Distributed Optimization and Machine Learning

Prof. Mayank Baranwal

Computer Science & Engineering, Electrical Engineering, Mathematics

Indian Institute of Technology Bombay

Week-1

Lecture - 1: Introduction to Optimization

So, welcome to the first lecture of SC 646 which is on Distributed Optimization and Machine Learning. So, some of you may have taken courses on optimization right, convex optimization, or maybe combinatorial optimization, and integer optimization. So what is distributed optimization? So can you give an example like do we use or at least I mean do we know of any real-life instance where distributed optimization is being used? When you think of Google board, G board, when you're typing something, you get the next word prediction. Now, from the point of your Google, let's say it has deployed a neural network-based model on your device, and that model pretty much is suggesting the next best word that needs to be auto-completed. So the learning for that, you have your own private data. He may have his own private data and so on.

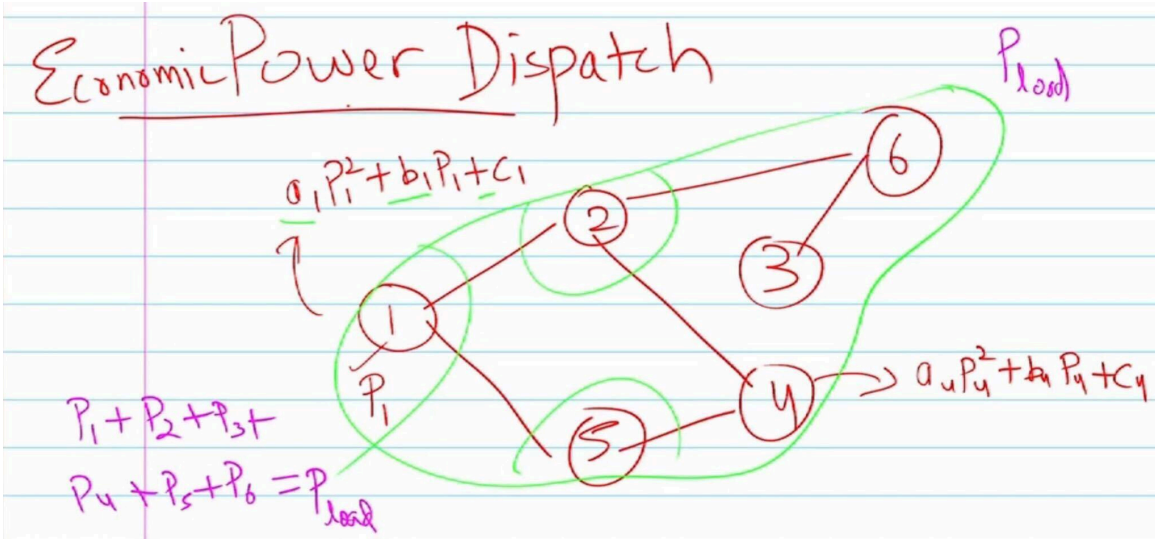
So we are not sharing the data every time with Google. We also have our privacy issues as well. So from the perspective of Google, what is it doing? It's collecting maybe. So it has a global copy of a neural network.

It basically computes a gradient. Gradient on your data, on your data, it accumulates those gradients, averages those gradients, and performs a global update for the neural network. And the new neural network with the new set of weights, they get deployed to your devices, right? And that's how you get better and better predictions every time. So in IITB, for instance, one very popular lingo is everyone uses max, right? Infimax, Coolmax, at least that used to be the case when I was an undergrad. So if more students from IITB, they get dispersed to different parts of the world and then they start using, basically, they popularize that lingo, you would possibly start seeing that as one of the suggested words by Google.

But because there are so many users, so many data points, it cannot store all the data in one place. It cannot perform global update at every time. So it maybe selects 60 to 70% of the users at a time. just computes a gradient on those data points, update the global weights, and then pushes the new set of weights and that's how it works, right? So, in some sense the computation, I mean even so the computation of at least the gradients that may happen locally in a distributed sense, and it gets translated and then gets the weights get updated in a centralized fashion. So this is not really if not fully distributed in the sense because there is centralized computing and distributed like there is centralized communication and distributed computing.

We can also think of a scenario for instance so some of you are from maybe electrical engineering background. So we have this problem called economic power dispatch. let me write it economic power dispatch and in fact, this is one of the problems that we are going to be looking at later when we towards the second half of the course. So, you have a network of generators and loads. So, think of it this way.

