Deep Learning for Computer Vision

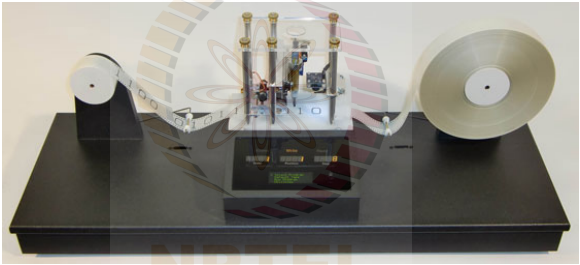
Other Attention Models in Vision

Vineeth N Balasubramanian

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

Neural Turing Machines[1]

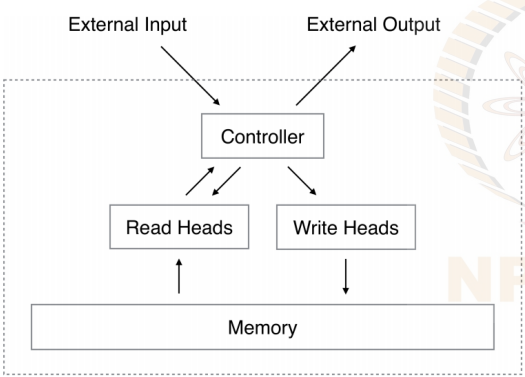
- Neural Turing Machines (or NTMs) are a class of neural networks designed to be analogous to a Turing Machine



- 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

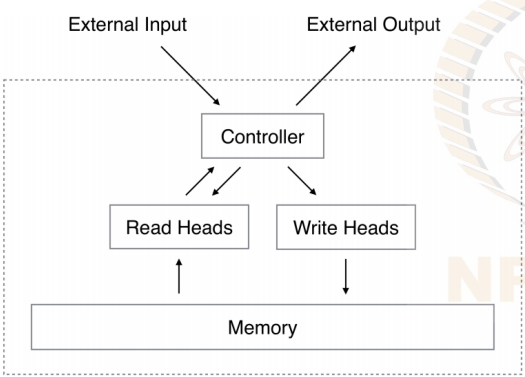
Neural Turing Machines



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

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

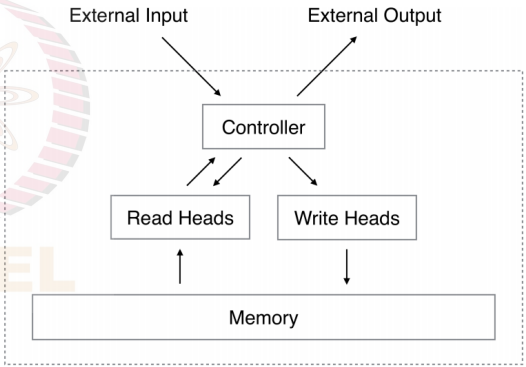
Question: Like a Turing Machine, if we do read/write operations on NTM by specifying a row and column index, can we train the NTM end-to-end?

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?

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



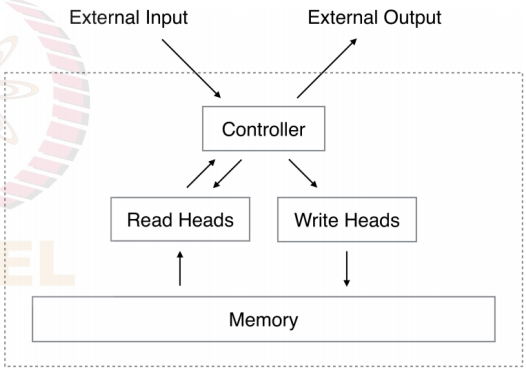
Neural Turing Machines: Reading

- 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:

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



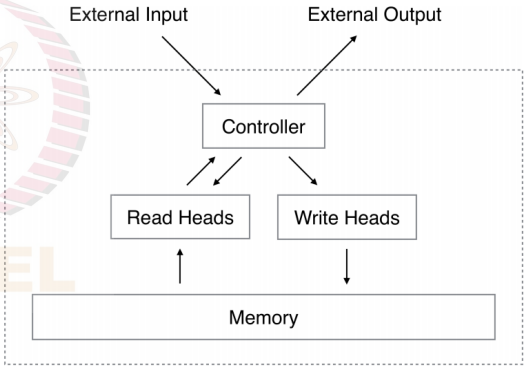
Neural Turing Machines: Writing

- 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

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



Neural Turing Machines: Attention

- How to generate attention vector w_t?

