EPISODE 306

[INTRODUCTION]

[0:00:01.2] JM: Deep Learning uses neural networks to identify patterns. Neural networks allow us to sequence layers of computing, with each layer using learning algorithms such as unsupervised learning, supervised learning, and reinforcement learning. Deep learning has taken off in the last few years, but it has been around for much longer.

The history of deep learning is quite interesting. What makes this time unique? Why is deep learning starting to become popular? In this episode, we get into that. Adam Gibson founded Skymind, the company behind Deeplearning4j. Deeplearning4j is a distributed deep learning library for Skala and Java. It integrates with Hadoop and Spark, and is specifically designed to run in business environments on distributed GPU's and CPU's.

Adam joins the show today to discuss the history and the future of deep learning. This was a very enlightening episode for me, because I am not much of a deep learning expert. It's one of the topics that I really wish I was more well-versed in, and today is hopefully a step towards that direction.

I would love to do more shows on deep learning, and if you know somebody who would be a good guest on this topic, I would love to hear from you. Send me an email. I hope you enjoy today's episode.

[SPONSOR MESSAGE]

[0:01:25.5] JM: You are building a data intensive application. Maybe it involves data visualization, a recommendation engine, or multiple data sources. These applications often require data warehousing, glue code, lots of iteration, and lots of frustration.

The Exaptive Studio is a rapid data application development studio. Optimized for data projects, it minimizes the code required to build data-rich web applications, and maximizes your time spent on your expertise. Go to exaptive.com/sedaily to get a free account today. The Exaptive

Studio provides a visual environment for using back end, algorithmic, and front end components.

Use the open source technologies you already use without having to modify the code, unless you want to, of course. Access K-Means clustering algorithm without knowing R, or use a complex visualization, even if you don't know D3.

Spend your energy on the part that you know well, and less time on the other stuff. Build faster and create better. Go to exaptive.com/sedaily for a free account. Thanks to Exaptive for being a new sponsor of Software Engineering Daily. It's a pleasure to have you onboard as a new sponsor.

[INTERVIEW]

[0:02:55.8] JM: Adam Gibson is the founder of Sky Mind, Adam, welcome to Software Engineering Daily.

[0:03:00.2] AG: Thanks for having me.

[0:03:01.4] JM: This conversation is going to center around deep learning. In past shows about deep learning, I feel like I have gone a little too granular sometimes, and then other shows I've gone too high-level. You have a lot of experience teaching deep learning to people, so I'm hoping that we can hit the right level of abstraction.

If you were explaining deep learning to a front-end engineer or an operations engineer, someone who is technical, but doesn't have any expertise in machine learning, how would you describe deep learning?

[0:03:35.0] AG: At the end of the day, its machine perception. What is deep learning good at? It's good at working with human generated data. What does that mean? Images, text, audio, media. Why is it good at that? It's because the neural net is able to learn its own patterns. You have pattern recognition at its core.

Machine learning is also pattern recognition, though, but deep learning is just a subset of field of machine learning that specializes in — normally, deep learning can refer to other things, actually. Go read the papers, without getting too much in details here, but mainly neural networks though.

When you have a neural network, you have inputs and outputs. A neural network, I'll just give you something you can Wikipedia later, is a universal approximator. It's capable of mapping arbitrary inputs to arbitrary outputs. What we're able to do then is to stack those, stack a neural net, a series of inputs and outputs, into layers.

What that allows us to do is it allows us to kind of learn a compressed representation of our input. Such that we can map it more accurately to outputs, where we can start then making generalizable assumptions, such as that's a cat or that's a dog.

[0:04:50.5] JM: Yeah. Each layer is one core abstraction that is learning something, that is processing something in an input, and outputting some result. Give an example of what a layer might be, and how we compose these layers together to do productive things?

[0:05:08.7] AG: At the end of the day — let's actually step back a second. The most basic neural network is actually, let's just say logistic regression. I think most people, if they have a vague interest in this, probably know the basics about regression.

You have coefficients you're trying to optimize for. What you do then is you have a set of usually, let's say four or five coefficients, or however many inputs you have, and your goal is to basically slap a label, yes or no, based on these coefficients, based on an input.

What you do is you have this idea of gradient descent, where what you're trying to do is you're trying to basically minimize errors. You repeatedly show a number of inputs, and it says yes or no, this is right, this is wrong, and you give it feedback. In that feedback loop, the guesses gradually get better over time.

What you're essentially doing is you’re learning, based on learning this input, you basically build an algorithm that can then dynamically say yes or no, based on random input. At its core, it’s a very simple problem of basically beating around a bunch of weights. Kind of like ping pong.

You’re beating around a bunch of weights, giving it feedback, those numbers gradually get better and better, and eventually those numbers basically represent a function, actually. That function can map “cat or dog” to “yes or no.”

[0:06:33.1] JM: Right. Now, when we’re talking about these different layers that we serially stack to create this neural network, what components of this are supervised and what components are unsupervised? Because you have supervised learning, which requires us to have labeled data, and you have unsupervised learning, that does not require us to label the data?

[0:06:57.8] AG: What you’re doing there is — building on my regression example, why deep learning is powerful is now, all of the sudden, imagine taking that input and then adding another layer, and then adding another layer. You’re basically doing [unintelligible] regressions at once, and each layer basically learns from the previous layer’s output.

What you eventually get is a more abstract output, very similar to what Support Vector Machines do, where they kind of remap the input space on to a new space, where it’s easier to make a decision. Building on that, most neural networks are supervised. All of them have this idea of back propagation.

