# EPISODE 441

[INTRODUCTION]

**[0:00:00.0] JM:** Machine learning models can be built by plotting points in space and optimizing a function based off of those points. For example, I can plot every person in the United States in a three-dimensional space, age, geographic location and yearly salary. Then I can draw a function that minimizes the distance between my function and each of those data points.

Once I define that function, you can give me your age and a geographic location and I can predict your salary. Plotting these points in space is called embedding. By embedding a rich data set and then experimenting with different functions, we can build a model that makes predictions based on those data sets.

Yufeng Guo is a developer advocate at Google working on Cloud ML. In this show, we described two separate examples for preparing data, embedding the data points and iterating on the function in order to train the model. In a future episode, Yufeng will discuss Cloud ML and more advanced concepts of machine learning. Unfortunately, we were time constraint in this episode, but it's really elegant. It actually turned out to be a really elegant episode where we discussed this embedding process and the simple model training process that had been very confusing to me for a long time.

Hopefully, this key aspect of software engineering will be made more clear by this episode. I hope you like it.

[SPONSOR MESSAGE]

**[0:01:42.4] JM:** Simplify continuous delivery with GoCD, the on-premise, open-source, continuous delivery tool by ThoughtWorks. With GoCD, you can easily model complex deployment workflows using pipelines and visualize them end to end with the value stream map. You get complete visibility into and control over your company's deployments.

At gocd.org/sedaily find out how to bring continuous delivery to your teams. Say goodbye to deployment panic and hello to consistent predictable deliveries. Visit gocd.org/sedaily to learn more about GoCD. Commercial support and enterprise add-ons, including disaster recovery are available.
Thanks to GoCD for being a continued sponsor of Software Engineering Daily.

[INTERVIEW]

**[0:02:41.7] JM:** Yufeng Guo is a developer advocate working at Google on Cloud ML. Yufeng, welcome to Software Engineering Daily.

**[0:02:47.4] YG:** Thanks Jeff. Really happy to be here.

**[0:02:50.0] JM:** In today's episode, I would like to talk through some elements of machine learning and then talk about how Google is building some tooling for that in the cloud. Let's start with a running example that you use in many of your talks, which is an app that predicts what users want to eat given a phrase.

For example, a user could say American food. Maybe the user should get a hamburger. Maybe the user should get a steak. We don't really know it's just an abstract query, but the idea is that we can use machine learning to figure out how to answer a query for a general type of food with a specific response.

I should be able to say salty snack, and it could recommend potato chips. I can say dessert, and it will recommend a milkshake and it knows who I am and it would recommend a good milkshake based on that. You've talked through that example in so many of your presentations and I think it's worth going over as a nice example to the listeners. Let's first talk through the most naïve version of that food recommendation app. What do you need in order to build this recommendation system for foods?

**[0:03:57.2] YG:** Well, the first version I talked about is perhaps too naïve. It's literally just a string matching system, so it's – if you say fried rice. It will just look up in its catalog of foods available and pick one at random that has the words fried and rice. That might get you some esoteric dish like fried rice pudding, or it might get you something like actual fried rice; chicken fried rice or something like that. I assume we're talking about the first machine learning option here.

**[0:04:26.9] JM:** Yeah, absolutely.

**[0:04:29.5] YG:** Yeah. So basically I talk through the string matching example as a joke, and then that doesn't go so well. The first thing you do is you try out just a simple linear model. That's just the linear classifier. One that we see – we've seen used to a great effect for many, I guess decades now and one that now have deep learning and stuff is coming in and improving upon that. But this first linear model helps just categorize and memorize really what a given user's preferences are and memorize these associations between, say salty food and potato chips.

**[0:05:09.7] JM:** To build this recommendation system, we might want to use what is called an embedding space to map different elements. What is an embedding space?

**[0:05:20.6] YG:** Yeah. An embedding space is intended to be a way to represent the relationships between in our case the objects or items on our food menu, so to speak. What it does for us that's better than say just having a raw list, literally just say an array of all your different food items, just all in a row, all your ducks in a row is that you can get this kind of, almost a spatial relationship between what is similar and what is more different. So the further apart something is, the more different it might be. We see this a lot with text processing, where similar words are clumped together and words that mean things that are very different from each other are farther apart.

