令和 4 年度茨城大学工学部情報工学科

卒業研究論文

Zero-Shot Cross-Lingual Transfer における言語の多様性

所属 情報工学科

著者 佐藤匠真 (19T4035N)

指導教員 新納浩幸教授

令和 5 年 2 月 3 日 (金)

令和 4 年度茨城大学工学部情報工学科　卒業研究論文

Zero-Shot Cross-Lingual Transfer における言語の多様性

著者

佐藤匠真 (19T4035N)

指導教員

新納浩幸教授

論文要旨

multilingual BERT のような多言語の事前学習済みの自然言語処理モデルでは，あるタスクに対して言語 A で学習したモデルが別の言語 X の同じタスクに対してそのまま利用できる，いわゆる Zero-Shot Cross-Lingual Transfer が可能である．Zero-Shot Cross-Lingual Transfer は通常言語 A を英語などのメジャーな言語，言語 X をリソースが少ない，あるいはないようなマイナーな言語に設定することで低資源の学習の問題解決に利用される．zero-shot cross-lingual transfer を行う場合，もとのモデルは単言語のモデルであっても可能であることが示されており，多言語の言語モデルにおける言語の多様性が Zero-Shot Cross-Lingual Transfer に影響を与えているかどうかはっきりしていない．

本論文では，上記の言語 A に当たる部分を複数の言語を組み合わせた場合と 1 つの言語の場合を比較する．複数の言語を組み合わせた場合の結果が良い場合は言語の多様性が Zero-Shot Cross-Lingual Transfer に良い影響を与えていると考えられる．

実験では，多言語の事前学習済みモデルである multilingual BERT を利用した．そして，Amazon レビューの評判分析をタスクとし，モデルを学習する言語を英語，ドイツ語，フランス語の各組合わせとした．そして，テストデータは日本語の Amazon レビュー 2000 文として Zero-Shot Cross-Lingual Transfer を行った．実験の結果，訓練データに言語の多様性を含める効果があることが判明した．

また，Zero-Shot Cross-Lingual Transfer ではない場合の訓練データに言語の多様性を含める効果については今後の課題とする．

目次

第 1 章 序論 7

第 2 章 関連研究 9

2.1 ニューラルネットワーク 9

2.2 Word2Vec 10

2.3 Zero-Shot Learning 13

2.4 BERT 13

2.5 Multilingual BERT 18

2.6 Zero-Shot Cross-Lingual Transfer 19

第 3 章 訓練データの言語の多様性 21

3.1 言語の多様性の効果の確認 21

3.2 言語の多様性を含む訓練データの作成 21

第 4 章 実験 24

4.1 事前学習済みモデル 24

4.2 実験用データセット 24

4.3 文書分類器の作成 25

4.4 実験結果 25

第 5 章 考察 27

第 6 章 結論 29

参考文献 31

付録 33

A 文書分類器のプログラム 33

表目次

4.1 単言語での学習 26
4.2 言語比 2:1 での学習 26
4.3 言語比 1:1:1 での学習 26

5.1 単言語の訓練データ 1000 文. 27
5.2 日本語 1000 文で fine-tuning 28

図目次

2.1 分類問題の例. 10
2.2 CBOW の具体例. 11
2.3 Skip-Gram の具体例. 12
2.4 BERT の内部構造. 14
2.5 Scaled Dot-Product Attention の処理の流れ. 15
2.6 転移学習の流れ. 18
2.7 Zero-Shot Cross-Lingual Transfer の例. 20

3.1 1100 文の訓練データ 22
3.2 言語の多様性を持たせた 3300 文の訓練データ. 22

4.1 実験結果の平均値のグラフ. 25

第 1 章

序論

近年，multilingual BERT(mBERT) [1]，XLM [2]，XLM-RoBERTa-XL [3] など多言語間の汎用的な特徴表現を学習する多言語の事前学習済みモデルが提案されている．このようなモデルの応用の 1 つとして zero-shot cross-lingual transfer がある．

機械学習を用いて何らかのタスクを解く場合，一般的に訓練データの数が多いほどタスクを高い精度で解くことができる．自然言語処理の分野では，解きたい言語の解きたいタスクの訓練データとテストデータが必要である．英語やドイツ語などのメジャーで多くの人が話している言語ならば多くのデータがある．しかし，ヒンディー語やスワヒリ語などの話者が少ない言語の場合，データも少ない．この問題解決に Zero-Shot Cross-Lingual Transfer が有効である．

Zero-Shot Cross-Lingual Transfer は，あるタスクに対して言語 A で fine-tunining したモデルを別の言語 X の同じタスクに対してそのまま利用する手法である [4]．言語 A を英語などのメジャーな言語，言語 X をリソースが少ない，あるいはないようなマイナーな言語に設定することで，低資源言語の学習の問題解決に利用される．

一方，zero-shot cross-lingual transfer を行う場合，もとのモデルは単言語のモデルであっても可能であることが示されており [5]，多言語の言語モデルにおける言語の多様性が zero-shot cross-lingual transfer に影響を与えているのかどうかがはっきりしていない．本研究では評判分析をタスクに設定して，この点に対する調査する．

調査の方法としては言語 A から言語 X への zero-shot cross-lingual transfer を行う際に，言語 A の訓練データの他に，言語 B や言語 C の訓練データも利用することで言語に多様性のある訓練データから学習する場合と，それと同サイズの言語 A (あるいは B や C) の言語に多様性のない訓練データから学習する場合とどちらが zero-shot

cross-lingual transfer において効果があるのかを調べる．

実験ではタスクを Amazon レビューの評判分析とし，上記言語 A，B，C を英語，ドイツ語，フランス語，言語 X を日本語とした．つまりテストデータは日本語の Amazon レビュー文書である．実験の結果，訓練データに言語の多様性を含める効果があることが判明した．

第 2 章

関連研究

2.1　ニューラルネットワーク

ニューラルネットワークは，なんらかの変換を行う層の組み合わせによって構成される．各層は一般に，入力値に対して適当な線形変換と，活性化関数の合成変換が施された出力値を返す．活性化関数は，Sigmoid 関数や，ReLU 関数など様々な種類がある．ニューラルネットワークは，複数の層を組み合わせることによって高い表現能力を示す．

各層で行われる変換は，その層が持つパラメータに依存する．これらのパラメータが学習の過程で調整されることで，ニューラルネットワークは適切に問題を解くことができるようになる．パラメータを調整する際に，必要となる現在のパラメータの「良さ」を表す基準が必要である．この「良さ」は

- そのパラメータを持つニューラルネットワークが表現する入出力関係
- タスクで獲得したい理想的な入出力関係

を比較することで確認でき，ズレが少なければ少ないほど良いパラメータである．このズレを定量的に表現する関数を損失関数という．

2.1.1　分類問題

自然言語処理の多くのタスクは，与えられたデータに対して，与えられた複数のカテゴリーの中から，そのデータに対応するカテゴリーを決める分類問題として扱うことができる．

カテゴリーの数を N, データを表現するベクトルを x とし，カテゴリーを 1 から N の整数で表現したものをラベルといい，l とする．分類問題とはデータ x が与えられたときに，そのラベル l を予測するモデルを作ることである．

分類問題はニューラルネットワークを用いて解くことができる．ニューラルネットワークの出力を

$$y = F(x, \theta)$$

と置く．θ は調整可能なパラメータである．出力 y はカテゴリーの数と同じ次元である．出力 y の各要素の値は，それぞれカテゴリーへの予測の確度を表すものである．それを分類スコアと呼ぶ．y の j 番目の要素の値は j 番目のカテゴリーに対する分類スコアを表す．この分類スコアが最も高いカテゴリーをニューラルネットワークの予測とする．図 2.1 を例にするとラベル 2 の分類スコアが最も高いため，ニューラルネットワークはラベル l が 2 と予測する．

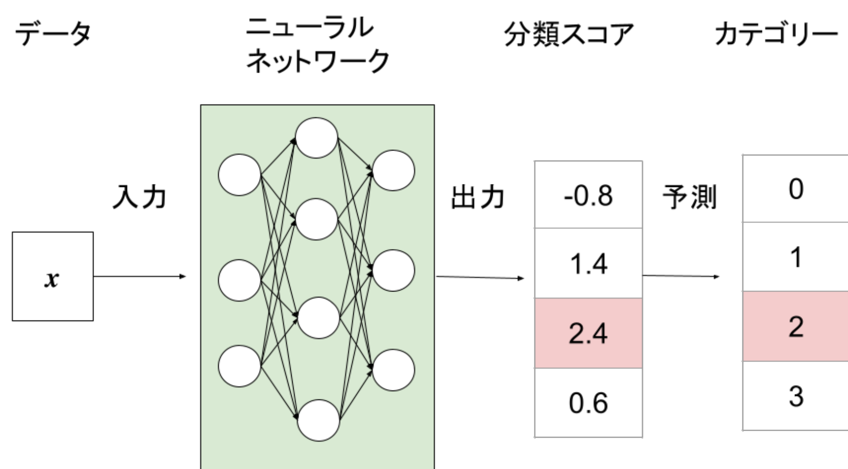

図 2.1: 分類問題の例.