As I mentioned with gradient descent earlier, where you’re basically — you’re kind of playing ping pong with coefficients, to basically build a better approximator, and then stacking those things. That’s what back propagation is doing is it allows you to stack these things, and then basically optimize the whole function end to end at once.

When you’re doing unsupervised learning, that’s where you’re starting to get into simply auto encoders. Back in 2006, Restricted Boltzmann machines, nowadays, generative adversarial networks, right? There’s always some form of a neural net that can basically teach itself how to reconstruct the original input, or how to approximate it. You have loss function that basically maps input to output, and maps reconstructions onto – maps, basically, a reconstruction of the original input, based on the neural network's output.

You basically compute the difference. You can mix these things. You can use auto encode. What you used to do in deep learning is, you still sometimes do this, you mix unsupervised and supervised. The whole innovation back in 2006 was unsupervised pre-training. That's where you basically have a neural net that learns to reconstruct the original input layer by layer, and then you basically built — then what you do is you build a classifier on top of that. You only have the final layer, as you only train the final layer.

Actually, we use a form of this today for purely supervised learning, where you take pre-train models, like image net models, and you basically only train the final layer. That's called transfer learning. What you basically do is you chop off the output layer, and you just basically — you kind of just kind of attach your problem onto a pre-trained neural network.

Why does that work exactly? It actually goes back to unsupervised pre-training. It's actually good enough, it basically starts in a good enough location that's very easy to start learning, it's basically able to learn about your problem very quickly, because it starts from a good location.

What I mentioned earlier was those coefficients, right? That's the core of it. Those coefficients basically represent a location, and basically, the better those coefficients get, the more accurate they are. Both unsupervised and supervised do a form of gradient descent. They both kind of bat around coefficients to approximate or basically map an input space on to an output space.

You're kind of doing both. The core mechanics are the same though.

[0:09:52.3] JM: Are you doing — at each layer, are you deciding, "At this layer, I'm doing a supervised learning operation, at another layer, I might to do an unsupervised learning operation," or — okay. Now, there's also reinforcement learning, which is a goal-oriented form of learning, and in reinforcement learning you define a model in terms of an agent, and an environment, and the interactions between the agent and the environment can produce positive or negative progress towards a goal.

You have this reward function that you're optimizing for. If you're trying to teach a model to win at a video game, like let's say pong, you mentioned pong earlier. Let's say the goal might be to increase the number of points. If the model, if the agent just takes a bunch of random actions and then tries to learn which ones of those actions are going to correlate with an increase in points, then the agent is learning over time how to achieve those points just through sampling its random actions, is that accurate?

[0:10:55.6] AG: Yes. Basically, that's called a policy. You're basically learning — reinforcement learning has this idea of a policy, where you want a policy, and that policy is basically the reward function, the heuristic that says, "this is good, this is bad." When you kind of like, think of it as you're kind of teaching an agent pain.

It touches a soft surface, and it's good, so you reward it. Then it touches like, an electric wire, and it gets shocked. That's negative. You basically attach plus one, minus one to an agent that explores a space.

[0:11:29.6] JM: Now, is this another thing that would define what is going on at the layer level? Would you have either supervised learning, unsupervised learning, or reinforcement learning? Or is reinforcement learning something that overlaps with…

[0:11:42.3] AG: The power of reinforcement learning is that it's not necessarily defined by a gradient. It's actually a more open-ended problem. That's why deep reinforcement learning, combining the two has been so successful. It's because it basically — the idea with deep reinforcement learning is, you basically have a more open-ended space you're exploring, but then being able to map it on to a gradient space. That's a lot easier to learn and use.

The gradient space is more stable, the reinforcement learning is more open-ended, which is why you combine the two. Reinforcement learning, at its core, is just basically a search problem that allows you to search in a non-differentiable way. Differentiable meaning in terms of calculus.

[0:12:20.9] JM: Now, in combining reinforcement learning with supervised learning, if we're still talking about pong, how would that work? What would they be doing from the point of view of reinforcement learning, and how am I synthesizing that with supervised learning?

[0:12:39.0] AG: Reinforcement learning basically sets the label. It sets a direction on the gradient. Usually what you have is you just have — in supervised learning, you just have a static-state space, right? 0 or 1, yes or no. Then you back propagate the error. What it guessed was right or wrong, right?

In reinforcement learning, you're basically having the reinforcement learning set the policy for the neural network, and that determines how it learns. Kind of what you're doing is you're having an agent search a state space, and then you're having the neural network learning how to approximate the agent's actions.

[0:13:10.1] JM: Got it.

[0:13:10.7] AG: You're controlling the direction of the neural network with reinforcement learning.

[0:13:14.9] JM: Okay, let's talk a little bit about the history and where we are now. What are the origins of deep learning?

[0:13:23.1] AG: Realistically, neural networks have been around for 50 some odd years now. Actually, most of our — a lot of the innovations, for example, AlphaGo for example, that actually descended from an older technique, Monte Carlo tree search, right?

Basically, a lot of what we're doing nowadays is we're reusing stuff we did in the past. We're just retrying it and combining it with some of today's techniques. This stuff has been around a long time. You know, it's not anything new. It's kind of peak hype, but at the end of the day, what we have now is just better hardware, but the techniques have been around quite a while.

