Introduction to Machine Learning
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Lecture – 30
Deep Neural Network

Good morning, so now we have in the last two classes in neural network. We have looked at single layer and multilayer neural network, and the back propagation algorithm. Today, we will talk briefly about deep neural networks. So, this is a very new topic, and with the lot of interest, and I will just give you a brief exposure to certain very basic deep neural architectures and tell you what it is all about.

(Refer Slide Time: 00:48)

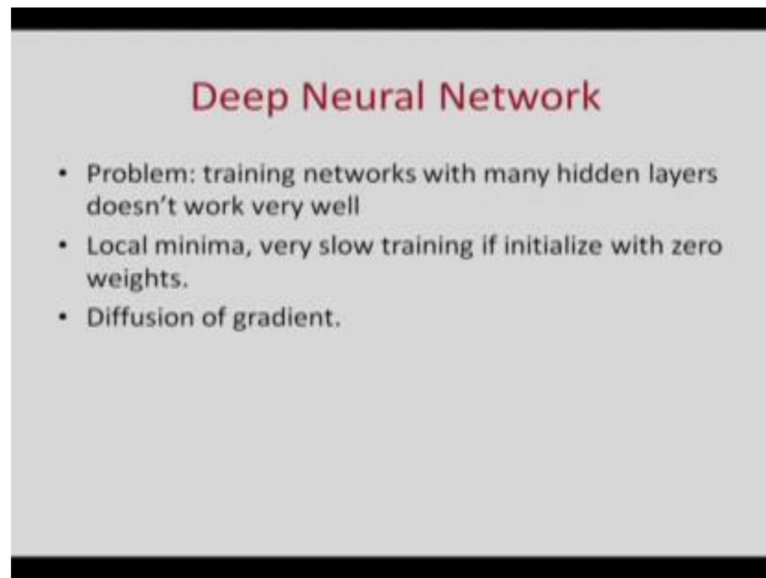
Deep Learning

- Breakthrough results in
  - Image classification
  - Speech Recognition
  - Machine Translation
  - Multi-modal learning

image audio text

In recent years there have been a lot of excitement with deep learning, deep neural networks, and there have been breakthrough results in various domains including starting with image classification, then in speech recognition; and finally, in natural language processing like machine translation multi-model learning developing text from speech, text from images automatically and many such. And the excitement continues, so I will talk briefly about what is a deep neural network.

(Refer Slide Time: 01:38)



So, a deep neural network you know very simply speaking is a network which is deep that is which has many hidden layers. We have seen that in we have feed forward networks with hidden layers, normally in a normal network what use to happen is that we also talked about in the last class that a network with two hidden layers is enough to represent any function. But it may not be easy to learn all complex functions using a network in two layers.

As you increase the layers in the network, it is easier to be able to represent complex function in terms of simple functions, relatively simple function. So, you can have a network with many hidden layers and you can call this a deep network. Why was it not an attractive proposition to start with? People found that when you have a deep network, the number of the errors surface becomes very rough and there is lot of local minima, so you cannot always converge, the convergence is extremely slow, the slow training.

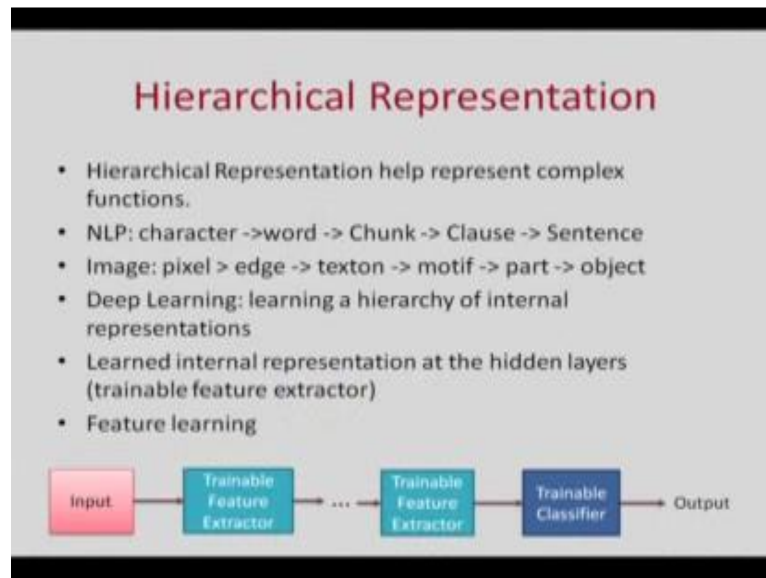
(Refer Slide Time: 03:00)

And very important there was one condition, suppose you have this input and you have the different hidden layers H 1, H 2, H 3, H 4 and then you have output. Now, we have connection from this layer to this layer, this layer to this layer, this layer and so on. Now in back propagation the error at the output was propagated here and this error the back propagation error was propagated here, here, here, here. Now there are few things most of the error based on this error, these weights where been changed. As we go further here, there would be less you know less updates of the weights in the initial layers, and one of the reasons was diffusion of gradient.

When you do gradient decent, you find the error gradient and do a and do a gradient decent, but the gradient can get very diffused, get attenuated as you go deeper from the output towards the input. When you use a function like the sigmoid function, you look at the sigmoid function, in the sigmoid function, in most of the regions gradient is almost close to 0; only in this region the gradient has some decent value.

Most of the region the gradient is close to 0. If a small value and as you are taking propagating them backwards a gradient becomes attenuated and very close to 0. And very little change was happening here. They can also be in certain cases gradient can increase very much, and there also there is a problem, which can of course, we handle by simpler methods the gradient clipping, but diffusion of gradient is a problem and the slowness of convergence is a problem, when you use a deep layer network.

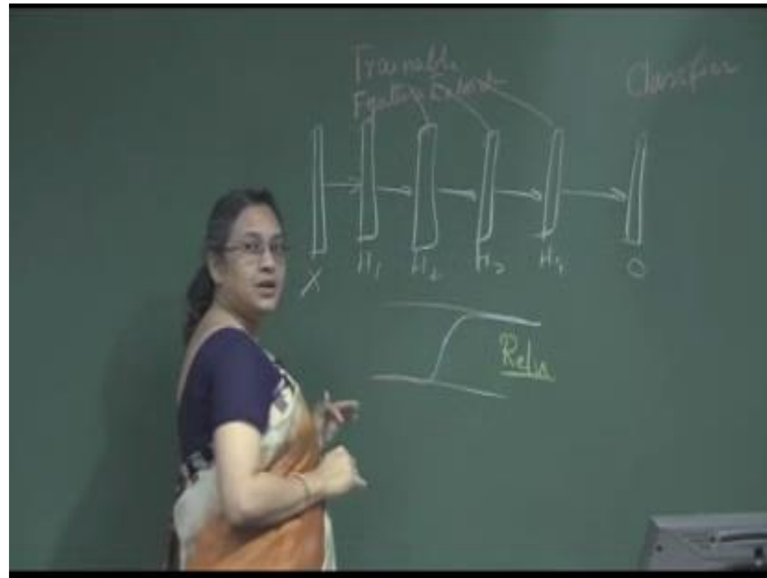
(Refer Slide Time: 05:07)



And the breakthrough that came in using deep layer network is because people came up with methods which could handle this problem in various ways some of which we will discuss today. First of all let us see why deep representation is good. In many tasks, hierarchical representation help represent complex functions.