**[0:06:05.0] JM:** We did a show about Word2vec, which is an embedding space for words essentially. The idea with Word2vec is if you take all these words and you plot them on a geometric space, just all the words in a corpus, you would have words where they would have these relationships for example, king would have a geometric relationship in this in-dimensional space to queen and it would have the same geometric relationship that male would to female.

If that sounds really confusing, go listen to that Word2vec episode. But basically, the idea is that given enough data, you can use these fairly naïve ways of plotting relationships between words and you can get a really good semantic understanding for what people mean whey they say one thing and you're looking for synonyms for what they might be meaning. How would building an embedding space for foods work? How would you do that?

**[0:07:17.3] YG:** Like you said in my talk to self, that particular example we go through using tenser flow is that basically it's just a call, it's just an argument in our construction of the model. We'd pick a dimensionality for the embedding space and we just say like 8 or to 16. There is not much more literal work involved in terms of programming that. But conceptually what's happening is all these items are just being scattered in this in-dimensional space, let's say 8.

As the model trains on the training data, it learns these relationships between different things and it basically starts moving each items – all items relative to each other. You could think of it like if you had a space and you were getting all these foods streamed into you and you're saying, "Okay, strawberry goes here," and then you get like pineapple, and you're like, "Okay, that's a fruit. Let me put in the other strawberries."

You get one that's like a beef steak. It's like, "Yeah, that's really different. So I'm going to just take it over here." Cake comes in, bread comes in, and you put them in different places, and as things come in you are allowed to move them from their original positions. So then you can make adjustments and say, "Oh, I messed that up." I need this actually be between this two other items, so I'm going to just shift that all the way over here and that's how the embedding space gets built out over time as the training happens.

**[0:08:45.9] JM:** Let's say we have a bunch of foods in a text file. I don't know, maybe they have some fields that describe the different foods. How are we turning these fields into numerical dimensions? Because we need to plot these foods in an in-dimensional space. What would some of those dimensions be?

**[0:09:11.9] YG:** That's a really good kind of phrasing Jeff. The actual dimensions themselves are not necessarily always interpretable, which leads to various note-around interpretability of machine learning models and like – but the parameters you have and usually when you're training the system, you also have some kind of goal that you're kind of training it to work, but the various parameters, let's say how can we describe this characterizing things are always tricky, right?

**[0:09:46.2] JM:** Of course.

**[0:09:47.4] YG:** You can take each column of your data, each of these features like how salty is it, how soft is it, how crunchy, yes or no, and turn them into numerical features rather string categorical ones. You can say the hardness of a piece of hard to soft based on like 1 through 10 or 1 through a 100 depending on your level of granularity want. Things that are just yes or no, can be zeros and ones.

Once they are converted into numbers, then the model can then take that and build the mathematical relationships, because with just words, there is no numerical relationship. They can't add and subtract words, but once they turn into numbers, they can be worked with, not only more quickly on a machine that's thinking on numbers, but also it becomes a tractable problem.

**[0:10:37.9] JM:** Let's say, we've got all of these different foods that have been plotted in terms of their crunchiness and their softness and their savoriness, and I'm getting hungry just talking about this. Then maybe we have some of that data that's labeled. Let's say, a billion people have – or I guess for the purposes of querying, we have some labeling thing where it's like from on a scale of 1 to 10, how much did this food satisfy your query, so that people who have gone through and queried this system and they've gotten some return – or maybe we don't even need that querying data in order to build this correctly.

Okay, we have this data that has the crunchiness and the saltiness and so on. We can use those to plot these foods in an in-dimensional space where you've got every food is plotted in terms of saltiness and savoriness and so on, plotted in an in-dimensional space. What are we doing with that embedding space in order to build a querying engine?

**[0:11:47.5] YG:** Once we have this, and I want to emphasize that we're not plotting this manually. We're not literally taking each of our potentially thousands or millions of food items and trying to position them by hand into a place. This is done during the train process automatically by the machine learning process.

