JOSHUA TENENBAUM: So we saw in Laura's talk this introduction to both this idea of the child as scientist, all the different ways that children's learning seems to follow the many different practices that scientists use to learn about the world, not just the data analysis, in a sense. As much as various kinds of statistics on a grand scale, whether it's Hebbian learning or backprop important for learning, we know that that's not all that children do, just like analyzing patterns of data is not all that scientists do. And then Laura added this other cool dimension, thinking about the costs and rewards of information, when is it worth it or not.

And you could say, maybe she was suggesting that we should develop the metaphor, expand it a little bit from child as scientist to maybe like the child as-- oh, oops, sorry-- lab PI or maybe even NSF center director. Because as everyone knows, but Tommy certainly can tell you, whether you're a lab PI who's just gotten tenure or the director of an NSF Center, you have to make hard-nosed pragmatic decisions about what is achievable given the costs and which research questions are really worth going after and devoting your time and other resources to. And that's a very important part of science. And it's an important part of intuitive knowledge.

I want to add another practical dimension to things, a way of bringing out, fleshing out this idea of the child as scientist. You can think about all these, these are metaphors. But they're things that we can formalize.

And if our goal is to make computational models of and ultimately to get some kind of theoretical handle on, then these are helping us. By adding in the costs and benefits, you bring in utility calculus. And there's not just naive utility calculus, there's a formal mathematical utility calculus of these kinds of decisions. Julian Jara-Ettinger, who Laura mentioned, was driving a lot of the work towards the end of the talk, has done some really interesting actual mathematical computational models of these issues, as has [? Choi ?] and the other students you talked about.

So the direction I want to push here is what you might call the child as hacker. This is trying to make the connection back to this idea of formalizing common sense knowledge and intuitive theories as probabilistic programs. Or just more generally, the idea of a program-- a structured algorithm and a dat-- or some combination of algorithms, data structures, networks of functions that can scribe interesting causal processes in the world, like, for example your intuitive physics or your intuitive psychology. That's an idea that we talked a lot about last week or whenever it was, earlier in the week or last week.

And this idea that if your knowledge is something like a program, or a set of programs, that you can, say, for example, run forward to simulate physics, like we had last time, then learning has to be something like building a program or hacking. And I think you could make-- this is, again, a research program that I wish I had. Like Laura talked to the end about the research she wished she had.

It's not really just a wish for her. She actually is working towards that research program. And this isn't just an empty wish either. It's something that we're working on, which is to try to take-- just as Laura had that wonderful list of all the things scientists do in their practices to learn about the world, I think you could make a similar list of all the things that you do in your programming or hacking.

By hacking I don't mean like breaking into a secure system, but modifying your code to make it more awesome. And I use awesome very deliberately, because awesome is a multi-dimensional term. It's just awesome.

But it could be faster, more accurate, more efficient, more elegant, more generalizable, more easily communicated to other people, more easily modularly combined with other code to do something even more awesome. I think there's a deep sense in which that aesthetic behind hacking and making awesome code, in both an individual and a social setting, that's a really powerful way to think about many of the cognitive activities behind learning. And it goes together with the idea of the child as scientist if the form of your, quote, "intuitive science" are computer programs or something like programs.

So we've been working on a few projects where we've been trying to capture this idea and to say, well, what would it mean to describe computationally this idea of learning as either synthesizing programs, or modifying programs, or making more awesome programs in your mind. And I'll just show you a few examples of this. I'll show you our good, our successful case studies, places where we've made this idea work.

But the bottom line to foreshadow it is that this is really, really hard. And to get it to work for the kinds of knowledge that, say, Laura was talking about or that Liz was talking about, the real stuff of children's knowledge, is still very, very open. And we want to, basically, build up to engaging why that problem is so hard. From this to what Tomer and then Laura will talk about later on in the afternoon.

But here's a few at least early success stories that we've worked on. One goes back to this idea that I presented in my lectures last week. And it connects to, again, something that Laura was saying.