Obviously, I think 2006 is kind of where Geoff Hinton, and the Restricted Boltzmann machine, and unsupervised pre-training, and landmark results in computer vision. That was kind of the first paper, I think, that came out that started using the terms deep learning. You know, because like CNNs have been around for a long time. Basically, Canada, the Canadian Research

Institute over there, they were just chugging along on neural nets while we all were not paying attention, and that's where all this kind of really started.

[0:14:30.6] AG: Okay, you said we're in peak hype, but you know, when I look at deep learning, I see a field that has, at this point, whether it may be — like you could argue that the processors have been kind of an inflection point that have made this more practical, but the overall trend seems to be as important as something like cloud or mobile, where somebody would have said, "This is peak cloud," five years ago, that probably would not have been peak cloud. Because we're still — now it still feels like we're kind of in the early game of cloud. Do you really feel like this is like a hype thing or — because it feels to me like we're just getting started?

[SPONSOR BREAK]

[0:15:25.8] AG: Couchbase is a NoSQL database that powers digital businesses. Developers around the world choose Couchbase for its advantages in data model flexibility, elastic scalability, performance, and 24x365 availability to develop enterprise web, mobile, and IoT applications. The Couchbase platform includes Couchbase, Couchbase Lite, which is the first mobile NoSQL database, and Couchbase Sync Gateway.

Couchbase is designed for global deployments, with configurable cross data center replication to increase data locality and availability. Running Couchbase in containers on Docker, Kubernetes, Mesos, or Red Hat OpenShift is easy. At developer.couchbase.com, you could find tutorials on how to build out your Couchbase deployment. All Couchbase products are open source projects.

Couchbase customers include industry leaders, like AOL, Amadeus, AT&T, Cisco, Comcast, Concur, Disney, Dixons, eBay, General Electric, Marriott, Neiman Marcus, PayPal, Ryanair, Rakuten, Viber, Tesco, Verizon, Wells Fargo, as well as hundreds of other household names.

Thanks to Couchbase for being a new sponsor of Software Engineering Daily. We really appreciate it.

[INTERVIEW CONTINUED]

[0:16:54.0] AG: Okay, I wouldn't be in business in this if I didn't believe in it, but we have to be realistic about where things are right now. There's a lot of misconceptions in the media, despite the researchers, great people actually advocating for, "No, we're not building cognizant AI."

I think the problem we're running in to right now in the field is, you know, we're basically doing kind of recognition. We have computers that can pick out diseases, we have — they're doing very cool things, they're playing games, but at the end of the day, you basically — I love the way someone described this to me, basically, de-boarding is good at problems where you can frame it as an image of some kind.

You can frame it as a static state space that's learnable. That's not cognizant AI, right? People keep — the problem is, is people keep trying to map it on to that. The analogy of the brain and cognizant AI, they're actually pretty dumb still.

Even like, the dynamic memory networks out of Metamind, and some of the stuff coming out of Facebook, where they're like remembering. Where they have memory. The neural touring machines out of Google, that stuff's all awesome, but it's not cognizant. It's just more fancy pattern recognition with a little more concept applied.

It looks cool, but it's not realistically applicable to most of today's problems. It's going to take — the expectations are just way too high for what we have right now.

[0:18:21.1] AG: Why do you say that? It seems like this is just like a field that is growing in accessibility, it's growing in popularity, and it's almost like the exposure or two, the cross section of like, where we are just — like today, defining the deep learning primitives. What you said about, "Okay, you have to define this as an image." If you could define the problem as like, an image classification problem, or an image problem, or whatever you just said, then you can start to make progress, and it's like we're only starting to get the language for discussing these problems.

The knowledge sharing, for example. I was looking at the Deeplearning4j website, which is an open source project that you are working on, we'll talk about that in a moment, but it was like the first resource I looked at that — I've looked at some other ones that are good, but yours was extremely concise, extremely good. I was like, "Well, the knowledge sharing is really starting to get good for deep learning," and I do — I guess maybe you're right about the hype cycle.

It is hyped in the sense that there is a diff between where many people have the perceptions of where the current day is, versus where we could potentially get to. The ceiling for where we could potentially get to with deep learning is, does seem very high, right? Am I mistaken?

[0:19:40.9] AG: No, you're not at all, actually. What I just want to say is the gap with the media versus what we can realistically do is just, that's one of the problems constantly plaguing the field still. That's really all it is. It's going to be a constantly growing field with a lot of innovation happening.

I would say deep learning is actually still highly volatile yet, in a lot of areas. There's a lot of open ended research problems, there's still a lot of unsolved problems, there's still a lot of room for innovation, there's even still a lot of room for just applications.

I think the problem is people are putting deep learning on a pedestal, saying, "You have to have a PHD to do it. Everybody's hiring the talent, and I'm never going to be able to do it myself."

[0:20:19.9] JM: Right, let's go there. When you were first getting started with deep learning, that impenetrable assumption was kind of valid, because the way that you got started was reading dense academic papers. My impression is, that is the only way to get started back then, but the field is opening up.

What approach would you take to understanding deep learning if you were just starting today? If there's an engineer listening to this, just getting started, what approach can they take today?

[0:20:46.3] AG: Well, if anything, I would recommend the course by Jeremy Howard, the fast.ai that just got announced recently. The massively online course for that. That's if you want to start directly with deep learning, I wouldn't advise you to do that.

Pick up your fundamentals, statistics, and linear algebra. Following that, do machine learning, and then do deep learning. One of the things that people do is they jump into deep learning too early, because they think, “I can just do this out of the box, no problem, without understanding any of the concepts.” Don’t treat it like a black box, and actually try to understand the basic components, and then from there, it’s actually fairly easy to jump onboard and start solving problems.

