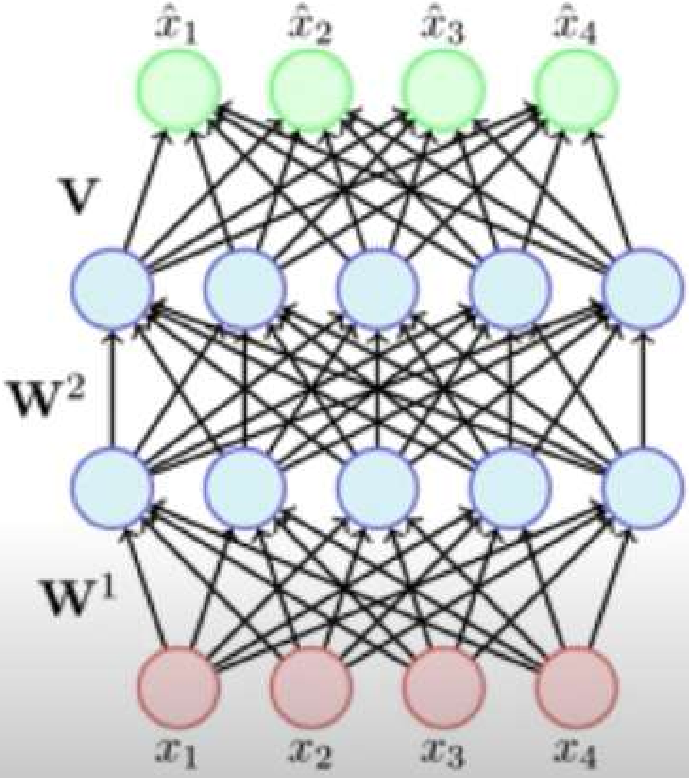
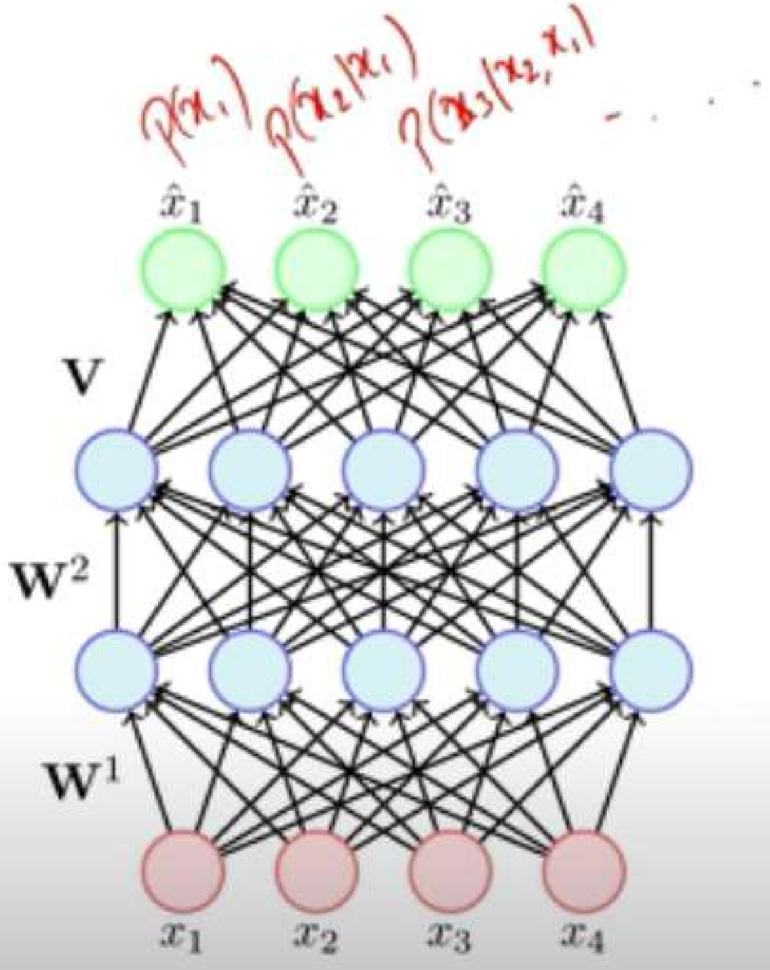
Module 21.2 : Masked Autoencoder Density Estimator (MADE)



