EPISODE 691

[INTRODUCTION]

[0:00:00.3] JM: Google Brain is an engineering team focused on deep learning research and applications. One growing area of interest within Google Brain is that of generative models. A generative model uses neural networks and a large data set to generate new data similar to the ones that the network has seen before. One approach to making use of generative models is GANs, Generative Adversarial Networks.

GANs can use a generative model, which creates new examples together with a discriminator model, which can classify examples. This topic can be hard to explain, but I think it's really important and I want to try to explain it in this preamble, as well as in the show today with Doug Eck.

To explain this, think of two modules and these two modules are machine learning systems that are called a generator and a discriminator. Let's say we want to generate brand new pictures of cats. We want these pictures of cats to be completely novel pictures of cats, but they're generated by our machine. How can we do that? We want an artificial cat picture generator.

Well first, we know how to train a classifier of cats. We know how to train what is called a discriminator, by feeding it billions of example pictures of cats that are labeled to be cats, and the machine figures out the contours of an image that map to a cat. We now have a model that can tell what a cat is. If we feed it a picture of a cat, it'll say that's a cat. If we feed it a picture of a dog, it will say that is not a cat. It can distinguish cat versus not cat.

Next, we make a model that generates completely random images, and we feed those randomly generated images to the discriminator. The discriminator will not only tell us that these are not cats, but it will output a loss for these random images. A loss is a metric that we can use to represent how far off this given image is from being something that the discriminator would recognize as a cat. You have this metric of how bad of a job is your other module, this generator. How bad of a job is it doing in generating pictures that resemble cats?

The generator can use that information, the loss information to adjust its weights in a way that will reduce loss. Overtime, the generator gets better and better at reducing loss, so the generator gets better and better at figuring out how to fool the discriminator into thinking that this randomly generated image is a cat. Eventually, this discriminator will start to believe the generator. This is why it's called a generative adversarial network. You have a generator and a discriminator, and they're competing against one another. The generator is making new images, the discriminator is telling whether or not the image is something that is believable.

The composition of these two things together allows you to produce novel examples, because eventually the discriminator gets fully tricked and the generator is producing images that look like cats. Generative model systems have produced some really useful applications. Some examples include object detection models, image editing and text to image generation.

Today's guest is Doug Eck. Doug works on the Magenta team at Google Brain. Magenta uses applications of deep learning to produce tools and experiments around music and art and creativity. As you've seen in the example that I just gave, if we know how to train a model to produce new images of cats, that's a creative act. What else can we utilize this creative tool of the generative adversarial network, or just generative models more broadly, what applications can we build out of that?

In a previous show with Doug, we talked about his vision for humans and computers working together to do creative tasks, such as music. Today, we dive into some of the core machine learning building blocks that make machine creativity possible. It was a great conversation. I really enjoyed talking to Doug and I think you'll enjoy it as well.

Before we get on with this episode, I want to mention a few roles that we're hiring for and you can find these at softwareengineeringdaily.com/jobs. We are looking for podcaster, we are looking for some writers. We have very high standards for who we hire to do editorial material, but please do apply if you're a good fit for either of those roles. We are also looking for advertisers for Q4. If you're interested in advertising and reaching 50,000 developers through Software Engineering Daily, we would love to talk to you and you can send us an e-mail. We've got a form on the website on softwareengineeringdaily.com for contacting us.

With that, let's get on with this episode with Doug Eck.

[SPONSOR MESSAGE]

[0:05:31.1] JM: Kubernetes can be difficult. Container networking, storage, disaster recovery, these are issues that you would rather not have to figure out alone. Mesosphere's Kubernetes as a service provides single click Kubernetes deployment with simple management, security features and high availability, to make your Kubernetes deployments easy.

You can find out more about Mesosphere's Kubernetes as a service by going to softwareengineeringdaily.com/mesosphere. Mesosphere's Kubernetes as a service heals itself when it detects a problem with the state of the cluster, so you don't have to worry about your cluster going down. They make it easy to install monitoring and logging and other tooling alongside your Kubernetes cluster.

With one-click install, there's additional tooling like Prometheus, Linkerd, Jenkins and any of the services in the service catalog. Mesosphere is built to make multi-cloud, hybrid cloud and edge computing easier. To find out how Mesosphere's Kubernetes as a service can help you easily deploy Kubernetes, you can check out softwareengineeringdaily.com/mesosphere, and it would support Software Engineering Daily as well.

One reason I am a big fan of Mesosphere is that one of the founders Ben Hindman is one of the first people I interviewed about software engineering back when I was a host on Software Engineering Radio. He was so good and so generous with his explanations of various distributed systems concepts. This was back four, or five years ago when some of the applied distributed systems material was a little more scant in the marketplace. It was harder to find information about distributed systems in production, and he was one of the people that was evangelizing and talking about it and obviously building it in Apache Mesos.

I'm really happy to have Mesosphere as a sponsor. If you want to check out Mesosphere and support Software Engineering Daily, go to softwareengineeringdaily.com/mesosphere.

[INTERVIEW]

[0:07:50.0] JM: Doug Eck, you are a Principal Scientist at Google, you're a Research Lead on the Google Brain team. Welcome back to Software Engineering Daily.

[0:07:57.9] DE: Hey Jeff, thanks for having me back.

[0:07:59.7] JM: The last time we spoke, we spent a lot of time talking about music and machine learning and the interaction between humans and computers on music and art projects. I really enjoyed that conversation, but this one I'd like to go a little bit deeper into some elements of engineering. To my mind, one of the core ideas at least in terms of what you're working on in Magenta is applications of generative models. How would you describe a generative model?

[0:08:36.2] DE: I would say there the phrase is used in a couple of different ways. One group uses generative models to talk about Bayesian inference, so probabilistic models that not only look at the probability of some prediction, but actually try to model the distribution – well, prior distribution. I use the term more generally to talk about machine learning models that learn to generate some instance of the data upon which they're trained.

[0:09:02.0] JM: What are some examples where a generative model under your definition is useful?

[0:09:06.3] DE: Well, useful is a great question. It's not clear to me yet how useful these models will be. We might use them to make music, that's what Magenta is about largely, or to help us make art. We might use them to generate summaries of long documents, but I think this area of research is still in its infancy, so we haven't seen I think a homerun yet in terms of actual utility.

