Jinpoint abstraction

DL

6/03/23

Matrix Calculus:-

$Y = f(X) \rightarrow X, Y$ can be scalars, vectors & matrices.

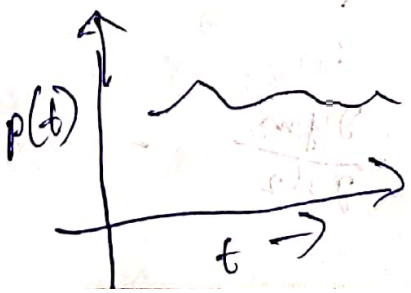
Let y be a scalar function of 3 independent variables.

i.e $y = f(x, y, z)$.

$$\nabla f = \frac{\partial f}{\partial x}\hat{x} + \frac{\partial f}{\partial y}\hat{y} + \frac{\partial f}{\partial z}\hat{z} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \end{bmatrix}^T$$

Derivatives with vectors.

$\vec{y} \in \mathbb{R}^m$ w.r.t scalar x, $\vec{y} = [y_1, y_2 \cdots y_m]^T$



$$p(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$$

$\frac{\partial p(t)}{\partial t}$ = Velocity

$$\frac{\partial \vec{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_n}{\partial x} \end{bmatrix}$$

2) Diff. scalar by a vector $y \in R, x \in R^n$.

Let, $x = [x_1, x_2 \ldots x_n]^T$

$$\frac{\partial y}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

3) Both x and y are vectors $y \in R^m, x \in R^n$

$y = [y_1, y_2 \ldots y_m]^T$

$x = [x_1, x_2 \ldots x_n]^T$

Jacobian Matrix

$$\frac{\partial \vec{y}}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}_{m \times n}$$

Derivatives with Matrix :-

Case 1 :-

Matrix by scalar

$$\frac{\partial Y}{\partial x} = \begin{bmatrix} \frac{\partial y_{11}}{\partial x} & \frac{\partial y_{12}}{\partial x} & \cdots & \frac{\partial y_{1n}}{\partial x} \\ \frac{\partial y_{21}}{\partial x} & \frac{\partial y_{22}}{\partial x} & \cdots & \frac{\partial y_{2n}}{\partial x} \\ \vdots & & & \\ \frac{\partial y_{m1}}{\partial x} & \frac{\partial y_{m2}}{\partial x} & \cdots & \frac{\partial y_{mn}}{\partial x} \end{bmatrix}$$

Case 2 :-

$X \in \mathbb{R}^{m \times n}$, y is a scalar

$$\frac{\partial y}{\partial X} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{m1}} \\ \frac{\partial y}{\partial x_{12}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{m2}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial y}{\partial x_{1n}} & \frac{\partial y}{\partial x_{2n}} & \cdots & \frac{\partial y}{\partial x_{mn}} \end{bmatrix}_{m \times n}$$

1) $y = Ax$

$$\frac{\partial y}{\partial z} = A \frac{\partial x}{\partial z}$$

2) $\alpha = y^T x = [x_1 y_1 + x_2 y_2 + \cdots + x_n y_n]$

$$\frac{\partial \alpha}{\partial z} = \left(x_1 \frac{\partial y_1}{\partial z} + y_1 \frac{\partial x_1}{\partial z} \right) + \left(x_2 \frac{\partial y_2}{\partial z} + y_2 \frac{\partial x_2}{\partial z} \right) + \cdots$$

$$= \left(x_1 \frac{\partial y_1}{\partial z} + x_2 \frac{\partial y_2}{\partial z} + x_3 \frac{\partial y_3}{\partial z} + \cdots \right) + \left(y_1 \frac{\partial x_1}{\partial z} + y_2 \frac{\partial x_2}{\partial z} + \cdots \right)$$

$$= x^T \frac{\partial y}{\partial z} + y^T \frac{\partial x}{\partial z}$$

Gradient Descent

Loss function $J\left(f(x^{(i)}, \theta), y\right)$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla_\theta J$$

$$L = \frac{1}{N} \sum_{i=1}^{N} J\left(f(x^{(i)}, \theta), y^{(i)}\right)$$

Stochastic gradient Descent :-

Select one instance randomly.

$$L = J\left(f(x^{(r)}, \theta), y^{(r)}\right)$$

Mini Batch $\rightarrow \{ x^{(1)} \dots x^{(m)} \}$

$$L = \frac{1}{m} \sum_{i=1}^{m} J\left(f(x^{(i)}, \theta), y^{(i)}\right)$$

$\rightarrow \epsilon_k = (1-\alpha)\epsilon_0 + \alpha \epsilon_\tau$

$$\alpha = \frac{k}{\tau}$$

Deep learning

9/3/23

Gradient Descent

Sufficient condition on the learning rate for the convergence of the gradient descent or SGD.

$$\rightarrow \sum_{k=1}^{\infty} \epsilon_k = \infty \quad \& \quad \sum_{k=1}^{\infty} \epsilon^2 < \infty$$

Common practice :- use a decaying value of ϵ_k.

$$\boxed{\epsilon_k = (1-\alpha)\,\epsilon_0 + \alpha\,\epsilon_\tau} \qquad \alpha = \frac{k}{\tau}$$

$$\epsilon_\tau < c\,\epsilon_0 \quad (\text{Typically } 0.01\,\epsilon_0)$$

Consider a scalar function,

$$\phi = \phi(x, y, z)$$

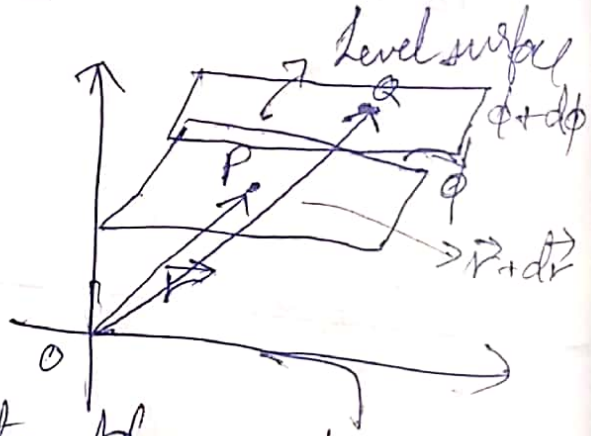
$$\nabla = \hat{i}\frac{\partial}{\partial x} + \hat{j}\frac{\partial}{\partial y} + \hat{k}\frac{\partial}{\partial z}$$

$$\nabla\phi = \hat{i}\frac{\partial\phi}{\partial x} + \hat{j}\frac{\partial\phi}{\partial y} + \hat{k}\frac{\partial\phi}{\partial z}$$

