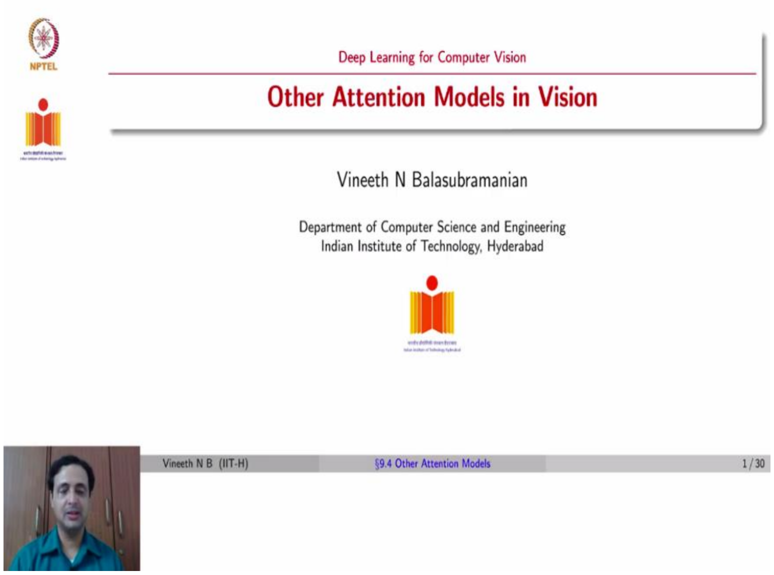
Deep Learning for Computer Vision
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 59
Other Attention Models in Vision

(Refer Slide Time: 00:14)



Deep Learning for Computer Vision

Other Attention Models in Vision

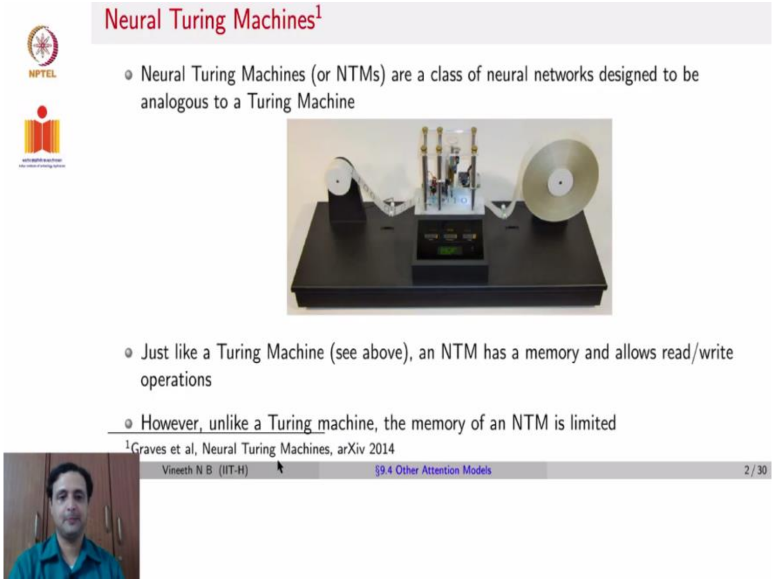
Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Vineeth N B (IIT-H) §9.4 Other Attention Models 1 / 30

Moving on from Visual Q and A and the visual dialog will now look at how attention has been modeled and used in various ways. In particular, we look at 3 specific models, Neural Turing machines, DRAW, which stands for Deep Recurrent Attentive Writer, and Spatial Transformers. Each of these has had a significant impact on the field of Deep Learning and Computer Vision. And they either use or implement attention in very different ways. Let us see each of them in this lecture.

(Refer Slide Time: 01:03)



Neural Turing Machines[1]

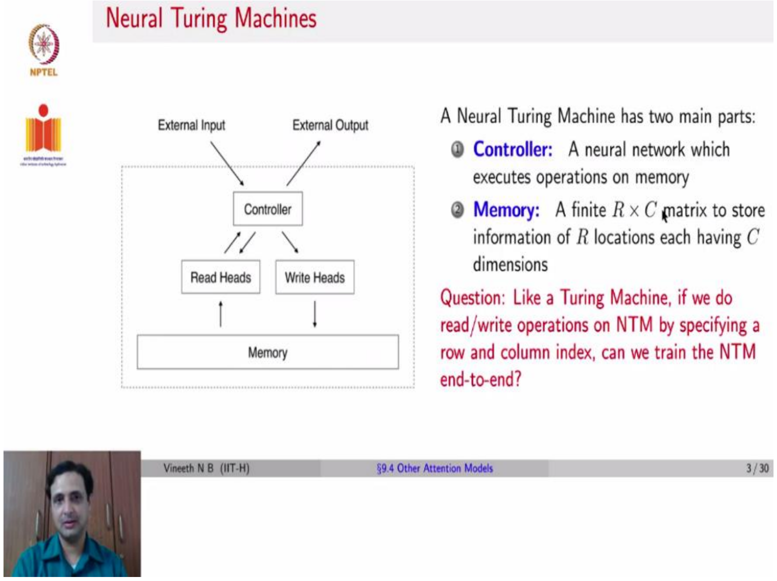
- Neural Turing Machines (or NTMs) are a class of neural networks designed to be analogous to a Turing Machine
- Just like a Turing Machine (see above), an NTM has a memory and allows read/write operations
- However, unlike a Turing machine, the memory of an NTM is limited

[1] Graves et al, Neural Turing Machines, arXiv 2014

Vineeth N B (IIT-H) §9.4 Other Attention Models 2 / 30

Let us start with Neural Turing machines. Neural Turing machines, as the name states denote a class of neural networks that are intended to perform tasks analogous to the Turing machine, just like a Turing machine, and NTM or a Neural Turing machine has memory and read-write operations to access to and from memory to perform subsequent tasks. However, unlike a Turing machine, you need to have fixed memory to be able to treat this as a neural network and learn operations.

(Refer Slide Time: 01:43)



Neural Turing Machines

A Neural Turing Machine has two main parts:

1. **Controller:** A neural network which executes operations on memory
2. **Memory:** A finite $R \times C$ matrix to store information of R locations each having C dimensions

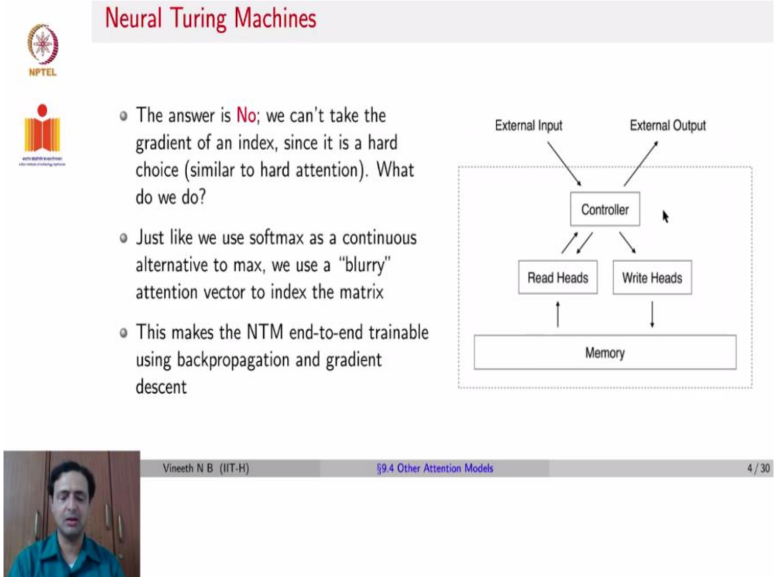
Question: Like a Turing Machine, if we do read/write operations on NTM by specifying a row and column index, can we train the NTM end-to-end?

Vineeth N B (IIT-H) §9.4 Other Attention Models 3 / 30

So here is a high-level visualization of a neural Turing machine where you have a controller, you have a memory, and the controller access accesses the memory using read and writes heads to read from and write into the memory. So the controller is a neural network, certain layers, which executes operations on memory, read and write.

And the memory by itself is an RxC matrix with R rows, each of C dimensions. Now let us ask a question, like a Turing machine. If we do read-write operations on an NTM by specifying a row and a column index, let us say we want to read the 3rd row and the 10th column. Can we train such an NTM end to end using backpropagation?

(Refer Slide Time: 02:48)



Neural Turing Machines

- The answer is No; we can't take the gradient of an index, since it is a hard choice (similar to hard attention). What do we do?
- Just like we use softmax as a continuous alternative to max, we use a "blurry" attention vector to index the matrix
- This makes the NTM end-to-end trainable using backpropagation and gradient descent

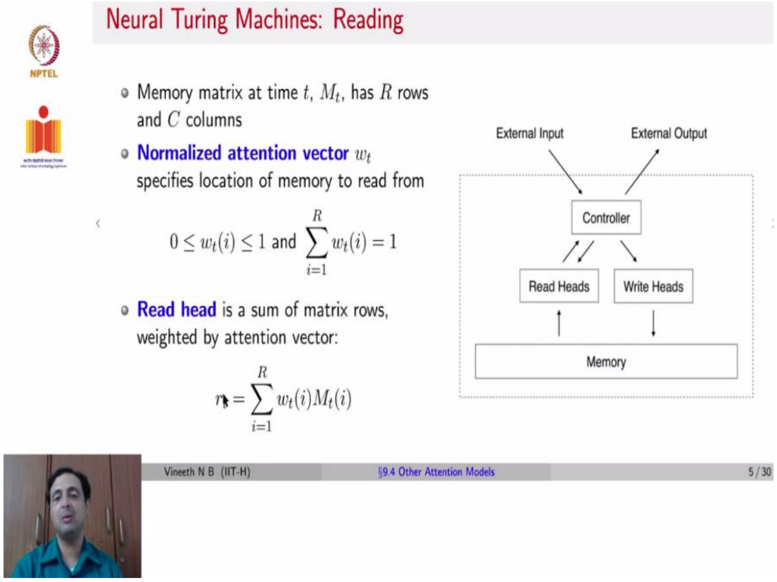
Vineeth N B (IIT-H) §9.4 Other Attention Models 4 / 30

The answer is, unfortunately, no because you cannot take the gradient of an index, since it could be a hard choice, similar to hard attention. Just like how instead of a matrix, you had a feature map, and let us assume that a C in a CNN, you wanted to focus only on one patch in a particular feature map and that patch could be decided differently for different inputs. And only that should be processed through the rest of the network that would be equivalent to hard attention, where you are focusing only on one part and that cannot be backpropagated through.