For example, in natural language processing, we have an input sentence, and we can look at first identifying the words there is and we can look at the part of speech or morphology of the word, we can then identify chunks or small phrases then larger phrases or clauses and then the sentence. And we can process it in this order in the NLP pipeline. When you doing image processing, you can initially find out the edges in the image that from the pixels you can find the edges then the texton, then motifs, then parts then objects, and if you do it hierarchically the process may become simpler. So in deep learning, we are basically learning a hierarchy of internal representation.

(Refer Slide Time: 06:22)

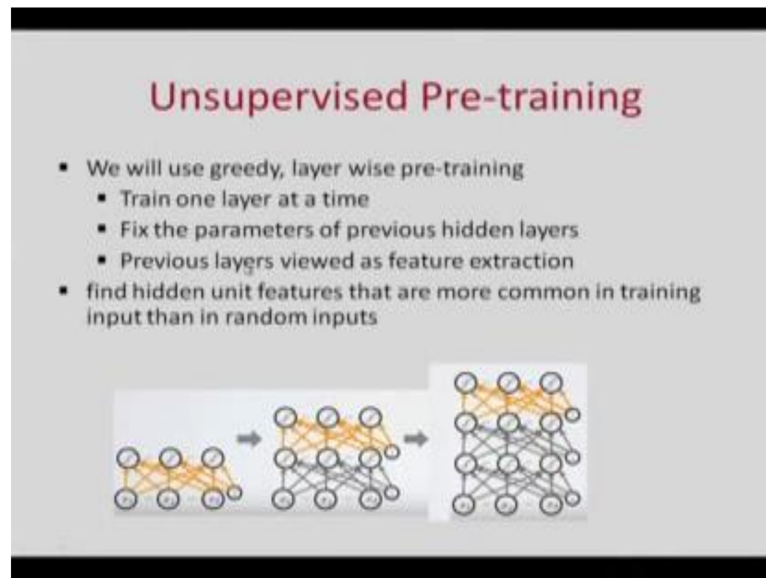


We can think of that the nodes at every hidden layer are representing certain features. So, this is the input these are some low level features, and these features are composed of this low level features, this is a further compose, further compose, further compose, so we have a hierarchy of low level to higher level features.

And the learned internal representation can be thought of as features, and we can say that these hidden layers are acting as feature extractor. So, deep learning can be looked upon as feature learning. And one way of looking at it is that we have the input, and here we have a trainable feature extractor. So, these are all feature extractors. And finally, at the output, we have a classifier, if you have a classification problem.

Now one of the ways in which deep networks can be trained is by using unsupervised pre-training. There are several ways one is that you could try to setup the units, so that the gradient diffusion or gradient blowing a problem can get reduced. And there are some ways of using activation functions other than sigmoid functions. For example, people use a function called Relu, which we will see which avoids the problem of gradient diffusion. The other approach is using unsupervised pre-training, so instead of learning a deep network at one go, we learn the network one-step at a time using a not to deep network. And by growing the layers one at a time, we can avoid the problem which deep networks have.

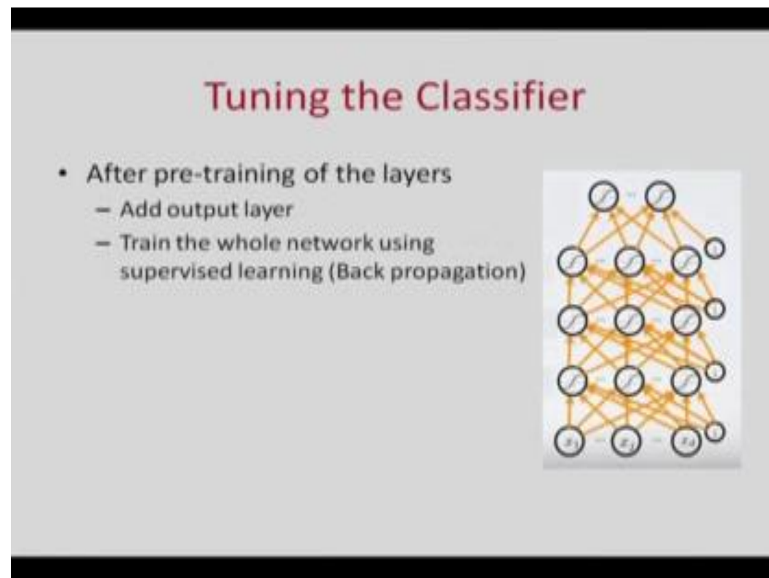
(Refer Slide Time: 08:47)



If you look at this slide, we show the schematic of doing greedy layer wise pre-training. This is the input. We first learn the first hidden layer by using some mechanism; one of the mechanisms is auto encoder, which I will discuss briefly today. Another is by using restricted Boltzmann machine. So, by several such approaches is possible to first learn this layer.

When you learn this layer, you fix this layer; after you fix this layer in terms of this you learn the next layers, then you fix it, and then you learn the next layer and so on. So, here you are fixing the parameters of the previous hidden layers, and you are assuming that these are different features, and then you are trying to find the next layer. So this is the idea of greedy layer wise pre-training.

(Refer Slide Time: 09:44)



After you have learnt the layers individually, finally you can look at the classification problem. And based on the training example, you can setup the classifier at the last layer of the output layer and then can tune the whole network with back propagation. Initially, you do greedy layer wise training; finally, you put the classification problem using the training examples and then tune the entire network. So, this is one scheme which is followed by stacked restricted Boltzmann machine and stacked auto encoder.

