令和 2 年度茨城大学工学部情報工学科卒業研究論文

# BERT を用いた文書分類タスクへの Mix-Up 手法の適用

**所属** 情報工学科

**著者** 菊田尚樹 (17T4028N)

**指導教員** 新納浩幸教授

令和 3 年 2 月 5 日 (金)

令和 2 年度茨城大学工学部情報工学科卒業研究論文

# BERT を用いた文書分類タスクへの Mix-Up 手法の適用


## 著者

菊田尚樹 (17T4028N)

## 指導教員

新納浩幸教授



### 論文要旨

自然言語処理のタスクを機械学習のアプローチで解決しようとした場合, 訓練データの構築コストが高いことが問題になる．このため従来より様々な試みがなされている．その中で近年のトピックの一つとしてデータ拡張 (Data Augmentation) [1] がある．データ拡張手法は大きく加工と生成の 2 種類に分けられる．例えば画像の識別であれば，訓練データ内の画像を反転させたり，切り取ったりした画像であってもその画像のラベルに変化はないので，そのようにして加工した画像を訓練データに追加することで訓練データを増やすことができる．あるいは GAN などを利用して人工的なデータを生成する手法もデータ拡張の一種といえる．データ拡張の生成手法の一つである Mix-Up 手法 [2] は，簡単に実装でき効果も高いことが知られている．Mix-Up 手法は画像識別を対象にした手法であるが，自然言語処理にも応用が可能である [3].

本論文では BERT [4] を用いた文書分類タスクに Mix-Up 手法を適用することを試みた．具体的には BERT は 2 文の入力が可能であるため，ラベルの異なる 2 文書の ID 列を連結して入力し，one-hot ベクトルを利用してマルチクラスの出力を教師データとした．livedoor ニュースコーパスを用いた実験で，連結させる文書の 2 通りの選出方法による文書分類について，通常の文書分類との精度比較を行った.

実験の結果，訓練データにおいて不足しているラベルの文書を優先して選出しデータ拡張を行った際に，通常の文書分類に比べ精度が向上することが分かった．これにより，Mix-up に用いる文書の選出方法が結果に大きく影響を与えることが示された．今後は，Mix-Up に用いる文書の組み合わせ方，文書とラベルそれぞれの混合方法について，他の手法の提案・実験を重ね，さらなる理解・進展を図りたい.

# 目次

# 表目次

# 図目次

# 第 1 章

# 序論

自然言語処理のタスクを機械学習のアプローチで解決しようとした場合，訓練データの構築コストが高いことが問題になる．このため従来より様々な試みがなされている．その中で近年のトピックの一つとしてデータ拡張 (Data Augmentation) [1] がある．データ拡張手法は大きく加工と生成の 2 種類に分けられる．例えば画像の識別であれば，訓練データ内の画像を反転させたり，切り取ったりした画像であってもその画像のラベルに変化はないので，そのようにして加工した画像を訓練データに追加することで訓練データを増やすことができる．あるいは GAN などを利用して人工的なデータを生成する手法もデータ拡張の一種といえる．データ拡張の生成手法の一つである Mix-Up 手法 [2] は，簡単に実装でき効果も高いことが知られている．Mix-Up 手法は画像識別を対象にした手法であるが，自然言語処理にも応用が可能である [3].

本論文では BERT [4] を用いた文書分類タスクに Mix-Up 手法を試みる．具体的には BERT は 2 文の入力が可能であるため，ラベルの異なる 2 文書を組み合わせて入力し，マルチクラスの出力を教師データとする．livedoor ニュースコーパスを用いた実験で提案手法の有効性を示した.

# 第 2 章

# 関連研究

## 2.1　埋め込み表現

機械学習にて自然言語を処理する際，テキストをそのままの形で用いることはできず，ベクトルの形に表現し直す必要がある．ベクトル化することを埋め込みといい，埋め込みによってできたベクトルを埋め込み表現と呼ぶ．埋め込み表現を獲得するモデルとして代表的なものに Word2Vec [6] がある．

## 2.2　ニューラルネットワーク

人間の脳の神経細胞ネットワークを元に考えられた数理モデルであり，複数のニューロンから構成される．(図 2.1 参照) 基本的に入力層，中間層，出力層の三層構成で，入力に重み (w) をかけた後，バイアス (b) を加えたものが次の層への出力される．w と b を最適化し，理想的な出力値 y を求めることを目的とする．また，このニューラルネットワークを多層化し，さらに複雑になったものがディープニューラルネットワークであり，それを用いた学習がディープラーニングと呼ばれる．

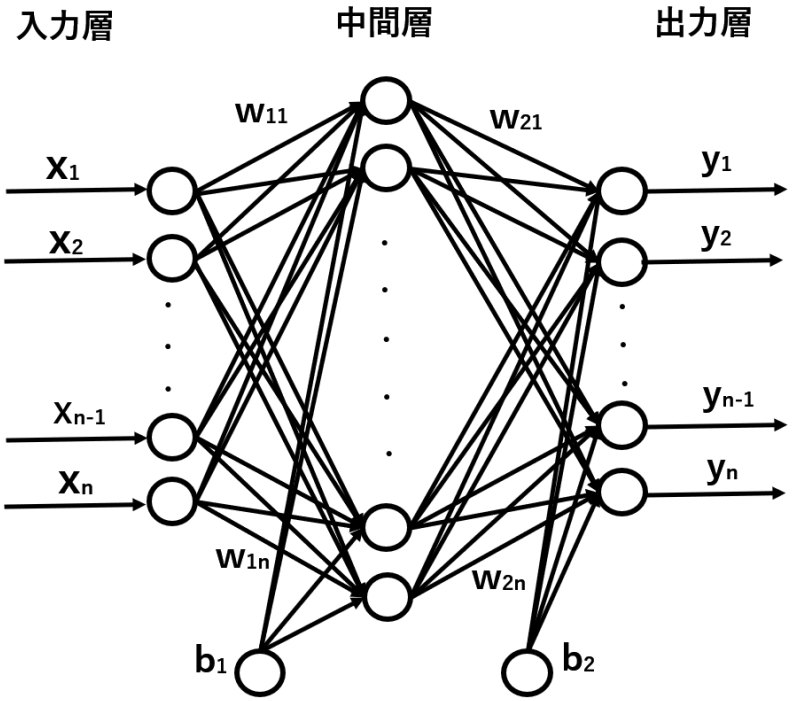


図 2.1: ニューラルネットワークの構造

## 2.3　RNN

文脈を正しく理解する際, 文章中の時系列を正しく捉える必要がある．RNN(Recurrent Neural Network の略) はニューラルネットワークを拡張し，時系列データを扱えるようにしたものである．RNN の構造を図 2.2 に示す．RNN は，中間層がループする構造となっており，中間層が前の時刻の中間層の影響を受けるので，ニューラルネットワークが以前の時刻における情報を保持できる．そのため，過去の情報を用いて判断を下すことが可能になっている．しかし RNN では長期の記憶を保持することに難点がある．

