

Medical Image Analysis
Professor Ganapathy Krishnamurthi
Department of Engineering Design
Indian Institute of Technology, Madras
Lecture 41

Feed forward neural Networks

(Refer Slide Time: 00:14)



MRL Medical Imaging &
Reconstruction Lab



Medical Image Analysis

Feed Forward Neural Networks

Hello, and welcome back. In this week, we are going to look at feed forward neural networks. So, last week, we looked at linear regression. And we will continue with the discussions. And to see the need for the motivation for neural networks, and then go on to describe the operations in the feed forward neural network.

(Refer Slide Time: 00:35)



Learning XOR

- Linear regression involves modelling output y as a linear combination of inputs X i.e., $\hat{y} = W^T X = f(X)$ $\leftarrow \text{if } X = (x_1, x_2, \dots, x_n)$
- However, this function is inadequate for a lot of real world data, one example is the exclusive OR gate which we will look at shortly 1000×1000
- A better model would be non-linear combinations of input vector $\hat{y} = W^T \phi(x)$ $\rightarrow x_1, x_2, x_1^2, x_2^2, x_1 x_2$
- One can either handcraft or learn $\phi(X)$ \rightarrow neural networks

So, we saw that linear regression, we modeled the output y , we denote the output of the model as \hat{y} . And we said $\hat{y} = w^T x = f(x)$, where W is the vector of weights and, or the parameters of the model, and the X is the input. So, X is actually again a vector, so X has its own components small, so X would be $(X=x_1, x_2, \dots, x_n)$ features. So, each one of them, these are called features, it has n features.

And we had a bunch of these training vectors along with their output. So, for each of these X 's, capital X 's there is a corresponding y , which was also given. And we use that to estimate the parameters W , unknown parameters W , through gradient descent. This is what we saw last week. But you will see that this function, this linear model, so basically, this $f(x)$, as we call it, is inadequate for a lot of the real-world data.

So, we are going to look at one example, which is basically the XOR gate or the exclusive OR gate. So, very simple toy example, and we will see a lot of things learn from a lot of things from there. So, why would I say that this function is not good for a lot of real-world data, because a lot of real-world data has a lot more features.

And sometimes the relationship between, the correlation between the features, and the outcome y or the output y is not known a lot of times. And also, the dimensionality of the data is an issue. So, for instance, let us say we have a picture, we are trying to label that picture as belonging to, let us say, a particular class. So, for instance, if you have a chest X-ray, and you want to say if it is a normal or abnormal chest X-ray, so chest X-ray, a digitized chest X-ray would be of the size 1000×1000 .

So, which gives rise to 10^6 features, like, so this is $(x_1, x_2, x_n \text{ up to } x_{10^6})$ So, with something like that having a linear model might not be a very suitable thing. And we will also see that maybe the outcomes are not exactly linear. So, better model would be a nonlinear combination of the input vector, so something like this, but we still have, this is still linear because it is still w^T , but instead of X , we have $\phi(x)$. This is some nonlinear combination of X 's.

So, for instance, if you have x_1, x_2 , so you will also have x_1, x_2, x_1^2, x_2^2 and you know other combination, $x_1^2 \cdot x_2$, so on and so forth. You can keep going like this. So, then, of course, the problem is how do we estimate $\phi(x)$, so that we can do something like what I did here is to write down a lot of possible functions, and then use that to fit the model.

And another version is if you study the problem, so for instance, we are looking, we are talking about chest X-ray. And we know now how to make $\phi(x)$. We now look at the picture and identify some aspects or some properties of that image, let us say chest X-ray, it looks very bright. We say okay, that is a feature. So, we extract these features, quote, unquote, these “hand engineered features”, that would be the $\phi(x)$.

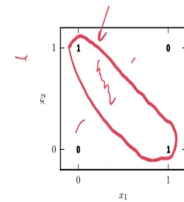
Or the size of the anomaly. So, that is the, that is what we will be looking at as far as chest X-rays are concerned. So, we can either write this down like I did here, or we can handcraft by observing your data and figuring out some functions, which will serve us features and has a good linear relationship to the output or we can learn $\phi(x)$. So, this is where neural networks come in, because they help you learn. So, neural networks here.

You can learn this by modeling $f(x)$ using a certain structure. So, basically, it is a composition of functions. We will see that later. So, by modeling this in a different way we can actually learn these features and this is where real networks make a big difference in terms of, if you compare it with other machine learning techniques.

(Refer Slide Time: 05:02)



Learning XOR

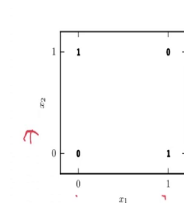


x1	x2	Y
0	0	0
0	1	1
1	0	1
1	1	0

- We cannot use a linear model to estimate y
- For $x_1=0$, the output is an increasing function of x_2
- For $x_1=1$, the output is a decreasing function of x_2
- Since there is only one weight associated with x_2 , it can't be both



Learning XOR



x1	x2	Y
0	0	0
0	1	1
1	0	1
1	1	0

- We cannot use a linear model to estimate y
- For $x_1=0$, the output is an increasing function of x_2
- For $x_1=1$, the output is a decreasing function of x_2
- Since there is only one weight associated with x_2 , it can't be both

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2$$

X

NOT Linearly separable

So, let us take a look at this XOR model, that learning XOR problem. So, XOR is an exclusive OR, it is a logic gate. So, it usually has, it has two inputs, x_1 and x_2 in a corresponding output. So, in the case of XOR, these are the only there are only four combinations possible because x_1 and x_2 are either 0 and 1. So, this is the real toy example, because there is no such thing as test data, what we are trying to do is just try to fit the training data. So, now if you look at this (0,0) will give output 0.