Let, $\vec{OP} = \vec{r}$

$\vec{OQ} = \vec{r} + d\vec{r}$

Then shifting $\vec{r}$ to $\vec{r} + d\vec{r}$



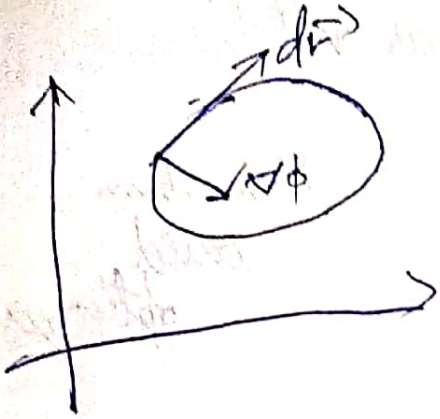
Shifting the level surface from ϕ to $\phi + d\phi$.

$$d\phi = \frac{\partial\phi}{\partial x}dx + \frac{\partial\phi}{\partial y}\cdot dy + \frac{\partial\phi}{\partial z}\cdot dz$$

$$= \left(\hat{i}\frac{\partial\phi}{\partial x} + \hat{j}\frac{\partial\phi}{\partial y} + \hat{k}\frac{\partial\phi}{\partial z}\right)\cdot\left(\hat{i}\cdot dx + \hat{j}\cdot dy + \hat{k}\cdot dz\right)$$

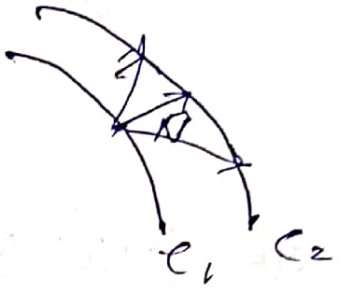
$$= \nabla\phi \cdot d\vec{r}$$

If we change 'P' to 'Q' on the same surface ϕ. Then $d\phi = 0$. Then $\nabla\phi \cdot d\vec{r} = 0$



$\nabla\phi$ points in a direction normal to the level surface

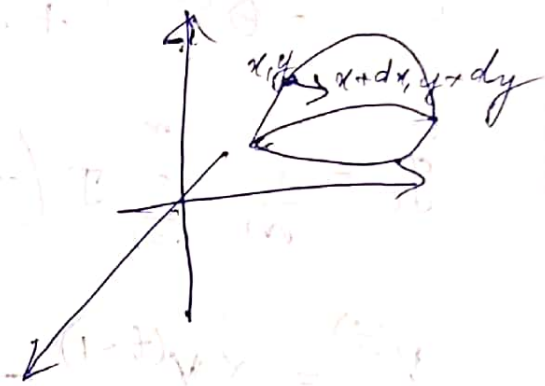
$$\vec{\nabla}\phi = |\vec{\nabla}\phi| \cdot \hat{N}$$



$$d\phi = \vec{\nabla}\phi \cdot d\vec{r} = |\vec{\nabla}\phi| \cdot |d\vec{r}| \cdot \cos\theta$$

$$= |\vec{\nabla}\phi| \, d\vec{N}$$

$$\frac{d\phi}{d\vec{N}} = |\vec{\nabla}\phi|$$



Challenges of Gradient Descent:-

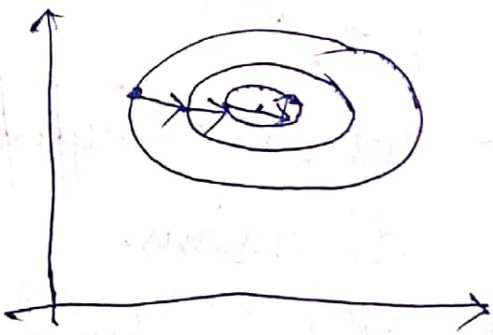
→ Choice of the proper learning rate or step size.

→ Predefined learning rate for all directions.

→ Problem of saddle points.

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta g_t$$

10/03/2023



→ We assume a momentum force given as $v^{(t)}$

$$v^{(t)} \leftarrow \alpha v^{(t-1)} - \eta g_t$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + v^{(t)}$$

Momentum based optimizer

$\theta^{(0)} \leftarrow$ Randomly initialized parameters.

$v^{(0)} = 0.$

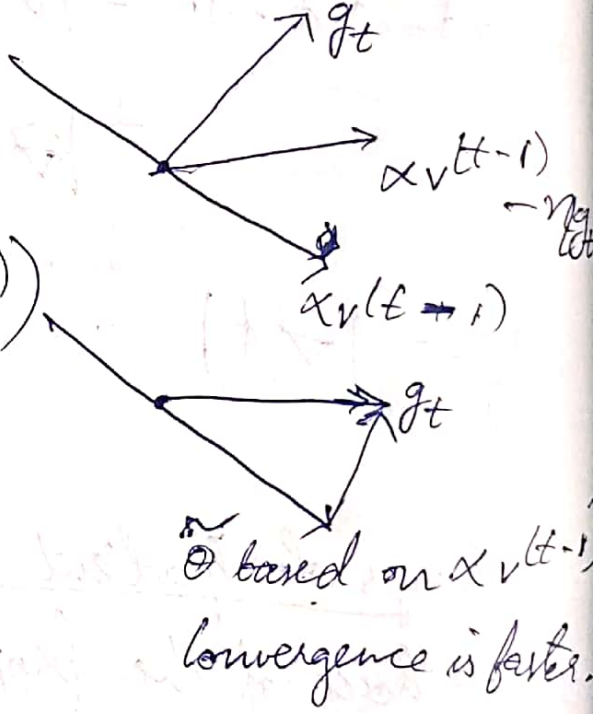
→ Nesterov Acceleration of Gradient :-

$$\tilde{\theta}^{(t)} = \theta^{(t-1)} + \alpha v^{(t-1)}$$

$$\tilde{g}_t = \frac{1}{m} \sum_{i=1}^{m} \nabla \left(f(x^{(i)}, \tilde{\theta}), y^{(i)} \right)$$

$$v^{(t)} = \alpha v^{(t-1)} - \eta \tilde{g}_t$$

$$\theta^{(t+1)} = \theta^{(t)} + v^{(t)}$$



Problem with momentum based optimizer :-

→ The hyperparameters α & η, determine the learning rate.

→ Uses same hyperparameters values across all dimensions.

→ Better if tuned according to gradient.

