2023 年度卒業研究レポート

BERT での感情分析における Optuna を使った最適化

明治大学 総合数理学部 現象数理学科

4 年 1 組 20 番 2610200047 木之下哲汰

2024 年 2 月 29 日

目 次

第1章　はじめに

1.1　本研究の動機

私は近年の機械学習技術の急速な発展とその社会への影響に大きな興味を抱いている。特に、BERT や Optuna のような最先端技術がどのようにして機械学習モデルの性能を飛躍的に向上させているのか、そのメカニズムを深く理解したいと考えた。機械学習はただ技術を学ぶだけでなく、その技術を通じて実社会の問題を解決するための強力なツールであると認識している。機械翻訳や文章分類のような機械学習のタスクの中でも、感情分析は特に興味深い分野であり、人々の意見や感情を正確に把握し解析することで、より良い社会の実現に寄与する可能性を秘めている。また、先輩が取り組んでいた自然言語処理に関する研究が関心を大きく高めることとなり、この分野で新たな貢献を目指すきっかけとなった。

1.2　本研究の目的

本研究の究極的な目的は、BERT による感情分析の精度をさらに向上させるための有効な手法を探求し、従来の手法との比較を通じてその優位性を明らかにすることである。本研究の実験においては、BERT を用いた感情分析における Optuna を活用したハイパーパラメータ最適化の効果を検証し、感情分析の精度と効率性を向上させることを目的とする。この研究を通じて、感情分析のための BERT モデルの性能を最大化するための実践的なガイドラインを提供することを目指す。

第2章　理論的背景

2.1　はじめに

本章では機械学習や自然言語処理、Transformer などの理論的背景について述べる。

2.2　機械学習 (Machine Learning)

機械学習は大量のデータでモデルを効率的に学習させ、何らかの問題を解くことである。ここで機械学習のモデルは、入力に対し出力を導き出す仕組みのことである。機械学習の応用できる技術として、検索エンジン、機械翻訳、文章分類、市場予測、作画や作曲などのアート、音声認識、医療、ロボット工学など多岐にわたる。

機械学習には学習と推論の二つの過程がある。例えば、後述する教師あり学習と呼ばれる方法では、まず入力データとそれに対する正解のデータ (ラベル) を合わせたラベル付きデータを大量に用意する。それを用いて望ましい入出力関係をモデルに学習させる。これを学習と呼ぶ。そして、新たな入力データに対して、学習済みのモデルを用いてラベル (正解) を予測することで、実際に問題を解く。これを推論と呼ぶ。

機械学習には教師あり学習、教師なし学習、強化学習の 3 つの主要な学習方法がある。

- 教師あり学習:正解となる答えが含まれたデータをモデルに学習させる方法のことである。また、自動的に正解となる答えを生成して学習を行う手法を「自己教師あり学習」と呼ぶ。
- 教師なし学習:正解データを使用せず与えられたデータの本質的な構造やアルゴリズムから自動的にデータの特徴を捉える方法のことである。
- 強化学習:与えられた環境のやり取りから、自らの成果を最大化するように何度も試行錯誤を繰り返し、最適な挙動をするように学習する方法のことである。

本研究では教師あり学習を行い、後述する BERT というモデルを用いて文章分類を行なった。

2.3　ニューラルネットワーク (Neural Network)[2]

ニューラルネットワークは、人間の脳神経をモデルにした情報処理システムである。ニューラルネットワークを構成する最小の要素をユニットと言い、ユニットは複数の入力を受け取り、1 つの数を出力として計算する。図 2.1 のような 4 つの入力 x_1, x_2, x_3, x_4 の場合、各入力に異なる重み w_1, w_2, w_3, w_4 を掛けバイアス b を加算した総入力 u は下記のように計算される。

$$u = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \tag{2.1}$$

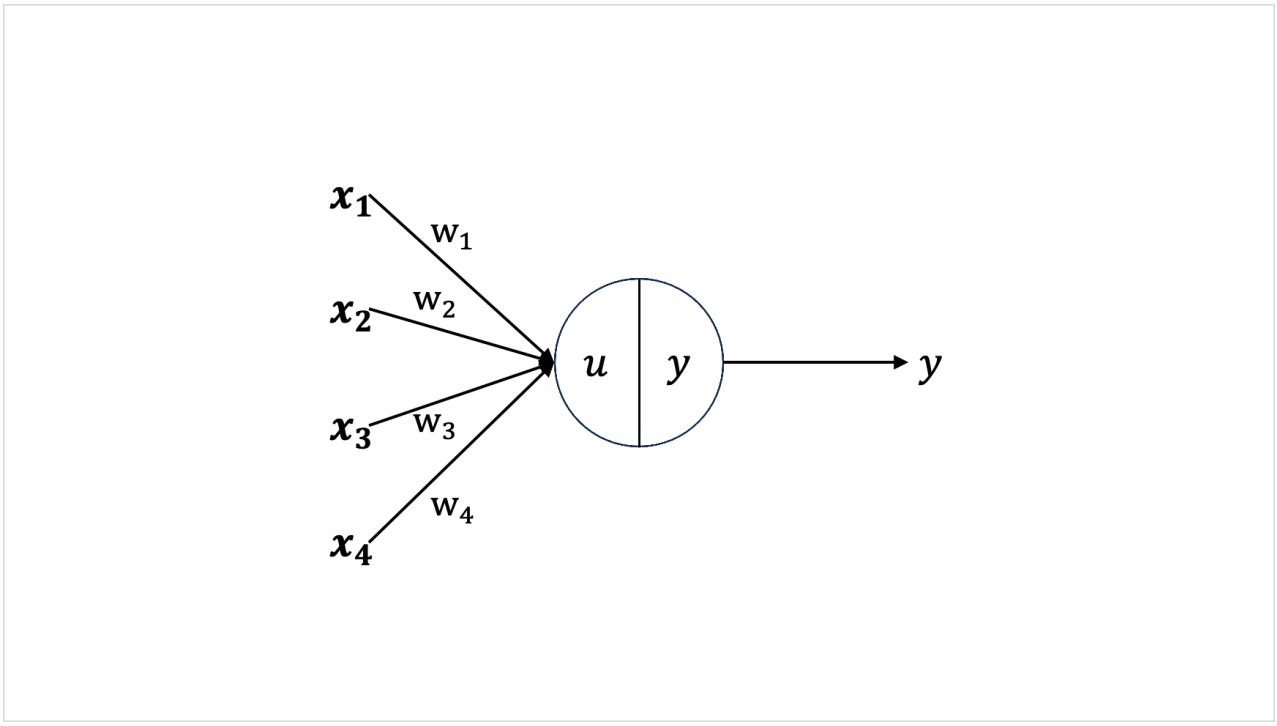

図 2.1: ユニット

この u を活性化関数と呼ばれる f に入れて出力 y が計算される

$$z = f(u) \tag{2.2}$$

活性化関数は、主に ReLU(Rectified Linear Unit)、シグモイド関数が用いられている。

ReLU(Rectified Linear Unit) は、

$$f(x) = \max(0, x) \tag{2.3}$$

シグモイド関数は、

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{2.4}$$

で示される。

この図 2.1 に層 l を一つ追加し、$l = 1, 2, 3$ のネットワークを考える。ここで層は、ユニットの縦方向の集まりのことである。なお $l = 1$ は入力層、$l = 2$ は中間層 (隠れ層)、$l = 3$ は出力層という。

各ユニットの入出力を区別するために、各変数の右肩に層の番号を付け、$u^{(l)}$ や $z^{(l)}$ のように書くことにする。また、入力とユニットの数を一般化し、入力を $i = 1, 2, \ldots, I$、ユニットを $j = 1, 2, \ldots, J$、層を $l = 1, 2, \ldots, L$ とする。これらを用いて第 l 層の i 番目のユニットに紐づいている重みのうち、j 番目の重みを ${w^{(l)}}_{j,i}$ と表現すると、図 2.2 のように示される。

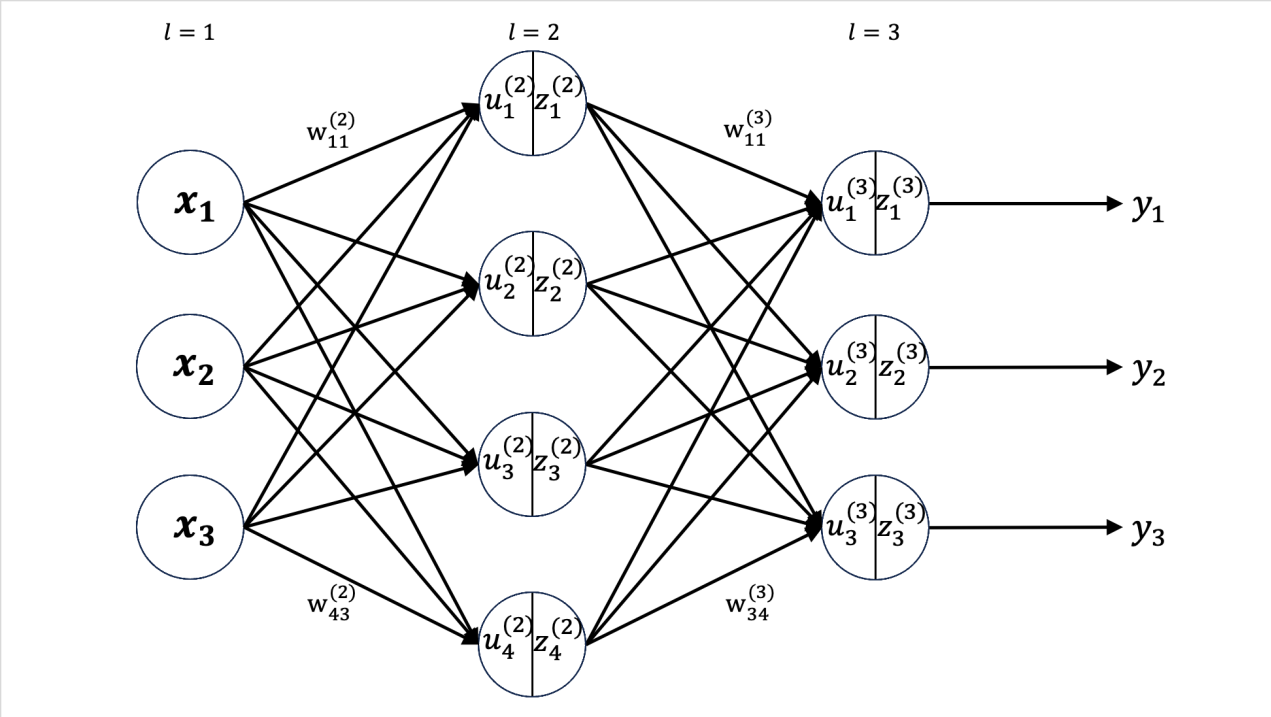

図 2.2: ニューラルネットワーク

各ユニット $(j = 1, 2, \ldots, J)$ の出力は次のように計算される。

$$\mathbf{u}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{z}^{(l)} + \mathbf{b}^{(l+1)} \tag{2.5}$$

$$\mathbf{z}^{(l+1)} = \mathbf{f}(\mathbf{u}^{(l+1)}) \tag{2.6}$$

ただし $\mathbf{z}^{(1)} = \mathbf{x}, \mathbf{y} \equiv \mathbf{z}^{(L)}$ とし、それぞれのベクトルと行列を次のように定義する。

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_J \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_I \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_J \end{bmatrix}, \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_J \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1I} \\ w_{21} & w_{22} & \cdots & w_{2I} \\ \vdots & \vdots & \ddots & \vdots \\ w_{J1} & w_{J2} & \cdots & w_{JI} \end{bmatrix}, \mathbf{f}(\mathbf{u}) = \begin{bmatrix} f(u_1) \\ f(u_2) \\ \vdots \\ f(u_J) \end{bmatrix}$$

ここで、表記の簡素化のため単一の活性化関数 $\mathbf{f}$ を用いているが、各層で異なっていても構わない。特に出力層のユニットの活性化関数はタスクに応じて選定され、中間層のものとは一般に異なる。

また、本章において $\mathbf{y}$ は $\mathbf{x}$ の関数であるから、以降 $\mathbf{y}(\mathbf{x})$ と表す。

2.3.1 学習の概要

入力 $\mathbf{x}$ に対する望ましい出力 $\mathbf{d}$ のペアが複数与えられているとして、その集合 $\mathcal{D}$ を下記のように表す。

$$\mathcal{D} = \{(\mathbf{x}_n, \mathbf{d}_n)\}_{n=1,\ldots,N} = \{(\mathbf{x}_1, \mathbf{d}_1), (\mathbf{x}_2, \mathbf{d}_2), \ldots, (\mathbf{x}_N, \mathbf{d}_N)\} \tag{2.7}$$

