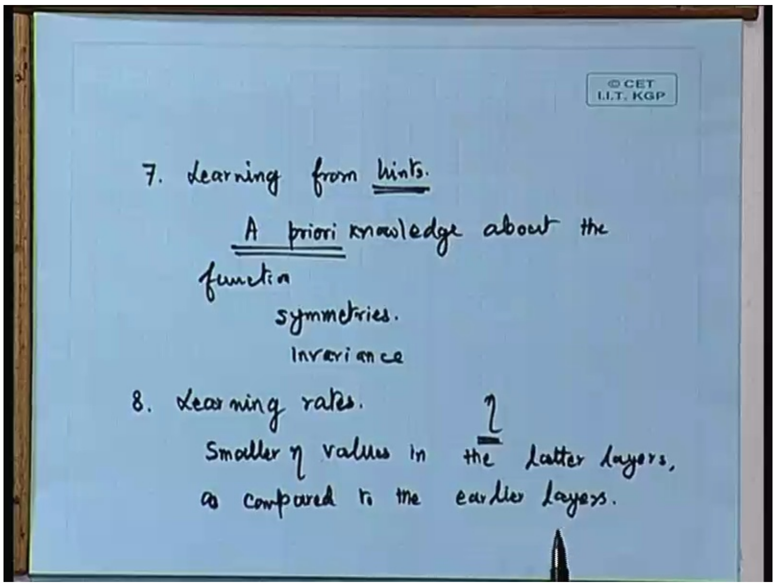
Neural Network and Applications
Prof. S. Sengupta
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture No - 23
Multi - Class Classification using Multi – Layered Perceptrons

The topic for today's class is Multi-Class Classifications using Multi-Layered Perceptrons. So, to many classes and too many multies in our title. Anyway before we begin the actual topic of the day. Let us go back to what we were discussing in the last class. We were basically discussing, some of the heuristics that would help in making the back propagation or multi-layered perceptron networks perform better. And in that, we had already listed out 5 or 6 factors, which effect the performance of the multi-layered perceptrons.

And there are couple of heuristics, which we need to cover further they are quite simple enough. So, it will take the first few minutes of today's class. I will be taking in order to explain the remaining heuristics, which are used for the performance improvement of the multi-layered perceptrons.

(Refer Slide Time: 02:27)


© CET I.I.T. KGP

7. Learning from hints.
A priori knowledge about the function
symmetries.
invariance
8. Learning rates. η
Smaller η values in the latter layers, as compared to the earlier layers.

So, we have listed up to 6 heuristics, so far and the seventh one that we need to talk about is the learning from hints. In fact, what we mean by this is that, when we are actually training neural network, what we are doing is that we are providing the information pertaining to the input output pairs. So, we have got set of inputs in the form of vectors and there corresponding outputs.

So, this input output pairs that we are feeding effectively is helping us in determining the unknown input output mapping function, after all what we are going to find out is the unknown function, which describes the particular input output relationship. Now, we do not feed any mathematical equation to those functions, but from the examples we derived that what kind of function it is, from the examples that the neural network is made to learn.

Now, all though typically a neural network can approximate a function and in fact, we had seen the functional approximation in the case of single layer perceptrons earlier. But, anyway such kind of functional approximations can also be generalized for the case of multi-layered perceptrons. But, the thing is that in order to expedite the learning process, it will be better in order to expedite the process and also make the performance better. It is always good if you can impart some a priori knowledge.

So, what we are for looking for is some, a priori knowledge about the function. We may not know the exact mathematical relationship describing the function, but we may be referring to some properties pertaining to the function. We may be having a priori idea about that. Properties in the sense let us say symmetries supposing. We know that a function is having is odd symmetric about the point zero or it is even symmetry or if the function is invariant with respect to some of the parameter, so symmetries invariance.

Like that several such a priori ideas can be imparted into the training process. So, as to make the performance better and this is exactly what we mean by learning from hints, it is as if to say that when a question is asked, which one finds difficultly in answering. One does not have the exact solution of that. Then lot of times we provide some hints to that problem that how exactly one can attempt to solve this problem.

So, it is by that same notion that the word hints is used where, some amount of a priori knowledge is imparted in to the process. And the eighth and the last heuristics which we are going to consider is the aspect of learning rates. Now, we have been assuming, so far that is more for the simplicity that for a multi-layer perceptrons it is learning rule that we had used earlier. You will be remembering that all the time we are having a factor eta, which is the learning rate.

And the very fact, that we are using eta without giving any further suffixes to it means that as if the eta is going to remain constant, throughout the iterations and also for the… Since it is a multi-layer perceptron and essentially it is structured in to several layers, it is having the input layer and one or more number of hidden layers and finally, it is followed by an output layer where, so many layers are existing then also the inherent assumption by taking a constant learning rate eta. The learning rate is the same for the synaptic weight updating, no matter whether it is output to the last hidden layer or hidden layer two to hidden layer one like that, we are assuming that the learning rate is same throughout. But really speaking should it be same, for the purpose of simplicity we may say that that simplifies our task. Because, we do not have to handle too many etas at the same time, we do not have to fine tune, so many different parameters, if the eta is made variable.

