Distributed Optimization and Machine Learning

Prof. Mayank Baranwal

Computer Science & Engineering, Electrical Engineering, Mathematics

Indian Institute of Technology Bombay

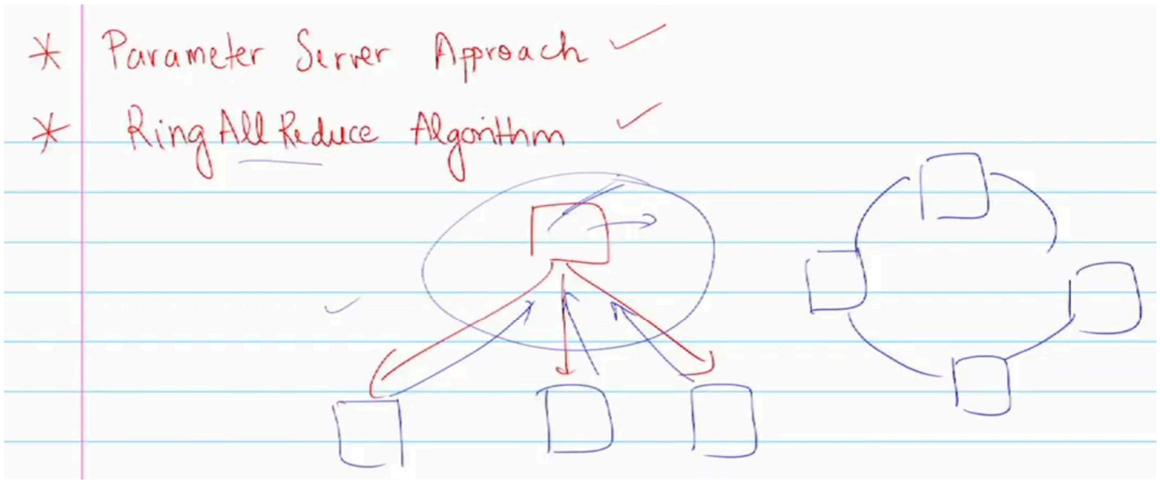
Week-11

Lecture 40: Decentralized Stochastic Gradient Descent

Alright, so in the last lecture, so we looked at two different approaches, one was the parameter server approach. and the other one was ring all reduce algorithm right. This is so both these algorithms were used for distributed training of let us say neural networks when you had data points distributed across different servers or workers right. So, what was particularly attractive about the both like about both the approaches. So, what did we have in the context of let us say parameter server approach? So, you had a central like it would require fewer iterations or fewer rounds of information exchange right. So, there is a centralized server or parameter server that would relay information about the current weights to the individual workers or servers, they would compute gradients on their on their local devices or it may be they would compute the gradients locally and aggregate and basically send the gradient information back to the parameter server which is then going to aggregate the gradients and use it to update the weights of the neural network right.

So, the good thing about this particular approach is it has this nice structure it I mean it works fine the everything is synchronized, but the shortcoming of this particular approach is the bandwidth requirement on the centralized server. if it is communicating with n different agents. So, the bandwidth cost is going to be significantly. On the other hand we had ring all reduce algorithm where it connected.