Instead, what can we do very similar to what we talked about as soft attention, you could now assume that you have a soft weighting function over different parts of a matrix. In an image, it was called soft attention. Here, we are looking at the memory matrix, which is RxC, and we have soft attention over all the locations in the memory, which can be used to read and write from the memory.

So this makes the neural Turing machine end-to-end trainable using backpropagation and gradient descent. So we use a very similar idea as Soft Attention.

(Refer Slide Time: 04:14)



Neural Turing Machines: Reading

- Memory matrix at time t, M_t, has R rows and C columns
- **Normalized attention vector** w_t specifies location of memory to read from

$$0 \leq w_t(i) \leq 1 \text{ and } \sum_{i=1}^{R} w_t(i) = 1$$

- **Read head** is a sum of matrix rows, weighted by attention vector:

$$r_t = \sum_{i=1}^{R} w_t(i) M_t(i)$$

External Input

External Output

Controller

Read Heads

Write Heads

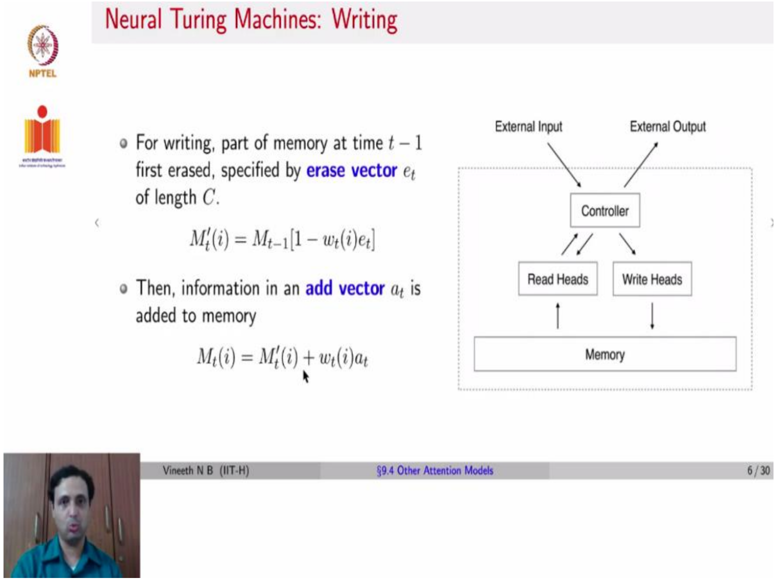
Memory

Vineeth N B (IIT-H) §9.4 Other Attention Models 5 / 30

Now let us look at these components in more detail. So you have a memory matrix at time t given as M_t, which has R rows and C columns as we already saw, the normalized attention vector, which is used to read from memory is given by a set of weights $W_t(i)$, and the weights add up to one and each of those weights lies between 0 and 1. And the read head is a sum of the matrix rows weighted by the attention vector.

So if each row of the memory was empty of i, i th row of the memory at time t. Then you have a corresponding weight for each row in the memory given by an attention mechanism. And a weighted sum of all your memory rows becomes what the read head gets from memory.

(Refer Slide Time: 05:19)



Neural Turing Machines: Writing

- For writing, part of memory at time $t-1$ first erased, specified by **erase vector** e_t of length C.

$$M_t'(i) = M_{t-1}[1 - w_t(i)e_t]$$

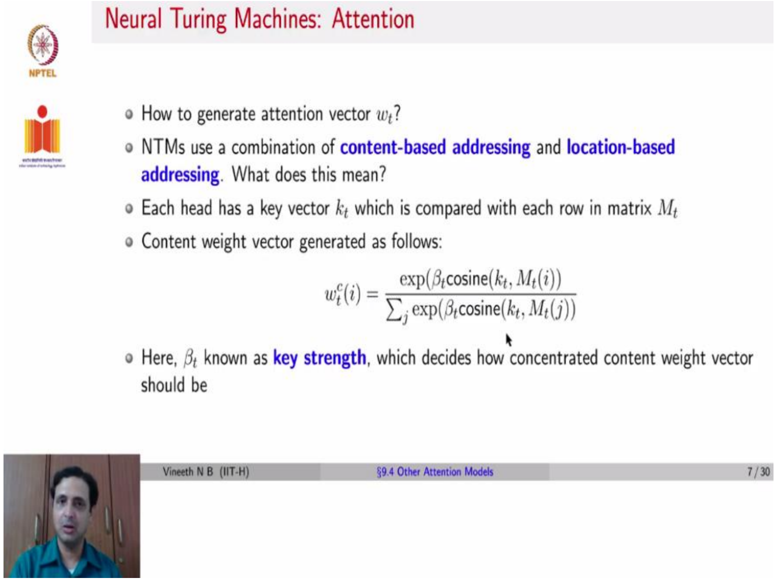
- Then, information in an **add vector** a_t is added to memory

$$M_t(i) = M_t'(i) + w_t(i)a_t$$

Vineeth N B (IIT-H) §9.4 Other Attention Models 6 / 30

Similarly, for writing a part of memory at time $t-1$ is erased using an erase vector. So, if you have an erase vector et 1 minus the erase vector is what is retained. So, if you had M_{t-1}, which is the memory of $(M_{t-1})(1 - W_t(i))$, which are the weights into arrays, vector et would give you the memory at the new time step. And now you add a weighted combination of any external input, let us call that input an added vector at, you have a weighted combination of that add vector, which is added to the memory and that particular i th row.

(Refer Slide Time: 06:09)



Neural Turing Machines: Attention

- How to generate attention vector w_t?
- NTMs use a combination of **content-based addressing** and **location-based addressing**. What does this mean?
- Each head has a key vector k_t which is compared with each row in matrix M_t
- Content weight vector generated as follows:

$$w_t^c(i) = \frac{\exp(\beta_t \text{cosine}(k_t, M_t(i))}{\sum_j \exp(\beta_t \text{cosine}(k_t, M_t(j))}$$

- Here, β_t known as **key strength**, which decides how concentrated content weight vector should be

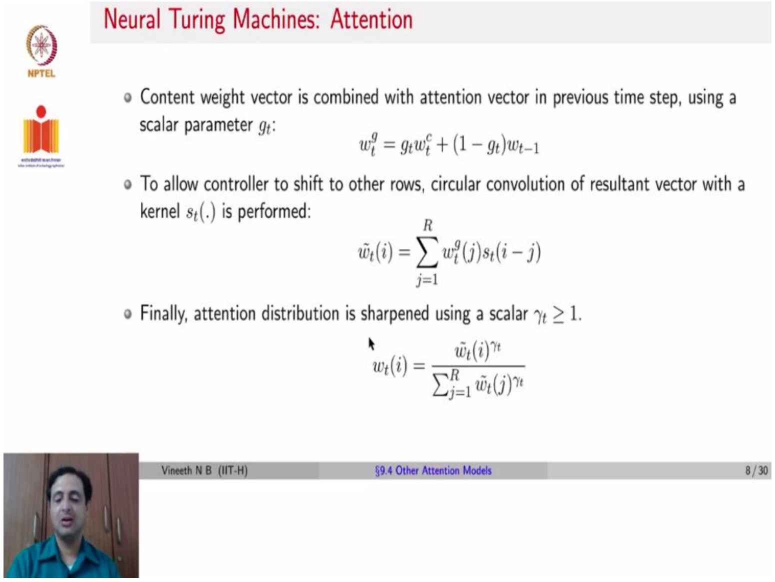
Vineeth N B (IIT-H) §9.4 Other Attention Models 7 / 30

How do you obtain this attention vector W_t? So, to do that, NTM uses both what is known as Content-based addressing and Location-based addressing similar to how you do it on a computer. What does this mean, in terms of learning a Neural network? Each head be it the right head, or the read head has a key vector k_t at time step t, which is compared with each row in the memory matrix M_t. How is it compared? So, you take the vector k_t, the key k_t, take the cosine distance between k_t and i th row of the matrix M_t. So that gives you how similar that key is with respect to one specific location in your memory.

So this is like content-based addressing where you are trying to compare the content that you have with the content in one particular row in the memory. And the final attention is obtained similar to a softmax over the alignment of the current key vector with every row divided with a particular row divided by the sum of the alignment with all rows.

You do have a constant β_t here, which is you could consider a weighting factor or also what the Neural Turing machines work calls key strength, which decides how concentrated the content vector weight vector should be, you could consider that a scaling factor at this time. This is the first step which is largely content-based addressing.

(Refer Slide Time: 07:57)



Neural Turing Machines: Attention

- Content weight vector is combined with attention vector in previous time step, using a scalar parameter g_t:

$$w_t^g = g_t w_t^c + (1 - g_t) w_{t-1}$$

- To allow controller to shift to other rows, circular convolution of resultant vector with a kernel $s_t(.)$ is performed:

$$\tilde{w}_t(i) = \sum_{j=1}^{R} w_t^g(j) s_t(i-j)$$

- Finally, attention distribution is sharpened using a scalar $\gamma_t \geq 1$.

$$w_t(i) = \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_{j=1}^{R} \tilde{w}_t(j)^{\gamma_t}}$$

Vineeth N B (IIT-H) §9.4 Other Attention Models 8 / 30

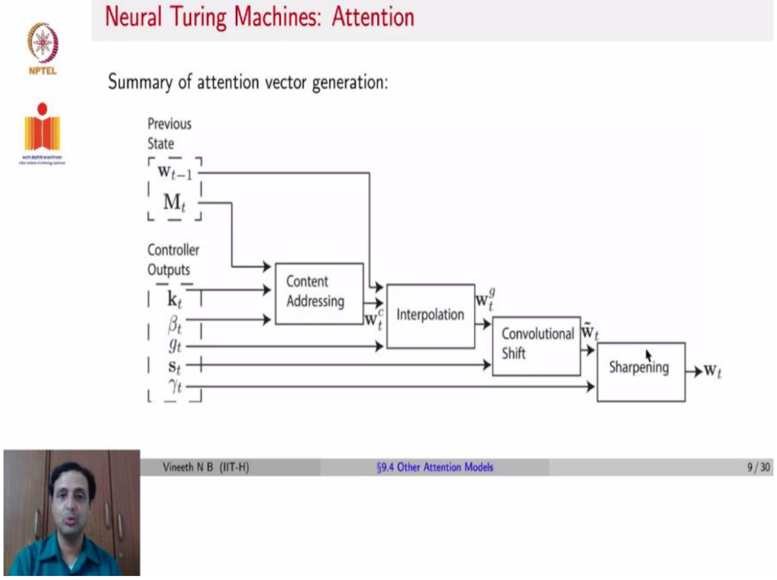
Now once you finish the content-based addressing, you combine this with the attention vector in the previous time step. So your final attention now is given by your current attention into the previous attention W_{t-1} the previous time step. So you interpolate those two using some scalar parameter gt. And that is what you get as the attention vector that you are going to use in this time step. Are we done?

