Distributed Optimization and Machine Learning

Prof. Mayank Baranwal

Computer Science & Engineering, Electrical Engineering, Mathematics

Indian Institute of Technology Bombay

Week-11

Lecture - 42: Introduction to Federated Learning

Alright, so today is going to be an introduction to what we know as federated learning and it is a very hot topic in distributed machine learning and in general if you want to train a single neural network model based on data that is distributed across millions of workers or millions of servers. So a typical example would be, if you use let's say an Android phone, so essentially your Gboard or any Samsung keyboard or Android keyboard. So the idea is, as you type something, the next word, and it starts suggesting you the next word, right? So it really depends on how people use their keyboards, and basically the best prediction would be modeled accordingly. So how do you think such kind of models are trained? So let's say this is a snapshot of Gboard from Google. So how do you think such kind of models are trained? So it has to be, first of all, like let's say Google wants to roll out something, right? It has to be a single model that has to be rolled out across all devices, right? So the question is how do we, like when we have data distributed across so many workers, so here we are talking about training neural networks on edge devices like mobile phones or IoT devices, which may not have the luxury of having like enormous GPUs to work with. So far we had looked at decentralized SGD or parameter server approach where you had few clients or servers working on their private data set.

They were computing gradients on their own data and then Either they were, like if it was decentralized SGDs, so there was no centralized server. So essentially they were doing some kind of gossip algorithm to ensure that the parameters of their neural network or the model converge to the same value. But then, there I mean you still assume that there are a few maybe tens or hundreds of such workers. The moment you start having thousands or millions of such workers or clients, the challenge is how do you in real-time get information from so many different agents? and that becomes practically infeasible right.

So in that regard, federated learning was sort of devised to essentially work with training of neural networks on edge devices and that is what we are going to look at it today. So when we say edge devices, so essentially you have, what do we mean by devices? So cell phones or your IOT devices. So, essentially that basically have contain a lot of devices collect huge amounts that can be informative for ML model. So when we say huge amount of data, so every user, basically they have their own pictures or their own text to work with. So everyone is just accumulating a huge amount of data or also creating a huge amount of data.

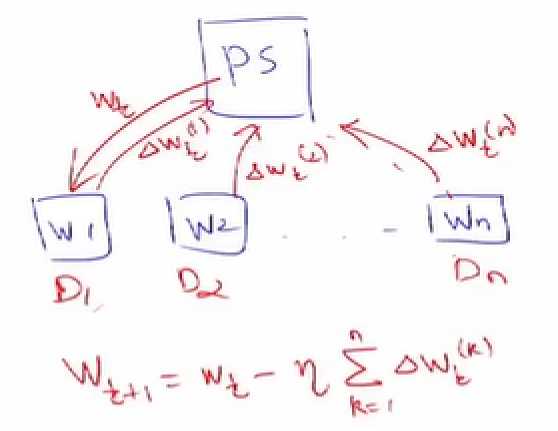
And whatever model that needs to be deployed, it essentially should try to derive common features across all different edge devices. And such a model would be very informative because that would be largely applicable to a broad audience. For instance, as I think towards the beginning of this course, I had mentioned that like IITB uses this lingo that I mean you guys add max to a lot of words, infimax, coolmax and things like that. So, if you have lot of IITB users, just start typing like this and eventually spread out to the rest of the world and you continue to use the same lingo, eventually you would see that that kind of thing also starts popping up when in the Gboard as well. So, even though everyone's data is highly heterogeneous, it is private and so on, most of the training is basically most of the feature that essentially I mean the two different individuals may share.

So, essentially this learning happens on common features between different agents or different workers. I mean there are these edge cases which I mean as we train the neural network right, there will always be outliers and we do not try and fit to those outliers. So essentially you would see that if once I mean if once you are communicating with lot of IoT users or lot of cell phone users, you are trying to essentially learn those common features that connect these different and that is how the Gboard or any other prediction models are based on. So that is an aside thing, but why do we have qwerty keyboards and not keyboards which have ABCD and so on. It's an aside thing.

