Neural Network and Applications
Prof. S. Sengupta
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 36
Cooperative and Adaptive Processes in SOM

Today's lecture is on the, Cooperative and Adaptive Processes in Self Organizing Maps. In the last class, we were discussing about some of the essentially steps that one has to follow in the self-organizing maps and the first one of that we discussed in the last class itself, that is the competition. So, at first we have to see, that there is a competition among the neuron which are there in the output layer. And then, following the competition we have to go through a competitive cooperative and then the adaptive processes.

Now, we know that what is meant by the competition, that we will be having m as the number of inputs and then, there will be l outputs which will be competing amongst themselves, meaning that we will be computing w j transpose x vector. For all j's, where j is equal to 1 to l where l is the number of outputs and out of all these j's that means, to say out of these l number of neurons in the output layer, there will be a competition.

And, the winner will be the one which is having maximum value of w transpose w j transpose x vector, or the minimum of the Euclidean distance between the w j vector and the input x vector. And the index corresponding to the neuron, which is having the minimum Euclidean distance with respect to the input pattern, that index will correspond to the index of the winning neuron.

And the corresponding weight vector that we are having is the synaptic weight corresponding to the winning neuron. And in fact, in response to this input vector the winning neuron will be required to adjust it is weight. The question is that should the weight adjustment be done only by the winning neuron, or should it also be done by the neurons, which are in the close neighborhood of it.

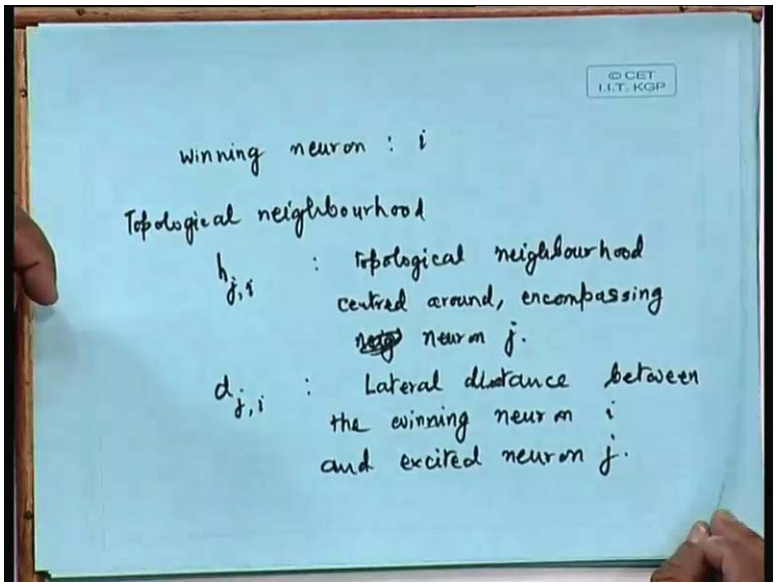
Now, the answer lies that, in addition to the winning neurons even all the neurons, which are lying close to it in the close neighborhood of it should also change their weights. So,

there should be some mechanism of cooperation between the winning neuron and the neurons, which are there in it is surroundings.

So, that is the cooperative processes that are we referring to, so the competition among the neuron is always followed by, what is known as the cooperation meaning that, when a neuron is fired then it also excites the neurons, which are in its neighborhood. And it is intuitively obvious that, if we go farther away from the winning neuron. Then it is neighborhood function should gradually decrease, the neighborhood function should be having maximum at the position of the winning neuron.

When, the distance from the winning neuron to the neuron, where we want to determine the neighborhood function is equal to 0 that is where we are going to find the neighborhood function as the maximum. And as we go farther and farther away in the limit, if we take d i j to be infinity there the neighborhood function should ultimately decay down to 0. So, the neighborhood function should be a function that, monotonically decreases with the distance from the winning neuron. So, we have to define a neighborhood and how do we definite that?

(Refer Slide Time: 05:43)



So, let us say that the winning neuron is the neuron i, so say the winning neuron is having an index i and, we want to find out a neighborhood around this winning neuron.

