International Journal of Smart Computing and Artificial Intelligence
International Institute of Applied Informatics
2020, Vol. 4, No. 2, 39 – 61

# Automatic Comment Generation for Source Code Using External Information by Neural Networks for Computational Thinking

Sakuei Onishi [*], Akiyoshi Takahashi [†],
Hiromitsu Shiina [‡], Nobuyuki Kobayashi [§]

## Abstract

To support the understanding of programs and understanding of procedures, we think need to automatically generate comments from source code. As a method, we learn the source code and comment pair by Encoder–Decoder translation model using LSTM, thereby generating comments of the source code that was the target of learning. In the Encoder–Decoder translation model, the source code of a program and its pair of comments are learned. However, that method does not know the domain of the program being used, so the comments that can be generated automatically may not be appropriate. In the use of learning, such as programs and algorithms for the purpose of computational thinking, it is possible to obtain relate information, such as problem statements and explanations. Therefore, we propose some methods to improve generated comments by using relate external information, such as question sentences. Also, we generate comments for each line and each block.

*Keywords:* Programming learning, Comment Generating, Encoder–Decoder Translation Model, TF–IDF, Distributed Representation

## 1 Introduction

With the introduction of new learning guidelines, programming education will become a compulsory subject in elementary schools and will be implemented in 2020 [1, 2]. This has led to a great deal of discussion about programming education in elementary and middle schools. Additionally, with the new learning guidelines, the emphasis will be placed on computational thinking, rather than programming itself. Many studies on programming education have been done [3]. As a related study, there is research on programming education for beginners using operation logs [4, 5]. There have been many initiatives made in programming education, and support systems using Web and card-type learning of program processing have been proposed [6]. A rubric has been proposed for the programming

[*] Graduate Shcool of Informatics, Okayama Univerity of Science, Okayama, Japan
[†] Icom Sytech, Tokyo, Japan
[‡] Okayama Univerity of Science, Okayama, Japan
[§] Sanyo Gakuen University, Okayama, Japan

education curriculum. Furthermore, for computational thinking, the description and assembly procedures, rather than the program itself, are considered important. Shinkai et al. [7] conducted a study into algorithm learning based on manual procedures.

In this study, the purpose is to assist in the understanding of programs, by automatically generating procedures from the source code, thus aiding the understanding of programs and procedures. On the other hand, natural language translation and statement generation are being introduced with the development of methods using neural networks. In particular, the Encoder–Decoder translation model [8, 9, 10] using LSTM [11] has been proposed, and this has improved translation accuracy. The method used in this study involves learning source code and comment pairs using the Encoder–Decoder translation model, and generating new source code comments and procedures. However, the generated comments will not become appropriate simply by applying LSTM to source coding and comment learning. This is because, in machine learning, the quality and quantity of the learning data affect the accuracy that follows. In applications with a limited application scope, it is difficult to secure the quantity of learning data, but there are thought to be innovations that can be used in terms of quality. In other words, if we assume the use of new networks with light quantities of data, even if we are unable to acquire a large amount of learning data, we can take the opposite approach and aim to achieve our objectives using reinforcement of related data.

In educational area for computing thinking purposes, it is possible to obtain relevant information such as problem statements and explanations for the use of program understanding and learning procedures. Therefore, we are trying to improve generated comments by using relate external information such as question sentences.

In particular, in regard to the source code and comments used in learning, the description of comments is considered to be particularly important. When creating comments for the learning data, we used C language examples and 53 problems used in lectures for first-year Information Studies students at universities, which have a low level of fluctuation. Additionally, for the neural network method, we first considered simply partially copying matching comments to similar areas in other places, but the aim was to generate comments that reconciled somewhat with the problem statements for the issues.

We propose two methods for the use of external information. Encoder–Decoder translation model consists of three parts: the part that learns the source code of the input program, the part that learns the comments that correspond to the source code and the part that generates the comments. The first method is to increase the probability of generating comments in the Encoder–Decoder translation model by increasing the probability of generating words with high TF–IDF values, which are important for words obtained from external information. The second method is to fine-tune the data of external information using Word2Vec [12], which is a distributed representation of the embedded layer created in the part that learns the comments of the Encoder–Decoder translation model.

Furthermore, when explaining programs at the university lectures covered by this study, it is thought that the explanation is given in consideration of the subject. Therefore, if the explanation used in the lecture is used for learning, it is assumed that the generated comments will be easy to match with target learners. In addition, problems regarding the covered issues and comment generation using the description of the lecture materials are considered to be very important.

We were studying to comment generation from C language by LSTM [13]. In addition, related study has been done to generate comments from Java language [14]. Since this study does not assume the program domain, it evaluates the score of automatic translation

evaluation such as BLEU [15], but the target is different from this study which is intended for educational purposes.

From the perspective of considering the learning target, code simply generated using LSMT is insufficient, and it is necessary to know about changes in the variables, particularly at the beginner level. However, with only the source code originally used for learning, there seems to be a shortage of information about variables, so the flow concerning source code variables tends to be difficult to understand.

Based on the above, in this study, we attempted to generate comments that were better and closer to the problem statement, by adding
(1) Processing that applies the influence of important terms in the problem statements for the issues to generation results, through the use of external information;
(2) Processing that reflects variable information in comment generation.

## 2 Procedure Learning System

Programming involves several procedures such as declaring variables, inputting values, calculation, and the output of results. We have been developing a system [16] for procedure learning in programming that converts source code into Japanese procedures and then rearranges the procedures to aid in understanding the concepts of the program. In the developed system, shown in Figure 1, the problem text is displayed at the top of the screen and the source code with the procedures described in Japanese is displayed in the center of the screen as an ellipse. These proceduralized ellipses can be dragged and dropped, and the problem can be solved by rearranging the steps in the gray area in the center of the screen in the correct order from the top to the bottom. After the sorting is finished, you can press the check button to check the order in which the questions are sorted. This test is utilized for understanding the learning situation in lectures.

For example, when given a foreign currency conversion problem, and told to “create a program where you can input amounts in yen, and it will convert it to US dollars, pounds, or euros”, the procedure for solving the problem can be broken down as shown in Figure 2. In this study, we have developed a system for testing the problem of rearranging this text (Figure 1).

To create the procedure automatically, it is considered to be created based on the program’s source code. Understanding the program requires an understanding of the source code line-by-line processing and an understanding of the blocks that integrate them into a somewhat cohesive processing unit. There are two types of block understanding: bottom-up and top-down. It can be said that we aim to understand bottom-up processing because it is based on the source code. In this study, we aim to understand programs’ processing by automatically generating procedures for line-by-line and block-by-block processing.

