How Machine Learning Improves Algorithms with Ellen Vitercik

Sanjay Dasgupta: Well, hello, everybody. Welcome to another episode of Science Like Me Plus. I am Sanjay Dasgupta. I'm a professor here in the computer science department at UC San Diego. Today, I'm very pleased to have our guest, Ellen Vitercik. She is an assistant professor at Stanford University. She has a joint appointment in computer science and also in management science and engineering. Ellen got her PhD in 2021. She has won a whole bunch of awards at that time, and since then, she got multiple awards for her dissertation. She won the Miller Fellowship at UC Berkeley. She won the AI 2050 Fellows program, and she also has an NSF Career Award. All of this is very impressive, and we are extremely lucky and pleased to have her with us today. Ellen, I was hoping that we could get a little bit of a sense of some of the things that you work on, and also your view of the field in general because these are times of great change. Perhaps a good place to start would be on something that seems to be a really central theme in algorithms these days, namely beyond worst-case complexity. It's what is that?

How does it relate to your stuff?

Ellen Vitercik: Sounds good. Thank you for the introduction. I work on using machine learning to improve algorithm design and improve the time it takes us to solve really hard optimization problems, namely, NP-hard optimization problems. If we think about the way that we learn algorithms in undergrad, we learn algorithms in the framework of worst-case analysis, where we're trying to understand, well, what problems are polynomial time solvable in the worst-case and what problems are not polynomial-time solvable in the worst-case. What is the runtime of those algorithms in this worst-case analysis framework? I got interested in this area of beyond worst-case analysis in my PhD because I was curious about other ways of thinking about how we design algorithms and yeah, just other ways of thinking about computability. In large part, this was because if we think about a lot of the hard optimization problems that we actually encounter in practice. These often are NP-hard problems, but somehow that doesn't scare practitioners away from trying to solve them.

There's just this gap between the way I was taught algorithms, especially as a more like mathematical, theory-oriented student and the way that people seem to think about algorithmic problem solving in practice. I became really interested in this idea of beyond worst-case analysis. If there's other ways we can think about how hard a problem actually is to solve. The type of research I've done from this perspective is understanding, if we're trying to solve a really hard problem day after day after day, the shipping problems that, a shipping company has to solve day after day, those routing problems, is there some structure shared across the problem instances that are coming up in practice that could somehow allow us if we can uncover the structure and leverage the structure, solve these problems faster?

Sanjay Dasgupta: I see. I mean, when I was in grad school, I think the very pessimistic view of algorithms was still in full force, and it really was a very damning picture. It really seemed like just about everything we want to solve is NP-complete, which means that either you can solve all of them or you can solve none of them. Somehow people seem to overwhelmingly believe that actually you can solve none of them. It's nice that now, the news seems somewhat better, that actually these pessimistic results were only if you look at the very worst possible case, and that in practice, the picture is somewhat rosier. But I feel like we're still teaching the bad news to undergrads. Is that just a UCSD thing or is that also a Stanford phenomenon?

Ellen Vitercik: Well, I think that this worst-case framing is very important, and certainly the way that we're still teaching algorithms at Stanford, as well as through this worst-case framing. But I

did when I taught algorithms this past quarter, I did try to, hint at this beyond worst-case framing for perhaps some more practically inclined students who do have this question mark in their head as they're learning about worst-case analysis. We also have at Stanford, a class that was designed by Tim Roughgarden who's one of the, early people in this space on beyond worst-case analysis. I taught that. That was more of a liked sorry, upper level undergrad and PhD level class. But still, I think the curriculum there is still being developed. I guess the interesting thing about this area is that there's, new research all the time that could be developed from a curriculum perspective. It's an ever evolving area.

Sanjay Dasgupta: I feel like we should be doing a lot more of that here. Thanks for saying that. It definitely inspires me to create a class of this kind. One of the strange things about the theory of NP-completeness is that you have this huge list of problems, like tens of thousands of problems, and they all sound really different. There's Boolean satisfiability, the traveling salesman problem, scheduling problems, and yet they all turn out to be equivalent to one another when it comes to the issue of polynomial time solvability. That's bizarre because they seem so different. Do you feel that now that we're thinking more about beyond worst-case is that these problems regain some of their individuality, that we are now not just clumping them all together any longer.