2.2　Word2Vec

Word2Vec はトークンに対して文脈に依存せず一意な表現を与えるモデルである．トークンに対して文脈に応じた表現を与えるモデルとしては「ELMo」がある．Word2vec は単語に対して文脈非依存の分散表現を学習するモデルである．分散表現とは，単語が持つ意味に注目して単語をベクトル化する手法である．単語の意味をベクトル化すること

で，同じような意味や使われ方をする単語に近しいベクトルを与えることができる．単語に文脈非依存の分散表現を与えることを単語埋め込みと呼ぶ.

Word2Vec により得られる分散表現は，単語間の意味的類似性を表すだけではなく，分散表現の加算，減算ができる．例えば，以下のようになる．

$$y(\text{日本}) - v(\text{東京}) \fallingdotseq v(\text{フランス}) - v(\text{パリ})$$

$$v(\text{日本}) + v(\text{首都}) \fallingdotseq v(\text{東京})$$

これは，「日本」と「首都」分散表現の加算が日本の首都の「東京」の分散表現とほとんど同じであることを示す．Word2Vec では，CBOW,skip-gram と呼ばれる 2 つのモデルが提案されている．

2.2.1　CBOW

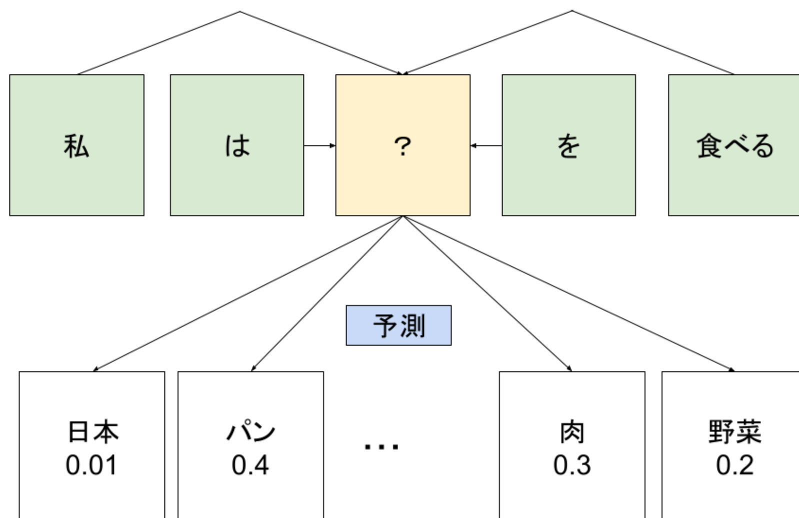

図 2.2: CBOW の具体例.

Continous Bad-Of-Words(CBOW) モデルは文章 S=(w_1,w_2,...,w_n) が与えられたとき，その i 番目に位置する単語 w_i をその周りの単語から予測する言語モデルである．CBOW は

- 文脈の単語をベクトルに変換する埋め込み層
- 予測する単語をベクトルに変換する埋め込み層

から構成される．例えば，図 2.2 の場合，「私」，「は」，「を」，「食べる」という 4 つの単語から 3 番目に入る単語を予測する．確率が高い単語がより適していると予測される．

巨大な語彙を扱う学習の場合，計算量が多くなってしまうが，Hierarchical Softmax と Noise Contrastive Estimation(NCE) のどちらかの手法を用いて大規模な語彙の学習を可能にしている．

2.2.2 Skip-Gram

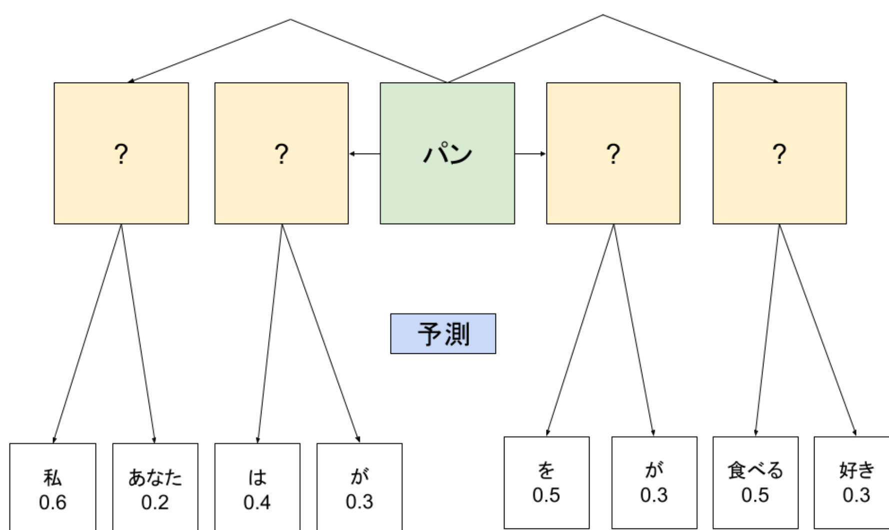

図 2.3: Skip-Gram の具体例.

Skip-Gram は CBOW とは逆のような，文中の単語が与えられたとき，その文脈に出現する単語の確率分布をモデル化する．図 2.3 のように 1 つの単語から他の単語を予測する．

2.3　Zero-Shot Learning

Zero-Shot Learning は機械が見たことないもの，知らないものを予測するための機械学習の技術の一つである．通常の分類タスクの場合，例えば,「縞模様」と「馬」の画像を学習して未学習の「シマウマ」が画像が入力された場合，縞模様か馬のクラスに分類される．しかし，Zero-Shot Learning の場合，未知クラスのデータ「シマウマ」が入力された場合でも未知の正しい「シマウマ」クラスに分類できる．これは,「シマウマ」の画像から「馬の特徴」と「縞柄の特徴」をエンコーダが獲得し，Word2Vec などの事前学習済みモデルで「シマウマ」が縞柄の馬であるという事前知識を取り込むことが出来ているからである．

Zero-Shot Learning の評価には主に

- 未知クラスに対する認識性能を測る Zero-Shot Learning
- 既知クラスと未知クラスに対する認識性能を測る Generalized Zero-shot Learning

がある．

2.4　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．様々な言語タスクで当時の最先端のモデルの性能を大きく上回る性能を示した．BERT は文脈を考慮した分散表現を生成するモデルである．BERT 以前から文脈を考慮するモデルは存在していたが，BERT では Attention という手法で離れた位置の情報も取り入れることができる．BERT は Transformer というモデル [6] で提案された Transformer Encoder と呼ばれる Attention を用いたニューラルネットワークを用いている．Transformer Encoder のそれぞれの層はおもに，Multi-Head Attention と Feed Forward Network から構成される．

BERT の内部構造は図 2.4 のように，入力データがベクトル化され，層に渡されその出力が次の層に渡される．そして，最後の層の出力が BERT としての出力となる．

また，BERT には事前学習とファインチューニングの二つの学習の過程がある．

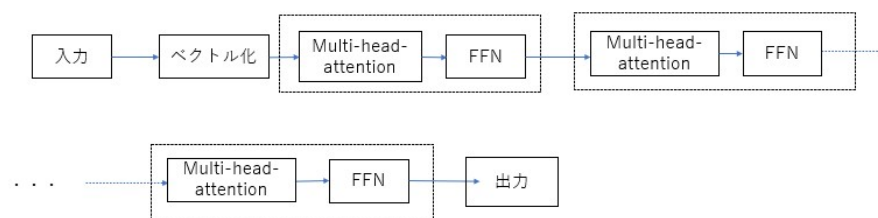

図 2.4: BERT の内部構造.