[0:09:26.6] JM: How does a generative model, a model that can create new pieces of music, or new pieces of art, or new animals, how does a generative model get trained?

[0:09:38.4] DE: Great question. Let's talk about one specific generative model, and I think talking about the concrete details of that model will give us some ideas about how these things work in general. Let's talk about what's called a variational auto-encoder. It's a bunch of words, but the basic idea is if you have some data, that data might be the pixels from an image, that

data might be a sequence of pen strokes, that data might be a bunch of musical notes, but let's talk about the pixels from an image.

What we're going to do is take a neural network that is able to process those pixels, so learn weighted connections over small patches of those pixels such that it can predict something about those pixels and imagine what we're going to do is try to just reproduce that image using the neural network, but we're going to pass that information through a bottleneck, such that at some point in this neural network there just aren't enough neurons to memorize all those pixels, right?

What's going to happen if we're training the network and we're rewarding it with gradient descent to reproduce the image is this model is going to do as good of a job as it can over lots and lots of images. If things work well, that model is going to learn some general properties of the images, because by learning those general properties, it's going to do a better job overall in reproducing the images. What you should be thinking about in your mind's eye is an hourglass, where we have the image on one side and we have lots of neurons patch-like processing over this image, funneling in to a narrow bottleneck, a narrow waist and then fanning back out as we try to reproduce the image. Are you with me so far?

[0:11:13.0] JM: I am.

[0:11:13.6] DE: Okay. That's an autoencoder, and it's called an autoencoder because it's trying to reproduce the information upon which it's trained, right? The same picture in, the same picture out and the gradient descent method we're using is simply rewarding the model for learning to reproduce those pixels, all right?

We're not quite to a generative model yet, but we've got one building block and that building block is this narrow waist of the model. I'm going to call that the embedding space of the model, the activations at the most narrow point in the model. Then let's introduce a couple more terms. Let's talk about the left-hand side, if you will. I usually visualize this hourglass on its side with the source image on the left and the reproduced image on the right.

On the left-hand side, you'll see this hourglass funneling into the middle, let's call that the encoder, because its job is to take the pixels and encode them into this narrow waist, what I'm calling the embedding space. Then on the right-hand side, you're going to imagine fanning back out so that we can try to reproduce the pixels, okay. Still with me so far?

[0:12:14.1] JM: I am.

[0:12:14.9] DE: Okay. Now, let's say that we just generate a random embedding, okay? We take that narrow waist and instead of using the values that would be calculated from some image on the left-hand side, we just roll the dice and generate some random floating-point numbers, and then we decode those numbers and try to make an image from them. In your mind's eye, what's going to happen?

[0:12:39.7] JM: You're asking me?

[0:12:40.8] DE: Yeah.

[0:12:42.0] JM: I think the random number set that is trying to get transformed into an image is going to map to some strange visual representation.

[0:12:52.9] DE: I think you're exactly right. I would add that in practice, if the only way that you're training this neural network is mean squared error over reproducing the original pixels, that middle embedding space most of the time is going to mean nothing, that the model will only use certain parts of that embedding space to get its job done. When you roll the dice, you're not very likely to land on a point that actually looks like anything to us. It's just not going to work. Basically, you're going to get static images that don't look like anything, but the snow on a TV set.

[0:13:23.3] JM: By the way, the embedding space here – would you say the embedding space is translating this random vectors of numbers into a meaningful visual representation? It's a visual representation, not necessarily meaningful, but it's just mapping it to some visual representation?

[0:13:42.0] DE: Yeah. That's the right way to think about it. At a more abstract level, think of the actual numeric values for the pixels, right? You have some image, 256 by 256. For each of those pixels in that image, you probably have three values, RGB. Each of those values is maybe between 0 and 255, or 0 and 127, okay? That in and of itself, the image lives in some high-dimensional numeric space, right? You can think of it as a 256 times 256 times 3 or whatever that number is dimensional space, right? It's a point in that space. Does that make sense so far?

[0:14:19.8] JM: Yeah, 256 by 256 in terms of the dimensions of the image and then 3 in terms of the RGB values.

[0:14:26.9] DE: Sure. As a computer scientist, you can just imagine making a vector, right, of that length and that that would perfectly represent uncompressed your image, okay. Now what we're learning is a much more reduced dimensionality. We might only have 16 floating point values, or maybe let's work with 16. We have a vector of size 16 floating point values and that's our reduced dimension the waste of this hourglass that is our embedding space. What we're doing is we're learning a projection.

From what we'll call data space, which is that 256, 256 times 3, from data space into this latent space and that's the job of the encoder. The job of the decoder is to take any point in this 16 dimensional latent space and move back into data space, hopefully landing really nearby the source image.

We're necessarily compressing, because we don't have enough – by the way, set aside the distinction between floating point numbers and integers. I mean, someone in the audience is saying, “Wait, this floating point value can store a lot more information than the integer values can.” Yes, but we're still compressing. We're compressing down to 16 floats and we're losing some information.

Now my question was what happens if you roll the dice and generate a random length 16 floating point vector and just run it through your decoder? You said you'll get something, right? You're right. You'll get something visual, because we're generating a new image that we can then display on our computer screen, right? By definition, we'll get something visual. It turns out

that let's say that we train our model only on pictures of dogs, right? Turns out, if all we do is train for this reconstruction loss, which is the Euclidean distance between the pixel values we're trying to make and what the model predicts, then we won't get much meaningful by rolling the dice, okay. Here is where the first –

[0:16:13.5] JM: It won't even look faintly like a dog.

[0:16:15.3] DE: Well, it might every once in a while, right? Sometimes we're going to land on a point in that 16 dimensional space that is near some of our training data and we'll get something that looks okay, all right? One way to think about this is if we were to look at the distribution of all of our training data across these 16 values, so imagine I take one dog image and I project it and I make its length 16 vector, and then instead of doing that, I do it for all of my training data and I look at how the numbers are distributed over these 16 vectors. What you might find is that one of these dimensions, let's say dimension 7 of 16, there's actually only one or two spikes that get used, one or two areas of the space between 0 and 1, if we limit our – let's just say we limit our embedding values to be between 0 & 1.

What doesn't happen is the model doesn't learn to fill the entire 16 dimensional space, and because it hasn't learned to fill it when we roll the dice, we're almost guaranteed to land somewhere that's not being used by the model, and so it doesn't generate a useful image. I think this will make more sense if I tell you the trick that gets used to solve this problem, or one trick. It's more than a trick, it's a pretty influential model by Dirk Kingma and others called the variational autoencoder.