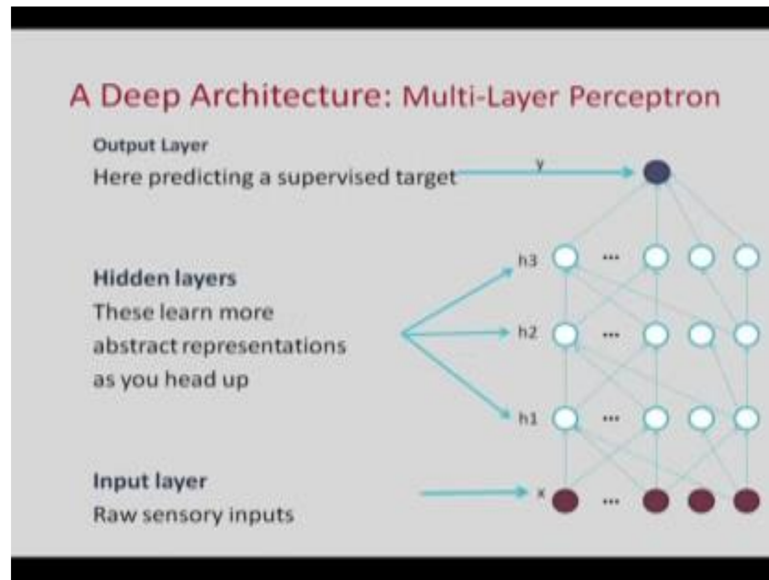
(Refer Slide Time: 10:25)

Deep neural network

- Feed forward NN
- Stacked Autoencoders (multilayer neural net with target output = input)
- Stacked restricted Boltzmann machine
- Convolutional Neural Network

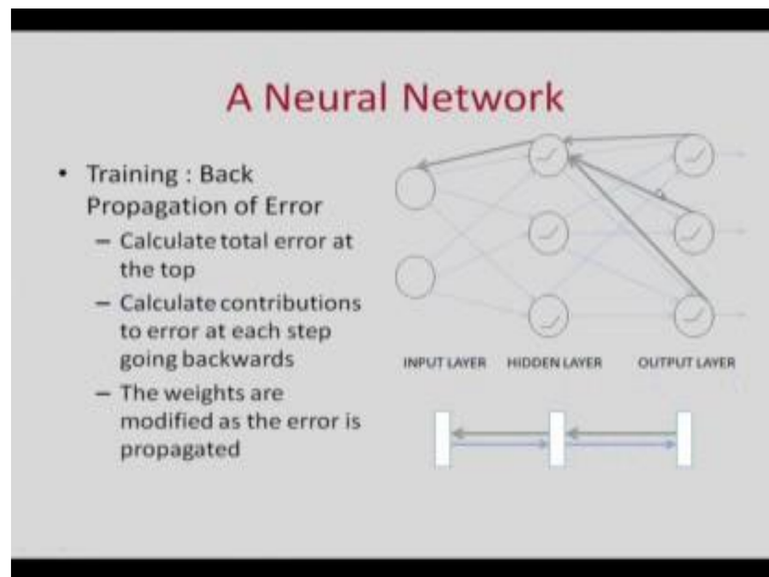
So, in deep neural network, so what we will look at is feed forward neural network. Stacked Autoencoder which I will talk about briefly; stacked restricted Boltzmann machine will not talk about in this class, but that is another approach and very briefly we will talk about Convolutional Neural Network.

(Refer Slide Time: 10:45)



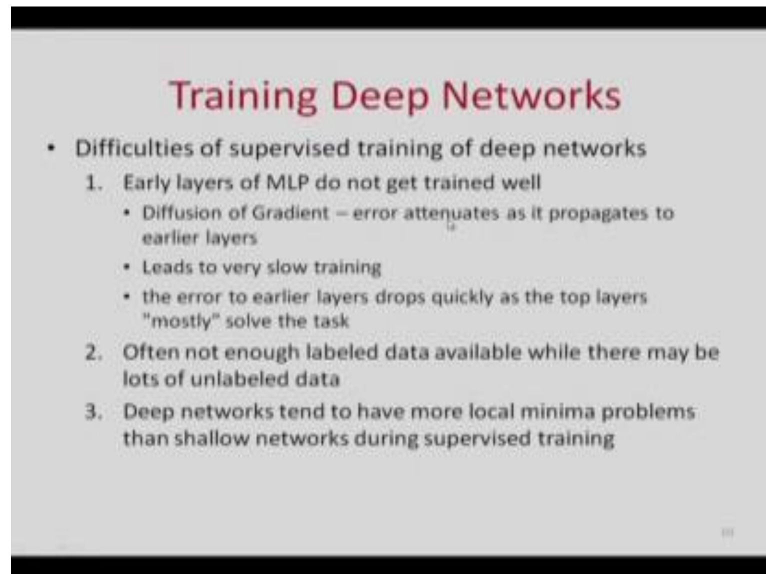
So, this is an example of a deep architecture a multilayer perceptron. We have the output layer. This is the input layer comprising of raw sensory input. These are three hidden layers which learn more abstractive representations as you go upwards.

(Refer Slide Time: 11:05)



And finally, this is the output. Now we have seen in a neural network, there is a back propagation of error. We have already talked about it.

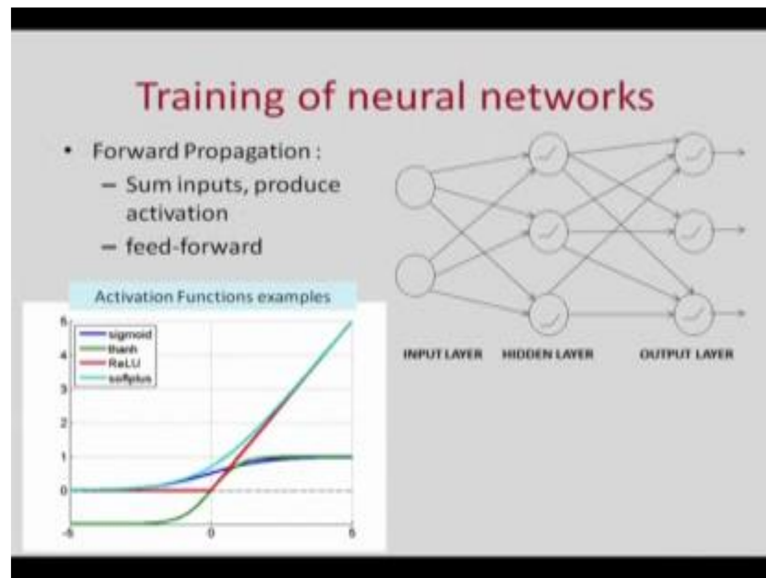
(Refer Slide Time: 11:13)



And when you train deep networks, then there are some problems. The early layers of the network do not get trained well, because there is a diffusion of gradient error attenuates as it propagates to earlier layers. It has very slow training and the error to earlier layers drops quickly as the top layers mostly solve the task so not much update happens in the beginning layers.

And there is often not enough labeled data. If you have a deep network, it will have many parameters. When you have we have seen that if you have too many parameters too few training example then it may lead to over fitting. However you can take advantage of unlabeled data, instead of using only labeled data which may be restricted in number, you can take advantage of unlabeled data, and this is what some of this approaches to. Deep networks tend to have more local minima problems than shallow networks. So, these are some problems with deep networks.

