Intelligent Systems and Control

Prof. Laxmidhar Behera

Department of Electrical Engineering

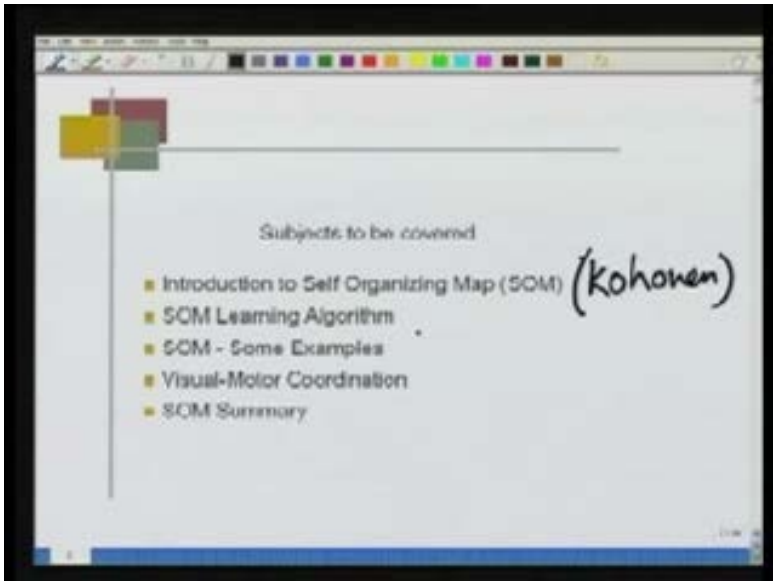
Indian Institute of Technology, Kanpur

Module – 1 Lecture – 11

Self Organizing Map: Multi Dimensional Networks and Application

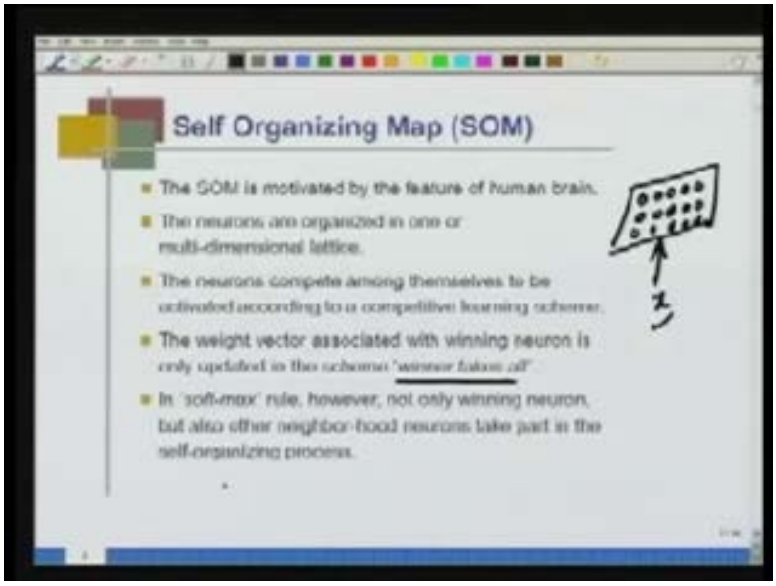
Today, we will discuss a different type of neural network. We have discussed feed-forward network, back propagation network, which is normally multilayer network, also radial basis function network. Similarly, another kind of network we discussed is recurrent network but today's network will be different. It will talk about self-organizing map and specifically, we will only focus on Kohonen feature map, Kohonen self-organizing map.

(Refer Slide Time: 00:53)



Today, we will be discussing about the self-organizing map, specifically, Kohonen. Specifically, we will be considering only Kohonen network. We will discuss what a Kohonen SOM learning algorithm is. We will simulate some examples, we will also show some actual simulation, and we will talk about clustering in a visual-motor coordination and summary.

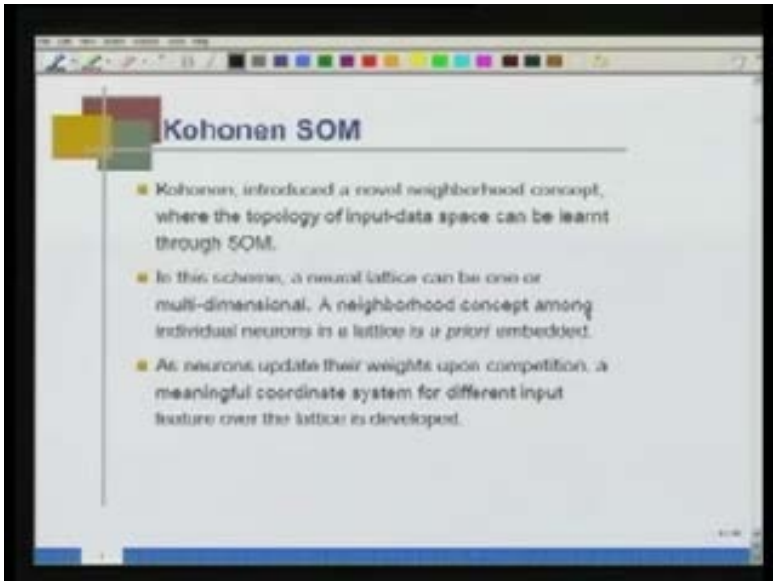
(Refer Slide Time: 02:05)



Self-organizing map is motivated by certain features in the brain, as usual. In this case, the neurons are organized in one or multi-dimensional lattices – 1-D lattice, 2-D lattice, and 1-D lattice. We can think of even higher dimensional lattices also, because in engineering application, we can even become little more abstract. The neurons compete among themselves to be activated according to competitive learning scheme. There are many neurons in a lattice. They are excited by the same input vector or input feature or input, whatever the data is. All the neurons in the space, in the lattice, are all excited simultaneously and there is a competition. The weight vector associated with the winning neuron is only updated and the scheme ‘winner takes all’.

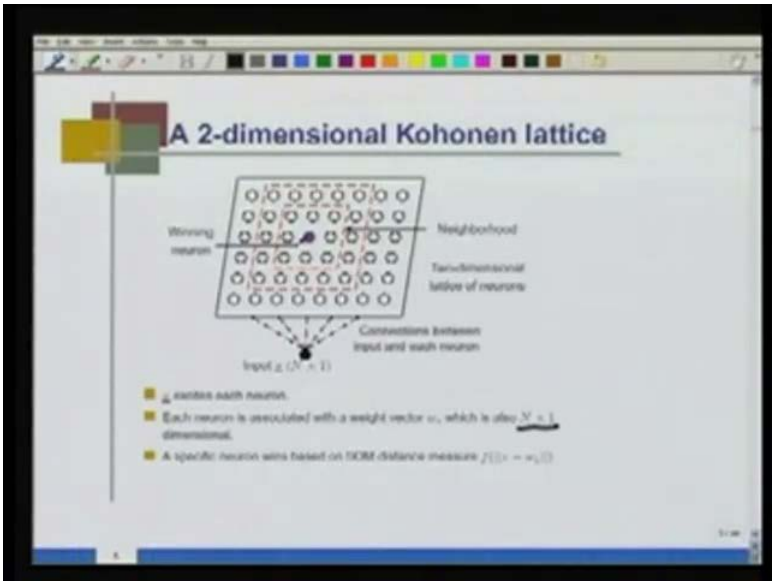
For example, if I have a two-dimensional lattice, imagine I have many neurons placed in this. When an input excites this lattice, one of these neurons becomes the winner. Then, in the ‘winner takes all’ scheme, only the weight associated with this neuron is updated, whereas another scheme that is being employed is the soft-max rule, where not only the winning neuron but also the nearest neighborhood neurons also get updated. The weights associated with those neurons also take part in the decision-making process. That is the principle.

(Refer Slide Time: 03:56)



This is in general, in a self-organizing map, but Kohonen proposed a novel neighborhood concept, where the topology of the input data space can be learnt through SOM. For example, a data that is coming from which kind of geometry can be understood or can be captured through SOM, self-organizing map. In this scheme, a neural lattice can be one or multi-dimensional as usual and a neighborhood concept among individual neurons in a lattice is a priori embedded. As neurons update their weights upon competition, a meaningful coordinate system for different input features over the lattice is developed. We will soon see how this happens in a Kohonen network.

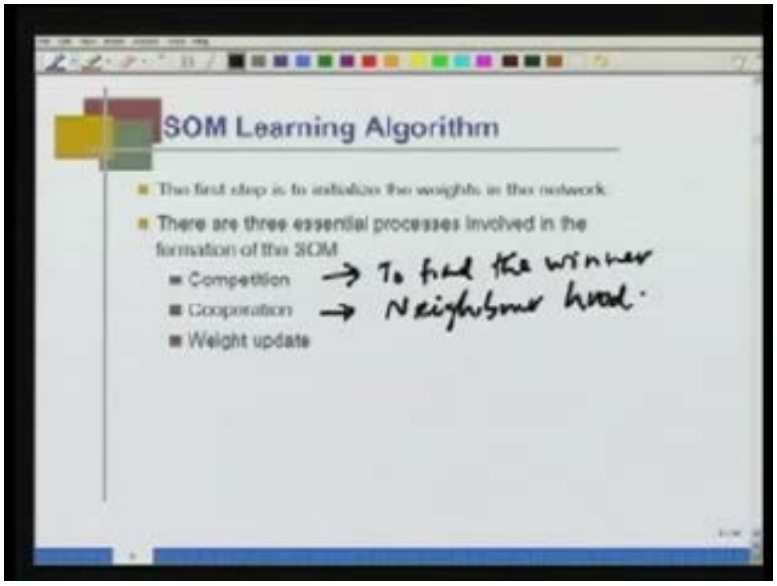
(Refer Slide Time: 05:00)



This is a two-dimensional Kohonen lattice, where you are seeing the neurons all spread over. This is your input vector x, which is an N-dimensional vector. This excites all the neurons. As the data comes, all the neurons are excited. As usual, each neuron is associated with a weight vector w_i, which is also N-dimensional (Refer Slide Time: 05:37). A specific neuron wins based on SOM distance measured, which is some kind of function between a distance measure between x and associated weight vector with a specific neuron. Based on that, a winner is declared. Normally, when this distance measured is minimum in a specific case, that specific neuron is declared as winner.

