

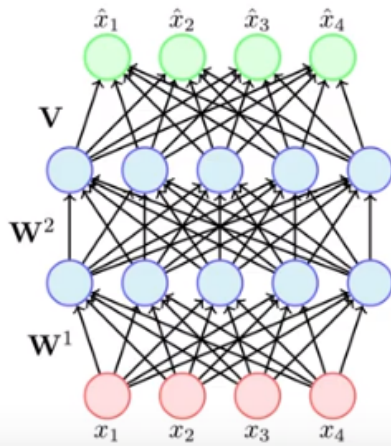
Lecture – 21.2
Masked Autoencoder Density Estimator (MADE)

Refer Slide Time (0:14)

Module 21.2 : Masked Autoencoder Density Estimator (MADE)

Now, the next thing that you're looking at is made, which is masked. Okay? Sure! So, what you're saying is, that I have assumed a certain ordering of the random variables. But that ordering can be different. So, it is up to you. Right? So, you could so what you're saying is, that I am assuming, that X_K depends one very thing from I to K minus 1. You want it to depend on a neighbourhood. Right? both of these are assumptions. Do you agree with that? and the first whatever I have assumed Right? I have assumed a certain ordering of the random variables; you could assume a different kind of ordering. So, that this factor becomes one of the factors in the distribution. Does that make sense? It's not trivial, but you could do that. Okay? and also, I mean I know in when we're doing conversation here and in networks we argued, that it only depends on the neighbourhood and so on. But we have also seen other cases where it could depend on something at the top something at the bottom and so on. So, this is a modelling choice which you are making. Once you make that choice these questions are not really do not really exist Right? I mean you have made a choice that is going to depend on all the previous pixels, if that's a choice you can use this if you're not happy with that choice you'll have to do something different. So, then you have to talk in terms of whether the KL divergence between the true and the approximate is similar or something, yes. It's again an approximation. Just as variational inferences an approximation just as good sampling is an approximation this is also an approximation. Yes. So, you're looking for some theoretical guarantees on the true distribution versus this. I am not aware of those. If I don't think the origins of paper also has it but, you can go and check but I don't think there is such a guarantee given on that. So, this is masked auto-encoder density estimator. Density estimator is always fine with us. Now, given that auto-encoder appears in the name. We're going to start with an autoencoder and then try to come up with a density estimator by modifying it. Okay?

Refer Slide Time (2:03)



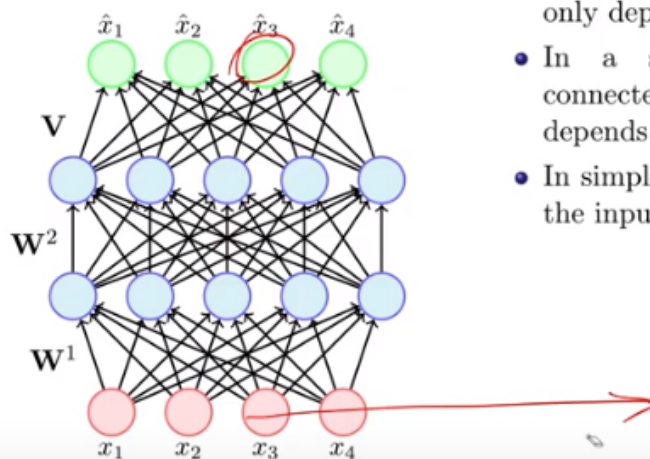
- Suppose the input $\mathbf{x} \in \{0,1\}^n$, then the output layer of an autoencoder also contains n units
- Notice the explicit factorization of the joint distribution $p(\mathbf{x})$ also contains n factors

$$p(\mathbf{x}) = \prod_{k=1}^n p(x_k | \mathbf{x}_{<k})$$

- **Question:** Can we tweak an autoencoder so that its output units predict the n conditional distributions instead of reconstructing the n inputs?

So, suppose the input is again n -dimensional, then what's the output of an autoencoder what's the dimension of the output, and it's an input is n output is n . Now, coincidentally how many factors do we have in the explicit factorization, or the chain rule n . Right? So, the question that I'm going to ask you now is, that can we somehow tweak the auto encoder. So, that instead of reconstructing the input using its n output units, it can instead predict these n probability distributions, that I am interested. Is that fine. Okay? So, that's the question, that we are asking can we take a regular auto encoder tweak it in a certain way. So, that instead of predicting or reconstructing it the input. it predicts these n factors from our joint probability distribution. Okay?

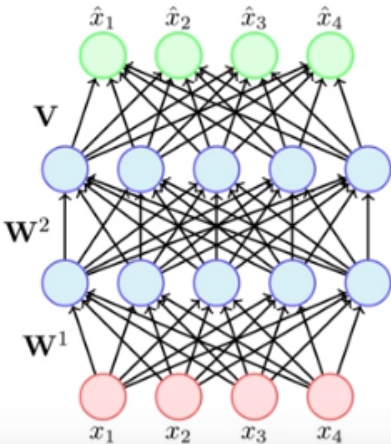
Refer Slide Time (2:52)



- Note that this is not straightforward because we need to make sure that the k^{th} output unit only depends on the previous $k-1$ inputs
- In a standard autoencoder with fully connected layers the k^{th} unit obviously depends on all the input units
- In simple words, there is a path from each of the input units to each of the output units

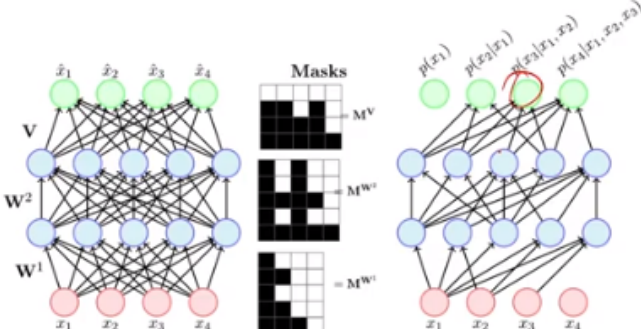
So, what are these outputs going to be in that case. This is going to be $P(x_1)$, $P(x_2)$ given x_1 , $P(x_3)$ given x_1, x_2 and so on. Right? Now, why is it not straight forward to do this with the regular auto encoder. Because, it is fully connected. So, if it's fully connected every output unit sees every input unit and that's not fine that's cheating. Because you are trying to predict x_3 and you're already seeing x_3 as well as everything that follows it, also. Right? So, we will have to do something to make sure this auto encoder is not fully connected and we have to make sure, that when I'm looking at x_3 there is no path to x_3 from x_3, x_4 onwards up to x_n there can be only paths to x_3 from x_1 and x_2 . Does that make sense? Very same to what we had done in the made architecture. Does that Okay? the same idea.