No, we still have the location-based addressing to do to address this NTMs perform a circular convolution of the resultant vector with a kernel. So, if you have an attention vector Wt remember, which is a vector over all possible memory rows. Now, you perform a convolution operation with a kernel St this is done in case you have to swap say locations in memory based on this content.

So, this is a kernel that could be decided based on what the controller needs to achieve. So, if locations of different content have moved for some reason, those kinds of changes can be addressed by convolving this attention vector with the suitable kernel to get the final attention vector.

And this attention vector or distribution is then sharpened using a scalar by raising each attention value to a particular γ_t value to get a sharpened attention vector. So sharpened here would mean that the value increases for a high attention scalar value for particular, for a particular position and decreases for another attention value for a particular row where the original value may have been low.

(Refer Slide Time: 10:01)



Neural Turing Machines: Attention

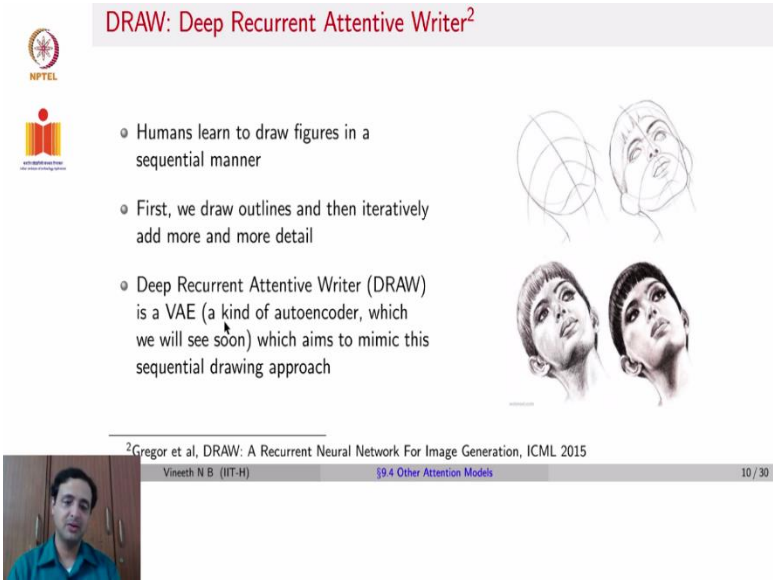
Summary of attention vector generation:

Vineeth N B (IIT-H) §9.4 Other Attention Models 9 / 30

One can summarize all these steps as you have the attention vector from the previous time step, W_{t-1}, you have the memory matrix at time t, M_t, and you have your controller outputs, k_t, that is your key β_t, that is your key strength g_t, that is a scalar to control the interpolation between the previous step potential and the step attention. Similarly, S_t and γ_t, which is to sharpen, so, you have you first perform content addressing, then you perform interpolation between your current attention weights, and your previous time steps attention weights.

Then you perform a convolution will shift with respect to a kernel, this address is location-based issues that the controller may want to achieve. And finally, a sharpening of the weights to decide which part of the memory to, to be attending to at a particular time step.

(Refer Slide Time: 11:04)



DRAW: Deep Recurrent Attentive Writer[2]

- Humans learn to draw figures in a sequential manner
- First, we draw outlines and then iteratively add more and more detail
- Deep Recurrent Attentive Writer (DRAW) is a VAE (a kind of autoencoder, which we will see soon) which aims to mimic this sequential drawing approach

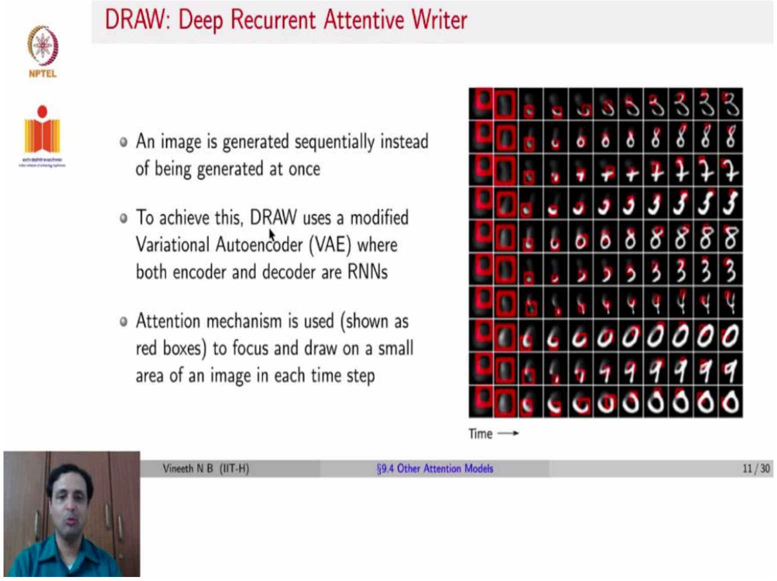
[2] Gregor et al, DRAW: A Recurrent Neural Network For Image Generation, ICML 2015

Vineeth N B (IIT-H) §9.4 Other Attention Models 10 / 30

So, that is the working of the neural Turing machine, as you saw a different use of what we saw as attention. Now we will move on to the second method that we said we will talk about in this lecture, which is known as DRAW or Deep, Recurrent Attentive Writer. This work was published in 2015. And this work is based on the intuition that when humans generate images, such as say drawings and sketches, Humans learn to draw sequentially.

One first draws the outlines, then keeps adding details iteratively. DRAW is used what is known as a variational autoencoder. It is a variant of an autoencoder that is used for generative models to generate images, which we will see very soon in the coming week's lectures. It uses a VAE to mimic this kind of approach to generate images. Why are we talking about this here, although we have not covered Generative models, yet, it uses a unique attention mechanism, which may be of interest in this lecture.

(Refer Slide Time: 12:27)



DRAW: Deep Recurrent Attentive Writer

- An image is generated sequentially instead of being generated at once
- To achieve this, DRAW uses a modified Variational Autoencoder (VAE) where both encoder and decoder are RNNs
- Attention mechanism is used (shown as red boxes) to focus and draw on a small area of an image in each time step

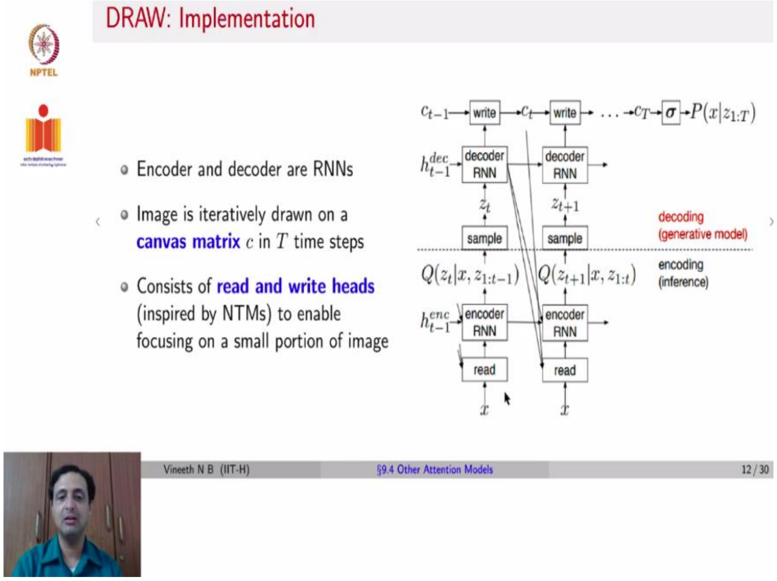
Vineeth N B (IIT-H) §9.4 Other Attention Models 11 / 30

So here is an example of how the DRAW method works. An image is generated sequence, instead of being generated at once, what you see here are 10 different generations, where each row is a generation of an image, done iteratively. So in the initial steps, if you look at the last row here, in the initial step, you have a blank canvas. The model then attends to a particular region, which is given by this red block here, then it draws something in that block, then the canvas gets updated.

The model focuses on another block, draw something there. Then in the next time step, the model focuses on the third area of an image, draws something there, and iteratively over time, the model ends up generating this particular image at the end. This is an MNIST image. This is a model trained on the MNIST image dataset. As we already mentioned, DRAW uses a variational autoencoder, which is a variant of an autoencoder, which we will see soon.

This autoencoder has encoders and decoders both of which are RNNs and an Attention mechanism is used to focus on specific parts to generate parts of an image iteratively. Let us see each of these components in more detail.

