令和 元年度茨城大学工学部情報工学科卒業研究論文

BERT を利用した文書の特徴ベクトルの作成

所属 情報工学科

著者 田中裕隆 (16T4032N)

指導教員 新納浩幸教授

令和 2 年 2 月 5 日 (水)

令和 元年度茨城大学工学部情報工学科卒業研究

BERT を利用した文書の特徴ベクトルの作成

著者

田中裕隆 (16T4032N)

指導教員

新納浩幸教授

論文要旨

近年，事前学習モデルを利用することで，自然言語処理システムの性能が大きく向上している．そのような事前学習モデルの中でも，BERT は特に大きな性能の向上を示している．BERT のモデルは，Transformer の Multi-head Attention を多層に積み重ねた，ディープニューラルネットワークに類するモデルである．Masked Language Model と Next Sentence Prediction の二つの手法を用いて，大規模なコーパスで事前学習を行う．この学習済みのモデルを用いることで，文脈に応じた単語の埋め込み表現列を得ることができる．BERT の事前学習は高コストだが，公開されている日本語の事前学習モデルを用いることで，日本語の文書に対しても高い精度で処理できる．

感情分析のような文書分類のタスクの場合，文書を BERT に入力し，その出力から文書の特徴ベクトルを構築する方法によって処理できる．しかし，BERT に入力できるシーケンスの長さには上限がある．この制限によって，上限より長い文書は，その上限を超える部分が切り捨てられてしまう．そのため，長い文書を扱う場合，標準的な手法では文書分類に必要な情報を十分に得られないと考えられる．

そこで，長い文書に対して BERT への入力をスライドさせて，文書中の全ての単語に対応する埋め込み表現を得て，それらの埋め込み表現を全て足し合わせたベクトルを文書の特徴ベクトルとする手法を提案する．

日本語の感情分析のデータに対して提案手法の実験を行い，長文に対して精度が向上することを示した．また，提案手法は feature based なアプローチによるものだが，fine tuning によるアプローチも試みて比較する．さらに BERT 以外の手法として，word2vec による分散表現列，XLNet による手法を試し，提案手法との比較を行った．

Bachelor Thesis in Scholastic 2019,
Department of Computer and Information Sciences, Ibaraki University

Construction of document feature vectors using BERT

Auther
Hirotaka Tanaka (16T4032N)

Adviser
Prof. Hiroyuki Shinnou

Abstract

In recent years, the performance of various natural language processing systems has been greatly improved by pre-training models. Bidirectional Encoder Representations from Transformers (BERT) is a model similar to a deep neural network in which Transformer's Multi-head Attention is stacked in multiple layers. The BERT is pre-trained in a large corpus by two methods, Masked Language Model and Next Sentence Prediction. By the pre-training model, we obtain word embeddings depending on the context. Although BERT pre-training is expensive, we can process Japanese documents with high accuracy by Japanese pre-training model, which is publicly available.

In document-classification tasks such as emotion anarysis task, document feature vector can be constructed from the BERT output. However, the length of the BERT input sequence is upper-limited. Due to this limitation, documents longer than the upper limit are truncated. In the case of handling long documents, we considered that we don't obtain enough information by standard methods for document classification.

Against this background, we propose a method that constructs document feature vector in BERT by embedding all words in long documents.

The effectiveness of the proposed method is demonstrated on Japanese emotion analysis data. In addition to the proposed method, which is feature-based approach, we also conduct standerd fine-tuning approach and compare them. As methods other than BERT, we conduct the distributed representation by word2vec and the XLNet method, and compare them with the proposed method.

目次

1 **序論** 6

2 **関連研究** 7
- 2.1 BERT を利用した文書分類 7
- 2.2 BERT と TF-IDF を併用した文書の特徴ベクトル 8

3 **事前学習モデル** BERT 10
- 3.1 BERT 10
- 3.2 XLNet 14
- 3.3 ALBERT 17

4 **文書分類タスク** 19

5 **提案手法** 20

6 **実験** 22
- 6.1 日本語 BERT 事前学習モデル 22
- 6.2 実験データ 23
- 6.3 実験結果 24

7 **考察** 26
- 7.1 分散表現列との比較 26
- 7.2 Fine-tuning 27
- 7.3 XLNet 28

8 **結論** 29

参考文献 31

付録 33
- A BERT を feature based 利用で実行するソースコード 33

表目次

1 Amazon レビュー文書のデータセットの内訳 23
2 Amazon レビュー文書のテストデータでトークン列が 128 を超える文書数 23
3 実験結果（各手法の正解率） 25
4 長文に対する結果（正解率） 25
5 分散表現列を用いた手法との比較（正解率） 26
6 Fine Tuning との比較（正解率） 27
7 XLNet との比較（正解率） 28

図目次

1 BERT と TF-IDF による特徴ベクトル 8
2 最大長を超える単語列部分の切り捨て 9
3 BERT に 2 文入力する場合のトークン列 11
4 BERT の固定入力長への対応 12
5 BERT のモデル図 13
6 XLNet に 2 文入力する場合のトークン列 14
7 XLNet の固定入力長への対応 15
8 BERT による特徴ベクトル作成の提案手法 21
9 実験結果のグラフ 24

1 序論

近年，自然言語処理の多くのタスクで，事前学習モデルを利用する有効性が示されている [1] [2]．事前学習モデルは様々なものが提案されているが，その中でも BERT [3] が特に優れた性能を示している．

BERT (Bidirectional Encoder Representations from Transformers) は Transformer [4] で用いられた Multi-head attention を 12 層（あるいは 24 層）重ねたモデルであり，パラメータの学習は Masked Language Model と Next Sentence Prediction という 2 つのタスクを解くことで，教師なしの枠組みの下で行われる．学習できたモデルを利用すると，入力文あるいは入力文対に対して，その単語埋め込み表現列を得ることができる．

ただし，BERT はその入力となるシーケンスの最大長が固定である．したがって，BERT の事前学習時の最大長より長い文書を入力とすると，最大長を超える部分の単語に対して埋め込み表現を得ることはできない．そのために，通常の手法で BERT を利用する場合，実際のタスクを解くために必要な情報を文書全体から得ることができない．BERT の事前学習時に十分大きな最大長を設定することで，長い文書に対応することもできるが，入力長に上限が存在することに変わりはなく，またその事前学習のコストも高い．

そこで，任意の長さの入力シーケンスから BERT を利用した埋め込み表現列を得て，そこから文書の特徴ベクトルを作成する手法を提案する．

実験では Webis-CLS-10[*1]の日本語の感情分析のデータを利用して，提案手法の有効性を示した．考察では，提案手法による BERT からの特徴ベクトルと，word2vec による単語の分散表現から得られる特徴ベクトルとの比較を行う．また，XLNet [5] は BERT の問題点に対処し，任意の入力長から単語の埋め込み表現を得ることができるモデルである．この XLNet の手法と BERT による提案手法との比較も行う．

[*1] `https://webis.de/data/webis-cls-10.html`

2　関連研究

2.1　BERT を利用した文書分類

Adhikari らの DocBERT [6] では，BERT を利用した文書分類に関して詳しく分析している．DocBERT の手法では，fine-tuning と蒸留の 2 つの手法を組み合わせている．

蒸留とは，パラメータ数の多い複雑な大きいモデルから，より単純な小さいモデルを学習させ，モデルの軽量化と精度の向上を図る手法である．学習済みのモデルの出力を教師データにとり，これをソフトターゲットとして損失を計算して，新たな小さいモデルの学習を行う．さらに，通常のラベル付けされた教師データを用いて，これをハードターゲットとして損失を求めて，ソフトターゲットと併用して学習する手法も存在する．

DocBERT では，まず BERT で提案された fine-tuning の標準的な手法によって，特殊トークンの `[CLS]` に対する埋め込み表現を新たに追加した分類器に入力して，モデル全体を学習させる．これに加えて，fine-tuning された BERT のモデルから，より小さい LSTM のモデルへ蒸留を行う．目的関数 (損失関数) は，通常の教師データであるラベルによるハードターゲットを用いた $\mathcal{L}_{classification}$ と，学習済みの BERT によるソフトターゲットを用いた $\mathcal{L}_{distill}$ を組み合わせて，重み係数 λ を用いて式 (1) のように表す．

$$\mathcal{L} = \mathcal{L}_{classification} + \lambda \cdot \mathcal{L}_{distill} \tag{1}$$

Adhikari らの研究では BERT の最大長に関する分析も行っている．Reuters と IMDB の 2 つのデータセットに対して，最大長を 128, 256, 512 とする 3 つの設定で比較されている．この実験によれば，BERT の base, large の両方のモデルで，2 つのデータセットのいずれでも最大長を長く設定するほど精度が高くなることが示されている．

ただし，DocBERT の手法では，最大長より長い文書に関して，BERT の標準的な手法と同様に最大長以降を切り捨てている．

2.2　BERT と TF-IDF を併用した文書の特徴ベクトル

我々は既に「BERT による単語埋め込み表現列を用いた文書分類」において，文書分類を解く場合の BERT による特徴ベクトルを扱う手法について提案した [7].

そこでは，BERT による単語埋め込み表現列から求めた平均ベクトルと，TF-IDF によって求めた特徴ベクトルを求め，これらを単位ベクトルに正規化し連結したベクトルを文書の特徴ベクトルとする（図 1 参照）.

