[Invited] Quality Assurance of Machine Learning Software

Shin NAKAJIMA
National Institute of Informatics
Tokyo, Japan
nkjm@nii.ac.jp

***Abstract*—Functionalities of machine learning software are dependent on a set of data input to them; a slight change in a training dataset has much impact on learning parameter values and thus on inference results. ML-based systems bring about a new challenge to quality assurance methods. This paper reviews two traditional views of service and product qualities. Furthermore, it introduces a platform view, in which co-creation of value is a major concern.**

***Index Terms*—dataset diversity, metamorphic testing, service dominant logic, independent assessment**

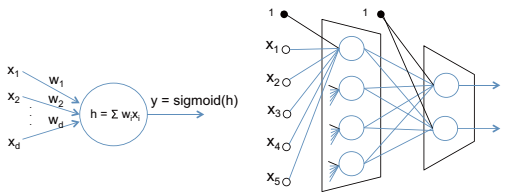
I. Introduction

Machine learning technologies (e.g. [6]) will be matured enough to be a basis of core components in a wide variety of software intensive systems. Because they are nothing more than engineered software products, quality assurance of these systems is mandatory. On one hand, systems to employ machine learning technologies, or ML-based systems for short, have characteristics quite different from those of the other software systems. Functionalities of ML-based systems are dependent on a set of data (a dataset) input to them; a slight change in such a training dataset has much impact on learning results and thus on functional behavior of inference programs. ML-based systems bring about a new challenge to quality assurance methods.

This paper first recalls machine learning problems, especially neural networks [7]. It then presents three distinct views on ML-based systems, which demonstrates that their quality is, indeed, inter-disciplinary. Roughly, the machine learning research community focuses more on service quality than product quality that the software engineering community cares about. Such discrepancy comes from particular characteristics of ML-based systems that producing new unexpected results is a concern. This paper goes further to introduce an alternative view, a platform, in which value-in-context [8] is a primary notion relating to the quality. The platform is, from a technology side, supported with continuous software engineering [4]. Last, an ecosystem is mentioned providing an evaluation system of an independent assessment on quality assurance of ML-based systems.

II. Machine Learning Software

A neural network (NN) is a general framework for machine learning tasks, regressions or classifiers. A perceptron (Figure 1 (a)) is a basic unit. Receiving a set of input $\{x_i\}$, it emits a signal y calculated by applying an activation function σ to a weighted sum of its input; $y = \sigma(\sum_{i=1}^{d} w_i x_i)$.



(a) Perceptron (b) Two Layered Network

Fig. 1. Neural Networks

An NN is a two layered network of perceptrons (Figure 1 (b)). All the input signals are fed into perceptrons in the hidden layer, and then all the signals from these are input to those in the output layer. Let h and r be two activation functions. Given D-dimensional vector $\vec{x}$ and M perceptrons in the hidden layer, signals ($k = 1, \ldots, R$) from the output layer is mathematically defined as below.

$$y_k(\vec{W};\ \vec{x}) = r(\ \sum_{j=0}^{M}\ v_{kj} h(\sum_{i=0}^{D}\ w_{ji} x_i)\ )$$

where v_{kj} and w_{ji} are weights, compactly written as $\vec{W}$, which are collectively called learning parameters.

For a set of input data $\{\langle \vec{x}^n,\ \vec{t}^n \rangle\}$ ($n = 1, \ldots, N$), NN machine learning is an optimization problem using an error function $\mathcal{E}$ to obtain weight values, $\vec{W}^*$.

$$\vec{W}^* \ = \ arg\min_{\vec{W}}\ \mathcal{E}(\vec{W}; \{\langle \vec{x}^n, \vec{t}^n \rangle\})$$

$\mathcal{E}$ is defined in terms of a function ℓ which basically calculates how $\vec{y}(\vec{W}; \vec{x}^n)$ and $\vec{t}^n$ differ. where $\vec{y}(\vec{W}; \vec{x})$ is a vector of the output signals.

$$\mathcal{E}(\vec{W}; \{\langle \vec{x}^n, \vec{t}^n \rangle\}) = \frac{1}{N} \sum_{n=1}^{N}\ \ell(\vec{y}(\vec{W};\ \vec{x}^n),\ \vec{t}^n)$$

$\vec{y}(\vec{W}^*; \vec{x})$ is an inference function to be a core component[1] of ML-based systems.

