Deep Learning for Computer Vision
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 24
Feed-forward Neural Networks and Back-propagation Part-2

(Refer Slide Time: 0:16)

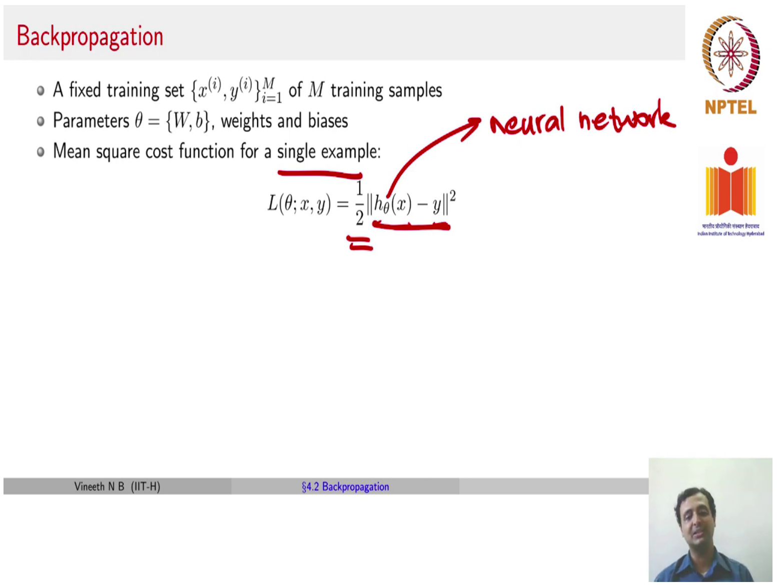
Gradient Descent

In the next few slides, we introduce **backpropagation**, a procedure which combines gradient computation using chain rule and parameter updation using Gradient Descent, thus fully describing the neural network training algorithm.

Vineeth N B (IIT-H) §4.2 Backpropagation

This leads us to the popular methodology which we talked about in the history of Neural Networks too which is known as Back-propagation, back-propagation is a procedure that combines gradient computation using chain rule and parameter updation using Gradient Descent. Remember we said, we want to minimize the loss function, we said gradient descent is the way to do minimize the loss function but, then gradient descent means, computation of derivatives for which we need the chain rule that is going to be the overall strategy for us to train the neural network and let us see more details.

(Refer Slide Time: 0:55)



Backpropagation

- A fixed training set $\{x^{(i)}, y^{(i)}\}_{i=1}^{M}$ of M training samples
- Parameters $\theta = \{W, b\}$, weights and biases
- Mean square cost function for a single example:

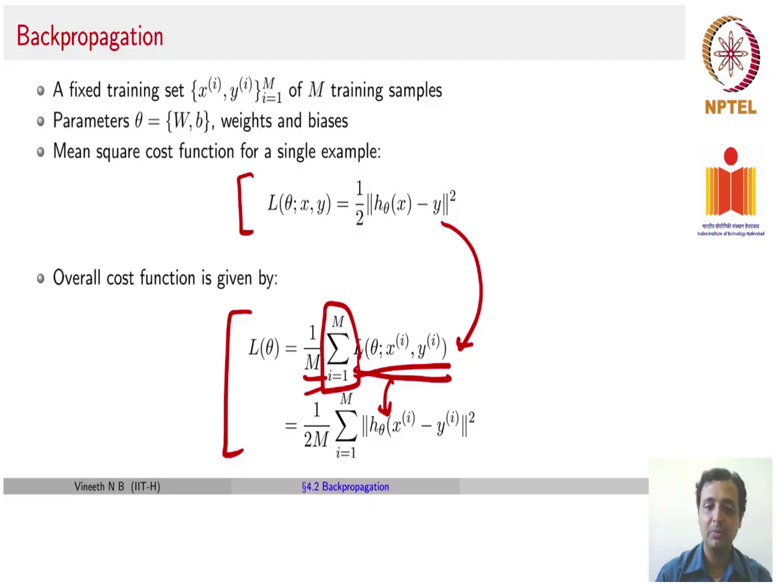
$$\underline{L(\theta; x, y)} = \underset{=}{\frac{1}{2}}\|\underline{h_\theta(x) - y}\|^2$$

Let us, consider once again the simple feed-forward neural network or what is the other name for multi-layer perceptron let us, consider once again a set of m training samples given by $\{x^i, y^i\}_{i=1}^{M}$, θ can constitute all the weights and biases in the neural network the mean square cost function for a single example be given by let us, assume that $h_\theta(x)$ corresponds to your neural network at this time that is your neural network and y is the expected output because, we know that is the output for the correct corresponding training sample.

So, given one particular sample x and its correct label y your loss function is given by $(h_\theta(x) - y)^2$ remember, we said mean squared error we are not doing the mean here because it is a single sample we are going to introduce we multiplied by half purely for mathematical simplicity you will see why a bit later but, the reason why it does not matter is because we are interested in the θ that minimizes the loss function and not the loss function value itself.