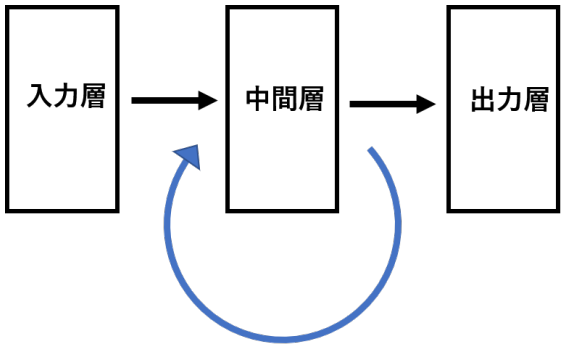


図 2.2: RNN の構造

## 2.4　LSTM

RNN の長期記憶を保持することが難しい点を克服したのが LSTM(Long short-term memory の略) である．LSTM はゲートと呼ばれる仕組みを採用することで，RNN と違い，長期と短期両方の記憶を保持することが可能となっている．LSTM の構造を図 2.3 に示す．LSTM は中間層の代わりに LSTM ブロックを配置し，過去の情報を忘れるか保持するかを判断することで，必要な情報のみを次の時刻に引き継ぐことを可能にしている．

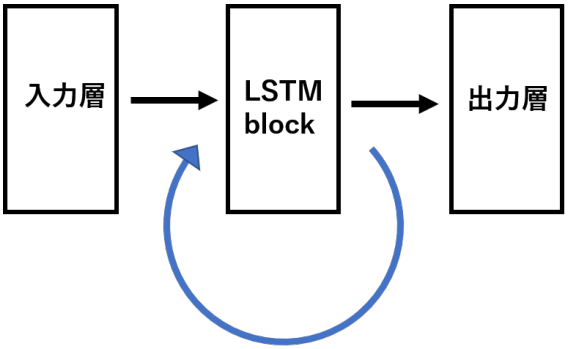


図 2.3: LSTM の構造

## 2.5　Transformer

2017 年に Ashish Vaswani らによって発表された深層学習モデル [5] であり，自然言語処理の多くのタスクに利用できる．Transformer は Attention 機構と呼ばれる仕組みに基づき作られており，それ以前に時系列データの分析に使われていた RNN や LSTM の性能を上回り，代替となるものである．RNN などでも Attention 機構を追加したものが利用されていたが，Transformer では Attention 機構のみを利用して単語間の関係の分析，文脈判断を行うことが可能である．モデルの構造を図 2.4 に示す．

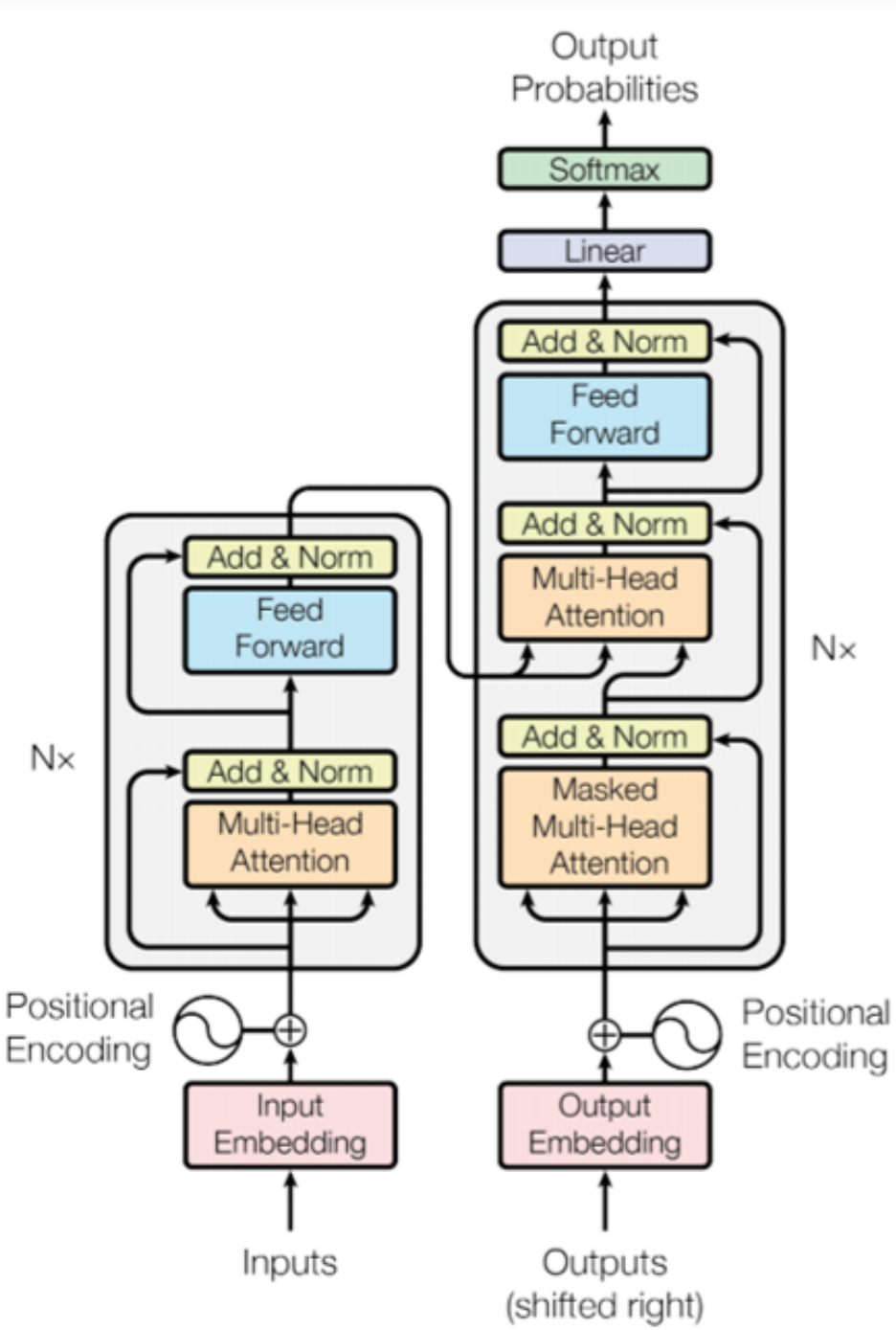


図 2.4: Transformer の構造 (元論文 [5] からの引用)