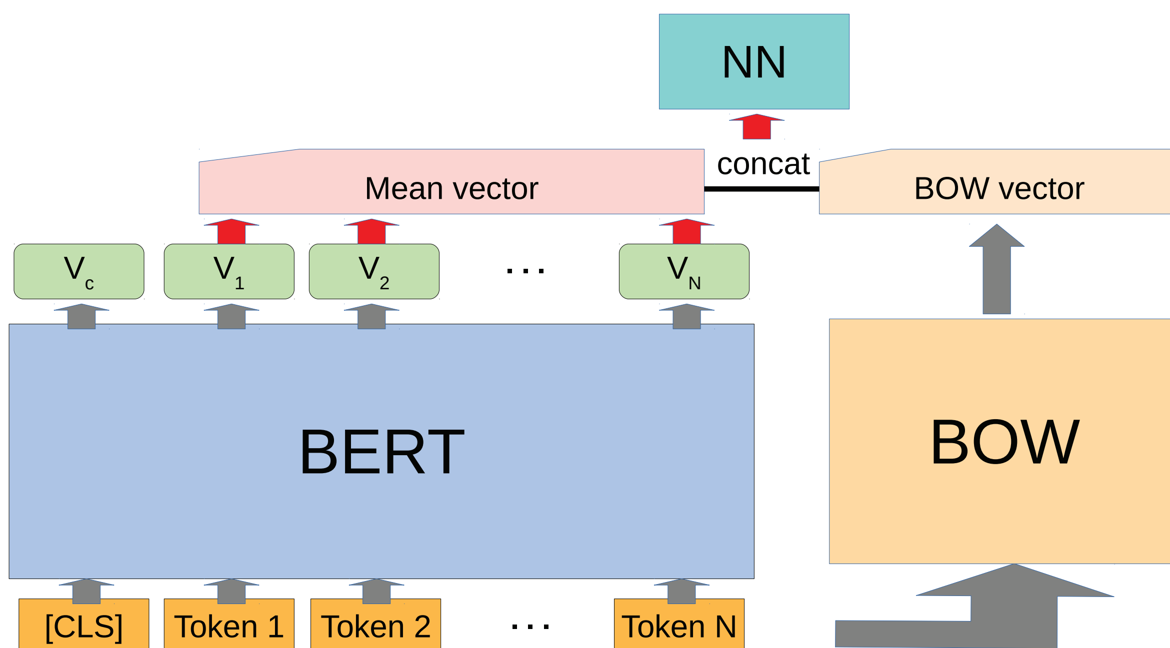


図 1: BERT と TF-IDF による特徴ベクトル

ただし上記の手法は BERT に入力する単語列の最大長の制限から，最大長を超える長い文書に対しては，最大長以降を切り捨てた文書にしてから BERT への入力を行っている（図 2 参照）. 上記の手法は BERT 単独の手法と比較して，大きな改善を果たしているが，それは切り捨てられた単語列の情報を BOW のモデルに取り込めているからと考えられる.

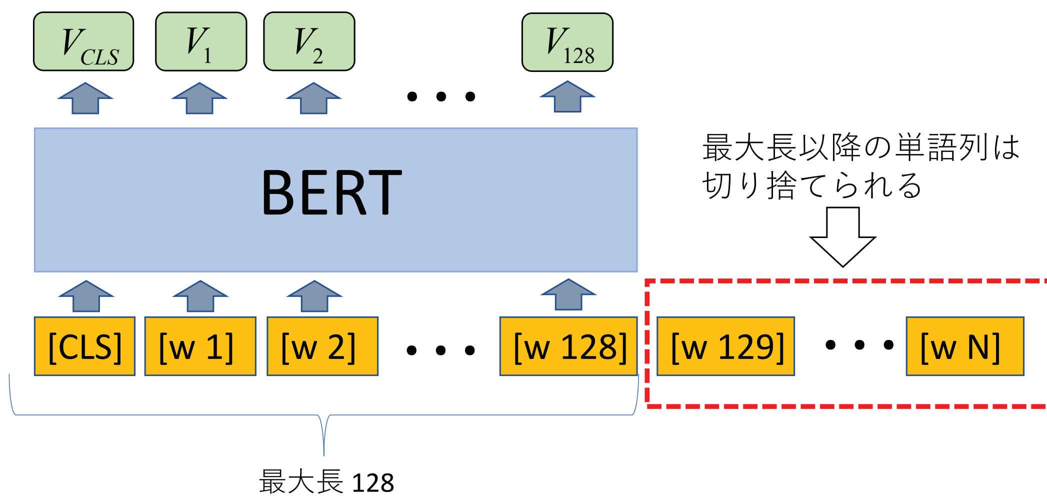


図 2: 最大長を超える単語列部分の切り捨て

3 事前学習モデル BERT

3.1 BERT

3.1.1 言語モデル

言語モデルとは，言語を確率モデルで表したものである．ある文章の単語列 $\mathrm{x} = (x_1, x_2, x_3, \cdots, x_T)$ があるとき，その出現確率は次のように表せる．

$$P(x_1, x_2, x_3, \cdots, x_T) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)\cdots P(x_n|x_1, x_2, \cdots, x_{n-1}) \tag{2}$$

$$logP(\mathrm{x}) = \sum_{t=1}^{T} logP(x_t|\mathrm{x}_{<t}) \tag{3}$$

LSTM に代表されるような RNN では，時系列データに対して事後確率を求めることができるので，言語モデルに対して用いられる．ただし，事後確率を求めるモデルであるために，ある単語の予測には，それより前に出現した単語列しか考慮されない．

Bi-LSTM ではこの問題に対処し，単体では単方向である LSTM を双方向に 2 つ用いることで，ある単語の予測に前後両方の単語列の情報を用いて精度を向上させた．しかし，LSTM であることには変わりないため，前後の情報は別々に扱われる．

BERT では，上記のような単語ごとの事後確率を求める確率モデルは用いない．代わりに，元の文章 $\bar{\mathrm{x}}$ に対してノイズを入れた文章 $\hat{\mathrm{x}}$ を作り，ノイズのある文章 $\hat{\mathrm{x}}$ から元の文章 $\bar{\mathrm{x}}$ の確率を式 (4) のようにモデル化する．

$$logP(\bar{\mathrm{x}}|\hat{\mathrm{x}}) \approx \sum_{t=1}^{T} m_t logP(x_t|\hat{\mathrm{x}}) \tag{4}$$

このようにモデル化することで，BERT ではある単語の予測に前後 (双方向) の単語列を一度に考慮することができる．

3.1.2 学習

BERT の事前学習は，2 種類の教師なし学習によって行われる．

1 つは Masked Language Model である．これは，文の中のいくつかの単語を別の単語に置き換えて，そこにあるべき単語を予測する問題である．一般的な手法では，入力

トークンの内 15% に対して，以下のように置き換える．

- 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．

2 つ目は Next Sentence Prediction である．これは，2 つの入力文に対して連続しているか否かを予測する問題である．この問題を解く場合，入力トークン列は図 3 のようになる．

図 3: BERT に 2 文入力する場合のトークン列

このとき，特殊トークン [CLS] に対する埋め込み表現を用いて，連続しているかを予測する．

3.1.3 入力ベクトル

BERT の入力は基本的に 3 種類ある．一つは Token Embeddings であり，これは単語それぞれに対応した埋め込み表現である．BERT ではこの埋め込み表現も学習する．2 つ目は Segment Embeddings であり，これは 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．最後に 3 つ目は Position Embeddings である．BERT の用いる 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，入力単語ごとに位置に関する情報を付与するため，Position Embeddings を用いる．

BERT では，モデルの入力長は固定される．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 4 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．

3.1.4 モデル

BERT は，Attention を基本とする構成が特徴のモデルである．

Attention モデルは，Query, Key, Value の 3 組を入力とする．それぞれ Q, K, V とし

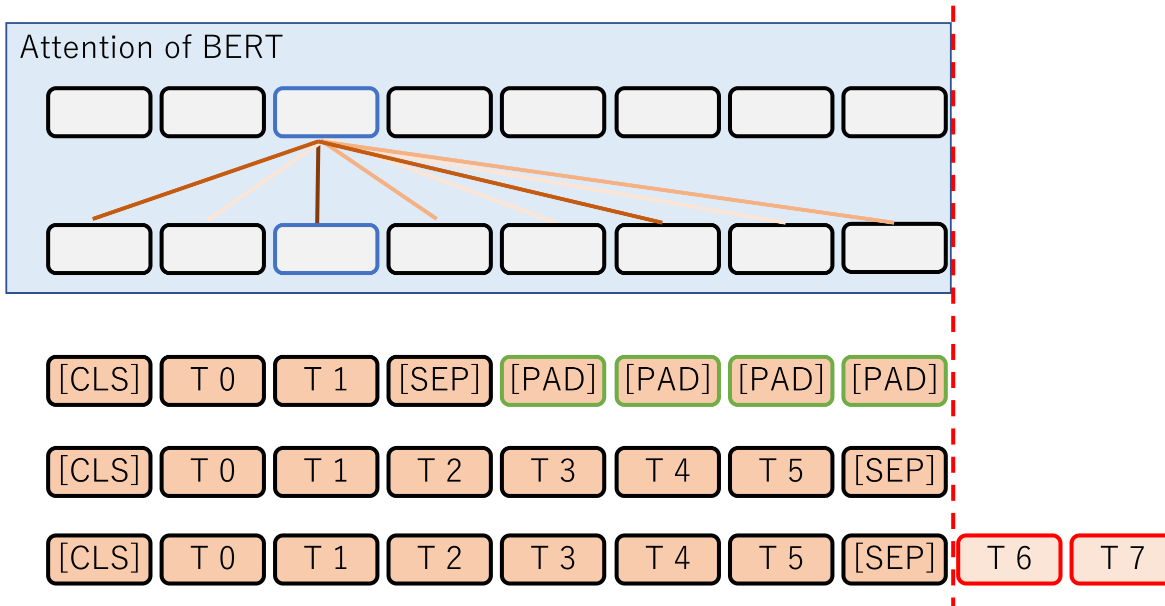


図 4: BERT の固定入力長への対応

て表される．Q と K の内積を取り，これと V をかけることで，Q と K の類似度から適切な V を取り出す操作が行われる．翻訳モデルでは，例えば英語から日本語に翻訳する場合に対応する単語の語順が文法的に異なっても，適切な単語に注意を当てて翻訳できるように用いる．

