

**Data Mining**  
**Prof. Pabitra Mitra**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 31**  
**Artificial Neural Networks – IV**

We will continue with multi layer perceptrons. As I had mentioned in the last lecture, we use a approximation of the step function which is the sigmoid function.

(Refer Slide Time: 00:28)

**Sigmoid unit**

Diagram: Inputs  $x_1, x_2, \dots, x_n$  with weights  $w_1, w_2, \dots, w_n$  and bias  $x_0=1$  with weight  $w_0$  enter a summation node  $\Sigma$ . The net input is  $net = \sum_{i=0}^n w_i x_i$ . This net input enters an activation function node  $f$ , which outputs  $o = f(net)$ .

- $f$  is the sigmoid function
- Derivative can be easily computed:
- Logistic equation
  - used in many applications
  - other functions possible (tanh)
- Single unit:
  - apply gradient descent rule
- Multilayer networks: **backpropagation**

Equation:  $f(x) = \frac{1}{1 + e^{-x}}$

Derivative:  $\frac{df(x)}{dx} = f(x)(1 - f(x))$

Handwritten note:  $f(x) = \tanh(x)$

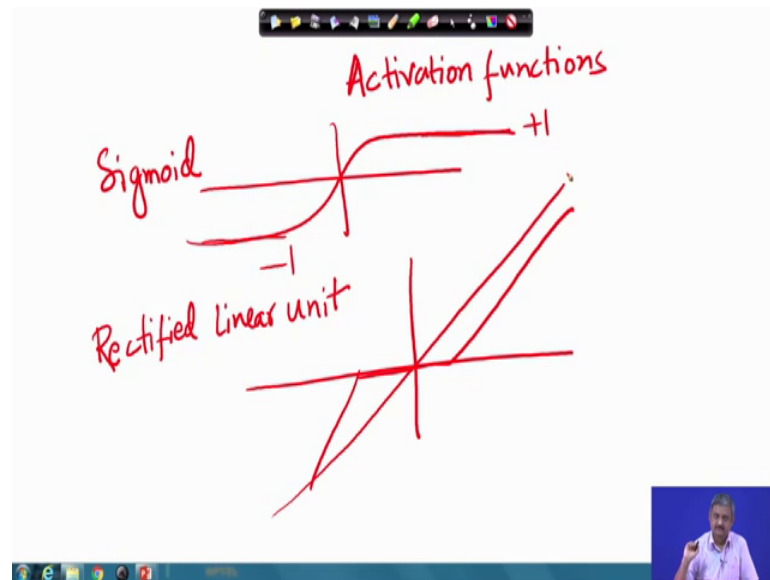
Graph: A plot of the sigmoid function showing its characteristic S-shape.

So, instead of having something like this we will have something like this ok. And the sigmoid function as this one and it many possible algebraic equation forms are or have the same for example, 10 hyperbolic another functions have a similar S-shape this is a separate. One of the popular functions used is this form.

So, the named sigmoid is for a general class of a separate function this is one how particular form. The other form for example,  $f(x) = \tanh(x)$  is also of this form ok. The good thing about this particular form is that you can compute the derivative of this function from the function itself. So, if you take the derivative with respect to  $x$ , you will find that it is nothing but  $f(x) \cdot (1 - f(x))$ . So, the value of the function into 1 minus the value of the function ok. So, that becomes the derivative.

So, if you plot the derivative it is  $f'(x)$  into  $1 - f(x)$  ok. Sometimes actually a another form that is popular in deep neural network is that because, you have something called a rectified linear unit, I will I will quickly explain it.

(Refer Slide Time: 02:21)



Saturates 2 plus 1 and minus 1 in between it is like this. So, this is sigmoid. Sometimes you have unit which is like this does not saturate just keeps on increasing, all right? Or maybe in a special case it is just like a proportional type. So,  $y$  is some constant time  $x$ , all right?

So, this is about the activation functions. We will so, the multi-layer perceptron has 2 extensions over the normal perceptron, it has hidden layers and instead of a step activation it uses a sigmoid activation.