But, the thing that we should note is that when we are making a neural network learn, when we are making a multiple multi-layer perceptron learn. How we are doing that we are finding out what is the error that is existing at the output layer, at the output neuron? We are having the expected values of the output that is to say the targets are available at the outputs. We are also computing the outputs based on the weights, which are synaptic weights which are existing in the network right now.

And then we are finding out the error and based on that error, we are updating the weights. In fact the error that we are having, now the weights are updated in accordance with the local gradients and local gradients is definitely dependent upon the errors. And the local gradients in fact, happens to be of, what would you accept would you accept the local gradient to be higher in the layers that are closer to the output or higher in the layers which are closer to the input.

I mean any guess that you should, that you are in a position to apply.

Student: ((Refer Time: 09:15))

Closer to the output, you will have more gradients yes in fact, that is, so because what happens is that we are finding the output the error, the first error that we are computing is at the output itself, at the output layer itself. And then we are propagating those errors backwards in fact, while propagating it is seen that the contributions of those error terms from the previous layers that is less because, the errors are actually contributed from all the neurons that is feeding to the output layer.

So, that is why the individual contributions of the hidden layer, which is previous to the output will be expected to be less than that of the output and likewise as we go backwards, the local gradient is expected to be less. So, if the local gradients are less as we proceed from the output to the input, then it make sense in having what, higher etas closer to the outputs or lower etas.

Student: ((Refer Time: 10:29))

If the local gradient is more as it is, for the layers closer to the output there we should have because of high local gradients, we should use smaller etas. So, we should use small eta values in the latter layers, by latter layers once that are closer to the output as, so becomes a smaller eta values in the latter layers as compared to the earlier layers by earlier layers to say the layers which are closer to the input.

So, we are not really, if we want the multi-layer perceptron to work well we are not going to keep eta as a constant, eta can be made to vary according to the layer and we should have smaller eta values in the latter layers and we should have larger eta values in the earlier layers. So, that actually the more difficulty in learning will be for the earlier layer. So in order to ensure that the network uniformly learns, at the layers closer to the input as well as the layers closer to the output, we should have the eta distribution like this smaller in the latter layers and larger in the earlier layers.

So, that is the heuristics point number eight, so now that all this heuristics are covered we can go in to the next topic, which is the multi-class classification. But I can just give a brief pause, if you have any questions pertaining to the heuristics aspects that we have discussed.

Student: ((Refer Time: 12:35))

How do we vary eta in this case, well again it is very difficult to give any explicit answer to that, while designing the learning, we have to just use a smaller etas in the later and larger in the earlier and it again the choice of that, should be heuristically determined. If we have some heuristics we can be dictated by that otherwise, we can simply follow some definite rule that as you go to the earlier layers, increase in some manner either linearly, nonlinearly whatever.

So, I think it is very difficult to give any definite answer to this, that how exactly one varies this eta it is more or less like a free choice. But, the thing is that yes rather than using uniform eta everywhere or to say one can go by the experience that, it is seen that by the initial choice of etas that we are having is definitely not, uniformly learning throughout. It is not uniformly having a good learning for the input layers as well as for the layers closer to the outputs, in that case we can think of modifying the eta distribution.

Student: ((Refer Time: 14:22))

For the neurons in the same layer we keep the same eta, we keep the same eta right yes.

Student: ((Refer Time: 14:33))

In fact, yes what happens is that yes the question is that how do we impart the hints, yes again that is also a very interesting questions. In fact, what we can do is that, if we are knowing about such kind of properties symmetries and invariance's then in the training itself, it is possible for us to generate some extra training sets, they may not be from the exact experience. But, if it is known that a function is symmetric or anti-symmetric you can create the extra examples out of that and or you can use the function properties, in whatever ways you can think of.

So, these are the heuristics points that one should keep in mind and now we would like to go over to the.

Student: ((Refer Time: 15:43))

