EPISODE 1124

[INTRODUCTION]

[00:00:00] JM: Hyperparameters define the strategy for exploring a space in which a machine learning model is being developed. Whereas the parameters of a machine learning model are the actual data coming into a system, the hyperparameter define how those data points are fed into the training process for building a model to be used by an end-consumer.

A different set of hyperparameter will yield a different model. Thus, it is important to try different hyperparameter configurations to see which models end up performing better for a given application. Hyperparameter is an art and a science.

Richard Liaw is an engineer and researchers and the creator of Tune, a library for scalable hyperparameter tuning. Richard joins the show to through hyperparameter and the software that he has built for tuning them.

We're looking for an engineer who can also write. If you're interested in helping us write articles and contribute content to Software Engineering Daily and prepare for episodes, send me an email, jeff@softwareengineeringdaily.com. I'm also looking for investments. I'm making investments in developer tooling and infrastructure. You can send me an email, jeff@softwareengineeringdaily.com. Thanks.

[SPONSOR MESSAGE]

[00:01:15] JM: Bridgecrew is a developer-first cloud security platform. In addition to helping teams address cloud resources in production , Bridgecrew shifts cloud security left by scanning for fixing and preventing missed configurations in the infrastructure as code level. We've done a show about Bridgecrew, and it supports AWS, Azure, Google Cloud, Kubernetes, Terraform, many other things. Bridgecrew makes cloud security accessible to developers by integrating earlier in the development lifecycle, VS VCI/CD, and implementing fixes directly back into developer workflows via pole requests.

You can go to bridge crew.io/sedaily to learn more. That's bridge crew.io/se daily. Checkup Bridgecrew, and it would help out the show if you go to Bridgecrew.io/sedaily.

[INTERVIEW]

[00:02:11] JM: Richard, welcome back to the show.

[00:02:12] RL: Thanks, Jeff. Flattered to be here.

[00:02:14] JM: Last time we talked about a bunch of applications that people can build with Ray and some of the higher level platforms that have been built on top of the Ray primitives. And we meant to talk about hyperparameter tuning in much more detail, but we just got lost into the details of Ray. Today I'd like to take the reverse approach and just focus on hyperparameter tuning and then hopefully we can get into Ray at the margins. But let's just start with hyperparameter. What is a hyperparameter?

[00:02:47] RL: Yeah. Obviously, in order to sort of explain hyperparameters, I have to give a little bit of context. I'd like to start with machine learning in general where typically you have a model and an algorithm that compose like a machine learning task or something that does machine learning for you. I need to explain what a model is and the algorithm is and to really provide the context for talking about hyperparameters in general.

So a model is a representation. It's essentially a data structure. It holds information and typically especially for deep learning, these include a lot of matrices. On the other hand, the algorithm is the component of the machine learning process that crafts or creates this model. So it decides how to update this model, how fast to update it, what is the particular methodology that we're going to distill information into the model, right?

So now we have to find like high-level concepts for model and algorithm and we can talk about hyperparameters. So to go back to your original question of what is a hyperparameter, the hyperparameter is typically used to describe settings and configurations for either/or both the model and the algorithm. So it's a configuration value for this training procedure that typically doesn't change during the training process.

[00:04:30] JM: The use of hyperparameters in the development of a machine learning model, can you just go a little bit deeper into why hyperparameters are necessary for machine learning?

[00:04:42] RL: Yeah, that's a great question. I think perhaps a good analogy is like when you're baking or when you're cooking. The common joke is that cooking is just like hyperparameter search. Like you might have a recipe that works right out of the box, but if you're doing something new or you're trying something different, there are just so many different decisions that you have to make and so many different quantities and essentially configurations, and everything affects the performance of the final creation. In the cooking analogy, it's your food. In hyperparameter tuning and machine learning, it's the model that you're going to use for your business later on.

So, sticking true to the analogy, if you include too much of a certain quantity, like in cooking, say, salt, then your food is going to taste not so appetizing. On the other hand, holding this analogy back into the machine learning world, if you set, say, the model to be too big, it's possible that the final thing that you learn is just not like you don't learn the right – You don't train the right model and it underperforms your expectations.

[00:06:04] JM: I see. The cooking analogy, like we're always looking for the right recipe. We're always looking for the perfect food dish, which is prepared at the end of the cooking procedure. And that cooked dish is the machine learning model that we're going to cook. And the hyperparameter are things like how much salt am I adding? How much spices am I adding? What are the order in which I do the operations within cooking? Do I put in the eggs first? Or do I put in the flour first? Changing all of these different things will change the end result of the recipe.

