DOI:10.1145/3277567

R. Benjamin Shapiro, Rebecca Fiebrink, and Peter Norvig

▸ **Mark Guzdial,** Column Editor

Education

How Machine Learning Impacts the Undergraduate Computing Curriculum

The growing importance of machine learning creates challenging questions for computing education.

IMAGE BY METAMORWORKS

MACHINE LEARNING NOW powers a huge range of applications, from speech recognition systems to search engines, self-driving cars, and prison-sentencing systems. Many applications that were once designed and programmed by humans now combine human-written components with behaviors learned from data. This shift presents new challenges to computer science (CS) practitioners and educators. In this column, we consider how machine learning might change what we consider to be core CS knowledge and skills, and how this should impact the design of both machine learning courses and the broader CS university curriculum.

Thinking Like a Scientist, Not a Mathematician

Computing educators[1,6] have historically considered the core of CS to be a collection of human-comprehensible abstractions in the form of data structures and algorithms. Deterministic and logically verifiable algorithms have been central to the epistemology and practices of computer science.

With machine learning (ML) this changes: First, the typical model is likely to be an opaque composite of millions of parameters, not a human-readable algorithm. Second, the verification process is not a logical proof of correctness, but rather a statistical demonstration of effectiveness. As Langley[5] observed, ML is an empirical science that shares epistemological approaches with fields such as physics and chemistry.

While traditional software is built by human programmers who describe the steps needed to accomplish a goal (how to do it), a typical ML system is built by describing the objective that the system is trying to maximize (what to achieve). The learning procedure then uses a dataset of examples to determine the model that achieves this maximization. The trained model takes on the role of both data structure

and algorithm. The role that each parameter plays is not clear to a human, and these computational solutions no longer reflect humans' conceptual descriptions of problem domains, but instead function as summaries of the data that are understandable only in terms of their empirically measurable performance.

To succeed with ML, many students will not concentrate on algorithm development, but rather on data collection, data cleaning, model choice, and statistical testing.

ML Education within CS Education

ML has historically been a niche area of CS, but now it is increasingly relevant to core CS disciplines, from computer architecture to operating systems.[3] It may even be fair to say that ML is now a core area of CS, providing a parallel theoretical basis to the lambda calculus for defining and reasoning about computational systems. The growing importance of ML thus raises challenging questions for CS education: How should practical and theoretical ML topics now be integrated into undergraduate curricula? And how can we make room for expanded ML content in a way that augments—rather than displaces—classical CS skills, within undergraduate degree programs whose duration must remain fairly static?

Changes to the Introductory Sequence. Most CS undergraduate programs begin with introductory courses that emphasize the development of programming skills, covering topics like control structures, the definition and use of functions, basic data types, and the design and implementation of simple algorithms.[4]

In many cases, assignments in these courses make use of existing library functions, for instance to read and write data to the filesystem. Students are not expected to fully understand how these libraries and the underlying hardware infrastructure work, so much as to use the interfaces that these libraries present. The aims of introductory courses are students' development of notional machines[2] for reasoning about how a computer executes a program, and the development of the pragmatic skills for writing and debugging programs that computers can execute.

ML has historically been a niche area of CS, but now it is increasingly relevant to core CS disciplines.

These same two aims can also describe introductory courses for an ML-as-core world. We do not envision that ML methods would replace symbolic programming in such courses, but they would provide alternative means for defining and debugging the behaviors of functions within students' programs. Students will learn early on about two kinds of notional machine—that of the classical logical computer and that of the statistical model. They will learn methods for authoring, testing, and debugging programs for each kind of notional machine, and learn to combine both models within software systems.

We imagine that future introductory courses will include ML through the use of beginner-friendly program editors, libraries, and assignments that encourage students to define some functions using ML, and then to integrate those functions within programs that are authored using more traditional methods. For instance, students might take a game they created in a prior assignment using classical programming, and then use ML techniques to create a gestural interface (for example, using accelerometers from a smartphone, pose information from a webcam, or audio from a microphone) for moving the player's character up, down, left, and right within that game. Such assignments would engage students in creating or curating training examples, measuring how well their trained models perform, and debugging models by adjusting training data or choices about learning algorithms and features.

Such activities do not require deep understanding of ML algorithms, just as reading from a filesystem using high-level APIs does not require deep understanding of computer hardware or operating systems. Yet these activities can introduce new CS students to epistemological practices core to ML, laying the foundation for encountering ML again in other contexts (whether an elective in ML theory, advanced electives in computer vision or architecture, or in professional software development). Such activities additionally enable the creation of new and engaging types of software (for example, systems that are driven by real-time sensors or social-media data) that are very difficult for novice programmers (and even experts) to create without ML.

Changes to the Advanced Core. In most CS degree programs, the introductory sequence is followed by a set of more advanced courses. How should that more advanced core change in light of ML?