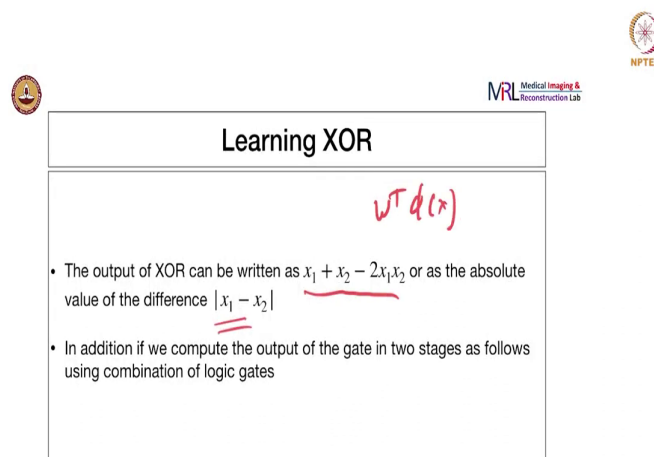
(0,1) And (1,0) will outputs 1, and (1,1) will give you output 0, which is what is plotted here. So, this horizontal axis is x_1 , and the vertical axis is x_2 . So, now we have these four points plotted. So, now what are we trying to here? So, let us treat this like a classification problem. So, the only way we could classify this is to find this curve, I am going to draw one hand

drawn curve. Let us say if you find this curve, let us say this is a curve, this region encompassed by the equation of this curve.

If you find it, then we can say everything inside this region is 1, everything outside is 0. So, I am going to wipe this out just for make this much more clear. On the other hand, if you see, we cannot draw any line, there is no line possible or plain possible which can divide this into two regions, one side is all ones, the other side is all zeros. It is not possible to draw a state. In other words, if you are doing a model $\hat{y} = w_0 + w_1x_1 + w_2x_2$ then this simply will not work. The reasons are here right there.

So, if you consider the $x_1 = 0$, then the output y is increases as x_2 increases, when $x_1 = 1$, the output y decreases as x_2 increases. So, which is not possible to be that kind of behavior cannot be captured with this, just this parameter w_2 by tweaking one parameter. So, that becomes a difficult problem to solve. So, the only way we can do that is to do that curve I showed you. So, this kind of problem is not linearly separable, it is not linearly separable. So, basically, we cannot do this model, and hope to fit this data. So, then what kind of things will work? Let us look at that further.

(Refer Slide Time: 07:47)



The slide is titled "Learning XOR" and features two logos at the top: a circular emblem on the left and the "MRL Medical Imaging & Reconstruction Lab" logo on the right. The main content area contains two bullet points. The first bullet point states: "The output of XOR can be written as $x_1 + x_2 - 2x_1x_2$ or as the absolute value of the difference $|x_1 - x_2|$ ". The expression $|x_1 - x_2|$ is underlined, and the handwritten red text $w^T \phi(x)$ is written above it. The second bullet point states: "In addition if we compute the output of the gate in two stages as follows using combination of logic gates".

So, if we look at the output of XOR, then we can actually rewrite it in this form, $x_1 + x_2 - 2x_1x_2$, where we are actually treating it like numbers, like this is an algebraic formula. You can plug in (0,1) and see that it works. We also write it in this form $|x_1 - x_2|$. You can see that both of these formulations are nonlinear in x in the features or nonlinear

features. And if we use this kind of function, then we can actually fit by using a linear model, when I say linear model I am talking something like $w^T \phi(x)$.

(Refer Slide Time: 08:31)

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2$$

↑
will fit y

$$\hat{y} = w^T \phi(x)$$

$$\begin{bmatrix} x_1 & x_2 & x_1 x_2 \end{bmatrix}$$

chest x-ray  $x_i x_j$ 10^6
 $x_i x_j x_k$




So, so, for instance, how do we do that? Let us let us look at this. So, what I mean is $\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2$, this model will fit y, because this \hat{y} will fit y. So, the truth table will be able to guarantee for. We can estimate w's, w_0 , w_1 , w_2 , w_3 for some value. We already know the value, w_0 is 0, w_1 is 1, w_2 is 1, w_3 is -2. So, this is what we are seeing. So, using this model we can fit.



So, what we have, this is the case where we have $\hat{y} = w^T \phi(x)$ where $\phi(x)$ just turns out to be this collection $[x_1, x_2 \text{ and } x_1 x_2]$. Now, once again we go back to the image example where we have chest X-rays. So, let us just do a toy chest X-ray. Now, each of these individual pixels in a chest X-ray or the x_i 's, small x_i 's. Now, in 1000×1000 image, digitized image you have 10^6 such pixels.

So, we have 10^6 pixels to begin with. And then not be able to construct imagine how many we have the combinations we can do, how many $x_i x_j$ and then we can also do $x_i x_j x_k$, so on and so forth. So, this is an explosion of features, if you do it this way, the explosion of features. And that becomes very hard to manage both in terms of memory and computation.

