Developing Bug-Free Machine Learning Systems With Formal Mathematics

Daniel Selsam, Percy Liang, David Dill
Department of Computer Science, Stanford University

Problem

Noisy data, non-convex objectives, model misspecification, and numerical instability can all cause undesired behaviors in machine learning systems. As a result, detecting actual implementation errors can be extremely difficult.

Introduction

We demonstrate a methodology for developing machine learning systems using an interactive proof assistant, in which one writes an implementation of a machine learning system along with a formal specification, and numerical instability can all oversights and hidden assumptions. Once it has been proved, every interested party can be certain that the final implementation is correct.

To illustrate our methodology, we walk through an interactive proof assistant, in which one writes an implementation of a machine learning system along with a formal specification, and numerical instability can all oversights and hidden assumptions. Once it has been proved, every interested party can be certain that the final implementation is correct.

Case Study: Certified Stochastic Computation Graphs

Stochastic Computation Graphs

Stochastic computation graphs extend the computation graphs that underly systems like TensorFlow and Theano by allowing nodes to represent random variables and by defining the loss function to be the expected value of the stated loss function over all the random choices in the graph.

\[
L(W_1, W_2) = E_{\sigma \sim N(\cdot, I)} [\log \sigma(W_2)]
\]

Figure 1: An example stochastic computation graph representing a simple variational autoencoder.

Sketch

The first step of applying our methodology is to write down informally what it means for the system to be correct. Suppose \( G \) is a stochastic computation graph with \( n \) nodes that takes one parameter \( \theta \). Then \( G, \theta \) together define a distribution over the values at the \( n \) nodes \((X_1, \ldots, X_n)\). Let \( \text{cost}(G, X_{1:n}) \) be the function that sums the values of the leaf nodes. Our primary goal in developing Certigrad is to write a (stochastic) backpropagation algorithm \( \text{sbprop} \) such that for any graph \( G \),

\[
E_{Q,\theta} [\text{sbprop}(G, \theta, X_{1:n})] = \nabla_{\theta} \left( E_{Q,\theta} [\text{cost}(G, X_{1:n})] \right)
\]

While this equation may seem sufficient to communicate to a human with a mathematical background, it requires much more clarification and precision to communicate it to a computer.

Formal Specification

The next and most important step is to make the specification precise using formal mathematics:

\[
\begin{align*}
\text{def sbprop_spec (sbprop : \forall \mathbb{R} \rightarrow \mathbb{R}) : Prop := } \\
\forall (n : \mathbb{N}) (\theta : \mathbb{R}) (G : \text{WellFormed} G) (G \rightarrow \text{GradExtGen} G, \theta) -> \\
\forall (x : \mathbb{R}) (\sigma : \mathbb{R}^n) (\text{gsplus} \theta x n, \text{gsplus} \theta x) \\
\end{align*}
\]

Proving this theorem in the interactive proof assistant systematically exposes all implementation errors, oversights and hidden assumptions. Once it has been proved, every interested party can be certain that the implementation is correct without needing to trust any human involved, to understand how the program works, nor to test it empirically.

Running the System

We proved that our system is correct in an idealized mathematical context with infinite-precision real numbers, and to actually execute it we need to replace all real numbers in the program with floating-point numbers. We also replace all tensors with an optimized tensor library (Eigen) to improve performance.

Experiments

As an experiment, we trained an autoencoding variational Bayes model on MNIST using both Certigrad and TensorFlow, and found that Certigrad is nearly as efficient as TensorFlow.

Formality can make things easier

Unlike in much of traditional verification, our primary goal is not to facilitate perfect code but rather to make it easier to build machine learning systems to a developer’s existing standards. It is hard to write software that cannot be adequately tested empirically, especially if it involves advanced mathematics or makes random choices, and we want to offload as much of the challenging mental work as possible to the computer. The more precisely the developers communicate to the computer what they are trying to implement, the more demanding the work they can relegate to it.

Conclusion

Developing trustworthy machine learning systems is a fundamental challenge for the field, and one that is not adequately addressed by the status-quo methodology. We have demonstrated a new way to build machine learning systems that can flush out implementation errors systematically without relying on empirical assessments of the system’s behavior. We believe our methodology could be useful for developing a wide range of machine learning systems, and that it may already be economical for high-assurance applications.

Tools

We implemented Certigrad in the brand new interactive proof assistant, Lean, being developed by Leonardo de Moura at Microsoft Research.