これらのペア $(\mathbf{x}_n, \mathbf{d}_n)$ 一つ一つを訓練サンプルと呼び、その集合を訓練データと呼ぶ。

ニューラルネットワークにおける学習は、与えられた訓練データのすべての訓練サンプル $(\mathbf{x}_n, \mathbf{d}_n)(n = 1, ..., N)$ について、入力 $\mathbf{x}_n$ を与えたときのニューラルネットワークの出力 $\mathbf{y}(\mathbf{x}_n)$ が、なるべく $\mathbf{d}_n$ に近くなるように重みとバイアスを調整することである。以降、モデル内の全ての重み $\mathbf{W}^{(2)}, \ldots, \mathbf{W}^{(L)}$ とバイアス $\mathbf{b}^{(2)}, \ldots, \mathbf{b}^{(L)}$ を成分に持つベクトル Θ をパラメータと呼ぶ。

$$\Theta = [\mathbf{W}^{(2)}, \mathbf{W}^{(3)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(2)}, \mathbf{b}^{(3)}, \ldots, \mathbf{b}^{(L)}] \tag{2.8}$$

ニューラルネットワークでは重みとバイアスという「学習するパラメータ」を持つ。ニューラルネットワークにおいては、学習するパラメータ以外にも、層の数、各層のニューロンの数、ミニバッチ学習を行う際のミニバッチサイズ、エポック数、学習率などのパラメータがある。これらのパラメータをハイパーパラメータと呼ぶ。

2.3.2 損失関数 (Loss Function)

損失関数は機械学習のモデルが算出した予測値と、正解値のずれを計算するための関数である。本論文のようなポジティブ、ネガティブの二値分類における損失関数は、事後分布 $p(k|\mathbf{x})$ を用いて計算される。一つの文章 $\mathbf{x}$ に対し、$k = 0$ ならネガティブ、$k = 1$ ならポジティブというように、種類を 2 値を変数 $k \in \{0, 1\}$ を表現する。入力 $\mathbf{x}$ が与えられた時、事後確率 $p(k = 1|\mathbf{x})$ を計算し、その値が 0.5 を越えれば $k = 1$、下回れば $k = 0$ と判断する。目標は入力 $\mathbf{x}$ から k を推定することである。

事後確率をモデル化するのにニューラルネットワークを用いる。このネットワークは出力層にユニットを 1 つだけ持ち、活性化関数をシグモイド関数とする。このネットワークが $\mathbf{x}$ に対し与える出力 $y(\mathbf{x}; \Theta)$ は、事後確率 $p(k = 1|\mathbf{x})$ を表現するものと考える。

$$p(k = 1|\mathbf{x}) = y(\mathbf{x}; \Theta) \tag{2.9}$$

また $p(k = 0|\mathbf{x})$ は、

$$p(k = 0|\mathbf{x}) = 1 - y(\mathbf{x}; \Theta) \tag{2.10}$$

と表現する。

この事後分布のモデルが訓練データ $\{(\mathbf{x}_n, \mathbf{d}_n)\}_{n=1,\ldots,N}$ と最も整合するようにネットワークのパラメータ Θ を決定する。ここで $\mathbf{d}_n$ はスカラーであり、$\mathbf{x}$ が $k = 0$ か $k = 1$ を表し、$\mathbf{d}_n \in \{0, 1\}$ である。事後分布 $p(k|\mathbf{x}) = p(k|\mathbf{x}; \Theta)$ は $k = 0$ と $k = 1$ の事後確率を用いて下記のように計算される。

$$p(k|\mathbf{x}) = p(k = 1|\mathbf{x})^k p(k = 0|\mathbf{x})^{1-k} \tag{2.11}$$

ここで、与えられた訓練データ $\{(\mathbf{x}_n, \mathbf{d}_n)\}_{n=1,\ldots,N}$ に対する Θ の尤度を求め、それを最大化する Θ の値を選ぶ。尤度はある前提条件に従って結果が出現する場合に、逆に観察結果からみて前提条件が「何々であった」と推測する、尤もらしさを表す数値である。Θ の尤度 $L(\Theta)$ は、

$$L(\Theta) \equiv \prod_{n=1}^{N} p(d_n|\mathbf{x}; \Theta) = \prod_{n=1}^{N} \{y(\mathbf{x}_n; \Theta)\}^{d_n} \{y(\mathbf{x}_n; \Theta)\}^{1-d_n} \tag{2.12}$$

この式に対数をとり、符号を反転した式が、二値分類における損失関数である。

$$E(\Theta) = -\sum_{n=1}^{N} [d_n \log y(\mathbf{x}_n; \Theta) + (1 - d_n) \log \{1 - y(\mathbf{x}_n; \Theta)\}] \tag{2.13}$$

また、出力層のユニットの活性化関数にシグモイド関数を選択したことから、事後確率 $p(k=1|\mathbf{x})$ は条件付き確率の定義より下記のように解釈することができる。

$$p(k=1|\mathbf{x}) = \frac{p(\mathbf{x}, k=1)}{p(\mathbf{x}, k=0) + p(\mathbf{x}, k=1)} \tag{2.14}$$

ここで、

$$u \equiv \log \frac{p(\mathbf{x}, k=1)}{p(\mathbf{x}, k=0)} \tag{2.15}$$

とおくとシグモイド関数 $f(u)$ は

$$\begin{aligned} f(u) &\equiv \frac{1}{1+\exp(-\log \frac{p(\mathbf{x},k=1)}{p(\mathbf{x},k=0)})} \\ &= \frac{1}{1+\exp(\log \frac{p(\mathbf{x},k=0)}{p(\mathbf{x},k=1)})} \\ &= \frac{1}{1+\frac{p(\mathbf{x},k=0)}{p(\mathbf{x},k=1)}} \\ &= \frac{p(\mathbf{x}, k=1)}{p(\mathbf{x}, k=1) + p(\mathbf{x}, k=0)} \end{aligned}$$

より、事後確率 $p(k=1|\mathbf{x})$ は u のシグモイド関数と一致することがわかる。これにより、ニューラルネットワークの出力を直接、ある入力 $\mathbf{x}$ がクラス $k=1$ に属する確率として解釈できるようになる。

2.3.3 バッチ学習

本項ではバッチ学習の仕組みについて説明する。

ニューラルネットワークの学習は、訓練データ $\mathcal{D}$ に対し計算される損失関数 $E(\Theta)$ を最小化することに帰着される。そのためには損失関数を最小化するようにパラメータ Θ の更新を行う。更新には損失関数の勾配を用いる。損失関数の勾配を下記のようなベクトルで示す。

$$\nabla E = \frac{\partial E}{\partial \Theta} = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_M}\right]^\top \tag{2.16}$$

M は $\mathbf{w}$ の成分数 $(=2(L-1))$ である。最適な Θ の値を求めるため、反復法を用いる。反復の t ステップ目の Θ の値を Θ_t と表す。更新量の大きさを定める学習率を η とすると更新は以下の式で行われる。

$$E(\Theta_t) = \sum_{n=1}^{N} E_n(\Theta_t x) \tag{2.17}$$

$$\Theta_{t+1} = \Theta_t - \eta \nabla E \tag{2.18}$$

この時の η を学習率と呼ぶ。このように、すべての訓練サンプルを使って Θ を更新する方法をバッチ学習と呼ぶ。

2.3.4 ミニバッチ学習

バッチ学習は最小化する目的関数が常に同じなので、局所的な極小解から抜け出すことができないことがある。そこで一定数の訓練サンプルの集合単位で重みを更新する方法が用いられることが一般的である。この訓練サンプルの集合のことをミニバッチと呼ぶ。学習で t 回目の更新に用いるミニバッチを $\mathcal{D}_t$ とするとき t 回目の更新は下記の式のように計算される。

$$E_{\text{batch}}(\Theta) = \frac{1}{N_t} \sum_{n \in \mathcal{D}_t} E_n(\Theta) \tag{2.19}$$

$$\Theta_{t+1} = \Theta_t - \eta \nabla E_{\text{batch}}(\Theta_t) \tag{2.20}$$

この時の $N_t = |\mathcal{D}_t|$ はこのミニバッチが含むサンプル数であり、バッチサイズと呼ぶ。

ミニバッチを $\mathcal{D}_1, \mathcal{D}_2, \ldots$ と取り出して使い、一巡したら再度、最初から $\mathcal{D}_1, \mathcal{D}_2, \ldots$ の順に取り出して使う。ミニバッチ学習による w の更新回数をステップ数と呼ぶ. また、ミニバッチが一巡することをエポックと呼ぶ。

2.4 自然言語処理 (NLP: Natural Learning Processing)

日本語や英語など、私たちが日常生活で用いる言語を自然言語と呼ばれ、一方でプログラミング言語のような特定の用途のためだけに人工的に作られた言語を人工言語と呼ばれる。自然言語処理は、自然言語の関わる問題 (タスク) をコンピュータで解くことである。タスクは、固有表現抽出のような言語の持つ意味や構造を扱う基礎的なものから、文章分類や文章生成のような人間の行動を模倣・代替するような応用的なものまで幅広く存在する。

2.4.1 言語モデル

言語モデルは文章の出現しやすさを確率によってモデル化する手法である。言語モデルは「文章の自然さを確率によって表現している」と捉えることができる。例えば、下記のような人間がもっともらしいと感じる文章には高い確率を、そうでない文章には低い確率を与える。

$$p(\text{”私はパンを食べた”}) > p(\text{”私は家を食べた”})$$

$$p(\text{”私はパンを食べた”}) > p(\text{”私にパンを食べた”})$$

この時、「私」「は」「パン」「を」「食べ」「た」のような、モデルが扱いやすいように文章を分割されたものをトークンという。ここで、文章 S のあるトークン $(w_1, w_2, \ldots, w_n)$ を考える。文章 S の出現確率は、トークンの同時確率と同値である。

$$p(S) = p(w_1, w_2, \ldots, w_n) \tag{2.21}$$

ここで、あるトークンの出現確率が、それ以前に出現したトークンに依存していることは直感

的に明らかである。このことから、トークンの同時確率は、$p(w_1) = p(w_1|\cdot)$ であることを考慮して下記のような条件付き確率の積で求められる。

$$p(w_1, w_2, \ldots, w_n) = p(w_1) \times p(w_2 \mid w_1) \times p(w_3 \mid w_1, w_2) \times \ldots \quad = \prod_{i=1}^{n} p(w_i \mid \mathbf{c}_i) \qquad (2.22)$$

ここで $\mathbf{c}_i$ は w_i を予測する際の前提条件であり、この場合は w_i より前のトークン列 $\mathbf{c}_i = (w_1, w_2, \ldots, w_{i-1})$ である。以降この $\mathbf{c}_i$ を文脈と呼ぶ。
以上から、文章の出現確率をモデル化するためには、ある文脈化でのトークンはトークン出現確率である $p(w_i \mid \mathbf{c})$ を数学的に表現すれば良いということがわかる。

2.4.2 大規模言語モデル

大規模言語モデルは、大規模なテキストデータで訓練された大規模なパラメータで攻囲せされるニューラルネットワークである。(ここでの「大規模」がニューラルネットワークのパラメータ数、訓練に使われるコーパスの容量、訓練時の計算量のうちどれを指すのか、またパラメータ数を指すならば、どの程度のパラメータ数のモデルから大規模言語モデルに含まれるかは、不明確である。) 大規模言語モデルには二つの学習の過程がある。

1. 事前学習：大規模な文章の集合 (コーパス) から汎用的な言語のパターンを学習する。
2. ファインチューニング：個別のタスクのラベル付きデータを用いてそのタスクに特化させるように学習する。

後述する Transformer も大規模言語モデルの一つであり、2023 年時点で Transformer を大規模のテキストデータでの自己教師あり学習で事前学習し、そのモデルをファインチューニングして解く方法が、自然言語処理の標準的な手法になっている。

2.4.3 トークン化

トークン化は、文を適当な単位に分割することである。モデルが扱う基本的な単位を「トークン (token)」、トークンに分割する実装を「トークナイザ (tokenizer)」という。このトークン化により得られたトークンが、ニューラルネットワークの入力に変換される。トークン化は以下の手順によって行われる。

1. 事前に適当な方法で入力として扱いたいトークンの集合 (語彙) を作成し、これに含まれる各トークンに対して順番に ID を割り当てておく。
2. 渡されたトークンを語彙に従い ID に変換する。

BERT では、サブワード分割と呼ばれる分割方法が採用されている。
サブワード分割は、単語をさらに部分列文字に分割するような方法のことである。サブワードは部分文字列を意味している。たとえば下記のようなサブワードの語彙を考えてみる。