また，Transformer を利用した事前学習モデルとして BERT がある．

## 2.6　BERT

BERT (Bidirectional Encoder Representations from Transformers の略) は 2018 年に Google が作成して以降利用が注目されている高性能な事前学習済みモデル [4] であり，分類，単語予測，文脈判断などに活用ができる．構造としては，Transformer で用いられた Multi-head attention を 12 層重ねて作られている．従来のモデルでは対象単語の右または左の単語のみから文脈を学習していたのに対し，BERT では対象単語の前後の単語から文脈を判断するため，より正確に文章を処理できるようになっている．また，従来のモデルは文章分類，翻訳，感情分析など特定のタスクごとに 1 つのモデルを必要としていたため，タスクが変わればモデルを一から作り直す必要があったが，BERT は一つの言語モデルを事前学習させ，そこから目的に合わせた転移学習やファインチューニングを行い，様々な処理を行わせることができる．タスクが変わっても事前学習モデルに層を追加するだけでよいため汎用性が高いと言える．BERT の詳細を以下に記す．

### 2.6.1　BERT の学習

BERT は以下の二つの方法にて事前学習を行う．

(1)Masked Language Model(**単語の予測**)

文中のいくつかの単語が別の単語に置換され，本来そこにあるべき単語を予測するというものである．具体的には以下の動作が行われる．

1. 15% の確率で各単語がマスクされる．

   my dog is hairy → my dog is [MASK]

2. マスクトークンが，10% の確率でランダムで別の単語に，

   my dog is apple

   別の 10% の確率で元の単語に，

   my dog is hairy

   残りの 80% の確率でマスクのままにされる．

   my dog is [MASK]

3. マスク部分を前後の文脈から予測する．

(2)Next Sentence Prediction(2 **文の関係性の理解**)

入力として前後の関係性を持った 2 つの文があり，50 %の確率で後ろの文が関係性のない文に書き換わる．文のつながりが正しいなら IsNext，間違っているなら NotNext の判定を出す．具体例を以下に示す．

**入力：**　[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]

**判定：**　IsNext

**入力**:　[CLS] the man went to [MASK] store [SEP] penguin [MASK] are flight ##less birds [SEP]

**判定**:　NotNext

### 2.6.2　BERT の性能評価

モデルの精度を測る指標である以下のベンチマークタスク 11 種で既存のモデルを超え最高精度を達成している．

GLUE　9 種類の言語の理解に関するタスク うち 8 種類で達成

SQuAD　質疑応答タスク 陳述文から質問文の解答を抽出

CoNL　固有表現抽出タスク 単語に人物，組織，位置などをタグ付ける

SWAG　入力文に続く文を 4 つの候補の中から選ぶ

ここまで述べたように，BERT は自然言語処理において利用価値の高いモデルであり，本研究では Mix-Up 手法を利用することで，BERT を用いた文書分類タスクのさらなる精度向上を試みる．

## 2.7　画像分野でのデータ拡張 (Data Augmentation)

### 2.7.1　画像の加工

既存の訓練データ画像に対して，回転，平行移動，拡大縮小，色彩の変更，ノイズ [マスク] の挿入などの加工を施し，新たな訓練データとして追加するといった手法がある．回転を例にとると，元の画像を時計回りに 45 度ずつ回転させたものを各画像に対して作っていけば一周するまでに 7 枚の新たな画像を作成でき，結果的に訓練データ数を 8 倍にできる．この場合，画像の正解ラベルは元のままである．

### 2.7.2　GAN による生成

GAN(Generative Adversarial Network： 敵対的生成ネットワーク) は生成モデルの一種であり，元画像そっくりの偽物の画像の作成を行うことができる．それにより出来上がった偽画像を新たな訓練データとして使うことでデータ拡張となる．GAN では，生成器ネットワークと識別器ネットワークが競い合うようにして学習を行う．前者は一般に Generator(贋作者) と呼ばれ，後者は一般に Discriminator(鑑定者) と呼ばれる．Generator は Discriminator を騙せるように偽画像の完成度を高め，Discriminator は Generator に騙されないように識別能力を高めていくといった具体に学習を進める．GAN の構造を図 2.5 に示す．

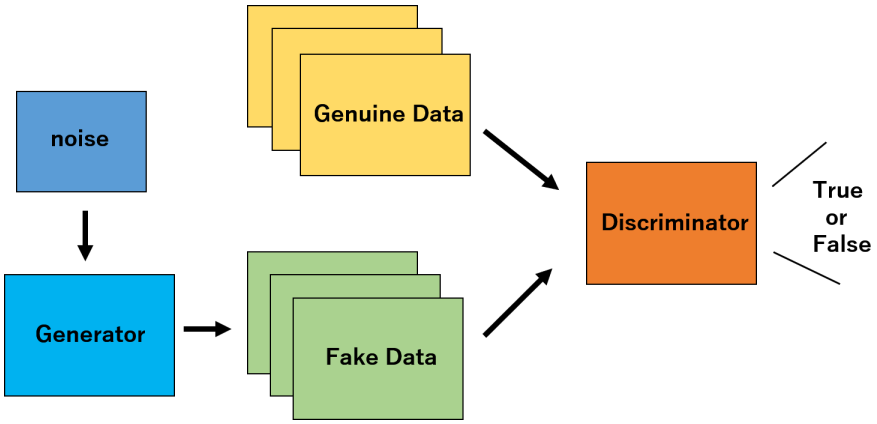


図 2.5: GAN の構造

### 2.7.3　Mix-Up 手法

Mix-Up とは，2017 年に Hongyi Zhang により発表された画像分野でのデータ拡張手法である [2]．画像データに関しては式 2.1，ラベルに関しては式 2.2 によってデータ拡張を行う．

$$x = \lambda x_i + (1 - \lambda)x_j \tag{2.1}$$

$$y = \lambda y_i + (1 - \lambda)y_j \tag{2.2}$$

x は画像データのベクトル，y はラベルの one-hot ベクトルである． λ は混合の割合を表わしている．これまでのデータ拡張と異なり，拡張して得られるデータのラベルが，元のラベル同士の混合ラベルとして新たに作られるのがこの手法の特徴である．単純な動作ながら効果を得られるため注目すべきデータ拡張手法と言え，本実験で利用した．