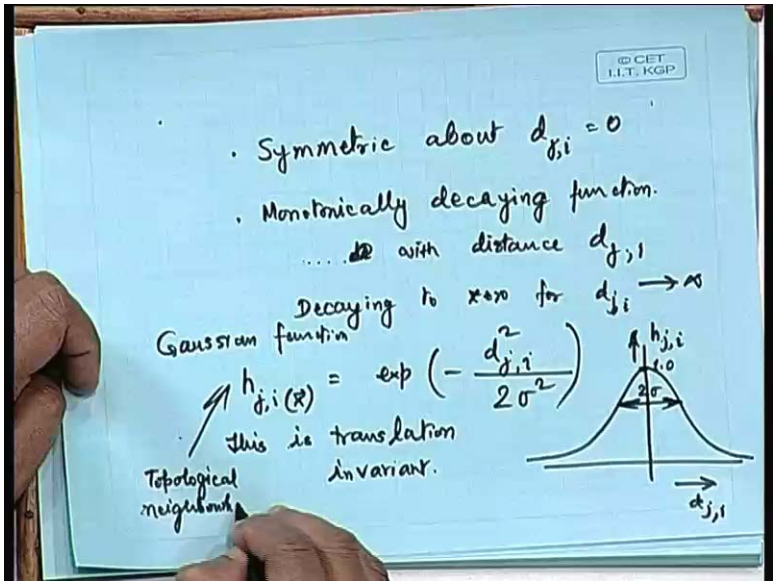
So, we define a topological neighborhood, s o by topological neighborhood we define a quantity let us say, which is h of j comma i, which is the topological neighborhood centered around i. So, this is topological neighborhood centered topological neighborhood centered around i and it is encompassing all the cooperative neurons, including j. So, we are measuring the topological neighborhood at the point j, so centered around i encompassing neuron j, where we are measuring it encompassing neuron j.

So, it is a cooperation between the neuron j and the winning neuron i. Now, naturally this topological neighborhood function should decrease, with the lateral distance that we are going to have between the winning neuron and the neuron j. So, the distance between the i and j the lateral distance.

So, let us denote that lateral distance by d j comma i, so this is lateral distance between the winning neuron i and the excited neuron j. Now, the neuron j we are calling as the excited neuron, so excited neuron is the one, which is excited as an effect of this winning neuron. So, even the weights of these excited neurons will have to be adjusted and to what extent, that will be decided by the neighborhood.

So, we define a neighborhood h of j i and we also define the lateral distance between the winning neuron and the excited neuron j, so that is given by, d j i. Now, this topological neighborhood h of j i this function should satisfy two properties.

(Refer Slide Time: 08:40)



One is that, it should be symmetric about d i j is equal to 0 is not it, do not you feel that it should be that way. Obviously, because no matter that wherever, I move from this if I make d i j equal to minus 1, minus 2 like that, and if I make d i j is equal to plus 1, plus 2 like that, that should not matter to us. It is d k should be uniform in all directions. Whether we make d j i as positive or whether we make d j i as negative, so it should be a symmetric function about d j i is equal to 0.

And the other thing, which I already said that it should be a monotonically decaying function, so it should be a monotonically decaying function with, so it decays with the lateral distance d j i, decaying with the lateral distance, with distance d j i. And it should be, decaying to 0 at d i, d j i tending to infinity, so it decay, so it is decaying to 0 for d j i tending to infinity. So, can we suggest any typical function that, should fulfill this property any common popular function.

Student: Gaussian.

Gaussian, because Gaussian should be monotonically decaying, Gaussian should be symmetric about d j i is equal to 0 and, the Gaussian should be decaying to 0 for d i j for d j i tending to infinity. So, considering a Gaussian neighborhood is quite logical, that should satisfy these properties.

So, a typical choice of this is the Gaussian function, so, that we can express if we choose the Gaussian function, then we can express h of j i as the exponential to the power minus d j i square upon 2 sigma, square where sigma is the width of the Gaussian function. So, basically what we mean to say is that, we will be having an h i, h j i function like this. So h j i will be plotted on the y axis and d j i will be plotted in the x axis.

So, at d j i is equal to 0, what should be the value of h j i out here, that should be equal to 1, so that is the maximum value, so the maximum value is one corresponding to d j i is equal to 0. And it should, decay exponentially, so this is the kind of the curve that you can expect out of it. So, this is a Gaussian curve according to this equation and it is width will be here, sigma with respect to this. So, that this total width is two times sigma that is the sigma of the Gaussian function.

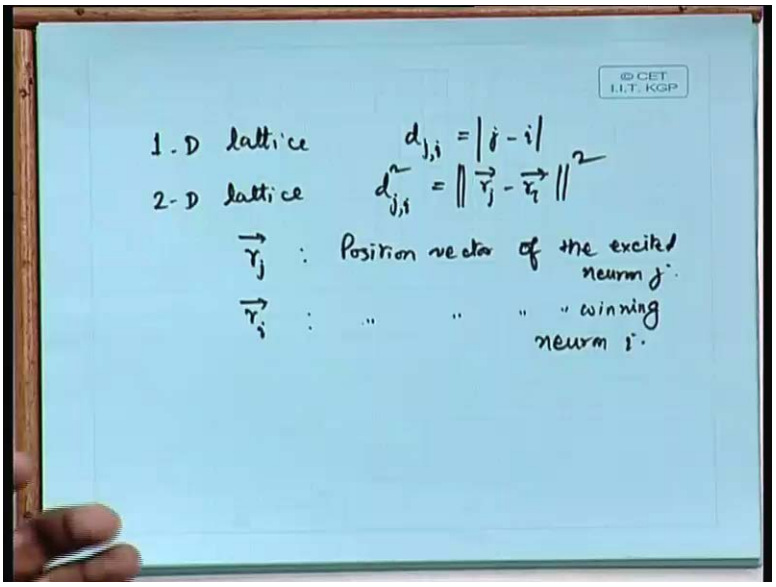
Now, you can notice one thing also, that does this function depend upon the position of the winning neuron. No, no matter whether we take the position of the winning neuron

here, or we take the position of the winning neuron, here, or here, or here, it does not matter. So, it should be translation invariant, so this is translation invariant it does not depend upon, where exactly the winning neuron is located.