- 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

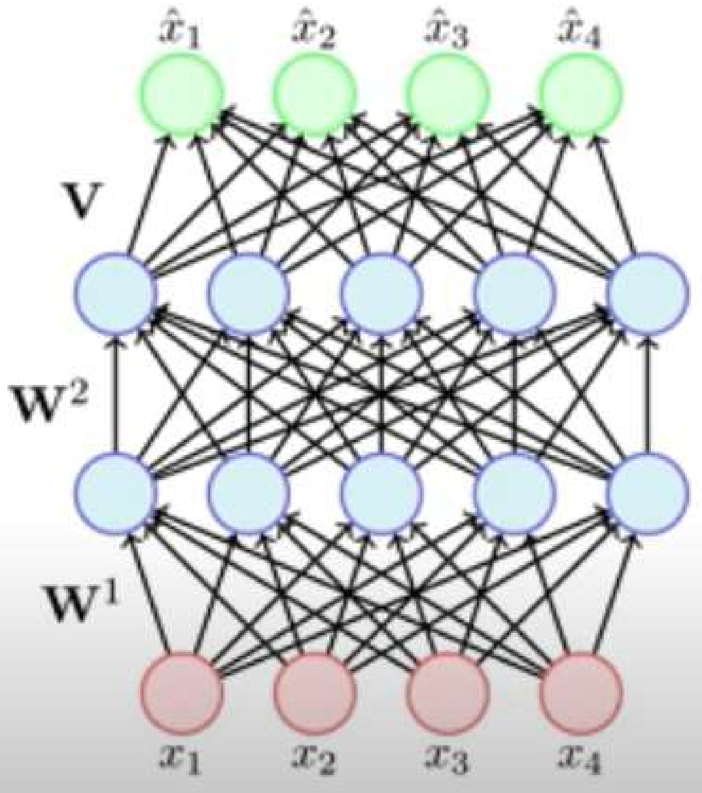
$$p(\mathbf{x}) = \prod_{k=1}^{n} p(x_k|\mathbf{x}_{<k})$$

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

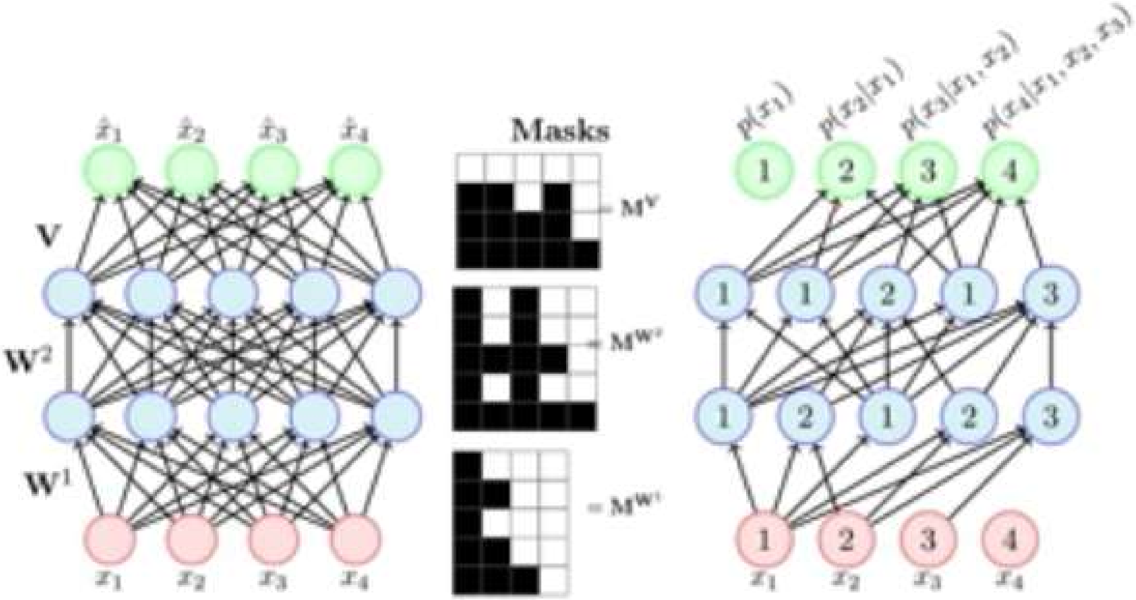


- Note that this is not straightforward because we need to make sure that the k^{th} output unit only depends on the previous k-1 inputs

