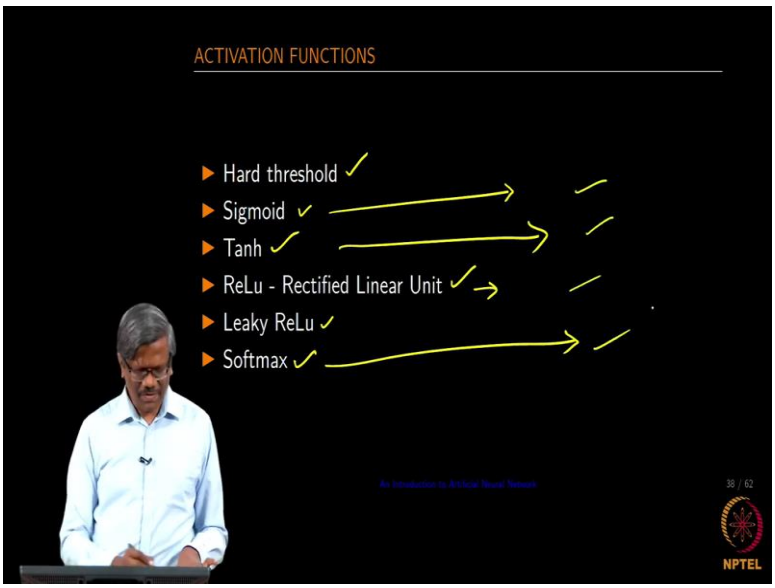
Applied Natural Language Processing
Prof. Ramaseshan Ramachandran
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 36
Activation Functions

(Refer Slide Time: 00:15)



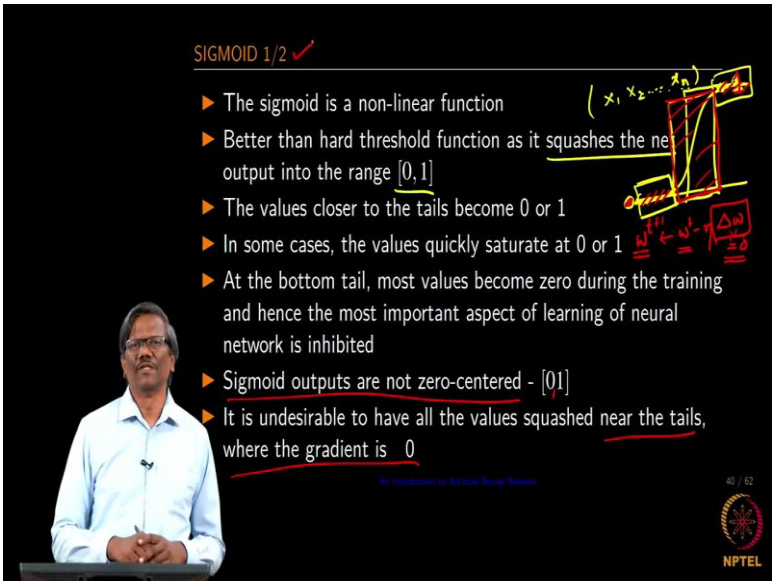
The next important aspect in the nuorolet part is the Activation Function; we remember we spoke about the activation function even in the perceptron. There are we only spoke about one simple activation function there right, it is a hard threshold part.

So, now we going to talk about the sigmoid, hyperbolic tangent, rectified linear unit, and also leaky rectified linear unit and finally, its softmax. So, all are useful you would see in many cases sigmoid and softmax; softmax is usually used at the end of the network layer or at the output layer. Sigmoid tangent or another hyperbolic tangent to you are all used in the hidden layer portions.

(Refer Slide Time: 01:23)

Why do we need different sets of activation function? So we will talk about that now ok. I do not need to explain this in detail a hard threshold is something where values beyond a point it is 1, otherwise, this equals 0 right. So, this is a hard threshold that we used in the perception.

(Refer Slide Time: 01:41)



So, we will now get into a very interesting activation function called sigmoid. Sigmoid is a non-linear function ok, so it converts a set of values that you provide and squashes it into a sigmoidal function as I had shown here in the graph. So, it actually squashes the

values in the range of 0 to 1. So, you have 0 here, and then at so any value that you provide to this activation function, it will always be limited to this range 0 to 1.