## 3 Generating Comments from Source Code Using LSTM

### 3.1 Overview of source code and comment learning, and comment generation

Encoder–Decoder translation model learning, in relation to comment generation, and the comment generation itself, consists of the following four processes(Figure 3).

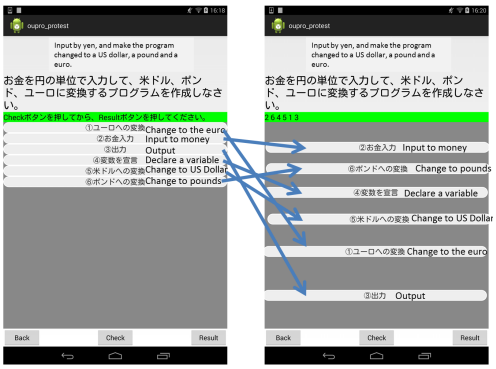


Figure 1: Sort of procedures in the Quiz

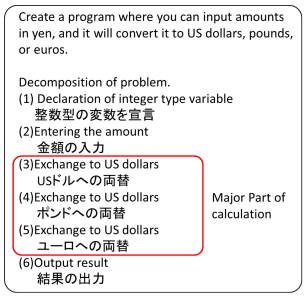


Figure 2: Test of procedure learning

(1) Learning: (1-1) Separates source code and comments from source code with comments. Case of generating comments for a line of source code, make a line-by-line source code and comment pair. On the other hand, generating comments for a block of source code makes a source code and comment pair for each block. (1-2) After processing the source code of variable names, performs Encoder–Decoder translation model learning of source code and comment pair.
(2) Comment generation probability: (2-1) Uses the Encoder–Decoder translation model in relation to the source code without any comments attached and generates the word link probability with comment generation. (2-2) Changes the important terms in the problem statements for source code issues without any comments attached to increase the comment generation word link probability and increase the influence of non-learning data information.
(3) Comment correction for the case of generating for each line of source code: (3-1) The variables in the learning source code are converted to temporary variables as preprocessing and learned. (3-2) The correspondence between variables in the source code without comments and temporary variables is stored as a list, and after generating comments by the Encoder-decoder model. The temporary variables are complemented by the original variables.

## 3.2 Encoder–Decoder translation model learning and generation

In cases where comments are already attached to the program list, it is possible to generate procedures using the summary technology in the previous section, but not all program lists

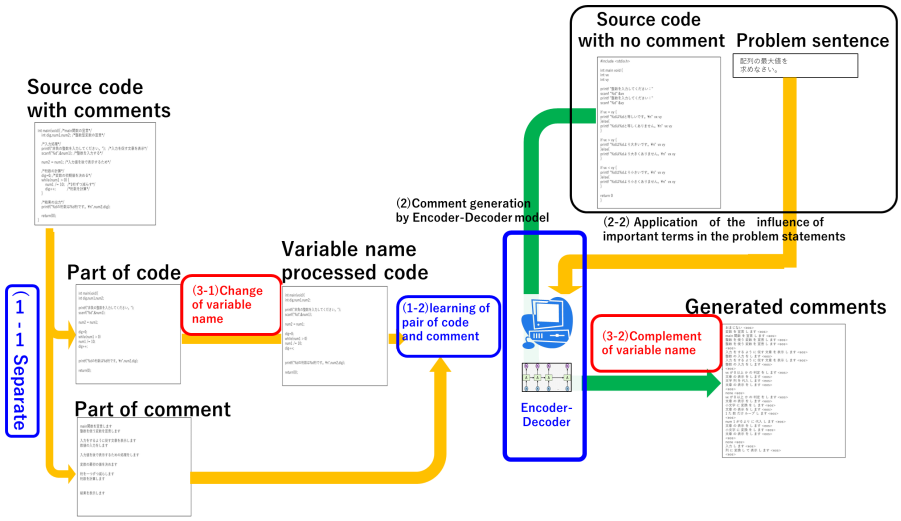


Figure 3: Comment generation flow

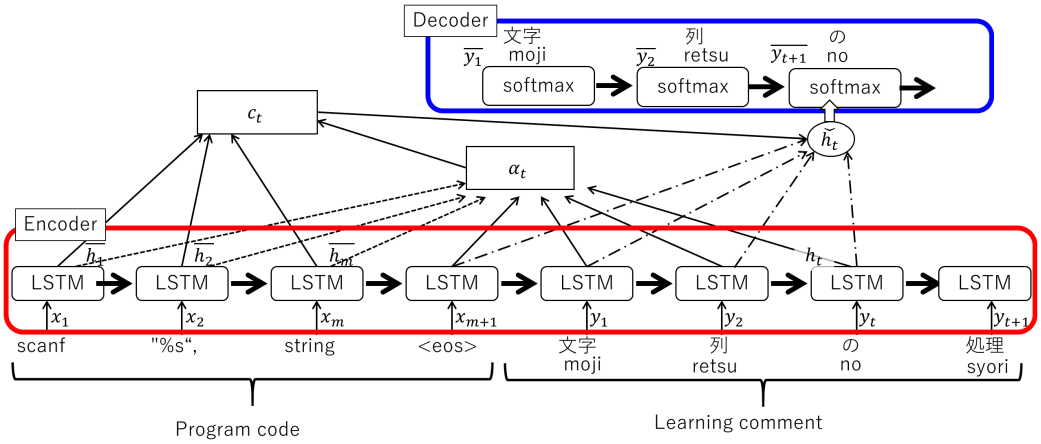


Figure 4: Source and comment learning using the Encoder–Decoder translation model

have comments attached. It is necessary, therefore, to learn from program lists to which comments are attached and generate comments from program lists without contents. There is research into the use of the Encoder–Decoder translation model using a deep learning LSTM to translate from Japanese to English or other languages. In this study, we consider the translation model as a model to translate from the program list to comments and attempt to generate comments and procedures. Furthermore, there are two kinds of comment generated. The first is the generation of comments one line at a time from the program list. The second is the generation of comments related to program block units across multiple lines, such as through if and while constructs. The generation of comments in relation to the program list is achieved by learning using the Encoder–Decoder translation model using LSTM on the program list and comment pair discussed in the previous section and generating comments for programs to which comments are not attached.

To actually generate comments using the Encoder–Ddecoder tranlation model, it is necessary to perform pre-processing on the learned data, define the Encoder–Decoder translation model, and learn the data. The overall follow of the comment generation model is shown in Figure 4.

