拡張後の BERT を応用タスクデータで追加事前学習する．

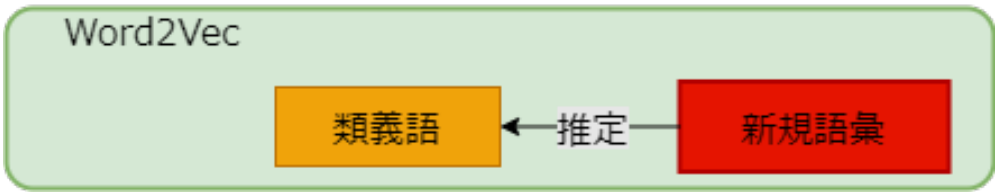
類義語の推定は Word2Vec のような静的な単語埋め込み表現を出力する分散表現モデルを用いる．分散表現モデルは，追加語彙と BERT の語彙の両方を学習したモデルを利用する．追加語彙を含めた語彙集合で BERT を一から事前学習するよりも，分散表現モデルを学習する方が低コストに学習できる．

語彙拡張した段階では類義語の埋め込み表現と追加語彙の埋め込み表現は同一である．しかし，追加事前学習によって BERT の Token Embeddings を含む学習パラメータ全体を学習すれば，それぞれ最適な埋め込み表現を学習できる．追加事前学習は応用タスクデータを学習データとして，Masked Language Model で BERT モデルの学習パラメータ全体を finetuning する．

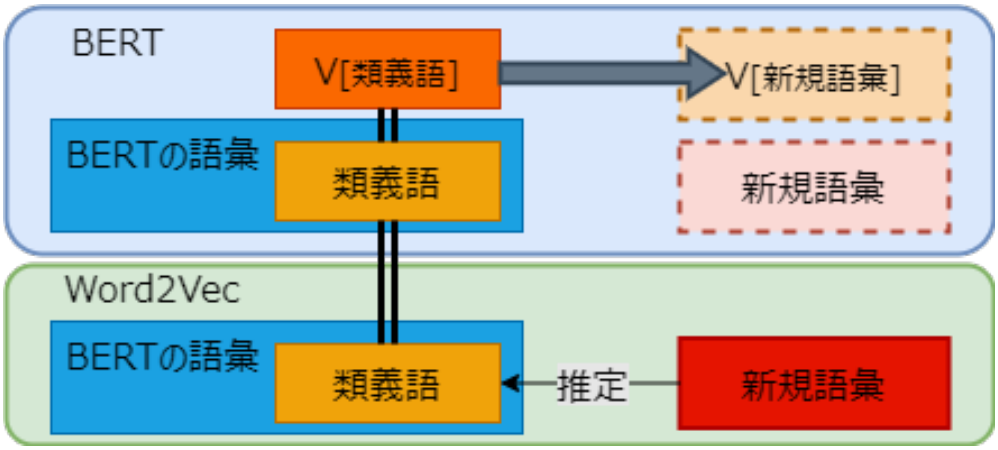
本提案手法 (図 3.1 参照) の具体的な手順は次のようになる．

1. 追加語彙を学習済みの Word2Vec のような分散表現モデルを用意する．
2. 追加語彙の類義語を，分散表現モデルで推定する．
3. 推定した類義語の内，BERT の語彙に含まれる語の BERT の埋め込み表現を，追加語彙の埋め込み表現として BERT に追加する．
4. 追加語彙を含むコーパスで語彙追加後の BERT を Masked Language Model で追加事前学習する．