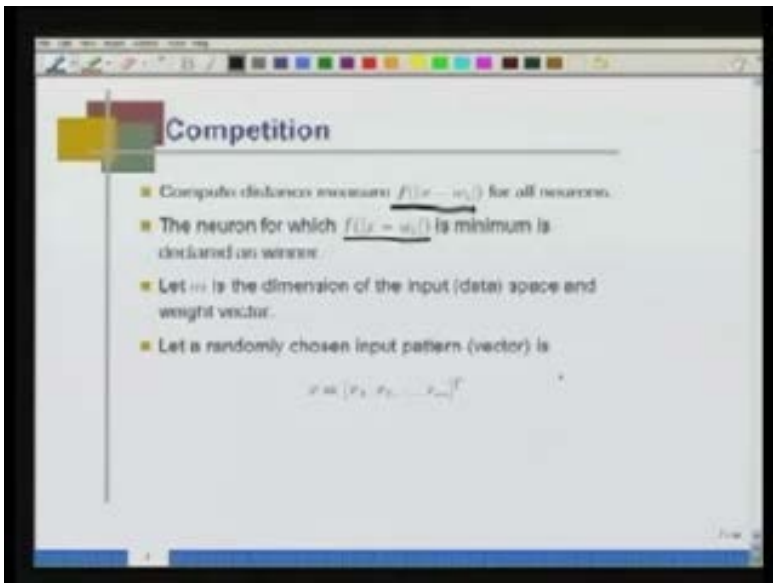
Once we understand the principle, now we will talk about the algorithm. Obviously, as you saw, we can have a one-dimensional neural lattice or two-dimensional or three-dimensional and all the neurons are associated with a weight vector, which has the same dimension as that of the input vector. Before we start the learning process, these weights are all associated with some random weight vector. This random weight vector has nothing to do with the actual input data; it can be anything from anywhere.

(Refer Slide Time: 07:05)



Three things are involved in the SOM algorithm, self-organizing map learning algorithm. The first step is competition, that is, we find the winner and then cooperation, that is, the winner selects its neighborhood, and finally, weight update.

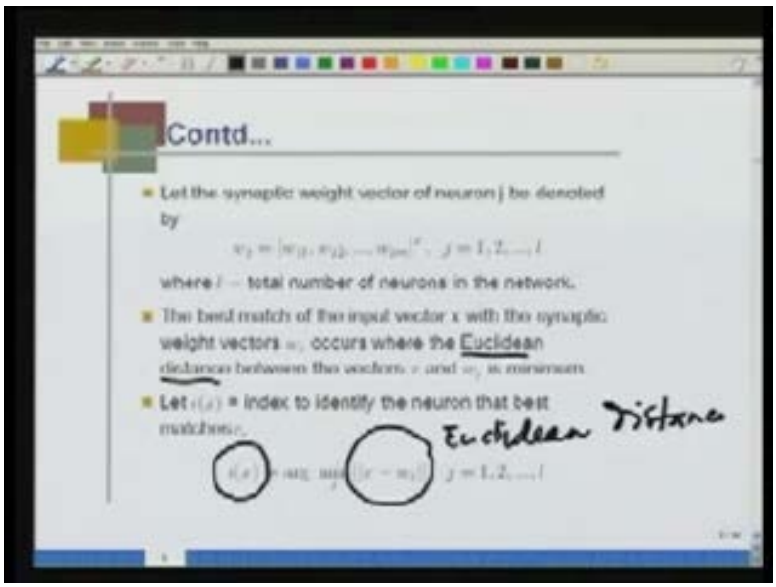
(Refer Slide Time: 07:43)



In competition, we take some kind of distance function, a function of a distance measure and for all neurons, we compute this function. The neuron for which this function is

minimum is declared as the winner. Let m be the dimension of the input (data) space and weight vector, that is, the weight vector and input data vector have m dimensions. Let a randomly chosen input pattern be x. We select this input pattern randomly. It is given by this vector x, which is m-dimensional as you can easily see.

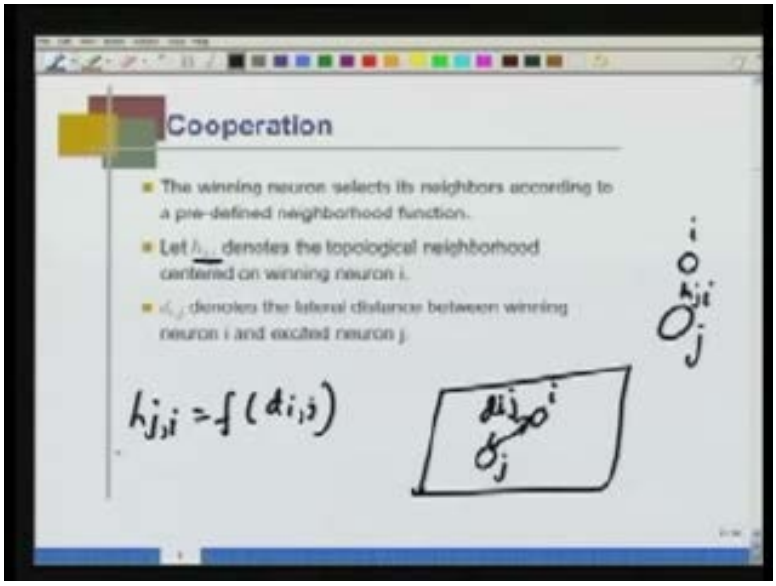
(Refer Slide Time: 08:31)



The synaptic weight vector for neuron j, which is w_j, is w_{j1}, w_{j2} up to w_{jm}. The best match of the input vector x with a synaptic weight vector w_j occurs where the Euclidean distance…. This is just one measure and it is not necessary. There are many SOM algorithms. They use a different distance measure but today, you simply understand… because we are all aware of Euclidean distance.

We compute Euclidean distance between the input vector and the weight vector and for that neuron, where this distance Euclidean distance is minimum, we declare that neuron is the winner. Let i be the index to identify the neuron that best matches x, that is, i of x is actually the winning neuron index for the winner. Obviously, this is the … over j, where this quantity is the Euclidean distance measure (Refer Slide Time: 09:43) and this Euclidean distance is minimum for a specific j and that j is actually i of x for the winner.

(Refer Slide Time: 10:04)



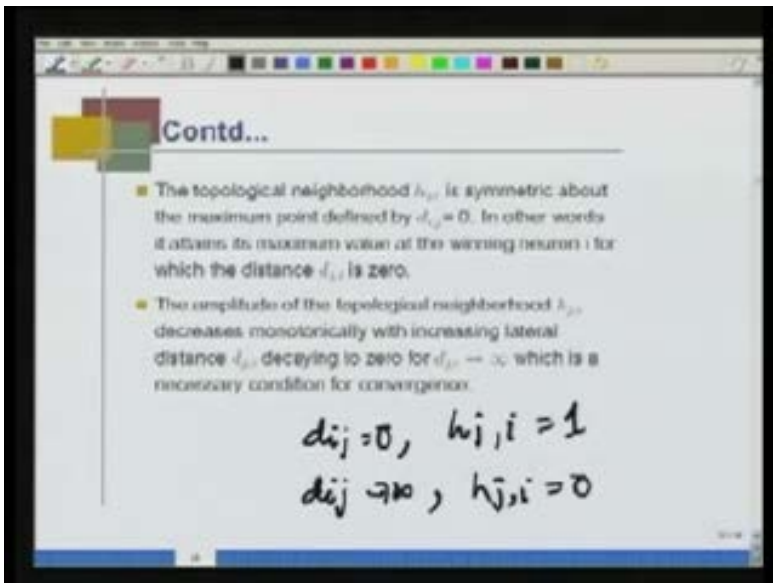
The winning neuron selects its neighbors according to a pre-defined neighborhood function. This is called cooperation, that is, as I told in the beginning, the 'winner takes all' scheme, where only the winner takes the final decision, whereas in soft-max rule, not only the winner, winner allows to be cooperative, that is, it allows its neighbors also to participate in decision making. The winning neuron selects its neighbor according to a pre-defined neighborhood function. Let $h_{j,i}$ denote the topological neighborhood centered on the winning neuron i.

This is my winning neuron i, this is my typical neuron j, and distance between this is $h_{j,i}$. Any neuron around this winning neuron.… The distance from the winning neuron i is $h_{j,i}$. $d_{i,j}$ denotes the lateral distance between the winning neuron i and the excited neuron j. There are two things. Let $h_{j,i}$ denote the topological neighborhood centered on the winning neuron and $d_{i,j}$ denotes the lateral distance between the winning neuron i and excited. There are two things.

That is, for example, I have a lattice here, this is my winning neuron i and this is my j th neuron. They have a lattice distance. Within the lattice, the distance is $d_{i,j}$, whereas the topological neighborhood, that is, how close this neuron is with this neuron is defined by a function $h_{j,i}$. Obviously, $h_{j,i}$ is a function of $d_{i,j}$ or $d_{j,i}$ (whatever you say is all right).

Hopefully, you are very clear about the lateral distance, that is, the absolute distance between any two neurons i and j, whereas $h_{j, i}$ defines the notion of neighborhood, that is, how close is this neuron to this neuron compared to other neurons. This is the kind of a function. So this is a function of $d_{i, j}$.

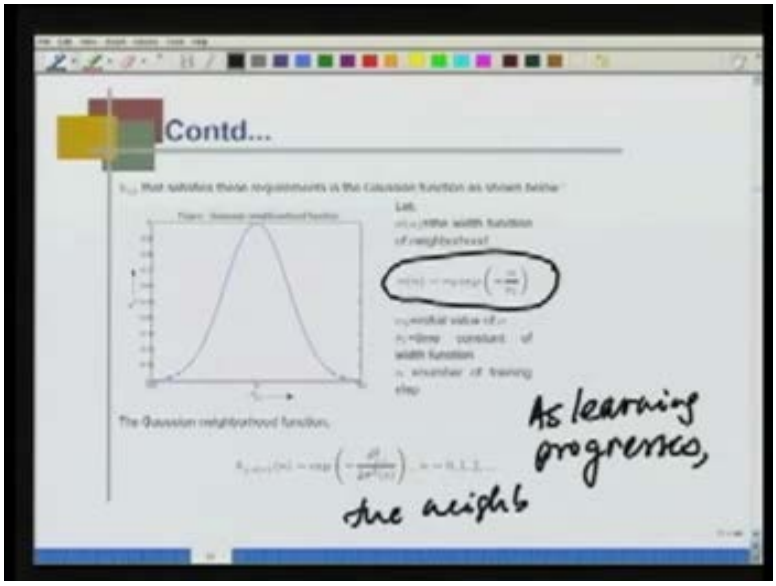
(Refer Slide Time: 12:52)



The topological neighborhood $h_{j, i}$ is symmetric about the maximum point, defined by d_{ij} = 0, that is, when distance is 0, lateral distance, that is, when the neuron for which we are updating the weights is same as the winning neuron, obviously, the distance between winning neuron and the same is 0. For this, $x_{j, i}$ has to be maximum and we can say that is 1 and h, the lateral distance, increases, the function $h_{j, i}$ tends to 0.

In other words, it attains the maximum value at the winning neuron i, for which the distance $d_{j, i}$ is 0. The amplitude of the topological neighborhood $h_{j, i}$ decreases monotonically with increasing lateral distance $d_{j, i}$ decaying to 0 for $d_{j, i}$ tends to infinity, which is a necessary condition for convergence. I hope you understood that if $d_{i, j}$ = 0, then $h_{j, i}$ is maximum and as $d_{i, j}$ tends to infinity, $h_{j, i}$ becomes 0.

