EPISODE 507

[INTRODUCTION]

[0:00:00.6] JM: Training a deep learning model involves operations over tensors. A tensor is a multidimensional array of numbers. For several years, GPUs were used for these linear algebra calculations and that's because graphics chips are built to efficiently process matrix operations.

Tensor processing consists of linear algebra operations that are similar in some ways to graphics processing, but not identical. Deep learning workloads do not run as efficiently on these conventional GPUs, graphical processing units, as they would on specialized chips that are built specifically for deep learning. In order to train deep learning models more efficiently, new hardware needs to be designed with tensor processing in mind.

Xin Wang is a data scientist with the artificial intelligence products group at Intel and he joins today's show to discuss deep learning hardware and flex point, which is a way to improve the efficiency of space that tensors take up on a chip. Xin presented his work at NIPS, the Neural Information Processing Systems Conference, and we also talked about what he saw it NIPS that excited him.

Full disclosure; Intel, where Xin works, is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:01:30.5] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes. You can quickly provision clusters to be up and running in no time while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked into any one vendor or resource. You can continue to work with the tools that you already know, such as Helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications off-line. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

Check out the Azure Container Service at aka.ms/acs. That's aka.ms/acs, and the link is in the show notes. Thank you to Azure Container Service for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:02:57.8] JM: Xin Wang is a data scientist with the artificial intelligence products group at Intel. Xin, welcome to Software Engineering Daily.

[0:03:04.7] XW: Thank you.

[0:03:05.5] JM: Today we're talking about deep learning and its impact on hardware, because it is such a widely used workload type that we actually want to change the hardware that our deep learning workloads are running on, an we're going to get into that, but I want to just start with talking about the state of deep learning, kind of where we are at the end of 2017, beginning of 201, because you are just at NIPS, and at NIPS there were all kinds of discoveries announced all throughout the stack. What are some of the coolest deep learning success stories that you've seen recently?

[0:03:46.2] XW: I think one of my favorite recent deep learning successes, I should say, probably is machine learning in general is definitely AlphaGo Zero. I think deep learning as part of machine learning is getting a lot of momentum these days, but deep learning itself is basically just a very fancy function of proxy meter I think at NIPS this year we see a lot of very cool examples of combining deep learning with traditional machine learning techniques. In general, deep learning action requires a lot of the traditional deep learning — Requires a lot of data to training, but we see a lot of nice techniques, for example, like self0learning, meta-learning

techniques that allow a very high performant system to be built with very few training data, or in the case of a game play, no data at all.

[0:04:42.6] JM: So when you look at the success stories across all of the different domains, why is it the deep learning is able to be a successful application in such a wide scope of different domains?

[0:04:59.1] XW: I think the general answer to this question is really twofold. The basic of the theory of deep neural networks and the current way of training deep neural networks are already there for decades and it's widely believed that why recently in the past decade picked up such big momentum is because, number one, we have a deluge of data from all aspects of our life, all industries, and second is a lot of computational power. This is really not there, both of these factors back in 1990s, for example. So this is why deep learning has gained so much momentum. It has become so successful.

[0:05:43.7] JM: For people who are less familiar with it, why don't you just describe a typical deep learning task? We've explored in-depth in previous episodes, but I do like to hear it from multiple people. So describe what a typical deep learning task is.

[0:05:58.2] XW: Right. I can explain probably using a very simple example. It's probably everywhere in the introductory deep learning. Let's say take a very simple task, like recognizing or classifying a certain group of data, like images. You have lot of images and each one of them have certain label on it signifying the content of that image, like cats or dogs, and basically you build a deep neural network that has many layers and it has many, many parameters of all those layers and they have — Basically, it gives you a deep nonlinear transformation to approximate a function that takes into it data, and it gives you the label.

Basically, this kind of explains how inference, what we call [inaudible 0:06:48.6] in deep learning, is basically just like forward passing a piece of data and the deep neural network will output a label, that's the prediction. Deep neural networks are actually trained through a lot of data by doing what I just described just forward passing, and also the output will be compared to the target, let's say in a training data, every example data point is labeled and there is certain loss function that you can design to penalize errors in the prediction, and this error is going to be

back propagated through the network. So basically, in the form of the gradient of the arrow with respect to its parameters, and this is basically the second operation involved in training. The third one is basically in optimization step. Basically, you update the parameters of the network so that the network will actually adjust itself toward a direction that gives you a better prediction.

