令和 3 年度茨城大学工学部情報工学科卒業研究論文

BERT の転移学習と Mis-leading データの削除による識別精度の改善

所属 情報工学科

著者 岩本昇太 (18T4012T)

指導教員 新納浩幸教授

令和 4 年 2 月 4 日 (金)

令和 3 年度茨城大学工学部情報工学科卒業研究論文

BERT の転移学習と Mis-leading データの削除による識別精度の改善

著者

岩本昇太 (18T4012T)

指導教員

新納浩幸教授

論文要旨

自然言語処理のタスクの多くは，機械学習の手法により解決できる．ただし，そのためには大量のラベル付き訓練データが必要となる．ターゲット領域には十分な量のラベル付き訓練データが存在しないことがあるため，そのような場合にも対応できる手法が必要とされている．

BERT のような事前学習済みモデルを用いる手法はターゲット領域のラベル付き訓練データの必要性を低減させる上で有用な手法である．fine-tuning 前にドメイン適応型事前学習・タスク適応型事前学習を行うことで識別精度を更に改善する手法も提案されている．ただ，事前学習済みモデルを用いる手法でも fine-tuning に用いるターゲット領域のラベル付き訓練データは必要である．転移学習はターゲット領域のラベル付き訓練データが全く無い場合にも有効な手法であるが，「負の転移」と呼ばれる問題に対処する必要がある．ドメイン適応型事前学習・タスク適応型事前学習を行っても，負の転移の問題が解消されるわけではない．

本稿では BERT を用いた文書分類タスクで転移学習を行うとき Mis-leading データの削除と BERT の追加学習 により識別精度を改善する手法を提案する．Mis-leading データの削除は，ソース領域のラベル付き訓練データのうち負の転移を引き起こすおそれがあるものを削除してからラベル付き訓練データをモデルの fine-tuning に使用する手法である．BERT の追加学習は，タスク適応型事前学習により識別精度を更に改善する手法である．Webis-CLS-10 データセットを用いた実験により，文書分類タスクで識別精度を改善できることを確認した．

目次

表目次

図目次

第 1 章

序論

自然言語処理のタスクの多くは，機械学習の手法により解決できる．ただし，そのためには大量のラベル付き訓練データが必要となる．BERT [1] のような事前学習済みモデルを用いる場合も，fine-tuning のためのラベル付き訓練データは必要となる．転移学習を用いれば，ターゲット領域の訓練データが全く無い場合にも対応できる．しかし，転移学習を用いる場合は負の転移 [2] と呼ばれる問題に対処する必要がある．負の転移は，ソース領域のデータとターゲット領域のデータの性質が著しく異なるときにソース領域のデータを訓練に用いると識別精度が悪化する現象である．

本稿では，BERT を用いた文書分類タスクで転移学習を行うときに Mis-leading データの削除と BERT の追加学習を行い，識別精度の改善を試みる．Mis-leading データは，ソース領域の訓練データのうちターゲット領域での識別精度に悪影響を及ぼすものである．事前に Mis-leading データを削除することで，ターゲット領域での識別精度の向上が期待できる．また，ターゲット領域のラベルなしデータを用いて BERT の追加学習を行うことで，fine-tuning 後の識別精度の向上が期待できる [3]．Mis-leading データの削除に加えて BERT の追加学習も行い，識別精度のさらなる改善を図る．

第 2 章

関連研究

2.1 Bag-of-Words

自然言語で記述された文書を機械学習の手法で処理するためには，文書をベクトルに変換する必要がある．Bag-of-Words は，単語の出現回数を用いて文書をベクトルに変換する手法である．Bag-of-Words により文書をベクトルに変換する手順は次の通りである．

1. 文書を単語分割する．
2. 文書に登場する各単語に対して一意な id を割り当て，辞書（単語と id の対応表）を作成する．なお，辞書はベクトルに変換する文書とは別の文書や文書の集合を基に作成したものを用いてもよい．
3. 辞書に登録されている各単語が文書中に登場する回数を数え，各単語が文書中に登場する回数を並べてベクトルにする．

Bag-of-Words により「私は茨城大学の日立地区の学生です」という文をベクトルに変換する例を示す．

1. 文を単語分割すると次のようになる．

   私 / は / 茨城大学 / の / 日立 / 地区 / の / 学生 / です
2. 文書に登場する各単語に対して一意な id を割り当てると，辞書は表 2.1 のようになる．
3. 辞書に登録されている各単語が文書中に登場する回数を数え，各単語が文書中に登場する回数を並べると表 2.2 の 3 行目のようになる．したがって，特徴ベクト

表 2.1: Bag-of-Words で用いる辞書の例

単語	私	は	茨城大学	の	日立	地区	学生	です
id	1	2	3	4	5	6	7	8

表 2.2: Bag-of-Words で文書をベクトルに変換した例

単語	私	は	茨城大学	の	日立	地区	学生	です
id	1	2	3	4	5	6	7	8
登場回数	1	1	1	2	1	1	1	1

ルは $(1, 1, 1, 2, 1, 1, 1, 1)$ となる．

Bag-of-Words により作成したベクトルの次元数は辞書に登録された単語の数である．よって，ベクトルの次元数が数万から数百万に及ぶこともある．

2.2　ニューラルネットワーク

ニューラルネットワークは，入力となるベクトルを線形変換と非線形変換の繰り返しによって変換する手法である．後述するように，ニューラルネットワークは関数 $f : \mathbb{R}^m \to \mathbb{R}^n$ とみなせる．

2.2.1　ニューラルネットワークの構造

図 2.1 は入力層・中間層（隠れ層）・出力層の 3 層で構成されるニューラルネットワークである．入力層のユニットは入力された値をそのまま出力する．中間層のユニットに入力される値は，入力層のユニットから出力された値とユニット間で定められた重みの積である．中間層のユニットから出力される値は，ユニットへ入力された値の総和に活性化関数を適用した後の値である．出力層のユニットへの入力は，入力層から中間層へのそれと同様である．出力層のユニットから出力される値は，ユニットへ入力された値の総和である．なお，バイアスのユニットから出力される値は 1 に固定されており，バイアスのユニットと次の層のユニットの間で定められた重みが次の層のユニットへの入力となる．

活性化関数には様々なものがあるが，例えば式 (2.1) で表されるシグモイド関数 σ が

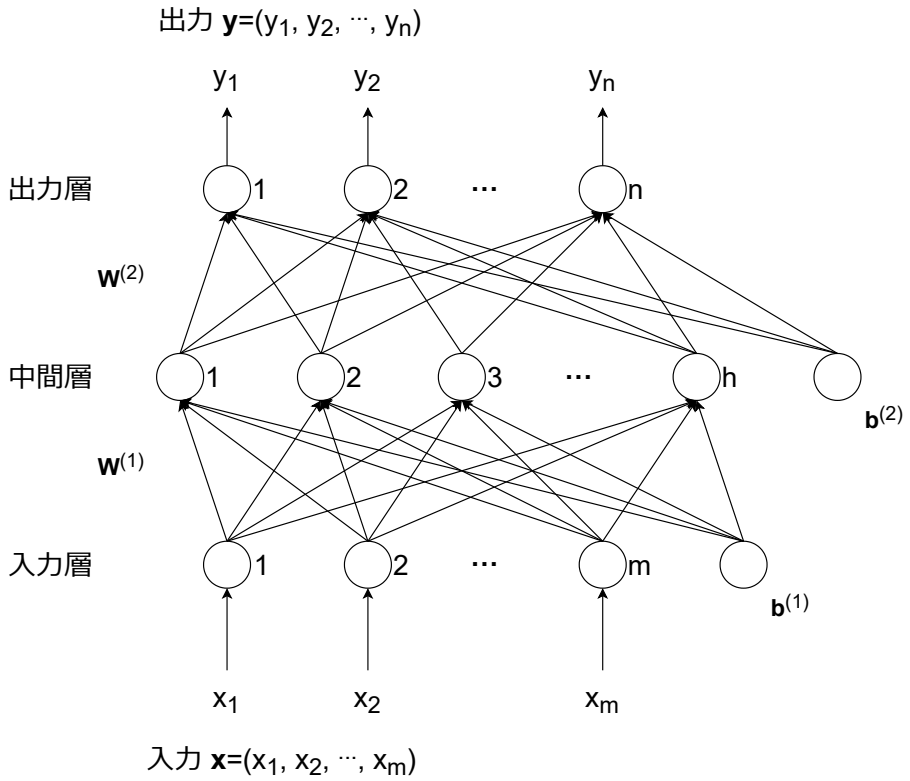

図 2.1: 3 層のニューラルネットワーク．図中の丸がユニット．入力層には $m+1$ 個，中間層には $h+1$ 個，出力層には n 個のユニットがある．入力層と中間層にある $\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ はバイアス．ユニットから出力される値にユニットとユニットの間で定められた重みの積が次の層のユニットへ入力されるが，ユニット間で定められた重みは行列 $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}$ で表現できる．