Soln :- Adaptive hyperparameter settings..
- Adagrad. (Technique)

→ Adagrad :-

$$g_t = \frac{1}{m} \sum \nabla_\theta J\left(f(x, \theta^{(t)}), y\right)$$

$$r_t = \sum_{\tau=1}^{t} g_\tau \circ g_\tau$$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta\, g_t}{\sqrt{\epsilon I + r_t}}$$

$$g_t = \begin{bmatrix} \frac{g^1}{\sqrt{\epsilon + r_1}} \\ \frac{g}{\sqrt{\epsilon + r_2}} \\ \vdots \end{bmatrix}$$

RMS Prop :-
↳ Discard the history from the extreme past.

$$r_{t+1} = \beta r_t + (1-\beta) s_t \qquad \boxed{r_0 = 0}$$

$$s_t = \sum_{\tau=1}^{t} g_\tau \circ g_\tau \qquad r_1 = (1-\beta) s_0$$

$$r_2 = \beta(1-\beta) s_0 + (1-\beta) s_1$$

$$r_3 = \beta^2(1-\beta) s_0 + \beta(1-\beta) s_1 + (1-\beta) s_2$$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{\sqrt{\epsilon I + r_t}} \cdot g_t$$

This creates a bias based on value of β.

$$\hat{s}_t = \frac{s_t}{1-\beta} \qquad \hat{r}_t = \frac{r_t}{1-\beta}$$

Adam (Adaptive Mom

$$s_t = \beta_1 s_{t-1} + (1-\beta_1) g_t \qquad \hat{s}_t = \frac{s_t}{1-\beta_1}$$

$$r_t = \beta_2 r_{t-1} + (1-\beta_2) \cdot g_t \circ g_t \qquad \hat{r}_t = \frac{r_t}{1-\beta_2}$$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{\sqrt{\epsilon I + \hat{r}_t}} \cdot \hat{s}_t$$

Deep Learning

16/03/23

→ Batch Normalization

internal covariate shift

happens when the distribution of the input features, vary across the mini batches.

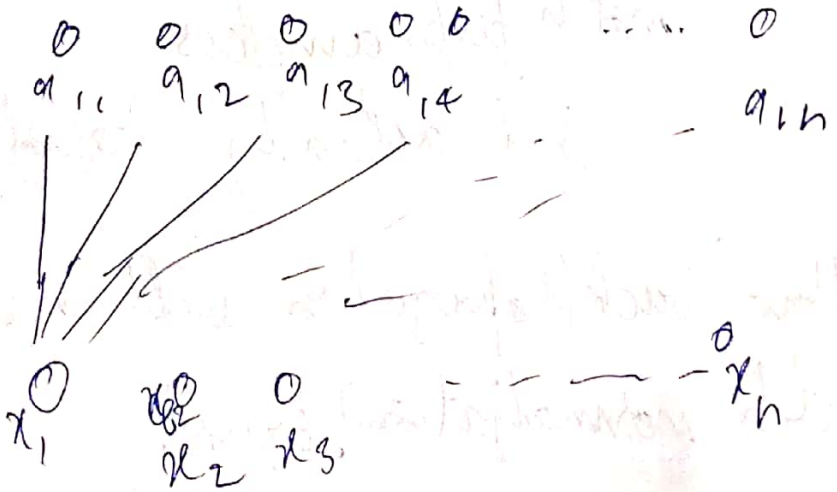
$f_2((f_1(x_i, \theta_1), \theta_2))$

$f_2(u, \theta_2)$

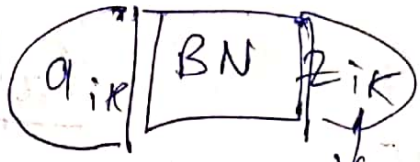
Sol:- It would help if the pre-activations at each layer is normalized.

↳ many ways, they proposed gaussian normalization.

→ It is answered by standardizing the pre-activation at each unit.



$$\hat{a}_{ik} = \frac{a_{ik} - E[a_{ik}]}{SE[a_{ik}]}$$



activation fnc. (ReLu, ... etc)

some scaling and transformation

$X_1, X_2, \ldots, X_m$

↓ no. of minibatches.

→ Once batch normalization is done, re-normalization required (as scaling and transformation had been done).

$$y^{(k)} = \gamma^{(k)} a_{ik} + \beta^{(k)}$$

doing rescaling at last layer

not hyperparameters but actually learnt.

- HW → How backpropagation will happen for batch normalization? eqns?

↳ . what if we do it at each layer?

As :- no effect of normalization would be visible.

⇒ CNN

(i) images can be very large, so feed forward networks can't handle.

(ii) images have higher dimensional ~~relations~~ relations in pixels, spatial dependence.

(iii) transformations
localization

Convolution serves better in this case. still a lot.

