EPISODE 97

[INTRODUCTION]

[0:00:10.8] SC: Hello and welcome to another episode of TWiML Talk, the podcast where I interview interesting people doing interesting things in machine learning and artificial intelligence. I'm your host, Sam Charrington.

Last week, I spent some time at CES, the Consumer Electronics Show in Las Vegas exploring the vast sea of drones, cameras, paper-thin TVs, robots, laundry-folding closets and other smart devices. You name it, it was there. Of course, I was also able to sit down with some really interesting folks working on some pretty cool AI-enabled products.

Head on over to our YouTube channel to check out some behind-the-scenes footage from my interviews and other quick takes from the show. Be on the lookout for our AI and Consumer Electronics Series right here on the podcast coming soon.

The show you're about to hear is part of a series of shows recorded at the RE•WORK Deep Learning Summit in Montreal back in October. This was a great event. In fact, their next event the Deep Learning Summit San Francisco is right around the corner on January 25th and 26th, and will feature more leading researchers and technologists like the ones you'll hear on the show this week, including Ian Goodfellow of Google Brain and Daphne Koller of Calico Labs and more. Definitely check out and use the code TWIMLAI for 20% off of registration.

In this show, I speak with Greg Diamos, Senior Computer Systems Researcher at Baidu. Greg joined me before his talk at the Deep Learning Summit, where he spoke on the next generation of AI chips. Greg's talk focused on some of the work his team was involved in that accelerates deep learning training by using mixed 16-bit and 32-bit floating point arithmetic.

We cover a ton of interesting ground in this conversation. If you're interested in systems level thinking around scaling and accelerating deep learning, you're really going to like this one. Of course, if you like this one, you're also going to like TWiML Talk #14 with Greg's former colleague, Shubho Sengupta, which covers a bunch of related topics. If you haven't already listened to that one, I encourage you to check it out.

Now, on to the show.

[INTERVIEW]

[0:02:38.6] SC: Hey, everyone. I am here at the RE•WORK Deep Learning Conference in Montreal. I've got the pleasure to be seated across from Greg Diamos from Baidu. He's a Senior Researcher there. Greg, welcome to This Week in Machine Learning and AI.

[0:02:52.4] GD: Thanks for having me.

[0:02:54.2] SC: Awesome. Why don't we get started by having you tell us a little bit about your background and how you got interested in ML and AI.

[0:03:00.9] GD: Sure. Absolutely. My background is traditionally been in high-performance computing. I've been really excited about building really fast processors for important applications that enable new applications that people can use. It's actually a strange story how I got to AI. I used to be an AI skeptic.

[0:03:20.6] SC: Okay. That's always a good place to start.

[0:03:22.7] GD: Yeah. I always felt like AI would be valuable, but I just felt there is no way that simple algorithms like stochastic gradient descent could ever solve these complex, highly multi-dimensional optimization problems.

Then at one point, I remember sitting at NVIDIA Research and hearing a talk from Yann LeCun and just realizing, "Oh, I was wrong. I was totally wrong." Immediately after that, I joined Baidu Research. Andrew Ng was founding the Silicon Valley AI Lab and it seemed like a great opportunity to learn more about AI. Man, it's been such a crazy ride to get to this point.

[0:04:05.3] SC: I bet. What was your path to get to that point that you even had an opinion that involves stochastic gradient descent?

[0:04:12.1] GD: Sure. Well, let's see. I like building things that are useful for people. I feel computing in general has enabled many new capabilities, like the internet and like videos, so many things that we take for granted every day, but really makes our lives better.

I was always really passionate about making computers even better. It's this belief that although you might not know it going into it if you make a faster computer, or more efficient computer, someone will find a way of building something amazing on top of that.

I spend a lot of time looking at applications. What are the things that you can use computers to do? AI was always there. Just the feeling with AI has – for me before, deep learning was that, it would just be too hard that a lot of existing theory was stirring you down and had all of these really difficult challenges. I'd seen a lot of people, very smart people spend a lot of time trying to tackle these problems and not quite getting all the way there.

[0:05:17.4] SC: Do you recall what was it about Yann's talk that made the lightbulb go off and made you realize that stochastic gradient descent was the answer?