[0:21:21.9] JM: I want to get in to the discussion of the deep learning frameworks. This will get us towards Deeplearning4j, but bring us back to that earlier discussion we’re having. Take me through the workflow of a typical programmer that’s building a deep learning model?

[0:21:38.1] AG: At the end of the day, what you’re basically doing is you need to start with a data pipeline. This is what most people don’t know. That’s actually where you spend 80% of your time, literally. Cleaning data, getting it to a form that a machine learning or deep learning can understand. This is actually why I say you should know machine learning, because you spend your time learning those things when you do machine learning.

Granted, you’re not going to learn this as well as you should in the classes. The classes give you clean data and have you focus on the math, which in my opinion, is a big mistake. You really should learn even just basic data pre-processing and cleaning. Because you need that in order to build an actual algorithm, but after you get a good data pipeline going, and you understand how to set one up, then you basically need to build the model.

You take your model, let’s just say a basic Multi-Layer Perceptron, and you say how many inputs does my data pipeline output? What do I mean by that? By taking raw data like a CSV, and you have four columns. But maybe you don’t need your fourth column, so maybe only three. Your data pipeline outputs three columns.

Then you say, “From my neural network, I’m going to have three inputs.” Then from there, you build out your neural network. If you’re just doing the classification problem, like a simple yes, no, then you have two outputs, yes and no.

You basically have to then, based on your domain and your data pipeline, you build your model, you define the basic model, and then you basically just start running it. You basically just start tuning from there. What do I mean by that? I mean, applying the concepts and the math. That's where you're basically monitoring the tuning, you're basically monitoring the tuning of the neural network over time, and then figuring out something that will converge fairly well.

From there, then you go in to production. That's when you're attaching your model to an application, where you can input an image, upload an image, and get out cat or dog. It's actually — there's a lot of engineering, and that's actually a big thing that's not discussed.

A lot of frameworks tend to hand label that. They don't spend time on how do I actually build a product with this? Which is what most people want to get to.

[0:23:40.4] JM: Absolutely. Given the description that you just gave of the process of building a deep learning model, from the data cleaning process to the end process, where you can actually identify a dog or a cat, and have an improving model over time, how have the tools used to build deep learning models changed in the last few years?

[0:24:02.8] AG: The landscape has evolved from research towards product. That being said, I would say 80% of it is still research oriented. People are building products with it, but a lot of it doesn't get to large scale use or traction. You can actually correlate a lot of the — this is my personal opinion, but I have vested interest in saying this, but you can almost map deep learning frameworks to startups.

They have a very similar idea. Like a lot of them basically look at Madbits, look at a lot of the acquisitions in the space, deep mind, a lot of these things, like they have a research focus. They build a really cool product and then they sell. Deep learning is I think still kind of in its early stages there. It's the same thing, where you have research oriented towards research with other researchers, talking to each other about research.

Deep learning is still an echo chamber. The frameworks are the same right now. They're all very homogenous, they're on Python, most of them are in Python, and it's all just people trying to optimize for publishing papers.

What does that mean? That means throwing away code. You prove a concept and you move on. The frameworks have optimized for flexibility, which is great. You know, if you want to build a product with it, and some frameworks you're trying to solve this, not just us, moving to production or whatever, trying to build a product with it, but it's only been an incremental move. There still haven't been much effort in that space by a lot of the framework.

We're seeing a gradual move towards industry with some applications, but mostly optimizing towards flexibility and research yet.

[0:25:41.1] JM: Yeah, it does make sense, and when you talk about industry, the language that comes to mind is Java. I read, I think it was a *Wired* interview with you, and I don't know what was correct in that interview and what wasn't correct, but the interview focused a lot on the Hadoop ecosystem, and something that has not changed in the last five years or so is that large volumes of data are often stored in Hadoop, particularly in HDFS, the Hadoop Distributed File System, and some people will say that Hadoop is going away. What they really mean is that Hadoop MapReduce is going away, HDFS, the Hadoop file system, distributed file system will be around for a very long time.

Can you describe the machine learning ecosystem that has built up around Hadoop, and in what sense is the programming language ecosystem around machine learning, how much is it closely tied to java in the Hadoop ecosystem?

[0:26:40.3] AG: You know? Let me back up and define industry.

[0:26:43.5] JM: Okay.

[0:26:45.2] AG: There's various versions of industry. What I'm about to say is, most of it, most of people trying to do deep learning and doing it in industry, are actually startups. Most people, most startups have a Python stack, usually. They're optimizing for getting a quick, minimum, viable product out the door. They only have large scale data. They don't need it, and the people are going to hire the researchers who know how to do deep learning, or maybe they did a little bit of research engineering at a company, or whatever.

They're going to have a Python stack. 80% of the time, they are going to have a Python stack. They don't have terabytes of data, they don't' need it, they don't need all the overhead, they don't like compiled languages, usually, they like — no JS, Ruby/Python, right?

If anybody's doing deep learning in industry and they're likely to write a blogpost about it because it's cool, it's going to be a startup. I would say that's where most of that — well they need to hire people, they need to do marketing, they need to differentiate themselves. The incentives are very different. Whereas enterprise, no one wants to do actual deep learning for enterprise.

Because 80% of it is bureaucracy, no one wants to do Java. Java is the number one language in the world. That's what people in the Midwest, people in the south, people in the United States, they're doing java and .net.