The idea is to add a second loss. By loss, I mean a second training signal for the neural network. That second training signal is to try to force this embedding space, these 16 values, each of those 16 values should look like a Gaussian noise distribution. By that I mean, the shape of it should look Gaussian, like a bell curve.

This is done by training the network to look more and more like – it's a little bit hard to explain, but we're going to have a random Gaussian noise process. It's just a random number generator, and we're going to train the model to try to have that same shape when we sample over and over from the model. What this means is we're trying to force the model to look like it's got some

random component, and at the same time jointly, we're going to try to get it to reproduce these images.

Now this is a funny game to play, because if we do too good of a job at minimizing this variational loss, then we've successfully trained a noise generator and that's not what we're trying to do. The game is to only train a certain proportion of time on this variational loss. For those of you that are technical, you basically train against the KL divergence, the distance between the distributions of your embedding space and this random noise process. Now, what we have is imagine in your mind's eye, without this loss, there's no constraints on what parts of the 16-dimensional space will get used by the model.

With this loss in place, the model is forced to use the entire space, because it's being forced to randomly spread itself out into that space. Now, when you roll the dice and you generate a random embedding and decode it, it's very likely that you're going to land near something in the training data. You're going to land in a space that makes sense to the model if it's populated by the training data. You'll get something that looks meaningful to us. You'll get things that look like dogs.

[0:19:38.8] JM: That's because your dice rolls now have a normal distribution enforced among them?

[0:19:45.9] DE: Sure. You'd roll a normal dice. That's right.

[0:19:48.1] JM: Okay. Once you have that happening your model is guaranteed to generate some images that do look like dogs and that's what you've accomplished there. I mean, most of it is still junk?

[0:20:04.9] DE: First, we never – there are no guarantees or warranties given to the user by machine learning researchers who knows what's going to happen with your data, but it works better, how about that? It also tends to model – if your distribution is multimodal, which most real-world distributions are. If you look at pictures of dogs, you've got some that look like Dalmatians and some that look like – there's multiple modes sitting there. It tends to model most or all of them.

One example of this that I like – I'm biased, because I this was done by David Ha and I helped out a little bit is a model called sketch ironing, which learns to sketch that is actually generate pen movements to draw different things like cats and dogs and fire trucks, and in fact any class that was used in the Google Creative Labs game quick-draw where you're playing pictionary against the computer.

It's quite striking to look at random samples from say, the yoga class. People trying to draw pictures of people doing yoga, the model just draws a huge variety of different kinds of yoga poses. If we turn off that variational loss, it simply doesn't do that. It doesn't work.

[0:21:10.0] JM: Now there is something called a discriminator that I don't think we have introduced yet into –

[0:21:15.3] DE: We have not talked about GANs yet. That's right. That's another way to force generative models to be near in some ways the training data and thus, would generate very sharp, very high-quality generated output.

[SPONSOR MESSAGE]

[0:21:36.8] 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-in to any one vendor or resource. You can continue to work with the tools that you already know, so just 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 offline. 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.

To learn more about Azure Container Service and other Azure services, as well as receive a free e-book by Brendan Burns, go to aka.ms/sedaily. Brendan Burns is the creator of Kubernetes and his e-book is about some of the distributed systems design lessons that he has learned building Kubernetes.

That e-book is available at aka.ms/sedaily.

[INTERVIEW CONTINUED]

[0:23:12.2] JM: Before we get into the discriminator, explain what we have accomplished once again with the generator, or dog picture generator and what the problems of the output are that we might want to solve with a discriminator.

[0:23:26.2] DE: Oh, perfect. Okay. The challenges that we face with generative models are that. It's actually, philosophically it's interesting I think. We're trying to solve an ill-posed problem usually. Certainly if we're trying to generate new pictures of dogs, we're trying to pose as an ill-posed problem. Well, we don't want to reproduce our training data, right? That's just memorization. For machine learning folks, memorization is usually pretty boring. What we want to do is generate something that looks like a dog to us, but actually isn't exactly the same dog as the one that we saw in the training set.

That means, what we want to do is if you imagine every dog as a 16-dimensional point, we want to really have a really good model of exactly where those points lay in that 16-dimensional space, and we want to be able to generate a point that's nearby one of the dogs maybe, but not too close. It's a bit of an ill-posed problem. In fact, it's really hard to evaluate whether we've succeeded or not, right? Because who knows? What do you do? Do you go out to people and say did that look like a dog to you? Actually that may be one perfectly valid thing to do, but it's hard to come up with a machine learning loss that both forces you to stay near the training data, but also ensures that you don't get too near the training data.

[0:24:47.2] JM: Right. Okay, so I follow that.

[0:24:48.9] DE: Great. What tends to happen is that you're always playing with these trade-offs between a model will learn to get near the training data, but not too near, okay. What it will do is it will generate something that looks like a dog, right, but it doesn't look exactly like a dog, because it's being forced to not over fit and not memorize the training data. What it does very often is generate blurry dogs, right? That's one way the model solves the problem. If you imagine this distribution and there's a number of points that are clustered together and if we looked at those individual points, we'd see they're all Dalmatians. That one thing we could do is just drop a point right in the middle of all those Dalmatians, right, and generate from it and be perfectly happy because we've got this blob that has black circles and white patches in it, right?

That's actually one thing that happens with generative models. Even the VAE, the variational autoencoder that I talked about, in certain cases it does generate – it models all the modes of the distribution, or many of them. It manages to figure out that there are Dalmatians and Doberman Pinschers that look different, but it generates blurry Dalmatians and blurry Doberman Pinschers.
That's a challenge and I think we're going to start to head towards your discriminator as one possible solution to this challenge, but we can pause for a second if I wasn't clear, or you have questions.

[0:26:11.6] JM: No. I think I follow so far. I guess, one question I would ask is are you getting blurry dogs here, or are we at the point where we're getting dogs with too many eyes, or too many arms? Do these problems exist at different phases of this GANs model that we're going to get towards?