Self-Attention では，Q, K, V の 3 組に対して同じ入力を与える．ある単語を予測する際に，その周辺単語に適切に注意を当てて予測する．BERT では，Self-Attention として Attention モデルを用いる．

BERT の基本のパーツは Multi-head attention である．Multi-head attention は n 単語埋め込み表現列を入力として，各埋め込み表現をより適切なものに変換して出力する．つまり出力は変換された n 単語埋め込み表現列である．

Multi-head attention の概略を述べる．基本は self attention なので Q, K, V の 3 組が入力である．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，以下の式[*2]により self attention を行う．

$$softmax\left(\frac{Q'K'^T}{\sqrt{d_k}}\right)V' \tag{5}$$

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，これらを横に連結することで，$n \times m$ 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．

BERT はこの Multi-head attention を 12 層（あるいは 24 層）重ねたモデルである (図 5 参照)．結局，BERT は n 単語埋め込み表現列を入力とし，それをより文脈に合った n 単語埋め込み表現列に変換していると捉えることができる．

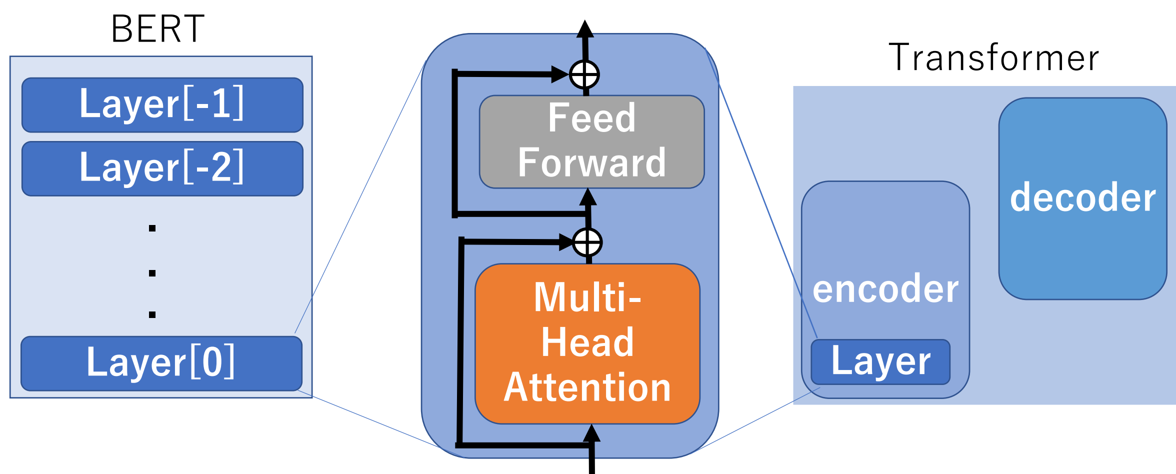


図 5: BERT のモデル図

[*2] Scaled Dot-Product Attention

3.2 XLNet

3.2.1 言語モデルと学習手法

XLNet は，BERT の問題点を改良した Yang らによるモデルである．その問題点の一つは，BERT の事前学習手法の一つの Masked Language Model である．Masked Language Model による学習では，入力単語列の一部を特殊トークンである [MASK] に置き換える．この [MASK] トークンは，事前学習時にのみ使用され，文書分類などの実際の問題を解く時には使用されない．そのため，事前学習でのみ使用される [MASK] はノイズになり，実際の問題の精度を落としていると Yang らは主張している．

XLNet では，新たに Permutation Language Model を提案し，この問題に対処している．Permutation LM は，Masked LM より純粋な自己回帰モデルベースの言語モデルである．ある入力単語列に対して，その単語列がとりうる全ての順列で予測する．Positional Encoding による元の入力単語列の位置情報と，Mask をかけることによって，入力単語列そのものの順番を入れ替えるコストをかけずに計算する．Masked LM では，単語列の一部に [MASK] のノイズを入れる方式のため，一度に学習できる量は限られていたが，Permutation LM ではその点も対処している．

XLNet の事前学習時は，全ての単語を予測することはせずに，順列オーダーの後方の単語のみを予測する．Large のモデルでは，入力文長に対して 1/6 のみ予測する．

また，XLNet では事前学習手法に BERT で用いられた Next Sentence Prediction を用いない．ただし，入力として複数文とるタスクでは，BERT 同様に文を連結した入力ができる．XLNet は事後確率を求めていく言語モデルを用いているため，BERT では先頭にあった特殊トークン [CLS] は，XLNet では最後に置く (図 6 参照).

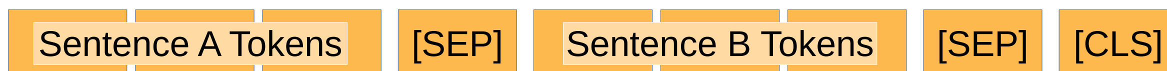


図 6: XLNet に 2 文入力する場合のトークン列

3.2.2 任意入力長への対応

XLNet のもう一つの特徴は任意の入力長に対応していることである．これは，Transformer の問題点を改良した Transformer-XL [8] の Segment recurrence mechanism を

用いているからである．通常の Transfomer は，長い文章を扱う際，数百の固定長列 (セグメント) に切り分けて，各セグメントを別々に処理する．Transformer-XL で提案されている Segment recurrence mechanism は，前のセグメントで計算された特徴表現を，その次のセグメントを処理する際に再利用する．ネットワークの層の深さを N とすると，単語間の依存関係の長さは最大で N 倍になる．

XLNet のモデルは任意の入力長に対応するが，実際の学習ではバッチ処理を行うために，入力長を揃えて同じサイズのテンソルに格納する (図 7 参照)．このとき設定する入力長より単語列が短い場合，単語列の前を特殊トークンである [PAD] で埋める．入力長より単語列が長い場合，入力長より長い単語列の後ろの部分を切り捨てる．

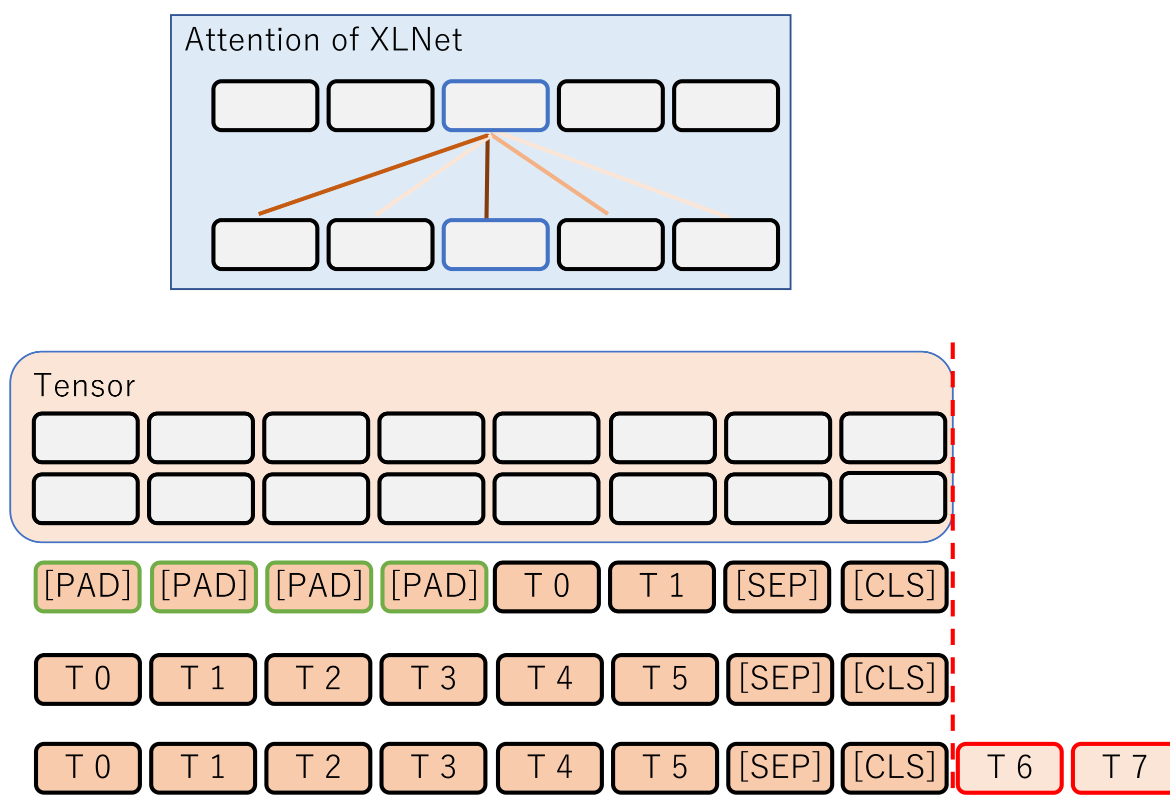


図 7: XLNet の固定入力長への対応

3.2.3 モデル

XLNet のモデルは，基本的には BERT と同様の Self-Attention を用いている．ただし，Permutation LM では単語列をランダムにするために，単純に同様のモデルを用いるだけでは次の単語の効率的な学習ができない．そこで，XLNet では Two stream