A standard approach to solving the optimization problem is a steepest descent method (SD). It is basically a numerical

[1] an inference program in vivo

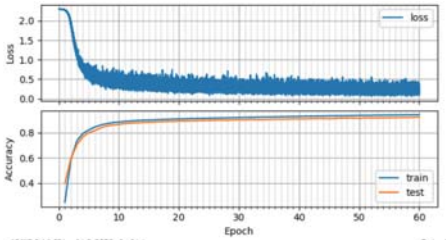


Fig. 2. Loss and Accuracy

calculation such that $W^{new} = W^{old} - \eta \nabla \mathcal{E}$. Calculating new weight values is repeated until the values are converged. In particular, a hyper-parameter η denotes a learning rate. It is a given constant in the iteration of SD. Such parameters are called *hyper* as compared with learning parameters. Choosing an appropriate value of η has much impact on convergence speed and resultant accuracy as well. The search is, however, not guaranteed to reach a global minimum, as the optimization problem is non-convex.

In NN learning, error rates for a training dataset provide a piece of information when to stop the iteration; the iteration terminates when an error rate becomes smaller than a certain threshold. NN learning procedures usually monitor two metrics, loss and accuracy. The loss is a graph of an error function. The accuracy is calculated from rates of reproducing correct tags for a testing dataset. Figure 2 shows such graphs as iterations or epochs proceed.

The graphs actually demonstrates that the search converges at a certain desirable point in the solution space because the loss decreases to be almost stable below a threshold, and the accuracy reaches a satisfactory level. However, a fault-injected program may show almost the same behavior in terms of loss and accuracy [10]. The two metrics are not appropriate to check whether learning programs are free from faults.

III. Quality Assurance: Traditional Views

With regard to quality of machine learning software, we may think of two distinct notions, service quality and product quality.

A. Service Quality

Services denote intangible outputs provided by products, and two notions of quality are related, but not the same. Occasionally, discussions on service quality implicitly assume that product quality is guaranteed. If our interest is service quality, then we do not care much whether machine learning programs have faults or not. Service quality may be acceptable when an accuracy of inference is better than expected, even if the program may contain faults.

From viewpoints of service quality, robustness of machine learning methods is a primary concern. It has been studied in regard to *dataset shift* [13], most of the work concerns with robustness even when training and testing datasets have different probabilistic distributions.

An adversarial example [15] is a well-elaborated data that fools DNN inference programs. They mis-infer when such an input data is given. Responding appropriately to adversarial data is important in view of achieving high service quality.

Work on adversarial attack is categorized into two. The first makes use of detailed knowledge of neural network characteristics (e.g. [5]). The second relies on input and output signals only, and is called black-box attacks [11]. Black-box attacks are more general than those using detailed knowledge, and thus are potentially applicable to a wide variety of DNN machine learning problems. However, computational costs are high because they solve numerical optimization problems.

DeepTest [16] takes another approach to high service quality. It employs a notion of *neuron coverage* to guide generation of training datasets. DeepTest first identifies a set of inactive neurons in a DNN model and then generates a new training dataset to activate such inactive neurons. The method is more concerned with debugging *DNN models* than implemented programs. Identifying such metrics suitable for discussing service quality is important.

B. Product Quality

Product quality is a concern in software engineering, to focus on methods to eliminate faults from implemented programs. Although not rigorous enough, software testing is a practical method for informal assurance on the quality and reliability of computer programs. It has an often forgotten, assumption that correctness criteria must be explicit. Let $f(x)$ be a test target program or system under test (SUT)[2]. Usually, a correct value C^a for an input value of a is known from functional specification documents or as a theoretical value. Then, software testing of $f(x)$ is checking whether the computation result $f(a)$ is equal to C^a. Programs, whose correctness criteria are not known in advance, are called *non-testable* [17].

