




# Düzce University Journal of Science & Technology

Research Article

## LSTM Based Source Code Generation for Turkish Pseudo Code of Algorithm \*

 Murat İNCE <sup>a,\*</sup>

<sup>a</sup> Department of Computer Technology, Isparta VHS, Isparta University of Applied Sciences, Isparta, TÜRKİYE

\* Corresponding author's e-mail address: muratince@isparta.edu.tr

DOI: 10.29130/dubited.824799

### ABSTRACT

Algorithmic thinking and programming abilities of students is controversial and popular issue in technological education programs in schools and universities. Students that have not best mathematical and analytical background may have difficulties in learning computer programming. Moreover, learning programming is highly difficult for a single individual to establish connection between discrete pseudo code of algorithm and source code. Another problem is required time to write a piece of program code. In order to solve this problem, there are some tools that tutor students to get analyze and realize relation between pseudo code and source code. In this study, we propose a deep learning method that is Long Short Term-Memory (LSTM) based source code generator from Turkish pseudo codes. For this purpose, we used Introduction to programming course exams in vocational high school as dataset to train LSTM. When users query a Turkish pseudo code of algorithm, C# source code is generated. In order to measure success of proposed system, generated source code and instructor's source code is analyzed with text similarity methods. Results show that proposed system is useful for students to learn fundamental programming skills.

**Keywords:** Automatic code generation, Deep learning, Long short term-memory

## Türkçe Sözde Algoritma Kodu için LSTM Tabanlı Kaynak Kod Üretimi

### ÖZET

Öğrencilerin algoritmik düşünme ve programlama yetenekleri, okullarda ve üniversitelerde teknolojik eğitim programlarında tartışmalı ve popüler bir konudur. İyi bir matematiksel ve analitik geçmişe sahip olmayan öğrenciler, bilgisayar programlamayı öğrenmede zorluklar yaşayabilir. Dahası, programlama öğrenme, bireyin sözde algoritma kodu ile kaynak kodu arasında bağlantı kurması için oldukça zordur. Başka bir problem, bir program kodu yazmak için gereken zamandır. Bu problemi çözmek için öğrencilere sözde kod ile kaynak kod arasındaki ilişkiyi analiz etme ve fark etme konusunda eğitim veren bazı araçlar vardır. Bu çalışmada, Türkçe sözde kodlardan LSTM tabanlı kaynak kod üretici olan bir derin öğrenme yöntemi önerilmiştir. Bu amaçla, Uzun-Kısa Süreli Bellek (LSTM)'i eğitmek için veri seti olarak meslek yüksek okulundaki programlamaya giriş dersi sınavları kullanılmıştır. Kullanıcılar bir Türkçe sözde algoritma kodunu sorguladığında C # kaynak kodu üretilmektedir. Önerilen sistemin başarısını ölçmek için üretilen kaynak kodu ve öğretim elemanının kaynak kodu metin benzerlik yöntemleri ile analiz edilmiştir. Sonuçlar, önerilen sistemin öğrencilerin temel programlama becerilerini öğrenmeleri için yararlı olduğunu göstermektedir.

**Anahtar Kelimeler:** Otomatik kod üretimi, Derin öğrenme, Uzun kısa süreli bellek

Received: 11/11/2020, Revised: 01/01/2021, Accepted: 03/01/2021

\* This study was presented at the ICAIAME 2020 Conference.

# **I. INTRODUCTION**

Computer programs are useful and inevitable software in information technologies. Moreover, they are used in all area of the science and life such as medicine [1], agriculture [2], law [3], physics [4], education [5] etc. These programs are written by programmers and software engineers. Education of programming is controversial and also has some difficulties [6]. There are a lot of ways to learn programming such as online platforms, books, undergraduate and graduate programs at colleges and universities. Especially, academic education of how to write a program is sometimes difficult because some students have not mathematical background and analytical thinking skills [7]-[9]. Generally, students follow step by step instructions that is called pseudo code and convert them to source code with a programming language. In this process, understanding how to use and where to use variables, loops etc is can be taught by the auxiliary tools or systems. By using these tools, students can compare their codes and also learn how to convert pseudo code to source code. These tools are called as automatic code (program) generator [10].