When you have, let's see. When you're doing the training and let's say in our example, we have labeled data that we say people prefer when they ask for salty food or crunchy food and they get potato chips or something. With that data, you can then iteratively train your model. When a model has the embedding stuff, but it also has whether it's a linear network or a deep network, the representation of this embedding space is simply – it's just a way for the model to hold the data in some form that's not just a raw list of values.

By having it an embedding space, it allows it to understand more like we talked about before between like king and queen. It learns these relationships where it can almost "understand." When I say understand, I guess it really is that we looked at the model of humans and said, "Oh, wow. Look." It like created this mathematical relationship, so we say it understands, but it doesn't understand like in the brain sense of it. Yeah, does that sort of answer your question?

**[0:13:18.7] JM:** And so it explains the relationships between them, so we've plotted these relationships. What exactly is going to happen then once we have those relationships in geometric space when I ask a question, or when I say something like American food?

**[0:13:33.9] YG:** Right, right. This is now we're getting to the other side of machine learning, right? I always think of machine learning as the great divide of "training and prediction." Having this data or examples and answering questions. The answering questions side is what makes machine learning useful, this making predictions.

You're absolutely right that when you then ask for American food or you ask for seafood, one way to do it would be and conventionally you could look at just basically saying in the embedding space, you have these clusters of different types of food groupings, if you will. If you say American food, you have a bunch of American food, like in your embedding space, you can think of it like in space, there are all these different American foods and it would "just pick one."

But with a trained model, it has also learned your individual preferences. There is that additional nuance, wherein it can pick on based on your preferences, beyond just saying, "Okay, I'm just going to pick a random item off of the menu of lots and lots of American foods out there." There may be and probably should be other criteria involved as well when you are training the model of things, like time of day; what do you have before, where are you located, things like that.

[SPONSOR MESSAGE]

**[0:15:01.5] JM:** DigitalOcean Spaces gives you simple object storage with a beautiful user interface. You need an easy way to host objects like images and videos. Your users need to upload objects like PDFs and music files. DigitalOcean built spaces, because every application uses objects storage. Spaces simplifies object storage with automatic scalability, reliability and a low cost. But the user interface takes it over the top.

I've built a lot of web applications and I always use some kind of object storage. The other object storage dashboards that I've used are confusing. They're painful, and they feel like they were built 10 years ago. DigitalOcean Spaces is modern object storage with a modern UI that you will love to use. It's like the UI for Dropbox, but with the pricing of a raw object storage. I almost want to use it like a consumer product.

To try DigitalOcean Spaces, go to do.co/sedaily and get two months of spaces plus a $10 credit to use on any other digital ocean products. You get this credit, even if you have been with DigitalOcean for a while. You could spend it on spaces or you could spend it on anything else in DigitalOcean. It's a nice added bonus just for trying out spaces.

The pricing is simple. $5 per month, which includes 250 gigabytes of storage and 1 terabyte of outbound bandwidth. There are no cost per request and additional storage is priced at the lowest rate available. Just a cent per gigabyte transferred and 2 cents per gigabyte stored. There won't be any surprises on your bill.

DigitalOcean simplifies the Cloud. They look for every opportunity to remove friction from a developer's experience. I'm already using DigitalOcean Spaces to hose music and video files for a product that I'm building, and I love it. I think you will too. Check it out at do.co/sedaily and get that free $10 credit in addition to two months of spaces for free. That's do.co/sedaily.

[INTERVIEW CONTINUED]

**[0:17:20.3] JM:** Let’s jump to another example. Let’s go to the example that you give around census data, and this will motivate further how to build the model, then we’ll talk through building a model. Given a census data set with things like marital status and age and education, these numerical fields. The question is can we build a model that predicts whether each person has $50,000 or more a year in income?

All of these data is labeled. Let’s say we know the salaries of all the people. So we’ve got marital status, age, education, we’ve also got their salary. But we would like to be able to predict without labels. We would like to have a situation where even with unlabeled data where we just have marital status and age and education, we can predict what their income is whether they produce $50,000 a year in income or not.

So much of the work involved in training this model is in preparing the data for input. Let’s say we’ve got this giant CSV of data. We’ve got a data set and we want to prepare the model for the training. Describe what happens in this input function? This is sometimes described as data cleaning or data preparation. What goes on if we just have this giant CSV of labeled data?