### 3.3 Pre-processing of learning data of program code and comments.

To learn the Encoder–Decoder translation model, it is necessary to separate the input programe souce code and the procedure comments. The LSTM model uses token sequence of programe source code tokens and word sequrence of procedure comments for input as $X = x_1,\ldots x_i\ ,\ldots$ . Actually, the token is converted to the number of the token id. In a similar manner, the word converts the number of the word id.

### 3.4 Encoder–decoder translation model using LSTM

With LSTM blocks, the sigmoid layer is used, and the information that should be depended on and the information that should not be depended on are set to 0 and 1, respectively. Furthermore, for the translation pair, the error from the data saved in advance is lost. The loss for each LSTM block is accumulated. Finally, the parameters are learned by performing the error back propagation method on the loss.
(1) The source code comments are broken down into their respective vocabulary units and the word linkage relationships are learned. The source code broken down into words is input into LSTM one word at a time, and while accumulating context information, words are generated, and these are learned. With the Encoder–Ecoder model $\alpha_t$, normalization of the degree of similarity is performed for output $h_t$ in relation to $y_t$ , and $\overline{h_i}$ on the Encoder side. $c_t$ creates context vectors using the degree of similarity obtained with $\alpha_t$ and $\overline{h_i}$.
(2) Comments are generated in relation to new source code based on the learning carried out in (1). The source code is divided into words and input into LSTM. Once input is complete, the first word from the comment is output based on context information, and the next word is then generated based on the output word and the context information. LSTM output with the Encoder–Decoder translation model, $c_t$ and $h_t$ are linked to create a vector, and weight is applied with a linear operator. By applying the activation function tanh to this, the intermediate layer $\check{h_t}$ is output in relation to $y_t$ . Finally, a weight is applied with a linear operator in respect to $\check{h_t}$, and $\overline{y_{t+1}}$ is output with the softmax function.

## 4 Generation of Comments Using External Information

In the previous section, we performed learning of the program source code and comment line unit pairs, and generated comments in relation to the new source code. In this case, the information that can be used is considered to generate comments in relation to similar source code. However, it is difficult to generate source code comments in relation to issues, just using this. In a simple Encoder–Decoder translation model, even if the topics do not match, similar source code comments are copied; thus, the information may be too insufficient to generate comments that are even slightly related to those you wish to generate. In this study, to utilize information related to the problem statements of issues in comment generation, we applied the influence of important terms in the problem statements of issues, as external information, to the generation results (Figure 5). In particular, we used TF–IDF to extract important terms from the problem statements of issues. The procedure for generating comments using important terms is as follows.

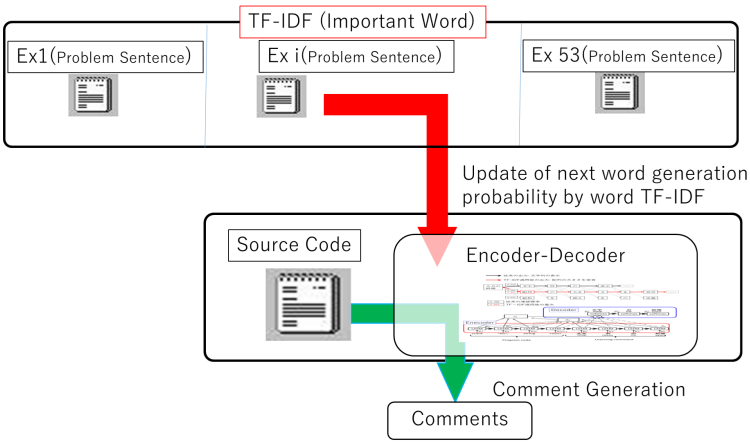


Figure 5: Relationship between problem statement and comment

The Encoder–Decoder translation model consists of three parts. The part that learns the source code of the input program, the part that learns the comments that correspond to the source code, and the part that generates the comments. Of these three processing areas, information such as teaching materials is related to the comments that are generated, so the part that learns the comments and the part that generates them are highly related. Therefore, the following method was used.

(Method 1) The first method is to increase the probability of generating comments with high TF–IDF values, which are important words obtained from external information.
(Method 2) The second method is to fine-tune the data of external information using Word2Vec, which is a distributed representation of the embedded layer created in the part that learns the comments of the Encoder–Decoder translation model.

## 4.1 Method to tune the probability of sentence generation according to the important words in TF–IDF.

The TF–IDF is used to extract important words that have a large impact on the problem sentence. The procedure for generating comments based on important words is described below.

(1)First, we load the document for the original problem as a whole, and after that, input the program problem statements, seeking the importance of each term within the problem statement related to the document as a whole using TF–IDF.
(2) We then input the source code into LSTM, and if there are words that match important terms within the problem statement when outputting the words, we multiply the generation probability by $\alpha$ times the TF–IDF, and set the highest of these numbers to the LSTM output. Table 1 shows the word TF–IDF for the following problem statements covering problem statements for the 53 issues used in the learning.

Table 1: TF–IDF for words within the problem statement

| Word | TF–IDF | Word | TF–IDF |
|---|---|---|---|
| Minimum<br>最小 | 0.45541 | Maximum<br>最大 | 0.40027 |
| Store<br>格納 | 0.38594 | Array<br>配列 | 0.32545 |
| Numeric<br>数値 | 0.24959 | SCANF<br>scanf | 0.24959 |
| SCANF<br>scanf | 0.24959 | Subscript<br>添字 | 0.24959 |
| Howerver<br>ただし | 0.2277 | IS<br>いる | 0.2277 |
| IS<br>いる | 0.2277 | Use<br>用い | 0.16841 |
| Seek<br>求める | 0.16272 | Integer<br>整数 | 0.131 |
| Integer<br>整数 | 0.131 | Output<br>出力 | 0.09667 |

Problem statement

“ numerals(integers) in a 5-item array are stored using scanf, and the maximum and minimum values sought. However, also output the subscript of the array in which the minimum and maximum values are stored. ”
配列 5 個の数値 (整数) を，scanf を用いて格納し最大値と最小値を求める．ただし，最小値と最大値の格納されている配列の添字も出力せよ．

Additionally, Figure 6 shows the image of how the “配列 (array)” with high TF–IDF values is used in comment generation, when generating comments from the folloing source code in relation to the above example of problem statements.