What could be the reason why people decided to work with this kind of keyboard? If you remember, I think Micromax at one point had launched this phone with ABCD keypad instead of QWERTY keypad. I mean eventually Micromax didn't last. No, that's not really the reason. So before we had this electronic revolution, all keyboards used to be mechanical keyboards. So the typewriters used to be mechanical typewriters.

And the problem with mechanical typewriters is they were based on hydraulics. Essentially if you start typing too fast, in fact you can show that like if you start having ABCD instead of QWERTY, you can have a better typing speed. But if you start typing too fast, it will result in paper jams. So, they arranged it in such a manner, so that like it is generally not possible to type too fast and once people like once the society transition to these electronic keyboards, the teachers or the ones who were teaching typewriting, they were the ones who were trained on those quality keyboard. So, it kind of got I mean the same sort of trend kind of got translated to the digital revolution as well and even the digital keyboards are now quality keyboard, so it has no I mean This has been designed so that you are not too efficient.

So when we say about training data, so what would be a training data on edge device like cell phones? So what every user types on their phone. So again, to start with, we have millions of edge clients. Essentially, we are talking about cell phones and these tiny-edge devices. And the question is, how can one utilize the parameter server framework, something that we have already looked at, right? So just to recap, what was parameter server framework? So you had a centralized server, so parameter server, and there are these workers, right? Let's call them workers W1, W2, and so on. So, everyone has their own private data.



So, let us call these data sets d1, d2, dn, and so on right. And in the parameter server approach the idea was that the parameter server would broadcast and let us say t th iteration, it would broadcast the weights wt. So, these servers on their private data set they would end up computing the gradients on their own data set, and let us say they would they would communicate delta Wt 1 essentially the gradient information right this would communicate delta Wt 2, and so on delta Wt n and then these and then the parameter server would perform this update which is Wt plus 1 is Wt minus some step size times let us say k equal 1 through delta Wt. So this was the parameter server approach. So what are some problems with this particular strategy? Can you think of certain potential issues with this particular strategy? Synchronization of updates is one thing.

So the biggest challenge that you see, again you have to look at this particular point. So there are millions of edge clients. So, if you have millions of edge client that means you have enormously huge data and for the parameter server it has to wait for information from so many edge devices and it has to actually have huge communication bandwidth. So, when we talk about in terms of, so parameter server approach requires in fact prohibitively large large communication bandwidth right because you are working and that is because we have we are working with millions of edge devices. So, you have to have that much bandwidth to be able to communicate with that many edge devices right communication bandwidth since it exchanges information with millions of edge devices.

What is the other issue with this? So while I mean that may not directly be the case you can show that so another concern is data privacy. And even though you are not directly sharing your data in certain cases you can show that if even if you are sharing your gradient information to a parameter server or a centralized Just by looking at the gradient information and looking at your own state, there is a way to estimate the kind of data that a user may have. So, you can try and recreate. So, there are some data privacy and sensitivity concerns. So, sensitive information is another concern.

And what is the third issue? So these edge devices may have limited internet connectivity, right? So to be able to communicate with your centralized server, you would need a good internet connectivity at all times, right? Because it's waiting to receive data from all the devices or all the users and all the users need to be connected to internet to be able to communicate. Otherwise, this would just keep on waiting for data from all the users and only when it receives data from all the users, then it can think of performing the update, right? So edge devices may have limited internet connectivity. So, to account for this protected learning was sort of devised as a framework to be able to perform machine learning at scale or distributed machine learning at scale. And the key idea is instead of trying to train a neural network at a centralized server or centralized location where you would need either access to all the information to start with or you would require a complete like let us the other solution is to have a completely centralized approach right where all the data is like all users private data is stored at a centralized way and the centralized server basically looks at those data points and then performs like a and trains a single neural network model. So, this either this or you have a parameter server approach where you wait to communicate with all the users or all the agents or all the clients in each iteration right and that basically places a huge communication bandwidth requirement.