Specifically, machine learning programs are non-testable because weight parameters $\vec{W}^*$ are unknown. If those values are already known, running machine learning computer programs is not necessary. We may use such known parameter values to implement inference programs that work on new data.

For cases of non-testable programs, $f(a)$ may be checked against some concrete value R^a. Such an R^a is a *golden output*, an execution result of a certain program other than the current SUT. Because usually programs are not proved, correctness is not guaranteed. R^a can be used as a certain criterion, which might be called pseudo oracles.

Metamorphic testing (MET) [1] is a pseudo oracle approach and uses golden outputs as R^a. MET is actually an iterative process to test[3]. A new test data, a follow-up test data, is generated systematically from a current test data. Given an m-th test input $x^{(m)}$, a translation function T generates a new follow-up test input $x^{(m+1)}$ such that $x^{(m+1)} = T(x^{(m)})$. T is

[2]Programs are regarded simply as functions.

[3]Original MET [1] repeats the iteration two times. An extension [9] [10] allows more than two times if necessary.

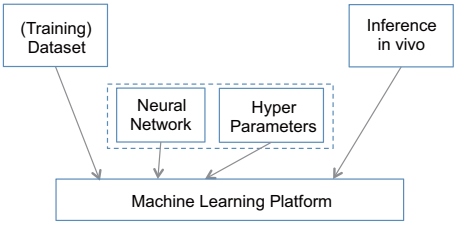


Fig. 3. Machine Learning Platform

derived from functional specifications of $f(x)$ such that an appropriate metamorphic relation $Rel^T(f(x^{(m)}), f(x^{(m+1)}))$ is satisfied. If the relation $Rel^T(f(x^{(m)}), f(x^{(m+1)}))$ is violated for two executions of the same program $f(x)$, one with $x^{(m)}$ and another with $x^{(m+1)}$, then the program $f(x)$ is considered to contain some faults in it. In simple cases, Rel^T can be just an equality, $f(x^{(m)}) = f(x^{(m+1)})$.

MET is, indeed, successful in testing machine learning classifiers such as support vector machines (SVM). [18] introduces six general metamorphic properties that translation functions must satisfy for machine learning classifiers. Further, [9] illustrates a systematic way to derive such translation functions from declarative SVM problem definitions.

Machine learning results are sensitive to *distributions* of data in the training dataset. Software testing with a series of input dataset, that slightly differ from each other, may be effective to enable corner-case testing. Furthermore, note that standard code coverages focusing on control aspects such as C0 or C1, are inadequate, because control flows in machine learning programs are not complicated and thus such coverages are readily satisfied by any non-trivial datasets (e.g. [9]).

Dataset diversity [10] takes into account those characteristics of machine learning problems. Let $D^{(m)}$ be a training dataset, which is also an input to a test target machine learning program $f(X)$; $f(X)$ is a program to implement algorithmic solution of numerical optimization problem. In dataset diversity, a follow-up dataset $D^{(m+1)}$ is such that

$$D^{(m+1)} \ = \ T(\ D^{(m)},\ f(D^{(m)})\ ),$$

where $f(D^{(m)})$ refers to a learning result for $D^{(m)}$. $D^{(m+1)}$ is so chosen to have a data distribution slightly different from $D^{(m)}$. A series of datasets can be calculated starting with a certain initial dataset $D^{(0)}$. The approach is successful in software testing of neural network programs of classifying handwritten numbers [10].

IV. Platform: An Alternative View

Developing ML-based systems involves two distinct phases, the learning and inference. The latter may mis-infer a coming new data even if the learning phase results in parameter values to satisfy a desired accuracy level. Data at the inference may not follow probabilistic distributions of datasets used in the learning phase. Such data are either outliers to be rejected, or new instances to be accepted. The latter include cases of adversarial examples.

