Distributed Optimization and Machine Learning

Prof. Mayank Baranwal

Systems and Control Engineering

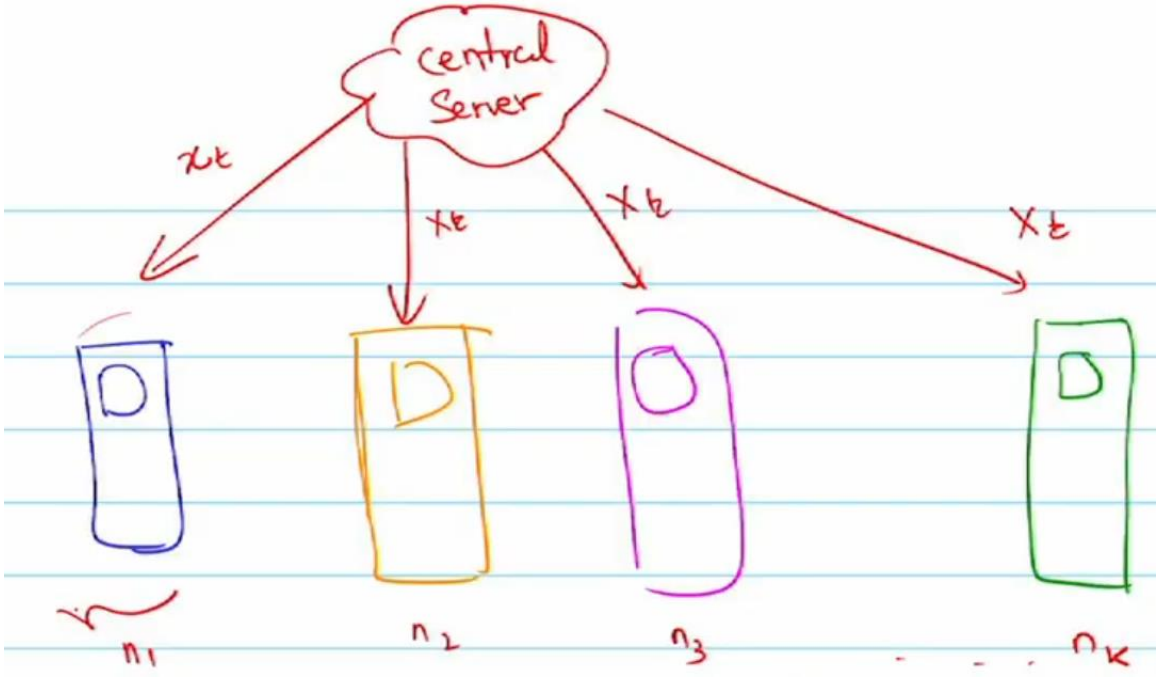
Indian Institute of Technology Bombay

Week-12

Lecture 45: Sources of Computational Heterogeneity in FL

We had started looking at the basics of edited learning. As I mentioned, this course is largely about distributed optimization. So we won't go too deep into distributed learning, but it is an emerging area. In fact, I have already gained a lot of attention.

So, we basically want to understand the fundamentals of edited learning. So, just to briefly recap what edited learning is. So, this is the kind of setup we have. And this is essentially about taking the computation to the edge device, right? So, you have a central server, and let's say you have plenty of edge devices, okay? And this is how it works: the central server will have several rounds of communication with the edge devices, which, let us say, we are using to train a neural network.