And then we have a tail that is this region is called a tail region ok. So, in this particular reason if your value that your computing in the h 1 in the hidden layer if you are getting it here, there will be saturated to either 0 or if the values are here is equal to 1. So, every time when you see that you are very quickly getting into space, you will notice that you are getting into the trouble of learning.

Here, in this case, there is there will be no learning; since the values here every time when you start computing the value would always be equal to 0 right like we have to see in the bias part. In the x, r computation right, so I told you that we are going to be removing that bias because the weight is 0; when the weight values are 0, there is going to be no learning that you will get or if you have it at this point; at this point as well, there will no learning. So, we ideally want to keep all the values in this region maybe let me use a different color here, in this region.

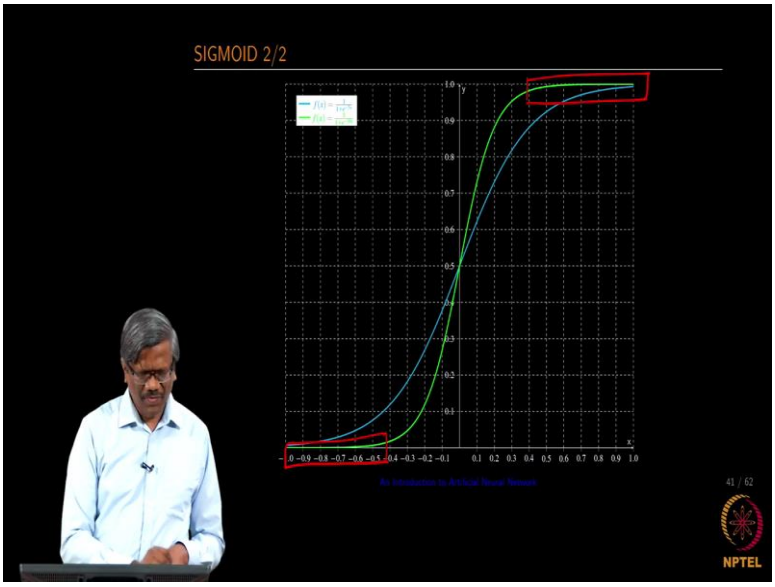
The sigmoid outputs are not zero centered, so as we had seen here 0 is here and then 1 is at this point. It is desirable or rather it is undesirable to have all the values squashed near the tails, where the gradient is 0. So, the reason why the gradient should not be equal to 0 is if we remember we are a calculation of weight is like this, this is the iteration number, ok, so this is our gradient part right.

So, every time when we want to adjust the weights, there has to be some value associated with this if this is equal to 0 which means, that is going to be no more learning or if this system is already learned it. And if you very quickly reached the stage of delta w equal to 0 that means, there is something wrong with your input or something wrong with your network design, something wrong with your weight, insulation and so on ok.

So, you may want to check your architecture and the input parameter that you are giving, so that the network does not get into the stage of delta w equal to 0 in the second iteration or third iteration itself. So, we need to have some difference between the previous iteration and this the iteration, so that this value is not equal to 0 which means, we want to have some gradient managed or maintained.

