Towards Deep Learning Specification

Shibbir Ahmed
shibbir@iastate.edu
Iowa State University
Ames, Iowa, USA

Abstract

Deep Learning techniques are applied in software systems rapidly. Therefore, it becomes necessary to specify Deep Learning (DL) application programming interfaces (APIs) for desired output. Unlike traditional software, DL-specific development occupies bugs that exhibit not only crashes but also yield low accuracy and high training time issues. Inspired by the design-by-contract (DbC) methodology, this work proposes a preemptive measure against such bugs, we call it *DL Contract*. *DL Contract* aims to document properties of DL libraries and provide developers with a mechanism to prevent low accuracy and high training time-related bugs during development. One of the main challenges towards *DL Contract* is to specify properties of the training process, which is inaccessible at the functional interface of the DL libraries. Thus, we introduce the notion of *ML variable* that allows developers to specify the properties of model architecture, data, and training behavior. To evaluate the utility of *DL Contract*, we intend to utilize benchmarks from prior works on DL bug detection and repair after implementing *DL Contract* for Python-based DL libraries.

CCS Concepts: • **Software and its engineering** → *Software notations and tools.*

Keywords: Specification, deep learning

ACM Reference Format:
Shibbir Ahmed. 2021. Towards Deep Learning Specification. In *Proceedings of SPLASH Doctoral Symposium (SPLASH 2021).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Problem

Design by Contract (DbC) [19] improves the reliability of software with precise specifications. The contracts cover mutual obligations (preconditions), benefits (post-conditions), and consistency constraints (invariants) [15]. For traditional software approaches like Eiffel [19], JML [16], Spec# [10], Dafny [17] have provided support for writing at the application programming interfaces (API). Those specifications comprise conditions ensured by the client prior to calling the method (*preconditions*), and if the method is called correctly, what it will guarantee (*postconditions*).

Unfortunately, the prevailing DbC methodology, which specifies constraints over the functional interface (formal parameters and return values) of the API methods and fields of the API classes, is not sufficient for documenting deep learning software. In particular, a significant concern is that major DL-specific development issues are related to bugs having accuracy and training time issues. These bugs do not appear as crashes or discrepancies at the functional interface of the DL API. Recent work in software engineering has characterized these bugs, studied fix patterns, and proposed detection and localization strategies. However, there are still research gaps in detecting bugs of DNN models without modifying anything in the client code, importing tools, or instrumenting a model during training. There are several challenges to specify DL libraries, mainly to document clients' code obligations and ensure desired output from the DL software.

2 Motivation

To motivate the need for *DL Contract*, we consider a simple convolution network in Fig. 1 that achieves 99% test accuracy on the MNIST dataset from *Keras* documentation [8]. This DL model is intended to classify images of digits from the MNIST dataset. In this code, MNIST images have been scaled to the [0, 1] range (lines 5-6) and converted to binary class matrices (lines 7-10). After that, at lines 11-19, a Sequential model has been declared with model and layer architecture. The model has been configured for training (line 20) using `Compile` API, and `Fit` API (line 22) will train the model.

Fig. 1 shows four different types of bugs that could appear in this model. 1 high dropout rate of *Dropout* API [9] could cause overfitting, i.e., the model fit exactly against its training data. 2 If `'relu'` activation function has been used in the last layer (line 19) of `Dense` API [1, 4, 5]. 3 If `loss='binary_crossentropy'` (line 20) has been applied as the loss function of `Compile` API [1, 2, 7]. Lastly, 4 if the data is not normalized before it is fed into the `Fit` API [5, 6] (lines 5, 6). Any of the above statements could result in bugs yielding low accuracy and high training time. The *Keras* documentation [3] suggests that all of these values are valid for