[0:07:57.2] JM: You often hear the term tensor associated with deep learning. Remind us what a tensor is.

[0:08:04.8] XW: You can think of tensor as a generalization of vectors and matrices. Basically, it's like a structure that gives you a multi-linear algebra on top of a vector space. Basically, in deep neural networks, a lot of values can be organized into multidimensional arrays. These are basically the tensors what we're referring to. For example, like activations, weights, these things can all be ingredients. These things can all be tensors, and the tensor operations are basically just linear operations, and these are basically the most expensive or the majority of the computation of the deep learning task.

[0:08:49.9] JM: When we look at a tensor as a data structure, how is a tensor laid out in memory or on disk?

[0:08:57.7] XW: Typically, just like any multidimensional array structure. So it's basically a kind of a — You can think of it as just a multidimensional data structure. So it's pretty much like a linear layout, but you have certain different levels of strides that defines the axes of these dimensions and the axes of the data structure.

[0:09:20.5] JM: When we take deep learning operation and we're looking at it at the programmer level, for example, the Python level or maybe you're looking at a specific framework. Take me through how that task looks at different levels of compilation. If I compile it from the Python world down to the lowest level and I'm going to run it on specific hardware, how does it look at those different levels of compilation?

[0:09:51.7] XW: So I can probably just give you a glimpse, a view of that. If you run a deep learning task, let's say just a training of deep neural network, you typically just describe your deep neural network and how your data, cost functions, optimizers in a framework. Basically,

there are a lot of Python-based frameworks and these are basically the way you describe the model at the algorithm level, and then it's going to be converted into intermediate representation, typically, a computational graph, and that is going to be further compiled to a kind of machine-specific version that can be run on all kinds of infrastructures, hardware infrastructures, like CPUs and GPUs, which are most common hardware that you see these days.

[0:10:47.8] JM: At the lowest level, what kinds of operations are being done to execute a deep learning task? When we talk about how the deep learning task is being executed at the hardware level, what's going on there?

[0:11:01.7] XW: Right. So I think for the current status of deep neural networks, main operations down there are basically just dense linear algebra operations, basically, multiplications and additions, accumulations of huge and matrices. So it's basically — There's probably going to be differences in the future, because deep learning is also evolving very fast, but currently generally speaking, deep neural networks, the lower-level operations have dominated, basically, by linear algebra.

[0:11:37.2] JM: And what kinds of resources is a deep learning task consuming? When we think about from a high-level perspective across — When we're on a big deep learning task, are we doing the — Is there lots of memory that's associate — And I guess you could break this into training or actually just a classification instance, like if I train a classification model, that's one set of questions, and then if I actually classify a given piece of data, that's another question. What kinds of resources are we consuming in these different deep learning tasks? Are we consuming lots of memory or disk space or CPU or multiple machines? What are the bottlenecks?

[0:12:20.5] XW: Actually, you are absolutely right. The story is quite different between training and inference. Let's say take inference first. Actually, inference can be very cheap, because it's just a forward pass, and a lot of — Basically, you need a memory to hold all the parameters of your network certainly, and forward pass basically just take a batch of your data and go through the network. Actually, training can be a lot more expensive in terms of both memory and computation.

In general, deep learning is really not cheap, especially for training, and think about the additional steps that you need for training. When you back propagate, you really actually need to store all the activation values through all the layers when fprop, forward propagation, and you also need — If your models are huge, and it's typically the case, the modern deep learning, deep neural networks can have millions and millions of parameters. In this case, the bottlenecks are really number one. A lot of computation needed to be done, those tensor operations on these data structures. Number two, there is a lot of data being moved around and you need to do a lot of reads and writes to memory.

[0:13:41.4] JM: How parallelizable are these deep learning — I guess we should probably talk more about training, because training seems like the more interesting, if we're talking about training versus inference. Training seems like it's much more intense. So when we talk about training a model, how parallelizable is that?

[0:14:02.8] XW: I think there a lot of levels that we can talk about this. There are actually a lot of parallel training schemes, for example, you can divide your data. For example, if I have huge batches, you can actually divide data, this is called data parallelism, or you can have model parallelism. Basically, it's kind of dividing your weight matrices, for example. So they have their advantages and disadvantages. I think this is not a topic today, but I think there are a lot of advancement in this area. So I think that this problem is really a kind of deep learning, specific problem. So it has something to do with the typical arrangement of deep neural networks.