[00:06:46] RL: Yup. I think analogy will last a while. I mean, or go pretty far. Yeah.

[00:06:52] JM: Okay. In that analogy, it's obvious what happens if I put in salt at the beginning of the dish versus salt at the end of the dish. The salt is probably going to be more evenly

distributed throughout the food that I'm cooking. What about hyperparameters? How does changing a hyperparameter change a machine learning run?

[00:07:11] RL: Right. That gets into a more advanced topic of these like dynamic hyperparameter during training. Note that this is very different from typical hyperparameter tuning where you might train a single model with a single set of hyperparameter statically until completion. And then after that model is trained, you like iterate again and again and change hyperparameter across different models. So that's the way how you might traditionally do it.

Now your question is more about how do we change things in the middle of training? So one example of – Or two standard sort of configuration settings that you might change during training is the first one is called learning rate and the second one is called batch size. So this is obviously the regime of these like neural networks. But learning rate might basically tells me how fast is my model going to train, right? So that means we have this like iterative – We have, say, this iterative updating process, and every time there's an iteration, I update the model a little bit. It learns a little bit more. And then eventually it gets to a final converged state.

And all learning rate says is per each update of this model, how much do I change the neural network? And if I change it a lot, maybe I could learn faster. But if I change it too much, I could totally just destroy – It could just diverge. The training could diverge. So one characteristic of neural network training is that there are certain times where you want to do like a large update and certain times where you want to do a small update. So that's learning rate. Does that sort of make sense? I could talk about batch size right after this.

[00:09:13] JM: Maybe let's drill a little bit deeper into learning rate. Do you have a concrete example of when you would want a small learning rate versus a big learning rate?

[00:09:22] RL: Yeah, that's actually a great question. Typically, you want – Let's see. You want small learning rate when you're at the end of training typically. And the reason is because you can sort of think of your neural network as, say, like a position on a bowl. And what you want to do is you want – You're standing on the edge of the bowl and you want to get to the very, very center, the very, very middle, which is the lowest part, but it's like the best part to be.

You can imagine if you take a step, a decent down into the bowl, you can go very fast by taking a very large step. But once you're in the bottom, near the bottom of the bowl, if you're taking large steps, you're never going to step in the middle, because you're just going to bounce in and out, in and out of that center piece. And so with this bowl analogy, eventually you want your updates to get smaller and smaller so that you eventually converge into that very center of the bowl.

Now, there's a little bit of art to it, right? Because if you decrease your step too quickly, it's possible that you stop moving when you're still not at the very middle of the bowl, right? You are at like the very edge. And so part of the art of hyperparameter tuning now is all about making sure that you take these steps in like a smart way and eventually you start like taking advantage of this like convergence property that you expect.

[00:11:05] JM: Okay. And there are other hyper parameters. You mentioned learning rate. Another one is batch size. What is batch size?

[00:11:15] RL: Great. So in standard traditional deep learning, say, like when you're training a convolutional neural network, which is one of these very popular deep learning models for image recognition and computer vision. So, say you're training this sort of image neural network, the way you would train it is you first curate a large dataset of millions or thousands of images. Say, they're like cat images and dog images, and the goal is for your model to figure out whether or not you're training of small – Whether or not the image has a cat or a dog, okay?

So the batch size tells me that per each update of my model, how many data points am I going to look at in order to craft this update, right? So somehow there is like some calculus that goes into this process and you can come up with like a decent, like an update value. And this update value is, again, predetermine – Is determined by running these data points that you're sampling from your larger dataset through your model. Checking what your model got wrong, and then calculating this update.

So at very, very high-level, you have this subsample of data points from your dataset. You have your model and then you have this like back – You have this like updating mechanism. And the

batch size simply tells me how big of a subset am I going to take that is going to going to inform this update that I'm going to take next. Does it make sense?

[00:13:09] JM: It does. Yes.

[00:13:11] RL: So the way – What's a good way of thinking about it? Okay. So here is kind of like – The high-level idea is that you want to increase the batch size as you go on training, as you continue training. And the reason is because if you have a small batch size, that means your update is actually a little bit noisy. So the update, the true update is what you might imagine like if you had all the infinite amount of data and you had the perfect dataset, like your model should calculate this – Should be moving in this particular direction. It should be going straight into the middle of the bowl.