So, in some cases where we know such expressions are possible, and we know that it is easy to write down by hand, or maybe it is not, maybe it is only hundreds of features rather than 1000's of features, or millions of features, this is probably a very good model to use. But it will be nice if there is another way of doing this. The other way of doing this is to have more layers of computations. And let us see what we mean by that just using this example.

(Refer Slide Time: 10:41)



Learning XOR

- You can implement XOR gate as $(x_1 + x_2) \cdot (x_1 \cdot x_2)$
- The new features h_1 and h_2 can now be used to estimate the XOR

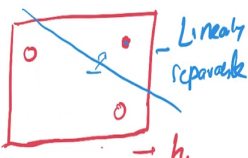
$h_1 = x_1 + x_2 \rightarrow \text{OR}$

$x_1 \cdot x_2 \rightarrow \text{AND}$

$h_2 = \overline{x_1 \cdot x_2} \rightarrow \text{NOT}$

x_1	x_2	$h_1 = (x_1 + x_2)$	$h_2 = \overline{(x_1 \cdot x_2)}$	Y
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

h_2



So, we will now look at what we mean by different layers of computations. So, you can implement an XOR gate as in this form as $(x_1 + x_2) \rightarrow$ OR gate, $(x_1 \cdot x_2) \rightarrow$ AND gate, and $(x_1 \cdot x_2) \rightarrow$ NOT gate you can think of. So, we are now computing two intermediate features, I am calling them h_1 and h_2 , call this h_1 and this is h_2 .

And when we compute these two intermediate features, we will see now upon computation of these intermediate features, the problem becomes linearly separable. So, I will just draw that in there and then so that you can see. And then we will also look at these computations in a different way, like in the form of a network of computations with different layers of computation, and then we will see why we call it intermediate computations, etc.

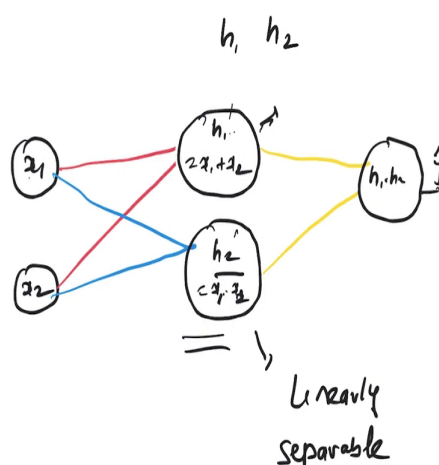
So, if we plot this image, we plot this, not the image I mean the graph of, in this case, this is h_1 , this is h_2 . So, if you look these two points (0,1) and (1,0) for x_1 and x_2 respectively, has now collapsed to (1,1). So, if h_1 is 1 and h_2 is 1, the output is 1. This is all I am going to indicate with the solid dot. And if you look at the other options you have, for (0,0) you get 0. I am going to put (0,0) here, I could have put it at the corner, but let us just put (0,0) here.

And if I have let us say, this is for h_1 and h_2 , so (0,1) is also 0. So, I think I made a mistake, sorry. Let me just wipe this out. I meant, so we are looking at h_1 and h_2 . So, what I have plotted here is (1,1) is 1, that is fine. And if you look at (0,1), it is also 0, so (0 1), 0 is one type of output. And if you look at (1,0) that is also 0. So, now it is linearly separable, because you can always draw this line.

Of course, there is a whole bunch of lines you can draw here. So, that is another issue there, but we can have a line which will split this into two regions depending on the value of h_1 and h_2 which will give you the correct answer. So, if your h_1 is 1 and h_2 is 1 it will fall automatically in this line this side. If for all other values it will be there, thought that it is able to do that, because by doing this intermediate computation, I have collapsed both (0,1) and (1, 0) into the same numbers, (1,1) and (1,1).

So, these two got collapse to one point. So, then I cannot draw this line, let us split that. Of course, now there is the problem is because you can draw like a bunch of lines which will do that, but now the problem is linearly separable. Now, what do we mean by intermediate computations? What do you mean by feed forward layers of computations? That is what we will look at in the next.

(Refer Slide Time: 14:20)



So, now we will look at what I meant by an intermediate layer of calculations. So, we saw that in the previous slide, where if we split it into a set of two different Boolean operations, then we got the problem to become linearly separable in terms of the new variables we calculated. We calculated these new variables h_1 , Boolean variables h_1 and h_2 . So, I am just going to illustrate that in terms of graphs so that now we can make then, we can go on to make connections about how neural networks work.



Remember, all the operations I am showing here are actually Boolean operations not algebraic operations. So, Boolean operations. So, let us look at x_1 is an input, x_2 is an input and we connect these to another computation. We call that the h_1 which is $(x_1 + x_2)$. Then we


do another connection to other computational unit where you compute h_2 which is $(x_1 \cdot x_2)$. Of course, you can always take these two output from here and connect it to one more box where we do h_1 and h_2 and that output is your \hat{y} .

So, this intermediate layer of computation is what enabled our problem to become linearly separable, it became linearly separable in h_1 and h_2 , these new features we call them new computed features. So, the two ways we looked at one is to incorporate this non-linearity or nonlinear combinations of your input features. So, we got the x_1 x_2 , $(x_1 \cdot x_2)$ or we did this computation in two stages.