(Refer Slide Time: 14:05)



DRAW: Implementation

- Encoder and decoder are RNNs
- Image is iteratively drawn on a **canvas matrix** c in T time steps
- Consists of **read and write heads** (inspired by NTMs) to enable focusing on a small portion of image

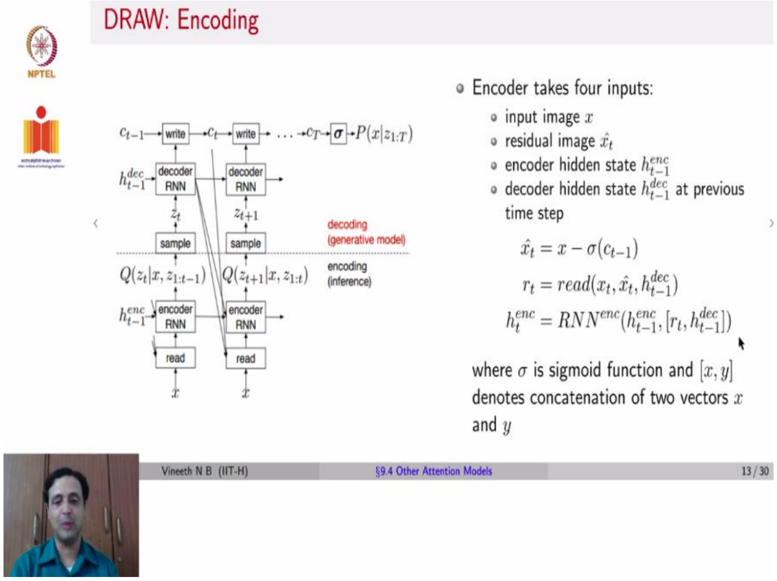
Vineeth N B (IIT-H) §9.4 Other Attention Models 12 / 30

So the encoder and decoder are RNNs, which means your overall architecture diagram is going to be like an autoencoder, which is one column of this diagram here. So you have an x input, you have a read module, which is an Attention module. Then you have an encoder RNN then you have an intermediate step, which you can ignore. Now we will talk about this in more detail when we go to Variational autoencoders.

Then from that intermediate layer, you go to the decoder RNN and then you finally have a write head that writes onto a canvas to generate some content. And both the encoder and the decoders are RNNs, which means they share weights and you have that particular block repeated over multiple time steps. So the image now is drawn on a canvas matrix C, which is what you see on top here in t time steps.

And DRAW has both read and write it is you could say it is inspired by NTMS, to focus on specific parts of the image for reading, as well as writing.

(Refer Slide Time: 15:22)



DRAW: Encoding

- Encoder takes four inputs:
  - input image x
  - residual image $\hat{x}_t$
  - encoder hidden state h_{t-1}^{enc}
  - decoder hidden state h_{t-1}^{dec} at previous time step

$$\hat{x}_t = x - \sigma(c_{t-1})$$
$$r_t = read(x_t, \hat{x}_t, h_{t-1}^{dec})$$
$$h_t^{enc} = RNN^{enc}(h_{t-1}^{enc}, [r_t, h_{t-1}^{dec}])$$

where σ is sigmoid function and $[x, y]$ denotes concatenation of two vectors x and y

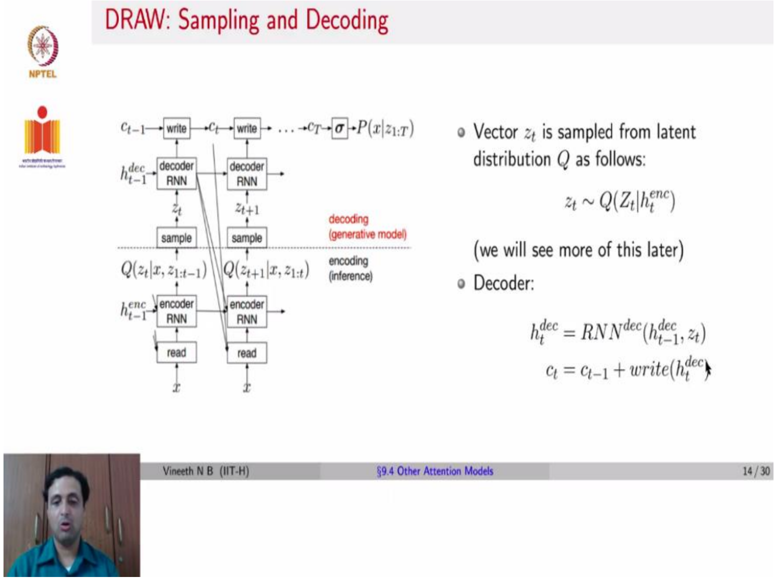
Vineeth N B (IIT-H) §9.4 Other Attention Models 13 / 30

So the encoder RNN takes four inputs, the input image X, a residual image x hat, we will see what x hat is, in a moment. The encoder hidden state in the previous time step, the decoder hidden state at the previous time step, you can see that the decoder hidden, hidden state at the previous time step is passed to the encoder in the next time step. So notationally speaking, $\hat{x} = x - \sigma(c_{t-1})$.

c_{t-1} is the canvas that has been drawn at time step $t - 1$. Sigmoid of that gives you values between 0 and 1, x minus that gives us $\hat{x}$, which is the portion of the image which the canvas so far has not focused on. So we are looking at the residual image, which is what we call $\hat{x}$. r_t is the output of the read module, which takes in x_t, $\hat{x}$ and h_{t-1} from the decoder, the hidden state at time step $t - 1$ from the decoder using these, it decides which part of the image to read at this time step.

And ht of the encoder is an RNN that receives the hidden state from the previous time step h_{t-1}, it receives the output of the read module r_t. And as we just mentioned, it also receives the hidden state in the previous time step from the decode. So here σ is a sigmoid function and when we write something in square brackets, we mean a concatenation of those values.

(Refer Slide Time: 17:21)



DRAW: Sampling and Decoding

- Vector z_t is sampled from latent distribution Q as follows:

$$z_t \sim Q(Z_t|h^{enc}_t)$$

(we will see more of this later)

- Decoder:

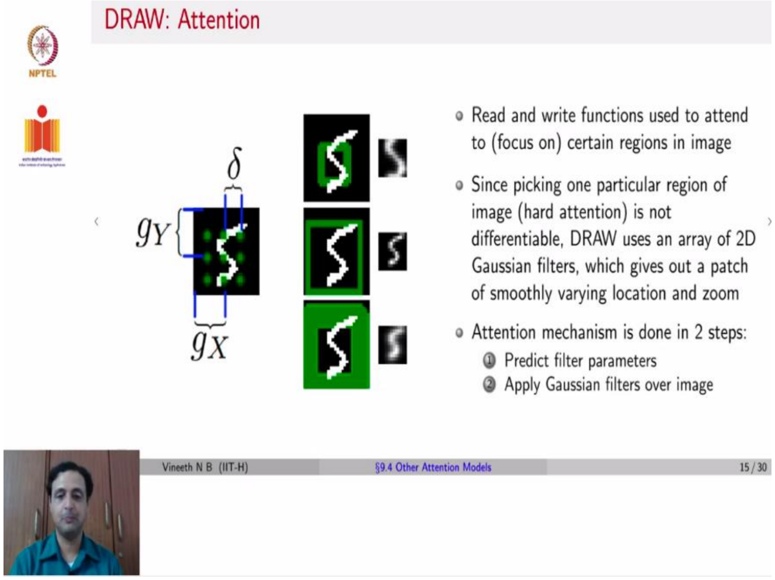
$$h^{dec}_t = RNN^{dec}(h^{dec}_{t-1}, z_t)$$
$$c_t = c_{t-1} + write(h^{dec}_t)$$

Vineeth N B (IIT-H) §9.4 Other Attention Models 14 / 30

So, when you for the decoder, when you use the decoder, you have a sampling step here, which we will talk about when we come to Variational autoencoders. So you sample a certain vector, which is passed to the deep decoder, RNN. And the decoder RNN uses the hidden state of the decoder at the previous time step. And the sample that came from for now, what we call the bottleneck layer of this variational autoencoder.

And the canvas finally, at time step t is updated as the canvas at the previous time step plus the output of the right module at that particular time step. This is the overall functioning of the DRAW framework.

(Refer Slide Time: 18:14)



DRAW: Attention

- Read and write functions used to attend to (focus on) certain regions in image
- Since picking one particular region of image (hard attention) is not differentiable, DRAW uses an array of 2D Gaussian filters, which gives out a patch of smoothly varying location and zoom
- Attention mechanism is done in 2 steps:
  1. Predict filter parameters
  2. Apply Gaussian filters over image

Vineeth N B (IIT-H) §9.4 Other Attention Models 15 / 30

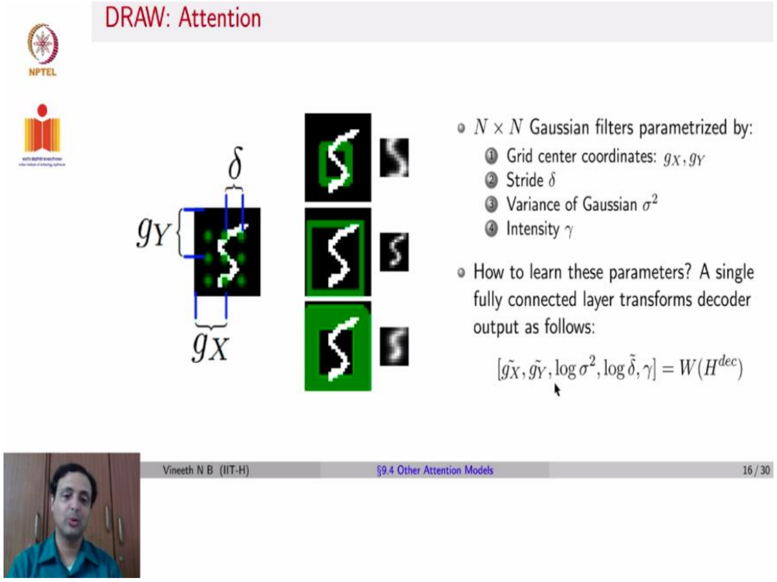
Now, let us try to understand how the Read and Write modules work here, which are attended, which are our Attention mechanisms in this particular framework. So the goal of the Read and Write modules are to focus on specific parts of an image. The way draws goes about this is because we cannot focus just on one region, then you would end up having a problem because you cannot differentiate through that one region which could change for every input.