2.4.1 Scaled Dot-Product Attention

Scaled Dot-Product Attention は Multi-Head Attention の構成要素である.

例として，n 個のトークンで構成される文章を処理するとする．一つ目の層での i 番目のトークンに対応する出力をベクトル x_i で与えられるとする．それぞれの出力に対して行列 W^q,W^k,W^v で線形変換を行うことにより，クエリ q_i, キー k_i, バリュー v_i と呼ばれる三つの d 次元ベクトルを準備する.

$$q_i = x_i W^q$$

$$k_i = x_i W^k$$

$$v_i = x_i W^v$$

Scaled Dot-Product Attention ではそれぞれのトークンはこれらの三つのベクトルにより特徴づけられる．そして，それぞれのトークンに対してベクトル a_i を出力する．a_i はバリュー v_i の重み付き平均

$$a_i = \sum_{j=1}^{n} a_{n,j} v_j$$

で与えられるとする．重み $a_{i,j}$ は i 番目のトークンを処理する際に，j 番目のトークンの情報を重視する度合いを表している．重みはキーとクエリから決まる．i 番目のトークンを処理する時には，そのクエリと関連度が大きいようなキーを持つトークンほど出力に大きく寄与するようになる．Transformer では，Scaled Dot-Product でクエリとキーを評価する.

Scaled Dot-Product は q_i と k_j の内積を $\sqrt{d}$ で割って得られる $\hat{a}_{i,j}$ をスコアとして用いる.

得られたスコア $\hat{a}_{i,1}$,$\hat{a}_{i,2}$,...,$\hat{a}_{i,n}$ に Softmax 関数を適用することで、最終的に重み $a_{i,1}$,$a_{i,2}$,...,$a_{i,n}$ を得る．

$$[a_{i,1}, a_{i,2}, ..., a_{i,n}] = Softmax(\hat{a}_{i,1}, \hat{a}_{i,2}, ..., \hat{a}_{i,n})$$

また，異なるトークンに対する出力は別々に計算するのではなく，一つの行列演算で効率よく計算できる．入力、クエリ，キー，バリュー，出力のそれぞれを縦に結合した行列を，それぞれ X, Q, K, V, A と置く．出力 A は

$$A = Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d}})V$$

と表せる．Scaled Dot-Product Attention の処理の流れは図 2.5 のように表せる．

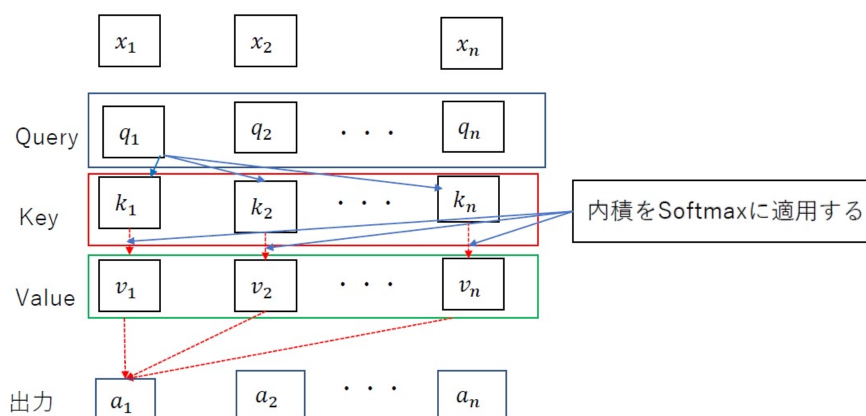

図 2.5: Scaled Dot-Product Attention の処理の流れ.

Attention ではあるトークンを処理するときに，すべてのトークンの情報を直接用いて出力を計算する．そのため，遠く離れたトークンの情報も適切に考慮することが可能である．

2.4.2　Multi-Head Attention

Multi-Head Attention は, クエリ, キー, バリューの組を複数用意し, それぞれの組に対して Scaled Dot-Produc Attention を適用し，最後に出力を一つに集約する手法である．Scaled Dot-Product Attention では，前の層の出力 x_i に対して，行列 W^Q, W^K, W^V を適用して q_i，k_i，v_i を得た．そのため，行列の組 $(W^Q_{(l)}$，$W^K_{(l)}$，$W^V_{(l)})(l = 1, 2, ..., h)$ を複数用意することで，クエリ，キー，バリューの組を複数作成することが出来る．それぞ

れの行列の組を用いて，Scaled Dot-Product Attention の出力 $a_i^{(l)}$ を得る．そして，それらを横に連結して一つのベクトルにし，さらに W_o により線形変換を行うことにより，最終的な出力 a_i を得る ($W_{(l)}^Q$，$W_{(l)}^K$，$W_{(l)}^V$，W^o はパラメータである)．

Multi-Head Attention もすべてのトークンをまとめて処理することができる．入力、クエリ，キー，バリューのそれぞれを縦に結合した行列を，それぞれ X, Q, K, V と置き，Scaled Dot-Product Attention の出力をすべてのトークンで縦に結合して行列にしたものを $A^{(l)}$ とし，Multi-Head Attention の出力を同様に行列にしたものを A と置く．すると，Multi-Head Attention の出力 A は

$$A = hstack(A^{(1)}, A^{(2)}, ..., A^{(h)})W^o$$

と表せる ($hstack$ は行列を横に結合する関数)．また Scaled Dot-Product Attention の出力 $A^{(l)}$ は

$$A^{(l)} = Attention(XW_{(l)}^Q, XW_{(l)}^K, XW_{(l)}^V)$$

と表せる．$XW_{(l)}^Q$ は X がどの部分を処理するかを決め，$XW_{(l)}^K$ は X への注目の仕方を決め，$XW_{(l)}^V$ は X を回して出力の様子を調整する役割を持つ．

2.4.3　BERT の入出力

BERT では 1 つの文または 2 つの文章のペアが入力される．単一の文章の場合は文章をトークン化したのち，トークン列の先頭に [CLS] トークンを，末尾に [SEP] トークンを加える．2 つの文章のペアの場合は，一つ目の文章のトークン列と 2 つ目の文章のトークン列を並べ，文の境界に [SEP] トークンを置き，トークン列の先頭に [CLS] トークンを，末尾に [SEP] トークンを加える．

文章をトークン列に変換したあとに，トークンをベクトルに置き換えて BERT に入力する．そのため，以下のようにトークン，文章タイプ，文章中の位置それぞれに応じたベクトルの和を BERT へ入力する．

- サイズが m(語彙数) の行列 E^T を用意し，語彙中の j 番目のトークンが現れた場合，E^T の j 行目の行ベクトルに置き換える．このように文章中の i 番目のトークンをベクトルに置き換えたものを e_i^T と置く．
- サイズが (2,m) の行列 E^S を用意する．BERT に入力する文章が 1 文のみの場合はそれぞれのトークンを E^S の 1 行目の行ベクトルに置き換える．BERT に入

力される文章が 2 文から構成される場合は，最初の文に含まれるトークンを E^S の 1 行目の行ベクトルに置き換え，2 つ目の文に含まれるトークンを E^S の 2 行目の行ベクトルに置き換える．このように文章中の i 番目のトークンをベクトルに置き換えたものを e_i^S と置く．

- BERT に入力可能な最大のトークンの数を L とし，サイズが (L,m) の行列 E^P を用意する．文章中の i 番目のトークンを行列 E^P の i 行目の行ベクトルに置き換え，e_i^P と置く．

最終的に，文章中の i 番目のトークンは 3 つの過程で得た 3 つのベクトルを足した e_i に置き換えて BERT に入力する．

$$e_i = e_i^T + e_i^S + e_i^P$$

2.4.4　事前学習

BERT の学習は先述したように，事前学習と fine-tuning を行う．

事前学習には比較的容易に大量のデータを収集できるため，ラベルのついていないデータのみで行う．また，事前学習には Masked Language Model(MLM) と Next Sentence Prediction(NSP) の 2 つがある．

Masked Language Model は，ある単語を m 割の単語から予測するというタスクである．まず，入力されたトークンの 15% を特殊トークン [MASK] に置き換える．そして，置き換えられた文章を BERT に入力し，[MASK] トークンの位置に元々あったトークンを予測する．

