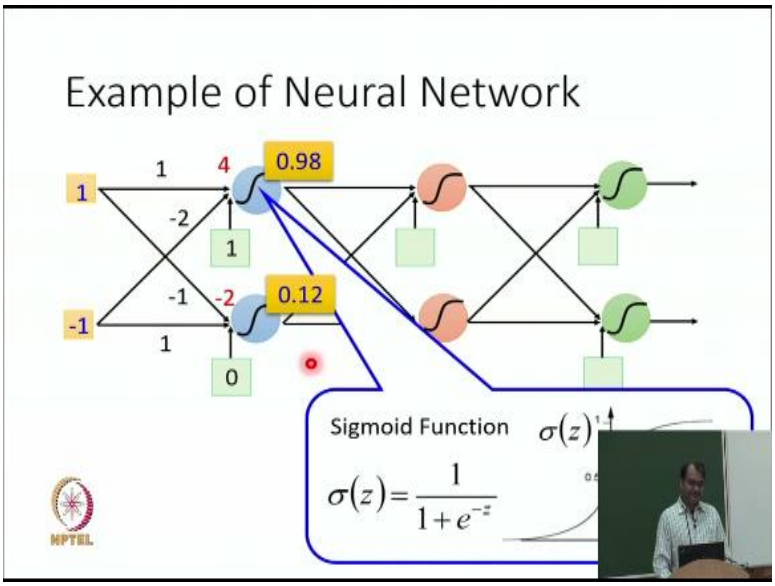
Artificial Intelligence
Prof. Mausam
Department of Computer Science and Engineering
Indian Institute of Technology-Delhi

Lecture - 86
Deep Learning: Neural Layer as Matrix Operations

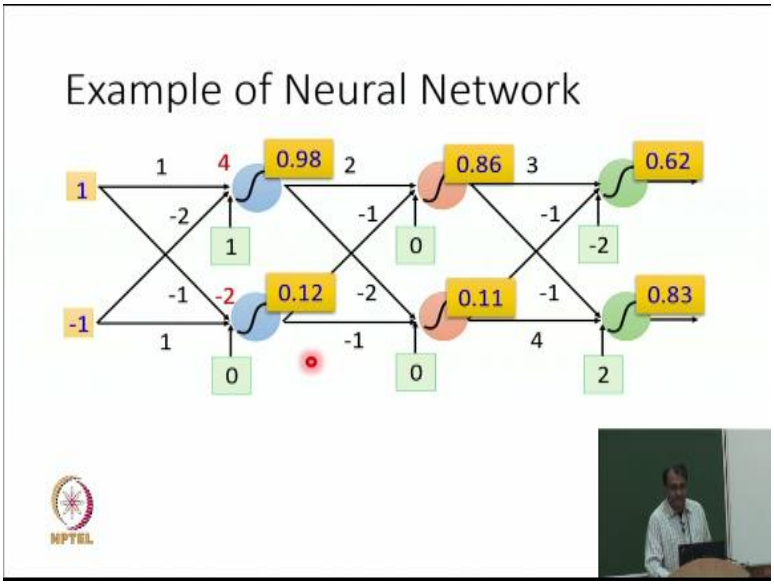
(Refer Slide Time: 00:19)



So let us do one quick example of a neural network. So we will have inputs and weights and bias terms. And this is how a neural network will be defined. So let us say my inputs are 1 and -1. So my first term would be 1 into 1 times plus -2 into -1 plus the bias term 1, right. 1 times 1 plus - 2 times -1 plus the bias term 1 so the output is 4. Then there is the activation function. That is 1 by 1 plus e to the power -z.