So you have to somehow distribute the weights across the full image. So this is done in two steps in the drawing framework. One, it predicts certain filter parameters that we will talk about, and using those filter parameters, it places Gaussians over a grid of locations in the image, whose coordinates are also learned and that becomes your attention mechanism.

To, to explain this image, given a particular image, the attention mechanism has to focus on a certain region, this region is given by a grid of locations, and the center of that grid, which is given by gx and gy is learned as well as δ, which is the stride between different points in that grid is also learned. So by designing a 3x 3 grid, and learning a gx, gy, and δ, the grid is completely specified.

We still have some more details to add, but at this time, the grid is specified. You can see here that if you had the grid at a particular location in the image, you see that it focuses on a certain part. And as the grid locations are moved, the model focuses on a different part of the image.

(Refer Slide Time: 20:15)



DRAW: Attention

- $N \times N$ Gaussian filters parametrized by:
  1. Grid center coordinates: g_X, g_Y
  2. Stride δ
  3. Variance of Gaussian σ^2
  4. Intensity γ
- How to learn these parameters? A single fully connected layer transforms decoder output as follows:

$$[\tilde{g_X}, \tilde{g_Y}, \log \sigma^2, \log \tilde{\delta}, \gamma] = W(H^{dec})$$

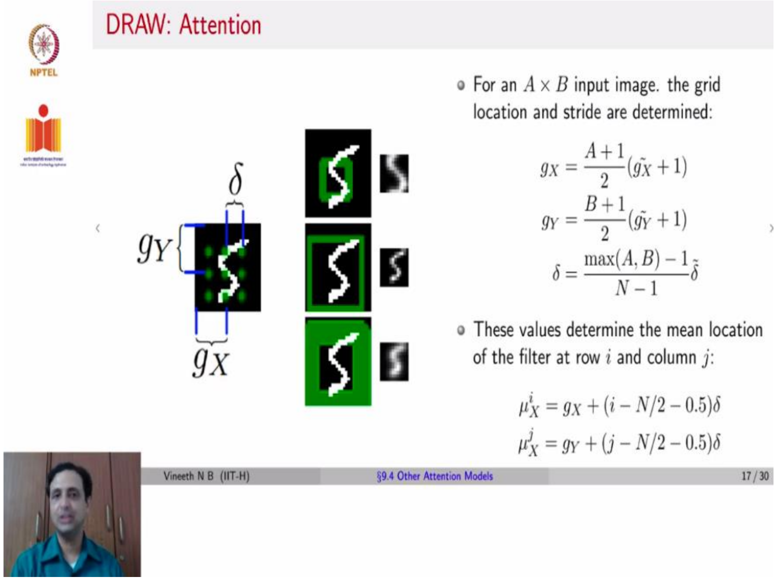
Vineeth N B (IIT-H) §9.4 Other Attention Models 16 / 30

So what are these grid locations and what is done to get the attention? So at each of these grid locations, let us say it is 3x 3 or Nx N, in general, a Gaussian filter is placed at each of these grid locations. So, if you have a Gaussian filter, you have to specify a variance for that Gaussian. So in addition to the grid center coordinates gx gy, and the stride δ, the model also learns the variance of the Gaussian σ square, which is going to be placed at each of these grid locations.

And also an intensity value γ, which we will see very soon on how it is used, how it is used. How do you learn all of these parameters, these are all learned as part of the read or write module. So in the case of the decoder or the right module, given the hidden state of the decoder at a particular time step, you have a set of weights, which outputs all of these values that we want gx gy, σ^2, δ, and γ.

Why is it $\tilde{\delta}$, $\widetilde{gx}$, $\widetilde{gy}$? We will see that very soon. Just it is only to get a go from absolute locations to relative locations. We will see that now.

(Refer Slide Time: 21:45)



DRAW: Attention

- For an $A \times B$ input image. the grid location and stride are determined:

$$g_X = \frac{A+1}{2}(\tilde{g_X}+1)$$
$$g_Y = \frac{B+1}{2}(\tilde{g_Y}+1)$$
$$\delta = \frac{\max(A,B)-1}{N-1}\tilde{\delta}$$

- These values determine the mean location of the filter at row i and column j:

$$\mu_X^i = g_X + (i - N/2 - 0.5)\delta$$
$$\mu_X^j = g_Y + (j - N/2 - 0.5)\delta$$

Vineeth N B (IIT-H) §9.4 Other Attention Models 17 / 30

So given a gx hat, and the gy hat, which is the center of this grid, which could be a value lying between 0 and 1, if we want to translate this to a particular position on the image, and let us assume the image is an Ax B input image, then gx would be given by a plus 1 by 2 into detail the x plus 1 and gy will be given by B plus 1 by 2 into gy tilde plus 1, we are converting the predicted values gx tilde and gy tilde in the specific positions on an Ax B image given by gx and gy.

Similarly, δ tilde could be the stride in the same dimensions that you were looking at for g tilde, now, you are converting that to a stride on a given Ax B image by saying that $\delta = \frac{\max(A,B)-1}{(N-1)}\tilde{\delta}$. N is the number of grid locations that you have, as we said, a 3x 3 grid, or a 5x 3 5x 5 grid, so on and so forth.

What you see in the figure here is a 3x 3 grid attention map, in that case, N would be 3. So you are considering the $\max(A,B)$. If A and B were 100x at for instance, if that was your side size of the image, this would be 100-1, 99/3-1,2. So you would get you would be multiplying $\tilde{\delta}$ by a factor of 49, to be able to separate the at, the grid by the stride that is required for that image and gets a δ.

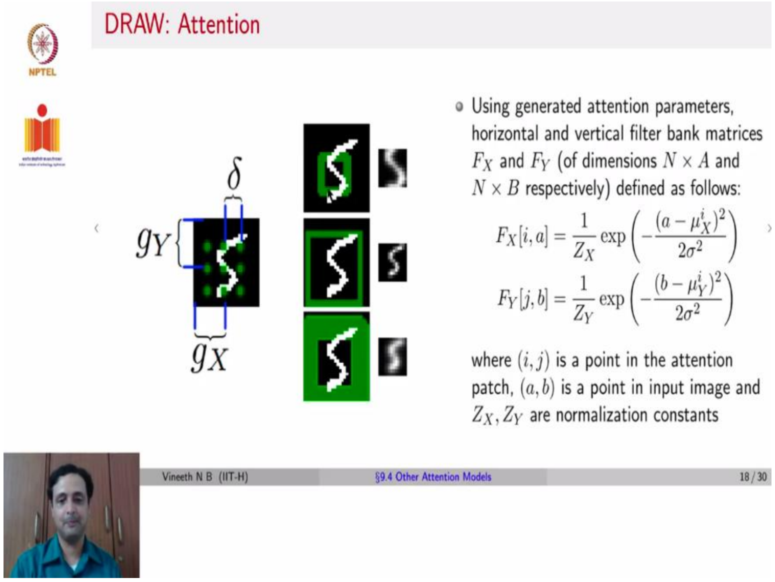
Now, to find out the grid point locations of this say 3x 3 grid, you have the final equations here, which are given by μ_X^i and μ_X^j, which gives us the location for the ij th coordinate of that 3x 3 grid.

So let us say the 1x 1 or the 1x 2 or the 1x 3, so on and so forth, is so the $\mu_X^i = g_x + (i - \frac{N}{2} - 0.5)\delta$. Similarly, $\mu_X^j = g_y + (i - \frac{N}{2} - 0.5)\delta$.

This would give you the locations if gx gy was the location of a particular of the center of the grid. Let us assume that that was at 50, 40, for instance, then i minus N by 2 would give you so if you are looking at the first location, N by 2 would be 3 by 2, so that would be minus 0.5, minus 0.5, that would get a minus 1. And you would say gx minus 1 δ and if δ was 10 pixels, then you would go 10 times to the left 10 pixels to the left.

Similarly for j, you would get 10 pixels to the top, and that would give you the top-left location of the grid point and so on.

(Refer Slide Time: 25:10)



DRAW: Attention

- Using generated attention parameters, horizontal and vertical filter bank matrices F_X and F_Y (of dimensions $N \times A$ and $N \times B$ respectively) defined as follows:

$$F_X[i,a] = \frac{1}{Z_X} \exp\left(-\frac{(a - \mu_X^i)^2}{2\sigma^2}\right)$$

$$F_Y[j,b] = \frac{1}{Z_Y} \exp\left(-\frac{(b - \mu_Y^i)^2}{2\sigma^2}\right)$$

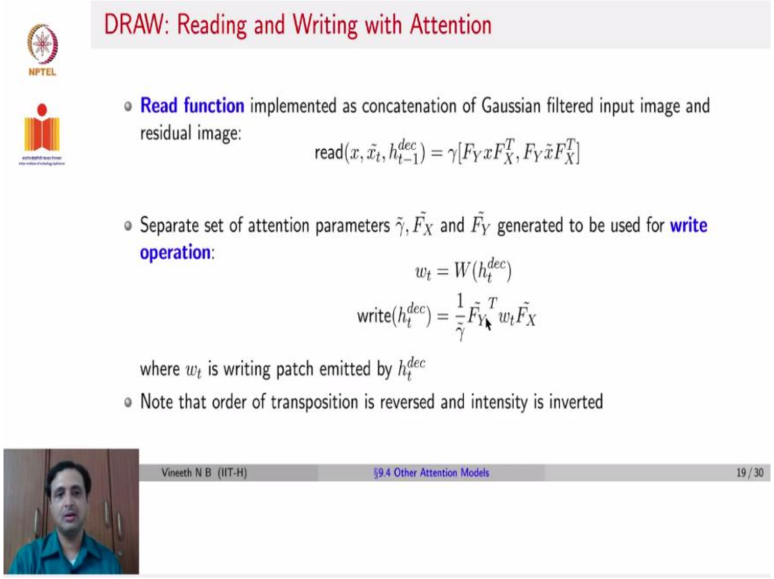
where (i,j) is a point in the attention patch, (a,b) is a point in input image and Z_X, Z_Y are normalization constants

Vineeth N B (IIT-H) §9.4 Other Attention Models 18 / 30

What do you do once you have these grid locations, as we said, at each of these grid locations, you place a Gaussian filter given by the variance σ^2 Fx and Fy denote those Gaussian filters here? So, the Gaussian filter placed at a specific grid location, which could be at a better particular part of the image gives you different portions of the image. So, what you see here are the bottom-most and the second from the bottom are perhaps positioned at the same central grid location.

