修　士　論　文

題　　目　　関数単位のソースコードを入力とした
ドキュメント生成モデルの研究

主任指導教員　　水野　修　教授

指導教員　　崔　恩瀞　助教

京都工芸繊維大学大学院 工芸科学研究科

情報工学専攻

学生番号　　18622023

氏　　名　　塩津　拓真

令和2年2月10日提出

様式 2 号

学位論文内容の要旨（和文）

令和 2 年 2 月 10 日

京都工芸繊維大学大学院
工芸科学研究科長　殿

工芸科学研究科　　情報工学専攻
平成 30 年入学
学生番号　　18622023
氏　　名　　塩津　拓真　㊞

（主任指導教員　　水野　修　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

関数単位のソースコードを入力としたドキュメント生成モデルの研究

2.　論文内容の要旨（400 字程度）

ソフトウェア開発において，その工数の約半分はソースコードの挙動理解に費やされている．ドキュメントはその挙動理解に役立つため，工数削減に大きく影響する．近年，ニューラル機械翻訳技術の発展により，これを応用したドキュメント自動生成モデルの研究が注目されている．しかし，これらの生成モデルでは，未だに RNN や単純なソースコードベクトル化手法が使われており，最新の技術を適用することで改良できると考えられる．

本研究では，既に用いられているソースコードベクトル変換技術の SBT やニューラル機械翻訳モデルの Seq2Seq を最新のモデルに置き換えることが，精度向上に寄与するかを検証する．そこで，新たなソースコードベクトル変換技術として Statement Encoder，ニューラル機械翻訳モデルとして Transformer を用いて，これらの組み合わせによって 4 つのモデルを構築し，それぞれの BLEU スコアを比較した．

実験の結果，Statement Encoder と Seq2Seq の組み合わせが最も良い BLEU スコアを出すことを確認した．これは，より良いソースコードベクトル変換技術を応用することが，ドキュメント自動生成モデルの改良に繋がることを示唆している．

Does Code Representation From Statement Trees and a Novel NMT Architecture Improve Documents Auto-generation Model?

2020 18622023 *SHIOTSU Takuma*

Abstract

In the software development, it is said that about half of the efforts are spent on understanding the behavior of the source code. Documents are useful for understanding the code and effective for reducing such efforts. In recent years, due to the development of neural machine translation technology, attention has been paid to the research on automatic document generation models. However, these models still use a RNN or simple source code vectrization techniques, and are likely to be improved by applying the latest techniques.

In this thesis, I verify whether replacing the conventional technologies, such as SBT, which is the source code vectrization technique, and Seq2Seq, which is the neural machine translation to state-of-the-art techniques contributes to the improvement of accuracies. Then, using the Statement Encoder as a new source code vectrization technology and the Transformer as a neural machine translation model, I constructed 4 models by combining them and compared BLEU scores of 4 models.

As a result of the experiment, the combination of Statement Encoder and Seq2Seq marked the best BLEU score. This suggests that a better source code vectrization technology contributes to the automatic document generation model's improvement.

目　次

1. 緒言

ソフトウェアは我々の生活に多大な影響を与えており，これらの開発・運用・保守において，その工数を下げる試みは非常に重要な意味を持つ．この工数の多くはソースコードの挙動理解に費やされており，全体の 50%にも及ぶ [1].

コード中に書かれているコメントやドキュメントはソースコードの挙動を理解するのに役立つため，これを管理することで全体の工数を下げることができる．この管理を効率化するために，JavaDoc [2] や Doxygen [3] などのツールが研究開発された．しかし，依然として開発者はドキュメントの作成や保守の努力が必要であるため，工数削減の十分な解決方法であるとは言い難い．

近年，機械学習技術の発展により，多くのタスクで機械学習を用いた手法が取り入れられている．特に翻訳システムでは，以前は統計的手法が多く用いられていたが，最近は機械学習を用いた手法の事例が増加している [4–7]．これはニューラル機械翻訳と呼ばれ，RNN (Recurrent Neural Network) アーキテクチャを用いた Seq2Seq (Sequence-to-Sequence) [4] や Attention を用いた Transformer [6] などが広く使われている．

同様に，ソフトウェア工学の分野でも精度向上のために機械学習を用いた研究が多数発表されている [8–10]．中でも，ソースコードをニューラルネットワークモデルに入力することで得られた表現ベクトルを，特定の分類タスクに応用する研究は注目を浴びており，様々な手法が提案されている [11–13].

また，ニューラル機械翻訳技術を応用してソースコードからドキュメントを生成する研究が近年では活発に行われている [14–18]．これも翻訳システムと同様に，統計的手法より機械学習を用いた手法の事例が増加している．しかし，これらの生成モデルでは，未だに RNN や単純なソースコードベクトル化手法が使われており，先述の関連技術が十分に応用されているとは言い難い．そのため，未だ多くの改善の余地が残されていると考える．

本研究では，最近のニューラル機械翻訳技術やソースコードのベクトル変換モデルに注目し，それらが要約文書生成モデルの改善に寄与するかどうかを明らかにする．ニューラル機械翻訳技術として Transformer と既に生成モデルで用いられている Seq2Seq の 2 つを，ソースコードベクトル変換モデルとして Statement Encoder [19]

と既に生成モデルで用いられているSBT [15] の2つを対象として，合計4種類の生成モデルにおいて学習を行い，BLEUスコアを評価値として比較した．GitHub上で公開されている人気Javaプロジェクト1,000個から取得した約100万個のメソッドをデータセットとした．

以降の本稿の構成を紹介する．第2章では本研究の背景となるニューラル機械翻訳モデルやソースコードを入力としたベクトル変換手法，要約文章生成モデルについて述べる．第3章では本研究の目的と研究設問について述べる．第4章では本研究の比較実験に関する手法について述べる．第5章では実験結果を示し，それを元に研究設問に回答する．第6章では実験結果についてより深く考察する．最後に，第7章で本研究の結論を述べる．

2. 関連研究と背景

2.1 ニューラル機械翻訳

2.1.1 Seq2Seq

Seq2Seq とは LSTM (Long Short-term Memory) を用いた Encoder-Decoder モデルのことである [4,20]．Attention 機構を取り入れた，複数の LSTM で構成される Seq2Seq モデルが有名である [4]．図 2.1 にその概要図を示す．

それぞれの層について以下に示す．

(1) 埋め込み層

埋め込み (Embedding) 層とは各単語のベクトル表現を保持する層のことである．事前に作成した辞書が保持する単語数を $|V|$，単語ベクトルのサイズを d とすると，埋め込み層は $W_e \in \mathbb{R}^{|V| \times d}$ の重みを持つ．この W_e の各行が各単語の表現ベクトルに該当する．

(2) LSTM

LSTM とはシンプルな RNN に内部状態を保持する記憶セルと 3 つのゲートを加えた RNN アーキテクチャのネットワークである [21]．図 2.2 にその概要図を示す．以下にそれぞれの要素について記す．

出力ゲート

図 2.2 の ‘o’ がこれに該当し，式 2.1 で表される．

$$o = \sigma(x_t W_x^o + h_{t-1} W_h^o + b^o) \tag{2.1}$$

W_x^o は入力 x_t に対する重みであり，W_h^o は一時刻前の隠れ状態 h_{t-1} に対する重みである．b^o はバイアスであり，σ はシグモイド関数を表している．