20/03/23

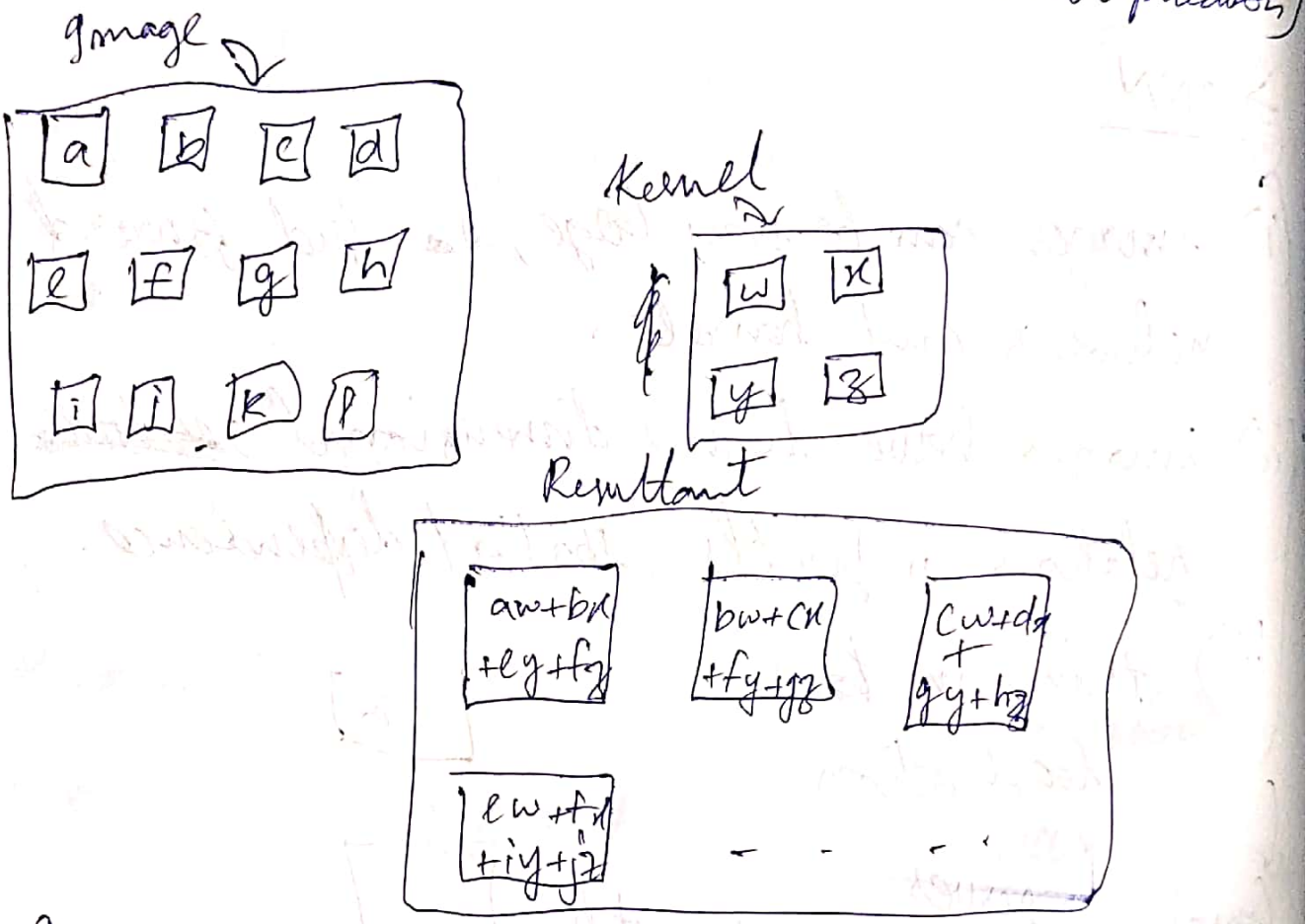
CNN properties

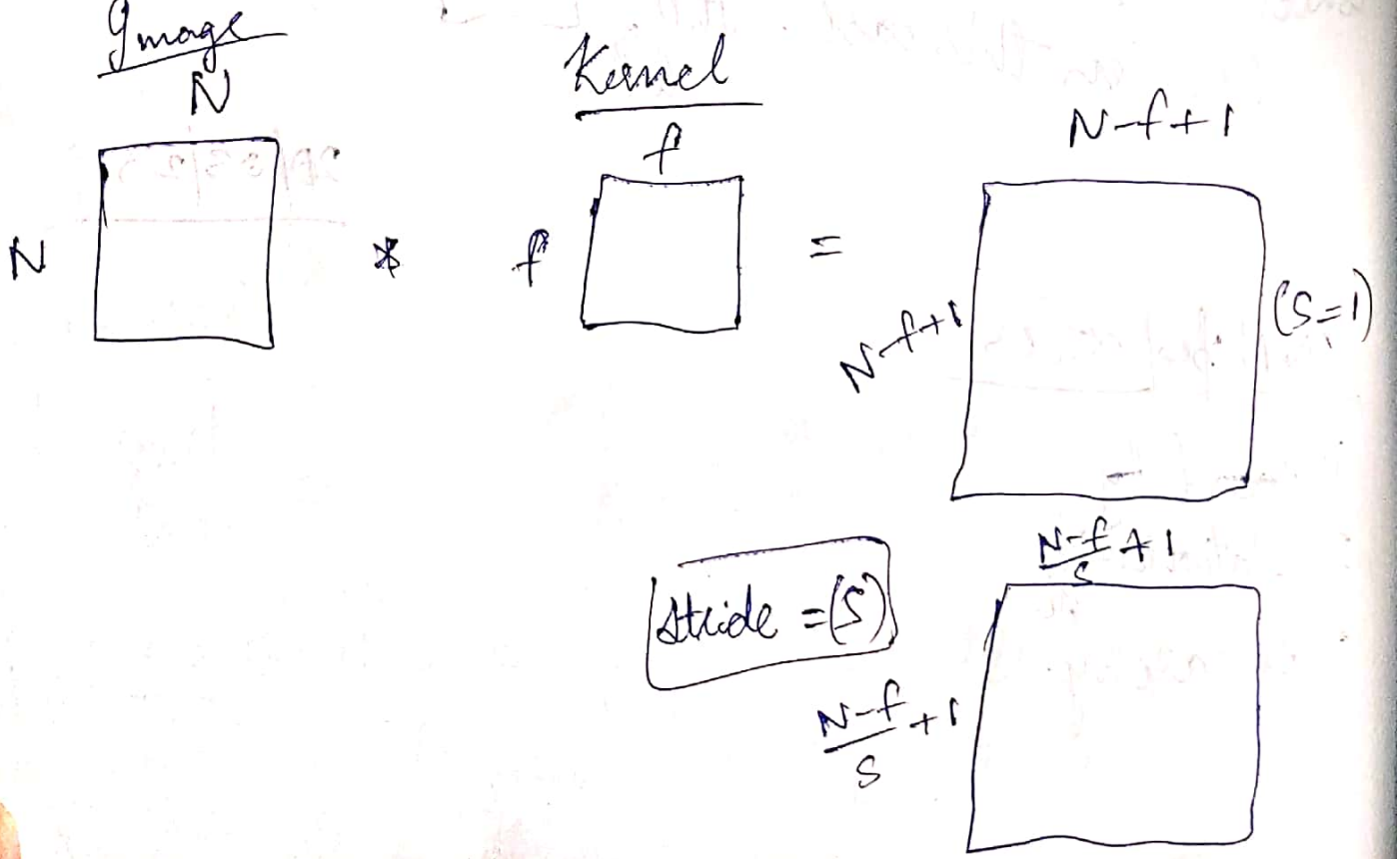
1) locality
2) Stationarity
3) Hierarchy

CNN tasks

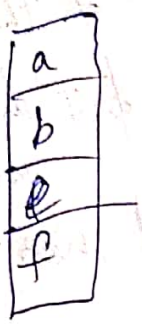
① Convolution layer (→ Kernel (filters) & Predefined size matrix)

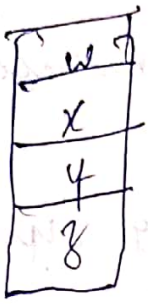
(Element-wise multiplication)





Stride is the length by which the kernel is shifted.

$I =$ 

$F =$ 

$$aw + bx + ey + fz$$

$$= I^T \cdot F = I \cdot F$$

→ Advantages:-

* Sparse interaction
* Parameter sharing

The no. of inputs in the lower layer, which affects an output unit in the higher layer is called the 'receptive field' of the output unit.

CNN is also known as DNN, as the loss in receptive field is covered up by increasing the no. of layers in the network.

Also, the parameters are shared (which captures locality.)

Invariance :- (Output shouldn't change much).

eg:- Translational Invariance, scaling invariance, Rotational Invariance.

$f(g(I)) = f(I)$ (Invariance) i.e. the order of the layers but shouldn't matter).

$f(g(I)) = g(f(I))$ (equivariance)

CNN layers are neither fully invariant nor fully equivariant.

24/03/23

Deep Learning

Alexnet Architecture

Input :- 227 × 227 × 3

Conv 1 :- 96 filters of dimension 11 × 11 × 3, stride = 4

$$\frac{N-f}{s} + 1 = \cancel{55}$$

$$\frac{227 - 11}{4} + 1$$

$$= 55 \times 55 \times 96$$

(Output size)

Parameters

$$(11 \times 11 \times 3) \times 96 \simeq 35\text{K}$$

Max-pool - 3×3 filters of stride 2

$$\frac{(55 - 3)}{2} + 1 = 27 \times 27 \times 96$$

Normalization

Conv-2 :- 256 filters of dimension 5 × 5 × 96.

stride = 1. (Padding = 2)

$(27 + 4 - 5) + 1 = 27 \times 27 \times 256$

$(27 + 05)$ ~~= 12 × 12 × 256~~

Parameters :-

$(5 \times 5 \times 96) \times 256$

Max-Pool-2

3×3 filters of stride = 2 $\frac{(27-3)}{2} + 1 = 13 \times 13 \times 256$

Normalization-2

Conv-3

384 filters 3×3, stride 1, pad 1.