Automatic code generation is a difficult problem. In order to overcome this issue, there are several studies that uses different methods such as natural language processing (NLP) [11],[12], synthesizing methods [13], probabilistic methods [14],[15] and deep neural networks [16],[17]. Manshadi et al. [18] used probabilistic model and NLP to generate source code with very few training data. These studies are Programming with NLP (PBNL) examples. Also, there are Programming by Example (PBE) approach [19],[20]. Another study is compositional synthesis framework based on NLP [21]. Lei et al. [22] use Bayesian generative model and NLP techniques to generate source code from English input specifications. Case based reasoning is another method to automated code generation [23]. Algosmart is another classical source code generator based on tagged XML documents [24]. In addition, there are deep learning methods to generate source code [25],[26]. In the proposed study, we developed source code generator for Turkish pseudo codes by using both NLP and deep learning methods. Thus, success of the system is increased.

The rest of the article is organized as follows; in section 2, used methods are explained. In section 3, application of the proposed system is explained. Evaluation of the calculations are discussed in section 4 and results are given in the last section.

## **II. METHODS**

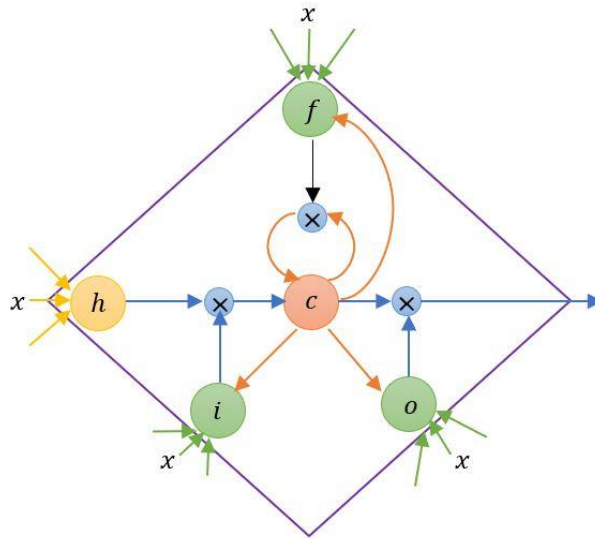
### **A. NATURAL LANGUAGE PROCESSING AND LATENT SEMANTIC ANALYSIS**

Natural language processing (NLP), one of the popular applications of artificial intelligence, is the analysis, understanding and reconstruction of language according to certain rules [27]. NLP operations mostly include text preprocessing, morphological analysis, syntactic analysis and semantic analysis sections [28]. In morphological analysis, it includes the processes of finding the roots by separating the suffixes of the words. Thus, it is determined which type the word belongs to, such as noun, adjective and verb [29]. In the syntactic analysis part, the usage purposes of the words in the sentence such as subject, object, adverb are determined [30]. The use of these words in different places and numbers in the sentence may have different meanings. Semantic analysis, on the other hand, examines the association of discrete words with appropriate objects [31]. It is provided to reveal the semantic connections of the words and to reveal the concept maps. Thus, words such as synonyms and close meanings are determined [32]. After these operations, text normalization is performed. To do this, natural language processing tools such as NLTK [33] and Zemberek [34] are used to convert capital letters, convert numbers to text, remove punctuation and markups, remove spaces, open abbreviations and remove unnecessary words [35]. When there are a lot of texts to process, sometimes it is necessary to decrease the amount of the texts. For these reasons, Latent Semantic Analysis (LSA) which is a statistical method for information retrieval problems [36], is used for elimination of irrelevant texts and

also comparison of texts based on corpus. Thus, texts are represented as vectors [37] that if the meaning of individual words is in the related corpus or not. The basic aim of LSA is to determine the similarity between words and documents by computing the similarity of corresponding vector representations. Singular value decomposition (SVD) method is used in LSA to calculate statically term and document co-occurrence information to decrease document size.

## B. LONG SHORT-TERM MEMORY

Deep learning algorithms are used frequently in many areas such as image processing, classification, and natural language processing [38]. These methods, which we can also call deep learning networks, differ from classical artificial neural networks in several ways, such as the application of layer numbers [39]. Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) are the most known deep learning algorithms [40],[41]. RNNs are able to process input sequences of arbitrary length and time series problems. In training phase of long sequence RNN problems, gradient vector component may grow or decay [42]. So that can cause gradient vanishing problems and can cause learning problems to find correct relations in the sequences of the RNN model. Overwhelming of this issue, LSTM, a specific version of classical RNN, is developed. LSTM is used in areas such as handwriting recognition, voice recognition [43],[44]. Success of the LSTM on long sequences is based on its memory cell [45]. Memory cell ( $c$ ) can preserve state over long periods of time and consists of an input gate ( $i$ ) to decide whether an input data be prevented or conveyed to the memory cell, an output gate ( $o$ ) to produce or prevent an output itself by repetitive links in two sequential time steps ( $t$ ), a forget gate ( $f$ ) to decide recall or omit previous cell (Figure 1).