[0:05:30.5] GD: Sure. You can look at these hierarchical future representations in **[0:05:35.7]**. When people look at images, you can tell – the world is hierarchical. You can break a chair down into pieces; it has arms and legs, you can break those pieces down recursively. There is a lot of existing work that provides some evidence that vision algorithms will do similar things for recognition tasks.

That wasn't ever really a question and people were able to build by hand things like feature detectors and these hierarchical systems that worked reasonably well. They were just very difficult to build. The interesting thing about Yann's talk was that systems could do it automatically. I thought before that that you get stuck in this intractable optimization problems, where even if a solution exists it's one out of some enormously large number like 2 to the power of 10, to the power of 30, or like something amazingly big. You think about different sizes of numbers. Sometimes I think of the number of atoms in the universe as being a big number. This is far, far bigger than that.

[0:06:46.2] SC: You were thinking that that number represents what?

[0:06:50.7] GD: How many things you'd have to search through to find a good solution –

[0:06:54.0] SC: The search base for your – Okay. Yeah, it's like the needle in an enormous haystack. More atom, just [inaudible 0:06:58.9] the universe. How could you ever possibly hope to search through it efficiently? Especially with these very simple algorithms, but reliably I've seen since then for one application after another for image recognition, for speech recognition, for synthesis, for language understanding, it works very reliably.

[0:07:20.1] SC: Did you happen to catch any of Jeff Hinton's talk prior to this?

[0:07:24.2] GD: Yes.

[0:07:24.9] SC: What do you think about his post-SGD capsule? Well, it's not post-SGD. He actually – the starting point is that SGD is really the only thing that we know works. It's more post the traditional model of the neuron.

[0:07:42.0] GD: Yeah. I almost think of it as post covnets. One thing we've realized recently is just there is a lot of complexity in modeling, that while we like to think of deep learning as a general purpose learning algorithm, as you start applying it to different applications, like my experience has been spending a lot of time applying it to speech recognition. You do get some benefits from more data, and from some general purpose aspects of the learning algorithm to the extent that it's robust to different speakers or different variations in different environments.

You also get a lot of benefits from specialization. Finding the right neural network architecture seems like it matters a lot. As we look into details for different applications, as we spend time turning neural network or architectures for different applications, you see very different structures emerge. It wouldn't surprise me at all if there is a more efficient, more general purpose structure for vision than covnets.

[0:08:49.8] SC: Yeah, one of the – really help me see that was a blog post by Steven [inaudible 0:08:55.4] a while ago. It may be almost a year ago at this point, but he talked about network architecture being the new feature engineering. [Inaudible 0:09:03.7] was post-covnets, but he started off by talking about, like calling into question the basic neuron structure. He did pivot to talking about the network architecture at a higher level. Was there a piece in there where he suggested what might be the successor to the traditional neuron architecture?

[0:09:30.5] GD: I think it's about this concept of capsules, which might be groups of cooperating neurons. Then the way that they cooperate together might be more complex. Let's see. I feel like from a computational perspective it's actually hard to get away from the formulation of neurons that we have as the basic building block, where even if there is something that's algorithmically more efficient or more well-matched to the problem, the computational building blocks that we have have been so highly tuned, that if you made a very substantial change it might be better at the algorithm level, but it might be very inefficient on the type of computer that we know how to build today.

[0:10:11.8] SC: It breaks this whole ecosystem that we've built up around this traditional way of building neurons and networks and solving them.

[0:10:21.2] GD: Yeah. So much of the existing technologies are built on top of hardware support, also algorithm support for linear algebra, dense linear algebra. It's actually surprising to me how effective that's been given that the primitives are so old and that there is a simple – it's a very surprising to me that those building blocks have gotten us as far as they have.

[0:10:42.8] SC: We took a little digression, I guess, before we got to what you're up to at Baidu.

[0:10:50.2] GD: Sure. Baidu, really the Silicon Valley AI Lab is about building new breakthrough technologies in AI that especially have connections or enabled new products. We focus on things that we can't currently do today.

There are really multiple ways we end up attacking this. The thing that I focus a lot on is just the idea of scale that as we have faster computers that can train larger and more complex neural networks that it's not the only way, but that's a very reliable way of improving accuracy or enabling new capabilities.