(Refer Slide Time: 03:42)

**Error Gradient for a Sigmoid Unit**

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_{d \in D} (t_d - o_d) \left( -\frac{\partial o_d}{\partial w_i} \right) \\ &= -\sum_{d \in D} (t_d - o_d) \frac{\partial o_d}{\partial \text{net}_d} \frac{\partial \text{net}_d}{\partial w_i}\end{aligned}$$

net: linear combination  
o (output): logistic function

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, that derivative can be easily taken. Now, let us come to the problem of finding out the proper set of weights.

(Refer Slide Time: 03:53)

**Training Multilayer Perceptrons**

1. Initialize all weights to random values
2. Update weights
  - Forward propagation
  - Backward propagation
3. Stop when  $\Delta W$  is small over successive iterations

Diagram showing a neural network with input nodes  $x_1, x_2, x_3$  and output nodes  $z_1, z_2$ . Weights are labeled  $w_{ij}^{mn}$ .

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we will use the same methodology, we will use the same methodology.

So, let me draw the picture I am drawing only one hidden layer. Now let us roughly drawing it and there are weights oh let this is one one one one ok. So, the as I mentioned the weight the superscript is the layer numbers from which layer to which layer say layer

m to layer n, and the subscript is that which neuron to each neuron. It h neuron of m th layer to j th neuron of n th layer that is how I terminate the weight.

So, the steps are that following, initialize all the weights to random values just like the normal perceptron then, update weights I will explain in detail. And finally, stop when delta w change in weight value update value is small over successive iterations ok. So, but well this is same as the multi-layer perceptron; but the only thing is that this weight update is now more complex than the perceptron.

Now, it has 2 steps. Forward propagation 2 steps and backward propagation, all right?. So, I will explain this in detail, but the philosophy is something like this, it is like suppose you are you are you are trying to trying to say park your bike. So, whatever what do you do? You see the current orientation of the bike and let us place it, if it does not fit well you use the amount of error as a feedback, amount of mistake you made as a feedback and change your position again ok.

So; that means, once with the current estimate of the values you try to do that ask an amount of error you make in the task use that as a feedback to current your current position to change your current position correct your current position, right? So, you first so, it is like you are trying to solve some problem if I solve it using whatever knowledge you have then, you see how much error you have made use that error to correct your knowledge again ok.

(Refer Slide Time: 08:26)

Forward Propagation

- Assume some value of the weight (in that iteration)
- Look at each training example  $(x_1, x_2, x_3)$
- For every neuron from input to output layer
  - Compute  $\sum w_{ij} x_i$
  - apply activation

Compute Mean Squared error  $\frac{1}{n} \sum (y_i - t_i)^2$

The diagram shows a neural network with three input nodes ( $x_1, x_2, x_3$ ) and three output nodes. Weights  $w_{ij}$  are shown between nodes. The output is labeled  $y_{out}$ .

So, and do this repeatedly till you get the answer to certain accuracy. So, I will explain this 2 step first I will explain forward propagation. So, this is my network this is my network and at any stage of the training say maybe the initial stage, or some intermediate iteration all these links will have some weight value some value may be not the correct value, but some value some value of the weights in that iteration. Whatever iteration is in that iteration, whatever I mean the value of tau it then what you do note that.

So, look at every training example let us say  $x_1, x_2, x_3$  ok. So, what we will do? You have this values and then use the weight value and do 2 steps add up add activation apply activation function. So, let me properly write it down compute  $x_i$  overall all the  $x_{ij}$  all the inputs, and then apply activation function so, layer by layer. First for this all this neuron then, their output as input to this for all these neurons all this neuron finally, output you get.

So; that means, for every input for these set of weights this is output; that means, we are propagating the input to the output that is why it is called a forward prevention. Now, this value of output you get it may not match the desired output. So, let me call it as  $t_i$  let me call it the desired output  $S$  target output  $t_i$ . So, this why this forward person produces may not match.