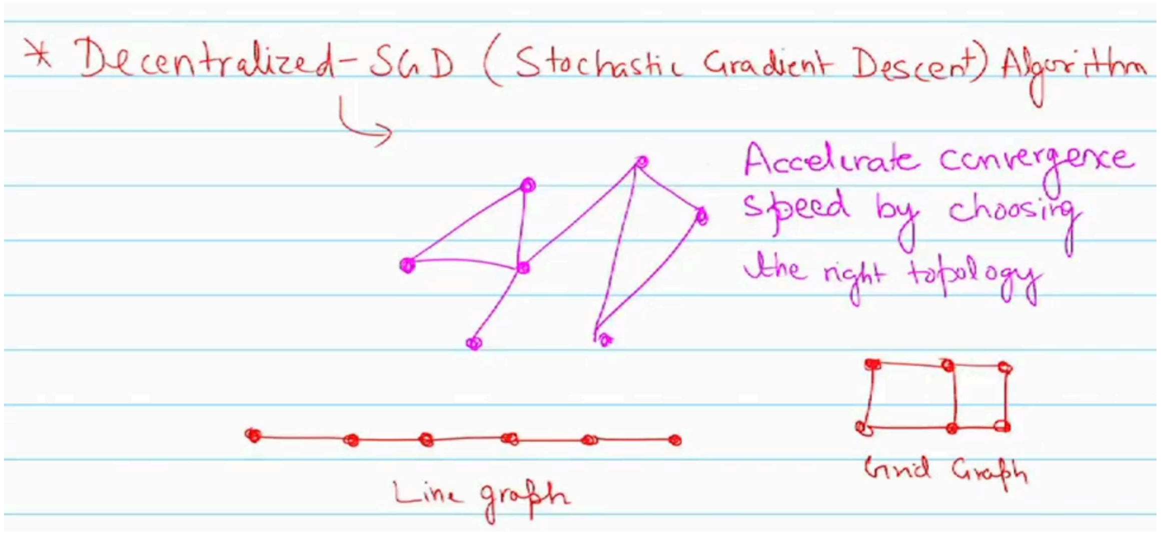
So, there was no centralized entity different servers were connected in a ring kind of fashion and they would use scatter reduce and all gather steps again in a ring kind of fashion and this through this they would be able to aggregate the gradient information at with the communication cost being order 1 or constant in the number of agents, but the shortcoming is that you would require multiple such rounds right as opposed to single parameter server approach. So, in both the approaches though we had assumed that every agent or every worker has the same W naught to start with right. In this case anyway there is it is being in the parameter server approach it is anyways being monitored through the centralized server. that wk is being synchronized with all the server, all the workers at all times. But in case of let's say ring all reduce, we assume that w0 was known to every agent a priori and it's the same w0 that every agent has and then they compute the gradient information and then every agent ends up computing the same w1 because they have the same gradient information.



And then it's the same w1 that every agent now has, so they would again end up computing the same w2. But if you really want to decentralize the training of neural networks, where every agent starts with their own estimate of how the neural network should look like, what should be the weights of the neural network let's say. Every agent has its own copy of W naught or rather has its own estimate of W naught. And then in that case you would also need a consensus on W naught, not just the gradient aggregation part, but there would always be a consensus step, which is similar to distributed optimization algorithm or the DTD algorithm. distributed gradient descent algorithm that we looked at in maybe few lectures ago ok.

So, in the same context we are going to look at something popularly known as decentralized SGD or SGD or decentralized stochastic gradient stochastic gradient descent. this implies stochastic gradient descent algorithm right. So, this is what we are going to look at. When we talk about how quickly the algorithm would converge. So, what do you think would the factors be? So, let us say we want to run the distributed optimization algorithm right.

So, what could control the convergence speed of the algorithm? So, you have a network of agents. So, essentially we are working in this regime where you have let us say different workers or different servers. So, they may be connected through some topology like this right. So, the topology of the network is going to play a significant role in terms of the convergence speed right. so we can potentially accelerate convergence speed by choosing the right topology.