[”東京”,”京都”,”明治”,”＃＃大学”,”＃＃タワー”]

「＃＃」は単語の途中に現れる要素を示している。この語彙では「東京」や「京都」などの単語の他に「明治」と「＃＃タワー」を組み合わせて「明治タワー」のような単語も表現できる。
サブワード分割は単語単位で分割する方法に比べて少ない語彙数で多くの入力を表現することができ、日本語の BERT で採用されている。

2.4.4 Word2Vec

Word2Vec は、2013 年に発表された [6] 単語の意味を表現したベクトルを大規模なテキストから学習できることを示したニューラルネットワークのことである。Word2Vec は「ある単語の意味は周辺に出現する単語によって表現できる」という分布仮説の考えに基づいて設計されている。本来はモデルに高次元のベクトルが与えられるが、ここでは例として「みかん」と「マウス」に低次元を持つベクトルを考える。

$$\text{みかん} = \begin{bmatrix} 0.356 \\ 0.246 \\ -0.224 \\ -0.105 \\ 0.542 \end{bmatrix} \qquad \text{マウス} = \begin{bmatrix} 0.342 \\ -0.143 \\ 0.425 \\ -0.382 \\ -0.152 \end{bmatrix}$$

この操作を「埋め込み」を呼ぶ。埋め込みは、「マウス」のような複数の意味を持つ単語 (ネズミとコンピュータの入力機器) も一つのベクトルとして表現される。

2.5 Transformer[4]

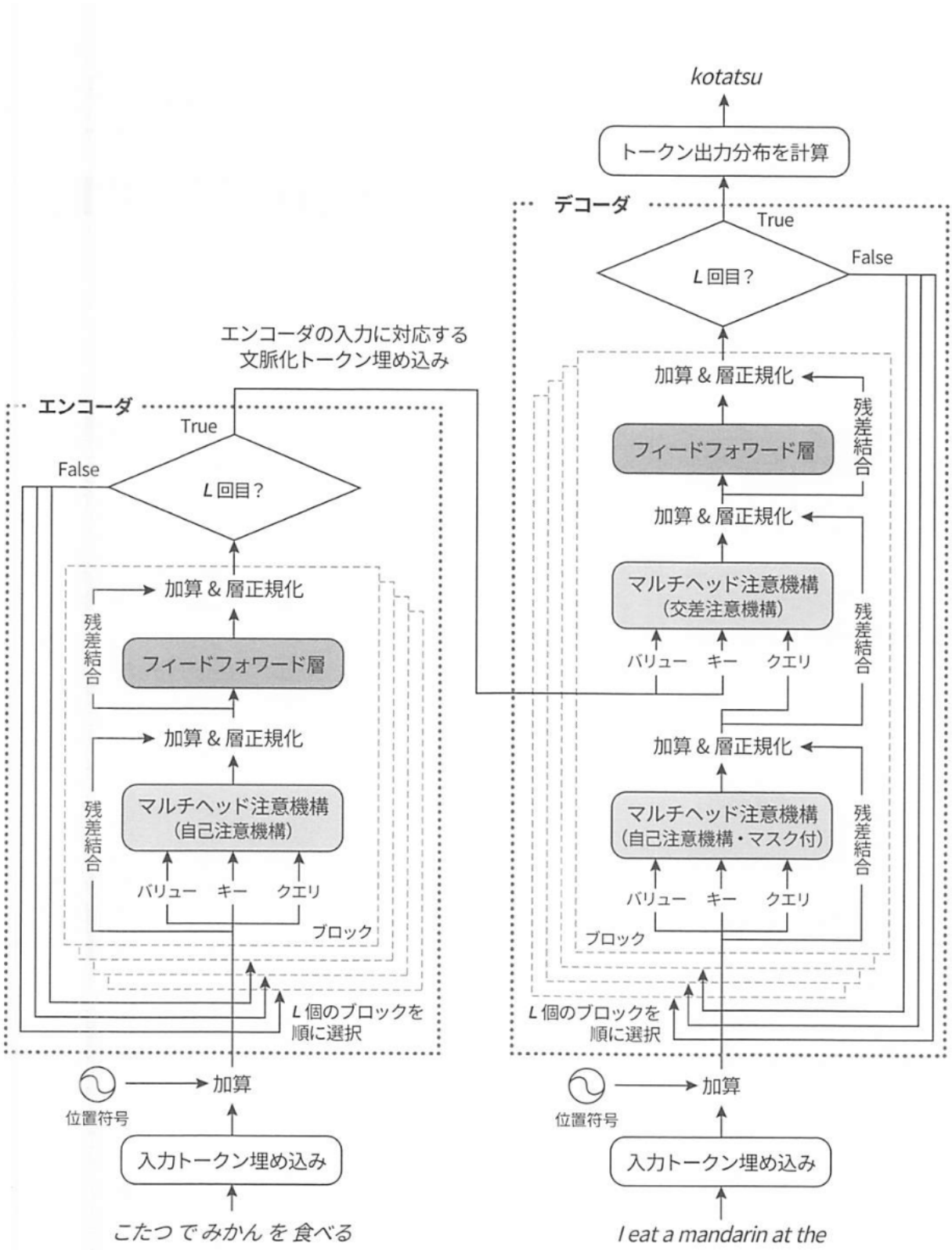

図 2.1: Transformer エンコーダ・デコーダの構造

図 2.3: Transformer エンコーダ・デコーダの構造

Transformer は、自然言語処理の分野で使用される深層学習のモデルである。元々は機械翻訳のモデルとして提案されていたが、2023 年時点では自然言語処理において、文章生成、文章分類、音声認識など幅広いタスクで標準的に利用されるニューラルネットワークになっている。

本研究で用いる BERT(3 章参照) では Transformer で提案された、Transformer Encoder という自己注意機構 (Attention) を用いたニューラルネットワークを採用し、文章分類タスクをタスクに用いる。

図 2.3 は山田, 鈴木, 山田, 李による大規模言語モデル [3] による、Transformer のエンコーダ・デコーダの構造である。BERT では、Transformer のエンコーダのみ使う。

1. 「入力トークン埋め込み」で入力文章をベクトルに変換
2. 「位置符号」で位置情報を加算
3. 「マルチヘッド注意機構」
4. 「加算＆正規化」
5. 「フィードフォワード層」
6. 「加算＆正規化」

3〜6 を L 回繰り返す

2.5.1 マルチヘッド注意機構 (Multi-Head Attention)

マルチヘッド注意機構は、複数の「自己注意機構」という仕組みを適用している。そのため、まずは自己注意機構について説明する。

自己注意機構 (Attention) は、任意のトークンの重要度を加味しながら入力トークン埋め込みに対し、文脈の情報を付加する仕組みである。例として、文章

$$S_2 = [\text{マウスでクリックする}] \rightarrow [\mathbf{e}_{\text{マウス}}, \mathbf{e}_{\text{で}}, \mathbf{e}_{\text{クリック}}, \mathbf{e}_{\text{する}}] \rightarrow [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4]$$

を考える。S_2 における $\mathbf{x}_1$ =「マウス」の意味が、「動物」か「パソコンの入力機器」かのどちらかを捉えるために、周辺のトークンの情報を x_1 に伝播させる必要がある。この例では、$\mathbf{x}_3$ の「クリック」が「マウス」の意味を「パソコンの入力機器」であることを捉えているために重要だと考えられる。

このように、自己注意機構では、モデル入力埋め込み列を入力として受け取り、それらを相互に作用して新しい列を計算する。

自己注意機構は、3 つの d 次元ベクトルのクエリ $\mathbf{q}_i$、キー $\mathbf{k}_i$、バリュー $\mathbf{v}_i$ という埋め込みで計算される。$\mathbf{q}_i$ は入力のうち「検索をかけたいもの」、$\mathbf{k}_i$ は「検索すべき対象と $\mathbf{q}_i$ の近さを測るためのもの」、$\mathbf{v}_i$ は「k_i に基づいて適切な表現をされるもの」を示す。n 個のトークンで構成される文章を処理することを考える。また、前の層での i 番目の出力 (モデル入力埋め込み) を行ベクトル $x_i (i = 1, 2, ..., n)$ とすると、クエリ埋め込み $\mathbf{q}_i$、キー埋め込み $\mathbf{k}_i$、バリュー埋め込み $\mathbf{v}_i$ は下記のように計算される。

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i \tag{2.23}$$

$$\mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i \tag{2.24}$$

$$\mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i \tag{2.25}$$

ここで $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ はそれぞれの D × D 次元の行列を表す。これらの行列を訓練時に学習することで、重要度を加味した文脈化ができるようになる。

i 番目のトークンから見た j 番目のトークンの関連性スコア $s_{i,j}$ は、内積を用いて

$$\tilde{\alpha}_{i,j} = \frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{D}} \tag{2.26}$$

として表される。

分母 $\sqrt{D}$ は次元 D が増えるのに伴い、内積の値が大きくなることを防ぎ、訓練を安定化させるために導入している。

これらを用いて、自己注意機構内で出力される出力埋め込み o_i は、$s_{i,j}$ をソフトマックス関数で正規化した重み $\tilde{\alpha}_{i,j}$ によるバリュー埋め込み v_j の重み付き平均

$$\alpha_{i,j} = \frac{\exp(\tilde{\alpha}_{i,j})}{\sum_{j'=1}^{N} \exp(\tilde{\alpha}_{i,j'})} \tag{2.27}$$

$$\mathbf{o}_i = \sum_{j=1}^{N} \alpha_{i,j} \mathbf{v}_j \tag{2.28}$$

で表される。

ここで例として $S_2 =$ [マウスでクリックする] に自己注意機構のバリュー埋め込みを求める。

まずモデル入力埋め込みに対応するバリュー埋め込みが計算され、マウス x_1 に対する出力埋め込み $\mathbf{o}_1$ は、バリュー埋め込み $v_{1,2,3,4}$ の重み付き和

$$\mathbf{o}_1 = 0.3\mathbf{v}_1 + 0.2\mathbf{v}_2 + 0.4\mathbf{v}_3 + 0.1\mathbf{v}_4$$

$$\mathbf{o}_{\text{マウス}} = 0.3\mathbf{v}_{\text{マウス}} + 0.2\mathbf{v}_{\text{で}} + 0.4\mathbf{v}_{\text{クリック}} + 0.1\mathbf{v}_{\text{する}}$$

として表現される。

マルチヘッド注意機構に話を戻す。マルチヘッド注意機構は複数の自己注意機構を適用しているので、複数の観点から入力トークン埋め込みに文脈の情報を付加できる。例えば、文章 $S_2 =$ [マウスでクリックする] の「マウス」について、タスクによってはトークンの意味のほかに品詞や係り受けなどの文法的な情報が重要になる場合も考慮しなければならない。

マルチヘッド注意機構では、M 個の自己注意機構を同時に適用する。ここで、M は D の約数である必要がある。前の層での i 番目の出力 x_i に対応する $m \in \{1, 2, \ldots, M\}$ 番目の注意機構の埋め込みは、先程の自己注意機構の元に下記のようになる。

$$\mathbf{q}_i^{(m)} = \mathbf{W}_q^{(m)} \mathbf{x}_i \tag{2.29}$$

$$\mathbf{k}_i^{(m)} = \mathbf{W}_k^{(m)} \mathbf{x}_i \tag{2.30}$$

$$\mathbf{v}_i^{(m)} = \mathbf{W}_v^{(m)} \mathbf{x}_i \tag{2.31}$$

ここで $\mathbf{W}_q^{(m)}, \mathbf{W}_k^{(m)}, \mathbf{W}_v^{(m)}$ は $\frac{D}{M} \times D$ の行列、$\mathbf{q}_i^{(m)}, \mathbf{k}_i^{(m)}, \mathbf{v}_i^{(m)}$ は $\frac{D}{M}$ 次元のベクトルである。

$$\tilde{\alpha}_{i,j}^{(m)} = \frac{\mathbf{q}_i^{(m)\top}\mathbf{k}_j^{(m)}}{\sqrt{\frac{D}{M}}} \tag{2.32}$$

$$\alpha_{i,j}^{(m)} = \frac{\exp(\tilde{\alpha}_{i,j}^{(m)})}{\sum_{j'=1}^{N}\exp(\tilde{\alpha}_{i,j'}^{(m)})} \tag{2.33}$$

$$\mathbf{o}_i^{(m)} = \sum_{j=1}^{N}\alpha_{i,j}^{(m)}\mathbf{v}_j^{(m)} \tag{2.34}$$

マルチヘッド注意機構は、M 個の出力埋め込みを連結して計算される。

$$\mathbf{o}_i = \mathbf{W}_o \begin{bmatrix} \mathbf{o}_i^{(1)} \\ \vdots \\ \mathbf{o}_i^{(M)} \end{bmatrix} \tag{2.35}$$