Now, you compute the error, root mean squared how you compute very simple. You know  $y_i$  you have computed actual output by forward propagation subtract from  $t_i$  take the square add up over all the training examples ok, that is your and sorry and take the average over all this. So, that is the mean square error. So, forward preparation will find out this mean square error.

(Refer Slide Time: 13:31)

Back propagation

- Propagate the error backward and update  $W_j$ 's
- $W_{\text{new}} = W_{\text{old}} + \eta \Delta W$
- $\Delta W$  is found by gradient descent

Diagram: A simple neural network with input  $x_i$  and output  $t_i$ . Arrows show the flow of error  $\Delta W$  back to the weights.

Now, next step. So, forward propagation it propagated. So, not this not this propagated the  $x$  values the output computed the  $y$  take the difference with  $t_i$  compute error. Now, this error you feedback as I have mentioned earlier, you feedback and update the  $w$ 's. So, I guess update what is the object same form?. Same form  $w_{\text{new}}$  is  $w_{\text{old}}$  plus some learning rate  $\eta$  times  $\Delta w$  ok.

(Refer Slide Time: 15:40)

Weight Space - Gradient descent

Graph: A 2D plot with axes  $w_1$  and  $w_2$ . The vertical axis is labeled 'Error  $E$ '. A curve represents the error surface. A point  $w_{\text{old}}$  is marked on the curve, and a point  $w_{\text{new}}$  is marked further down the curve, indicating the direction of gradient descent.

$\Delta W = \nabla E$

And this  $\Delta w$  gradient descent simpler take all the weights, I am not there will be more weights. Take the surface they read whatever you have value of  $w_{\text{old}}$  is, you move

it to  $w$  new move it to  $w$  new in the direction of the steepest descent. So,  $\Delta w$  is derivative of this error function the tangent the steepest descent.

(Refer Slide Time: 16:39)

**Error Gradient for a Sigmoid Unit**

$$\nabla E = \frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \rightarrow \text{Mean squared error}$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \quad \checkmark$$

$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \quad \checkmark$$

$$= \sum_{d \in D} (t_d - o_d) \left( -\frac{\partial o_d}{\partial w_i} \right) \quad \checkmark$$

$$= - \sum_{d \in D} (t_d - o_d) \frac{\partial o_d}{\partial \text{net}_d} \frac{\partial \text{net}_d}{\partial w_i} \quad \checkmark$$

*computed after forward propagation*

net: linear combination  
o (output): logistic function

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

I will go through the derivation of this I will go through the derivation of the delta E delta w.


So, this is my sorry, I forgot this half term in the error you can as well add a half term. So, this is the error propagation. I am just doing it this is the thing, this is the thing and this is the thing you just follow these steps, right? You just if you just go through the steps. Now, I am just taking normal derivative ok. So, follow it is they are in the uploaded slides if you are followed it, then these terms can be further expanded like this ok.

(Refer Slide Time: 17:55)

### Error Gradient for a Sigmoid Unit

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_{d \in D} (t_d - o_d) \left( - \frac{\partial o_d}{\partial w_i} \right) \\ &= - \sum_{d \in D} (t_d - o_d) \frac{\partial o_d}{\partial \text{net}_d} \frac{\partial \text{net}_d}{\partial w_i} \end{aligned}$$

$\frac{df(x)}{dx} = f(x)(1-f(x))$   
 net: linear combination  
 o (output): logistic function  
 $\text{net}_d = \sum w_{ij} x_i$   
 $o = f(\text{net}_d)$

$$\left. \begin{aligned} \frac{\partial o_d}{\partial \text{net}_d} &= \frac{\partial f(\text{net}_d)}{\partial \text{net}_d} = f(\text{net}_d)(1 - f(\text{net}_d)) = o_d(1 - o_d) \\ \frac{\partial \text{net}_d}{\partial w_i} &= \frac{\partial (\vec{w} \cdot \vec{x}_d)}{\partial w_i} = x_{i,d} \end{aligned} \right\}$$


When net is this summation over  $w_{ij} x_i$  from the previous layer ok. And this  $O$  is  $f$  of net  $f$  is the sigmoid, if you write down from the previous property that  $f(x)$  you get this which is this. Similarly, if you do this you get this, all right? So, if I go on further this is my update rule for the weights between output and hidden.