(Refer Slide Time: 14:25)

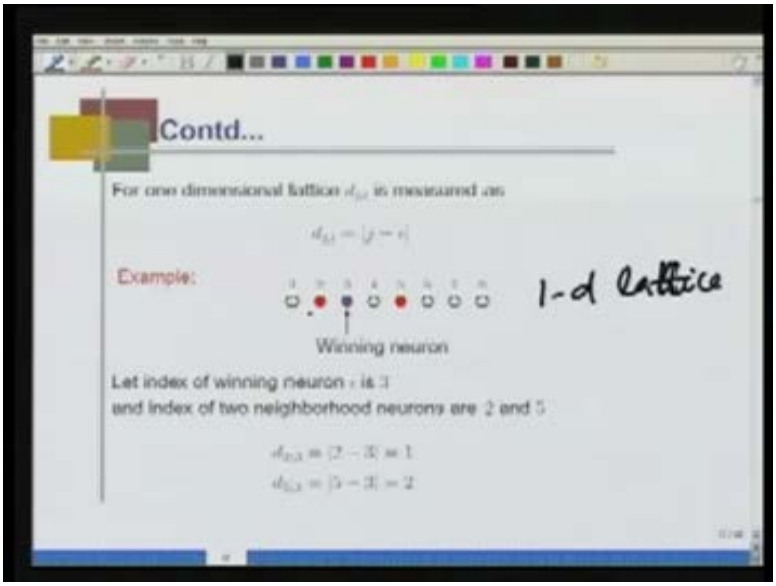


A typical neighborhood function $h_{j, i}$ is a Gaussian function. There can be other functions also, but one of the very typical ones is a Gaussian function. When $d_{j, i} = 0$, its maximum value is 1 and as it goes to this direction and the other direction, its value finally decreases to 0. Normally, this function $h_{j, i}$ is exponential minus $d_{j, i}$ square by 2 sigma square N, where sigma N is defined like this, the weight (Refer Slide Time: 15:06).

What is the meaning of this? If you can see this, when N is the specific learning step…. In the beginning, N = 1. So you can see when N = 1, this value becomes almost like 1. So, in the beginning, sigma N is sigma 0, the initial value. But, when N increases and goes to higher values like 10,000 or 50,000 iterations, then this value will become almost 0. So, sigma N goes to 0.

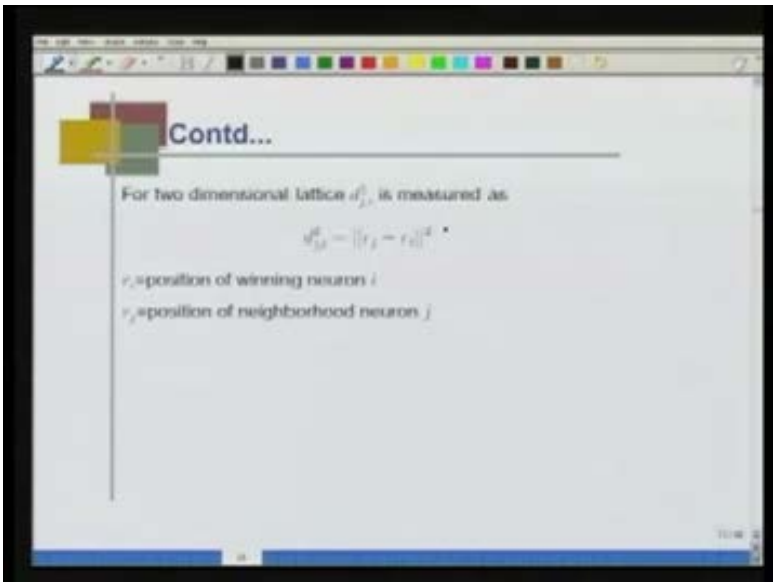
What you are actually trying to do is that using this kind of function, what you are saying is that in the beginning, what happens when a neuron… because the neural lattice knows nothing about the input data. In essence, almost all the neurons in the lattice are considered as a neighbor for the winning neuron but as learning progresses, the neighborhood decreases. That is the meaning of this. The meaning is that as learning progresses, the neighborhood shrinks – that is the idea of this neighborhood function.

(Refer Slide Time: 16:48)



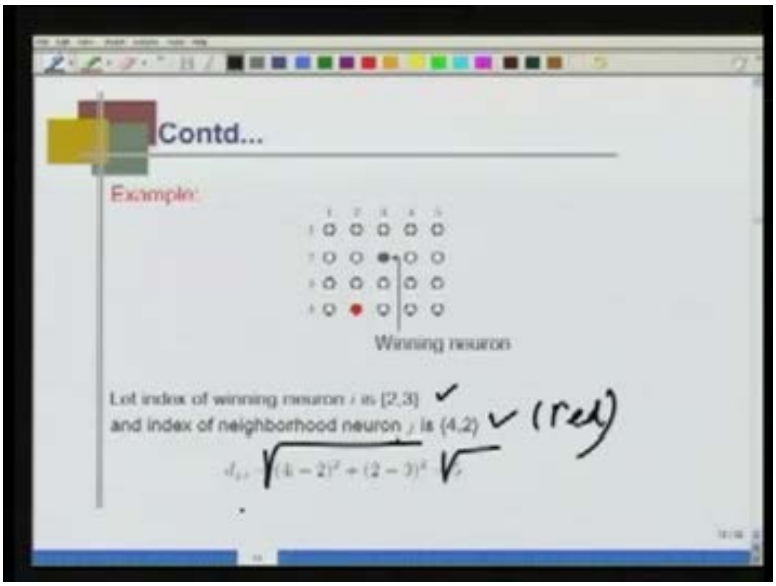
Now, I will give you some example of what is a lateral distance. For example, this is a 1-D neural lattice. In this lattice, this black is the winning neuron and I want to compute what is the lateral distance between this red one 2…. Obviously, this is the absolute magnitude between 3 and 2, which is 1. Similarly, the distance between 5 and 3, we can say this distance major, the lateral distance is 5 minus 3 and the absolute value is 2. This is an example of 1-D lattice.

(Refer Slide Time: 17:37)



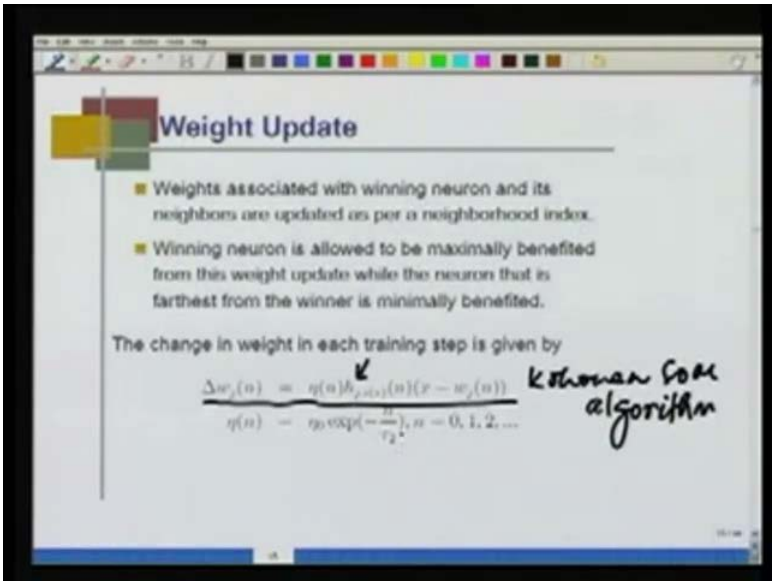
Similarly, we can see the example of 2-D lattice also. In a 2-D lattice, we talk in terms of position vector.

(Refer Slide Time: 17:44)



The example is like this. This is a 2-D lattice. You can easily see this is the winning neuron and this is the neuron for which we want to compute the lateral distance. Obviously, the position of this neuron is 3, 2 or 2, 3 and this is 4, 2. Obviously, the index of the winning neuron is 2, 3 and the index of the neighborhood neuron, the red one, is 4, 2 (this is the red-colored neuron). The distance between these two is 4 minus 2 whole square and 2 minus 3 whole square. This 4 minus 2 whole square is.… Actually, this should be distance like this (Refer Slide Time: 18:37). So 4 minus 2 whole square, that is, 2 square is 4 and 2 minus 3 whole square is 1. So, root over 5 is the lateral distance.

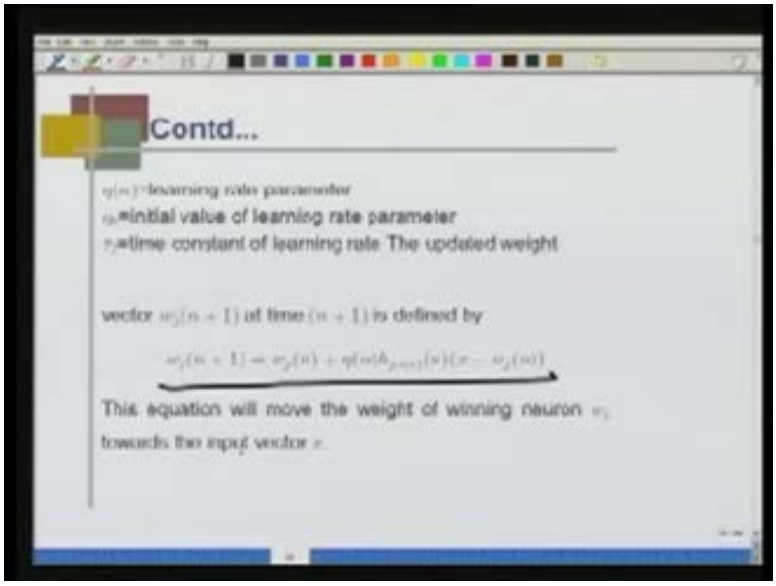
(Refer Slide Time: 18:50)



Once we understood this preliminary concept, then.… Kohonen proposed this algorithm. This is the Kohonen algorithm. What you are seeing is that this is the weight for a specific neuron that is being updated at any instant. This is the learning rate. This is the neighborhood function of the j th neuron corresponding to the winning neuron, which is ix, and this is the input vector, input data x, and this is the weight associated with the j th neuron.

You can easily see if this is 1 for the winning neuron, then what is actually happening is that w is actually closing towards x, that is, that specific weight is moving towards the specific input vector. This is the specific standard one. The weight associated with the winning neuron and its neighbors are updated as per a neighborhood index. The winning neuron is allowed to be maximally benefited from this weight update, while the neuron that is farthest from the winner is minimally benefited. That is the idea of this neighborhood function. We also make sure that the learning rate in the beginning is maximal (the value) and as the learning progresses, its rate becomes very small.