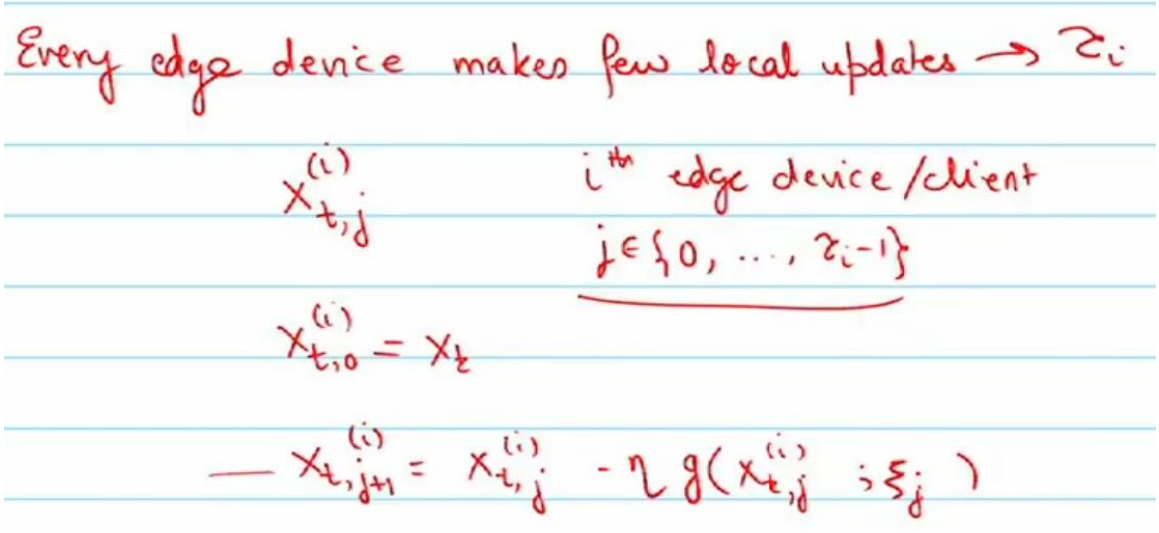


So, what this central server would do is, in some communication round t, it would basically broadcast its weight \(x_t \) to these edge devices. So, this x t is broadcasted.

Now, these edge devices are equipped with their own data samples, right? So, let us say this has n_1 data points, n_2 data points, n_3 data points, and n_k data points. And what do these edge devices do with these weights. So, what happens once the central server communicates the XTS? So, it basically runs a few local updates, right? So, every edge device makes a few local updates, and we denote these local updates as tau sub i for the ith edge device.

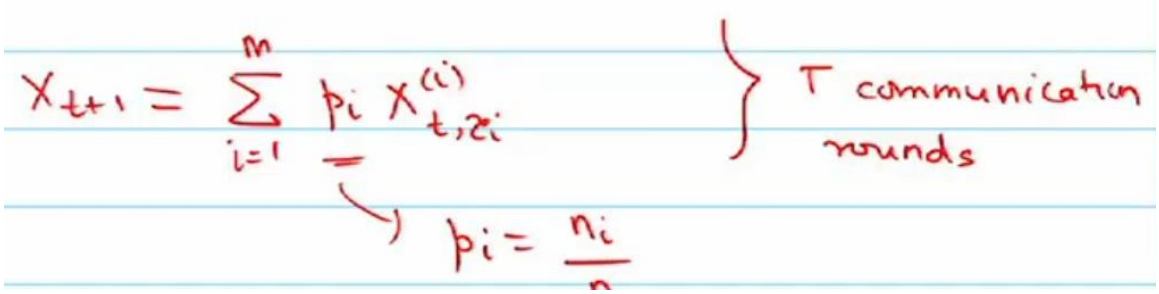
So, it is going to run tau sub i number of local updates, and the way this works is that we are essentially going to define something called x at t plus 1, or let us call it x x, or rather, let us call it x at t instead of x at t j, which essentially implies. So, you have the ith edge device or ith client, and j is an index from 0 to τi minus 1. Therefore, xtj0 (or x, sorry, xt0) is the same as xt that has been communicated by the central server. And then, in a mini-batch, we make an update, which will be \(x_{t,j} = x_{t,j-1} - 1 \). Some step size times, you take the gradient evaluated at \(x_{t,j-1} \) and the data points that have been included in that mini-batch, okay? So, this is how we update, and this update runs forever from 0 to tau i minus 1, right? I mean j goes from 0 to tau, 1 minus 1.