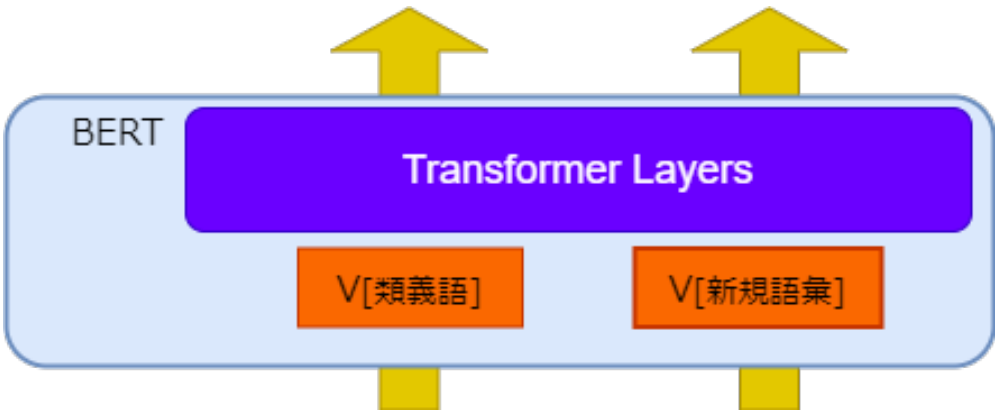
(a) 新規語彙を学習している分散表現モデルを用意



(b) 新規語彙の類義語を推定



(c) BERT の語彙に含まれる類義語に対する BERT の埋め込み表現を新規語彙の埋め込み表現に



(d) 埋め込み表現を含む BERT 全体を追加事前学習

図 3.1: 提案手法の概念図

第 4 章

実験

4.1 実験設定

本実験では，事前学習済みの日本語 BERT モデルを対象に語彙拡張を行う (図 4.1 参照)．提案手法の他に，静的な単語埋め込み表現を利用する手法と，subword の平均ベクトルの手法と，BERTRAM の手法を試みる．

提案手法において，追加語彙の類義語を求める時の類似度の計算は cos 類似度で求める．

全ての手法で，以下の設定で実験を行う．

- 全ての実験手法で形態素解析は Mecab で処理する．
- 各手法で語彙追加を行った BERT モデルは，いずれも応用タスクデータの学習データで追加の事前学習を行う．この追加事前学習は，BERT の事前学習で標準的に行われる Masked Language Model で行う．
- 評価は，新規語彙のみを MASK トークンに置き換えて，新規語彙を推定できる精度を測る．
- 追加の事前学習と評価のプロセスはランダムシードを変えて繰り返し 5 回行い，実験結果ではその平均値を示す．

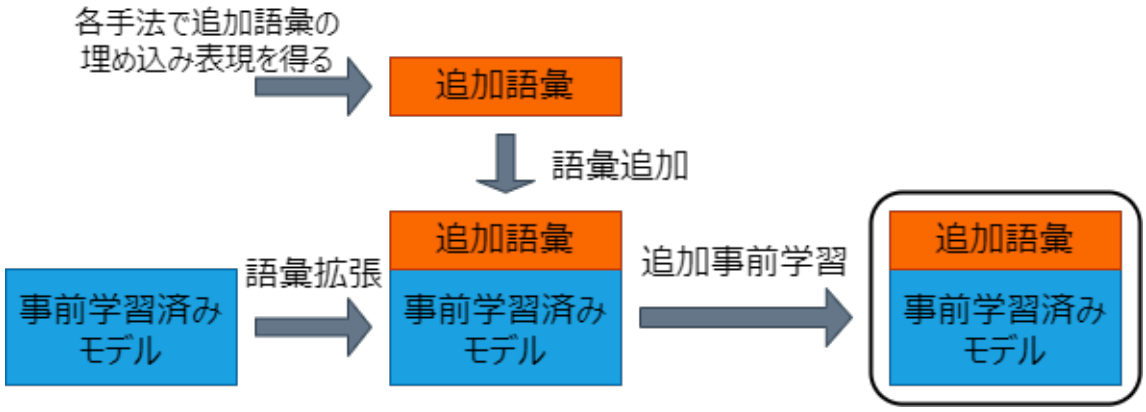


図 4.1: 語彙拡張の流れ

4.1.1　事前学習済み BERT モデル

BERT の事前学習済みモデルは，東北大学乾研究室が公開しているモデルを用いる．これは，Hugging Face 社の Transformers ライブラリ[*1] から，モデル名’cl-tohoku/bert-base-japanese’ で利用できるモデルである．事前学習コーパスには日本語 Wikipedia が用いられている．語彙数は 32000 語，モデルは Transformer 層が 12 層の BASE モデル，最大入力長は 512 である．

4.1.2　分散表現モデル

静的な単語埋め込み表現を出力する分散表現モデルには，以下のサイトで公開されている日本語 Wikipedia エンティティベクトル [16] を用いる．これは Word2Vec モデルで学習されている．この分散表現モデルは，提案手法と静的な単語埋め込み表現を利用する手法で利用する．

`http://www.cl.ecei.tohoku.ac.jp/~m-suzuki/jawiki_vector/`

4.1.3　BERTRAM

BERTRAM の学習は，以下の 2 つのサイトで公開されている BERTRAN の論文の著者実装プログラムを参考に，日本語 BERT に対応したプログラムを用いる．BERTRAM の学習データには，日本語 Wikipedia を用いる．