**[0:18:43.9] YG:** Absolutely. This data cleaning, or setting up the data for machine learning definitely ends up being the bulk of the work. You see that in the code, just like most of the code is setting it up, and then running them and learning is just like the couple of minds at the bottom. When you first start out, let’s say in our data set and in the example that we’re talking about, even the labels have are not 01. They’re actually just the string field that is like it says greater than 50k or less than 50k.

It’s literally, the >50k or < = 50k. So we need to process that into a 0 or 1 process them into classes. If we had a situation where there was more than one class, let’s say it was – we wanted to segment it into three different buckets of income, so just above or below 50, then you would classify those as 0, 1, or 2. Giving these indexes typically a 0 based indexes.

Then you would also perhaps want to do some transformations on some of your other features, some of your other columns. For example, you might look at the age column for instance and realize that we don't need to have literally the fine graininess of age of every single age number, like 24, 25, 26, all the way up, and that it would be more useful to have age buckets, age groupings, because we care about the 18 to 24 age range and what kind of income do they tend to make? That can oftentimes lead to more useful insights than being so granular that the noise starts really affecting our model.

You can bucketize your data in that way. That converts your data in that particular column from being continuous, like age is just continuous value, to being categorical, where you just have a handful of different categories. Oftentimes, the question comes up around even something like age where it's value that's like sort of both. You can look at it both ways.

You could say, "Well, age is a number so it's continuous." But you could also say, "Well, there is only probably no more than around a 100 different possible values, so that's categorical too." At their limits, they tend to blur a little bit. Typically when we think of categorical, we're referring to generally on the smaller side of things that has been pushed. Then continuous on the far reaches of it, it's like just truly continuous numbers, like rational numbers. It can be any decimal number, but oftentimes you can use bucketization or things like that, techniques to turn a continuous column or feature into a categorical one.

Having those kinds of operation – and you can also do feature crosses. Let's say you had a column around education and a column for occupation. So you might look at your data and you say, "It's just back there. There might be some correlation between the highest degree someone degree and what occupation they're in to help predict income, right?" That's not entirely unreasonable, but you don't want to do the statistical analysis to figure out the relationship between these two columns.

You can just do make a third feature called age – not age. The occupation cross with education. It's asking the question of given that someone got to this level of education and they work in some field, what will they be above or below $50,000 in annual income? Having these additional parameters can help round out your model, especially in situations where maybe you don't have as many columns.

This allows you to get some more columns out of your data and allows your model to take advantage potentially of some other ways of looking at your data, because the model itself doesn't understand. Let's go back to the age example. It doesn't know that you're okay with bucketizing the age column. That's why it can be useful to provide that upfront and that will make the model oftentimes more accurate and you can also make it easier or fast to train. Things will converge better.

**[0:23:05.4] JM:** Yeah. Yufeng, I am obviously not an expert in machine learning. Tell me if I'm wrong about what we're doing here. We are plotting all of these data points, these rows across each of the different columns. We're defining values for each of these columns, because some of them might be weird. They've got a text input and you need to figure out what value would be assigned to a text input, numerical value, because you're plotting them in Cartesian space. So you're plotting them on a numerical space.

The reason that you're plotting them in this in-dimensional space is because you're going to use these plotted points to describe as data points to interpolate a function that goes through as many of these data points as possible. With that function, you will be able to accept new data points and see where those would plot given the function that you have defined over the training data points.

Because there are many different functions that you could describe that would go through these data points, that's what the total training process is about you're saying, how am I going to train? What kind of function do I want that – and what do I want them optimized for? Because maybe under certain circumstances, your function would hit, would pass through 50% of the data points, and other functions it would pass through 30% of the data points, but you would be emphasizing some particular aspects of those data.

Maybe you're more likely to cross through the salty regions of in the food example, because you think that people happen to always want to optimize for salty foods. Am I describing this process that we're towards accurately?

**[0:25:06.2] YG:** Roughly, yeah. It sounds like you must have a very particular math background to be bringing up all these functions. The machine learning models, especially when it comes to the more recent known networks and things like and even linear models can often be thought of as trying to approximate a given function.