## 2.8　nlp でのデータ拡張 (Data Augmentation)

### 2.8.1　字句置換

テキスト中の字句をなんらかの指標に基づき，他の字句に置き換えることによって，新たな訓練データを作成する手法がある．指標の例としては，同義語，ベクトル表現の近いもの，省略形などがある．同義語であれば辞書に基づき「永久」を「永遠」に置換するといった具合である．ベクトル表現の近いものへ置換する方法では，主に Word2Vec など，単語の埋め込み表現を獲得するツールを利用する．同義語の時とは違い， cat が lion に置換されるなどといった場合が考えられる．省略形というのは，主に英語で書かれた文章中において She is と She's を置換するといったものである．しかし注意すべき点として，She is ならば She's が成り立つのに対して，She's ならば She is とは限らない．というのは She has の省略形である場合があるからである．よって，このような複数の可能性がある曖昧な省略形を元に戻すのは許可しないといった工夫が必要である．

### 2.8.2　ノイズの追加

機械学習にてモデルを作る際，汎化性能を意識する必要がある．訓練データやテストデータでの評価値が高くても，他のデータでは性能を発揮できないモデルは良いモデル

とは言えない．また，扱うデータに人的不備があるなどの不測の事態に対しても，ある程度耐えられるモデルを作ることは重要である．ノイズや外的な影響を受けないことをロバスト (頑強，堅牢) 性があるといい，モデルがロバストになることを目的としたものが，ノイズの追加によるデータ拡張である．具体的には，単語に対しての意図的にスペルミスを加える方法や，文章の順番をシャッフルするなどの方法がある．スペルミスの追加方法として，アルファベットを qwerty キーボードの配置上近いアルファベットに置換するといったものがある．

### 2.8.3　逆翻訳

テキストを他言語にいったん翻訳してから，再び元の言語に翻訳し直すことを逆翻訳と呼ぶ．機械翻訳を利用し逆翻訳を行い，元のテキストと異なるテキストになった場合に新たな訓練データとして加えるといったデータ拡張手法がある．例えば，Google 翻訳[*1]を利用し，「明日は晴れるといいね」を英語に翻訳すると，”I hope it will be fine tomorrow” になり，再び日本語に翻訳し直すと「明日晴れるといいのですが」になる．この場合，元の文と逆翻訳後の文が異なっているので新たな文を獲得できたことになる．

### 2.8.4　テキストへの Mix-Up

2019 年，Hongyu Guo により，Mix-Up 手法を nlp に用いた研究が行われた [3]．Mix の方法は前節の画像分野での方法と同じく，式 2.1 と式 2.2 によって行っている．nlp の場合は，x が単語の埋め込み表現または文の埋め込み表現である．

(例) 6:4 の割合で Mix-Up する場合

**文書 1 のベクトル**

[0.2, -0.3, 0.5, ⋯]

**文書 2 のベクトル**

[0.4, 0.1, -0.5, ⋯]

**新たな文書のベクトル**

$[0.2, -0.3, 0.5, \cdots] \times 0.6 + [0.4, 0.1, -0.5, \cdots] \times 0.4 = [0.28, 0.22, 0.1, \cdots]$

[*1] https://translate.google.co.jp/?hl=ja&tab=wT

# 第 3 章

# 提案手法

先行研究での Mix-Up 手法を BERT での文書分類にも採用するとなると問題が生じてしまう．先行研究の手法では，NN の学習以前に文書の特徴ベクトルを作っておく必要があるが，BERT を利用した文書分類では NN の学習プロセスで文書の特徴ベクトルを得るため，先行研究とは順序が異なる．仮に先行研究の手法を採用するとなると，BERT による特徴ベクトルの計算部分は学習の外でやることになり，分類精度が期待できない．(図 3.1 参照)

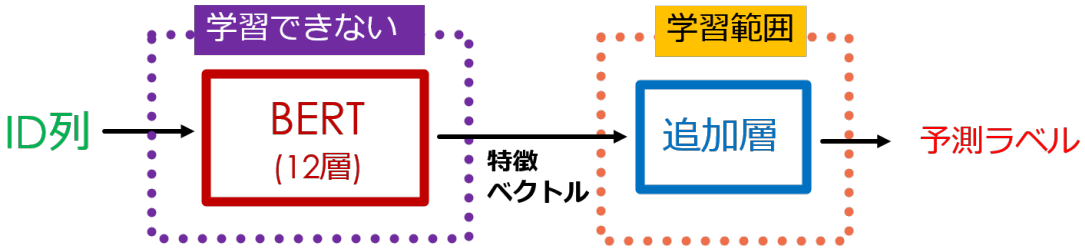


図 3.1: 先行研究手法を BERT で採用する場合の学習範囲

そこで，本論文では以下の手法を提案する．

## 3.1 データの Mix 方法

2 つの文書において，BERT に入力する際の ID 列同士を連結するという手法である．id”2”は文の先頭を表し，後続の ID 列には不要なので除いた．この手法であれば文書の特徴ベクトルを得る BERT 部分の学習が可能である (図 3.2 参照)．また，BERT の入力最大列長は 512 のため，それを超えないように前半と後半の ID 列長はそれぞれ 252 ま

での制限をかけた.

(例)

**1 つ目の ID 列**

[2, 6259, 9, 12396, 14, 3596, 3]

**2 つ目の ID 列**

[2, 11475, 9, 3741, 5, 12098, 75, 3]

**Mix-up 後の新たな ID 列**

[2, 6259, 9, 12396, 14, 3596, 3, 11475, 9, 3741, 5, 12098, 75, 3]

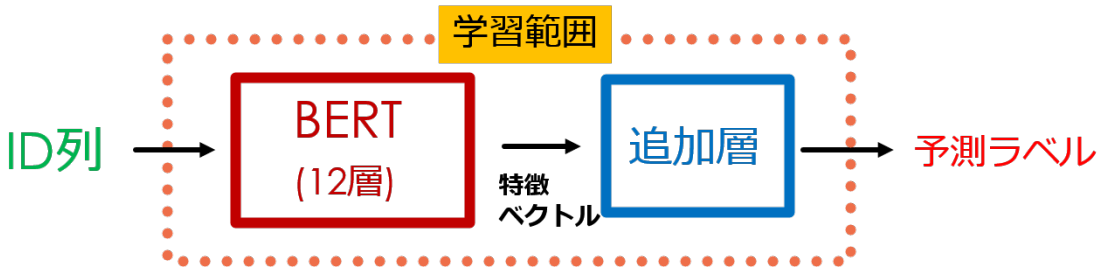


図 3.2: 本研究手法での学習範囲

## 3.2　ラベルの Mix 方法

各ラベルをまず 0 と 1 からなる one-hot ベクトルで表現した．Mix させる場合は 0.5 が 2 箇所で残りが 0 のベクトルを作成した．Mix 後のラベルが 0.5 同士なのは 2 つの文書が等価値に混ざっていることを表している.