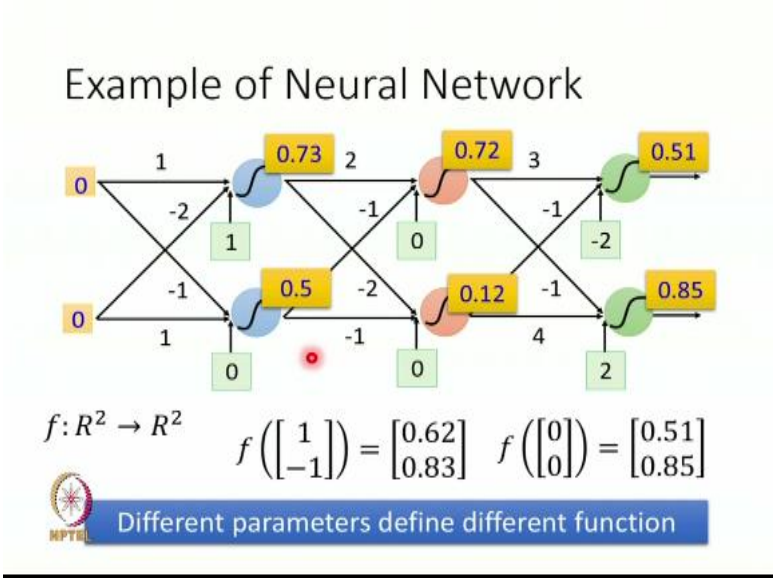
So z is 4. So that is 1 by 1 plus e to the power -4, which is whatever point I need. So the output of this particular neuron is going to be 0.98. And similarly if this is -2, then the output of this neuron is going to be 0.12.

(Refer Slide Time: 01:13)



Then you may have weights there and their activation is there so you can compute the outputs there. You can compute the outputs there. So far so good.

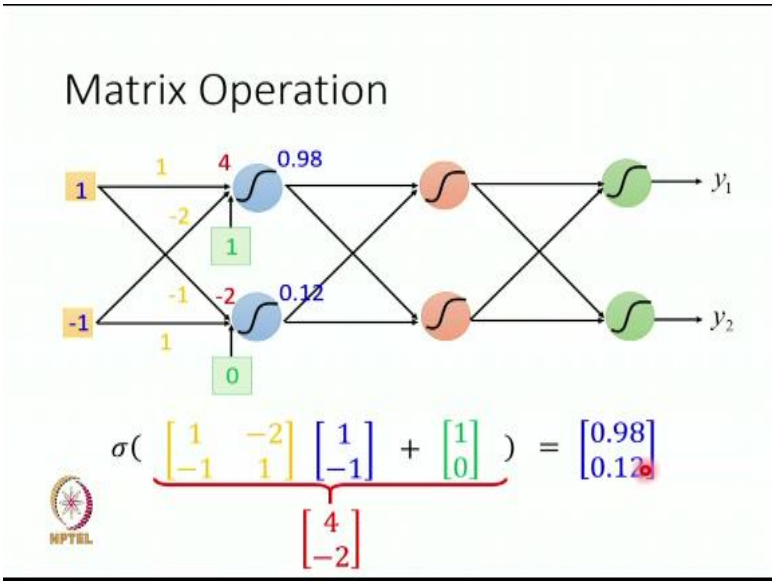
(Refer Slide Time: 01:21)



Now if I have a different set of inputs, so here I had the output 0.62, 0.83. So you can say that this function from 1, -1 is computing the output 0.62, 0.83. Similarly if I have a different input like 0, 0 let us say the function is computing 0.51 0.85. So in this very small example, tiny example, you can say it is a function from R square to R square. And the function from 1 -1 is 0.62 0.83. The function from 0, 0 is 0.51, 0.85.

And you can say that different set of parameters will define a different function. So I have a function and I have the parameters for it which are all the weights and all the bias terms.

(Refer Slide Time: 02:11)



Now I can think of it as matrix computations and this is an important step and we will stop here after this. So this is a very important step. You can say that this particular neuron is 1, -2. And this particular neuron is -1, 1. And what is the computation I am having when my input is 1 gamma -1. My input it is 1 in times 1 plus -1 times -2. Similarly, it is -1 times 1 plus 1 times -1. So basically you can think of it as matrix multiplication.