attention と呼ばれる, Content stream attention と Query stream attention の 2 種類の機構を用いる．今，単語列 $\mathrm{x} = [x_1, x_2, \cdots, x_T]$ があったとする．位置の埋め込み表現を $\mathrm{z} = [z_1, z_2, \cdots, z_T]$ とする．Content stream attention が扱う Content representation を $h_\theta(\mathrm{x}_{z_{\leq t}})$ または単に h_{z_t} と表す．同様に，Query stream attention が扱う Query representation を $g_\theta(\mathrm{x}_{z_{<t}}, z_t)$ または単に g_{z_t} と表す．ここで θ はパラメータである．初めに，$g_i^{(0)} = w$，$h_i^{(0)} = e(x_i)$ とそれぞれ初期化する．w は学習可能な重みパラメータであり，$e(x_i)$ は単語 x_i に対応する埋め込み表現である．それぞれの Self-Attention 層を $m = 1, 2, \cdots, M$ のように表すとき，Two stream attention は式 (6)，(7) のように表せる．Query stream attention では，ある位置 t に関して位置埋め込み表現 z_t を用いて予測するが，その場所の単語表現 x_{z_t} を見ない．Query stream attention は，予測する単語自身の情報にマスクして Attention を求めている．Content stream attention では，ある位置 t に関して位置埋め込み表現 z_t と，単語表現 x_{z_t} を両方用いて予測する．Content stream attention は，実質純粋な Self-Attention である．

$$g_{z_t}^{(m)} \leftarrow Attention(Q = g_{z_t}^{(m-1)}, KV = h_{\mathrm{z}_{<t}}^{(m-1)}; \theta) \tag{6}$$

$$h_{z_t}^{(m)} \leftarrow Attention(Q = h_{z_t}^{(m-1)}, KV = h_{\mathrm{z}_{\leq t}}^{(m-1)}; \theta) \tag{7}$$

3.3 ALBERT

3.3.1 モデル

ALBERT [9] は，Lan らによる BERT を軽量化したモデルである．BERT の手法のまま単に埋め込み次元数を増やしても，性能の向上に寄与しない．そこで Lan らは，BERT のモデルを軽量化した上で埋め込み次元数を増やすことで，様々な NLP タスクで性能が向上することを示した．

軽量化のための手法の一つは Factorized embedding parameterization である．BERT では，初めに入力単語列を対応する埋め込み表現に置き換える．扱う語彙数を V，ベクトルの次元数を H とすると，全語彙の埋め込み表現のパラメータ量は $V \times H$ と表せる．扱うことのできる語彙数は大きく，本研究で用いる日本語事前学習モデルでは 3 万を超える．したがって，この埋め込み表現に関するパラメータ量は大きくなる．そこで Factorized embedding parameterization では，埋め込み表現をボトルネック構造にすることで全体のパラメータ量を小さくする．初めの埋め込み表現のベクトル次元数を小さい次元数 E に落とす．その後，線形変換器で次元数 H に変換する．このとき，パラメータ量は $V \times E + E \times H$ と表せる．$E < H$ の条件で，元の $V \times H$ より軽量化される．

軽量化のためのもう一つの手法は Cross-layer parameter sharing である．BERT のモデルは，Multi-head Attention と線形変換のセットを Transformer Layer の 1 層として数えて，これを 12 層，あるいは 24 層重ねてできている．ALBERT のパラメータシェアリングは，この Transformer Layer で全ての層が同じパラメータを持つようにシェアする手法である．これによってモデルのパラメータ量が大きく削減される．

3.3.2 学習

ALBERT では，BERT の Next Sentence Prediction の問題点に対処している．Next Sentence Prediction は，2 つの文が連続しているか否かを学習する．Next Sentence Prediction の問題点は Negative Sampling にある．正例に関しては，データ中の連続する 2 文として作成できる．ただし，負例に関して，Next Sentence Predictioin ではコーパス中のランダムな 2 文の組み合わせとしている．この手法では，次文予測より単純な他の要因によって問題を解けてしまい，次文予測の学習に寄与しにくい．

この問題点に対して，ALBERT では Sentence Order Prediction という手法を用いて

いる．この手法では，Negative Sampling において正例の文の順序を逆にした 2 文を用いる．これによって，次文予測をより効率的に学習する．

4 文書分類タスク

文書分類は分類問題の一種であり，一般に教師あり学習を用いることで解決できる．そのため従来より数多くの研究がある．またディープラーニングを利用する場合でも，CNN [10] や RNN [11] を利用するなど多くの研究がある．

一方，文書分類を含め自然言語処理の多くのタスクにおいて，事前学習モデルを利用する有効性が示されている．事前学習モデルを利用する場合，大きく 2 つの利用法がある．一つは fine tuning である．これは事前学習モデルが出力する情報を，タスクを解決するためのネットワークの入力とし，その事前学習モデルを含めたネットワーク全体を学習の対象とするものである．この場合，事前学習モデルの部分は既に大量のデータから学習できた形となっているため，比較的少量のデータを用いるだけで，連結したネットワークを学習できる．OpenAI GPT [2] はニューラルネット翻訳である Transformer [4] の decoder 部分を利用した言語モデル[*3]あり，このような fine tuning の利用を念頭においている．ULMFiT [12] においても事前学習モデルを言語モデルに設定し，目的のタスクに対して fine tuning を行う．

事前学習モデルのもう一つの利用法は feature based のものである．これは事前学習モデルが出力する情報を，目的のタスクを解くための素性として利用するものである．word2vec のような単語分散表現も事前学習モデルと捉えることができる．単語の分散表現をタスク解決のための素性とした研究には文書分類を含め多くの研究がある．fastText は Subword [13] に対する分散表現を構築し，高速かつ高精度な文書分類が行えることを実験で示している [14]．ELMo [1] は文脈を考慮した単語の分散表現を導くモデルである．実体は 2 層の双方向 LSTM であり，大規模コーパスを利用して言語モデルを学習する．これが事前学習モデルとなり，feature based の形で利用できる．

本論文で利用する BERT は従来の事前学習モデルを改善しており，様々なタスクで従来の事前学習モデルの性能を上回っている．このため本論文で扱う文書分類であっても，その効果が期待できる．

[*3] 言語モデルも一種の事前学習モデルである．

5 提案手法

標準的な手法では，BERT の学習時に設定した最大長を超えた入力シーケンスからは，最大長を超えた部分の単語列に対してその埋め込み表現を得ることはできない．ここでは，BERT の最大長を超えるシーケンスに対して，全ての単語に対する埋め込み表現を得る手法について示す（図 8 参照）．

提案手法は，以下の手順で任意の長さの文書に対して，その特徴ベクトルを得る．

1. 対象となる文書の先頭から BERT の最大長分を入力として BERT の出力を得る．
2. 文書の全てが BERT の入力列に入りきらなかった場合，文書の先頭から BERT の最大長の半分だけずらした位置から最大長分を入力として BERT の出力を得る．
3. 文書の全てから BERT の出力を得られるまで手順 (2) を繰り返す．
4. 得られた BERT の出力である単語埋め込み表現のベクトル全てを足し合わせる．

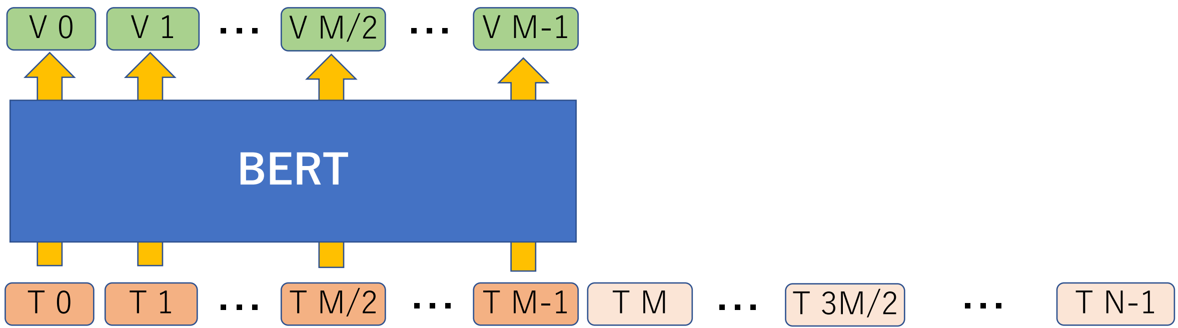
このようにして得られたベクトルを BERT による文書の特徴ベクトルとする．

数式で表すと次のようになる．入力となる文書を d として，先頭から i 番目の単語トークンを d_i とする．文書の長さを N，BERT の最大長を L とする．入力の d_i に対応する BERT の出力を v_i^j とする．BERT による特徴ベクトル b_l は，次のように表せる．