Now, h of j i actually, the h of j i that we are getting that is in response to some input vector, because we are feeding an input vector in response to it some input vector x in response to it, there is a winning neuron. And this winning neuron is trying to excite the neurons, in its neighborhood with the increasing distance, the function is decaying out. So, here this h of j i we should write this j i as a function of the x vector, so this is in response to the input x.

So, that is why we normally write it as, h of j i x vector is equal to this, so this is when we are choosing the Gaussian function as the neighborhood function. Now, in this case we have taken the width of the Gaussian to be equal to sigma or 2 sigma. Now, d j i now according to this formula d j i can be positive or negative. So, in the case of one dimensional lattice, if we talk in terms of distance and if express the distance always as a positive quantities.

(Refer Slide Time: 14:42)



Then in case of 1 D lattice, the distance d j i can be expressed as if we take distance to be positive we should express it as, mod of j minus i. And in the case of, 2 D lattice or for that matter even higher dimensional lattice of course. As I mentioned that for the self organizing maps, we do not normally go in for a dimension higher that 2, for the lattice

part of it. So, we may be having m dimensional vector that does not matter, but considering the lattice it is normally not higher than two dimension.

So, there what we do is that we define the position vectors of the neuron j and, the and for the neuron for the winning neuron i. So, let us say that the position vectors are r j and r i, so r j is a position vector of the neuron j of the excited neuron j, and r i is the position vector of the winning neuron i. So, in the case of 2 D lattice, we can write that d j i square is going to be equal to the Euclidean norm of r j minus r i square.

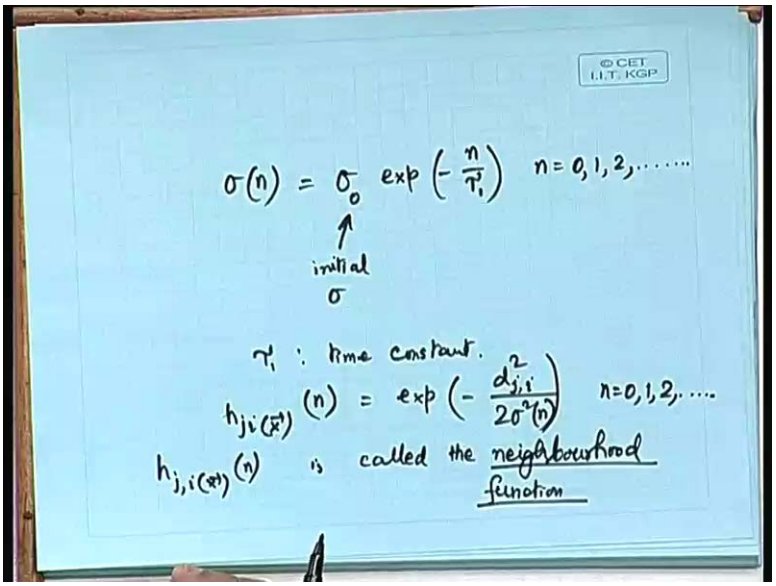
So, this is the d i j that we are going to have, for the case of 1 D and 2 D in fact, if we think of higher dimension then also this relation will hold good. Because, only thing is that in the case of higher dimensional lattice there this r j and r i will not be consisting of just two elements, it will be consisting of multiple elements, depending upon what dimension we choose for the lattice. But as i told you that 2 D lattice is normally good enough.

Now, another unique property of the self organizing map is that, this sigma that we have defined in the Gaussian function, this sigma is not constant with respect to time, by time to say the iterations that take place. So, the very first iteration or at the very beginning of it, we will be calling it as n is equal to 0. And with the iterations increasing we will be calling it as n is equal to 1, 2, 3 etcetera, like that the iteration progresses.

So, as the iteration progresses this sigma is going to decrease in time, the sigma is going to decrease with time meaning that the neighborhood shrinks gradually with time. So, to start with, when we start the organization of this network the self organization classes. When it starts that time this a g i function consequently is quite large, but with iterations progressively it is neighborhood is shrinked and narrowed down.