(Refer Slide Time: 19:03)


### Error Gradient for a Sigmoid Unit

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_{d \in D} (t_d - o_d) \left( - \frac{\partial o_d}{\partial w_i} \right) \\ &= - \sum_{d \in D} (t_d - o_d) \frac{\partial o_d}{\partial \text{net}_d} \frac{\partial \text{net}_d}{\partial w_i} \end{aligned}$$

net: linear combination  
 o (output): logistic function  
**Backpropagation Rule**

$$\frac{\partial o_d}{\partial \text{net}_d} = \frac{\partial f(\text{net}_d)}{\partial \text{net}_d} = f(\text{net}_d)(1 - f(\text{net}_d)) = o_d(1 - o_d)$$

$$\frac{\partial \text{net}_d}{\partial w_i} = \frac{\partial (\vec{w} \cdot \vec{x}_d)}{\partial w_i} = x_{i,d}$$

$$\frac{\partial E}{\partial w_i} = - \sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$




(Refer Slide Time: 19:14)

... Incremental Version

- Batch gradient descent for a single Sigmoid unit

$$E_D = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \quad \frac{\partial E_D}{\partial w_i} = - \sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

— batch

- Stochastic approximation

$$E_d = \frac{1}{2} (t_d - o_d)^2 \quad \frac{\partial E_d}{\partial w_i} = - (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

incremental

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Similarly, you can repeat this where output will become the output of the hidden units for the output input. This way you keep on doing and you call this as a gradient descent back propagation algorithm. So, this is the. So, called sorry this is the rule ok.

So, now, again there can be a incremental or a gradient; that means, either you take error to be sum of all the thing or for a single a single update. So, this is batch you can have a say single so, not batch sorry this is incremental. For every training input you object earlier as waiting as accumulating over the entire batch obtaining input then updating. So, that is why this sigma is there, but the functional form is same.

(Refer Slide Time: 21:01)

**Backpropagation procedure**

- Create FFnet
  - $n_i$  inputs
  - $n_o$  output units
    - Define error by considering *all* output units
  - $n$  hidden units
- Train the net by propagating errors backwards from output units
  - First output units
  - Then hidden units
- Notation:  $x_{ji}$  is input from unit  $i$  to unit  $j$   
 $w_{ji}$  is the corresponding weight
- Note: various termination conditions : Error, # iterations,...

*Steps of Back prop*  
*atio*

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this I summarized here, propagation ok. So, this is nothing, all right? If you look at the steps just whatever I discussed. This is the step this is the steps whatever I discuss, this time I am written it down as a single step by step thing.

(Refer Slide Time: 21:29)

