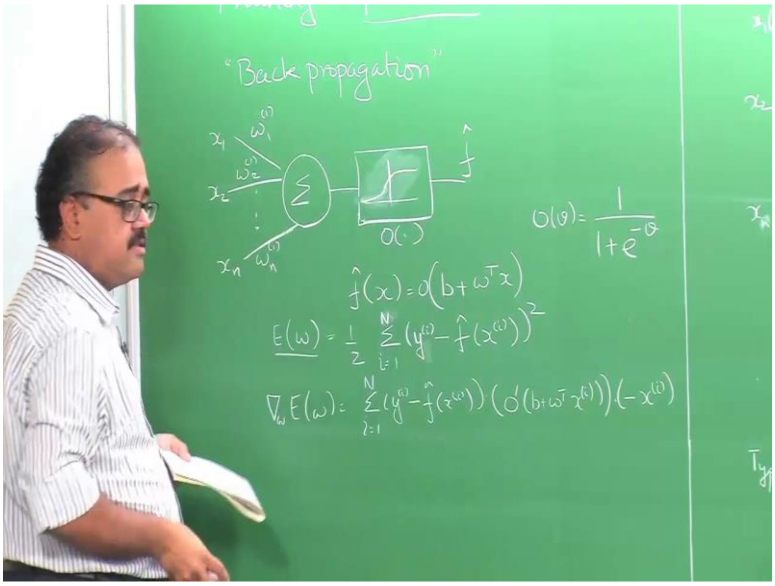
Introduction to Data Analytics
Prof. Nandan Sudarsanam and
Prof. B. Ravindran
Department of Management Studies and
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 06
Lecture - 35
Neural Networks

Hello and welcome to the module on back propagation or how you are going to train artificial neural networks and determine the weights.

(Refer Slide Time: 00:13)

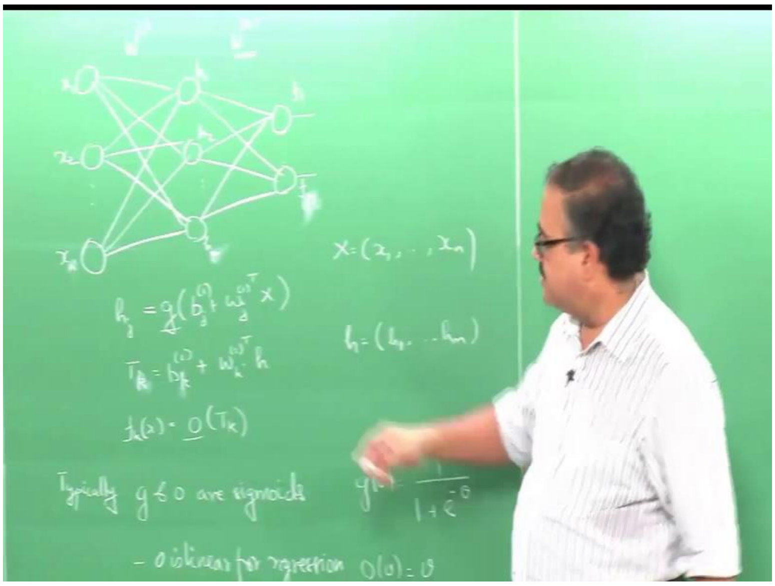

"Back propagation"
x_1
x_2
x_n
$\omega_1^{(1)}$
$\omega_2^{(1)}$
$\omega_n^{(1)}$
Σ
$\hat{f}$
$o(\cdot)$
$o(v) = \frac{1}{1+e^{-v}}$
$\hat{f}(x) = o(b+\omega^T x)$
$E(\omega) = \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \hat{f}(x^{(i)}))^2$
$\nabla_\omega E(\omega) = \sum_{i=1}^{N}(y^{(i)} - \hat{f}(x^{(i)}))(o'(b+\omega^T x^{(i)}))(-x^{(i)})$

So, to keep things simple, let us start off with a single neuron and that is going to have a logistic sigmoid as the output function. So, you are going to write here $\hat{f}(x)$ is = b + w^Tx and then, you pass that through your sigmoid function o. So, o in this case would be the logistic sigmoid, where we will say o(v) = $\frac{1}{1+ e^{-v}}$. So, we saw this in the last module, so the error measure that we will be using is the squared error.