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?

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

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:

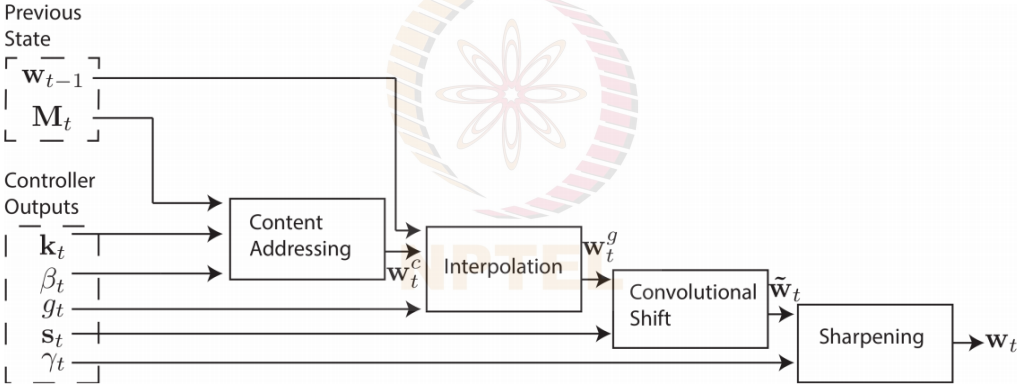
$$\tilde{w}_t(i) = \sum_{j=1}^{R} w_t^g(j) s_t(i - j)$$

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

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

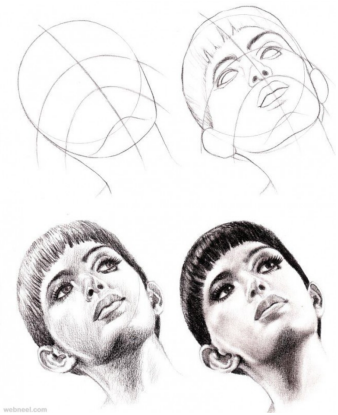
Neural Turing Machines: Attention

Summary of attention vector generation:



DRAW: Deep Recurrent Attentive Writer[2]

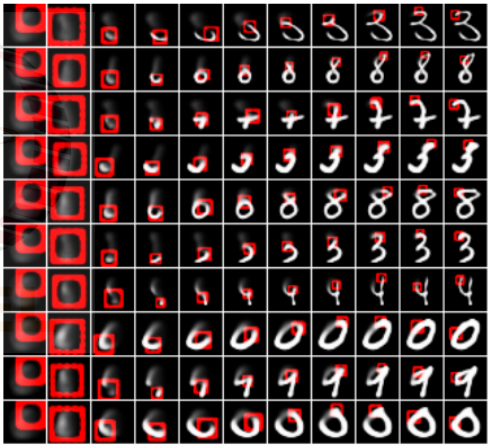
- Humans learn to draw figures in a sequential manner
- First, we draw outlines and then iteratively add more and more detail
- Deep Recurrent Attentive Writer (DRAW) is a VAE (a kind of autoencoder, which we will see soon) which aims to mimic this sequential drawing approach