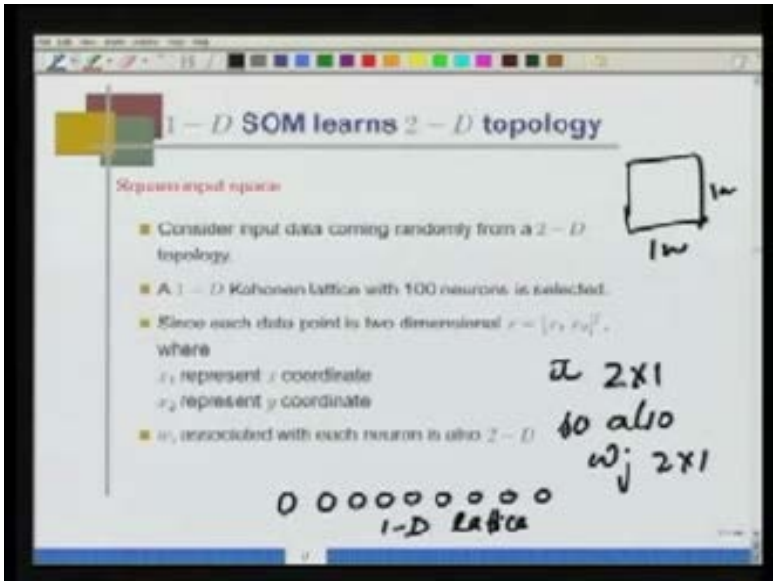
(Refer Slide Time: 20:59)



We already talked about this. This equation that we talked about is our normal Kohonen learning algorithm. This equation will move the weight of winning neuron w_i towards the input vector x – this is very important. This equation will move the weight of the winning neuron w_i toward the input vector x, to maximum, whereas its neighborhood will go towards x according to the distance from the winning neuron.

The objective is that.… Normally, the input data, input space is infinite. So the objective of SOM or self-organizing map is how can I represent a voluminous data or infinite number of data using finite number of samples? This is the basic idea behind this competitive network and in classical terminology, this specific thing is known as how to form clusters. We have already learned clusters are used in pattern recognition, in image coding, in image processing and we used clusters in many other things, specifically, pattern recognition.

(Refer Slide Time: 22:27)



What we understood until now is Kohonen lattice. That lattice can be of any dimension – 1-D, 2-D, 3-D and there is a competition once they are excited – one is winning and others are its neighbors. In the beginning, the neighborhood is very large. As learning progresses, the neighborhood shrinks, that is the idea and finally, the objective is that Kohonen shows that such a scheme preserves the topology of the input space. We will see that now.

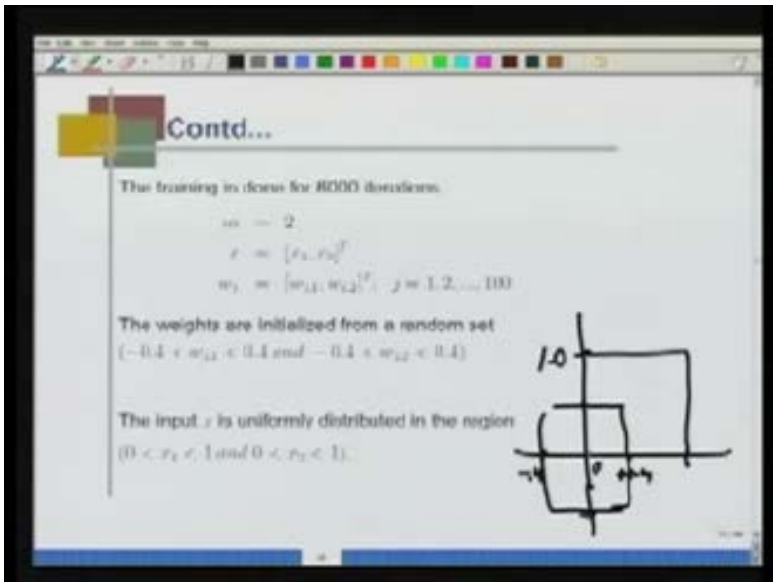
Here, what you are seeing is that a 1-D SOM learns 2-D topology, that is, I have a 1-D neural lattice, a neural lattice that is 1-D. For example, if my neurons are placed in a line, this is 1-D. Now, I will excite this 1-D lattice with data from a 2-D structure. That 2-D structure can be a square, can be L shaped or can be anything, even a triangle. Now, can I say that this 1-D lattice can actually preserve the topology? After I form the clusters using this 1-D lattice from the data that is coming from a 2-D topology or 2-D geometry, then looking at these weight vectors of 1-D lattice, can I conclude something about the structure of the 2-D input space? This is what the question is.

My input space is a square. I take a perfect square. This is a perfect square of 1 meter and 1 meter. My data is coming randomly from this square, I do not know and I excite this data. I excite this 1-D lattice with this data. Now, looking at this the weight vector of this

1-D lattice, can I conclude that my data is coming from a 2-D lattice, which is a square structure? This is the question.

Now, we will do that SOM relation to let you know what actually a Kohonen lattice is, a Kohonen SOM. A 1-D Kohonen lattice with 100 neurons is selected. I selected a 1-D lattice having 100 neurons. Each data point is two-dimensional because any data here has x coordinate and y coordinate and obviously, the weight associated with each neuron of this lattice will also be two-dimensional. So, x is two-dimensional and w_j is also two-dimensional.

(Refer Slide Time: 25:47)

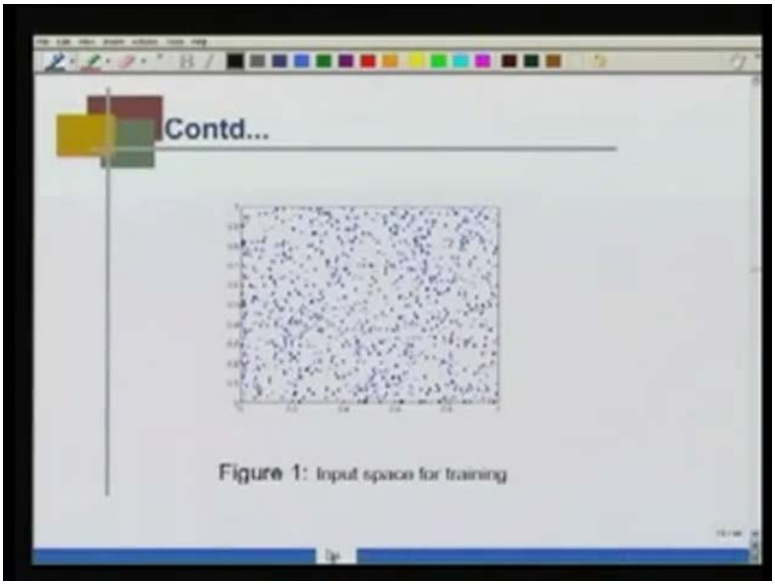


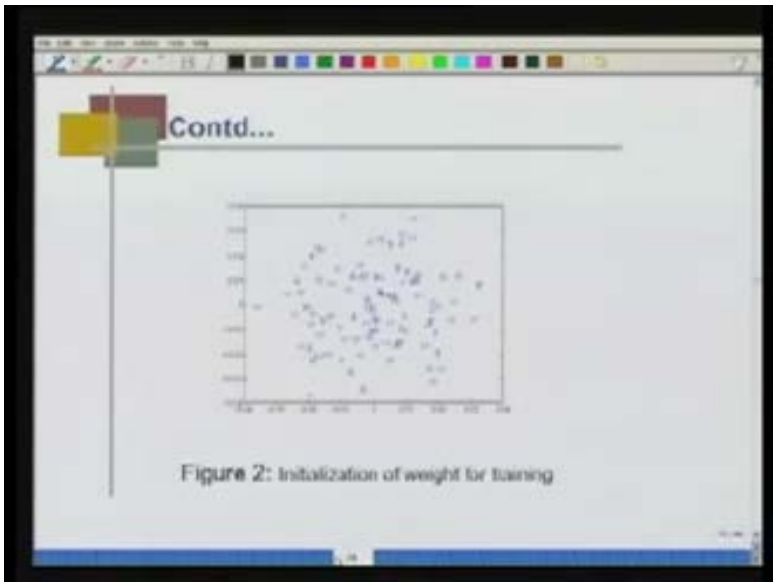
Training is done for 6,000 iterations. Now for this, the dimension is m = 2, x is a two-dimensional vector and each weight vector is two-dimensional. Now, the weights are all randomly initialized for the lattice. The first element of all weights is lying between —0.4 and 0.4 and so also, the second one is —0.4 to 0.4 and randomly distributed. The input x is uniformly distributed in the region 0 to 1 and 0 to 1, that is, my input space is actually 0, 1, 1 (Refer Slide Time: 26:46).

This is my input space and my data is coming from this space, whereas my weights are initialized. This is —4.4, +0.4 and also, this is 0.4 and this is 0.4. This is like this (Refer Slide Time: 27:22). My weights are all confined to this zone, the weights of the 1-D

lattice, whereas my data is actually coming from here. Now, we will see what is happening.

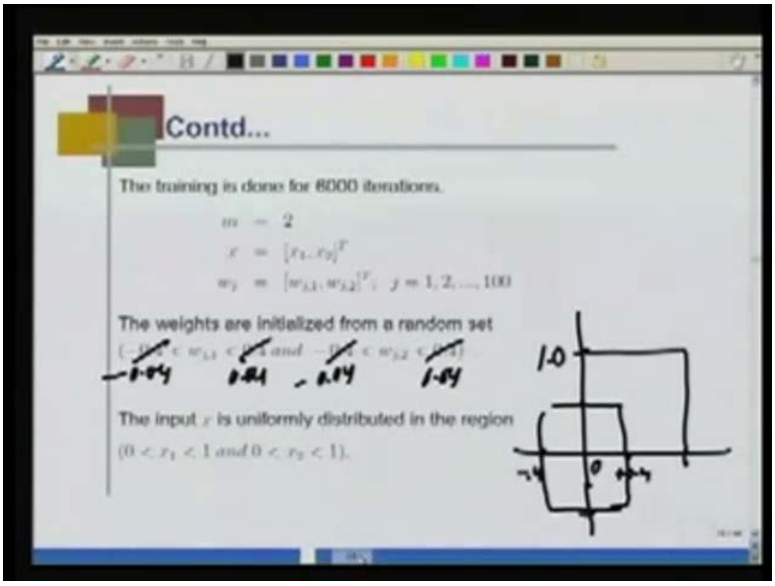
(Refer Slide Time: 27:36)





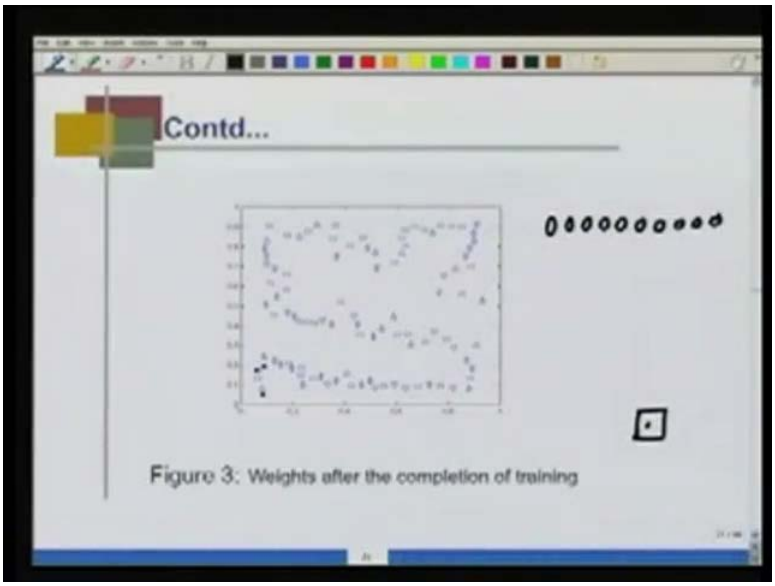
This is my input space, random data and this is the initialization of the weight. You can easily see this is from 0.04 to 0.04 and 0.04 to 0.04. It is a very small zone around the origin and all the weights are all initialized. Obviously, the weights have no value. This is a mistake here, which I showed you.