[0:28:02.1] JM: It's blue collar programmer.

[0:28:04.8] AG: That's enterprise now. That's the Hadoop ecosystem, that's the people using Hadoop. Sometimes you grow to a bigger company, and then you start using Hadoop. AirBNB, Stripe, a lot of these kind of unicorns out there now, they adopt Hadoop eventually. Most of the time you don't need it.

At enterprise, if we just focus on the Hadoop ecosystem, the tooling there is Python interfaces to something written in java, or Skala, or something on the job of virtual machine. That's where most of the stack is. That's all going to be enterprise.

[0:28:37.7] JM: When you say that…

[0:28:39.6] AG: It's all stored there.

[0:28:40.5] JM: When you say Hadoop, you are talking about HDFS right?

[0:28:42.8] AG: Yes. Hadoop, HBase, all that. Yeah, anything to do with HDFS. Back to your point about MapReduce. Yes, MapReduce, people are moving on. There's multiple kind of distributed execution engines. Spark, Flank, what was it?

[0:29:01.3] JM: What's the Google one?

[0:29:03.9] AG: DataFlow.

[0:29:04.6] JM: Beam, or whatever?

[0:29:05.3] AG: DataFlow, yeah.

[0:29:06.6] JM: Beam is an interface of various distributor execution engines. That's where you're getting into Spark, Flank, Apex was the other one I was thinking.

[0:29:15.1] JM: Yeah, okay, that's right. Sorry.

[0:29:16.9] AG: A lot of those are moving towards streaming now.

[0:29:20.1] JM: MapReduce being definitively batch.

[0:29:23.4] AG: Right. MapReduce is definitively batch, but you have these more flexible execution engines that are basically functional style, data manipulation DSL's, right? How do you think Spark started? They took Skala and made it distributed. The default data structure was an RDD. That's what you manipulated.

Most people are moving away from MapReduce. The biggest vendor in that space is Cloudera. Cloudera does quite a bit with spark, they've even kind of declared MapReduce dead. That being said, there's a lot of hype around that, because even today, there's still a lot of problems with the in-memory platforms as well.

Some people actually still use MapReduce for certain things, because they only need batch and the data center is too large. There's even a lot of variance in the big data tooling space. We could spend days talking about that as well.

[0:30:12.8] JM: Well let's spend a few minutes. I've done a ton of shows about streaming, versus batch, versus new things, and there's also this question of online machine learning that has come up in a number of interviews recently, where ideally you would like to have a machine learning model where you just give it one training example and it updates the entire model, but in actuality, that's hard to do.

You end up having to batch your training data to the model. Is that the problem that you're discussing with the in-memory systems, and why they sometimes have to use Hadoop MapReduce? That's the problem.

[0:30:48.8] AG: It doesn't actually even have to do with anything with machine learning. Most people using these systems don't even knew machine learning. Most of them count things. They're not doing machine learning. Machine learning is again…

Machine and deep learning, that's what's talked about the most, but 99% of the world's not actually doing it. It's only what's talked about in the news.

[0:31:05.1] JM: Of course.

[0:31:06.2] AG: Most large enterprises, in this case we deal with banks, Telco, things like that, they do things in Excel still. They're doing things in Excel, they might have some java programmers. It's rare, right?

[0:31:19.6] JM: I'm sorry, I should have just asked, what are the problems with those in-memory systems?

[0:31:25.9] AG: Right, the in-memory systems in general, they let out — stability is the biggest one. Stability and reliability and full tolerance are still a huge problem. That's something MapReduce still solves fairly well.

[0:31:39.5] JM: Pretty automatic.

[0:31:41.7] AG: It's a well understood problem, and people don't need innovation. A lot of enterprises don't actually have very complex requirements in that space. They just need to run a job, it gets done, and they move on.

It's usually use-case specific. Whether you deploy spark or MapReduce or whatever, it's usually use-case specific, it depends on who you employ, you might have a bunch of MapReduce engineers. Maybe for most things, they're fine and then maybe you'll hire a few spark people for your in-memory stuff eventually.

What you then do is you have a gradual migration. You don't have — we're just going to move everything over at once and spend six months doing this. That's not realistic for most companies.

[SPONSOR BREAK]

[0:32:26.8] 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 you can visualize them end to end with its value stream map. You get complete visibility into and control of your company's deployments.

At gocd.io/sedaily, you can find out how to bring continuous delivery to your teams. Say goodbye to deployment panic and hello to consistent predictable deliveries. Visit gocd.io/sedaily to learn more about GoCD. Commercial support and enterprise add-ons, including disaster recovery, are available.

Thank you to GoCD, and thank you to ThoughtWorks. I'm a huge fan of ThoughtWorks and their products, including GoCD, and we're fans of continuous delivery. Check out gocd.io/sedaily.

[INTERVIEW CONTINUED]

[0:33:39.3] JM: Since we're talking about deep learning, machine learning, whatever. Getting back to that topic. Apache Mahout has been around for a while. This provides machine learning on top of Hadoop. How have people used apache? What did Apache Mahout offer, because eventually, we'll get to talking about Deeplearning4j shortly, and this is a deep learning framework for Java. So maybe you could talk about Mahout and what it does and what it doesn't do?

[0:34:06.1] AG: So Mahout has deprecated a lot of things. They've actually moved towards being a winner algebra engine now. So they've modernized, they have a scale IP idle, all sorts of stuff. At its core, the best thing about Mahout with the use-case was mainly recommender engines. It had a good enough collaborative filter that worked at large scale. So that's what I see most people using Mahout for, even still today, because you need sparse data structures, you need a lot of storage, you need a lot of RAM.