Deciding whether data are either cases, outliers or not, is dependent on functional behavior of ML-based systems under consideration. If they are new instances to be responded correctly, a re-learn phase is initiated with a training dataset including new data[4]. Therefore, techniques to enable continuous improvements are mandatory.

ML-based systems involve several key elements as shown in Figure 3; what dataset is used for training, what neural network learning model is selected, what value is chosen for hyper-parameters, and what feedback the inference program in vivo has for initiating re-learning. The inference program is a direct beneficiary and thus may be at a demand side, while the others are at a supply side.

This naive picture is somewhat changed from viewpoints of continuous improvements. Monitoring the inference in vivo has impacts on preparing dataset for training and testing as well. It initiates another learning phase, calculating learning parameter values followed by finalizing adequate hyper-parameter values. Each involves engineering efforts and is worth deserved appropriate rewards because all are contributing to *co-creation of value* [8]; machine learning is, indeed, a service enabling *value-in-context*[5] supported with a platform.

The platform consists of technologies to support evolution of ML-based systems so that they can be of high values continuously, whose importance is recognized well in the software engineering community. When a problem at hand is categorized as complex [14], an agile style software development is mandatory; it is exploratory programming to go hand-in-hand constructing programs and understanding requirements specifications until customers become satisfied with them. Additionally, DevOps is a practice to seamlessly integrate the development and operation.

For ML-based systems, the operation phase may involve monitoring how often they mis-infer for incoming data in vivo, which demonstrates probabilistic distributions of such data are different from a distribution of training dataset at the learning time. Re-learning with a modified dataset would be desirable. Deciding whether re-learning is conducted is dependent on what beneficiaries expect.

The platform to support ML-based systems is, indeed, what continuous software engineering (e.g. [4]) is looking at. Enabling high quality of platform includes process aspects of continuous evolution as well as techniques contributing to both service and product quality.

V. Independent Assessment

Consider, for a moment, that ML-based systems are software products to provide intangible outputs, inference results (cf. Section IV). Because these systems bring about the value-in-exchange, suppliers and clients may exchange formal contracts referring explicitly to technical constraints under

[4]We here assume off-line learning algorithms for simplicity of discussions.
[5]This is contrasted with the value-in-exchange of Good-Dominant Logic.

which products are used. If clients use them in conditions violating the constraints, then disaster resulting from mis-inference is attributed to clients. Clients are responsible for avoiding outliers in view of probabilistic distributions of training dataset. Alternatively, clients make claims, if products do not show expected functional behavior, that the supplier must be responsible for deficiencies.

Unfortunately, capabilities and limitations of ML-based systems are difficult to define in a technically compact manner; capabilities are dependent on a training dataset (cf. Section II). Furthermore, limitations cannot be clearly mentioned, which is demonstrated with adversarial examples; finding new adversarial attacks is continued.

Alternatively, in view of the platform, each technology role (cf. Figure 3) is contributing to co-creation of value. No single role can guarantee capabilities of an ML-based system. The machine learning platform may incorporate evaluation systems to provide a certain formal account of capabilities and limitations.

One crucial aspect of evaluation systems is introducing several levels assigned to products. For example, ISO/IEC 61508, a standard of functional safety, defines four Safety Integrity Levels (SILs). ISO 26262, a safety standard customized to automobiles, also defines four Automobile Safety Integrity Levels (ASILs). Furthermore, ISO/IEC 15408 defines seven Evaluation Assurance Levels (EALs) of the criteria for IT security. Each product may have appropriate such a level depending on its required safety or security concern. Costs for low level products can be low, while high level products are worth invested; satisfying requirements concerns with engineering trade-offs.