$(13+2-3)+1 = 13 \times 13 \times 384$

(output size)

Parameters:-

$(3 \times 3 \times 256) \times 384$

Conv-4

384 filters 3×3, strides = 1, pad = 1.

Output size

$13 \times 3 \times 384$

Parameters:-

$(3 \times 3 \times 384) \times 384$

Conv-5:- 256 filters. 3×3, stride=1, pad=1

13×13×256 (Output size)

Parameters:-

(3×3×384) × 256

Maxpool-3 :- 3×3 filters, stride=2

$\frac{(13-3)}{2} + 1 = 6 \times 6 \times 256$

FC-1 :-	4096 neurons	(Output size) 4096	(Parameters) 9216×4096
FC-2 :-	4096 neurons	(Output size) 4096	(Parameters) 4096×4096
FC-3 :-	1000 neurons		(Parameters) 4096×1000

25/03/23

Problems:-

→ It was a bit shallow, one of the first architectures to use deep neural network.

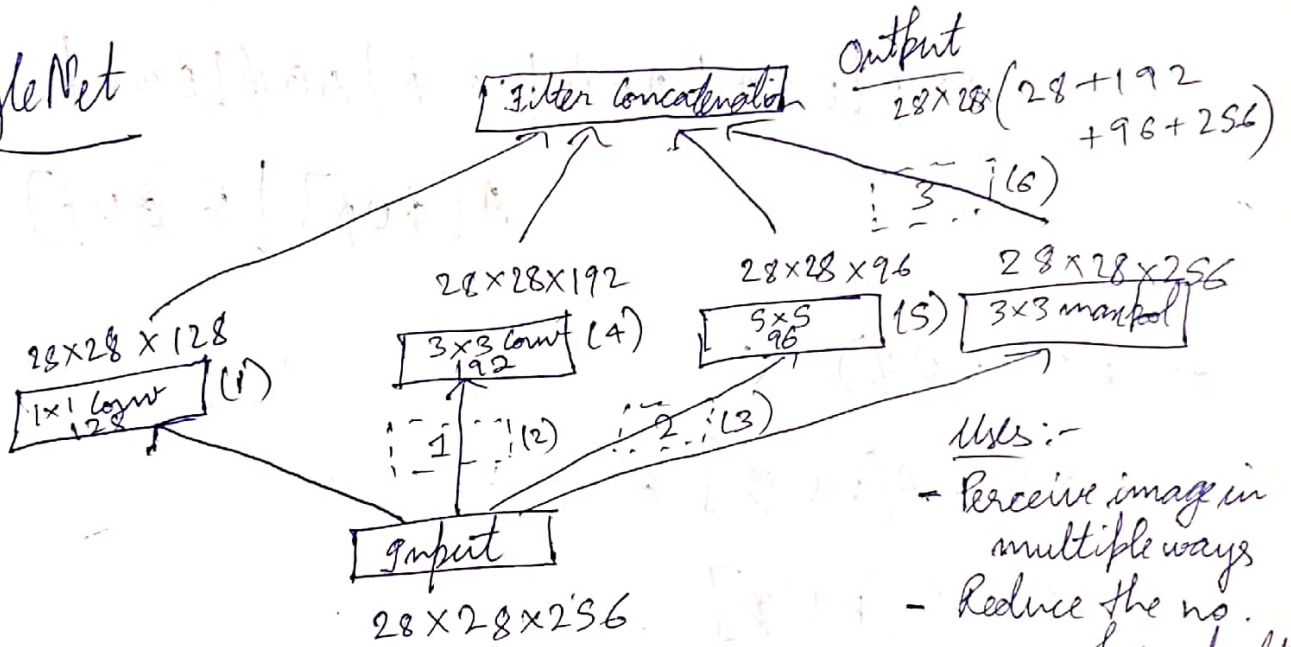
→ Vgg Net

- 138 million parameters
- 19 layers

→ Complexity of the CNN's

Measured by the # Floating Point Operations (FLOPs)

→ GoogleNet



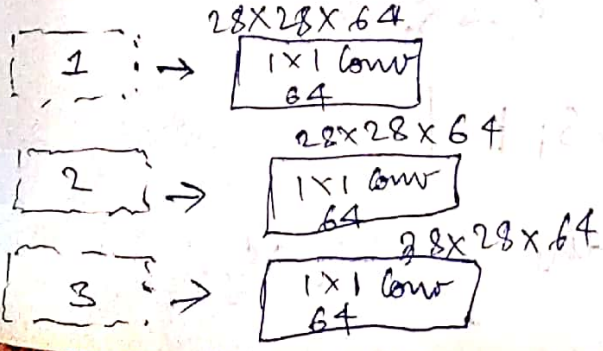
Uses:-

- Perceive image in multiple ways
- Reduce the no. of computations

Considers the MULTS

1) $28 \times 28 \times 256 \times 1 \times 1 \times 128$
2) $28 \times 28 \times 256 \times 3 \times 3 \times 192$
3) $28 \times 28 \times 256 \times 5 \times 5 \times 96$

Total FLOPs = 854 M



Computations

1) $28 \times 28 \times 256 \times 1 \times 1 \times 128$
2) $28 \times 28 \times 256 \times 1 \times 1 \times 64$
3) $28 \times 28 \times 256 \times 1 \times 1 \times 64$
4) $28 \times 28 \times 64 \times 3 \times 3 \times 192$
5) $28 \times 28 \times 64 \times 5 \times 5 \times 96$
6) $28 \times 28 \times 256 \times 1 \times 1 \times 64$

Mults = 358 M

28/03/23

Deep Learning

CNN:- Parameters shared across grids

RNN:- Parameters shared across long sequences.

