Report of Software Engineering and Machine Learning Systems

Huaifeng Zhang

1 Introduction

My research topic is Debloating Machine Learning Systems. According to the definition of Wikipedia, software bloat is a process whereby successive versions of a computer program become perceptibly slower, use more resources like memory, disk space, or processing power. Software bloat has some adverse effects including wasting resources and increasing security vulnerabilities. The process of reducing software bloat is called debloating.

Machine learning systems are proliferating in our life, being part of almost every modern software stack. Like many other software systems, machine learning systems are bloated. Although many researchers in recent years have focused on debloating traditional software systems, few studies have addressed the problem of debloating machine learning systems.

Automated software testing refers to the process of using other software to execute test cases and generate test reports automatically. Software architecture serves as a blueprint for a system, which provides an abstraction to manage the complexity of software systems and establish a communication and coordination mechanism among components. Project management is the process of leading the work of a team to achieve all project goals within the given constraints Automated software testing, architecture design, and project management are critical to software quality.

In this report, we will discuss some research challenges of the above three topics about machine learning systems and debloating.

2 Automated Software Testing

Automated testing for traditional software systems is widely used in industries. The main problems in software testing include huge input and output space, complex system behaviors, poorly understood specifications, etc. Some techniques are proposed to mitigate these problems such as input generation, test input selection and prioritization, adequacy testing criteria, mutation testing and formal verification.

From my point of view, there are 2 main differences between traditional software systems and machine learning systems:

- Behaviors of traditional software systems are programmed by developers but the behaviors of machine learning systems are learned by models. Thus, bugs and unexpected behaviors of traditional software systems are easier to be perceived and explained by humans than that of machine learning systems. For example, if traditional software crashes, it's sure that there is something wrong with the software. But if an image classification system classifies a cat image as a dog, this may be because the model itself does not recognize the image, or because there is an implementation error in another part of the machine learning system.

- Machine learning systems include two different sub-systems: training system and serving systems. The training system is used to train a model and the serving system is

used to deploy the trained model for inference. When training a model, the behaviors of the model are not determined so it's hard to identify if some behavior is expected or unexpected. For some large models, the cost of training the models is very high, which makes it very difficult to test because of a limited budget.

3 Architecture and Design

Software architecture lays the foundation of a software system. A badly designed architecture could cause a lot of difficulties in software development and maintenance. As described in [1], there is some hidden technical debt in machine learning systems including boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues and changes in the external world. Besides, it's common for machine learning systems to have high-debt design patterns. For example, because of the wide usage of general-purpose packages, a lot of glue code is written to get data into and out of these packages. Compared with traditional software systems, there is also a lack of widely accepted abstraction for machine learning systems, such as an appropriate interface to describe a model or a prediction.

Debloating machine learning systems could be a potential way to reduce this technical debt. But compared to traditional software systems, the bloat of machine learning system may be more difficult to detect because bloat exists at the architecture level rather than the code level. Here is a list of potential challenges or research questions about debloating machine learning systems that I'm interested in:

- ML models are usually written in Python or C/C++ based on some framework like TensorFlow or PyTorch. This programing pattern is quite different from traditional programing pattern. How can we analyze if this programing pattern exists any bloat?
- The trained ML models are usually distributed in a certain file format on a platform, such as ModelZoo. Users could download and use these trained models directly. But the size of these models may be too large for embedded devices. So does there exist bloat in these model files? Are there any approaches to debloat it if bloat exists?
- ML models usually need a large amount of runtime resources, such as memory, GPU etc. Are there any memory bloat or GPU bloat in ML models? If it exists, how can we debloat it?
- ML systems usually include data storage modules, data preprocess modules and feature extraction modules. These modules do some data-intensive tasks. Do these modules have any bloat? If they have, how to debloat it? Can we develop a new bloat-aware design pattern for these tasks?
- Traditional machine learning approaches require centralizing and training data on one machine or in a datacenter which has system level bloat. Can we improve these kind of systems at a system level to reduce the bloat? For example, a new approach called Federated Learning is proposed, which enables mobile phones to collaboratively learn a shared model while keeping the training data on device, decoupling the ability to do machine learning from the need to store the data in cloud. Compared to centralizing ML systems, Federated Learning consumes less power. So can we develop a new ML system design pattern which avoids bloat or is easier to debloat?

4 Project Management

There are some fundamental differences between traditional software project management and ML-based software project management as described in [2]. The differences are listed below.

- Compared to traditional software, the data schema of ML-based software change frequently. And there is no well-designed technology to version data.
- Machine learning models are hard to customize and reuse due to difficulties of modifying the models and parameters.
- It's harder to modularize ML-based software. Because: (1) models are hard to extend to new functionality; (2)models interact in non-obvious ways. Each model's results will affect one another's training and tuning processes.

The above differences in project management between traditional software and ML-based software may result in some research challenges and opportunities in debloating machine learning systems. For example, as described in [2], data management is ranked as the top challenge of the ML-based software development process. Debloating traditional software systems focus more on *code bloat*. Due to a lack of well-designed technology to version data, a lot of outdated data may still occupy many storage resources. Thus, *data bloat* could be a potential issue for machine learning systems.

5 Future Trends

ML will "feed software", i.e. software developers will be assisted by ML technology. Both the speed and quality of software development will be improved with the help of ML. More and more auxiliary tools based on ML will emerge. For example, GitHub Copilot uses ML to suggest code and entire functions in real time. And this tool is integrated into IDE. Developers could just write a comment describing the logic they want and GitHub Copilot will immediately suggest code to implement the solution. In doing so, code is more standardized and developers could spend less time creating boilerplate and repetitive code patterns. Both the speed and quality of software development get increased. Besides, ML will also empower more software to have smarter features. So end users of the software will also benefit from ML. For instance, ML-based speech-to-text software helps users record meeting minutes. In summary, ML will "feed software". Both software developers and software users will benefit from ML.

Data Science and ML will just become another part of Software Engineering. ML has only shown great capabilities in specific tasks, such as image classification, natural language processing, etc. These tasks are well-defined and a lot of hand-crafted, structured training data can be obtained. But there are some other tasks in software engineering that are not well-defined and lack training data. Or some tasks are strongly related to business requirements which are not general enough to make it worthwhile to train a model for them. In other words, ML only addresses a few problems in the area of software engineering. ML's ability to solve these problems is encapsulated in interfaces or libraries that are used by other parts of the software. Thus, Data Science and ML are a part of Software Engineering. They can be deemed as another way to implement certain functionalities.

Software engineering needs to adapt to machine learning systems. As described in [1] and [2], existing machine learning systems have some technical debt and differences from traditional software systems. Some traditional software Engineering techniques can't be used in machine learning systems. Such as Automated testing. Software engineering faces many

challenges in the machine learning system area. Like how to design software systems that support the entire machine learning lifecycle, which includes programming interfaces, data pre-processing, interpretation and analysis of output, debugging, and monitoring. How to design machine learning systems that meet the requirements of energy consumption, latency, privacy, fairness, etc. New software engineering techniques need to be developed to address these difficulties.

References

[1] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[2] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 291–300, 2019.