[*1] `https://github.com/huggingface/transformers`

https://github.com/timoschick/bertram

https://github.com/timoschick/form-context-model

4.1.4　データセット

本実験で学習・評価を行う応用タスクデータには，以下のサイトで公開されている Amazon レビューコーパスを用いる．

https://webis.de/data/webis-cls-10.html

このコーパスは，レビュー対象の商品の種類によってドメインが分けられている．このコーパスのドメインは，Books と DVD と Music の 3 つある．本実験では，Books のドメインのデータを用いる．Amazon レビューコーパスのデータセットは，複数の文からなるレビュー文書と，そのレビューで付けられた評価の星の数がラベル付けされている．本実験は Masked Language Model による教師なし学習を行うため，データセットの教師ラベルは使用しない．複数文からなる文書を各々の文に分割し，追加対象の語彙を含む文のみを学習・評価のデータセットとして用いる．

追加する語彙は，表 4.1 に示す 20 語とする．これらの語は，本実験で使用する事前学習済み BERT では複数 Token で処理される語である．また，事前学習した Wikipedia コーパスでの出現頻度が低く，応用タスクの Amazon レビューコーパスに複数回出現する語である．学習・テストデータの数量と各語彙の用例数は表 4.1 に示す通りである．

4.2　実験結果

提案手法において求めた追加語彙の類義語とその類似度は表 4.2 に示す．提案手法では，これらの類義語の BERT の Token Embeddings を追加語彙の Token Embeddings として語彙追加を行った．

各手法の実験結果の正解率は表 4.3 に示す．表 4.3 では，静的な単語埋め込み表現を利用した手法の結果を分散表現として示している．提案手法と分散表現の手法および平均ベクトルの手法は，追加語彙を Masked LM で同程度に予測することができている．しかし，BERTRAM の手法の実験では，追加語彙の予測精度は著しく低くなった．

表 4.1: 追加する語彙とデータ数

追加語彙	学習データ	テストデータ
不愉快	9	7
羅列	8	13
読み応え	8	16
読後感	8	17
駄作	7	9
鵜呑み	7	10
苦笑	7	9
読み手	7	10
蛇足	6	6
名著	6	16
好き嫌い	6	10
後味	5	12
いい加減	5	11
話し方	5	11
長文	5	8
醜悪	4	10
脱帽	4	9
誤字	4	8
金儲け	4	7
敷居	3	6
計	118	205

表 4.2: 追加する語彙と提案手法で利用した類義語

追加語彙	類義語	cos 類似度
不愉快	不快	0.7330
羅列	単語	0.5981
読み応え	読者	0.6012
読後感	文体	0.5903
駄作	酷評	0.6094
鵜呑み	間違っ	0.6639
苦笑	困惑	0.7338
読み手	難解	0.5126
蛇足	筆	0.4621
名著	叢書	0.6759
好き嫌い	嫌い	0.7076
後味	味わい	0.6447
いい加減	真面目	0.7478
話し方	話す	0.6276
長文	文章	0.6828
醜悪	邪悪	0.6625
脱帽	絶賛	0.5795
誤字	誤り	0.6326
金儲け	金持ち	0.6791
敷居	棚	0.5344

表 4.3: 実験結果（正解率）

提案手法	分散表現	平均ベクトル	BERTRAM
0.3703	0.3723	0.3792	0.1320

第 5 章

考察

5.1 各追加語彙毎の予測精度による分析

Masked Language Model による追加語彙の予測精度について，各語彙毎の予測精度を表 5.1 に示す．この語彙毎の予測精度から結果を分析する．

BERTRAM の手法による語彙追加を行ったモデルの予測精度は全体的に低い結果となった．語彙別に見ると，BERTRAM が高精度に予測できる語彙は，その他の手法においても概ね高精度に予測することができている．他の手法と BERTRAM の手法の違いの一つは Tokenize 処理にある．BERTRAM の Tokenize 処理では，追加語彙「○○」を「<BERTRAM: ○○>」の形式の特殊トークンとして処理した．特殊トークンの処理は優先的に行われるため，形態素解析の結果に違いを生んだ可能性がある．BERTRAM の研究において対象としていた言語は英語である．英語の場合，日本語と違いスペース区切りで大まかな単語分割が可能である．英語を対象とした BERT の Tokenize 処理は，スペース区切りで単語分割したあと BPE による subword 分割をする流れであり，日本語で行うような形態素解析器による分割は行われない．このような差があるため，日本語と違い英語では特殊トークンに置き換えることによる差異が小さかった可能性が考えられる．今後の課題として，日本語で追加語彙の表現を特殊トークンとして扱う場合とそうでない場合との予測精度への影響度を調査する必要があると考えている．

