Cognitive Computing Safety: The New Horizon for Reliability

YUHAO ZHU
VIJAY JANAPA REDDI
University of Texas at Austin

Recent advances in cognitive computing have brought widespread excitement for various machine learning–based intelligent services, ranging from autonomous vehicles to smart traffic-light systems. To push such cognitive services closer to reality, recent research has focused extensively on improving the performance, energy efficiency, privacy, and security of cognitive computing platforms.

Among all the issues, a rapidly rising and critical challenge to address is the practice of *safe cognitive computing*—that is, how to architect machine learning–based systems to be robust against uncertainty and failure to guarantee that they perform as intended without causing harmful behavior. Addressing the safety issue will involve close collaboration among different computing communities, and we believe computer architects must play a key role. In this position paper, we first discuss the meaning of safety and the severe implications of the safety issue in cognitive computing. We then provide a framework to reason about safety, and we outline several opportunities for the architecture community to help make cognitive computing safer.

Safety in Cognitive Computing Systems

Safety in cognitive computing is inherently associated with the accuracy issue in machine learning. Just as with any system-level behavior, accuracy can be classified into two categories: average-case and worst-case accuracy. The latter determines a cognitive computing system's safety. Improving the worst-case accuracy, however, is hard due to the lack of interpretability in machine learning. Machine learning algorithms are inherently stochastic approximations of the ground truth.

Meaning of Safety

Most machine learning systems today focus on the average-case accuracy. In mature machine learning–based application domains, such as image classification, the average accuracy of well-trained machine learning models could be 99 percent. However, just as datacenter systems suffer from the long-tail latency issue,[1] machine learning systems suffer from tail accuracy, in which a few requests exhibit poor accuracy due to uncertainties in the machine learning models, and form a long accuracy tail distribution that is hard to curtail.

Tail accuracy leads to poor worst-case accuracy guarantees. In mission-critical systems, worst-case accuracy is known to raise serious safety concerns. For example, the autopilot system in a self-driving car might be working ideally for millions of miles, but it only takes one life-threatening incident to cause significant distrust in such autonomous systems.[2]

Worst-Case Accuracy

The issue of tolerating worst-case accuracy is not unique to the cognitive computing domain. For instance, in the aviation industry, engineers constantly address the issue of safety. But there is a notable difference between cognitive systems and today's prevalent aviation systems that make it fundamentally hard to guarantee safety in cognitive computing—that is, the *system interpretability.*

System interpretability means the ability to rationalize and identify the root cause of a system failure. From time to time, the aviation industry suffers from mid-air tragedies, but for every such tragedy, investigators can understand what exactly went wrong, how to fix it, and, most importantly, how to prevent such incidents from happening again in the future. This is because flight control is based on interpretable control theory, physics, and aerodynamics. For example, in the case of Air France Flight 447,

Editor's note: We invited some domain experts to discuss the opportunities and challenges surrounding cognitive architectures. The following are their views on the topic. —*Alper Buyuktosunoglu and Pradip Bose*

Published by the IEEE Computer Society

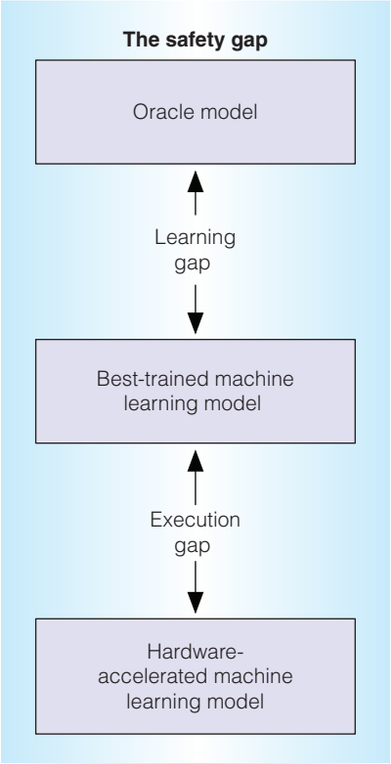

Figure 1. The safety gap in cognitive computing.