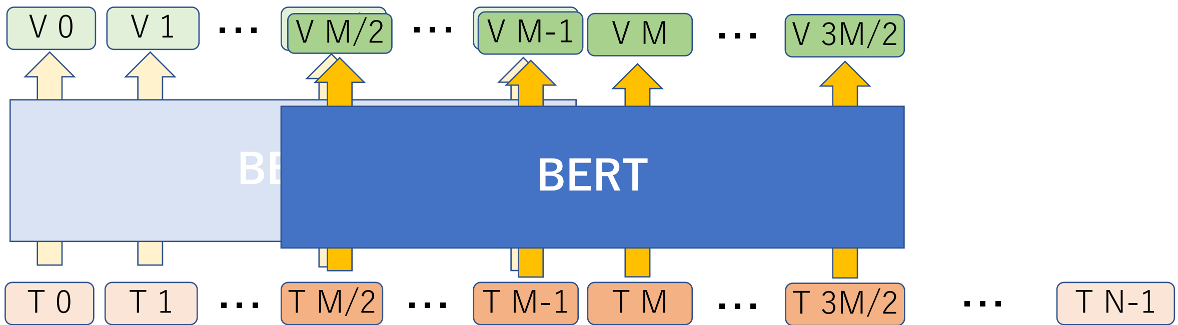
$$\begin{aligned}
b_l = {} & v_0^0 + v_1^0 + \cdots + v_{L/2-1}^0 \\
& + v_{L/2}^0 + v_{L/2}^1 + v_{L/2+1}^0 + v_{L/2+1}^1 + \cdots \\
& + v_{L-1}^0 + v_{L-1}^1 + v_L^1 + v_L^2 + \cdots \\
& + v_i^j + \cdots + v_N^{1+(N-1)/L}
\end{aligned}$$

実験では，TF-IDF によるベクトル t と連結した $[b_l; t]$ を文書 d の特徴ベクトルとして文書分類を行う．

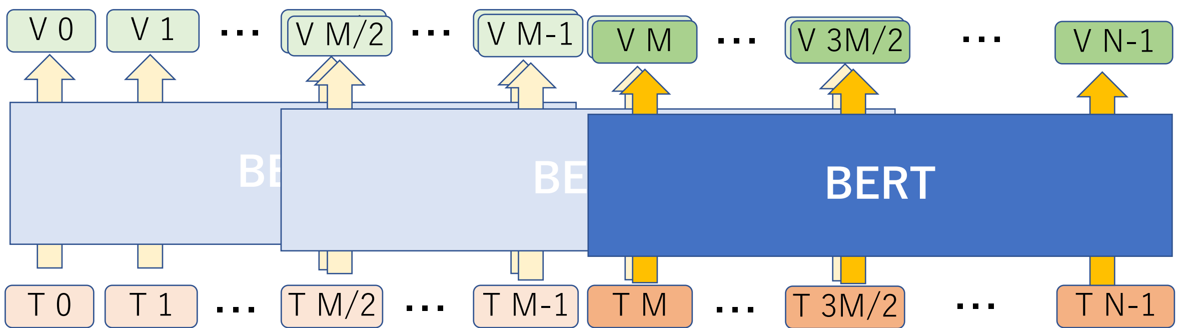
文書分類のための分類器には 3 層ニューラルネットワークを用いた．具体的には各層は全結合であり，順に 400 次元，50 次元，2 次元ベクトルへと線形変換される．活性化関数にはシグモイド関数を用いており，出力層に対しては softmax 関数を用いている．交差エントロピー誤差を損失関数として損失を求め，Adam によって最適化した．



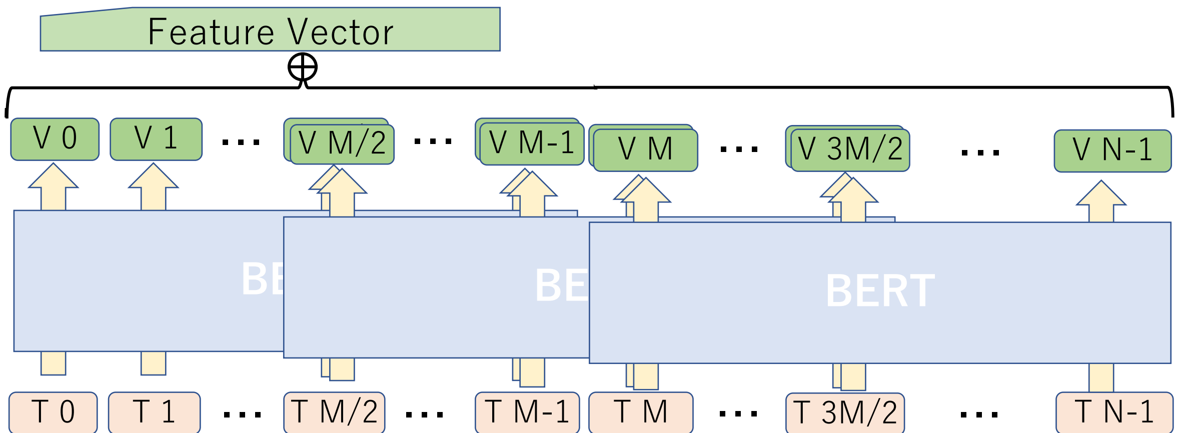
(a) 先頭から BERT の最大長分を入力する．



(b) 最大長の半分だけスライドして BERT へ入力する．



(c) 最後のトークンが入力されるまで繰り返す．



(d) BERT の出力全てを足し合わせる．

図 8: BERT による特徴ベクトル作成の提案手法

6 実験

6.1 日本語 BERT 事前学習モデル

公開されている BERT の多言語モデル[*4] には日本語も含まれており，日本語のタスクに対して多言語の事前学習モデルを利用することも可能である．しかし，これを利用すると基本単位が文字になってしまい，適切ではないと考えられる．例えば,「この本はすばらしい.」という文は，本研究で用いる実験データに実際に存在する文である．この文を多言語モデルが扱う場合,「この」「本」「[UNK]」「.」のように分割される．[UNK] とは，BERT の特殊トークンの一つで，未知語に対して与えられる．本研究で行う感情分析のタスクでは,「すばらしい」という単語は重要だと考えられるが，多言語モデルはこの表現を捉えることができない.

そこで本研究では，日本語に対応した事前学習モデルとして，京都大学黒橋・河原研究室が以下のサイトで公開している日本語事前学習モデルを使用する.

http://nlp.ist.i.kyoto-u.ac.jp/index.php?BERT%E6%97%A5%E6%9C%AC%E8%AA%9EPretrained%E3%83%A2%E3%83%87%E3%83%AB

このモデルは，訓練コーパスを日本語 Wikipedia(約 1800 万文) とし，モデルの構造は BASE と同じ 12 層，語彙数は 32000，最大長は 128 としている.

この事前学習モデルの入力となるテキストは，初めに，同じく京都大学黒橋・河原研究室が公開している Juman++[*5] によって形態素解析され，形態素単位に分割される．その後，BPE (Byte Pair Encoding) によって subword に分割される．例として「この本はすばらしい.」の文では,「この」「本」「は」「すば」「##らしい」「.」のように扱われる.

[*4] https://storage.googleapis.com/bert_models/2018_11_23/multi_cased_L-12_H-768_A-12.zip

[*5] http://nlp.ist.i.kyoto-u.ac.jp/index.php?JUMAN++

6.2 実験データ

実験で使用したデータセットは，以下のサイトで公開されている Amazon のレビュー文書である．評価の 4 と 5 を positive，1 と 2 を negative とした感情分析データとして用いる．

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

データセットの内訳を表 1 に示す．このデータセットは books，DVD，music の 3 つの領域がある．領域毎に訓練データとして 2,000 文書，テストデータとして 2,000 文書が存在する．

表 1: 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
テストデータ	2,000	2,000	2,000	6,000

この実験で使用する BERT 事前学習モデルの最大長は 128 である．この実験のテストデータには，トークン列が 128 を超える文書が各領域にそれぞれ 905 文書, 872 文書 及び 727 文書存在する（表 2）．つまりデータ全体に対して約 40 %程度の文書が，標準的な手法では扱いきれない長さの文書である．