Refer Slide Time (3:47)



- Note that this is not straightforward because we need to make sure that the k^{th} output unit only depends on the previous $k-1$ inputs
- In a standard autoencoder with fully connected layers the k^{th} unit obviously depends on all the input units
- In simple words, there is a path from each of the input units to each of the output units
- We cannot allow this if we want to predict the conditional distributions $p(x_k | \mathbf{x}_{<k})$ (we need to ensure that we are only seeing the given variables $\mathbf{x}_{<k}$ and nothing else)

Refer Slide Time (3:49)

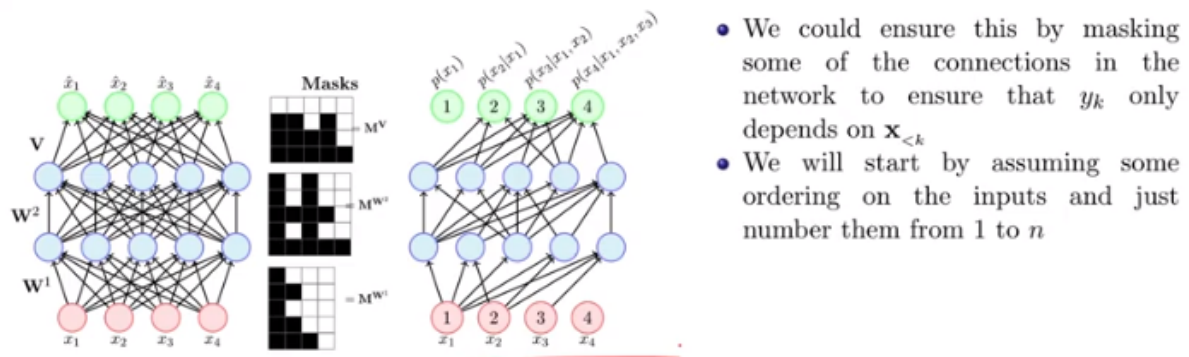


- We could ensure this by masking some of the connections in the network to ensure that y_k only depends on $\mathbf{x}_{<k}$

and so, what we're going to do is, that we are going to use masking again, which means, that we are going to mask out some connections. So, that our output every node in the output is only connected to

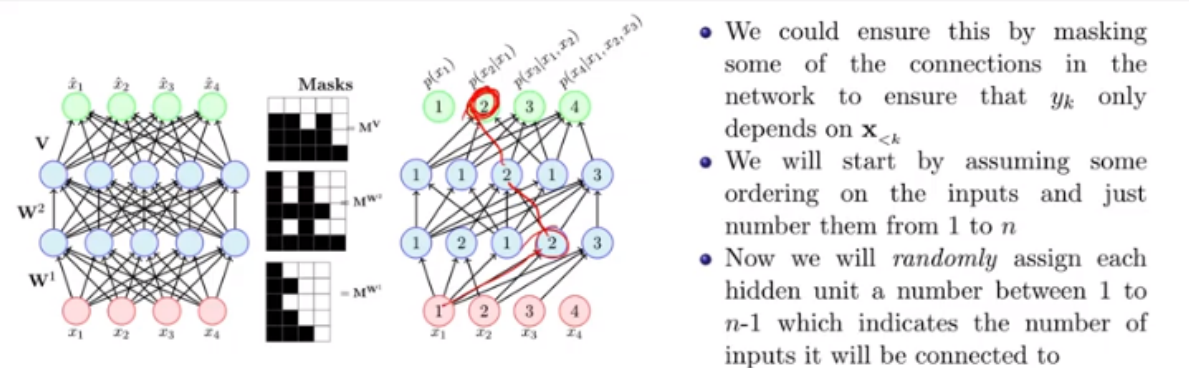
the relevant inputs, and what are the relevant inputs, the ones which appear on the given side. So, it's X3 given X1 X2 so X3 should only see X1 and X2. Does that make sense everyone? Okay?

Refer Slide Time (4:16)



So, what we'll do is we'll start by assuming some ordering on the input. So, we will assume, that your inputs are order 1 to n Okay? say in the case of image as a straight forward you can just go from left to right, and then top to bottom, and that's your ordering. Okay? So, I'm going to number these pixels 1 to 4 1 to n , and of course the outputs are also numbered 1 to n because it's X1 probability of X2 probability of X3 and so on. Okay? Does this numbering make sense?

Refer Slide Time (4:44)



Now, what I'm going to do is, I'm going to assign random numbers between 1 and n minus 1. to every node in the hidden layer. Okay? What does that mean? What Willi do with this numbering can you take a guess? What would I want with the node number 2? It should only look at those inputs which are less than equal to 2. Okay? Why what will then happen if I connect this part Okay? So, Now, let's see, if I have the output node 2. What would I want the numbering of all the nodes, which appear on the path from this output node to the input nodes less than or equal to 2? no sorry actually less than 2 less than 2 Right? So, does that make sense. So, that's where we are headed as we are going to number these nodes make sure, that the number on the node defines, how many inputs is going to

connect it to or rather, which are the inputs that it is seeing, and from there we can connect a path from the output to the input such, that the given output sees only the relevant inputs is the overall idea clear to you. Okay?

Refer Slide Time (5:53)

The diagram shows a neural network with three layers: an input layer with nodes x_1, x_2, x_3, x_4 (red circles), a hidden layer with nodes W^1, W^2 (blue circles), and an output layer with nodes V (green circles). To the right, three binary masks M^V, M^{W^1}, M^{W^2} are shown, each with a specific pattern of black squares. Further right, a probability distribution is shown for each node in the hidden layers, with probabilities $p(x_1), p(x_2|x_1), p(x_3|x_1, x_2), p(x_4|x_1, x_2, x_3)$ and node numbers 1, 2, 3, 4.

- We could ensure this by masking some of the connections in the network to ensure that y_k only depends on $\mathbf{x}_{<k}$
- We will start by assuming some ordering on the inputs and just number them from 1 to n
- Now we will *randomly* assign each hidden unit a number between 1 to $n-1$ which indicates the number of inputs it will be connected to
- For example, if we assign a node the number 2 then it will be connected to the first two inputs
- We will do a similar assignment for all the hidden layers

Refer Slide Time (5:56)

The diagram is identical to the previous one, but the node in the second hidden layer with the number 2 is highlighted in blue, indicating its specific connectivity to the first two input nodes.