There is a couple of nuances there. I wouldn't necessarily say that we're trying to have the function of pass through as many points as possible. That's generally not the objective or goal of the training, but rather that if you were to measure the distance away from all the different data points, how far are they from the function you have chosen to plot?

We summed up all of those distances, how far off we are? Let's say average then or something. Then we want to try to minimize that value. We try to get as close as all the things is possible, exactly that's oftentimes referred to as loss, sometimes it's called error. The danger here is where when you have data that's maybe a little bit noisier and things like that, if you actually create a function that passes through every single data point, it's just going to be going all over the place, right?

If you ever have a data set in Excel and instead of having the curve plot through a nice fitted curve, but to actually just draw set line segments between every single point. It's just this jagged all over the place value. If you were to just interpolate, you might what used to be a trend line that just goes nice up smoothly, but now is up and down, up and down, up and down and for any given point, it might be going down or it might be going up and you just get all sorts of wacky values in it.

**[0:26:52.9] JM:** Okay. Let's say we've plotted all – let's say we had these billion data points of people in a census data set where we got marital status, age, education and we also had the labeled data, whether they had $50,000 in income or not and now we get some new – Actually no. Let's say we wanted to iterate over. We want to do machine learning. We've got these example points plotted. We want to do machine learning. Describe what happens in each training step? How does training work once we have these examples to plot?

**[0:27:29.1] YG:** Yeah. What we take then is we take each training step. So we're going to train this system iteratively as you mentioned. At each training step, we're going to grab a batch of our training data. Instead of trying to determine how to adjust our function – so the goal here is each step, we're going to adjust their function slightly, hopefully to make it so that the loss is a little bit lower than it was in the last step. So that's the overarching goal is how can we iteratively adjust the loss, such that it decreases with each training step?

Then we say, okay ideally we would be able to figure out how to adjust their function by looking at our entire data set. We would take our, in our case, let's say a billion rows of data and we would figure out how – by running it through the existing model, that's like let's say step one we would figure out, okay we need to adjust the function in these ways to make the loss a little bit lower. Then this next step, step two would be it would take same billion training data and you would come back and you would adjust the function and make the loss a little bit lower again.

There is a practical limitation here in the taking a billion data points just to do one training step, because you might be doing thousands of training steps is the reality here. Computing a billion values and then adjusting a function just to get a little bit of change oftentimes it's not worth it.

What you do instead is you take a representative batch of the training – let's say you just take a thousand data points, you just take a random sample of a thousand data points. The idea here is that those thousand would be representative of the entire data set. So we say, "The thousand, it's good enough." We'll use that to adjust their functions slightly, and then we'll check to see what our error is and then we say, "Okay, we did a decent job. Let's grab another representative thousand, a different thousand and adjust the function against that."

By doing it this way, you had become so much more practical and computationally tractable problem, where you can actually train for many, many training steps, thousands of training steps without having each step take minutes or hours at a time. Now you can do a thousand training steps in just a couple minutes or something.

**[0:29:55.0] JM:** Okay, that's great. Can you define the term hyper parameter tuning?

**[0:29:58.1] YG:** Yeah. As we go through the process of training, there is a lot of parameters that we end up, sometimes arbitrarily picking. Really when we talk about embedding spaces I said, "You can just do 8 dimensions, 16 dimensions. However many you want." Or just now when we're talking about the batches that we do in training, I just said we grab a thousand, but what if we grab 2,000? What if we grab 500?

These parameters are typically referred to as hyper parameters in our model, because the actual parameters is the function. It's what defines the function of the model, so hyper parameters that's through hyper in front of anything and become fancy, right?
You can adjust these values and they will affect your training. If you embed your data in 8 dimensional space versus 16 dimensional space, depending on the complexity of data, that can have a real material difference. Sometimes the batch size or sometimes when you're training, you'll go through your entire training set multiple times and loop through it a lot, and that kind of affect how strongly the function is fitted to the data set, whether it's just a loose, smooth it, or does it become more tightly bound to all the existing training data than that kind of get us into over-fitting and stuff, and we don't have to necessarily go with that way.

