Journal of Computer Science

Original Research Paper

A Backpropagation Neural Network for Splitting Identifiers

[1]Ahmed Sabah Ahmed AL-Jumaili, [2]Huda Kadhim Tayyeh and [2]Ruqaia Jawad Kadhem

[1]*Department of Business Information Technology (BIT), College of Business Informatics, University of Information Technology and Communications, Baghdad, Iraq*
[2]*Department of Informatics Systems Management (ISM), College of Business Informatics, University of Information Technology and Communications, Baghdad, Iraq*

Article history
Received: 23-07-2018
Revised: 24-09-2018
Accepted: 29-10-2018

Corresponding Author:
Huda KadhimTayyeh,
University of Information Technology and Communications, Baghdad, Iraq
Email: haljobori@uoitc.edu.iq

Abstract: Splitting identifiers is a task that has been addressed in the past few years in order to contribute toward improving the Feature Location task. Feature Location aims at determining the exact position of a specific feature within a source code. Several research studies have addressed the process of splitting multi-word identifiers. However, one of the endure gaps that still face the use of machine learning lies on using probabilistic algorithms which may seem insufficient compared to other sophisticated algorithms such as the Backpropagation Neural Network (BPNN). Therefore, this paper proposes a BPNN for the splitting identifiers task. A benchmark of source code dataset has been used in the experiments. In addition, different objective functions have been used including Tanh, Sigmoid and Softmax. Results showed that Softmax has outperformed the other objective funciton by achieving a 71.4% of f-measure. This results implies the usefulness of BPNN in terms of handling character-based problems.

Keywords: Splitting Identifiers, Feature Location, Source Code Mining, Neural Network, Backpropagation

Introduction

With the dramatic expansion of information over web, nowadays, numerous websites are providing vast amount of source code to support developers (Alshaikhdeeb and Ahmad, 2015). Splitting identifiers is a task that has been addressed in the past few years in order to contribute toward improving the Feature Location task. Feature Location aims at determining the exact position of a specific feature within a source code (Thomas, 2011). Generally, analyzing source codes have been considered in the software engineering literature by proposing approaches that may facilitate the process of re-using a source code (Sjøberg *et al*., 2005). This can be represented by improving the efficiency of a source code for specific system.

Any software code would definitely contain identifiers which are the entities used to declare a feature (Feild *et al*., 2006). Assume a system that is intended to manage a college, one of the important identifiers that could be declared is the student. Such student would have an Id which can be declared as 'intStudentId', as well as, a name which also can be declared as 'string StudentName'. However, declaring these identifiers is mainly depending on the developer who may use different strategies to separate the multi-words. For instance, developer may use CamelCase such as 'StudentId', special characters such as 'Student_Id', or even an abbreviated form such as 'StudId' (Lawrie *et al*., 2007). Since there is no a standard way to declare these identifiers therefore, the multi-word identifiers are still a significant challenge face the feature location task (Lawrie and Binkley, 2011). In this vein, the need of separating such multi-word identifiers is imperative.

Several research studies have addressed the process of splitting multi-word identifiers. The state of the art of these studies have concentrated on machine learning techniques especially with the emergence of some benchmark datasets that contain both the multi-word identifiers associated with their appropriate separation. Machine learning aims to build a statistical model based on example data or historical data for the purpose of classifying new and unseen data (Alshaikhdeeb and Ahmad, 2016; 2017). However, one of the endure gaps that still face the use of machine learning lies on using probabilistic algorithms which may seem insufficient compared to other sophisticated algorithms such as the Backpropagation Neural Network (BPNN). BPNN has been addressed in similar tasks such retrieving roots of words where the terms are being stemmed to their actual root (Mezher and Omar, 2015; Al-Serhan, 2008). Hence, this study aims to examine the capability of BPNN for the task of splitting identifiers.

Related Work

Different research efforts have been proposed for the process of feature location. Maletic and Marcus (2001) proposed a method for feature location using Latent Semantic Indexing (LSI). The proposed method aimed at identifying the matches within the source code in order to classify its components. One of the significant components that the authors have focused on is the identifiers. LSI has been applied on the source code in order to identify the similar identifiers. Furthermore, Poshyvanyk *et al.* (2005) proposed a search tool for the .Net framework programming environment. The proposed tool has been inspired by the Natural Language Processing (NLP) tasks aiming to find the similar portions of the code in accordance with the query posted by the developer.