ここで、$\mathbf{W}_o$ は、$D \times D$ 次元の行列である。

2.5.2 入力トークン埋め込み (Input Embedding)

Transformer では、語彙 V に含まれるすべてのトークンに対し、D 次元のの入力トークン埋め込みを付与する。Transoformer に含まれるすべての入力トークン埋め込みは、各行が個別のトークンに対応する $|V| \times$ D 次元の入力トークン行列を

$$\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_{|V|}]^\top \tag{2.36}$$

で表す。例として、文章 S_1=[こたつでみかんを食べる] を考える。トークン w に対する入力トークン埋め込みを $\mathbf{e}_w$ とすると、文章 S_1 の各トークンに対応する入力トークン埋め込みの列は

$$\mathbf{e}_{\text{こたつ}}, \mathbf{e}_{\text{で}}, \mathbf{e}_{\text{みかん}}, \mathbf{e}_{\text{を}}, \mathbf{e}_{\text{食べる}}$$

となる。

2.5.3 位置符号 (Positional Encoding)

位置符号は、正弦関数を使いトークン列の中でトークンの位置をベクトルで表現する方法のことである。入力トークン埋め込みに対し、トークンの順序や位置に関する情報を考慮するために位置符号 (位置埋め込み) を付加する。トークン列中の位置 i に対応する D 次元の位置符号の埋め込みベクトルを $\mathbf{p}_i$ とすると、下記のようになる。

$$\mathbf{p}_i = \begin{bmatrix} \sin(i) \\ \cos(i) \\ \sin\left(\frac{i}{10000^{\frac{2k}{D}}}\right) \\ \cos\left(\frac{i}{10000^{\frac{2k}{D}}}\right) \\ \vdots \\ \sin\left(\frac{i}{10000^{\frac{(D-2)}{D}}}\right) \\ \cos\left(\frac{i}{10000^{\frac{(D-2)}{D}}}\right) \end{bmatrix} \tag{2.37}$$

このとき、ベクトル $\mathbf{p}_i$ の j 番目の要素 $p_{i,j}$ は、$k \in \{0, 1, \ldots, \frac{D}{2} - 1\}$($D$ は偶数) に対し、

$$p_{i,2k+1} = \sin\left(\frac{i}{10000^{\frac{2k}{D}}}\right) \tag{2.38}$$

$$p_{i,2k+2} = \cos\left(\frac{i}{10000^{\frac{2k}{D}}}\right) \tag{2.39}$$

となる。

位置 i のトークンの w_i の入力トークン埋め込みを $\mathbf{e}_{w_i}$ とすると、モデルへの入力埋め込み x_i は、

$$\mathbf{x}_i = \sqrt{D}\mathbf{e}_{w_i} + \mathbf{p}_i \tag{2.40}$$

となる。この式では、位置符号の L2 ノルム $||\mathbf{p}_i||$ は $\sqrt{\frac{D}{2}}$ であるため、スケールを揃えるために入力トークン埋め込みを $\sqrt{D}$ 倍してると考えられる。

文章 S_1=[こたつでみかんを食べる] の例において、「こたつ」の入力埋め込み $\mathbf{x}_1$ は $\sqrt{D}\mathbf{e}_{\text{こたつ}} + \mathbf{p}_1$、「みかん」の入力埋め込み $\mathbf{x}_3$ は $\sqrt{D}\mathbf{e}_{\text{みかん}} + \mathbf{p}_3$ となる。

2.5.4 フィードフォワード層 (Feed Forward)

フィードフォワード層は、活性化関数を間に挟んだ 2 層のニューラルネットワークのことである [7]。フィードフォワード層への入力ベクトルを $\mathbf{u}_i$ とすると、出力ベクトル $\mathbf{z}_i$ は下記の式で求められる。

$$\mathbf{z}_i = \mathbf{W}_2 f(\mathbf{W}_1\mathbf{u}_i + \mathbf{b}_1) + \mathbf{b}_2 \tag{2.41}$$

$\mathbf{W}_1, \mathbf{W}_2$ は、それぞれ $D_f \times D$ 次元、$D \times D_f$次元の行列 であり、 $\mathbf{b}_1, \mathbf{b}_2$ はそれぞれ D_f, D 次元のベクトル、$f(\cdot)$ は活性化関数である。ここで、フィードフォワード層は、全ての位置の入力ベクトルを使う注意機構とは異なり、入力された位置のベクトルのみに閉じて計算されることに注意する。

大規模言語モデルでは、滑らかで経験的に良い収束性能を発揮するガウス誤差線形ユニット (gaussian error linear unit; GELU) が標準的に用いられている。標準正規分布 $\mathcal{N}(0, 1)$ の累積分布関数を $\Phi(x)$ と置くと、ガウス誤差線型ユニットは,

$$\text{gelu}(x) = x\Phi(x) \tag{2.42}$$

のように計算される。

提案時の Transformer では、入力次元 $D = 512$ に対して、中間層の次元は 4 倍の $D_f = 2048$ が使われている。この結果、フィードフォワード層に含まれえるパラメータ数は、Transformer の全体のパラメータ数の約 $\frac{2}{3}$ を占めている。フィードフォワード層は、文脈に関連する情報をその豊富なパラメータの中に記憶しており、入力された文脈に対して関連する情報を探す役割を果たしていると考えられる [8]。

2.5.5 残差結合 (residual connection)

残差結合は、Transformer の訓練を安定させるための仕組みのことである。図 2.1 より、Transformer のエンコーダでは、各ブロックの二つの層 (マルチヘッド注意機構とフィードフォワー

ド層) にそれぞれ残差結合を適用している。そのため、L 個のブロックを含んだモデルでは、出力が $2L$ 個の残差結合を適用した層を通って計算される。

ここで、$k \in \{1, 2, \ldots, 2L\}$ に対し、k 番目の残差結合を適用する前の元々の層を処理を $\mathcal{F}^{(k)}(\mathbf{X})$、層への入力ベクトル列を行列表記 $\mathbf{X}^{(k)} = [\mathbf{x_1}^{(k)}, \mathbf{x_2}^{(k)}, \ldots, \mathbf{x_N}^{(k)}]^T$ とする。残差結合を適用した層の入力 ($k+1$ 番目の層の入力)$\mathbf{X}^{(k+1)}$ は、元々の層の出力 $\mathcal{F}^{(k)}(\mathbf{X}^{(k)})$ に入力 $\mathbf{X}^{(k)}$ を加算し計算する。

$$\mathbf{X}^{(k+1)} = \mathcal{F}^{(k)}(\mathbf{X}^{(k)}) + \mathbf{X}^{(k)} \tag{2.43}$$

この式から、エンコーダの出力 $\mathbf{X}^{(2L+1)}$ を展開すると、

$$\mathbf{X}^{(2L+1)} = \mathbf{X}^{(1)} + \mathcal{F}^{(1)}(\mathbf{X}^{(1)}) + \mathcal{F}^{(2)}(\mathbf{X}^{(2)}) + \cdots + \mathcal{F}^{(2L)}(\mathbf{X}^{(2L)}) \tag{2.44}$$

となる。

2.5.6 層正規化 (layer normalization)

層正規化は、過剰に大きい値によって訓練が不安定になることを防ぐために、ベクトルの値を正規化する仕組みのことである。まず、層正規化する D 次元の入力ベクトル $\mathbf{x}$ に対し、ベクトルの要素 x_i の平均 $\mu_{\mathbf{x}}$ と標準偏差 $\sigma_{\mathbf{x}}$ を求める。

$$\mu_{\mathbf{x}} = \frac{1}{D}\sum_{i=1}^{D} x_i \tag{2.45}$$

$$\sigma_{\mathbf{x}} = \sqrt{\frac{1}{D}\sum_{i=1}^{D}(x_i - \mu_{\mathbf{x}})^2} \tag{2.46}$$

層正規化関数 $\mathrm{layernorm}(\mathbf{x})$ の k 番目の要素は、

$$\mathrm{layernorm}(\mathbf{x})_k = g_k \frac{(x_k - \mu_{\mathbf{x}})}{(\sigma_{\mathbf{x}} + \epsilon)} + b_k \tag{2.47}$$

となる。

ここで g_k と b_k は、ゲインベクトル $\mathbf{g}$ とバイアスベクトル $\mathbf{b}$ の k 番目の要素である。この二つのベクトルは、層正規化の表現を向上するために導入されている。また ε は、0.000001 のような非常に小さい値が用いられる。

第3章　BERTと感情分析

3.1 BERT (Bidirectional Encoder Representations from Transformers)[3][5]

BERT は、2018 年に google が提案した自然学習において代表的な大規模言語モデルの一つである。Transformer のエンコーダー (enocder) のみを使用し、i 番目のトークンにおける $i-1$ 以降と $i+1$ 以降のトークン列の双方向 (bidirecitonal) から文脈を捉えることが名前の由来となっている。

BERT の論文では、Wikipedia と 7,000 冊の書籍を合わせた大規模なコーパスで訓練 (=事前学習) し、指定されたデータセットで微調整 (=ファインチューニング) することで、自然言語処理における複数のタスクの性能を大幅に改善されることが報告されている [18]。

BERT に入力するデータを作る際は、「特殊トークン」と呼ばれる事前学習とファインチューニングの間の齟齬を少なくするトークンが使用される。

- ［CLS］トークン：入力の開始を表す。
- ［SEP］トークン：入力の区切れ目を表す。
- ［MASK］トークン：先行するトークン列と後続するトークン列の双方の文脈の情報を用いて隠したトークンを予測するために使用する。
- ［UNK］トークン：トークンが語彙に含まれていないことを示す。

例として［こたつでみかんを食べる］から入力を作成する場合、BERT に入力されるテキストは、

$$［CLS］こたつでみかんを食べる［SEP］$$

となる。［こたつでテレビをみる］と合わせてテキストを入力する際は、

$$［CLS］こたつでみかんを食べる［SEP］こたつでテレビをみる［SEP］$$

となる。

BERT の入力は、入力トークン埋め込みと位置埋め込みに加え、それぞれのテキストの範囲を区別しやすくするためのセグメント埋め込みが導入されている。入力トークン長を K と置くと、トークン列 $w_1, w_2, \ldots, w_K$ のトークン w_m が入力テキスト $m \in \{1, 2\}$ に属する時、入力埋め込み $\mathbf{x}_i$ は、トークン埋め込み $\mathbf{e}_{w_i}$、位置埋め込み $\mathbf{p}_i$、セグメント埋め込み $\mathbf{s}_m$ を用いて下記のように計算される。