Next Sentence Prediction は二つの文章の関連性を理解するためのタスクである．事前学習には常に二つの文のペアが入力される．データの 50% は連続する文になっており，残りの 50% は連続していない文になっている．そして，2 つの文が連続しているかどうかを判定するタスクを用いて学習する．

2.4.5　fine-tuning

fine-tuning では解きたいタスクのラベル付きデータから，BERT がそのタスクに特化するように学習を行う．fine-tuning を行うときには，モデルのパラメータの初期値として，BERT のパラメータは事前学習で得られたパラメータを用い，新たに加えられた分類

器のパラメータにはランダムな値を与える．そして，ラベル付きデータを用いて BERT と分類器の両方のパラメータを学習する．事前学習で得られたパラメータを初期値として使うことで，比較的少数の学習データからでも高い性能のモデルを得ることができる．

事前学習と fine-tuning をまとめて転移学習 (Transfer Learning) と呼ばれる．一連の流れは図 2.6 のようになる．

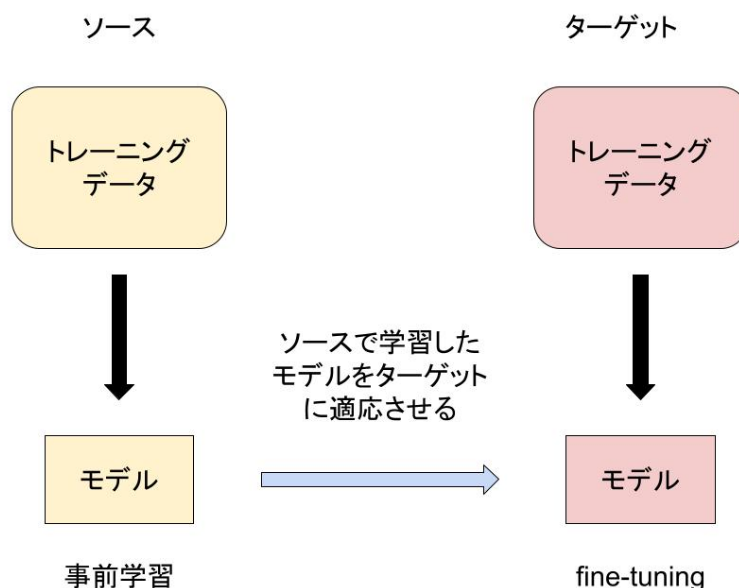

図 2.6: 転移学習の流れ．

2.5　Multilingual BERT

Multilingual BERT(mBERT) は BERT と同じモデルアーキテクチャと学習方法を持っている多言語の事前学習済みモデルである [1]．mBERT が事前学習に用いる Wikipedia のデータには 104 の言語が含まれている．mBERT では WordPiece モデリングにより，モデルが言語間で埋め込み表現を共有することが出来ている．語彙には様々な言語のキャラクタが入っており，語彙数は約 12 万 token ある．日本語の漢字は 1 文字で 2 文字の漢字単語はない．日本語のの 2 文字以上の token は約 1000 個ほどで，「になっている」，「となっていた」などの長いものもある．

Cross-Lingual な表現学習をしたモデルとして Goyal らの XLM-R がある [7]．XLM-R は様々なクロスリンガルベンチマークにおいて mBERT より大幅に優れている．ま

た，XLM-R は低リソース言語において特に優れた性能を示している．従来の XLM モデルと比較して XNLI の精度がスワヒリ語やウルドゥー語で向上している．このような，結果を得るためには

- 正の伝達と容量の希薄化
- 高リソース言語と低リソース言語のスケール

のトレードオフが性能に関係している．さらに，XLM-R は GLUE と XLNI において強力な単言語モデルと非常に高い競争力を有している．

2.6　Zero-Shot Cross-Lingual Transfer

Zero-Shot Cross-Lingual Transfer は Single-Source Transfer とも呼ばれ，ソース言語 (多くの場合，高リソース言語) でモデルを訓練し，その後ターゲット言語へ直接転送する手法である [4].

近年の BERT 等の事前学習済みモデルは，言語をまたいだ転移学習が可能であることが知られている．例えば英語やフランス語などの間には，似た語彙も多く，英語で学習した知識がフランス語のタスクに有用なのは当然のことである．さらに，この転移学習は事前学習用の言語 (L1) と微調整用の言語 (L2) との間に，共通の語彙が全く無くても可能である．この結果から，言語をまたいだ転移学習には人間の言語の何らかの構造的特徴が関連していると考えられている．

Li らの事前学習された言語モデルにおける新たな cross-lingual 構造 [8] によると，多言語エンコーダの最上層に共有パラメータが存在するため，単言語コーパス間で共有する語彙がない場合やテキストが非常に異なる領域のものであっても Zero-Shot Cross-Lingual Transfer が可能であると示されている．

Artexte らの単言語表現の言語伝達可能性について [5] では，言語 L1 で事前学習された単言語モデルを言語 L2 コーパスを利用し新しいトークンの埋め込みを学習することにより新しい言語 L2 に転送した (共有サブワードの概念がない) モデルは最先端の多言語モデルと Zero-Shot Cross-Lingual Transfer ベンチマークで同等に機能した．これは，多言語モデルでは語彙の共有も共同事前トレーニングも必要ないことを示している．

つまり，図 2.7 のようにメジャーな言語 A で fine-tunig をしたモデルで，訓練データに用いられていないマイナーな言語 B のテストデータで評価をすることが出来る．

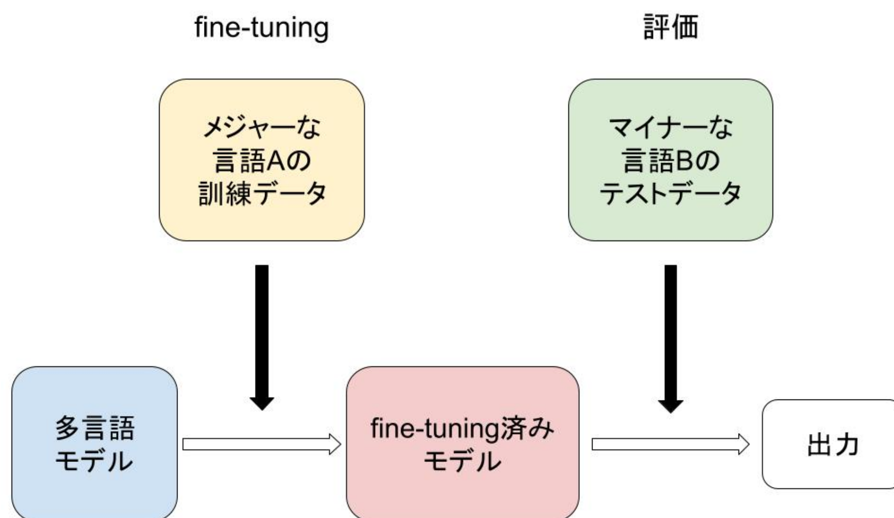

図 2.7: Zero-Shot Cross-Lingual Transfer の例.