全体の精度としては平均ベクトルの手法が僅かに高いものの，語彙別の精度を見るとそれぞれ特徴がある．「駄作」「醜悪」などの語彙については，分散表現の手法や平均ベクトルの手法ではほとんど予測できなかったものの，提案手法では比較的高い精度で予測することができている．提案手法で用いたこれらの類義語はそれぞれ「酷評」「邪悪」と

表 5.1: 全体と追加語彙毎の予測精度（正解率）

追加語彙	提案手法	分散表現	平均ベクトル	BERTRAM
不愉快	0.7429	0.7429	0.6571	0.4000
羅列	0.3692	0.4615	0.4462	0.0154
読み応え	0.6875	0.6625	0.6375	0.1250
読後感	0.1059	0.0941	0.2117	0.0000
駄作	0.1500	0.0000	0.0750	0.0000
鵜呑み	0.8400	0.9400	1.0000	0.9200
苦笑	0.6889	0.5556	0.5556	0.1111
読み手	0.0400	0.1400	0.1800	0.0000
蛇足	0.0333	0.1667	0.1000	0.0000
名著	0.0250	0.0875	0.1000	0.0000
好き嫌い	0.5000	0.4800	0.4200	0.0000
後味	0.6000	0.8364	0.7818	0.8545
いい加減	0.2769	0.2615	0.2923	0.0000
話し方	0.0000	0.0000	0.0000	0.0200
長文	0.0250	0.0000	0.0000	0.0000
醜悪	0.4400	0.0200	0.2400	0.1600
脱帽	0.5333	0.6222	0.4222	0.0000
誤字	0.4667	0.6000	0.5333	0.0000
金儲け	0.6286	0.4857	0.4286	0.0000
敷居	0.5667	0.5667	0.6333	0.0000
全体	0.3703	0.3723	0.3792	0.1320

なっており，cos 類似度は 0.60 を超える．

一方で，「読み手」「蛇足」などの語彙については，分散表現の手法や平均ベクトルの手法による予測精度よりも提案手法の予測精度が低く，殆ど予測することができなかった．提案手法で用いたこれらの類義語はそれぞれ「難解」「筆」となっており，cos 類似度は 0.5126 と 0.4621 で，比較的低い類似度である．

このような結果から，提案手法において類義語の埋め込み表現を用いて語彙追加するとき，追加語彙と類義語との類似度の高さが，追加事前学習後の埋め込み表現の精度の高さに影響することが分かる．類似度の高い類義語を利用できる語彙については，他の手法に比べて提案手法が有効であると考えられる．

他の手法と比較して平均ベクトルの手法で予測精度が高くなった語彙は，「読後感」「鵜呑み」「読み手」「敷居」などである．これに対して，「不愉快」「脱帽」「金儲け」などの語彙は，他の手法と比較して平均ベクトルの手法の予測精度が低い結果となった．

分散表現の手法による予測精度が高くなった語彙は，「脱帽」「誤字」などの語彙である．逆に，「醜悪」については他の手法ではある程度の予測精度があるのに対して，分散表現の手法ではほとんど予測することができなかった．

5.2　応用タスクへの適応

本実験では，Masked Language Model による追加語彙の予測精度を評価した．しかし，実用的には語彙追加後の事前学習モデルは応用タスクを解くために使用される．ここでは，応用タスクとして日本語の文書分類と英語の固有表現抽出について実験を行い，各タスクの精度を評価する．

5.2.1　日本語の文書分類

語彙追加後の事前学習モデルで応用タスクの一つである文書分類を解く精度を評価する．

データセットは Amazon レビューコーパスを用いる．「Books」「DVD」「Music」の 3 つのドメインに対してそれぞれ文書分類を行う．データセットの内訳は表 5.2 に示す．教師データはレビューの星の数が与えられている．文書分類では，レビューの星 1 と星 2 をネガティブなサンプル，レビューの星 4 と星 5 をポジティブなサンプルとして 2 値分類する．