So, when you did that it actually made the problem a little bit more easier to solve. So, because when you did this h_1 and h_2 we saw that the problem becomes linearly separable in h_1 and h_2 . If we start with x_1 and x_2 compute h_1 and h_2 , in h_1 h_2 space the problem was linearly separable, here it is linearly separable.

(Refer Slide Time: 17:07)



Learning XOR
Feed Forward Neural Networks

MLP

- Feed Forward Neural Network or Multi Layer perceptrons, perform a sequence of computations on inputs and maps to an output, hence the name feedforward (and also no feedback from any intermediate computations)
- They are a composition of different functions, each function defines a 'layer' of computations
- For Example : $\rightarrow f^{(2)} \left(f^{(1)} \left(f^{(0)}(x) \right) \right)$ *INPUT*
Hidden
OUTPUT layer

So, now we will get to the point where we are actually talking about feed forward neural networks. So, we looked at in the previous example, where I showed the XOR gate if we have one additional layer of Boolean operations of computations then we can actually make the problem linearly separable in terms of the outputs of the intermediate computations. So, that is exactly what a feed forward neural network does.

That it does layers of computations and we will see what those layers mean. So, feed forward neural networks or multi-layer perceptrons they are called MLP's I think you must have come

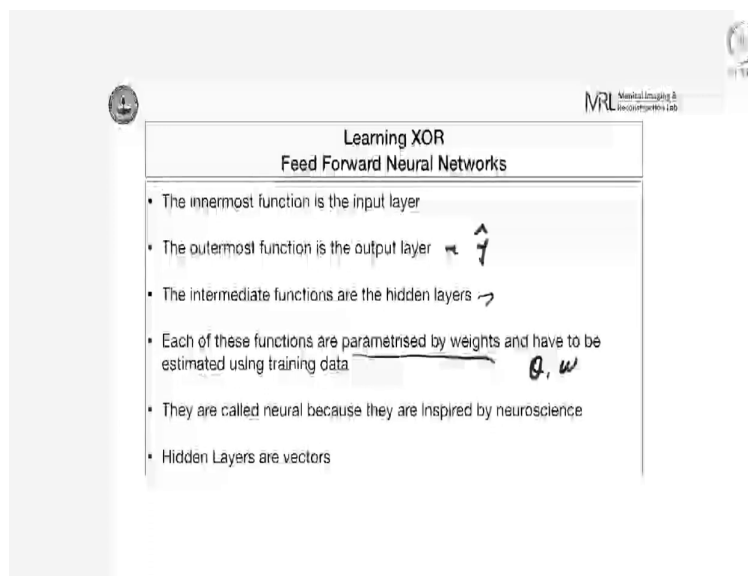
across this acronym somewhere. They perform a sequence of computations on inputs and eventually map to an output. Typically, we will only look at scalar outputs. So, it is, of course, it is because it is a sequence of computations and there is no feedback it is called a feed forward network.

And this is accomplished using the composition of different functions. And each function defines a layer of computations. Of course, we will typically write it in this form, so, you will have something like your input is $(f^{(1)}(x))$ and then this will be input, the output of this $(f^{(1)}(x))$ will be given to another function $f^{(2)}$. And the output of this $f^{(2)}$ can be given to another function $f^{(3)}$, so on and so forth $f^{(3)}[f^{(2)}(f^{(1)}(x))]$. So, this is actually a composition of functions.

So, this will be the input layer is called, the first layer will be called the input layer, the second layer or the intermediate layers are called the hidden layer, so input, hidden. And the outermost function you see there is your output layer. So, this is a composition that functions that we apply, and each function is a layer. So, we will see how it is represented soon. But this is typically how you interpret a neural network.

But of course, the computations you will do will be in sequence and it will be done on discrete inputs. So, all computations are done on discrete inputs, even though I am saying x here is teams, they are continuous variable, x is a discrete input, it is a vector of elements typically. So, the inputs are always discrete, and the outputs of these functions also lead to discrete variables, we will see how they come about shortly.

(Refer Slide Time: 19:41)



The slide is titled "Learning XOR Feed Forward Neural Networks". It contains a list of bullet points with handwritten annotations:

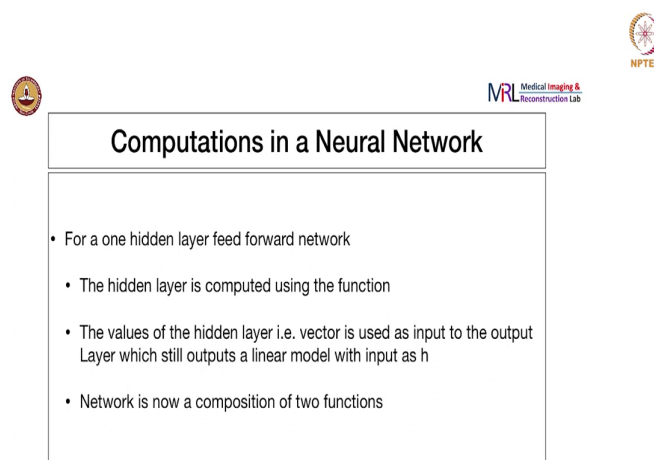
- The innermost function is the input layer
- The outermost function is the output layer \hat{y}
- The intermediate functions are the hidden layers \rightarrow
- Each of these functions are parametrised by weights and have to be estimated using training data θ, w
- They are called neural because they are inspired by neuroscience
- Hidden Layers are vectors

So, to summarize, the innermost function is the input layer. The outermost is the output layer which produces your output, which is \hat{y} whatever this might be, we will see what are the different types of outputs possible. The intermediate layers are also called the hidden layers or activations, we will see why they are called activations. So, each of these layers or we can take them as a function they are parametrized by weights w .

