Distributed Optimization and Machine Learning

Prof. Mayank Baranwal

Computer Science & Engineering, Electrical Engineering, Mathematics

Indian Institute of Technology Bombay

Week-11

Lecture - 43: FedAvg Algorithm

So, in the fed average algorithm that we talked about right in the original federated learning paper by McMahon here. So this particular paper, it introduces this fed average algorithm. So it has its local objective function at individual clients and that it has its global objective function. So what would be the local objective function that it is trying to solve? So that is going to be fi of x and that is going to be, so how many data points are there at the ith client? So this is local objective function at the ith client. So, this is a function. So, eta j is basically, so there are ni sub points.

$$F_i(x) = \frac{1}{n_i} \sum_{j=1}^{n_i} f_i(x, \xi_j)$$

So, this is the local objective function fi of x, which is the average of all fi's in some sense. What would be the global objective function f of x? So, local objective function is at the ith client, global objective function is basically from the perspective of the central server. So, what is the global objective function? something that we have already been looking at. When we talk about distributed optimization, what is the objective function that we are trying to minimize? Every agent has its own private objective function f sub i.

$$F(x) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(x) = \sum_{k=1}^{K} p_k F_k(x)$$

where $p_k = \frac{n_k}{n}$

So, sum of it, but instead it will be a weighted sum of it, weighted by the number of data points. So, essentially it would be, let us say k equal 1 through capital K, let us say n is the total number of data points times k of x which is where this fraction p k is essentially nk over total number of data points. So, it is basically the weighted sum of the local objective functions. Is this clear? So, let us look at how this particular fed averaging algorithm works. So we have two different updates, one is the server-side update which is

on the global objective function and then you have local objective function, so we have the client-side update right.

So as I said what this algorithm does is basically it communicates xt at each let's say at the tth round, tth communication round it communicates xt the current set of weights. Now using those weights every edge device or the participating edge device would actually train a model would and then they would get their own let us say new set of parameters, let us call them xti for the ith client and then they will be aggregated together. So, they will be performing tau i number of local updates here right, something that we have already looked at. So, they will be performing tau, tau sub i number of local updates and then this information would then be sent to the centralized server. So let's see how this particular algorithm works.

So this algorithm has two components. One is the server update. This is so we are describing the fed average algorithm here. So you have the server update. So, in server update what happens is that the first thing is the initialization.

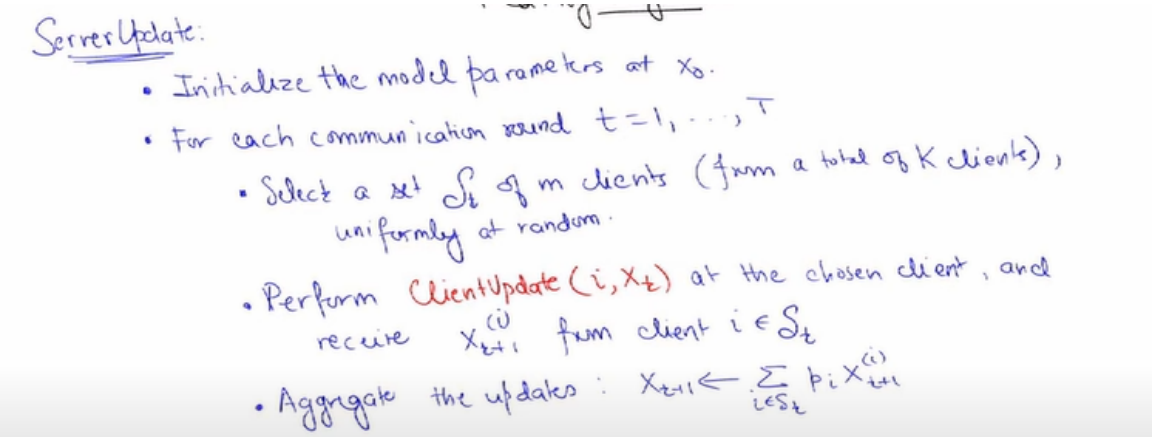
So, initialize the model parameters and let us say x naught ok. Now, for each communication round. So, for each communication round t. So, t is going to be a number from 1 to some capital let us say T minus 1. So, what happens or if it is the start from 1 then there is no need of T minus 1 here, ok.

So, for each communication round we select a set St of m clients. So, you randomly select a set of m clients out of those K, capital K available total clients. So, m clients from k clients. So this is selected uniformly at random. Then we perform a function called client basically we run this function called client update.

So client update would have the client index i and to the client it would be sending its current weights or the current estimate of the weights which is going to be Xt right and the chosen client and receive. So, what are you going to be receiving? You are going to be receiving x t plus 1 i. So, after client i basically uses this current weight x t and then it performs multiple local updates, it is going to have an updated value of x t, let us call this x t plus 1 i, where i indicates it is the ith client. So, it receives that information from the from client i in basically in your set st right. And then essentially aggregate So, this is what you do.