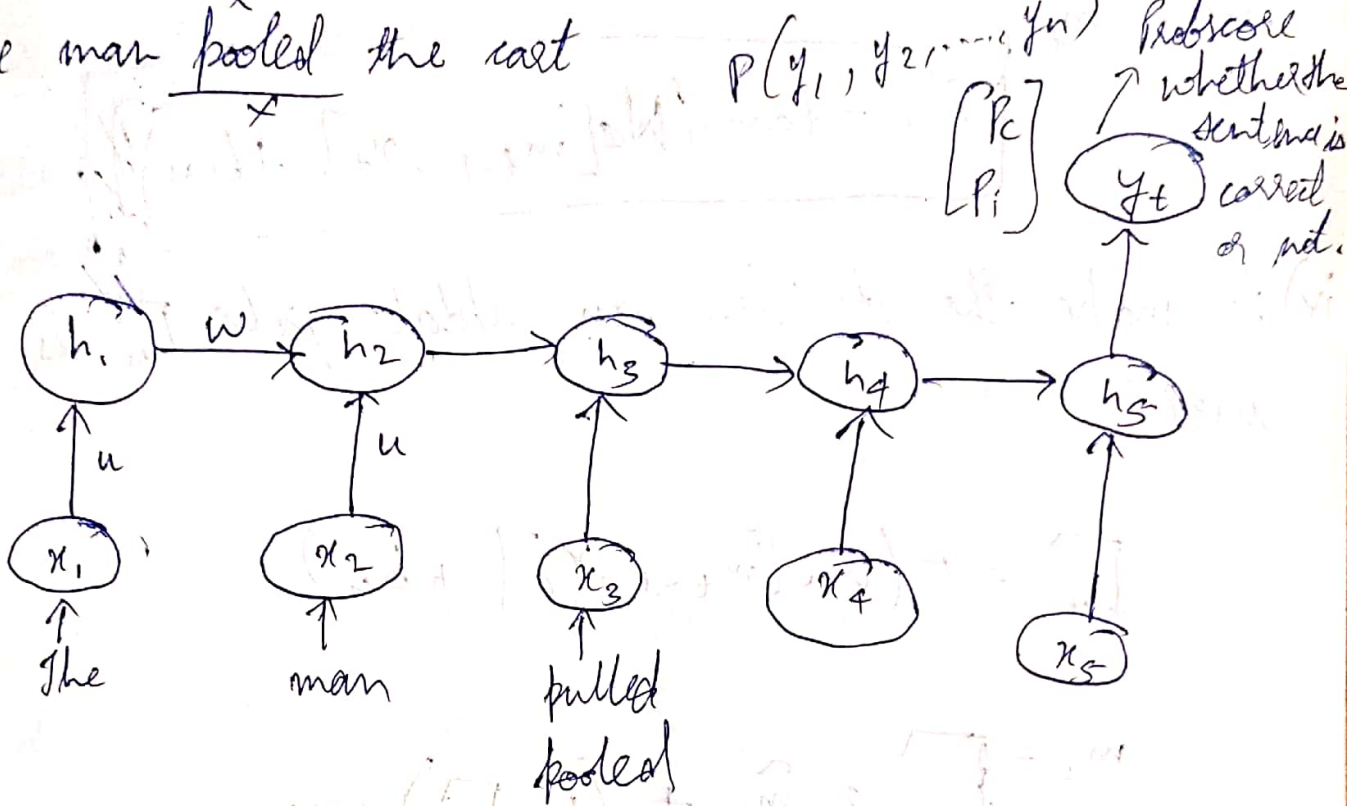
28/03/23

31/03/23

Deep learning

The man pulled the cart.

The man pooled the cart



$$h_t = \sigma_h \left(u x_t + w h_{t-1} + b_x \right)$$

$$\hat{y} = \sigma_y \left(v h_t + b_y \right)$$

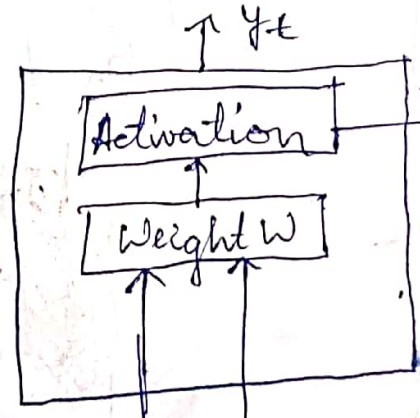
$$L = y \log \hat{y} + (1 - y) \log (1 - \hat{y})$$

GRU

i) Maintains a memory cell that contains relevant information like whether the noun encountered was singular / plural.

ii) Maintains a memory cell that contains relav. information $M_t = h_t$

Vanilla RNN



like whether the noun encountered was singular / plural

iii) At every step, the GRU needs to decide whether m_t needs to be replaced by an alternate / candidate value $\tilde{m}_t$

$$\tilde{m}_t = \tanh\left(W_c[m_{t-1}, x_t] + b_m\right)$$

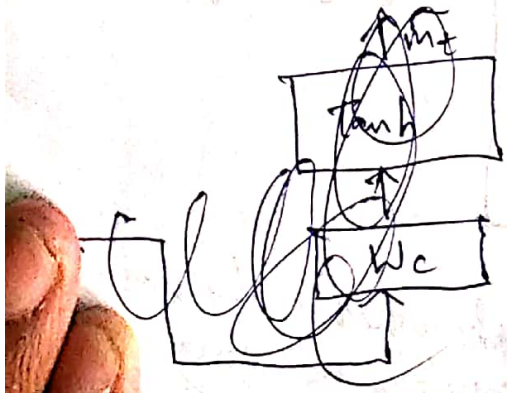
iv) To make the decision, an update gate Γ_u is used

$$\Gamma_u = \sigma\left(W_u[m_{t-1}, x_t] + b_u\right)$$

$$m_t = \Gamma_u \odot \tilde{m}_t + (1 - \Gamma_u) \odot m_t$$

$\odot \rightarrow$ hadamard product

$$\vec{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \quad \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad \vec{a} \odot \vec{b} = \begin{bmatrix} a_1 . b_1 \\ a_2 . b_2 \\ \vdots \\ a_n . b_n \end{bmatrix}$$



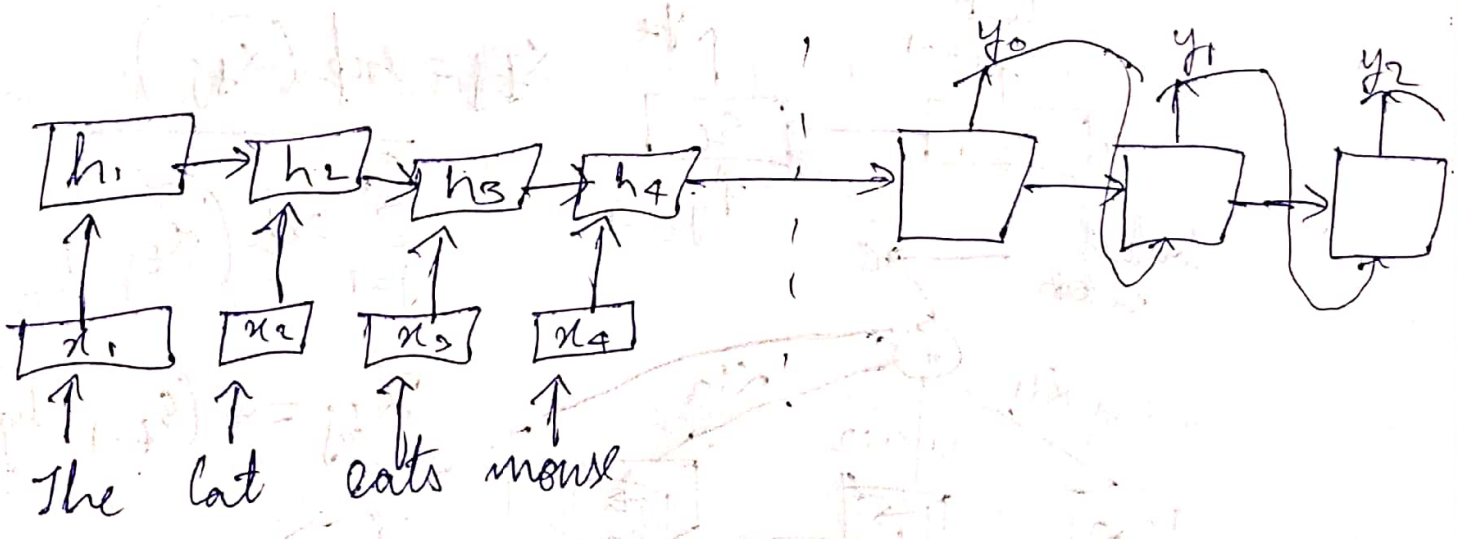
Attention-Mechanism DL 10/04/23

Neural machine translation by jointly learning to align and translate, Bahdanau et al ICLR 2015.