*Figure 1. LSTM memory cell [45]*

Sigmoid gates are used to control read and write process of the memory cell. At given time step  $t$ , LSTM get inputs: current input  $x_t$ , the previous hidden state of all LSTM units  $h_{t-1}$  and previous memory cell state  $c_{t-1}$ . At this time step, gates are updated for  $x_t$ ,  $h_{t-1}$  and  $c_{t-1}$  as follows [45];

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad (1)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad (2)$$

$$o_t = \sigma(W_{oi}x_t + W_{ho}h_{t-1} + b_o) \quad (3)$$

$$g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (4)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (5)$$

$$h_t = o_t \odot \phi(c_t) \quad (6)$$

where the produced weight matrices are  $W$  and bias vectors are  $b$ . The sigmoid activation function is presented with  $\sigma(x) = 1/(1 + \exp(-x))$  and hyperbolic tangent is presented with  $\phi(x) = (\exp(x) - \exp(-x)) / (\exp(x) + \exp(-x))$  equations.  $\odot$  symbol represents the dot products of vectors. The hidden output with  $K$  different possible outcomes;  $h_t = \{h_{tk}\}_{k=0}^K, h_t \in R^K$  will be used to predict next word by softmax function with parameters  $W_s$  and  $b_s$  [45]:

$$F(p_{ti}; W_s, b_s) = \frac{\exp(W_s h_{ti} + b_s)}{\sum_{j=1}^K \exp(W_s h_{tj} + b_s)} \quad (7)$$

where  $p_{ti}$  is the estimated word probability.

### C. EVALUATION METRICS

Texts created or translated in natural language processing methods should be evaluated. One of the frequently used methods for this is BLEU [46]. In this method, the similarity between the desired reference text and the produced or translated text to be tested is examined [47]. If the similarity between texts is close, the value converges to 1, and if the similarity is distant, it approaches 0. While doing this, it is calculated according to the frequency of how many times the n-grams between the reference text and the text tested. In the proposed study, each similarity metric is tested on same parameters, respectively. For this purpose, mscocoapi [48] that can be downloaded from github was used.

Mean Absolute Error (MAE) is defined as average of absolute variations between target values and estimations [49]. MAE is a linear score, that is, all singular variations are on average equivalent weight. Since the MAE value is easy to interpret, it is frequently used in regression and time series problems and it is used for measuring average errors ( $e$ ) in a range of predictions ( $n$ ) without considering direction (Equation 8).

$$MAE = \frac{1}{n} \sum_{j=1}^n |e_j| \quad (8)$$

The MAE value can range from 0 to  $\infty$  but it is desired to have small value. For example, predictions with lower values shows better performance.

## III. APPLICATION OF PROPOSED CODE GENERATION SYSTEM

The proposed method uses LSTM to process user query inputs as problem statements to produce corresponding source code. The general working process of the proposed system is that, firstly, characters are read one by one from input up to the end of sequence in each time step. Same process is applied to the output with a softmax layer. This layer counts the occurrence probabilities of each word to produce a vector. The vector that has highest probability is selected as output and sent to the model as an input at the next time step.

In the system, users can query problem statements in Turkish. Turkish is an agglutinative language so it is difficult to process according to English [50]. When an input is given as raw text, words such as synonyms and close meanings are determined and then text normalization is performed by using NLP techniques. For this purpose, Zemberek is used to convert capital letters, convert numbers to text, remove punctuation and markups, remove spaces, open abbreviations and remove unnecessary words. The dataset of the system is gained from last 10 years' exams of the "Introduction the Algorithm and

Programming” lesson in associate degree. 8 basic problem statement is defined from 72 students’ exam papers. Thus,  $72 \times 10 = 720$  source code is prepared for each problem statement. LSA is implemented on these source codes with reference correct source code (instructor’s answer key) to eliminate unrelated data because the students’ answers can be incorrect. So, a threshold value that is 0.7 is defined to eliminate the low successful papers. Exam papers that higher similarity value than threshold for each problem statement are selected (Table 1).