So, if I let us say, let's even write j plus 1 here. So, this essentially becomes j, and j is an index number in this set, right? So, this is how every agent or client is updating these x_t_j's, and then what happens is that these clients will communicate back to the central server x_t_i_tau_i_tau_sub_i, right? So, in this case, \(x(t_1, \tau_1) \) and \(x(t_2, \tau_2) \). So, all this information is going to be communicated back to the central server, and the central server then basically computes x at time t plus 1, or let us say x for now; let us assume we call it x at time t plus 1. So, weighted \(i \) equals 1 if there are \(m \) agents involved, or let's say \(m \) clients involved. So, p sub i x i t tau is okay, and then this information is communicated back to the clients.



This process continues with communication rounds. The sentence is already

grammatically correct. However, if you want to rephrase it for variety, you could say, "Is this clear to you?" So, how do we choose this \(p_i \)? So, p sub i is the relative weight applied to a particular agent, and how do we choose it? Usually, it depends on the number of data points. So, if there are \(n \) sub \(i \) data points. So, p_i is usually n_i divided by the total number of data points.

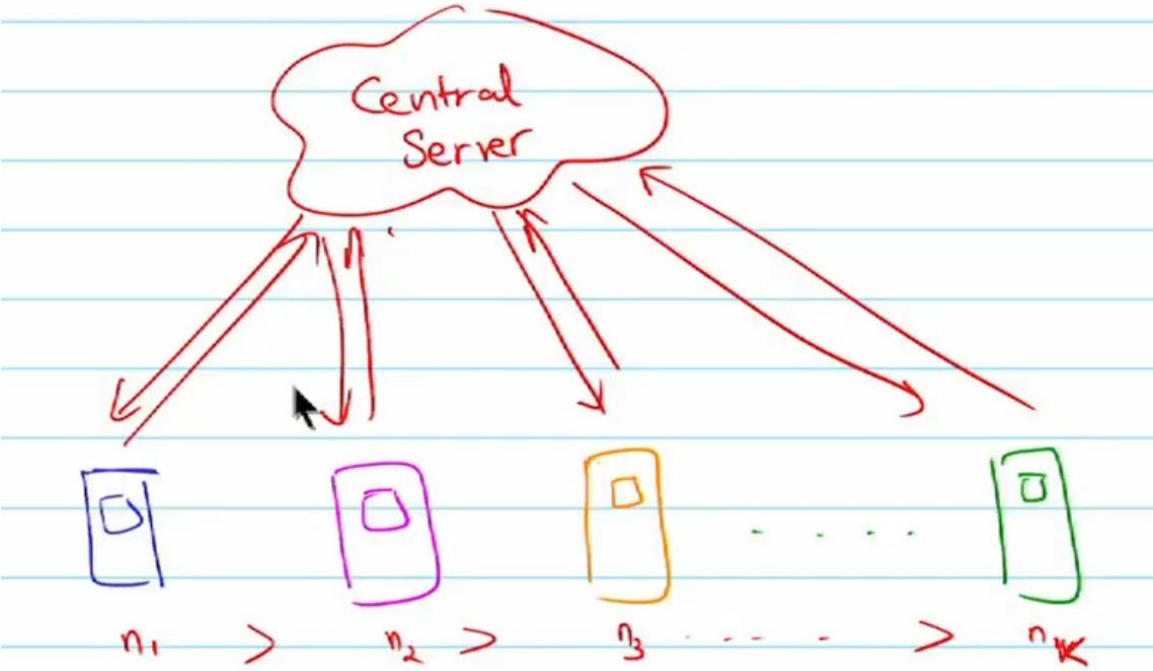


This is typically how we choose these weights, p sub i. So, in the last class, we looked at the effect of certain factors. The parameters that essentially govern the overall federated learning framework are, let's say, the number of local updates that you run, right? Should we run too many local updates or too few? So, what is the effect if we run, let us say, too few local updates in terms of error convergence? Is it going to take longer to converge? If we take very few local updates, is it going to take longer to converge, right? So ideally, we should run many local updates. But if you run a lot of local updates, what ends up happening? The average of the individual optimal solutions is correct, and that is also something that is not desired.