So, , the excepted error w is going to be summation i = 1 to n, where n is the number of training data points that you have of the squared error for each training data point. So, the way we are going to use this for changing the weights is essentially to compute the gradient of the error. So, essentially that will be the error * the derivative of the output function * (- x_i).

It is essentially taking the derivative of this with respect to the w function. Once, you have computed the gradient of the error with respect to the weights, then we essentially just change the weights in the direction opposite to the η times the gradient of this weights, where η is the essentially the step says parameter. So, this was fine when you have a single neuron. So, what about the case when you have layered networks like this?

(Refer Slide Time: 02:39)

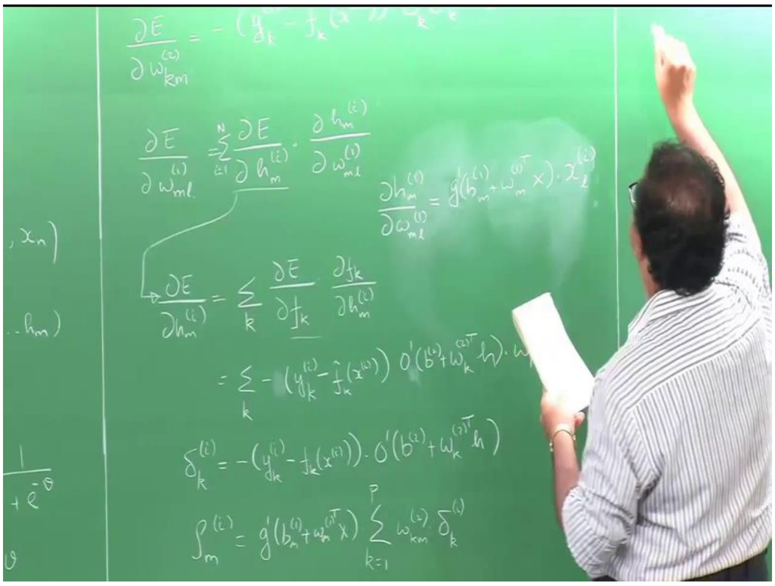

x_1
x_2
x_n
h_1
h_2
h_m
f_1
f_k
$X=(x_1, \dots, x_n)$
$h_j = g(b_j^{(1)} + w_j^{(1)T} x)$
$h=(h_1, \dots h_m)$
$T_k = b_k^{(2)} + w_k^{(2)T} h$
$f_k(x) = \underline{o}(T_k)$
Typically g & o are sigmoids
$g(v) = \frac{1}{1+e^{-v}}$
– o is linear for regression $o(v) = v$

So, this is our standard three layer neural network. So, the first layer is essentially just the inputs. So, the second layer of neurons takes in the inputs and computes the output, these are called the hidden neurons that is we discussed in the last module and then the output layers finally, take the output from the hidden units, again do the appropriate transformation and give you the final outputs. So, here we will denote the hidden layer outputs by h and h is given us g of the weighted summation of the inputs and let me introduce the temporary variable here called t, which essentially is the summation of the outputs of the hidden units and f is finally, the transformation o of the outputs of the hidden units.

So, one thing to note here is that, so I have used different functions g and o for the hidden layer and the output layer and typically for two class classification problems both g and o are logistic sigmoid as we saw earlier. So, g as well as o would be of the same form $\frac{1}{1+ e^{v}}$, but then if you are having a regression problem you can essentially use the same setup that we have here, except that o would be linear for the regression problems, so in which case o would be essentially just passing on the inputs that it is getting.

So, suppose I have this multiple layers, how do I go about finding the weights of this standard three layer output? So, in some sense finding training rule for w_2 is not very hard. So, w_2 if you think about it, it is just like a single neuron network. So, I can just take all the weights that come to f_1 and then, essentially use the same rule that I used earlier here for a single neuron. I can use the same rule for training f_1, except that instead of x_i I will be using the output h, so that is clear. So, for finding w_2 I really can just stick with what I did earlier.

