Lesson 3: Talking PyTorch with Soumith Chintala

(Lesson 3 has no picture)

63. So, in this nano degree, we covered a few error functions, but there are a bunch of other error functions around the world that made the shortlist, but we didn't have time to study them. So, here they are. These are the ones you met: there is Mount Everest and Mount Kilimanjerror. The ones you didn't meet: there is Mt. Reinerror, he's a big Seahawks fan. Straight from Italy, we got Mount Ves-oops-vius, and my favorite, from Iceland, we got the Eyjafvillajokull. This is the famous volcano that stopped all the European flights back in 2012. See what I did there is I took the original name which is Eyjafjallajokull and then I changed the word Jalla to Villa, which is Icelandic for error.

64. So, today I'm joined by Soumith Chintala who works at Facebook AI Research, and is the creator of PyTorch. I want to start this interview by asking you how you got your start in AI research. What got you interested in the field? Sure. I always wanted to be a visual effects artist, at least when I started my undergrad, and then I interned at a place and they said, "You're not good enough." No. I was like in the art angle. So, I was good at programming since I was a kid, so I tried to find the next most magical thing and that was computer vision. To me it was really compelling that you could just magically stitch things together and your programs could understand what a code is, what visual concepts are. That's how I got started. I basically tried to do stuff online, but Udacity wasn't around back then, so I had to find a professor in India, which is really hard to who's doing this kind of stuff, and there's just like one or two. I spent six months with the professor's lab. I started picking up some things, then went to CMU, tried my hand at robotics. Then finally I landed at NYU in Yann LeCun's lab doing deep learning, where I did ask him what are neural networks because I frankly didn't know. Yeah, you need a place to start. Yeah. So, that's how I got started. What motivates me is this; how do we make computers understand the world the way we see it? That just seems pretty magical to me and I want to get there. I completely agree. So, it sounds like your journey here was an exploratory one, would have been by curiosity and sort of having this idea of graphics in mind. I wonder how you ended up feeling motivated to create something like PyTorch and this kind of new usable defining framework. Sure. Since I got to NYU, I've been working on building up tooling for deep learning. I worked on this project called EBLearn which was like two generations before in terms of deep learning framework timelines. I started there, as that was my first tool to do deep learning research. It was very hard. Every time we kicked off a program, it would take 15 minutes, and then came around Torch, which is written by a few people who I knew. Then I started getting pretty active and helping people out using Torch and then delving on Torch. That was mostly, again, I wanted to do my research and then help other people do their research well. At some point, we decided that we needed a new tool because as the field moves the tools that you need to make research progress change as well. So, you can take a tool from the 90s and say, "Hey, I'll do my research on that", because fundamental flexibilities, the ideas you need to express, they keep changing. So, in end of 2016, we decided that was the time, again, and so I went about building PyTorch mostly because we had a really stressful project that was really large and hard to build in Torch. What was it that you are working on, if you don't mind telling me? It's around object detection. We were trying to enter the cocoa detection challenge. Yes. Our network was really complicated, it had multiple subsystems, and it

was not always differentiable. It was this really humongous thing and Torch wasn't the right tool. Then we briefly gave Transic a little try, but then it didn't go with our sensibilities. So, after that project was done, we had some self-recollection, and me and a couple of other people decided to just build PyTorch. Yeah, that makes sense. I think I also find it very true where deep learning is this dynamic field and as more tools are coming out, you see researchers really push their limits and then ask for more and more. Yeah. I think that's a good way to approach designing a product with these users in mind. Yeah. So, I wanted to ask you too like removed a bit from your individual experience, who were the first users of PyTorch and how did that community inform how this framework evolved? Yeah. So PyTorch had an interesting development experience. We started with just the three of us, and then we got other people interested within Facebook. So, it was me, Adam Pashka, who was an intern with me, and Sam Gross, who's another engineer. Were you still all working with Yann LeCun at this point? Yeah. So, we were working at Fair and Yann LeCun Rand Fair. It was a fairly large organization by then, like 70, 80 people. Fairly large fair. Right. So, what happened was the three of us did a core iteration of the product, saying this is how it should look. We based off of our designs of there was a package called Torch auto grad and like the old lower Torch. Then there's a package called Chainer that's still around from engineers in Japan. We thought those packages made sense, but we also wanted the back-end of Torch, which is really powerful. So, combined all of that, made something, and then we showed it to some people internally. There's a bunch of friends internally who were excited and wanted to contribute to the project. Then about eight or nine people joined in part time just adding this feature, doing things like that. Then slowly and steadily we started giving access to other people who I knew in the community, the non-existent community, but I knew they were looking for a new tool and they would like it. So, I gave access to James Bradbury from Salesforce, a bunch of people from the lower Torch community, Andrej Karpathy and a few people here and there. It would be like a review week, we would give access to about 10 people. Wow. Then they would give us feedback saying, "This is broken code." That's good feedback too. Yeah, so we created on that for like about six months and then in Jan we released PyTorch to the public. That's pretty much how it all started. The community grew. Again, power users were our initial crowd, and it grew from that. It's like more completely through word of mouth. It grew where each person tells their friend and they switch and then the whole research labs started Torching. It is definitely not any of the production crowd or the enterprise people, it was more of the fun long tail of people. Yeah, I like that. That's how it all started.