Ellen Vitercik: Yeah, that's an interesting question. I think at least from my perspective, the way that I come at beyond worst-case analysis is really from a more, mathematical programming, perspective. I'm actually very passionate about the ways that we base, broad ways that we can represent NP-hard problems like integer programming or, satisfiability problems and trying to develop the solvers that do solve these very broad and broadly encompassed types of formulations. But at the same time, something that I'm really excited about in this particular space is these solvers for, say, like integer programming solvers or SAT solvers. They have been developed very generically. When you give a problem to say an integer programming solver like Gurobi, it should be able to solve the average integer program, pretty quickly. But I think with the rise of Gen AI and LLMs, there's a lot more scope We've already seen a lot of examples of this to develop using machine learning, a lot of amazing, very problem-specific heuristics.

If you're trying to solve your specific type of vehicle routing problem or your specific type of medical resource allocation problem, I think there's becoming more and more ways to design algorithms and heuristics that are very well designed for the particular problems that you're trying to solve. There could be scope to take, a broad framework like integer programming or like a Gurobi optimization solver and integrate these very problem-specific heuristics into this broader framework. I think, LLMs have really opened up this space to both take advantage of, the broad frameworks like integer programming and Gurobi, but also take advantage of, the very problem-specific heuristics that you can possibly develop with the aid of machine learning.

Sanjay Dasgupta: I see. There's something like integer programming that can apply to tons of different problems and is applied to tons of different problems, but in the worst-case, takes a long time. Then there are heuristics for speeding it up. These heuristics do not impact the correctness of the procedure. It gives you the right answer at the end.

Ellen Vitercik: That itself, that's a really important question to ask in this space. It is really important that when we use these tools like integer programming solvers or SAT solvers, that we are maintaining some guarantee of optimality. That's why we use them. But I guess a very interesting research problem is that when I use machine learning or LLMs and try to improve these optimization solvers, What is the right way to be leveraging these tools in such a way that we don't risk losing that very important formal optimality guarantee?

Sanjay Dasgupta: I see. Somehow ML is used to select heuristics for improving the performance of these solvers.

Ellen Vitercik: Yeah, exactly.

Sanjay Dasgupta: Has that worked out pretty well so far?

Ellen Vitercik: I think it's definitely still an evolving space. Already, so for example, with, AlphaEvolve or these types of pipelines coming out of DeepMind. We've seen a lot of interesting problem-specific heuristics developed that seem to have a pretty impressive performance. But I think there's still a gap between these very problem-specific heuristics, which you don't have any, proof of optimality, and then these solvers that where the proof of optimality is like a central importance and understanding the landscape between them is something I'm really interested in.

Sanjay Dasgupta: I see. Now, I know that you also have a lot of work on LLM reasoning because these language models seem to have some reasoning ability, and they certainly seem to express opinions with confidence and then provide some justification. But I know you've been trying to get to the bottom of how sound this reasoning actually is. If you could say a little bit about your work in that.

Ellen Vitercik: We've been interested in an algorithmic reasoning as we and other labs have been interested in algorithmic reasoning as a way to understand LLM reasoning because with algorithmic reasoning, well, oftentimes, the question and the answer. Say the question is, here's a graph? What's the shortest path between A and B, or here's an array? What would be a sorted version of the array? It's extremely verifiable, the final answer. We can programmatically understand what the final ground-truth answer ought to be. But when we ask a model one of these questions, we don't know: How is it solving this problem? Does it look anything like how we as humans solve the problem, or is it doing something completely different? What's cool about algorithmic reasoning is that we have a lot of well-developed hypotheses of if we were a human approaching, say, a shortest path reasoning problem, how would we approach it? We might approach it using Dijkstra's algorithm or using the Bellman-Ford algorithm.