Now, the problem is if you only take a very small subsample, it's like your map is very noisy, right? So then if your map is very noisy, you kind of going to go in some arbitrary direction and might be somewhat correlated with the True North, but it's full of noise and it's random. There's a certain amount of randomness to it.

Now, what's interesting about deep learning is that there has been empirical like research being done and showing that it's typically actually better to have a small batch size in the very beginning and a large batch size at the very end of training. So what does that really mean? So that means that at the very end of training, you want to have very precise steps towards your goal and you want to de-noise your updates at the very end.

And on the other hand, it's okay in the very beginning that you have very small batch steps. And I guess maybe the physical analogy to think about is like you're not simply in a bowl, but you're in a landscape and there are ups and downs, ups and downs. There's a lot of like local minima and then there is are very, very deep place that you want to be. But maybe you start like way far ahead, and you might not be able to reach the center of this larger, deeper valley immediately.

So what the research has shown is that if you take these small, like these very noisy steps, the noisy updates somehow guide you into that deeper, wider valley in which after you venture that

wider valley, wider, deeper valley, you can just take – You can be very confident about reaching the very bottom.

[SPONSOR MESSAGE]

[00:15:53] JM: Today's episode is sponsored by DataDog, a cloud scale monitoring service that provides comprehensive visibility into cloud, hybrid and multi-cloud environments with over 250 integrations. DataDog unifies your metrics, your logs and your distributed request traces in one platform so that you can investigate and troubleshoot issues across every layer of your stack. Use DataDog's rich customizable dashboards and algorithmic alert to ensure redundancy across multi-cloud deployments and monitor cloud migrations in real-time. Start a free trial today and DataDog will send you a T-shirt. You can visit softwareengineering.com/datadog for more details. That's softwareengineeringdaily.com/datadog and you will get a free T-shirt for trying out DataDog. Thanks to DataDog for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:16:54] JM: There are other hyperparameters we could mention. Do you want to give an overview of a few other hyperparameters that just illustrate why the hyperparameter sweep and the hyperparameter tuning are important concepts?

[00:17:08] RL: Yes. There's other hyperparameters that you might commonly tune include model size or the shape of your model, the different layers that you're going to use. Or you might choose some algorithm hyperparameters such as learning rate, batch size, etc. And finally there is some – Oftentimes you might actually tune the way you process your data. So you might make some images. You might have some sort of perturbation that you apply to the images so that your model learns better. And those perturbations are usually configured in such a way that there are parameters to be set. So those are typically the wide, like the broad categories of hyperparameters that you might tune.

And of the more complicated things about hyperparameter tuning is that some of these hyperparameters, some of these values might actually be interdependent or related to each

other. So the tricky thing about having very precise and very fast hyperparameter tuning is that you have to tradeoff between – Like optimizing one single hyperparameter at once versus optimizing many. Typically, say you have a bunch of different hyperparameters that you could change. It would be exponentially – It would basically be infeasible to try out every single combination of every single hyperparameter that you have. So it's very much up to the user to sort of narrow down that scope and decide what is the most reasonable space to be searching over, and then maybe just take like the best one that you find from that reasonable subspace as the parameters that you end up using for your machine learning model.

[00:18:59] JM: Can we explore the term hyperparameter sweep in more detail? Could you give a definition and an example?

[00:19:06] RL: Yeah. I would say a hyperparameter sweep we would simply be an evaluation of multiple different hyperparameters. That's like the most simple definition. So an example of this might simply be, say, I have different models that I want to choose. The first model has a size of like 10 megabytes. And then the largest model has the size of 100 gigabytes. And I'll take the same, like I'll take everything else constant and I'll train each model until they converge. And then finally now I have 10 different models with different performances, and what I'll do is take the best one to use for my application.

[00:20:00] JM: So, when I'm training a model, when am I going to be tweaking the hyperparameters?

[00:20:08] RL: Right. I guess is your question more like during the training of a single one of these hyperparameters, like a different model size, when do I change hyperparameter in that process? Or do you mean like when I'm developing a model from like the entire machine learning workflow, where does hyperparameter tuning come in?

[00:20:31] JM: Entire machine learning workflow.