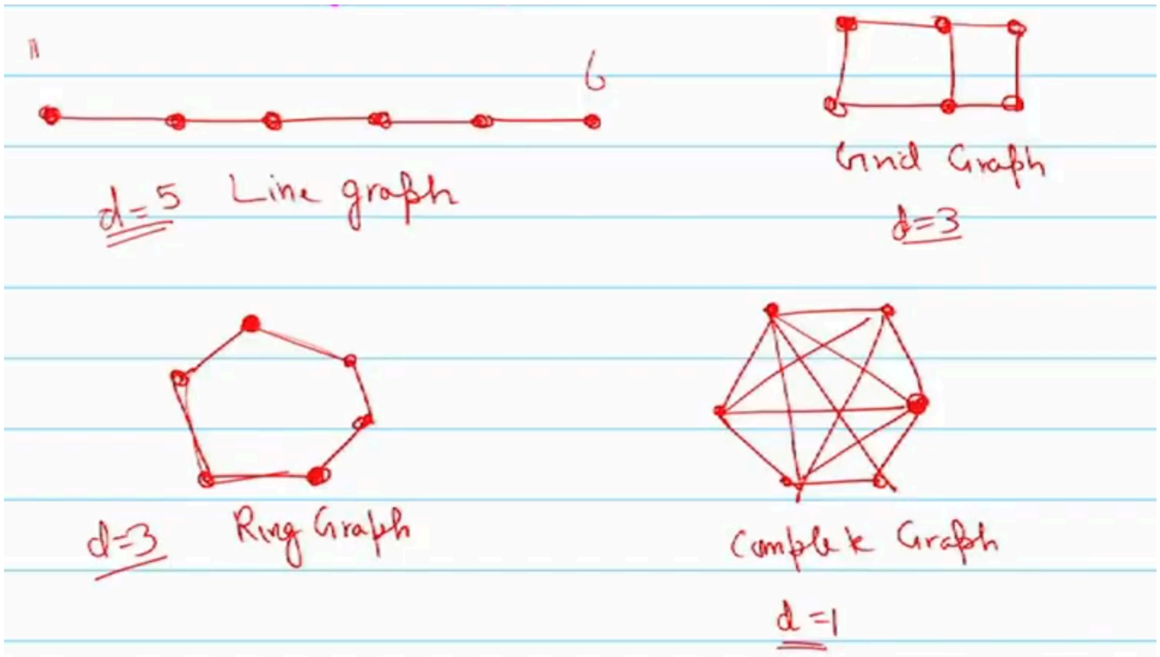

* Decentralized-SGD (Stochastic Gradient Descent) Algorithm
Accelerate convergence speed by choosing the right topology
Line graph
Grid Graph

So for instance, if we work with let's say we have n agents or n workers right and we connect these n workers using a line graph like this. So let's say that we have 6 workers and we connect them using line graph versus you have 6 workers where you connect them let's say using a grid graph. or you have you connect them using ring graph. So, these are common topologies and you can also have a complete graph right, where every worker is connected to every other worker basically it can exchange information with any worker whatsoever. So, this is your complete graph.

So, which graph do you think would have the slowest conversion speed if we run a decentralized stochastic gradient descent algorithm. So, again unlike the parameter server approach or the ring all reduce approach. every agent initializes at a different W naught right. So, not only they would have to optimize the weights of the neural network on their own data, they would also need to ensure that all of the agents they arrive at a common W right, because eventually it is the same neural network that is being trained locally and at different workers. So, the question is among these 4 topologies which one do you think would have these fastest or let us say the slowest speed of convergence.

The line graph, right? Why? Yeah, the Fiedler. So, essentially the diameter of this is large in the sense that I mean in this case the diameter is 5, right? So, you would need at least 5 steps for the information to propagate from let us say node 1 to node 6 or the worker 1 to worker 6. You would need at least 5 steps. So, diameter is 5. What about ring graph? What is the diameter? So, d is equal to 5 here which is diameter.

what about d equal to so 3 right 3. So, let us say you want to propagate information from this node to this particular node you would need 3 at least 3 steps in all to all other nodes you can reach from this particular node in 2 steps right. So, d is equal to 3 here. What about grid graph? What is the diameter of this particular graph? 3 again and complete graph what is the diameter? So complete graph is that every node is connected to every other node, right? So in one step you can reach. So which graph do you think is a better graph to work with when you want to run decentralized STD? Yeah, so if you ignore the communication cost, then complete graph would be an ideal thing, right? Because in that case, you would be able to communicate your information with your neighbors in just one step.