65. Is there a story you have for an error or some feature that, it was like a bug that was found early on? Yeah. This was before public release. I think we were in- so, public release was 0.1.12 and I think this was like 0.1.6 or something - Something really early - and Justin Johnson, who was interning at a fair, who's a student at Stanford, also runs- Yes, I think I've seen some of his computer vision videos. Yeah. So, he co-runs the CS231n course on ConvNets at Stanford. So, he was doing his research and then he was like, "My networks aren't training, and I started investigating. It turns out if you have a non-contiguous tensor and send it through a linear layer, it'll just give you garbage really." Just garbage. And we were like, "Oh man, that just sounds bad." So, we had pretty robust unit testing, but we forgot to add non-contiguous checks to all of our unit tests. So, that was a particularly fun one. Yeah, that's a big bug. That's good though that someone notices it before going fully public. One of the reasons why I like PyTorch is, especially for our teaching, it just merges so nicely with Python, which a lot of our students already know and it's something that's like fairly intuitive to pick up as a programming language. But I think about Python and I also think there's

a trade off there where the readability comes at a cost of it being a little bit slow. So, maybe you could talk a bit about that decision. Sure. Initially, when we wrote PyTorch, the autograd and- the entire internal autograd engine is all written in Python. Then, one of our aims for PyTorch was that it should be very imperative, very usable, very Pythonic, but at the same time as fast as any other framework out there. So, we put special focus on that. One of the consequences of that was large parts of PyTorch live in C++, except whatever is user facing is still in Python, so that you can attach your debugger, you can print. All of those are still very hackable. But, yeah, it gives you the best of both worlds where the main parts that the userland would need to know are in Python, but of the critical parts that are internals, they're all in C++. Okay. That makes sense.

66. So, recently, PyTorch has become pretty popular in the research community and I'm wondering if you think you know like why that is. True. I think it's again going back to how we built and traded on PyTorch. We gave it to a bunch of researchers and we took a rapid feedback from them and improve the product before it became mature. So, the core design of PyTorch is very, very researcher friendly by doing this reinforcement loop with researchers as we developed it. I think that is the main reason. It's very easy to do debugging. It's very Pythonic. If you know any of the popular Python machine learning or data science packages, it feels very natural that, the RBI looks very like numpy. So, I think that's the main thing that's been pulling researchers to PyTorch. Yeah. So, it sounds like PyTorch is designed with users and just their feedback in mind. I think some other libraries that makes me think of like sometimes they're designed with scalability and production needs first rather than users and ease of use. So, I'm wondering if you could talk a bit about how PyTorch, especially in its latest version, does also add features that make it easier to deploy models to production. Sure. So, I think the fundamental concept that if you want to build a usable library, it is intention that production is I think not correct. What happens is people from day one, if they put a focus on production, they don't need to make the library usable as well as, which means that it becomes like fairly inflexible or like unusable. So, with PyTorch 1.0, something that we did was we said you researchers, you want to do imperative code, all of that is great. It all works out very well. But if you want, unlike the code, PyTorch 0.4, it scaled up to hundreds of GPUs of pilot training. What we mean by production is you want to export your whole model and do like a C++ runtime or you want to run things like you'd take your network and quantize it and start running into the 32-bit, you want to run it at eight bits or four bytes. That kind of stuff, what we built PyTorch or when we could schedule for production is you do your research but when you want it to be production ready, you just add function annotations to your model which are like these one-liners that are top of a function. Then, PyTorch will parse your model, your Python program itself, and then make it into our own internal format that can be shipped to production. So, its kind of converting all of your Python code into, is it C++ code or just sort of an intermediate representation? It's an intermediate representation that can be run in C++. We provide an interpreter, the C++ only. But anyone can take that IR, which is called intermediate representation and then they can run it in their own virtual machine that they've built for themselves. That's great. I mean, it kind of makes me think of, if someone writes a Word document or something and pages and you just export it to PDF and then if you have is like you can send it anywhere and anyone can read it. Exactly. So, when you are moving in between say like a model you've written in PyTorch and Python and an intermediate representation. See there is a bug in there, what will happen if you try to debug with intermediate representation or is that something you don't ever want to do? So, you don't ever have to do that. So, what happens is you're building your model, you add your function annotation as an, it's just optional, it's there, but you can disable