Here, a very basic kind of learning. It's just the problem of learning some generalizable concepts at all from very sparse evidence. Like one-shot learning-- again, something you heard from Tommy, a number of the other speakers.

We've all been trying to wrap our heads around this. How can you learn, say, any concept at all from very, very little data, maybe just one or a few examples. So you saw this kind of thing last time.

And I briefly mentioned how we had tried to capture this problem as something like this by building this tree structured hypothesis space. And you could think of this as a kind of program induction. If you think that there's something like an evolutionary program which generated these objects, and you're trying to find the sub procedure of it that generated just these kinds of objects.

But that's not at all how we were able to model this. We had a much simpler model. But let me show you briefly some work that we did in our group a couple of years ago. It's really just getting out into publication now.

This is work that was mostly done by two people-- Ruslan Salakhutdinov, who is now a professor at Toronto, although about to move to Carnegie Mellon, I think, and Brenden Lake. He's a machine learning person, also very well known for deep learning. And then Brenden Lake-- this is really mostly what I'll talk about is Brenden Lake's work, who is now a post-doc at NYU.

And again, where we think we're building up to is trying to learn something like the program of an intuitive physics or intuitive psychology. But here we're just talking about learning object concepts. And we've been doing this work with a data set of handwritten characters, the ones you see on the right here.

I'll just put it up in contrast or by comparison to, say, this other much more famous data set of handwritten characters, the MNIST data set. How many people have seen the MNIST data set, maybe in some of the previous talks? How many people have actually used it?

Yeah, it's a great data set to use. It's driven a lot of basic machine learning research, including deep learning. Yann LeCun originally collected this data set and put this out there. And Geoffrey Hinton did most of the development.

The stuff that now wins object recognition challenges was done on this data set. But not only that. Also a lot of Bayesian stuff and probabilistic generative models. Now, the thing about that data set, though, is it has a very small number of classes, just the digits 0 through 9, and a huge number of examples, roughly 10,000 examples in each class or maybe 6,000 examples, something like that.

But we wanted to construct a data set which was similar in some ways in its complexity and scale, but where we had many, many more concepts and, perhaps, many fewer examples. So here we got people to write by hand characters in 50 different alphabets. And it's a really cool data set.

So that total data set has 1,623 concepts drawn. You could call them handwritten characters. You could just call them simple visual concepts as a sort of a warm up for bigger problems of, say, natural objects.

And there's 20 examples per class. So there's roughly 30,000 total data points in this data set, very much like MNIST. You can see, just to illustrate here, there's many different alphabets that have very different forms. You can see both the similarities and differences between alphabets here.

So in that sense, there's kind of a hierarchical structure. Each one of these is a character in an alphabet. But there's also the higher level concept of a sort of a Sanskrit form, as distinct from, say, to Gaelic, or Hebrew, or Braille. There's some made-up alphabet.

But one of the neat things about this domain is that you can make up new concepts, and you can make up whole concepts of concepts, like whole new alphabets. You can do one-shot learning in it. So let's just try this out here for a second.

You remember the tufa demo. We can do the same kind of thing here. Like let's take these characters. Anybody know the alphabet that this is? OK, that's good.

Most of you have not seen these before. That's good that you know. But we'll do this experiment run on the rest of you.

So here's one example of a concept. Call it a tufa if you like. And I'll just run my mouse over these other ones. And you just clap when I get to the other example of the same class, OK?

[SOUND OF CLAPS]

OK, very good. Yeah, people are, basically, perfect at this. It doesn't take-- I mean, again, it's very fast and almost perfect.

And again, you saw me talk a little about this last time. Just like with natural objects, not only can you learn one of these concepts from one example and generalize it to others, but you can use that knowledge in various other ways. So you can parse these things into parts. We think that's part of what you're doing.

You can generate new examples. So here are three different people all drawing the same character. And in fact, the whole data set was generated that way.