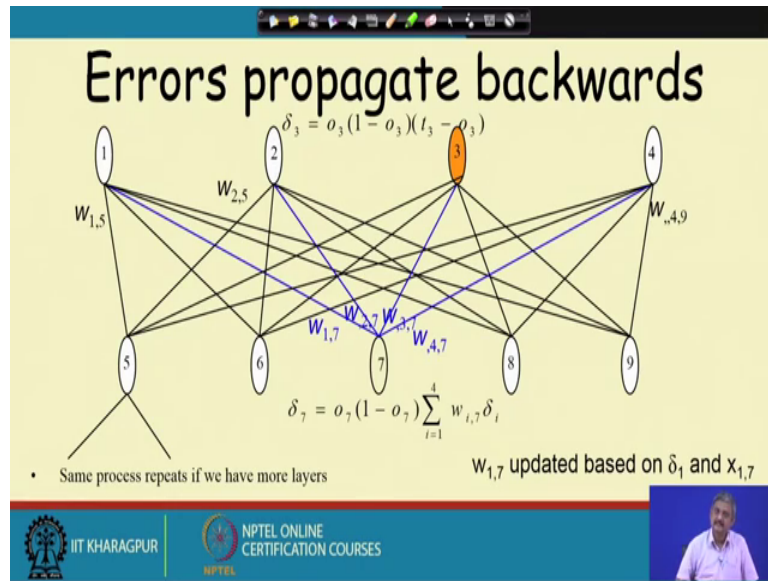
**Backpropagation (stochastic case)**

- Initialize all weights to small random numbers
- Repeat
  - For each training example
  - 1. Input the training example to the network and compute the network outputs
  - 2. For each output unit  $k$   
*delta*  $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
  - 3. For each hidden unit  $h$   
 $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{k,h} \delta_k$  *delta -> dE*
  - 4. Update each network weight  $w_{j,i}$   
 $w_{j,i} \leftarrow w_{j,i} + \Delta w_{j,i}$   
where  $\Delta w_{j,i} = \eta \delta_j x_{j,i}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

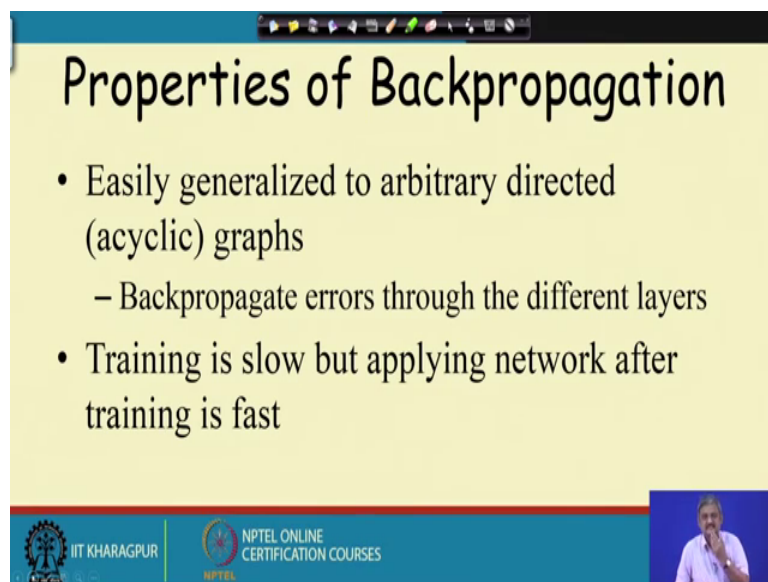
So, this is delta this actually is sometime called the called the delta value ok, and delta w. So, this is this is this delta k delta is actually the derivative we are talking about, and delta w is running rate times the delta times x ij. If you remember perceptron this delta was not there beta times x ij only was there ok.

(Refer Slide Time: 22:23)



So, here, is an example if we if we just take this inputs and propagate you will have this kind of delta values. I want you to go through this examples this is the rule I explained you should by hand work it out ok.

(Refer Slide Time: 22:47)



So, this is a very powerful rule it is can be generalized to any number of hidden layers and although some of the, one of the disadvantages if we increase the number of layer it becomes slow. There is another disadvantage is the convergence.

(Refer Slide Time: 23:06)

**Convergence of Backpropagation**

- Convergence
  - Training can take thousands of iterations → slow!
    - Gradient descent over entire network weight vector
    - Speed up using small initial values of weights:
      - Linear response initially
  - Generally will find local minimum
    - Typically can find good approximation to global minimum
  - Solutions to local minimum trap problem
    - Stochastic gradient descent
    - Can run multiple times
      - Over different initial weights
    - Committee of networks
    - Can modify to find better approximation to global minimum
      - include weight momentum  $\alpha$ 
        - $\Delta w_{ij}(t_n) = \eta \delta_j x_{ij} + \alpha \Delta w_{ij}(t_{n-1})$  ✓
        - » Momentum avoids local max/min and plateaus

*Perceptron Error Surface* (Handwritten note above a parabolic graph)

*Gradient descent will always work* (Handwritten note above the parabolic graph)

*local minima* (Handwritten note above a jagged error surface graph)

*Global minima* (Handwritten note below the jagged error surface graph)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

In the perceptron the error surface looked like this parabola. So, this gradient descent will always work. This is perceptron whereas, for the multi-layer perceptron this error circuit may look like this, these are the local minimas and the system.