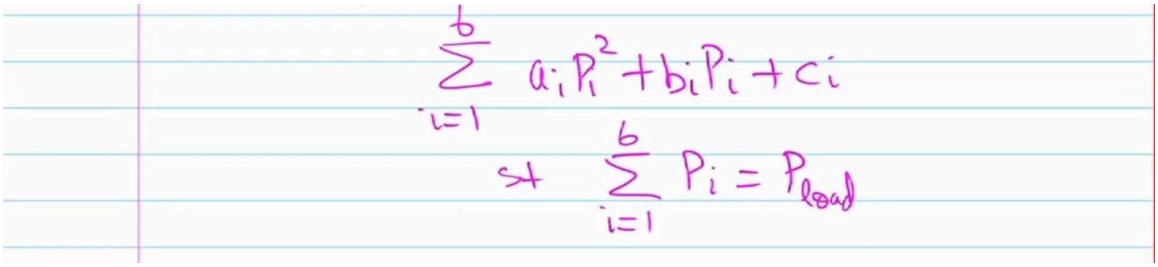
So, let me just number them in a random order. Suppose this is the network that you have, right? So generator 1 or the substation 1 or the bus 1 is connected to bus 2, 2 is connected to 4, 2 is also connected to 6, 6 is connected to 3, and so on, right? So these edges, they kind of indicate the connectivities between any two substations. Now this has gotten like I mean this particular node has got a generator. Let's say the cost of generation, let's say this particular node ends up generating a total power of P1, okay? And the cost of generation, it can be a quadratic cost. which is of this form a1P1 square plus b1P1 plus c1 something of this form right.



Likewise, generator 4 would have its own generation cost which would be proportional to P4 ok. So, this entire network. So basically, if you look at individual generators these generators may be owned by different companies. I mean, they can be a centralized aggregator, but these centralized independent system operator, but these generators can be owned by different companies and they probably would not want to reveal what their generation coefficients are, right? How powerful their generator is or how economical their generator is, right? So at any time, so let's say you have the total load demand, which is let's say P load. And the goal is if you have six generators in the network, you would want to ensure that P1 plus P2 plus P3 plus P4 plus P5, and P6, this should be equal to the total load demand at any time.

So this is one of the objectives that you want to meet the total net load demand. But that's not the only objective right. From a global goal would be that you want to minimize the total cost of generation. So generator should not be greedily producing, let's say I mean this P1, it is upper bounded by some maximum generation capacity P1 max. So it would try to produce basically all the power required and get as much money as possible or generate as much revenue as possible.

But this would not be a social goal. This would be a greedy goal in some sense. So the objective is from the point of view of a centralized or a centralized aggregator or a social aggregator, you would want to minimize this particular cost. Subject to.



Is this clear? I mean there may be some constrained version of this problem.

So for instance you can also have that Pi is less than equal to some Pi max that a generator cannot generate more than a certain value and it cannot generate anything lesser than Pi min. In most cases Pi min you can assume it to be 0. So the generation values are going to be between 0 and Pi max. Now this is a simple quadratic optimization, constrained quadratic optimization and easy to solve. So the challenge lies when these ais, bis, and cis are not known globally.

For instance, generator 1, everyone wants to solve this common problem but without having to share their cost coefficients ai's, bi's, and ci's. And how can we do that, right? So that's the question. So if I do not know about the coefficients ai, bi, and ci, I am allowed to exchange certain information with my immediate neighbors. So generator 1, for instance, can exchange certain information with generator 2 so that it gets a sense of how much it is generating and what can be its likely cost coefficients without having to estimate it. And likewise, it will also be receiving some information from generator 2.

And these edges, they may also indicate geographical sort of connectivity, right? So maybe generator two is located close to generator one. So it's easier to exchange information, right? with a generator which is somewhat nearby located whereas maybe generator 6 which is far off from generator 1, it may be very difficult to communicate any kind of information with that. So by exchanging a certain amount of information and by performing, so there is like you can imagine there is no centralized aggregator right. Now we are not trying to solve this problem in a centralized fashion. We are not, we don't have the access we cannot access ai, bi, and ci for all the generators.