Machine learning platform will be defining Trustworthiness Levels (TLs) of ML-based systems. For examples, autonomous cars are given a high TL, while ensuring the level is not easy because they are in operation against open-world contexts. Threats are new adversarial attacks such as [3]. On the contrary, although playing games may be elucidating a certain aspect of human intelligence, these ML-based systems have a lower TL than autonomous cars. The ML-based game systems usually do not have harmful side effects outside. Mis-inference may just be loss in game, and thus their trustworthiness is a less concern from a viewpoint of harmfulness outside.

ML-based systems must be classified regarding to socially required levels of trustworthiness. Those ISO/IEC standards are the results of intensive and extensive discussions in each application domain or sector. However, machine learning technologies are evolving and, at the same time, not matured enough; possibilities of new applications are more emphasized than dependability concerns. A certain systematic methodology is called for to classify capabilities and limitations of ML-based systems.

Evaluation systems will be an integral constituent of the machine learning platform, as they help ensuring expected trustworthiness of an ML-based system. They can be another role contributing to the co-creation of value in view of enhancing the trustworthiness. This observation leads to a notion of independent assessments to implement formal evaluation methodologies. In fact, an independent assessment is practically adapted in ISO/IEC 15408 for IT security. ML-based systems and IT security are common in that both are inevitable to consider the *openness*.

VI. Concluding Remarks

The machine learning platform is basically an ecosystem to enable advanced ML-based systems, and at the same time to avoid *normal accidents* that new technologies sometimes are bringing about [12]. Machine learning technology is more on *can-do* than on *can-assure*. As the technology will be matured enough to be a basis of social infrastructure, academia as well as industry is paying attention to the latter aspect of *can-assure*, which needs further research activities.

Acknowledgment

The work is partially supported by JSPS KAKENHI Grant Number JP18H03224.

References

[1] T.Y. Chen, S.C. Chung, and S.M. Yiu, "Metamorphic Testing - A New Approach for Generating Next Test Cases," HKUST-CS98-01, The Hong Kong University of Science and Technology, 1998.

[2] 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), Article No.4, 2018.

[3] I. Evtimov, E. Eykholt, E. Fermandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song, "Robust Physical-World Attacks on Deep Learning Models," Proc. IEEE CVPR, 2018.

[4] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," The Journal of Systems and Software, No. 123, pp.176-189, 2017.

[5] I.J. Goodfellow, J. Shelens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," Proc. ICRL2015, 2015.

[6] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, The MIT Press 2016.

[7] S. Haykin, Neural Networks and Learning Machines (3ed.), Pearson India 2016.

[8] R.F. Lusch and S.L. Vargo, Service-Dominant Logic: Premises, Perspectives, Possibilities, Cambridge University Press, 2014.

[9] S. Nakajima and H.N. Bui, "Dataset Coverage for Testing Machine Learning Computer Programs," Proc. 23rd APSEC, pp.297-304, 2016.

[10] S. Nakajima, "Dataset Diversity for Metamorphic Testing of Machine Learning Software," Proc. 8th SOFL+MSVL, 2018.

[11] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Berkay Celik, and A. Swami, "Practical Black-Box Attacks against Machine Learning," Proc. Asia CCS 2017, pp.506-519, 2017.

[12] C. Perrow, Normal Accidents: Living with High-Risk Technologies, Princeton University Press 1999.

[13] J. Quinonero-Candela, M. Sugiyama, A. Schwaighofer, and N.D. Lawrence (eds.), Dataset Shift in Machine Learning, The MIT Press 2009.

[14] D.J. Snowden and M.E. Boone, "A Leader ' s Framework for Decision Making," Harvard Business Review, pp.108-119, 2008.

[15] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruma, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," Proc. ICLR, 2014.

[16] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars," Proc. 40th ICSE, 2018.

[17] E.J. Weyuker, "On Testing Non-testable Programs," Computer Journal, 25 (4), pp.465-470, 1982.

[18] X. Xie, J.W.K. Ho, C. Murphy, G. Kaiser, B. Xu, and T.Y. Chen, "Testing and Validating Machine Learning Classifiers by Metamorphic Testing," J. Syst. Softw., 84(4), pp.544-558, 2011.