[00:20:33] RL: Got it. Yeah. No, that's a great question. So, typically what you might do is you might collect your data. You build some prototypes of your model and your – Your model, your algorithm, your machine learning a process, right? So you have this prototype, and what you

might do then is you might start to scale it out or you might start to make things like your model a little bit bigger. Your algorithm a little bit more complex. And it's actually a mix-and-match, or it sound like intertwined within the training process. So some people, they take some rough prototype of their model and machine learning training thing and they'll just run a hyperparameter sweep across all the different configurations that could try on this prototype. And because it's a prototype, it's easy to evaluate. It's not so expensive. The hyperparameter tuning job ends up very quick.

Then like the prototype will then – The best configuration and such will graduate, so to speak, into something more complex and they'll just train that thing from start to end. So that's one way of looking at it. The other way of looking at it is you might build out your full prototype. It gets very complex. And now it's so complex you have so many different configuration parameters to tune. Then before you ship something to release, you'll set into a hyperparameter tuning job. This massive job will produce a single model that you can ship off to your service there that that will deploy the model.

[00:22:18] JM: Okay. Now, what is hard about the process of tweaking and tuning hyperparameters?

[00:22:28] RL: So, let's see. Difficulty, as you know, is like sort of dependent on the user, right? So, I guess there is there's a couple ways to answer this. One is that the landscape of hyperparameters, the different configurations that you can try, it's a very none – Like it's not a friendly landscape, right? By definition, this sort of tuning process can take a very long time. It can require complicated algorithms. And a lot of this expertise might not simply be in the knowledge domain of all of practitioners.

I'd say the second big complication about hyper parameter tuning is it's easily scalable and it ends up being that if you really want to scale things out, you dive into a bunch of distributed systems problems, right? And so as a data scientists, right now it's a little bit – I mean, you want to be doing the data. You want to be working with the data. You don't really want to be building out distributed systems and such.

And so, yeah. I guess to summarize my answer, there's the complicated problem itself and then also the complexity around the infrastructure that is needed to orchestrate and execute this workflow that doesn't necessarily reside in the domain knowledge of data scientists today.

[00:24:01] JM: I see. So if I understand correctly, the ideal world is that I have a configuration file that includes my hyper parameters and I can just update these and seamlessly have my infrastructure respond to the updated hyperparameters. The reality is that, today, if I update my hyperparameters, that's going to change the number of cores that are running, and like the type of cores I need, the type of memory configuration I need, which is not like going to be magically taken care of under the hood. It's going to cause bunch of exceptions when I try to run this new model.

[00:24:36] RL: Yeah. I would say, yeah, definitely, like if you're trying out a new model, that's typically one of the biggest problems. The models are getting bigger and bigger by the day, and whatever worked for your previous infrastructure will just like not work next year.

[00:24:53] JM: Okay. But was what I was saying correct? Like if I update my hyperparameters, is that going to potentially make the underlying infrastructure not good enough or not the right infrastructure I need?

[00:25:05] RL: I guess I would clarify that to say, typically, what you might do this you would just make it simple for yourself and you wouldn't actually change these hyperparameter that blow up your infrastructure. If it does, then you realize, "Okay, I'm not going to touch this. It's too complicated for me right now. And let me just do something similar."

[00:25:26] JM: Okay. Got it. What kinds of tooling do we need to build to make hyperparameter training and tuning easier to do?

[00:25:36] RL: Right. So that's a great question. I think the way I would look at this is from sort of like the data scientists perspective. They might be – This person might be using R. They might like barely know Python. And if you ask them to do like large scale distributed hyperparameter tuning, that's going to be like a years' worth of work just simply setting up that infrastructure in a reliable and cost-efficient way, right?

I would say some of the more complicated issues regarding hyperparameter tuning is that, A, because it's so expensive, you want to leverage. You probably won't be buying machines. You'll be leveraging the cloud. And on the cloud, you'd probably be leveraging spot instances in order to reduce your GPU or your machine learning process by 3X. And if you're on spot instances, then you have to deal with fault tolerance, right? That's the first problem.

The second problem then is like if you have this sort of large machine learning sweep, you probably want to have something that takes care of all of the distributed systems, like plumbing, where all of the logs centralized in a place. All of the model, if you're downloading rates or downloading data, you take care of – That has to be taken care of too. And you probably don't want machines to be – Your hundreds of machines to be up at all times, because you may be only running a hyperparameter sweep like once a week.

And so then you run into problems of auto scaling and logging, right? So I would just say, maybe traditionally, fault tolerance, auto scaling, logging, these are all like just some of the problems that someone is trying to do distributed hyperparameter tuning will face and most likely will not have the expertise to do that.