So, whether we minimize half of the loss function or the full loss function it is the same theta that will give you the minimum in both these cases so, it does not matter to us and we introduce that purely for mathematical simplicity.

(Refer Slide Time: 2:24)



Backpropagation

- A fixed training set $\{x^{(i)}, y^{(i)}\}_{i=1}^{M}$ of M training samples
- Parameters $\theta = \{W, b\}$, weights and biases
- Mean square cost function for a single example:

$$L(\theta; x, y) = \frac{1}{2}\|h_\theta(x) - y\|^2$$

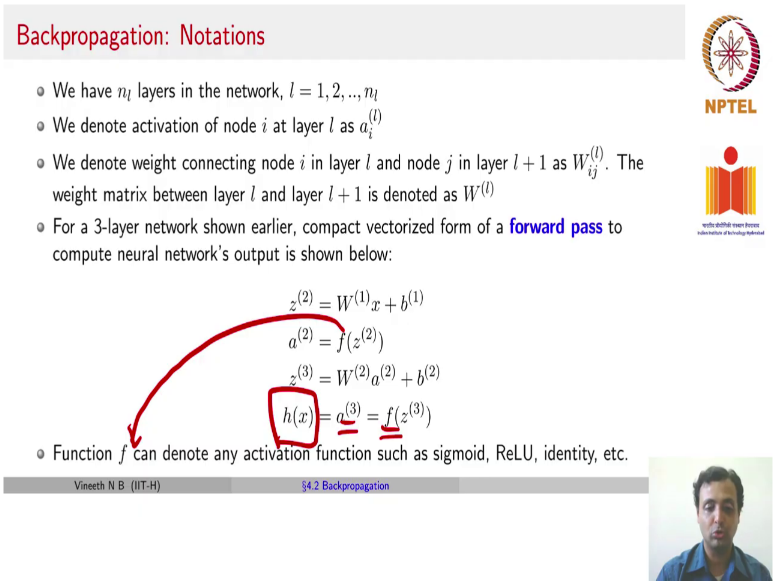
- Overall cost function is given by:

$$L(\theta) = \frac{1}{M}\boxed{\sum_{i=1}^{M}} L(\theta; x^{(i)}, y^{(i)}) = \frac{1}{2M}\sum_{i=1}^{M}\|h_\theta(x^{(i)} - y^{(i)}\|^2$$

Vineeth N B (IIT-H) §4.2 Backpropagation

Now, the overall cost function across all examples we saw is the cost function for a single example. The overall cost function across all of these examples is going to be given by $(1/M)\sum_{i=1}^{M} L(\theta; x^{(i)}, y^{(i)})$. So, the same loss function summed up over all training samples and then averaged and this term can be replaced by $(1/2M)\sum_{i=1}^{M} ||h_\theta(x^{(i)}) - y^{(i)}||^2$, your mean square error definition that is the overall cost function that we are going to go with. We still have to compute the gradient but that is the overall cost function.

(Refer Slide Time: 3:01)



Backpropagation: Notations

- We have n_l layers in the network, $l = 1, 2, .., n_l$
- We denote activation of node i at layer l as $a_i^{(l)}$
- We denote weight connecting node i in layer l and node j in layer $l+1$ as $W_{ij}^{(l)}$. The weight matrix between layer l and layer $l+1$ is denoted as $W^{(l)}$
- For a 3-layer network shown earlier, compact vectorized form of a **forward pass** to compute neural network's output is shown below:

$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$\boxed{h(x)} = \underline{a^{(3)}} = \underline{f}(z^{(3)})$$

- Function f can denote any activation function such as sigmoid, ReLU, identity, etc.

Vineeth N B (IIT-H) §4.2 Backpropagation

Let us now define the notations to be able to define the back-propagation procedure completely.

Let us, assume that there n_l layers in the neural network, n_l layers in the neural network going from $l = 1, \cdots n_l$ let us, denote the activation of a node i, at layer l as a_i^l rather, remember every layer has a set of neurons. We are now looking at the i^{th} neuron which is the node i, that we are talking about and we are looking at the l^{th} layer of the neural network.

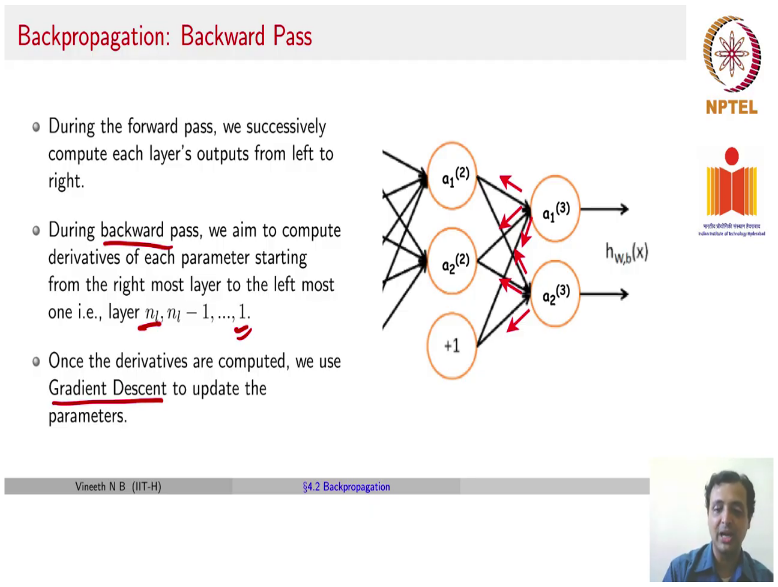
And the activation is simply the output of that particular neuron remember every neuron is now individually like a perceptron and we already said that for a perception instead of having a threshold function we can have activation functions such as, sigmoid, hyperbolic, tangent, so on and so forth the output of that function is what we denote as an activation on that particular node i, again to repeat that node is like a perception and in the l^{th} layer we call it a_i^l .