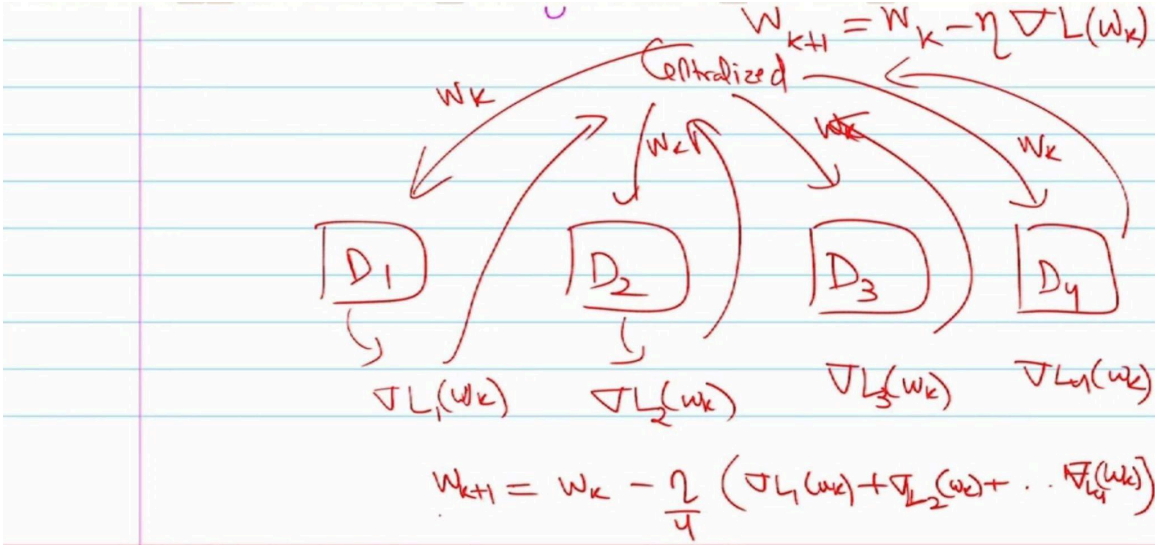
So we cannot solve this problem in a centralized fashion. We can only solve it locally. So everyone is doing some local optimization, but at the same time, they're exchanging certain information with their neighbors and that helps them solve this global problem. So this level of, basically local optimization and then local communication. It is I mean pretty much in the purview of what we call distributed optimization or decentralized optimization.

Sometimes it's called decentralized optimization in certain texts because I mean distributed optimization in most cases would also mean decentralized optimization. But

sometimes people want to differentiate between the computing part and the communication part. So the computation can still be local, but the communication can be with a centralized server. So we won't call it decentralized. But in this case, the communication is also going to happen locally with the nearest neighbors and the computation is also happening at the local level.

So you can also call it decentralized optimization. So sometimes people also call it decentralized optimization, but If you look at the earlier text at least in the control community this would be largely referred to as distributed optimization. It is distributed because information is distributed across different agents. So, every generator is acting as an agent which is trying to solve the problem locally, solve the global problem locally by having its own estimate of how much everyone is producing, and then through exchange of some information it would try and get to the optimal values of optimal solution of this particular problem. Is this clear? So, any questions so far on what distributed optimization is? So, what is a federated learning? Something that yeah.

right so yeah more or less so so the idea with federated learning is let's say you have multiple servers or maybe data centers or multiple devices right let's call it D1 so what is the popular most popularly used algorithm for updating weights of a neural network gradient descent right So in gradient descent typically what do we do weights at iteration k plus 1 is wk minus step size times the gradient of your loss function computed at wk. So the loss function for this particular server would be dependent on the data that this particular server has right. For this server, it would be again dependent on the data that this particular server has. So what may happen is maybe there is a centralized aggregator. centralized server that tells everyone what wk's are at kth iteration okay.



So, then everyone ends up computing the gradient on their own data. So, that would be let's call it gradient L1 wk, why gradient L1 because even though the loss function the mathematical form of the loss function may still be the same the data points are different right. So, everyone would be having a different value of the gradients. So think of it as different batches when you do. And what one can potentially do is one can update the weights.

So this information is then maybe sort of relayed to the centralized server. So this is the typical centralized communication architecture. So these gradient informations are communicated to the centralized server. Whatever information it's receiving, it's simply going to add those. maybe take an average of those wk gradient L2 wk and so on right gradient L4 wk it's going to add those and going to have a centralized update like this and then this new set of weights will then be relate back to these agents right.