We've seen this in vision to a large extent. It's interesting. When we started working at Baidu, there is a question whether you could apply this outside of vision. We spend a lot of time looking into speech recognition. I think looking back on that, it works very reliably. You can definitely apply deep learning outside of vision to many different applications.

Sometimes now instead of thinking what is the new application that you can apply deep learning to, I sometimes wonder are there any applications that are not well matched, that we won't be able to make significant progress on by just applying a simple recipe of deep architecture's large data sets, large scale compute. I haven't found one yet.

[0:12:13.1] SC: Nice. We talked a little bit about – before we got started, we talked a little bit about the fact that you work with one of our previous guest, Shubho Sengupta, who was at Baidu and is now at Facebook, is that right?

[0:12:26.0] GD: Yeah. He's playing DOTA.

[0:12:29.4] SC: Nice. He and I spoke pretty extensively about the speech translation – I forget the specific name of the project, but the Baidu Speech –

[0:12:41.3] GD: Deep Speech.

[0:12:42.1] SC: The Deep Speech, right. In the course of that conversation, we talked a little bit about some of the scalability challenges that your team ran into in tackling that problem. Here at this conference, you're talking about – you have a talk tomorrow in fact, about some even further work that you've done. Can you tell us about what you're planning to talk about?

[0:13:07.1] GD: Sure, definitely. This is definitely along the lines of scaling deep neural networks. We pretty consistently find that if you throw more data at the problem, it isn't the only way, but it is a very effective way of reducing error arrays and improving accuracy.

Oftentimes, when you keep throwing data at the problem, you eventually run into some limitation; sometimes you run out of data and sometimes you run out of patience to wait for your system to train.

There's one example that I think drives this point home that we once had a model that ran – I let it run on a large cluster on about 64 GPUs for about six months.

[0:13:53.8] SC: Wow.

[0:13:54.9] GD: We were still getting improvements in accuracy at the time. I decided to pull the plug.

[0:14:00.1] SC: If you're still running your training model for – you were saying your incremental error is decreasing as you ran it.

[0:14:11.7] GD: Yeah. This was a state of the art model. It's actually improving the state of the art as it runs. Every minute, it's getting a little bit better than any model that we've had before. Then after six months, you really go back and look at that and say, "Well, I could let it run for another few years, but I'd like to use it now."

We're always looking for ways to improve speed. Oh, man there's something that's really need to that but I can't talk about yet. It's a weird situation to sometimes be in where, like you know why something happens, or you know that something will keep happening, but you can't talk about it.

One of the things that I know is that for many applications, we will continue to see improvements in the state of the art from faster computers. I can't tell you why, but I'm pretty sure. I'll tell you why soon.

[0:15:02.6] SC: Okay. Are we talking about a theoretical results, or a – Okay.

[0:15:08.5] GD: Yeah, I think so.

[0:15:09.6] SC: You're nodding yes, for those who can't see.

[0:15:11.8] GD: I'm nodding yes. Yeah. This is one of those things that will definitely be surprising to people when we can finally talk about it, but sorry I can't talk about it today. You'll just have to take my word for it that we need faster computers. The talk tomorrow is going to be about a way that we can make computers faster for deep learning.

I spend a lot of my time thinking about this, like what's the best that you could do? How fast could you possibly make a computer, even our understanding of physics and our existing technology? I think one thing the industry is realizing is that we spend a lot of time focusing on general purpose computation, so building computers to run Windows, or your browser.

If you specialize, if you build a computer that's good at only a few things and not everything, it can do a lot better. We're exploring right now how do you build computers that are good at AI, are good at deep learning.

[0:16:08.8] SC: Is this different than what others in the space are doing, like TPUs and things of that nature?

[0:16:16.3] GD: Let's see. This is about a very specific technique. It might be one technology that might go into a chip, like a GPU or a TPU. Right now, many of these designs are using a lot of the same technologies. This is a new one. This one has a pretty high upside. This one has maybe order of magnitude upside.

[0:16:38.3] SC: Okay. Before we dive into that, can we take a second to characterize the thing that GPUs and TPUs are doing, that's gotten them the benefit? Then we'll dive into this approach and what makes it different.

[0:16:52.1] GD: Sure, definitely. Let's see. I feel like one of the other two – okay, there are a lot of differences. One thing that's worth keeping in mind is that modern processors incorporate probably thousands, probably even more optimizations.