We denote, the weight connecting node i, in layer l and node j in $(l + 1)$ as W_{ij}^l and the entire weight matrix between layer l and layer $(l + 1)$ is denoted as W^l, for a three layer network that we saw earlier a compact vectorized form of a forward pass can actually be written like this and hopefully this will give you clarity on the entire procedure.

Given, input x you take the first layers weights and biases and you get the output on the next layer that is $z^{(2)}$ second layer, you take z and apply an activation function f and the output of that becomes $a^{(2)}$ in the second layer which is going to assume one neuron in each layer just to make the notations simpler but, you can extrapolate this when x is a vector, z is a vector, a is a vector, so on and so forth.

Now, you take the activation take the next layers weights which is given by $W^{(2)}$ and $b^{(2)}$ and you do $z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$, that is the z in the third layer you apply an activation function on that to get $a^{(3)}$ which is the output of the third layer, which is also the output of the neural network. In this case, the function f denotes an activation function such as sigmoid, tanh, identity, so on and so forth.

(Refer Slide Time: 5:52)



Backpropagation: Backward Pass

- During the forward pass, we successively compute each layer's outputs from left to right.
- During backward pass, we aim to compute derivatives of each parameter starting from the right most layer to the left most one i.e., layer $n_l, n_l - 1, ..., 1$.
- Once the derivatives are computed, we use Gradient Descent to update the parameters.

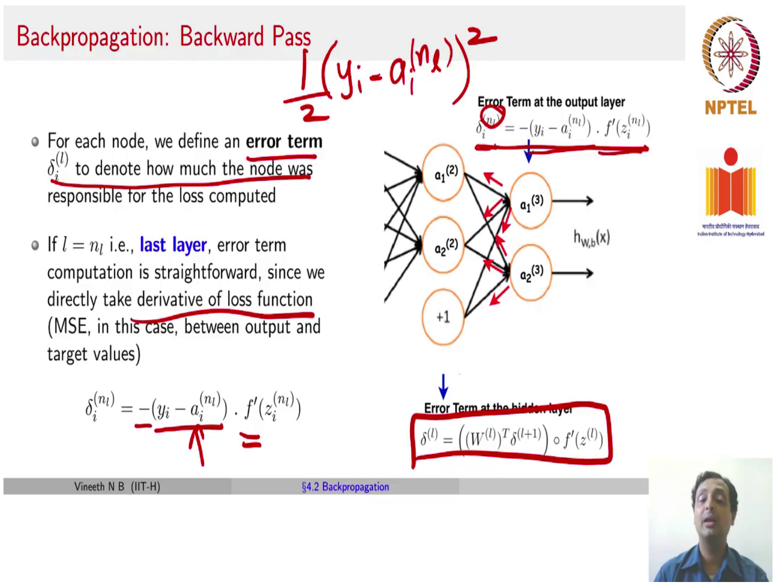
Let us now try to see what back-propagation actually does while training a neural network. During the forward pass for neural networks, every neural network can now be divided into a forward pass and backward pass. Forward pass is when you give input to a neural network and you propagate the input through the weights of the various layers and you get an output at the out layer or at the output layer you call that the forward pass.

Now, based on the output at that output layer you would get a certain loss based on the loss function that you chose you expected a certain output because, it is training data you know what the expected output should be you know what the neural network is throwing out the difference between the two or whatever loss function it need not be just difference you could use other loss functions that loss function of the output is what we now, have at the last layer and we now have to use that loss to compute the gradient for each of the weights in the neural network that is the backward pass.

So, during the forward pass we simply compute each layer's outputs and move from left to right during the backward pass we compute the loss in the last year and move from the rightmost layer to the leftmost layer starting from nl all the way going to the first layer, we compute gradients going from the rightmost layer to the leftmost layer and once we have all the gradients computed we can use gradient descent to update the parameters how do we do current new parameter is equal to old parameter minus eta which is the learning rate or step size, times the gradient with respect to that parameter.

Now, let us also try to see how this specific propagation of gradients happens from the last layer till the first layer.