SPLASH 2021, October 17–22, 2021, Chicago, Illinois, United States

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

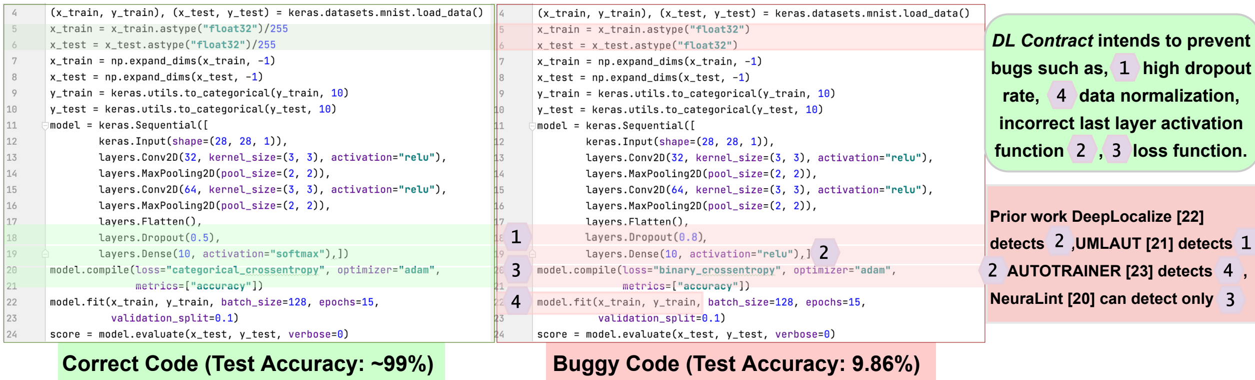

Figure 1. Buggy code (right) [1, 2, 5–7] achieves 9.86% test accuracy, whereas correct code (left) from *Keras* documentation [8] achieves ∼ 99% test accuracy. *DL Contract* intends to detect such bugs with informative error messages localizing the bugs.

the respective APIs. Thus, the documentation is still insufficient to prevent such bugs.

To detect such types of bugs in deep learning programs, we propose a technique for applying the concept of *Design by Contracts* [19] to DL APIs. The expert-defined solutions from the Stack Overflow posts, GitHub are considered client code obligations (*preconditions*). The conditions for better accuracy and training time as the expectation from the DL software are *postconditions* in our proposed *DL Contract* mechanism. Prior work, specifically DeepLocalize [22], UMLAUT [21], AUTOTRAINER [23], and NeuraLint [20] failed to detect all those bugs. *DL Contract* is not dependent on the model's implementation nor on other debugging techniques like those approaches. *DL Contract* also does not require any additional compiler or package installation rather, the mechanism of our approach is designed to be embedded on top of *Keras* APIs by the library developers. *DL Contract* intends to detect all of these bugs and present the end-users with corrective error messages earlier than previous work.

3 Approach

In order to design and implement *DL Contract*, we need to expose meta-level properties of the deep learning training process and deep learning model structure as variables, which we call *ML variable*. Using *DL Contract* with the help of *ML variable*, developers can specify various properties, including expectations about the model structure, to prevent accuracy and training time related bugs. *DL Contract* enables library developers to write contracts on DL library APIs using annotation based approach with `@contract` utilizing the notion of model abstraction and *ML variable*. Thus, those contracts need to facilitate end-users of run time assertion checking on the model, data, and training properties at different program points, e.g., before and after model compilation or during training deep learning models.

For instance, when training a DNN with ReLu as the activation function, the gradients of a large percentage of the neurons are zero, and the training accuracy is low [23]. To specify this, '`zero_gradients_percentage`' (percentage of neurons whose gradients is 0 in recent iterations) *ML variable* is utilized. Thus, we can capture properties of hidden layers' activation functions and corresponding training problems by writing the following contract on *Keras* `Fit` API.

```
@contract(context = 'hidden_layers',activation!='relu',
  zero_gradients_percentage ≤ λ, accuracy ≤ θ)
def fit(self, x=None, y=None,...):
```

Developers can specify by λ and θ thresholds so that training can happen without dying ReLu [23] problem and expected accuracy. In case of a contract violation during training, following message can be shown to the end-users,

```
Contract violation for Model:fit(). zero_gradients_percentage of 12.07 caused
dying relu problem, activation function should not be relu.
```

Next, we describe the technical challenges for design and implement *DL Contract* and how we plan to address them.