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

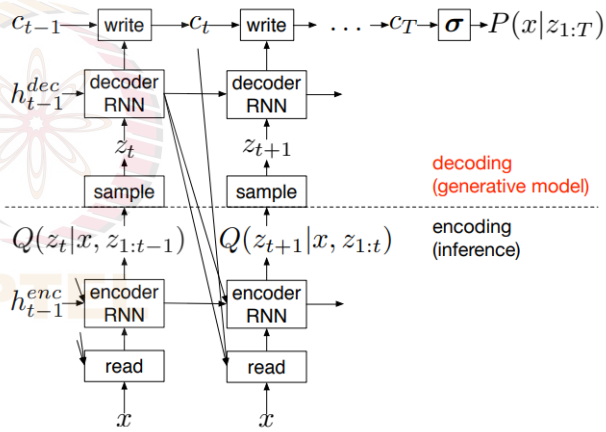
DRAW: Deep Recurrent Attentive Writer

- 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

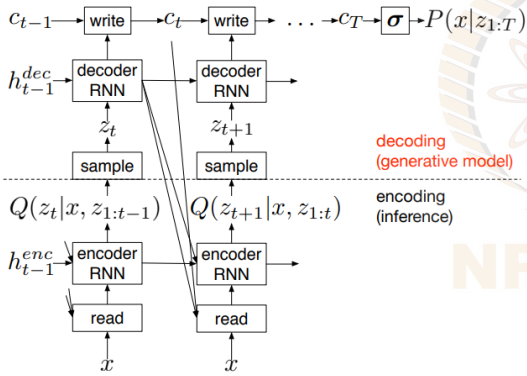


DRAW: Implementation

- 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

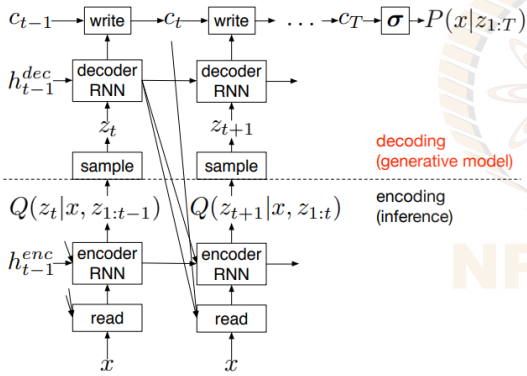


DRAW: Encoding



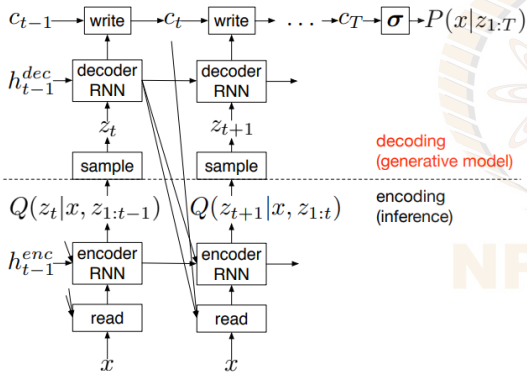
- Encoder takes four inputs:

DRAW: Encoding



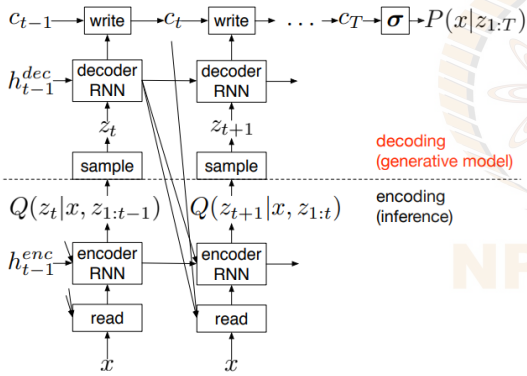
- Encoder takes four inputs:
  - input image x

DRAW: Encoding



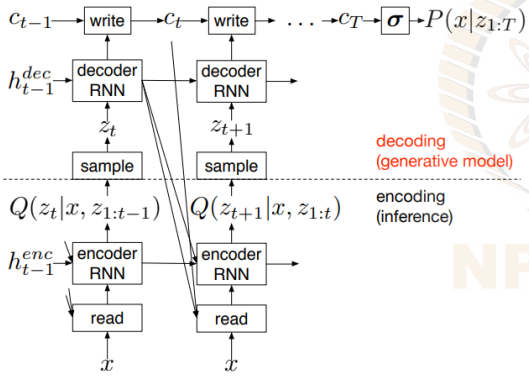
- Encoder takes four inputs:
  - input image x
  - residual image $\hat{x_t}$

