1

A Review Study on Bibliography Analysis Using Deep Learning and SE

Rahul Krishna[1], Rajalakshmi Biju[2], Aron Jose[3]

[1]Santhigiri College of Computer Sciences,
Vazhithala, Thodupuzha
bcab19_2223@santhigiricollege.com

[2]Santhigiri College of Computer Sciences,
Vazhithala, Thodupuzha
bcab19_2224@santhigiricollege.com

[3]Santhigiri College of Computer Sciences,
Vazhithala, Thodupuzha
mca2022_aronjose@@santhigiricollege.com

***Abstract*- Deep learning is a technique that allows computational models composed of multiple processing layers to learn representations of data with multiple levels of abstraction. Deep learning recently plays an important role for solving SE tasks. Software engineering researchers solve problems of several different kinds. To do, we produce several different kinds of results, and they should develop appropriate evidence to validate these results. Deep learning is increasingly prevalent in the field of Software Engineering. The practicability becomes a concern in utilizing deep learning techniques and how to improve the effectiveness, efficiency, understandability, and testability of deep learning based solutions may attract more SE researchers in the future.**

***Keywords*-** Deep learning, AutoEncoder, Bug report summarization, Effectiveness and efficiency, Metamorphic testing.

I. INTRODUCTION

By the success of deep learning in data mining and pattern recognition, recent years have been witnessed an increasing trend for the industrial practitioners and academic researchers to integrate deep learning into Software Engineering tasks. For typical SE tasks, deep learning helps SE participators to extract the requirements from natural language text, generate source code, predict defects in software, etc. As an initial statistics of research papers in SE in this study, deep learning has achieved competitive performance against previous algorithms on about 40 SE tasks.

Despite the encouraging amount of papers and venues, there exists a little overview analysis on deep learning in SE. For eg, the common way to integrate deep learning into SE, the SE phases facilitated by deep learning, the interests of SE practitioners on deep learning, etc. Understanding these questions is important. On the one hand, it helps the practitioners and researchers get an overview understanding of deep learning in SE. On the next hand, practitioners and researchers can develop more practical deep learning models according to the analysis. For this purpose, this study conducts a bibliography analysis on research papers in the field of Software Engineering that use deep learning techniques. In contrast , bibliography analysis can reflect the overview trends, techniques, topics on deep learning in SE by statistical data

We find that the deep learning has increased significantly in SE in recent years, Both communities of SE and Artificial Intelligent (AI) show great interests in utilizing deep learning in SE. Surprisingly, more than one fifth of research papers have industrial practitioners to participate in, which means that industrial practitioners are also interested in integrating deep learning into their SE solutions. Despite the encouraging success of deep learning, there are several concerns about using deep learning in SE. Practitioners and researchers worry about the practicability of utilizing a complex method with almost opaque internal representations like deep learning. Hence, the effectiveness and efficiency, understandability, and testability has become the burden to use deep learning in practice. Fortunately, recent studies have conducted some initial investigation on these problems. These findings may lead the future studies of using deep learning in SE.

I. EXAMPLE OF USING DEEP LEARNING IN SE

Deep learning is a technique that allows the computational models composed of multiple processing layers to learn representations of data with multiple levels of abstraction. In this section, we are presenting an example of using deep learning in SE. In this example, we apply the deep learning model Auto Encoder on a typical SE task. That is, bug reports summarization.