The cat eats mouse

Billi Chuha khati hain

Alkatta Aklathinanat

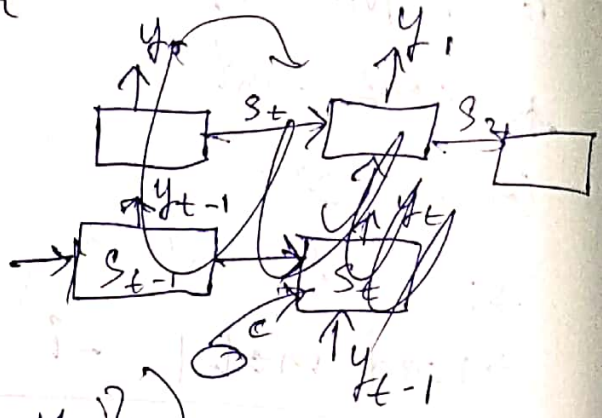


RNN based encoder

$X = (x_1, x_2, \ldots, x_{T_x})$ — Encoder

$h_t = f(h_{t-1}, x_t)$

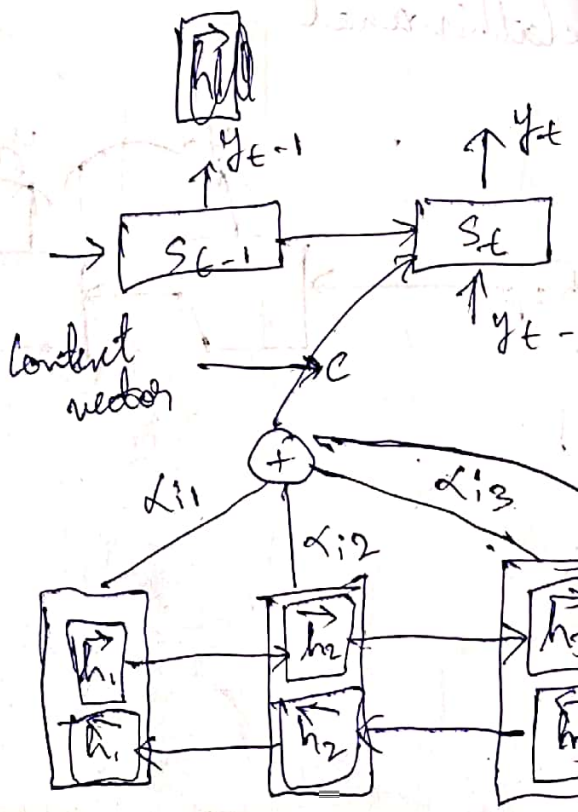
$c = g(h_1, h_2, \ldots, h_{T_x})$ — context vector



Decoder

$$p(y) = \prod_{t=1}^{T_y} p(y_t \mid \{y_1, y_2, \ldots, y_{t-1}\})$$

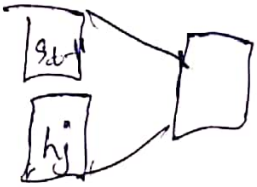
$$p(y_t \mid \{y_1, y_2, \ldots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$



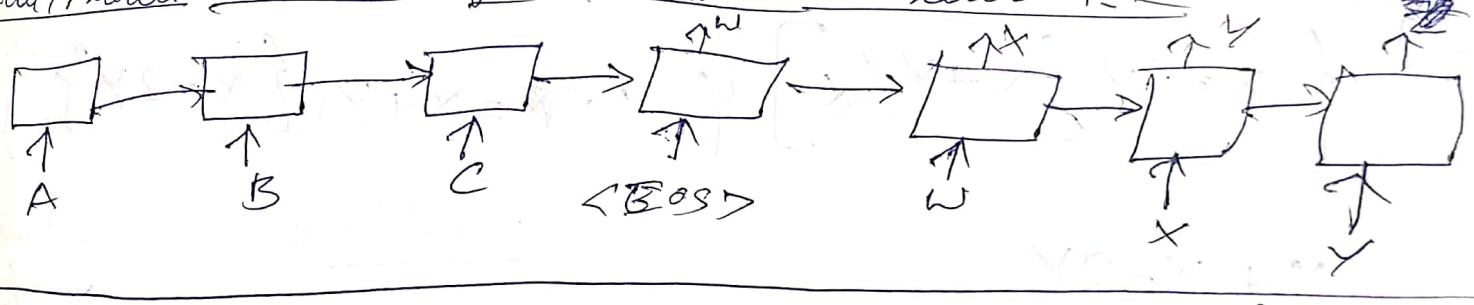
$$c^{(t)} = \sum_{j=1}^{T_x} \alpha_{tj} h_j$$

$$\alpha_{tj} = \frac{\exp(\tilde{\alpha}_{tj})}{\sum_{j=1}^{T_x} \exp(\tilde{\alpha}_{tj})}$$