Context and *ML variable*: To apply DbC for deep learning APIs, it is necessary to have additional variables that could capture model abstraction and training behavior. Standard precondition and postcondition contracts are enforced on values of the formal parameters and return values of an API method on the attributes of an API class. Therefore, we introduce context and *ML variable* to specify the model abstraction, its data properties, and training behavior.

DL Contract Runtime Assertion: The traditional assertion technique works on code-level, whereas our proposed *DL Contract* provides model abstraction and exposes *ML variable*. Unlike a traditional contract checker, the *DL Contract* must be aware of context and *ML variable*. By using the model abstraction and *ML variable*, *DL Contract* enables library developers to annotate *Keras* APIs. Thus, it benefits end-users to check their model, data, and training properties at different program points. For instance, *DL Contract* can be applied before and after model compilation or at any training stage. Here, we propose an effective algorithm to enforce runtime assertion capturing those properties.

Contextualized Inter-API Call Contracts: *DL Contract* needs to support *ML variable* involving multiple APIs at different stages of the ML pipeline. Conventionally, a contract for an API specifies constraints on the formal parameters of that API method which does not look beyond the scope of that method. *DL Contract* needs to constrain formal parameters across multiple API calls (e.g., at lines 19-20 in Fig. 1).

Post-training Contracts: *DL Contract* needs to support properties afterward training to capture the training behavior of DNNs at different stages of the ML pipeline. As in DL applications, it is often necessary to put constraints on post-training constructs. To solve this issue, we specify desired training behavior with postconditions to prevent training bugs using *DL Contract* for respective training-related APIs.

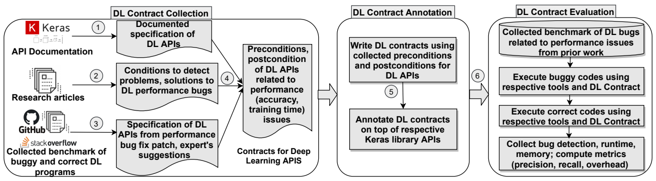

Figure 2. Methodology to collect and evaluate *DL Contract*.

4 Evaluation Methodology

In order to evaluate the *DL Contract*, we plan to collect deep learning contracts (in Fig. 2). In 1, we can use official *Keras* library documentation [3]. In particular, we consider the selection criterion from DL bugs from prior works [18, 20–23]. In 2, we plan to collect a list of state-of-the-art research articles [12–14]. In 3, we intend to collect contracts from state-of-the-art approaches that provide benchmarks of buggy and correct DL programs [20–23]. We plan to use these benchmarks to gather DL programs with better performance (e.g., accuracy, runtime). These benchmarks help to extract performance-related contracts from fix patches and expert-defined solutions. Lastly, in 4, we plan to combine all extracted contracts from all sources into a curated list, which targets DL program performance. To implement *DL Contract*, we propose to extend an open-source package *PyContracts* [11] and address discussed challenges (§3).

We plan to demonstrate the effectiveness of *ML variable* to provide an abstraction of the model architecture and training behavior. To show *DL Contract*'s applicability for real-world programs with performance bugs, in 5 we intend to write those contracts and annotate them on respective *Keras* APIs. After that, in 6 we plan to conduct experiments using the benchmarks and reported the results and analysis in terms of accuracy. We also plan to compare and contrast it against existing tools based on precision and recall performance metrics. Finally, we intend to compute the runtime and memory overhead of our proposed *DL Contract*.

References

[1] 2021. CNN with keras, accuracy not improving. https://stackoverflow.com/questions/50079585/. [Online; accessed Aug-2021].