(Refer Slide Time: 7:44)



Backpropagation: Backward Pass

$\frac{1}{2}(y_i - a_i^{(n_l)})^2$

- For each node, we define an **error term** $\delta_i^{(l)}$ to denote how much the node was responsible for the loss computed
- If $l = n_l$ i.e., **last layer**, error term computation is straightforward, since we directly take derivative of loss function (MSE, in this case, between output and target values)

$$\delta_i^{(n_l)} = -(y_i - a_i^{(n_l)}) \,.\, f'(z_i^{(n_l)})$$

Error Term at the output layer

$$\delta_i^{(n_l)} = -(y_i - a_i^{(n_l)}) \,.\, f'(z_i^{(n_l)})$$

Error Term at the hidden layer

$$\boxed{\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \circ f'(z^{(l)})}$$

Vineeth N B (IIT-H) §4.2 Backpropagation

The first thing is we denote for each node an error term that denotes how much that node was responsible for the loss at the output layer. Let us imagine now that the error term at the output layer is given by the error term at the ith node in the output layer. Remember, even the output layer would have multiple output nodes, the output need not be a single value, you could be giving a vector as an output too.

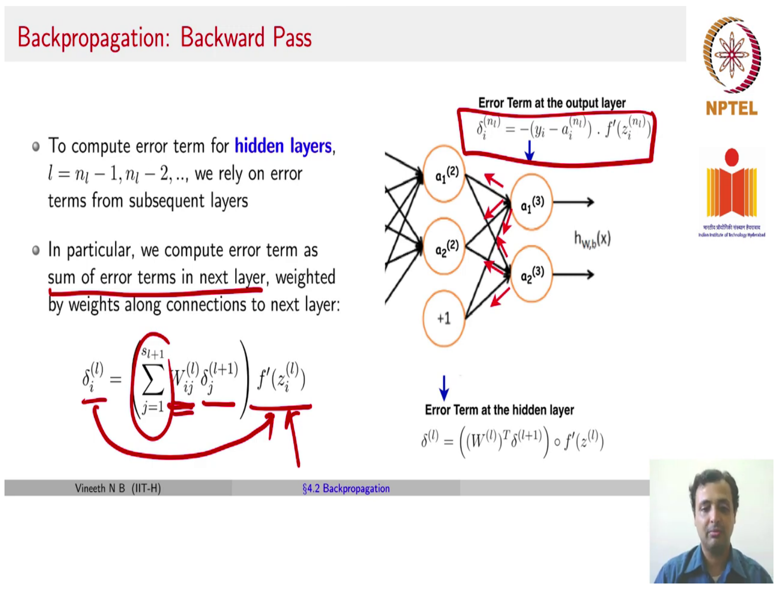
For example, you may want to predict at what location an event may occur next. So, let us say you want to use some logic, you want to predict using machine learning to predict where the next FIFA World Cup will happen so, you may want to predict that in longitude and latitude for instance. So, that could be the two outputs that you may want to predict, so your output layer could have many values in that layer.

So, $\delta_i^{(n_l)}$ is the last layer delta i, is the delta or the error for the i^{th} node in that layer and the error term at that output layer is given by $-(y_i - a_i^{(n_l)}) \,.\, f'(z_i^{(n_l)})$. How did we get that? This is simply the gradient of the mean square and remember we said we are taking mean squared error as the loss function, we simply take the derivative of the mean square error.

So, in our case it is going to be $y_i - a_i^{(n_l)}$. Let us, assume that is your final layers output after applying the activation function whole square and remember we said we will have a half for convenience I mean, when you take a gradient of this with respect to ai you are going to have minus two and two would get cancelled and you would have $y_i \; - \; a_i^{(n_l)}$ that would be your gradient of the mean square error but, remember that a_i itself is a function of z_i, the way we defined it, you apply an activation function f and then you get a_i which means in the chain rule you would then have $f^{'}(z_i^{(n_i)})$, where $a_i \, = \, f(z_i^{(n_i)})$

So, that is going to be the error term at the output layer. Now, we are going to claim that the error term of the hidden layer is going to be given by this quantity and let us try to find how.

(Refer Slide Time: 10:34)



Backpropagation: Backward Pass

- To compute error term for **hidden layers**, $l = n_l - 1, n_l - 2, ..,$ we rely on error terms from subsequent layers
- In particular, we compute error term as sum of error terms in next layer, weighted by weights along connections to next layer:

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ij}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