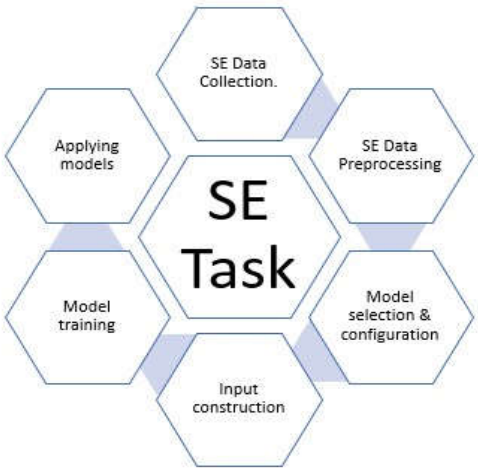


Fig.1 A framework to summarize bug reports

Bug reports are texts to describe the bugs in software. Facing numerous bug reports, bug report summarization aims to generate a summary by extracting and highlighting informative sentences of a bug report to shorten the reading time. To identify informative sentences, the researchers utilize Auto Encoder to encode the words in bug report sentences in an unsupervised way. Since the hidden state of Auto Encoder provides a compressed representation of the input data, the weights of words in a bug report can be measured by calculating how much information of a word is reserved in the hidden states. Based on the word weights, informative sentences are identified. As shown in Fig. 1, the example consists of six important steps.

1. **SE data collection** It decides the available data for an SE task. For a bug report summarization, the commonly available data are bug reports. Each bug report mainly contains a title, a description of the bug, and several comments.
2. **SE data preprocessing** In this process, we removes the noises in SE data. For a bug report, the English stop words and some programming-specific ones are the noises. Besides, extremely short sentences are also noises, since they may be uninformative.
3. **Model selection** and configuration select suitable deep learning models for SE data and decide model configurations, e.g., the number of layers and neural units of each layer. The widely used deep learning models include AutoEncoder, CNN, RNN, etc. (explained in Fig. 3). These standard models usually have several variants, e.g., LSTM, Bi-LSTM, and attention-based RNN are classical variants of RNN. In this example, AutoEncoder is selected. AutoEncoder usually has a symmetric architecture, i.e., the number of neural units of input and output layers are the same. The output layer is defined as a pattern to reconstruct the input layer. The number of neural units of hidden layers decreases as towards the middle of the network. After training, the hidden states reserve the key information for reconstructing the input layer.
4. **Input construction** transforms SE data into vectors for deep learning models. For bug report summarization, researchers calculate the word frequency in bug reports and transform the word frequency values into vectors. These vectors are regarded as a training set for Auto Encoder.
5. **Model training** trains the designed model with the training set. A deep learning model usually has thousands of parameters representing the weights of connections among neural units. Hence, training the model is to tune these parameters according to the training set. For Auto Encoder, the parameters are trained by minimizing the difference between the input and output layers in an unsupervised way.
6. **Applying models** is to utilize the trained model to solve SE problems. In this example, the trained model can encode the word frequency vector of a new bug report into the hidden states. We can trace and calculate the changes of the value in each vector dimension along with the encoding process, and then deduce the weights of words in each dimension. These word weights help researchers assign weights of the sentences and select informative ones.

II. DATA COLLECTION

To collect deep learning related papers in SE, we design three criteria to search research papers from four well-known digital libraries, including Web of Science, ACM Digital Library, IEEE Xplore, and Scopus.

1. Research papers should contain at least one of the following SE phrases, including "software engineer*", "software develop*", "software test*", "software design", "requir* analysis", "software require*", "software maintain*", and "software manage*". The sign "*" is a wildcard character to match zero or more characters in a word.

2. Research papers should contain at least one phrase about deep learning concept,that is, "deep learn*" and "neural network*".
3. Research papers are conference or journal papers written in English on the topic of computer science.

Under these criteria, 4,443 candidate research papers published before March 2018, including 414 from Web of Science, 207 from ACM Digital Library, 2,271 from IEEE Xplore, and 1,551 from Scopus. We remove the duplicate papers and short papers with less than 4 pages. At last, 3,351 research papers are reserved. We download and manually examine the contents of the papers:

1. We remove 35 papers that the full-contents cannot be downloaded.

2. We remove 2,441 papers that the searching phrases in C1 and C2 merely match some supplementary information in the paper. For example, "software engineer*" may match the phrase of "school of software engineering" in author biography or the publication venue "Transaction on Software Engineering".

3. After step 1 and 2, another 812 papers are removed as they do not focus on SE or deep learning. For example, "deep learning" is also a concept in computer education and "neural network" may refer to a shallow network structure with a single hidden layer.