Source code to convert into comments

```
#define NUMBER 5
```

## 4.2 Fine-tuning the distributed representation of the part that learns the comments

In the comment learning part of the Encoder–Decoder translation model, the distributed representation, which is an embedding layer, is learned. Source code and comment pairs alone can be difficult to tune when the amount of training data is low or in a particular domain. In code generation and procedural learning for program learning, which is the subject of this study, the problem statement of the task is used as external data. It is thought that distributed representations for words generated from problem sentences can influence the overlearning of relevant information to be too close to the one-to-one of codes and comments. The procedure is described below(Figure 7).

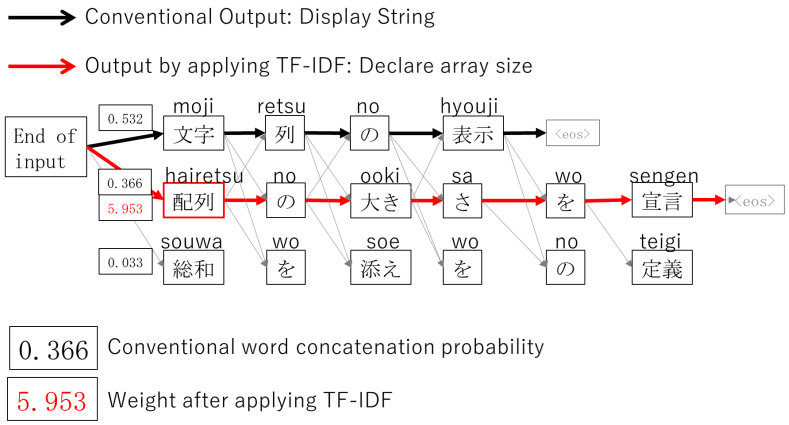


Figure 6: Differences in comment generation by applying TF–IDF

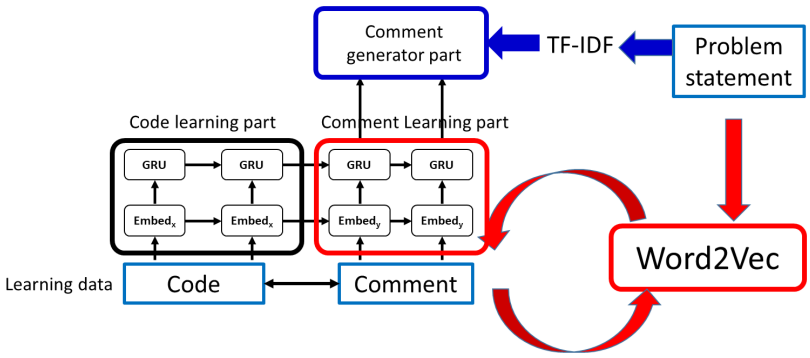


Figure 7: Overview of comment generation using Encoder–Decoder translation model

(1) We learn the Encoder–Decoder translation model from the source code and comment learning data. Next, we obtain a distributed representation of the embedded layer of the part that learns the comments.
(2) We use Word2Vec with the CBOW model for the problem sentence, with the distributed representation as the initial value. After learning, the distributed representation is returned to the embedding layer of the comment learning part of the Encoder–Decoder translation model(Figure 8).
(3) The Encoder–Decoder translation model using Word2Vec is used to generate comments for the test code.

## 5 Complementing Generated Comments with Variable Information

When generating comments from source code, the learning stage should be taken into consideration. At the stage when grammar is learned, it is necessary to understand changes in variables. However, for normal comments, the description of comments does not contain changes in variables. For this reason, there often seems to be a shortage of information about variables, which tends to make it difficult to understand the flow related to program

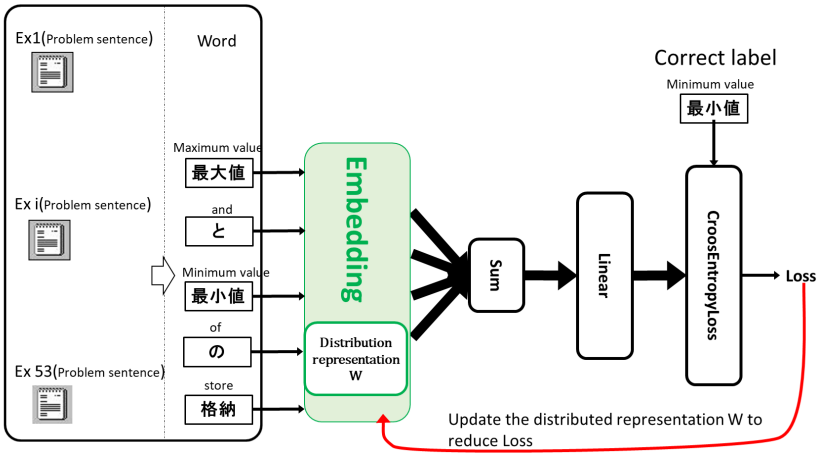


Figure 8: Learning external information with Word2Vec

variables. When processing the complements of variable names, variable processing when learning comments, and processing when generating comments, can be separately discussed. Note that provision of the variable information is intended for learners in the grammar learning stage. Therefore, we do not include variable information in the generation of procedures for each block.

(1) Variable processing when learning comments
(1-1) When processing changes in source code variable names, set the same source code variables. For example, substitute the variable min with x, and variable vx with temporary variable x.
(1-2) Add (x) as variable information for the comments corresponding to the source code.
(1-3) Learn the data for which the variable complement processing is carried out in the Encoder–Decoder translation model.
(2) Processing when generating comments
(2-1) For the source code for which the comments are generated, unify the variables to the same names as in the learning data, as part of the variable name complement processing.
(2-2) Obtain the variable information from each line of the test data, in order to apply the variable information when generating comments.
(2-3) Then, generate comments based on the learned Encoder–Decoder translation model.
(2-4) After generating comments, return the temporary variables(x) within the comments to the variables in the source code.

# 6 Evaluation of Generated Comments

The generated comments are evaluated in two types, one for each line and the other for each processing block. For the evaluation of generated comments, we used a manual questionnaire and an automatic evaluation by BLUE. A questionnaire was obtained for each generated comment on a six-point scale ((bad) 1 to 6 (good)) by six students in the laboratory.

## 6.1 Evaluation generated comments for each line

The comments generated by Encoder–Decoder tranlation model(Encoder–Decoder), the comments generated considering TF–IDF of the words in the problem sentence (+ TF–IDF), the comments with variable information completion (+ TF–IDF + Var), the examples generated by Word2Vec and the average rank of each comment are shown in Table 2, 3, 4 and 5. The average of the evaluation by method is shown in Table 6.