investigators were able to accurately attribute the crash to an aerodynamic stall, which was then attributed to the mishandling of the pitot tubes being obstructed by ice crystals. The whole accident was a pivot point for the civil aviation industry, leading to an overhaul of how measurement devices are installed and how pilots are trained to handle instrument anomalies.[3]

In contrast, the same ability to find an incident's root cause and learn from it cannot be said for cognitive systems because, at least at present, we lack a deep understanding of how and why machine learning model works in the first place. The lack of system interpretability will only worsen as we begin to compose many such systems together, effectively increasing the system's opaqueness. If we cannot explain how each component works, we cannot attribute the root cause of a whole system failure to a particular component. For example, it is unclear how we would improve a self-driving car—a multicomponent system involving several decision-making stages—to prevent even the same incident that led to a particular crash from recurring.

Bridging the Safety Gap

Safety in cognitive computing systems is a multidisciplinary problem. However, computer architects are no strangers to building safe systems. We commonly refer to it as "reliability." For decades, the computer architecture community has managed to build highly resilient and fault-tolerant architectures. For instance, we guardband, or allocate a large operating margin, to tolerate process, thermal, and voltage variations.[4] We have also established fundamental redundancy-based techniques, such as triple module redundancy (TMR) and instruction duplication, to detect and recover from errors.[5]

Resilient architectures and safe cognitive systems share a similar goal: guaranteeing expected system behaviors in the event of a failure. Therefore, architects have the unique opportunities to transfer the experience of building resilient architectures to building safe cognitive computing architectures, and to design new solutions to overcome new safety challenges. To that end, we introduce the notion of the safety gap—a framework for computer architects to reason about the safety issue in cognitive computing.

The Safety Gap

A cognitive system's safety gap refers to the gap between the worst-case accuracy guarantee that an oracle system demands and the actual accuracy that a particular implementation provides. The safety gap is the composition of two gaps: learning and execution (see Figure 1).

The learning gap refers to the gap between the oracle and the best-trained machine learning model. The learning gap exists because even the best machine learning model is likely not a perfect representation of the objective reality, and thus introduces error in certain scenarios. For instance, an image classifier mistakenly recognizing a cat in an image as a dog is the result of the learning gap. The learning gap is typically introduced during the model training stage.

Using reliability lingo, errors introduced by the learning gap can be regarded as a form of permanent (as opposed to transient) faults caused by design bugs. Therefore, they are not amenable to traditional backward error-recovery techniques (such as checkpointing), nor to forward error-recovery techniques (such as TMR or execution duplication[6]).

The execution gap, on the other hand, is introduced during the model execution (inference) stage. The execution gap is introduced in two forms. First, to improve the performance and energy efficiency of model inference, hardware architects often build specialized accelerators that rely on various optimizations, such as weight compression, data quantization, and static RAM fault injection.[7,8] These optimizations are unsafe because they intentionally trade off accuracy with execution efficiency. Second, model execution could also suffer from traditional reliability emergencies, such as memory failures, circuit defects, and real-time violations.[9,15] Overall, hardware execution introduces additional sources of error, exacerbating the worst-case accuracy.

To improve the safety of cognitive computing, computer architects should have two objectives. First, we must ensure that in building hardware accelerators, we do not introduce an execution gap, while still providing the performance and energy benefits of hardware specialization. Second, we must provide vehicles that help improve model learning accuracy and thereby minimize the learning gap.

To that end, we discuss a few potential directions under both objectives to foster research in this emerging and important topic. Broadly, our suggestions involve tools, architecture-level design, and principles that dictate accountable system operation.

Avoiding the Execution Gap

We must build systems that provide the performance and energy benefits of hardware specializations with a minimal execution gap.

Define safety semantics. In fault-tolerant systems, failure semantics are well-established.[11] Failure semantics dictate how a system degrades gracefully and in what expected manner if a failure occurs. We need to define a similar set of safety semantics at the architecture layer for systems containing machine learning accelerators. The semantics need to specify which safety violations are tolerable and intolerable, and how the system degrades during a failure.