(Refer Slide Time: 28:13)



This is not 0.4, this is 0.04 (Refer Slide Time: 28:18) to let you know why we have taken a very small value, just to show that you can start from any initial value for the weight and you still capture the topology using this Kohonen lattice (Refer Slide Time: 28:34).

(Refer Slide Time: 28:37)

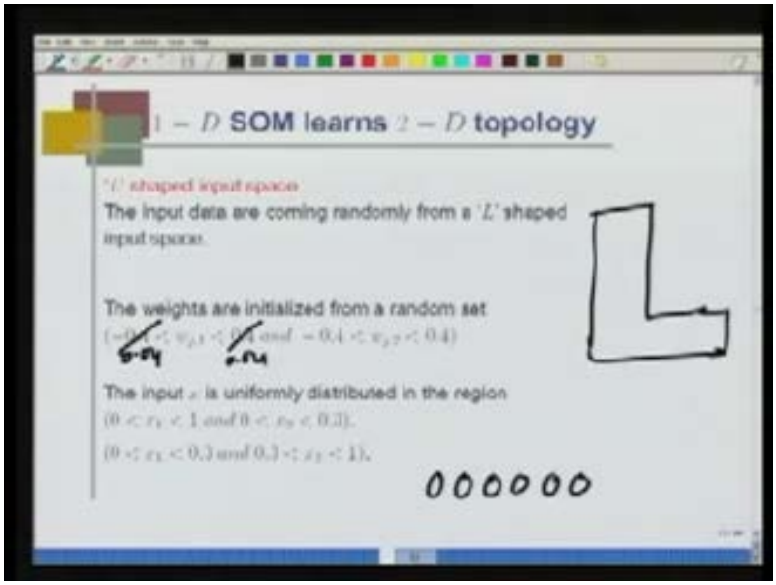


After 6,000 iterations, surprisingly, if you plot the weights sequentially, that is, for example, this is my sequence of 100 neurons and I plot this one (Refer Slide Time:

29:02), then this one is this one, this one is this one, this one is this one and so on. If you look at it, it has developed a structure in such a way that by looking at it, I can easily see that this 1-D lattice captures a 2-D topology. You can see that this 1-D lattice, which was confined to a very narrow zone around the origin, expanded itself – the neurons. (Refer Slide Time: 29:39) in such a way that the weights were so organized that they captured the 2-D lattice. I will show you now in a movie style now how this actually looks like (Refer Slide Time: 29:58).

You see how this 1-D lattice which was (Refer Slide Time: 30:06) the one line in the beginning and slowly, it is developing into a very nice structure and telling us that the data is actually coming from a square. I hope you could appreciate that. Again just to let you know and for those who could not follow it, I will again repeat it (Refer Slide Time: 30:30). You see that in the beginning, if you look at the structure, the lattice is actually taking the shape or it is spreading to let us know that the topology is a 2-D square.

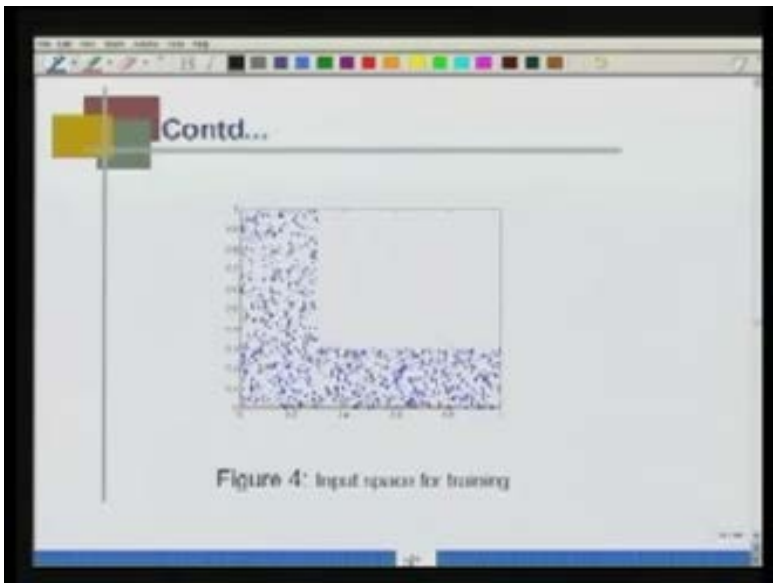
(Refer Slide Time: 30:59)



That was a square. You may say that is nothing surprising, but now, we will take an L-shaped input space. This is my L shape. My data is not coming from a square but it is coming from an L shape. Now if I train, if I update my weight of a 1-D lattice, that is, again, I have 100 neurons in 1-D and they are again in a small.… Again, this is not 0.04,

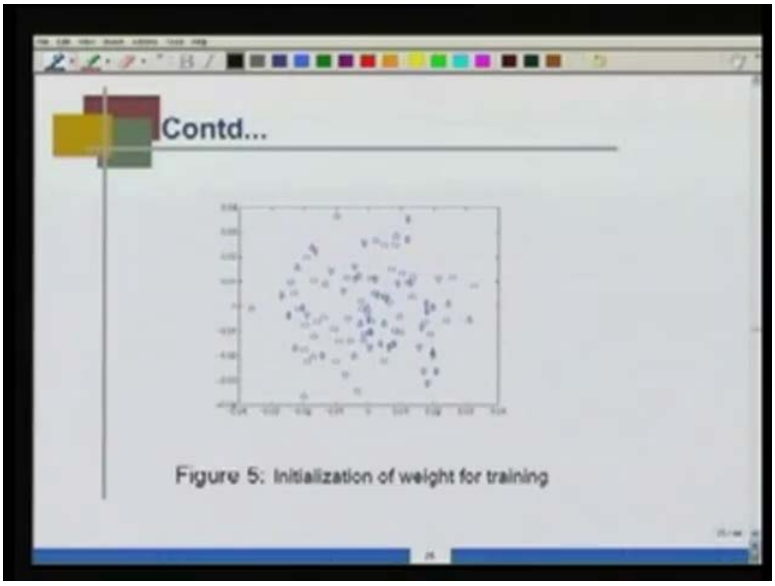
it is 0.04 (Refer Slide Time: 31:42). The weights are randomly initialized. Can I say once again…. Once they are trained, they will give me information that topology is from actually the input space has a topology of L shape. Just looking at the weights of this neuron, can I tell?

(Refer Slide Time: 32:09)



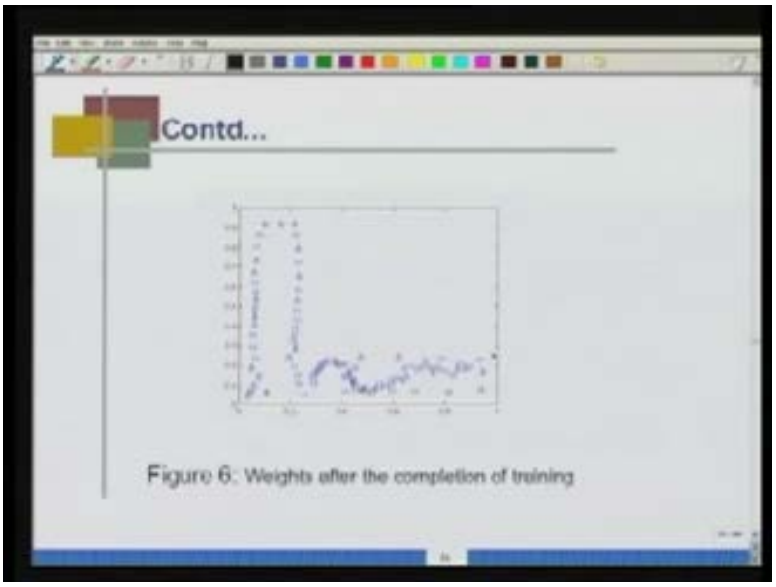
In fact, this is also true here. This is my input space. The data is all randomly generated from this L shape.

(Refer Slide Time: 32:18)



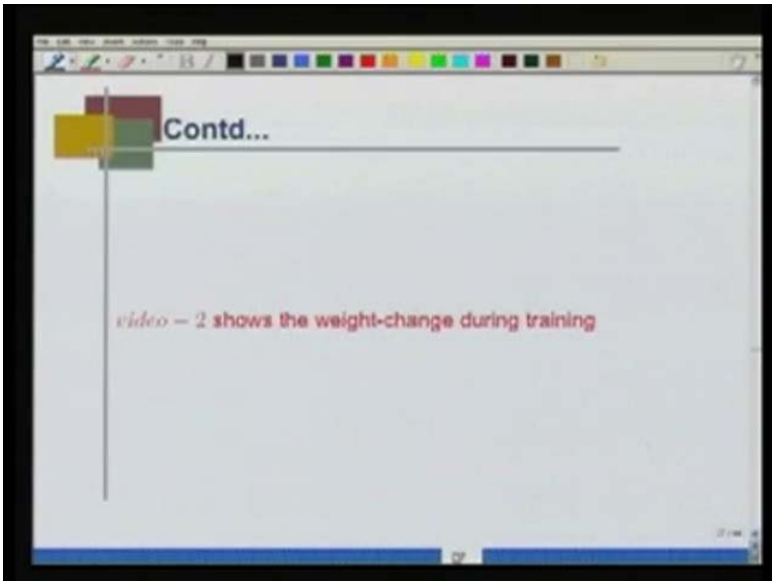
These are my small weights, initial weights – plus or minus 0.04 is the range.