I mean as it organizes more and more its neighborhood is narrowed, narrowed down and then only the winning neuron and perhaps, a very small neighborhood around it is considered, because if the sigma is made too small. Then you can see that it is effect will be felt only in a very close surrounding, so that is what is normally done that this sigma shrinks with time. And that shrinks according to some time constant, so normally the sigma has a function of iteration number or n.

(Refer Slide Time: 18:56)



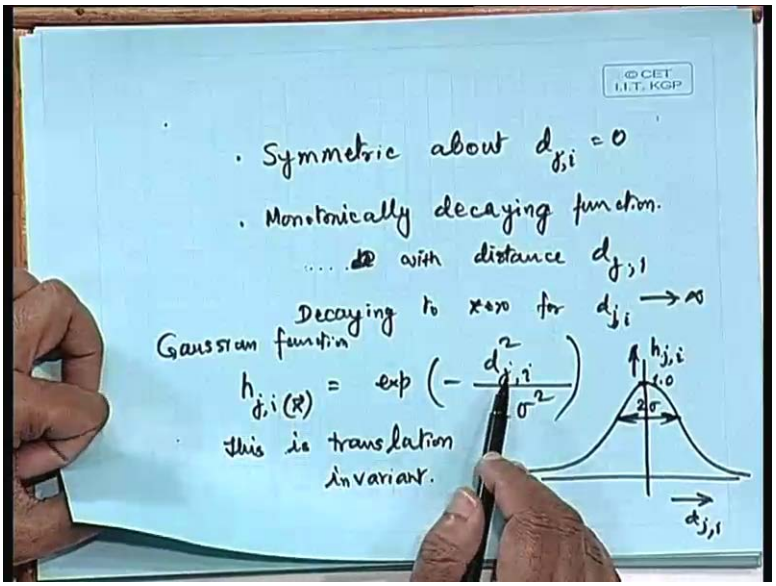
So, sigma as a function of n is expressed as, sigma n equal to sigma 0, where sigma 0 is the initial sigma that means, to say at n is equal to 0. So, it is normally expressed as sigma n equal to sigma 0 exponential to the power minus n by tau 1, where tau 1 is a time constant. So, when n equals to tau 1 then, sigma n decreases to 0.37 of it is maximum value.

So, here the n that is to say the iteration number will be starting with 0, but it can progress from 0 to 1 to etcetera, it can continue with the iteration number. So, that is how the neighborhood will gradually shrink and as a result of this sigma n shrinking with time consequently h of j i, that we are going to write. That is the neighborhood it is to be written with n as an argument.

So, we are going to write it as h j i x vector with n as an argument, which will be equal to exponential to the power its usual definition, is exponential to the power minus d j i square by 2 sigma square. But in order to make it time varying, we have to make it as sigma square n sigma square with parameter n that is how we have to write down. So, we are going to modify the write up as d j i square by 2 sigma square n for n equal to 0 1, 2, etcetera.

And this, h of j i x of n this is called as the neighborhood function is called the neighborhood function.

(Refer Slide Time: 21:30)



Now, I might have already called this h of j i, the fixed h of j i I was calling sometimes loosely I was calling this also as a function, but terminology wise this is called as the topological neighborhood. So, this is the definition of topological neighborhood whereas, when h of j i of x is a function of n, then we are going to call it as neighborhood function. Just let us, be little careful about the terminology this is function whereas, without this n it is just the topological neighborhood.

So, this is the process of cooperation, but the question is that why is the cooperation at all needed the question can come to our mind is that as, if to say that the way this self organizing map algorithm is organized that. Initially, we are going to have a large number of neurons coming in the cooperative process, because there the sigma is quite high, so a large number of neurons will be cooperatively updated.

The reason here, is that we are not updating, so you find here that the neuron itself is not only updated the winning neuron in addition to the winning neuron, we are also updating the neighborhood. The next training pattern, that you are going to feed that may not be exactly the pattern that caused that is corresponding to the pattern for, which the earlier neuron let us say, neuron I had worn for some pattern x.

Let us say and now, a new pattern let us say x 1 is fed and in response to that, i is not the winner, but another let us say neuron k is the winner, which is perhaps close to i. Now, what happens is that now, neuron k being the winner will have the maximum weight

adaptation, but the neuron i, which is close already to it which was the earlier winner that will also have some effect of weight adjustment. So, as a result what happens is that if you are feeding let us say too many patterns, which are close to the is x vector supposing there is a cluster of patterns different from each other, but very close to this x which are being fed one after the other.

So, the winners are sometimes different, but what happens is that everybody is getting some share of weight adaptation. So, ultimately what happens is that the topology of the network will be adjusted according to this cluster, if instead I make the patterns uniformly distributed in the input space. I make the pattern uniformly distributed then even my winning neurons also will be having their weights uniformly distributed.

So, the ultimate weight vectors, which will be associated with the output neurons will correspond to the input distribution and that, is the concept of the topological arrangement. So, let us be very clear about it anybody having doubt about, the topological arrangement that I am talking of, you see this is highly biological motivated you see once you find a winning neuron, you are also exciting the neurons around it.

