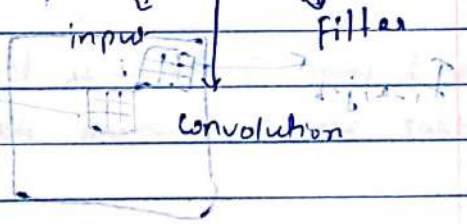
* Convolutional Neural Network

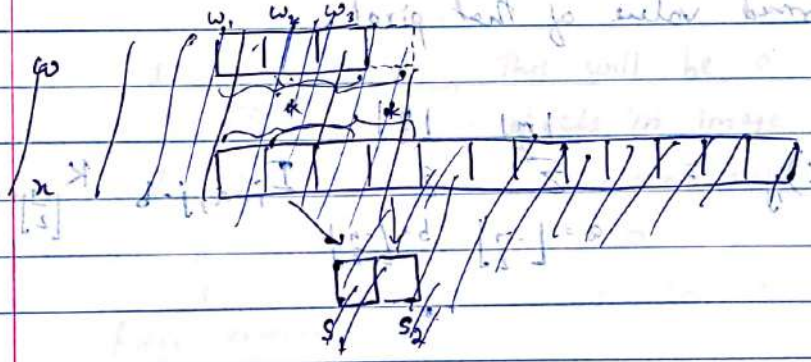
→ 1D Convolution

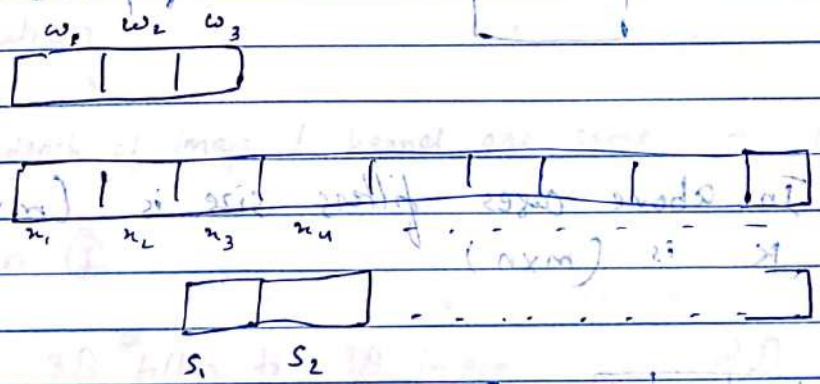
$$s_t = \sum_{a=0}^{n} x_{t-a}\, w_a \qquad \text{(n → size of filter)}$$

$$s_t = (x * w)_t$$



→ Let's say $n = 3$





$$s_1 = x_1 w_1 + x_2 w_2 + x_3 w_3$$

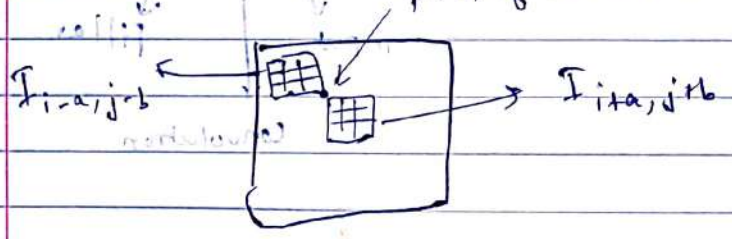
$$s_2 = x_2 w_1 + x_3 w_2 + x_4 w_3$$

⋮

→ 2D Convolution

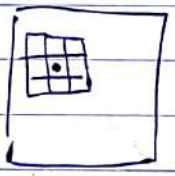
- $S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a,b}$

: lets say we have image



we multiply matrix after or before the pixel with filter to get transformed value of that pixel

- $S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a, j-b} K_{\lfloor \frac{m}{2} \rfloor + a, \lfloor \frac{n}{2} \rfloor + b}$



get value of pixel by multiplying matrix of pixels around it by filter

Note: In above cases filters size is (m x n) ie size of K is (m x n)

a	b	c	d
e	f	g	h
i	j	k	l

w	x
y	z

$\Rightarrow$

$aw+bx+ey+fz$	$bw+cx+fy+gz$	$cw+dx+gy+hz$
$ew+fx+iy+jz$	$fw+gx+jy+kz$	$gw+hx+ky+lz$

→ Types of Filters

- $\frac{1}{9}\begin{matrix}1&1&1\\1&1&1\\1&1&1\end{matrix}$ → replacing this pixel in image with weighted average of all pixel arround it

3 x 3

↑ Blur

- $\begin{matrix}0&-1&0\\-1&5&-1\\0&-1&0\end{matrix}$ → central pixel is made 5 times & all other pixels arround it are subtracted

↑ Sharpen

- $\begin{matrix}1&1&1\\1&-8&1\\1&1&1\end{matrix}$ → This will be 0 only if all the corresponding pixels in image are same

$= 1x+1x+1x+1x+-8x+1x+1x+1x+1x$

$= 0$

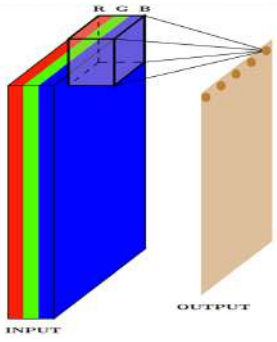
↑ Edge detector

→ 3D Convolution

Assume that depth of image & kernel are same — (1)

with assumption (1)

if we apply 3D filter to 3D image a 2D feature map is obtained



Ishan Modi

DELUXE
PAGE NO.:
DATE:

* Relation (input size, output size, filter size)

→ let an image have dimensions $W_1 \times H_1$
filter have dimensions $F \times F$

The transformed image will have dimensions $W_2 \times H_2$

$$W_2 = W_1 - F + 1 = W_1 - (F-1)$$
$$H_2 = H_1 - F + 1 = H_1 - (F-1)$$

→ In almost all cases we might have to add padding (Transform all pixels by taking them at centre)
$P = 1$ is added to all four size of input matrix
The formula then becomes as follows

$$W_2 = W_1 + 2P - F + 1$$
$$H_2 = H_1 + 2P - F + 1$$

→ We may keep a stride as per output dimensions needed

$$W_2 = \frac{W_1 + 2P - F}{S} + 1$$
$$H_2 = \frac{H_1 + 2P - F}{S} + 1$$

Main formula

(Q)

$227 \times 227 \times 3$ image * ($11 \times 11 \times 3$) 96 filter $\Rightarrow$ 9 output

$W_2 \times H_2 \times D$

Ans Since filter depth & img depth are same
Thus for each filter 2D feature map is obtained

Ishan Modi

Since there are 96 filters

$D = 96$

→ given → $s = 4$, $p = 0$

$$W_2 = \frac{227 - 11}{4} + 1 = 55$$

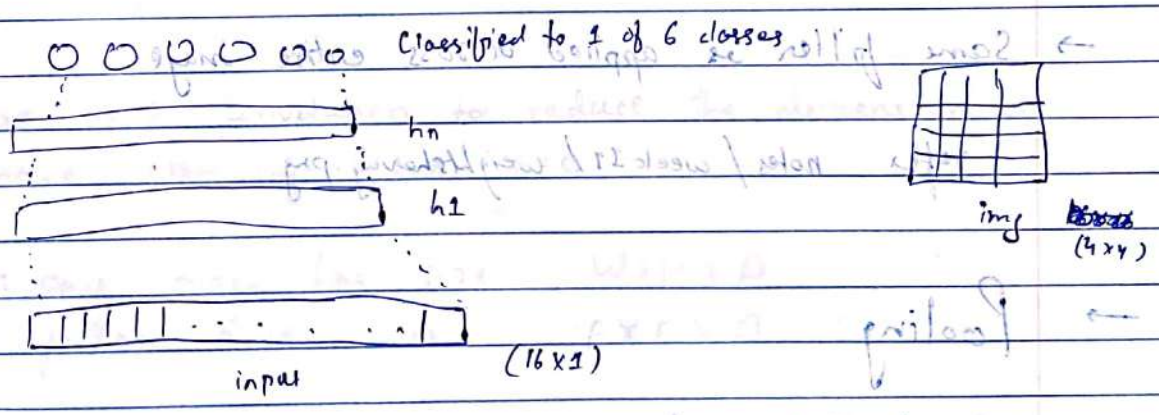
$$H_2 = \frac{227 - 11}{4} + 1 = 55$$

* CNNs

ML approach → Till now we focused on applying predefined static kernels/filters on image. Then we can learn weights to classify

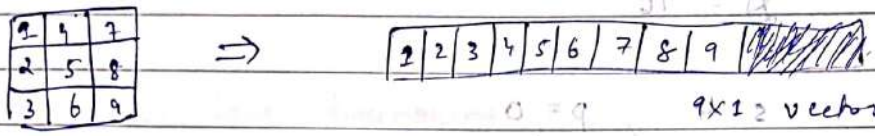
DL approach → Deep learning focuses on learning these filters. Then we learn weights to classify

→ For Feed Forward NN



It is a dense connection and there are a lot of parameters since each neuron in previous layer contributes to formation of neuron in next layer.

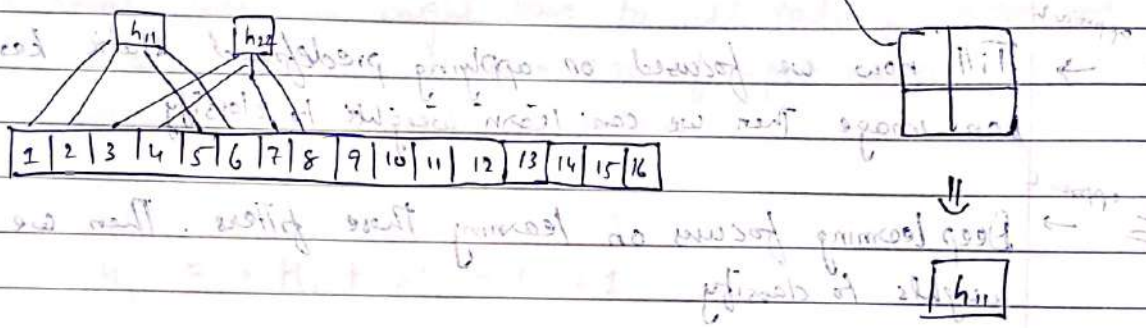
→ Convolution NN



3×3 img. 9×1 vector

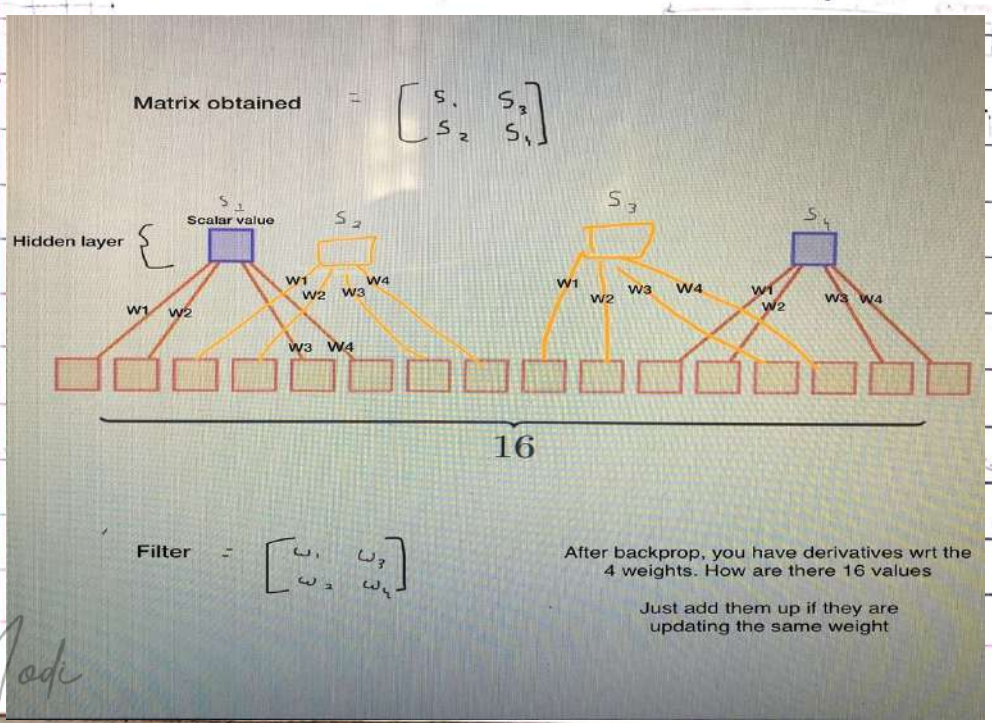
(1)