That's what hyper parameter tuning is, it's to improve, and the whole purpose of it is to improve the final outcome. Whatever objective function you have, where you're trying to improve some accuracy, or you're trying to improve the – decrease the loss. Doing hyper parameter tuning means you're just running through the training cycle with different training parameters, so to speak; batch size, the dimensionality.

Which features you start tinkering with, bucketizing which feature, crossing which features together, those are also decisions that you make that will impact the outcome of the training and you can make those adjustments to end up with different outcomes. So that's kind of what hyper parameter tuning. You can explore these parameter space, if you will.

[SPONSOR MESSAGE]

**[0:32:06.6] JM:** The Octopus, a sea creature known for its intelligence and flexibility. Octopus Deploy, a friendly deployment automation tool for deploying applications like dotnet apps, java apps and more. Ask any developer and they'll tell you that it's never fun pushing code at 5 PM on a Friday, and then crossing your fingers hoping for the best. We've all been there. We've all done that.

That's where Octopus Deploy comes into the picture. Octopus Deploy is a friendly deployment automation tool, taking over where your build or CI server ends. Use Octopus to promote releases on-prem or to the cloud. Octopus integrates with your existing build pipeline, TFS and VSTS, Bamboo, Team City and Jenkins. It integrates with AWS, Azure and on-prem environments. You can reliably and repeatedly deploy your dotnet and java apps and more. If you can package it, Octopus can deploy it.

It's quick and easy to install, and you can just go to octopus.com to trial Octopus free for 45 days. That's octopus.com, O-C-T-O-P-U-S.com

[INTERVIEW CONTINUED]

**[0:33:38.5] JM:** We're optimizing. We've got these set of data points that are plotted in space. We're trying to figure out what is the best function that we're going to build across this space and that – in changing that function over time, we're optimizing for something. Maybe we're optimizing for accuracy. Maybe we're optimizing for the lowest loss. We're trying to minimize loss.

Can you contrast those two things that we could be optimizing for and how those would change our model tuning over iteration? By the way, the things that we're optimizing are the hyper parameters, right? Those are basically the knobs that we're tuning overtime to create this model.

**[0:34:26.5] YG:** Yeah, that's a very good way of putting it. They're the knobs that we can turn in our model. In terms of accuracy and loss, the notable difference here is that accuracy is it's exactly what is sounds like. If you had a 100 different data points and you got 99 of them correct and one of them wrong, you have a 99% accuracy.

On the other hand, loss can be thought of as a little more nuanced, because with accuracy you don't really capture how wrong you were. In some models, it's not just above or below a certain threshold, right? It might be you need to predict which one of these 30 categories something falls in, or you might need to predict how a large – or you may need to predict a real value, a number needs to come out.

Measuring accuracy may not be as meaningful there, because you might not be able to even hit a number right on the head, especially if it's like, the goal that you're supposed to hit is 37.2 and you predicted 37.1. So you're wrong. But actually, you're quite close. So loss is a little bit more nuanced. It can capture the distance, if you will, from what you predict versus what the true value is.

That is often, like in terms of choosing whether you want to use loss or accuracy as your metric for thinking about how well your model is doing, it comes down to your particular use case, so like using the examples that I gave there is one way to think about that. The other aspect of it is that we talked a lot about the training and the training data so far and strictly speaking that you would also actually want to cut off a piece of your existing data set and hide it away, and never show your model and that data.

We'll use that for, it's called either validations, somebody with a evaluation or your test data. Basically, you never show your model of this data, so that you can then later present this data to your fully trained model as a way to evaluate how it did, how did the training go. Because if you train your data on these thousand data points and then just present those same thousand data points to your model, your model could very easily get away with just memorizing those thousand data points, and then it wouldn't be able to generalize well to more real-world data that comes along down the road.

You have this evaluation data set that's representative of all the other ones, but isn't exactly the same. And we also know the "correct" answers to those values. We show the model that and that's the evaluation step and that's what we used to evaluate accuracy or loss or whatever, because that's our metric for figuring out how did our training go, how well did we do.

**[0:37:11.4] JM:** Okay. I think I understand things pretty well. But we're talking about deep learning too. We haven't really said the word layer at all in this conversation.

