令和 5 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

学習済み日本語分散表現と学習済み英語 BERT を用いた異言語間転送学習

所属　　情報工学専攻

著者　　菅波新 (22NM721N)

指導教員　　新納浩幸教授

令和 6 年 2 月 2 日 (金)

令和 5 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

学習済み日本語分散表現と学習済み英語 BERT を用いた異言語間転送学習

著者

菅波新 (22NM721N)

指導教員

新納浩幸教授

論文要旨

Artexte らによって行われた単言語表現の言語転送可能性を検証する研究では，英語で事前学習された BERT モデルの Embedding 層を除く Transformer 全層のパラメータを凍結して，Embedding のみを別の言語の単言語コーパスで学習した．その後，英語でファインチューニングしたモデルの Embedding 部分を別の言語で学習した Embedding で置き換えてゼロショットのクロスリンガル転移学習を行った結果，Multilingual BERT の精度と競うことを示した．

この結果から、Transformer の上位層は言語に依存していないとも考えられる．そこで Embedding 層だけを日本語向けに調整するだけで高機能な英語の学習済みモデルを日本語向けにそのまま利用できると考えた．実現すれば，低リソース言語でゼロからの事前学習が難しい状況でも，既存モデルの Embedding 層の学習だけで高機能なモデルを構築できる可能性が考えられる．

本研究では、Artexte らの研究で検証されなかった英語と日本語間の言語転送可能性を調査しつつ，Transformer の上位層が言語に依存しているかどうかを考察した．実験では学習済み英語 BERT のパラメータをベースとして，Transformer 層を全て凍結した状態で Embedding 層のみを日本語コーパスで学習することで日本語向け BERT を構築した．モデルの評価には Amazon レビューデータの感情分析に対する正解率を採用した．

提案手法によって構築された日本語 BERT は Multilingual BERT の精度に迫ることを実証し，Embedding 層のみの調整によって既存の高機能なモデルを別の言語にある程度適応できることを示した．一方で，東北大 BERT の精度には及ばなかったことから、Transformer の Embedding より上位の層は依然として言語に依存しているという考察に至った．

Master's Thesis in Scholastic 2024, Major in Computer and Information Sciences,
Granduate School of Science and Engineering, Ibaraki University

Construction of Japanese BERT with Fixed Token Embeddings

Author

Arata Suganami (22NM721N)

Adviser

Prof. Hiroyuki Shinnou

Abstract

In a study conducted by Artexte et al. to test the language transferability of monolingual representations, they froze the entire Transformer layer of a learned English BERT and trained only Embedding on a monolingual corpus in another language. They then replaced the Embedding in the fine-tuned English model with Embedding learned in that language and performed zero-shot cross-lingual transfer learning, showing that it competes with Multilingual BERT's accuracy.These results suggest that the upper layers of Transformer are language-independent. Therefore, I thought that English trained models could be used as-is for Japanese by simply adjusting the Embedding layer for Japanese. If this is realized, it may be possible to construct a highly functional model by simply learning the Embedding layer of an existing model, even in a low-resource language where pre-training from scratch is difficult.

In this study, I investigated language transferability between English and Japanese, which was not verified in Artexte et al. and considered whether the upper layers of Transformer are language-dependent. In the experiment, I trained only the Embedding layer on a Japanese corpus, with all the Transformer layers frozen, based on a trained English BERT. I demonstrated that the Japanese BERT constructed by the proposed method approaches the accuracy of MultilingualBERT, and that the existing model can be adapted to another language to some extent just by adjusting the Embedding layer. On the other hand, it did not reach the accuracy of TohokuBERT, suggesting that the layers above Embedding in Transformer are language-dependent.

目次

表目次

図目次

第 1 章

序論

Artexte ら [1] の単言語表現の言語転送可能性の研究では，英語コーパスで事前学習した BERT [2] の Embedding 層を除く Transformer [3] 全層のパラメータを凍結させた状態で, 別の単言語コーパスを用いて Embedding のみを学習した．その後, 英語でファインチューニングしたモデルの Embedding を学習した Embedding で置き換えるゼロショットのクロスリンガル転移学習を行った結果，精度が Multilingual BERT(MBERT) [2] と競争することを示し，多言語表現を使わない単言語表現による言語の転送可能性があることを示した.

このことから，Transformer は Embedding 層より上位の層は言語に依存していないとも考えられる．そこで，Embedding 層だけを日本語向けに調整することで高機能な英語の学習済みモデルをそのまま利用することができ，少ないコストで日本語向けのモデルを用意できるのではないかと考えた.

成功すれば，低リソース言語でゼロからの事前学習を行う場合に高機能なモデルを作ることが難しい状況においても，既存のモデルの Embedding 層の学習だけで比較的高機能なモデルを構築できる可能性が考えられる.

本研究では，Artexte らの研究で調査されていなかった英語と日本語間の言語転送の可能性を調査するとともに，Transforemer の Embedding より上位の層が言語に依存しているかどうかを調査することを動機とし，学習済み英語 BERT のパラメータをベースとして，Transformer 層を全て凍結した状態で Embedding 層のみを日本語のコーパスで学習することで日本語向けのモデルとなるように学習することで日本語向け BERT を構築した．モデルの評価には Amazon レビューデータの感情分析に対する正解率を用いた．比較するモデルとして多数の言語コーパスから共同学習されている MBERT と東北

大 BERT を挙げた.

提案手法によって構築されたモデルは MBERT の精度に迫ることを実証し，Embedding 層のみの調整によって既存の高機能なモデルを別の言語にある程度適応させることができることを示した．一方で，東北大 BERT の精度には全体的に及ばなかった事から，やはり Transformer の Embedding 層より上位の層は言語に依存しているのだろうという考察に至った.

第 2 章

関連研究

2.1 単語分散表現

単語分散表現は自然言語処理において単語を連続したベクトル空間上に埋め込む手法で，単語の意味や文脈を捉えるのに有用である．ベクトル空間に単語の特徴をベクトルとして埋め込むことから，埋め込み表現とも呼ばれる．単語がベクトルで表されることにより，自然言語，つまり我々が普段使う言語を機械が扱うことができる．Bag-of-Words や TF-IDF のように文章中の単語の出現回数からベクトルを作成する手法や，後述する Word2Vec のように大量のコーパスからニューラルネットワークを使って単語の分散表現を学習する手法がある．

2.2 word2vec

word2vec [4] [5] は 2013 年に Tomas Mikolov らが発表した単語分散表現を構築する手法である．入力層，中間層，出力層からなるニューラルネットワークで構成される簡潔なネットワークで，高密度の行列の計算が不要であるため，高速で単語分散表現を学習することが出来る．論文 [4] では 16 億語のデータセットから 1 日以内で単語分散表現を学習できる上，精度が大幅に向上することを実験により検証し，従来の手法と比べ低い計算コストで精度が向上することを示した．word2vec のモデルのアーキテクチャの詳細を以下で説明する．

2.2.1　CBOW

CBOW(Continuous Bag-of-Words Model) は周辺の単語から中心の単語を予測するモデルである．学習時間が短く，大きなデータセットに対する分散表現の学習に適している．

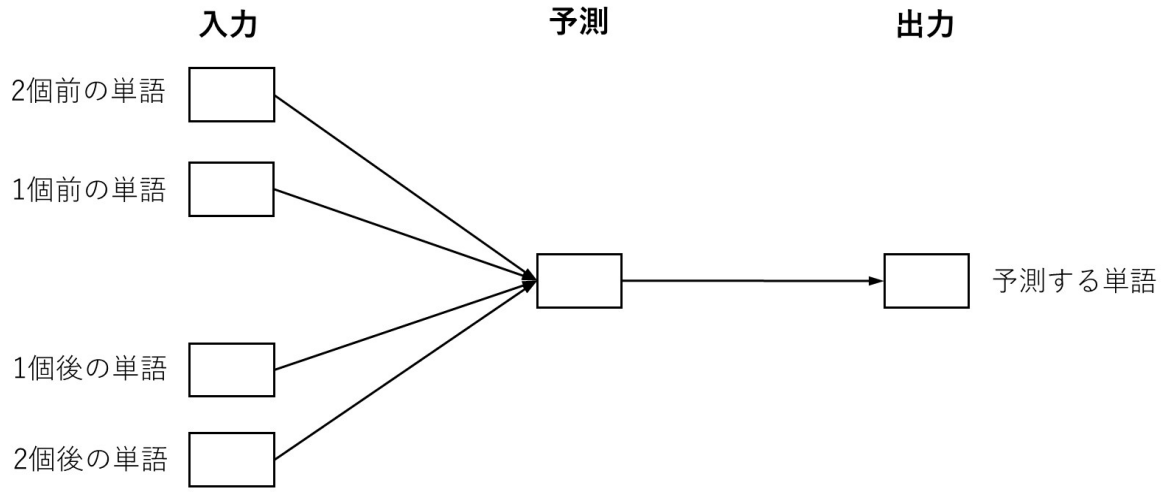

図 2.1: CBOW のイメージ図

2.2.2　Skip-gram

Skip-gram(Continuous Skip-gram Model) は中心の単語から周辺の単語を予測するモデルである．CBOW と比べ学習に時間はかかるが，構築する単語分散表現の性能は良くなっている．

