EPISODE 80

[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.

We've got a special treat for you with this show. This episode is actually part of the OpenAI series of shows we ran last week, but because it features hot off the press's research, we weren't able to release it with those shows. But your wait is now over.

This episode features an interview with Durk Kingma, a research scientist at OpenAI. Although Durk is probably best known for his pioneering work on variational auto-encoders, he joined me this time to talk through his latest project on block sparse kernels, which OpenAI just published this week.

Block sparsity is a property of certain neural network representations. OpenAI's work on developing on block sparse kernels helps make it more computationally efficient to take advantage of it. In addition to covering block sparse kernels themselves and the background required to understand them, we also discuss why they're important and walk through some examples of how they can be used.

I'm happy to present another fine nerd alert show to close out this Open AI series and I know you'll enjoy it. Support for our Open AI series is brought to you by our friends at Nvidia, a company which is also a supporter of Open AI itself.

If you're listening to this podcast, you already know about Nvidia and all the great things they're doing to support advancements in AI research and practice. As you'll hear in this show, Nvidia's DGX1 deep learning super computer was instrumental to Durk and his team's work on the block sparse kernels project.

If you happen to be at NIPS this week, be sure to check out Nvidia's booth at the expo, as well as the four accepted papers being presented by their team. To learn more about the Nvidia presence at NIPS, head on over to twimlai.com/nvidia.

Now, on to the show.

[INTERVIEW]

[0:02:20.0] SC: All right, everyone. I am on the line with Durk Kingma. Durk is a research scientist with Open AI. Durk, welcome to This Week in Machine Learning and AI.

[0:02:29.7] DK: Thank you very much. It's my pleasure.

[0:02:31.4] SC: It's great to have you on the show. I am looking forward to learning a bit about your background and what you're working on. Why don't we start with the first of those and have you tell us a little bit about how you got involved in machine learning and AI research.

[0:02:46.0]DK: Sure. I started, I think my first research position when I was doing my master's degree, I e-mailed Yann LeCun out of the blue with a couple of research ideas. I had no clue where he was. He responded and like he was welcoming me into his lab for six months, so I looked up and knew where he was, it turned out to be New York. I was very happy with that, of course. That was 2009.

[0:03:13.0] SC: Was that for a post doc – or not a post doc, but a research position?

[0:03:18.0] DK: I was very interested in AI. I did the master's program in Utrecht in the Netherlands. A friend of mine was studying in Cornell and he got me interested in a couple of topics. I spend about a year reading about the various ideas in the field, and I quickly discovered that Yann LeCun had a lot of interesting research going on. I applied for a position in his lab and was accepted happily. This was about three years before deep learning resolution started. I had a great time there. Yeah, that was 2009; I published a paper with him.

I decided to do a startup afterwards with a friend of mine in the Netherlands. We did that for a couple of years, and then I decided to go back to research and I spend again about over a year in Yann LeCun's lab and applied for a PhD position with Max Welling. He's a professor in – who just started in Amsterdam. Before, he was a professor at Irvine.

My girlfriend at the time was still studying in the Netherlands, so I preferred to have a position there. I was very lucky to get a position with Max Welling in 2013. I was his first PhD student there. Basically, when you start as a professor you get a free student as your first student. I was

the free student. Because of that, I was also lucky to not be bound to a particular research, direction or topic. I had relative freedom to pursue my interests from the start.

What I was interested in was combining probabilistic models with deep learning. I thought this was a field that was under-appreciated, the combination. First paper I published with Max Welling was on the variational autoencoder, which is a paper that introduces a way to combine Bayesian inference in a natural way with deep learning.

This method allows you to train generative models in a principal way and a scalable way. Skills, you have two large data sets and two large models. Basically that were the key advantages.

[0:05:45.8] SC: Okay. We've talked about variational autoencoders on the podcast a few times, but I'm looking forward to digging into that topic with you. What else did you work on there?

[0:05:59.4] DK: right. Variational autoencoder, that's part a wave of various application papers and extensions also in our lab. We worked on trying to get uncertainty estimation work well in deep learning models. As you know, most deep learning models all you do is basically learn a single value of the parameters that we need to in a good prediction under test that and your training set.