These are technologies that will improve their performance in some way; they might be circuit level, they might be architecture level, they might be in the software stack. It's very hard, because real designs are composed of – you've pick out your favorite – out of this pool of thousands of technologies and that becomes the new processor that you build.

These distinctions like TPU, GPU, CPU, they're very high-level and they gloss over all of those details. I think what's more important than the name is what it does, how fast is it actually. What is the result that you get from it? How fast does it run a model that you care about?

We've seen things that were called CPUs being commonly used for training maybe 10 years ago. There is a transition repeat, we started using GPUs. The important thing about GPUs was optimization for parallelism and that there is abundant parallelism in neural network computations.

Some of the things that are a few, like a couple out of that list of thousand things that are being added into the next generation are optimizations around locality and low precision. The technology I'm going to talk about tomorrow is focused on low precision.

There is a big difference. When a lot of previous technologies have been discussed or proposed for low precision, it's mostly been focused on inference, and not training. This will be one of the first results. As far as I know, large scale result that focused on using low precision for training. I think the high-level conclusion is it finally works. It was enormously difficult.

It was actually a weird surprise that when you try doing low precision for training versus inference, we didn't really know that we would see this. We started looking into this, but it just turns out that for some reason, inference is so much easier than training. That even very drastic reductions in precision, like moving from double precision down to even 8-bit, or possibly even lower fixed point representations.

It works just fine across many different models, but if you try and do the same thing for training, things fail. It's actually a funny point to me that we made this implicit transition. CPUs commonly support a high-performance double precision. GPUs don't. GPUs have historically optimizer on single precision, so 32-bit folding point instead of 64-bit folding point.

It turns out it's expensive to do this in a GPU, to do 64-bit on a GPU. Whereas, it's pretty cheap at a CPU, because you don't have to replicate these things, this unit very many times. On a GPU you have to replicate it a lot. If you replicate something big a lot, it becomes expensive.

It's surprising that the whole industry – when I started watching people train deep neural network they might write scripts in MATLAB, or call CPU libraries directly. Those things by default use double precision. When the industry switched to GPUs, they switch from double precision to single precision.

We got so lucky. It turns out that it didn't really matter. I think that was just by luck. When we tried doing the next step, we tried moving from single precision to half precision, so moving from 32-bit floating point to 16-bit floating point, things started failing all over the place.

[0:20:50.9] SC: What caused those failures?

[0:20:54.4] GD: There were a lot of them. Let me try and draw a couple big categories. One was just differences in range. One of the points of having flowing point as opposed to fixed point is that you have a very large dynamic range. You might be able to do an operation, like an add of a number that's – where one number is a billion and the other number is 10 to minus 5. That works.

You need your range to extend from the smallest numbers that you want to deal with to the biggest numbers that you want to deal with. It turns out that if you look at all of the operations that go on in forward propagation, back propagation, nonlinearities in the SGD algorithm, there is actually a pretty large dynamic range.

[0:21:45.0] SC: Aren't we typically normalizing to try to get rid of some of that?

[0:21:49.4] GD: Yeah. It's interesting. I'll come back to that point. It's an interesting thing related to that point. Yeah, let me come back to that. I feel like the number one reason why when we just – the first experiments we did were just convert all of the 32-bit numbers to 16-bit half precision numbers and try using exactly the same algorithm.

Also at Baidu, because we're working on speech recognition, we started doing this for recurrent neural networks. It turns out, that was one of the harder examples. We started with one of the harder cases it turned out. We would see all sorts of failures.

[0:22:27.5] SC: What makes RNNs particularly harder?

[0:22:30.9] GD: I think it has to do with accumulated errors. As you keep doing this repeat application of a matrix multiplication with the same weights, you're thinking about, or overtime this just encourages extreme values; your extremely small values, or extremely large values. Sometimes people call this the vanishing gradients, or exploding gradient problems.

For speech recognition, we see very long time series. We might see hundreds of iterations of an RNN, or maybe thousands. We saw large accumulated errors over time. One of the biggest source of errors we came across was when you're actually combining gradients with – the gradient update with the master copy of the weights.

When you have this model, it turns out – it seems like SGD just makes these repeated small updates to a model. If you look at it from a range perspective, there is a large difference in magnitude between the magnitude of a gradients and the magnitude of the weights. When you and do operations on those numbers that have very different magnitudes, you get loss of information, or you get errors.