2.3　Transformer

Transformer [3] は 2017 年に Vaswani らによって提案されたモデルアーキテクチャである．それまで主流だった RNN(Recurrent Neural Network) の並列計算に向かないという問題や，CNN(Convolutional Neural Network) の長文の依存関係を捉えることが難しいという問題を解決した，Attention 機構をベースにしたモデルである．8 つの GPU で 3.5 日学習した Transformer モデルは英語からフランス語への翻訳タスクにおいて，当時の最先端を記録した．

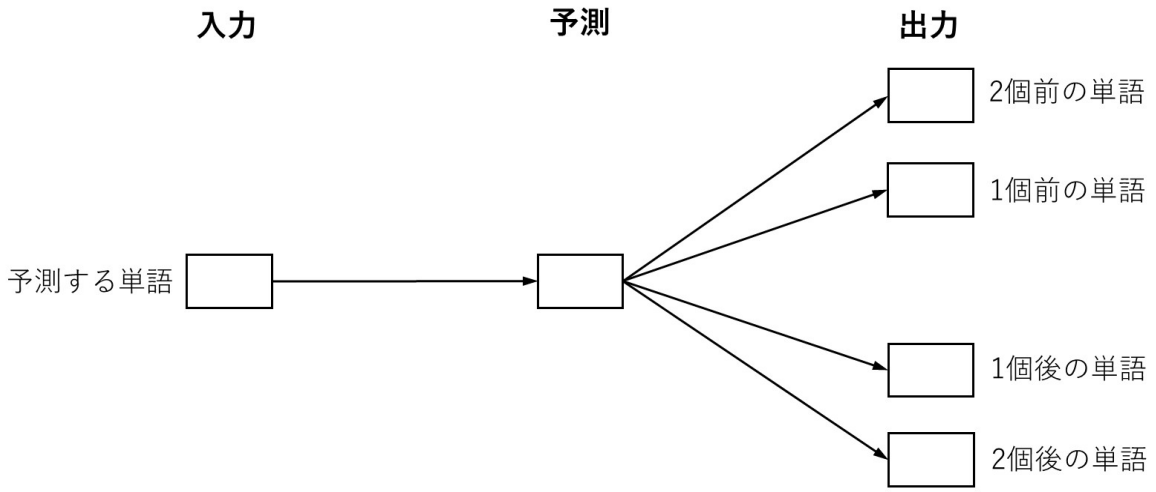

図 2.2: Skip-gram のイメージ図

2.3.1　Transformer のモデルアーキテクチャ

Transformer はエンコーダとデコーダから成り立つ．エンコーダは 6 層の同一レイヤが重なったもので，各層がそれぞれ Multi-Head Attention と position wise fully connected feed-forward network というサブレイヤを持つ．各サブレイヤの後には残差結合とレイヤ正規化が行われる．Transformer の全てのサブレイヤの次元は演算を容易にするために 512 次元の出力をする．デコーダも 6 層の同一レイヤが重なったもので，エンコーダの 2 つのサブレイヤの間にエンコーダの出力を受け取る Multi-Head Attention が追加されるため，3 層のサブレイヤを持つ．デコーダでも同様に，各サブレイヤの後には残差結合とレイヤ正規化が行われる．また，1 つ目の Multi-Head Attention は Masked Multi-Head Attention に変更され，未来の情報をマスキングすることで既知の出力にのみ依存するようになっている．デコーダを抜けた後は Linear と Softmax によって確率分布が得られる．

2.3.2　Attention と Multi-Head Attention

Attention を求める式を式 2.1 に示す．

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{D_k}})V \tag{2.1}$$

Attention は次元 D_k の Query(Q) と Key(K) のドット積を計算し，$\sqrt{D_k}$ でスケーリングを行い，softmax 関数を通した結果と Value(V) でドット積を計算したものを出力とする．つまり，Query と Key の類似度の計算を行って，Query に対して最も近い Key を求め，Key に対応する Value を求めようということである．

また，Multi-Head Attention は 512 次元の単一の Attention を用いるのではなく，分割した複数の小さい次元の Attention をそれぞれ求めて，出力を一つに投影するものである．Vaswani らは Attention を 8 分割し，64 次元の Attention から出力を求めた．Multi-Head Attention は異なる位置の表現から情報を得ることを可能にし，計算コストを抑えたまま性能を向上することができる．

2.4　BERT

BERT [2]（Bidirectional Encoder Representations from Transformers）は 2018 年 10 月に Google から発表された高性能な事前学習済みモデルで，Transformer [3] で使用された Multi-Head Attention という層を 12 層または 24 層重ねたモデルである．文章を単一方向に学習する従来のモデルとは異なり，事前学習に Masked Language Model と Next Sentence Prediction というタスクを採用することで双方向からの学習が可能なため，文脈を理解する能力が高くなっている．また，1 つの BERT モデルに対して，転移学習やファインチューニングを行うことで様々な自然言語処理タスクに適応できる優れた汎用性を持つ．発表されたモデルには 768 次元 12 層からなる BERT-base と 1024 次元 24 層からなる BERT-large がある．

2.4.1　BERT の事前学習

BERT は以下の 2 つの教師なしタスクを用いて事前学習される．

- タスク 1：Masked Language Model (MLM)
  入力文に含まれるいくつかの単語をマスクし，マスクされた部分に入る元の単語を予測するタスクである．以下の例は元論文 [2] からの引用である．
  (1) 入力文に含まれる単語の内 15% をランダムに選ぶ．
  (2) 選ばれた単語を 80% の確率で [MASK] トークンとする．
  my dog is **hairy** → my dog is [**MASK**]

(3) 選ばれた単語を 10% の確率でランダムな単語にする.

my dog is **hairy** → my dog is **apple**

(4) 選ばれた単語を 10% の確率でそのままの単語にする.

my dog is **hairy** → my dog is **hairy**

(5) [MASK] トークンに何の単語が入るか前後の文脈から予測する

MLM によって，[MASK] トークンによって隠された単語を文の前後から予測していくため，双方向性を持つ事前学習モデルが得られる.

- タスク 2：Next Sentence Prediction (NSP)
  入力として実際に続いている 2 つの文を受け取り，50% の確率で 2 文目を別の入力からのランダムな 1 文に置き換えた後，1 文目と 2 文目が実際に続いているかどうかを予測するタスクである．以下の例は元論文 [2] からの引用である.

  例 1　=　[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]　(男は??な店に行った/彼は 1 ガロン??牛乳を買った)
  → IsNext (2 文は連続していると予測)
  例 2　=　[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]　(男は店に??した/ペンギン??は飛べない鳥だ)
  → NotNext (2 文は連続していないと予測)

  NSP によって，BERT は 2 文の関係性を学習していくため，Q&A タスクや自然言語推論といった複数の文の関係を理解する必要があるタスクにも適用することができる.

2.4.2　ファインチューニング

ファインチューニングは BERT を特定のタスクに特化させるように学習するプロセスを指す．BERT は事前学習を通して豊かな言語表現を得るが，ファインチューニングによって特定のタスクに適応させることで，そのタスクに対して高い性能を実現することができる.