Table 2: Comment generation for each line by applying Encoder–Decoder

| Line | Source code | Generate comments by Encoder–Decoder | Evaluation (1-6) |
|---|---|---|---|
| 1 | `#include <stdio.h>` | 入出力を行うための宣言 | 5.667 |
| 2 | `#define NUMBER 5` | 文字列を表示する | 3.167 |
| 3 | `int main (void){` | main 関数の宣言 | 5.833 |
| 4 | `int i, j ;` | 整数型変数の宣言 | 5.667 |
| 5 | `int min, max;` | 整数型変数の宣言 | 5.667 |
| 6 | `int min_loc, max_loc ;` | 整数型変数の宣言 | 5.333 |
| 7 | `int vx [NUMBER];` | 整数型の配列の宣言 | 5.333 |
| 8 | `for(i=0;i<NUMBER; i++){` | 定義づけた回数分ループ | 5.000 |
| 9 | `printf("vx [%d]=", i);` | 内に数値を入力 | 4.167 |
| 10 | `scanf("%d",&vx [i]);` | 整数を入力 | 5.500 |
| 11 | `}` | そうでない場合 | 2.833 |
| 12 | `min=vx [0];` | min の初期値を vx[0] とする | 5.667 |
| 13 | `min_loc =0 ;` | min のあり場所を 0 | 3.833 |
| 14 | `max=vx [0];` | max の初期値を vx[0] とする | 5.667 |
| 15 | `max_loc=0 ;` | max のあり場所を 0 | 3.833 |
| 16 | `for(i=1; i<NUMBER ; i++){` | 定義づけた回数分ループ | 5.500 |
| 17 | `if(min>vx[i]){` | min の方が大きい場合 | 5.167 |
| 18 | `min=vx[i];` | 最小値の値を記憶 | 4.667 |
| 19 | `min_loc = i;` | 最小値の場所を記憶 | 4.667 |
| 20 | `}` | None | 4.833 |
| 21 | `if(max<vx[i]){` | max の方が小さい場合 | 5.167 |
| 22 | `max=vx[i];` | max はいつも一番大きく | 3.167 |
| 23 | `max_loc=i;` | max のあり場所を 0 | 3.167 |
| 24 | `}` | 行内の繰り返し | 2.667 |
| 25 | `}` | そうでない場合 | 2.667 |
| 26 | `printf ("最大値 %d , 添え字 %d\n", max, max_loc);` | 最大値と添え字を表示 | 5.500 |
| 27 | `printf ("最小値 %d , 添え字 %d\n",min, min_loc);` | 最小値と添え字を表示 | 5.500 |
| 28 | `return (0);` | 関数を終了させる | 5.333 |
| 29 | `}` | そうでない場合 | 3.167 |

Table 3: Comment generation for each line by applying TF–IDF

| Line | Source code | Generate comments by +TF–IDF | Evaluation (1-6) |
|---|---|---|---|
| 1 | `#include <stdio.h>` | 入出力を行うための宣言 | 5.833 |
| 2 | `#define NUMBER 5` | 配列の大きさを宣言 | 5.333 |
| 3 | `int main (void){` | main 関数の宣言 | 56667 |
| 4 | `int i, j ;` | 整数型変数の宣言 | 5.500 |
| 5 | `int min, max;` | 整数型変数の宣言 | 5.500 |
| 6 | `int min_loc, max_loc ;` | 整数型変数の宣言 | 5.500 |
| 7 | `int vx [NUMBER];` | 整数型の配列の宣言 | 5.500 |
| 8 | `for(i=0;i<NUMBER; i++){` | 配列内の添え字すべてに数値を入力させる | 3.500 |
| 9 | `printf("vx [%d]=", i);` | 配列のすべての添え字内に 数値を入力 | 4.333 |
| 10 | `scanf("%d",` | 整数を入力 | 5.500 |
| 11 | `}` | そうでない場合 | 3.000 |
| 12 | `min=vx [0];` | 最小値の値を記憶 | 3.333 |
| 13 | `min_loc =0 ;` | min のあり場所を 0 | 4.000 |
| 14 | `max=vx [0];` | max の初期値を vx[0] とする | 5.833 |
| 15 | `max_loc=0 ;` | max のあり場所を 0 | 4.000 |
| 16 | `for(i=1; i<NUMBER ; i++){` | max のあり場所を 0 | 5.500 |
| 17 | `if(min>vx[i]){` | min の方が大きい場合 | 5.167 |
| 18 | `min=vx[i];` | 最小値の値を記憶 | 5.167 |
| 19 | `min_loc = i;` | 最小値の場所を記憶 | 5.000 |
| 20 | `}` | None | 5.000 |
| 21 | `if(max<vx[i]){` | max の方が小さい場合 | 5.167 |
| 22 | `max=vx[i];` | 最大値の値を記憶 | 5.333 |
| 23 | `max_loc=i;` | 最大値の場所を記憶 | 5.333 |
| 24 | `}` | None | 5.000 |
| 25 | `}` | 数値を入力する | 1.833 |
| 26 | `printf ("最大値 %d , 添え字 %d\n", max, max_loc);` | 最大値と添え字を表示 | 5.667 |
| 27 | `printf ("最小値 %d, 添え字%d\n",min, min_loc);` | 最小値と添え字を表示 | 5.667 |
| 28 | `return (0);` | 関数を終了させる | 5.500 |
| 29 | `}` | 数値を入力する | 2.000 |