表 5.2: Amazon レビュー文書のデータセットの内訳

	books	DVD	music	3 領域の合計
訓練データ	2,000	2,000	2,000	6,000

語彙追加の対象とする語は，BERT に一つの語として学習されていない語彙で，Amazon レビューコーパスの学習データに存在する語を追加する．

使用する BERT の事前学習済みモデルは，実験と同様に Hugging Face 社の Transformers ライブラリでモデル名’cl-tohoku/bert-base-japanese’ として公開されているモデルである．

精度は，文書分類の正解率で見る．

実験は学習とテストをランダムシードを変えて 5 回試行する．この実験結果を図 5.1 に示す．

5.2.2　英語の固有表現抽出

本実験では，Masked Language Model による追加語彙の予測精度を評価した．しかし，実用的には語彙追加後の事前学習モデルは応用タスクを解くために使用される．ここでは，語彙追加後の事前学習モデルで応用タスクの一つである固有表現抽出を解く精度を評価する．

固有表現抽出は，人名や日付，組織名などの固有表現を文中から抽出するタスクである．このタスクは，単語の埋め込み表現の性能がタスクの精度に大きく影響するタスクである．

データセットは，CHEMDNER コーパスを用いる．このデータセットは化学物質と医薬品のドメインに関する英語のコーパスである．専門性のあるドメインのため，事前学習に使用しているコーパスのドメインと大きく離れていると考える．

使用する BERT の事前学習済みモデルは，Hugging Face 社の Transformers ライブラリ[*1] で，モデル名’bert-base-uncased’ として公開されているモデルである．学習コーパスは Wikipedia と BookCorpus である．語彙数は 32000 語，モデルは Transformer 層が 12 層の BASE モデル，最大入力長は 512 である．

使用する分散表現モデルは，gensim ライブラリでモデル名’glove-wiki-gigaword-300’ で利用できるモデルである．このモデルは，静的な単語埋め込み表現を出力する分散表現モデルの GloVe を学習したモデルである．学習コーパスには Wikipedia が用いられている．

評価する語彙追加手法は分散表現の手法と BERTRAM の手法である．ベースライン

[*1] https://github.com/huggingface/transformers

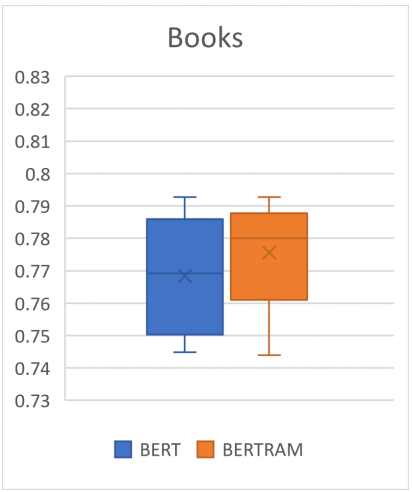
として語彙拡張を行わない標準の BERT モデルによる実験も行う．

語彙追加の対象は，BERT に一つの語として学習されていない語彙で，CHEMDNER コーパスの学習データに存在する 311 語を追加する．

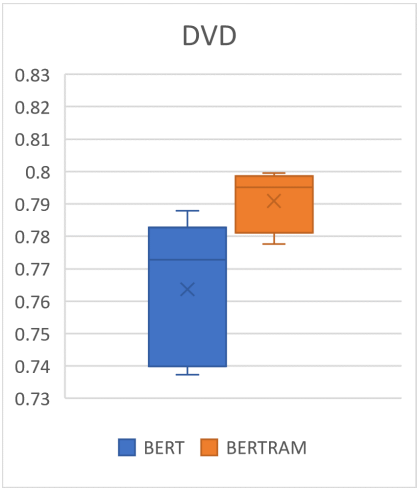
テストデータは，追加語彙の存在するデータのみとする．精度は，F 値，Precision，Recall の 3 つの値を見る．