ファインチューニングでは BERT の最終層にタスクに合わせた分類器など，追加の層を組み込む．その後，事前学習で得られたパラメータを初期値として，ラベル付きデータを用いて BERT と分類器のパラメータを学習する．比較的少数のデータから高性能なモデルを構築できる点や様々なタスクに応用できる点から，広く利用されている．

2.4.3　BERT の入力ベクトル

BERT の入力ベクトルには 3 種類の要素が関わる．その詳細を図 2.3 に示す．まず与えられた単語列は辞書によりそれぞれ単語に対応する id(token id) に変換され，BERT に入力される．Token Embedding は単語 (token id) 一つ一つに対応する埋め込み表現，すなわち単語分散表現である．この単語分散表現は BERT の事前学習を通して学習される．Segment Embedding はそれぞれの文に対応する埋め込み表現である．Position Embedding は入力された各単語の位置に対応する埋め込み表現である．これらの 3 つの埋め込み表現を合計したベクトルが BERT の入力ベクトルとなる．

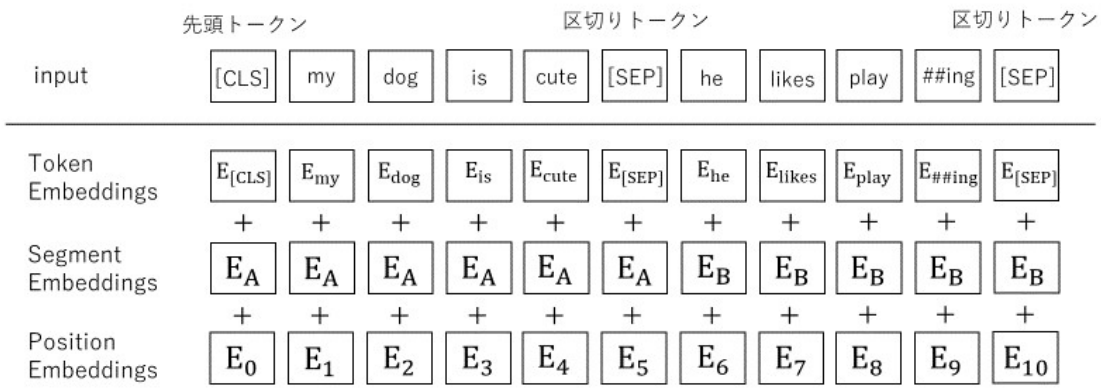

図 2.3: BERT の入力表現

2.5　RoBERTa

RoBERTa(Robustly optimized BERT approach) [6] は，2019 年に Liu らによって提唱された BERT の改良版モデルである．RoBERTa は BERT が自然言語処理において優れた性能を示す一方で，BERT は著しく訓練不足であり，さらなる改良が可能であるという仮説に基づいている．BERT から変更された点を以下で説明する．

- より多くのデータ，より長い時間，より大きなバッチでモデルを訓練する
  BERT では事前学習用のデータセットとして英語 Wikipedia と BookCorpus の合計 16GB のデータセットを使用しているが，RoBERTa では CC-News, OpenWebText，Stories というデータセットを追加し，合計 160GB ものデータセットを事前学習に使用している．これにより精度が大幅に向上したことが示されている．また，BERT ではバッチサイズ 256 で 100 万ステップの訓練が行われているが，BERT と同じデータセットを用いてバッチサイズを変化させた論文内の実験により，バッチサイズ 2000 で 12 万 5000 ステップの場合の方が精度が優れていることが示されている．

- Next Sentence Prediction を削除する
  NSP の有無以外を同条件で事前学習したモデルの比較実験により，NSP を行わない方が精度が向上すると示され，NSP の有効性が否定された．

- より長いシーケンスで訓練する
  データの入力方法以外を同条件で事前学習したモデルの比較実験により，１つ以上のドキュメントから連続した 512 単語を入力とする場合の精度が向上すると示され，より長いシーケンスが精度向上に寄与することが述べられた．

- 訓練データに適用するマスキングパターンを動的に変化させる
  オリジナルの BERT では学習時に１度だけマスキングを行っている (静的マスキング)．１つの文章を 10 個複製し，それぞれ違うマスキングの文章を生成するが，40 エポック回しているため同じマスクが 4 回出ることになる．対して RoBERTa では，学習する度にマスキングを行うため，毎回違うマスクの文章がモデルに供給される (動的マスキング)．静的マスキングと動的マスキングを比較した論文内の実験では動的マスキングにより BERT とほぼ同等か若干良い精度が出ることが示されている．

これらの手法を取り入れた RoBERTa は事前学習する際のコーパスを増やす前の状態においても全てのデータセットで BERT-Large の精度を超えた．その後，コーパスを増やすことと学習のステップ数を増やすことによってさらなる精度を示し，提案した学習方

法の有効性を示した.

2.6 Multilingual BERT

Multilingual BERT [2] は，多言語対応の事前学習モデルとして発表されたモデルである．MBERT は BERT のアーキテクチャをベースにし，104 言語の Wikipedia データを用いて事前学習されている．通常の BERT では単一言語のデータセットで学習されるのに対し，MBERT では複数言語のデータセットを同時に用いて，共通のトークン化やエンコーダーを共有する．これにより，モデルが複数言語にわたる汎用的な言語表現を学習し，異なる言語のタスクに適用可能になる.

また，Pires ら [7] の MBERT の多言語表現能力を調査する研究では，MBERT のファインチューニング時に入力として与えていない別の言語に対するタスクに対しても良い性能が出ることが示され，MBERT が転移性能を有していると述べられている．一方で，英語と日本語のような文の構造が似ていない言語に対しては転移性能が低いことが示され，MBERT は汎用的な多言語表現は学習できているが，語順に関係する体系的な構造はあまり学習できていないという見解を述べている.

2.7 On the Cross-lingual Transferability of Monolingual Representaions (単言語表現の言語伝達可能性)

Artetxe らは Pires ら [7] が MBERT がゼロショットでの異言語転送に成功したことを受け，MBERT の言語を超えた汎化能力を評価しつつ，MBERT の複数言語にわたる共同訓練や共有語彙の考え方に対抗する代替アプローチとして単言語表現による異言語転送を提案した.

提案された手法を図 2.4 に示す．まず，(a) 言語 L1 のコーパス (英語) によって BERT を事前学習した後，(b) 言語 L1 で事前学習したモデルの他の全ての層のパラメータを凍結した状態で言語 L2 のコーパスを用いて言語 L2 の新しい埋め込み行列を学習することで新しい言語に転送している．これにより単言語で学習されたモデルが別の言語向けのモデルへと調整される.

また，論文内で行われたゼロショットによる異言語転送を図**??**に示した．学習済み言語 L1 モデルを言語 L2 に転送した後，(c) 言語 L1 のモデルの埋め込み行列を凍結した

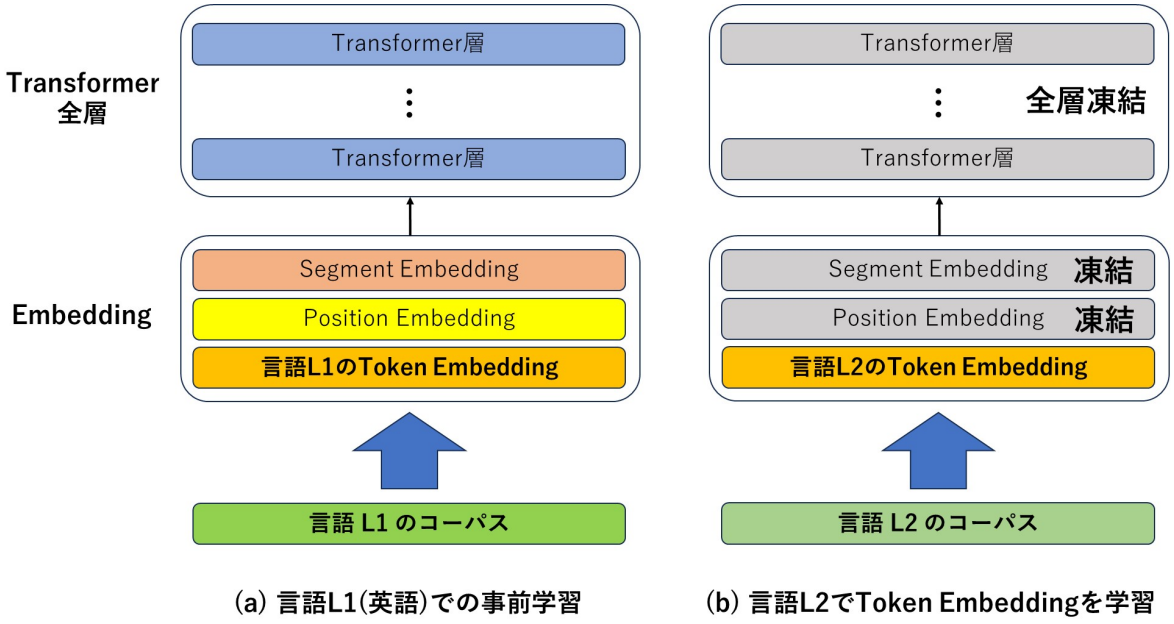

図 2.4: 単言語表現の伝達

まま言語 L1 のラベル付きデータでファインチューニングを行い，(d) 最後に，手順 (b) で学習した言語 L2 の埋め込み行列で置き換えることでゼロショットによる異言語転送を行っている．