**[0:37:25.0] YG:** Yeah. Somehow we managed to time away for that. Yeah.

**[0:37:28.3] JM:** We're talking about this fairly simple process of relatively speaking of plotting a bunch of data points, drawing function through those data points and then testing that function against new data and tuning it over time to optimize for accuracy, or to minimize loss. Where does the layer conversation fit in? Where does deep learning fit into this?

**[0:37:54.2] YG:** Yeah. So everything we've talked about – a lot of what we talked about so far is just more general machine learning concepts. It's almost agnostic to what model you're working with. Deep learning is one particular way to do machine learning, and basically at the simplest level you can think of it as you're taking classic linear regression and stacking a bunch of basically layers of linear regression.

So we say, we're going to use a linear classifier or regressor and do one. But then we're going to take the results of that and just not actually use it as our answer, but instead feed it into yet another layer of "linear regressions." We'll do that a couple of times. So you have these hidden layers, if you will, before you reach your true output that you're using to predict against your training data. So you have this kind of a lattice structure or webbed structure.

Each layer has typically a different number of nodes, if you will, or neurons as it often gets called. But the resulting structure kind of sometimes looks like a river delta almost, where all these rivers are converging and then there is one now that, like if you look at a satellite photo of the Nile for instance.

Then the reason you want to use these layers and why would they be useful is the natural next question, right? What makes it better than, or oftentimes I should say, sometimes better than linear regression if you have really clean data is not always the case. But would deep learning generalize this better, because between each layer, there is what's called an activation function.

The activation function acts as a way to introduce a non-linearity to the model. This means that while before we were only able to learn linear relationships, now we can stack these linear – basically stack these linear models on top of each other and you at first would say, "Well, that's still linear right?" Some of a bunch of linear functions, it's still a linear function.

What we do then is we say, "Aha, we'll add in a little bit of a non-linearity in between each one." Traditionally, it was a sigmoid, and now it is a lot of them use what's called a rectified linear unit, which is literally just a horizontal line from negative infinity to zero, and then that zero it's just a 45 degree Y equals X line. It just goes straight up. No, not straight up, but diagonally.

That kink in the functions to speak, right at 00 is enough of a nonlinearity for the network then to adjust its parameter such that it utilizes that, because that function, what it's going to do is it's going to wipe out any negative values. All the outputs from the previous layer of going into the next one is going to filter out all the negative ones and it's going to map all the positive ones directly forward, because it's just Y equals X.

By shifting how each layer it can shift its parameters to cause which aspects to fall into the negative half versus the positive half. Then if you have a bunch of different layers, it can do that repeatedly and that can basically act as a gaining function, so to speak between the layers and introduce those nonlinearities and produce highly complex functions to fit any data set, given a large enough network.

**[0:41:20.2] JM:** Okay, I think we've given a really good explanation for how to train models. Unfortunately, we're up against time so we can't go super deep into Cloud ML like I wanted to, so maybe we should do another show in the future. But let's talk a little bit about that. Let's talk about deploying these models. Let's say whether we've created this food recommendation model, or this census data classifier. How are we deploying these to the cloud and maybe you could tell me something about what Google has built with Cloud ML.

**[0:41:55.3] YG:** Sure. So with Google's Cloud ML engine, it's basically a managed service for running your Python tensorflow code in the cloud at scale. It really comes in handy when you have a really big data set and also really big compute needs, which typically – that go hand in hand, and you don't want to deal with a hassle of managing infra both for the storage of that data, as well as the compute side for actually doing that training.

Instead, what you can do is you can just take your existing tensorflow code that hopefully you've tested out locally on a smaller subset of your data and you know it's working and it's things like that. You push that up to the cloud, you package that up into a Python package and you use Cloud ML engine to run your tensorflow code for you.

What it does is it basically will provision all the virtual machines that it will run it on, network them together and install tensorflow and whatever packages it needs, and then push all your training data through that and run the training. At the end of your training, you're going to want to export your model, because like we – I touched on briefly, we touched on the whole notion of training and prediction, how important the other side is.