it whenever you want. Either you can comment out the function annotation or you can have a global variable that says like switch off all compilation. So, it's like 90 percent of the time when you're doing experimentation and research, you're adjusting your irregular patronage mode. But then when you decide "This model is super solid, I've like very trustworthy of this thing. It's been around for a while. I want to ship it to production." You add function annotations and it becomes this magical black box that you don't usually need to touch. But if you, again, want to experiment with it, you just remove the annotations. Okay. So, experimentation, prototyping and the Python and PyTorch that we know and love. Then, once everything's look good to go, you can export it using annotations.

67. That's really interesting that you can do it piece by piece as well. At Udacity, we use a lot of Jupyter Notebooks to sort of break up tasks as far as like defining a model and training it, and I think it's really valuable to look at things like one pea at a time as these building blocks. Yeah. Absolutely. The other part that I think is worth telling is, if you have a giant model, you might not want to change big parts of the model. So, lets say there is a ResNet block. You might want to change how the ResNet looks and everything but the blocks in the ResNet, you almost never generally change. Like, they're just processing input data in the same kind of way. Yeah. Exactly, there like an LSTM cell. LSTM cell is a standard definition cell. You might change how the LSTM is put together and how many layers there are and stuff, but the cell as the core of it you never usually change. So, you can just add functional annotations to these smaller components while training the other components a lot. Which is why we call the new programming model hybrid front end, because you can make parts of a model compiled, parts of model still experimenting, and then that just gives you the best of both worlds. Yeah. So, this hybrid front end is allowing you to just sort of switch in between Python and basically like a C++ representation? Yeah. That's really interesting. So, if I was building a model and I wanted to- is there a way that I could sort of use optimization to train it faster or is it strictly these annotations are used strictly for after a model is trained? So, all of this is powered by what we call the JIT compiler in PyTorch. So, the [inaudible] digit compiler, in the short-term is to make sure we can export everything to production ready. Because we asked our users what's the thing that's most missing in PyTorch that's holding you back. They said, we use it at all these companies, like I've worked at my startup or I work at my big tech company and constantly other people around me are like, you can use PyTorch but remember you can't ship it to production. So, we wanted to first focus and remove that big constraint, make our users happy, open up that market. But the long-term investment of the JIT compiler remained to, is the parts of the model they're compiled, we're going to make them non-trivially faster by fusing operations, making more of the memory bandwidth bound operations into compute bond operations, and as newer hardwares come through, we can apply special tricks that are particular to each hardware that can do various things when they see a larger graph. Like make it more memory efficient and things like that. Yeah. But that's the longer term plan, to not just leveraged this JIT compiler we're using for exporting a train model, but also while you're training to make sure things get faster. So, before PyTorch 1.0, what were some of the process for moving a model to deployment? So, before PyTorch 1.0, about six to nine months ago we introduced an open standard called ONNX, O-N-N-X, and it's a standard that is not even like just focusing on PyTorch, it was a standard we tried to develop where all deep learning frameworks can talk to each