That was one of the biggest problems we had with the training and half precision. It seemed like, yeah moving – for some reason, the errors introduced from the – In floating point, things work out well if the numbers are different magnitudes, but not by too much. It turns out that the difference for a single precision versus double precision was okay. It ended up being borderline for multiple applications when we were looking at the difference between single precision and half precision. We had to introduce some changes in order to deal with that.

[0:24:25.8] SC: One of the questions that came up in this previous conversation with Shubho, and which we touched on some of the stuff, like I think pretty tangential is like the end of our conversation, I think, if I remember correctly. We were talking about reduced precision and I think I asked the question like, you can reduce the precision in multiple places. You can reduce the precision in your weights, you can reduce the precision in your outputs.

When you're talking about reduced precision, are you talking – it sounds like you're talking about reduced precision everywhere, just running on reduced precision infrastructure, or on a reduced precision mode and not being particularly discriminating in terms of where you reduce the precision. Is that what you're referring to?

[0:25:13.1] GD: Yes. We're trying to keep it simple. We feel like if it ends up getting very complex and it's difficult for people to know how you would actually apply this to a real model.

[0:25:25.0] SC: Then you get back into your architecture feature engineering complexity issues?

[0:25:29.7] GD: Yeah, we definitely didn't want to introduce this as another hyper-parameter. Maybe this only works for a few layers, but it doesn't work in these places and so you can – you have to make this hard choice of deciding which ones to convert and which ones not. We wanted it just to be like a switch, and you would turn on the switch and you would get the performance improvement.

I think we finally got to that point. But for this kind of reason, there are a lot of problems along the way. I mentioned the difference in magnitudes between the updates and the weights as a source of errors. The other big one was just accumulated errors in long dot products.

It turns out that taking weights convert quantizing them to 16-bit, and then doing multiplications and activations with those weights didn't introduce too many errors. In neural networks, especially in recurrent neural networks as the layers get big, you end up with these long dot products. You're doing a running sum over each row, or all of the inputs of a neuron. Each operation has an accumulated error, so everyone in the sequence is going to add some amount of error.

[0:26:46.8] SC: Now we're not talking about error in the machine learning modeling since we're talking about floating point error?

[0:26:54.0] GD: Yeah. We're talking about just you really wanted to do this multiply operation. You didn't get the exact result. We had to clamp it to a value that's representable by the computer. Each time you do that, you introduce quantization error.

Normally, as long as you have enough bits, that quantization error is small enough that it doesn't really affect the final result too much. Exactly what too much means is very application-dependent. Then the complex systems like neural networks it's really hard to know how much error is too much error, other than just trying it on a real application.

What we found for real applications, like for speech recognition, or for translation, the error introduced by doing ads in 16-bit was too much error. Models would diverge, or models would achieve significantly worse accuracy than the 32-bit baselines.

We went back to that and we tried a whole bunch of things like we tried hierarchical reductions and a bunch of things that ended just being complicated. Eventually, we went back and looked at the circuits and came to the conclusion that it wasn't that expensive just to put in a 32-bit adder. We have a bunch of 16-bit multipliers and then you have a few 32-bit adders. If you look at the performance improvement that you get from that, it ends up being most of the performance improvement that you would've got if you would've built 16-bit multipliers and the 16-bit adders.

[0:28:23.7] SC: Okay. Yeah, we ended up with a mixed precision format. You end up doing multiplication in 16-bit, but then the addition in 32-bit. There are a lot of other things we ended up looking at. There is still some other failure cases, but those are really the two big things. As long as you keep the master copy of weights in 32-bit and as long as you do all the additions in 32-bit, you can do all the multiplications and you can represent all the activations and intermediate copies of weights and weight gradients in 16-bit.

[0:28:57.8] SC: To take a step back and make sure I understand why we're doing this; are we talking about performance and computational cost, or are we talking about unit compute cost for this chip by having narrower buses and things like that? Are we talking about training time performance? What are the factors that are driving us to say, "We want to do this," and not just we can do it and reduce it precision. We want to do this and reduce precision.