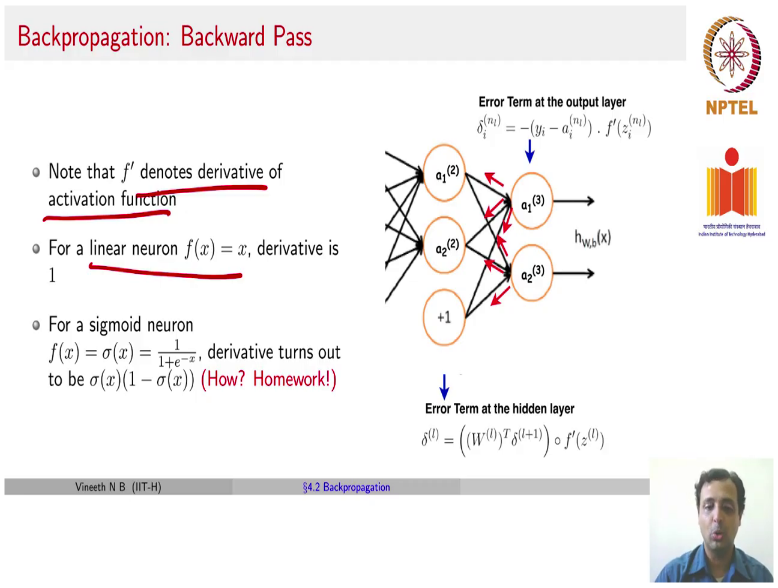
Error Term at the output layer

$$\delta_i^{(n_l)} = -(y_i - a_i^{(n_l)}) \, . \, f'(z_i^{(n_l)})$$

Error Term at the hidden layer

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)} \right) \circ f'(z^{(l)})$$

Vineeth N B (IIT-H) §4.2 Backpropagation



Backpropagation: Backward Pass

- Note that f' denotes derivative of activation function
- For a linear neuron $f(x) = x$, derivative is 1
- For a sigmoid neuron $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$, derivative turns out to be $\sigma(x)(1 - \sigma(x))$ (How? Homework!)

Error Term at the output layer

$$\delta_i^{(n_l)} = -(y_i - a_i^{(n_l)}) \, . \, f'(z_i^{(n_l)})$$

Error Term at the hidden layer

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)} \right) \circ f'(z^{(l)})$$

$a_1^{(2)}$
$a_2^{(2)}$
+1
$a_1^{(3)}$
$a_2^{(3)}$
$h_{W,b}(x)$
NPTEL

Vineeth N B (IIT-H) §4.2 Backpropagation

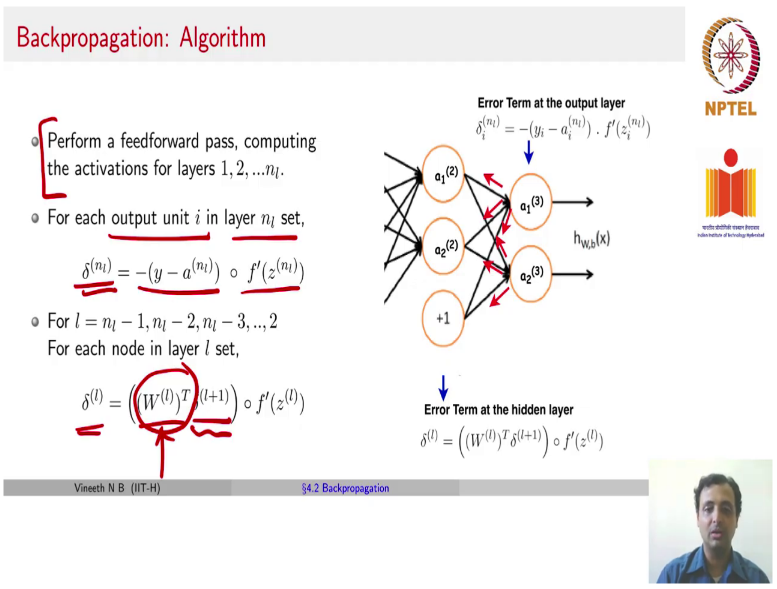
For the hidden layer we have to rely on the error terms from the subsequent layers. So, once again let us assume that the error term at the output layer is given by this. Now the error term at the hidden layer is given by some of the error terms in the next layer which means, $\delta_i^{(l)} = (\sum_{j=1}^{n_{l+1}} W_{ij}^{(l)} \delta_j^{(l+1)}) f'(z_i^{(l)})$.

$f'(z_i^{(l)})$ is simply the derivative of the activation function at that particular node. Remember, this is the reason why we said it is nice that the sigmoid function is continuous and differentiable because we need this derivative across the activation function to be able to compute your chain.

Now, you could ask me why am I multiplying it by the weights? Because, that is the contribution that this node had to the error on that particular node in the next layer the i^{th} node contributed by W_{ij} times to the error at the j^{th} node in the next layer, that is what we are denoting it by the sum of errors as we just mentioned $f'(z_i^{(l)})$ denoted the derivative of the activation function for a linear neuron $f(x) = x$ derivation is 1. Simply that term would not matter for the rest of the computations.

But, for a sigmoid neuron $f(x) = \sigma(x)$, derivative is equal to $1/(1 + e^{-x})$. The derivative turns out to be $\sigma(x)(1 - \sigma(x))$. Suppose, you already know it for those of you who did Assignment 0 in this course you should have seen it but, you can also work it out as homework.

