**Interview 4**

**Interviewer 1:** [00:00:00] So I will start by explaining what is the objective of this interview. So we are conducting a study to create a taxonomy of problems that developers encounter when they develop systems that use deep learning and so this study has a different parts and one of them is that we analyze GitHub commits, issues and Stackoverflow discussions that are related to deep learning Frameworks such as Keras, Tensorflow Pytorch. But we think that this sources of information are not representative of all issues that developers might face and therefore we are conducting this interview with developers to get a wider picture on real world problems in deep Learning Systems. So, first, could you just confirm that you allowed us to take this interview and share your transcribed interview?

**Interviewee:** [00:00:51] Yeah, sure.

**Interviewer 1:** [00:00:53] So we would like to start by asking you some background information. So could you tell us what is your current position?

**Interviewee:** [00:01:03] I'm a PhD student at the parallel programming laboratory here at [University: Removed for Anonymity]. Well, my research is on the intersection of machine learning and parallel programming. So, so far I've been involved in both applying neural networks in my research and tuning them. So tuning their hyperparameters with regards to optimizing, for example not just the accuracy but also the Speed, Energy, Efficiency, and size but on the other hand, I'm also using neural networks for the task of compiler optimization and..Yeah, that's it.

**Interviewer 1:** [00:01:54] Thank you. So could you tell us what is your overall work experience? And what is your experience specifically in deep learning?

**Interviewee:** [00:02:04] Well, my work experience is mostly developing Java applications, writing SQL codes, but then I, then I also developed using Python and for machine learning specifically I'm mostly using Pytorch framework.

**Interviewer 1:** [00:02:33] So, in terms of years how how many years of experience do you have overall and in Deep Learning, some approximate.

**Interviewee:** [00:02:46] Overall? Let's say well, I don't count some of the it as experience even but let's say if I compress it down, it will be five solid years of programming and then with deep learning it s probably no more than two years.

**Interviewer 1:** [00:03:04] Okay, you told us that you're using Python and Pytorch while working with the deep learning problems, right?

**Interviewee:** [00:03:14] Yes.

**Interviewer 1:** [00:03:15] Okay. So thank you, all so as I told you our general question is what are the problems and bugs that you face? So maybe we can start with this very general question and you could tell us the ones that you face a lot, which you think are problematic and then we can go from there.

Ah, sorry, one one question that I forgot to ask what type of deep learning networks have you developed and implemented like supervised, unsupervised, reinforcement learning?

**Interviewee:** [00:03:51] I've done supervised learning and reinforcement learning. So with regards to neural networks, I've experimented with something as simple as MLPs and maybe, for example, I've also used ResNets, convolutional neural networks, DenseNets. What else, I've used a recurrent neural networks such as GRU, LSTM. Or, what else, the DNC - differentiable neural computer. Yeah, I guess that's it.

**Interviewer 1:** [00:04:27] Okay, and were you trying to tackle specific problems when using these deep learning networks such as image classification, speech recognition?

**Interviewee:** [00:04:36] So yeah, with convolutional neural networks with ResNet, DenseNet. Yeah, sure. I was I wasn doing image classification. With recurrent neural networks, I was actually processing sequences of data, but, lets say tokenising sequences of source code.

**Interviewer 1:** [00:04:57] Okay, now we can go back to our very general question of what problems bugs have you faced while doing all of these things?

**Interviewee:** [00:05:12] So, one of the most common bugs I think, and not just for deep learning, but in general for software engineering is called repetition, so control C control V or whatever. That produces some really nasty bugs and I've actually fallen victim for such behavior. Yeah, so that that may lead to some some wasted hours and days.

But, for example, when you do supervised learning you sample from different sets such as training, validation and test sets and then, If you develop the training algorithm and then you want to evaluate at the end of epoch, It's very easy to copy paste the training algorithm and then forget to modify the proper parts and then you just sampling from the training set instead of validation set and then your accuracy skyrockets and you're very happy, but let's in reality things are not as fun as they seem. So yeah, that's one of the mistakes but I guess that's with software engineering in general not just deep learning. With deep learning, I guess, in general deep learning is differentiation and when you do supervised learning you have X and Y's and then you have to somehow propagate signal from Y back to X and there's this chain in between and if the chain is broken, then the gradients do not back propagate. First thing let's say the chain is broken.