[0:26:31.0] DE: I think the answer is always it depends on the data, it depends on the model. We definitely – these models for image generation are usually what are called convolutional networks. What they're doing is pull out your linear algebra folks, or your signal processing, they're sharing their weights across the entire image and they're sweeping, like imagine a little window of 8 by 8 pixels and they're sweeping across and they're looking for reliable shapes, or patterns that fit in these 8 by 8 pixels.

That means that that patch might work high in the upper left-hand corner of the image, or down in the lower right-hand corner of the image. As you move through the convolutional network, you're seeing the patch-like structure keeps zooming further and further out from the image. One of the behaviors of generating images from conv nets, the short term for convolutional neural networks is that yeah, you might get four legs, or five legs, or three legs, because the patch-like nature of the model hasn't actually figured out where the legs should be and how many there should be and what their orientation should be. This is a side effect of this particular architecture.

When you're using generative models for text, you get very different kinds of blurriness. It's not visual blurriness now, it's a kind of, if you would, semantic blurriness, where you realize this model isn't able to carry the same topic over three paragraphs, right? I'm using blurriness in a very abstract way. For music, it's that maybe the model just can't quite repeat that phrase twice, right? It's not quite getting the structure.

[0:27:59.6] JM: Understood. Okay, and so this is going to depend on the sample input and the way that the model is built. In the case of dogs, maybe if you have a bunch of pictures of Dalmatians at the center of them, you might get a completely different model from a bunch of Dalmatians that are positioned at different orientations in the picture, maybe in the upper left-hand corner versus the lower right-hand corner, you're going to get different generative outputs.

Let's move on to the discriminator. I think we've articulated the challenges that we're going to have post-generator and a GANs consists of the generator and the discriminator, so let's get into the discriminator.

[0:28:42.1] DE: Right. Ian Goodfellow had a nice idea. He was at University of Montreal at the time, and it was to train a discriminator to discriminate between counterfeit dogs, so to speak, and real ones. That discriminator looks at the output pixels of the model and it learns to discriminate between images that are generated by our generative model and ones that come from real dogs. Then that loss is propagated back through the generator, so the generator gets hints via gradient descent as to why it generated a fake bad-looking dog.

[0:29:23.3] JM: What the discriminator is doing, just to pair it what you just said is you've got your generator that's producing fake dogs, we've got a data set of real dog images. The discriminator learns the characteristics that separate real dogs from fake dogs, and then you can feed the lost there back into the generator, so the generator can get smarter about what it's doing wrong with its dog generation.

[0:29:53.5] DE: That is precisely right. You're hired.

[0:29:55.8] JM: Okay.

[0:29:57.8] DE: This is actually a job interview. Your guests know that, right? You're trying to get a job at Google. My next question for you Jeff will be –

[0:30:05.6] JM: It even turns into a game show.

[0:30:07.5] DE: That's right, exactly.

[0:30:08.0] JM: Getting hired at Google live.

[0:30:09.8] DE: That's right.

[0:30:12.2] JM: I think that would actually be popular. If you look at – and my Google Home just woke up.

[0:30:17.9] DE: Oh, yeah. That's good. I can do that too. Okay, Google. Buy some – never mind. I won't do this.

[0:30:24.1] JM: Okay. Cool. We've got the basic feedback loop articulated. What's challenging about this? I mean, why doesn't this solve all of our generative problems in reality?

[0:30:36.5] DE: Well, first to be fair, both of these models the VAE and the GAN have a untapped potential. There's a ton of active research being done in this space and advances

being made every month. I think the main reason that the GAN doesn't solve all of our problems is that unless treated with great care, it will learn only to generate a few kinds of dogs very well.

Actually, there's nothing in this discriminator to ensure that the model try to generate all kinds of dogs. Instead, the model will just maybe learn, “Oh, I know.” The generator goes, “I know, this model I can get really, really good at generating Doberman Pinschers, so I'm going to keep generating Doberman Pinschers.” Then the discriminator is happy. It stops propagating loss back to the generator, because there's not much more to learn about Doberman Pinschers, but the generator doesn't really learn to become a general-purpose dog generator, called mode collapse, by the way. It's even got a name. The model collapses on to only a few modes of the distribution that are in the data.

[0:31:40.8] JM: That's overfitting basically, right? I guess, you could call it that, or maybe that's just bad –

[0:31:47.2] DE: No, no. It's okay. I think it's that you're choosing the model carefully fits only a certain part of the distribution and actually doesn't fit the rest of it. Even if we had the next 8 hours, I would start to run out of my expertise because we've got hundreds of really smart people working in this space. We'd have to sit down and read all of the accepted and most of the rejected papers at NIPS in this area.

The basic idea is can we reliably model multiple modes of the distribution? Can we have the ability to randomly generate from latent spaces, covering the distribution and also being really sharp? You have other practical issues, like if you want to use these things to get a job done, you want the inference to be fast, right? You don't want to have to say, “I'll show you another dog, give me an hour.” Maybe you want to even give users some ways to tune the output so they get what they want, which is an entirely different set of challenges there.

[0:32:40.1] JM: My impression though is that if we can constrain the generation and discrimination process appropriately, we can start to do things well, or do things productively at least. We may not understand how they're working, or why they're doing certain things, but if we can constrain it appropriately, we can be productive whether we're trying to just write a piano sonata in A-flat. Maybe that is well-defined enough to have a generator and a discriminator start

to spit out compositions that are appropriate for that constraint. What kinds of applications are we actually seeing useful results with GANs, or maybe you could talk about applications of GANs within Magenta?

[0:33:26.4] DE: More generally about generative models, because I think that this GAN loss is only one part of the picture. A lot of what we're doing is sequence learning over language models. That is we're learning from either text, or music as notated. You can look at it as a bunch of characters. We see a whole family of models coming into play, not just GANs. With that aside, I think we're seeing lots of uses of generative models where they provide an expanded set of raw materials for people to work with.

I don't think I've seen again any killer app, right? Anything that I've seen in production used by millions of users that is great, yet you can for example, given an image, suggest to users a bunch of different versions of that image, you can provide the raw materials for a composition for someone. It's generative models on rails where the human is providing the rails saying, “Okay. Yeah, you got close there, but I'll fix that for you.”

I do firmly believe that we're in a horizon, this isn't one of these crazy in 40 years. I mean, I think we're in the horizon of 18 months, two years seeing generative models be part of the core algorithmic toolkit for design, for writing. I think maybe we need a little bit more time than that for writing than for images and for video.