[SPONSOR MESSAGE]

[0:15:01.0] JM: Your company needs to build a new app, but you don't have the spare engineering resources. There are some technical people in your company who have time to build apps, but they're not engineers. They don't know JavaScript or iOS or android, that's where OutSystems comes in. OutSystems is a platform for building low code apps. As an enterprise grows, it needs more and more apps to support different types of customers and internal employee use cases.

Do you need to build an app for inventory management? Does your bank need a simple mobile app for mobile banking transactions? Do you need an app for visualizing your customer data? OutSystems has everything that you need to build, release and update your apps without needing an expert engineer. If you are an engineer, you will be massively productive with OutSystems.

Find out how to get started with low code apps today at outsystems.com/sedaily. There are videos showing how to use the OutSystems development platform and testimonials from enterprises like FICO, Mercedes Benz and Safeway.

I love to see new people exposed to software engineering. That's exactly what OutSystems does. OutSystems enables you to quickly build web and mobile applications whether you are an engineer or not.

Check out how to build low code apps by going to outsystems.com/sedaily. Thank you to OutSystems for being a new sponsor of Software Engineering Daily, and you're building something that's really cool and very much needed in the world.

Thank you, OutSystems.

[INTERVIEW CONTINUED]

[0:16:54.0] JM: We did a show with Will [inaudible 0:16:55.7], who I think you work with, and he was talking about this model, parallelism versus data parallelism, and how you have to figure out are there mathematical prop — Well, if I remember correctly, are there mathematical properties where if I break up my data and I feed my data to two separate instances of the same model, can I just average those two models together to get one model at the end of it? It sounded like that's possible, like you can do that, which is pretty interesting.

[0:17:37.0] XW: Right, in basically like the stochastic gradient descent is like the common optimization method that we use to train. So in this case, actually, you can actually divide the data in average later, but for this particular scheme, I think a drawback is like you need a — At

the end of the propagation, you basically need to kind of broadcast all the gradients to all the workers, like different nodes that's running the same model.

[0:18:06.0] JM: Getting to hardware, why does the typical hardware that we use, like in my iPhone or my laptop wire. Why are these chips — Actually, we could even talk about server chips, modern server chips, most of the modern server chips. Why do these chips have difficulty processing deep learning workloads efficiently?

[0:18:31.4] XW: I wouldn't say that it's difficult, because right now, at least all the deep learning tasks are run on these general-purpose hardware, like CPUs and GPUs, and most of the training actually are done on GPUs. So I think it's fair to call it not very efficient, is because these general-purpose hardware are designed for all types of computations and it has its generality, and the generality is actually pretty costly. When you come to pretty specific workloads of the specific kind, just like you mentioned earlier, i if you have a specific type of math that you do again and again and has a specific type of processing and memory read and write pattern, then a lot of general kind of procedures may be streamlined, and then you can save a lot.

[0:19:27.0] JM: So in Bitcoin mining, we've seen the rise of ASICs, these application-specific integrated circuits. I'm not a hardware guy, so these things were some of my — Just learning about Bitcoin was some of my first understanding of why you would want to have specific chips for specific operations, because there are certain operations that a Bitcoin miner is going to do so often that it makes sense to have a specialized ship for it, and this is also the case for deep learning. There are specializations that you can make to a chip to optimize it for deep learning. What are some of the specializations that you can make to a chip to optimize?

[0:20:10.2] XW: I think I'm glad that you brought this up, Jeff, and this is actually a perfect example of showing why a specialized hardware can gain you a lot. So, like Bitcoin is a specific case. I think it might differ from deep learning in a few ways, but in general it's the same idea. So if you have general-purpose hardware, like CPU or GPU, these are basically like kind of one tool fits all type of solutions. So you can, basically, solve any kind of problems. But in the particular case of Bitcoin mining, it's a very kind of fixed problem with the very fixed solutions. So you basically just need to crunch a lot of numbers to pipe through like hash functions.

It is a perfect example to show how you can basically use specific hardware to, number one, save silicon and at the same time save power, and also reach your solution faster. But deep learning is a little bit different in this case because deep neural networks are kind of diverse. It has its fixed pattern, yes, and this has its type of problem, but the problem size is more variable than a specific case of Bitcoin mining. I think, also number two, is like Bitcoin mining, algorithmically, it is kind of trivial to parallelize, but deep neural network architectures come in many different flavors, and we have recurring neural networks too. More generalities maybe need needed to be put in there in terms of parallelizability.