But it doesn't necessarily give you a good estimate of how certain you should be about your predictions given your limited data. The combination with variational inference is a potential way to achieve that. Debate with variational autoencoders introduced a trick called reparameterization trick, which was also applicable to estimating a posterior distribution over the parameters, which is basically gives you a quantitative estimate of the uncertainty over your parameters given your training data. We did some paper on that. Yeah, so that was one extension we did.

Also, because of the variational autoencoder paper, there was a similar research direction within the lab of deep mind that happened about at the same time. I happen to publish my paper a bit earlier, but so they developed independently a very similar trick. Because of this coincidence, we started also collaborating on this topic. I went to deep mind in 2014 and collaborated there on applying this method to the problem of semi-supervised learning, which means that if you want to train a classifier from training data, but not all your training images are labeled, then you

would want to make use of the unlabeled data to get better predictions that what you would get if you would solo be trained on the labeled images.

Yeah, so you basically proposed a method using the variational autoencoder. The results that we got at the time were state of the art, but of course were quickly eclipsed by a wave of auto-papers. That's all right.

[0:08:29.1] SC: How it happens, right?

[0:08:30.0] DK: Yeah. It's definitely how it happens.

[0:08:32.2] SC: I mean, the variational autoencoder, I've heard that come up in so many different context. I mean, you mentioned that even in your prior lab, the initial paper was followed up by a bunch of applications, papers that you worked on, but a lot of folks have worked on papers using that, papers and methods I should say.

I think one of the first places I saw it was in the context of like generative models for text, is that a popular – am I correct in that? Is that a popular place to use variational autoencoders? Or maybe it was – there was another one where someone took a – I think they took a bunch of like movie videos or movie clips or something like that and ran them through a variational autoencoder and tried to generate a movie clip, which was interesting as well.

[0:09:25.2] DK: Before we dive into variational autoencoders, we haven't gotten yet to OpenAI, have we?

[0:09:31.8] SC: Right. In the end of – or I think somewhere mid-2015, I was approached by Greg Rodman, who was assembling an initial team for OpenAI. To me it was very intriguing, because yeah, it mentioned very well with my own philosophy that I think it's good to publish and to write open source code for your experiments, but also to put emphasis on maximizing the capability of a positive future with AI.

Yeah, there is a course currently – a lot of concentration of talent in the field at a small number of places. I thought it would be good to have more of an – to perhaps safeguard the opens of the field and the collaboration.

Yeah, there is – because of the inflow of talents into commercial labs, there is a risk that eventually a lot of these labs will – you close up. The benefits of the field might not be as evenly distributed as we would like. That is one of the goal of OpenAI, to make sure that the benefits are distributed eventually in a fair way. Also, it was of course located in California, which is not a bad place to live.

[0:11:10.0] SC: Not at all. Not at all. You've been at OpenAI for how long?

[0:11:13.0] DK: Since the start basically. The first few months I was still working from Amsterdam, and then I moved here in 2016. Yup.

[0:11:24.8] SC: Okay. Awesome. Awesome. Well one of the things that I wanted to ask you given that we're recording this the week before NIPS and I am going to NIPS actually for the first time is I'm assuming NIPS is like your home conference. You've been there many times. I wanted to get a sense from you both if there's anything in particular you're looking for to about this particular conference, or any tips on approaching NIPS as this become quite a large conference.

[0:12:00.1] DK: Yeah. This has indeed become an enormous conference. As you probably know, the field is still more or less doubling every year. I think the attendants of NIPS follows a similar trend. Personally for me, it's also a big social event. Of course, it's probably the biggest conference in our field now.

Old friends and old colleagues from all over the world, assemble in a single place. This is a great moment to discuss collaborations or to discuss the newest results and either be or together, etc. In terms of preparation, would recommend people to read just through the agenda and make short of a list of papers they think are most interesting before they go to the conference. Because whenever you're there, it is overwhelming. You have notes on and to orient.

[0:12:59.6] SC: Even the agenda is overwhelming, to be honest.