At the end, 63 research papers are remained. We take these papers as seeds to search their references and citations. If a new SE research paper about deep learning is found, we recursively examine the new paper. Finally, another 35 research papers are found. Hence, we collect in total 98 published or accepted research papers for analysis.

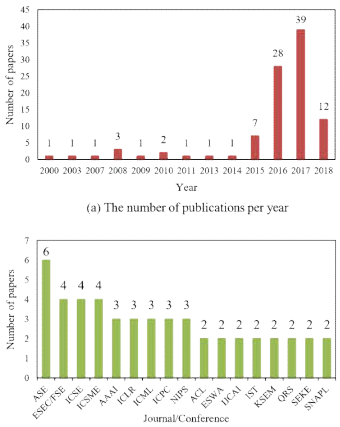
III. BIBLIOGRAPHY ANALYSIS

A. The prevalence of deep learning in SE

Counting the number of research papers each year and the venues of the publications in Fig.2(a) and Fig. 2(b) respectively. In Fig. 2(a), we find that deep learning attracts little attention in SE for a long time, that is, only less than 3 papers are published each year before 2015. The reason may be that although deep learning performs well on image processing, speech recognition, etc., it takes time for the practitioners and researchers in SE to validate deep learning on domain-specific SE tasks.

For these research papers, we count the publication venues. Fig. 2(b) presents the publication venues that publish more than one paper. We explain these venue names in Fig. 2(c). We find that using deep learning in SE attracts the attention from both communities of SE and AI, including some premier SE venues like ICSE, ESEC/FSE, ASE, ICSME, ICPC and some renowned AI venues like AAAI, ICLR, ICML, NIPS, ACL, IJCAI. These venues may be a good guidance to study the progress of deep learning in SE.

To conclude, deep learning is prevalent in SE. It attracts the attention from both SE and AI communities. Deep learning recently plays an important role for solving SE tasks. Software engineering researchers solve problems of several different kinds they should develop appropriate evidence to validate these results.



(a) The number of publications per year

(b) Top venues of the publications

(c) Full names of publication venues

Venue	Explanation
ICSE	Int'l Conf. on Softw. Eng.
ESEC/FSE	Joint European Softw. Eng. Conf. and Symposium o the Foundations of Softw. Eng.
ASE	Int'l Conf. on Automated Softw. Eng.
ICSME	Int'l Conf. on Softw. Maintenance and Evolution
AAAI	AAAI Conf. on Artificial Intelligence
ICLR	Int'l Conf. on Representation Learning
ICML	Int'l Conf. on Machine Learning
ICPC	Int'l Conf. on Program Comprehension
NIPS	Conf. on Neural Information Processing Systems
ACL	Annual Meeting of the Association for Computation Linguistics
ESWA	Expert Systems With Applications
IJCAI	Int'l Joint Conf. on Artificial Intelligence
IST	Information and Softw. Tech.
KSEM	Int'l Conf. on Knowledge Science, Eng. and Management
QRS	Int'l Conf. on Quality Softw
SEKE	Int'l Conf. on Softw. Eng. and Knowledge Eng.
SNAPL	Summit on Advances in Programming Languages

Fig.2 Basic information of deep learning in SE

B. The way to integrate deep learning into SE

As the prevalence of deep learning in Software Engineering, we analyse the way to integrate deep learning into SE. Fig. 3 shows the name of deep learning models and the number of papers using these models. Most studies directly transfer standard deep learning models into SE, including Auto Encoder, CNN, DBN, RNN and a simple fully-connected DNN. Meanwhile, the classical variants of these models in AI are also widely used such as SDAEs, LSTM, etc. The above models are used in 84.7% research papers. Besides using a single model, combined deep learning models also show competitiveness in SE, for eg a combination of RNN and CNN. For the remaining papers, researchers design specific deep learning architectures for SE data like Stepped AutoEncoder and TBCNN. The above phenomenon suggests that when integrating deep learning into SE tasks, a new practitioner may be willing to first try some standard models and their variants to investigate whether deep learning works or not.