使われる．

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

ユニット間で定められた重みを行列 $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}$ で表現し，活性化関数として σ を用いると，図 2.1 のニューラルネットワークは $\boldsymbol{y} = f(\boldsymbol{x}) = \boldsymbol{W}^{(2)}\sigma(\boldsymbol{W}^{(1)}\boldsymbol{x})$ と表せる．

一般に，入力層のユニットが m 個，出力層のユニットが n 個のニューラルネットワークは関数 $f : \mathbb{R}^m \to \mathbb{R}^n$ とみなせる．図 2.1 のニューラルネットワークは中間層が 1 層のみであったが，中間層を更に増やしたニューラルネットワークをディープニューラルネットワークという．ディープニューラルネットワークを用いた機械学習がディープラーニングである．

2.2.2　ニューラルネットワークの学習

ニューラルネットワークの学習は，出力層から望ましい結果が出力されるようにユニット間で定められた重みを最適化する作業である．厳密には，$f : \mathbb{R}^m \to \mathbb{R}^n$ でパラメータ $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_K)$ を持つ関数 $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ のパラメータ $\boldsymbol{\theta}$ を訓練データ（入力データとそれに対応する出力データの組）$\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})\}$ から推定する問題である．図 2.1 のニューラルネットワークの場合は，訓練データを用いて行列 $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}$ の値を最適化する作業が学習となる．

学習を行うときは，モデル（ここではニューラルネットワーク）のパラメータ $\boldsymbol{\theta}$ を変数とする損失関数を設定し，損失関数を最小化するようなパラメータを求める．損失関数はモデルの出力と実際の正解との誤差を計算する関数である．様々な損失関数があるが，例えば式 (2.2) で表される平均二乗誤差が使われる．

$$MSE(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{N} \| f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) - \boldsymbol{y}^{(i)} \|^2 \tag{2.2}$$

分類問題（識別の問題）を解く場合は，式 2.3 で表される交差エントロピー誤差を損失関数とすることがある．ただし，式 2.3 においては $\boldsymbol{y}^{(i)} = (y_1^{(i)}, y_2^{(i)}, \ldots, y_n^{(i)})$ であり，$f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) = (f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})_1, f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})_2, \ldots, f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})_n)$ である．また，$\boldsymbol{y}^{(i)}$ は one-hot ベクトル（正解のクラス k の要素 $y_k^{(i)}$ のみが 1 で他の要素は 0 のベクトル）であり，$f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$ は確率とみなせるものとする．つまり，すべての j $(j = 1, 2, \ldots, n)$ について $0 \leq f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})_j \leq 1$ を満たし，なおかつ $\sum_{j=1}^{n} f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})_j = 1$ を満たすものとする．交差エントロピー誤差を損失関数とする場合は，$f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$ を確率とみなせるようにするため，ニューラルネットワークが出力したベクトルを変換しなければならない場合がある．そのようなときによく用いられるのが，式 2.4 で表される softmax 関数である．

$$E(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \sum_{j=1}^{n} y_j^{(i)} \log f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})_j \tag{2.3}$$

$$y_i = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \quad (i = 1, 2, \ldots, n) \tag{2.4}$$

損失関数を最小化するようなパラメータを解析的に求めることは一般には困難であるため，数値解析の手法で求めるのが一般的である．実際には，勾配降下法もしくはそれを

改良したアルゴリズムがよく用いられる．

2.3　埋め込み表現

自然言語で記述された文書を機械学習の手法で処理するためには，文書をベクトルに変換する必要がある．単語を（低次元の）実数ベクトルに変換することを単語埋め込みといい，単語埋め込みによって生成されたベクトルを埋め込み表現という．

埋め込み表現を生成するモデルとしては， Word2vec [4] が代表的である．Word2vec はニューラルネットワークを用いて埋め込み表現を生成するモデルであり，埋め込み表現の生成には Continuous Bag-of-Words(CBOW) モデルまたは Skip-gram モデルのいずれかを用いる．CBOW モデルでは，周辺の単語（文脈）をもとにしてある単語を予測する．Skip-gram モデルでは，ある単語をもとにして周辺の単語（文脈）を予測する．CBOW モデルのほうが処理速度は優れているが， Skip-gram モデルのほうが精度は優れているとされる．

2.4　Transformer

Transformer [5] は 2017 年に発表された深層学習のモデルである．Transformer は Attention 機構を基に作られたモデルであり，翻訳タスクで当時の State of the art を達成した．

Transformer 登場以前は時系列データの処理に RNN や LSTM が用いられていたが，RNN や LSTM ではデータを逐次的に処理するため並列計算が困難という問題があった．Transformer では並列計算が可能となっているため，適切なハードウェアを用いれば訓練時間を削減できる．

Bidirectional Encoder Representations from Transformers(BERT) は Transformer をベースとした事前学習済みモデルである．

2.5　BERT

Bidirectional Encoder Representations from Transformers(BERT) [1] は 2018 年に Google より公開された事前学習済みモデルである．構造としては， Transformer で用いられた Multi-Head Attention を 12 層重ねたものである．BERT は入力としてトーク

ン列（単語もしくはサブワード）を受け取り，それに対応する埋め込み表現列を出力する．ここで，出力される埋め込み表現は文脈を考慮したものとなっている．大規模コーパスにより事前学習したモデルを下流タスクのラベル付きデータで fine-tuning することで，文書分類・固有表現抽出・質問応答など様々なタスクに対応できる．日本語版の事前学習済みモデルとして，東北大学で公開されているモデル[*1]などがある．

2.5.1　入出力

BERT へ入力するものは，1 つまたは 2 つの文を単語分割することで作成したトークン列である．ただし，いくつかの特殊トークンを追加する．

- トークン列の先頭には，[CLS] トークンを追加する．
- トークン列の末尾には，[SEP] トークンを追加する．また，2 つの文を入力する場合は，文と文の間にも [SEP] トークンを追加する．

出力されるものは，入力された各トークンに対応する埋め込み表現列である．入出力の具体例を図 2.2 に示す．

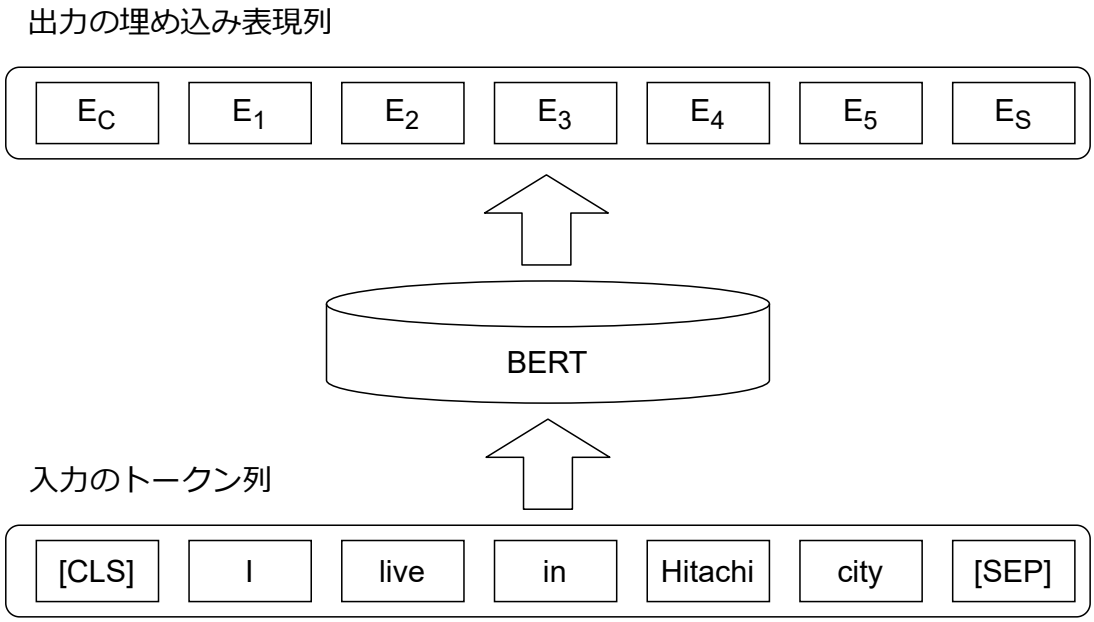

図 2.2: BERT の入出力の具体例

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

2.5.2　事前学習

先述した通り， BERT は事前学習と fine-tuning の 2 段階で学習を行う．事前学習で行うタスクは Masked Language Model(MLM) と Next Sentence Prediction(NSP) の 2 つである．