- Let us see what this means
- For the first hidden layer this numbering is clear - it simply indicates the number of ordered inputs to which this node will be connected
- Let us now focus on the highlighted node in the second layer which has the number 2

So, let's look at it. So, let's for the input layer the interpretation is very clear, that if I have a numbered node number 2, then it's only going to look at inputs, which are less than equal to 2. Okay? that means it's only going to be connected to 1 and 2, and if I have a input if I have a node number 3 then it's going to be connected to 1 2 and 3 is that fine! Okay? This is straight forward, because it is directly connected to the input and I know what the input numbers are. Now, if I look at the highlighted node in the second layer or any subsequent deep layer. if I have the node number 2 again using the same semantics, that it should only see the inputs which are less than or equal to 2. How do I ensure that? It

should be connected to nodes in the previous layer, which which are less than equal to 2. Right? So, it should only be connected to those nodes, which I have seen less than or equal 2 na inputs, which means those nodes which have a number less than or equal to two. Does that make sense? Again, gets this Please raise your hands. If you get this.

Refer Slide Time (7:03)

- Let us see what this means
- For the first hidden layer this numbering is clear - it simply indicates the number of ordered inputs to which this node will be connected
- Let us now focus on the highlighted node in the second layer which has the number 2
- This node is only allowed to depend on inputs x_1 and x_2 (since it is numbered 2)
- This means that it should be only connected to those nodes in the previous hidden layer which have seen only x_1 and x_2
- In other words it should only have connections from those nodes, which have been assigned a number ≤ 2

So, Now, this guy will only be connected to all those nodes, which are numbered one to two. So, I have got rid of many connections and only kept those connections which are relevant.

Refer Slide Time (7:16)

- Now consider the node labeled 3 in the output layer
- This node is only allowed to see inputs x_1 and x_2 because it predicts $p(x_3|x_2, x_1)$ (and hence the given variables should only be x_1 and x_2)
- By the same argument that we made on the previous slide, this means that it should be only connected to those nodes in the previous hidden layer which have seen only x_1 and x_2

Refer Slide Time (7:49)

- Now consider the node labeled 3 in the output layer
- This node is only allowed to see inputs x_1 and x_2 because it predicts $p(x_3|x_2, x_1)$ (and hence the *given* variables should only be x_1 and x_2)
- By the same argument that we made on the previous slide, this means that it should be only connected to those nodes in the previous hidden layer which have seen only x_2
- We can implement this by taking the weight matrices W^1 , W^2 and V and applying an appropriate mask to them so that the disallowed connections are dropped

and now, I can take this all the way up to the output layer if I have a node number three in the output layer, which are the nodes that it'll be connected to in the previous layer, anything less than equal to two. I'm I think I'm making a I'm using less than equal to and less than interchangeably. But I think that's clear from the context Right? So, this is only connected to these nodes these nodes in a turn are only connected to certain nodes in the input, and those nodes are only connected to these two inputs Right? So, I have transitively made sure, that the third output node only sees the relevant input nodes. Okay? Fine?

Refer Slide Time (7:50)

- For example we can apply the following Mask at layer 2

$$\begin{bmatrix} W_{11}^2 & W_{12}^2 & W_{13}^2 & W_{14}^2 & W_{15}^2 \\ W_{21}^2 & W_{22}^2 & W_{23}^2 & W_{24}^2 & W_{25}^2 \\ W_{31}^2 & W_{32}^2 & W_{33}^2 & W_{34}^2 & W_{35}^2 \\ W_{41}^2 & W_{42}^2 & W_{43}^2 & W_{44}^2 & W_{45}^2 \\ W_{51}^2 & W_{52}^2 & W_{53}^2 & W_{54}^2 & W_{55}^2 \end{bmatrix} \odot \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Refer Slide Time (7:52)

- Now consider the node labeled 3 in the output layer
- This node is only allowed to see inputs x_1 and x_2 because it predicts $p(x_3|x_2, x_1)$ (and hence the *given* variables should only be x_1 and x_2)
- By the same argument that we made on the previous slide, this means that it should be only connected to those nodes in the previous hidden layer which have seen only x_1 and x_2
- We can implement this by taking the weight matrices W^1 , W^2 and V and applying an appropriate mask to them so that the disallowed connections are dropped

This is again very similar to what I am dropping out certain connections. Very similar to drop out. Okay? what's the loss function going to be lots of loss function going to be? Some of cross entropies. What's the learning algorithm? Back propagation only through the relevant paths, and this thing which I have shown here, it's just not for the sake of it. and I'm gonna explain what that mask is not. Right?

Refer Slide Time (8:15)

- For example we can apply the following Mask at layer 2

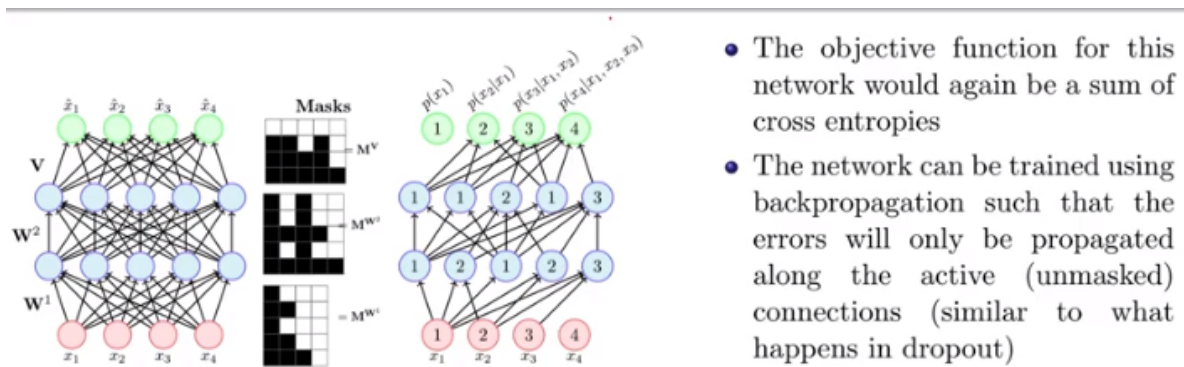
W_{11}^2	W_{12}^2	W_{13}^2	W_{14}^2	W_{15}^2
W_{21}^2	W_{22}^2	W_{23}^2	W_{24}^2	W_{25}^2
W_{31}^2	W_{32}^2	W_{33}^2	W_{34}^2	W_{35}^2
W_{41}^2	W_{42}^2	W_{43}^2	W_{44}^2	W_{45}^2
W_{51}^2	W_{52}^2	W_{53}^2	W_{54}^2	W_{55}^2

1	0	1	0	0
1	0	1	0	0
1	1	1	1	0
1	0	1	0	0
1	1	1	1	1

So, in this case, this was the old original fully connected W matrix. So, I'll always have the full W matrix. Only thing I'm going to do is, I'm going to use an appropriate mask to it to do the relevant computations, and the way I had numbered these nodes. It turns out that if I apply the following mask to the weight matrix. Right? Then it will ensure, that only the relevant nodes participate in the in a in the computation. Right? So, I think this is very straightforward. So, look at the first column it's all