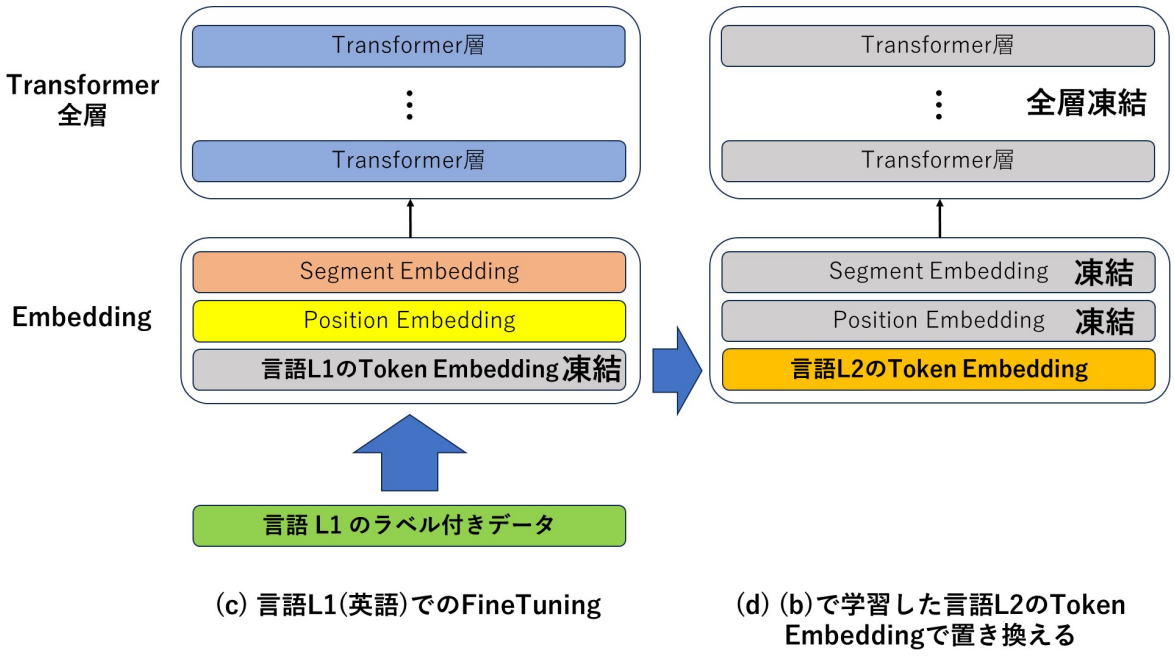

図 2.5: ZeroShot による異言語転送

この手法によって英語 BERT から他の言語に転送されたモデルは標準的なゼロショットクロスリンガルベンチマークにおいて MBERT と競うことを示した．これにより，言語間でのサブワードの共有や多数の言語にわたる共同訓練が MBERT にとって必要ではないことを実証し，単言語モデルが言語をまたいだ，言語の抽象的な概念を学習することを示唆した．

第 3 章

提案手法

3.1 英語 BERT と日本語 Embedding による異言語転送学習

英語から日本語への異言語転送学習が可能であるかを検証するために，Artexte ら [1] による単言語間転送学習の手法を参考にして，以下のような手順によって学習済み英語 BERT と日本語 Embedding の学習による BERT を構築する．提案手法のイメージ図を図 3.1 に示す．また，細かい設定等は実験の章で説明する．

- (1)：既存の学習済み英語 BERT をベースとして用意する．
- (2)：学習済み英語 BERT の Embedding 層を日本語 BERT の Embedding 層で置き換える．
- (3)：日本語コーパスを使って Embedding 以外の層のパラメータを全て凍結したまま，新しい TokenEmbedding を学習することでモデルを日本語に移行させる．

3.2 比較するモデル

本研究では，提案手法によって構築されたモデルと MBERT，そして東北大 BERT を，日本語の感情分析タスクの正解率を用いて比較する．もし，提案手法によるモデルが Multilingual BERT と競り合うか上回る精度を示すならば，英語から日本語への転送が有効であること，加えて多数の言語を一度に学習させることなく単言語間で十分に学習させることが有効かどうかを判断できる．また，提案手法によるモデルと東北大 BERT で大幅に異なる点は Transformer 層にあるため，精度に大きな差が見られれば，

Transformer が言語に依存していないことを検証できる．

この比較実験を通じて、提案手法が言語転送において MBERT を上回り，同時に Transformer が言語に依存しないことを示すことが期待される．

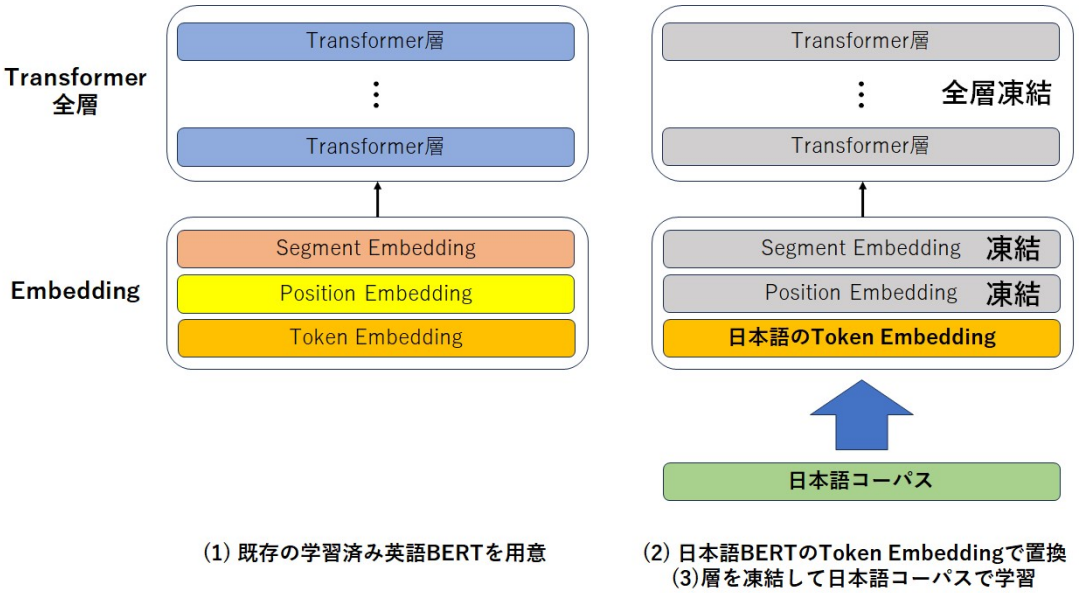

図 3.1: 提案手法のイメージ図

第 4 章

実験

4.1 モデルの事前学習に使用したコーパス

2023 年 11 月 20 日時点の日本語 Wikipedia のダンプファイル[*1]からテキストコーパスを生成した．wikiextractor を用いてテキストクリーニングを行った後，複数の txt ファイルを結合して前処理を行い，1 行 1 文からなる 1 つのテキストにした．生成されたコーパスファイルは 3.4GB で，約 2400 万の文で構成されている．

4.2 モデルの事前学習

事前学習を行う際のモデルの初期パラメータとして学習済みの英語 BERT である，BERT-base「bert-base-uncased」[*2]を読み込んだ．さらに，読み込んだ学習済み英語 BERT の word_embedding(TokenEmbedding にあたる埋め込み) 部分のパラメータを東北大学の乾・鈴木研究室が作成した日本語版 BERT(以後，東北大 BERT と呼ぶ)「cl-tohoku/bert-base-japanese」[*3]の word_embedding のパラメータで置き換えた．この際，embedding の変更に伴って学習するモデルのトークン数を英語 BERT の 30522 から東北大 BERT の 32000 に変更した．その他の次元数や層など，モデルのアーキテクチャは BERT-base から変更していない．この時点でのモデルのイメージ図を図 4.1 に示す．