Now, depending on your initial initialization random values of the  $w$ , your gradient descent may get you stuck in a, local minima that then a global minima. So, training always may not give you the least error. So, that may happen, right? So, this is a limitation of the back-propagation algorithm. There are ways to rectify it by adding something called a momentum term, which will sort of throw you out of this local minima. So, that you come here hm. So, those expressions are there.

Here is the momentum term; you can add additional term proportional to the velocity of that point. So, that it rolls down till it reaches the lowest point.

(Refer Slide Time: 24:59)

**Example of face recognition**

- Task: recognize faces from sample of
  - 20 people in 32 poses
  - Choose output of 4 values for direction of gaze
  - 120x128 images (256 gray levels)
- Can compute many functions
  - Identity/direction of face (used in book)/...
- Design issues
  - Input encoding (pixels/features/?)
    - Reduced image encoding (30x32)
  - Output encoding (1 or 4 values?)
    - Convergence to .1/.9 and not 0/1
  - Network structure (1 layer of 3 hidden units)
  - Algorithm parameters
    - $\text{Eta}=.3$ ,  $\text{alpha}=.3$ ; stochastic descent method
- Training/validation sets
- Results: 90% accurate for head pose

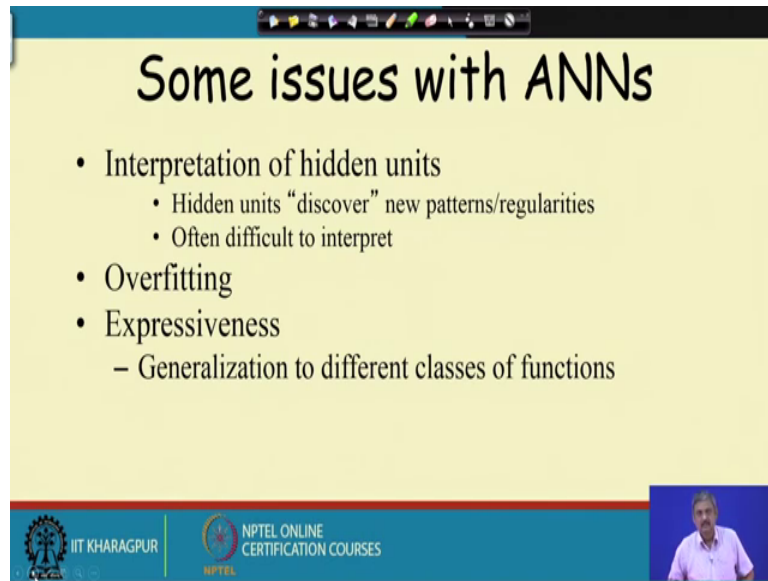
The diagram shows a neural network with an input layer of 32 nodes, a hidden layer of 3 nodes, and an output layer of 4 nodes. The output is labeled '231'. There are also some handwritten notes and a small video inset of a person in the bottom right corner.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, how to use this multi-layer perceptron? Here is an example, suppose I want to recognize people or since space biometric. So, you know images are represented as images are represented as pixels of matrix of pixel values some gray values 0 to 255. Each of these values you input to a neural network and then have hidden layer, and the output maybe is the id of the percept sorry again this type of connection may not be there it is only from one layer to the next layer ok.

And you know that this person's ID is say 231 another image comes another training example some other id, this is the desired output this is the output. You repeatedly iterate through forward propagation, back propagation forward propagation back propagation till your weight do not change. And that would give you when a new image comes it will predict which person id it is, right? So, this has been very popular in many applications this is popular and, I would request you to try out neural network on some problem. There are some issues like you cannot interpret this is connections you do not know any logical rule which gives this.