実験は学習とテストをランダムシードを変えて 5 回試行する．この実験結果を図 5.2 に示す．

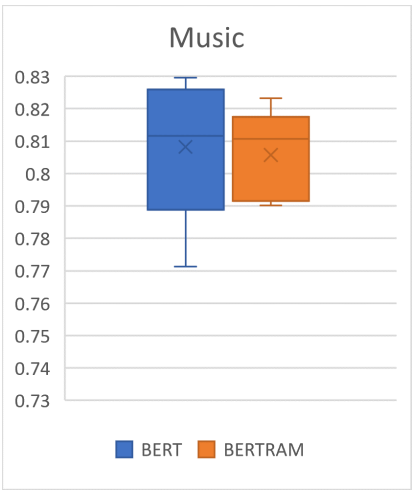
各々の手法の結果は似通った精度であるが，BERTRAM の手法による Recall の値は安定して高くなった．より多くの固有表現を安定して精度良く捉えることができると考えられる．



(a) Books 領域の正解率

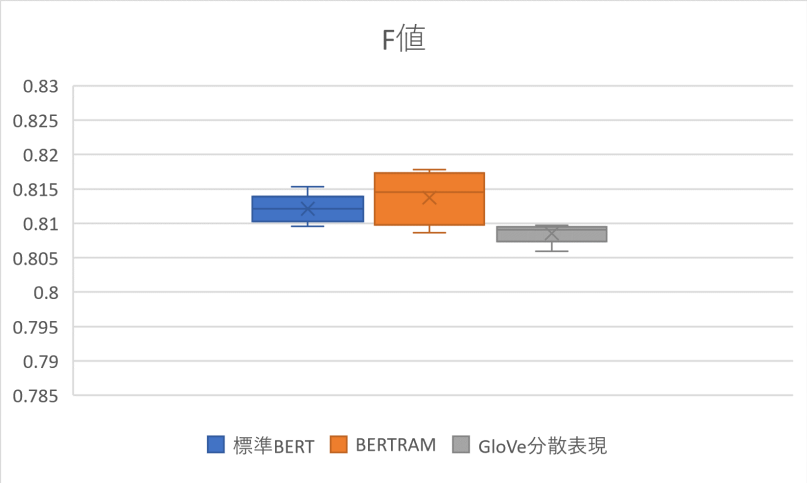


(b) DVD 領域の正解率

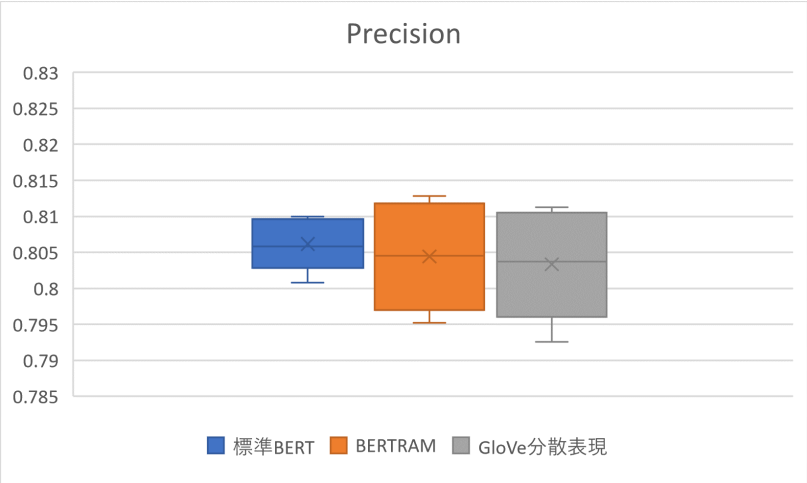


(c) Music 領域の正解率

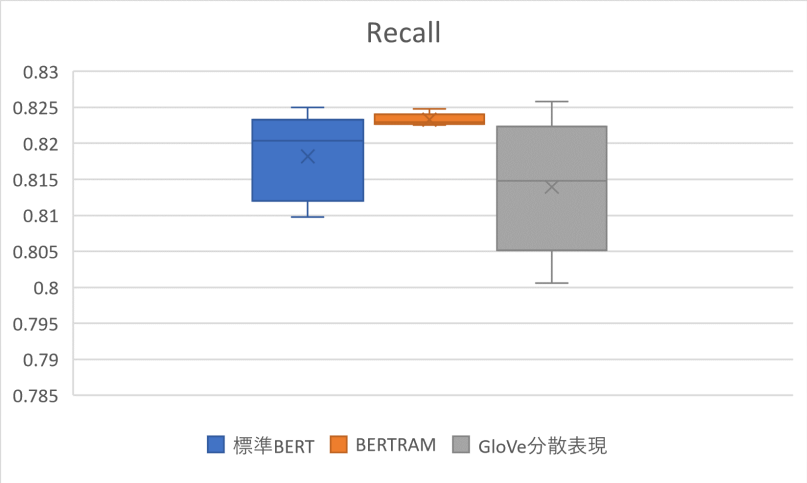
図 5.1: 文書分類タスクの精度（正解率）



(a) F 値による精度



(b) Precision による精度



(c) Recall による精度

図 5.2: 固有表現抽出タスクの精度

第 6 章

結論

本研究では，複数の語で構成される複合語や固有表現，句の表現といった複数 Token で表現される語を対象に，事前学習済みの BERT モデルを語彙拡張する際の，追加語彙の埋め込み表現の構築手法について提案した．

提案手法では，応用タスクデータに対して追加の事前学習を行うことを前提として，追加語彙の類義語を分散表現モデルから推定し，事前学習済み BERT の類義語に対する埋め込み表現を追加語彙の埋め込み表現として語彙追加を行った．

実験では提案手法の他に，静的な単語埋め込み表現を利用する手法と subword に対する BERT の埋め込み表現の平均ベクトルを用いる手法，BERTRAM の手法による語彙拡張を行った．Masked Language Model による追加語彙の予測精度で各手法を評価した．各語彙毎の予測精度と，提案手法で用いた類義語の類似度との関連から，類似度が高い類義語を用いて語彙拡張を行う提案手法が有効であることを示した．

本実験で対象とした語彙は，BPE によって複数 Token に分割される語であった．今後の課題としては，複数の単語で構成される固有表現や句といった表現に対しても有効な埋め込み表現を得ることのできる語彙追加手法への拡張が考えられる．

謝辞

本研究を進めるにあたって，多くのご指導を頂いた指導教員の新納浩幸教授に感謝致します．また，日常の議論を通して多くの知識，示唆を頂いた新納研究室の皆様に感謝します．

参考文献

[1] Matthew E. 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)*, pp. 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[2] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *Technical report, OpenAI.*, 2018.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-2019*, pp. 4171 –4186, 2019.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[5] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8342–8360, Online, July 2020. Association for Computational Linguistics.