[2] 2021. How to train and tune an artificial multilayer perceptron neural network using Keras? https://stackoverflow.com/questions/34673164/. [Online; accessed Aug-2021].

[3] 2021. Keras API reference. https://keras.io/api/. [Online; accessed Aug-2021].

[4] 2021. Keras unreasonnably slower than TensorFlow. https://stackoverflow.com/questions/47352366/. [Online; accessed Aug-2021].

[5] 2021. Loss becomes NaN. https://stackoverflow.com/questions/55328966/. [Online; accessed Aug-2021].

[6] 2021. Low accuracy after training a CNN. https://stackoverflow.com/questions/59325381/. [Online; accessed Aug-2021].

[7] 2021. Sigmoid layer in Keras. https://stackoverflow.com/questions/45442843/. [Online; accessed Aug-2021].

[8] 2021. Simple MNIST convnet example from Keras documentation. https://keras.io/examples/vision/mnist_convnet/. [Online; accessed Aug-2021].

[9] 2021. What happens if my Dropout is too high? https://stackoverflow.com/questions/64289118/. [Online; accessed Aug-2021].

[10] Mike Barnett, Manuel Fähndrich, K Rustan M Leino, Peter Müller, Wolfram Schulte, and Herman Venter. 2011. Specification and verification: the Spec# experience. *Commun. ACM* 54, 6 (2011), 81–91.

[11] Brett Graham, William Furr, Karol Kuczmarski, Bernhard Biskup, and Adam Palay. 2010. *PyContracts*.

[12] Nargiz Humbatova, Gunel Jahangirova, Gabriele Bavota, Vincenzo Riccio, Andrea Stocco, and Paolo Tonella. 2020. Taxonomy of real faults in deep learning systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1110–1121.

[13] Md Johirul Islam, Giang Nguyen, Rangeet Pan, and Hridesh Rajan. 2019. A Comprehensive Study on Deep Learning Bug Characteristics. In *ESEC/FSE'19: The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) (ESEC/FSE 2019)*.

[14] Md Johirul Islam, Rangeet Pan, Giang Nguyen, and Hridesh Rajan. 2020. Repairing deep neural networks: Fix patterns and challenges. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1135–1146.

[15] J.-M. Jazequel and B. Meyer. 1997. Design by contract: the lessons of Ariane. *Computer* 30, 1 (1997), 129–130. https://doi.org/10.1109/2.562936

[16] Gary T. Leavens, Albert L. Baker, and Clyde Ruby. 2006. Preliminary Design of JML: A Behavioral Interface Specification Language for Java. *SIGSOFT Softw. Eng. Notes* 31, 3 (May 2006), 1–38.

[17] K Rustan M Leino. 2010. Dafny: An automatic program verifier for functional correctness. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*. Springer, 348–370.

[18] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 175–186.

[19] B. Meyer. 1992. Applying 'design by contract'. *Computer* 25, 10 (1992), 40–51. https://doi.org/10.1109/2.161279

[20] Amin Nikanjam, Houssem Ben Braiek, Mohammad Mehdi Morovati, and Foutse Khomh. 2021. Automatic Fault Detection for Deep Learning Programs Using Graph Transformations. *ACM Trans. Softw. Eng. Methodol.* 30, 5 (2021), 26 pages.

[21] Eldon Schoop, Forrest Huang, and Björn Hartmann. May 8–13, 2021. UMLAUT: Debugging Deep Learning Programs using Program Structure and Model Behavior. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*.

[22] Mohammad Wardat, Wei Le, and Hridesh Rajan. 2021. DeepLocalize: Fault Localization for Deep Neural Networks. In *ICSE'21: The 43rd International Conference on Software Engineering*.

[23] Xiaoyu Zhang, Juan Zhai, Shiqing Ma, and Chao Shen. 2021. AUTOTRAINER: An Automatic DNN Training Problem Detection and Repair System. In *ICSE'21: The 43rd International Conference on Software Engineering*.