Build a safe whole out of less-safe parts. Although one accelerator might introduce the execution gap, an ensemble of them might not.[12] One promising approach is to build a cluster of accelerators in which some implement simple models that are interpretable but provide less average-case accuracy (for example, linear regression, decision tree), while others implement complex models that are difficult to interpret but more accurate on average (such as neural networks). Such a system improves the worst-case accuracy by routing requests to the simple models when they can be interpreted as safe on simple models.

Combine control theory with machine learning. It is promising to leverage machine learning to take care of average-case accuracy while relying on control theory principles for worst-case accuracy.[13] Computer architects have already started using control theory to optimize various architectural metrics, such as performance and power.[14] Accuracy is a natural next step.

Minimizing the Learning Gap

We must investigate systems and architectural support that helps reduce the learning gap.

Codesign machine learning algorithms with hardware. The acceleration of machine learning algorithms would let algorithm designers quickly iterate ideas and speed up the process of closing the learning gap. The key is to codesign the algorithm and the hardware. Although most architecture research so far has primarily focused on convolutional neural networks (CNN), more focus should be dedicated to understanding state-of-the-art algorithms and learning techniques beyond just CNNs.[15]

Profiling and debugging support. In aviation systems, the flight data recorder and cockpit voice recorder are critical in assisting investigators to pinpoint the root cause of an air crash. Similarly, we need to provide extensive profiling and debugging support to help system designers understand instances where the cognitive system does not perform as we expect. We will make tradeoffs between instrumentality and intrusiveness—that is, to balance between the ability to collect as much relevant information as we want and the amount of perturbation introduced.

Hardware support for formal methods. Formal methods will be ever-important in providing safety guarantees in cognitive computing. Hardware support is necessary to enable practical formal verification on large-scale machine learning systems. The key is to realize that machine learning is an application domain that relies heavily on linear algebra, and thus it is possible to specialize the hardware support for formal verification to maximize efficiency.

Cognitive computing is redefining how we perceive and interact with the world. As engineers who build systems, the responsibility rests on us to design systems that enable safe and robust cognitive computing from the bottom up, thereby pushing cognitive systems one step closer toward widespread usage. We hope that the perspectives we have shared here will raise the awareness of cognitive computing safety in the architecture community and foster meaningful discussions that lead to new research in building safe cognitive computing. MICRO

References

1. J. Dean and L.A. Barroso, "The Tail at Scale," *Comm. ACM*, vol. 56, no. 2, 2013, pp. 74–80.
2. J. Golson, "Tesla Driver Killed in Crash with Autopilot Active, NHTSA Investigating," *Verge*, 30 June 2016; www.theverge.com/2016/6/30/12072408/teslaautopilot-car-crash-death-autonomous-model-s.
3. "Final Report on the Accident on 1st June 2009 to the Airbus A330-203 Registered FGZCP Operated by Air France Flight AF 447 Rio de Janeiro–Paris," BEA, July 2012; www.bea.aero/docspa/2009/f-cp090601.en/pdf/f-cp090601.en.pdf.
4. S. Borkar et al., "Parameter Variations and Impact on Circuits and Microarchitecture," *Proc. 40th Ann. Design Automation Conf.*, 2003, pp. 338–342.
5. D.J. Sorin, *Fault Tolerant Computer Architecture*, Morgan and Claypool, 2009.
6. J. Chang, G.A. Reis, and D.I. August, "Automatic Instruction-Level Software-Only Recovery," *Proc. Int'l Conf. Dependable Systems and Networks*, 2006, pp. 83–92.
7. B. Reagen et al., "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators," *Proc. 43rd Int'l Symp. Computer Architecture*, 2016, pp. 267–278.
8. S. Han et al., "EIE: Efficient Inference Engine on Compressed Deep Neural Network," *Proc. 43rd Int'l Symp. Computer Architecture*, 2016, pp. 243–254.
9. J. Yoshida, "ARM Raises Bar for Safety, Determinism," *EE Times*, 19 Sept. 2016; www.eetimes.com/document.asp?doc_id=1330483.
10. O. Temam, "A Defect-Tolerant Accelerator for Emerging High-Performance Applications," *Proc. 39th Int'l Symp. Computer Architecture*, 2012, pp. 356–367.