Because, generally it is found in the human brain also, in the human brain it is found that the certain functions are computed by a particular area, or a particular zone of the brain is computing certain function. So, that is why the excitations are there in the neighborhood area also and by topological arrangement, we want to mean that as you find as you feed patterns in association with the statistical distribution the of the patterns, the weight vectors that are associated with the neurons, they also will be topologically distributed.

So, that is the idea that we are trying to draw may be, that we can talk about one or two examples little later on. So, this is about the cooperative process and next to the cooperative process comes the third one that, is the synaptic weight adaptation process. Now, synaptic weight adaptation process, when we have to think of a synaptic weight adaptation process naturally there, is learning that is associated with it, now what type of learning mechanism are we employing.

We have gone through several learning mechanisms, but for the self organizations the learning mechanism that one commonly employs is the Hebbian learning. Hebbian learning as you remember is that when the presynaptic and the postsynaptic activities are

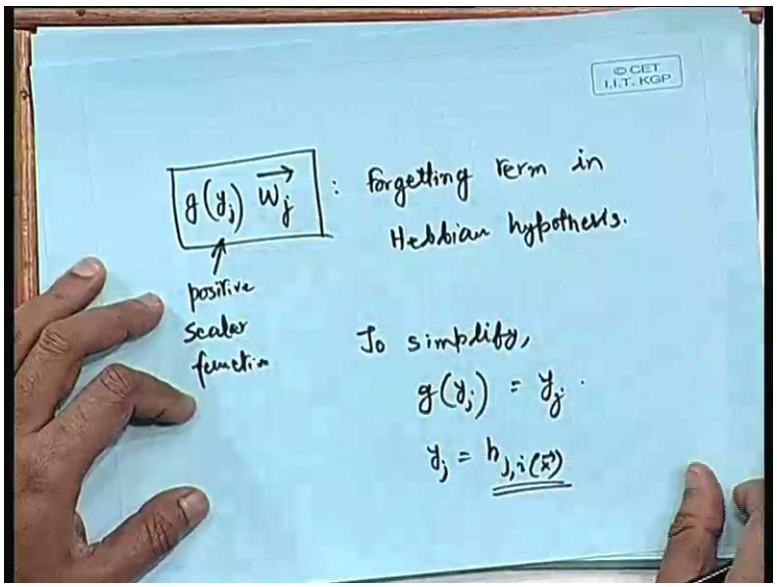
correlated, then the synaptic connection is strengthened and if they are not correlated then the synaptic connection is simply weakened.

Now, Hebbian learning is this updates the weights according to some kind of a positive feedback mechanism, but inherently it has got some limitation that if we continuously feed the same training pattern then there is a possibility that the weights will saturate. So, at some point, the weights will keep on increasing and then, it will reach a saturation where it would not be able to increase the weight any further.

So, we have to prevent this saturation from occurring, so we should not be considering the Hebbian learning mechanism in an unmodified way, we should modify the Hebbian learning mechanism by introducing what is known as a forgetting term. As if to say that Hebbian learning why, it is continuously increasing the weight with every feeding of the same pattern is that, it is sort of over learning, it is learning the same thing again and again. And, it is increasing its strength continuously and you cannot allow this continuous increase to happen.

So, you have to deliberately forget something so that unlimited learning does not take place that is the whole idea. So, to prevent unlimited learning we have to introduce a forgetting term.

(Refer Slide Time: 29:14)



So, what we do is that in the Hebbian hypothesis, we introduce a term which will be given by g y j g as a function of y j now, this is a positive scalar function. And this, function g this g is expressed as a function of y j and what is y j y j is the output, so we take the output neuron j. So, that is the y j and g that, is the positive scalar function is expressed as a function of y j.

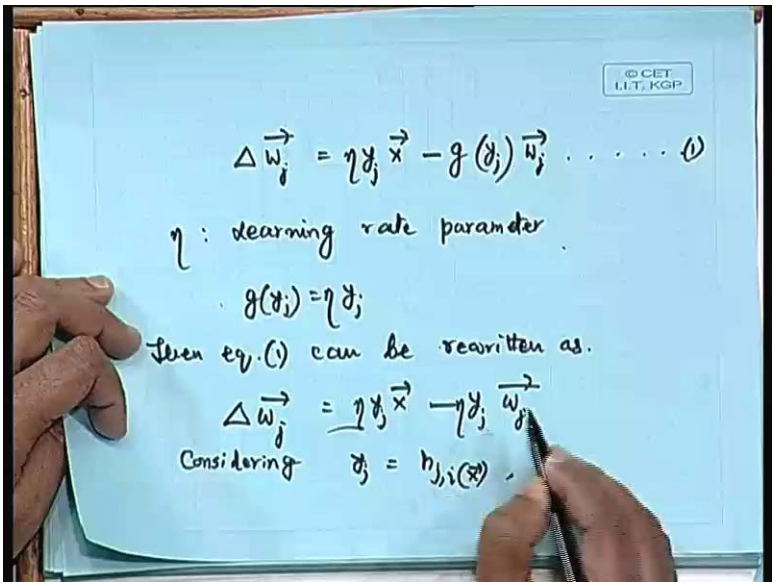
So, g y j times w j vector, so this whole thing that is d g y j w j vector is a will act as a forgetting term in Hebbian hypothesis. So, this is the forgetting term in Hebbian hypothesis, so we have to express this as a function. Now, g is a any general function it does not really say that, whether g will be a linear function or not, but we can take g to be a linear function just to simplifier our job.