Table 4: Comment generation for each line by applying TF–IDF+Var

| Line | Source code | Generate comments by +TF–IDF+Var | Evaluation (1-6) |
|---|---|---|---|
| 1 | `#include <stdio.h>` | 入出力を行うための宣言 | 5.833 |
| 2 | `#define NUMBER 5` | 配列の大きさを定義づける | 5.667 |
| 3 | `int main (void){` | main 関数の宣言 | 5.833 |
| 4 | `int i, j ;` | 整数型変数 (i)(j) の宣言 ['i','j'] | 5.833 |
| 5 | `int min, max;` | 整数型変数 (min)(max) の宣言 | 6.000 |
| 6 | `int min_loc, max_loc ;` | 整数型変数 (min) の宣言 | 5.167 |
| 7 | `int vx [NUMBER];` | 整数型変数 (vx) の宣言 ['vx'] | 4.833 |
| 8 | `for(i=0;i<NUMBER; i++){` | 配列内の添え字 (i) すべてに数値を入力 ['i'] | 4.000 |
| 9 | `printf("vx [%d]=", i);` | 配列の表示 ['i','vx'] | 3.500 |
| 10 | `scanf("%d",` | 整数を入力 (i)(vx)['i','vx'] | 5.333 |
| 11 | `}` | None | 4.500 |
| 12 | `min=vx [0];` | 配列の文を表示 ['min','vx'] | 3.833 |
| 13 | `min_loc =0 ;` | 最小値の定義づけ ['min','min_loc'] | 3.000 |
| 14 | `max=vx [0];` | x の計算 ['max','vx'] | 2.500 |
| 15 | `max_loc=0 ;` | 最大値 (max) の文章を表示 ['max','max_loc'] | 3.500 |
| 16 | `for(i=1; i<NUMBER ; i++){` | 配列内の添え字 (i) すべてに数値を入力 ['i'] | 3.500 |
| 17 | `if(min>vx[i]){` | x の方が大きい場合 ['i','min','vx'] | 4.333 |
| 18 | `min=vx[i];` | 計算を行う (i)(min) ['i','min','vx'] | 3.167 |
| 19 | `min_loc = i;` | 最小値の場所 (i) を記憶 ['i','min','min_loc'] | 5.500 |
| 20 | `}` | None | 4.500 |
| 21 | `if(max<vx[i]){` | x の方が小さい場合 ['i','max','vx'] | 4.333 |
| 22 | `max=vx[i];` | 計算を行う (i)(max) | 3.8333 |
| 23 | `max_loc=i;` | 最小値の場所 (i) を記憶 ['i','max','max_loc'] | 4.000 |
| 24 | `}` | None | 4.333 |
| 25 | `}` | None | 4.833 |
| 26 | `printf("最大値%d, 添え字%d\n",max,max_loc);` | 最大値 (max) と添え字を表示 ['max','max_loc'] | 6.000 |
| 27 | `printf("最小値 %d, 添え字%d\n,min,min_loc);` | 最小値 (min) と添え字を表示 ['min','min_loc'] | 6.000 |
| 28 | `return (0);` | 関数を終了させる | 5.500 |
| 29 | `}` | None | 4.667 |

Table 5: Comment generation for each line by applying Word2Vec

| Line | Source code | Generate comments by +Word2Vec | Evaluation (1-6) |
|---|---|---|---|
| 1 | `#include <stdio.h>` | 入出力を行うための宣言 | 5.833 |
| 2 | `#define NUMBER 5` | 配列の大きさを定義づける | 5.833 |
| 3 | `int main (void){` | main 関数の宣言 | 5.833 |
| 4 | `int i, j ;` | 整数型変数の宣言 | 5.667 |
| 5 | `int min, max;` | 整数型変数の宣言 | 5.667 |
| 6 | `int min_loc,max_loc;` | 整数型変数の宣言 | 5.500 |
| 7 | `int vx [NUMBER];` | 整数型の配列の宣言 | 5.333 |
| 8 | `for(i=0;i<NUMBER; i++){` | 定義づけた回数分ループ | 5.000 |
| 9 | `printf("vx[%d]=",i);` | 配列の添え字内に数値を入力 | 4.833 |
| 10 | `scanf("%d",` | 整数を入力 | 5.333 |
| 11 | `}` | 変数 | 2.833 |
| 12 | `min=vx [0];` | min の初期値を vx[0] とする | 5.833 |
| 13 | `min_loc =0 ;` | min のあり場所を 0 | 4.000 |
| 14 | `max=vx [0];` | max の初期値を vx[0] とする | 5.833 |
| 15 | `max_loc=0 ;` | max のあり場所を 0 | 4.000 |
| 16 | `for(i=1; i<NUMBER ; i++){` | 定義づけた回数分ループ | 5.500 |
| 17 | `if(min>vx[i]){` | min の方が大きい場合 | 5.000 |
| 18 | `min=vx[i];` | 最小値の値を記憶 | 5.333 |
| 19 | `min_loc = i;` | 最小値の場所を記憶 | 5.167 |
| 20 | `}` | の繰り返し | 2.833 |
| 21 | `if(max<vx[i]){` | max の方が小さい場合 | 5.333 |
| 22 | `max=vx[i];` | max はいつも一番大きく | 3.167 |
| 23 | `max_loc=i;` | 最大値の場所を記憶 | 5.000 |
| 24 | `}` | の繰り返し | 3.333 |
| 25 | `}` | None | 4.8333 |
| 26 | `printf("最大値%d, 添え字%d\n",max,max_loc);` | 最大値と添え字を表示 | 5.667 |
| 27 | `printf("最小値%d, 添え字%d\n",min, min_loc);` | 最小値と添え字を表示 | 5.833 |
| 28 | `return (0);` | 関数を終了させる | 5.500 |
| 29 | `}` | そうでない場合 | 2.667 |

Table 6: Evaluation distribution for each line by each method

| | Rank1 | Rank2 | Rank3 | Rank4 | Rank5 | Rank6 | Average |
|---|---|---|---|---|---|---|---|
| Encoder–Decoder | 0.132223 | 0.03456 | 0.03456 | 0.114920 | 0.235641 | 0.448378 | 4.6322 |
| + TF–IDF | 0.086215 | 0.04027 | 0.04027 | 0.126422 | 0.218438 | 0.488585 | 4.816 |
| + TF–IDF + Var | 0.166729 | 0.011492 | 0.01723 | 0.155227 | 0.143725 | 0.505788 | 4.6149 |
| + Wrod2Vec | 0.097717 | 0.03446 | 0.011492 | 0.1034418 | 0.235641 | 0.517290 | 4.8966 |

Table 7: Automatic evaluation by BLEU

| Methods | BLEU |
|---|---|
| Encoder–Decoder | 0.1459 |
| + TF–IDF | 0.6313 |
| + TF–IDF + Var | 0.1376 |
| + Word2Vec | 0.72661 |

The BLEU is used as a measure of translation performance. The results of each evaluatoin are shown in Table 7. The following is a summary of evaluations of generated comments for each line.

- In the BLEU evaluation, the addition of variable information is scored poorly because the correct comment that serves as the reference does not include the variable information. In the manual evaluation, the average of the average rank of each of the four types of comments was calculated: 4.6322 for Encoder–Decoder only, 4.8161 for TF–IDF, 4.6149 for adding variable information, and 4.8966 for fine-tuning by Word2Vec. The best assessment was made using the undergraduate information by Word2Vec. For TF–IDF + var, the score of BLEU is considered to be low because the variable information is mismatched because the correct answer data does not include the variable information.

- Fore the 12th line, the Encoder-Decoder transfer model and Wor2Vec receive good evaluations in manual evaluation by questionnaire. However, authors consider the results of the TF–IDF to be good, and even the manual evaluation can cause disagreements with authors.