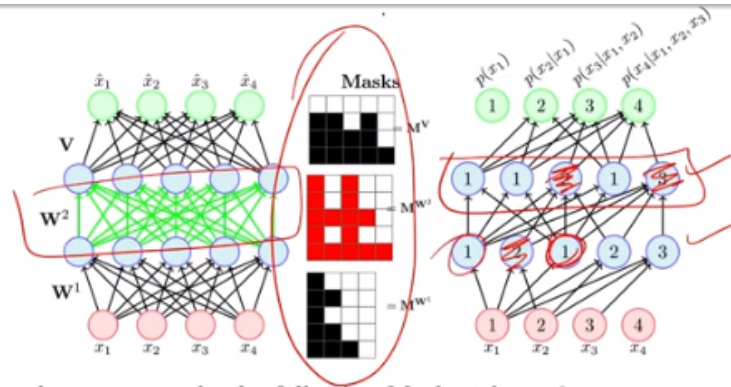
ones. I don't want to mask out any of these weights why? Because everything in this second layer is computed, cut is connected to the first node. Right? Now, let's look at similarly for this layer. Right? Everything in this layer is going to be connected the third node. So, that column has all ones I don't need to mask out anything from there. Now, let's look at this layer. So, only the third node here not only this these two nodes Right? Third and second are supposed to be connected to this node. So, I have two ones over there. Is that fine? So, that's exactly what these masks are showing. We'll apply this mask to the weight vector or weight matrix. It's going to be a harder met product, which reduces some entries to zero and retains the others, and this makes sure that in your computation at the next layer, only the relevant inputs are participating. Is everyone okay with this? Do I need to explain this masking again?

Refer Slide Time (9:54)



The objective function is again going to be a sum of the cross entropies. The network will be again trained using back propagation and the back propagation would only happen through relevant nodes yeah! So, you do this random number assignment once. Right? After that the masking is hard-coded. Manually do means you have to just define this mask matrix. Right?

Refer Slide Time (10:12)

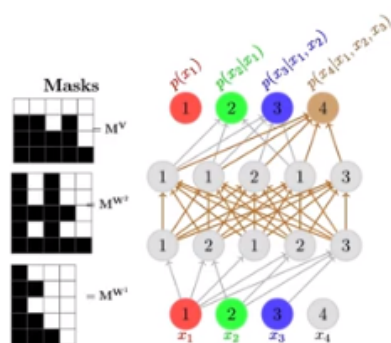


- For example we can apply the following Mask at layer 2

$$\begin{bmatrix}
 W_{11}^2 & W_{12}^2 & W_{13}^2 & W_{14}^2 & W_{15}^2 \\
 W_{21}^2 & W_{22}^2 & W_{23}^2 & W_{24}^2 & W_{25}^2 \\
 W_{31}^2 & W_{32}^2 & W_{33}^2 & W_{34}^2 & W_{35}^2 \\
 W_{41}^2 & W_{42}^2 & W_{43}^2 & W_{44}^2 & W_{45}^2 \\
 W_{51}^2 & W_{52}^2 & W_{53}^2 & W_{54}^2 & W_{55}^2
 \end{bmatrix}
 \odot
 \begin{bmatrix}
 1 & 0 & 1 & 0 & 0 \\
 1 & 0 & 1 & 0 & 0 \\
 1 & 1 & 1 & 1 & 0 \\
 1 & 0 & 1 & 0 & 0 \\
 1 & 1 & 1 & 1 & 1
 \end{bmatrix}$$

Yeah! But this is just an element-wise multiplication Right? This is not an expensive operation. No so, you don't need to randomly assign numbers to the nodes you can directly randomly created mass matrix. Instead of going from the numbers to the mass matrix as defined the mass matrix. Right? In fact, this explanation is done in a way. So, that you understand it but in practice you just have the mass matrix, that tells you what numbers the nodes have. Yes! So, you have all images of size 32 cross 32 or something. Right? If you don't have images of size 32 cross 32 just resize them to 32 cross 32. Right? So, that's always assumed all the inputs will of the same size. Okay? Because these are random variables. So, the number of random variables has to be fixed. Anything else?

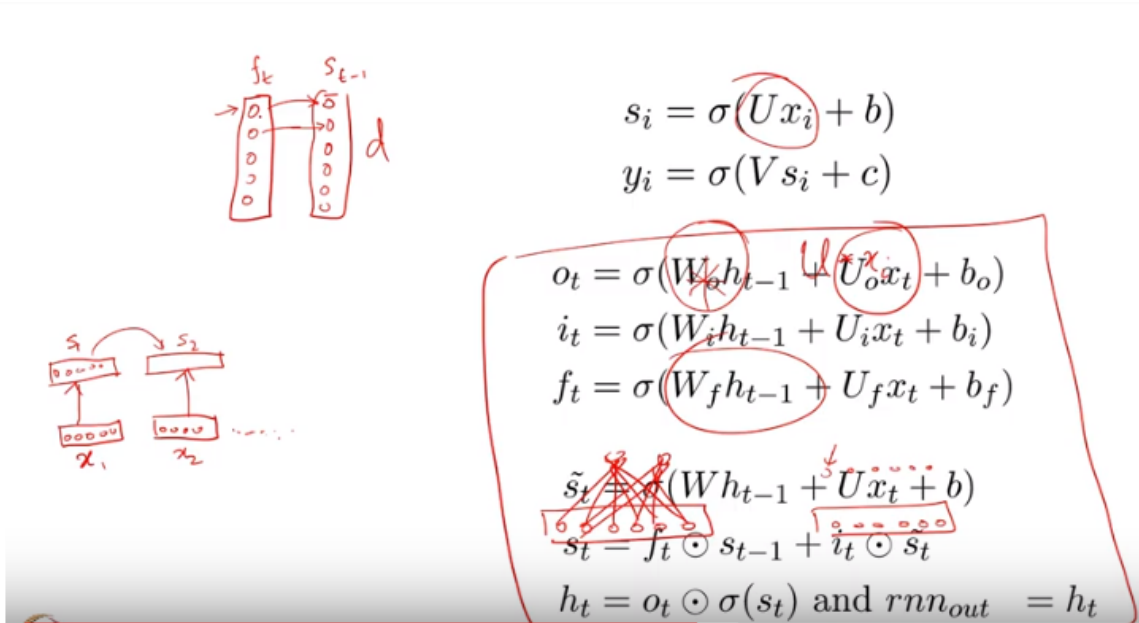
Refer Slide Time (10:58)



- Similar to NADE, this model is not designed for abstraction but for generation
- How will you do generation in this model? Using the same iterative process that we used with NADE
- First sample a value of x_1
- Now feed this value of x_1 to the network and compute y_2
- Now sample x_2 from $Bernoulli(y_2)$ and repeat the process till you generate all variables upto x_n

