令和 3 年度茨城大学工学部情報工学科卒業研究論文

単語分散表現の固定化による日本語 BERT の構築

所属　　情報工学科

著者　　菅波新 (18T4048A)

指導教員　　新納浩幸教授

令和 4 年 2 月 1 日 (火)

令和 3 年度茨城大学工学部情報工学科卒業研究論文

単語分散表現の固定化による日本語 BERT の構築

著者

菅波新 (18T4048A)

指導教員

新納浩幸教授

論文要旨

BERT [1] は自然言語処理分野における高性能な事前学習済みモデルであるが，モデルサイズが巨大なため，構築に多大な時間や計算資源がかかってしまうという問題がある．また，事前学習済みモデルの高性能化のためにデータセットやモデルサイズを増加させることによる，モデルの構築時間や費用の増大化も問題視されている [2] .

そこで本稿では BERT の構築時間を削減するために単語分散表現の固定化による日本語 BERT の構築を提案する．具体的には，word2vec を利用して単語分散表現をあらかじめ学習しておき，その単語分散表現を BERT の Token Embedding として固定することで日本語 BERT を構築する．これにより本来 BERT の構築時に行われる単語分散表現の学習にかかる時間が削減され，構築時間が軽減されるのではないかと考えた.

実験では，128 次元 24 層の日本語 BERT と 1024 次元 4 層の日本語 BERT をそれぞれ従来の方法と提案手法により構築し，モデルの構築時間と日本語ニュース記事に対する文書分類タスクでの精度を比較することにより提案手法の効果を検証した．実験の結果，128 次元 24 層では，精度は改善されなかったが，1024 次元 4 層では精度が改善された．また，どちらのモデルも提案手法によって構築時間が短縮されることを確認し，構築時間の削減効果を実証できた.

Bachelor's Thesis in Scholastic 2022,
Department of Computer and Information Sciences, Ibaraki University

Construction of Japanese BERT with Fixed Token Embeddings

Author

Arata Suganami (18T4048A)

Adviser

Prof. Hiroyuki Shinnou

Abstract

BERT [1] is a high-performance pre-trained model in the field of natural language processing, but its large model size requires a lot of time and computational resources to pretrain. Another problem is the increase of model construction time and cost due to the increase of dataset and model size as the performance of the pre-trained model increases [2].

In this paper, we propose to construct Japanese BERT by fixed token embedding in order to reduce the construction time of BERT. Specifically, we propose to construct Japanese BERT by learning word embeddings in advance using word2vec, and then fixing the word Embeddings as the Token Embedding for BERT. We thought that this would reduce the time needed to learn the word embedding and thus reduce the construction time of BERT.

In the experiments, we constructed 128-dimensional 24-layer Japanese BERT and 1024-dimensional 4-layer Japanese BERT using the conventional method and the proposed method, and verified the effectiveness of the proposed method by comparing the model construction time and the accuracy in the document classification task for Japanese news articles. The experimental results showed that the accuracy of the proposed method did not improve for the 128-dimensional 24-layer model, but improved for the 1024-dimensional 4-layer model. We also confirmed that the proposed method reduces the construction time for both models, demonstrating the effectiveness of the proposed method in reducing the construction time.

目次

表目次

図目次

第 1 章

序論

BERT [1] は自然言語処理分野における高性能な事前学習済みモデルであるが，モデルサイズが巨大なため，構築に多大な時間や計算資源がかかってしまうという問題がある．また，事前学習済みモデルの高性能化を実現するためにデータセットやモデルサイズを増加させることによる，モデルの構築時間や費用の増大化も問題視されている [2] .

そこで本稿では BERT の構築時間を削減するために単語分散表現の固定化による日本語 BERT の構築を提案する．具体的には，word2vec [3] [4] を利用して単語分散表現をあらかじめ学習しておき，その単語分散表現を BERT の Token Embedding として固定することで日本語 BERT を構築する．これにより，本来ならば BERT の構築時に行われる単語分散表現の学習にかかる時間が削減され，構築時間が軽減されるのではないかと考えた.

実験では，128 次元 24 層の日本語 BERT と 1024 次元 4 層の日本語 BERT をそれぞれ従来の方法と提案手法により構築し，モデルの構築時間と日本語ニュース記事に対する文書分類タスクでの精度を比較することにより提案手法の効果を検証した．実験の結果，128 次元 24 層の BERT では，精度は改善されなかったが，1024 次元 4 層の BERT では精度が改善された．また，どちらの BERT においても提案手法により構築時間が短縮されることを確認した.

第 2 章

関連研究

2.1 単語分散表現

単語分散表現とは単語を表現する実数ベクトルのことである．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．意味が似ている単語は近く，意味が似ていない単語は遠くなるように各単語がベクトルで表される．単語をベクトルで表すことにより，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．単語分散表現は以下で説明する BERT や word2vec 等を利用して構築することができる．

2.2 BERT

BERT [1]（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 [5] 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 fine tuning 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．

2.2.1　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 [1] 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 [1] 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．

2.2.2　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 2.1 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．