第 3 章

訓練データの言語の多様性

3.1　言語の多様性の効果の確認

なんらかのタスクを Zero-Shot Cross-Lingual Transfer で解くとき，訓練データに言語の多様性を含ませた場合と訓練データに言語の多様性を含ませなかった場合の精度を比較することによって訓練データに言語の多様性を含ませる効果を確認できる．本研究では，文書分類タスクを Zero-Shot Cross Lingual Transfer で解いて精度を比較することによって言語の多様性を含ませる効果を確認する．

3.2　言語の多様性を含む訓練データの作成

mBERT では様々な言語を用いることが出来るが，本研究では英語，ドイツ語，フランス語のラベル付きデータを訓練データとして，日本語をテストデータとして用いる．すべてのデータに 1,2,4,5 の何れかのラベルが付いている．

本研究で用いる Webis-CLS-10 データセット[*1]には，すべての言語が「train」と「test」のフォルダに 2000 文ずつに分けられてあるが，どちらのデータも同じで整数値のラベルと Amazon レビューのテキストの組のため，「train」と「test」フォルダのデータを合わせた 4000 文のデータから抽出して訓練データとして用いる．日本語のテストデータは「test」フォルダの 2000 文のデータを用いる．

図 3.1 のように英語のデータ 3300 文から訓練データ 1000 文，検証データ 100 文，計 1100 文の組を 3 つ作りそれぞれ E1，E2，E3 とした．ドイツ語も同様に訓練データ 1000

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

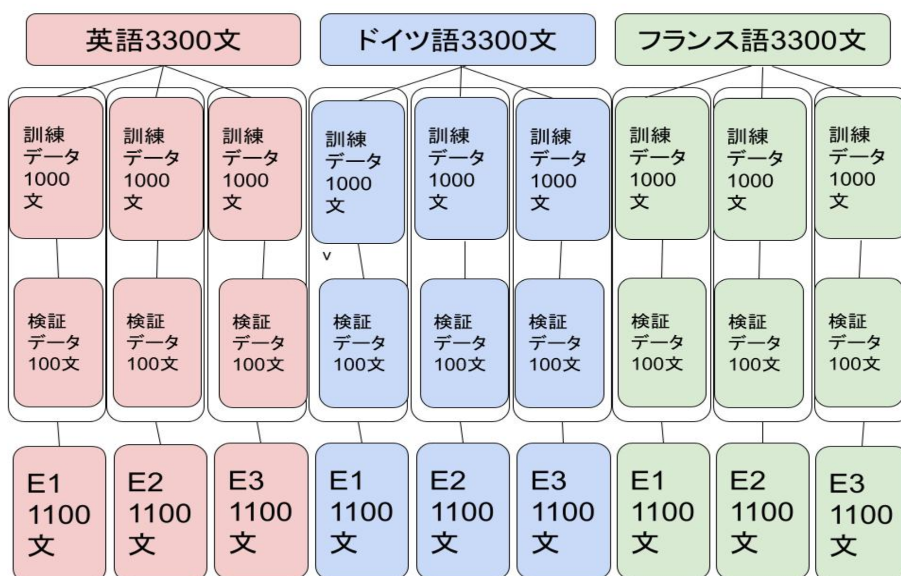

図 3.1: 1100 文の訓練データ

文，検証データ 100 文，計 1100 文の組を 3 つ作りそれぞれ D1，D2，D3 とした．フランス後も同様に訓練データ 1000 文，検証データ 100 文，計 1100 文の組を 3 つ作りそれぞれ F1，F2，F3 とした．

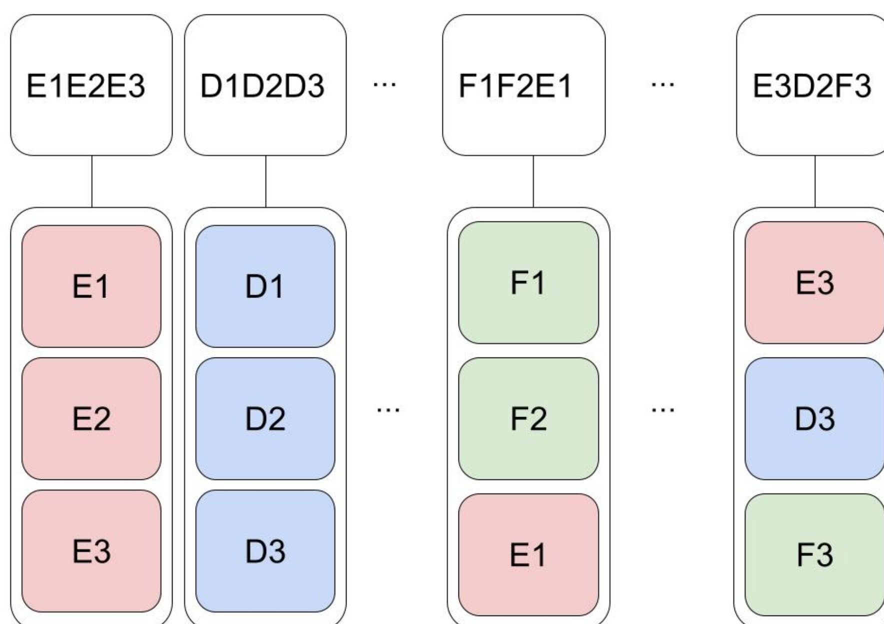

図 3.2: 言語の多様性を持たせた 3300 文の訓練データ.

さらに, 訓練データに言語の多様性の違いを持たせるために図 3.2 のように, 図 3.1 で作

成したデータを 3 つ合わせて訓練データ 3000 文, 検証データ 300 文の新たな組を作った. 1 つの言語から構成される言語の多様性を持っていない E1E2E3，D1D2D3，F1F2F3 の 3 組, 2 つの言語から構成される言語の多様性を持っている E1E2F1, E1E2D1, D1D2E1, D1D2F1，F1F2E1，F1F2D1 の 6 組，3 つの言語から構成される最も言語の多様性を持っている E1D1F1，E2D2F2，E3D3F3 の 3 組，計 12 組を作成した.

Zero-Shot Cross-Lingual Transfer を行うため訓練データには日本語を含んでいない.

第 4 章

実験

Google が公開している mBERT を用いた文書分類タスクで Zero-Shot Cross-Lingual Transfer を行うときに，訓練データに言語の多様性を持たせた場合の識別精度と持たせなかった場合の識別精度を比較し，訓練データに言語の多様性を持たせる効果を確認した．

4.1　事前学習済みモデル

Google で公開されているモデル (BERT-Base, Multilingual Cased) を使用した．これは Hugging Face 社の Transformers ライブラリから，モデル名 'bert-base-multilingual-cased' で利用できるモデルである．The lagest Wikipedias 内でトップ 104 の言語のデータが事前学習コーパスに利用されている．

4.2　実験用データセット

実験には Webis-CLS-10 データセット[*1]を用いた．このデータセットには日本語，英語，ドイツ語，フランス語が収録されている．ラベルはレビューの星の数であり，1 から 5 までの 5 段階評価である．ただし，ラベルが 3 のデータは存在しない．本実験ではラベルが 1，2 のデータを negative，4，5 のデータを positive として評判分析 (2 値分類) を行った．

このデータセットには日本語，英語，ドイツ語，フランス語それぞれに books，dvd，music の 3 つの領域がある．各言語の各領域に訓練データ 2000 文，テストデータ 2000

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

文がある．本実験ではすべて music の領域のデータを用いた．

4.3 文書分類器の作成

作成した訓練データ 3000 文，検証データ 300 文の 12 組で BERT-Base，Multilingual Cased に対して 10epoch の fine-tuning を行った．

4.4 実験結果

3.2 章で作成した訓練データを用いて fine-tuning を行ったモデルで評判分析を解いた．すべて日本語のテストデータ 2000 文で評価した．単言語の訓練データで fine-tuning を行い評価を行った場合の結果を表 4.1，訓練データの言語比 2:1 で fine-tuning を行い評価を行った場合の結果を表 4.2 に，訓練データの言語比 1:1:1 で fine-tuning を日本語で評価を行った結果を表 4.3 に示す．また，実験結果の平均値のグラフを図 4.1 に示す．

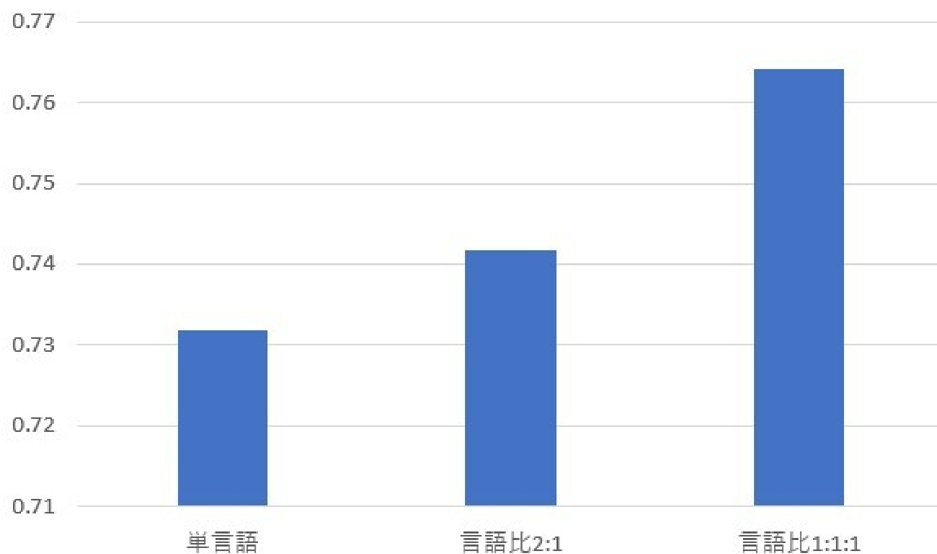

図 4.1: 実験結果の平均値のグラフ．

表 4.1: 単言語での学習

訓練データ	日本語で評価
E1E2E3	0.7310
D1D2D3	0.7450
F1F2F3	0.7190
平均値	0.7317

表 4.2: 言語比 2:1 での学習

訓練データ	日本語で評価
E1E2D1	0.7395
E1E2F1	0.7190
D1D2E1	0.7615
D1D2F1	0.7535
F1F2E1	0.7070
F1F2D1	0.7705
平均値	0.7418