So this is comparatively sparse connection



(2) Weight Sharing

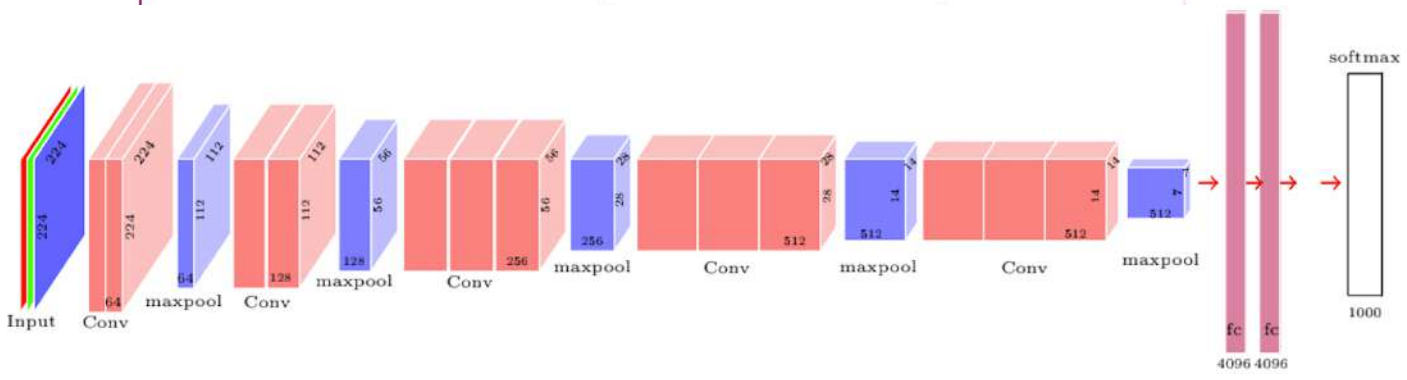
→ Same filter is applied accross entire image



Ishan Modi

---> ZFNet

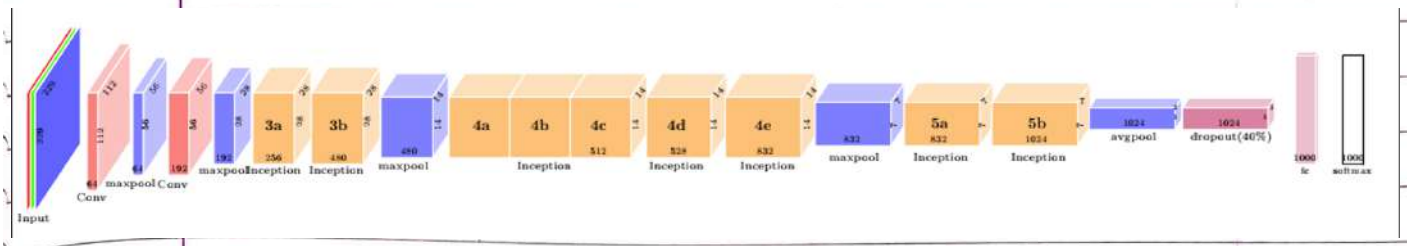
---> VGGNet



- Kernel size is 3×3 throughout
- Total parameters in non FC layers = $\sim 16M$
- Total Parameters in FC layers = $(512 \times 7 \times 7 \times 4096) + (4096 \times 4096) + (4096 \times 1024) = \sim 122M$
- Most parameters are in the first FC layer ($\sim$ 102M)

Ishan Modi

---> GoogleNet



- Use 1×1 convolutions to reduce the dimension of image assuming s=1, p=0

(i) let say image has size $W \times H \times D$
& filter size is $F \times F \times D$

Total Computations required = $F \times F \times D$
by each element of o/p

(ii) if we use $1 \times 1 \times D$ convolutions & D' such convolutions then

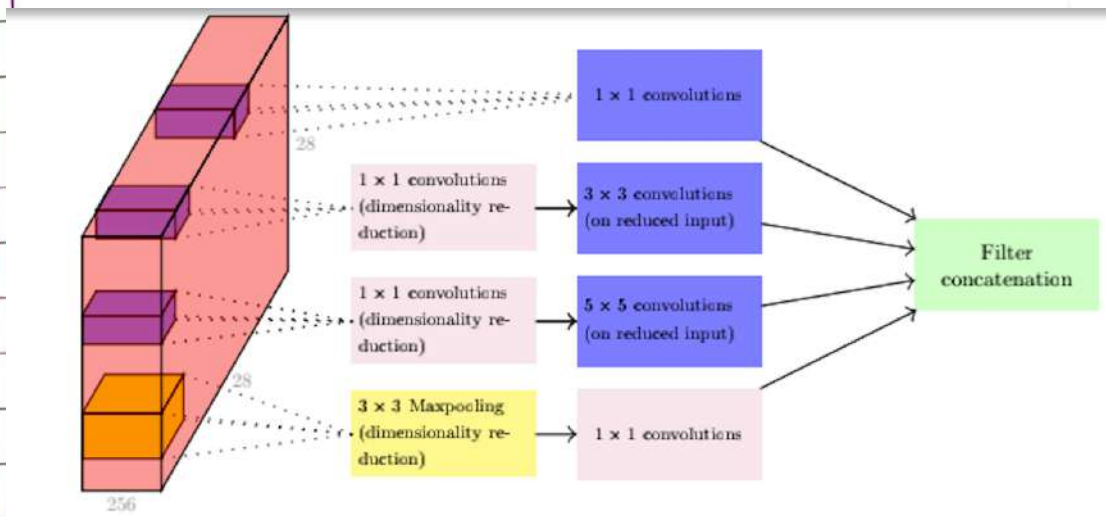
$(W \times H \times D) * (1 \times 1 \times D) \Rightarrow W \times H \times D'$

If $D' < D$ computations are reduced significantly

now applying filter size $F \times F \times D'$

Total computation for each element of o/p = $F \times F \times D'$ reduced

---> GoogleNet has various inception modules with basic structure as follows



Filter concatenation adds up all feature maps after applying appropriate padding

- In the last convolution layer

$7 \times 7 \times 1024 \xrightarrow{\text{avg pool}} 1 \times 1 \times 1024$

↓

equivalent to a vector can be connected with dense layer with less parameters than earlier

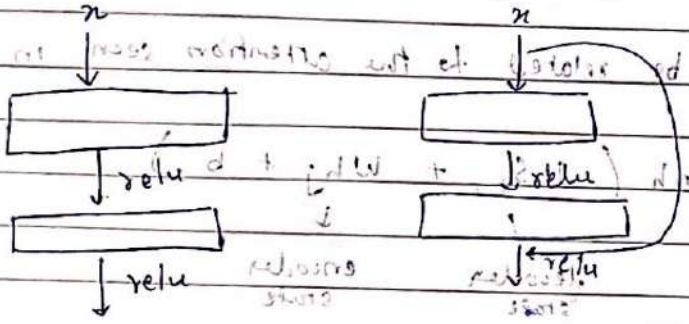
Ishan Modi

---> ResNet

Bag of tricks

- Batch Normalizaton after every CONV layer
- Xavier/2 initialization from [He et al]
- SGD + Momentum(0.9)
- Learning rate:0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

ResNet, 152 layers



Skip Connection (deep network can perform as good as a better shallow network)

In residual network, in forward pass current layer that has finer features is aggregated with previous layers [Ex 2+3) layers before the current layer is aggregated]

Generally during backpropagation if network is huge / contains many layers the gradient vanishes during backprop. But if we have residual connection it doesn't allow the gradient to vanish and thus relevant update is made to the weight & the network learns.

WEEK 12

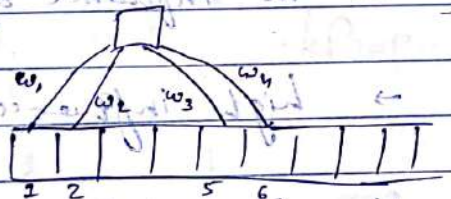
* Visualizing Patches

→ given a neuron in any layer one can always find a patch in the image which is responsible for that neuron to fire

* Visualizing Filters of a CNN

→ refer autoencoders deep learning (2) how to visualize input using weights for a neuron to fire

→ Similar to that we here use learned weights to determine the input & thus determine which patch of input is responsible for a neuron to fire



$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} \Rightarrow \boxed{w_1 | w_2 | w_3 | w_4} = w'$$

$$\boxed{x_1 | x_2 | x_5 | x_6} = \frac{w'}{\|w'\|}$$

Thus for these values of input the given neuron will fire.

* Occlusion Experiments

→ Take a patch in image & replace it by black lo
~~now~~ now try to classify

If you get correct classification that patch doesn't affect o/p has a very less impact on o/p

Similarly do it for all patches to understand important patches for classification

* Finding influence of Input Pixels using Backprop

→ Gradients with respect to input give the influence

lets say h_j is a neuron of hidden layer j

Now, if

$\frac{\partial h_j}{\partial x_i} = 0$ → no influence of x_i on h_j

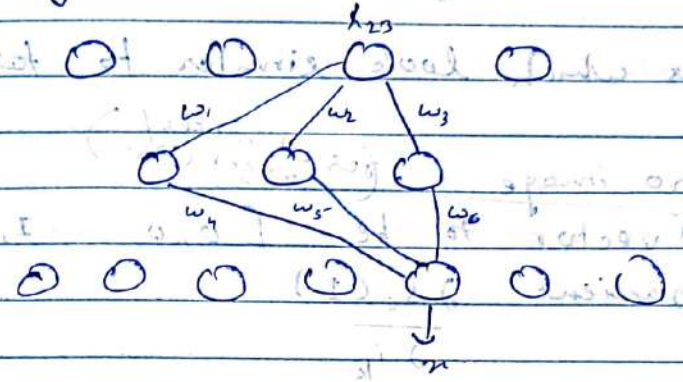
$\frac{\partial h_j}{\partial x_i}$ = large → high influence

$\frac{\partial h_j}{\partial x_i}$ = small → low influence.

So we represent images as gradients

$\frac{\partial h_j}{\partial x_{00}}$	$\frac{\partial h_j}{\partial x_{01}}$		
⋮	⋮	⋮	⋮
			$\frac{\partial h_j}{\partial x_{mn}}$

- Finding gradient wrt input



So we want to find

$$\frac{\partial h_{23}}{\partial x} = \frac{\partial h_{23}}{\partial h_{11}}\frac{\partial h_{11}}{\partial x} + \frac{\partial h_{23}}{\partial h_{12}}\frac{\partial h_{12}}{\partial x} + \frac{\partial h_{23}}{\partial h_{13}}\frac{\partial h_{13}}{\partial x}$$

$$= w_1 w_4 + w_2 w_5 + w_3 w_6$$

Now, plot, image calculate similarly for all input & plot img.

* Guided Backpropagation (Technique used visualizing the impact of image pixels on result)

1	-1	5
2	-5	-7
-3	2	4

→

1	0	5
2	0	0
0	2	4

become zero because of ReLU

} Forward pass

-2	0	-1
6	0	0
0	-1	3

←

-2	3	-1
6	+3	1
2	-1	3

} Normal Backprop

0	0	0
6	0	0
0	0	3

←

-2	3	-1
6	-3	1
2	-1	3

All negative gradients in back pass are made zero

} Guided Back Prop

* Optimization Over Images

→ Generating images which look similar to target class

(1) Start with zero image → (result vector / target class)
(2) Set the (Score) vector to be $[0, 0, \ldots -1, 0, 0]$
(3) Compute the gradient $\frac{\partial S_c(I)}{\partial i_k}$
(4) Update pixel $i_k = i_k - \eta \frac{\partial S_c(I)}{\partial i_k}$
(5) Now again do forward Pass
(6) Go to step 2

→ We can also create image such that a particular neuron of any hidden layer fires.

(1) Feed img to network
(2) Make activation such that selected neuron is one others are zero
(3) Backprop to image
(4) $i_k = i_k - \eta \frac{\partial A(I)}{\partial i_k}$, $A(I)$ is the activation of i^{th} neuron in some layer

* Creating Embeddings for Image

(used to construct an image that would give same embedding at wanted layer)

Any layer in the CNN can be considered as the embedding

Let i_{img} be the original image

We pass this image through the Network

At the fully connected layer some set of values are obtained

These values are called embedding values and entire set of values is one embedding

Now we take a random image $\hat{i}$ and try to reconstruct it such that when it is passed through network the corresponding Fully connected (FC) layer embedding is same.

loss

$$L(\hat{i}) = \| \underbrace{\phi(x)}_{\text{obtained embedding}} - \underbrace{\phi_0}_{\text{required embedding}} \|^2 + \lambda \| \phi(x) \|_6^6$$

update rule

$$\hat{i}_k = \hat{i}_k - \eta \frac{\partial L(\hat{i})}{\partial \hat{i}_k}$$

This abstractions of image can be constructed using image to image embeddings.

Similar is possible for convolution layers as well.

* Deep Dream

→ Here we make our loss function & update rule such that certain neurons can fire more.