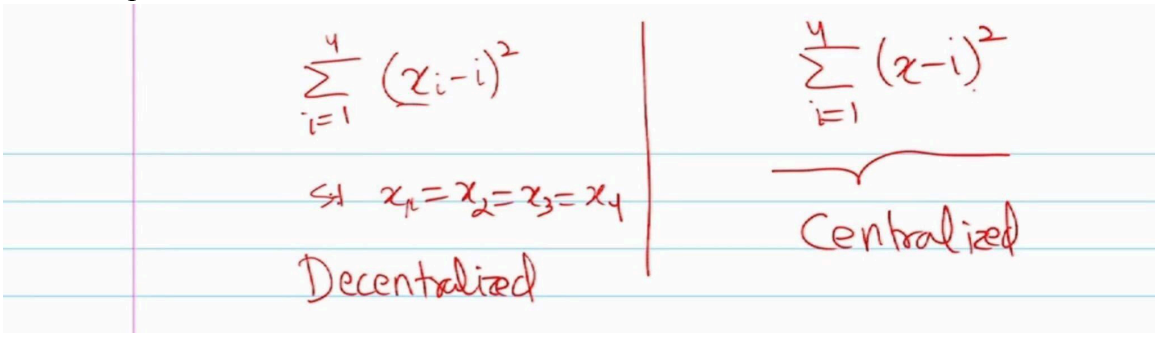
So the gradient computation is happening at a local level, but then there is centralized communication protocol. So in federated learning, what happens is like if you have, I mean, in this case, we just have about four agents or four servers. You can have thousands of such agents, right? And if you were to like wait for everyone's data to arrive, I think it's going, so basically it's going to take a lot of time, right? So instead what is done is, I mean you assume that the data distribution is more or less i.i.d. between agents. So at a time, you are only going to be extracting data from let's say 50 or 60% of the users and going to perform an update based on the data, like based on the gradients from those 50 to 60% of users, and not wait for everyone's data to arrive. And that is what federated learning is. So typical federated learning is somewhat closer to a centralized kind of optimization. not so much with decentralized or the distributed optimization. But then the focus for this course will be more towards distributed or decentralized optimization because we are also kind of looking into issues like privacy for instance where I do not want to share any amount of data that I have with any centralized entity.

So the other aspect of distributed optimization is limited required. So in this case, for instance, there is a single point of failure, right? So if centralized entity sort of fails, there is no way to update the weights anymore. In this case, for instance, if let's say one of the nodes fail, when there's still a possibility that you can still try to maybe when you have enough connectivity with other networks, other other agents in the network, and you can still try to solve this problem as I mean, to optimality as much as possible. So there is also some robustness associated with the distributed or decentralized optimization because there is no single point of failure in general. In centralized optimization, there's a single point of failure and the communication bandwidth requirement for the centralized entity is pretty large.

Like if you have lots of users, then you would have as much communication bandwidth requirement for the centralized entity. With distributed optimization, communication bandwidth requirement is related to the number of agents that it is connected to. So that is also another advantage of distributed optimization. You are going to be communicating locally with your neighbors, only with your neighbors. So, in this case, for instance, I mean it is just going to exchange information with just two neighbors.

So, it does not have like I mean the connectivity, I mean it does not have to have bandwidth 6 times the individual bandwidth of the agents. So, that is another advantage of distributed optimization. Is this clear? And why is communication important in general in distributed optimization? Let's say I am trying to solve this particular problem.

Subject to x1 equal to x2 equal to x3 equal to x4. So this is same as saying that I want to solve this problem.

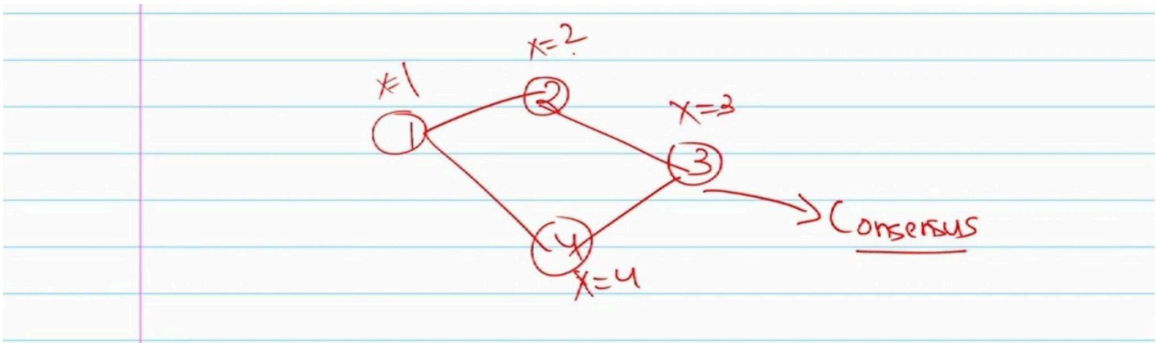


This is the centralized version. This is a distributed or the decentralized version where everyone just knows about its own xi. But they want to ensure that towards the end everyone gets to the same value, which is what the centralized optimization problem would get to. Is this clear? Now if everyone thinks greedily, so for the first agent what is the optimal solution if they want to minimize let's say? So, for the first agent, so what is the objective for the first agent x1 minus 1 whole square right? So, the optimal solution for the first agent is simply 1 right. It can minimize this when x1 star happens to be 1.