But they have different Gaussian variances, which you can see in the output of the attended region. And on top, the center grid location, and perhaps the δ is small, so, you are focusing on a narrower location with its variance for the Gaussian.

(Refer Slide Time: 26:13)



DRAW: Reading and Writing with Attention

- **Read function** implemented as concatenation of Gaussian filtered input image and residual image:

$$\text{read}(x, \tilde{x}_t, h_{t-1}^{dec}) = \gamma[F_Y x F_X^T, F_Y \tilde{x} F_X^T]$$

- Separate set of attention parameters $\tilde{\gamma}, \tilde{F_X}$ and $\tilde{F_Y}$ generated to be used for **write operation**:

$$w_t = W(h_t^{dec})$$

$$\text{write}(h_t^{dec}) = \frac{1}{\tilde{\gamma}} \tilde{F_Y}^T w_t \tilde{F_X}$$

where w_t is writing patch emitted by h_t^{dec}

- Note that order of transposition is reversed and intensity is inverted

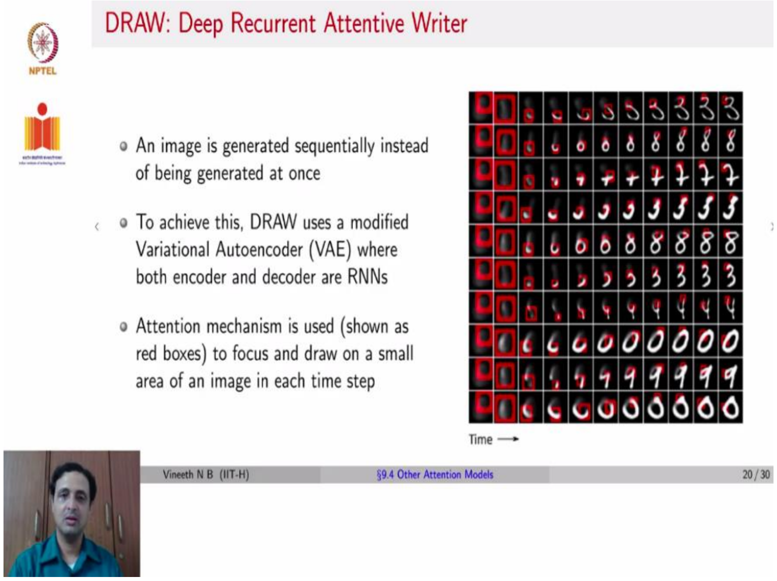
Vineeth N B (IIT-H) §9.4 Other Attention Models 19 / 30

To complete this discussion, the read function is finally implemented as a concatenation of a Gaussian filtered input image and the residual image. So, you have x you have $\hat{x}$, which is your residual image, and Fy, an Fx are your Gaussian filters. So, you concatenate them, you apply γ, which was an intensity factor, if you recall, so, you can now vary the intensity and then γ is also learned. Why do we have the intensity factor?

You could assume that when you draw, you may initially draw a light shade and then make it darker, right? So, in a specific time step γ controls how much intensity that you want to put into generating that part of the image in this case read but γ is also used for the right even in read, you can read only a certain intensity of a part of the image. And the right operation follows a complement where the right operation is given by $\frac{1}{\delta} F_y^T w_t F_x^T$.

So, you can see here that for the right patch, the order of the transposition is reversed. Here you have $F_y x F_x^T$, here you have $F_y^T w_t F_x^T$. So, the order of the transporters, transposition is reversed to maintain the dimensions and also to maintain complementarity. But otherwise, the attention mechanism is implemented in a very similar manner for the read and the write.

(Refer Slide Time: 27:58)



DRAW: Deep Recurrent Attentive Writer

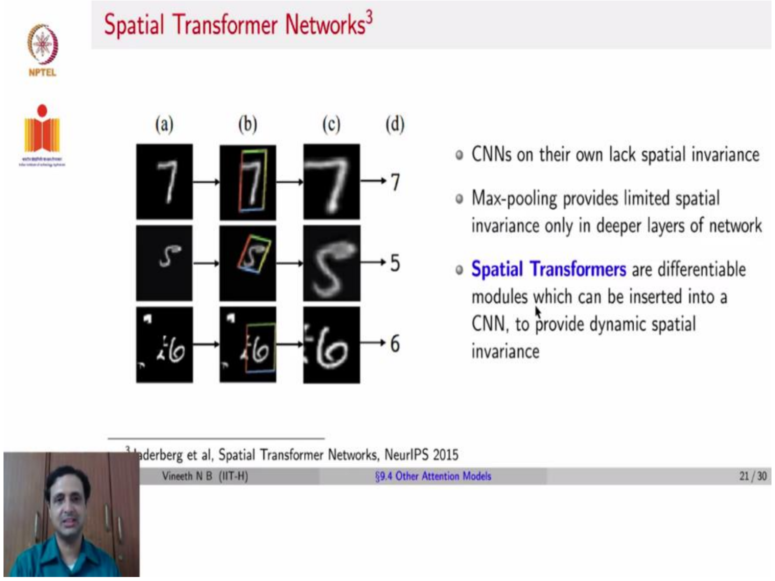
- An image is generated sequentially instead of being generated at once
- To achieve this, DRAW uses a modified Variational Autoencoder (VAE) where both encoder and decoder are RNNs
- Attention mechanism is used (shown as red boxes) to focus and draw on a small area of an image in each time step

Vineeth N B (IIT-H) §9.4 Other Attention Models 20 / 30

And to recall, once you have this attention mechanism, at each time step, each of these rows is different generations. So, at each time step, the grid location and the Gaussians tell you the grid locations, the stride, and the Gaussians tell you which region in the image you are focusing on. And in each time step, the region that you are focusing on can change. In this case, what you are seeing here is the output of the right head.

And you can see that you are seeing different regions are in each region of the canvas, you are drawing something which is obtained as the output of the decoder. And you repeat these over time steps to get your final generated image.

(Refer Slide Time: 28:51)



Spatial Transformer Networks[3]

- CNNs on their own lack spatial invariance
- Max-pooling provides limited spatial invariance only in deeper layers of network
- **Spatial Transformers** are differentiable modules which can be inserted into a CNN, to provide dynamic spatial invariance

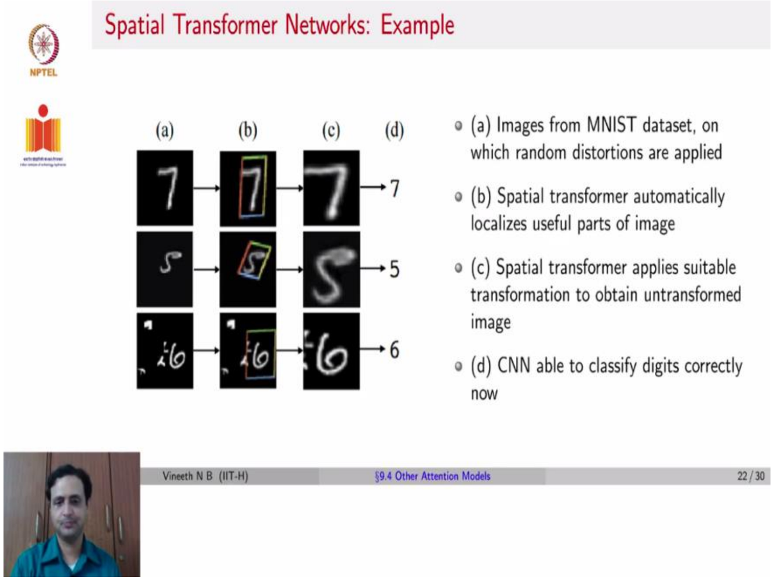
[3] aderberg et al, Spatial Transformer Networks, NeurIPS 2015

Vineeth N B (IIT-H) §9.4 Other Attention Models 21 / 30

The third method that we are going to talk about in this lecture is known as Spatial Transformer networks, which is again a very different approach to use attention and benefit from the value attention brings to CNNs. So, the broad idea here is that CNNs on their own can lack spatial invariance, so if a certain object was rotated in different ways, was located at different corners, it may not work in practice.

To a certain extent, Max pooling operations help, but they only help with variations or invariance within a small neighborhood, especially in deeper parts of the networks. Spatial transformers are a set of our comprises is a fully differentiable module, and hence can be inserted in any CNN to bring a certain attention module to solve a particular problem. Let us see spatial Transformers in more detail.

(Refer Slide Time: 30:01)



Spatial Transformer Networks: Example

- (a) Images from MNIST dataset, on which random distortions are applied
- (b) Spatial transformer automatically localizes useful parts of image
- (c) Spatial transformer applies suitable transformation to obtain untransformed image
- (d) CNN able to classify digits correctly now

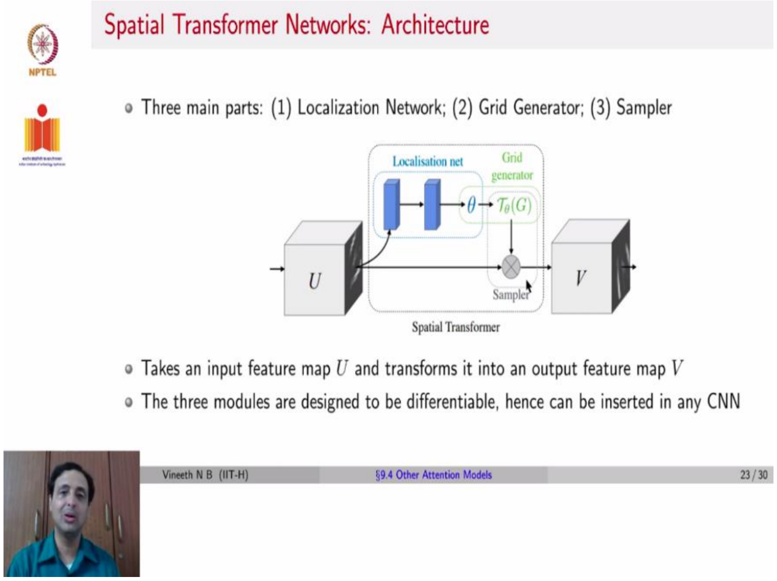
Vineeth N B (IIT-H) §9.4 Other Attention Models 22 / 30

So here is an example of how spatial transformer networks work. So in the first column, you have three different images. As you can see, in the first image 7 is slightly modified from how a 7 would look like in an MNIST data set. Similarly, a 5 is reduced in size and rotated a little bit, and a 6 is moved from the central patch and there are a few noise patches that are included to confuse a model that may want to look at this and classify it as a digit 6.