Masked Language Model

Masked Language Model は，入力となるトークンの一部を [Mask] トークンでマスクしたトークン列が与えられたとき，マスクする前のトークンが何であるかを当てるタスクである．つまり，穴埋め問題である．具体的な動作は以下の通りである．

1. 入力となるトークンの 15% がマスクする対象となり， [Mask] トークンに置き換えられる．
   例： I live in Hitachi City → I live in [Mask] City
2. [Mask] トークンのうち 80% はそのままとなる．
   例： I live in [Mask] City → I live in [Mask] City
   [Mask] トークンのうち 10% は元のトークンに置き換えられる．
   例： I live in [Mask] City → I live in Hitachi City
   [Mask] トークンのうち 10% はランダムに選ばれた別のトークンに置き換えられる．
   例： I live in [Mask] City → I live in Mito City
3. マスクされた部分について，文脈から元のトークンを予測する．

Next Sentence Prediction

Next Sentence Prediction は，入力として 2 つの文が与えられたとき，それが連続する 2 つの文であるか否かを当てるタスクである．具体的な動作は以下の通りである．

1. 2 文 1 組の連続した文を 1 組用意する． 50% の確率で後ろの文をランダムに選ばれた別の文に置き換える．
2. 2 つの文を入力する．2 つの文が連続していると予測した場合は IsNext, 連続していないと予測した場合は NotNext の判定を出す．

例 1： [CLS] I am a student at Ibaraki University [SEP] I live in Hitachi City [SEP]

判定： IsNext

例 2： [CLS] I am a student at Ibaraki University [SEP] This is a pen [SEP]

判定： NotNext

2.5.3　fine-tuning

BERT を文書分類などの下流タスクで利用するときは，出力層を 1 つ追加し，解きたいタスクのラベル付きデータで fine-tuning する．文書分類の場合は，[CLS] トークンに対応する出力を出力層に接続する．その後，文書分類のラベル付きデータで fine-tuning を行い， BERT と出力層の学習を行う．ただし， BERT は事前学習済みであるため，この段階での BERT の学習は微調整である．BERT を用いた文書分類器の構造を図 2.3 に示す．

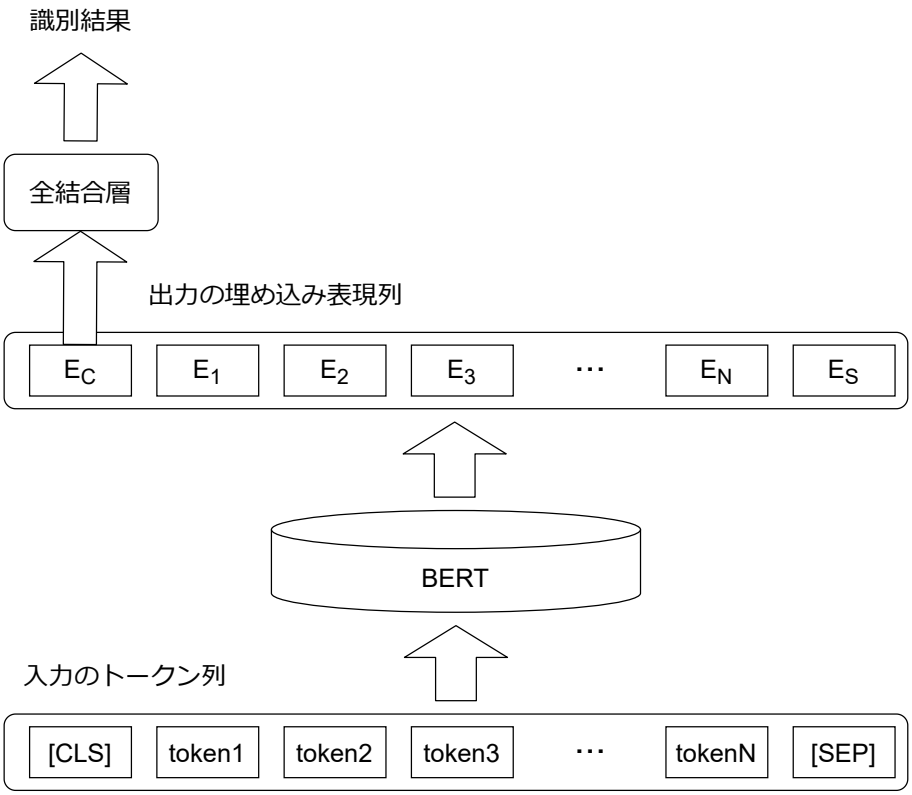

図 2.3: BERT を用いた文書分類器の構造．図中の全結合層が出力層となる．

2.6　低リソース環境での自然言語処理

自然言語処理のタスク機械学習の手法で解決するためには，大量のラベル付き訓練データが必要となる．しかし，大量のラベル付き訓練データを作成するためには膨大なリソースが必要となるため，ターゲット領域には十分な量のラベル付き訓練データが存在しないことがある．そのような場合に有用な手法として，論文 [6] ではデータ拡張 (Data Augmentation) と転移学習 (Transfer Learning) が挙げられている．

2.6.1　データ拡張

データ拡張 (Data Augmentation) は，既存のラベル付きデータを用いて新たなラベル付きデータを得る手法である．新たなラベル付きデータを低コストで得ることにより，ラベル付きデータの不足に対処しようとする手法といえる．画像分野では，図 2.4 のように既存のラベル付きの画像に対して回転などの加工を施して新たなラベル付きデータを得る手法が用いられることがある．画像に回転などの加工を施してもラベル（図 2.4 の場合は「猫」）は変わらないため，ラベル付けのコストはかからない．

自然言語処理においても，同義語置換や逆翻訳によるデータ拡張が行われることがある．同義語置換は，文書中のある単語を同じ意味の別単語に置き換えることである．例えば

私は卒業論文を作成します

という文の「作成」を「制作」に置き換えると

私は卒業論文を制作します

という文を得られる．逆翻訳は，文を機械翻訳などで別の言語に翻訳したあと，翻訳後の文を元の言語に翻訳し直すことである．例えば

私はパンを食べます

という文を Google 翻訳で英語に翻訳すると

I eat bread

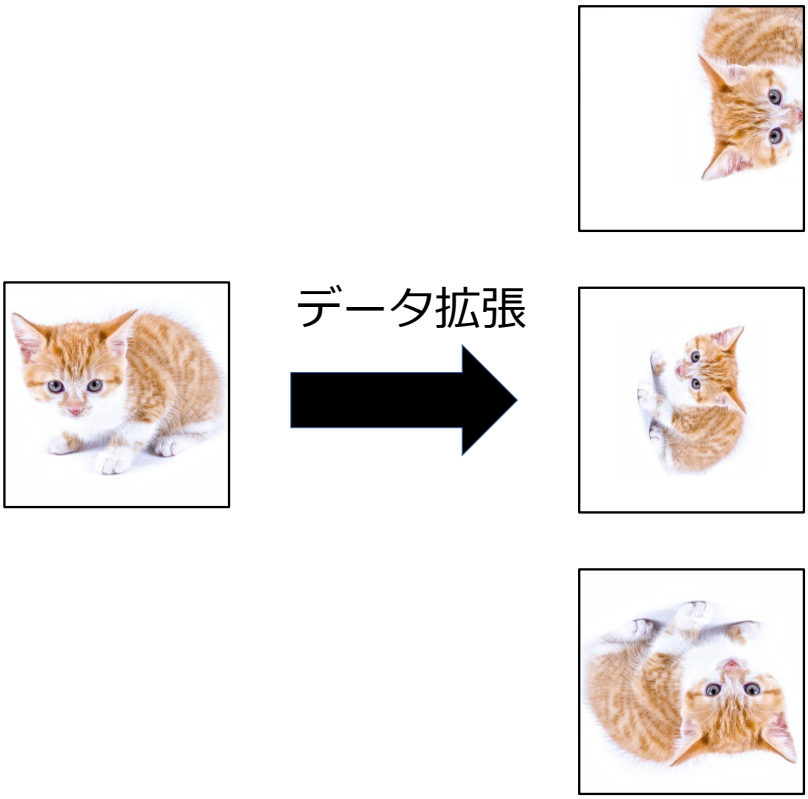

図 2.4: 画像分野でのデータ拡張の例．図の左側にある猫の画像に対して回転・拡大縮小・平行移動などの加工を施し，画像の枚数を増やしている．

という文を得られるが，この文を Google 翻訳で日本語に翻訳し直すと

パンを食べる

という文を得られる．文に対してこのような加工を施してもラベルは変わらないため，ラベル付けのコストはかからない．

2.6.2　転移学習

転移学習 (Transfer Learning) は，学習済みのモデルを転移させることでターゲット領域のラベル付きデータの必要性を低減させる手法である．十分なラベル付きデータが