[6] Yunzhi Yao, Shaohan Huang, Wenhui Wang, Li Dong, and Furu Wei. Adapt-and-distill: Developing small, fast and effective pretrained language models for domains. In *Findings of the Association for Computational Linguistics: ACL-*

IJCNLP 2021, pp. 460–470, Online, August 2021. Association for Computational Linguistics.

[7] Jimin Hong, TaeHee Kim, Hyesu Lim, and Jaegul Choo. AVocaDo: Strategy for adapting vocabulary to downstream domain. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 4692–4700, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[8] Timo Schick and Hinrich Schütze. BERTRAM: Improved word embeddings have big impact on contextualized model performance. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 3996–4007, Online, July 2020. Association for Computational Linguistics.

[9] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. LUKE: Deep contextualized entity representations with entity-aware self-attention. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6442–6454, Online, November 2020. Association for Computational Linguistics.

[10] 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*, pp. 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, Vol. 27. Curran Associates, Inc., 2014.

[12] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, Vol. 5, pp. 135–146, 2017.

[13] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of Tricks for Efficient Text Classification. *arXiv preprint arXiv:1607.01759*, 2016.

[14] Tomás Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting similarities among

languages for machine translation. *CoRR*, Vol. abs/1309.4168, , 2013.

[15] 平子潤, 笹野遼平, 武田浩一. 静的な単語埋め込みによるカタカナ語を対象とした bert の語彙拡張. 言語処理学会第 27 回年次大会 (NLP2021), 2021.

[16] 鈴木正敏, 松田耕史, 関根聡, 岡崎直観, 乾健太郎. Wikipedia 記事に対する拡張固有表現ラベルの多重付与. 言語処理学会第 22 回年次大会 (NLP2016), 2016.