What spatial transformer does is shown in column B, its job is to find out which part of the image to focus on, that is wherein a sense, the attention mechanism comes in. So you can see here that the top image, focuses on this part, which is the 7. Similarly in the middle image, it focuses on 5, but the rotated box shows the orientation in which the model needs to look at the content. And similarly, for the third image, which is 6.

In column C, the spatial transformer applies this transformation and converts this original image to an untransformed uncorrupted version, where you can find each digit to occupy the full image and be centered. And with the kind of images that you get in column C. In column D, we show how a CNN can now give good classification performance if you attend to the specific part of the image. Rather than be confused with the full image, which could have other artifacts.

(Refer Slide Time: 31:52)



Spatial Transformer Networks: Architecture

- Three main parts: (1) Localization Network; (2) Grid Generator; (3) Sampler

- Takes an input feature map U and transforms it into an output feature map V
- The three modules are designed to be differentiable, hence can be inserted in any CNN

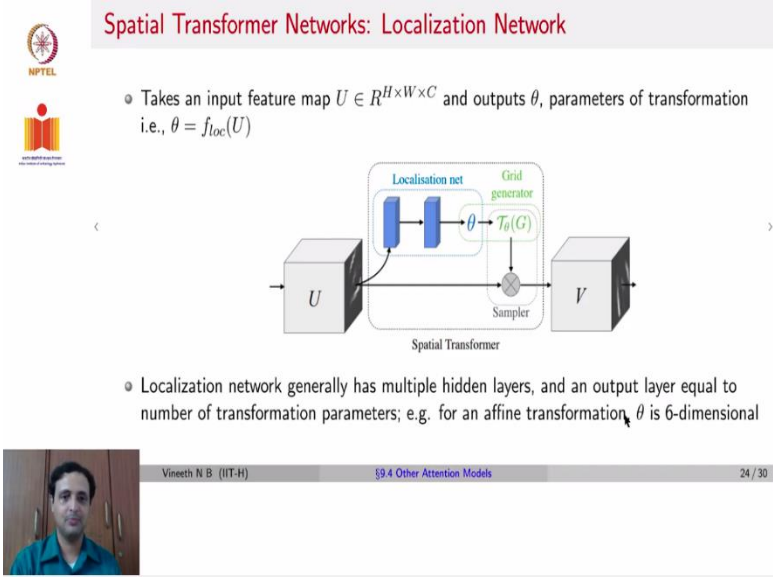
Vineeth N B (IIT-H) §9.4 Other Attention Models 23 / 30

The spatial transformer network architecture consists of three modules, a Location network, a Localization network, a Grid generator, and a Sampler. As you can see here, the spatial transformer network module is something that you can insert between any two convolution layers in a CNN. So if this was Con 4, you could insert a spatial transformer module in between Con 4 and Con 5.

What do you expect to happen? If this was an image, where the 6 was rotated, taken to a corner, there were other noisy artifacts in the image, we expect the spatial transformer to focus on only region 6 and pass it on to the next step V for further processing. Why is this challenging? Why is this required? Remember now, that in a sense, we are doing Hard Attention, we are now trying to focus on a specific part in the image which could vary for each input.

How do we manage to do this in a differentiable manner is the core contribution of the Spatial Transformer networks. The three modules are Localization network, Grid generator, and a Sampler. So the idea is to take an input feature map U and transform it into an output feature map V, which can then be given for further processing, such as to perform classification. So let us see each of these modules which are differential, which is differentiable and designed in a way to be differentiable so that they can be included in any CNN.

(Refer Slide Time: 33:49)



Spatial Transformer Networks: Localization Network

- Takes an input feature map $U \in R^{H \times W \times C}$ and outputs θ, parameters of transformation i.e., $\theta = f_{loc}(U)$

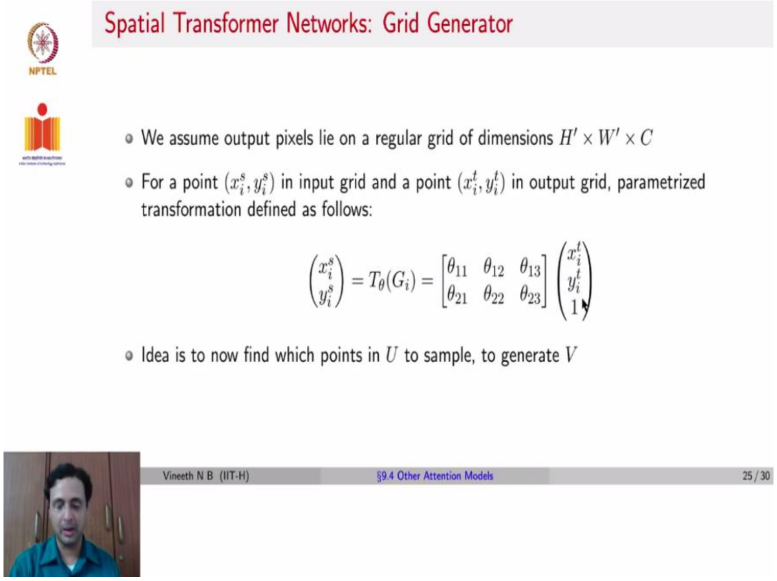
- Localization network generally has multiple hidden layers, and an output layer equal to number of transformation parameters; e.g. for an affine transformation, θ is 6-dimensional

Vineeth N B (IIT-H) §9.4 Other Attention Models 24 / 30

The localization network takes an input feature map U, which let us assume is of dimension Hx Wx C. And it outputs a set of parameters theta, which we are going to say are the parameters of the transformation. What does this mean? A localization network could have multiple hidden layers, but the output layer of the localization network contains the number of transformation, transformation parameters that may be required to model the problem.

For example, if we wanted spatial transformer networks to deal with only fine transformations, then you would have to output six different values that would be the Localization network.

(Refer Slide Time: 34:43)



Spatial Transformer Networks: Grid Generator

- We assume output pixels lie on a regular grid of dimensions $H' \times W' \times C$
- For a point (x_i^s, y_i^s) in input grid and a point (x_i^t, y_i^t) in output grid, parametrized transformation defined as follows:

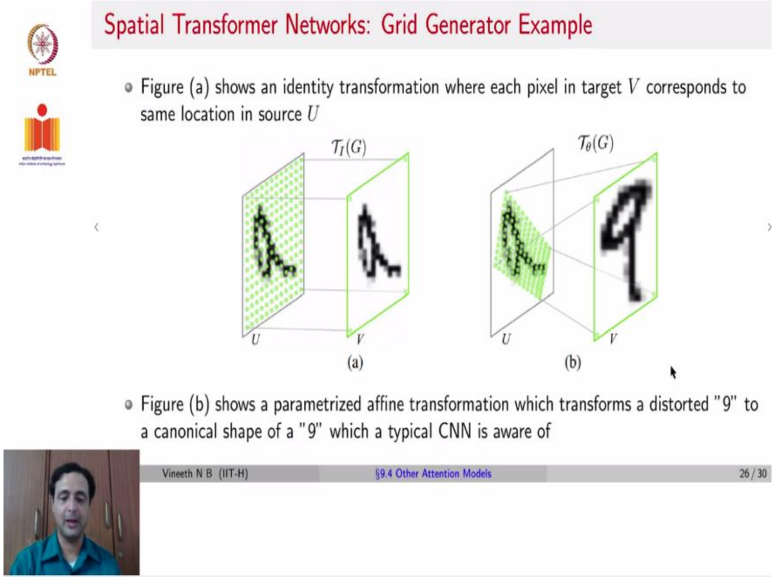
$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = T_\theta(G_i) = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

- Idea is to now find which points in U to sample, to generate V

The next step in the pipeline, as we saw was a Grid generator and the final step was a Sampler. So in the grid generator, we want to now take the transmission values and find out the actual grid. So, if we assume that the output V has dimensioned H'xW'x C for a particular point x_i^s , y_i^s in the input grid and a point x_i^t, y_i^t in the output grid, the transformation would be given by x_i^s , y_i^s would be given by an affine transformation of the point x_i^t, y_i^t.

Remember, affine can include rotation, scaling, and translation for the moment. So, now, we have an equivalence between the coordinates in the output feature map V and the coordinates in the input feature map U. So, now the next step is to find out how do you sample the points in U in a differentiable manner.