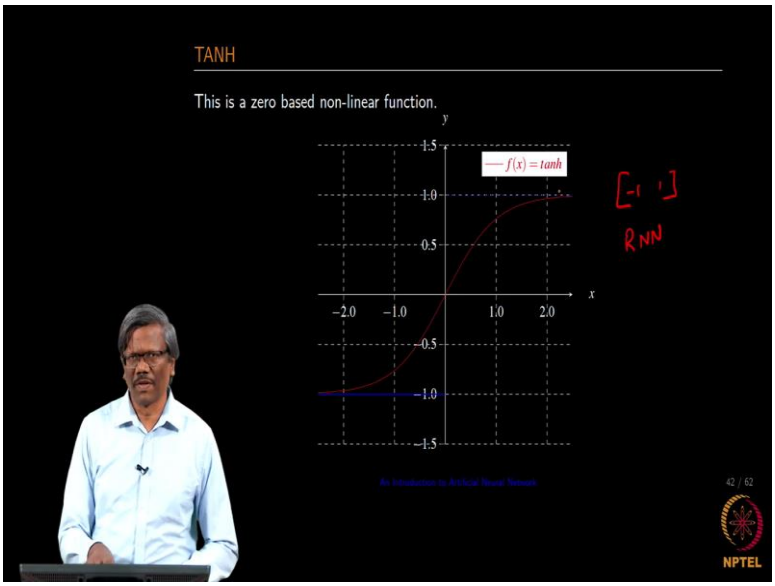
When you have a gradient which is in this place, then you are doing good and if you are gradient if you are already into space, the gradient would be equal to 0, there will be no learning in the network right. The learning or the model preparation is all about finding the right values of the weights, is not it? So, in order for us to get a very stable weight mattress, so you need to have a good activation function that is differentiable ok; note the word differentiable, if the activation function is not differentiable, then it is not possible for you to do the learning for the neural network that is why sigmoid is chosen.

(Refer Slide Time: 06:49)



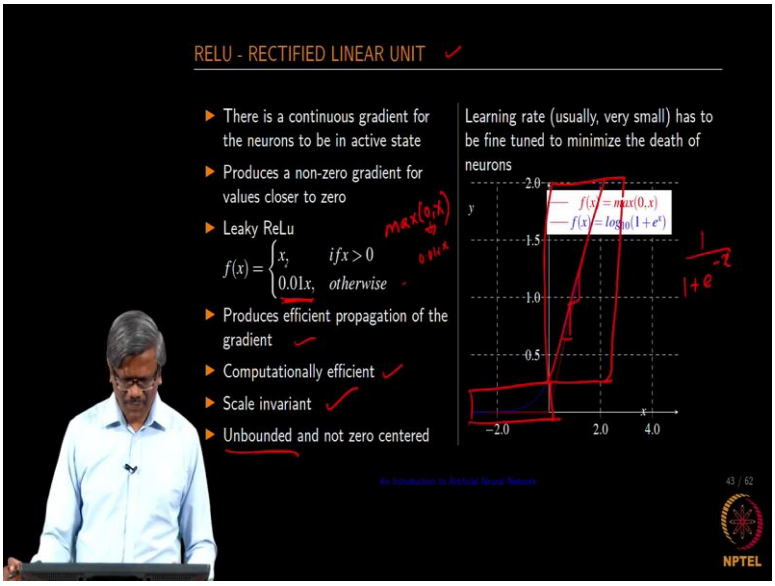
So, there are different varieties you will get an ultimately we do not want to have a very long tail like this, because the values it very quickly saturates to 0 or 1 ok, this is one activation function.

(Refer Slide Time: 07:13)



Then the second one is the hyperbolic tangent. So, this is again very similar to the sigmoid, but the only difference that you notice here is the range is between minus 1 to plus 1. So, you will see this in many recurrent neural network models. So, in the hidden layer computation, you will notice tan h will play a great role in terms of translating or this question the values between minus 1 and plus 1. So, again you have a function which is differentiable.

(Refer Slide Time: 08:01)



And you have one more activation function called rectified a liner unit.

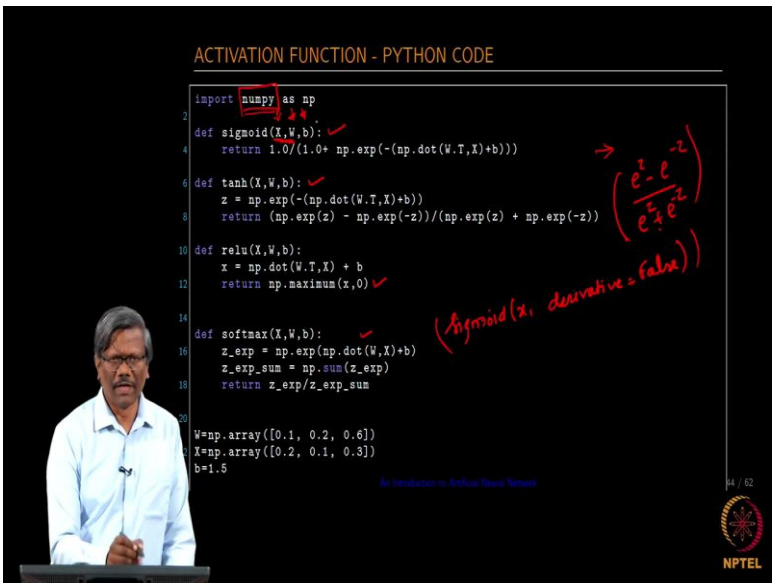
$$f(x) = \begin{cases} x_2, & x > 0 \\ 0.01x, & x \leq 0 \end{cases}$$