As mentioned earlier, one of the serious limitations face the feature location task is the multi-word identifiers therefore, some researchers have attempted to introduce approaches that have the ability to separate such identifiers. Lawrie *et al.* (2007) presented a method for splitting identifiers automatically. The proposed method has utilized the regular expression in order to address the punctuations, numbers and capitalized letters within the multi-word identifiers.

On the other hand, Lawrie *et al.* (2007) has presented a method for splitting identifiers based on dictionary. In fact, the authors have used a dictionary that contains the splitted identifiers. Hence, a lexical similarity has been performed between the dictionary contents and the source code in order to extract the similar identifiers.

Enslen *et al.* (2009) presented a statistical method for splitting identifiers using term frequency. The idea behind such method lies on the possibility of frequent occurrence of a particular identifier. For example, the identifier 'Patient' would be frequently occurred with other words such as 'PatientID', 'PateintName' and 'PatientAge'. In this manner, addressing the frequent occurrence of such identifier may help the process of splitting its instances.

Lawrie and Binkley (2011) presented a normalization method for the vocabulary within the source code. The proposed method was intended to extend the abbreviations with their corresponding words. Consequentially, a matching process has been conducted between the source codes with a dictionary in order to perform the splitting.

Recently, more interests have been shown toward utilizing the machine learning techniques in the process of splitting the identifiers. Basically, machine learning techniques have showed superior performance regarding to its powerful mechanism of learning from examples. An example data of the multi-word identifiers along with their splitting form can be used by the machine learning to build a model that has the ability to separate new data.

Based on such concept, Alanee and Murad (2017) has proposed a machine learning method based on Naïve Bayes (NB). The authors have utilized different textual features along with the NB in order to split identifiers. These features consist of camel case, numbering, punctuations and spell checker.

The Proposed BPNN

Basically, the proposed method of this paper is composed of different phases as shown in Fig. 1. Such phases discuss the data used in the experiments. In addition, the preprocessing tasks that have been conducted to make the BPNN works properly. Finally, carrying out the BPNN.

Next sub-sections will discuss each phase in further detail.

Dataset

Apparently, addressing the multi-word identifiers requires acquiring a benchmark dataset that contains massive source code syntaxes. In this manner, the dataset of Enslen *et al.* (2009) has been used in this paper. This dataset is composed of numerous source code syntaxes written by different programming languages such as Java, C and C++. The dataset has been processed using experts in the domain of feature location in order to annotate each multi-word identifiers with its actual separation. Therefore, the dataset has been formed with different features such as the type of programming language, type of browsers that extract such code, actual separation of identifiers and the number of characters needed to separate the identifiers. Figure 2 shows a sample of such dataset.