So to simplifier our work, we can take g y j to be a linear function of y j, so that in the simplest case we can talk about g y j as equal to y j, exactly equal to y j, so it is a linear, so it is linear in y j. And what is y j, y j is the output neuron j r, now we will have to as I told you that we have to adjust the weight not only for the winning neuron, but also for the excited neurons. So that means, to say that we not only consider the j who is the winner, but also the j's which are excited and close to the winner.

So that means, to say that we must be logical in formulating that, this y j can be expressed as h of j i cannot we, we can express y of j as h of j i x that is to say the topological neighborhood. Because, that itself takes care of this that h of j i is obviously, the maximum when j is the winner and as the distance from the as the lateral distance from the winning neuron is progressively increased, the y j also will progressively decrease.

So, the as an output function we can naturally consider this h of j i anyway. So, let us first of all write down the Hebbian hypothesis, itself by introducing this forgetting term.

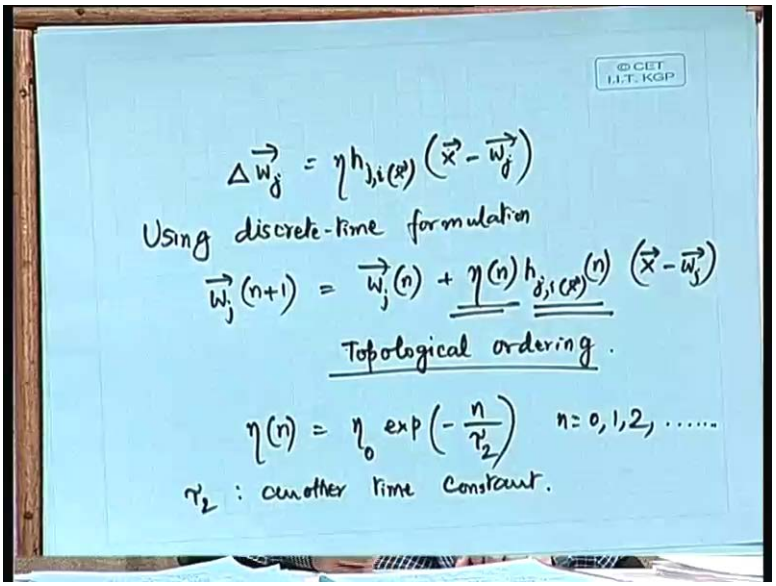
(Refer Slide Time: 32:34)



So, the normal way of writing the Hebbian hypothesis as you know is that, delta w j is equal to yes Hebbian is known to us eta yes, y j x vector, so that is the normal Hebbian term, but we are also introducing a forgetting terms here g y j w j. So, this will be used as a minus to it, so this will be minus g y j, w j vector and as you know that, here once again this eta is the learning rate parameter of the algorithm x vector is the input y j is the output.

So we are considering this, now if we take g y j is equal to y j as we suggested just sometimes back that, for simplicity if we take g y j is equal to y j. Then, we can rewrite this equation called this as equation 1, then equation 1 can be rewritten as delta w j vector is equal to eta y j x vector minus y j, w j vector and if we take this y of j to be equal to h of j i. In order to include the winner as well as, the excited neurons all the neurons in the topological neighborhood, then so considering y j is equal to h of j i as a function of x.

(Refer Slide Time: 35:05)



We can write it down as, delta w j i delta w j is equal to eta h of j i h of j comma i as a function of x and this, should be multiplied by ((Refer Time: 35:24)) what remains, because both these terms contain h of j i. So, it can be taken outside the parenthesis, so what remains inside the bracket is x vector minus w j vector. So, that basically proves that we are going to adjust this w j such that it should move closer to x vector.

So, in the limit means, when it learns w j vector will align itself with the x vector, because in the limit when the network has learned out of this pattern, then delta w j vector should be equal to 0.

Student: Learning rate is not multiply with more than.

Learning rate is not, one minute, one minute. So, yeah that is, that is right.