[*1] https://dumps.wikimedia.org/jawiki/latest/ 内の”jawiki-latest-pages-articles.xml.bz2”
[*2] https://huggingface.co/bert-base-uncased
[*3] https://huggingface.co/cl-tohoku/bert-base-japanese

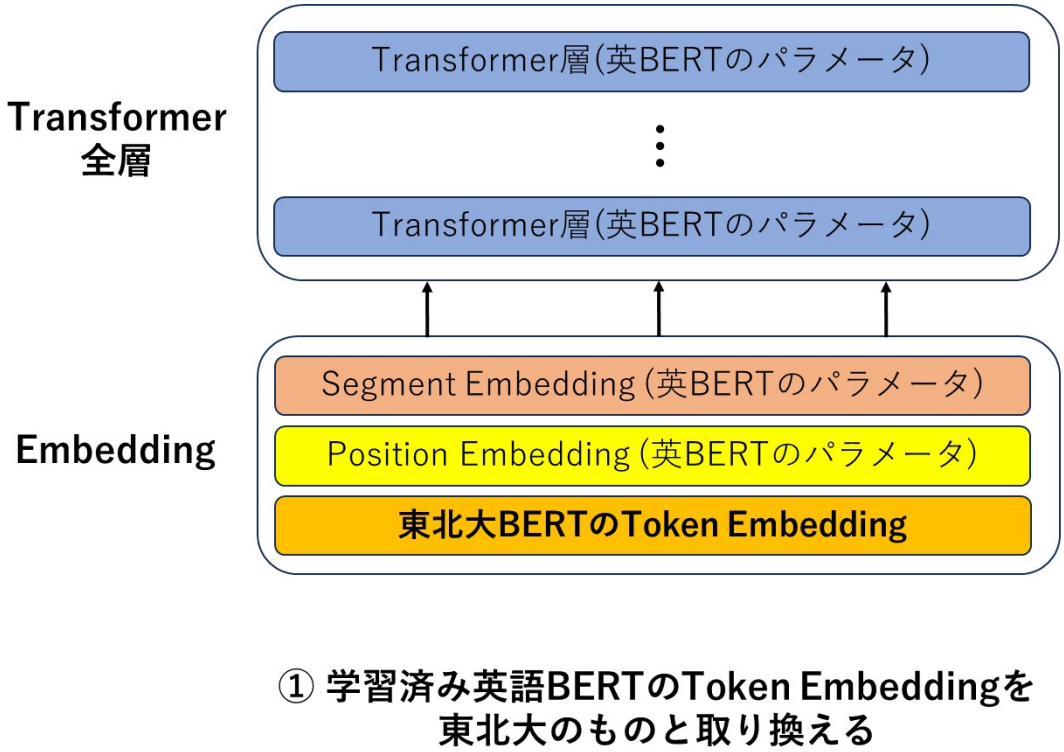

図 4.1: Embedding 層の置換

次に，word_embedding 以外のパラメータを凍結し，4.1 節で述べた日本語 Wikipedia コーパスと東北大 BERT の tokenizer「cl-tohoku/bert-base-japanese」を使って MLM のみの学習を行った．MLM の学習もオリジナルの BERT と同様に単語の 15% を [MASK] で置き換えている．この手順でのモデルのイメージ図を図 4.2 に示す．

モデルを学習する際の GPU には NVIDIA GeForce RTX 3090(24GB) を使用し，バッチサイズ 16 で 10 エポック分学習した．事前学習にはおよそ 200 時間かかった．

4.3　Amazon レビューのネガポジ判定

モデルを構築した後，日本語の Amazon レビューを対象とした感情分析タスクにおける正解率を他の学習済み言語モデルと比較する事でモデルの評価を行った．

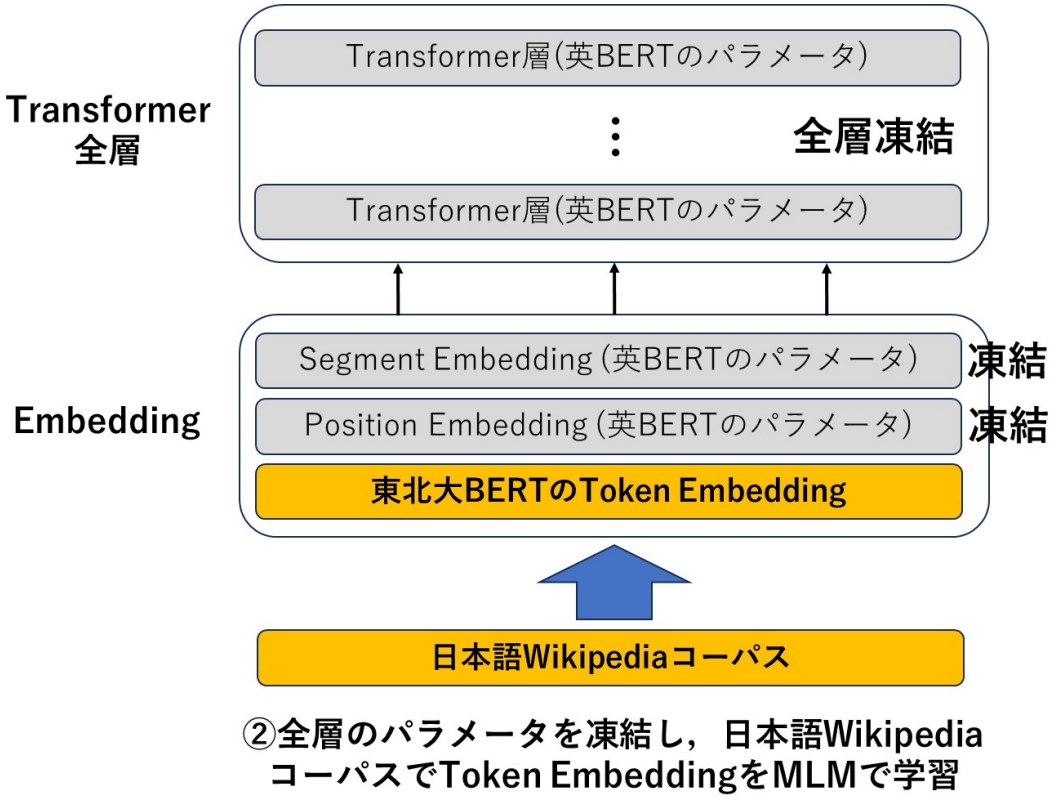

図 4.2: Embedding 層の学習

4.3.1　評価用データセット

評価データには Webis-CLS-10[*4]というデータセットを使用した．このデータセットにはドイツ語，フランス語，日本語，英語の Amazon レビュー文書が収録されているが，本実験では日本語の文書を使用した．データのラベルは Amazon レビューの星の数を表しており，３を除く１から５までの４段階で表現される．本実験では，ラベルが４，５のデータを高評価 (positive)，ラベルが 1，２のデータを低評価 (negative) として与えたレビュー本文から評価を分類させる２値の感情分析を行った．

また，このデータセットには言語ごとに books，dvd，music の３領域があり，それぞれに訓練データ 2000 個，テストデータ 2000 個が収録されている．この内，訓練データの先頭から 100 個を訓練用データ，101 個目から 200 個目を検証用データとした．また，テストデータは 2000 個全て使用した．使用したデータの内訳を表 4.1 に示す．データセット内の<text>タグ内の文章をレビュー本文としているが，books のテストデータに

[*4] https://zenodo.org/records/3251672

１つだけ<text>タグ内にレビューが無いデータがあったため，books のテストデータ数のみ 1999 個になっている．

表 4.1: データセットの内訳

	日 books	日 dvd	日 music
訓練データ	100	100	100
検証データ	100	100	100
テストデータ	1999	2000	2000

4.3.2　モデルの評価

訓練データを 100 個用いてモデルをファインチューニングし，1 度学習が終わるごとに学習したモデルに検証データを 100 個用いて loss を計測した．loss の計算には交差エントロピーを使用した．図 4.3 に示すように学習と検証のループを 30 エポックまで行い，検証データに対する loss の最小値が 3 回更新されなかった場合に early stopping とみなしてモデルのファインチューニングを途中で止めて，その時点までで loss が最小だったモデルをベストモデルとして保存した．