Ishan Modi

So if h_{ij} is a neuron in convolution layer,

we want to maximize $L(I)$

i.e. $\max_I L(I)$

where

$$L(I) = h_{ij}^2$$

During backprop. update pixel of image

$$\frac{\partial L(I)}{\partial i_{mn}} = \frac{\partial L(I)}{\partial h_{ij}} * \frac{\partial h_{ij}}{\partial i_{mn}}$$

update rule

$$i_{mn} = i_{mn} + \eta \frac{\partial L(I)}{\partial i_{mn}}$$

→ Result of this will be as follows:

During training it might have developed relation that if one neuron fires for sky other neurons might fire for bird, castle, etc.

Thus when we give image of sky and increase the probability of neuron denoting sky to fire. Then it automatically takes knowledge from training data & generates birds, castle as foreground.

Ishan Modi

* Deep Art

→ Making an original image a cartoon type ie styleing the image so it looks like cartoon

(1) Content Target

$$L_{content}(\vec{p}, \vec{x}) = \sum_{ijk} (\vec{p}_{ijk} - \vec{x}_{ijk})^2$$

($\vec{p}$: pixels of embeddings of actual image; $\vec{x}$: pixels of embeddings of new image.)

New image is generated such that all image representations / embeddings generated using new image are similar to embeddings generated by actual image

(2) Style (cartoon)

According to CV (computer vision) study $V \in R^{64 \times (256 \times 256)}$ (256 × 256 → dimension of embedding)

$V^T V \in R^{64 \times 64}$ (gives style of image)

$$\therefore L_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$ } we want the style to match for all embeddings / layers

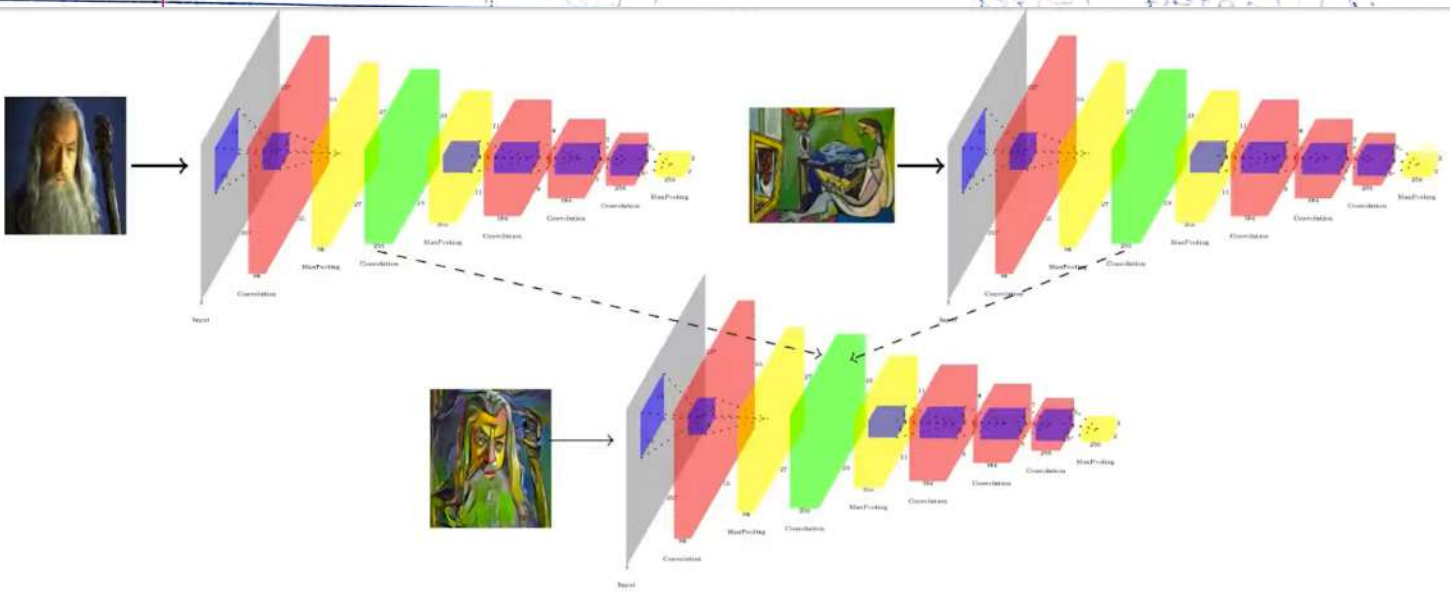
Here $E_l = \sum_{ij} (G_{ij}^l - A_{ij}^l)^2$

G_{ij}^l: style gram matrix of original image [style matrix generated from embedding obtained from original image]

A_{ij}^l: style gram matrix of generated image [style matrix generated from embedding obtained from new image]

→ Total loss function

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \cdot L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x})$$



- The total loss is given by :-

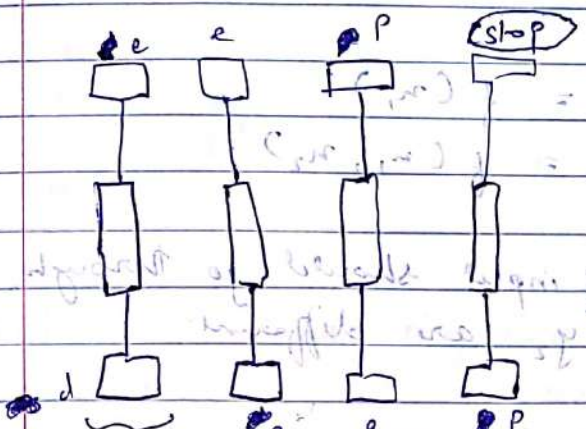
$$\mathscr{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathscr{L}_{content}(\vec{p}, \vec{x}) + \beta \mathscr{L}_{style}(\vec{a}, \vec{x})$$

* Fooling CNNs

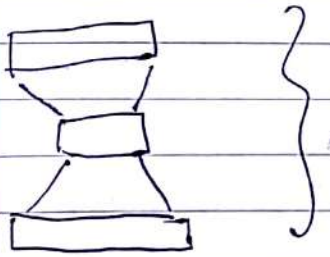
→ Change the target class value & backpropagate and make changes to original image such that you get minimum loss.

WEEK 13

* Sequence Learning Problems

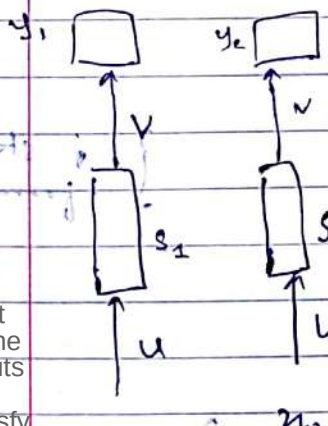


Predicting next character in a word depends on previous characters thus it is a sequence problem.



Each one of them is a feed forward NN.

* Recurrent Neural Networks (RNN)



$$s_i = \sigma(Ux_i + b)$$

$$y_i = O(Vs_i + c)$$

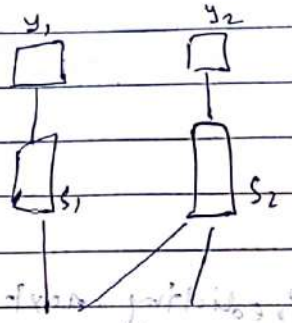
activation

i = timestamp

This does not account for the previous inputs and thus, does not satisfy the wishlist

Wishlist

- Account for dependence between inputs
- Account for variable number of inputs
- Make sure that the function executed at each time step is the same
- We will focus on each of these to arrive at a model for dealing with sequences



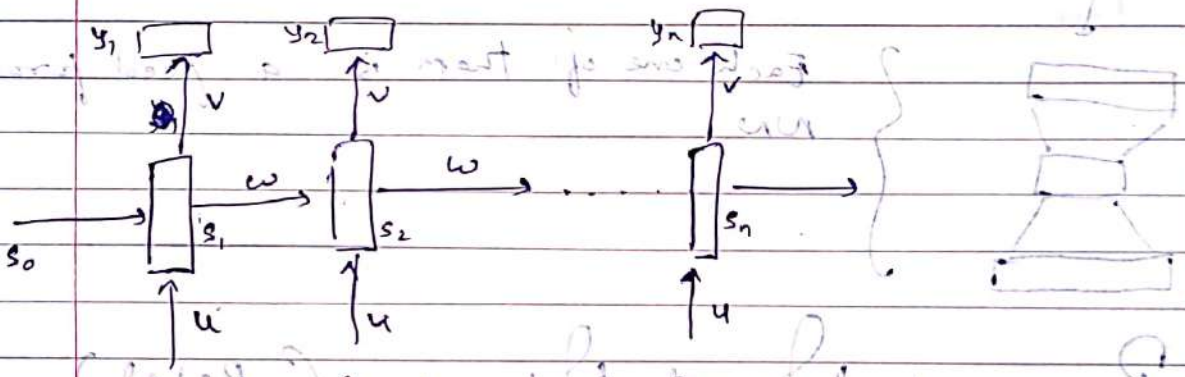
If we connect it like this

$y_1 = f(x_1)$

$y_2 = f(x_1, x_2)$

As per wish list every input should go through same function but y_1 & y_2 are different.

→ Thus we make recurrent connections



Now here

$s_i = \sigma(Ux_i + Ws_{i-1} + b)$; i starting from 1

$y_i = O(Vs_i + c)$

OR

$y_i = f(x_i, s_i, W, U, V, b, c)$

Note

s_i is state of network at timestamp i

Weights W, U, V and biases are shared across all timestamps

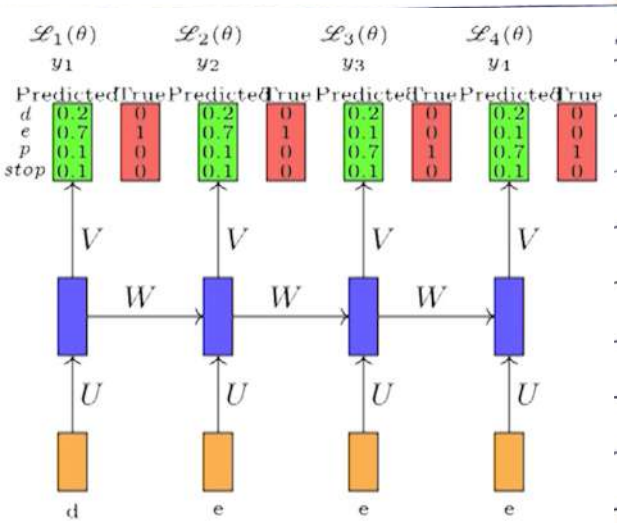
* Back Propagation Through Time

Lets say,

$x_i \in R^n$ $\quad U \in R^{n \times d}$

$s_i \in R^d$ $\quad W \in R^{d \times d}$

$y_i \in R^k$ $\quad V \in R^{d \times k}$



Here in this example we are predicting the next character in the word using RNN

- Activation at output layer = Softmax
- Loss

$L_t(\theta) = -\log(y_{tc})$ } cross entropy at each time stamps

$L(\theta) = \sum_{t=1}^{T} L_t(\theta)$ } total loss

y_{tc} = predicted probability of true character at time-stamp

T = number of timesteps...