The whole point of all these training is to answer questions. It's to do something with this model. Not just to train with a good accuracy and say, "That's it." To then do the predictions, we want to deploy our model to production in some form. .ML engine helps with that as well. It has a whole prediction service, and what you can do is you can literally take the exported model from tensorflow, which if you do it in a cloud it's just a couple of files that sit in a folder.

You just point prediction service at those files, give your model a name, just as a text and just label it and you're basically done. You end up with an auto-scaling rest API that can handle just Torrance of predictions. Not only does it scale up automatically for you, it also scales down all the way to zero if there is no one calling it.

In a lot of ways, I see that side, the prediction side as being like the real awesome feature of this, because once you're done doing your training, you've already ranked all of your data, you spend all this time and energy writing a code, now you have to go figure out how to deploy this service to production and make sure it's secure, make sure you have a good API and make sure all these things that it's just like I just want to get back to my data and to my model somewhere.

Having a really simple way to deploy a model to production is a really awesome feature of ML engine. With these exported models, I will also add that you can do the training locally. Say you have a cluster of machines and you run your tensorflow code, you do your training and then you export your model, so you just got some files on your hard drive. You can upload those files to cloud storage and at that point you can pick up in the middle of those steps where you just point the prediction service at those files and you have another scaling API. You don't necessarily have to do training and prediction in the cloud. You can do one or both. You can do one or the other or both.

**[0:45:08.8] JM:** Okay. Yeah, I wish we could go deeper on that, and we should definitely do another show in the near future, because we uncover nearly all the stuff I wanted to cover with you. Because some of your talks you go into detail in to how to deploy this stuff and some of the other features that you could be doing with Cloud ML.

Let's save that for another time and wrap it up here. We did a really good job of covering the model training and the basic Cloud ML deployment stuff. So thank you Yufeng. Thanks for coming on the show.

**[0:45:37.8] YG:** Yeah. Thanks a lot Jeff. I guess, the only thing I'll add here is that in terms of other resources and materials, recently I've been putting out a set of videos, a series called Cloud AI Adventures. So that's a series that's available on YouTube, on the Google Cloud YouTube channel. If folks want to learn more about both like machine learning, as well as Cloud ML and just the entire data science ecosystem, the subtitle of that show is To Explore the Art, Science and Tools of Machine Learning. That's been really fun to make and produce. I hope folks can get a lot out of that.

**[0:46:15.4] JM:** It sounds like great content for me to watch. Let's get some on the calendar for another show and I'll consume some of that content before we do the next show.

**[0:46:26.9]YG:** Awesome. Sounds good. Thanks so much for the opportunity. It's been really fun chatting with you.

[END OF EPISODE]

**[0:46:33.4] JM:** Who do you use for log management? I want to tell you about Scalyr, the first purpose-built log management tool on the market. Most tools on the market utilize text indexing search. This is great for indexing a book, for example. But if you want to search logs at scale fast, it breaks down.

Scalyr built their own database from scratch and the system is fast, most of the searches take less than a second. In fact, 99% of the queries execute in less than a second. That's why companies like OkCupid and Giphy and Career Builder use Scalyr to build their log management systems. You can try it today free for 90 days if you go to the promo URL, which is softwareengineeringdaily.com/scalyr, S-C-A-L-Y-R. That's softwareengineeringdaily.com/scalyr.

Scalyr was built by one of the founders of Writely, which is the company that became Google Docs. If you know anything about Google Docs' history, it was quite transformational when the product came out. This was a consumer grade UI product that solved many distributed systems problems and had great scalability, which is why it turned into Google Docs.

The founder of Writely is now turning his focus to log management, and it has the consumer grade UI, it has the scalability that you would expect from somebody who built Google Docs. You can use Scalyr to monitor key metrics. You can use it to trigger alerts. It's got integration with pager duty, and it's really easy to use. It's really lightning fast and you can get a free 90-day trial by signing up at softwareengineeringdaily.com/S-C-A-L-Y-R, softwareengineeringdaily.com/scalyr.

I really recommend trying it out. I've heard from multiple companies on the show that they use Scalyr, and it's been a real differentiator for them. Check out Scalyr and thanks to Scalyr for being a new sponsor of Software Engineering Daily.
[END]