11. F. Cristian, "Understanding Fault-Tolerant Distributed Systems," *Comm. ACM*, vol. 34, 1993, pp. 56–78.
12. T.G. Dietterich, "Ensemble Methods in Machine Learning," *Proc. Int'l Workshop Multiple Classifier Systems*, 2000, pp. 1–15.
13. D. Beyer, *The Future of Machine Intelligence: Perspectives from Leading Practitioners*, O'Reilly, 2016.
14. R.P. Pothukuchi et al., "Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures," *Proc. ACM/IEEE 43rd Ann. Int'l Symp. Computer Architecture*, 2016; doi:10.1109/ISCA.2016.63.
15. R. Adolf et al., "Fathom: Reference Workloads for Modern Deep Learning Methods," *Proc. IEEE Int'l Symp. Workload Characterization*, 2016; doi:10.1109/IISWC.2016.7581275.

Yuhao Zhu is a PhD candidate in the Department of Electrical and Computer Engineering at the University of Texas at Austin. Contact him at yzhu@utexas.edu.

Vijay Janapa Reddi is an assistant professor in the Department of Electrical and Computer Engineering at the University of Texas at Austin. Contact him at vj@ece.utexas.edu.

The Design and Evolution of Deep Learning Workloads

Robert Adolf
Saketh Rama
Brandon Reagen
Gu-Yeon Wei
David Brooks
Harvard University

The past decade has witnessed the reemergence of a connectionist approach to solving several classes of challenging artificial intelligence problems. This family of strategies is collectively known as representation learning, hierarchical learning, or, most popularly, deep learning. The success of deep learning, like many facets of cognitive computing, is the result of a confluence of progress in three separate areas, rather than a single, monumental breakthrough. These three areas include the collection and curation of massive datasets, advances in machine learning algorithms, and the ever-increasing power of computational hardware. These three phenomena form a virtuous cycle. Success in one area facilitates growth in the other two, along with increasing demand for it. For instance, the landmark win of a deep neural network at the ImageNet Large Scale Visual Recognition Challenge in 2012 was the result of a massive new set of training data[1] (two orders of magnitude larger than its closest predecessor), several clever novel modifications to a many-layer convolutional neural network,[2] and use of high-performance hardware (among the first to leverage GPUs for deep learning).

That cognitive computing should be characterized as much by data and hardware as algorithms is not surprising: the very definition involves learning by example at scale. However, it does suggest that carrying out research in this field is perhaps unique, in that one cannot make ample headway without considering all three aspects. This multidimensional constraint is felt keenly in the creation and curation of representative workloads for deep learning problems. Benchmarks and proxy applications must strike a balance between simplicity and faithful reproduction, accurately capturing all fundamental aspects of the programs they represent while remaining easy to understand, use, and transform. Doing this across several axes is challenging, even more so given the frenetic pace of innovation and upheaval in the field. We believe the right approach is first to design workloads that capture the unique aspects of deep learning models, data, and implementations, and then to embrace change and plan for continuous evolution.

Design

Building good deep learning benchmarks means getting three things right: choosing the right models, respecting the impact of data, and faithfully reproducing unique implementation details. We