So, that is one thing. The other thing is regarding the batch size. If the batch size is large, do we have more updates or fewer updates? If the batch size is large, then we have fewer updates. So again, a larger batch size means that it would take multiple communication rounds—many more communication rounds—for the algorithm to converge. So, all these parameters play a role.

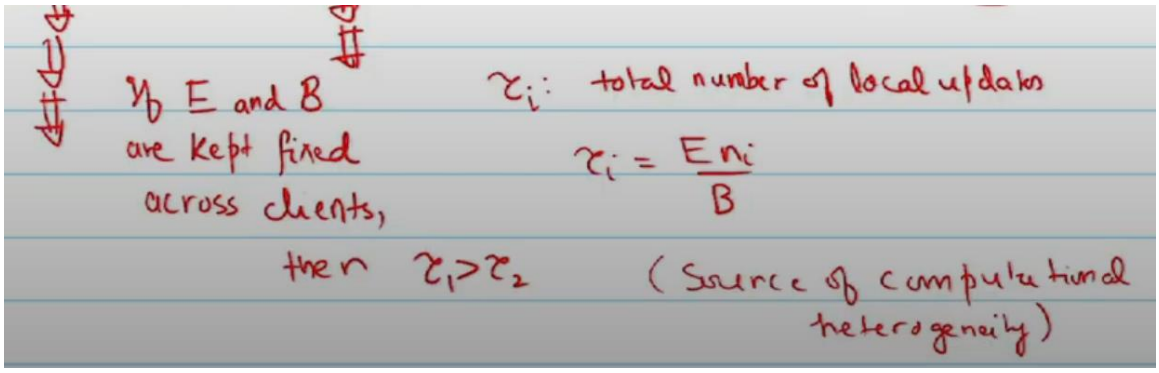
So, today we are going to examine the sources of computational heterogeneity in federated learning. The effect of "like" essentially updates a generalized update rule for what is referred to as the generalized update rule for federated learning. Then we would look at something called the FedNova algorithm. So, FedAverage is what we have examined up to this point. We would look at something called the FedNova algorithm.

We would also consider fairness because each agent has received its own data points. So, how do we account for fairness in federated learning? So, that is what the measures of fairness are. So, these will be the topics of discussion in today's lecture, okay? So, the sources of computational heterogeneity are. So, once again, consider the same setup: we have a central server and multiple clients. There is an exchange of information between the server and the clients in this manner.



So, what are the possible sources of computational heterogeneity that you can think of among these clients? So, let us say this particular client has n1, n2, n3, n4, and nk data points, right? Suppose this is a setup, and it is also assumed that n1 is greater than n2 and so on, up to nk. So, what do you think would happen if, let's say, we use a fixed number of tau? So, tau is the total number of local updates, which is kept fixed. Let's say tau_i is the total number of local updates, right? So, what is the formula for τ_1? E times n is over b, right? So, one source that you can consider as a potential source of computational heterogeneity is. So, if n1 is, let us say, greater than n2 for the same batch size and the same number of local epochs, E, you would have a larger $\tau 1$ compared to $\tau 2$, correct? So, that means if E and B are kept fixed across clients, then in this case, we have τ_1 greater than τ_2. So, we would have more local updates here then and fewer local updates there, right? That means the x to which it will converge will depend more on this particular client's set of points than on the other client.

Thus, the client with a larger number of data points will have significantly more influence on overall performance than a client with fewer data points. This is one source of computational heterogeneity. Which is also going to be large, correct? But then, just as I said, forget about the waiting part; the total number of local updates is also related to error convergence and other factors, isn't it? So, if I have more data points here, that means I am performing more local updates; essentially, you have significantly more local updates happening here than in this case, where you would have fewer local updates. And this would decrease as you move to the right. So, fewer local updates are occurring.