So, this is the server side of things and look at this particular function client update. So, let us see how this particular function works. So, what you are going to do is you are going to initialize the weights. initialize the model with xt right whatever xt that you are going to be receiving you are going to be initializing the model with that subset of it. So,

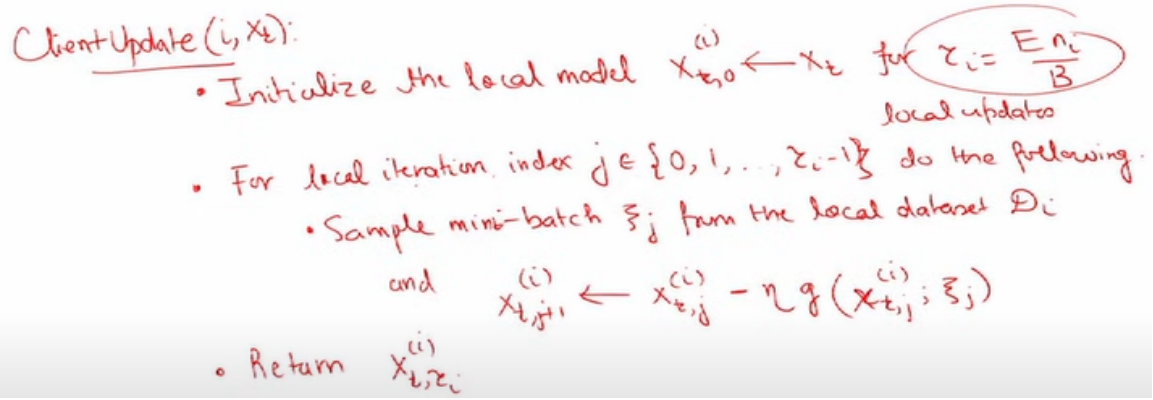
initialize the local model is Xt let me use a different notation.



So, we are going to be using xt 0 to indicate. So, first of all it is going to be performing tau i number of local updates the ith client. So, this 0 is going to indicate the number of like basically how many updates have elapsed. And i is going to indicate that it is going to be the ith client, t is basically indicates the communication round. So, it is a tth communication round.

So, this is going to be essentially xt right. So, you are going to be initializing this with xt for and you run this for tau i number of local updates which is a n i. So, we are going to perform these many local updates. Next, what do we do? each local update index j in 0, 1 tau i minus 1. So, what do we do? Do the following.

So, we perform a gradient descent on the mini batch that we are going to be selecting during each local update, ok. So, that is essentially the sample mini-batch, sample mini-batch, let us call that mini-batch I think we will just use the same notation. So, zeta j from the local data set. d sub i right. So, every agent has got this every client has gotten this data set d sub i and we are going to be sampling a minibatch zeta zeta j from that data set and on that data set you perform.



So, you are going to get x t j plus 1 as x t j at the ith client minus the step size times look at this stochastic gradient g evaluated at x i e j ok. So this is what you are going to be

doing. So essentially performing a gradient descent on the local batch and then you will be having these many rounds of local updates. And after this is done, so you are going to be returning So, what do you need to return? x i t tau, tau sub i. So, that is what you are going to be returning.

And once you return this, so this is treated as x t plus 1 i from the client side perspective and then it basically aggregates. So, this is the fed averaging algorithm. Is this clear? So, essentially there are multiple parameters involved. One is the the total number of local epochs which is capital E, the batch size is going to have a role and then there are other parameters like data heterogeneity and others that are going to have a role. The number of clients that you end up choosing a sampling that is also going to play a role essentially the fraction of clients that are involved in each round of communication.

So, that is so let us see how these different parameters affect the learning behavior and then we will try and come up with a mathematical result that basically captures our intuition and so on. So, this is an exam this is the result from So, we essentially trained two different types of neural network. So, is everyone aware of this MNIST data set which is a data set of handwritten essential digit data set from 0 through 9. I think it is a 28 by 28 cross 28 sort of binary-like image or grayscale images. So, in this experiment two different types of neural networks have been trained.

One is the two-layer simple feed-forward neural network and the other one is a convolutional neural network. And essentially what we show over here is the total number of communication rounds needed to ensure that this two-layer neural network is basically attains this 97 percent accuracy and the CNN that achieves 99 percent accuracy ok. So, on MNIST if you design a good enough neural network you can easily get 99.4, 99.6 kind of accuracy. So, for CNN to achieve 99 percent accuracy it is not something unheard of on MNIST. So the first thing that we, so I mean two different types of experiments were performed when you had the IID or the IID data distribution across multiple clients. So in that case, so in this case we are considering 100 clients. So MNIST dataset if you know, it has 60k images. So 60k images means, 60k images of, so that means you have 6k images of each label, basically from 0 through 9.

And when you have these 100 clients, they are each receiving 600 examples. So you randomly shuffle and partition. So on an average, every agent would have the same number of, so everyone would have the same 600 images to work, like similar distribution of 600 images to work with. So that means out of those 600 images, on an average, 60 would be from digit 0, 60 would be from digit 1, 60 from digit 2 and so on for every agent.