And sometimes we use w 's or θ either was fine. And they are usually estimated like we did for a linear regression they are actually also estimated using training data. The reason why we called the neural networks is because they are inspired by neuroscience wherein the sequence of computations where we go from one layer to the other starting from an input layer to an output layer is how the neurons in the brain are connected.

So, it is inspired by that but there are no means a very good model of how the neurons work. Also remember the so-called hidden layer even though I say it is a function, they are actually vectors. So, we will see how that works out. And why they are vectors using a few examples.

(Refer Slide Time: 20:51)



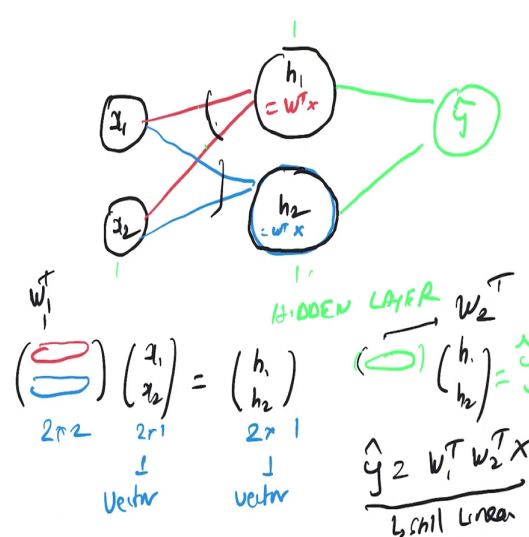
The slide is titled "Computations in a Neural Network". It features logos for IIT Bombay, MRL (Medical Imaging & Reconstruction Lab), and NPTEL. The main content is a bulleted list:

- For a one hidden layer feed forward network
 - The hidden layer is computed using the function
 - The values of the hidden layer i.e. vector is used as input to the output Layer which still outputs a linear model with input as h
 - Network is now a composition of two functions

So, for a one hidden layer feed forward network. The hidden layer is computed using the function we will see what that function is. And the values of the hidden layer which is basically another vector is used as input to the output layer, which eventually outputs only a linear function of the h , of the hidden layer. And we treat that as the output and then we do all sorts of training using that. Now, this kind of network is a composition of two functions.

So, let me illustrate that. So, what we saw especially for the XOR, exclusive OR function, in terms of the OR gate and the NAND gate is basically a one layer you can think of like a one-layer neural network, but we will see what these eventually translate to for in terms of algebraic operation. So, let me just illustrate this a small example here.

(Refer Slide Time: 21:44)



So, for instance, your inputs are still x_1 and x_2 . So, then you have these connections which go to another computation, calculate h_1 from there you have a set which goes to another computation, we can use black actually just to be consistent. So, we will do h_2 . So, let us and we can of course always write down the output, this is the one-layer network. So, then, take this and we calculate \hat{y} .

Now, what is the relationship between x_1 , x_2 , h_1 , and h_2 ? So, this is the hidden layer. And why do we interpret this as a function we will see. So, the way to calculate this hidden layer in a feed forward neural network is the following. We will first do just a linear combination, then we will see why it does not work, and then we will go back, and we will include the non-linearity and we will be able to understand.

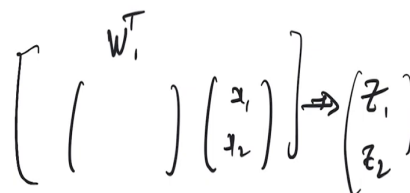
Now, this red line, each of these red lines refer to a weight. So, I am going to write the formula here. This $h_1 = w^T x$, this w is a vector. And this is $h_2 = W^T x$. But this blue W and red w are vectors. So, these are weights, each of them unique. One set of weights, which multiply x_1 and x_2 , in order to calculate h_1 , this blues are another unique set of weights, which multiply x_1 and x_2 and to get h_2 .

So, then we can write h_1, h_2 as. So, you have a red, I am going to write this is a red column and a blue column of it. So, it is a 2×2 matrix, this is a 2×1 vector, which gives you a 2×1 . So, that is why we have the input to every layer is a vector, output is also a vector, except for the last layer, which typically has, well, sometimes it is a vector, but in the case of regression it is going to be a scalar.

Now, what about this layer here? So, here too once again, I am going to multiply, pre-multiply in this case h_1, h_2 , and I have a w which is basically two weights. So, I am going to have a couple of weights here, and that will give you my \hat{y} . So, we will call this so what I will do is I am going to call this first layer matrix, this as W_1 , the second layer matrix here I am going to call W_2 , or $W_1^T W_2^T$.

So, basically, your output $\hat{y} = W_1^T W_2^T X$. Now, if you think about it, it is still a linear model, still linear. Because W_1 and W_2 , I can write it as another matrix, does not help. We are still in the linear regime. So, typically, so, what happens in that in a neural network is you would do a point wise non-linearity.