[0:34:43.2] JM: If you take this is a case study in machine learning tool development and we look at the past machine learning tools that have developed, you've got a good historical knowledge of this space. Can you help orient me how the machine learning community makes advances on this? Do they look at the problem of GANs of the modality or whatever you called it, like the over-modality overfitting and say, “Okay, let's design an experiment that is subject to modality overfitting and let's figure out how to solve that specific example of modality overfitting, and then we'll just take that as the base case and then try to extrapolate that to other domains and gradually we'll learn how to solve this more general problem of the modality overfitting.” Or, take me through the experimental process of a machine learning researcher who's trying to break this bottleneck.

[0:35:41.1] DE: Great question. As far as I know, Ian Goodfellow who invented the GAN, he works here with us, so I should probably just hit pause and go ask him. The stories I've heard was that people were talking over beers about this problem and Ian had this idea, “Well, what if we just had a counterfeit detector, this discriminator?” Later in the night, he tried it out and it worked and kept working on it and wrote it up and the paper was really popular, because as simple as that.

I can say something a little more general though, which is in a very real way we're limited by existing models and by the data available to us. Machine learning moves, I think based upon current technologies in terms of computational scale and it also moves in terms of what data sets are available. Beyond that, we're standing on each other's shoulders. For example, we spent a lot of time doing image recognition classifying what objects are in images. For a long time, we were limited by data. We still aren't limited – we're always limited by data, but we were limited by data and a lot of research was going on using a dataset called MNest, which is the handwritten digits zero through nine, taken I believe from by the postal service from handwritten addresses train a zip code reader.

That drives a certain model and a certain research. Another researcher came out with a different data set called image net, which gives us thousands of classes of things identified in images; horses, cats, trains, cars. This drove a whole another wave of research in image recognition.

I think we're seeing work in generative models. One reason is that that same data is there to train on and there's no low-hanging fruit left in recognition and classification. You see people saying, “Hey, I've got all of these images. People are doing a pretty good job of identifying what things are in these images. What else can I do with these images?” Instead, we start trying to generate new horses and new cars and the field moves in that direction.

[0:37:40.9] JM: What about video? Isn't the YouTube video data set – have people been doing interesting things with that?

[0:37:48.3] DE: Yeah. People are very – including my group, are very actively looking at video generation. If it's hard to generate pictures of Dalmatians, wow, it’s really hard – it’s really hard to generate video. A whole another set of challenges come up with video, I guess the most

intuitive one is that you really need a video of course is comprised of a number of images, right? Maybe what is it? 24, or 30 frames per second? Call it 30.

A video makes sense to us, because there is some meaningful change in those images, a coherent and meaningful change in those images over time. Not only do you have to generate, let's say it's a picture of a Dalmatian running around the yard, not only do you have to generate a nice Dalmatian 30 times a second, but it has to be the same Dalmatian and that Dalmatian has to move in a meaningful way. There's all of this additional need to model the structure of the world as it changes over time.

There's a certain blurriness that we see in video generation that I think is a very good analog to generating blurry Dalmatians in images that are just blobs of white and black. That is having a perfectly crisp Dalmatian in frame one, just get gradually more and more blurry as the model is unable to choose a particular trajectory in time for those objects to take as they change over time.

[0:39:14.8] JM: Not to volley us out of the world of art and music and more innocent applications, but the world of maybe applied discrimination to discriminating real from fake, this seems like something is going to become very important, if not already important. I guess today, you could argue it's already important in the realm of text, or arguably I mean, in the realm of video already with these fake political videos that people are generating, do the fake models – do they reliably leave some artifact that is going to keep us from getting completely discombobulated by fake video spam?

[0:40:01.2] DE: No unfortunately. I think, all of us are aware that we have to be very careful with this technology and we have to be very careful to be gently doing research and how to detect and verify the media that's out there in the world. I won't try to dissuade your listeners to say, “Oh, no. This isn't a problem at all.” I think this is a very serious issue and it's an arms race between the people that are going to want to abuse this technology and the people that are going to want to use it to do good things.

[0:40:29.4] JM: Yeah. This is one of the reasons that I think Magenta is – you talked about it being questionable, whether music or art generation are useful, but one thing that’s for sure is

you're going to be exploring areas that are unexplored to the areas of machine learning generation and creativity. These things are going to have wider applicability beyond music and art. Actually, I can turn this into a question. You've been doing Magenta for I guess a couple years now and you're publishing papers. Are you seeing the results of papers that are coming out of Magenta get taken and readapted into areas that are applicable beyond music and art?

[0:41:19.9] DE: Great questions. First, I will say that we're finally starting to see the ideas and the work from Magenta actually being used in the space of music. That's already been a two-and-a-half-year challenge. We can come back to this later if you'd like, but the very short version of it is that musicians don't tend to like to set up Docker images of Python. Just a thing, but JavaScript it turns out, JavaScript, it's just the way that creative coders that are working in the arts are wanting to work. As soon as we were able to release Tensorflow.js, which is a JavaScript Tensorflow API, we quickly wrote Magenta.js.

Now we've got users that are combining Magenta.js, Tensorflow.js with Tone.js and P5. These are the building blocks of today's creative coders working in the arts. Once we were able to speak literally to them in their own language, things have taken off quite quickly. We have a number of I think really brilliant coders.

Toropa, that's Taru in Finland, Toropa is his Twitter handle. He's just doing really gorgeous music experiments using Magenta and a bunch of other tools from other people. That was an aside. I don't think that we've seen yet real use of generative models in writing, or in drug discovery, or in other scientific applications where we think we will. That doesn't surprise me, because you can get a lot done without using a generative model. For drug discovery, you don't necessarily need a generative model to be exploring the space of possible chemicals that might make useful drugs, but we also see that we have a direction with generative models that has such promise that it's definitely worth the investment of some number of months, or years of research to see where this goes.

I know that's not the sexiest answer, right? You want to hear hey, look at this super application. It's hard, right? People have been working on these problems with other approaches for so many years. I make another point and I don't know how this fits from before, but it's not just in terms of how science or specifically machine learning progresses. It's not just data and the

existing models. It's actually computational power, memory size. It's very clear that neural networks were not all that useful when we had limited memory and limited computation.