So, again this model is not really designed for abstraction. But can you do generation using this model. How will you do generation using this model? We just did need like five minutes back or ten minutes back. How will you do generation? same thing. What will you do first sample a value for x_1 ? Feed that in, compute a value for P of x_2 given x_1 and you can do that, because it only depends on X_1 only those parts will be activated and you'll get a P of x_2 given x_1 . Now, you'll sample a value of x_2 from this Bernoulli distribution feed that in. compute a value for X_3 you can do that because only those parts will be activated. Okay? Again, feed that. Again, one pixel at a time you can do the generation. Is that fine? Everyone okay with this okay? So, these are two fundamentals or almost foundational auto regressive models that, you need to know. The idea is that you need to pick from here are the following. One is you can work with the explicit factorization, where you have these n factors without any independence assumptions. You can then use a neural network, to parameterize this distribution. So, that you don't have an exponential number of parameters. The idea of masking is central to both these approaches, because masking ensures, that only the relevant inputs contribute to particular outputs. Because their outputs have a certain ordering, x_i given X_1 to X_{i-1} and this masking ensures, that at the end of the day both of these are just neural networks. So, you can use back propagation, and the last function is just going to be a sum of the Cross enterprise, and back propagation only needs to happen over the relevant paths. This is not something new. They've already seen this in drop outs. Even in drop words we ensure, that the backpropagation only happens through are levant part. So, it's the same. Now, these are not state-of-the-art in the sense, that today when you hear about autoregressive models, what people talk about is pixel RNNs and pixel CNNs. so, that's what I want to do next. By next I mean today itself I'll just continue. So, I think it's already 12 o'clock. So, people who have to leave can leave. The others can stay back. I need roughly half an hour more to finish off pixel RNNs. It's again an auto regressive model. I'll make sure, that this recording is up today itself. So, that you can immediately go and see it. So, people who want to leave can leave I'll just finish off pixel RNNs also. Now, if you have lectures you can just. I just take a break for five minutes and then finish. Right?

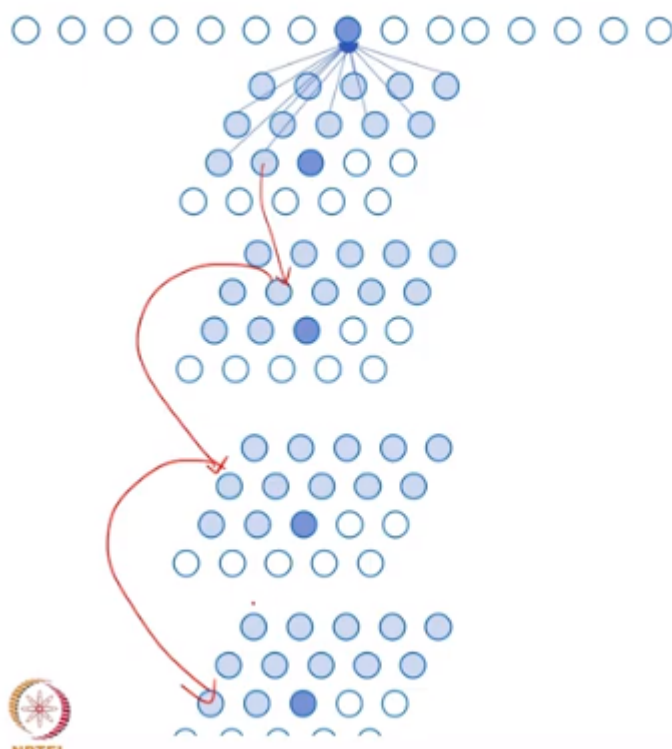
Refer Slide Time (13:37)



It isn't. But what you did is you started with an autoencoder which has n inputs and n outputs, and tweak that into a density estimator I'm going to say density estimator. But, Okay? Right? So, that's why I mean it's just I just wordplay in my opinion, but yeah, it's not really an autoencoder. I mean the motivation came that, okay? you have this an autoencoder which has n inputs and outputs and here again you have a setup where you need n inputs and n outputs. So, why not just take that and tweak it a bit Okay? So, what are you going to do now, is something one is pixel RNNs? Which is actually the basis of a lot of cool generative models, including wavelets. how many of you have heard of wavelets? There's a Google music generation stuff. Right? It can go and look it up. It generates pretty nice. So, the basis for that is pixel RNNs and that's what you're going to look at it. it's again an autoregressive model. the moment I say Autoregressive model what does it mean the factorization, that you are going to use is going to be the default chain rule. They are not going to be any assumptions on that, and we the entire quest in proposing or building an auto regressive model, is to come up with ideas such that, X_i only depends on X_1 to X_i minus 1. Right? The entire engineering goes into that and that's what we saw in need, as well as made. We define the inst equations, or the masks in a way, such that X_i only depends on X_1 to X_i mention that's the main part. Otherwise the setup is clear you're given n inputs, you are given n outputs the outputs are actually going to predict quality distributions. Right? and even the sampling part remains the same you are going to sample one pixel at a time. So, all that remains same it's the only way they differ in it is how do you go from this input to the output? while ensuring your criteria, that you should only look at the relevant. Right? That's the only thing which changed. Okay? So, Now, with that in mind we are going to start, since this is pixel RNNs. We are going to start with RNNs. Okay? And then kind of build towards what pixel are RNNs and start with certain characteristics of RNNs. So, if you have a regular RNNs. So, let me just see how to draw that. So, you have this input X . Okay? This rather X_i Right? So, you have X_1 X_2 and so on X_1 X_2 and so on. Right? Now, what you have is first in s_1 and then s_2 Right? and