時刻 t の隠れ状態 h_t は記憶セル c_t から計算されるが，その際にこのゲートはその計算結果の各要素をどれだけ伝えるかを調整する役割を持つ．

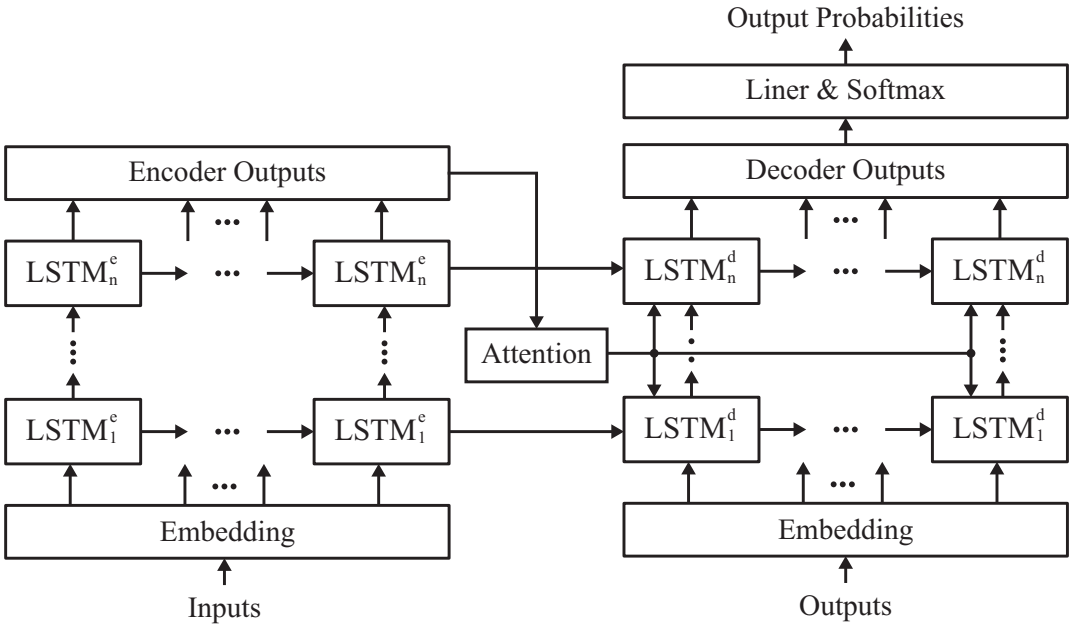

図 2.1　Attention 付き Seq2Seq モデルの概要図

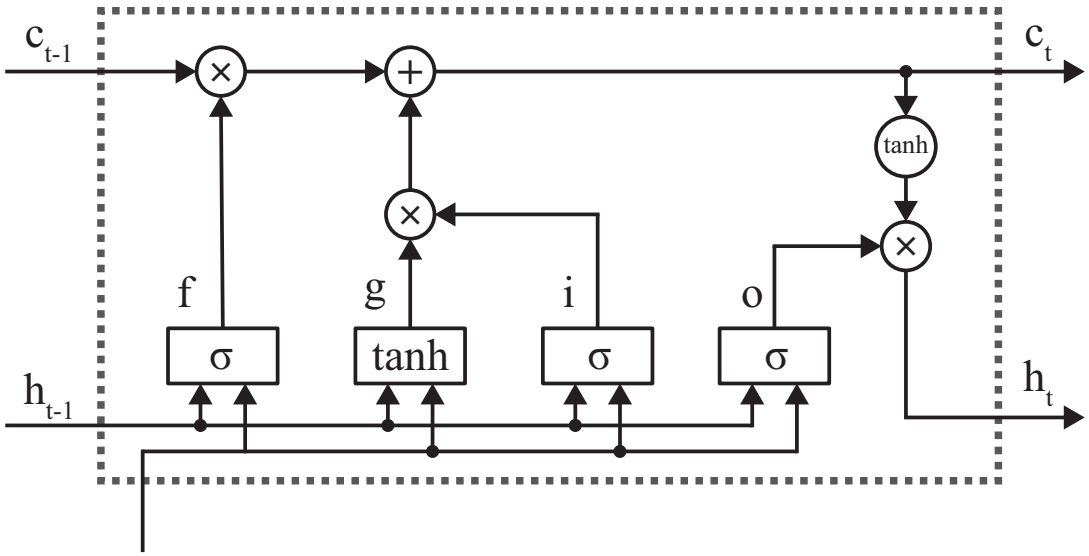

図 2.2　LSTM

忘却ゲート

図 2.2 の ‘f’ がこれに該当し，式 2.2 で表される．

$$f = \sigma(x_t W_x^f + h_{t-1} W_h^f + b^f) \tag{2.2}$$

このゲートは 1 つ前の記憶セル c_{t-1} から不要な情報を捨てる働きをする．

入力ゲート

図 2.2 の ‘i’ がこれに該当し，式 2.3 で表される．

$$i = \sigma(x_t W_x^i + h_{t-1} W_h^i + b^i) \tag{2.3}$$

記憶セル c_t は各時刻で新しく情報を付加される．図 2.2 の ‘g’ がこれに該当し，式 2.4 で表される．

$$g = tanh(x_t W_x^g + h_{t-1} W_h^g + b^g) \tag{2.4}$$

この生成された g に対して，各要素をどの程度使用するかを調整するのが入力ゲート i である．

これらのゲートを用いて記憶セル c_t と隠れ状態 h_t は下記の通りの式で計算される．$\circ$ はアダマール積を表す.

$$c_t = f \circ c_{t-1} + g \circ i \tag{2.5}$$

$$h_t = o \circ \tanh(c_t) \tag{2.6}$$

以上のゲートと記憶セルを用いた計算によって，RNN アーキテクチャで長期の時系列データを学習した場合に問題となる勾配消失を抑制することができる．

(3) Attention

Attention とは 2 つのベクトル，Query と Memory を入力とし，それらの関連度を取ることで，その Query における Memory の重要なベクトルを抽出する機構である [22,23]．Encoder-Decoder モデルにおいて，その伝達に使われている固定長のコンテキストベクトルでは適切に情報を圧縮できない問題を解決するために提案された．

Seq2Seq モデルでは，Query が Encoder の各時刻毎の隠れ層の状態であり，Memory が Decoder の隠れ層の状態となる．本研究では Bahdanau らが提案した Attention [22] を採用し，下記に示す式で計算した．

$$\mathrm{score}(q, m) = v_n^T * \tanh(W_1 q + W_2 m) \tag{2.7}$$

$$attention\_weight = \mathrm{softmax}(\mathrm{score}(q, m)) \tag{2.8}$$

$$context\_vector = attention\_weight * m \tag{2.9}$$

q，m はそれぞれ Query と Memory を表している．$attention\_weight$ が 2 つの関連度合いを表しており，最終的に得られる $context\_vector$ が抽出された Memory の重要なベクトル表現である．

2.1.2 Transformer

Transformer は 2017 年に発表された，現在の機械翻訳を含む自然言語処理で注目されているニューラルネットワークアーキテクチャである [6, 24]．図 2.3 にその概要図を示す．

Seq2Seq と同じ Encoder-Decoder モデルであるが，RNN を使用せず Attention 機構を基礎としたアーキテクチャとなっている．それぞれの層について以下に示す．

(1) Positional Encoding

Transformer では再帰的な計算が行われないため，入力されるトークン列に対して位置情報を注入する必要がある．PE (Positional Encoding) では下記の式によって位置情報を算出する．pos はそのトークンのトークン列での位置を表しており，i はそのトークンの埋め込み表現における次元を表している．

$$\mathrm{PE}_{(pos,2i)} = \sin(pos/10000^{\frac{2i}{d_{model}}}) \tag{2.10}$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos(pos/10000^{\frac{2i}{d_{model}}}) \tag{2.11}$$