$$\mathbf{x}_i = \mathbf{e}_{w_i} + \mathbf{p}_i + \mathbf{s}_m \tag{3.1}$$

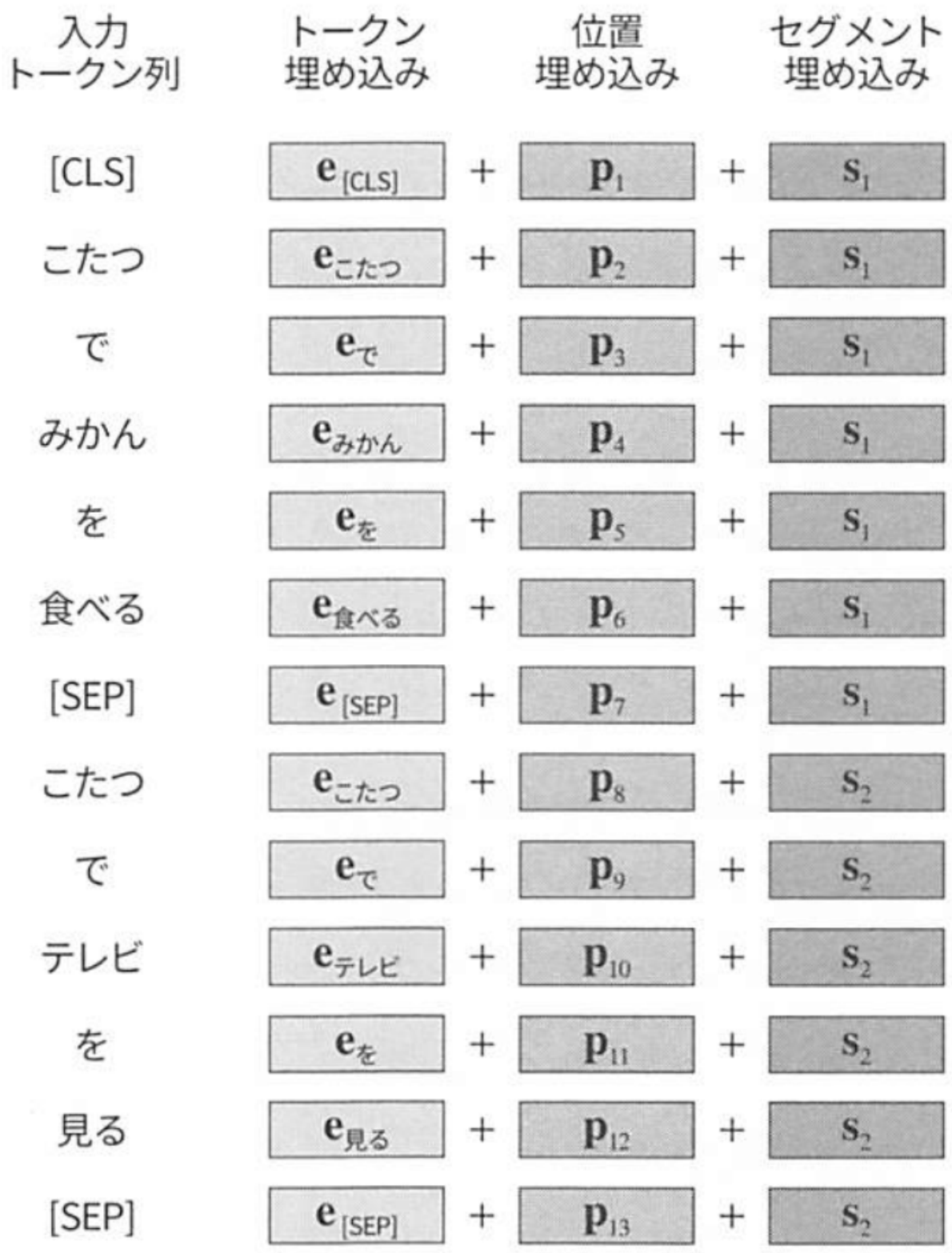

図 3.1: BERT の入力埋め込みの計算

この図は山田, 鈴木, 山田, 李による大規模言語モデル [3] による、BERT の入力埋め込みの計算を表している。

3.2 BERT における事前学習 [4]

word2vec のように実際に解きたいタスクを解く前に、モデルをあらかじめ別のタスクで訓練することを事前学習と呼ぶ。以降、事前学習したモデルを適用する先のタスクを「下流タスク」と呼ぶ。BERT の事前学習は「マスク言語モデリング」と「次文予測」の二つのタスクで構成される。

3.2.1 マスク言語モデリング (masked language modeling; MLM)

マスク言語モデリングは、トークンの穴埋めを行うタスクのことである。BERT では、トークン列中のランダムなトークンを隠して、先行するトークン列と後続するトークン列の双方の文脈情報を用いて隠したトークンを予測することで、双方向から文脈を捉える訓練を実現する。このタスクでは、下記のルールに従ってモデルに入力される。

1. テキスト中の 15 %のトークンを選択し、そのトークン中の 80 %を [mask] トークンに置換
2. 残りの 20 %のうち、10 %は語彙に含まれるランダムなトークンに置換
3. 残りの 10 %は置換せず、元のトークンをそのまま入力する

ランダムなトークンを置換したり、元のトークンをそのままにする理由は、事前学習タスクと下流タスクの間の齟齬を少なくためとされている。BERT の論文では、すべてのトークンを [mask] に置き換えた時と比べ、上記の変換を行った場合の方が能力が改善すると報告されている。

BERT にトークン列 $w_1, w_2, \ldots, w_K$ を与えたとき、トークン w_i の予測確率 P は位置 i の出力埋め込み $\mathbf{h}_i$ を用いて下記のように計算される。

$$\hat{h_i} = \mathrm{layernorm}(\mathrm{gelu}(\mathbf{W}_{\mathrm{mlm}}\mathbf{h}_i)) \tag{3.2}$$

$$p(w_i \mid w_1, w_2, \ldots, w_{k-1}, w_{k+1}, \ldots, w_K) = \mathrm{softmax}_{w_i}(\mathbf{E}\hat{h_i} + \mathbf{b}) \tag{3.3}$$

ここで,$\mathbf{W}_{\mathrm{mlm}}$ は $D \times D$ の行列、layernorm(・) は層正規化、gelu(・) はガウス誤差線形ユニット、$\mathbf{E}$ は入力トークン埋め込み行列、$\mathbf{b}$ はバイアスを表すベクトルである。
マスク言語モデリングの損失関数は、予測の対象として選択されたトークンについて、確率 P の負の対数尤度として計算される。

3.2.2 次文予測 (next sentence prediction; nsp)

次文予測は、入力された二つのテキストが一つの文章の中の連続したテキストかどうかを判断する 2 値分類のタスクのことである。

次文予測では、D 次元の [CLS] トークンの出力トークン埋め込み $\mathbf{h}_{\mathrm{cls}}$ から $\mathbf{h}_{\mathrm{pool}}$ を下記のように計算する。

$$\mathbf{h}_{\mathrm{pool}} = \tanh(\mathbf{W}_{\mathrm{pool}}\mathbf{h}_{\mathrm{hls}}) \tag{3.4}$$

ここで $\mathbf{W}_{\mathrm{pool}}$ は $D \times D$ の行列で、活性化関数として双曲線正接関数 $\tanh(x)$ が用いられている。

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{3.5}$$

分類結果 $y \in \{\text{true, false}\}$ は、$\mathbf{h}_{\text{pool}}$ を使って下記のように予測される。

$$P(y|w_1, w_2, \ldots, w_K) = \text{softmax}_y(\mathbf{W}_{\text{nsp}}\mathbf{h}_{\text{pool}}) \tag{3.6}$$

ここで $\mathbf{W}_{\text{pool}}$ は $2 \times D$ の行列である。

次元予測の損失関数は、予測の対象として選択されたトークンについて、確率 P の負の対数尤度として計算される。

次文予測タスクでは、入力トークン全体の情報を $\mathbf{h}_{pool}$ に集約して、入力を構成する二つのテキストが同一の文書から取り出されたかを予測していると捉えられる。

実際の入力は文単位ではなく、ほとんどが複数の文を含むテキストで構成されている。

BERT の事前学習では入力テキストに対し、マスク言語モデリングと次文予測の双方を同時に適用し、損失関数はこの二つのタスクの損失関数を加算したものが使われる。

3.3 BERT におけるファインチューニング (Fine-tuning)[4]

ファインチューニングは、事前学習されたモデルを下流タスクのデータセットで微調整することである。

トークン列 $w_1, w_2, \ldots, w_K$ と正解ラベル $y \in Y$ が与えられた時、予測確率は [CLS] トークンの出力埋め込み $\mathbf{h}_{cls}$ と $|Y| \times D$ の行列 $\mathbf{W}_{ft}$ を用いて下記のように計算される。

$$P(y|w_1, w_2, \ldots, w_K) = \text{softmax}_y(\mathbf{W}_{\text{ft}}\mathbf{h}_{\text{cls}}) \tag{3.7}$$

訓練は下記の負の対数尤度を最小化することで行う。

$$L(\Theta) = -\sum_{((w_1,w_2,\ldots,w_K),y)\in D} \log P(y|w_1, w_2, \ldots, w_K, \Theta) \tag{3.8}$$

ここで Θ はモデルに含まれるすべてのパラメータを示す。

3.4 感情分析

感情分析とは、文章に含まれる感情を推定するタスクのことである。 本論文ではポジティブ、ネガティブの二種類の感情を扱うものとする。

第4章　Optuna

4.1　Optuna[10][11]

Optuna は、2018 年に株式会社 Preferred Networks によって開発された、ブラックボックス最適化のためのツールであり、機械学習におけるハイパーパラメータチューニングを自動的に行うためのフレームワークである。内部の処理ではベイズ最適化を行なっている。元々は社内用として開発されたが、現在では毎日約 2 万回ダウンロードされるフレームワークに成長した。

4.1.1　Oputuna を使用した背景

機械学習の技術によって、多くの仕事が自動化されるなかで、依然として手動で決めなければならない設定事項 (ハイパーパラメータ) が多く存在する。ハイパーパラメータは時折、機械学習モデルの性能を大きく左右するため、その調整は機械学習の実用に重要である。しかし一般に実験とその性能の評価のサイクルを何度も繰り返す必要があるため、ハイパーパラメータの調整には膨大な時間と手間がかかる。

ブラックボックス最適化はハイパーパラメータ調整のサイクルを自動化・効率化することを表しており、性能を改善するようなハイパーパラメータを自動で探すのがブラックボックス最適化の仕事である。Optuna は Python 言語で記述されたブラックボックス最適化のためのツールであり、多目的最適化や分散並列最適化などの豊富な機能を提供している。またグリッドサーチやランダムサーチに比べ、より効率的な SMBO(sequential model-based optimization)[14] と呼ばれるアプローチを実装している。これらの理由から Optuna を用いてハイパーパラメータを最適化することを決めた。

4.1.2　Oputuna の概要

ブラックボックス最適化の枠組みでは、ハイパーパラメータと機械学習のモデルの性能の関係を一つの関数に見立て、ハイパーパラメータを関数への入力、性能を関数からの出力とする。また、最適化する対象を目的関数として抽象化し、それを通じてハイパーパラメータを最適化する。ここで目的関数は入力としてハイパーパラメータを受け取り、出力として評価値を返す関数である。

本論文では、深層学習の使用時に精度を最大化させるために、3 つのハイパーパラメータ (学習率、バッチサイズ、エポック数) を調整したいと考えている。ここで、3 つのハイパーパラメータを入力として受け取り、実際に深層学習を行い、計測された精度の数値を結果として出力する一連のプロセスが目的関数となる。

ブラックボックス最適化では、関数の評価を何度も繰り返しながら優れた入力を探索する。Optuna では、目的関数の 1 回の評価をトライアルと呼び、トライアルを繰り返す一連の最適化プロセスをスタディと呼ぶ。また、トライアルによって提案されたすべてのハイパーパラメータを含んだ集合を探索空間と呼び、探索空間内の 1 点が選択されて次回の目的関数への入力 (結果) として返される処理のことを探索点選択と呼ぶ。

4.1.3 ベイズ最適化

ベイズ最適化は、有望なパラメータの推定に確率モデルを仮定してベイズ的な取り扱いをする手法のことである。ベイズ最適化はOptunaを含む探索点選択において共通の枠組みとなっている。

有界な集合 $D(\subset \mathbb{R}^d)$ を定義域とする変数 x について、最小化すべき目的関数を f とおく ($\mathbb{R}^d$ は d 次元の実数全体の集合)。

ここで、t 番目のトライアルにおいて探索点として $x_t \subset D$ を選ぶ時、これを目的関数 f に与えた評価値 y_t は下記のように計算される。

$$y_t = f(x_t) + \epsilon \tag{4.1}$$

ϵ は正規分布に従うノイズ $\epsilon\, \mathcal{N}(0, \sigma^2)$ を表す。

今、t 番目のトライアルにおいて、$(t-1)$ 番目までのトライアルが終了していると仮定すると、それまでの探索点と目的関数の評価値のペアの系列 $\mathcal{H}_t$ を下記のようにおく。

$$\mathcal{H}_t = \{(x_i, y_i)\}_{i=1}^{t-1} \tag{4.2}$$

以降 $\mathcal{H}_t$ を履歴と呼ぶ。次項で説明するTPEと呼ばれる手法では、目的関数 f や、その目的関数の評価値 y が従う確率的なモデルを仮定し、それを用いてある基準を定め、各トライアルで探索点を選択する。この探索点を選択するための基準を獲得関数といい、獲得関数 α_t は仮定した確率的なモデルと履歴 $\mathcal{H}_t$ によって定まる D 上の実数値関数として、下記のように定式化される。

$$\alpha_t : D \to \mathbb{R} \tag{4.3}$$

各トライアルでは獲得関数 α_t を最大化するするように x_t を選ぶ。

$$x_t = \arg\max \alpha_t(x) \tag{4.4}$$

このように、x_t を逐次的に選ぶことがベイズ最適化の枠組みとなっている。

4.1.4 TPE(tree-structured Parzen estimator)

TPEは、ベイズ最適化の一種でありOptunaのデフォルトの探索点選択アルゴリズムである。単変量TPEでは d 次元の変数

$$x = (x^{(1)}, x^{(2)}, \ldots, x^{(d)}) \in D \tag{4.5}$$

についての最適化を行い、探索点の各次元 $x^{(i)} \in \mathbb{R}$ と目的関数の評価値 $y \in \mathbb{R}$ の間の確率分布を仮定し、次元ごとに定めた確率的なモデルを利用して、次元ごとに獲得関数を定める。また、$x \in D$ と y の関係ではなく、各次元ごとの $x^{(i)}$ と y の関係であることに注意する。具体的には、$x^{(i)}$ と y の間の確率分布を下記のようにモデル化する。

$$p(x^{(i)}|y, \mathcal{H}_t) = \begin{cases} l(x^{(i)}|\mathcal{L}_t) & (x \le y^*) \\ g(x^{(i)}|\mathcal{G}_t) & (\text{if })x \ge y^*) \end{cases} \tag{4.6}$$

$$\mathcal{L}_t = \{x_j | (x_j, y_j) \in H_t, (y_j < y^*)\} \tag{4.7}$$

$$\mathcal{G}_t = \{x_j | (x_j, y_j) \in H_t, (y_j \ge y^*)s\} \tag{4.8}$$

y^* はあらかじめ定められた定数 $\gamma(= p(y < y^* \mid \mathcal{H}_t))$ を満たすように選ばれた閾値とする。また、$p(y)$ の具体的なモデルは何も定めないこととする。
$l(x|\mathcal{L}_t) : \mathbb{R} \to \mathbb{R}$ と $g(x|\mathcal{G}_t) : \mathbb{R} \to \mathbb{R}$ はともに 1 次元実数値関数であり、$x^{(i)}$ の型に応じて定義される。