[0:13:03.4] DK: Yeah.

[0:13:04.1] SC: There is a ton of stuff going on there.

[0:13:06.1]DK: Yeah, absolutely. Of course, I think an interesting thing is that a lot of the work that you will see is already published and archived, right? As soon as you see a paper, you can actually already read it long before the paper has been published on the NIPS website. Often, follow-up work has already followed their archives. Some papers are already outdated. At the moment you see them at NIPS which is a good tragic.

[0:13:38.5] SC: That's funny. You're publishing some work next week as well?

[0:13:43.8] DK: That's right. Yeah. With two of my colleagues, we worked on publishing a set of new kernels, the GPU kernels which is a software for the GPU, which allow you to train and build models with block sparse layers. Block sparse GPU kernels.

[0:14:05.5] SC: Maybe talk to me before we dig into the block sparse element of this. Tell me a little more about GPU kernels and where they fit in the software stack, like I think about CUDA, I think about – at the lowest level I think about frameworks like Tensorflow and things like that at a much higher level. Where do kernels fit in?

[0:14:29.2] DK: Right. GPU kernels are – you can view them as middleware. They are libraries that require you to program on the very low level. They allow you to maximize usage of GPUs, which are of course it is hardware that runs very fairly low. So it regards in specialized software to make full use of that.

Basically, they are a set of middleware that is typically already implemented in your framework. You basically call various functions that make use of these kernels when you build your models. For example when you implement the convolutional network in Tensorflow, then you typically use various – pre-built layers that themselves call GPU kernels to efficiently evaluate the forward pass and the network pass and the gradient's computation on the GPUs.

[0:15:37.8] SC: Are you developing the kernels in CUDA, or at a lower level than that?

[0:15:42.9] DK: Right. I did not develop the kernels myself. This was all done by Scott Gray, who is a GPU kernel expert. He both give route to all of the kernels, so all the credits go to him.

[0:15:58.4] SC: Right. I guess, I was more asking like – or I guess I should have asked, does one develop kernels in CUDA, or is this at an even lower level than something like a CUDA? I'm just trying to get a sense for where they fit in.

[0:16:14.0] DK: These kernels were developed both at the C level and at the assembly level.

[0:16:21.0] SC: Fairly low level.

[0:16:23.1] DK: Yeah.

[0:16:24.7] SC: Then you mentioned, brought up convolutional neural nets as an example. Can you give some examples of the types of kernels that a framework might implement? Is this for things like tensor operations, or are they at a higher level or lower level than that?

[0:16:40.4] DK: Your question is like how you can use these kernels in your models?

[0:16:44.8] SC: No. I guess, I'm still talking about GPU kernels generally and what operations they tend to represent in for example the case of the CNN and a framework that is calling down to a bunch of lower level kernels. I'm just looking for examples of what those kernels might be.

[0:17:04.0] DK: Right. In a typical CNN, you have a GPU kernel for – the foreword is convolutional computation, so you have a convolutional layer and a convolutional network, which is basically a linear layer that applies like a convolution or the inputs and the result is the output of layer. That's where you use GPU kernels, and then there are for example GPU kernels for element-wise operations. Yeah.

[0:17:34.1] SC: Okay. Got it. The kernels that you are publishing are for block sparse operations. What are those?

[0:17:44.5] DK: Block sparse operations, so what we release is it is a generalization of the usual matrix multiplication and convolutional kernels you typically use. Typically, what you have is when you use a convolutional layer in your model is a weight matrix, or like a weight tensor, which is debts. Meaning that all the entries in the weight in tensor or weight matrix have a no-zero value. The thing is that as you increase the width of the network, the number of weights increases quadratically.

[0:18:29.6] SC: By width, you meant the number of features?

[0:18:32.8] DK: The number of features. Right. Right. There is a quadratic relationship between the number of features and the number of weights, if you use a dense new kernel, like a dense weight matrix. An obvious solution to this is to instead of using a dense weight matrix, you use a weight matrix where not necessarily all blocks, or all weights in the tensor have a no-zero value. What the block sparse kernels allow you to do is to define which of – you basically divide your weight matrix, or your weight tensor into blocks of either 8 by 8, or 16 by 16, or 32 by 32.

