A Testing Tool for Machine Learning Applications

Yelin Liu
School of Computing and IT
University of Wollongong
Wollongong, NSW 2522, Australia
yl908@uowmail.edu.au

Yang Liu
Booking.com B.V.
Herengracht 597, 1017 CE Amsterdam
Netherlands
flint.liu@booking.com

Tsong Yueh Chen
Department of Computer Science & Software Engineering
Swinburne University of Technology
Hawthorn, VIC 3122, Australia
tychen@swin.edu.au

Zhi Quan Zhou*
School of Computing and IT
University of Wollongong
Wollongong, NSW 2522, Australia
zhiquan@uow.edu.au

ABSTRACT

This talk proposal presents the design of MTKeras, a generic metamorphic testing framework for machine learning applications, and demonstrates its effectiveness through two case studies in different domains, namely, image classification and sentiment analysis.

KEYWORDS

Metamorphic testing, metamorphic relation pattern, MR composition, oracle problem, neural network API, Keras, MTKeras

ACM Reference Format:
Yelin Liu, Yang Liu, Tsong Yueh Chen, and Zhi Quan Zhou. 2020. A Testing Tool for Machine Learning Applications. In *MET '20: IEEE/ACM International Workshop on Metamorphic Testing, 2020.* ACM.

1 INTRODUCTION

Researchers have applied metamorphic testing (MT) to test machine learning (ML) systems in specific domains such as computer vision, machine translation, and autonomous systems [8, 9]. Nevertheless, the current practice of applying MT to ML is still at an early stage. In particular, the identification of *metamorphic relations* (MRs) is still largely a manual process, not to mention the implementation (coding) of MRs into test drivers. MRs are the most important component of MT, referring to the expected relations among the inputs and outputs of multiple executions of the target program [3]. It has been observed that MRs identified for different application domains often share similar viewpoints, hence the introduction of the concept of *metamorphic relation patterns* (MRPs) [5, 9]. For example, "equivalence under geometric transformation" is an MRP that can be used to derive a concrete MR for the *time series analysis* domain and another concrete MR for the *autonomous driving* domain [9].

*All correspondence should be addressed to Dr. Z. Q. Zhou.

In this research, therefore, we ask the following research questions. **RQ1**: Can we develop a generic, domain-independent automated metamorphic testing framework to allow developers and testers of ML systems to define their own MRs? Here, "define" means "identify and implement." **RQ2**: What is the applicability and effectiveness of our solution?

To address RQ1, we have developed the first version of an automated metamorphic testing framework named *MTKeras*, which allows the users to define their own MRs based on a prescribed collection of operators. We have also conducted preliminary case studies to investigate RQ2.

2 MTKERAS: AN MT FRAMEWORK FOR ML

ML platforms and libraries, such as *TensorFlow* and *Theano*, are now widely available to allow users to develop and train their own ML models. We have built our MT framework, *MTKeras*, on the *Keras* platform. Keras (https://keras.io) is a popular *high-level* neural networks API, developed in Python and working on top of low-level libraries—those backend engines such as Tensorflow and Theano can be plugged seamlessly into Keras.

The Keras API empowers users to configure and train a neural network model based on datasets for various tasks such as image classification or sentiment analysis. MTKeras enables automated metamorphic testing by providing the users with an **MR library** for testing their ML models and applications. We have designed the MR library based on the concept of a hierarchical structure (levels of abstractions) of MRPs [9]. MTKeras also allows the users to define and run new MRs through the composition of multiple MRs. The *source test cases* are provided by the users whereas *follow-up* test cases are generated by MTKeras. MR-violation tests are automatically recorded during testing.

The design of MTKeras is centered around two basic concepts: *metamorphic relation input patterns* (MRIPs) [9] and *metamorphic relation output patterns* (MROPs) [5], which describe the relations among the source and follow-up inputs and outputs, respectively. Both MRIPs and MROPs can have multiple levels of abstractions. Examples of MRIPs include *replace* (changing the value of part of the input to another value; cf. $MR_{replace}$ of [6]), *noise* (adding noise to the input data; cf. [7]), *additive* and *multiplicative* (modifying the input by addition and multiplication, respectively; cf. "metamorphic properties" defined by Murphy et al. [4]). Examples of MROPs include *subsume*/*subset* [5, 10], *equivalent* and *equal* [5]. MTKeras

is extendable as it allows a user to plug in new MRIPs and MROPs and configure them into concrete MRs. We have implemented it as a python package for ease of use and **open sourced** it at Github (https://github.com/lawrence415610/Mtkeras.git). The user can perform MT in a simple and intuitive way by writing a single line of code in the following format:

```
Mtkeras(<sourceTestSet>,<dataType>[,<modelName>]).<MRIPs>[.<MROP>]
```

where <sourceTestSet> points to the place where the source test cases are stored; <dataType> declares the type of each element (test case) of <sourceTestData> (e.g., grayscaleImage, colorImage, text, etc); <modelName> (optional) gives the name of the ML model under test. <MRIPs> represents an MRIP or a sequence of MRIPs; and "[.<MROP>]" represents an optional MROP. Note that <modelName> and <MROP> always go together—they are either both present or both absent. For example, when testing an image classification model, we could write:

```
Mtkeras(myTestSet,colorImage,myDNNModel).noise().fliph().equal()
```

which tells MTKeras to use "myTestSet" (an array name) as the set of source test cases, where each test case is a color image, to generate follow-up test cases by first adding a noise point to each image and then horizontally flipping it. The name of the ML model under test is "myDNNModel." The last term, equal(), tells MTKeras to check whether the classification results for the source and follow-up test cases are the same. MTKeras then performs MT automatically and identify all the violating cases. Mtkeras returns an object and the violating cases are stored in its variable named "violatingCases." Note that the model name "myDNNModel" and the MROP "equal()" are optional, without which MTKeras will return a set of follow-up test cases without further tests. The user can then use this set of test cases for various purposes, including but not limited to MT (such as for *data augmentation*).