You can also make higher level generalizations, recombining the parts into totally new concepts, the way there's that weird kind of like unicycle thing over there, unimotorcycle. Here, I can show you, as you'll see 10 characters in a new alphabet, and you can make up hypothetical, if, perhaps, incorrect examples in it. Again, I'm just going to show you a couple of case studies of where this idea of learning as program synthesis might work.

So the idea here is that, as you might see, these are three characters down on the bottom. And this is just a very schematic diagram of how our model tries to represent these as simple kinds of programs. Think about how you would draw, say, that character down to the bottom. Just try to draw it in midair. How would you draw that one in the lower left there?

Are many of you doing something like this? Is that what you are doing? OK, yeah. So basically, everyone does that.

And you can describe that as sort of having two large parts or two strokes, where you pick up your pen between strokes. And one of the strokes has two sub strokes, where you stop your pen. And there's a consistent relationship. The second stroke has to begin somewhere on a particular general region of the first stroke.

And basically, that's the model's representation of concepts-- part, subparts, and simple relations-- which, you can see, it might scale up, arguably, to more interesting kinds of natural objects. And the basic idea is that you represent that, though, as a program. It's a generative program.

It's kind of like a motor program. But it's more abstract. We think that when you see these characters and many other concepts, you represent something about how you might create it.

But it doesn't mean it's in your muscles. You could use other hands. You could use your toe.

Or you could even just think about it in your imagination. So the model, basically, tries to induce these simple-- think about them as maybe simple hierarchical plans, simple action programs. And it does it by having a program generating program that can itself have parameters that can be learned from data.

So this right here, this is a program called GenerateType. And what that does is it's a program-- a type means a character concept, like each of those three things is a different type. This is a program which generates a program that generates the actual character.

The second level of program is called GenerateToken. That's a program which draws a particular instance of a character. And just like you can draw many examples of any concept, you can call that function many times-- GenerateToken, GenerateToken, GenerateToken.

So your concept of a character is a generative function. And in order to learn this, you have, basically, a prior on those programs that comes from a program generating program. That's the GenerateType program.

So there's a lot of details behind how this works. But basically, the model does a kind of learning to learn up from a held out unsupervised set and learns the parameters of this program generating program, which would characterize how we draw things in general, what characters look like in general. And then, when you see a new character, like this one, effectively, what the model is doing is it's both parsing this into its parts, and subparts, and relations. But that parsing is, basically, the program synthesis.

It is pretty much the same thing. You're constructing, you're looking at that output of some program and saying, what would be the best simple set of parts, and subparts, and relations that could draw that? And then I'm going to infer the most likely one, and then use that as a generalizable template or program I can then generate other characters with.

So here, maybe to just illustrate really concretely, if you were to see this character here-- well, here's one instance of one class. Here's an instance of another class. Again, I have no idea which alphabet this is.

Now, what about this one? Is it class 1 or class 2? What do you think? 1, yeah. Anybody think it's class 2? OK.

So how do we know it's class 1? Well, at the pixel level, it doesn't look anything like that. So this is, again, an example of some of the issues that Tommy was talking about-- a really severe kind of invariance.

But it's not just translation or scale invariance, although it does have some of that. But it also has this kind of interesting within-class invariance. It's a rather different shape. It's been distorted somewhat.

For a program, there's a powerful way to capture that where you can say, well if you would do something like the program for generating this, which is like one stroke like that and then these other two things shown with the red and green, and here's a program that you might induce to generate that. And then the question is, which of these two programs, simple hierarchical motor programs, is most likely to generate that character? Now, it turns out that it's incredibly unlikely to generate any character from one of these programs.

These are the log scores, the log probabilities. So this one is like 2 to the negative 758. And this one is like 2 to the negative 1,880. I don't know if it's base e. It's maybe 2 or e, but whatever.

So each of these is very small. But this one is like 1,000 orders of magnitude more likely than that one. And that makes sense, right? It just is easier to think intuitively about generating this shape from that distortion.