$$\tilde{\alpha}_{tj} = f(s_{t-1}, h_j)$$

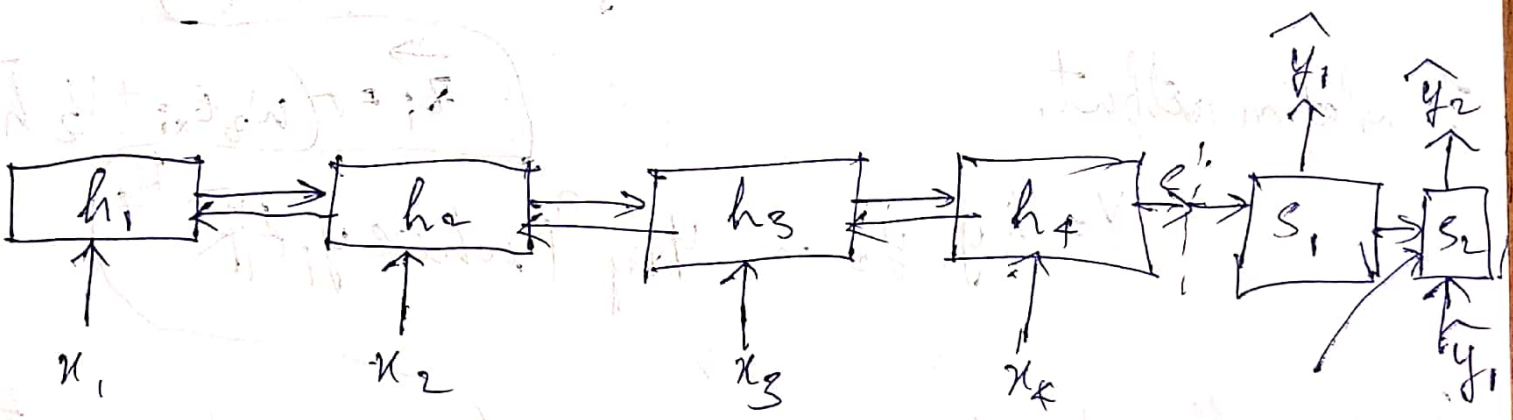


Encoder / Decoder Architecture for machine translation:—

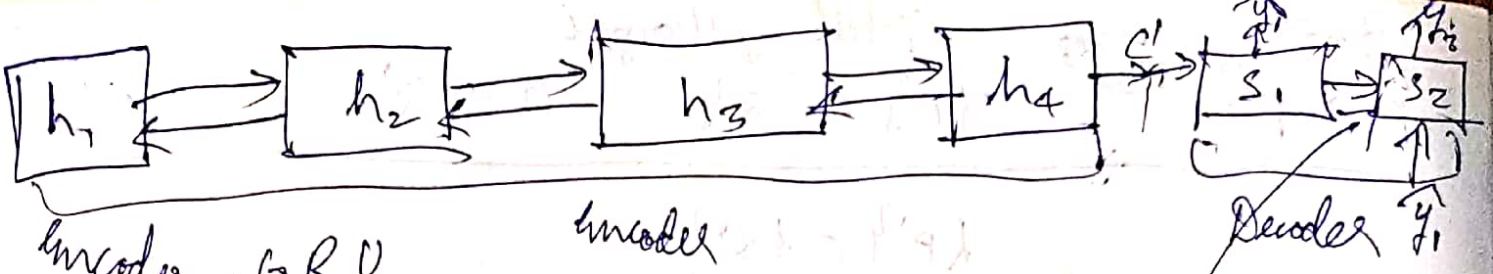


Deep Learning

13/04/23



Problem :- In case of planary sentences, 'c' became the bottle neck.



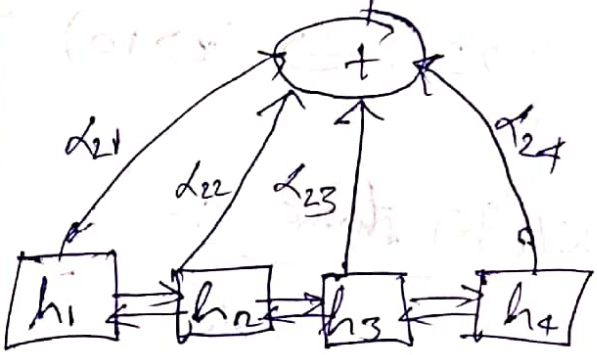
Encoder - GRU

Encoder

Decoder

Source sentence - Word token

1 of K vectors



Construct a vocabulary of K_x words

$$\vec{h}_i = (1 - \vec{z}_i) \circ h_{i-1} + \vec{z}_i h_i$$

Update gate

$\rightarrow$ 1 of K $\rightarrow$ one hot encoding over K_x words

$$\vec{h}_i = \tanh(wE_{x_i} + u[\vec{r}_i \circ \vec{h}_{i-1}])$$

Candidate cell

Relevance gate

Input is $x = \{x_1, x_2, \ldots, x_{T_x}\}$ where $x_i \in \mathbb{R}^{K_x}$

Translation output,

$$\vec{z}_i = \sigma(W_z E_{x_i} + U_z \vec{h}_{i-1})$$

$$Y = \{y_1, y_2, \ldots, y_{T_y}\}$$ where $y_i \in \mathbb{R}^{K_y}$

Encoder :- forward states

$$\vec{r}_2 = \sigma(W_r E_{x_i} + U_r \vec{h}_{i-1})$$

same equation for backward states.

Decoder:-

$$s_i = (1 - z_i) \circ s_{i-1} + z_i \circ s_{i-1}$$

$$\underline{s_i} = \tanh(W E y_i + U[r_i \circ s_{i-1}] + C c_i)$$

↑ candidate

Weight ↓ on $C c_i$

c_i ↑ context vector

$$c_i = \sum_{j=1}^{T_y} \alpha_{ij} h_j$$

$$s_0 = \tanh(W_s \overleftarrow{h_1})$$

$W_s \in \mathbb{R}^{n \times n}$

$$z_i = \sigma(W_z E y_{i-1} + U_z s_{i-1} + C_z c_i)$$

$$r_i = \sigma(W_r E y_{i-1} + U_r s_{i-1} + C_r c_i)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{i=1}^{T_x} \exp(e_{ij})}$$

$$e_{ij} = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$

Prob of output

$$p(y_i | s_i, y_{i-1}, c_i) \propto \exp(y_i^T W_o t_i)$$

$$t_i \sim U_o s_{i-1} + V_o E y_{i-1} + C_o c_i$$

Blue score :- Performance measure in machine translation.

Consider 2 sentences :-

- The ball is blue (Predicted)
- The ball has a blue colour (Target)
- The ball the Ball → 100% overlap
- The ball colour blue has } Should not have high blue score.

Step 1

Ngram

1-gram "The", "ball", "is", "blue"

2-gram "The ball", "ball is", "is blue".

3-grams "The ball is", "ball is blue".

Step 2

$$\text{Precision} = \frac{\#\text{ of correct predicted words}}{\text{Total predicted words}}$$

$$\text{Precision (Pred / Target)} = \frac{3}{4} \quad \text{(The, ball, blue)}$$

Prob But there can be repetition, (The, The, The, The)

Solution Use clipped precision instead of precision, or use multiple target sentences.

~~Blue score~~ @

→ To find the blue score, find the precision for
(P1) 1-gram, 2-gram (P2), 3-gram (P3), ..., n-gram.
(Higher the value of 'n', better the ~~blue score~~ result significance)

→ find geometric average precision score

$$= \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

$$= \exp\left(\sum_{n=1}^{N} \log p_n^{1/N}\right)$$

$$= \exp\left(\log\left(p_1^{1/N} \cdot p_2^{1/N} \cdots p_N^{1/N}\right)\right)$$

$$= p_1^{1/N} \cdot p_2^{1/N} \cdot p_3^{1/N} \cdots p_N^{1/N}$$

for (N=3) $GAP = p_1^{1/3} p_2^{1/3} p_3^{1/3}$

Problem (High probab. for smaller n-grams)

→ Find brevity penalty (penalize shorter sentences)

$$= \begin{cases} 1, & \text{length} > r \\ e^{\left(1 - \frac{r}{\text{length}}\right)}, & \text{length} < r \end{cases}$$

↓ predicted length

Bleu score = GA P. Brevity penalty