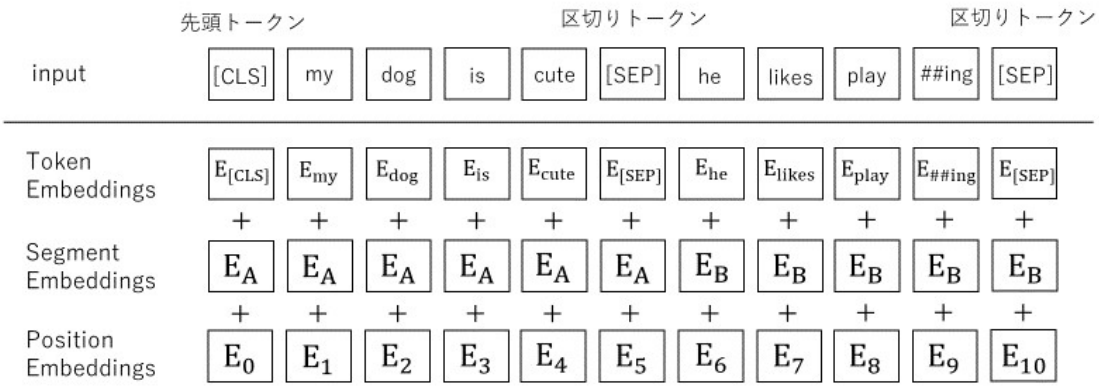

図 2.1: BERT の入力表現

出典：論文 [1] 中の Figure2 を元に筆者が作成

2.3　word2vec

word2vec [3] [4] 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 [3] 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．

2.3.1　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．本研究では word2vec の CBOW モデルにより単語分散表現を構築し，日本語 BERT を構築する際に Token Embedding に固定することで BERT による単語分散表現の学習の省略を試みる．

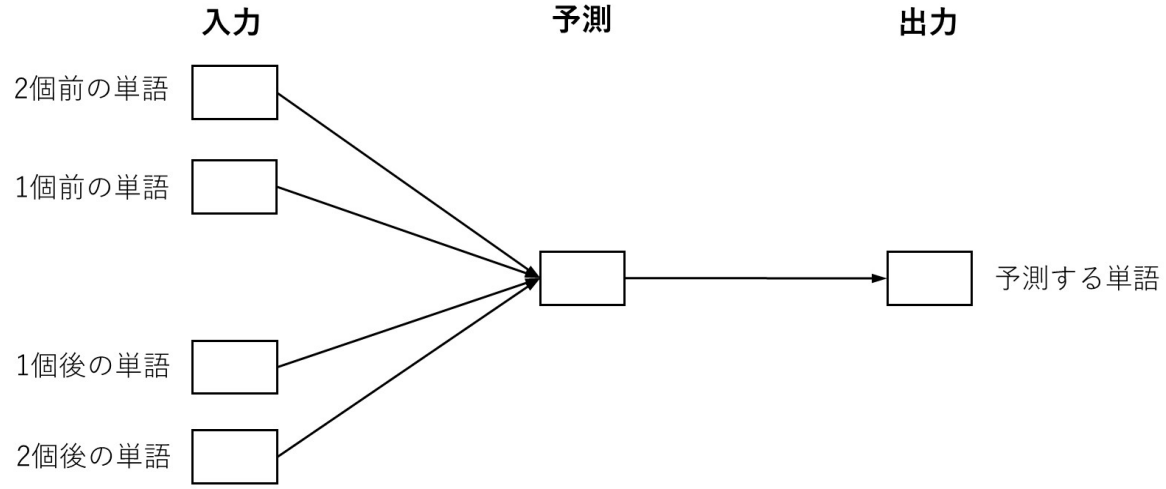

図 2.2: CBOW のイメージ図

出典：論文 [3] 中の figure1 を元に筆者が作成

2.3.2　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．

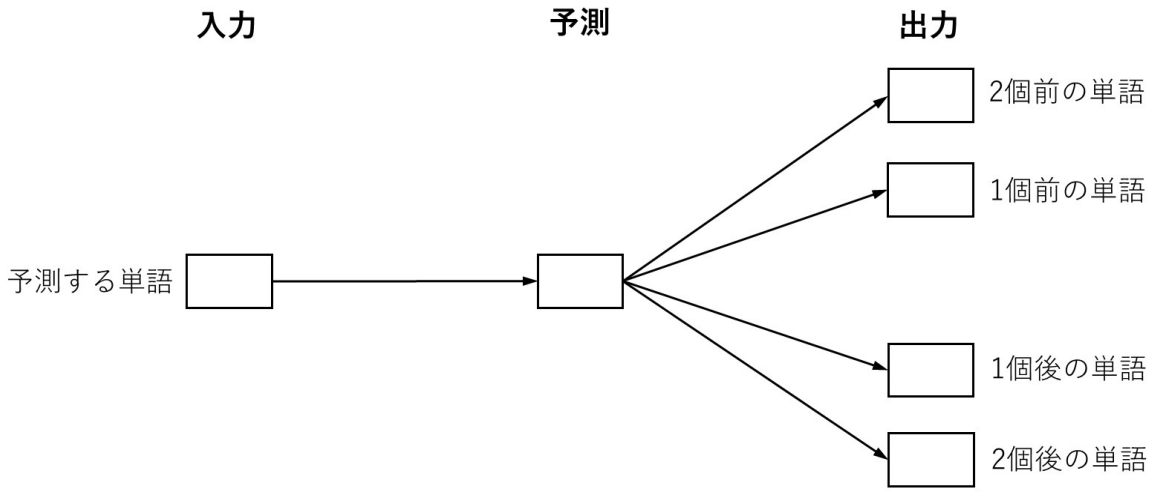

図 2.3: Skip-gram のイメージ図

出典：論文 [3] 中の figure1 を元に筆者が作成

2.4　モデルの巨大化

論文 [2] ではハードウェアの最適化やフレームワークの最適化等により浮動小数点演算のコストが削減されているにもかかわらず，NLP モデルのトレーニングコストが全体的に増大していると述べている．その要因としてモデルサイズが増大していることを挙げ，データセットやパラメータ数に加え，埋め込み次元や層数が年々増大していることを示している．表 2.1 に示す，GPT-2 [6] や GPT-3 [7] といったモデルは BERT 以降に発表されたモデルで，BERT と同様に Transformer をベースにしたモデルである．非常に大きなコーパスを使用して巨大なモデルで学習することにより，fine tuning なしで様々な自然言語処理タスクに適用できる優れた汎用性を得た．