So that's basically what the system does. And it's able to do this remarkable thing that you were able to do too-- this one-shot learning of a concept. Here's just another illustration of this.

We show people one example of a new character in an alphabet they don't know and ask them to pick out the other one. Everybody see where it is here? It's not that easy, but it's doable. Down here, right.

So people are better than 95% correct at this. This is the error rate. So the error rate is less than 5% for humans and also for this model.

But for a range of more standard deep learning models, this one here is, basically, like an image net or MNIST type 1. So this is the kind of model that was really sort of massive convolutional classifier. The best deep learning one is actually something for this problem.

It's what's called a Siamese ConvNet. And that can do somewhat better. But it's still more than twice as bad as people. So we think this is one place where, at least in a hard classification problem, you can see that deep learning still isn't quite there.

Whereas this-- even the best thing-- this was a network that was, basically, specifically worked out by one of Ruslan's students for about a year to solve exactly this problem on this data set. And it substantially improved over a standard deep learning classifier, which substantially improved over a different deep learning model that Ruslan and I both worked on. So there's definitely been some improvement here.

And never bet against deep learning. I can't guarantee that somebody sets, spends their PhD. They could work out something that could do this well. But still, it's a case which still has some room to push, where, for example, just a pure pattern recognition approach might go.

But maybe more interesting is, again, going back to all the things we use our knowledge for, kids might use our knowledge for. As we don't just classify the world. We understand it. We generate new things. We imagine new things.

So here's a place where you can use your generative program that none of these networks do, at least by nature. Maybe you could think of some way to get them to do it. And this is to say, not just classify, but to produce, imagine new examples.

So here's an illustration of this where we gave people an example of one of these new concepts. And then we said, draw another example of the same concept. Don't just copy it. Make up another example of the concept.

And what you can see here is a set of nine examples that nine different people did in response to that query. And then you can also see on the other side nine examples of our program doing that. Can anybody tell which is the people and which is the program? Let's try this out.

So which is the machine for this character, the left or the right? How many people say the left? Raise your hand. How many people say the right? About 50-50, very good.

How many people say this is the machine for this one? How many people say this is the machine? May be slight preference there. How many people say this is the machine? How many people say this is the machine? How many people say this is the machine?

Some people really like the left. How many people say that's the machine? Basically, it's 50/50 for all of them. Here's the right answer. I don't know, you could decide if you were right or not. I don't know,

Here's another set. Again, I hope it's clear that this is not an easy task. And in fact, people are, basically, at chance.

We've done a bunch of studies of this. And most people just can't tell. People on average are about 50% correct. You basically just can't tell.

So it's an example of a kind of Turing test that a certain interesting program learning program is solving. At the level that's confusable with humans, this system is able to learn simple programs for visual concepts. And not just classify but use them to create new things. You can even create new things at this higher level that I mentioned.

So here, the task, which, again, people and machines are roughly similar on, is to be given 10 examples each of a different concept in a higher level concept, like an alphabet, and then draw new characters in that alphabet. And we give people only a few seconds to do this. So they don't get too artistic.

But again, you can see that machine is able to do this. People are also kind of similar. So let me say, that was a success story, a place for the idea of learning as program induction kind of works. What about something more like what we're really most deeply interested in-- children's learning?

Like the ability, for example, to, say, understand goal-directed action. These cases we've talked a lot about. Or intuitive physics-- again, cases we've talked about.

And it's part of our research program for this center, something we'd love all of you guys, if you're interested, to help work on. It's a very big problem. It's how do you characterize the knowledge that kids are learning over the first few years and the learning mechanisms that build it, which we'd like to think of in some similar way.

Like could we say there's some intuitive physics program and intuitive physics program learning programs that are building out knowledge for these kinds of problems? And we don't know how to do it. But again, here are some of the steps we've been starting to take.