(例)

**ラベル 3**

[0, 0, 0, 1, 0, 0, 0, 0, 0]

**ラベル 6**

[0, 0, 0, 0, 0, 0, 1, 0, 0]

**ラベル 3 と 6 の Mix**

[0, 0, 0, 0.5, 0, 0, 0.5, 0, 0]

# 第 4 章

# 実験

## 4.1 概要

Mix-Up 手法を用いることで訓練データ数を増加させた状態での BERT による文書分類と，BERT を使った通常の文書分類の精度を比較し，本研究手法の効果を調べた.

## 4.2 条件

### 4.2.1 実行環境

実験は Google Colaboratory[*1] の GPU(Tesla T4) 環境を利用して行った.

### 4.2.2 使用した BERT モデル

BERT は，東北大学 乾・鈴木研究室が作成した日本語版 BERT の事前学習モデル[*2] のうちの一つである，「bert-base-japanese-whole-word-masking」を使用した.

### 4.2.3 使用したコーパス

livedoor ニュースコーパス[*3] を使用し，以下の 9 ジャンルに対しての文書分類を行った.

---

[*1] https://colab.research.google.com/notebooks/intro.ipynb

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

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

- ラベル 0：独女通信
- ラベル 1：IT ライフハック
- ラベル 2：家電チャンネル
- ラベル 3：livedoor HOMME
- ラベル 4：MOVIE ENTER
- ラベル 5：Peachy
- ラベル 6：エスマックス
- ラベル 7：Sports Watch
- ラベル 8：トピックニュース

## 4.3　実験手順・手法

### 4.3.1　データの準備

本実験では livedoor ニュースコーパスより 6623 個の記事 (本文) を抽出し，表 4.1 のように振り分けた.

表 4.1: 使用データの内訳

| ラベル | train | val | test | sum |
|---|---|---|---|---|
| 0 | 87 | 128 | 566 | 781 |
| 1 | 87 | 125 | 571 | 783 |
| 2 | 86 | 111 | 581 | 778 |
| 3 | 51 | 72 | 335 | 458 |
| 4 | 87 | 114 | 582 | 783 |
| 5 | 84 | 106 | 565 | 755 |
| 6 | 87 | 106 | 590 | 783 |
| 7 | 90 | 131 | 589 | 810 |
| 8 | 77 | 107 | 508 | 692 |
| sum | 736 | 1000 | 4887 | 6623 |

train は訓練時に用いたデータ，val はハイパーパラメータを調整するために分類精度

をこまめに確認する際に使用した検証用データ，test は出来上がったモデルの最終的な分類精度を確認する際に使用したテストデータである．

また，以降の手順において各文書は BERT により形態素解析と ID 化を済ませた状態で扱っている．

### 4.3.2　訓練データの Mix-Up

訓練データについて，Mix-Up 手法を用いてデータ拡張を行った．Mix させる文書の選出方法は以下の 2 通りを試した．

**Mix させる文書の選出方法 (1)**

1 つ目の選出方法は全てのラベルの文書をランダムで Mix させるというものである．乱数を用いて 736 個の文書をランダムに並び替え，一番目から順に隣り合った 2 つの文書 (とそのラベル) を Mix させた．この方法の場合，文書間の隙間の数だけペアが作られるため，735(736-1) 個の拡張データができる．その結果訓練データは 736 個から 1471 個に拡張した．また，プログラムを起動するたびに選出される組み合わせは変わるようになっている．

**Mix させる文書の選出方法 (2)**

2 つ目の選出方法は，不足しているラベルの文書を対象に Mix-Up によってデータ数を増加させるというものである (図 4.1 参照)．本実験では表 4.1 に示す通り，訓練データではラベル 3 の文書が不足しているため，ラベル 3 の文書が必ず選出されるような動作にした．具体的には，51 個のラベル 3 文書からランダムに 1 つ選び，次にラベル 3 以外の 685 個の文書からもランダムに 1 つ選び，その 2 つの文書を Mix させるという手順を繰り返した (51 × 15 ループ)．連結の順序はラベル 3 の文書が前半でラベル 3 以外の文書が後半である．この動作により 765 個の新たな訓練データを作成し，訓練データは 1501 個に拡張した．また，選出方法 (1) と同じようにプログラムを起動するたびに組み合わせは変わるようになっている．

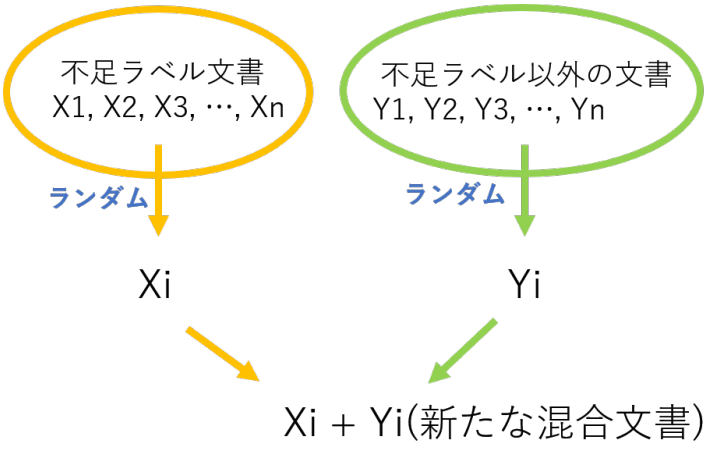


図 4.1: 選出方法 (2)

### 4.3.3　作成した分類器

分類器として使ったニューラルネットワークのモデルは BERT の直後に全結合層を一層追加したものを用いた．長さ 512 以下の ID 列を BERT に入力し，768 次元の文書の特徴ベクトルを出力として得る．それを全結合層に入力し，9 次元の各ラベルに対する予測値を出力として得るという流れである．細かな設定を以下に示す．

**損失関数**

交差エントロピー：　今回，ラベルは one-hot 表現になっており，nn.CrossEntropyLoss() に直接入力することはできないため，nn.LogSoftmax() を利用し，交差エントロピーの定義 (式 4.1) 通りに計算して損失を求めた．本実験ではバッチを使っているため，式 4.1 のバッチ平均 (式 4.2) が損失値となる．