Table 1. The Fathom workloads

Model	Dataset	Style	Purpose and legacy
Seq2Seq	WMT-15	Supervised, recurrent	Direct language-to-language sentence translation. State-of-the-art accuracy with a simple, language-agnostic architecture.
MemNet	bAbI	Supervised, memory network	Facebook's memory-oriented neural system. One of two novel architectures that explore a topology beyond lattices of neurons.
Speech	TIMIT	Supervised, recurrent, full	Baidu's speech recognition engine. Proved purely deep-learned networks can beat hand-tuned systems.
Autoenc	MNIST	Unsupervised, full	Variational autoencoder. An efficient, generative model for feature learning.
Residual	ImageNet	Supervised, convolutional	Image classifier from MSR Asia. Dramatically increased the depth of convolutional networks. ILSVRC 2015 winner.
VGG	ImageNet	Supervised, convolutional, full	Image classifier demonstrating the power of small convolutional filters. ILSVRC 2014 winner.
AlexNet	ImageNet	Supervised, convolutional, full	Image classifier. Watershed for deep learning by beating hand-tuned image systems at ILSVRC 2012.
DeepQ	Atari ALE	Reinforcement, convolutional, full	Atari-playing neural network from DeepMind. Super-human performance on many Atari2600 games, without any preconceptions.

present our case in the context of our experience in designing Fathom, a set of reference workloads for deep learning (see Table 1).[3]

Models

The most visible decision for a workload suite is the choice of which models to include. In Fathom, we used three criteria to select eight models from a wide array of candidates: representativeness, diversity, and impact. The first is clear: our choices should reflect the best of what the deep learning community has come up with. Because there are many models that could rightly claim this status, the need to limit the size implies a need for diversity; each model should bring something unique to the table. Finally, "impact" reflects the degree to which a particular technique has changed the landscape of deep learning research. We cannot predict the future of deep learning, so we instead tried to choose methods that have imparted fundamental lessons to the work that came after—lessons that will continue to be relevant even as subsequent research builds on them.

Datasets

Data also plays a central role in machine learning workloads, even for architects and system designers. Although it is true that some fundamental deep learning techniques (such as matrix math, convolution, and backpropagation) are somewhat agnostic to the values of their inputs, the role of data is broader. Many problem domains are heavily affected by how their data is being used. For instance, most supervised learning problems have two different operational modes: training, which involves massive amounts of fixed data and an emphasis on throughput, and inference, which involves a stream of unseen data and a lean towards latency. The same model can exhibit different computational characteristics depending on which environment it is used in. Additionally, much of the research in executing deep learning problems centers around exploiting features unique to neural networks or a specific model structure. Sparsity in weight values, batchsize-convergence tradeoffs, and the degree of downsampling in pooling operations are just a few features that depend heavily on the characteristics of the inputs under consideration.

Implementation

Writing reference workloads involves a balancing act between faithfully mimicking praxis while preserving ease of use for researchers. One example of this is the widespread adoption of high-level programming frameworks such as TensorFlow or Torch. These frameworks provide two main benefits: they abstract the underlying hardware interface away from the programmer, and they provide tested libraries of kernels that act as a productivity multiplier. They have changed the development landscape, largely for the better, and it is no longer possible to create a realistic set of deep learning workloads without taking them into account. All eight Fathom models are written on top of TensorFlow. On the other hand, no such consensus has been reached on the layout of learning models, the staging and preprocessing of data, or the mechanisms that drive high-level control flow. It is common to see two implementations of the same model that are almost unrecognizable. Because these choices are more a matter of taste than any fundamental property of deep learning models, Fathom imposed a standard structure and set of interfaces over all its workloads. This greatly simplified cross-model instrumentation, data collection, and experimentation for its users.

Evolution

Deep learning is a field in flux, and a workload suite designed for such an environment must have a plan for adapting. Graceful evolution is an extension of good design: the core principle is to understand which aspects of a workload are intrinsic and which are a product of the current state of the art. For instance, although it's likely that the set of models included in Fathom will change, their selection criteria will not. One convenient way to understand this idea is to look backwards at the developments leading to the present—that is, to understand what changes Fathom would have had to weather had it been released earlier.

Models

All but one of the current Fathom workloads were published since 2014, but most have predecessors that would have been replaced. For instance, DeepSpeech was a breakthrough in pure deep learning speech recognition, but many prior state-of-the-art systems used a combination of hidden Markov models and neural networks. The more interesting change would have been the introduction of read-write networks. Memory networks and neural Turing machines both arrived in 2014, and while no work is built in a vacuum, it is unlikely Fathom would have had something similar. The same is probably true for reinforcement learning: the concept has a long history, but it needed a champion, DeepMind, to make it a core theme in deep learning. Fathom would probably have grown in size over the past several years, in addition to needing to replace its speech model. This is a trend that is almost certain to continue. Even now, it seems likely that additional advances in speech and language processing will require both of Fathom's recurrent models to evolve, and new architectures like binary-valued networks are on the horizon.