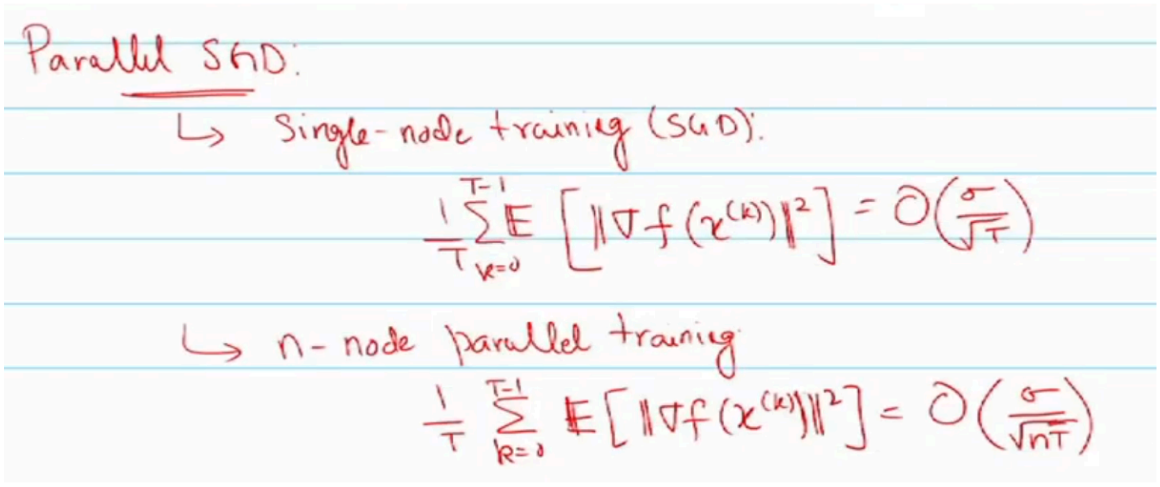
But the downside of this is this increases your communication cost right because every agent would be communicating with n minus 1 agents in every step and that would require a huge bandwidth cost for each of the agents right. Whereas with line graph every agent or line graph or a ring graph every agent would at most require to communicate with two neighboring agents. So, even though the diameters are small. I mean it may take more iterations to converge, but communication or the bandwidth cost is going to be much smaller for these graphs than something like complete graph. So, the question is does there exist a topology and a graph topology that would actually lead a very sweet I mean a sweet spot between or a good tradeoff between the communication requirement or the bandwidth requirement and the convergence speed.

And we would look at the answer to this particular question towards the end of the lecture, but let us try and see how like what are the iterations complexity of different algorithms ok. So, let us start with the. So, we know what parallel SDD is. So, let me sort of reiterate. So, in the last lecture towards the end of the last lecture we looked at single node training or single node stochastic gradient descent rate.

So, if you have single node training so that means everything is centralized there is there are no distributed workers there is just one server or one agent which has access to all the data. And if you want to train a model let us say if you want to train a model on a single server In the last class, we looked at one particular result where we showed that this expected value of summation k equal to 0 from t minus 1. This is of the order sigma over square root of t, where sigma was the bounded variance the stochastic gradients right. So, this was the result that we looked at towards the end of the last lecture which is that there is just one centralized server or one server to start with. All the data is aggregated at one particular server and you are using the entire sort of gradient descent like you are basically running the stochastic gradient descent and you are using that to update the weights and this is the kind of convergence guarantees that you have.

For an n node parallel training. So, let us say you have n you have this particular setup in the parameter server approach where the data is distributed across n nodes. So, every node sort of computes a gradient and then it sort of relays it back to the centralized server. So, for n node parallel training. So, you basically get a linear speedup.

So, what do we mean by that? So, expected value of essentially of order sigma over square root of n t. So, that means, if we want to achieve like if we want to achieve epsilon accurate solution, So, we would have to equate in case of single node training. So, we would have to equate this to let us say epsilon right. So, the value of t turns out to be sigma square over epsilon square. So, you would need these many iterations in order to get epsilon close to the optimal solution right or epsilon accurate solution for n node or n parallel node.



Parallel SGD:

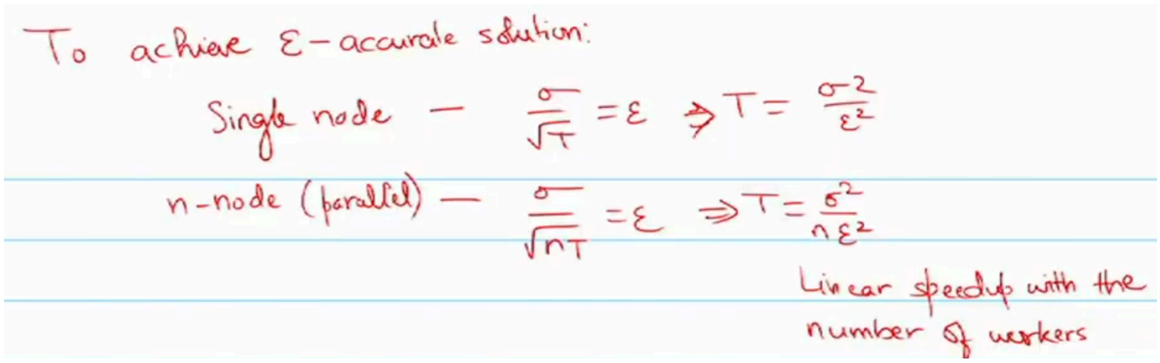
↳ Single-node training (SGD):