Normalization of inputs as one of the heuristics,

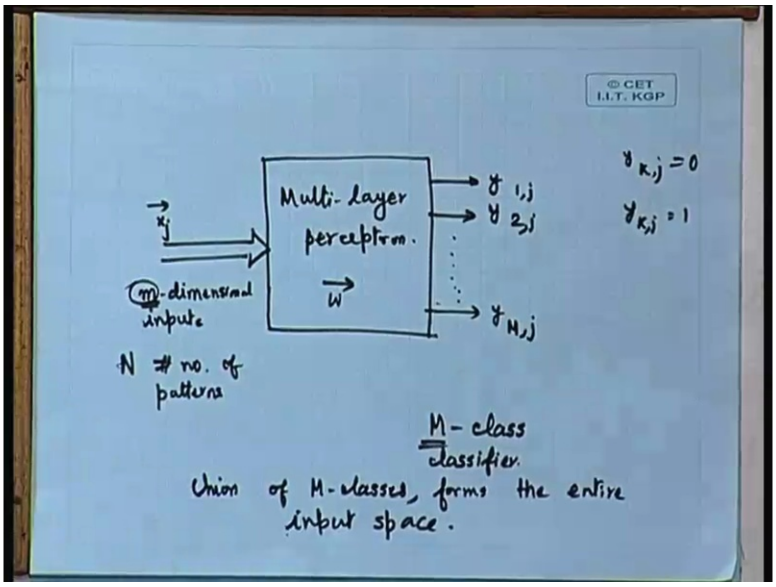
Student: ((Refer Time: 15:49))

Yes, mathematical transformations need to be done there yes.

So, this is about the topic of the day multi-layer, multi-class classifications using multi layer perceptrons, now here what we are fundamentally going to consider is that, we are considering a network, which has got the capability to generate multiple classes. In fact, we did a classification problem in the single perceptrons also and just for our over simplicity sake, we used to take it as a two class classifier problems, a perceptron that could classified in to either the class c 1 or in to the class c 2, that is what we had discussed.

And now that we have got a multi-layered perceptrons, we can think of perceptrons having multiple number of outputs.

(Refer Slide Time: 17:05)



So, we have got multi-layered perceptron network over here and this multi-layered perceptron will be described by its weight vector, so the w vector that we are writing is the synaptic weight vector for that and we will be having to this multi-layered perceptron the inputs and we are assuming, M dimensional inputs as before. So, there are m dimensional inputs and we can assume that there are capital N as the number of patterns.

So, let us take any pattern, let us take the j th pattern, so we can denote the j th pattern by x of j in fact, we are going to write it as the x vector suffix j because all the x's that is the inputs are going to be vectors, since they are in the M dimensional space. And the multi-layered perceptron is going to generate outputs, and in fact, it is multi-class classification problem, so we should have multiple number of outputs.

So, let us say that we are considering an m class classifier, meaning that there are M number of classes, so if there are M number of classes then we should have M number of outputs there. So, we can denote the output as number 1 2 etc up to M over here in fact, the outputs that we are generating will be indicated by y, so we should have we should be writing it as y 1 comma j, the first one we should be writing as 1 comma j and y comma j because that is the output of 1 in response to the pattern x of j.

So, there are n number of patterns and out of that we have picked up the j th pattern and y 1 comma j is the output of the neuron 1 at the output layer in the response to the pattern x of j. And likewise this will be written as y 2 j the last one will be written as y m j, so these are the outputs and there are in affect M distinct classes; that means, to say that the inputs that we are having, the inputs are in fact, going to be points in the small M dimensional space.

So, effectively if we imagine an m dimensional space, all this inputs will be having some discrete points and now each of these points will be indicating one pattern, now each of this patterns will be belonging to some class, one of those M classes and if we make an union of all the M classes. So, if we make an union of all the M classes then it leads to the input space, the entire input space, so union of M classes forms the entire input space.

Now, what happens that there are M number of outputs, but definitely x of j is 1 pattern, so x of j will belong to any one out of these M classes, it cannot belong to more than one classes certainly in that case it is an ambiguous classification. If for a particular pattern, we declare that there are two possible classes, that is not the unambiguous answer that we are giving, we are giving ambiguous answers if we have multiple number of classes for the same input.

So, definitely for unambiguous classification, which is our aim only one of the classes should be active and all the other classes should be inactive, all the other classes should not be there. In fact, we can simply look at the classification problem, in the easiest way that we can imagine that all these y's that we are generating as outputs are all binaries, binaries indicating that when the outputs are 0; that means, to say that it does not the pattern x of j.