*Table 1. Number of total source codes related with pseudo code statements*

<b>Pseudo Code Statement</b>	<b>Total source code</b>
1	630
2	520
3	470
4	420
5	380
6	510
7	450
8	350

Based on the most similar base problem statement, the system retrieved the representation of user input in structured format. After being preprocessed, each word in the structured text was converted into its index which was in numerical format to mapping corresponding source code. Each of the character of the context statement will be processed in the input layer as the input sequence. Since the learning process is the type of supervised learning, we need to include the corresponding source code as the output sequence. After the last character of input sequence, the model will start to generate the output sequence by using softmax layer which is described in the previous component.

The network used 3 LSTM layers with 128 units in each layer. Training iterations size is 50 epochs. After each LSTM layer, a Dropout layer with probability 0.2 was added. The probability 0.2 means that one in five inputs will be randomly excluded from each cycle of updating process. Dropout is a regularization technique to prevent overfitting. The basic concept is this technique randomly select neurons to be ignored during training. When a neuron is dropped out, it cannot contribute to the network and receive some values in the backpropagation process. The next layer was Dense layer with the total number of words in the vocabulary as the dimension. The last layer used softmax as the activation function. The output of this function is modelled as the probability distribution over K different possible outcomes. Specifically, it shows the probability of each word in the vocabulary to be the most suitable output for the specific element in the output vector.

## **IV. EVALUATION AND DISCUSSION**

Effectiveness of the system is tested with 8 pseudo codes to generate (predict) C# source code. Each generated source code is compared with reference source code that is provided by instructor. Similarity of the source codes are scored with BLEU (Table 2). Moreover, testing errors of the algorithm is measured with MAE to evaluate the accuracy. Generated codes may include some errors so they must be corrected before the compiling. Maximum number of the correction in the test is 6 errors.

*Table 2. BLEU and MAE scored of the tested pseudo code statements*

<b>Pseudo Code Statement</b>	<b>BLEU Score</b>	<b>MAE</b>	<b>Number of Corrections</b>
1	0.863	0.152	0
2	0.817	0.168	2
3	0.687	0.184	3
4	0.542	0.198	5
5	0.649	0.191	4
6	0.721	0.178	5
7	0.803	0.161	3
8	0.482	0.229	6

Example pseudo code statement 1 (Figure 2) is tested and source code is generated with no error (Figure 3). So, it can be compiled and run directly with any C# development tool. Generated code may include different styled variable names such as “s1” and “sayi (in English: number)” to hold reading input numbers.

- 1- Start
- 2- Read X
- 3- Read Y
- 4- Sum=X+Y
- 5- Write Sum
- 6- Stop

*Figure 2. Pseudo code statement example 1*

```
int s1 = int.Parse(Console.ReadLine());
int sayi = int.Parse(Console.ReadLine());
int top = s1 + sayi;
Console.WriteLine(top);
```

*Figure 3. Generated source code with no error for pseudo code statement example 1*

Example pseudo code statement 2 (Figure 4) is tested and source code is generated with two errors (Figure 5). Firstly, variable “i” is defined twice time in the same code block. Other one is starting value of the variable “i” is 0 in the for loop. However, in the pseudo code, “i” is start from 1. After correcting these errors, source code can be compiled and run.

- 1- Start
- 2- I=1
- 3- Write I
- 4- I=I+1
- 5- If I<=100 then go to 3
- 6- Stop

*Figure 4. Pseudo code statement example 2*

```
int i = 1;
for (int i = 0; i < 100; i++)
{
    Console.Write(i);
}
```

Figure 5. Generated source code with two errors for pseudo code statement example 2

## V. CONCLUSION

Learning programming is difficult because there are a lot of discrete concepts. Lack of algorithmic thinking and programming abilities of programmers that have not best mathematical and analytical background may cause difficulties in learning computer programming. Thus, writing a piece of program code may be time consuming for beginner programmers. For these reasons, automatic source code generator is developed for Turkish pseudo codes to help programmers to analyze and realize some concepts. For this process, LSTM, a deep learning method, is used to generate C# source code for example pseudo codes. Proposed system success is evaluated MAE and BLEU score and results showed that source codes can be automatically generated with the help of NLP and deep learning methods successfully. Users can use generated codes with software development environments (editors) to investigate the corresponding output program. Thus, students have not mathematical background and analytical thinking skills can compare their codes and understand how to convert pseudo code to source code, how to use and where to use variables, loops etc by the help of the proposed study. In the future, by using this proposed system architecture, automatic content generators can be developed for other programming languages. Also, the proposed system can be integrated with software development environments by using services and APIs.