Also, I think another difference is, basically for the case of Bitcoin mining, energy is probably the primary concern over there, but for deep learning use cases, you may also consider time, and energy, of course, is also a very important factor here.

[0:22:20.6] JM: Ultimately, new hardware needs to be designed with tensor processing in mind. The tensor is this multidimensional array. GPUs can efficiently process multidimensional arrays. My understanding is that much of the operations that we're doing with tensors is similar to the kinds of processing we're going to do on multidimensional arrays for graphics cards. Why do we need something new? Why do we need new hardware specifically for deep learning? Can't we just reuse our GPUs?

[0:22:56.7] XW: Indeed. As I mentioned, dense tensor, processing dense tensor operations are basically the major part of deep learning, basically almost all the workload is there. So that's why you see GPUs are actually very useful these days for accelerating deep learning, especially training, and you can think of GPU as a kind of specialized hardware in this sense, but there are several major differences between graphics and deep neural networks training, and also the deep learning hardware is also evolving and these, basically, differences will basically kind of determine the future of deep learning hardware and I believe it's going to diverge significantly from graphics, because its demand is different.

I think the number one difference is it's basically — For deep learning, we also want to cut a lot of corners in terms of using low precision. So this is some, the topic of today, the flex point. It's

one of the schemes that use low precision, but graphics typically use like single precision floating-point, and that's the difference.

Also, for deep learning, we have a lot of new techniques of trying to basically grow a network or prune a network and make certain parameters more sparse, and so on, and this will basically deviate from dense matrix operations. So this may also result in some kind of different considerations in hardware. Also, there is a lot of kind of — It is a big bottlenecking in moving data, so there's new type of deep learning hardware that move the processing pretty close to memory or even use like emerging memory technologies to do computing itself. So this is another big difference.

Also, for the current GPU architecture, typically a lot of optimizations can be done by optimizing the hierarchy of memory caches. For graphics and for deep learning, this optimization may be kind of different for this. Finally, deep learning is very different from graphics in that the size of the problem can be very big and it's not a fixed one. So a lot of cases, you basically need to scale up the problem that you want to have some kind of very fast networking amongst your different nodes of your hardware. So you also want to have very good networking built-in, which is not also required for graphics.

[0:25:43.3] JM: The first thing you mentioned with trading off accuracy in order to improve efficiency, why is that okay to do in a neural network? Why can we sacrifice accuracy in exchange for efficiency? I guess if you wanted to draw a comparison, why wouldn't you be able to do that in graphic, for example?

[0:26:08.5] XW: Right. I think this is basically the — I would say it's basically the legacy, because all of these mathematical, underlying mathematical operations that we do on deep learning today and also on graphics, these are all from theory, and this theory all basically have very high precision numbers, and we all started in theory with real numbers. We have those standard formats, typically floating-point formats, that allow us to encode these values in a very efficient way with a certain bit width. For example, 16, 32, 64-bits that gives you a very well-balanced dynamic range and a precision.

In these cases, basically, we kind of inherited that in our current hardware. As you said, we may cut corners further, because if we want to make a specific processor for deep neural networks and if a certain layer, for example, and a certain tensor in a certain layer only requires, let's say, 8-bits precision for certain operation and the values that's going to be in this particular tensor seldom exceeds the encoding range, then it's pretty safe for us to use very cheap and low precision and without taking any or significant hit on its performance.

[0:27:32.9] JM: Could you give an example? I feel like we should be a little more concrete here. Is there an experiment that was run recently where they looked at maybe doing some sort of image classification training model, and then they lower the accuracy of the numbers that were used to represent the different — The weights on that model, and maybe the model was still good enough despite reducing the accuracy. Are there any recent examples that come to mind?

[0:28:08.1] XW: Yeah. This paper I'm going to talk about today is basically like flex point, 16-bit flex point. Basically, this is a new numerical format that we invented, and this is exactly what we did. So this is a technology actually underlying the Intel Nervana neural network processor, and basically we did an experiment by using this particular low precision format in 16-bit and compared to, basically, the most commonly used 32-bit single precision floating-point that's widely used in training today, and we basically showed tensor by tensor along every step of a training process that it actually matches pretty well in deep learning, like a training process of big networks.