有るソース領域で学習したモデルをそのままターゲット領域で使用する手法（図 2.5）や，学習済みのモデルをターゲット領域のラベル付きデータにより fine-tuning する手法（図 2.6）がある．十分なラベル付きデータが有るソース領域で学習したモデルをそのままターゲット領域で使用する手法ではターゲット領域のラベル付きデータを使わないため，ターゲット領域にラベル付きデータが全く無い場合にも対応できる．学習済みのモデルをターゲット領域のラベル付きデータにより fine-tuning する手法ではモデルを fine-tuning するためターゲット領域のラベル付きデータが必要となるが，転移学習を行わない場合と比較するとターゲット領域のラベル付きデータの所要量を抑えられる．

自然言語処理においては，大量のラベルなしデータを用いてモデルの事前学習を行い，その後少量のラベル付きデータを用いて事前学習済みモデルを fine-tuning する手法が用いられることがある．例えば BERT のような文脈を考慮した埋め込み表現列を生成するモデルがこれに該当する．事前学習済みの BERT は文書分類や固有表現抽出など様々な下流タスクに使い回せるため，異なる複数の下流タスクを解きたい場合も事前学習は一度だけ行えばよい．一般的に fine-tuning は事前学習よりも低コストで行えるため，既存の事前学習済みモデルを使用できる状況であれば，比較的少ない計算資源で異なる複数の下流タスクに対応できる．また各タスクにおけるラベル付きデータの所要量も抑えられるため有用である．

ただし，モデルの事前学習を行うためには大量のラベルなしデータと膨大な計算資源を要する．既存の事前学習済みモデルを使用できる場合は問題ないが，既存の事前学習済みモデルが存在せず，なおかつ利用可能な計算資源が限られている場合にこの手法を用いるのは難しい．実際に事前学習を行うとき，十分なリソースがある言語の場合は高品質なラベルなしデータを用意できるため問題ないが，リソースが限られている言語の場合はラベルなしデータであっても品質の低いデータしか用意できないおそれがあるという問題もある．

また，モデルの事前学習に用いられるラベルなしデータとして，自然言語処理の場合はニュースや Wikipedia の文章など一般的な領域のデータを用いる場合が多い．一般的な領域のデータを用いて事前学習を行ったモデルを科学論文など特殊な領域に適用すると，識別精度が悪化することがある．

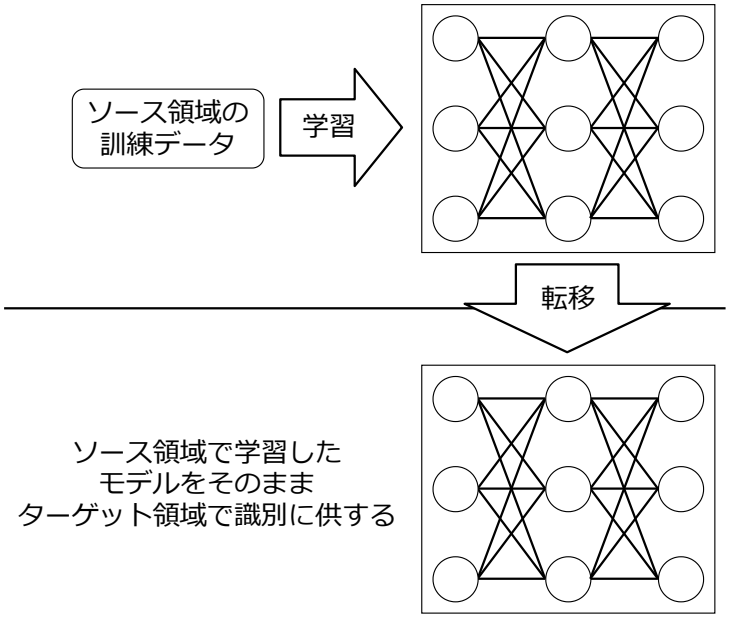

図 2.5: 十分なラベル付きデータが有るソース領域で学習したモデルをそのままターゲット領域で使用する手法．ターゲット領域のラベル付きデータは使わない．

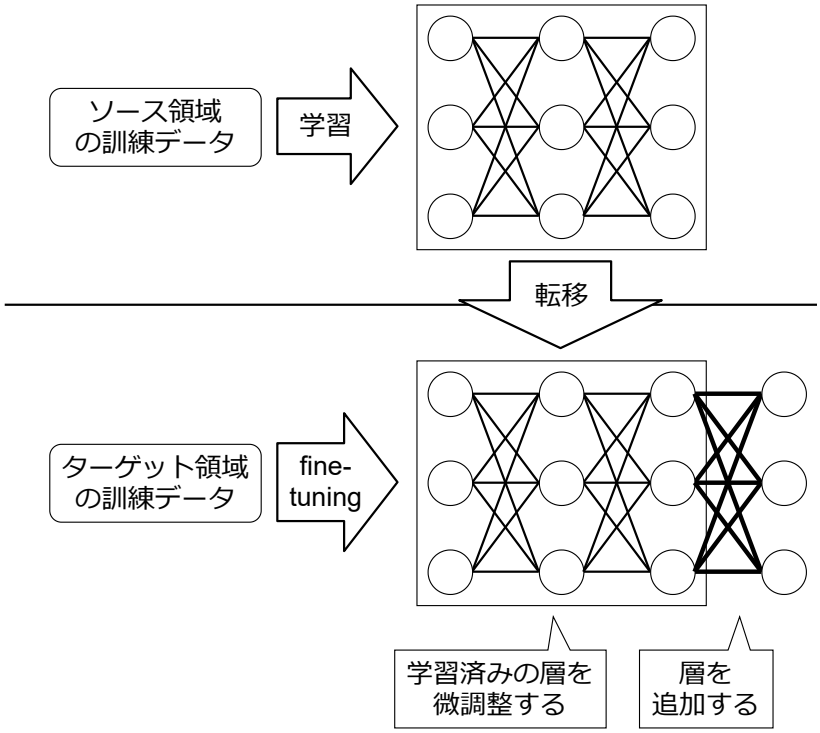

図 2.6: 学習済みのモデルをターゲット領域のラベル付きデータにより fine-tuning する手法．BERT はこれに該当する．

2.7　ドメイン適応型事前学習・タスク適応型事前学習

Gururangan ら [3] は BERT の派生モデルである RoBERTa [7] の領域適応のためにドメイン適応型事前学習 (Domain-Adaptive Pretraining; DAPT) 及びタスク適応型事前学習 (Task-Adaptive Pretraining; TAPT) を提案した．

ドメイン適応型事前学習は，事前学習済みモデルに対して解きたいタスクが属する領域のラベルなしデータを用いた追加の事前学習を行う手法である．例えば Amazon レビューの分類がタスクの場合は，Amazon レビューが領域のラベルなしデータとなる．

タスク適応型事前学習は，事前学習済みモデルに対して解きたいタスクのテキストデータを用いた追加の事前学習を行う手法である．例えば Amazon レビューの分類がタスクの場合は，データセットに含まれるラベルなしデータを用いた追加の事前学習を行う．

十分な計算資源がある場合は，ドメイン適応型事前学習を行ったあとで更にタスク適応型事前学習を行うことで識別精度を改善できる．また，ドメイン適応型事前学習を省略してタスク適応型事前学習のみを行っても識別精度を改善できる．

2.8　負の転移

負の転移 (Negative Transfer) [2] は，ソース領域のデータとターゲット領域のデータの性質が著しく異なるときにソース領域のデータを訓練に用いると識別精度が悪化する現象である．転移学習では，どのようにして負の転移を検知・回避するかが問題となる．