With these well-formed and verifiable hypotheses, we can then test, is the LLM doing something like Bellman-Ford? Is the LLM doing something like Dijkstra's. The specific angle from which we've been looking at this problem is, well, in ICML this summer, we have a paper on data structure reasoning. Looking at data structure tasks as a very well structured and very easily verifiable way to test like reasoning primitives, I would say, linear reasoning, hierarchical reasoning, and things like that. For example, we can ask things, here's an array. If you were to add this element to the array and maintain the state that the array must be sorted. What answer do you get? Very simple, linear reasoning skills. Then you can start to probe, compositional reasoning ability. If I add multiple elements to this array, if I delete some elements, what state do we end up with? Also more complex, hierarchical forms of reasoning, which can be represented via trees and graphs and things like that. My favorite aspect of this work is also understanding very out-of-distribution reasoning abilities.

For example, if you ask, an LLM, a data structures task on a very nice, uniform problem instance, the LLM will also often get the correct answer because that's what the data structures task would look like in a textbook. But if you ask the LLM to do this data structures reasoning on a very non-uniform, very like a structured instance, oftentimes the reasoning will somehow break down. Our hypothesis is that non-uniform instances are very different from the types of things that you see in textbooks. It displays this out-of-distribution lack of generalization.

Sanjay Dasgupta: But then it sounds like it's really not learning the definition of the data structure at all.

Ellen Vitercik: It's somehow it's doing some amount of pattern matching. It's doing some amount of shortcutting. I guess what I've liked about this is that it allows us to very precisely like

diagnose reasoning breakdown.

Sanjay Dasgupta: These are tasks that, would say, a human five-year-old be able to do these pretty easily. Here's a sorted list of numbers. Insert the Number 7 into the list.

Ellen Vitercik: I think so. Some of them might be a little more difficult than what a five-year-old could do. For example, you could imagine building what we learn in intro algorithms or intro data structures class would be like a binary search tree, a balanced binary search tree to maintain like a sorted list of integers or something like that. Then in, say, graphics, you start to work not with 1D data, but 2D data, 3D data, and then we have beyond binary trees, we have 2D trees, 3D trees, K-D trees. These might be more difficult than what, say a five-year-old could do, but definitely an eight-year-old or something. If you taught them the definition of, say, a 2D tree, the generalization of a binary tree, they ought to be able to do it. But what we find is that the LLMs, when you give them data that's not uniformly distributed in a 2D plane, but is somehow like concentric circles or moon shapes or things like that, somehow it's not able to build these 2D trees very well.

Sanjay Dasgupta: Well, that's definitely very disturbing. I see. Well, it's great that you are developing these benchmarks that can be used to test the reasoning skills of these machines in a sensible way where the instances actually test the underlying concepts rather than just pattern matching or just coming from some very easy distribution.

Ellen Vitercik: Exactly.

Sanjay Dasgupta: Assuming that these benchmarks are put out there and that perhaps companies are able to train models to be able to solve these questions, suppose they are able to actually get them to do what these basic data structures do, maybe that would make it easier for them in the future to actually learn better algorithms.

Ellen Vitercik: We view it as like a first step or, a very diagnostic step towards understanding, more general reasoning capabilities of models because when we ask a model to do some, complex reasoning task for us, it does have to do a lot of these, learn primitives and then compose primitive reasoning skills into longer, more complex chains of reasoning. We view this as like a diagnostic way of understanding, what are the different types of primitives when it comes to, reasoning and in particular algorithmic reasoning? Where do these break down? One could hope that if the models get better at this type of compositional reasoning, that they'd be better able to help us do, complex tasks, like planning and routing and things like that.

Sanjay Dasgupta: That sounds like a really beautiful and impactful line of work.

Ellen Vitercik: Thank you.

Sanjay Dasgupta: Congratulations on doing this stuff. Actually, I thought that maybe we could, find out a little bit about your background. For example, at what stage did you decide that you wanted to be an academic in this mathematically very sophisticated field? When did you decide to get into this racket?