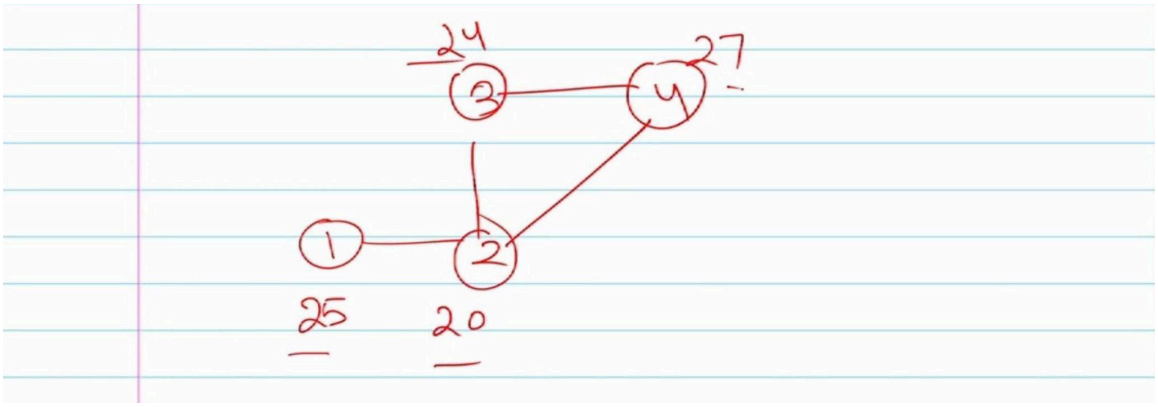
ok. Likewise for the second agent x2 minus 2 whole square. So, x2 star turns out to be 2. So, their local optimal solutions happen to be different from the global optimal solutions which in this case happens to be I think so 1, 2, 3, 4.

So, you have 2.5 as the optimal solution ok. So, if you try to solve this in a global fashion 2.5 would be your optimal solution But then everyone's local optimal solution is different from the global optimal solution. And the question is how can you, I mean basically if you have let's say, if you try to solve it completely locally without having any communication with your neighbors, you are never going to arrive at the optimal solution. So you need some level of connectivity between the agents so as to be able to arrive at the optimal solution for the centralized problem. ok, and that's when the role of the network or the graph plays an importance.

So, let's say you have a graph like this and depending on the connectivity. So, 1 can exchange information with 2 and 4, 2 can exchange with 1 and 3 and so on. So, every time let's say 1 has a certain value, let's say it says my guess is that the optimal solution is going to be at 1. x2 would say no my guess is their optimal solution would be at 2 and they would try and come to in some sense a consensus.



So, they will try and arrive at a consensus. So, that not only they are at they are solving the global problem, but their optimal solutions or the estimates of the optimal solutions they happen to be the same number right? Is this clear? So, what can be another application of this kind of problem? Let us say you in this I mean in this room we have 4 ACs installed right and every AC every air conditioner has a set point that it needs to work with and the every air conditioner has got its own temperature sensor. So, let us look at this problem.



So, let us say this is the connectivity that and the temperatures are 25,20,24 and 27. So, these are the temperature measurements by the 4 air conditioners right? So, 1 can exchange information with 2, 2 can exchange information with 3, 3 can exchange with 4 and so on. So, you can assume that the temperature sensors are somewhat faulty ok.