第 3 章

提案手法

タスク適応型事前学習を行えば，BERT を用いた文書分類タスクにおける識別精度の改善は実現できる．しかし，転移学習を用いる場合に負の転移が生じうるという問題は解消されていない．本稿では，BERT を用いた文書分類タスクで転移学習を行うときに Mis-leading データの削除と BERT の追加学習を行うことで識別精度を改善する手法を提案する．Mis-leading データを削除する方法は以下の通りである．

- ターゲット領域のラベルなしデータを離散確率分布 P で表現する
- ソース領域の個々の訓練データを離散確率分布 Q_k で表現する（k 番目の文書に Q_k が対応）
- Kullback-Leibler 情報量 $D_{KL}(P \parallel Q_k)$ 計算し，その値が大きいラベル付きデータを削除する

3.1 BERT を用いた文書分類タスクの転移学習

本稿では，下記の手順により BERT を用いた文書分類タスクの転移学習を行う．

1. BERT をソース領域のラベル付きデータで fine-tuning して文書分類器を作成する．
2. 作成した文書分類器により，ターゲット領域で文書分類を行う．

BERT の fine-tuning では，ターゲット領域のラベル付きデータは使用しない．

3.2　Kullback-Leibler 情報量

Kullback-Leibler 情報量は，情報理論や統計学において 2 つの確率分布の類似度を表す尺度である．ある 2 つの離散確率分布 P, Q に対する Kullback-Leibler 情報量 $D_{KL}(P \parallel Q)$ は，式 (3.1) となる．

$$D_{KL}(P \parallel Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \tag{3.1}$$

ここで $P(i), Q(i)$ はそれぞれ離散確率分布 P, Q に従って i が選ばれる確率である．

3.3　文書の離散確率分布表現

文書を離散確率分布として表現するためには，まず文書をベクトル化しなければならない．文書のベクトル化には Bag-of-Words を用いる．このとき，単語の重み付けには文書内での単語の出現頻度 (Term Frequency) を用いる．文書 Doc のベクトル表現は式 (3.2) のようになる．

$$(tf_1, tf_2, \ldots, tf_{N_w}) \tag{3.2}$$

ここで， tf_i は単語分割時に用いた辞書で i 番目に登録されている単語が文書 Doc 内で登場した回数，N_w は単語分割時に用いた辞書に収録されている単語数である．

式 (3.2) より文書を実数値のベクトルで表現できるようになったが，単語を確率変数とし，文書を離散確率分布として表現するためには，式 (3.2) のベクトルをさらに変換する必要がある．そこで，式 (3.2) のベクトル表現を式 (3.3) のように変換する．

$$(v_1, v_2, \ldots, v_{N_w}) \tag{3.3}$$
$$v_i = \frac{tf_i}{Z}, \quad Z = \sum_{i=1}^{N_w} tf_i$$

確率は次の 2 条件を満たす．

- $v_i \geq 0$ （確率の値は非負の実数）
- $\sum_{i=1}^{N_w} v_i = 1$ （確率の総和は 1）

式 (3.3) のベクトルは， Z の定め方より $\sum_{i=1}^{N_w} v_i = 1$ （確率の総和は 1）を満たす．また， tf_i は明らかに非負の値をとるため， $v_i \geq 0$ （確率の値は非負の実数）を満たす．

したがって，上記の方法により文書を離散確率分布として表現できる．

3.4　Mis-leading データの削除

Mis-leading データは，ソース領域の訓練データのうちターゲット領域での識別精度に悪影響を及ぼすものである．本節では Mis-leading データを削除する方法を説明する．

訓練データとなるラベル付きの文書の集合を $D_S = \{Doc_1, Doc_2, \ldots, Doc_N\}$ とする．D_S には N 件の文書が属し， D_S に属する文書はすべてソース領域に属する．また，ラベルなしの文書のうちターゲット領域に属するものの集合を D_T とする．

Mis-leading データを削除する手順は以下の通りである．

1. D_T に属する文書すべてを 1 つの文書とみなし，3.3 節で説明した手法により離散確率分布 P を得る．
2. D_S に属する文書に 3.3 節で説明した手法を適用し, 離散確率分布 $Q_1, Q_2, \ldots, Q_N$ を得る．k 番目の文書 Doc_k に対応する離散確率分布が Q_k である．
3. 各離散確率分布 $Q_1, Q_2, \ldots, Q_N$ について Kullback-Leibler 情報量 $D_{KL}(P \parallel Q_k)$ を計算する．
4. ラベルに基づいて訓練データをグループ分けする．各グループ内で Kullback-Leibler 情報量 $D_{KL}(P \parallel Q_k)$ の値の大きい順に文書を並べ替え， $D_{KL}(P \parallel Q_k)$ の値の大きい方から一定数の文書を取り出す．取り出した文書，すなわち $D_{KL}(P \parallel Q_k)$ の値の大きい文書を Mis-leading データとみなし，訓練データの集合から削除する．

3.5　BERT の追加学習

BERT を fine-tuning して文書分類器を作成する前に，タスク適応型事前学習を行う．本稿ではデータセットに含まれる全領域のラベルなしデータを用いて追加学習を行い，その後ターゲット領域のラベルなしデータを用いて追加学習を行った（図 3.1 ）．

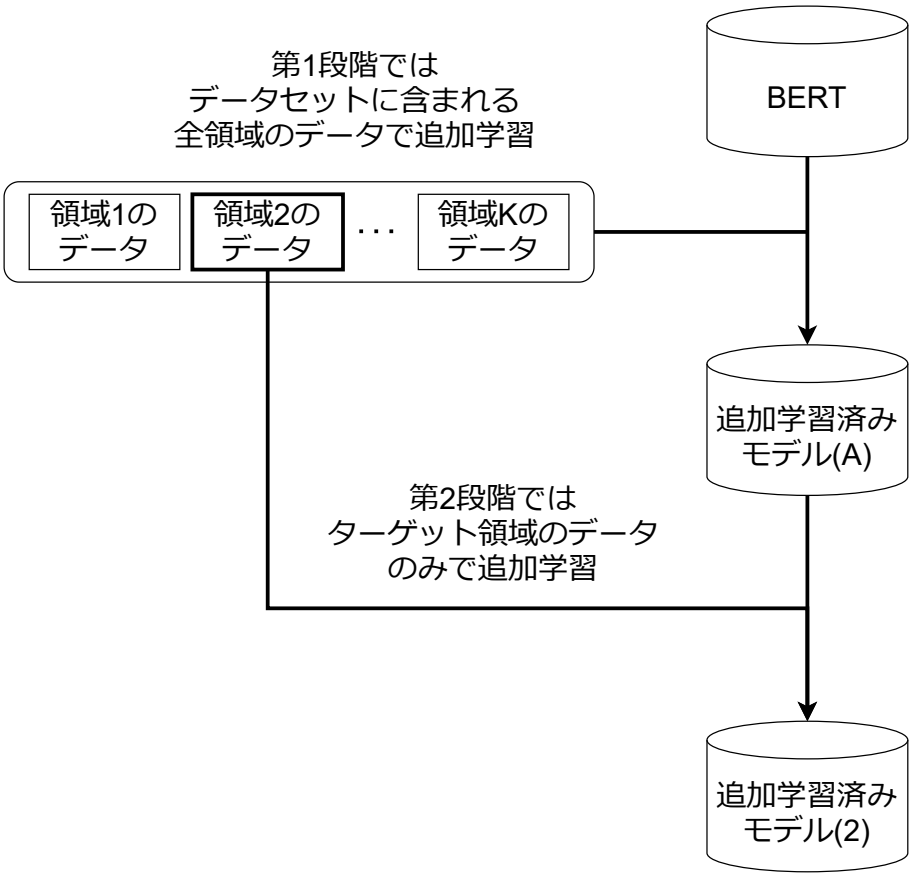

図 3.1: 本図では K 個の領域を持つデータセットを使用し，領域 2 がターゲット領域である場合の追加学習の方法を示している．BERT の追加学習は 2 段階で行う．第 1 段階でデータセットに含まれる全領域のラベルなしデータを用いた追加学習を行い，その後ターゲット領域のデータのみを用いて更に追加学習を行う．「追加学習済みモデル (A) 」はデータセットに含まれる全領域のラベルなしデータを用いた追加学習を済ませたモデル,「追加学習済みモデル (2) 」は「追加学習済みモデル (A) 」に対して領域 2 のデータのみを用いて更に追加学習を行った後のモデルである．