表 2.1: 自然言語処理モデルの巨大化

	次元数	層数	パラメータ数 (億)
BERT-base(2018)	768	12	1.1
BERT-large(2018)	1024	24	3.4
GPT-2(2019)	1600	48	15
GPT-3(2020)	12288	96	1750

2.5 DistilBERT

BERT を使用する際にかかる計算資源や学習時間の削減を目的とし，BERT を小型化したモデルに DistilBERT というモデルがある．DistilBERT [8] は BERT を教師モデルとして小型化した 768 次元 6 層からなる事前学習済みモデルであり，BERT-base の性能を 97% 維持したまま，パラメータ数を 40% 削減し，60% の高速化に成功している．

DistilBERT は Knowledge distillation(知識の蒸留) と呼ばれる手法を用いてコンパクトな生徒モデルが教師モデルの分布を再現するように事前学習を行う．Knowledge Transfer は triple loss と呼ばれる以下の 3 つの損失関数の和によって実現され，この損失関数の和を減らしていくことで学習していく．

- L_{ce}(distillation loss)
  生徒モデルが教師モデルの分布を再現するようにするための損失であり，以下の式で表される．
  $$L_{ce} = \sum_i t_i \times \log(s_i)$$
  t_i は教師モデルでの予測確率で s_i 生徒モデルでの予測確率である i は i 番目の単語を表す．
- L_{mlm}(masked language modeling loss)
  2.1.1 節で説明したものと同様の Masked Language Model によって求められる損失である．
- L_{cos}(cosine embedding loss)
  生徒モデル (DistilBERT) と教師モデル (BERT) の中間層のベクトルのコサイン類似度による損失である．コサイン類似度とは 2 つのベクトルが作る角の余弦値のことであり，その角の角度が小さいほどそれらのベクトルは類似していることを意味する．この損失は生徒モデルの埋め込み次元の表現を教師モデルの埋め込み次元の表現に近づける役割を持つ．

2.6　MobileBERT

DistilBERT と同様に BERT を使用する際にかかる計算資源や学習時間の削減を目的とし, BERT を小型化したモデルに MobileBERT というモデルがある。MobileBERT [9] は BERT-Large を教師モデルとして小型化した 128 次元 24 層からなる事前学習済みモデルであり，スマートフォンである Google Pixel 4 上で動作するコンパクトな BERT である．BERT-base と比べてモデルサイズが 4.3 倍小さく，5.5 倍高速でありながら，GLUE スコアで 0.6 だけ下回る小型ながら高い精度を持つモデルである．

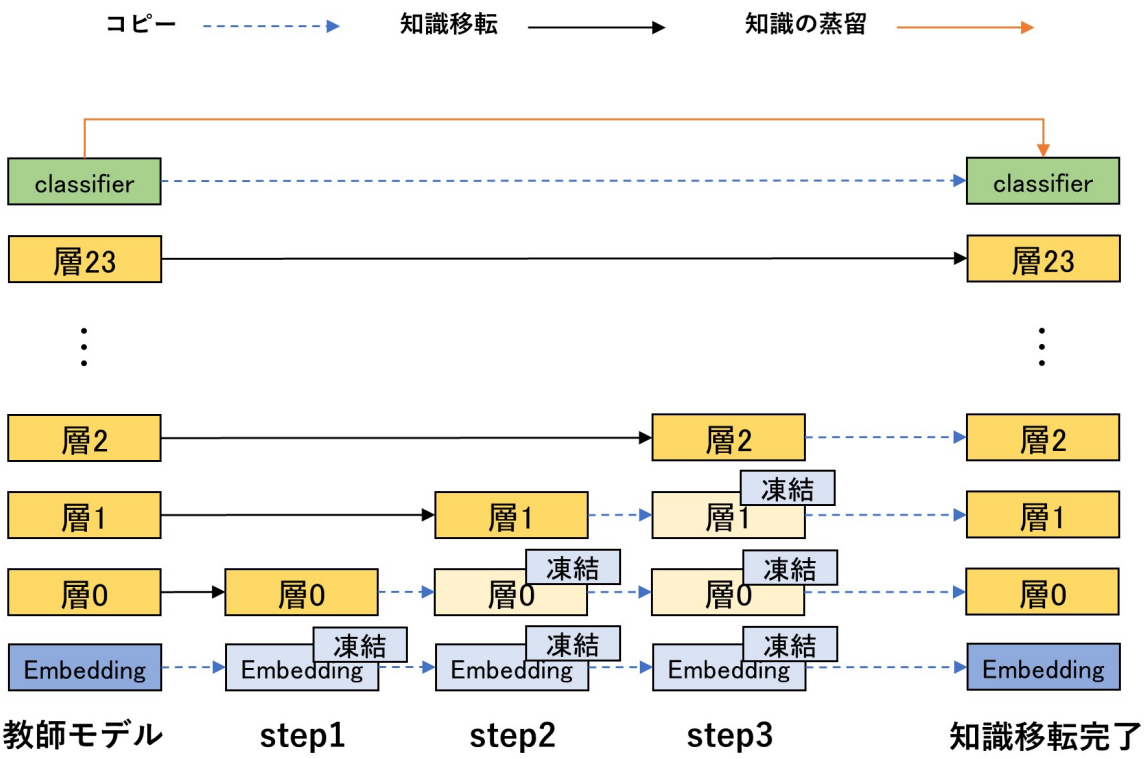

図 2.4: Progressive Knowledge Transfer

出典：論文 [9] の Figure 2 を元に筆者が作成

MobileBERT は逆ボトルネック構造と呼ばれる埋め込み表現を線形変換する機構を追加した特別な BERT-Large を構築したのち，MobileBERT への知識移転を行う際に，図 2.4 に示す，Progressive Knowledge Transfer と呼ばれる下位層から段階的に知識移転す