(Refer Slide Time: 32:29)



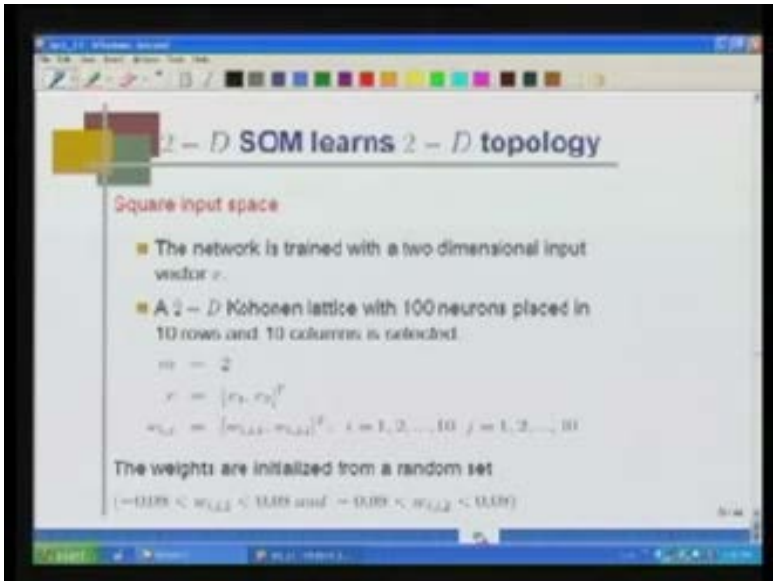
Then, you see that after 6,000 training steps, if you look at the weights, they have exactly L shape. You can easily see the topology of the input space merges to be L shape. Looking at the data, you can infer. Again, I can show you a video on this.

(Refer Slide Time: 32:55)



(Refer Slide Time: 32:57) This is an L shape, the input data, and unlike the other one, which was rectangular, now it is L shape and you see how surprisingly, things are moving in a different manner. So, this is L shape. Finally, for those who could not capture, I (Refer Slide Time: 33:22) to show how actually the neurons self-organize themselves in such a way that we can say that the input data is coming actually from an L shape using 1-D lattice. You see that the first part is already… this part is L (Refer Slide Time: 33:41) and again, suddenly, this is spreading to say that this is L shape. That is something nice to see.

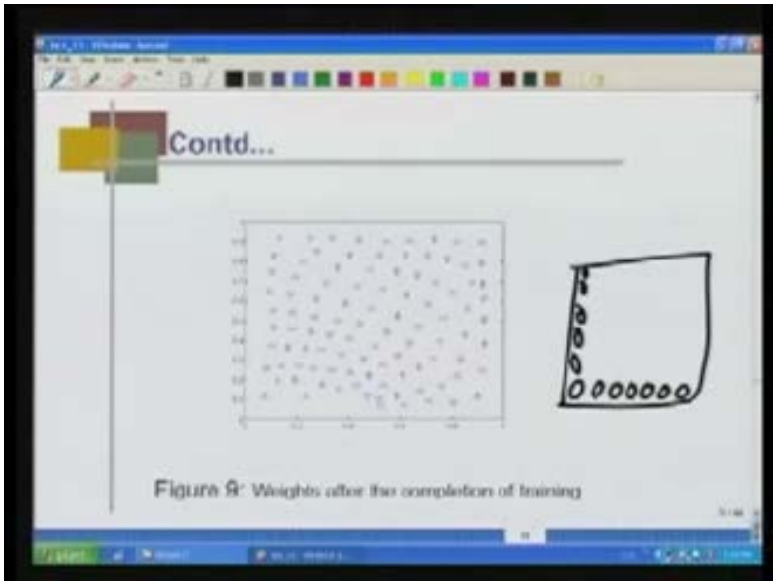
(Refer Slide Time: 33:54)



You can all do it by sitting in front of your computer. Write the program, a simple program. You can write in C or MATLAB and you can have fun with this kind of structure to learn how topology of the input space can be actually recognized using this self-organizing map learning algorithm, only specifically Kohonen lattice. Earlier, we used 1-D lattice. Now, we will use 2-D lattice and again, we will take the same input space and we will see how it varies in the 2-D case.

Again, the square input space – the structure of the input space is a square and we take again a 2-D Kohonen lattice, so, 10 rows and 10 columns. With that, you see that the weight space is initialized in very small values. This is my input space and this is my initial weight, very small weights around origin. After training is over, we can easily see that they are very regularly arranged and these neurons in 2-D (Refer Slide Time: 35:15) in such a way that you can easily see the topology is a square.

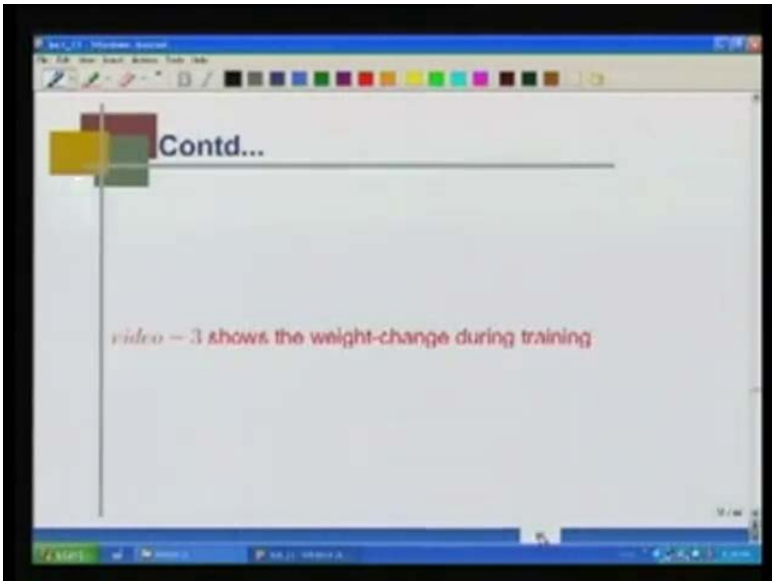
(Refer Slide Time: 35:29)



These neurons we have (Refer Slide Time: 35:24). This is neuron 1, neuron 2, 3, 4, 5, 6, 7, 8, 9. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Similarly, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. You can easily see that: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. What you are seeing is that we have plotted all the neurons in the neural lattice and they exactly occupy a space, perfectly distributed in such a way that the topology is now regular and it looks like a square lattice.

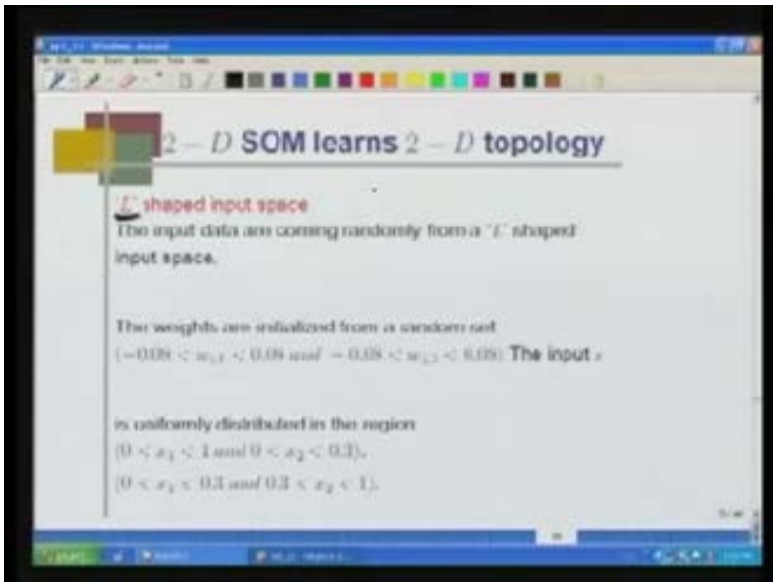
This square lattice will look even better if I increase the number of neurons, 10 neurons in one lattice. I have taken a 2-D lattice and I have placed 10 neurons in this axis and 10 neurons in this axis. If you increase these neurons to be 100 and 100, this regularization of this structure will be much more pronounced. You can do that experiment also.

(Refer Slide Time: 36:27)



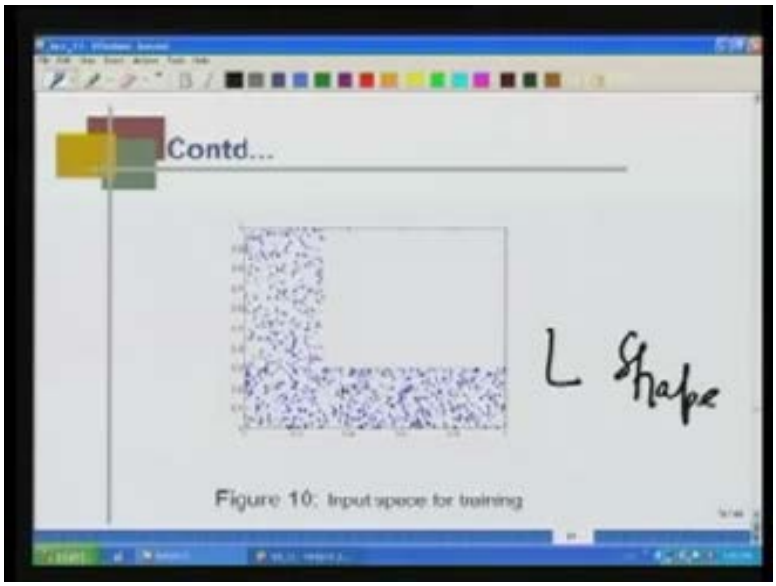
We can easily again see another video. (Refer Slide Time: 36:35) You see that earlier, the neurons were associated and they had no idea about the input space. Slowly, you see that the neurons are exactly taking the shape of a square lattice. For those who did not see it properly, again, for their benefit, I am repeating the same video (Refer Slide Time: 36:59). You can easily see how the neurons did not have any idea in the beginning and they again take the shape of the actual input space. We go to the other example.

(Refer Slide Time: 37:20)



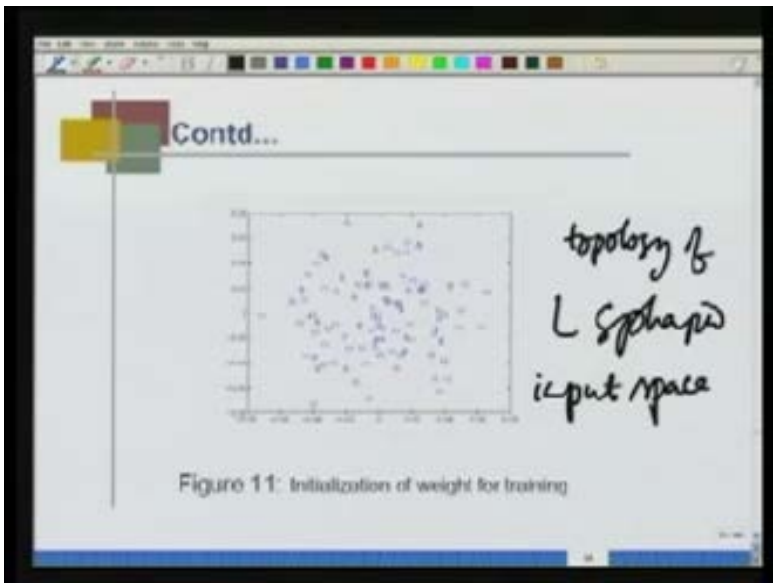
Just like we took the square space, now we take an L space but again 2-D lattice. So, 2-D SOM learns 2-D topology. Again, we will do it in the same manner.