第 4 章

実験

BERT を用いた文書分類タスクで転移学習を行うとき，Mis-leading データの削除と BERT の追加学習を行った場合の識別精度を確認した．Mis-leading データの削除と BERT の追加学習を行わずに転移学習を行った場合（ベースライン）の識別精度と提案手法を用いたときの識別精度を比較し，提案手法の有効性を確認した．

4.1 事前学習済みモデル

東北大学で公開されているモデル[*1] (BERT-base-japanese) を使用した．

4.2 実験用データセット

実験には Webis-CLS-10 データセットを用いた．このデータセットには日本語及び英語の Amazon レビュー文書が収録されている．本実験では日本語の文書を用いる．ラベルは星の数であり，1 から 5 までの 5 段階である．ただしラベルが 3（星 3 つ）のデータは存在しない．本実験ではラベルが 4, 5 のデータを positive, ラベルが 1, 2 のデータを negative として感情分析（2 値分類）を行った．

このデータセットには books, dvd, music の 3 つの領域がある．各領域には訓練データ 2000 件，テストデータ 2000 件が収録されている．この 2 つはラベル付きデータである．また，訓練データ・テストデータとは別にラベルなしデータが収録されている．データセットの内訳を表 4.1 に示す．

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

表 4.1: Amazon レビュー文書のデータセットの内訳

	books	dvd	music
訓練データ	2000	2000	2000
テストデータ	2000	2000	2000
ラベルなしデータ	169780	68326	55892

実験では，ソース領域の訓練データを fine-tuning の訓練データとした．また，ターゲット領域の訓練データを検証用データとし，識別精度の最終的な評価にはターゲット領域のテストデータを用いた．

4.3　Mis-leading データの削除

3.4 節で説明した手法により，ソース領域の訓練データから Mis-leading データを削除した．文書の単語分割には BERT-base-japanese の tokenizer を用いた．削除する件数は 200, 400, 600, 800, 1000 のいずれかとし，検証用データの識別精度をもとに削除する件数を選択した．

4.4　BERT の追加学習

BERT-base-japanese に対してタスク適応型事前学習を行った．まず全領域のラベルなしデータ及び訓練データを用いて 10epoch の追加学習を行った．ここで，ラベルなしデータの件数は領域により異なる．本稿では books と dvd についてはラベルなしデータをランダム抽出し， music のラベルなしデータと同じ件数のラベルなしデータを用意した． music については全てのラベルなしデータを用いた．全領域のラベルなしデータ及び訓練データを用いた追加学習により得られたモデルをモデル A とする．

更に，モデル A に対してターゲット領域のラベルなしデータ及び訓練データを用いて 10epoch の追加学習を行った．どの領域についても，モデル A の追加学習で用いたものと同一のラベルなしデータで追加学習を行った．

なお，追加学習では Masked Language Model のみを行い，Next Sentence Prediction は省略した．

表 4.2: 実験結果（正解率）

ソース	ターゲット	ベースライン	提案手法
books	dvd	0.8610	0.8895
books	music	0.8550	0.8870
dvd	books	0.8420	0.8785
dvd	music	0.8665	0.9005
music	books	0.8425	0.8720
music	dvd	0.8535	0.8735

4.5　文書分類器の作成

モデル A に対してターゲット領域のラベルなしデータ及び訓練データを用いて追加学習を行ったモデルの直後に全結合層を 1 層追加したものを fine-tuning し，文書分類器を作成した．fine-tuning にはソース領域の訓練データのうち，3.4 節で説明した手法により Mis-leading データを削除したものを用いた．

4.6　実験結果

提案手法を用いて転移学習を行った場合の識別精度及び提案手法を用いずに転移学習を行った場合（ベースライン）の識別精度を表 4.2 に示す．ベースラインの数値は，追加学習を行っていない BERT-base-japanese をソース領域の訓練データ全てで fine-tuning して作成した文書分類器をターゲット領域で用いた場合の正解率である．ソース領域とターゲット領域の組み合わせは計 6 パターンあるが，全パターンについて文書分類タスクを行い，識別精度を評価した．

表 4.2 の通り，ソース領域とターゲット領域の組み合わせの全パターンで提案手法での正解率がベースラインの正解率を上回った．

第 5 章

考察

以下，ソース領域とターゲット領域の組を（ソース領域，ターゲット領域）と表記することがある．

ソース領域とターゲット領域の各組み合わせについて，Mis-leading データを削除するために計算した Kullback-Leibler 情報量 $D_{KL}(P \parallel Q_k)$ のばらつきを図 5.1 の箱ひげ図に表した．

ベースラインの正解率と提案手法の正解率を比較したとき，正解率の上がり幅が最も大きかったソース領域とターゲット領域の組は (dvd, books) である．また，正解率の上がり幅が最も小さかった組は (music, dvd) である．

図 5.1 の箱ひげ図より，(dvd, books) では他の組よりも Kullback-Leibler 情報量の散らばりが大きく，Kullback-Leibler 情報量の中央値は小さいとわかる．一方で，(music, dvd) では他の組よりも Kullback-Leibler 情報量の散らばりが小さく，Kullback-Leibler 情報量の中央値は比較的大きいとわかる．このような Kullback-Leibler 情報量 のばらつきが，Mis-leading データを削除する効果の大きさに影響していると考えられる．

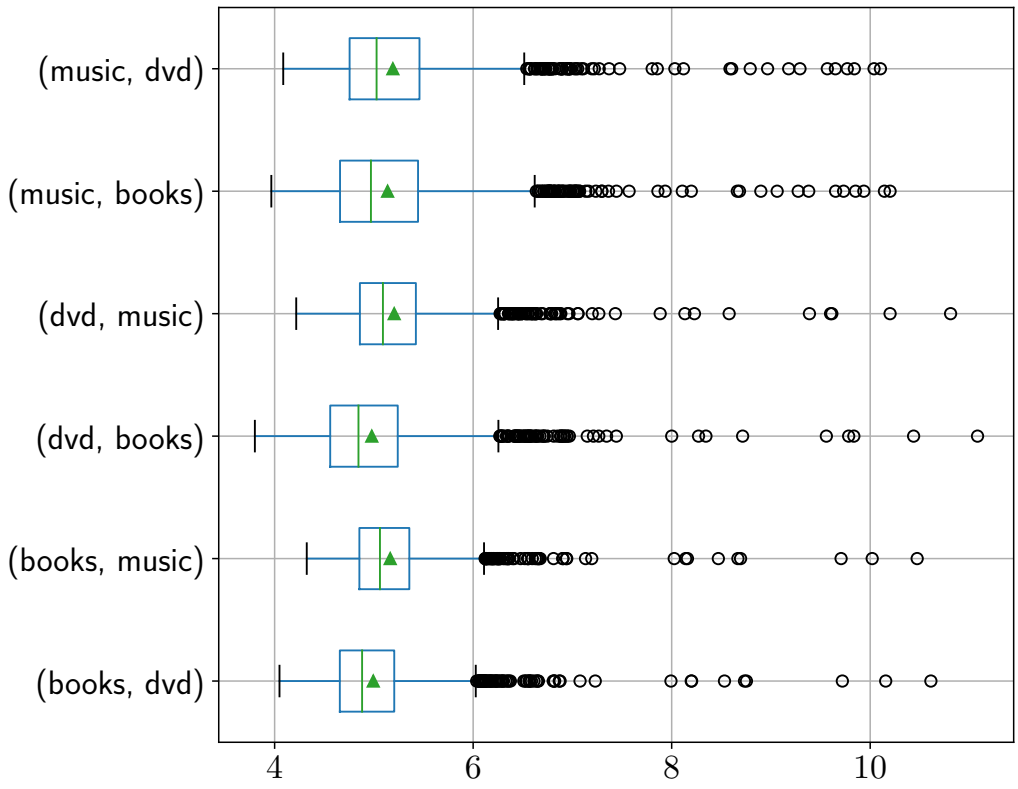

図 5.1: Kullback-Leibler 情報量 $D_{KL}(P \parallel Q_k)$ のばらつき．箱の左のラベルはソース領域とターゲット領域の組（ソース，ターゲット）．

第 6 章

結論

本稿では, BERT を用いた文書分類タスクで転移学習を行うときに Mis-leading データの削除と BERT の追加学習を行い，識別精度の改善を試みた．実験により，Mis-leading データの削除と BERT の追加学習を行わない転移学習と比較したときに識別精度が改善されることを確認できた.