(Refer Slide Time: 12:47)



Backpropagation: Algorithm

- Perform a feedforward pass, computing the activations for layers $1, 2, \ldots n_l$.
- For each output unit i in layer n_l set,

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \circ f'(z^{(n_l)})$$

- For $l = n_l - 1, n_l - 2, n_l - 3, .., 2$
  For each node in layer l set,

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \circ f'(z^{(l)})$$

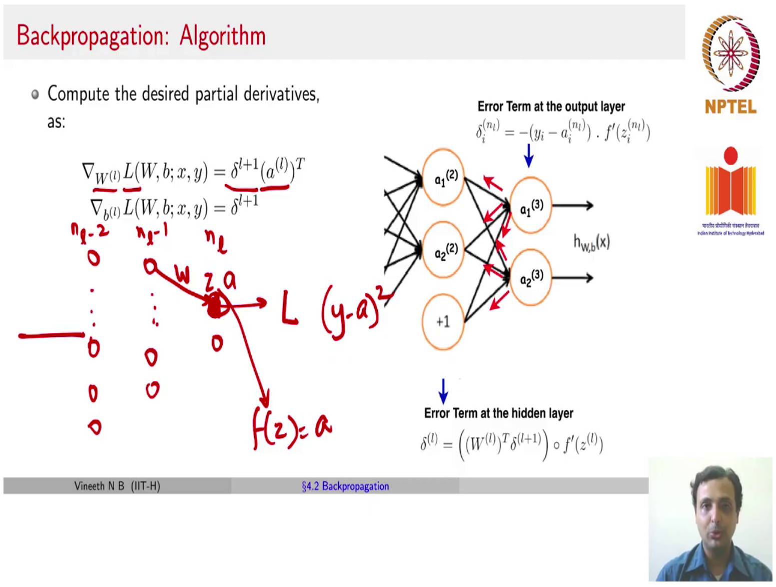
Error Term at the output layer
$\delta_i^{(n_l)} = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$
$a_1^{(2)}$
$a_2^{(2)}$
+1
$a_1^{(3)}$
$a_2^{(3)}$
$h_{W,b}(x)$
Error Term at the hidden layer
$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \circ f'(z^{(l)})$

Vineeth N B (IIT-H) §4.2 Backpropagation

So, the way we are going to train the full network now, is we perform a feed forward pass computing the activations for all layers, for each output unit i, in the set of neurons in the last layer denoted by n_l, we compute the error which is $\delta^{(n_l)}$ which is the derivative of the loss function with respect to the final activation into the derivative of the final activation with respect to $z^{(l)}$ which is before applying the activation function and then on for the previous layers each $\delta^{(l)}$ can be written as $W^{(l)^T} \delta^{(l+1)}$.

Remember, now we have written this as a matrix and that is why the summation disappeared and this $W^{(l)}$ matrix times this $\delta^{(l+1)}$ vector would ideally give you the summation that you are looking for that we talked about on the earlier slide.

(Refer Slide Time: 13:48)



Backpropagation: Algorithm

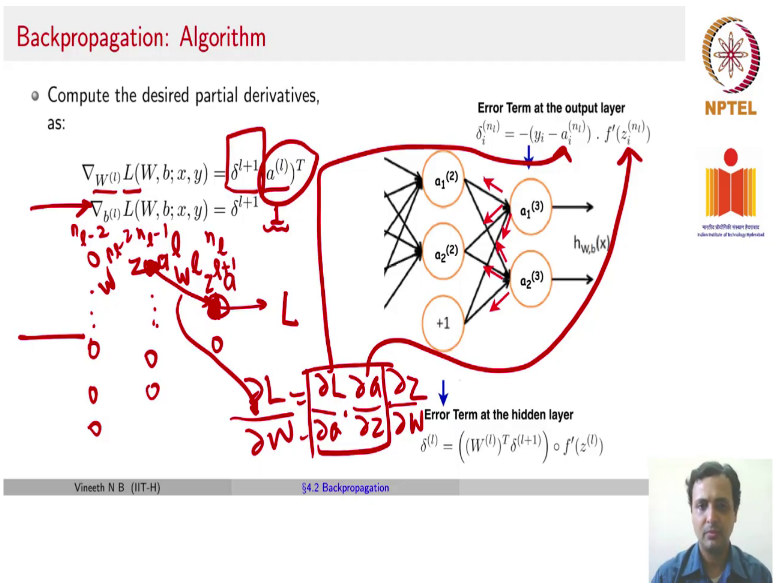
- Compute the desired partial derivatives, as:

$$\nabla_{W^{(l)}} L(W,b;x,y) = \delta^{l+1}(a^{(l)})^T$$
$$\nabla_{b^{(l)}} L(W,b;x,y) = \delta^{l+1}$$

Error Term at the output layer
$$\delta_i^{(n_l)} = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

Error Term at the hidden layer
$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \circ f'(z^{(l)})$$

Vineeth N B (IIT-H) §4.2 Backpropagation



Backpropagation: Algorithm

- Compute the desired partial derivatives, as:

$$\nabla_{W^{(l)}} L(W,b;x,y) = \delta^{l+1}(a^{(l)})^T$$
$$\nabla_{b^{(l)}} L(W,b;x,y) = \delta^{l+1}$$