DRAW: Encoding



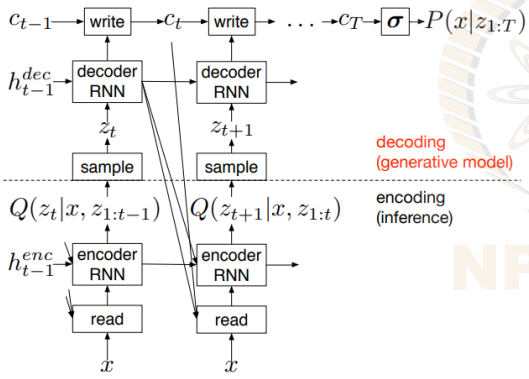
- Encoder takes four inputs:
  - input image x
  - residual image $\hat{x_t}$
  - encoder hidden state h_{t-1}^{enc}

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

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

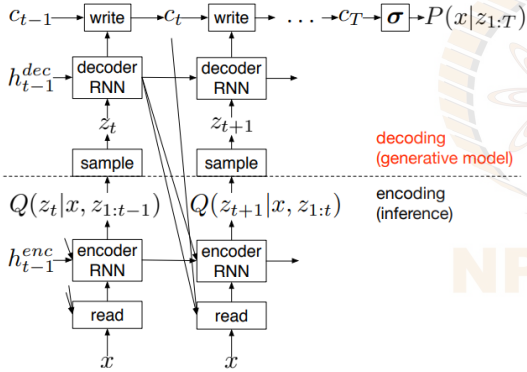
$$\hat{x}_t = x - \sigma(c_{t-1})$$

$$r_t = read(x_t, \hat{x}_t, h_{t-1}^{dec})$$

$$h_t^{enc} = RNN^{enc}(h_{t-1}^{enc}, [r_t, h_{t-1}^{dec}])$$

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

DRAW: Sampling and Decoding



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

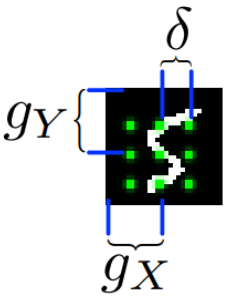
$$z_t \sim Q(Z_t|h_t^{enc})$$

(we will see more of this later)

- Decoder:

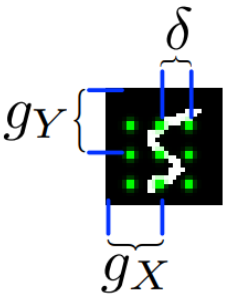
$$h_t^{dec} = RNN^{dec}(h_{t-1}^{dec}, z_t)$$
$$c_t = c_{t-1} + write(h_t^{dec})$$

DRAW: Attention



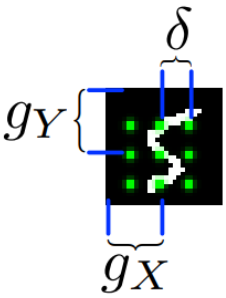
- Read and write functions used to attend to (focus on) certain regions in image

DRAW: Attention



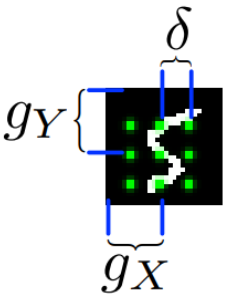
- Read and write functions used to attend to (focus on) certain regions in image
- Since picking one particular region of image (hard attention) is not differentiable, DRAW uses an array of 2D Gaussian filters, which gives out a patch of smoothly varying location and zoom

DRAW: Attention



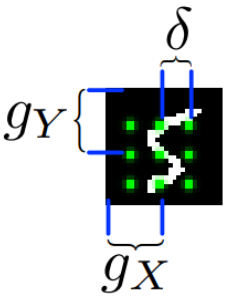
- Read and write functions used to attend to (focus on) certain regions in image
- Since picking one particular region of image (hard attention) is not differentiable, DRAW uses an array of 2D Gaussian filters, which gives out a patch of smoothly varying location and zoom
- Attention mechanism is done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

DRAW: Attention



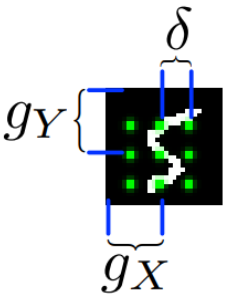
- $N \times N$ Gaussian filters parametrized by:
  1. Grid center coordinates: g_X, g_Y
  2. Stride δ
  3. Variance of Gaussian σ^2
  4. Intensity γ

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?