Then you can say beforehand for each block, whether that block has value of zero for every entry in that block, or actually has an actual learned weight value. Yeah, so basically if the weight values are zero, which is equivalent to having a block that is all zero, then you don't have to compute the block, because the output of the layer of the operation is not a function of that block, so you can skip that computation.

[0:19:52.8] SC: You would use this in a scenario where – or maybe I should just ask you to give us some examples. I'm imagining that what you're essentially saying by using a block sparse matrix is that you don't care about the relationship between some features and some layers. Is that the idea?

[0:20:12.1] DK: It gives you more flexibility in choosing how your neurons are connected between layers. Basically in all the existing architectures given a particular layer, all the input and output features are connected. But this is not necessarily the optimal way to use your parameters, so given a particular budget of parameters, you might want to use for example a wider network, but where not all features are interconnected, but only half of them are interconnected, or 25% of them.

Or you might want to say that, "Okay, I'm going to just use interconnecting half of the neurons with each other and I'm going to increase the depth by a factor of two." Basically by introducing sparsity, given the parameter budget you can have either much wider or much deeper networks. This is one particular application that we investigated, but there are much more applications possible that we haven't even touched upon.

There is a whole – like a whole series of papers that are coming out now that show that after training a neural network, it is typically possible to remove 99% of the waste that are

significantly affecting performance, which is in the security is finding. It turns out that most of the weight are useless for prediction.

These kernels, they will allow you to actually make use of these redundancy. After training, you could potentially still not move all the weights that are useless inside a neural network. It is much faster to evaluate. It would give you a speed up and future work would also involve learning the connectivity. This is something we have not done yet, but we hope that we or other will do in future work.

Basically a bit similar to the brain, your neurons should be connected, which should not be connected. Basically this is the fore of learning the structure of the model beyond just the value of the weights. You could also learn where you have weights. Yeah, we also have some preliminary work that we did with Crystals with us, an internet with here to summer.

[0:22:40.0] SC: Okay. Right now you are saying before training, you're specifying like where the sparsity blocks are, is that correct?

[0:22:51.2] DK: Yeah.

[0:22:52.7] SC: You're suggesting that there is work – you're hoping to see work, where that's learned as part of the training process as opposed to being specified upfront?

[0:23:03.8] DK: Yeah. This is something where we have some preliminary work. But I think this is a very exciting direction that we or others can pursue in future work.

[0:23:15.8] SC: How do you determine where the sparse blocks are currently, or conversely what were the connections are? Is it essentially another kind of hyper parameter or meta-hyperparameter or something that you are training and evaluating with regard to some optimization that you're trying to make or the performance of your network as a whole? Or is there some set of heuristics or intuition that tells you where you should have the connections.

[0:23:53.2] DK: Right. In choosing the sparsity better in our published work, we took inspiration from the field of small world networks.

[0:24:03.1] SC: Small world networks. What are those?

[0:24:04.4] DK: Yeah. Small world networks, they are a type of graph basically that you find in various systems including to bring. You also find it in social networks. You or any person on earth is connected to any other person on earth in a small number of steps.

[0:24:24.6] SC: Right. Six Degrees of Kevin Bacon.

[0:24:27.5] DK: Right. This means that even though the number of connections you have and the number of connections that any person in the world has is relatively low, you are still connected to any other person in a small number of steps. You find the same property in the human brain at a functional level at least. The functional modulus in the brain are often mostly locally connected, but there are some more or less random one range connections, and due to these more or less long range long-term connections, the whole brain is connected in these small number of steps, which means that information is spread throughout the brain relatively quickly, even though you have a huge number of neurons.

This is something we took inspiration from when choosing the sparsity models in our networks. There are very simple algorithms for generating graphs that have this property. These are called small world graphs. They just require you to have a certain percentage of your connections to be random. This is indeed a hyperparameter, but we do have the guarantee that information mixes in the route of these small number of steps.