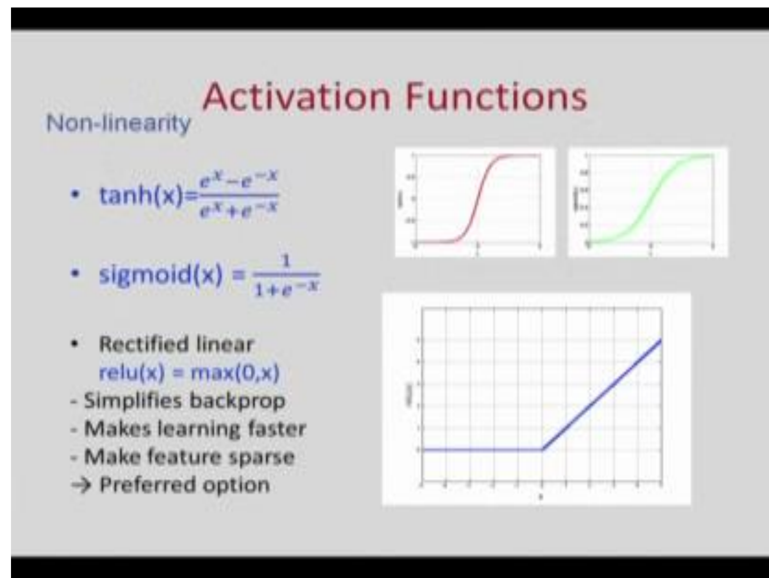
(Refer Slide Time: 12:22)



Now as I said that we have talked about the sigmoid function, there is also the tanh function, Relu function, softplus function, these are some other activation function. So, this blue line is the sigmoid function which have discussed; it is between 0 and 1. The tanh function is also an h shaped function similar to the sigmoid function; it is ranges from minus 1 to 1. The Relu function the red function Relu function is 0, when the input is less than 0; and after that it is a linear function. So, this is a Relu function.

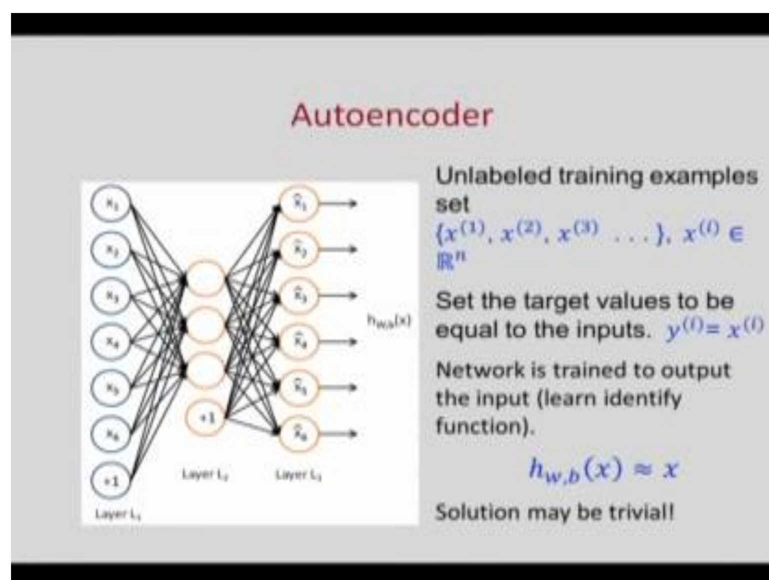
And this light blue function is the soft max function, which is a smooth function similar to the Relu function. And these functions are unbounded when the input is high they becomes very large and these functions have recently been used to overcome the diffusion of gradient problem in deep neural networks.

(Refer Slide Time: 13:28)



So, the tanh sigmoid function we have already seen. Tanh function is given by this equation. So this is the tanh function, this is the sigmoid function, and this is the relu function which is given by max of 0 x. It simplifies back propagation makes learning faster. And this is often a preferred option when you have many hidden layers in neural network.

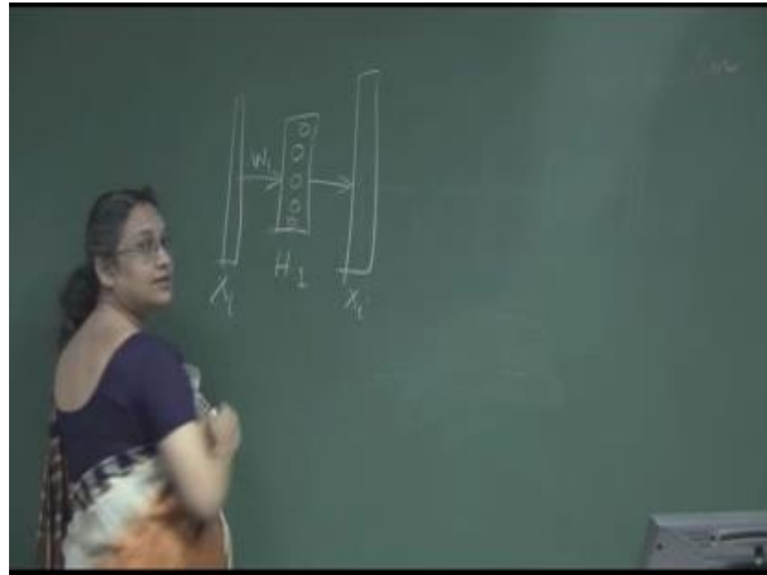
(Refer Slide Time: 13:56)



Now, let us very briefly talk about the Autoencoder. Suppose, you have unlabeled training examples x 1, x 2, x 3 etcetera, which are all real value of vectors. Now in

autoencoder what we will do is that we will learn the initial hidden layers based on the unlabeled units only unlabeled inputs only.

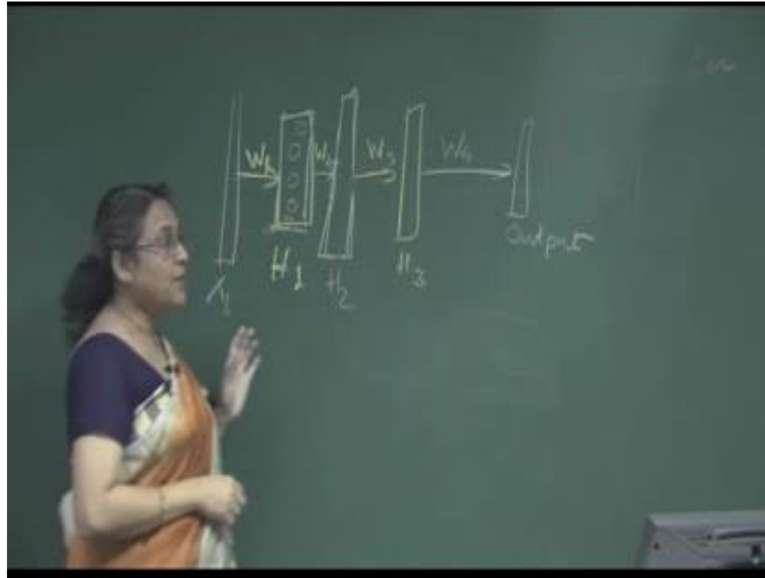
(Refer Slide Time: 14:27)



How do we set it up? Suppose, this is an input X i, this is my hidden layer H 1, we will set up a problem where X i is the input and X i is the output. So, we will setup a network to learn the identity function. And we will have connections from here to here, here to here. So, you can see that this H 1 will act as a representation of the input because from H 1 the input can be reconstructed.