る手法を用いる．これは下位層からの学習のずれが上位層での知識移転に影響を及ぼすのを防ぐために行われ，各層を段階的に訓練していく際に，現段階で訓練している層の下の層のパラメータを全て凍結する手法である．これにより MobileBERT は教師モデルである BERT-Large の性能を知識移転によって受け継ぎつつ，モデルの小型化に成功している．

第 3 章

提案手法

3.1 単語分散表現の固定化

BERT の構築時間を削減するために単語分散表現の固定化を提案する．以下の手順により実現される．

- (1)：分かち書き済みのテキストから word2vec を利用して単語分散表現を構築する．これによりテキスト内の全単語に対する単語分散表現を得る．

- (2)：BERT の学習に使う単語の id に word2vec で学習した単語分散表現の単語の id をあわせておく．

  (実例)
  ・word2vec により学習した単語分散表現では「犬」の token id は 2408
  ・BERT 構築時に作られる辞書では「犬」の token id は 2657

  「犬」という単語が入力された際，BERT に入力される token id は 2657 としつつ，word2vec により学習済みの単語分散表現を参照するようにする．

- (3)：(1) で構築しておいた単語分散表現を，BERT を構築する際に Token Embedding として固定し，Token Embedding のパラメータを新しく学習しないように凍結する．

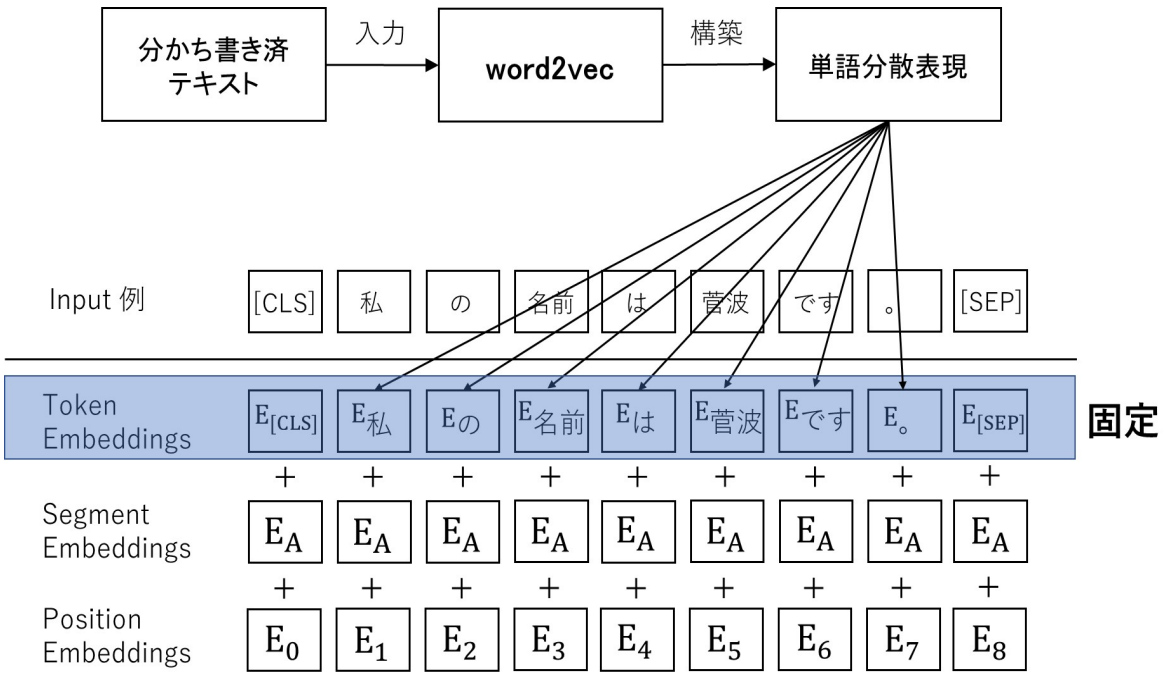

図 3.1: 単語分散表現の固定

補足：E は入力の埋め込み表現 (単語分散表現) を表す

提案手法のイメージを図 3.1 に示した．例として「私 の 名前 は 菅波 です 。」という単語列が BERT に与えられた際に，1 から単語分散表現を学習していくのではなく，あらかじめ word2vec を利用して学習しておいた単語分散表現を Token Embedding の入力単語に対応する場所に渡し，パラメータを凍結させることによって単語分散表現を固定する様子を表している．

Token Embedding のパラメータを凍結することで，word2vec によって学習済みの単語分散表現で固定され，BERT 構築時の単語分散表現の学習が省略されるため，構築時間の削減が期待できる．

3.2　構築する日本語 BERT の構成

次元数が多く層の数も多い BERT-base や BERT-large を 0 から構築するには多くの時間がかかってしまう。東北大学が公開した日本語 BERT である bert-japanese[*1]の事

[*1] https://github.com/cl-tohoku/bert-japanese

前学習には CloudTPU を使用して BERT-base で 5 日，BERT-large で 14 日かかる．そこで本研究では，代わりに小型の日本語 BERT を構築することにより実験を行う．BERT-large と比べ，次元数が少なく層数が等しい 128 次元 24 層の日本語 BERT と次元数が等しく層数が少ない 1024 次元 4 層の日本語 BERT を従来手法と提案手法によって構築し，構築時間と文書分類タスクでの精度を比較することで検証する．

第 4 章

実験

4.1 単語分散表現の構築

コーパスとして 11 月 1 日時点の日本語 Wikipedia のダンプデータ[*1]を使用した．wikiextractor を用いてテキストクリーニングを行い，テキストを前処理後，複数の txt ファイルを結合し一つのテキストにした．完成したテキストを日本語 BERT の”cl-tohoku/bert-base-japanese”の tokenzer を使ってテキスト内の文章を単語に分割し，分かち書き済みのテキストを作成した．