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いたします．また，日常の議論を通して多くの知識，示唆を頂いた新納研究室の皆様に感謝します．

参考文献

[1] 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] Michael T Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G Dietterich. To transfer or not to transfer. In *NIPS 2005 workshop on transfer learning*, Vol. 898, 2005.

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

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

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

[6] Michael A. Hedderich, Lukas Lange, Heike Adel, Jannik Strötgen, and Dietrich Klakow. A survey on recent approaches for natural language processing in low-resource scenarios. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language*

Technologies, pp. 2545–2568, Online, June 2021. Association for Computational Linguistics.

[7] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, Vol. abs/1907.11692, , 2019.

付録

A　文書分類器のプログラム

実験で作成した文書分類器の基となるプログラムを A.1 に，文書分類器を作成するプログラムを A.2 に，文書分類器の精度を評価するプログラムを A.3 に示す.

ソースコード A.1: 実験で作成した文書分類器の基となるプログラム

```
1  from transformers import BertModel, BertConfig
2  import pickle
3  import torch
4  import torch.nn as nn
5  import torch.optim as optim
6  import torch.nn.functional as F
7  import numpy as np
8  import pathlib
9  import argparse
10 import time # 実行時間計測
11 import random
12
13 DATADIR = pathlib.Path('/home/iwamoto/cls-acl10/cls-acl10-mycode001')
14 SEED = 42
15
16 class MyModel(nn.Module):
17     def __init__(self, output_dim, model_src='cl-tohoku/bert-base-
           japanese'):
18         super().__init__()
19         config = BertConfig.from_pretrained(
20             model_src
21         )
22         # BERT
23         self.bertmodel = BertModel.from_pretrained(
```

```
24              model_src, config=config
25          )
26
27          # BERT の出力をもとにクラス分類を行う NN
28          self.fc1 = nn.Linear(768, output_dim)
29
30          # 重みの初期化
31          for v in [self.fc1]:
32              nn.init.normal_(v.weight, std=0.02)
33              nn.init.normal_(v.bias, 0)
34
35      def forward(self, token, attention):
36          bertoutput = self.bertmodel(
37              input_ids=token,
38              attention_mask=attention
39          )
40          hidden = [v[0] for v in bertoutput[0]]
41          hidden = torch.stack(hidden, 0)
42          return self.fc1(hidden)
43
44  class MyDataset(torch.utils.data.Dataset):
45      def __init__(self, tokens, attentions, labels, transform=None):
46          self.transform = transform
47          self.token = []
48          self.attention = []
49          self.label = []
50          for v in tokens:
51              self.token.append(torch.tensor(v))
52          for v in attentions:
53              self.attention.append(torch.tensor(v))
54          for v in labels:
55              # [0, 4]の 5段階になっているラベルを [0, 1] (negative or
                    positive) に直す
56              # 星 2つ以下はネガティブ　星 3つ以上はポジティブ
57              self.label.append(torch.tensor(0 if v < 2 else 1))
58          self.data_num = len(self.label)
59
60      def __len__(self):
61          return self.data_num
62
63      def __getitem__(self, idx):
```

```
64          out_token = self.token[idx]
65          out_attention = self.attention[idx]
66          out_label = self.label[idx]
67
68          if self.transform:
69              out_token = self.transform(out_token)
70
71          return out_token, out_attention, out_label
72
73  def train(dataloader, model, criterion, optimizer, device):
74      model.to(device)
75      model.train()
76      loss_total = 0.0
77      for text, attention, label in dataloader:
78          text = text.to(device)
79          attention = attention.to(device)
80          label = label.to(device)
81          output = model(text, attention)
82          loss = criterion(output, label)
83
84          optimizer.zero_grad()
85          loss.backward()
86          torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
87          optimizer.step()
88          loss = loss.item()
89          loss_total += loss
90
91      print("loss_total: {}".format(loss_total))
92      print("loss_ave: {}".format(loss_total / len(dataloader)))
93
94  def test(dataloader, model, device):
95      model.to(device)
96      model.eval()
97      correct = 0
98      with torch.no_grad():
99          for text, attention, label in dataloader:
100             text = text.to(device)
101             attention = attention.to(device)
102             label = label.to(device)
103             output = model(text, attention)
104             correct += (output.argmax(1) == label).long().sum().item
```

```
                ()
105     return correct
106
107
108 if __name__ == "__main__":
109     parser = argparse.ArgumentParser(description='テキスト分類プログラム')
110     parser.add_argument('--srcmodel', type=str,
111     default='cl-tohoku/bert-base-japanese', help='使用する事前学習済みモデル')
112     parser.add_argument('--batch_size', type=int, default=5, help='バッチサイズ')
113     parser.add_argument('--epochs', type=int, default=5, help='fine-tuning の epoch 数')
114     parser.add_argument('--train_cat', type=str, default='books', help='訓練データのカテゴリ')
115     parser.add_argument('--test_cat', type=str, default='books', help='テストデータのカテゴリ')
116     parser.add_argument('--data_dir', type=str, help='ラベル付きの訓練データ・テストデータがあるディレクトリを指定する')
117
118     args = parser.parse_args()
119
120     device = "cuda" if torch.cuda.is_available() else "cpu"
121     random.seed(SEED)
122     torch.manual_seed(SEED)
123     print("Using {} device".format(device))
124     if args.data_dir:
125         DATADIR = pathlib.Path(args.data_dir)
126
127     train_catname = args.train_cat
128     token_train_name = DATADIR / 'token_train_{}_jp.pkl'.format(
            train_catname)
129     attention_train_name = DATADIR / 'attention_train_{}_jp.pkl'.
            format(train_catname)
130     label_train_name = DATADIR / 'label_train_{}_jp.pkl'.format(
            train_catname)
131
132     with open(token_train_name, mode='rb') as tokenf, \
133         open(attention_train_name, mode='rb') as attenf, \
134         open(label_train_name, mode='rb') as labelf:
```

```
135         token_train = pickle.load(tokenf)
136         attention_train = pickle.load(attenf)
137         label_train = pickle.load(labelf)
138
139     data_set = MyDataset(token_train, attention_train, label_train)
140     # ミニバッチに固める
141     num_data_set = len(data_set)
142     batch_size = args.batch_size
143     dataloader = torch.utils.data.DataLoader(data_set, batch_size=
            batch_size, shuffle=True)
144
145     model = MyModel(2, args.srcmodel)
146     model.to(device)
147     criterion = nn.CrossEntropyLoss()
148     # ファインチューニングも行う
149     optimizer = optim.AdamW([
150         {'params': model.parameters(), 'lr': 2e-5},
151     ])
152     epochs = args.epochs
153     start = time.time() # 訓練時間計測
154     for i in range(epochs):
155         print("epoch: {} / {} -----".format(i+1, epochs))
156         train(dataloader, model, criterion, optimizer, device)
157     print("train time: {:.5f}".format(time.time()-start)) # 訓練時間
            計測
158
159     test_catname = args.test_cat
160     token_test_name = DATADIR / 'token_test_{}_jp.pkl'.format(
            test_catname)
161     attention_test_name = DATADIR / 'attention_test_{}_jp.pkl'.format(
            test_catname)
162     label_test_name = DATADIR / 'label_test_{}_jp.pkl'.format(
            test_catname)
163
164     with open(token_test_name, mode="rb") as tokenf, \
165         open(attention_test_name, mode="rb") as attenf, \
166         open(label_test_name, mode="rb") as labelf:
167         token_test = pickle.load(tokenf)
168         attention_test = pickle.load(attenf)
169         label_test = pickle.load(labelf)
170     testdata_set = MyDataset(token_test, attention_test, label_test)
```

```
171     testdataloader = torch.utils.data.DataLoader(testdata_set,
            batch_size=1)
172
173     print("srcmodel: {}, train_cat: {}, test_cat: {}".format(args.
            srcmodel, args.train_cat, args.test_cat))
174     res = test(testdataloader, model, device)
175     print("result: {:.5f} ({} / {})".format(res/len(testdata_set),
            res, len(testdata_set)))
```