Furthermore, we investigate what types of SE data are usually fed into these models. The inputs can be categorized into five important categories.

1. Predefined software metrics - Researchers first manually define and calculate some software metrics, eg, lines of code, the number of bugs in source code. Then, they construct vectors based on the values of these metrics to feed into deep learning models.
2. Dynamic software status - This category takes the dynamic information when running the software as input, e.g., the CPU utilization, the invoked APIs. The values of these dynamic information can be transformed into vectors for deep learning models.
3. Raw text or source code without sequence - These papers treat the bag-of-words of text and source code as deep learning input without considering word sequences. Based on the bag-of-words, word embedding, one-hot representation and word frequency are widely used to transform words into vector space for deep learning.
4. Raw text or source code in sequence - In contrast to category 3, this category considers as the sequence of words. Such inputs are usually associated with RNN-based models, which utilizes the order of words to predict the next word or class label of software documents and source code, eg learning and program synthesis.
5. Others, int this category most of the other inputs are multimedia data such as images. For example, researchers utilize images to test deep learning models. The pixels of the images are fed into deep learning models.

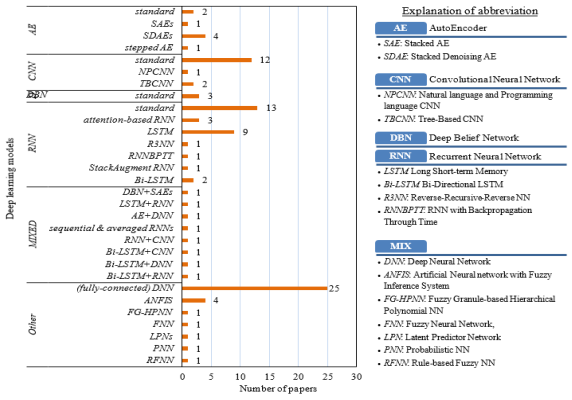


Fig.3 Deep learning models in the research papers

To conclude, practitioners and researchers can achieve competitive results on more than 80% SE problems when only using standard deep learning models and their variants. Deep learning can well handle many types of SE data, including predefined software metrics, dynamic software status, and raw text or source code.

C. The SE phases facilitated by deep learning

Due to the diversity of SE tasks, it is important to identify the existing SE tasks facilitated by deep learning, since it helps practitioners find the potential to leverage deep learning in their own problems.

As suggested by classical SE models, that is Waterfall Model and Incremental Model, SE can be divided into five phases, including requirement analysis, software design, development, testing and maintenance. In addition, since SE is an activity involving many stakeholders (developers, testers, project managers, etc.), we also add project management as an SE phase. Fig. 4 shows the SE tasks facilitated by deep learning in the six phases.

In requirement analysis, deep learning helps requirement analysts automatically extract requirements from natural language texts. In software design, design patterns of software can be recognized. In software development, deep learning helps developers on 14 SE tasks from 30 research papers, including program learning and program synthesis, code suggestion, etc. Besides, software testing and maintenance are also major phases to attempt deep learning. There are 54 research papers in these two phases which cover 21 SE tasks like defect prediction and reliability or changeability estimation. For the 41 SE tasks, program learning and program synthesis, malware detection, defect prediction, reliability or changeability estimation, and development cost or effort estimation are the top 5 tasks studied by the researchers. Hence, practitioners may have a board selection of methodologies and deep learning models when using deep learning on these tasks.