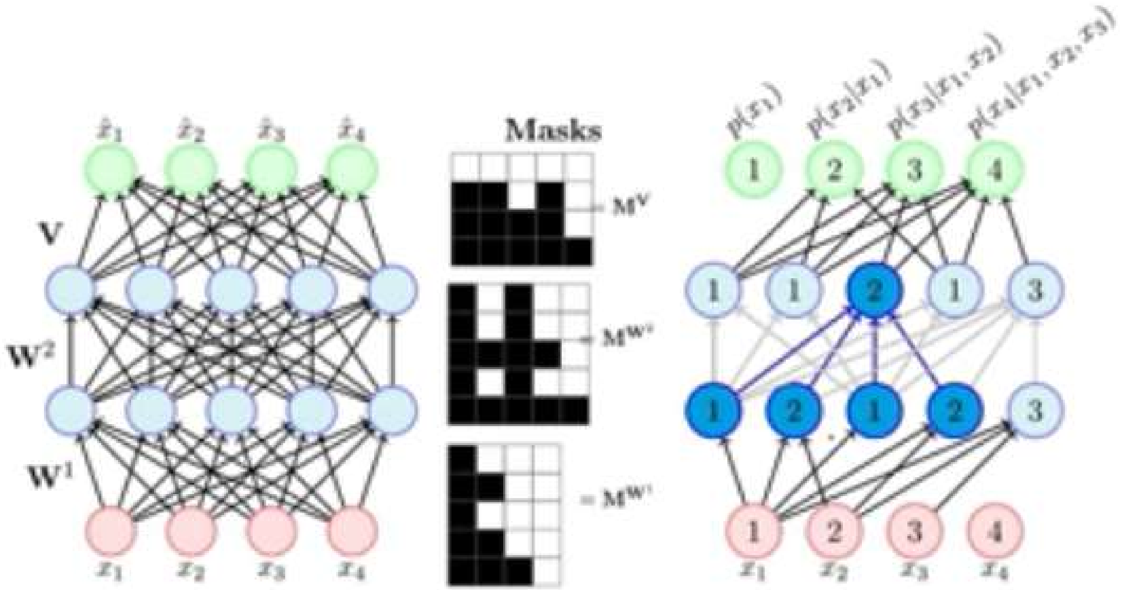
3:19 / 35:20



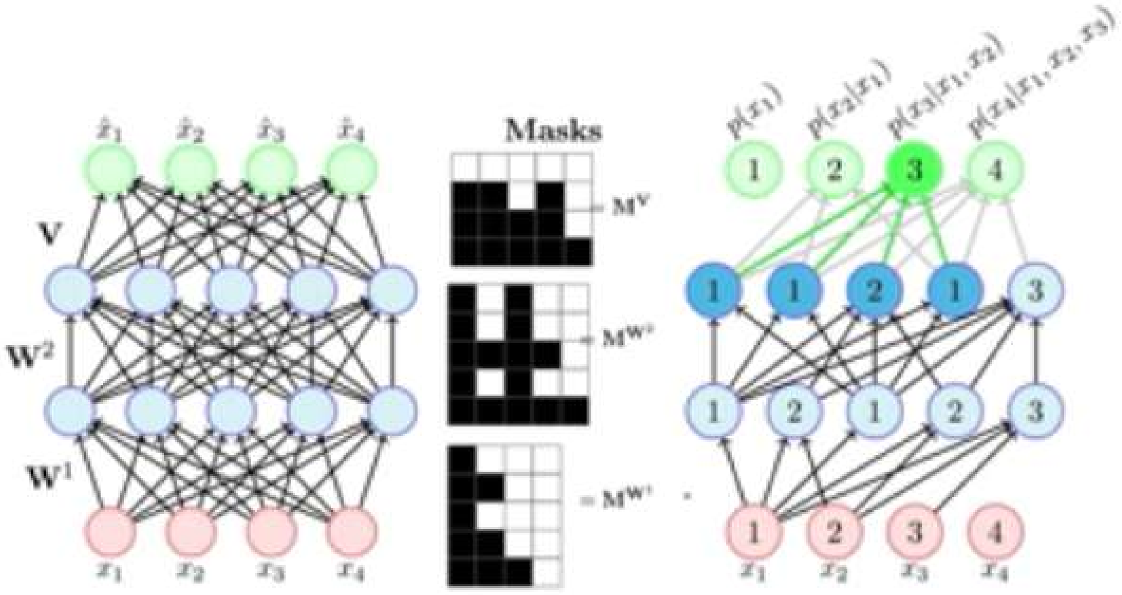
- Note that this is not straightforward because we need to make sure that the k^{th} output unit only depends on the previous k-1 inputs
- In a standard autoencoder with fully connected layers the k^{th} unit obviously depends on all the input units
- In simple words, there is a path from each of the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