(Refer Slide Time: 25:52)



pointwise non-linear function

$$W_1^T x \rightarrow z \rightarrow g(z)$$

$$[W_1^T x + c] \rightarrow z \rightarrow g(z)$$

$$= g(W_1^T x + c)$$

So, we will see what that is, immediately. I am not going to, let me just produce the figure, but what I will do is. So, typically, what you will do you have this matrix W_1^T your x 's, this will give you a vector z , so this will give you z_1, z_2 , it is the output if you just do that. And you apply a pointwise nonlinear function to this. What would be a good, what the nonlinear function we will see.

So, basically, this will go from $W_1^T X \rightarrow z$, then to z you will apply $g(z)$, some function. Once again, I am kind of not indicating the bias term. So, because this is still $W_1^T X$ is still linear. So, we can always add a bias term. So, $W_1^T X + C$. We are using one bias term for all the weights, that is your z . So, you would apply the non-linearity to the whole thing, so to $g(z)$, so which is nothing but $g(W_1^T X + C)$.

Similarly, at the output layer, you would also do that from h_1, h_2 you might or might not want to do the non-linearity depending on what you are trying to estimate. Output layer is still a linear layer. Because you had, for instance, if you go back, so you had h_1, h_2 . Let me add one more here.

(Refer Slide Time: 27:38)

$$\begin{pmatrix} h_1 \\ h_2 \end{pmatrix} \rightarrow W_2^T h \rightarrow \hat{y}$$

$$g(\hat{y}) \rightarrow \hat{y}$$



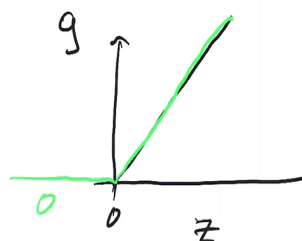
$$\begin{pmatrix} h_1 \\ h_2 \end{pmatrix} \rightarrow W_2^T h \rightarrow \hat{z}$$

$$g(\hat{z}) \rightarrow \hat{y}$$

$$g(h_1)$$

$$g(h_2)$$

$$\max(a, z)$$



So, the last array output layer we had h_1, h_2 . So, you can actually just have some $W_2^T h \rightarrow \hat{y}$. You can apply your non-linearity $g(\hat{y})$ eventually give you the \hat{y} . So, I will not call this \hat{y} . So, for instance, let me call a different name. So, that is not confusing. So, this $W_2^T h \rightarrow \hat{z}$, then you would apply this $g(\hat{z}) \rightarrow \hat{y}$, can be done.

So, what is this g function, it is like I said, you have to apply it point by point. So, for every h , so h_1 , so you would calculate $g(h_1), g(h_2)$. That is why you would do that in this case. So, in the previous case, your output would come from every z_1 and z_2 , we saw that. So, what would be this idea of non-linearity, the one that is often used these days is called the ReLU. ReLU is something, so if you have an input, in this case, I am just going to call this z for positive values of z , it is just identical to z , 45-degree line, so I am going to use blue, so that green, so that is easy.

For everything less than or equal to 0 you can 0. So, you basically it is a function which reads $\max(0, z)$. So, this function, so, this is a non-linearity because it zeros out all negative values and keeps only the positive values. So, this is one of the simplest nonlinearities used apparently works really well for a lot of the problems that come about these days as applications of neural networks.

(Refer Slide Time: 29:35)



Computations in a Neural Network

- The different Layers cannot be just a linear layer because then they can be combined into one linear model
- A non-linearity is required to extract features. ✓
- A linear combination followed by a point wise non-linearity is typically used in the hidden ~~lanes~~ ^{layers}
- By point wise - non-linearity is applied to every element of the hidden layer vector
- The values after applying the non-linearity is also referred to as hidden activations ✓

So, just to recap, the different layers in a feed forward neural network cannot be just nonlinear layers, because then they can be combined into a one big linear model. The non-linearity is required, so that the problem becomes linearly separable or we can get extract features which make it linearly separable.

So, a linear combination followed by a pointwise non-linearity is what is typically used in the hidden layers, sorry not lanes. So, by pointwise non-linearity, we mean non-linearity is applied to every element of the hidden layer vector. The values that you get after applying the non-linearity to the find transform that is z is also referred to as hidden activations.

(Refer Slide Time: 30:25)

$$\begin{aligned}
 & \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \rightarrow \mathbf{w}_1^T \mathbf{x} + c \\
 & \begin{bmatrix} \text{ } \end{bmatrix}_{p \times n} \begin{bmatrix} \text{ } \end{bmatrix}_{n \times 1} = \begin{bmatrix} \text{ } \end{bmatrix}_{p \times 1} \\
 & \text{hidden layer } z \\
 & (\mathbf{w}_2^T \mathbf{h} + c) \rightarrow \left(\frac{1}{1 + e^{-z}} - y \right)^2
 \end{aligned}$$

Here is how we summarize the computations in a neural network. Let us say we are looking at one hidden layer, how can we summarize its computations. So, we have an input X , I will use blue, X is some vector. So, at the input layer, we are going to multiply this X with a weight matrix. So, some $W^T \cdot X$ is what we are going to use. The dimensions of W depend on what kind of what the dimensionality of the output that you want.

So, this is a vector of dimension n . So, we will be multiplying it with the matrix, so this X has dimensions $n \times 1$. So, we are multiplying with W^T . So, W^T should have dimension, let us say $p \times n$, so that your output will be of dimension $p \times 1$. So, this output again this is the hidden layer that we want. Hidden layer, the first step in the hidden layer, we call this z vector.

