Distributed Optimization and Machine Learning

Prof. Mayank Baranwal

Computer Science & Engineering, Electrical Engineering, Mathematics

Indian Institute of Technology Bombay

Week-12

Lecture - 44: Convergence Analysis of FL

So, just like we had the just like we had this algorithm this theorem for decentralized SGD We also have a similar result on federated learning and how and when federated learning works and what are the sort of error rates that we can or error convergence rates that we can provide similar to the centralized SGD that we looked at and the decentralized SGD that we looked at. So, the assumptions, the key assumptions, it is going to be very similar to what we had made during decentralized SGD. So, the first thing is the Lipschitz smoothness of the local objective function. So, that means function fi is L Lipschitz. for every, so this is true for every function, every local objective function and for any points x and y. This is the Lipschitz smoothness of the local objective function.

Then we assume that the gradients are going, stochastic gradients are unbiased estimates right? So, that is another assumption that we had made in the context of decentralized SGD. So, stochastic gradients is an unbiased estimate. So, meaning the expected value of the stochastic gradient is equal to ok. Then we are also assuming even in the context of decentralized SGD that the stochastic gradients have bounded variance.

Stochastic gradient $g_i(x;\xi)$ is an unbiased estimate of $\nabla F_i(x)$.

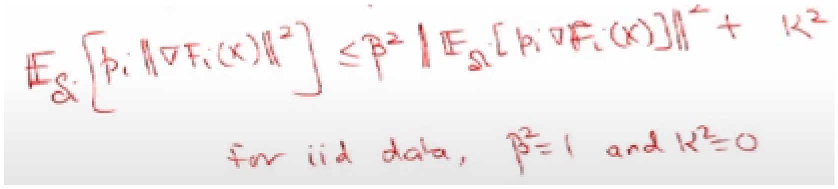
$$\mathbb{E}_\xi[g_i(x;\xi)] = \nabla F_i(x)$$

So, essentially to say that variance of gi this is going to be bounded or the expected value of combining the unbiased gradients and the bounded variance.

$$\mathrm{Var}(g_i(x;\xi)) \leq \sigma^2$$

$$\underline{\mathbb{E}_\xi[\|g_i(x;\xi)\|^2]} \leq \|\nabla F_i(x)\|^2 + \sigma^2$$

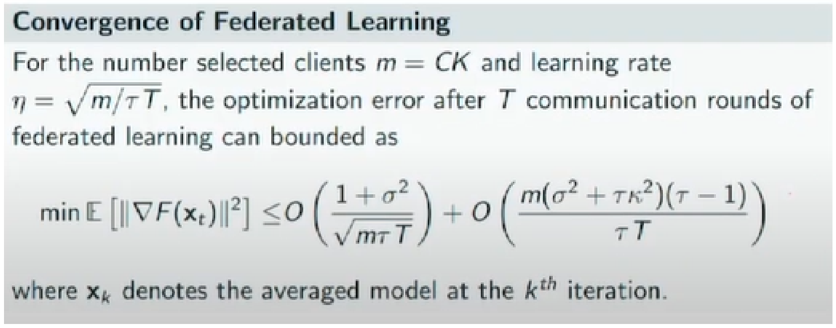
So, this is what you get ok. And finally, bounded dissimilarity is again related to data heterogeneity the kind of assumption that we made in the also made in the context of decentralized SGD. So, bounded dissimilarity says that there exists there exist parameters beta square greater than equal to 1 and kappa square greater than equal to 0 such that. square plus and if for iid data if the data is iid.



$$\mathbb{E}_{S}\left[p_i \|\nabla F_i(x)\|^2\right] \leq \beta^2 \|\mathbb{E}_{S}[p_i \nabla F_i(x)]\|^2 + \kappa^2$$

for iid data, $\beta^2 = 1$ and $\kappa^2 = 0$

So, you have beta square equal to 1 and kappa square is equal to 0. So, assuming you have these assumptions you can show that if you work with m clients which is c times k, c is the fraction of clients that you are going to be working with. You choose a learning rate, which is basically like this where tau. So for now we are going to be assuming that every agent performs the same number of local updates, which is going to be tau. So tau sub i is what we had, but then we assume that it's the same number of local updates that every agent works with.



Convergence of Federated Learning

For the number selected clients $m = CK$ and learning rate $\eta = \sqrt{m/\tau T}$, the optimization error after T communication rounds of federated learning can bounded as

$$\min \mathbb{E}\left[\|\nabla F(\mathbf{x}_t)\|^2\right] \leq O\left(\frac{1+\sigma^2}{\sqrt{m\tau T}}\right) + O\left(\frac{m(\sigma^2 + \tau\kappa^2)(\tau - 1)}{\tau T}\right)$$

where $\mathbf{x}_k$ denotes the averaged model at the k^{th} iteration.