So, we should not take this g y i to be exactly equal to y j, let us introduce a learning rate here also. So, g y j we can take it to be equal to eta y j, so that this function also does not take the y j directly it takes it multiplied by this. So, that then we have this is equal to eta y j x vector minus eta y j w j vector. So, then we can take the eta term out eta h j term can be taken out and then, what remains is this x vector minus w j vector and this using the discrete time formulation.

So, using discrete time formulation we can write it down as the same thing can be rewritten as w j for the step n plus 1 is equal to w j for n plus this term eta. And now we

can write it as in order to make it more general. We can express eta as a function of n that means, to say what that instead of taking eta to be same for all the iterations, we may like to vary eta with n.

So, we introduce eta n and the rest of the terms could be h of j i x vector and as an, argument of h also we should write n simply, because as we had also seen last time that this h of j i shrinks with time. So, that is why in general, we should be writing this h of j i x as a function of n and then, we can write down the rest of the term that is x vector minus w j vector.

So, you can see that since, the ultimate tendency will be to align the w j vector with this x vector and this, will not only be done for the winning it will also be done, for all the loser neurons also. Only thing is that for them what happens is that, for the loser means which are losers, but are excited in the neighborhood for them this h of j i term will be dropping down. So they will be learning slower, but they will also learn it is not the winner alone.

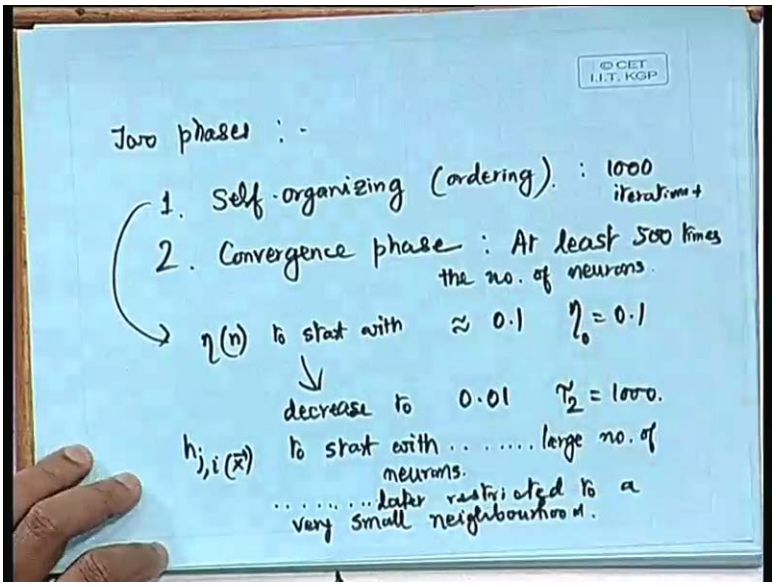
Now, the topological ordering aspect, if it is not, if it is still not clear to your mind we will be telling some examples later on, but ultimately this is the formula, which will ultimately lead to the topological ordering, because that is the greatest strength of the self organizing map. Now, here we have got two heuristics to watch one is this, eta n the choice of the eta and the other is h of j i of n that is also to be heuristically chosen.

Now, h of j i n is decay function we have already defined, that it will be decaying in accordance with the decay of the sigma, so since the sigma shrinks down that is why h of j i also shrinks down. And likewise, eta n to start with the leaning rate should be high for a quick topological ordering, but once the topological ordering is there and we are looking for a convergence there the eta n should be made smaller.

So, that is why eta n should also decrease with time, so what we have to do is to use some kind of an expression like this that eta n is equal to eta 0, where eta 0 is the starting learning rate. So, it should be eta 0, exponential to the power minus n by some time constant since, we used the last time constant we called as tau 1. That is to say the time constant of decaying of sigma this one, we can call as tau 2 the time constant of the decaying of the eta.

So, here n is equal to 0, 1, 2 etcetera, etcetera, where tau 2 is another time constant. So, that is the mechanism of the updating, which we have to follow. Now there are basically two stages of the adaptation this third process which, we have just now described that is to say the adaptive process, this adaptive process actually is divided into two sub steps.

(Refer Slide Time: 42:38)



There, are two phases rather, so the two phases of adaptive process are one is the self organizing or what is called as the ordering phase, so the first one is the self organizing or the ordering phase. And the second one is known as, the convergence phase, now during the self organizing phase what happens, is that the topology is arranged and during the convergence phase all those positions, which are decided, which are obtained by there topological ordering, they are fine tuned during the convergence phase in order to reduce the error as much as possible.

So, it is the self organizing phase, which takes relatively lesser number of iterations that is the topological ordering is seen to be quite fast as compared to the actual convergence phase convergence phase takes a lot of iterations, because convergence phase is normally slow. And one has to do it with much smaller values of eta in order to have the best learning, now the first phase that is to say during the self organizing phase, one has to choose eta n to start with, the eta n has to be chosen quite large.