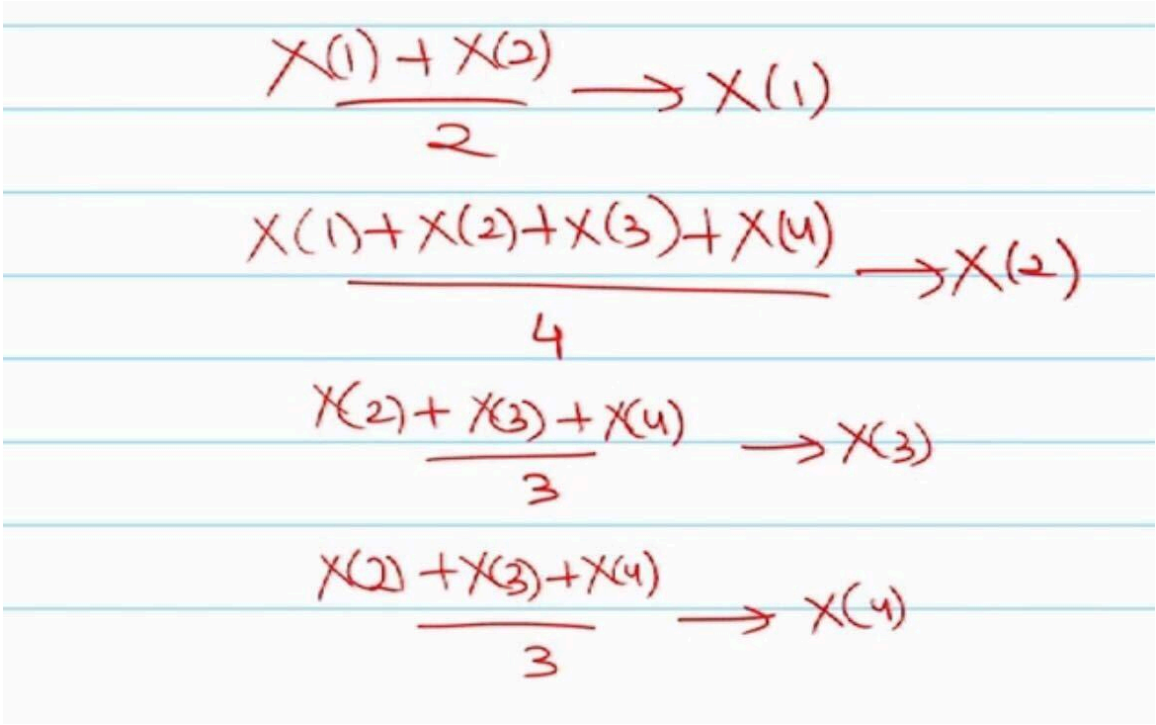
So, and you do not know the right temperature. So, let us say you want to estimate the temperature based on like individual sensor measurement. What would be a good estimate based on the 4 measurements that you have seen? One way to like let us say if the noise is 0 mean, then you just take the mean of all the temperatures and that would be a good estimate right? So, the mean of these numbers is 24 ok. But then you have a bird eye view of all the temperatures right. So, you can see that the temperatures are 25, 20, 24 and 27 and I can simply compute the mean and this becomes a centralized problem. I want to be able to compute this mean, but without having like so for instance 1 does not know what 2, 3, and 4 like 1 does not know what 3 and 4 has right.

1 maybe can exchange information with 2, 2 can exchange information with 3, 3 can with 4, and 4 can exchange with 2. Suppose you were to design an algorithm that basically estimates this using local sort of communication, what would that algorithm look like? So, what would 1 try to do? So, 1 has its own estimate which is 25. Let us say 1's current estimate is x1 and 2's current estimate is x2. So, how would 1 update its value based on what it sees from 2? a very naive algorithm that you can think of. So, everyone wants to arrive at a common value right which may be in this case happens to be the temperature of like the mean temperature value.