表 2: Amazon レビュー文書のテストデータでトークン列が 128 を超える文書数

books	DVD	music	3 領域の合計
905	872	727	2,504

TF-IDF で扱う特徴語は，全訓練データ 6,000 文書に出現する 41,400 語とする．

6.3 実験結果

実験結果を表 3 と図 9 に示す．この結果は，3 領域それぞれ 10 回ずつ試行した平均値である．

b_s は，BERT の最大長を超える情報は切り捨てて，求めた BERT の単語埋め込み表現列の平均ベクトルを文書の特徴ベクトルとする手法による結果である．t は，TF-IDF によるベクトルを文書の特徴ベクトルとする手法による結果である．$[b_s; t]$ は，b_s と t を連結して得られた特徴ベクトルを用いる手法の結果である．b_l は，提案手法のように BERT から求めたベクトルを文書の特徴ベクトルとする手法である．b_s より良い結果となっている．$[b_l; t]$ は，b_l と t を連結して得られた特徴ベクトルを用いる提案手法の場合の結果である．この提案手法は，実験の中で最も良い結果となった．

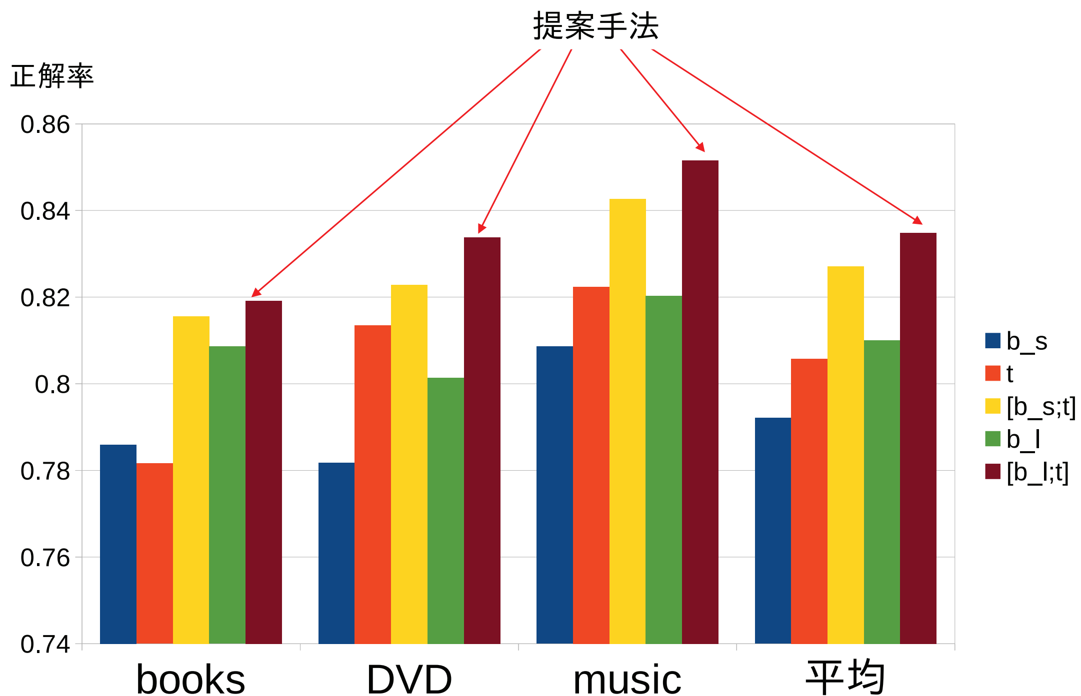


図 9: 実験結果のグラフ

また，提案手法 $[b_l; t]$ が最大長 128 を超える文書 (以下長文とする) に対して切り捨てる手法 $[b_s; t]$ と比較して有効となっていることを確かめるために，長文のみのテストデー

表 3: 実験結果（各手法の正解率）

	books	DVD	music	3 領域の平均
b_s	0.7859	0.7818	0.8086	0.7921
t	0.7816	0.8135	0.8224	0.8058
$[b_s; t]$	0.8156	0.8229	0.8427	0.8271
b_l	0.8086	0.8014	0.8203	0.8101
$[b_l; t]$	**0.8192**	**0.8338**	**0.8516**	**0.8349**

タで評価を行った結果を表 4 に示す．表 4 より，提案手法が長文に対して有効であることがわかる．

表 4: 長文に対する結果（正解率）

	books	DVD	music	3 領域の平均
$[b_s; t]$	0.8079	0.8360	0.8355	0.8265
$[b_l; t]$	**0.8202**	**0.8464**	**0.8488**	**0.8385**

7 考察

7.1 分散表現列との比較

実験では BERT による単語埋め込み表現列を用いた．ここでは，word2vec による分散表現列を用いて同様に実験を行い，その結果を比較する．ここで用いる word2vec は，以下のサイトで公開されている学習済みのモデルである．このモデルは，本論文で用いた BERT の日本語事前学習モデルと同様に，日本語 Wikipedia データから学習している．

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

入力文書に対して形態素解析を行い，得られた単語列から word2vec により分散表現列を求めた．次にそれら分散表現の平均ベクトル w を求め正規化し，TF-IDF によるベクトル t と連結して $[w; t]$ を作成した．この $[w; t]$ を入力文書の特徴ベクトルとし，先の実験を行った．結果を表 5 に示す．

表 5: 分散表現列を用いた手法との比較（正解率）

	books	DVD	music	3 領域の平均
$[w; t]$	0.7899	0.8207	0.8330	0.8146
$[b_s; t]$	0.8156	0.8229	0.8427	0.8271
$[b_l; t]$	**0.8192**	**0.8338**	**0.8516**	**0.8349**

この実験から，分散表現列を用いた場合と比較して，BERT を用いた手法はより良い結果となった．分散表現列と BERT による単語埋め込み表現列は，同じような情報を表現しているが，BERT の方が分散表現よりも有用であることが言える．

7.2 Fine-tuning

BERT は，fine-tuning を行うことによって高いパフォーマンスを出すことのできるモデルである．ここでは，BERT における fine-tuning の標準的な手法で実験を行い，結果を比較する．fine-tuning の標準的な手法とは，BERT で入力シーケンスに付与する特殊トークンである [CLS] の埋め込み表現を文書の特徴ベクトルとして学習する手法である．

ここでは BERT のソースと一緒に公開されている `run_classifier.py` [*6]を使うことで，実験データに対して fine tuning を行った．学習率は 2e-5，エポック数は 3 に設定した．その結果を表 6 に示す．f_{bert} が fine tuning の結果である．f_{bert} は $b_{[CLS]}$ よりも改善されてはいるが b_l に及ばない．当然，提案手法である $[b_l; t]$ よりも正解率ははるかに低い．

表 6: Fine Tuning との比較（正解率）

	books	DVD	music	3 領域の平均
$b_{[CLS]}$	0.7629	0.7567	0.7779	0.7658
b_l	0.8086	0.8014	0.8203	0.8101
t	0.7816	0.8135	0.8224	0.8058
$[b_l; t]$	**0.8192**	**0.8338**	**0.8516**	**0.8349**
f_{bert}	0.7894	0.7799	0.8019	0.7904

文書分類では単純に [CLS] の埋め込み表現を用いた BERT の fine tuning よりも，提案手法のように BERT を feature based で利用する方が有効であることが確認できた．ただし本論文の提案手法のように文書の特徴ベクトルを構築し，そこから fine tuning することも可能だと思われる．今後の課題としたい．

[*6] `https://github.com/google-research/bert`

7.3 XLNet

XLNet の特徴の一つは任意の入力長に対応していることである．本論文では，BERT が任意の入力長を扱えないことを問題点としてあげているが，XLNet も同様にこの問題に対処している．そこで，BERT による提案手法と XLNet による手法とを比較する実験を行う．

以下で公開モデルされている日本語 XLNet のモデル (NFKC 版) を利用した実験を行った．このモデルは tokenizer に MeCab+NEologd 及び Sentencepiece を利用している．MeCab を利用することで形態素として正しく分割し，その上で Sentencepiece を行うことで，語彙のカバー範囲を広げている．

`https://qiita.com/mkt3/items/4d0ae36f3f212aee8002`

結果を表 7 に示す．表中の f_{xlnet} が XLNet を用いて fine tuning を行った結果である．BERT を用いた fine tuning の結果である f_{bert} よりもわずかに正解率が低く，XLNet を用いた効果はなかった．

表 7: XLNet との比較（正解率）

	books	DVD	music	3 領域の平均
$[b_l; t]$	**0.8192**	**0.8338**	**0.8516**	**0.8349**
f_{bert}	0.7894	0.7799	0.8019	0.7904
f_{xlnet}	0.7750	0.7750	0.8000	0.7833

XLNet の fine tuning の利用法では，本タスクに対しての効果はなかったが，feature base の利用法では良い結果を得ることができる可能性がある．XLNet の feature base の利用法に関しては今後の課題である．

8 結論

本論文では BERT の最大長が固定である制約に対して，最大長より長い文書に対しても BERT の情報を有効に用いる手法を提案した．入力シーケンスに対して最大長の半分ずつスライドして複数回 BERT の出力を求めることによって，最大長を超える文書に対しても全ての単語の埋め込み表現を取得し，それらを足し合わせて文書の特徴ベクトルを作成した．

京都大学の黒橋・河原研究室が公開している日本語の BERT 事前学習モデルを用いて，Webis-CLS-10 の日本語の Amazon レビュー文書を利用した感情分析の実験により，提案手法が長文に対して有効であることを示した．さらに，TF-IDF と併用する手法が，この提案手法に対しても有効であることも示した．また，fine-tuning の標準的な手法と比較し，feature based なアプローチである提案手法の有効性も示した．BERT とは別の手法として，word2vec による分散表現列と，XLNet による手法を試し比較した．特に XLNet は任意の入力長に対応しており，高い精度になると考えていたが，実験結果は BERT の標準的な fine-tuning による手法と大差なく，いずれも提案手法が有効であることが確認できた．

今後の課題は，一般的に fine tuning することで精度向上が望めるため，提案手法を fine tuning への拡張を行うことや，標準的な手法の fine tuning であまり高い精度にならなかった XLNet では，feature base の利用を試みることとしたい．

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，示唆を頂いた古宮研究室，新納研究室の皆様に感謝します．

参考文献

[1] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *NAACL-2018*, pp. 2227–2237, 2018.