表 4.3: 言語比 1:1:1 での学習

訓練データ	日本語で評価
E1D1F1	0.7630
E2D2F2	0.7610
E3D3F3	0.7685
平均値	0.7642

実験結果より，mBERT を用いて Zero-Shot Cross-Lingual Transfer を行う場合の訓練データは言語の多様性を含ませた場合の結果が最も良く，単言語の場合の結果が最も悪いことがわかった.

第 5 章

考察

3.2 章で作成した訓練データ 1000 文，検証データ 100 文の組を用いて fine-tuning を行ったモデルで 4 章同様に日本語のテストデータを用いた Zero-Shot Cross-Lingual Transfer を行った．結果を表 5.1 に示す．

表 5.1: 単言語の訓練データ 1000 文.

訓練データ	日本語で評価
E1	0.6940
E2	0.7315
E3	0.7000
D1	0.7660
D2	0.7005
D3	0.7480
F1	0.6625
F2	0.7080
F3	0.7130
平均値	0.7137

英語，ドイツ語，フランス語の中で最も精度が良かった E2，D1，F3 を用いて新たな組を作り，その組で fine-tuning を行ったモデルで実験した．精度は **0.7490** だった．これは，精度が良い訓練データを合わせても精度が向上するわけではないことを示している．

さらに，Zero-Shot Cross-Lingual Transfer ではない日本語の訓練データを用いた評

判分析を解く場合に言語の多様性を持たせる効果を確認するために実験を行った．日本語の訓練データ 1000 文，検証データ 100 文の組 J1 を作成した．そして，J1 を用いて**??**節で使用したモデルに対して fine-tuning を行い日本語のテストデータ 2000 文で評価した．実験結果を表 5.2 に示す．

表 5.2: 日本語 1000 文で fine-tuning

J1 で fine-tuning したモデル	日本語で評価
E1E2E3	0.7905
D1D2D3	0.7450
F1F2F3	0.8175
単言語の平均値	0.7843
E1E2D1	0.8035
E1E2F1	0.8230
D1D2E1	0.8105
D1D2F1	0.7985
F1F2E1	0.7855
F1F2D1	0.8185
言語比 2:1 の平均値	0.8066
E1D1F1	0.8015
E2D2F2	0.8030
E3D3F3	0.8095
言語比 1:1:1 の平均値	0.8047

表 5.2 より日本語を含む訓練データで fine-tuning する場合も Zero-Shot Cross-Lingual Transfer の場合と同様に訓練データに言語の多様性を含ませた結果が多様性を含ませなかった場合より良くなった．しかし，J1 の 1000 文だけで fine-tuning した結果は **0.8265** だった．この結果は訓練データに目的言語データが含まれている場合 (つまり Zero-Shot Cross-Lingual Transfer ではない場合) は，訓練データに別言語のデータを含ませない方ががよいことを示している．この点についてはさらに調査をする必要がある．

第 6 章

結論

本研究では mBERT を用いた Zero-Shot Cross-Lingual Transfer を行うときに，訓練データとして 3 種類の言語を含むデータ 3 組と 2 種類の言語を含むデータ 6 組と単言語のデータ 3 組を作成し，それぞれで fine-tuning を行い評判分析で精度を比較した．言語の多様性を最も持たせた場合の精度が最もよく，単言語の場合の精度が最も悪かった．

また，Zero-Shot Cross-Lingual Transfer ではない場合の訓練データに言語の多様性を含めることの効果について調べることを今後の課題とする．

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] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining, 2019.

[3] Naman Goyal, Jingfei Du, Myle Ott, Giri Anantharaman, and Alexis Conneau. Larger-scale transformers for multilingual masked language modeling, 2021.

[4] Shijie Wu and Mark Dredze. Beto, bentz, becas: The surprising cross-lingual effectiveness of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 833–844, Hong Kong, China, November 2019. Association for Computational Linguistics.

[5] Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4623–4637, Online, July 2020. Association for Computational Linguistics.

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

[7] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guil-

laume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8440–8451, Online, July 2020. Association for Computational Linguistics.

[8] Alexis Conneau, Shijie Wu, Haoran Li, Luke Zettlemoyer, and Veselin Stoyanov. Emerging cross-lingual structure in pretrained language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6022–6034, Online, July 2020. Association for Computational Linguistics.

付録

A　文書分類器のプログラム

英語 3300 文で fine-tuning して，日本語で Zero-Shot Cross-Lingual Transfer を行うためのプログラムを A.1 に示す．実験では 63,64 行目で指定する訓練データと検証データを変えることで言語の多様性の効果を確認した．また，5 章で行った Zero-Shot Cross-Lingual Transfer ではない日本語の訓練データ用いた評判分析を行うプログラムを A.2 に示す．このプログラムも 63,64 行目で指定する訓練データと検証データを変えることで言語の多様性の効果を確認した．