- 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

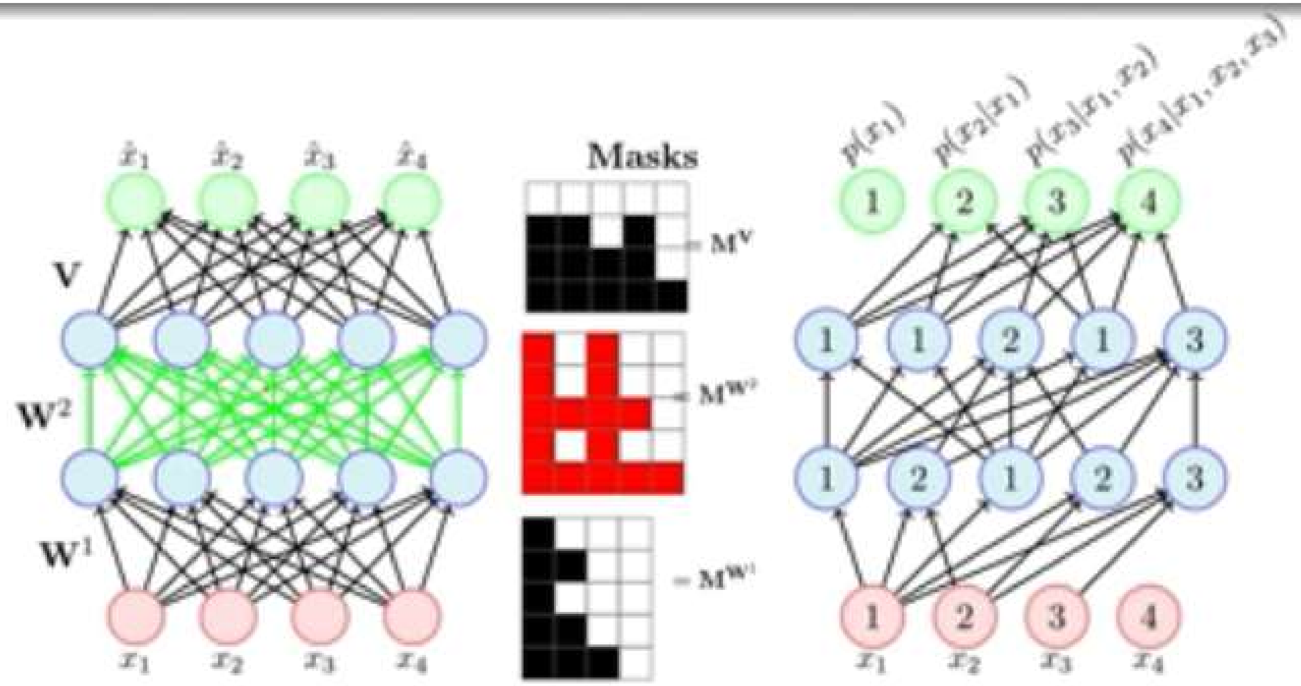


- Let us see what this means
- For the first hidden layer this numbering is clear - it simply indicates the number of ordered inputs to which this node will be connected
- Let us now focus on the highlighted node in the second layer which has the number 2
- This node is only allowed to depend on inputs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