Python ライブラリ ”gensim” の word2vec を利用して分かち書き済みのテキストから 128 次元の単語分散表現と 1024 次元の単語分散表現を構築した．単語分散表現の構築時に使用した学習アルゴリズムは両方とも CBOW である．

4.2 日本語 BERT の構築

GitHub[*2]に公開されている日本語 BERT を構築するプログラムを使って，128 次元 24 層の日本語 BERT と 1024 次元 4 層の日本語 BERT を従来の方法によりそれぞれ 8 エポック分構築した．参照元は.ipynb 形式のプログラムとなっているが，まとめて.py 形式のプログラムにして一括で実行した．また、4.1 節で構築した単語分散表現を Token Embedding に固定して日本語 BERT を構築するようにプログラムを改変し，128 次元 4 層の日本語 BERT と 1024 次元 4 層の日本語 BERT を提案手法により再びそれぞれ 8 エ

[*1] https://dumps.wikimedia.org/jawiki/latest/ 内の”jawiki-latest-pages-articles.xml.bz2”
[*2] https://github.com/Kosuke-Szk/ja_text_bert/blob/master/text_bert.ipynb

ポック分構築した．モデル構築時，各条件で 1 エポックごとにモデル構築時間を計測した．ここまで説明した日本語 BERT 構築までの流れを図 4.1 に示す．

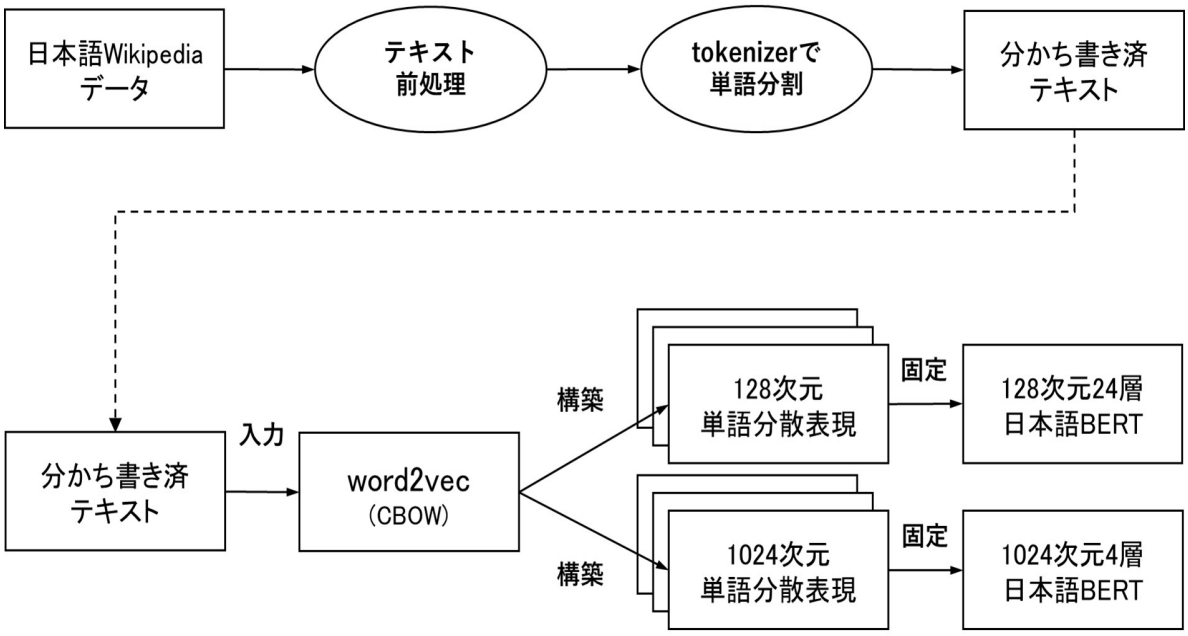

図 4.1: 日本語 BERT の構築

4.3　モデルの評価

日本語 BERT を構築した後，文書分類タスクでの正解率を比較することでモデルの評価を行った．評価データには株式会社ロンウィットから公開されている livedoor ニュースコーパス[*3]を使用した．記事のカテゴリに応じて 0 から 8 までのラベルを付与し，コーパスに含まれる全記事 7376 個を表 4.1 のように訓練データ (train)，検証データ (valid)，テストデータ (test) に振り分けた．

訓練データを用いてモデルを学習し，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用いて正解率を計測した．図 4.2 に示すように学習→検証のループを最大 15 エポックまでに設定し，検証データに対する正解率の最高値が 5 回更新されなかった場合に early stopping としてモデルの学習を途中で止め，その時点までで正解率が最高だっ

[*3] https://www.rondhuit.com/download.html#ldcc

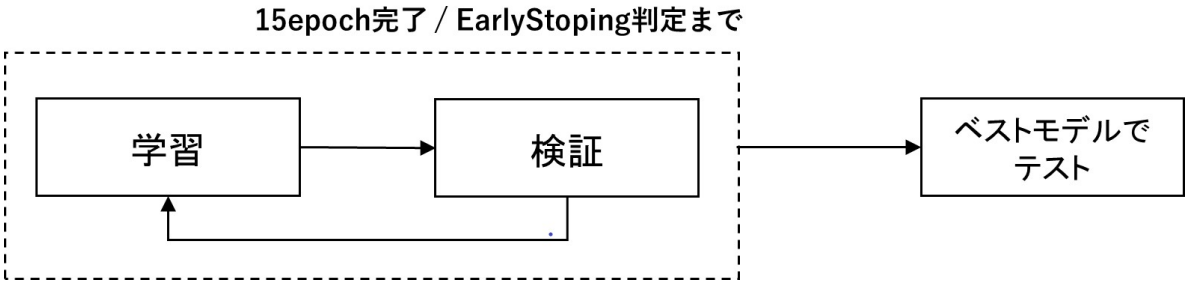

図 4.2: モデルの評価

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使用して記事の本文から記事が属するカテゴリを予測する文書分類タスクを行い，正解率を計測した．