Another thing is when you initialize your neural network improperly, let's say you initialize it with zeros and then your gradients are just 0 so nothing falls back neural network doesn't train. Another thing is similar to this your gradients may become 0 at some point during training so it may stop training.

So I think one important way you can avoid this is by visualizing gradients. I think some of the tools, for example, TensorBoard allow this. I haven't used it, but with PyTorch, it's customary to use wisdom visualization framework and I've used that but it doesn't have such capability.

So in general if the gradients are bad, then your training doesn't progress and the gradients are bad in two cases if they are either zero or they're exploding and there is a way to overcome this as well, but I guess these are common problems when it comes to machine learning.

What else?

Yeah, I think that these are the common but I mean there are so many mistakes you can make, you can use..

**Interviewer 1:** [00:08:40] Yes, tell us about ones that you have made.

**Interviewer 2:** [00:08:43] Everything.

**Interviewee:** [00:08:44] You want to shame me on camera. What else, what I have made. Maybe I don't have enough experience to make enough mistakes. But he copy/paste mistake was the most ridiculous one.

**Interviewer 1:** [00:09:09] So what kind of error you were getting because like of this copy paste things.

**Interviewee:** [00:09:17] Well the error is that you think that your network performs well, but in fact, it doesn't, so the accuracy is 99%, but in reality it Isn't. This is a bit hard to catch because you want to believe that it does really well. That's the first thing and the second thing is that ,well with the exploding gradients scenario or gradients of 0 your network just doesn't train so, you know that something's wrong. There is a signal. When you do something like code repetition you have exactly as I said, the accuracy is really high and you're happy and that's not good. What else? Yeah, so another thing, another example is with the chain. I've broken the chain and what happens is that the gradients don't flow back and it you see that the network is training. So it's not that all the gradients are 0 but at some point they become zero and the gradients don't flow back and you don't know why it happens. But then the tools that let you visualize this hole. So at each point, it shows what are the gradients and if the gradients are bad. I guess this would be the perfect scenario to let the testing framework with you know test for (???). This is another thing that happened to me. That the gradients didn't flow back and that was because I messed up the logic so it's not just the software engineering part. You have to know the math under that underlies this training process. There's a leak of abstraction. You just you cannot just combine classes that makes sense from semantic point of view as a software engineer, but they don't really make sense from the mathematical point of view. So both the software part and the math part should work.

**Interviewer 1:** [00:11:37] Okay, So maybe I can ask a question. So did you use existing data sets to train your network or did you collect your own data?

**Interviewee:** [00:11:51] So I've used existing data sets before but when I do reinforcement learning, as part of reinforcement learning is a process of exploration and exploitation and in the process of exploration, you have to basically collect the samples. And I've done both.

**Interviewer 1:** [00:12:11] So when you were collecting your data for this reinforcement learning, how did you collect it? Where did you get it from?

**Interviewee:** [00:12:21] So in my case it was an environment, certain framework which provided me with the information - given the source code it provided me with an information if the source code is well formed or not.

**Interviewer 1:** [00:12:38] Okay. So was it like automatic collection of data, did you do you have to do any manual labelling?

**Interviewee:** [00:12:45] Well, wity with reinforcement learning? You don't have to do that, with supervised learning - yes.

**Interviewer 1:** [00:12:52] So in your experience you had to do manual labeling for the supervised ones?

**Interviewee:** [00:12:58] I didn't because it wasn't my dataset.

**Interviewer 1:** [00:13:02] Okay, so did you process the collected data or the data of the existing dataset in any way before training your data?

**Interviewee:** [00:13:13] Pre-processing? Yes. I did.

So what kind of pre-processing did you apply?

So there is this. When it comes to images and specifically to convolutional neural networks, first thing when it comes to any input, you want to normalize it so that the network learns faster. So normalization is the first part, the second pre-processing step is... So, for example, if you rotate an image by 90% ,let's say, then the neural network trained on an image, on a still image not rotated version, will fail to recognize whatever is displayed. So basically what happens is you usually flip the images during pre-processing step. And other thing is you zoom into different parts of the image so you can, for example for an image of a cat. It's still a cat's if you crop some parts of it. So that's one technique to make neural network more durable more robust towards the different kinds..

**Interviewer 1:** [00:14:25] So this is data augmentation, right?

**Interviewee:** [00:14:27] Yes.

That you applied. So do you have the cases when you did not pre-process the data in a good way you run your network. It wasn't performing well, and you went back and applied some kind of pre-processing.