So, again it is similar to sigmoid or in the hyperbolic tangent only change would be you have only a tail here and you do not have anything, you can see that it is continuously differentiable in this fashion. So, you always have some value associated ok.

Again it makes sure that there is no 0, so that most of the values would always be kept in this region, so that when you do the error correction mechanism you have some delta value available for you so that the system learns. So, it produces an efficient propagation of the gradient and also it is computationally efficient. So, you do not need to compute that you know if you use the sigmoidal function; the sigmoidal function right. And this is pretty simple, we just very similar to your max value 0 of x, it is just trying to find out whichever is the maximum keep that value, ok, so here it is always so it is very easy to implement as well.

And then it is scale-invariant does not matter how many you have times in multiply the value with scale factor, it does not change. The only difference between those sigmoidal and hyperbolic tangents is this is not a bounded function. So, the values can be anything depends on what value of x he get ok and it is also not zero centers like sigmoidal, ok.

(Refer Slide Time: 10:05)



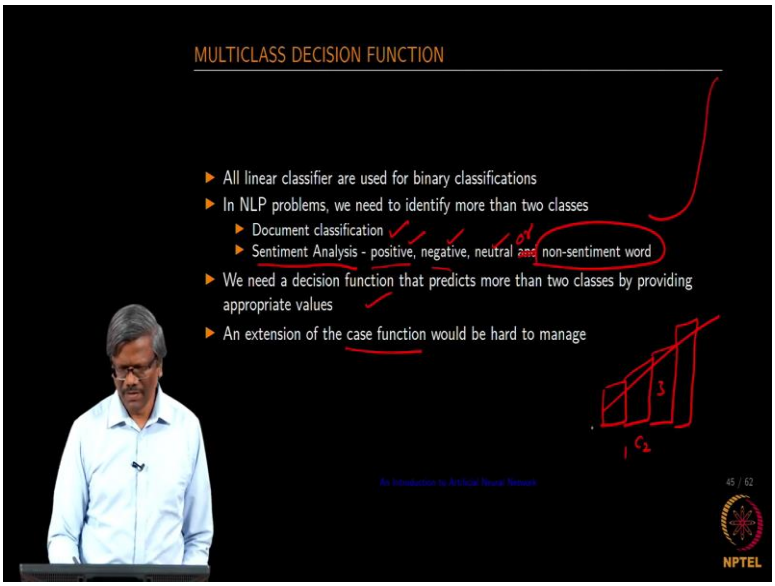
So, I have given a simple implementation of all of this, and like any other application that I have showing as a demo, this also is available as part of the GitHub. So, we can take a look at this, this is a very simple implementation and normal if you include this as part of the class, a network class knows you probably would not have this you just input sigmoid x, ok. If it is part of the class may be a python class, you will have this as x.

And then still since you are going to be used in derivatives of the sigmoid ok, in the next stage of network building. We will also have another variable instead of this, sorry about that sigmoid so that when a pass the value either you want to find the sigmoid or you want to find the derivative of this, ok. So, in one simple function you can achieve both ok.

So, use NumPy you know it is there is a big debate going on in the radiate in terms of whether it is NumPy or NumPy or something else right. So, I will be using NumPy or NumPy, whichever comes to at the point in time. We have tan hit is very simple, but tan his and this is what is implemented and then relu, I use a maximum I am just using a very simple thing between 0 and the value not using the leaky relu that I had shown earlier.

And then the next one is softmax, will talk about that and then come to this one. So, in this case you want to find the sigmoid of this using this function, you need to provide the input the weight vector and the bias and it will automatically give you the sigmoidal value for that ok, all right.