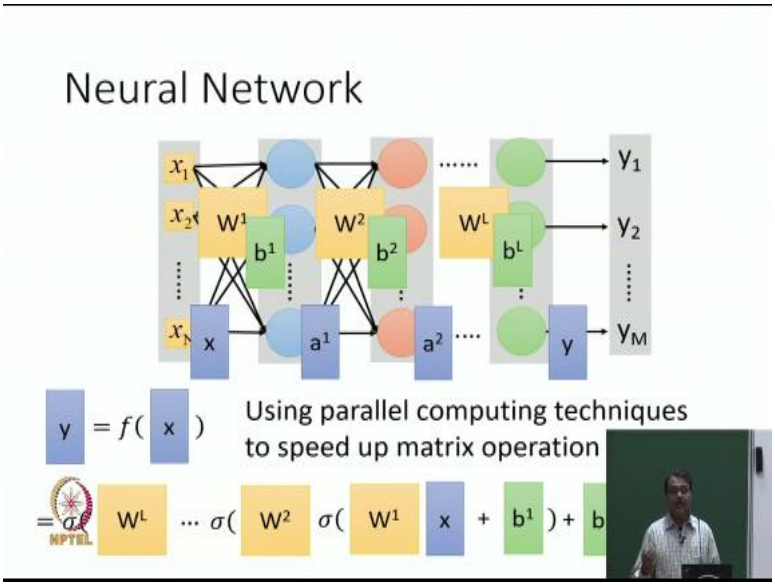
I would pause here. This is an important step because at least I get very confused with linear algebra. How many of you have taken a linear algebra course? A fair number of you. So that is good. So you will have an easier time visualizing this. But for others this is the time that we just memorize this and move on. So you can think of the first layer of neurons as a w matrix where each row represents 1 neuron.

Each row represents the weights of 1 neuron. And finally I am going to multiply it with the input 1, -1. So then if I compute the row times column, the row times column and take the sum I will exactly compute the weighted sum of inputs. But of course I have to also add the bias terms. So I will add the bias term separately. So before the activation, all the computation of a layer of neurons, not 1 but the layer of neurons can be represented as w x + b.

And it would give me an output again as the output form which is in the all the outputs of neurons put together. So this 4, -2 is the output of neurons before activation

function for the whole layer. So each layer is nothing but some matrix multiplication and a bias addition. And then I can add non-linearity on top of it which will be nothing but adding non-linearity on 4 and -2. So that will give me the output.

(Refer Slide Time: 04:46)



So technically one full layer can be represented as parameters w i w 1 b 1 or second layer w 2 b 2, ith layer w i b i and the function that is being computed is non-linearity on top of w i x + b i. And what is x? Well x is the input from the previous layer. So technically the ith layer is computing non-linearity over or sigma over w i a i minus 1 plus b i where a i minus 1 is the output of the previous layer.

And so the whole neural network is computing y equal to f x, which is basically taking x multiplying it with w i adding b 1 non-linearity. Then taking this multiplying it with w 2 adding bias b 2 adding non-linearity and so on and so on and so on. And because now we have cast the whole neural network as a sequence of matrix multiplications and additions. There is something called a graphical processing unit a GPU, which is very important in gaming.

That can speed up these matrix operations using parallel computation. And this is what led to the major speed up that deep learning algorithms achieved. Why because, because the gaming softwares, the gaming hardwares actually, not gaming software, because the gaming hardwares. NVIDIA had to speed up a lot of the physical computation, physical manipulation right.

In video games you have characters, you have gravity, they shoot. So the projectile computation has to happen and so on so forth. There is a lot of real computation that needs to happen in terms of the x y z coordinates or x y coordinates, right. And all of that typically got done using matrix computation and moreover because this you need to do this fast because gaming was a big business and is still a big business.

You know every year I have at least one student who has a big fat laptop and I ask him why do you have such a heavy laptop? And the answer is gaming. This big heavy laptop is stronger than most you know computers that you know you by which are sitting computers. They do not, not laptops desktops, and they are very powerful hardware they have.

So because these gaming requires this kind of a computation again and again and again, they said look, we cannot optimize it in the software at all. We will have to just make specialized hardware for it. And they created what is called the graphical processing unit the GPU. The idea was that it is a graphics card of some kind which will really speed up these kinds of computations.

And at the core of it they could just do matrix multiplication very fast. It was just hardware to do lot of matrix multiplications and additions. That is it. So if you can do this very fast, and if a neural network just requires these are some of these to be applied successively then you can think of the full matrix computation to be done in unit time assuming you have enough memory in your GPU.

And even if you do not have enough memory in your GPU, you can think of, you know one matrix multiplication s divided into 2, matrix multiplications or 4 matrix multiplications and so on so forth because these are independent operations because each weights each set of weights are different. You just divide the weight matrix and send one part of the weights to 1 GPU and the other part of the weights to the same GPU after a little bit of time or whatever it is.