[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] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv preprint arXiv:1906.08237*, 2019.

[6] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. DocBERT:BERT for Document Classification. *arXiv preprint arXiv:1904.08398*, 2019.

[7] Hiotaka Tanaka, Hiroyuki Shinnou, Rui Cao, Jing Bai, and Wen Ma. Document classification by word embeddings of bert. In *PACLING-2019*, pp. XXX–XXX, 2019.

[8] Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attention language models beyond a fixed-length context. *arXiv preprint arXiv:1901.028660*, 2019.

[9] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Sorlcut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *arXiv preprint arXiv:1909.11942*, 2019.

[10] Yoon Kim. Convolutional Neural Networks for Sentence Classification. In

EMNLP-2014, pp. 1746–1751, 2014.

[11] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *AAAI-2015*, pp. 2267–2273, 2015.

[12] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *ACL-2018*, pp. 328–339, 2018.

[13] 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.

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

付録

A　BERT を feature based 利用で実行するソースコード

BERT や XLNet, ALBERT などは, 公開されている transformers[*7]のソースコードを利用することで実行できる．このソースコードは Python で書かれており，ディープラーニングのフレームワークである Pytorch を用いて実行される．ただし，transformers には feature based な利用ができるソースコードが存在しない．そこで，transformers の前のバージョンである pytorch-pretrained-BERT に存在した `extract_features.py` と，transformers の `examples/run_glue.py` を参考にして，transformers のソースコードに対応した feature based 利用可能なソースコード `extract_features.py` を作成した．このソースコードを A.1 に示す．

また，ソースコード A.1 を実行するための Bash スクリプト `bert.sh` のソースコードを A.2 に示す．

ソースコード A.1: extract_features.py

```
1 """Extract pre-computed feature vectors from a PyTorch BERT model."""
2
3 from __future__ import absolute_import
4 from __future__ import division
5 from __future__ import print_function
6
7 import argparse
8 import collections
9 import logging
10 import json
11 import re
12 import random
13 import numpy as np
14 import pathlib
15 import os
16 import copy
17 import sys
18 import torch
```