So, eta n should be, so to start with eta n to start with should be, of the order of should be of the order of point one that is some typical values. And then, it should decrease to

decrease to values like 0.01 and, we have to so here, we take eta 0 to be equal to 0.1 some typically, and it decreases to 0.01 and the time constant tau 2, that we use that is to say the leaning rates time constant decayed. That is, of the order of say 1000's. And then the self organizing phase normally takes around 1000 iterations, plus more then 1000 iterations are normally needed for a typical self organizing map to work.

And another thing that is, done in the self organizing or the topological ordering phase is that the h of j I, this h of j i x to start with should be very large to start with should include a large number of neurons. It should include a large number of neurons and when, and gradually it should be this h of j i it shrinks the neighborhood shrinks, and it is thereafter restricted the only one or two neuron surrounding the winning neuron.

Or may be the winning itself, we may like to turn off all the excited neurons around it and, we may like to only update the winning neuron. So, later restricted to a very small zone later restricted to a very small neighborhood may be even the winning neuron itself. So, that is the self organizing phase and during the next phase that is, in the convergence phase there we try to go through, many more number of iterations as compared to the self organizing.

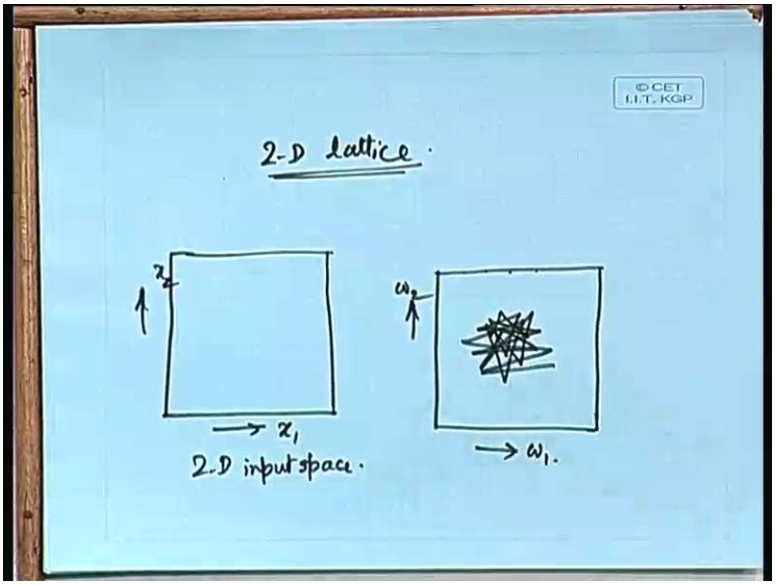
In fact, in the convergence it is generally seen that, we have atleast we have to train this convergence phase for atleast, 500 times the number of neurons in the network. So, atleast 500 times the number of neurons, so you can imagine that, if you have a 4 by 4 topology, let us say four by 4 neuron topology meaning that 16 neurons is you have. Then you have, to go through 16 times 500, which means to say 8000, it leads to 8000 number of iterations.

But, 16 is a small number for many practical applications, we require a much larger size of topology or the lattice size is normally, quite large for typical problems. So, where as you increase the topology size the convergence phase will require more and more number of iterations. Now, there as you can well understand that for the convergence phase the eta n starts with a value of 0.01.

That, order and generally it is maintained there, because we do not normally decrease the eta n in the convergence phase, but already the value is quite low, but h of j i if to start with we have got a small neighborhood. Then, as the convergence phase progresses we may like to narrow it down further may be restricted to the winning neuron itself. So, this

is what people do from the practical consideration, so now coming to the topology organization in order to explain the concept, let us say that we have got a two dimensional lattice.

(Refer Slide Time: 49:32)



Let us say, 2 D lattice we consider and we have got, the input space also supposing the input space also is a 2 D input space. So, we are considering 2 D input space, so they the variables are x 1 and x 2 only two dimensional. So, there will be a large number of patterns will be existing in this space, so we define the limits of this x 1 and x 2. So, supposing this is the zone from, which we have to choose the inputs and another thing, that we have decided is that it is a 2 D lattice.

2 D lattice means, what that we are spacing the neurons at some uniform distance apart the distance of separation between the neurons, that we considered to be absolutely uniform in nature. And here, what we have to do is that since the neuron positions are decided you know, where your j is located and you know where your i's are located, so that you can compute for everything that, what this d j i's are and according to that h j i function.

That is, to say topological neighborhood function you will be able to calculate the h j i functions also for that. But coming to the question of the input vector organization to start with you can have, any arbitrary weight vector am I you see that, supposing this is

our weight vector space. So, weight vector could be here, w 1 and w 2 it is consisting of two elements w 1 and w 2.

And there, we can have supposing the first neuron in the topology is having the weight vector here, the second one is here, the third one is here. So, that ultimately when we connect all these things, it may be just a haphazard arrangement like this that, all these weight vectors the topology.

Thank you.