Datasets

Surprisingly, most of Fathom's current datasets are relatively stable. ImageNet has not seen radical changes since its introduction, and MNIST and TIMIT have long histories. The largest change would have been the introduction of the Arcade Learning Environment—the Atari emulator used by Fathom's DeepQ model. Although ALE's inputs are not substantially different from older image datasets, its use and integration are. Training and inference with deep-Q learning is a substantially different beast because it requires two-way, online communication. The underlying trend here is refreshingly optimistic: datasets change because deep learning is improving. While ImageNet will probably remain in Fathom, it seems likely that a new source of visual data will augment it, because recent models have surpassed human performance. Additionally, it seems likely that new datasets using video, graphs, or mixed-mode inputs could merit inclusion.

Implementation

Superficially, an older Fathom would appear substantially different, because TensorFlow was not made public until late 2015. However, the use of high-level frameworks has been a clear trend for several years, so it is likely that Theano, Caffe, or Torch would have been used instead. All four frameworks share similarity in their designs and interfaces. The largest difference would have been the effort required to implement some of the models. Although today's frameworks are all converging on support for most of the techniques Fathom uses, that was not true three years ago. Many of the primitives were implemented in only one library, and most of the analysis tools constructed to characterize the Fathom workloads would have been substantially more difficult. This is largely a result of maturity. Modern deep learning frameworks have larger user bases and more developers, and most common operations are well-supported in all platforms. Given this convergent evolution, it is unlikely that Fathom will need to change its implementation framework in the future. On the other hand, Fathom is facing a clear need to adapt to another trend: fixed-precision and packed arithmetic. The use of non-floating-point math and limited precision has been known for decades, but most of the work in deep learning has been focused on improving accuracy, discovering new models, and applying them to new problems. As deep learning applications are deployed in mainstream scenarios, however, efficiency and speed have become a central concern. Many frameworks have introduced some form of packed arithmetic, supported by vector instructions on CPUs and more recently by double-speed, half-width operations on GPUs. This trend is only increasing in importance, and Fathom will need to adopt some form of it to keep up.

Deep learning is a protean field, and workloads for it must be living projects. This is a challenge for maintainers as well as researchers, but it also reflects the success of the virtuous cycle that drives it. Evolution implies that all three facets—models, data, and hardware—are still moving forward in lockstep. Moreover, our experience with Fathom suggests that there are consistent principles underpinning the process that can guide the requisite adaptation. We look forward to the bright future of deep learning, and we believe that accurate, practical, and fluid workloads will continue to play an important role in its progress. MICRO

References

1. O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge," *Int'l J. Computer Vision*, vol. 115, no. 3, 2014, pp. 211–252.
2. A. Krizhevsky, I. Sutskever, and G.E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems* (NIPS), 2012, pp. 1097–1105.
3. R. Adolf et al., "Fathom: Reference Workloads for Modern Deep Learning

Methods," *Proc. IEEE Int'l Symp. Workload Characterization* (IISWC), 2016, pp. 1–10.

Robert Adolf is a PhD candidate in the School of Engineering and Applied Sciences at Harvard University. Contact him at rdadolf@seas.harvard.edu.

Saketh Rama is a PhD student in the School of Engineering and Applied Sciences at Harvard University. Contact him at rama@seas.harvard.edu.

Brandon Reagen is a PhD candidate in the School of Engineering and Applied Sciences at Harvard University. Contact him at reagen@fas.harvard.edu.

Gu-Yeon Wei is the Wei Gordon McKay Professor of Electrical Engineering and Computer Science at Harvard University. Contact him at guyeon@eecs.harvard.edu.

David Brooks is the Haley Family Professor of Computer Science at Harvard University. Contact him at dbrooks@eecs.harvard.edu.