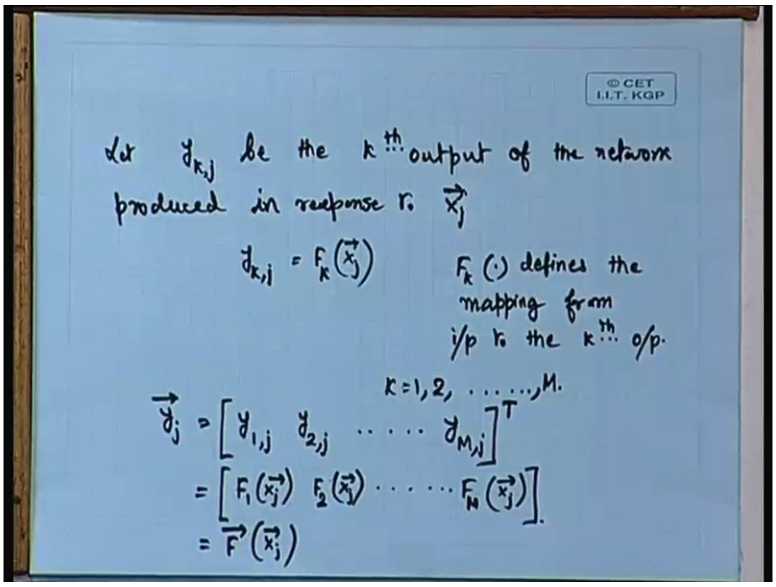
I mean if the output k or if y k j is equal to 0, this will mean that the pattern x of j does not belong to the class k and if we have y k j equal to 1; that means, to say that the pattern j belongs to the class k. So, we can imagine like a binary or other possibility is that, if we do not have because the problem is of what nature we do not know, if it is not exactly like a binary because, after all we will be using the activation functions like the logistic function or tan hyperbolic we will be using any sigmoidal functions, which are essentially continuous function, so it can have values in between 0 and 1 as well.

In which case, the classification has to be looked upon based on some membership function, like if we feed a pattern x of j to the input of the system and observe that what are the outputs, what the outputs are then it may be that we find. I mean, supposing we find that one of the outputs is let us say 0.8 or 0.9 something of that nature whereas, all the other classes are indicating membership values of 0.1 0.2 of that type, in that case there is a very reason to believe that one, which is giving us a much higher score is the correct class for that particular pattern.

So, it is not mandatory that it has to be a binary we can take real numbers also, but then the point is that the question that we have to answer is that, if we have multi-layer perceptron that can classify M classes then what is the optimal number of, what is the optimal decision boundary that we can find for such kind of a classifier. That is the question that we are going to, we are trying to answer in today’s class, so, now we are feeding x of j as the input and we are getting such m outputs as the classifier output.

So, now let us take the k th output of such classifier.

(Refer Slide Time: 25:07)



© CET I.I.T. KGP

Let $y_{k,j}$ be the k th output of the network produced in response to $\vec{x}_j$

$y_{k,j} = F_k(\vec{x}_j)$

$F_k(\cdot)$ defines the mapping from i/p to the k th o/p.

$k = 1, 2, \ldots\ldots, M.$

$$\vec{y}_j = \left[y_{1,j} \;\; y_{2,j} \;\cdots\cdots\; y_{M,j}\right]^T$$
$$= \left[F_1(\vec{x}_j) \;\; F_2(\vec{x}_j) \;\cdots\cdots\; F_M(\vec{x}_j)\right].$$
$$= \vec{F}(\vec{x}_j)$$

So, let y k j be the kth output of the network produced in response to x of j, so that we can write y k j equal to F of xj, where F is the function. In fact, we are going to write it as Fk of xj, where Fk is the mapping function that is associated with the kth output neuron, so we are writing it as y k j is equal to Fk of xj. Now, where in this case Fk defines the mapping from input to the k th output, so now we are feeding the pattern x of j and we should be getting M such responses, we should be getting M such responses.

So, in our case k will be varying from 1, 1 up to M, so that we can form an output vector, so far we were writing y k j is a scalar quantity that we were writing because, we were only considering the k th output, but when we consider all these outputs together, then we have to indicate it with a vector representation. So, we define a vector y of j, whose components are y 1 comma j the first output, y 2 comma j the second output up to y M comma j the Mth or the last output.

And because we are writing it like this, if the original definition is a column vector we should write this one as a transpose of this, so now what is y 1 j, y 1 j is F 1 xj, so this is the first representation. So, the F 1 is the mapping from the input to the first output and likewise mapping from input to the second output is going to be F 2 xj and the last mapping will be FM xj, so in fact, what we are having is a vector of functions you see.

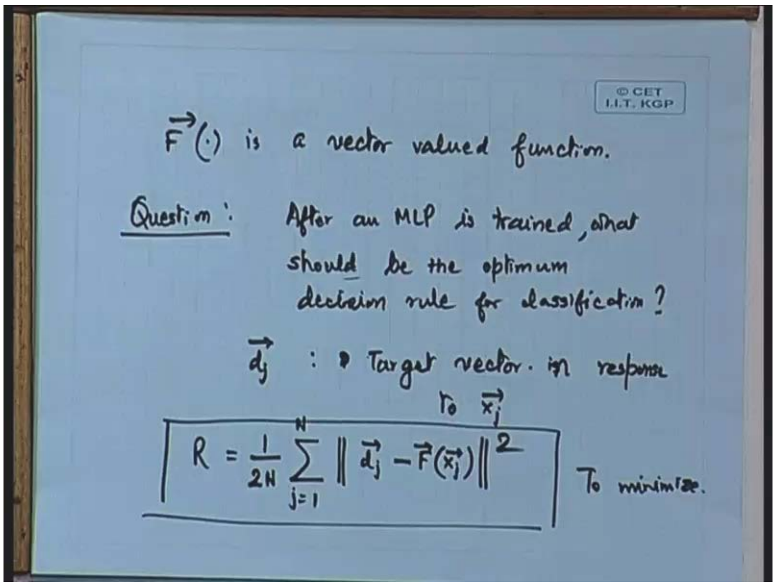
If this F 1 is a mapping function, F 2 is another mapping function, FM is another mapping function, so like that we have got M different mapping functions associated with M different outputs. So, this mapping functions, so we have got a vector of the mapping functions, so in a combined way we can write the function itself with a vector notation, so this is one and the same as writing, F vector of xj, F vector of xj vector which means to say what that the definition of this F vector xj is that, where it has got M different such elements associated with it F 1 xj, F 2 xj up to FM xj.