So, parameters this federated learning was a I mean basically was devised to essentially elevate these issues and the sort of key idea is you instead bring the training to the edge well not data but devices. Essentially you bring the training to edge devices and we will look at how federated learning works. But it is somewhere hybrid between centralized like a parameter server kind of approach, but there is some level of randomness also involved. And this was in the seminal paper by McMahon et al from Google in 2016. So this is the title of the paper where they came up with this fed-average algorithm.

So, this is already used for next word prediction on android cell phones and the key idea is you only train the neural network. So, you only when it is basically I mean on your device on your edge device only when it is basically plugged in for charging. So, essentially do not drain the battery that way that you are always like at the back end it is always training a neural network that it does not happen like that. So, a particular user or a particular edge device is only involved when it is plugged in for charging and it is only few MBs of data that is used on your own sort of cell phone and you are not using data from anywhere else. So, you are not communicating with any other edge device, you are just I mean you are essentially cell phone is a standalone sort of trainer of neural network.

And there's a nice sort of, so Google came up with this nice federated learning comic strip that I think you guys should give a read. So essentially it talks about what federated learning is and how it sort of came into picture. So it starts with someone coming in back from a conference enthusiastically, having discovered a way to maintain data privacy, but at the same time being able to perform distributed machine learning. So, it kind of starts at like the company the head basically the head of the organization basically allocates few interns and then the interns they start listing some issues one by one, and then this basically the main character then basically talks about how these issues are kind of addressed in this federated learning framework. So, you should give it a read first.

So, one of the issues is the personal data or the sensitivity of the information. And so, there are these different aspects and eventually talks about how federated learning works. There is also basically encryption to ensure that whenever data is basically being sent from user to a centralized sort of server essentially it is properly encrypted. So, no one can actually reconstruct the data and so on. So, all of these edge devices the data is encrypted and it basically goes here.

and it in such a manner basically the data is masked. So, it is so that it is basically a zero sum mask. So, essentially you perturb the data in or shuffle the data in such a manner. So, that on an average everything is basically kept intact. So, these masks basically exactly cancel out and so on and eventually basically talks about how federated learning kind of works.

So, should give it a read it is a nice comic strip. So, the idea is you have a central server and the central server broadcasts its current set of weights. Let us say you are training on your single neural network right and it broadcasts the current model parameters to the different edge devices. So, instead of doing this with all the edge devices, it would select a fraction of edge device at a time. So, it would not do it for all the edge devices in one go, it would select a fraction of the edge devices and it would relay the information about the current parameters.

So, let us say this information x t or the current set of weights are being shared. Now, on this let us say model 1 or the device 1 what happens is essentially device 1 receives those current this parameter values of the weights and they start training the neural network on their own private data for certain let us say few epochs. So, tau i number of or basically it performs tau i number of local updates. So essentially it trains a neural network based on the parameters that it has received or based on the current set of weights that it has received, it basically starts training its neural network on its own private data for certain number of epochs. So essentially you perform gradient descent multiple times locally and let's say it ends up getting a weight, let's call it Xt1.

Similarly, the other device would do the same thing Xt2, Xt3 and so on. And then this particular, let us say there are k devices involved, this particular information is then propagated back to the centralized server. At the outside it almost looks like the decentralized approach or SDD or the parameter server approach. But the key idea is we do not work with all the clients or all the edge devices. We only select a handful of those and we do not perform a single round of stochastic gradient descent and then share the gradients.