(Refer Slide Time: 35:59)



Spatial Transformer Networks: Grid Generator Example

- Figure (a) shows an identity transformation where each pixel in target V corresponds to same location in source U

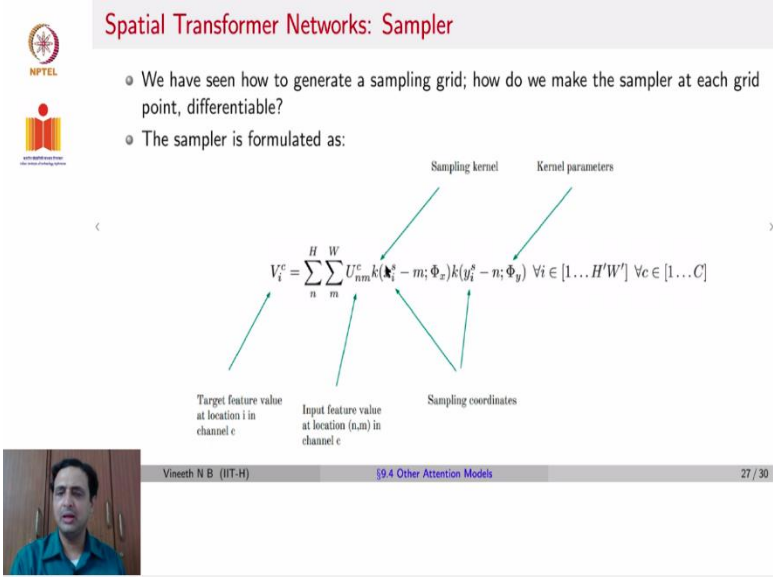
- Figure (b) shows a parametrized affine transformation which transforms a distorted "9" to a canonical shape of a "9" which a typical CNN is aware of

Vineeth N B (IIT-H) §9.4 Other Attention Models 26 / 30

Before we go there, let us see an example of the grid generator. So, here you see that in Figure a here, you have an identity transformation, where you have an original image and the transformation learned retains the same positions for every pixel from U to V. And here is another example B where U has rotated 9 and V learns a transformation that takes a particular set of pixels in U and maps them to all four corners of V.

Now, we need to find out how do you sample these pixels, which would form the output pixels inside V. So, you can see that in this case, a distorted 9 has got transformed to a canonical 9, which is CNN is aware of and can classify.

(Refer Slide Time: 37:00)



Spatial Transformer Networks: Sampler

- We have seen how to generate a sampling grid; how do we make the sampler at each grid point, differentiable?
- The sampler is formulated as:

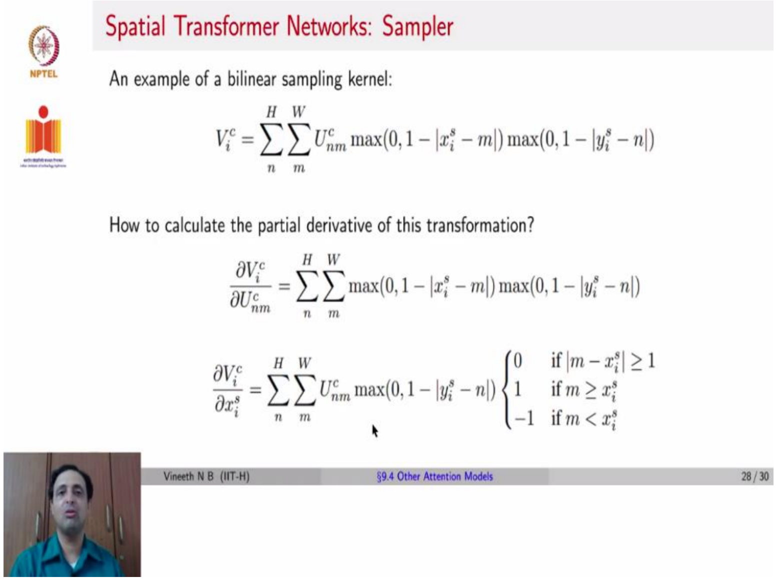
$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m; \Phi_x) k(y_i^s - n; \Phi_y) \ \forall i \in [1 \dots H'W'] \ \forall c \in [1 \dots C]$$

Sampling kernel
Kernel parameters
Target feature value at location i in channel c
Input feature value at location (n,m) in channel c
Sampling coordinates

Vineeth N B (IIT-H) §9.4 Other Attention Models 27 / 30

This leads us to the final and important step sampler which draws this equivalence between V and U. How do we do this? Vic denotes the target feature value at i th location in channel c. U_{nm}^c denotes the input feature value at location nm in channel c. And then you have iterates that go to H and W which are the entire dimensions for each channel in U. And then you have a sampling kernel, which, which tells you at which location x_i^s and x_i and y_i^s you need to sample to get a particular output in the output feature map V. So, what kind of a kernel do we can we use?

(Refer Slide Time: 37:54)



Spatial Transformer Networks: Sampler

An example of a bilinear sampling kernel:

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

How to calculate the partial derivative of this transformation?

$$\frac{\partial V_i^c}{\partial U_{nm}^c} = \sum_n^H \sum_m^W \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

$$\frac{\partial V_i^c}{\partial x_i^s} = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |y_i^s - n|) \begin{cases} 0 & \text{if } |m - x_i^s| \geq 1 \\ 1 & \text{if } m \geq x_i^s \\ -1 & \text{if } m < x_i^s \end{cases}$$

Vineeth N B (IIT-H) §9.4 Other Attention Models 28 / 30

One example that you can use is a bilinear sampling kernel, which states that the kernel k can be given by $\max(0{,}1 - |x_i^s - m|)$. Similarly, $\max(0{,}1 - |y_i^s - n|)$. What does this do? If you observe, let us assume that for a particular pixel location in V, the corresponding x_i^s and y_i^s given by the transformation was, say 45 and 34 for some reason. So, 45 minus m is an iterator that would go from 1 to capital W and n is an iterator that would go from 1 to H.

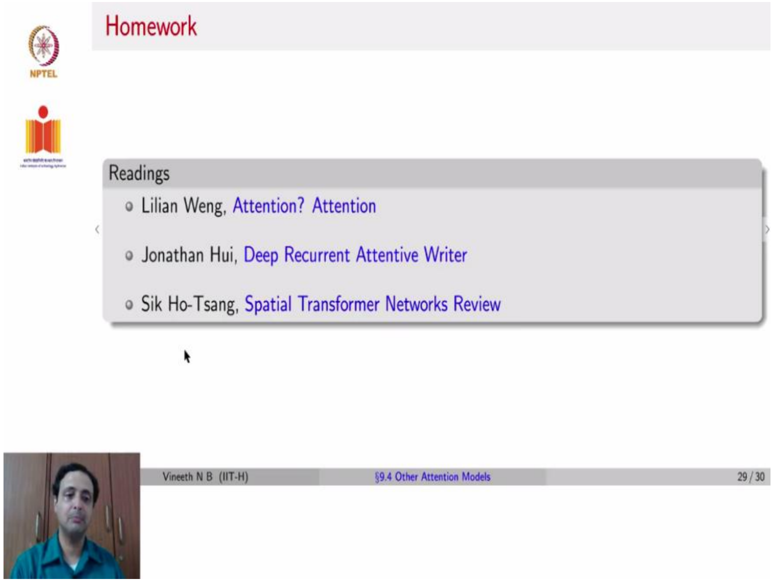
So, you can see here that if x_i^s were 45 and m was 50, for instance, you would then have 5, 1-5 would be minus 4, the max would be 0 and you would not sample the value at 50. So, where would you sample? This would get activated only when m is equal to x_i^s. So, you would only sample the pixel at x_i^s to get the corresponding location at V_i^c. So, this becomes a differentiable way of sampling and focusing on one specific region of U to get the subsequent feature map V.

How do you backpropagate through such a kernel? We know how to backpropagate through the rest of the operations. But how do you backpropagate through such a sampling kernel? It is not hard. The $\max(0{,}1 - |x_i^s - m|)$. is similar to many other operations that we have seen so far for differentiation. So, $\frac{\partial V}{\partial U}$ would simply be the both these max terms themselves.

So, that is what you would get because you would have a U here, and $\frac{\partial V_i^c}{\partial x_i^s}$ would similarly be, $U_{nm}^c \max(0{,}1 - |y_i^s - n|)$. So, this first term which depends on x goes away, and you would have that this term would become 0, if $m - x_i^s$ are greater than equal to 1, it will be 1 if m is greater than x_i^s and minus 1 if m is less than x_i^s. That is how you can compute the partial derivative of this transformation.

And this completes each of the modules, the Localization network, the Grid generator, and the Sampler to be able to go to one specific part of the input image for use for further processing.

(Refer Slide Time: 40:54)



Homework

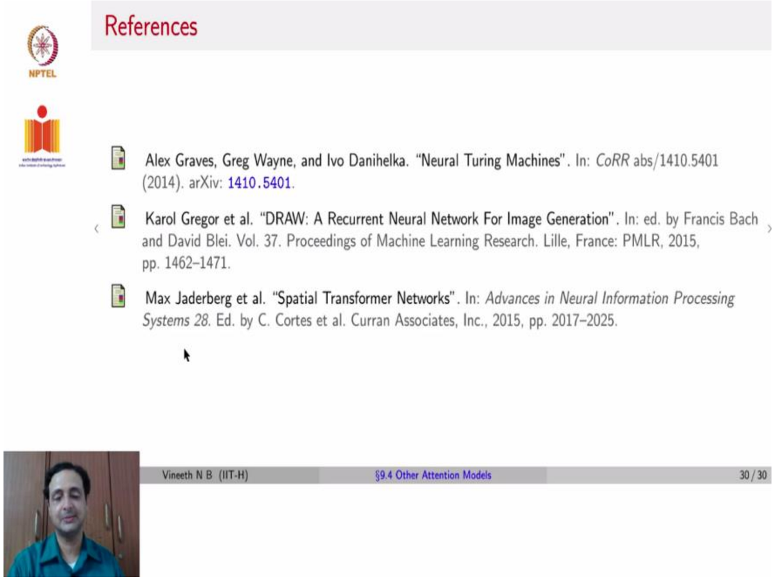
Readings

- Lilian Weng, Attention? Attention
- Jonathan Hui, Deep Recurrent Attentive Writer
- Sik Ho-Tsang, Spatial Transformer Networks Review

Vineeth N B (IIT-H) §9.4 Other Attention Models 29 / 30

Your homework readings for this lecture are once again the nice blog of Lillian Wang on Attention. A nice introduction and overview of DRAW by Jonathan Hui and a nice review of Spatial Transformer networks by Sik Ho-Tsang.

(Refer Slide Time: 41:13)



References

Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines". In: *CoRR* abs/1410.5401 (2014). arXiv: 1410.5401.

Karol Gregor et al. "DRAW: A Recurrent Neural Network For Image Generation". In: ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 1462–1471.

Max Jaderberg et al. "Spatial Transformer Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 2017–2025.

Vineeth N B (IIT-H) §9.4 Other Attention Models 30 / 30

Here are references.