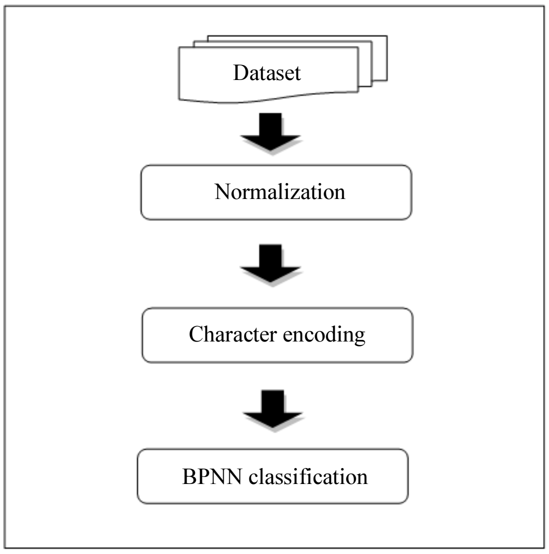

Fig. 1: Proposed method

1 CreateProcesscpp mozilla-source-1.1 Create-Process 1 Create-Process 1
2 DrawThemeTabcpp mozilla-source-1.0 Draw-Theme-Tab 1 Draw-Theme-Tab 2
3 GetFrontWindowOfClasscpp mozilla-source-1.0 Get-Front-Window-Of-Class 1 Get-Front-Window-Of-Class 4
4 GetPrivateProfileSectioncpp mozilla-source-1.0 Get-Private-Profile-Section 1 Get-Private-Profile-Section 3
5 GetScriptManagerVariablecpp mozilla-source-1.0 Get-Script-Manager-Variable 1 Get-Script-Manager-Variable 3
6 GetTokenAtcpp mozilla-source-1.0 Get-Token-At 1 Get-Token-At 2
7 JS_DefineUCPropertycpp mozilla-source-1.0 JS-Define-UC-Property 1 JS-Define-UC-Property 3
8 SetMenuItemHierarchicalIDcpp mozilla-source-1.0 Set-Menu-Item-Hierarchical-ID 1 Set-Menu-Item-Hierarchical-ID 4
9 SetResLoadcpp mozilla-source-1.0 Set-Res-Load 1 Set-Res-Load 2
10freadcpp eMule0.46c-Sources fread 1 fread 0

Fig. 2: Sample of the dataset

Table 1: Output of normalization phase

Identifier	Class (required number of separation
CreateProcess	1
DrawThemeTab	2
GetFrontWindowOfClass	4
GetPrivateProfileSection	3
GetScriptManagerVariable	3
GetTokenAt	2
JS_DefineUCProperty	3
SetMenuItemHierarchicalID	4
SetResLoad	2
fread	0

Normalization

Basically, the data has different attributes that are irrelevant to the BPNN. In contrast to other machine learning techniques that may need these attributes to construct a statistical model, BPNN in this study will work by addressing the characters encoding of the identifiers. Therefore, normalization task aims to filter out the irrelevant attributes, meanwhile, maintaining the identifier attribute and class label attribute (i.e., which represented by the number of separation required to split the identifiers). Table 1 shows the output of this phase.

As shown in Table 1, the class label is a numeric where it reflects the number of operations required to split an identifier. For example, the first row which yields 'CreateProcess' identifier requires a one operation to split 'Create' from 'Process'.

Character Encoding

In this phase, both the identifier and class label attributes will be encoded in order to be processed by the BPNN. Such encoding aims to convert the characters in the identifier attribute into a binary representation. For this purpose, this study has brought the mechanism of encoding the letters from different studies (Mezher and Omar, 2015; Al-Serhan, 2008). This has been performed by converting the characters into their equivalent ASCII code. Table 2 shows the equivalent binary values for each letters.

As shown in Table 2, every letter will be replaced with its equivalent encoding. In terms of dealing with the numbers, all the digits will be replaced with a unified encoding. Similarly, all the punctuation will be replaced with a unified encoding, this will include symbols such as '=' and '+'.

Now, in order to encode the class label attribute, it is necessary to identify all the possibilities of class labels. From the observation of the dataset, it turns out that the classes are ranging from 0 to 13 in addition to another class label which is 16. In this manner, each class will be encoded using binary representation. However, unlike the letters encoding, class encoding will takes a form of one-hot-encoding where a 15 bit is being used (i.e., number of classes). Every bit will be encoded as 0 with an exception for the bit that reflects the occurrence of the selected class label which will be encoded as 1. Table 3 shows the encoding mechanism for each class label.

After explaining the mechanism of encoding both characters and class label, let us consider the last row in Table 1 from the normalized dataset. Figure 3 shows the mechanism of processing the identifier with its class label of such row via the network of backpropagation.

Adjusting the Network

In order to adjust the neural network, it is necessary to identify multiple parameters including input neurons, hidden layer, hidden neurons, output neurons, training mechanism and the objective function. First, the input neurons have been identified using the unified length mechanism that has been used by Hashim and Omar (2016). Such mechanism aims to find the longest token within the identifier attribute. Then, using the length of such token to unify the input size with the remaining bits. Note that, the remaining bits will be encoded with zero values. After observing the dataset, it turns out that the longest token had a length of 72 characters. This will lead to 432 bit due to each character has a corresponding 6 bit encoding. In this manner, the input neurons size will be 432. Consider an example of '010110' as an input, if we want to unify its length to be matched with the longest one, it is necessary to add extra zeros bits as shown in Fig. 4.