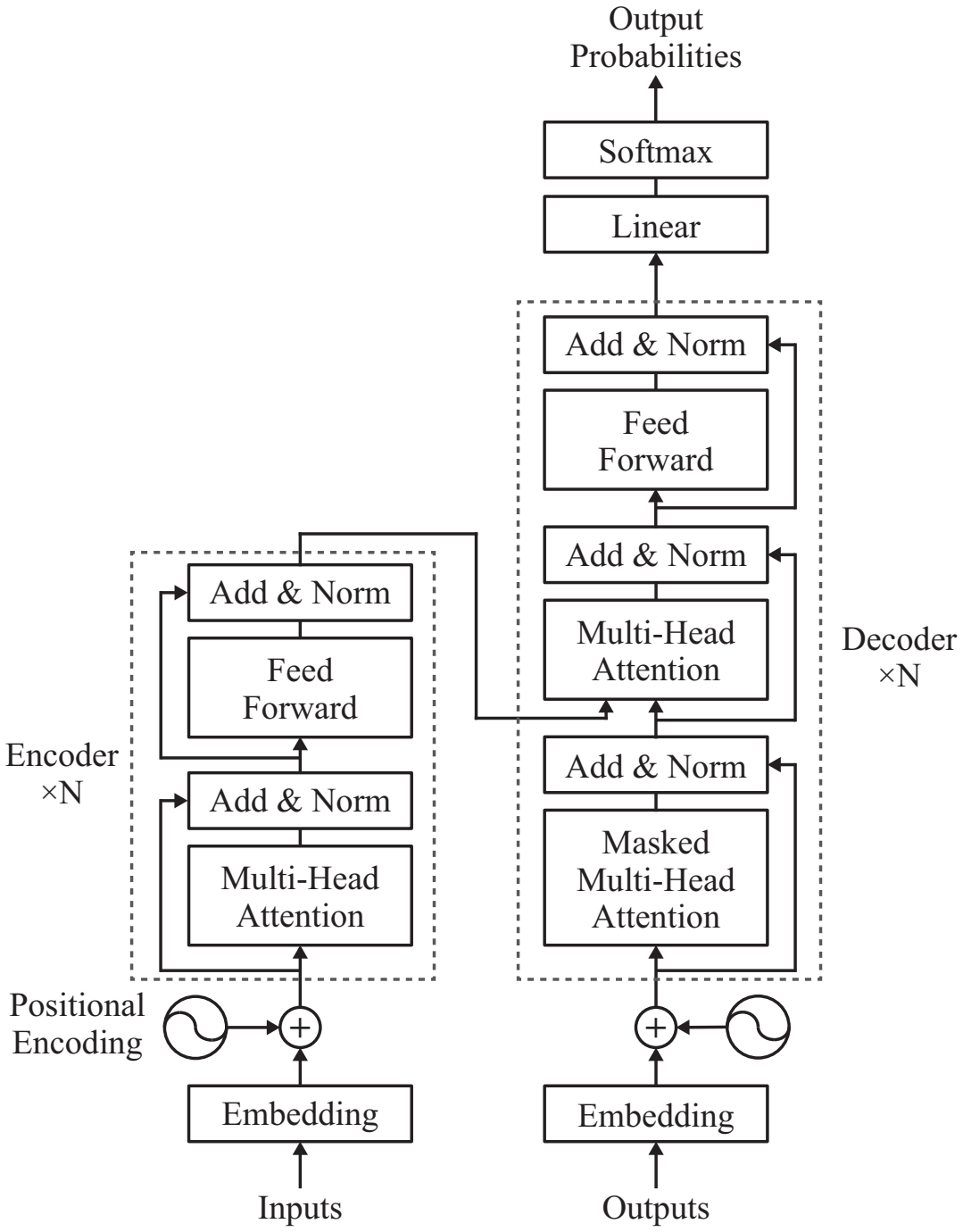

図 2.3　Transformer の概要図

上記の式 2.10，2.11 より算出した位置情報を埋め込み表現ベクトルに足し合わせる．

(2) Multi-Head Attention

図 2.4 と 2.5 に Multi-Head Attention とその中で使用されている Scaled Dot-Product Attention の概要図を示す．

Scaled Dot-Product Attention では，下記の式のようにして *context_vector* を算出している．

$$context\_vector = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.12}$$

Q，K，V はそれぞれ Query，Key，Value を表しており，先述した Attention では Memory が Key と Value に該当する．d_k は K の次元数を表しており，図 2.5 の Scale 層の処理がこれに該当する．図 2.5 の Mask は Softmax 後，padding 部分の重みが 0 になるように十分大きな負の整数を足しておく層である．

Multi-Head Attention では上記の Scaled Dot-Product Attention を使用して，下記の式のようにベクトルを算出している．

$$\text{MultiHeadAttention} = \text{Concat}(head_1, ..., head_h)W^O \tag{2.13}$$

$$where\ head_i = \text{ScaledDotProductAttention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.14}$$

入力される 3 つのベクトルはそれぞれ，h 個のベクトルに分割され，各々の Q_i, K_i, V_i のセットに関して Scaled Dot-Product Attention による計算がされる．最終的に算出されたベクトルを連結し 1 度全結合層に通すことで 1 つのベクトル表現を得る．

また，図 2.3 の通り，Transformer の Encoder では Q，K，V の全てに入力シーケンスのベクトルまたは Encoder の出力を使用している．これは Self Attention と呼ばれ，時系列データの各データ間の関係性を双方向 LSTM のような再帰型ネットワークより低いオーダー $O(n)$ で計算することを可能としている．Decoder では K，V に Encoder の出力を，Q に出力シーケンスのベクトルまたは Decoder の出力を使用している．

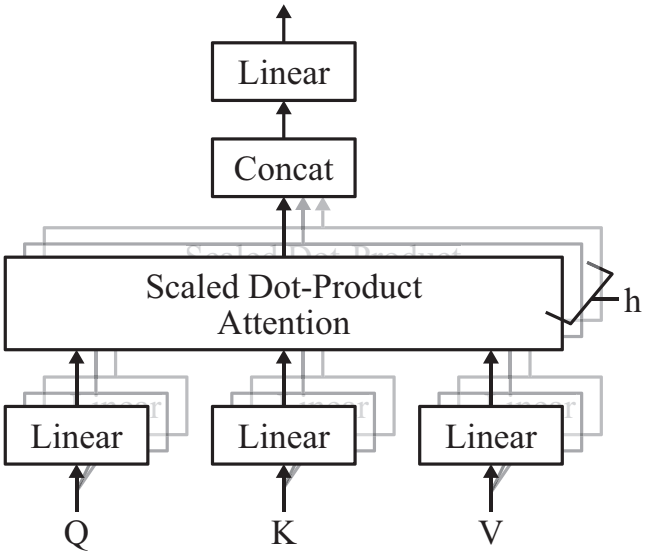

図 2.4　Multi-Head Attention の概要図

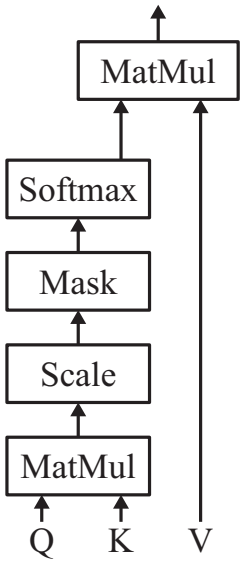

図 2.5　Scaled Dot-Product Attention の概要図

図 2.3 の Decoder に存在する Masked Multi-Head Attention では，計算時に未来の単語トークン情報が入らないように，padding トークンだけでなく予測時刻よりも先のトークン情報も重みが 0 になるように処理している．

(3) Feed Forward

FFN (Feed Forward Networks) は 2 層の全結合層から成るネットワークである．それぞれ $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$，$b_1 \in \mathbb{R}^{d_{ff}}$，$W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$，$b_1 \in \mathbb{R}^{d_{model}}$ を重みとすると，下記の通りに計算される．

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.15}$$

1 層目の活性化関数には ReLU 関数を用いている．

(4) Normalization

DNN では学習過程において各層の出力分布がバッチによって変わっていくため，これを原因として勾配の消失や爆発が起き，効率的な学習の妨げになる．このような現象を抑えて学習効率を上げるために正規化を行う手法が存在する [25]．Transformer では Layer Normalization [26] を使用する．

(5) 学習と最適化

Transformer では最適化手法に Adam [27] が使用される．学習率 l_r は以下の式によって算出される．$step\_num$ は学習時のエポック数を表しており，$warmup\_steps$ は学習率の増加を制限するためのパラメータである．

$$l_r = d_{model}^{-0.5} * \min(step\_num^{-0.5}, step\_num * warmup\_steps^{-1.5}) \tag{2.16}$$

論文中では $warmup\_steps = 4000$，Adam のパラメータとして $\beta_1 = 0.9$，$\beta_2 = 0.99$，$\epsilon = 10^{-9}$ が使用されており，本研究でもこのパラメータを採用している．

また，Transformer は埋め込みベクトルの次元数や各層での入力及び出力ベクトルの次元数を示す d_{model}，FFN の 1 層目の出力ベクトルの次元数を示す d_{ff}，Multi-Head

Attention の head 分割数を示す n_{head}，そして Encoder 及び Decoder の深さを示す n_{layer} が主要なハイパーパラメータとして使用される．

2.1.3 BLEU

機械翻訳で使用されている評価値の 1 つで，本研究でも用いる BLEU (Bilingual Evaluation Understudy) について説明する [28].