(Refer Slide Time: 26:28)



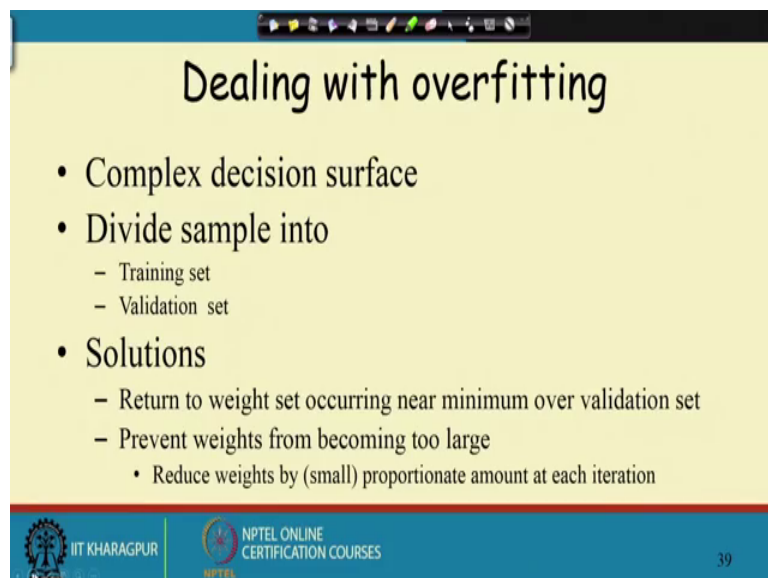
## Some issues with ANNs

- Interpretation of hidden units
  - Hidden units “discover” new patterns/regularities
  - Often difficult to interpret
- Overfitting
- Expressiveness
  - Generalization to different classes of functions

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



(Refer Slide Time: 26:40).



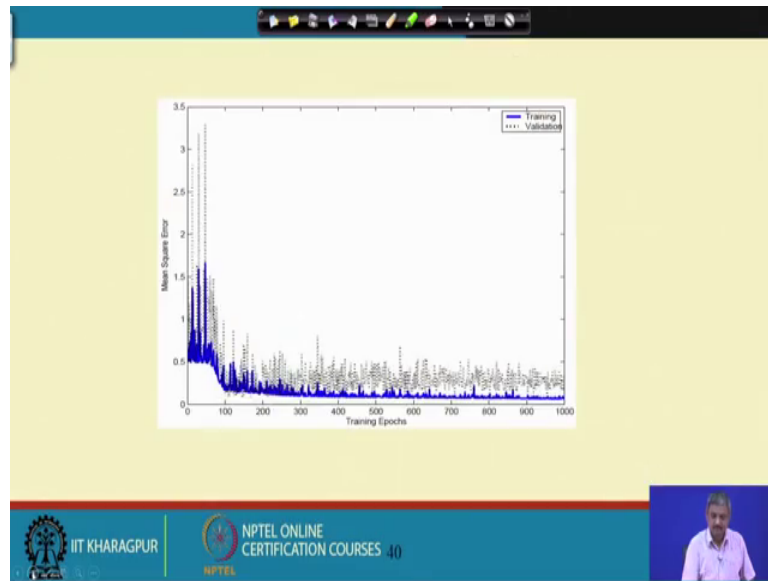
## Dealing with overfitting

- Complex decision surface
- Divide sample into
  - Training set
  - Validation set
- Solutions
  - Return to weight set occurring near minimum over validation set
  - Prevent weights from becoming too large
    - Reduce weights by (small) proportionate amount at each iteration