other. That's another formatting kind of stuff. Yeah. That is correct. We partnered with Microsoft, with various big players to make sure all the deep learning frameworks like Chainer, Caffe 2, PyTorch and TensorFlow, someone can take a model that's trained in one and then export it to another framework. So, with PyTorch, the model then before 1.0, was to only export it to ONNX and then run it as another framework like TensorFlow. I see. The shortcomings of that which we felt were that ONNX is a standard, which means that all PyTorch models couldn't be exported because the standard hasn't developed fast enough to cover that. So, which means it was turning at models that are more complicated, wouldn't be ONNX exportable. Which is why we talk like we want a robust solution, and that's the 1.0 story. That's great. So, it's so instead of these separate steps of exporting and then importing into a different framework, you have kind of like squished all these steps into one thing that you can do an a hybrid front end. Exactly. That does sound ideal.

68. When you talk about the pace of research and thinking about deploying models, are there projects that you find really exciting, that are happening in PyTorch uniquely or not uniquely? There are a lot of products I'm interested in, just because they're very crazy ideas. Those are my favorite as well. Yeah. So, there was this one paper written by one person, Andrew Brock and it's called SMASH, where one neural network would generate the weights that would be powered by for another neural network. Okay. So, it was like this neural network architects of search except that it's you're exploring all possible neural network architectures at once because the first neural network is generating the architecture and the weights for your neural network. That is the final neural network. I see. It is very funky. It took me a long time to even figure out like how he managed to do it in code. It was always in PyTorch and it looked very like, oh my god. It's like you give someone a calculator and they make Mario Kart out of it. Yeah or like a calculator programs another calculator or something. Yeah. So, there are several other examples. There's the [inaudible] from Facebook that have been closely following. It was more of a text-to-text processing model. So, you give some text and then it would generate another text. Initially, it was used for machine translation and all the standard language modeling, that kind of stuff. But more recently, some of the researchers Angela Fan and her collaborators, they published what you call hierarchical story generation. So, you would seed a story that like, "Hey, I want a story of a boy swimming in a pond." Then it would actually like generate a story. That's interesting. Of the dead flood. Yeah. A character like a vague situation. I find that very fun and interesting. That is that is also, I think text analysis can be really interesting when it comes to like, how do you actually like formalize ideas of understanding the context of something or where it might be going in the future. I think that's pretty good. I think too, when I look at research and some of my favorite papers are paired with like openly available GitHub repositories of work. We actually have someone [inaudible] who was doing the cycle again of formulation and his code is publicly available on GitHub and implemented in PyTorch and it's also just like very readable where you look at something you can clearly see like, here are the inputs, here is what's happening in as far as it being transformed, and here are the desired outputs. I think that's something that I really like to see and it's something that I see more with PyTorch than other languages.

69. So, you were talking about PyTorch being informed by what users are wanting, and especially with being able to put models to production. I was wondering if there are any other features that you hear people clamoring for what kind of demands do you hear now, and maybe where do you think PyTorch might move in the future? Yes. So, one thing people do ask for on the research side is, when they're exploring new ideas, they don't want to be seeing like a Tenex drop in performance.

So as an example, in PyTorch to provide an [inaudible] LSTM, which gives you a standard LSTM, you can configure the number of layers and what it does. It goes really fast because it uses GPU library called cuDNN, which was written by NVIDIA, and so it's about 10 times faster than if you just handwrite an LSTM with four loops. The problem that the researchers have been telling us about is that, if they want to try some new paper idea, this interface, the LSTM interface is very limiting, because let's say you wanted to do something like recurrent dropout, and [inaudible] LSTM doesn't support that, it only supports one formulation in particular. So, if they write it as a four loops and within LSTM cell, they see a 10 times slow down. I see. So, their request recently has been, "Hey, can we do our recurrent nets may be creative, at the same time can get something close to cuDNN performance?" That's an interesting request. Again, that comes in through the GIT investment that we did. We're trying to see if we can get users the speeds of cuDNN by stitching high-performers GPU kernels on the fly in the backend, based on what the users model is, that it looks promising. Those are requests that researchers have been asking for. One of the things that we heard from startups, and also people doing online courses, they want more interactive tutorials like based on iPython Notebooks. Also probably in embedding some digits. They want first-class integration with Colab, because you get free GPU [inaudible] . That's right. They've been asking for support for Google TPUs. These are all the request for getting and for handling all of them. Yes. It looks like the main cloud providers that I can think of are happy to support the latest versions. Yes. PyTorch for that reason. Yes. Amazon Azure, they made PyTorch a first-class citizen. Google announced support not only for PyTorch being a first-class citizen, GCP, but also they announced TPU support that we've been working with closely with them. Yes. Tensor Board is going to have PyTorch integration directly as well. That's great.