On the other hand, the output size has been determined based on the length of the class labels which was 15. However, for the hidden layer, a single hidden layer has been used in this study. Regarding to the number of hidden neurons, this study has been using the mechanism of Haykin and Network (2004) for identifying an initial number of hidden neurons. Such mechanism can be emphasized using Equation 1:

$$hidden\ neurons = \sqrt{input\ size \times output\ size} \quad (1)$$

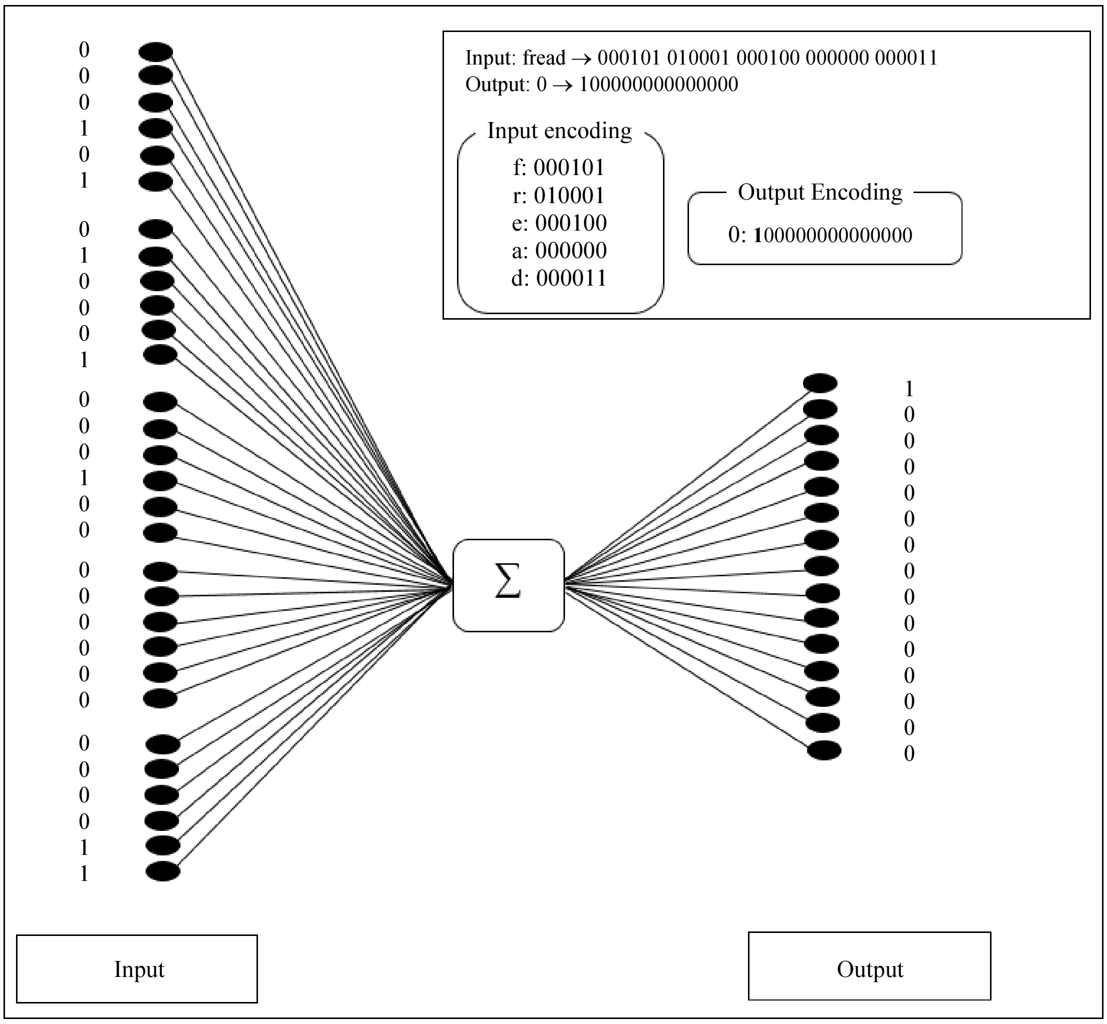

Fig. 3: Example of representing the identifier 'fread' over the network

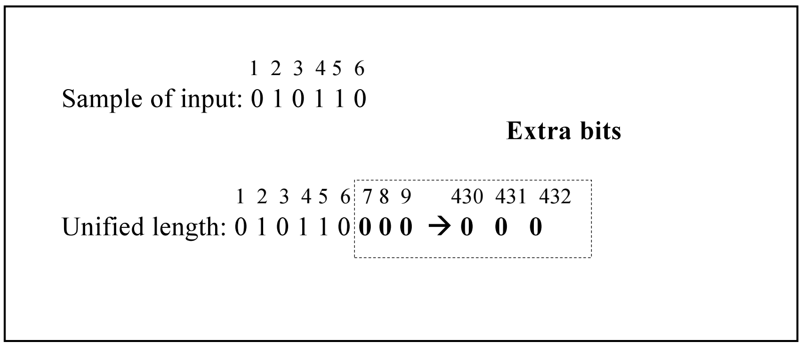
1 2 3 4 5 6
Sample of input: 0 1 0 1 1 0
Extra bits
1 2 3 4 5 6 7 8 9 430 431 432
Unified length: 0 1 0 1 1 0 0 0 0 → 0 0 0

Fig. 4: Unified input length

Table 2: Letters encoding using binary representation

Letter	Decimal	Binary	Letter	Decimal	Binary
a	0	000000	A	27	011011
b	1	000001	B	28	011100
c	2	000010	C	29	011101
d	3	000011	D	30	011110
e	4	000100	E	31	011111
f	5	000101	F	32	100000
g	6	000110	G	33	100001
h	7	000111	H	34	100010
i	8	001000	I	35	100011
j	9	001001	J	36	100100
k	10	001010	K	37	100101
l	11	001011	L	38	100110
m	12	001100	M	39	100111
n	13	001101	N	40	101000
o	14	001110	O	41	101001
p	15	001111	P	42	101010
q	16	010000	Q	43	101011
r	17	010001	R	44	101100
s	18	010010	S	45	101101
t	19	010011	T	46	101110
u	20	010100	U	47	101111
v	21	010101	V	48	110000
w	22	010110	W	49	110001
x	23	010111	X	50	110010
y	24	011000	Y	51	110011
z	25	011001	Z	52	110100
Digit	26	011010	Punctuation	53	110101

Table 3: Classes encoding

Classes	Encoding
0	**1**00000000000000
1	0**1**0000000000000
2	00**1**000000000000
3	000**1**00000000000
4	0000**1**0000000000
5	00000**1**000000000
6	000000**1**00000000
7	0000000**1**0000000
8	00000000**1**000000