→ Here during backprop we need to find gradients for

V, W, U

(1) Gradients for V

$$\frac{\partial L(\theta)}{\partial V} = \sum_{t=1}^{T} \frac{\partial L_t(\theta)}{\partial V}$$

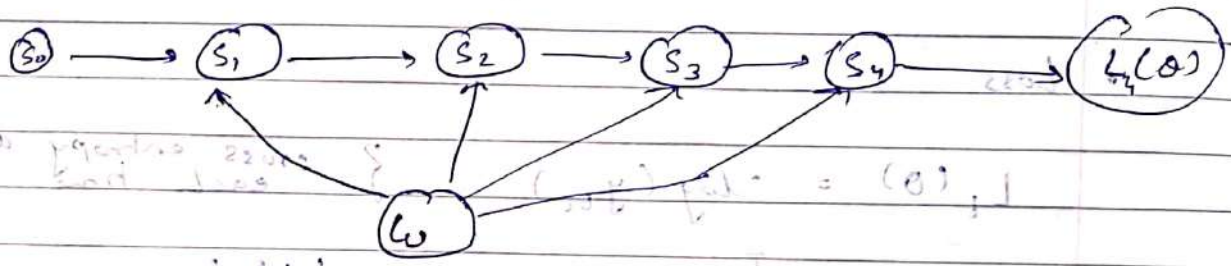
↓ matrix

} add gradient matrix obtained at every time stamp

(2) Gradient wrt W

$$\frac{\partial L(\theta)}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t(\theta)}{\partial W}$$

For the network



Now,

$$\frac{\partial L_4(\theta)}{\partial W} = \frac{\partial L_4(\theta)}{\partial S_4} \frac{\partial S_4}{\partial W}$$

refer Backprop { derivative wrt first hidden layer

Now, $S_4 = \sigma(WS_3 + b)$

Here $S_4, S_3, \ldots, S_1$ are functions of W thus we need as follows

Here we have ignored ∂ for simplicity

$$\frac{\partial S_4}{\partial w} = \underbrace{\frac{\partial^+ S_4}{\partial w}}_{\text{Explicit (has direct path to } w)} + \underbrace{\frac{\partial S_4}{\partial S_3}\frac{\partial S_3}{\partial w}}_{\text{implicit (has indirect path to } w)}$$

$$= \frac{\partial^+ S_4}{\partial w} + \frac{\partial S_4}{\partial S_3}\left[\underbrace{\frac{\partial^+ S_3}{\partial w}}_{\text{Explicit}} + \underbrace{\frac{\partial S_3}{\partial S_2}\frac{\partial S_2}{\partial w}}_{\text{implicit}}\right]$$

$$= \frac{\partial^+ S_4}{\partial w} + \frac{\partial S_4}{\partial S_3}\frac{\partial^+ S_3}{\partial w} + \frac{\partial S_4}{\partial S_3}\frac{\partial S_3}{\partial S_2}\frac{\partial^+ S_2}{\partial w} + \frac{\partial S_4}{\partial S_3}\frac{\partial S_3}{\partial S_2}\frac{\partial S_2}{\partial S_1}\frac{\partial^+ S_1}{\partial w}$$

$$= \frac{\partial S_4}{\partial S_4}\frac{\partial^+ S_4}{\partial w} + \frac{\partial S_4}{\partial S_3}\frac{\partial^+ S_3}{\partial w} + \frac{\partial S_4}{\partial S_2}\frac{\partial^+ S_2}{\partial w} + \frac{\partial S_4}{\partial S_1}\frac{\partial^+ S_1}{\partial w}$$

$$\frac{\partial S_4}{\partial w} = \sum_{k=1}^{4}\frac{\partial S_4}{\partial S_k}\frac{\partial^+ S_k}{\partial w}$$

$$\therefore \frac{\partial L_4(\theta)}{\partial w} = \frac{\partial L_4(\theta)}{\partial S_4}\left[\sum_{k=1}^{4}\frac{\partial S_4}{\partial S_k}\frac{\partial^+ S_k}{\partial w}\right]$$

General

$$\frac{\partial L_t(\theta)}{\partial w} = \frac{\partial L_t(\theta)}{\partial S_t}\left[\sum_{k=1}^{t}\frac{\partial S_t}{\partial S_k}\frac{\partial^+ S_k}{\partial w}\right]$$

(3) Gradient wrt u

use chain rule through v & w and find u

Ishan Modi

* Exploding & Vanishing Gradients

→ Now, we find

$$\frac{\partial s_t}{\partial s_k}$$

$$\frac{\partial s_t}{\partial s_k} = \frac{\partial s_t}{\partial s_{t-1}} \frac{\partial s_{t-1}}{\partial s_{t-2}} \cdots \cdots \frac{\partial s_{k+1}}{\partial s_k} = \prod_{j=k}^{t-1} \frac{\partial s_{j+1}}{\partial s_j}$$

→ Thus

$\frac{\partial s_j}{\partial s_{j-1}}$ is to be found

let,

$$a_j = W s_{j-1} + b + U x_j$$
$$s_j = \sigma(a_j)$$

$$a_j = [a_{j1}, a_{j2}, a_{j3} \ldots a_{jd}] \quad - (1)$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}) \ldots \sigma(a_{jd})] \quad - (2)$$

→ Now

$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

↓ will be diagonal matrix × W

Ishan Modi

from (1) & (2)

$$\frac{\partial s_j}{\partial a_j} = \begin{bmatrix} \frac{\partial s_{j1}}{\partial a_{j1}} & \frac{\partial s_{j1}}{\partial a_{j2}} & \cdots \\ \vdots & & \\ 0 & & \end{bmatrix} \times W$$

(× multiplied)

$$= \begin{bmatrix} \sigma'(a_{j1}) & 0 & \cdots \\ 0 & \sigma'(a_{j2}) & \\ 0 & & \sigma'(a_{jd}) \end{bmatrix} \times W$$

$$= \text{diag}(\sigma'(a_j)) \times W$$

Taking magnitude

$$\left\| \frac{\partial s_j}{\partial s_{j-1}} \right\| = \| \text{diag}(\sigma'(a_j)) \, W \|$$

$$\leq \| \text{diag}(\sigma'(a_j)) \| \, \|W\|$$ (by property of norm: $\|ab\| \leq \|a\| \|b\|$)

- $\sigma'(a_j) \leq \frac{1}{4} = \gamma$ (if σ is logistic)

  $\sigma(a_j)(1 - \sigma(a_j)) \leq \frac{1}{4}$

  ↓ max value taken by derivative is $\frac{1}{4}$

- $\sigma'(a_j) \leq 1 = \gamma$ (if σ is tanh)

Ishan Modi

$\leq \gamma \|w\|$

$\leq \gamma\lambda$

Now,

$$\left\| \frac{\partial s_t}{\partial s_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}} \right\|$$

$$\leq \prod_{j=k+1}^{t} \gamma\lambda$$

$$\leq (\gamma\lambda)^{t-k}$$

→ $\gamma\lambda < 1$ vanishing gradient

→ $\gamma\lambda > 1$ exploding gradient

* Important

$$\frac{\partial L_t(\theta)}{\partial w} = \frac{\partial L_t(\theta)}{\partial s_t} \sum_{k=1}^{t} \frac{\partial s_t}{\partial s_k} \frac{\partial^+ s_k}{\partial w}$$

$d \times d$ ✓ $\quad d \times d$ ✓ $\quad d \times d$ ✓ $\quad d \times d \times d$

fetched values are known to us

→ lets compute any one element of tensor $dx \, dx \, d$

$\frac{\partial^+ s_{kp}}{\partial w_{qr}}$: $(p,q,r)^{th}$ element of tensor

Note: diff wrt w are 0

$a_k = w s_{k-1} + b + U x_k$

$s_k = \sigma(a_k)$

$\therefore a_k = w s_{k-1}$

$$\begin{bmatrix} a_{k1} \\ a_{k2} \\ \vdots \\ a_{kn} \end{bmatrix} = \begin{bmatrix} w_{11} & \cdots \\ w_{p1} & w_{p2} \cdots w_{pd} \\ w_{d1} & \cdots \cdots w_{dd} \end{bmatrix} \begin{bmatrix} s_{k-1,1} \\ s_{k-1,2} \\ \vdots \\ s_{k-1,p} \\ \vdots \\ s_{k-1,d} \end{bmatrix}$$

$$\therefore a_{kp} = \sum_{i=1}^{d} w_{pi} \, s_{k-1,i} \quad \text{---} \quad (1)$$

$s_{kp} = \sigma(a_{kp})$

$$\frac{\partial s_{kp}}{\partial w_{qr}} = \frac{\partial s_{kp}}{\partial a_{kp}} \frac{\partial a_{kp}}{\partial w_{qr}}$$

$$= \sigma'(a_{kp}) \frac{\partial a_{kp}}{\partial w_{qr}}$$

Ishan Modi

$$\frac{\partial a_{kp}}{\partial w_{qr}} = \frac{\partial \sum_{i=1}^{d} w_{pi} \cdot s_{k-1,i}}{\partial w_{qr}}$$

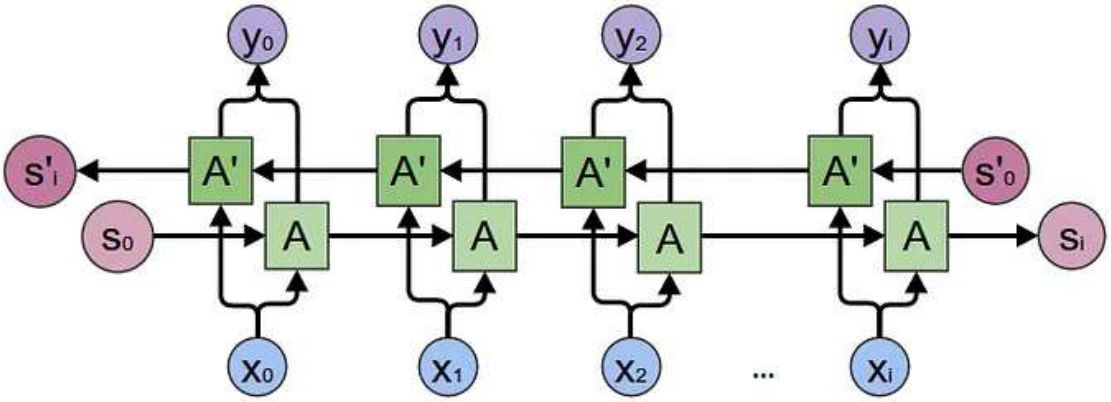
$$= s_{k-1,i} \quad \text{if } p = q \text{ & } i = r$$

$$= 0 \quad \text{otherwise}$$

$$\therefore \frac{\partial s_{kp}}{\partial w_{qr}} = \sigma'(a_{kp})\, s_{k-1,r} \quad \text{if } p = q \text{ & } i = r$$

$$= 0 \quad \text{otherwise}$$

* Bi-Directional RNNs



WEEK 14

* Selective Read, Write, Forget

→ In a RNN the problem of vanishing and exploding gradient occurs during backprop.

But, during forward prop also we keep on updating the state vector. This vector is also of finite dimension so by reaching at end of RNN it will also forget info from 1st state

During Forward pass & Back pass info is tend to be forgotten.

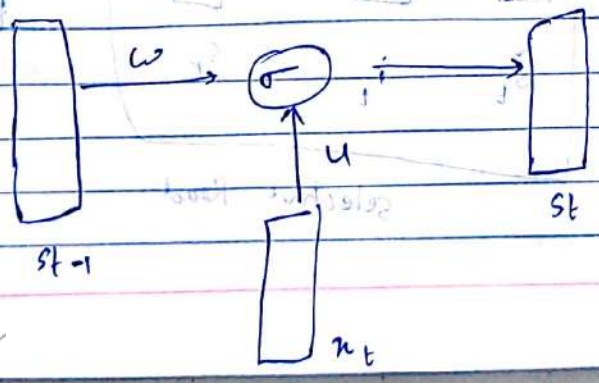
→ Selective read, write, forget using white-board analogy

* LSTM (long-short term memory)

→ lets consider task of sentiment analysis

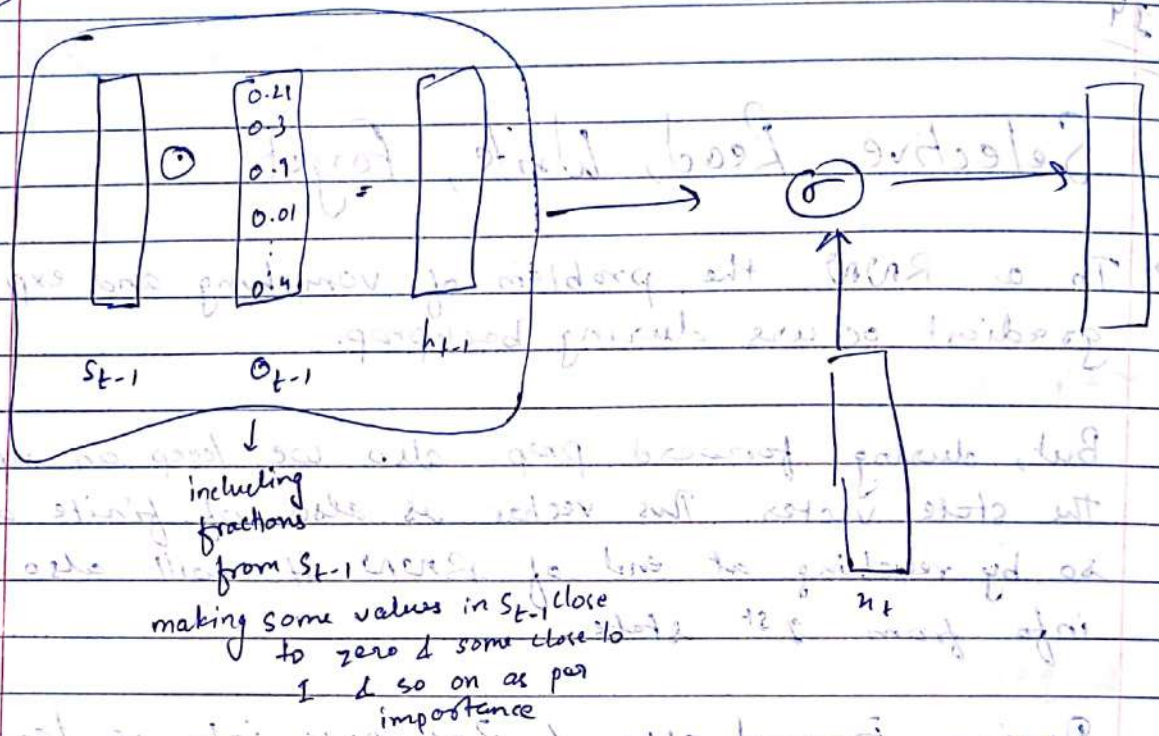
- Forget - the information like stopwords
- Selective Read - the info added by previous sentiment bearing words
- Selective Write - new information from current word to state.