So therefore what started happening is that this forward propagation, this computing the output of one layer did not require too many additions and multiplications in terms of the amount of time. It only required one large matrix to be multiplied in unit time.

And that really sped up the neural network computation. So earlier where neural network was somewhat slower now all these things could be done incredibly fast.

So I have my students who sometimes compare the CPU version of neural network versus the GPU version of neural network, and they can get 10 times and sometimes even more speed up on the computation. So things are much faster, at least one order of magnitude faster. But it might get even work better because GPUs have now become started to become even better and better and better.

Because the deep learning finances or the deep learning market is bigger than the gaming market. So now Nvidia has even more incentives and by the way, Nvidia was a good company, but now it has become like a great company. Like Nvidia is at the core of AI. They have not done any AI in life. Or at least in the last until the last 3, 4 years. They were a gaming company.

They were a hardware company, but now Nvidia is the middle of all this AI revolution because everybody had to use their hardware their GPUs because they were the ones who created these GPUs ahead of time because of gaming and so their GPUs got used in all this. And the first person who used those GPUs was somebody sitting in Toronto. And he recognized and this is Alex who his last name I cannot pronounce.

It starts with a Krizhevsky or something. He and his advisor Geoff Hinton but Alex was the guy who basically came up with the idea that can we not use graphical processing units to compute deep learning, right to create deep learning. And so his code could use that hardware and could train much better therefore. And therefore he got massive reductions in errors on the image data set.

That was a major challenge data set for the community and that is what sort of caused the whole revolution, right. So I mean the and Nvidia has become so central that when PM Narendra Modi was interested in learning more about artificial intelligence he did not call Geoff Hinton. He did not call Andrew Ng. He did not call the CEO of Google or Microsoft or their AI heads.

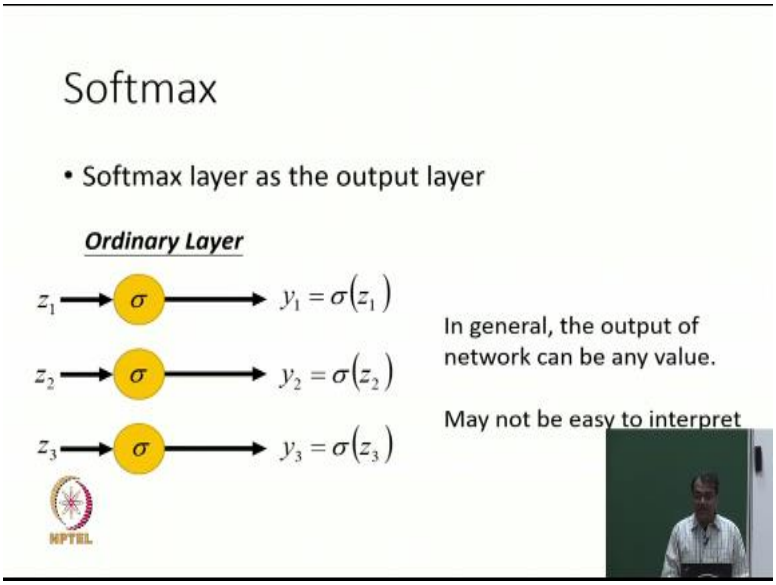
He called Jensen Huang the CEO of Nvidia to talk about AI. So and Jensen was honest. He said that he was he was nice guy. The point is he realizes who is. He also realizes he is in the middle. He is making the most of it. He is getting Nvidia from wherever it was. It is good to you know great company. All of that is awesome. And he also recognizes the importance that Nvidia plays, of the role that Nvidia plays in the context of AI okay.

So this is sort of this is a small story. But let us come back to the topic at hand. We were trying to classify an image into a digit, okay. And now we have basically created our neural network. This is the input layer. Input layer was 16 cross 16. So 256 dimensional layer input of zeros and ones and eventually we want to produce an output. And we have discussed that we want to produce an output which is a probability distribution, which is 10 dimensional or now.

Now unfortunately, if you see the way we have organized our computation these are sigmoids. They may output any number between 0 and 1. They do not have to be normalized to 1 for the different 10 outputs. But more importantly they may not be sigmoids. They may be tanh, they may be ReLU, they may be anything in the middle because we cannot be married to a certain activation function.