(Refer Slide Time: 12:43)



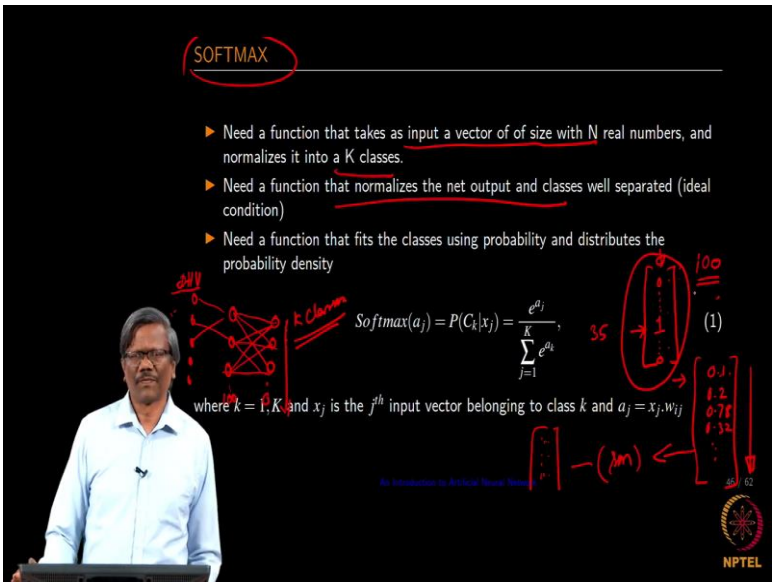
So, we are going to be having more number of classes that you want to figure out, as I mentioned earlier in the previous lectures. If we have collected a lot of document related to the computer science topic you know, you would like to have a folder called an operating system, you want to have a compiler design for one folder and then you want to have discrete mathematics and so on and so forth, right; so that going to be multiple classes that you will find in the real a situation.

So, we should be able to accommodate the multiple classes in the case of the neural networks as well. So, how do I accommodate that so most of the activation that function that we are talking about either squashes the value between 0 and 1 or minus 1 and plus 1? So, it is going to be very difficult for me to use multi-class using that rights of you are having several values, I cannot say that use the value for class 1, this for class 2, 3 and so on, right. So, instead we need to have a separate decision function that allows you to create multiple classes in itself.

So, let us talk about that we need this for document classification. Even in the sentiment analysis you know, especially in the movie reviews you want to find out whether the sentiments are positive sentiment, negative sentiments, neutral or it is not related to the movie. For example you have received a lot of documents, out of which one document is an outer layer meaning that the document does not belong to any one of the movie reviews, it is something else ok.

Again you want to find out whether the word that you are talking about is positive, negative or something else ok, let us put it as or. So, in those cases, it is necessary to have a decision function that allows us to predict more number of classes and not just two rights, as I mention the extension of the case function is very hot to manage.

(Refer Slide Time: 15:15)



So, we need to definitely have one function that allows us to predict more classes that are where, softmax comes into the picture. So, it has to take a set of vectors of size N, and then finally creates a K class says. For example, I have a network that takes input parameters such as this and this is my input layer and I have a hidden layer and they are interconnected, and then I have output layer again they are interconnected in this fashion. And it should output K classes ok, let us assume that I have more of these, so K classes. Is given in equation

$$\text{Softmax}(a_j) = P(\frac{c_k}{x_j}) = \frac{e^{a_j}}{\sum_{j=1}^{k} e^{a_k}}$$

So, the values should be able to tell me whether the one that I have recently computed using the network belongs to one of the K classes or not, so that is the idea of having a K classifier in the neural net model. So, let me give you one example let us assume that we

have captured all the words as a dictionary, and then we have created the one hard vector for all the words.

You know what one hard vector right for each word, suppose if you have about 100 words that we are going to be using for the training, we will have a vector containing 100 elements and the index of that element would point to the word that we are talking about right. So, I am going to be inputting these 100 elements one hard vector into this and then at the end in the output layer, I want to be able to predict whether it is one of those 100 words; it is not about positive or negative, especially in the case of word embedding this is what we are going to be doing ok.