Current courses in software verification and validation stress two points: proof of correctness and tests that verify Boolean properties of programs. But with ML applications, the emphasis is on experiment design and on statistical inference about the results of experiments. Future coursework should include data-driven software testing methodologies, such as the development of test suites that evaluate whether software tools perform acceptably when trained using specific datasets, and that can monitor measurable regressions over time.

Human-computer interaction (HCI) courses may be expanded to reflect how ML changes both the nature of human-facing technologies that can be created and the processes by which they are created and evaluated. For instance, ML enables the creation of applications that dynamically adapt in response to data about their use. HCI education currently emphasizes the use of empirical methods from psychology and anthropology to understand users' needs and evaluate new technologies; now, the ability to apply ML to log data capturing users' interactions with a product can drive new ways of understanding users' experiences and translating these into design recommendations. Future HCI coursework will need to include these ML-based systems design and evaluation methodologies.

Operating systems courses describe best practices for tasks such as allocating memory and scheduling processes. Typically, the values of key parameters for those tasks are chosen through experience. But with ML the parameter values, and sometimes the whole approach, can be allowed to vary depending on the tasks that are actually running, enabling systems that are more efficient and more adaptable to changing work loads, even ones not foreseen by their designer. Future OS coursework may need to include the study of ML techniques for dynamically optimizing system performance.[3]

Changes to Prerequisite and Concurrent Expectations. It is typical for CS curricula to require coursework outside of CS departments, such as courses in mathematics and physics. In many cases, and especially when CS programs are housed within schools of engineering, these requirements emphasize calculus coursework. Many programs include coursework in probability and statistics, though notably the authors of ACM and IEEE's joint Computing Curricula 2013 "believe it is not necessary for all CS programs to require a full course in probability theory for all majors."[4]

Are these recommendations still appropriate? Many programs require coursework in probability and statistics, which we enthusiastically encourage, as they are crucial for engaging with the theory behind ML algorithm design and analysis, and for working effectively with certain powerful types of ML approaches. Linear algebra is essential for both ML practitioners and researchers, as is knowledge about optimization. The set of foundational knowledge for ML is thus both broad and distinct from that conventionally required to obtain a CS degree. What, therefore, should be considered essential to the training of tomorrow's computer scientists?

Conclusion

The ACM-IEEE Computer Science Curricula 2013[4] identifies 18 different Knowledge Areas (KAs), including Algorithms and Complexity, Architecture and Organization, Discrete Structures, and Intelligent Systems. The definitions and recommended durations of attention to the KAs reflect a classic view of CS; ML is referred to exclusively within a few suggested elective offerings. We believe the rapid rise in the use of ML within CS in just the past few years indicates the need to rethink guiding documents like this, along with commensurate changes in the educational offerings of computing departments.

In addition, research on how people learn ML is desperately needed. Nearly the entirety of the published computing education literature pertains to classical approaches to computing. As we have mentioned earlier in this column, ML systems are fundamentally different than traditional data structures and algorithms, and must, therefore, be reasoned about and learned differently. Many insights from mathematics and statistics education research are likely to be relevant to machine learning education research, but researchers in these fields only rarely intersect with computing education researchers. Therefore, we call upon funding agencies and professional societies such as ACM to use their convening power to bring together computing education researchers and math education researchers in support of developing a rich knowledge base about the teaching and learning of machine learning.

References

1. Aho, A.V. Computation and computational thinking. *The Computer Journal 55*, 7 (July 2012), 832–835.
2. Boulay, B.D., O'Shea, T., and Monk, J. The black box inside the glass box: Presenting computing concepts to novices. *International Journal of Man-Machine Studies 14*, 3 (Apr. 1981), 237–249; https://doi.org/10.1016/S0020-7373(81)80056-9.
3. Dean, J., Patterson, D., and Young, C. A new golden age in computer architecture: Empowering the machine-learning revolution. *IEEE Micro 38*, 2 (Mar./Apr. 2018), 21–29.
4. Joint Task Force on Computing Curricula, Association for Computing Machinery, IEEE Computer Society (2013). Computer science curricula 2013; https://bit.ly/2E6dDGR
5. Langley, P. Machine learning as an experimental science. *Machine Learning 3*, 1 (Jan. 1998), 5–8.
6. Wing, J.M. Computational thinking. *Commun. ACM 49*, 3 (Mar. 2006), 33–35.

R. Benjamin Shapiro (ben.shapiro@colorado.edu) is an assistant professor in the ATLAS Institute, the Department of Computer Science, and (by courtesy) the School of Education and the Department of Information Science at the University of Colorado, Boulder, USA.

Rebecca Fiebrink (r.fiebrink@gold.ac.uk) is a senior lecturer in the Department of Computing at Goldsmiths, University of London.

Peter Norvig (pnorvig@google.com) is Director of Research at Google, Inc.