Now you may be tempted to think that we can have a very simple trivial function, we just takes each unit and transfers in there, but we want to prevent it using some constraints. We want to put some constraints, so that H 1 learns some interesting function at the nodes; and not a trivial function. This function will be a representation of the input, but they will be nontrivial by using some constraints. We will discuss about this. Thus, once we learn this autoencoder, we have basically a neural network with one hidden layer; this is easy to train and then we fix the weights, so we fix these weights W 1.

(Refer Slide Time: 15:57)

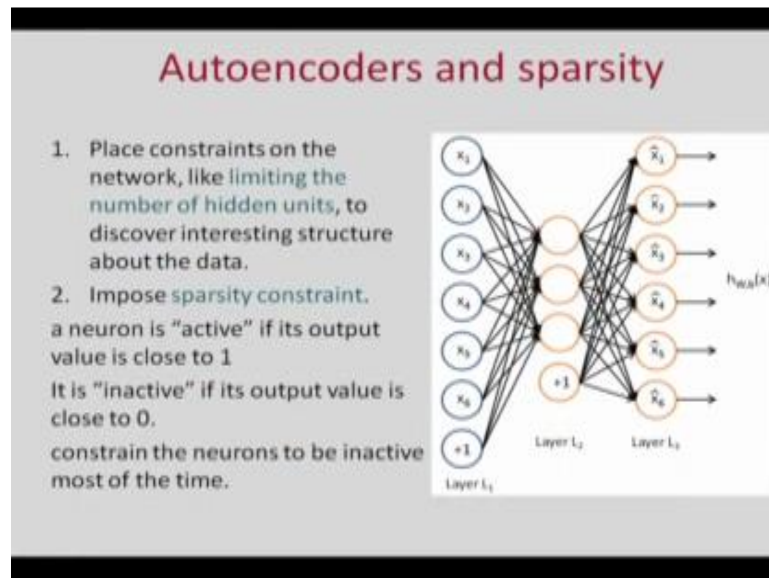


Now we remove this. And now we think of given the input we have this value of H 1 and we setup another learning problem where we have the hidden layer H 2. So this H 1, we have fixed; this W 1 we have fixed, so this we have fixed. And we setup H 2, and we put H 1 at the output. So, for every training example we find the value of H 1 and the same value we put here; we generate this training example and then we learn this weights. So, these weights let us call them W 2. And after that we will fix this layer and this weight and we will remove this. So, we will learn layer by layer.

Similarly, we will learn H 3 and W 3. Finally, after we have finished learning this unsupervised layers in a layer wise approach now with respect to each training example will find the value H 3 and we will bring in the output. And for the output, we will train a two layer neural network or some other function like SVM or a linear regression, we will train some function. And then we will get these weights W 4.

So up to this, we are using only an unlabeled data, here we bring in the labeled data, and then we will get a deep neural network. And if you wish, we can tune this network by feeding in the input output term here. So, this is the idea of autoencoder. So, we have unlabeled training examples we set the target values to be equal to the inputs. So, we have x 1 here we have x 1, x here, x here, x here. And then we train the first hidden layers, so that the hidden layer represents x.

(Refer Slide Time: 18:20)



Now as I said the solution to this autoencoder may be trivial. And we want to put some constraints in order to prevent it. One of the constraints could be that in order to prevent a trivial mapping, we can set the number of nodes in the hidden layer to be much smaller than the number of inputs. And we can constraint that suppose the input has 100 dimensions, we can put 20 dimensions in the hidden layer.

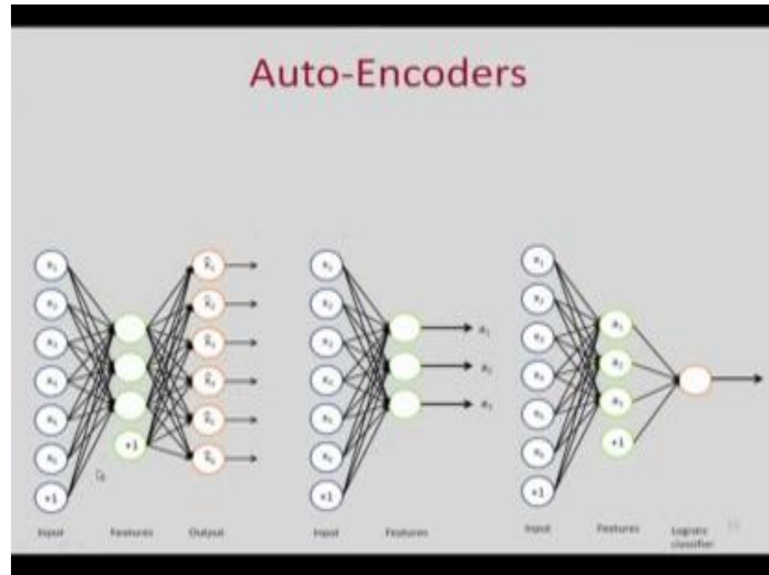
And if a function can be represented by 20 dimensions that is a nontrivial function, so that could be one constraint we put. Another very popular constraint that we use is the sparsity constraint. In sparsity constraint, first sparsity constraint first of all with respect to one neuron, we say that the neuron is active if its output value is close to 1, if its output is close to 1.

Now among all the training examples, for some of the training examples, the neuron will active. We can put a constraint on the fraction of training examples on which the neuron can be active, so that is we constraint the neuron to be inactive most of the time, and to be active only for some training examples, which have certain commonalities.

So, we can impose this sparsity constraint on the these network, and we can learn the weights which satisfy this sparsity constraint that is preventing the trivial mapping to be learned this is another way of imposing a constraint. And we are as we saw that we are stacking different layers. It has been found that this spars autoencoders can be stacked

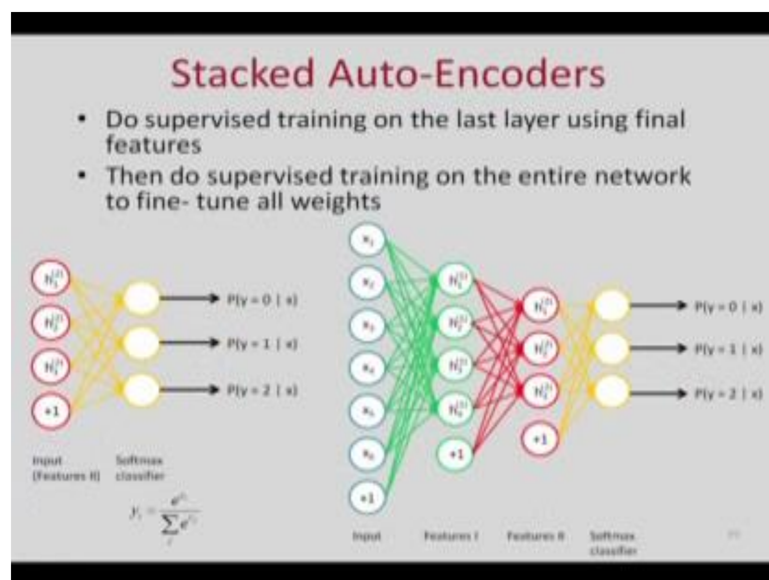
easily, whereas the previous constraint where you can strain the number of hidden units, they do not after you know they are not so much amenable to stacking.