BLEU とは生成文章と正解文章の n-gram 一致度から算出される評価値である．生成文章，正解文章の単語トークン列をそれぞれ T，R とする．それらの間で n-gram 一致した数を m_n，T の全ての n-gram の数を l_n とすると，2 つの文章の一致度は下記のように計算される．

$$p_n = \frac{m_n}{l_n} \tag{2.17}$$

ここでの n-gram 一致では候補語は 1 度しか使用されない．

また，T の単語数が R より少ない場合にペナルティを課す係数 BP (Brevity Penalty) を下記のように定義する．len() はその文章の単語トークン数を表す.

$$BP = \begin{cases} 1 & (\mathrm{len}(T) > \mathrm{len}(R)) \\ e^{1-\frac{\mathrm{len}(T)}{\mathrm{len}(R)}} & (\mathrm{len}(T) \leq \mathrm{len}(R)) \end{cases} \tag{2.18}$$

以上から，BLEU スコアは下記のように算出される．w_n はそれぞれの次元 n に対する重みである．

$$\text{BLEU Score} = BP * \exp(\sum_{n=1}^{N} w_n \log p_n) * 100 \tag{2.19}$$

一般的に $N = 4$，$w_n = \frac{1}{N}$ が使用される．

また，高次元の n-gram 不一致によるスコア消失を緩和するため，この BLEU スコア計算に平滑化処理を挟む [29].

まず，m_n をアルゴリズム 1 のように計算する．

そしてその m_n を下記の式のように計算して平滑化した m'_n を算出する．

$$m'_0 = m_1 + 1 \tag{2.20}$$

アルゴリズム 1 BLEU の m_n を平滑化する処理

$invcnt = 1$

$K = 5$

for $n = 1$ **to** N **do**

  if $m_n = 0$ **then**

    $invcnt = invcnt * \frac{K}{\ln(len(T))}$

    $m_n = 1/invcnt$

  end if

end for

$$m'_n = \frac{m_{n-1} + m_n + m_{n+1}}{3} \tag{2.21}$$

この m'_n を用いて以上のように BLEU スコアを算出する．本研究では，この BLEU スコア計算に NLTK ライブラリ(注 1)を使用する．

2.2 ソースコードのベクトル表現

ソフトウェアの種類分別やコードクローンの発見，バグ混入の発見など様々な分野でソースコードを入力としたモデルの提案が行われている [11–13, 19]．

その中でも 2019 年に発表された，本研究でも使用する ASTNN について下記に示す．

2.2.1 ASTNN

より良いソースコードベクトル表現を得るために，Zhang らによって提案された DNN モデルである [19]．図 2.6 にそのモデル概要図を示す．それぞれの層について説明する．

ASTNN では AST を一度 Statement 単位で分割し，Statement Tree を作成する．Statement Encoder では，このそれぞれの部分木に対して下記の式の通りに計算することで，各 Statement の表現ベクトルを算出する．

$$e_t = [\max(v_{i1}), ..., \max(v_{id})], i = 1, ..., N \tag{2.22}$$

このとき，各ノードの埋め込み表現 v_n は事前学習によって作成された重みを初期値とする．事前学習では Skip-gram の Word2Vec [30] を使用し，Statement Tree を行きがけ順に深さ優先探索して作成した時系列データを入力とする．また，末端ノードの識別子はキャメルケースやスネークケース，ケバブケースの命名規則に則って単語レベルに分割する．

そうして得られた Statement の表現ベクトルのリストである Statement List を時系列データとして双方向 GRU に入力し，各ステップの出力をプーリング層に入力することで最終的な AST 全体の表現ベクトルを得る．GRU とは LSTM と同じ再帰型ネットワークの 1 つで，2 つのゲートを持ち LSTM より軽量な構造となっている．

(注 1) : https://www.nltk.org/_modules/nltk/translate/bleu_score.html

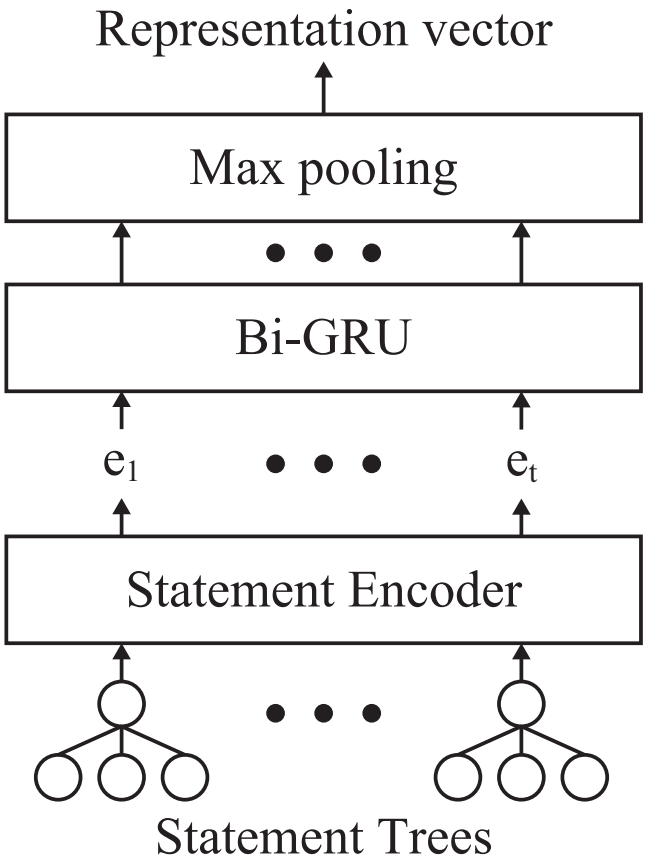

図 2.6　ASTNN モデルの概要図

本研究では AST の分割から Statement Encoder までを応用する．

2.3 要約文章生成

JavaDoc などの仕様に沿って，ソースコード内の各メソッドに書かれている，その挙動を簡潔に説明している自然言語文章を要約文章と定義する．元々，要約文章を生成するモデルは統計的手法が主流であった [14]．しかし近年ニューラル機械翻訳技術を利用したモデルが数多く提案されている [15–18,31]．そして，そのいずれにおいても，ソースコードを自然言語としてそのまま入力するより，AST などの構造データを扱った方が良い結果が得られることが分かっている．

ニューラル機械翻訳を用いた要約文章生成モデルの 1 つである DeepCom について説明する [15]．

2.3.1 DeepCom

2018 年に Xu らによって提案されたモデルである [15]．ソースコードから取得した AST を SBT (Structure-Based Traversal) という方法で時系列データに変換し，Seq2Seq モデルに入力する．

図 2.7 に SBT の具体的な変換例を示す．AST を行きがけ順の深さ優先探索で走査しながら見つけたノード列の間に ‘(’ と ‘)’ を挿入することで，構造的な情報を残しつつ Seq2Seq モデルで扱うことのできる時系列データに変換する．