We want to create the general version of neural network. So what do we do? Okay. And so the general approach that most neural network designers take is that they say we will let that computation be in any general form. So it could be any number between minus infinity to infinity as the output of the neural network. We will assume that. But then at the end we will put in a specialized layer to convert that representation into the output representation, right.

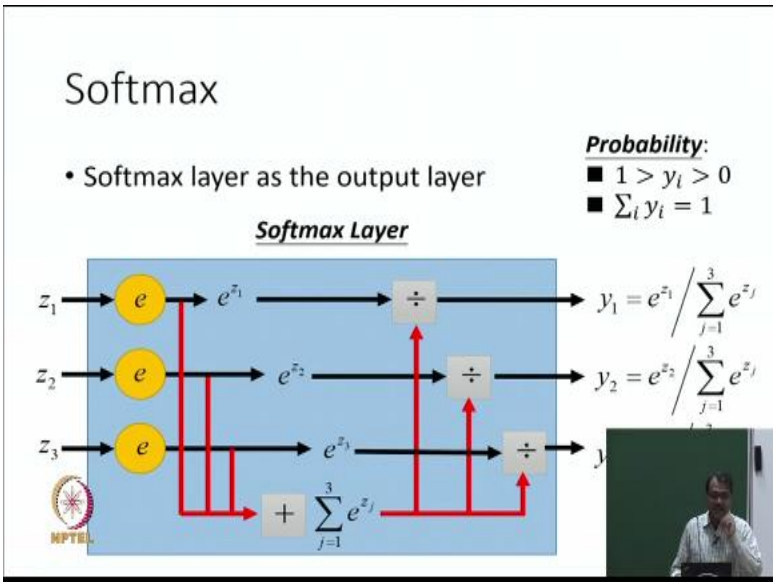
(Refer Slide Time: 13:17)



So suppose my general representation is z 1, z 2, z 3 up to z 10 or z 0 whatever and I want to compute convert the z 1 into y 1 or z 2 into y 2 and a z 3 into y 3 such that this y is the probability distribution then I have to think about this non-linearity sigma which would achieve it, right? What should be that sigma function.

So ordinarily we will put in a sigmoid and if you put in a sigmoid y 1 would be between 0 and 1, y 2 would be between 0 and 1, y 3 would be between 0 and 1 but it may not necessarily be, it may not necessarily sum to 1 and moreover z i may not be easy to interpret because they are coming from a some complex combination of the network and we do not know what is going on and why is it producing a large number?

(Refer Slide Time: 14:11)



Well, we still do not know whatever happens. But let us say we want to enforce that these probabilities sum to 1 and it is not very hard to do this. And the way we are going to do this is what is called the softmax layer and in the softmax layer, we recognize that z i's can be positive or negative. But we want to say that if it is a higher number then it should lead to a higher probability.

So what function could be used such that the number which is between which is either positive or negative becomes only positive because probabilities can only be positive. What function could we use and which also says that higher z i means higher output. What function can use? Exponent, very simple. So we will just do e to the power z. So when you do e to the power z, if z 1 was small we will get a small e to the power z 1.

If z 2 was larger you will get larger e to the power z 2. If it is negative, it will still be positive. If it is positive it is positive. Moreover we want these numbers to sum to 1. So what can we do? Normalize. So just add them and then divide, right? So send it back and divide. So that creates our y 1 and this function is called the softmax function, okay.

So whenever you see a softmax function assume that lots of inputs are coming between minus infinity to infinity and they become converted into a probability distribution of the same dimensionality. That is the job of the softmax function. And moreover you may ask why is it called a softmax function. And the answer is simple. You know we essentially want to do max. What do we want to say?

We want to say that output the digit which has the highest z 1 or highest z i. That is what we want to do. But eventually that may we can still do that. We can still actually do the max function there. There is nothing wrong with a max function, even though it is not differentiable completely we can do what is called sub gradients and we can even differentiate it. But softmax is a differentiable version.

It is a softer version of the max function. We are not simply saying that you are the max. You are saying you are point a to the max you are point to the max and so on. So that is sort of like a softmax, softer version of the max. It is still monotonic and it is

non-local because the softmax output depends on all GI’s, not just the individual z, okay.