Of course, to each one of these elements here we will apply the non-linearity that is the function, nonlinear function we saw as ReLU and that will eventually give you the hidden layer activations. So, each of these will have, so we p rows and n columns. So, you would take one row and multiply it here. So, an affine combinations of each time with different set of weights will give you as many a hidden layer outputs.

So, then we will take this capital W , I am going to call this $W_1^T \cdot X$, so you will get this hidden layer h . Of course, I have left out the, always leave out the bias term, feed free to add it as b you usually have. So, in order to do that, you will actually have a one here and then the leftmost column, you will have the W_0 , the biased term. But anyway, so, in this case let us say $W^T \cdot X + b$.

Then once you get the z 's, z made z vector. You apply point wise non-linearity and which will eventually give you your h . So, now, you will take h multiplied with the second layer of weights plus C and you will produce $(W_2^T \cdot h + C) \rightarrow \hat{y}$. Now, in this case depending on the problem you can have different kinds of outputs.

So, once again the dimensions of W_2 will depend on what kind of outputs you are looking for in the sense, for instance, what is dimensionality of output. So, for instance, if you are doing regression problem using a feed forward network, it is just one. So, then you can adjust the output of size of W_2 accordingly. You will have create as many weights. So, this tells you that these weights, the number of weights that you have in every layer is the hyper parameter, so is the number of hidden units.

So, each of these elements of the hidden vector that you calculate is a hidden unit. So, the number of hidden units in a neural network is also a hyper parameter, something that you have to decide based on the problem. And this number can be greater than or smaller than n . So, hidden unit or your input unit layer has n units for n , it is a vector of length n then your hidden unit can be either smaller than number of elements or higher number of elements, that depends on the problem.

So, that is like a hyper parameter that we have to tune in order to solve this problem. Similarly, depending on the size of your output, you will have as many, in this case, if you want to think of it in terms how we did this, we will have as many rows in this W_2 matrix as there are outputs. So, this is the typical construction. And what we do not know is or the weights.

So, just to understand problem, we do not know what W_1 and W_2 are, we would initialize them randomly, and then estimate them using gradient descent, because we have for every \hat{y} we have the corresponding y and we can use formulate an appropriate loss function. So, for feed forward neural network, there are two types of problems. One is the regression and in the classification form.

So, for the regression problem, we will be outputting the target value y , we call this \hat{y} , and then we can do a least squares loss. Now, the advantage behind this, if you look at the neural network, you will see that, you can actually output whatever value you want. Now, output whatever value you want, you just have to assign that meaning.

In my case I output y , which is \hat{y} , which I think, which I say that is the same as the target value y for linear regression. But it could be some other parameter also from which I further calculate \hat{y} . So, that those things are possible with the feed forward neural network.

(Refer Slide Time: 35:47)



Output Units

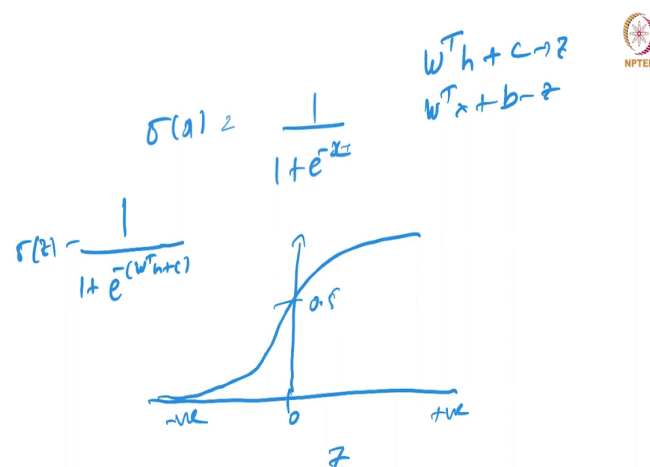
- For classification, the output should be a probability value
- Parameter of the Bernoulli distribution should lie in the interval $[0,1]$
- A sigmoid function is typically used

$$\sigma(W_2^T h + c) \in [0,1] \quad W_2^T h \rightarrow g$$

So, for classification problem, the output should be a probability value. So, if you are doing a two-class classification, then the output should be the parameter of the Bernoulli distribution. So, we still call it p , but I will call it let us say a v in this case, or μ also is fine sometimes. And what do we do, so because when we do W transpose, let us say $W_2^T \cdot h \rightarrow \hat{y}$, and we get one output, it is a scalar, and we do not know what to do with it.

So, the best way is to squash it. So, people usually use this sigmoid function, the $\sigma(W_2^T \cdot h + C)$. And this maps it the interval 0 to 1. What is this sigmoid function? Sigmoid function, we will look at very briefly.

(Refer Slide Time: 36:40)

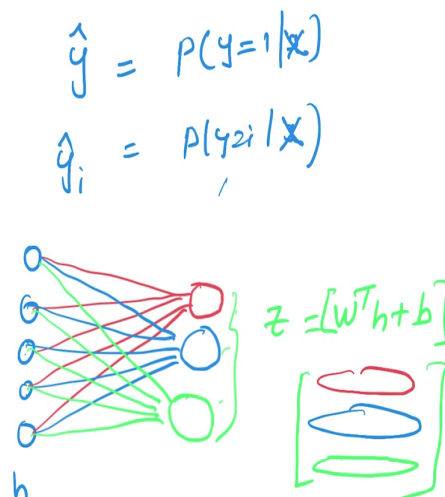


The sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$. So, all it does is it will map your values between 0 and 1. So, if you want to plot it, it is basically looking something like this. So, 0, so negative values, this is positive values, say this is z . Intermediate the affine transform that is basically the $W_2^T \cdot h + C \rightarrow z$ or $W_2^T \cdot h + b \rightarrow z$.