Even though you introduce sparsity in your network – for example, we have trained a big LSDM that is 98% sparse, which means that 98% of the connections between type sense are now there. Any neuron is only connected to about in a 2% of the neurons in the previous timestamp.

Even though it's only 2% in a small number of steps, every neuron is connected with every other neuron. Which means that we basically do is we product a number of internal steps in the network, between external blind steps that allow information to spread through the whole network before you process a new input.

[0:26:36.9] SC: Can you say that last part again?

[0:26:39.2] DK: Right. Because of the small world property, we basically only need to introduce a small number of intermediate steps between inputs to basically have network that is fully connected. After receiving input in that step, you want the [inaudible 0:27:01.4] brain to basically

integrate that information across all neurons. Because of the small property, you only need about five steps of the internal blank steps in order to make sure that the information is spread through your whole network.

[0:27:25.3] SC: The time steps are property of LSTMs. Does this apply to other types of models as well, like CNNs?

[0:27:35.7] DK: Yes, definitely. We applied the same technique to convolutional kernels. In this case, we introduce sparsity in the future dimensions of additional kernels. We also found there that it would need help to get better performance. Yeah, we've done a couple of experiments. One is on –

Basically, what we showed is that if you take an existing architecture with an existing convolutional kernel, so you take like a ResNet, or the Pixel CNN and you replace your regular convolutional kernels with block sparse kernels, then you either widen or deepen the network while keeping the number of parameters the same, you get better performance in many situations.

[0:28:29.9] SC: What's the analog to introducing additional time steps in the LSTM and the convolutional network?

[0:28:39.0] DK: Right. In the convolutional network, you can for example in a ResNet what worked well is that we simply doubled the depth of the ResNet. A ResNet consists of a couple of stages. Between stages you down sample. In one experiment, we took a ResNet for even [inaudible 0:29:00.6] and we doubled the depth of network and we introduced 50% sparsity, or 50% of the weights are not there anymore, which means that the total number of parameters is approximately the same as before. Because of the additional depth, you still have a good mixing of information. We kept the rest of the architecture the same. The same loading rates, the same hyperparameters, we kept everything fixed and we saw that it led to an improvement of the accuracy of the model.

[0:29:38.7] SC: The small world model, that's giving you an algorithm that you can apply to determine which 50% of the data you're basically getting rid of and which you're keeping? Or is it giving you some kind of bound or guarantee that if you get rid of X percent of information,

you'll still have the number of – given degrees of connectivity, or convergence or something? What exactly is that telling you?

[0:30:07.5] DK: We're not actually getting rid of any data, because – the state is still dense of the model. All we're doing is introducing sparsity into weights. We're not removing any information from the state.

[0:30:25.6] SC: While you're not getting rid of data strictly speaking, you're still getting rid of data in motion in a sense, like you're making it harder for the network to learn, or to get a piece of data in a given step, right?

[0:30:41.6] DK: You mean that the number of weights per step is reduced by a factor two, so the number of parameters is reduced a factor two.

[0:30:49.7] SC: I guess, what I mean is that ultimately when you do this, the network is still operating on less information than it had in a fully connected sense. Maybe before you do – before you do things like add time steps and brought in or deepen your network, just if you were to have a fully connected network and then you zero out some of the weights, like the network is then – Is it fair to say that the network is operating on less information than in the fully connected sense?

[0:31:30.7] DK: Yeah. It is important to know that this is something we do before training, right? We actually train with sparsity. Yeah, so before you deepen or lighten, you remove hold weights, then indeed you do simply have halt to capacity, just do anything in your weights obviously.

[0:31:50.3] SC: Right, right. It is important I think to keep the number of parameters at least equal. It's just basically you are assigning your parameters in a different way through the model. The model has still in principle the same capacity to store information. It's just that the way it uses the parameters is a little bit different.

[0:32:14.8] SC: Just to summarize that, you are indeed reducing the model's capacity to store information when you remove half let's say of the weights, but you're compensating for that by either increasing your breadth or your depth.

[0:32:30.9] DK: Yeah. That's what we are to be doing. Yeah. Then in the future work, I have believed that we can figure out how to learn the sparsity and actually doing training be able to remove a large percentage of the weights without me getting worst performance. That's the vision.