Basically in the parameter server approach what was happening is give the current set of weights i'll perform a single single round of update on my data set and i'll basically give my gradient estimate like essentially average gradient value send it back to the parameter server or the centralized server in this case you perform multiple rounds of updates okay on your own data you perform multiple rounds of update and then you have sent the updated parameters to the centralized server and this is done only for a handful of devices not for all devices because not all devices are going to be for instance plugged in for charging at all times right so only a handful of devices are selected randomly and this update is performed and eventually because devices are being selected at random so it would like everyone's like let's say at some point you would be basically at some point you I mean your own phone will be picked up right because So, every edge device has an equally likely probability of being selected right. So, because it is being selected at random. So, this is how, so it is not like your data will never be used for training. So, that is not going to be the case. But then as I said because you are averaging over a lot of devices, so only the common patterns or the common features are going to get sort of captured and it is not like your let us say you prefer to use a particular lingo that not every everyone else around you uses.

So that you would not expect this to be picked up by neural network because it will be treated as an outlier. So this is the idea. So if we compare decentralized SGD or even let us say parameter server approach. with federated learning. So what is the difference in terms of number of workers or servers or clients that both the algorithms need or both the approaches need.

So with parameter server at a time as I said we cannot work with millions of devices right. So usually when we talk about parameter server or decentralized So when we talk about tens or at best hundreds of clients, whereas when we talk about federated learning, we really talk about a huge scale. So essentially millions of clients. So that's one difference. What about availability of workers? Do we in case of parameter server, do we require the workers to be available at all times? So, we assume that we are going to be receiving information from every worker involved.

So, again in parameter server, we require the workers to be available at all times. In federated learning I mean that is not needed right. So, these work with a handful of clients right, work with a handful of clients. What about data distribution? remember the decentralized sgd theorem that we had looked at so there was this b square term that was on the data like heterogeneity or the non iid data right so essentially more or less you want the data heterogeneity to be somewhat bounded so with parameter server or decentralized sgd we require that the data is roughly homogenous, I mean it may not be IID, but we require the data to be roughly homogenous for it to work efficiently right. So, essentially the data that is distributed across different devices or different clients we require them to be roughly homogenous.

So, if you are talking about let us say classification of handwritten digits from 0 through 9. So, they are in these 10 classes. So, you would require that all these 10 different classes are equal roughly I mean equally distributed across it may not be the same image, but you will have maybe equal roughly equal number of 0s and 1s and 2s and 3s and so on.

So, we require the data. roughly homogenous. With federated learning, you can work with really heterogeneous data right because since we are talking about edge devices you have very I mean you are going to have your own personal data or lot of personal data is going to be there right. So, the data distribution is going to be highly heterogeneous. What about types of workers? So, essentially when we talk about the compute requirements of different clients in parameter server or the decentralized SGD, do we require them to be homogeneous or heterogeneous? So, usually they I mean when we talk about When you think of parameter server approach always think of as let us say maybe a GPU enabled device at one particular location right. So, there are multiple such locations and they are interacting with each other. So, essentially in parameter server approach the type of the clients are largely homogenous in terms of computer compute power right.

with federated learning. When we say edge devices, you can have a very powerful cell phone or someone can work with their own tabs. So you have different types of edge devices, you have different types of IoT devices, right? So the devices or the clients are also going to be heterogeneous. What about privacy? So, essentially while at least with parameter decentralized SDD, there is no information sharing with the centralized server. You are still communicating with your neighbors and as I said you can potentially use gradients to estimate the data distribution that one particular agent has. So, I would say that data distribution can be rearranged.

Since that you can try and estimate what are the in federated learning data is private. So, and secure. So, you want you do not want to exchange your own private data with your neighbors or a centralized server and so on. And as we saw in that comic data is usually encrypted when it goes to the server. So within federated learning, there are these two different terms that are often used.

One is cross-device versus cross-silo kind of federated learning. So when we say cross-device, it is the usual use case of federated learning when we have like different edge devices like cell phones. So, it is basically you are using different devices like a cell phones or other edge devices to essentially maybe train a single model that would be cross-device federated learning. Cross-silo federated learning is when you use instead of devices when we talk about different entities. So, let us say there is a hospital or there is a bank or there is some other firm and so on and they have their own let us say a server room or something like that right and they where they use their own different types of data like a hospital would use a different completely different type of data right then a bank for instance but then again like you want to train a single model based on these different heterogeneous data distributions so these are not edge devices particularly may or may not be edge devices anymore So, you have more sort of a again as I said like it can be a server room. So, you can have a GPU enabled device for instance. So, more homogeneous sort of compute devices as compared to the cross-device federated learning. So, this kind of federated learning is called cross-silo federated learning. So, where you have these different organizations that are involved and not the individual edge devices.