You really don't see neural network start to shine until you're able to process huge data sets on GPUs, or other kinds of accelerators like our Google TPUs with very large amounts of memory and really fast matrix multiplication. In the 90s, it was true that support vector machines, SVMs and other approaches really did outperform neural networks at that scale. When we're able to scale up to larger computational speeds and memory sizes, then different technology works.

There's a very, very suggestive bit of research that was done here at Google. We've been working – a group working with me, but I'm not on these papers, is working on generative document summarization. Imagine that you have this long document, often it's some Wikipedia page, but that tends to be a good research data set. What you want to do is summarize that document in a paragraph. The most useful state-of-the-art right now is a method called extractive method. Most useful method is to extract from this document that we're trying to treat, or source documents linked to it, the sentences necessary to summarize it. The model is not generating any new sentences, it's just finding the ones in these documents that does the best job of summarizing the topic.

The alternative is to actually try to generate a number of sentences that aren't actually in the source documents, that are new sentences that do a better job we hope of summarizing that document. These generative methods are getting better and better and depending upon the measure will soon outperform extracted methods.

Okay, but that was just setting the stage. With that in mind, you take a model trained on all of Wikipedia to try to generate summaries. Instead of training it to generate summaries, you just train it to try to reproduce the documents. You learn what we'd call a language model over Wikipedia. Think of it just by analogy to generate the pixels of Dalmatians, slightly different technology but it's the same idea.

Now if you take a 100,000 parameter model. By that, the model has a 100,000 weights and you fit it to Wikipedia and you generate a Wikipedia document from that, it's gibberish, it's garbage. If you take a 4 million to 8 million parameter model and you fit it to Wikipedia, it actually starts to

generate fake Wikipedia documents that have some structure and start to have a little bit of coherence.

[0:46:00.5] JM: Oh, man.

[0:46:02.3] DE: This is from a modeling framework called transformers, which is an alternative to recurrent neural networks; an entirely different conversation. Then if you take a 4 trillion or even a 100 billion parameter model and you fit it to Wikipedia, wait, the cool part is we don't quite know how to do that yet, but we're getting close. You start to see this very suggestive idea that okay, a 100,000 parameters was garbage. Order 1 million parameters is actually cool, but missing a ton of structure. What if we add another order of magnitude?

We're just getting there with the work. I'll focus on the Google work, but by saying very fairly, it's a very competitive space. There's a lot of great work going on everywhere like at Nvidia. At Google, we have TPUs, which are an accelerator framework that allows us to scale to very, very large datasets and very large models and we're just getting the infrastructure in place to be able to run these order of magnitude larger models.

There's a lot of debate even internally among researchers like well, is it actually going to help or are we just going to over-fit? Can we fit a billion parameter model to Wikipedia and is that enough to actually just get really nice generated Wikipedia-like documents? We'll see. I'm on the fence, but I'm also very, very interested to see where this goes, and to see really if just scale is one of the crucial ingredients and getting a little bit closer to having great generated outputs.

[0:47:29.5] JM: Is this unprecedented scale? Because why doesn't this just boild down to a gigantic MapReduce problem that Google has solved before with previous large-scale applications?

[0:47:41.1] DE: We need to share those parameters, right? Eventually, distributing this over – okay, so the thought experiment would be let's distribute this computation over a million CPUs, and it just turns out that the cost of moving the data around these shared parameters that need to be seen by all other – you have a very densely connected huge neural network. You're constantly having to shuttle back and forth the intermediate calculations that you've done on this massive matrix multiplication.

In fact to be fair, I think this was really the goal of the early work of Google Brain was to say, “Hey, can we do neural networks at scale on CPU?” The answer of using dedicated hardware for matrix multiplication, specifically GPUs, these sorts of accelerators have just become the Swiss Army knife of what we're trying to do. The math just scales differently.

[SPONSOR MESSAGE]

[0:48:40.1] JM: DigitalOcean is a reliable, easy-to-use cloud provider. I've used DigitalOcean for years, whenever I want to get an application off the ground quickly. I've always loved the focus on user experience, the great documentation and the simple user interface. More and more, people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU-optimized droplets perfect for highly active frontend servers, or CICD workloads.

Running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily. As a bonus to our listeners, you will get a $100 in credit to use over 60 days. That's a lot of money to experiment with.

You can make a $100 go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure and that includes load balancers, object storage, DigitalOcean spaces is a great new product that provides object storage, and of course computation. Get your free $100 credit at do.co/sedaily.
Thanks to DigitalOcean for being a sponsor. The co-founder of DigitalOcean Moisey Uretsky was one of the first people I interviewed and his interview was really inspirational for me, so I've always thought of DigitalOcean as a pretty inspirational company. Thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[0:50:47.8] JM: It's not a stupidly parallelizable challenge, because you don't have these atomic units that can later on get reduced really easily. You have to do data sharing.

[0:50:59.1] DE: Yeah, that's right. I mean, this amounts to a very large matrix multiplication, multiplying a tensor against a matrix. It's known in scientific computing from other areas outside of machine learning, that multiplying huge matrices doesn't scale well to just a bunch of CPUs sitting around in a data center. I mean, this is why we have supercomputers for other applications.

What we've done so far is largely used data centers full of GPUs, or TPS. What really happens is you're running the same model on multiple GPUs or TPUs. Then each one is seeing a slice of the data. Now we're moving towards being able to have mesh technology where we can actually run a huge model running chunks of the model, parts of the graph on different TPUs or GPUs, in our case TPUs and then pull the intermediate results back together in a way that's scalable.

[0:51:48.7] JM: Model parallelism.

[0:51:50.0] DE: Correct. Thank you.

[0:51:51.4] JM: Versus data parallelism.

[0:51:52.8] DE: Precisely exactly.

[0:51:55.9] JM: The timeline there, I don't know, of the forefront experiments, these are months away?

[0:52:01.5] DE: We're going to see this mesh technology coming out to the public this year. We've announced Tensorflow 2.0 and it should land Q4, that's the plan. Mesh Tensorflow will be part of it. That'll allow us to do model parallelism over multiple TPUs. Let me mention in all honesty to your listeners, that this is not an area of expertise for me. If I got any of the details wrong about how this works, I mean, it's just so hard to keep up with this stuff. I sit near experts,

but it turns out sitting near someone that does this work doesn't actually mean you know how it works. I try to keep up as well as I can.