[0:29:29.0] GD: Sure. Yeah. Why we want to do this and reduce precision? It's really so we can build more efficient hardware. With and without this technique, you can just do a comparison. If you're building the same processor with and without this technique, there's a fair amount of performance at play. It might be something like 4 to 8X difference in really both sides of the total performance or energy per operation, which would translate into efficiency.

[0:29:53.9] SC: Okay. By going to reduce performance, we can – or by going to reduce precision, we can increase some positive performance and energy consumption by 48X nearly order of magnitude.

[0:30:10.5] GD: Yeah, we could finish my six-month model in maybe just a single month.

[0:30:15.6] SC: I guess, I'm trying to get at this question. I don't know if the question makes sense, but is it – is it something inherent about the lower precision, or is it the fact that the lower precision allows us to use new compute architectures that are faster in other ways?

[0:30:33.6] GD: Sure, definitely. Do you get this performance improvement on existing computers? You get some performance improvement, because you're moving around less data, but it might be closer to 2X. It really depends on whether your compute bound for bandwidth bound, but the maximum might be more like 2X. If you build another computer, if you built a new processor that was optimized around this idea, you could do even better. You can realize the 48X.

[0:30:59.4] SC: Okay. Low precision fundamentally allows you to do – train these neural nets by moving around less data, right? 16-bits instead of 64, for example. You get some advantage in doing that, even if you're just in low precision mode on a general purpose computer. It also allows you to build chips that are specific to running in low precision and that gives you – that's where you get the big opportunity to bump up your speeds.

[0:31:30.8] GD: Yes, exactly.

[0:31:32.5] SC: Okay. You were here talking about the actual chip, is that correct?

[0:31:36.7] GD: Yeah. We're going to talk about the Volta GPU from NVIDIA. This was a collaboration with NVIDIA. It's worth noting, this hardware has been shipping for a while. But the side of it that we're talking about now is the validation that we've done it. We've actually shown that you can train models in low precision. We've looked at over 15 large scale complete end-to-end deep learning applications. It really easy to build hardware that gets great performance numbers, but isn't able to run any real algorithms.

[0:32:11.0] SC: From the point of view of low precision, the Volta is – like it's general purpose, right? It's not a chip that's specifically designed for low precision.

[0:32:22.5] GD: It did actually have – they're called tensor course. It's the name for them is tensor core. That is this operation I'm talking about. It's a specialized unit that does 16-bit multiplication floating point with 32-bit floating point addition. That unit was designed as a result of this study.

[0:32:43.3] SC: Got it. Got it. Now if I remember correctly when this was announced, they made a big deal about not the floating point side of things, but like N8 performance and things like that. How does that all fit in?

[0:32:55.2] GD: Sure. Definitely. I alluded to this maybe in the beginning that inference just is easier for some reason than training.

[0:33:04.5] SC: That's like we can do N8 on inference side and is easy and it just works and it's faster.

[0:33:11.0] GD: Yeah, exactly.

[0:33:11.4] SC: Got it. Okay.

[0:33:13.1]GD: I don't know. I don't know that this whole topic has been really fully explored yet. Maybe someday in the future we might see someone who gets in-date training to work, but as far as know I've never seen it.

I know there are a lot of – there's a lot of work on very reduced precision, like even down to binary. One thing that's worth noting about these approaches is that they either have accuracy losses, so you trade precision for accuracy on the complete application, or they only apply to inference and not to training.

[0:33:49.5] SC: Okay. Got it. Reduced precision. You did some validation that shows that essentially running in this mode is a generalized approach you can take. Now things that you need to do or switches that you need to flip when you're training your model in order to get it to work accurately, or to work correctly?

[0:34:12.3] GD: Sure. One switch that you need to flip is you need to decide to do this. You need to decide to represent things in 16-bit and do your matrix multiplications or convolution operations in this mixed 16-bit, 32-bit format. That's somewhat of a global switch. You can just turn that on for the entire program.

The other thing that you need to do that we found is essential is you need a master copy of the weights. In your optimization algorithm, like your implementation of SGD, you need to have a separate copy in 32-bit of all the weights. Only when you're doing forward propagation or back propagation do you convert from that into 16-bit.

Both of those changes we founds, or the first one is really easy. The second one can be encapsulated inside the optimization algorithm. At least when you're designing a network, you don't have to think about this.

[0:35:07.4] SC: I can envision how I might do this if I was writing the – if I was implementing SGD myself to the higher level frameworks and toolkits all know how to do this, or is that yet forthcoming?