ソースコード A.2: 文書分類器を作成するプログラム

```
1  import textClassify003
2  from textClassify003 import MyDataset, MyModel, train
3  import pickle
4  import torch
5  import torch.nn as nn
6  import torch.optim as optim
7  import random
8  import pathlib
9  import argparse
10 import time
11
12 if __name__ == "__main__":
13     parser = argparse.ArgumentParser(description='テキスト分類プログラ
           ム')
14     parser.add_argument('--srcmodel', type=str,
15     default='cl-tohoku/bert-base-japanese', help='使用する事前学習済み
           モデル')
16     parser.add_argument('--batch_size', type=int, default=5, help='バ
           ッチサイズ')
17     parser.add_argument('--learning_rate_bert', type=float, default=2e
           -5, help='BERT の学習率')
18     parser.add_argument('--learning_rate_fc', type=float, default=2e
           -5, help='全結合層の学習率')
19     parser.add_argument('--epochs', type=int, default=5, help='fine-
           tuning の epoch 数')
20     parser.add_argument('--train_cat', type=str, default='books', help
           ='訓練データのカテゴリ')
21     parser.add_argument('--output_dir', type=str, required=True, help
           ='モデルの出力先ディレクトリ')
22     parser.add_argument('--save_checkpoint', action='store_true', help
```

```
        ='このオプションをつけると 1epoch ごとにモデルを保存する')
23      parser.add_argument('--data_dir', type=str, help='ラベル付きの訓練
            データ・テストデータがあるディレクトリを指定する')
24      parser.add_argument('--data_id', type=str, help='ラベル付きの訓練
            データのID を指定する')
25      parser.add_argument('--output_param', action='store_true', help='
            このオプションをつけると各種パラメータを出力')
26
27      args = parser.parse_args()
28      srcmodel = args.srcmodel
29      train_catname = args.train_cat
30      is_save_checkpoint = args.save_checkpoint
31      is_output_param = args.output_param
32      batch_size = args.batch_size
33      learning_rate_bert = args.learning_rate_bert
34      learning_rate_fc = args.learning_rate_fc
35      epochs = args.epochs
36
37      output_dir = pathlib.Path(args.output_dir)
38      if not output_dir.exists():
39          output_dir.mkdir()
40
41      device = "cuda" if torch.cuda.is_available() else "cpu"
42      SEED = textClassify003.SEED
43      DATADIR = textClassify003.DATADIR
44      if args.data_dir:
45          DATADIR = pathlib.Path(args.data_dir)
46      random.seed(SEED)
47      torch.manual_seed(SEED)
48      if is_output_param:
49          print("Using {} device".format(device))
50          print("srcmodel: {}".format(srcmodel))
51          print("train_category: {}".format(train_catname))
52          print("output_dir: {}".format(output_dir.absolute()))
53          print("batch_size: {}".format(batch_size))
54          print("learning_rate_bert: {}".format(learning_rate_bert))
55          print("learning_rate_fc: {}".format(learning_rate_fc))
56          print("epochs: {}".format(epochs))
57
58      token_train_name = DATADIR / 'token_train_{}_jp.pkl'.format(
            train_catname)
```

```
59      attention_train_name = DATADIR / 'attention_train_{}_jp.pkl'.
            format(train_catname)
60      label_train_name = DATADIR / 'label_train_{}_jp.pkl'.format(
            train_catname)
61      if args.data_id is not None:
62          token_train_name = DATADIR / args.data_id / 'token_train_{}_{}
                _jp.pkl'.format(train_catname, args.data_id)
63          attention_train_name = DATADIR / args.data_id / '
                attention_train_{}_{}_jp.pkl'.format(train_catname, args.
                data_id)
64          label_train_name = DATADIR / args.data_id / 'label_train_{}_{}
                _jp.pkl'.format(train_catname, args.data_id)
65      # print(xtrain_name)
66      with open(token_train_name, mode='rb') as tokenf, \
67          open(attention_train_name, mode='rb') as attenf, \
68          open(label_train_name, mode='rb') as labelf:
69          token_train = pickle.load(tokenf)
70          attention_train = pickle.load(attenf)
71          label_train = pickle.load(labelf)
72
73      data_set = MyDataset(token_train, attention_train, label_train)
74      # ミニバッチに固める
75      num_data_set = len(data_set)
76      dataloader = torch.utils.data.DataLoader(data_set, batch_size=
            batch_size, shuffle=True)
77
78      model = MyModel(2, srcmodel)
79      model.to(device)
80      criterion = nn.CrossEntropyLoss()
81      # ファインチューニングも行う
82      optimizer = optim.AdamW([
83          {'params': model.bertmodel.parameters(), 'lr': 2e-5},
84          {'params': model.fc1.parameters(), 'lr': 2e-5},
85      ])
86      start = time.time() # 訓練時間計測
87      for epoch in range(epochs):
88          print("epoch: {} / {} -----".format(epoch+1, epochs))
89          train(dataloader, model, criterion, optimizer, device)
90          if is_save_checkpoint:
91              torch.save(
92              {
```

```
93                  "epoch": epoch+1, # 学習再開時の開始番号　兼　1-
                        indexed で数えたときの epoch 数
94                  "model_state": model.to('cpu').state_dict(),
95                  "optimizer_state": optimizer.state_dict(),
96              },
97              output_dir / '{}.model'.format(epoch+1))
98      print("train time: {:.5f}".format(time.time()-start)) # 訓練時間
            計測
99
100     if not is_save_checkpoint:
101         torch.save(
102         {
103             "epoch": epochs, # 学習再開時の開始番号　兼　1-
                    indexed で数えたときの epoch 数
104             "model_state": model.to('cpu').state_dict(),
105             "optimizer_state": optimizer.state_dict(),
106         },
107         output_dir / '{}.model'.format(epochs))
```

ソースコード A.3: 文書分類器の精度を評価するプログラム

```
1  import textClassify003
2  from textClassify003 import MyDataset, MyModel, test
3  import pickle
4  import torch
5  import torch.nn as nn
6  import torch.optim as optim
7  import random
8  import pathlib
9  import argparse
10 import time
11
12 if __name__ == "__main__":
13     parser = argparse.ArgumentParser(description='テキスト分類プログラ
           ム')
14     parser.add_argument('--srcmodel', type=str,
15     default='cl-tohoku/bert-base-japanese', help='使用する事前学習済み
           モデル')
16     parser.add_argument('--batch_size', type=int, default=16, help='
           バッチサイズ')
17     parser.add_argument('--test_cat', type=str, default='books', help
```

```
        ='テストデータのカテゴリ')
18      parser.add_argument('--model_path', type=str, required=True, help
        ='評価するモデルのパス')
19      parser.add_argument('--output_rate_only', action='store_true',
        help='このオプションをつけると正解率のみ出力')
20      parser.add_argument('--validation', action='store_true', help='こ
        のオプションをつけるとバリデーションデータでテスト')
21      parser.add_argument('--data_dir', type=str, help='ラベル付きの訓練
        データ・テストデータがあるディレクトリを指定する')
22
23      args = parser.parse_args()
24      model_path = pathlib.Path(args.model_path)
25      is_output_rate_only = args.output_rate_only
26      batch_size = args.batch_size
27
28      device = "cuda" if torch.cuda.is_available() else "cpu"
29      SEED = textClassify003.SEED
30      DATADIR = textClassify003.DATADIR
31      if args.data_dir:
32          DATADIR = pathlib.Path(args.data_dir)
33      random.seed(SEED)
34      torch.manual_seed(SEED)
35      if not is_output_rate_only:
36          print("Using {} device".format(device))
37
38      test_catname = args.test_cat
39      phase = "test"
40      if args.validation:
41          phase = "train"
42      token_test_name = DATADIR / 'token_{}_{}_jp.pkl'.format(phase,
        test_catname)
43      attention_test_name = DATADIR / 'attention_{}_{}_jp.pkl'.format(
        phase, test_catname)
44      label_test_name = DATADIR / 'label_{}_{}_jp.pkl'.format(phase,
        test_catname)
45
46      with open(token_test_name, mode='rb') as tokenf, \
47          open(attention_test_name, mode='rb') as attenf, \
48          open(label_test_name, mode='rb') as labelf:
49          token_test = pickle.load(tokenf)
50          attention_test = pickle.load(attenf)
```

```
51          label_test = pickle.load(labelf)
52      testdata_set = MyDataset(token_test, attention_test, label_test)
53      testdataloader = torch.utils.data.DataLoader(testdata_set,
            batch_size=batch_size,
54      num_workers=0, drop_last=False)
55
56      model = MyModel(2, args.srcmodel)
57      model.load_state_dict(torch.load(model_path)["model_state"])
58      res = test(testdataloader, model, device)
59      if not is_output_rate_only:
60          print("model_path: {}".format(model_path))
61          print("test_category: {}".format(test_catname))
62          print("result: {:.10f} ({} / {})".format(res/len(testdata_set
                ), res, len(testdata_set)))
63      else:
64          print("{:.10f}".format(res/len(testdata_set)))
```