So, this is the vector valued function so.

Student: ((Refer Time: 29:18))

Is it alright?

(Refer Slide Time: 29:30)



© CET I.I.T. KGP

$\vec{F}(\cdot)$ is a vector valued function.

Question: After an MLP is trained, what should be the optimum decision rule for classification?

$\vec{d_j}$: Target vector in response to $\vec{x_j}$

$$\boxed{R = \frac{1}{2N}\sum_{j=1}^{N} \| \vec{d_j} - \vec{F}(\vec{x_j}) \|^2}$$ To minimize.

So, now here F this is a vector valued function, so now the question that we are trying to answer is that, after an MLP after an multi-layered perceptron is trained, what should be the optimum decision rule, decision rule means decision rule for classification. So, we can write that, so let us see if we can answer to this question, now this F which is a vector valued function, it is definitely a continuous function because after all what we are trying to do is that.

Now, just like the way we have defined an output vectors, there will be desired response vector also, is it is it correct because we are having M different output, so for each of these M outputs we should expect something at the output. So, there should be an expected output of that, so we should have a dj vector that describes the decision or the target vector, so dj is the target vector in response to xj.

Student: ((Refer Time: 31:36))

Can be, cannot be we are only making a mathematical basis for this later on, later on we can always choose for the simplest type of case, yes if it is a clear classification problem, then definitely we should have dj to be the binary values. Even though yj's may not be binary because, yj's will be dependent upon the activation functions that we choose, dj's could be binary because after all at the end we are looking for a crisp classification, an definite classification where, we do not want any ambiguity to be there.

If for some reason, we want some kind of fuzziness to be incorporated in the system itself and we define a membership function, yes in that case we can consider that, even for dj we can have some kind of deviation from this 1 and 0. In fact, another thing that should be looked forward is that, I think we mentioned about that point, while talking about the heuristics that if you were feeding 1 and 1 as the expected outputs, in that case your region of operation is close to the saturation part of the curve.

Where the weight adjustment is difficult and in fact, there is a risk that it may get stuck up, so that is why we were referring to that if it is close to 1 then you take 1 minus epsilon and if it is close to 0, then you take 0 plus epsilon just to avoid keeping the exact 1 and the 0 values. Because sometimes competition this 1 and 0es could lead to problems, so anyway whether we take the exact 0 and 1 or we take some perturbed 0 and 1 that is a matter of choice that we can talk about that we can work out later on.

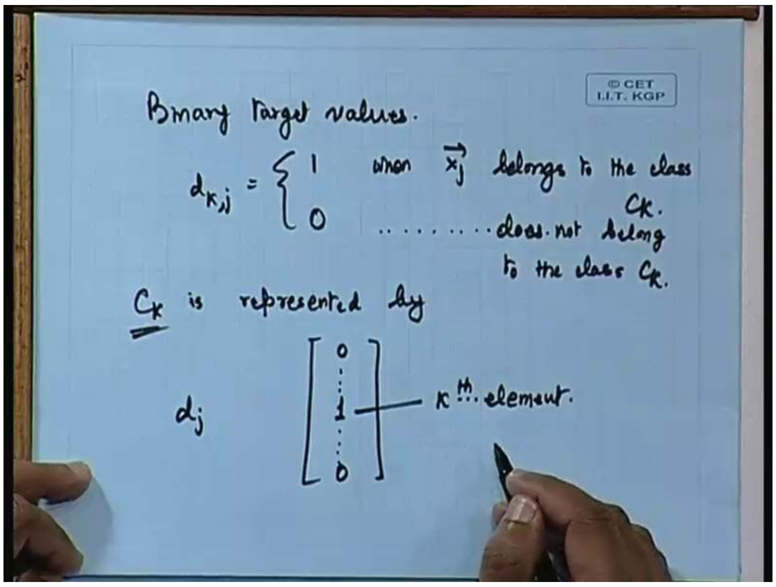
But yes, we need a target vector that is definite and we are deriving the, we are just presenting the mathematical formulation of the problem. Now, F is definitely a continuous function, this vector valid function that we are talking of, it is a continuous function and in fact, it should, what this function should look for is to minimize the risk factor. And what is the risk factor that is associated, simply the reduction or simply the minimization of the mean square error and how do we define the mean square error.