表 4.1: 記事データの内訳

ラベル	カテゴリ	train	valid	test	sum
0	独女通信	87	87	697	871
1	IT ライフハック	87	87	697	871
2	家事チャンネル	86	86	693	865
3	livedoor HOMME	51	51	410	512
4	MOVIE ENTER	87	87	697	871
5	Peachy	84	84	675	843
6	エスマックス	87	87	697	871
7	Sports Watch	90	90	721	901
8	トピックニュース	77	77	617	771
合計		736	736	5904	7376

4.4　実験結果

4.4.1　モデルの構築時間

日本語 BERT の構築にかかった 1 エポックあたりの平均時間を表 4.2 に示す．モデルの H は埋め込み次元 (単語分散表現の次元) を表し，L は層数を表している．N は従来の手法 (Normal) により構築されたことを示し，W は提案手法 (Word2vec による単語分散表現の固定) により構築されたことを示す．例えば，”H1024-L4-W”は提案手法により構築された 1024 次元 4 層の日本語 BERT であることを示す．

提案手法により，1 エポック当たり 128 次元 24 層では 4 分，1024 次元 4 層では，27 分短縮された．

表 4.2: 1 エポックあたりの構築時間

モデル	構築時間
H128-L24-N	16h49m
H128-L24-W	16h45m
H1024-L4-N	18h05m
H1024-L4-W	17h38m

4.4.2　文書分類タスクにおける正解率

構築した日本語 BERT を用いて行った文書分類タスクでの正解率を表 4.3 に示す．太字は各条件におけるベストの正解率である．提案手法により，128 次元 24 層では精度が改善されなかったが，1024 次元 4 層では精度が改善された．また，今回構築した中で提案手法により構築した 1024 次元 4 層の BERT が文書分類タスクにおける正解率が最高となった．

表 4.3: 文書分類の正解率

	H128-L24-N	H128-L24-W	H1024-L4-N	H1024-L4-W
epoch0	0.7033	0.6982	0.6386	**0.7236**
epoch1	0.7100	0.7060	0.6811	0.7107
epoch2	0.6922	**0.7148**	0.6765	0.6816
epoch3	0.6865	0.6738	0.6822	0.6685
epoch4	0.7122	0.6963	0.6811	0.6992
epoch5	0.6729	0.6975	0.7041	0.7002
epoch6	0.7029	0.6897	**0.7043**	0.6753
epoch7	**0.7204**	0.7126	0.6941	0.6692

第 5 章

考察

5.1 構築時間について

表 4.2 より，構築時間が 128 次元 24 層では 4 分，1024 次元 4 層では 27 分短縮されていることから，提案手法によってモデルの構成に関わらず構築時間が減少することが分かった．これは，BERT での単語分散表現の学習が省略されたことによる構築時間の短縮だと考えられる．また，1024 次元 4 層の場合の方がより短縮されていることから，日本語 BERT の次元数が大きいほど提案手法による構築時間の削減効果が大きいのではないかと考える．

5.2 精度について

表 4.3 より，提案手法によって 128 次元 24 層のモデルでは精度の改善が見られなかったが，1024 次元 4 層のモデルでは精度の改善が見られた．また，通常の手法によって構築された日本語 BERT では，H128-L24-N の 8 エポック目のモデル，H1024-L4-N の 7 エポック目のモデルでベストスコアが記録されているのに対し，提案手法によって構築された日本語 BERT では，H128-L24-W の 3 エポック目のモデル，H1024-L4-W の 1 エポック目のモデルでベストスコアが記録されていることから，word2vec を利用して作成した単語分散表現を固定した日本語 BERT は，早いエポック数で精度が収束する傾向があると考えられる．

表 4.3 の H1024-L24-W に見られるように精度を改善しつつ早いエポック数で精度が収束されるのであれば，通常の手法と比べて，BERT を構築する際に少ないエポック数

で高い精度を持つモデルを構築することができる．これにより構築時のエポック数を減らすことができ，構築時間を大幅に削減できるのではないかと考えられる．

第 6 章

結論

本稿では，BERT の構築時間を削減するために，BERT の構築時に行われる単語分散表現の学習にかかる時間に着目し，word2vec を利用してあらかじめ学習しておいた単語分散表現を Token Embedding に固定して日本語 BERT を構築する手法を提案した．実験では 128 次元 24 層と 1024 次元 4 層の日本語 BERT を構築し，構築時間を計測したところ，どちらのモデルにおいても構築時間を削減できることが分かった．また，文書分類タスクでの精度を計測したところ，1024 次元 4 層の日本語 BERT において精度が改善され，早いエポック数で精度が収束することが分かった．これにより，単語分散表現の固定化により BERT の構築時間を削減できることを示した．

今後は BERT-base や BERT-large と同じサイズで日本語 BERT を構築し，モデルの構築時間や性能の比較を行うほか，性能を落とさず構築時間を削減できる他の手法を模索していきたい．

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] 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.

[2] Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training NLP models: A concise overview. *CoRR*, Vol. abs/2004.08900, , 2020.

[3] 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.

[4] 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.

[5] 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.

[6] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya

Sutskever. Language models are unsupervised multitask learners. 2019.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, Vol. 33, pp. 1877–1901. Curran Associates, Inc., 2020.

[8] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, Vol. abs/1910.01108, , 2019.