So, among the initial few sets of clients, they would have a larger say than the others, I mean, over the entire learning process, right? Is this clear? So, that is one source of computational heterogeneity. How can we potentially avoid this particular source, or, essentially, how can we eliminate the effect of a larger number of data points resulting in a greater number of local updates? Having different ENVs essentially fixes your tau. So, make sure that the tau is consistent among all agents, right? So, that is one way you can potentially avoid this kind of computational heterogeneity. That is one thing, but what if we, let's say, dedicate a fixed amount of processing time? So, if we dedicate a fixed amount of processing time to individual clients, we can determine the number of local updates. So, what do you think will happen? So, essentially, we can have only a capital "T" number of clients.

Maybe the capital T refers to the total time that each client can spend performing local updates. So, what do you think would happen in that situation? Yeah. So, this is it, right? So, if there are slower clients, So, if there are slower clients, a slower client would make fewer updates. So, if there are slower clients, a slower client would make fewer updates. Then the clients that are faster, right? Because if you fix the total computation time, that is what matters.

So, that is another source of computational heterogeneity. So if we fix let say tau across clients then slow clients will take much longer to finish No, I am saying that if, let us say, you fix tau and the number of local updates you are going to make, then slower clients will take much longer to finish. Additionally, if you fix the computational time, you will not be able to complete as many updates, right? Finish their updates, and if that is the case, what are the consequences of this particular situation? The central server would have to wait for a certain amount of time before it can make the next update or initiate the next communication round. So, this is essentially a bottleneck.

So, it is bottlenecked. Each communication round is important. A quick way to fix this is to allocate capital T times your total computational budget. So, this is called straggling, by the way, and I mean these are the stragglers. So, if you want to eliminate the straggling effect, you need to address it. Some capital T's allow clients to make as many

updates as they want within that allotted time and project.

* If we fix τ across clients, then slow clients will take much longer to finish their updates bottlenecking each communication round.
* A quick way to eliminate the straggling effect is to fix total computational budget to T and allow clients to make as many updates as they can.
* However, faster clients will make more local updates.

[Source of computational heterogeneity]

Now, the downside of this effect is that the faster clients will make many more updates than the slower clients, right? Corrected: Therefore, we will make more local updates; however, we will also consider other factors. And again, this would have a similar effect to what we observed in the context of having a different number of data points, right? So, the client with a larger number of data points was making more updates; this is a very similar situation. The faster clients here will make a greater number of local updates within the allotted computational budget, correct? So, this is another source of computational heterogeneity. This is another source of computational heterogeneity. So, besides these two sources, can you think of any additional sources of computational heterogeneity? If you look at this particular algorithm, for instance, you will see its unique features.

So, let's say that every agent starts using different learning rates or hyperparameters, right? Maybe they are using a second-order optimization algorithm, such as Adam or something else. So, not only the step size, but perhaps the momentum term will also be included. Therefore, not only will the step size be included, but perhaps the momentum term will also be included. So, now for every given gradient—let us say, the gradient.

* Variation in the hyperparameters, such as learning rates and momentum is another source of computational heterogeneity.

Agents may be making, let us say, faster updates in the sense that they could be moving too much in the direction of the gradient, or perhaps a particular client is using a very small step size. So, even though it may be making many local updates, it is not progressing much toward what it should have achieved. So, the differences in the learning rates and the differences in the hyperparameters are other sources of computational heterogeneity. Is this clear ? Variation in hyperparameters, such as

learning rate and momentum, is another source of concern. So, why do we care about computational heterogeneity or other forms of heterogeneity in federated learning? As I mentioned, we are trying to work with clients that are either computationally very different in terms of computing power or in terms of the kind of data that each client possesses, which is also likely to be highly variable.

And so, in the presence of all these uncertainties, what kind of guarantees we can provide is what we are trying to understand in today's lecture. Therefore, in light of all these uncertainties, we are trying to understand what kind of guarantees we can provide in today's lecture. So, it is essentially about variations in computing power, variations in hyperparameters, variations in the total number of local updates, and so on. Can we still guarantee something meaningful within the framework of federated learning? So, that is what we are trying to study.