9	000000000**1**00000
10	0000000000**1**0000
11	00000000000**1**000
12	000000000000**1**00
13	0000000000000**1**0
16	00000000000000**1**

In addition, the training mechanism has been adjusted based on 80% for training and 20% for testing. Note that, the network has been trained with a learning rate $\eta = 0.2$. Training was performed until an acceptable convergence was found for 515 epochs with mean squared error less than 0.01.

Finally, regarding to the objective function, three functions have been used including Tanh, Sigmoid and Softmax.

BPNN

Backpropagation is one of the famous architectures of Artificial Neural Network (ANN) (Yin *et al*., 2011). It was used vastly in the literature of text categorization. BPNN composed of input layer, single hidden layer and output layer. In order to make BPNN work properly, the data should be encoded like in the previous sections. After encoding the data, an initialization process will be conducted in order to generate the weights' values between the input and the hidden layers. A multiplication process will be applied where the inputs' values will be multiplied by the initiated weights. Similarly, the weights' values between the hidden layer and the output layer will be generated randomly. As well as, the same multiplication process will be held by multiplying the hidden values by the weights. Now, the acquired value will undergo a normalization task using the objective function which aims to limit the value between 0 and 1. Consequentially, a comparison will be conducted between the achieved value, which represents the predicted output, with the actual output. Such comparison aims to compute the error rate. If the error rate is acceptable then, the weights will be generalized onto all the data instances.