[00:27:42] JM: Okay. Tune is the scalable hyperparameter training library that you have built. Explain what Tune is.

[00:27:49] RL: Ray Tune is a scalable library for parameter tuning and experiment execution. It solves the following problems. So there's three problems that solves. It provides great optimization algorithms. It simplifies the distributed systems aspect and it reduces the code complexity for this machine learning engineers.

Yes. So let me expand and all of these. A lot of the times nowadays, and more so every day, we have these machine learning tasks that are becoming more and more complex and they depend on these hyperparameters that we just talked about, right? And in order to navigate this very difficult problem of optimizing hyperparameters, there's a bunch of algorithms that researchers have developed in order to tackle these.

Now, what Tune provides is a law of many of these algorithms straight out-of-the-box. They're turnkey solutions with easy-to-use abstractions that users can plug and play to their machine learning script, and it will just work out-of-the-box and scale seamless.

The second problem that we address is simplification of these distributed systems, right? A lot of model developers are [inaudible 00:29:01] distributed systems engineers. I think you might be unique in talking to – Like maybe in the RISELab at UC Berkeley, you will have this like both character traits in a couple of people. But I would say like majority of users or practitioners doing machine learning do not have distributed systems backgrounds or even coursework.

So what Ray Tune provides is it abstracts away all these problems of distributed systems that you might need to leverage where you might encounter where you're doing distributed hyperparameter tuning, including, but not limited to, fault tolerance, auto scaling, distributed logging, resource scheduling and such. So on and so forth.

Finally, Ray Tune provides law of utilities in order to simplify the code complexity that might come up when you're doing these distributed systems work. Tune automatically handles check pointing. It automatically puts visualizations and it automatically provides great log files all centralized in one place for users to easily access and work with.

[00:30:18] JM: These all sound like useful features. Why is this a new thing? I mean, we've had machine learning systems in production for many, many years. And at this point, like it's less than a decade, but still that's a lot of time. Why now? Why is there a time for a new system for scalable hyperparameter tuning? Didn't exist in the Spark ecosystem?

[00:30:43] RL: Spark has one of these this really famous library, this MLLib, right? And what Spark is great for is traditional machine learning, where you're working with linear models, random forest, etc. Now, one of the more complicated things about deep learning, which sort of came a little bit after this wave of Spark popularity, or perhaps around the same time, is that deep learning is, A, very compute-intensive. Has a different optimization. It's sort of like a different algorithm providing you opportunities to further optimize, and it finally requires the law of like GPU's, right?

So I would say one of the key aspects that Spark is unable to fulfill is this idea that – Well, let me try to structure this is a bit better. So, Spark works in sort of a bulk synchronous processing mechanism. It executes things in stages. And often times in hyperparameter tuning for deep learning models, that sort of like stage-wise execution doesn't really quite – It ends up forcing you to wait for stragglers and it ends of making it very difficult for you to implement some of these more advanced optimization methods. So you need something a little bit more flexible than large like MapReduce structure, and Ray was sort of built for these machine learning, these fine-grain machine learnig tasks. And therefore providing a scalable hyperparameter tuning system on top of Ray allows us to leverage better optimizations. It's more machine learning ecosystem friendly, because it supports GPUs and it also simplifies all distributed systems problems that you have before.

[00:32:54] JM: So Tune is built on top of Ray. Can you describe the interface between Tune and what sits beneath it?

[00:33:02] RL: Right. Yeah. So that's a good question. So, Ray, for the listeners that don't know about it, is a distributed system mainly for distributing Python right now. And it provides a nice simple to use interface for distributing components of your program. So there is an actor API and there's like a task API, and those allow you to distribute functions and classes in Python.

Essentially, if you have a standard machine learning, or standard Python program, you can compose different parts of your Python program into Ray components. And then now you can scale your Python program seamlessly without having changed much of your code.

So given that we have this actor API and this task API, Tune leverages Ray's actor APIs to run distributed hyperparameter tuning jobs. Each actor, which is essentially a class and could run on different process, different GPU, different node, contains a single execution of this machine learning process, right? It contains the model. It contains the algorithm that you're optimizing the model. And this allows us to easily parallelize the training process and interact with these training processes in a more fine-grained fashion.