So if we have this, then after T communication rounds, capital T communication rounds from the server. to T community central server, this is how you can bound the error in terms of the total number of iterations. So, now if you want to get epsilon close and you would have to assume that this is less than equal to let us say epsilon over 2 and epsilon over 2 and then that basically gives you an estimate of the total number of iterations required capital T to get epsilon close to the to make the error epsilon close right or epsilon small. Is this clear? So this is very similar to what we the result that we looked at in the context of decentralized SGD. Obviously we have more parameters here because in federated learning let's say the number of selected clients is an additional hyperparameter that we work with or the number of local updates that we perform which is tau that is another parameter that we work with.

So this is the result. So let's try and understand like basically revisit the basic understanding of federated learning and see what different how different parameters affect the learning behavior. So, the first question is. So, does the convergence between error and communication rounds. So, that improve does it improve with or deteriorates with the if you change these hyperparameters? So, essentially this error essentially convergence of error as a function of total number of communication rounds.

Does it improve if you increase the fraction of participating workers? What do you think? You also have the result here, but what do you think? Does it improve as you increase the number of participating workers or clients? If you gather information from more clients, do you think the error would converge faster? right. So, it is better as you increase the value of c right, something that we have already looked at which is over here. As you increase c, you see the total number of communication rounds required that gets become smaller. So, as you increase m, you see you have a square root of m kind of thing which basically supersedes this linear growth again right? So, that is one thing. If you increase the mini batch size, what happens with the total number of local updates? It becomes smaller right? So, if you have smaller number of local updates, we saw that you would require more rounds of communication right.

So, this becomes worse. What about increase in total number of epochs? Better right. So, it will be better. But do we really want to increase the total number of local epochs to a significantly large number? So in most cases it is better. Like if you have too many local updates happening at the same time and we will look at one particular example.

But if we had too many local updates happening, so then what would happen is that the model would overfit on the local training data. So, when you aggregate information from everyone, so you would have to first of all unlearn the overfitted behavior because your objective is to get a common set of features from every client right. So, then it would require more number of more rounds of communication. So, in general as you increase E, increase the total number of local epochs. it basically improves the convergence behavior in the sense that you would require fewer communication rounds to get to the same error, but beyond a point if it once it starts overfitting them like the model starts overfitting the data that is when you will have an issue right.

So, it is largely better, but if E increases too much. then they can be overfitting right. What about higher data heterogeneity? If you have higher data heterogeneity across clients, do you think does it improve the performance or makes it makes it worse? Worse right. So, this would be worse. right and what about these parameters beta and kappa.

So, beta and kappa if these parameters increase that means there is more dissimilarity right? So, if you look at the assumption beta square is greater than equal to 1 and kappa square is greater than equal to 0 and if you increase them beyond this for iid data beta square is equal to 1 and kappa square is equal to 0 right. So, as you increase beta and kappa that means you are tending more towards heterogeneous like non-iid data and this will also worsen the performance. So, this would also become worse. Is this clear to everyone? So, let us see how what happens to the anticipated wall clock time per communication round. So, during each communication round.

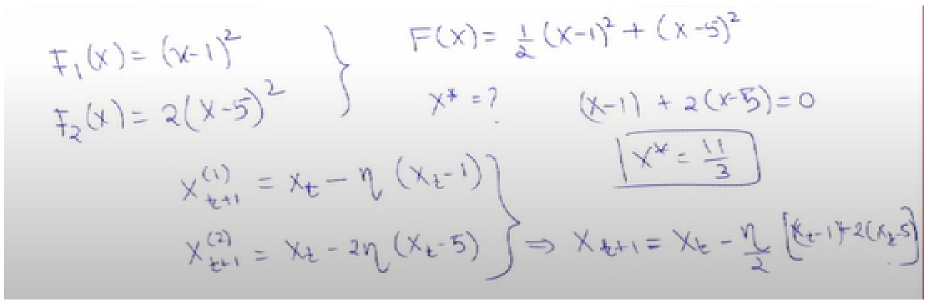
So, when what do you mean by each communication round? The central server basically runs. So, this server update is essentially what we are talking about. So, how much time does it take for this server update to happen? So, each communication round. So, as increase in the fraction of participating clients, what do you think happens to the wall clock time? You are going to be communicating with lot more workers or lot more clients. So, that increase in the fraction of participating clients would increase in the wall clock time, the time it takes right.