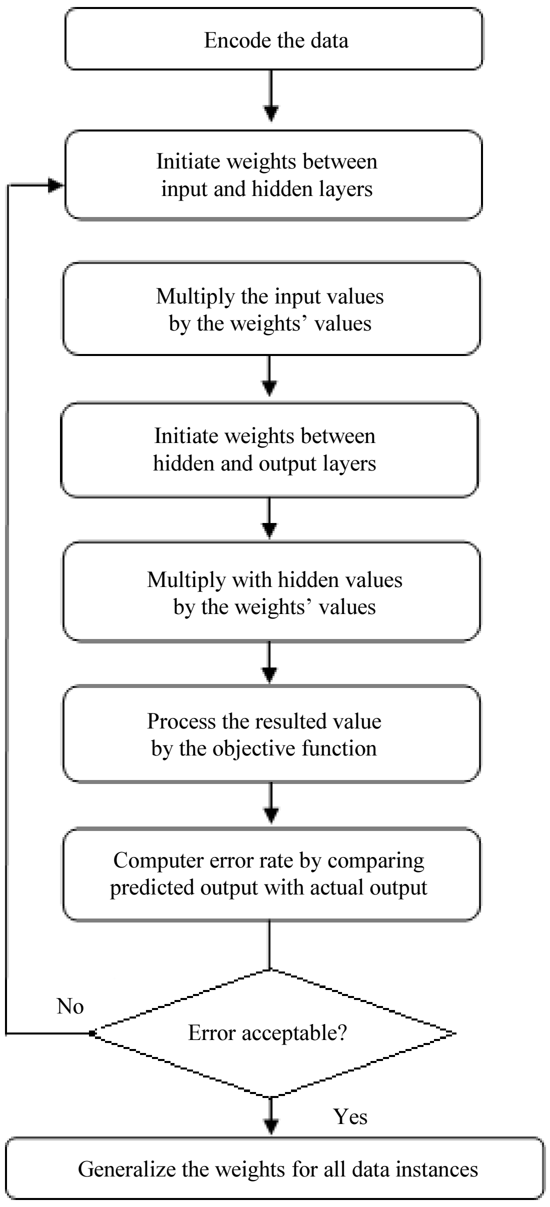
Encode the data
Initiate weights between input and hidden layers
Multiply the input values by the weights' values
Initiate weights between hidden and output layers
Multiply with hidden values by the weights' values
Process the resulted value by the objective function
Computer error rate by comparing predicted output with actual output
Error acceptable?
No
Yes
Generalize the weights for all data instances

Fig. 5: Workflow of BPNN

Otherwise, BPNN will return back and update the weights with new values and the procedures will be iteratively accommodated until reaching an acceptable error rateure Fig. 5 shows the workflow of BPNN.

Results

Prior to show the results of the proposed BPNN, it is necessary to highlight the evaluation method used in this study. In fact, the common evaluation metrics that have been used for machine learning techniques are being used in this study including precision, recall and f-measure. First, precision aims to calculate the number of correctly classified identifiers (i.e., True Positives TP) in accordance to the incorrectly classified identifiers (i.e., False Positives FP). Equation 2 shows how precision is computed:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

On the other hand, recall aims to calculate the number of correctly classified identifiers (i.e., True Positives TP) in accordance to the number of instances that have not been classified (i.e., False Negatives FN). Equation 3 shows how recall is being computed:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

Finally, f-measure is the final accuracy that is being computed by finding the harmonic between precision and recall. Equation 4 shows how f-measure is being calculated:

$$F - measure = \frac{2\,\mathrm{Pr} \times \mathrm{Re}}{\mathrm{Pr} + \mathrm{Re}} \tag{4}$$

Based on the evaluation metrics, the results of BPNN will be assess based on different objective functions including Tanh, Sigmoid and Softmax. Table 4 and Fig. 6 show such results.

As shown in both Table 4 and Fig. 6, the highest values of precision, recall and f-measure have been obtained by the Softmax function by achieving 73.1%, 69.8% and 71.4% respectively. This has been followed by the Tanh where the precision was 72.6%, recall was 65.3% and f-measure was 68.7%. The lowest values have been achieved by the Sigmoid function by obtaining 70.1%, 64.9% and 67.3% for precision, recall and f-measure respectively.