Mahout has traditionally been very good at collaborative filtering, and matrix factorization, and that kind of stuff. I didn't see a lot of people using a lot of the different algorithms. Some people would build logistic regression, or classification or whatever, but actually what a lot of people would do is they built – it's actually kind of an open seeker enterprise. You actually write your own machine learning libraries.

If you want to scale, you typically wrote your own. You might have started from open source, you eventually write your own cold source stuff that works with your stack. That's what most people who did things in Java do. So Mahout, mainly recommender engines, a lot of people will mostly disrupt their own cold source stuff though. That's why you don't see having a lot of users or support. There was a community around it, but there was never one big commercial backer, not a lot of traction.

That's why most people move to MLlib was because of that. It was the first thing that came along, was more usable, more oriented to from a machine learning, and solved the problem that you needed machine learning. You need fast in-memory for that, traditionally, otherwise you just wrote your own.

[0:35:38.7] JM: An MLlib is Java.

[0:35:40.1] AG: MLlib is written in Skala, actually.

[0:35:41.8] JM: Skala, okay. So okay, I think we framed the problem well enough at this point and we can get to the solution. So you've said that many of the frameworks that people are doing deep learning in are Python. This are things like TensorFlow, Theano, Torch, KERIS. So I was looking at your documentation, and KERIS can be a bridge. You can bridge between – well, it's a DSL. Okay.

[0:36:10.2] AG: Domain Specific Language for Deep Learning. So when you think of the library people are using, to me, a library is just an interface to a set of GPU's and all these other things, so I can count KERIS on that. Mainly because — I want to push KERIS a little bit, because, well one, I'm biased. We integrate with it as kind of a Python interface, but also because it's a way to interact with several of the lower level frameworks, like TensorFlow or Theano, or not PyTorch yet, but that might happen.

PyTorch is a similar interface, but actually KERIS came from Torch. I should clarify that, and now you have PyTorch. There's a continuous evolution, but a deep laying framework is mostly using Python or just DSL with a lot of math. So KERIS is what most people use on Taggle, for example. I would consider it as the thing that most people would probably be familiar with.

[0:37:01.8] JM: Okay. So anyway, there's lots of Python, and you want to be able to integrate with Hadoop, and Spark, and the Java ecosystem. Will you explain what is the motivation for Deeplearning4j? What is Deeplearning4j? What are the philosophies behind it?

[0:37:19.0] AG: The main thing with Deeplearning4j is first class integration with the Javian ecosystem. So what does that mean? That means you're guiding and central IT can control the configuration from Java, from the Java virtual machine. The guys maintaining the production pipelines can still code in Java, so that's a big win for them, because their Java tech has been around for 20 years, code that was written 20 years ago in Java works today still. Java has done a tremendous job of backwards compatibility, among other things, and so that also means security.

So deep learning frameworks don't know what Kerberos is. They don't have guaranteed integrations with vendors. A lot of Java code is written by companies, and it's proprietary. There's proprietary connectors, and all sorts of other bureaucracy involved with Java-based systems, but that's what people deploy and trust.

So when you think about integrations of thw Hadoop ecosystem, usually there's other vendors involved. Again, I'm biased in the space, though, because we integrate with those vendors.Wwe talk to those vendors, but that's realistically what we see. You see a supported version of Hadoop out there, some people roll their own, that's rare though, but those guys if they roll their own, they have security. They have best practices in place, and they're familiar with it.

So in the Javian has stood the test of time. You have a lot of innovations, like just Google "JRuby and Graal," it's the best time compiler out there. So people see speed, they see stability, they see liability, and they understand it, and you can hire for it. That's the other thing, you can actually hire for it. So for us, we're the framework aimed at them. So our retraction does not come from research. It comes from enterprises moving from Python to production.

So they already have a production Java stack, the architecture or whatever, once they do this deep learning thing, but they see adoption as risky. Then they see us, they instantly get it. They instantly understand what we're trying to do. So what we do is we take Python code from KERIS, and then we can run it straight in the JVM with all the same bells and whistles that you're used to, and the best part is, we're using Java.

The number one thing is Java is where the data is stored. So you don't have a Python thing, talking to a Python thing, with a Python garbage collector, and a Python run time, talking to a Java thing, with a Java garbage collector, and data moving back and forth. You don't have that. That's typically what Spark does. Instead, what you have is you just have a Python DSL that talks to something in Java, and then Java does the rest. Java does the storage, it does the compute, it does basically runs everything.

So that's what I mean by first class on the JVM. So they have something they understand, they have something that they can code with. If you have data engineers, data engineers already know Java, and it's very straightforward to port things from Python over to Java. So the main thing there is familiar environment, stable run time, and data in compute, co-located with where things are stored.

[0:40:26.6] JM: So when you're talking about enterprise adopting Deeplearning4j, because they want to adopt deep learning and they've got some pre-existing Hadoop pipeline infrastructure, basically, so you mentioned that most of these companies are just doing counting, for example, which is an operation that Hadoop solved quite nicely.

So is it basically like okay, they've got this counting pipeline, maybe they've got some richer queries built on top of Hadoop, like some Pig stuff. Or some, what's the other one? Hive, right? Or some Hive queries built on top of it, and then they basically want to take the things that they've already got in Hive and Pig that are written on top of Hadoop, and they want to build some richer learning facilities on top of those pre-existing lines of code. Is that right?