## VI. REFERENCES

- [1] B. Robson, "Computers and viral diseases. Preliminary bioinformatics studies on the design of a synthetic vaccine and a preventative peptidomimetic antagonist against the SARS-CoV-2 (2019-nCoV, COVID-19) coronavirus," *Computers in Biology and Medicine*, vol. 119, pp. 103670, 2020.
- [2] B. Drury, M. Roche, "A survey of the applications of text mining for agriculture," *Computers and Electronics in Agriculture*, vol. 163, pp. 104864, 2019.
- [3] R. Warner, S. D. Sowle, and W. Sadler, "Teaching law with computers," *Rutgers Computer & Tech*, vol. 24, no. 107, pp. 156-158, 1998.
- [4] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, pp. 467-488, 1982.
- [5] M. Duran, T. Aytay, "Students' opinions on the use of tablet computers in education," *European Journal of Contemporary Education*, vol. 15, no. 1, pp. 65-75, 2016.
- [6] Y. Qian, J. Lehman, "Students' misconceptions and other difficulties in introductory programming: a literature review," *ACM Transactions on Computing Education (TOCE)*, vol. 18, no. 1, pp. 1-24, 2017.
- [7] E. Lahtinen, K. Ala-Mutka and H. M. Järvinen, "A study of the difficulties of novice programmers," *Acm Sigcse Bulletin*, vol. 37, no. 13, pp. 14-18, 2005.

- [8] P. H. Tan, C. Y. Ting, S. W. Ling, "Learning difficulties in programming courses: Undergraduates' perspective and perception," in *International Conference on Computer Technology and Development*, 2009, pp. 42-46.
- [9] V. Renumol, S. Jayaprakash, and D. Janakiram, "Classification of cognitive difficulties of students to learn computer programming," *Indian Institute of Technology*, vol. 12, pp. 1-12, 2009.
- [10] M. Egea, C. Dania, "SQL-PL4OCL: An automatic code generator from OCL to SQL procedural language," *Software & Systems Modeling*, vol. 18, no. 1, pp. 769-791, 2019.
- [11] M. Allamanis, D. Tarlow, A. Gordon, and Y. Wei, "Bimodal modelling of source code and natural language," in *International Conference on Machine Learning*, 2015, pp. 2123-2132.
- [12] M. Raghthaman, Y. Wei, and Y. Hamadi, "Swim: synthesizing what i mean-code search and idiomatic snippet synthesis," in *IEEE/ACM International Conference on Software Engineering (ICSE)*, 2016, pp. 357-367.
- [13] J. Galenson, P. Reames, R. Bodik, B. Hartmann, and K. Sen, "Codehint: dynamic and interactive synthesis of code snippets," in *International Conference on Software Engineering*, 2014, pp. 653-663.
- [14] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "A statistical semantic language model for source code," in *Joint Meeting on Foundations of Software Engineering*, 2013, pp. 532-542.
- [15] C. Maddison, and D. Tarlow, "Structured generative models of natural source code," in *International Conference on Machine Learning*, 2014, pp. 649-657.
- [16] E. Parisotto, A. R. Mohamed, R. Singh, L. Li, D. Zhou, and P. Kohli, "Neuro-symbolic program synthesis," 2016. [Online]. Available: arXiv:1611.01855.
- [17] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, "Deepcoder: learning to write programs," 2016. [Online]. Available: arXiv:1611.01989.
- [18] M. H. Manshadi, D. Gildea, and J. F. Allen, "Integrating programming by example and natural language programming," in *AAAI Conference on Artificial Intelligence*, 2013, pp. 661-667.
- [19] H. Lieberman, *Your Wish is my Command: Programming by Example*, Burlington, Massachusetts, USA: Morgan Kaufmann Publishers, 2001.
- [20] S. Gulwani, W. R. Harris, and R. Singh, "Spreadsheet data manipulation using examples," *Communications of the ACM*, vol. 55, no. 8, pp. 97-105, 2012.
- [21] M. Raza, S. Gulwani, and N. Milic-Frayling, "Compositional program synthesis from natural language and examples," in *International Joint Conference on Artificial Intelligence*, 2015, pp. 792-800.
- [22] T. Lei, F. Long, R. Barzilay, and M. Rinard, "From natural language specifications to program input parsers," in *Annual Meeting of the Association for Computational Linguistics*, 2013, pp. 1294-1303.
- [23] Y. Danilchenko, and R. Fox, "Automated code generation using case-based reasoning, routine design and template-based programming," in *Midwest Artificial Intelligence and Cognitive Science Conference*, 2012, pp. 119-125.
- [24] S. Mukherjee, T. Chakrabarti, "Automatic algorithm specification to source code translation," *Indian Journal of Computer Science and Engineering (IJCSE)*, vol. 2, no. 2, pp. 146-159, 2011.