→



Here if we want to selectively write values of S_{t-1}

Selective Write



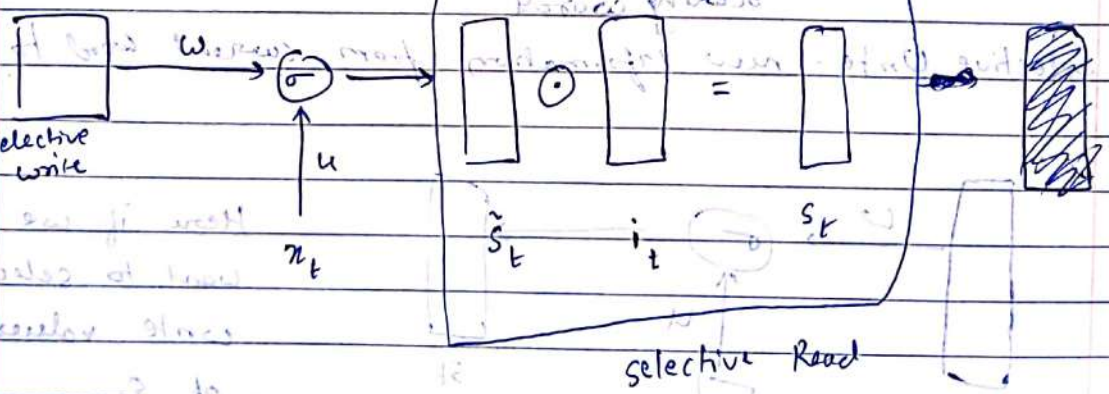
Now, we need to learn O_{t-1} (Output Gate)

$$O_{t-1} = \sigma(W_o h_{t-2} + U_o x_{t-1} + b_o)$$

→ (U, V, W) are parameters of RNN

U_o, b_o, W_o are parameters of RNN (O_{t-1})

Selective Read



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

$$s_t = \tilde{s}_t \odot i_t$$

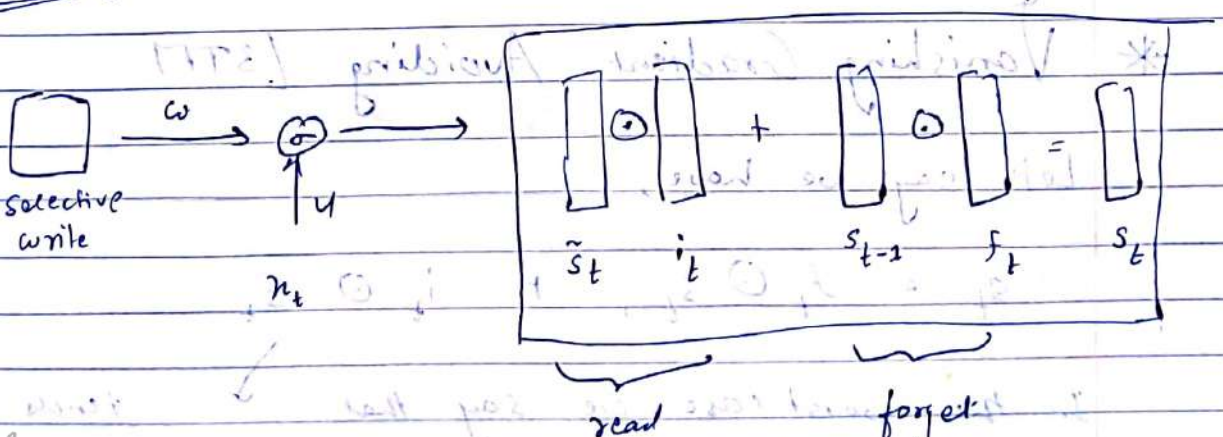
$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

Note → W, U, b & W_i, U_i, b_i are different parameters

Till now

- Previous state → s_{t-1}
- Output gate → $o_{t-1} = \sigma(W_o h_{t-2} + U_o x_{t-1} + b_o)$
- Selective Write → $h_{t-1} = o_{t-1} \odot \sigma(s_{t-1})$
- Current (temp) state → $\tilde{s}_t = \sigma(W h_{t-1} + U x_t + b)$
- Input gate → $i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$
- Selective Read → $i_t \odot \tilde{s}_t$

Selective Forget



Ishan Modi

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

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

* GRU (Gated Recurrent Unit)

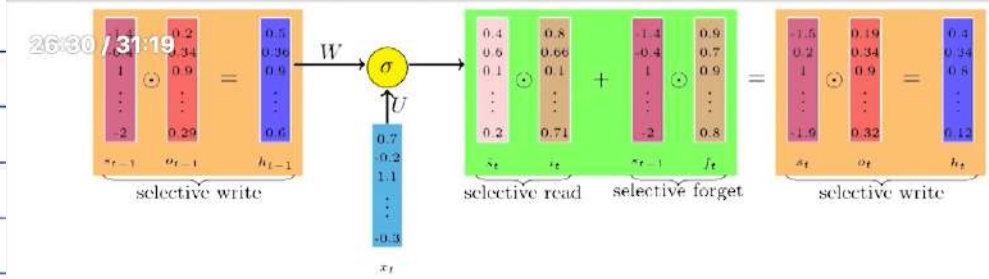
→ Gates

- $o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$
- $i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$

→ States

- $\tilde{s}_t = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$
- $s_t = (1 - i_t) \odot s_{t-1} + i_t \odot \tilde{s}_t$

LSTM (Diagram - 1)



- We now have the full set of equations for LSTMs
- The green box together with the selective write operations following it, show all the computations which happen at timestep t

Gates:

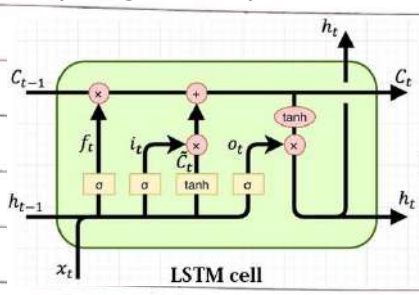
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

States:

$$\tilde{s}_t = \sigma(W h_{t-1} + U x_t + b)$$
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$
$$h_t = o_t \odot \sigma(s_t) \text{ and } rnn_{out} = h_t$$

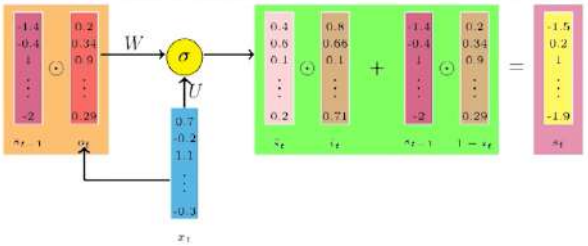
Ishan Modi

LSTM (Diagram - 2)



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

GRU (Diagram - 1)



The full set of equations for GRUs

Gates:

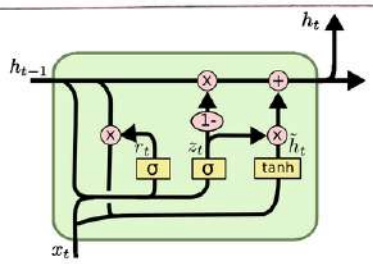
$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$
$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

States:

$$\tilde{s}_t = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$$
$$s_t = (1 - i_t) \odot s_{t-1} + i_t \odot \tilde{s}_t$$

- No explicit forget gate (the forget gate and input gates are tied)
- The gates depend directly on s_{t-1} and not the intermediate h_{t-1} as in the case of LSTMs

GRU (Diagram - 2)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

* Vanishing Gradient Avoiding LSTM

Lets say we have,

$$S_t = f_t \odot S_{t-1} + i_t \odot \tilde{S}_t$$

In worst case we say that $\downarrow$ tends to 0 or vanishes

Ishan Modi

Thus we have

$$S_t = f_t \odot S_{t-1}$$

$$\boxed{\frac{\partial S_t}{\partial S_{t-1}} = f_t}$$

as we go back pass $f_t \times f_{t-1} \times f_{t-2} \times \dots \dots f_1$

$$= (f_t)^T$$

Here also during back pass f_t vanishes

But during forward pass

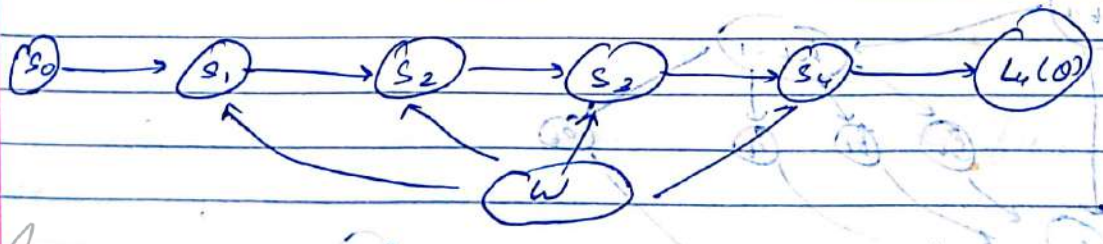
f_t information goes from 1 state to next

$\therefore (f_t)^T$ for forward.

Conclusion

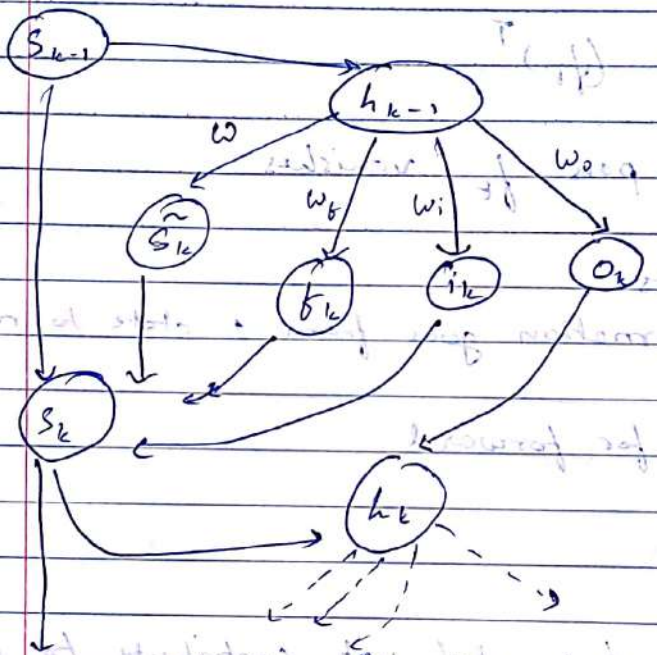
→ If the state at time $t-1$ did not contribute to state at time t (ie, if $\|f_t\| \to 0$ and $\|o_{t-1}\| \to 0$) then during backprop the gradients flowing into s_{t-1} will vanish.

* Proof (Vanishing doesn't Occur)



→ In general, the gradient of $L_t(O)$ wrt θ_i vanishes when the gradients flowing through each and every path from $L_t(O)$ to θ_i vanishes.

→ The gradient of $L_t(O)$ wrt θ_i explodes when the gradient flowing through at least one path explodes.