what kind of connection do you have from X_1 to s_1 . sparse or fully connected. So, s_1 has some D nodes. This every node in s_1 depend on every input in X_1 . It's a fully connected layer Right? That's what this is Right? and what about the connection from s_1 to s_2 . it's fully connected every node in s_1 contributes to every node in s_2 Right? and that's obvious, because you have this matrix multiplications Right? Now, a cryptic hint but if I want the input or if I want the output Right? S_i to depend only on certain inputs. Not all of them. That means is a fully connected network. I want a sparse connectivity. What do we know which helps us in sparse connectivity? Asking okay? if I had not done today's lecture. sparse connectivity. What kind of an operation allows us pass connectivity? I just draw something in the meantime while you're answering. What operation is this is my drawing so bad? What kind of operation allows us this pass connectivity? Convolutional neural networks how sparse connections Right? So, how do you write a convolution operation? Instead of product how do you write it? Wait what does this mean and use some kind of a filter, it will go over only some portions of X_i and compute a new value. Right? So, it would mean that I have this X_i Okay? Now, Right? Now, what I have is a U which is like a fully connected weight matrix. So, it all my outputs depend on my inputs. Instead of this if I use a convolution operation, what would happen? I'll have my inputs, I'll apply a convolution filter, every filter every pass of the filter or every stride of the filter is going to give me forgotten everything in the board conditioner. It is going to give me one pixel in the output Right? So, I'm going to get a similar-sized output. Except that every guy here only depends on certain inputs. Right? Why am I ranting about this certain inputs depending on certain inputs, because what are we studying? auto regressive models. What do you want there? That the outputs should depend only on certain inputs? All I'm telling you is, that convolution operation gives you one way, of making sure, that your output depends only on certain inputs. Okay? Is that fine is that Okay? Fine! So, that's what a standard neural network does and standard recurrent neural network does, and of course a LSTM is much more complicated, than that, but if you look at all you remember all these five four equations, that you have for a list they're all similar. Right? you have a W you have a U and a V have h_{t-1} X_t and b . Right? What matters is all these are fully connected operations, and I'm not Okay? with fully connected operations, because that means, every output depends on every input. Okay? So, that's I'm not saying this is a problem with RNNs. I'm just saying, that this is what standard are a means and LSTM do. Every output depends on every input every gate depends one very input. Okay? Is it fine Okay? So, remember now, let's look at this gate again. So, say this is your f_t and your f_t is going to multiply by s_{t-1} . Okay? so, this is an element-wise product. This is an element-wise product, but what I'm trying to tell you is, that every element here depends on all the previous guys. Right? because it's a fully connected layer. Do you get that? So, you can think of these as, if the dimension is D you have these D LSTM cells. Okay? But all of these cells are fully connected to each other's output. the gate for the D^{th} cell depends one everything, that happened in the previous time step. Okay? So, all of these are fully connected. Yes, is that Okay? Have you in fine with this. Okay?

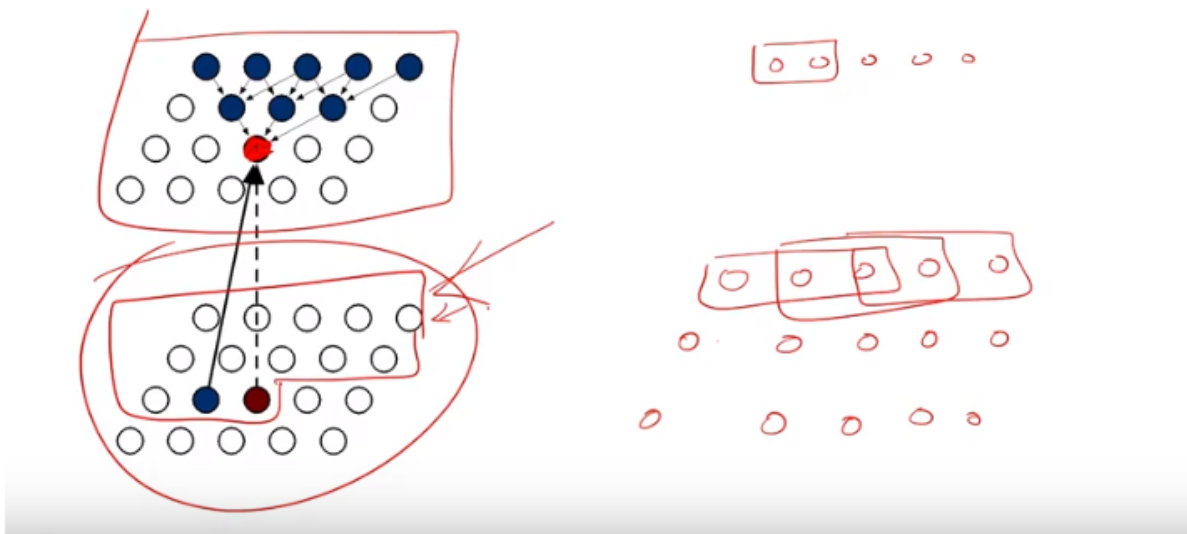
Refer Slide Time (21:20)



So, this is just re iterating about LSTM Okay? Now, coming back to our goal our goal is, that given an input and now, I'm going to talk only in terms of images. So, given an input it which is an image which would be n cross M or I'm going to assume n cross n for this discussion. Right? So, this is an N cross n input given to it. Our job is to compute n square probability distributions. So, you have this P of X_1 P of X_2 given X_1 . All the way up to P of X_n squared given X_1 to X_{n-1} is that fine that's what our job is in a case of an autoregressive model. What's the other thing, that we need to ensure? I have highlighted a certain node. What do I need to ensure? It only depends on previous inputs and here we are going to define previous as the following. Right? You see the bottom figure. So, here there is 1 to n square. Here again there is 1 to n square. There is a one-to-one mapping between them. so, when I am going to compute the I^{th} unit. It should only depend on 1 to $I-1$ that. Okay? That's what I want to ensure fine! and what pixel are ends do is they use something known as spatial LSTM. Okay? We will see, what a spatial LSTM is you will have multiple of these layers stacked upon one other. So, this is actually a spatial LSTM. I will define what it is, but you have 12 such layers stacked on top of each other, and this is your final output layer, from here you are going to predict these picks P of X_i given X_1 to $I-1$ is that okay? That means this guy should see only everything which appears before this and I should ensure that all of these in turn are only dependent on these blue guys. Is that fine so I have to define my LSTM operation in a way, that right from the input layer onwards I have this one-to-one correspondence between these layers. So, that I know, that when I am looking at the I^{th} guy here it only takes input from 1 to $I-1$ guys. Right?

So, in particular this Ayad guy will only depend on these 1 to I minus 1 guys and I have to make sure that in turn they also depend only off on the corresponding inputs Right? So, if I can ensure this for every pair of layers all the way up to the input, then I have ensured that the output has only seen the relevant inputs is the setup clear dude? How many of you are fine with this Please raise your hands? Okay? Now, let's look at one such pair of layers okay.