So again if it if we look at the distinction between these cross-devices and cross-silo in terms of the number of devices obviously you are going to have a more like in cross-device you will have more number of devices as compared to cross-silos right. So this is again in the range of tens to hundreds whereas would be in the range of few millions right. What about availability of workers? Since we are talking about federated learning in both cases, unlike this particular case where we had for parameter server or decentralized LCT, we required the workers to be available at all times. Since we are performing federated learning, I mean they will have the same requirements that you do not expect all of them to be available at all times right. Data heterogeneity again it is going to be heterogeneous in both the scenarios whether it is cross-silo or cross-device.

So, data is going to be data distribution is heterogeneous. So, this is again a slight departure from the parameter server approach where we require the data to be somewhat homogenous right. Worker types, so in worker types cross-silo would have more homogenous workers as compared to cross-device which is going to have more heterogeneous workers. So, cross-silo will have homogeneous workers and cross-device would be heterogeneous. And privacy constraints, it is the same privacy constraint. So, using the data is private and secure. So, that because both are again part of edited learning. So, we still have the same privacy constraint. So, data is private and secure. So, again this is again going to be a slight departure from decentralized SGD.

So, we are going to use certain notations. So, the first thing is the total number of workers or total number of clients that are going. So, workers, clients, agents. So, all these terms would be used interchangeably. In certain cases I mean you can also think of it as devices and so on. So we assume that their total of I mean total such number of clients is k out of which we would like at a time in each communication round only c clients would be part like of basically it is a fraction of client that would be participating that fraction is c.

So essentially c times k is in total number of clients that are going to be participating. So as I said if in federated learning let us say the current set of weights is xt and on this using these weights a model is trained on an edge device for certain number of certain number of rounds right. So, you perform multiple local updates on the local data right. So, we are going to assume that the mini batch size for this model like updating the model weights that is going to be B, capital B and that is going to be the same across all devices. Is this clear? Then number of stored data samples at the ith client, so that is going to be n sub i.

So, this would have like let us say n1 number of points, n2 number of points, n k number of points and so on ok. Then the learning rate is we assume that it is going to be constant across all devices. So, this is going to be eta. So, learning rate because these devices themselves will be training neural networks. So, they would be using certain learning rate for these stochastic gradient descent algorithm and that is going to be eta and number of local epochs per client.

So, what do we mean by local epochs? So, local epochs would be. So, let us say you are using a batch size of capital B. So one local epoch means that essentially you have iterated over the entire data set using a batch size of v that is one local epoch. So how many total local epochs you would be performing per client that is going to be capital E. So that means you have iterated over the entire batch, entire data set capital E times. say E is the total number of local epochs, capital B is the batch size and N sub i is the total number of data points per client or let's say client i.

So, how many local updates will be performed at client i? So, what is tau? So, we call this tau i. What is this tau sub i equal to? So, again what is the definition of local epoch? One local epoch means that you have iterated over the entire data set once using a batch size of b. So, if n sub i is the number of data points. So, ni by b is the total number of updates per local epoch right and you have e number of total.

$$\tau_i = E \frac{n_i}{B}$$

So, this is E times ok. So, this is what tau i is equal to. Is everyone with me on this? So the total number of local updates, so a local update would mean like let's say I give you a batch and you use that batch to update the weights. So how many such local updates are being performed? So that is going to be equal to how many times you iterate over the entire data set of n i points using a batch size of capital B times the total number of local epochs which is capital E. So this is going to be tau sub i. Is this clear to everyone? Thank you very much.