So the data distribution is IID. In the non-ID experiment what was done was the data was sorted by labels. So essentially you had all the zero digits followed by one digit. So instead of random shuffling and partitioning, it was sorted and then sort of partitioned. So divided into 200 shards of size 300 each. So every agent and then for each client you essentially share two shards.

So essentially what you end up doing is every agent would have these 300 images from specific examples. So, at a time every client would only have at would only have two sort of unique digits to work I mean only two digits to work with. So, essentially either 0 1 similarly other client would may have 0 2 and so on right. So, that is that is how that is the non-ID experiment.

So, the data distribution. So, each client does not have let us say visibility to every like other images right other than those two classes it would not have visibility to all other instances. But because other models are also being trained on this and that there is information sharing with central server which is aggregating information from all other clients. So it is learning from their sort of training. So, that is the non-ID experiment and you would imagine that if it is a non-ID experiment the number of total number of communication rounds is going to be required for the similar setting it is going to be much larger than the ID experiment that is what you see everything else remaining the same. So, this is the ID setting this is the non-ID setting.

So, you can see that the total number of communication rounds required I think here you would see maybe a drastic change. and so on. So, this hyphen indicates that I mean the model could not reach this particular accuracy. So, essentially you could not train the model to the desired accuracy. So, that is when the I mean the experiment could not be completed and that is noted by this hyphen here.

And you can see that when you have 75 and 70 whereas with non-IDKs you had 443, 380 number of communication rounds and so on. Similarly, over here you have these many. So, when you have data heterogeneity, if the data is going to be heterogeneous, then total number of communication rounds required to reach the desired accuracy that is for certain error threshold that is going to be much larger. So, that is the effect of data heterogeneity. So, B is equal to infinity means you run the full batch gradient.

So, the batch size is essentially the entire data set whereas, B equal to 10 means you have you have a mini-batch of size 10 ok. So, we are looking at the effect of data heterogeneity. So, that is clear if the data is going to be heterogeneous you would require I mean many number of communication rounds then if the data is homogeneous. The other is about client participation. So when we see that the fraction C is tending towards

1, this number keeps on reducing, right? The number of communication rounds required keeps on reducing.

So that means as you are aggregating information in the same communication step, in the same communication round, if you are aggregating information from let's say all the clients then you would require the fewest number of communication rounds because then you would you would receive the entire data distribution whereas let's say this would be particularly relevant in the context of heterogeneous data right. So when you have heterogeneous data and you are not aggregating information from all the clients so that means you may be missing out on certain digits and that's why the communication I mean the number of total number of communication rounds required may go up. As you increase this thing, the total number of communication rounds required basically goes down. If more and more agents are involved or more and more clients are involved, the total number of communication rounds, it basically decreases. Similarly, if the data is homogeneous or the data heterogeneity is lesser, the total number of communication round also becomes lesser.

Is this clear? So, this is about the effect of total number of local epochs or capital E. So if capital E is lesser that means you are performing fewer local updates and if you are performing fewer local updates then you would require as you can see if you are performing fewer local updates then the number of number of round communication rounds required is much more right. So, but then for the same set of weights, if you sort of train the model enough times on this, then basically perform multiple local updates on the same data set, then the communication total number of communication rounds required that sort of starts reducing as we So, this is for the MNIST CNN example, then there is a Shakespearean data set. So, this is training in LSTM for predicting the next world based on the previous words and again you can see as you increase the total number of local epochs and now total number of rounds required basically keeps basically decreases right. So, what about the bath size, mini bath size? How does that affect? So, first of all, it is clear to everyone that as the total number of local epochs increases, you need fewer communication rounds to reach the target accuracy.

That is clear, right. What about the batch size? So, if you have a larger batch size, you would require fewer local updates, right. If you have a larger batch size, tau sub I was E and ni by B, right. If you require larger, if you work with larger batch size, you would have fewer local updates and fewer local update means that essentially you increase the total number of communication rounds, right? As we saw that when E was a small, we had fewer local updates and in fewer local updates, we needed more communication rounds. Similarly, if you had larger batch size, that means fewer local updates and therefore you will have more communication rounds and that's what you see, right? If

you have a larger batch size, the communication rounds required was 2488 if you have a smaller batch size then it becomes lesser similarly you have 401 here and when it is 10 it becomes 192 and so on. Is this clear? So, essentially if you have larger number of local updates, then you would have actually fewer communication requirement of fewer communication rounds to reach the target accuracy and that is how E and B are affecting your essentially E and B affect your tau i and tau sub i and therefore, they eventually affect the total number of communication rounds required.

Is this clear? So, this is just illustration of how the batch size sort of impacts the total number of rounds required and you can see that when you have smaller batch sizes you reach the target accuracy much sooner than if you have the if you have small larger batch sizes. Thank you very much.