- Fine-tuning in Word2Vec has been rated as good overall. On the other hand, the addition of variable information was not rated as particularly bad. The result is a split between good and bad evaluations.

## 6.2 Evaluation generated comments for each block

The comments generated for block by Encoder–Decoder tranlation model(Encoder–Decoder), the comments generated considering TF–IDF of the words in the problem sentence (+ TF–IDF), the examples generated by Word2Vec(+ Word2Vec) and the average rank of each comment are shown in Table 8, 9 and 10. The average of the evaluation by method is shown in Table and 11. In addition, the evaluation by BLUE is shown in Table 12 to evaluate the comments for each block as a translation. Note that the presentation of variable information is intended for learners in the grammar learning stage. Therefore, we do not evaluate the processing of variable information in the generation of procedures for each block.

Table 8: Comment generation for each block by applying Encoder–Decoder

| Block | Source code | Generate comments by Encoder–Decoder | Evaluation (1-6) |
|---|---|---|---|
| 0 | `#include <stdio.h>` | ヘッダー処理 | 5.6667 |
| 1 | `#define NUMBER 5` | 配列の加算 | 2.3333 |
| 2 | `int main(void) {` | main 関数の宣言 | 5.8333 |
| 3 | `int i, j;`<br>`int min, max;`<br>`int min_loc, max_loc;`<br>`int vx[NUMBER];` | 整数型変数の宣言 | 4.8333 |
| 4 | `for(i=0; i < NUMBER; i++) {`<br>`printf("vx[%d]=", i);`<br>`scanf("%d", &vx[i]);`<br>`}` | 数値の入力 | 5.0000 |
| 5 | `min = vx[0];`<br>`min_loc = 0;`<br>`max = vx[0];`<br>`max_loc = 0;` | 最小値最大値を探す | 5.1667 |
| 6 | `for(i=1; i < NUMBER; i++) {`<br>`if(min > vx[i]) {`<br>`min = vx[i];`<br>`min_loc = i;`<br>`}`<br>`if(max < vx[i]) {`<br>`max = vx[i];`<br>`max_loc = i;`<br>`}`<br>`}` | 繰り返し処理 | 5.0000 |
| 7 | `printf("最大値%d, 添え字%d\n", min, min_loc);`<br>`printf("最小値%d, 添え字%d\n", max, max_loc);` | 結果の出力 | 5.5000 |
| 8 | `return(0);`<br>`}` | 関数の終了 | 6.0000 |

Table 9: Comment generation for each block by applying TF–IDF

| Block | Source code | Generate comments by +TF–IDF | Evaluation (1-6) |
|---|---|---|---|
| 0 | `#include <stdio.h>` | ヘッダー処理 | 5.5000 |
| 1 | `#define NUMBER 5` | 配列の格納 | 3.0000 |
| 2 | `int main(void) {` | main 関数の宣言 | 5.5000 |
| 3 | `int i, j;`<br>`int min, max;`<br>`int min_loc, max_loc;`<br>`int vx[NUMBER];` | 整数型変数の宣言 | 5.3333 |
| 4 | `for(i=0; i < NUMBER; i++) {`<br>`printf("vx[%d]=", i);`<br>`scanf("%d", &vx[i]);`<br>`}` | 数値の入力 | 4.8333 |
| 5 | `min = vx[0];`<br>`min_loc = 0;`<br>`max = vx[0];`<br>`max_loc = 0;` | 最小値最大値を探す | 5.1667 |
| 6 | `for(i=1; i < NUMBER; i++) {`<br>`if(min > vx[i]) {`<br>`min = vx[i];`<br>`min_loc = i;`<br>`}`<br>`if(max < vx[i]) {`<br>`max = vx[i];`<br>`max_loc = i;`<br>`}`<br>`}` | 繰り返し処理 | 4.8333 |
| 7 | `printf("最大値%d, 添え字%d\n", min, min_loc);`<br>`printf("最小値%d, 添え字%d\n", max, max_loc);` | 結果の出力 | 5.5000 |
| 8 | `return(0);`<br>`}` | 関数の終了 | 6.0000 |

Table 10: Comment generation for each block by applying Word2Vec

| Block | Source code | Generate comments by +Word2Vec | Evaluation (1-6) |
|---|---|---|---|
| 0 | `#include <stdio.h>` | ヘッダー処理 | 5.8333 |
| 1 | `#define NUMBER 5` | 配列の添え字を定義づける | 5.0000 |
| 2 | `int main(void) {` | main 関数の宣言 | 5.3333 |
| 3 | `int i, j;`<br>`int min, max;`<br>`int min_loc, max_loc;`<br>`int vx[NUMBER];` | 整数型変数の宣言 | 4.8333 |
| 4 | `for(i=0; i < NUMBER; i++) {`<br>`printf("vx[%d]=", i);`<br>`scanf("%d", &vx[i]);`<br>`}` | 数値の入力 | 5.3333 |
| 5 | `min = vx[0];`<br>`min_loc = 0;`<br>`max = vx[0];`<br>`max_loc = 0;` | 最小値最大値を探す | 5.3333 |
| 6 | `for(i=1; i < NUMBER; i++) {`<br>`if(min > vx[i]) {`<br>`min = vx[i];`<br>`min_loc = i;`<br>`}`<br>`if(max < vx[i]) {`<br>`max = vx[i];`<br>`max_loc = i;`<br>`}`<br>`}` | 繰り返し処理 | 5.0000 |
| 7 | `printf("最大値%d, 添え字%d\n", min, min_loc);`<br>`printf("最小値%d, 添え字%d\n", max, max_loc);` | 結果の出力 | 5.3333 |
| 8 | `return(0);`<br>`}` | 関数の終了 | 6.0000 |

Table 11: Evaluation distribution for each block by each method

| | Rank1 | Rank2 | Rank3 | Rank4 | Rank5 | Rank6 | Average |
|---|---|---|---|---|---|---|---|
| Encoder–Decoder | 0.0555 3 | 0.0185 1 | 0.0370 2 | 0.1481 8 | 0.2037 11 | 0.5370 29 | 5.0370 |
| + TF–IDF | 0.0185 1 | 0.0370 2 | 0.0370 2 | 0.2037 11 | 0.1666 8 | 0.5370 29 | 5.0741 |
| + Wrod2Vec | 0.0000 0 | 0.0000 0 | 0.0370 2 | 0.2037 11 | 0.1481 8 | 0.6111 29 | 5.3333 |

Table 12: Automatic evaluation by BLEU

| Methods | BLEU |
|---|---|
| Encoder–Decoder | 0.6853 |
| + TF–IDF | 0.6853 |
| + Word2Vec | 0.8788 |

The following is a summary of evaluations of generated comments for each block.