最終的にベストモデルに対しテストデータを 2000 個使用して Amazon レビューからその商品に対する評価を予測する 2 値の感情分析タスクを行い，正解率を計測した．このモデルの評価を books，dvd，music の領域ごとにそれぞれ行っている．

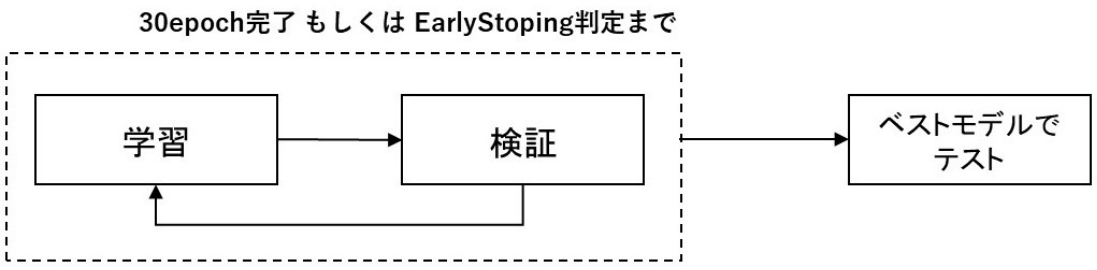

図 4.3: モデルの評価

4.3.3　評価対象のモデル

本実験では以下の 3 つの条件でモデルを評価した．括弧内には以後の名称を記す．また，評価データをトークン化する際は評価するモデルに対応するトークナイザーを使用した．

- 日本語の訓練データでファインチューニングした MultilingualBERT(JP-MBERT)
- 日本語の訓練データでファインチューニングした提案手法による BERT(My-BERT)
- 日本語の訓練データでファインチューニングした東北大 BERT(東北大 BERT)

4.4　実験結果

4.3 節で説明した 3 つのモデルに Amazon レビューデータを対象とした感情分析タスクを解かせた．正解率を表 4.2 と図 4.4 に示す．図表の通り，提案手法により構築されたモデルは，日本語でファインチューニングした MBERT の精度より全体的に少し劣っているが，dvd データセットにおいて精度を上回ったことが分かった．

表 4.2: 感情分析タスクの正解率

	JP-MBERT	My-BERT	東北大 BERT
books	**0.640**	0.553	0.765
dvd	0.611	**0.638**	0.806
music	**0.651**	0.602	0.719

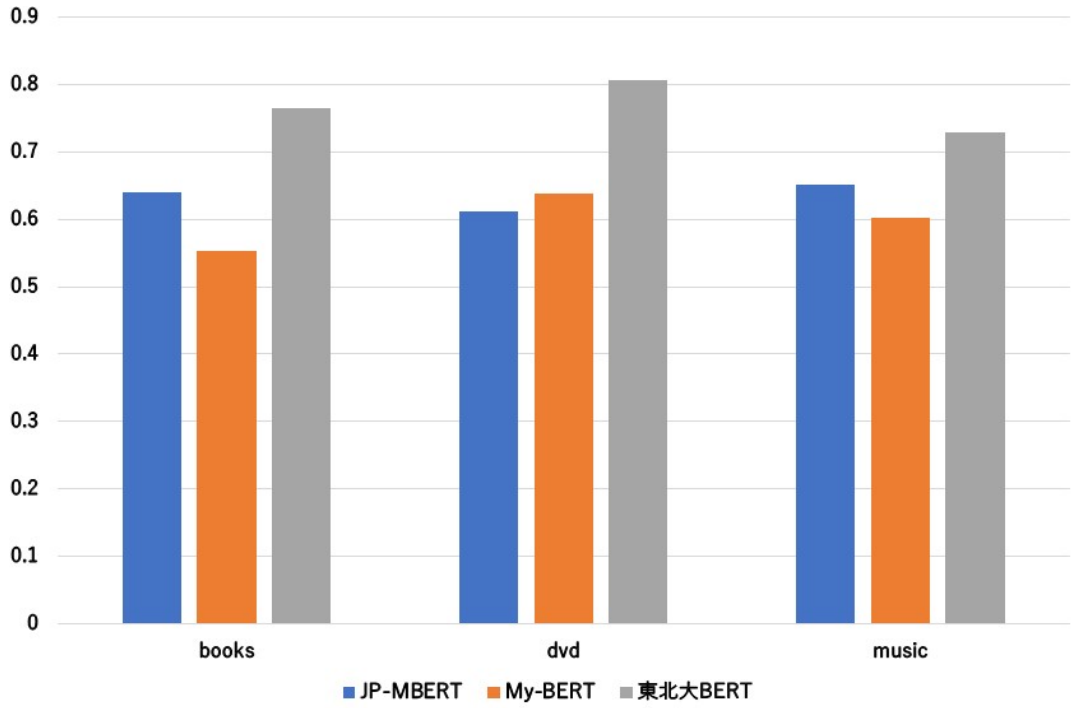

図 4.4: 感情分析タスクの正解率

第 5 章

考察

5.1 英語と日本語間の言語転送の可能性について

表 4.2 より，提案手法によって構築された BERT は DVD レビューデータにおいて日本語の訓練データでファインチューニングした MBERT の精度を上回った．books と music では精度が劣っているが，全体的に見ると MBERT に精度が迫っていると言えることが分かった (表 5.1)．このことから，英語から日本語への言語転送であってもEmbedding 層のみを学習することである程度言語を転送出来るのではないかと考えられる．

表 5.1: 感情分析タスクの正解率の平均値

	JP-MBERT	My-BERT	東北大 BERT
正解率の平均値	0.634	0.597	0.763

5.2 Transformer が言語に依存するかどうか

表 4.2 より，提案手法によって構築された BERT は全てのレビューデータにおいて東北大 BERT の精度を大幅に下回った．提案手法によって構築されたモデルは学習済み英語 BERT の Embedding を Transformer 全層のパラメータを凍結した状態で日本語のコーパスによって学習されているため，Embedding 層は日本語向けに調整されているが，Transformer は英語を学習したパラメータのままである．対して，東北大 BERT は日本

語の学習コーパスによってゼロから学習されているため，Embedding 層も Transformer 層も当然日本語を学習したパラメータとなっている．

Embedding 層はどちらのモデルも日本語を学習しているが，Transformer 層が学習した言語の違いによって，表 5.1 に示されるように正解率の平均に 0.17 ポイントもの違いがあった．よって，Transformer 層が学習した言語によって正解率が代わることから，Transformer 層はやはり言語に依存しているのではないかと考えられる．

第 6 章

結論

本稿では，英語と日本語間の言語転送の可能性を調査するとともに，Transforemer の Embedding より上位の層が言語に依存しているかどうかを調査することを動機とし，学習済み英語 BERT のパラメータをベースとして，Transformer 層を全て凍結した状態で Embedding 層のみを日本語のコーパスで学習することで日本語向けのモデルとなるように学習することで日本語向け BERT を構築した．Amazon レビューデータの感情分析タスクによって正解率を他のモデルと比較したところ，提案手法によって構築されたモデルは Multiligual BERT の精度に迫ることが分かり，Embedding 層のみの調整によって既存の高機能なモデルを別の言語にある程度適応させることができることを示した．一方で，東北大 BERT の精度には全体的に及ばなかった事から，やはり Transformer の Embedding 層より上位の層は言語に依存しているのではないかという考察に至った．

今後の展望としては，Transformer 全層を凍結して日本語の Embedding 層のみを学習する際のコーパス量や学習方法を変更すること，英語から日本語へと転送させるのではなく，語順などの構文が日本語により近い言語から日本語へと転送させることが考えられる．

謝辞

本研究を進めるにあたって、多くのご指導，ご協力を頂いた指導教員の新納浩幸教授に深く感謝いたします．また，日々の活動を通して多くの知識や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] Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. On the cross-lingual transferability of monolingual representations. *CoRR*, Vol. abs/1910.11856, , 2019.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc., 2017.

[4] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, Vol. 26. Curran Associates, Inc., 2013.

[6] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen,

Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, Vol. abs/1907.11692, , 2019.