$$\frac{1}{T}\sum_{k=0}^{T-1} \mathbb{E}\left[\|\nabla f(x^{(k)})\|^2\right] = O\left(\frac{\sigma}{\sqrt{T}}\right)$$

↳ n-node parallel training

$$\frac{1}{T}\sum_{k=0}^{T-1} \mathbb{E}\left[\|\nabla f(x^{(k)})\|^2\right] = O\left(\frac{\sigma}{\sqrt{nT}}\right)$$

So, we would have sigma over square root of n t and this implies essentially it the number of iterations required is 1 over n of the. So, you would actually have a linear speed of So that means parallel STD would have linear speeder. So what does parallel STD do? So it basically has local computation of the gradient. So at each iteration k, it would compute the gradient which turns out to be gradient of the loss function that you are trying to optimize at the current value of x k and the data that is known to server i. This is your local computation and then you have x k plus 1 is simply x k minus step size this is a global communication step right.

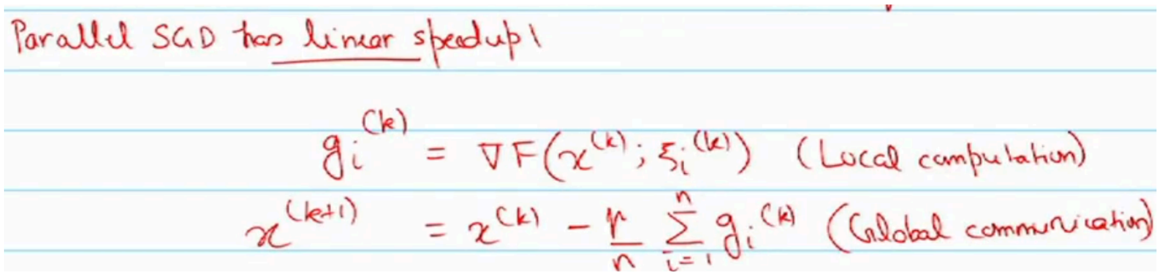


To achieve ε-accurate solution:

Single node — $\frac{\sigma}{\sqrt{T}} = \varepsilon \Rightarrow T = \frac{\sigma^2}{\varepsilon^2}$

n-node (parallel) — $\frac{\sigma}{\sqrt{nT}} = \varepsilon \Rightarrow T = \frac{\sigma^2}{n\varepsilon^2}$

Linear speedup with the number of workers



Parallel SGD has linear speedup!

$$g_i^{(k)} = \nabla F(x^{(k)}; \xi_i^{(k)}) \quad \text{(Local computation)}$$

$$x^{(k+1)} = x^{(k)} - \frac{\gamma}{n} \sum_{i=1}^{n} g_i^{(k)} \quad \text{(Global communication)}$$

This is when you need a global communication because you are going to be aggregating gradients from different servers. So, if I were to compare the two algorithms in terms when I say two algorithms which means ring all reduce and parameter server. So, let us in terms of the iteration cost or the bandwidth cost and the latency and so on. So, you have the algorithm here, you have bandwidth cost you have latency. So, and you have total cost which is the sum of latency as well as the bandwidth cost.

So, if I look at the ring wall reduce algorithm. So, what is the bandwidth cost of ring wall reduce algorithm? It is constant in the number of agents right. So, get a constant bandwidth cost, but what about the latency, we have 2 times n minus 1 iterations right, we have 2 times n minus 1 iteration, we have n minus 1 iteration for the scatter reduce and we have n minus 1 iterations for the all gather step. So, latency grows like order n, essentially total cost is order n plus 1 or order n right. If I look at parameter server approach.

what is the bandwidth cost for the centralized server order n right there are n agents n servers. So, latency is one step right in one step you would be able to aggregate all the information. So, latency is one step. So, if I look at the total cost it is again order n. So, in both approaches total cost is ok.