(Refer Slide Time: 05:15)



$\frac{\partial E}{\partial w^{(2)}_{km}} = -(y_k - f_k(x$
$\frac{\partial E}{\partial w^{(1)}_{ml}} = \sum_{i=1}^{N} \frac{\partial E}{\partial h^{(i)}_m} \cdot \frac{\partial h^{(i)}_m}{\partial w^{(1)}_{ml}}$
$\frac{\partial h^{(i)}_m}{\partial w^{(1)}_{ml}} = g'(b^{(1)}_m + w^{(1)T}_m x) \cdot x^{(i)}_l$
$\frac{\partial E}{\partial h^{(i)}_m} = \sum_k \frac{\partial E}{\partial f_k} \frac{\partial f_k}{\partial h^{(i)}_m}$
$= \sum_k -(y^{(i)}_k - \hat{f}_k(x^{(i)}))\, o'(b^{(2)} + w^{(2)T}_k h) \cdot w$
$\delta^{(i)}_k = -(y^{(i)}_k - f_k(x^{(i)})) \cdot o'(b^{(2)} + w^{(2)T}_k h)$
$\delta^{(i)}_m = g'(b^{(1)}_m + w^{(1)T}_m x) \sum_{k=1}^{p} w^{(2)}_{km} \delta^{(i)}_k$
$, x_n)$
$\ldots h_m)$
$\frac{1}{+e^{-\theta}}$

So, I am going to write that here again just for clarity sake. So, I have rewritten this in an appropriate fraction for working with this multi layer networks. So, I am going to look at the gradient of the error with respect to a single weight that runs from the m^th^ neuron here to the kth output neuron, this weight runs from the mth hidden neuron to the kth output neuron. So, this is essentially the kth output - the prediction given by the neural network. So, that is essentially our error times the derivative of the output function of the kth neuron with respect to the input that it receives times the input that is coming on the mth line, which will be essentially h^i_m.

So, if you think about it that is exactly the update that we had here except that I have change the notation to apply to the second layer weights in the three layer network. So, this we can just get from the single neuron update. So, what we do about the first layer weights? So, let us see how you will do this, this essentially uses the very simple idea of chain rule from differentiation, I am going to take a very specific weight here. So, I would like to find the derivative of the error with respect to the first layer weight that runs from the l^{th} input neuron to the m^{th} hidden neuron.

So, I am just looking at this one weight here, we are trying to find out what is the gradient of the error at the output with respect to that one weight. So, I can rewrite this as follows, so the derivative of the error with respect to the first layer weight is essentially the derivative of the error with respect to the output of the m^{th} neuron times the derivative of the m^{th} neuron with respect to the weight.

So, this makes sense, because the weight to the m^{th} neuron affects the output only via the output of the m^{th} neuron, so it affects the overall output of the network only via the output of the m^{th} neuron. So, I can essentially apply the chain rule here. So, let us take this bit by bit. So, if you look at this, so you can see that this is immediately obvious, because this is the function we are talking about here.

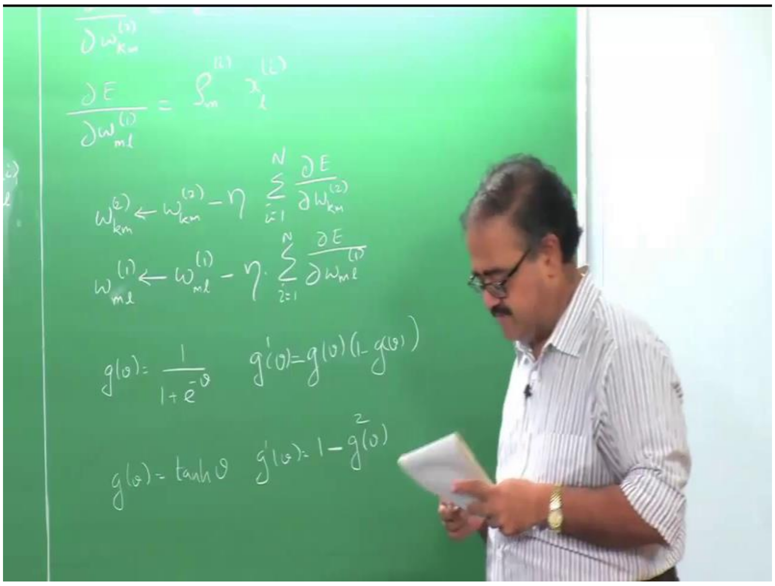
So, if we take the derivative of h^i_m with respect to w_{lm}, then you essentially going to get the derivative of g times the derivative of this expression with respect w_{lm} which will just be x_l^i. So, this is the second term in the derivative. So, the first term in this derivative is the one that requires a little bit more work, so let us see how we will do that. So, I am going to try and evaluate that expression here, so if you think about it, so the different ways in which the output of the m^{th} neuron can influence the overall error is essentially through each one of the output neuron that m^{th} hidden neuron connects to.