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

```
19 from tqdm import tqdm
20 from torch.utils.data import TensorDataset, DataLoader,
       SequentialSampler
21 from torch.utils.data.distributed import DistributedSampler
22
23 current_dir = pathlib.Path(__file__).resolve().parent
24 sys.path.append(str(current_dir) + '/../')
25
26 from transformers import BertJapaneseTokenizer, BertModel, BertConfig
27
28
29 logging.basicConfig(format = '%(asctime)s␣-␣%(levelname)s␣-␣%(name)s␣-␣
       ␣␣%(message)s',
30                     datefmt = '%m/%d/%Y␣%H:%M:%S',
31                     level = logging.INFO)
32 logger = logging.getLogger(__name__)
33
34 class InputExample(object):
35
36     def __init__(self, unique_id, text_a, text_b):
37         self.unique_id = unique_id
38         self.text_a = text_a
39         self.text_b = text_b
40
41     def __repr__(self):
42         return str(self.to_json_string())
43
44     def to_dict(self):
45         """Serializes this instance to a Python dictionary."""
46         output = copy.deepcopy(self.__dict__)
47         return output
48
49     def to_json_string(self):
50         """Serializes this instance to a JSON string."""
51         return json.dumps(self.to_dict(), indent=2, sort_keys=True) +
               "\n"
52
53 class InputFeatures(object):
54     """A single set of features of data."""
55
56     def __init__(self, unique_id, tokens, input_ids, attention_mask,
```

```
        token_type_ids):
57          self.unique_id = unique_id
58          self.tokens = tokens
59          self.input_ids = input_ids
60          self.attention_mask = attention_mask
61          self.token_type_ids = token_type_ids
62
63      def __repr__(self):
64          return str(self.to_json_string())
65
66      def to_dict(self):
67          """Serializes this instance to a Python dictionary."""
68          output = copy.deepcopy(self.__dict__)
69          return output
70
71      def to_json_string(self):
72          """Serializes this instance to a JSON string."""
73          return json.dumps(self.to_dict(), indent=2, sort_keys=True) +
              "\n"
74
75  def set_seed(args):
76      random.seed(args.seed)
77      np.random.seed(args.seed)
78      torch.manual_seed(args.seed)
79      if args.n_gpu > 0:
80          torch.cuda.manual_seed_all(args.seed)
81
82  def convert_examples_to_features(examples, tokenizer,
83                                        max_length=512,
84                                        pad_on_left=False,
85                                        pad_token=0,
86                                        pad_token_segment_id=0,
87                                        mask_padding_with_zero=True):
88      features = []
89      for (ex_index, example) in enumerate(examples):
90          if ex_index % 10000 == 0:
91              logger.info("Writing␣example␣%d" % (ex_index))
92
93          inputs = tokenizer.encode_plus(
94              example.text_a,
95              example.text_b,
```

```
96              add_special_tokens=True,
97              max_length=max_length,
98          )
99          input_ids, token_type_ids = inputs["input_ids"], inputs["
                token_type_ids"]
100         tokens = []
101         for input_id in input_ids:
102             tokens.append(tokenizer._convert_id_to_token(input_id))
103
104         # The mask has 1 for real tokens and 0 for padding tokens. Only
                 real
105         # tokens are attended to.
106         attention_mask = [1 if mask_padding_with_zero else 0] * len(
                input_ids)
107
108         # Zero-pad up to the sequence length.
109         padding_length = max_length - len(input_ids)
110         if pad_on_left:
111             input_ids = ([pad_token] * padding_length) + input_ids
112             attention_mask = ([0 if mask_padding_with_zero else 1] *
                    padding_length) + attention_mask
113             token_type_ids = ([pad_token_segment_id] * padding_length)
                     + token_type_ids
114         else:
115             input_ids = input_ids + ([pad_token] * padding_length)
116             attention_mask = attention_mask + ([0 if
                    mask_padding_with_zero else 1] * padding_length)
117             token_type_ids = token_type_ids + ([pad_token_segment_id]
                    * padding_length)
118
119         assert len(input_ids) == max_length, "Error␣with␣input␣length␣
                {}␣vs␣{}".format(len(input_ids), max_length)
120         assert len(attention_mask) == max_length, "Error␣with␣input␣
                length␣{}␣vs␣{}".format(len(attention_mask),
121


122         assert len(token_type_ids) == max_length, "Error␣with␣input␣
                length␣{}␣vs␣{}".format(len(token_type_ids),
123
```

```
124
125         if ex_index < 5:
126             logger.info("***␣Example␣***")
127             logger.info("unique_id:␣%s" % (example.unique_id))
128             logger.info("tokens:␣%s" % "␣".join([str(x) for x in
                    tokens]))
129             logger.info("input_ids:␣%s" % "␣".join([str(x) for x in
                    input_ids]))
130             logger.info("attention_mask:␣%s" % "␣".join([str(x) for x
                    in attention_mask]))
131             logger.info("token_type_ids:␣%s" % "␣".join([str(x) for x
                    in token_type_ids]))
132
133         features.append(
134                 InputFeatures(unique_id=example.unique_id,
135                               tokens=tokens,
136                               input_ids=input_ids,
137                               attention_mask=attention_mask,
138                               token_type_ids=token_type_ids,))
139
140     return features
141
142 def to_list(tensor):
143     return tensor.detach().cpu().tolist()
144
145 def read_examples(input_file):
146     """Read a list of `InputExample`s from an input file."""
147     examples = []
148     unique_id = 0
149     with open(input_file, "r", encoding='utf-8') as reader:
150         while True:
151             line = reader.readline()
152             if not line:
153                 break
154             line = line.strip()
155             text_a = None
156             text_b = None
157             m = re.match(r"^(.*)␣\|\|\|␣(.*)$", line)
158             if m is None:
159                 text_a = line
```

```
160             else:
161                 text_a = m.group(1)
162                 text_b = m.group(2)
163             examples.append(
164                 InputExample(unique_id=unique_id, text_a=text_a, text_b
                        =text_b))
165             unique_id += 1
166     return examples
167 
168 def load_and_cache_examples(args, tokenizer, evaluate=False,
        output_examples=False):
169     if args.local_rank not in [-1, 0] and not evaluate:
170         torch.distributed.barrier() # Make sure only the first process
                 in distributed training process the dataset, and the others
                  will use the cache
171 
172     # Load data features from cache or dataset file
173     input_file = args.input_file
174     cached_features_file = os.path.join(os.path.dirname(input_file), '
            cached_{}_{}_{}'.format(
175         'feature',
176         list(filter(None, args.model_name_or_path.split('/'))).pop(),
177         str(args.max_seq_length)))
178     # if os.path.exists(cached_features_file) and not args.
            overwrite_cache and not output_examples:
179     if os.path.exists(cached_features_file) and not output_examples:
180         logger.info("Loading␣features␣from␣cached␣file␣%s",
                cached_features_file)
181         features = torch.load(cached_features_file)
182     else:
183         logger.info("Creating␣features␣from␣dataset␣file␣at␣%s",
                input_file)
184         examples = read_examples(input_file=input_file)
185         features = convert_examples_to_features(examples=examples,
186                                                 tokenizer=tokenizer,
187                                                 max_length=args.
                                                        max_seq_length,
188                                                 pad_on_left=bool(args.
                                                        model_type in ['
                                                        xlnet']),
189                                                 pad_token=tokenizer.
```

```
                                                    convert_tokens_to_ids
                                                    ([tokenizer.
                                                    pad_token])[0],
190                                              pad_token_segment_id=4
                                                    if args.model_type
                                                    in ['xlnet'] else
                                                    0,)
191         if args.local_rank in [-1, 0]:
192             logger.info("Saving␣features␣into␣cached␣file␣%s",
                    cached_features_file)
193             torch.save(features, cached_features_file)
194
195     if args.local_rank == 0 and not evaluate:
196         torch.distributed.barrier() # Make sure only the first process
                 in distributed training process the dataset, and the others
                  will use the cache
197
198     # Convert to Tensors and build dataset
199     all_input_ids = torch.tensor([f.input_ids for f in features],
            dtype=torch.long)
200     all_attention_mask = torch.tensor([f.attention_mask for f in
            features], dtype=torch.long)
201     all_token_type_ids = torch.tensor([f.token_type_ids for f in
            features], dtype=torch.long)
202     all_example_index = torch.arange(all_input_ids.size(0), dtype=
            torch.long)
203
204     dataset = TensorDataset(all_input_ids, all_attention_mask,
            all_token_type_ids, all_example_index)
205     if output_examples:
206         return dataset, examples, features
207     return dataset
208
209 def main():
210     parser = argparse.ArgumentParser()
211
212     ## Required parameters
213     parser.add_argument("--input_file", default=None, type=str,
            required=True)
214     parser.add_argument("--output_file", default=None, type=str,
            required=True)
```

```
215     parser.add_argument("--model_name_or_path", default=None, type=str
            , required=True,
216                         help="Bert␣pre-trained␣model␣selected␣in␣the␣
                                list:␣bert-base-uncased,␣"
217                              "bert-large-uncased,␣bert-base-cased,␣bert
                                     -base-multilingual,␣bert-base-chinese.
                                     ")
218
219     ## Other parameters
220     parser.add_argument("--do_lower_case", action='store_true', help="
            Set␣this␣flag␣if␣you␣are␣using␣an␣uncased␣model.")
221     parser.add_argument("--layers", default="-1,-2,-3,-4", type=str)
222     parser.add_argument("--max_seq_length", default=128, type=int,
223                         help="The␣maximum␣total␣input␣sequence␣length␣
                                after␣WordPiece␣tokenization.␣Sequences␣
                                longer␣"
224                         "than␣this␣will␣be␣truncated,␣and␣sequences␣
                                shorter␣than␣this␣will␣be␣padded.")
225     parser.add_argument("--batch_size", default=32, type=int, help="
            Batch␣size␣for␣predictions.")
226     parser.add_argument("--local_rank",
227                         type=int,
228                         default=-1,
229                         help = "local_rank␣for␣distributed␣training␣on␣
                                gpus")
230     parser.add_argument("--no_cuda",
231                         action='store_true',
232                         help="Whether␣not␣to␣use␣CUDA␣when␣available")
233     parser.add_argument('--seed', type=int, default=42,
234                         help="random␣seed␣for␣initialization")
235
236     args = parser.parse_args()
237
238     args.config_name = ""
239     args.tokenizer_name = ""
240     args.model_type = 'bert'
241
242     if args.local_rank == -1 or args.no_cuda:
243         device = torch.device("cuda" if torch.cuda.is_available() and
                not args.no_cuda else "cpu")
244         args.n_gpu = torch.cuda.device_count()
```

```
245     else:
246         torch.cuda.set_device(args.local_rank)
247         device = torch.device("cuda", args.local_rank)
248         torch.distributed.init_process_group(backend='nccl')
249         args.n_gpu = 1
250 
251     logger.info("device:␣{}␣n_gpu:␣{}␣distributed␣training:␣{}".format(
            device, args.n_gpu, bool(args.local_rank != -1)))
252 
253     set_seed(args)
254 
255     # Load pretrained model and tokenizer
256     if args.local_rank not in [-1, 0]:
257         torch.distributed.barrier() # Make sure only the first process
                in distributed training will download model & vocab
258 
259     args.model_type = args.model_type.lower()
260     config_class, model_class, tokenizer_class = (BertConfig,
            BertModel, BertJapaneseTokenizer)
261     config = config_class.from_pretrained(args.config_name if args.
            config_name else args.model_name_or_path,
262                                           num_labels=2000,
                                                  finetuning_task="japanese
                                                  ")
263     tokenizer = tokenizer_class.from_pretrained(args.tokenizer_name if
            args.tokenizer_name else args.model_name_or_path,
264                                                 do_lower_case=args.
                                                        do_lower_case)
265     model = model_class.from_pretrained(args.model_name_or_path,
266                                         from_tf=bool('.ckpt' in args.
                                                model_name_or_path),
267                                         config=config)
268 
269     if args.local_rank == 0:
270         torch.distributed.barrier() # Make sure only the first process
                in distributed training will download model & vocab
271 
272     model.to(device)
273 
274     dataset, examples, features = load_and_cache_examples(args,
            tokenizer, evaluate=True, output_examples=True)
```

```
275
276     args.eval_batch_size = args.batch_size
277     # Note that DistributedSampler samples randomly
278     eval_sampler = SequentialSampler(dataset) if args.local_rank == -1
             else DistributedSampler(dataset)
279     eval_dataloader = DataLoader(dataset, sampler=eval_sampler,
            batch_size=args.eval_batch_size)
280
281
282     layer_indexes = [int(x) for x in args.layers.split(",")]
283
284     model.eval()
285     with open(args.output_file, "w", encoding='utf-8') as writer:
286         for batch in tqdm(eval_dataloader, desc='Evaluating'):
287             batch = tuple(t.to(device) for t in batch)
288             with torch.no_grad():
289                 inputs = {'input_ids':batch[0],
290                           'attention_mask':batch[1]}
291                 if args.model_type != 'distilbert':
292                     inputs['token_type_ids'] = batch[2] if args.
                            model_type in ['bert','xlnet'] else None # XLM
                            , DistilBERT and RoBERTa don't use segment_ids
293                 example_indices = batch[3]
294                 outputs = model(**inputs)
295                 all_hidden_states = outputs[2]
296                 #all_hidden_states = all_hidden_states.cpu().numpy()
297
298             for b, example_index in enumerate(example_indices):
299                 feature = features[example_index.item()]
300                 unique_id = int(feature.unique_id)
301                 output_json = collections.OrderedDict()
302                 output_json["linex_index"] = unique_id
303                 all_out_features = []
304                 for (i, token) in enumerate(feature.tokens):
305                     all_layers = []
306                     for (j, layer_index) in enumerate(layer_indexes):
307                         layer_output = all_hidden_states[int(
                                layer_index)].detach().cpu().numpy()
308                         layer_output = layer_output[b]
309                         layers = collections.OrderedDict()
310                         layers["index"] = layer_index
```

```
311                         layers["values"] = [
312                             round(x.item(), 6) for x in layer_output[i
                                    ]
313                         ]
314                         all_layers.append(layers)
315                     out_features = collections.OrderedDict()
316                     out_features["token"] = token
317                     out_features["layers"] = all_layers
318                     all_out_features.append(out_features)
319                 output_json["features"] = all_out_features
320                 writer.write(json.dumps(output_json) + "\n")
321
322
323 if __name__ == "__main__":
324     main()
```

ソースコード A.2: bert.sh

```
1  #!/bin/bash
2
3  export BERT_BASE_DIR=/PATH/TO/BERT/PRETRAINED/MODEL
4
5  input_file_name=$1
6  output_file_path=$2
7  output_extension=".json"
8
9  if [[ ! -f ${input_file_name} ]]; then
10     echo "input␣file␣not␣found"
11     exit 1
12 fi
13
14 list=(${input_file_name//./ })
15 file_name=${list[0]}
16 if [[ -n ${output_file_path} ]]; then
17   file_name=${output_file_path}/$(basename ${file_name})
18   if [[ ! -d ${output_file_path} ]]; then
19     mkdir -p ${output_file_path}
20   fi
21 fi
22 output_file_name=${file_name}${output_extension}
23
```

```
24 python3 /PATH/transformers/examples/extract_features.py \
25         --input_file ${input_file_name} \
26         --output_file ${output_file_name} \
27         --model_name_or_path ${BERT_BASE_DIR} \
28         --layers -1 \
29         --max_seq_length 128 \
30         --batch_size 8
```