[9] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2158–2170, Online, July 2020. Association for Computational Linguistics.

付録

A　本実験で使用したプログラム

word2vec を利用して分かち書き済みのテキストから単語分散表現を構築するソースコードを A.1 に示す.

ソースコード A.1: use-gensim.py

```
1  # https://github.com/RaRe-Technologies/gensim/wiki/Migrating-from-
       Gensim-3.x-to-4
2  # バージョン変更に伴い、コマンドが変わったので上記のを見ること
3
4  import logging
5  import os.path
6  import sys
7
8  program = os.path.basename(sys.argv[0])
9  logger = logging.getLogger(program)
10
11 logging.basicConfig(format='%(asctime)s␣:␣%(levelname)s␣:␣%(message)s')
12 logging.root.setLevel(level=logging.INFO)
13 logger.info("running␣%s" % '␣'.join(sys.argv))
14
15
16 from gensim.models import word2vec
17
18 # モデル構築時に使った分かち書き済みのテキスト
19 sens = word2vec.LineSentence('rowsMAX.txt')
20
21 from gensim.models import Word2Vec
22 model = Word2Vec(sens, vector_size=128)
23 #model = Word2Vec(sens, vector_size=1024)
```

```
24
25 model.save('128dim1223.bin') # 128モデル dimWord2Vec
26 #model.save('1024dim1223.bin') # 1024モデル dimWord2Vec
```

構築した日本語 BERT に訓練データを fine-tuning させ，earlystopping により 1 エポックごとにモデルの検証を行うソースコードを A.2 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.2: mylearn_early.py

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  import torch
5  import torch.nn as nn
6  import torch.optim as optim
7  import torch.nn.functional as F
8  from torch.utils.data import Dataset, DataLoader
9  from torch.nn.utils.rnn import pad_sequence
10
11 # import numpy as np
12 import pickle
13 import sys
14
15 # argvs = sys.argv
16 # argc = len(argvs)
17
18 from mybert import TorchVocab, Vocab, WordVocab, BERT, BERTEmbedding,
       TransformerBlock, TokenEmbedding, PositionalEmbedding,
       SegmentEmbedding, Attention, MultiHeadedAttention
19
20 #from transformers import AutoTokenizer
21
22 #tknz = AutoTokenizer.from_pretrained("cl-tohoku/bert-base-japanese")
23
24 from transformers import BertJapaneseTokenizer
25 tknz = BertJapaneseTokenizer.from_pretrained('cl-tohoku/bert-base-
       japanese')
26
```

[*1] https://qiita.com/ku_a_i/items/ba33c9ce3449da23b503

```
27 ################################################
28
29 hidden=128
30 layers=24
31 attn_heads=8
32 seq_len=256
33
34 dropout=0.0
35
36 vocab_path='./vocab.txt'
37 vocab = WordVocab.load_vocab(vocab_path)
38
39 bert = BERT(len(vocab), hidden=hidden, n_layers=layers, attn_heads=
       attn_heads, dropout=dropout)
40
41 modelfile = '../../../data/suganami/H-128-L-24-normal-model/model/
       bertmodel2021-12-0912:10:01.961076.ep7'
42
43 bert = torch.load(modelfile)
44
45 dic = {}
46 with open('mydic.pkl','br') as fr:
47     dic = pickle.load(fr)
48
49 henkan = {}
50 with open('henkan.pkl','br') as fr:
51     henkan = pickle.load(fr)
52
53 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu"
       )
54
55 class EarlyStopping:
56     """クラス earlystopping"""
57
58     def __init__(self, patience, verbose, path):
59         """引数最小値の非更新数カウンタ、表示設定、モデル格納:path"""
60
61         self.patience = patience #設定ストップカウンタ
62         self.verbose = verbose #表示の有無
63         self.counter = 0 #現在のカウンタ値
64         self.best_score = None #ベストスコア
```

```
65          self.early_stop = False #ストップフラグ
66          self.val_acc_max = 0 #前回のベストスコア記憶用
67          self.path = path #ベストモデル格納 path
68
69      def __call__(self, val_acc, model):
70          """特殊
71          (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
72          loss
73          """
74          score = val_acc
75
76          if self.best_score is None: #1目の処理 Epoch
77              self.best_score = score #1目はそのままベストスコアとして記録する Epoch
78              self.checkpoint(val_acc, model) #記録後にモデルを保存してスコア表示する
79          elif score < self.best_score: # ベストスコアを更新できなかった場合
80              self.counter += 1 #ストップカウンタを +1
81              if self.verbose: #表示を有効にした場合は経過を表示
82                  print(f'EarlyStopping␣counter:␣{self.counter}␣out␣of␣{self.patience}') #現在のカウンタを表示する
83              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に変更 True
84                  self.early_stop = True
85          else: #ベストスコアを更新した場合
86              self.best_score = score #ベストスコアを上書き
87              self.checkpoint(val_acc, model) #モデルを保存してスコア表示
88              self.counter = 0 #ストップカウンタリセット
89
90      def checkpoint(self, val_acc, model):
91          '''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'''
92          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
93              print(f'Validation␣accuracy␣increased␣({self.val_acc_max:.6f}␣-->␣{val_acc:.6f}).␣␣Saving␣model␣...')
94          torch.save(model.state_dict(), self.path) #ベストモデルを指定したに保存 path
95          self.val_acc_max = val_acc #その時のを記録する loss
96
97
98  # DataLoader
```