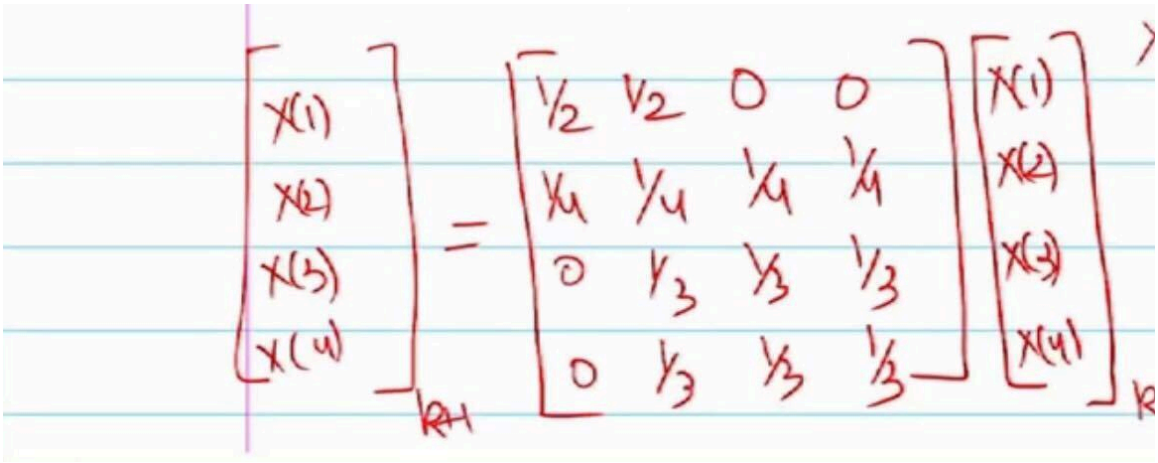
So, but then 1 cannot see what 3 and 4 have measured, it can only have information about 2. 2 can see what 1 and 3 have measured, 3 can see what 4 and 2 have measured and so on right. Now from one's perspective, how does it arrive, like how does it know what is the average or the mean value based on what it sees from 2? So, it has to, like it

will start with maybe some its own estimate of what the average temperature is, and then it would, I mean you would try and converge to the mean value, right? Every temperature sensor or every measurement unit would try and converge to the mean value. So, the question is what would be the update rule for 1 look like? So, something like this right.

So, maybe take x1 plus x2 by 2 and assign it to x1. What about 2? So, all the neighbors. So, for 2 it would turn out to be something like x1 plus x2 plus x3 plus x4 by 4 and that becomes your x2 ok. 3 would be x2 plus x3 plus x4 by 3 and that gets assigned to x3 and then 4 would be 4 is connected to 2 3 and 4. So, the same thing here x2 plus x3 plus x4 by 3 Is this clear? So everyone is just averaging its own belief and the belief of its neighboring senses.



Right? So believe our estimates are the mean. So I'm going to use them interchangeably, but they mean the same thing. So everyone has its own estimate of what the other device is measuring or basically what what what is it measuring and what other device is measuring and they're just going to average those and that becomes the update or the estimate for the global average value. Right. So let's let's try and see what this comes down to. So if I write this in an algorithmic fashion, So, at iteration k plus 1, x1, x2, x3 and x4, this is at iteration k x1, x2, x3, and x4 and for 1, so this would be half 1 half 0 and 0.



right. For 2 it would be one-fourth all the way, for 3 it would be 0, one-third, one-third, one-third and for 4 again it is the same thing right? So, everyone would start with let us say in the beginning they have their own measurements 25, 20, 24, and 27. So, everyone would start with those measurements. So, that would be x1, x2, x3, x4 at iteration 0 and then they would be updating the estimates based on this. And the hope is that it is converges to their average value which is 24 right? This is the most usual thing to do right or most naive thing to do when you want to arrive at the consensus value or when you want to establish a consensus between different devices.

So, let us see how this particular. So, by the way if I look at this particular matrix. So, this matrix has 0 entries whenever there is not whenever there is not an edge between 2 agents right? Otherwise, it has certain values. So, in this case, this edge from 1 to 2 it has a value 1 to itself it has a value of half, and then 1 to 2 it again has a value of half right. So, we can relate this to something called adjacency matrix of a graph and that is something that again that we are going to look into in the latter half of the course. So, your algorithm is xk plus 1 in the vector form is A times xk where A happens to be the adjacency matrix and it has this particular form.

Is this clear? So, let us see what this is like if you run this algorithm and what this converges to. So, these are the 4 temperature measurements So, 25, 20 and 24 and 27. So, these are the individual temperature measurements right and A1 is our adjacency matrix. So, which was one half, one half, 0, 0 is what we started with then all one fourth then 0, one third, one third and then the same thing right and then we run this for 20 iterations and let us see what and what it produces. So, all we are doing is we are like our new x is basically A times x.

```
[1] import numpy as np
    from matplotlib import pyplot as plt

    x0 = np.array([25., 20., 24., 27.])
    A1 = np.array([[0.5, 0.5, 0., 0.], [0.25, 0.25, 0.25, 0.25], [0., 1/3, 1/3, 1/3], [0., 1/3, 1/3, 1/3]])

    n_iter = 20

    x    = x0
    alg1 = []
    alg1.append(x)
    for n in range(n_iter):
      x = np.dot(A1,x)
      alg1.append(x)
```

xk plus 1 is a times xk and that is a new value that we are again going to be updating. So, it is a simple update, it is a local update because one for instance in this case one just uses information from itself and its neighbor, and likewise every agent is using information from itself and its neighbor. So, this is fine and if you look at what this algorithm produces after 20 iterations.