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

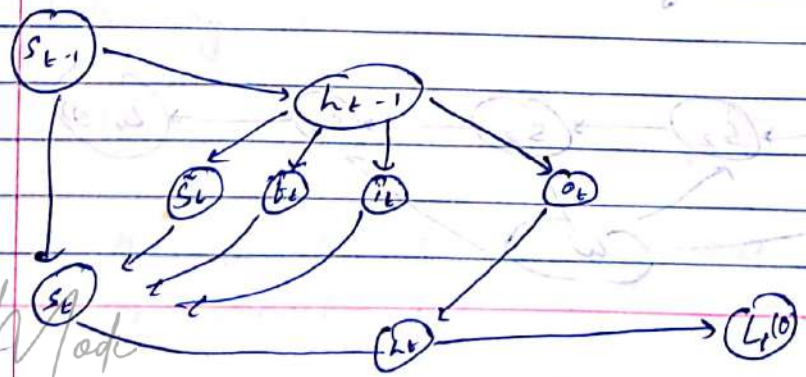
$$i_t = \sigma(w_i h_{t-1} + v_i x_t + b_i)$$

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

$$\tilde{s}_t = \sigma(w h_{t-1} + v x_t + b)$$

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

$$h_t = o_t \odot \sigma(s_t)$$



Path $L_t(\theta) \to h_t \to s_t \to s_{t-1} \dots s_{k+1} \to s_k$

→ Now lets say w_f causes error in output & we want to backprop gradient to w_f

Then we need to prove that there exist at least one path from $L_t(\theta)$ to s_k that allows gradient to flow so that it doesn't vanish

$$\frac{\partial L_t(\theta)}{\partial s_k} = \frac{\partial L_t(\theta)}{\partial h_t} \frac{\partial h_t}{\partial s_t} \left(\frac{\partial s_t}{\partial s_{t-1}} \cdots \frac{\partial s_{k+1}}{\partial s_k} \right)$$

$\frac{\partial L_t(\theta)}{\partial h_t}$ → can be found

(1) $\frac{\partial h_t}{\partial s_t}$ (2) $\frac{\partial s_t}{\partial s_{t-1}}$

~~$\frac{\partial}{\partial s_t} [f_t \odot s_t + (i_t \odot \tilde{s}_t)]$~~ → ignoring in worst case

(1) $\frac{\partial h_t}{\partial s_t} = \frac{\partial}{\partial s_t} [o_t \odot \sigma(s_t)] = \text{Diagonal}\left(o_t \odot \sigma'(s_t)\right)$ matrix

(2) $\frac{\partial s_t}{\partial s_{t-1}} = \frac{\partial}{\partial s_{t-1}} [f_t \odot s_{t-1} + (i_t \odot \tilde{s}_t)]$ → ignoring worst case

$= D(f_t)$

↓ diagonal matrix

$$\left(\frac{\partial s_t}{\partial s_{t-1}} \cdots \frac{\partial s_{k+1}}{\partial s_k} \right) = D(f_t \odot \dots \odot f_{k+1})$$

Ishan Modi

$$= L_t'(h_t) \cdot D(o_t \odot \sigma'(s_t)) \cdot D\left(\bigodot_{i=k+1}^{t} f_i\right)$$

Thus gradient vanish in backprop only if it vanished in forward prop. It vanish only when it needs to this proved

* Proof (Exploding can be stopped)

path $= (L_t, L_t(o) \to h_t \to o_t \to h_{t-1} \dots h_k \to o_k \to h_{k-1})$

$$= \frac{\partial L_t(o)}{\partial h_t} \left(\frac{\partial h_t}{\partial o_t} \frac{\partial o_t}{\partial h_{t-1}}\right) \dots \left(\frac{\partial h_k}{\partial o_k} \frac{\partial o_k}{\partial h_{k-1}}\right)$$

$$= L_t'(h_t) \cdot (D(\sigma(s_t) \odot o_t') \cdot W_0) \dots (D(\sigma(s_k) \odot o_k') \cdot W_0)$$

$$\|t_1\| \le \|L_t'(h_t)\| (\|k\| \|W_0\|)^{t-k+1}$$

Depending on norm of $\|W_0\|$ gradient may explode

Thus use (Gradient Clipping) . If norm exceeds certain value bring it down to a threshold.

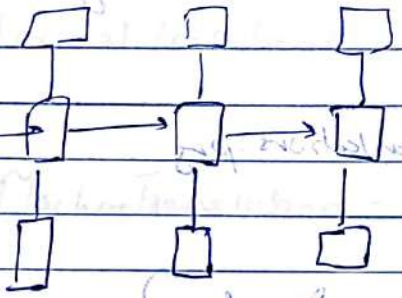
Ishan Modi

WEEK 15

* Introduction To Encoder Decoder

→ Predicting next word in sequence

$$y^* = \text{argmax}\ P(y_t = j \mid y_1, y_2, \ldots, y_{t-1})$$



at each stage in RNN, it will predict a probability distribution & word with max prob will be chosen.

Using RNN we can compute

$$P(y_t = j \mid \underbrace{y_1^{t-1}}_{(y_1, y_2 \ldots y_{t-1})}) = \underbrace{\text{softmax}(Vs_t + c)_j}_{\text{j vector of size equal to vocab.}} \rightarrow j^{th} \text{ element of vector}$$

$j \in V$, V is vocabulary

$$P(y_t = j \mid \underbrace{y_1^{t-1}}_{(y_1, y_2, \ldots y_{t-1})}) = P(y_t = j \mid \underbrace{s_t}_{\text{All the information of } (y_1, y_2 \ldots y_{t-1}) \text{ is contained in } s_t})$$

Ishan Modi

We will now use the following representations of RNNs, GRUs and LSTMs

$$s_t = \sigma(U\,x_t + W s_{t-1} + b)$$

$$\tilde{s}_t = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$$
$$s_t = i_t \odot s_{t-1} + (1 - i_t) \odot \tilde{s}_t$$

$$\tilde{s}_t = \sigma(W\,h_{t-1} + U x_t + b)$$
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$
$$h_t = o_t \odot \sigma(s_t)$$

$$s_t = \text{RNN}(s_{t-1}, x_t)$$

$$s_t = \text{GRU}(s_{t-1}, x_t)$$

$$h_t, s_t = \text{LSTM}(h_{t-1}, s_{t-1}, x_t)$$

→ Image Captioning (Encoder - Decoder)

$P(y_t \mid y_1^{t-1}) = P(y_t \mid s_t)$

∴ Here we have image as well

∴ $P(y_t \mid s_t, I)$ → cannot take entire image we need representation of image

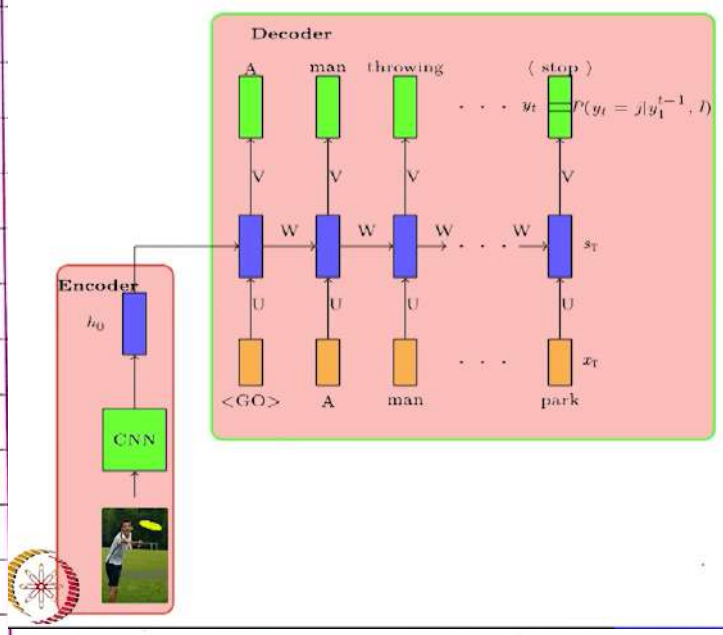
$P(y_t \mid s_t, fc_7(I))$ ← fully connected layer in VGG.

This becomes s_0,

∴ $s_0 = fc_7(I)$ now we feed it to RNN for caption generation

Ishan Modi

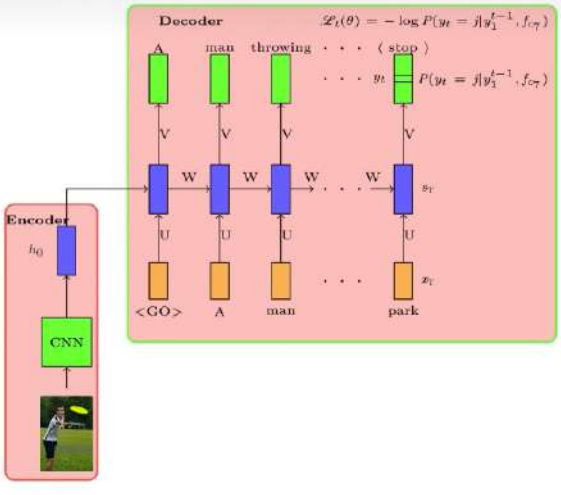
We can Also feed FC7(I) to all states,



- Let us look at the full architecture
- A CNN is first used to **encode** the image
- A RNN is then used to decode (generate) a sentence from the encoding
- This is a typical **encoder decoder architecture**

✳ Application of Encoder-Decoder

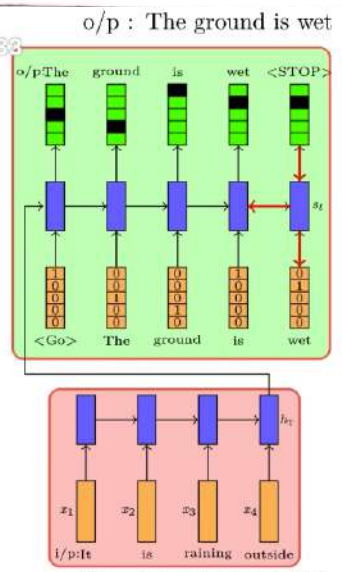
---> Image Captioning



- **Task:** Image captioning
- **Data:** $\{x_i = image_i,\ y_i = caption_i\}_{i=1}^{N}$
- **Model:**
  - **Encoder:**
    $$s_0 = CNN(x_i)$$
  - **Decoder:**
    $$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
    $$P(y_t|y_1^{t-1}, I) = softmax(Vs_t + b)$$
- **Parameters:** $U_{dec},\ V,\ W_{dec},\ W_{conv}, b$
- **Loss:**
  $$\mathscr{L}(\theta) = \sum_{t=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, I)$$
- **Algorithm:** Gradient descent with back-propagation

Ishan Modi

---> Text Entailment



- **Task:** Textual entailment
- **Data:** $\{x_i = premise_i,\ y_i = hypothesis_i\}_{i=1}^{N}$
- **Model (Option 1):**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

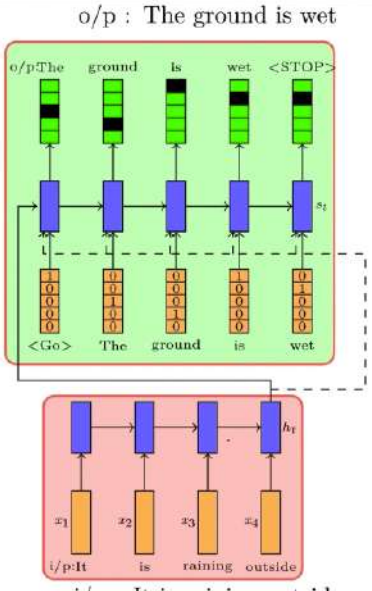
  - **Decoder:**

$$s_0 = h_T \quad (T \text{ is length of input})$$
$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
$$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$

- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc}, b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_i(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_i|y_1^{t-1}, x)$$

- **Algorithm:** Gradient descent with back-propagation



- **Task:** Textual entailment
- **Data:** $\{x_i = premise_i,\ y_i = hypothesis_i\}_{i=1}^{N}$
- **Model (Option 2):**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

  - **Decoder:**

$$s_0 = h_T \quad (T \text{ is length of input})$$
$$s_t = RNN(s_{t-1}, [h_T, e(\hat{y}_{t-1})])$$
$$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$

Concatenate

- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc}, b$
- **Loss:**

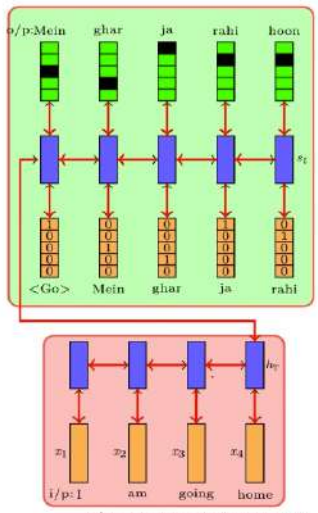
$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, x)$$

- **Algorithm:** Gradient descent with back-propagation

Ishan Modi

---> Machine Translation

o/p : Mein ghar ja rahi hoon



i/p : I am going home

- **Task:** Machine translation
- **Data:** $\{x_i = source_i,\ y_i = target_i\}_{i=1}^{N}$
- **Model (Option 1):**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

  - **Decoder:**

$$s_0 = h_T \quad (T \text{ is length of input})$$
$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
$$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$

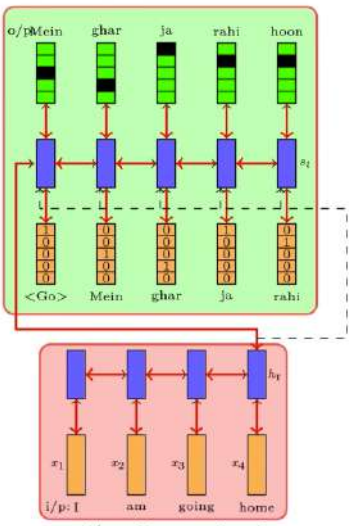
- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc}, b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, x)$$