[0:52:34.4] JM: Okay. Well say that if somebody corrects you, in order to be able to correct you they also have to e-mail me and potentially come on the show and talk about this from their perspective as well, because if they know how to correct you on it, they're probably somebody that will be interesting to talk to.

[0:52:50.6] DE: Hey, that's great. Okay, good point.

[0:52:52.8] JM: Let's talk about putting models into production, because and this is something that would have applicability to a lot of people out there and this is a continual problem you hear about among machine learning people like at Uber, or at DoorDash, or any of these places that have rapidly changing models. I don't know how rapidly changing your models are, maybe it's one and done, but you have at least put some models into production. You've got NSynth, you've got the piano thing that plays alongside you. Do you have any advice or best practices around developing and releasing models?

[0:53:31.9] DE: Yes. There's a tension between productionization and research. I think the answer really depends upon the goals of the user. If the goal is to learn Tensorflow, or to build your own model and put it in production for example in small scale, there's tooling for that. I think there's tooling that allows you to move really fast in one direction. Of course, there's always a trade-off. I'll give you an anecdote that will highlight this.

Someone recently joined the Google Brain team from one of the large groups responsible for building out Tensorflow serving capacity. When he joined this research group, he himself is trying to come up to speed on building novel new Tensorflow models and he said, "I finally understand why researchers code the way they do." He said, "I thought they were just lazy and didn't want to do good coding practices. Then I spent a day building a model and of course, I wrote some tests. I did some unit tests and then I wrote some more tests and I cleaned up the code and I linted it. Then I realized that it didn't work. It just wasn't a very good machine learning model and I had to throw it away."

I said, “I get it, right?” All of the tooling and the care that is placed into being able to take a model and serve it at scale reliably, that tooling takes work and using that tooling rather takes work. The point I want to make is that what I think we need is a framework where researchers can move fast and try things out and do small-scale experiments that can easily be turned into large-scale experiments by turning a knob. At the same time, we need folks in industry who want to take a trained model and serve it and we need to give them the tools for productionizing and doing all the things that it takes to make the model bulletproof and fault tolerant and not likely to wake you up in the middle of the night with some page, right?

That lays the groundwork. I guess I'd like you to maybe a follow-up question, like what specifically can we discuss in that space?

[0:55:28.1] JM: Well, one thing that I've heard is you train your model, you deploy it and then you've got some new data for the next version of the model. I guess, figuring out how to integrate new datasets into existing models can be difficult. Making sure I guess tested on subsets of the traffic to make sure that this data doesn't totally ruin your model. If you're overfit on – if you're building a ride-sharing service in Utah and then you get some new data from Texas, can you use the data from Texas on the model for Utah, or can you combine the models, and then maybe you combine the models and it works well and then you try to be the same for Oregon and all of a sudden, Oregon data ruins everything and then it's like, “Okay, where do we roll back to? What was the safe state? Can we easily roll back?” I don't know. Maybe it's just like any other continuous delivery process to some degree.

[0:56:24.6] DE: I mean, I think the answer is it's harder than other continuous delivery processes, because of what you just described. I think the tooling for that is getting better by the quarter, but it's going to be really tough to have a perfect solution. We do have a, I think it's already pretty good and growing serving platform for a Tensorflow called Tensorflow serving. I'm afraid I'm not up to date on what's going to be released when, but I think I can say we realized that if you want people to trust to Tensorflow and actually want to use it in production, you've got to have high quality production tools.

We do have serving tools that are continually getting better on our cloud and we also have of course APIs that clients can call out too. In which case, it's our job to keep those APIs trained, like the translation keeps getting better, etc.

On our side, just even if we talk about the small-scale work in getting Magenta.js to work, it turns out that even something as simple as serving the trained weights of a neural network for making melodies comes with its own challenges and is always a tough balance between simplicity and flexibility, right? Had to make hard decisions everywhere. I suspect your listeners are given what I know about this particular podcast here, my sense is most of the people that I'm talking to know more about this than I do.

I did work on a production team, play music. I ran the recommendations team. I was on pager rotation for many years, but I never really became an expert as serving. I was always too drawn back into the machine learning.

[0:57:51.9] JM: Okay. Well, let's begin to draw to a close by talking more about Magenta, because we've really just gone circuitously around it, you mentioned that people are showing more of an interest in using this, your set of Magenta projects in their actual music production. Thanks to Magenta.js, which is simplifying the model delivery process to the browser, I believe. Can you talk about how those musicians are using Magenta in the browser?

[0:58:26.8] DE: Most of it so far has been what I would call playful. That is we're seeing increasingly good music models that actually generate really cool music, or generate really cool rhythms and we're seeing some pretty amazing JavaScript coders wrapping them into code pens, or other commonly used JavaScript wrappers. For the most part, these are fun toys to play with. Like, here's a smart drum machine you can play with. We're just now seeing these same coders start to take the next step of for example, allowing you to drive these with a midi clock that lets you have more control over the process, or getting the pipelining in place to move these signals around not just between different JavaScript tools, but being able to move this back into a really sophisticated digital audio workstation like Ableton.

This is happening as we speak. In the next month or so, we'll continue to release better and better integration points for musicians to work with the tools they love. This comes from us doing

a lot of user studies. When I started Magenta, I did not think – I thought as a musician myself, I understood what musicians want. It turns out, I didn't. I think those of us working on it, we really thought like programmers, we thought like machine learning developers. We really didn't get what musicians who aren't coders really want.

[0:59:48.7] JM: Oh, tell me more about. What were you wrong about?

[0:59:52.2] DE: One of the things was we thought that people would be comfortable completely abandoning their current tools and just jumping in and playing around with an interactive coding environment. One thing we learned was you can pry Ableton from my – what is it? You can pry it from my dead hands, what is it? From my dead body.

[1:00:08.5] JM: Cold dead hands.

[1:00:09.7] DE: There it is. Right. People, the kind of musician we're trying to get to is probably someone that's using a tool like Ableton. They're probably – they might be, but they're probably not an oboe player in a symphony orchestra and only an oboe player in a symphony orchestra. They're probably using technology already, the early adopters and they're probably in love with this technology. What we've realized is that we need to do things like, right now, we're trying to wrap our JavaScript tools in a headless browser what is called Chromium, I think. Then have these tiny little JavaScript-powered musical instruments be able to just talk back and forth to something like Ableton. We're trying to pull them out of the browser, but we're trying to leave them in containers that make it easy to integrate them with other tools.