I'm pretty sure I did that but it wasn't because it was not performing, it was performing but maybe a little under its potential. So yeah, I've done that.

Okay. Do you remember any kind of problems related to training data that you have faced? In general.

**Interviewer 2:** [00:15:06] And bugs, even little technical bugs are interesting for this study.

**Interviewee:** [00:15:14] The training data,you mean. I mean, for example, image \*inaudible\*, let's say, it has certain number of training data in the data set that are mislabeled. So. Yeah, for example, it's a cat, but it's labeled as a dog or something else and to deal with this you, basically, the perfect scenario is just to eliminate them. But if you cannot eliminate them, if their number is small enough that won't basically influence your training process as much. Yeah. So that's the only thing I can think of when it comes to problems with the datasets. It's just either it's mislabeled or for example, it can be that some classes of the data are misrepresented. So then you have to go back and either find more data or basically adjust your cost function so that the training samples from a certain class get a higher cost function - high weight.

**Interviewer 1:** [00:16:32] Okay. Thank you. So can you tell us more about the models you have been using, their structure in terms of number and types of layers and whether you ever had problems related to the wrong model structure?

**Interviewee:** [00:16:54] Number of layers. So I've trained various neural networks from the most shallow ones to the pretty deep ones. So the deepest I've trained is probably 540 layers deep.

**Interviewer 1:** [00:17:12] Okay.

**Interviewee:** [00:17:13] So that is the deepest ResNet that I've trained but It's not usual to go that deep. It's just for the for the reference. Usually, it's more like 20, 30, 40, in this range, because it's fast enough and it doesn't really make much sense to go much further than that because at some point it's just better to scale the network by increasing its width.

**Interviewer 1:** [00:17:51] So when training these models, do you have to decide on this model structure or do you get the model structure from somewhere? Were there scenarios where you had to alter the structure of the model if yes, then why?

**Interviewer 2:** [00:18:06] Like because of some problems you encountered.

**Interviewee:** [00:18:12] Because of the problems? Well, I have experimented with both state-of-the-art models, for example ResNet residual networks. This is published work and the same goes for DenseNet. When it comes to my neural networks. I've tried again to come up with neural networks and yeah, what I've done is I've modified dense Network, for example, so it's applicable to one dimensional data or a sequence of data - sequential data. Yeah, but I don't think that it's it's a problem. It's more like for how to say it, it's the application of the DenseNet to one dimensional data. When it comes to problems, sometimes, for example, once I forgot to do average pooling before passing the data to the final layer, and that's when the size of the network grows. You may forget that these sort of details, for example, we have. As I said with images, sequences, whatever you usually have.. When you do deep learning you usually do end-to-end deep learning. So you have X and Y and network has to figure out how to find how to get from X to Y and usually this network that's in between is structured in a way that there is a feature extractor and there's a classifier so it first comes feature extractor then comes classifier. This is the usual structure for many problems. And one thing to mess up is first of all, each of these you can... It's hard to mess up the classifier because it's usually just an MLP you can mess up when you pass the input from the feature extractor to classifier. For example, you don't shrink it, you don't average pool it, then the input grows, the computational demand grows. But this I wouldn't consider as a bug. That's more like omission of some sort. You forget it.

**Interviewer 1:** [00:20:41] Okay, so maybe we can talk about hyperparameters tuning, problems related to that. So, if I understood your work is about tuning so that it's like more fast than etcetera. So...

**Interviewee:** [00:21:07] This is a quite, I mean this problem has been tackled with from so many teams right now. It's a pretty hot problem. And there are so many different ways to tackle the problem. So..

**Interviewer 1:** [00:21:23] We are interested mostly is that like in your experience you had problems because of hyperparameters, you change them in some way you've got better results. So if you could tell us about specific things that happened to you.

**Interviewee:** [00:21:44] Okay. I mean any change to the neural network can be can be viewed as a change in hyper parameters. So usually usually to arrive at the prototype solution you start with something really simple and then you add more complexity, for example, in my tasks I start with with very simple structures and then if It doesn't work or works not as good enough then I progress. So I don't start with something very complex. So in terms of tuning... I think you're more interested in bugs with tuning and. The bugs with tuning are usually.. So It's buggy if you, for example, apply the wrong activation function. That's one maybe mug, for example, if your input is 0, Sorry?