So, this basically increases. What about increase in mini batch size? it will, so if you increase the mini-batch size that means you are doing full gradient descent and usually it takes time then to perform gradient descent in batches. But it like, so it can both increase and decrease depending on the model parameters. So there is no conclusive sort of way to say that wall clock time will always increase. Let's say if I have, if I work with one batch, batch size of one and I have hundreds of examples, so I have to run a for loop for hundred times right.

If I have batch size of five maybe I will have to run for loop 20 times. So this probably works better but if I work with a batch size of 100 maybe I have to store 100 data points and like extract it and so on. So it really depends on model hyperparameters but it would mostly increase but it may increase and decrease depending on the model parameters. So, there is no conclusive way to say depends on how many hyper like model hyperparameters are there and how many model weights are there. What about increasing the total number of local epochs? Increase right? So, this would increase.

Data heterogeneity? it does not affect right data heterogeneity has nothing to do with that wall-flop time per per communication downright. So, it does not affect. Similarly, if you increase beta and kappa that also does not matter right because this is related to data heterogeneity and that does not matter. So, the last thing that I wanted to talk about which is stressing upon this particular point here, this overfitting issue right as we mentioned. So, should we use I mean as we had seen in this particular example on training two-layer neural network or the CNN for MNIST dataset. As we increase T, as we increase the total number of local epochs capital E basically requires that basically translates to fewer number of communication rounds. But is it always the case and I mean it may it will be the case that it will largely require fewer local rounds, but there is an another issue with this and with this data heterogeneity. So, let us say you have two functions there are two clients just consider very simple like should we have more number of local epochs or not. So, that is what we are trying to address.

Let us say you have two functions and function 1 is x minus 1 whole square and f 2 is 2 times x minus 5 whole square ok. So, if I look at since I mean typically we are looking at functions for which stochastic I mean you have data-driven evaluation of functions right like on the loss function. So, in this case we are assuming deterministic functions no stochastic gradients. So, that global objective would be let us say we use 50 50 percent like 0.5 0.5 of those. So, it would be half of x minus 1 square plus x minus 5 square. ok this is the global objective function. So, what is x star here the optimal solution? So, for that if you were to compute the gradient and then set it to 0 right. So, that would be x minus 1 plus 2 times x minus 5 you set it to be equal to 0 and this gives you x star is equal to So, this is the optimal solution that you expect the neural network or the federated learning framework to converge to.



$$F_1(x) = (x-1)^2 \qquad F_2(x) = 2(x-5)^2$$

$$F(x) = \frac{1}{2}(x-1)^2 + (x-5)^2$$

$$x^* = ? \qquad (x-1) + 2(x-5) = 0 \qquad \boxed{x^* = \frac{11}{3}}$$

$$x^{(1)}_{t+1} = x_t - \eta(x_t - 1)$$

$$x^{(2)}_{t+1} = x_t - 2\eta(x_t - 5) \Rightarrow x_{t+1} = x_t - \frac{\eta}{2}\left[(x_t - 1) + 2(x_t - 5)\right]$$

And this was largely this would largely be the case let us say if I let us say perform just one round of local update. So, then agent 1 would have x1 t plus 1 is ok and likewise x2 t plus 1 is x t minus 2 eta times x t minus 5 ok. So, this is just using one local epoch right. So, this would give you if I just average this, this is nothing but saying that x t plus 1 is going to be simply x t minus like if I just combine 0.

5, 0.5 of this, it is going to be eta over 2 times x t minus 1 plus 2 times x t minus 5. This is just performing one round of look like basically one round of local epoch or one local epoch. And when this converges, that means the gradient becomes 0 and you would see that xt would converge to the optimal solution which is 11 by 3. But what happens if I perform multiple rounds of updates? If I perform too many rounds of update, locally what would this converge to? xt would converge to or xt1 would converge to? Just one, because it will converge to its local objective value, local optimal solution Similarly this would converge to 5 and now if you take 50-50% of those it is going to converge to 3 instead of converging to 11 by 3 right. So you do not want to overfit or you do not want to perform too many local updates at the same time.

so that I mean so that you avoid potential overfitting and so that you can also ensure that the model converges to the global optimal solution of the global objective function and not the combination like some weighted combination of the optimal solutions of the global objective functions ok. So, that is the effect of data heterogeneity. if you have like in this case we have heterogeneous data. So, we are assuming they have different local objective value or different local optimal solution. And in that case, if you perform too many local updates, you would have this issue that it does not like the federated learning framework does not converge to the desired optimal solution.

Is this clear? So yeah, that is all I wanted to cover in today's lecture. So, we will look at more of federated learning in the coming weeks. Thank you.