This is the linear combination of your inputs or the input or any of the layers. So, now you can z . If you plot this sigmoid, it will be something like for. So, this is at 0.5, at 0, if you 0, you get 0.5. So, for negative values, you get output less than 0.5, for positive values you get greater than 0.5. So, of course, you can tweak that by the slope and all that by adding some fudge factor here, multiplying some fudge factor here, but we will not get that into that issue.

So, we will be for the output layer, we will do $\frac{1}{1+\exp(-(W_2^T \cdot h + C))}$, this is what you would calculate. This is a sigmoid function that we will calculate. And this will, this is what we will use to convert our output to a probability value between 0 and 1. So, this is the case for two class classification. So, then, what do we do if we have.

(Refer Slide Time: 38:37)



So, what we did for two class problem is we produce this output \hat{y} , which is nothing but the probability $\hat{y} = p(y = 1/X)$. So, this is for two class classification. So, for k classes. So, if for k classes, we need this probability $\hat{y}_i = p(y = i/X)$ it belongs to the i -th class. Basically, this is what we want to figure out, given your input vector. So, I am using small and large interchangeably, I will just use X .

So, we need, so we need to produce as many outputs as there are. So, if you have k classes, we are going to be evaluating these probabilities. So, how do we do that. So, we know that from the hidden layer, we have so many hidden vectors, the size of the vector and the hidden layer h , then we have weights, let us say we are doing three classes, just want to show what we have to do.

We are doing three classes, let us say 1 2 3 4 5, we have five components in the hidden layer vector, then let us see, if you want to do three different outcomes. We are looking at three class classification problem. So, let us just do this red. So, this will give you 1. So, similarly, we have, give you 2, and for the green. So, you have 3 and this again before we apply the non-linearity you have $z = [W_2^T \cdot h + b]$.

So, of course, the W transpose is composed of, if you look at it in terms of a matrix is composed of these three colors. Here, the middle one, three rows. And I have not specified the length of the h vector, so it is five, so you will have each of them will have five components. So, that so this will give you three outputs, this is z and it is unnormalized.

So, then, we have to convert that into a probability. Because usually, the outcome is only one of them. So, they are independent. So, if one, there is only one, if one of the classes is probable, then the other two are not. So, in that case, then the probability should add up to 1. And of course, they should all be in the range 0 to 1 and they should all add up to 1. So, in this scenario, you will have this known as the softmax.

(Refer Slide Time: 41:53)

$$z_2 = W^T h + b = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$$

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^K \exp(z_j)}$$

$$= \frac{\exp(z_1)}{\exp(z_1) + \exp(z_2) + \exp(z_3)}$$



So, if you write this $z = W_2^T \cdot h + b$ then you are, convert that into a probability you will do this $\text{softmax}(z) = \frac{\exp(z_i)}{\sum \exp(z_j)}$ for every one of them. So, there are k classes, $j = 1$ to, let us say capital k , $\exp(z_j)$. z_i where does that come from? So, if you have three classes, when you do $z = W_2^T \cdot h + b$, you will have z_1, z_2, z_3 here. So, then you will have.

So, just to illustrate for one, the probability associated with class one would be $\frac{\exp(z_1)}{\exp(z_1) + \exp(z_2) + \exp(z_3)}$. For class 2 you have to change it to 2, 3, so on. So, of course, you have to fix what each of these classes are, when you write the program, you say, first element corresponds to class 1, second element corresponds to class 2, third element corresponds to class 3.

So, that is output, and that is how we interpret the output. So, if you want probability for one class, for two class problem, you use the sigmoid. And you interpret the output as the Bernoulli parameter which indicate the probability yes or no. And if you have a multi class problem, where of course, the choice is 1 of k outcomes, 1 of k classes, let us say one of 3 classes, in our case, which we looked at, then you calculate the softmax function using z , using this formula right here, and treat those as the probability of the outcome.

So, so far, what you had looked at is the so-called forward pass through a fully connected neural network. We also, not defined fully connected yet, but we will get there. So, what we have done is taken a vector of inputs, so each input data point is a vector. And then we multiplied it by a matrix of weights to get this linear transformation of the inputs.

And then that is also another vector, then we applied a nonlinearity, a point wise non-linearity each one of them to get the so-called hidden layer, or activation, or hidden activations. We can do that sequentially as many times as you want. Finally, we estimate an output, the output can be either just a linear combination of the last hidden layer, which we call the output layer, where we or the penultimate layer from which we estimate this \hat{y} , which can be a real number if you are doing a regression problem.

Or it can be interpreted as apply a sigmoid function and convert it into a Bernoulli parameter. If you are doing a two-class problem that is yes or no kind of problem, or use the softmax function to convert that into a vector of probabilities, all of them adding up to 1 if you are looking at a 1 of k classification. So, that is the idea behind the feed forward neural network.