70. So, when I think about Pitrad, especially it'd been supportive by all these different cloud providers, I think of it as being a separate entity from Facebook. Which I think it definitely has its own life and community, but I'm wondering where it fits in the larger Facebook product or if it fits at all, where might it be used in that product? So, Facebook has needs and the AI space. They have Facebook AI Research and as fundamental research. We do research in the open, we use open datasets, we build our research, we publish it onto archive, to peer review. We also have a huge set of needs for products at Facebook. Whether it's our camera enhancements, or whether it is our machine translation, or whether it's our accessibility interfaces, or integrity filtering. So, we need tools that can do all of these. I think Facebook's investment is mostly on the perspective that if you need tools, we are going to invest in tools anyways, might as well make it an open-source investment. Because Ferris mission is to do AI research in the open and advance humanity using AI. So, this seems very, very fitting with that mission as well.

71. Then I kind of have two sort of last questions for you, like big picture. Like if you think about maybe just a year from now, so I'm not like thinking too far for this tooling, where would you want to see like PyTorch being used or where would you want that community to sort of be? Yeah. So, I've been thinking about this pretty hard. I mean we have all these awesome community. If you see people who use pi torch, they publish in ICLR, in NIPS, in ICML, CVPR all of these like top tier conferences. We got the deep learning crowd especially in research and startups maybe not the United Airlines. Right. We got these people. Next thing I was thinking was, deep learning itself is becoming a very pervasive and essential competent and many other fields. If you look at health care plus data that sub-field. Or you look at computational chemistry, you take what CERN does, you know, particle physics. All of these areas like you take like a Neurobiology, Neuroscience,

they're all starting. If you carefully look, they're all starting to use deep learning. The way they use it, is it's a very, very rudimentary- for them it's like oh look there's this GitHub repo that shows us how to put in our like our brain scans, and then get out segmented version. It's like some unit implementation somewhere. I think more- empowering them more by lowering- their main problem is they didn't know deep learning. So, my thinking is, maybe we should just go into these fields and build them packages that are- that lower their barrier of entry to use deep learning. So, there's like 10 research labs they use neuroscience who do neuroscience who want to use deep learning capabilities, just go understand what they actually need, and then build a cute package which they can relate to. When we say a PyTorch is pythonic, Python people who use use Python can relate to it, right? Yeah. I want PyTorch to be neurosciencic and particle physicic. So, I'm just making sure that those communities are empowered and not just like poking holes randomly. Yeah. Yeah, I like the idea of linear, going to ask people what they need as far as like kind of a data analysis tool. Yeah.

72. And then my last question for you is if you have any advice for people who are just sort of getting into the field of AI and deep learning, and curious to learn more. If you have any recommendations for them. Sure. I would just say, be hands-on from day one. I think I spent a long time trying to collect all the material that is possibly available, I think get a 100 textbooks because it seemed like if I just got the textbooks, I already got the knowledge in some fractional form. And then like also just do a lot of passive stuff, like listening to an android lecture or reading some blog post, and then be like, "Oh, oh, yeah. Those types, those types." Then there's always the barrier which is, when you actually sat and try to do something, you'd just black out. "I know all of this stuff. I read all of it, why I can't apply it?" I think that details as students are trying to get into the field of deep learning, either to apply it to their own stuff or just to learn the concepts, it's very important to make sure you do it from day one. Yeah. Stay on track. [inaudible] you encourage people to do projects, and review the projects, and give people feedback. I think that's a very very good interaction, that's my only advice to people. It's to make sure you do lesser, but do it hands-on, rather than do a lot more, but then just glance over it. Yeah. That's a sort of in-depth exploratory approach, almost doing like a full circle back to just really being consuming the information you need, but then really being curious and exploratory on your own design. Well, cool. Well, thank you so much for joining us. And I'm excited to try out PyTorch [inaudible]. Absolutely. Thank you.