[0:28:58.7] JM: What do you mean by that? Can you explained that in a little more detail when you say it matches?

[0:29:03.4] XW: Basically, we look at several metrics. Number one, we can just pull out certain tensors at a certain stage of training and see how it matches numerically, like percentage of error. Also, we can train a network and use the network to do inference, basically like prediction of a certain class labels, if it's a classifier. We also trained like generated models. We can generate data and use other metrics to evaluate how well the network performs. In both cases, we see a pretty good matching performance.

[0:29:41.4] JM: Let's go lower level. So one way that we can improve this lower level efficiency is by reducing the amount of space that each number takes up. So this is what you are saying with accuracy. We've got 8-bit integers, 16-bit integers, 64-bit managers, we have doubles. What are the numerical formats that we have historically used to represent our tensors?

[0:30:16.6] XW: Typically, right now, a lot of the training are done on single precision floating-point. Basically, these are 32-bit floating-point numbers. You mentioned one thing that's absolutely important is the number bits. So the more bits you have, the higher precision you have, basically. In integer case, it basically means you have more integers you could represent. So there are two notions that are important here, its dynamic range and precision. Precision, I just explained. Dynamic range is basically determined by the span between the smallest and the largest number you can encode.

In the integer case, it's pretty much the same as your precision, because there's no kind of another value that tells you how to scale it. For floating-point, you also have an exponent within the total number of bits. For example, in a 16-bit floating-point, you have five bits of exponent, and basically this is like a scientific notation. So you have an exponent actually tells you how to put a radix point, if you may. It basically moves along the scale in the log axis of your values and basically gives you kind of the logarithmic scaling. So it will widen your range of encoding by a lot with the same number. That's why floating-points are widely used in scientific computing, and typically we'll have single precision, double precision in our general scientific computing, and you have single precision mostly for graphics. That's how we kind of inherited in the GPU for our deep learning training.

[0:32:03.4] JM: Each number of formats has a dynamic range and a precision. Explain what these terms mean. What is dynamic range and what is precision?

[0:32:16.3] XW: Basically, dynamic range is the span between the smallest and the largest absolute value that you can represent in that format, and precision basically means how many representable numbers are there within the dynamic range. Basically, for a specific, let's say, floating-point number, the dynamic range is pretty much going to be determined by the exponent, the range of your exponent and the precision is basically the number of bits that you use for the significant, and it's called mantissa in the floating-point.

[0:32:54.6] JM: When we're using a number format, that does not effectively capture the precision in a number. If we have a number that is actually much, much longer than would fit in the number of bits that we have available, we need to quantize it. Is there anything interesting to say about quantization? When I think of quantizing I think of just rounding up or down. Is quantizing anything more interesting than that?

[0:33:24.9] XW: Right. There are basically likes three different types of quantization errors. You just mentioned one. So if within your encoding range you basically round up or down a specific number, you incur a rounding error. But if you have a very large number that you cannot represent by the largest representable number, then this is called overflow, and also there is a corresponding underflow. Basically, a very small number, that's smaller than the smallest number you can represent, then it's going to be basically cut to zero. All three types of quantization errors may contribute to loss of performance in the deep learning case.

[0:34:08.7] JM: Now, just take a step back here. Some people listening may not know the connection between hardware efficiency and number formats. Why does it matter that we are discussing how to get more efficiency out of our numerical representation. How is that going to translate to better performance of these deep learning models at the hardware level if all we're doing is reducing the amount of space it takes to represent these numbers?

[0:34:46.3] XW: Indeed. The representation, basically, the quantization of numbers is not just the amount of space each number takes. We basically do computations on these numbers. The operations are basically the cost of computation. Typically, the computation is realized by specific logic on the hardware, certain arithmetic.

There're two trends that's very important. Number one, it's pretty obvious. So if you have a lower precision, fewer bits, then you use, basically, fewer transistors to do the logic and it will result in a smaller footprint on your silicon and it consumes less power and the takes less time also. The second trend, which is also very important may not be so intuitive, is integer arithmetic actually is cheaper than floating-point arithmetic at a same bit width. Basically, you don't need to kind of separate — Basically, the integer multiplications can be very efficient.