$$E = -\sum_{k} t_k \log y_k \tag{4.1}$$

$$E = -\frac{1}{B}\sum_{b}\sum_{k} t_k \log y_k \tag{4.2}$$

$t_k$ は正解値，$y_k$ は予測値，B はバッチサイズである．

**最適化関数**

確率的勾配降下法 (SGD)：学習率は検証データを用いて分類精度と学習効率の両方を考慮 (図 4.2 参照) した結果，0.01 を採用した．

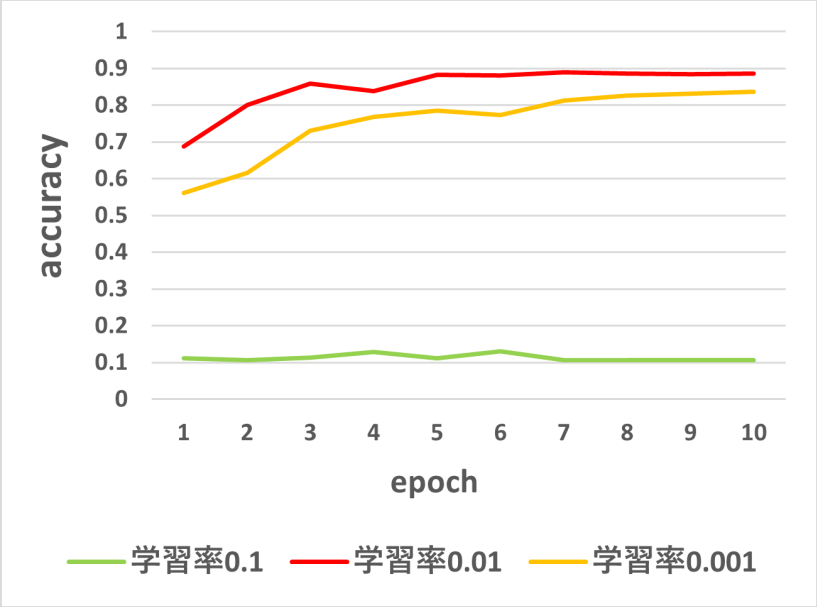


図 4.2: 検証データに対する学習率ごとの正解率 (Mix-Up なし)

※図 4.2 は各学習率ごとに 1 つのモデルの結果を掲載しており，複数モデルの平均を出しているわけではない．

**訓練データのバッチサイズ**

バッチサイズは実行環境である Google Colaboratory にて可能な値における最大値であった 10 を採用した．

**エポック数**

検証データでの分類精度の上がり幅を考慮 (図 4.2 参照) した結果，10 エポック学習すれば精度は収束値に達していると判断した．

## 4.4　実験結果

通常版，選出方法 (1) の Mix-Up 版，選出方法 (2) の Mix-Up 版の 3 つにおいて，訓練データで 10 エポック学習したモデルをそれぞれ 10 個用意し，テストデータに対する正解率を求めた．各モデルを比較した箱ひげ図を図 4.3 に示す．また，平均値の比較を表 4.2 に示す．※正解率は小数点第 4 位を四捨五入している．

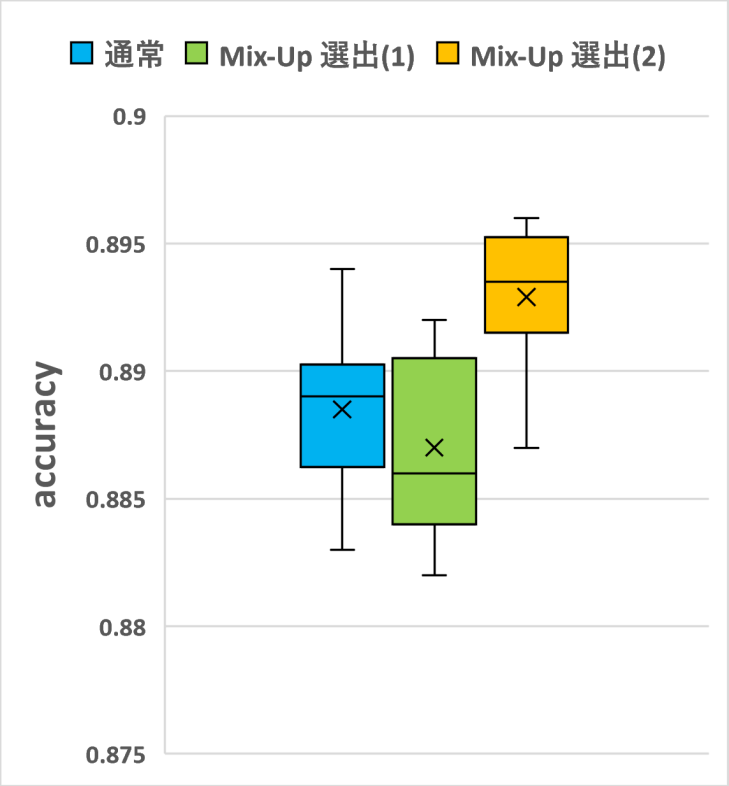


図 4.3: 実験結果

表 4.2: 平均値の比較

| | 正解率 (平均) |
|---|---|
| 通常 | 0.889 |
| Mix-Up 選出 (1) | 0.887 |
| Mix-Up 選出 (2) | 0.893 |

精度の高い順に，選出方法 (2) の Mix-Up，通常の BERT，選出方法 (1) の Mix-Up という結果になった．

# 第 5 章

# 考察

選出方法 (1) では通常よりも精度が落ち，選出方法 (2) では精度が上がっていることから Mix させる文書の選出方法が精度に大きく影響を与えることがわかった．本実験からは，不足しているラベルを中心に Mix させてデータ拡張していくのが有効であると言える．様々な選出方法を試して，最終的な結論を出したい．

また，今回は Mix させる二つの文書を等価値 (割合を 0.5 : 0.5) にして実験を行ったが，連結させる ID 列長に差を設けてその割合をラベルの one-hot 表現に採用するといった手法も試してみる価値はあると考えている．

Mix-Up は nlp における他のデータ拡張手法に比べ実装が容易で，かつ精度の向上が示されたことから，今後，主流な手法として広まることが期待できる．

# 第 6 章

# 結論

本論文では BERT を用いた文書分類タスクに Mix-Up 手法を試みた．具体的には BERT は 2 文の入力が可能であるため，ラベルの異なる 2 文書の ID 列を連結したものを入力とし，one-hot ベクトルにて 0.5 の要素が二つあるラベルを作り、マルチクラスの出力を教師データとした．livedoor ニュースコーパスを用いた実験を行った結果，訓練データにおいて不足しているラベルの文書を優先して選出しデータ拡張を行った際に，通常の文書分類に比べ精度が向上することが分かった．これにより，Mix-up に用いる文書の選出方法が結果に大きく影響を与えることが示された．

今後は，Mix-Up に用いる文書の組み合わせ方，文書とラベルそれぞれの混合方法について，他の手法の提案・実験を重ね，さらなる理解・進展を図りたい．

# 謝辞

最後に，本論文を作成するにあたり指導・助言を頂いた，指導教官の新納浩幸教授に心より感謝申し上げます．また，普段の勉強会にて指摘・助言を頂いた新納研究室の皆さんに感謝致します．

# 参考文献

[1] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, Vol. 6, No. 1, p. 60, 2019.