[7] Telmo Pires, Eva Schlinger, and Dan Garrette. How multilingual is multilingual bert? *CoRR*, Vol. abs/1906.01502, , 2019.

付録

A　本実験で使用したプログラム

4.1 節で説明した部分で使用した，Wikiextractor でテキストクリーニングを行った後の日本語 Wikipedia データの集まりに対して前処理を行い，1 文 1 行のテキストデータとなるように整形するソースコードを A.1 に示す.

ソースコード A.1: TestPreprocess.py

```
# 日本語 Wikipedia データを整形する

import glob
from bs4 import BeautifulSoup

with open('./tmp.txt','w') as f:
    for directory in glob.glob('./OUTPUT/*'):
        for name in glob.glob(directory+'/*'):
            with open(name, 'r') as r:
                for line in r:
                    # title を削除する
                    if '<doc␣' in line:
                        next(r)
                        next(r)
                    elif '</doc>' in line:
                        f.write('\n')
                        continue
                    else:
                        # 空白・改行削除、大文字を小文字に変換
                        text = BeautifulSoup(line.strip()).text.lower
                            ()
                        f.write(text)

```

```
print("Finish␣Step1...")

input_file_path = 'tmp.txt'
output_file_path = 'JPWikiCorpus.txt'

# JPWikiCorpus.txt に書き込む
with open(input_file_path, 'r', encoding='utf-8') as input_file, open(
    output_file_path, 'w', encoding='utf-8') as output_file:
    for line in input_file:
        # 行から文を抽出し、１文１行ずつ JPWikiCorpus.txt に書き込む
        sentences = line.split('。')
        for sentence in sentences:
            if sentence.strip(): # 空白行を無視
                output_file.write(sentence.strip() + '。\n')

print("Finish...")
```

4.2 節で説明した部分で使用した，モデルの事前学習を行うソースコードを A.2 に示す.

ソースコード A.2: PretrainBert.py

```
# BERT の事前学習

# GPU の使用
import torch
device = "cuda:0" if torch.cuda.is_available() else "cpu"
print(device)


# Tokenizer の設定
from transformers import BertJapaneseTokenizer, BertModel

Jbert = BertModel.from_pretrained("cl-tohoku/bert-base-japanese")
tokenizer = BertJapaneseTokenizer.from_pretrained("cl-tohoku/bert-base-
    japanese") #東北大 BERT の
    tokenizer

text = "如何なるものを描かんと欲するかとの御質問であるが、私は、如何なる
    ものをも書きたいと思う。"
print("tokenizer␣反映確認")
print(tokenizer.tokenize(text)) # tokenizer の反映確認

```

```
19
20 # モデルの設定
21 import copy
22 from transformers import BertForMaskedLM
23
24 EbertMLM = BertForMaskedLM.from_pretrained("bert-base-uncased")
25
26 EbertMLM = EbertMLM.to(device)
27
28 EbertMLM.resize_token_embeddings(32000)
29 EbertMLM.train()
30 EbertMLM.bert.embeddings.word_embeddings.weight.requires_grad = False
31 EbertMLM.bert.embeddings.word_embeddings.weight = copy.deepcopy(Jbert.
       embeddings.word_embeddings.weight)
32
33 print("config␣確認")
34 print(EbertMLM.config)
35
36
37 # パラメータの凍結
38 for param in EbertMLM.bert.parameters():
39   param.requires_grad = False
40
41
42 # word_embedding のみ更新
43 EbertMLM.bert.embeddings.word_embeddings.weight.requires_grad = True
44
45 print("パラメータ凍結完了")
46
47
48 # 事前学習用データセットの準備
49 from transformers import LineByLineTextDataset
50
51 # テキストを１行ずつ読み込んでトークンに変換
52 dataset = LineByLineTextDataset(
53       tokenizer=tokenizer,
54       file_path='JPWikiCorpus.txt', # コーパス
55       block_size=512, # tokenizer の max_length
56 )
57
58 from transformers import DataCollatorForLanguageModeling
```

```
59
60 # データセットからサンプルのリストを受け取り、テンソルの辞書としてバッチ
       に照合
61 data_collator = DataCollatorForLanguageModeling(
62     tokenizer=tokenizer,
63     mlm=True,
64     mlm_probability= 0.15
65 )
66
67 # 事前学習
68 from transformers import TrainingArguments
69 from transformers import Trainer
70
71 # 事前学習に関するパラメータの設定
72 training_args = TrainingArguments(
73     output_dir= '231201jpenBERT/',
74     overwrite_output_dir=True,
75     num_train_epochs=10,
76     per_device_train_batch_size=16, # ではメモリ不足 32
77     save_steps=10000,
78     save_total_limit=2,
79     prediction_loss_only=True
80 )
81
82 # 事前学習するインスタンス
83 trainer = Trainer(
84     model=EbertMLM,
85     args=training_args,
86     data_collator=data_collator,
87     train_dataset=dataset
88 )
89
90 trainer.train()
91 trainer.save_model('231201jpenBERT/')
```

4.3 節で説明した部分で使用した．５段階のレーティングを Positive と Negative の 2 値に変換したのち、tsv ファイルに出力するソースコードを A.3 に示す.

ソースコード A.3: Mkldcc.py

```
1 # データをファイルに整形するプログラム tsv
2 # 段階のレーディングを値に変換する 52
```

```
3 import re
4
5 with open('train.review') as f:
6 #with open('test.review') as f:
7   data = f.read()
8   data = data.replace('\n', '').replace('\r', '')
9
10  reviews = re.findall(pattern=r'<item>(.+?)</item>',string=data) #
        rating はあるのに text がない感想を取り
        除く
11
12  alldata = []
13  for item in reviews:
14    raiting = re.findall(pattern=r'<rating>(.+?)</rating>',string=item
          )
15    text = re.findall(pattern=r'<text>(.+?)</text>',string=item)
16    text = '\n'.join(text)
17
18    raiting = int(float(raiting[0])) - 1 # ５段階の評価を取得
19    label = 0 if raiting >= 3 else 1 # ポジティブを０，ネガティブを１
20
21    text = str(label) + "\t" + text
22
23    alldata.append(text)
24
25
26 with open('train100.tsv','w',encoding='utf-8') as f:
27     # train は先頭から 100 個を使用
28     for i in range(100):
29         f.write(alldata[i])
30         f.write('\n')
31
32 with open('valid100.tsv','w',encoding='utf-8') as f:
33     # は先頭から train101 〜200 番目を使用
34     for i in range(100,200):
35         f.write(alldata[i])
36         f.write('\n')
37
38 """
39 with open('test.tsv','w',encoding='utf-8') as f:
40     # は全て使用 test
```

```
41      for i in range(len(alldata)):
42          f.write(alldata[i])
43          f.write('\n')
44  """
```

tsv ファイルを pickle ファイルに出力するソースコードを A.4 に示す.

ソースコード A.4: Mkdata.py

```
1  from transformers import BertJapaneseTokenizer, BertTokenizer
2  import pickle
3  import re
4
5  #tknz = BertJapaneseTokenizer.from_pretrained('cl-tohoku/bert-base-
       japanese')
6  tknz = BertTokenizer.from_pretrained('bert-base-multilingual-cased')
7
8  xdata, ydata = [],[]
9  #with open('train100.tsv','r',encoding='utf-8') as f:
10 #with open('valid100.tsv','r',encoding='utf-8') as f:
11 with open('test.tsv','r',encoding='utf-8') as f:
12     for line in f:
13         line = line.rstrip()
14         result = re.match('^(\d+)\t(.+?)$', line)
15         if result:
16             ydata.append(int(result.group(1)))
17             sen = result.group(2)
18             tid = tknz.encode(sen)
19             if (len(tid) > 512): # 最大長は512
20                 tid = tid[:512]
21             xdata.append(tid)
22
23 # xtrain が単語 id 列のリストの集合
24 #with open('xtrain.pkl','bw') as fw:
25 #with open('xvalid.pkl','bw') as fw:
26 with open('xtest.pkl','bw') as fw:
27     pickle.dump(xdata,fw)
28
29 # ytrain がラベル
30 #with open('ytrain.pkl','bw') as fw:
31 #with open('yvalid.pkl','bw') as fw:
32 with open('ytest.pkl','bw') as fw:
```

```
33      pickle.dump(ydata,fw)
```