In a specific case, let's say, if we go from the current training standard of floating-points, 32-bit to use in neural network training down to 16.6, 16-bit, then the floating-point 16-bit, the half precision will be a little more expensive than pure integers in 16-bit.

[SPONSOR MESSAGE]

[0:36:21.1] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers because I talked to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many newer hungry software engineers who are looking to level up quickly and prove themselves, and to find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineering.com.

If you're listening to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company. So send me an email at jeff@softwarengineeringdaily.com.

Thank you.

[INTERVIEW CONTINUED]

[0:37:49.3] JM: So when we lower the precision of — Are we talking about lowering the precision of the weight? Is that specifically the type of number that we are lowering the precision of?

[0:38:00.7] XW: Right. In the current work that we published, we are basically trying to make it the general case. So it’s one level lower than the algorithm or the network itself. We don't care

about what type of tensor it is. For any tensor, we basically want to have a low precision representation of the values in it.

[0:38:21.9] JM: Literally, all matrix operations that are occurring at the lowest level, you're just taking those operations and making them less precise, and from the results that you've seen, it has not been problematic. They have been “unreasonably effective” Is that right?

[0:38:45.6] XW: Right. Specifically, for inference, you do see a lot of works, reporting that if you even quantize the values very, very aggressively to even binary and the network still performs pretty well. But that’s for inference. It’s probably not the case for many cases when you do training. Actually, indeed, a lot of people found that you will have a significant hit, take a hit on your performance of training when you quantize numbers very aggressively even down to 16-bit.

Just as we mentioned that if you kind of do some extra engineering to quantize aggressively for certain tensors in your network and leave others in high precision, and typically for activations and gradients, we can probably use a pretty low precision, but when you accumulate it into a changes of ways and these things are — Because of the continuous nature in the underlying math, you probably need to accumulate it in a pretty high precision. This is actually something that you can potentially do to figure out the optimal allocation of different positions for different parts of your network and trying to cut down the cost at certain parts, but not others.

Here, what we are trying to achieve with 16-bit flex point is to have kind of a universal down quantization to 16-bit. In this case, basically, you don't even need a care about doing extra engineering doing this allocation.

[0:40:27.0] JM: Do we have any ideas why it's okay to lose precision at the inference level, but it's not okay to lose precision at the training level? Do we have any deterministic understanding of that or is this just based off of experimental data?

[0:40:44.5] XW: Right. This came as a surprise to many people when you actually very aggressively quantize like weights and activations in a network. Some binary networks just taken from a trained network with full precision actually works pretty well, and this is really kind

of puzzling. I think there's an interesting theoretical question. A lot of people are actually researching on that and we'll see some pretty recent results giving us hints on why this actually works. The same reason I think back propagation of gradients probably should be the same and given the similarity of this. I think the biggest problem is for training, and the training doesn't work in low precision is pretty much because of the third step, which is updating the weights. It's basically based on a gradient-based optimization procedure.

The gradient basically requires a continuous and differentiable structure of the functions defined on that. Typically, in many cases, when you represent a number even in the16-bit, if you are encoding range of your weight update does not overlap at all with the weight itself, then you don't end up doing anything there. So this is always a problem.

For now, in this particular paper, we're ere not investigating in those very aggressive quantization schemes, let's say lower than 16-bit. We're are only sitting at a 16-bit and ask the question whether we can use integer arithmetic and integer formats on our accelerator processors to make training work.

[0:42:33.3] JM: We've now explored some of the trade-offs between the different number systems when we're talking about representing our tensors, and we can now get into flex point, which is a new number format that you created. Explain what flex point is.

[0:42:51.9] XW: Basically, flex point is just a kind of data structure wrapping around an integer tensor. Basically, a flex point tensor is an integer tensor in essence. Every element in this tensor is an integer. So in this particular case of this paper, we studied the 16-bit in this particular, then every element in this tensor is in 16-bit integer, because a low precision integer has a very limited dynamic range. In order to make it more like a floating-point, we need to be able to freely scale it, and it turned out to be necessary for deep learning training, because for many different tensors, they have different dynamic ranges, and a lot of tensors, especially like gradients, naturally, it kind of decays during the course of training. So the dynamic range also needed to be dynamically adjusted.