So, it has converged. So, that is a good thing. It has arrived at a consensus. So, all the sensors are in consensus, but instead of basically converging to 24 which is what was desired which it has I mean converged to a number which is maybe 23.58 something right? Every sensor has converged to 23.58 even though we wanted to arrive at a value which is 24. we wanted to arrive at an average consensus and not so we didn't just want to ensure that there was there is consensus among these devices we want to ensure that there is consensus there is average consensus between the devices so that means the value has converged to the average of all these sensors right and that hasn't happened is this clear? so Now, I am going to choose slightly different form of adjacency matrix.

```
[3]  array([23.58310312, 23.58330371, 23.58342982, 23.58342982]),
     array([23.58320341, 23.58331662, 23.58338778, 23.58338778]),
     array([23.58326002, 23.5833239 , 23.58336406, 23.58336406]),
     array([23.58329196, 23.58332801, 23.58335067, 23.58335067]),
     array([23.58330998, 23.58333033, 23.58334312, 23.58334312]),
     array([23.58332016, 23.58333164, 23.58333886, 23.58333886])]

x0 = np.array([25., 20., 24., 27.])
A2 = np.array([[3/4, 1/4, 0., 0.], [0.25, 0.25, 0.25, 0.25], [0., 1/4, 5/12, 1/3], [0., 1/4, 1/3, 5/12]])

n_iter = 20

x    = x0
alg2 = []
alg2.append(x)
for n in range(n_iter):
  x = np.dot(A2,x)
  alg2.append(x)
```

The connectivity still remains the same, the graph connectivity. So, again 1 is not going to be exchanging any information with 3 and 4. But then instead of using half and half, I am giving more emphasis like in this case 3 fourth to like to 1 from the point perspective of 1. It is going to assign a larger weight which is 3 fourth to itself and a smaller weight to its neighbor which is 1 fourth. So, this time it still adds up to 1 right, 3 4th, 1 4th still adds up to 1, all 1 4th for the second one, then you have 1 4th, 5 12th, and 1 3rd which again adds to 1, but then we are using a different weighting scheme now. All the communication is still going to be local, there is no like for instance 3 was not connected to 1.

So, there is not going to be any update like when updating 3, there is not going to be any information exchange from 3 to 1, and likewise for the 4th. Is this clear? So, let us run this and with the same initialization 25, 20, 24, and 27 you can see now it has everyone has converged to the same number.

So, 23.9989 is roughly 24. So, 24, 24 and all. So, all the sensors or all the devices have converged to the same estimate, and which this estimate also happens to be the average consensus value that we wanted the devices to converge to. So it really depends on what kind of adjacency matrix that we end up choosing that would guarantee not just consensus, but average consensus. And that is again something that we are going to look into in the later half of the course. Is this clear? So much of distributed optimization is not

just about optimization. Optimization is something that you would have studied in other courses, convex optimization, integer optimization and so on.

When you do not have the centralized or the global information available, how do you still guarantee convergence for the global optimal solution when you are exchanging information locally with your neighbors? So, this is what this particular course is all about. The other aspect of this course and that is more related to the recent developments in this area is actually on accelerated optimization. So remember the problem on economic dispatch where we talked about meeting the load demand at each time point? Now that we know that the load fluctuates with our usage rate. So every 10 to 15 minutes, maybe it gets a little hot.

So someone would turn on the AC. So there's a sudden load demand rate or maybe someone like using like someone has a swimming pool at their home and they want to heat up the water. So there will be a sudden increase in load demand and so on. So it varies with time. Now the question is like if you end up spending a lot of time in solving the optimization problem by the time you try to meet that particular load demand the load the demand point has already shifted and whatever you have produced may not be valid for the future time points right.

So you want to be able to solve the optimization problem rather quickly. So you want to have faster convergence guarantees and that is where the first half of the course which is going to be focused more towards accelerated optimization algorithms. So optimization algorithm is something that you have already seen but we are going to look at ways through which we can accelerate the process of arriving at the optimal solution. So we would still be starting with the single agent centralized optimization and that would be pretty much the first half of the course. In the second half is when we are going to move to distributed optimization where we would also read some basics on graph theory and then move to designing algorithms which basically solve problems like economic dispatch. Is this clear? Thank you very much.