- The processing block with the greatest difference in the questionnaire evaluation was block 1. Block 1 is a block defining constants that are used to declare the array size.

  Encoder-Decoder and TF-IDF generated comments contain the required word “配列 (array)” but incorrectly generate “加算 (addition)” and “格納 (storage).” However, the method using Word2Vec can correctly generate the word for the operation “定義 (define)” in addition to the target word “配列 (array).” On the average of evaluation for each block, the method with Word2Vec is also 5.3333, which is about 0.26 or more improvement compared to the other methods. The ratio of rank 1 and 2 are 0% and the ratio of rank 6 is more than 60 %.

- We trained up to epoch 74 and generated comments for each block at each epoch for the training of generated comments for block and compared three different methods. In epoch 18, the baseline Encoder–Decoder is completely correct and converges. At epoch 15, which is shown as an example for generating block comments, the +Word2Vec method is completely correct, and the BLEU is 0.8788, an improvement of about 0.19. Therefore, it is considered to accelerate the model’s convergence by adapting the distributed representation on the comment side to the domain of the problem statement using Word2Vec. Also, since the baseline method was completely correct in epoch 18, it is considered that this task with a small amount of learning data and a short comment length is not so difficult.

## 7 User Evaluation for Procedure Learning System

The proposed procedure learning system was implemented on each line and each block of source code. Then an open-ended questionnaire was conducted by the students. The following is a summary of the questionnaire.

- The advantage of the line-based learning system is that it allows us to think deeply about the process flow because the procedures are written in detail. It also allows students to learn how to use sequential, branching, and repetition in a sensible way. However, since the procedure's unit is subdivided into lines, the amount of reordering procedures is large and may be confusing when learning.

- The block-based learning systems allow us to imagine the flow of processing in abstract units, which is useful in the early learning stages. A disadvantage of block-based learning systems is that they are abstract unit procedures, so the concrete processes are a too black box. At this stage, it is considered that the block-based procedure cannot handle the nesting of blocks.

## 8 Summary and Future Tasks

In this study, as learning of the target data occurs as a pair of the source code and comments, procedure generation occurs to understand the algorithm procedure. In particular, through the use of problem statements of issues as external information, comment generation is considered to have led to a decrease in the extraction of unrelated information. In this study, problem statements of issues are used as external information and, by using information related to lecture materials, etc., it is considered that more appropriate comments can be generated. Future challenges include measurement of structural resemblance using the distributed representation of parse tree information, generation of comments that utilizes similar programs and generation of specification forms. In particular, we plan to develop controls for comment generation for blocks within blocks, which are also noted in user evaluations of the procedure learning system.

## References

[1] Ministry of Education, Culture, Sports, Science and Technology, "Elementary school programming education guide (2nd edition)," 2018. [Online] Available: http://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1403162.htm, [Accessed May. 5, 2020] (in Japanse).

[2] Ministry of Education, Culture, Sports, Science and Technology. "How to programming education at elementary school level (Summary of discussion)." 2016. [Online] Available: http://www.mext.go.jp/b_menu/shingi/chousa/shotou/122/attach/1372525.htm , [Accessed May. 5, 2029] (In Japanese).

[3] H. Kanamori, T. Tomoto and T. Akakura, "Development of a Computer Programming Learning Support System Based on Reading Computer Program," *Human Interface and the Management of Information. Information and Interaction for Learning, Culture, Collaboration and Business (HIMI) 2013*, pp. 63-69, Springer, 2013. DOI:10.1007/978-3-642-39226-9_8

[4] K. Okimoto, S. Matsumoto, S. Yamagishi and T. Kashima, "Developing a source code reading tutorial system and analyzing its learning log data with multiple classification analysis," *Artificial Life and Robotics*, Vol 22, No. 7, pp. 227-237, 2017. DOI:10.1007/s10015-017-0357-2

[5] S. Matsumoto, K. Okimoto, T. Kashima and S. Yamagishi, "Automatic Generation of C Source Code for Novice Programming Education, Human-Computer Interaction," *Theory, Design, Development and Practice 2016*, pp. 65-76, 2016. DOI:10.1007/978-3-319-39510-4_7

[6] S. Matsumoto, Y. Hayashi, T. Hirashima, "Development of a Card Operation-Based Programming Learning Sytem Focusing on Thinking between the Relations of Parts," *IEEJ, Transaction on Electronics, Information and Systems*, Vol. 138, No.8, pp.999-1010, 2018. (in Japanese)

[7] J. Shinkai, Y. Hayase, I. Miyaji, "A Trial of Algorithm Education Emphasizing Manual Procedures," In Proc. Society for Information Technology & Teacher Education International Conference 2016, pp. 113-118, 2016. (in Japanese)

[8] I. Sutskever, O. Vinyals and Q. Le, "Sequence to Sequence Learning with Neural Networks," Advances in Neural Information Processing Systems 27 (NIPS 2014), pp. 3104-3112, 2014.

[9] M. Luong, H. Pham and D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," arXiv preprint arXiv:1508.04025v5, 2015.

[10] A. Rush, S. Chopra and C. Weston, "A Neural Attention Model for Sentence Summarization," In Proc. EMNLP 2015: Conference on Empirical Methods in Natural Language Processing, pp. 379-389, 2015.

[11] K. Greff, et al., "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 28, Issue10, pp. 2222-2232, 2017.

[12] T. Mikolov, et al., "Efficient estimation of word representations in vector space,", arXiv preprint arXiv:1301.3781, pp. 1–12, 2013.

[13] A. Takahashi, H. Shiina, N. Kobayashi, "Comment Generation System for Program Procedure Learning", In Proc. International Conference on Advanced Applied Informatics 2018, pp.38–42, 2018.

[14] X. Hu, G. Li, D. Lo, Z. Jin, "Deep Code Comment Generation," IN Proc. ICPC '18, pp. 200–210, 2018.

[15] K. Papineni, S. Roukos, T. Ward, W. Zhu, "BLEU: amethod for Automatic Evaluation of Machine Translation", In Proc. 40th Annual Meeting of the Association for Computational Linguistics, pp. 311318, 2002.

[16] K. Sakane, N. Kobayashi, H. Shiina and F. Kitagawa, "Kanji Learning and Programming Support System which conjoined with a Lecture," *IEICE Technical Report*, ET2014–86, Vol. 114, No. 513, pp. 7–12, 2015.