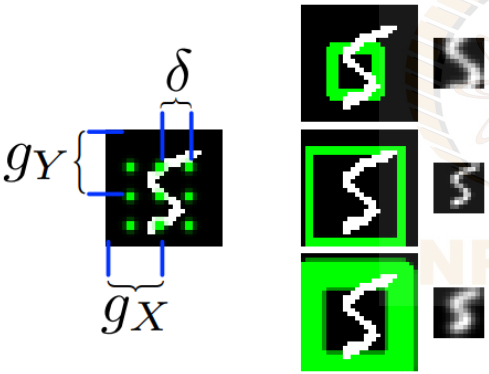
DRAW: Attention



- $N \times N$ 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:

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

DRAW: Attention



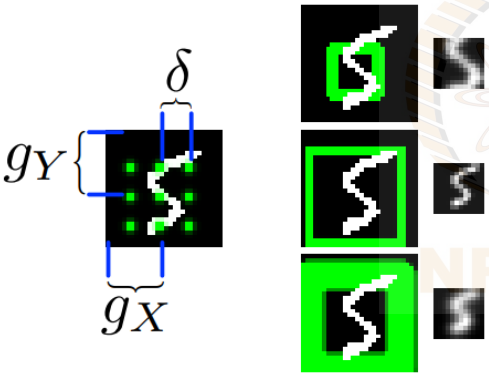
- For an $A \times B$ input image. the grid location and stride are determined:

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

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

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

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

DRAW: Reading and Writing with Attention

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

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

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}

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**:

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

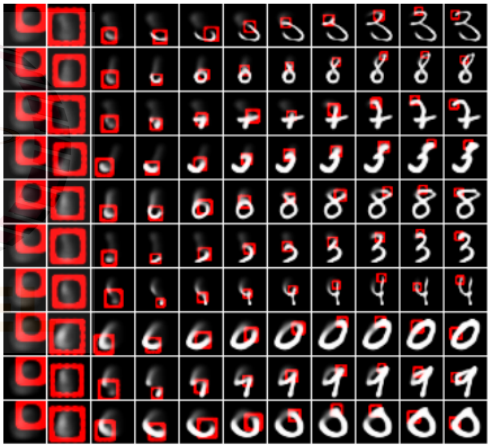
$$\text{write}(h_t^{dec}) = \frac{1}{\tilde{\gamma}} \tilde{F_Y}^T w_t \tilde{F_X}$$

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

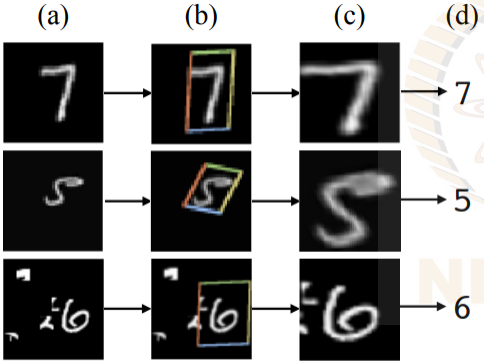
- Note that order of transposition is reversed and intensity is inverted

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



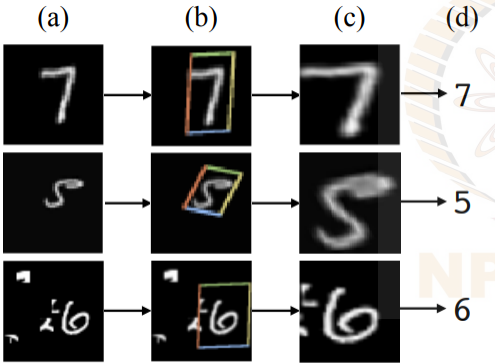
Spatial Transformer Networks[3]