入力空間 $D \in \mathbb{R}^d$ の各次元 $x^{(i)}$ がカテゴリカルな変数の場合、w_0 を定数、$x^{(i)}$ の取りうる値をの集合を $C = \{C_1, C_2, \ldots, C_K\}$ とする。

ここで、各 $x^{(i)} \in \mathcal{L}_t$ について、その第 i 成分が $C_K \in C$ に等しいという事象の指示関数の値に w_0 を加えたものを $N_{j,k}$ とする。すなわち、

$$N_{j,k}^{(i)} = 1_{\{x^{(i)}=C_k\}}(x_j) + w_0 \tag{4.9}$$

とする。この時、

$$N_j^{(i)} = \sum_{k=1}^{K} N_{j,k}^{(i)} c_{j,k}^{(i)} = \frac{N_{j,k}^{(i)}}{N_j^{(i)}} = \frac{1_{\{x^{(i)}=C_k\}}(x_j) + w_0}{\sum_{k=1}^{K}(1_{\{x^{(i)}=C_k\}}(x_j) + w_0)} \tag{4.10}$$

とおいて、$l(x^{(i)}|\mathcal{L}_t)(g(x^{(i)}|\mathcal{G}_t))$ を下記のように定める。

$$l(x^{(i)} = C_k|\mathcal{L}_t) = \sum_{j=1}^{|\mathcal{L}_t|} w_j c_{j,k}^{(i)} \tag{4.11}$$

w_j は重みであり、デフォルトでは $n =\mid \mathcal{L} \mid$ として

$$w(n) = \begin{cases} \text{全要素が 1 の n 次元ベクトルを正規化したもの } (n < 25) \\ \frac{1}{n} \text{から 1 までを (n-25) 等分したベクトルと全要素が 1 の} \\ \text{25 次元ベクトルをつなげた n 次元ベクトルを正規化したもの (otherwise)} \end{cases} \tag{4.12}$$

と設定されている。

4.1.5 期待改善量 EI(expected improvement)

単変量 TPE では獲得関数として期待改善量と呼ばれる式を用いる。

$$EI(x|H_t) = \int_{-\infty}^{y^*} \max(y^* - y, 0) p(y|x, H_t) dy \tag{4.13}$$

この式を積分しやすい形に変形する。

$$= \int_{-\infty}^{y^*} (y^* - y) p(y|x, H_t) dy$$

$$= \int_{-\infty}^{y^*} (y^* - y) \frac{p(x|y, H_t) p(y|H_t)}{p(x|H_t)} dy$$

$$= \int_{-\infty}^{y^*} (y^* - y) \frac{l(x|L_t) p(y|H_t)}{p(x|H_t)} dy$$

$$= \frac{l(x|L_t)}{p(x|H_t)} \int_{-\infty}^{y^*} (y^* - y) p(y|H_t) dy$$

この時、積分 $\int_{-\infty}^{y^*}(y^* - y)p(y|H_t)dy$ は $x^{(i)}$ に依存しない正の定数なので、獲得関数の最適化を考える時は無視できる。また、$p(x|H_t)$ を下記のように変形する。

$$p(x|H_t) = p(y < y^*|H_t)p(x|y < y^*, H_t) + p(y \geq y^*|H_t)p(x|y \geq y^*, H_t)$$

$$= \gamma l(x|L_t) + (1 - \gamma)g(x|G_t) \quad (\because \gamma = p(y < y^* \mid \mathcal{H}_t)) \tag{4.14}$$

したがって、

$$\frac{l(x|L_t)}{p(x|H_t)} = \frac{l(x|L_t)}{\gamma l(x|L_t) + (1 - \gamma)g(x|G_t)} = \frac{1}{\gamma + (1 - \gamma)\frac{g(x|G_t)}{l(x|L_t)}} \tag{4.15}$$

この時 $0 < \gamma < 1$ に注意すると、

$$\frac{1}{\gamma + (1 - \gamma)\frac{g(x|G_t)}{l(x|L_t)}}$$

を $x^{(i)}$ について最大化することは、

$$\frac{g(x|G_t)}{l(x|L_t)}$$

を $x^{(i)}$ について最小化することと等価である。

ゆえに、獲得関数の式 $EI(x|H_t)$ の最大化問題は、次式の最大化問題と等価である。

$$\alpha_t(x) = \frac{l(x|L_t)}{g(x|G_t)} \tag{4.16}$$

第5章　実験

5.1　はじめに

本章では BERT を用いて感情分析を行い、Optuna を使用して最適化されたハイパーパラメータで学習したニューラルネットワークの精度と Optuna を使用していない場合の精度を比較実験と考察を行う。本研究の究極的な目的は、BERT による感情分析の精度をさらに向上させるための有効な手法を探求し、従来の手法との比較を通じてその優位性を明らかにすることである。

5.1.1　村田 [1] の実験

村田 [1] の「BERT による感情分析」の論文 [1] から引用する。

この論文では株式会社リクルートの提供している Japanese Realistic Textual En- tailment Corpus[13] というデータセット (5.2.1 項参照) を用いて BERT を学習させ、機械学習を行わない感情極性辞書 [19] での手法と BERT での手法で感情分析を比較実験を行なっている。 ポジティブ、ネガティブの二種類の感情をラベルとして実験を行い、結果は BERT の精度が 96 %、感情極性辞書の精度が 80 %となり、BERT の手法の方が 16 %高いことがわかった。

この結果から、Japanese Realistic Textual Entailment Corpus に対して感情極性辞書による手法よりも BERT による手法が有効であることが分かった。

5.1.2　本研究の実験

本研究では [1] による感情分析の研究成果を基に、さらなる精度の向上を目指す。具体的には BERT を用いた感情分析において、ハイパーパラメータの最適化に Optuna を活用し、その効果を検証する。[1] の研究では、BERT を用いて感情分析を行い、その精度が感情極性辞書を用いた従来手法を大きく上回ることが示された。しかし、ハイパーパラメータの選択がその精度に大きな影響を及ぼす可能性があるため、本研究では Optuna を用いた最適化プロセスを導入し、その精度向上の可能性を探求する。

この実験では、まず Optuna を使用していない状態での BERT モデルの学習を行い、その結果をベースラインとして設定する。次に、Optuna を用いてハイパーパラメータを最適化したモデルを同様のデータセットで学習させ、両者の精度を比較することで、Optuna の最適化がモデルの性能に与える影響を定量的に評価する。このプロセスを通じて、BERT による感情分析の精度をさらに向上させるための有効な手法を探求し、従来手法との比較を通じてその優位性を明らかにすることを目指す。

5.2　実験の方法

この実験は、BERT を用いた感情分析の精度を Optuna を使用して最適化されたハイパーパラメータを用いた場合とそうでない場合の精度を比較し、Optuna の最適化がモデル性能に与

える影響を定量的に評価することが目標である。データセットの一つの文章から、ユーザーがその宿泊施設にどのような感想を持ったか、ポジティブ、ネガティブの 2 択、あるいはポジティブ、ネガティブ、ニュートラルの 3 択で判定することをタスクとし、そのタスクを BERT が遂行できるように学習させた上でテストを行ない、精度を確かめる。

具体的には、以下のステップで実験を進める。

- まず、Optuna を使用せずに BERT モデルを学習させ、その精度をベースラインとして設定する。ここでは、[1] の論文で選択されたハイパーパラメータを使用してモデルを学習させた時の精度をベースラインとして、感情分析のタスクにおける基本的な性能を測定する。

- 次に、Optuna を用いてハイパーパラメータの最適化を行う。Optuna は、最も効果的なハイパーパラメータの組み合わせを自動で見つけ出すツールである。このプロセスを通じて、モデルの重要なハイパーパラメータを最適化し、最終的なモデルの性能を向上させることを目指す。

- 最後に、最適化されたハイパーパラメータを用いて学習させたモデルと、ベースラインのモデルの精度を比較する。この比較を通じて、Optuna によるハイパーパラメータの最適化がモデルの性能にどの程度影響を与えるかを定量的に評価する。

5.2.1 データセット

pn17q05958	1	夕食も朝食も個室でいただけたので、家族	{"1": 3}	train
pn17q05959	1	10年ぶりに宿泊いたしましたが、ご担当	{"1": 3}	train
pn17q05960	-1	気になる点として、BSが映らない(テレビ	{"-1": 3}	train
pn17q05962	1	ホテル目の前で、車の外まで漂う硫黄臭に	{"0": 1, "1": 2}	dev
pn17q05963	1	朝食は毎日少しずつ変わるメニューでした	{"1": 3}	train
pn17q05964	-1	スクランブルエッグにとても大きな殻がス	{"-1": 3}	train
pn17q05965	-1	何組か食事してましたが、明らかにレスト	{"-1": 2, "0": 1}	train
pn17q05967	0	静かで居心地も良いしまた行きたい宿にな	{"-1": 1, "0": 2}	train
pn17q05968	-1	チェックインして、部屋に入るとベッドル	{"-1": 3}	train
pn17q05970	0	朝食はサービスなのでこんなものかな...	{"-1": 1, "0": 2}	dev

図 5.1: Japanese Realistic Textual Entailment Corpus

データセットは、株式会社リクルートの提供している Japanese Realistic Textual En- tailment Corpus[13, 20] を用いた。このデータセットは旅行情報サイト「じゃらん net」が元にして作られたコーパスである。

それぞれのカラムは以下を表す。

0. ID	1. ラベル	2. テキスト	3. 評価者	4. 用途
pnXYZq00001	1, 0, -1	駅まで近い。	"0": 1, "1": 2	train, dev, test

このデータセットは、じゃらんのクチコミデータから文を抽出し、それらを加工した文とアノテーション作業者が付与した判定ラベルを含んでいる。

ID は、様々な理由で一部のデータが削除されているため順番通りではない。

ラベルは、1 がポジティブ、0 がニュートラル、-1 がネガティブを表す。

評価者は、テキストが「1」「0」「−1」のうち、どのキーに当てはまるかを判断し、3 名の多数決でラベルが決まる。

用途は、train が訓練データ、dev が開発 (検証) データ、test がテストデータを表す。

このデータセットの総数は 5553 件で構成されている。そのうち訓練データは 3338 件、開発データは 1112 件、テストデータは 553 件である。また、ラベル 0 を抜いたデータ数 (ニュートラルを抜いた、ポジティブとネガティブのデータ数) は、全体が 4224 件、訓練データが 2959 件、開発データが 851 件、テストデータが 414 件となっている。

5.2.2 ハイパーパラメータの設定

ハイパーパラメータは重みやバイアスといった学習するパラメータとは異なり、変更されず固定されたままのパラメータのことをいう。ハイパーパラメータには層の数や各層のニューロン数など複数あるが、この章の実験では「学習率」「バッチサイズ」「エポック数」の三つをハイパーパラメータとする。

5.3 実験の概要

データセットの一つの文章から、ユーザーがその宿泊施設にどのような感想を持ったか、ポジティブ、ネガティブの 2 択、あるいはポジティブ、ネガティブ、ニュートラルの 3 択で判定することをタスクとした。この章の実験ではポジティブ、ネガティブの 2 択のみで実験を行う。モデルには東北大乾研が日本語 Wikipedia により事前学習を行なったモデル [15] を扱える、huggingface 社が提供している BertForSequenseClasification[16] を使用した。このモデルは BERT-base である。このモデルにおけるニューロンの数は 768 である。また、クラス分類は BERT の出力層の [CLS] の特徴量ベクトル (1 × 768) に対して線形変換、活性化関数、線形変換を順に行いラベルの数 N と同じ次元を持つベクトル (1 × N) を出力する。