Ellen Vitercik: Sure, my dad is retired, but he was a professor of music history, actually at Middlebury College in Vermont. I thought that I was going to be a professional violinist till I was 17, and then I decided I wanted to do academics instead. Really, it was I feel like my dad had been telling me my whole life that being a professor is the best job. [Laughter] in the world to have.

Sanjay Dasgupta: It really is.

Ellen Vitercik: Once I decided I wanted to do academics, I felt I had no choice but to become a professor in whatever field I decided to go into. When I was in maybe, sophomore year, well, my mom was a high school math teacher, so I always loved math. I continued taking math in college. I was afraid to do proof-based math before I finally started doing proof-based math, and then I discovered I loved writing proofs. Then finally, in my sophomore year, I started taking computer science classes and loved computer science, continued to love math. I majored in math and minored in computer science. But what really made me decide to go into, more theoretical computer science was I studied abroad in Budapest in my junior year, and the Hungarian math tradition, is extremely strong in combinatorial optimization and combinatorics and, this theoretical computer science. That is what made me decide to do more, theoretical computer science in my PhD.

Sanjay Dasgupta: I see. It sounds like your work actually is across several communities. There's theoretical computer science, there is machine learning. There's maybe operations research and so on. Have you found them very welcoming as a whole, or is there one that you really like the best?

Ellen Vitercik: Yeah, I really went into a PhD thinking I would just do theoretical computer science, and that was it. But I got into, more theoretical machine learning in my PhD. Also, so that was my one PhD advisor, Nina Balcan, who's very central in the theoretical machine learning community. Then my other PhD advisor, Thomas Sandholm is, the direction that we ended up pursuing was much more oriented towards, integer programming and operations research. I in particular, started to think about more beyond worst-case analysis applied to integer programming and math optimization. That's how I got into operations research as well. I think, let's see. Of the, most welcoming communities, I would say that I've been a part of, actually, let's say a quarter of my research has been in the more economics and computer science community. That community, I've found to be, incredibly welcoming and incredibly warm community. I love the community in part because all my friends are in that community.

I continue wanting, even though my research, I feel like has slowly drifted away from Econ CS, I still want to publish in that community just because I love going to EC and spending time with my friends in that community so much. I think that really speaks to the value of having like a warm and welcoming community and that encouraging people to want to be a part of it and contribute to that community.

Sanjay Dasgupta: I see. Is the community a little bit smaller than CS theory?

Ellen Vitercik: Yeah, perhaps a little bit smaller. Towards that end, actually, and you've helped us with this. Some friends of mine and I have been working on developing similar mentorship and community building activities for the theoretical machine learning community as well. Trying to do mentorship events, how to talks, all these things to try to get, young students to feel welcome and included also in, other communities, too.

Sanjay Dasgupta: Absolutely. Well, obviously, times are, changing very rapidly. In terms of where academia is now positioned, do you feel that it's a great time to be an academic or was this a time where a lot of stuff is actually happening in industry, and maybe that's the place to be. It's hard to tell. I mean, I wouldn't dream of leaving this, because, there's so much that's good, about the setting. But how do you feel about this?

Ellen Vitercik: Definitely, it's an incredible time to be in industry. But for me in particular, I think I've always felt very just incredibly lucky since, the first day of my PhD, to be able to spend all day every day, thinking about things that I think are super interesting. I always say that, my favorite state of being is the state of being, intrigued. [Laughter] That's my favorite state of being. For me personally and my own personality type, I think that being in academia gives me the opportunity to just be very curiosity driven and try to, pursue that state of always being

intrigued. I have never done anything other than being in academia, really. It could be like a bias, but I personally feel like academia is the best place for me.

Sanjay Dasgupta: No, I feel exactly the same way. Just the ability to think about stuff all day long. Sometimes it doesn't even look like work to other people. I'll be lying on the couch at home thinking about stuff, and my family doesn't believe that I'm actually working. They think I'm taking a nap or something like that. They just don't understand. Ellen, thank you very much for sharing your perspective and your experiences. It has been an absolute delight chatting with you.

Ellen Vitercik: Thank you. Thank you so much. It's been a pleasure.