[2] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2017.

[3] Hongyu Guo, Yongyi Mao, and Richong Zhang. Augmenting data with mixup for sentence classification: An empirical study, 2019.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers forlanguage understanding, 2018.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

# 付録

## A　本実験で使用したプログラム

選出方法 (1) での Mix-Up のソースコードを A.1 に，選出方法 (2) での Mix-Up のソースコードを A.2 に，モデルを評価する際のソースコードを A.3 に示す．※以下のプログラムは掲載の都合上.py 形式だが，実験の際は.ipynb 形式で Google Colaboratory で実行した．

ソースコード A.1: mixup.py

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
from transformers import BertModel,BertJapaneseTokenizer
import numpy as np
import pickle
import sys
import re
import random

bert = BertModel.from_pretrained('cl-tohoku/bert-base-japanese-whole-word-masking')
tokenizer = BertJapaneseTokenizer.from_pretrained('cl-tohoku/bert-base-japanese-whole-word-masking')

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)

```

```
X_train = []
with open('/content/drive/My␣Drive/livedoor/xtrain.pkl','br') as fr:
    X_train = pickle.load(fr)

y_train = []
with open('/content/drive/My␣Drive/livedoor/ytrain.pkl','br') as fr:
    y_train = pickle.load(fr)

y_train_oh = np.identity(9)[y_train].tolist()

l = list(range(736))
r = random.sample(l,736)

#ランダムに全ラベルをミックス
for n in range(len(X_train)-1):
  Xf = X_train[r[n]][:256]
  Xr = X_train[r[n+1]][1:257]
  lf = len(Xf)
  lr = len(Xr)
  Xf[lf-1] = 3
  Xr[lr-1] = 3
  X_mix = Xf + Xr
  X_train.append(X_mix)

  y_mix = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
  y_mix[y_train[r[n]]] = 0.5
  y_mix[y_train[r[n+1]]] = 0.5
  y_train_oh.append(y_mix)

class MyDataset(Dataset):
    def __init__(self, xdata, ydata):
        self.data = xdata
        self.label = ydata
    def __len__(self):
        return len(self.label)
    def __getitem__(self, idx):
        x = self.data[idx]
        y = self.label[idx]
        return (x,y)

def my_collate_fn(batch):
```

```
61      images, targets= list(zip(*batch))
62      xs = list(images)
63      ys = list(targets)
64      return xs, ys
65
66  batch_size = 10
67  dataset = MyDataset(X_train,y_train_oh)
68  dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True,
        collate_fn=my_collate_fn)
69
70  class DocCls(nn.Module):
71      def __init__(self,bert):
72          super(DocCls, self).__init__()
73          self.bert = bert
74          self.cls=nn.Linear(768,9)
75      def forward(self,x1,x2):
76          b_output = self.bert(input_ids=x1,attention_mask=x2)
77          bs = len(b_output[0])
78          h0 = [ b_output[0][i][0] for i in range(bs)]
79          h0 = torch.stack(h0,dim=0)
80          h1 = self.cls(h0)
81          return h1
82
83  model = DocCls(bert)
84  model.to(device)
85  optimizer = optim.SGD(model.parameters(),lr=0.01)
86
87  model.train()
88
89  epoch_num = 10
90
91  for ep in range(epoch_num):
92    print("---------------",ep+1,"エポック目
          ----------------------------------------------------")
93    cnt = 1
94    for xs, ys in dataloader:
95      xs1, xmsk = [], []
96      for k in range(len(xs)):
97        tid = xs[k]
98        xs1.append(torch.LongTensor(tid))
99        xmsk.append(torch.LongTensor([1] * len(tid)))
```