So this is work that Tomer did as part of his PhD, and it's something that he's continuing to do with Liz and others as part of his post-doc. So we're showing people-- again, it's much like what you saw from me and from Laura. We're really interested in learning from sparse data. Because all the data is sparse in a sense.

But in the lab, you push things to the limit. So you study really sparse things, like one-shot learning of a visual concept. Or here, this is like we've been interested in what can you learn about the laws of physics from just watching that for five seconds. So we show people videos like this.

Think of this as like you're watching hockey pucks on an air hockey table. So it's like an overhead view of some things bouncing around. And you can see that they're kind of Newtonian in some sense.

They bounce off of each other. Looks like there's some inertia, inertial collisions. But you might notice that there's some other interesting things going on that are not just F equals m a, like other interesting kinds of forces. And I'll show you other ones. Tomer made a whole awesome set of these movies.

Hopefully, you've got some idea of what's going on there. Like interesting forces of attraction, and, repulsion, different kinds of things. So here, each of those can be described as a program.

And here's a program generating program, if you like. So the same kind of idea, just as in the handwritten character model I showed you. It's not like it's learning in a blank slate way from scratch. It knows about objects, parts, and subparts.

What it has to learn is, in this domain of handwritten characters, what are the parts and relations like. And then for the particular new thing you're learning, like this particular new concept, what are its particular parts and relations. So there's these several levels of learning where the big picture of objects and parts is not learned.

And then the specifics for this domain of handwritten characters, the idea of what strokes look like. That's learned from sort of a background set. And then your ability to do one-shot learning or learning from very sparse data of a new concept takes all that prior knowledge, some of which is wired in, some of which is previously learned, and brings it to bear to generate a new program very sparsely.

So you have the same kind of thing here. We were wiring in, in a sense, F equals m a, the most general laws of physics. And then we're also wiring in sort of the possibility that there could be kinds of things and forces that they exert on each other, as some kinds of things exert other kinds of forces on others. And that there could be latent properties, things like mass and friction.

And then what the model is trying to do is, basically, to learn about these particular properties. What's the mass of this kind of object? What's the friction of this kind of surface? Which objects exert which kind of forces on each other? Is there something like gravity blowing everything to the left, or the right, or down?

What this is showing here is the same kind of plots we saw from me last time. It's a plot of people versus model, based on a whole bunch of different conditions of the sort you saw. People are judging these different physical properties. And they're making greater judgments of how likely it is, basically, to have one of these properties or another. And there's the model on the x-axis, people on the y-axis.

And what you can see is a sort of OK decent fit. We characterize this experiment as a kind of a mixed success. I mean, it's sort of shocking people can learn anything at all. Like how much could you learn about the laws of physics from five seconds of observation?

Well, it's also kind of shocking that Newton could learn about the laws of physics by just looking at, you know, in the history of the universe, about five seconds or less worth of data that people had collected for the planets going around. So it is the nature of both science and intuitive theory building that you can get so much from so little. But people are not Newton here.

They're just using intuition. They're making quick responses. And they're OK. There's a correlation, but it's not perfect by any means.

One of the things that we're working on right now is looking at, say, what happens if you can, unlike, say, Newton, go in and actually intervene and push these planets around. Hopefully you'll do better. But stay tuned for that.

The basic thing here, though, is that people can learn something from this. But the way our model works is it's not very satisfying for us as a view of kind of program induction or program construction. Because we think it just knows too much or has, basically, all the form of the program.

And it's estimating some parameters. It's like one of the things you do as a hacker, as a coder, is you have your code and you tune some parameters. Or you try to decide if this function is the right one to use or that one. And this is doing that.

But nowhere is this like actually writing new code, in a sense. And that's just the really hard problem that I wanted to mostly leave you with and set up going what we're going to do for the rest of the afternoon. Like if you wanted to not just tune the parameters and figure out the strength or existence of different forces, but actually write the form of laws, how would you do this? What's the right hypothesis space?

So you'd need programs that don't just generate programs but actually write the code of them in a sense. And what's an effective algorithm for searching the space of these theories? It's very, very difficult.