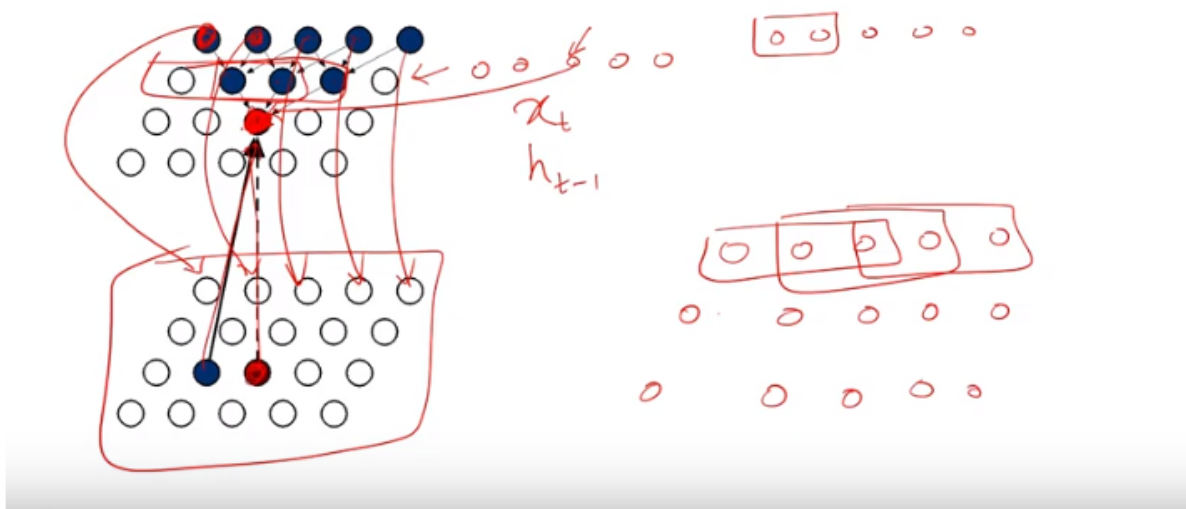
Refer Slide Time (24:04)



So, I'm looking at so, I'm trying to I have computed this layer already and Now, I'm trying to compute this layer. Okay? I want to make sure, that this red node in this layer only depends on their relevant inputs from the previous layer. Okay? And all of these nodes that you see they are actually LSTM cells. They are single dimensional LSTM cells. Okay? That means all of them have their forget gates, all of them have their input gates, all of them have their output gates, all of them take an input, and all of them take a give an output. Okay? and all of them also depend on the previous hidden state. Now, what I need to tell you is, how do I define the input? and how do I define the previous hidden state? So, that everything is closer and what do I mean by everything is closer? that every guy here sees only the relevant inputs. Is that Okay? Fine! So, Now, let's look at this LSTM cell. So, what I'm going to ensure is, that in a normal LSTM cell. All these inputs all these n square pixels that I have shown here, would have contributed to this one cell. Right? that's exactly the case which I was trying to make on the previous slide, that each of this is fully connected. So, even if I look at the red guy in the top layer. It would have been fully connected to everything from the previous layer. Do we want that no we just want to work within this boundary? Okay? let me just we just want to work within this boundary Okay? So, the way you define this spatial LSTM is, that I will first apply a convolution operation for every row on the previous layer. Okay? Is that fine? So, let's see this. So, I have this previous layer. I'm going to apply a row-wise convolution operation, or a one-dimensional convolution operation.

Okay? that means I'm going to take a filter and slide it across the row. What is the output that I'm going to get? Another row Right? and every pixel in that row would depend only on the previous pixels. Is that fine? Ever one okay with that? So, if I take this input, which I had and I apply a convolution on, that then I'm making sure, that in the output of the convolution every guy only depends on the previous inputs. Is that fine? Everyone ok with that Okay? So, now, from this you can imagine, I have taken one more from here I've computed an intermediate layer, from this layer I have computed an intermediate layer, where every pixel now, depends on the previous convoluted pixel. Okay? It only depends on the previous guys, it does not depend on any kind fine Okay. So,

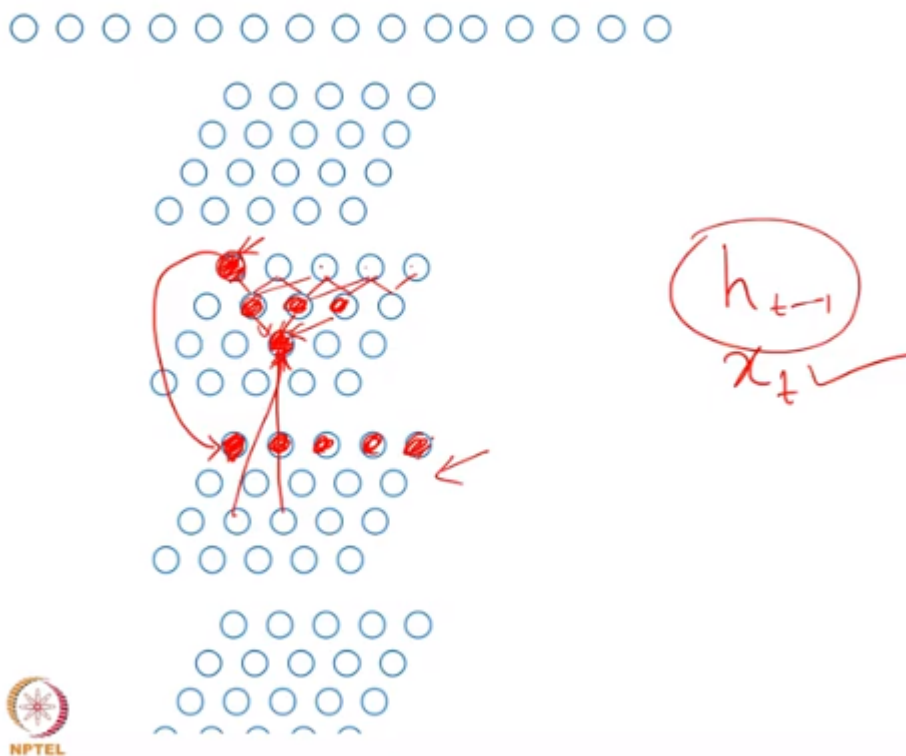
Refer Slide Time (27:16)



Now, let's imagine this is the output of that convolution here. Where every guy only depends on the previous input. Now, I want to compute the next layer from there, and I want to use an LSTM. Okay? If I want to use an LSTM, I need to define two things I need to define the input, and I need to define the connections from the previous state. These are the two things, that I need to define so Now, suppose I am at this point the input to the LSTM is only going to be this pixel and the previous pixel. Okay? Is that fine? What's the problem with this? What did I want the pixels to depend on all the previous inputs? What are done? Now, only one previous one. So, I have to give a justification for that. Okay? But I will come back to, that for now. Is it ok okay? Now, the other thing which I'm going to do is that for this guy you can think of all of these as h_{t-1} . These are the things which have already been computed so far Right? So, Now, one and the standard LSTM what I would have done is this guy would have been connected to all of them. Right? So, I'm not going to do that I am again going to define a convolution operation which ensure that this guy only depends on this row. It only depends on the three pixels above it. You can imagine a convolution operation which does, that it just takes the previous row, applies a three cross one mask on it three cross one convolution on that.