So, I am going to be inputting one word using the one hard vector and then the weights are initialized in some fashion and I have more than one hidden layer rather hidden units let us assume that says about 100, ok. And then I have K classes assume that there are 100 classes there we going to be looking at meaning, each word will be considered as 1 class.

So, when the output is generated it is not going to be in an ideal situation like one hard vector right, in this way. So, it will have some values all the elements are all the neurons will have some values, let us say that they range from 0 1 0 and so on and they are not bounded in some way. So, what we want to do is we want to see whether the output values since we are expecting and output very similar to these values except for this particular index, let us assume that this is the 35th element. The 35th element should have some value; the rest of them should have 0 values.

But ideally since we are doing the computation in the real space, it is not going to have 0 or 1 in the binary form, this going to have some values. What I want to this ok, so I want to have a distribution of values between 0 and 1. And then the 35th element I want that to have a higher value, you know when the network is trained very well.

So, when the network is not trained, any value in the output element could have a higher value correct. So, for me to distinguish which one should have the right value, I need to have a proper distribution of values on this scale. So, what we do is we do softmax computation, where the values of the output would be translated into a probability distribution and every element will have a probability value. And the sum of all the values if you take it, you know after week translate this into softmax; if you push it

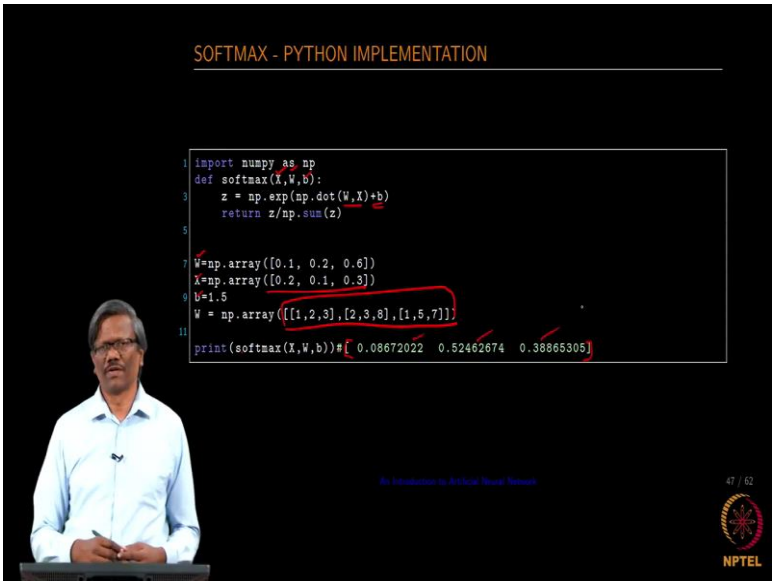
through softmax, we will have again a different set of real values and if you some all these values will be equal to 1 that we know right in the probability distribution part.

So, we are distributing the probability mass of one to all the elements, 100 elements in this vector space. And then only one of that would be should be having a higher value rest should be having a very low value, so that we can propagate the error back. So, in orders for us to do this we use this softmax and softmax picks up the values between 0 and 1, every element will have some probability and we have the entire thing as a probability distribution.

And when the 35th element is not equal to 1, but let us assume that 23rd element is it is closer to 1 that has the higher value, we know that there is an error that is not the word that I am expecting; I am expecting a word, where the 35th element should have a higher value. So, now we know that there is an error and then do the backpropagation. So, for you to do that operation we require softmax, we will I will talk about this in detail later though, but thought this is essential for you to understand, why we are bringing in the softmax.

So, this actually normalizes the net output and then the classes are really well separated. So, when you do that you know I have to a good amount of training, you will see that the 35th element is gaining more and more value in the probability distribution ok.

(Refer Slide Time: 22:33)



A simple implementation would be as follows. So, you have the input and the output and the bias and we need to find the softmax for W, when you input W, X, and bit should in, it should give you the distribution. And if you do that you will see this case, we are feeding the W as arrays and then we have some x values.

And then the values are computed using W X the dot product of W X and W and X and then it summed with the bias. You are assuming that the b is also having some weight values that I am not showing here. When you run this softmax function what you get like this, so it is a distribution now the values are distributed and you can see that the middle value has a higher probability.