また走査時，末端ノードはその型と値がアンダースコア (_) によって結合される．自然言語を扱うモデルではメモリの都合上，ボキャブラリサイズを制限することが多い．AST の中で最もその内容の種類が多岐に渡るのは末端ノードに配置される値である．そのため，ボキャブラリサイズの制限によってボキャブラリ外のノードを辞書 id に変換する場合，例えば `SimpleName_<UNK>`のように，未定義語彙を示す`<UNK>`を値としたトークンが使用される．これによって型情報が最低限付与されるように学習することができる．

この時系列データが入力される Seq2Seq モデルには 2 層の LSTM が使用されている．

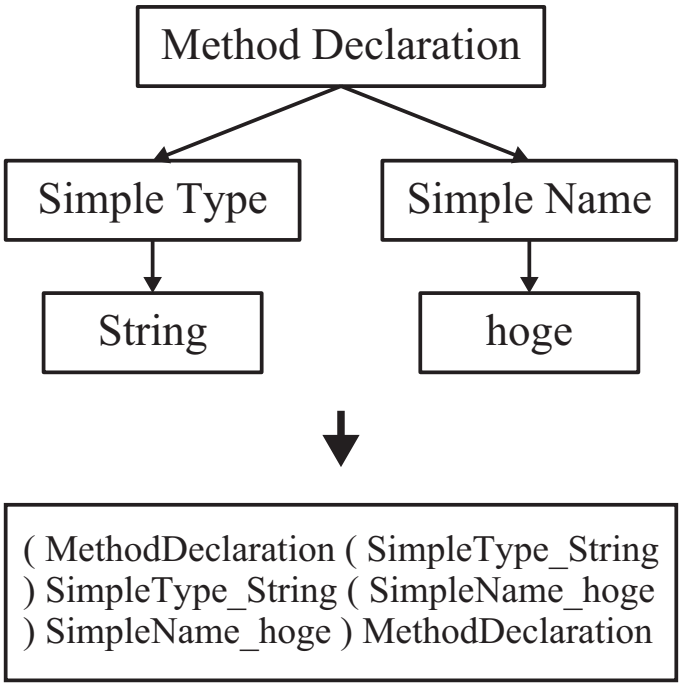

図 2.7　SBT によって木構造データを時系列データに変換する例

3. 研究の目的

本研究の目的は，ソースコードのベクトル表現手法として最近注目されている Statement Encoder と，ニューラル機械翻訳アーキテクチャとして注目されている Transformer が，ソースコードを入力とした要約文章生成モデルに活用できるかどうかを明らかにすることである．

そのため，本研究では以下に示す研究設問について検証を行う．

RQ1 Statement Encoder は精度向上に寄与するか

RQ2 Transformer は精度向上に寄与するか

RQ3 生成した文章はそのメソッドの内容を説明しているか

4. 実験内容

4.1 モデルの概要

先行研究より，2 つのソースコードベクトル表現取得モデルと，2 つの Encoder-Decoder モデルの組み合わせを対象とする．表 4.1 にその概要図を示す．先述した通り，SBT と Seq2Seq の組み合わせはすでに DeepCom というモデルとして提案されている [15]．これらの 4 つのモデルについて，どの組み合わせが一番良い BLEU スコアを出すのかを確認する．

4.2 データセット

データセットを用意するにあたって，以下の手順を行った．

1. GitHub 上で公開されている人気 Java プロジェクトの取得
2. Java ソースコードを関数レベルの AST に変換
3. AST からドキュメントを取り出し整形
4. データセットの分割
5. ソースコードベクトル表現モデルのための前処理

4.2.1 人気 Java プロジェクトの取得

GitHub API (注 2) を使用して，公開されている Java プロジェクトを Star 順で上から 1000 個取得した．プロジェクトは全て 100 個以上のスターを保持している．

4.2.2 AST への変換

JavaParser (注 3) という Java ライブラリを用いて Java ソースコードを AST に変換する．文章生成モデルは Python で実装するため，変換した AST は YAML 形式で出力した．ソースコード 4.1 にその変換例を示す．

(注 2) : https://developer.github.com/v3/
(注 3) : https://github.com/javaparser/javaparser

表 4.1　ソースコードのベクトル表現と文章生成モデルの組み合わせ

	Seq2Seq	Transformer
SBT	DeepCom [15]	-
Statement Encoder	-	-

ソースコード 4.1　Java から AST の YAML 表現への変換例

```
public class Hoge {
    /**
     * This is a sample.
     *
     * @return constant
     */
    static int foo() {
        return 1;
    }
}
```

```
root(Type=MethodDeclaration):
    body(Type=BlockStmt):
        statements:
            - statement(Type=ReturnStmt):
                expression(Type=IntegerLiteralExpr):
                    value: "1"
    type(Type=PrimitiveType):
        type: "INT"
    name(Type=SimpleName):
        identifier: "foo"
    comment(Type=JavadocComment):
        content: "\n * This is a sample.\n *\n * @return constant\n "
    modifiers:
        - modifier(Type=Modifier):
            keyword: "STATIC"
```

4.2.3 ドキュメントの取得と整形

ソースコード 4.1 では `root(Type=MethodDeclaration)` ノード直下にある `comment(Type=JavadocComment)` ノードが JavaDoc コメントに該当する．本研究では JavaDoc コメントのタグの付いていない最初の文章を，その関数の要約文章とした．ソースコード 4.1 では “This is a sample.” が要約文章として扱われる．

得られた AST に対して英語の要約文章を持つかどうかでフィルタリングを行い，5,528,259 個の関数から 1,005,418 個の関数を取得した．また，要約文章はピリオドで分割し最初の 1 文のみを生成対象文章とし，ピリオドが無い場合は最初の改行を文の終わりとした．AST から JavaDoc に関するノードは除外し，入力 AST に要約文章が入らないようにした．

以上の前処理を行い，得られたデータセットに関する情報を表 4.2 に示す．

4.2.4 データセットの分割

1,005,418 個の全てのデータセットをランダムに学習データ 804,328 個，検証データ 100,545 個，テストデータ 100,545 個に分けて実験を行った．学習データはモデルの学習のために用いられ，検証データは学習が 1 エポック終了する毎にモデルの性能を検証するために用いられる．テストデータは学習が終了したモデルの最終的な評価を行うために用いられる．学習データ，検証データ，テストデータの個数の比率はおよそ 8:1:1 となっている．

4.2.5 ソースコードベクトル表現モデルのための前処理

SBT と Statement Encoder の入力のために前処理を行った．学習データにおける各前処理後の SBT の長さの分布，Statement の数の分布を表 4.3 に示す．

Statement Encoder の埋め込み表現事前学習には gensim ライブラリ (注 4) を使用した．

4.3 学習環境

本研究では実験環境として ABCI (AI Bridging Cloud Infrastructure) [32] を使用した．計算機の構成要素としては，Intel®Xeon®Gold 6148 CPU @ 2.40GHz 2 基，

(注 4) : https://radimrehurek.com/gensim/index.html

表 4.2　要約文章について前処理をしたデータセット

	最小値	第一四分位数	中央値	第三四分位数	最大値
ファイル毎の AST 数	1	1	2	5	3,070
AST 毎のノード数	6	32	57	122	70,991
AST 毎の深さ	4	9	11	14	163
要約文章の長さ	2	6	9	13	249

表 4.3　各ソースコードベクトル表現モデル用の入力データセット情報

	最小値	第一四分位数	中央値	第三四分位数	最大値
AST 毎の SBT の長さ	12	76	136	296	161,036
AST 毎の Statement の数	1	2	2	6	2,184