(Refer Slide Time: 20:31)



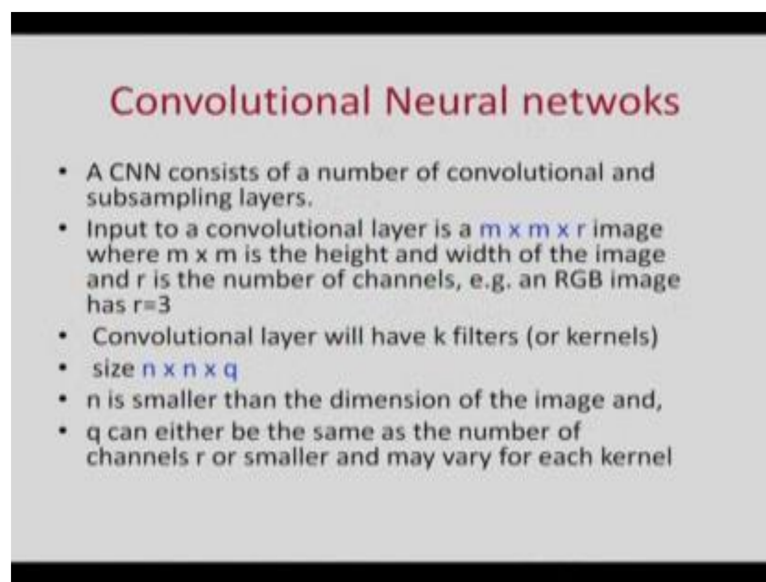
So, we can add such constraints, and then we can put the input here, input here, learn the first hidden layer. We can fix the weights from the input to the first hidden layer. And then we can after fixing the weights then at the end we can learn the classifier, of course, we can do more than one hidden layer.

(Refer Slide Time: 20:54)



And by stacking autoencoder we can use more than one hidden layer. This is the first hidden layer then, there is a second hidden layer. So, we do supervise, so we do layer by layer using unlabeled examples. Finally, we do supervise training on the output layer and then we tune the entire network. So, this is the idea of stacked autoencoder and this is one way of learning a deep network, where we avoid the problems the standard problems that is where they are with deep networks. There is other as I said we have stacked RBM, where we use energy minimization to learn layer by layer using bidirectional neural network, but that we will not discuss in this class.

(Refer Slide Time: 21:37)

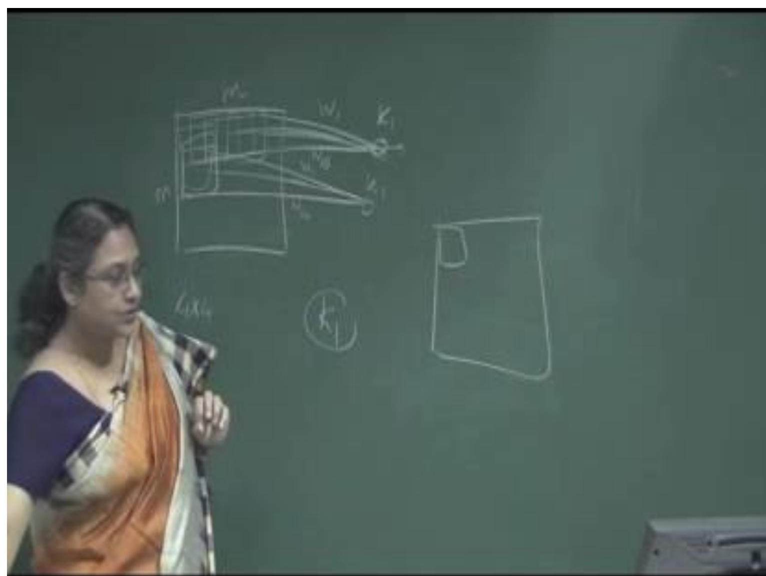


Now we will briefly talk about another approach to deep neural networks using convolutional neural network. Convolutional neural network has been very popular for working with images. And now it has also been applied to other radius including natural language process. So, convolutional neural network consists of a number of layers, and these layers have some specific functions. They are two types of layers, which are very popular convolutional layer and subsampling or pooling layer. The input so convolutional layer is an image.

(Refer Slide Time: 22:41)

Suppose, you know it could be another thing, but let us talk about an image. Suppose, there is an image, which consists of m by m pixels; and for each pixel, we may have a number of channels. For example, if it is an r g b image, there are three channels. So the number of inputs corresponding to the image is m by m by r. So, these are the different channels - r g b, so this is the image. And the convolutional layer we use k filters, so use k functions; and each of this functions will do a mapping. So, let us explain how this functions convolution function works.

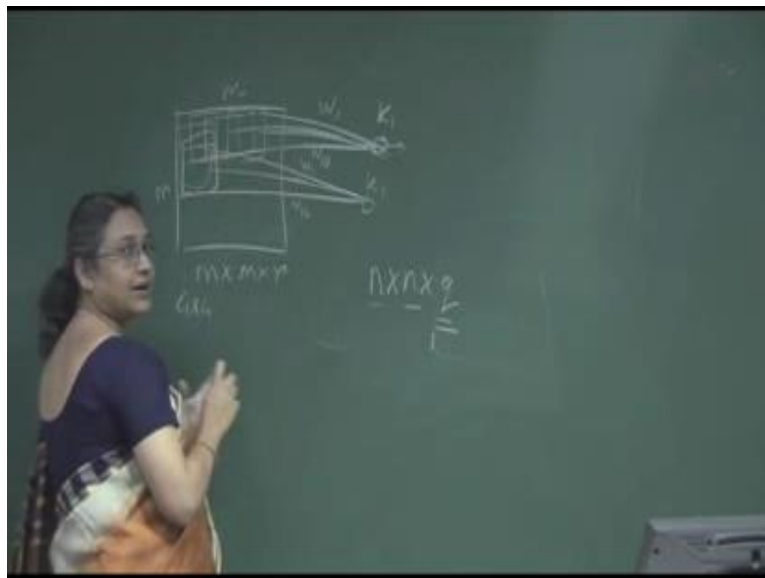
(Refer Slide Time: 23:37)



For simplicity, let us assume we have m by m image and there is only one channel. Now in convolution what we will do is that we will take a sub rectangle. Suppose, we take 4 by 4 sub rectangle, then we can take different 4 by 4 sub rectangles. Now corresponding to this sub rectangle we have 16 inputs, and this 16 inputs will feed to this corner this is k 1 is one feature. And if there are w 1 to w 16, these are the 16 weights, so this is an output.