Error Term at the output layer
$$\delta_i^{(n_l)} = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

Error Term at the hidden layer
$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \circ f'(z^{(l)})$$

$a_1^{(2)}$
$a_2^{(2)}$
+1
$a_1^{(3)}$
$a_2^{(3)}$
$h_{W,b}(x)$
n_l-2
n_l-1
n_l
L
$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial W}$

Vineeth N B (IIT-H) §4.2 Backpropagation

Having computed these deltas for different layers, remember delta is the contribution of every node to the error we are now, going to use those deltas to find out the partial derivatives of the loss with respect to every weight in the neural network. How we do it is very simple. Now, it is going to be defined as the loss or the gradient of the loss with respect to every weight in any layer l is simply given by take the $\delta^{(l+1)}$ the contribution of the error term in the next layer and multiply it by activation of the l^{th} layer $a^{(l)}$, let us try to understand why this is correct. Let us consider a neural network let us say this is these are some layers of a neural network and these are nodes.

Let us assume that there are many more layers before this but, this is the last layer which is n_l this is the penultimate one $n_l - 1$, one before that $n_l - 2$ Now, in the last node in the output layer, remember let us draw that a little bit bigger, so this side is going to be this w is what we want to compute it with respect to the gradient z and when you apply the activation on z it becomes a.

So, the way z and a are related is $f(z) = a$ where f is the activation function of your choice and this a is finally used to compute your final loss function, remember we said it is $(y - a)^2$ where, a is the output and y is the expected output. So, to compute the gradient with respect to w that is unnecessary information here, so to compute $\partial L/\partial W$ for the moment let us consider this weight that is what we are computing with respect to which we apply chain rule, chain rule tells us this is $\partial L/\partial a * \partial a/\partial z * \partial z/\partial w$.

Now, $\partial L/\partial a$ is exactly this term here that is the derivative of the loss with respect to a. $\partial a/\partial z$ is exactly this term here which is the gradient of the activation function f'. $\partial z/\partial w$ is now what we have to compute but, we know now that $\partial z/\partial w$ is nothing but the activation in the previous layer. So, let us probably write it this way the activation is right here the previous layer z would be here and activation is applied and a is what would come here remember, for $W^{(l)}$ we will go from $a^{(l)}$ to $z^{(l+1)}$.

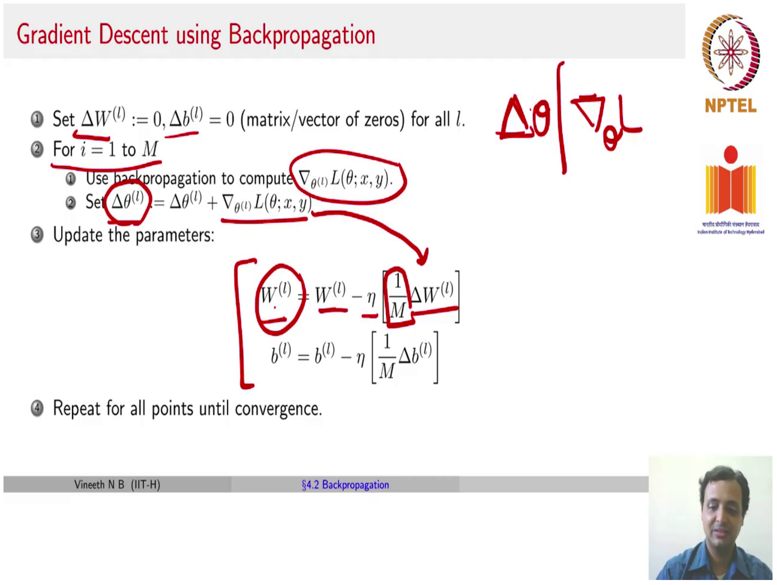
So, if you want to compute the derivative with respect to $W^{(l)}$, $\partial z/\partial W^{(l)}$ will simply be a from the previous layer, why? Because, $a^{(l)}W^{(l)}$ will be say $z^{(l+1)}$, we are simplifying it and talking in terms of scalars but, you could now expand this to all the other nodes in that layer and also write it as a vector but, that is the broad idea.

So, once again if you instead had a delta that was, let us assume now that we want to compute the gradient with respect to a previous layer let us call that W^{n_l-2}, so one of these weights once again is going to be the same. You just have to keep computing the chain rule over and over again and you will find that the chain rule will give you delta at this node as the contribution of that node of the error.

Now, you simply have to differentiate that with respect to w which will give you the activation in the previous layer as the additional term that gives you your gradient, work it out a bit carefully and you will follow that this is fairly straightforward. Now, this gives you the gradient with respect to w, similarly if you noted the second equation here that has the gradient with respect to the bias it is exactly the same thing where the activation is 1 remember because the activation that goes into the bias in any layer is going to be 1 remember once again, that a bias could there for every layer not just the first layer could be therefore, every layer but, that value will come out to be 1.