Algorithm	Bandwidth Cost	Latency	Total cost
Ring AllReduce	✓ $\Omega(1)$	$\Omega(n)$	$\Omega(n+1)$
Parameter Server	$\Omega(n)$	$\Omega(1)$	$\Omega(n+1)$

In both approaches, total cost is $\Omega(n)$.

* D-SGD (Decentralized SGD)

No global synchronization

D-SGD = local SGD update + partial averaging

Now the question is can we try and optimize this total cost which is basically a combination of the bandwidth cost and the latency. Obviously if you try and optimize this, if you want to optimize this So that means, we would be instead of an agent exchanging information with n neighbors or let us say, so in the parameter server approach, the centralized server exchanges information with n neighbors. In ring or reduce, it exchanges information with just 1 or 2 rather. So, somewhere in between where you have some like let us say, so because just you, I mean in this case you exchange information with just one neighbor, the latency grows up right because you would need that many iterations. order n number of iterations for this entire scatter and all reduce scatter reduce and all gather operation to get completed and therefore, the total bandwidth cost increases.

So, the question is how do we sort of optimize this where we basically we want to reduce this total cost which is a combination of both bandwidth cost as well as latency. So, we want to optimize this. So, we want to basically get to a sweet spot somewhere in between ok. So, you can also think of total cost as the per iteration complexity of the algorithm that is what we are trying to minimize and that is where decentralized SGD would be useful. So, let us look at decentralized SGD now.

$$x_i^{(k+1/2)} = \underline{x_i^{(k)}} - \mu \nabla F(x_i^{(k)}; \xi_i^{(k)}) \longleftarrow$$

$$x_i^{(k+1)} = \sum_{j \in N_i} \underline{W_{ij}}\, x_j^{(k+1/2)}$$

So, in decentralized SGD as we mentioned there is no global synchronization. So that means, we the agents they do not start or the workers they do not start at the same initial condition or initial weights or initial x naught. So, there is no global synchronization ok and it basically combines local SGD update and then you have partial averaging which is your consensus kind of update ok. Something that we have already seen as I mentioned if you remember the maybe the last few lectures we had already looked at an algorithm for distributed solving distributed optimization problem where we combine the gradient step and the consensus step right. So, that is what this particular algorithm is.

So, we define x i k plus half. let us say every agent has their own current estimate of the weights of the neural network which let us call them xik or xjk. So, we define xik plus half which is going to be defined using a gradient descent step. So, this is a local SDD update right. So, you have local SDD update followed by partial averaging and partial averaging is basically j over the neighborhood set of i, we have w i k. So, in terms of the algorithm what happens? Every agent performs this local local SGD update.

So, they update and they get let us say x i k plus half, x j k plus half and so on and then you get the x i k plus 1. which is going to be sort of averaging of the neighbors estimates. So, there is as I said there is no global synchronization unlike the previous examples of parameter server or ring already used where we always assume that the xik every agent has access to the same xik or every agent and then they again have like basically we try and get make sure that the agents end up getting access to the same aggregated gradient information. So, that their xik plus 1 is also going to be the same and the algorithm sort of keeps going on. Here there is no sort of global synchronization, we only have local estimates and then local I mean we kind of run a consensus type of update on the local estimate.

So, that these estimates are eventually synchronized. So, let us say you are like you have your own data and it is the same neural network that we are trying to train. but you have a local copy of that neural network, you start with your own set of random weights. I have my own set of random weights and while I perform a local computation or the local update on my own private data, I again basically exchange certain information and then through this information exchange, I try to get a sense of your data distribution so that the overall neural network which is going to be trained, it has the same set of weights. So, it is the same network that I am using that you are using or anyone else is using, right.

Is that clear? So what is the per iteration communication cost for this algorithm? What is per iteration communication cost? So, per iteration communication cost is going to be dependent on the largest degree of the node right. So, the node with the largest degree that is going to be exchanging information with most of the most of the neighbors right.