Okay? So, you'll get a new row. Every guy here depends on, three guys around here. Right? and now, this guy can feed to this input. Is that okay? it's okay, if you don't understand the details of the implementation, but can you imagine, that I am only making it dependent on the previous three pixels. Now, these three pixels in turn depended on three pixels from the previous layer. Is that fine okay? and what was the input to this guy? Is that fine? What was the input to this guy? This and so on. Right? So, in turn because of this transitive thing this pixel has now, become dependent on all the previous pixels, that it had. Is that clear? Do you get this? Previous answer if you do.

Refer Slide Time (29:55)



So, what I was saying is, that this layer has been computed. Okay? Fine! and now, I'm interested in computing this value I need to ensure that it depends only on the previous guys that's what my goal is Okay? it's a goal cleared everyone Okay? Now, each of these things each of these pixels here, is actually an LSTM, that means it has an input gate, it has an output gate, it has a forget gate, it takes input from the previous layer, that's the X , and it also has recurrent connections from h_{t-1} . Okay? Now, what I am trying to define is, X and H . so, forget everything that we did so far, I'll just start fresh I need to define X and H so what I'm saying is that the X is just going to be this pixel and the previous pixel. Okay? Is that straight forward. I'm just going to make it instead of having a fully connected representation. So, in the fully connected case all of these pixels would have contributed to this one pixel, but I don't want that, because I want to make sure that every output only depends on the

relevant inputs. Okay? So, to ensure that what I wanted is, that it should have dependent on all of these. Okay? As, of now, what I'm telling you is, that I will only make it dependent on this pixel. and the previous pixel, that's what I'm telling you. As of now, Okay? So, that's I have defined what X_i looks like. now, I need to define what H_t minus 1 looks like. Okay? So, my definition for H_t minus one is that, it will only depend on the three pixels above it. Is that fine? Again, it could have been a fully connected connection. But I'm just going to make it depend on eon the three pixels before it. Is that Okay? But this pixel in turn was dependent on three pixels above it, and this was dependent on these three pixels and this was dependent on these three pixels. Okay? Now, in turn this pixel would have taken input from, here Right? is that Okay? So transitively now, this since this pixel the one which I'm shading right now? Contributes to the central pixel, transitively this pixel also contributes to the central pixel. Is that fine? The same thing I can argue about all these pixels, which I am shading now, all of them transitively can't contribute to the pixel of interest. Is that Okay? Again, gets this. Okay? So, Now, in some way I've made sure that X_i depends on everything from X_1 to X_i minus 1. Is that fine okay? So, that's exactly how these LSTM operations are defined.

Refer Slide Time (32:58)

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

$$h_t = \tilde{s}_t = \sigma(W h_{t-1} + U x_t + b)$$

$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

$$h_t = o_t \odot \sigma(s_t) \text{ and } \text{output} = h_t$$

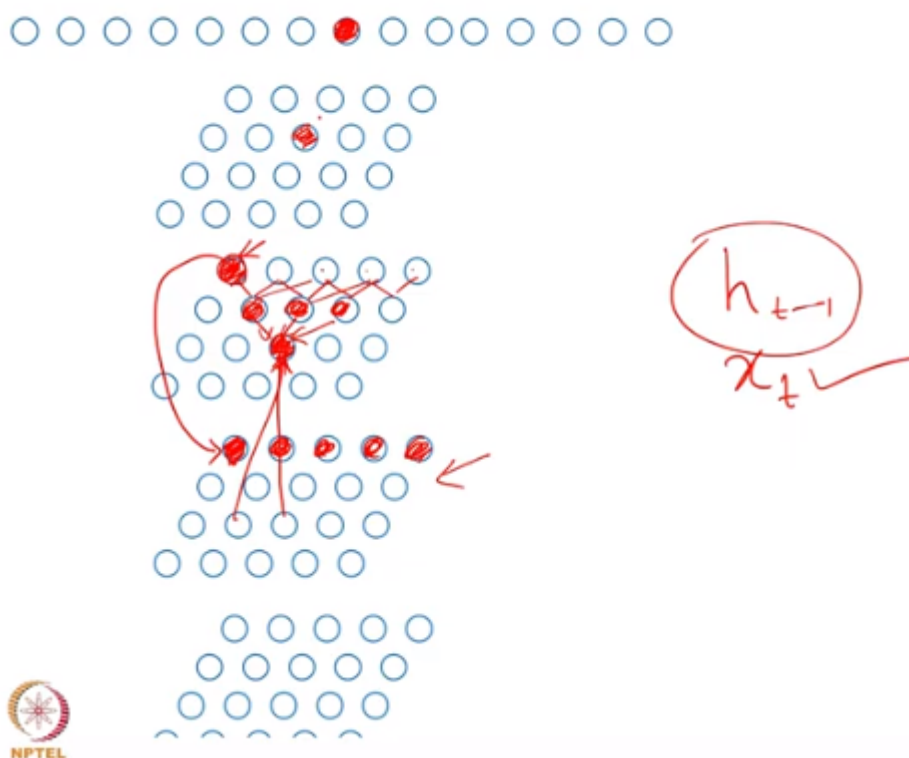
$$[o_i, f_i, i_i, g_i] = \sigma(K^{ss} * h_{i-1} + K^{is} * x_i)$$

$$c_i = f_i \odot c_{i-1} + i_i \odot g_i$$

$$h_i = o_i \odot \tanh(c_i)$$

So, remember in the original LSTM equations, you had this matrix multiplication. Now, we have replaced all of them by convolution operations, and the convolution operation ensures, that there is this sparse connectivity where only the relevant pixels, contribute to the next pixel. Is that Okay? So, we have taken the LSTM equations and just converted them to replace the fully connected operations to convolution operations. Right?

Refer Slide time (33;28)



So, that's exactly what a pixel RNN. Does it has these it has this deep layer of spatial LSTM's. all these LSTM's have a unique connection. They only depend on the previous guy, and transitively, then I ensure, that this particular pixel only depends on the relevant inputs, that it should depend on. Right? and now, instead of having a normal feed-forward neural network. I have these LSTM neural network. Because they also take inputs from the previous state. So, that there is are current connection between these. The only thing is that instead of having a fully connected recurrent connection you have a convolution driven fully connected conversion driven hidden connection or a recurrent connection. Right? so that's what pixel RNNs do at a very high level. So, now, you can go back and read the paper and try to understand it. So, the paper actually has many more details beyond this basic framework. What I wanted to explain to you is that starting from an input and given this goal, that your output has conditional probability distributions such, that each value in the output depends only on certain inputs. How do you ensure that? So, that's the intuition which I have tried to give you. Now, the paper has several more details they use. Something known as row LSTM something known as diagonal LSTM then some skipped connections and so on. So, now, with this intuition, if you try to understand the paper, you should be able to understand it better. So, you can go back and take a look at this. So, I'll just end here.