4.3.2 節で説明した部分で使用した，モデルをファインチューニングするソースコードを A.5 に示す．また，early stopping 部分の実装にはこちらの記事[*1] を参考にした．

ソースコード A.5: DocclsEarly.py

```
1  import torch
2  import torch.nn as nn
3  import torch.optim as optim
4  import torch.nn.functional as F
5  from transformers import BertModel, BertConfig
6
7  import numpy as np
8  import pickle
9
10 class EarlyStopping:
11     """earlystopping クラス"""
12     """参考サイト：
           https://qiita.com/ku_a_i/items/ba33c9ce3449da23b503"""
13
14     def __init__(self, patience=5, verbose=False, path='best_model.
           model'):
15         """引数：最小値の非更新数カウンタ、表示設定、モデル格納 path"""
16
17         self.patience = patience #設定ストップカウンタ
18         self.verbose = verbose #表示の有無
19         self.counter = 0 #現在のカウンタ値
20         self.best_score = None #ベストスコア
21         self.early_stop = False #ストップフラグ
22         self.val_loss_min = np.Inf #前回のベストスコア記憶用
23         self.path = path #ベストモデル格納 path
24
25     def __call__(self, val_loss, model):
26         """特殊
27         (call)メソッド実際に学習ループ内で最小を更新したか否かを計算させ
               る部分
28         loss
29         """
30         score = -val_loss
31
```

[*1] https://qiita.com/m__k/items/6f71ab3eca64d98ec4fc

```
32          if self.best_score is None: #1Epoch 目の処理
33              self.best_score = score #1Epoch 目はそのままベストスコアとして記録する
34              self.checkpoint(val_loss, model) #記録後にモデルを保存してスコア表示する
35          elif score < self.best_score: # ベストスコアを更新できなかった場合
36              self.counter += 1 #ストップカウンタを +1
37              if self.verbose: #表示を有効にした場合は経過を表示
38                  print(f'EarlyStopping␣counter:␣{self.counter}␣out␣of␣{self.patience}') #現在のカウンタを表示する
39              if self.counter >= self.patience: #設定カウントを上回ったらストップフラグを True に変更
40                  self.early_stop = True
41          else: #ベストスコアを更新した場合
42              self.best_score = score #ベストスコアを上書き
43              self.checkpoint(val_loss, model) #モデルを保存してスコア表示
44              self.counter = 0 #ストップカウンタリセット
45
46      def checkpoint(self, val_loss, model):
47          '''ベストスコア更新時に実行されるチェックポイント関数'''
48          if self.verbose: #表示を有効にした場合は、前回のベストスコアからどれだけ更新したか？を表示
49              print(f'Validation␣loss␣decreased␣({self.val_loss_min:.6f}␣-->␣{val_loss:.6f}).␣␣Saving␣model␣...')
50          torch.save(model.state_dict(), self.path) #ベストモデルを指定した path に保存
51          self.val_loss_min = val_loss #その時の loss を記録する
52
53
54  device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
55
56  # Data setting
57
58  with open('xtrain.pkl','br') as fr:
59      xtrain = pickle.load(fr)
60
61  with open('ytrain.pkl','br') as fr:
62      ytrain = pickle.load(fr)
63
64  with open('xvalid.pkl','br') as fr:
```

```
    xvalid = pickle.load(fr)

with open('yvalid.pkl','br') as fr:
    yvalid = pickle.load(fr)

# Define model
bert = BertModel.from_pretrained('bert-base-multilingual-cased')


# 学習用
class DocCls(nn.Module):
    def __init__(self,bert):
        super(DocCls, self).__init__()
        self.bert = bert
        #self.cls=nn.Linear(768,9)
        self.cls=nn.Linear(768,2)
    def forward(self,x):
        bout = self.bert(x)
        bs = len(bout[0])
        h0 = [ bout[0][i][0] for i in range(bs)]
        h0 = torch.stack(h0,dim=0)
        return self.cls(h0)


# 検証用
class DocCls2(nn.Module):
    def __init__(self,bert):
        super(DocCls2, self).__init__()
        self.bert = bert
        #self.cls=nn.Linear(768,9)
        self.cls=nn.Linear(768,2)
    def forward(self,x):
        bout = self.bert(x)
        bs = len(bout[0])
        h0 = [ bout[0][i][0] for i in range(bs)]
        h0 = torch.stack(h0,dim=0)
        return self.cls(h0)


# model generate, optimizer and criterion setting

```

```
106 net = DocCls(bert).to(device)
107 net2 = DocCls2(bert).to(device)
108
109 optimizer = optim.SGD(net.parameters(),lr=0.001)
110 criterion = nn.CrossEntropyLoss()
111
112 earlystopping = EarlyStopping(patience=3, verbose=True) # loss が３連
        続で更新されなかったら EarlyStopping
113
114 # Learn
115
116 net.train()
117
118 earlycheckloss = 0.0
119
120 for ep in range(30):
121     lossK = 0.0
122     for i in range(len(xtrain)):
123         x = torch.LongTensor(xtrain[i]).unsqueeze(0).to(device)
124         y = torch.LongTensor([ ytrain[i] ]).to(device)
125         out = net(x)
126         loss = criterion(out,y)
127         lossK += loss.item()
128         if (i % 50 == 0):
129             print(ep, i, lossK)
130
131             lossK = 0.0
132         optimizer.zero_grad()
133         loss.backward()
134         optimizer.step()
135     outfile = "doccls-" + str(ep) + ".model"
136     torch.save(net.state_dict(),outfile)
137     print(outfile, "␣saved")
138
139
140     net2.load_state_dict(torch.load(outfile))
141     net2.eval()
142     vlossK = 0.0
143     for i in range(len(xvalid)):
144         vx = torch.LongTensor(xvalid[i]).unsqueeze(0).to(device)
145         vy = torch.LongTensor([ yvalid[i] ]).to(device)
```

```
146         out2 = net2(vx)
147         vloss = criterion(out2,vy)
148         vlossK += vloss.item()
149         if (i % 50 == 0):
150           print(ep, i, vlossK)
151           earlycheckloss = lossK
152           vlossK = 0.0
153
154     earlystopping(earlycheckloss, net)
155     if earlystopping.early_stop:
156       print("Early␣Stopping!")
157       break
```

4.3.2 節で説明した部分で使用した，テストデータでモデルを評価するソースコードを A.6 に示す.

ソースコード A.6: DocclsTest.py

```
1  import torch
2  import torch.nn as nn
3  import torch.optim as optim
4  import torch.nn.functional as F
5  from torch.utils.data import Dataset, DataLoader
6  from torch.nn.utils.rnn import pad_sequence
7  from transformers import BertModel, BertConfig
8
9  import numpy as np
10 import pickle
11 import sys
12
13 argvs = sys.argv
14 argc = len(argvs)
15
16 config = BertConfig.from_pretrained('bert-base-multilingual-cased')
17 bert = BertModel(config=config)
18
19 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu"
       )
20
21 # Data Setting
22 with open('xtest.pkl','br') as fr:
23     xtest = pickle.load(fr)
```

```
24
25 with open('ytest.pkl','br') as fr:
26     ytest = pickle.load(fr)
27
28 # Define model
29
30 class DocCls(nn.Module):
31     def __init__(self,bert):
32         super(DocCls, self).__init__()
33         self.bert = bert
34         #self.cls=nn.Linear(768,9)
35         self.cls=nn.Linear(768,2)
36     def forward(self,x):
37         bout = self.bert(x)
38         bs = len(bout[0])
39         h0 = [ bout[0][i][0] for i in range(bs)]
40         h0 = torch.stack(h0,dim=0)
41         return self.cls(h0)
42
43 # model generate, optimizer and criterion setting
44
45 net = DocCls(bert).to(device)
46 #net.load_state_dict(torch.load(argvs[1]))
47 net.load_state_dict(torch.load('best_model.model'))
48
49 # Learn
50
51 real_data_num, ok = 0, 0
52 net.eval()
53 with torch.no_grad():
54     for i in range(len(xtest)):
55         x = torch.LongTensor(xtest[i]).unsqueeze(0).to(device)
56         ans = net(x)
57         ans1 = torch.argmax(ans,dim=1).item()
58         if (ans1 == ytest[i]):
59             ok += 1
60         real_data_num += 1
61 print(ok, real_data_num, ok/real_data_num)
```