So, we can talk about the empirical risk functional as 1 upon 2 N, in fact it should have been 1 upon N, but again because all the time we are expressing the errors as half of the error squares, for the reasons that we all ready know, so that is why this 1 upon 2 N. But then we are having N as the number of patterns, N as the number of distinct patterns with which are going to train, so then we have summation j equal to 1 to N and the minimization of the risk functional is the norm of this vector dj minus F xj.

And what is Fxj, this Fxj is nothing, but it is the vector of all the outputs, that is to say k is equal to 1 to M, so this is an M dimensional vector and this is also an M dimensional vector and we are taking the squared norm of this, so this is definitely the squared norm of this error that we are computing. And this is for 1 j and then it has to be summed up, this squared norm is to be summed up for all the j's; that means, to say all the training pattern.

So, this is the risk functional that we are going to minimize, so this is what we intend to minimize. Now, supposing that the network is trained with binary values, as one of you were already suggesting in fact, most of the cases it will be that.

(Refer Slide Time: 36:39)



© CET I.I.T. KGP

Binary target values.

$$d_{k,j} = \begin{cases} 1 & \text{when } \vec{x}_j \text{ belongs to the class } C_k. \\ 0 & \text{.........does not belong to the class } C_k. \end{cases}$$

$\underline{C_k}$ is represented by

$$d_j \quad \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \text{— } k^{th} \text{ element.}$$

So, if we train it with binary values, then we should definitely have dk j equal to 1, when xj belongs to belongs to the class ck and it will be equal to 0 in the other case, that is when it does not belong, when xj does not belong to the class ck and why class ck because, we are considering the kth classification and that is why we are taking dk j. So, the kth classifier will be having this kind of a distinct thing, so that actually the ck or the class output is represented by M dimensional target vector.

So, ck basically is represented by with this kind of a binary target values, so here we are having binary target values and for binary target values ck is represented by an M dimensional target vector, so where the target vector will be 0 0 0 0 like that and if this is the kth element, the kth element of this, if it belongs to ck. So, since it belongs to the class ck, the kth element will be 1 and all other elements starting from k equal to 1 up to k equal to M is going to be 0, so this is the M dimensional target vector.

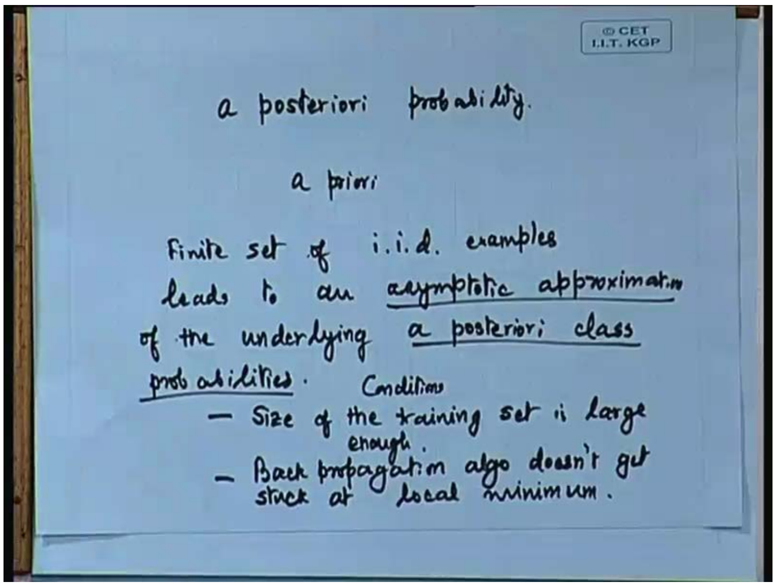
Student: ((Refer Time: 38:40))

This is dj, yes this is dj, so ck is the class, so it is a class pertaining to the pattern x of j simply that, so we have got M number of elements in to it because, it could be in one of the M classes and only at the right kind of class, we are going to put it as 1 and all other elements are going to be 0 understood no confusions. Now, one thing is there that, if we do like this; that means, to say what, that we are after all going to feed a large number of such patterns is in it, we had x of j as one instance of the input.

And like that there will be several inputs that we are feeding and for each of these inputs we are going to specify, during the training for each one of this inputs we are going to specify, that what is the correct class of that?. So, ultimately when our training is completed with all examples fed in, when we have fed all the example, then we have inherently made a class probability, we have in incorporated a class probability.

What I mean by that is, that you have fed n number of such patterns or rather to say n number of distinct points in the small M dimensional space, you have got. Now, once you have specified that to what class each of those points belong, then at the end of feeding that you are getting a class probabilities, which will be associated with it. So, that class probability we are going to call as, in statistical parlance we are going to call it as a posteriori probabilities, a posteriori probability are you familiar with this term yes or no yes.