- 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

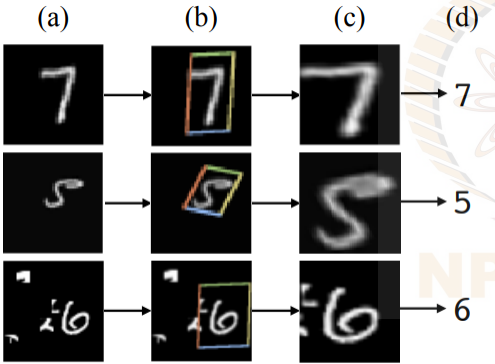
[3] Jaderberg et al, Spatial Transformer Networks, NeurIPS 2015

Spatial Transformer Networks: Example



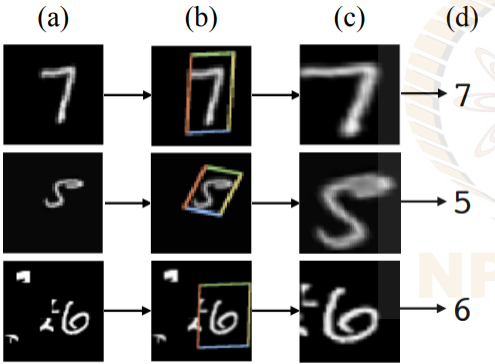
- (a) Images from MNIST dataset, on which random distortions are applied

Spatial Transformer Networks: Example



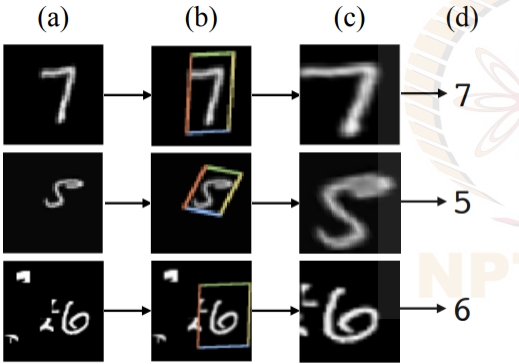
- (a) Images from MNIST dataset, on which random distortions are applied
- (b) Spatial transformer automatically localizes useful parts of image

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

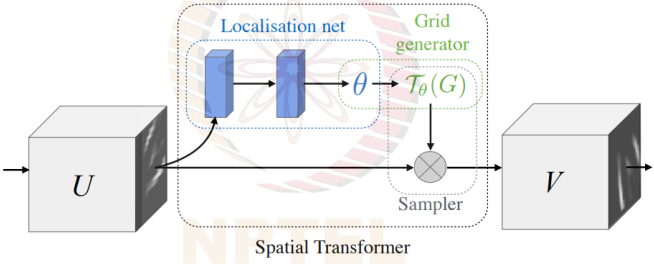
Spatial Transformer Networks: Example



- (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

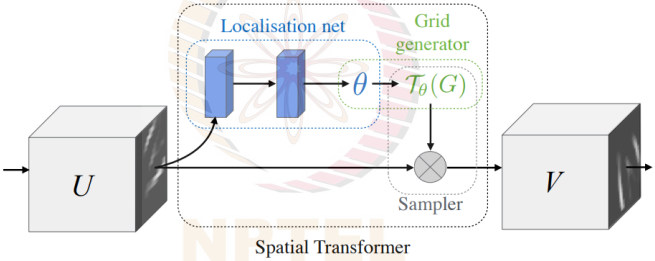
Spatial Transformer Networks: Architecture

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



Spatial Transformer Networks: Architecture

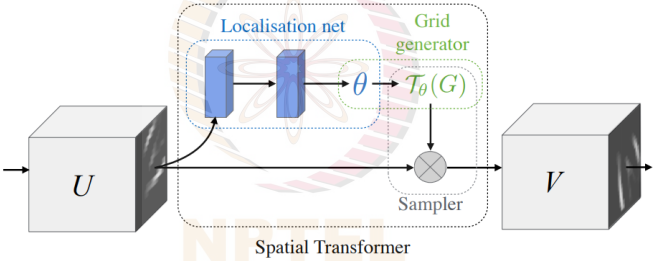
- Three main parts: (1) Localization Network; (2) Grid Generator; (3) Sampler