(Refer Slide Time: 37:39)



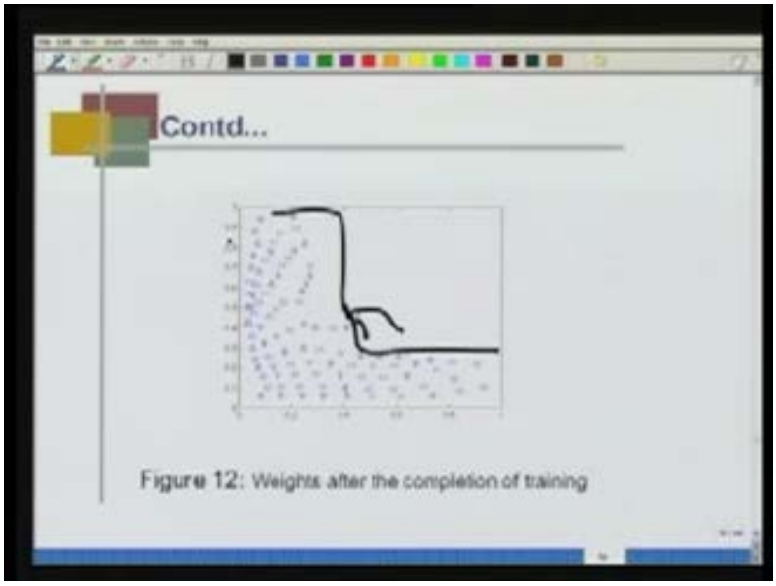
This is L space. This is L shape structure and this is the input data.

(Refer Slide Time: 37:51)



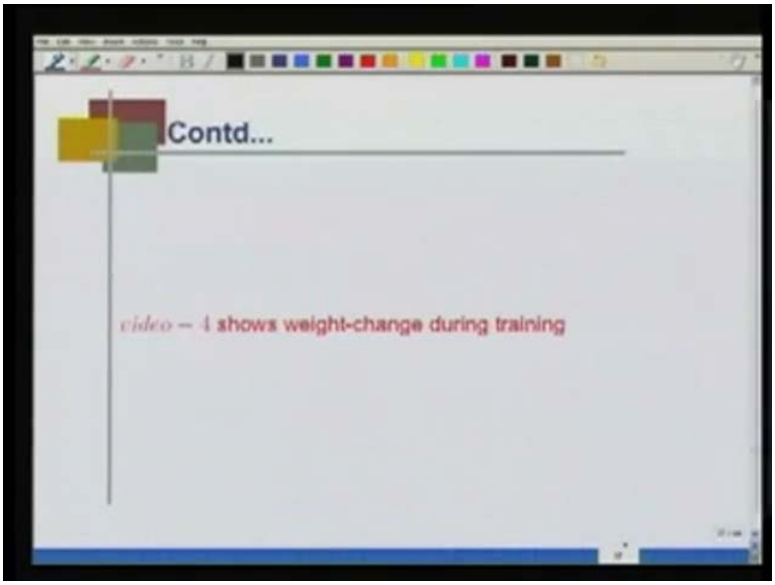
This is the initial weight vector of the 2-D lattice for capturing the topology of an L shape input space. We want to capture the topology of L-shaped input space.

(Refer Slide Time: 38:21)



This is the final structure after learning and you can easily see the topology of the neurons is like an L shape here.

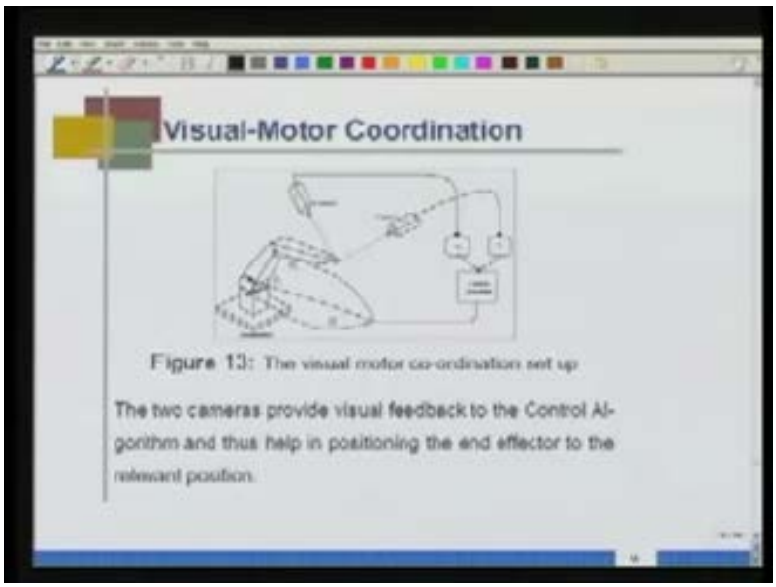
(Refer Slide Time: 38:33)



You can easily see again how this happens in the movie (Refer Slide Time: 38:38). What you are seeing is how neurons in a 2-D lattice are learning L shape in a topology space. You can easily see this is L and this is the other half. For those who could not see it, for

them, I repeat it (Refer Slide Time: 39:00). You can easily see how suddenly this is the structure. The structure of the input space was L shape and that has been nicely recognized. As I told you, we can always improve this by using more number of neurons for each.

(Refer Slide Time: 39:40)



What we learnt until now is that we can use Kohonen lattice to learn the topology of the input space. From which geometry is the data coming? Is it 3-D, 2-D or is it coming from various shapes – circle, sphere, L shape, triangular shape, prism shape? We can actually learn from where the data has come, the geometry of the data – from where it is coming. That is a very nice identification of the feature space, which we will be learning later. We are ready to represent, because this is a very difficult problem. We will learn in this course this (Refer Slide Time: 40:26) but I will just introduce to you what is the visual motor coordination problem.

Visual-motor coordination. For example, I want to manipulate anything, I want to write something on the blackboard. In a biological organism, there is a very nice correlation between the visual feedback and the movement of hand or movement of leg. There is nice hand-eye coordination that we have particularly in biological organisms, specifically for

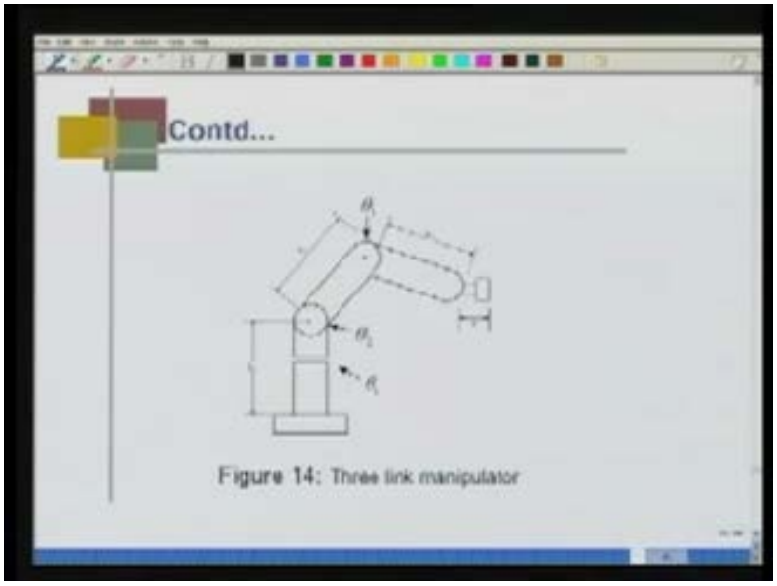
us humans. Some animals also have very high degree of coordination between hand and eye, the visual information and the movement of their organs.

Can we do this kind of things in machines? One example here is that this is a robot manipulator. You can easily see the three-link robot manipulator and you can see that there are two cameras camera 1 and camera 2. They give the information about the target, where these end effectors will go and based on this feedback, where is the target? This end effector has to reach this point. How can that happen through learning mechanism?

This is the camera input. I have a control algorithm that will formulate a map between Cartesian space to joint space. Joint space means theta_1, theta_2, and theta_3. If I give a specific angle set point theta_1, the base will go to that specific angle, this first link will go to specific angle theta_2 and the second link will go to theta_3. Once I know what is the Cartesian space, based on that information, how do I actuate theta_1, theta_2 and theta_3 such that this end effector reaches this point? That is the problem of visual-motor coordination.

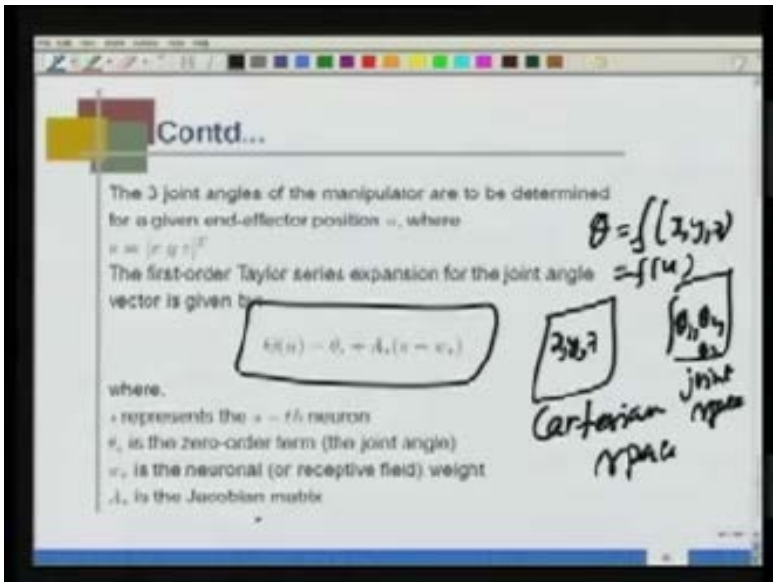
The more difficult problem is that if this target is continuously moving…. If it is static target, the problem is simple, but if the target is moving continuously, how will this go? If it is here now (Refer Slide Time: 42:59), it will take a step in this direction and suddenly, this has gone somewhere else. How can this actually follow a moving target? This is the problem. Two cameras provide visual feedback to the control algorithm and thus help in positioning the end effectors to the relevant position. That is the visual-motor coordination.

(Refer Slide Time: 43:27)



This is our robot manipulator, three-link manipulator.

(Refer Slide Time: 43:31)



The idea is actually that we want to learn a map from Cartesian space, which is actually now four-dimensional, because you see that here, we used the two camera system (Refer Slide Time: 43:48), which is a stereo vision system. If I have one camera, it is very.… Even if I am looking at a three-dimensional object, in the camera plane, it is only a 2-D

point, a 2-D projection of the three-dimensional object. In that sense, the depth perception using a single camera may not be properly done. That is why we talk about stereo vision. We have two cameras to make sure that we have a proper depth perception of the 3-D object and hence, we can exactly locate where the 3-D point is. That is why we are taking two cameras.