付録

A　提案手法の実験のために使用したソースコード

提案手法の手順 2 で，分散表現モデルから追加語彙に対する BERT の語彙に含まれる類義語を推定し，その類義語をファイル出力するプログラムをソースコード A.1 に示す．提案手法の手順 3 で BERT に語彙を追加するプログラムをソースコード A.2 に示す．

ソースコード A.1: get_similar_vocab.py

```
1  from gensim.models import KeyedVectors
2  import os
3  from transformers import BertJapaneseTokenizer
4
5
6  def load_vocab(vocab_file: str):
7      words = []
8      with open(vocab_file, 'r') as reader:
9          for line in reader:
10             line = line.strip()
11             word = line.split('\t')[0]
12             words.append(word)
13
14     return words
15
16
17 def get_similar_list(w2v_model, new_vocab, bert_vocab):
18     similar_list = []
19
20     for word in new_vocab:
21         if word not in w2v_model:
22             print('not␣found␣"{}"'.format(word))
23             continue
```

```
24
25          similars = w2v_model.most_similar(word)
26          for index, (similar, degree) in enumerate(similars):
27              if similar in bert_vocab:
28                  similar_list.append([word, similar, degree])
29                  break
30          else:
31              print('not␣found␣"{}"'.format(word))
32
33      return similar_list
34
35
36  def save_similar_list(similar_list, similar_file='similar.txt'):
37      with open(similar_file, 'w') as writer:
38          for similar in similar_list:
39              writer.write('\t'.join(list(map(str, similar))) + '\n')
40
41
42  def get_common_vocab(w2v_vocab, bert_vocab):
43      return list(set(w2v_vocab) & set(bert_vocab))
44
45
46  def get_save_similar_list(w2v_model, new_vocab, bert_vocab,
        similar_file='similar.txt'):
47
48      common_vocab = get_common_vocab(w2v_model.index_to_key,
            bert_vocab)
49
50      with open(similar_file, 'w') as writer:
51          for word in new_vocab:
52              if word not in w2v_model:
53                  print('not␣found␣"{}"␣in␣word2vec␣vocab'.format(word)
                        )
54                  continue
55
56              similar = w2v_model.most_similar_to_given(word,
                    common_vocab)
57              writer.write('\t'.join([word, similar]) + '\n')
58
59  if __name__ == '__main__':
60      CWD = os.path.dirname(__file__)
```

```
61
62      model = KeyedVectors.load_word2vec_format(os.path.join(CWD, '
            jawiki.all_vectors.300d.txt'))
63
64      # 追加対象の語彙ファイル
65      new_vocab = load_vocab(os.path.join(CWD, 'amazon-corpus', 'books
            ', 'selection_vocab.txt'))
66
67      tokenizer = BertJapaneseTokenizer.from_pretrained('cl-tohoku/bert
            -base-japanese')
68      bert_vocab = tokenizer.get_vocab().keys()
69
70      get_save_similar_list(model, new_vocab, bert_vocab)
```

ソースコード A.2: add_similars_to_bert.py

```
1   from transformers import BertJapaneseTokenizer, BertForMaskedLM
2   import os
3
4
5   def load_similars(file_name):
6       similars = {}
7       with open(file_name, 'r') as reader:
8           for line in reader:
9               line = line.strip()
10              row = line.split('\t')
11              similars[row[0]] = tokenizer._convert_token_to_id(row
                    [1])
12
13      return similars
14
15  def add_similars_to_bert(model, tokenizer, similars):
16      new_words = list(similars.keys())
17
18      tokenizer.add_tokens(new_words, special_tokens=True)
19
20
21      new_vocab = {}
22      for word in new_words:
23          new_vocab[word] = tokenizer.encode(word, add_special_tokens=
                False)[0]
```

```
24
25      model.resize_token_embeddings(len(tokenizer))
26
27      for new_word in new_vocab:
28          model.bert.embeddings.word_embeddings.weight.data[
29              new_vocab[new_word], :] = model.bert.embeddings.
                    word_embeddings.weight.data[
30                  similars[new_word], :]
31
32      return model, tokenizer
33
34  if __name__ == '__main__':
35      CWD = os.path.dirname(__file__)
36
37      model_name = 'cl-tohoku/bert-base-japanese'
38
39      tokenizer = BertJapaneseTokenizer.from_pretrained(model_name)
40      model = BertForMaskedLM.from_pretrained(model_name)
41
42
43      similar_file = os.path.join(CWD, 'amazon-corpus', 'books', '
            similar.txt')
44
45      similars = load_similars(similar_file)
46
47      model, tokenizer = add_similars_to_bert(model, tokenizer,
            similars)
48
49      save_dir = os.path.join(CWD, 'proposed_add_model')
50      os.makedirs(save_dir, exist_ok=True)
51      tokenizer.save_pretrained(save_dir)
52
53      model.save_pretrained(save_dir)
54      tokenizer.save_vocabulary(save_dir)
```