Now what we will do is that for other sub rectangles also we will go to this function K 1, and they will also share the same weight W 1, W 2, W 16. So, what we do is that in convolution for every node, we will for every rectangle we will so for a particular kernel that we are using for every rectangle we will find the function. And this function the weights of this function will be the same weights as we are using for all these sub rectangles. So, this is the convolution. So, we are taking a mass and convolve it to every position of the image and we get the convolution output.

(Refer Slide Time: 25:47)



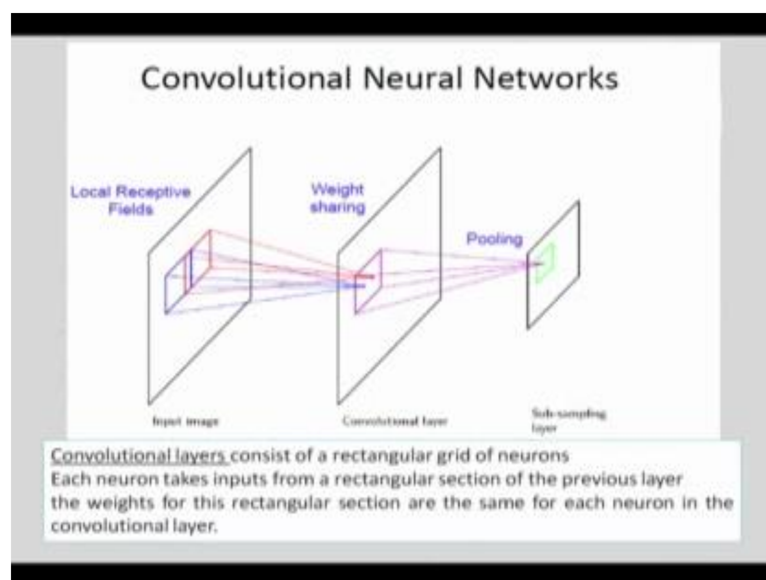
Now, because the weights are shared the number of parameters is less. So, you can think of you are trying to find particular pattern in the pattern could be in this place of the image or this place, or this place, or this place. So, this is the convolution layer. And the output of the convolution layer is n by n by q, so n can be equal to m or less than m. And q is the number of channels; here r is the number of channels.

And we can have instead of looking at a sub rectangle, we can think of a volume, we can look at a volume to the convolution and get the output, which is n by n by q. And we can have several such filters which we call kernels.

(Refer Slide Time: 26:20)

This is one function K 1; we can have another function k two which will have a different set of weights.

(Refer Slide Time: 26:26)



So, we can look at this picture here. So, we have this sub rectangles or sub volumes. And for one convolution layer, we are computing this function; and this function will share

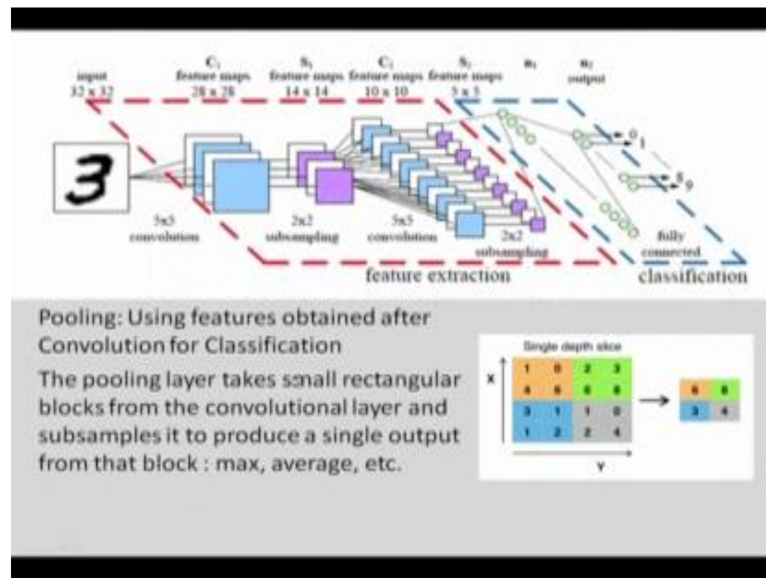
the weight between the different local fields. Next, what we can do is that we can do a pooling; we want to know if this particular pattern occurs anywhere in the image.

(Refer Slide Time: 27:01)

So, we can take for K 1, we will have output from different parts of the image then some these outputs can be combined using max or average. If you do max, it means max pooling. Suppose, we take this region suppose this filter corresponds to if an edge is there now that edge can be here in the image, here in the image, here in the image. So, if you take max, it means does the edge occur anywhere in the image.

So, we can do some pooling and so that we can take this n by n by q, and reduce it further. So, pooling or subsampling, what it does, it takes the output of a function and combines them into smaller value. So convolutional layer consist of rectangular grid of neurons; each neuron takes input from a rectangular section of the previous layer, the weights of this rectangular section and the same for each neuron in the convolutional layer.

(Refer Slide Time: 28:05)



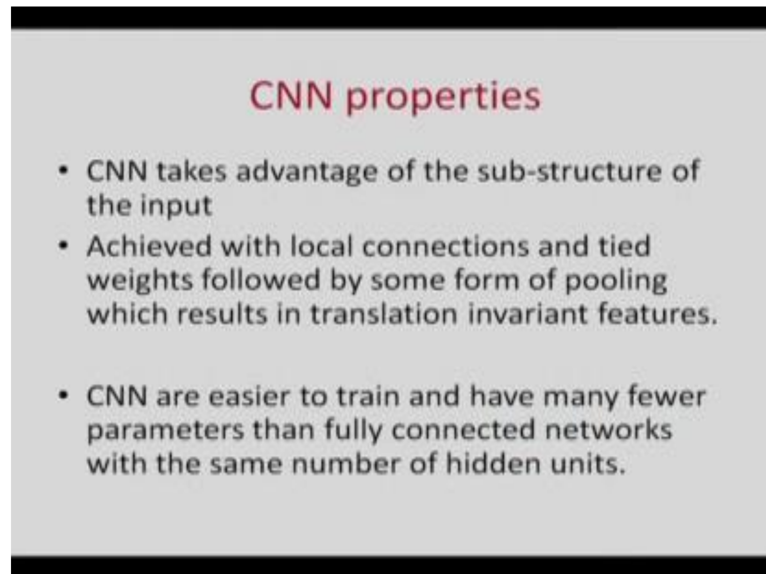
Now, in pooling, we use features obtained after convolution. The pooling layer takes small rectangular blocks from the convolutional layer, and subsamples it to produce a single input from that block. For example, after convolution, these are the values that you get. Now we can divide it into small sub rectangles, and for each rectangle we can do a pooling. For example, from this sub rectangle we by max pooling they get six; from this max pooling gives us 8; from this max pooling gives us 3, from this max pooling gives us 4. So, after pooling we reduce the image size from this to this.