[0:32:55.8] SC: How sensitive are – in the case where I think with the LSTM, you said you got rid of 98% of the weights. I would imagine that which weight you get rid of is very important, or conversely which weight you keep are very important. Do you have some way of measuring the sensitivity of a given network's performance to which weight you remove?

[0:33:22.4] DK: We choose a particular sparsity better before training. Then we use either a wider or a deeper network. Then we train the weights, so the model has to figure like what meaning to assign to each neuron and to each weights, right? We leave it up to the model to use the given sparsity better.

What we found that worked well in case of LSTM is the so called Barabási Albert graph, which is a type of small world network where you have a small number of neurons that are connected to a very large number of other neurons, and then you have a long till of neurons that are very sparsely connected. We found that this works really well for text. What we did in this case is that we increased the size of the states of the LSTM, so you have a much wider model, which it also gets you a longer memory of the past as you get to fit more information into the state of the LSTM then otherwise.

What we found is that if we create a sentiment classifier based on that state – so we first train an LSTM completely unsupervised on text, and then we train a linear classifier on the state of the LSTM to predict the sentiment of reviews.

What we found is that we get state of the art results in predicting sentiment based on that model. The previous [inaudible 0:35:06.4] was published by Alek Redford, which is also co-author of this paper a couple of months ago. Now we found that if you train a much wider network at sparse, you get even better results. This basically you gave us state of the art results on library five benchmarks of classifying sentiment in text.

[0:35:29.9] SC: Right. This is an example of how within the same parameter budget reconfiguring the way you use that parameter budget can give you much better results.

[0:35:40.5] DK: Exactly.

[0:35:42.2] SC: Is there any intuition as to where you're likely to see that effect, or where you would likely want to apply block sparse kernels, or is it something that will just have come out of experimentation?

[0:35:56.0] DK: Yeah. Even the space of possible architectures that you can train with these kernels is so huge that there's no way that we can explore all. Let's see, what we aim – what is released is to basically give it to the world and let everyone experiment with it, because we are waiting small to explore this full space. Yeah, I have some limited intuition.

[0:36:25.7] SC: What is that? What's your intuition telling you that where might folks want to look first, or where would you like to see folks looking to apply these?

[0:36:35.0] DK: Right. Scott is building in the capability now of actually having like a dynamic sparsity. Hopefully this will be finished before the release. We will see.

[0:36:47.1] SC: Meaning that it varies during training?

[0:36:49.5] DK: Yes, so –

[0:36:50.3] SC: Interesting.

[0:36:50.9] DK: - we can actually learn the sparsity mask during training, and you could then potentially optimize it to get more performance. For example, in us humans it's we know that way our own neurons are interconnected is also learned based on data. Yeah, so this is a way of learning the architecture of your model. Personally, I think this is going to be a very interesting area of research.

[0:37:18.3] SC: Yeah. Maybe I'm beating a dead horse here. It sounds like there is really two things that could be accomplished here. They're different and I'm wondering if there is some way that you think about this. One is given a parameter budget, re-architecturing your network using block sparsity to improve your results. But separate from that, there is this issue of this whole finding the right 2% of parameters that actually matter, and then using block sparsity as a way to implement the network that just has what matters, and presumably the result is that you are able to compute those much more quickly. Am I thinking about that the right way? Do you

think of those are the same problem, or are they two different problems that are enabled by this block sparse kernel approach?

[0:38:21.4] DK: I think you are right. Yeah. These are indeed key problems that you can now tackle. Indeed, yeah so either static sparsity, which is what we had done in our experiments, or dynamic sparsity, which is where you are learn the sparsity better. Yeah, it is also an interesting application of this.

[0:38:43.3] SC: It strikes me that a big part of the reason why you care about any of these at all is because of computational limitations. Is that the main idea?

[0:38:53.0] DK: That's the main idea. But that's I think the main idea of the whole field of computer science, right?