ソースコード A.1: Zero-Shot Cross-Lingual Transfer の評判分析のプログラム

```
1  from cmath import log
2  import os
3  import argparse
4  from pathlib import Path
5  import pathlib
6  from sre_constants import OP_IGNORE
7  from turtle import forward
8  import torch
9  from torch.utils.data import DataLoader, Dataset
10 from torch.optim.lr_scheduler import StepLR
11
12 import pytorch_lightning as pl
13 from pytorch_lightning.callbacks import EarlyStopping, ModelCheckpoint
14
15 from transformers import BertForSequenceClassification, BertTokenizer,
       AdamW, AutoTokenizer
16 import torchmetrics
17
18 from pytorch_lightning import loggers as pl_loggers
```

```
19 from pytorch_lightning import seed_everything
20
21 class ClassificationDataset(Dataset):#データセットの作成
22     def __init__(self, data_file: Path, tokenizer: BertTokenizer) ->
           None:
23         super().__init__()
24         self.input_ids = []
25
26         with open(data_file, 'r') as reader:
27             for line in reader:
28                 row = line.strip().split(',')
29                 if int(row[0]) < 3:#のラベルを 1,20(negative)に，のラベ
                       ルを 3,41(positive)にする
30                     l = 0
31                 else:
32                     l = 1
33                 label = l
34                 text = '␣'.join(row[1:])
35
36                 encoded = tokenizer.encode_plus(
37                     text,
38                     add_special_tokens=True,
39                     padding='max_length',
40                     truncation=True,
41                     return_attention_mask=True,
42                     return_tensors='pt'
43                 )
44
45                 self.input_ids.append(
46                     dict(
47                         input_ids=encoded['input_ids'].flatten(),
48                         attention_mask=encoded['attention_mask'].
                               flatten(),
49                         labels=torch.tensor(label)
50                     )
51                 )
52
53     def __len__(self) -> int:
54         return len(self.input_ids)
55
56     def __getitem__(self, index) -> dict:
```

```
57          return self.input_ids[index]
58
59
60  class ClassificationDataModule(pl.LightningDataModule):
61      def __init__(self, dataset_dir: Path, batch_size : int,
            pretrained_model_name: str) -> None:
62          super().__init__()
63          self.train_file = dataset_dir.joinpath('train_E1E2E3.csv')#訓練
                データの指定
64          self.val_file = dataset_dir.joinpath('val_E1E2E3.csv')#検証デー
                タの指定
65          self.test_file = dataset_dir.joinpath('test_ja.csv')#テストデー
                タの指定
66          self.batch_size = batch_size
67          self.tokenizer = AutoTokenizer.from_pretrained(
                pretrained_model_name)
68
69      def setup(self, stage = None) -> None:
70          self.train_dataset = ClassificationDataset(self.train_file,
                self.tokenizer)
71          self.val_dataset = ClassificationDataset(self.val_file, self.
                tokenizer)
72          self.test_dataset = ClassificationDataset(self.test_file, self
                .tokenizer)
73
74      def train_dataloader(self) -> DataLoader:
75          return DataLoader(self.train_dataset, batch_size=self.
                batch_size, shuffle=True, num_workers=os.cpu_count(),
                pin_memory=True)
76
77      def val_dataloader(self) -> DataLoader:
78          return DataLoader(self.val_dataset, batch_size=self.batch_size,
                 shuffle=False, num_workers=os.cpu_count(), pin_memory=
                True)
79
80      def test_dataloader(self) -> DataLoader:
81          return DataLoader(self.test_dataset, batch_size=self.batch_size
                , shuffle=False, num_workers=os.cpu_count(), pin_memory=
                True)
82
83
84  class Classifier(pl.LightningModule):#分類器の作成
```

```
85      def __init__(self, bert_model: BertForSequenceClassification) ->
            None:
86          super().__init__()
87          self.model = bert_model
88
89          self.metric = torchmetrics.Accuracy()
90
91      def forward(self, input_ids, attention_mask, labels=None):
92          outputs = self.model(input_ids, attention_mask=attention_mask,
                 labels=labels)
93          return outputs['loss'], outputs['logits']
94
95      def training_step(self, batch, batch_idx):
96          loss, preds = self.forward(input_ids=batch["input_ids"],
97                                      attention_mask=batch["
                                            attention_mask"],
98                                      labels=batch["labels"])
99          return {'loss': loss,
100                 'batch_preds': preds,
101                 'batch_labels': batch["labels"]}
102
103     def validation_step(self, batch, batch_idx):
104         loss, preds = self.forward(input_ids=batch["input_ids"],
105                                     attention_mask=batch["
                                            attention_mask"],
106                                     labels=batch["labels"])
107         return {'loss': loss,
108                 'batch_preds': preds,
109                 'batch_labels': batch["labels"]}
110
111     def test_step(self, batch, batch_idx):
112         loss, preds = self.forward(input_ids=batch["input_ids"],
113                                     attention_mask=batch["
                                            attention_mask"],
114                                     labels=batch["labels"])
115         return {'loss': loss,
116                 'batch_preds': preds,
117                 'batch_labels': batch["labels"]}
118
119     def training_epoch_end(self, outputs) -> None:
120         epoch_loss = torch.sum(torch.tensor([x['loss'] for x in
```

```
                outputs]))
121         self.log(
122             'train_loss',
123             epoch_loss,
124             prog_bar=True,
125             logger=True,
126             on_epoch=True
127         )
128 
129     def validation_epoch_end(self, outputs) -> None:
130         epoch_loss = torch.sum(torch.tensor([x['loss'] for x in
                outputs]))
131         self.log(
132             'valid_loss',
133             epoch_loss,
134             prog_bar=True,
135             logger=True,
136             on_epoch=True
137         )
138 
139     def test_epoch_end(self, outputs) -> None:
140         epoch_loss = torch.sum(torch.tensor([x['loss'] for x in
                outputs]))
141         epoch_preds = torch.cat([x['batch_preds'] for x in outputs])
142         epoch_labels = torch.cat([x['batch_labels'] for x in outputs])
143         self.metric(epoch_preds, epoch_labels)
144         self.log(
145             'test_loss',
146             epoch_loss,
147             prog_bar=True,
148             logger=True,
149             on_epoch=True
150         )
151         self.log(
152             'test_accuracy',
153             self.metric,
154             prog_bar=True,
155             logger=True,
156             on_epoch=True
157         )
158 
```

```
159     def configure_optimizers(self):
160         optimizer = AdamW(self.model.parameters(), lr=2e-5)
161         scheduler = {'scheduler': StepLR(optimizer=optimizer,
                step_size=1, gamma=0.2)}
162         return [optimizer]
163
164
165 if __name__ == '__main__':
166     parser = argparse.ArgumentParser()
167     parser.add_argument('--seed', type=int, default=50, help='seed')
168     args = parser.parse_args()
169
170     seed=args.seed
171     seed_everything(seed=seed, workers=True)
172     num_epoch = 10#エポック数
173
174     pretrained_model_name = 'bert-base-multilingual-cased'#事前学習済み
            モデルの指定
175     bert_model = BertForSequenceClassification.from_pretrained(
            pretrained_model_name, num_labels=2)
176     model = Classifier(bert_model)
177
178     dataset_dir = pathlib.Path('dataset', 'multi')
179     batch_size = 8#バッチサイズ
180
181     run_name = 'E1E2E3_10_zeroshot'
182
183     checkpoint_callback = ModelCheckpoint(
184         dirpath="./checkpoints/{}-seed{}".format(run_name, seed),
185         filename='{epoch}',
186         verbose=True,
187         monitor='valid_loss',
188         mode='min'
189     )
190
191     trainer = pl.Trainer(
192         max_epochs=num_epoch,
193         gpus=1,
194         callbacks=[checkpoint_callback],
195         logger=pl_loggers.TensorBoardLogger(save_dir='logs/{}/'.format(
                run_name))
```

```
196     )
197
198     data_module = ClassificationDataModule(dataset_dir=dataset_dir,
            batch_size=batch_size, pretrained_model_name=
            pretrained_model_name)
199
200     trainer.fit(model=model, datamodule=data_module)
201
202         #正解率の表示
203     result = trainer.test(ckpt_path=checkpoint_callback.best_model_path
            , datamodule=data_module)
```