NVIDIA®Tesla®V100 for NVLink 16GiB HBM2 4 基，RAM384GiB，CentOS 7.5 となっている．

4.4 実験パラメータ

用いたボキャブラリサイズについて表 4.4 に示す．このボキャブラリサイズはデータセット中のトークン出現回数や学習環境のメモリ容量を踏まえて決定した．

また，表 4.5 に用いた SBT と Statement List の最大長を示す．これはデータセット中の 80%以上が含まれるように決定した．

各モデルに用いるハイパーパラメータについて説明する．Seq2Seq モデルでは，*embedding* に (32, 64, 128, 256, 512)，*hidden* に (32, 64, 128, 256, 512, 1024, 2048)，*dropout* に 0.1 を使用した．Transformer モデルでは，d_{model} に (32, 64, 128, 256, 512)，d_{ff} に (32, 64, 128, 256, 512, 1024, 2048, 4096)，n_{head} に (1, 2, 4, 8)，n_{layer} に (2, 4, 6, 8) を使用した．以上の候補を元にいくつかの組み合わせを試した．

表 4.4　ボキャブラリサイズ

	ボキャブラリサイズ	最低出現回数
Description	18,455	9
SBT Node	46,692	50
Statement Encoder Node	25,128	10

表 4.5　生成モデルで用いた SBT と Statement List の最大長

	最大長
SBT	356
Statement List	40

5. 実験結果

5.1 定量的結果

表 5.1 に各モデルの BLEU スコアを示す．Statement Encoder + Seq2Seq が最もよい BLEU スコアである 46.6 を出した．

各モデルのハイパーパラメータについて表 5.2，5.3 に示す． また，各モデルの学習時間について表 5.4 に示す．1 エポックあたりの学習時間が一番長かったのは SBT + Seq2Seq モデルの 13,000 秒で，一番短かったのは Statement Encoder + Transformer モデルの 2,250 秒であった．

5.2 定性的結果

テストデータからランダムに 100 個メソッドを取り出し，BLEU スコアが最も優れていた Statement Encoder + Seq2Seq モデルに入力した．図 5.1-5.4 にその例を示す．

図 5.1 では正解文章に完全一致する文章の生成に成功している．図 5.2，5.3 は文章として成立しているが，正解文章とは異なる．これに関しては 6.2 章で考察する．図 5.4 は文章として成立していない．これに関しても 6.2 章でさらに考察する．

5.3 研究設問への回答

5.3.1 RQ1: Statement Encoder は精度向上に寄与するか

表 5.1 から，Statement Encoder を用いたモデルは SBT を用いたモデルより良い BLEU スコアを出していることが分かった．

5.3.2 RQ2: Transformer は精度向上に寄与するか

表 5.1 から，Transformer が Seq2Seq モデルより良い BLEU スコアを得ることはできないことが分かった．

5.3.3 RQ3: 生成した文章はそのメソッドの内容を説明しているか

図 5.1-5.4 の生成例を見てみると，4 つ目の例を除いて，文章として成立している．

表 5.1　各モデルの BLEU スコア

	Seq2Seq	Transformer
SBT	43.7	29.3
Statement Encoder	46.6	40.3

表 5.2　最も良いスコアを出した Seq2Seq モデルのパラメータ

	embedding	*hidden*	*dropout*
SBT + Seq2Seq	512	512	0.1
Statement Encoder + Seq2Seq	512	512	0.1

表 5.3　最も良いスコアを出した Transformer モデルのパラメータ

	d_{model}	d_{ff}	n_{head}	n_{layer}	*dropout*
SBT + Transformer	256	1024	8	6	0.1
Statement Encoder + Transformer	512	2,048	8	6	0.1

表 5.4　各モデルの学習時間

	エポック数	1 エポック毎の学習時間 (sec)
SBT + Seq2Seq	32	13,000
SBT + Transformer	194	3,320
Statement Encoder + Seq2Seq	60	4,400
Statement Encoder + Transformer	234	2,250

図 5.1　Statement Encoder + Seq2Seq モデルの文章生成例 1

ソースコード: apache/camel SmppEndpointBuilderFactory.java [33]

正解文章: defines the interval in milliseconds between the confidence checks .

生成文章: defines the interval in milliseconds between the confidence checks .

```
default AdvancedSmppEndpointBuilder enquireLinkTimer(
        Integer enquireLinkTimer) {
    doSetProperty("enquireLinkTimer", enquireLinkTimer);
    return this;
}
```

図 5.2　Statement Encoder + Seq2Seq モデルの文章生成例 2

ソースコード: aosp-mirror/platform_frameworks_base ApnSetting.java [34]

正解文章: returns the current status of APN .

生成文章: returns whether this device has been enabled .

```
public boolean isEnabled() {
    return mCarrierEnabled;
}
```

図 5.3　Statement Encoder + Seq2Seq モデルの文章生成例 3

ソースコード: apache/hadoop DirectoryWithSnapshotFeature.java [35]

正解文章: clean an inode while we move it from the deleted list of post to the deleted list of prior.

生成文章: remove all deleted nodes from the terrain tree .

```
1  private static void cleanDeletedINode(INode.ReclaimContext reclaimContext
       ,
2      INode inode, final int post, final int prior) {
3    Deque<INode> queue = new ArrayDeque<>();
4    queue.addLast(inode);
5    while (!queue.isEmpty()) {
6      INode topNode = queue.pollFirst();
7      if (topNode instanceof INodeReference.WithName) {
8        INodeReference.WithName wn = (INodeReference.WithName) topNode;
9        if (wn.getLastSnapshotId() >= post) {
10         INodeReference.WithCount wc =
11             (INodeReference.WithCount) wn.getReferredINode();
12         if (wc.getLastWithName() == wn && wc.getParentReference() == null) {
13           // this wn is the last wn inside of the wc, also the dstRef node
                has
14           // been deleted. In this case, we should treat the referred file/
                dir
15           // as normal case
16           queue.add(wc.getReferredINode());
17         } else {
18           wn.cleanSubtree(reclaimContext, post, prior);
19         }
20       }
21       // For DstReference node, since the node is not in the created list
             of
22       // prior, we should treat it as regular file/dir
23     } else if (topNode.isFile() && topNode.asFile().isWithSnapshot()) {
24       INodeFile file = topNode.asFile();
25       file.getDiffs().deleteSnapshotDiff(reclaimContext, post, prior, file)
             ;
26     } else if (topNode.isDirectory()) {
27       INodeDirectory dir = topNode.asDirectory();
28       ChildrenDiff priorChildrenDiff = null;
29       DirectoryWithSnapshotFeature sf = dir.
             getDirectoryWithSnapshotFeature();
30       if (sf != null) {
31         // delete files/dirs created after prior. Note that these
32         // files/dirs, along with inode, were deleted right after post.
33         DirectoryDiff priorDiff = sf.getDiffs().getDiffById(prior);
34         if (priorDiff != null && priorDiff.getSnapshotId() == prior) {
35           priorChildrenDiff = priorDiff.getChildrenDiff();
36           priorChildrenDiff.destroyCreatedList(reclaimContext, dir);
37         }
38       }
39
40       for (INode child : dir.getChildrenList(prior)) {
41         if (priorChildrenDiff != null && priorChildrenDiff.getDeleted(
42             child.getLocalNameBytes()) != null) {
43           continue;
44         }
45         queue.addLast(child);
46       }
47     }
48   }
49 }
```

図 5.4　Statement Encoder + Seq2Seq モデルの文章生成例 4

ソースコード: apache/incubator-pinot QuantileDigest.java [36]

正解文章: convert a 64-bit lexicographically-sortable binary to a java long (two's complement representation) .