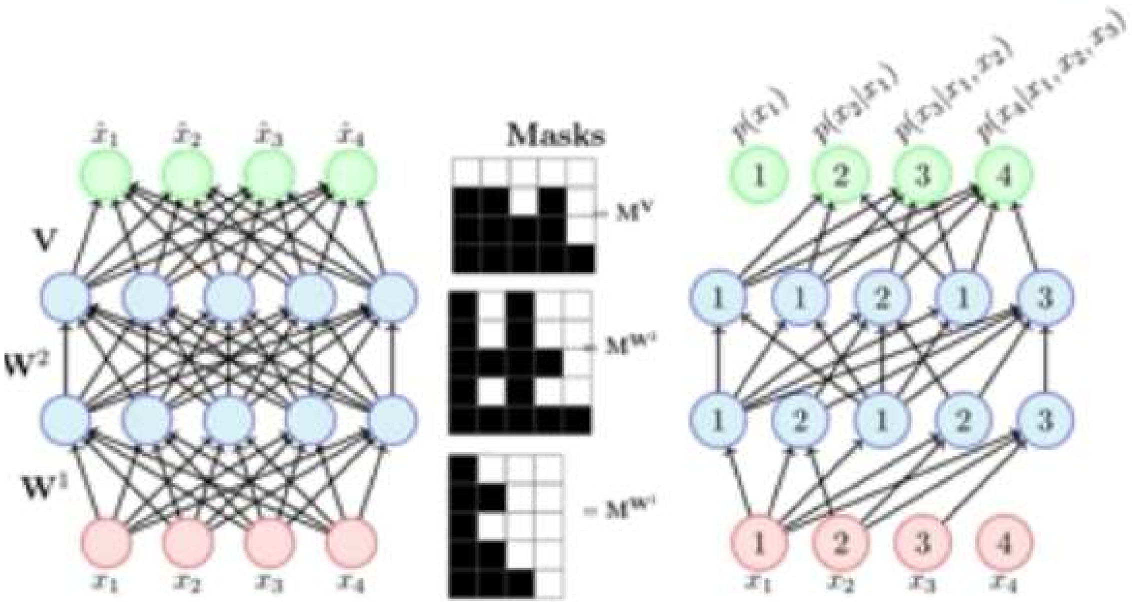
- Now consider the node labeled 3 in the output layer
- This node is only allowed to see inputs x_1 and x_2 because it predicts $p(x_3|x_2, x_1)$ (and hence the *given* variables should only be x_1 and x_2)
- By the same argument that we made on the previous slide, this means that it should be only connected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

8:08 / 35:20

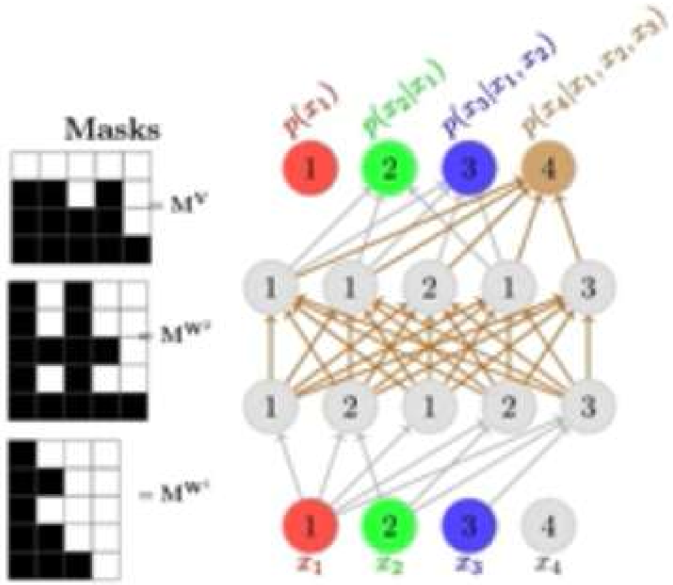


- For example we can apply the following Mask at layer 2

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



- The objective function for this network would again be a sum of cross entropies
- The network can be trained using backpropagation such that the errors will only be propagated along the active (unmasked) connections (similar to what happens in dropout)



- 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