So, what have we done to summarize we have computed those deltas that are the errors of how each node contribute to the output and simply use that to make our chain rule computation simpler, we could have instead not written out delta and worked out the chain rule but, you would have got exactly the same value, delta gives us a placeholder to hold what how a particular node in any layer contributed the error and now, it is very easy to compute the gradient using the error for a previous layers weight.

(Refer Slide Time: 19:20)



Gradient Descent using Backpropagation

$\Delta\theta \,|\, \nabla_\theta L$

1. Set $\Delta W^{(l)} := 0, \Delta b^{(l)} = 0$ (matrix/vector of zeros) for all l.
2. For $i = 1$ to M
   1. Use backpropagation to compute $\nabla_{\theta^{(l)}} L(\theta; x, y)$.
   2. Set $\Delta\theta^{(l)} = \Delta\theta^{(l)} + \nabla_{\theta^{(l)}} L(\theta; x, y)$
3. Update the parameters:

$$W^{(l)} = W^{(l)} - \eta\left[\frac{1}{M}\Delta W^{(l)}\right]$$
$$b^{(l)} = b^{(l)} - \eta\left[\frac{1}{M}\Delta b^{(l)}\right]$$

4. Repeat for all points until convergence.

Vineeth N B (IIT-H) §4.2 Backpropagation

That gives us our final algorithm for training a neural network, let us go over it. So, let us assume that $\Delta W = 0$ and $\Delta b = 0$. Just a minor clarification here to differentiate Δ and ∇, Δ defines the change in a value, ∇ defines the gradient of a function with respect to a value so, please keep this in mind in the notations. So, ΔW and Δb are the changes that we are going to make to W and b in a given iteration of gradient descent.

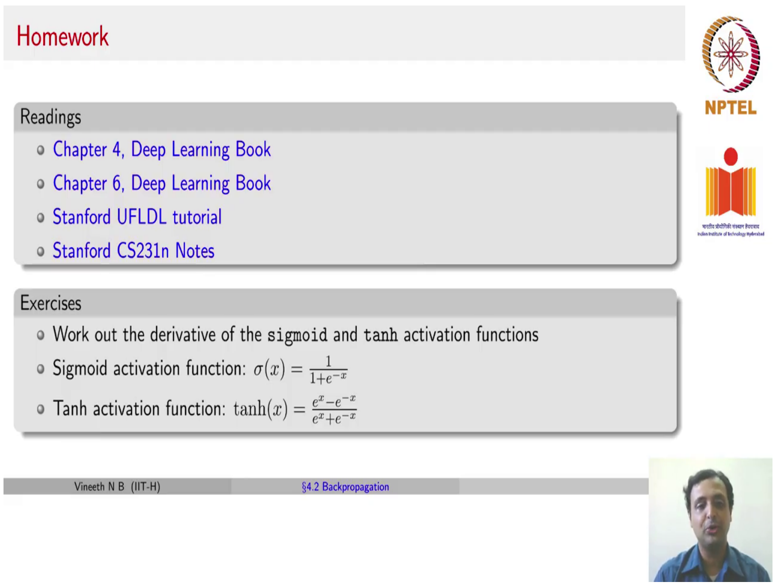
Let us, initialize that to 0 then for each data point that you have in your training data set you forward propagate that data point, you compute the error, compute the loss, compute the gradient and use back propagation to compute the gradient with respect to every weight in the neural network that gives you $\nabla_\theta L$.

Now, your $\Delta\theta$ composed of ΔW and Δb you increase that by this much gradient now and you would keep going this for a loop for every data point in your data set and that gives you the total delta theta you have ΔW and Δb. That is the total change that you want to make to your W and b.

Now, to update the parameters using gradient descent you are going to say, $W^{(l)} = W^{(l)} - \eta * 1/M(\Delta W^{(l)})$. Divide by m is simply to average the gradient across all of your data points. Remember here we were summing up the gradients for all of the data points.

Now, we simply want to divide by m to average the gradient across all of your data points and that tells you how your current weights had an impact on the loss across all of your training data points that is the gradient that you are going to use to update w then you will get $W^{(l)}$ at the next time step and similarly, b at the next time step and you keep repeating this until convergence. Remember, convergence is when the gradient of the loss with respect to w and b becomes 0. In practice this may be difficult. How do we handle it in practice? We will talk about it in a later lecture.

(Refer Slide Time: 21:52)



Homework

Readings
- Chapter 4, Deep Learning Book
- Chapter 6, Deep Learning Book
- Stanford UFLDL tutorial
- Stanford CS231n Notes

Exercises
- Work out the derivative of the sigmoid and tanh activation functions
- Sigmoid activation function: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Tanh activation function: $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

Vineeth N B (IIT-H) §4.2 Backpropagation

For further, readings please read chapter 4 and chapter 6 of deep learning book not all sections in these two chapters may be relevant so, I did advise you to read what is relevant to the sections that we have covered so far also, go through the Stanford tutorial and the CS231 endnotes the links are right on this slide if you are interested.