- **Algorithm:** Gradient descent with back-propagation

✳

o/p : Mein ghar ja rahi hoon



i/p : I am going home

- **Task:** Machine translation
- **Data:** $\{x_i = source_i,\ y_i = target_i\}_{i=1}^{N}$
- **Model (Option 2):**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

  - **Decoder:**

$$s_0 = h_T \quad (T \text{ is length of input})$$
$$s_t = RNN(s_{t-1}, [h_T, e(\hat{y}_{t-1})])$$
$$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$

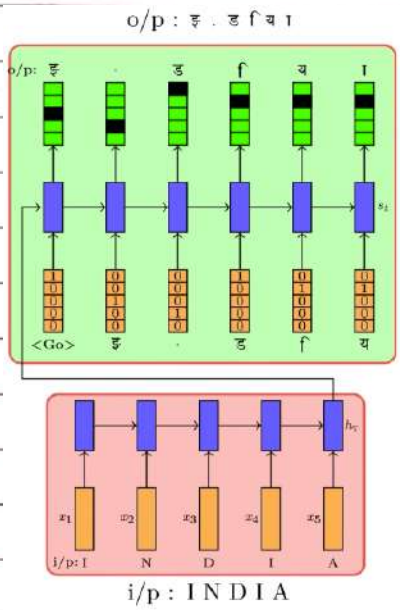
- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc}, b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, x)$$

- **Algorithm:** Gradient descent with back-propagation

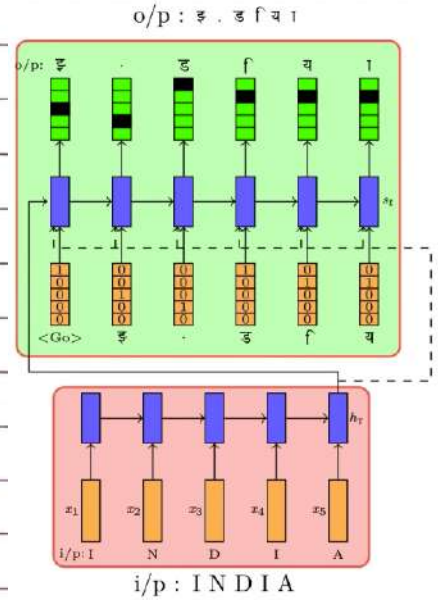
Ishan Modi

---> Transliteration



- **Task:** Transliteration
- **Data:** $\{x_i = srcword_i,\ y_i = tgtword_i\}_{i=1}^{N}$
- **Model (Option 1):**
  - **Encoder:**
    $$h_t = RNN(h_{t-1}, x_{it})$$
  - **Decoder:**
    $$s_0 = h_T \quad (T \text{ is length of input})$$
    $$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
    $$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$
- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc},\ b$
- **Loss:**
  $$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, x)$$
- **Algorithm:** Gradient descent with back-propagation

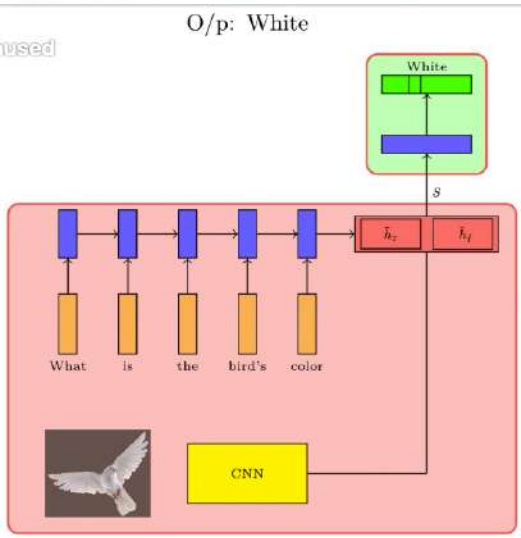
✳



- **Task:** Transliteration
- **Data:** $\{x_i = srcword_i,\ y_i = tgtword_i\}_{i=1}^{N}$
- **Model (Option 2):**
  - **Encoder:**
    $$h_t = RNN(h_{t-1}, x_{it})$$
  - **Decoder:**
    $$s_0 = h_T \quad (T \text{ is length of input})$$
    $$s_t = RNN(s_{t-1}, [e(\hat{y}_{t-1}), h_T])$$
    $$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$
- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc},\ b$
- **Loss:**
  $$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, x)$$
- **Algorithm:** Gradient descent with back-propagation

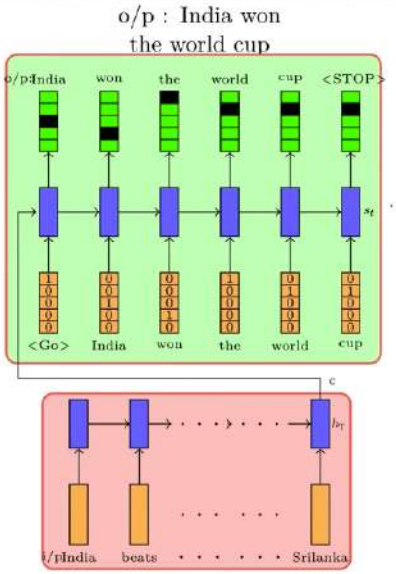
Ishan Modi

---> Image Q&A



- **Task:** Image Question Answeing
- **Data:** $\{x_i = \{I, q\}_i,\ y_i = Answer_i\}_{i=1}^{N}$
- **Model:**
  - **Encoder:**
    $\hat{h}_I = CNN(I),\ \tilde{h}_t = RNN(\tilde{h}_{t-1}, q_{it})$
    $s = [\tilde{h}_T; \hat{h}_I]$
  - **Decoder:**
    $$P(y|q, I) = softmax(Vs + b)$$
- **Parameters:** $V,\ b,\ U_q,\ W_q,\ W_{conv}, b$

✳> Document Summarization

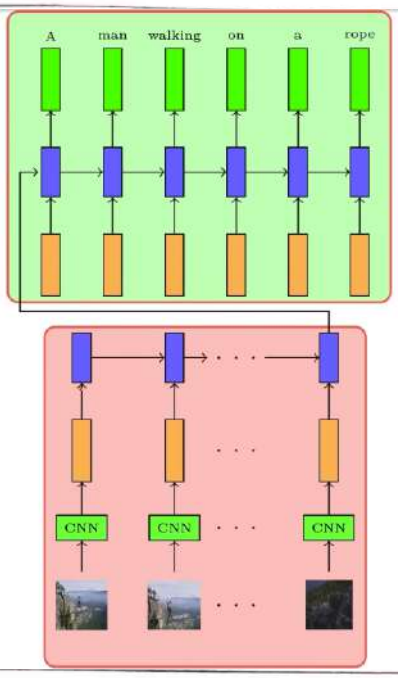


i/p : India beats Srilanka to win ICC WC 2011.
Dhoni and Gambhir's half centuries help beat SL

- **Task:** Document Summarization
- **Data:** $\{x_i = Document_i,\ y_i = Summary_i\}_{i=1}^{N}$
- **Model:**
  - **Encoder:**
    $$h_t = RNN(h_{t-1}, x_{it})$$
  - **Decoder:**
    $$s_0 = h_T$$
    $$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
    $$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$
- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc}, b$
- **Loss:**
  $$\mathcal{L}(\theta) = \sum_{i=1}^{T} \mathcal{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, x)$$
- **Algorithm:** Gradient descent with backpropagation

Ishan Modi

---> Video Captioning



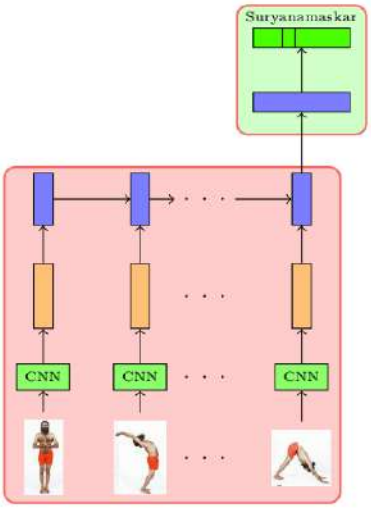
- **Task:** Video Captioning
- **Data:** $\{x_i = video_i,\ y_i = desc_i\}_{i=1}^{N}$
- **Model:**
  - **Encoder:**
    $$h_t = RNN(h_{t-1}, CNN(x_{it}))$$
  - **Decoder:**
    $$s_0 = h_T$$
    $$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
    $$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$
- **Parameters:** U_{dec}, W_{dec}, V, b, W_{conv}, U_{enc}, W_{enc}, b
- **Loss:**
  $$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, x)$$
- **Algorithm:** Gradient descent with back-propagation

27/59 27/59

*

---> Video Classification

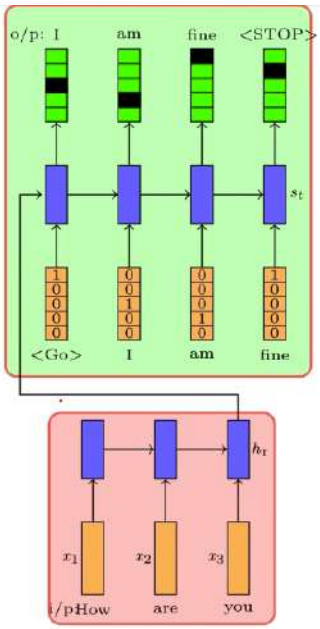
o/p: Surya Namaskar



- **Task:** Video Classification
- **Data:** $\{x_i = Video_i,\ y_i = Activity_i\}_{i=1}^{N}$
- **Model:**
  - **Encoder:**
    $$h_t = RNN(h_{t-1}, CNN(x_{it}))$$
  - **Decoder:**
    $$s = h_T$$
    $$P(y|I) = softmax(Vs + b)$$
- **Parameters:** V, b, W_{conv}, U_{enc}, W_{enc}, b
- **Loss:**
  $$\mathscr{L}(\theta) = -\log P(y = \ell|Video)$$
- **Algorithm:** Gradient descent with backpropagation

Ishan Modi

---> Dialog



i/p: How are you

- **Task:** Dialog
- **Data:** $\{x_i = Utterance_i,\ y_i = Response_i\}_{i=1}^{N}$
- **Model:**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

  - **Decoder:**

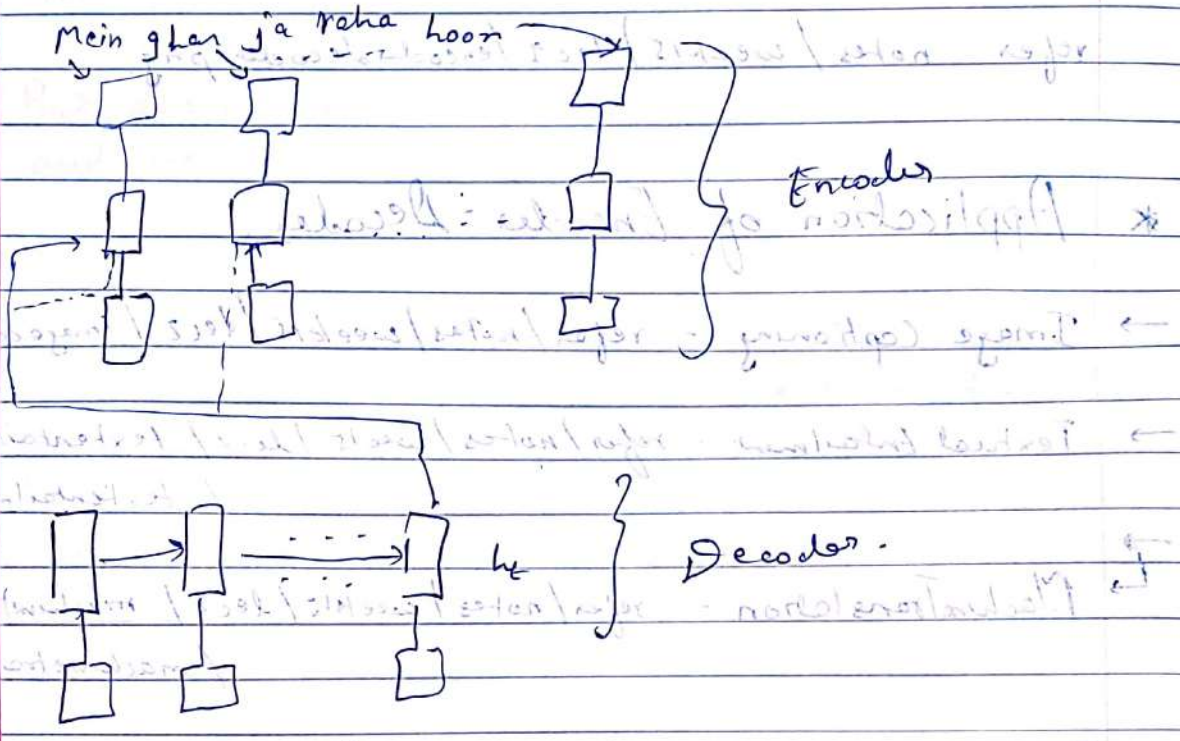
$$s_0 = h_T \quad (T \text{ is length of input})$$
$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
$$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$