So, here you know this is a digit learn you know recognition of hand written digits task. We have an image, we do a convolution and we use five features. So initially input images 32 by 2, we get 5 28 by 28 images after convolution. Then we do 2 into 2 subsampling, so we take a 2 into 2 neighboring rectangle doing max pooling and then reduce it to 14 by 14 then we again do a convolution 5 by 5 convolutions, and we get 10 by 10 feature maps and again do subsampling.

So, we can have a convolution and subsampling layers and then they can be stacked together. We can also have optionally some layers, which do fully connected feed forward neural network like we have seen earlier and finally, we have the classification. So, this is a one of the architectures one configuration of a convolutional neural network which consist of convolution layer, pooling layer, fully connected layer. Of course, the

convolutional layer will be there and there these layers will be stacked together pooling may or may not be there and the fully connected layer may or may not be there.

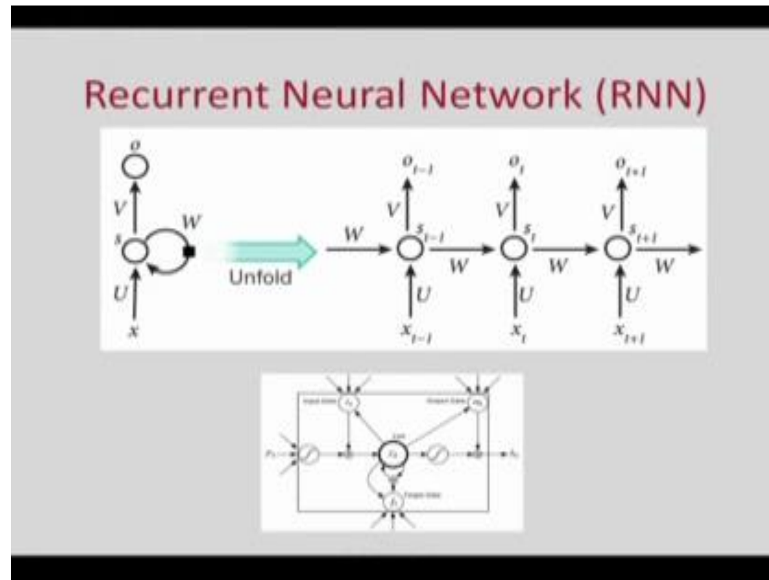
(Refer Slide Time: 30:11)



So, let us look at some properties of convolutional neural network. The convolutional neural networks are takes advantage of the sub-structure of the input, which is achieve with local connections and weights. So, locally, the convolution is over a local rectangle. So, it looks at some feature in a local region; and the same feature it tries to find in different regions and this is achieved by tying the weights by using the same weights for this rectangle as well as this rectangle, we use the same weights.

So, this results in translation in variant features. Convolutional neural networks are easier to train and they have many fewer parameters than the corresponding fully connected network, because the weights are shared. So, this makes convolutional neural network easier to learn.

(Refer Slide Time: 31:13).



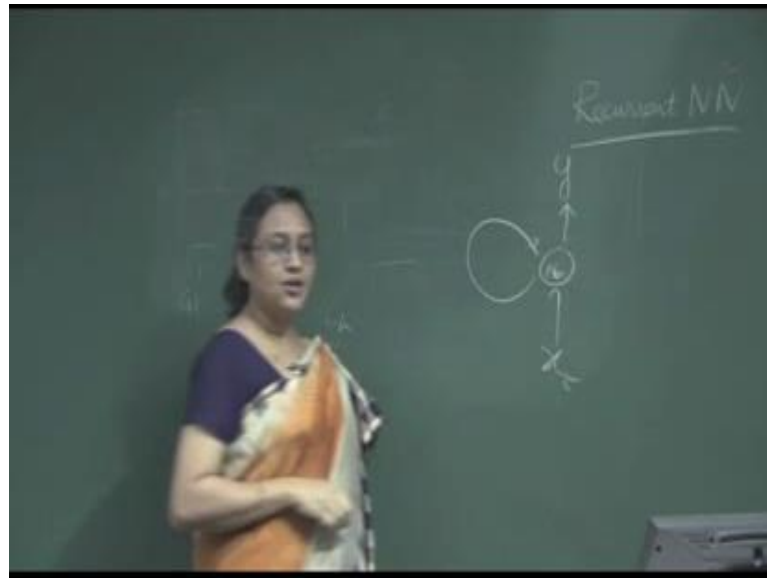
Then there are many other neural deep neural network architectures, of course, we will not be able to cover everything. I will just mention recurrent neural networks, which are very useful for representing temporal sequences as in speech or sequences of words in a sentence, videos etcetera.

(Refer Slide Time: 31:36)

In recurrent neural network, what we have is that we have input and we have a output and there is a hidden layer. Now we want to capture the input from the previous time steps. Now one of the ways in which we could do is that we could have the input at x t x

t minus 1, x t minus 2 dot dot dot x t minus k. So, we could pick a fixed size window and treat it as the input and then we have a hidden layer, and we could have the output. These constraints as from taking input from a sliding window. Alternately, we could take the input theoretically from anytime in the past infinitely back in the past by having a connection from H to H.

(Refer Slide Time: 32:49)



So this is x t, this is h t, and we can have a connection from h to itself, which corresponds to if we look at this picture which corresponds to you can unfold it and have this is x t minus 1, x t x t plus 1 and this is the hidden state at t minus 1, t, t plus 1. And this weights are shared this is the common weight. This is the common weight and this is the weight u, this is the weight v, this is the weight w, and this is the recurrent neural network. And you can unfold it infinitely to get a very deep neural network. And you can use back propagation the particular term we use is back propagation overtime to find the value of the weights.

Now recurrent neural network have similar problem as deep neural networks have with respect to back propagation. And one of the very nice ideas people have come up with is use some units instead of using simple perceptron, we use certain gates units by which one can store which can act as a memory, where one can store some information, so that that information stays till when you want to use it. So, information which is long back in

the past can be stored using this gated units, and there are various such units which have been used.

(Refer Slide Time: 34:34)

LSTM, which stands for long short term memory; and GRU, which stands for gated recurrent unit these are some of the units which are used in recurrent neural network to enable them to work effectively.

Unfortunately, today we do not have much scope in this class to talk about this and this is a little more complex topic which we will not have time to go into detail, but I just wanted to give you a glimpse into this, so that in future you can read it. This entire topic of today's is a little bit advanced, but we wanted to tell you a little bit about this there are many are very nice architectures of deep neural networks like encoder, decoder architectures, and there are a different models using models which use external memory. And these are very exciting and they have been used for solving extremely interesting tasks. I hope that you will have some interest and be able to study them later.

Thank you.