So you might have things structured in separate. So you might have a chain process that's structured into multiple stages of a single machine learning model that you're going to train in

multiple stages. And because you have this class interface, you can intercept and do different things in between these different stages. And this sort flexibility that's provided by Ray's actor API presents like a great opportunity, or like provides us a great opportunity to implement these optimizations that you might do in between the stages of a machine learning training process.

[00:35:18] JM: You've touched on this a little bit, but can you go little bit deeper into why actors and tasks are necessary primitives for making Tune work the way that it does?

[00:35:31] RL: One interesting thing is that we actually don't use – Ray Tune doesn't use the task API. It actually just fully leverages the actor API. And the reason why you don't use task is because a lot of these machine learning libraries carry heavy initialization costs. So what they'll do is they'll kickoff the GPU kernel. They will have to like claim GPU memory. And initializing the GPU+, initializing higher-level libraries like TensorFlow or PyTorch, that stuff all takes a lot of time.

And so the reason why task don't quite work for us, is because if you say you did one update, or say you split your training process into multiple stages and you wanted to like execute these different stages in different in different tasks, you would be incurring a huge overhead cost of starting up.

A typical stage of what I'm talking about in the machine learning process might be a single pass over the dataset, and you want to do multiple passes over the dataset. In machine learning, we call that an epoch. And every pass over the dataset, the machine learning model gets better and better. But you now introduce – Because each pass over of the dataset is dependent on – You're training the same model over and over again, it becomes unwieldy to execute each pass of the dataset in a separate task as we described before.

Now, what's unique about being able to separate these stages, these passes over the dataset into separate like execution chunks is that it allows us to modify the trading process during training. So that might include terminating a training process that is underperforming, right? So that allows us to release resources. Train our model and hopefully it will do better. Or it might allow us to perturb the hyperparameters during training. So this gets along the lines of this dynamic hyperparameter schedules that we talked about before. And finally, it allows us to

maybe even allocate more resources to this model this training process, right? So maybe it can train faster or even better because we know that it's a very promising model.

I guess to really take a step back, the actor model allows us to break up the training process into executable chunks while maintaining state on that without incurring the startup overhead. And by breaking things up to chunks, we can insert like better optimizations or hyper parameter tuning techniques to make our tuning process more efficient.

[SPONSOR MESSAGE]

[00:38:42] JM: Code doesn't always behave as you expect, and your team might be wasting time trying to understand why. You can empower your engineers with Rookout. Get the data that you need from live systems instantly and start shipping software better, faster and more reliably. You can visit rookout.com/sedaily today. That's rookout.com/se daily.

[INTERVIEW CONTINUED]

[00:39:11] JM: One thing I want to know is does hyperparameter tuning play a different role in reinforcement learning than in traditional supervised learning processes?

[00:39:26] RL: Yeah. So, hyperparameter tuning and reinforcement learning is tricky. The reason is because the reinforcement learning models, they don't train very well or they don't train very consistently, right? So you might have a – Say, when you're doing reinforcement learning, you might be doing some sort of control task. And by control task, you can think of, say, like a robot walking. And the reason why something might have very high – The robot walking task might have high variance is because it's inherently sequential. And like the amount of reward that you get for the sequential process increases as you progress, right? And it's very possible that because you have the sequential nature of reinforcement learning task, the probability of failure is quite high in between tasks. Or like it's quite high inside this reinforcing during task.

And so as a result, you might imagine like you have – Let's say like you're in the maze, and your reinforcement learning task tells you to get to like the other side of the – To get to one particular destination in the maze. Now, what we can imagine is like you're in the middle of the maze and suddenly you take like a wrong turn. And then you get penalized like super badly for taking that wrong turn. But it was just part of – It was part of like randomness that you took that wrong turn.

So what ends up happening is these tasks often have a very high-variance, and you might be trying to optimize – When you're optimizing the sort of high-variance metric, it becomes very hard for you to do that. In traditional, hyperparameter optimization for traditional machine learning assumes that like these variances eventually – Or like the variance is quite low and the performance of your model can be easily measured. That's not the case for reinforcing learning, and that's why it makes it difficult to do traditional hyperparameter tuning reinforcement learning. Does that make sense?

[00:41:46] JM: It does. So you're saying that because reinforcement learning is it's harder to judge exactly where you are in the exploration space. It's harder to do intelligent hyperparameter tuning.