- [25] L. Mou, R. Men, G. Li, L. Zhang, and Z. Jin, "On end-to-end program generation from user intention by deep neural networks," 2015. [Online]. Available: arXiv:1510.07211.
- [26] X. Chen, C. Liu, and D. Song, "Tree-to-tree neural networks for program translation," in *Advances in Neural Information Processing Systems*, 2018, pp. 2547-2557.
- [27] V. V. Nabiyev, *Yapay Zeka*, 4. baskı, Ankara, Türkiye: Seçkin Yayıncılık, 2012.
- [28] M. H. Stefanini, Y. Demazeau, "TALISMAN: A multi-agent system for natural language processing," in *Brazilian Symposium on Artificial Intelligence*, 1995, pp. 312-322.
- [29] S. Sun, C. Luo, and J. Chen, "A review of natural language processing techniques for opinion mining systems," *Information fusion*, vol. 36, pp. 10-25, 2017.
- [30] T. Strzalkowski, F. Lin, J. Wang, and J. Perez-Carballo, "Evaluating natural language processing techniques in information retrieval," in *Natural Language Information Retrieval*, Dordrecht: Springer, 1999, pp. 113-145.
- [31] T. Nasukawa, J. Yi, "Sentiment analysis: Capturing favorability using natural language processing," in *International Conference on Knowledge Capture*, 2003, pp. 70-77.
- [32] Y. Aktaş, E. Y. İnce, and A. Çakır, "Doğal dil işleme kullanarak bilgisayar ağ terimlerinin wordnet ontolojisinde uyarlanması," *Teknik Bilimler Dergisi*, vol. 7, no. 2, pp. 1-9, 2017.
- [33] J. Cushing, R. Hastings, "Introducing computational linguistics with NLTK (natural language toolkit)," *Journal of Computing Sciences in Colleges*, vol. 25, no. 1, pp. 167-169, 2009.
- [34] S. Savaş, N. Topaloğlu, "Data analysis through social media according to the classified crime," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 27, no. 1, pp. 407-420, 2019.
- [35] E. Y. İnce, "Spell checking and error correcting application for Turkish," *International Journal of Information and Electronics Engineering*, vol. 7, no. 2, pp. 68-71, 2017.
- [36] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391-407, 1990.
- [37] S. T. Dumais, "Latent semantic analysis," *Annual Review of Information Science and Technology*, vol. 38, no. 1, pp. 188-230, 2004.
- [38] L. Deng, D. Yu, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3, pp. 197-387, 2014.
- [39] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.
- [40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097-1105.
- [41] J. Salamon, J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279-283, 2017.
- [42] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157-166, 1994.

- [43] A. Graves, and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2009, pp. 545-552.
- [44] T. Hughes, and K. Mierle, "Recurrent neural networks for voice activity detection," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 7378-7382.
- [45] C. Wang, H. Yang, C. Bartz, and C. Meinel, "Image captioning with deep bidirectional LSTMs," in *ACM International Conference on Multimedia*, 2016, pp. 988-997.
- [46] M. N. Al-Kabi, T. M. Hailat, E. M. Al-Shawakfa, and I. M. Alsmadi, "Evaluating English to Arabic machine translation using BLEU," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 1, 2013.
- [47] S. Stoll, N. C. Camgoz, S. Hadfield, and R. Bowden, "Text2Sign: Towards sign language production using neural machine translation and generative adversarial networks," *International Journal of Computer Vision*, vol. 128, pp. 891–908, 2020.
- [48] T. Y. Lin, and P. Dollar. (2016, Feb 14) *Mscocoapi* [Online]. Available: <https://github.com/cocodataset/cocoapi>.
- [49] W. B. Langdon, J. Dolado, F. Sarro, and M. Harman, "Exact mean absolute error of baseline predictor MARP0," *Information and Software Technology*, vol. 73, pp. 16-18, 2016.
- [50] E. Arısoy, H. Dutağacı, and L. M. Arslan, "A unified language model for large vocabulary continuous speech recognition of Turkish," *Signal Processing*, vol. 86, no. 10, pp. 2844-2862, 2006.