I think, Tomer, are you going to show this figure at all? Yes, so mostly I'll leave this to Tomer. But there's a very striking contrast between the nice optimization landscapes for, say, neural networks or most any standard scalable machine learning algorithm, whether it's trained by gradient descent or convex optimization, and the kinds of landscapes for optimization and search that you have if you're trying to generate a space of programs.

If you want to see our early attempts to try to do something like learning the form of a program, look, for example, at stuff that Charles Kemp did. Part of his thesis that was published in PNAS a few years ago, where he tried to generate or, basically, have-- think of generative grammars for graphs. Think about the problem-- so Laura mentioned Darwin. How did Darwin figure out something about evolution without understanding any of the mechanisms? Or the more basic problem of figuring out that species should be generated by some kind of branching tree process versus other kinds.

Remember last time when I talked about various kinds of structured probabilistic models, tree structures, or spaces, or chains for threshold reasoning. So Charles did some really nice work, basically trying to use the idea of a program for generating graphical models, like there's a grammar that grows out graphs. And he showed how you could take data drawn from different domains, like, say, those data sets you saw before of animals and their properties. We spent an hour on that last time.

So Charles showed how you could induce not only a tree structure but the higher level fact that there is a tree structure. Namely, a rule that generates trees being the right abstract principle to, say, give you the structure of species in biology, whereas other rules would generate other kinds of structures. So for example, he took similar data matrices for how Supreme Court judges voted and was able to infer a left-right, liberal-conservative spectrum.

Or data from the proximities between cities and figure out a sort of cylinder, like latitude and longitude map of the world just from the distances between cities. Or take faces and figure out a low dimensional space as the right way to think about faces. So in some sense, this was really cool.

We were really excited. Oh, hey, we have a way to learn these simple programs which generate structures, which themselves generate the data. It's where that idea of hierarchical Bayes meets up with this idea of program induction or learning a program.

And it even captured-- OK, this is really the last slide I'll show. It even captured something which captured all of our imaginations. We use this phrase "the blessing of abstraction" to tie back into one more theme of Laura's, which is this idea that when kids are building up abstract concepts, there's a sense in which, unlike, say, a lot of maybe traditional machine learning methods or a lot of traditional ideas and philosophy about the origins of abstract knowledge, it's not like you just get the concrete stuff first and layer on the more abstract stuff.

There's a sense often, in children's learning as in science, in which the big picture comes in first. The abstract idea comes there, and then you fill in the details. So for example, Darwin figured out, in some sense, the big picture. He figured out the idea that there was some kind of branching process that generated species that was random. Not a nice perfect Linnean seven-layer hierarchy but some kind of random branching process. And he didn't know what the mechanisms were that gave rise to it.

And similarly, Newton figured out something about the law of gravitation and everything else in his laws, though he didn't know the mechanisms that gave rise to gravity. And he didn't even know g. He didn't even know the value of the gravitational constant. That couldn't be estimated for 100 years later. But somehow he was able to get the abstract form.

And these nice things that Charles Kemp did were also able to do that. So for example, from very little data, to figure out that animals should be generated by some kind of a tree structure, as opposed to, say, the simpler model of just a bunch of flat clusters. That model was able to figure that out, over here on the right, from just a small fraction of the data. And then with all the rest of the data, it was able to figure out the right tree in a sense.

And we called this "the blessing of abstraction," this idea that often, in these hierarchical program learning programs, you could get the high level idea before you got the lower level idea and then fill in the details. And I still think there's something fundamentally right about this idea of children's learning, both representationally and mechanistically. And that this dynamics of sometimes getting the big picture first and using that as a constraint to fill in the details is fundamentally right. But actually, understanding how this-- either algorithmically-- how to search the space of programs for anything that looks like an intuitive causal theory of physics and relate that to the dynamics of how children actually learn. That's the big open question that I will now hand over to our other speakers.