[1:00:55.6] JM: Are you making VSTs?

[1:00:57.0] DE: We're also looking at VSTs. Since we're moving just MIDI around, there are other ways for us to do what we want to do. We're trying different things, but think of moving in that direction, especially if we start working with tools that generate audio. At the same time, I'm actually – I was shocked at – we did this neural synthesis project called NSynth, which is basically a wave net, which is a generative model for audio from deep mind. We did a variant of wave net called NSynth, which can generate musical notes.

People loved it. It was quite a popular project. Then another group inside of Google from London decided they wanted to actually build a piece of hardware that built a hardware synth that uses NSynth internally, called NSynth Super. This was so popular. Everybody that got their hands on NSynth Super, physically had it in their hands plugged in a set of headphones, plugged in a MIDI keyboard and started move your finger around on this grid and change the sounds. By moving it just a little bit make really subtle changes in tamper. People really loved the physicality of it.

I have renewed respect for trying to build physical devices and I'm hoping with Raspberry Pi and with other platforms, we can see an ability to put out an open-source, the code and the plans for building guitar pedal-sized, AI powered musical instruments. That's not happening yet, but that's one direction I want to go in. That thinking of the physicality of it and the ease of use and actually caring about to user design UX, that's all learned the hard way because we spent a year building complicated protocol buffers and big Docker packages for you to load onto your Mac.

[1:02:29.4] JM: Other than the musicians being stuck in their ways and you can pry my digital audio workstation for my cold dead hands, any other insights over the last year since we've talked any overriding insights, or changes in the long-term goals of Magenta, or you more spiritually as a musician technologist? I know this is really your passion is figuring out technology built for music.

[1:02:59.5] DE: I think I'm on the same path, more or less, but I'm more and more interested in ways in which to allow users to control generative models and less and less interested in making the generative models better. I want the generative models to get better. It's important work. I'm equally interested in simple techniques, like conditioning the generative model on something we know, or something we want to use or to be able to control. A concrete example always helps.

We do have these really nice RNN, recurrent neural network and actually LSTM-based music generators that if you do – they generate MIDI, but you listen to them as piano and they sound really cool. You can just could turn one of these on and it'll just wander away, making quasi-realistic piano music for you, but that's not that interesting long-term. I think you want to have

some controls. One of the researchers in our group Ian Simon, had the simple idea – it wasn't just Ian, it was a couple of people, Sagiv [inaudible 1:03:51.8] as well, the simple idea of saying, "Well, it'd be really nice if we could control how busy the music sounded, the note density, how many quarter notes per second, or how many quarter notes per measure." Because if you make it sound really sparse, or you can make it sound really crazy with lots of notes.

They did something that I thought was a clever trick. I don't claim the first to do it. They just, as they're training the model, the code counted like a running average of the number of notes that were happening per measure. Then provided that value to the neural network while it was learning. Now the neural network can learn just fine without that conditioning value. I mean, it's not that hard for a neural network to count, right? It turns out that the model, the neural network actually learns to rely on that value of how many notes per second, or notes per measure are being generated and take advantage of it to make the job easier.

In doing so, it means that later when you're generating music, you can actually just change that number. You can say, just give me two notes per measure, or give me 16 notes per measure, and the model will obey it. It's like, "Oh, look. We can condition these models on everything from genre, to mood, to any other thing a user might want to control, and in doing so we give them ways to control that generation." I think it's just as important for an artist to be able to control the generation as it is for the generation to be good, because in controlling it the artist makes it belong to her, or belong to him.

[1:05:13.7] JM: Yeah, that's profound. It actually reminds me of –

[1:05:15.9] DE: I don't know if it's profound, but it's an interesting direction to go in.

[1:05:19.3] JM: Well, what I think is profound is, I did an interview with the Splice CTO a while ago. You know this company Splice?

[1:05:26.6] DE: Mm-hmm. Yeah, they're cool.

[1:05:27.9] JM: Yeah, they are. What's interesting is they started off focusing on music collaboration and they still have that. They still have music collaboration and version control and

so on, but the bread and butter of the business has become a sample rental library, because that's what the musicians wanted, a sample rental library, a synth rental library. Just funny how the demands of musicians can be, or can maybe seem fickle until you really examine them and you gradually understand why they are a certain way. It sounds like you've been going through that.

[1:06:01.2] DE: Exactly right. We're trying in some sense to follow the market with the constraint that we still remember that our main job is to do science. I mean, who we are is researchers and we have a responsibility to whatever else we do to keep trying to move the field a lot.

[1:06:16.3] JM: Okay Doug. Well, I'll let you go. This has been a great conversation. I really enjoyed talking to you.

[1:06:20.3] DE: All right, Jeff. It's been great talking to you too. We'll get back to you on our hiring decision as soon as possible. I think, I want to tell you did a great job and whichever way it goes, you're doing great.

[1:06:32.6] JM: We'll see you next time on – who wants to get hired at Google.

[1:06:38.9] DE: Exactly right.

[END OF INTERVIEW]

[1:06:43.0] JM: Nobody becomes a developer to solve bugs. We like to develop software, because we like to be creative. We like to build new things, but debugging is an unavoidable part of most developers' lives. You might as well do it as best as you can. You might as well debug as efficiently as you can. Now you can drastically cut the time that it takes you to debug.

Rookout rapid production debugging allows developers to track down issues in production without any additional coding. Any redeployment, you don't have to restart your app. Classic debuggers can be difficult to set up. With a debugger, you often aren't testing the code in a production environment, you're testing it on your own machine, or in a staging server.

Rookout lets you debug issues as they are occurring in production. Rookout is modern debugging. You can insert Rookout non-breaking breakpoints to immediately collect any piece of data from your live code and pipeline it anywhere, even if you never thought about it before or you didn't create instrumentation to collect it. You can insert these non-breaking breakpoints on-the-fly.

Go to rookout.com/sedaily to start a free trial and see how Rookout works. See how much debugging time you can save with this futuristic debugging tool. Rookout integrates with modern tools like Slack, Datadog, Sentry and New Relic. Try the debugger of the future. Try Rookout at rookout.com/sedaily. That's R-O-O-K-O-U-T.com/sedaily.

Thanks to Rookout for being a new sponsor of Software Engineering Daily.

[END]