生成文章: converts a 90 encoded 64-bit 64-bit integer to a java double .

```
1 private static long bitsToLong(long bits) {
2   return bits ^ 0x8000_0000_0000_0000L;
3 }
```

図 5.2 の生成例に関して，正解文章の APN (Access Point Name) とは携帯端末でデータ通信をするための接続先情報のことを指している．つまり端末に対してデータ送信が可能かどうかを返すメソッドであると説明している．生成文章では APN という単語は使用していないものの，そのデバイスが使用可能かを返すメソッドであると説明しているため，意味は同じである．図 5.3 も生成文章は正解文章よりも短いが，おおよそ同じ内容を説明している．

一方で，図 5.4 は変換という表現自体は合っているが，その後に続く文章がメソッドの内容を説明していない．

6. 考察

6.1 研究設問に対する考察

6.1.1 RQ1: Statement Encoder は精度向上に寄与するか

表 5.1 より，SBT よりも Statement Encoder を用いることで良い BLEU スコアが得られた結果について考察する．

SBT では AST 上の全てのノードを同一の価値として表現している．一方で Statement Encoder は，Statement という単位で AST を分割し，表現ベクトルを得た後，それを用いて AST 全体の表現ベクトルを生成している．プログラムはトークン単体で意味を持つのではなく，それらを組み合わせた式や文，または宣言によってその命令の意味を持つ．よって，それを学習する余地が存在する構造となっている Statement Encoder がより良い精度を出したのだと考える．

6.1.2 RQ2: Transformer は精度向上に寄与するか

表 5.1 からは Transformer が精度向上に寄与することが確認できなかった．しかし，表 6.1 の BLEU スコアの遷移を見てみると微量ではあるがその数値が上がり続けている．Transformer の提案論文では学習時のエポック数が 10 万から 30 万であるため，収束にかなり時間がかかると考える．そのため，学習率の調整と共に，現時点からさらに学習を進めることで，Seq2Seq を用いたモデルより精度が高い結果を出す可能性があると考える．

一方で，Seq2Seq モデルは Transformer よりも短時間で収束し，一定の精度の要約文章の生成を可能とする．全体の学習時間という点においては，Seq2Seq の方が優れていると考える．

6.1.3 RQ3: 生成された文章はそのメソッドの内容を説明しているか

図 5.1 について考察する．この例では正解文章に完全一致する文章の生成に成功している．しかし，元のソースコードを見てみると，同じクラス内に要約文章と処理の内容が同一のメソッドが他に 5 つ存在した．学習データにこれらのコードクローンが存在したため，完全一致の文章が生成できたのだと考える．

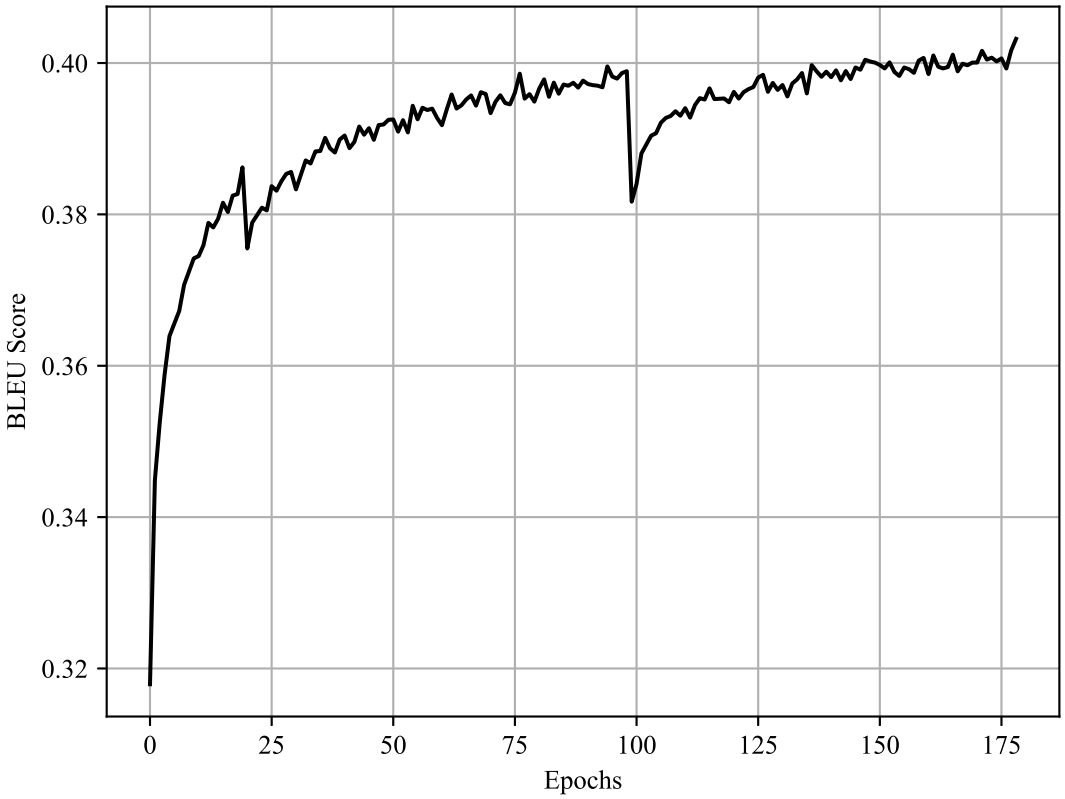

図 6.1　Statement Encoder + Transformer の BLEU スコアの推移

図 5.2 について考察する．APN という固有名詞の再現はできてはいないものの，“device” という言葉に言い換え，同様の内容を説明している．これらの単語はメソッド内部では出現していない．このことから，Statement Encoder + Seq2Seq モデルは一定のプロジェクト内の固有情報を表現ベクトルを取得し，文章生成に活用できていると考える．

図 5.3 について考察する．49 行という他の例よりも大きなメソッドにも関わらず，情報削除に関する文章を生成できている．このことから，サイズが大きなメソッドでも本質的な情報を持った表現ベクトルの取得に成功していると考える．

図 5.4 について考察する．このメソッドは渡された 64bit 整数を 2 進数の補数表現に変換して返している．しかし，生成された文章では 64bit 整数を double 型に変換させるという意味を持つような文章が生成されている．補数表現の意味は`^ 0x8000_0000_0000_0000L` から予測するしかなく，このような数値計算の表現ベクトルを正しく取得するのは，Statement Encoder + Seq2Seq モデルでは困難であると考える．

6.2 妥当性の脅威

6.2.1 構成概念妥当性

構成概念妥当性では実験における評価手法の妥当性について議論する．

本実験では生成された文章の精度の評価指標として BLEU を用いた．この指標は機械翻訳における性能評価指標として一般的に利用されており，本実験における手法の評価指標として妥当であると考える．一方で，これは英文法の正確さやプロジェクト内特有の知識を有するかどうかの評価をするための指標ではない．実際に使用できるかどうかを判断するための評価指標を新たに導入する必要がある．

また，ボキャブラリサイズや入力時系列データの最大長などをいくつかの粒度で決定し，それぞれに対して実験を行うことで，より厳密な比較が行えると考える．他にも，本実験では時間的な理由によりビームサーチ [37] を行えていない．時系列データの生成を扱うモデルでは，ビームサーチを行うことでより正確な予測を行えることが分かっているため，これを実装する必要があると考える．

本実験では GitHub 上でスターを 100 個以上所持しているリポジトリ 1,000 個から

約 100 万のメソッドを対象とした．機械翻訳タスクで用いられているデータセットはそのデータ数が約 500 万程度であることから [23]，今回用意したデータセットのサイズは適切であると考える．しかし，広く使われている公開データセットではないため，これらのメソッドに書かれている要約文章に誤りがないことや，バグが無いことは保証できないことから，構成概念妥当性への脅威となりえる．

6.2.2 外的妥当性

外的妥当性では実験結果の一般性についての妥当性を議論する．

本実験では各モデルの BLEU スコアは取得したデータセットの 10%にあたる約 10 万個のメソッドで算出した．この対象としたメソッドがその他の対象としていないメソッドの特徴を網羅しているかは妥当性の脅威となりえる．

また，Java 以外の他のプログラミング言語での検証を行っていないため，用いたモデルがプログラミング言語を横断した汎用的な性能を有しているかを判断することは難しいと考える．

6.2.3 内的妥当性

内的妥当性では本実験の方法の妥当性について議論する．

本実験で利用したプログラムはほぼ全て著者が実装した．都度テストやデバッグをしながら進めたが，不具合が残っている可能性は否定できない．そのため，内在するバグが内的妥当性への脅威となりえる．