So, it can influence the error through this output or it can influence error through this output. So, we are essentially summing over all possible outputs k of the gradient of the error with respect to f_k and the gradient of f_k with respect to the output of the hidden neuron. So, you can easily evaluate the derivative, so the derivative of the error with respect to f_k. So, if think about it, so this is the expression we have.

So, the derivative of the error with respect to f_k, so going from this expression derivative of the error with respect f_k going to be... So, negative of y_i - $\hat{f}_k x_i$, because we are taking derivative with respect to f_k here. So, we do not have to worry about the further terms that constitute f_k. The second term is concerned that is essentially looking at this. So, derivative of f_k with respect to h^i_m's essentially the derivative of o with respect to h times the derivative of the argument of o with respect to h which gives us just the weight w_{km} complex expression for this.

So, before I put these things together to make things a little simple let me introduce a small notational thing. So, that it makes lie for little easier for us, so I am going to say δ_k for the input i this essentially the error term times the derivative of the last layer's output function and also define this term row m for the input i as the essentially the derivative of g with respect to w times the summation of $w_{km} * \delta_k^i$.

(Refer Slide Time: 12:42)


$\partial w_{km}^{(2)}$
$\frac{\partial E}{\partial w_{ml}^{(1)}} = \rho_m^{(i)} x_l^{(i)}$
$w_{km}^{(2)} \leftarrow w_{km}^{(2)} - \eta \sum_{i=1}^{N} \frac{\partial E}{\partial w_{km}^{(2)}}$
$w_{ml}^{(1)} \leftarrow w_{ml}^{(1)} - \eta \cdot \sum_{i=1}^{N} \frac{\partial E}{\partial w_{ml}^{(1)}}$
$g(v) = \frac{1}{1+e^{-v}}$
$g'(v) = g(v)(1-g(v))$
$g(v) = \tanh v$
$g'(v) = 1 - g^2(v)$

So, the error of the derivative of the error with respect to the output layer weights, with respect the w to weights is essentially δ_k^i * h^i_m, so likewise the derivative of the error with respect to the first layer weights, the weights from the input layer to the hidden layer is given by $\rho_m^i * x_l^i$.

So, if you think about it, so this part corresponds to this and that comes from here and x_l^i is what is left out here. So, this expression look pretty compact and now we essentially have... So, you essentially update the parameters by w_{km} is just change by accept the gradient here and w_{lm} is again change by the gradient that we are computed here. So, one thing I want to point out here that if you are function, the function g or your function o happens to be a sigmoid, then g' = g(v)(1 – g(v)).

And suppose you are using a tanh function, because you want here outputs to run from -1 to + 1, suppose then essentially this gives you then g' = 1 – g^2(v) So, you can see that the sigmoid functions have a very convenient form for the derivatives. So, one thing that we have to be careful about here is the fact that these are gradient following methods and therefore, and there is a good chance that will get stuck in local optima and there are many techniques for getting out of these local optima, but we are not discuss too many of them there one of the simplest one is of course, to try this with multiple starting stage, starting points for your weights and taking the set of weights that gives you the best possible result at the end of some kind of experimentation.

So, that brings us to the end of this module on training your neural network using back propagation. So, but this form of training has it is own draw backs will see what is that in the next module.