In addition to the integer tensor, it has an externally managed exponent. It's just like an exponent in floating-point, but it's shared across all the integer elements of the tensor. In

addition to that, we also have some other data structures associated with each tensor that gives you a kind of recorded history, recent history of the scale of the tensor so that the optimization, the prediction algorithms of tensor scale can be carried out.

[0:44:19.3] JM: Explain that name, flex point. Where does that come from?

[0:44:23.2] XW: Okay. I don't know exactly where — I think maybe flex comes from flexible. I think it's basically a very similar idea to what is in the literature called blocks, block fixed point. It's basically a fixed point format. So all the elements inside a tensor is am integer, but it's in tensor, so it's a kind of a of block that share an external exponent. So that's basically the main difference.

[0:44:58.2] JM: Why is flex point so useful and flexible for these deep learning operations that we're talking about?

[0:45:07.3] XW: The first question is whether the deep learning training workloads can still be carried out in 16-bit floating-point, and the answer is no. If you just naïvely just translate all the — Cast all the numbers into a floating-point, 16-bit for every element. Usually, for a lot of cases, you take a hit in your performance.

Number two question is whether we can basically use a 16-bit integer. I shall remind people here that a kind of a 16-bit integer actually has all the 16-bits for its mantissa. So it kind of has a higher precision than the 11-bits mantissa for the floating-point. Also, if you just do a fixed point kind of 16-bit training for deep neural networks, it also doesn't work unless you have kind of a dynamic kind of — Or tensor by tensor exponent management.

Basically, that's why fixed flex point works well. It's basically just taken the advantages of both fixed point and floating-point. It has a very efficient representation, like the 16-bit integers for every element, but at the same time, for every tensor as a whole, it can kind of slide its dynamic range along the scale like a floating-point.

[0:46:40.5] JM: And you get these hardware advantages, but there is the cost of this added complexity of managing an exponent. Can you explain what that means? What is it mean to have to manage an exponent at this numerical representation level?

[0:46:57.3] XW: This is basically an added complexity, because for every integer tensor in flex point, they share — All the integers share an exponent, and this exponent is external. Basically, that means as far as your accelerator hardware is concerned, every tensor is an integer tensor, but the host computing device will keep track of this external exponent, and it needs to be managed, because the exponent should change to adjust the dynamic range when the scale of the tensor changes across different elements of your network and each value in your network may also change dramatically during the course of training. This is where the exponent management comes in. The exponent management actually is done independent from the accelerator hardware. It's on a host.

[0:47:52.5] JM: When you were designing the system, can you describe the process of testing it?

[0:47:59.0] XW: Basically, for testing, typically you need to establish a benchmark. So in this particular case, it's pretty easy. We basically compare with a 32-bit floating-point and 16-bit floating-point, and, basically, single precision and half precision. Number one, single precision is de facto standard right here, so most of the network are trained in single precision, and actually if you take any recent papers of the new network, all the results reported are pretty there. So it's a very natural thing to compare to.

The second is in order to go down to 16 bit, we can also try to have a floating, half precision 16-bit floating-point number for every element at the tensor, and basically you we can just compare these against floating-point 32-bit, and our result basically showed that in a few cases, we showed three cases, these are all convolutional nets and we found that in all of the cases, a flex point, a 16-bit, actually worked pretty well given the algorithm, what we call the auto-flex. Basically, this is the algorithm we managed to the exponent, while if you have kind of naïve casting into floating-point 16-bit, in many cases it doesn't faithfully match the original training process.

[0:49:22.9] JM: And what's the plan for rolling out this technology?

[0:49:27.4] XW: Basically, this is the underlying technology of our NNP, and I think this is an interesting trend that we see in new hardware, deep learning hardware design for low precision, especially in training, which is that we basically have kind of a co-design of your algorithm. In this particular case, it's really not the algorithm of the network itself, but some kind of low-level numerical algorithm that's detached from the operations on a hardware itself.

Basically, we use as cheap as possible operations on hardware. In this case, a 16-bit integer arithmetic on hardware. As far as the hardware is concerned, all the numbers that's being crunched inside a tensor are integers, and these are really expensive operations traditionally, because every element of the tensor have to undergo this this operation, but we kind of offload other, what we call exponent management here, operations on to the host and these operations actually do not scale with the dimensionality of tensors and actually it's constant for each tensor.