実験は下記の二つの内容で行なう。

実験 1

設定

データセット中のニュートラルを除いた、データセットの総数 4224 件を使用し、Optuna を用いた最適化されたハイパーパラメータと用いていない場合のハイパーパラメータ (村田 [1] の論文で選択されたハイパーパラメータ) を比較する。

目的

ポジティブとネガティブのデータセットを用いることで、Optuna によるハイパーパラメータの最適化がモデルの性能に与える影響を評価する。

実験 2

設定

4224 件のうち、訓練データ数を 30 %に制限した、データセットの総数 1556 件を使用し、Optuna を用いた最適化されたハイパーパラメータと用いていない場合のハイパーパラメータを比較する。

目的

訓練データの量を意図的に制限し、データが限られている状況での Optuna によるハイパーパラメータの最適化がモデルの性能に与える影響を評価する。

以降、Optuna を用いた最適化されたハイパーパラメータをデータ A、用いていない場合のハイパーパラメータをデータ B とする。

5.4 実験

この実験の目的は、BERT を用いて感情分析を行い、Optuna を用いた最適化されたハイパーパラメータ (データ A) とそうでない場合のハイパーパラメータ (データ B) の精度を比較を検証することである。

データセットをポジティブ、ネガティブの 2 種類のラベルを限定して、BERT で学習と推論を行なった。精度は下記の式で計算される。

$$\text{精度} = \frac{\text{推論で正解したテストデータの数（正解数）}}{\text{テストデータの総数}} \tag{5.1}$$

5.4.1 実験 1 の概要

実験 1 の目的は、ポジティブとネガティブのデータセットを用いることで、Optuna によるハイパーパラメータの最適化がモデルの性能に与える影響を評価することである。実験 1 ではデータセットの総数を 4224 件、訓練データを 2959 件、開発データを 851 件、テストデータを 414 件として推論を行い、精度を比較した。

Optuna を使用したデータ A は次式のように計算された。

- 学習率：3.744×10^{-5}
- バッチサイズ：16
- エポック数：2

Optuna を使用していないデータ B は次式のように置いた。

- 学習率：1.0×10^{-6}
- バッチサイズ：32
- エポック数：10

5.4.2 実験 1 の結果

実験 1 のデータ A とデータ B の結果は下記の図の通りになった。

グラフの横軸はステップ数、縦軸は損失関数の値である。それぞれの図の左の曲線は開発データに対する損失の値の時間変化、右のグラフは訓練データに対する損失の値の時間変化である。ステップ数は処理したミニバッチの数のことである。また、開発データは訓練データで学習した BERT が過学習しているかどうかを確認するために使うデータである。

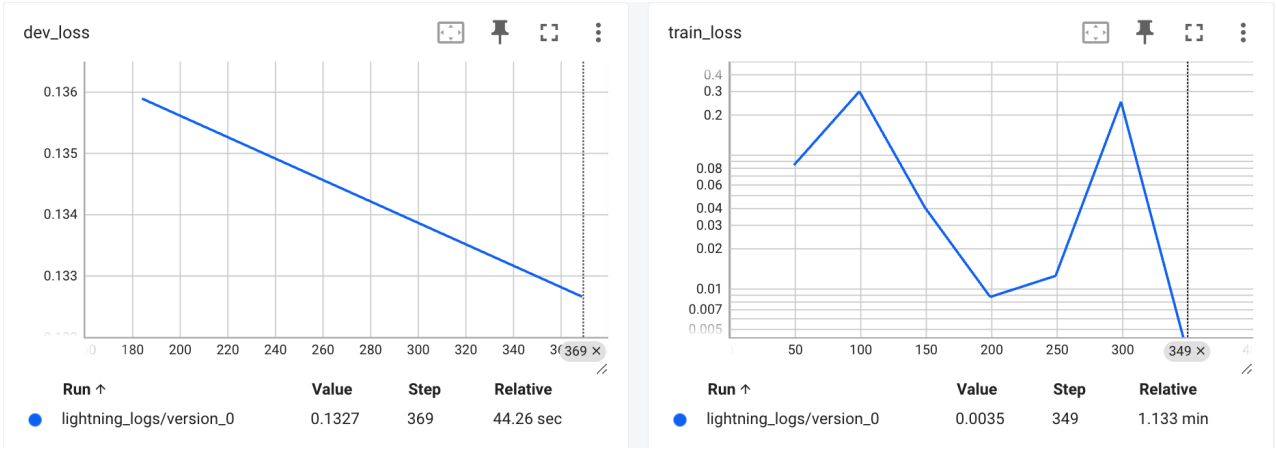

図 5.2: 実験 1 のデータ A の訓練データと検証データの学習曲線

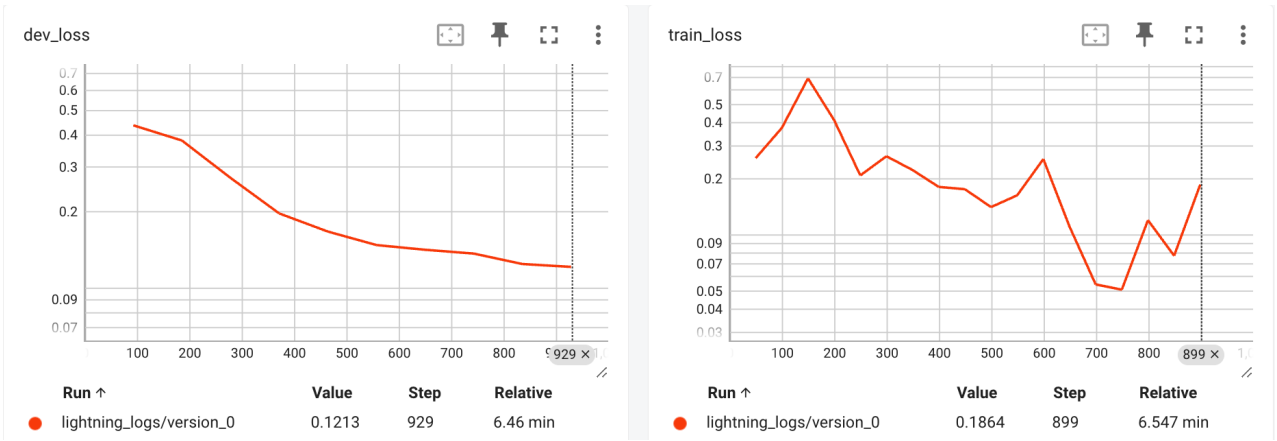

図 5.3: 実験 1 のデータ B の訓練データと検証データの学習曲線

データ A の精度は 97.3 %であり、データ B の精度は 96.4 %であった。

5.4.3 実験 2 の概要

実験 2 の目的は、訓練データの量を意図的に制限し、データが限られている状況での Optuna によるハイパーパラメータの最適化がモデルの性能に与える影響を評価することである。実験 2 ではデータセットの総数を 1556 件、訓練データを 887 件、開発データを 255 件、テストデータを 414 件として推論を行い、精度を比較した。

Optuna を使用したデータ A は次式のように計算された。

- 学習率：1.608×10^{-5}
- バッチサイズ：32
- エポック数：2

Optuna を使用していないデータ B は次式のように置いた。

- 学習率：1.0×10^{-6}
- バッチサイズ：32
- エポック数：10

5.4.4　実験 2 の結果

実験 2 のデータ A とデータ B の結果は下記の図の通りになった。

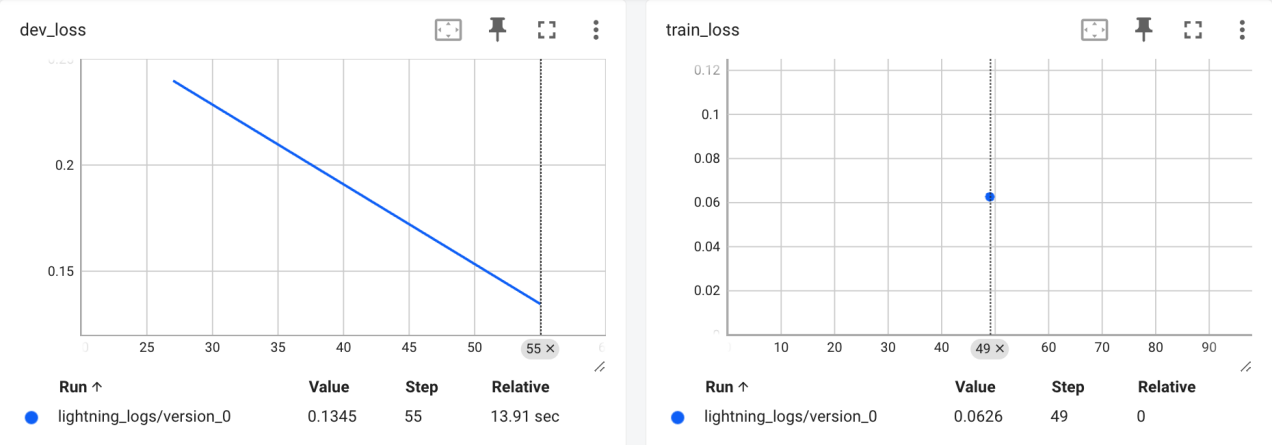

図 5.4: 実験 2 のデータ A の訓練データと検証データの学習曲線

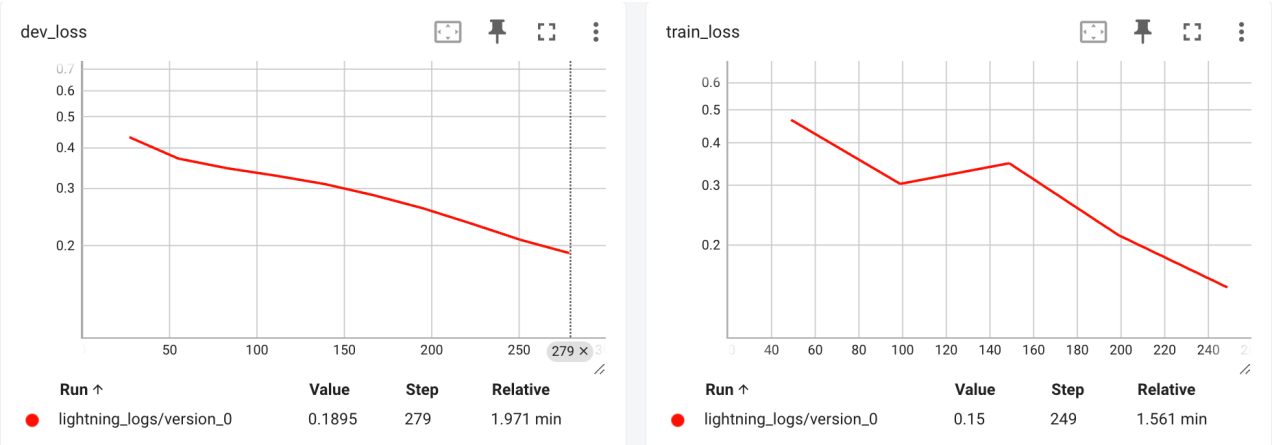

図 5.5: 実験 2 のデータ B の訓練データと検証データの学習曲線

データ A の精度は 93.2 %であり、データ B の精度は 85.7 %であった。

5.5　実験の考察

実験 1 の結果から Optuna を使用したデータ A の精度は、使用していないデータ B の精度よりも約 1 %高いことがわかった。また、実験 2 の結果からデータ A の精度は、データ B の精度よりも約 8 %高いことがわかった。このことから Optuna で求めたハイパーパラメータを設定した BERT の感情分析は、自身で設定するハイパーパラメータよりも精度が上回ると考えられる。さらに実験 1,2 より学習率が最も最適化されていることから、人手で設定するのは困難な学習率を Optuna で求める必要性は高いと考えられる。

第6章　結論と今後の課題

6.1　結論

本研究の目的は、BERT による感情分析の精度をさらに向上させるための有効な手法を探求し、従来の手法との比較を通じてその優位性を明らかにすることであった。データセットの総数 4224 件の実験 1 では、Optuna を使用したデータ A の精度は 97.3 %であり、使用していないデータ B の精度 96.4 %より高いことがわかった。データセットの総数 1556 件の実験 2 では、Optuna を使用したデータ A の精度は 93.2 %であり、使用していないデータ B の精度 85.7 %より高いことがわかった。このことから Optuna を用いたハイパーパラメータの最適化が、BERT による感情分析の精度を向上させる有効な手法であることが確認できた。