[00:42:06] RL: People at Google figured it out, was that a lot of times, if you just tweak the hyperparameters in the middle of training, sort of arbitrarily, it's somehow the reinforcement learning model does better. And this sort of technique is called population-based training, but essentially you have this model that you're executing in stages. And after a couple minutes you decide, "Hey, I'm going to increase the learning rate, or decrease the learning rate, and see how that works. And if it works well, then good. If it works poorly, then we're going to terminate it." And somehow they've shown that this population-based training method works very well for our reinforcement learning better than all their methods.

[00:42:53] JM: The tagline for your library, Tune, is scalable hyperparameter tuning. What's hard to scale about hyperparameter tuning? Or what is unscalable?

[00:43:05] RL: Right. I guess, it's again quite – Because every task is independent of each other, it's easy – Like you might think, "Okay, this program is easily parallelizable." I guess what

gets a little bit complicated is that you will probably want to do both distributed training like distributed data parallel training in addition to distributed hyperparameter tuning.

So now you have like a tree of things that you want to scale. You want to scale all the different machine learning jobs, and each of the machine learning jobs themselves want to scale why – Like leverage multiple nodes. And so I guess the tagline really is about as a user, as a person who's doing data science, you can beverage all of these in like 10 lines of code.

One of the magical things about Ray that, or in Ray Tune, is that when you write your code, you write it for a single laptop or a single a single function, or a single core. And suddenly like you turn on like non-GPUs equals a hundred or a thousand, and suddenly you can leverage your entire AWS cluster without having changed their code at all. I think that puts us in the position that, yes, makes it scalable in comparison to what people might be doing today.

[00:44:40] JM: Tell me more about what happens in a distributed experiment with Tune.

[00:44:46] RL: So let's see. In the distributed experiment, we will have – Obviously, you have some driver that is driven. Essentially, it's an event loop. And the events of this event loop are these different training processes, or it's like – The completion of a stage of a different training process. So you can imagine you have a thousand machines. You have a thousand machine learning processes. Each of them is executing one of these stages. It's doing a pass over dataset.

As soon as one of them completes, it will notify the event loop through Ray, and Tune will have certain callbacks within the processing stage that either change resource allocation, change hyperparameters or like terminate the job, or whatever. And that sort of like processing takes place on the driver and is then sent back to that pending task, that pending execution, that pending training process that has just finished their stage.

And so this event loop model allows us to then also detect failures when we had nodes that go down and tell us that we have to migrate a particular training process to another node. Or if more nodes come up because the auto scaling is kicking in, then we will identify, "Hey, look,

there's new resources that we can leverage to place more training processes on." Does that answer your question?

[00:46:32] JM: It does. Now that we're talking about distributing experimentation, let's talk about fault tolerance. Why is fault tolerance important in distributed training?

[00:46:43] RL: Great questions. So I think fault tolerance is most important when your jobs are expensive to train. So what you might – The failure mode here is like, suddenly, during your training processes, it's 8 hours. You've been running your hyperparameter tuning job overnight. And then at the very last minute, Amazon takes away 90% of your cluster, because – I don't know. Like Pinterest is starting to run their jobs in the morning.

And so now you're left with 20% of your cluster and a bunch of your jobs died. And because they ran for 8 hours, it's probably worth $600, $700 of an experiment. And what would really suck is if you had to do that all over again. Ideally, the dream here is that you have a thing that you can just go to sleep and wake up the next morning and you might have had all of your machines gone, but except for one end, somehow your hyperparameter turning job still completes. So, yeah. So I guess the fault tolerant story here is to make sure that you don't lose work and that also you can continue executing when you do have failures.

[00:48:16] JM: This all sits on top of Ray. Can you talk more about why Ray presents good primitives for building Tune?

[00:48:25] RL: Yeah. So one of the things that was nice about Ray is its ability to handle GPUs very nicely. And what that means is it helps to isolate the GPU for a particular training process, whereas typically you might need to set a bunch of different environment variables just to configure and target a particular GPU. And that's only –I mean, the typical processes usually is you sell all these environment variables and you have to do that across a cluster on every single machine. Tune takes care of that for you so that it ends up being like a great feature add – Or like a great usability feature for machine learning or deep learning jobs.

Now, I would say another big thing is that Ray is Pythonic. It's very much targeted towards like the Python ecosystem. So machine learning, also being in the Python ecosystem and plays

really nicely with Ray. Finally, like the actresses system, I think there's a big gap in the machine learning ecosystem for an Actor API. There does exist, but I think the benefit of Ray's actor APIs include like the program model being very simple. And in addition to the ability to leverage shared memory across different – On a single machine across different processes. So there's, I guess, a very unique set combination of features that Ray provides that allows us to easily implement hyper parameter tuning and these distributed deep learning jobs for building a hyperparameter tuning tool.