And let us say the largest degree in the graph is going to be d max. So, this is going to be the per iteration communication cost for this particular algorithm. And this is much much smaller than a parameter server approach where the per iteration communication cost is omega n right.

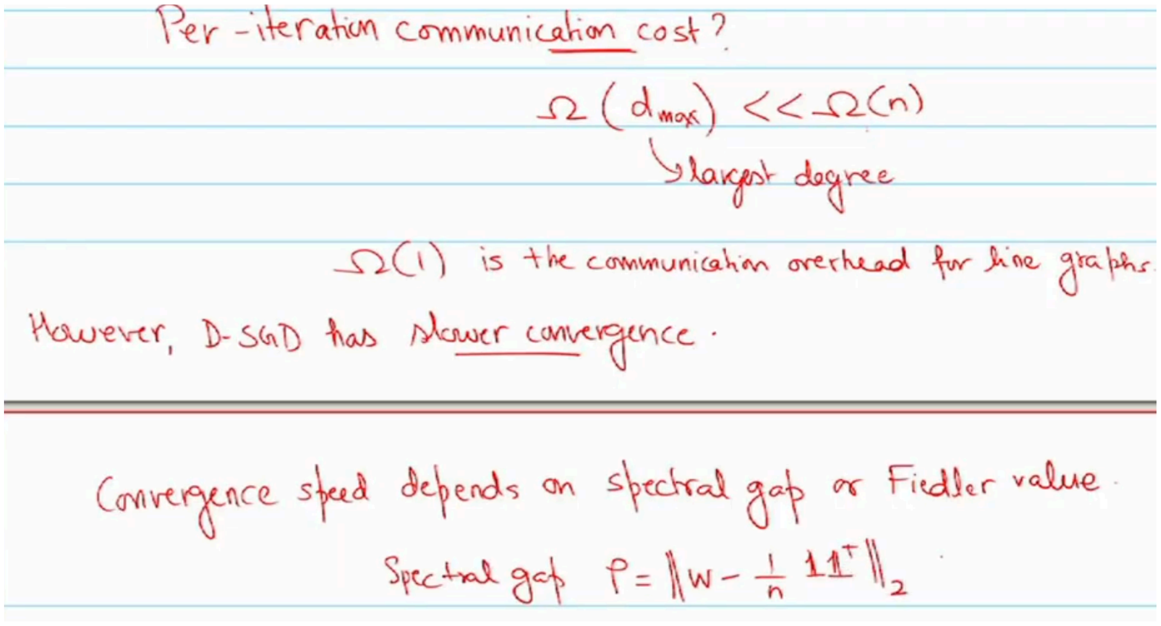
So, the point is this is your largest degree. When you say we want the optimal pathology means you want to optimize this W ij. We want to optimize this W ij, yes. not just wij it is also well I mean to some extent yes. So, when we when I say that we want a suitable topology that reaches a sweet spot between the bandwidth cost and the latency requirement and so on. Also there is something called transient iteration that I will come to.

So, not just like one thing is you fix the topology and you try and get the right set of wijs that is one thing. The other thing is you also alter the topology. So, it is not just about getting to the right values of wyj for a fixed topology, but also like how the nodes are connected is also going to as I said right like how the nodes are going to connected it plays a significant role in terms of how quickly the algorithm is going to converge ok. So, what is the communication overhead for a line graph it is. So, for instance for line graph it is essentially constant right.

is the communication overhead for line graphs. So, if using d is like decentralized would actually have very small communication overhead particularly when the graph is sparse. When the graph is sparse you would have very small communication overhead because if the graph is sparse then every node is connected to very few nodes in the graph. So, the graph is sparse then you would have very I mean the communication overhead is going to be small. But then what is the disadvantage or what is one particular shortcoming of decentralized compared to let us say the single node training or the parallel node training or the ring all reduce algorithm. It is this iteration number of iterations k that you are going to take to in order to guarantee similar convergence to epsilon accurate solution that is going to be significantly larger right because I mean there is no consensus on the weights.