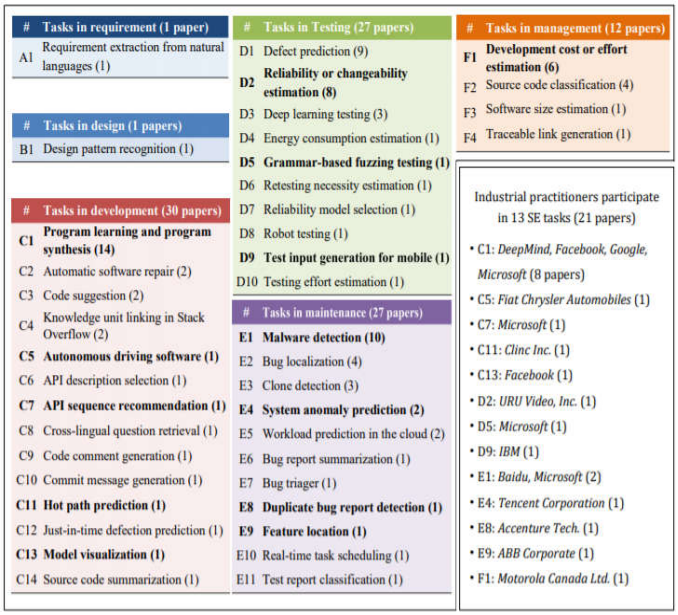


Fig.4 The SE tasks solved by deep learning and participated by industrial practitioners.

To conclude, deep learning has facilitated at least 41 SE tasks in all Software Engineering phases including requirement

analysis, software design, development, testing, maintenance, and project management.

D. Research interests of industrial practitioners

We analyse the SE tasks participated by industrial practitioners to understand research interests in practice.

The industrial practitioners are identified when at least one author affiliation in the author list of a research paper is a company. In Fig.4, we label the industry participated SE tasks in bold and list the company names. There are 21 research papers (more than 1/5) on 13 SE tasks with at least one industrial practitioner, which implies the interest of industrial practitioners in integrating deep learning into SE problems. Among the 13 tasks, program learning and program synthesis attract the most attention. Eight research papers from four companies have tried deep learning on this task, including DeepMind, Facebook, Google, and Microsoft. Besides, practitioners also apply deep learning on SE tasks like malware detection, development cost or effort estimation etc., and achieve competitive results. Hence, deep learning is useful on these tasks from the perspective of practitioners. This finding provides a guidance for academic researchers to apply deep learning in practice.

However, we finded out a mismatch from the top researched SE tasks and the industrial interests. For the top 5 tasks studied by deep learning in Fig. 4, only program learning and program synthesis, and malware detection attract more than one industrial practitioner to participate in. The reason may be that, on the one hand, practitioners have not found a suitable way to apply deep learning on other SE tasks in practice. On the other hand, practitioners already select some lightweight methods to solve these tasks. Hence, there is still a long way to apply a complex method like deep learning in industry.

To conclude, practitioners participate in more than one fifth research papers. They benefit from deep learning on 13 SE tasks, including program learning and program synthesis, malware detection and lot more.

E. Concerns to use deep learning in SE

Despite the prevalence of improving SE tasks with deep learning, many concerns emerges on the practicability of using deep learning in SE. As this is a complex and almost opaque model, several factors limit the practicability of deep learning, it including the effectiveness, efficiency, understandability , and testability. But these kinds of issues may influence the development of deep learning in SE.

Effectiveness and Efficiency - Recent studies show that by applying a simple optimizer Differential Evolution to fine tune SVM, it achieves similar results with deep learning on linking the knowledge unit in Stack Overflow. Most importantly, this method is 84 times faster than training deep learning models. The same phenomenon is also observed on code suggestion, in which an adapting n-gram language model specifically designed for software surpasses RNN and LSTM. Although techniques like off-line training and cloud computing may partially resolve the efficiency problem, there is still a trade-off between deep learning and other lightweight, domain-specific models. Such trade-off drives a deep investigation on deep learning. For eg, what types of SE data and tasks are suitable for deep learning and how to integrate the domain knowledge into deep learning.

Understandability - The understandability is a burden to "control" deep learning. Recently, several methods are proposed to improve the understandability of deep learning. For example, practitioners in Facebook explore to visualize industry-scale deep neural networks. The proposed tool help software engineers understand the neuron activations, individual instances, classification results, and differences between activation patterns of deep learning. Such tool is a good start to increase the understandability of deep learning in SE.