[0:35:22.5] GD: It's straightforward to do this in most frameworks, but there needs to be developers who are working on those frameworks will actually add support for this. When we did this in the framework that we have in Baidu, it's something like 15 lines of code change. It's really minor, but you still have to do that. Otherwise, you won't get access to the improved performance.

[0:35:46.0] SC: Okay. Anything else you talked about in your – I keep saying it in past tense. Anything else you're going to talk about in your talk that you want to mention?

[0:35:54.9] GD: I guess the last thing is that this is one piece, this is one technology that gives us a large improvement in performance. I think we're aware of a lot of them that haven't been realized yet. I mentioned before, the hardware industry for a long time has been creeping along. It's actually very difficult to realize large improvements in the sequential performance. At least for parallel programs, like graphics applications and things would run on GPUs, performance has been increasing following something like the popular form of Moore's Law, so exponential growth.

For AI, if you're only thinking about running deep neural networks, you can probably do a lot better than that in a short term. We might not have to wait 10 years to get a 1000X faster. It might happen in just a few years.

I'm going to mention some of the other ways that haven't been implemented yet, but that we know about or likely to happen in the future.

[0:36:53.9] SC: Can you rattle those off? This podcast will not be published before your talk tomorrow.

[0:36:58.0] GD: Sure. One of them is a array parallelism. I think this is one of the other big one is a array parallelism. I had a Forbes article about this where I was talking about locality; the importance of locality. If you build processors around the idea of locality and parallelism, not just parallelism, you end up with something that it looks – I call it like an array processor, rather than a vector processor.

You're thinking about the core instruction that you're doing instead of adding or multiplying two vectors together, you're adding or multiplying two arrays together. You see things like these in designs like the TPU. I feel like the thing that is wrong with those designs is that they don't find any of the curve that this is beneficial, but you don't have to go all-in on it to get most of the benefits. You actually are trading off.

[0:37:48.6] SC: You don't find any of the curve. What exactly does that mean?

[0:37:51.8] GD: It means working on arrays is a good idea, but they don't have to be enormous arrays. You actually are trading off flexibility for performance when you're making the arrays bigger. You shouldn't make them enormous. You should make them big enough to get most of the savings and energy.

[0:38:09.9] SC: In terms of order of magnitude, are you talking about little teeny ones, like convolutional kernel sizes, or are we talking about something bigger than that, or?

[0:38:19.8] GD: Yeah, it's more like 16 by 16 than 256 by 256, if that makes sense.

[0:38:25.9] SC: Okay. Yeah. Yeah. Interesting. Any others on that list that come to mind?

[0:38:31.0] GD: One of the ones that doesn't work yet, but I think had is very promising is sparsity. Working with sparse representations rather than dense representations. It might seem like that's incompatible with the one that I just send the array parallelism. We haven't shown this yet, but I suspect that they're not incompatible.

[0:38:50.4] SC: What I'm hearing putting the two together is that we're living in a world that thinks about all this stuff as composite vector operations. By thinking about this at the level of matrixes, there are opportunities there.

[0:39:04.1] GD: Yes. Yeah, that's a good way of thinking about it.

[0:39:07.1] SC: Interesting. Well, I really enjoyed this chat. Thank you so much for taking the time.

[0:39:11.4] GD: Glad to be here.

[0:39:12.4] SC: Great. Thanks, Greg.

[0:39:13.2] GD: Thank you.

[END OF INTERVIEW]

[0:39:16.4] SC: All right everyone, that's our show for today. Thanks so much for listening and for your continued feedback and support. Thanks to you, this podcast finished the year as a top 40 technology podcast on Apple Podcasts. My producer says that one of his goals this year is to crack the top 10, and to do that we will need your help.

Please head on over to the podcast app, rate the show, hopefully we've earned five stars, leave us a glowing review and share it with your friends, family, co-workers, Starbucks baristas, Uber drivers, everyone. Every review and rating and share goes a long way, so thanks in advance.

For more information on Greg or any or the topics covered in this episode, head on over to twimlai.com/talk/97.

Of course, we'd be delighted to hear from you either via a comment on the show notes page or via Twitter at @twimlai.

Thanks once again for listening, and catch you next time.

[END]