[0:41:19.2] AG: Actually, no. It turns out, a lot of people use HDFS as an archive. So they have lots of videos and pictures, and they don't know what to do with them. So you have all these enterprises who've just been destroying all these data all these years.

[0:41:33.5] JM: Oh, I see. Interesting.

[0:41:34.2] AG: And they're looking to monetize it now. That's why they want deep learning, actually.

[0:41:38.6] JM: Fascinating, okay. So we're just talking about basically HDSF, like all the API's into HDFS are Java, I guess, or…

[0:41:50.8] AG: Right, there's other interfaces in other languages, like Python, like you can access the data, but just accessing HDFS isn't enough. There's a lot of configurations you have to do. If you have to get one level, you'll eventually end up in Java anyways.

[0:42:04.6] JM: Oh.

[0:42:04.9] AG: Yeah.

[0:42:05.2] JM: Right, okay and so this is why you would just want it, want everything in Java, because if you build within Deeplearning4j, Deeplearning4j can do some of the heavy lifting beneath. If you just have a function call in Deeplearning4j, under the covers is going to do some complex calls into Java stuff that is HDFS, but that isn't going into HDFS, is that right?

[0:42:31.4] AG: Right. Yeah, so basically, what you can do is it's embeddable. So the whole point of this is that when you run a job, when you run a distributed job, everything is going to end up on the JVM anyways. So what you can actually do is you can actually — you basically have access to everything from the job itself. So you know your in-memory, you know you're in the same process, there's no IPC — that means Inner Process Coordination — that needs to happen.

There's no client or anything, everything is just in place. So that means, by default, everything is going to be faster. One thing that people don't realize is in deporting, everybody hypes the GPU and how long it takes to compute everything else, but in reality, most of the time is actually spent accessing the data. So it's accessing its storage from disk, right? And so you actually have to design everything around that. The whole pipeline around it is actually going to be your bottleneck.

[0:43:21.0] JM: And so…

[0:43:21.5] AG: Especially when you're in production. You're not spending most of your time running math. You're spending time, alluding to your discussion earlier, setting up the batch job, setting up the batch size, configuring that properly, making sure it's fault tolerant, you're doing everything else around that. So it's a lot easier to do that from Java.

[0:43:40.7] JM: This is bringing some other things into context for me, because I've done some shows recently about Apache Arrow, which is this data sharing protocol, basically, but it allows you to share data between Python and Java. Now I'm starting to understand this in a little more context, because if you have your deep learning stuff in Python, and you're accessing HDFS under the covers, then you're going to have to do data transfer between your Python memory and your Java memory, I guess. That's burdensome and you would rather just have it all in Java, is that right?

[0:44:20.5] AG: Right. Well, or use Python for RPC only. That's what a lot of people do is if anything is optimal, they're using Python for RPC. So it's really easy to write RPC code in Python and then have Java execute all of it.

[0:44:33.7] JM: Okay, interesting. So on the Deeplearning4j website, there's the quote, "This is the first deep learning framework adapted for Microservice Architecture." What does that mean? What is the intersection between deep learning and Microservices?

[0:44:49.3] AG: Okay, well what is a Microservice? It's basically a standalone web API. So you have a standalone service, so you're going to deploy an image recognition service. So most micro framework adoption has actually come from the JVM. Java, Skala, that ecosystem. That's where a lot of Microservices are happening and go, but I would say if there's enterprise adoption on Microservices, pushed by Pivotal and some of these other bigger vendors you don't normally think about when thinking about deep learning, that's going to be Java.

So we actually have the best facilities for taking something from Spark or what have you, like a pipeline, and embedding that in just a web service. So it's then easy to spin up a web service, for production, out of the box, where everything just runs without Spark or this other stuff. One of the things I want to stress here is we're embeddable. We even run on Android. There's a lot of frameworks that do now.

It's not a defining feature anymore, but the fact that you can write an Android app with us, based on a model that you trained on the Cloud, and deploy it to the phone or deploy it as a Microservice, all of a sudden you have the ability to have your production system to be in one language that you're used to, have it run and deploy in a way you understand, a JAR file, and deploying it to some form of platform as a service or what have you.

There's a lot of infrastructure around making Microservices easy, like Docker, Containers, Spring Boot, everybody has their own Microservice thing they do. There's a lot of infrastructure around that, and the Python ecosystem can't leverage that because they have to do IPC. So what I mean by that is this in process, it's in memory already so you can control everything. Your Microservice has direct access to the model in the same process, which is huge.

[0:46:30.1] JM: Makes a lot of sense. Okay, so we're running up against time, and I want to totally zoom out and talk about something totally different. Since you spend a lot of time in the deep learning space, you are exposed to the hype, as you've already said. What are your thoughts on AI risk? Is this something that we should be thinking about at all as engineers?

[0:46:54.2] AG: Yes, very much so. Not quite to some of the extremism that's out there, where AI is going to kill us all, but you know it's more about…

[0:47:03.7] JM: Tail risk.

[0:47:04.3] AG: Where is this going to applied, and how should we set up the system here? So in our case, the way we do things we do a lot of fraud detection, we do a lot of compliance stuff. We're in a very different space from your typical deep learning applications, and what I can say about that is, Human-in-the-Loop AI is key. Deep learning as an assistant is a great idea, because you have machine perception, being able to sift through terabytes of data and being able to more accurately pinpoint bad guys, or what have you, right?