```
100
101     xs1 = pad_sequence(xs1, batch_first=True).to(device)
102     xmsk = pad_sequence(xmsk, batch_first=True).to(device)
103
104     outputs = model(xs1,xmsk)
105     #print("outputs:" , outputs)
106
107     ys = torch.LongTensor(ys).to(device)
108     #print("ys: ", ys)
109
110     logsoftmax = nn.LogSoftmax()
111     outputs_logsoft = logsoftmax(outputs)
112     #print("outputs_logsoft:" , outputs_logsoft)
113
114     #print("ys * outputs_logsoft: " ,ys * outputs_logsoft)
115
116     loss = (ys * outputs_logsoft).sum(axis=1)
117     loss = -1*(torch.sum(loss))/batch_size
118
119     print(cnt,"バッチ目␣","␣loss:",loss.item())
120     optimizer.zero_grad()
121     loss.backward()
122     optimizer.step()
123     cnt += 1
124
125   outfile = "/content/drive/MyDrive/bert/model/mix010505_lr001_ep_"+str
          (ep+1)+".model"
126   torch.save(model.state_dict(),outfile)
127   print(outfile,"␣saved")
```

ソースコード A.2: mixup-label3.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,BertJapaneseTokenizer
8 import numpy as np
9 import pickle
```

```
10 import sys
11 import re
12 import random
13
14 bert = BertModel.from_pretrained('cl-tohoku/bert-base-japanese-whole-
       word-masking')
15 tokenizer = BertJapaneseTokenizer.from_pretrained('cl-tohoku/bert-base-
       japanese-whole-word-masking')
16
17 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu"
       )
18 print(device)
19
20 X_train = []
21 with open('/content/drive/My␣Drive/livedoor/xtrain.pkl','br') as fr:
22     X_train = pickle.load(fr)
23
24 y_train = []
25 with open('/content/drive/My␣Drive/livedoor/ytrain.pkl','br') as fr:
26     y_train = pickle.load(fr)
27
28 y_train_oh = np.identity(9)[y_train].tolist()
29
30 X_train_3 = []
31 y_train_3 = []
32
33 for a in range(len(y_train)):
34   if y_train[a] == 3:
35     X_train_3.append(X_train[a])
36     y_train_3.append(y_train[a])
37
38 X_train_not3 = []
39 y_train_not3 = []
40
41 for a in range(len(y_train)):
42   if y_train[a] != 3:
43     X_train_not3.append(X_train[a])
44     y_train_not3.append(y_train[a])
45
46 l3 = list(range(len(X_train_3)))
47 r3 = random.sample(l3,len(X_train_3))
```

```
lnot3 = list(range(len(X_train_not3)))
rnot3 = random.sample(lnot3,len(X_train_not3))

#ラベル固定で他ラベルとミックス 3
for a in range(15):
  for b in range(len(X_train_3)):
    Xf3 = X_train_3[r3[b]][:256]
    Xr3 = X_train_not3[rnot3[b+a]][1:257]
    lf3 = len(Xf3)
    lr3 = len(Xr3)
    Xf3[lf3-1] = 3
    Xr3[lr3-1] = 3
    X3_mix = Xf3 + Xr3
    X_train.append(X3_mix)

    y3_mix = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
    y3_mix[3] = 0.5
    y3_mix[y_train_not3[rnot3[b+a]]] = 0.5
    y_train_oh.append(y3_mix)

class MyDataset(Dataset):
    def __init__(self, xdata, ydata):
        self.data = xdata
        self.label = ydata
    def __len__(self):
        return len(self.label)
    def __getitem__(self, idx):
        x = self.data[idx]
        y = self.label[idx]
        return (x,y)

def my_collate_fn(batch):
    images, targets= list(zip(*batch))
    xs = list(images)
    ys = list(targets)
    return xs, ys

batch_size = 10
dataset = MyDataset(X_train,y_train_oh)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True,
    collate_fn=my_collate_fn)
```

```
88
89 class DocCls(nn.Module):
90     def __init__(self,bert):
91         super(DocCls, self).__init__()
92         self.bert = bert
93         self.cls=nn.Linear(768,9)
94     def forward(self,x1,x2):
95         b_output = self.bert(input_ids=x1,attention_mask=x2)
96         bs = len(b_output[0])
97         h0 = [ b_output[0][i][0] for i in range(bs)]
98         h0 = torch.stack(h0,dim=0)
99         h1 = self.cls(h0)
100        return h1
101
102 model = DocCls(bert)
103 model.to(device)
104 optimizer = optim.SGD(model.parameters(),lr=0.01)
105
106 model.train()
107
108 epoch_num = 10
109
110 for ep in range(epoch_num):
111   print("---------------",ep+1,"エポック目
          ------------------------------------------------")
112   cnt = 1
113   for xs, ys in dataloader:
114     xs1, xmsk = [], []
115     for k in range(len(xs)):
116       tid = xs[k]
117       xs1.append(torch.LongTensor(tid))
118       xmsk.append(torch.LongTensor([1] * len(tid)))
119
120     xs1 = pad_sequence(xs1, batch_first=True).to(device)
121     xmsk = pad_sequence(xmsk, batch_first=True).to(device)
122
123     outputs = model(xs1,xmsk)
124     #print("outputs:" , outputs)
125
126     ys = torch.LongTensor(ys).to(device)
127     #print("ys: ", ys)
```

```
128
129     logsoftmax = nn.LogSoftmax()
130     outputs_logsoft = logsoftmax(outputs)
131     #print("outputs_logsoft:" , outputs_logsoft)
132
133     #print("ys * outputs_logsoft: " ,ys * outputs_logsoft)
134
135     loss = (ys * outputs_logsoft).sum(axis=1)
136     loss = -1*(torch.sum(loss))/batch_size
137
138     print(cnt,"バッチ目␣","␣loss:",loss.item())
139     optimizer.zero_grad()
140     loss.backward()
141     optimizer.step()
142     cnt += 1
143
144 outfile = "/content/drive/MyDrive/bert/model/mix010505_label3_lr001_ep"
        +str(ep+1)+".model"
145 torch.save(model.state_dict(),outfile)
146 print(outfile,"␣saved")
```

ソースコード A.3: cls-test.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,BertJapaneseTokenizer
8  import numpy as np
9  import pickle
10 import sys
11 import re
12
13 bert = BertModel.from_pretrained('cl-tohoku/bert-base-japanese-whole-
       word-masking')
14 tokenizer = BertJapaneseTokenizer.from_pretrained('cl-tohoku/bert-base-
       japanese-whole-word-masking')
15
16 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu"
```

```
    )
print(device)

X_test = []
with open('/content/drive/MyDrive/livedoor/xtest.pkl','br') as fr:
    X_test = pickle.load(fr)

y_test = []
with open('/content/drive/MyDrive/livedoor/ytest.pkl','br') as fr:
    y_test = pickle.load(fr)

X_val = X_test[:1000]
X_test = X_test[1000:]
y_val = y_test[:1000]
y_test = y_test[1000:]

class MyDataset(Dataset):
  def __init__(self, xdata, ydata):
    self.data = xdata
    self.label = ydata
  def __len__(self):
    return len(self.label)
  def __getitem__(self, idx):
    x = self.data[idx]
    y = self.label[idx]
    return (x,y)

def my_collate_fn(batch):
  images, targets= list(zip(*batch))
  xs = list(images)
  ys = list(targets)
  return xs, ys

batch_size_val = 100
dataset_val = MyDataset(X_val,y_val)
dataloader_val = DataLoader(dataset_val, batch_size=batch_size_val,
    shuffle=True, collate_fn=my_collate_fn)

batch_size_test = 100
dataset_test = MyDataset(X_test,y_test)
dataloader_test = DataLoader(dataset_test, batch_size=batch_size_test,
```

```
        shuffle=True, collate_fn=my_collate_fn)
56
57 class DocCls(nn.Module):
58     def __init__(self,bert):
59         super(DocCls, self).__init__()
60         self.bert = bert
61         self.cls=nn.Linear(768,9)
62     def forward(self,x1,x2):
63         b_output = self.bert(input_ids=x1,attention_mask=x2)
64         bs = len(b_output[0])
65         h0 = [ b_output[0][i][0] for i in range(bs)]
66         h0 = torch.stack(h0,dim=0)
67         h1 = self.cls(h0)
68         return h1
69
70 for a in range(10):
71   model = DocCls(bert)
72   model.load_state_dict(torch.load("/content/drive/MyDrive/bert/model/
         mix010505_label3_lr001_ep10_"+str(a+1)+".model"))
73   model.to(device)
74   model.eval()
75   with torch.no_grad():
76     ok = 0
77     n = 0
78     i = 1
79     for xs, ys in dataloader_test:
80       n += len(ys)
81       xs1, xmsk = [], []
82       for k in range(len(xs)):
83         tid = xs[k]
84         xs1.append(torch.LongTensor(tid))
85         xmsk.append(torch.LongTensor([1] * len(tid)))
86       xs1 = pad_sequence(xs1, batch_first=True).to(device)
87       xmsk = pad_sequence(xmsk, batch_first=True).to(device)
88       ans = model(xs1,xmsk)
89       ans = ans.to('cpu')
90       ans = torch.argmax(ans,dim=1)
91       #print予測値 (":",ans)
92       ys1 = torch.LongTensor(ys)
93       #print正解値 (":",ys1)
94       #print(iバッチ目," 正解数:",(ys1 == ans).sum().int().item())
```

```
95        ok += (ys1 == ans).sum().float().item()
96        i += 1
97      print("model"+str(a+1)+"␣正解率:",int(ok),"/",n,"␣=␣",ok/n)
```