- **Parameters:** U_{dec}, V, W_{dec}, U_{enc}, W_{enc}, b
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P($$

- **Algorithm:** Gradient descent w gation

* Attention Mechanism



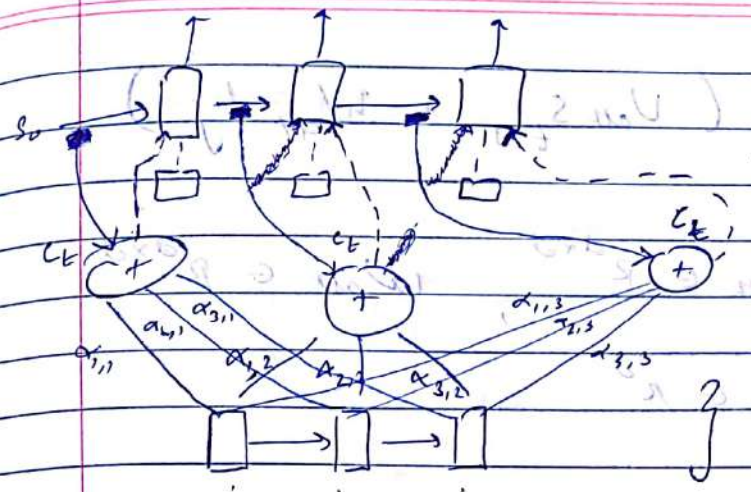
I am going home

So here we are passing h_t as S_0 for another network, this h_t contains all the information ie (I am going home) but we can observe that:

I → Mein
home → ghar
am → hoon
going → ja raha

These are words that are hugely related

So the intuition is why not pass this relationship instead of h_t



Here $\alpha_{1,1}$ $\alpha_{1,2}$ $\alpha_{1,3}$ | $\alpha_{2,1}$ $\alpha_{2,2}$ $\alpha_{2,3}$ | $\alpha_{3,1}$ $\alpha_{3,2}$ $\alpha_{3,3}$

are all parameters. which are to be learnt

→ Now,

$$e_{jt} = f_{ATT}(s_{t-1}, h_j)$$

(s_{t-1}: decoder state; h_j: encoder state.)

j^{th} word in encoder

t^{th} time stamp in decoder

$$\alpha_{jt} = \frac{\exp(e_{jt})}{\sum_{j=1}^{M} \exp(e_{jt})}$$

This α_{jt} is probability measure of how important is j^{th} word with respect to t^{th} time stamp.

Ishan Modi

$$e_{jt} = V_{att}^T \tanh\left(U_{att} s_{t-1} + W_{att} h_j\right)$$

scalar

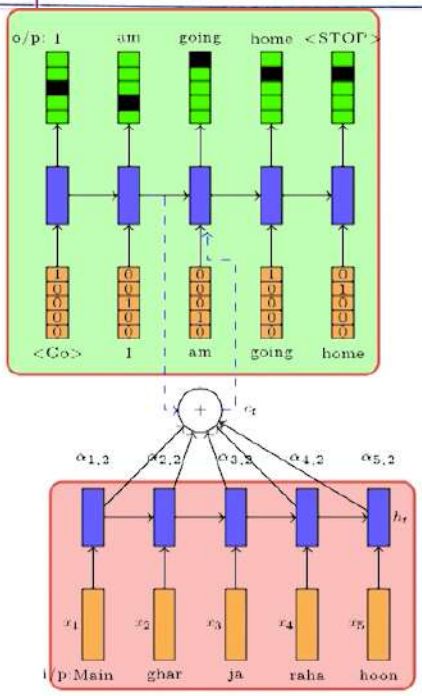
$V_{att}^T \in R^d$, $U_{att} \in R^{d \times d}$, $W_{att} \in R^{d \times d}$

$s_{t-1} \in R^d$, $h_j \in R^d$

This thing works because it is better modeling choice.

Also,

$$c_t = \sum_{j=1}^{T} \alpha_{jt} h_j$$ } for encoder.



- **Task:** Machine Translation
- **Data:** $\{x_i = source_i,\ y_i = target_i\}_{i=1}^N$
- **Encoder:**

$$h_t = RNN(h_{t-1}, x_t)$$
$$s_0 = h_T$$

- **Decoder:**

$$e_{jt} = V_{attn}^T \tanh(U_{attn} h_j + W_{attn} s_t)$$
$$\alpha_{jt} = softmax(e_{jt})$$
$$c_t = \sum_{j=1}^{T} \alpha_{jt} h_j$$
$$s_t = RNN(s_{t-1}, [e(\hat{y}_{t-1}), c_t])$$
$$\ell_t = softmax(Vs_t + b)$$

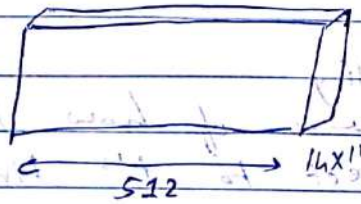
- **Parameters:** U_{dec}, V, W_{dec}, U_{enc}, W_{enc}, b, U_{attn}, V_{attn}
- **Loss** and **Algorithm** remains same

Ishan Modi

* Attention Over Images

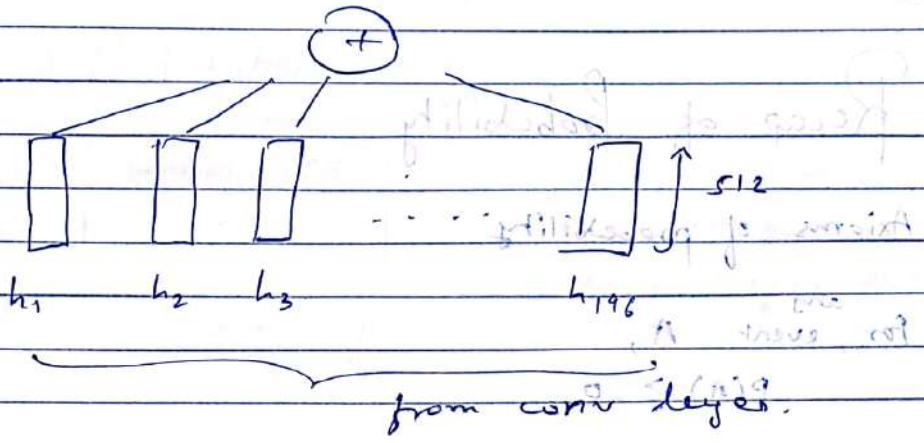
Unlike RNN, where each state gives word CNN for image gives FC_7 which doesn't give us which entry is for which pixel in image

Thus we use the convolution layers before FC_7



Each of pixel from 14X14 represents some portion in image.

14 X 14 = 196



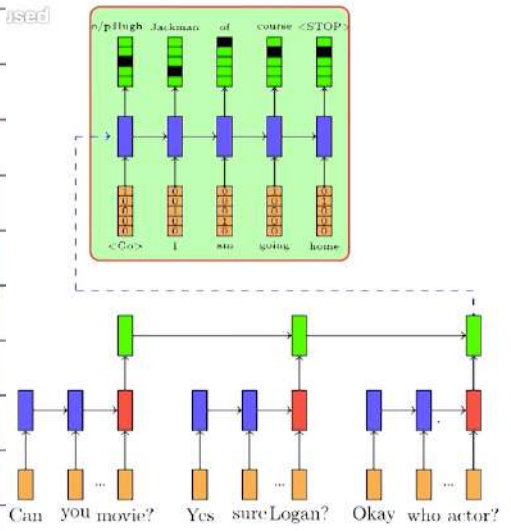
$$\alpha_{ij,t} = f_{ATT}(s_{t-1}, h_j)$$

$$f_{ATT} = v^T \tanh(U^* s_{t-1} + W h_j + b)$$

✳ Hierarchical Attention

---> Understanding hierarchical RNN structure

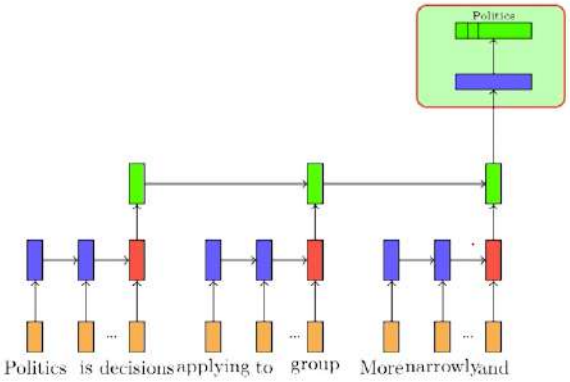
Dialog heirarchical RNN



- We could think of a two level hierarchical RNN encoder
- The first level RNN operates on the sequence of words in each utterance and gives us a representation
- We now have a sequence of utterance representations (red vectors in the image)
- We can now have another RNN which encodes this sequence and gives a single representation for the sequences of utterances
- The decoder can then produce an output sequence conditioned on this utterance

Document heirarchical RNN

Politics is the process of making decisions applying to all members of each group. More narrowly, it refers to achieving and ...



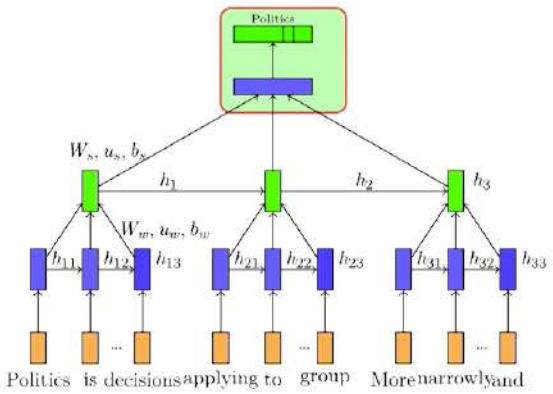
- **Data:** $\{Document_i, class_i\}_{i=1}^{N}$
- **Word level (1) encoder:**
  $h_{ij}^1 = RNN(h_{ij-1}^1, w_{ij})$
  $s_i = h_{iT_i}^1$ *[T is length of sentence i]*
- **Sentence level (2) encoder:**
  $h_i^2 = RNN(h_{i-1}^2, s_i)$
  $s = h_K^2$ *[K is number of sentences]*
- **Decoder:**
  $P(y|document) = softmax(Vs + b)$
- **Params:** W_{enc}^1, U_{enc}^1, W_{enc}^2, U_{enc}^2, V, b
- **Loss:** Cross Entropy
- **Algorithm:** Gradient Descent with backpropagation

Ishan Modi

Now we use attention mechanism in these heirarchical module

Encoder (heirarchical attention)

Politics is the process of making decisions applying to all members of each group. More narrowly, it refers to achieving and ...



- **Data:** $\{Docuement_i, class_i\}_{i=1}^{N}$
- **Word level (1) encoder:**

$$h_{ij} = RNN(h_{ij-1}, w_{ij})$$
$$u_{ij} = tanh(W_w h_{ij} + b_w)$$
$$\alpha_{ij} = \frac{exp(u_{ij}^T u_w)}{\sum_t exp(u_{it}^T u_w)}$$
$$s_i = \sum_j \alpha_{ij} h_{ij}$$

- **Sentence level (2) encoder:**

$$h_i = RNN(h_{i-1}, s_i)$$
$$u_i = tanh(W_s h_i + b_s)$$
$$\alpha_i = \frac{exp(u_i^T u_s)}{\sum_i exp(u_i^T u_s)}$$
$$s = \sum_i \alpha_i h_i$$

Decoder (heirarchical attention)

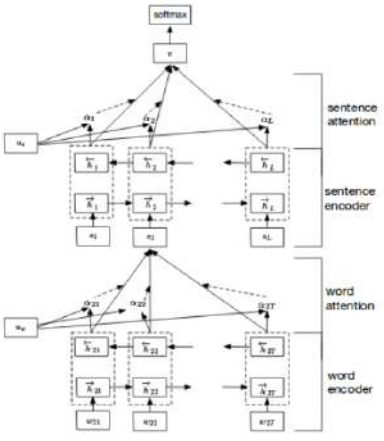


Figure: Hierarchical Attention Network [Yang et al..]

- **Decoder:**

$$P(y|document) = softmax(Vs + b)$$

- **Parameters:** $W_w, W_s, V, b_w, b_s, b, u_w, u_s$
- **Loss:** cross entropy
- **Algorithm:** Gradient Descent and backpropagation

Ishan Modi