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

39

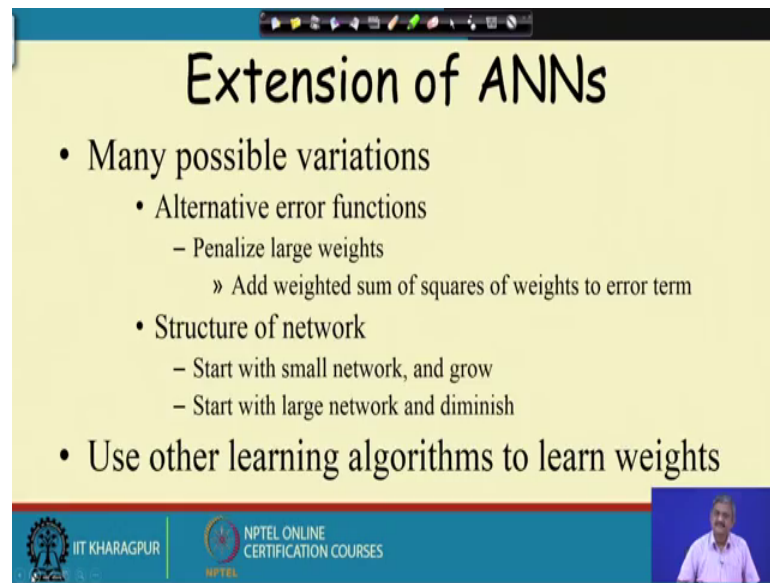
(Refer Slide Time: 26:46)



There is a problem of over fitting it is like you can there is a problem of over fitting; that means, after some time is you increase the number of hidden layers it fits the training data very well, but sacrifices generalization error ok. So, you so, the basic problem is how deep a network how many hidden layers you should take to solve the problem, or there is no real solution you have to do trial and error there are methods like regularization to control how many neurons you should have.

So, those are some problems you would face with, but it is very popular especially with complex stocks like, stock market prediction or climate prediction it is very popular ok.

(Refer Slide Time: 27:33)



The slide is titled "Extension of ANNs" and contains the following bulleted list:

- Many possible variations
  - Alternative error functions
    - Penalize large weights
      - » Add weighted sum of squares of weights to error term
  - Structure of network
    - Start with small network, and grow
    - Start with large network and diminish
- Use other learning algorithms to learn weights

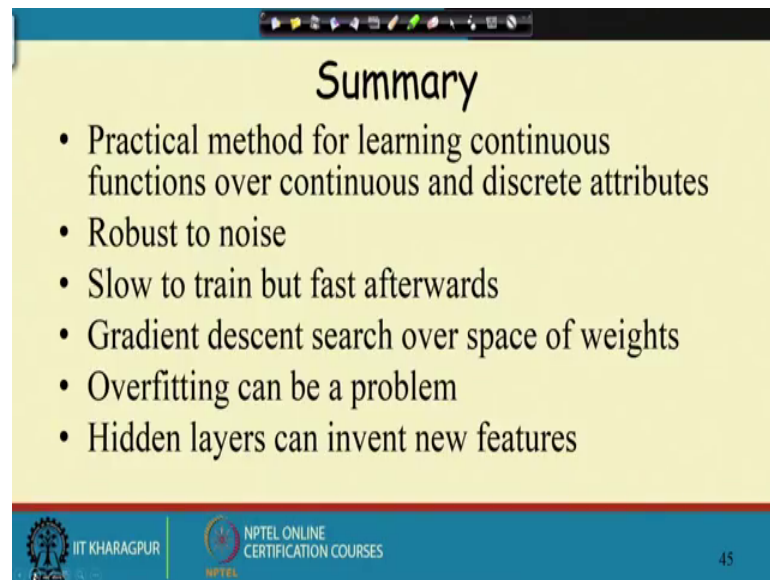
The slide footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video inset of a speaker.

So, I think these are the reasons that it has a very lot of expressive power. Many complex functions it can realize there are many extensions like, I said before backward, feedback, recurrent network. There is a self-organizing map if there are many other numerous types of neural network, but this is the most popular that we will use in data mining.

And many commercial software actually implement it. So, I request you to just write a small program to train this backward propagation for a small network on some task and do it I hope that it will help you.



(Refer Slide Time: 18:12)



## Summary

- Practical method for learning continuous functions over continuous and discrete attributes
- Robust to noise
- Slow to train but fast afterwards
- Gradient descent search over space of weights
- Overfitting can be a problem
- Hidden layers can invent new features

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 45

So, this closes my discussion on the neural network only the multi-layer perceptron and the perceptron I am not time to go to other you can read it from the notes and other sources I will give you, but it is a very popular tool.