(Refer Slide Time: 41:14)



© CET
I.I.T. KGP

a posteriori probability.

a priori

Finite set of i.i.d. examples leads to an asymptotic approximation of the underlying a posteriori class probabilities.

Conditions

— Size of the training set is large enough.

— Back propagation algo doesn't get stuck at local minimum.

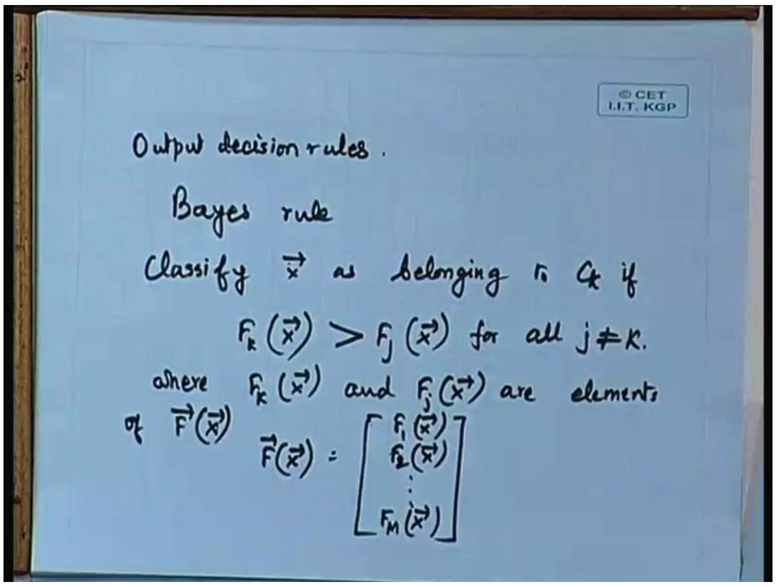
I mean, you are knowing what a priori is, a priori means that before feeding it only you know it, you are starting with the knowledge whereas, a posteriori means that after getting the data you know what it is. So, it is a posteriori probability computation that leads to, so the multi-layered perceptron that we are using for the multi-class classification problem, ultimately it should lead to some asymptotic approximation of the a posteriori class probabilities.

I mean, as you train it more and more, with all this M different classes, datas belonging to all this M different classes then where is it going to converge, it is going to converge in to the a posteriori probability of classifications. So, if we have let us say a finite set of independently, identically independently distributed inputs that is to say, if we consider i i d examples, this leads to an asymptotic approximation of the underlying a posteriori class probabilities.

It is going to happen, that is it definitely leads to this asymptotic approximation of a posteriori class probabilities, but it is going to happen provided you fulfill two conditions and the conditions of that is that the size of the training set is large enough. So, the conditions we are stating the conditions for that, size of the training set is large enough and the other assumption is that, the back propagation algorithm does not get stuck at local minimum.

So, if these two conditions are fulfilled in that case, ultimately we are going to have I mean whatever a posteriori, classification probabilities, we have got out of these training examples, it is going to lead to an asymptotic approximation of that. So, but we were in fact, trying to know that, which is the optimal decision rule and in order to determine this optimum output decision rule, one of the output decision rules that we can simply frame is the Bayes rule.

(Refer Slide Time: 45:19)



© CET
I.I.T. KGP

Output decision rules.

Bayes rule

Classify $\vec{x}$ as belonging to C_k if

$$F_k(\vec{x}) > F_j(\vec{x}) \text{ for all } j \neq k.$$

where $F_k(\vec{x})$ and $F_j(\vec{x})$ are elements of $\vec{F}(\vec{x})$

$$\vec{F}(\vec{x}) = \begin{bmatrix} F_1(\vec{x}) \\ F_2(\vec{x}) \\ \vdots \\ F_M(\vec{x}) \end{bmatrix}$$

So, one of the output decision rules, if we apply the Bayesian rule then that says that, classify x of j or any vector x as belonging to ck as belonging to the class ck, so we have got classes c 1, c 2 up to cM. So, classify it in to ck, if Fk of x is greater than Fj of x for all j not equal to k where, in fact, Fk x and Fj x are the elements of the F vector, elements of F vector x vector, so what is that basically we can just express F vector x vector as the vector F 1 x vector F 2 x vector up to FM x vector.

So, what Bayes rule says is very simple that, you simply examine that which is the highest output value, but basically that starts with assumption that there is a unique maximum value, that is existing there is only one winner that is what you are assuming. But if there are multiple number of winners, if there is a tie then what is going to be your tie breaking rule, so that is not very clearly defined from Bayes rules.

Now, Bayes rule in fact, will work best if we have only one of the, if it really belongs to the class k, if it really belongs to class ck then if we have Fk of x vector equal to 1 and all the others are 0 all the other Fj x vectors are 0; that means, to say that the output is absolutely distinct and unambiguous, in that case applying Bayesian rule makes some sense.