In order to declare the novelty of this study, it is necessary to mention the accuracy of the state of the art. Lawrie *et al.* (2007) have obtained an f-measure of 62% using regular expression method. While Enslen *et al.* (2009) have obtained 60% of f-measure using statistical method based on term frequency. Moreover, Lawrie and Binkley (2011) have achieved 61% of f-measure using a semantic method based on a dictionary for abbreviations. Finally, Alanee and Murad (2017) have achieved an f-measure of 64.7% using a probabilistic method of Naïve Bayes. Comparing these results with all BPNN functions used in this study, it is clear that BPNN has outperformed the state of the art in terms of splitting the multi-word identifiers. This emphasized the usefulness of BPNN when handling character-based problems compared to other methods such as probabilistic, semantic or statistical.

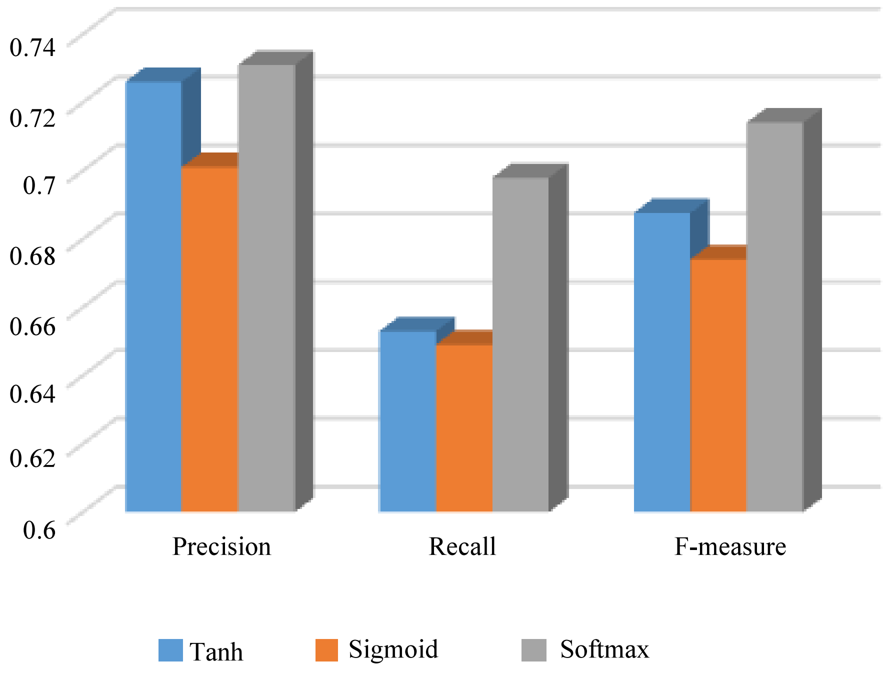

Fig. 6: Results of precision, recall and f-measure for each function

Table 4: Results of BPNN with different functions

Objective function	Precision	Recall	F-measure
Tanh	0.726	0.653	0.687
Sigmoid	0.701	0.649	0.673
Softmax	0.731	0.698	0.714

Conclusion

This paper has proposed a backpropagation neural network for the task of splitting identifiers. A benchmark of source code dataset has been used in the experiments. In addition, different objective functions have been used in the process of carrying out BPNN including Tanh, Sigmoid and Softmax. Softmax has achieved the highest f-measure value by obtaining 71.4%. Comparing such results with the related work which utilized probabilistic, semantic and statistical methods has revealed a superior performance of BPNN. Future work may address the use of word embedding in accordance with BPNN or other ANN architectures.

Acknowledgement

This study has been funded by the University of Information Technology and Communications.

Author's Contributions

Ahmed Sabah Ahmed Al-Jumaili: Identifying the domain and the exact gap. In addition, proposing a method to solve such gap.

Huda Kadhim Tayyeh: Implementing the proposed method by identifying the dataset and applying the method.