```
99
100 class MyDataset(Dataset):
101     def __init__(self, xdata, ydata):
102         self.data = xdata
103         self.label = ydata
104     def __len__(self):
105         return len(self.label)
106     def __getitem__(self, idx):
107         x = self.data[idx]
108         y = self.label[idx]
109         return (x,y)
110
111 def my_collate_fn(batch):
112     images, targets= list(zip(*batch))
113     xs = list(images)
114     ys = list(targets)
115     return xs, ys
116
117 xdata = []
118 ydata = []
119
120 xval = [] # 検証用
121 yval = [] # 検証用
122
123 # データ読み込み train
124 with open('../livedoornews/random_train.tsv','r') as f:
125     lines = f.read().split('\n')
126     for line in lines:
127         if (line == ""):
128             continue
129         a = line.split()
130         cls = a[0]
131         text = a[1]
132         ids0 = tknz.encode(text)
133         ids = []
134         for tid in ids0:
135             if tid in henkan:
136                 ids.append(henkan[tid])
137             else:
138                 ids.append(1)
139         xdata.append(ids)
```

```
140         ydata.append(int(cls))
141
142 # データ読み込み valid
143 with open('../livedoornews/random_valid.tsv','r') as f:
144     lines = f.read().split('\n')
145     for line in lines:
146         if (line == ""):
147             continue
148         a = line.split()
149         cls = a[0]
150         text = a[1]
151         ids0 = tknz.encode(text)
152         ids = []
153         for tid in ids0:
154             if tid in henkan:
155                 ids.append(henkan[tid])
156             else:
157                 ids.append(1)
158         xval.append(ids)
159         yval.append(int(cls))
160
161
162 batch_size = 8
163 dataset = MyDataset(xdata,ydata)
164 dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True,
        collate_fn=my_collate_fn)
165
166 dataset_valid = MyDataset(xval, yval)
167 dataloader_valid = DataLoader(dataset_valid, batch_size=batch_size,
        shuffle=True, collate_fn=my_collate_fn)
168
169 # Define model
170
171 class DocCls(nn.Module):
172     def __init__(self,bert):
173         super(DocCls, self).__init__()
174         self.bert = bert
175         self.cls=nn.Linear(128,9)
176 # self.cls=nn.Linear(768,9)
177     def forward(self,x1,x2):
178         bout = self.bert(x1,x2)
```

```
# bs = len(bout[0])
        bs = len(bout)
# h0 = [ bout[0][i][0] for i in range(bs)]
        h0 = [ bout[i][0] for i in range(bs)]
        h0 = torch.stack(h0,dim=0)
        h1 = self.cls(h0)
        return h1

class DocCls2(nn.Module):
    def __init__(self,bert):
        super(DocCls2, self).__init__()
        self.bert = bert
        self.cls=nn.Linear(128,9)
# self.cls=nn.Linear(768,9)
    def forward(self,x1,x2):
        bout = self.bert(x1,x2)
# bs = len(bout[0])
        bs = len(bout)
# h0 = [ bout[0][i][0] for i in range(bs)]
        h0 = [ bout[i][0] for i in range(bs)]
        h0 = torch.stack(h0,dim=0)
        h1 = self.cls(h0)
        return h1

# model generate, optimizer and criterion setting

model = DocCls(bert)
model.to(device)

model2 = DocCls2(bert)
model2.to(device)

# optimizer = optim.SGD(model.parameters(),lr=0.001)

optimizer = optim.Adam([
    {'params': model.bert.parameters(), 'lr': 0.0005},
    {'params': model.cls.parameters(), 'lr': 0.001}
])

criterion = nn.CrossEntropyLoss()

```

```
220
221 #earlystopping = EarlyStopping(patience=5, verbose=True, path='./best-
        model.model')
222 earlystopping = EarlyStopping(patience=5, verbose=True, path='./ep7/
        best-model.model')
223
224
225 # Learn
226
227 epoch_num = 15
228
229 #model.train()
230 for ep in range(epoch_num):
231     model.train()
232     i = 0
233     for xs, ys in dataloader:
234         xs1, xmsk = [], []
235         for k in range(len(xs)):
236             tid = xs[k]
237             if (len(tid) > 512):
238                 tid = tid[:512]
239             xs1.append(torch.LongTensor(tid))
240             xmsk.append(torch.LongTensor([1] * len(tid)))
241         xs1 = pad_sequence(xs1, batch_first=True).to(device)
242         xmsk = pad_sequence(xmsk, batch_first=True).to(device)
243         outputs = model(xs1,xmsk)
244         ys = torch.LongTensor(ys).to(device)
245         loss = criterion(outputs, ys)
246         print(i, loss.item())
247         optimizer.zero_grad()
248         loss.backward()
249         optimizer.step()
250         i += 1
251     outfile = "./ep7/model-" + str(ep) + ".model"
252     torch.save(model.state_dict(),outfile)
253     print(outfile,"␣saved")
254
255     model2.load_state_dict(torch.load(outfile))
256
257     model2.eval()
258     #i2 = 0
```

```
    accurate = 0
    num_data = 0
    with torch.no_grad():
        for xxs, yys in dataloader_valid:
            num_data += len(yys)
            xxs1, xxmsk = [], []
            for k in range(len(xxs)):
                tid = xxs[k]
                if (len(tid) > 512):
                    tid = tid[:512]
                xxs1.append(torch.LongTensor(tid))
                xxmsk.append(torch.LongTensor([1] * len(tid)))
            xxs1 = pad_sequence(xxs1, batch_first=True).to(device)
            xxmsk = pad_sequence(xxmsk, batch_first=True).to(device)
            ans = model(xxs1,xxmsk)
            ans = ans.to('cpu')
            ans1 = torch.argmax(ans,dim=1)
            yys1 = torch.LongTensor(yys)
            accurate += (yys1 == ans1).sum().float().item()
            #i2 += 1

    accuracy = accurate / num_data
    print(accuracy)

    earlystopping(accuracy, model) #メソッド呼び出し call
    if earlystopping.early_stop: #ストップフラグがの場合、でループを抜ける Truebreakfor
        print("Early␣Stopping!")
        break
```