**Interviewer 1:** [00:22:53] Did you encounter this bug related to activation function?

**Interviewee:** [00:22:59] I have not encountered it, but I may have encountered it. So I just, I'm just trying to come up with some stuff that maybe may happen but hasn't happened to me during hyper parameter tuning. I haven't really... So let me think maybe for a minute.

**Interviewer 1:** [00:23:23] Of course.

**Interviewee:** [00:23:59] Well, we have, with ??? activation functions. For example, you can..Again in this hasn't happened to me, but when you initialize Network or at some point you have zeros, then the neuron basically is dead. So it may be possible that the neuron will be dead. If you initialize the network improperly. I guess you can view this this way. So the hyperparameters set is pretty huge and one of the ways, one of the hyperparameters you may define is how do you initialize your network? There are also so many ways that you can initialize, you can randomly initialize it there is Xavier initialization, different ways. So if you initialize the network improperly then. the training will not go as you plan. So the training won't basically start or what happened, but this is yeah, maybe this Is one thing that happened and this happened to me is well. Yes. I've done it improper initialization. Maybe I don't view it as hyper parameter tuning but this can be viewed as tuning of hyperparameter because you have to choose a method used to initialize the network.

**Interviewer 1:** [00:25:19] Okay, regarding the loss functions. Do you usually use predefined loss function, do you use, do you have a custom written one, any problems related to loss functions?

**Interviewee:** [00:25:37] I've used mostly predefined loss functions, something like cross entropy loss or mean squared error loss when it's a regression task. I don't really develop this, you know, very custom tuned, because I haven't faced a task which required to tune this cost function. It's usually something built in. Maybe a combination of different cost functions.

**Interviewer 1:** [00:26:17] So did you have a scenario very use one pre-defined loss function and then because of some reasons you has to switch to another one or your first guess is usually as a job.

**Interviewee:** [00:26:30] No, I mean, I have, for example there's a binary classification function and sigmoid. Let's say a sigmoid function in the end when you need to predict either 0 or 1, right, you know, something is or isn't true. With this is hard to visualize the network.. When you do image classification, it's or whatever classification sometimes you want to see why the network performs the way it does and it can be hard with neural networks because they're black box methods, but there are techniques like Class Activation mapping, which allow you to mmm, so which allow you to visualize the sensitivity of the network, whichever parts of your input, does it pay attention to, when it decides something and sigmoid costs sigmoid won't allow you to basically visualize the network when it output 0 because then the features are turned off.

**Interviewer 1:** [00:27:41] Okay

The way to overcome this is use classification when you have to.. So when you have two classes either one or the other and then there's always something that's active and then when there's activity, then you can back propagate C,Y and I guess that isn't a huge problem, just.

Okay, and what kind of Hardware do you use? Where do you train your models? Did you ever have problems related to the hardware?

**Interviewee:** [00:28:19] So I've used, if it's a really simple model you can use a CPU, but mostly I'm using GPUs. I've used single GPU I've used multiple GPUs on the same node. I've also tried to use multiple nodes with multiple GPUs, but that's that's not as common. Problems.. Well, one of the problems is.. When it comes to using hardware, usually you want it to be used efficiently. So you're caring more about efficiency. It's possible that, for example, adding another GPU doesn't really make sense and it actually damages the speed. So training speed. I don't think this is a bug but yeah, this is a mistake that you can do. That you can actually use more resources and have less, much less efficiency and that has happened. Another problem is you can overflow the memory. Yeah, that's what I guess that can be considered the bug. For example, when you forget to delete the resources or what do you call it, release the resources that you kept for, for example, for processing a certain batch. That definitely can happen and that is a bug. And what happens is that yeah, there are so many ways that you can forget to release the resource or keep the resource for too long and then you cannot train and then you think that okay, my network is too big and I have to reduce its size. But in fact you're doing something wrong with managing the resources. And other thing is you... So the extreme part of this problem is whatever you run you fail because of memory consumption at some point of your training. Just because the resources that you allocate accumulate and then you just overflow the GPU memory. These are the problems.

Okay, so I have one question. While you were describing things you were saying that this is not a bug, this is a bug. So I think that the bug in a machine Learning Systems is.. So the perception is different from normal software engineering. So what is a bug in Machine Learning System to you? What would you classify as a bug what you would classify just as a problem, maybe?