- Takes an input feature map U and transforms it into an output feature map V

Spatial Transformer Networks: Architecture

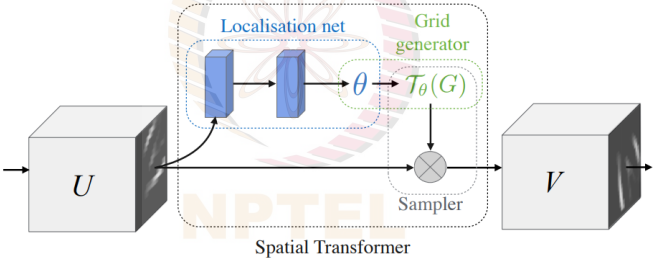
- Three main parts: (1) Localization Network; (2) Grid Generator; (3) Sampler



- 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

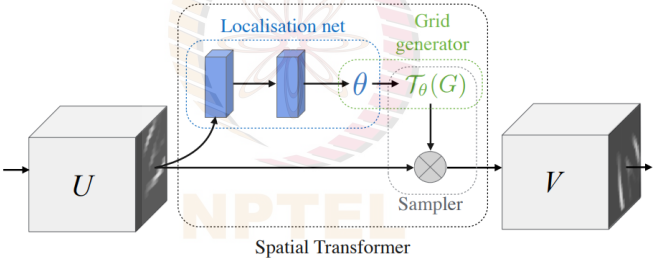
Spatial Transformer Networks: Localization Network

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



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

Spatial Transformer Networks: Grid Generator

- We assume output pixels lie on a regular grid of dimensions $H' \times W' \times C$

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:

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

Spatial Transformer Networks: Grid Generator

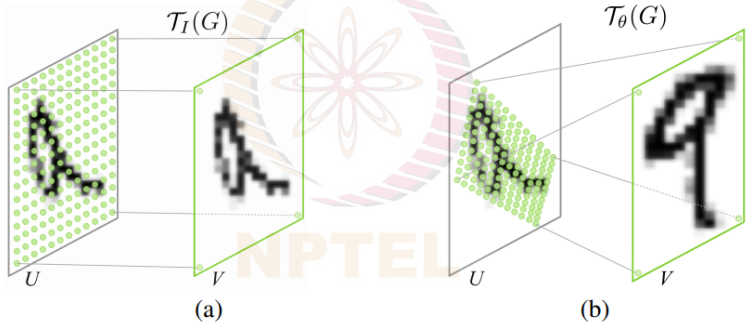
- We assume output pixels lie on a regular grid of dimensions $H' \times W' \times C$
- For a point (x_i^s, y_i^s) in input grid and a point (x_i^t, y_i^t) in output grid, parametrized transformation defined as follows:

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

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

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



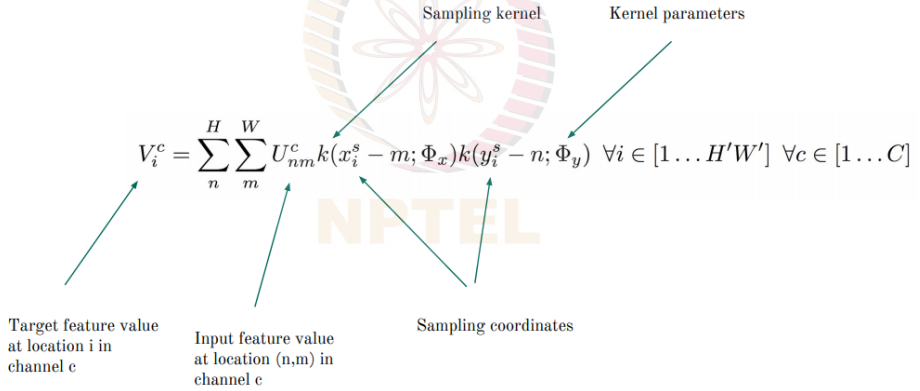
- Figure (b) shows a parametrized affine transformation which transforms a distorted "9" to a canonical shape of a "9" which a typical CNN is aware of

Spatial Transformer Networks: Sampler

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

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:



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?

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?

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

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

Homework

Readings

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

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.