Testability – As this is considered as a complex model, the testability limits the security of applying deep learning in SE. Hence, researchers attempt to use software testing techniques to improve the testability of deep learning, that is deep learning testing. To test deep learning models, coverage testing and metamorphic testing are recently applied. Coverage testing validates whether all the neural units in deep learning are correctly activated or not. Metamorphic testing generates the test oracle for coverage testing. These studies demonstrate the importance of SE techniques on validating artificial intelligence techniques like deep learning. To conclude this session, the practicability of deep learning is still a rising and hot topic for SE practitioners and researchers.

IV. CONCLUSION

Deep learning recently plays an important role for solving SE tasks. In this study, we conduct a bibliography analysis on the status of deep learning in SE. We find that deep learning has been integrated into more than 40 SE tasks by both industrial practitioners and academic researchers. Most studies use standard deep learning models and their variants to solve SE problems. The practicability of deep learning may hider Software Engineering practitioners from using deep learning in practice, which is a rising and hot topic for further investigation and future use.

V. REFERENCES

[1] Madala, K., Gaither, D., Nielsen, R., & Do, H. Automated identification of component state transition

model elements from requirements. International Requirements Engineering Conference Workshops. 2017. (pp.386-392).

[2] Joulin, A., & Mikolov, T. Inferring algorithmic patterns with stack-augmented recurrent nets. NIPS'15. (pp. 190-198).

[3] Tong, H., Liu, B., & Wang, S. Software defect prediction

[4] using stacked denoising autoencoders and two-stage ensemble learning. IST'17.

[5] Pang, Y., Xue, X., & Wang, H. Predicting vulnerable software components through deep neural network. International Conference on Deep Learning Technologies. 2017. (pp.6-10).

[6] Dahl, G. E., Stokes, J. W., Deng, L., & Yu, D. Large-scale malware classification using random projections and neural networks. International Conference on Acoustics, Speech and Signal Processing. 2013. (pp. 3422-3426).

[7] Fu, W., & Menzies, T. Easy over hard: a case study on deep learning. FSE'17. (pp.49-60).

[8] Kahng, M., Andrews, P.Y., Kalro, A., & Chau, D.H.P. ActiVis: visual exploration of industryscale deep neural network models. IEEE Transactions on Visualization and Computer Graphics, 24(1), 2018. (pp.88-97).

[9] Tian, Y., Pei, K., Jana, S., Ray, B. DeepTest: automated testing of deep-neural-network-driven autonomous cars. ICSE'18.

[10] Li, X., Jiang, H., Liu, D., Ren, Z., & Li, G. Unsupervised deep bug report summarization. ICPC'18.

[11] Munassar, N.M.A., & Govardhan, A. A comparison between five models of software engineering. International Journal of Computer Science Issues, 5, 2010.
(pp.95-101).

[12] Dwivedi, A. K., Tirkey, A., Ray, R. B., & Rath, S. K. Software design pattern recognition using machine learning techniques. Region 10 Conference. 2016. (pp.222-227).

[13] Huang, X., Ho, D., Ren, J., & Capretz, L.F. Improving the COCOMO model using a neuro-fuzzy approach. Applied Soft Computing, 7(1), 2007. (pp.29-40).

[14] Abd-El-Hafiz, S. 2000. Identifying objects in procedural programs using clustering neural networks, Automated Software Engineering 7(3): 239–261.

[15] Bergadano, F. and Gunetti, D. 1996. Testing by means of inductive program learning, ACM Trans. Software Engineering and Methodology 5(2): 119–145.

Author Profile

Aron Jose
Pursing the Master of Computer Application from Santhigiri College of Computer Sciences, Vazhithala in 2020 – 2022.

Rajalakshmi Biju
Pursing the Bachelor of Computer Application from Santhigiri College of Computer Sciences, Vazhithala in 2019 – 2022.

Rahul Krishna
Pursing the Bachelor of Computer Application from Santhigiri College of Computer Sciences, Vazhithala in 2019 – 2022.