[00:50:22] JM: Let's zoom out. How does Tune help a machine learning researcher? What value are they going to get out of Tune?

[00:50:31] RL: Right. So maybe the high-level three sentence is scale out your – Like burst out to 100 nodes without needing to think about distributed systems and leverage the large hyperparameter tuning techniques for optimizing your hyperparameters. And finally, you don't need to worry about – Or like take care of downstream tasks such as analyzing your hyperparameter and obtaining checkpoints and logs on like really easy to use solution.

[00:51:11] JM: And what value do you see Tune having in the future? Are you adding additional features and functionality to it?

[00:51:19] RL: I think, yeah. I mean, I would say the two biggest directions that we probably want to go, towards the sort of serveress model of deployment and just going towards a more – Trying to push this sort of elastic nature of tuning and training. So I would say the two visions here would be like you can you deploy – Or you can launch your hundred GPU tuning job from your local Jupyter Notebook or your local book or shell without – Actually your local Python process without leaving your development environment that's local to your laptop. So that's one way of looking at it. The other way of looking at it is pushing this like nature of the elasticity. So, hyperparameter tuning oftentimes requires – It's known to be long, expensive, time-consuming, but I think what if we could do hyperparameter tuning in the order of minutes, right? So you have models that train very quickly. You burst to a thousand GPUs at once, and suddenly everything is done and you didn't need to pay – You pay the same amount as if you had waited 8 hours for the hyperparameter tuning job to finish.

So some the research that we're doing with the RISELab is trying to explore what are the limitations of very fast very, very elastic, hyperparameter tuning and what are the barriers towards like a perfect server this vision for a machine learning and hyper parameter tuning.

[00:53:04] JM: Okay. Well, any other perspectives on the future that you can predict for us? Taking a broader view of the landscape, taking in things like TensorFlow and other machine learning tools, machine learning companies like Weights & Biases, when you zoom out and you look at the context overall, where is the industry headed?

[00:53:28] RL: Interesting question here. Let's see. So, I would imagine there is going to be many services and libraries that are built more towards people who are not machine learning experts, right? So this ends up allowing us to deploy in different applications and perhaps also allows business analysts and business intelligence units to leverage state-of-the-art machine learning tools.

So maybe something like this sort of like no code wave will also reach machine learning. And I would say another big part here is like the ease of – The one I think is just like faster. Things to get faster, right? We have these cloud GPUs that are becoming easier and easier to leverage. In fact, TensorFlow is baking into their training interfaces a cloud endpoint. So now we'll automatically essentially have a sort of this feel to it and you can deploy things or you can execute your training run on like dozens of GPU's on their Google AI platform by simply having added a single parameter to your TensorFlow code.

So I think more of this ability for developers to leverage cloud resources in a very seamless fashion and obviously to scale out their cloud resources I think will be a huge trend in the near future.

[00:55:18] JM: Okay. That's a great conclusion. Richard, thanks for coming back on the show's. It's been great talking to you.

[00:55:22] RL: Thanks, Jeff. This is a pleasure.

[END OF INTERVIEW]

[00:55:33] JM: If you are a retailer, a big sales day like Cyber Monday can make or break your business. If you sell accounting software, the tax deadline day is like your Super Bowl. And if you're a sports broadcaster, then the Super Bowl is your Super Bowl. Every company has days or seasons that are more critical than the rest. If your systems are ready for the moments that matter most to you, then you're going to be doing much better. And that's the theme of this year's Chaos Conf, the world's largest chaos engineering conference. Chaos Conf is a free online conference taking place on October 6th through 8th, and you can register at chaosconf.io.

Ever since the first Chaos Conf in 2018, the objective has been to create a community around resilience and SRE best practices. Attendees range from having a decade of experience to those who are totally new to chaos engineer. And this year's keynote speakers are Gene Kim, and has been a guest on the show several times; and Adrian Cockroft, who's the VP of cloud architecture strategy at AWS. There will be 20 sessions for all experienced levels focusing on the practice of reliability, completing the DevOps loop and how to build a data-driven culture of reliability. You can register for free at chaosconf.io and the first 1,000 registrants will receive a limited edition swag pack. Claim yours at chaosconf.io and be prepared for the moments that matter.

[END]