So AI as an assistant is a great idea. I mean look at how miserably Chatbots have failed. They're still not that good yet. The best companies are working on this problem, but natural language understanding is still very hard for most people yet. Unless you can have restricted domain, and so Chatbots, I think, is the easiest analogy to this, but basically, what we need to think about is what is this AI responsible for? How are we using it? So it's more about this being pragmatic about the expectations of AI and where it's being applied. The return on investment for having choices be automatic.

An easy way to think about this is you need to set a confidence interval, and then you need to figure out, "How confident am I in this? What happens when this goes wrong? Is it okay?" Right? So I think that's the major thing. It's more about responsible design decisions.

[0:48:22.9] JM: Okay, interesting. That's totally different than what I expected when you said yes. So you're not at all thinking about the Terminator case. You're thinking about the case where you have an AI in your fridge, and you design the model wrong, so it sends your refrigerator into super low temperatures and freezes everything.

[0:48:47.2] AG: Right, yeah. That's where AI is today. We're not even thinking about Terminator. To me that's not even a real — It's a problem we should be thinking about, but to me, that's more of an academic thing. It's not something we should be worrying about today.

[0:49:02.9] JM: What do you mean by the academic thing, though?

[0:49:04.6] AG: It's abstract. It doesn't exist. It won't exist for a long time. It's a research problem.

[0:49:10.7] JM: So are you a fan of the OpenAI approach?

[0:49:14.7] AG: I've actually done quite a bit with OpenAI. So I know Greg and all these other guys, I've been to their campus, we're a combinator startup. So actually saw OpenAI, and when they were first starting and all that, and we've contributed to their gym project as well. Their Java SDK is ours, so we contributed to that. I'm a big believer in everything they're doing, and I think somebody should be thinking about the long term. So personally, I love their approach and what they're doing in part. It's actually pragmatic, which is unusual.

[0:49:48.6] JM: Yes, so frame that pragmatism a little bit more. What makes what they are doing pragmatic, as opposed to perhaps other people decreeing that the sky is falling, or the sky is going to fall, or something?

[0:50:02.4] AG: Right, so the major thing there is they're publishing a research. It's still small yet, but they're publishing research. They're opening everything so people can assess for themselves where is this at, what can this be applied to, because somebody is going to build this eventually. So having it out in the open is a better idea, because then, with more transparency…

[0:50:22.4] JM: What is “this?” When you say somebody is going to build “this,” what are you talking about?

[0:50:25.4] AG: The AI agent that will do something to us or whatever.

[0:50:29.5] JM: The general AI.

[0:50:30.6] AG: Yeah, the general AI, exactly. Cognizant AI, what have you. Somebody is going to build that. So taking this long term approach, having everything in the open, and doing applied research that moves the needle in the space. Opening a universe for example, they have all these open stuff, and they’re facilitate — they’re enabling the masses to do it. That’s interesting, because it opens up new applications, and it also keeps us grounded and realistic as to where things actually are.

[0:50:56.1] JM: Now the general AI manifestation, I’ve heard several hypothesis about how this is going to eventually happen. Some people will say that this is going to be an ensemble of very specific AI’s, that maybe eventually your Google home has enough API’s that are soldered onto it that it accommodates the terms that we will define for general AI, or it will be something much more definitive, where deep mind, I think the deep mind perspective is that the way you get to general AI is you train neural nets that learn how to train neural nets.

Maybe you can tell me if I’m wrong. What’s your prediction for how this is going to take place? How the singularity is going to tip?

[0:51:44.2] AG: You know, my thing is a lot of people say rules are outdated, but I think pragmatically, I think pre-canned knowledge, in combination with statistical perception, will be the way to go.

[0:51:55.8] JM: So the format.

[0:51:56.5] AG: I think I’m actually an advocate of a hybrid approach, and this is actually the stance that the MIT folks take. You don’t typically think of MIT in deep learning, but they are doing a lot of great work over there. They just opened a new AI lab recently. They do deep learning over there, and they take this mixed approached where there is perception underneath some things, underneath static rules and facts, because one of the things that AI is going to run into is it needs semantics of some kind.

So encoding semantics is still a very difficult problem. The problem with semantics only is its brittle. The problem with statistical learning only is you can only define limited semantics. So you're going to need some sort of hybrid approach, I think, to make this work really well.

[0:52:40.7] JM: Fascinating. Okay. Well Adam, it's been a real pleasure talking to you. It's been very educational. I encourage people who are curious about deep learning to check out the Deeplearning4j website. Even if you have no interest in Java, there's just some really good simple explanations for some deep learning concepts. So I really applaud you for putting those up. Thanks for coming to Software Engineering Daily.

[0:53:03.2] AG: Alright, thank you.

[END OF INTERVIEW]

[0:53:09.4] JM: Listeners have asked how they can support Software Engineering Daily. Please write us a review on iTunes, or share your favorite episodes on Twitter and Facebook. Follow us on Twitter @software_daily, or on our Facebook group, called Software Engineering Daily.

Please join our Slack channel, and subscribe to our newsletter at softwareengineeringdaily.com, and you can always e-mail me, jeff@softwareengineeringdaily.com, if you're a listener and you want to give your feedback, or ideas for shows, or your criticism. I'd love to hear from you.

Of course, if you're interested in sponsoring Software Engineering Daily, please reach out. You can e-mail me at jeff@softwareengineeringdaily.com. Thanks again for listening to this show.

[END]