What is the objective? I will tell you how we solve this problem using a learning framework or the type of neural network kind of learning that we talk about. I have a Cartesian space here (Refer Slide Time: 44:52). This is the Cartesian space and this is my joint space. To simplify, I say this is a point x, y, z in the space coordinate and this joint $theta_1$ coordinate, that is, the robot manipulator is $theta_1$, $theta_2$ and $theta_3$.

How do I learn the map from Cartesian space to joint space in such a way that the end effector of the robot manipulator reaches exactly the target position through visual feedback? That is the objective. What we do is that from the Cartesian space, we create small clusters and each cluster, we define a linear mapping from the Cartesian space to joint space. This is called…. (Refer Slide Time: 45:51). In essence, what I am saying is that theta is a function of x, y and z, or in this context, it is f of u. u is the four inputs given by the cameras. Two cameras are there and each camera's output is two-dimensional output. So, two cameras will give us four-dimensional output and that is u.

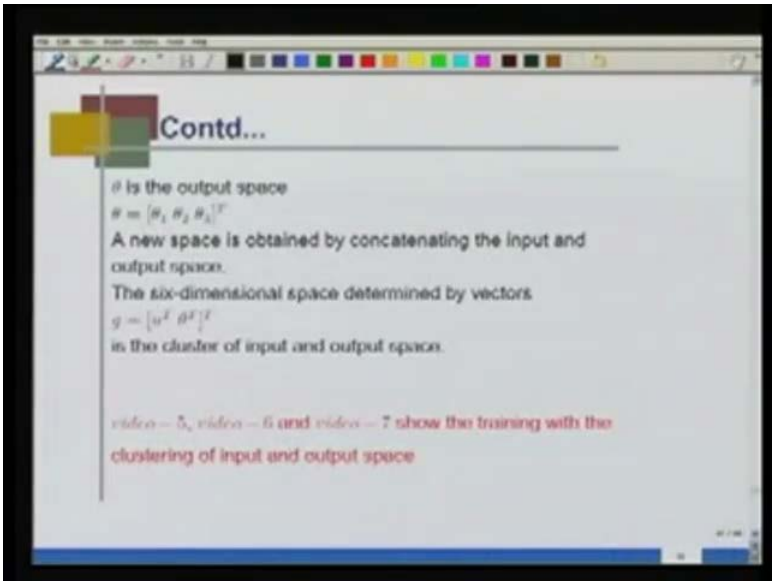
Given u, how do I go to theta? What happens is I divide my Cartesian space into small clusters and within each cluster, I define a linear relationship using Taylor series expansion from theta, that is, theta u equal to $theta_s$ plus A_s into u minus w_s, where u is the input from the camera, w_s is the weight associated with a specific neuron, that is, what I am trying to do is that like a Cartesian space is a 3-D space, I take a 3-D neural lattice (a neural lattice is a 3-D neural lattice) and each neuron represents a discrete cell in the Cartesian space.

Then, within this discrete cell, I define a linear relationship using Taylor series first-order expansion, which is theta. The actual theta is $theta_s$. This $theta_s$ is the specific discrete cell. The center of the discrete cell is correlated to $theta_s$ in the joint space and s is the Jacobean matrix, u is the input from the camera and w_s is the weight associated with the

specific neuron that we are talking about, that discrete cell. So, s represents the s th neuron, theta is the zero-order term, w is the neuronal weight, and s is the Jacobian matrix. This is the idea of forming a map between input and output. Now, what we are trying to learn is a cluster.

We will construct a 3-D neural lattice and this 3-D neural lattice, first of all, must capture the robot workspace. Obviously, the robot workspace may be a 3-D workspace, exactly cubical. There are two ways we can do the cluster. In one case, we can form a cluster formation only in the Cartesian space and the other case is the Cartesian and joint space together – we can also do that. After doing that, we can actually make sure given a target position, how the robot moves or given a target position, how the end effector moves to the target position through learning. That is what we will now learn.

(Refer Slide Time: 49:05)

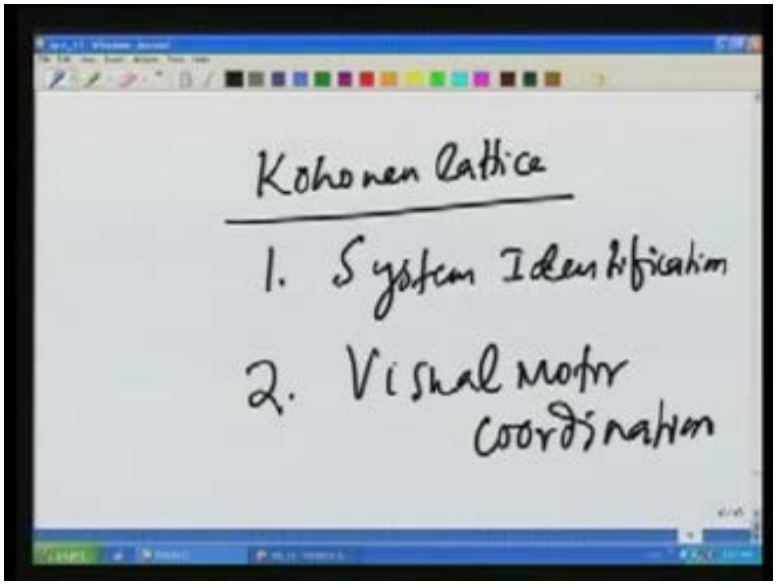


(Refer Slide Time: 49:28) What you are seeing is how a 3-D neural lattice is learning a Cartesian workspace. You can easily see that the data is coming from Cartesian workspace. For those who could not follow this, I will do it again. You can easily see this (Refer Slide Time: 49:53) and you can easily see how the topology of the Cartesian workspace is being learnt by a 3-D neural lattice. This is the first one. Now, I will combine the Cartesian space and joint space and again, we will try it. We will see how

that learning takes place (Refer Slide Time: 50:23). You can easily see that the 3-D neural lattice learns the workspace of a robot manipulator very accurately and this is much better actually in this case when we combine the Cartesian space and joint space and finally, using this notion, I will show you how a robot will actually capture (Refer Slide Time: 50:48)

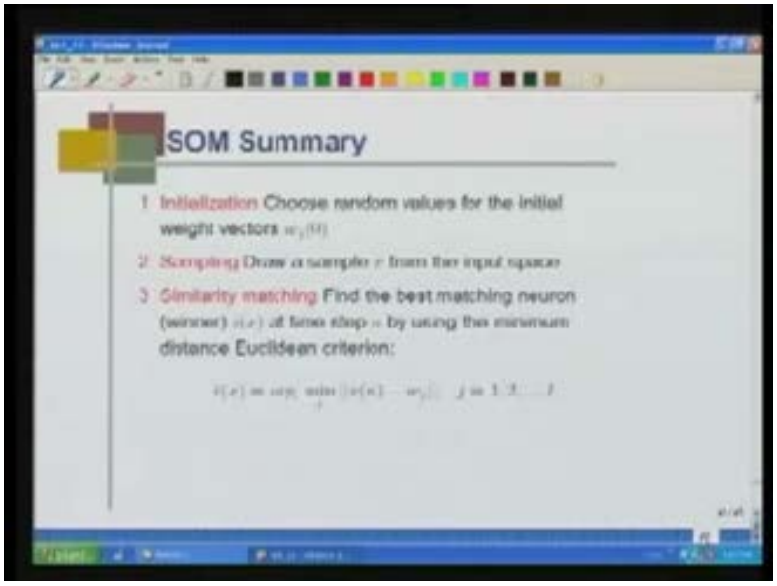
This is my target and this is my end effector and it moves around it and it goes until it exactly converges. You see that it has exactly converged now. For those who did not follow, I again start this (Refer Slide Time: 51:18). You see that this is my end effector and it is going to try to reach this target. It goes very close and finally, it has reached exactly. You can see now that this has exactly reached. Using Kohonen SOM, we can do many things. First of all, let me summarize. Before I summarize, let me….

(Refer Slide Time: 52:07)



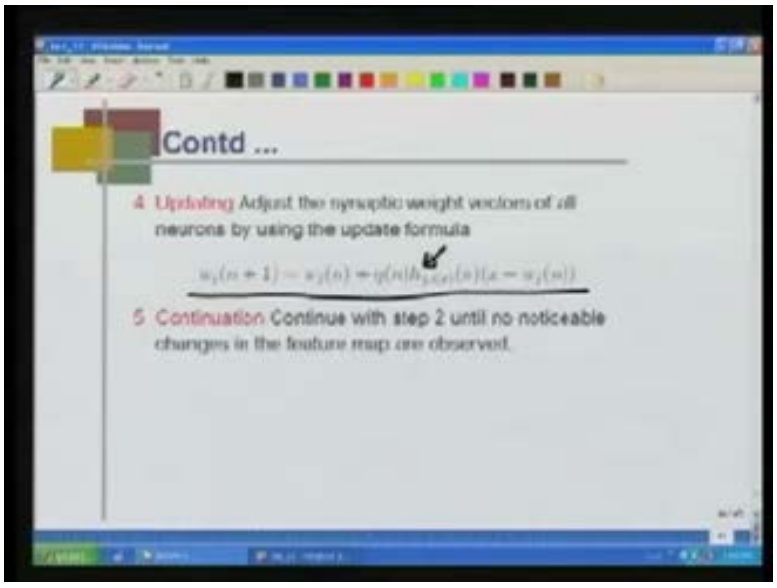
Using a Kohonen lattice, we can do many things. One of the things that I will teach you in this course is how to do system identification. We can easily do system identification and some of the difficult tasks like visual-motor coordination. We will take these two topics exclusively when we go into detailed application of neural networks in intelligent control.

(Refer Slide Time: 52:46)



Let us now summarize what we discussed today. For a Kohonen SOM, first what we have to do is initialize. We select a specific lattice, a Kohonen lattice. It can be 1-D, 2-D, 3-D or even higher dimensional lattice. Then, we assign the associated weight vector with each neuron of the lattice randomly. Usually, the associated vector of each neuron has the same dimension as that of the input data or input vector. You draw a sample x from the input space randomly and then excite all the neurons, find the winning neuron.

(Refer Slide Time: 53:36)



Then after you find the winner, you do the weight update. This is actually the Kohonen learning algorithm (Refer Slide Time: 53:39); this is the sum and substance. It contains a very important item called neighborhood function $h_{j,\,i}$. As I told you in the beginning of the learning, the neighborhood function is adjusted in such a way that almost all the neurons in the lattice take part in decision-making and slowly as learning progresses, very few in the nearest neighborhood of the winning neuron take part in the decision-making. Then, we continue the step from 2 to 4 until the training is over. Finally, when training is over, we can easily see that a Kohonen lattice has actually learnt the topology of the workspace. Thank you very much.