As a bug, I would classify some behavior that was unintended. Unintended behavior that is also harmful. So, you may not want.. So some of the bugs are easy to spot, like you cannot figure out or your network doesn't progress. Or maybe another sort of bugs., which is a bug but maybe I haven't classified it. For example, when your network shows a high validation accuracy and in fact, it's not really a good estimator of the accuracy of a network. And that's that is I think the most,the hardest type of bugs to figure out because all this gradient stuff is, you know, something's not working and you have a signal you don't have to wait and in production something breaks. It doesn't work in development, but when you have the high validation accuracy, which is not a good estimator, that's pretty bad and to check that, so.. Your data set must be balanced and correct and a good representation of the real world data set. That's one important thing. So It's pretty hard with neural networks to satisfy certain criteria or to prove that the network is as accurate as it is. There is only the.. The method that's used today is just splitting the data set, I'm talking about supervised learning, squeezing the data set and then having a good representation of your data in this set of real world data. Yeah, it's a pretty tricky bug.

**Interviewer 1:** [00:33:16] Okay. So at this point, did you remember any new bugs or problems that you could tell us and Interviewer 2 has a question. Go ahead. Okay. Do you have anything in mind that you remember during our conversation or not?

**Interviewee:** [00:33:39] Any what?

**Interviewer 1:** [00:33:40] Like any type of bug, problems, do you have anything to add? That's a question.

**Interviewee:** [00:33:47] And maybe I under emphasised this last problem, because another way of looking at at machine learning and deep neural networks in general is that, I was talking about this training process, right? And training process can be buggy, but the harder bugs actually is that you produce a neural network, which is some sort of logic. Right? And this logic is Black Box and the important thing is that you produce something that Is not buggy and it's hard to check for bugs this black box because it's a black box and the way you test the black boxes is you have this some data set on which you test it. I think there is a lot of research in this area, but it would be great to have some hints, some ways of, you know, that could help a programmer say that even though validation set is high, there is a warning sign. It would be cool to have a warning sign: oka something is off with your network. For example, it's not sensitive towards some input. It may happen. For example, if you pass X which is a tuple, let's say, it's the X1 X2 X3, right, and it should pay attention to x2 because x 2 is, for example, person's age and X1 is whatever and prediction is will the person dies soon or not? Let's say. If it Doesn't pay attention to X1, for example, then it it's not working. Right? It should pay attention to that and it's hard to test these sort of things - the sensitivity to correct input. I think this is really important, maybe more important that anything I said before.

**Interviewer 1:** [00:35:43] Okay. Okay. Thank you.

**Interviewer 2:** [00:35:44] Just one question, when you are using like multiple gpus or multiple gpus on multiple nodes, as you said. Did you face any problems while distributing the workload between GPUs, when using shared resources, messaging, etc.?

**Interviewee:** [00:36:03] The problems I've faced were mostly to.. Because a lot of frameworks are really, or at the time when I used it, were really new with this functionality was really not that tested. So it was hard to set this up. But it all boils down to efficiency rather than bugs, if it's a bug then there will be resource leakage and then at some point it will stop training. Yeah, and I addressed that before, but otherwise... It can be a bug actually if you do not, for example, distribute your resources correctly. It if you distribute them over the network one way, one better way would be to have have the data on every worker that performs a training for example. Yeah, there are so many things that can go wrong..

**Interviewer 2:** [00:37:10] Like race conditions or something.

**Interviewee:** [00:37:13] Race conditions?

**Interviewer 2:** [00:37:18] The shared resources, like when workers fights over the resources.

**Interviewee:** [00:37:24] A, you mean the concurrency bugs?

**Interviewer 2:** [00:37:27] Yes.

**Interviewee:** [00:37:28] That is certainly possible. If you really want to, you can be equipped with that, but I think it's hard to... If you're developing this system from scratch the distributed training then yes. Usually when you do it, most people do it by using some sort of framework which has been tested and harder to mess it up. But if you're doing it by yourself, then certainly you can mess it things up such as dead locks \*inaudible\* whatever you wants from parallel programming.

**Interviewer 2:** [00:38:08] Did you have anything like that?

**Interviewee:** [00:38:10] No, I haven't.

**Interviewer 1:** [00:38:14] Okay, so we don't have any more questions left for you if you have anything to add to your welcome.

**Interviewee:** [00:38:24] No, that's it.