ソースコード A.2: Zero-Shot Cross-Lingual Transfer ではない評判分析のプログラム

```
1 from cmath import log
2 import os
3 import argparse
4 from pathlib import Path
5 import pathlib
6 from sre_constants import OP_IGNORE
7 from turtle import forward
8 import torch
9 from torch.utils.data import DataLoader, Dataset
10 from torch.optim.lr_scheduler import StepLR
11
12 import pytorch_lightning as pl
13 from pytorch_lightning.callbacks import EarlyStopping, ModelCheckpoint
14
15 from transformers import BertForSequenceClassification, BertTokenizer,
       AdamW, AutoTokenizer
16 import torchmetrics
17
18 from pytorch_lightning import loggers as pl_loggers
19 from pytorch_lightning import seed_everything
20
21 class ClassificationDataset(Dataset):#データセットの作成
22     def __init__(self, data_file: Path, tokenizer: BertTokenizer) ->
           None:
23         super().__init__()
24         self.input_ids = []
25
```

```
26         with open(data_file, ’r’) as reader:
27             for line in reader:
28                 row = line.strip().split(’,’)
29                 if int(row[0]) < 3:#のラベルを 1,20(negative)に，のラベ
                       ルを 3,41(positive)にする
30                     l = 0
31                 else:
32                     l = 1
33                 label = l
34                 text = ’␣’.join(row[1:])
35 
36                 encoded = tokenizer.encode_plus(
37                     text,
38                     add_special_tokens=True,
39                     padding=’max_length’,
40                     truncation=True,
41                     return_attention_mask=True,
42                     return_tensors=’pt’
43                 )
44 
45                 self.input_ids.append(
46                     dict(
47                         input_ids=encoded[’input_ids’].flatten(),
48                         attention_mask=encoded[’attention_mask’].
                               flatten(),
49                         labels=torch.tensor(label)
50                     )
51                 )
52 
53     def __len__(self) -> int:
54         return len(self.input_ids)
55 
56     def __getitem__(self, index) -> dict:
57         return self.input_ids[index]
58 
59 
60 class ClassificationDataModule(pl.LightningDataModule):
61     def __init__(self, dataset_dir: Path, batch_size : int,
           pretrained_model_name: str) -> None:
62         super().__init__()
63         self.train_file = dataset_dir.joinpath(’train_E1E2E3.csv’)#訓練
```

```
            データの指定
64        self.val_file = dataset_dir.joinpath('val_E1E2E3.csv')#検証デー
            タの指定
65        self.test_file = dataset_dir.joinpath('test_ja.csv')#テストデー
            タの指定
66        self.batch_size = batch_size
67        self.tokenizer = AutoTokenizer.from_pretrained(
            pretrained_model_name)
68
69    def setup(self, stage = None) -> None:
70        self.train_dataset = ClassificationDataset(self.train_file,
            self.tokenizer)
71        self.val_dataset = ClassificationDataset(self.val_file, self.
            tokenizer)
72        self.test_dataset = ClassificationDataset(self.test_file, self
            .tokenizer)
73
74    def train_dataloader(self) -> DataLoader:
75        return DataLoader(self.train_dataset, batch_size=self.
            batch_size, shuffle=True, num_workers=os.cpu_count(),
            pin_memory=True)
76
77    def val_dataloader(self) -> DataLoader:
78        return DataLoader(self.val_dataset, batch_size=self.batch_size,
             shuffle=False, num_workers=os.cpu_count(), pin_memory=
            True)
79
80    def test_dataloader(self) -> DataLoader:
81        return DataLoader(self.test_dataset, batch_size=self.batch_size
            , shuffle=False, num_workers=os.cpu_count(), pin_memory=
            True)
82
83 class ClassificationDataModule2(pl.LightningDataModule):
84    def __init__(self, dataset_dir: Path, batch_size : int,
        pretrained_model_name: str) -> None:
85        super().__init__()
86        self.train_file = dataset_dir.joinpath('train_J1.csv')#日本語で
            fine-するための訓練データの指定 tuning
87        self.val_file = dataset_dir.joinpath('val_J1.csv')#日本語で
            fine-するための検証データの指定 tuning
88        self.test_file = dataset_dir.joinpath('test_ja.csv')#テストデー
            タの指定
89        self.batch_size = batch_size
```

```
90          self.tokenizer = AutoTokenizer.from_pretrained(
                pretrained_model_name)
91
92      def setup(self, stage = None) -> None:
93          self.train_dataset = ClassificationDataset(self.train_file,
                self.tokenizer)
94          self.val_dataset = ClassificationDataset(self.val_file, self.
                tokenizer)
95          self.test_dataset = ClassificationDataset(self.test_file, self
                .tokenizer)
96
97      def train_dataloader(self) -> DataLoader:
98          return DataLoader(self.train_dataset, batch_size=self.
                batch_size, shuffle=True, num_workers=os.cpu_count(),
                pin_memory=True)
99
100     def val_dataloader(self) -> DataLoader:
101         return DataLoader(self.val_dataset, batch_size=self.batch_size,
                 shuffle=False, num_workers=os.cpu_count(), pin_memory=
                True)
102
103     def test_dataloader(self) -> DataLoader:
104         return DataLoader(self.test_dataset, batch_size=self.batch_size
                , shuffle=False, num_workers=os.cpu_count(), pin_memory=
                True)
105
106 class Classifier(pl.LightningModule):#分類器の作成
107     def __init__(self, bert_model: BertForSequenceClassification) ->
            None:
108         super().__init__()
109         self.model = bert_model
110
111         self.metric = torchmetrics.Accuracy()
112
113     def forward(self, input_ids, attention_mask, labels=None):
114         outputs = self.model(input_ids, attention_mask=attention_mask,
                 labels=labels)
115         return outputs['loss'], outputs['logits']
116
117     def training_step(self, batch, batch_idx):
118         loss, preds = self.forward(input_ids=batch["input_ids"],
```

```
119                                    attention_mask=batch["
                                           attention_mask"],
120                                    labels=batch["labels"])
121         return {'loss': loss,
122                 'batch_preds': preds,
123                 'batch_labels': batch["labels"]}
124 
125     def validation_step(self, batch, batch_idx):
126         loss, preds = self.forward(input_ids=batch["input_ids"],
127                                    attention_mask=batch["
                                           attention_mask"],
128                                    labels=batch["labels"])
129         return {'loss': loss,
130                 'batch_preds': preds,
131                 'batch_labels': batch["labels"]}
132 
133     def test_step(self, batch, batch_idx):
134         loss, preds = self.forward(input_ids=batch["input_ids"],
135                                    attention_mask=batch["
                                           attention_mask"],
136                                    labels=batch["labels"])
137         return {'loss': loss,
138                 'batch_preds': preds,
139                 'batch_labels': batch["labels"]}
140 
141     def training_epoch_end(self, outputs) -> None:
142         epoch_loss = torch.sum(torch.tensor([x['loss'] for x in
                outputs]))
143         self.log(
144             'train_loss',
145             epoch_loss,
146             prog_bar=True,
147             logger=True,
148             on_epoch=True
149         )
150 
151     def validation_epoch_end(self, outputs) -> None:
152         epoch_loss = torch.sum(torch.tensor([x['loss'] for x in
                outputs]))
153         self.log(
154             'valid_loss',
```

```
155             epoch_loss,
156             prog_bar=True,
157             logger=True,
158             on_epoch=True
159         )
160
161     def test_epoch_end(self, outputs) -> None:
162         epoch_loss = torch.sum(torch.tensor([x['loss'] for x in
                outputs]))
163         epoch_preds = torch.cat([x['batch_preds'] for x in outputs])
164         epoch_labels = torch.cat([x['batch_labels'] for x in outputs])
165         self.metric(epoch_preds, epoch_labels)
166         self.log(
167             'test_loss',
168             epoch_loss,
169             prog_bar=True,
170             logger=True,
171             on_epoch=True
172         )
173         self.log(
174             'test_accuracy',
175             self.metric,
176             prog_bar=True,
177             logger=True,
178             on_epoch=True
179         )
180
181     def configure_optimizers(self):
182         optimizer = AdamW(self.model.parameters(), lr=2e-5)
183         scheduler = {'scheduler': StepLR(optimizer=optimizer,
                step_size=1, gamma=0.2)}
184         return [optimizer]
185
186
187 if __name__ == '__main__':
188     parser = argparse.ArgumentParser()
189     parser.add_argument('--seed', type=int, default=50, help='seed')
190     args = parser.parse_args()
191
192     seed=args.seed
193     seed_everything(seed=seed, workers=True)
```

```
194    num_epoch = 10#エポック数
195
196    pretrained_model_name = 'bert-base-multilingual-cased'#事前学習済み
           モデルの指定
197    bert_model = BertForSequenceClassification.from_pretrained(
           pretrained_model_name, num_labels=2)
198    model = Classifier(bert_model)
199
200    dataset_dir = pathlib.Path('dataset', 'multi')
201    batch_size = 8#バッチサイズ
202
203    run_name = 'E1E2E3J1_10_zeroshot'
204
205    checkpoint_callback = ModelCheckpoint(
206        dirpath="./checkpoints/{}-seed{}".format(run_name, seed),
207        filename='{epoch}',
208        verbose=True,
209        monitor='valid_loss',
210        mode='min'
211    )
212
213    trainer = pl.Trainer(
214        max_epochs=num_epoch,
215        gpus=1,
216        callbacks=[checkpoint_callback],
217        logger=pl_loggers.TensorBoardLogger(save_dir='logs/{}/'.format(
               run_name))
218    )
219
220    data_module = ClassificationDataModule(dataset_dir=dataset_dir,
           batch_size=batch_size, pretrained_model_name=
           pretrained_model_name)
221    trainer.fit(model=model, datamodule=data_module)
222
223        #日本語でファインチューニングする前のモデルの正解率の表示
224    result = trainer.test(ckpt_path=checkpoint_callback.best_model_path
           , datamodule=data_module)
225
226    ###J1finetuning###
227
228    ckpt_path=checkpoint_callback.best_model_path
```

```
229
230     data_module2 = ClassificationDataModule2(dataset_dir=dataset_dir,
            batch_size=batch_size, pretrained_model_name=
            pretrained_model_name)
231
232     trainer2 = pl.Trainer(
233         resume_from_checkpoint=ckpt_path,#最良のモデルの読み込み
234         max_epochs=num_epoch,
235         gpus=1,
236         callbacks=[checkpoint_callback],
237         logger=pl_loggers.TensorBoardLogger(save_dir='logs/{}/'.format(
                run_name))
238     )
239
240     trainer2.fit(model=model, datamodule=data_module2)
241
242     #日本語でファインチューニングしたモデルの正解率の表示
243     result = trainer2.test(ckpt_path=checkpoint_callback.
            best_model_path, datamodule=data_module2)
```