3 TWO CASE STUDIES WITH MTKERAS

All ML models tested in our case studies are taken from the official website of Keras [1]. The first case study uses the MNIST handwritten digit dataset [2], which contains 70,000 grayscale 28×28-pixel images of handwritten digits, with 60,000 and 10,000 images being the training and test sets, respectively. The Keras model under test has a 98.40% accuracy. To test this ML model, we first define MR_1 as follows: Increasing the gray scale (by 10%) of each of the images should not improve the ML model's accuracy because, intuitively, a darker image is more difficult to recognize. As expected, our experiment returns 0 violations. We then define MR_2 as follows: Adding a random noise point to each of the images should not improve the ML model's accuracy because, intuitively, an image with a noise point is more difficult to recognize. It is interesting to find that, out of 1,000 experiments (hence 1,000 pairs of source and follow-up accuracy scores), 387 have violated MR_2, meaning that in 38.7% of the situations, adding a random noise point has helped to increase the model's accuracy. We plan to conduct further investigation to convey some understanding of the reasons for this phenomenon, for example, *is it because the addition of the noise point happened to accentuate an important ML feature for a specific model?*

A more interesting observation is that combining MR_1 and MR_2 has yielded a violation rate of 45.5% (455 out of 1,000), providing evidence that **composition/combination of MRs can effectively increase the fault-detection effectiveness**.

The second case study applies MTKeras to test four different types of ML models (CNN, RCNN, FastText, and LSTM [1]) trained on an IMDB sentiment classification dataset, a collection of movie reviews labeled by "1" for positive and "0" for negative feelings. We define MR_3 as follows: Randomly shuffling (permuting) the words in each movie review shall dramatically reduce the accuracy of the ML models. The *permutative* MRIP is very popular in MT practice (cf. [4]). The validity of MR_3 is obvious as shuffling the words makes the sentence meaningless. The experimental results, however, is surprising. The results of 100 MT experiments show that shuffling the words only decreases the accuracy by a very small degree (RNN: around 4%, CNN: around 7%, FastText: 0%, LSTM: around 3.5%), indicating that the ML models under test are insensitive to word orders. This case study shows that MRs can help to enhance system understanding, confirming our previous report [9].

4 CONCLUSIONS AND FUTURE WORK

We have presented the design of MTKeras, a generic metamorphic testing framework on top of the Keras machine learning platform, and demonstrated its applicability and problem-detection effectiveness through case studies in two very different problem domains: image classification and sentiment analysis. We have shown that the composition of MRs can greatly improve the problem-detection effectiveness of individual MRs, and that MRs can help to enhance the understanding of the underlying ML models. This work demonstrates the usefulness of metamorphic relation patterns. We have open sourced MTKeras at Github. Future research will include an investigation of the time cost associated with the learning curve for a novice tester to use the tool as well as further extensions and larger-scale case studies of the framework.

ACKNOWLEDGMENTS

This work was supported in part by an Australian Government Research Training Program scholarship and a Western River entrepreneurship grant. We wish to thank Morphick Solutions Pty Ltd, Australia, for supporting this research.

REFERENCES

[1] [n.d.]. https://github.com/keras-team/keras/tree/master/examples
[2] [n.d.]. http://yann.lecun.com/exdb/mnist/
[3] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys* 51, 1 (2018), 4:1–4:27. https://doi.org/10.1145/3143561
[4] C. Murphy, G. Kaiser, L. Hu, and L. Wu. 2008. Properties of machine learning applications for use in metamorphic testing. In *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering*. 867–872.
[5] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortés. Metamorphic testing of RESTful web APIs. *IEEE Transactions on Software Engineering* 44, 11 (2018), 1083–1099. https://doi.org/10.1109/TSE.2017.2764464
[6] L. Sun and Z. Q. Zhou. 2018. Metamorphic testing for machine translations: MT4MT. In *Proceedings of the 25th Australasian Software Engineering Conference (ASWEC 2018)*. IEEE, 96–100.
[7] C. Wu, L. Sun, and Z. Q. Zhou. 2019. The impact of a dot: Case studies of a noise metamorphic relation pattern. In *Proceedings of the IEEE/ACM 4th International Workshop on Metamorphic Testing (MET '19)*. IEEE, 17–23.
[8] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. 2019. Machine learning testing: Survey, landscapes and horizons. https://arxiv.org/abs/1906.10742
[9] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey. Metamorphic relations for enhancing system understanding and use. *IEEE Transactions on Software Engineering*. https://doi.org/10.1109/TSE.2018.2876433
[10] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. H. Tse, F.-C. Kuo, and T. Y. Chen. Automated functional testing of online search services. *Software Testing, Verification and Reliability* 22, 4 (2012), 221–243.