[0:39:00.3] SC: Touche. That was a little bit of a segway into in spite of the fact that we're getting around computation limitations here, you actually had access to some pretty fancy hardware to try this out on. Can you talk a little about A, the specific problem that you – the problems that you were looking at to push the limit and then how the experimental results you saw on the hardware that you're using?

[0:39:35.4] DK: Absolutely. Absolutely. One of the problems that we wanted to solve is to train huge LSTM on a very large data set of text in the Amazon reviews. We were so lucky to have access to Nvidia DGX-1, which is hardware from Nvidia that allowed us to train much larger models than we could've trained otherwise. Yeah, this was something that enabled us to basically get state of the art performance on the Amazon review data set. This was also instrumental to get the results I talked about on access of lying sentiment.

[0:40:22.0] SC: What was it about the LSTM that made it huge?

[0:40:26.9] DK: The Amazon reviews data set is just a very large data set of reviews on Amazon obviously.

[0:40:33.9] SC: Do you remember how many reviews are in that data set?

[0:40:37.0] DK: I don't know. I don't have it, but it's like a large portion of all them. Basically the model of that you could fit on this data is like an order, a magnitude larger than anything we tried earlier. You also need hardware to be able to fit in such a model.

[0:40:56.6] SC: If you weren't using the DGX-1, what would you have used otherwise and how did the results that you saw compare?

[0:41:05.2] DK: If we wouldn't had the DGX-1, then we would've had to use a cluster of GPUs. Recently, it has become clear that for some problems, it is possible to train with very mini-batches, or to spread out training and go to a large cluster. But then you will still get big problems of your number of parameters, so that is not ideal still.

Typically, you still need to fit your parameters on a single GPU. Even if you can split your mini-batch data across multiple machines, then you're still bottleneck by the memory of seeing the machine.

[0:41:52.8] SC: Okay. Do you have a sense for at the end of the day, well how long does it take you to train your models and how – do you have a sense for how much faster it was relative to what you would've done otherwise?

[0:42:06.3]DK: I think it allowed us to train the model about twice as fast as otherwise. It still took I believe about two weeks to train the model.

[0:42:17.0] SC: Wow. Wow. It sounds like a really interesting project with some potentially broad applications that I'll be keeping an eye out for. Clearly, you'll be publishing a paper around this. Are you also publishing code, or is it more the research results that are the important takeaway for folks that want to build on it?

[0:42:38.8] DK: Yeah. It's the second. We are actually publishing the code. We are publishing the GPU kernels. I think this is by far the most interesting part of what we released, because this actually allows practitioners and researchers to do completely new things very easily. It's just basically a matter of importing the new library and replacing your existing convolutions, or metric multiplications with the block sparse ones, and you're good to go. Yeah, it is actual software to use for others.

[0:43:18.1] SC: Okay. Do you at this point have a place that you can point folks to, or do you know where folks will be able to find this work once it's published?

[0:43:26.1] DK: You could find the work on openai.com, and you go to the black boat there. You'll find a black boat on this topic and the link to the Github.

[0:43:38.8] SC: Okay. Fantastic. Great. Great. Well, Durk I have really enjoyed chatting with you. Is there anything else that you like to leave the audience with?

[0:43:47.0] DK: Yeah. I think we had a very interesting conversation. Thank you for inviting me to the show. Yeah, I encourage everyone to keep sharing results, to publishing source code of your experiments and keep an eye out on the research of OpenAI. That's it.

[0:44:04.1] SC: Great. All right, well thanks very much.

[0:44:06.9] DK: Okay. My pleasure.

[END OF INTERVIEW]

[0:44:14.2] SC: All right everyone, that's our show for today. Thanks so much for listening and for your continued feedback and support. For more information on Durk or any of the topics covered in this episode, head on over to twimlai.com/talk/80. To catch up on our Open AI series, visit twimlai.com/openai.

Of course, you can send along your feedback or questions to me via Twitter to @samcharrington, or @twimlai, or leave a comment right on the show notes page.

Thanks once again to Nvidia for their support of this series. To learn more about Nvidia and their presence at NIPS, remember to head on over to twimlai.com/nvidia.

Thank you once again for listening and catch you next time.

[END]