For typical problems, like training the deep neural network where you have huge tensors, this external management overhead is actually pretty negligible. I would predict that this kind of design will be pretty powerful and popular in the future so that we have accelerators that take care of very dense operations in a very low precision, but externally you have some kind of operations that does not scale with dimensionalities of the problem that can be effectively managed in a software in an algorithm.

[0:51:22.5] JM: Okay. I think we've talked at length about this hardware advancement. I want to zoom out. Once again, we talked a little bit earlier about NIPS and you've mentioned AlphaGo Zero. What was exciting about AlphaGo Zero for you? So my understanding is that this was a — They generalized AlphaGo, which was a system that learned to play Go and beat all of the humans and they have generalized it to work for any system?

[0:51:54.7] XW: Right. I think this is actually a very good point, and I think every year at NIPS you see some kind of new theme emerging, and I think this year there are actually quite a few. I think the most impressive thing for me is to see how meta-learning is taking off, and this is basically kind of learning to learn, if you may. So I think in the past few years, all the very exciting papers that you see that achieved like beats human level performance, for example in

image classification, like RES.NET, there are all like engineering on this structure of the network itself, also tuning the parameters, hyperparameters. These are also like traditional kind of engineering of the network topology. It's just like back in many years ago that in computer vision people are using handcrafted features, where basically now using networks, but we're kind of handcrafting hyperparameters, handcrafting topologies. It still kind of requires a lot of data to train.

For meta-learning, you can actually use very few data points and you can actually come up in a very principled way network structures and its hyperparameters and also combined with a lot of traditional machine learning techniques together with deep neural networks. We see a lot of schemes are actually very useful. I think in deep reinforcement learning, I think AlphaGo Zero is basically a kind of a very extreme case and you basically don't need any data to train it. All you need is the rule of the game.

[0:53:48.5] JM: Okay. Well, just to close off, I know that you spent much of your career in biology actually, and the earliest applications of deep learning to biology that I have seen are based around image detection. So you have things like people building systems that can diagnose diabetic retinopathy or heart disease or skin cancer. Given your experience and your time at NIPS, what's going to be the next wave of deep learning applications within health?

[0:54:23.8] XW: I'm basically trained as a neuroscientist, a computational neuroscientist, and it's actually — You can say a lot of deep learning deep neural network ideas actually stems from neuroscience at the beginning, but this is really a kind of abstract mathematical idea, and we actually see in the field of neuroscience that modern data science and machine learning has actually helped a lot, because the field of neuroscience itself is very data intense and these days experimenters are getting a lot of data out of the nervous system, the brain itself.

I think a lot of data and using techniques like deep learning actually really helped a lot in understanding the brain. For the future, I would like to see actually — I think it's going to happen. It happens — The influence going in the other direction again. So just like originally we have the abstract artificial neural inspired from a biological principles. Maybe a high level understanding of how the brain works also influence machine learning. Again, I think this is

probably going to be a very interesting and very high impact revolutionary change in the future if it happens.

[0:55:46.4] JM: Hopefully it'll be powered by Flexpoint.

[0:55:48.3] XW: I'm not very sure about that, but I will certainly hope.

[0:55:51.7] JM: We'll see.

[0:55:51.9] XW: A lot of hardware innovations is going to help in this way.

[0:55:56.1] JM: Okay. All right, Xin. Well, thank you for coming on Software Engineering Daily. It's been great talking to you about the recent advances in deep learning.

[0:56:03.3] XW: My pleasure. Thank you, Jeff.

[END OF INTERVIEW]

[0:56:07.5] JM: GoCD is an open source continuous delivery server built by ThoughtWorks. GoCD provides continuous delivery out of the box with its built-in pipelines, advanced traceability and value stream visualization. With GoCD you can easily model, orchestrate and visualize complex workflows from end-to-end. GoCD supports modern infrastructure with elastic, on-demand agents and cloud deployments. The plugin ecosystem ensures that GoCD will work well within your own unique environment.

To learn more about GoCD, visit gocd.org/sedaily. That's gocd.org/sedaily. It's free to use and there's professional support and enterprise add-ons that are available from ThoughtWorks. You can find it at gocd.org/sedaily.

If you want to hear more about GoCD and the other projects that ThoughtWorks is working on, listen back to our old episodes with the ThoughtWorks team who have built the product. You can search for ThoughtWorks on Software Engineering Daily.

Thanks to ThoughtWorks for continuing to sponsor Software Engineering Daily and for building GoCD.

[END]