6.2　今後の課題

本研究では、BERT というモデルを用いた感情分析というタスクについてのハイパーパラメータの最適化を行ったが、GPT や T5 などの他のモデル、機械翻訳や文章生成などの他のタスクでも Optuna はハイパーパラメータを最適化できるのか実験したい。また、Optuna の最適化の手法として TPE を使用したが、ガウス過程のような他の手法を用いた実験も行いたい。

付 録A　プログラム

```
# 必要なライブラリのインストール
!pip install transformers fugashi ipadic pytorch-lightning pandas

# 標準ライブラリおよびサードパーティライブラリのインポート
import random
import glob
from tqdm import tqdm
import os
# os.environ['CUDA_LAUNCH_BLOCKING'] = "1"

import pandas as pd
import torch
from torch.utils.data import Dataset, random_split, DataLoader,
    TensorDataset
from transformers import BertJapaneseTokenizer,
    BertForSequenceClassification
from transformers import AdamW
from pytorch_lightning import Trainer
from pytorch_lightning.callbacks import ModelCheckpoint
import pytorch_lightning as pl

# 使用するモデル名の設定
MODEL_NAME = 'cl-tohoku/bert-base-japanese-whole-word-masking'
# 日本語用の BERT トークナイザーの初期化
tokenizer = BertJapaneseTokenizer.from_pretrained(MODEL_NAME)
# シーケンス分類用の BERT モデルの初期化（ 2 つのラベル）
bert_sc = BertForSequenceClassification.from_pretrained(MODEL_NAME,
    num_labels=2)
# モデルを GPU に移動
bert_sc = bert_sc.cuda()

# Google Colab で Git リポジトリをクローンする
!git clone https://github.com/megagonlabs/jrte-corpus
# 正しいパスから TSV ファイルを DataFrame に読み込む
df = pd.read_csv('/content/jrte-corpus/data/pn.tsv', delimiter='\t',
    header=None)
df = df[df.iloc[:, 1] != 0]
# ネガティブなデータのラベルを -1 から 0 に変更
df.loc[df.iloc[:, 1] == -1, 1] = 0

# データセットファイルを読み込み
with open("jrte-corpus/data/pn.tsv", "r", encoding="utf-8") as f:
    lines = f.readlines()
train_df = df[df.iloc[:, 4] == 'train']
dev_df = df[df.iloc[:, 4] == 'dev']
test_df = df[df.iloc[:, 4] == 'test']
```

```
train_sentences = train_df.iloc[:, 2].apply(str).tolist()
dev_sentences = dev_df.iloc[:, 2].apply(str).tolist()
test_sentences = test_df.iloc[:, 2].apply(str).tolist()

train_label_list = train_df.iloc[:, 1].tolist()
dev_label_list = dev_df.iloc[:, 1].tolist()
test_label_list = test_df.iloc[:, 1].tolist()


# 実験 2 の場合このコメントアウトを外す
# split_index_train = int(len(train_sentences) * 0.3)
# train_sentences = train_sentences[:split_index_train]
# train_label_list= train_label_list[:split_index_train]

# # dev データセットの分割
# split_index_dev = int(len(dev_sentences) * 0.3)
# dev_sentences= dev_sentences[:split_index_dev]
# dev_label_list= dev_label_list[:split_index_dev]


class TextDataset(Dataset):
    def __init__(self, texts, labels, tokenizer):
        self.encodings = tokenizer(texts, padding='max_length',
           truncation=True, max_length=256, return_tensors='pt')
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: val[idx] for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

# データセットからデータローダを作成
train_dataset = TextDataset(train_sentences, train_label_list,
   tokenizer)
dev_dataset = TextDataset(dev_sentences, dev_label_list, tokenizer)
test_dataset = TextDataset(test_sentences, test_label_list, tokenizer
   )

dataloader_train = DataLoader(train_dataset, batch_size=32, shuffle=
   True, num_workers=4)
dataloader_dev = DataLoader(dev_dataset, batch_size=32, shuffle=False
   , num_workers=4)
dataloader_test = DataLoader(test_dataset, batch_size=32, shuffle=
   False, num_workers=4)

# Optuna の目的関数
def objective(trial):
    # でチューニングするハイパーパラメータOptuna
    # 学習率
    lr          = trial.suggest_float('lr', 1e-5, 1e-3, log=True)
    # エポック数
    num_epochs = trial.suggest_int('num_epochs', 1, 10)
    # バッチサイズ
```

```
    batch_size = trial.suggest_categorical('batch_size', [1, 32])

    # データローダーの作成
    dataloader_train = DataLoader(train_dataset, batch_size=
        batch_size, shuffle=True, num_workers=4)
    dataloader_dev = DataLoader(dev_dataset, batch_size=batch_size,
        shuffle=False, num_workers=4)


    # PyTorch Lightning モデルのインスタンス化
    model = BertForSequenceClassification_pl(MODEL_NAME, num_labels
        =2, lr=lr)

    # PyTorch Lightning トレーナーの設定
    trainer = pl.Trainer(
        max_epochs=num_epochs,
        callbacks=[pl.callbacks.EarlyStopping(monitor="dev_loss",
            patience=3)],
        # progress_bar_refresh_rate=0  # プログレスバーの非表示
    )

    # トレーニングと検証の実行
    trainer.fit(model, dataloader_train, dataloader_dev)

    # 検証データにおける最終的な損失を返す
    return trainer.callback_metrics["dev_loss"].item()

class BertForSequenceClassification_pl(pl.LightningModule):

    def __init__(self, model_name, num_labels, lr):
        # model_name: のモデルの名前Transformers
        # num_labels: ラベルの数
        # lr: 学習率

        super().__init__()

        # 引数の num_labels とを保存。lr
        # 例えば、self.hparams.lr で lr にアクセスできる。
        # チェックポイント作成時にも自動で保存される。
        self.save_hyperparameters()

        # BERT のロード
        self.bert_sc = BertForSequenceClassification.from_pretrained(
            model_name,
            num_labels=num_labels
        )

    # 学習データのミニバッチ(‘batch‘)が与えられた時に損失を出力する関数を書く。
    # batch_idx はミニバッチの番号であるが今回は使わない。
    def training_step(self, batch, batch_idx):
        output = self.bert_sc(**batch)
        loss = output.loss
        self.log('train_loss', loss) # 損失を'train_lossの名前でログをとる。
        '
        return loss
```

```
    # 検証データのミニバッチが与えられた時に、
    # 検証データを評価する指標を計算する関数を書く。
    def validation_step(self, batch, batch_idx):
        output = self.bert_sc(**batch)
        dev_loss = output.loss
        self.log('dev_loss', dev_loss) # 損失を'val_lossの名前でログをとる。
            '
```

```
    # テストデータのミニバッチが与えられた時に、
    # テストデータを評価する指標を計算する関数を書く。
    def test_step(self, batch, batch_idx):
        labels = batch.pop('labels') # バッチからラベルを取得
        output = self.bert_sc(**batch)
        labels_predicted = output.logits.argmax(-1)
        num_correct = ( labels_predicted == labels ).sum().item()
        accuracy = num_correct/labels.size(0) #精度
        self.log('accuracy', accuracy) # 精度を'accuracyの名前でログをとる。
            '
```

```
    # 学習に用いるオプティマイザを返す関数を書く。
    def configure_optimizers(self):
        return torch.optim.Adam(self.parameters(), lr=self.hparams.lr
            )

   # 学習時にモデルの重みを保存する条件を指定
checkpoint = ModelCheckpoint(
    monitor='dev_loss',
    mode='min',
    save_top_k=1,
    save_weights_only=True,
    dirpath='model/',
)

# チェックポイントのディレクトリを指定
checkpoint_dir = 'model/'
# 最新のチェックポイントファイルを探す
latest_checkpoint = None
if os.path.exists(checkpoint_dir):
    checkpoints = [os.path.join(checkpoint_dir, f) for f in os.
        listdir(checkpoint_dir) if f.endswith('.ckpt')]
    if checkpoints:
        latest_checkpoint = max(checkpoints, key=os.path.getctime)
        print(f"最新のチェックポイントを見つけました:␣{latest_checkpoint}")

# 学習の方法を指定（途中から再開する場合は、を指定）latest_checkpoint
trainer = Trainer(
    max_epochs=10,
    callbacks = [checkpoint],
)

# PyTorch Lightning モデルのロード
model = BertForSequenceClassification_pl(
    MODEL_NAME, num_labels=2, lr=1e-6
)

# ファインチューニングを行う。
trainer.fit(model, dataloader_train, dataloader_dev)
```

```
# ファインチューニング後のファイルパスと損失の値
best_model_path = checkpoint.best_model_path
# printベストモデルのファイル(': ', checkpoint.best_model_path)
print('ベストモデルの検証データに対する損失: ', checkpoint.best_model_score)

# 学習したモデルでテストデータを評価
test = trainer.test(dataloaders=dataloader_test)
print(f'Accuracy: {test[0]["accuracy"]:.2f}')

# グラフを表示
%load_ext tensorboard
%tensorboard --logdir ./

# # optuna を実行する
# study = optuna.create_study(direction='minimize')
# study.optimize(objective, n_trials=20)
# print最適なハイパーパラメータ(': ', study.best_params)
# print最適な損失値(': ', study.best_value)
```

謝辞

この場を借りて、研究の遂行にあたりご支援いただいたすべての方々に心から感謝申し上げます。特に、指導教員の桂田准教授には、貴重なご意見と熱心な指導を賜りましたことを深く感謝いたします。また、本研究に関連する実験にご協力いただいた桂田研究室の皆様にも厚く御礼申し上げます。

参考文献

[1] 村田龍也, BERT による感情分析, 明治大学総合数理学部現象数理学科卒業研究レポート, 2022 年 2 月 13 日

[2] 岡谷貴之 機械学習プロフェッショナルシリーズ 深層学習 改訂第 2 版, 講談社, 2022

[3] 近江崇宏, 金田健太郎, 森長誠, 江間見亜利, BERT による自然言語処理入門 Transformers を使った実践プログラミング, オーム社, 2021.

[4] 山田育矢, 鈴木正敏, 山田康輔, 李凌寒, 大規模言語モデル, 技術評論社, 2023

[5] 我妻幸長, BERT 実践入門 PyTorch + Google Colaboratory で学ぶあたらしい自然言語処理技術, 翔泳社, 2023

[6] Tomas Mikolov and Kai Chen and Greg Corrado and Jeffrey Dean, Efficient Estimation of Word Representations in Vector Space, https://doi.org/10.48550/arXiv.1301.3781, 2014

[7] 小林悟郎, 栗林樹生, 横井祥, 乾健太郎, 東北大学,Langsmith 株式会社, 理化学研究所,Transformer におけるフィードフォワードネットの作用, https://www.anlp.jp/proceedings/annual_meeting/2022/pdf_dir/A5-5,2022

[8] Mor Geva, Roei Schuster, Jonathan Berant, Omer Levy, Transformer Feed-Forward Layers Are Key-Value Memories, https://arxiv.org/abs/2012.14913, 2021

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, https://arxiv.org/abs/1810.04805, 2019

[10] 佐野政太郎, 秋葉拓哉, 佐野正太郎, 太田健, 水野尚人, 柳瀬俊彦, Optuna によるブラックボックス最適化, オーム社, 2023

[11] Optuna-株式会社 Preferred Networks, https://www.preferred.jp/ja/projects/optuna/

[12] 毎日 2 万回ダウンロードされる Preferred Networks（PFN）の“Optuna”「探索空間」と「目的関数」でパラメーターを最適化する, https://logmi.jp/tech/articles/324834

[13] 森鳩武史, 野田朋裕, Sequential Model Based Optimization を用いた Support Vector Regression による自律神経指標 値予測モデル, https://www.jstage.jst.go.jp/article/bjsiam/28/3/28_23/_pdf/-char/ja

[14] Frank Hutter, Holger H. Hoos and Kevin Leyton-Brown, Sequential Model-Based Optimization for General Algorithm Configuration

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

[16] https://github.com/huggingface/transformers

[17] https://github.com/megagonlabs/jrte-corpus

[18] Yukun Zhu, Ryan Kiros, Richard S, Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. ln ICCV, 2015.

[19] 東山昌彦, 乾健太郎, 松本裕治, 述語の選択選好性に着目した名詞評価極性の 獲得, 言語処理学会第 14 回年次大会論文集, pp. 584-587, 2008. / Masahiko Higashiyama, Kentaro Inui, Yuji Matsumoto. Learning Sentiment of Nouns from Selectional Preferences of Verbs and Adjectives, Proceedings of the 14th Annual Meeting of the Association for Natural Language Processing, pp. 584-587, 2008.

[20] 林部祐太. 知識の整理のための根拠付き自然文間含意関係コーパスの構築. 言語処理学会第 26 回年次大会論文集, pp. 820-823. 2020.