So, not only we are trying to perform local gradient updates, we are also trying to perform a consensus or the local or the partial averaging right and that makes a task difficult right because you have your own estimate, I have my own estimate and then I am also trying to solve this problem cooperatively. So, essentially unlike the previous case where everyone was already synchronized. So, the amount of iterations needed to synchronize all the agents that is not there with the centralized training right. The moment we look at decentralized SGD we have an extra effort. So, the number of iterations required to get to epsilon accurate solution that is going to be larger ok compared to the centralized training approaches.

So, however So, DSDD is going to has is going to have a slower convergence because a number of iterations required you also have to take care of the consensus part which was not there with the centralized training ok. So, if you want to speed up this convergence what do you need to optimize for the graph? So, we look at Fiedler value or the diameter of the graph is something that we try and optimize on. But then if you want to, the moment you change it to a complete graph, that is when it gets optimized. The moment you change it to complete graph, but that increases the bandwidth cost, right. So, typically, so this conversion speed that depends on something called spectral gap.



Per-iteration communication cost?

$\Omega(d_{max}) << \Omega(n)$

largest degree

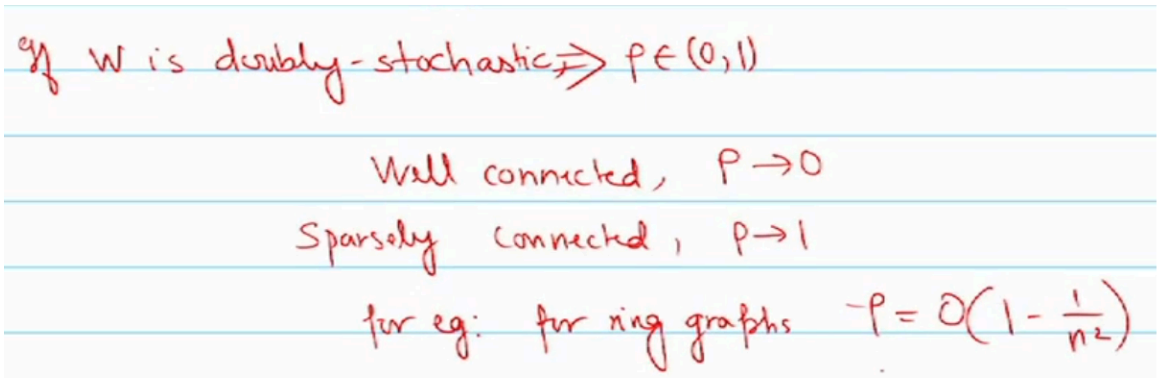
$\Omega(1)$ is the communication overhead for line graphs.

However, D-SGD has slower convergence.

Convergence speed depends on spectral gap or Fiedler value.

Spectral gap $\rho = \|W - \frac{1}{n}\mathbf{1}\mathbf{1}^T\|_2$

or Fiedler value. So, spectral gap is given by, if you look at the two norm of this matrix 1 1 1 transpose. So, this is your spectral gap rho. So, you can show that if w is doubly stochastic and rho the spectral gap is a number between 0 and 1. And so the graph is well connected that means it is close to a complete graph. What do you think the value of rho should be? Should it be large or small? Just by looking at this definition.



If W is doubly-stochastic $\Rightarrow \rho \in (0,1)$

Well connected, $\rho \to 0$

Sparsely connected, $\rho \to 1$

for eg: for ring graphs $\rho = O(1 - \frac{1}{n^2})$

So, if graph is well connected, all the w entries are going to be 1 over n, 1 over n, 1 over n kind of entries. So, this rho is almost 0. So, if the graph is well connected, so again just like Fiedler value spectral gap is also used to measure the connectivity. So, spectral gap, diameter, Fiedler values, these are sort of interrelated to each other. And again this quantity is also used to measure the connectivity of the graph.

And if the graph is well connected, rho is close to 0. And if the graph likewise is partially connected, then the row is going to be close to 1. So, for instance for example, for ring graphs your row is the order 1 minus 1 over n square say where n is in total number of nodes. So, as you make n very large and large right. So, it will take forever for the information to propagate and you can see that the spectral gap is going to be almost close to 1 ok. And that you can also imagine if it is a ring, you would need at least n by 2 steps to propagate your information with like all your Thank you very much.