7. 結言

本研究ではソースコードベクトル表現手法とニューラル機械翻訳技術を用いて，Java メソッドの要約文章を生成するモデルについて比較実験を行った．ソースコードベクトル表現手法には SBT と Statement Encoder の 2 つを採用した．ニューラル機械翻訳技術には Seq2Seq と Transformer の 2 つを採用した．対象のソースコードは GitHub 上で公開されている 100 個以上の Star を持つ Java リポジトリ 1000 個から取得した約 100 万個のメソッドである．

各モデルの比較のために BLEU スコアを使用した．実験の結果，Statement Encoder + Seq2Seq のモデルが最も良いスコアを出すことが分かった．また，そのモデルから生成された文章は一定のコンテキスト情報を含んだ文章であることが確認できた．

今後の課題として，ビームサーチの実装や十分な Transformer の学習などによって，より精度の高い文章生成モデルの構築があると考える．

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学　人間科学系　水野修教授，並びに崔恩瀞助教に深く感謝致します．本報告執筆にあたり貴重な助言を多数頂きました，産業技術総合研究所　情報技術研究部門　ソフトウェアアナリティクス研究グループ　森彰博士，本学ソフトウェア工学研究室の皆さん，学生生活を通じて著者の支えとなった友人に深く感謝致します．

参考文献

[1] F. Lv, H. Zhang, J.-g. Lou, S. Wang, D. Zhang, and J. Zhao, “CodeHow : Effective Code Search based on API Understanding and Extended Boolean Model,” Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp.260–270, 2015.

[2] Oracle，Javadoc ガイド，（オンライン），入手先〈https://docs.oracle.com/javase/jp/12/javadoc/javadoc.html〉（参照 2020-01-29）.

[3] D. vanHeesch, Doxygen, （オンライン）, 入手先〈http://www.doxygen.jp/〉（参照 2020-01-29）.

[4] I. Sutskever, O. Vinyals, and Q.V. Le, “Sequence to sequence learning with neural networks,” Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS), vol.2, pp.3104–3112, Jan. 2014.

[5] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y.N. Dauphin, “Convolutional Sequence to Sequence Learning,” Proceedings of the 34th International Conference on Machine Learning (ICML), vol.70, pp.1243–1252, may 2017.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L.u. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS), eds. I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, pp.5998–6008, Curran Associates, Inc., 2017.

[7] P. Koehn and R. Knowles, “Six challenges for neural machine translation,” Proceedings of the First Workshop on Neural Machine Translation (WMT), pp.28–39, Association for Computational Linguistics, Vancouver, Aug. 2017.

[8] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, “Deep learning for just-in-time defect prediction,” Proceedings of the 2015 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp.17–26, Aug. 2015.

[9] S. Wang, T. Liu, and L. Tan, “Automatically learning semantic features for defect

prediction," Proceedings of the IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp.297–308, May 2016.

[10] J. Li, P. He, J. Zhu, and M.R. Lyu, "Software defect prediction via convolutional neural network," Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp.318–328, July 2017.

[11] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI), pp.1287–1293, feb 2016.

[12] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), pp.87–98, Association for Computing Machinery, Inc, aug 2016.

[13] H.-H. Wei and M. Li, "Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code," Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI), p.30343040, 2017.

[14] P.W. McBurney and C. McMillan, "Automatic Source Code Summarization of Context for Java Methods," IEEE Transactions on Software Engineering, vol.42, no.2, pp.103–119, 2016.

[15] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," Proceedings of the 26th IEEE/ACM International Conference on Program Comprehension (ICPC), pp.200–210, 2018.

[16] U. Alon, S. Brody, O. Levy, and E. Yahav, "code2seq: Generating sequences from structured representations of code," Proceedings of the 2019 International Conference on Learning Representations (ICLR), 2019.

[17] A. LeClair, S. Jiang, and C. McMillan, "A neural model for generating natural language summaries of program subroutines," Proceedings of the 41st International Conference on Software Engineering (ICSE), p.795806, IEEE Press, 2019.

[18] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing source code using a neural attention model," Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), vol.1, pp.2073–2083, Association for Computational Linguistics, Berlin, Germany, Aug. 2016.

[19] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," Proceedings of the 41st International Conference on Software Engineering (ICSE), pp.783–794, ICSE '19, IEEE Press, 2019.

[20] K. Cho, B. vanMerriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp.1724–1734, Association for Computational Linguistics, Doha, Qatar, Oct. 2014.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol.9, no.8, p.17351780, Nov. 1997.

[22] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," Proceedings of the 3rd International Conference on Learning Representations (ICLR), 2015.

[23] T. Luong, H. Pham, and C.D. Manning, "Effective approaches to attention-based neural machine translation," Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp.1412–1421, Association for Computational Linguistics, Lisbon, Portugal, Sept. 2015.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), vol.1, pp.4171–4186, Association for Computational Linguistics, Minneapolis, Minnesota, June 2019.

[25] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training

by reducing internal covariate shift," Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML), vol.37, pp.448–456, International Machine Learning Society (IMLS), July 2015.

[26] J.L. Ba, J.R. Kiros, and G.E. Hinton, "Layer normalization," ArXiv e-prints, 2016. abs/1607.06450.

[27] D.P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Proceedings of the 3rd International Conference on Learning Representations (ICLR), dec 2014.

[28] K. Papineni，S. Roukos，T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL), pp.311–318, ACL '02, Association for Computational Linguistics, USA, 2002.

[29] B. Chen and C. Cherry, "A systematic comparison of smoothing techniques for sentence-level BLEU," Proceedings of the Ninth Workshop on Statistical Machine Translation (WMT), pp.362–367, Association for Computational Linguistics, Baltimore, Maryland, USA, June 2014.

[30] T. Mikolov, G.s Corrado, K. Chen, and J. Dean, "Efficient estimation of word representations in vector space," Proceedings of the 1st International Conference on Learning Representations (ICLR), pp.1–12, 01 2013.

[31] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, and P. Yu, "Improving automatic source code summarization via deep reinforcement learning," Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE), pp.397–407, 09 2018.

[32] National Institute of Advanced Industrial Science and Technology (AIST), ABCI, (オンライン), 入手先〈https://abci.ai/〉(参照 2020-02-01).

[33] Apache Software Foundation, camel/SmppEndpointBuilderFactory.java, (オンライン), 入手先〈https://github.com/apache/camel/blob/9f0b866922/core/camel-endpointdsl/src/main/java/org/apache/camel/builder/endpoint/dsl/

SmppEndpointBuilderFactory.java#L1912-L1925〉（参照 2020-02-08）.

[34] aosp-mirror, platform_frameworks_base/ApnSetting.java, （オンライン）, 入手先〈https://github.com/aosp-mirror/platform_frameworks_base/blob/337df70e17/telephony/java/android/telephony/data/ApnSetting.java#L569-L579〉（参照 2020-02-08）.

[35] Apache Software Foundation, hadoop/DirectoryWithSnapshotFeature.java,（オンライン）, 入手先〈https://github.com/apache/hadoop/blob/b3119b9ab6/hadoop-hdfs-project/hadoop-hdfs/src/main/java/org/apache/hadoop/hdfs/server/namenode/snapshot/DirectoryWithSnapshotFeature.java#L435-L491〉（参照 2020-02-08）.

[36] Apache Software Foundation, incubator-pinot/QuantileDigest.java, （オンライン）, 入手先〈https://github.com/apache/incubator-pinot/blob/75dbe8cf78/pinot-core/src/main/java/org/apache/pinot/core/query/aggregation/function/customobject/QuantileDigest.java#L890-L895〉（参照 2020-02-08）.

[37] Y. Wu, M. Schuster, Z. Chen, Q.V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Ł. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, “Google ’ s Neural Machine Translation System : Bridging the Gap between Human and Machine Translation,” ArXiv e-prints. abs/1609.08144v2.