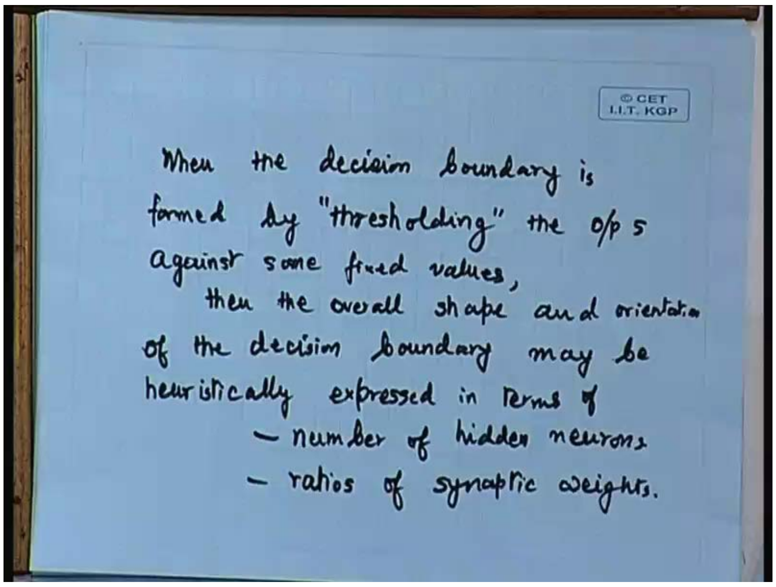
But if we have if we do not have that, if we do not have such kind of binary outputs, in that case there is a risk of getting a tie, the only risk of not getting a tie is that if it is absolutely binary where there is only 1 output and all other 0 outputs no tie no ambiguity. But if the movement we allow the values other than 0 and 1, you can have a tie there and there the Bayesian rule may not be all the time applicable.

And in such cases, definitely we have to go in for a different approach and what are the other approaches that one can take, one can think of keeping some threshold. Let us say that we decide upon a threshold, that if the Fj x vector is above the threshold, then we classify it in to some membership. That means to say that we consider that, this could be one of the probable classes and if it is less than its threshold then we threshold it out, meaning that we do not consider it, belonging to the classes.

But the latter; obviously, leads to multiple class assignments in fact, that is a tie case, which can easily arise out of that, so whatever way we can think of, you will feel that the Bayesian rule is still the best. Even if we have the outputs not exactly in the binary, then also it is a good thing to apply this. But if they are still ambiguous and we cannot have even we cannot even impart any a priori idea about to which exact class it is going to belong, well then you toss a coin, you either put it in to this class or the other class.

But the output decision making function can also be described in an alternative way, you see that if somebody asks us, to talk about the decision surface or the decision boundary then that decision boundary can be modeled, in terms of the networks in terms of its hidden layer also, so the decision boundary. So, when we have in fact, let me explain it in a more clearer term.

(Refer Slide Time: 51:29)



© CET I.I.T. KGP

When the decision boundary is formed by "thresholding" the o/p s against some fixed values,
then the overall shape and orientation of the decision boundary may be heuristically expressed in terms of
- number of hidden neurons
- ratios of synaptic weights.

When the decision boundary is formed by the thresholds that I was explaining, is formed by thresholding the outputs against some fixed values, then the overall size, overall shape and orientation of the decision boundary may be heuristically expressed in terms of its hidden number of hidden neurons and in terms of the ratios of synaptic weights.

When you use such kind of a thresholding schemes, then it is a question of just expressing the decision boundary, where instead of expressing the decision boundary as a simple Bayesian rule, you can express it in terms of the number of hidden layers and ratios of the synaptic weights, the heuristic approach to that. In fact, the hidden neurons are acting as the future detectors ultimately, where the purpose of those future detectors is as I think we have all ready discussed.

That from the input space, we are the input space is actually not linearly separable, in general it is a nonlinearly separable space, but we are using the hidden layers, so that at the hidden space, it becomes linearly separable. And then you are applying your classification rules, that is from the original input space, you are going to the space of the hidden layer activations, where it is more likely to be separable and then you are classifying it.

So, that is something that I wanted to talk about, so we can accept the Bayes rule to be the decision boundary, as the optimal decision boundary, we are all most coming to the end of the discussions related to the multi-layered perceptron. In fact, there are many other aspects of multi-layered perceptrons that one can discuss, I think that with the amount of information, which is all ready generated by the various research groups and people.

One can spend perhaps a full semester on the multi-layered perceptrons itself, but we have to restrict our discussion, we have to put a limit to it, so we are going to put that shortly. So, in the next class we are going to begin the next topic, which is related to the radial basis functions, but before we do that just again, one or two discussions related to the multi-layer perceptron, especially about its generalization capability, so till then.

Thank you.