Ruqaia Jawad Kadhem: Writing the manuscript and analyzing the results.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Alanee, N. and M.A.A. Murad, 2017. A hybrid method of feature extraction and Naive Bayes classification for splitting identifiers. J. Theoretical Applied Inform. Technol., 95: 1549-1557.

Al-Serhan, H.M., 2008. Extraction of Arabic word roots: An approach based on computational model and multi-backpropagation neural networks. PhD Thesis, De Montfort University-UK.

Alshaikhdeeb, B. and K. Ahmad, 2015. Integrating correlation clustering and agglomerative hierarchical clustering for holistic schema matching. J. Comput. Sci., 11: 484-489. DOI: 10.3844/jcssp.2015.484.489

Alshaikhdeeb, B. and K. Ahmad, 2016. Biomedical named entity recognition: A review. Int. J. Adv. Sci. Eng. Inform. Technol., 6: 889-895. DOI: 10.18517/ijaseit.6.6.1367

Alshaikhdeeb, B. and K. Ahmad, 2017. Feature selection for chemical compound extraction using wrapper approach with Naive Bayes classifier. Proceedings of the 6th International Conference on Electrical Engineering and Informatics, Nov. 25-27, IEEE Xplore Press, Langkawi, Malaysia, pp: 1-6. DOI: 10.1109/ICEEI.2017.8312421

Enslen, E., E. Hill, L. Pollock and K. Vijay-Shanker, 2009. Mining source code to automatically split identifiers for software analysis. Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories, May 16-17, IEEE Xplore Press, Vancouver, BC, Canada, pp: 71-80. DOI: 10.1109/MSR.2009.5069482

Feild, H., D. Binkley and D. Lawrie, 2006. An empirical comparison of techniques for extracting concept abbreviations from identifiers. Proceedings of the IASTED International Conference on Software Engineering and Applications, (SEA' 06), Citeseer.

Hashim, B. and N. Omar, 2016. A back propagation neural network for identifying multi-word biomedical named entities. Int. Rev. Comput. Software, 11: 682-690.

Haykin, S. and N. Network, 2004. A comprehensive foundation. Neural Netw.

Lawrie, D. and D. Binkley, 2011. Expanding identifiers to normalize source code vocabulary. Proceedings of the 27th IEEE International Conference on Software Maintenance, Sept. 25-30, IEEE Xplore Press, Williamsburg, VI, USA, pp: 113-122. DOI: 10.1109/ICSM.2011.6080778

Lawrie, D., H. Field and D. Binkley, 2007. Quantifying identifier quality: an analysis of trends. Empirical Software Eng., 12: 359-388. DOI: 10.1007/s10664-006-9032-2

Maletic, J.I. and A. Marcus, 2001. Supporting program comprehension using semantic and structural information. Proceedings of the 23rd International Conference on Software Engineering, May 19-19, IEEE Xplore Press, Toronto, Ontario, Canada, pp: 103-112. DOI: 10.1109/ICSE.2001.919085

Mezher, K. and N. Omar, 2015. A backpropagation neural network to improve Arabic stemming. J. Theoretical Applied Inform. Technol., 82: 385-394.

Poshyvanyk, D., A. Marcus, Y. Dong and A. Sergeyev, 2005. IRiSS-a source code exploration tool. Proceedings of the 21st IEEE International Conference on Software Maintenance - Industrial and Tool, Sept. 25-30, Budapest, Hungary, pp. 69-72.

Sjøberg, D.I.K., J.E. Hannay, O. Hansen, V.B. Kampenes and A. Karahasanovic *et al.*, 2005. A survey of controlled experiments in software engineering. IEEE Trans. Software Eng., 31: 733-753. DOI: 10.1109/TSE.2005.97

Thomas, S.W., 2011. Mining software repositories using topic models. Proceedings of the 33rd International Conference on Software Engineering, May 21-28, ACM, Waikiki, Honolulu, HI, USA pp: 1138-1139. DOI: 10.1145/1985793.1986020

Yin, F., H. Mao, L. Hua, W. Guo and M. Shu, 2011. Back Propagation neural network modeling for warpage prediction and optimization of plastic products during injection molding. Mater. Design, 32: 1844-1850. DOI: 10.1016/j.matdes.2010.12.022